# Table of Contents

1.0 BUILDING SNMP AGENTS ................................................................. 8

2.0 SNMP AGENT ARCHITECTURE ....................................................... 9

3.0 STANDARDS SUPPORTED .............................................................. 11

4.0 DEFINING A MIB ............................................................................. 12
   4.1 MIB Overview .............................................................................. 12
   4.2 Creating a New MIB ................................................................. 12
   4.3 Adding Imports ......................................................................... 13
   4.4 Adding a Scalar ......................................................................... 14
   4.5 Adding a Table ........................................................................... 15
   4.6 Adding a Trap/Notification .................................................... 18
   4.7 Adding Constructs Specific to SMIv2 .................................... 19
   4.8 Creating User Defined Data Type using Textual Convention ... 22
   4.9 Converting the MIB Module ................................................. 22
   4.10 Dragging and Dropping a Node ........................................... 23
   4.11 Parsing MIBs .......................................................................... 23

5.0 CREATING A SIMPLE SNMP AGENT USING MIB COMPILER .......... 26
   5.1 Overview .................................................................................... 26
   5.2 Creating a New Project ............................................................ 26
   5.3 Loading the MIB in MIB Compiler ........................................... 26
   5.4 Generating Source Code for the MIB .................................... 27
   5.5 Compiling the Generated Code .............................................. 27
   5.6 Starting and Stopping the Agent .......................................... 28

6.0 CONFIGURING AGENT SETTINGS .............................................. 29
   6.1 Overview .................................................................................... 29
   6.2 Configuring General Settings .............................................. 29
   6.3 Configuring Source Generation Settings ............................... 31
   6.4 Packaging the Agent .............................................................. 33
   6.5 Configuring Root OID ............................................................. 34
   6.6 Initializing the Agent by Reading a Configuration File .......... 34
   6.7 Command Line Options ....................................................... 35
   6.8 Other Options .......................................................................... 35
7.0 TESTING THE AGENT ................................................................. 36
  7.1 Overview .................................................................................. 36
  7.2 Starting MIB Browser .............................................................. 36
  7.3 Using MIB Browser ................................................................ 36
  7.4 Testing the Agent With SNMP Operations ......................... 48
  7.5 Using Row Status ................................................................. 52

8.0 INSTRUMENTING THE GENERATED CODE ............................ 55
  8.1 Overview .............................................................................. 55
  8.2 Generated Code Structure .................................................. 55
    8.2.1 Main File ................................................................. 55
    8.2.2 Request Handler File for a Scalar/Tabular Group ........ 56
    8.2.3 Instrument File for a Scalar ....................................... 56
    8.2.4 Entry File for a Table ................................................ 57
    8.2.5 Persistence Files ....................................................... 57
    8.2.6 Notification File ....................................................... 57
  8.3 Instrumenting the Generated Code ....................................... 57

9.0 SNMP TABLE HANDLING IN GENERATED CODE ............ 60
  9.1 Overview .............................................................................. 60
  9.2 Storing the Tabular Data ...................................................... 60
    9.2.1 Agent Table Model ................................................... 60
    9.2.2 User Storage Model .................................................. 61
  9.3 Initializing Tables Before Agent Start-Up ............................ 63
    9.3.1 createAndAddEntry Method ...................................... 63
    9.3.2 setTableVector Method ............................................ 64
  9.4 Index Handling ..................................................................... 65
    9.4.1 Supported Syntax Types in Index ............................... 65
    9.4.2 Types of Indexing ..................................................... 65

10.0 AUTHENTICATING SNMP V1/V2C REQUESTS ..................... 67
  10.1 Overview ............................................................................ 67
  10.2 Authentication Support in the Agent .................................. 67
  10.3 Community Details for Authentication ............................ 67
  10.4 Configuring Managers to a Community ......................... 68
  10.5 If Authentication Fails ..................................................... 70
  10.6 Implementing Your Own Authentication ......................... 70
  10.7 Running the Example ..................................................... 70
11.0 ENABLING AUTHORIZATION IN SNMP V1/V2C USING VACL............... 71
11.1 Overview.................................................................................................................. 71
11.2 View-based Access Support in the Agent................................................................. 71
11.3 Details of MIB Views to Managers.......................................................................... 71
11.4 Adding MIB Views for Communities ....................................................................... 72
11.5 Implementing Your Own View Access..................................................................... 73

12.0 SENDING TRAPS AND INFORMS.................................................................. 74
12.1 Overview.................................................................................................................. 74
12.2 Generating Traps..................................................................................................... 74
12.3 Manager Details for Sending Traps.......................................................................... 75
12.4 Adding Manager Entries ......................................................................................... 76
12.5 Disable Startup trap............................................................................................... 79
12.6 Testing the Agent for Traps.................................................................................... 79
12.6 Knowing the Details of the Trap Generated (Reliable Traps)................................. 79
12.7 Validating and Filtering Outgoing Traps................................................................. 80
12.8 Implementing Your Own Table for Storing the Manager Information............... 81
12.9 Implementing Informs............................................................................................ 81
12.10 Sending Traps for Log4j Log Messages................................................................. 82
   12.10.1 Using the Agent's own SnmpTrapService......................................................... 82
   12.10.2 Using the XML file Configuration................................................................. 83
   12.10.3 Testing the Agent......................................................................................... 84
12.11 Running the Example......................................................................................... 84

13.0 IMPLEMENTING SNMP PROXY..................................................................... 85
13.1 Overview.................................................................................................................. 85
13.2 SNMP Master Agent.............................................................................................. 85
13.3 SNMP Subagent...................................................................................................... 85
13.4 Creating Master Agent........................................................................................... 86
13.5 Creating Subagent.................................................................................................. 86
13.6 Registering Subagents in the Master Agent........................................................... 87
   13.6.1 Before Agent Startup.................................................................................. 87
   13.6.2 During Run Time...................................................................................... 89
13.7 Different Types of Master - Subagent Relationship.................................................. 90
13.8 Knowing the Status of the Subagent (Heart Beat Mechanism).............................. 91
13.9 Sending Traps From Subagent Through Master Agent ............................................. 93
  13.9.1 Filtering Traps Received from Subagent Using Trap Filtering Table ......................... 93
  13.9.2 Enabling Trap Filtering Table .................................................................................. 93
  13.9.3 Adding Subagent Entries to Trap Filtering Table ...................................................... 94
13.10 Security in Proxy (Using SNMPv3) ........................................................................ 96
13.11 Making AdventNet Agents as Subagents to Third-Party Agents
        (including OS Native agents ) .................................................................................. 96
  13.11.1 Configuring AdventNet Agent as Subagent to Solaris Agent ................................... 96
  13.11.2 Configuring AdventNet Agent as Subagent to Windows Agent Using apadll ............... 97
  13.11.3 Configuring AdventNet Agent as Subagent to Linux Agent ....................................... 99
  13.11.4 MIB-II Support Using Native Agent as Subagent ................................................... 99
13.12 Running the Example ............................................................................................ 99

14.0 CONFIGURING SNMPV3 AGENTS ........................................................................... 101
14.1 Overview ................................................................................................................ 101
14.2 Security Levels in SNMPv3 ................................................................................... 101
14.3 Default Users of SNMPv3 Agents .......................................................................... 102
14.4 Developing a Sample V3 Agent ............................................................................. 103
14.5 Testing the SNMPv3 Agent With Default Users ..................................................... 103
14.6 Adding More Users for v3 Agents (USM) ............................................................... 105
14.7 Storing Details of V3 Users in Your Own Model ...................................................... 107
14.8 Enabling Authorization in V3 Using VACM ............................................................. 108
14.9 Using USM Without VACM .................................................................................... 111
14.10 Authenticating Requests From v1/v2c Managers (Coexistence Support) ................. 111
14.11 Making the Agent Strictly SNMPv3 ........................................................................ 114
14.12 Sending Notifications .......................................................................................... 115
14.13 Making the v3Agent Compliant to SNMPv3 Standards ........................................... 118

15.0 SNMPV3 ADMINISTRATION TOOL ...................................................................... 120
15.1 Overview ................................................................................................................. 120
15.2 Connecting to SNMPv3 Agent .............................................................................. 120
15.3 USM Operations ..................................................................................................... 121
15.4 VACM Group Operations ..................................................................................... 125
15.5 VACM Group Access Operations ........................................................................ 125
15.6 VACM View Operations ....................................................................................... 126
16.0 DEVELOPING AND PORTING SNMP AGENT IN J2ME CDC ENVIRONMENT ................................................................. 127
  16.1 Overview .......................................................................................................................................................... 127
  16.2 Compatible J2ME CDC Specification ............................................................................................................ 127
  16.3 Developing the SNMP Agent ......................................................................................................................... 127
  16.4 Porting the Agent in J2ME CDC Environment ................................................................................................. 128
  16.5 Testing the Agent ........................................................................................................................................... 129
  16.6 SNMP Agent Memory and CPU Requirement ................................................................................................. 129

17.0 IMPLEMENTING BUSINESS LOGIC USING RULE ENGINE ................................................................. 130
  17.1 Overview ........................................................................................................................................................ 130
  17.2 What Is Rule and Ruleset? ............................................................................................................................. 130
  17.3 Enabling Rule Engine ...................................................................................................................................... 130
  17.4 Configuration Files Used for Rule Engine Operation ..................................................................................... 131
    17.4.1 File Containing Agent Details .................................................................................................................. 131
    17.4.2 File Containing Ruleset Details ............................................................................................................... 132
  17.5 Functioning of Rule Engine ............................................................................................................................ 135
  17.6 Customizing Rule Engine .............................................................................................................................. 137
  17.7 Packaging Details .......................................................................................................................................... 137

18.0 MAKING THE SNMPV2C AGENT COMPLIANT TO SNMPV2C STANDARDS ...................................................... 139

19.0 SUPPORTED STORAGE TYPES FOR SCALARS AND TABLES ........................................................................... 142

20.0 LOGGING IN SNMP AGENT .......................................................................................................................... 145
  20.1 Overview ........................................................................................................................................................ 145
  20.2 Logging Levels ................................................................................................................................................ 145
  20.3 Setting Logging Levels .................................................................................................................................... 145
  20.4 Viewing Logging Reports ................................................................................................................................ 146

21.0 RUNNING MULTIPLE AGENTS IN A SINGLE JVM ............................................................................................ 147

22.0 FAILOVER SUPPORT FOR API TABLES ........................................................................................................... 150

23.0 PLUGGING IN OTHER TRANSPORT PROTOCOLS .......................................................................................... 153

24.0 WORKING WITH JBUILDER ............................................................................................................................ 155

25.0 EMBEDDED JVM SUPPORT ............................................................................................................................ 162

26.0 PACKAGING AND STARTING AGENT AS A SERVICE .................................................................................... 167
  26.1 Overview .......................................................................................................................................................... 167
  26.2 Jars required for Packaging an SNMP Agent ................................................................................................. 167
  26.3 Running the Agent in Any Environment ...................................................................................................... 167
26.4 Packaging the Agent as a Service ........................................................................... 167
   26.4.1 Installing the Service .......................................................................................... 168
   26.4.2 Starting the Agent as a Windows Service .......................................................... 169
   26.4.3 Uninstalling the Service ...................................................................................... 169
   26.4.4 Testing the Agent Started as a Service ............................................................... 169
   26.4.5 Event Logging ..................................................................................................... 169

27.0 BUILDING MULTI-PROTOCOL AGENT WITH SNMP SUPPORT -
SNMP ADAPTOR .................................................................................................. 170

28.0 SNMP FAQS .................................................................................................. 171
   General FAQs ............................................................................................................ 172
   MIB Editor FAQs ...................................................................................................... 177
   MIB Compiler FAQs ................................................................................................. 179
   MIB Browser FAQs ................................................................................................. 180
   Proxy FAQs .............................................................................................................. 182
   Trap and Notification FAQs .................................................................................... 186
   SNMPv3 FAQs .......................................................................................................... 189
   Authentication and Authorization FAQs ................................................................. 192
   Table Handling FAQs ............................................................................................. 194
   OS Agent FAQs ....................................................................................................... 195
   Rule Engine FAQ .................................................................................................... 196

29.0 TROUBLESHOOTING ................................................................................... 199
   SNMPv1/v2c Agents ................................................................................................. 200
   SNMPv3 Agents ....................................................................................................... 203
   Rule Engine ............................................................................................................. 206

30.0 REFERENCE IMPLEMENTATION - STANDALONE SNMP AGENT ........... 209
   Introduction to SNMP Agents .................................................................................. 210
   Objective of the Tutorial ......................................................................................... 211
   Getting Started with the Tutorial ........................................................................... 212
   Application Notes .................................................................................................. 213
   Running and Testing the Application .................................................................... 214
   Running the Agent ................................................................................................ 214
   Testing the Agent with Default Values ............................................................... 215
   Placing Orders for Pets ......................................................................................... 215
   Testing The Agent for Traps .................................................................................. 217
Steps Involved in Developing the RI ................................................................. 219
Created a MIB .............................................................................................. 219
Generated and Compiled Code for the MIB ............................................. 224
Implemented Scalars .................................................................................. 227
Implemented Tables .................................................................................... 229
Implemented SNMP Traps ......................................................................... 234

31.0 MIGRATION GUIDE FOR SNMP AGENTS ........................................ 236
Building SNMP Agents

Overview

The Simple Network Management Protocol (SNMP) is by far, the dominant protocol in Network Management. This Protocol (SNMP) was designed to meet network management needs with ease.

Steps to Build SNMP Agents

Following are the steps to build Standalone SNMP Agents using the AdventNet Agent Toolkit.

1. Create a new project using MIB Compiler.
2. Load a MIB as input to the agent.
3. Generate and compile the code for the standalone SNMP agent using MIB Compiler.
4. Start the agent using MIB Compiler.
5. Test the SNMP agent using MIB Browser.

Additionally, the guide provides complete reference to the features of SNMP Agent and its implementations. AdventNet Agent Toolkit helps to provide both Standalone SNMP Agent and Multi-Protocol agent that supports SNMP. To build Multi-Protocol agent using SNMP Adaptor, refer to Building Multi-Protocol Agent with SNMP Support - SNMP Adaptor section. The FAQ explains all queries regarding SNMP Agent and its features. The Troubleshooting manual assists the users to resolve their issues, when they encounter any error or problem while working with the tool. The Tutorial section demonstrates complete implementation of the SNMP agent with other applications. Migration Guide helps to know more about migrating from 4.2 to 5.0/5.1.0 and 5.1.0 to 6.0.0.
2.0 SNMP Agent Architecture

The AdventNet Standalone SNMP Agent architecture supports all versions of SNMP and is designed in a manner to suit the requirements of Enterprise Vendors, Service Providers and the OEM markets. The following diagram gives an overview of the different components in SNMP agent and the interactions between them.

![AdventNet SNMP Agent Runtime Architecture](image)

2.1 Transport Provider

SNMP messages are sent over various transport protocols. The protocols currently supported by AdventNet SNMP Agents are UDP/IP and TCP/IP. The communication between a manager and agent takes place through these transport providers.

2.2 Message Processing Unit

The Message Processing Unit is responsible for sending and extracting data from received messages. The Message Processing Unit potentially contains multiple Message Processing Models. Each Message Processing Model defines the format of a particular version of an SNMP message and co-ordinates the preparation and extraction of each such version-specific message format.

In this level, code is generated for a MIB given as input. The Request Handlers generated takes care of processing the request from the manager.

Thus, Message Processing Models receives the messages and processes them based on the version of the message.
2.3 Instrumentation

The message processed can be instrumented to return specific values of the application. The generated code of the MIB Compiler has to be modified to get/set a MIB object value. Usually the MIB instrumentation differs based on the agent implementation. Database, runtime memory, xml, and text file storage support is available to store MIB objects value.
### 3.0 Standards Supported

<table>
<thead>
<tr>
<th>SNMP Version</th>
<th>RFC Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNMPv1</td>
<td>RFC 1155</td>
<td>Structure and Identification of Management Information for TCP/IP-based Internet</td>
</tr>
<tr>
<td>-do-</td>
<td>RFC 1157</td>
<td>Simple Network Management Protocol (SNMP)</td>
</tr>
<tr>
<td>SNMPv2</td>
<td>RFC 1901</td>
<td>Introduction to Community-based SNMPv2</td>
</tr>
<tr>
<td>SNMPv3</td>
<td>RFC 3411</td>
<td>SNMP Framework MIB</td>
</tr>
<tr>
<td>-do-</td>
<td>RFC 3412</td>
<td>SNMP Message Processing and Dispatching (MPD).</td>
</tr>
<tr>
<td>-do-</td>
<td>RFC 3413</td>
<td>SNMP Target MIB and SNMP Notification MIB</td>
</tr>
<tr>
<td>-do-</td>
<td>RFC 3414</td>
<td>SNMP User Based Security Model (USM) MIB</td>
</tr>
<tr>
<td>-do-</td>
<td>RFC 3415</td>
<td>SNMP View Based Access Control Model (VACM) MIB</td>
</tr>
<tr>
<td>-do-</td>
<td>RFC 3584</td>
<td>SNMP Coexistence between Version 1, Version 2, and Version 3 MIB</td>
</tr>
</tbody>
</table>
4.0 Defining a MIB

4.1 MIB Overview

MIB is a file that contains definitions of management information so that networked systems can be remotely monitored, configured, and controlled. The rules for writing a MIB is defined in a collection of documents called the Structure of Management Information (SMI).

Defining a MIB is the first step towards development of an SNMP Agent. The steps involved in developing a MIB file are:

- **Data identification**: Identify the data or the objects of your device that needs to be managed using SNMP agent and lay them out in the form of scalar or tabular objects.
- **Data definition (MIB Building)**: Construct ASN.1 MIB definitions for the objects using the MIB Editor tool in AdventNet Agent Toolkit. The MIB Editor has an easy-to-use wizard interface through which various constructs conforming to SMIv1 and SMIv2 specifications and ASN.1 rules can be created. The wizard interface helps in easy MIB designing by hiding the unimportant details of the MIB syntax rules, clauses etc.

4.2 Creating a New MIB

The topics covers creating an SMIv1 and SMIv2 MIB module using MIB Editor tool. You can add Scalars, Tables, Imports, and various constructs conforming to SMI standards to make it a complete MIB. Please go through all the topics for complete knowledge on defining a MIB.

4.2.1 Creating an SMIv1 MIB Module

To create the MIB,

- Start the MIB Editor
- Select **File -> Create MIB** from the menu bar or by clicking on the toolbar icon.
- Fill in the following details in the wizard that opens up:
  - **Module Name** of the MIB (Module name should start with an upper case letter).
  - **Module Directory** in which the MIB should be placed.
  - **Version** of the MIB (Choose SMI V1) and
  - **Root OID** of the MIB
- And click **OK**.
- You can see the Parent Node (MIB file name) added under the MIB tree frame in the left side.
4.2.2 Creating an SMIv2 MIB Module

To create the MIB,

- Start the MIB Editor
- Select File -> Create MIB from the menu bar or by clicking on the toolbar icon.
- Fill in the following details in the wizard that opens up:
  - Module Name of the MIB Module (Module Name should start with an upper case letter).
  - Module Directory in which the MibModule should be placed.
  - Version of the MIB (Choose SMI V2) and
  - The Root OID of the MIB
- And click OK.
- You can see the Parent Node (MIB file name) added under the MIB tree frame in the left side.

Note:
- An already existing MIB can be loaded into the MIB Editor using File -> Load MIB Module menu bar or using tool bar icon.
- A MIB created can be modified using the Operations -> Modify Node option in the menu bar or using tool bar icon.

4.3 Adding Imports

Imports specify the items which need to be included from another MIB Module into the Current MIB Module. They are mainly used when an index of another MIB or a Textual Convention of another MIB need to be used. To add imports,

1. Select Add Imports from the Operations menu in the menu bar or select the icon Add imports to the MIB present in the toolbar.
2. From the frame that pops up, use the Import MIB button to add MIB(s) from the mibs directory.
3. Select the MIB which you like to import and click "Load". This loads the respective MIB to the frame.
4. Click "OK" button which loads it in the MIB tree.
5. The Indexes and the TCs of this MIB can also be used.
   - To use the Indexes : While defining a table for the new MIB, index or augments are chosen using Add External Indices/ Augments option. If index is chosen, it helps you to utilize any selected index of the imported MIB. If Augments option is chosen, all the indices under the selected entry will be added from the imported MIB
   - To use the TCs : While defining an object for the new MIB, the Syntax option combo box includes all the TCs of the imported MIB from which you can choose for the new MIB.

Note:
- The same procedure has to be followed for both V1 and V2 Module.
- The MIB file to be imported and the MIB file to be created must be in the same directory.
- By default, if no imports are specified, the MIB Editor will import mib-2, and enterprises objects from the RFC1213-MIB.
- It is not necessary that only MIBs created by MIB Editor can be imported. MIB Editor conforms to SMIV1 and V2 specifications and ASN.1 rules for MIB definitions. Therefore, any MIB following this standard can be imported.
4.4 Adding a Scalar

Scalar is a leaf node that can hold only a single value. To store single values in the MIB, scalar variables can be used. To add a scalar variable to the MIB,

1. Select **Scalar Object** from **Operations -> Add Node** in the menu bar. The corresponding template appears on the right frame.

2. Fill in the template with the following details:
   - Name for the Object Type.
   - Sub ID of the node (need not be modified).
   - Parent ID for the Scalar Node (non-editable).
   - Syntax to be used for the Scalar Node. The "..." button opens up with a wizard to specify the Simple or Value/Range or Enumeration for the syntax chosen. These syntax are explained below in Syntax Definition section below.
   - Maximum Access for the Node.
   - Status of the Node.
   - Description of the Scalar Node (If Required).
   - Reference for the Scalar Node (If Required).
   - Format in which the Default Values have to be generated : either Decimal or Binary or Hexa. Just choosing the required format in the combo box will generate default values in that format.

3. Finally **Add to Module** and this will add a scalar variable to the MIB module.

**Syntax Definition**

- **Simple** option defines the default integer range. Takes the format specified in DefVal.
- **Enumeration** option defines a variable for a particular value. Sample Definition: Value - 1, Label - true. This enumeration can be added to the Syntax type.
- **Value/Range** option enhances you to specify your own range of Integer. The MIB will respond only with the values that lie within the Range.

**Maximum Access Definition**

- **Read_Only** - Access for retrieving values of the node.
- **Read_Write** - Access for retrieving and modifying the values.
- **Write_Only** (only for v1) - Access for modifying the existing values of the node.
- **Read_Create** (only for v2) - Access for retrieving, modifying, and creating the values.
- **Not_Accessible** - Access for not being able to perform any operation on this node.
- **Accessible for Notify** (only for v2) - Access for notifying a significant change in the node.

**Status Definition**

- **Mandatory** (only for v1) - indicates that the definition is valid and should be implemented.
- **Current** (only for v2) - indicates that the definition is valid currently.
- **Deprecated** - indicates that the definition will soon be made obsolete and need not be implemented.
- **Obsolete** - indicates that the definition is not valid and should not be implemented.
- **Optional** (only for v1) - indicates that the definition is valid and may or may not be implemented.
4.5 Adding a Table

When a group of information is to be given in the MIB, a table can be used. A table can maintain a row with various columns. Each table has an index column and every row of the table is identified by the index column value. To define a table and add it to the MIB, you will have to define a row entry for the table object first and define one or more columnar objects with at least one index column.

4.5.1 Defining the Table

To add a Table to the MIB, the first step is to define the Table construct. Follow the steps given below for defining the Table.

- Select **Table Object** from the **Operations -> Add Node** menu bar. The corresponding template appears on the right side frame.
- Fill in the template with the following details:
  - Name for the Object Type.
  - SubID of the Node (Need not be modified).
  - Parent OID of the Node.
  - Syntax Sequence gets loaded automatically as `xxxEntry`, `xxx` being the prefix of the table created. This is a special value of the SYNTAX, which indicates that a table object is being defined.
  - Maximum Access for the node. By default, it will be not-accessible and cannot be configured.
  - Status of the table node.
  - Description of the Node, if required.
  - Reference of the Node, if required.
- Finally, **Add Table**.

4.5.2 Adding Entry to the Table

Once the table is added, the template for adding an entry opens up. To add an entry, Specify the

- Object Type.
- Sub ID value (Need not be modified).
- Table name
- Syntax which is the entry name
- Max-Access of the row.
- Status of the row.
- Description and Reference (If Required) and
- External index/indices can also be added using **Add External Indices/Augments** option. External Index is an index column from another table added to the table being created. Click the **Choose Index** button to add external index to the table. A **Select Node** dialog pops up with a list of indices of various tables (if imported). Select the index columns and click the Add button. The selected index columns are added as the External index to the table and is visible only in the MIB tree view. The **Augments** option is disabled, if it is a V1 MIB. Here, all the indices under the selected entry will be added as external indices. For more details on External Index, please refer Adding External Index section given below.
- Add the entry using **Add Entry** option which adds the row to the table.
4.5.3 Adding a Column

Once the entry is added, the columns have to be defined and the template for the same is made available. To add a column to the table,

1. Specify the following:
   - Name for the Object Type.
   - Sub ID of the tabular node (Need not be modified).
   - Entry of the table.
   - Index of the table - it should be an Implied Index or an ordinary Index column to the table. For more details on Implied Indexes, refer section Adding Implied Index given below.
   - Syntax to be used for the table node. The "..." button opens up with a wizard to specify the Simple or Value/Range or Enumeration for the syntax chosen.
   - Maximum Access for the Node.
   - Status of the Node.
   - Description of the Table Node (If Required).
   - Reference for the Table Node (If Required).
   - Format in which the default values have to be generated: either Decimal or Binary or Hexa.

2. And Add Column to Table. Finally, "Add to Module" is chosen which will add the table node under the specified parent ID and display it in the MIB Tree.

<table>
<thead>
<tr>
<th>Note:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Table Name, by convention should end with &quot;Table&quot; and it should start with lower case letters.</td>
</tr>
<tr>
<td>• To add a row to a table having Row Status column please refer to Testing the Agent section.</td>
</tr>
<tr>
<td>• At least one column of the table should be defined as the Index column.</td>
</tr>
<tr>
<td>• The Max Access of SMIv1 Table's Index Column are Read-Only and the Maximum Access of SMIv2 Table's Index columns are Not-Accessible.</td>
</tr>
<tr>
<td>• There are provisions for the following conversions from the existing table: Column -&gt; Index Column -&gt; Implied Index Index -&gt; Column Index -&gt; Implied Index Implied Index -&gt; Column Implied Index -&gt; Index</td>
</tr>
</tbody>
</table>

4.5.4 Adding Multiple Indexes

This requirement of adding multiple index will arise, when a particular row needs to be identified by two index columns. A table can have any number of index columns and to add multiple indexes to a table, follow the steps given in Adding a Table topic. By checking the Index option for two or more columns in the table, multiple index support is achieved. Apart from multiple indexing, external indexing is also supported that is explained below.
4.5.5 Adding External Index

External Indices are indices which are inherited from another table of the same MIB or from a different MIB. The table from which the index has been taken is called as the “Base Table” and the table which has been formed using the index of the Base table is called as the “Derived table”.

- Hereby, it is clear that the Base Table and the Derived Table are interrelated
- Any row can be added to the Derived Table only after specifying the instances of the Base Table.
- Adding a row in the Base Table need not necessarily add a row in the Derived Table.
- Please note that, a Derived Table can also have additional index columns (local indexes) apart from the Base Table's index column, as every normal table would have.
- And, while generating code for an External Index table, the corresponding Base Table has to be included before generation.

For having an External Index in the Table,
1. Check the **Index** option that comes while adding entry.
2. **Choose Index** is used to choose the External index from the indices listed.
3. Only when the MIB is imported, will it be shown in External Index UI.
4. The **Augments** option is disabled incase of SMI v1 MiBs. It is similar to external index.
5. Please refer to the topic Adding Entry to the Table given above for the option.

4.5.6 Adding Implied Index

While querying a table, the OID of the column to be queried along with its instance values is given. The length of the Instance value is also specified in cases where the Syntax is of STRING types like OCTET-STRING, OID, Display String. In case, Implied Index support is availed, the length need not be specified. For a Table to have an Implied Index,
1. Check the **Implied Index** option that comes across **Index** while Adding a Column to the Table. Please refer to the topic **Adding a Column** given above for the option.

**Note:**
- Implied Index cannot be assigned for Index columns with Syntax such as Integer, Counter, etc. (which has no varying length).
- If the table contains more than one index column, the last object specified in an index clause can be specified as the Implied Index, i.e., Implied Index column will be available only as the last Index.
- A table cannot have more than one Implied Index defined in it.

4.5.7 Row Status Support

Row Status is a Textual Convention (TC) defined by the SNMPv2-TC MIB. This support enables to add and delete rows in SNMPv2 tables from the management stations, at ease. This is one among the syntax provided by v2c agents. Hence, while defining a column this syntax option can be chosen. The status of whether the row is active or not-in-service etc., can be known here. The status can be any one among the following:

- active (1)
- notInService (2)
- notReady (3)
- createAndGo (4)
4.6 Adding a Trap/Notification

To notify the significant change in the state of a scalar or tabular variable, a trap or notification has to be defined for that particular variable. To define a Trap Type construct (for SMI V1) or Notification Type Construct (for SMI V2), please follow the steps given below.

4.6.1 Adding a Trap

1. Select an OID.
2. Add Trap Type, using Trap Type Construct from the Operations -> Add Node menu bar.
3. A Template opens up wherein the following details need to be filled in for Trap Type Construct:
   - **Trap Type**: The first 6 Trap types mentioned here are of **Generic Type** and are defined by the snmp group of MIB-II.
     - **coldStart**: coldStart traps indicate that a device has just powered up or has reinitialized completely (hard-reset). The configuration of the device may have changed and any data collected prior to the reset may have been lost.
     - **warmStart**: The warmStart is an indication that device has reinitialized but neither the device configuration nor the stored data gets changed. The device is expected to continue in the same state as seen before the reinitialization.
     - **linkDown**: The linkDown trap is an indication that a communication link or a port on a network node has failed.
     - **linkUp**: A linkUp trap is an indication that a communications link or a port on a node has come up after the failure and is ready for service.
     - **authenticationFailure**: The authenticationFailure trap indicates that an agent has received an SNMP message that cannot be authenticated. This is typically an indication that the SNMP PDU contained an unrecognized community name, or that the message was sent from a node residing at a network address not authorized to send SNMP requests to the agent.
     - **egbNeighborLoss**: The agent is reporting that the peer relationship between an External Gateway Protocol (EGP) neighbor and an EGP peer no longer exists.
     - **enterprisespecific**: This trap is used to define all events other than the six generic traps specified above. An “enterprise specific trap” is indicated by a value other than “snmp” for the ENTERPRISE field. The sending agent has detected an enterprise-specific event. The value of the specific trap type field indicates the nature of the event.
   - **Enterprise**: A Trap message is identified by the value in this Enterprise field. If the value specified is SNMP(.1.3.6.1.2.1.11), then the value is for generic-trap. Generic-traps are defined traps and so you need not define them. Otherwise, they are specific-traps. You can define Specific from the listed entries.
   - **Variables**: Used to specify one or more scalar or columnar objects whose value describes the event. The variables are listed by pressing the Var button.
4.6.2 Adding a Notification

1. Select an OID/ ModuleIdentity/ ObjectIdentity.
2. Add Notification Type, using Notification Type Construct from the Operations -> Add Node menu bar.
3. A Template opens up wherein the following details need to be filled in for Notification Type Construct:
   - The Template for Notification Type includes:
     - Name of the Notification Type.
     - Sub ID of the Notification (Need not be modified).
     - Parent OID of the Notification Type.
     - Status of the Notification - Current / Deprecated / Obsolete.
     - Scalar and Tabular Objects for which notification has to be generated. These objects need to be browsed and chosen.
     - Description and Reference for the Notification Type can be given, if required.
4. Finally click Add to Module to add the Notification.

Note: The Notification-Group construct is used in SMI v2 MIB modules to define a collection of related event definitions. The grouping is done to show the logical grouping of events as a hint for the Agent and Management Application designers/developers. At least a single Notification Type should be present in a Notification Group.

4.7 Adding Constructs Specific to SMIv2

4.7.1 Object Group

The OBJECT-GROUP construct is used to define a collection of related object type definitions. These object type may be a member of more than one object group. At least a single Object Type should be present in the Object group. This grouping is done by the MIB Module authors to show the logical grouping of object types and to help agent and management application developers. Only the object types defined in the same MIB Module may be members of the OBJECT GROUP.

To add this construct,
1. Select a Node.
2. Add Object Group from the Operations -> Add Node menu bar.
3. The corresponding template opens for the details to be filled in for the following:
   - Name of the Object Group.
   - SubID of the Group (Need not be modified).
   - Parent ID of the Group.
   - Objects to be present in the Group.
   - Status of the Group.
   - Description and Reference for the Node, if required.
4. Finally, Add to Module. The construct gets added in the MIB Tree.
4.7.2 Object Identity

The OBJECT-IDENTITY construct is used to assign an OID value to an identifier in the MIB module. The OID value assigned should be unique. To add an Object Identity to the V2 MIB,

1. Select Object Identity from the Operations -> Add Node menu bar.
2. The corresponding template opens up with the following details to be filled in,
   - Name of the Object Identity.
   - Sub ID for the Node (Need not be modified).
   - Parent ID of the node.
   - Status of the Node. It can be deprecated, current, or obsolete.
   - Description and Reference can be given for the node, if required.
3. Finally, Add to Module.
4. This will include the Object Identity in the tree.

4.7.3 Module Compliance

The MODULE-COMPLIANCE construct is used in SMI v2 MIB Modules to define implementation requirement specifications for agents. A requirement specification names groups of object types / events to be implemented. One or more requirement specifications may be defined within a MIB module. It may refer to the items defined in the containing MIB module and/or in other MIB Modules.

Currently, the primary use of the MODULE-COMPLIANCE construct is to allow the MIB module author to specify the minimal list of object types and events defined in the containing MIB module that an agent developer must implement. The construct consists of a header, followed by a list of MIB module requirement specifications. Within a MIB module requirements specification, we have a list of groups which are unconditionally required, followed by a list of conditionally required groups and exception specifications.

To define the construct,

1. Select Module Compliance icon from the Operations -> Add Node menu bar.
2. The respective template opens up with the following details to be filled in.
   - Name of the node to be created to define this construct.
   - Sub ID of the construct (Need not be modified).
   - Parent ID of the construct.
   - Status of the definition. It can be Current, Deprecated, or Obsolete.
   - Textual description of the node or item being defined.
   - Reference to specify the source of the definition.
   - Clicking the View Module Compliance Definition button and clicking New Button in it, will open up a wizard in which the values for Module and Mandatory groups are entered.
   - The MODULE sub-clause specifies the module identifier and by convention the name should start with an uppercase letter. Specify a Mandatory Group also.
   - When you "Add" these details you can find the Module Compliance with two groups namely Groups and Objects. Selecting Group and clicking the New button enables the group details.
   - GROUP: The GROUP clause is used to specify a conditionally required object or notification group. The DESCRIPTION clause, which is paired with the GROUP clause specifies the conditions when the group is required.
group may not be specified in the MANDATORY-GROUPS clause and in a GROUP clause. Also, a group may not be specified in two different GROUP clauses. The description clause describes the group description.

- Here again "Add", which will add the details under the Group.
- On selecting Object and clicking New button, the details required for the Object get enabled.
- OBJECT: The OBJECT clause and its associated clauses specify reductions in required behaviors of an object to meet a requirements specifications. This is typically used to specify that an implementation of a table need not support row creation to meet a requirements specification, even though the definition of the table defines row creation. This is accomplished by specifying that the minimum required value for access is either read-write or read-only instead of read-create in the MIN-ACCESS clause.
- MIN-ACCESS: The MIN-ACCESS clause can also be used to specify that an object in an object group need not be implemented. This is done by specifying the value of not-accessible for the MIN-ACCESS clause. The value of accessible-for-notify is used to indicate that only an object needs to be present to return an instance in an event.
- SYNTAX & WRITE-SYNTAX: The purpose of SYNTAX and WRITE-SYNTAX clauses is to specify a reduction in the implemented behavior of an object.
- DESCRIPTION: The DESCRIPTION clause associated with SYNTAX clause specifies the reduction in required behaviors of SYNTAX clause. The DESCRIPTION clause associated with GROUP clause, specifies the conditions when the group is required.

3. Finally, press Finish button to return to the initial screen and
4. Click Add to Module button.
5. The Module Compliance construct is successfully added in the Mib tree.

4.7.4 Agent Capabilities

The AGENT-CAPABILITIES construct is used in SMI v2 MIB Modules to specify the implementation characteristics of an SNMP Agent sub-system with respect to object types and events. The term "implement" means that the agent provides all defined behaviors of these items, unless documented by exception clauses in the AGENT CAPABILITIES construct.

To define the construct, 
1. Select Agent Capabilities from the Operations -> Add Node menu bar.
2. Enter the following details in the template that appears on the right frame :
   - **Name** of the node to be created to define this construct.
   - Sub ID of the construct (Need not be modified).
   - Parent ID of the construct.
   - PRODUCT RELEASE to describe the product release which includes the implemented capabilities.
   - **Status** of the definition. It can be Current, Deprecated, or Obsolete.
   - Description and Reference for the construct, if required.
   - On clicking View Agent Capabilities icon you get a template to be filled in with details :
     - **SUPPORTS**: Supports subclass specifies the module identifier. By default, it should start with upper case letters.
**INCLUDES**: The INCLUDES clause specifies object and event groups that an agent implements. Unless specified in a VARIATION clause, an agent fully implements all the behaviors specified in the definitions of the objects and/or events that are members of the groups in the INCLUDES clause.

- Press "Add" button and the screen reflects the additions.
- Select the newly created item in the left frame and press **New** button.
- The text boxes Variation, Syntax, Write Syntax, Access, Creation Requires, Deferral, and Description gets enabled.

- **VARIATION**: The VARIATION clause may be used to specify a change of behavior of an event or object, or to specify that an event/object is not implemented. The ACCESS clause with a value of not implemented is used for the latter case. This is the only clause other than DESCRIPTION that may be specified for an event variation.

- **SYNTAX & WRITE SYNTAX**: The purpose of SYNTAX and WRITE SYNTAX clauses is to specify a reduction in the implemented behavior of an object.

- **CREATION REQUIRES**: The CREATION REQUIRES clause is used to document the objects that are required in a single SET operation to create an instance of a row in a table.

- **DEFVAL**: The DEFVAL clause specifies a default value that an agent uses when an instance of a columnar object is created, instead of that specified in the definition of the object.

2. Finally, press **Finish** button to return to the initial screen and
3. Click **Add to Module** button.
4. The Agent Capabilities Compliance construct is successfully added in the Mib tree.

### 4.8 Creating User Defined Data Type using Textual Convention

TEXTUAL CONVENTION constructs are used to create new Data Types. This is done by allowing restrictions on an existing base type or previously created TEXTUAL CONVENTIONS. Textual conventions are nothing but redefined data type. To define the construct,

1. Select **Textual Convention** from the **Operations -> Add Node** menu bar and enter the following details in the template:
   - Name of the TEXTUAL-CONVENTION. Please note that the name has to start with Upper case letter.
   - SYNTAX Type for the TC. Add any one of the SNMP Base Data Type.
   - Also fill in details for the Display Hint, Status, Description and Reference fields.

2. And **Add to Module**.
3. The Textual Convention is added and displayed in the MIB tree of the left frame. Note that the defined TC cannot be modified or deleted.

### 4.9 Converting the MIB Module

To convert a MIB module,

1. Select **File -> Convert MIB** option from the menu bar which converts the MIB module from SMI v1 to SMI v2 or vice versa.
2. Also **Save as** option from the **File** in menu bar can be used for saving MIB Module. It can also be used for saving the MIB in the required location in the preferred name.
4.10 Dragging and Dropping a Node

The MIB Editor application facilitates dragging and dropping of nodes. You can drag a node from one group, drop it into another group and save the MIB to store the changes. Dragging and Dropping of a scalar group or a tabular group or a combination of both onto an Object Identifier (in case of v1 and v2) and Module Identity (in case of v2) is possible. You can also move the child node of a parent to another parent node.

Note: It is not possible to drag and drop a Trap/Notification and a Textual Convention. You cannot move a Module Identity either under its child node or its parent node.

4.11 Parsing MIBs

While loading MIB files in MIB Editor, it performs the following operations.

- Parsing and validating the syntax of the MIB module
- Constructing the MIB module into the tree structure

While performing the parsing and validation of the MIB files, if the MIB modules fail to conform to the SMI standards the loading will not be done. However, the Editor application requirements might mandate the loading of the non-standard files. On the other hand, some applications might require a stricter check on the compliance to the standards.

The parsing and validating syntax of the MIB file can be configurable. Agent Toolkit provides the following set of parsing levels which facilitates to select the level of parsing required by the Mib Editor application.

- Lenient
- Normal
- Serious
- Critical

In the Mib Editor application, parsing level has to be set first before loading a MIB. This level, once set, is used for subsequent MIBs that are loaded. If the level needs to be modified for the next set of MIBs loaded, it has to be set again.

Constructing the MIB Module into the Tree Structure

If parsing is completed successfully, the API resolves the parent and child nodes in the current module. If there are any unresolved nodes, it tries to load from the imported module that is defined in the IMPORTS section. If the unresolved object is not present even in the imported module, unresolved TC construct \{objectName1, objectName2, ...\} exception is thrown.

If the parsing level is NORMAL, SERIOUS, or CRITICAL, and if the MIB file contains errors, then the compiled files (cmi and cds files) or the serialized files (ser files) will not be created.

Checks for Various Parsing Levels

The following tables describes the different levels of parsing that can be set and their corresponding checks.

<table>
<thead>
<tr>
<th>S.No.</th>
<th>Level of Parsing</th>
<th>Checks</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Lenient</td>
<td>No Checks</td>
<td>This level accepts all types of MIB files. For example, it allows both SMIv1 and v2.</td>
</tr>
<tr>
<td>2</td>
<td>Normal</td>
<td>Default checks</td>
<td>This level is the default level conforming to the obsolete standards, such as RFC 1902, RFC 1903, etc. Most MIBs follow the obsolete standard.</td>
</tr>
<tr>
<td>S.No.</td>
<td>Level of Parsing</td>
<td>Checks</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>------------------</td>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>3</td>
<td>Serious</td>
<td>Serious Checks</td>
<td>This level strictly follows the current standard. It accepts the constructs with inter-operability and implementation problems.</td>
</tr>
<tr>
<td>4</td>
<td>Critical</td>
<td>Critical Checks</td>
<td>This level completely follows the SMIv1 and v2 standards. However, it does not accept the backward compatibility constructs, constructs with inter-operability and implementation problems, etc.</td>
</tr>
</tbody>
</table>

**Normal Parsing Level**

When the parsing level is normal (default parsing level), the following checks are included.

- OBJECT_IDENTIFIER_CONSTRUCT
- CHECK_DEFAULT

**Serious Parsing Level**

When the parsing level is serious, the following checks are done in addition to Normal checks.

- IMPORTS_CONSTRUCT
- MODULE_IDENTITY_CONSTRUCT
- OBJECT_TYPE_CONSTRUCT
- TRAP_TYPE_CONSTRUCT
- NOTIFICATION_TYPE_CONSTRUCT
- TEXTUAL_CONVENTION_CONSTRUCT

**Critical Parsing Level**

When the parsing level is critical, the following checks are done in addition to Serious checks.

- AGENT_CAPABILITIES_CONSTRUCT
- OBJECT_GROUP_CONSTRUCT
- NOTIFICATION_GROUP_CONSTRUCT
- OBJECT_IDENTITY_CONSTRUCT
- MODULE_COMPLIANCE_CONSTRUCT
- CHECK_IDENTIFIERS

When the parsing level is Lenient, none of the above checks are done.

There are some rules that a MIB file should follow, without which the MIB tree is not formed properly. When the parser encounters such violations, MibException is thrown.

- The OID construct should contain atleast two suboids.
- The second and its subsequent suboids should be number or nameNumber to identify their ancestors and their position in the MIB tree.
- In the OBJECT_IDENTIFIER construct, if the first suboid is a number, it should be 0, 1, or 2. If the suboid is a name Number, it should be ccit(0), iso(1), or joint-iso-ccit(2).
- The label of the last suboid should be same as the descriptor.
- The table entry should be the first child of the table node.
- The table entry should be defined as a child of the corresponding table object.
- The module name of the MIB file should start with uppercase letter.
- The TC name should not start with lowercase letter.
Therefore, the following checks are done even when the parsing level is set to lenient. These checks are termed as "Very Critical Checks" and are done irrespective of the parsing level.

- CHECK_ATLEAST_TWO_SUBOID
- CHECKSECOND_SUBOID
- CHECKFIRST_SUBOID
- CHECKLAST_SUBOID
- CHECKROW_OBJID
- CHECK_WRONG_TABLE_OBJECT
- VALIDATE_MODULE_NAME
- VALIDATE_TC_NAME
5.0 Creating a Simple SNMP Agent using MIB Compiler

5.1 Overview
This section will help you develop a simple SNMP agent using AdventNet Agent Toolkit. The basic operations that need to be performed for developing an SNMP Agent are:

1. Define MIBs using MIB Editor. To know how to define a MIB, please refer Defining a MIB section. It is not necessary that MIBs should be defined using MIB Editor. If MIBs are defined already, they can also be used.
2. Create a project and generate source code for the defined MIB using MIB Compiler.
3. Compile the generated code
4. Instrument the agent stub files and
5. Start the agent.

5.2 Creating a New Project
The first step towards creating an agent is Creating a Project for the agent. To create a project in MIB Compiler,

- Start the MIB Compiler application. You will be prompted with the following options.
  - Recent Workspaces
  - Open Workspace
  - New Project
  - Choose New Project here or from the File -> New Project menu bar.
  - You have to fill in the following details in the wizard for new project:
    - Create or Add to Existing Workspace - This will add the project to a new workspace. Use "Browse" option to store the project under an already existing workspace (.wsp).
    - Project Name - Specify a name for the project. By default it would be as snmpproject01. The project is identified by its .prj extension.
    - Project Directory - Specify the directory location in which the project should be placed. Use Browse option to navigate the location.
    - Click OK. The new project gets created and displayed under the workspace area.

5.3 Loading the MIB in MIB Compiler
Now that a project is created, you will have to load the MIB for which the agent has to be created. To load a MIB in the Compiler,

- Select MIB -> Load MIB option from the menu bar.
- Select a MIB from the list, say AGENT-SAMPLE-MIB. The MIBs in <Agent Toolkit Home>/mibs directory get listed by default. You can make choose these MIBs or from any other location where it is defined and
- Click Load MIB. The MIB gets loaded under the Loaded MIB Modules in the MIB View. You can also load Multiple MIBs following the same procedure.
5.4 Generating Source Code for the MIB

Source files can be generated either from MIB compiler or using commandline options. The MIB Compiler generates source files by reading the loaded MIB file. Bulk of effort involved in developing an agent is complete with the generation of source files.

The agent to be developed can be of v1, v2c, or v3 type. The Project -> Settings option in the menu bar can be used to configure the settings for an agent. By default, the version of the agent is v3.

**From MIB Compiler UI**

To generate source code for a given MIB,

- Select Build -> Generate Source from the menu bar.
- The files generated are listed in the Debug Window.
- On successful generation 'Code generation succeeded for project : snmpproject01.prj' can be seen.
- These files get stored under <Agent Toolkit Home>/snmpprojects/projectname/agent/src/com/myCompany/myPackage directory.
- They also get listed in the File View frame of MIB Compiler UI.

**From CommandLine Tool**

Agent Toolkit provides command line options for code generation. This feature is not designed to be used as a standalone utility but intended to add command line control to GUI-assisted agent development process. The project file serves as the input.

**Location of Commandline Utilities**

The script files responsible for invoking the Command Line Utilities are available under <ATKT_Home>/bin directory where <ATKT_Home> is the working directory of Command Line Utility. The relevant script files are generated under the working directory as SnmpAgentSourceGen.bat / SnmpAgentSourceGen.sh files.

**Operating the Commandline Utilities**

To accomplish the command line source generation, the following command has to be executed.

```
//Syntax of the Command in Windows Operating Environment
ATKT_Home\bin\XXXAgentSourceGen.bat -x <Project file location>

//Syntax of the Command in Linux/Solaris Operating Environment
sh ATKT_Home\bin\XXXAgentSourceGen.sh -x <Project file location>

//A typical input will look like this:
ATKT_Home\bin\SnmpAgentSourceGen -x C:\AdventNet\JavaAgent\snmpprojects\TestProject\TestProject.prj
```

5.5 Compiling the Generated Code

The MIB Compiler generates a set of source files for each MIB file. Some of the source or stub files need to be instrumented to include the desired function. This section is explained with the notion of developing a simple agent with the default values. Hence the instrumentation part is not covered here. To know more on the same, refer Instrumenting the generated code of a MIB section.
The next step is to compile the generated code to form a complete agent. The agent can be compiled either from MIB Compiler UI or from Command line. For compiling the code,

**From MIB Compiler UI**
- Select **Build -> Compile Source** from the menu bar.
- The files compiled are listed in the Debug Window.
- On successful compilation "Compilation succeeded." message can be seen.

**From Commandline**
- Open a command prompt/ terminal.
- Go to the `<Agent Toolkit Home>/snmpprojects/projectname` directory and set the classpath.
- Then, execute the following command:
  
  ```bash
  javac -d agent\bin agent\src\com\myCompany\myPackage\*.java
  ```
- The above command will compile all the source files under `agent/src` directory.
- If compilation is successful, the compiled class files will get stored under `<Agent Toolkit Home>/snmpprojects/projectname/agent/bin` directory.
- Now, all the files have been compiled and the simple SNMP agent has been created.

### 5.6 Starting and Stopping the Agent

Once the agent is developed, it can be started from MIB Compiler UI or from commandline.

**From MIB Compiler UI**
- Select **Build -> Start Agent** from the menu bar. By default, this would start the agent at port 8001. To stop the agent, use **Build -> Stop Agent** in the menu bar.

**From Commandline**
- Open a command prompt/ terminal.
- Go to `<Agent Toolkit Home>/snmpprojects/projectname/agent/bin` directory.
- Execute the `run.bat/ run.sh` command.
- The agent gets started at the default port 8001.
- To stop the agent, use Ctrl + C. This would kill the application.

Now that the agent is developed and started, the next step is to test the agent by sending SNMP requests. To know how to perform Testing on the Agent, please refer to Testing the Agent section.
6.0 Configuring Agent Settings

6.1 Overview
MIB Compiler allows you to configure the project from the User Interface (UI) itself. This configuration is done using the Settings panel invoked using any one of the following: (1) Project -> Settings menu option (2) Settings icon from the toolbar (3) Ctrl+Shift+S hot key.

This chapter describes various settings that can be configured while working with the agent. The "General Settings" topic talks about some of the general parameters that need to be configured such as Version, Compliance to Standards etc. and the "Source Generation Settings" lists the various parameters required while generating code for the agent.

6.2 Configuring General Settings
You can configure some general settings pertaining to your SNMP agent project using Project -> Settings -> General option. The options provided are :-

- Version
- Compliance to Standards
- Target, Community and Notification MIB Support
- Port Settings
- Debug Level
- Mode of Request Handling and
- Transport Protocol to be used.

The panel for configuring these parameters is shown:

6.2.1 SNMP Version
The supported SNMP versions are :

V1 : Selecting this option will generate the agent as SNMPv1 agent.
V2c : Selecting this option will generate the agent as SNMPv2c agent.
V3 : Selecting this option will generate the agent as SNMPv3 agent, which supports USM as the default security model and VACM as the default access control model.

The default version selected is v3.

6.2.2 Compliance to Standards
The AdventNet SNMP agent has implemented standard MIBs to provide SNMP v1, v2c, and v3 compliance. By default, the implementations of these standard MIBs are added to the agent while generating the source code, depending upon the version selected. If the standard MIB implementations need not be added to the agent, then disable the options provided for compliance.
6.2.3 Target, Community and Notification MIB Support

The option "Target, Community and Notification MIB Support" has to be enabled if Coexistence and Notification MIB support is required in a V3 agent. Only if this option is enabled can the respective tables under SNMPv3 panel be accessed.

6.2.4 Port Settings

The Agent Port option is used to specify the port at which the SNMP agent, being developed, is to be started. By default, the SNMP agent is started at port number 8001. You can specify any other port from which the agent will be started. The agent processes the request (from the manager) from this port.

The Trap Sending Port is the port used by the agent to send traps to managers. By default the agent sends traps through port number 8002. You can specify the port from which the agent has to send traps, using this option.

6.2.5 Debug Level

When the agent is running, it generates log messages that contains details regarding the agent processing. These log messages are stored in a text file called agent.log. Each log message has an associated log level. The level gives a rough idea about the importance and urgency of a log message. You can select the level of log message to be recorded in the log file. Each of these Debug levels are explained in Logging in SNMP Agent section.

6.2.6 Mode

Asynchronous Message Processing enables the manager to send many requests to the SNMP agent simultaneously. This saves lot of time as the manager need not wait for the response from the agent for the previously sent message. The processing also enables more than one manager to access the SNMP agent and get the processes done in parallel.

Asynchronous Message processing is achieved by using Multi-threading. Every functionality of the SNMP agent runs in a separate thread and can be accessed for Asynchronous Message processing.

Enabling Asynchronous Message Processing

Asynchronous Message Processing can be enabled either using MIB Compiler UI options or using API calls.

Using MIB Compiler UI

1. Select Project -> Settings from the menu bar of MIB Compiler UI.
2. A dialog appears with Settings tree on the left frame of the box.
3. Choose General Panel from the Settings tree.
4. Here, you can enable or disable Asynchronous Message Processing support.

After enabling the Asynchronous Mode, the combo box for configuring the number of threads gets activated. You can configure the maximum number of threads that needs to be handled by the SNMP agent for Asynchronous Message Processing. Here you can increase or decrease the number of threads. The default number is 1 and the maximum number of threads that are permitted is 19.

Using API Calls

For running in asynchronous mode, add the following piece of code

```java
super.setAsyncMode(true);
```
in the main file generated. By default the number of threads created will be one and the maximum number of threads that can be handled is 19. This can be altered by calling `super.setMaxThreads(int)` in the main file. This code gets generated, if async mode is set "true".

6.2.7 Transport Protocol

The Transport Protocol option in General panel of `Project -> Settings` menu specifies the communication protocol that will be used by the developed agent.

TCP/IP

Selecting this option will use TCP/IP as the communication protocol.

**Note:** The agent developed using TCP/IP as the communication protocol cannot be tested using the MIB Browser. Please use commandline applications (TCP/IP) for testing. They are available in `<Agent Toolkit Home>/examples/snmp/low_level_tcpapps` directory.

UDP/IP

Selecting this option will use UDP/IP as the communication protocol. The default transport protocol used is UDP/IP.

