ModSnmp Manual v3.14

Table of Contents

Introduction 3
What is ModSnmp? 3
Running ModSnmp 4
The GUI 4
Quick setup guide 5
Saving and restoring the settings 6
SNMP Interfaces 7
Defining SNMP Interfaces 7
Editing SNMP Interfaces 7
Deleting SNMP Interfaces 7
SNMP Users 8
Defining SNMP Users 8
Deleting SNMP Users 8
Engine ID 8
Modbus Interfaces 10
Defining Modbus Interfaces 10
Editing Modbus Interfaces 12
Deleting Modbus Interfaces 12
Device Types 13
Adding Device Types 13
Editing Device Types 13
Deleting Device Types 14
Saving and loading device types 14
32/64-bit Value Settings 14
Object Types 16
Modbus Variables 16
SNMP Objects 16
Defining Object Types 17
Editing Object Types 19
Devices 20
Defining devices 20
Editing devices 20
Deleting Devices 20
Reordering Devices 20
Constant OID Settings 21
Defining Constant OIDs 21
Editing Constant OIDs 22
Deleting Constant OIDs 22
Modbus server 23
MIB File 24
MIB Settings 24
Logging 25
Tracing 26
Modbus tracing 26
SNMP tracing 26
Running as a server 27
Command line options 27
Running under Windows 27
Running under Unix/Linux 28
Changing server settings 29
A. Troubleshooting 31

SNMP 31
Modbus 31
General Communications 31
Socket Communications 34
Serial Communications 35
RTU Packet Type 36
ASCII Packet Type 36
TCP Packet Type 37
Miscellaneous 37
B. Release Notes 38
C. Modbus protocol documents 41
D. SNMP documents 41
E. Acknowledgments 42
SNMP4J 42
Apache log4j 42

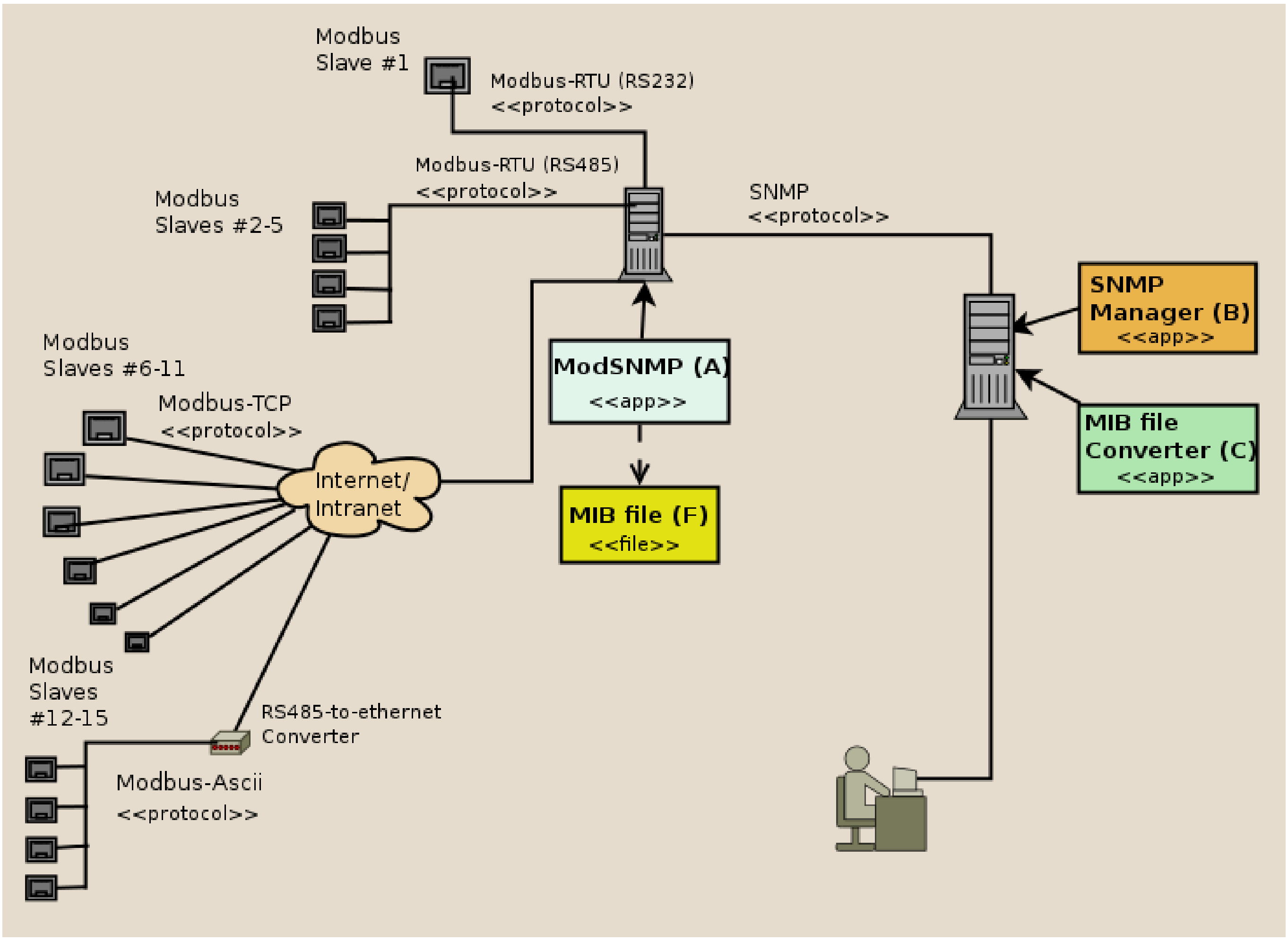
Introduction

Note: This manual is also available on the Wingpath website in HTML and PDF formats.

What is ModSnmp?

ModSnmp is an SNMP to Modbus bridge - that is, it converts between SNMP (Simple Network Management Protocol) and the Modbus protocol.

ModSnmp acts as an SNMP agent [1], handling requests from SNMP managers [2]. It handles an SNMP request by sending a corresponding Modbus request (or requests) to a Modbus device (or devices) [3] and using their responses to construct an SNMP response. Note that ModSnmp acts as a Modbus client [4] when it communicates with Modbus devices.



Configuration example

In this configuration example, the SNMP manager (B) and ModSnmp (A) are on 2 independent machines, but they might equally well be on the same machine if the location of devices and their connections allow it. ModSnmp produces an MIB script file (F) containing the OIDs needed by the SNMP manager. This can be imported to the manager directly or by a 'converter' application (C). On the left side of the figure ModSnmp communicates with local Modbus devices over RS232 (Slave #1), RS485 (Slaves #2-5), and with more remote devices over the internet or an intranet (Slaves #6-11, #12-15).

[1] An SNMP agent may also be called an "SNMP command responder". In normal network terminology, it would be called an "SNMP server".
[2] An SNMP manager may also be called a "Network Management Station" (NMS) or an "SNMP command generator". In normal network terminology, it would be called a "SNMP client".
[3] A Modbus device is traditionally called a "Modbus slave". In normal network terminology, it would be called a "Modbus server".
[4] A Modbus client is traditionally called a "Modbus master".

In addition to being a SNMP-Modbus bridge, ModSnmp can also act as a Modbus/TCP server. This enables Modbus clients, as well as SNMP managers, to access the Modbus devices via ModSnmp.

Running ModSnmp

ModSnmp can be run with a GUI (Graphical User Interface), or without a GUI as a server process. It is normally run as a server process (see Running as a server), but must be initially run with a GUI in order to configure various settings.

To run ModSnmp with a GUI, use the command:

```
java -jar modsnmp3.14.jar
```

You will need to use the full pathname for the `modsnmp3.14.jar` file if it's not in your current directory.

To run ModSnmp under Windows, you can also use the Windows Run dialog or double-click on the icon or filename.

To exit from ModSnmp, select the **File** → **Exit** menu item or click the close button of the main window.

The GUI

When you start ModSnmp the main window is displayed.

At the top of the window are the usual menu bar and tool bar.

On the left side of the window there is a side panel showing a list of pages. When you select an item in this list, the page is displayed in the main panel on the right side of the window.

At the bottom of the window is a status bar, which is used to display error and information messages.

Side panel

The pages listed in the side panel are in eight groups:

- SNMP Interfaces: used to configure how the SNMP agent component of ModSnmp communicates with SNMP managers - see SNMP Interfaces. You only need to change these settings if you want ModSnmp to use non-standard or additional interfaces.
- SNMP Users: used to configure the SNMP users that are allowed access - see SNMP Users.
- Modbus Interfaces: used to configure how the Modbus client component of ModSnmp communicates with Modbus devices - see Modbus Interfaces.
- Device types: used to configure the conversion between the SNMP and Modbus protocols for each Modbus device type - see Device Types.
- Devices: used to configure the type and interface of each Modbus device - see Devices.
- Constant OIDs: used to configure fixed responses to SNMP requests - see Constant OID Settings.
- MIB Settings: used to configure the generation of the MIB file required by SNMP managers - see MIB File.
- Logging: used to configure the logging of error and information messages. - see Logging.

Entries in the list of pages are colour-coded to indicate the state of the corresponding page:

- Blue/grey: The page is selected (i.e. displayed in the main panel).
- Yellow: The page has unapplied changes. The changes themselves are also highlighted in yellow within the page.
- Red: The page has an error message in its status bar.

Main panel

Each page that may be displayed in the main panel has buttons for various actions. These typically include an **Apply** button (which saves the changes you have made), a **Reset** button (which restores the settings to their last saved state), and a **Help** button (which provides help on the page - you can also get help by pressing the F1 key).

Each page also has a status bar below its buttons, which is used to display error and information messages that are specific to the page.

Status bars

At the bottom of the main window is a "status bar", which is used to display error and information messages. Each page displayed in the main panel also has its own status bar, which is used to display messages that are specific to that page.

Many of the error messages displayed in the status bars have an **Error Help** button, which provides help on the error message (you can also get this help by pressing the F4 key).

