

# Modbus Client

Modbus Client Manual

February 7, 2024



Federico Turco  
federico.turco97@gmail.com





# Contents

<b>1</b>	<b>Homepage</b>	<b>1</b>
1.1	Modbus Secure . . . . .	2
<b>2</b>	<b>Main window</b>	<b>3</b>
2.1	FC01 - Coils . . . . .	3
2.2	FC02 - Discrete inputs . . . . .	5
2.3	FC03 - Holding registers . . . . .	6
2.4	FC04 - Input registers . . . . .	8
2.5	Diagnostic . . . . .	9
<b>3</b>	<b>Packet Log</b>	<b>10</b>
<b>4</b>	<b>Settings</b>	<b>11</b>
<b>5</b>	<b>Custom Templates</b>	<b>13</b>
5.1	Group definition . . . . .	15
5.2	Datatypes . . . . .	16
5.3	Datatype modifiers . . . . .	17
<b>6</b>	<b>Tools holding registers</b>	<b>19</b>
6.1	Bit command tool . . . . .	19
6.2	Byte command tool . . . . .	21
6.3	Word command tool . . . . .	22
<b>7</b>	<b>Database management</b>	<b>23</b>
7.1	Save configuration . . . . .	23
7.2	Configuration path . . . . .	23
7.3	Load configuration . . . . .	24
<b>8</b>	<b>Dropdown</b>	<b>25</b>
8.1	File Menu . . . . .	25
8.2	View Menu . . . . .	25
8.3	Database Menu . . . . .	26
8.4	Tools Menu . . . . .	26
8.5	Import Menu . . . . .	26
8.6	Export Menu . . . . .	27
8.7	Info Menu . . . . .	27
<b>9</b>	<b>Shortcuts</b>	<b>28</b>
<b>10</b>	<b>Modbus Secure</b>	<b>29</b>
10.1	Modbus Secure Specification . . . . .	30
10.2	X509v3 extensions . . . . .	30
10.3	Certificate generation . . . . .	31

<b>11 Other notes</b>
-----------------------

<b>32</b>
-----------

# 1 | Homepage

This client is fully compliant with Modbus specifications, it allows to read and write registers through both Modbus RTU and TCP protocols. The main window contains the basic information to configure the protocol communication. A USB-485 converter is required to query RTU slaves. Starting from version v2.38 the client implements the new standard Modbus Secure.

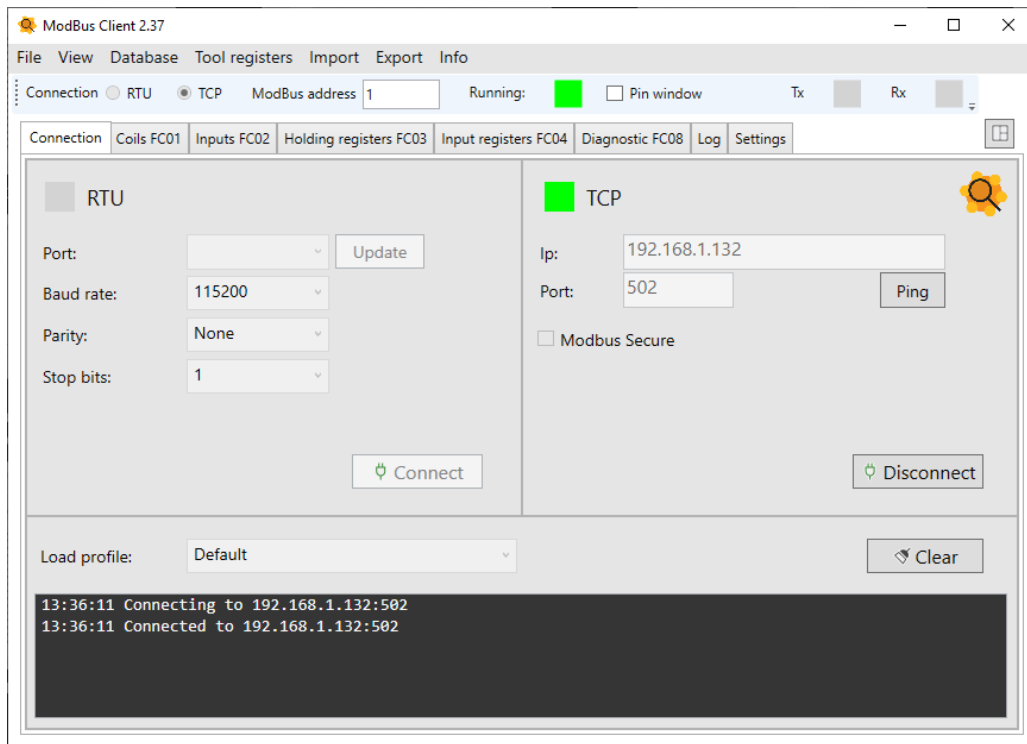
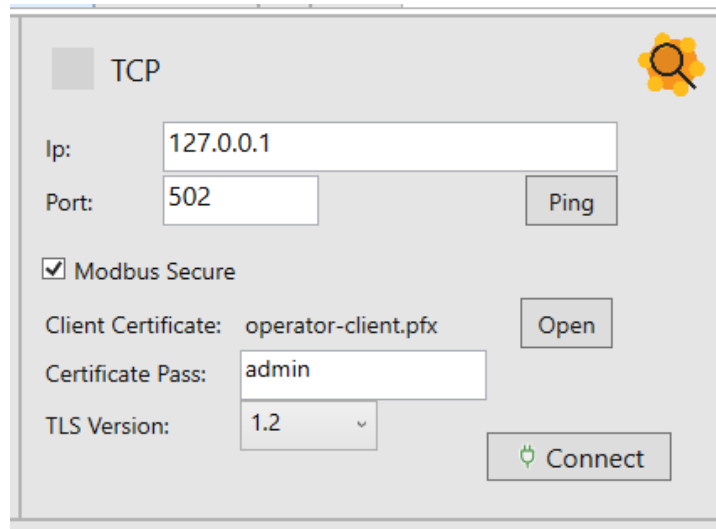


Figure 1.1: Homepage

To test the connection to an IP address use the ping button, if the ping is successful the button colors green otherwise red. When the connection to a slave is successful the "Running" cube is colored green (in the case of RTU connections it is considered successful if it succeeds in open the selected COM port correctly). The next part of the document will describe the various tabs and associated functions.

## 1.1 Modbus Secure

Starting with version 2.38, support has been introduced for the ModBus Secure standard, according to the specifications described in the document MB-TCP-Security-v21\_2018-07-24. Flagging the checkmark "Modbus Secure" enables the "Secure" configuration as shown in the following image:



The image shows a configuration window for a Modbus Client. At the top left, there is a tab labeled 'TCP' with a small square icon to its left. In the top right corner, there is a yellow gear icon with a magnifying glass. The main configuration area contains the following fields and buttons:

- Ip:** A text box containing '127.0.0.1'.
- Port:** A text box containing '502'.
- Ping:** A button located to the right of the Port field.
- Modbus Secure:** A checkbox that is checked, located below the Port field.
- Client Certificate:** A text box containing 'operator-client.pfx'.
- Open:** A button located to the right of the Client Certificate field.
- Certificate Pass:** A text box containing 'admin'.
- TLS Version:** A dropdown menu showing '1.2'.
- Connect:** A button with a green plug icon, located at the bottom right of the configuration area.

Figure 1.2: Modbus Secure

The Modbus Secure protocol will be discussed in more detail later in section 10. It still works via TCP but standard datagrams are encrypted through an SSL/TLS layer. The Modbus Secure protocol requires (in addition to IP and port) to load into the client a password-protected certificate (password to be entered in the appropriate field). The certificate should be uploaded in .pfx format, this format contains, along with the certificate itself, also the private key to be used to encrypt the communication. The client supports TLS v1.2 or 1.3 (the standard requires v1.2 or higher).

## 2 | Main window

Next to main tab there are 4 tabs to read the various Modbus resources "FC 01 Coils", "FC02 Discrete Inputs", "FC 03 Holding Registers", and "FC04 Input Registers".

### 2.1 FC01 - Coils

The coils tab allows reading and writing digital outputs with the functions FC01/FC05/FC15. Read-/Loop/Write buttons are unlocked only if the connection to a device has been successfully opened.

