AgenPro 3.1

A Java SE Application
for SNMP Code Generation and Agent Simulation

Table Of Contents

1 AgenPro Manual Overview

The AgenPro manual is organized into the following main topics:

- **Code Generation Benefits**
  What are the benefits when generating code with AgenPro?
- **Setup**
  How to install AgenPro.
- **Managing MIBs**
  How to compile MIBs, store them in a MIB repository, and loading MIBs into the MIB tree.
- **Projects**
  How to create, open, and save code generation projects.
- **Code Generation**
  How to generate program code (stubs) for AGENT++ (C++) and SNMP4J-Agent (Java) agent development.
  How to create code generation templates for third party SNMP agent APIs and how to modify existing templates for AGENT++ and SNMP4J-Agent to meet special needs.
- **Simulation Agent**
  How to run a SNMP agent based on your project settings instantly from the AgenPro graphical user interface.
- **Tools**
  How to search the MIB tree, how to find duplicate OIDs, and how to export MIB modules to HTML and plain text files.

2 System Requirements

Minimum system requirements for AgenPro are:

- *Java Runtime Environment* (JRE) 6 or later installed.
- 128 MB RAM (AgenPro will use up to 64 MB RAM by default)
- ~ 50 MB free hard disk space (JRE not included)
- Color display with at least 1024x768 pixels (GUI only)
- Mouse or comparable input device (GUI only)

For the AgenPro Maven-Plugin:

- Maven 2.2.1 or later
- *Java Runtime Environment* (JRE) 6 or later installed.

3 Setup

Please read the section “System Requirements” on page 2 for prerequisites needed to install and run AgenPro. Then follow the below steps to install and setup AgenPro.

3.1 Installation

AgenPro can be installed as *Java Web Start* application or regular Java application. Java Web Start provides operating system integration, such as icons on the desktop and start menu integration. In addition, Java Web Start installation will provide better integration with your operating system and will keep AgenPro up-to-date automatically[1].

Java Web Start is supported by any recent Web browser. Depending on operating system and system setup, you need to install a Java Runtime Environment from *http://www.java.com* (see also “Java Runtime Environment (JRE) Installation” on page 3).

To run (install) AgenPro using Java Web Start, simply click on the link *https://www.agentpp.com/agen/AgenProWebStart.jnlp*.

To install AgenPro as classic Java application, first make sure you have installed a Java Runtime Environment (JRE) version 6 or later (see “System Requirements” on page 2). Then follow the steps set forth in section “Installation as Classic Java Application” on page 4.

3.1.1 Java Runtime Environment (JRE) Installation

Note: On many operating systems a JRE is already installed or the Web browser comes with a JRE. In that case you do no need to install another JRE if the installed one is of version 6 or later.

1. Download the latest JRE or SDK[2] from *http://www.java.com.*
2. Install the Java Runtime Environment (JRE) 6 or later on your system and add the `bin` directory of the JRE (or SDK) installation to your `PATH` environment variable (this is often already done by the installation process).
3. Log on to your system as the user who is supposed to use AgenPro and install AgenPro following “Installation with Java Web Start” or “Installation as Classic Java Application” on page 4.

[1] *Although Java Web Start caches resources locally, it checks for new versions regularly when the application is used and updates the application automatically.*

[2] *The Software Development Kit (SDK) is needed only if you want to generate code for Java or if you want to use the AgenPro Maven Plugin.*

3.1.2 Installation with Java Web Start

1. Log on to your system as the user who is supposed to use AgenPro.
2. Open the link *https://www.agentpp.com/agen/AgenProWebStart.jnlp* in your Web browser.
3. Java Web Start will download AgenPro and start it.
4. Follow the instructions to setup AgenPro in section "Setup" on page 4.

3.1.3 Installation as Classic Java Application

Download the `agenpro3.jar` file in a folder of your choice. Start the AgenPro application by

- double clicking it from your system's file explorer, or
- running:

```
java -jar agenpro3.jar
```

3.2 Starting AgenPro

If you have used Web Start to install AgenPro then you can start it from your systems application start menu or clicking on *https://www.agentpp.com/agen/AgenProWebStart.jnlp*.
Otherwise double click the downloaded `agenpro3.jar` file or run `java -jar agenpro3.jar` from the command line. When AgenPro is started for the first time, you will be prompted for your license information.
If you are using a restricted license you can upgrade it later without reinstalling AgenPro by choosing Help>License... from the main menu.

Please enter your license key including blanks! The license key is case sensitive.

3.3 Setup

Once you have started AgenPro and entered your license information, choose File>Install... to install AgenPro MIB files and repository as well as other accompanied files on your system.
At first application start, you will be automatically asked to specify an empty installation directory for AgenPro accompanied files.
Every time AgenPro is updated, regardless whether through Web Start or manually, and the structure or version of the accompanied file set has changed, you will be asked to install/update those files again. You have then the choice to install the files to a new location or update the existing location.

Note: When updating accompanied files, modifications on that files may get lost. Thus, it is recommended to use a new installation directory and delete the old manually if it is not needed any more.

3.3.1 Install Templates, Example Projects, and MIBs

Once you have started AgenPro and entered your license information, choose File>Install... to install AgenPro MIB files and repository as well as other accompanied files on your system.

3.4 Upgrade

When installed through Java WebStart, AgenPro will be automatically updated through Web Start on application startup, if a newer version is available on the AgenPro web site.
If a newer version of the accompanied file set is available with the new version, AgenPro will ask you to install them over the current installation location. If you confirm the installation, AgenPro will overwrite existing files with their newer version

3.5 Uninstall

If AgenPro has been installed using Java Web Start, then run

```
javaws -viewer
```

to bring up the Java Web Start viewer. Select “AgenPro” from the cached applications list and use Delete from the context menu to remove AgenPro from your Web Start cache. This will also remove any desktop integration from your start menu.
If you have not used Java Web Start to run AgenPro it is sufficient to remove the `agenpro3.jar` file.
Both uninstall procedures described above do not remove any accompanied file sets installed by AgenPro.
AgenPro holds its configuration data in the `AgenPro3.cf` (`AgenPro2.cf` for AgenPro 2.x) file in your home directory. To completely uninstall AgenPro, this file has to be removed manually. By removing it, you will have to reenter your license information - as well as other configurations - when you reinstall AgenPro.

3.6 Preferences

With the preferences dialog accessible from the toolbar (☑) or with Edit>Preferences from the menu bar, AgenPro‘s settings are configured.

3.6.1 Persistency

AgenPro stores compiled MIB modules in a MIB repository (see “MIB Repository” on page 7). Here you can specify where the MIB repository directory is located. Alternatively, you can also create a new (*empty!*) directory for using it as MIB repository.

3.6.2 View

With the view options, you can split the main window horizontally or vertically, change the font size for the SMI text panel and whether SMI text should be syntax highlighted using colors.

3.6.3 General

Some look & feels may cause exceptions on certain platforms, if you encounter such an exception and you cannot start AgenPro to change the look & feel to the default again, then remove the row with starting with `LookAndFeel` *from the* `AgenPro3.cf` *file in your home directory.*

In the general settings section, you can select a different look & feel for the GUI. You can either select on of the look & feels in the combo box, or enter the class name of a look & feel which then must be part of AgenPro's class loader path.

In addition, you can let AgenPro ask you before it overwrites files or you can activate the MIB list support. When activated, you can read the paths of the MIB files to compile from a text file, where each row contains a MIB file path.

4 MIB Repository

A MIB repository is a directory that AgenPro exclusively uses to store compiled MIB modules in an internal format. Before a MIB module can be loaded into AgenPro's MIB tree and then used by subsequent code generation operations, it has to be compiled and stored into a MIB repository.

To Create a MIB Repository:

1. From the file menu choose Set MIB Repository. A File Open menu dialog box will appear.
2. Navigate through the file system to the directory where you want to create the MIB Repository.
3. Within that directory, create a new folder by clicking on the Create New Folder (🗀) button. The new folder can be renamed by double-clicking it.
4. Choose the new (or any other *empty* folder) by selecting it. Click Open.

Note: Do not double-click the new folder! Otherwise you cannot select the folder itself.

As long as a MIB Repository directory is used by AgenPro, it must not be altered outside AgenPro. Once a valid MIB Repository has been set, you may compile MIB files to store them in the repository.

To Select a MIB Repository:

1. From the file menu choose Set MIB Repository. A File Open menu dialog box will appear.
2. Navigate through the file system and select the MIB Repository directory you want to use.
3. Click Open.

Note: Do not double-click the new folder! Otherwise you cannot select the folder itself.

The MIB repository will be verified. If any inconsistent or corrupted MIB modules are found, a dialog will be displayed with instructions to repair the repository.

5 MIBs

SNMP Management Information Base (MIB) specifications are documents containing definitions of management information so that network systems can be remotely monitored, configured, and controlled. AgenPro makes extensive use of all machine readable information within MIBs. This information is available as so called MIB modules and is used when running code generation jobs.

To avoid errors in the code generation process that could be hard to find and problematic to solve, AgenPro does an extensive and precise syntax and semantic check when compiling MIB files.

5.1 Getting MIB Files

MIB specifications developed by the IETF working groups contain prose descriptions and references to other documents that enclose the actual MIB module(s). AgenPro compiles SMIv1 (RFC 1155) and SMIv2 (RFC 2578-2580) conforming MIB modules. However, MIB modules have to be extracted from RFC specifications before they can be compiled.

Whereas extracting MIB modules from RFC documents can be done manually by removing any prose descriptions, page headers, and footers from an RFC MIB text document, it is much easier and faster to use the Tool “Extract SMI Modules from RFC Documents” on page 62.

5.2 Compiling MIBs

Before you can compile MIB modules into AgenPro's internal format, a MIB repository has to be created where the compiled MIBs are stored. During the first startup of AgenPro you will be asked to specify a MIB repository.

Precompiled MIBs

AgenPro comes with a set of precompiled SMIv2 MIBs which are located in the repository directory of the AgenPro installation. AgenPro uses that directory as its initial default repository.

To Compile MIBs

1. From the **File** menu, choose **Compile MIBs** (or from the main tool bar). A file open dialog will appear.
2. Choose a MIB file, ZIP file or a directory and click **Open**. If you choose a file, then that file will be compiled and all contained MIB modules (typically one) are stored into the MIB repository

   If you choose a directory or a ZIP file, then recursively all contained files will be parsed. All successfully parsed MIBs will be automatically sorted by their dependencies and then compiled into the MIB repository. Directories may also contain ZIP files.

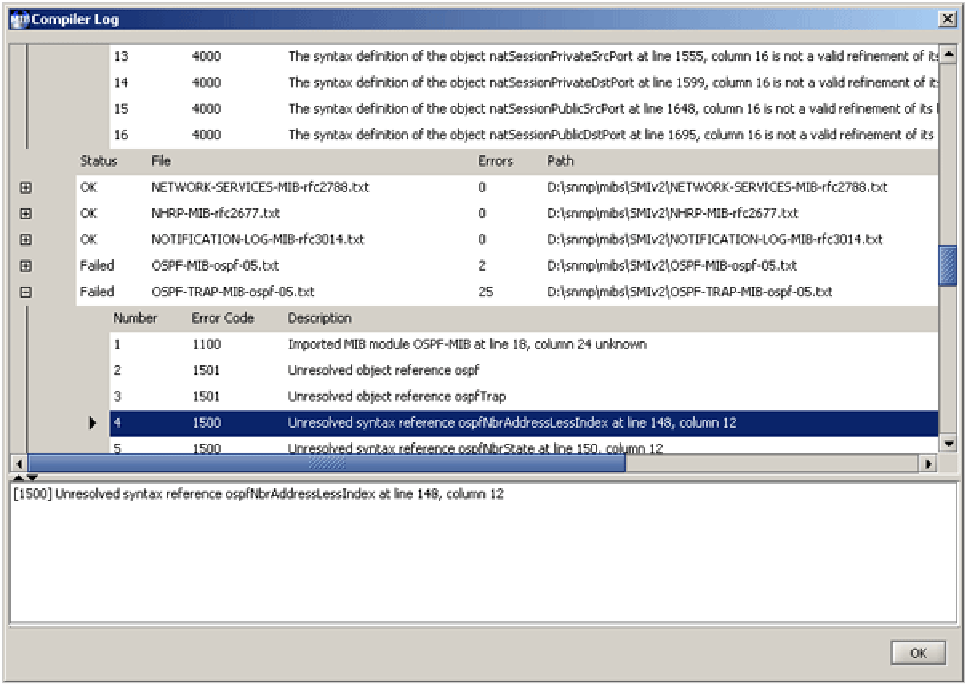
   If MIB list support is activated in Preferences *selecting a single file for MIB compilation will cause AgenPro to parse the file as a list of MIB file names to compile. The text file must then contain a single file path per row.*

3. After compilation a message dialog with summary information is shown.
4. Press **Details** to open the Compiler Log window (see Figure 1). It lists status information for each MIB file compiled. A MIB file that failed to compiled has a status *Failed* in the status column. To view the errors detected for that file, click on the + sign in the first column of that row. The error list will then be expanded. By double clicking on an error description, the failed MIB file is opened for editing and later recompilation.
5. The MIB modules of the successfully compiled MIB files are automatically stored in the MIB repository. From there, the MIB modules can be loaded into the MIB Explorer application.
   Existing MIB modules will be overwritten (updated). If you do not want to change any MIB modules that already exist in the current MIB repository, then use **Compile New MIBs** from the **File** menu.