6.3 Configuring Source Generation Settings

Source generation settings allow you to customize the Agent and make it respond as required. These settings relate to parameters required for generating java source files for the Agent. If these settings are not configured, MIB Compiler generates an Agent which responds with default values. These parameters can be configured using the General option of `Source Generation panel` invoked using `Project -> Settings` menu. The Storage Model option under the `Source Generation Panel` helps you to store the details of the Agent in a Text file, XML file or RAM. For more information refer to "Supported Storage Types Option".

This chapter describes how to configure the various parameters such as:

- Package Name / Agent Name
- Instrument with Interface
- Merging with Previous Stub Files
- Synchronized Set Methods
- Initialize with Default Values
- Generate Trap on Set
- Generate code for API Tables Info

The panel for setting the above said parameters is shown in the screen shot below.

6.3.1 Package Name/Agent Name

**Package Name**

The Package Name provided in the Source Generation panel, will be taken as the package name for the generated Java files. The default package name provided is `com.myCompany.myPackage`. You can also specify any other name other than the default name provided.
Agent Name

The Agent Name provided in the Source Generation panel, refers to the Main File Name. By default, the Main file name gets generated as AdventNetSnmpAgent.java. You can specify any other name other than the default name.

6.3.2 Instrument with Interface

The AdventNet Agent Toolkit supports interface mode of code generation, if this option is enabled. For a scalar node/non-table group node, the Interface file will be generated as xxxInterface.java. For a columnar node/table node, the Interface file will be generated as xxxEntryInterface.java file. You have to implement this interface.

For example, the default code generated for AgentSystem group of AGENT-SAMPLE-MIB in the main file is given below:

```java
public AgentSystemInterface
    getAgentSystemInstance()
    {
        AgentSystemInterface instrument = null;
        if(listener != null){ instrument
            =listener.getAgentSystemInstance();
        } if(instrument == null){ instrument = new
            AgentSystemInstrument();
        } return instrument;
    }
```

In the above code, AgentSystemInterface instruments the new AgentSystemInstrument() which is the default file generated by the MIB Compiler. You can replace this with your own instrument file, say, MyAgentSystemInstrument(). Now, the code will look like this,

```java
public AgentSystemInterface
    getAgentSystemInstance()
    {
        AgentSystemInterface instrument = null;
        if(listener != null){ instrument =
            listener.getAgentSystemInstance();
        } if(instrument == null){ instrument = new
            MyAgentSystemInstrument();
        } return instrument;
    }
```

Thus, you need not edit any generated Java files from MIB Compiler. You also need not worry on migrating to releases (or) upgrading the MIB knowledge of SNMP Agent.

6.3.3 Merge With Previous Stub Files

On regenerating the source files, the newly generated files will be over written or will not be modified. The information that you added will be lost if it is overwritten. To avoid this and merge your code with the newly generated code, you should select this option. By default, merge with stub files option is checked.

This option is used when you are moving from one version of the product to another version. If there is any change in the generated code, then using the Merge option you can merge your changes with the newly generated code of the MIB Compiler. If the merging is not proper, a new file is created and thus intimates you that the merging for that particular file could not be done due to certain errors.

For more information on Merging, please refer to the topic Code Merging in "Instrumenting the Generated Code" (8.0) section.
6.3.4 Synchronized Set Methods

The Synchronized set methods option in the Source Generation panel generates code with the set methods in the entry/instrument files as synchronized. This is extremely useful when running the Multiple Agents in a single JVM with the same stubs answering for requests on all the Agents.

6.3.5 Initialize with Default Values

This option, when enabled initializes all variables to the default values given in the MIB. If DEFVAL field is not present in the MIB then MIB Compiler will generate source code with default values based on data types.

For example, If Data Type is integer and DEFVAL is not present in the MIB, then MIB Compiler will initialize the variable to 1. For string type MIB Compiler will initialize the variable to "< node name > not initialized".

6.3.6 Generate Trap on Set

If the MIB has trap definition then the MIB Compiler will generate corresponding trap functions in the "AdventNetSnmpAgentTrap.java" file. On calling this function the corresponding trap will be sent to the registered Managers. This function can be called from anywhere in the Agent.

If the Generate Trap on Set option in the Source Generation panel of Project -> Settings menu is selected, then the MIB Compiler will generate code fragment in the set method of the handler file of each trap variable. Thus, a trap is sent on every SET operation performed on the trap variable. If this option is not selected, then the code fragment will not be generated in the handler file.

Trap variable is one of the objects defined in the trap or notification construct of the MIB. Hence, it is required to define the trap type or notification type before generating traps on SET.

6.3.7 Failover Support for API Tables

This option enables the provision to get the API table info from the Main Agent. To know the necessity of this option please refer to the section Failover Support for API Table.

6.3.8 Disable Startup Trap

By default, the generated agent sends coldstart trap at the agent startup to the managers being configured. The user can disable the trap at agent startup by enabling the option provided in the MIB Compiler UI by choosing Project->Settings->Source Generation ->Disable Startup Trap.

6.4 Packaging the Agent

When an SNMP Agent is generated, the code for the Agent is generated in the <Agent Toolkit Home>/snmpProjects/<projectName>/agent/src directory. But some library files required for running the Agent are accessed from folders in <Agent Toolkit Home> directory.

When the Package Agent option is selected from the Build menu, the required binaries are copied into <Agent Toolkit Home>/snmpprojects/<projectName> directory. This makes the Agent self-contained and it can be used as an independent project. The Agent project is made available as a zip file and thus it can be run in any environment.
The changes that occur when this option is selected are:

- The jars required for running the Agent will be copied from `<Agent Toolkit Home>/jars` directory to the `/snmpprojects/<projectName>/agent/bin/jars` directory.
- The working directory of the SNMP Agent specified in the `run.bat/run.sh` changes automatically to reflect the modifications mentioned above.

### 6.5 Configuring Root OID

Root OID of a MIB can be changed after generating the code for an Agent. By modifying the root oid, the Agent is accessed from that particular oid. It is not necessary that you have to stick on to an oid that is already present in the MIB. To configure the Root OID, use the API call given below.

Using API calls

Add the following piece of code in the generated Main file, below the code for traplistener.addRegistrationListener.

```java
hdlr.setRootOid(required root, existing root);
```

For example, if the existing root is .1.3.6.1.2.1 and you prefer to change this to .1.3.6.1.7.2 this feature can be used. These root oids have to be specified in the above said code and after compilation and you will be able to access the Agent only with the root oid .1.3.6.1.7.2 thereafter. The required root OID and existing root OID are to be represented as a String.

### 6.6 Initializing the Agent by Reading a Configuration File

The SNMP agent can be initialized by defining the parameters in a configuration file, `<projectname>AgentStartup.xml`. The xml file gets generated at the `<Agent Toolkit Home>/snmpprojects/projectname/agent/bin/conf` directory while generating code using MIB Compiler.

The configuration file is used to persist the agent configuration details which comprises SNMP version, agent port number, port from which agent sends the trap, debug level, compliance details, transport protocol, configuration table-related details etc. The advantage of the configuration file is that it can be edited dynamically i.e., the agent can be initialized again by just restarting the agent. This avoids the user for regenerating, compiling, and starting the agent again.

The following are the steps to initialize the agent after editing the agent configuration file:

1. Generate an SNMP agent using the MIB compiler. The configuration file is generated in the above-mentioned location with the details as shown below:

```xml
<AgentSettings>
    <SNMP targetComNotifSupport="false"  v1Compliance="true" debugLevel="3"  agentPort="8001"  trapPort="8002" version="V3"  v3Compliance="true"  v2Compliance="true" protocol="UDP" ></SNMP>
    <MsgProcessing maxThreads="2"  asyncMode="true"
</MsgProcessing>
    <ConfigTables>
        <ConfigTable dynamicConfiguration="true"
            tableName="V1V2TfTable"  isEnabled="false"
            persistType="xml"  persistence="false"
            fileName="V1V2TrapForwardingTable.xml"
        ></ConfigTable>
        <ConfigTable dynamicConfiguration="true"
            tableName="Vac1Table"  isEnabled="false"
            persistType="xml"  persistence="false"
            fileName="ViewAccessControlTable.xml"
        ></ConfigTable>
    </ConfigTables>
</AgentSettings>
```
<ConfigTable dynamicConfiguration="true" tableName="V3TfTable" isEnabled="true" persistType="xml" persistence="true" fileName="V3TrapForwardingTable.xml"></ConfigTable>
<ConfigTable dynamicConfiguration="false" tableName="SysORTable" isEnabled="true" persistType="xml" persistence="true" fileName="SysORTable.xml"></ConfigTable>
<ConfigTable dynamicConfiguration="true" tableName="AclTable" isEnabled="true" persistType="xml" persistence="true" fileName="AccessControlTable.xml"></ConfigTable>
</ConfigTables>
</AgentSettings>

2. Compile and start the agent
3. To edit the file, stop the agent.
4. Open the generated configuration file and make the changes.
5. Restart the agent.

Note: The functionality is also enhanced by a DTD file, AgentStartup.dtd generated under the same location, which is used to validate the changes made in the configuration file. The user can also view the DTD file to know the agent configuration details.

6.7 Command Line Options

In addition to the configuration file provided for initializing the agent, command line option can also be used. The options can be used by giving <Agent Toolkit Home>\snmpprojects\projectname\agent\bin>run - in the command line. The following are the keys listed and can be used as mentioned:
- [-d Logging level (0-6)] - To change the debug level which could be between 0 to 6
- [-p port] - To change the port at which the agent is running.
- [-v version] - To change the version which can be v1/v2c/v3
- [-t TrapSendingPort] - To change the trap sending port.

6.8 Other Options

Following options present under the Project -> Settings menu are discussed in the topics mentioned below.

<table>
<thead>
<tr>
<th>Options</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage Model options in Source Generation Panel</td>
<td>discussed under the topic Supported Storage Types for Scalars and Tables (Section 16.0).</td>
</tr>
<tr>
<td>V1V2 Authentication Panel</td>
<td>discussed under the topic Authenticating SNMPv1/v2c Requests (Section 10.0)</td>
</tr>
<tr>
<td>Proxy</td>
<td>discussed under the topic Implementing SNMP Proxy (Section 13.0)</td>
</tr>
<tr>
<td>Trap</td>
<td>discussed under the topic Sending Traps &amp; Informs (Section 12.0)</td>
</tr>
<tr>
<td>SNMPv3</td>
<td>discussed under the topic Configuring SNMPv3 Agents (Section 14.0)</td>
</tr>
<tr>
<td>Compile Options</td>
<td>This panel shows the jars used in the classpath of the Project. The jars used by default are listed out. You can add or remove a jar based on your requirement.</td>
</tr>
</tbody>
</table>
7.0 Testing the Agent

7.1 Overview

This section explains the various kinds of SNMP operations that can be performed on the Agent for Testing purposes. SNMP Operations include SNMP GET, GET-NEXT, SET, WALK, GET-BULK, and so on. Let us see in detail how these have to be used on the Agent. It is assumed that a v2c Agent is created using AGENT-SAMPLE-MIB following the steps given in Creating a Simple Agent using MIB Compiler section.

7.2 Starting MIB Browser

An SNMP Manager environment is created using the MIB Browser tool. Please follow the steps given below to start the MIB Browser application and configure the Manager settings.

1. To start this tool, use MIBBrowser.bat or .sh file in <Agent Toolkit Home>/bin directory.
2. Load the MIB e.g, AGENT-SAMPLE-MIB for which an SNMP Agent is created.
3. Click the Settings sub menu in the Edit menu bar.
4. A wizard opens up with a General Tab open.
5. Check the version, v2c (for our case).
6. And click OK.
7. The Manager's Port can be specified in the field given for Port in the MIB Browser main UI.

7.3 Using MIB Browser

The AdventNet MIB Browser can be used for MIB browsing and to view and operate on data available through a SNMP agent. The MIB Browser can be configured in tune with performing the SNMP operations.

In order to configure the MIB Browser,

- Click on the MIB Browser Settings button or
- Select Edit-->Settings menu item or
- Use a short cut of Alt + S

On performing any of the above action, a dialog box opens which is the MIB Browser settings frame that has three-tabbed panel. The three tabs are

- General
- Mib Settings
- v3Settings

Now, let us have an overview on the configuration of the MIB Browser under various tabs.

- Setting Common Parameters
- Setting MIB Parameters

This topic also covers Graphs and Debugging and Decoding of Messages.
7.3.1 Setting Common Parameters

The common parameters are set in the General tab of the MIB Browser Settings frame. The General field. The figure depicted below shows the General Settings in the MIB Browser.

The various protocol-related options to SNMP are listed in the table below.

<table>
<thead>
<tr>
<th>Options</th>
<th>Default Values</th>
<th>Other Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Snmp Version</td>
<td>v1</td>
<td>v2c or v3</td>
</tr>
<tr>
<td>Time out</td>
<td>5 sec</td>
<td>any user defined value</td>
</tr>
<tr>
<td>Retries</td>
<td>0</td>
<td>any user defined value</td>
</tr>
<tr>
<td>Encoding</td>
<td>ISO8859_1</td>
<td>any encoding scheme that supports text format</td>
</tr>
<tr>
<td>Max repetitions</td>
<td>50</td>
<td>any user defined value</td>
</tr>
<tr>
<td>Non-repeaters</td>
<td>0</td>
<td>any user defined value</td>
</tr>
</tbody>
</table>

**Note:**
- Timeout is the time interval that an application waits for a response message from an agent before timing out. Retries is the number of times a request is sent when a timeout occurs. If the retry value is 0, the request is re-transmitted on timeout.
- The options Max-repetitions and Non-repeaters are enabled only when the SNMP version is set to either v2c or v3. This is because, the GETBULK operation is available only in v2c and v3. A GETBULK request is performed by giving an OID along with two other parameters, Max Repetitions value and Non Repeaters value.
• Encoding, in general, means modifying information into the required transmission format. Computers around the world store information using a variety of encoding schemes. Net MIB Browser support the ISO8859_1, which means ISO 8859_1, Latin alphabet No.1. There are various other encoding schemes that support various text formats. You can use the encoding scheme that best suits your requirement while performing SNMP operations. To view the encoding scheme that is supported by Java Development Kit, see: http://java.sun.com/products/jdk/1.1/docs/guide/intl/encoding.doc.html.

The Validate Broadcast Address check box enables you to check the validity of the broadcast address provided. You need to provide the Netmask address to validate the broadcast address. A Netmask is a string of 0's and 1's that hides the network part of the IP address and allows only the host ID to remain.

In the v3 Options section, Context Name and the ContextID are to be provided as additional parameters for an SNMPv3 request. An SNMP context name is a collection of management information accessible by an SNMP entity. An item of management information may exist in more than one context. An SNMP entity potentially has access to many contexts. In other words, if a management information has been defined under certain context by an SNMPv3 entity, any management application can access that information by giving that context name. The ContextID uniquely identifies an SNMP entity that may recognize an instance of a context with a particular context name within an administrative domain.

The next section is the v3Settings section. The following are the security-related parameters for accessing the SNMPv3 agents. You can add, modify, and delete users by clicking the Add, Modify, Delete buttons.

<table>
<thead>
<tr>
<th>Options</th>
<th>Default Values</th>
<th>Other Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>User name</td>
<td>null</td>
<td>any user-defined value</td>
</tr>
<tr>
<td>Security level</td>
<td>noAuth noPriv</td>
<td>Auth noPriv and Auth Priv</td>
</tr>
<tr>
<td>Authentication Protocol</td>
<td>MD5 (if authentication is chosen in security level)</td>
<td>SHA</td>
</tr>
<tr>
<td>Privacy Protocol</td>
<td>CBC-DES (if privacy is chosen in security level)</td>
<td>not available</td>
</tr>
<tr>
<td>Authentication password</td>
<td>any user defined value</td>
<td>-</td>
</tr>
<tr>
<td>Privacy password</td>
<td>any user defined value</td>
<td>-</td>
</tr>
<tr>
<td>Target host</td>
<td>localhost</td>
<td>any host with SNMPV3 agent or proxy agent</td>
</tr>
<tr>
<td>Target port</td>
<td>161</td>
<td>any user-defined port</td>
</tr>
</tbody>
</table>

If the security level is "NoAuthNoPriv", no additional parameters are required. If the security level is "AuthNoPriv", the parameters AuthProtocol and AuthPassword are be set. If the user security level is "AuthPriv", the privacy password needs to be set in addition to the other parameters.
The v3Settings section has an option for storing the v3 table entries. The v3 table entries can be stored in:

- A serialized file
- A database

**Storing table entries in a serialized file**

To enable serialization of v3 table entries, select the Save v3 Settings to File option. If this option is selected, the user information is stored in the serialized files namely UserEntry.ser and EngineEntry.ser. When MIB Browser is invoked the next time, the serialized files are deserialized and the v3 table is updated. The advantage of storing table entries in serialized files is that the operation is faster.

**Storing table entries in a database**

To store v3 table entries in a database, select the Save v3 Settings to Database option. To use this, the database connection has to be established. Clicking the "Database Settings" button displays the Database Parameters dialog box. The image of the dialog box that appears is given below.

Enter the necessary database parameters in this dialog box, and click the OK button. If the database connection is established successfully, all the user information entered is saved in the database. When the MIB Browser is invoked the next time, v3 details are restored and the v3 table is updated.

The advantages of storing v3 table entries in a database are:

- Scalability - Any number of entries can be maintained in the database.
- Accessibility - All the authenticated users of the database can access the entries.

The last section is the field entry section in which the corresponding fields in the v3 table are displayed for data entry. The various buttons available in the field entry section are the Add, Modify, and Delete buttons.

To add an entry, enter the required parameters in the respective fields and click the Add Entry button. Based on the parameters and the security level, Discovery and Time Synchronization are done and USM Table is updated and listed in the v3 table.

To modify an entry, select the entry in the v3 table, modify the required fields, and click the Modify button. Time Synchronization is done and the USM table is updated and listed in the v3 table.

**Note:** The NoAuthNoPriv entry cannot be modified. Only the password fields in the AuthNoPriv entry and AuthPriv entry can be modified.

To delete an entry in the v3 table, select the entry in the v3 table and click the Delete button. The entry is removed from the USM table.

**Note:** The settings are saved only on exiting the MIB Browser application and not every time the settings are modified. The 'Save v3 Settings to File' option is enabled only in the MIB Browser application and not in the MIB Browser applet because of certain security restrictions in applets.
7.3.2 Setting MIB Parameters

The Mib Settings tab is focused on the loading of MIBs in MIB Browser. The image below displays the Mib Settings tab.

The first section gives the MIB loading options. The next section displays the various parsing levels.

7.3.2.1 MIB Loading Options

The various Options available for Loading the MIB are:

- Load MIBs directly
- Load MIBs From Compiled File
- Load MIBs From Database

Note: "Load MIBs From Compiled File" is by default selected when a MIB Browser is used as an Application. In case of MIB Browser used as an Applet, the default option selected is "Load MIBs directly".
7.3.2.1 Load MIBs directly

The MIB file is usually read and parsed into MIB modules and displayed in the MIB tree. In this case, when you load a MIB file, it is parsed and then loaded. This is time consuming because parsing is done every time a MIB file is loaded.

7.3.2.1.2 Load MIBs From Compiled Files

The next option is loading of the MIB files as compiled files. The main advantage here is that the loading time is reduced. This ultimately leads to the improvement in performance. To store the MIB information in a formatted structure, the following two new file types are made available.

- **cki** - This file type contains MIB information, such as MibNode, MibModule, naming hierarchy, etc.
- **ds** - This file type contains the description and reference of the MIB nodes.

When the option Load MIBs from Compiled File is selected, the MIB Browser loads the MIB from the cmi and cds files. If these files are not present, MIB Browser parses the MIB file, writes the output in the cmi and cds files, and loads the MIB file. For example, if we load the RFC1213-MIB, the MIB file is parsed and stored in the compiled MIB files as RFC1213-MIB.cmi and RFC1213-MIB.cds provided RFC1213-MIB is the module name of the RFC1213-MIB file.

When this MIB file is loaded again, the MIB is loaded from the cmi and cds files and no parsing is done. The advantage of using this option is, we need not parse the MIB each time we load, thus optimizing the load time and improving the performance. While loading the compiled MIBs, we need to only load the cmi file. The cmi file has a reference to the cds files. Therefore, the cds file need not be loaded directly.

**Note:** Any changes made to the MIB file after it has been loaded as a compiled MIB file, are not reflected when it is loaded again. You have to remove the existing cmi and cds files and load the MIB again to get the latest changes shown. To overcome this, select the option "Overwrite existing Compiled MIB Files" can be selected. If this option is set to true, the cmi and cds files are created each time the MIB is loaded. However, enabling this option is recommended only if you have changed the contents of the MIB file. Otherwise, this serves as a redundant option and increase the load time of the MIBs.

7.3.2.1.3 Loading MIB's from database

The third option is loading of the MIB files from a database. The MIB files can be stored in any RDBMS such as MySql or Oracle. Applications can load these MIB files directly from the database. This feature is particularly useful when the number of MIB files to be loaded is high in number.

The MIB Browser uses JDBC (Java Database Connectivity) for the database support. Applications should use a valid JDBC driver of the respective databases to enable the database support.

Selecting the option Load MIBs from Database enables the text fields in the JDBCParams section. On initializing the necessary database parameters in this section, the database support can be provided for loading MIBs. The various JDBCParams required are:

- **Drivername** - Name of the Database driver.
- **URL** - URL pointing to the Database file name
- **UserName** - user name
- **Password** - password
After selecting this loading option, select the required MIB file to be loaded from the Open tab of the Load a MIB File dialog box. If the selected MIB file is already present in the database, the MIB file is read, parsed, and loaded from the database. If the MIB file does not exist in the database, MIB Browser parses the MIB file, writes the output to the database and then loads the MIB file.

### 7.3.2.2 Parsing MIBs

MIB Browser enables you to parse the given MIB file and check for the macro constructs. It allows different levels of parsing and the parsing is done as per the standard definition of the macros. The parsing levels can be set in the MIB Browser Settings dialog box. The following tables describes the different levels of parsing that can be set and their corresponding checks.

<table>
<thead>
<tr>
<th>S.No.</th>
<th>Level of Parsing</th>
<th>Checks</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Lenient</td>
<td>No Checks</td>
<td>This level accepts all types of MIB files. For example, it allows both SMIv1 and v2.</td>
</tr>
<tr>
<td>2</td>
<td>Normal</td>
<td>Default checks</td>
<td>This level is the default level conforming to the obsolete standards, such as RFC 1902, RFC 1903, etc. Most MIBs follow the obsolete standard.</td>
</tr>
<tr>
<td>3</td>
<td>Serious</td>
<td>Most checks throw exceptions on first misbehavior.</td>
<td>This level strictly follows the current standard. It accepts the constructs with interoperability and implementation problems.</td>
</tr>
<tr>
<td>4</td>
<td>Critical</td>
<td>All possible checks throw exceptions on first misbehavior</td>
<td>This level completely follows the SMIv1 and v2 standards. However, it does not accept the backward compatibility constructs, constructs with interoperability and implementation problems, etc.</td>
</tr>
</tbody>
</table>

Applications, while loading MIB files, perform the following operations.

- Parsing and validating the syntax of the MIB module
- Constructing the MIB module into the tree structure

While performing the parsing and validation of the MIB files, if the MIB modules fail to conform to the SMI standards the loading will not be not done. However, the application requirements might mandate the loading of the non-standard files. On the other hand, some applications might require a stricter check on the compliance to the standards.

The parsing and validating the syntax of the MIB file can be made configurable to suit the application requirements. MIB Browser handles this by providing parsing levels which facilitate to select the level of parsing required by the applications.

In addition to the above four parsing levels, MIB Browser supports another level, which is user-defined. In case of user-defined level, you can define your own parsing level with the required checks at runtime.

**Note:** It is recommended to use the higher parsing level (SERIOUS, CRITICAL) for validating the MIB file and not for loading the MIB file in the application. It affects the performance of the application while loading the MIB files, because it takes considerable amount of time and resources, such as memory, CPU usage, etc.

### User Defined Parsing Levels

In addition to the four parsing levels, you also have another level, which is user-defined to define your own parsing levels at runtime. To add a user-defined parsing level, select the User-Defined option and click the Add button in the User-Defined Levels section. This displays the Customized Level dialog box.
By default, all the checks are included. Provide a name for the level in the Level Name text field. To add or remove checks from the level, select or deselect the checks and click OK. Note that if you select (deselect) a parent check, all its child checks are also selected (or deselected). Click OK to add this level to the user-defined level list.

The level of the parser has to be set in the MibParser before loading a MIB. This level, once set, is used for subsequent MIBs loaded. If the level needs to be modified for the next set of MIBs loaded, it has to be set again in the MIB Parser. In the Mib Settings tab of the MIB Browser Settings dialog box, select the required Parsing level, and click Apply.

The MIB file can contain one or more MIB modules. MIB Browser loads all the dependency files to resolve the MIB module. If the dependency file is not present, the IMPORTS failed error is thrown.

The parsing level can be set for the dependency file by selecting the Import File option and choosing Parsing Levels.

7.3.3 Graphs

One of the vital features of the Net MIB Browser is the Graphs. The graphs depict the real-time plotting of the SNMP data. Currently, two types of graphs are supported—line graph and bar graph. The SNMP data to be polled should be of integer or unsigned integer data type. Typically, the values that are plotted will be of type Counter, Gauge, or Timeticks.

Although the steps followed in invoking a graph and working on it are similar to the line graph and bar graph, let us understand it better under the following two sections:

- **Line Graph**
- **Bar Graph**

**Line Graph**

The line graph depicts the real-time plotting of the SNMP data. Follow the steps below to invoke a line graph from the MIB Browser.

1. Specify the proper agent hostname or IP address in the host field of the MIB Browser.
2. Load the MIB in the MIB Browser.
3. Specify a valid variable. The variable must be an integer or unsigned integer (Counter, Gauge, Timeticks). This variable can be entered directly in the variable field or it can be chosen by browsing through the MIB in the Mib Tree.
4. Click on the ‘View real-time graph’ button in the toolbar (or) choose View-->Line Graph menu item from the Menu Bar (or) use a shortcut Alt+L. By default, the Line graph option is selected in the MIB Browser settings panel, hence the line graph would be invoked when you do any of these operations.
5. This would invoke the updated Line graph showing the results of periodically polling the specified agent for the specified OID.

The various options available in the line graph are tabulated below:

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Polling Interval (sec's.)</td>
<td>This specifies the polling interval time. The default value is 5 secs. You can change the interval time as desired.</td>
</tr>
<tr>
<td>Average over Interval?</td>
<td>Generally the actual values are plotted, selecting this option would take the average of the values.</td>
</tr>
<tr>
<td>X-axis Scale (&gt;300)</td>
<td>This specifies the X-Axis scale. The default value is 300 secs and this is the minimum seconds. Changing this would alter the X-axis scale of the table. By default it is disabled, only on clicking 'Show polled values?' option the x-axis scale option is enabled.</td>
</tr>
<tr>
<td>Options</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Show Absolute Time?</td>
<td>By default the time is depicted in the graphs as only seconds. Selecting this option would give you the time in hours:Secs.</td>
</tr>
<tr>
<td>Max Poll Duration (sec's)</td>
<td>In order to see all the polled values in a particular time period, this option is used. This option by default it is disabled. Only on clicking 'Show polled values?' option this is enabled. the default value is 3600 secs.</td>
</tr>
<tr>
<td>Show Polled Values?</td>
<td>By default this option is disabled. When selected, all the polled values in a particular period of time is shown. Only on selecting this option the Max Poll Duration and Log FileName options are enabled.</td>
</tr>
<tr>
<td>Log FileName</td>
<td>The file name for the log file can be set here. By default the log file name is graph.txt. If the adjacent &quot;Log Polled Values&quot; is selected all the polled values are logged in this file. This option is not enabled when the MIB Browser is run as an applet, this is because of the security restrictions in case of applets.</td>
</tr>
<tr>
<td>Log Polled Values?</td>
<td>Selecting this option would log the polled values. This would enable the option Log FileName for the log file name to be set.</td>
</tr>
<tr>
<td>Show Absolute Counters</td>
<td>By default the graph plots only the difference between the two values. On selecting this the plotting of the absolute value is performed.</td>
</tr>
<tr>
<td>Restart</td>
<td>The restart button is used to restart the polling</td>
</tr>
<tr>
<td>Stop</td>
<td>The stop button is used to stop the polling</td>
</tr>
<tr>
<td>Close</td>
<td>The close button is used to close the graph window.</td>
</tr>
</tbody>
</table>

Yet another way of invoking a graph is through the Table options. The MIB Browser can plot multiple graphs showing values for different variables from different hosts.

**Bar Graph**

The bar graph depicts the real-time plotting of the SNMP data. Follow the steps below to invoke a bar graph from the MIB Browser which is similar to invoking a line graph.

1. Specify the proper agent hostname or IP address in the host field of the MIB Browser.
2. Load the MIB in the MIB Browser.
3. Specify a valid variable. The variable must be an integer or unsigned integer (Counter, Gauge, Timeticks). This variable can be entered directly in the variable field or it can be chosen by browsing through the MIB in the Mib Tree. In case of a Scalar OID, the bar graph works fine by appending a .0 to it. When it is a Columnar OID, you need to append the index in order to enable the plotting of bar graph.
4. Click on the 'View real-time graph' button in the toolbar (or) choose View -->Bar Graph menu item from the Menu Bar (or) use a shortcut Alt+B. By default, the Line graph option is selected in the MIB Browser settings panel, hence the line graph would be invoked when you do any of these operations. If you need to invoke the Bar graph by default, then opt for the Bar Graph option in the Setting Common parameters in the MIB Browser Settings option.
5. This would invoke the updated Bar graph showing the results of periodically polling the specified agent for the specified OID.

**Note:** The bar graph does not have the option of plotting multiple variables in the graph. Therefore, in case of a columnar OID, you need to append the instance of the index in order to enable the plotting of Bar graph. For example, to plot a bar graph for the values of the first row of the columnar OID ifOperStatus (ifTable), you need to first select the node ifOperStatus. Then, in the Object ID text field, append ".1" with the OID and select View --> Bar Graph from the menu. This plots the value of the first row of the column ifOperStatus.
The various options available in the bar graph is tabulated below:

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Polling Interval (Sec.)</td>
<td>This specifies the polling interval time. The default value is 5 secs. You</td>
</tr>
<tr>
<td></td>
<td>can change the interval time as desired.</td>
</tr>
<tr>
<td>Average over Interval?</td>
<td>By default, the actual values are plotted, selecting this option would</td>
</tr>
<tr>
<td></td>
<td>take the average of the values.</td>
</tr>
<tr>
<td>Range</td>
<td>Altering the range would change the X-axis scale of the table. The default</td>
</tr>
<tr>
<td></td>
<td>value is 300 secs in the drop-down list box. The other options available in</td>
</tr>
<tr>
<td></td>
<td>it are 600 secs and 1000 secs.</td>
</tr>
<tr>
<td>Show Absolute Time?</td>
<td>By default the time is depicted in the graphs as only seconds. Selecting</td>
</tr>
<tr>
<td></td>
<td>this option would give you the time in hours:Secs.</td>
</tr>
<tr>
<td>Restart</td>
<td>The restart button is used to restart the polling.</td>
</tr>
<tr>
<td>Stop</td>
<td>The Stop button is used to stop the polling.</td>
</tr>
<tr>
<td>Close</td>
<td>The close button is used to close the bar graph window.</td>
</tr>
</tbody>
</table>

### 7.3.4 Debugging and Decoding

The MIB Browser application provides facility to view the debug output of the SNMP operations.

**Invoking the Debugging window**

1. Click on the Debug icon in the toolbar (or) choose View-->Debug from the menu bar (or) use a shortcut of Alt+D. This would invoke the debug window.
2. As long as this window is opened, debugging is turned on, and debugging output is displayed in the debug window. When this window is closed the debugging is turned off. The image below depicts a Debug window.
3. The three icons in the debug window provide the following function:
   - Save MIB Browser Debug Results - Saves the debug information to a file
   - Print MIB Browser Debug Results - Prints the debug information to a file
   - Snmp Decoder - Switches to Decoder window

Invoking the Decoder Window

1. To switch from the Debug window to the Decoder window, click on the Decoder icon.
   To switch from Decoder window to the Debug window click on the Debug icon.
2. The three icons in the debug window provide the following function:
   - Save MIB Browser Decoder Results - Saves the debug information to a file
   - Print MIB Browser Decoder Results - Prints the debug information to a file
   - Debug - Switches to Debug window

Note: The Save and Print options are available only when the MIB Browser is invoked as an application and not in an applet. This is because of security restrictions in case of applets.

Performing the Decoding operation

The Snmp Decoder is used to decode the SNMP debug messages. The decoding can be done by two ways by using the
   - Copy-Paste option and
   - File option

The figure below depicts the Snmp Decoder.
Copy-Paste option (This method is applicable for both MIB Browser application and applet).

This option can be used for frequent debugging.

- Copy the debug information whatever is displayed in the debug window
- Click on the Snmp Decoder icon. This will switch to the decoder window.
- Paste the debug message into the "Hex PDU" text area.
- Click on the Decode button.

The decoded message will be displayed in the bottom panel of the decoder window.

File option (This method is applicable only for MIB Browser application because the saving and loading of debug information files is done only in applications and not in applets). This option can be used if the debug message was stored in a file and decoding has to be done.

- Load the file which contains the debug information by clicking on the
- Browse button in the Debug frame. (or) Enter the URL in the File
- URL text field and click 'Enter' key.

This will display the decoded message in the bottom panel of the decoder window.

**Important:** You can select the entire PDU debug message displayed in the debugger window with all the strings and paste it in the Hex PDU text area or load it in the decoder. The decoder will decode all the PDU dumps leaving the informative strings. The limitation in this is that the two continuous PDUs should have a string delimiter as a new line in between them. A sample PDU is depicted below

```
30 26 02 01 00 04 06 70 75 62 6c 69 63 a0 19 02 01 04 02 01
00 02 01 00 30 0e 30 0c 06 08 2b 06 01 02 01 01 05 00 05 00
Packet from: 192.168.1.215 : 161
DATA:
30 2e 02 01 00 04 06 70 75 62 6c 69 63 a2 21 02 01 04 02 01
00 02 01 00 30 16 30 14 06 08 2b 06 01 02 01 01 05 00 04 08
4b 41 4e 4e 41 4e 4b 41
```

The highlighted strings will be the delimiter between the two continuous PDU dumps. In case there is no string delimiter as above, only the first PDU will be decoded properly and the remaining PDUs will not be decoded.

3. The three icons in the debug window provide the following function:
   - Save MIB Browser Debug Results - Saves the debug information to a file
   - Print MIB Browser Debug Results - Prints the debug information to a file
   - Snmp Decoder - Switches to Decoder window.

**Invoking the Decoder Window**

1. To switch from the Debug window to the Decoder window, click on the Decoder icon.
   To switch from Decoder window to the Debug window click on the Debug icon.

2. The three icons in the debug window provide the following function:
   - Save MIB Browser Decoder Results - Saves the debug information to a file
   - Print MIB Browser Decoder Results - Prints the debug information to a file
   - Debug - Switches to Debug window

**Note:** The Save and Print options are available only when the MIB Browser is invoked as an application and not in an applet. This is because of security restrictions in case of applets.
7.4 Testing the Agent With SNMP Operations

7.4.1 SNMP GET

An SNMP GET Request can be sent to the Agent to get the values of the variable. Requests can be sent either from MIB Browser UI or from Command line. To perform the testing,

From MIB Browser UI
- Select `agentDescr` from the `agentSystem` group of `AGENT-SAMPLE-MIB`.
- Click `Operations -> GET` menu.
- The result with the default value will be as:
  Sent get request to localhost : 8001
  `agentDescr.0` -> `agentDescr` not initialized.

From Commandline
To send a GET Request from Command line, go to `<Agent Toolkit Home>/examples/snmp/low_level_udpapps` directory from the command prompt and use the following options.

```
sh run.sh snmpget -v v1 -d -c public -p 8001 localhost
.1.3.6.1.4.1.2162.4.1.1.0
```

where,
- `sh run.sh` or `run` indicates `run.sh` and `run.bat` options respectively. These options are used for running the Agent. Based on the OS, the batch or shell file is used.
- `snmpget` indicates the SNMP GET Operation to be performed.
- `-v` indicates the version in which the request is to be sent.
- `-d` debugs the snmp messages.
- `-c` indicates the community of the Agent.
- `-p` indicates the port in which the Agent is running and
- `.1.3.6.1.4.1.2162.4.1.1` indicates the OID of `agentDescr`.
- `.0` indicates the instance for the scalar variable.

Similar steps have to be followed for testing a Table in the Agent.

7.4.2 SNMP GET-NEXT

An SNMP GET-NEXT request can be sent to the Agent to acquire the next OID of the selected variable. To perform the testing on a scalar variable,

From MIB Browser UI
1. Select `agentDescr` from the `agentSystem` group of `AGENT-SAMPLE-MIB`.
2. Add the instance value `.0` to the OID in the Object ID field.
3. Click `Operations -> GET NEXT` menu.
4. The result with the default value will be as:
   Sent getnext request to localhost : 8001
   `agentSupportContact.0` -> `agentSupportContact` not initialized
From Commandline

To send a GET NEXT Request from Commandline, use the following options.

```
sh run.sh snmpgetnext -v v1 -d -c public -p 8001 localhost .1.3.6.1.4.1.2162.4.1.1.0
```

Please refer to SNMP GET for explanation of the Options used. Similar steps have to be followed for testing a Table in the Agent. Please specify the exact instance before sending a request to the Table. .0 instance is used only for Scalars.

### 7.4.3 SNMP SET

An SNMP SET request can be sent to the Agent to SET a new value to the selected variable. To perform the testing on a Scalar variable,

**From MIB Browser UI**

1. Select `agentDescr` from the `agentSystem` group of AGENT-SAMPLE-MIB.
2. Give a value, say `abc` in the SET Value text-field.
3. Click Operations -> SET menu.
4. The result with the new value will be as:
   
   Sent set request to localhost : 8001
   
   `agentDescr.0` --> `abc`

To perform the testing on a Table,

1. Select `adiskName` from the `agentDisk` group of AGENT-SAMPLE-MIB.
2. Give a value, say `abc` in the SET Value text-field.
3. Click Operations -> SET menu.
4. The result with the default value will be as:
   
   Sent set request to localhost : 8001
   
   `adiskName.1` --> `abc`

Please note that the instance value .1 has to be given in the Object ID field toward the end of the `adiskName` ID. Since the Index type of `adiskTable` is Integer32, .1 has been given as the instance value. It can be any integer value.

**From Commandline**

To send a SET Request to the scalar variable `agentDescr` from Commandline, use the following options.

```
sh run.sh snmpset -v v1 -d -c public -p 8001 localhost .1.3.6.1.4.1.2162.4.1.1.0 STRING 1
```

where,
- .1.3.6.1.4.1.2162.4.1.1.0 indicates the OID of `agentDescr`.
- .0 indicates the instance of the scalar variable.
- STRING refers to the Syntax type of the Scalar node and
- 1 refers to the new value set for `agentDescr`.

To send a SET Request to the `adiskName` of `adiskTable` from Commandline, use the following options,

```
sh run.sh snmpset -v v1 -d -c public -p 8001 localhost .1.3.6.1.4.1.2162.4.2.1.1.2.1 STRING 25
```
Here, .1.3.6.1.4.1.2162.4.2.1.1.2.1 refers to the OID of adiskName, .1 refers to the instance OID of the table, STRING refers to the Syntax type of the column, and 25 refers to the new value to be set for the column.

7.4.4 SNMP WALK

The SNMP walk operation does a continuous get-next operation. It takes the oid of the starting node, performs a get-next and if there is no error in response, repeats the get-next operation for all the nodes right from the starting node specified until it encounters any error in response i.e., an End-of-MIB view error or a change in the starting node OID.

This operation is supported through command line and is similar to GET-NEXT operations. To perform the testing on a Scalar variable, follow the steps given below:

From Commandline

To send a WALK Request to a scalar group from Command line, use the following options.

```sh
sh run.sh snmpwalk -v v1 -d -c public -p 8001 localhost .1.3.6.1.4.1.2162.4.1
```

where,

- 1.3.6.1.4.1.2162.4.1 indicates the OID of agentSystem group.

To send a WALK Request to a Tabular group from Commandline, use the following options,

```sh
sh run.sh snmpset -v v1 -d -c public -p 8001 localhost .1.3.6.1.4.1.2162.4.2
```

where,

- 1.3.6.1.4.1.2162.4.2 indicates the OID of agentDisk.

7.4.5 SNMP GET-BULK

Get Bulk operation is performed on the Agent to get the values from a group of variables. This operation is supported only on v2c and v3 Agents.

To perform the testing on a Scalar variable,

From MIB Browser UI

- Go to Edit -> Settings menu -> General Tab of MIB Browser UI.
- Change the Max-Repetitions value to 5, from 50.
- Now, select agentSystem group of AGENT-SAMPLE-MIB.
- Click Operations -> GETBULK menu.
- The result with the default values will be as:
  
  Sent getbulk request to localhost : 8001
  agentDescr.0:-->agentDescr not initialized
  agentSupportContact.0:-->agentSupportContact not initialized
  agentLocation.0:-->agentLocation not initialized
  agentAvailMemory.0:-->agentAvailMemory not initialized
  agentPlatformName.0:-->agentPlatformName not initialized

Please note that Maximum Repetitions and Non-Repeaters values need to be specified for a GET BULK operation. Based on this count, the query is done on the group. You can find these text-fields in Edit -> Settings menu -> General Tab of MIB Browser UI. GET BULK operation will not be successful if the Max-Repetitions and Time Out value do not synchronize with each other. To know more about these terms refer to the FAQs - > General section.
To perform the testing on a Table,

1. Select agentDisk group of AGENT-SAMPLE-MIB.
2. Click Operations -> GETBULK menu.
3. The result with the default value will be as:
   Sent getbulk request to localhost : 8001
   adiskID.1:-->1
   adiskName.1:-->1
   adiskCapacity.1:-->1
   adiskUsed.1:-->1
   adiskFree.1:-->1

From Command line

To send a GET BULK Request to a scalar group from Command line, use the following options.

```
sh run.sh snmpbulk -v v2 -d -c public -p 8001 localhost
.1.3.6.1.4.1.2162.4.1 0 5
```

where,
- .1.3.6.1.4.1.2162.4.1 indicates the OID of agentDescr.
- 0 indicates the non-repeaters value
- 5 indicates the Max-Repetitions value.

To send a GET BULK Request to a Tabular variable (adiskTable) from Commandline, use the following options,

```
sh run.sh snmpbulk -v v2 -d -c public -p 8001 localhost
.1.3.6.1.4.1.2162.4.2 0 5
```

7.4.6 MultiVarbind Requests

Multi-varbind operation sends many requests containing varbinds in the same PDU, i.e. the manager can query the agent with more than one request at a time. The functioning of the operation is such that when multiple requests are sent by the manager, the agent will not process each request one after another, but groups all the varbinds under a parent OID as a single request before calling the instrumentation layer. This minimizes the number of requests processed by the agent and also increases the performance of the agent as the response time is minimal.

For example, add the nodes of scalar group and nodes of tabular group in AGENT-SAMPLE-MIB to the multi-varbind text field as shown below:

```
.iso.org.dod.internet.private.enterprises.adventnet.demo.agentSystem.agentDescr.0:
.iso.org.dod.internet.private.enterprises.adventnet.demo.agentSystem.agentLocation.0:
.iso.org.dod.internet.private.enterprises.adventnet.demo.agentSystem.agentAvailMemory.0:
.iso.org.dod.internet.private.enterprises.adventnet.demo.agentDisk.adiskTable.adiskEntry.adiskName.1:
.iso.org.dod.internet.private.enterprises.adventnet.demo.agentDisk.adiskTable.adiskEntry.adiskUsed.1:
```

Here the agent processes **two requests for five varbinds** send from the manager by grouping the scalar nodes of agentSystem as a request and tabular nodes of agentDisk as another.
To enable Multivarbind operation in MIB Browser UI,

1. Select View-> Display-> Multi-Varbind from the MIB Browser UI. A dialog or text field provided for Multivarbind requests opens in UI.
2. Check Multi-Var option.
3. Select a node, say agentDescr and click Add. Similarly add other varbinds.
4. Then do a GET/SET (for a SET you should have specified a value in the SET value text-field)/ GET-NEXT/ GET-BULK operation.

If there is a read-only variable, the SET operation will not be successful. Similarly GET-BULK will not be successful if the Max-Repetitions and Time Out value do not synchronize with each other.

7.4.6.1 Atomicity or MultiVarbind SET Requests

While processing a Multivarbind SET request, if the SET fails for any subsequent varbind, the previous successful SETs in the multivarbind are rolled back to the original values. This concept is known as Atomicity (Roll Back). This feature is handled within the "processSetRequest" of the XXXRequestHandler class and the Agent API.

Enabling Rollback Method

By default, Roll Back method is enabled in an Agent. When a Multivarbind SET request arrives from the Manager the processes carried out in an Agent are,

- The Agent API first gets the existing values from the Instrument file and stores it temporarily.
- Then it processes the Multivarbind Set Request and waits for the result.
- If the Multivarbind SET request is successful, then the Agent's API discards the previous values and returns the values set as a response.
- If the Multivarbind SET request fails, say, for the third varbind in a table having 5 columns, then an appropriate error message is returned for the failure occurred and all the values are restored with the previous values.
- Also note, requests will not processed for the 4th and 5th varbind.

Disabling Rollback Method

This Rollback functionality method can also be disabled using the following piece of code in the "initSnmpExtensionNodes" method of the Generated Main File of the Agent,

```java
hdlr.setRollBackEnabled(false);
```

- By stating false, this will remove the Rollback functionality with the Agent. As stated earlier, by default this functionality will remain "true" in the Agent.
- If this functionality is disabled and the Multivarbind SET Request is successful, the Agent responds with the latest values sent by the Manager.
- In case the Multivarbind Request is unsuccessful, then in that case, the varbinds for which the request is successful, would get SET with the latest values. Whereas the other varbinds remain with the old values. Further requests will not be processed as they are unsuccessful.

7.5 Using Row Status

SNMP Operations also include Adding a Row and Deleting a Row from a Table using Row Status Textual Convention (v2c). To define Row Status, please refer to Defining a MIB section.
Different Levels in Row Status

The different levels supported by Row Status are:

- **active (1)**: The active status indicates that the conceptual row is available for use by the Managed device. A row can be added and deleted using this active status.
- **notInService (2)**: The conceptual row that exists is "Not in service" as per this range.
- **notReady (3)**: This range indicates that the conceptual row exists but is not in an usable state by the managed device.
- **createAndGo (4)**: As the name indicates, the conceptual row gets created on specifying this range from the management station and automatically becomes active, thus allowing to add any number of rows.
- **createAndWait (5)**: Using this range will create a row from the Management station but would not be accessible by the managed device. The values can be set only if the row is made "active".
- **destroy (6)**: The conceptual row can be removed from a table by specifying the respective instance and making the status as "destroy".

Adding a Row

To add a new row to a Table from the Manager that particular Table should be an SNMPv2 Table having RowStatus column i.e., the MIB should be a V2 MIB supporting Row Status. A new row can be added to the Table using any of these three ways.

- CreateAndWait (5).
- CreateAndGo using Multiple-Varbind Set.
- CreateAndGo using SNMP table UI.

**CreateAndWait**

In this method, the RowStatus column should be first SET with the value 5 (CreateAndWait). Then, the other column values should be added. Finally, the RowStatus column should be made "Active".

1. For example, in the applicationTable of Agent-Sample-MIB, the applicationTablestatus is the **Row Status** Column. To add a row (with index value 3) in this table,
2. Select applicationTablestatus and in the Object ID field add the table instance ".3" to the OID as follows
   .iso.org.dod.internet.private.enterprises.adventnet.demo.aapplications.applicationTable.applicationEntry.applicationTablestatus.3
3. and set the value with "5" (CreateAndWait) in the Set value field.
4. Similarly, select applicationName and set the value for .3 instance as "MIB Editor".
5. Select applicationVersion and set the value for .3 instance as "5.0" in the Set value field.
6. Select applicationInstallDate and set the value for .3 instance as "2002" in the Set value field.
7. Select applicationTablestatus and set the value for .3 instance as "1" (Active) in the Set value field.
8. Thus, a new row is created in the application table with the applicationID (index) value as 3.

**CreateAndGo Using MultivarbindSet**

Instead of sending multiple SET requests to the Agent, a new row can be created in a Table in a single SET method. For this, add some relevant values to the columns of a Table with the RowStatus column as "4" (CreateAndGo) using MultivarbindSET option. Check Multivarbind option available in **Edit -> Settings** menu of MIB Browser UI.

To explain it with an Example, take the applicationTable of Agent-Sample-MIB. The applicationTablestatus is the Row Status Column. To add a row (with index value 4) in this table,
1. Select `applicationTablestatus`. In the Object ID field add the instance for the Table OID as ".4". Then, set the value for .4 instance as "4" in the Set Value field.

2. In the right side bottom most part frame of MIB Browser UI, you can find the Multivar check box (after enabling it in MIB Browser settings). Check the MultiVar option and Add to list. This will add the RowStatus column with its value in the list.

3. Similarly, add the following columns to the list.

4. Select `applicationName` and in the Object ID field add the table instance ".4" to the OID. Then set the value for .4 instance as "MIB Browser". Later add it to the Multivarbind list.

5. Select `applicationVersion` and in the Object ID field add the table instance ".4" to the OID. Then set the value for .4 instance as "1.0". Later add it to the Multivarbind list.

6. Select `applicationInstallDate` and in the Object ID field add the table instance ".4" to the OID. Then set the value for .4 instance as "2002". Later add it to the Multivarbind list.

7. Now, do a single SET operation by which all the values are added to the Table. Thus a new row is created in the application table with the applicationID (index) value as 4.

CreateAndGo using SNMP Table

A new row for a table can also be created using SNMP table User Interface (UI). On enabling the SNMP Table icon after selecting the appropriate table from the left side MIB tree a dialog box opens.

- For example, in the applicationTable of the AGENT-SAMPLE-MIB, the column applicationTable status is the Row Status Column.
- To add a row in this table, select the Table and choose SNMP Table from the View menu bar or from the toolbar icon.
- Click Add button of the SNMP Table.
- The Row Addition Editor pops up. Enter values for each column and press OK. Leave the RowStatus column to be the default one (4).
- You can see the entry added to the Table by clicking the Start button.

Deleting a Row

By changing the Row Status value to 6 (Destroy) a row can be deleted from the MIB Browser UI itself. You can also delete an entry using the SNMP Table. By selecting the entry to be removed, changing the row status value to 6 and refreshing the table, the entry gets removed.
8.0 Instrumenting the Generated Code

8.1 Overview

To get the device or application specific data, a few files in the generated source files need to be instrumented to include the desired function. This will depend on how the MIB is defined, how the data associated with the MIB can be accessed, how it needs to be delivered to the Manager etc. This section clearly explains the files generated for a MIB and the steps involved in instrumenting the same for retrieving application specific details.

8.2 Generated Code Structure

On generating code for a MIB, the following files get generated by default in `<Agent Toolkit Home>/snmpprojects/projectname/agent/src/com/myCompany/myPackage` directory.

- Main file
- Instrument Files
- Entry Files
- Persistence Files and
- Other Files (Trap File and Base Table Files)

Let us see in detail how these files are organized.

8.2.1 Main File

The Agent Main file is normally generated as AdventNetSnmpAgent.java. This Main File name can be modified as per your requirement by specifying it in the text field provided for Agent Name in Project -> Settings menu bar -> Source Generation Panel -> General of MIB Compiler UI.

The Main methods are generated in the Main file and it is very important to know the code organization of this file as this would help you implement methods at a later stage for attaining specific functionality.

Composition of Main File

Any main file will be organized as given below.

- It extends `com.adventnet.snmp.snmp2.agent.SnmpAgent`
  - Since `<MainFile Name>` object extends SnmpAgent, SnmpAgent will get instantiated when `<MainFile Name>` object is instantiated.
  - On instantiating the SnmpAgent, the Agent will be started at the specified port and will wait for SNMP requests.
  - All requests sent to the Agent will reach the SnmpAgent and later given to PduRequestHandlers, if the request is valid.
- Instantiates `com.adventnet.snmp.snmp2.agent.PduRequestHandler`
  - Instantiates all the generated xxxRequestHandler's (for every scalar and table group) and registers them with the PduRequestHandler with the group/tableEntry OIDs.
The registration tree in the PduRequestHandler will be registered with OID's corresponding to the group or tableEntry.

PduRequestHandler takes care of finding out the proper listener based on the OID after getting the request from the SnmpAgent

- Instantiates all the Registration Listeners.
- Provides Agent Param Options for starting the Agent in commandline.
- Provides methods for Acl and Trap Forwarding Table Instantiation.
- Provides methods for implementing V2 Compliance.
- Provides some Variable Declarations.
- Provides code for stopping the Agent.

8.2.2 Request Handler File for a Scalar/Tabular Group

Request Handler Files get generated for every group of the MIB and these handler files takes care of forwarding the requests from the Manager to the respective OID. Thus, the basic functionality of this file is to register each module with the Agent and handle all requests from the Managers such as GET, GET-NEXT, and SET. The Request Handler files also handles row creation in case of a table. The methods get generated by default and you need not edit this file.

XXXRequestHandler.java file for a Scalar Group

- Extends SimpleRequestHandler class.
- The processGetRequest, processSetRequest, processGetNextRequest methods are generated in this class overriding the SimpleRequestHandler's methods.
- These methods will take care of processing various requests received by the Agent.

XXXRequestHandler.java file for a Tabular Group

- Similar to Scalars, this file extends BaseTableRequestHandler class.
- The processGetRequest, processSetRequest, processGetNextRequest methods are generated in this class overriding the BaseTableRequestHandler's methods.
- These methods will take care of processing the various requests received by the Agent.
- These Request Handler Files also have methods for adding row entries to a Table at agent start-up. Please refer to SNMP Table Handling to know more on the createAndAddEntry method and other methods.

8.2.3 Instrument File for a Scalar

These are the files that get created for every scalar group and implements InstrumentHandlerInterface. The file can be edited to retrieve specific or expected values, i.e. you can modify this file depending upon your requirements. Each instrument file contains a getter and setter for every scalar variable, and also a getAttributes and setAttributes method.

The getAttributes and setAttributes method will be called from their respective handlers when there is a GET, GET-NEXT, or SET requests to the scalar variables. The advantage of using these methods are

- It reduces the user implementation in every getter and setter methods generated for every scalar variable.
- It is not required to collect data from the external application every time the getter and setter request is processed. The getAttributes and the setAttributes method contact the external application once, to retrieve the data for all the scalar variables.
- The methods are used for grouping the attributes of a scalar group while performing multi-varbind requests.
8.2.4 Entry File for a Table

These files get generated for tables and are similar to Instrument files generated for scalar variables, but extends BaseTableEntry class. They contain getters and setters for each variable in the table and also getAttributes and setAttributes method. These entry files are stub files to perform user implementation and pertains to the same advantages mentioned in the Instrument file.

