

MIB Designer 5.1

A Java SE Application
for Visual MIB Design and Editing
of SMIv1/v2 MIB Modules
and SPPI PIB Modules



Copyright © 2001-2023, Frank Fock. All rights reserved.

1	System Requirements	1
2	Installation	2
2.1	Using Native Installer	2
2.2	Other Platforms	2
2.3	Starting MIB Designer	3
2.4	Updates and Upgrades	3
2.5	Uninstall	4
3	What Is MIB Designer?	5
4	Setup	7
4.1	Selecting a MIB Repository	7
4.2	Compiling MIB Files	7
4.3	Deleting MIB Modules	10
5	Using MIB Designer	12
5.1	Creating a New MIB	13
5.1.1	New MIB Wizard	14
5.2	Editing a MIB	17
5.2.1	Import	17
5.2.2	Add	18
5.2.3	Copy	18
5.2.4	Cut	19
5.2.5	Paste	19
5.2.6	Edit	20
5.2.7	ASN.1 Comments	20
5.2.8	Moving Objects	21
5.2.9	Renumbering Objects	21
5.2.10	MIB Object Editing Dialogs	22
5.2.11	Object Identifier	24
5.2.12	Object Identity	25
5.2.13	Module Identity	26
5.2.14	Textual-Convention	28
5.2.15	Object Type	30
5.2.16	Table	32
5.2.17	Notification	34
5.2.18	Group	35
5.2.19	Module Compliance	36
5.2.20	Agent Capabilities	38
5.2.21	MIB-Tree Colors and Icons	40
5.3	Built-in Spell Checking	41
5.4	Finding MIB Objects	42

5.4.1	Search MIB Repository for Importing Objects	42
5.4.2	Search MIB Repository for References	43
5.4.3	Navigate Between MIB Objects	43
5.4.4	Refactor Object Names and Descriptions	43
5.5	MIB Validation	44
5.6	Saving and Exporting a MIB	44
5.6.1	Exporting MIBs to XML, HTML, XSD, PDF, and Text	45
5.7	Printing a MIB module	46
5.8	MIB File Editor	46
5.8.1	Checking a MIB File	46
5.8.2	Saving and Compiling a MIB File	46
5.8.3	Auto Syntax Completion	47
5.8.4	Printing with Syntax Highlighting	48
5.8.5	Search and Replace by Regular Expressions	48
6	MIB Design	50
7	Revision Control	60
8	MIB Comparison	62
8.1	Comparing Two MIB Modules	62
8.2	Clearing a Comparison	64
9	SMI Conversion	65
9.1	SMIv1 to SMIv2	65
9.1.1	Fully Automated	65
9.1.2	Manual Intervention or Review Needed	66
9.1.3	Not Supported	67
9.2	SMIv2 to SMIv1	68
9.2.1	Fully Automated	68
9.2.2	Not Supported	69
10	Correction	70
10.1	Index Range Correction	70
10.2	INTEGER Usage Correction	70
10.3	Case Correction	70
10.4	SMI Macro Import Correction	70
11	Tools	72
11.1	Extracting SMI from RFC Documents	72
11.2	Tool Configuration	72
11.3	Tool Execution	77
12	Preferences	78
12.1	General	78
12.1.1	MIB Compiler	78
12.1.2	Other Options	78

12.1.3	MIB Generation	79
12.2	Repository	80
12.3	View	80
12.3.1	Look & Feel	80
12.3.2	Other View Settings	81
12.4	Spell Checking	81
12.5	Defaults	81
12.6	Syntax Highlighting	82
12.7	Printing	82
12.8	Internet Proxy	82
13	Trouble-Shooting	84
14	Error Messages	87
15	Regular Expression Syntax	96

1 System Requirements

Minimal system requirements for MIB Designer version 5.0 or later:

- ▶ 100 MB free disk space - not including disk space required for the Java runtime environment installation.
- ▶ 512 MB free RAM (1 GB or more recommended).
- ▶ Java Runtime Environment (JRE) 9 or later installed and the JRE's `bin` directory added to the system's `PATH` environment variable.
- ▶ File system that supports filenames with up to 64 characters.

2 Installation

There several MIB Designer installation packages available for download from <https://agentpp.com/download.html>.

In general, there are the following types of installation packages available, however only the JAR is available for all supported target platforms:

- ▶ Installation package with platform integration (start menu, application icon) including the OpenJDK Java Runtime without JavaFX.
- ▶ JAR file - Help display with built-in Java HTML5 browser if JavaFX is available on the classpath of the used Java Runtime. Otherwise system browser is used to access help online or from the installed accompanied files.

See section “System Requirements” on page 1 for the system requirements of the supported platforms.

The `mds-<version>.jar` file can be used on all platforms, including Windows, but without start menu integration.

2.1 Using Native Installer

The native installation packages provide best operating system integration, for example start menu entry and an application icon on the desktop/launch menu.

To start the installation simply download and run the native installation file after download and follow the instructions.

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

2.2 Other Platforms

Download the `mds-<version>.jar` file in a folder of your choice. Start the MIB Designer application by

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

```
java -jar mds-<version>.jar
```

Once you have started MIB Designer and entered your license information, choose **File>Install...** to install MIB Designer MIB files and

repository as well as other accompanied files on your system.

2.3 Starting MIB Designer

If you have used an OS installer to install MIB Designer then you can start it from your systems application start menu.

Alternatively double click the downloaded `mds-<version>.jar` file or run `java -jar mds-<version>.jar` from the command line.

When MIB Designer 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 MIB Designer by choosing **Help>License...** from the main menu.

A MIB file can be specified as command line parameter which is then compiled and loaded on startup:

```
java -jar mds-<version>.jar <mibfile>
```

2.4 Updates and Upgrades

You can use **Help>Check for Updates** to check online (see also “Internet Proxy” on page 82) if there are free updates or upgrades available for your MIB Designer installation from <https://agentpp.com>.

If a new version is available you can choose to download it and replace your current version in-place with the update and restart the application immediately thereafter.

After the restart and if a newer version of the accompanied file set is available with the new version, MIB Designer will ask you to install them over the current installation location. If you confirm the installation, MIB Designer will overwrite existing files with their newer version unless you have activated the setting “Warn before overwriting files”. See “Repository” on page 80.

MIB Designer will not overwrite or replace any files of the `mibrepository` or `mibrepo-*` directories of your previous installation. Instead a newer version of the compiled MIB files - the “MIB Repository” - is stored in a folder named `mibrepo-<date>` where `<date>` represents the date when that versions has been built by AGENTPP.

Please enter your license including blanks! The license key, which is case sensitive, must be entered without any blanks!

MIB Designer will contact a service on <https://updates.snmp.app> to check if new updates are available for the installed version and license. When you confirm the update, the new version will be downloaded from <https://agentpp.com> from the same location you would use for a manual download.

MIB modules in a repository must not be replaced individually, because they include a unique ID that is generated during compilation of each MIB module. When mixing MIB module files from different repositories, IDs might no longer be unique, which can cause inconsistencies when loading MIB modules.

Note: The files installed by the Install menu item must be uninstalled manually, if they are no longer needed.

2.5 Uninstall

For an installation with one of the native installer packages, please use the platform specific uninstall mechanisms to remove the software itself.

Otherwise it is sufficient to remove the `mds-<version>.jar` file.

In any of both cases, you may manually remove the accompanied files installed by MIB Designer during initial startup at a location you had then chosen.

MIB Designer holds its configuration data in the `MIBDesigner4.cf` file in your home directory. To completely uninstall MIB Designer, 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 MIB Designer.

3 What Is MIB Designer?

MIB Designer is a tool to visually create and edit MIB modules that comply with the Structure of Management Information (SMI) rules. With MIB Designer there is no need to be familiar with ASN.1 or SMI syntax notation. Providing tools like drag & drop of MIB nodes, error checking, and preview with syntax highlighting, MIB Designer makes writing MIBs a question of minutes - on nearly any operating system.

While designing MIBs with MIB Designer, its tree view with integrated SMI preview guarantees best overview of the MIB information. Its search function provides fast access to any portion of the MIB. The intuitive graphical user interface ensures building syntactically correct MIBs that will compile with all SMIV1 and SMIV2 compliant MIB compilers.

MIB Designer can be used to create new SMIV2 MIB modules but it can also be used to edit existing SMIV1 or SMIV2 MIB modules. Creating of SMIV1 MIB modules is not supported since the IETF requires new MIB modules to be written in SMIV2, however you may create your MIB as SMIV2 and convert it back to SMIV1 for compatibility with SMIV1-only compilers (see “SMIV2 to SMIV1” on page 68).

The MIB Designer features are:

- ▶ Round-trip editing for SMIV1 and SMIV2 MIB modules. as well as SPPI PIB modules
- ▶ New MIB modules are automatically created using SMIV2.
- ▶ Supports editing all SMIV2 objects (including Agent-Capabilities) with popup dialogs and directly with the SMI object editor which has SMI syntax completion built-in (via <Ctrl>-<Space>)
- ▶ Intuitive graphical user interface with easy table editing.
- ▶ Preview of module and single objects with syntax highlighting.
- ▶ Text, HTML, XML, XML Schema (XSD), and PDF export of MIB modules.
- ▶ Formats and pretty prints existing and new MIB modules.
- ▶ Search MIB tree and MIB repository by regular expression
- ▶ Runs everywhere (where Java Runtime Environment 6 or later is available).
- ▶ Virtually unlimited Undo/Redo.

MODULE-IDENTITY editing is supported via popup dialog only in order to enforce revision control.

DEFVAL and DISPLAY-HINT clause content checking, for example, is not supported by most other MIB editors and compilers.

- ▶ Various consistency checks strictly following the SMIV1, SMIV2, and SPPI standards.
- ▶ Automatic refactoring of object references when the referred object is changed.
- ▶ Validates the SMI syntax of an edited MIB module or a single MIB object and selects invalid nodes in MIB tree and invalid SMI text in SMI editor window. SMI text of current MIB object is annotated with found errors and spelling issues.
- ▶ Edit object references by selecting MIB objects from option menus.
- ▶ Easy import of MIB objects and textual-conventions from other MIB modules.
- ▶ Leniently import a MIB with minimized error checking in order to be able to correct it.
- ▶ Optional revision control of MIB modules to protect them against incompatible changes.
- ▶ Editing of several MIB modules at the same time with the ability to copy, cut, and paste objects between modules.
- ▶ Visual comparison of MIB modules. Differences are shown by colored nodes in the MIB tree; differences between objects are shown by underlined text in the SMI preview.
- ▶ MIB files and objects can be edited in a text editor supporting search and replace using Perl 5 regular expressions and syntax completion through <Ctrl>-<Space>.
- ▶ Edit and print MIB files with syntax highlighting.
- ▶ Integration of external tools by configuration, for example to issue SNMP requests, view a MIB module as PDF using a PDF viewer, generate program code from a MIB module or a set of MIB modules.
- ▶ Spell checking of entered text on the fly with correction suggestions.
- ▶ GZIP compression of compiled MIB modules in the MIB repository for less disk usage and faster loading of MIB files from disk storage.
- ▶ SMI conversion from v2 to v1 and vice versa (with full undo/redo).
- ▶ Automated correction of some typical SMI(v2) syntax errors.

4 Setup

A few things need to be setup, before MIB Designer can be used to edit or create MIB modules. Please follow the steps set forth below.

4.1 Selecting a MIB Repository

When MIB Designer is started for the first time it asks to create a MIB repository if there is not any set. A MIB repository is a directory where MIB Designer stores MIB information using an internal format. A new MIB repository is created by simply creating an empty folder.

You can do this with means of your operating system or by using the Open MIB Repository Directory dialog directly.

By default, MIB Designer already installs a MIB Repository directory where you installed the accompanying files and selects this directory as MIB repository. You may change this, but you do not need to.

At any later point, the MIB repository can be changed by choosing File>Set Repository from the main menu. Switching from the current MIB repository to any other MIB repository does not alter any data of the repositories.

The option to create a new folder might not be available on all look & feels.

Please make sure when selecting the MIB repository directory that its name is shown at the bottom of the MIB repository selection window within the file name field before you press OK (see Figure 1).

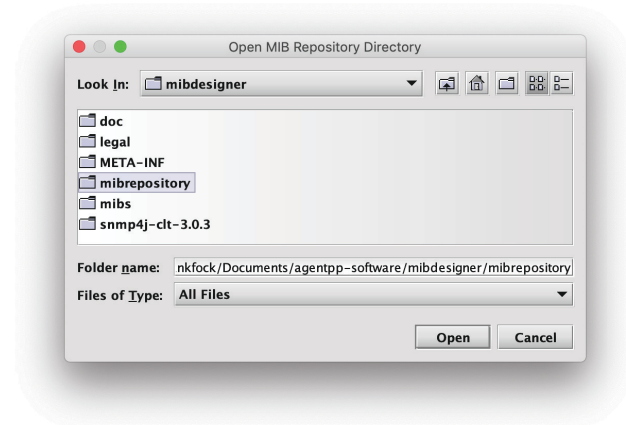





Figure 1: MIB repository selection dialog

4.2 Compiling MIB Files

MIB files may be imported into a MIB repository as follows:

MIB file ZIP archives must have a file extension of '.ZIP' or '.zip' to be recognized by the MIB compiler.

1. Use the **File>Import** menu () to import a single MIB file, check its syntax and - if it is OK - to add it to the MIB repository and load it for editing.
2. If the MIB file has errors, a text editor will open (shown by Figure 2). The encountered errors are listed above the edit area. Once the MIB file is correct, the Import button () can be used to save and import the corrected file. If the file has still errors then these errors will be displayed in the editor.
3. Use the **File>Compile MIBs** menu () to add a directory or a list of files to the MIB repository by updating any existing MIB modules. By opening a *directory* all files in that sub-tree are sorted by their import dependencies and then loaded into the repository. *ZIP* archives found in the sub-tree are opened and the included MIB files are sorted and loaded into the repository as if the archives were unpacked.
4. After having processed all MIB files, MIB Designer will report any errors in a message box. The detailed error messages can be viewed by choosing *Details...* from that message box. The *Compiler Log* window is then shown as illustrated by Figure 3. The compiler log table can be printed from the log table's context menu and sorted by column by clicking on the column's header. By clicking on a row corresponding to a MIB file, the MIB file editor right pane can be used to correct the syntax error. After pressing the editor's **Import** button, the corresponding row will be updated in the compiler log table.
5. Use the **File>Add MIBs** menu to add a directory or a list of files to the repository without updating/changing any existing MIB modules. As for the rest this method behaves similar to the above.
6. Use the **File>Import Leniently** menu if the MIB module has errors that you want to correct with MIB Designer. Please check the imported MIB module immediately after loading as described in section "Refactor Object Names and Descriptions" on page 43, because it may contain errors although it has been loaded successfully. This option reduces the error checking of the SMI parser to a reasonable minimum to facilitate the process of correcting broken MIB modules.

Section "Error Messages" on page 87 shows a list of all error messages.

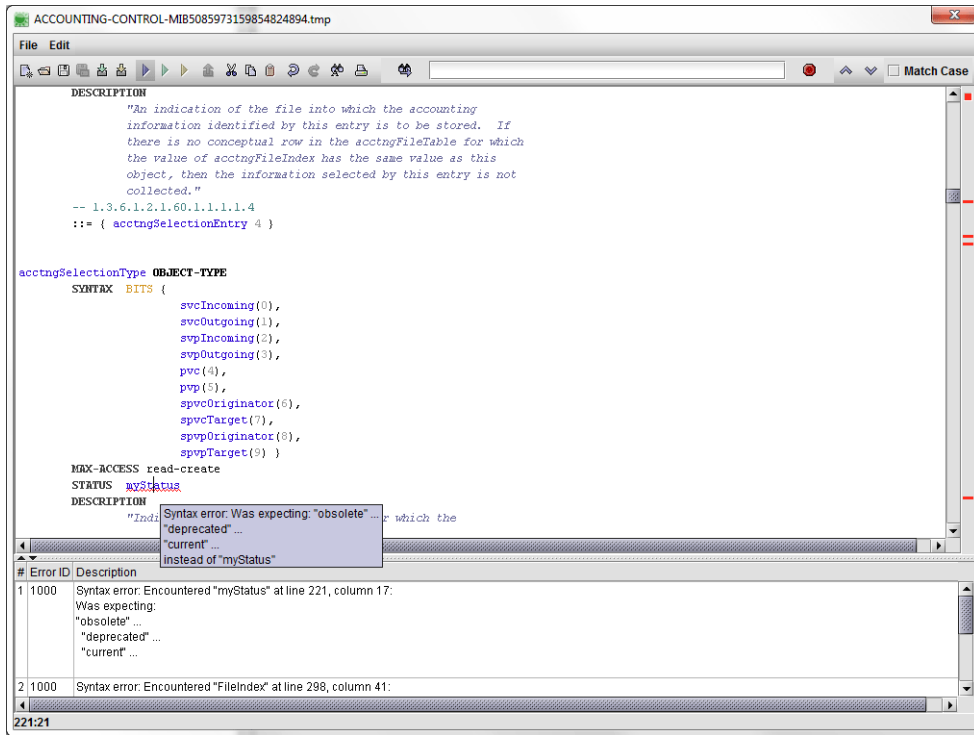


Figure 2: MIB File Editor

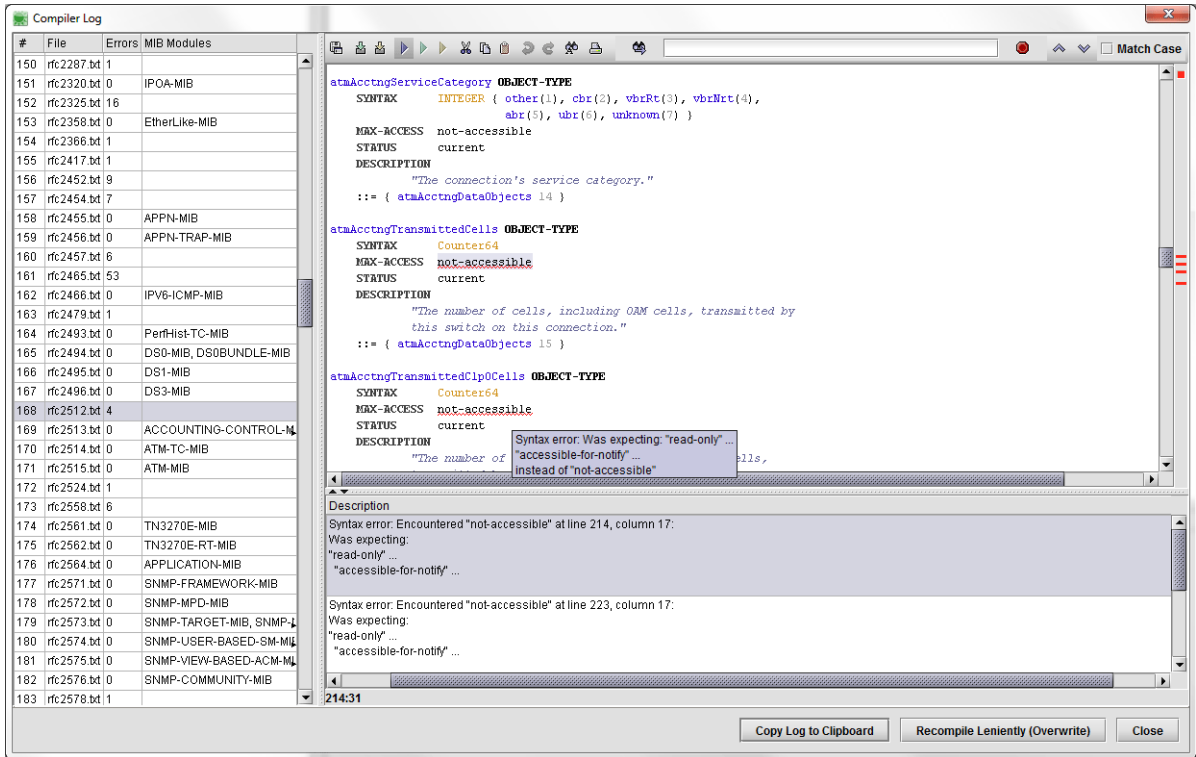


Figure 3: Compiler Log

4.3 Deleting MIB Modules

Caution: By pressing OK on the MIB deletion confirmation dialog, the MIB will be removed from the repository without providing any means for undoing the deletion!

MIB modules are removed from the MIB repository by choosing File>Delete... (🗑️) from the main menu. A shuffle dialog is opened which allows selecting the MIB module(s) to be deleted. By pressing OK, all MIB modules on the right list will be deleted. Because there is no Undo for this action, a confirmation dialog is shown.