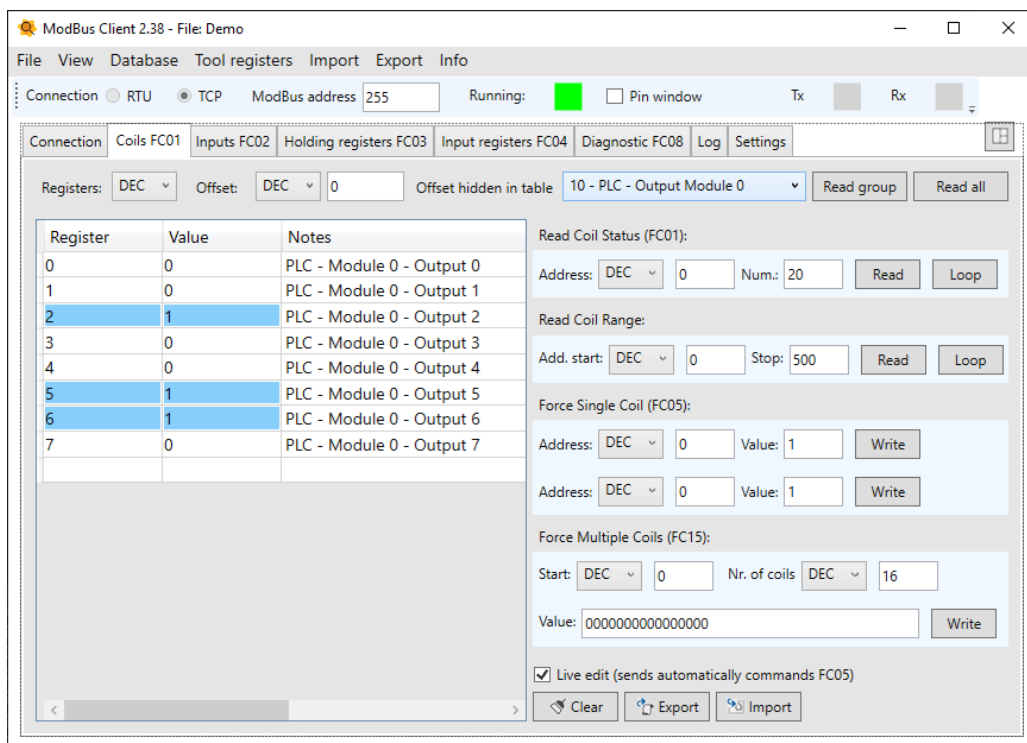


Figure 2.1: FC01 - Coils

The loop function allows the indicated registers to be read in polling, the polling interval is configured from the tab "Settings". By default, read cells are colored blue if the contents are populated (understood as  $> 0$ ). At the top right there are commands to read resources from a group or all resources of type coils configured in the current profile. The configuration of resources as well as the creation and association of groups is described in the chapter 5.

The "Read Coil Range" box is a user-defined range of digital outputs that can be read, the program will then eventually divide the command into multiple FC01 requests each of n coils indicated above (in the example below equal to 20). It is also possible to force multiple coils (FC15) using the form below (Force Multiple Coils) or by importing a previously exported csv file.

Coils set to 1 are colored green if the write is successful. successful:

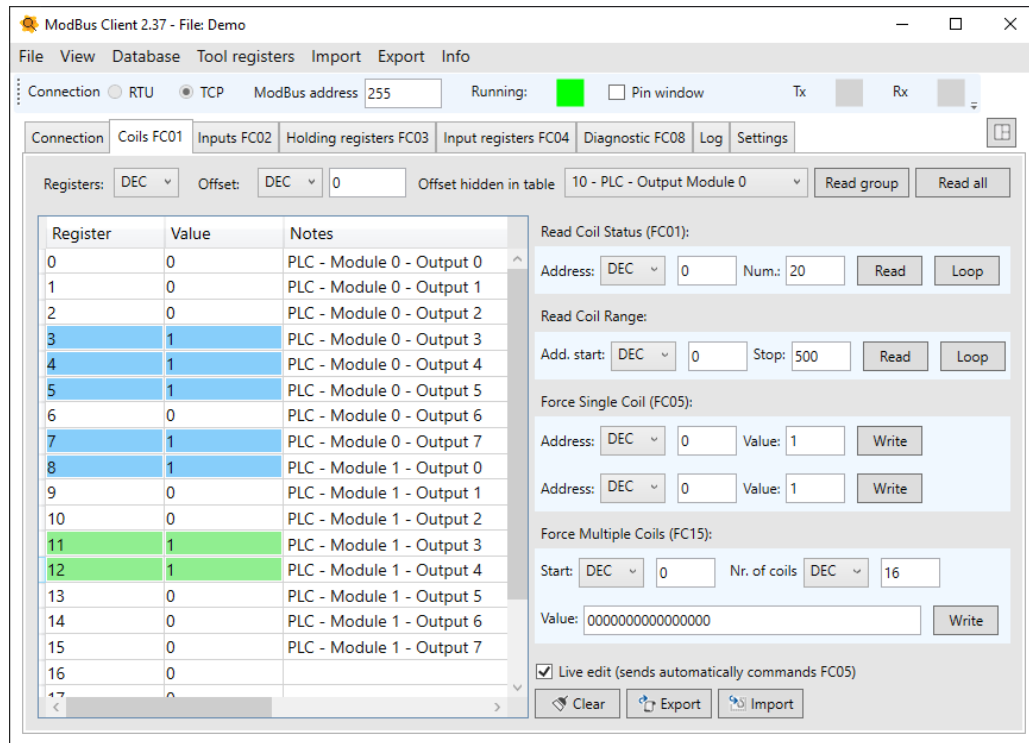


Figure 2.2: FC05 - Write Coils

By enabling the "Live Edit" mode, it is possible to directly edit the coils by editing the "Value" column, in this case when editing the row the program automatically sends the FC05 command to set the coil to the entered value.



## 2.2 FC02 - Discrete inputs

The Inputs tab allows reading digital inputs with FC02 functions. The Read/Loop buttons as for the Coils tab are unlocked only if a connection to a device has been successfully opened. The "Read group" and "Read all" buttons allows to read registers previously configured in the custom template (registers are read individually whether or not they are consecutive with each other).

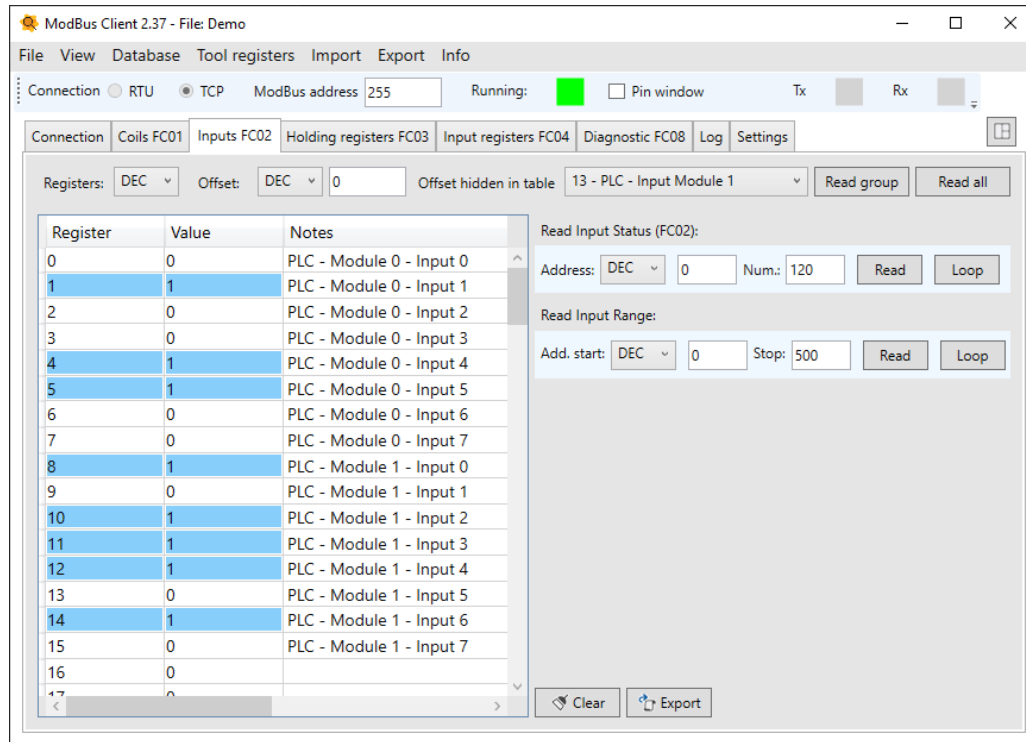


Figure 2.3: FC02 - Discrete Inputs

The "Read Input Range" box is a user-defined range of digital inputs can be read, the program will then eventually split the command into multiple FC02 requests each of n inputs specified in the single reads box (in the image above equal to 120).

## 2.3 FC03 - Holding registers

The Holding Registers tab allows you to read and write 16-bit digital registers with the functions FC03/FC06/FC16.