8.2.5 Persistence Files

xxxFileToVector.java file is generated when the storage option chosen during generation is Text file. This file implements com.adventnet.snmp.snmp2.agent.UpdateListener and has methods such as readFromFile() and writeToFile() for reading and writing into the text file. It is generated to make it simple for the users and to make the data available to the Agent in a flat file. You can also disable the option for generating "xxxFileToVector" by choosing RAM storage option in the MIB Compiler UI. This does not store the values anywhere except in runtime memory. Alternatively, the xxxXMLToVector.java file is generated when the XML storage option is chosen before generation.

8.2.6 Notification File

<MainFileName>Trap.java: This file has required methods for sending traps/notifications for the objects, which are grouped under Notification-Type/Trap-Type macros in the MIB module.

8.3 Instrumenting the Generated Code

Editing a file for a desired function (Instrumentation) needs to be done based on the application and the requirement of the user. This section has been explained assuming that the code generated for RFC1213 MIB is instrumented for retrieving specific values from the "sysDescr" scalar node of systemGroup and "ifDescr" column of ifTable group. Assume that the Agent is a v2c Agent.

Editing Files Generated for Scalars

The getter method of the scalar variable sysDescr in the System group of RFC 1213 MIB would be generated as follows in the SystemInstrument.java

```java
/*
* Handles the SNMP Get Request for sysDescr
*/
public String getSysDescr() throws AgentException {
    // Fill up the stub with required processing
    return sysDescr;
}
```

On sending a GET request, the value of the sysDescr scalar variable is retrieved.

You can fill in your own stubs here. The instrumentation code has to be filled between these tags in order to support code merging when regenerated.

```java
/* User code starts here */
/* User code ends here */
```

These are the custom tags which can be used anywhere in the generated code. You just have to include your code in the file within these tags. When code is regenerated, the MIB Compiler looks for these tags and preserves the changes in the newly generated code. If either of the tags is not given, code merging will not be proper. Please refer to the topic "Code Merging" for more information.
Similarly, the setter method of the scalar variable sysDescr would be generated as follows.

```java
/**
 * Handles the SNMP Set Request for sysDescr
 * @param value - The String value to be set
 */
public synchronized void setSysDescr(String value)
{
    // Fill up the stub with required processing
    sysDescr = value;
}
```

On sending a SET request to this scalar object with a new value, the new value is set to the OID. You can also edit these files to fill up your required processing.

**Editing Files Generated for Tables**

The getter methods of the table variable "ifDescr" in the ifTable group would be generated as follows in the `ifEntry.java`

```java
/**
 * Handles the SNMP Get Request for ifDescr
 */
public String getIfDescr()
throws AgentException {
    // fill up with necessary processing
    return ifDescr;
}
```

On doing a GET request for this column with the instance value would retrieve the value of that columnar variable.

Similarly, the setter methods of the table variable "ifDescr" would be generated as follows. On sending a SET request to this variable the value will be SET.

```java
/**
 * Handles the SNMP Set Request for ifDescr
 * @param value - The String value to be set
 */
public synchronized void setIfDescr(String value)
{
    // Fill up with necessary processing
    ifDescr = value;
}
```

Edit these files to include the required values.

**Code Merging**

Code merging, is an option provided for users who prefer to shift between releases. It is mainly used for migration purposes. Say, if a developer using 5.0 version of Agent Toolkit decides to migrate to the 5.1 version, then he can make use of this Merging option wherein the manually added code (using the tags) is merged with the code, present in the file generated by the new version of the toolkit, to attain the functions available in the old version. If this option is not enabled before regenerating a file, all the code added by the user would be lost.
The user code is normally added to the generated files using the following tags.

```java
// User code starts here
// Add your code here....
// User code ends here
OR

/* User code starts here */
Add your code here
/* User code ends here */
OR

// AdventNet code ends here
// Your code can be added here
// AdventNet code starts here
```

These are the custom tags which can be used anywhere in the generated code. The user has to just include his code in the file with these tags. When regenerated, the Mib Compiler looks for these tags and preserves the changes in the newly generated code. If either of the tags are not given merging will not be proper. Please note that the last tags are also supported and when these get generated in certain files, the user has to just include the codes between these tags which is generated, by default. Thus code merging is supported on all generated java files.

**Enabling Code Merging Option in the Mib Compiler UI**

This feature is enabled by default in the tool. It can be enabled/disabled in the Mib Compiler UI as follows:

1. Click **Project->Settings->Source Generation->General**
2. Enable/Disable **Merge with previous stub files** in the General panel of Source Code Generation
9.0 SNMP Table Handling in Generated Code

9.1 Overview
This section describes the code generated for tables using different storage models. Firstly, it is required to know how the data of a table are stored in the agent.

9.2 Storing the Tabular Data
The data given in tables can be stored in two formats, either in Agent Table Model or in Your Own Storage Model.

9.2.1 Agent Table Model
Agent Table Model is the default storage model provided by Agent Toolkit. As per this storage model,
- For every row in the table, an xxxEntry object gets created.
- The xxxEntry object stores the values of all the columns of that single row.
- This xxxEntry object has getter and setter methods for getting and setting the value of a particular column.
- It also stores the instance OID of the row as an int array. Instance OID is nothing but the index value of a table that identifies a particular row.
- All the xxxEntry objects are then put into an AgentTableModel.
- The ’AgentTableModel’ internally stores all the xxxEntry objects in a vector sorted by their instance.

The figure given below illustrates how the data are stored in AgentTableModel.

9.2.1.1 Methods Generated for Agent Table Model
The Request Handler file has the following lines generated for a table.

For Getting an xxxEntry from Agent Table Model
The following method is called. This would return the xxxEntry corresponding to the specified instance.

```java
tModelComplete.get(inst);
```

**For Getting the First Entry from Agent Table Model**

The following method is called. This would return the xxxEntry corresponding to the first row (instance oid) in the table.

```java
tModelComplete.getFirstEntry();
```

**For Getting the Next xxxEntry from Agent Table Model**

To get the xxxEntry corresponding to the instance oid immediately after the specified instance oid (inst), the following method is called.

```java
tModelComplete.getNext(inst);
```

**For Creating a New Entry in the Agent Table Model**

For creating a new entry in the Agent Table Model,

- A new instance of the xxxEntry object is to be created.
- The instance OID (i.e., the index in dotted OID format) is set in the xxxEntry object by calling `xxxEntry.setInstanceOid(instance);`
- The values of other columns are also set in the xxxEntry object by calling the respective setter methods.
- The above created xxxEntry object is added to the AgentTableModel.

The 4 steps given above are repeated for all the rows in the Table.

**For Adding the Row to the Agent Table Model**

A row is added to the Agent Table Model by calling this method.

```java
tModelComplete.addRow(entry);
```

### 9.2.2 User Storage Model

Sometimes, you may prefer to have other storage mechanisms than using the default Agent Table Model. Agent Toolkit also provides an option to implement your own storage model. To achieve this, storage model for tables can be selected as "User Storage".

Follow the given steps to select 'User Storage' for tables from the MIB Compiler UI while generating the agent code,

1. Select `Project->Settings` from menu bar. The Settings dialog will pop up.
2. Select `Source Generation-> Storage Model` node from the tree in the left frame of the Settings dialog.
3. To select User Storage Model for tables, select 'User Storage' option from the combo box.
4. To select the option for individual tables, select the corresponding table from the list in right panel and choose 'User Storage' option from the combo-box provided.
5. Generate the source.

This generates a XXXTableModelListener class for each table selected using the User Storage option. This class implements the `com.adventnet.utils.agent.TableModelListenerExt`
interface. All the methods that you want to instrument will be listed in this generated file. You can plug-in your implementation here.

If 'User Storage' is used for tables, then in the generated agent code (like XXXRequestHandler class), XXXTableModelListener instance will be used instead of the Agent Table Model instance.

```java
// To return the entry for the given instance oid
public com.adventnet.utils.agent.TableEntry get(int[] inst);
This method gets called when there is a GET request for the table. The request is sent with the instance oid and on identifying the instance, the value is returned for that entry. If no such instance exists or for an invalid entry, a NULL value is returned.

// To return the next entry for the given instance oid
public com.adventnet.utils.agent.TableEntry getNext(int[] inst);
This method gets called when there is a get next request for an instance oid value. In that case the agent searches for the next immediate instance and returns the value of that instance. A NULL value is returned if no such instance exists.

// To return the first entry
public com.adventnet.utils.agent.TableEntry getFirstEntry();
This method gets called when there is a getnext request for the table without any instance oid.

// This method will be called when any entry is to be added
public void addRow(com.adventnet.utils.agent.TableEntry entry) throws Exception;
This method gets called when there is a SET request that adds a new row to the table, which has the row status.

// This method will be called when any entry is to be deleted
public void deleteRow(int[] inst) throws Exception;
This method gets called when there is a SET request to delete a particular row in the table that is already existing in the table.

// To set the vector of TableEntry objects
public void setTableElements(Vector tableElements);
This method gets called when you want to set the table vector elements.

// To get the vector of TableEntry objects
public Vector getTableElements();
This method gets called when you want to get the vector entry objects.

// To modify the rows in the table
public void modifyRow(String instance, Hashtable list, Vector indexList)
This method gets called during the set request for the table. If the row denoted by 'instance' string is already present in the table, that row should be modified using the incoming values. If the row does'nt exist, new row should be created using the incoming arguments.
```
9.3 Initializing Tables Before Agent Start-Up

Any Table is initialized with the default values provided on Agent start-up. The default values for a Table do not get generated unless the "Initialize variables with Default Values" is checked in the MIB Compiler UI. (Project -> Settings menu -> General Category).

Instead of having the default values, if you prefer your Agent to be started with a Table holding the values you require, the following methods can be utilized. But please note when you instrument the code for the Tables to have specific values, the Agent should be restarted for the changes to take effect.

9.3.1 createAndAddEntry Method

To create a row in the Table, the following methods can be used.

- createAndAddXXXEntry or
- createAndAddNewXXXEntry

createAndAddXXXEntry

Using this method will add an entry to the Table with the Instance. The method takes all the column values and its instances as inputs and creates the row. The method name and the parameters used will differ according to the Tables used.

For example: The code for creating and adding a new entry in the AapplicationTable of AGENT-SAMPLE-MIB is given below.

```java
applicationTableListener.createAndAddApplicationEntry(new String("1.1"), new Integer(1), "agenttoolkit", "5.0", "6/10/2002", new Integer(1));
applicationTableListener.createAndAddApplicationEntry(new String("1.2"), new Integer(2), "windows", "2000", "5/5/2000", new Integer(1));
```

This code should be included in the initSnmpExtension Nodes generated in the Main file.

createAndAddNewXXXEntry

This method is used to create a new entry to the table without specifying the instance. It accepts all the column values as inputs. To create an instance the createInstance method is called. When the instance value is created and the values for the columns exist, the createAndAddEntry method is called. This creates a new entry in the Table.

For example: The code for creating and adding a new entry in the AapplicationTable of AGENT-SAMPLE-MIB is given below.

```java
applicationTableListener.createAndAddNewApplicationEntry(new Integer(1), "agenttoolkit", "5.0", "6/10/2002", new Integer(1));
applicationTableListener.createAndAddNewApplicationEntry(new Integer(2), "windows", "2000", "5/5/2000", new Integer(1));
```

This code should be included in the initSnmpExtension Nodes generated in the Main file.

createInstanceForEntry

This method is used for creating an instance for a Table by taking the index values as inputs. Once the instance is defined, they are given as inputs along with the columnar values and used for creating an entry.
For example: The code for creating an instance in the AapplicationTable of AGENT-SAMPLE-MIB is given below. This method gets generated by default in the AapplicationTableRequestHandler file.

```java
/* This Method is for creating the Instance OID for the AapplicationTable. */
public String createInstanceForApplicationEntry(Integer valueForApplicationID) {
    int[] inst = null;
    int[] ins = null;
    int insInt = valueForApplicationID.intValue();
    ins = new int[] {insInt};
    inst = utils.addIntArrays(inst, ins);
    return utils.intArrayToString((int[])inst);
}
```

### 9.3.2 setTableVector Method

The setTableVector(Vector) method in xxxRequestHandler enables you to set your own Vector to the Agent. But make sure of the following:

- The Vector contains elements of xxxEntry type.
- The INSTANCE should be SET using setInstanceOID() in the entries (xxxEntry) in the Vector e.g.,

```java
Vector aapVec = new Vector();
//Here the entry is created newly.
AaplicationEntry aapEntry1 = new AaplicationEntry();
try{
    aapEntry1.setInstanceOID(new int[]{1});
    aapEntry1.setAaplicationID(new Integer(1));
    aapEntry1.setAaplicationName("agenttoolkit");
    aapEntry1.setAaplicationVersion("5.0");
    aapEntry1.setAaplicationInstallDate("6/10/2002");
    aapEntry1.setAaplictionTablestatus(new Integer(1));
}catch(Exception e){
    e.printStackTrace();
}
aapVec.addElement(aapEntry1);

//Here the entry is got from the getAaplicationEntryInstance which will be generated
//in the Agent Main File.
AaplicationEntry aapEntry2 = getAaplicationEntryInstance();
try{
    aapEntry2.setInstanceOID(new int[]{2});
    aapEntry2.setAaplicationID(new Integer(2));
    aapEntry2.setAaplicationName("windows");
    aapEntry2.setAaplicationVersion("2000professional");
    aapEntry2.setAaplicationInstallDate("5/5/2000");
    aapEntry2.setAaplictionTablestatus(new Integer(1));
} catch(Exception e1){
    e1.printStackTrace();
}
aapVec.addElement(aapEntry2);
applicationTableListener.setTableVector(aapVec);
```

These codes have to be added in the initSnmpExtensionNodes method of the Agent Main file.
9.4 Index Handling

Any Table should have an Index column defined in it. Each row of a Table is identified by its index value. There are various types of Indexing and this topic provides the details of the supported syntax types in indexes and the types of Indexing.

9.4.1 Supported Syntax Types in Index

An Index in a Table can have only the following types of Syntax :-

- INTEGER
- OCTET STRING
- DISPLAY STRING
- UNSIGNED32
- OBJECT IDENTIFIER
- TIMETICKS
- IPADDRESS
- NETWORKADDRESS
- OPAQUE.
- COUNTER32

9.4.2 Types of Indexing

An Index of a Table may belong to the category of : Single Index, Multiple Index, Implied Index or External Index. To know what exactly each index means and how they can be defined in the MIB, please refer to Adding a Table topic in Defining a MB section.

9.4.2.1 Single Index

For a Table having single index column, the code gets generated as given below. Assume that the code is generated for ifTable of RFC1213 MIB in requestProcessing Methods. ifTable maintains a single index column namely, ifIndex whose syntax type is Integer,

```java
int [] inst = AgentUtil.getInstance oid,ifTableOidRep.length + 1);
entry = (IfEntry) tModelComplete.get (inst);
```

For any Table, the resolve Index method is called. All the SET requests for the Table reaches the Resolve Index method wherein the requests from the Manager are first checked whether such an Instance exists. If a matching instance does not exist then an error is thrown or a new entry is created. The code for resolving index resolves the value of the indices from the instance oid given.

9.4.2.2 Multiple Index

When it comes to a Table having Multiple Index, the code is generated in the same manner. The only change in Multiple Index is that while resolving the Index, the instance and its syntax type are verified.

9.4.2.3 External Index

When a Table has an external index column, then the request for that particular Table should first check if such an index column exists in the Base Table. The presence of external index value is checked by the code generated as follows in the respective Table's Request Handler File, (say for a Table having its first index syntax as Integer)

```java
if(!agentName.checkExternalAaPPlicationID((Integer
```
In this code, the agentName is the reference of the SNMP Agent Main file generated for the MIB and the indexVar[1] is the value of the external index from the request Varbind.

The checking for the value present in the External Table is done using the checkExternalXXX method in the Agent Main file as,

```java
public boolean checkExternalXXX(Integer value){
    Vector vec = XXXTableListener.getTableVector();
    return checkExternalIndex(vec, value);
}
```

When the entry reaches the Agent with the instance oid value, the resolve index method gets called in the XXXRequestHandler file. The method for the same is,

```java
// Resolving the Index
indexVar = AgentUtil.resolveIndex(inst,type);
if(indexVar == null){
    AgentUtil.throwWrongValue(pe.getVersion());
}
int req = node.getSubId();
switch(req){
```

Based on the instance type, the index is resolved. If the index value is null or if no such instance exists, an error is thrown after resolving the index.

9.4.2.4 Implied Index

A Table having an Implied index column is similar to a Table having Multiple index columns. The code gets generated similar to other Tables. But while resolving the Index, the API call that gets called is,

```java
resolveIndex(int[] inst, byte[] type, boolean[] isImplied)
```

If isImplied is set to "false", then the Index is resolved taking into consideration the length of the instance. If set to "true", then it is handled in the manner as any Table that has multiple Index columns is handled.

Example

For better understanding of all indexes please go through the example available in `<Agent Toolkit Home>/examples/snmp/tablehandling/tables` directory. The information in readme.html will help you run the example.
10.0 Authenticating SNMP v1/v2c Requests

10.1 Overview

An SNMP Agent exposes critical information about the device or application being managed through its management interface. To ensure some security to the system, a check has to be kept on the people trying to access such information and capable of performing operations on it. This Authentication mechanism aids us in restricting the user and the kind of information he can avail, thus ensuring security from trespassers who are denied admission.

The AdventNet Java Agent supports Community-based Authentication for SNMPv1 and SNMPv2c requests. On receiving a SNMPv1 or SNMPv2c request from the Manager, the Agent will check for Authentication with the received Community string and the received SNMP Request type (GET, GET-NEXT, SET etc.). Later processes the request.

Apart from Authenticating v1/v2c requests, Java Agent also supports View-based access. This facility is provided only when vACL is enabled. Please refer to Enabling Authorization using VACM for more details.

10.2 Authentication Support in the Agent

By default v1/v2c Agents have support for Authentication. The code generated in the Main file implementing Authentication Support is given below:

```java
// Acl Table Support
private com.adventnet.snmp.snmp2.agent.AclTable aclTable = null;
private com.adventnet.snmp.snmp2.agent.AclTableRequestHandler acl = null;

The code for aclTable instantiation under the initSnmpExtensionNodes are:
aclTable = new com.adventnet.snmp.snmp2.agent.AclTable((SnmpAgent)this, "AccessControlTable.xml", "xml");
```

10.3 Community Details for Authentication

An Agent authenticates a request based on the Community. Hence it is required to store the community details and the details of the Manager given access for that particular Community in the Agent side. To store these details you can make use of the aclTable present under AGENT-SNMP-CONFIG-MIB -> agentConfiguration group -> v1v2AuthenticationTables. This aclTable (.1.3.6.1.4.1.2162.10.3.1.2.1) maintains the set of authentication parameters given below:

- aclCommunity& - the Community used by the Manager to communicate with the Agent.
- aclAccess - the maximum access for the Community being either of these: No Access (0), Read_Only (1), Write_Only (2), Read_Write (3)
- **aclManager** - IP address of the Managers who are allowed specified access for the specified community. The default value is '0:0:0:0' which states access is provided to all Managers for the corresponding community.

- **aclStatus** - the Row Status column.

### 10.4 Configuring Managers to a Community

You can also include new communities and specify the Managers to be given access for the community. There are two ways to add communities to the Managers in the Authentication Table. It can be done either: (1) Before Agent start-up or (2) During Run time.

#### 10.4.1 Before Agent Start-Up

Entries can be added to the Authentication Table before Agent start-up either using MIB compiler UI or using Text/XML file or using API Calls. To specify the entries before Agent start-up,

**Using MIB Compiler UI**

1. Create a Project and load a MIB.
2. Choose Project ->Settings menu from the menu bar of MIB Compiler UI.
3. Select aclTable in the v1v2Authentication Panel.
4. Now, Click Add.
5. A wizard pops up wherein you can specify the community for a Manager entry.
6. Click OK.

**Text File / XML File / RAM (Runtime Memory)**

The entries configured through MibCompiler UI get stored in the configuration file, AccessControlTable.xml or AccessControlTable.txt under `<Agent Toolkit Home>/snmpprojects/projectname/agent/bin/conf` directory, provided the storage type is chosen. For this purpose,

1. Choose Project -> Settings menu of MIB Compiler UI.
2. Select aclTable from the v1v2Authentication Panel.
3. Choose XML File or Text File from the Storage Type Option. By default XML File is chosen.

This file has to be edited for adding Manager Entries. The `AccessControlTable.xml` given below has been edited for adding a New Manager with Community "xxx" and IP Address as "1.192.68.200".

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Table>
  <row>
    <column name="aclCommunity" value="public" />
    <column name="aclAccess" value="3" />
    <column name="aclManager" value="0:0:0:0" />
    <column name="aclStatus" value="1" />
  </row>
  <row>
    <column name="aclCommunity" value="xxx" />
    <column name="aclAccess" value="3" />
    <column name="aclManager" value="1.192.68.200" />
    <column name="aclStatus" value="1" />
  </row>
</Table>
```
Please note that the Agent has to be re-started for the changes to take effect. It is also possible to access the new entries without restarting the Agent. Make use of the following method in the Main file for the Agent to read the entries directly from the text file.

```
    acl.setAutoRefresh(boolean flag, int autoRefreshTimeInt)
```

This method has to be added after the code for aclTable Request Handler Instantiation.

**Using RAM option (Run Time Memory)**

Run time memory can be used to store the Manager information in the Agent Memory itself. Using this option will not store the entries in text files or in XML files. To use the run time memory for adding Manager entries dynamically,

1. Choose **Project -> Settings** menu from the menu bar of MIB Compiler UI.
2. Select **aclTable** in the **V1V2Authentication** Panel.
3. Select **Runtime Memory** from Storage Type combo box.

After choosing the storage option, follow the steps given in adding entries "From the Manager" given under the heading "Adding Entries During Runtime"(10.4.2). Please note that once the Agent is killed, the entries added are removed from the memory.

**Using API calls**

If the RAM storage type option is selected for aclTable, then the code gets generated in the Main file as given below. Adding the example code highlighted would authenticate the request from the Manager IP Address "127.0.0.1" with community "xxx" and also provide READ_WRITE Access to the same.

```
    aclTable = new com.adventnet.snmp.snmp2.agent.AclTable((SnmpAgent)this);
    aclTable.addAclEntry(com.adventnet.snmp.snmp2.agent.AclTableRequestHandler.createAclEntry("public", 3, "0:0:0:0"));
    aclTable.addAclEntry(com.adventnet.snmp.snmp2.agent.AclTableRequestHandler.createAclEntry("private", 1, "0:0:0:0"));
    aclTable.addAclEntry(com.adventnet.snmp.snmp2.agent.AclTableRequestHandler.createAclEntry("xxx", 3, "127.0.0.1"));
```

**10.4.2 During Run Time**

Communities can be added to the aclTable dynamically during run time.

**From the Manager**

To add an entry to the aclTable from the Manager,

- Load the AGENT-SNMP-CONFIG-MIB in MIB Browser.
- Select aclTable from the v1v2AuthenticationTables module of agentConfiguration group.
- By selecting the respective table and clicking **SNMP Table icon** in MIB Browser will open up a wizard wherein entries can be added to the aclTable.
- The entries added from remote get updated in the text/XML file.
- Please note that it is possible to access the table from remote only if "Remote Configuration" option is enabled in the aclTable in **v1v2Authentication Panel** of **Project -> Settings** menu in MIB Compiler UI.
You can also enable "Remote Configuration" using the following API call in case the UI option is not chosen before generation.

```java
// For Acl Table Remote Configuration
acl = new com.adventnet.snmp.snmp2.agent.AclTableRequestHandler((SnmpAgent)this, aclTable);
ACL.addRegistrationListener(hdlr);
```

This method has to be included in the initSnmpExtensionNodes of the Main File.

### 10.5 If Authentication Fails

You have seen how a Request is authenticated and how to add a community to the Agent. Let us see the consequences of a community not being authenticated.

If the received Community does not exist then the received message will be dropped and also an "AuthenticationFailure" Trap will be generated by the Agent and sent across to all the registered Managers provided the "snmpEnableAuthenTraps" flag of the SNMPv2 MIB `snmpGroupCounters` is set. By default the Failure Traps will be generated on a community not being authenticated.

You can also disable the facility of receiving Authentication Failure Traps. To do the same,

- Load SNMPv2 MIB in the MIB Browser and
- Change the value of `snmpEnableAuthenTraps` scalar variable in the `snmp` module to 2(false) from 1(true).
- Now the Agent will not generate any Traps for this purpose.

In case the Community exists but does not match with the aclAccess and Request Type, then a noAccess Error or noSuchName error is thrown depending on the version of the Agent.

### 10.6 Implementing Your Own Authentication

You can also define your own Access Control Table instead of using the default Access Control Table in `AGENT-SNMP-CONFIG-MIB`. For this purpose:

1. Define a similar aclTable in your MIB using the OBJECT TYPE TABLE construct in MIB Editor. Please refer to "Adding a Table" in Creating a MIB section for using the construct.
2. The OID structure of the newly defined table should be same as the aclTable in `AGENT-SNMP-CONFIG-MIB.txt` after the enterprise level as .1.3.6.1.4.1.your enterpriseOID.
   .1.3.6.1.4.1.2345.10.3.1.2.1 is a sample entry oid.
3. Add the following piece of code above the code for restartSnmpAgent in the Main file.

```java
setOidRep(int[] aclTableOidRep) OR
setRegisteredOid(java.lang.String oid)
```
4. Now, compile the code and query the Agent with your defined community.
5. Request will be authenticated by the Agent and response can be received.

### 10.7 Running the Example

To know more about Authentication, please go through the readme.html of the example available in `<Agent Toolkit Home>/examples/snmp/aclandvacl` directory.
11.0 Enabling Authorization in SNMP v1/v2c using VACL

11.1 Overview

For security reasons, it is valuable to restrict the access rights of some groups to only a subset of the Management information in the Management domain. To provide this capability, access to a community is via a "MIB view" which details a specific set of managed object types within that community.

For example, for a given community, there will be one MIB view which provides access to all management information in that community, and often there will be other MIB views each of which contains some subset of the information.

So, the access allowed for a group can be restricted in the desired manner by specifying its rights, in terms of the particular (subset) MIB view it can access.

By implementing the View-based access feature, this requirement can be achieved.

Note: View-based access Control for v1/v2c Agents is given based on the Community specified in the aclTable.

11.2 View-based Access Support in the Agent

By default, v1/v2c Agents do not support View-based Access feature.

The code that gets generated in the Main file on implementing View-based. Access is given below:

```java
// Vacl Support
private com.adventnet.snmp.snmp2.agent.VaclTable vaclTable = null;
private com.adventnet.snmp.snmp2.agent.VaclTableRequestHandler vacl = null;
Code generated for View-based Access control implementation:
vacl = new com.adventnet.snmp.snmp2.agent.VaclTable((SnmpAgent)this, "ViewAccessControlTable.xml, "xml");
```

11.3 Details of MIB Views to Managers

AdventNet Java Agent provides vaclTable (.1.3.6.1.4.1.2162.10.3.1.2.1.2) to store the information of Mib views and Community in the Agent side. The vaclTable present under AGENT-SNMP-CONFIG-MIB -> agentConfigurationGroup -> v1v2Authentication module has the following columns defined in it:

- **aclCommunity (external index)** - This column present in the aclTable acts as an external index column for the vaclTable. Any Manager entry is given MIB View Access, based on the community specified in the aclTable.
- **vaclmibViews** - The scalar or table OID for which the particular community (specified earlier) is given view access
- **vaclviewStatus** - The RowStatus column with which rows or columns are added to the table.
11.4 Adding MIB Views for Communities

Mib Views for a specified community can be specified in the vaclTable either: (1) Before Agent Start-Up or (2) During Run time.

11.4.1 Before Agent Start-Up

To add Mib Views to a Community before Agent start-up,

Using MIB Compiler UI

- Select Project-> Settings menu from the MIB Compiler UI.
- Select V1V2Authentication Panel.
- Select vaclTable.
- Click Add to add entries to the wizard.
- Please note that the aclCommunity column in aclTable should have some entries before adding entries in vaclTable.

Using Text File / XML File / Runtime Memory

The entries configured using MIB Compiler UI gets stored in the configuration file, ViewAccessControlTable.xml or ViewAccessControlTable.txt gets under <Agent Toolkit Home>/snmpprojects/projectname/agent/bin/conf directory. To choose the type of storage,

- Select Project->Settings menu from the MIB Compiler UI.
- Select V1V2Authentication Panel.
- Choose vaclTable.
- From the Storage Type combo box, select XML File or Text File or Runtime Memory as you prefer. By default XML File is chosen.

This file has to be edited for adding View Access to communities. The ViewAccessControlTable.xml given below has been edited for adding a new entry with Community "private" and mibViews ".1.3.6.1.4.1."

```xml
<Table>
  <row>
    <column name="aclCommunity" value="public"> </column>
    <column name="vaclmibViews" value=".1.3.6"> </column>
    <column name="vaclviewStatus" value="1"> </column>
  </row>
  <row>
    <column name="aclCommunity" value="private"> </column>
    <column name="vaclmibViews" value=".1.3.6.1.4.1"> </column>
    <column name="vaclviewStatus" value="1"> </column>
  </row>
</Table>
```

Please note that the Agent has to be re-started for the changes to take effect.

Run time memory can be used to store the Manager information in the Agent Memory. Using this option will not store the entries in text files or in xml files. After choosing the storage type, follow the steps given in adding entries "From the Manager" i.e During Runtime (11.4.2). Please note that once the Agent is killed, the entries added are removed from the memory.
Using API Calls

Adding the code highlighted below in the Main file generated provides view access to the community xxx from the OID .1.3.6.1.4.1.

```java
vaclTable = new com.adventnet.snmp.snmp2.agent.VaclTable((SnmpAgent)this);
vaclTable.addVaclEntry(com.adventnet.snmp.snmp2.agent.VaclTableRequestHandler.
("public", new String[]{".1.3.6"});
vaclTable.addVaclEntry(com.adventnet.snmp.snmp2.agent.VaclTableRequestHandler.
("xxx", new String[]{".1.3.6.1.4.1"}));
```

11.4.2 During Run Time

From the Manager

To add an entry to the vaclTable from the Manager,
1. Load the AGENT-SNMP-CONFIG-MIB in MIB Browser.
2. Select vaclTable from the v1v2AuthenticationTables module of agentConfiguration group.
3. By selecting the respective table and clicking SNMP Table icon in MIB Browser will open up a wizard wherein entries can be added to the vaclTable.
4. The entries added from remote get updated in the text/xml file.
5. Please note that it is possible to access the table from remote only if "Remote Configuration" is enabled in the v1v2Authentication Panel of Project -> Settings menu in MIB Compiler UI.
6. You can also enable "Remote Configuration" option using the following API call if it is not chosen before generation.

```java
// For Vacl Table Remote Configuration
vacl = new
com.adventnet.snmp.snmp2.agent.VaclTableRequestHandler((SnmpAgent)this, vaclTable);
vacl.addRegistrationListener(hdlr);
```

This method has to be included in the initSnmpExtensionNodes method.

11.5 Implementing Your Own View Access

You can also define your own View Access to Managers instead of using the default View Based Access Control Table in AGENT-SNMP-CONFIG-MIB. For this purpose:

1. Define a similar vaclTable in your MIB using the OBJECT TYPE TABLE construct in MIB Editor. Please refer to "Adding a Table" in Creating a MIB section for using the construct.
2. The OID structure of the newly defined table should be same as the vaclTable in AGENT-SNMP-CONFIG-MIB.txt after the enterprise level as .1.3.6.1.4.1.your enterpriseOID. .1.3.6.1.4.1.2345.10.3.1.2.2 is a sample entry oid.
3. Add the following piece of code above the code for restartSnmpAgent in the Main file.

```java
setRegisteredOid(java.lang.String oid)
```

5. Now, compile the code and query the Agent with your defined community and check if it has view access.
6. You will be able to receive the response.
12.0 Sending Traps and Informs

12.1 Overview
In addition to retrieving data from the Managed Resource and sending response to them, Agents have the ability to send unsolicited messages to Managers when they detect some significant event. An unsolicited message of this sort is called **Trap** (in SNMPv1) or **Notification** (in SNMPv2 and SNMPv3). The AdventNet Java Agent supports both Traps and Notifications in all the three versions of SNMP. Informs, on the other hand, are Notifications sent from the Agent, with a request to the Manager to acknowledge the receipt of such a Notification.

12.2 Generating Traps
For the SNMP Agent to generate a Trap/Notification, the respective MIB should have definition for that Trap/Notification. To define a **Trap Type Construct** (in SNMPv1 MIB Module) or **Notification Type Construct** (in SNMPv2c Module) using MIB Editor, please refer to the 4.6 Adding a Trap/Notification section.

12.2.1 Method Generated for Traps
For each Trap/Notification type construct in the MIB modules of the agent, the following method will be generated in the file `<MainFileName>Trap.java`:

```java
public boolean send<Notification Type/Trap Type name>(HashtablenotifHash,
            Hashtable instanceHash, Vector extraVarbinds) {

    return sendNotification(".1.3.6.1.4.1.2162.4.7", 1, notifHash, instanceHash,
                        extraVarbinds);
}
```

Here the user is required to provide the inputs to the above methods. The required inputs are as follows:

- **notifHash**: Here a Hashtable having Table/Scalar GroupName is provided as key and a Vector having list of objects under this group as element.
- **instanceHash**: A Hashtable having TableName is provided as key and instance (as int array) as element. For sending traps/notifications for Table objects, instance (as int array) is required. If instance part is not provided the objects are considered as scalar.
• **extraVarbinds**: List of SnmpVarBind objects. In case, the user need to send some extra variable bindings with this trap/notification, this option can be used.

With the above inputs, the method sendNotification() is called internally to send the trap/notification to the manager.

```
Note: The names provided for the managed objects in notifHash and instanceHash should be exactly the same as that of the MIB module used for agent development.
```

### 12.2.2 Generating Trap on SET

It is possible to generate traps while setting a value to a variable for which trap is defined. The trap generation method is invoked on SNMP SET and is called from the respective xxxRequestHandler.java file. By default, this option is enabled. To view this,

- Choose **Project -> Settings** menu from MIB Compiler UI.
- Select **General** from the **Source Generation** panel.
- You can find the **Generate Trap on Set** option checked.

### 12.3 Manager Details for Sending Traps

To notify the manager about the significant state of change in the agent, it is required to store the manager information in the agent side. To maintain the manager's information, we make use of the Trap Forwarding Tables of AGENT-SNMP-CONFIG-MIB.

Before moving into the Trap Forwarding Table, let us see a brief on IPv6 implementation in SNMP agent.

IPv4 is the generally used addressing model. Agent Toolkit introduces IPv6 (IP Version 6 protocol) model, which is a completely new and redesigned network-layer protocol defining an advanced addressing format of IPv4. IPv6 is a 128-bit identifier, addressing interfaces and set of interfaces. Agent Toolkit SNMP agent also provides the IPv6 addressing support apart from IPv4 for any network communication between the agent and the manager. Accordingly, the changes have been implemented in the trap forwarding table.

```
Note: When the agent needs to support IPv6 addressing model, it must run on JDK1.4.1 and later.
```

#### 12.3.1 v1v2TrapForwardingTable

The v1v2TrapForwarding Table with the OID .1.3.6.1.4.1.2162.10.3.1.2.2.1 has the following columns defined in it for storing v1/v2c Manager's information:

- **v1v2ManagerHostType** (Index Column): The type of addressing model to represent the network address (IPv4/IPv6).
- **v1v2ManagerHost** (Index Column): The network address of the Manager to which the Trap should be sent.
- **v1v2ManagerPort** (Index Column): The Port Number of the Manager in which Traps will be listened.
- **v1v2ManagerVersion**: SNMP Version of the Trap sent.
- **v1v2ManagerCommunity**: Community in the Trap.
- **v1v2ManagerTimeOut**: Time Out Value.
- **v1v2ManagerRetries**: Retries Count.
- **v1v2ManagerStatus**: Status of the Manager Entry.
12.3.2 v3TrapForwardingTable

The v3TrapForwardingTable with the OID .1.3.6.1.4.1.2162.10.3.1.2.2.2 has the following columns defined in it for storing v3 Manager's information:

- **v3ManagerHostType**: The type of addressing model to represent the IP Address (IPv4/IPv6).
- **v3ManagerHost (Index Column)**: The IP Address of the Manager to which the Trap should be sent.
- **v3ManagerPort (Index Column)**: The Port Number of the Manager in which Traps will be listened.
- **v3ManagerVersion**: SNMP Version of the Trap.
- **v3ManagerCommunity**: Community in the Trap.
- **v3ManagerUserName**: The name of the user.
- **v3ManagerUserSecModel**: The Security Model used.
- **v3UserSecLevel**: The Security Level of the user.
- **v3ManagerUserContextName**: The Context Name of the user.
- **v3ManagerTimeOut**: Timeout value.
- **v3ManagerRetries**: Number of retries that can be made.
- **v3ManagerStatus**: Row Status of the entry.

This table is multilingual by nature, i.e., it can forward any version of traps to the managers registered in the table. For a v3 trap, the agent calls the IsAccessAllowed() method according to VACM MIB for each variable binding OID and trap OID. If there is no access for the user in the VACM tables, the notification will not be sent to that management target.

12.4 Adding Manager Entries

There are two ways to add Manager entries in the Trap Forwarding Table. It can be done either: (1) Before Agent start-up or (2) During runtime.

12.4.1 Before Agent Startup

To specify the manager entries at startup, you can use MIB Compiler UI, Text File / XML File, Runtime Memory or the API.

**Using MIB Compiler UI**

1. Choose the **Project -> Settings** menu of MIB Compiler UI. Please note that the version of the Agent has to be V1/V2c for working with v1v2TrapForwarding Table and V3 for working with V3TrapForwardingTable. This can be specified in the **Project -> Settings menu -> General** panel.
2. Select **v1v2TrapForwardingTable or v3TrapForwardingTable** in the **Trap** Panel.
3. Click **Add**. By default, you will have an entry with values: IPv4/IPv6, 127.0.0.1/0:0:0:0:0:0:0:1, 8003, Version 2, public, 5000, 0 and Active 1 in V1V2TrapForwardingTable and IPv4/IPv6, 0.0.0.0.0.0.0.1/127.0.0.1, 8003, Version 3, public, noAuthUser, 3, noAuthNoPriv, noAuth, 5000, 0, Active 1 for V3 Trap Forwarding Table. A wizard pops up wherein you can specify the manager entries (v1/v2 or v3).
4. Click **OK**. The manager entry is added.

**Using Text File / XML File**

The entries configured in MIB Compiler UI are stored in the configuration files, V1V2TrapForwardingTable.xml or V1V2TrapForwardingTable.txt after generation. These
files can be found under the `<Agent Toolkit Home>/snmpprojects/projectname/agent/bin/conf` directory.

To specify the type of storage option,
1. Choose the **Project -> Settings** menu of MIB Compiler UI.
2. Select the **v1v2 or v3TrapForwardingTable** from the **Trap** panel.
3. Choose **XML File or Text File** from the **Storage Type** option. By default, XML File is chosen.

These files can also be edited to add manager entries. The **V1V2TrapForwardingTable.xml** given below has been edited for adding a new manager with host "128.0.0.1" and manager port "8004".

```xml
<?xml version="1.0" encoding="UTF-8"?>
<table>
  <row>
    <column name="managerHostType" value="1"></column>
    <column name="managerHost" value="127.0.0.1"></column>
    <column name="managerPort" value="8003"></column>
    <column name="version" value="2"></column>
    <column name="community" value="public"></column>
    <column name="timeOut" value="1000"></column>
    <column name="retries" value="0"></column>
    <column name="rowStatus" value="1"></column>
  </row>

  //Adding new row
  <row>
    <column name="managerHostType" value="1"></column>
    <column name="managerHost" value="128.0.0.1"></column>
    <column name="managerPort" value="8004"></column>
    <column name="version" value="2"></column>
    <column name="community" value="public"></column>
    <column name="timeOut" value="1000"></column>
    <column name="retries" value="0"></column>
    <column name="rowStatus" value="1"></column>
  </row>
</table>
```

Similarly, IPv6 entries can be added in the **V1V2cTrapForwardingTable.xml** or .txt file and v3 entries in the **V3TrapForwardingTable.xml** or .txt file. Please note that the agent has to be restarted for the changes to take effect. You can also access the new entries without restarting the agent. Make use of the following method in the main file for the agent to read the entries directly from the text file.

```java
trapListener.setRefreshTrapTable(int refreshTime)
```

This method has to be added after the code for trapForwardingTable Request Handler instantiation. Please note that this method is in the **SnmpTrapService** class common to both V1V2 and V3 Trap Forwarding Tables.

**Using Runtime Memory**

Runtime memory can be used to store the Manager information in the Agent Memory. Using this option will not store the entries in text files or xml files.

To use the runtime memory for adding Manager entries dynamically,
1. Choose the **Project -> Settings** menu of MIB Compiler UI.
2. Select `v1v2TrapForwardingTable` or `v3TrapForwardingTable` from the Trap panel.
3. Select Runtime Memory from Storage Type.
4. Add entries using the Add option.

After this, follow the steps given in adding entries “From the Manager” section of During Runtime (12.4.2) explained below. Please note that once the agent is killed, the added entries are removed from the memory.

Using API calls

For adding an entry through API calls, an entry similar to the one given below has to be included next to the trapListener.setTrapForwardingTable (forwardingTable) method in the generated Main file.

For v1v2TrapForwarding Table

```java
forwardingTable.addForwardingEntry(com.adventnet.snmp.snmp2.agent.
SimpleTrapForwardingTable.createForwardingEntry(new Integer(1), "127.0.0.1", new
Integer(8004), new Integer(2), "public", new Long(5000), new Long (0));
```

For v3TrapForwarding Table

```java
forwardingTable.addV3ForwardingEntry(com.adventnet.snmp.snmp2.agent.
V3SimpleTrapForwardingTable.createV3ForwardingEntry(new Integer(1), "127.0.0.1", new
Integer(804), new Integer(3), "public", "noAuthUser", new Long(3), new
Integer(1), "noAuth", new Long(5000), new Long (0));
```

After adding these entries save the Main file and compile it. As per the entries added in V1 V2 Trap Forwarding Table / v3 Trap Forwarding Table, Traps will be forwarded to the Manager running at port 8004. You should be able to receive Traps at port 8004.

12.4.2 During Run Time (Dynamic Registration)

Manager entries can be added dynamically during runtime.

From the Manager

To add an Entry from remote,
1. Load the AGENT-SNMP-CONFIG-MIB in MIB Browser application.
2. Select v1v2TrapForwardingTable or v3TrapForwardingTable from the traptables module of agentConfiguration group.
3. By selecting the respective table and clicking SNMP Table icon in MIB Browser will open up a wizard wherein entries can be added to the Trap Forwarding Table on clicking Add.
4. The entries added from remote get updated in the text/xml file.
5. Please note that it is possible to access the table from remote only if "Remote Configuration" is enabled in the Trap Panel (V1V2TrapForwardingTable/V3TrapForwardingTable) of Project -> Settings menu in MIB Compiler UI.
6. You can also enable "Remote Configuration" option using the following API call if it is not chosen before generation.

```java
forTable = new
com.adventnet.snmp.snmp2.agent.SimpleTrapForwardingTable
(forwardingTable);
forTable.addRegistrationListener(hdlr);
```

This method has to be included in the initAdventNetSnmpAgent() method.
12.5 Disable Startup Trap

By default, the generated agent sends coldstart trap at the agent startup to the managers being configured. The user can disable the trap at agent startup by enabling the option provided in the MIB Compiler UI by choosing Project->Settings->Source Generation ->Disable Startup Trap.

The Startup Trap can also be disabled by adding the following code in the main file before restartSnmpAgent( ) method

```java
super.disableStartUpTrap(true);
```

12.6 Testing the Agent for Traps

To test the Trap feature,
1. Start the MIB Browser application.
2. Open the Trap Viewer in MIB Browser application and start listening for Traps in the Manager's port specified in the Trap forwarding Table.
3. Load the respective MIB in the MIB Browser application and start the Agent at the port (say default port) 8001 using run.bat or run.sh from <Agent Toolkit Home>/snmpprojects/projectname/agent/bin directory.
4. At first, you receive a cold start trap. The details of the Trap can be viewed by selecting the Trap and right clicking on it.
5. Now, SET a value to a variable for which Trap is defined.
6. Value would be set for the variable and a Trap will be received in the Manager's Port (mentioned in the Trap Forwarding Table).

12.6 Knowing the Details of the Trap Generated (Reliable Traps)

There are times where the generated Traps could have been missed or lost during communication. In a situation like this, the Manager might not be aware as to how many traps were generated by the Agent and the details of those Traps. To overcome the uncertainty involved in receiving traps, the option of sending Reliable Traps is provided.

12.6.1 Implementing Reliable Traps

This feature can be implemented either from MIB Compiler UI or by using API calls.

**Using MIB Compiler UI**
- Select Project -> Settings menu from the menu bar of MIB Compiler UI.
- Select General from the Trap Panel.
- Check Reliable Traps option.

By default, the Maximum number of Traps that can be stored by the Agent is 10 and so the last 10 traps are stored here. This value is configurable.

**Using API Calls**

Add the following piece of code in the Main file generated before the code for restartSnmpAgent.

```java
trapListener.addRegistrationListener(hdlr);
trapListener.setReliableTraps(true, 10);
```
12.6.2 Accessing the Trap Details

To know the information of Traps generated from the Agent's two tables, namely the notificationLogTable and notificationVarbindTable have been included under the notificationLogs group of AGENT-SNMP-CONFIG-MIB file. These tables help the Manager in finding out the details of the Traps by just sending a query to the Agent. Try acquiring the Trap details by loading the MIB in MIB Browser and querying.

The notificationLogTable stores information of the Traps sent from the Agent (Request ID, Version of the Trap, Time etc..). The Varbind Table stores information of the varbind values present in the Trap PDU (such as OID, Value, Type, etc.).

12.7 Validating and Filtering Outgoing Traps

There are times where the Traps generated by an Agent might be of no use to a Manager. In this case, the Agent drops those Traps without sending them to that particular Manager. This restriction is specified in a condition whereby every generated trap checks this condition and it is validated. If necessary, it is sent to the Manager, else dropped.

To implement this feature, add the following piece of code in the Agent Main File for implementing the feature. Instead of instantiating SnmpTrapService, you have to instantiate the TrapValidator class (replace SnmpTrapService with TrapValidator) wherever it is present in the Main file that is generated, along with the correct constructor that you are using in this class.

```java
class TrapValidator extends SnmpTrapService{
    /**
     * The Constructor with TrapSessionPort from where the Traps will be sent from the Agent.
     */
    public TrapValidator(int port) {
        super(port);
    }
    /**
     * The method to Validate the Traps. This will get called before the sendTrap method.
     * Sends the Traps to the Managers in the TFTable. You have to Instrument the validating conditions in this method fulfilling your requirement.
     * @param pdu The Trap SnmpPDU.
     * @param managerNode The ManagerHost for which the Trap is targeted.
     * @param managerPort The ManagerPort for which the Trap is targeted.
     * @returns true if the Trap is Valid and false if the Trap is Invalid.
     */
    public boolean validateTrap(com.adventnet.snmp.snmp2.SnmpPDU pdu, String managerHost, int managerPort) {
        if (managerPort == 8004 && managerHost.equals("127.0.0.1")) {
            return false;
        } else {
            return true;
        }
    }
}
```

// Sample Condition
// This will restrict any Trap from the Agent, if the TargetPort is 8004 // and Host is localhost.
public boolean validateTrap(com.adventnet.snmp.snmp2.SnmpPDU pdu, String managerHost, int managerPort) {
    if (managerPort == 8004 && managerHost.equals("127.0.0.1")) {
        return false;
    }
    else {
        return true;
    }
}
Load Agent-Sample-MIB in MIB Compiler. Let the Agent be a v2c Agent.
The v1v2 Trap Forwarding Table in Project -> Settings menu should have an entry with the Manager port as 8004 and Host as 127.0.0.1.
Generate code for the same.
Instrument the Main file by adding the piece of code given above. SnmpTrapService should be replaced with TrapValidator.
Compile the code. The Agent gets created at the output directory.
Test the Agent, by sending a SET request to a Trap defined variable.
Open the Trap Browser and listen for Traps. If Traps are listened at port 8004, this particular Trap should not be received.

12.8 Implementing Your Own Table for Storing the Manager Information
You can also define your own Trap Forwarding Table instead of using the default Trap Forwarding TABLE in AGENT-SNMP-CONFIG-MIB. To achieve this purpose:

- Define a similar Trap Forwarding Table in your MIB using the OBJECT TYPE TABLE construct in MIB Editor. Please refer to "Adding a Table" in Creating a MIB section for using the construct.
- The OID structure of the newly defined table should be same as the TrapForwardingTable in AGENT-SNMP-CONFIG-MIB after the enterprise level. (.1.3.6.1.4.1. your enterpriseOID).
- Add the following piece of code above the code for restartSnmpAgent in the Main file, if it is a v2c Agent. The entry is similar for a V3 Agent also.

```
trapListener.setForwardingEntryOID(".1.3.6.1.4.1.2345.10.3.1.2.2.1.1");
trapListener.addRegistrationListener(hdlr);
```
- .1.3.6.1.4.1.2345.10.3.1.2.2.1.1 is a sample entry oid.
- Now, compile the code and test the Agent for Traps.
- Traps will be generated to the interested Managers as specified in your Trap Forwarding Table.

12.9 Implementing Informs
SNMP Notifications can be sent as Traps or Inform requests. Traps are unreliable because the Manager does not send any acknowledgment on receiving the Trap. So, the Agent cannot determine if the Trap was received by the Manager. However, an SNMP Manager that receives an Inform request acknowledges the message with an SNMP response PDU. If the Manager does not receive an inform request, it does not send a response. If the Agent does not receive a response at all, the inform request can be sent again.

12.9.1 Implementing Informs
Informs can be implemented by adding the following piece of code in the generated Main file before the code for restartSnmpAgent

```
trapListener.setInformFlag(true);
```
Save the file and compile the code. This would send all the Trap messages to the Manager in the form of Informs and wait for an acknowledgment message. To know more on testing, please refer to Testing Informs topic.
12.9.2 Testing Informs

To test the same,

- Start the MIB Browser application.
- Load the MIB (using which the Agent is created) and start the Agent.
- Now, SET a value to a variable for which Trap is defined. (Assuming that "Generate Trap on Set" option is enabled in the MIB Compiler UI Project -> Settings menu before generation).
- Value would be set for the variable and an Inform Request will be sent to the Manager.
- This Inform Request sent to the Manager can be viewed using Trap Viewer icon. Hence, open the Trap Viewer wizard before sending a request
- This will listen for Inform requests. Please note that it is not possible to send Informs from an SNMPv1 Agent.

12.10 Sending Traps for Log4j Log Messages

Agent Toolkit has a facility for sending Traps for the Log4j Log Messages using SNMPTrapAppender utility. To enable SNMPTrapAppender utility, a new class extending SNMPTrapAppender class of SNMPTrapAppender utility is bundled with the product. Using this class, there are two ways by which Traps can be sent for the Log4j log messages. They are:

- Using the Agent's Own SnmpTrapService Method and
- Using the XML File Configuration

12.10.1 Using the Agent's own SnmpTrapService.

In this approach, the existing SnmpTrapService (i.e. in AdventNet's Agent) is used to send the Log messages as Traps. Here the Traps are forwarded to all the Managers present in the Trap Forwarding Table which is associated with the Agent's SnmpTrapService. The Trap PDU is filled with the details of the Entries in the Forwarding Table.

The following are the steps that illustrates the method:
Add the following code in the generated main file.

```java
// The logger to log the messages
private org.apache.log4j.Logger logger =
    org.apache.log4j.Logger.getLogger(getClass().getName());

// boolean to specify whether the log4j has been initialized
private boolean log4jInitialized = false;

/**
 * This method initializes the SNMPTrapAppender with the Agent's own TrapService.
 * This must be used if the Traps are to be forwarded based on the Forwarding Table
 * associated with this Agent.
 */
private void initAppenderWithTrapService(){
    // Instantiating Trap Appender with our Agent's SnmpTrapService
    com.adventnet.utils.appender.SnmpTrapAppenderExtension appender = new
    com.adventnet.utils.appender.SnmpTrapAppenderExtension();
    appender.setName("WARN_LOG");
    appender.setImplementationClassName("com.adventnet.utils.
    appender.SnmpTrapSenderImpl");
    appender.setApplicationTrapOID(".1.3.6.1.4.1.2162.4.3.2");
```
appender.setEnterpriseOID(".1.3.6.1.4.1.2162");
appender.setTrapIndex(4);
appender.setThreshold(org.apache.log4j.Level.WARN);
appender.setForwardStackTraceWithTrap("true");
appender.setLayout(new org.apache.log4j.PatternLayout());
appender.setTrapSource(1);
appender.setThresholdSource(trapListener);
log4jInitialized = true;
logger.addAppender(appender);
logger.debug("This is a Debug Message");
logger.error("An Error occurred here");
logger.info("Here's some info about that");
logger.fatal("Help! I'm dying out here!");
logger.fatal("Here's an Exception!", new Exception("Exception message"));

Call the method initAppenderWithTrapService() in the generated main file after restartSnmpAgent() method.

The log messages that are at the higher level of the threshold being set will be converted to trap. Here the default threshold level is WARN and all the log messages higher to this level will be sent as trap by using this utility.

12.10.2 Using the XML file Configuration

In this approach, the SNMPTrapAppender is configured with the details read from a XML file. Here the Log Message Traps are forwarded to a Single Manager specified in the XML file. A sample appender configuration file, SnmpTrapAppenderTestConfig.xml is bundled with the product under <Agent Toolkit Home>/conf/appenderConfig/ directory.

The following are the steps that illustrates the method:

Copy the configuration file to <Agent Toolkit Home>/snmpprojects/projectname/agent/bin/conf directory.
Add the following code in the generated main file.