If a MIB module depends on a MIB module that is added to the right list, the dependent MIB module will also be moved to the right.

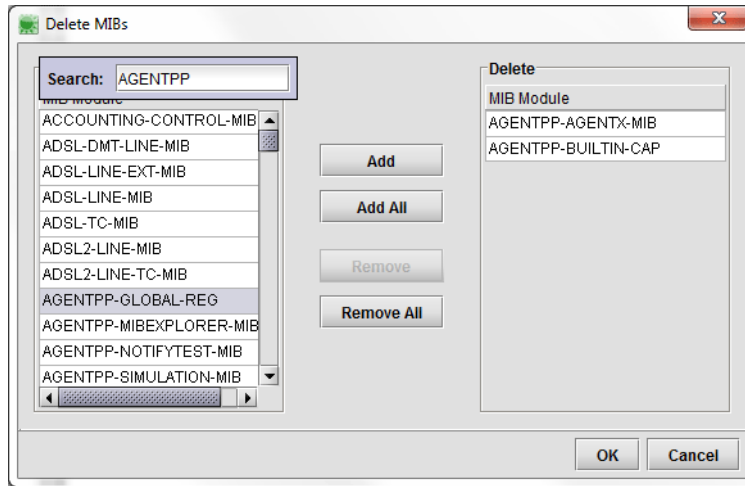


Figure 4: Deleting MIB modules.

5 Using MIB Designer

After starting MIB Designer the left (tree) window will contain the last edited (and saved) MIB module. If there is no such module you will be asked to create one with the MIB creation wizard.

The panel on the right side is divided into the SMI (object) editor and a read-only preview and navigation panel:

The syntax completion is available in the SMI Editor and the MIB File editor. See “Auto Syntax Completion” on page 47.

- ▶ **SMI Editor** - The SMI object editor shows the SMI definition of the current node under the *Objects* and *Textual-Conventions* root nodes. For objects other than a MODULE-IDENTITY construct, the SMI editor can be used to directly edit the objects definition. By pressing <Ctrl>-<Space> SMI syntax completion tries to complete the SMI text. If there is only a single valid completion, then the token (if any) at the cursor position will be replaced by the completion. If there are several completions, a popup dialog appears to select the appropriate completion by pressing <Enter> or double clicking on the item to use for completion. If there is no completion, then no changes to the text will be done. To cancel the completion dialog press <ESC>.
- ▶ **SMI Preview/Navigation** - The preview and navigation panel provides a SMI preview of the selected node. In contrast to the editor, the preview SMI panel contains additional navigation comments for easy navigation through the MIB tree using hyper links. Comparison results are also displayed in this panel. For more information about MIB module comparison see “MIB Comparison” on page 62.

The following sections describe which steps are necessary to create a new SMIv2 MIB module (“Creating a New MIB” on page 13) and how MIB Designer’s object editing dialogs can be used to edit such a MIB module (“Editing a MIB” on page 17).

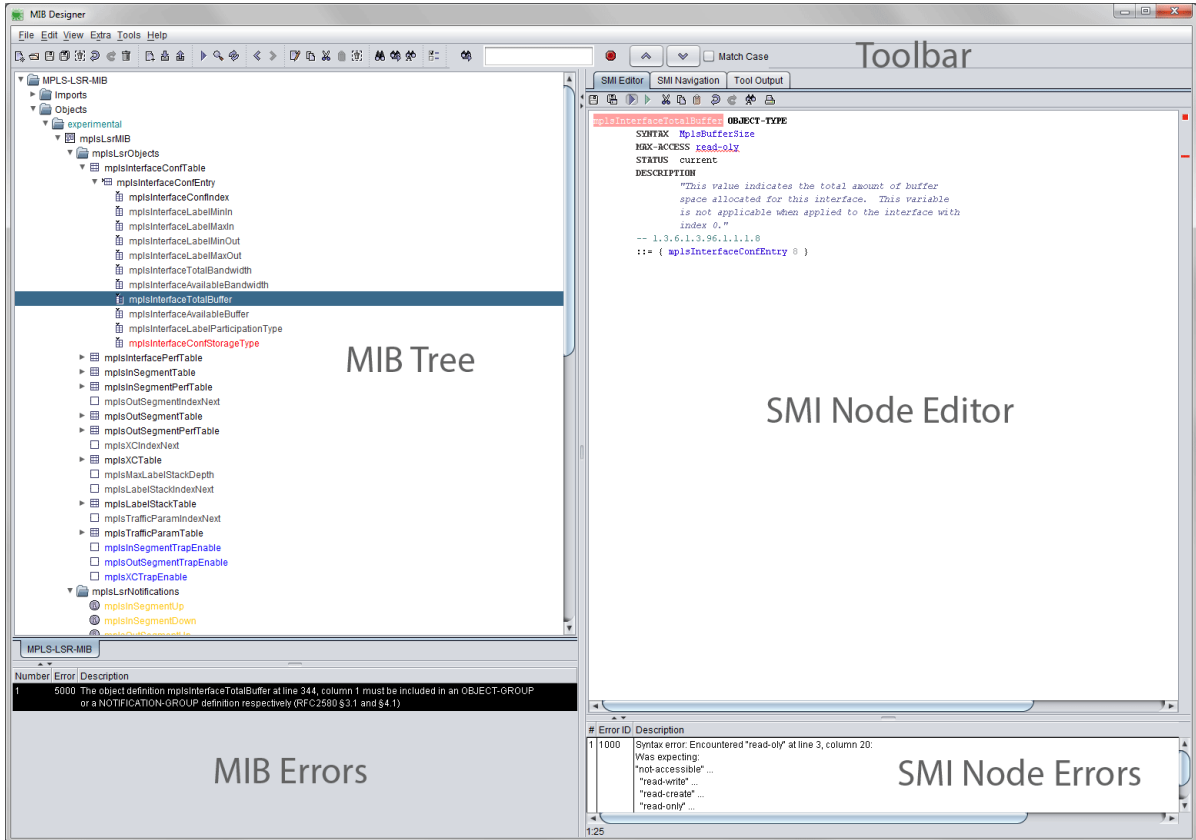


Figure 5: MIB Designer’s main window.

5.1 Creating a New MIB

In order to create a new MIB module select **File>New** (L) from the main menu. A three step wizard opens. By following the wizard step-by-step, a new MIB module can be easily created - including a top level object structure and a basic set of object groups.

Before the wizard opens you will be asked to save unsaved changes. You should do so, to ensure that the verification of the MIB module object identifier after finishing the wizard finds up-to-date OID information in the MIB repository.

The New MIB Wizard is described in detail by the following section “New MIB Wizard”. If you cancel the wizard then a minimal new MIB module with the name `NEW-SNMP-MIB` will be created.

The following steps have to be performed only if the wizard has been

canceled, before the newly created module can be saved:

1. Edit the module's name by selecting the tree's root node, opening the tree's context menu by pressing the right mouse button and selecting **Edit**.
2. Import any object identifiers and textual-conventions you wish to use with the new module from other MIB modules by choosing **Edit>Imports...** from the main menu. For details on the **Imports** menu see subsection "Import" on page 17. The imported MIB objects will then appear in the MIB tree under the **Objects** and the **Textual-Conventions** node.
3. Add any **OBJECT IDENTIFIERS** or **OBJECT IDENTITIES** that need to be defined above your module identity node. A SMIV2 MIB must define a module identity node exactly once. Object identifiers and object identities are added by selecting **Add>Object Identifier** or **Add>Object-Identity** from a node's context menu, respectively.
4. Add a module identity node by selecting **Add>Module-Identity** from the context menu of the node under which you want to define it.

5.1.1 New MIB Wizard

The wizard for new MIB (and PIB) modules has three steps:

1. Specify the name of the new MIB module and a common prefix for object names of this MIB module and whether the new module should be a SNMP MIB or a non-SNMP PIB module.
2. Specify the root object identifier for the new MIB module.
3. Specify whether a default object structure and basic object groups should be created by the wizard.

Step 1

The prefix for all new object names created for this MIB module is defined by the with the **Common Object Name Prefix**. The object name prefix must start with a lower case letter (typically a lower case word) and must not contain hyphens or spaces.

You can change the prefix for a MIB module later by editing the MIB Designer configuration file located in your user directory: Search for the line that starts with DefaultObjectName. and the MIB module name for which you want to modify the default prefix: DefaultObjectName.<MODULE-NAME>=<prefix>.

Then simply replace the <prefix> text with the new object name prefix you want to set and save the text file.

The name of the module is specified with the **New MIB Module Name** field content. The module name must contain upper case letters and the hyphen (-) character only.

By checking the **Create a Policy Information Base (PIB) module** box, MIB Designer will create a PIB module instead of a SNMP MIB module. If you want to create MIBs for SNMP, you should leave the box unchecked.

Step 2

When defining a new MIB module, the second step, after creating a name for it, is to define a root object identifier (OID). There is typically only a single root OID for a MIB module, but that is not mandatory. Additional root objects can be added later using the **Edit/Imports** menu.

A possible root OID for an enterprise specific MIB module is the enterprises ID assigned by IANA (<http://www.iana.org>) or any OID defined in a subtree of that OID by an enterprise specific registration MIB module.

When creating a registration MIB module, then import the enterprises OID and assign your IANA ID in the spin field left of the respective check box.

Otherwise, choose an OID from another MIB module (**Select MIB to import root OID from**), which can be imported optionally from an external MIB file.

Use an unique sub-identifier to *avoid duplicate registrations which causes severe problems at runtime which could render your product unusable!*

MIB Designer will check uniqueness with the current MIB repository when you finish the wizard. If the OID (including the given sub-identifier) you have chosen is already defined elsewhere, then you will be warned to you can return to this step.

Step 3

With this step, a basic structure could be created that is needed by typical MIB modules. Any MIB objects created by this wizard step can be easily

removed from the MIB module if not actually needed (any more). Therefore it is recommend the use the default: create all structure objects.

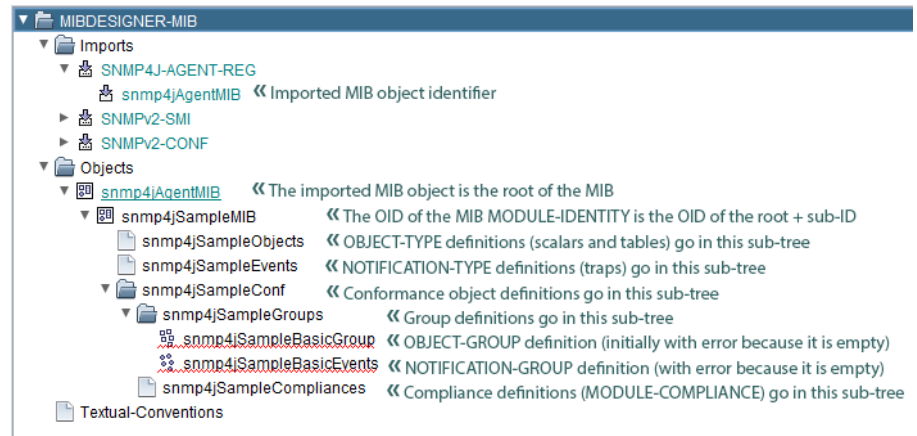


Figure 6: A sample MIB module structure created by the New MIB Wizard.

▶ Top Level MIB Structure

This option will create the top level structure including all the object identifier nodes shown by Figure 6.

▶ Create Basic Object Groups

This option will create an object and a notification group definition. Both are initially empty, which is not allowed for such groups by the SMIV2 standard. Therefore, a syntax error is displayed when the MIB has been checked for errors.

In Preferences you may define a default object and notification group by giving patterns matching those default object group names. By default those patterns are “Basic” in both cases. Thus, whenever you create a new OBJECT-TYPE or NOTIFICATION-TYPE, its object name will be added to the respective group.

By confirming the wizard with the Finish button, MIB Designer will search the MIB repository for object identifier registrations that conflict with the new MIB module’s object identifier. If such a conflict is detected you will be asked whether you really want to create the MIB module now. If you choose No, the wizard will open again and you can modify the selected root OID (or the suffix ID) in “Step 2” and try again.

5.2 Editing a MIB

With MIB Designer MIB objects can be added, edited, copied, cut, pasted and removed. These functions can be accessed through a node's context menu or through the **Edit** menu. Undo (↶) and redo (↷) of edit operations is available for the last 50 actions.

Depending on the type of the node (i.e., whether it is an OBJECT-TYPE, OBJECT IDENTIFIER, etc.) some of the menu items may be disabled. If the MIB object, for example, is an OBJECT-TYPE then the **Add** menu will be disabled, because SMI does not allow defining an object below an OBJECT-TYPE definition. All menu items of the context menu, apart from **Add**, have also counter parts on the tool bar (📁📄🗑️📄📄✂️).

With a Drag&Drop mouse operation, Cut&Paste or Copy&Paste can be performed by a single mouse click. The default drag action is 'move' which cuts the entire sub-tree rooted at the selected node and pastes that sub-tree as a new (last) child under the target node.

By pressing <Ctrl> while selecting the node to be dragged, the drag action can be changed to 'copy'. It copies a selected leaf node or sub-tree. A copy of the dragged node (and its sub-tree) is then inserted as a new (last) child of the target node. Please refer to sections "Copy", "Cut", and "Paste" on page 19 for further details on cut, copy, and paste.

The *object ID* of any MIB object that is a descendant of the **Objects** node can be dragged to any external application capable of text or string dropping.

5.2.1 Import

Before an object from another MIB module can be used or referenced in a module it must be imported. Objects are imported using the **Edit>Imports...** menu item (alternatively: <Ctrl-Alt-I> or **Import...** from the node context menu) or via the **Search MIB Repository** dialog (see "Search MIB Repository for Importing Objects" on page 42). Choosing **Edit>Imports** opens the Imports window which is shown by Figure 1.

To import an object definition or ASN.1 macro from a MIB module:

1. Select the MIB module that defines the object definition from the *Source MIB Module* list. If you are unsure where the object is defined, use the Search MIB Repository function to look it up.
2. With the **Add** or **Add All** buttons, you can select the object to be included in the MIB modules **IMPORT** clause.

To remove an object definition or ASN.1 macro from a MIB module:

1. Select the MIB module node under the *Imports* node in the MIB tree from which the definition has been imported.

Caution: When SMI macros are imported automatically, unnecessary MACRO imports will be removed from the imports clause and MACRO imports will be grouped at the bottom of each import source statement!

2. Select **Edit** from the context menu.
3. Select the object definition to be removed in the right table (named “Imported”) and press the **Remove** button. If the button is disabled then the MIB module has still references to this node. You will then have to remove those references before you can remove the import.

By activating the option “Automatically import SMI macros” in **Edit>Preferences**, MIB Designer automatically imports all SMI macros necessary for the current module whenever the module is checked by **View>Check**. When activated this is also done automatically when saving a module.

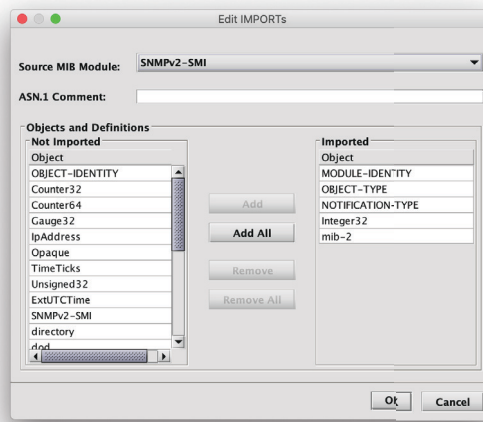



Figure 7: Imports window

5.2.2 Add

To add a MIB object to the current module, select the object under which you want to create the new object and choose **Add** from the **Edit** menu. Alternatively, you may choose **Add** from the context menu. The new object is created as the last child of the selected node. The new object has a default name and an automatically assigned object ID. Further details on editing MIB objects and an overview of all possible MIB objects and their editor windows can be found in section “Moving Objects” on page 21.

5.2.3 Copy


MIB objects are copied to an internal clipboard which is not shared with other applications.

A MIB object (and its sub-tree) is copied to the internal clipboard by selecting the corresponding node within the MIB tree and pressing <Ctrl-C> (alternatively: , **Edit>Copy**, or **Copy** from node context menu). The

copy is identical to the original nodes except for the object names. The copy's object names are changed to '*<original_name>n*', where *n* is the number of the copy starting from 0. Its object ID and those of all objects in the copied sub-tree are adapted when the object is being pasted. When the sub-tree contains a MODULE-IDENTITY construct, then this object will be transformed to an OBJECT-IDENTIFIER in the copy, because a MIB module must contain exactly one MODULE-IDENTITY.


Whenever a node is copied to the internal clipboard, its OID is copied to the system clipboard. Thus, copying a node can be used to export the OID of an object as a string to an external application. Because textual-conventions do not have an OID their object name is copied instead. Please note that when copying/cutting a sub-tree only the OID of its root node is copied to the system clipboard.

5.2.4 Cut

A sub-tree or a single MIB object is cut to the clipboard by selecting the root node of the sub-tree or by selecting a leaf node, respectively and then pressing <Ctrl-X> (alternatively: , Edit>Cut, or Cut from node context menu). A cut sub-tree can be pasted more than once, provided that it does not contain a MODULE IDENTITY node.

A cut sub-tree, or any cut MIB object other than a TEXTUAL-CONVENTION, can be pasted to OBJECT IDENTIFIERS (nodes), OBJECT IDENTITIES, and MODULE IDENTITIES only. If the cut object is a TEXTUAL-CONVENTION it can be pasted to the Textual-conventions node only.

5.2.5 Paste

A sub-tree or a single object cut or copied to the clipboard, can be inserted beneath a selected node by pressing <Ctrl-V> (, Edit>Paste, or Paste from node context menu). If an object name of any of the pasted objects is already used within the module then it will be renamed by appending 0 to its name. If its name ends on a number the number will be incremented by one. The OID of the pasted node (sub-tree) is changed to the next available OID after the last child's OID.

If the target node of the paste operation is a leaf object of type OBJECT-TYPE (i.e. a scalar MIB object) or NOTIFICATION-TYPE then the copied node will be pasted as next sibling to the target node.

Existing sibling objects and their children will be renumbered if necessary to place the pasted node.

The prefix of the object ID is given by the parent node and therefore fixed. The OID's suffix however, can be given by one or more sub-identifiers (unsigned numbers) separated by dots.

5.2.6 Edit

A selected node is edited by pressing <Ctrl>-E (📄, Edit>Edit, or Edit from node context menu). The editor windows vary from object type to object type, but common to all windows is the *Object Definition* group. Here the object's name, ID, status, description and an optional reference can be edited. Please note that depending on the edited object some of the above listed fields may be disabled. Textual conventions, for example, do not have an object ID. Module identities do not have a status.

Changes to the edited object are not committed until the user closes the editor window by pressing its **Save** button. When saving the changes, the object's ID and the name are checked for being valid and not ambiguously defined within the current MIB module. In the case of an invalid object ID or name, an error dialog is shown and the user may then correct the invalid ID or name.

In addition to editing a SMI object through editor dialogs, unreleased SMI objects can also be edited directly by using the **SMI editor** (see "Editing a MIB" on page 17). Within the editor you can enter the SMI specification of the selected node in the MIB tree. Only valid SMI syntax may be saved into the tree by either selecting another node or pressing <Alt>-S. MIB Designer allows to save a node even if the change renders the whole MIB module invalid. Thus, it is recommended to check the whole MIB module by pressing <Alt>-C after all changes have been made to a MIB module.

5.2.7 ASN.1 Comments

For each node ASN.1 comments can be entered or edited respectively. A comment can be placed at the top of each node or inline before the object identifier assignment¹.

To edit a comment, choose **ASN.1 Comments** from the context menu and then select either **Edit Top Comment** or **Edit Inline Comment**. Each entered comment line must start with two consecutive hyphens as long as the line is not empty. The next sequence of two consecutive hyphens would end the ASN.1 comment. But in most cases this is not desirable, so it is wise to avoid them.

If the leading hyphens are left out then they will be added by MIB Designer.

1. *The inline ASN.1 comment is only available for MIB objects with an object identifier assignment.*

ASN.1 comments should be used rarely, because most MIB browsers are not able to show such comments. Thus, any information that is needed to understand a MIB object or module should be described in its DESCRIPTION attribute.