Most of the messages that are displayed in the status bars are removed automatically when they are no longer applicable. This is not possible for some messages, and a **Clear** button is provided for these so that they can be manually removed.

Entering data

You can select a data field in which to enter data either by clicking on it with the mouse or by moving to it using the Tab or Shift-Tab key.

When you select a field, all the text within the field is initially selected, so what you type will replace the original value. If you want to edit the value, rather than replace it, click on the value again or press the right-arrow or left-arrow key.

If you place the mouse cursor over a data field (or its label), a tooltip is displayed, which provides a brief description of the field's purpose. You can also display the tooltip of the currently selected data field by typing Ctrl-F1. The display of tooltips can be disabled or enabled using the **View** → **Tool tips** menu option or by typing Shift-F1. For a more detailed description of a field's purpose, click the **Help** button or by press the F1 key.

If you start entering or editing a field value and change your mind, you can restore the original value by pressing the Esc key. To restore the values of *all* the fields in a page, click the **Reset** button.

When you have finished entering or editing a field value, you can click on another field or a button, or press Tab or Enter to move to the next field.

Quick setup guide

To configure ModSnmp, you must perform at least the following steps:

- **Modbus Interface.** Configure the Modbus interface(s) to be used to communicate with the Modbus device(s) (see Modbus Interfaces).
- **Device Type.** Configure the type(s) of Modbus device to be used (see Device Types and Object Types).
- **Device.** Define the Modbus device(s) to be used (see Devices).
- **MIB File.** Generate an MIB file and load it into the SNMP manager (see MIB File).

You may also want to change some of the following settings:

- **SNMP User.** As is conventional for SNMP agents, ModSnmp is configured by default to use 'public' as the user-name/community-string that it accepts in requests from SNMP managers. For security reasons, you should delete the 'public' user and use less well-known user names. (see SNMP Users).

- **Logging.** You will probably want to send log output to a file instead of the terminal (see Logging).
- **Tracing.** Once you have dealt with any configuration problems, you may want to disable tracing of SNMP and Modbus messages (see Tracing);
- **SNMP Interface.** ModSnmp can receive SNMP requests on alternative or additional interfaces (see SNMP Interfaces).
- **Constant OIDs.** These may be used to provide fixed system identification to the SNMP manager (see Constant OID Settings).
- **MIB Settings.** You can modify some of the header information in the MIB file (see MIB File). You may need to do this if you are running more than instance of ModSnmp.

Click the **Run** button to start ModSnmp listening for and processing SNMP requests. Click the **Run** button again if you want to stop ModSnmp.

Saving and restoring the settings

ModSnmp's settings can be saved to a disk file by selecting the **File** → **Save Settings** menu item. This displays a dialog that enables you to enter or select the name of the file to save the settings to. ModSnmp's settings are saved to the file in an XML format.

To restore the settings from a file, use the **File** → **Load Settings** menu item. This displays a dialog that enables you to enter or select the name of the file to restore the settings from. If ModSnmp is "running" (i.e. the **Run** button is depressed), you will have to click the **Run** button before you can load the settings.

You can also load the settings from a file when you start ModSnmp, by passing the name of the file as a command-line argument, e.g.:

```
java -jar modsnmp3.14.jar config.xml
```

See Command line options for more information.

There is, currently, no formal DTD (Document Type Definition) or XML schema for the format of the XML files in which the settings are saved. However, the format should be fairly evident to a human reader, and can easily be edited, if necessary, using a text editor.

SNMP Interfaces

The SNMP Interfaces pages enable you to configure the interfaces on which ModSnmp listens for SNMP requests.

By default, ModSnmp will listen on the standard SNMP interface (port 161/UDP). You only need to change these settings if you want ModSnmp to use non-standard or additional interfaces.

Details of all the SNMP interfaces that you have added are displayed in the SNMP Interfaces page. See Defining SNMP Interfaces for explanations of the values displayed in each column.

Defining SNMP Interfaces

The Add SNMP Interface page enables you to define SNMP interfaces. To add an SNMP interface, enter the following data and then click **Apply**.

- **Type.** **UDP** is normally used for SNMP, but it will work over **TCP** if that is what your SNMP manager uses.
- **Host.** You would normally leave this field empty, which allows requests to be accepted via any network interface. But if you enter a host name or IP address of a network interface on the machine running ModSnmp, then ModSnmp will only accept requests via that network interface. For example, if you enter **`localhost`** (or **`127.0.0.1`**) then ModSnmp will only accept requests from processes running on the same machine.
- **Port.** Enter the port that ModSnmp should listen on for requests. The standard SNMP port number is 161, but this is a "privileged" port and you will not be able to listen on this port on a Unix/Linux machine unless you are "superuser". This is not normally advisable, so it's better for testing purposes to use a non-privileged port (above 1023).

Editing SNMP Interfaces

An SNMP Interface definition can be edited using the Edit SNMP Interface page.

Use the **Select Interface** drop-down list to select the SNMP interface to be edited, change the values as required, and click **Apply** to save the changes.

Deleting SNMP Interfaces

SNMP interfaces can be deleted using the **SNMP Interfaces** page:

- To delete a SNMP interface, select the SNMP interface and then click the **Delete** button.
- To delete several SNMP interfaces, select them by clicking on them while holding down the control key, and then click the **Delete** button.
- To delete *all* SNMP interfaces, click the **Delete All** button.

At least one SNMP interface must be defined in order to run ModSnmp.

SNMP Users

The SNMP Users pages allow you to configure SNMP users and the SNMP engine ID.

Details of all defined users are displayed in the SNMP Users page.

As is conventional for SNMP agents, ModSnmp is configured by default to use 'public' as the user-name/community-string that it accepts in requests from SNMP managers. For security reasons, you should delete the 'public' user and use less well-known user names. If possible, use SNMPv3 and use an authentication protocol for all users.

Defining SNMP Users

The Add User page enables you to define the users that are allowed SNMP access.

SNMPv3 uses the User-based Security Model (USM) to authenticate users, and optionally encrypt the data in SNMP messages.

SNMPv1 and SNMPv2c use "community strings" instead of users, and do not support authentication or encryption. If you define a user without authentication, then ModSnmp will allow that user's name to be used as a community string in v1 and v2c messages.

To add an SNMP user, enter the following data and then click **Apply**:

- **Name.** Enter the name of the user (for SNMPv3), or the community string (for SNMPv2c or SNMPv1).
- **Authentication Protocol.** Select the authentication protocol to be used. Select **None** if the **Name** is to be used as a community string.
- **Authentication Passphrase.** Enter the authentication passphrase. The passphrase is used to generate an authentication key, and is then discarded.
- **Privacy Protocol.** Select the method to be used to encrypt SNMP messages.
- **Privacy Passphrase.** Enter the privacy passphrase. The passphrase is used to generate an encryption key, and is then discarded.
- **Authorization.** Select the **Allow Write** checkbox if this user is to be allowed to write to Modbus registers.

Note that it is a requirement of the SNMP standard that, for security reasons, passphrases are not stored. If you want to change a passphrase, you should delete and re-add the user.

Deleting SNMP Users

Users may be deleted using the SNMP Users page.

- To delete a user, select the user and the click the **Delete** button.
- To delete several users, select them by clicking on them while holding down the control key, and then click the **Delete** button.
- To delete *all* users, click the **Delete All** button.

Engine ID

The **Engine ID** in the SNMP page is used in SNMPv3 to uniquely identify the SNMP agent. It is also known as the "Agent ID". The default value is automatically generated from the IP address of the machine that ModSnmp is running on. Do not change the engine ID unless you know what you are doing.

To change the engine ID, enter the new ID in the **Engine ID** field as hex bytes and click **Apply**.

WARNING: The engine ID is used in the generation of authentication and encryption keys. If you change the engine ID, you will have to re-enter all "secure" users, i.e. users that have authentication pass-phrases.

Modbus Interfaces

The Modbus Interface pages allow you to define one or more interfaces for communicating with Modbus devices.

Details of all the Modbus interfaces that you have defined are displayed in the Modbus Interfaces page. See Defining Modbus Interfaces for explanations of the values displayed in each column.

Defining Modbus Interfaces

The Add Modbus Interface page enables you to define Modbus interfaces. To add a Modbus interface, enter the data described in the following sections and then click **Apply**.

General settings

- **Name.** Enter a unique name for the interface. This name will be used in Device definitions to identify the interface to be used.

- **Interface Type.** Select **Serial** for a RS232/485 serial interface. Select **TCP Socket** for a TCP/IP network interface. Select **UDP Socket** for a UDP/IP interface (use of UDP for Modbus is non-standard). Note that the **Serial** option will only be available if you are using Windows or Linux (the **Help** → **About** window tells you whether the serial comms library has been loaded).

- **Packet Type.** You would normally select **RTU** if you are using a serial interface, and **TCP** if you are using a socket interface. You may have to use **ASCII** for some legacy serial interfaces or devices. It is possible to use RTU or ASCII packets over a socket interface - this is non-standard, but can be useful for software testing. It is also possible to use TCP packets over a serial interface, but there is no good reason to do so (and there would be no error-checking of the packets).

- **EOM Timeout.** Enter the maximum delay (in milliseconds) expected within a message. The default value works fine in most situations, but you may need to increase the value if you are using a serial interface or operating system that introduces large delays in the middle of messages.

- **Idle timeout.** Period in seconds after which a connection will be closed if idle. A value of 0 will disable the timeout.

  A connection is considered to be idle if no valid responses have been received from the slave device within the timeout period.

  Setting an idle timeout can be used to avoid holding on to resources that are not currently in use. It can also be useful for recovering from some error conditions.

- **Allow Request Pipelining.** When using the TCP packet type, ModSnmp may send a request to a slave before responses to earlier requests have been received. This is known as "pipelining". The "transaction ID" in the request and response messages allows ModSnmp to match the responses to the corresponding requests.

  Deselect **Allow Request Pipelining** if the slave cannot handle pipelined requests.