```java
// The logger to log the messages
private org.apache.log4j.Logger logger =
org.apache.log4j.Logger.getLogger(getClass().getName());
// boolean to specify whether the log4j has been initialized
private boolean log4jInitialized = false;
/**
* Method to initialize the SnmpTrapAppender based on a XML config file.
* /
public void initAppenderWithConfigFile()
{String confFile =
"conf/SnmpTrapAppenderTestConfig.xml";
if (!log4jInitialized) {
try {
org.apache.log4j.xml.DOMConfigurator.configure(confFile);
log4jInitialized = true;
} catch (Exception ex) {
org.apache.log4j.helpers.LogLog.error("Error initializing logging!", ex);
}
}
if (!log4jInitialized)
```
org.apache.log4j.helpers.LogLog.error("Logging initialization error.");
else {
    logger.debug("This is a Debug Message");
    logger.error("An Error occured here");
    logger.info("Here's some info about that");
    logger.fatal("Help! I'm dying out here!");
    logger.fatal("Here's an Exception!", new Exception("Exception message"));
}

Call the method initAppenderWithConfigFile() in the generated main file after restartSnmpAgent() method.

The log messages that are above the threshold level will be converted to trap. Here by default, the threshold level is DEBUG and all the log messages above this level will be sent as traps.

Thus the utility can be used according to the environment chosen.

12.10.3 Testing the Agent

For simple testing with AdventNet Agent Toolkit,

- Create a project.
- Generate the code for the agent.
- Add "log4j-1.2.8.jar" and "snmpTrapAppender_1_2_91.jar" in the classpath settings of run.bat\sh file. These jars are available in <Agent Toolkit Home>/jars directory.
- Implement SNMPTrapAppender functionality by using anyone of the above methods.
- Save the file & Compile the code.
- Start the TrapBrowser at localhost : 8003 port from the MIB Browser application.
- Also, start the Agent.
- You will get the following traps namely:
  - The Standard coldStart trap,
  - Traps, if the log messages are above the threshold level.

Thus Traps can be sent for any kind of log messages.

12.11 Running the Example

To know more about Traps, please go through the example available in <Agent Toolkit Home>/examples/snmp/traps directory. The "simpletrap" example sends a Trap to the Agent on every SET request. The main file TrapExample.java forwards the trap to the Manager specified in it. The readme.html will help you understand how to run this example.
13.0 Implementing SNMP Proxy

13.1 Overview

In original model of SNMP Management, it is a monolithic agent that used to carry out all the management responsibilities on a given network element (node). This solution was not flexible enough to provide an effective management of increasingly complex and distributed systems. In addition to the agents typically provided by computer manufacturers for hardware and operating system information, agents are also produced by other software vendors for SQL, Application servers, etc.

Also in distributed systems where different components of the product will be present in different network nodes (different IP address), each component needs its own management requirement for which each component will have an SNMP agent. However, there should be a single gateway for managing the entire product (distributed components).

This disadvantage of the original SNMP model, being unable to accommodate such complex and heterogeneous systems, is solved using Master - Subagent concept where the Master agent acts as a proxy for other SNMP agents. AdventNet Agent Toolkit provides generic comprehensive Master Subagent architecture for making agents Subagent to Master agent on the fly. This dynamic behavior can be accomplished by the "Dynamic Registration concept in Master Subagent Architecture".

13.2 SNMP Master Agent

The Master agent is an entity or process on a managed node that exchanges SNMP messages with the Management applications such as AdventNet Web NMS, HP Openview, IBM Tivoli, etc. It acts as a primary interface between the Network Manager and Subagents.

13.3 SNMP Subagent

Subagents are processes that access the management information and provide manageability to various applications/components within a system. These Subagents interact with the Master agent using SNMP. They do not interact with the Managers directly.
13.4 Creating Master Agent

All the agents generated using MIB Compiler can act as both Master agent and Subagent. By default, the generated agent acts as a standalone monolithic agent. Follow the steps given below to make the standalone agent, a Master agent.

Using MIB Compiler UI

- Create a Project and Load a MIB (MIB that you would like to make as a Master agent).
- Select Project -> Settings menu from the menu bar of MIB Compiler UI.
- Choose the Proxy Panel.
- Check Generate Agent as Proxy option.
- By default, the OID Based Proxy option is selected. If preferred, the other options Context Based Proxy or Instance Based Proxy can be selected.
- After selecting these options, generate and compile the code.
- The agent gets created as a Master agent with Proxy support.
- By default you will have a Subagent entry registered for every type of Proxy Table.
- To register more Subagents, refer to the topic "Registering Subagents in the Master Agent".

Note: Master agent can be created with or without using a MIB.

Using API

To enable Proxy using API calls, add any one of the following piece of code to the Master agent's Main File before the code for populating entries for aclTable based on the registration.

```java
//For OID Based Proxy
dyn = new DynamicRegistration(false, "conf", "ProxyTable.xml");
dyn.addRegistrationListener(hdlr, true);

//For Context Based Proxy
dynCom = new DynamicRegistrationWithCommunity(false, "conf", "ProxyTable.xml");
dynCom.addRegistrationListener(hdlr, true);

//For Instance Based Proxy
dynIns = new DynamicRegistrationWithInstance(false, "conf", "ProxyTable.xml");
dynIns.addRegistrationListener(hdlr, true);
```

These API call implements Proxy. Please note that the api calls will work provided the variable declaration private DynamicRegistration dyn = null; / private DynamicRegistrationWithCommunity dynCom = null; / private DynamicRegistrationWithInstance dynIns = null; are included respectively (based on the type of registration) in the Main file where all declarations are present.

13.5 Creating Subagent

All the agents generated using MIB Compiler can act as both Master agent and Subagent. By default, the generated agent can directly act as Subagent. There is no specific option to make an agent act as a Subagent. Any SNMP agent can act as a Subagent. The Subagent's host, port number details are
to be registered in the Master agent for forwarding the requests to the exact Subagent. So, create a
simple agent using a MIB (from which you prefer to access the details) and make it act as a
Subagent.

13.6 Registering Subagents in the Master Agent

There are two ways to register Subagents in the Master agent. It can be done either : (1) Before
Agent Startup or (2) During Run Time.

13.6.1 Before Agent Startup

By registering the Subagent to the Master agent before agent startup, the Subagent
information is hard coded in the Java file. To register the Subagent before agent-start up you
can use either the MIB Compiler UI or Text/XML Files or API calls.

Using MIB Compiler UI

- Select Project -> Settings menu from the MIB Compiler UI.
- Choose the Proxy Panel.
- Check Generate Agent as Proxy option.
- By default, the OID Based Proxy option becomes enabled.
- Select proxyTable from the Panel.
- Add Subagent entries to the table and click OK.

Using Text / XML Files / Runtime Memory

Once the entries are added through MIB Compiler UI, the configurations are saved in a text
or XML file namely ProxyTable.txt / .xml (for OID Based Proxy) under <Agent Toolkit
Home>/snmpprojects/projectname/agent/bin/conf directory. To make use of this text file or
xml file storage,

- Select Project -> Settings menu from the MIB Compiler UI.
- Select proxyTable from Proxy Panel.
- Choose XML File or Text File from the Storage Type Option. By default, XML is
  chosen.

These files can be edited to add Manager entries. The ProxyTable.xml given below has been
edited for adding a new Subagent entry with OID ".1.3.6.1.2.1.1" ; Manager Port : "8004".

```xml
<?xml version="1.0" encoding="UTF-8"?><Table><row>
  <column name="proxyOid" value=".1.3.6.1.2.1.1"></column>
  <column name="proxyHost" value="localhost"></column>
  <column name="proxyPortNumber" value="161"></column>
  <column name="proxyVersion" value="2"></column>
  <column name="proxyCommunity" value="public"></column>
  <column name="proxyTimeout" value="5000"></column>
  <column name="proxyRetries" value="0"></column>
  <column name="proxyRowStatus" value="1"></column>
</row><row>
  <column name="proxyOid" value=".1.3.6.1.2.1.1"></column>
  <column name="proxyHost" value="localhost"></column>
  <column name="proxyPortNumber" value="8004"></column>
  <column name="proxyVersion" value="2"></column>
  <column name="proxyCommunity" value="public"></column>
  <column name="proxyTimeout" value="5000"></column>
</row></Table>
```
Similarly, the text/xml files get generated for Context Based Proxy and Instance Based Proxy and they can be edited for adding new Subagent entries. Please note that the agent has to be restarted for the changes to take effect.

**Run-Time Memory**

Run-Time memory can be used to store the Manager information in the agent Memory. Using this option does not store the entries in text files or xml files. To use the run time memory for adding Manager entries dynamically, follow the instructions given below.

1. Choose **Project -> Settings** menu from the menu bar of MIB Compiler UI.
2. Select **proxyTable** from the Proxy Panel (can be any Proxy Table you prefer).
3. Select **Runtime Memory** from the Storage Type.
4. Add entries using the **Add** option.

After this, follow the steps given in adding entries From the Manager i.e During Runtime (13.6.2). Please note that once the agent is killed, the added entries are removed from the memory.

**Using API Calls**

Add the following piece of code in the generated Main file of Master agent before the code for aclTable registration to register a Subagent entry. (OID Based Registration)

```
dyn = new DynamicRegistration(false);
Vector subAgentEntries = new Vector();

com.adventnet.utils.agent.DynamicRegistrationEntry entry0 =
dyn.createDynamicRegistrationEntry(".1.3.6.1.2.1","localhost",
    new Integer("161"), new Integer("2"), "public", new Long("4000"),
    new Long("0"));
subAgentEntries.addElement(entry0);

dyn.setTableVector(subAgentEntries);
dyn.addRegistrationListener(hdlr, true);
```

Using this api method will also include Subagents to the Master agent. For Context Based Proxy and Instance Based Proxy the following api calls can be used

```
//For Community Based Dynamic Registration

dynCom = new DynamicRegistrationWithCommunity(this, false);
Vector subAgentEntries = new Vector();

com.adventnet.utils.agent.DynamicRegistrationEntry entry0 =
dynCom.createDynamicRegistrationEntry(".1.3.6.1.2.1", "public",
    "localhost", new Integer("161"), new Integer("2"), "public", new Long("4000"),
    new Long("0"));
subAgentEntries.addElement(entry0);

dynCom.setTableVector(subAgentEntries);
dynCom.addRegistrationListener(hdlr, true);
```

```
//For Instance Based Dynamic Registration
```
**SNMP Agent**

```java
dynIns = new DynamicRegistrationWithInstance(false);
Vector subAgentEntries = new Vector();

com.adventnet.utils.agent.DynamicRegistrationEntry entry0 =
dynIns.createDynamicRegistrationEntry(".1.3.6.1.2.1.2.2.1",".
1", "localhost", new Integer("161"), new Integer("2"), "public", new Long("4000"), new Long("0"));
subAgentEntries.addElement(entry0);
dynIns.setTableVector(subAgentEntries);
dynIns.addRegistrationListener(hdlr, true);
```

### 13.6.2 During Run Time

Subagents can also be registered in the Master agent during run time.

**From the Manager**

To add Subagent entries to the Master agent's Proxy Table from the Manager, follow the steps given below:

1. Load **AGENT-SNMP-CONFIG-MIB** in the MIB Browser application.
2. Select `proxyTable` (or whichever preferred Subagent table) from the `subAgentTables` module of `agentConfiguration` group.
3. Selecting the respective table and clicking **SNMP Table** icon (View SNMP Table Data) in MIB Browser open up a wizard wherein entries can be added to the corresponding proxyTable.
4. Click Add in that SNMP Table wizard. The corresponding columns of the Table are listed.
5. Include the Subagent entries. The entries added from remote get updated in the text/XML file provided the storage option is selected.
6. Enable "Remote Configuration" option by selecting **Proxy Panel of Project -> Settings** menu in MIB Compiler UI or including any of the following API in the generated main method. Please note that it is possible to access the table from remote only if "Remote Configuration" is enabled.

By just including the below API for a particular type of Proxy in the initSnmpExtensionNodes() method in the generated main file, remote configuration of the respective Proxy Table is possible.

**For OID Based Proxy**

```java
dyn = new DynamicRegistration(false, "conf", "ProxyTable.xml");
dyn.addRegistrationListener(hdlr, true);
```

**For Context Based Proxy**

```java
dynCom.setTableVector(subAgentEntries);
dynCom.addRegistrationListener(hdlr, true);
```

**For Instance Based Proxy**

```java
dynIns.setTableVector(subAgentEntries);
dynIns.addRegistrationListener(hdlr, true);
```

By making the argument as false [dynIns.addRegistrationListener(hdlr, false);], the facility of Remote configuration is disabled.

AdventNet Inc. 89
13.7 Different Types of Master - Subagent Relationship

Considering the design and the complexity of the distributed systems, it is very difficult to have just normal OID-based Master - Subagent relationship. This made us evolve the Master - Subagent relationship further and made us support different types of Master - Subagent relationship such as Context-based registration and Instance-based registration.

13.7.1 OID-based Registration

This is the normal registration where all Subagents implement different MIBs or different groups in the MIB. The proxyTable in AGENT-SNMP-CONFIG-MIB is used for this type of registration. Following are the columns present in the proxyTable: -

- proxyOid - OID of the Subagent.
- proxyHost - Host of the Subagent.
- proxyPortNumber - Port Number in which the Subagent is running.
- proxyVersion - Version of the Subagent using which the Subagent is to be contacted or queried.
- proxyCommunity - The Community with the Subagent is to be contacted or proxied.
- proxyTimeout - The time for which the Master agent should wait for the response from the Subagent.
- proxyRetries - Number of Retries to be made, in case the Subagent does not send a response within the time out value specified before.
- proxyRowStatus - Row Status column facilitating addition and deletion of rows. If the status is not active then the request will not be proxied to the corresponding Subagent.

So, "Create a Master agent and a Subagent". To "Register Subagents in the Master Agent", please refer to the previous topic. With the explanation provided in "Registering Subagents" topic, you can add Subagent entries to the Proxy Table. Thus, General Master - Subagent feature is achieved. To test the same, please have a look at Running the Example, which explains the feature with an example.

13.7.2 Context-based Registration

This kind of registration helps in having more than one Subagent implementing the same MIB or a subtree in a MIB. In this case, the Master agent should have a criterion to identify the Subagent to forward the request. Community of the Subagent is made as the criteria. The contextBasedProxyTable in AGENT-SNMP-CONFIG-MIB stores the details of the Subagents. Following are the columns present in the contextBasedProxyTable: -

- proxyContextOid OID of the Subagent.
- proxyContextName - The Community Name of the Subagent to which the request is to be proxied i.e the request from the Manager for the Subagent will be authenticated only if such a community exists in the Master agent's aclTable.
- proxyContextHost - Host of the Subagent.
- proxyContextPortNumber - Port Number in which the Subagent is running.
- proxyContextVersion - Version of the Subagent using which the Subagent is to be contacted or queried.
- proxyContextCommunity - The Community with the Subagent is to be contacted or proxied i.e. the request from the Master agent will be authenticated by the Subagent only if such a community exists in the Subagent's aclTable.
- proxyContextTimeout - The time for which the Master agent should wait for the response from the Subagent.
• proxyContextRetries - Number of Retries to be made, in case the Subagent does not send a response within the time out value specified before.

• proxyContextRowStatus - Row Status column facilitating addition and deletion of rows. If the status is not active then the request will not be proxied to the corresponding Subagent and so there will not be any response from the Subagent.

So, "Create a Master Agent and a Subagent". To "Register Subagents in the Master Agent", please refer to the previous topic. With the explanation provided in "Registering Subagents" topic, you can add Subagent entries to the Context-based Proxy Table. Thus, Context-based Master - Subagent feature is achieved. To test the same, please have a look at Running the Example, which explains the feature with an example.

13.7.3 Instance-based Registration

This kind of registration is used when the Subagents implement different rows of a table. The Master agent identifies the Subagent based on the instance value, present in the request received from the Manager, and forwards the request to the respective Subagent. The instanceBasedProxyTable in AGENT-SNMP-CONFIG-MIB stores the details of the Subagents. Following are the columns present in the instanceBasedproxyTable :-

• proxyEntryOid - Entry OID of the Subagent Table.

• proxyEntryInstance - Instance value of the Subagent.

• proxyInsHost - Host of the Subagent.

• proxyInsPortNumber - Port Number in which the Subagent is running.

• proxyInsVersion - Version of the Subagent using which the Subagent is to be contacted or queried.

• proxyInsCommunity - The Community with the Subagent is to be contacted or proxied.

• proxyInsTimeout - The time for which the Master agent should wait for the response from the Subagent.

• proxyInsRetries - Number of Retries to be made, in case the Subagent does not send a response within the time out value specified before.

• proxyInsRowStatus - Row Status column facilitating addition and deletion of rows. If the status is not active then the request will not be proxied to the corresponding Subagent and so there will not be any response from the Subagent.

So, "Create a Master Agent and a Subagent". To "Register Subagents in the Master Agent", please refer to the previous topic. With the explanation provided in "Registering Subagents" topic, you can add Subagent entries to the Instance-based Proxy Table. Thus, Instance-based Master - Subagent feature is achieved. To test the same, please have a look at Running the Example, which explains the feature with an example.

13.8 Knowing the Status of the Subagent (Heart Beat Mechanism)

As the name implies, Heart Beat Mechanism is helpful to determine the existence of a connection between Master - Subagent. This feature is implemented to constantly monitor the status of the Subagent and make it available to the Master agent.

By knowing the status, the Master agent does not forward the request to Subagents, which are not alive. Instead, the Master agent throws a 'general failure' error directly to the Manager.

13.8.1 Enabling Heart Beat Mechanism

Heart Beat mechanism can be enabled using MIB Compiler UI options or the API calls. To enable this feature in the Master agent,
Using MIB Compiler UI

- Select **Project -> Settings** menu from the MIB Compiler UI while creating a Master agent.
- Choose the **Proxy** Panel.
- Check **Generate Agent as Proxy** option.
- By default, the **OID Based Proxy** option is enabled. Select the preferred type of Proxy.
- Check **Heart Beat Mechanism** option to implement the polling mechanism in Proxy.

Using API calls

Instead of the MIB Compiler UI options, the following API calls can be used for enabling Heart Beat Mechanism. Add the following piece of code in the initSnmpExtensionNodes method of the Master agent's generated Main file after the Dynamic Registration class instantiation to enable Heart Beat with the default values. For OID-based Proxy, add the following code.

```java
dyn.checkSubAgentHeartBeat(true, null, null, null, this);
```

Here, 'dyn' represents the instance of Dynamic Registration. This dyn varies according to the type of registration. For Instance-based and Context-based Proxy the code has to be as follows:

```java
dynIns.checkSubAgentHeartBeat(true, null, null, null, this);
dynCom.checkSubAgentHeartBeat(true, null, null, null, this);
```

The Parameters true, null, null, null, this represents the following respectively:
- boolean toPoll - Here, 'true' enables polling and 'false' disables polling.
- Long pollInterval - Specifies the required interval in Polling (by default, 5000). null takes the default value.
- Integer timeOut - States the time out value of the requests (by default, 1 second). null takes the default value.
- Integer retries - States the number of retries to be done (by default, one time). null takes the default value.
- SnmpAgent agent - Specifies the reference to the SnmpAgent for sending traps. Because the generated main file extends SnmpAgent, the value for the parameter SnmpAgent can be passed as "this".

Thus the code for Heartbeat is:

```java
checkSubAgentHeartBeat (boolean toPoll, Long pollInterval, Integer timeOut, Integer retries, SnmpAgent agent);
```

To know how this Heart Beat functionality works refer to Running the Example section.

13.8.2 LinkUp and LinkDown Traps

Subagent's status is intimated to the Master agent through Heart Beat support. As per this support, the Row Status column of the proxyTable in the Master agent is updated frequently with NOT IN SERVICE and ACTIVE status. When the Subagent status is down, the Row Status column becomes NOT_IN_SERVICE and when it is up, the row status column becomes ACTIVE.

On becoming NOT_IN_SERVICE, a Link Down Trap can be generated and forwarded to the Managers by the Master agent. Similarly, a Link Up Trap can be generated when the status shifts to ACTIVE.
To know which Subagent is down or active, the Trap Details can be viewed. The Trap PDU contains the details of the Subagent Host and Port number and using these details Subagents can be identified by the Manager.

These Trap details can be viewed by loading the AGENT-SNMP-CONFIG-MIB in MIB Browser application and querying the related tables. The OID of the General Based Proxy linkUp and linkDown traps in the AGENT-SNMP-CONFIG-MIB are .1.3.6.1.4.1.2162.10.3.1.4.1 and .1.3.6.1.4.1.2162.10.3.1.4.2 respectively. The Managers to whom these traps should be sent are identified using the entries in Trap Forwarding Table.

13.9 Sending Traps From Subagent Through Master Agent

Subagents generate traps and send them through the Master agent to the Manager. Even though traps can be sent directly to the Manager from the Subagent, normally the traps are forwarded to the Master agent. The Master agent forwards them to the interested Managers. This is most applicable in the real time environment and the details of the Managers (to whom these traps should be sent) can be availed from the Trap Forwarding Table. For more information about the Trap Forwarding Table and configuring Managers in the Forwarding Table, refer to Traps section.

13.9.1 Filtering Traps Received from Subagent Using Trap Filtering Table

To restrict the generation of certain traps to the manager, the Subagent specifies a condition wherein every Trap passes through that condition before it is forwarded to the Manager through the Master agent. Only if that condition is satisfied, the trap reaches the manager. Thus, filtering is done and if required by the Manager, traps are sent across or else they are dropped.

To filter and send traps through the Master agent, a Trap Filtering Table is maintained by Agent Toolkit in the Master agent. This Trap Filtering table maintains information of the Subagents registered with the Master agent (Proxy Table). The Subagent registrations in the Trap Filtering Table purely depends on your requirement. You can either include all the Subagent details present in the Master agent (proxyTable) or a particular set of Subagents from which you prefer to receive traps. This table can also be used even if Master Subagent concept is not used.

The Trap Filtering Table contains the following columns defined in it:

- **agentHostType (Index Column)** - Type of Addressing Model (IPv4/IPV6)
- **agentHost (Index Column)** - Host of the Subagent.
- **agentTrapPortNumber (Index Column)** - Port number from which the Subagent sends traps.
- **agentCommunity** - Community of the Subagent.
- **agentStatus** - Row Status column facilitating addition and deletion of rows.

13.9.2 Enabling Trap Filtering Table

You can enable Trap Filtering Table either by using MIB Compiler UI or by using API Calls.

**Using MIB Compiler UI**

- Select **Project -> Settings** menu from the MIB Compiler UI.
- Select **Proxy** Panel.
- Check **Generate Agent as Proxy** option.
- By default the **OID Based Proxy** option becomes enabled.
- Check **Trap Filtering Table** option.
- Enter the port number for **TrapReceiver Port**. This port opens a different session for receiving traps for the Master agent and forwards it to Managers. The default port mentioned here (8005). Thus support for Trap Filtering Table is enabled.

  **Note:**
  - The TrapReceiverPort can be configured without enabling the Trap Filtering Table.
  - v1v2ManagerPort implies the target port to which the traps are to be sent. In TrapForwardingTable of the master-subagent relationship, the above port in the Master Agent refers to the port where the MIB Browser listens for traps. In the case of a Sub-agent this should be the port where the Master agent listens for traps.

**Using API calls**

Alternatively, you can add the following piece of code in the Master agent Main file toward the end of initSnmpExtensionNodes() after the SnmpTrapService instantiation, to enable Trap Filtering Table support.

```java
// This takes care of forwarding the traps from Sub agents received at the port 8005. They get generated by default if Proxy is enabled. IN case proxy entries are added using API calls, then this entry has to be added.
SnmpTrapReceiverService trapReceiverService = new SnmpTrapReceiverService(8005);
trapReceiverService.addTrapRequestListener(trapListener);

// For TrapFilteringTable.
trapTableListener = new com.adventnet.snmp.snmp2.agent.TrapTableRequestHandler (this, false);
trapTableListener.addRegistrationListener (hdlr);
trapReceiverService.setAgentTableModel (trapTableListener.getAgentTableModel());
```

The following declaration has to be included in the Main file where all Variable declarations are present. The API call will work only when this declaration is included.

```java
private com.adventnet.snmp.snmp2.agent.TrapTableRequestHandler trapTableListener = null;
```

**13.9.3 Adding Subagent Entries to Trap Filtering Table**

Subagent entries can be added to the Table either: (1) Before Agent Startup or (2) During Run Time.

**13.9.3.1 Before Agent Startup**

To add Manager Entries at Agent Startup, either use MIB Compiler UI option or Text/XML File / Runtime Memory option or API calls.

**Using MIB Compiler UI**

1. Select **Project -> Settings** menu from the MIB Compiler UI.
2. Select **Proxy** Panel.
3. Check **Generate Agent as Proxy** option.
4. By default, the **OID Based Proxy** option is enabled. Choose whichever proxy you prefer.
5. Check **Trap Filtering Table** option. By default, the Trap Receiver Port is 8005.
6. Select **trapFilteringTable** from the Proxy Panel and
7. Click **Add** to add the Subagent entries already existing in the Proxy Tables.

**Using Text / XML File/ Runtime Memory**

Once the entries are added through MIB Compiler UI and code is generated, the configurations are saved in a text or xml file namely TrapFilteringTable.xml/txt under `<Agent Toolkit Home>/snmpprojects/projectname/agent/bin/conf` directory. To make use of this text file or XML file storage,

1. Select Project -> Settings menu from the MIB Compiler UI.
2. Choose the Proxy Panel.
3. Check Generate Agent as Proxy option.
4. By default, the OID Based Proxy option is selected.
5. Select **trapFilteringTable**.
6. Choose XML or Text File from the Storage Type Option. By default, XML File is chosen.

These files can be edited to add Manager entries. The TrapReceiverTable.xml given below has been edited for adding a new Subagent entry with agent Trap Port : "8004" and agentCommunity : "private".

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Table>
  <row>
    <column name="agentHostType" value="IPv4"></column>
    <column name="agentHost" value="127.0.0.1"></column>
    <column name="agentTrapPortNumber" value="161"></column>
    <column name="agentCommunity" value="public"></column>
    <column name="agentStatus" value="1"></column>
  </row>
  <row>
    <column name="agentHostType" value="IPv4"></column>
    <column name="agentHost" value="127.0.0.1"></column>
    <column name="agentTrapPortNumber" value="8004"></column>
    <column name="agentCommunity" value="private"></column>
    <column name="agentStatus" value="1"></column>
  </row>
</Table>
```

Please note that the agent has to be restarted for the changes to take effect.

**Runtime Memory**

Run-time memory can be used to store the Manager information in the agent Memory. Using this option does not store the entries in text files or XML files. To use the run-time memory for adding entries dynamically, follow the instructions given below:

1. Choose Project -> Settings menu from the menu bar of MIB Compiler UI.
2. Select Trap Filtering Table from the Proxy Panel.
3. Select Runtime Memory from Storage Type.
4. Add entries using the Add option.

After this, follow the steps given in adding entries From the Manager i.e During Runtime (13.9.3.2). Please note that once the agent is killed, the entries added are removed from the memory.
Using API Calls

Add the following piece of code in the generated Main file of Master agent after the code of Trap Table Listener instantiation i.e., above "trapTableListener.addRegistrationListener(hdlr);

```
trapTableListener.createAndAddTrapFilteringEntry(new Integer(1), "127.0.0.1", 161, "public");
```

This will also add entries to the Trap Filtering Table.

13.9.3.2 During Run Time

From the Manager

To add Subagent entries to the Master agent's Trap Filtering Table from the Manager, follow the steps given below:

1. Load AGENT-SNMP-CONFIG-MIB in the MIB Browser application.
2. Select trapFilteringTable from the subAgentTables module of agentConfiguration group, in the MIB.
3. Selecting the respective table and clicking on SNMP Table icon in MIB Browser open up a wizard wherein entries can be added to the trapFilteringTable.
4. The entries added from remote get updated in the text/xml file.
5. Please note that it is possible to access the table from remote only if "Remote Configuration" is enabled in the Proxy Panel of Project -> Settings menu in MIB Compiler UI.

For better understanding on sending traps through Master agent please go through the example available in <Agent Toolkit Home>/examples/snmp/traps/trapthroughmasteragent directory. The information in readme.html will help you run the example.

13.10 Security in Proxy (Using SNMPv3)

AdventNet SNMP Agent also supports Proxy Concept in SNMPV3. Mostly, the Master and Subagents run in a private network and the Manager in a private network. So the only communication that takes place is between a Manager and the Master agent. Hence, the agents running in a private network may not require SNMP V3 as they would be well secured.

Having this in mind, AdventNet's implementation provides only v1/v2c support to Subagents. Even if the Manager sends a request of V3 type, the Master agent being a V3 agent would convert the V3 request to a V1/V2c Request and forward the PDU to the Subagents. The V1/V2c Response would be further converted to a V3 response and sent back to the Manager.

13.11 Making AdventNet Agents as Subagents to Third-Party Agents (including OS Native agents)

Because the Master - Subagent communication uses SNMP PDU, any agent that understands SNMP can be made as Subagent to AdventNet Master agent. Even the Native agents can be made as Subagent to AdventNet Master agent. AdventNet agents can also be made as Subagents to Third-Party agents / Native agents. This helps to derive the data of the OS you prefer. The supported OS by AdventNet are Windows, linux, and Solaris.

13.11.1 Configuring AdventNet Agent as Subagent to Solaris Agent

To make AdventNet agent as Subagent to Solaris agent,

- Generate an SNMP Agent using Agent Toolkit with the MIB file that you want to make as a Subagent.
- Compile the class files.
To register this AdventNet agent as a Subagent to Solaris agent, go to the directory 
<Agent Toolkit Home>/snmpprojects/projectName/agent/bin from the command line 
and add the compiled classes in to a jar file.

This jar has to be created for the generated classes to be present in the classpath.

```
jar -cvf com.jar com
```

Change the following entries in the files "sample-agent.reg" and 
"RunSolsticeAgent.sh" present under <Agent Toolkit Home>/subagents/solarisutils 
directory.

The "sample-agent-reg" file contains the details of the port number for which the 
Subagent is running and the OID to be proxied.

In RunSolsticeAgent.sh file, include the jar in the CLASSPATH that was earlier 
created from generated classes. Also, change the corresponding class that should 
run for this : com.myCompany.myPackage.AGENT_SAMPLE_MIB

Then, copy the following files from <Agent Toolkit Home>/subagents/solarisutils to 
/etc/snmp/conf directory of Solaris OS.

- AdventNetSnmp.jar
- AdventNetSnmpAgent.jar
- AdventNetAgentUtilities.jar
- AdventNetLogging.jar
- AdventNetAgentRuntimeUtilities.jar
- crimson.jar
- jaxp.jar
- xalan.jar
- The GENERATED JAR
- sample-agent.rsorc
- sample-agent.reg
- RunSolsticeAgent.sh

Go to /etc/rc3.d and execute S*snmpdx as follows (Where * represents a number).

Use the command sh S*snmpdx stop for stopping the Solaris Master agent that is 
running already and sh S*snmpdx start to start the Solaris Master agent to start the 
AdventNet Subagent.

The Master agent starts at port 161 and the Subagent at 8001.

Now, invoke the MIB Browser application and load AGENT-SAMPLE-MIB.txt.

Change the port in the MIB Browser settings to 161 where the Master agent is 
running and do a GET request to the node ftpServername. The results will be given.

### 13.11.2 Configuring AdventNet Agent as Subagent to Windows Agent Using apadll

To make AdventNet agent as Subagent to Windows agent,

- Create an SNMP agent (Subagent) using Agent Toolkit.
- Go to <Agent Toolkit Home>/subagents/NTutils from the command prompt. Execute 
  proxyReg.exe file. The "proxyReg.exe" is used to register the apadll.dll into the 
  registry.
- Enter the following details (as specified here) in the proxyreg file, which opens in the 
  command prompt:
  - Enter the dll path : <Agent Toolkit Home>/dll/proxy/apadll.dll and press Enter
• Enter the conf file path: <Agent Toolkit Home>/dll/proxy/snmp.conf and press Enter.

• Instead of entering the details on prompt messages, the dll path and conf details can be given in the command line directly. This is possible by giving a space after proxyreg and entering the dll and conf path. Thus, the proxyreg directly starts with the required dll and conf path.

• The snmp.conf file in <Agent Toolkit Home>/dll/proxy directory contains details about the port and the registered OID.

• To start the registry, use regedit in Start -> Run which starts the registry of the Windows.

• Now, start the command line agent (AdventNet agent) by executing the run.bat/run.sh file in the output directory.

• The agent must be started at the port specified in the snmp.conf file. By default, the port is set to 8001.

• Stop the SNMP service by selecting the Settings->Control Panel->Services option in Windows NT and restart it, which will invoke the apadll.dll and will wait for the PDUs (No JVM invocation is done here).

• To test the same, start the MibBrowser application and load the MIB with which the Subagent was created.

• Go to MibBrowser Settings and change the SNMP Version to Snmp_Version1 and port to 161 as Windows Master agent by default runs at port 8001.

• Do a GET request to the OID specified in the snmp.conf file.

• You will receive the response without any problem.

13.11.2.1 Sending Traps Through Windows Master Agent

Trap generation support is also provided when the AdventNet agent is made to act as Windows Subagent. To avail Trap generation support here:

• Run the proxyReg.exe file from the directory <Agent Toolkit Home>/subagents/NTutils.

• When started, it will prompt the following:
  o Enter the dll path : Enter the absolute path of apadll.dll file bundled with this example.
  o Enter the conf file path: Enter the absolute path of the snmp.conf file bundled with this example.

• The following details should have been edited in the snmp.conf file before running the proxyReg.exe :

  TRAP_RECEIVER(Y/N)=N : Make this as Y (Yes) to receive traps.

  TRAP_RECEIVER_PORT= 8004. Change the port according to the requirement. The Master agent (Windows agent) will listen for traps in this port.

Setting the Properties of the SNMP Service

• Click the Start Button.

• Go to Control Panel of the Settings Menu.

• Double-click the Network icon in the Control Panel. Network Dialog will be opened.

• Select the Services tabbed pane from the Network Dialog.

• From the Network Services, double-click the SNMP Services. SNMP Properties dialog will be opened.

• Select the traps tabbed pane from the SNMP Properties dialog.
Stopping and Restarting the SNMP Service

- Click the Start Button.
- Go to Control Panel of the Settings Menu.
- Double-click the Services icon in the Control panel. Services dialog will be opened.
- Select SNMP from the Services. Click the Stop button.
- Select SNMP from the Services. Click the Start button.
- Start the Trap Browser of AdventNet Mib Browser and listen to Trap at the port 162 for public community.
- Run the agent by executing the file run.bat.
- The Startup(Cold Start Trap) trap sent by the agent will be received in the Trap Browser at port 162.
- Do an SNMP set operation on the node agentNetstat of the utilities group (Mib: AGENT-SAMPLE-MIB). The Trap sent can be seen received in the Trap Browser at port 162.

13.11.3 Configuring AdventNet Agent as Subagent to Linux Agent

Generate an agent for AGENT-SAMPLE-MIB using Mib Compiler.

Compile the agent.

Start the AdventNet agent in port 8001.

Add a proxy entry in the ucd-snmp agent in the file /etc/snmp/snmpd.conf as

```
proxy -c public -v 1 -p 8001 localhost .1.3.6.1.4.1.2162
```

If the UCD-SNMP Version is 5.0 and above then the entry in the conf file should be added as

```
proxy -c public -v 1 127.0.0.1:8001 .1.3.6.1.4.1.2162
```

After adding the above entry in snmpd.conf file, restart the ucd-snmp agent.

Open MibBrowser and load AGENT-SAMPLE-MIB.

Access the subagent information browsing through the master agent, i.e., sending the request to the master agent.

This proxy feature is supported by 4.2.1 and higher versions of UCD-SNMP. We support Redhat Linux 7.2 that provides 4.2.1 UCD SNMP. We also support Red Hat Linux Advanced Server release 2.1AS/i686 to enable this feature.

13.11.4 MIB-II Support Using Native Agent as Subagent

All Operating Systems support MIB-II. Because our agent architecture support makes other third-party agents as Subagent to AdventNet Master agent, MIB-II support is not implemented. By making the Enterprise agent built using Agent Toolkit as Master agent and Native OS agent as Subagent, MIB-II support will be available through the Enterprise agent built using Agent Toolkit.

13.12 Running the Example

The example present under <Agent Toolkit Home>/examples/snmp/proxy directory will help you test the following functionalities.

- staticproxy
- oid-based proxy
- context-based proxy
- instance-based proxy
- heartbeat
- subagent

Have a look at the readme.html for running the example.
14.0 Configuring SNMPv3 Agents

14.1 Overview

The version 3 of Simple Network Management Protocol addresses some of the long pending issues related to the large scale deployment of SNMP. Due to lack of security in using SNMP, system and network administrators were using other means such as telnet, ascii, etc., for configuration, accounting, and fault management. The primary goal of SNMP version 3 (SNMPv3) is to define a secure version of the SNMP. SNMPv3 also facilitates remote configuration of the SNMP entities, which make remote administration of SNMP entities a much simpler task.

AdventNet has implemented SNMPv3 as defined from RFC2570 to RFC2576.

14.2 Security Levels in SNMPv3

As explained earlier, SNMP version 3 (SNMPv3) is used to provide a secured environment in managing the systems and networks. The SNMPv3 Agent supports the following set of security levels as defined in the SNMP-FRAMEWORK-MIB (RFC 2571):

- **noAuthNoPriv** - Communication without authentication and privacy.
- **authNoPriv** - Communication with authentication and without privacy. The protocols used for Authentication are MD5 and SHA (Secure Hash Algorithm).
- **authPriv** - Communication with authentication and privacy. The protocols used for Authentication are MD5 and SHA and for Privacy, DES (Data Encryption Standard) protocol is used. For Privacy Support, you have to install some privacy packages. Please refer the topic "Supported Privacy Packages" for more details.

A framework for definition of different authentication and privacy protocols is available in V3. Currently, the MD5 and SHA authentication protocols and the CBC_DES privacy protocol are supported in USM.

14.2.1 Supported Privacy Packages

For privacy support, the Encryption packages that can be used are "Cryptix" and "JCE".

**14.2.1.1 To make use of JCE classes**

- Download JCE classes 1.2 or 1.2.1 from the following URL: http://java.sun.com/products/jce/
- In case JCE 1.2 classes are downloaded, you get the following jar: jce12-rc1-dom.jar
- In case JCE 1.2.1 classes are downloaded, you get the following four jars: jce1_2_1.jar; local_policy.jar; sunjce_provider.jar, and US_export_policy.jar
• Make sure the jars are placed under `<Agent Toolkit Home>` directory.

• Also make sure the jars are included in the setenv.bat file CLASSPATH (available in `<Agent Toolkit Home>/bin directory`) in the beginning. Please note that the jars are required to be in the CLASSPATH settings of run.bat/sh file, that are used for running the Agent.

• Edit the `java.security` file present in the `jre/lib/security` folder under the JDK installed in your machine. And add the following piece of line below:

```
security.provider.1=sun.security.provider.Sun
```

• Save the `java.security` file.

• The `USMUtils.class` required for encrypting v3 requests and responses is available in `AdventNetSnmp.jar` (<Agent Toolkit Home>/jars directory).

• Now, the v3Agent is ready for supporting Privacy.

```
security.provider.2=com.sun.crypto.provider.SunJCE
```

Note: Note : If JDK 1.4 is used, then JCE privacy jars are not required to be in the class path.

14.2.1.2 To make use of Cryptix classes

• Download Cryptix classes 3.1 or 3.2 from the following URL: `http://www.cryptix.org/`

• Make sure the jars are included in the `setenv.bat` file CLASSPATH (available in `<Agent Toolkit Home>/bin directory`) in the beginning. Please note that the jars are required to be in the CLASSPATH settings of run.bat/sh file, that are used for running the Agent.

• The `USMUtils.class` required for encrypting v3 requests and responses is available in `AdventNetSnmp.jar` (<Agent Toolkit Home>/jars directory).

• Edit the `java.security` file present in the `jre/lib/security` folder under the JDK installed in your machine. And add the following piece of line below:

```
security.provider.3=cryptix.provider.Cryptix
```

• Now, the v3 Agent is ready for supporting Privacy.

14.2.2 Export Restrictions

Encryption packages are bound by Export restrictions.

• If JCE 1.2 or its implementations are used in developing application and applets, they cannot be used outside US and Canada.

• JCE 1.2.1 does not have any export restrictions and it can be used in applications, which can be distributed throughout the world.

• The latest `JDK version` (JDK 1.4 beta) comes integrated with the JCE 1.2.1.

• Cryptix package does not have any such export restrictions.

14.3 Default Users of SNMPv3 Agents

By default, the SNMPv3 Agent provides support for three level of users, namely:

- **noAuthUser** - Users with security level noAuthNoPriv
- **authUser** - Users with security level authNoPriv
- **privUser** - Users with security level authPriv.

The details about the users get stored in the XML or Serialized Files or Runtime Memory depending upon the type of storage option chosen.
14.4 Developing a Sample V3 Agent

Please follow the steps given below to develop a Sample v3 Agent.

1. Start the MIB Compiler application from `<Agent Toolkit Home>/bin directory.
2. Create a New Project with the name say : snmpproject01.
3. Load any MIB from the MIBs directory say : AGENT-SAMPLE-MIB
4. By default, the version of the Agent will be specified as V3 in the Project -> Settings menu.
5. The Storage Type option chosen in the Project -> Settings menu -> SNMPv3 Panel -> USM and VACM group will be XML. Keep the default settings.
6. Generate code for the Agent. XML Files having the details of the v3 users will get generated under agent/bin/conf directory.
7. On successful generation, compile the generated code.
8. On successful compilation, a Sample V3 Agent is created.

14.5 Testing the SNMPv3 Agent With Default Users

Now that a Sample Agent is created, it has to be tested with the Default Users.

14.5.1 Testing the V3 Agent for noAuth Users

The default entry of noAuthUser in USM Table will be as follows.

<table>
<thead>
<tr>
<th>Context Name</th>
<th>Security Level</th>
<th>User Name</th>
<th>Auth Protocol</th>
<th>Priv Protocol</th>
<th>Auth Password</th>
<th>Priv Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>noAuth</td>
<td>noAuthNoPriv</td>
<td>noAuthUser</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

To test the Agent for noAuthUser,

1. Make sure the Agent is started from `<Agent Toolkit Home>/snmpprojects/snmpproject01/agent/bin` directory using `run.bat` or `.sh` file.
2. Start the MIB Browser application from `<Agent Toolkit Home>/bin directory.
3. Load AGENT-SAMPLE-MIB.
4. Click MIB Browser Settings in the toolbar icon.
5. A wizard opens up wherein you have to choose the version of the Manager. Choose v3.
6. Click Add in this wizard.
7. A Snmp Parameter Panel appears wherein the following details need to be filled:
   - Target Host : localhost (by default)
   - Target Port : 8001.
   - User Name : noAuthUser
   - Security Level : noAuth,noPriv
   - Context Name : noAuth
8. Click OK.
9. The entry gets listed in v3 Settings.
10. Select the entry and click OK to close the MIB Browser Settings wizard.
11. Move on to the MIB Browser Main UI.
12. Now, test the SNMPv3 Agent by sending a query (say GET request) to the scalar variable agentDescr in the `agentSystem group` under the `demo group`. You will receive the response as

```
Sent get request to localhost : 8001
agentDescr.0:-->agentDescr not initialized
```
14.5.2 Testing the V3 Agent for authUsers

Default entry of authUser in USM Table

<table>
<thead>
<tr>
<th>Context</th>
<th>Security Level</th>
<th>User Name</th>
<th>Auth Protocol</th>
<th>Priv Protocol</th>
<th>Auth Password</th>
<th>Priv Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>auth</td>
<td>authNoPriv</td>
<td>authUser</td>
<td>MD5</td>
<td>-</td>
<td>authUser</td>
<td>-</td>
</tr>
</tbody>
</table>

To test the Agent for authUser,

1. Follow the steps given above (in Testing the Agent for noAuthUser) till setting the Target Port as 8001.
2. Later, specify the User Name as authUser for this case.
3. Select the Security Level as Auth,noPriv from the combo box.
4. Specify the Authentication password as authUser.
5. You can see the entry added in the Table. Select the entry and click OK to close the MIB Browser Settings wizard.
6. Move on to the MIB Browser Main UI.
7. Now, test the SNMPv3 Agent by sending a query (say GET request) to the agentDisk group under the demo group.
8. You will receive the response. If you try accessing the agentDescr of agentSystem group, you will not be able to access it as authUsers are not given View Access to that particular scalar variable. To know more on View-based Access, please refer to VACM.

14.5.3 Testing the Agent for Privacy Users

Default entry of privUser in USM Table

<table>
<thead>
<tr>
<th>Context</th>
<th>Security Level</th>
<th>User Name</th>
<th>Auth Protocol</th>
<th>Priv Protocol</th>
<th>Auth Password</th>
<th>Priv Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>priv</td>
<td>Auth,Priv</td>
<td>privUser</td>
<td>MD5</td>
<td>SHA</td>
<td>privUser</td>
<td>privUser</td>
</tr>
</tbody>
</table>

To test the Agent for privUser,

1. It is required to install all the jars necessary for Privacy support. Please follow the directions specified in Supported Privacy Packages.
2. Then, follow the steps given in Testing the v3Agent for noAuthUsers till setting the Target Port as 8001.
3. Later, specify the User Name as privUser.
4. Select the Security Level as Auth,Priv from the combo box.
5. Specify the Authentication password as authUser and Privacy password as privUser.
6. Click OK.
7. You can see the entry added in the Table. Select the entry and click OK to close the MIB Browser Settings wizard.
8. Move on to the MIB Browser Main UI.
9. Now, test the SNMPv3 Agent by sending a query to the agentDisk group under the demo group. You will receive the response without any problem.

There is also an example available in <Agent Toolkit Home>/examples/snmp/snmpv3/simpleagent directory which will give you a clear idea of a Simple v3 Agent. Please refer readme.html for more information.
14.6 Adding More Users for v3 Agents (USM)

User-based Security Model (USM) is a default security model defined by SNMPv3. It provides different types of security levels using various authentication and privacy protocols as explained earlier in this section. To add more users for accessing v3 Agents, AdventNet provides a table called the USMTable. User entries can be added to the Table either: (1) Before Agent Startup or (2) During Run Time.

Please note that though user entries are added these users will not be able to access the Agent unless View Access is given to them. Please refer to "Enabling Authorization in V3 using VACM" section for more details.

14.6.1 Before Agent Startup

Entries can be added to the USMTable before Agent Startup using any of the following options: (1) Using MIB Compiler UI or (2) Using XML/Ser Files/Runtime Memory or (3) Using API calls.

Using MIB Compiler UI
1. Create a Project and load a MIB.
2. Choose Project -> Settings menu from the menu bar of MIB Compiler UI.
3. Select usmUserTable in the SNMPv3 Panel.
4. Click Add.
5. A wizard pops up wherein you can specify the user entries.
6. Click OK. A new USM User entry gets added.

Using XML/Ser File/Runtime Memory

The entries configured through Mib Compiler UI get stored in the configuration file "UsmUserTable.xml" after code generation. They get stored under <Agent Toolkit Home>/snmpprojects/projectname/agent/bin/conf directory, provided the type of storage is chosen. To choose the Storage Type,
1. Choose Project -> Settings menu of MIB Compiler UI.
2. Select usmUserTable from the SNMPv3 Panel -> USM Group.
3. Choose XML or Serialization or Runtime Memory from the Storage Option. By default XML File is chosen.

This XML file can be edited to add new entries. Please note that the Agent has to be restarted for the changes to take effect. The serialized formatted file cannot be edited to add new entries. They can be updated only by adding entries during runtime.

Run time memory can be used to store the v3 users information in the Agent Memory itself. Using this option will not store the entries in ser files or xml files. After choosing the Runtime memory storage option, follow the steps given in adding entries "From the Manager" given under the heading "Adding Entries During Runtime"(14.6.2). Please note that once the Agent is killed, the entries added are removed from the memory.

Using API Calls

Add the following piece of code in the Main file generated in the initializeV3Settings() method in case you want to add the entries using API calls. The following is a sample entry for adding a newUser entry.

```
usmUserTableListener.createAndAddUSMUserEntry("newUser", new Integer(0), "0.0", new Integer(com.adventnet.snmp.snmp2.usm.USMUserEntry.NO_AUTH),
        new Integer(com.adventnet.snmp.snmp2.usm.USMUserEntry.NO_PRIV),
        "userPublic", " ", " ", new Integer(3), new Integer(1));
```
14.6.2 During Runtime

Entries can be added during run time from the Manager.

From the Manager - Using a Command line Utility

To add entries from the Manager, it is important to enable the Remote Configuration option in "Project -> Settings menu -> SNMPv3 panel". Else, it is not possible to access the Table from remote.

- After enabling the Remote Configuration option (as said earlier), make use of the command line tool "snmpUSMRemoteConfigure.java" available in <Agent Toolkit Home>/examples/snmp/low_level_udpapps directory.
- Go to run.bat or run.sh from the command prompt (available in the above specified directory) and specify the following options. An user entry will be added to the USM Table using these options.

**Usage**:

```
```

- For better understanding, please refer to the sample entry given below:

```
<AdventNet\JavaAgent>\examples\snmp\low_level_udpapps>run
snmpUSMRemoteConfigure -d -p 8001 -a MD5 -w authUser -n auth -y xxxUser authUser xxxUser localhost
```

- Here,
  - -d : dumps the SNMP message.
  - -p 8001 : represents the Agent Port number.
  - -a MD5 : represents the admin user's (template user) and new user's authProtocol.
  - -w authUser : represents the admin user's authentication password.
  - -n auth : represents the auth user's context name.
  - -y xxxUser : represents the new user's authentication password.
  - authUser : represents the existing template's user name.
  - xxxUser: represents the new user's name.
  - localhost : represents the host in which the Agent is running.

- If the user is added to the USMTable successfully "User Successfully cloned!!" can be seen in the command prompt. Else an error will be displayed.

14.6.2.1 Changing the Password of an Existing User

The Manager can also change the password of an existing user, using the utility called "snmpUSMKeyChange.java" available in <Agent Toolkit Home>/examples/snmp/low_level_udpapps directory.

The usage of this utility is as follows:

**Usage**:

```
```

AdventNet Inc. 106
The password can be changed in the following ways:

- Own AuthKey change.
- AuthKey Change on behalf of another user.
- Own PrivKey change.
- PrivKey Change on behalf of another user.

The following is an example for Own AuthKey change. Assuming that View access is given for the user, options are explained.

```
run snmpUSMKeyChange -d -p 8001 -a MD5 -w authUser -n auth -y authUserNew authUser localhost
```

Here,

- `-d`: dumps the SNMPmessage.
- `-p 8001`: represents Agent port number.
- `-a MD5`: represents adminUser's authProtocol.
- `-w authUser`: represents the adminUser's authentication password
- `-n auth`: represents authUser's context name.
- `-y authUserNew`: represents the new authentication password of the user authUser.
- `authUser`: represents the user whose authentication password will be changed.
- `localhost`: represents the host where the agent is running.

The above inputs will modify the authentication password of the user "authUser" from "authUser" to "authUserNew". The user's authentication protocol is "MD5".

### 14.7 Storing Details of V3 Users in Your Own Model

SNMP V3 agent, by default, gets started reading the USM and VACM configurations in the snmpv3.ser file, a serialized file in an unreadable format. In case you want the USM and VACM details to be stored elsewhere, say for example in a database format, and if you expect the v3 agent to read the configurations from that database then you will have to implement certain interfaces provided by Agent Toolkit. These interfaces will implement your methods based on your application. On initializing these interfaces, the V3 details will be read from that method.

#### 14.7.1 Interfaces Used

The two interfaces used for this feature are:

- V3Initialiser Interface and
- RemoteV3Client Interface

##### 14.7.1.1 V3Initialiser Interface

An example class is provided here as a reference. The following line in the main file registers a V3InitialiserImpl to the SnmpAgent.

```
super.addV3Initialiser((V3Initialiser)(
    new V3InitialiserImpl(this)));
```

This class contains API calls to the various V3 tables supported by the Agent. When the Agent is started, the initializing methods in this V3Initialiser class recalled with the respective table given as the parameter. For example, the initialiseUsmUserTable() is called, passing down the instance of the USMTable used by the SnmpAgent.
It is your responsibility to fill this table with the proper values for which the Agent will authenticate the requests.

Say for example, the following code adds an USMUser with the NO_AUTH_NO_PRIV security level to the USMTable.

```java
byte[] engineId = SnmpAgent.getSnmpAPI().getSnmpEngineID();
USMUserEntry noAuthUser = new USMUserEntry (new
String("noAuthUser").getBytes(), engineId);
noAuthUser.setSecurityLevel(Snmp3Message.NO_AUTH_NO_PRIV);
USMTable.addEntry(noAuthUser);
```

The above code adds an USM User with the following parameters.
- **UserName**: noAuthUser
- **SecurityLevel**: NO_AUTH_NO_PRIV

Similarly, VACM Group table, VACM Access Table and VACM Family Table are populated. The other tables like TargetAddrExtTable, TargetParamsTable, NotifyTable, NotifyFilterTable, NotifyFilterProfileTable are also to be initialized when implementing V3 Initializer.

### 14.7.1.2 RemoteV3Client Interface

Similar to the V3Initialiser Interface, this interface is implemented and registered to the SNMP Agent through the method addRemoteV3Client. When any addition, deletion, or modification of entry needs to be made in the USM / VACM tables, the RemoteV3Client Interface is used. The methods to add, delete, or modify the entries can be implemented inside the interfaces based on your application.

For example, the following method gets called when there is any addition in the USM table.

```java
public void usmUserEntryAdded(USMUserTable usmTable, USMUserEntry entry){
    usmTable.serialize(outMain);
}
```

In this example, the Agent is made to apply the modification in the serialized file itself. But in normal circumstances, you will be storing the details elsewhere.

### 14.7.2 Running the Example

To know more on v3 user storage model, make use of the example present in `<Agent Toolkit Home>/examples/snmp/snmpv3/remotev3initialiser` directory. Also, refer to the `readme.html` which helps in running the example.

### 14.8 Enabling Authorization in V3 Using VACM

View-based Access Control Model (VACM) is a default access control model defined by SNMPV3 framework (RFC 2575). It is possible to restrict a particular group in accessing an OID in the MIB using VACM. SNMPv3 Agent has implemented the VACM MIB as a default access control model.

#### 14.8.1 Details for VACM

The details for View-based Access are to be specified in the four tables of VACM MIB namely:
- **vacmContextTable** - This table will have a set of context names supported by the SNMPv3 Agent. The context name received will be checked with this table in the access validation phase. It is not configurable through SNMP.
- **vacmSecurityToGroupTable** - This table will have a set of security to group mappings. If the received context name is valid then the group name is obtained from this table by giving user (security) name and security model as an input. It is configurable through SNMP.

- **vacmAccessTable** - This table will have a set of access supported by the Agent. By giving group name, context name, security model, and security level, you can get a view name based on the received request type. It is configurable through SNMP.

- **vacmViewTreeFamilyTable** - This table will have a set of views supported by the Agent. By giving view name and received OID, you can specify whether the received request has valid view or not. It is configurable through SNMP.