The built-in spell checker marks incorrectly spelled words on the fly by a dashed line. To correct a word, a context menu with up to ten suggestions can be opened by pressing the right mouse button. If a user dictionary has been specified in the MIB Designer preferences (see “Spell Checking” on page 81) the context menu provides an **Add** button to add the selected word into the user dictionary or to always ignore it.

5.2.8 Moving Objects

MIB objects other than textual conventions can be moved upwards or downwards on their tree level by using <Alt>-<Up> and <Alt>-<Down> respectively:

1. Moving an object upwards, swaps the object identifier (OID) of the moved object with its preceding sibling. All OIDs of the objects registered in the sub-trees of the moved and the preceding object will be changed accordingly. To move an object upwards, choose **Edit>Move>Up** from the main menu or **Move>Up** from the object node’s context menu.
2. Moving an object downwards, swaps the object identifier (OID) of the moved object with its following sibling. All OIDs of the objects registered in the sub-trees of the moved and the following object will be changed accordingly. To move an object downwards, choose **Edit>Move>Down** from the main menu or **Move>Down** from the object node’s context menu.

An addition to MIB objects also import sources can be ordered by moving them up or down within the *Imports* node.

5.2.9 Renumbering Objects

The child objects of a node can be renumbered using the OID increment set in preferences by using the **Subtree>Renumber** menu item of the context menu on the parent node. The child objects are then renumbered starting with one and each next sibling child node’s last sub-identifier is assigned the last sub-identifier value of its predecessor plus the value of the OID increment setting (see “Defaults” on page 81). The descendant objects below each child are renumbered accordingly.

If a MIB module is exported with activated „generate MIB Designer comments” option (see “General” on page 78) and re-imported afterwards then the generated OID comments appear as inline comments. While exporting the module another time, MIB Designer detects that the comment is already there and will not regenerate it.

Press <Alt>-<Up> to move a node upwards in the tree.

Press <Alt>-<Down> to move a node downwards in the MIB tree.

5.2.10 MIB Object Editing Dialogs

All MIB editor windows are divided into groups that group the properties of the edited MIB object. The *Object Definition* group is common to all node editor windows and contains fields for defining the object that are:

Object Name


The Object Name field specifies the node's name. The name must start with a lower case letter for all MIB objects except textual conventions. Textual conventions must start with an upper case letter. In any case, the name must be unique with the current MIB module.

When changing a name, all references to that name within the same MIB module will be changed accordingly. For example, if a name of an index column is changed, then the INDEX clause of the corresponding table as well as the OBJECTS clause of all OBJECT-GROUP definitions referencing that columnar object will be changed too.

A default object name for new objects can be specified in the preferences dialog of MIB Designer. It is recommended to use your company's name and an abbreviation of the product or purpose that uniquely identifies your set of MIB objects in order to avoid object name clashes with other MIB modules.

Object ID

The Object ID field specifies the object identifier assigned to the node. This property consists of a read-only field denoting the parent's object name (OID prefix) and a changeable field for the node's OID suffix. In most cases, this suffix is a single sub-identifier which may be any unsigned integer value between 0 and $2^{32}-1$. In some cases it may be necessary to define a node without defining an object identifier for its direct parent, in particular when defining a module identity that is not the root node of a new MIB module.

When changing the OID suffix of a node, MIB Designer will not move the node to the assigned new location until the user refreshes the view (). This provides a more easy way of tracking changes.

Please note that the assigned OID must be unique for all nodes. Also it is allowed to define different names for the same OID by using an OBJECT IDENTIFIER construct, it is not wise to do so, because many tools available today cannot handle this correctly and there is no need for it. Because of these reasons MIB Designer does not allow defining more than one name for an OID.

Because registered OIDs must be globally unique, MIB Designer provides an easy way to check whether an OID is already assigned to any other MIB module (in the current MIB repository). By clicking on the

Object ID button, the current MIB repository will be searched for any occurrence of the assigned OID for this object. If the edited MIB module has already been saved to the MIB repository, occurrences in that MIB module will also be shown, although it is normally save to ignore them.

Status

The status field specifies the validity of the object definition. If the field is disabled a status cannot be specified for the given node. The status is then assumed to be *current*. The possible values for SMIV2 modules are:

- ▶ **current** – The definition is valid.
- ▶ **deprecated** – The definition is valid in limited circumstances, but has been replaced by another. The new definition typically encompasses a wider scope, or has been changed for ease implementation.
- ▶ **obsolete** – The definition is not valid. It was found to be flawed; could not be implemented; was redundant or not useful; or was no longer relevant.

Reference

The reference field specifies the source of the definition. It may refer to a document from another standards organization, or an architectural for a proprietary system. Although only a single line is displayed at once, multiple lines can be entered. By pressing the Reference button a text editor will open which allows a more comfortable editing of the reference text.

Like the comment editor, text entered in the reference field is background checked by the built-in spell checker. Misspelled words are marked by a dashed red underline. Words can be corrected using a suggestion list of up to ten words by opening a context menu with the right mouse button.

Description

The description field provides a textual description of the object being defined. By pressing the Description button a text editor will open which allows a more comfortable editing of the description text. In addition, the edited text is background checked for spelling errors. Misspelled words are marked by a dashed red underline. Words can be corrected using a suggestion list of up to ten words by opening a context menu with the right mouse button.

Please note that the above descriptions for the common properties of all objects are not repeated in the following subsections which describe the special properties of the respective objects.

By holding down the <Ctrl> button while pressing the Description button, spell checking for the description field can be invoked directly from the object editor.

Changes made to an object definition will not take effect until the editor window is closed by pressing the <Save> button. If an editor window is closed via the <Cancel> button any changes made to the object will be discarded.

5.2.11 Object Identifier

The OID value assignment construct OBJECT IDENTIFIER is used to assign an OID value to an identifier in the MIB module. It does assign an object name to an OID, thus the common object definition properties status, reference, and description as described in “Moving Objects” on page 21 are not available for an object identifier definition (see Figure 8). Only an identifier (the object name) and an object ID must be specified.

SMI allows assigning multiple names to a single OID. It does not allow registering an OID to multiple object definitions. The former is not recommended because there currently exist a lot of tools in the SNMP world that are not capable of handling ambiguous OID to object name mappings. MIB Designer will display a warning message when such an assignment is attempted. In case of a duplicate OID registration (done with one of the editors below), MIB Designer will display an error message and will not save the definition until the OID is changed to an unregistered one.

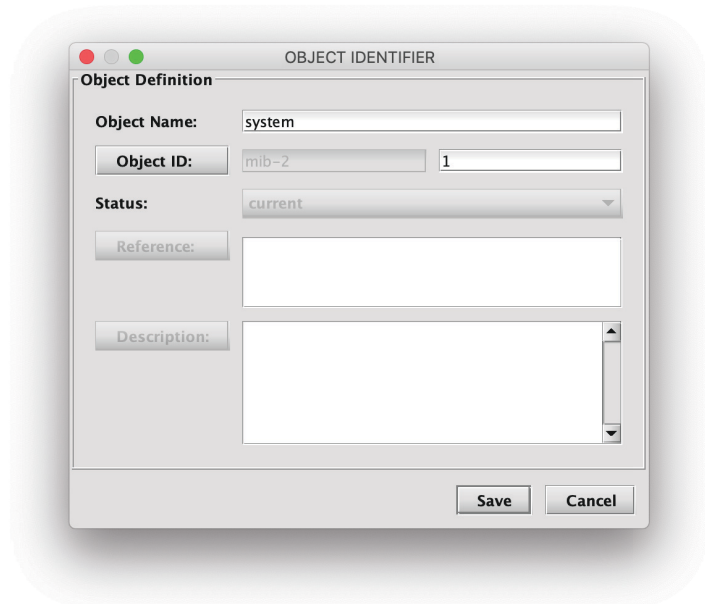


Figure 8: Object identifier editor dialog

5.2.12 Object Identity

In contrast to an OBJECT IDENTIFIER, an OBJECT-IDENTITY definition uniquely registers an OID value with an object name. A registration is a permanent assignment of an OID, which means that no other item may be registered with the same OID value.

An object identity definition supports all the properties listed in “Moving Objects” on page 21 and can be used to register an OID for an item. For example, a product, contact information for a sub-tree, or any other item which need not to be necessarily related to SNMP. The editor window for an object identity (Figure 9) looks similar to the object identifier window (Figure 8), except that the status, reference, and description fields are editable.

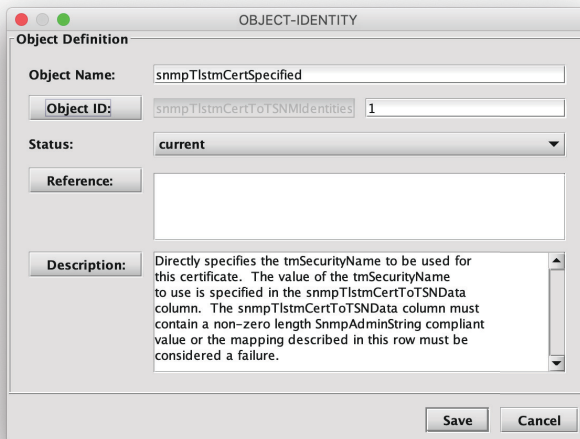


Figure 9: Object-Identity editor dialog.

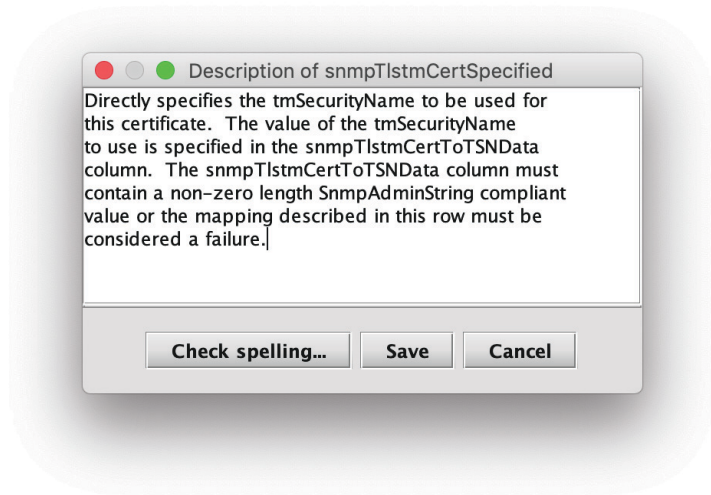


Figure 10: Text editor for a description field.

5.2.13 Module Identity

The MODULE IDENTITY construct is used to specify information about SMIV2 MIB modules. As any other object definition it consists of the *Object Definition* group described in “Moving Objects” on page 21 and a *Module* group whose fields are described below (see Figure 11). Please use the Description property to specify copyright and grant of use license when creating an enterprise specific MIB.

Last-Updated

Specifies the last date and time the current module has been modified in the ExtUTCTime data type format. The format used is an extended subset of the UTC (coordinated universal time) time format from ASN.1. The format is [YY]YYMMDDHHmmZ where:

- ▶ [YY]YY is the 2 or 4 digit year (using 4 digits is required for years after 1999);
- ▶ MM is the month (01 through 12);
- ▶ DD is the day (01 through 31);
- ▶ HH is the hour (00 through 23);
- ▶ mm is the minute (00 through 59); and
- ▶ Z is the uppercase letter Z which denotes Greenwich Mean Time (GMT).

The value of the last-updated field property must be identical to the date and time from the first revision/description clause, if present. Thus, if there is at least one revision entered, this field will be updated automatically, otherwise it can be updated to the current date and time by pressing the **Update** button.

Organization

The organization field specifies the name of the organization that has authority over the definitions created in the current MIB module.

Contact

The contact field specifies contact points for technical information.

Revision/Description

The paired REVISION/DESCRIPTION clauses are optionally used to specify information about the creation and revision of the module in reverse chronological order. In order to add a revision, press the **Add** button. A new list entry will be added to the top of the list and the Last-Updated field will be updated too. The new revision is then edited by pressing the **Edit** button. The revision editor window allows freely editing the date and time of the revision and its description or editing the UTC time by a calendar popup dialog. The popup dialog is opened by clicking on the **Revision** button. With the **Remove** button one or more revisions can be removed from the list.

If revision control is activated (see section “Exporting MIBs to XML, HTML, XSD, PDF, and Text” on page 45) in the general preferences menu, then adding a new revision will lock all objects defined in the current MIB module, that have not been locked yet through a previous revision.

Removing the latest revision will unlock all associated objects, thus all objects that have been added since the preceding revision. Removing an intermediate revision will associate the locks of that revision with the subsequently revision.

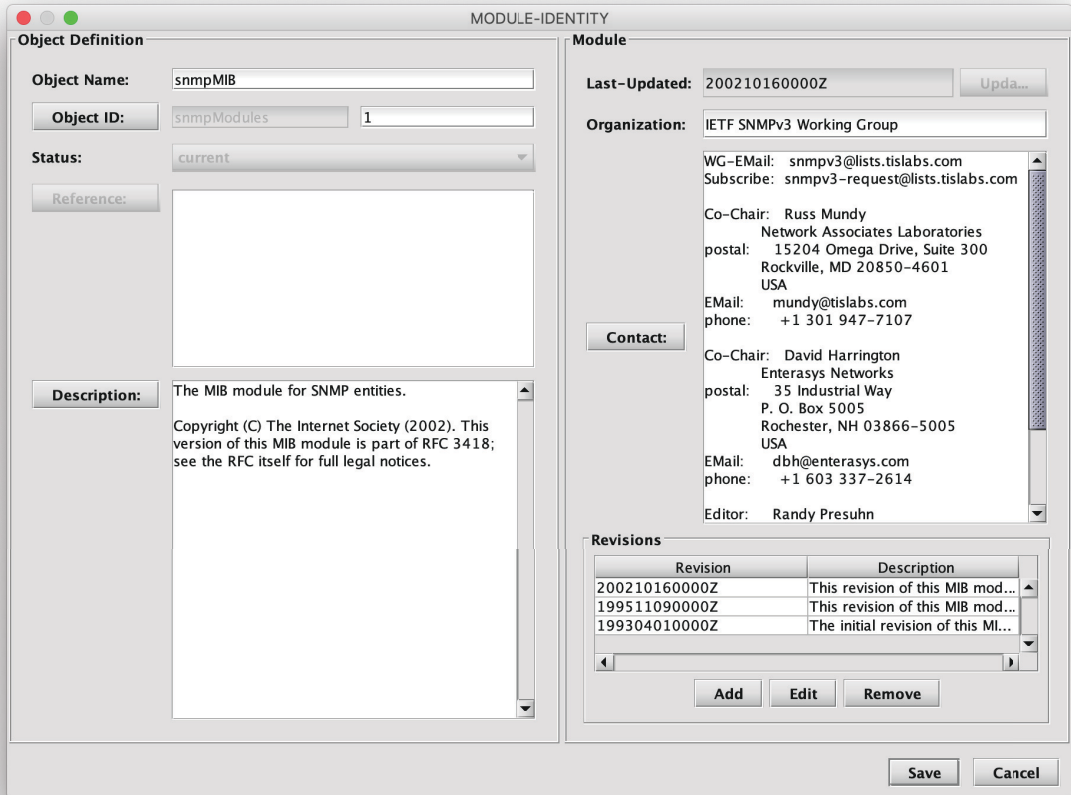


Figure 11: Module identity editor dialog.

5.2.14 Textual-Convention

Basic SMIV2 (SNMPv2/v3) data types are INTEGER, Integer32, Gauge32, Counter32, Counter64, Unsigned32, OBJECT IDENTIFIER, BITS, OCTET STRING, and Opaque.

TEXTUAL-CONVENTION definitions are used to create a new type. Since the basic data types supported by SNMP cannot be dynamically extended, new types can only be defined by adding constraints to an existing base type or a reduction in length of strings.

Although the textual-convention editor contains an Object Definition group (see Figure 12), the object name of a textual-convention must start with an upper case letter. The properties shown by the textual-convention

specific group are described below.

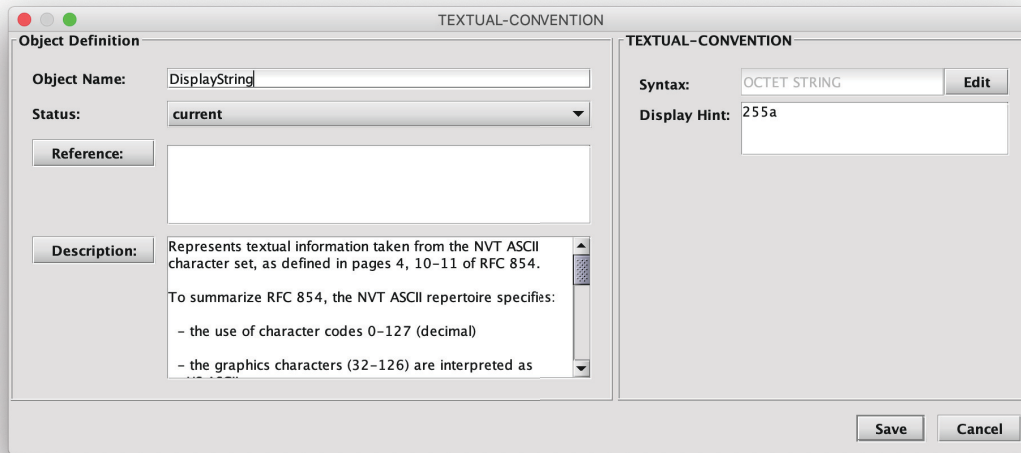


Figure 12: Textual-convention editor dialog

Syntax

The syntax field specifies the type of the syntax. The type string is shown in a read-only text field without showing possible enumeration or range values. The complete syntax definition is available by the field's tool tip. The syntax definition can be edited by pressing the Edit button. The syntax editor dialog window appears where you can choose from all possible built-in syntax types, type assignments and textual-conventions that were imported or were defined in this MIB module.

If you want to use a type assignment (SMIv1) or textual convention (SMIv2) that is not already imported then you can use the Import button to select the definition and add it to the modules IMPORTS clause.

Additionally, the syntax editor window lets you specify valid string sizes and number ranges as well as enumerated values.

When defining enumerated values, you may add associated ASN.1 comments by either using the *Comment* column of the comment's context menu of the *Enumeration* table.

A scalar type cannot have ranges and enumerated values at the same time. Non-scalar types, for example OCTET STRING based types, can have size ("range") restrictions only.

Adding an object import from within the syntax editor has immediate effect although it can be undone after the MIB object editor is closed (regardless whether changes are saved or not).

Using the context menu is recommended for multi-line comments.

Display-Hint

The DISPLAY-HINT property, which need not be present, gives a hint as to how the value of an instance of an object with the syntax defined using this textual convention might be displayed. It can only be specified for types that are based on integer or octet string. Please refer to RFC 2579 section 3.1 for further details on the allowed formats.

5.2.15 Object Type

The OBJECT-TYPE construct is used to specify definitions of columnar and scalar object types, which are also called *leaf objects*. The pairing of the identity of a leaf object (its OID) and the value to identify an instance of that leaf object is called an SNMP variable. SNMP variables are the operands and results of SNMP operations.

Figure 13 shows the OBJECT-TYPE editor window, like any other object editor window it contains an *Object Definition* group and an *OBJECT-TYPE* specific group.

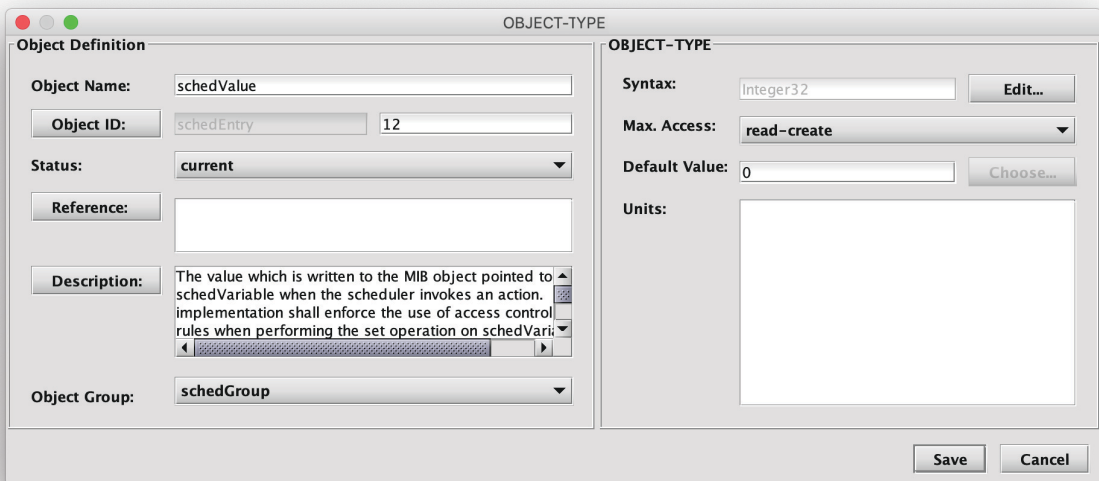


Figure 13: Object-type editor dialog.