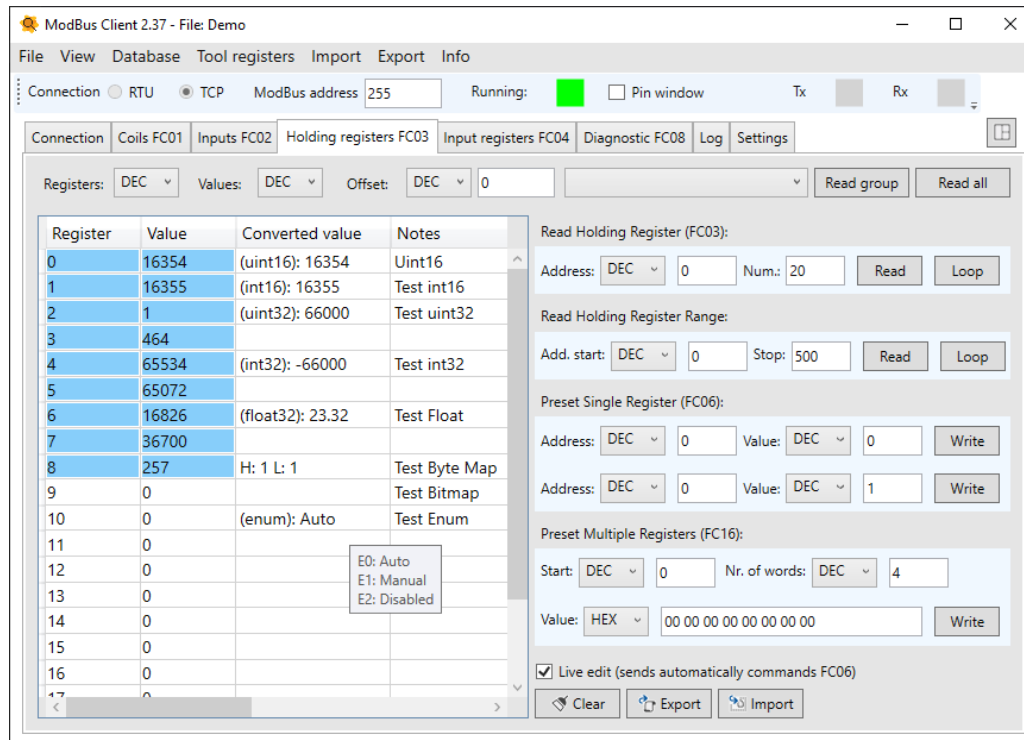


Figure 2.4: FC03 - Holding Registers

The content of the "Value" field can be displayed in either decimal (DEC) or hexadecimal (HEX). In the table it is also possible to enable the display of the corresponding binary value. By expanding the window it is possible to display additional information (which is configured in the template window), to a holding register in fact it is possible to associate a label to identify its contents or display the value converted to integer/ float/string/etc.. Use the "View" drop-down menu to configure the columns you want to display in the table as shown in the following image.

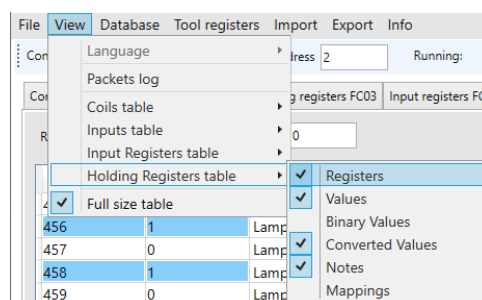


Figure 2.5: FC03 - View Tab Holding Registers

With the "Read Holding Register Range" box, it is possible to read a register range defined by the user (also > 123), the program will then eventually split the command into multiple FC03 requests of n registers specified in the box above (in image 120) and populate the table with all the requested registers.

By checking the "Live Edit" flag, it is possible to send the register write automatically by editing a cell in the "Value", "Binary Value" or "Converted Value" columns. The "Live edit" is very useful for editing displayed registers.

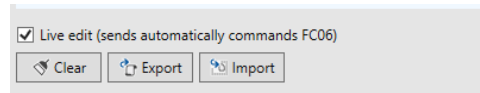


Figure 2.6: FC03 - Live edit

Note that changes in the "Value" column are sent with the function FC06 (write single register), changes to the "Binary Value" column are converted from binary and always sent as FC06 while changes to the "Converted Value" column are sent as FC06 or FC16 depending on the data type (the function FC16 is used for 32- or 64-bit integer or float variables). In the value column you can also enter values in hexadecimal; if the display is already in hexadecimal, the value entered will be considered always hexadecimal, while in the decimal display, prepend a "0x" or "x" or postpone an "h" to indicate that the value you want to send is written in hexadecimal.

The displayed table can be exported in .csv or .json format with the "Export" button at the bottom right. Tables exported to csv can be imported and sent to the slave with the "Import" button. This function is convenient when you want to copy a memory map from one PLC to another or simply export a backup that can be restored at any time.

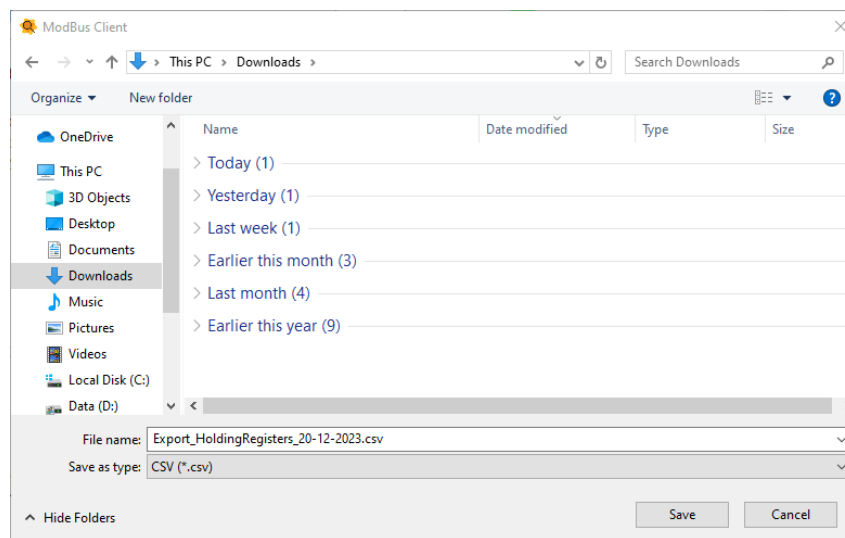


Figure 2.7: FC03 - CSV import

## 2.4 FC04 - Input registers

The Input Registers tab allows you to read 16-bit registers with the FC04 functions. The buttons Read/Loop as for the other tabs are unlocked only if the connection to a device is successful.

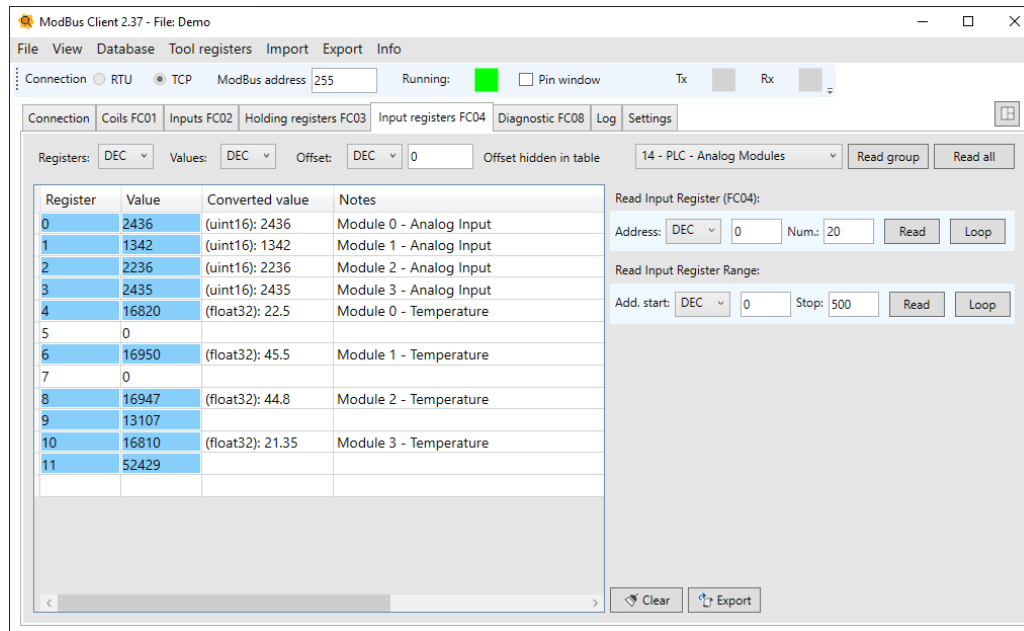


Figure 2.8: FC04 - Input Registers

The content of the "Value" field can be displayed in either decimal (DEC) or hexadecimal (HEX) representation. The value in binary is also shown next to it. By expanding the window, it is possible to display additional information (which is configured in the template window). To a register in fact, it is possible to associate a label to identify its contents or to display the value converted to integer/float/string/etc.. See the Template section for further clarification of this part. With the "Read Input Register Range" box, it is possible to read a user-defined register range (even > 123), the program will then eventually split the command into multiple FC04 requests to populate the table with all the requested registers.

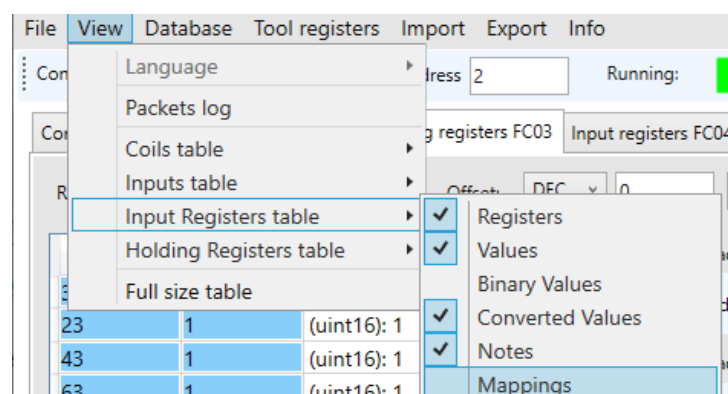


Figure 2.9: FC04 - Input Registers

As with holding registers, it is possible to select from the view menu which columns to show to the user.