### 14.8.2 Adding Entries to VACM Tables

An user can be given view access to a managed node by specifying their views in the VACM Tables. Entries can be added to the VACM Tables either : (1) Before Agent Startup or (2) During Run Time. Please note that only when all the vacm tables are configured for an user will the user entry have view access. Lets have a look at the default entries in VACM Tables and move on to the steps involved in adding view access to users.

#### 14.8.2.1 Default VACM Entries

**VacmContextTable**

Context Name - noAuth, auth, priv

**Vacm Security to Group Table**

<table>
<thead>
<tr>
<th>Model</th>
<th>Security Name</th>
<th>Group Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>USM (3)</td>
<td>noAuthUser</td>
<td>noAuthGroup</td>
</tr>
<tr>
<td>USM (3)</td>
<td>authUser</td>
<td>authGroup</td>
</tr>
<tr>
<td>USM (3)</td>
<td>privUser</td>
<td>privGroup</td>
</tr>
<tr>
<td>1</td>
<td>noAuthUser</td>
<td>noAuthGroup</td>
</tr>
<tr>
<td>2</td>
<td>noAuthUser</td>
<td>noAuthGroup</td>
</tr>
</tbody>
</table>

**Vacm Group Access Table**

<table>
<thead>
<tr>
<th>GroupName</th>
<th>Prefix</th>
<th>Model</th>
<th>Level</th>
<th>Match</th>
<th>Read</th>
<th>Notify</th>
<th>Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>noAuthGroup</td>
<td>noAuth</td>
<td>USM(3)</td>
<td>noAuth,noPriv</td>
<td>Exact</td>
<td>noAuthView</td>
<td>noAuthView</td>
<td>noAuthView</td>
</tr>
<tr>
<td>authGroup</td>
<td>auth</td>
<td>USM(3)</td>
<td>Auth,noPriv</td>
<td>Exact</td>
<td>authView</td>
<td>authView</td>
<td>authView</td>
</tr>
<tr>
<td>privGroup</td>
<td>priv</td>
<td>USM(3)</td>
<td>Auth,Priv</td>
<td>Exact</td>
<td>privView</td>
<td>privView</td>
<td>privView</td>
</tr>
<tr>
<td>noAuthGroup</td>
<td>noAuth</td>
<td>2</td>
<td>noAuth,noPriv</td>
<td>Exact</td>
<td>noAuthView</td>
<td>noAuthView</td>
<td>noAuthView</td>
</tr>
<tr>
<td>noAuthGroup</td>
<td>noAuth</td>
<td>1</td>
<td>noAuth,noPriv</td>
<td>Exact</td>
<td>noAuthView</td>
<td>noAuthView</td>
<td>noAuthView</td>
</tr>
</tbody>
</table>

**Vacm View Tree Family**

<table>
<thead>
<tr>
<th>View Name</th>
<th>Mask</th>
<th>Type</th>
<th>Sub Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>noAuthView</td>
<td>ff</td>
<td>included</td>
<td>1.3.6</td>
</tr>
<tr>
<td>authView</td>
<td>ff</td>
<td>included</td>
<td>1.3.6</td>
</tr>
<tr>
<td>privView</td>
<td>ff</td>
<td>included</td>
<td>1.3.6</td>
</tr>
</tbody>
</table>
14.8.2.2 Adding Entries before Agent Startup

Entries can be added to the tables before Agent Startup using (1) MIB Compiler UI option or (2) Using XML/Ser files/Runtime Memory or (3) Using API calls.

Using MIB Compiler UI
1. Create a Project and load a MIB.
2. Choose Project -> Settings menu from the menu bar of MIB Compiler UI.
3. Select vacmContextTable in the SNMPv3->VACM Panel.
4. Now, Click Add.
5. A wizard pops up wherein you can specify the user entries.
6. Click OK.

Similarly other VACM Tables have to be populated.

Using XML/Ser File/Runtime Memory

The entries configured through MibCompiler UI get stored in the configuration file, VacmContextTable.xml or VacmContextTable.ser (name of the table varies according to the table chosen) under <Agent Toolkit Home>/snmpprojects/projectname/agent/bin/conf directory, provided the storage type is chosen. To choose the type of storage,
1. Choose Project -> Settings menu from the menu bar of MIB Compiler UI.
2. Select a table from the VACM group of SNMPv3 Panel.
3. Choose XML File or Serialization format or Runtime Memory from the Storage Type Options provided. By default XML File is chosen.
4. Choosing this would generate files in xml or serialized for each table in the VACM group.

These XML files can be edited to add new entries. Please note that the Agent has to be restarted for the changes to take effect. The serialized formatted file cannot be edited to add new entries. They can be updated only by adding entries during runtime.

Run time memory can be used to store the v3 users information in the Agent Memory itself. Using this option will not store the entries in ser files or xml files. After choosing this storage option, follow the steps given in adding entries "From the Manager" given under the heading "Adding Entries During Runtime". Please note that once the Agent is killed, the entries added are removed from the memory.

Using API calls

These code snippets have to be added in the main file generated in the initializeV3Settings() method. Adding the piece of code will add new entries in the vacm tables.

Adding entries to the vacmContextTable

```java
vacmContextTableListener.createAndAddVacmContextEntry("newNoAuth");
```

Adding entries to the vacmSecurityToGroupTable

```java
vacmSecurityToGroupTableListener.createAndAddVacmGroupEntry(new Integer(3), "newNoAuthUser", "newNoAuthGrp", new Integer(3), new Integer(1));
```
Adding entries to the `vacmAccessTable`

```java
vacmAccessTableListener.createAndAddVacmAccessEntry("authGrp", "newNoAuth", new Integer(3), new Integer(0), new Integer(1), "newNoAuthView", "newNoAuthView", "newNoAuthView", new Integer(3), new Integer(1));
```

Adding entries to the `vacmViewTreeFamilyTable`

```java
vacmviewTreeFamilyTableListener.createAndAddVacmFamilyEntry("newNoAuthView", ".1.3.6", "ff", new Integer(1), new Integer(3), new Integer(1));
```

### 14.8.2.3 Adding entries during Run Time

You can also add entries to the tables during run time.

#### From the Manager

To configure the Agent from the Manager, follow the steps given below:

- Enable the Remote Configuration option in **Project -> Settings** menu -> SNPV3 -> VACM Panel. If this option is disabled, then entries cannot be added to the VACM table from remote.
- Then, start the MIB Browser application.
- Load **SNMP-VIEW-BASED-ACM MIB** from `<Agent Toolkit Home>/mibs` directory.
- This MIB contains four tables in which the View-based Access control has to be configured.
- Selecting the respective table and clicking SNMP Table icon in MIB Browser open up a wizard wherein entries can be added to the required Tables by sending SET requests.
- Please note that vacm context table is not configurable from the Manager (remotely).

**Note:** The User-based Security Model (USM) and View-based Access Control (VACM) Tables are implemented by default when the Agent is started as SNPV3 Agent.

### 14.9 Using USM Without VACM

To make use of USM without VACM, follow the steps given below:

You have to include the following statement in the generated Main file after the code for restarting the Agent. The statement actually turns off the VACM check, that is, the VACM check always returns true.

```java
super.getSnmpVacm().setAcmUsed(false);
```

### 14.10 Authenticating Requests From v1/v2c Managers (Coexistence Support)

In a typical deployment scenario, the management applications and applets will be required to communicate with SNMP Agents of different versions. They will also be required to communicate with multilingual agents, i.e., SNMP Agents that support all the three SNPV versions (v1, v2c and v3).

The multilingual SNMP Agents support multiple SNMP message versions and coexist with entities which support only a single SNMP message version. So, management applications with SNPV1 or v2c support can also communicate with SNPV3 agents.
This is called as coexistence in v3 as defined in RFC 2576. SNMPv3 Agent entities with coexistence support implement the **SNMP-COMMUNITY-MIB**. This MIB contains objects for mapping between community strings and version-independent SNMP message parameters. Apart from this, a complete implementation of Coexistence Support require the implementation of **SNMP-TARGET-MIB**. The implementation of this MIB facilitates the source address validation on the incoming requests.

In case the Agent is strictly v3, it will drop the requests sent from v1/v2c Managers. To know how to make an Agent strictly v3, please refer to Making the Agent Strictly V3" (14.11) topic.

### 14.10.1 Enabling Co-existence Support

You can enable Coexistence Support either through MIB Compiler UI or through API Calls.

**Using MIB Compiler UI**

- Create a Project and Load a MIB.
- Select **Project -> Settings** menu from the menu bar of MIB Compiler UI.
- From the General Panel, check **Target, Community and Notification MIB Support** option.
- This will enable **Coexistence Support and Notification Filtering Support** (Notification Support is explained under "Sending Notifications" topic in this section).

**Using API calls**

On enabling the option in MibCompiler UI, the following code gets generated in the Main file. Without enabling the option in UI, you can enable Coexistence by adding the following line of code before the code for restarting SNMP Agent.

```java
super.getSnmpAPI().setCommunityAuthentication(true);
```

Once the community Authentication is set as true, the Agent is ready to authenticate requests from all Managers. Thus Co-Existence is enabled.

**Note:** To enable Coexistence Support and add a new entry to the Table, the following import statement has to be included in the Main file.

### 14.10.2 Default Users of Coexistence Support

The Tables of Community MIB and Target MIB used for Coexistence support include SnmpCommunityTable, SnmpTargetAddrTable, SnmpTargetParamsTable, and SnmpTargetAddrExtTable. The configurations present by default in these Community tables are as follows:

#### SnmpCommunityTable

<table>
<thead>
<tr>
<th>SnmpCommunity Index</th>
<th>SnmpCommunity Name</th>
<th>SnmpCommunitySecurityName</th>
<th>SnmpCommunityContextEngineID</th>
<th>SnmpCommunityContextName</th>
<th>SnmpCommunityTransportTag</th>
<th>SnmpCommunityStorageType</th>
<th>SnmpCommunityStatus</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>public</td>
<td>noAuthUser</td>
<td>127.0.0.18003</td>
<td>noAuth</td>
<td>public</td>
<td>nonVolatile(3)</td>
<td>Active(1)</td>
</tr>
</tbody>
</table>

#### SnmpTargetAddrTable

<table>
<thead>
<tr>
<th>SnmpTargetAddrName</th>
<th>SnmpTargetAddrDomain</th>
<th>SnmpTargetAddrTimeOut</th>
<th>SnmpTargetAddrRetryCount</th>
<th>SnmpTargetAddrTagList</th>
<th>SnmpTargetAddrParams</th>
<th>SnmpTargetAddrStorageType</th>
<th>SnmpTargetAddrRowStatus</th>
</tr>
</thead>
<tbody>
<tr>
<td>localhost</td>
<td>1.3.6.1.6.1.1</td>
<td>127.0.0.18003</td>
<td>5</td>
<td>1</td>
<td>public</td>
<td>advent</td>
<td>nonVolatile(3)</td>
</tr>
</tbody>
</table>
14.10.3 Adding Managers for Supporting Coexistence

Manager entries can be added to the Community Tables either: (1) Before Agent Startup or (2) During Run time.

Before Agent Startup

Entries can be added to the Target and Community tables either using the option in **MIB Compiler UI** or using **API calls** or using the **XML files/Serialized file/Runtime memory** storage options.

Using MIB Compiler UI

1. Create a Project and load a MIB.
2. Choose Project -> Settings menu from the menu bar of MIB Compiler UI.
3. Select a table under the Community or Target group in SNMPv3 Panel.
4. Now, Click Add.
5. A wizard pops up wherein you can specify the user entries for each table.
6. Click OK. The entries are added.
7. Following the same steps add entries in all the Community and Target Tables.

Using XML/Ser File/Runtime Memory

The entries configured through Mib Compiler UI get stored in the configuration file, **USMTable.xml** under `<Agent Toolkit Home>/snmpprojects/projectname/agent/bin/conf` directory, provided the storage type is chosen. For this purpose,

1. Choose Project -> Settings menu from the menu bar of MIB Compiler UI.
2. Select Community Table/ Target Tables from the SNMPv3 Panel.
3. Choose XML or Serialized or Runtime Memory from the Storage Option. By default xml is chosen.

Please note that the Agent has to be re-started for the changes to take effect.

Run time memory can be used to store the v3 users information in the Agent Memory itself. Using this option will not store the entries in ser files or xml files. After choosing the storage option, follow the steps given in adding entries "From the Manager" given under the heading "Adding Entries During Runtime". Please note that once the Agent is killed, the entries added are removed from the memory.

Using API Calls

The code snippet required for configuring the tables for Coexistence support through API calls is provided below. Add these sample codes in the main file generated before the code for restarting the SNMP Agent.

```
snmpCommunityTableListener.createAndAddCommunityEntry("newpublic", "newpublic", "newUser", "127.0.0.18003", "newNoAuth", "newTag", new Integer(3), new Integer(1));
```
SnmpTargetAddrExtTable

```java
snmpTargetAddrExtTableListener.createAndAddSnmpTargetAddrExtEntry("localHost1", new Integer(484), "ff");
```

SnmpTargetAddrTable

```java
snmpTargetAddrTableListener.createAndAddSnmpTargetAddrEntry("localHost1", "1.3.6.1.6.1.1", "127.0.0.1#8003", new Integer(5), new Integer(1), "newTag", "addressParamsName", new Integer(3), new Integer(1));
```

During Run Time - From the Manager

To Configure the Agent from the Manager,

- Start the MIB Browser application.
- Load SNMP-COMMUNITY-MIB and SNMP-TARGET-MIB from `<Agent Toolkit Home>/mibs` directory.
- This SNMP-COMMUNITY-MIB contains the SnmpCommunityTable, and SnmpTargetAddrExtTable while SNMP-TARGET-MIB contains SnmpTargetAddrTable in which v3 user entries can be added for Community Authentication.
- Selecting the respective table and clicking SNMP Table icon in MIB Browser will open up a wizard wherein entries can be added to the required Tables by sending SET requests.

**Note:** By default Co-Existence and Notification Filtering is not available for a v3 Agent. The option for enabling this is available in the Project - > Settings menu bar of MIB Compiler UI.

14.11 Making the Agent Strictly SNMPv3

Please follow the steps given below to make an SNMPv3 Agent strictly V3:

After generating the Agent as V3, override the callback method of SNMP Agent by adding the following code in the Main File.

```java
/* User code starts here */
/**
 * Overriding the callback of the SnmpAgent.
 * This method will be called by the SnmpSession class. We check
 * for the Pdu's version and if it is not V3, drop the PDU.
 * else call the SnmpAgent's callback.
 * @param sess - the SnmpSession which handles the request.
 * @param pdu - the incoming request PDU
 * @param reqid - the unique identifier for the request PDU
 * maintained by
 * the SnmpSession
 * @return boolean indicating if the request is processed or not.
 */
public boolean callback(SnmpSession sess, SnmpPDU pdu, int reqid ){
    if(pdu.getVersion() < SnmpAPI.SNMP_VERSION_3)
    {
        System.out.println("Lower Version received ...dropping the PDU");
        return false;
    }
    else
    return super.callback(sess, pdu, reqid);
}
/* User code ends here */
```
14.12 Sending Notifications

SNMPv3 Notification PDU though similar to SNMPv2 Notification PDU, differs in the message format. It contains SNMPv3 message headers such as message ID, version, security level, maximum supported size, security model, etc., and security parameters depending upon the security model and scoped PDU parameters such as context name, context engine ID, varbinds, etc.

Notifications can be sent to both v3 Managers (using the Manager information in v3Trap ForwardingTable) and v1/v2c Managers (using Notification Filtering Mechanism). Please go through the following topics for more details.

14.12.1 Notifications To v3 Managers

SNMPv3 framework recommends SNMP-TARGET-MIB to identify the targets for sending Notifications, which is implementation specific. AdventNet Toolkit has a proprietary table called the V3 Trap Forwarding Table that contains all the information of the target. For more information about this table, please refer to Traps section. This section explains how Traps can be sent from v3 Agents to v3 Managers using v3 Trap Forwarding Table.

14.12.2 Notifications to v1/v2c Managers

The Trap Forwarding Table mentioned above is by default bilingual in nature. Hence can forward Notifications from a v3 Agent to v1/v2c Managers also. If the proprietary implementation is not required, then you can go for the Notification Filtering Mechanism support which is implemented as per RFC 2573.

14.12.2.1 Enabling Notification Filtering Support

Notification Filtering support can be enabled using MIB Compiler UI or using API calls.

Using MIB Compiler UI

- Create a Project and Load a MIB.
- Select Project -> Settings menu from the menu bar of MIB Compiler UI.
- From the General Panel, check Target, Community and Notification Support option.
- This will enable Coexistence Support and Notification Filtering Support. (Coexistence is explained under "Authenticating v1/v2c Requests" topic in this V3 section).

Using API Calls

Notification Filtering support can also be availed in AdventNet SnmpV3 Agent by adding the following piece of code in the main file before the code for restarting the Agent.

```java
super.getSnmpAPI().setNotificationFiltering(true);
```

**Note:** While enabling Notification Filtering Support through API calls and adding a new entry to the Tables, the following import statement has to be included in the Main file (in the beginning).

```java
import com.adventnet.snmp.snmp2.agent.notification.*;
```

Coexistence support should also be enabled to support Notification Filtering Mechanism. Hence, include the API calls and the imports for the same.
14.12.2.2 Default Users for Notification Filtering Support

The Tables of Notification MIB and Target MIB used for Filtering Mechanism support include SnmpTargetAddrTable, SnmpTargetParamsTable, SnmpNotifyTable, SnmpNotifyFilterProfileTable, and SnmpNotifyFilterTable. The default configurations present in these Notification Filtering Tables are as follows:

**SnmpTargetParamsTable**

<table>
<thead>
<tr>
<th>SnmpTargetParamsName</th>
<th>SnmpTargetParamsModel</th>
<th>SnmpTargetParamsSecurityModel</th>
<th>SnmpTargetParamsSecurityName</th>
<th>SnmpTargetParamsSecurityLevel</th>
<th>SnmpTargetParamsStorageType</th>
<th>SnmpTargetParamsRowStatus</th>
</tr>
</thead>
<tbody>
<tr>
<td>advent</td>
<td>3</td>
<td>3</td>
<td>noAuthUser</td>
<td>0</td>
<td>NonVolatile (3)</td>
<td>active</td>
</tr>
</tbody>
</table>

**SnmpNotifyTable**

<table>
<thead>
<tr>
<th>Snmp Notify Name</th>
<th>Snmp Notify Tag</th>
<th>Snmp Notify Type</th>
<th>Snmp Notify Storage Type</th>
<th>Snmp Notify RowStatus</th>
</tr>
</thead>
<tbody>
<tr>
<td>local host</td>
<td>public</td>
<td>1(trap)</td>
<td>Non Volatile (3)</td>
<td>Active</td>
</tr>
</tbody>
</table>

**SnmpNotify FilterProfileTable**

<table>
<thead>
<tr>
<th>SnmpTargetParamsName</th>
<th>SnmpNotifyFilterProfileName</th>
<th>SnmpNotifyFilterProfileStorageType</th>
<th>SnmpNotifyFilterProfileRowStatus</th>
</tr>
</thead>
<tbody>
<tr>
<td>advent</td>
<td>profile</td>
<td>Non Volatile (3)</td>
<td>Active (1)</td>
</tr>
</tbody>
</table>

**SnmpNotify FilterTable**

<table>
<thead>
<tr>
<th>SnmpNotifyFilterProfileName</th>
<th>SnmpNotifyFilterSubTree</th>
<th>SnmpNotifyFilterMask</th>
<th>SnmpNotifyFilterType</th>
<th>SnmpNotifyFilterStorageType</th>
<th>SnmpNotifyFilterRowStatus</th>
</tr>
</thead>
<tbody>
<tr>
<td>profile</td>
<td>.1.3.6</td>
<td>1</td>
<td>NonVolatile (3)</td>
<td>Active (1)</td>
<td></td>
</tr>
</tbody>
</table>

14.12.2.3 Adding Managers to the Notification Tables

After enabling Notification Filtering Support, it is required to specify the v1/v2c Managers to whom these Notifications should be sent. Manager Entries can be added to the Notification tables either: (1) Before Agent Startup or (2) During Run Time.

**Before Agent Startup -**

**Using MIB Compiler UI**

1. Create a Project and load a MIB.
2. Choose Project -> Settings menu from the menu bar of MIB Compiler UI.
3. Select a table from the Notification Group in SNMPv3 Panel.
4. Now, Click Add.
5. A wizard pops up wherein you can specify the user entries.
6. Click OK. Thus the entries get added to the table selected.
7. Following the same steps add entries to all the tables in the Notification group.

**Using XML File/Ser File/Runtime Memory**

The entries configured through Mib Compiler UI get stored in the configuration file, NotificationTable.xml under <Agent Toolkit>
Home>/snmpprojects/projectname/agent/bin/conf directory, provided the storage type is chosen. For this purpose,

1. Choose Project -> Settings menu from the menu bar of MIB Compiler UI.
2. Select each table in the SNMPv3 Panel -> Notification Group.
3. Choose XML or Serialized or Runtime Memory from the Storage Option. By default xml is chosen.
4. By choosing either of the option for each Notification related table, a file gets generated either in xml or serialized format. By default the storage type chosen is xml.

These XML files can be edited to add new entries. Please note that the Agent has to be re-started for the changes to take effect.

Run time memory can be used to store the v3 users information in the Agent Memory itself. Using this option will not store the entries in ser files or xml files. After choosing the storage option, follow the steps given in adding entries "From the Manager" given under the heading "Adding Entries During Runtime". Please note that once the Agent is killed, the entries added are removed from the memory.

Using API calls

To add the entries, follow the links and include the code (given under respective links) in the generated main file.

**SnmpTargetAddrTable**

```java
snmpTargetAddrTableListener.createAndAddSnmpTargetAddrEntry("localHost1", ".1.3.6.1.6.1.1", "127.0.0.1#8003", new Integer(5), new Integer(1), "newTag", "addressParamsName", new Integer(3), new Integer(1));
```

**SnmpTargetParamsTable**

```java
snmpTargetParamsTableListener.createAndAddSnmpTargetParamsEntry("advent1", new Integer(3), new Integer(3), "newUser", new Integer(0), new Integer(3), new Integer(1));
```

**SnmpNotifyTable**

```java
snmpNotifyTableListener.createAndAddSnmpNotifyEntry("localhost2", "public", new Integer(1), new Integer(3), new Integer(1));
```

**SnmpNotifyFilterTable**

```java
snmpNotifyFilterTableListener.createAndAddSnmpNotifyFilterEntry("advent1", ".1.3.6", "ff", new Integer(1), new Integer(3), new Integer(1));
```

**SnmpNotifyFilterProfileTable**

```java
snmpNotifyFilterProfileTableListener.createAndAddSnmpNotifyFilterProfileEntry("newProfile", new Integer(3), new Integer(1));
```

During Run Time - From the Manager

To configure the Agent from the Manager,

- Enable the **Remote Configuration** option in **Project -> Settings** menu -> **SNMPv3 -> Notification Panel**. If this option is not enabled, entries cannot be added to the Notification tables from remote.
• Then, start the MIB Browser Application.
• Load SNMP-TARGET-MIB and SNMP-NOTIFICATION-MIB from <Agent Toolkit Home>/mibs directory.
• These MIBs contain the SnmpTargetAddrTable, SnmpTargetParamsTable, SnmpNotifyTable, SnmpNotifyFilterTable, and SnmpNotifyFilterProfileTable in which Notification Filtering Support is to be configured.
• When you select the respective table and click SNMP Table icon (View SNMP Table Data icon), MIB browser opens up a wizard wherein entries can be added to the required Tables.
• Click OK and you can see the entries added to the Table.
• Check for the same by sending a GET request to the Table.

14.12.2.4 Testing Notification Filtering Support

Now that you have enabled this feature and added entries to the Notification Table, the next step is checking whether the Agent works with this support. To ensure the same, follow these steps:
• Start the MIB Browser application.
• Load the MIB with which v3 Agent is developed.
• Configure the MIB Browser settings. The Manager has to be of v1 or v2c type for testing purpose.
• Start the v3 Agent.
• Send a SET Request to a variable in the MIB for which Notification is defined.
• Values should be set and Notification should be generated.
• Notifications can be viewed in the Trap Viewer of MIB Browser application.

14.13 Making the v3Agent Compliant to SNMPv3 Standards

The SNMPv3 Agent can be made compliant to SNMPv3 Standards. This topic deals with Snmpv3 Compliance support provided by AdventNet Agent Toolkit.

14.13.1 Enabling SNMPv3 Compliance

By default, v3Compliance is enabled in the Agent. If version v3 is disabled, v3 compliance also gets disabled. Version v3 can be enabled or disabled in

```
Project -> Settings menu -> General Panel of MIB Compiler UI.
```

V3 compliance can also be enabled using API calls. The following piece of code gets generated in the Main file by default. By setting the value as true or false, v3 compliance is enabled or disabled respectively.

```
super.setSnmpV3Compliance(true);
```

'super' here represents SNMPAgent.

14.13.2 Supported MIBs

SNMPv3 Compliance supports the following RFCs along with their implementations.
• RFC 2571 - SNMP Framework MIB supporting snmpEngineID, snmpEngineBoots, snmpEngineTime, and snmpEngineMaxMessageSize of 'SnmpEngineGroup'.
- **RFC 2572** - SNMP Message Processing and Dispatching (MPD) MIB supporting `snmpUnknownSecurityModels`, `snmpInvalidMsgs`, and `snmpUnknownPDUHandlers` of 'SnmpMPDStats group'.

- **RFC 2573** - SNMP Target MIB and SNMP Notification MIB supporting `snmpTargetSpinLock`, `snmpUnavailableContexts`, and `snmpUnknownContexts` of 'SnmpTarget Objects Group'. The 'Snmp Target Tables' and 'Snmp Notify Tables' are implemented for Notification Filtering and Coexistence support.

- **RFC 2574** - SNMP User-based Security Model (USM) MIB supporting `usmStatsUnsupportedSecLevels`, `usmStatsNotInTimeWindows`, `usmStatsUnknownUserNames`, `usmStatsUnknownEngineIDs`, `usmStatsWrongDigests`, `usmStatsDecryptionErrors` of 'USMStats Group', and USM User Tables implemented for 'User-based Security Model'.

- **RFC 2575** - SNMP View-based Access Control Model (VACM) MIB supporting 'VACM Tables' implemented for View-based Access Control Model.

- **RFC 2576** - SNMP Coexistence between Version 1, Version 2, and Version 3 MIB supporting 'Snmp Community Tables' for Coexistence Support.

Please have a look at the standard RFC documents for more information.
15.0 SNMPv3 Administration Tool

15.1 Overview

This document will help you use the SNMP V3Admin tool which is used for configuring USM user tables and VACM tables for SNMPv3 agent at runtime.

The SNMP V3 administration tool allows you to add, modify, and delete rows in the User Security Model (USM) table and in the View-based Access Control Model (VACM) tables.

The SNMP V3Admin tool can be started in two ways:

1. From Launcher: Double click on SNMPV3Admin icon under SNMP agent.
2. Using Scripts: Run the SnmpV3AdminTool.bat/.sh script file present under '<JavaAgent_Home>/bin' directory.

15.2 Connecting to SNMPv3 Agent

The SNMP V3Admin tool can be connected to an existing SNMPv3 agent as follows:

15.2.1 Connect to SNMPv3 Agent:

Click on Operations->Connect to connect to the SNMPv3 agent. The User Profile Information dialog pops up.

![User Profile Information dialog]

Enter the following details in the User Profile Information dialog:

- **Target Host**: The host in which the SNMPv3 agent is running.
- **Target Port**: The port in which the agent is running.
- **Security Name**: The security name of the user on whose behalf the operations are to be carried out in the SNMP v3 agent. Refer to the Default Users of SNMPv3 Agent section to know the default security names of the users to connect to the SNMPv3 agent.
- **Auth Protocol**: The authentication protocol to be used for authenticating the user. Either MD5 or SHA protocol can be used. If no authentication is required, select NO_AUTH from the list.
- **Auth Password**: The authentication password of the user. This is not required for NO_AUTH users.
- **Priv Protocol**: The type of privacy protocol to be used for encryption. Choose either NO_PRIV or CBC_DES.
- **Priv Password**: The private key used for encryption. This is not required for NO_PRIV users.
- **Context Name**: The name of the SNMP Context.

Click **OK** to connect to SNMPv3 agent.

Once the security credentials that are provided using the tool are successfully authenticated by the agent, then the request will be sent to the agent to retrieve the USM and VACM table information. After the information are retrieved it will be exposed in the left side panel of Admin Tool. The purpose of connecting to the agent is as follows:

- helps the admin user to know about the type of user configured to the agent.
- you can perform minor validations using these details.

### 15.2.2 Disconnect from SNMPv3 Agent:

Choose **Operations->Disconnect** to disconnect from the agent. You can also click the Disconnect icon from the toolbar.

### 15.3 USM Operations

The USM operations possible using SNMPv3 administration tool are:

#### 15.3.1 Add User:

This operation lets you add a new user to the existing list of users in the USM table of the SNMPv3 agent.

**Note**: When you add a new user, the auth and priv protocol details of the user must be the same as the user for which you connected to the SNMPv3 agent, i.e., the type of user connected to the agent using the admin tool acts as a 'clone from user' for any new user added to the USM table. By default, you would connect to the agent as 'auth' user and hence you can add only 'auth' user.

You have two options while adding a new user:

- Create a new user and map it to the existing VACM details.
- Create a new user along with the authorization (VACM) details. The user need not do the VACM operations separately.

#### 15.3.1.1 Creating a New User and Map it to the Existing VACM Details

To create a new user, choose **USM operations->Add User** from menu bar. Now, the **User Settings** dialog pops up.
The following are the information in this dialog. In the User Settings dialog, provide the Security Name, and Auth Password or Priv Password or both, depending on the user type:

- **Security Name**: Name of the user to be added to the existing list of users in the USM User Table.
- **Auth Protocol**: The authentication protocol used for authenticating the request sent to the SNMPv3 agent by the manager on behalf of this newly added user. User with either MD5 protocol and SHA protocol can be used.
- **Auth Password**: The authentication password for the user. This is disabled, if NO_AUTH user is selected while connecting to the agent.
- **Priv Protocol**: The type of privacy protocol to be used for encrypting the requests sent to the agent by the manager on behalf of this newly added user.
- **Priv Password**: The private protocol used for encryption. This is disabled, if NO_PRIV user is selected while connecting to the agent.

Click on *Next*. The *Group Settings* dialog pops up.

All the existing user groups are listed in the ‘Group Name’ field. Select the appropriate ‘Group Name’ from the list and click *Update* to complete the operation.

### 15.3.1.2 Creating a New User along with the Authorization (VACM) Details

To create a new user, configure the user details as explained in 15.3.1.1 section for 'User Settings'. All the authorization details for the newly created user, i.e., adding a group, access, and view to the VACM table, can be configured by doing the following steps:
15.3.1.2.1 Adding a New Group Name

To add a new group name, provide the name in the GroupName field of the 'Group Settings' dialog. This creates a new group name. Click 'Next' to add a new access name.

15.3.1.2.2 Adding a New Access Name

The Access Settings dialog is as shown below:

![Access Settings Dialog]

Provide the following information in this dialog:

- **groupName**: The group under which the newly created user will be categorized.
- **Context Prefix**: The prefix for the context name with which the manager is querying the agent.
- **Security Level**: The security level of the user with regard to authentication and privacy. A security level of **noAuthNoPriv** is less than **authNoPriv** which in turn is less than **authPriv**.
- **Security Model**: The security model used for gaining access allowed by this entry. In our case, this is USM (User Security Model) and is not editable.
- **Context Match**: If the value of this object is **exact(1)**, the value in Context Prefix should exactly match Context Name. If the value is **prefix(2)**, then it is enough for the Context Name to have Context prefix as the prefix.
- **Read View Name**: The MIB View for which read access is allowed.
- **Write View Name**: The MIB View for which write access is allowed.
- **Notify View Name**: The MIB View for which notification access is allowed.
15.3.1.2.3 Adding a New View Name

To specify a new View name, click the '...' symbol of Read/Write/Notify View Name. The View Settings dialog pops up.

The following fields are present in this dialog:

- **View Name**: The name of the MIB view.
- **SubTree**: The subtree OID in the MIB for which Read/Write/Notify access is allowed for the user.
- **FamilyMask**: The field used to control the elements of the OID sub tree that are considered relevant when determining the view in which an OID is in. The default value is "ff".
- **FamilyType**: This can be either "included" or "excluded" where 'included' denotes the view of all the ObjectID under the OID specified in SubTree. The type "excluded" denotes the specified view of ObjectID under the OID specified in SubTree, i.e., the OID with the next group, say .1.3.7 cannot be accessed by the user.

By default, the existing views are listed and the fields such as Family Mask and Family Type are disabled. When you create a new view by specifying the new View name, the Family Mask and Family Type fields are enabled.

After providing the necessary details, click **Update** to complete the operation.

15.3.2 Modify User

You can modify the password details of existing users using the Modify User option. To do this option:

- Select USM Operations -> Modify User or
- Select Modify User icon from the toolbar

Upon invoking Modify User, the User Settings dialog pops up. Specify the necessary details and click **Update** to complete the operation.

**Note**: You cannot change configuration for NO_AUTH user. For NO_PRIV users, the oldPrivPassword and PrivPassword fields will be disabled.

15.3.3 Delete User

You can delete an existing user from USM user table using the Delete User option. To do this option:

- Select USM Operations -> Delete User or
- Right-click on the user name in the left hand panel and select 'Delete User' or
Select the 'Delete User' icon from the toolbar.

**Warning:** Any request from the admin tool will be sent to the agent on behalf of the type of user with which you have connected to the agent from the tool. Hence, care must be taken while deleting entries related to this type of user from USM and VACM tables as it hampers the processing of the request sent from the tool in the agent.

### 15.4 VACM Group Operations

The users in the USM table can be further categorized into VACM groups. This group along with other constraints is used for deciding the access permissions for a particular user. The following VACM group operations can be performed:

#### 15.4.1 Add Group

The *Add Group* operation is used for adding a new VACM group. You can invoke *Add Group* operation by any of the following ways:

- Select 'VACM Operations -> Group -> Add Group' from the menu bar
- Click the 'Add Group' icon in the toolbar.

Upon invoking the 'Add Group' operation, the *Group Settings* dialog pops up. Specify the appropriate details as specified in the 'Add Group Name' section and click *Update* to add a new VACM Group. These details will be persisted in *VacmSecurityToGroupTable* present under the *JavaAgent_Home*/snmpprojects/<project_name>/agent/bin/conf directory.

**Note:** The value of the SecurityName field to the Security Model field must be unique in the *VacmSecurityToGroupTable* file.

#### 15.4.2 Modify Group

The *Modify Group* operation lets you modify the Group Name of an existing group. The SecurityModel and SecurityName fields cannot be modified.

This operation can be invoked by selecting 'VACM Operations -> Modify User' from the menu bar. The *Group Settings* dialog pops up. Specify the appropriate Group Name and click *Update*.

#### 15.4.3 Delete Group

You can delete an existing group from the VACM Group list by invoking Delete Group operation. Select the group to be deleted and invoke the Delete Group operation. The corresponding entry will be deleted from the *VacmSecurityToGroupTable* file.

### 15.5 VACM Group Access Operations

The VACM Group Access Operations manipulate the *VacmAccessTable* file, which determines the access rights of each group.

#### 15.5.1 Add Access

This operation lets you add a new access in the VACM Access table. This operation can be invoked by selecting 'VACM Operations -> Access -> Add Access' from the menu bar.

The *Access Settings* dialog comes up. Provide relevant details as explained in the Add New Access Name section. Click on *Update* to complete the Add Access operation.
15.5.2 Modify Access

If you need to modify the access settings in the VACM Access table, then you need to perform the Modify Access operation. The access details can be modified from the 'Access Settings' dialog. Only Context Match, Read View Name, Write View Name, and Notify View Name fields can be modified. After providing the relevant details, click Update to complete the operation.

15.5.3 Delete Access

This operation is used to delete the access settings from the VACM Access table and can be invoked the same way you invoke the 'Delete User' operation. The corresponding entry will get deleted from the VacmAccessTable file.

15.6 VACM View Operations

The VACM View operations are used for defining the View details for the entries in the VACM table.

15.6.1 Add View

This operation lets you add a new view in the VacmViewTreeFamilyTable. Upon invoking the Add View operation, the View Settings dialog pops up. After providing the relevant details as explained in the 'Add New View Name' section, click Update to add view for the group.

15.6.2 Modify View

This operation lets you modify an existing view in the VacmViewTreeFamilyTable. Upon invoking this operation, the View Settings dialog is displayed. Only the FamilyMask and FamilyType fields can be modified here. After providing the relevant details, click Update to complete the operation.

15.6.3 Delete View

This operation lets you delete an existing view from the VacmViewTreeFamilyTable. Select the view to be deleted and invoke the Delete View operation. The corresponding entry will be deleted from the VacmViewTreeFamilyTable file.
16.0 Developing and Porting SNMP Agent in J2ME CDC Environment

16.1 Overview

The existing platforms and specifications are targeted at devices equipped with more memory, powerful networking capabilities, more capable user interfaces etc. When devices require less memory footprint, specific-purpose, limited-function applications, J2ME CDC can be preferred to address the needs. J2ME CDC is introduced to address consumer and embedded devices with the same network capabilities, user interfaces etc but with small memory space.

AdventNet Agent Toolkit SNMP agent is newly designed to be ported in the J2ME CDC environment that targets the device with less-memory and limited-resource requirement. The toolkit renders end-to-end development of the SNMP agent that can be easily deployed to manage the connected device. By this feature, AdventNet facilitates the customer to port the SNMP agent into such small memory devices for providing SNMP manageability. The document will further detail the steps involved in developing the SNMP agent and deploying it in the J2ME CDC environment.

16.2 Compatible J2ME CDC Specification

The SNMP agent is compatible with two implementation of J2ME CDC specification i.e the agent is developed, instrumented, ported, and tested under these implementations. They are

- SUN J2ME CDC Implementation
- IBM J2ME CDC Implementation

16.3 Developing the SNMP Agent

AdventNet provides combined features of developing an SNMP(v1/v2c) agent and with less memory footprint. When you have Management Information Base (MIB) of your device and need to manage the managed information through SNMP, you can develop an SNMP agent where you just require to instrument the generated stub files. Developing the agent is very simple using the GUI tools which is briefed below:

Steps to build an SNMP Agent using the toolkit:

1. Open the MIB Compiler and choose the project type as J2ME.
2. Load the MIB
3. Select **Project->Settings** that opens the Settings dialog. Configure the agent settings. This includes
   - **SNMPv1/v2c Compliance**: Implements managed objects as per SNMP (IETF) standards that expose the behavior of the SNMP entity.
   - **AclTable**: Provides authentication to users to access the agent based on community. The table stores the details of management applications for particular community.

AdventNet Inc.
v1/v2cTrapForwardingTable: The table stores information of the traps, which gets generated due to some undesirable events in the agent, and its target.

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>• The table information is persisted in the runtime memory (RAM). The J2ME CDC SNMP Agent does not support any other storage model.</td>
</tr>
<tr>
<td>• The logging option can be availed, i.e. the log files are created by default under <code>&lt;AdventNet/Javaagent&gt;/snmpprojects/&lt;snmpprojname&gt;/agent/bin</code> directory.</td>
</tr>
</tbody>
</table>

4. Generate the agent, which generates the stub files.
5. Instrument the agent stub files and compile them. While compiling the agent, use JDK 1.3.1 as per the J2ME CDC specifications.

16.4 Porting the Agent in J2ME CDC Environment

Once the SNMP agent is developed, it is ready to be deployed in the device. The agent can be generated and compiled in any operating system and then the compiled class files are ported in the J2ME CDC environment. As mentioned under “Compatible J2ME CDC Specifications”, let us go through the steps involved in porting the SNMP agent in each implementation.

16.4.1 Sun Implementation of J2ME CDC Environment

Before porting the SNMP agent, the user must have the Sun implementation libraries in the device.

The following are the steps required to port the SNMP agent:
1. Set the Java Home to the cdc home directory, i.e. the location where the J2ME CDC library files are available.
2. Once the Sun specification libraries are placed in the classpath, add AdventNet's generated stub files to the classpath with AdventNetLogging.jar and AdventNetJ2meSnmpAgent.jar.
3. Then to run the application, C Virtual Machine which comes with Sun package is used, i.e. to invoke the application, the command used is:
   ```shell
   ```

16.4.2 IBM Implementation of J2ME CDC Environment

Before porting the SNMP agent, the user must have the IBM WSDD implementation libraries in the device.

The following are the steps required to port the SNMP agent:
1. Set the Java Home to the cdc home directory, i.e. the location where the J2ME CDC library files are available.
2. Place the IBM libraries in the classpath along with AdventNet's implementation files, i.e. the generated stubs, AdventNetLogging.jar, and AdventNetJ2meSnmpAgent.jar.
3. To invoke the application, use the following command:
16.5 Testing the Agent

After developing and porting the agent in the J2ME CDC environment, start the SNMP agent in the J2ME CDC environment using the commands with respect to each implementation mentioned in the "Porting the SNMP Agent" section. The SNMP agent can be tested using MIB Browser to access the management information of the device. The agent can also be accessed through any standard SNMP manager. The following steps explain how to access the management information through MIB Browser:

1. Open the MIB Browser that is at `<AdventNet/Javaagent>/bin` directory.
2. Load the same MIB that is loaded in the agent.
3. Select a managed object and send GET/GET-NEXT/SET/GET-BULK request.

The agent responds to the MIB Browser with the managed object name and its value. To know more about accessing the information through MIB Browser, refer "Testing the Agent" section of J2SE SNMP agent.

16.6 SNMP Agent Memory and CPU Requirement

**Test Environment Details:**

The performance test was conducted on a Linux machine. In Windows machine, the test has been performed using Foundation Profile packages and not with CDC environment packages.

OS: Linux
RAM: 505340 KB
Processor Speed: 2392.082
JDK version: 1.3

**Agent Details:**

A SNMP J2ME agent was developed, instrumented, and ported in IBM J2ME CDC environment. Then, the agent was started and the agent memory usage and CPU usage details were noted down as shown below:

<table>
<thead>
<tr>
<th></th>
<th>By SNMP Agent</th>
<th>By JVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flash Memory Used</td>
<td>350 KB</td>
<td>-</td>
</tr>
<tr>
<td>RAM Used</td>
<td>3 MB</td>
<td>3.8 MB</td>
</tr>
</tbody>
</table>
17.0 Implementing Business Logic Using Rule Engine

17.1 Overview
Agent Toolkit SNMP Agent introduces the concept of Rule Engine that defines the users' business logic as a set of simple rules to monitor attribute(s) and perform tasks. These rules are connected together as ruleset in the rule engine and initiate actions such as sending traps/notifications, sending mails, and performing set operation when a given expression/logic is evaluated as true. These logics are stored as XML documents and can be manipulated manually by the user. The XML documents contain the set of data which is required to perform the defined operations in rule engine.

The advantage of Rule Engine is that it makes the users' business logic declarative through the XML file instead of hard-coding or instrumenting the decision process in the stub files to perform any action.

17.2 What Is Rule and Ruleset?
Rules are a set of expression/logic containing parameters or managed objects to monitor the attributes of the users' application/device and then execute some actions. It defines the conditions/criteria for the managed objects of the application and performs operations based on the expression. The input of a rule can be one or more expressions that result in an output of single value invoking some action.

The key concept/input of Rule Engine is Ruleset. They are collection of rules executing all the rules defined under it one-by-one.

17.3 Enabling Rule Engine
Rule Engine can be enabled by choosing Settings->Source Generation->Enable Rule Engine Service from the MIB Compiler UI. The following is the code generated in the agent's main file while enabling Rule Engine Service:

```java
//Enabling Rule Engine Service
private void initRuleEngine() {
    try {
        snmpLog.trace("Initializing Rule Engine");
        RuleEngineFactory.setAgentDetailFile("rule/AgentDetails.xml");
        RuleEngineFactory.setRuleDir("rule/");
        if (agentOptions.getDebugLevel() != -1) {
            RuleEngineFactory.setLoggingLevel(agentOptions.getDebugLevel());
        }
        RuleEngineFactory.setSnmpAgent(this);
        RuleEngine ruleEngine = RuleEngineFactory.createRuleEngine(1, "conf/ruleenginefactory.xml");
        RuleEngineFactory.setResource("SnmpSetAction", new SnmpSetActionImpl());
    }
```
Enabling the rule engine generates the configuration files under the 
<AdventNet/Javaagent>/snmpprojects/ruleengine/agent/bin/rule directory.

Based on the execution mode, the rule engine acts in either triggered mode or scheduler mode.

**Triggered-Based Rule Execution Mode:** Here, the action specified in the rule engine configuration file is performed, once the expression is "true". You have to manually execute a rule from your code specifying the corresponding RULEID. Add the piece of code wherever the rule needs to be triggered.

```
RuleEngineFactory.getRule().execute(1)
```

Here, 1 denotes the RULEID as mentioned in the .rule file.

The main advantage of triggered mode is that you can also use the functionality without processing all the rules provided in the rule file. By default, the rule engine operates in the triggered mode.

**Scheduler-Based Rule Execution Mode:** Here, a scheduler will be started, which will execute all the rules defined in the .rule file at periodic intervals specified in the setRefreshTime() method. Add the following piece of code to the initRuleEngine() method, to enable scheduler service by which the rule engine changes its mode of operation to scheduler:

```
RuleEngineScheduler scheduler = new RuleEngineScheduler(ruleEngine);
scheduler.setRefreshTime(1000);
scheduler.start();
System.out.println("Started Scheduler");
```

**Note:** The functionality of executing only a particular rule explained in the Triggered mode is not applicable to Scheduler mode.

### 17.4 Configuration Files Used for Rule Engine Operation

The operations of rule engine are executed by persisting the set of related data in some configuration file. These are editable files and you can configure or edit any data both at design time and at runtime. The modifications in the rule file or addition of any new rule file at runtime is updated without restarting the agent, if the HotDeployment is enabled, for every interval mentioned in the setHotDeploymentRefreshTime() method.

The configuration files are:
- AgentDetails.xml
- RuleSet1.rule

**Note:** The above HotDeployment applies to .rule file only and the interval for refresh time is specified in milliseconds.

#### 17.4.1 File Containing Agent Details

The configuration file, `AgentDetails.xml`, contains SNMP agent-related details such as the tables used for agent configurations. The following are the tables present in the file:
AGENT_TABLE: The table consists of AGENTID, AGENTNAME, and TYPEID.

AGENTID: This serves as a unique identification for the SNMP agent and associates the SNMP-specific details of the agent with the RuleSet in the rule file, i.e., all the operations that are associated with the ID will be evaluated. The default value mentioned in the rule engine is 1.

AGENTNAME: This implies Agent name, i.e., the name of the main class of the agent is reflected here. By default, it is AdventNetSnmpAgent.

TYPEID: This is a unique identification for different tables used for storing information of different table that are explained below. The default TYPEID for AGENT_TABLE is 1. In the AGENT_TABLE, the TYPEID is ‘2’ that represents the next table (SNMP_AGENT_DESCRIPTION_TABLE) to be processed.

SNMP_AGENT_DESCRIPTION_TABLE: This table holds the details of SNMP agent such as AgentID, AccessType, version, hostname, port, and community. If the version is 1 or 2, it implies that the agent is SNMPv1 or SNMP v2c respectively and the agent processing is performed with the mentioned community that is required for authenticating the requests. If the version is 3, then it implies that the agent is SNMPv3 and the SNMP_V3_AGENT_DESCRIPTION_TABLE is processed.

SNMP_V3_AGENT_DESCRIPTION_TABLE: As the name implies, this table contains all the security-related details required for SNMPv3 request processing with the context name.

TYPE_TABLE: This table has 2 columns:

- TYPEID: Uniquely identifies the tables used in the Rule Engine.
- TABLENAME: Name of the table used in Rule Engine.

17.4.2 File Containing Ruleset Details

The configuration file, RuleSet1.rule consists of the Ruleset with a set of rules defined in it and the operations resulting in the actions to be performed for the Ruleset. By default, the managed objects-related rules defined in notification-type/trap_type macros in the MIB module for which the agent is generated, are given in the rule file. The following are the tables available in the file:

RULESET_TABLE: A rule engine can have any number of rulesets using different rule files. A RULESETID is a unique identification of a ruleset in a rule file. By default, there is one rule file with RULESETID as 1. The STATUS column in the table provides the status of the table. That is, 0 denotes that the ruleset status is disabled and 1 denotes that it is enabled.

AGENT_RULESET_TABLE: This table describes the relation between the RULESETID and the AGENTID. Though there can be any number of rulesets in the rule engine, only those with the AGENTID as mentioned in the AgentDetails.xml are executed.

RULE_TABLE: Every rule (identified using RULEID) will have an EXPRESSIONID, ACTIONSETID, STATUS, and polling interval details. The EXPRESSIONID and ACTIONSETID are explained in their respective tables.

- STATUS: This column in the table provides the status of the table. That is, 0 denotes that the rule status is disabled and 1 denotes that it is enabled.

Polling Interval: This mentions the interval after which the rule engine polls the agent for data contained in the managed objects. By default, the polling interval is 5000ms.

RULESET_RULE_RELATION_TABLE: A ruleset is composed of one or many rules. This table is used to get the rules for the specified ruleset. This is obtained by mapping the set of
rules using RULEID to the RULESETID which is performed in this table. The combination of RULEID and RULESETID should be unique.

**EXPRESSION:** It consists of EXPRESSIONID that specifies the different expressions to be executed in the rule engine. An expression performs the work of a program and consists of a series of variables and operators that are evaluated to a single value called the output. Here, the expression is identified using an expressionID which contains LEFTEXPID and RIGHTEXPID with an operator. Both the LEFTEXPID and RIGHTEXPID refers to the DATAID. These expressions can be either Composed Expression or Expression Operator Expression (EOE).

**EOE:** Any simple expression that compares two variables or constants using operators such as GreaterThan(>), LessThan(<), GreaterThanOrEquals(>=), LessThanOrEquals(<=), equals(==), ! =, IsChanged, etc. are mostly considered the Expression Operator Expression (EOE).

**Composed Expression:** Any compound expression that evaluates complex or more than one expressions is mostly considered as the composed expression. The EOE can be made as a part of composed expressions with operators such as AND, OR, etc.

**ACTION_TABLE:** When any expression is executed, an action should be performed as an acknowledgment to the expression being successfully completed. The outputs from a rule engine may be viewed through actions such as sending traps/notifications, mails, or setting another expression. These actions are denoted by ACTIONID. The different actions supported by the rule engine are:

- **Traps:** Traps are generated from the agent usually when the values of the data objects defined by trap-type are modified. Using Rule Engine, traps can also be sent while executing some expression/logic. These traps also contain information similar to the traps sent through trap definition.

- **EMAIL:** This action sends an e-mail to the email ID mentioned in the rule engine. This also sends mail with To, Cc, and Bcc address. The subject and message body of the mail is modifiable and any message related to the rule can be mentioned, say “E-mail received for $object$ from $host$ and $port$”, where the string passed between $ sign mentions the macros. To know about macros, refer to MACRO_TABLE.

- **SET Action:** When an expression is executed, a SET request is sent to a managed object as defined in the SNMP_SET_ACTION_TABLE.

**ACTIONSET_TABLE:** An ActionSet contains one or more actions. So the ACTIONID is mapped to the ACTIONSETID using this table. For example, if there are 3 actions to be performed, then the actions can be grouped under one ACTIONSETID. There can be more than one ACTIONSETID.

**DATA_TABLE:** This table includes mapping of DATAID with TYPEID. The DATAID is the identification for the data used in the expression. An expression consists of variables, constants, and operators combined to perform some useful computation. The variables or constants are referred to using DATAID. The DATAID will in turn refer to a table such as SNMP_DATA_TABLE or CONSTANT_DATA_TABLE, using TYPEID for retrieving data.

**DATA_LIST_TABLE:** The table has DATALISTID and DATAID. The Datalist concept is used for grouping data. The DATALISTID is used to send extra varbinds through trap with the data mentioned using DATAID. This table specification is optional.

**MACRO_TABLE:** Macros are messages included in all the e-mail sent from the agent when configured in the rule engine. This reduces your manual work as you need to send any message that is unique to all e-mail, say a signature. This table consists of MACROID,
MACRONAME, and DATAID. Here, MACROID and MACRONAME must be unique. Once the MACRONAME is passed between "$" sign in the EMAIL_ACTION_TABLE, then all the e-mail messages carry the data mentioned in the DATAID which is specific to that MACROID.

**SNMP_DATA_TABLE:** This table is used for retrieving the value of an SNMP attribute from the associated agent. The fields include DATAID, GROUPNAME, SUBID, and INSTANCE. The managed objects or the data used for the expression is identified using the DATAID mentioned in the DATA_TABLE. The other parameters define the details of the object.

- **GROUPNAME:** This holds the name of the group (scalar or tabular) to which the managed object belongs.
- **SUBID:** Here the SUBID of the object must be specified.
- **INSTANCE:** For a scalar variable, the instance must be specified as "NONE". For a tabular variable, the instance must be mentioned. The rule engine supports expression for tabular variables with both index and external index. For example, both `<instance>.1</instance>` and `<instance>.6.87.23.56.35.24.67</instance>` are allowed. If `<instance>ALL</instance>` is mentioned, then values from all rows will be retrieved.

**CONSTANT_DATA_TABLE:** This table is used to store the values of the variables, i.e, constants. Here, the TYPEID denotes the TypeID of the type of the constant value (for example, java.lang.String).

Thus, the expression is evaluated using the above-mentioned tables and requires to acknowledge using actions which are enabled using the below tables.

**EMAIL_ADDRESS_TABLE:** For sending e-mail, the mail address is mentioned in the configuration file using this table. Any number of address can be given and they are identified using EMAILID.

**EMAIL_SERVER_SETTINGS_TABLE:** This table is used to specify the details of the SMTP server through which you need to send the mail. The server name is mentioned in the SERVERNAME and its port in PORT. If the mails are supposed to be sent through more than one server, they can be provided using a different SERVERID.

**EMAIL_ACTION_TABLE:** This table provides e-mail details such as the content of the mail, subject, and message body. The table parameters include:

- **ACTIONID:** The ID is taken from ACTION_TABLE.
- **SERVERID:** Identifies the SMTP server to send the e-mails and the value is obtained from EMAIL_SERVER_SETTINGS_TABLE.
- **EMAILID:** Refers to the “From” address and can be any string mentioned in the EMAIL_ADDRESS_TABLE.

**EMAIL_ACTION_RELATION_TABLE:** This table is used for finding out the recipients of this e-mail. It gives the relation between EMAILID and ACTIONID where the ACTIONID decides whether the e-mail needs to be sent for the EMAILID. The e-mail will be sent only if the EMAILID is mapped to the ACTIONID that is specified in the EMAIL_ACTION_TABLE. Also, the e-mail can be send as To, Cc, and Bcc through RECIPIENTTYPE where 1 denotes To, 2 denotes Cc, and 3 denotes Bcc.