Syntax

Specifies the syntax of the object-type in the same manner as the syntax clause of a textual convention (see subsection “Textual-Convention” on page 28).

Max-Access

Specifies the maximum allowed access to the leaf object. Possible values are:

- ▶ **not-accessible** – The object-type is a column in a table used as an index (or an index part) and may not be used as an operand in any operation.
- ▶ **accessible-for-notify** – The object-type is special operand for event report operations.
- ▶ **read-only** – The object-type may be an operand in only retrieval and event report operations.
- ▶ **read-write** – The object-type may be an operand in modification, retrieval, and event report operations.
- ▶ **read-create** – The object-type may be operand in modification, retrieval, and event report operations. Additionally, it may be an operand in a modification operation creating a new instance of the object-type.

Default Value

Specifies an acceptable value which may be used when an instance of a row is created via an SNMP modification request and the object-types value is not initialized by that request. A default value cannot be specified for index objects of tables.

For enumerated and BITS syntaxes the default value have to be chosen from the available values by using the **Choose** button.

To remove the DEFVAL clause from an OBJECT-TYPE definition empty the default value field (for enumerated or BITS values use the **Clear** button of the default value selection dialog accessible through the **Choose** button).

Units

Specifies a textual description of the units associated with the data type.

5.2.16 Table

Two OBJECT-TYPE constructs along with a SEQUENCE construct specify a definition for SMI table object. An SNMP table contains rows and columns. A table cannot be an operand or result of an SNMP operation. Thus, the maximum access for the two object-type constructs defining a table is not-accessible. Because a table is defined by two object definitions, the table editor window shown by Figure 14 has two *Object Definition* groups, named *Table* and *Entry*.

Table Editor

Table

Object Name: schedTable

Object ID: schedObjects 2

Status: current

Reference:

Descripti... This table defines scheduled actions triggered by SNMP set operations.

Entry

Object Name: schedEntry

Object ID: schedTable 1

Status: current

Reference:

Descripti... An entry describing a particular scheduled action.
Unless noted otherwise, writable objects of this row can be modified independent of the current value of schedRowStatus, schedAdminStatus and schedOperStatus. In particular, it is legal to modify schedInterval and the objects in the schedCalendarGroup when schedRowStatus is active and schedAdminStatus and schedOperStatus are both enabled.

Index

INDEX Implied length of last index object

Index Objects

schedOwner

schedName

Up Down

Add Remove Choose...

Columns

ID	Name	Access
1	schedOwner	not-accessible
2	schedName	not-accessible
3	schedDescr	read-create
4	schedInterval	read-create
5	schedWeekDay	read-create
6	schedMonth	read-create
7	schedDay	read-create
8	schedHour	read-create
9	schedMinute	read-create
10	schedContextName	read-create
11	schedVariable	read-create
12	schedValue	read-create
13	schedType	read-create
14	schedAdminStatus	read-create

Up Down

Add Edit Remove

Save Cancel

Figure 14: Table editor dialog

By convention the name for the table object definition ends with “Table” and the name for the entry (row) object ends with “Entry”.

The description property of the table object should describe the information in the table or its usage as well as estimations on the maximum number of rows and any objects whose values are associated with the table.

The description property of the entry (row) object should document whether rows can be created or deleted via SNMP operations, and if so, then what is required for this to happen. It should supplement the description of the RowStatus object of such a table.

Index / Augments

The INDEX / AUGMENTS property specifies how rows are indexed in the table. The INDEX clause lists the ordered index items for a table. Typically, the index items are names of not-accessible columns in the table. If a table consists of index columns only, then the last index column has to be read-only. In addition, read-only index columns are allowed when porting SMIV1 MIB modules to SMIV2.

The AUGMENTS clause documents a special relationship between two tables. The item specified is the entry object of another table, the *base table*. For every row in the augmenting table there has to be exactly one corresponding row in the base table with the same index value.

The total length of all index-sub-identifiers plus the length of the OBJECT-TYPE's OID must not exceed 128 sub-identifiers.

Implied Length of Last Index Object

This may be only specified for index objects whose base type is a varying length string (i.e., OCTET STRING and Opaque) or an object identifier.

The rows in a table are ordered by the value of the table's indices. If an index object has a varying length string base type, its contribution to the index OID is built by $n+1$ sub-identifiers, where the first sub-identifier is the length of the string and each following sub-identifier is the ASCII value of the corresponding character of the string. If a string or an object identifier has a fixed length then sub-identifier denoting the length is omitted. This can be forced by checking the IMPLIED property for the last sub-index.

The index objects for the table can be easily chosen using a shuffle dialog which is opened by pressing the **Choose** button.

Sub-index values with IMPLIED length must have at least one sub-identifier.

Columns

The *Columns* group specifies the columns that are part of the table. In order to add (append) a column to the table, press the **Add** button below the columns overview table. A column may then be moved within the table by editing its OID. A column may be modified by selecting it and then pressing the **Edit** button. As usual, one or more columns may be removed from the table by selecting the appropriate row(s) in the columns table and then pressing the **Remove** button.

5.2.17 Notification

The NOTIFICATION-TYPE construct is used to specify the events that can be reported by an agent (i.e., a notification originator). The OID value assigned to a notification-type is sent with a notification in order to identify it. Figure 15 shows the notification-type editor window with its *Object Definition* and *Objects* group.

Objects

The optional OBJECTS clause can be used to specify one or more scalar or columnar objects whose values describe the event. Objects can be added and removed from the notification-type definition by pressing the **Choose** or the **Remove** button respectively. Alternatively, you may press **Choose** to open a shuffle dialog with which you can choose the objects that must be at least provided with a notification.

By clicking on the right table's header of shuffle dialog you may sort the objects in ascending or descending order.

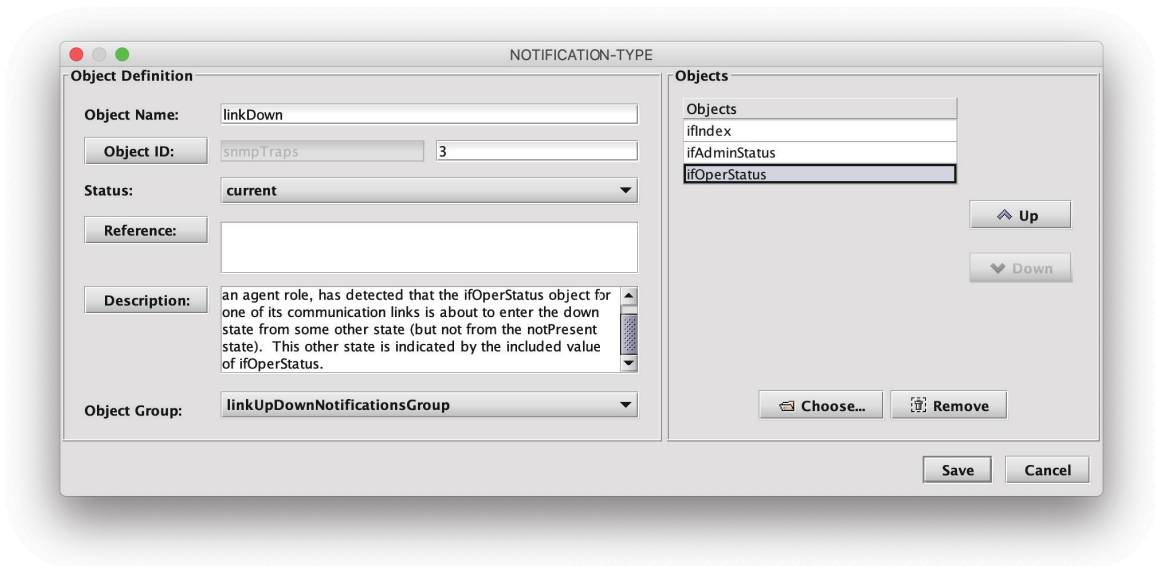


Figure 15: Notification-type editor dialog.

Object Group

Being part of at least one notification object group is mandatory for any NOTIFICATION-TYPE. The notification type editor allows to select a group directly from the editor. This is especially useful, when creating a new notification type. To edit group memberships when a notification-

type should be part of more than one notification group, then editing those groups is necessary as described in the following section.

5.2.18 Group

The OBJECT-GROUP and NOTIFICATION-GROUP constructs are used to define a collection of related object type definitions and notification type definitions respectively. Consequently, both editor windows are very similar. They consist of an *Object Definition* group and an *Objects* group.

Every object type with a value for the MAX-ACCESS clause other than “not-accessible” must be a member of at least one object group. A similar rule applies to notifications. Each notification type must be a member of at least one notification group.

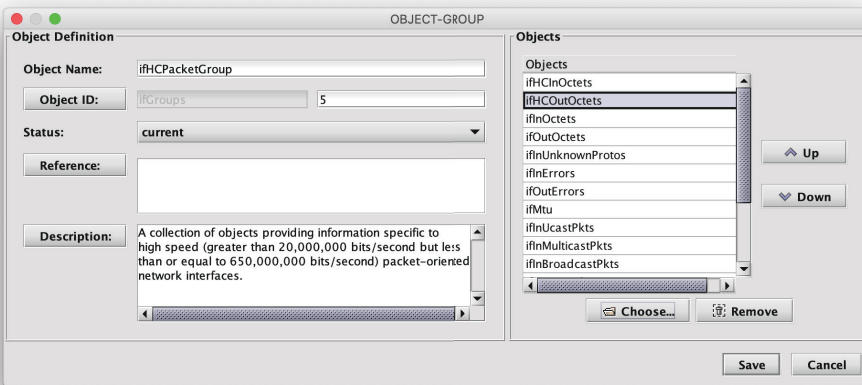


Figure 16: Object/notification group editor

Objects

The Objects group specifies one or more scalar or columnar objects that are related to each other. Objects can be added and removed by pressing the **Choose** or the **Remove** button respectively. The **Choose** button opens a shuffle dialog which can be used to add all or any subset of available objects to the group. With the special button **Add Ungrouped** adds all objects to the edited group which have not been assigned to any object/notification group yet.

Analogous to the objects editor of the NOTIFICATION-TYPE construct, objects may be sorted in ascending or descending alphabetical order by clicking on the table header.

Please note that the object types grouped through an OBJECT-GROUP should conform to the status clause of that object group definition.

5.2.19 Module Compliance

The MODULE-COMPLIANCE construct is used to convey a minimum set of requirements with respect to implementation of one or more MIB modules. Besides the *Object Definition* group, the module compliance editor window (Figure 17) contains an *Objects* group which can be used to specify a list of MIB modules for which the module compliance statement defines requirements.

Modules

Specifies a non-empty list of MIB modules for which compliance requirements are being specified. Each MIB module is named by its module name which can be selected from a combo box, which is shown when clicking on a list item. The module name may be (left) blank to refer to the encompassing MIB module. The details of a compliance requirement can be edited by selecting the corresponding module name and then pressing the Edit button.

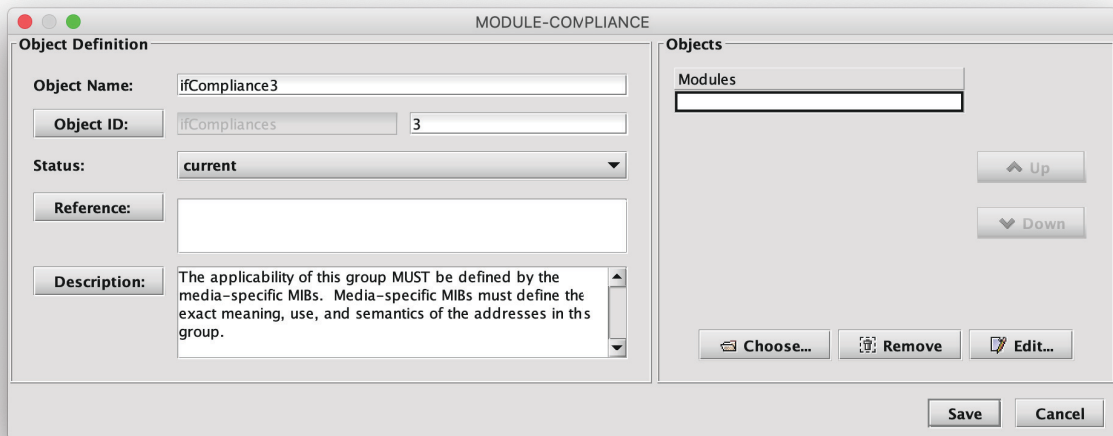


Figure 17: Module compliance editor dialog

The requirements for a compliant implementation of a module can then be edited with the dialog window shown by Figure 17.

Mandatory Groups

The Mandatory Groups clause specifies a possibly empty list of names of object or notification groups within the correspondent MIB module which are unconditionally mandatory for implementation.

Conditional Groups and Exceptions

Specifies a mix of the following two types of items:

1. Object and notification groups which are conditionally mandatory for compliance to the MIB module. In addition, unconditionally optional groups can be specified. In any case a group specified as being conditional must not be listed in the mandatory groups property at the same time.
2. MIB objects for which compliance has a refined requirement with respect to the MIB module definition. The refinement details for a list entry are shown in the *Details* group of the dialog window when the entry is selected (see Figure 17). The details can then be edited. The description property must be given. All other properties are optional and can be specified by checking the box on the right side of the property label. The required syntax and write-syntax are edited as described in section “Textual-Convention” on page 28.

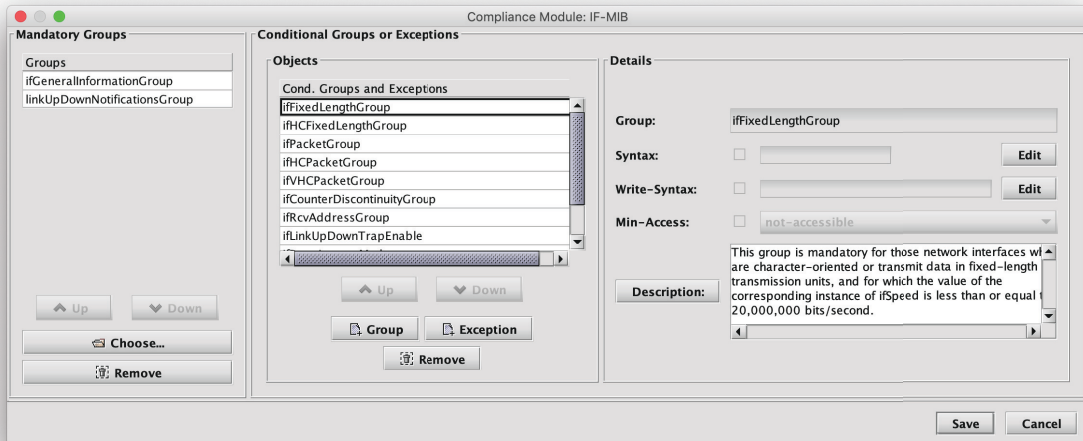


Figure 18: Dialog for editing implementation requirements details.

5.2.20 Agent Capabilities

The AGENT-CAPABILITIES construct is used to specify implementation characteristics of an SNMP agent sub-system with respect to object types and events.

Product-Release

The Product-Release field contains a textual description of the product release which includes this set of capabilities.

Supported Modules

The Supported Modules clause specifies a possibly empty list of MIB modules for which the agent claims a complete or partial implementation. Details about the implementation of a module can be edited by selecting it and then pressing the Edit button.

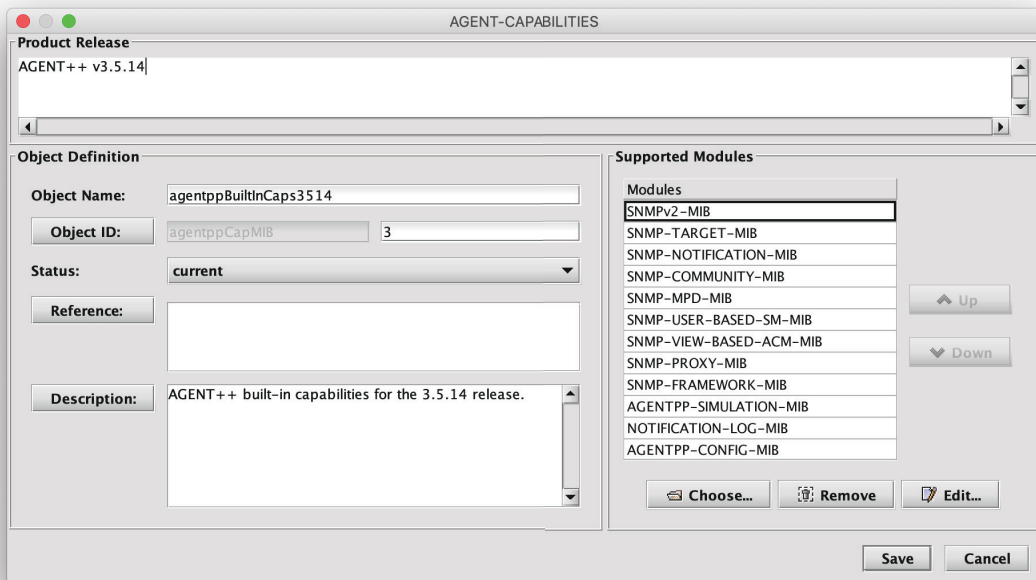


Figure 19: Agent-capabilities editor window

The details about a module implementation can be edited by using the dialog window shown by Figure 20.

Includes

The Includes field specifies a non-empty list of MIB groups associated with this supported MIB module which the agent claims to implement.

Variations

The Variations field specifies a possibly empty list of objects or notifications which the agent implements in some variant or refined fashion with respect to the correspondent OBJECT-TYPE or NOTIFICATION-TYPE definition. In order to edit the refinement details of such an object or notification, select the corresponding object name and details will be shown in the *Details* group where they can be modified too.

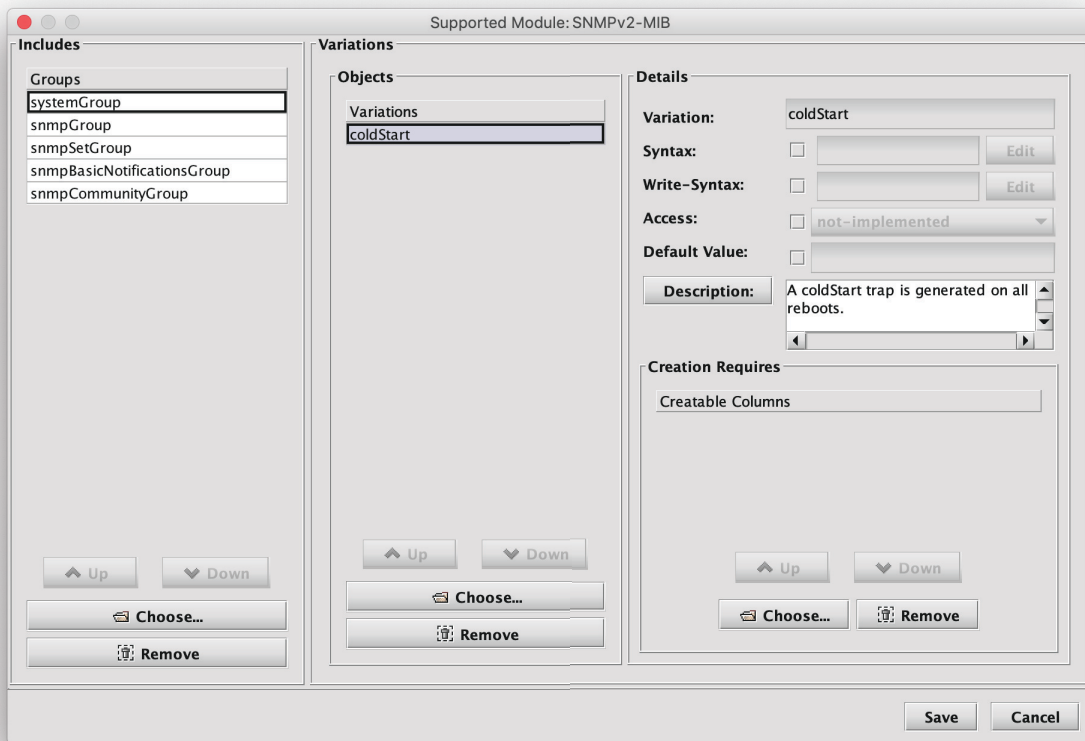














Figure 20: Supported module editor window.