The **Interface Type** and **Packet Type** together determine the variant of the Modbus protocol that will be used. The table below lists the possible combinations:

Table 1. Modbus protocol variants

Interface type	Packet type	Protocol variant
Serial	RTU	Modbus RTU
Serial	ASCII	Modbus ASCII
Serial	TCP	Not recommended since no error checking (non-standard)

Interface type	Packet type	Protocol variant
TCP Socket	RTU	Modbus RTU encapsulated in TCP (non-standard)
TCP Socket	ASCII	Modbus ASCII encapsulated in TCP (non-standard)
TCP Socket	TCP	Modbus TCP
UDP Socket	RTU	Modbus RTU encapsulated in UDP (non-standard)
UDP Socket	ASCII	Modbus ASCII encapsulated in UDP (non-standard)
UDP Socket	TCP	Modbus UDP (also known as Modbus TCP encapsulated in UDP) (non-standard)

Response Handling

- **Always responds.** When using Modbus RTU or ASCII over a serial interface, a device should not respond to a request for a different slave ID (this is necessary for RS485 multi-drop to work). When using Modbus TCP over a TCP socket interface, the device/bridge should respond to requests to unknown/unreachable devices with an exception 10 or 11 response. When using non-standard protocol variants (such as RTU encapsulated in TCP), the behaviour in these situations is not defined. You should select **Always responds** if the device behaves in the Modbus TCP manner, or deselect it if it behaves in the serial manner.
- **Number of Retries.** This field specifies how many times ModSnmp should retry a request that it has not received a response to.
- **Response Timeout.** This field specifies how long (in milliseconds) ModSnmp should wait for a response before re-trying the request or reporting a failure.

Socket settings

The following items must be configured for a socket (TCP or UDP) interface:

- **Host.** The host name or IP address of the device(s) (or Modbus bridge/gateway).
- **Port.** The port number that ModSnmp should connect to on the device (or Modbus bridge/gateway).
- **Local host.** You would normally leave this field empty. However, if you are using a computer that has multiple network interfaces, you can enter the host name or IP address of one of the network interfaces to force that interface to be used for the connection to the device.
- **Local port.** You would normally leave this field set to 0, which allows dynamic allocation of the local port to be used for the connection. However, some firewalls or NAT (Network Address Translation) systems may require you to set a specific port.

The same host and port may be used in more than one interface. If this is done, ModSnmp will open a separate socket connection for each interface. This can be useful if you have more than one device on the same host and port, and want to access the devices concurrently.

Serial settings

The following items must be configured for a serial interface:

- **Port.** Select or enter the name of the serial port that ModSnmp should use to talk to the device(s). The same serial port cannot be used in more than one interface.
- **Speed.** Select or enter the speed in bits/second. Note that serial interfaces and drivers vary in the speeds that they support.

- **Parity.** It's usual to use no parity, since the CRC in RTU packets is a good error check on its own, but some devices may require it. The serial specification requires even parity to be the default.
- **Data Bits.** Must be 8 for RTU. Some ASCII devices may require 7 (the serial specification actually specifies 7 for ASCII).
- **Stop Bits.** Usually 1, but some devices may require 2. The serial specification actually specifies 2 stop bits if parity is not being used, but this requirement is normally ignored.
- **RTS Control.** The default setting of **High** will be OK for most purposes.

  Selecting **Flow Control** will enable hardware flow control ("handshaking") using RTS and CTS. Since Modbus messages are short, flow control is not normally necessary.

  Selecting **RS485** will use RTS to control a RS232 to RS485 converter: ModSnmp will raise RTS when it is transmitting data, and lower RTS to receive data. Note that the lowering of RTS may be delayed by the operating system, and this may cause loss of data. For this reason, we recommend the use of RS232-RS485 converters that do not require RTS control.

Editing Modbus Interfaces

A Modbus Interface definition can be edited using the Edit Modbus Interface page.

Use the **Select Interface** drop-down list to select the Modbus interface to be edited, change the values as required, and click **Apply** to save the changes.

Deleting Modbus Interfaces

Modbus interfaces can be deleted using the **Modbus Interfaces** page:

- To delete a Modbus interface, select the Modbus interface and then click the **Delete** button.
- To delete several Modbus interfaces, select them by clicking on them while holding down the control key, and then click the **Delete** button.
- To delete *all* Modbus interfaces, click the **Delete All** button.

A Modbus interface cannot be deleted if it is in use by a device definition.

Device Types

The Device Types pages enable you to define the types of Modbus device that ModSnmp is to communicate with.

Details of all the device types that you have added are displayed in the Device Types page. See Adding Device Types for explanations of the values displayed in each column.

Adding Device Types

The Add Device Type page enables you to define Modbus device types. To add a Modbus device type, enter the following data and then click **Apply**.

- **Name.** Enter a name for the device type. This is used to refer to the device type when adding a device.
- **Description.** Optionally enter a description of the device type. This is displayed in the Device Types page, but is not used by ModSnmp in any other way.
- **32/64-bit Values.** If devices of this type use 32-bit or 64-bit values, select the appropriate checkboxes to indicate how the devices transmit these values. These options are described in 32/64-bit Value Settings.
- **Use Write-Single function.** ModSnmp normally uses function code 15 (Write Multiple Coils) and function code 16 (Write Multiple Registers) to do Modbus writes. If the device does not support function codes 15 and 16, you can select this checkbox to use function code 5 (Write Single Coil) and function code 6 (Write Single Register) instead.
- **Supports Device Identification.** Select this checkbox if the device supports function code 43/14 (Read Device Identification). ModSnmp will then use this function to read identification information from the device and make it available to the SMNP manager.

When you have added the device type, you should use the Edit Device Type page to define the object types used by the device type.

Editing Device Types

A Modbus device type can be edited using the Edit Device Type page. Use the **Select Device Type** drop-down list to select the device type to be edited.

To edit the name, description or 32/64-bit values settings, change the values as required and click **Apply** to save the changes.

ModSnmp uses a table of "object types" (see Object Types) to map SNMP objects to the variables of a Modbus device type. This table is displayed in the Edit Device Type page. You can select which columns are displayed in the object types table using the **View** → **Object Type Columns** menu item.

See Defining Object Types for how to add object types to the table, and Editing Object Types for how to edit object types.

Object types can be deleted using the table in the Edit Device Type page:

- To delete an object type, select it in the table and the click the **Delete** button.
- To delete several object types, select them by clicking on them while holding down the control key, and then click the **Delete** button.
- To delete *all* the object types, click the **Delete All** button.

To move one or more object types in the table, select them and then drag-and-drop them to the required position using the mouse.

To move object types using the keyboard, select them and type Ctrl-X. Use the arrow keys to move to the required position and then type Ctrl-V.

If you are moving more than one object type, the object types must be next to each other in the table.

Deleting Device Types

Devices can be deleted using the **Device Types** page:

- To delete a device type, select the device type and then click the **Delete** button.
- To delete several device types, select them by clicking on them while holding down the control key, and then click the **Delete** button.
- To delete *all* device types, click the **Delete All** button.

A device type cannot be deleted if it is in use by a device definition.

Saving and loading device types

The currently selected device type definition can be saved to a disk file by selecting the **File** → **Save Device Type** menu item. This displays a dialog that enables you to enter or select the name of the file to which the device type should be saved.

To load a device type from a file, use the **File** → **Load Device Type** menu item. This displays a dialog that enables you to enter or select the name of the file from which the device type should be loaded. If ModSnmp is "running" (i.e. the **Run** button is depressed), you will have to click the **Run** button before you can load the device type.

Device types are saved to files in XML format. There is, currently, no formal DTD (Document Type Definition) or XML schema for the format of these XML files. However, the format should be fairly evident to a human reader, and can easily be edited, if necessary, using a text editor.

32/64-bit Value Settings

The official Modbus protocol (in both the original Modbus specification and the current Modbus specification) only allows 1-bit and 16-bit integer values to be transferred. Many manufacturers have extended the protocol to allow 32-bit and 64-bit values, and also to allow floating-point values. Fortunately, everyone seems to have used the IEEE format for floating-point numbers, but that is where the agreement ends. The **32/64-bit Values** group of settings (see Adding Device Types) enables you to configure ModSnmp to handle all implementations of 32/64-bit values that we know of.

- **Little Endian.** Modbus is a "big-endian" protocol: that is, the more significant byte of a 16-bit value is sent before the less significant byte. It seems obvious that 32-bit and 64-bit values should also be transferred using big-endian order. However, some manufacturers have chosen to treat 32-bit and 64-bit values as being composed of 16-bit words, and to transfer the words in little-endian order. For example, the 32-bit value 0x12345678 would be transferred as 0x56 0x78 0x12 0x34. You should select the **Little Endian** checkbox to use this mixed ordering.
- **Word Registers.** Each register in the Modbus protocol holds a single (16-bit) value. The simplest way to extend the protocol to handle 32-bit and 64-bit values is to allow registers that contain these larger values. However, some manufacturers have chosen to keep to the 16-bit register size, and use 2 registers to hold a 32-bit value, and 4 registers to hold a 64-bit value. For example, if a 32-bit value is stored at address 100, then register 100 would hold one half of the value and register 101 would hold the other half of the value. Some devices will actually allow you to access the halves of the value independently; others will only allow accesses that transfer the complete value (thus making address 101 in the example an invalid address).

  You should select the **Word Registers** checkbox to use multiple registers to store large values, and deselect the checkbox to use a single register to hold each value. Note that when you use the **Word Registers** option, ModSnmp will only allow access to complete values.
- **Word Count.** Some Modbus requests (e.g function 3 Read Holding Registers, and function 16 Write Multiple Registers) include a count of how many registers/values are to be transferred. There are three possible interpretations for this count:

- The number of values to be transferred.
- The number of 16-bit words to be transferred.
- The number of registers to be transferred.

  In the official Modbus protocol, these three interpretations are equivalent, since all values and all registers are 16-bit. When 32-bit and 64-bit values and/or registers are allowed, the three interpretations become distinct. However, the third interpretation will be equivalent to one or other of the first two, depending on the **Word Registers** setting, so ModSnmp only deals directly with the first two interpretations. You should select the **Word Count** checkbox if the count is to be interpreted as the number of 16-bit words to be transferred. You should deselect the **Word Count** checkbox if the count is to be interpreted as the number of values to be transferred.