**SNMP_SET_ACTION_TABLE:** As explained above, rule engine performs a SET operation, when a given expression is evaluated. The expression or the details for the SET operation are provided in the table. It includes parameters such as

- **ACTIONID:** The ID mentioned for SNMP SET action in the ACTION_TABLE.
- **VERSION:** Version of the request.
- **OID:** The OID of the managed object.
TYPEID: This denotes the data type of the managed object to which the SET operation is to be performed. Here, the mapping for the SNMP data type is done based on how it is defined in the SNMP API. For reference, see the SNMP API javadocs.

VALUE: Value to be set.

INSTANCE: Instance of the object.

**SNMP_TRAP_ACTION_TABLE**: This table gives information on the mapping of ACTIONID and TFTABLEID. The TFTABLEID is an identification for the trap forwarding table.

**TRAP_FORWARDING_TABLE**: This contains the Trap Forwarding Table details that include the following:

- **TFTABLEID**: An identification for the trap forwarding table and the trap will be send only to those entries present in the SNMP_TRAP_ACTION_TABLE. If given a value of "0", this parameter will send trap to all entries available in the Trap Forwarding Table of the agent.
- **MANAGERHOST**: This implies the IP address or the host of the manager.
- **MANAGERHOSTTYPE**: This implies the host type (IPV4 or IPV6).
- **MANAGERPORT**: This refers to the port at which the manager is running.

### 17.5 Functioning of Rule Engine

Now that we have known the tables in the rule engine configuration files, let us see the functioning of the rule engine to execute any expression based on the tables.

1. **Rule Engine Initialization**: The SNMP agent-specific details such as SNMP port, version, etc., will be obtained by the Rule Engine and it will be initialized. Now all the .rule files in the `<AdventNet Home>/<project_name>/agent/bin/rule` directory will be checked and the rules given in the files will be executed.

2. **Rule Execution**: First, the set of Ruleset(s) corresponding to the SNMP agent is determined. The details about the Ruleset(s) are available at RULESET_TABLE. If the Ruleset is active (Status = 1), then all the rules under this Ruleset are executed.

   From RULESET_RULE_RELATION_TABLE, the rules corresponding to a Ruleset are taken and executed one-by-one. The details about the rule(s) are taken from the RULE_TABLE. For each rule, a expression will be evaluated using the EXPRESSION table and if the expression is evaluated true, a set of actions can be performed.

   **Note**: All the rules defined in .rule file will be executed only in the Scheduler mode. In the Trigger mode, you have to manually execute a particular Rule by using the code.

3. **Expression Evaluation**: EXPRESSION table has the ID that associates itself to a table that provides the description of the expression. The expression may be simple such as attributeA != attributeB or complex such as {attributeA != attributeB} AND {attributeB != 'A'}. To handle expressions, the following tables are used:

   - **COMPOSEDEXPRESSION**: This table is used for defining the complex expression. It can be of the form:
     - SimpleExpression1 BooleanOperator SimpleExpression2
   - **EOE**: This table is used for defining the simple expressions, which will be of the form
     - LeftOperand Operator RightOperand

   **Note**: If the Operator used is a unary operator such as IsChanged, then RightOperand is not necessary.
To evaluate an expression of a rule, the corresponding EXPRESSIONID is considered. The EXPRESSION table will be checked for this EXPRESSIONID. If it is not present, this expression will be evaluated false or the Composed expression table is checked. In this table, the left and right operands (which again may be a simple/complex expression) will be checked recursively and proceeded further.

If there is no entry for composed expression, then the EOE table is considered. To obtain the values of the left and right operands in simple expression defined in EOE table, the columns LEFTEXPID and RIGHTEXPID are used and they refer to the DATAID.

For Example, the following composed expression must be evaluated using Rule Engine,

\{ (agentNetstat == 'net') OR (agentPing == 'ping') \} AND \{applicationUserName == 'testing' \}

For calculating the preceding composed expression,
In composed expression table, following entries should be present
C1 -> the whole expression
C2 -> \{ (agentNetstat == 'net') OR (agentPing == 'ping') \}

In EOE table, following entries will be present
E1 -> \{applicationUserName == 'testing' \}
E2 -> (agentNetstat == 'net')
E3 -> (agentPing == 'ping')

The above expression can be represented as
C1 -> C2 AND E1

where C2 -> E2 OR E3

Using the DATAID, the constant or variable data used in the expressions can be retrieved in the following ways:

1. The DATAID is checked in the DATA_TABLE where it is associated to TYPEID depending on whether the data type is constant or SNMP, (that is, the value of attributes defined in MIB modules for which the agent is generated).
2. If the data is constant, the value of the data will be mentioned in the CONSTANT_DATA_TABLE. The TYPEID column defines the type of the constant data (That is, java.lang.String, java.lang.Integer etc) in the AgentDetails.xml file.
3. If the data is SNMP, the SNMP_DATA_TABLE is used to retrieve the value from the agent. In case of retrieving values from the table groups (defined in MIB), the relevant instance part should be given in the INSTANCE column.

4. **Performing Actions:** Once the expression is evaluated true, the corresponding ActionSet, having a list of actions to be performed for this rule, is specified by the column ACTIONSETID. The list of actions in a particular ActionSet can be obtained from the ACTIONSET_TABLE.

The details about each action will be present in the ACTION_TABLE. Now depending on the TYPEID column in ACTION_TABLE, three types of actions can be performed. They are as follows:

- **SNMP Trap Action** - This action is used for sending SNMP traps/notifications/informs to the manager.
- **SNMP Set Action** - This action is used for performing an SNMP SET operation for an attribute defined in the MIB module for which the SNMP agent is generated.
- **Email Action** - This action is used for sending e-mails.
4.1 **SNMP Trap Action:** For performing SNMP Trap action, SNMP_TRAP_ACTION_TABLE is used. For sending traps, the configuration tables of the agent v3TrapForwardingTable/v1v2TrapForwardingTable are used. Here the relevant details such as manager host address and manager port are present. The OID of the traps generated by the Rule Engine is specified in the column TRAPOID.

There is also a provision for sending additional Snmp varbinds along with the Trap PDU using DATA_LIST_TABLE.

4.2 **SNMP Set Action:** For performing this action, SNMP_SET_ACTION_TABLE is used. The value to be set is given in the VALUE column and the SNMP type of the value to be set is given by the corresponding TYPEID. For performing SNMP Set operations for the attributes in table groups, the instance part should be specified in the INSTANCE column.

- **Note:** For sending String data, the type STRING (notation from SNMP API class), not java.lang.String must be used.

4.3 **E-mail Action:** The EMAILID column denotes the 'From' address of the e-mail to be sent. The SMTP server configurations will be taken using the SERVERID column. Using the ACTIONID, the EMAIL_ACTION_RELATION_TABLE will be used for finding the intended recipients of the e-mail. There is a provision for using Macros in Subject and Message body. To use the macros, the ISMACROALLOWED column in ACTION_TABLE should be 1(true).

### 17.6 Customizing Rule Engine

The rule engine can be customized using the various interfaces packaged with the Agent Toolkit. The interfaces helps you to implement your own classes that in turn evaluates the rule file accordingly. The interfaces generated for rule engine are as follows:

- Action
- ComposedExpression
- DataCollector
- EmailAction
- Expression
- Macro
- PersistenceHandler
- Rule
- RuleEngine

While enabling the rule engine, it loads the classes related to the ruleset operations from the ruleenginefactory.xml file. This file is generated under the `<AdventNet/Javaagent>/snmpprojects/<projectname>/bin/conf` directory and contains the default implementation class names for expression, action, rule etc. You can enjoy your own functioning of the rule engine by providing his implementation classes in place of the default classes in the xml file.

### 17.7 Packaging Details

The jars and the configuration files required for rule engine are as follows:

- AdventNetRuleEngineFramework.jar
- AdventNetSnmp.jar
- AdventNetSnmpAgent.jar
- Mail.jar
- Activation.jar
- AdventNetAgentRuntimeUtilities.jar
- conf/ruleenginefactory.xml
- rule/RuleEngine_DBSchema.xml (optional)
- rule/AgentDetails.xml
- rule/RuleSet1.rule
18.0 Making the SNMPv2c Agent Compliant to SNMPv2c Standards

18.1 Overview

The SNMPv2c Agent does a lot of processing such as Authentication, Processing Requests, Sending Notifications, etc., as defined in the standards. You may encounter errors when doing the above functions. In such a case, the Management stations require the details of the errors. They might also need some common (standard) information such as sysName, sysUpTime, sysObjectID, etc., from the agents in order to define uniform management.

The above requirements are addressed using RFC 3418 (SNMPv2-MIB). It has different groups namely: System Group, Set Group, SNMP Group Counters, SYSORTable, and SNMP Notification Group. AdventNet SNMP Agent has implemented this MIB and provides support for the same.

18.2 Enabling V2 Compliance

SNMPv2c Compliance can be enabled, either using MIB Compiler UI options or using API Calls.

Using MIB Compiler UI

By default, V2 Compliance is enabled for an SNMPv2c and SNMPv3 Agent. To disable it,

```
Clear the V2 Compliance option in Project -> Settings menu -> General Panel of MIB Compiler UI.
```

Using API calls

To implement V2 Compliance through API calls, add the following piece of code in the main file generated below all the registrations made in initSnmpExtensionNodes()

```
// For Snmp Group counters support (for both v1 and v2)
AgentSnmpGroup grp = new AgentSnmpGroup();
grp.addRegistrationListener(hdlr);
super.addAuthenticationListener(grp.getAuthenticationListener());

// SysORTable support for SnmpV2 Compliance
sysORTable = new com.adventnet.snmp.snmp2.agent.SysORTableRequestHandler((SnmpAgent)this, "conf", "SysORTable.xml", false);
sysORTable.addRegistrationListener(hdlr);

// System Group support for SnmpV2 Compliance
SystemGroupInstrument instru = new SystemGroupInstrument();
super.addSystemGroupListener(instru);

// SnmpSet Group support for SnmpV2 Compliance
snmpSetGroupListener = new com.adventnet.snmp.snmp2.agent.SnmpSetRequestHandler((SnmpAgent)this);
super.addSnmpSetGroupListener(snmpSetGroupListener);
```

This would implement V2 Compliance. Also, include the following lines between the codes given for Variable Declarations in the Main file.
18.3 Supported Standards

SNMPv2 Compliance supports the following groups along with their implementations.

18.3.1 SNMP Group Counters (Compliant to v1 Standards)

SNMP Group Counters are defined in SNMPv2 MIB to count the number of packets received in each category such as valid PDU, invalid PDU, bad authentication, GET, GET-NEXT, SET, etc., of an SNMP entity. The AdventNet Agent Toolkit has implemented these counters as per the RFC1213 MIB specification. This implementation is optional.

18.3.2 System Group Support

The AdventNet Agent Toolkit provides methods for instrumenting the System Group of SNMPv2 Mib. This group is a collection of objects common to all managed systems. The objects that are defined in this group are as follows:

- **sysDescr** - Description of the managed node.
- **sysObjectID** - Object Identifier of the node.
- **sysUpTime** - Time (in hundredth of a second) since the network management portion of the system was reinitialized.
- **sysContact** - Contact person for the managed node.
- **sysName** - Name of the managed node.
- **sysLocation** - Physical location of the node.
- **sysServices** - Value that indicates the set of services offered by this node.
- **sysORLastChange** - The value of sysUpTime at the time of the most recent change in state or value of any instance of sysORID.

Implementing a Listener To Read Your System Values

The SystemGroupListener class in com.adventnet.snmp.snmp2.agent package acts as an interface for the SystemGroup. You can implement this class to read the original values from your managed object. For example, you can have your own implementation of SystemGroupListener and add the registration code in the Main file, below all the registrations made in initSnmpExtensionNodes(),

```java
SystemGroupListener impl = new MySystemGroupImpl();
super.addSystemGroupListener(impl);
```

18.3.3 SNMP SET Group Support

The AdventNet Agent Toolkit provides methods for instrumenting the SnmpSet Group of the SNMPv2 MIB. The objects defined in this group provide a locking mechanism for the Managers for providing SNMP SET requests. The object defined in this group is snmpSetSerialNo.

18.3.4 sysORTable

The sysORTable defined in SNMPv2 Mib has the following columns defined in it:

- **sysORIndex** - the index column.
- **sysORID** - the OID of the Mib Module supported by the Agent.
- **sysORDescr** - the Description of the Mib Module.
- **sysORUpTime** - the sysUpTime when the row was last instantiated.
### 18.3.4.1 Storing the Table Data

The sysORTable information gets stored in a data file called the `SysORTable.xml`. This file gets created under the conf folder in `<Agent Toolkit Home>/snmpProjects/projectname/agent/bin` directory. The Agent is started by reading this text file that contains the default entry as:

<table>
<thead>
<tr>
<th>Index</th>
<th>OID of Mib Module</th>
<th>Description of Mib Module</th>
<th>sysORUpTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>.1.3.6.1.6.3.1</td>
<td>The Mib Module for SnmpV2 entities.</td>
<td>agent uptime</td>
</tr>
</tbody>
</table>
19.0 Supported Storage Types for Scalars and Tables

19.1 Overview
This section deals with the storage type options provided by Agent Toolkit for Scalars and Tables.

19.2 Storage Types for Scalars and Tables
The supported storage types for a Table include:

Text File
This is a flat file, which stores the data of a Table in the following format (Sample entry).

```
1|1|column3 not initialized|1|
```

This file created under `<Agent Toolkit Home>/snmpprojects/projectname/agent/bin/textFiles` directory is read on Agent startup. To add a new row entry to this Table a similar entry has to be included to the text file with the values that you require. For scalars, all the scalar variables under a scalar group will be listed as given in the sample entry differentiated with the pipe symbol. Please note that the Agent has to be restarted for the changes to take effect. When this storage option is chosen, the Agent reads the values from text files only.

XML Files
This is an XML-formatted file that stores the data of a Table as given below (Sample entry).

```
<?xml version="1.0" encoding="UTF-8"?>
<table>
  <row>
    <column name="column1" value="1" ></column>
    <column name="column2" value="1" ></column>
    <column name="column3" value="column3 not initialized" ></column>
  </row>
</table>
```

This file created under `<Agent Toolkit Home>/snmpprojects/projectname/agent/bin/xmlFiles` directory is read on Agent startup. To add a new entry to this Table, a similar entry (starting from `<row>` to `</row>`) has to be included to the XML file with the values that you require. Please note that the Agent has to be restarted for the changes to take effect. When this storage option is chosen, the Agent reads the values from XML files only.

Database
In some cases, the application databases have the potential to be extremely large and dynamic. Storing these information in the AgentTableModel (default storage model for Tables) would occupy considerable memory space and would result in wastage of resources. This problem gets resolved when storing the information in a database. With this model, there will be no problems on memory hogging. As and when required, the values are retrieved by specifying the URL.
Agent Toolkit supports database access for any SNMP request. The Agent API establishes the connection between the Agent and the Database before Agent startup.

Runtime Memory

Runtime Memory does not store the details in a text file, XML file, or a database. Instead it stores the data in Agent memory. Entries can be added to the Table during run time from the Manager when Runtime Memory option is chosen. But these data get lost once the Agent is stopped.

The supported storage type for a Scalar includes RAM and Text file alone.

19.3 Configuring the Storage Type

Now that you know the different storage type options, it is required to know how to configure the Agent for availing these storage options. Configurations can be done using MIB Compiler UI or using API Calls.

To configure the settings using MIB Compiler UI, follow the steps given below:

- Create a Project and load a MIB.
- Select Project -> Settings from the menu bar of MIB Compiler UI.
- Select Storage Model from the Source Generation panel.
- You can select the storage type for the Tables in a MIB from the combo box given across Table.
- You can select the storage type for the Scalars in a MIB from the combo box given across Scalar.
- To customize the storage type for a particular table or a scalar node, use the Node Storage Type. Select the node and choose the storage option.
- Configuring Database is dealt with as a separate section. Refer to the same to know how to configure for database storage.

19.3.1 Configuring Database

If the Storage Type is Database for all or any of the Tables in a MIB, the Database parameters have to be provided to access the database. To configure the database, using MIB Compiler UI options follow the steps given below:

Using MIB Compiler UI

- Select Project -> Settings menu from the menu bar of MIB Compiler UI.
- Select Storage Model from the Source Generation panel.
- Click Configure - Database and give the following inputs:
  - Database Name - Name for the Database.
  - URL - URL from which the Database is to be located.
  - Driver - Name of the Driver.
  - User name - An User name.
  - Password - A Password for the User.

For example, assume that you have to provide storage in Database for the adiskTable of the AGENT-SAMPLE-MIB. In order to proceed with this, do the following:

1. Add the database file (from an application which provides a ODBC driver) to the ODBC Data Source Administrator. Provide a unique name for the file as "DatabaseTest"
2. Once this is done, you have to associate this data source to the adiskTable object in the MIB file. In this case, the DATABASE-URL should be "jdbc:odbc:DatabaseTest"
3. Specify the Database driver name as "sun.jdbc.odbc.JdbcOdbcDriver".
4. Specify a User Name and Password. For Example, the User Name can be "AdventNet" and the Password can be "adventnet".

**Note:** The Database Name, URL, Driver, UserName, and Password must be the same for all the Tables, which provide database storage.
20.0 Logging in SNMP Agent

20.1 Overview

Log messages or Debug messages produces details of the Agent processing. These messages help in tracing out the problems or issues faced while working with the Agent. They get stored in a text file and provide information that you require from the Agent. This comfort of receiving debug messages would be beneficial while testing the developed Agents.

Before learning to enable the logging facility, you have to know the various logging levels.

20.2 Logging Levels

The Logging options available for SNMP Agents are as follows:

- **Disabled (0)** - This level indicates that the debugging mode is disabled. Hence, log messages do not get stored in the text file.
- **Fatal (1)** - This level indicates that the application has encountered a severe error. These errors need to be attended with care. When this level is chosen, critical messages such as details on Agent crash, Authentication errors, Agent already open in another session, etc., get stored in the log file.
- **Error (2)** - This level indicates that an error has occurred and the application cannot handle the request as designed. This may be automatically corrected during execution or the application should be capable of handling those situations when a similar request comes again. Selecting this level store all the Error and Exception messages in the log file along with the fatal messages.
- **Warning (3)** - This level indicates that the application has met with an expected error. All Warning messages get stored in the log file when debug level is WARN.
- **Info (4)** - This level does not signify the occurrence of any error but logs all important events. Any information that might be required for reference gets stored in the log file under this info level.
- **Debug (5)** - This level indicates that the application moves into the debug mode. It will be used for generating debugging information and should not be used in general cases.
- **Trace (6)** - This level indicates that the application logs every activity from entering a method to exiting. Hence, selecting this level may affect performance. The Trace level should be used only to solve any complex problem.

20.3 Setting Logging Levels

The logging levels can be enabled, either using MIB Compiler UI or using API Calls.

**Using MIB Compiler UI**

- Create a Project and Load a MIB.
- Select Project -> Settings menu from the menu bar of MIB Compiler UI.
- The General Panel has the options for Debug level.
- By default, WARN level is chosen in the combo box.
Using API Calls

By default, the Agent generates code for debugging messages in the init method of the Main file as,

```java
int debugLevel = com.adventnet.agent.logging.Level.WARN;
```

This debug level can be changed by making WARN to whichever level you prefer.

20.4 Viewing Logging Reports

- As specified earlier, the log messages get stored in a text file named `agent.log`.
- This file gets created as soon the Agent is started and produces reports based on the specified debug level.
- **Log messages include debug levels equal to and lesser than the specified level.**
- These messages can be viewed from the `logs` folder in `<Agent Toolkit Home>/snmpProjects/projectName/agent/bin` directory.
- Once the Agent is stopped and restarted, the old log messages get shifted to `agent1.log` and the newly started Agent transfers its log messages to `agent.log`.
- This keeps continuing until `agent9.log`.
- Later, the log messages, except the last one, are zipped and placed under the `archives` folder of the logs directory.
- Thus, `agent.log` always shows the log messages of the recently started Agent.
21.0 Running Multiple Agents in a Single JVM

21.1 Overview
As the name indicates, this section guides you through the steps involved in running Multiple Agents in a single JVM. Starting a JVM for each Agent accumulates a lot of resources. Hence, it is not preferred. For accessing Multiple Agents in a single JVM, each Agent should be configured with different ports or different virtual IP addresses. It is also possible to simulate a network using Multiple Agents.

21.2 Running Agents With Virtual IP Addresses in a Single JVM
The first step towards running the Agent in Virtual IP addresses is to configure the virtual IP addresses.

21.2.1 Configuring Virtual IP Addresses
This section explains how to configure Multiple IP Addresses in varied OS such as Linux, Solaris, Windows NT, and Windows 2000.

In LINUX
You can use the "linuxconf" tool provided by Linux OS to configure virtual IP Addresses.

1. For setting multiple IP Addresses on an interface on a Linux system, you should have SUPER USER permission.
2. Log in to SUPER USER and start the "linuxconf" tool after starting X-windows. (just type linuxconf and press "Enter"). The following tree will be displayed.

```
+ config
  + NetWorking
    + ServerTasks
      IP Aliases for virtual hosts
  + Control
```
3. Enter at IP Aliases for virtual hosts.
4. From the two interfaces eth0/lo, assign the Virtual IP Addresses for physical interface eth0.
5. After completing the configuration, reboot your machine for the changes to take effect.
6. Upwards of 10,000 interfaces can be configured this way. If you are configuring a lot of interfaces it may take some time to boot up.
7. We recommend using two interface cards on your linux system, and using it as a router. Set the virtual interfaces up on the interface not connected to your LAN. From other systems on your LAN, you can set the linux system as the router to get to the newly configured networks. Solaris refers to these as logical interfaces.

In SOLARIS
- For Solaris you can use "ifconfig" utility to setup logical interfaces. See the ifconfig main page for more details.
- For making more aliases for an IP Address in Solaris, you should have the SUPER USER permission.
• Log in to SUPER USER and use "ifconfig" to configure the physical interface with different logical interfaces.

For example,

ifconfig hme0:1 132.253.154.0 netmask 255.255.255.0 up
ifconfig hme0:2 132.253.154.1 netmask 255.255.255.0 up

After the configuration, check whether the driver for the Virtual IP Addresses is activated using ping.

In WINDOWS NT

The steps involved in configuring virtual IP addresses in a Windows NT OS are as follows:

1. Click "Start" menu.
2. This displays the "Settings" option in it.
3. The "Control Panel" in settings menu provides the Network details.
4. From the network properties, choose "Protocols Tab". It lists various protocols used by the OS.
5. Select TCP/IP and view its properties.
6. The Properties tab of the protocol has an "Advanced" tab in it which on clicking, displays the IP Address of the machine.
7. The "Add" option helps to add more IP addresses to the machine.
8. Specifying the IP Addresses here would help you start the Agents in various IP addresses.

In WINDOWS 2000

Follow the steps given below to configure the IP Addresses in a Windows 2001 OS.

1. Click the Start menu.
2. This displays the Network and Dial-up connection in the Control Panel.
3. The dial-up connection has a New Connection facility and a Local Connection option.
4. Selecting Local connection option provides tabs that can be configured.
5. The Protocols tab in that option displays the protocols configured for the machine.
6. Clicking TCP/IP's Advanced properties provides an option for Adding.
7. This Add option helps to add more than one virtual IP address to the machine.

Activating the Agent in Virtual IP Address

Say you have configured virtual IP Addresses and created an Agent using AGENT-SAMPLE-MIB. The Main File of the Agent has the following code for the agent param options.

```java
public AdventNetSnmpAgent(String[] args){
    //this takes care of the options
    agentOptions = new AgentParamOptions( args);
    Thread th = new Thread(this);
    th.start();
}
```

On adding a similar code and specifying the entry for virtual IP addresses, the Agent gets started in virtual IP Addresses.
public AdventNetSnmpAgent(String ipAddress, String[] args){
    super(ipAddress);
    //this takes care of the options
    agentOptions = new AgentParamOptions(args);
    Thread th = new Thread(this);
    th.start();
}

Please note that the following piece of code has to be included in the Main File (towards the end) to register the virtual IP Addresses.

AdventNetSnmpAgent agent_sim3 = new AdventNetSnmpAgent("192.168.200.1", args);
AdventNetSnmpAgent agent_sim4 = new AdventNetSnmpAgent("192.168.200.2", args);

21.3 Running Agents With Multiple Ports in a Single JVM

For the Agent to run in different ports, add the following piece of code in the Main File generated.

public AdventNetSnmpAgent(int port, String[] args){
    super (port);
    //this takes care of the options
    agentOptions = new AgentParamOptions(args);
    Thread th = new Thread(this);
    th.start();
}

The following code should also be included towards the end of the main file for the Agent to run at different ports.

Say you have an Agent created using AGENT-SAMPLE-MIB. To start it from the ports 8001, 8002 and 8003 include the following code in the Main file.

AdventNetSnmpAgent agent_sim = new AGENT_SAMPLE_MIB(8001, args);
AdventNetSnmpAgent agent_sim1 = new AGENT_SAMPLE_MIB(8002, args);
AdventNetSnmpAgent agent_sim2 = new AGENT_SAMPLE_MIB(8003, args);
22.0 Failover Support for API Tables

22.1 Overview

In the real time environment there is a lot of possibility for the Agent to crash. At this moment a Standby Agent can be made to run, reading the values from the Main Agent. The Standby Agent can also be made to run along with the Main Agent and periodical updation of data is done in the Standby Agent, in this case. This avoids loss of data. Transferring the MIB data to the Standby Agent is possible. To transfer the data of the API Tables (like Trap Forwarding Tables, Proxy Tables, Access Control Table etc.,) which keeps changing frequently during runtime is quite difficult.

Agent Toolkit has a provision to get the API Table info from the Main Agent. For this purpose, a separate thread is started in the Agent and this thread takes the responsibility of updating the values of the API Tables in a Hash Table, within the Main Agent and also converting them to a byte array (serialized form) format. When the Main Agent crashes, the latest values of the API Tables are converted to a Byte Array (serialized form) and transferred to the Standby Agent. The Standby Agent de-serializes the data and stores it back in a Hash Table. Using the key (Table name), the values are identified and updated in the respective tables. To know how this provision can be enabled and how the API Table information are converted to a byteArray, go through the topics following.

22.2 Enabling the Failover Support for API Tables

To enable the Provision for getting the API Table Info, use the option available in MIB Compiler UI or the API calls.

Using MIB Compiler UI
1. Create a Project and Load a MIB.
2. Go to Project -> Settings menu in the MIB Compiler UI.
3. Choose the General Panel.
4. Enable Generate Code to get APITablesInfo option. This enables the Provision.

Using API Calls

Enabling the UI option, generates the following code in the Main File of the Agent toward the end of the file. Adding this code in the generated Main file without enabling the UI option also enables the provision to get the api table information.

```
Note: By default View Based Access for v1/v2c Agents (vaclTable) is not implemented in the Agent. Hence, ensure whether the vaclTable implementation is made in the Agent before adding the code for transferring the details of vaclTable to the Hash Table. This can be checked in the viv2Authentication -> vaclTable Panel in the MIB Compiler UI. (Project -> Settings menu).

/**
 * This method is to make avail of the Provision.
 * You have to call this method Periodically in a Separate Thread
 * and Update the Standby Agent, so as to avoid the API Table data.
 * You also have to instrument the Transferring portion of the
 * API Table details to the Standby Agent.
 */
```
public void getAPITablesInfo()
{

    Hashtable apiTablesHash = null;
    boolean isAPIModified = false;

    if (aclTable.isModified()){
        if (apiTablesHash == null){
            apiTablesHash = new Hashtable();
        }

        apiTablesHash.put("AclTable", aclTable.getTableVector());
        isAPIModified = true;
        aclTable.setModified(false);
    }

    if (vaclTable.isModified()){
        if (apiTablesHash == null){
            apiTablesHash = new Hashtable();
        }

        apiTablesHash.put("VaclTable", vaclTable.getTableVector());
        isAPIModified = true;
        vaclTable.setModified(false);
    }

    if (forwardingTable.isModified()){
        if (apiTablesHash == null){
            apiTablesHash = new Hashtable();
        }

        apiTablesHash.put("V1V2TrapForwardingTable",
                        forwardingTable.getTableVector());
        isAPIModified = true;
        forwardingTable.setModified(false);
    }

    if (isAPIModified){
        return convertHashtableToByteArray(apiTablesHash);
    }
    return null;
}

/**
 * This method is for Converting the API Tables Info Hashtable
 * in to a Byte[] format so that, it can be transferred easily.
 * @param apiHash - The Hashtable with the API table details.
 * @return - Byte[] format of the apiHash.
 */

public byte[] convertHashtableToByteArray(Hashtable apiTablesHash){
    try{
        ByteArrayOutputStream byteOut = new ByteArrayOutputStream();
        ObjectOutputStream out = new ObjectOutputStream(byteOut);
        out.writeObject(apiTablesHash);
        out.flush();
        out.close();
        return byteOut.toByteArray();
    }
    catch(IOException ioe){
        System.out.println("Exception while writing");
    }
    return null;
As per these codes the Existing Agent keeps updating a Hash table with the API Table values periodically. Periodical updation is done only if the values of the API tables are modified. Else the method will be called but no updation is done in the Hash Table.

On Agent crash, the Hash Table details are converted to a Byte Array (serialized format) and are made available to the Standby Agent. The byte Array details can be transferred to the standby Agent as you prefer. i.e., the transfer method or communication part purely lies on the hands of the user.

**22.3 Updating the API Table values in the Standby Agent**

The Standby Agent that receives the ByteArray converts it back to a Hash Table (de-serializes). The Hash Table information uses the key or the Table name for identification. Say for example: the key accessControlTable indicates that particular byteArray represents the values of the Acl Table. The values get updated accordingly. Similarly the keys for other tables like vaclTable and Trap Forwarding Tables are:

The following method takes the values from the Hash table in the Standby Agent and updates it in the respective tables.

```java
public Hashtable convertByteArrayToHashtable(byte[] apiTablesArr){
    try{
        ByteArrayInputStream byteIn = new
        ByteArrayInputStream(apiTablesArr);
        ObjectInputStream ois = new ObjectInputStream(byteIn);
        return (Hashtable)ois.readObject();
    }catch(IOException ioe){
        System.out.println("Exception while creating the Input Stream");
        }catch(ClassNotFoundException cce){
        System.out.println("Exception while downcasting the byte array");
    }
    return null;
}
```
23.0 Plugging In Other Transport Protocols

23.1 Overview

SNMP standards define UDP to be used as the default protocol for communication. UDP, being a connectionless protocol, is most suitable for exchanging messages with smaller packet sizes and also where the Network Management System requires to do quick queries on a lot of devices.

When the packet sizes are large like in the case of database retrievals and updates, TCP might be the preferred protocol. You might also want to choose your own protocol for implementation such as IPX, HTTP, TCP, Serial port communication, etc. You can also implement SSL below the SNMP for secured communication.

To facilitate this, a new transport provider framework is provided, whereby you can plug in your own transport protocol. This enables you to implement the protocol of your choice for SNMP communication. The API implements UDP/IP as the default protocol implementation. Also, TCP/IP is provided as the reference implementation that uses the transport provider framework.

The transport provider is responsible for all communication between the SNMP Manager and Agent and uses the configured protocol to do so. So, it essentially acts as a bridge between the SNMP API and the actual (underlying) transport protocol used.

The advantage of using this approach is that the SNMP API need not be aware of the underlying protocol used and you can virtually run SNMP over anything. For using a particular protocol as the transport, you have to implement that protocol and plug in (or register) it with the SNMP transport provider. Two transport protocols can be registered with the provider at a time, i.e., you can use TCP and UDP protocols for communication at the same time.

We have provided a reference implementation of SnmpAgent on TCP/IP protocol over the SNMP transport provider.

23.2 APIs Used for Plugging In Your Protocol

These interfaces are a part of the com.adventnet.snmp.snmp2 package and are used for plugging your protocol.

SnmpTransportProvider

This interface provides the basic Input-Output (I/O) operations that can be expected by any transport protocol. It contains the following API methods (Please see the javadocs files for the exact syntax).

- open() - Opens the transport interface through which the data is sent/received.
- close() - Closes the transport interface after communication is over.
- read() - Receives data from the peer through the transport interface.
- write() - Sends data to the peer through the transport interface.

ProtocolOptions

This interface defines some parameters that have to be implemented by the user. It contains the following API method (Please see the javadocs files for the exact syntax):
getSessionId(): Returns a unique sessionId for each SnmpSession. This is used by the SnmpV3 module and need not be implemented by those using Snmp V1 and Snmp V2.

SnmpTransportPacket

This class contains the details of the ProtocolOptions, which is used by the SnmpAPI and the details of data, i.e., the SNMP message that is sent or received. This contains the following API methods (Please see the javadocs files for the exact syntax):

- getProtocolOptions() - Returns the transport parameters set on the transport packet for sending/receiving messages.
- setProtocolOptions() - Sets the protocol options on the SnmpTransportPacket.
- getProtocolData() - Gets the SNMP message to be sent or received.
- setProtocolData() - Sets the protocol data on the SnmpTransportPacket.

23.3 Plugging the Protocol in SNMP Transport Provider Framework

Provide implementation of SnmpTransportProvider for the protocol (default implementation class is com.adventnet.snmp.snmp2.agent.AgentTcpTransportImpl) - This class should provide the open(), close(), read(), and write() implementation required for TCP/IP communication. This can be used to set up the basic communication and transfer data between the SNMP entities.

The transport provider option can be enabled in the SnmpAgent by setting the protocol as SnmpAgent.TRANSPORT_PROVIDER. The code looks like this:

```java
snmpagent.setProtocol(SnmpAgent.TRANSPORT_PROVIDER);
```

Similarly, to use the transport provider for SnmpTrapService, the code should look like this:

```java
snmptrapservice.setProtocol(SnmpAgent.TRANSPORT_PROVIDER);
```

Provide implementation of ProtocolOptions - This class should provide the necessary parameters needed for the protocol communication, e.g., the host, port, etc. in case of TCP/IP. In the case of V3 module, this class should contain the implementation for getSessionId(). This is used by the SNMP V3 module to uniquely identify each session connection between the SNMP entities. So, for TCP/IP, this can be a combination of the destination host and port. The default implementation class used by the SnmpAgent class is com.adventnet.snmp.snmp2.agent.AgentTcpProtocolOptionsImpl. If you wish to have your own implementation of this protocol options, you need to override the method createProtocolOptions in SnmpAgent.

```java
public ProtocolOptions createProtocolOptions(String remoteHost, int remotePort, int localPort) {
    return new MyProtocolOptions(remoteHost, remotePort, localPort);
}
```

Similarly, for traps to be sent using the transport provider framework, the user needs to override the method createProtocolOptions in SnmpTrapService.

Provide a configuration file in the name snmpTransport.config, which contains the name of the implemented SnmpTransportProvider class file.

To avail the reference TCP implementation, change the implementation class name in snmpTransport.config in the <Agent Toolkit Home>/jars/AdventNetSnmp.jar file from com.adventnet.snmp.snmp2.TcpTransportImpl to com.adventnet.snmp.snmp2.agent.AgentTcpTransportImpl.
24.0 Working with JBuilder

Please follow the steps given below to compile an Agent using JBuilder 6.0.

1. Start the MibCompiler application from `<Agent Toolkit Home>/bin` directory and create a new project. For example, give the name of the project as "snmptest" and the project location as `snmpprojects`.

2. You can also select other required options or use the default options from the Project -> Settings menu bar of MIB Compiler UI.

3. Now load AGENT-SAMPLE-MIB and generate source code for the project. The source files will be generated under `<Agent Toolkit Home>/snmpprojects/snmptest/agent/src/com/myCompany/myPackage` directory, in the drive where the product is installed.


5. The Project Wizard interface opens. Here, enter the Project name and the project directory name for the JBuilder project. For example, in the screenshot given below, the Project name is given as `Agent` and the directory as `<Agent Toolkit Home>/snmpProjects/Agent`. This means, the JBuilder Project will be created in `<Agent Toolkit Home>/snmpProjects/Agent` directory.

![Project Wizard - Step 1 of 3](image-url)
6. After entering all the options as specified, select the Next tab. This will go to the next screen, which will list the settings of your new project.

7. In this screen, modify the Source path and Output path as specified in the screen-shot below. Please note that the source path is the path where the source files have been generated by the MibCompiler project, as specified in Point no.2.

8. Now, add the Required libraries as specified in Point no.5.

![Project Wizard - Step 2 of 3](image)

### Specifying Project Paths

<table>
<thead>
<tr>
<th>Specification</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>JDK</td>
<td>java 1.3.1-b24</td>
</tr>
<tr>
<td>Output path</td>
<td><code>\</code>:\AdventNet\JavaAgent\snmpProjects\snmpTest\myAgentBinaries`</td>
</tr>
<tr>
<td>Backup path</td>
<td><code>\</code>:\AdventNet\JavaAgent\snmpProjects\Agentbak`</td>
</tr>
<tr>
<td>Working directory</td>
<td><code>\</code>:\AdventNet\JavaAgent\snmpProjects\snmpTest`</td>
</tr>
</tbody>
</table>

### Required Libraries

<table>
<thead>
<tr>
<th>Default</th>
<th>Test</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><code>E:\products_builds\4.2_product\AdventNet\JavaAgent\snmpTest</code></td>
</tr>
</tbody>
</table>
9. To add the required libraries, select the Add option which is available in the Required libraries of the above screen-shot. This will prompt the following screen.

10. Select the New option available at the bottom of the screen.

11. Now, you will be prompted with the New Library Wizard as shown below. Here, provide the name of the new library, for example adventnet_jars.

12. Select the Add option to add the library paths. You will be prompted with a UI, from where you can browse and select the required jar files from h:/<Agent Toolkit Home>/snmpProjects/snmptest/jars directory and press OK.
13. This will add the jar files as shown in the screen-shot below. Now, the adventnet_jars will be added in the Required Libraries option as shown in the screen-shot of point 4.
14. Now, select the next tab. This will prompt you with the following screen where you can enter the following details, which are optional.
15. Now, select the Finish button, which will prompt you with the next screen. Here, select
snmpAgent.jsp, right click and select the option "Add files/Packages" as shown in the screen-
shot below.
16. From the listed directories, select all the files under `<Agent Toolkit Home>/snmpprojects/snmptest/myAgentSources/com/mycompany/mypackage` and press O.K. This will add the files in the tree as shown in the screen-shot below.

17. Now, select the `Project -> Rebuild Project` option, to compile your project. If you get any warning messages, please ignore.

18. Select the `Run -> Run project` option, to run the project. In the runtime properties dialog, select the main class as `AdventNetSnmpAgent` and press O.K as shown in the screen-shot.

19. Now, the Agent is started at port 8001. You can test your Agent by loading `AGENT_SAMPLE_MIB` using the MIB Browser application available in `<Agent Toolkit Home>/bin` directory.
25.0 Embedded JVM Support

25.1 Overview

Embedded JVMs are preferred in machine critical applications requiring less memory footprint, with a limited set of APIs. AdventNet agents now support a variety of Embedded JVMs that are compliant to Personal Java Specification released by Sun Micro Systems. This is achieved by AdventNet Agent Toolkit, as it is compliant to Sun's JDK1.1.6.

The basic idea here is that, the Agent Toolkit can be used to build an agent and bundle it as a Jar and this jar is given as an input to the Embedded JVM to build an executable file, targeted for a specific OS on a specific hardware.

AdventNet officially supports compliance to the following JVMs.

- Newmonics (PERC VM)
- GCJ (GNU)
- IBM sualAgeME

Note: It is possible to run any AdventNet Agents in Embedded JVM compliant to Personal Java specifications.

25.2 Newmonics Support

Here is the procedure to create an executable SNMP Agent using the NewMonics Perc3.1 package,

- Generate an SNMP Agent with the MIB Compiler
- Generate the jar for the Agent like "myAgent.jar"
- Place the following jars in the <Perc3.1>lib directory,
  - AdventNetSnmpAgent_NEWMMONICS.jar
  - myAgent.jar
  - jaxp.jar
  - xalan.jar
  - crimson.jar
- Add the above jars in the PERC_CLASSPATH environment variable
- Edit the image file in the <PERC3.1>examples\WinNT directory with the Agent's main file class, like com.myCompany.myPackage.AGENT_SAMPLE_MIB
- Edit the command caller Romizer to the following line at line no: 56 in the Makefile present in the <PERC3.1>examples\WinNT directory
  - cmd /c romize -target X86 -os Windows -v imagefile
- Then run nmake command from commandline from the <PERC3.1>examples\WinNT directory.
After successful completion of the nmake command, a pvm.exe will be created in the \PERC3.1\examples\WinNT directory

The Agent can be executed with the command pvm com.myCompany.myPackage.AGENT_SAMPLE_MIB

25.2.1 Limitations

The main limitations of using NEWMONICS to produce an executable SNMP Agent are:

- No SNMP V3 Agent support
- There is no security packages support in PERC3.1 so SNMP V3 Agents are not supported
- Object serialization and de-serialization is not supported in PERC3.1 so SNMP V3 Agents are not supported
- Single Agent Beans are not supported since Beans are not supported in PERC3.1

25.3 IBM Visual Agent Micro Edition (VAME) Support

The procedure to create an SNMP Agent using the VAME IDE version 1.3 is :-

- The VAME IDE at startup, asks to open an existing Workspace or create a new Workspace,
  o Tell the destination of the new workspace and click OK.
- Add a new project as a library project and name it as JCL_CORE
  o Select the JCL_CORE project and select, Import menu-item from the Selected menu.
  o In the upcoming dialog box, select Jar/Zip file and specify the location <VAME>\ive\jclMax\classes.zip.
  o Select in the Import files as option as Jar (requires project with no packages) and click Finish.
- Add a new project as a library project and name it as DUMMY_APPLET
  o Create a Dummy java.applet.Applet class with package java.applet and create a DummyApplet.jar
  o Select the JCL_CORE project and select, Import menu-item from the Selected menu.
  o In the upcoming dialog box, select Jar/Zip file and specify the location <YourImplementation>\DummyApplet.jar
  o Select in the Import files as option as Jar (requires project with no packages) and click Finish.
- Add a new project as a library project and name it as SNMP_CORE
  o Select the SNMP_CORE project and select, Import menu-item from the Selected menu.
  o In the upcoming dialog box, select Jar/Zip file and specify the location <Agent Toolkit Home>\jars\AdventNetSnmp.jar
  o Select in the Import files as option as Jar (requires project with no packages) and click Finish.
- Add a new project as a library project and name it as SNMP_AGENT_CORE
  o Select the SNMP_AGENT_CORE project and select, Import menu-item from the Selected menu.
  o In the upcoming dialog box, select Jar/Zip file and specify the location <Agent Toolkit Home>\jars\AdventNetSnmpAgent.jar
  o Select in the Import files as option as Jar (requires project with no packages) and click Finish.
• Add a new project as a library project and name it as **JAXP**
  - Select the JAXP project and select, Import menu-item from the Selected menu.
  - In the upcoming dialog box, select Jar/Zip file and specify the location `<Agent Toolkit Home>/jars/jaxp.jar`
  - Select in the Import files as option as Jar (requires project with no packages) and click Finish.

• Add a new project as a library project and name it as **XALAN**
  - Select the XALAN project and select, Import menu-item from the Selected menu.
  - In the upcoming dialog box, select Jar/Zip file and specify the location `<Agent Toolkit Home>/jars/xalan.jar`
  - Select in the Import files as option as Jar (requires project with no packages) and click Finish.

• Add a new project as a library project and name it as **CRIMSON**
  - Select the CRIMSON project and select, Import menu-item from the Selected menu.
  - In the upcoming dialog box, select Jar/Zip file and specify the location `<Agent Toolkit Home>/jars/crimson.jar`
  - Select in the Import files as option as Jar (requires project with no packages) and click Finish.

• Add a new project for the Agent classes as **AGENT**
  - Select the AGENT project and select Import menu-item from the Selected menu.
  - In the upcoming dialog box, select Directory file and specify the location.
  - Select in the Import files as option as Individual files and click Finish.

• Select Properties menu-item from the Workspace menu
  - In the popup select the Class Path Tab.
  - Select **JCL_CORE** and Select the execution path to be `<VAME>/ivc/jclMax/classes.zip` and move it up so that it appends itself to the Boot Classpath settings.
  - Select **DUMMY_APPLET** and Select the execution path to be `<YourImplementation>/DummyApplet.jar` and make sure this remains at the top of the Application Classpath settings.
  - Select **SNMP_CORE** and Select the execution path to be `<Agent Toolkit Home>/jars/AdventNetSnmp.jar`
  - Select **SNMP_AGENT_CORE** and Select the execution path to be `<Agent Toolkit Home>/jars/AdventNetSnmpAgent.jar`
  - Select **JAXP** and Select the execution path to be `<Agent Toolkit Home>/jars/jaxp.jar`
  - Select **XALAN** and Select the execution path to be `<Agent Toolkit Home>/jars/xalan.jar`
  - Select **CRIMSON** and Select the execution path to be `<Agent Toolkit Home>/jars/crimson.jar`

• Select **Workspace-->Full Build** to do the build.

• Now select the Agent project, **AGENT**, and then select **Smartlinker-->Generate options** menu-items from Selected menu.

• In the pop-up specify the name of the JXE to be created and click Finish.

• Now the error output will show a missing of Java security packages and some classes such as `java.sql.*` are not found, these errors are known errors which will not influence the Agent's runtime performance.
Now, the Agent can be started from this directory, 

```bash
j9 -ms:32 -jxe:<NEW_WORKSPACE>Caches\Target\Agent\SmartLinker\JXE_GENERATED\JXE_GENERATED.jxe <options>
```

### 25.3.1 Known Issues

- The XML Storage facility in the Agent, makes VAME throw an Unsupported Encoding Exception, but this does not restrict the Agent's performance.
- In some cases, VAME throws a Class Not Found Exception on crimson.jar classes. If the agent is developed without XMLToVector persistence this error can be prevented.

### 25.3.2 Limitations

- Vame does not support Security packages and hence SNMP V3 Agents are not supported.

### 25.4 GCJ Compiler Support

The GNU Compiler for Java or GCJ compiler is a portable, optimizing, ahead-of-time compiler for the Java Programming Language. It can compile:

- Java source code to native machine code,
- Java source code to Java bytecode (class files),
- Java bytecode to native machine code.

Since 4.2 release, support for GCJ compilation is included for the AdventNet Agent Toolkit. Developers can use this support for developing native run-time SNMP agents. The jar that is bundled for supporting GCJ is AdventNetSnmpAgent_GCJ.jar This jar can be found in the <Agent Toolkit Home>/jars directory.

This jar contains AdventNet's implementation of java.applet.Applet, com.adventnet.snmp.snmp2.SASClient, and other classes for GCJ compilation. These classes provide a work around for those java classes that are not supported by the GCJ compiler (like the java.applet.Applet). The jar also includes classes for serializing USM and VACM Tables which is necessary for SNMPv3 support.

The GCJ compilation libraries - libgcj and the GCJ compiler itself. The libgcj consists of;

- The core class libraries.
- A garbage collector library.
- An abstraction over the system threads.
- And, optionally, a bytecode interpreter.

This can be downloaded from http://sourceware.cygnus.com/java.

Extracting the AdventNetSnmpAgent_GCJ.jar : Since the GCJ compiler does not look into jar files for the classes, the AdventNetSnmpAgent_GCJ.jar file has to be manually extracted. This can be done using the JDK's jar tool in any one of the following ways:

#### 25.4.1 Environment Setting for GCJ Compilation

There is a file called snmp.properties in the directory (classes/com/adventnet/snmp/snmp2/). This file has to be edited to have the option called GCJ_COMPILE which needs to be set equal to 1. The following line has to be present in the snmp.properties file.

```
GCJ_COMPILE = 1
```

The default value of this option in the AdventNetSnmpAgent_GCJ.jar is 1. Compiling with the AdventNet SNMP Low Level API. The following diagram illustrates how the user could develop a native SNMP Agent application using the AdventNet API with the GCJ compiler.
Step I - To create a native library of the AdventNet agent package

All the classes are given to the GCJ compiler to get the SNMP Agent libraries. Either a shared or a static library archive can be created. The shared library (eg libsnmp2.so) will be created by the GCJ compiler itself. To create a static library the classes should be converted first to object files (extension .o files) using GCJ. Then a library archive (e.g., using the ar archive command) of the object files can be created. For example:

To create a Shared Library (in Unix based systems):

```
gcj -shared classes classes -o libsnmp2
```

So, this will create a shared library file where the classes are the classes that were extracted from the jar.

To create a Static Library (in Unix based systems):

Create the object (.o) files

```
gcj -c classes
```

Create the object files archive created in step 1

```
ar rv libsnmp2.a *.o
```

Create the library

```
ranlib libsnmp2.a
```

Step II - Creating the executable of the SNMP application that uses the AdventNet's package. The user application could be compiled and linked with the snmp2 library (created in step 1) to produce the native Application. For example

```
gcj -lsnmp2 user classes or java source files -o UserExecutable
```

This will create the native executable called UserExecutable linked with the libsnmp2 library created in Step 1. The user classes or java source files are the classes or .java files of the user's Application.

25.4.2 Known Issues in the GCJ Compiler

- It does not work with gcj 2.95.1 release. This is because of a problem with receive() methods in Datagram sockets when used inside threads. The application hangs indefinitely inside the receive() call. Also the library does not have a full implementation of the java.security package which is needed in case of SNMPv3.
  - WorkAround: It is better to use the latest cvs or the latest rpm version of the compiler.

- Latest RPM and CVS versions of the compiler works only for the noAuthNoPriv security level in SNMP v3. The authPriv and authNoPriv security levels in SNMPv3 version cannot be supported since the libgcj library (from the latest rpm and cvs) does not have an implementation of the MD5 algorithm by default (as in JDK) which is used for calculating digests of the SNMP messages for these levels. At run time this will result in NoSuchAlgorithmException thrown since the MD5 algorithm will not be found. Also for Privacy we require the DES encryption which is not available in the library.
  - WorkAround: Third party implementations of these algorithms (sources) can be downloaded and integrated with the libgcj library source and compiled. There is an implementation of Message Digest and DES encryption from Cryptix and this can be downloaded for freely from http://www.cryptix.org.
26.0 Packaging and Starting Agent as a Service

26.1 Overview
Once the agent is deployed, it is ready for packaging. Please follow the steps given in this section to know how to package an SNMP agent.

26.2 Jars required for Packaging an SNMP Agent
To package an SNMP Agent, the following jars are required under the `<Agent Toolkit Home>/jars` directory:

- The compiled class files generated by the MIB Compiler in the output (bin) directory.
- AdventNetSnmp.jar
- AdventNetSnmpAgent.jar
- AdventNetAgentRuntimeUtilities.jar
- AdventNetAgentUtilities.jar
- AdventNetLogging.jar
- XML-related-jars [When XML Storage option is chosen for Scalars and Tables in MIB Compiler UI]
  - crimson.jar
  - jaxp.jar
  - xalan.jar

26.3 Running the Agent in Any Environment
The Agent packaged and deployed can be run provided the above-said jars are available in the `<Agent Toolkit Home>/jars` directory. To run the Agent in any environment, these jars need to be placed under the specified Agent's directory. For this purpose, a Package Agent option is provided using which all the jars are copied to the Agent directory.

To enable the same,
1. Select the Build menu from the menu bar of MIB Compiler UI.
2. Click Package Agent option to avail the facility.
3. The jars directory is copied under the `<Agent Toolkit Home>/snmpprojects/projectname/agent` directory with the required jars.

26.4 Packaging the Agent as a Service
Agent Toolkit supports configuring the developed Agent as a service. By installing this service, AdventNet Agent gets started automatically in the specified port. You need not start the agent every
time you log into the system. The steps to be followed in configuring the agent as service are as follows:

### 26.4.1 Installing the Service

Before moving into installation steps, let us see how any application can be run using JavaService.

JavaService can be used to run any Java class as a Service in Windows. It can start the service using the standard static void main(String[] args) method, or any other static method that takes a String[] as its only parameter. In its simplest incarnation JavaService will simply terminate the JVM to stop the service, but you can optionally have it call another static method inside the JVM. If JavaService calls another method it will wait either until both the start and the stop methods have returned or until a 30 second timeout has expired before terminating the JVM. JavaService can also optionally redirect the System.out and/or the System.err streams into a file.

The JavaService executable has no dependencies on any files or directories. This was intentional so that it could easily be used with any Java software. Additionally, it does not have a dependency on the name JavaService.exe. This means that you can, and probably should, rename it to something more appropriate. This can be quite useful when checking the Task Manager to see how much CPU time a particular service is using, or how many threads it has created. Instead of seeing multiple entries for JavaService.exe you see a unique entry for each renamed version of the executable. Follow these instructions to use JavaService with your Java application:

1. Copy JavaService.exe packaged with Agent Toolkit (in `<Agent Toolkit Home>/subagents/NTutils directory) into a directory appropriate for your application.
2. You can optionally rename JavaService.exe to something more appropriate.
3. Start a command prompt and change to the directory where the JavaService.exe is located.
4. Type 'JavaService.exe -install' followed by the proper parameters, as described below. Be sure to use quotation marks around parameters with spaces (italics signify an application-dependent value):

   - **service_name:** The name that you want to use for this service. This is what the service will show up as in the service control manager.

   - **jvm_library:** The location of the jvm.dll file that you want to use as your Java Virtual Machine. For Sun's Java 2 SDK, this is usually `{JDK_HOME}/jre/bin/classic/jvm.dll` or `{JDK_HOME}/jre/bin/hotspot/jvm.dll`.

   - **jvm_option:** Specify any necessary parameters to pass to the JVM upon invocation. These may include parameters such as "-Djava.class.path=" to specify a classpath. Any parameters that you need to use when invoking the java.exe command tool should be specified here. There is no limit to the number of parameters specified. For example,

     - `-Djava.class.path=<JavaAgent Home>\snmpprojects\myproject\agent\bin;` (also give all the required jars in class path).

   - **start start_class:** The name of the class that you wish to use when starting the service. This must be the fully qualified class name. For example,

     - `-start com.myCompany.myPackage.AdventNetSnmpAgent`

   - **out out_log_file (optional):** A file into which System.out will be redirected. If this parameter is not specified, System.out will not be redirected.
**SNMP Agent**

**-err err_log_file (optional):** A file into which System.err will be redirected. If this parameter is not specified, System.err will not be redirected.

**-current current_dir:** A directory to use as the current working directory for the service. If this parameter is specified, all relative paths in the service will be relative to the specified directory.