5.2.21 MIB-Tree Colors and Icons

The node label *colors* in the MIB tree have the following meaning:

- ▶ Black denotes a not-accessible or accessible-for-notify MIB object as well as textual conventions or type assignments.
- ▶ Gray denotes a read-only MIB object type.
- ▶ Light-Gray denotes any MIB object that is obsolete or deprecated.
- ▶ Blue denotes a read-write MIB object type.
- ▶ Red denotes a read-create MIB object type.
- ▶ Orange denotes a trap or notification type.

By checking the option “Use SMI object type specific tree icons” under Preferences>View, object type specific tree icons are displayed instead of the default tree icons of the used Look&Feel. The icons displayed represent the following SMI object types:

-  ▶ A SMIv2 MODULE-IDENTITY definition.
-  ▶ Scalar OBJECT-TYPE definition.
-  ▶ Tabular OBJECT-TYPE definition characterized by a SYNTAX clause using “SEQUENCE OF”.
-  ▶ Table entry OBJECT-TYPE definition that defines the INDEX and columns of a conceptual table row.
-  ▶ A columnar OBJECT-TYPE definition.
-  ▶ An SMIv2 OBJECT-IDENTITY definition.
-  ▶ A SMIv2 TEXTUAL-CONVENTION definition or SMIv1 type definition.
-  ▶ An OBJECT-GROUP definition.
-  ▶ An SMIv2 NOTIFICATION-TYPE or a SMIv1 TRAP-TYPE definition.
-  ▶ A NOTIFICATION-GROUP definition.
-  ▶ A MODULE-COMPLIANCE definition.
-  ▶ An AGENT-CAPABILITIES definition.

5.3 Built-in Spell Checking

MIB Designer contains a built-in spell checker that is available for any text input field within the object editors. The spell checker contains an American English main lexicon containing more than 100,000 words as well as a British English main lexicon also containing more than 100,000 words. Besides these lexicons, a user dictionary may be specified within the **Edit>Preferences...** (F8) dialog, that can be used by a MIB author to extend the above dictionaries.

Text input fields are background checked and misspelled words are marked by a dashed red underline. By pressing the right mouse button over a (misspelled) word, a context provides a selection of up to ten correction suggestions. In addition, if a user dictionary has been specified, the selected word can be added to the user dictionary or ignored always.

The spell checker dialog can be invoked wherever the label for a text input field is a button (see also “Moving Objects” on page 21). By holding down the <Ctrl> key when pressing the label button, the spell checking for the text in the text input field is started. If the spell checking encounters an error the window shown by Figure 21 will pop up. It allows ignoring, replacing, changing, or learning the word in question.

By just pressing the label button, a text editor window can be opened that provides a more convenient way to edit extensive texts. In addition, that window has a button to start spell checking as well.

If the **Add** button is insensitive then the file for the user dictionary has not yet been specified within the preferences dialog.

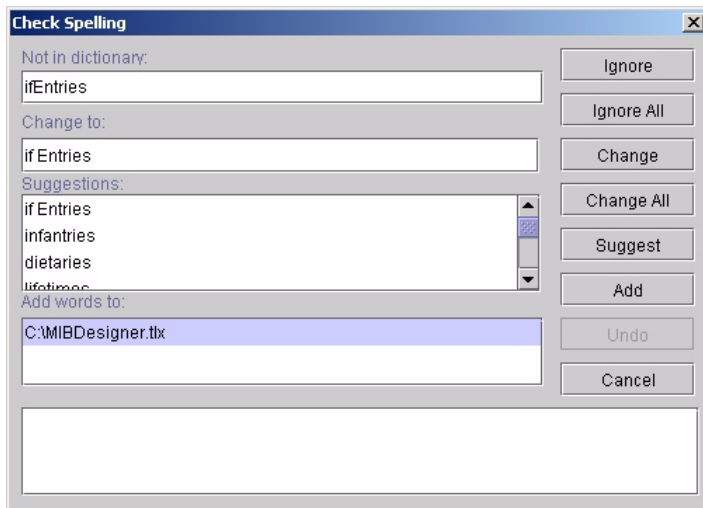


Figure 21: Spell checking dialog.

MIB Designer uses Perl 5 regular expressions which are described in the documentation of the GNU regular expression library documentation that is distributed with MIB Designer.

5.4 Finding MIB Objects

MIB Designer has the capability of searching the current MIB module or the whole MIB repository by a given regular expression. The search dialog shown by “Finding MIB Objects” on page 42 is accessed by choosing **Edit>Find** from the main menu (🔍), which will search the current MIB module or by choosing **Edit>Search MIB Repository** (see “Search MIB Repository for Importing Objects” on page 42), which will search the whole MIB repository.

Figure 22: Finding a MIB object.



Objects are searched by matching the given regular expression with the objects’ attributes that have been checked. By checking *All*, the whole SMI text, including key words (as shown in a node’s preview) is matched against the given regular expression.

When using the **Edit>Find...** menu or 🔍 button the MIB tree is searched from its root until the first matching node is found. The next matching node can then be found by using the **Edit>Find Again** menu or 🔍 button. Please note that **Find Again** always starts searching at the currently selected node in depth first order.

The incremental search panel in the tool bar provides the same search capabilities with a quick access. In addition, previous and next matches can be accessed using the **Up** and **Down** arrow of the incremental search tool bar.

5.4.1 Search MIB Repository for Importing Objects

By searching the MIB repository using **Edit>Search MIB Repository** the search results will be displayed in a table. Each row in the table represents a MIB object that matched the given search criteria. By selecting one or more rows and then pressing the **Import Selected** button, those objects are added to the **IMPORTS** clause of the currently edited MIB module. If an object is already imported by the current module, a warning message

will be displayed. If an object is a TRAP-TYPE or NOTIFICATION-TYPE an error message will be displayed, since those objects cannot be imported by a MIB module.

In addition to the search options available by the Find MIB Object dialog shown by “MIB repository selection dialog” on page 7, the search dialog for the searching the MIB Repository provides the search option *Imports* which allows to search import references. This option searches references in IMPORT, MODULE-COMPLUANCE, and AGENT-CAPABILITIES clauses that match the specified search pattern. To narrow the search results to references of a certain set of MIB modules, a search pattern for MIB module names followed by a colon (':') may be prepended.

5.4.2 Search MIB Repository for References

To avoid inconsistencies when editing a set of MIB modules, it is often useful to be able to search for references to a MIB object in other MIB modules. MIB Designer provides this feature through the menu item **Search References** in the context menu of a MIB object.

The MIB modules in the MIB repository are searched for any references (by IMPORT clauses) to the selected MIB object. All matched MIB modules and the referring objects are listed in the displayed search result dialog. By selecting one or more list items, the corresponding MIB modules can be opened for editing.

5.4.3 Navigate Between MIB Objects

An easy navigation between the recently visited MIB objects is provided through the Forward (➤) and Back (⏪) buttons on the tool bar as well as the corresponding menu items in the **View** menu.


To navigate to the adjacent MIB objects of a selected node, the “Show object navigation links” option from the View preferences can be used. When activated, this option displays navigation links within the SMI preview window. The object links are displayed as ASN.1 comments and the underlined object names of the adjacent objects can be clicked.

5.4.4 Refactor Object Names and Descriptions

When the name of an organization or product changes, it can be necessary to change the object names of a sub-tree when creating a new MIB module for the new organization or product that should include the old objects with new names for backward compatibility.

The Search and Replace function of MIB Designer can be used to replace object names and/or descriptions by a regular expression.

Do not change OIDs of released MIB objects as this would violate SMI rules and break existing applications.


To search and replace object names and/or descriptions in a sub-tree of a MIB module, select the root of the sub-tree in the MIB tree and then choose **Edit->Replace** (). A dialog that is similar to the search dialog shown by Figure 22 is displayed.

The search option for OIDs is only recommended for advanced users, because it is often easier to change OIDs by rearranging sub-trees with Copy & Paste, Drag & Drop, Move Up/Down, or changing a sub-tree OID by editing a MIB object. Nevertheless, since MIB Designer 5.0 you can search and replace OIDs too. This operation will not replace OIDs of locked MIB objects which have been released already. See “Revision Control” on page 60 for more information.


Enter the search expression in the search field and the replacement expression or string into the replace field. The replace expression may contain regular expression group references (\$1, \$2 etc.) to include parts of the matching string into the replacement string.


For each match found, you will be asked whether it should be replaced or skipped. When an object is locked, because it has already been released, it will not be included in the search result. If you choose to replace all occurrences you can undo (and redo) all replacements at once. Otherwise undo is available step by step.


5.5 MIB Validation

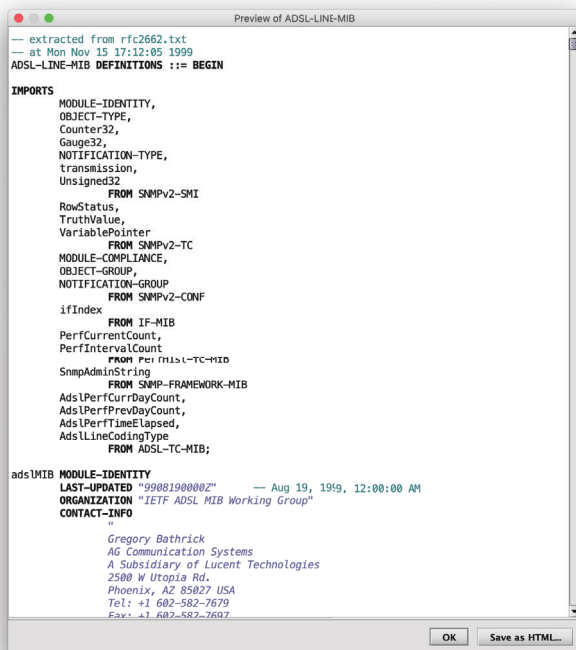
A MIB module can be checked for errors at any time by choosing **View>Check** from the main menu (). If the current MIB has any errors then they will be shown in the errors table below the SMI preview and the MIB object containing the first (selected) error is selected in the MIB tree. The error location is then highlighted in the node’s SMI preview.

5.6 Saving and Exporting a MIB

The current MIB module can be saved by choosing **File>Save** from the main menu () , which saves the module into the current MIB repository. The MIB is not checked for errors.

In order to be able to use a MIB outside MIB Designer, it is necessary to export it as a text file. This is done by choosing **File>Export as...** from the main menu (alternatively: ). You will be prompted for a file name which could be every valid file name on the used operating system. A default name is provided for convenience.

The preview function of MIB Designer (**View>Preview**, <Ctrl-P>, ) not only provides a preview of the whole SMI definition of the current MIB module, further more, it can be used to export it to HTML. Figure 23, for example, shows a detail of the preview for the ADSL-LINE-MIB.



```
Preview of ADSL-LINE-MIB
--- extracted from rfc2662.txt
--- at Mon Nov 15 17:12:05 1999
ADSL-LINE-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-IDENTITY,
    OBJECT-TYPE,
    Counter32,
    Gauge32,
    NOTIFICATION-TYPE,
    transmission,
    Unsigned32
        FROM SNMPv2-SMI
    RowStatus,
    TruthValue,
    VariablePointer
        FROM SNMPv2-TC
    MODULE-COMPLIANCE,
    OBJECT-GROUP,
    NOTIFICATION-GROUP
        FROM SNMPv2-CONF
    ifIndex
        FROM IF-MIB
    PerfCurrentCount,
    PerfIntervalCount
        FROM PERFORMANCETC-MIB
    SnmpAdminString
        FROM SNMP-FRAMEWORK-MIB
    AdslPerfCurrDayCount,
    AdslPerfPrevDayCount,
    AdslPerfTimeElapsed,
    AdslLineCodingType
        FROM ADSL-TC-MIB;

adslMIB MODULE-IDENTITY
    LAST-UPDATED "9908190000Z" -- Aug 19, 1999, 12:00:00 AM
    ORGANIZATION "IETF ADSL MIB Working Group"
    CONTACT-INFO
        Gregory Bathrick
        AG Communication Systems
        A Subsidiary of Lucent Technologies
        2500 W Utopia Rd.
        Phoenix, AZ 85027 USA
        Tel: +1 602-582-7679
        Fax: +1 602-582-7697
```

Figure 23: Example SMI preview (ADSL-LINE-MIB).

5.6.1 Exporting MIBs to XML, HTML, XSD, PDF, and Text

MIB modules can be exported from the MIB repository as XML, XML Schema (XSD), HTML, PDF, or plain text files. The text colors that can be set for printing (see “Printing a MIB module” on page 46) and syntax highlighting of the MIB file editor apply also to the PDF export function.

To export a set of MIB modules:

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`, `.html`, `.xml`, `.xsd` or `.pdf`.

Note: Any files that already exist in the destination directory might be overwritten without warning!

6. Press **Apply** to export the selected MIBs with the selected settings (dialog remains open).
7. When exporting to PDF, you will be prompted by an additional dialog for page layout and other document settings. You can choose the page size, footer, outline structure and font size.
8. Alternatively, press **OK** to export and immediately close the dialog or **Cancel** to leave the dialog without exporting any data.

5.7 Printing a MIB module

The current MIB module can be printed with syntax highlighting by choosing **Print** from the **File** menu. The operating systems print dialog will be opened, where printer and pages to be printed can be specified. The MIB file is printed black-and-white with header, footer, and line numbers by default. To change these settings go to the MIB Designer preferences dialog and select the **Printing** tab.

5.8 MIB File Editor


The current MIB module can be edited as a text file by choosing **MIB File Editor** from the **Edit** menu. The MIB File Editor is opened (see Figure). The editor has the usual capabilities of a text editor including undo and redo. In addition it has four special features that are explained in the following paragraphs.

5.8.1 Checking a MIB File


By choosing **Check (SMI Standard)** from the **File** menu the MIB file is checked for syntax errors that violate the SMIv1 or SMIv2 standard respectively. The file is not saved automatically while it is checked. In case syntax errors have been found they are displayed in the error list.


By choosing **Check (Leniently)** from the **File** menu the MIB file is checked for fundamental syntax errors. The file is not saved automatically while it is checked. In case syntax errors have been found they are displayed in the error list.

5.8.2 Saving and Compiling a MIB File

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

The default above method to import a MIB uses the syntax checking that is configured in MIB Designer preferences. In order to use a specific level of syntax checking, i.e. either the SMI standard check or a lenient syntax check.

In order to import a MIB file with SMI standard syntax checking, use the green import icon () or the corresponding menu item in the File menu.

In order to import a MIB file with lenient syntax checking, use the yellow import icon () or the corresponding menu item in the File menu.

5.8.3 Auto Syntax Completion

The syntax completion works similar to the code completion features of many Java or C++ IDEs. By pressing <Ctrl>-<Space> at any position in the edited text, MIB Designer shows possible replacements for the token under to cursor.

If there is only a single possible replacement then the text under the cursor will be replaced with it. If that is not what you wanted you need to press <Ctrl>-Z to undo the operation. Otherwise, the possible completions will be displayed in a popup dialog. An entry can be applied by double clicking on it or pressing <Enter>.

The completion alternatives are listed in alphabetically ordered. You can jump within the list by typing the first letter of the entry you search. From the possible replacements listed below only those are displayed that would syntactically fit at the cursor position:

- ▶ SMI tokens like DESCRIPTION, SYNTAX, read-write, etc.
- ▶ Lowercase names of the MIB module edited
- ▶ Uppercase names of the MIB module edited
- ▶ 0 as placeholder for a positive number including zero
- ▶ 1 as placeholder for a positive number excluding zero
- ▶ ' ' b as placeholder for a binary string constant
- ▶ ' ' h as placeholder for a hexadecimal string constant
- ▶ "255d" as placeholder for an OCTET STRING DISPLAY-HINT format definition
- ▶ "d" as placeholder for an DISPLAY-HINT format specification of a numeric SMI variable.
- ▶ " " as placeholder for a character string constant.
- ▶ "YYYYMMDDhhmmZ" representing the current date and time in ExtUTCTime format. See "Last-Updated" on page 26.

Note: The shown completion alternatives are based on syntax analysis only. Semantically, some of those alternatives may not make any sense. This will be, however, reported by the validation checks which need to be run manually. See "MIB Validation" on page 44.

If the contents of the edited MIB file cannot be analyzed, lower- and upper case names cannot be listed. Instead, the placeholders `lowerCaseName` and `UpperCaseName` are displayed.

5.8.4 Printing with Syntax Highlighting

To print the MIB file loaded into the MIB File Editor, choose **Print** (🖨️) from the editor's **File** menu. See also section "Exporting MIBs to XML, HTML, XSD, PDF, and Text" on page 45.

5.8.5 Search and Replace by Regular Expressions

A powerful way to make modifications to a MIB file is searching and replacing by regular expression. Section "Regular Expression Syntax" on page 96 gives a brief description of regular expression syntax.

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-`.

Many common errors in MIB files can be corrected by using the RE search and replace function. Here are three examples:

1. `INTEGER` may only be used for enumerations in SMIv2. To replace `INTEGER` by `Integer32` for other definitions use:
 Search Expression: `(\s+) INTEGER(\s+) (?!\{)`
 Substitution Expression:
`$1Integer32$2`
2. Within `SEQUENCE` constructs sub typing (i.e. range or size restriction) is not allowed. To delete such sub typing in `SEQEUNCE` con-

structs use (enter search expression as single line without spaces):

Search Expression:

```
(\n\s* [a-z] [a-zA-Z0-9]*\s*\n*\s+  
[A-Z] [a-zA-Z0-9]*\s*[A-Z]*) \s*\ (\s*  
(\d+.* ) | (SIZE\s*\ (.*\))) \)
```

Substitution Expression:

```
$1
```

3. The under bar “_” character is not allowed in enumeration labels. To delete the “_” and change the following letter to uppercase use:

Search Expression:

```
( [a-z] [a-zA-Z0-9]* ) _ ( [a-zA-Z0-9]+ \s* \ (\d+ \) )
```

Substitution Expression:

```
$1\u$2
```

6 MIB Design

This section contains descriptions, explanations, and solutions for the top ten MIB Design errors. These issues have been collected over years from support questions and consulting projects. Soon it turned out, that a few misunderstandings of the Structure of Management Information RFCs produce a majority of over 80% of the syntax errors. This section should help MIB authors to identify and avoid these, unfortunately very common, errors in order to increase interoperability and usability of SNMP based solutions.

Readers are encouraged to view also the following documents:

- ▶ *Guidelines for Authors and Reviewers of MIB Documents* (RFC 4181).
- ▶ *Configuring Networks and Devices with Simple Network Management Protocol (SNMP)*, section 3, Designing a MIB Module (RFC 3512).

Why do so many (enterprise) MIB modules contain syntax errors and other design flaws? The main reason is probably, a lack of good MIB design tools (editors and compilers) in the early years of SNMP. MIB authors relied on inaccurate implementations of MIB parsers that were not developed to do strict syntax and semantic checking but rather designed to be error-forgiving. With an increasing number of available SNMP tools, interoperability problems also increased caused by the diversity of different error checking levels and capabilities.

Another reason for many interoperability issues is likely to be the “bad habit” of many MIB compilers and tools to provide customizable error reporting levels allowing users to disable reporting of errors/warnings although these errors - or even more worse - warnings report SMI standard violations.

MIB Designer fills this gap with an unreached combination of a SMI conforming MIB compiler with strict syntax checking and an intuitive graphical user interface. MIB Designer has only two levels of syntax checking: *lenient* and *SMI standard*. With the second level you can be sure to avoid interoperability issues caused by SMI standard violations. The lenient level should be used to more easily fix a MIB module *only!*

The following list of common MIB Design issues is by far not complete by means of a complete collection of MIB design errors or pitfalls. Nevertheless it tries to shed some light on the most commonly made and fewest understood errors:

- ▶ Every SMIV2 MIB module must define exactly one MODULE-IDENTITY immediately following IMPORTS.
- ▶ Descriptors must start with a lower case letter and MIB module names and type or textual convention definitions names with an upper case letter.
- ▶ In SMIV2 sub-typing and enumerating values are forbidden in SEQUENCE clauses.
- ▶ Descriptors *must not* contain underscore ('_') characters
- ▶ The ASN.1 primitive type 'INTEGER' should only be used for named-number lists in SMIV2.
- ▶ Every accessible OBJECT- and every NOTIFICATION-TYPE definition must be contained in at least one object group.
- ▶ The ExtUTCTime format used for LAST-UPDATED and REVISION clauses is [YY]YYMMDDhhmmZ.
- ▶ A TEXTUAL-CONVENTION cannot refer to a previously defined TEXTUAL-CONVENTION.
- ▶ The elements in a SEQUENCE clause must match a table's lexicographic ordered columns exactly.
- ▶ Mixing SMIV1 and SMIV2 constructs and clauses in the same MIB module.