5.2.1 Compiler Log

The Compiler Log dialog lists the status of all MIB files of a compilation run. If the compilation of a MIB file failed, the **Status** column displays the text *Failed*. The error messages for that file can be expanded by clicking on the + sign of the file's row. By selecting an error message, its description text will be displayed in the text pane on the bottom of the dialog.

The file name of the MIB file is displayed in the File column, whereas its complete path is displayed in the Path column.



13	4000	The syntax definition of the object natSessionPrivateSrcPort at line 1555, column 16 is not a valid refinement of it
14	4000	The syntax definition of the object natSessionPrivateDstPort at line 1599, column 16 is not a valid refinement of it
15	4000	The syntax definition of the object natSessionPublicSrcPort at line 1648, column 16 is not a valid refinement of its
16	4000	The syntax definition of the object natSessionPublicDstPort at line 1695, column 16 is not a valid refinement of its

	Status	File	Errors	Path
⊞	OK	NETWORK-SERVICES-MIB-rfc2788.txt	0	D:\snmp\mibs\SMIv2\NETWORK-SERVICES-MIB-rfc2788.txt
⊞	OK	NHRP-MIB-rfc2677.txt	0	D:\snmp\mibs\SMIv2\NHRP-MIB-rfc2677.txt
⊞	OK	NOTIFICATION-LOG-MIB-rfc3014.txt	0	D:\snmp\mibs\SMIv2\NOTIFICATION-LOG-MIB-rfc3014.txt
⊞	Failed	OSPF-MIB-ospf-05.txt	2	D:\snmp\mibs\SMIv2\OSPF-MIB-ospf-05.txt
⊟	Failed	OSPF-TRAP-MIB-ospf-05.txt	25	D:\snmp\mibs\SMIv2\OSPF-TRAP-MIB-ospf-05.txt

Number	Error Code	Description
1	1100	Imported MIB module OSPF-MIB at line 18, column 24 unknown
2	1501	Unresolved object reference ospf
3	1501	Unresolved object reference ospfTrap
4	1500	Unresolved syntax reference ospfNbrAddressLessIndex at line 148, column 12
5	1500	Unresolved syntax reference ospfNbrState at line 150, column 12

```
[1500] Unresolved syntax reference ospfNbrAddressLessIndex at line 148, column 12
```

Figure 1: Compiler Log window with compilation errors displayed per MIB module.

To Correct a MIB File

Double click on the row corresponding to the MIB file you want to edit. The MIB file editor window will appear (see section "MIB File Editor" on page 55). Alternatively, you may double click on an error message to directly jump to that error location in the file.

If the error message selected includes location information about the error's line and column, then the editor's cursor will be placed at that location in the MIB file. When you have clicked on the file, the first error will be located. Otherwise, the first occurrence of the object name corresponding to a semantic error will be searched. The next occurrence of the object name may be found with the Find Again button .

After having fixed the error, the MIB file can be saved and compiled again by using the Import button .

If the compilation was successful, the editor window will be closed. Otherwise the cursor of the editor will be positioned on the new error.

Tip: You can print the expanded compiler log's content from the context menu.

5.3 Loading MIB Modules

AgenPro needs to load MIB modules from a MIB Repository into its memory to be able to display and use the contained information for code generation. For a better overview and performance, it is recommended to not load unneeded MIBs. Nevertheless, load all MIB modules used in any of your code generation projects, in order to avoid unexpected results during code generation.

AgenPro will load MIB modules configured in a code generation project even if those modules are not loaded in the GUI. However, you will not be able to see and review property definitions for MIB objects from those modules. This may cause unexpected code generation behavior.

To Load MIBs:

1. From the File menu, choose Open/Close MIB (or 📂 from the main tool bar). A shuffle dialog will appear. It contains two lists of MIB modules. The left list shows all MIB modules currently not loaded but available from the MIB repository. The right list shows the MIB modules currently loaded. MIB modules displayed in bold text are members of the current project. See “Project Wizard” on page 23 for details.
2. Select any MIB modules you want to load from the left list of available MIB modules. Click on the Add button to move the selected modules to the right list of MIBs to be loaded. If a MIB that is moved to the right list depends on another MIB module that is currently not loaded, then that MIB (and all MIBs it depends on) will be also moved to the right list. This ensures that AgenPro has always a consistent view on MIB data.
3. Select any MIB modules you want to unload (close) from the right list. Click on the Remove button to move the selected modules to the left list of available MIBs. Loaded MIB modules that depend on the removed (unloaded) MIB modules will also be unloaded and thus moved to the left list.
4. Click on the OK button to execute the changes made. Depending on the number of MIB modules that need to be loaded, it may take a while until all modules are loaded and the MIB tree is refreshed.

5.4 Deleting MIB Modules

Deleting a MIB module from a MIB Repository cannot be undone. A MIB module can only be deleted together with those MIB modules that depend on it by importing any MIB objects from it.

1. From the File menu, choose Delete MIB (or 🗑 from the main tool bar). A shuffle dialog will appear. It contains two lists of MIB modules. The left list shows all MIB modules available from the current

MIB repository. The right list shows the MIB modules that are to be deleted.

2. Select any MIB modules you want to delete from the left list of available MIB modules. Click on the Add button to move the selected modules to the right list of MIB modules that should be deleted. Any MIB modules that depend on a MIB that is moved to the right list will be moved to the right list too. This ensures that AgenPro has always a consistent view on MIB data.
3. Select any MIB modules you want to preserve from deletion in the right list. Click on the Remove button to move the selected modules to the left list of available MIBs. Any MIB modules that preserved MIB module depends on will also preserved from deletion.
4. Click on the OK button to execute the changes made.
5. Confirm the deletion of the displayed number of MIB modules by choosing the Yes option.

6 Projects

An AgenPro code generation project contains a set of user defined properties, a job list, and a list of MIB modules for which program code should be generated. AgenPro automatically saves the settings of the current project in its configuration file located in your home directory.

Using project files has the following advantages:

- More than one agent project or setup can be used at a time.
- Support for command line code generation which can be integrated into any build process.
- Easy creation of backups, also useful to test agent setup variations.
- Project files may be created and edited externally.

6.1 Accessing the Project Wizard

To edit the job list and code generation MIB module set, use the Project Wizard:

1. Choose Edit from the Project menu or from the tool bar.
2. Provide the information requested by the Project Wizard steps. See “Project Wizard” on page 23 for details.
3. Click Finish to save your settings.

Note: This will not run your code generation jobs. To run them, choose Generate *from the* Project *menu or* ▶ *from the toolbar.*

6.2 Managing Projects

To create a new project:

1. Choose New from the Project menu.
2. Provide the information requested by the Project Wizard steps.
3. Click Finish to save your settings.

Note: This will not run your code generation jobs. To run them, choose Generate *from the* Project *menu or* ▶ *from the toolbar.*

To save a project to a project file:

1. Choose Save As from the Project menu.
2. Select or enter a file in the opened file chooser.
3. Choose Save to save the project file.

To open a project:

Note: All properties set for the current project will be discarded and replaced by the properties set by the opened project.

1. Choose Open 📂 from the Project menu.
2. Select a previously saved project file with the opened file chooser.
3. Choose Open to load it into AgenPro.

6.3 Properties

The standard code generation templates that come with AgenPro are using several properties to customize the code generation process. Each property consists of a *key* and a *value*.
In many cases, the value part is used only for determining whether a property is defined or not (comparable to the `#define` and `#ifdef` macros in C++). From a Velocity code generation template you may call

```
$agenUtils.isDefined(String oid, String key)
```

to check whether a property is set to `yes`, `true`, or an empty string. In that case, `true` the method returns true, `false` otherwise. To get the assigned value of a property, you may call

```
$agenUtils.getProperty(String oid, String key)
```

The AgenPro API JavaDoc is located in the doc folder of the installation directory and at http://www.agentpp.com/agenpro/doc/index.html.

For more information on these methods see the *AgenPro API documentation*.

Properties are associated with MIB nodes. The properties associated with a MIB object are also inherited by all children of that MIB node. Inherited properties cannot be removed from a MIB object, but they can be redefined. Nodes with associated properties have a green background in the MIB tree as shown by Figure 2.

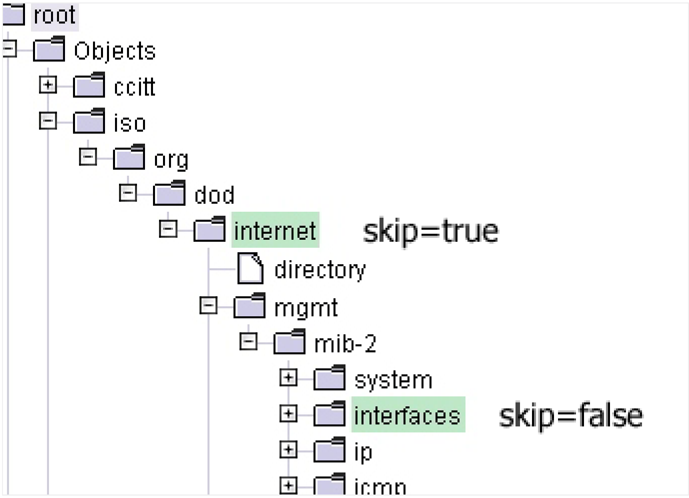


Figure 2: AgenPro MIB tree with properties set.

The following example illustrates how property assignment works:

The property `skip`, if defined, instructs AGENT++ and SNMP4J-Agent code generation templates to not generate any code for those objects that `skip` property is defined for. Thus, in the above example, no code will be generated for objects under the internet node. However, the `skip` property is redefined with value `false` for the interfaces node (by actually adding it to the interface node again). This redefinition causes AgenPro to generate code for the objects in that subtree although their generation had been suppressed for a parent node.
You can define and use any property you like for your own code generation templates or for customized versions of the code generation templates included in AgenPro. For the AGENT++ code generation templates there are some standard properties defined, that can be imported into AgenPro from the file

```
<installation_directory>/templates/agent++v3.5/
agent++v3_5.props
```

In order to customize and fine tune the code generation process, AgenPro provides means to define arbitrary properties for each MIB object in a project. A property (also called attribute) consists of a key and a value string. A property can be assigned to any MIB object with an OID. Once assigned to a MIB object node, the property is also assigned to all its children. The following sections provide additional information about properties and their usage:

- Why using properties?
- Working with properties (Properties Tab)
- Properties used by AGENT++ code generation templates
- Properties used by SNMP4J-Agent code generation templates
- Reading properties from an existing source file

6.3.1 Properties Tab

With the Properties tab, as its name suggests, properties can be associated with MIB nodes. The properties tab consists of two areas. On the right, a table that lists the properties defined for the selected MIB node including any inherited properties. The first column of the table contains the OID (object name) of the MIB node that originally defined the property. The table is always sorted by the OID column first. The second column contains the keys/identifiers of the properties and the third column contains the associated values.
To toggle the value of simple properties that can be enabled or disabled only, open the context menu on the row of the property you want to change and chose either Enable or Disable.

To view the properties of a MIB node:

1. Select the Properties tab.
2. Select the MIB node whose properties you want to view (edit) in the MIB tree. If there are any properties associated with a node, then this node will have a green background.

To add a property to a MIB node:

1. Follow the steps above to view the properties of the MIB node.
2. Choose Add from the Properties panel.
3. A new row will be appended to the table. The key of the new property is set to `<name>` and its value is set to `<value>`.
4. Modify key and value.

To remove a property from a MIB node:

1. Follow the steps above to view the properties of the MIB node.

2. Select the property you want to remove in the property table. You can only select properties that are originally defined for the select MIB node. Inherited properties will not be affected.

3. Choose Remove from the Properties panel.

To remove all properties from a MIB node:

1. Follow the steps above to view the properties of the MIB node.

2. Choose Remove All from the Properties panel.

To import properties from a file:

1. Follow the steps above to view the properties of the MIB node.

2. Choose Import from the Properties panel.

3. Select a plain text file that follows the Java properties file format. That is, each line is formatted as `<key>=<value>`.

4. Choose Open to assign the properties to the current MIB node.

To export properties to a file:

1. Select a MIB node whose properties you want to export.

2. Select the Properties tab.

3. Choose Export from the Properties panel. A file dialog box will be opened.

4. Enter a file name and choose Save to save the properties.

5. The properties will be saved to a plain text file where one property will be stored per line as `<key>=<value>`.

To read properties from an existing source file:

1. Select a MIB node for which you want to read the properties (for example, the root node).

2. Select the Properties tab.

3. Choose From Source from the Properties panel. A file dialog box will be opened.

4. Select an implementation file containing source code generated by AgenPro.

5. The properties that can be identified from the source file will be added to the properties of the currently selected node.

6.3.2 AGENT++ Code Generation Properties

AgenPro provides sets of code generation templates to generate program code for AGENT++ and AgentX++. Please use the most recent version of the templates that can be used with you version of AGENT++. The version of the templates indicates the earliest AGENT++ release with which those templates can be used. Examples:

- If your AGENT++ version is < 3.5.10 then you will have to use the AGENT++v3.5 templates.
- If your AGENT++ version is 3.5.10 or later then you should use the AGENT++v.3.5.10 templates.
- For new projects you should use the `agent++v3.5.23.prj` project from the projects directory.

The following properties control the code generation for AGENT++. To enable a property, add it to the properties table of a MIB node and assign a value `true` or `yes` to it. Then the property will be activated for that MIB node and all its children (unless it has been redefined for a child).