For example,

- `current <JavaAgent Home>\JavaAgent\snmpprojects\myproject\agent\bin`

**-path extra_path (optional):** An addition to the path for the service. The specified path will be appended to the system path before the service is started. This can be used to specify where additional DLLs that native libraries are dependent upon can be found.

```java
//A simple example might look like the following
C:\users\name\service\bin>JavaService.exe -install
"AgentService" c:\JDK\jdk1.3.1_02\jre\bin\hotspot\jvm.dll
-Djava.class.path=<JavaAgent Home>\snmpprojects\myproject\agent\bin;(also append the required jars here) -start
com.myCompany.myPackage.AdventNetSnmpAgent -out
c:\user\js.out -err c:\user\js.err -current <JavaAgent Home>\snmpprojects\myproject\agent\bin
```

### 26.4.2 Starting the Agent as a Windows Service

Once the service is installed,

1. Go to Control Panel -> Services menu in your Explorer.
2. "Start" AdventNet Agent Service from the list. Your agent will be started at the port which was configured while generating the sources.

### 26.4.3 Uninstalling the Service

1. Start a command prompt and change to the directory where the JavaService.exe is located.
2. Type 'JavaService.exe -uninstall' followed by the name of the service.

### 26.4.4 Testing the Agent Started as a Service

You can test the service Agent by sending queries to the Agent.

- Load the MIB (using which the Agent was created) in MIB Browser application.
- Send a GET request to the Agent.
- The agent has to respond with the values.

### 26.4.5 Event Logging

JavaService logs both informational events and errors into the Application portion of the NT Event Log. Informational messages include a message when the service is started and one when it is stopped. Error messages may be logged because of a configuration problem or other system error. If you are having trouble getting JavaService to run your application, you should first check the Application portion of the Event Log to see if any errors were reported. Additionally you can check the redirected System.err file, as any Java exceptions will be written there by the Virtual Machine.
27.0 Building Multi-Protocol Agent with SNMP Support - SNMP Adaptor

The AdventNet Agent Toolkit supports Multi-Protocol Agent development. The agent developed can be accessed by SNMP, HTTP, TL1, RMI, CORBA and HTML adaptors. The SnmpAdaptor class in com.adventnet.adaptors.snmp package acts as adaptor for SNMP to make the Multi-Protocol agent accessible from the management applications (for e.g., AdventNet MIB Browser) outside the Agent's JVM.

As per the multi-protocol architecture, an agent created and registered in a MBean server is accessible in adaptors/connectors like SNMP, CORBA, RMI, CORBA, HTML, and HTTP. The instrumentation done for any specific function, is common for all, though utilized in various protocol adaptors. Apart from an enhanced version of the Standalone SNMP Agent, the common instrumentation also provides support for the SNMP Protocol.

Please refer to the SNMP Adaptor available in Building Multi-Protocol Agents section to learn about the steps involved in creating the Multi-Protocol agent with SNMP support using JMX Compiler and testing it using MIB Browser.
28.0 SNMP FAQs

Some of the Frequently Asked Questions in:

- General
- MIB Editor
- MIB Compiler
- MIB Browser
- Proxy
- Traps
- SNMPv3
- Authentication and Authorization (vacl)
- Table Handling
- OS Subagents
- Rule Engine
General FAQs

1. What is a MIB?
A MIB (Management Information Base) is defined as a structured collection of data variables called objects. Each object represents some resource to be managed. Each system in a network maintains a MIB that reflects the status of the managed resources.

2. What are the SNMP versions supported by the Agent?
The Agent supports SNMPv1, v2c and v3 versions.

3. When processing a Set request, does the AdventNet SNMP Agent support the atomicity requirement?

4. Can you tell me how/where we can register to get our own enterprise number?

5. How is the selection of SNMP V2 Compliance/SNMP V3 Compliance related to Version V1/V2/V3 option in Project -> Settings menu of MIB Compiler UI?

6. If we choose to use the AdventNet Agent Development Kit (Java) to develop our network management system, what other language bindings are available to us? Do you support interfaces to C/C++?

7. What are variable bindings?

8. Say that ifTable has as its indexes ifTableType and ifTableSubType. Lets assume that for a particular row, those values are "11245" and "4543", respectively. Would you do a GET like: .ifTable.ifEntry.ifColumnIWant.11245.4543?

9. What do I need if I want to install a minimal version of the agent toolkit with only the MIB browser?

10. Can an Agent have multiple clients (e.g. EMS + direct SNMP monitoring access)?

11. Can you, please, tell me if your SnmpTimeticks class can successfully convert a long value? I'm having issues with it.

12. I know there are some fields inside the getResponse packet like "error Messages" that are SNMP standard (for example "bad value" etc.). Can I insert through your Snmp API my own message inside this field? For example, the SNMP agent should return the Snmp Get Response with the error field named as "Frequency SET operation Failed !"

13. How can I compile the agent in JBuilder 6.0?

14. Is AdventNet support Solaris as development plat form?

15. Can we know the method to create the *.class files without the toolkit GUI. As far as we know the toolkit uses the javac-compiler. Can you give us some information about the environment the toolkit uses??

16. What are Maximum Repetitions and Non-Repeaters? Where do I have to specify these values.?

17. Is it possible to generate code for Multiple MIBs? If so, how?

18. I've upgraded to the latest release of the java SDK. I uninstalled the previous version and how do i run the Agent Toolkit with the new version of java?

19. While trying to perform an SNMP GET for the "sysUptime" parameter of RFC-1213 of the SNMP Agent, it doesn't get updated. Why?
3. When processing a Set request, does the AdventNet SNMP Agent support the atomicity requirement?

The AdventNet SNMP Agent supports the atomicity requirements for processing a Set request with multiple varbinds.

4. Can you tell me how/where we can register to get our own enterprise number?

You have to register with the Internet Assigned Numbers Authority (IANA), to get your own unique enterprise number. Here is the contact info for IANA:

   Internet Assigned Numbers Authority
   4676 Admiralty Way, Suite 330
   Marina del Rey, CA 90292
   USA
   +1-310-823-9358 x20 (voice/mail)
   +1-310-823-8649 (facsimile)
   iana@iana.org

   You can also look into http://www.iana.org/ and click on Protocol Numbers and Assignment Services link. In that click on E and click on Enterprise Numbers link for seeing the Enterprise Numbers listing or click on Online Application for a Private Enterprise Number for registering. http://www.iana.org/numbers.html

5. How is the selection of SNMP V2 Compliance/SNMP V3 Compliance related to Version V1/V2/V3 option in Project -> Settings menu of MIB Compiler UI?

The "Version" in Project -> Settings menu of MIB Compiler UI allows you to create an agent of your choice. v1, v2c and v3 are the standard versions of SNMP. By default, the v3 option is checked and the Agent is generated as a SNMPv3 agent. You can also create a SNMPv2c agent or a SNMPv1 agent.

On the other hand, any agent created as a v2c agent is by default V2 compliant. Meaning the standard RFC 1907 specifies some compliances to be implemented if an Agent claims as an SNMP V2 Agent. The Agent will be compliant to the following items:

   SNMP Group Counters
   System Group
   SNMP Notification Group
   sysORTable

However if the option is unchecked the necessary codes for the SNMP V2 compliance won't be generated. This holds good for SNMP V3 compliance also. But the RFC's implemented for v3 are different. Please refer SNMP v2 compliance and SNMP V3 compliance section for detailed information. And with regard to Compliance, the agent can be v3 compliant only when the agent is of versionv3.

6. If we choose to use the AdventNet Agent Development Kit (Java) to develop our network management system, what other language bindings are available to us? Do you support interfaces to C/C++?

Using JNI, you can interact with C, C++. You can also refer to the Window-System-Example bundled in examples directory section for detailed explanation on this.
7. What are variable bindings?

The Variable Bindings (or varbinds for short) are the actual data that is transported back and forth inside SNMP messages. This data is stored in the varbinds list of each message.

They are so named because they bind an OID name with a value. The OID is that of a single managed object defined in a MIB. The value may be any of the ASN.1 scalar data types or the NULL value if the value field is not used. The size of each binding and the number of bindings that may be included in any PDU is variable.

8. Say that ifTable has as its indexes ifTableType and ifTableSubType. Lets assume that for a particular row, those values are "11245" and "4543", respectively. Would you do a GET like: .ifTable.ifEntry.ifColumnIWant.11245.4543?

Yes. The GET operation should be done in the manner you have mentioned. To GET a particular column in a particular row you will have to specify all the index values in the same order.

9. What do I need if I want to install a minimal version of the agent toolkit with only the MIB browser?

You can download the SNMP Utilities product from our website. This has the MIB Browser and Trap Browser applications only.

10. Can an Agent have multiple clients (e.g. EMS + direct SNMP monitoring access)?

Yes, any number of Manager can access the Agents. There is no restriction as to how many managers can access an agent.

11. Can you, please, tell me if your SnmpTimeticks class can successfully convert a long value? I'm having issues with it.

Here I am explaining SnmpTimeTicks behavior.

For example if you create a SnmpTimeticks object for the long value 100 then the corresponding timetick value will be (using toString() method)

0 Hours, 0 minutes, 1 seconds

For 6000

0 Hours, 1 minutes, 0 seconds

For 6100

0 Hours, 1 minutes, 1 seconds

i.e. A timeticks value has the unit of hundredth of a second

12. I know there are some fields inside the getResponse packet like "error Messages" that are SNMP standard (for example "bad value" etc..). Can I insert through your Snmp API my own message inside this field? For example, the SNMP agent should return the Snmp Get Response with the error field named as "Frequency SET operation Failed !"

You can send your own ErrorStatus value (but not "error message") from the agent but the standard manager application would not know any thing about your own error status value in the response PDU. So you have to map the response PDU errstat to your own Error Message on the manager application. For Example in the generated xxxInstrument.java file import the new package as
import com.adventnet.snmp.snmp2.agent.*;
and in the set Methods should throw AgentSnmpException instead of
AgentException. The below is the code snippet for set Method
public void setxxxMethod(String value) throws AgentSnmpException{
if(value.equals("frequency"))
    throw new AgentSnmpException("",(byte)50);
}

13. How can I compile the agent in JBuilder 6.0?

Please refer our Help documentation for the same.

14.

1. Is AdventNet support Solaris as development plat form? - Yes
2. Is AdventNet supports GCC 2.95.2. on Solaris? - Yes
3. Is AdventNet support psos 2.5.0 and dcc rel 4.2 for target agent running? - Yes. Some of
   our customers are using it.
4. Is AdventNet supports Real time Linux as target platform? - Yes
5. Is AdventNet supports subagent development? - Yes. Both Master agent and subagent
   can be developed using AdventNet Agent Toolkit.
6. Can I add sub-agent to Master Agent at run time with out bring other agents down? - Yes.
   The Master Agent acts as an extensible agent and can include any number of subagents
during runtime. The Master Agent maintains a Proxy Table for this purpose and adding
Subagent entries to this table is very simple. It is similar to adding a row to any SNMP Table.
Additionally we provide support for configuring subagent entries in the Master Agent based on
OID, Context and Instance.
7. What is the protocol between Master Agent and sub agent? - SNMP
8. Is AdventNet support C++ development? - Yes, our source code can be compiled with G++
   compiler.

15. Can we know the method to create the *.class files without the toolkit GUI. As far as we
know the toolkit uses the javac-compiler. Can you give us some information about the
environment the toolkit uses??

You can make use of the setenv.bat/sh file that is available in the AdventNet\JavaAgent\bin directory.
This will set all the environment that you require for compiling the *.java files that has been
generated. Kindly do follow the following steps to convert the *.java files to *.class files with out using
the GUI,

1. Go to  AdventNet\JavaAgent\bin  in a Command Prompt.
2. Execute the setenv.bat/sh file by setting a valid Java path. This will set to the required
   environment under the AdventNet\JavaAgent directory.
3. Change Directory to "AdventNet\JavaAgent" using the command "cd .."
4. Use javac -d <output dir> <source dir>\*.java This will put the *.class files under the output dir
   that you have mentioned in the command.

AdventNet Inc.
16. What are Maximum Repetitions and Non-Repeaters? Where do I have to specify these values?

Maximum Repetitions and Non-Repeaters value need to be specified in the MIB Browser settings while sending a v2c GET Bulk request. By default it will be 0 and 50 respectively. If these values are not set properly errors might be thrown when a request is sent.

As per RFC 1905 : The values of the non-repeaters and max-repetitions fields in the request specify the processing requested. One variable binding in the Response-PDU is requested for the first N variable bindings in the request and M variable bindings are requested for each of the R remaining variable bindings in the request. Consequently, the total number of requested variable bindings communicated by the request is given by N + (M * R),

where N is the minimum of: a) the value of the non-repeaters field in the request, and b) the number of variable bindings in the request.

M is the value of the max-repetitions field in the request; and

R is the maximum of: a) number of variable bindings in the request - N, and b) zero.

17. Is it possible to generate code for Multiple MIBs? If so, how?

Yes, you can generate code for Multiple MIBs. All that you have to do is load both the MIBs, right click on it and Add to List. This gets loaded in the Selected Nodes Panel of MIB View tab. When you click on Generate Stubs icon or menu item, two options namely : Generate Code for Selected MIB or Selected Nodes are shown. You can choose Selected nodes which will generate code for Multiple MIBs as we have selected both the mibs already.

18. I've upgraded to the latest release of the java SDK. I uninstalled the previous version and how do i run the Agent Toolkit with the new version of java?

Set the JAVA_HOME for the new JDK version in setenv.bat/setenv.sh which is available at <AdventNet/Javaagent>/bin directory. This file is used to store information about the environmental variables, which are used in AdventNet Agent Toolkit product to execute the tools.

19. While trying to perform an SNMP Operations on the System Group Parameters of RFC-1213 of the SNMP Agent, it doesn't get updated. Why?

The System Group Counters are implemented as per the RFC1213 MIB specification. While generating the source using MIB Compiler, please go to the Settings (under Project Menu) window and de-select the "V2 Compliance" available in "General" panel and then generate the agent. Now when the agent generated by Agent-Toolkit-Java Edition is queried, it responds with the updated information.
MIB Editor FAQs

1. What is MIB Editor?

MIB Editor is one of the modules of Agent Toolkit which can be used to create, edit and instrument MIB files.

2. What does a node in the Mib Tree refer to?

A node in the Mib Tree refers to an Object to be managed by the agent. Each object whether it's a device or a characteristics of a device, must have a name by which it can be uniquely identified. That name is the object identifier. It is written as a sequence of integers, separated by periods. For example, the sequence 1.3.6.1.2.1.1.1.0 specifies the system description within the system group, of the management subtree.

3. Can I load more than one Mib file in the Mib tree? Are there any limitations?

Yes, you can load more than one Mib file in the Mib tree provided all the mib files have different module names.

4. Is there a big difference between TRAP-TYPE and NOTIFICATION TYPE ?

Yes. TRAP-TYPE is a Snmp v1 macro, whereas NOTIFICATION TYPE is a Snmp v2 macro. In Snmp v1 (TRAP-TYPE) you have enterprise OID and trap index. The TRAP-TYPE varies from 0 to 6. where 0-5 -> are generic traps

coldStart (0),
warmStart (1),
linkDown (2),
linkUp (3),
authenticationFailure (4),
egpNeighborLoss (5).

value 6 is for specific trap. If trap is specific then it can be described using the trap index.

In Snmp v2 (NOTIFICATION TYPE), the trap is described by the trap OID (.1.3.6.1.6.3.1.1.4.1 (rfc1907)). This table gives the difference between the SNMPv1 and SNMPv2 traps.
<table>
<thead>
<tr>
<th>Trap</th>
<th>Notification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contains Agent Address.</td>
<td>Does not contain Agent Address.</td>
</tr>
<tr>
<td>It has information about specific trap and generic trap value.</td>
<td>It has Trap OID in the second varbind</td>
</tr>
<tr>
<td>It does not have error index and status.</td>
<td>It has error index and status.</td>
</tr>
<tr>
<td>Does not support confirmed trap</td>
<td>Supports Confirmed Notification (INFORM).</td>
</tr>
</tbody>
</table>

5. Sometimes when I try to reload a MIB after making changes the changes are not seen. Why?

Please make sure you have saved the MIB module after modifying. (You have to select Save MIB Module) from the menu. The "Save Node" in the screen will save in the memory, once you select "Save Mib Module" from the menu, it will be saved into the file. So If you reload after saving MIB you can see the changes.

6. I installed the Agent Toolkit and JDK 1.2.2 on Win 98. When I invoke the MibEditor it crashes, the DOS window shows 2 errors. The 1st error is "Out of environment space" while setting CLASSPATH. The next error says "Exception in thread "main" java.lang.NoClassDefFoundError: com/adventnet.....

There are two things you can do to avoid this:

1. Change the memory settings of the Dos prompt. This can be done by choosing 'properties' icon in the toolBar. Go to 'Memory' tab, Set the Initial environment to a greater number (say 4096). This will avoid this 'Out of environment' space problem.

2. We have packaged a 'Launcher' application which will invoke all the other applications. This was specifically done to avoid these 'Out of environment' issues. (Agent Toolkit.bat).

7. Why am I not able to load RFC-1212, RFC-1315 and SNMP V2- SMI?

Actually these are ASN.1 Modules and not mib files. It contains macros that describes the various constructs of a MIB. Hence it just defines the format or the structure of a mib construct. If these are present in the imports column it will be ignored while parsing.

8. What does each icon in a MIB represent?

Some of the symbols in a MIB are described below:

<table>
<thead>
<tr>
<th>Image</th>
<th>MIB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image</td>
<td>Indicates the Origin of the MIB tree</td>
</tr>
<tr>
<td>Image</td>
<td>Indicates the Leaf node of the tree</td>
</tr>
<tr>
<td>Image</td>
<td>Indicates a leaf node which is read-only and not accessible</td>
</tr>
<tr>
<td>Image</td>
<td>Indicates a table node</td>
</tr>
<tr>
<td>Image</td>
<td>Indicates the index in a table</td>
</tr>
<tr>
<td>Image</td>
<td>Indicates a leaf node with read-create access</td>
</tr>
<tr>
<td>Image</td>
<td>Indicates the notifications in the MIB.</td>
</tr>
</tbody>
</table>
MIB Compiler FAQs

1. What is MIB Compiler?

MIB Compiler is one of the modules of Agent Toolkit which generates Java stub files for building agents after reading MIB files.

2. I wrote a very simple MIB with the MIB editor, then built and run the agent but I couldn't contact the agent with the Mib Browser....

Yes you can do that. You can start as many subagents in the same CPU at different port numbers.

3. What does the compile option "Implement SNMP group counters" of the Mib Compiler refer to?

Sub Agents can be registered to the Master Agent using dynamic registration or static registration. The Dynamic registration feature enables subagents to be registered through SNMP Manager i.e. you can configure subagents through SNMP Manager application and make any agent as subagent to your Agent. Multiple agents can also be registered as subagents to AdventNet Master agent. Dynamic Registration also supports OID, Context and Instance based registration.

4. When I compile, I keep getting the following error (both on NT and Solaris). Why does this error come?

Error: Class not found java.lang.NoClassDefFoundError: suntools/javac/Main
The classpath for the running
sun.tools.javac.main not found, please check the batch/shell file which executes this application Compilation Failed.

This error will occur when tools.jar is not available in the classpath. To avoid this error, ensure that tools.jar is available in the CLASSPATH settings.

5. Can Multiple Mibs be generated using MIB Compiler UI?

Yes, MIB Compiler provides this option in the UI. Load Multiple mibs and select the MIB ->Add to List, which gets added to the selected nodes. Then to generate the source files for the mibs added to selected nodes, select Build ->Generate Source ->Select Nodes.
MIB Browser FAQs

1. How do I give Hex strings for SET values in MibBrowser?

The SnmpString class accepts Hex strings in a certain format. Any string that starts and ends with a single quote(') is interpreted as an Hex string. The individual bytes should be separated using a colon(:). For example, if you need to enter 0x2a304cab, it should be supplied as '2a:30:4c:ab'.

2. How do I give Hex values in the ContextEngineID or in the ContextName text fields?

The Hex values should start with a 0x or 0X. Therefore, if you set a value for contextID or contextName, it should be 0xHHHHHH.

3. What are the units for timeout and retry values?

Both the timeout and retry values should be given in seconds. If you give the timeout value in milliseconds, it takes much time to get timed out. For example, a value of 1000 waits for 1000 seconds.

4. How do I set values for the table variables?

To create a new row in a table:

- Define a column with SYNTAX RowStatus, and the definition for the table should have RowStatus object defined.
- Select the Table node from the tree and the Table button from the toolbar to display the corresponding table.
- Right-click on the table header where the name of the column is displayed. It displays a menu with the following options.
  - View Graph for Selected Rows
  - Add a New Row to the Table
  - Delete the Selected Rows from the Table
  - Select Add a New Row to the Table. It displays a window for entering the values of the table.
  - The value for the column with RowStatus syntax should be 4 for creating a new row.
  - Click OK after entering all the values.
- If RowStatus is not present in the table definition, you can only modify the existing row by double-clicking the corresponding cell in the table.
5. If I load the MibBrowser applet, I get the error "Error Sending PDUSecurity Exception connecting to remote host" in the browser. Why is this so?

Applets are not allowed to talk to any host apart from the Web server from which they were downloaded. Make sure that SAS is also running along with the Web server.

6. I do not get the "NO HOST Specified" error. What should I do?

Before making any request, the host name or the IP address of the machine in which the agent is running should be specified in the "Host" text field of MibBrowser.

7. How can I load multiple MIB files in MibBrowser?

To load multiple MIBs, files should be separated by a blank space and be given within double quotes.

For example,

```
java MibBrowserApplication -m "mibs/RFC1213-MIB mibs/RFC1271-MIB mibs/RFC1155-MIB" -h localhost -c public.
```

If you use MibBrowser.sh, edit the file accordingly.

8. When I ask for 10 rows in an SNMP table, the GETBULK returns only 6 rows and the last attribute of the sixth row is null. The sixth row seems to be truncated. What should I do?

The number of rows you get back may be limited by the PDU size permitted by your agent, manager, or transport.

9. What is Atomicity (Roll Back)?

While processing a Multivarbind SET request, if the SET fails for any subsequent varbind, the previous successful SETs in the multivarbind are rolled back to the original values. This concept is known as Atomicity (Roll Back). This feature is handled within the "processSetRequest" of the XXXRequestHandler class and the Agent API.
Proxy FAQs

1. Can I create a subagent for the existing master agent?

Yes. The Agent Toolkit allows to create Windows subagent and Solaris subagent for the existing Master Agent.

2. Can we run multiple subagents on one CPU?

Yes you can do that. You can start as many subagents in the same CPU at different port numbers.

3. How are the subagents registered with the Master Agent?

Sub Agents can be registered to the Master Agent using dynamic registration or static registration. The Dynamic registration feature enables subagents to be registered through SNMP Manager ie. you can configure subagents through SNMP Manager application and make any agent as subagent to your Agent. Multiple agents can also be registered as subagents to AdventNet Master agent. Dynamic Registration also supports OID, Context and Instance based registration.
4. I have a legacy Master that supports SNMPv1, SNMPv2, SNMPv3 subagents, and I want to use AdventNet toolkit to build a Master Agent proxying SNMP request to the legacy Master Agent (now behaving as a subagent in this scenario). Does the AdventNet Master Agent have to support SNMPv3, SNMPv2 or SNMPv1?

Yes, both static and dynamic proxy concept can be used in your Master agent. Please refer the documentation in Building SNMP Agents -> Implementing SNMP Proxy section. These documents will help you know about the various kinds of Proxy implementation.

**Note:** The Static implementation and the Dynamic Registration should not use the same Subagent registration OID, for them to co-exist.

5. What are the capabilities provided by a Proxy Agent?

Proxy Agent provides the following advantages: -

- Context Based Proxy Support.
- Instance Based Proxy Support.
- Registering Subagents with the Proxy agent during Runtime. (Dynamic Registration)
- Removal of the Subagent Registrations during Runtime. You can also make it suspended for some time.
- Any standard SNMP Agent can be registered as a Subagent.
- Proxy Agent can be of SNMPv3 version and Subagents can be of v1 or v2c version.
- Communication is by standard SNMP PDU

6. How can we delete and add an entry to the Proxy Table?

Adding and deleting an entry from the Proxy Table is very simple. Rows can be added or deleted using any of the following options

1. Remote Configuration
2. Text files
3. API Calls
4. Before Agent start up.

Please refer Adding Subagent Entries in Implementing Proxy section of Building SNMP Agents for more information.

7. We are using HP-UX like a production platform and it is coming with some Snmp agents implementing the HP-UX MIB. All agents are controlled by a Master agent. So the question is can we use this same Master agent to control our agent developed with your tool.

The requirement is to integrate the AdventNet agent as a subagent to a Master agent, that has HP-UX agents also registered as subagents. It is possible to do so provided the Master agent is NOT a HP-UX agent. Just in case, you want to make the AdventNet agent as a subagent to the Master Agent developed using HP-UX. It CANNOT be done so, as HP-UX requires some shared libraries for its implementation which is not currently supported by AdventNet. A solution for this case may be, the AdventNet agent can be made as the Master agent and the HP-UX agent as its subagent.
8. What is different of Master Agent and Subagents job?

Master Agent :
Master Agent does not contain any information other than the subagent details. It just acts as a forwarder between the Manager and the Subagent. The Manager would forward the request to the Master Agent which in turn would forward to the Subagent registered in it. The Master Agent identifies the Subagents either using their OID or Community or Instance (three ways of registering subagents).

Subagent :
Subagents contain the MIB information and thus provides the required information to the Manager through the Master Agent.

9. Without using proxy features, I wonder if SNMP Agents support the Master Agent and Subagents architecture

The concept of Master-Subagent comes into picture only when the Proxy feature is implemented. Hence, without Proxy, there is no Master-Subagent.

10. Can Subagent work as Master Agent?

Yes. This is possible. In this case you can have two Master Agents and register one of the Master Agent as subagent to the other Master agent. The Master agent that has been registered as a subagent will forward the requests to the subagents registered under it.

11. Do you use your own designed proxytable (noStandard) without supporting RFC 2573?

AdventNet uses a proprietary ProxyTable for registering the subagent details in the Master Agent and our Agent supports RFC 2573 clause. In case you want to use your own proxy table, then the table has to be defined in the format similar to the AdventNet’s proxy table.

12. Is there possibility for Master agent reaction, if wrong values are configured for some tables?

If a wrong value is specified, say for example, in the subagent registration table you have specified an Integer value in place of a String value, then in that case the Master Agent would throw an error. OR If no such subagent exists and you have specified a wrong port or a wrong host in the Master Agent then in that case the Master Agent would look up for such a host or port and knowing that no such subagent exists, a general failure error will be thrown for a request.

13. Where can I use context Based Proxy, Instance Based Proxy and OID Based Proxy.

OID Based Proxy can be used when two different OIDs need to be accessed as Subagents. Context Based Proxy can be used when Agents running with two different communities need to be accessed as Subagents. And, Instance Based Proxy can be used when two different instances (rows) of a Table need to be accessed as Subagents.

14. Is AdventNet Agent an Extensible Agent?

Yes. AdventNet Agents are Extensible Agents. Using Proxy feature, you can register as many agents as subagents to the Master Agent (extensible agent) even during runtime.
15. Is it possible to use my own Table to register subagents dynamically using AdventNet Agent API?

Yes. It is possible for the user to choose his own table for adding the subagents dynamically. To achieve this, the user has to specify the entry-level OID of his own table using setRegisteredOid (<user table entry oid>); method, available in DynamicRegistration class for OID-based proxy, DynamicRegistrationWithInstance class for Instance-based proxy, DynamicRegistrationWithCommunity class for Community-based proxy, provided his table has the same number of columns as defined in the respective proxy tables.
**Trap and Notification FAQs**

1. Is trap generation supported in AdventNet SNMP Agent?

   The AdventNet SNMP agent supports to generate SNMPv1 trap and SNMPv2/v3 notification. You can generate SNMPv1 trap by defining a TRAP-TYPE construct in the MIB and SNMPv2/v3 Notification can be generated by defining a NOTIFICATION-TYPE construct in the MIB. Please refer to Sending Traps and Informs section in Building SNMP Agent for more details.

2. Can I define my own Trap Forwarding Table?

   Yes, you can define your own TrapForwardingTable instead of the default table bundled with Agent Toolkit.

3. How do I avoid the creating of a trap forwarding table in XML format? I would like to have the text file format, if possible.

   To choose the type of storage for Trap Forwarding Table, go to Project -> Settings menu in the MIB Compiler UI. The Trap Panel has the options for storage type. They can be either Text or XML or RAM.
4. What's the difference between cold and warm start trap?

The coldStart trap defines, "A coldStart trap signifies that the SNMPv2 entity, acting in an agent role, is reinitializing itself and that its configuration may have been altered. (from RFC 1907 description)

The warmStart trap defines, "A warmStart trap signifies that the SNMPv2 entity, acting in an agent role, is reinitializing itself such that its configuration is unaltered. (from RFC 1907 description)

5. Do you have a list of the alarms that are sent and how they are represented?

The traps are mainly generic traps, and AdventNet has implemented three of these traps, ColdStart Trap - generic type(0) WarmStart Trap - generic type(1) Authentication Failure Trap - generic type(4)

The rest are "linkUp, linkDown, and egpNeighbourLoss" traps. As the later mentioned traps are based on user's implementation of the Agent we have provided a method called "SnmpAgent.sendGenericTrap(int trapType, Vector varBindVector)".

Subagent linkUp and linkDown Traps are also supported.

6. What is the difference between SNMPv1 TRAP and SNMPv2 NOTIFICATION?

This table gives the difference between the SNMPv1 and SNMPv2 traps.

<table>
<thead>
<tr>
<th>TRAP</th>
<th>NOTIFICATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Contains agent address.</td>
<td>1. Does not contain agent address.</td>
</tr>
<tr>
<td>2. It has information about specific trap and generic trap value.</td>
<td>2. It has Trap OID in the second varbind.</td>
</tr>
<tr>
<td>3. It does not have error index and status</td>
<td>3. It has error index and status.</td>
</tr>
<tr>
<td>4. Does not support confirmed trap.</td>
<td>4. Supports Confirmed Notification (INFORM).</td>
</tr>
</tbody>
</table>

7. How can we add and delete an entry from the Trap Forwarding Table?

Entries can be added and deleted from the Trap Forwarding Tables using the Remote Configuration option, Text file option and through API calls.

8. Is it possible to convert the trap version or how to specify v1 or v2 in Traps?

The version of Traps can be configured at runtime. To do so, use the Trap forwarding Table. This table has a column defined as version. Specifying the version here, sends the traps in that version to the Managers registered in the Table.

9. Can we send traps through the Master Agent? If so, what is the procedure?

Yes. Traps can be sent through the Master Agent. For this purpose the Trap Listen and Forward feature or the Trap Filtering Table can be used.

10. How do we add and delete an entry from Trap Filtering Table?

Similar to other configuration tables entries can be added and deleted from Trap Filtering Table either using Remote Configuration option or Text Files or using API calls.
11. How do we test Notifications?

By default, the option "Generate Trap On Set " is enabled in the MIB Compiler UI. If this is selected and code is generated, traps get generated for the scalars and tables for which traps are defined on every set operation. Refer Sending Traps and Informs in Building SNMP Agents section for details. The Trap Viewer in MIB Browser helps to view the Traps generated.

12. I don’t want to create an agent, but just send a SNMP trap out of any JAVA program. So can I just call sendXXXNotification() without having to be an agent

Yes you can make use of this generated Method to send a Trap out of your Java Program, but you have to instantiate the SnmpTrapService class inside the Method.

13. Can I send a trap with oids that are not currently registered in the SnmpGroup component

Yes it is possible to send bogus Traps. You can send the Trap for an OID which is not currently registered in the SnmpGroup Component.

14. But is there a nice way to populate the objects to go with the trap i.e add an additional varbind to the Trap messages?

Yes. It is possible to add an additional varbind in the Trap message.

15. Is there any way to start the Agent in one port (161, standard) and open a different session in port 162 in order to receive the traps here?

Yes it is possible to start the Agent in one port and listen for Traps in an other port.

16. Many Agents support the possibility of sending different version traps to different destination hosts. Do you have anything that supports this in your software?

Yes, it is possible to send different version traps to different destination posts. This is possible as the Trap Forwarding Table, provides a column called version wherein the Trap version can be specified and sent to all the Managers specified in the Trap Forwarding Table.

17. Is there any option to check whether the code can be generated as Inform from the Mib Compiler?

As of now, using the options in MIB Compiler the code cannot be generated with Informs support. You need to implement Informs in the generated code.

18. Is there any advantage if the traps generated by the subagent are sent through master agent to the manager?

The most important advantage of sending traps generated by subagent to manager through master agent is mostly, the master and subagents run in a private network and the Manager in a public network. So the only communication is between the Manager and the Master Agent. Hence, the Subagents running in the private network may not be able to send traps directly to the Manager, neither can receive traps from it.
SNMPv3 FAQs

1. What is the primary goal of SNMPv3?

The primary goal of SNMP version 3 (SNMPv3) is to define a secure version of the SNMP protocol. SNMPv3 provides a secure environment for the management of systems and networks.

2. What are the security goals of SNMPv3?

SNMPv3 is designed to protect against:

- Modification of Information - Protection against some unauthorized SNMP entity altering in-transit SNMP Messages generated on behalf of an authorized principal
- Masquerade - Protection against attempting management operations not authorized for some principal by assuming the identity of another principal that has the appropriate authorizations
- Message Stream Modification - Protection against messages getting maliciously re-ordered, delayed or replayed in order to effect unauthorized management operations
- Disclosure - Protection against eavesdropping on the exchanges between SNMP engines

3. What are the security levels of SNMPv3?

SNMPv3 provides the following three levels of security:

- No Authentication and No privacy (noAuthNoPriv)
- Authentication but no Privacy (authNoPriv)
- Authentication and Privacy (authPriv)

A request can be made on behalf of a user with any one of the above security levels.
4. What are the third party packages available that can be used to download and test the v3agent with privacy support?

You have to install JCE or Cryptix packages for encryption of v3 messages. Please refer to Configuring SNMP v3 Agent to know about the third party packages that are used to provide privacy support.

5. Are those default template users (i.e. authUserMD5 etc) required in the USM user table in advance in order to add a new user to the USM user table?

Yes, the template users are required in order to add a new user to the USM user table from the remote (through snmp manager). But the template users need not be the same as defined by the Agent Toolkit. What is there are just examples. You can define your own template user (name, password etc) with view to the USM and VACM tables for the different security levels from the agent at startup or runtime through function calls. Please refer SNMPv3 in Building SNMP Agents section to add a new user through function call.

6. Is it possible to add a new user in the USM Table through remote configuration?

Yes, it is possible to add a new user in the USM Table from the remote. For this, the template user for that security level must already exist in the USM and VACM tables. Please refer SNMPv3 in Building SNMP Agents section for more information.

7. I have developed an agent which is V3 enabled. Now, I want to add a new user and provide them views for the MIBs that I have in the agent. What do I do?

To add a new user to the v3 agent, you have to configure the new user information in the USM and VACM tables. Please refer Configuring a SNMPv3 Agent in Building SNMP Agents section of help documentation.

8. I have developed a SNMPV3 agent without privacy support. But through SNMPv1 or V2 I am able to access whole MIB without any authentication. How can I prevent it from accessing the data (OR) How can I turn off the v1/v2 access in the v3 agent? (OR) How can I forbid the v2c browsing MIB? I tried using v2c, it still could browse the MIB.

Please follow the steps given below to make a V3 Agent strictly V3: -

1. After generating the agent, override the callback method of SNMP Agent by adding the following code in the Main File,

```java
/* User code starts here */
/**
 * Overriding the callback of the SnmpAgent.
 * This method will be called by the SnmpSession class. We check
 * for the Pdu's version and if it is not V3, drop the PDU.
 * else call the SnmpAgent's callback.
 * @param sess - the SnmpSession which handles the request.
 * @param pdu - the incoming request PDU
 * @param reqid - the unique identifier for the request PDU maintained by
 * the SnmpSession
 * @return boolean indicating if the request is processed or not.
 */
public boolean callback(SnmpSession sess, SnmpPdu pdu, int reqid ){
    if(pdu.getVersion < SnmpAPI.SNMP_VERSION_3) {
        System.out.println("Lower Version received ...dropping the PDU");
    
```
Adding this method will drop the requests from v1 and v2c Managers. Save the file and compile it. Start the MIB Browser application and test the V3 agent by sending a v2c request. The request will be dropped.

9. I got a user added in the USM user table. But, when I test the new user in the v3 manager, I do not get the required information from the v3 agent. What should I do?

This is because the new user's read/write/notify view and other information are not provided. These information has to be added to the VACM Tables (VacmContextTable, VacmAccessTable, VacmSecurityToGroupTable and VacmViewTreeFamilyTable.) For more information about SNMPv3, USM and VACM tables please refer the relevant topics in our product help document.

10. When I want use SNMPv3 and privacy. I install third part software, I wonder if these product is available in Europe?

For privacy support, the Encryption packages that can be used are "Cryptix" and "JCE". i.e.: -

- Cryptix 3.1.1
- Cryptix 3.1.3
- Cryptix 3.2
- JCE 1.2
- JCE 1.2.1

Encryption packages are bound by export restrictions.
--- If JCE 1.2 or its implementations are used in developing application and applets it cannot be used outside of US and Canada.
--- JCE 1.2.1 does not have any export restrictions and it can be used in applications which can be distributed throughout the world.
--- The latest JDK version (JDK 1.4 beta) comes integrated with the JCE 1.2.1.
--- Cryptix package does not have any such export restrictions.
Authentication and Authorization FAQs

1. What are the various types in adding and deleting an entry from an Authentication Table?

An entry can be added and deleted from an Authentication Table in either of the following ways:

- Remote Configuration
- Text File
- API calls and
- Runtime Memory

The Authentication section in Building SNMP Agents will explain more on this.

2. Is there any manner of avoiding the generation of this code. The reason is that we want to avoid creating an extra file, as well as the extra library that is needed for XML table handling.

```java
// For Community Based Access Control Implementation
acl = new com.adventnet.snmp.snmp2.agent.AclTableRequestHandler((SnmpAgent)this, aclTable);
```

The code you specified for the acl implementation is generated by default. To avoid the creation of an extra file, as well as the extra library that is needed for XML table handling in acl implementation you can use the Runtime Memory storage option. It will store the data in the memory. When we restart the Agent the memory will be lost and the newly added entries will not be there.

3. I start one of my agents like this: java.exe ZCEBTSPDGSIM.StartSimulator -p 1161 -c Z -d 3 and another agent like this: java.exe" ZCEBTSPDGSIM.StartSimulator -p 2161 -c N -d 3. But for some reason both of my agents are using the community string "Z".

The first community is taken from command line before instantiating the acltable. When the acltable is instantiated the community will be set to the community in the acl.txt file. Here you have the community as Z in the acl.txt file also. That is the reason you have got the same output in both printout statements. To process the command line option community then acltable should not instantiated.

When you add a new entry, it will write the entries in the xml file. Hence you can use the RAM storage option to avoid the creation of an extra file, as well as the extra library that is needed for XML table handling in acl implementation.

When you add the new entry for the acl table, it will store the data in the memory. When you restart the Agent the memory will be lost and the newly added entries will not be there.
4. Actually I don't understand very well the way the aclTable is used. Do I have to write some code and to include the table definitions in my own mib?

To Configure the Manager Entries in aclTable the following methods can be utilized: -

- Using the wizard interface of MIB Compiler UI. (Project -> Settings menu)
- Using API calls - Instrumenting the generated code for adding Manager entries.
- Using AccessControlTable.txt file.
- Remote Configuration.
- Or from Command Line.

5. I need to send traps to several managers. And possibly also restrict access to only those specific managers. How do I restrict access to the agent? Is this easy? (I want to restrict on ipaddresses)

The TFTable, available in our Agent Toolkit is for sending Trap messages to Multiple Managers who are interested in receiving the Traps and registered with the Agent to receive the Trap messages generated by the Agent.

If you want to send a Trap message to say your Machine "localhost" at port "162" and to one of your neighbor's Machine named "xxxxx" to the port number "9001", then you have to add these entries while generating the Agent, which will send all the generated Trap messages to the registered Managers at their respective ports.

You can also add additional Managers to the TFTable by simply doing a Set request to the forwardingTable present in our agent-snmp-config-mib.txt. The same can be applicable for removing or restricting a Manager from receiving traps by deleting that particular row.

Regarding the restricted access to the Managers registered in the Agent, at present it is based on the Community that is specified in the aclCommunity column present in AclTable (In agent-snmp-config-mib.txt). By using this you can restrict the access to particular Manager by changing the aclAccess level in the AclTable. To know more about Access Control Table kindly refer to Authentication section in our Help docs.

Using this Table, you can specify the Access Levels for the Various Managers by simply doing a Set Request as similar to that of the TFTable.
Table Handling FAQs

1. How can multiple parameter sets be done as an atomic transaction, so that clean rollback can be done if there is a failure? Can this work using SNMP?

We do provide support for Atomicity. While processing a Multi Varbind SET request, if a SET request fails for a particular varbind after passing through a few varbinds, then in that case, the original values of the nodes will be set again.

2. What is Persistence Storage?

The Persistence storage of the Table entries stored in a Text File format is to avoid any loss of Data due to Agent Crash or any other accidental stopping of the Agent. The persistence storage table should be having all the entries that is available in the Agent during termination.
OS Agent FAQs

1. Does both the native agent and a java agent run on the same box?
   Yes. Both the Native agent & the Java agent run on the same box.

2. Can AdventNet Agent be made as subagent to Windows, Linux and Solaris master agent?
   Yes. You can make Solaris, Linux, and Windows agents as AdventNet Agent Toolkit's subagents. AdventNet agents can also be made as subagents to Windows, Linux, and Solaris Agents.
Rule Engine FAQ

1. What is Rule engine and why it is used?

Rule Engine can execute a set of rules which consists of expressions about managed objects that are used to manage the attributes of the users' application/device and then execute some actions like traps/notifications, sending mails, and perform set operation when a given expression/logic is executed.

The need for Rule Engine is that it makes the users' logic declarative through the XML file instead of hard-coding or instrumenting the decision process in the stub files to perform any actions.

2. How to map the agent details with the ruleset?

The rule engine provides two configuration files namely AgentDetails.xml and RuleSet1.rule. The AGENTID serves as an unique identification for the SNMP agent in AgentDetails.xml file and associates the SNMP specific details of the agent with the RuleSet in rule file. Hence the agent details are mapped with the ruleset by mapping AGENTID with RULESETID.

3. How do i execute the rule engine for SNMPv1/v2c or v3 agent?

The AgentDetails.xml file contains SNMP agent related details such as the tables used for agent configurations. Thus, it contains a table to store the details of SNMP agent such as version, host name, port and community called SNMP_AGENT_DESCRIPTION_TABLE.

If the version mentioned in the table is 1 or 2, then it implies that agent is SNMPv1 or SNMPv2c agent respectively and the agent processing is performed with the community mentioned that is required for authenticating the requests. If the version is 3, then it implies that the agent is SNMPv3 and the SNMP_V3_AGENT_DESCRIPTION_TABLE is processed.

4. How to define a rule if there are more than one expression to perform some action?

The rule engine supports both Composed Expression and Expression Operator Expression (EOE). If more than one expression needs to be evaluated, then it can be performed using Composed Expression where
Composed Expression are any compound expression that evaluates complex or more than one expressions. The EOE can be made as the part of composed expressions with operators such as AND, OR, etc.

5. How can I make the agent to poll rule engine and generate traps/mails at some time interval?

The rule engine polls the rule file and generate traps/mails in the Scheduler execution mode. While invoking scheduler, setRefreshTime() method is used to provide the refresh time for which the polling is performed by the rule engine. The scheduler starts sending traps/mails once the expression is true and performs the action repeatedly by refreshing the rule file for every refresh time interval.

6. How to stop the traps/mails generated for some time interval?

The traps/mails are generated continuously for some regular time interval i.e refresh time, when the rule engine is running in Scheduler mode. This does not gets generated either when the expression becomes false or the rule engine execution mode is changed to Trigger mode.

7. Is there an option to trigger only particular rules than all the rules mentioned in the rule file?

Yes. Rule engine provides the functionality of evaluating only particular rules, though there are many rules specified in the rule engine. This enhances the performance of the rule engine. For further details on its implementation, refer Triggered Based Rule Execution Mode section of Enabling Rule Engine.

8. Can there be more than one rule file?

Yes. There can be more than one rule file containing one ruleset each.

9. Does the rule engine permit to configure rules dynamically? If so, is it required to restart the agent for updating the changes?

Yes. The existing rule file can be edited or new rule files (ruleset) can be added at runtime to the rule engine by enabling HotDeployment feature as the setHotDeploymentRefreshTime() method updates the rule engine at equal refresh time intervals mentioned in the method. It is also not required to restart the agent using the feature, as it gets updated automatically for every refresh time.

10. What are the different actions performed using rule engine?

The different actions performed using the rule engine are

- sending traps/notifications
- sending mails
- perform set operation when a given expression/logic is evaluated

11. Does the rule engine allow customizing user's implementation classes instead of the default classes available in the ruleenginefactory.xml file?

Yes. For further details, refer Customizing Rule Engine section of Rule Engine.
12. Can I operate the agent without Rule engine?

Yes. Rule Engine can be disabled by choosing Settings->Source Generation->Enable Rule Engine Service from the MIB Compiler UI and then generate the agent again.
29.0 Troubleshooting

The Troubleshooting tips for various module are listed below:

- SNMPv1 and v2c Agents
- SNMPv3 Agents
- Rule Engine
SNMPv1/v2c Agents

The Troubleshooting tips for SNMPv1/v2c agents are as follows:

Request Timed Out to Agent Host

**Reason** : This message appears in the Mib Browser on the following conditions:

- If the MIB Browser is not configured with the same Community String as the Agent then it will send an authentication trap and will not send any response to the manager.
- When the Agent is not running or if the port number and /or host at which the Agent listens, is different from the port number specified in the Mib Browser settings.
- If the request sent from the manager is of higher version than it is configured in the agent, then the agent will not send any response to the manager.
- When the agent is not able to send the response to the manager within the specified time out value.

**Solution** : The problem can be resolved by giving the same Community string, host, port number in the Mib Browser Setting as it is configured in the agent. If the problem occurs due to time out value then increase the timeout value.

General Error

The reasons for the error to be thrown in the Master Subagent Architecture is explained below:

**Reason** :

- The "generr" will be thrown by the master agent when the response from the subagent is not received within the specified timeout period.
- If the host and the port number entry of the subagent in the proxy table is not the same as it is configured in the subagent.
- When the subagent receives a request of higher version from the master agent, then it does not respond to the master agent.
- When the Community String in the subagent does not match with the community entry of the subagent in the proxy table.
- If the proxyRowStatus column of the particular subagent entry in the Master agent is not active.

**Solution** : Please ensure whether the Subagent is running in the specified port, community, and host. To resolve the error due to timeout value, use proxyTimeout column in the proxy table of the particular subagent.

No data available in this MIB

**Reason** : This error occurs when a node is not registered (implemented) in the agent or when there is no entry available in the table and a request is sent.

**Solution** : Find out the reason for your problem from the above listed reasons and correct it.
End of MIB View

Reason: This message occurs when a GET-BULK is made. It also occurs when GET-NEXT request for v2c request is made to the last variable of the agent.

Solution: If you reduce the max repetitions value, then you will get the proper response from the agent.

No ObjectID specified

Reason: This message occurs when a request is made without selecting an object in the MibTree i.e when the OID is not specified before making a request

Solution: Select a node and send a request.

Invalid OID format

Reason: This message appears when an invalid OID is mentioned for any variable.

Solution: Check the OID of the variables.

No Such Variable Name

Reason: This error message is thrown when doing a GET operation and
  1. The OID is not instrumented by the Agent.
  2. The Read Community specified is not the right one.
  3. If the variable chosen for GET request is write only variable.
  4. When querying a columnar variable with an invalid instance or scalar variable with instance other than 0.

Solution: The error can be avoided by choosing a valid variable as implemented in the agent.

No Host Specified

Reason: If the Host Name is not specified before making any query this error occurs.

Solution: Enter the correct Host name before sending a request to the Agent.

sysDescr in RFC 1213 MIB value being "mySysDescr" and not "mib2AgentSimulator"

Reason: The sysDescr, in SystemInstrument.java file, initializes as protected String sysDecr = "mib2 Agent Simulator"; But it actually gets "mySysDescr" not "mib2 Agent Simulator" when queried. This is because, we have implemented SnmpV2-MIB in the agent which contains systemGroup, sysOrTable and snmp Group. So the generated System group of RFC1213MIB is not registered in the Agent because the System Group of SnmpV2-MIB which we have implemented is registered (the oid for both are same (.1.3.6.1.2.1.1)). So when you do a query to the agent for sysDescr then the agent returns "mySysDescr" default value of the Snmp Group of SnmpV2-MIB which we have implemented.
Solution: If you want to get the value presented in the SystemGroupInstrument.java file, comment the code snippet mentioned below in the Main File i.e RFC1213.java

```java
/* sysORTable = new
   com.adventnet.snmp.snmp2.agent.SysORTableRequestHandler((SnmpAgent)this);
sysORTable.addRegistrationListener(hdlr);
SystemGroupInstrument instru = new SystemGroupInstrument();
super.addSystemGroupListener(instru);
*/
```

If the above mentioned code is commented then Agent Toolkit's System Group and SysOrTable are not registered in the agent. Now compile and start the agent then do a query to the agent then the value will be "mib2 Agent Simulator" which is initialized in the SystemInstrument.java file can be retrieved.

---

Wrong Value Error

Reason: The error "wrong value" will occur only when the value is not being able to assigned to the variable. Say for example, if the managed object has a range defined from 0 to 10 and you are trying to assign the value 11. In this case the Agent will return this error.

---

Trap not being received by the Manager

Reason: When a Trap is not received for a specific user you have to check if all the entries specified in the v3 tables are correct. If the trap is not received in the Manager side then check whether the community, port, and host on the Manager side is the same as it is mentioned in the TrapForwarding table.

---

Bind Exception

Reason: This error is thrown, if another application is running in the port where the agent is already running.

Solution: Stop the application in that specified port and start it in a port after verifying whether the port is in use or not. This can be assured by using `netstat -an` command to know the ports that are in use. To start the agent in some other port, the option `run -p [port number]` can also be used where [port number] implies the port which is not used.

---

Compilation error while compiling the generated code:

Reason: When the code is generated for the list of selected nodes/MIB without adding its imported MIB to the list, then this error is thrown.

Solution: While loading a MIB, the other MIB imported in it also gets loaded. So while adding any MIB to the list, the imported MIB in it should also be added which will avoid this error while code generation.
SNMPv3 Agents

The Troubleshooting tips for SNMPv3 Agents are as follows:

Time Sync Error in V3

Reason: A Time sync error occurs, when the agent is not configured for a particular user and we try doing a GET/SET request for that user. Say for example, after starting the agent, we give the following values in the MIB Browser v3 settings.

- Target Host - localhost (by default)
- Target Port - 8001
- User Name - senior
- Security Level - Auth,noPriv
- Auth Password - senior

and "Add the Entry". You will get the time sync error as no such user senior exist by default.

Solution: Enter the correct UserName and password as provided in the USM user table.

General Failure in V3

Reason: A general failure occurs when you add the user entries properly but do not give a context name or give a wrong context name in the MIB Compiler test tab. Also when view access in the MIB is restricted up to a particular OID, in the VACM Family View Table and you want to access it further.

Solution: Specify the exact context name.

Discovery Failure in V3

Reason: This error occurs when you try to add an user without the agent running. Of course, it is possible to add an agent with "noAuthnoPriv" security level as the agent will be treated as a v2c agent.

Solution: Start the Agent in the specified port before sending a query to the Agent.

Authorization Error

Reason: This error occurs when the security level and the context name contradicts each other. Say, you give the security level as "noAuthnoPriv" and the User Name as "auth". After adding the entry in the MIB Browser V3 settings, specify the context name in MIB Compiler test tab as "auth" and try doing a GET/SET request. An "authorization failure" error occurs.

Solution: Find out the cause of your problem from the above listed reasons and do the necessary changes.
Exceptions thrown - (class Cipher not found)

**Reason**: "Class not found" exceptions are thrown when we send a request for the private users (privUser). Though the entry gets added in the MIB Browser v3 settings, the exceptions are thrown in the command prompt. This happens when, the privacy packages required for v3 agent's private users are not present in the classpath. Please go through the privacy package settings in V3 for more information.

**Solution**: Ensure that the Privacy Packages settings are configured properly in the classpath.

---

End of MIB view error

**Reason**: You will get this error while trying to do a get-next for the v3 agent. For Example: If the context name provided in the Mib Browser if not the same as that configured in the vacmContextTable for that user. For example, the context name provided for the user authUserSHA in the vacmcontexttable is "auth". If this user tries to access the v3 agent by providing the context name as "noauth" in the Mib Browser.

**Solution**: Find out the cause of your problem from the above listed reason and do the necessary changes.

---

usmStatsNotInTimeWindows(.1.3.6.1.6.3.15.1.1.2.0)

**Reason**: The meaning of the error, "Not In Time Window" is when the sent request EngineTime and EngineBootCnt are different with the agent's EngineTime and EngineBootCnt. The possible reason is, the agent would have restarted.

**Solution**: The subsequent queries will be success, when the "Not In Time Window" error response has both the latest EngineTime and EngineBootCnt of agent.

---

User not successfully cloned error

**Reason**: You will get this error when you try to create a new user in the agent using an existing user from the remote. The possible reasons are,

- Remote agent may not be running.
- Clone-from user name or password or security level may be wrong.
- Clone-from user's context name is not having access in USM table.
- Clone-from user and new user's authProtocol (MD5 & SHA) may be different

**Solution**: Find out the reason of your problem from the above listed reasons. Delete the row from the USM table of new user if created. Run snmpUSMRemoteConfigure.java command line tool again with correct inputs.

---

Getting "USM keychange is not successful" error

**Reason**: You will get this error when you try to change the auth password or priv password of another user or yours. The possible reasons if the key changes for another user are,

- Remote agent may not be running.
- Template user name or password or security level may be wrong
- Template user's context name is not having access in USM table.
- Template user's and the other user whose key is to be changed authProtocol (MD5 & SHA) may be different

The possible reasons if key change for same user are,

- Remote agent may not be running.
- User name or password or security level may be wrong
- User's context name is not having access in USM table.

**Solution**: Find out the reason for your problem from the above listed reasons and provide the correct inputs.

---

**Unable to Decode PDU**

**Reason**: This error will occur when the cryptix classes or jdk1.2 classes are not in the classpath for the privacy support. Also when the `java.security` file is not edited properly to include the exact privacy packages.

**Solution**: The classes should be set correctly in the classpath of setenv.bat file.

---

**Unable to Encode PDU**

**Reason**: This error will occur when the Context Name and ContextEngineID are not specified. This exception is thrown for get, getnext, getbulk & set operations.

