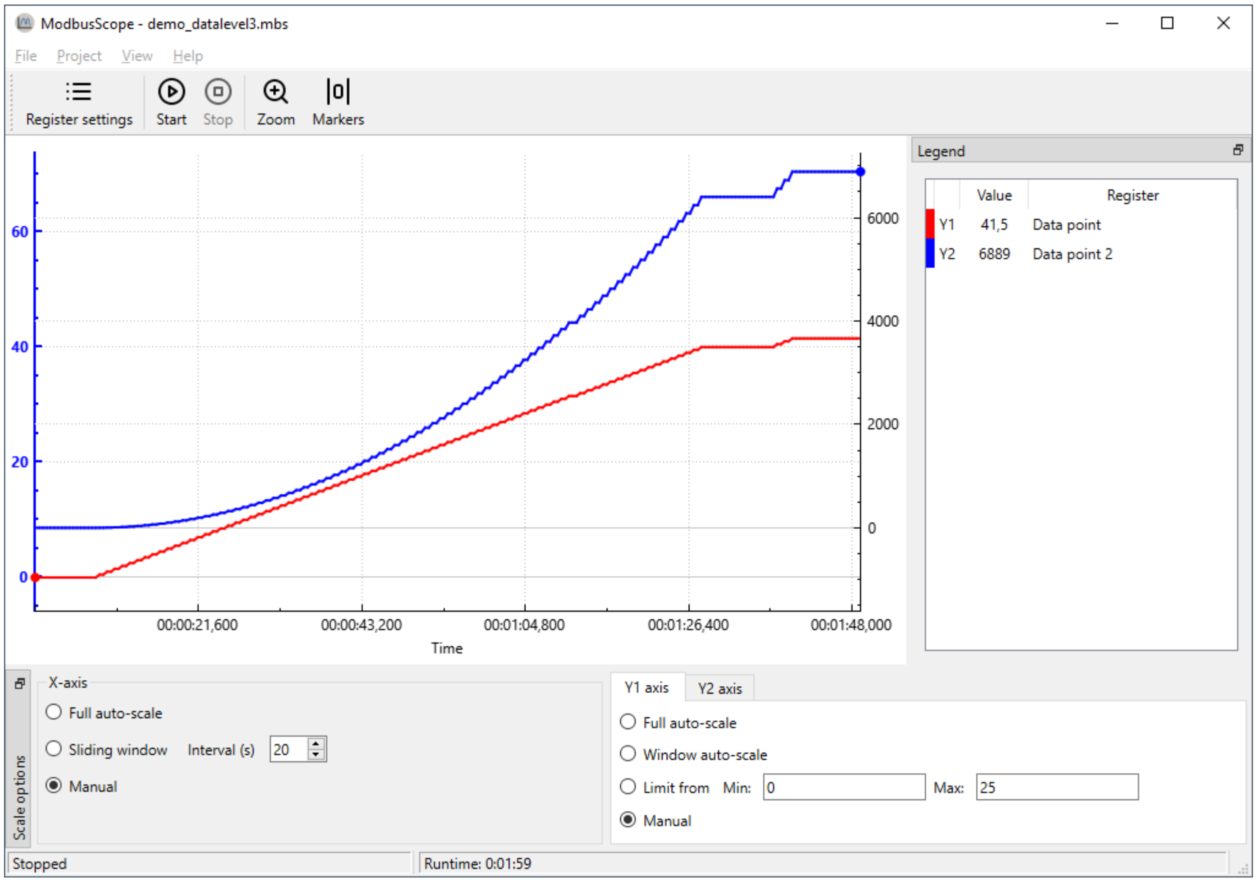
ModbusScope

Jens Geudens

Dec 04, 2023

CONTENTS:

ModbusScope is a GUI tool to log data using the Modbus protocol and put the data into a graph. Data can be examined while the logging continues. It is also possible to export the data to a csv file for analyzing the data afterwards.



		Value	Register
🟥	Y1	41,5	Data point
🟦	Y2	6889	Data point 2

X-axis
- Full auto-scale
- Sliding window Interval (s) 20
- [✓ Manual]

Y1 axis
- Full auto-scale
- Window auto-scale
- Limit from Min: 0 Max: 25
- [✓ Manual]

Stopped   Runtime: 0:01:59

CHAPTER ONE

DOWNLOAD

The latest version of *ModbusScope* can always be downloaded from the release page on Github. This page provides both the installer version and the standalone version of the software. The Windows installer is also available as Chocolatey package.

CHAPTER

TWO

OVERVIEW

This guide explains the basics of using ModbusScope. The software is designed to collect and display data from Modbus registers in a graphical format, making it easy to monitor and analyze the dynamic behavior of a system.

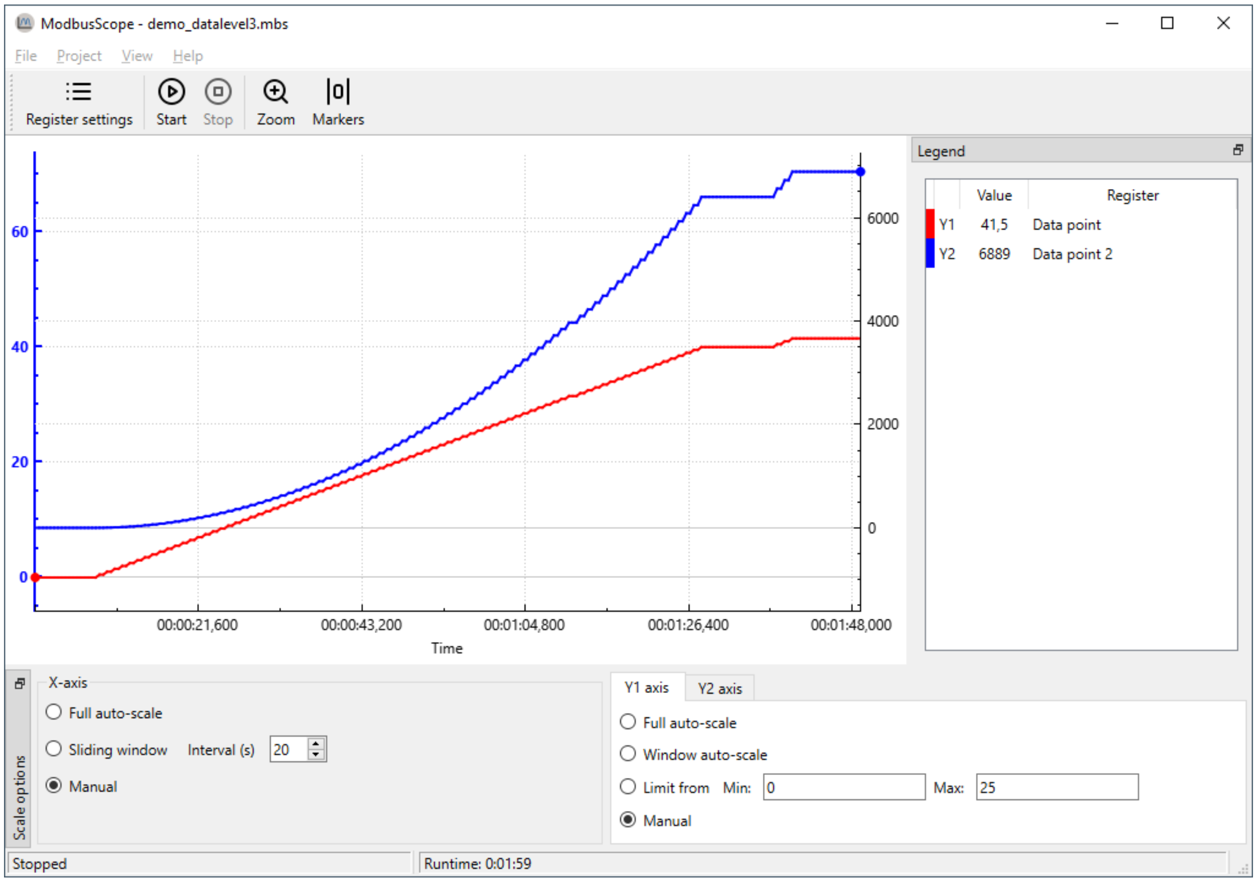
2.1 Installing

The *ModbusScope* installer or standalone version can always be downloaded from the release page.