The SMIV2 MODULE-IDENTITY must immediately follow the IMPORTS construct:

RFC 2578 §3 requires that every SMIV2 MIB module starts with a MODULE-IDENTITY construct (immediately following the IMPORTS clause). Because the Structure of Management Information (SMI) lacks explicit versioning, the absence or presence of the MODULE-IDENTITY is the only usable indication for a SMI parser whether a module is written for SMI version 1 (MODULE-IDENTITY is absent) or version 2 (MODULE-IDENTITY is present). Probably caused by some MIB compilers that cannot handle object identifier forward referencing correctly, some MIB authors do not place the MODULE-IDENTITY immediately following the IMPORTS clause as shown by the example below.

```

BAD-MODULE-IDENTITY-MIB DEFINITIONS ::= BEGIN

IMPORTS
    enterprises FROM SNMPv2-SMI;

mibDesignDonts OBJECT IDENTIFIER ::= ( enterprises 4976 )
mibDesignDontsReg OBJECT IDENTIFIER ::= ( mibDesignDonts 1 )

mibDesignDontsRegMIB MODULE-IDENTITY
    LAST-UPDATED "200409201012Z"
    ORGANIZATION "AGENT++"
    CONTACT-INFO
        "Internet: http://www.agentpp.com
         http://www.mibdesigner.com
         http://www.mibexplorer.com
         Email:   fock@agentpp.com"
    DESCRIPTION
        "This MIB module illustrates a typical error when
         defining SMIV2 MIB modules. A correct MIB SMIV2
         module would have placed this MODULE-IDENTITY
         immediately following the
         IMPORTS enterprises FROM SNMPv2-SMI;"
    REVISION "200409201012Z"
    DESCRIPTION
        "The initial version."
 ::= ( mibDesignDontsReg 1 )

END

```

Figure 24: MODULE-IDENTITY does not immediately follow IMPORTS.

The attentive reader will have recognized a second error in the above example: The missing import of the MODULE-IDENTITY macro (see RFC 2578 §3.2).

MIB parsers that differentiate between SMIV1 and SMIV2 (what any validating MIB parser should) will report an error about the unexpected MODULE-IDENTITY construct in the above example. A correct version of the above MIB module would read as follows:

```
LEGAL-MODULE-IDENTITY-MIB DEFINITIONS ::= BEGIN

IMPORTS
    enterprises,
    MODULE-IDENTITY
    FROM SNMPv2-SMI;

mibDesignDosRegMIB MODULE-IDENTITY
    LAST-UPDATED "200409201012Z" -- Sep 20, 2004 10:12:00 AM
    ORGANIZATION "AGENT++"
    CONTACT-INFO
        "Internet: http://www.agentpp.com
         http://www.mibdesigner.com
         http://www.mibexplorer.com
         Email: fock@agentpp.com"
    DESCRIPTION
        "This MIB module illustrates a correct MIB SMIV2
         module where the MODULE-IDENTITY
         immediately follows the IMPORTS clause."
    REVISION "200409201012Z" -- Sep 20, 2004 10:12:00 AM
    DESCRIPTION
        "The initial version."
    -- 1.3.6.1.4.1.4976.1.1 -- ::= { enterprises 4976 1 1 }

-- Alternatively also
-- { mibDesignDos 1 } or
-- { enterprises mibDesignDos(4976) mibDesignDosReg(1) 1 }
-- would have been valid, but these variants could cause
-- incompatibility issues with some broken MIB compilers.

mibDesignDos OBJECT IDENTIFIER
    -- 1.3.6.1.4.1.4976 -- ::= { enterprises 4976 }

mibDesignDosReg OBJECT IDENTIFIER
    -- 1.3.6.1.4.1.4976.1 -- ::= { mibDesignDos 1 }

END
```

Figure 25: Legal placement of the MODULE-IDENTITY construct.

All comments (green text) in this example are optional and need not to be present. In fact some old or buggy MIB compilers have problems correctly recognize the end of comments. The end of a comment is either marked by two consecutive hyphens ('--') or the end of the line. The first should be avoided for maximum interoperability.

Descriptors start with a lower case letter whereas module names with an upper case letter:

Descriptors, i.e., object names, enumeration labels, have to start with a lower case letter. MIB module names and type names, i.e., names of TEXTUAL-CONVENTIONS and SEQUENCES, have to start with an upper case letter. For more details see RFC 2578 §3.1.

In SMIV2 sub-typing and enumerating values are forbidden in SEQUENCE clauses:

RFC 2578 §7.1.12 requires that syntax clauses of the subordinate objects do not contain sub-typing or enumeration of values. Consequently the red marked content have to be removed from the following conceptual row definition example in order to be valid SMIV2:

In this example, only the red marked portions are invalid. The object names for the OBJECT-TYPE definitions have been chosen to have a common prefix as recommended by RFC 2578, that is unique (by best effort) across other MIB modules. For this example, the common prefix is “mibdesignInvalidSequence”.

```

mibdesignInvalidSequenceEntry OBJECT-TYPE
    SYNTAX MibdesignInvalidSequenceEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "Example of a conceptual row with invalid sub-typing
        and enumerated values in SEQUENCE clause."
    INDEX {
        mibdesignInvalidSequenceIndexRowClass,
        mibdesignInvalidSequenceIndexRange }
    -- 1.3.6.1.4.1.4976.1.2.1 -- ::= { mibdesignInvalidSequenceTable 1 }

MibdesignInvalidSequenceEntry ::= SEQUENCE {
    mibdesignInvalidSequenceIndexRowClass INTEGER {
        rowClassOne(1),
        rowClassTwo(2) },
    mibdesignInvalidSequenceIndexRange Integer32 {1..100} }

mibdesignInvalidSequenceIndexRowClass OBJECT-TYPE
    SYNTAX INTEGER {
        rowClassOne(1),
        rowClassTwo(2) }
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "An example index with two possible values that
        distinguish between two classes of rows."
    -- 1.3.6.1.4.1.4976.1.2.1.1 -- ::= { mibdesignInvalidSequenceEntry 1 }

mibdesignInvalidSequenceIndexRange OBJECT-TYPE
    SYNTAX Integer32 {1..100}
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Example index with range restriction."
    -- 1.3.6.1.4.1.4976.1.2.1.2 -- ::= { mibdesignInvalidSequenceEntry 2 }

```

Figure 26: Sub-typing in a SEQUENCE clause is not allowed.

The underscore ('_') and in SMIV2 the hyphen ('-') character are forbidden in descriptors:

Descriptors (including identifiers like MIB module names and type names) must not contain other characters than:

a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9 -

The hyphen ('-') is only allowed in MIB module names and for SMIV1 descriptors and identifiers (e.g. enumeration labels). In SMIV2, hyphens are only allowed if the MIB module was converted from SMIV1 (which is hard to prove by a MIB compiler). Hyphens in enumeration labels are not allowed in SMIV2. In any case a descriptor or identifier must not end with a hyphen. One reason for the latter, might be easier mapping of enumeration labels to programming languages, where the hyphen is commonly interpreted as minus sign.

The ASN.1 primitive type 'INTEGER' should only be used for named-number lists in SMlv2:

Although RFC 2578 not explicitly forbids using the INTEGER primitive type for (non-enumerated) integer types, it is recommended to use Integer32 for such type definitions instead. When ignoring this recommendation, one has to add a range restriction on the INTEGER primitive type, to narrow its value range at least to -2147483648 to 2147483647 which is a requirement by RFC 2576 (Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework) §2.1.1.

This restriction seems to be unnecessary because INTEGER and Integer32 are indistinguishable on the wire, but theoretically the ASN.1 primitive type can represent values outside the above range. RFC 2576 probably tried to avoid misunderstandings by MIB readers familiar with ASN.1 about the possible value range of such types. Future enhancements regarding 64bit signed INTEGER values might have been also a motivation for this rule.

The following example illustrates this typical error. The error is solved simply by replacing "INTEGER" with "Integer32" in the first red marked row:

```
mibdesignDonts OBJECT-TYPE
  SYNTAX INTEGER
  MAX-ACCESS read-only
  STATUS current
  DESCRIPTION
    "The INTEGER syntax should be used only for
    enumerated values. For other values, Integer32
    should be used instead. In any case INTEGER
    must not be used without a range restriction that
    narrows the values range to 32bit."
  -- 1.3.6.1.4.1.4976.1.1 -- ::= { mibdesignDontsObjects 1 }
```

The text "INTEGER MAX-ACCESS" was marked by MIB Designer because it expected a range restriction but found "MAX-ACCESS".

Figure 27: Use INTEGER for enumerated values only.

Every accessible OBJECT- and every NOTIFICATION-TYPE definition must be contained in at least one object group:

RFC 2580 §3.1 and §4.1 respectively require that each accessible OBJECT-TYPE definition must be contained in at least one OBJECT-GROUP definition and every NOTIFICATION-TYPE definition must be contained in at least one NOTIFICATION-GROUP definition. These requirements assure that every object of a MIB module can be referenced by a compliance statement.

This kind of error is usually introduced in a MIB module when a new object is added and the MIB author forgets to add it to a group.

MIB Designer offers the option to import a MIB module with a lenient

MIB compiler mode and then adding the missing object group entries by using a shuffle dialog that shows the unassigned OBJECT-TYPES or NOTIFICATION-TYPES respectively.

The ExtUTCTime format used for LAST-UPDATED and REVISION clauses is [YY]YYMMDDhhmmZ:

The correct format for the LAST-UPDATED and REVISION fields is as follows (see RFC 2578 §2):

YYMMDDhhmmZ or

YYYYMMDDhhmmZ

where the elements of the above have the following meaning:

- ▶ YY - last two digits of year (only years between 1900-1999)
- ▶ 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 - the character Z, which denotes GMT, must always be present.

A TEXTUAL-CONVENTION cannot refer to a previously defined TEXTUAL-CONVENTION:

RFC 2579 §3.5 requires that the SYNTAX clause of a TEXTUAL-CONVENTION refers to SMIV2 base types only. Thus, it is an error to derive a TEXTUAL-CONVENTION from another TC as the following example shows:

```

AppnTOSPrecedence ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "A DisplayString representing the setting of the three TOS
        Precedence bits in the IP Type of Service field for this APPN
        traffic type. The HFR over IP architecture specifies the
        following default mapping:

                APPN traffic type          IP TOS Precedence bits
                -----
                Network                    110
                High                       100
                Medium                     010
                Low                        001
                LLC commands, etc.        110
        "
    SYNTAX DisplayString (SIZE(3))
  
```

Figure 28: Textual convention derived from another.

The above TEXTUAL-CONVENTION would have been correctly defined as follows:

```
AppnTOSPrecedence ::= TEXTUAL-CONVENTION
    DISPLAY-HINT "255a"
    STATUS current
    DESCRIPTION
        "A DisplayString representing the setting of the three TOS
        Precedence bits in the IP Type of Service field for this APPN
        traffic type. The HPR over IP architecture specifies the
        following default mapping:

        APPN traffic type          IP TOS Precedence bits
        -----
        Network                    110
        High                        100
        Medium                      010
        Low                         001
        LLC commands, etc.         110

        "
    SYNTAX OCTET STRING (SIZE(3))
```

Figure 29: Legal textual convention “derivation”.

Since ASN.1 allows type assignments to derive types from other types an evil-minded MIB author could think about defining AppnTOSPrecedence as follows:

```
AppnTOSPrecedence ::= DisplayString (SIZE(3))
```

Figure 30: Type derivation.

Although this would be legal, it is not recommended for the following reasons:

1. There cannot be associated any parsable information to an ASN.1 type assignment. In the above example important information included in the description clause would be lost.
2. RFC 2576 §2.1.1 demands that all ASN.1 type assignments should be converted to TEXTUAL-CONVENTION definitions in a SMIV2 MIB module.
3. Although MIB Designer can resolve such derivation chains even across several MIB modules, some MIB compilers cannot which could cause interoperability issues. For example, there are MIB compilers that would not recognize that AppnTOSPrecedence in the above example has inherited the DISPLAY-HINT “255a” from DisplayString.

The elements in a SEQUENCE clause must match a table’s lexicographic ordered columns exactly:

RFC 2578 §7.1.12 requires that for every columnar object of a conceptual table definition a corresponding entry is present in the SEQUENCE clause defining the syntax of the conceptual row of the table. The entries in the

SEQUENCE clause must appear in the lexicographic order of the columnar objects (thus ordered by their last sub-identifier). Normally this is not problematic, since most MIB authors order the columns by the last sub-identifier. But if this is not the case, for example if columns have been added by a new revision of a MIB module, then attention has to be paid on the order of the elements in the corresponding SEQUENCE clause.

The following example illustrates an error caused by wrong ordering of the SEQUENCE elements. To correct the error, one would have to swap `mibdesignDontsInvalidSeqCol3` and `mibdesignDontsInvalidSeqCol2` entries as indicated by the red boxes. Changing the sub-identifiers of those columns is not allowed by a revision, because this would change the behavior of the table on the wire.

```

    appearance in the MIB file."
    ::= { mibdesignDontsObjects 3 }

mibdesignDontsInvalidSeqEntry OBJECT-TYPE
    SYNTAX MibdesignDontsInvalidSeqEntry
    MX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "The conceptual row definition referencing the
        invalid SEQUENCE definition."
    INDEX (
        mibdesignDontsInvalidSeqCol1 )
    ::= { mibdesignDontsInvalidSeqTable 1 }

MibdesignDontsInvalidSeqEntry ::= SEQUENCE (
    mibdesignDontsInvalidSeqCol1 Unsigned32,
    mibdesignDontsInvalidSeqCol3 TimeTicks,
    mibdesignDontsInvalidSeqCol2 Counter32 )

mibdesignDontsInvalidSeqCol1 OBJECT-TYPE
    SYNTAX Unsigned32
    MX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "First column and index."
    ::= { mibdesignDontsInvalidSeqEntry 1 }

mibdesignDontsInvalidSeqCol3 OBJECT-TYPE
    SYNTAX TimeTicks
    MX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The third column, appearing at second position."
    ::= { mibdesignDontsInvalidSeqEntry 3 }

mibdesignDontsInvalidSeqCol2 OBJECT-TYPE
    SYNTAX Counter32
    MX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Second column, appearing at third position."
    ::= { mibdesignDontsInvalidSeqEntry 2 }

```

Figure 31: Inconsistent order of column objects in SEQUENCE and sub-tree.

Mixing SMIV1 and SMIV2 constructs and clauses in the same MIB module:

RFC 2578 §3 explicitly forbids the usage of SMIV1 macro definitions in SMIV2 modules. The usage of SMIV2 constructs like TEXTUAL-CONVENTION is forbidden in SMIV1 too. These kind of errors often occur when manually converting a module from one version to another

and when the MIB parser/compiler used to check the conversion result does not properly distinguish between SMIV1 and SMIV2.

MIB Designer clearly distinguishes between SMIV1 and SMIV2 and will reliably report such errors. In addition, it provides automatic conversion from SMIV2 to SMIV1. See RFC 3584 for situations where an automatic conversion is not completely possible.

Any new SNMP MIB module should be written in SMIV2 (the corresponding RFCs 2578, 2579, and 2580 are STANDARD). That's why MIB Designer focusses on SMIV2 and does not allow to write new MIB modules in SMIV1. Nevertheless, MIB Designer warns MIB author's when defining a NOTIFICATION-TYPE that is not backward-compatible with SMIV1 and SNMPv1 (see RFC 3584 §3 for details), because the NOTIFICATION-TYPE's second to last sub-identifier is not zero. Although RFC 3584 defines a mapping for such notifications to SNMPv1 traps, it is wise to avoid such notification definitions for better interoperability.

Text must contain 7bit ASCII characters only

For interoperability, SMI does not allow using UTF-8 and other non-7bit-ASCII characters except newline (CR and LF), tab characters (e.g. TAB), and spaces (see RFC 2578 §3.1.1). This rule applies to all clauses with text enclosed in double quotes, like DESCRIPTION, CONTACT-INFO, and REVISION for instance.

7 Revision Control

Even a released MIB module that is already used by many sites may require maintenance over time. According to the SMI rules, changes to a released MIB module are subject to some restrictions which guarantee that changes are compatible with existing implementations of that MIB specification. Although MIB Designer cannot enforce all of these restrictions, it provides powerful means to prevent users from making incompatible changes.

MIB Designer has a revision control mechanism that can be activated via the **Edit>Preferences** menu. When this mechanism is activated, a revision of a MIB module may be released by adding a revision note to its **MODULE-IDENTITY** construct (see Figure 10). Whenever a MIB module revision is being released, all objects new to that revision will be locked. Locked objects are shown in the MIB tree with underlined object name. The restrictions that apply to locked objects are listed below:

- ▶ OID and object name may not be changed.
- ▶ Objects may not be moved within the MIB tree nor removed from the MIB.
- ▶ The only way to delete an object is to set its status to *obsolete*. If a table's status is set to *obsolete*, then MIB Designer will set the status of all columns to *obsolete* too. *All objects referencing obsolete objects must also have an obsolete status.*
- ▶ Descriptions may only be changed for clarification. The behavior of the object may not be changed.
- ▶ Object lists which are part of **OBJECT-GROUP**, **NOTIFICATION-GROUP**, **NOTIFICATIONS**, **MODULE-COMPLIANCE**, and **AGENT-CAPABILITIES** may not be changed. The **INDEX** clause of a table may not be changed neither.
- ▶ The **SYNTAX** clause of **TEXTUAL-CONVENTION** or **OBJECT-TYPE** definitions may be changed only if it is an enumeration. Then, new enumerations may be added. Existing labels may be changed only for clarification purposes.

Objects added to a MIB module after it has been released, are not subject to any restrictions. These objects are displayed not underlined in the MIB tree and may be directly edited with the SMI editor.

If revision control is enabled in Preferences (see “Preferences” on page 78), all MIB modules imported into MIB Designer’s repository will be locked. Sometimes it might be useful to edit an imported module as if it has not been released yet, for example, if the MIB module has never been released yet and has been imported from an external source using MIB Designer. The Extra>Unlock MIB menu can then be used to unlock such a MIB.

Since SMIV1 modules do not have a MODULE-IDENTITY construct, the revision control is very limited. Nevertheless, the Extra>Lock MIB menu can be used to entirely lock a MIB.

Unlocking a SMIV2 MIB will remove all information about which objects belong to which revision. To retain this information, the latest revision information can be removed from the MODULE-IDENTITY construct instead.

8 MIB Comparison

MIB Designer may be used to visually compare MIB modules. This unequaled feature shows the differences between two MIB modules independently from their formatting. It provides an easy way of tracking the differences between releases of MIB modules.

In order to be able to compare two MIBs, they must be named differently and both opened (loaded). If both MIBs have to same module name, then they can be imported one after the other. In doing so, the first MIB module imported should be the older revision of the MIB and saved under a different name before the second one is imported.

8.1 Comparing Two MIB Modules

The current MIB module may be compared with any other *loaded* MIB module by selecting the **Extra>Compare...** menu item. After selecting the comparative MIB module the objects of the current module that differ from objects of that module will be displayed with an altered background color in the MIB tree as shown by Figure 32.

In the **Navigation** pane of the SMI editor area on the upper right side of the MIB Designer window, differing content will be displayed underlined as shown in the example for the SYNTAX and MAX-ACCESS clauses content.