These three options (Little-endian, Word-registers, and Word-count) allow 8 possible variations of 32/64-bit value handling. At least 5 of these variations have actually been used in devices, so you may need to carefully read documentation, or to experiment, to determine which variation a particular device uses. The default settings (all options selected) should work with Modicon/Schneider devices. For Enron/Daniel devices you will probably need to deselect all the options.

Object Types

SNMP and Modbus are both concerned with reading and writing the values of variables, but there are differences in how they do this. SNMP refers to its variables as "objects", and identifies them using OIDs (Object Identifiers), whereas Modbus refers to its variables with names like "Holding Register", and identifies them using addresses. SNMP and Modbus also use different types for their variables.

The **Object Types** pages enable you to set up mappings from the OIDs and types used in SNMP messages to the addresses and types used in Modbus messages.

The SNMP manager that you are using will also need to know about any SNMP objects that you configure here. This is done using an MIB file.

Modbus Variables

The Modbus protocol uses four different types of variable: Holding Registers, Input Registers, Coils and Discrete Inputs.

Each of these variable types has its own address range, using addresses that are integers in the range 0 to 65535.

The manuals for many Modbus devices adopt the confusing convention of adding an offset to Modbus addresses to indicate the type of variable:

Variable type	**Offset**
Coil	1
Discrete Input	10001 or 100001
Input Register	30001 or 300001
Holding Register	40001 or 400001

The manuals for some Modbus devices simply add an offset of 1 to all addresses, so that, for example, "Holding Register 123" would have an address of 122.

ModSnmp requires actual Modbus addresses (as sent in Modbus messages), so you may have to subtract the appropriate offset from the addresses in device manuals.

Holding Registers and Input Registers are normally 16-bit variables, although ModSnmp supports extensions of the Modbus protocol that allow 32-bit and 64-bit variables. Holding Registers may be read or written, whereas Input Registers are read-only.

Coils and Discrete Inputs are 1-bit variables. Coils may be read or written, whereas Discrete Inputs are read-only.

The Modbus protocol places no interpretation on the values of its variables (they are simply bit-patterns to be transported in Modbus messages), although they are typically treated as integers.

SNMP Objects

SNMP has several types of variable, and it *does* place interpretations on the values of the variables. For example, the types **TimeTicks** and **Integer32** are both 32-bits in size, but have different interpretations. The **OctetString** type is an exception - it is specifically provided as an uninterpreted type (although it is often used to represent text).

SNMP uses a single address range for all its objects, independent of their types. The objects are addressed using OIDs (Object Identifiers), which are sequences of integers. For example, the "System Name" object is addressed using the OID "1.3.6.1.2.1.1.5.0".

SNMP distinguishes between "scalar" objects, which are individual variables, and "tabular" objects, which are groups of variables that are elements in tables. The last integer in an OID is called the "instance identifier". For tabular objects, the instance identifier is the index of the table row containing the variables. For scalar objects, the instance identifier is always zero.

Defining Object Types

The Add Object Type page enables you to add an object type to a Device Type.

Each object type maps a contiguous group of variables.

To add an object type, use the **Select Device Type** drop-down list to select the device type to which the object type is to be added. Then enter the data described in the following sections and click **Apply**:

SNMP settings

- **SNMP Name.** Enter a name for the object. This will be used to generate the object descriptor in the MIB file. It must start with a capital letter, contain only letters and digits, and be at most 18 characters long. The name "Description" is reserved for the device description.
- **SNMP Description.** Optionally enter a description of the object. This will be used as the object description in the MIB file. It may contain only displayable ASCII characters (including spaces, tabs, and newlines), and must not contain double quote (").
- **SNMP Number.** Enter the number of variables (SNMP instances) in the group. For a scalar object, enter 1. The number should not exceed the number of variables actually supported by the Modbus device: SNMP managers often "walk" through tables, and this will generate Modbus errors if the number is too large.
- **SNMP Type.** Select the SNMP type to be used in responses to SNMP requests.

  ModSnmp supports the following SNMP types (see RFC 2578 for fuller descriptions):

Table 2. SNMP types

SNMP type	Description
Integer32	An **Integer32** is a signed 32-bit integer (i.e. in the range -2147483648 to 2147483647 decimal). The INTEGER type used in SNMPv1 is identical to **Integer32**.
Unsigned32	An **Unsigned32** is an unsigned 32-bit integer (i.e. in the range 0 to 4294967295 decimal).
Gauge32	A **Gauge32** is also an unsigned 32-bit integer (i.e. in the range 0 to 4294967295 decimal).
Counter32	A **Counter32** is an unsigned 32-bit integer (i.e. in the range 0 to 4294967295 decimal) that monotonically increases until it reaches a maximum value of 4294967295, when it wraps around and starts increasing again from zero. A **Counter32** is read-only.
TimeTicks	A **TimeTicks** is an unsigned 32-bit integer (i.e. in the range 0 to 4294967295 decimal) that represents a time interval in hundredths of a second.
Counter64	A **Counter64** is an unsigned 64-bit integer (i.e. in the range 0 to 18446744073709551615 decimal) that monotonically increases until it reaches a maximum value of 18446744073709551615, when it wraps around and starts increasing again from zero. A **Counter64** is read-only.
OctetString	An **OctetString** is arbitrary binary or textual data. If you select **OctetString**, the Modbus value will be sent in the

SNMP type	Description
	SNMP response as a sequence of bytes, most significant byte first.
Float	This is not a genuine SNMP type. It is provided by ModSnmp as a means of transporting floating-point values via SNMP (SNMP does not currently have a floating-point type). If you select **Float**, ModSnmp will interpret the Modbus value as floating-point, convert it to a UTF-8 text string, and return it to the SNMP manager as an **OctetString**.
BITS	This is an **OctetString** with names for the individual bits of the value.

SNMP does not have 16-bit types, so you will normally have to select one of the 32-bit types for transferring values of 16-bit Holding or Input registers.

You can use any type except **Float** for transferring Coils and Discrete Inputs. You should set the **Modbus Size** to the number of Coils or Discrete Inputs that you want to transfer in each SNMP instance.

- **SNMP Bit Names.** If you selected the SNMP **BITS** type, you must enter a name for each bit of the value. Names may contain only letters and digits and must start with a lower-case letter. Each name may contain at most 64 characters.

Modbus settings

- **Modbus Type.** Select the Modbus type.
- **Modbus Size.** If you have selected **Holding Register** or **Input Register** as the **Modbus Type**, you should select the register size in bits (16, 32 or 64). If you selected **Float** as the **SNMP Type**, you must select 32 or 64. Note that 32-bit and 64-bit register sizes are non-standard.

  If you have selected **Coils** or **Discrete Inputs** as the **Modbus Type**, you should select or enter the number of Coils or Discrete Inputs that you want to transfer in each SNMP instance.
- **Modbus Address.** Enter the Modbus address of the first variable in the group. Subsequent variables in the group will have consecutive addresses, unless you are using 32-bit or 64-bit Modbus registers and have selected Word Registers in the **Device Type** page.

  Note that you may have to adjust the addresses published in device manuals - see Modbus Variables.
- **Modbus Offset** and **Scale.** These allow you to scale Modbus Holding Register and Input Register values when converting them to and from SNMP values. The scaling uses the formulae:

  SNMP_value = (Modbus_value + offset) / scale

  Modbus_value = SNMP_value * scale - offset

Cache settings

When ModSnmp reads a value from the Modbus device, it saves it in a cache. If the value is requested again (by an SNMP manager or a Modbus client) it can be returned from the cache instead of having to be fetched from the Modbus device again.

- **Lifetime.** This specifies how long (in seconds) the value should be retained in the cache. If you set the lifetime to zero, the value will always be fetched from the device when it is requested.
- **Poll.** If **Poll** is selected, ModSnmp will read the value from the device whenever its lifetime is due to expire. The value should then be available immediately when the SNMP manager requests it.

If polling is selected, the lifetime should be large enough to avoid excessive requests being sent to the device.

- **Cache Writes.** When a manager requests a value to be written to the device, ModSnmp will normally retain the value in the cache if the write is successful. The value should then be available immediately when the manager requests it, without having to read it from the device.

  Some devices have registers where the value read from the register is not the same as the value written to it. **Cache Writes** should be deselected for such registers.

Access settings

- **Writable.** Deselect the **Writable** checkbox to prevent SNMP managers writing to the Modbus variables.

Editing Object Types

An Object Type definition can be edited using the Edit Object Type page.

Use the **Select Device Type** drop-down list to select the device type containing the object type to be edited, and use the **Select Object Type** drop-down list to select the object type. Change the values as required, and click **Apply** to save the changes.

See Editing Device Types for how to delete object types.

See Editing Device Types for how to reorder object types.

Devices

The Devices pages enable you to configure the Modbus devices that ModSnmp is to communicate with.

Details of all the Modbus devices that you have configured are displayed in the Devices page. See Defining devices for explanations of the values displayed in each column.

Defining devices

The Add Device page enables you to define Modbus devices. To add a Modbus device, enter the following data and then click **Apply**.

- **Name.** Enter a name for the device. This will be used to generate the descriptors for the device and its objects in the MIB file. It must start with a capital letter, contain only letters and digits, and be at most 18 characters long.
- **Description.** Optionally enter a description of the device. This will be made available to the SNMP manager as the first object for the device. It may contain any UTF-8 characters (although some SNMP managers may only display the ASCII subset correctly).
- **Device Type.** Select the Device Type that describes this device.
- **Interface.** Select the Modbus interface to be used to communicate with the device.
- **Slave ID.** Enter the Modbus slave identifier. The **Slave ID** is what is called the "slave address" in the original Modbus specification, and is called the "unit identifier" in the Modbus/TCP specification. It should be in the range 1 to 255. The serial specification limits the range of slave identifiers to 1 to 247, for no apparent reason. The Modbus/TCP implementation guide suggests that Modbus/TCP devices that can be uniquely identified by their IP address should use slave identifier 255.