ModbusScope can be easily installed by double-clicking on the provided `.msi` installer file and following the on-screen instructions. The installer will handle the installation of all necessary files on your computer. At the end of the install process, you have the option to set ModbusScope as the default application for opening `.mbs` files.

2.2 Main screen

When you first open ModbusScope, you'll see several windows displayed on the main screen. Some of them, like the marker window, may not be visible immediately. This window will only appear when a marker has been enabled within the application.



		Value	Register
🟥	Y1	41,5	Data point
🟦	Y2	6889	Data point 2

X-axis
- Full auto-scale
- Sliding window Interval (s) 20
- [✓ Manual]

Y1 axis / Y2 axis
- Full auto-scale
- Window auto-scale
- Limit from Min: 0 Max: 25
- [✓ Manual]

The plot view is the central part of the *ModbusScope* window. It displays the data of the active registers that are being polled, represented as plots. The settings in the axis scale dock allow you to zoom a specific parts of the data or automatically scale the view to show all of the information. The legend provides an overview of the active registers and you can hide or show a specific register plot by double-clicking it in the legend. When a plot is hidden, the register is still being polled, it is only temporarily hidden in the plot view. Additionally, markers can be used to examine the data within a specific time frame, which allows for more detailed analysis.

2.3 Quick start

To get started with *ModbusScope*:

- Use the register settings window to add the desired Modbus registers
- Adjust the connection settings according to your connection the Modbus slave.
- Press the *Start Logging* button (play icon) to begin collecting data.
- Press the *Stop Logging* button to stop logging.
- Examine the logged data using the various windows and tools provided by the application.

CHAPTER

THREE

GRAPHVIEW

3.1 Start/stop log

Once you have added the desired Modbus registers, you can begin logging data by pressing the *Start Logging* button. ModbusScope will communicate with the Modbus slave specified in the connection settings. The Modbus slave can be connected through a TCP or RTU (serial) connection. Once the *Start Logging* button is pressed, ModbusScope will start logging the data and will automatically display the values on the graph.

> **NOTE**: When you press the *Start Logging* button, it will clear any data that is already present in the graph and start logging new data.

When you have finished testing and collecting data, you can stop logging by pressing the *Stop Logging* button. Once logging is stopped, you can further examine the collected data by using the various windows and tools provided by the application.

3.2 Adjust scale settings

While ModbusScope is logging data, you can view the already logged values in the graph view. The various scale settings allow you to examine the data in different ways and zoom in or out as needed. You can adjust the scale settings to suit your needs and easily examine the data while new values are being added to the log. The software provides several scale settings that you can use to customize the view of the data.

3.2.1 X-axis

The x-axis of the graph can be scaled in three ways: *full auto-scale*, *sliding window*, and *manual* settings.

- *Full auto-scale* will automatically adjust the maximum of the x-axis to include all time values in the graph.
- *Sliding window* allows you to view only the values of a specific time period, which is configurable.
- *Manual* scale setting means that the scaling is fixed and will not change automatically while logging, even when new values are logged, the current time period stays the same.

3.2.2 Y1- and Y2-axis

Compared to the x-axis, the y-axis has two similar settings: *full auto-scale* and *manual*. These options work the same way as they do for the x-axis. In addition to these, the y-axis also has two other modes: *window auto-scale* and the *limit from* setting.

- *Window auto-scale* automatically adjusts the range of the y-axis based on the values currently visible in the graph.
- *Limit from* setting allows the user to set the minimum and maximum values for the y-axis.

There are 2 Y-axis available, one for each graph, and the user can select which Y-axis to use per graph. This can be done in the *register* dialog or by double clicking the Y-axis indicator in the legend.

3.3 Zoom graph

The graph-view in *ModbusScope* supports zooming to allow for a more detailed examination of the logged data. By using the scroll wheel on your mouse, both the x- and y-axis can switch to manual setting and the range of the axis will be increased or decreased based on the scroll wheel movement. The current position of the mouse cursor is used as a reference point for the zoom action, allowing you to focus on specific areas of the graph.

ModbusScope allows you to select a single axis to zoom in and out of by clicking on it. This means that when you use the mouse wheel, only the selected axis will be zoomed while the range of the other axis will remain the same. To deselect an axis, you can click anywhere in the graph view. Double-clicking an axis will reset it to full auto-scale setting.

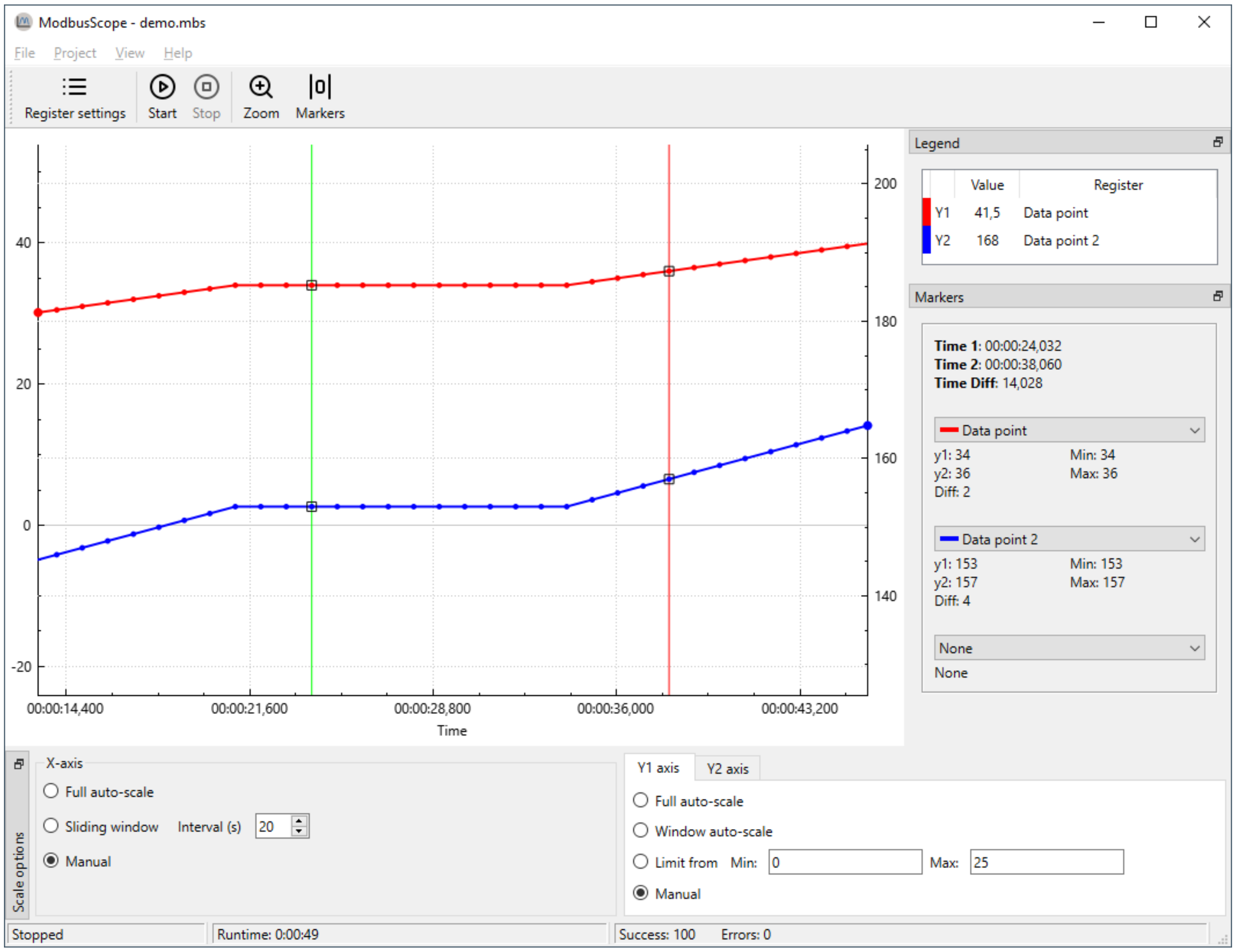
The *Zoom* button in *ModbusScope* allows the user to zoom in on a specific region of the graph by drawing a rectangle over the area of interest using the mouse. This makes it easy to focus on a specific section of the data.

3.4 Enable/Disable markers

ModbusScope offers a useful feature to investigate dynamic behavior of a system by measuring the time and value differences between two points on the graph. To use this feature, you need to add two vertical markers to the graph. Once the markers are added, you can move them to the desired positions and *ModbusScope* will calculate and display the time and value differences between them. This allows you to easily compare and analyze changes in the system over a specific period of time.