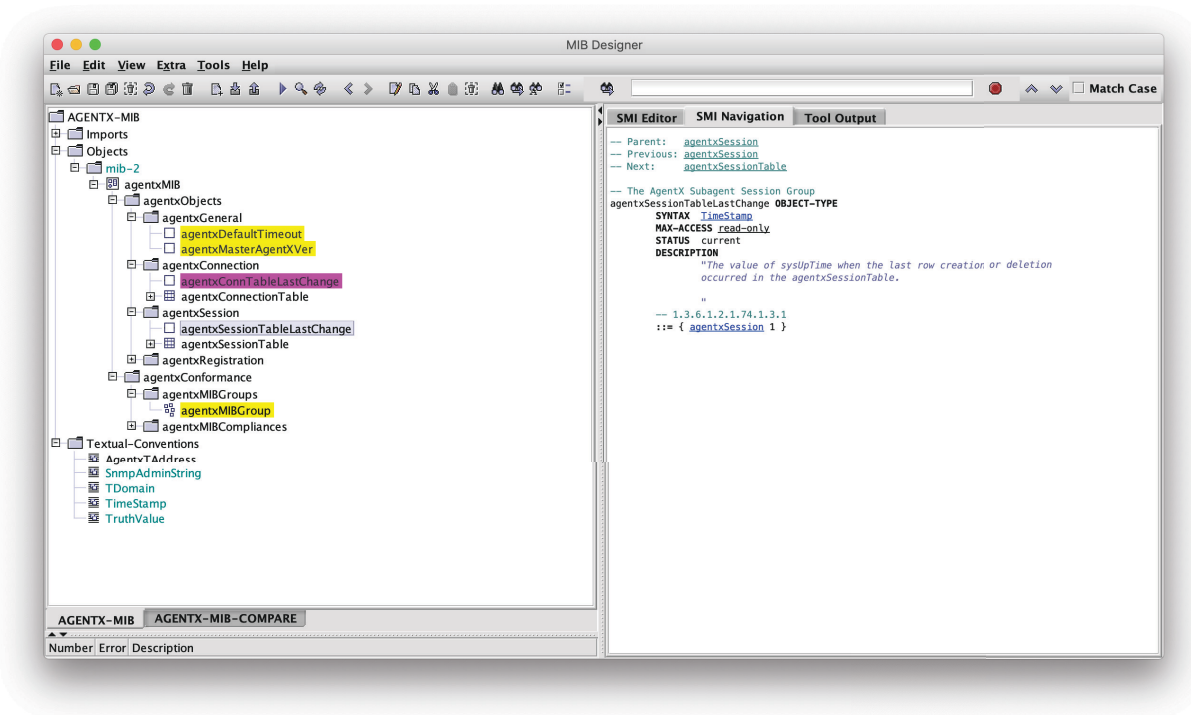


Figure 32: Example Comparison of two AGENTX-MIBs

The colors have the following meanings:

- ▶ Green – The object has been added to the current module, thus it is not part of the comparative module.
- ▶ Yellow – The object differs from the corresponding object of the comparative module.
- ▶ Magenta – The object has been changed in an incompatible way, for example, an OBJECT-TYPE has been changed into an OBJECT-IDENTIFIER. Thus, magenta indicates obvious violations of SMI rules.
- ▶ Black – The object has been deleted by changing its STATUS to *obsolete*.

For yellow and dark gray colored objects only, parts that differ from the corresponding comparative object are shown in the SMI preview as underlined text (provided that HTML preview is enabled). An object and its comparative counterpart can be displayed side by side by choosing the Show menu item from the object node’s context menu.

8.2 Clearing a Comparison

The results of a comparison can be cleared for the current module by choosing **Extra>Clear Comparison** from the main menu.

9 SMI Conversion

Within the **Extra** menu, MIB Designer provides automated SMI version conversion between SMIV1 and SMIV2 and vice versa. Although a fully automated conversion is not possible (and eligible), MIB Designer can save a lot of repetitive work.

9.1 SMIV1 to SMIV2

To convert a SMIV1 MIB module to SMIV2, choose **Extra->Convert to SMIV2**. Because SMIV2 requires a **MODULE-IDENTITY** construct, the following wizard dialog prompts for a parent OID of all objects to be created by the conversion, including the **MODULE-IDENTITY** construct.

If necessary, an **OBJECT-GROUP** and a **NOTIFICATION-GROUP** are created too.

The conversion comprises the steps set forth below. Not all steps can be performed fully automated. The list below is largely along the lines with the steps listed in §2.1 of RFC3584, however it is grouped by the level of manual intervention needed for each step.

9.1.1 Fully Automated

The following conversion steps are fully automated by MIB Designer and do not need manual intervention.

1. The **IMPORTS** statement references **SNMPv2-SMI**, instead of **RFC1155-SMI** and **RFC-1212**.
2. For any object with a **SYNTAX** clause value of **Counter**, the object's **SYNTAX** clause is changed to **Counter32**.
3. For any object with a **SYNTAX** clause value of **Gauge**, the object's **SYNTAX** clause is changed to **Gauge32**. If **Gauge32** is not the appropriate for type, it can be changed to **Unsigned32** can be manually after the conversion.
4. For all objects, the **ACCESS** clause is be replaced by a **MAX-ACCESS** clause. If the value of the **ACCESS** clause is "write-only", then the value of the **MAX-ACCESS** clause is set to "read-write".
5. For all objects, if the value of the **STATUS** clause is "mandatory" or "optional" it is set to "current" or "obsolete" respectively. Depending

on the usage of the object, its STATUS might have to be set to “deprecated” manually after the conversion.

6. If any INDEX clause contains a reference to an object with a syntax of NetworkAddress, then a new object is be created and placed in this INDEX clause immediately preceding the object whose syntax is NetworkAddress. This new object has a syntax of INTEGER, it is not-accessible, and its value is limited to the value 1.
7. For any object with a SYNTAX of NetworkAddress, the SYNTAX is be changed to IpAddress.
8. An OBJECT-GROUP is defined, and related object types are collected into that group - if there are any. Otherwise, the group object is not defined.
9. A NOTIFICATION-GROUP is defined, and related notification types are collected into that group - if there are any. Otherwise, thgroup object is not defined.
10. For any object with an integer-valued SYNTAX clause, in which the corresponding INTEGER does not have a range restriction (i.e., the INTEGER has neither a defined set of named-number enumerations nor an assignment of lower- and upper-bounds on its value), a range restriction is added to the object.
11. The value of an invocation of the NOTIFICATION-TYPE macro is an OBJECT IDENTIFIER, not an INTEGER, and is be changed accordingly. The value of the invocation is the value of the ENTERPRISE clause extended with two sub-identifiers, the first of which has the value 0, and the second has the value of the invocation of the TRAP-TYPE.

An empty DESCRIPTION clause is be added, if not already present. The ENTERPRISE clause is removed. The VARIABLES clause is renamed to the OBJECTS clause. A STATUS clause with value “current” is added. If this value is not appropriate for the objects usage then you should replace it by “deprecated” or “obsolete” as needed.

Note that the use of NetworkAddress in new MIB documents is strongly discouraged (in fact, new MIB documents should be written using SMLv2, which does not define NetworkAddress).

9.1.2 Manual Intervention or Review Needed

1. The MODULE-IDENTITY macro *must* be invoked immediately after any IMPORTs statement. You will have to specify the parent OID of the MODULE-IDENTITY construct in the wizard and then edit its DESCRIPTION, CONTACT, etc. clauses manually afterwards.

2. For any object not containing a DESCRIPTION clause, an empty DESCRIPTION clause is defined which needs to be filled manually after the conversion.

9.1.3 Not Supported

The following steps necessary for a conversion from SMIv1 to SMIv2 according to RFC 3584 are not supported by the conversion wizard. These steps have to be performed manually after conversion - if necessary:

1. For object types for which instances can be explicitly created by a protocol set operation, their object type's MAX-ACCESS clause is replaced by "read-create".
2. For any object corresponding to a conceptual row which does not have an INDEX clause, the object *must* have either an INDEX clause or an AUGMENTS clause defined. Although a missing INDEX clause is detected by the SMI check, it cannot be automatically corrected by MIB Designer.
3. For any object containing a DEFVAL clause with an OBJECT IDENTIFIER value which is expressed as a collection of sub-identifiers, the value *must* be changed to reference a single ASN.1 identifier. This may require defining a series of new administrative assignments (OBJECT IDENTIFIERS) in order to define the single ASN.1 identifier.
4. For any non-columnar object that is instanced as if it were immediately subordinate to a conceptual row, the value of the STATUS clause of that object *must* be changed to "obsolete". MIB Designer reports such an issue in its SMI check, but does not correct it automatically.
5. For any conceptual row object that is not immediately subordinate to a conceptual table, the value of the STATUS clause of that object (and all subordinate objects) *must* be changed to "obsolete". MIB Designer reports such an issue in its SMI check, but does not correct it automatically.
6. All textual conventions informally defined in the MIB module *should* be redefined using the TEXTUAL-CONVENTION macro. Such a change would not necessitate deprecating objects previously defined using an informal textual convention.
7. For any object which represents a measurement in some kind of units, a UNITS clause *should* be added to the definition of that object.
8. For any conceptual row which is an extension of another conceptual row, i.e., for which subordinate columnar objects both exist and are

identified via the same semantics as the other conceptual row, an AUGMENTS clause *should* be used in place of the INDEX clause for the object corresponding to the conceptual row which is an extension.

9.2 SMIv2 to SMIv1

Although the conversion of a MIB module from SMIv2 to SMIv1 can be almost fully automated, some information gets lost. MODULE-IDENTITY, OBJECT-GROUPs and NOTIFICATION-GROUPs definitions have to be removed from the MIB module, for example.

This conversion can be useful if a system does not support SMIv2 and that system cannot be changed to support it.

To convert a SMIv2 MIB module to SMIv1, choose Extra->Convert to SMIv1. The conversion compromises the steps set forth below.

9.2.1 Fully Automated

1. The MODULE-IDENTITY construct is replaced by an OBJECT IDENTIFIER.
2. All OBJECT-GROUP, NOTIFICATION-GROUP, and AGENT-CAPABILITIES constructs are removed.
3. For all object types, the MAX-ACCESS clause is replaced by a ACCESS clause. A value of “read-create” is replaced by “read-write”.
4. For any object with a STATUS clause of value “current” its value is replaced by “mandatory”.
5. For any object type with an AUGMENTS clause, that clause is replaced by an INDEX clause with the value of the INDEX clause of the referenced conceptual table.
6. All invocations of the TEXTUAL-CONVENTION macro are replaced by an informally defined textual convention.
7. Any UNITS clause is removed from the definition of that object.
8. The IMPORTS statement references RFC1155-SMI and RFC-1212, instead of SNMPv2-SMI.
9. For any object with a SYNTAX clause value of Counter32, the object’s SYNTAX clause is changed to Counter.
10. For any object with a SYNTAX clause value of Gauge32, the object’s SYNTAX clause is changed to Gauge.
11. For any object with a SYNTAX clause value of Unsigned32, the object’s SYNTAX clause is changed to Unsigned.

12. Any object with a SYNTAX clause value of Counter64 is removed.
13. The value of an invocation of the TRAPE-TYPE macro is an INTEGER, not an OBJECT IDENTIFIER, and is be changed accordingly. The ENTERPRISE clause is added. The OBJECTS clause is renamed to the VARIABLES clause. The STATUS clause is removed.
14. For any object with a SYNTAX clause value of “BITS” is replaced by “OCTET STRING” and a corresponding DEFVAL clause is converted from enumerated bit names to a hex string.

9.2.2 Not Supported

1. If the object identifier of a notification type has a second to last sub-identifier which is not zero, that notification type cannot be used with SMIv1.
2. If a notification type refers to an object type with effective syntax of Counter64, that notification type cannot be used with SMIv1.

10 Correction

MIB Designer provides some auto correction functions for ease of correction of common SMIV2 errors.

10.1 Index Range Correction

A numeric value used for an index must not have a negative value because a negative value cannot be represented by an OID sub-identifier. Therefore, all syntax definitions used for an index value, must have a range restriction which allows positive values only (including zero). This correction adds missing range restrictions or changes existing to exclude negative values for sub-index values.

10.2 INTEGER Usage Correction

The INTEGER type should be used for enumerations only in SMIV2. Thus, this correction changes occurrences of INTEGER to Integer32 where it is not part of an enumeration definition.

10.3 Case Correction

The SMI standard specifies the case of the first letter of descriptors (see “Descriptors start with a lower case letter whereas module names with an upper case letter:” on page 53).

Object identifiers have to start with a lower case letter, whereas SEQUENCE and MIB module descriptors have to start with an upper case letter, for example. Using an upper case letter as the first character of an enumeration descriptor is also a common error. Here a lower case letter is required.

The case correction function corrects the case of the first letter of descriptors.

10.4 SMI Macro Import Correction

The SMI specification language is derived from ASN.1. Although it is not ASN.1, it has inherited and used the ASN.1 MACRO elements. Because of that, macro definitions in the SMI standard have to be imported when used. The OBJECT-TYPE macro for instance, has to be imported from RFC1155-SMI (SMIV1) or SNMPv2-SMI (SMIV2) respectively.

This auto-correction function removes unnecessary macro imports and

adds any imports for used macros.

11 Tools

11.1 Extracting SMI 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.

11.2 Tool Configuration

External tools like a PDF viewer, SNMP tool, or code generator program like AgenPro 2 can be easily integrated with MIB Designer. Choose Tools>Configure from the main menu to configure an external program for usage with MIB Designer with the dialog shown on the left.

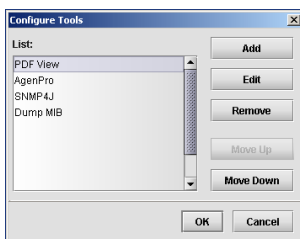


Figure 33: Tool Configuration.

In the above example, four tools have been configured. Each tool is listed by its title and gets populated in the Tools>Run Tool menu by the

same order as displayed in the configuration dialog.

To *add* a new tool, press the **Add** button and the Tool Editor dialog will be displayed where the tool can be defined by the following properties:

- ▶ A title (required), which is displayed under the **Tools>Run Tool** menu.
- ▶ The path of the tool's executable (required).
- ▶ An optional list of command line parameters. To allow a closer coupling between MIB Designer and external tool, a set of macros can be used in the parameter field. For an overview about available macros see the table below.
- ▶ An optional working directory for the external tool.

MACRO	Description
<code>\$MODULE_NAME</code>	This macro will be replaced by the currently edited MIB module name when the tool is executed.
<code>\$MODULE_AS_HTML_FILE [=<moduleName>]</code>	The current module (or the MIB module with the specified name after the optional = sign) is exported as a HTML file into a temporary file. The macro is then replaced by the file name of the temporary file on the tool's command line.
<code>\$MODULE_AS_PDF_FILE [=<moduleName>]</code>	Same as above, except that the MIB module is exported as a PDF file.
<code>\$MODULE_AS_TXT_FILE [=<moduleName>]</code>	Same as above, except that the MIB module is exported as a plain text file.
<code>\$MODULE_AS_XML_FILE [=<moduleName>]</code>	Same as above, except that the MIB module is exported as a XML file.
<code>\$MODULE_AS_XSD_FILE [=<moduleName>]</code>	Same as above, except that the MIB module is exported as a XML schema file.

Table 1: Macros for the tool configuration.

MACRO	Description
<code>\$MODULE_AS_OID2NAME_FILE [=<moduleName>]</code>	Same as above, except that the MIB module is exported as a text file where each line starts with an object identifier (OID) defined in the exported MIB module followed by an equal sign (=) and the name of the MIB object.
<code>\$REPOSITORY</code>	This macro is replaced by the path to the MIB repository of MIB Designer. This macro can be used to run command line versions of MIB Explorer and AgenPro.
<code>\$SELECTED_OID</code>	This macro is replaced by the object identifier of the currently selected node in the MIB tree of the edited MIB module. If the node does not have an OID (e.g. a textual convention) then “0.0” is inserted instead.
<code>\$LICENSE</code>	This macro is replaced by the MIB Designer license code enclosed in double quotes for entering the license part of the SNMP4J-CLT -L license option.
<code>\$LICENSE_KEY</code>	This macro is replaced by the MIB Designer license key for entering the license key part of the SNMP4J-CLT -L license option. To build the -L option completely use: -L \$LICENSE \$LICENSE_KEY

Table 1: Macros for the tool configuration.

To *remove* a tool from the configuration, select the tool in the list and press the **Remove** button. To change the order of the tools in the run menu, select a tool and press the **Move Up** or **Move Down** button.

The tables that follow provide a few example tool configurations that might be helpful illustrate the capabilities of the tool integration interface.

- ▶ The `PDF viewer` tool will use Adobe® Acrobat® to view the currently edited MIB module in its PDF representation.
- ▶ The `SNMP4JCLT Sub-Tree Browser` walks the sub-tree specified by the object identifier of the selected node in MIB Designer’s MIB tree by using `GETBULK SNMPv2c` requests. The target SNMP agent is the `localhost` on port `161`. The community used is “pub-

lic”. For more information on the command line parameters of SNMP4J see the `snmp4jclt_usage.txt` file.

- ▶ On a Windows system, the `Dump HOST-RESOURCES-MIB` tool dumps the text of the `HOST-RESOURCES-MIB` into the MIB Designer tool log.
- ▶ The `AGENT++ Stub Generation` tool executes `AgenPro` with the current MIB module name as the code generation project name. Of course, one needs to create and save the project under that name by using the `AgenPro` GUI before one can successfully run the tool. When MIB Designer and the `AgenPro` project share the same MIB repository, this tool definition automates the stub generation process.

Title	PDF Viewer
Program	C:\Program Files\Adobe\Acrobat 7.0\Acrobat\Acrobat.exe
Parameters	\$MODULE_AS_PDF_FILE
Working Directory	

Table 2: Sample configuration for a PDF Viewer.

E

Title	SNMP4JCLT Sub-Tree Browser
Program	/usr/bin/java
Parameters	-jar SNMP4J-CLT.jar -c public -v 2c -L \$LICENSE \$LICENSE_KEY walk udp:127.0.0.1/161 \$SELECTED_OID
Working Directory	~/snmp4jclt

Table 3: Example tool configuration for SNMP4JCLT sub-tree browsing. Note: You have to provide your MIB Designer license and key on first execution. The license has to be enclosed in double quotes. For more help run „-jar SNMP4-CLT.jar help“.

Title	Dump HOST-RESOURCES-MIB
Program	C:\WINDOWS\SYSTEM32\CMD.EXE
Parameters	/C type \$MODULE_AS_TXT_FILE=HOST-RESOURCES-MIB
Working Directory	

Table 4: Example tool configuration for dumping a MIB module to the console.

Title	AGENT++ Stub Generation
Program	/home/agentpp/agenpro-install/agenpro.sh
Parameters	projects/\$MODULE_NAME.prj
Working Directory	/home/agentpp/agenpro-install

Table 5: Example tool configuration for generating stub code with AgenPro by using a project file named by the current MIB modules name.

11.3 Tool Execution

To execute a tool, choose it from the **Tools>Run Tools** sub-menu. If there are no items in the sub-menu, no tools have been defined yet. See section “Tool Configuration” on page 72 for a description on how to configure tools then. The first ten configured tools can be directly run by pressing <Alt>+<1> through <Alt>+<9> and <Alt>+<0>.

Tools are executed synchronously. Thus, MIB Designer will not respond to key and mouse events until the executed tool as been terminated. The output of the last tool run is displayed in the lower right panel where MIB Designer also displays SMI checker error messages. If the executed process (tool) generated any output on stdout, the output will be displayed with blue foreground. If it generated output on stderr, then the output will be displayed with orange foreground.

12 Preferences

With the preferences dialog application wide settings can be defined and the behavior of MIB Designer can be customized. The individual settings are described in the following sections. By pressing the Save button any changes made to the preferences will be applied (except Look&Feel changes) and MIB Designer will then scan the MIB repository for available MIB modules. You can continue work while scanning or close the progress dialog if you do not want to wait until the scan is finished.

12.1 General

The general preferences section provides three sections for MIB compiler, MIB generation, and other settings.

12.1.1 MIB Compiler

The number of errors recorded during MIB compilation for each MIB file can be limited by the **Maximum Errors / MIB File** setting. If a MIB module contains more than the specified number of errors, those additional errors will not be displayed until either one of the displayed errors get fixed or the value is increased.

12.1.2 Other Options

- ▶ **Open MIB in a new Tab by default** - When enabled, new MIB modules are opened in a new tab by default, otherwise they will be opened in the current tab if that exists already.
- ▶ **Revision control** - Enables revision control, which prevents accidentally modification of already released MIB objects (see section MIB Maintenance and Revision Control).
- ▶ **Warn for Unsaved Changes** - Warns if a changed MIB module is closed or if MIB Designer is closed while there are changed MIB modules.
- ▶ **Auto save changes of SMI Editor** - Automatically save any changes made in the SMI editor area when moving the selection in the MIB tree to another node.

- ▶ **Validate MIB syntax in background** - Check MIB module syntax in background periodically and whenever node/SMI editors save their changes.
- ▶ **Warn before overwriting files** - Warns if MIB Designer tries to overwrite an existing file.

12.1.3 MIB Generation

The MIB Generation settings define how MIB modules are generated from the data imported into MIB Designer and/or entered in the same.