Editing devices

A Modbus device definition can be edited using the Edit Device page.

Use the **Select Device** drop-down list to select the Modbus device to be edited, change the values as required, and click **Apply** to save the changes.

Deleting Devices

Devices can be deleted using the **Devices** page:

- To delete a device, select the device and then click the **Delete** button.
- To delete several devices, select them by clicking on them while holding down the control key, and then click the **Delete** button.
- To delete *all* devices, click the **Delete All** button.

Reordering Devices

Device definitions can be reordered in the **Devices** page.

To move one or more device definitions, select them and then drag-and-drop them to the required position using the mouse.

To move device definitions using the keyboard, select them and type Ctrl-X. Use the arrow keys to move to the required position and then type Ctrl-V.

If you are moving more than one device definition, the definitions must be next to each other in the table.

Constant OID Settings

The **Constant OIDs** pages enable you to map specific SNMP OIDs to constant values. This is particularly useful for providing identification information to SNMP managers, but may have other uses. A typical use would to provide values for sysName (OID 1.3.6.1.2.1.1.5) and sysContact (OID 1.3.6.1.2.1.1.4) - see RFC 3418.

The Constant OIDs are not included in the generated MIB file, since they are assumed to be already defined in other MIBs (e.g. sysName is defined in the standard SNMPv2-MIB file).

Details of all the constant-valued OIDs that you have added are displayed in the Constant OIDs page. See Defining Constant OIDs for explanations of the values displayed in each column.

Defining Constant OIDs

The Add Constant OID page enables you to define OIDs with constant values.

To add a constant-valued OID, enter the following data and then click **Apply**:

- **Description.** Optionally enter a description of the OID. This is displayed in the Constant OIDs page, but is not used by ModSnmp in any other way.
- **OID.** Enter the OID to be mapped. This should be the complete OID, including the instance identifier on the end. For example, you would enter "1.3.6.1.2.1.1.5.0" for the 'sysName' OID.
- **Type.** Select the SNMP type to be used when sending the value to SNMP managers.

  ModSnmp supports the following SNMP types for constants (see RFC 2578 for fuller descriptions):

  ### Table 3. SNMP types

SNMP type	Description
Integer32	An **Integer32** is a signed 32-bit integer (i.e. in the range -2147483648 to 2147483647 decimal). The INTEGER type used in SNMPv1 is identical to **Integer32**.
Unsigned32	An **Unsigned32** is an unsigned 32-bit integer (i.e. in the range 0 to 4294967295 decimal).
Gauge32	A **Gauge32** is also an unsigned 32-bit integer (i.e. in the range 0 to 4294967295 decimal).
Counter32	A **Counter32** is an unsigned 32-bit integer (i.e. in the range 0 to 4294967295 decimal) that monotonically increases until it reaches a maximum value of 4294967295, when it wraps around and starts increasing again from zero. A **Counter32** is read-only.
TimeTicks	A **TimeTicks** is an unsigned 32-bit integer (i.e. in the range 0 to 4294967295 decimal) that represents a time interval in hundredths of a second.
Counter64	A **Counter64** is an unsigned 64-bit integer (i.e. in the range 0 to 18446744073709551615 decimal) that monotonically increases until it reaches a maximum value of 18446744073709551615, when it wraps around and starts increasing again from zero. A **Counter64** is read-only.
OctetString	An **OctetString** is arbitrary binary or textual data. If you select **OctetString**, the value will be sent in the SNMP response as UTF-8 text. You may need to restrict the

SNMP type	Description
	text to the ASCII subset of UTF-8 for it to be displayed correctly by the SNMP manager.
OID	The **OID** type represents administratively assigned names. Any instance of this type may have at most 128 sub-identifiers. Further, each sub-identifier must not exceed the value 2^32-1 (4294967295 decimal). The value should be entered in dotted-decimal form (e.g. "1.3.6.1.4.1.34171.1.35").
IpAddress	The **IpAddress** type represents a 32-bit internet address. The value should be entered in dotted-decimal form (e.g. "192.168.0.4").

- **Value.** Enter the value to be returned to SNMP managers.

Editing Constant OIDs

A constant OID definition can be edited using the Edit Constant OID Entry page.

Use the **Select Entry** drop-down list to select the constant OID entry to be edited, change the values as required, and click **Apply** to save the changes.

Deleting Constant OIDs

Constant OIDs can be deleted using the **Constant OIDs** page:

- To delete a constant OID, select the constant OID and then click the **Delete** button.
- To delete several constant OIDs, select them by clicking on them while holding down the control key, and then click the **Delete** button.
- To delete *all* constant OIDs, click the **Delete All** button.

Modbus server

In addition to its normal use as a SNMP-Modbus converter, ModSnmp can simultaneously operate as a Modbus bridge. This means that it can act as a Modbus TCP server/slave, receiving requests from Modbus clients/masters and responding to these requests with data obtained from the Modbus devices.

Ordinary Modbus read and write requests (function codes 1, 2, 3, 4, 5, 6, 15 and 16) will cache data when possible. Other Modbus requests will simply be forwarded to the device, and the device's response will be returned to the Modbus client/master.

The slave IDs (unit identifiers) used by the Modbus server to identify the devices are not the same as the slave IDs configured in the devices themselves (since multiple devices may have the same slave ID if they are on different interfaces). The Modbus client/master should use slave ID 1 to refer to the first configured device, slave ID 2 to refer to the second configured device, and so on. If more than 254 devices have been configured, only the first 254 will be accessible via the Modbus server.

The Modbus Server page enables you to configure the Modbus server:

- **Enabled.** Select this to allow Modbus clients/masters to connect to the Modbus server.
- **Host.** You would normally leave this field empty, which allows Modbus/TCP connections to be accepted via any network interface. But if you enter a host name or IP address of a network interface on the machine running ModSnmp, then ModSnmp will only accept connections that network interface. For example, if you enter **`localhost`** (or **`127.0.0.1`**) then ModSnmp will only accept connections from processes running on the same machine.
- **Port.** Enter the port that ModSnmp should listen on for Modbus/TCP connections. The standard Modbus port number is 502, but this is a "privileged" port and you will not be able to listen on this port on a Unix/Linux machine unless you are "superuser". This is not normally advisable, so it's better for testing purposes to use a non-privileged port (above 1023).
- **EOM Timeout.** The maximum delay (in milliseconds) expected within a message. The default value works fine in most situations, but you may need to increase the value if you are using a connection that introduces large delays in the middle of messages.
- **Idle timeout.** Period in seconds after which a connection will be closed if idle. A value of 0 will disable the timeout.

  A connection is considered to be idle if no requests from the client/master have been responded to within the timeout period.

  Setting an idle timeout can be used to avoid holding on to resources that are not currently in use.

MIB File

In order to communicate with an SNMP agent such as ModSnmp, an SNMP manager will normally need to know what SNMP objects are supported by the agent. This information is usually provided to the manager as an MIB (Management Information Base) file, which lists the OID of each object, its type, and a textual name that can be used instead of the OID for referring to the object.

When you have configured ModSnmp with the devices in your system, you can generate an MIB file using the **File → Generate MIB file** menu item. This displays a dialog that enables you to enter or select the name of the file to which the MIB should be written.

The generated MIB file should be loaded into your SNMP manager, together with the `Wingpath-MIB` file, which can be downloaded from http://wingpath.co.uk/modbus/resource.php?product=modsnmp. Most SNMP managers can directly load MIB files, but some (e.g. PRTG). require the use of a separate compiler or conversion program.

MIB files are text files that can viewed using a text editor. Their format is specified in RFC 2578, and RFC 4181 contains guidelines for their contents.

The OIDs generated by ModSnmp are provided as comments in the MIB file, so you can easily copy them if you have to manually enter them into an SNMP manager.

MIB Settings

The MIB Settings page enables you to configure the header information written to the MIB file. You only need to modify these settings if you are running more than one instance of ModSnmp.

The following MIB settings can be modified (see RFC 2578 for more details):

- **Module Name.** Enter a name for the module. This must start with a capital letter, be at most 18 characters long, and contain only letters, digits and hyphens. ModSnmp will automatically add the suffix "-MIB" to the name. See RFC 4181 Appendix C for recommended naming conventions.

  If you will be running multiple instances of ModSnmp and their configurations are different, you should use a different module name for each instance.

- **Module Number.** Enter a number for the module.

  If you will be running multiple instances of ModSnmp and their configurations are different, enter a different number for each instance. This ensures that each instance will use different OIDs.

- **Organization.** Enter the name of the organization responsible for the MIB file. It may contain only displayable ASCII characters (including spaces, tabs, and newlines), and must not contain double quote (").

- **Contact.** Enter contact details of the person responsible for handling technical queries about the module. It may contain only displayable ASCII characters (including spaces, tabs, and newlines), and must not contain double quote (").

- **Description.** Enter a description of the module. It may contain only displayable ASCII characters (including spaces, tabs, and newlines), and must not contain double quote (").

Logging

The Logging page is used to configure the logging of error and information messages.

The messages generated by ModSnmp are categorized into levels:

- **Fatal**: Messages about errors that are so serious that ModSnmp cannot continue running.
- **Error**: Messages about errors that may be recoverable, i.e. ModSnmp will attempt to continue running.
- **Warn**: Messages about potential problems.
- **Info**: Messages about the normal operation of ModSnmp.
- **Trace**: Tracing of Modbus and SNMP messages.

ModSnmp can log messages to four different destinations: **Terminal**, **File**, **Syslog** and **Windows**. You may log messages to more than one destination, with a different level for each destination.

For each destination, you should select the level of messages to be logged to that destination. When you select a particular level, messages of that level and higher levels will be logged. For example, if you select **Warn** then messages from the levels **Warn**, **Error** and **Fatal** will be logged. Select **None** if no messages are to be logged to that destination.

The destinations are:

- **Terminal**: This is a window, which is displayed automatically when you start ModSnmp with a GUI. If you close the logging window, you can re-open it using the **View**→**Log** menu option.