To add the left marker (green vertical line), press the `Ctrl` key and click on the location of interest with the left mouse button. The position must coincide with a sample in the graph. To add the right marker (red vertical line), press the `Ctrl` key and click on the location of interest with the right mouse button.

Once you have added the markers by using the `Ctrl` key and left or right mouse click, a panel called *Markers* will appear on the right side of the screen. This panel displays information about the markers, including the value of the registers at the left marker (Time 1) and the right marker (Time 2). It also shows the time and value difference between the two markers, which allows you to easily compare and analyze the data.



	Value	Register
🟥 Y1	41,5	Data point
🟦 Y2	168	Data point 2

Time 1: 00:00:24,032
Time 2: 00:00:38,060
Time Diff: 14,028

Data point
y1: 34 Min: 34
y2: 36 Max: 36
Diff: 2

Data point 2
y1: 153 Min: 153
y2: 157 Max: 157
Diff: 4

None
None

X-axis
Full auto-scale
Sliding window Interval (s) 20
[✓ Manual]

Y1 axis / Y2 axis
Full auto-scale
Window auto-scale
Limit from Min: 0 Max: 25
[✓ Manual]

Stopped | Runtime: 0:00:49 | Success: 100 Errors: 0

CHAPTER

FOUR

CONFIGURATION

4.1 Configure register settings

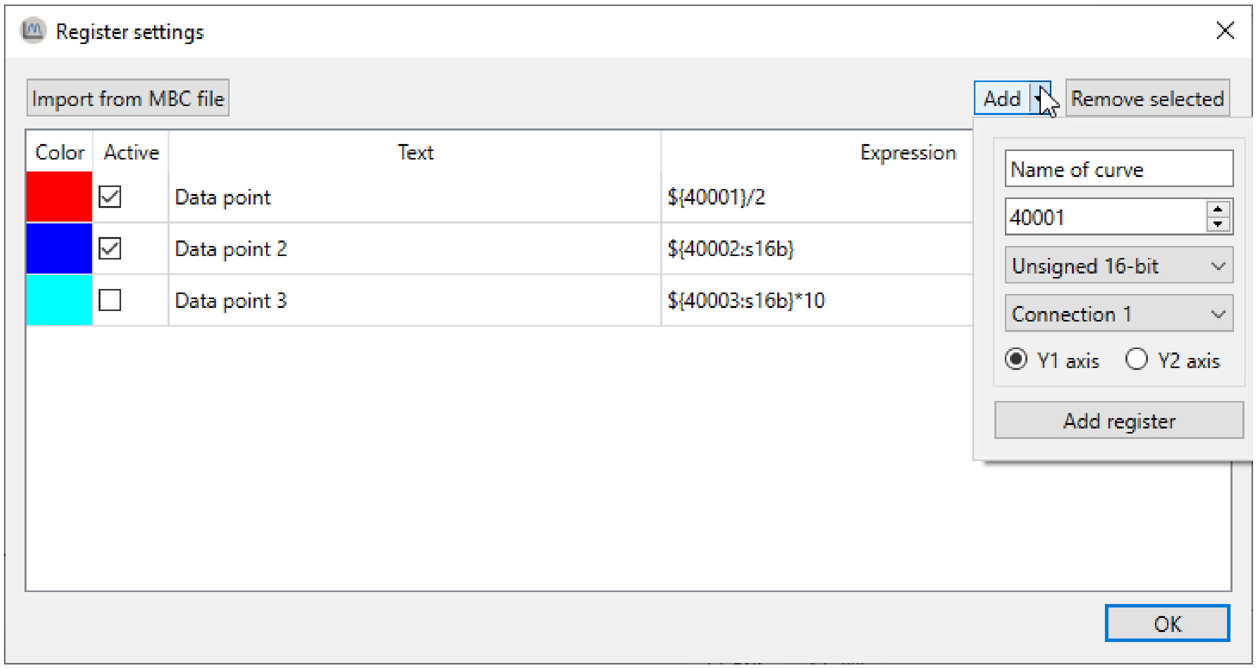
When you first open *ModbusScope*, no Modbus registers are added. To add registers, click on *Register Settings* in the tool-bar on the interface. This will open a window where you can add and edit the registers that you want to monitor.

4.1.1 Add Modbus registers

In the *Register settings* dialog, you can add Modbus registers either manually or by importing them from a *.mbc* file. When you open a *.mbs* file in *ModbusScope*, some registers may already be present in the dialog, which allows you to continue working with the previously used registers.

Once the registers have been added, you can adjust them as needed. This includes updating the name and changing the color. Expressions are used to define which data is added to the graph (and log).

> **NOTE**: The number of registers that are polled can greatly impact the sample rate. To achieve a higher resolution in time, you can either reduce the number of actively polled registers or make sure that the registers are in consecutive addresses so that they can be polled in one packet. This will help to increase the sample rate and improve the resolution of the data.



Color	Active	Text	Expression
🟥	✓	Data point	${40001}/2
🟦	✓	Data point 2	${40002:s16b}
🟦	☐	Data point 3	${40003:s16b}*10

Name of curve

40001

Unsigned 16-bit

Connection 1

[✓ Y1 axis] Y2 axis

Object types

In *ModbusScope*, a Modbus register is represented as `${REG[@CONN][:TYPE]}` where `REG` is the register address, `CONN` is the connection number, and `TYPE` is the type of the register.

- `${45332}` => 16-bit unsigned using connection 1
- `${45332:  s16b}` => 16-bit signed using connection 1
- `${45332@2:  32b}` => 32-bit unsigned using connection 2
- `${45332@2}` => 16-bit unsigned using connection 2

Register address

ModbusScope supports coils, discrete inputs, input registers and holding registers. The object type is derived from the offset of the register address.

Range	Object type	Modbus function code
1 - 9999	Coil	1
10001 - 19999	Discrete inputs	2
30001 - 39999	Inputs registers	4
> 40001	Holding registers	3

The Modbus standard defines the range for the holding register as 40001 to 49999. However, in the data packets that are sent, the offset of 40001 is removed. As the address field is 16-bit, the actual possible range is 40001 to 105536. To support as many devices as possible, *ModbusScope* supports the full range of addresses. At the moment, it isn't possible to access the full range of the other object types, the allowed range is limited to an address space of the first 10.000 registers.

Supported register types

Following types for holding and input registers are currently supported by ModbusScope:

- `16b`: Unsigned 16-bit value
- `s16b`: Signed 16-bit
- `32b`: Unsigned 32-bit value
- `s32b`: Signed 32-bit value
- `f32b`: 32-bit float (IEEE 754)

The endianness of 32-bit registers can be configured per connection with the `32-bit little endian` setting in the connection settings dialog. The register type is ignored for coils and discrete inputs.

4.1.2 Expressions

Expressions are used to define calculations or transformations of data in *ModbusScope*. They can be used to convert raw data from a Modbus register into a more meaningful value, or to perform mathematical operations on multiple registers. These expressions can be defined using a variety of mathematical operators and functions, as well as references to specific registers. For example, you could use an expression to calculate the power usage of a system by multiplying the voltage and current readings from two different registers.

ModbusScope supports a variety of binary operators that are commonly used in expressions such as `|` for bitwise OR, `&` for bitwise AND, << for bitwise left shift, and >> for bitwise right shift. It also supports basic arithmetic operators like `+`, `-`, `*`, `/`, `%`, and `^` for addition, subtraction, multiplication, division, modulus and exponentiation respectively. In addition to the above operators, *ModbusScope* also supports hexadecimal numbers represented with the `0x` prefix and binary numbers represented with `0b` prefix. It also supports floating-point numbers, and both a decimal point and comma can be used as floating-point separator, whichever is encountered first. This allows for a greater flexibility in creating expressions that suit the user's needs.