PROPERTY KEY	DESCRIPTION
agentX	Enables code generation for AgentX++ subagents. This option must not be defined for AgentX++ master agents.
agentXSharedTables	Tables are generated as `AgentXSharedTable` subclasses that support index allocation for shared tables. This option must not be used when the agentX option is not set!
lightTables	With this attribute set, read-only columnar objects are not generated as classes but instantiated directly as instances of `MibLeaf`. This option reduces code size.
tableAsComplexEntry	With this attribute set, tables are generated as subclasses of `MibComplexEntry` rather than as subclass of MibTable. Use this option for large tables with database or proxy interface.

Table 1: Code generation properties for AGENT++ templates.

PROPERTY KEY	DESCRIPTION
simulation	Generates a simulation agent where all objects may be modified when the agent is in configuration mode (see also the AGENTPP-SIMULATION-MIB), regardless whether they are read-only or not. This allows easy testing of your management software.
withModuleName	Generates object names with the MIB module name as prefix. This option can be used to avoid name clashes when two (or more) MIB modules use the same name for different objects.
leafSuperClass	Specifies the class name for leaf objects' super class. The default is '`MibLeaf`'. This property may be defined to derive all leaf objects from a customized subclass implementation of `MibLeaf`.
tableSuperClass	Specifies the class name for table objects' super class. The default is 'MibTable'. This property may be defined to derive all table objects from a customized subclass implementation of `MibTable`.
sharedTableSuperClass	Specifies the class name for AgentX shared table objects' super class. The default is 'AgentXSharedTable'. This property may be defined to derive all table objects from a customized subclass implementation of `AgentXSharedTable`.
complexEntrySuperClass	Specifies the class name for complex table objects' super class. The default is 'MibComplexEntry'. This property may be defined to derive all table objects of complex type from a customized subclass implementation of `MibComplexEntry`. See also “tableAsComplexEntry”.
notificationSuperClass	Specifies the class name for notification (trap) objects' super class. The default is 'NotificationType'. This property may be defined to derive all table objects from a customized subclass implementation of NotificationType.
skip	Set this property to "yes" to exclude all objects in this subtree from code generation.
useStaticConst	Set this property to "yes" use "`static const`" constants instead of `#define` for module wide definitions.

Table 1: Code generation properties for AGENT++ templates.

6.3.3 SNMP4J-Agent Code Generation Properties

The following properties control the code generation for SNMP4J-Agent. To enable a property, add it to the properties table of a MIB node and assign a value `true` or `yes` to it. Then the property will be activated for that MIB node and all its children (unless it has been redefined for a child).

PROPERTY KEY	DESCRIPTION
columnSuperClass	Specifies the class name for columnar objects' super class. The default is '`MOColumn`' for read-only columns and '`MOMutableColumn`' for all other columns. This property may be defined to derive all columnar objects from a customized subclass implementation of `MOColumn` or `MOMutableColumn`.
constructorAccess	If not set then a `public` constructor is generated.Valid other values include `protected`, `private`, and an empty string.
cleanup	If set to `yes` the `ManagedObject.cleanup(..)` method is overwritten for `MOScalar` subclasses.
commit	If set to `yes` the `ManagedObject.commit(..)` method is overwritten for `MOScalar` subclasses.
constantAccess	If not set then `public` OID constants are generated. Valid other values include `protected`, `private`, and an empty string.
contexts	A pipe (‚|‘) separated list of contexts that are supported for object instances in this sub-tree. See also “Simulation Agent” on page 41.
factoryColumn	Read-only columns or columns for which this property is set to `yes` will get created through calling the appropriate method of the `MOFactory` instance associated with this MIB module.
factoryScalars	Read-only scalars or scalars for which this property is set to `yes` will get created through calling the appropriate method of the `MOFactory` instance associated with this MIB module.
get	If set to `yes` the `ManagedObject.get(..)` method is overwritten for `MOScalar` subclasses.
import[.<1-n>]	The import property (properties) define additional imports needed for the module. If more than one import statement is needed the properties can be enumerated by appending a dot and a one based index value.
next	If set to `yes` the `ManagedObject.next(..)` method is overwritten for `MOScalar` subclasses.

Table 2: Code generation properties for SNMP4J-Agent.

Property Key	Description
noIndexValidator	Set this property to `yes` to disable generation of index validator anonymous classes. These classes can be used to implement custom index validation, for example, if such a validation cannot fully specified through SMI.
noRowFactory	Set this property to `yes` to disable generation of a custom `MOTableRowFactory` class. Otherwise an instance of the `DefaultMOMutableRow2PCFactory` class will be used to create rows. Using a custom row factory provides public access (getter) methods for the column values named by the columnar object names.
noTableGetter	Set this property to `yes` to disable generation of a public table getter method that makes a table accessible outside the MIB module class.
noValueValidator	Set this property to `yes` to disable generation of value validation anonymous classes for scalar and columnar objects.
moFactory	Specifies a non-default `MOFactory` to be used for the MIB module.
object.<trapName>	Maps an object instance to a trap (name). Because notification and trap type definitions specify which object types are included in a trap, but do not define which instances of that object types, this mapping is needed to be able to generate a (simulated) notification. The instance is identified by its row index value as defined by the corresponding `rows` property: `object.ifDownIf1 1` where `ifDownIf` is a trap name as defined by a traps property and `1` is the `ifIndex` index value. See “Simulation Properties” on page 44 for a complete example. If the value of this property is „`?`“ or is not specified at all, AgenPro will ask for its value, when this trap is simulated.
package	Set this property to the package name of the generated code.
reference	The value of this property is used when the `useReference` property is set to `yes` and there is no REFERENCE clause for the object type being generated.

Table 2: Code generation properties for SNMP4J-Agent.

PROPERTY KEY	DESCRIPTION
rows	The rows property defines the rows that are create for the table(s) in the node‘s sub-tree. The value of the rows property is a pipe symbol (‚|‘) separated list of row indexes. A row index value consists of sub-index values which are separated by semicolons (‚;‘). A sub-index value itself can be provided either as dotted numeric string (1.3.6.1.4.1) or as string representation of the sub-index value (snmpAdminString). To specify three indexes for the vacmAccessTable, the value of its rows property could be: v3group;;3;authPriv|v1;v1;1;noAuthNoPriv|v2c;;2;1
skip	Set this property to yes to exclude all objects in this subtree from the code generation.
staticMOFactory	Set this property to yes to use a static MOFactory. Default is no.
trapDescr.<trapName>	The trapDescr property defines a textual description for a particular notification template. This description is displayed by AgenPro when asking for missing notification payload references while firing this trap. In addition, the description is used to extend the JavaDoc of this notification generation method.
traps	The traps property defines the names of the notification templates used within the simulation agent and for code generation for a particular NOTIFICATION-TYPE. The names are separated by a pipe symbol („|“). To define two trap templates for the linkDown NOTIFICATION-TYPE, add the following property/value pair on the linkDown node within the IF-MIB: traps linkDownIf_1|linkDownIf_256 The content of a trap template is then defined by the object property for each OBJECT-TYPE included in the NOTIFICATION-TYPE‘s OBJECTS clause. Because the trapName is also used as method name, it should only contain letters and digits.
useReference	Set this property to yes to let AgenPro generate a link between the OID of the managed object type and the string value of its REFERENCE clause (or the value of the reference property) using a LinkedMOFactory.

Table 2: Code generation properties for SNMP4J-Agent.

PROPERTY KEY	DESCRIPTION

value[.rowIndex]	AgenPro uses the ‚value‘ property to generate code that initializes the values of columnar and scalar objects. In addition, this property is used to set initial values for the built-in simulation agent.
	Like other properties, this property affects (unless overwritten in a sub-node) all OBJECT-TYPE instances in the sub-tree of this node.
	This property‘s value must match with the SYNTAX clause of the instances. For example, an string value ‚`Text`‘ cannot be used as value for an OBJECT-TYPE with SYNTAX clause ‚`Integer32`‘.
	For columnar objects you need also to define a “rows” property to create row instances and thus the instances for the columnar objects. For each row index, you then have to define the appropriate column value by:
	`value.<rowIndex>`
	where `<rowIndex>` is the complete index value as defined in the corresponding `rows` property, for example:
	`value.v3group;;3;authPriv`

volatile	If this property is set to `yes` for a subtree, all objects generated for that subtree will be volatile and thus excluded from persistent storage and `ManagedObject` serialization. Otherwise the generated objects support `ManagedObjectSerializable` serialization.

Table 2: Code generation properties for SNMP4J-Agent.

6.3.4 The Code Generated for SNMP4J-Agent

A description of the code generated for SNMP4J-Agent and how to use it is explained by the `SNMP4J-Agent-Instrumentation-HowTo.pdf` file in the installation directory of AgenPro.

6.4 Project Wizard

With the Project Wizard you specify the Velocity templates that should be used for the code generation and the MIB modules that should be accessible to those templates.

The wizard is divided into three steps which are described by the following sections.

6.4.1 Job Configuration

Tip: In order to be able to merge existing code, make sure that the input directories (and file name templates respectively) are provided and point to (copies of) the existing source code.

The Job Configuration specifies the generation jobs to be processed for this project. If the "Do not merge with existing code" option is selected, any given input directory (see below) will be ignored and existing code will not be preserved.

Environment variables of the operating system can be used within the path strings too. Environment variables may be included in a path string by using the following format: `${<VARIABLE_NAME>}`, whereas `<VARIABLE_NAME>` must contain letters, digits and the underscore ('_') character only.

With the Root Directories panel, you can specify root directories for relative template paths and input/ouput directories. You may leave these two root directories empty, if you intend to use only absolute paths in the Job specification below.

Each job consists of the following attributes:

- **Job Type**
  The job type specifies whether this job is executed
  - only once per code generation,
  - once per MIB module in the module set defined in steps two and three respectively, or
  - once per context generated by a specified selection template or whether it is executed for each MIB module.
- **File Name Generation Template**
  The file name generation template specifies a Velocity template that generates a file name. This file name is then appended to the input and output directories respectively to determine the file name that is subject to the code generation actually done by running the *File Generation* template.
- **File Generation**
  The File Generation Velocity template is the template that actually is responsible for generating the program code. It generates the content of the file(s) whose name(s) has been determined by the File *Name Generation Template* of this job.
- **Selection Template**
  The optional selection template will be executed when the job's type is "By Selection". A selection template specified for any other job type will be ignored. A selection template has to produce an output text of the following format:

```
<context1>:
[<Module1>=]<object1>[,[<ModuleN>=]<objectN>..];
<...>
```

where ModuleN is the module name of the MIB specification defining the MIB object with name objectNameN. There must be generated at least one context per selection template which has to contain at least one object name. Specifying a MIB module name for an object name is optional but recommended since it prevents incorrect mappings when a project's MIB modules do not have unique object names.
For each context generated by a selection template, an output file will be generated and the code generation job will be executed with the contexts described in the section “Customizing Code Generation” on page 32 for the job type 3. The contexts *scalars*, *tables*, *traps*, etc. will be filled with the selected objects for the respective context name as generated by the selection template.

- **Input Directory (Optional)**
The input directory can be the same as the output directory or may be left empty. If the input directory is set to an valid directory, files in that directory will be parsed for AgenPro tags of the following form:

```
//--AgentGen BEGIN=<class>[::<method>]
<any program code that is protected during a (re)gen-
eration>
//--AgentGen END
```

where <class> denotes the class the code belongs to and the optional parameter <method> denotes the method the enclosed program code. In addition to this, the standard templates for AGENT++ are using some special "class" names, for example _INCLUDE and _END to identify code at the beginning and the end of a source file.

The program code protected by the above tags is then available to the generation template when it generates new code from a probably revised MIB module definition. Thus, the protected code may be preserved when an agent is regenerated. This is standard behavior with the AGENT++ and SNMP4J-Agent templates.

- **Output Directory**
The output directory specifies the directory where the generated files should be written to. If the input directory is the same as the output directory, existing files will be overwritten without further notice. So backup your agent sources before you run AgenPro.

6.4.2 AGENT-CAPABILITIES Selection

Note: Only leafs in the displayed tree of available AGENT-CAPABILITIES statements can be selected, because folders (other than the root node) represent the MIB modules defining the AGENT-CAPABILITIES under them.

If at least one of the loaded MIB modules contains an AGENT-CAPABILITIES statement, you may select the „Use AGENT-CAPABILITIES Statement“ check box. Step two will then replace step three of the wizard.

Instead of selecting the MIB modules with step three, using an AGENT-CAPABILITIES statement has the advantage, that you can specify the objects to be generated on the object level rather than on the module level. In addition, AGENT-CAPABILITIES can provide in depth documentation of an agent implementation.

6.4.3 MIB Module Selection

If you have not specified an AGENT-CAPABILITIES statement in step two, step three allows you to select a subset of the currently loaded MIB modules to be used for the code generation or to use all MIB modules available from the MIB repository at the time of code generation. If you want to select specific MIB modules deselect the checkbox at the top of the dialog and follow the steps below:

- To generate code for a MIB module, that module has to be added to the table on the right by selecting it and using the Add button.
- To disable code generation for a MIB module, remove it from the right table by selecting it and then using the Remove button.

If a MIB module is selected for code generation, then it is a member of the current project.

MIB modules that are members of the current project are displayed with bold text in the open MIB modules list left to the MIB tree. By using the list‘s context menu, the membership of a particular MIB module in the current project can be changed.