  By default, this destination is set to level **Trace** to help with initial configuration of ModSnmp.

- **File**: This is a text file. You will have to enter the name of the file in the **File Name** field, or use the **Browse** button to open a dialog to browse for the file.

  ModSnmp will append messages to the file, so output from previous runs of the program will not get lost.

  You may limit the size of the log file by entering a non-zero value in the **Maximum file size (MB)** field. Enter 0 if you do not want to limit the file size. If the log file grows bigger than the size limit, ModSnmp will rename the log file by adding the suffix ".bak", and start a new log file with the specified name.

- **Syslog**: This is the system logger available on Unix/Linux systems.

  ModSnmp sends messages to the system logger via UDP port 514, which may not be enabled by default. If you are using 'syslogd' for system logging, you can enable UDP port 514 by starting 'syslogd' with a '-r' option. If you are using 'rsyslogd', you can enable UDP port 514 by including the lines:

  ```
      $ModLoad imudp
      $UDPServerRun 514
  ```

  in the file '/etc/rsyslogd.conf'.

  ModSnmp uses the '`user`' message category ("facility") when sending messages to the system logger.

- **Windows**: This is the event logger available on Windows systems.

Messages of level **Info** and above are also displayed in ModSnmp's status bar when running with a GUI. If you want to run ModSnmp without a GUI, you must log error messages to at least one of the non-terminal destinations.

Tracing

ModSnmp can trace the SNMP and Modbus messages that are received and sent. The trace output is logged at **Trace** level.

Modbus tracing

To enable Modbus tracing, click the **Modbus Trace** button in the main window.

If Modbus tracing is enabled, each Modbus message sent or received is logged as two lines. The first line begins with the character 'D' to indicate a Modbus message to/from a device. This is followed by the character '<' for an incoming message or the character '>' for an outgoing message, and then the Modbus interface name. For a correctly formatted message, the rest of the line consists of the transaction identifier (only for TCP packets), slave identifier, PDU length (in bytes), function code, and then the first few bytes of the body of the message (in hex). If incorrectly formatted data is received, the rest of the line is the received data in hex.

The second line of a message trace provides an interpretation of the main fields in the message. This line also begins with the character 'D'. After the 'D' is the word "Request" for a request message, "Response" for a normal response message, "Error response" for an error response message, or "Discarded" for incorrectly formatted incoming data. "Request" or "Response" is followed by the function code (expressed in words) and the main fields of the message (address, count, etc.). "Error response" is followed by the error code (expressed in words), and possibly by further explanation of the error. "Discarded" is followed by the number of bytes discarded, and an explanation of why the data was discarded.

SNMP tracing

To enable SNMP tracing, click the **SNMP Trace** button in the main window.

If SNMP tracing is enabled, each SNMP message sent or received is logged as a single line starting with the character 'S'. This is followed by the character '<' for an incoming request or the character '>' for an outgoing response, and then the IP address and port of the SNMP manager. The rest of the line consists of a word indicating the type of the request or response, and the first variable OID in the message. For a SET request and for normal responses, the first variable value is also traced.

Running as a server

ModSnmp is normally run as a "server", which means that:

- ModSnmp is started automatically by the operating system when it boots. See Running under Windows or Running under Unix/Linux.
- ModSnmp "runs" automatically when it is started. See Command line options.
- No user interaction is normally needed, so a GUI is not required. See Command line options.
- Error messages must be logged somewhere, since there is no user watching the operation of the program. See Logging.
- There should be a convenient way of changing the settings while ModSnmp is running. See Changing server settings.

Command line options

ModSnmp can be passed the name of a settings file on the command line. For example:

```
java -jar modsnmp3.14.jar config.xml
```

This is optional when running interactively, but is *required* when running as a server.

To get ModSnmp to "run" automatically (i.e. connect to the Modbus devices and listen for SNMP requests), pass the **-auto** option on the command line:

```
java -jar modsnmp3.14.jar -auto config.xml
```

Note that the **-auto** option must come *after* `modsnmp3.14.jar`.

To run ModSnmp without a GUI, pass the **-nogui** option on the command line. The **-nogui** option implies **-auto**, so the command normally used to run ModSnmp as a server looks like:

```
java -jar modsnmp3.14.jar -nogui config.xml
```

Note that the **-nogui** option must come *after* `modsnmp3.14.jar`.

When ModSnmp is started with the **-auto** or **-nogui** option, it will automatically start listening for SNMP requests and also attempt to connect to the Modbus devices. If a non-fatal error occurs when attempting to connect a Modbus device (or if the connection is lost), ModSnmp will periodically try again to connect.

Running under Windows

In order to run ModSnmp as a server under Windows, you need to create a scheduled task that is started at system startup.

The following steps describe how to do it under Windows XP. Other versions of Windows are similar.

- Select **start** → **All Programs** → **Accessories** → **System Tools** → **Scheduled Tasks** to open the **Scheduled Tasks** window.
- Select **File** → **New** → **Scheduled Task** to create a new task, and enter a name for the task (e.g. "modsnmp").
- Double-click on the new task to open the **Properties** dialog.
- Enter the command to start ModSnmp in the **Run** field, for example:

  ```
  java -jar modsnmp3.14.jar -nogui config.xml
  ```

You will have to enter full pathnames for the `modsnmp3.14.jar` and `config.xml` files.

- If necessary, enter or edit the user name in the **Run as** field. This user must have administrative privileges and have a password.
- Click **Set password** and enter the user's password.
- The **Run only if logged on** checkbox should *not* be selected.
- The **Enabled** checkbox *should* be selected.
- In the **Schedule** tab, select **Schedule Task** → **At System Startup**
- In the **Settings** tab, deselect the **Stop task if it runs for** checkbox.

Running under Unix/Linux

Configuring

You will initially need to run ModSnmp with a GUI in order to register the program and to create a configuration file. If the computer that you want to run ModSnmp on does not have an X-server, you should use ssh to login to the ModSnmp computer from a computer that does have an X-server. You can then use ssh to tunnel the ModSnmp GUI output to the X-server.

Starting automatically

In order to run ModSnmp as a server under Unix/Linux, you will have to arrange for ModSnmp to be started automatically at boot time. Unix/Linux systems vary in how they start processes at boot time (e.g SysV init, systemd). If you are not sure what method is used by your particular version of Unix/Linux, you should consult your system documentation.

systemd

This is the preferred method of starting ModSnmp if it is available, since it allows manual stopping & starting and it will automatically restart ModSnmp if it fails.

To start ModSnmp using systemd, download the file modsnmp.service from http://wingpath.co.uk/modbus/resource.php?product=modsnmp. and put it in the '/etc/systemd/system' directory. You will need to edit the "Environment" lines in modsnmp.service to suit your setup.

As root, use the command:

```
    systemctl enable modsnmp
```

to enable automatic starting of ModSnmp at boot time.

You can check the status of ModSnmp using the command:

```
    systemctl status modsnmp
```

If ModSnmp failed to start, this will provide information on why it failed.

You may manually stop ModSnmp using the command:

```
    systemctl stop modsnmp
```

and restart it using the command:

```
      systemctl start modsnmp
```

SysV init

If your system uses System V init for system initialization, you can use the example shell script `rc.modsnmp` from http://wingpath.co.uk/modbus/resource.php?product=modsnmp. You will need to change the pathname definitions to suit your setup.

The details of how to get this script run at boot time vary with different versions of Unix/Linux. You may, for example, need to put this script in the `/etc/init.d` directory and run a script such as chkconfig or update-rc.d to set up appropriately named links to it from the `/etc/rcN.d` directories.

If ModSnmp fails to start, you should look at the OUTPUT file for information on the cause of the failure.

You can manually stop and start ModSnmp using the commands `rc.modsnmap stop` and `rc.modsnmp start`. However, the SysV init method of starting will not automatically restart ModSnmp if it fails.

rc.local

If your Unix/Linux system has a file called '/etc/rc.local' or '/etc/rc.d/rc.local', you can start ModSnmp by adding something like the following to the file:

```
      JAVA=/usr/bin/java
      JARFILE=/home/modsnmp/modsnmp3.01.jar
      PIDFILE=/var/run/modsnmp.pid
      CONFIGFILE=/home/modsnmp/config.xml
      OUTPUT=/home/modsnmp/modsnmp.out
      $JAVA -jar $JARFILE -nogui $CONFIGFILE &<$OUTPUT &
```

You will need to change the definitions (USER, JARFILE, etc.) to suit your setup.

If ModSnmp fails to start, you should look at the OUTPUT file for information on the cause of the failure.

This method of starting ModSnmp does not provide an easy way to stop and restart ModSnmp. A variation of this method is to use the rc.modsnmp script (see SysV init), and run that from rc.local using the commands:

```
      if [ -x /etc/init.d/rc.modsnmp ]
      then
          /etc/init.d/rc.modsnmp start
      fi
```

You could then manually stop and start ModSnmp using the commands `rc.modsnmap stop` and `rc.modsnmp start`.

Changing server settings

You may want to change some of ModSnmp's settings while it is running as a server. The simplest way to do this is to start an interactive copy of ModSnmp on the same computer, and use this to edit the settings file being used by the server copy:

- Use **File** → **Load Settings** to load the settings file being used by the ModSnmp server.
- Change the settings as required.
- Use **File** → **Save Settings** to save the settings file being used by the ModSnmp server.
- The server copy of ModSnmp will notice that the settings file has been modified and will re-read it.

- Check the log output from the ModSnmp server to ensure that the new settings have not caused any errors.

A. Troubleshooting

This section provides information on various errors that can occur when using ModSnmp, and suggests ways of tackling them.

Entry headings in quotes are messages that are displayed in ModSnmp's status bar or log output. For other entries, the heading is a description of the symptoms.

The easiest way to find a message in this section is to click on the **Error Help** button in the status bar (or press the F4 key), or to click on the message in the logging window (or move to the message and press the space bar).

SNMP

"Unknown host: XXX"

The host name that you entered could not be mapped to an IP address.