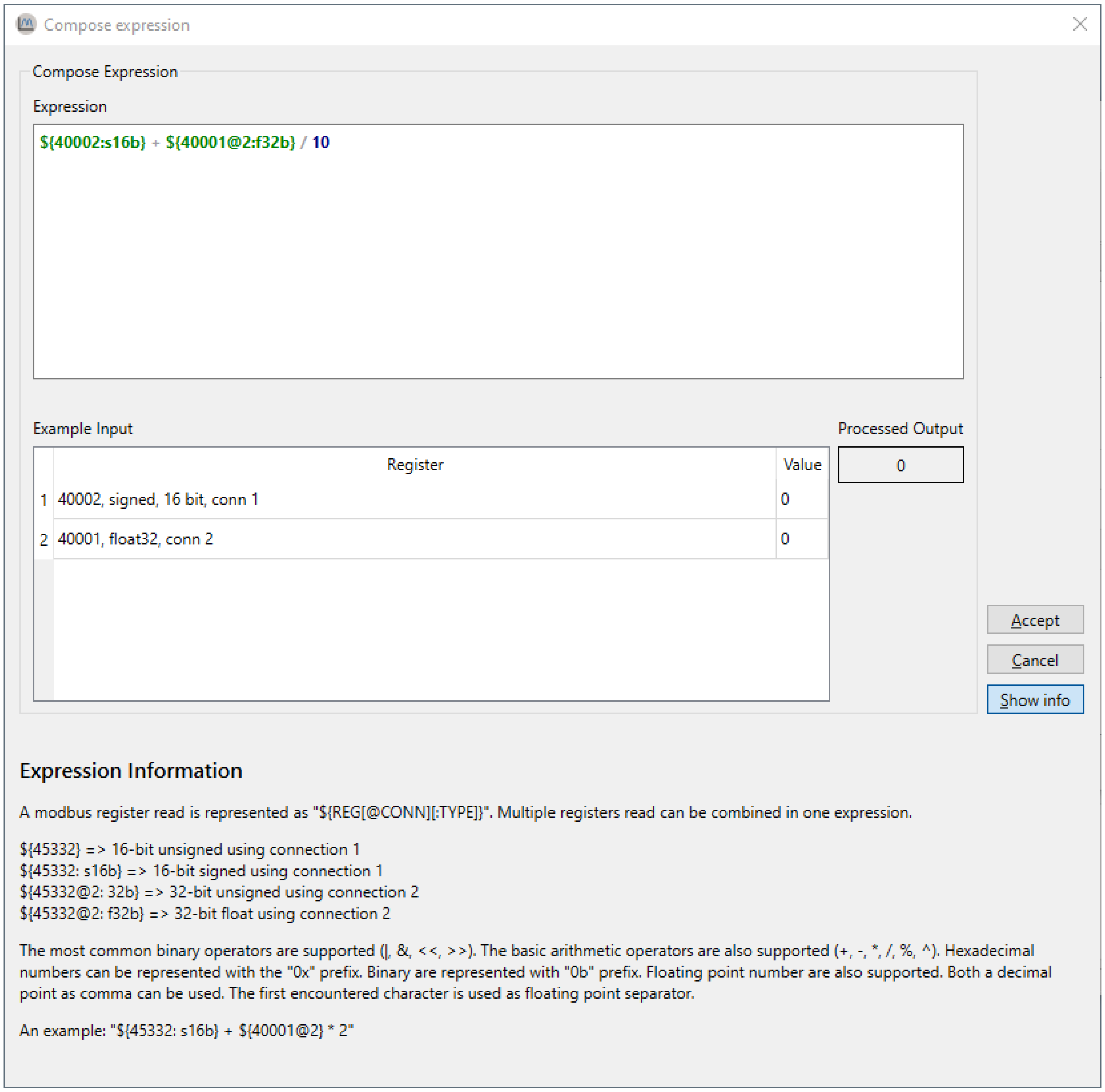
Some examples of valid expressions are

- `${40001:  s16b} + ${40002@2} * 2`
- `${40001:  s32b} * 1.1`
- `${30001} + 0x1000`
- `${40001} & 0b11111000`
- `(${30001} >> 8) & 0xFF`

4.1.3 Compose expression window

The compose expression window is a feature in *ModbusScope* that allows the user to create custom calculations using registers and other mathematical operations. Expressions allow for more flexibility in defining the data that is logged and displayed on the graph, and it can be used to create expressions that are specific to the user's needs. The compose expression window can be accessed from the register settings dialog, and it provides a user-friendly interface for creating and editing expressions.

This window can be opened by double-clicking the expression cell in the register settings dialog. This allows you to easily test and verify that the expression is functioning correctly before applying it to the logged data. The expression can be updated freely, and the register definition will be validated while you are entering it. When the register definition is green, it means that it is valid. The *example input* table can be used to enter values to test and verify the expression with actual values. This allows you to ensure that the expression is working as intended and to make any necessary adjustments before using it to log data. It is possible to combine multiple register reads in one expression by using mathematical operators and functions.



```
${40002:s16b} + ${40001@2:f32b} / 10
```

	Register	Value
1	40002, signed, 16 bit, conn 1	0
2	40001, float32, conn 2	0

Processed Output: 0

Expression Information

A modbus register read is represented as "${REG[@CONN][:TYPE]}". Multiple registers read can be combined in one expression.

${45332} => 16-bit unsigned using connection 1
${45332: s16b} => 16-bit signed using connection 1
${45332@2: 32b} => 32-bit unsigned using connection 2
${45332@2: f32b} => 32-bit float using connection 2

The most common binary operators are supported (|, &, <<, >>). The basic arithmetic operators are also supported (+, -, *, /, %, ^). Hexadecimal numbers can be represented with the "0x" prefix. Binary are represented with "0b" prefix. Floating point number are also supported. Both a decimal point as comma can be used. The first encountered character is used as floating point separator.

An example: "${45332: s16b} + ${40001@2} * 2"

Expression error

When an error is detected in the expression or when the combination of the expression with a specific input value generates an error, no output value will be shown in the *compose expression* window. A specific error message will be displayed to indicate the issue, and the register definition will be highlighted in red, this allows the user to easily identify and correct any errors in the expression. It's important to test the expression before using it log data, to ensure that it is working correctly and producing the desired results.

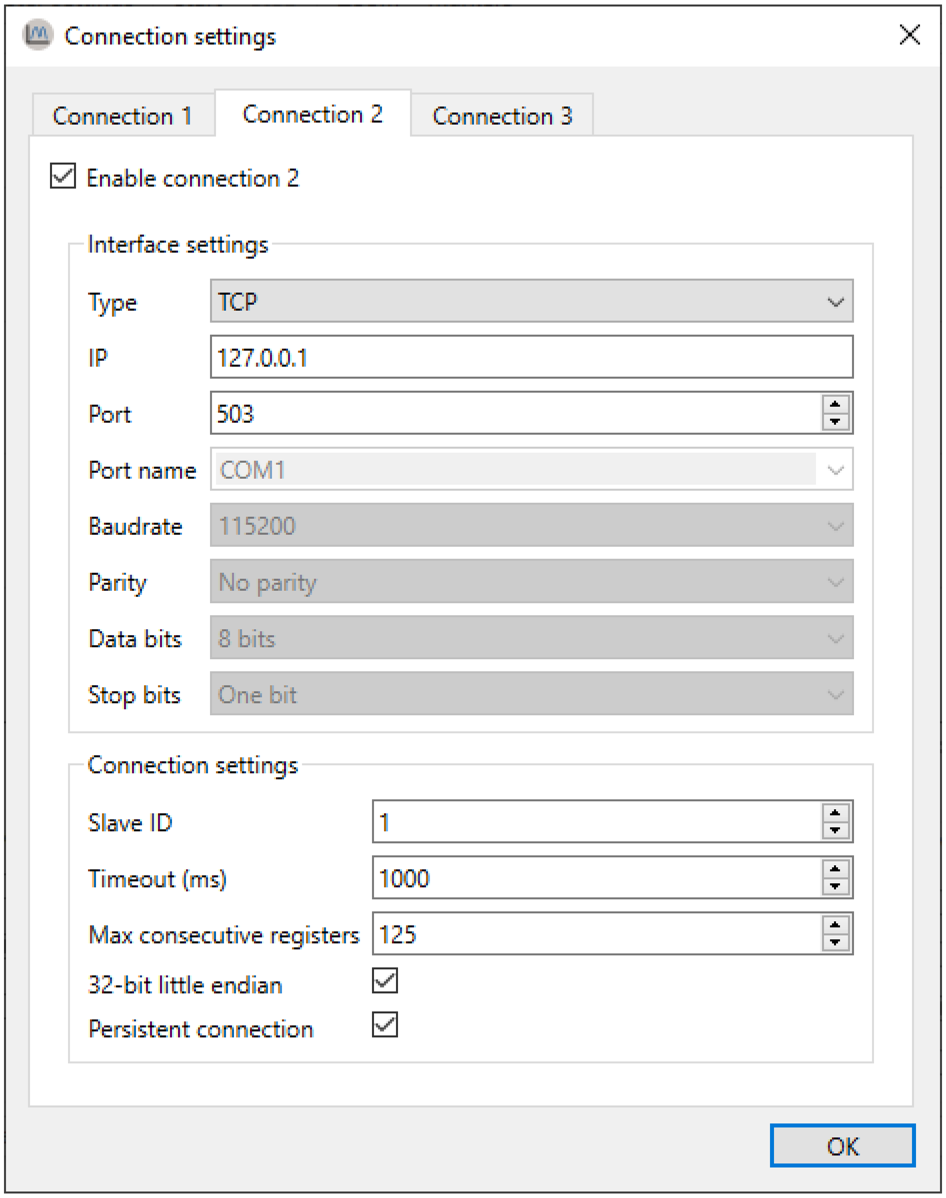
4.2 Configure connection settings

The *connection settings* window allows you to configure up to three connections, which means that several Modbus slaves can be polled in a single log session. Each connection can be configured with the Modbus protocol of the slave. ModbusScope support Modbus TCP and RTU. Modbus ASCII isn't supported.