7 Code Generation

The book *Understanding SNMP MIBs by David Perkins & Evan McGinnis* defines MIB specifications as follows:
SNMP Management Information Base (MIB) specifications are documents containing definitions of management information so that networked systems can be remotely monitored, configured, and controlled.
In other words, a MIB specification describes the interface between device and management software. A MIB specification is typically written before the corresponding SNMP agent and management systems are implemented. Thus, generating code from MIB specifications provides the following benefits:

- Code generation ensures that a MIB implementation strictly follows the corresponding MIB specification which improves the quality of the implementation.
- Code generation cuts months from your implementation schedule by automatically generating the SNMP agent. You are able to spend more time on the agent/manager functionality and less time on implementing interfaces.

7.1 Code Generation Benefits with AgenPro & AGENT++

The following benefits are specific to the code generation for AGENT++:

- Automatically generates a SNMP agent from MIB modules.
- Transparently provides SNMPv1/v2c/v3 functionality.
- Ensures correct implementation of multiple indices, table ordering, row status and storage type textual conventions, variable constraints, access rules, and notifications.
- Generates notification objects which allow sending a trap with a single method call while checking that columnar objects are sent with fully qualified index information and all required variables are provided.
- MIB changes can be easily applied to an existing agent implementation. The agent is simply regenerated and recompiled. Existing instrumentation code is preserved.

- No need for special (non SMI) tags in the MIB files. Thus, any SMI conform MIB can be used directly for code generation. The configuration can be entirely stored in project files.
- AgenPro can be run from command line for integration with your build process.

7.2 Code Generation Benefits with AgenPro & SNMP4J-Agent

The following benefits are specific to the code generation for SNMP4J-Agent:

- Automatically generates a SNMP agent from MIB modules.
- Transparently provides SNMPv1/v2c/v3 functionality.
- Ensures correct implementation of multiple indices, table ordering, row status, storage type and other SNMPv2-TC textual conventions, variable constraints, access rules, and notifications.
- MIB changes can be easily applied to an existing agent implementation. The agent is simply regenerated and recompiled. Existing instrumentation code is preserved.
- No need for special (non SMI) tags in the MIB files. Thus, any SMI conform MIB can be used directly for code generation. The configuration can be entirely stored in project files.
- AgenPro provides full Maven integration through its Maven Plugin.

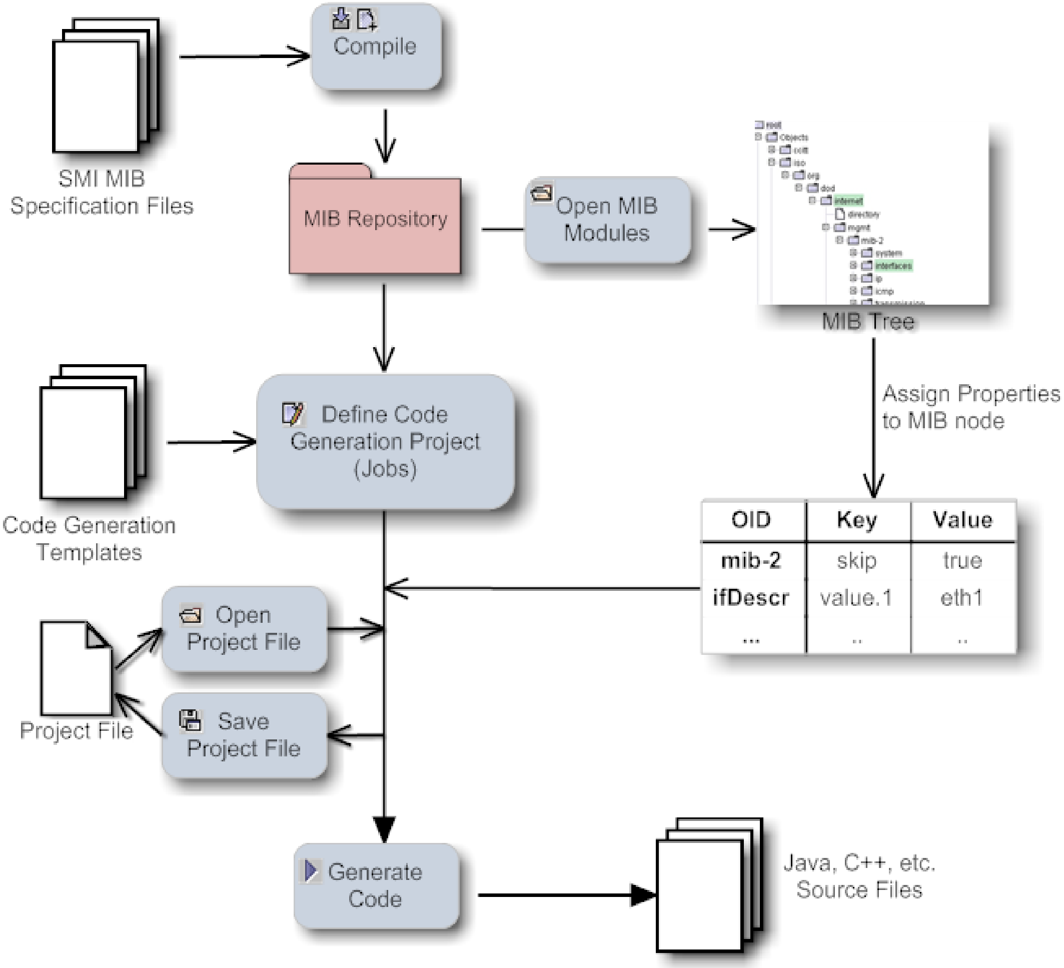
7.3 Code Generation Prerequisites

AgenPro's code generation process is illustrated by figure “Code Generation Process” on page 29. Before the code generation itself can be started by choosing Generate ▶ from the Project menu, there are a few tasks to be done beforehand (steps 1-5):

1. Compile your MIB files into AgenPro's MIB Repository.
2. Load the MIB modules you want to generate code for into the MIB tree.
3. Create a new project using the project wizard (see “Accessing the Project Wizard” on page 13). If you intend to generate code for AGENT++ or SNMP4J-Agent, then you might want to open the example projects located in the `projects` directory of the AgenPro installation. If you intend to generate code for any other SNMP agent or manager API, then you should read the section “Customizing Code

Generation” on page 32 first. This is also recommended if you would like to customize code generation for AGENT++ or SNMP4J-Agent.

4. Assign properties that parameterize the code generation process to MIB nodes.
5. Save the project.
6. Generate the program code.



OID	Key	Value
mib-2	skip	true
ifDescr	value.1	eth1
...	..	..

Figure 3: Code Generation Process

7.4 Running Code Generation Jobs

The code generation jobs specified for a project can be run by:

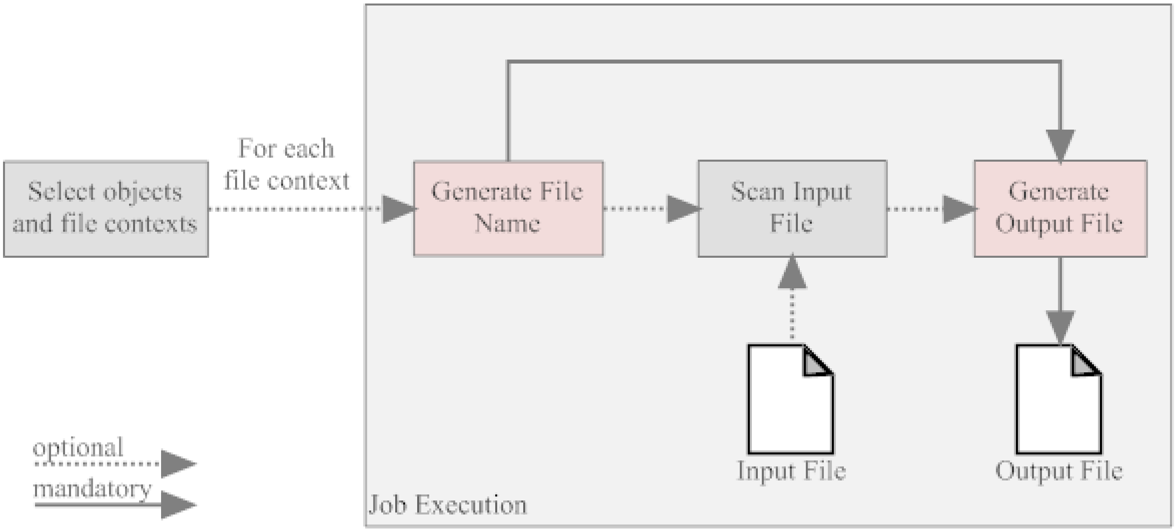


Figure 4: Code Generation Job Execution

From the command line:

1. Run AgenPro using the `agenpro.bat` (Windows) or `agenpro.sh` (UNIX) files in AgenPro's installation directory as follows:
   `agenpro.sh <project_file>`
   where `<project_file>` is the path of a previously saved project file.
2. To compile MIB modules into AgenPro's MIB repository from the command line use:

   ```
   agenpro.sh -a <MIBFileOrDirectory1> .. <MIBFil-
   eOrDirectoryN>
   ```

3. To use a specific MIB repository (other than that configured in AgenPro's configuration file) you can use the -r option on the command line:

   ```
   agenpro.sh -r <repositoryDirectory> [-l <license>
   <licenseKey>] [-c <AgenProConfigFile>] <project_file>
   ```

4. To directly generate code from a set of MIB files without compiling them into a MIB repository:

```
agenpro.sh -m <MIBFileOrDirectory1> [-m
<MIBFileOrDirectory2>] [-v] [-e <MIBModuleName1> [-e
<MIBModuleName2> ..]] [-l <license> <licenseKey>] [-c
<AgenProConfigFile>] <project_file>
```

The code will be generated for all MIB modules found in the input file(s) and which have not been excluded by the -e option - regardless of the settings in the project file.

5. The `-v` option replaces variable references of the form `${<name>}` in the value portions of the project properties file with the corresponding system variable with the name `<name>`.
6. To get help on the command line options, run:

```
agenpro.sh -?
```

Using the Graphical User Interface (GUI)

1. From the Project menu, choose Generate ▶.

7.5 How Jobs are Processed

Code generation jobs are processed following the below scheme. If specified, first the selection template is called where the contexts of the code generation (for example MIB module, MIB object, or any other context) are determined and the associated MIB objects are selected from the available MIB modules in the project. The output of the selection template has to be of the following form:

```
<context1>:
[<ModuleName1>.]<objectName1>[,[<ModuleName2>.]<obje
ctName2>..[,[<ModuleNameN>.]<objectNameN>]
;
<...>
```

where `ModuleNameN` is the module name of the MIB specification defining the MIB object with name `objectNameN`. There must be generated at least one context per selection template which has to contain at least one object name. Specifying a MIB module name for an object name is optional but recommended since it prevents incorrect mappings when a project's MIB modules do not have unique object names.

Using selection templates allows to individually select which objects to be generated in which files. By leaving the selection template column empty, one of the standard code generation options Once and For each module must be chosen. For most agents using the standard templates is sufficient and provides the best code generation performance. By using one of the standard code generation options, the code generation job starts with step 2 below.

The file name generation template is called. Depending on the job type, the context provided by the generation template is used to generate the file name for the file(s) generated by this job. The output of the template is appended to the job's input and output directory to form the paths for the input and output files respectively.
If an input directory has been specified for the job, then the corresponding input file will be scanned for user code protection tags of the form:

```
//--AgentGen BEGIN=<class>[::<method>]
<any program code that is protected during a
(re)generation>
//--AgentGen END
```

The program code found between the tags will then be stored in a `Hashtable` accessible through the context `existingCode`. The key for the `Hashtable` entries is built using the form: `<class>[::<method>]`.
Finally, the code generation template is called with the contexts described in the section “Customizing Code Generation” on page 32. The produced output is written to the output file(s) specified by the generated file name(s) and the given output directory. If output and input directory are the same, then the input file(s) will be overwritten by the generated output.

7.6 Customizing Code Generation

An outstanding feature of AgenPro is its flexibility. The complete code generation process can be customized which allows to write code generation from SNMP MIB specifications for virtually any programming language and SNMP agent or manager API.
AgenPro's code generation templates are based on VTL, the Velocity Template Language from Apache. Also VTL, in the first place, is designed for generating text or HTML output with dynamic content, its clear differentiation between model, view, and controller (MVC) makes it also a first choice for code generation.
The *control structures* provided by VTL are limited, but they also make it easy for users with little or no programming experience to write scripts based on VTL. Supported control statements are `#if..#else..#end` to conditionally execute statements and `#foreach..#end` to execute statements for each element of a given list. With the `#macro` statement parts of script that are frequently used can be combined into a function

that can be called in the template by `#<macro_name>`. Please refer to the VTL user guide or the VTL reference for a complete description of the VTL language.

An AgenPro template differs from any other Velocity Macro (VM) only by the *model* AgenPro provides for the template. The Model is accessed from a VM through *contexts*.

AgenPro offers three kinds of code generation jobs that provide different sets of contexts:

1. jobs that are run once per code generation
2. jobs that are run once per MIB module
3. jobs that are run once per each context generated by a selection template.

For most use cases, the first two job types are sufficient and provide the best performance. However if you need full flexibility, the third type would be the best choice. The first two types are used by the standard AGENT++ code generation projects whereas the third is used by the more advanced example project that selects the MIB objects to generate by a VTL template and generates each table and each scalar and notification object in its own set of source files. The SNMP4J-Agent default templates also use a context selection template.