- The address of the **Host** can be 'localhost' if ModSnmp and the SNMP manager are on the same machine, or can be left empty to accept requests via any network interface. Otherwise check with your System Administrator for the address of your machine.
- Check that your DNS server is working and reachable.

"Can't listen on port YYY"
"Can't listen on interface XXX port YYY"

- Check that the **Port** is not in use by any other server.
- If ModSnmp is run under a Unix/Linux operating system by an ordinary user (not super-user) then the **Port** needs to be above 1023 and the SNMP manager should be configured to use that port.
- The address of the **Host** can be 'localhost' if ModSnmp and the SNMP manager are on the same machine, or can be left empty to accept requests via any network interface. Otherwise check with your System Administrator for the address of your machine.

Modbus

General Communications

"Connection closed"

- Check that the device is still running and listening.

"Error response: ..."

ModSnmp received an error response from the device.

- The rest of the message says what kind of error response. See the troubleshooting entry for the rest of the message for further explanation.
- If tracing is turned on, you can view the received message in the log output.

"Invalid data received: ..."

Data was received by ModSnmp that was unexpected or not correctly formatted as a Modbus message. The received data is discarded without further processing.

- The rest of the message says what was wrong with the received data. See the troubleshooting entry for the rest of the message for further explanation.

- If tracing is turned on, you can view the received data in the log output.

"Discarded: ..."

This message is displayed in the log output when data is received by ModSnmp that is unexpected or not correctly formatted as a Modbus message. The received data is discarded without further processing.

- The rest of the message says what was wrong with the received data. See the troubleshooting entry for the rest of the message for further explanation.
- The received data is displayed in the log output preceding the "Discarded" message.

"Unexpected"

Unexpected data was received by ModSnmp when it was about to send a request message.

- The data may be a delayed response to an earlier request sent by ModSnmp (this should be apparent in the trace output). You may need to increase the response timeout setting if the device or the comms link is slow.

"PDU size (XXX bytes) exceeds maximum (XXX bytes)"

A received message exceeded the limit on PDU size. The message is discarded without further processing.

- Turn on tracing to get more information.

"Error response: Illegal data address"

The device has sent an exception response indicating that the address in the command sent by ModSnmp was incorrect.

- Check that the addresses of the Modbus variables defined in ModSnmp are consistent with the device.
- Check that the device allows the registers/coils to be written, if you are trying to write.

"Error response: Illegal data value"

The device has sent an exception response indicating that data in the command sent by ModSnmp was incorrect.

- Turn on tracing to get more information.
- If you are using 32 or 64 bit values check that the **Word Count** setting is consistent with the device. If ModSnmp and the device disagree over how to interpret the count in the Modbus message, then the number of bytes written by ModSnmp may be different from what the device is expecting.
- Check that the **Modbus Type** is consistent with the device.

"Error response: Illegal function"

The device has sent an exception response indicating that the function code in the command sent by ModSnmp is not supported.

- Turn on tracing to see which function code is not supported.

"Error response: Server failure"
"Error response: Acknowledge"
"Error response: Server busy"
"Error response: Memory parity error"
"Error response: Error code XXX"

The device/bridge/server sent an unexpected error response.

- Check that your device/bridge/server is functioning correctly.

"Wrong transaction ID in response: XXX instead of XXX"

The device sent a response containing a transaction ID that was different from the transaction ID that ModSnmp sent in the request. This may have occurred because the device responded late to an earlier request.

- Turn on tracing to get more information.
- Increase the Response Timeout if necessary.
- Check that your device is functioning correctly.

"Wrong function code in response: XXX instead of XXX"

The device sent a response containing a function code that was different from the function code that ModSnmp sent in the request. This may have occurred because the device responded late to an earlier request.

- Turn on tracing to get more information.
- Increase the Response Timeout if necessary.
- Check that your device is functioning correctly.

"Wrong slave ID in response: XXX instead of XXX"
"Wrong address in response: XXX instead of XXX"
"Wrong count in response: XXX instead of XXX"

The device sent a response that did not correctly echo the data that ModSnmp sent in the request.

- Turn on tracing to get more information.
- Check that your device is functioning correctly.

"Response PDU size incorrect: XXX instead of XXX"

The PDU size of a received response was not correct for the function code specified in the response. If the response included a byte count (e.g. a Read Holding Registers response) then the PDU size is inconsistent with the byte count. If the response did not include a byte count (e.g. a Write Multiple Holding Registers response), then the PDU size was not the expected fixed size.

- Turn on tracing to get more information.
- Check that your device is functioning correctly.

"Response PDU too short: XXX when it should be at least XXX"

The PDU size of a received response was not big enough to contain all the fields required for the function code specified in the response.

- Turn on tracing to get more information.
- Check that your device is functioning correctly.

"Wrong byte count in response: XXX when expecting XXX"

The byte count in the response sent by the device is not what was expected for the count that ModSnmp sent in the request.

- Turn on tracing to get more information.
- Check that the **Modbus Size** is consistent with the device.
- Check that your device is functioning correctly.

Socket Communications

"Unknown host: XXX"

The host name that you entered could not be mapped to an IP address.

- Check that the **Host** is consistent with the device.
- Check that your DNS server is working and reachable.

"Unknown local host: XXX"

The local host name that you entered could not be mapped to an IP address.

- Check that the local host name is correct. Leave the local host field empty if you are not sure.
- Check that your DNS server is working and reachable.

"Can't bind to port XXX"

The socket could not be bound to the local host and port that you entered.

- Check that the **Local host** corresponds to a network interface on the computer on which ModSnmp is running.
- Check that you are permitted to use the **Local port**, and that it is not already being used.

"Can't connect to host 'XXX': No route to host"

The host did not respond to the TCP connection request

- Check that the **Host** is consistent with the device.
- Check that the network is working.

"Can't connect to host 'XXX' port XXX: Connection refused"

The host appears to be reachable and responding, but would not accept the TCP connection request.

- Check that the **Host** and **Port** are consistent with the device.
- Check that device is listening.

"UDP Port Unreachable"

It was not possible to deliver a UDP message.

- Check that the **Host** and **Port** are consistent with the device.
- Check that device is listening.

"Error response: No path available to target"

The server/bridge sent a response indicating that the required device was not reachable.

- Check that the **Slave ID** is consistent with the device.
- Check that your server/bridge is correctly configured for the device you are trying to reach.

"Timed out"

A request was sent to the device, but no response was received.

- Turn on tracing to get more information.

- Check that the **Packet Type** is consistent with the device. The packet type is usually **TCP** for a socket connection.
- Check that the **Slave ID** is consistent with the device.
- Check that **Host** and **Port** are consistent with the device. You may have sent the request to the wrong device/server!
- If using UDP, check that the device is running/listening.
- If you sent a broadcast request, check that the **Always responds** setting is consistent with the device.

"Error response: Target device failed to respond"

The server/bridge sent a response indicating that the required device did not respond.

- Check that the **Slave ID** is consistent with the device.
- Check that your server/bridge is correctly configured for the device you are trying to reach.

Serial Communications

"Serial comms not available"

Serial communications is not supported on the system you are using.

"No serial port specified"

- You must select or enter a serial port name in the interface page.

"Can't open serial port"

The serial port could not be opened and configured correctly.

- Check that you have selected/entered the correct serial port name.
- On Unix/Linux systems, check that you have read and write access permission to the device file ('/dev/...').
- Check that the port is not in use by another program.

"Can't set serial speed"
"Can't set serial data bits"
"Can't set serial stop bits"
"Can't set serial parity"
"Can't set serial RTS control"

The serial port parameter could not be set.

- Check that the serial device supports the parameter value that you are trying to set.

"Parity error"

A parity error occurred when receiving data via the serial port.

- Check that the **Speed**, **Parity**, **Data bits** and **Stop bits** are all set correctly.

"Framing error"

A framing error occurred when receiving data via the serial port.

- Check that the **Speed**, **Parity**, **Data bits** and **Stop bits** are all set correctly.

"Overrun error"

An overrun error occurred when receiving data via the serial port. In other words, data was sent faster than it could be received by the machine running ModSnmp.

- Try using hardware flow control, if it is supported.
- Try using a slower speed.

"Timed out"

A request was sent to the device, but no response was received.

- Turn on tracing to get more information.
- Check that the **Packet Type** is consistent with the device. The packet type is usually **RTU** or **ASCII** for a serial connection.
- Check that the **Slave ID** is consistent with the device.
- If ModSnmp and device are on same machine, check that the **ports** are different.
- Check that the **Speed**, **Parity**, **Data bits**, **Stop bits** and **RTS control** settings are consistent with the device.
- Check that a cable is connecting the ModSnmp and device interfaces.
- If using **RTS control: Flow control** check that the cable is appropriately wired.
- Check that no other software is using the same **port**.
- If you sent a broadcast request, check that the **Always responds** setting is consistent with the device.

RTU Packet Type

"CRC failed"

Data received by ModSnmp failed the Cyclic Redundancy Check (CRC).

- Turn on tracing to get more information.
- If using serial communications, check that **Speed**, **Parity**, **Data bits** and **Stop bits** are all set correctly.
- Try increasing the **EOM Timeout** setting - messages may be getting fragmented by long delays introduced by the comms link or the operating system.

"Message too short (XXX bytes)"

An RTU message must be at least 4 bytes long (1 byte for the slave ID, 1 byte for the function code, and 2 bytes for the CRC).

- Turn on tracing to get more information.
- If using serial communications, check that **Speed**, **Parity**, **Data bits** and **Stop bits** are all set correctly.
- Try increasing the **EOM Timeout** setting - messages may be getting fragmented by long delays introduced by the comms link or the operating system.

"Message too long (> XXX bytes)"

A received message was too long to fit in ModSnmp's input buffer.

- Turn on tracing to get more information.
- If using serial communications, check that **Speed**, **Parity**, **Data bits** and **Stop bits** are all set correctly.