## 2.5 Diagnostic

In the diagnostics tab, diagnostic commands can be sent to the queried device, be aware that not all devices respond to the FC08 function.

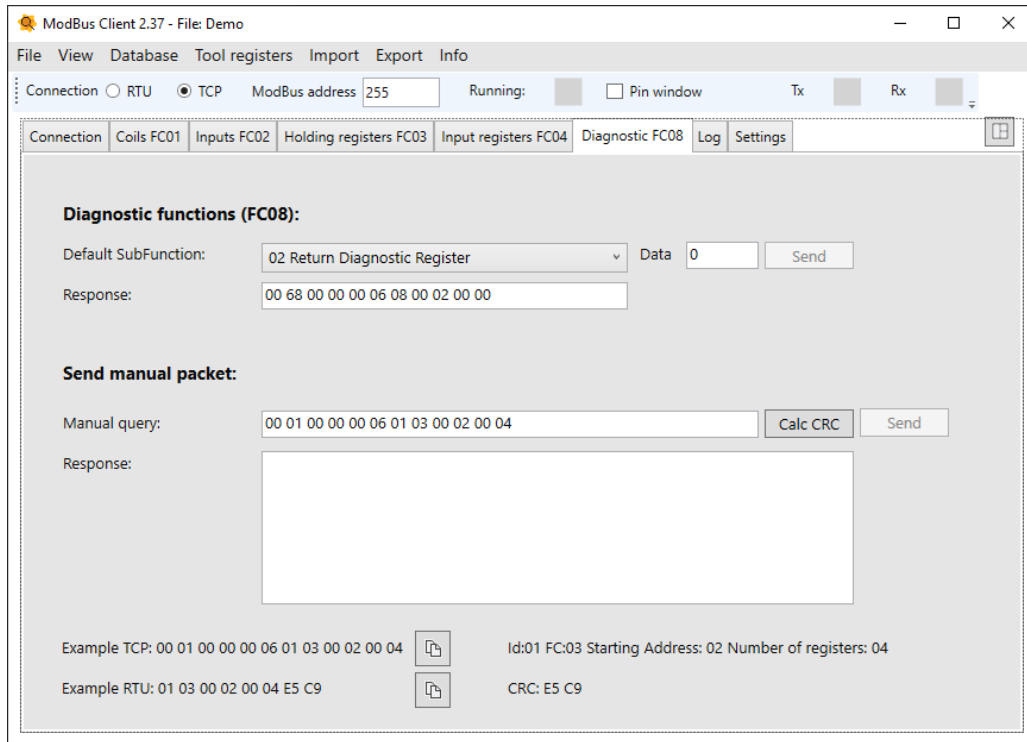


Figure 2.10: FC08 - Diagnostic

Supported FC08 functions:

- 00 Return Query Data
- 01 Restart Communications Option
- 02 Return Diagnostic Register
- 03 Change ASCII Input Delimeter
- 04 Force Listen Only Mode
- 10 Clear Counters and Diagnostic Register
- 11 Return Bus Message Count
- 12 Return Bus Communication Error Count
- 13 Return Bus Exception Error Count
- 14 Return Slave Message Count
- 15 Return Slave No Response Count
- 16 Return Slave NAK Count
- 17 Return Slave Busy Count
- 20 Clear Overrun Counter and Flag

Use this tab also if you need to build Modbus telegrams manually. In case of RTU connection use the CRC button to calculate and append the CRC16 according to the Modbus protocol specification. On serial communication infact the slave checks the CRC to see if the packet contains transmission errors, so if you don't append the CRC the slave ignores the command and doesn't respond to any request.

## 3 | Packet Log

The Log window shows the raw bytes sent and received.

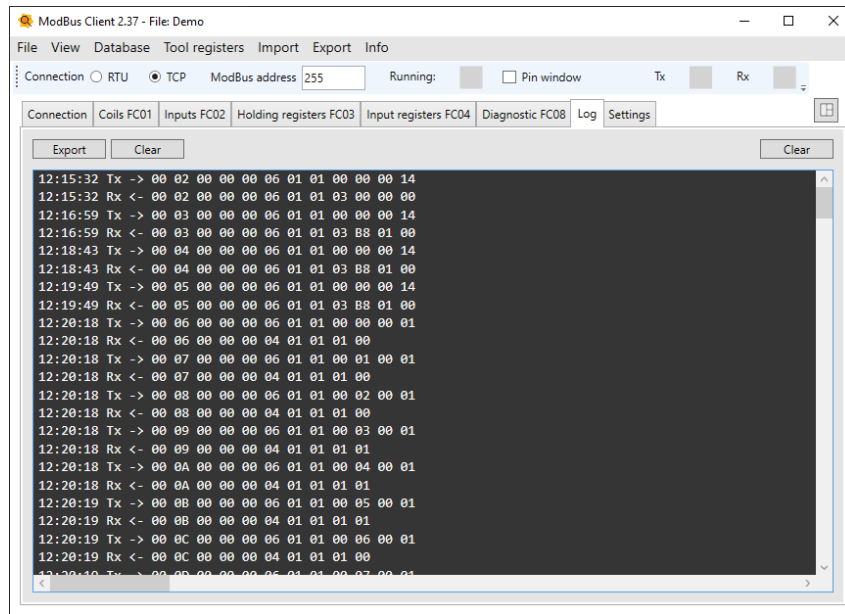


Figure 3.1: Log window

The log window can also be opened in a separate window from the menu "View" -> "Packet Log": (or with the shortcut keys Ctrl + L).

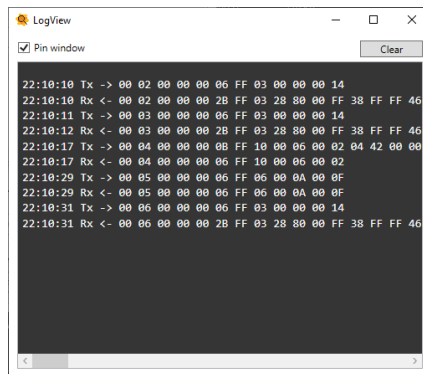


Figure 3.2: Separate log window

By flagging the "Pin window" checkbox, the window always remains on top.

## 4 | Settings

The settings tab contains operating parameters, beware that the settings are tied to the profile, changing the profile loads the respective settings.

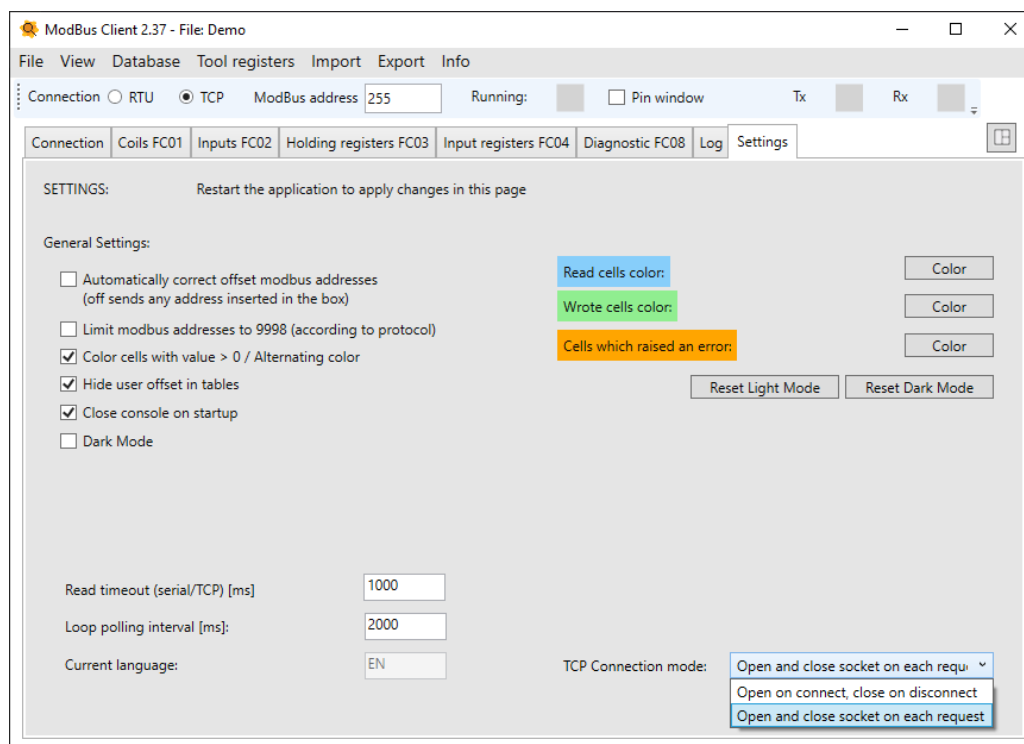


Figure 4.1: Settings tab

- **Automatically correct offset modbus addresses:** If checked when entering an address in the original protocol format e.g. holding register 40002 is automatically sent in the modbus request as address 00001 thus eliminating the offset provided for holding registers. Otherwise it is requested address 40002. Same for input registers (30001), discrete inputs (10001) or coils (00001).
- **Limit modbus addresses to 9998:** If checked, only address requests between 0 and 9998 (or corresponding address modbus e.g.: 10001-19999/30001-39999/40001-49999) are allowed. Otherwise, the request is sent whatever modbus address is entered. For example, some Beckhoff CPUs provide the words MW0,MW1,... starting at holding register 0x3000 (dec 12288). By not checking the box you can then also send requests to addresses beyond the value of 9998, up to 65535.
- **Color cells with value > 0 / Alternating color:** If checked, only table rows with a value > 0 are colored, otherwise rows are colored alternately.
- **Hide user offset in tables:** If checked, the general offset is not displayed in the tables but is

counted in the commands sent via ModBus. (User offset means the offset entered in the box "Offset" present in each tab).