The contexts supported by the three job types are listed in the below table and grouped as follows:

1. The first group of Velocity contexts provide utility functions and services that are independent from the chosen job generation type.
2. The second group of Velocity contexts provide access to objects available in the job's current file context. When running a job of type "by selection", then there might by more than one file context per job execution. In any other case, these Velocity contexts of this group will not change during the execution of the job.
3. The third and last group of Velocity contexts provide access to the MIB modules and module names that were selected in step three of the code generation wizard. When executing a job of type "by selection" the content of these Velocity contexts remain the same, regardless for which file contexts code generation templates are executed. To get the MIB modules selected by a selection template for a particular

file context, use the contextModules and contextModuleNames Velocity contexts.

CONTEXT	CLASS	JOB	DESCRIPTION
existingCode	Map	all	The existingCode context contains the protected code snippets collected from the input file, if an input directory had been specified. Otherwise, the Hashtable is empty. The key for the Hashtable entries is built using the form: <class>[::<method>].
agenUtils	AgenUtils	all	The agenUtils context provides various utility functions supporting the analysis of MIB content. For example, retrieving MIB objects by name and getting the effective syntax of an OBJECT TYPE. In addition, it provides access to the properties defined for a code generation project.
agenStringUtils	AgenString Utils	all	The agenStringUtils context provides string utility functions to search and replace strings using regular expressions.
stringUtils	StringUtil s	all	Additional string utilities provided by Velocity, for example reading a file into a String object.
agentCapabilities	IAgentCapa bilities	all	This context is only available if a AGENT-CAPABILITIES statement has been selected for this project in the corresponding project wizard. It provides access to the selected statement's SMI definition.
module	IModule	2,3*	The module context provides access to the target MIB module of this code generation job.
moduleName	String	2,3*	The name of the target MIB module (e.g. SNMPv2-MIB).
moduleNameNoHyphen	String	2,3*	The name of the target MIB with hyphen "-" replaced by underscores "_" (e.g. SNMPv2_MIB).

CONTEXT	CLASS	JOB	DESCRIPTION
`context`	`String`	2,3	The context name string generated by the selection template. This can be any string that must not contain ";", ",", and "=" characters. In case of jobs of type 2, the context name string equals the `moduleName`.
`contextNoHyphen`	`String`	2,3	The context name string generated by the selection template where hyphen characters ("-") are replaced by underscores ("_").
`contextModules`	`List`	2,3	The `contextModules` Velocity context provides access to all `IModule` instances selected for the current file context. In case of a job of type "once per module", the contents of this Velocity context is the same as `modules`.
`contextModuleNames`	`List`	2,3	The `contextModules` Velocity context provides access to all MIB module names selected for the current file context. In case of a job of type "once per module", the contents of this Velocity context is the same as `moduleNames`.
`contextObjects`	`List`	3	The `contextObjects` Velocity context provides access to all `IObject` instances selected for the current file context.
`scalars`	`List`	2,3	The `scalars` context provides access to all scalar OBJECT-TYPE definitions of `module`.
`tables`	`List`	2,3	The `tables` context provides access to all table objects defined in `module`. Table objects are those OBJECT-TYPE definitions that have an INDEX clause. The objects are ordered by their object identifier (OID)
`tablesByDependencies`	`List`	2,3	The `tablesByDependencies` context provides access to the same table objects as the context `tables`, but in different order. Tables with an INDEX clause „AUGMENTS“ will succeed tables which those tables depend on. Thr remaining tables are sorted by their object identifier (OID).

CONTEXT	CLASS	JOB	DESCRIPTION
`columns`	`Map`	2,3	The `columns` context provides access to all columnar OBJECT-TYPE definitions. The keys of the `Map` are the table objects in the `table` context `Vector`. By calling the `get` method of the columns `Map` all columnar objects of the corresponding table are returned as a `List` of `IObjectType` instances.
`indexes`	`Map`	2,3	The `indexes` context provides access to the index objects of table objects. The keys of the `Map` are the table objects in the `table` context `List`. By calling the `get` method of the columns `Map` all index objects of the corresponding table are returned as a `List` of `IObjectType` instances.
`notifications`	`List`	2,3	The `notifications` context provides access to all trap and notification objects defined in `module`. Notifications are TRAP-TYPE (SMIv1) or NOTIFICATION-TYPE (SMIv2) definitions.
`identities`	`List`	2,3	The `identities` context provides access to all OBJECT-IDENTITY definitions defined in `module`. This list is empty for SMIv1 MIB modules.
`fileName`	`String`	all†	The output (file name) generated by the file name generation template. Consequently, this context is not available in the file name generation template itself.
`fileNameNoDot`	`String`	all†	Same as above but dots "." in the filename are replaced by underscores "_".
`contexts`	`List`	3	All contexts generated by the selection template. For each of the context string in this vector there will be an output file generated. To get the index of the current context within a code generation template, the `$contexts.indexOf($context)` method can be used.

CONTEXT	CLASS	JOB	DESCRIPTION
`modules`	`List`	1,3	The `modules` context provides access to all MIB modules (`IModule` instances) for that program code should be generated.
`moduleNames`	`List`	1,3	The `moduleNames` context provides access to all MIB module names (`String` instances) of the target modules.

[*] Although these contexts are available in jobs of type 3, its use is not recommended because the MIB module referenced is the first MIB module that occurs in the generated contexts. Any other MIB modules used in the context will be hidden by these contexts. Instead of using such a context, the `$agenUtils.getModule(IObject)` method should be used to determine the MIB module for a MIB object.

[†] These contexts are not available in code generation templates.

7.7 AgenPro Maven Plugin

Although the AgenPro command line interface provides already means to integrate AgenPro into an automated build process, the AgenPro Maven Plugin further improves and facilitates this integration for Java.

The AgenPro Maven Plugin can be used to generate code for any language or format by using customized code generation templates. However the integration for Java is the primary use case.

7.7.1 Maven Plugin Installation

To install and use the AgenPro Maven Plugin you need

- Maven 2.2.1 or later
- AgenPro 3.0 or later.

Maven is available under the Apache License and is available for download from *http://maven.apache.org*.

Maven Repository Settings

Before you can start to install the AgenPro Maven Plugin you need to setup a local repository for your Maven installation and to specify a shortcut for the AgenPro Maven plugin prefix.
You can define your Maven settings in the `settings.xml` file in the `.m2` directory within your home directory. The `settings.xml` file should contain at least the following configuration:

```
<settings>
  <pluginGroups>
    <pluginGroup>com.oosnmp.agenpro.maven.plugins
    </pluginGroup>
  </pluginGroups>
```

```
    <localRepository>C:/maven/repo</localRepository>
</settings>
```

The pluginGroups element specifies a shortcut for the plugin and the localRepository element where local third-party libraries can be stored (cached).

SNMP4J and SNMP4J-Agent JARs

Download SNMP4J and SNMP4J-Agent JAR files from *https://server.oosnmp.net/dist/release/org/snmp4j* and unpack them in a local folder. Then run the following Maven commands from the dist and lib directories respectively:

```
mvn install:install-file -Dfile=SNMP4J.jar
 -DgroupId=org.snmp4j -DartifactId=snmp4j
 -Dversion=2.1 -Dpackaging=jar -DgeneratePom=true

mvn install:install-file -Dfile=SNMP4J-Agent.jar
 -DgroupId=org.snmp4j -DartifactId=snmp4j-agent
 -Dversion=2.0.6 -Dpackaging=jar -DgeneratePom=true
```

AgenPro JAR

Download the AgenPro.jar file and install it into your Maven repository:

```
mvn install:install-file -Dfile=agenpro3.jar
 -DgroupId=com.oosnmp.agenpro -DartifactId=agenpro
 -Dversion=3.1.3 -Dpackaging=jar -DgeneratePom=true
```

AgenPro Maven Plugin

Download the AgenPro Maven plugin and install it into your Maven repository:

Please always adjust the version *parameter to the actual version of the plugin you are installing.*

```
mvn install:install-file -Dfile=agenpro3-plugin.jar
 -DgroupId=com.oosnmp.agenpro.maven.plugins
 -DartifactId=agenpro-plugin -Dversion=3.1.3
 -Dpackaging=jar -DgeneratePom=true
```

7.7.2 Using the AgenPro Maven Plugin

After installation of the plugin and the JARs it depends on, you can setup your SNMP agent Maven project. As a starting point for your own project, use the template project provided in directory agenpro-mvn-task of the AgenPro‘s installation directory or download it from *http://www.agentpp.com/agenpro-mvn-task.zip*.

To get a description of the plugin options, execute the following Maven goal:

```
mvn help:describe -DartifactId=agenpro-plugin
 -DgroupId=com.oosnmp.agenpro.maven.plugins -Ddetail
```

The execution of the `help:describe` *goal should print a description of the* `agenpro` *goal and its parameters. If you get a Maven error instead, please check if the AgenPro plugin is properly installed according to the description in "Maven Plugin Installation" on page 37.*

Before you can use the AgenPro Maven plugin you will have to configure your license key first. Edit the `settings.xml` file in your `.m2` directory (see "Maven Repository Settings" on page 37) and add the `agenProLicenseKey` property as follows:

```
...
<profiles>
  <profile>
    <properties>
     <agenProLicenseKey>lic / key</agenProLicenseKey>
    </properties>
  </profile>
</profiles>
```

Now edit the `pom.xml` file in the `agenpro-mvn-task` directory to define

1. the directory where the plugin should search for MIB specification files
2. the MIB modules from that files you want to generate code for
3. the AgenPro project file that contains the code generation settings and properties.

By default, the project file for the latest SNMP4J-Agent release is used and the MIB specifications are read from the `src/main/smi` folder.

Creating and Compiling the SNMP Agent

To create a simulation agent based on Agent.java run Maven with

```
mvn clean install
```

in the `agenpro-mvn-task` directory. Maven will then generate the Java classes for the specified MIB modules in the `src/generated` folder, compile them together with the `Agent.java` class and build a JAR with (and without) all dependencies included in the `target` folder.

Running the SNMP Agent

To run the created agent simply execute

```
java -jar snmp4j-agenpro-agent-1.0-SNAPSHOT-jar-with-
dependencies.jar udp:0.0.0.0/4700
```

The above command will run the agent on all local IP addresses on port 4700 using the UDP protocol.

8 Simulation Agent

AgenPro provides a built-in simulation agent to facilitate and accelerate SNMP agent development. Without a complete code generation, compilation and agent startup cycle, the simulation agent provides an instant test environment for an early validation of a MIB design.

The simulation agent can simulate scalar and tabular values. In addition, notifications can be sent that use the simulated values. The simulation values and notifications are defined by properties (see "Properties" on page 14).

The simulation values are also used by the SNMP4J-Agent code generation templates to generate initialization code that sets initial values for scalars and creates rows for tables. The so generated code can be used

- to implement static MIB data or
- as example/template code by programmers to code instrumentation manually.

The simulation agent is also capable of sending notifications (traps). To avoid common errors and misunderstandings, the notification payload is not provided directly/interactively. Instead only the reference to the agent's data needs to be specified. This follows the SNMP recommendation *trap-directed polling*:

> *SNMP is built around a concept called trap-directed polling. Management applications are responsible to periodically poll SNMP agents to determine their status. In addition, SNMP agents can send traps to notify SNMP managers about events so that SNMP managers can adapt their polling strategy and basically react faster than normal polling would allow. [RFC 5345 §3.6]*

Applied to notification sending, the trap-directed polling concept requires that any value (information) sent by a notification is also retrievable by polling which is, in the end, sending GET-like SNMP requests. AgenPro supports this concept as it generates code to fire traps by providing references in place of values.

A reference in this context is the instance object identifier. The object type (class) identifier is defined by the OBJECTS clause of a NOTIFICATION-TYPE. The instance identifier is then the `.0`-suffix for scalars and the index OID for tabular objects.

When defining traps for a simulation agent, you first define the name of the trap. This name is different from the NOTIFICATION-TYPE object name as it identifies a reference set rather than a notification type. The second step is then defining the references (see “Simulation Properties” on page 44 for details).

The following sections describe how a simulation agent is *configured*, *started* and *used* at runtime.

8.1 Simulation Agent Configuration

The Agent panel has four settings defining the agent‘s start configuration which are described by Table 3 on page 43.

FIELD	DESCRIPTION

Parameters	Primarily, the parameter field defines the UDP and TCP addresses the simulation agent should listen on. The syntax is:
	`[(udp|tcp):]<address>/<port> ..`
	where `address` is any IPv4 or IPv6 IP address or host name, port is the UDP or TCP port. By default 161 is used for SNMP agents, which requires super user privileges on UNIX systems. At least one address must be specified otherwise the agent will be started with `udp:0.0.0.0/161`.
	In addition, boot counter and the persistent configuration storage file can be specified:
	`[-bc <bootCounterFile>]`
	`[-c <configFile>]`
	By default AgenPro uses `AgenProSim.bc` as boot counter file and `AgenProSim.cfg` as config file. Both files are stored in the user‘s home directory.

Initial Configuration	The optional initial configuration file is read at agent startup and should initialize the simulation independent MIB objects, like VACM, USM, and notification MIB. The file format is one of the following: 1. the SNMP4J properties format as defined by the *PropertyMOInput* class. 2. a XML file that complies to the XML schema *MIBConfig.xsd*.

Table 3: Simulation agent start configuration settings.