Some settings such as ip, port, port name, baud rate, parity and number of data and stop bits are specific to the type of connection (TCP or RTU) and are used to establish a connection to the slave device. The other settings such as slave ID, timeout, max consecutive register, and 32-bit little endian, are specific to the Modbus protocol implementation in the device and are used to configure how the application communicates with the slave device.

The timeout settings determine how long the application will wait for a response from the slave before timing out. It is possible to read multiple consecutive registers in a single request in Modbus. However, most devices have a limit on the number of consecutive registers that can be read in a single request. This limit is referred to as the *maximum consecutive registers*. In Modbus, 32-bit values are stored in two consecutive 16-bit registers, in either big-endian or little-endian format. In some devices, 32-bit values are stored in big-endian format by default, while in others they are stored in little-endian format. The 32-bit endianness setting in *ModbusScope* allows you to configure the endianness of the 32-bit values read from the registers, so that the application can correctly interpret the data. The persistent connection option is specific to *ModbusScope*. When enabled, it allows the application to keep the connection open between polling data points, which can increase the polling rate and reduce the time required to establish new connections. The connection will only be reinitialized when a connection error occurs. It's important to ensure that the connection settings are correct and that the correct protocol is selected before starting a log session. With correct configuration, the application will be able to communicate with the slave device and retrieve data from the registers.

In the *register settings* window, you can link each register to a specific connection. This allows you to poll multiple slaves simultaneously and display the data in a single graph for easy comparison.



Enable connection 2 ✓

Interface settings

Type	TCP
IP	127.0.0.1
Port	503
Port name	COM1
Baudrate	115200
Parity	No parity
Data bits	8 bits
Stop bits	One bit

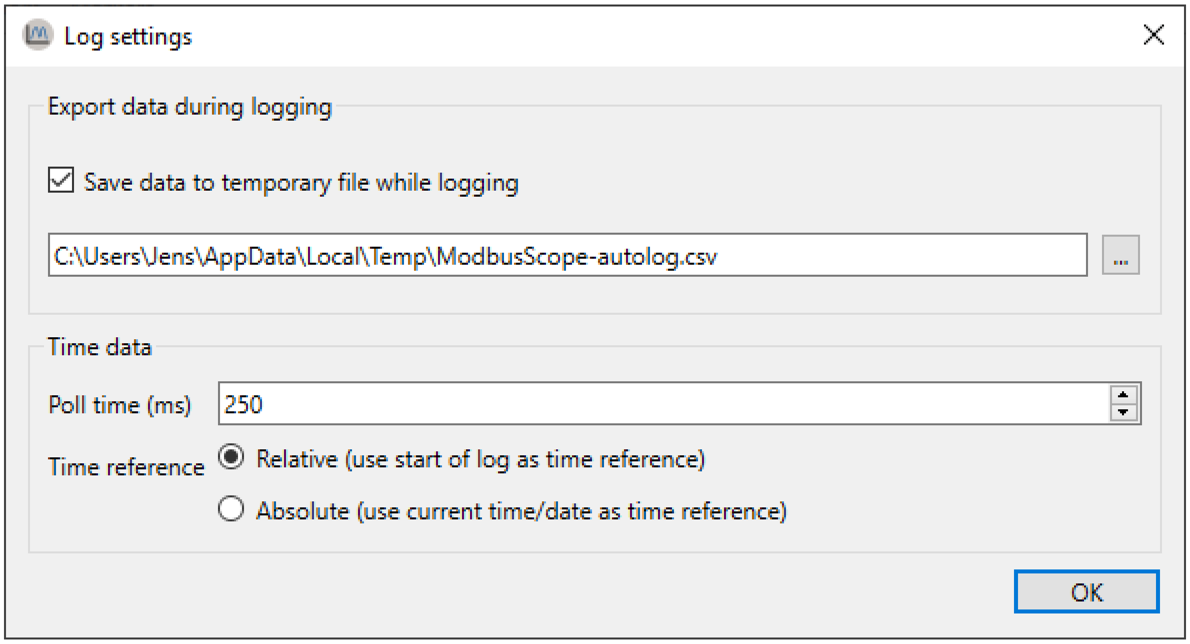
Connection settings

Slave ID	1
Timeout (ms)	1000
Max consecutive registers	125
32-bit little endian	✓
Persistent connection	✓

4.3 Configure log settings

ModbusScope creates a data file in the general temporary folder by default when a logging session is started. The data points are appended to the file during the logging session, so that the data can be recovered in case of an unforeseen crash or if the user forgets to save the data before quitting the application. The temporary file is cleared every time a polling session is started, so that new data can be logged. Some of this behavior can be customized in the *log settings* window. The user can choose to disable the feature or change the location of the temporary data file. This allows the user to ensure that the data is saved in a location that is convenient for them.

In the *log settings* window, this behavior can be disabled or the temporary data file can be changed.



[✓] Save data to temporary file while logging

C:\Users\Jens\AppData\Local\Temp\ModbusScope-autolog.csv

Time data

Poll time (ms) 250

Time reference [● Relative (use start of log as time reference)] ○ Absolute (use current time/date as time reference)

By default, *ModbusScope* will log data points every 250 milliseconds. This is the default sample rate and it can be adjusted in the *log settings* window. The user can increase or decrease the sample rate to suit their needs. Additionally, by default, *ModbusScope* will log timestamps relative to the start of the log session. This means that the time-stamp of each data point is recorded as the time elapsed since the start of the logging session. However, this behavior can be changed by enabling the *use absolute times* option in the *log settings* window. When this option is enabled, absolute timestamps are logged instead, meaning that the actual date and time of each data point is recorded in the log file.

This feature allows the user to choose the time-stamp format that is most appropriate for their use case and to easily compare the logged data with other data that may have been collected at different times.

4.3.1 Optimize logging interval

The minimum logging interval is determined by several factors such as the Modbus protocol and the register addresses. When the requested register addresses aren't in successive order, the Modbus protocol has an inherent slowdown and *ModbusScope* will split the read request into several packets. This will negatively impact the minimum logging interval because of the Modbus end of frame timeout. To achieve a fast logging interval, it's important to limit the number of registers and make sure that consecutive registers are polled. This can help minimize the inherent slowdown caused by the Modbus protocol and allow for a faster logging interval.

CHAPTER

FIVE

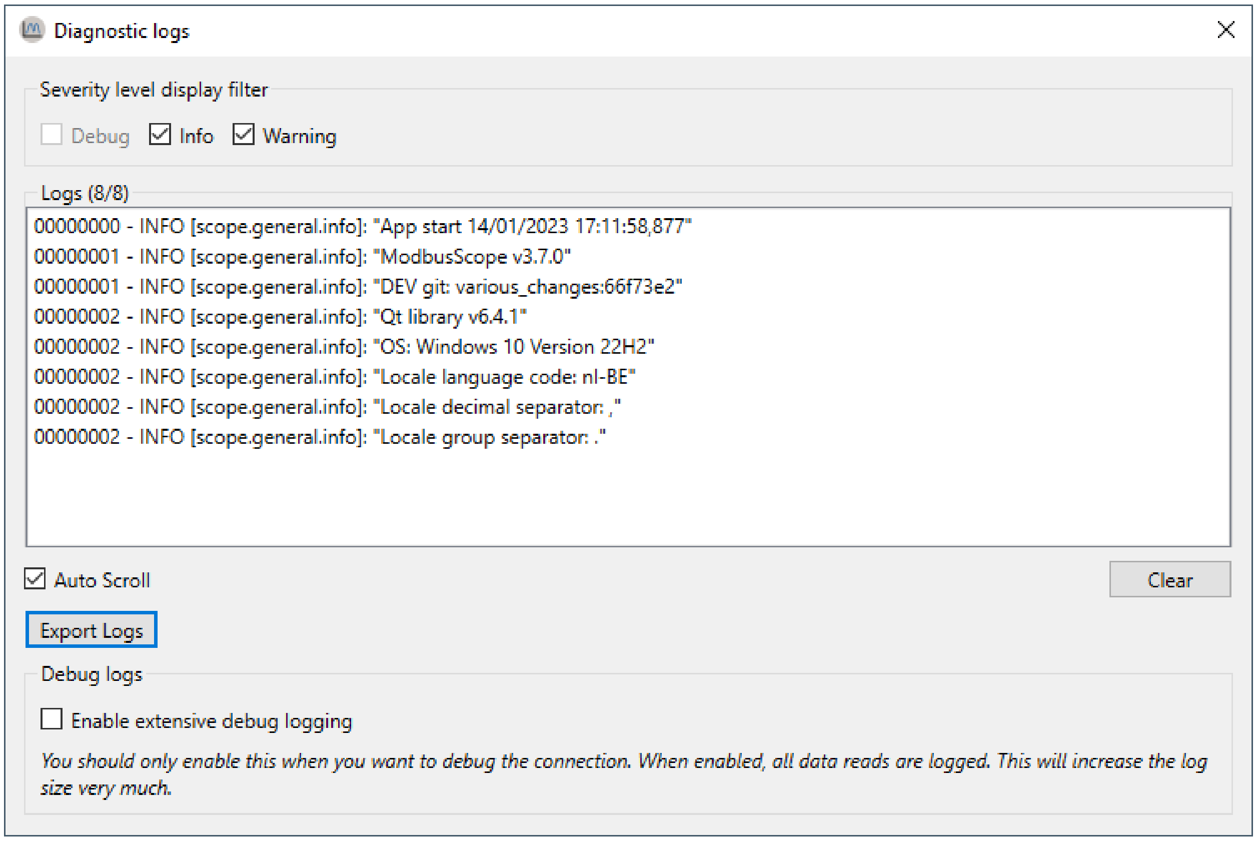
DIAGNOSTICS

5.1 Diagnostic logging

During a logging session, various errors may occur. To facilitate quick examination and resolution of these errors, *ModbusScope* has built-in logging capabilities. For example, if a slave responds too slowly, a timeout may be generated. All register reads are recorded in the diagnostic logs along with the results, allowing communication or connection errors to be easily examined. Other common errors include Modbus exceptions, such as when a register configuration is incorrect. The log will also include basic information about *ModbusScope* and the system it is running on, such as the operating system and its specific version.

5.2 Logs window

Opening the *diagnostic logs* windows can be done with *Help* > *Diagnostic logs…*



```
00000000 - INFO [scope.general.info]: "App start 14/01/2023 17:11:58,877"
00000001 - INFO [scope.general.info]: "ModbusScope v3.7.0"
00000001 - INFO [scope.general.info]: "DEV git: various_changes:66f73e2"
00000002 - INFO [scope.general.info]: "Qt library v6.4.1"
00000002 - INFO [scope.general.info]: "OS: Windows 10 Version 22H2"
00000002 - INFO [scope.general.info]: "Locale language code: nl-BE"
00000002 - INFO [scope.general.info]: "Locale decimal separator: ,"
00000002 - INFO [scope.general.info]: "Locale group separator: ."
```

You should only enable this when you want to debug the connection. When enabled, all data reads are logged. This will increase the log size very much.

The logs can be viewed in the logs window. The log list will dynamically update as new logs are added. By using the filters, specific categories of logs can be hidden or shown. Specific logs can be selected and copied to the clipboard by right-clicking on them. Alternatively, all logs can be exported using the *Export Logs* button. Additionally, extensive debug logging can be enabled to log extra (internal) information about the status of every read. However, this option will dramatically increase the number of logs generated.

CHAPTER SIX

IMPORTING AND EXPORTING

6.1 Storing and reusing configuration

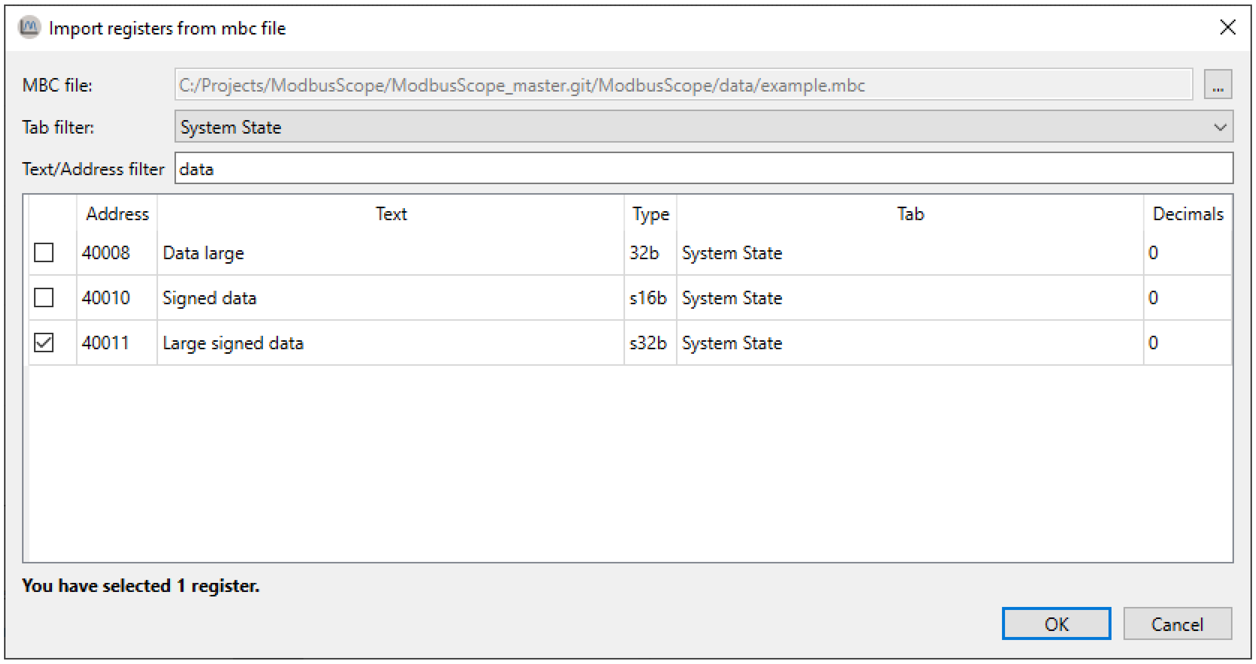
The configuration of registers, as described in previous sections, can be saved and loaded in a project file. These settings can be saved in a `.mbs` file by going to *File* > *Save Project As…* and loaded by going to *File* > *Open Project…* or by dragging a `.mbs` file into the application.

6.2 Exporting data/image

Current log results can be exported as an image or as data (`.csv`) file. You can select either *File* > *Save Data File As…* or *File* > *Export Image As…* to do so. It is important to note that saving a project/data file or exporting an image can only be done when logging is not active.

6.3 Import register definitions from *mbc* file

ModbusControl is an application that can be used to read and write data from Modbus slaves. It is possible to import the register definitions from a *ModbusControl* project file (`.mbc`) into *ModbusScope* by clicking on *Import from .mbc file* in the register dialog or by dragging and dropping the `.mbc` file into the main screen of *ModbusScope*. This makes it easy to add register definitions to *ModbusScope* from a *ModbusControl* project file.


Import registers from mbc file

MBC file: C:/Projects/ModbusScope/ModbusScope_master.git/ModbusScope/data/example.mbc

Tab filter: System State

Text/Address filter data

	Address	Text	Type	Tab	Decimals
☐	40008	Data large	32b	System State	0
☐	40010	Signed data	s16b	System State	0
✓	40011	Large signed data	s32b	System State	0

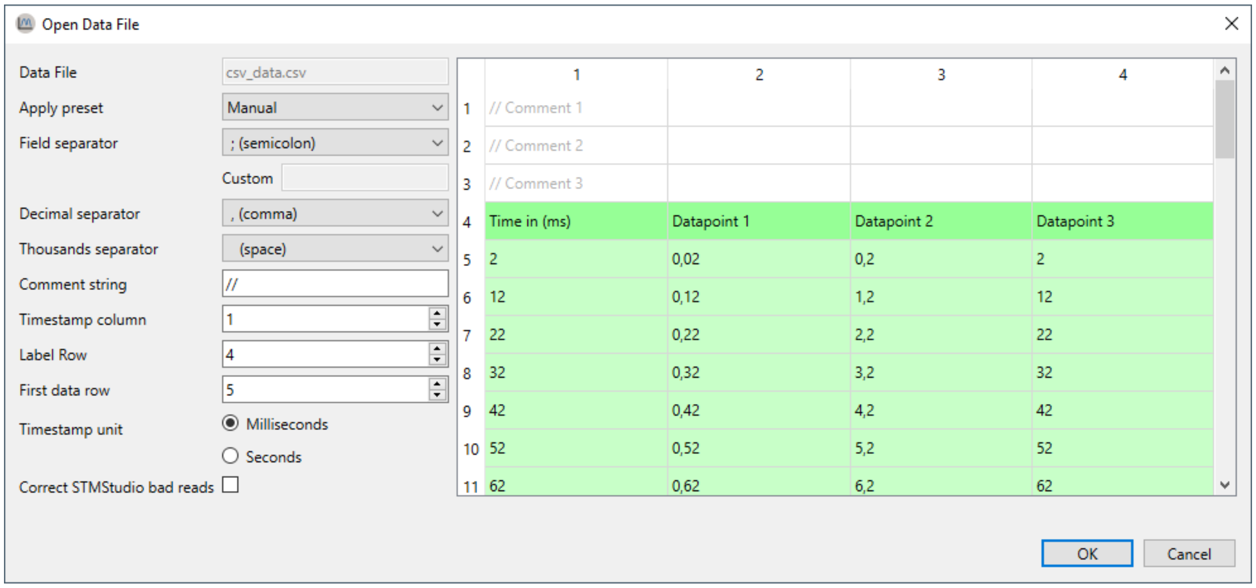
You have selected 1 register.

CHAPTER

SEVEN

OPEN DATA FILE

When a data file that was created by *ModbusScope* is opened, the data will be loaded automatically. This can be done by selecting *File > Open Data File...* or by dragging the file into the application. If the file format is not compatible with *ModbusScope*, the program will try to automatically determine the necessary settings for parsing the file. This includes the field and decimal separators. These settings can be adjusted manually if needed. Once the settings are configured, the rest of the file will be loaded and the data can be viewed in the graph as with a normal *ModbusScope* log.



Data File	csv_data.csv
Apply preset	Manual
Field separator	; (semicolon)
Custom	
Decimal separator	, (comma)
Thousands separator	(space)
Comment string	//
Timestamp column	1
Label Row	4
First data row	5
Timestamp unit	[✓ Milliseconds] Seconds
Correct STMStudio bad reads	☐

	1	2	3	4
1	// Comment 1			
2	// Comment 2			
3	// Comment 3			
4	Time in (ms)	Datapoint 1	Datapoint 2	Datapoint 3
5	2	0,02	0,2	2
6	12	0,12	1,2	12
7	22	0,22	2,2	22
8	32	0,32	3,2	32
9	42	0,42	4,2	42
10	52	0,52	5,2	52
11	62	0,62	6,2	62

7.1 Parse settings

The format of a `.csv` file can vary, so correct settings must be in place for parsing the file. These settings can be adjusted and the results of the parsing will be reflected in the grid display.

7.1.1 Locale related

Depending on the configured region, the format of the data will be different. To provide maximum flexibility when opening a data file, *ModbusScope* allows to freely select these settings.

- Field separator
  - A symbol used to separate data fields from each other.
- Decimal separator
  - A symbol used to separate the integer part from the fractional part of a number written in decimal form (e.g., "." in 12.45).
- Thousand separator
  - A symbol used to create a division between groups of digits. In this case

7.1.2 File structure related

ModbusScope is able to process and read data from a `.csv` file if it's formatted in a similar way to the files exported by *ModbusScope*. This includes having a column for timestamps and one or more columns for data.

Supported data file format

```
Time (ms);Datapoint 1;Datapoint 2;Datapoint 3
2;0,02;0,2;2
12;0,12;1,2;12
22;0,22;2,2;22
32;0,32;3,2;32
```

Settings

Since there is no standard for the contents of *.csv* file, some settings needs to filled in to be able to correctly parse time data.

- Comment String
  - Symbol(s) in the beginning of a line that indicate(s) that a line should be ignored when parsing.
  - Shouldn't be longer than 2 characters.
- Time stamp column
  - Sometimes the time data isn't in the first column. This setting can be used to select to correct column.
  - All columns to the right of the time data column will be parsed for data
- Label row
  - This setting indicates the row with the labels (names) for the graphs.

  - Can be used to skip header lines in the file.
  - The label row should contains the same number of fields as the data rows
- Data row
  - Similar to the the label row, but for the data

7.1.3 Functionality

ModbusScope adds some extra functionality when opening a data file. The *time-stamp unit* can be selected between milliseconds and seconds. When seconds is selected, the time-stamp will be converted to milliseconds during the load process.

Correct STMStudio bad read

This feature is only used for a very specific use case when loading a data file created with STMStudio. The STMStudio tool reads data directly from the RAM memory of a embedded devices. The main drawback is that reading a 16 bit variable can sometimes return a corrupt value on a 8 bit micro-controller. A single byte (most or least significant) of the 16 bit value will be reset to zero's or set to one's.

ModbusScope is able to detect most of these corrupt values and correct them based on previous and next value in the graph.

This feature will probably be deprecated and removed in a future release of *ModbusScope*.

7.2 Presets

When analyzing several data files of which the settings can't be auto-detected, it is handy to save the settings as a preset. *ModbusScope* allows to create a configuration file with custom presets. This configuration file will be loaded when opening a datafile and the correct preset can be selected. It is also possible to configure a keyword per preset. When a data file name contains the keyword, the preset will be automatically selected.

7.2.1 Locations

ModbusScope searches for the `presets.xml` configuration file in 2 specific locations. The first location is the documents folder of the current Windows user: `C:/Users/<USER>/Documents/`. The preset configuration file should be in a subfolder named `ModbusScope`. When the preset configuration file isn't found, *ModbusScope* will try to find the file in the same location as the main executable of *ModbusScope*. When the file isn't found in both location, *ModbusScope* will use the built-in presets. When a valid preset configuration file has been found, the built-in presets will be replaced will the preset mentioned in this file.

7.2.2 Keyword

As mentioned before, a preset can be automatically selected based on the presence of a keyword in the name of the data file.

7.2.3 Example preset configuration file

```
<modbusscope>
 <parsepreset>
  <name>Default (be)</name>
  <keyword>-be</keyword>
  <fieldseparator><![CDATA[;]]></fieldseparator>
  <decimalseparator><![CDATA[,]]></decimalseparator>
  <thousandseparator><![CDATA[ ]]></thousandseparator>
  <commentSequence><![CDATA[//]]></commentSequence>
  <column>1</column>
  <labelrow>1</labelrow>
  <datarow>2</datarow>
 </parsepreset>

 <parsepreset>
  <name>be-seconds</name>
  <fieldseparator><![CDATA[;]]></fieldseparator>
  <decimalseparator><![CDATA[,]]></decimalseparator>
  <thousandseparator><![CDATA[ ]]></thousandseparator>
  <commentSequence><![CDATA[//]]></commentSequence>
  <column>1</column>
  <labelrow>1</labelrow>
  <datarow>2</datarow>
  <timeinmilliseconds>false</timeinmilliseconds>
 </parsepreset>

</modbusscope>
```

CHAPTER

EIGHT

RELEASE NOTES

The latest *ModbusScope* installer or standalone version can always be downloaded from the release page.