- **Close console on startup:** Automatically closes the console when you start the application.
- **Dark mode:** Enable dark mode (black backgrounds, white text).

In addition to the ticks there are some parameters such as:

- **Read timeout:** Timeout in ms for response to a command.
- **Loop polling interval:** Interval in ms between two reads for loop read commands.

For non-secure TCP connections only, it is possible to choose between two modes of socket management:

- **TCP connection mode:** This parameter allows you to change the TCP socket handling only for non-secure mode. At the TCP socket level normally the connection is opened, kept open and closed at the end of its use. On unstable connections, however, it is more convenient to open and close the socket on each request. The correct mode of the protocol would be the first one, but it often happens on unstable connections (e.g., slaves connected via modem) that a connection is initiated, after a few reads it falls out, the socket fails and you have to reconnect again. With the socket going into timeout as it is no longer open on the slave when the connection returns. With the second mode this does not happen because the socket is opened for each read and closed immediately after so on a practical level this mode can come in very handy.



## 5 | Custom Templates

In the "Database" -> "Open template editor" menu you can configure custom labels, bit mappings and any integer/float/string conversions to be associated with the various registers.

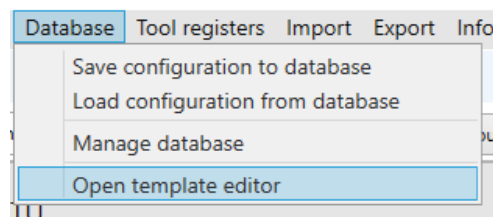


Figure 5.1: Database menu

The window for entering custom templates associated with the various registers is as follows:

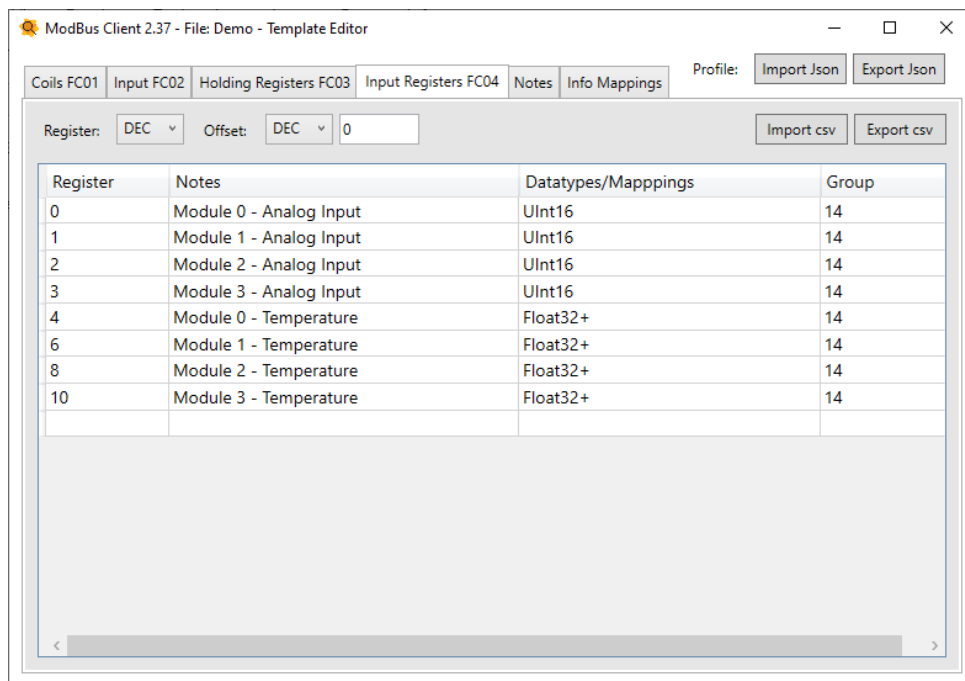


Figure 5.2: Template editor

For each register in the template window you associate a possible label, the corresponding datatype, and whether you want one or more resource groups to which the current group should be added. For each register you can specify whether the value is entered in decimal or hexadecimal and also you can add an offset to the registers. In the same window you can specify whether the registers entered are to

be considered in decimal or hexadecimal and a possible offset (positive or negative) to be added to the values entered. For example, an offset of 100 moves the label of register 10 to the position 110. This is convenient if, for example, a PLC has the %MW0 starting at offset 0x4000. In this case you fill in the table starting at 0 and then simply set as the offset the value HEX 4000.

In the case where different models of PLCs have different position of the %MW, to switch from one model to another is simply change the offset without having to go and change the registers one by one (negative offsets can also be applied should it be necessary). Following are screenshots of some examples based on the template of the previous image:

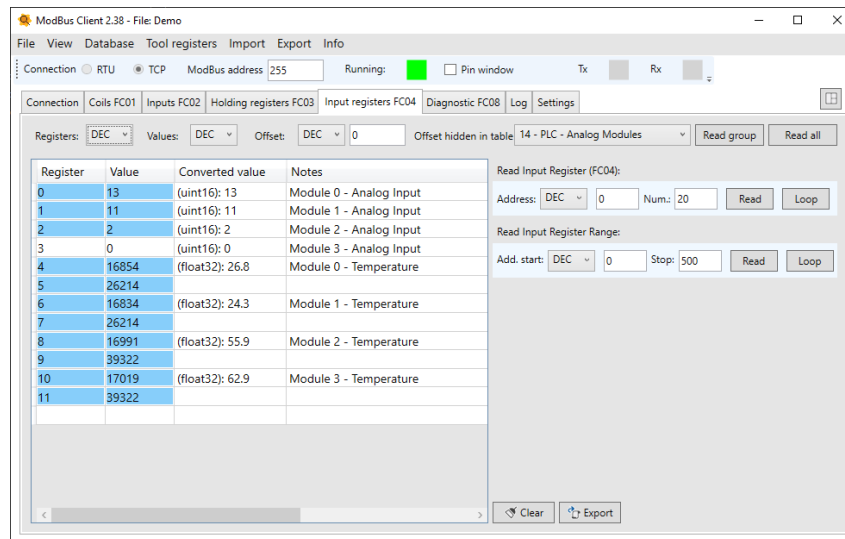


Figure 5.3: Template editor

The "Notes" column displays the descriptions of the various registers while the "Converted Value" column displays the contents of each register converted to the assigned datatype. In this way it is very simple to convert 32- or 64-bit values, floats or strings to the actual value. By selecting a group from the drop-down menu also the client goes to read the registers of the selected group in ascending order.

In the template window you can define multiple resource groups and associate them with the various registers, so in the main window you can read different registers (even non-consecutive ones) by calling up the corresponding resource group.

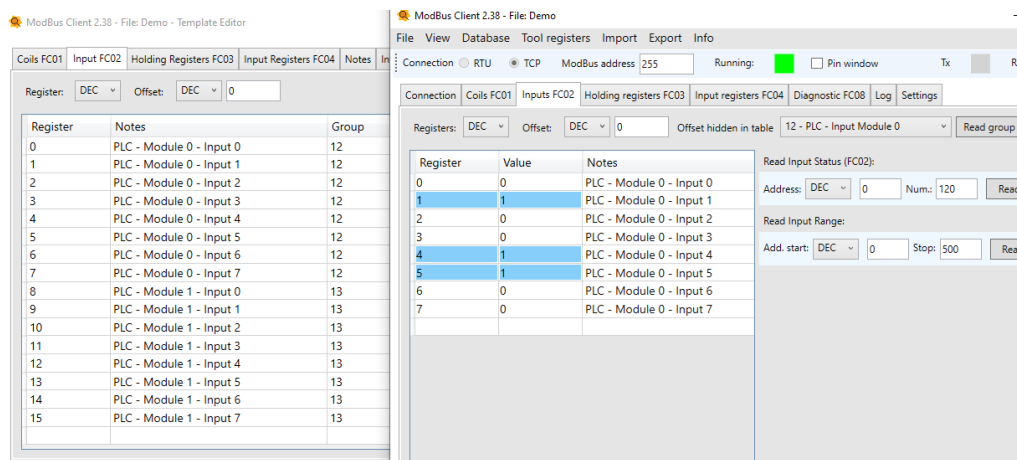


Figure 5.4: Group resources

## 5.1 Group definition

In the "Notes" tab of the template window, it is possible to define resource groups to be called up later in the main window. It is not necessary to enter the groups in order.

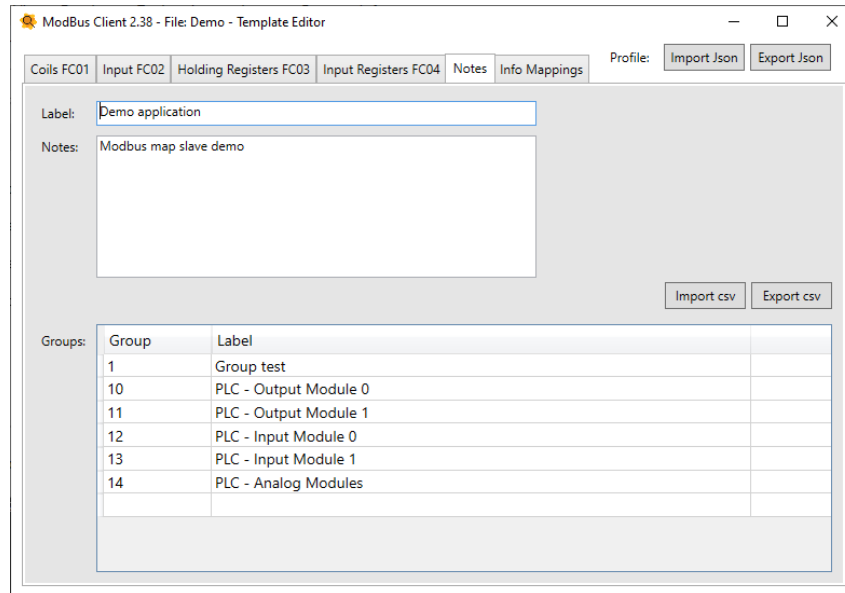


Figure 5.5: Group definition

The last tab "Info Mappings" contains a summary of the datatypes implemented and managed by the client.

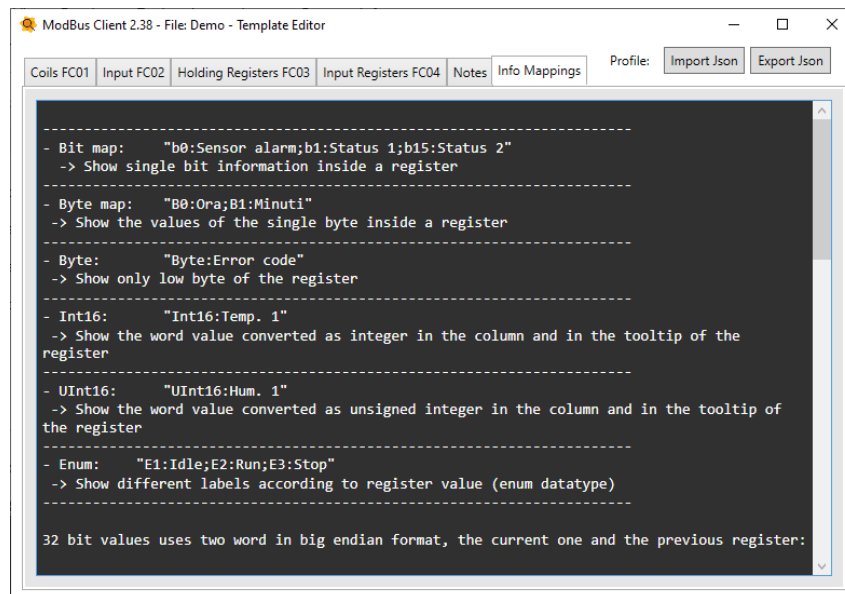


Figure 5.6: Mappings info

## 5.2 Datatypes

The following are the various datatypes supported by the program and configurable in a profile template.

### Bit map

It shows the individual bits of the word in the line tooltip with the resource label next to it.

```
b0:230v presence;b1:Status 1;b15:Status 2
```

### Byte map

It shows the two bytes of the word.

```
B0:Hour;B1:Minutes
```

### Int16 / UInt16

It shows the word as a signed integer.

```
Int16:Temp. 1  
UInt16:Counter
```

The following 32-bit (two word) variables use the previous (High Word) and current (Low Word) referenced in the Big Endian format.

### Float

It collects two words and displays the data in float in the row tooltip.

```
Float:Temperatura locale 1
```

### Int32 / UInt32

It collects two words and displays the data in int32 or uint32 in the line tooltip.

```
Int32:Temperatura locale 1  
UInt32:Temperatura locale 2
```

The following 64-bit variables (two words) use the three words of the previous (High Word) and current (Low Word) referenced in the Big Endian format. With the modifiers format you can specify whether to use the current register and the next three.

### Int64 / UInt64

```
Int64:Timestamp start time  
UInt64:Timestamp start time
```

### String(len[, offset])

With string objects, it is possible to convert the contents of registers to string (NULL terminated string). In the example below 8 bytes are converted into 8 ASCII characters with an offset of -2 (the string starts at the previous register).

```
String(8,-2):Model
```

## 5.3 Datatype modifiers

### Swap

Adding a dash "-" or "\_swap" string multiple words values are converted in little endian format.

Swap: "UInt32-" "UInt32\_swap"

### Word offset

Adding a plus sign "+" multiple words values uses the current register and the next ones to cast the content.

Offset: "UInt32+"

### Word offset + Swap

"UInt32-+" oppure "UInt32\_swap+"

Combine the previous two, use the current and next register in the Little Endian format.

The following screenshots contains some examples of different templates.

Template:

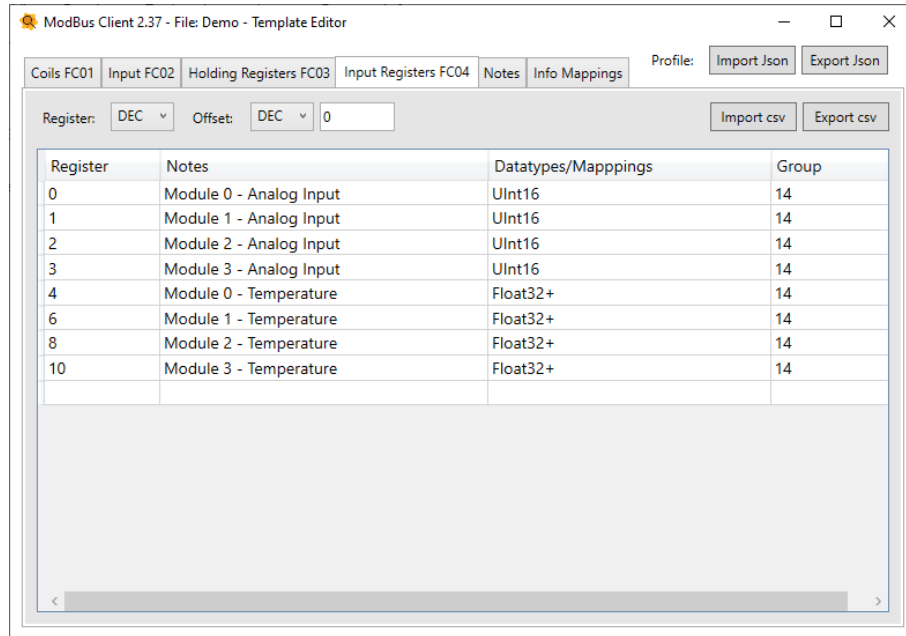


Figure 5.7:

Example of final conversion:

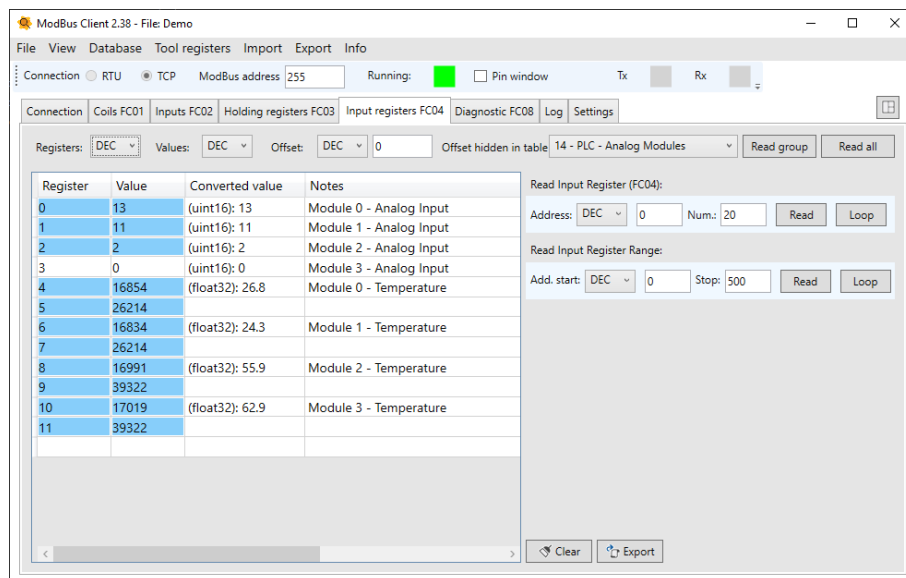


Figure 5.8:

As shown in the previous image 32 bit values are casted using two consecutive registers.

## 6 | Tools holding registers

To edit individual bits or individual bytes/words of holding registers there are a number of tools accessible from the drop-down menu "Tool registers". The following tools apply only to holding registers, they are mainly used to modify PLC objects of type %MX,%MB,%MW:

### 6.1 Bit command tool

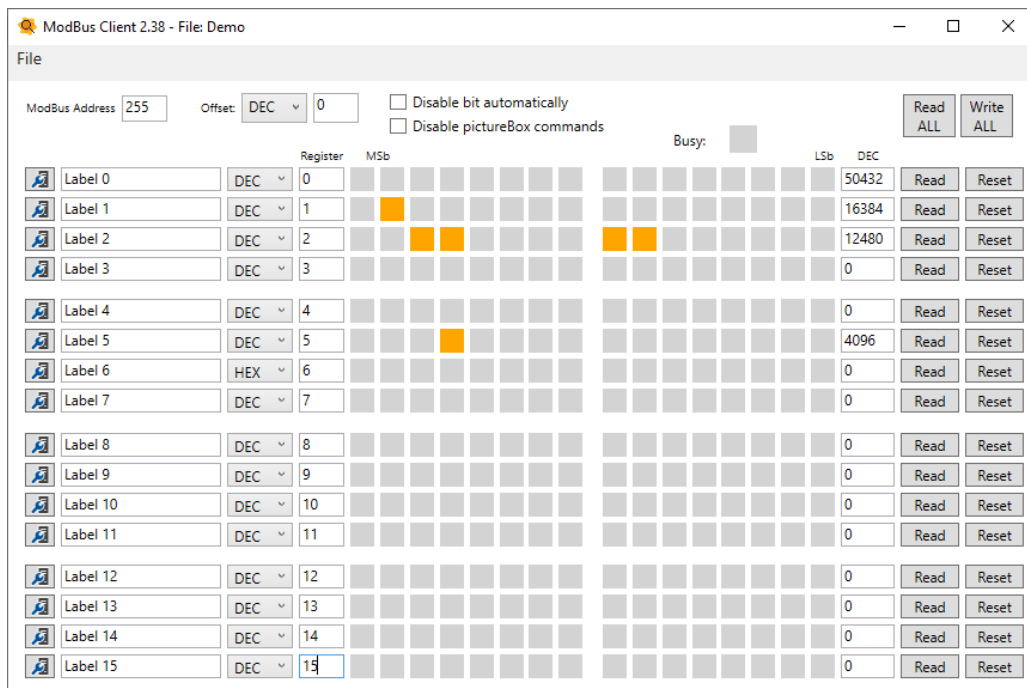


Figure 6.1: Bit command tool

In the Tool command bit window, you can read and view the registers in the individual bits that make them up. By pressing on the individual bit you can reverse their state (by default they work as a "step by step" otherwise if the option "disable bits automatically" is flagged on mouse click the bit is forced to 1 and then to 0).

Pressing the buttons next to the labels opens the window displayed in the next screenshot with which you can give a specific label to the individual bit within a word. The labels in question refer only to the window on the previous page; they are not displayed in the tab "Holding registers FC 03" of the main window.

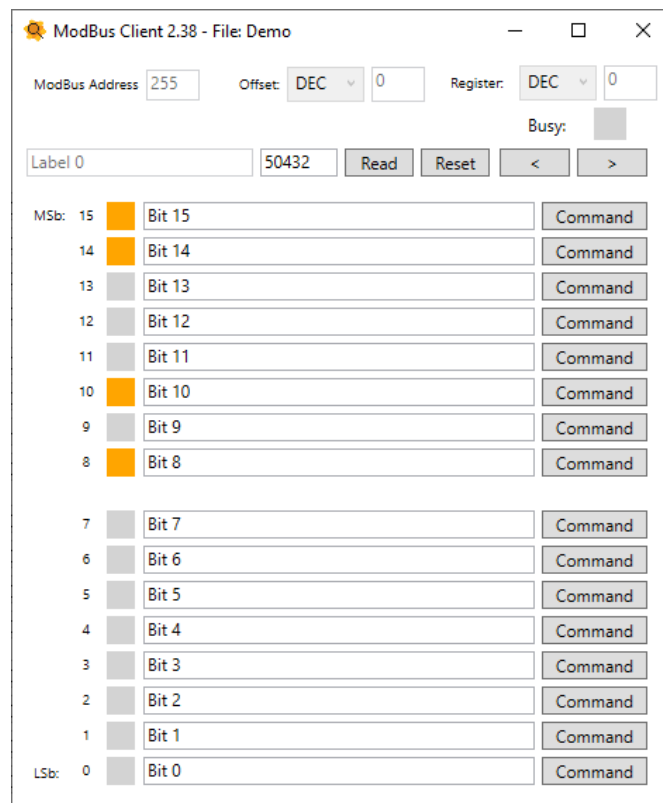


Figure 6.2: Bit command tool - Label Bits

The labels assigned to individual bits then become tooltips of the bits in the main window shown in the image 6.1. The buttons "<" e ">" in the upper right-hand corner allow to scroll between the words in the main window.



## 6.2 Byte command tool

The screenshot shows the 'Modbus Client 2.38 - File: Demo' window. The interface is divided into several sections:

- File menu:** Located at the top left.
- Modbus Address:** A text box containing '255'.
- Offset:** A dropdown menu set to 'DEC' and a text box containing '0'.
- Busy:** A checkbox that is currently unchecked.
- Navigation buttons:** '<' and '>' buttons in the top right corner.
- Read/Write buttons:** 'Read ALL' and 'Write ALL' buttons below the navigation buttons.
- Register/Value table:** A table with columns for 'Register' and 'Value'. The 'Register' column has a dropdown menu set to 'DEC'. The 'Value' column has a dropdown menu set to 'DEC'. The table contains 16 rows, each with an 'MSB' and 'LSB' label. The 'Register' values are 0 through 15, and the 'Value' values are all 0. The 'Register' dropdown for row 14 is highlighted in blue.
- Read/Write buttons:** 'Read' and 'Write' buttons are located to the right of each row in the table.

Figure 6.3: Byte command tool

From the window shown above, you can send commands to individual bytes that will then be written as word on the target. It is possible to create custom windows with which to send commands to registers referring to different locations as well. The "<" and ">" buttons in the upper right corner allow you to scroll between 4 different window profiles. Written cells are colored green while read cells are colored blue.

## 6.3 Word command tool

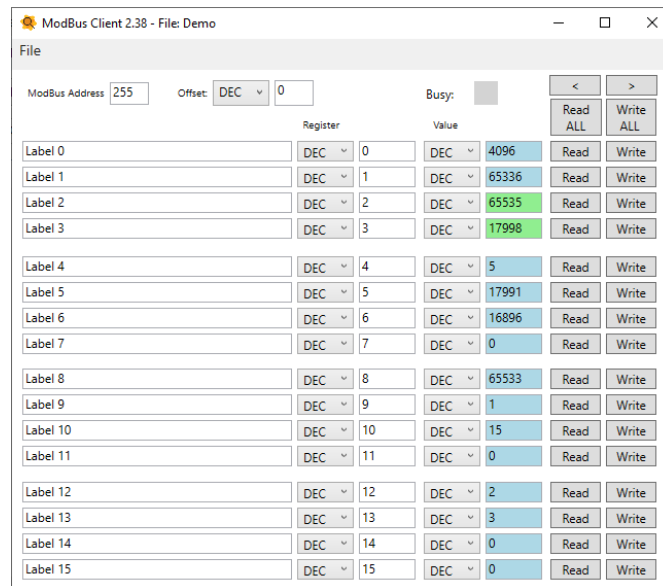


Figure 6.4: Word command tool

The "Word" window contains the same functions seen above for individual bytes, only divided by word. Like the previous one with the "<" and ">" buttons in the upper right corner you can scroll between 4 different window profiles. Written cells are colored green while read cells are colored blue.

## 7 | Database management

### 7.1 Save configuration

Pressing "Save Configuration to Database" opens a dialog box where you can enter the name of the new custom configuration:

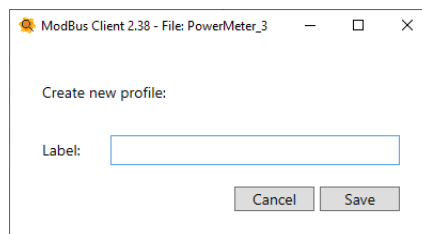


Figure 7.1: Save new profile

Pressing "Save" inside the "Json" folder creates a folder with the name inserted. DO NOT overwrite existing folders, to make changes to a custom configuration simply open it. The changes will be saved automatically when closing of the main window (eventually the user can save the current state from the "File"->"Save menu" or with the shortcut "Ctrl + S").

### 7.2 Configuration path

The "Json" folder contains a directory for each profile. The "Default" folder contains the program configuration when no custom configurations is selected (if you use the program without loading a specific profile, any change is saved in this folder). The other folders are generated when you save the current configuration as a new profile.

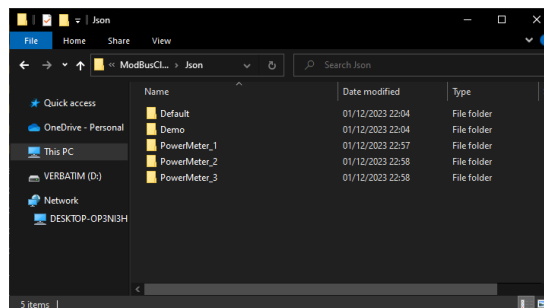


Figure 7.2: Database profile directory

The user in normal use does not need to make changes directly in this folder, simply use the forms to save and load profiles directly from the main window.