**Solution**: Check if the context engine id and context name are specified correctly. If not, modify them accordingly.
Rule Engine

The Troubleshooting tips for Rule Engine are as follows:

---

**Traps Not Received**

**Reason:** The following may be the cause:

- ACTIONID is not correctly mapped with the TFTTABLEID under SNMP_TRAP_ACTION_TABLE.
- ACTIONID is not mapped to the TYPEID that refers to the SNMP_TRAP_ACTION_TABLE, under ACTION_TABLE
- ACTIONID is not correctly mapped to the ACTIONSETID under ACTIONSET_TABLE
- TRAP_FORWARDING_TABLE must have correct details of the manager such as port, host, hosttype of the manager.

**Solution:** Provide the exact mapping between tables for sending traps.

**Reason:** Other reasons can be related to the expressions evaluated during the rule engine.

- Expression returns “false”
- The variable or constant is not correctly mapped to the DATAID which in turn is not mapped to the correct table for mentioning the expression details using TYPEID
- The details about the variable such as GROUPNAME, SUBID, and INSTANCE are specified wrong or misspelt.
- The operator used in the expression is mentioned wrong.
- The value for the constants hold a wrong data type mentioned by a table using TYPEID

**Solution:** Provide the related details once identifying the problem.

---

**Email Not Received**

**Reason:** The following may be the cause:

- ACTIONID is not mapped to the TYPEID that refers to the EMAIL_ACTION_TABLE, under ACTION_TABLE.
- ACTIONID is not correctly mapped to the EMAILID under EMAIL_ACTION_RELATION_TABLE that sends the email to the specified mail addresses.
- ACTIONID mapping to the SERVERID may be wrong under EMAIL_ACTION_TABLE.
- EMAILID mentioned under EMAIL_ADDRESS_TABLE may not be a valid mail ID.

**Solution:** Provide the exact mapping between tables and correct email address for sending email.

**Reason:** Other reasons can be related to the expressions evaluated during the rule engine as mentioned above in the last reason of “Traps Not Received” section.

---

**Cannot Perform SET Operations**

**Reason:**

1. ACTIONID is not mapped to the TYPEID that refers to the SNMP_SET_ACTION_TABLE, under ACTION_TABLE
2. ACTIONID is wrongly mentioned in SNMP_SET_ACTION_TABLE.
**Solution:** Provide the exact mapping between tables to perform SET operation.

**Reason:** The details of the managed object specified in the SNMP_SET_ACTION_TABLE to which the action is to be performed is invalid.

**Solution:** Specify the valid details such as the OID and its instance, value to be set and its data type.

**Reason:** The data type of the value to be set for the managed object, mentioned using the TYPEID is wrong

**Solution:** Check the data type that must be given as mentioned in the SNMP API Javadocs.

**Reason:** Other reasons can be related to the expressions evaluated during the rule engine as mentioned above in in the last reason of ”Traps Not Received” section.

---

**Cannot Stop the Traps/Mails that Gets Generated Continuously**

**Reason:** This may happen when the rule engine is started in scheduler mode. The scheduler starts sending traps/mails once the expression is true and performs the action repeatedly by refreshing the rule file for every refresh time.

**Solution:** The rule engine can be operated in the triggered mode by disabling the scheduler mode. The scheduler refresh time can also be increased to avoid frequent generation of the traps/email. For more information on scheduler mode of execution, refer the Rule Engine section.

**Note:** Even in Scheduler mode, the traps/mails generated periodically depending on the refresh time, stops its generation once the condition becomes false. This again starts performing action when the condition becomes true.

---

**Email Sent with Macros Does Not Work**

**Reason:** The same macro name mentioned for another MACROID.

**Solution:** The Macro name must be unique. Hence check whether the MACROID has a unique macro name.

**Reason:** The DATAID mentioned in the MACRO_TABLE may not contain a valid data in its table referred using TYPEID.

**Solution:** Provide the exact data and correct mapping of DATAID to the MACROID.

**Reason:** The Macro name is not given between the "$" sign.

**Solution:** While specifying the macros, ensure that the macro name is mentioned between $ symbol, otherwise it will not be considered as macros, for eg $test$, where test is the macro name. Also ensure that the macro name mentioned in the EMAIL_ACTION_TABLE is the same as that of the macro name specified in the MACRO_TABLE to send message with macros.

---

**Traps are Sent to Multiple Managers Even When Only One TrapForwardingTable is Set in the Rule File**

**Reason:** This might be due to specifying the TFTABLEID to be 0 which will send the traps to all the entries specified in the TrapForwardingTable of the agent.

**Solution:** If the trap must be send to only the TrapForwardingTable mentioned in the TRAP_FORWARDING_TABLE of the rule file, then define a TrapForwarding entry in the table and give the TFTABLEID corresponding to that entry.

---

**Ruleset Not Executed Even After Mapping to the AGENTID**

**Reason:** Though the Ruleset is correctly mapped to the AGENTID, it may not get executed if STATUS column in the RULESET_TABLE is 0.

**Solution:** Make the STATUS to be 1 to enable the ruleset.

---

**Rule Not Executed Even When Mapped to the RULESETID**

**Reason:** Though the Rule is correctly mapped to the RULESETID, it may not get executed if the STATUS column in the RULEID is 0.

**Solution:** Make the STATUS to be 1 to execute the rules under the ruleset
Cannot Perform More Than One Action
Reason: ACTIONID mapping to the ACTIONSETID for the respective tables performing different actions is not done.
Solution: Different actions have their unique ACTIONID. In case more than one action needs to be performed, map them correctly to the ACTIONSETID.

Action is Not Performed for a Tabular Variable
Reason: This might be due to mentioning an instance value of wrong type, value, length etc.
Solution: Rule Engine supports tabular group with both index and external index. While mentioning INSTANCE for such tabular variable, the instance must be specified along with the length of the instance, index and external index. For eg, <INSTANCE>.6.87.23.56.35.24.67</INSTANCE>

Cannot Send Mail to More Than One Email Address
Reason: EMAILID is not properly mapped to the ACTIONID.
Solution: The EMAILID defines different email address in EMAIL_ADDRESS_TABLE. Hence mail can be sent to more than one email address by mapping the ACTIONID used for sending mails to the EMAILID under EMAIL_ACTION_RELATION_TABLE.

Cannot Set Values with Data Types Such as Timeticks, IPaddress etc for SET Action
Reason: The rule engine will not perform set operation if the values of data type are mentioned wrong.
Solution: To mention the data types are required for setting values in SET operation, specify them as defined in the SNMP API Javadocs. By default, the data type mentioned in the rule file are String and Integer.
30.0 Reference Implementation - Standalone SNMP Agent

- Introduction to SNMP Agents
- Objective of the Tutorial
- Getting Started with the Tutorial
- Application Notes
- Running and Testing the Application
- Steps Involved in Developing the RI
  - Created a MIB
  - Generated Code for the MIB
  - Implemented Scalars
  - Implemented Tables
  - Implemented SNMP Traps
Introduction to SNMP Agents

AdventNet Agent Toolkit is a rapid prototyping and development tool used for building cross platform Java based SNMP Agents. Though, it is possible to develop SNMP, TL1 and Multi-Protocol Agents using the toolkit this tutorial has been organized to demonstrate the complete development process of implementing a SNMP management solution in Java. The architecture given below gives an overview of the SNMP Agent.

- The Mib Editor creates a MIB.
- Taking MIB as the input the MIB Compiler tool generates and compiles java code for that MIB.
- The created Agent, is later tested using the MIB Browser test tool.
- Please go through this tutorial to know how the Agent works.
Objective of the Tutorial

The intention of this tutorial is to make you familiar with the features supported by AdventNet Agent Toolkit for SNMP Agents. The power of the Tools present in the toolkit and the rich Agent features are illustrated in this tutorial by developing a Reference Implementation Agent for a Simple application. With this implementation, you can view the statistics of the application and also configure them from a remote place using a SNMP Manager i.e. the Agent exposes the application data to the Manager through this implementation. After reading this tutorial you will be able to

- Create a MIB using the MIB Editor
- Generate and compile code for the mib created, using MIB Compiler.
- Instrument Scalar variables created in the MIB.
- Instrument Tables using the Agent Table Model approach (vector approach)
- Implement Traps and Notification.
Getting Started with the Tutorial

- Agent Toolkit Product Installation
- Help documentation for the RI
- Application Specification

This tutorial leads you through the development of dynamic SNMP Agents using the framework and components provided by AdventNet Agent Toolkit. This tutorial is aimed at OEM who wish to instrument their objects to make them SNMP manageable and developers of SNMP Agents. The prerequisite of this tutorial would be reasonable knowledge in Java programming.

Agent Toolkit Java Edition Product Installation

For running the reference implementation and described tools in this tutorial, AdventNet Agent Toolkit Java Edition must be installed in your machine. Before reading this tutorial, you should be familiar with the tools (like MIB Editor, MIB Compiler etc..). For information on these tools, refer Agent Toolkit Java Edition Product Documentation. The documentation is also available online (http://www.adventnet.com)

HELP Documentation for the RI

To start with, please refer to the "Application Notes" section. And directly move on to the steps involved in running the application and testing the same. The steps have been covered in "Running and Testing the application".

To know how this Reference Implementation was developed, go through the "Steps Involved in Developing the RI" section. The "Created Mib" and "Generated Code" section provides explanation for creation of a simple Agent, starting from the definition of a MIB till compilation of the code. Other sections like Implemented Scalars, Tables and Traps talks about the features supported by the toolkit. On following these steps you can implement the features required for your application and the Agent will respond with the implementation values on request (not the default values).

To have detailed explanation on the terms explained here please refer the Building SNMP Agents section of the AdventNet Agent Toolkit Java Edition Product Help Documentation.

Application Specification

<table>
<thead>
<tr>
<th>Application Version</th>
<th>Shopping Cart Application, 1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent Toolkit Version</td>
<td>Agent Toolkit Java Edition 6</td>
</tr>
<tr>
<td>Tools Used</td>
<td>MIB Editor, MIB Compiler, MIB Browser</td>
</tr>
<tr>
<td>Platform Specific Requirements</td>
<td>No special requirements</td>
</tr>
</tbody>
</table>
Application Notes

- About Shopping Cart
- Identifying Resources for developing the Application

This section of the tutorial, talks about the Shopping Cart application which we are trying to make it SNMP manageable. The Shopping Cart application bundled along with the toolkit is present in `<Agent Toolkit Home>/reference/snmp` directory.

About Shopping Cart

*Shopping Cart* is basically an application used for placing orders for items in a shop. The items as per this reference implementation are "Pets". Four pets namely: Parrot, Dog, Cat and fish are available in the shop and to place orders for these pets the Shopping Cart application can be used.

Treat yourself as a customer and assume that the Shop Keeper has allowed you to place orders through this application. The Shopping Cart application gets started via the Agent. Hence the Agent need to be installed in your system.

To know more about starting the Agent/application and testing the same go through "Running and Testing the Application" section.

Identifying Resources for developing the Application

The first step toward Building an Agent for the application is to define the resources required for the Agent. Based on the requirements the resources are identified as follows:

- Pet Name
- Unit Cost of the Pet
- Quantity of the Pets
- Total Amount of the Pet based on the Quantity
- Total Stock available for a Particular Pet
- Total Pets ordered
- Total Price for the Pets ordered

Having identified these details, grouping has to be done. They can be classified as follows:

In the form of a Table

The following details can be presented in the form of a Table as they have details related to a particular variable: **The Pet**. Hence the Pet Name can be made as the index column and other columns representing the details of the Pet. Say the Table can be as follows:

<table>
<thead>
<tr>
<th>Pet Name (index)</th>
<th>Unit cost of the Pet</th>
<th>Quantity of the Pets</th>
<th>Total Amount of the Pet based on the Quantity (List Price)</th>
<th>Total Stock available for a Particular Pet</th>
</tr>
</thead>
</table>

In the form of Scalar Variables

The Total Price of the Pets ordered and the Total Pets ordered can be represented as two scalar variables as these details hold only a single value.

Using these details the MIB Module is defined.
Running and Testing the Application

- Running the Agent
- Testing the Agent with default values
- Placing Orders for Pets
- Testing the Agent for Traps

Reading through this section will help you start the Agent and the Shopping Cart application and work on it directly. To know how we developed the Agent and the Application please refer to the “Steps Involved in Developing the Application” section.

Running the Agent

1. Go to `<Agent Toolkit Home>/reference/snmp/agent/bin` directory from the command prompt.
2. Set the JAVA_HOME path.
4. This will start the Agent at default port 8001.
5. Once the Agent is started, the Shopping Cart application UI also gets opened as follows:

![Shopping Cart Application](image)

```plaintext
Item Count: 0
Total Price: $0
```

<table>
<thead>
<tr>
<th>Pet Name</th>
<th>Unit Cost</th>
<th>Quantity</th>
<th>List Price</th>
<th>Image</th>
</tr>
</thead>
</table>

Add | Edit | Delete |

AdventNet Inc.
Testing the Agent with Default Values

The Shopping Cart Agent can be tested using the MIB Browser application. This application creates a Manager environment. By querying from the Manager side, you will be acting as a Shopkeeper where you can add rows/delete rows and so on. Whereas by placing orders using the application you become the customer and you are restricted to place orders only on the available stock. To test the Agent using MIB Browser,

- Go to `<Agent Toolkit Home>/bin` directory and start the MIB Browser application using `MibBrowser.bat/.sh` file.
- Load the `ShoppingCartMib.mib` from `<Agent Toolkit Home>/mibs` directory either using the `Load MIB Module` toolbar icon or `File -> Load MIB` menu bar.
- A Load MIB Dialog opens up wherein you have to go to the URL using the `Browse` button. Various Load options are available. Let us leave them to be with the default settings.
- Click `OK` and the MIB gets loaded.
- Start querying the following groups:
  - shoppingCartTable
  - shoppingCartSummary
- The tables can be queried by doing a `GET / GET_NEXT / SET` request. By default there will not be any entries and hence you will get "No data available in this tree" error. Follow the steps given to below to add entries through the application or through the MIB Browser and query the Table for it to respond with the values.
- The Scalars can be queried with `GET / GET-NEXT` requests alone as they are read-only variables.
- You can find the Agent responding with the values present in the shopping cart.

Placing Orders for Pets

- Say for example, you add an entry in the Shopping Cart application as follows and click OK,
The entry gets added as below in the UI.

Then, on querying the Table columns from the MIB Browser, the following results gets displayed:

- **PetName**: Parrot
- **unit cost**: $20
- **Quantity**: 1
- **List Price**: $20

The scalar group responds with `totalItemCount` as 1 and `totalPrice` as 20.

Even if values are set from the MIB Browser the Shopping cart application gets updated with the values.

Say, if the pet "dog" needs to be purchased, the ASCII values of d-o-g has to be given as an instance in the MIB Browser object field and a Multivarbind request has to be sent to the Table column. This will add a row and the Shopping Cart Application UI also gets updated with these values. The screen shot given below depicts the scenario:
Even if the row is deleted or modified in the Shopping Cart Application, the values are removed/modified accordingly when queried from MIB Browser. This happens vice versa also.

Testing The Agent for Traps

For this Reference Implementation, the Agent has been configured such that it generates Traps when the stock level reaches 15 for any pet item. To test the same follow the steps given below. To know how Trap or Notification is implemented go through the steps mentioned in "Implemented SNMP Traps" section.

To view Traps from the Agent, follow the steps given below:

- Start the Agent from <Agent Toolkit Home>/reference/snmp/agent/bin directory using run.bat or run.sh file. Set the JAVA-HOME path in case it is not set.
- The Agent gets started along with the Shopping Cart Application UI.
- Start the MIB Browser application from <Agent Toolkit Home>/bin directory.
- Load the Shopping Cart MIB in MIB Browser.
- Also open the Trap Viewer in MIB Browser application. The Traps are listened at port 162 by default. Change the port settings to 8003 (Manager's Port) and click on Add in the Trap Viewer and Start listening for Traps.
- Now go to the Shopping Cart Application and use Add button to place an order for 16 Parrot items. You can also add 16 Parrot items from the Manager (MIB Browser) itself.
Whichever way the Pet item is added a Trap is generated and can be viewed in the Trap Viewer as below, as the inventory level has reached 15.

To know more on Testing, please refer to Testing the Agent section in Building SNMP Agents of Product Help documentation. Further chapters will discuss the implementation details of Scalar group and Tabular group.
Steps Involved in Developing the RI

Created a MIB

- Overview
- Defining the Shopping Cart MIB Module
- Defining the Scalar Group
- Defining the Tabular Group and
- Defining Notifications

Overview

Based on the resources discussed in Application Notes, the MIB has to be defined. In this section, the steps involved in defining the MIB for the application is discussed. This will be the first step for developing a SNMP Agent for the existing Shopping Cart application.

AdventNet Agent Toolkit provides the tool "MIB Editor" for defining SMIV1/v2 mibs. The application can be started from the Agent Toolkit launcher or from <Agent Toolkit Home>/bin directory using the MibEditor.bat or MibEditor.sh file.

We have defined a MIB module under which 3 groups exist. The basic work to land up with the "ShoppingCart" MIB module is:
- Define the Shopping Cart MIB Module
- Define the Scalar Group
- Define the Tabular Group and
- Define Notifications

Note:
1. To know more on defining trap type and notification type construct please refer the Defining a MIB section in the Building SNMP Agents section of Product Help documentation.
2. Also refer the product document for defining the external index column.
3. Please see the final MIB for the ShoppingCart application in <Agent Toolkit Home>/mibs directory.

Defining the "ShoppingCart" MIB Module

To define the Shopping Cart MIB Module the following steps need to be followed.
- Start the MIB Editor tool.
- Select File -> Create Mib Module menu or toolbar icon to create the mib module.
- A dialog box prompts asking to fill in the details of the mib. For this example, the following details have to be used.
  - Module name - ShoppingCartMibModule
  - ModuleDirectory - mibs (by default).
  - MIB Version - Choose SMIV2 for this case.
  - Root OID - enterprises (by default).
Click OK.

The **ShoppingCartMibModule** gets added to the left side MIB Tree view with a module identity under enterprises.

Select **moduleidentity** and right click on it to enable **Modify** option.

Click **Modify** that enables the Module Identity tab where again details of the MIB Module has to be specified. The following values are specified,

- Module Identity - adventnet (this has to start in lower case letters)
- Sub ID - The subid to be mentioned. Default value is 1.
- Parent ID - Disabled.
- Last-Updated - Disabled.
- Organization - AdventNet Inc.,
- Contact Info - agent-support@adventnet.com
- Description - Reference Implementation

Now click "SaveNode". Ignore Revision, Revision Description details. This is required when any changes are need to be made in the MIB and maintained later.

The **ShoppingCartModule** mib can be seen in the left side frame of the MIB Editor with adventnet below the enterprises key. Thus a new mib module is created.

Now, to define an Object Identifier, choose "adventnet" in the MIB tree and add a new node by selecting **Operations -> Add Node** from the menu bar (or) from the pop-up menu that appears on right click of the mouse (or) by clicking on the toolbar icon.

Note: As specified here, to add a new node any of the above said options can be used. These options are implied and hence can be used when you try to add a new node anywhere in the MIB.

Select Object Identifier and a tab gets displayed in the right frame.

Enter

- The Object Identifier name as "shoppingCartModule".
- Make sure that the Parent node selected in the combo box is "adventnet".
- Also fill the Sub ID no as 571. (i.e shoppingCartModule has an unique node for the enterprise named "adventnet")

Then Press **Add To Module** button present below. The added "shoppingCartModule" gets displayed under "adventnet" in the mib tree.

The next step is to define the 3 management objects under the Shopping Cart MIB module. To begin with, define the **shoppingCartSummary** group which has different scalars under it.

**Defining Scalars**

From the application resources explained in Application Notes section, you saw the "scalar" group information as **totalItemCount** and **totalPrice**. Hence a scalar group is defined in the MIB which contains scalar objects. (management objects which can hold only a single value).

Similar to the steps explained in the previous section (defining shoppingCartModule under adventnet) define an Object Identifier "**shoppingCartSummary**" under the shoppingCartModule node with id as "3". The summary group gets created.

To add scalars under that group select "**shoppingCartSummary**" from the mib tree in the left frame and right click on it. Use **Add -> Scalar Object** to add a scalar variable.

The MIB tree in the left frame gets deactivated and all the possible constructs (depending on whether the MIB file is v1 or v2) which can be added to the parent node gets activated in the wizard interface.
In the right side tab panel fill in the following details:
  - Object Type - totalItemCount
  - Sub ID - 1
  - Parent - shoppingCartSummary.
  - Syntax - Integer32 (Select from the combo-box items).
  - Max-Access - read-only.
  - Status - Current.
  - Description - The total number of Pet items.
  - Reference - (optional)
  - Defval - Need not specify anything.

Then click on Add to Module button. The "totalItemCount" object gets added in the mib tree of left frame.

Similarly define the node "totalPrice" under the "shoppingCartSummary" node with id as 2. Now that the scalars are defined the next step would be define tables for the mib.

**Defining Tables**

Steps involved in defining the table :-

- Use **Add-> Table Object** construct to add the table **shoppingCartTable** under **shoppingCartModule** group.

- The OBJECT TYPE Table construct opens up with the following columns in the right side tab. Fill in the required details:
  - Object Type - shoppingCartTable
  - Sub ID - 2
  - Parent - shoppingCartModule
  - Max-Access - not-accessible
  - Status - current
  - Description - This Table contains information about the items purchased in the shopping cart. Use "..." button for adding the content.
  - Reference - Need not specify.

- Click Add Table.

- The Table gets listed in the right side frame itself.

- Now to add the Table entry fill in the following details:
  - Object Type - shoppingCartEntry
  - SubID -1
  - Table - shoppingCartTable
  - Syntax - shoppingCartEntry
  - Max-Access - not-accessible
  - Status - current
  - Need not fill in Description, Reference and Index column.

- Now **Add Entry**.

- The **shoppingCartEntry** gets listed below **shoppingCartTable** and the text-fields for adding column entries gets listed.

- Just fill in the details as given below to add the first column petName,
  - Object-Type - petName
- Sub Id - 2.
- Entry - shoppingCartEntry
- Index - Check the Index box. Now this column acts as the index column for the whole Table.
- Syntax - DisplayString.
- Max access - Not Accessible
- Status - current
- Description, Reference, Default values are optional.

- Click "Add Column to Table".
- The next column's Object-Type is "unitCost". The syntax is "Integer32" with Max-Access "read-only" and Sub id "3". Index box need not be checked here. Finally, click Add Column to Table.
- The next column's Object-Type is "quantity". The syntax is "Integer32" with Max-Access "read-write" and Sub id "4". Index box need not be checked here. Finally, click Add Column to Table.
- The next column's Object-Type is "listPrice". The syntax is "Integer32" with Max-Access "read-only" and Sub id "5". Index box need not be checked here. Finally, click Add Column to Table.
- The next column's Object-Type is "rowStatus". The syntax is "Row Status" with Max-Access "read-write" and Sub id "6". Index box need not be checked here. Finally, click Add Column to Table.
- Add to Module gets activated only when an index column is present.
- Since the petName column is already made as the index column, this gets enabled in the beginning itself. Now Press Add To Module button.
- The Table gets added in the mib tree under "shoppingCartModule".

**Defining Notifications**

"out of Stock" Notification.

We have also defined a Notification group. Traps/Notifications are implemented in a manner such that Notifications get generated for a particular state of change in the Shopping Cart. The state of change can be an event happening in the application. Thus the notification implemented is,

- outofStockNotification

As per our case, the SNMP Agent which we are trying to develop has to send Notifications to Managers when the following occurs:

- Stock count reaches 15 : - The maximum inventory level is 20 for each item and when any of the pet item's stock reaches 15 a Notification is generated to the Managers stating that the stock level is reducing.

If this is the requirement, then we have to define a notification in the MiB (as it is of v2 type). It is very easy to define traps/notifications using MibEditor. First, define the object group notification under the shoppingCartModule node with ID as 1.

Then, to define the Notification Type construct, select shoppingCartModule and right click on it. Use Add -> Notification Type pop-up menu. The Notification Type construct Tab gets enabled in the right.

- The Notification Type is outofStockNotification.
- Sub ID is 1.
- Parent Node is shoppingCartModule.
- Status is current.
The Objects are "petName and Quantity". This object can be chosen using the Browse option.

Fill in the other details if required.

Finally press "Add to Module" which adds the notification construct under shoppingCartModule.

Thus the MIB is defined.
Generated and Compiled Code for the MIB

- Overview
- Generating Stub Files
- Compiling Stub Files

Overview

In the chapter given earlier you saw how the MIB was defined for the Shopping Cart application. In this chapter you will see how stub files and instrumentation files are generated for the defined ShoppingCartMIB. For the given MIB, the Java stub files are generated using the tool provided by Agent Toolkit called "MIB Compiler". For more information on the tool, have a look at the Agent Toolkit Product documentation.

Generating Stub Files

Steps given below were followed for generating the stub/instrumentation files. These stub files are available in <Agent Toolkit Home>/reference/snmp/agent/src/refimpl/shoppingCart/snmp directory.

Created a New Project

The first step is to create a project.

- The Mib Compiler application is invoked from Agent Toolkit launcher. It can also be invoked from <Agent Toolkit Home>/bin directory using MibCompiler.sh or MibCompiler.bat file.
- The application opens up with the New Project tab.
- A workspace has to be specified for the Project. By default it starts with workspace1. Taking it to be the same, the following details are to be filled in:
  - Name of the Project: ShoppingCartAgent.
  - Location for the Project: As per the default location it is <Agent Toolkit Home>/snmpprojects/ShoppingCartAgent.

- Click OK.
- This creates a New Project. You can see ShoppingCartAgent.prj listed in the File View Tab.
- Since the Agent is already developed, the agent directory in <Agent Toolkit Home>/reference/snmp folder contains the source files and output files related to the ShoppingCartAgent project.

Loaded the ShoppingCart MIB

The next step is to load the ShoppingCart MIB that was developed using MIB Editor application.

- Load the ShoppingCartMib.mib that we defined, from <Agent Toolkit Home>/mibs directory by selecting the File --> Load Mib from the menu bar or clicking the Load MIB module from file toolbar icon.
- The ShoppingCartMib gets automatically loaded in the MIB View tab and gets listed under Loaded MibModules.
Configured Agent Settings

Now you have to configure all the necessary settings for the Agent using the menus available.

- Go to **Project -> Settings** menu from the menu bar of MIB Compiler application.
- You get the general settings listed.
- Do the following changes leaving others to be as default:
  - **Version** : to **v2c** in General Panel.
  - **Package Name** : **com.adventnet.refimpl.shoppingcart.snmp** in Source Generation -> General Panel.
  - **Agent Name** : **ShoppingCartAgent**
  - **Table** : Choose Runtime Memory in Source Generation -> Storage Model Panel.

Generated Stub Files

- Now that the settings are configured source files can be generated.
- Selecting **Build-->Generate source** menu from the menu bar generates the source code.
- On successful code generation a confirmation message saying that "Code generation succeeded for Project : ShoppingCartAgent.prj" can be seen.
- The files get listed under the Shopping Cart Agent Project in the File View menu.
- The output and source directory are
  - **<Agent Toolkit Home>/reference/snmp/agent/bin**
  - **<Agent Toolkit Home>/reference/snmp/agent/src** respectively. (This is where the files related to Shopping Cart Agent project are present)

Thus code is generated for the Shopping Cart MIB.

Files Generated

Following files get generated for the shoppingCartMib.mib:

- **BaseTableEntry.java** - Required for ShoppingCartEntry.java
- **BaseTableRequestHandler.java** - Required for ShoppingCartTableRequestHandler.java
- **ShoppingCartAgent.java** - Main File
- **ShoppingCartTrap.java** - Trap File.
- **ShoppingCartEntry.java** - Table Entry file.
- **ShoppingCartTableRequestHandler.java** - Table Request Handler file.
- **ShoppingCartSummaryInstrument.java** - Scalar Instrument file.
- **ShoppingCartSummaryRequestHandler.java** - Scalar Request Handler file.

Explanation for the Table, Scalar and Trap files are provided in Implementing Tables, Scalars and Traps sections respectively.
Main File

The Main file takes care of all the registrations. The following code has been added in the main file. This file has been instrumented for the instantiating the Application Starter.

```java
/* User code starts here */
// Starting shopping cart application
ApplicationStarter appl = new ApplicationStarter();
Thread appThread = new Thread(appl);
appThread.start();
/* User code ends here */
```

This method is required for providing the Reference for the Application.

```java
/* User code starts here */
class ApplicationStarter implements Runnable{
    public void run()
    {
        ShoppingCartApplication frame = new ShoppingCartApplication();
        ReferenceProvider.setShoppingCart(frame);
        frame.setVisible(true);
    }
}while
/* User code ends here */
```

Reference Provider

This is an additional file created for the Shopping Cart application. The above code in the Main file provides reference for the Application. The instrumentation part in the scalar file and table file takes reference from this file.

Compiled Stub Files

The next step is to compile the stub files to form a complete Agent. Selecting Build -> Compile Source menu from the menu bar will compile all the source files and form the Agent. On successful compilation a confirmation message saying that the code has been compiled successfully can be seen.

The majority of work gets over here and now you can run the tutorial to test the Agent with default values. To run and test the Agent please refer to Running and Testing the Application. To know how the Shopping Cart application has been instrumented to return specific values on querying, please refer to the following sections : Implementing Scalars, Implementing Tables and Implementing Traps.
Implemented Scalars

- Overview
- Instrumenting Scalars

Overview

In the previous section you saw how the stub files are generated. This section deals with the files generated for the scalar group and how they are instrumented.

The file generated for shoppingCartSummary group are:

- shoppingCartSummaryInstrument.java and
- shoppingCartSummaryRequestHandler.java

The instrument file contains all the getter and setter methods for `totalItemCount` and `totalPrice` variables. Whereas the Request Handler file has the `processGetRequest`, `processGetNextRequest` and `processSetRequest` methods.

Instrumenting Scalars

Have a look at the extra code added for instrumentation in the `shoppingCartSummaryInstrument.java` file present in between the tags given below

/* User code starts here */
/* User code ends here */

```java
In SNMP GET Request for totalItemCount
/* User code starts here */
com.adventnet.agent.application.shoppingcart.ShoppingCart shoppingCart = ReferenceProvider.getShoppingCart();
if(shoppingCart == null)
    throw new AgentException("null reference got", CommonUtils.GENERR);
totalItemCount = new java.lang.Integer (shoppingCart.getTotalItemCount());
/* User code ends here */
```

```java
In SNMP GET Request for totalPrice
/* User code starts here */
com.adventnet.agent.application.shoppingcart.ShoppingCart shoppingCart = ReferenceProvider.getShoppingCart();
if(shoppingCart == null)
    throw new AgentException("null reference got", CommonUtils.GENERR);
totalPrice = new java.lang.Integer (shoppingCart.getTotalPrice());
/* User code ends here */
```

These methods actually gets the total count of items and total price from the Shopping Cart application and provides the data on request from the Manager.

The instrumentation code has to be filled between these tags in order to support code merging when regenerated. Note that `totalItemCount` and `totalPrice` are "read-only" variables defined in the MIB and doesn't make sense to do a SNMP SET to this variable. So, instrumenting for setter methods is not necessary.
When a get request is made from the manager for the scalar variable totalItemCount,

- The Agent calls the \texttt{getTotalItemCount()} method of \textit{ShoppingCartSummaryInstrument} object.

- Inside this method, the \textbf{ShoppingCart object} is obtained by calling the \texttt{getShoppingCart()} method of Reference Provider object.

- The required \textit{TotalItemCount} of the shoppingCart application is retrieved using \texttt{getTotalItemCount()} of \textbf{ShoppingCart object}.

- The \texttt{getTotalItemCount()} of \textit{ShoppingCartSummaryInstrument} object returns the value obtained from the ShoppingCart application. Thus, the Agent sends this response to the Manager.
Overview

In this chapter lets see how instrumentation is done for the table using the Vector approach (Agent Table Model). The files that get generated for a Table are:

- ShoppingCartTableEntry.java
- ShoppingCartTableRequestHandler.java.

Before going in detail into instrumenting tables using vector approach, let us see how table processing is done before adding our code in the generated files given above.

Processing a Table

Table handling differs from scalar handling. For a scalar there is only one instance. But in case of a table there can be any number of instances. A row in a table can be uniquely identified by the value of its index column. This unique identification is called as the 'instance' of the row. For e.g. the shoppingCartTable has an index column 'petName'. The variables in this column are used to uniquely identify a row in that table.

About Vector storage (AgentTableModel)

For every row in the shoppingCartTable, an instance of the ShoppingCartEntry object is created. The ShoppingCartEntry object stores the values of all the columns for a single row in the table. It has getter and setter methods for getting and setting the value of a particular column. It also stores the instance of the row as an int array. Thus a com.adventnet.refimpl.shoppingCart.shoppingCartEntry object will correspond to a single row of the table.

The shoppingEntry object is stored in a vector sorted by their instance. The vector resides inside the com.adventnet.utils.agent.AgentTableModel object associated with the shoppingCartTableRequestHandler class. (Thus all the xxxRequestHandler class generated for tables will have its associated AgentTableModel as a member variable).

But in our case, since we have used Runtime Memory option for Persistence of Tables, the vector does not store any of the values. Instead, the values are stored in an external application and on every query for that Table, that external application is called from the Request Handler file. The value received is updated in the vector and the response is sent. The vector removes the value from the memory soon after sending the response.

Processing a Request

When any Get / GetNext / Set request method has reached the Agent from the Manager meant for shoppingCartTable, the "processGetRequest", "processGetNextRequest" and "processSetRequest" will be called respectively in the shoppingCartTableRequestHandler. Following procedure is adopted while processing a request.
When a get request reaches the Agent for shoppingCartTable

- shoppingCartTableRequestHandler's processGetRequest method will be called. This method will find the instance meant for the incoming request.
- After finding the instance, it will take the shoppingCartEntry object corresponding to that instance from the external application.
- The following line is added to get the value from the external application:
  
```java
/* User code starts here */
GetDataFromApplication(inst, SnmpAPI.GET_REQ_MSG);
/* User code ends here */
```
- The method "GetDataFromApplication" is added towards the end of the java file.
- Later the value received is stored in the vector residing in the AgentTableModel.
- The value is retrieved from the Agent Table Model using the method given below:
  
```java
entry = (ShoppingCartEntry)tModelComplete.get(inst);
```
- Instance is nothing but the int[] holding the information of the indexes of the table. For shoppingCartTable, the "inst" holds the information of the index, "petName".
- Now find the exact column for which the request is made (for example "unitCost")
- Call the corresponding shoppingCartEntry object's getter method to get the value (in this case call getunitCost() method to get the value of "unitCost")

When a getnext request reaches the agent for shoppingCartTable:

- shoppingCartTableRequestHandler's processGetNextRequest method will be called.
- Similar to getrequest, the appropriate shoppingCartEntry will be picked up from the external application and stored in the vector.
- The following lines are added to get the value from the external application:
  
```java
/* User code starts here */
GetDataFromApplication(null, SnmpAPI.GETNEXT_REQ_MSG);
/* User code ends here */
/* User code starts here */
GetDataFromApplication(inst, SnmpAPI.GETNEXT_REQ_MSG);
/* User code ends here */
```
- The method "GetDataFromApplication" is added towards the end of the java file.
- Later the value received is stored in the vector residing in the AgentTableModel.
- The value is retrieved from the Agent Table Model using the methods given below:
  
```java
entry = (ShoppingCartEntry)tModelComplete.getFirstEntry();
entry = (ShoppingCartEntry)tModelComplete.getNext(inst);
```

When a set request reaches the agent for shoppingCartTable:

- shoppingCartTableRequestHandler's processSetRequest method will be called.
- Similar to getrequest, the appropriate shoppingCartEntry will be picked up from the external application and stored in the vector.
- The following code is added to get the value from the external application:
  
```java
/* User code starts here */
GetDataFromApplication(null, SnmpAPI.GETNEXT_REQ_MSG);
/* User code ends here */
/* User code starts here */
GetDataFromApplication(inst, SnmpAPI.GETNEXT_REQ_MSG);
/* User code ends here */
```
- The method "GetDataFromApplication" is added towards the end of the java file.
- Later the value received is stored in the vector residing in the AgentTableModel.
- The value is retrieved from the Agent Table Model using the methods given below:
  ```java
easy = (ShoppingCartEntry) tModelComplete.get(inst);
```
- This finds the exact column for which the set request is made (for example "quantity")
- And calls the corresponding shoppingCartEntry object's setter method to set the value (in this case call setQuantity(Integer) method to set the value of "quantity").
- The value is updated in the application using the following method: UpdateApplication().

**Instrumenting Tables**

Each shoppingCartEntry in the external application that gets stored in the vector will provide the information of each pet. Imagine if this vector is present inside the AgentTableModel. Then when any SNMP request reaches the shoppingCartTableRequestHandler, the AgentTableModel methods will return the appropriate shoppingCartEntry from the vector which is maintained by the shoppingCartResource.

Thus you need to add the below code from which the values are taken for storing in the vector. (referred as External application previously).

```java
/* User code starts here */

public void GetDataFromApplication(int[] inst, byte commandType)
{
    com.adventnet.agent.application.shoppingcart.ShoppingCart
    shoppingCart = ReferenceProvider.getShoppingCart();
    com.adventnet.agent.application.shoppingcart.ShoppingItem item =
    null;
    tModelComplete.getTableElements().removeAllElements();
    if (commandType == SnmpAPI.GETNEXT_REQ_MSG) {
        if (inst == null)
        {
            item = shoppingCart.getItemAt(0);
            if (item != null)
            {
                createAndAddNewShoppingCartEntry(
                    item.getPetName(),
                    item.getUnitCost(),
                    item.getQuantity(),
                    item.getListPrice(),
                    new Integer(ACTIVE));
            }
            return;
        }
        else{
            int[] tempInst = new int[inst.length - 1];
            for (int i=1; i<inst.length; i++)
            {
                tempInst[i-1] = inst[i];
            }
            String instPetName = utils.integerArrayToString(tempInst);
            for (int i=0; ; i++)
            {
                item = shoppingCart.getItemAt(i);
                if (item == null)
                {
```
return;
}  
else
{
    String petName = item.getPetName();
    if (petName.length() > inst[0])
    {
      createAndAddNewShoppingCartEntry(
          item.getPetName(),
          item.getUnitCost(),
          item.getQuantity(),
          item.getListPrice(),
          new Integer(ACTIVE));
    }
    else if(petName.length() == inst[0])
    {
      if (petName.compareTo(instPetName) > 0 )
      {
        createAndAddNewShoppingCartEntry(
            item.getPetName(),
            item.getUnitCost(),
            item.getQuantity(),
            item.getListPrice(),
            new Integer(ACTIVE));
      }
    }
    if(inst == null)
    {
      item = shoppingCart.getItemAt(0);
    }
    else
    {
        // Removing length part
        int[] tempInst = new int[inst.length - 1];
        for(int i=1; i<inst.length; i++)
        {
            tempInst[i-1] = inst[i];
        }
        String petName = utils.integerArrayToString(tempInst);
        item = shoppingCart.getItem(petName);
        // tModelComplete.getTableElements().removeAllElements() ;
        if (item != null)
        {
            createAndAddNewShoppingCartEntry(
                item.getPetName(),
                item.getUnitCost(),
                item.getQuantity(),
                item.getListPrice(),
                new Integer(ACTIVE));
        }
    }
    }
public boolean UpdateApplication()
{ 
    com.adventnet.agent.application.shoppingcart.ShoppingCart
    shoppingCart = ReferenceProvider.getShoppingCart();
    ShoppingCartEntry entry = null;
    for (int i = 0; i < tModelComplete.size(); i++)
    {
        entry = (ShoppingCartEntry)tModelComplete.getEntryAt(i);
        try{
            if
                (shoppingCart.updateItem(entry.getPetName(),entry.getQuantity())
                == false)
                { try{
                    shoppingCart.addItem(entry.getPetName(),entry.getQuantity());
                } catch(Exception e)
                { System.out.println("Exception while updating the application.");
                    return false;
                }
            }
        } catch(Exception e)
        { System.out.println("Exception while updating the application.");
        }
        return true;
    }
    /* User code ends here */
}

Conclusion on Vector Approach

This model of table instrumentation is very simple and straight forward. If the application has necessary details for the table as a static information which does not change dynamically at the runtime of the Agent, then this vector approach is highly recommended.
Implemented SNMP Traps

- Overview
- Requirement
- ShoppingCartAgentTrap.java (Trap File)
- ShoppingCartAgent.java (Main File)

Overview

Trap or Notification is generated by an Agent, to intimate the change of state of a particular variable in
the Agent to the Managers. In this Reference Implementation, Notification Type construct is defined
for a variable, the inventory level, and on any change of state of that variable, Notification is
generated. Lets discuss in detail the implementation details of sending Traps to the Managers from
our shoppingCart Agent.

Requirement

As per the Reference Implementation, the Agent has to send Notifications to the Managers when :

- inventory level is less than 5 when the user chooses 15 or above for every pet item.

The maximum inventory level available is 20. Say for example, if the customer places an order for
more than 15 for the pet Parrot, then a Notification is generated. This is done through the methods
implemented in the java file generated for Traps.

ShoppingCartAgentTrap.java

On generating source files for the MIB, the file ShoppingCartAgentTrap.java gets generated for
storing the details of Traps and Notifications. To generate Notifications for this particular case, a
separate thread is started in this Trap.java file. The thread monitors the Shopping Cart application
periodically and sends Notifications if any of the item quantity exceeds the limit (15).

Following is the code added in the trap file for declaring the variable.

```java
/* User code starts here */
monitorThreshold trapThread = null;
/* User code ends here */
```

Following is the code added in the trap file for instantiating the monitorThreshold application.

```java
/* User code starts here */
trapThread = new monitorThreshold();
/* User code ends here */
```

Following code added in the Trap file monitors the application periodically (5 seconds) and generates
notification when the quantity level reaches 15.

```java
/* User code starts here */
class monitorThreshold implements Runnable{
    public void run(){
        com.adventnet.agent.application.shoppingcart.ShoppingCart
        shoppingCart = null;
        com.adventnet.agent.application.shoppingcart.ShoppingItem item =
        null;
        while(true)
```
{ 
shoppingCart = ReferenceProvider.getShoppingCart();
for(int i=0; i<shoppingCart.getTotalItemCount(); i++)
{
shoppingCart = ReferenceProvider.getShoppingCart();
item = shoppingCart.getItemAt(i);
// For extra checking
if (item != null) {
// If the quantity exceeds 15 then trap will be sent periodically
try{
if(item.getQuantity().intValue() >= 15)
{
agentRef.getShoppingCartTable().createAndAddNewShoppingCartEntry(
    item.getPetName(),
    item.getUnitCost(),
    item.getQuantity(),
    item.getListPrice(),
    new Integer(1));
sendOutOfStockNotification(null);
agentRef.getShoppingCartTable().tModelComplete.getTableElements().
    removeAllElements();
}
} 
catch(Exception e)
{
 System.out.println("Problem in retrieving the data from the 
    application");
}
}
try{
Thread.sleep(5000);
} 
catch(Exception e)
{
 System.out.println("Problem in thread sleep");
}
}
public monitorThreshold getTrapThread()
{
return trapThread;

}  /* User code ends here */

It is enough if you call the above methods to send traps to the Manager.

ShoppingCartAgent.java

The main file ShoppingCartAgent.java has the following method included for starting the thread.

/* User code starts here */
Thread trapThread = new Thread(trapRef.getTrapThread());
trapThread.start();
/* User code ends here */
31.0 Migration Guide for SNMP Agents

Migration from 4.2 to 5.1.0

- Move Your Source Files
- Modify your Package Structure
- Methods Modified
- Methods Deprecated
- Methods Removed
- Tool for Migration

Migration from 5.0 to 5.1.0
Migration from 5.1.0 to 6.0.0

This section will be helpful for users who are using SNMP Agents in the earlier versions of Agent Toolkit Java Edition and now wish to migrate to the latest release of the Toolkit. The document explains how to

- Migrate from 4.2 to 5.1.0 release.
- Migrate from 5.0 to 5.1.0 release.
- Migrate from 5.1.0 to 6.0.0 release.

Migration from 4.2 to 5.1 Release

Move your Source files

- For easy migration, create a new Project with the 5.1.0 release product.
- The source files for the new project will be generated under <Agent Toolkit Home>/snmpprojects/projectname/agent/src/myCompany/myPackage directory.
- Remove all the generated files under this directory.
- Move all your source files from your previous release directory to the specified directory.
- Change the import statement as specified in "Package Structure Modified" section.
- Then compile the Agent.
- This will make your Agent work in 5.1 release.
- Please note that relevant absolute path has to be specified in the Main file where you have agentDir = <Agent Toolkit Home>/snmpprojects/projectname/agent/bin;.

Modify your Package Structure

The Package structure for certain imports have been modified and before generation the older imports have to be commented and new import statements have to be added. Package structure have been changed for the following import :

- com.adventnet.common.agent.*; - This import has to be commented and the following has to be included com.adventnet.agent.utilities.common.*;
Code Generation Changes

The following table lists the details of code generation that has to be implemented in 5.1 while migrating from 4.2.

<table>
<thead>
<tr>
<th>Functions</th>
<th>Current Implementation in 4.2</th>
<th>To be modified in 5.1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentSnmpGroup up() method</td>
<td>AgentSnmpGroup grp = new AgentSnmpGroup();</td>
<td>AgentSnmpGroup grp = new AgentSnmpGroup(this);</td>
</tr>
<tr>
<td>acltable</td>
<td>acl = new com.adventnet.snmp.snmp2.agent.AclTableRequestHandler((SnmpAgent)this, &quot;conf&quot;,&quot;acl.txt&quot;,false);</td>
<td>aclTable = new com.adventnet.snmp.snmp2.agent.AclTable((SnmpAgent)this,&quot;acl.txt&quot;,&quot;xml&quot;); acl = new com.adventnet.snmp.snmp2.agent.AclTableRequestHandler((SnmpAgent)this, aclTable); Also the following line should be added in variable declarations</td>
</tr>
<tr>
<td></td>
<td>super.setV1v2AccessControl(true); com.adventnet.snmp.snmp2.agent.VaclTableRequestHandler vACL = new com.adventnet.snmp.snmp2.agent.VaclTableRequestHandler ((SnmpAgent)this, &quot;conf&quot;,&quot;vacl.txt&quot;,false); vacl.addRegistrationListener(hdlr);</td>
<td>vaclTable = new com.adventnet.snmp.snmp2.agent.VaclTable((SnmpAgent)this,&quot;vacl.txt&quot;,&quot;xml&quot; );  vacl = new com.adventnet.snmp.snmp2.agent.VaclTableRequestHandler((SnmpAgent)this, vaclTable); Also the following line should be added in variable declarations</td>
</tr>
<tr>
<td></td>
<td>private com.adventnet.snmp.snmp2.agent.AclTable aclTable = null;</td>
<td>private com.adventnet.snmp.snmp2.agent.VaclTable vaclTable = null;</td>
</tr>
</tbody>
</table>

Note: The OCTET STRING is handled as java.lang.String in 4.2 whereas it is handled as byte[] in 5.1.0 release.

Methods Modified

If you have instrumented your implementation with the entry methods, then be aware that the name and argument are changed in the latest release. Some of the methods of 4.2 release modified in this 5.1 release are as follows:

- The method in AclEntry and AclEntryInterface has been changed.
  - `setAclAccess(int value)`; `--> setAccess(Integer value);`
Forwarding Entry Class

- `getManagerHost() -> getV1v2ManagerHost();`
- `getManagerPort() -> getV1v2ManagerPort();`
- `getRowStatus() -> getV1v2ManagerStatus();`
- `setManagerHost(SnmpVar var) --> setV1v2ManagerHost(java.lang.String host);`
- `setManagerPort(SnmpVar var) --> setV1v2ManagerPort(java.lang.Integer port);`
- `setRowStatus(SnmpVar var) --> setV1v2ManagerStatus(java.lang.Integer status);`

V3 Forwarding Entry Class

- `getManagerHost() --> getV3ManagerHost();`
- `getManagerPort() --> getV3ManagerPort();`
- `getRowStatus() --> getV3RowStatus();`
- `setManagerHost(SnmpVar var) --> setV3ManagerHost(java.lang.String host);`
- `setManagerPort(SnmpVar var) --> setV3ManagerPort(java.lang.Integer port);`
- `setRowStatus(SnmpVar var) --> setV3RowStatus(java.lang.Integer status);`
- `setV3ManagerUserContextName(SnmpVar var) --> setV3ManagerUserContextName(java.lang.String dbname);`
- `setV3ManagerUserName(SnmpVar var) --> setV3ManagerUserName(java.lang.String uname);`
- `setV3ManagerUserSecModel(SnmpVar var) --> setV3ManagerUserSecModel(java.lang.Long smodel);`
- `setV3SecurityLevel(SnmpVar var) --> setV3SecurityLevel(java.lang.Integer slevel);`

Methods Deprecated

Have a look at the deprecated methods:

1. `DynamicRegistrationWithCommunity, DynamicRegistrationWithInstance, & DynamicRegistration`  
   - `createDynamicRegistrationEntry. (Please note that new methods with extra arguments are included in the same name)`

2. `DynamicRegistrationWithInstance`  
   - `removeSubAgent(DynamicRegistrationEntry ent). (Please note that new methods with extra arguments are included in the same name).`

3. `SnmpAgent.`  
   - `setDebug.`
   - `setDebugOn.`
   - `setDebugOff.`

4. `SnmpTrapService.`  
   - `getTrapTableName`
   - `setTrapTableName`
   - `getTrapPathName`
   - `setTrapPathName`
   - `getV3TrapTableName`
   - `setV3TrapTableName`
   - `setPersistence`
   - `setFToVPersistence.`
5. AclTableRequestHandler
   - setPersistence
   - setFToVPersistence

6. VACLTableRequestHandler.
   - setPersistence,
   - setFToVPersistence.

7. Agent Param Options
   - getCommunity
   - getWriteCommunity

8. Traps
   - CreateV3ForwardingEntry.
   - createForwardingEntry
   - getId() and setId(SnmpVar var) in Forwarding Entry
   - getId() and setId(SnmpVar var) in V3 Forwarding Entry

Usage of PropertyRegistrationListener is not recommended. (SingleAgent Bean option is completely deprecated)

All these methods will still work with 5.1.0 release of Agent Toolkit (Java Edition) tool as they have not been removed.

Methods Removed

The Methods that are removed from the 5.1.0 release of Agent Toolkit (Java Edition) Which were already in the deprecated List of 4.2 Release)

1. DynamicRegistrationWithCommunity
   - Constructor
   - setV2Supported
   - setAgentRef

2. DynamicRegistrationWithInstance
   - setV2Supported

3. DynamicRegistration
   - setV2Supported
   - Constructor

4. AclTableRequestHandler
   - getFieldSeparator

5. Traps
   - createV3ForwardingEntry.
   - getId() and setId(SnmpVar var).
   - getId() and setId(SnmpVar var).

Tool for Migration

Need for the tool

In this 5.1.0 release of Agent Toolkit Java Edition a few extra columns have been added to the following tables to achieve complete functionality.
These tables earlier present in AGENT-SAMPLE-MIB are now available in AGENT-SNMP-CONFIG-MIB. The Tables have been renamed as

- v1v2TrapForwardingTable
- v3TrapForwardingTable
- proxyTable
- contextBasedProxyTable
- instanceBasedProxyTable.

respectively. The file names that created for these tables have also been renamed as follows:

<table>
<thead>
<tr>
<th>Old Name</th>
<th>New Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>forwardingTable.txt/xml</td>
<td>V1V2TrapForwardingTable.txt/xml</td>
</tr>
<tr>
<td>v3forwardingTable.txt/xml</td>
<td>V3TrapForwardingTable.txt/xml</td>
</tr>
<tr>
<td>Subagent.txt/xml</td>
<td>ProxyTable.txt/xml</td>
</tr>
<tr>
<td>SubagentWithCommunity.txt/xml</td>
<td>ProxyTable.txt/xml</td>
</tr>
<tr>
<td>SubagentWithInstance.txt/xml</td>
<td>ProxyTable.txt/xml</td>
</tr>
</tbody>
</table>

If you are an user of the previous release you may have stored entries in these text files with the old columns. To make the new columns available in the old text files, this utility has to be used. The additional columns will be created with some default values.

**How to run the tool**

- Go to `<Agent Toolkit Home>/bin` directory of 5.1.0 release.
- To run the tool make use of `SnmpMigrationTool.sh` or `SnmpMigrationTool.bat` depending on the OS.

**Details required for running the tool**

You will have to provide the following information once the utility is made to run.

- Name of the Table to be converted.
- Mode of the file (whether txt or xml).
- File name for the existing Table.
- File name to be created.

**Migration from 5.0 to 5.1.0 Release**

**SnmpTrapAppender Method**

This method has been modified in Release 5.1.0. To implement this feature in 5.1.0, please refer Sending Traps for Log Messages implemented using Log4j section in Sending Traps and Informs.

Other than this, there are no major changes involved while migrating from 5.0 to 5.1.0.

**Migration from 5.1.0 to 6.0.0 Release**

The following changes are involved while migrating from 5.1.0 to 6.0.0.
Modify your Package Structure

The package structure have been modified for certain imports. The following table depicts the changes in package structure in 6.0.0 compared to 5.1.0.

<table>
<thead>
<tr>
<th>Package Structure in 5.1.0</th>
<th>Package Structure in 6.0.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.adventnet.agent.utilities.common</td>
<td>com.adventnet.utilities.common (Common Utils)</td>
</tr>
<tr>
<td>com.adventnet.agent.utilities.xml</td>
<td>com.adventnet.utilities.xml.dom</td>
</tr>
<tr>
<td>com.adventnet.agent.utilities.xml.XMLIntender</td>
<td>com.adventnet.utilities.xml.indenter.XMLIntender</td>
</tr>
<tr>
<td>com.adventnet.agent.logging</td>
<td>com.adventnet.utilities.logging</td>
</tr>
</tbody>
</table>

Before generating the agent, you need to ensure that the latest package structure is followed.

In 5.1.0 release, the `java.util.TooManyListenersException` class was used. This has to be changed to `com.adventnet.utils.agent.RegistrationListenerException` class in 6.0.0.

Changes in Configuration Files

- **V3TrapForwardingTable** - new column 'v3ManagerHostType' added for IPv6 support.
- **v1v2TrapForwardingTable** - new column 'v1v2ManagerHostType' added for IPv6 support.
- **UsmUserTable** - two new columns added, UsmUserAuthKey and UsmUserPrivKey to provide persistence for AuthKey and PrivKey.

Backward compatibility is supported in the Agent Toolkit. So, if you are a 5.1 user having an older version of xml files, you can keep the 6.0.0 jars in the classpath and run the agent. The columns will be automatically added to the above-mentioned configuration files.