ASCII Packet Type

"No ':' at start of message"
"':' in middle of message"
"LF terminator missing"
"Invalid message length (XXX bytes) - should be odd"
"Non-hex character in message"
"Invalid LRC"

Data received by ModSnmp was not correctly formatted as an ASCII message.

- Turn on tracing to get more information.
- If using serial communications, check that **Speed**, **Parity**, **Data bits** and **Stop bits** are all set correctly.

"Message too short (XXX bytes)"

An ASCII message must be at least 9 bytes long.

- Turn on tracing to get more information.
- If using serial communications, check that **Speed**, **Parity**, **Data bits** and **Stop bits** are all set correctly.
- Try increasing the **EOM Timeout** setting - messages may be getting fragmented by long delays introduced by the comms link or the operating system.

"Message too long (> XXX bytes)"

A received message was too long to fit in ModSnmp's input buffer.

- Turn on tracing to get more information.
- If using serial communications, check that **Speed**, **Parity**, **Data bits** and **Stop bits** are all set correctly.

TCP Packet Type

"Incomplete message: only XXX bytes"

A TCP message must be at least 8 bytes long (7 bytes for the header and 1 byte for the function code).

- Turn on tracing to get more information.
- Try increasing the **EOM Timeout** setting - messages may be getting fragmented by long delays introduced by the comms link or the operating system.

"Incorrect protocol identifier: XXX"

The protocol identifier in the received message header was not zero.
"Length (XXX) too small"

The value in the length field of the received message header must be at least 2 (1 byte for the slave ID and 1 byte for the function code).
"Incomplete message: only XXX bytes when expecting XXX"

The received message was shorter than indicated by the length field of the header.

- Turn on tracing to get more information.
- Try increasing the **EOM Timeout** setting - messages may be getting fragmented by long delays introduced by the comms link or the operating system.

Miscellaneous

Value is read from or written to address in device that is wrong but near to address in command

- If you are using 32 or 64 bit registers, check that **Word Registers** setting is consistent with the device.
- Turn on tracing to get more information.

Wrong value is displayed when reading from device, or wrong value is written to device

- If you are using 32 or 64 bit registers, check that **Little-Endian** and **Word Registers** settings are consistent with the device.
- Turn on tracing to get more information.

The 16-bit words of 32 or 64-bit values are in reverse order

- Select/deselect **Little Endian** to reverse the order and make it consistent with the device.

"Not enough data in message: XXX bytes when expecting XXX"

A received message contains too little data for the count and register sizes specified in the request.

- Turn on tracing to get more information.
- If you are using 32 or 64 bit registers, check that the **Word Registers** and **Word Count** settings are correct.

"Excess data in message: XXX bytes when expecting XXX"

A received message contains too much data for the count and register sizes specified in the request.

- Turn on tracing to get more information.
- If you are using 32 or 64 bit registers, check that the **Word Registers** and **Word Count** settings are correct.

B. Release Notes

3.14 - 16 Apr 2021

Fixed problem with loading serial library on Raspbian Jessie.

3.13 - 12 May 2020

Fixed timing problem with handling of cached values.

3.12 - 4 Mar 2020

Fixed problem with handling of delays in received RTU messages.

3.11 - 28 Aug 2019

Fixed problem causing freezing in Windows serial communications.

3.10 - 2 Nov 2018

Resolved issues arising from changes in recent releases of Java JRE.

3.09 - 16 May 2018

Fix for problem loading serial library on Raspberry Pi.

3.07 - 17 Jan 2017

Added support for serial interfaces on ARM processor on Raspberry Pi.

Minor documentation changes.

3.06 - 26 Oct 2016

Fix for problem handling invalid slave IDs in Modbus server.

3.05 - 21 Oct 2016

Fix for problem loading serial library.

3.04 - 6 Jan 2015

Workaround for reading CSV files that have been edited by Microsoft utilities.

3.03 - 15 Dec 2014

Fix for a bug that could cause write delays on serial connections.

3.02 - 3 Nov 2014

Added option to read identification information from Modbus devices.

Support idle timeout for all Modbus connection types, not just TCP socket.

Minor changes to error reporting.

Upgraded to version 2.3.1 of the SNMP4J library.

3.01 - 18 Sep 2014

Added caching of Modbus values. The expiry period for a value (i.e. its cache lifetime) is configurable. Storing values in the cache means that, within their lifetimes, they are available immediately when requested by an SNMP manager.

Added the option to poll. ModSnmp can automatically update Modbus values when their lifetimes expire. This enables ModSnmp to have the new values ready for the SNMP manager when it requests them.

Added ability to function as a Modbus bridge. Modbus masters (as well as SNMP managers) can now communicate with the Modbus slaves via ModSnmp.

Implemented various measures to promote the smooth running of the system under adverse load conditions. The adverse load might be due, for example, to one or more slaves responding very slowly, or an SNMP manager sending requests (possibly with retries) at too high a rate.

Upgraded to version 2.3.0 of the SNMP4J library.

2.04 - 4 Aug 2013

Under Windows, scan for serial ports up to COM99 instead of COM20.

2.03 - 10 Jun 2013

Changes to registration dialogs to simplify handling of upgrades.

2.02 - 25 May 2013

Added option to use function 6 instead of function 16 for Modbus writes.

Minor GUI changes.

2.01 - 15 May 2013

Use device definitions to organize the SNMP-Modbus mapping. This facility enables easier entry of multiple devices that have the same Modbus registers. It also allows more meaningful names to be associated with OIDs.

Automatically generate the MIB file.

Support listening on multiple SNMP interfaces.

Support BITS and Gauge32 types.

Added the ability to log to a window.

Added support for higher serial line speeds.

Upgraded to version 2.2.0 of the SNMP4J library.

1.24 - 7 Mar 2013

Fixed a deadlock that could occur during shutdown.

1.23 - 25 Feb 2013

Minor GUI changes.

1.22 - 9 Jan 2013

Minor changes to trace and error messages.

1.21 - 21 Sep 2012

Added "offset" and "scale" fields to Modbus OID entries.

Allow configuration of the local host and local port for socket connections to slaves.

1.20 - 15 Aug 2012

Fix to linking of serial library under Linux.

Minor changes to registration dialogs.

1.19 - 14 Jul 2012

Support listening on multiple SNMP interfaces.

Minor GUI and documentation changes.

1.17 - 8 Jun 2012

Minor GUI changes, particularly to registration dialogs.

1.16 - 22 May 2012

Minor changes to error reporting.

1.15 - 14 May 2012

Upgraded to version 2.1.0 of the SNMP4J library.

Minor changes to error reporting.

1.14 - 2 May 2012

New serial library to replace the Java Comms API. Provides 32-bit and 64-bit support under Windows and Linux.

1.13 - 7 Nov 2011

Minor changes.

1.12 - 11 Oct 2011

Minor serial comms changes: don't use customer's copy of Java Comms API, and work around a bug in a device driver.

C. Modbus protocol documents

The Modbus protocol was invented by Gould Modicon (now a division of Schneider Electric), and is now administered by modbus.org. Gould-Modicon/Schneider/modbus.org have published five documents describing the Modbus protocol:

Modbus Protocol Reference Guide. PI-MBUS-300. Rev J. June 1996. This is the original Modbus specification, which has been superceded by the current Modbus specification and the serial specification listed below. This document is referred to as the "original Modbus specification" in this manual.

Open Modbus/TCP Specification. Andy Swales. Release 1.0. March 1999. This document specifies how Modbus messages are sent over a network using the TCP/IP protocols. It also contains some intelligent commentary on the Modbus protocol, and an appendix on client and server implementation, which may be useful if this area is new to you. Surprisingly, this document does not appear to be available from the official Modbus web-site. This document is referred to as the "Modbus/TCP specification" in this manual.

Modbus Application Protocol. V1.1b3. April 2012 This is the current definition of Modbus function codes and the format of the corresponding request and response messages. It does *not* cover how these messages are packaged in the RTU, ASCII or TCP variants of the protocol. This specification is referred to as the "current Modbus specification" in this manual.

Modbus Messaging on TCP/IP Implementation Guide. Rev 1.0b. December 2006 This consists mainly of what appear to be design notes for Schneider Automation's implementation of Modbus/TCP. You will probably find it hard to read if you don't already have a good understanding of the area, and not very useful if you do. It is, however, the only document on the official Modbus web-site that contains (in section 3.1) a description of the Modbus/TCP protocol. This document is referred to as the "Modbus/TCP implementation guide" in this manual.

Modbus over Serial Line - Specification & Implementation Guide. V1.02. December 2006 This is the current definition of the RTU and ASCII variants of the Modbus protocol. It also contains physical layer (electrical) specifications. This document is referred to as the "serial specification" in this manual.

D. SNMP documents

SNMP is specified in a series of IETF (Internet Engineering Task Force) documents.

The current version (V3) is specified by:

- RFC 3410 Introduction and Applicability Statements for Internet Standard Management Framework
- RFC 3411 An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks
- RFC 3412 Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)
- RFC 3413 Simple Network Management Protocol (SNMP) Applications
- RFC 3414 User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)
- RFC 3415 View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)
- RFC 3416 Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)
- RFC 3417 Transport Mappings for the Simple Network Management Protocol (SNMP)
- RFC 3418 Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)

The following version 2 documents are also still current:

- RFC 2578 Structure of Management Information Version 2 (SMIv2)
- RFC 2579 Textual Conventions for SMIv2
- RFC 2580 Conformance Statements for SMIv2

The original SNMP (version 1) is specified in:

- RFC 1157 A Simple Network Management Protocol (SNMP)

The following IETF documents are also relevant:

- RFC 3584 Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework
- RFC 4181 Guidelines for Authors and Reviewers of MIB Documents

E. Acknowledgments

ModSnmp uses the SNMP4J and Apache log4j libraries. These libraries are included in the ModSnmp JAR file.

SNMP4J

SNMP4J is licensed under the Apache License, Version 2.0. This license requires that the following NOTICE be provided:

The LICENSE file referred to in the NOTICE is available from the SNMP4J website.

Apache log4j

The Apache log4j is licensed under the Apache License, Version 2.0. This license requires that the following NOTICE be provided: