



Modbench

User manual

A Modbus client for Windows, Linux and macOS: manual requests, device maps, charts, a simulator and tools to find and check devices.

Contents

1	Introduction	1
1.1	What you can do with it	1
1.2	A few Modbus words	2
1.3	The window	3
2	Installation and updates	4
2.1	Windows	4
2.2	macOS	4
2.3	Linux	4
2.4	Updates	5
2.5	Files Modbench opens	5
3	The Main page	6
3.1	Step 1: Connection	6
3.2	Step 2: Request	7
3.3	Address notation	8
3.4	Step 3: Test	8
3.5	Finding devices	9
4	Device maps	10
4.1	Opening a map	10
4.2	The Device map tab	11
4.3	Is it the right map?	12
4.4	Map status	12
4.5	Byte order override	12
4.6	Exporting and recording	13
4.7	What is remembered per device	13
5	Reading and writing	14
5.1	The Read registers tab	14
5.2	Writing values	14
5.3	Writing safely	15
6	Charts	17

7	Raw data and the log	19
7.1	The Raw data tab	19
7.2	The Log tab	20
8	The device library	21
8.1	Your library	21
8.2	Adding maps	22
8.3	The online library	22
9	The simulator	24
9.1	Where it listens	24
9.2	Devices	25
9.3	Values	25
9.4	Generators	26
9.5	Its own window	26
9.6	Saving a setup	26
10	Tools	28
10.1	Identify	28
10.2	Find addresses	29
10.3	Data decoder	30
10.4	Frame decoder	31
10.5	Trend player	31
10.6	Statistics	32
10.7	Alarms	33
11	Settings	35
11.1	About	35
11.2	Appearance	36
11.3	Values	36
11.4	Device library	36
11.5	Updates	37
11.6	Connection	37
11.7	What Modbench keeps	37

12 Command line and AI assistants	38
12.1 Commands	38
12.2 Generators file	39
12.3 AI assistants (MCP)	39
A Troubleshooting	40
A.1 No answer (timeouts)	40
A.2 Values look wrong	40
A.3 Bad frames	40
A.4 Exception codes	41
B Reference	42
B.1 Function codes	42
B.2 Byte and word order	42
B.3 Files	42
B.4 Keyboard	43

Introduction

Modbench is a Modbus client: it sends requests to Modbus devices (meters, inverters, drives, PLCs, controllers) and shows what they answer. It reads and writes registers by hand, or by name with a *device map*, charts values over time, shows every frame on the wire, and simulates devices when there is no hardware at hand.

It runs on Windows, Linux and macOS, speaks Modbus over serial lines (RTU and ASCII) and over the network (TCP, UDP, and RTU frames over TCP or UDP), and keeps everything on your computer.

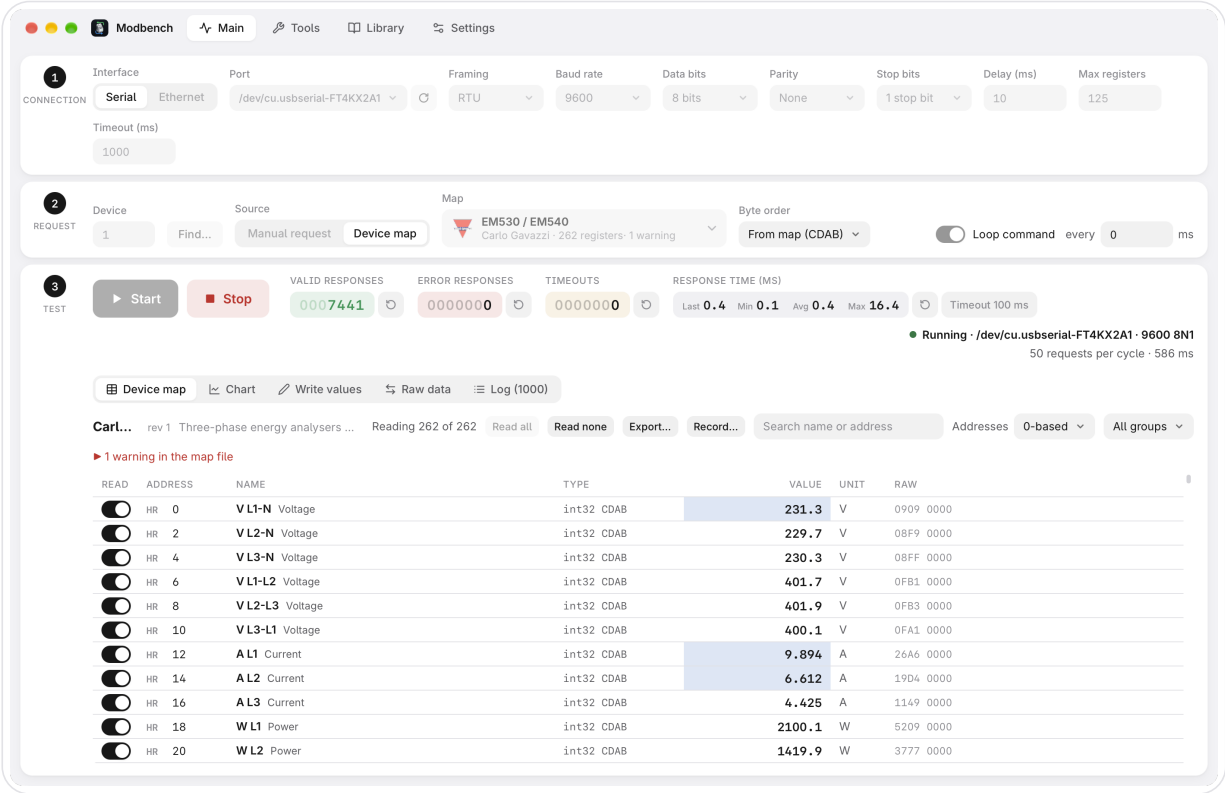


Figure 1. The Main page with a device map open.

1.1 What you can do with it

- **Send any request:** choose a function code, an address and a quantity, press **Start**, and see the values, once or in a loop.

- **Read a device by name:** open a map (`.mbmap.json`) and every register shows with its name, unit, scaling and state labels. Over 400 maps can be downloaded from the shared library.
- **Check that the map fits:** reference values catch a wrong address, base or byte order before you trust the other values.
- **Write safely:** typed controls, the map's limits, an optional confirmation, and a read-only mode that blocks every write.
- **See what happens:** charts, the raw bytes of each frame decoded field by field, a log of every request, and statistics of failures by cause.
- **Find and identify devices:** scan unit IDs and serial settings, find which addresses answer, ask a device who it is.
- **Simulate devices:** several devices shaped after maps, with values that move on their own, to try maps and other programs.

1.2 A few Modbus words

If you work with Modbus every day, skip this section.

Client and server

The client (Modbench) asks; the server (the device, also called slave) answers. A device never speaks first.

Unit ID

The address of a device on a serial line, or behind a gateway: 1 to 247. Over TCP to a single device it is often ignored, but must still be sent (usually 1 or 255).

Areas

Each device has four tables of data:

Area	Size	Read / write
Coils	1 bit	read (FC01), write (FC05, FC15)
Discrete inputs	1 bit	read only (FC02)
Input registers	16 bits	read only (FC04)
Holding registers	16 bits	read (FC03), write (FC06, FC16)

Address

The number of a coil or register in its area, from 0 to 65535 on the wire. Manuals often count from 1, or write the area in front (40001 is holding register 1, that is 0 on the wire). Modbench shows addresses the way you choose (Section 3.3).

Byte and word order

A register holds two bytes; 32- and 64-bit values take two or four registers. Devices disagree on which byte and which register comes first, so the same bytes can read as very different numbers. Maps say which order a device uses; the Data decoder (Section 10.3) helps find it.

Exception

An answer that says the request failed, with a code: 02 for an address the device does not have, 03 for a value it refuses, and so on (Appendix A.4).

1.3 The window

Four pages sit in the title bar. Click one to show it.

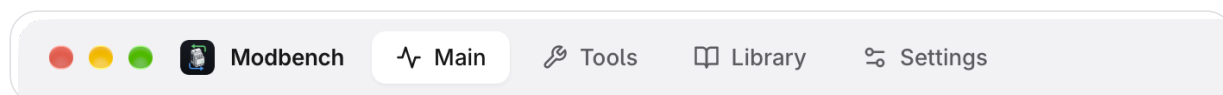


Figure 2. The title bar and its four pages.

Main

Connect to a device and test it, in three steps: *Connection*, *Request* and *Test* (Chapter 3).

Tools

The simulator, and tools to identify devices, find addresses, decode data and frames, play back recordings, see statistics and set alarms (Chapters 9 and 10).

Library

The device maps on this computer, and the shared online library (Chapter 8).

Settings

Theme, text size, safety, what Modbench remembers, updates (Chapter 11).

Right-click menus

Most lists and values have a right-click menu with what can be done there: copy a value or an address, add a register to the chart, write a value, export, remove. Text fields have Cut, Copy, Paste and Select all.

Light and dark

Modbench follows the system theme, or the one chosen in **Settings** > **Appearance**. The screenshots in this manual use the light theme.

Installation and updates

Installers are on the releases page: github.com/nobrega8/modbench/releases. Each release lists what changed.

2.1 Windows

Download `Modbench_x.y.z_x64-setup.exe` (recommended) or the `.msi`, and run it. The installer is about 2 MB: Modbench uses the WebView2 that comes with Windows.

2.2 macOS

Download `Modbench_x.y.z_universal.dmg`: one app for Apple Silicon and Intel Macs. Open it and drag Modbench to **Applications**.

First start

Modbench is not signed with an Apple Developer ID, so macOS asks before the first start. Right-click the app in **Applications**, choose **Open**, then **Open** again. Later starts need nothing.

2.3 Linux

- **AppImage**: download it, make it executable (`chmod +x Modbench*.AppImage`) and run it.
- **Debian, Ubuntu and derivatives**: install the `.deb`, or add the APT repository so that `apt upgrade` brings new versions:

```
curl -fsSL https://nobrega8.github.io/modbench/key.asc | sudo gpg --dearmor -o /usr/share/keyrings/modbench.gpg
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/modbench.gpg] https://nobrega8.github.io/modbench stable main" | sudo tee /etc/apt/sources.list.d/modbench.list
sudo apt update && sudo apt install modbench
```

Serial ports on Linux

To open serial ports without root, add your user to the `dialout` group (`sudo usermod -aG dialout $USER`) and log in again.

2.4 Updates

Modbench checks for a new version when it starts (this can be turned off in **Settings > Updates**). When there is one, a banner says what changed:

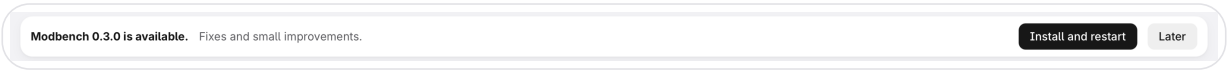


Figure 3. The update banner.

Install and restart downloads the update, checks its signature, installs it and starts Modbench again. **Later** hides the banner until the next start. **Settings > Updates** can also check by hand.

Installs from the APT repository update with the system (`sudo apt upgrade`) instead.

2.5 Files Modbench opens

The installers register two file types, each with its own icon. A double-click opens them in Modbench, also when it is already running.

File	What it is
<code>.mbmap.json</code>	A device map (Chapter 4). Opens on the Main page.
<code>.mbmap</code>	A device map in the older version 1 format; Modbench can convert it.
<code>.mbsim</code>	A simulator setup (Section 9.6). Opens in the simulator.

Note
Windows tells file types apart by their last extension only, so `.mbmap.json` files keep the icon of JSON files there. On Linux they get the map icon.

The Main page

The Main page tests a device in three steps, from top to bottom: *Connection* (how to reach it), *Request* (what to ask) and *Test* (send, and see the answers).

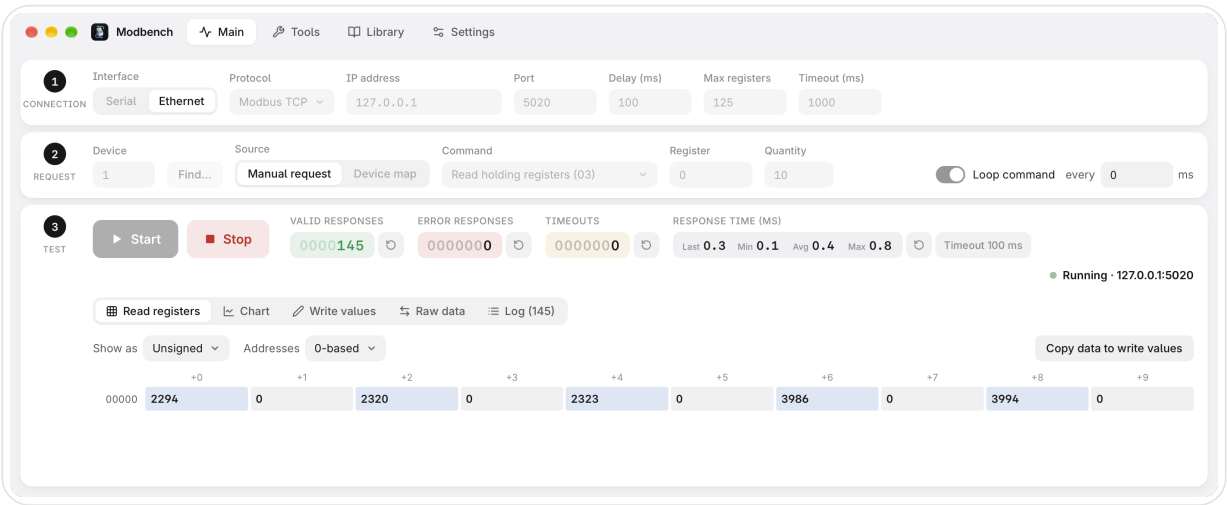


Figure 4. The Main page: Connection, Request and Test.

3.1 Step 1: Connection

Interface

Choose how the device is reached:

Interface	Use it for
Serial, RTU	RS-485 or RS-232 devices: the usual Modbus on a serial line.
Serial, ASCII	Older devices that send frames as text, checked with an LRC.
Modbus TCP	Devices and gateways on Ethernet (port 502).
Modbus UDP	Devices that answer Modbus TCP frames over UDP.
RTU over TCP	Serial-to-Ethernet converters that pass RTU frames as they are.
RTU over UDP	The same, over UDP.

SERIAL

Serial port lists the ports of this computer (🔄 looks again after plugging an adapter in). Set Baud rate, Data bits, Parity and Stop bits as the device is configured: every device on the line must use the same.

NETWORK

IP address and Port. The port is 502 unless the device or the converter uses another.

Device

Device (unit ID) is the device's address: 1–247 on a serial line or behind a gateway. Timeout (ms) is how long to wait for an answer before counting a timeout; Delay (ms) is the pause between two requests, which slow devices and some converters need.

Not sure of the settings?

Find... scans for unit IDs that answer, or, on a serial line, for the baud rate, parity and stop bits of a device (Section 3.5). The Statistics tool suggests a timeout from the measured response times (Section 10.6).

3.2 Step 2: Request

Source chooses between a *manual request* and a *device map*.

Manual request

Command

The function code: read coils (FC01), discrete inputs (FC02), holding registers (FC03) or input registers (FC04), or one of the writes (FC05, FC06, FC15, FC16).

Start address

The first address, in the notation chosen in Settings (below).

Quantity

How many coils or registers. Modbus allows up to 125 registers or 2000 bits per read.

Device map

With Source set to a map, Map chooses one of the maps of the library, an open file, or From a file.... Modbench then reads every register of the map by name (Chapter 4). The map's connection defaults (unit ID, baud rate, parity, port) fill step 1 when the map is picked; you can still change them.

Byte order

Byte order reads every value of a map with another byte and word order than the map says. It is meant for checking: if the values only make sense with another order, the map, or the device's configuration, is wrong.

Loop

With Loop command on, the request repeats until **Stop**. The time next to it is from the start of one cycle to the start of the next; 0 repeats as soon as Delay allows. It can be changed while the test runs. Without Loop, **Start** sends the request once.

3.3 Address notation

Manuals write addresses in many ways. The address menu on the **Device map** , **Read registers** and **Log** tabs chooses how Modbench shows them:

Notation	Holding register 100	Notes
0-based	100	The address on the wire.
1-based	101	As many manuals number registers.
Hex	0x0064	As in protocol documents.
Modicon	400101	Area digit in front (0, 1, 3, 4), 1-based.
IEC 61131-3	%MW100	As PLC programs write them.

Address fields also take the Modicon and IEC forms as typed, and switch the command to their area: typing 300010 reads input register 9.

3.4 Step 3: Test

Start sends the request (once, or in a loop); **Stop** ends it. The counters show Valid responses, Error responses (exceptions) and Timeouts; the button next to each counter sets it back to zero.

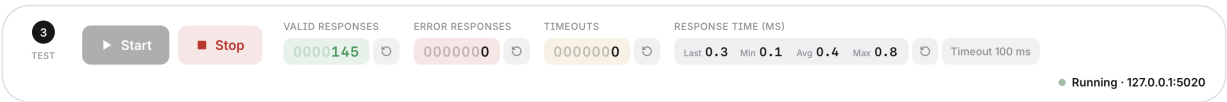


Figure 5. Start, Stop and the counters.

The tabs below show the results:

Device map or **Read registers**

The values: by register name with a map (Chapter 4), as a grid for a manual read (Chapter 5).

Chart

Values over time (Chapter 6).

Write values

Write coils and registers (Section 5.2).

Raw data

The bytes of each frame, decoded (Section 7.1).

Log

Every request with its result and time (Section 7.2).

When the link drops

With **Reconnect when the connection is lost** on (in Settings), a test that loses its link (a cable pulled, a gateway restarted) keeps trying to reconnect, and goes on where it stopped.

3.5 Finding devices

Find... in step 1 opens a scan:

Unit IDs

Asks every unit ID in a range (1–247 by default) and lists the ones that answer. Use a short **Wait (ms)** to go faster on a quiet line.

Serial settings

For a serial device whose settings are unknown: tries the chosen baud rates, parities and stop bits, most common first, until the device answers.

The screenshot shows a web-based configuration interface for a Modbus device. At the top, there are tabs for 'Device', 'Source', 'Command', 'Register', and 'Quantity'. Below these, there are input fields for 'Device' (1), 'Find...' (a button), 'Manual request' (a button), 'Device map' (a button), 'Read holding registers (03)' (a dropdown menu), 'Register' (0), and 'Quantity' (10). To the right, there is a toggle switch for 'Loop command' and a text field for 'every' (0) followed by 'ms'. Below this, there are two tabs: 'Unit IDs' and 'Serial settings'. The 'Unit IDs' tab is active, showing a table with columns 'From', 'To', and 'Wait (ms)'. The table has two rows: 'Unit 1' with 'From' 1, 'To' 10, and 'Wait (ms)' 300, and 'Unit 2' with 'From' 1, 'To' 10, and 'Wait (ms)' 300. There is a 'Scan' button next to the table. Below the table, there are two rows: 'Unit 1' with 'responds' and 'In use' buttons, and 'Unit 2' with 'responds' and 'Use' buttons. A 'Close' button is in the top right corner of the 'Unit IDs' tab.

Figure 6. Finding unit IDs and serial settings.

Use next to a unit ID that answered puts it in step 1.

Device maps

A device map describes the registers of a device model: their names, areas, addresses, types, byte and word order, scaling, units and state labels, plus the device's connection defaults. With a map, Modbench reads a device by name, as its manual describes it, instead of as a list of numbers.

Maps are `.mbmap.json` files in an open format. Over 400 of them, for meters, inverters, drives and controllers, are shared in the *mbmap-library* and can be downloaded from the Library page (Chapter 8).

4.1 Opening a map

- In step 2 of the Main page, set **Source** to the map, and pick it in **Map**: the maps of your library are listed there.
- **From a file...** opens any `.mbmap.json` or `.mbmap`.
- Double-click a map file: it opens in Modbench.
- From the Library page: right-click a map and choose **Use**.

Several maps can be open at once; step 2 switches between them. The map's connection defaults (unit ID, port, baud rate, parity) fill step 1 when it is opened.

Older maps

A `.mbmap` file in the version 1 format opens with a warning, and Modbench offers to convert it: the converted copy is saved next to it as `.mbmap.json`, and the original is kept.

4.2 The Device map tab

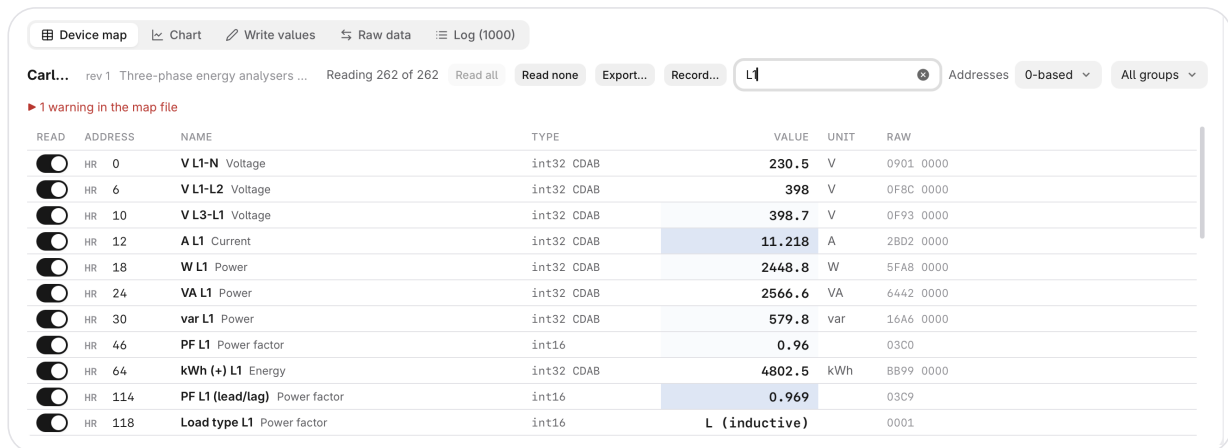


Figure 7. The Device map tab during a test.

Every register of the map is a row: its name, address, value with its unit, and its group. Values update with each cycle of the test.

Choosing what to read

Untick a register to stop reading it; **Read all** and **Read none** change them all. Fewer registers make each cycle shorter. Modbench joins neighbouring registers into as few requests as the device allows.

Search

Search name or address filters the list.

State labels

Registers that hold a state (0 Stopped, 1 Running...) show its label; reserved values the map declares (such as 0xFFFF for "not implemented") show their meaning instead of a number.

Bits and bit fields

Registers that pack several flags show each bit or field by its own name.

Changed values

A value that changes stays highlighted for a while (**Settings > Values**), so changes stand out in a long list.

Right-click

Copy a value, a name or an address, add the register to the chart, or write it.

Registers the map is missing

Add a register the map is missing... adds a register on this computer only: its address, type and name. It is read with the others and marked as added; it is not written into the map file.

4.3 Is it the right map?

A wrong map, base address or byte order gives values that look like numbers but mean nothing. Modbench checks in two ways:

Reference values

A map can give the value a healthy device holds in some registers (a model code, a fixed protocol identifier). Modbench reads them and shows *match*, *mismatch* or *failed* (no answer). A mismatch points to a wrong map or settings.

Models and firmware

When the map says which models and firmware versions it fits, and has registers that identify the device, Modbench compares them with the connected device. *Device recognised* confirms it; otherwise a banner says the device does not match the map. Once the firmware is known, registers that do not exist in that version are left out.

The connected device does not match this map. The map is for EM-200, EM-200-MID; the device reports 310. The map is for firmware 1 to 2; the device reports 3.2.
Values may be wrong. Writes ask for confirmation first.

Figure 8. A device that does not match the map.

Careful

After a mismatch, writes ask for confirmation even when confirmation is off.

4.4 Map status

Maps from the library carry a status, shown as an icon in the library and in the map picker:

Status	Meaning
Verified	Checked against a real device.
Unverified	Written from the manufacturer's documentation, not yet checked on a device.
Unofficial	Not from the manufacturer's documentation, for example reverse-engineered.

4.5 Byte order override

Byte order in step 2 reads every value with another byte and word order than the map says: *as the map says*, or one of the four orders. Use it to check a map whose values look wrong; it is remembered per device.

4.6 Exporting and recording

Export...

Saves the values shown, in the order shown, to a CSV file.

Record...

Appends every cycle of the test to a CSV file until recording stops: one row per cycle, one column per register. The Trend player plays such a file back (Section 10.5).

The CSV separator is set in **Settings** (comma or semicolon, for spreadsheets in languages that use a decimal comma).

4.7 What is remembered per device

For each device (map, connection and unit ID), Modbench remembers which registers were switched off, the byte order override and Loop. They come back when you return to that device; restoring them never writes and never starts a test. **Settings** > **Connection** turns this off, and forgets what was saved.

Reading and writing

5.1 The Read registers tab

With a manual request, the values come back as a grid: one cell per coil or register, with its address.



Figure 9. Holding registers read as 32-bit values.

Show as

Shows the registers as unsigned or signed 16-bit numbers, hexadecimal, binary, ASCII text, 32-bit signed or unsigned integers, or 32-bit floats, in the byte order chosen next to it. When the right type is not known, the Data decoder shows them all at once, 64-bit types included (Section 10.3).

Bits

Shows the 16 bits of each register. A bit of a holding register can be switched on its own: Modbench writes the register with only that bit changed.

Addresses

The address menu chooses the notation (Section 3.3).

Right-click

Copy a value, an address or all values, add the address to the chart, or **Write here...** and **Write all....**

5.2 Writing values

The **Write values** tab writes coils and registers.

Without a map

Choose the function (single or multiple coils, single or multiple registers), the start address and the values, then **Write**. **Copy data to write values** fills the list with the values last read, to change one and write them back.

With a map

The map's writable registers are listed by name, each with a control for its type: a number in engineering units, a list of the map's labels for state registers, a switch for bits. Modbench applies the inverse of the scaling, checks the type's range and the map's limits, and only then writes.

ADDRESS	NAME	CURRENT VALUE	NEW VALUE
HR 20488	Name Device Configurable name, 16 characters		Text Write
HR 4096	Password Setup 0 = no password, up to 9999	0	Value Write
HR 4098	Measuring system Setup 3Pn = 3-phase with neutral, 3P = without neutral, 2P = 2-phase with neutral. MID models accept only 0 and 1	3P	3P (1) Write Wrote 3P
HR 4099	CT ratio Setup 1.0 to 2000.0. Not available on AV2 models; read-only on MID models	0	Value Write
HR 4112	DMD integration time Setup	1 min	Choose... Write
HR 4116	Alarm enable Alarm	Disabled	Choose... Write

Figure 10. Writable registers of a map; one mode just written.

Limits

A value outside the map's limits is refused before anything is sent, with the limit it broke.

Bit fields

Writing a bit or a field reads the register first and changes only its bits.

Custom values

For a register with labels, a value that is not one of them can still be typed.

Templates

A template copies the settings of one device to another of the same model. saves the values last read from the writable registers; writes the values of a saved template to the device shown.

5.3 Writing safely

Writes change how a real machine behaves. Modbench has three guards, in **Settings > Connection**:

Confirm before writing to a device

Asks before every write: write commands, map values and templates.

Read-only mode

Blocks every write, also on a connection that is already open. Step 3 shows a lock while it is on; click it to go to the setting.

Map mismatch

After a device does not match its map, writes always ask first.

Careful

A write is sent as soon as it is confirmed. Check the unit ID: on a serial line, unit ID 0 is a broadcast that every device takes.

Charts

The **Chart** tab plots values over time while a test runs: map registers by name, or addresses of a manual read.

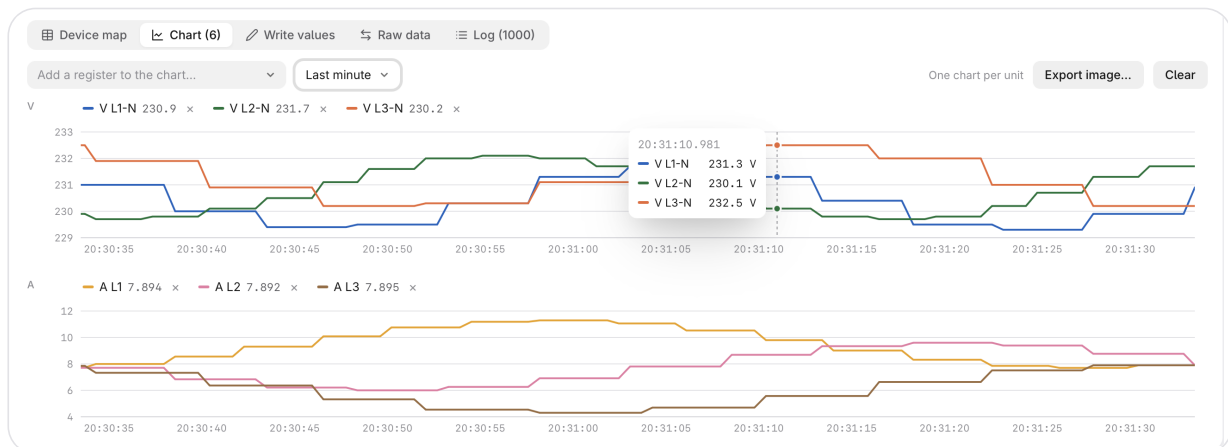


Figure 11. Voltages and currents of a meter, one chart per unit.

6.1 Adding lines

- With a map: **Add a register to the chart** lists the map's numeric registers.
- With a manual read: add an address from the list, or right-click a cell on the **Read registers** tab and choose **Add to chart**.

Up to eight lines can be shown, each in its own colour. Lines with the same unit share a chart, so values of different sizes (volts and amperes) do not flatten each other. The tab's name counts the lines: **Chart (3)** .

Values are recorded from the moment a line is added, while the test runs. To remove a line, click **x** next to its name in the legend.

6.2 Reading the chart

Time span

Time span shows the last minute, 5 or 15 minutes, the last hour, or everything since the lines were added.

Legend

Each line’s name and its newest value.

Hover

Move the pointer over a chart to see every line’s value at that moment.

Spikes

When there are more samples than pixels, each pixel keeps its lowest and highest value, so short spikes stay visible.

Gaps

A failed read leaves a gap in the line instead of a false value.

Clear empties the charts and keeps the lines.

6.3 Saving a chart as an image

Export image... saves the charts shown, for a report or a bug report:

Choice	What you get
PNG...	A picture at twice the screen size, sharp in documents.
SVG...	A vector drawing that scales to any size and opens in other programs (colours written out, no styles needed).
PNG, light theme...	As PNG, in the light theme even when Modbench is dark.
SVG, light theme...	As SVG, in the light theme.

The image holds every chart shown, each with its unit and legend, under a line with the date and the time span. The hover marks are left out. The image is taken when you choose, so it does not move on while the save dialog is open.

The Trend player (Section 10.5) has the same button.

Raw data and the log

When a device answers strangely, the frames tell why. These two tabs show what was really sent and received.

7.1 The Raw data tab

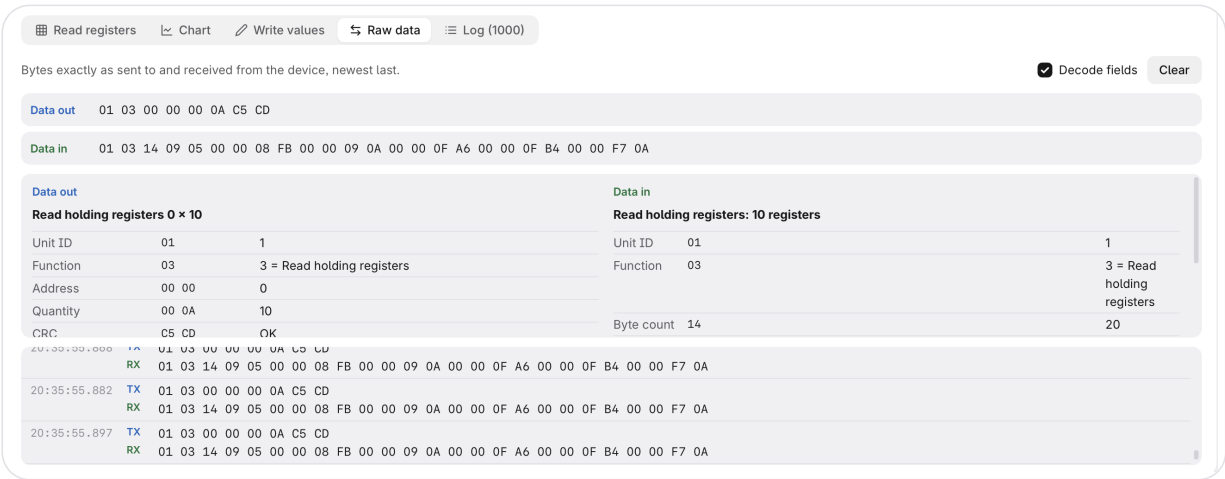


Figure 12. A request and its answer on the Raw data tab.

The bytes exactly as sent to and received from the device, newest last: **Data out** in blue, **Data in** in green, as in the logo. Each frame is decoded field by field (unit ID, function code, address, quantity, byte count, data), and its check is verified: the CRC on RTU, the LRC on ASCII, the header on TCP and UDP. A frame that fails its check is marked.

When no answer came, the tab says so (*No response.*). **Clear** empties it.

Tip

To read a frame from another program or a bus analyser, paste it into the Frame decoder (Section 10.4): it decodes it the same way.

7.2 The Log tab

Read registers Chart Write values Raw data Log (182)					
Errors only Addresses 0-based ▾					
Clear log Save log...					
TIME	FUNCTION	ADDRESS	QTY	DURATION	RESULT
20:36:42.319	03 Read holding registers	0	10	0.2 ms	OK
20:36:42.331	03 Read holding registers	0	10	0.4 ms	OK
20:36:42.342	03 Read holding registers	0	10	0.3 ms	OK
20:36:42.354	03 Read holding registers	0	10	0.3 ms	OK
20:36:42.368	03 Read holding registers	0	10	0.4 ms	OK
20:36:42.380	03 Read holding registers	0	10	0.4 ms	OK
20:36:42.391	03 Read holding registers	0	10	0.4 ms	OK
20:36:42.403	03 Read holding registers	0	10	0.3 ms	OK
20:36:42.417	03 Read holding registers	0	10	0.3 ms	OK
20:36:42.428	03 Read holding registers	0	10	0.3 ms	OK
20:36:42.439	03 Read holding registers	0	10	0.3 ms	OK
20:36:42.453	03 Read holding registers	0	10	1001.4 ms	timed out after 1000 ms
20:36:43.470	03 Read holding registers	0	10	1002.6 ms	timed out after 1000 ms
20:36:44.488	03 Read holding registers	0	10	1002.7 ms	timed out after 1000 ms

Figure 13. The Log tab, with three timeouts in red.

Every request: Time, Function, Address, Result (OK, the exception, or a timeout) and Duration. The tab's name counts them: **Log (120)** .

Save log...

Saves the log to a CSV file.

Clear log

Empties it.

Right-click

Copy line, Copy error, Copy all, and Show errors only to keep just the failed requests in view.

Size

Settings > Connection > Requests kept in the log sets how many requests are kept; older ones are dropped first.

For failures by cause over a long test, see Statistics (Section 10.6).

The device library

The Library page keeps your device maps in one folder, and brings maps from the shared online library.

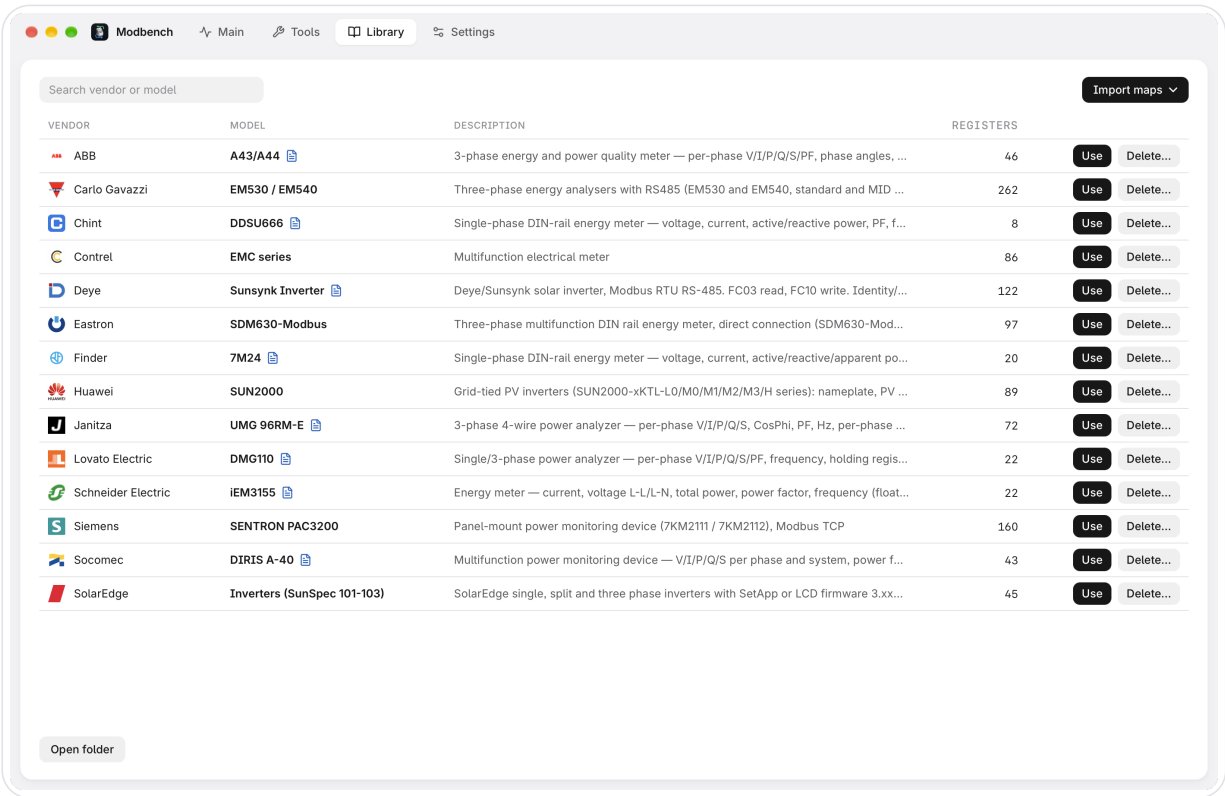


Figure 14. The Library page.

8.1 Your library

The list shows each map’s vendor (with its icon), model, description, number of registers and status (Section 4.4). Search vendor or model filters it.

Right-click a map:

Use

Opens it on the Main page.

Simulate this device

Opens the simulator with a device shaped after this map (Chapter 9).

Copy name, Copy file path

For notes and other programs.

Open the library folder

Shows the folder in your file manager.

Delete...

Removes the map from the library, after asking.

Maps with errors are listed with the problem, so they can be fixed in a text editor.

8.2 Adding maps

From a file...

Copies .mbmap.json or .mbmap files into the library. Version 1 files are converted.

From the online library...

Opens the shared library (below).

The library is a folder (Settings > Device library > Library folder): files copied into it by hand appear too.

8.3 The online library

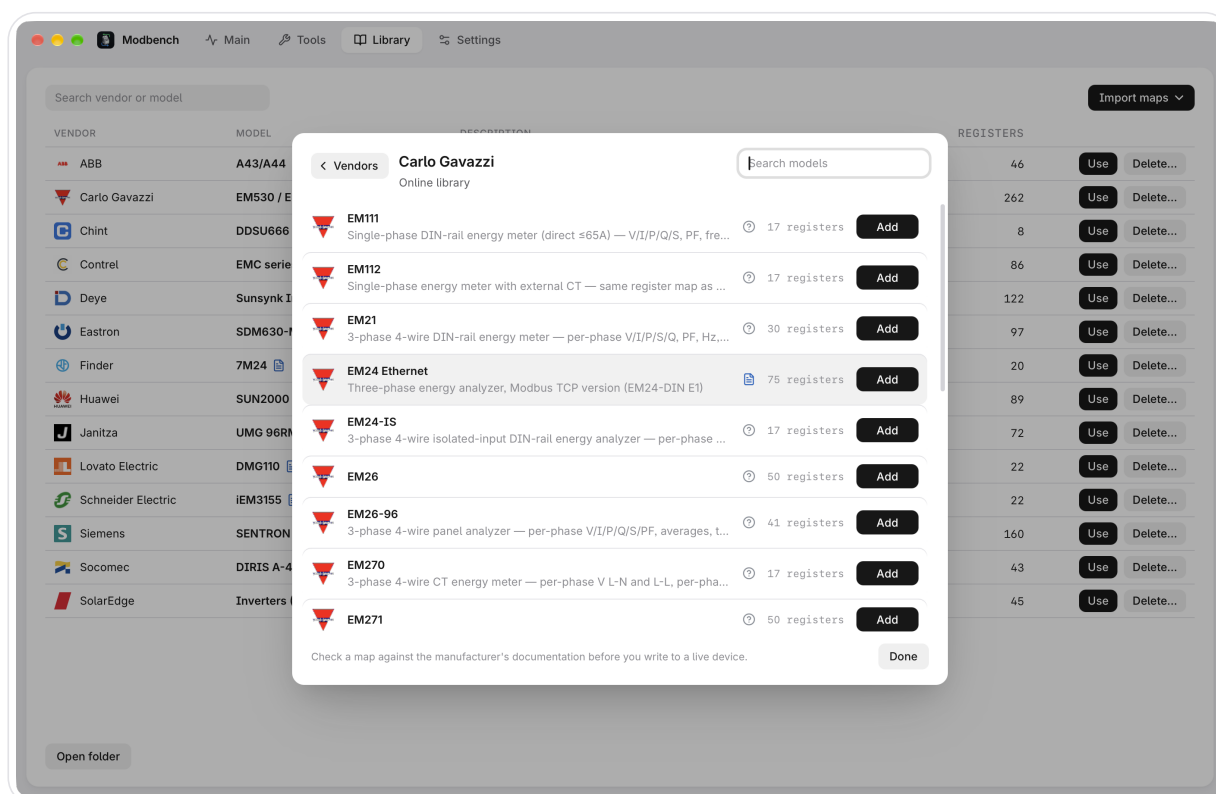


Figure 15. Choosing a map in the online library.

The online library lists the maps shared on GitHub (github.com/nobrega8/mbmap-library): over 400 maps of meters, inverters, drives and controllers.

- 1 Choose a vendor; **Back to all vendors** (or **Esc**) goes back.
- 2 **Add** next to a model downloads its map into your library.
- 3 **Done** closes the online library.

A map already in your library can be downloaded again, to replace your copy with the newest one.

Share your maps

A map that works with a real device helps the next person with the same device. The library's repository explains how to contribute one, and the format specification is linked from **Settings** › **Device library**.

The simulator

The simulator runs Modbus devices inside Modbench. Use it to try a map, an alarm or another program (a SCADA, a PLC, a script) without the hardware: each device answers like the real one, with the values of its map, and can make values move on its own.

Open it from Tools > Simulator.

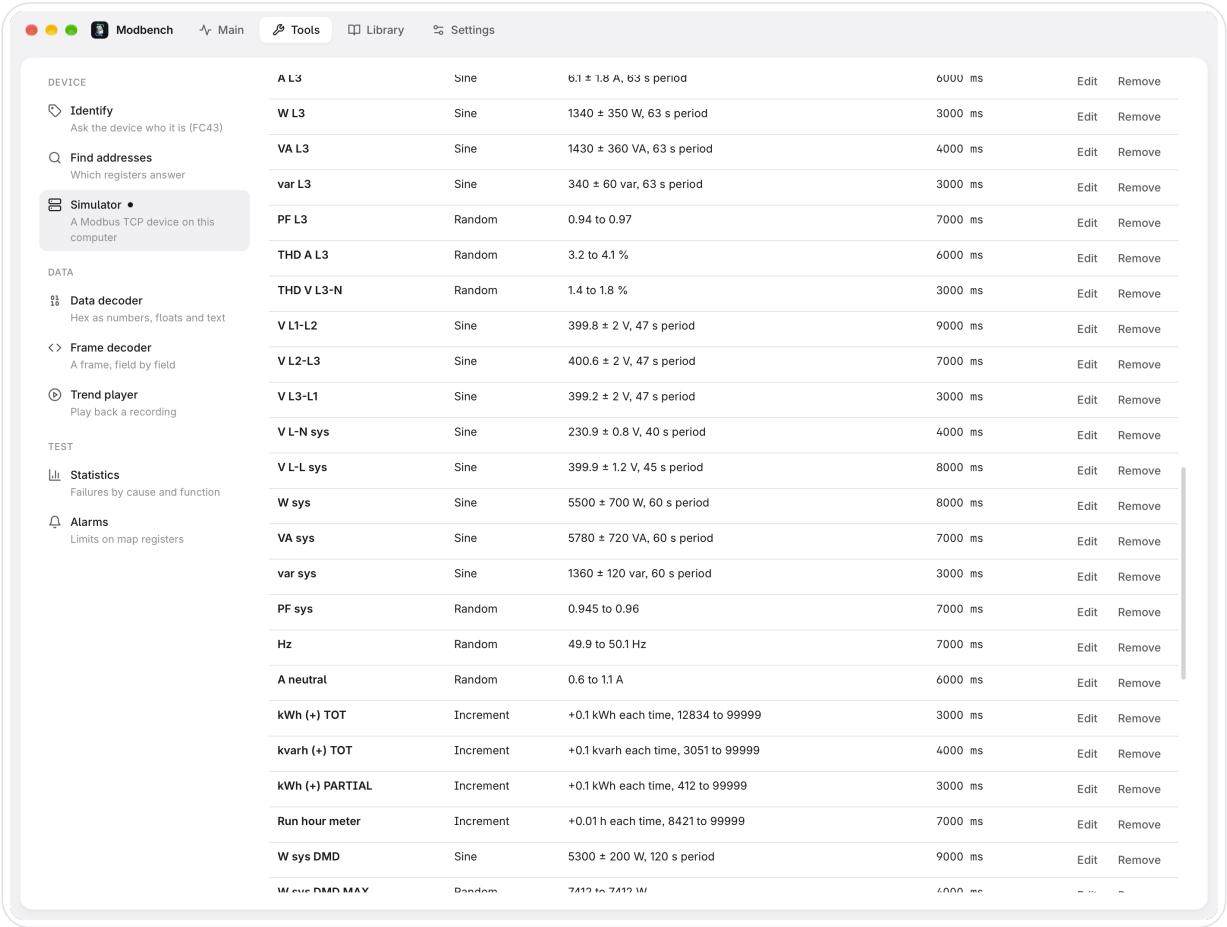


Figure 16. The simulator with two devices on a serial port.

9.1 Where it listens

Interface

Modbus TCP or UDP, RTU over TCP or UDP, or a serial port with RTU or ASCII. Over the network, set the Port (5020 by default, so it does not need administrator rights); Other

computers also accepts clients from the network, not only from this computer. On a serial port, set the line settings as the clients use them.

Start and **Stop**

Start listens; the status shows where (*Listening on 127.0.0.1:5020*).

Connect Main to it

Fills step 1 of the Main page with the simulator's address and the shown device's unit ID, and goes to Main.

Serial

Clients connect to the other end of the line. To try it on one computer, use a virtual null-modem pair (com0com on Windows, socat on Linux and macOS) and open the other port on the Main page.

9.2 Devices

The simulator starts with one device. Several devices can share one address, each on its own unit ID, like devices behind a gateway or on one RS-485 line; other unit IDs get no answer.

Add device

Adds a device on the lowest free unit ID. The tabs above switch between devices.

Unit ID

The unit ID the device answers. Empty answers every unit ID that no other device has. Unit ID 0 is a broadcast: every device takes the write and none answers.

Map

Shapes the device after a map: one open on the Main page, one from the library, or **Open a map file...** *Empty device* gives a memory where every address exists and holds zero.

Only the map's addresses

Addresses the map does not have answer exception 2, like the real device. Off, every address answers.

Reset values

Puts back the map's reference values.

Remove device

Removes the device shown; its unit ID gets no answer.

9.3 Values

The view switch shows the device's memory either as **Registers**, the map's registers by name with their values in engineering units, or as a raw **Grid** of addresses, in any area. Click a value to change it: clients read the new value from the next request.

Writes from clients show up immediately, and the recent requests are listed at the bottom, reads and writes apart, so a stream of reads does not push the writes out of sight.

9.4 Generators

Generators make values change on their own, every so many milliseconds (at least 50).

Add generator adds one; it drives a map register, in its own units, or a raw address.

Kind	What it does
Increment	Adds a step each time, from a minimum to a maximum, then starts again: counters, energy.
Sine	Swings around an offset with an amplitude and a period: voltage, temperature.
Random	A new value between a minimum and a maximum each time: noise.
Copy	Takes the value of another address: a status input that follows a command coil.

GENERATORS Values that change on their own while the simulator runs.				Add generator	
VALUE	KIND	CHANGES	EVERY	Edit	Remove
V L1-N	Sine	230 ± 5 V, 10 s period	500 ms	Edit	Remove
kWh (+) TOT	Increment	+0.1 kWh each time, 2210 to 99999	1000 ms	Edit	Remove
V L1-N (by phase)	Copy	Follows V L1-N	500 ms	Edit	Remove
Active energy import	Sine	the loaded map has no register "Active energy import"	1000 ms	Edit	Remove

Figure 17. Generators on a meter's registers.

A generator that cannot run (its register is not in the map, or its address does not exist) says why. Generators keep running across **Stop** and **Start**.

9.5 Its own window

Pop out moves the simulator into a window of its own, to watch it next to the Main page.

Back to the main window brings it back to the Tools page.

9.6 Saving a setup

A test bench with several devices, maps and generators takes a while to set up.

Save setup... keeps it in a .mbsim file:

- the interface and its settings;
- every device: unit ID, map, *Only the map's addresses*, and generators.

Open setup..., or a double-click on the file, puts it back. Each map is read again from its file when that is still there, so updates to the library are picked up; otherwise the copy kept in the setup is used. While the simulator runs, opening a setup replaces the devices but keeps the interface.

Note

Values changed by hand are not saved: devices start from their maps' reference values.

Tools

The Tools page groups the tools by what they work on: the *device*, the *data*, and the *test*. A dot next to a tool means it has something going on (the simulator running, a scan in progress).

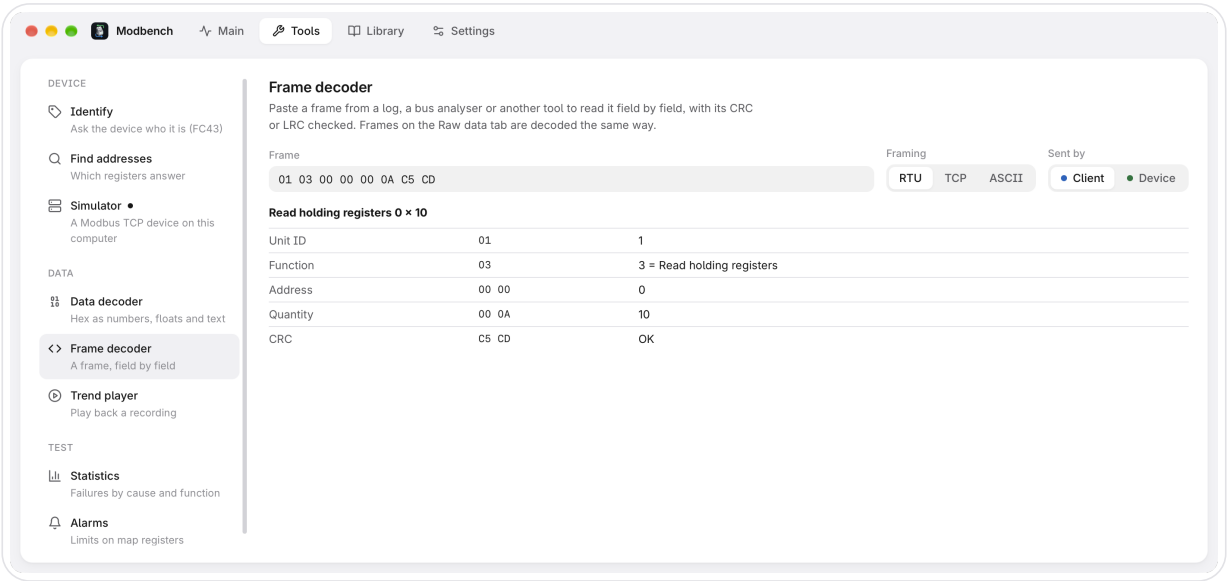


Figure 18. The Tools page.

Tools that talk to a device use the connection of step 1 on the Main page, and only read.

10.1 Identify

Asks the device for its vendor, product code and revision with *Read Device Identification* (FC43/14). Choose which **Objects** to ask for (basic, regular or extended) and **Read identification**. Each object the device returns is listed with its value.

Identify the device
 Asks the device for its vendor, product code and revision with Read Device Identification (FC43). Uses the connection of step 1 on the Main page: /dev/cu.usbserial-FT4KX2A1, unit 1. Reads only.

Objects

Basic
 Regular
 Extended
 Read identification

ID	OBJECT	VALUE
0x00	Vendor name	Carlo Gavazzi
0x01	Product code	EM530 / EM540
0x02	Revision	1.0

Conformity 0x01: basic objects, stream access only.

Figure 19. A device that answered FC43.

Not every device supports FC43: many answer exception 1 (illegal function).

Check function codes sends one harmless request for each read function (FC 1–4, 7, 8, 11, 17, 20, 24 and 43) and shows how the device answered: *Supported*; *Not supported* when it answered “illegal function”; *Supported, refused this request* when it answered another exception, which means it has the function but not, for example, address 0; or *No answer*. Nothing is written. FC 7, 8, 11 and 17 exist on serial lines only, so over TCP they often answer “illegal function”.

The functions are tried at the **Address** given (0 at first). With **Also try write functions**, Modbench also tries FC 5, 6, 15, 16 and 23 by reading coil and holding register **Address** and writing the same values back, and FC 22 with a mask that keeps every bit. It always asks first, and the option is off in read-only mode. A write whose value could not be read is *Not tried*. The values do not change, but a device can still act on a write: only do this on a device you may change.

Find largest read finds how many registers or bits the device returns in one read from **Address**, by halving, up to 125 registers or 2000 bits, and says why one more failed. “Illegal data value” means that is the device’s limit: set it as **Max registers** on the Main page. “Illegal data address” means the area has a gap there.

10.2 Find addresses

Finds which addresses answer on a device without a manual. It reads blocks of addresses, and splits the blocks that fail until it knows which addresses exist.

Area, From, To

What to search.

Block

Addresses read at once before splitting. Larger blocks are faster on devices with long runs of registers.

Wait (ms)

How long to wait for each answer.

The result lists the ranges that answered. **Read** next to a range sets up a manual request for it on the Main page.

Find addresses

Reads blocks of addresses and splits the ones that fail until it knows which addresses answer, so a device without a manual can still be mapped. Uses the connection of step 1 on the Main page. Reads only.

Area

From

To

Block

Wait (ms)

Scan

Addresses

0-based

0

2000

362 of 2001 addresses answered, in 22 ranges.

FROM	TO	COUNT	
HR 0	73	74	Read
HR 78	93	16	Read
HR 110	121	12	Read
HR 130	135	6	Read
HR 138	143	6	Read
HR 146	165	20	Read
HR 172	183	12	Read

Figure 20. Address ranges found on an energy meter.

10.3 Data decoder

Type or paste registers (in hex) to see them as every type at once, in all four byte orders: signed and unsigned 16-, 32- and 64-bit integers, floats, and text. **Use last read** takes the registers last read on the Main page.

This is the quickest way to find a value’s type and byte order before writing a map: look for the column where the number makes sense (a voltage near 230, a frequency near 50).

Data decoder

Type or paste registers to see them as every type at once, in all four byte orders. Useful to find a value's type and byte order before writing a map. Works without a connection.

Registers

4366 6666

Typed as

Hex

Decimal

Use last read

	ABCD	CDAB	BADC	DCBA
32-bit signed	1130784358	1717977958	1715693158	1717986883
32-bit unsigned	1130784358	1717977958	1715693158	1717986883
32-bit float	230.4	2.718469e+23	2.306876e+23	2.720077e+23
Text (ASCII)	Cfff	Cfff	fCff	fCff

ABCD is the Modbus standard. CDAB swaps the registers, BADC the bytes in each register, DCBA both. 32-bit values use the first two registers, 64-bit the first four.

#	HEX	UNSIGNED	SIGNED	BYTES SWAPPED	BINARY	BCD
0	4366	17254	17254	26179	0100 0011 0110 0110	4366
1	6666	26214	26214	26214	0110 0110 0110 0110	6666

Figure 21. Two registers decoded in every type and order: only 32-bit float in ABCD order reads as a voltage, 230.4.

It works without a connection.

10.4 Frame decoder

Paste a frame from a log, a bus analyser or another tool (for example 01 03 00 00 00 0A C5 CD) to read it field by field, with its CRC or LRC checked. Choose the Framing (RTU, ASCII, TCP) and who sent it (Client or Device), since a request and an answer with the same function code have different fields.

Frame decoder

Paste a frame from a log, a bus analyser or another tool to read it field by field, with its CRC or LRC checked. Frames on the Raw data tab are decoded the same way.

Frame

01 03 00 00 00 0A C5 CD

Framing

RTUTCPASCII

Sent by

Client

Device

Read holding registers 0 × 10

Unit ID	01	1
Function	03	3 = Read holding registers
Address	00 00	0
Quantity	00 0A	10
CRC	C5 CD	OK

Figure 22. An RTU request decoded.

10.5 Trend player

Plays back a CSV file made with **Record...** on the Device map tab, so a test can be looked at again later.

- 1 **Open recording...** and choose the file.
- 2 Tick the columns to plot (up to eight; text columns cannot be plotted).
- 3 **Play** at 1× to 60× speed, or drag the slider to any moment.

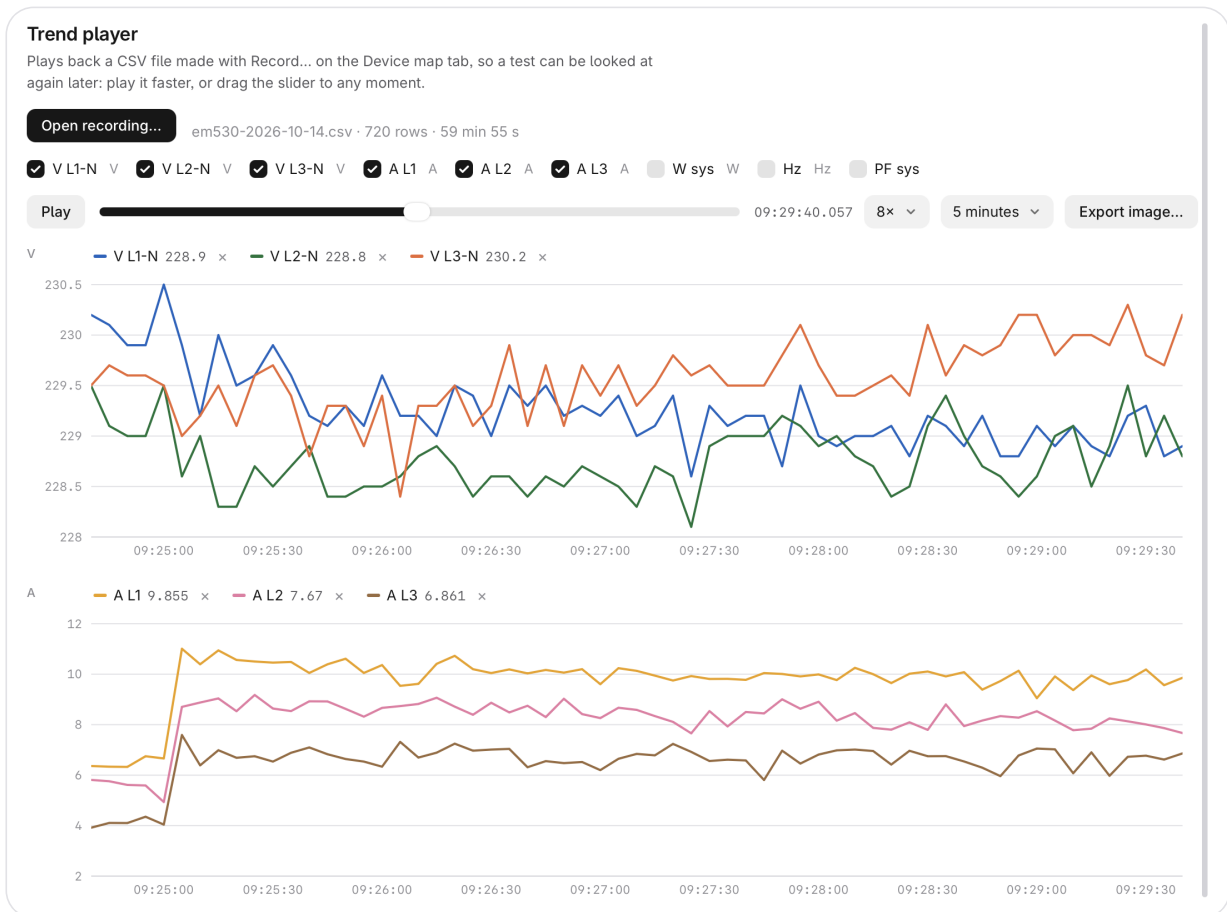


Figure 23. A recording played back.

Time span works as on the Chart tab, and **Export image...** saves the charts (Section 6.3).

10.6 Statistics

Every request sent on the Main page since the last **Reset**:

By result

How many succeeded, timed out, came back as each exception, or arrived as bad frames.

By function

Requests, failures, and minimum, average and maximum response time per function code.

A pattern points to the cause: timeouts to wiring, baud rate or unit ID; exception 2 to an address the device does not have; bad frames to noise or two devices with the same unit ID.

Statistics

Every request sent on the Main page since the last reset, by result and by function code. A pattern points to the cause: timeouts to wiring, baud rate or unit ID; exception 2 to an address the device does not have.

12438 requests

Reset

BY RESULT

OK		9102	73.2%
Exception 2: illegal data address		3330	26.8%
Timeout (no answer)		6	<0.1%

BY FUNCTION

FUNCTION	REQUESTS	FAILED	MIN MS	AVG MS	MAX MS
3 Read holding registers	12437	3336	0.1	0.4	8.0
43 Encapsulated interface (device identification)	1	0	0.3	0.3	0.3

Times count answers only (also exceptions): a timeout says nothing about how fast the device is.

SUGGESTED TIMEOUT

100 ms 4 × the slowest answer (8.0 ms), at least 100 ms. Step 1 uses 1000 ms.

Use in step 1

Figure 24. Statistics of a long test.

Suggested timeout is four times the slowest answer; **Use in step 1** puts it in the connection settings.

10.7 Alarms

Limits on the registers of the open map. A value below Low or above High, or a bit in its alarm state, turns red on the Device map tab.

- 1 Open a map on the Main page.
- 2 **Add an alarm on a register...** and choose the register.
- 3 Set the limits, or for a bit, the state that is an alarm.

Alarms

Set limits on the registers of the open map. A value below Low or above High, or a bit in its alarm state, turns red on the Device map tab. Limits are kept per map on this computer, and nothing is sent to the device.

Add an alarm on a register...

REGISTER	VALUE	ALARM WHEN	
V L1-N Voltage	231.7 V	below 220 or above 240 V	Remove
A L2 Current	9.5 A	below — or above 6 A	Remove
W L1 Power	2357.5 W	below — or above 2500 W	Remove

Figure 25. Alarms on a meter's map.

READ	ADDRESS	NAME	TYPE	VALUE	UNIT	RAW
☐	HR 0	VL1-N Voltage	int32 CDAB	231.5	V	090B 0000
☐	HR 2	VL2-N Voltage	int32 CDAB	231.3	V	0909 0000
☐	HR 4	VL3-N Voltage	int32 CDAB	231.6	V	090C 0000
☐	HR 6	VL1-L2 Voltage	int32 CDAB	398.9	V	0F95 0000
☐	HR 8	VL2-L3 Voltage	int32 CDAB	400.2	V	0FA2 0000
☐	HR 10	VL3-L1 Voltage	int32 CDAB	397.7	V	0F89 0000
☐	HR 12	AL1 Current	int32 CDAB	8.732	A	221C 0000
☐	HR 14	AL2 Current	int32 CDAB	9.056	A	2360 0000
☐	HR 16	AL3 Current	int32 CDAB	7.508	A	1D54 0000
☐	HR 18	WL1 Power	int32 CDAB	1885.1	W	49A3 0000

Figure 26. The value in alarm, in red on the Device map tab.

Alarms are kept per map on this computer. Nothing is sent to the device.

Settings

Settings are kept on this computer and apply at once. The list on the left groups them as on the Tools page: *Modbench* (About, Updates), *Display* (Appearance, Values) and *Devices* (Connection, Device library).

11.1 About

The version of Modbench and whether it is up to date, with [Check for updates](#). Below it are links to this manual, the release notes, a form to report a problem, the source code and the project's Ko-fi page.

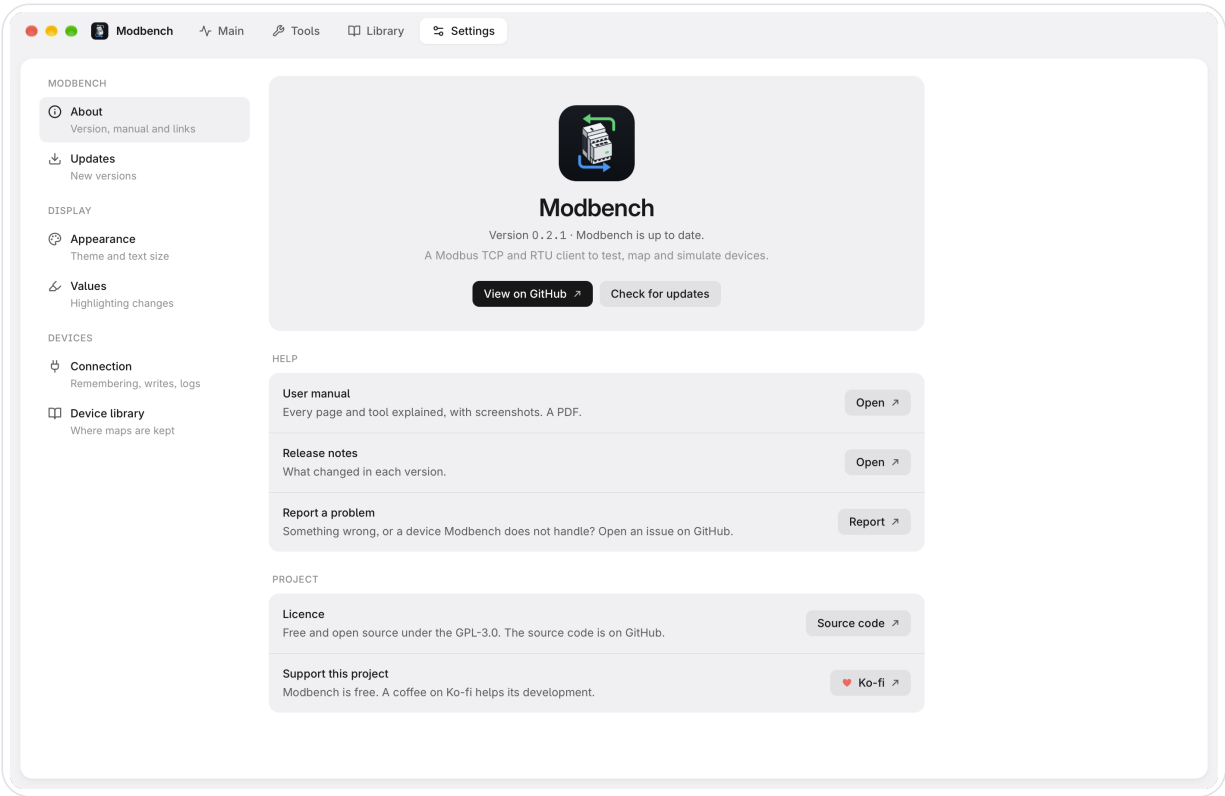


Figure 27. The About section.

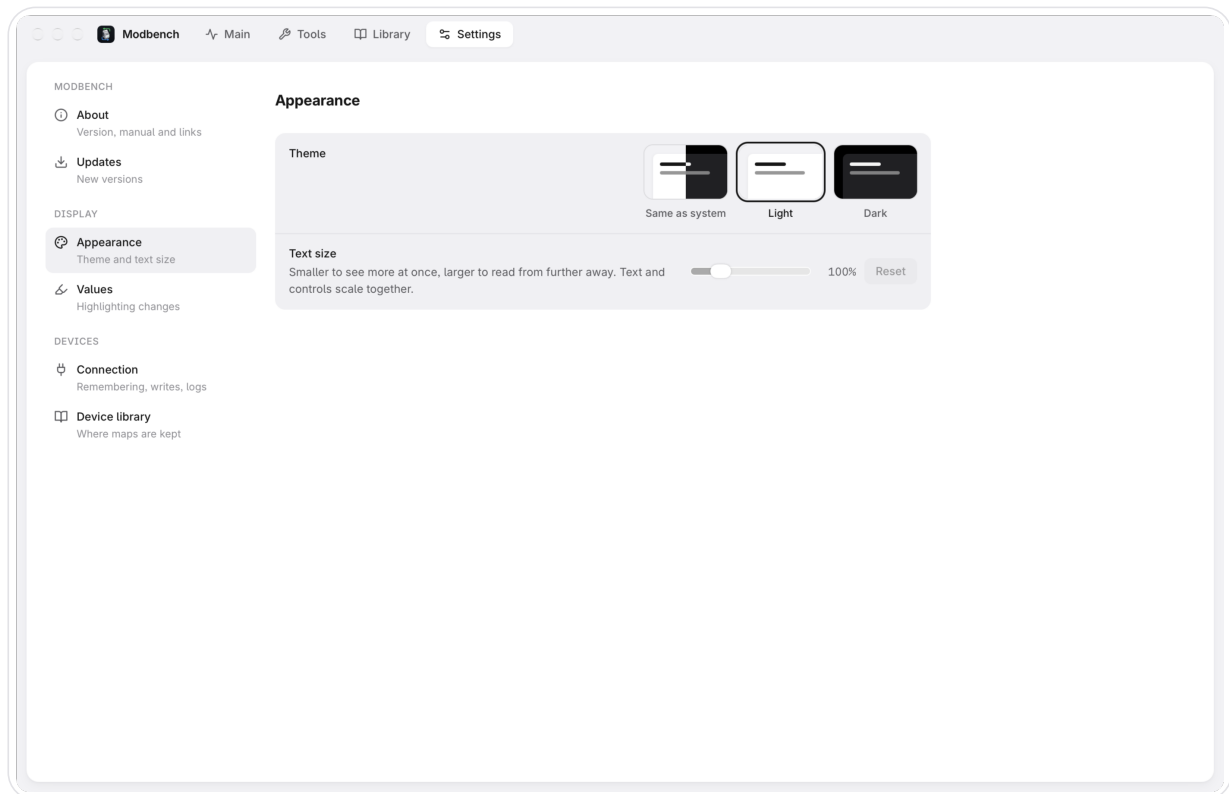


Figure 28. The Settings page, on Appearance.

11.2 Appearance

Theme

Light, dark, or as the system.

Text size

Smaller to see more at once, larger to read from further away. **Reset** goes back to 100 %.

11.3 Values

Highlight changed values

How long a value that changed stays highlighted: off, or 1 to 30 seconds.

11.4 Device library

Library folder

Where your maps are kept. **Open folder** shows it.

Shared maps

Links to the map format and to the repository of maps shared by other users.

11.5 Updates

Check for updates when Modbench starts

On by default. Installs from the APT repository update with `sudo apt upgrade` instead.

11.6 Connection

Remember the last connection

Port, baud rate, address, device and timing are filled in again when Modbench starts.

Remember register choices for each device

Which registers are read, the byte order and Loop come back for each device (map, connection and unit ID). Restoring them never writes or starts a test.

Forget saved device choices

Clears what was remembered per device.

Confirm before writing to a device

Asks before every write.

Reconnect when the connection is lost

While a test runs, a connection that fails is opened again, and the test goes on.

Read-only mode

Blocks every write, also on a connection that is already open.

CSV separator

For saved logs, exported values and recordings. Use a semicolon for spreadsheets set to a language with a decimal comma.

Requests kept in the log

Older requests are dropped first.

11.7 What Modbench keeps

Everything stays on this computer: settings, the last connection, choices per device, alarms, templates and the library folder. The only network requests besides your devices are downloads: the update check, the online library, and the vendors' icons from their websites.

Command line and AI assistants

`modbench-cli` reads devices and works with maps without the window: for scripts, for servers without a screen, and for a quick check. It comes with the app. It never writes to a device.

12.1 Commands

```
modbench-cli read --tcp 192.168.1.10 holding 0x0405 1
modbench-cli read --udp 192.168.1.10 holding 0 10
modbench-cli map --rtu COM3 --unit 18 device.mbmap.json --filter voltage
modbench-cli scan --rtu COM3 --baud 9600 --from 1 --to 32
modbench-cli id --tcp 192.168.1.10
modbench-cli simulate device.mbmap.json --strict
modbench-cli simulate device.mbmap.json --generators moving.json
modbench-cli simulate device.mbmap.json --rtu-over-tcp
modbench-cli simulate device.mbmap.json --rtu COM5 --baud 19200 --parity E
modbench-cli validate device.mbmap.json
modbench-cli convert old-device.mbmap
```

Command	What it does
<code>read</code>	Reads coils or registers, like a manual request.
<code>map</code>	Reads every register of a map from a device, by name; <code>--filter</code> keeps the names that match.
<code>scan</code>	Finds the unit IDs that answer.
<code>id</code>	Device identification (FC43).
<code>simulate</code>	A simulated device shaped after a map (port 5020 by default); <code>--strict</code> answers only the map's addresses.
<code>validate</code>	Checks a map against the format, with the same messages as the app.
<code>convert</code>	Converts a version 1 map to a <code>.mbmap.json</code> next to it.

`modbench-cli --help` lists every command and option.

12.2 Generators file

`--generators` takes a JSON list. Each entry names a map register (in engineering units) or a raw area and address, a kind, and how often it changes (`every_ms`, 1000 by default, at least 50):

```
[
  {"register": "Voltage L1", "kind": "sine", "offset": 230, "amplitude": 5, "
    period_s": 10, "every_ms": 500},
  {"area": "input", "address": 0, "kind": "increment", "step": 1, "min": 0, "max":
    100},
  {"area": "holding", "address": 1, "kind": "random", "min": 10, "max": 20},
  {"area": "discrete", "address": 0, "kind": "copy", "from": {"area": "coil", "
    address": 0}}
]
```

12.3 AI assistants (MCP)

`modbench-cli mcp` serves the same reads over the Model Context Protocol (stdio), so an AI assistant such as Claude Code can read devices and check maps: `read_map` reads a device with a map, `validate_map` checks a map file. Like the command line, it never writes to a device.

Troubleshooting

A.1 No answer (timeouts)

On a serial line

Check, in this order: the unit ID; baud rate, parity and stop bits (Find... tries them, Section 3.5); the A and B wires, which are often swapped; the termination at both ends of a long line; and that no other master talks on the same line.

On the network

Ping the address; check the port (502, or the converter's); some devices accept one client at a time, so close other programs connected to it. Through a gateway, the unit ID selects the device behind it.

Slow devices

Raise Timeout, and add a Delay between requests. Statistics suggests a timeout from the measured answers (Section 10.6).

A.2 Values look wrong

Off by one register

The manual counts from 1 and the map, or your request, from 0, or the other way round. Try the address one lower or higher, or check address_base in the map.

Huge or tiny numbers

Usually the byte or word order. Try the Byte order override in step 2, or paste the registers into the Data decoder (Section 10.3).

Wrong by a factor of 10, 100...

The scale. Maps give it per register; some devices also have a scale factor register.

A mismatch banner

The device is not the one the map is for, or its firmware differs (Chapter 4).

A.3 Bad frames

A CRC or LRC error means the bytes arrived damaged: noise on the line, a missing termination, a wrong baud rate, or two devices answering at once because they share a unit ID. The Raw data tab shows the frame (Section 7.1).

A.4 Exception codes

Code	Name	Usually means
01	Illegal function	The device does not support this function code.
02	Illegal data address	The address, or part of the range, does not exist. Read fewer registers, or check the address base.
03	Illegal data value	A value or a quantity the device refuses.
04	Server device failure	The device failed to do it.
05	Acknowledge	Accepted; the result comes later.
06	Server device busy	Try again later.
08	Memory parity error	The device's memory failed a check.
0A	Gateway path unavailable	The gateway has no route to that unit ID.
0B	Gateway target failed to respond	The device behind the gateway did not answer.

Reference

B.1 Function codes

Code	Function	Most per request
01	Read coils	2000 bits
02	Read discrete inputs	2000 bits
03	Read holding registers	125 registers
04	Read input registers	125 registers
05	Write single coil	1 bit
06	Write single register	1 register
15	Write multiple coils	1968 bits
16	Write multiple registers	123 registers
43/14	Read device identification	—

B.2 Byte and word order

How the 32-bit value 0x0A0B0C0D arrives, in the four orders:

Byte order	Word order	On the wire	Common name
big	big	0A 0B 0C 0D	Modbus standard (ABCD)
big	little	0C 0D 0A 0B	Word swap (CDAB), used by many meters and PLCs
little	big	0B 0A 0D 0C	Byte swap (BAD C), rare
little	little	0D 0C 0B 0A	Little-endian (DCBA)

B.3 Files

`.mbmap.json`

A device map, version 2: one JSON object with the device's connection defaults and its registers. The format is described in `SPEC.md` of the `mbmap`-library repository.

.mbmap

A device map, version 1 (older, line-based). Modbench opens it and converts it to version 2.

.mbsim

A simulator setup (JSON): the interface settings and each device's unit ID, map (with a copy of it), *Only the map's addresses* and generators.

Recordings (.csv)

Made with **Record...**: a time column, then one column per register, named with its unit; one row per cycle. The Trend player plays them back.

Logs and exports (.csv)

From **Save log...** and **Export...**, with the separator set in Settings.

B.4 Keyboard

Key	Does
Esc	Closes a menu or a dialog; in the online library, goes back to the vendors.
↑ ↓ Enter	Move through and choose in menus and lists.
Tab	Moves to the next field.