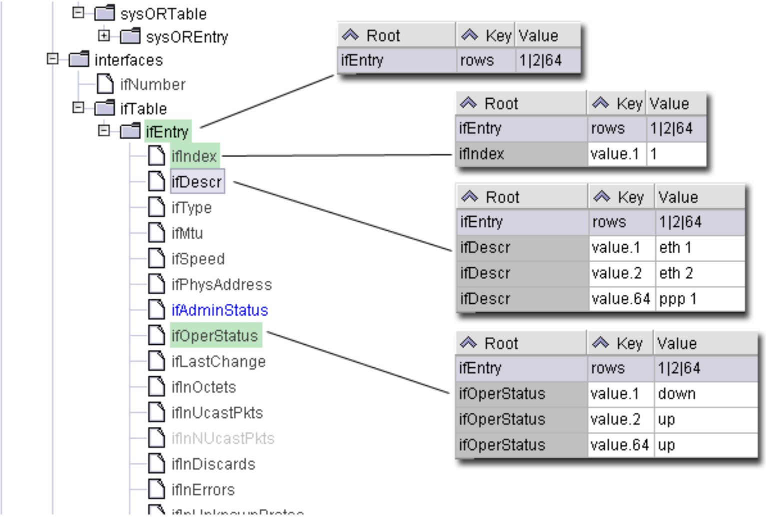
FIELD	DESCRIPTION
Simulation Data	With simulation data XML file the simulated MIB object values of the agent can be modified at runtime by modifying this file. The file format is a XML file that complies to the XML schema *MIBConfig.xsd*.
Simulation Refresh	Defines how often (in seconds) the file specified by Simulation Data is checked for updates. If a new file (updated file) is detected, its content is read and the MIB objects of the agent are updated accordingly.

Table 3: Simulation agent start configuration settings.

8.1.1 Simulation Properties

There are four properties that define the content and behavior of a simulation agent: `value`, `rows`, `traps`, and `object`. Scalar and tabular data is defined by `value` and `rows`. Traps (notifications) are defined by `traps` and `object`. For a description of those properties see table “Code generation properties for SNMP4J-Agent.” on page 20.

The figure “Example properties to simulate table data.” on page 45 shows a MIB tree detail from the IF-MIB (RFC 2863). The properties tabs‘ content for those tree nodes with properties defined are shown by overlay tables.



Root	Key	Value
ifEntry	rows	1|2|64

Root	Key	Value
ifEntry	rows	1|2|64
ifIndex	value.1	1

Root	Key	Value
ifEntry	rows	1|2|64
ifDescr	value.1	eth 1
ifDescr	value.2	eth 2
ifDescr	value.64	ppp 1

Root	Key	Value
ifEntry	rows	1|2|64
ifOperStatus	value.1	down
ifOperStatus	value.2	up
ifOperStatus	value.64	up

Figure 5: Example properties to simulate table data.

The first row shows the `rows` property defined for the `ifEntry` (1.3.6.1.2.1.2.2.1) object identifier. This property is then inherited by all columns under the table entry object. The inheritance has a welcome side effect: when entering the column instance value properties, you always see the rows property with the defined index value(s).

To ease entering the simulation data, row indexes and values are provided in their native format as described by Table 4 on page 46.

SMI SYNTAX	NATIVE FORMAT DESCRIPTION
OCTET STRING	Valid formats are: 1. If the object type has a DISPLAY-HINT clause, then the native form is that format. A common example is the `DisplayString` textual convention. As that format specification uses ASCII characters, you can specify such a value by simply using its textual representation as long as it does not contain control characters. 2. Otherwise, an OCTET STRING is specified as a hex-string where each byte is represented as a hexadecimal value and bytes are separated by colon or space. The string „`Hello 1 world`“ has to be formatted as „`04 0d 48 65 6c 6c 6f 20 31 20 77 6f 72 6c 64`“ for example.
OBJECT IDENTIFIER	Valid formats are: 1. the object name: `sysName` 2. a dotted OID string: `1.3.6.1.2.1.1.5` 3. an object name prefix and a dotted OID string suffix: `sysName.0`
Integer32, INTEGER, Counter32, Counter64, Gauge, Gauge32, Unsigned32	These values are provided as decimal numbers.
INTEGER { `enum`, ..}	Valid formats are: 1. the enumerated value‘s label (e.g., `enum`) or 2. the integer value associated with a label.
BITS { `label1`, ...}	Valid formats are: 1. a binary string where each bit of a byte is represented as 1 or 0 and bytes are separated by spaces, for example, if bits 1, 5, and 9 are set, the their native format is „`01000100 10000000`“ 2. a list of bit names enclosed in curly braces (`{,}`) and separated by commas, for example, provided that the bit labels are bit0 for first bit and bit7 for the eight, then the above BITS value can be written as „`{bit1,bit5,bit9}`“.

Table 4: The native representation formats for all SMI base syntaxes.

SMI SYNTAX	NATIVE FORMAT DESCRIPTION
IpAddress	An dotted string with for numbers: `192.168.1.1`
Opaque	a hex-string where each byte is represented as a hexadecimal value and bytes are separated by colon or space.
TimeTicks	An integer number represented in units of hundreds of a second.

Table 4: The native representation formats for all SMI base syntaxes.

The properties `value` and `rows` can be imported from a SNMP4J or MIB Explorer snapshot file by using the Import... button on the Agent tab. The import will add value and the corresponding rows properties for all variable bindings in the snapshot file. If a node of the MIB tree is selected, only those variable bindings are imported whose object identifiers belong to the selected sub-tree.

To remove all properties within a sub-tree, use Remove Properties *menu item in the MIB tree context menu of a MIB node.*

8.1.2 Simulation Data

As there are the properties `value` and `rows` to define simulation (and code generation) data information, you may ask why we need yet any other way to provide simulation data?

Properties are best to define initial and ad-hoc simulation data, but properties are not best suited to automate simulation data modification from third party applications or processes. For that purpose, XML is well supported and understood interchange format. The schema for AgenPro simulation data is defined by `MIBConfig.xsd` which is located in the `xsd` directory in the AgenPro accompanied files installation directory (see “Install Templates, Example Projects, and MIBs” on page 5 for information on how to install those files).

A simulation data XML file can be configured on the Agent tab as described by “Simulation agent start configuration settings.” on page 43. The file is read on agent startup after the configuration file (if present) and it is reread at the specified refresh rate. If the file is written by another application while AgenPro tries to read it, incomplete reads may occur which, in the worst case, prevent that the simulation data is being updated in this refresh cycle. To avoid such situations, it is recommended to replace the simulation data file by moving it on the file system instead of directly writing to it.

A XML simulation data file supports data initialization and delta updates:

- MIB object values not defined by a XML file will not be changed when the file is loaded.

- Scalar values defined in the XML define the corresponding value for the simulation agent.
- Tabular values (rows) can be *created*, *replaced* (includes creation), and *deleted*.

The following XML file is a small usage example:

```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="MIBConfig.xsd">

  <object name="nlmConfigGlobalAgeOut">
    <scalar>101</scalar>
  </object>

  <object name="nlmStatsLogTable">
    <row>
      <i>logName</i><i>1</i> <c>100</c><c>200</c>
    </row>
    <row>
      <i>logNameNext</i><i>2</i> <c>300</c><c>400</c>
    </row>
  </object>
</config>
```

This example can be loaded successfully only if the MIB module NOTIFICATION-LOG-MIB from RFC 3014 is loaded into AgenPro. Otherwise, AgenPro is not able to resolve the object names and it will not able to parse the values. As a simulation agent without corresponding MIB module(s) does not make any sense, this is not a real constraint, nevertheless important to bear in mind.
Instead of specifying object names, also object identifiers can be used:

```
<object oid="1.3.6.1.2.1.92.1.1.2">
 <scalar>101</scalar>
</object>
```

When using object names, it might by necessary to also specify the MIB module to avoid ambiguities:

```
<object name="sysContact" module="SNMPv2-MIB"
        oid="1.3.6.1.2.1.1.4">
 <scalar>System Administrator</scalar>
</object>
```

Tabular data is defined row by row. For each row, a value (or `null`) has to be specified for each column. The object identifier of a particular cell of a table is defined as

```
class-oid.sub-index-1[.sub-index-2[...]]
```

where `class-oid` is the object identifier of the column OBJECT-TYPE definition, and `sub-index-1` is the value of the first (virtual) object in the table‘s INDEX clause. The values of the index objects are encoded as OID. The simulation data XML provides two approaches to specify the index of a row:

1. Using the `index` attribute of the `row` element. The `index` attribute must specifies the complete index OID value (everything after „`class-oid.`“) as dotted string.
2. Using `<i>` elements (one `<i>` element per INDEX object). Each `<i>` element specifies a sub-index value in its native representation as described by “The native representation formats for all SMI base syntaxes.” on page 46. An `<i>` element must contain an OID only if, the corresponding sub-index object type has the base syntax OBJECT IDENTIFIER.

The following example shows how the simple INDEX clause of the `ifTable` can be specified:

The INDEX clause of the `ifEntry` *OBJECT-TYPE definition is simple because it contains only a single sub-index object, the* `ifIndex` *OBJECT-TYPE.*

```
<object name="ifTable" module="IF-MIB"
        oid="1.3.6.1.2.1.2.2.1">
 <row index="1000">
  <i>1000</i>
  <c>1000</c>
  <c>en1-0</c>
  <c>6</c>
  <c>1500</c>
  <c>100000000</c>
  <c>00:a0:f9:0c:4e:5b</c>
  <c>1</c>
  <c>1</c>
  <c>130156800</c>
  <c>1099008784</c>
  <c>1451652</c>
  <c>8240</c>
  <c>0</c>
  <c>0</c>
  <c>0</c>
  <c>1180600057</c>
  <c>1517714</c>
  <c>835</c>
  <c>0</c>
```

```
      <c>0</c>
      <c>0</c>
      <c>0.0</c>
    </row>
  </object>
```

The INDEX clause of the tcpConnTable (TCP-MIB RFC 4022) has four sub-index values with base type `IpAddress`, `INTEGER`, `IpAddress`, and `INTEGER`:

```
   <object name="tcpConnTable" module="TCP-MIB"
           oid="1.3.6.1.2.1.6.13.1">
    <row index="0.0.0.0.161.0.0.0.0.0">
     <i>0.0.0.0</i>
     <i>161</i>
     <i>0.0.0.0</i>
     <i>0</i>
     <c>0</c>
     <c>0.0.0.0</c>
     <c>0</c>
     <c>0.0.0.0</c>
     <c>0</c>
    </row>
  </object>
```

Exporting Simulation Data

When the simulation agent is running, its complete simulation data content can be exported as a XML file by using the Save Data... button on the Agent tab.

8.1.3 Simulation Agent Configuration with SNMP

Another way to configure the simulation agent is SNMP. Using SNMP has the advantage that a standard protocol is used that (to some extend) can be reused when developing the real agent generated by AgenPro.
The simulation agent has two modes: *operation* and *config*:

- The *operation* mode is the default mode after startup. In this mode, the agent behaves as any other SNMP agent.
- The *config* mode can be used to modify (create and update) simulated MIB objects regardless of their maximum access level defined by their MIB specifications. Although the maximum access level is ignored in this mode, security (VACM) still applies.

To change the agent‘s mode, set the MIB object `agentppSimMode` with OID `1.3.6.1.4.1.4976.2.1.1.0` to `1` (*operation*) or `2` (*config*).

Independently of the current mode, the objects agentppSimDeleteRow and agentppSimDeleteTableContents can be used to delete individual rows or all rows of a table.
To delete the row with index 100 of the ifTable, use the OID of an arbitrary object instance of that row, for example the ifDescr column:

```
SET 1.3.6.1.4.1.4976.2.1.2.0=1.3.6.1.2.1.2.2.1.2.100
```

To delete all rows of the ifTable table use the OID of the corresponding conceptual row object which is the ifEntry object:

```
SET 1.3.6.1.4.1.4976.2.1.3=1.3.6.1.2.1.2.2.1
```

8.2 Running a Simulation Agent

To run a simulation agent only a set of supported MIB modules need to be configured by setting up a code generation project using the “Accessing the Project Wizard” on page 13. Then simply press the Start button on the Agent tab and the simulation agent is started to listen on all local IP addresses on port 161, the default SNMP port for command responder applications.
If the simulation agent start fails, a popup is displayed with an error message and the root cause is displayed in the Log tab. In most cases, startup failures are caused by address or port conflicts or because on UNIX systems AgenPro does not have enough privileges to bind a port below 1024.
The running agent can be configured using standard SNMP means. The Table 5 on page 51 lists the MIB modules that are supported at minimum, that is, if no MIB modules for code generation have been configured for the current project. To initially access the agent you will need the security settings listed in the next section

To bind all local addresses, the special IP address 0.0.0.0 is used.

MIB Module Name	Description
SNMPv2-MIB	Contains general information about the agent.
SNMP-VIEW-BASED-ACM-MIB	Controls the view access control model (VACM) of the agent. The VACM controls which user/community has access to which contexts and MIB tree subsets.

Table 5: The MIB modules always supported by the AgenPro simulation agent.