8.1 v3.9.0 (04/12/2023)

8.1.1 Added

- Add support for other object types (discrete output coils, discrete input contacts, input registers, holding registers)

8.2 v3.8.1 (12/08/2023)

8.2.1 Added

- Allow in place editing of expressions

8.2.2 Changed

- Allow screenshot during logging (Github #280)

8.3 v3.8.0 (02/06/2023)

8.3.1 Added

- Improve visibility of errors (highlight curve in legend)

8.3.2 Fixed

- Fixed diagnostic log cleared when "extensive logging" is disabled (Github #265)

8.3.3 Changed

- Change add register dialog (wizard) to drop-down frame
- Keep focus of selected register when filtering during mbc import (Github #169)

8.4 v3.7.0 (03/02/2023)

8.4.1 Changed

- Rework toolbar (new icons and remove some actions)
- Update notification is now only visible after 14 days since release
- Improve marker indicators (Z-order)
- Select value axis scale options tab on axis selection (Github #253)
- Update dependencies

8.4.2 Added

- Add support for 32-bit floating point type (Github #250)
- Add indicator on axis to indicate value axis configuration of curve

8.5 v3.6.3 (21/11/2022)

8.5.1 Fixed

- Fix menu on incorrect screen (Github #248)

8.6 v3.6.2 (06/11/2022)

8.6.1 Fixed

- Fix marker data calculations (Github #240)
- Fix markers in combination with value axis (Github #240)

8.7 v3.6.1 (04/10/2022)

8.7.1 Fixed

- Fix zooming of axis with mouse wheel (Github #234)

8.8 v3.6.0 (02/10/2022)

8.8.1 Changed

- When starting only set scaling to auto when it is set to manual (Github #210)
- Improve import/export of csv data (especially when modifying file in Excel) (Github #220)
- Remove unused space in scale dock

8.8.2 Added

- Second Y-axis on right-side (Github #188)
- Add option to quickly add a register

8.8.3 Fixed

- Fix blurry tick labels in Qt6 build
- Fix crash when there is no data and marker is active (Github #223)
- Fix crash when a register is added and data is present (Github #229)

8.9 v3.5.1 (11/05/2022)

- Crash on showing tooltip with empty graphs (Github #208)
- Incorrect tab stop order when adding new registers (Github #209)
- Revert Windows build back to Qt5.15.2 to fix blurry tick labels

8.10 v3.5.0 (04/05/2022)

- Convert min/max settings of Y axis to floating point (Github #183)
- Add *Save Project* menu item
- Implement changing order of registers by drag and drop (Github #78)
- Allow scrolling to negative times when there is negative time data (Github #198)
- Update to Qt6

8.11 v3.4.0 (02/03/2022)

- Allow full 16-bit addressing of holding registers (Github #181)
- Expand expressions with register definition to provide more flexibility for user (Github project #6)
  - This changes adds the ability to combine multiple registers from potential multiple connections in one curve.
- Add syntax highlighting to expressions, including invalid token (Github #142)

8.12 v3.3.1 (23/12/2021)

- Select next register row after deletion in register dialog
- Improve note positioning when sliding window (Github #168)
- Hide markers on data load (Github #171)
- Fix opening project file via menu (Github #173)
- Let text filter also work on register number during mbc import (Github #174)

8.13 v3.3.0 (08/10/2021)

- Various user experience improvements
  - Replace check boxes with radio buttons
  - Rename menu items
  - Add extra menu items to tool bar
- Reload previous mbc file when import mbc dialog is opened (Github #156)
- Improve documentation (fixed Github #161 and open data file chapter)
- Fix that marker points are visible when graph isn't (Github #157)
- Include connection id in duplicate check when opening mbs file (Github #164)
- Updated used libraries and internal improvements

8.14 v3.2.1 (15/04/2021)

- Fix crash when starting ModbusScope with project file (Github #155)

8.15 v3.2.0 (03/04/2021)

Improvements

- Add more visible update notification
- Make documentation and project page (with issue) more visible (Github #152)

Fixes

- When data file is loaded, reset value in legend after inspection with `Control` key
- Reset scaling to auto when clearing data

8.16 v3.1.0 (23/02/2021)

Improvements

- Add support for modbus RTU (serial port)
- Add options to export diagnostic log
  - Copy specific logs to clipboard
  - Export complete log to file (Github #135)
- Minor graphical tweaks to markers

8.17 v3.0.0 (14/11/2020)

Improvements

- Replace fixed operations (multiply, divide, shift, . . .) with custom user-defined expression
  - Link to doc
  - Fixed operations will be automatically converted to custom expression on project load
- Rework and move the user manual to ReadTheDocs
- Expand information in logs
- Improve handling of large time periods (Github #139)
  - Don't wrap around when period is larger than one day

Backward compatibility When loading old ModbusScope files (pre v3.x.x), the existing operations (multiply, divide, shift, bitmask, . . .) will be converted to a single expression that is used in the new ModbusScope. This conversion makes sure that users won't notice any differences in functionality.

When exporting the settings (project file), the new expression will be saved. Older ModbusScope versions won't be able to parse this expressions. The operations will be reset to the defaults, but other register info will be loaded correctly.

8.18 v2.1.1 (03/07/2020)

- Fix update check (add OpenSSL dll to install) (Github #136)

8.19 v2.1.0 (15/06/2020)

Defects

- Fix tooltip

Improvements

- Add support for 32 bit registers (Github #129)
- Add support for persistent connection (default on) (Github #18)
- Minimize scale dock
- Disable bit mask for signed numbers
- Rework logging to be able improve logging in the future

8.20 v2.0.0 (03/03/2020)

Internal (code changes)

- Add more tests
- Fix most issues reported by static code analysis (Coverity)
- Change linking from static to dynamic
  - Application changes from one large executable to a smaller executable with extra dll's

Features

- Added possibility to poll 2 different slaves in the same log
- Added possibility to change graph color from legend
- Added filter (error/info) in diagnostics window
- Added mbc filter based on description / register number
- Toggle markers option
- Rectangle zoom
- Integrated most used features from GraphViewer
  - More flexible configuration of parsing settings
  - Advanced auto detection of parsed settings
  - Presets of parse settings
- Improved file loading/parsing
  - Improve file loading windows (keep showing raw data on invalid parse settings) (Github #120)
  - Speed up loading large data file (Github #121)
  - Add progress bar on file load (Github #122)

Bugs

- Small bug fixes (Github #111)

8.21 v1.6.1 (06/04/2019)

Bugfixes

- Fix error when writing notes to imported data file (Github #109)
- Automatically remove field separator from register names (Github #106)

Improvements

- When importing a mbc file already selected registers are now disabled dynamically.

8.22 v1.6.0 (25/01/2019)

Most of the work in this release isn't visible for the user. A complete rework of the communication module has been done. libmodbus was dropped in favor of Qt Modbus. The communication module has also been completely reworked to be able to implement integration and unit testing.

8.23 v1.5.0 (02/10/2018)

Features

- Add support for notes (small texts in graph)
  - Editable (even after data load)
  - Saved with data in csv
- Update legend component (thanks to @Fornax)
- Add tab filter when importing mbc files (implements Github #96)

Bug fixes

- Fix slow drag issue (Github #104)
- Keep visibility state when adding/removing graphs (Github #102)
- Absolute time is off by 2 hours (Github #103)

Under the hood

- Add initial unit tests for some modules
- Add Travis build

8.24 v1.4.0 (17/02/2018)

Features

- Rework Modbus communication code
- Update libraries
  - QCustomPlot v2.0.0 (final)
  - libmodbus to v3.14
- Improve support for absolute timestamp
- Add option to show/hide all graphs (Github #99)
- Add logging (Github #71)

Bugfixes

- Fix some minor bugs (Github #95) (thanks to @pluyckx), (Github #89)

8.25 v1.3.0 (01/04/2017)

Features

- Rework tooltip (show value under cursor in legend when control key is pressed) (Github #90)
- Import registers from mbc file (drag and drop or button in register dialog) (Github #91)
- Add window auto scale on y-axis (Github #36)
- Add meta data when exporting data (keep color when importing) (Github #63)
- Added extra marker calculations (minimum, maximum, average, median, slope, …) (Github #79)
- Use delete button to remove registers (Github #34)
- Improve communication (only split Modbus read on specific Modbus exception)
- Update QModbusPlot to v2.0.0 (beta)
- Add command line argument to enable OpenGL (–opengl)

Bugs

- Make sure legend window (when docked) is present on screenshot (Github #80)
- Small fixes (Github #82, Github #83, Github #85, Github #88)

8.26 Older releases

Older releases can be found on Github