- ▶ **Generate MIB Designer Comments** - Adds ASN.1 comments for object identifiers and UTC time values to generated MIB modules. This option makes the MIB modules more readable and it is recommended to activate this option.
- ▶ **Generate OID Comments Inline** - In order to save output lines, the OID comments can be generated into the same line where the OID value assignment is placed. Activating this option can lead to interoperability problems with third party MIB compilers that cannot handle ASN.1 comment closing correctly.
- ▶ **Automatically Import SMI Macros** - Imports SMI macros like OBJECT-TYPE, OBJECT-GROUP, etc. automatically from the appropriate MIB modules, when a MIB module is checked or saved.
- ▶ **Preserve Original Order of Imported Objects** - Preserves the order of MIB objects in a compiled MIB module file, when it is regenerated. Subsequently added objects will be placed at the end of the MIB module.
- ▶ **Order Generated Objects by Type First** - Ensures that the objects are ordered by their type first. Within each category the order is determined by an eventually enabled preserve option (above) and the lexi-

cographic ordering of the object's OID. The order of object type categories is as follows:

- ▶▶ MODULE-IDENTITY
- ▶▶ TEXTUAL-CONVENTION
- ▶▶ OBJECT-IDENTIFIER, OBJECT-TYPE, OBJECT-IDENTITY
- ▶▶ TRAP-TYPE, NOTIFICATION-TYPE
- ▶▶ OBJECT-GROUP
- ▶▶ NOTIFICATION-GROUP
- ▶▶ MODULE-COMPLIANCE
- ▶▶ AGENT-CAPABILITIES

12.2 Repository

The Repository setting defines the directory to store compiled MIB modules, see also section “Selecting a MIB Repository” on page 7.

- ▶ **Verify MIB repository on Save** - If checked, when closing the Preferences dialog with Save then the specified MIB repository will be verified by trying to load all MIB modules therein. Any errors found will be reported with an error dialog. No MIB module will be actually loaded!

12.3 View

12.3.1 Look & Feel

The Look & Feel setting determines the overall appearance of MIB Designer. There are several built-in look-and-feels that you can choose from. MIB Designer needs to be restarted before changes will take effect.

For maximum interoperability with all operating systems, provides the Kunststoff look-and-feel. By default the operating system default look-and-feel is selected.

- ▶ **Use SMI object type specific icons** - When activated (default), the node icons displayed in the MIB tree reflect the type of the SMI object represented by the node. See also “MIB-Tree Colors and Icons” on page 40. When deactivated, the tree icons of the current Look & Feel are used.

12.3.2 Other View Settings

- ▶ **Font size of preview text** - With the slider you can specify the relative font size for the SMI preview window.
- ▶ **Use n spaces instead of tabs** - When displaying and exporting MIB files tabulators are used by default to indent text. Instead of using tabulators, the specified number of spaces can be used for indentation.
- ▶ **Force using operating system default browser to view help online** - If JavaFX is available in the Java Runtime Environment running MIB Designer, then the JavaFX browser is used and the locally installed help is displayed using the **Help->Contents...** menu. By checking this option, even then the operating system default Web browser will be used to view the help online from <https://agentpp.com/help/mds/<version>>. When an Internet connection is available this option can be useful to get the latest help. The built-in JavaFX browser uses the Internet proxy settings from “Internet Proxy” on page 82.

12.4 Spell Checking

In order to be able to enhance the built-in dictionary, you will have to specify a custom dictionary here. Any new (empty) file will be OK if you do not have a custom dictionary yet.

12.5 Defaults

For new object creation, default values can be specified for common attributes of SMI objects. Specifying a default value can ease object creation. Default values can be specified for the following attributes:

- ▶ **Object name** - Defines the default object name for new objects. It is recommended to set this value to the mnemonic of your company or organization. You will then only have to append the individual object name when creating new objects.
- ▶ **Object group** - Defines a sub-string matching pattern for the default object group assignment when creating new OBJECT-TYPE definitions.

If there are more than one matching groups, then the first one with the smallest lexicographic OID is used.

For example, if you use “Basic” here and your MIB module has an OBJECT-GROUP definition with an object name “myMibBasic-Group” then this object group will be assigned to new OBJECT-TYPE definitions.

- ▶ **Notification group** - Defines a sub-string matching pattern for the default notification group assignment when creating new NOTIFICATION-TYPE definitions.
If there are more than one matching groups, then the first one with the smallest lexicographic OID is used.
- ▶ **OID increment** - Some organizations prefer to leave holes in object numbering to be able to insert objects at later time (otherwise they could only be appended on the same level). The default is 1 which leaves no holes.
- ▶ **Syntax** - The default syntax should be set to the syntax of the majority of your MIB objects to facilitate editing, for example OCTET-STRING.
- ▶ **Access** - The default access for new OBJECT-TYPE definitions.

12.6 Syntax Highlighting

The syntax highlighting settings defines if and with which colors SMI text is to be highlighted when

1. using the MIB file editor
2. using SMI preview and navigation tab
3. MIB modules are printed or exported to PDF.

The MIB Designer default color and text style scheme can be restored by pressing the **Set Defaults** button.

12.7 Printing

- ▶ **Print colored** - If checked, syntax highlighted text is printed with the colors defined in “Syntax Highlighting” preferences. Otherwise only the text styles defined therein are used.
- ▶ **Print header** - Prints the MIB module name as header.
- ▶ **Print footer** - Prints footer with print date and page number.
- ▶ **Print line number** - Prints line numbers.

12.8 Internet Proxy

With MIB Designer 5.0 and later, a Internet proxy can be configured for:

- ▶ Viewing MIB Designer help using with the built-in JavaFX browser.
See also “View” on page 80.

- ▶ Updating MIB Designer and notifying about new (free) updates and upgrades, see “Updates and Upgrades” on page 3.

By default, MIB Designer uses the same settings as your operating system for Internet proxy. There might be cases where the proxy settings of the operating system are wrong, incompatible with Java, or otherwise not accessible.

Then please switch off the check box **Use system proxy** and provide the following parameter manually:

- ▶ **Proxy Host:Port** - The IP(v4/v6) address or the fully qualified domain name of the Internet proxy host is configured by the first field. In the second field, the TCP port is configured, which is typically 80, 8080, or 3128.
- ▶ **No Proxy Hosts** - A list of domain names or IP addresses, separated by a pipe symbol (|) for which no proxy should be used. This should include the at least `localhost|127.*|[::1]` to allow MIB Designer’s Java Runtime services on the local host.
- ▶ **Proxy server requires password** - Check this option if the proxy you want to use requires authentication using user name and password.
- ▶ **Proxy User** - The user name for the proxy authentication.
- ▶ **Proxy Password** - The password for the proxy authentication.
Note: The password will be stored in clear text in the MIB Designer configuration file in your home directory.

13 Trouble-Shooting

This section provides information and guidance on how to approach the following issues:

- ▶ License information is not accepted.
- ▶ MIB file does not compile.
- ▶ How to increase the maximum memory size for MIB Designer.
- ▶ MIB objects seem to be read-only.
- ▶ SMI compiler reports error 1000 without an error description.
- ▶ How to get support if MIB Designer hangs or otherwise does not work as expected.

License Information Is Not Accepted

If you enter your license information and the license gets rejected with “The current license information invalid!” then please check the following:

- ▶ Is the Java Runtime Environment installation of version 1.5 or later and correctly installed? You can check this by running

```
java -version
```

from the command line.
- ▶ Use Copy&Paste to enter the license key value in order to avoid typing errors. License keys are case sensitive.
- ▶ If you are using a temporary license then check the system time.

MIB File Does Not Compile

When a MIB file does not compile because of syntax errors then follow the steps below to resolve the errors:

1. Look up the error code in section “Error Messages” on page 87.
2. Try to understand the error and correct it with the MIB editor described in section “MIB File Editor” on page 46.
3. If you do not understand why MIB Designer reports an error then consult section “MIB Design” on page 50 about common MIB design and syntax errors.

4. If there are too many errors to fix manually then you can try to compile the MIB file with lenient error checking by using **File>Import MIB leniently**. After you have successfully imported the MIB file, you can then fix the module by using the MIB Designer object editors. You can check a MIB module for syntax errors at any time by using **View>Check**.
5. If the steps above cannot solve your problem, ask for support at support@mibdesigner.com.

How To Increase the Maximum Memory Size

For most situations the default maximum memory size of the Java 2SE Runtime Environment is absolutely sufficient. When you need to compile several thousands of MIB files at once or if you are working with very large MIB modules, then increasing the maximum memory size above 256 MB can be necessary.

To specify a non-default maximum memory size of 512MB for MIB Designer, start it from the command line within the MIB Designer installation directory with:

```
java -Xmx512m mds-<version>.jar
```

MIB Objects Seem To Be Read-Only

SMI objects represented by nodes with underlined node name are read-only - to be precise - most attributes of those nodes are read-only. Such objects are released objects. Released objects are protected by MIB Designer against incompatible changes.

If you are sure that a MIB module has not been released or used yet, then you might unlock the whole module by choosing **Extra>Unlock MIB**. For more details see section “Revision Control” on page 60.

SMI Compiler Reports Error 1000 Without Error Description

If error 1000 is reported without an error description, then an internal error occurred. This might be a bug, but it can also be caused because MIB Designer ran out of memory. If you are unsure whether there is enough memory available, you can open the About dialog from **Help>About** to check the free memory.

If lack of memory can be ruled out as problem cause, then please contact support by writing an email to support@mibdesigner.com. Please specify the MIB Designer version and the operating system you are using.

Getting Help

If MIB Designer hangs or otherwise shows malfunction, detailed information about the error can be found on the console of the MIB Designer application which can be seen when running MIB Designer from the command line using `java -jar mds-<version>.jar`.

To get help, please send a email to support@agentpp.com with the following information:

- ▶ MIB Designer version number
- ▶ Java Runtime version and vendor
- ▶ Output from the console when the error occurred (see above).
- ▶ A brief description of the expected behavior and the actually observed behavior.

14 Error Messages

The following table lists the error messages of the MIB compiler. Most error texts contain placeholders, like <X>, <Y>, etc., which are replaced by the MIB compiler with values describing the context of the error. Please see the description text for an explanation of those placeholders.

Error Number	Error Text <i>Description & Hints for Error Recovery</i>
0000	File open error: <X>. <i>The file <X> could not be read, please check access rights.</i>
0010	The length of identifier <X> exceeds 64 characters (RFC2578 §3.1, §7.1.1, §7.1.4). <i>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.</i>
0050	Encountered lexical error at ... <i>The encountered character is not allowed in a SMI MIB module.</i>
1000	Syntax error: Encountered “ <i>token1</i> ” at row <i>r</i> , column <i>c</i> , expected one of the following: ... <i>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).</i>
1001	The DISPLAY-HINT clause value “ <i>token1</i> ” at row <i>r</i> , column <i>c</i> is invalid (RFC2579 §3.1) The DISPLAY-HINT clause does not correspond to any of the allowed formats for INTEGER or OCTET STRING base types.

- 1002 The UTC time value “*token1*” at row *r*, column *c* does not match the mandatory format YYMMDDhhmmZ or YYYYMMDDhhmmZ (RFC2578 §2)
- The UTC time value does not correspond to the format YYMMDDhhmmZ or YYYYMMDDhhmmZ where
- YY - last two digits of year (only years between 1900-1999)
 - 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)
- 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. Check whether the MIB module name is not misspelled (this is often the case for older RFC 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.*
- 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> (RFC2578 §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 (RFC2578 §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.*
- 1202 The OBJECT-TYPE <X> has inconsistent maximum access (RFC2578 §7.3).
- An object <X> has a MAX-ACCESS or ACCESS clause that does not match its context (RFC2578 §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 (RFC2580 §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 (RFC2580 §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> (RFC2580 §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 (RFC2580 §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 access read-create may be present in the CREATION REQUIRES clause of variation <X> (RFC2580 §6.5.2.4).
- Other objects and columns cannot be created and thus they cannot participate in a row creation.*
- 1500 Unresolved syntax reference <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>.*
- 1501 Unresolved object reference <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 (RFC2578 §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 (RFC2580 §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 with wrong type: <X>, expected type was <Y>, but found <Z> instead.
- The reference to object <X> must be of type <Y> but it is of type <Z>.*
- 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.*
- 1801 The SEQUENCE definition for table entry <X> does not match with the number of child objects of that node.
- All objects registered below a table entry node must be included in the SEQUENCE definition of that table entry.*

- 1810 The OBJECT-TYPE <X> has an invalid index definition (RFC2578 §7.7).
The OBJECT-TYPE <X> has an invalid INDEX clause, i.e., an empty clause.
- 1811 The OBJECT-TYPE <X> has an invalid index definition because <Y> may be negative (RFC2578 §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 RFC2578 §7.7 for more details.
- 1812 The OBJECT-TYPE <X> has an invalid index definition (RFC2578 §7.7) because the minimum total index length exceeds 128 which is the maximum SNMP OID length.
Instances of this OBJECT-TYPE <X> can never be accessed through the SNMP protocol, because the identifying OID is longer than 128 sub-identifiers and thus cannot be represented in SNMP.
- 1813 The OBJECT-TYPE <X> has an invalid index definition (RFC2578 §7.7) because the sub-index with the IMPLIED length can have a zero length.
Implied variable length sub-index values cannot be represented.
- 1850 The OBJECT-TYPE <X> has invalid index definition, because <Y> is not a columnar object (RFC2578 §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 (RFC2578 §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> (RFC2578 §3.6).
Once an object identifier has been registered it must not be reregistered. An object registration is any object definition other than OBJECT-IDENTIFIER.

2010	<p>Illegal object registration of <X> under <Y> for the object ID <Z>.</p> <p><i>For example, it is not legal to register objects in the sub-tree of an OBJECT-TYPE registration.</i></p>
3000	<p>The default value of OBJECT-TYPE <X> is out of range (RFC2578 §7.9).</p> <p><i>The values specified in a DEFVAL clause have to be valid values for the corresponding data type syntax.</i></p>
3001	<p>The size of the default value of OBJECT-TYPE <X> is out of range (RFC2578 §7.9).</p> <p><i>The length of the specified octet string exceeds the SIZE constraints defined for the corresponding data type syntax.</i></p>
3002	<p>The format of the default value of OBJECT-TYPE <X> does not match its syntax (RFC2578 §7.9).</p> <p><i>The value <X> is not properly defined for the corresponding syntax.</i></p>
3003	<p>A DEFVAL clause is not allowed for OBJECT-TYPE <X> which has a base syntax of Counter (Counter32 or Counter64) (RFC2578 §7.9).</p>
4000	<p>The syntax definition of the object <X> is not a valid refinement of its base syntax (RFC2578 §9).</p> <p><i>A refinement must not extend the range of valid values for a data type.</i></p>
4010	<p>The range restriction is invalid because ...</p> <p><i>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.</i></p>
4100	<p>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) (RFC2579 §3.1)</p> <p>Only textual conventions for INTEGER and OCTET STRING base types may have a DISPLAY-HINT clause.</p>

- 4101 The DISPLAY-HINT clause value “*token1*” of the TEXTUAL-CONVENTION definition <X> is not compatible with the used SYNTAX (RFC2579 §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 (RFC2580 §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 (RFC2580 §3.1).
- Only accessible objects and notifications may be included in object groups.*
- 5101 The OBJECTS clause of NOTIFICATION-TYPE <X> must not reference OBJECT-TYPE <Y> which has a MAX-ACCESS clause of 'not-accessible' (RFC2578 §8.1).
- It is impossible for an agent to implement View Access Control Model (VACM) correctly and sending an object which has a maximum access of 'not-accessible'.*
- 6000 The PIB-INDEX clause of OBJECT-TYPE definition <X> does not reference a columnar object with an 'InstanceId' syntax (RFC3159 §7.5)
- 6001 The PIB-TAG clause present in <x> must be absent because the SYNTAX is not 'TagReferenceId' (RFC3159 §7.11)
- 6002 The PIB-REFERENCES clause present in <x> must be absent because the SYNTAX is not 'ReferenceId' (RFC3159 §7.10)
- 6003 A PIB-TAG clause must be present in <x> because its SYNTAX is 'TagReferenceId' (RFC3159 §7.11)
- 6004 The PIB-REFERENCES must be present in <x> because its SYNTAX is 'ReferenceId' (RFC3159 §7.10)
- 6005 The UNIQUENESS clause of OBJECT-TYPE definition <x> must not contain the attribute <y> referenced in the PIB-INDEX clause (RFC3159 §7.9)

6006	The UNIQUENESS clause of OBJECT-TYPE definition <X> must not contain the attribute <Y> more than once (RFC3159 §7.9)
6007	The INSTALL-ERRORS clause of OBJECT-TYPE definition <X> has an invalid error number <N> for label <L> which is out of the range 0-65535 (RFC3159 §7.4)

Table 6: MIB compiler error messages and descriptions.

15 Regular Expression Syntax

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'll 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:

- | | |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| "." | (Dot.) In the default mode, this matches any character except a newline. If the <code>DOTALL</code> flag has been specified, this matches any character including a newline. |
| "^" | (Caret.) Matches the start of the string, and in <code>MULTILINE</code> mode also matches immediately after each newline. |
| "\$" | Matches the end of the string, and in <code>MULTILINE</code> mode also matches before a newline. <code>f^{oo}</code> matches both 'foo' and 'foobar', while the regular expression <code>f^{oo}\$</code> matches only 'foo'. |
| "*" | Causes the resulting RE to match 0 or more repetitions of the preceding RE, as many repetitions as are possible. <code>ab[*]</code> will match 'a', 'ab', or 'a' followed by any number of 'b's. |
| "+" | Causes the resulting RE to match 1 or more repetitions of the preceding RE. <code>ab⁺</code> 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. <code>ab[?]</code> will match either 'a' or 'ab'. |

?, +?, ??	The "", "+", and "?" qualifiers are all <i>greedy</i> ; they match as much text as possible. Sometimes this behaviour isn't 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 <i>non-greedy</i> or <i>minimal</i> fashion; as <i>few</i> characters as possible will be matched. Using . *? in the previous expression will match only '<H1>'.
{m, n}	Causes the resulting RE to match from <i>m</i> to <i>n</i> 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 <i>n</i> specifies an infinite upper bound; you can't omit <i>m</i> .
{m, n}?	Causes the resulting RE to match from <i>m</i> to <i>n</i> repetitions of the preceding RE, attempting to match as <i>few</i> 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.
[]	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 <i>complementing</i> 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: `[()]`.
- (?...) 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.
- (?=...) Matches if ... matches next, but doesn't consume any of the string. This is called a lookahead 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 lookahead assertion. For example, `Isaac (?!Asimov)` will match 'Isaac ' only if it's *not* followed by 'Asimov'.

The special sequences consist of "\ " and a character from the list below. If the ordinary character is not on the list, then the resulting RE will match the second character. For example, `\$` matches the character "\$".

<code>\number</code>	Matches the contents of the group of the same number. Groups are numbered starting from 1. For example, <code>(.+)\1</code> matches 'the the' or '55 55', but not 'the end' (note the space after the group). This special sequence can only be used to match one of the first 99 groups. If the first digit of <i>number</i> is 0, or <i>number</i> is 3 octal digits long, it will not be interpreted as a group match, but as the character with octal value <i>number</i> . Inside the "[" and "]" of a character class, all numeric escapes are treated as characters.
<code>\A</code>	Matches only at the start of the string.
<code>\b</code>	Matches the empty string, but only at the beginning or end of a word. A word is defined as a sequence of alphanumeric characters, so the end of a word is indicated by whitespace or a non-alphanumeric character. Inside a character range, <code>\b</code> represents the backspace character.
<code>\B</code>	Matches the empty string, but only when it is <i>not</i> at the beginning or end of a word.
<code>\d</code>	Matches any decimal digit; this is equivalent to the set <code>[0-9]</code> .
<code>\D</code>	Matches any non-digit character; this is equivalent to the set <code>[^0-9]</code> .
<code>\s</code>	Matches any whitespace character; this is equivalent to the set <code>[\t\n\r\f\v]</code> .
<code>\S</code>	Matches any non-whitespace character; this is equivalent to the set <code>[^\t\n\r\f\v]</code> .
<code>\w</code>	Matches any alphanumeric character; this is equivalent to the set <code>[a-zA-Z0-9_]</code> .
<code>\W</code>	Matches any non-alphanumeric character; this is equivalent to the set <code>[^a-zA-Z0-9_]</code> .
<code>\Z</code>	Matches only at the end of the string.
<code>\\</code>	Matches a literal backslash.