MIB Module Name	Description
SNMP-USER-BASED-SM-MIB	Controls the (SNMPv3) users and their authentication and privacy settings. SNMPv1/v2c communities are mapped to users. See also SNMP-COMMUNITY-MIB.
SNMP-COMMUNITY-MIB	The SNMP-COMMUNITY-MIB maps communities (v1 or v2c) to SNMPv3 security names (users). With that mapping, the VACM can be applied to community based SNMP version too.
SNMP4J-LOG-MIB	Controls the log levels for the simulation agent.
SNMP4J-CONFIG-MIB	Manages the persistent storage of the agent‘s MIB data. By default, AgenPro uses the `AgenProSim.cfg` and the `AgenProSim.bc` to store its MIB data and boot counter respectively. Both files are stored in the user‘s home directory who is running AgenPro.
AGENTPP-SIMULATION-MIB	With the simulation MIB all simulated MIB objects can be modified through SNMP. The `agentppSimMode` MIB object can be used to put the agent from the default mode operation into the *config* mode. In config mode, all MIB objects, regardless of their MAX-ACCESS limit, can be modified with a SNMP SET operation (as long as VACM provides access to the object itself). For more details, see “Simulation Agent Configuration with SNMP” on page 50

Table 5: The MIB modules always supported by the AgenPro simulation agent.

Security Credentials for SNMP Access

To be able to access the agent through SNMP, use one of the protocol version and community or security name, protocol, and passphrases combinations list by Table 6 on page 53:

COMMUNITY/ SECURITY NAME	SNMP VERSION	AUTH. PROTOCOL	AUTH. PASSPHRASE	PRIVACY PROTOCOL	PRIVACY PASSPHRASE
public	v1	n/a	n/a	n/a	n/a
public	v2c	n/a	n/a	n/a	n/a
unsec	v3	-	-	-	-
MD5	v3	MD5	MD5AuthPP	-	-
MD5DES	v3	MD5	MD5DESAuthPP	DES	MD5DESPrivPP
SHA	v3	SHA	SHAAuthPP	-	-
SHADES	v3	SHA	SHADESAuthPP	DES	SHADESPrivPP
SHAAES128	v3	SHA	SHAAES128AuthPP	AES128	SHAAES128PrivPP

Table 6: Initial (default) security crendentials of the AgenPro simulation agent.

8.3 Sending a Simulated Notification

Before a simulated notification can be sent, the following prerequisites have to be met:

1. The `traps` property has to be specified for the NOTIFICATION-TYPE to sent with at least one name.
2. Optionally, for each columnar OBJECT-TYPE of the NOTIFICATION-TYPE‘s OBJECTS clause, define the row index reference by a `object` property.
3. Configure notification targets for the simulation agent by customizing the “Simulation Agent Configuration” on page 43. Alternatively, the target configuration can also be changed using SNMP and the SNMP-TARGET-MIB as well as the SNMP-NOTIFICATION-MIB.
4. Start the simulation agent using the Start button on the Agent tab.
5. Now the Send Notification list should contain all the trap template names that have been specified by one or more `traps` properties.

Select the template you want to generate a trap for and press the Send Notification button.

The referenced MIB object instances are subject to the View Access Control Model (VACM) of the agent. Thus, the agent checks if the object instances are accessible for notify.

6. If all columnar object references have been specified beforehand, the trap is generated by using the values of the referenced MIB object instances and is then sent to all configured targets. For any reference that are not provided, a message dialog asks for selecting it from configured row indexes for that columnar object. If no such row indexes are specified, the notification cannot be sent.

9 MIB File Editor

The MIB file editor has the usual capabilities of a text editor including printing, undo and redo. The status bar displays row and column position of cursor. The text pane below the tool bar displays error messages from the integrated MIB compiler.

9.1 Save, Compile, and Load a MIB File at Once

By choosing Import MIB from the editor's File menu the edited file is saved, compiled, and loaded into the MIB tree. If compilation fails, then the edited MIB module(s) will not be imported into AgenPro. Instead an error text will be displayed in the text area below the editor's tool bar. On successful compilation, the MIB module(s) will be stored in the MIB Repository and loaded. At the same time the editor window will be closed.

9.2 Search and Replace Function

A powerful way to make modifications to a MIB file is searching and replacing by regular expressions.
To search a MIB file by a regular expression, choose Find from the Edit menu. Enter the expression to search for in the opened dialog. The combo box will remember ten expressions used last.
To search and replace found matches, choose Replace from the Edit menu. Enter the search expression and the substitution expression and press OK. A matched region in the MIB file will be selected and a confirmation dialog will be shown. Each substitution can be confirmed individually or all substitutions can be confirmed at once.
The substitution string may contain variable interpolations referring to the saved parenthesized groups of the search pattern. A variable interpolation is denoted by `$1`, or `$2`, or `$3`, etc. It is easiest to explain what an interpolated variable does by giving an example:
Suppose you have the pattern `b\d+:` and you want to substitute the `b`'s for `a`'s and the colon for a dash in parts of your input matching the pattern. You can do this by changing the pattern to `b(\d+):` and using the substitution expression `a$1-`. When a substitution is made, the `$1` means *„Substitute whatever was matched by the first saved group of the matching pattern“*. An input of `b123:` after substitution would yield a result of `a123-`.

9.3 Regular Expression Syntax

For details of the theory and implementation of regular expressions you may consult the following Internet site http://py-howto.sourceforge.net/regex/regex.html

A regular expression (or RE) specifies a set of strings that matches it. Thus, a regular expression can be used to check whether an input string is matched by that expression.
Regular expressions can be concatenated to form new regular expressions; if A and B are both regular expressions, then AB is also a regular expression. If a string p matches A and another string q matches B, the string pq will match AB. Thus, complex expressions can easily be constructed from simpler primitive expressions like the ones described here.

A brief explanation of the format of regular expressions borrowed from the Python Library Reference follows.
Regular expressions can contain both special and ordinary characters. Most ordinary characters, like A, a, or 0, are the simplest regular expressions; they simply match themselves. You can concatenate ordinary characters, so last matches the string 'last'. (In the rest of this section, we will write RE's in this special style, usually without quotes, and strings to be matched 'in single quotes'.
Some characters, like "|" or "(", are special. Special characters either stand for classes of ordinary characters, or affect how the regular expressions around them are interpreted.

The special characters are shown by Table 7 on page 56:.

EXPRESSION	DESCRIPTION
.	(Dot.) In the default mode, this matches any character except a newline. If the DOTALL flag has been specified, this matches any character including a newline.
^	(Caret.) Matches the start of the string, and in MULTILINE mode also matches immediately after each newline.
$	Matches the end of the string and in MULTILINE mode also matches before a newline. foo matches both 'foo' and 'foobar', while the regular expression foo$ matches only 'foo'.
*	Causes the resulting RE to match 0 or more repetitions of the preceding RE, as many repetitions as are possible. ab* will match 'a', 'ab', or 'a' followed by any number of 'b' s.

Table 7: Regular expression syntax characters with special meaning.

EXPRESSION	DESCRIPTION
+	Causes the resulting RE to match 1 or more repetitions of the preceding RE. `ab+` will match '`a`' followed by any non-zero number of '`b`'s; it will not match just '`a`'.
?	Causes the resulting RE to match 0 or 1 repetitions of the preceding RE. ab? will match either '`a`' or '`ab`'.
`*?,+?,??`	The `*`, `+`, and `?` qualifiers are all *greedy*; they match as much text as possible. Sometimes this behavior is not desired; if the RE `<.*>` is matched against '`<H1>title</H1>`', it will match the entire string, and not just '`<H1>`'. Adding `?` after the qualifier makes it perform the match in *non-greedy* or *minimal* fashion; as *few* characters as possible will be matched. Using `.*?` in the previous expression will match only '`<H1>`'.
`{m,n}`	Causes the resulting RE to match from *m* to *n* repetitions of the preceding RE, attempting to match as many repetitions as possible. For example, `a{3,5}` will match from 3 to 5 `a` characters. Omitting *n* specifies an infinite upper bound; you can't omit *m*.
`{m,n}?`	Causes the resulting RE to match from *m* to *n* repetitions of the preceding RE, attempting to match as *few* repetitions as possible. This is the non-greedy version of the previous qualifier. For example, on the 6-character string '`aaaaaa`', `a{3,5}` will match 5 `a` characters, while `a{3,5}?` will only match 3 characters.
\	Either escapes special characters (permitting you to match characters like `*`, `?`, and so forth), or signals a special sequence; special sequences are discussed below.

Table 7: Regular expression syntax characters with special meaning.

EXPRESSION	DESCRIPTION
[]	Used to indicate a set of characters. Characters can be listed individually, or a range of characters can be indicated by giving two characters and separating them by a "-". Special characters are not active inside sets. For example, [akm$] will match any of the characters "a", "k", "m", or "$"; [a-z] will match any lowercase letter, and [a-zA-Z0-9] matches any letter or digit. Character classes such as \w or \S (defined below) are also acceptable inside a range. If you want to include a "]" or a "-" inside a set, precede it with a backslash, or place it as the first character. The pattern []] will match ']', for example. You can match the characters not within a range by *complementing* the set. This is indicated by including a "^" as the first character of the set; "^" elsewhere will simply match the "^" character. For example, [^5] will match any character except "5".
|	A|B, where A and B can be arbitrary REs, creates a regular expression that will match either A or B. This can be used inside groups (see below) as well. To match a literal "|", use \\|, or enclose it inside a character class, as in [|].
(...)	Matches whatever regular expression is inside the parentheses, and indicates the start and end of a group; the contents of a group can be retrieved after a match has been performed (for example in a substitution expression), and can be matched later in the string with the \number special sequence, described below. To match the literals "(" or "')", use \(or \), or enclose them inside a character class: [(] [)].

Table 7: Regular expression syntax characters with special meaning.

EXPRESSION	DESCRIPTION
`(?...)`	This is an extension notation (a `"?"` following a `"("` is not meaningful otherwise). The first character after the `"?"` determines what the meaning and further syntax of the construct is. Extensions usually do not create a new group; `(?P<name>>...)` is the only exception to this rule. Following are the currently supported extensions.
`(?imsx)`	(One or more letters from the set `"i"`, `"L"`, `"m"`, `"s"`, `"x"`.) The group matches the empty string; the letters set the corresponding flags for the entire regular expression: `i` - Do case-insensitive pattern matching. `m` - Treat string as multiple lines. That is, change `"^"` and `"$"` from matching the start or end of the string to matching the start or end of any line anywhere within the string. `s` - Treat string as single line. That is, change `"."` to match any character whatsoever, even a newline, which normally it would not match. The `/s` and `/m` modifiers both override the `$*` setting. That is, no matter what `$*` contains, `/s` without `/m` will force `"^"` to match only at the beginning of the string and `"$"` to match only at the end (or just before a newline at the end) of the string. Together, as `/ms`, they let the `"."` match any character whatsoever, while yet allowing `"^"` and `"$"` to match, respectively, just after and just before newlines within the string. Extend your pattern's legibility by permitting whitespace and comments.
`(?:...)`	A non-grouping version of regular parentheses. Matches whatever regular expression is inside the parentheses, but the substring matched by the group *cannot* be retrieved after performing a match or referenced later in the pattern.
`(?#...)`	A comment; the contents of the parentheses are simply ignored.

Table 7: Regular expression syntax characters with special meaning.

EXPRESSION	DESCRIPTION
`(?=...)`	Matches if `...` matches next, but doesn't consume any of the string. This is called a look-ahead assertion. For example, `Isaac(?=Asimov)` will match '`Isaac`' only if it's followed by '`Asimov`'.
`(?!...)`	Matches if `...` does not match next. This is a negative look-ahead assertion. For example, `Isaac (?!Asimov)` will match '`Isaac`' only if it's *not* followed by '`Asimov`'.

Table 7: Regular expression syntax characters with special meaning.

10 Logging

AgenPro provides a highly configurable and well known logging mechanism, called Log4J from Apache. By default logging is enabled. Logging can be disabled or enabled by using the Log panel of AgenPro's user interface (see below). Logged events are shown in the logging table of the Log panel. They can be exported to a text file using the Save As button.

10.1 Configuration

1. Select the Log tab from the tools panel.
2. Press the Properties button. The logging properties window will be displayed.
3. Enter the maximum number of log records to be held by AgenPro in the log table. Zero will disable logging.
4. Browse through the event tree and assign priorities other than `FATAL` to the events you want to monitor. Assigning `FATAL` to the root priority will disable logging for all subtrees in the event hierarchy that do not override that priority.
5. Press Save to save the settings. The logging properties will be restored when AgenPro is started for the next time.

11 Tools

11.1 Searching the MIB Tree

AgenPro's MIB Tree can be searched by *regular expressions*. A node whose properties or attributes matches the given regular expression will be selected. With the Find Again menu item or button you are then able to find the next node that matches the expression.

To Find a Node:

1. Choose Find from the Edit menu or press from the main tool bar. The search dialog will be displayed.
2. Enter the search expression in regular expression syntax.
3. Select whether case should ignored or not. If selected, this will insert "`(?i)`" at the beginning of the used search expression.
4. Select what type of attributes of a node you want to be matched against the search expression. Choosing All will match the whole SMI text of a MIB object node, including key words, or the properties rendered as "`key= value`" node against the given search expression.

To Find a Node Again:

Choose Find Again from the Edit menu or press from the main tool bar. The next node in depth first search order from the currently selected node will be searched, that matches the previously specified search expression and options.

11.2 Identifying Duplicate OIDs

It could be problematic and it is not desirable for the code generation if an object identifier (OID) is not unique within the set of generated MIB objects. To avoid such a situation, AgenPro can list the duplicate OIDs of the loaded MIB modules in a table. From the Tools menu, choose Duplicate OIDs to open this list.

11.3 Extract SMI Modules from RFC Documents

SMI MIB module definitions are embedded in IETF RFC documents which also includes page headers within the module text. This extraction tool can read a RFC file or a directory of RFC files to extract any embedded SMI modules and save them into new files.

To Extract SMI Modules from RFCs:

1. Choose Extract SMI from RFC from the Tools menu.
2. Choose a source file or a source directory.
3. Choose a target file if you have chosen a source file or choose a target directory if have chosen a source directory.
4. Press the Ok button to run the extraction. A progress dialog will open where you can also cancel the operation if more than one file is being processed.

If two directories are specified, then the target file name is build from the source file name by appending „.smi“. If such a file exists already, then „-<n>.smi“ is appended where <n> is counted up from 1 to 999 until such a file does not exists.

11.4 Exporting MIB Modules

MIBs can be exported from the current MIB repository to plain text and HTML files.

To Export MIBs:

1. Choose Export MIBs from the File menu.
2. Choose the file format for the exported MIB modules.
3. Select the MIBs to export from the list of available modules and press the Add button to add them to the list of modules to be exported.
4. Choose the destination directory.
5. Press OK to start the export operation. Each MIB module will be exported to a file, whose name will be the MIB modules name concatenated with one of the suffixes `.txt` or `.html`.

Note: Any files existing in the destination directory might be overwritten!

12 MIB Compiler Error Messages

ERROR #	MESSAGE	DESCRIPTION/SOLUTION
0000	File open error: `<X>`.	The file `<X>` could not be read, please check access rights.
0010	The length of identifier `<X>` exceeds 64 characters (RFC 2578 §3.1, §7.1.1, §7.1.4).	It is recommended to use only identifiers with a length of less than 32 characters for interoperability issues. Identifiers that exceed 64 characters in length must be avoided.
0050	Encountered lexical error at …	The encountered character is not allowed in a SMI MIB module.
1000	Syntax error: Encountered „`token1`" at row `r`, column `c`, expected one of the following: ...	The parser encountered a string it did not expect. Please look at the list of expected tokens carefully in order to determine the trouble cause. If the parser complains about a SMIv2 keyword like MAX-ACCESS, please check whether the first statement after the IMPORTS clause is a MODULE-IDENTITY definition. This is a requirement for a SMIv2 MIB module (RFC2578 §3).
1001	The DISPLAY-HINT clause value „`token1`" at row `r`, column `c` is invalid (RFC 2579 §3.1).	The DISPLAY-HINT clause does not correspond to any of the allowed formats for INTEGER or OCTET STRING base types.

Table 8: AgenPro SMI compiler error messages.

ERROR #	MESSAGE	DESCRIPTION/SOLUTION
1002	The UTC time value “token1” at row r, column c does not match the mandatory format `YYMMDDhhmmZ` or `YYYYMMDDhhmmZ` (RFC 2578 §2)	The UTC time value does not correspond to the format `YYMMDDhhmmZ` or `YYYYMMDDhhmmZ` where `YY` - last two digits of year (1900-1999 only) `YYYY` - last four digits of the year (any year) `MM` - month (01 through 12) `DD` - day of month (01 through 31) `hh` - hours (00 through 23) `mm` - minutes (00 through 59) `Z` - denotes GMT (the ASCII character Z)
1020	Identifier `<X>` is ambiguous (RFC 2578 §3.1).	The identifiers (descriptors) in a MIB module must be unique.
1050	The clause `<X>` is not allowed within this context.	There are several clauses in SMI that are optional, but if specified those clauses need to be consistent with other clauses in the object definition. Examples for such clauses are the ACCESS, MIN-ACCESS, and SYNTAX clauses in MODULE-COMPLIANCE constructs, which must not be present for variations of NOTIFICATION-TYPEs.
1100	Imported MIB module `<X>` unknown.	The MIB module `<X>` could not be found in the MIB repository and neither in the MIB modules being compiled. Make sure that the MIB module name is not misspelled (this is often the case for older SMIv1 MIBs).
1101	Imported MIB module `<X>` contains a circular import.	The MIB module `<X>` imports from a module that either imports itself from `<X>` or any other module in the import chain imports from a preceding module.
1102	MIB module `<X>` is imported more than once.	The ASN.1 rules about IMPORTS that SMI is based on require that an import source is defined not more than once in a module.

Table 8: AgenPro SMI compiler error messages.

ERROR #	MESSAGE	DESCRIPTION/SOLUTION
1110	<X> imported from MIB module <Y> must be imported from <Z> instead.	For historical reasons, SMI requires to import the MACRO definitions SMI is based on from some ASN.1 modules. For SMIv1 and SMIv2 it is defined which MACRO (construct) is imported from which ASN.1 module. Since those ASN.1 modules (e.g. SNMPv2-SMI) are not SMI themselves, the MACRO definitions have to be removed in order to be able to compile them.
1111	Missing import statement for <X> (RFC 2578 §3.2).	To reference an external object, the IMPORTS statement must be used to identify both the descriptor and the module in which the descriptor is defined, where the module is identified by its ASN.1 module name.
1112	Imported object <X> is not defined in MIB module <Y>.	Use the Edit>Search MIB Repository to search for the MIB module that defines <X>.
1113	Object <X> is imported twice from MIB module <Y>.	An object definition shall only be imported once from a MIB module.
1114	<X> cannot be imported (RFC 2578 §3.2).	Notification and trap type definitions as well as SEQUENCE constructs cannot be imported by other MIB modules.
1150	Wrong module order within file.	The MIB file that failed to compile contains more than one MIB module and the order of those MIB modules does not correspond with their import dependencies.
1200	The SYNTAX clause of the columnar OBJECT-TYPE definition <X> does not match with the SYNTAX clause of the corresponding SEQUENCE definition.	The object <X>'s syntax differs in a SEQUENCE definition from its OBJECT-TYPE definition.

Table 8: AgenPro SMI compiler error messages.

ERROR #	MESSAGE	DESCRIPTION/SOLUTION
1202	The OBJECT-TYPE <X> has inconsistent maximum access (RFC 2578 §7.3).	An object <X> has a MAX-ACCESS or ACCESS clause that does not match its context (RFC 2578 §7.3). For example, a columnar object must not have a MAX-ACCESS value of "read-write" if any other columnar object in the table has a MAX-ACCESS value of "read-create".
1210	The conditionally GROUP clause <X> must be absent from the corresponding MANDATORY-GROUPS clause (RFC 2580 §5.4.2).	A conditionally group cannot be mandatory at the same time!
1211	OBJECT variation <X> must be included in a GROUP or MANDATORY-GROUPs reference (RFC 2580 §5.4.2).	The object reference <X> must be part of any object group specified as conditionally or mandatory for this compliance module.
1212	Only 'not-implemented' is applicable for the ACCESS clause of the notification type variation <X> (RFC 2580 §6.5.2.3).	If the notification has to be implemented, then the ACCESS clause should be removed.
1220	The CREATION-REQUIRES clause of variation <X> must only be present for conceptual row definitions (RFC 2580 §6.5.2.4).	The CREATION-REQUIRES clause must not be present unless the object named in the correspondent VARIATION clause is a conceptual row, i.e., has a syntax which resolves to a SEQUENCE containing columnar objects.
1221	Only columnar object type definitions with ‚read-create' access may be present in the CREATION REQUIRES clause of variation <X> (RFC 2580 §6.5.2.4).	Other objects and columns cannot be created and thus they cannot participate in a row creation.
1500	Undefined syntax(es): <X>[,…]	The syntax (data type) <X> is not defined in the parsed MIB module and it is not imported from another MIB module. Use the Edit>Search MIB Repository function to search the MIB repository for object name <X> and add the corresponding IMPORT FROM clause for <X>.

Table 8: AgenPro SMI compiler error messages.

ERROR #	MESSAGE	DESCRIPTION/SOLUTION
1501	Undefined object(s): <X>[,…]	The object name <X> is not defined in the parsed MIB module and it is not imported from another MIB module. Use the Edit>Search MIB Repository function to search the MIB repository for object name <X> and add the corresponding IMPORT FROM clause for <X>.
1502	The object <X> must be defined or imported (RFC 2578 §3.2).	The object <X> is not defined in the parsed MIB module and it is not imported from another MIB module. Use the Edit>Search MIB Repository function to search the MIB repository for object name <X> and add the corresponding IMPORT FROM clause for <X>.
1600	The object definition <X> references a <Y> definition, expected a reference to an OBJECT-TYPE conceptual row definition instead.	The AUGMENTS clause, for example, requires that the referenced object definition is a conceptual table definition, i.e., has a syntax which resolves to a SEQUENCE containing columnar objects.
1601	The GROUP clause <X> references a <Y> definition, expected a reference to an OBJECT-GROUP or NOTIFICATION-GROUP instead (RFC 2580 §5.4.2).	The GROUP clause requires a reference to an object group definition.
1602	The object reference <X> points to a <Y> definition, expected a reference to an OBJECT-TYPE or NOTIFICATION-TYPE definition instead.	The VARIATION clause, for example, requires a reference to an OBJECT-TYPE or a NOTIFICATION-TYPE definition.
1700	Object reference(s) with wrong type: <X> (expected <Y> but found <Z>) [,…]	The referenced to object <X> must be of type <Y> but it is of type <Z>.

Table 8: AgenPro SMI compiler error messages.

ERROR #	MESSAGE	DESCRIPTION/SOLUTION
1800	The SEQUENCE clause of the table entry definition <X> does not match the order or number of objects registered for that table at entry <Y>.	The column references in the SEQUENCE definition of a table must be lexicographically ordered by their object-identifiers. The object name Y is the name of the first object reference in the SEQUENCE definition that does not match the order of columnar objects of that table.
1810	The OBJECT-TYPE <X> has an invalid index definition (RFC 2578 §7.7).	The OBJECT-TYPE <X> has an invalid INDEX clause, i.e., an empty clause.
1811	The OBJECT-TYPE <X> has invalid index definition because <Y> may be negative (RFC 2578 §7.7).	Index values have to be encoded as OID suffixes on the wire. Since OID sub-identifiers are 32-bit unsigned integer values, negative values cannot be encoded over the wire. See RFC 2578 §7.7 for more details.
1850	The OBJECT-TYPE <X> has invalid index definition, because <Y> is not a columnar object (RFC 2578 §7.7).	The OBJECT-TYPE <X> has an invalid INDEX clause, because <Y> does not refer to a columnar OBJECT-TYPE definition. An OBJECT-TYPE is columnar object, if it is part of a table definition. See RFC2578 §7.7 for more details.
1851	OBJECT-TYPE definition <X> is a scalar and therefore it must not have an INDEX clause (RFC 2578 §7.7).	Scalar objects have a fixed instance identifier (“index”) of ‘0’, thus an INDEX clause must not be specified.
2000	Duplicate object registration of <X> after <Y> for the object ID <Z> (RFC 2578 §3.6).	Once an object identifier has been registered* it must not be reregistered.
2010	Illegal object registration of <X> under <Y> for the object ID <Z>.	For example, it is not legal to register objects in the sub-tree of an OBJECT-TYPE registration.
3000	The default value of OBJECT-TYPE <X> is out of range (RFC 2578 §7.9).	The values specified in a DEFVAL clause have to be valid values for the corresponding data type syntax.
3001	The size of the default value of OBJECT-TYPE <X> is out of range (RFC 2578 §7.9).	The length of the specified octet string exceeds the SIZE constraints defined for the corresponding data type syntax.

Table 8: AgenPro SMI compiler error messages.

ERROR #	MESSAGE	DESCRIPTION/SOLUTION
3002	The format of the default value of OBJECT-TYPE <X> does not match its syntax (RFC 2578 §7.9).	The value <X> is not properly defined for the corresponding syntax.
3003	A DEFVAL clause is not allowed for OBJECT-TYPE <X> which has a base syntax of Counter (Counter32 or Counter64) (RFC 2578 §7.9).	
4000	The syntax definition of the object <X> is not a valid refinement of its base syntax (RFC 2578 §9).	A refinement must not extend the range of valid values for a data type.
4010	The range restriction is invalid because …	The lower bound (first value) of range restriction must be less or equal than the corresponding upper bound (second value). In addition, bounds for unsigned values cannot be negative.
4100	The TEXTUAL-CONVENTION definition <X> must not have a DISPLAY-HINT clause because its SYNTAX is OBJECT IDENTIFIER, IpAddress, Counter32, Counter64, or any enumerated syntax (BITS or INTEGER) (RFC 2579 §3.1)	Only textual conventions for INTEGER and OCTET STRING base types may have a DISPLAY-HINT clause.
4101	The DISPLAY-HINT clause value „token1" of the TEXTUAL-CONVENTION definition <X> is not compatible with the used SYNTAX (RFC 2579 §3.1)	The integer DISPLAY-HINT format must be used with the INTEGER base type only whereas the string DISPLAY-HINT format must be used with OCTET STRING base type only.
5000	The object definition <X> must be included in an OBJECT-GROUP or a NOTIFICATION-GROUP definition respectively (RFC 2580 §3.1 and §4.1).	This requirement ensures that compliance statements for a MIB module can be written.
5100	Object group <X> must not reference OBJECT-TYPE <Y> which has a MAX-ACCESS clause of not-accessible (RFC 2580 §3.1).	Only accessible objects and notifications may be included in object groups.

Table 8: AgenPro SMI compiler error messages.

*. An object registration is any object definition other than OBJECT-IDENTIFIER.