## 7.3 Load configuration

Pressing "Load configuration from Database" allows you to load a configuration previously saved:

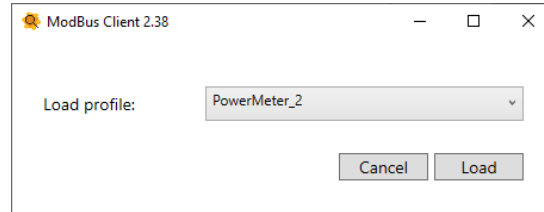


Figure 7.3: Load profile

Once a custom configuration is loaded any changes entered into the program will be saved in the custom folder without changing the structure of the configuration "Default".

To import or export a custom Profile use the "Manage database" tool, here you can export a profile as a .zip file, import a profile exported from another client or simply select the profile to be loaded. When you close the window, automatically the selected profile is loaded into the client.

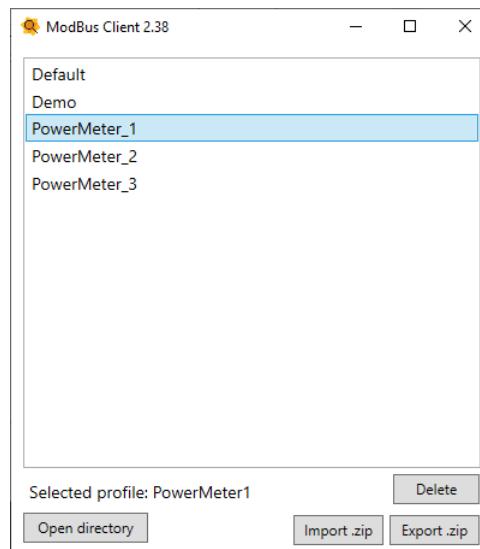


Figure 7.4: Database manager

Starting with version v2.37, profiles can be uploaded directly in the home tab via the drop-down menu provided:

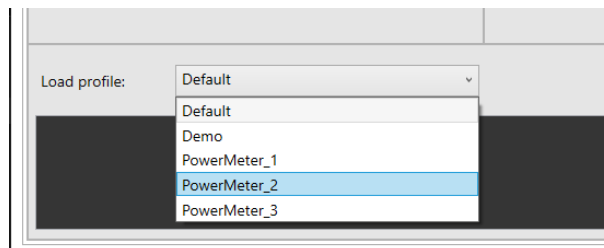


Figure 7.5: Homepage profile selection

## 8 | Dropdown

### 8.1 File Menu

The file menu contains commands to save the configuration and open/close the console.

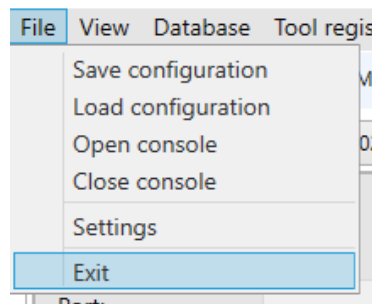


Figure 8.1: File Menu

### 8.2 View Menu

In the view menu you can change the program language as well as display and hide the register columns of the various tabs as seen in the following image:

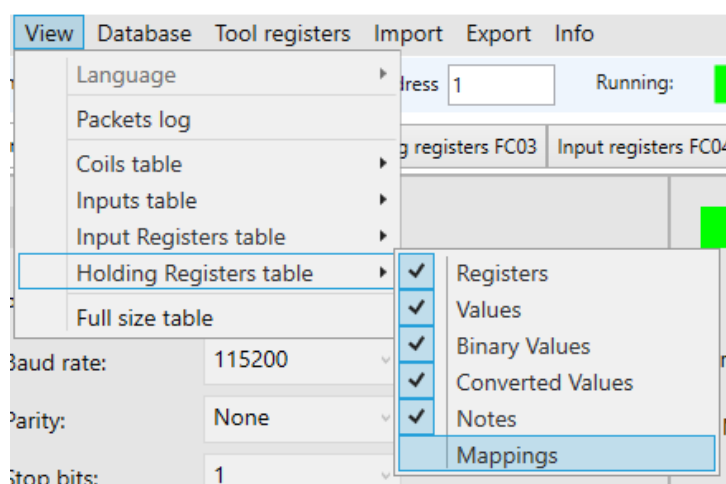


Figure 8.2: View Menu

### 8.3 Database Menu

In the database menu you can create/upload a profile and access the import/export tool of the profiles. The last menu item "Open template editor" opens the window for editing templates custom templates.

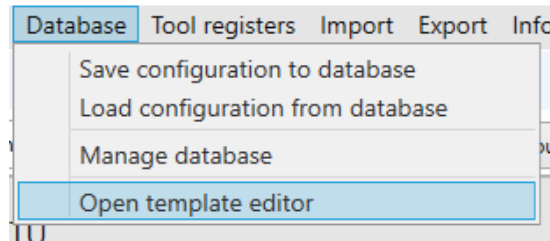


Figure 8.3: Database Menu

### 8.4 Tools Menu

In the tools menu, command windows can be opened to drive individual bits/bytes/words.

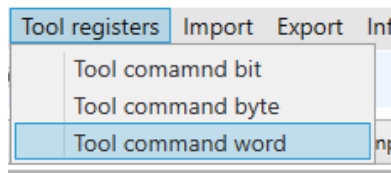


Figure 8.4: Tools Menu

### 8.5 Import Menu

In the import menu you can inport a table c oils or holding registers previously exported send it send it to the slave. In the case where the registers are consecutive you can choose whether to send them individually (write single coil/write single register) or in bulk (write multiple coils/write multiple registers).

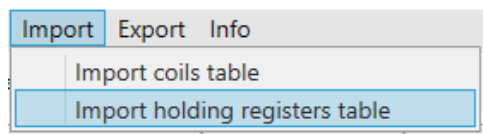


Figure 8.5: Import Menu

## 8.6 Export Menu

In the export menu you can export the tables of the various tabs in csv format.

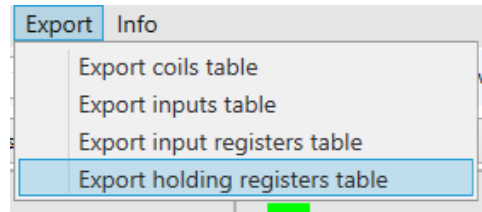


Figure 8.6: Export Menu

## 8.7 Info Menu

From the info menu you can open this guide, view the program license and date/build number.

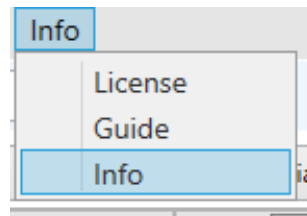


Figure 8.7: Info Menu

In the info window it is possible to read not only the current version but also the date and time of build.

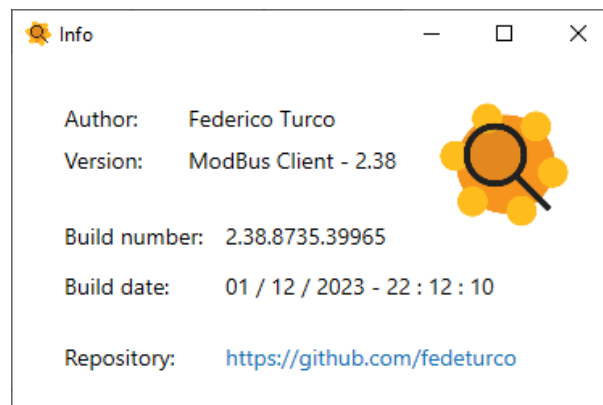


Figure 8.8: Info window

## 9 | Shortcuts

- **Ctrl + Num 1 - 9:** Select the tab with the index selected by the number pressed
- **Ctrl + Q/Ctrl + W:** Close the window
- **Ctrl + E:** Read the register range of the selected tab
- **Ctrl + R:** Read the registers of the selected tab (inputs/coils/input registers/holding registers)
- **Ctrl + T:** Open the template editing window for the currently selected profile
- **Ctrl + Y:** Open csv/json import window
- **Ctrl + U:** Open csv/json export window
- **Ctrl + I:** Open software version info window
- **Ctrl + O:** Open the menu for loading a profile
- **Ctrl + P:** Start/Stop polling for the selected tab (loop button)
- **Ctrl + S:** Save any changes to the currently selected profile
- **Ctrl + Shift + S:** Open the menu to save the current profile as a new profile
- **Ctrl + D:** Open the database management window
- **Ctrl + F:** Enable/disable full screen mode tables
- **Ctrl + G:** Read the group-related resources for the group selected in drop-down menu for the current tab
- **Ctrl + H:** Read all resources entered into template for current tab
- **Ctrl + K:** Start/Stop polling for the selected tab over the indicated range (loop button)
- **Ctrl + L:** Open log window
- **Ctrl + Shift + C:** Open/close client debug console
- **Ctrl + B:** Connect/disconnect
- **Ctrl + M:** Switch from TCP to RTU mode and vice versa
- **Del:** Clear table contents of current tab



## 10 | Modbus Secure

The Modbus Secure protocol uses the same frames as the TCP standard encapsulated through a TLS channel. The TLS protocol provides an authentication system using x.509v3 certificates. The Modbus Secure standard requires TLS with version 1.2 or higher (we are currently at version 1.3), the client supports both versions (1.2 and 1.3). The use of certificates requires the creation of a certificate and a private key for the server (slave), as well as for each client that will connect to the server (slave). A diagram is given below to clarify the roles of both server-side and client-side certificates:

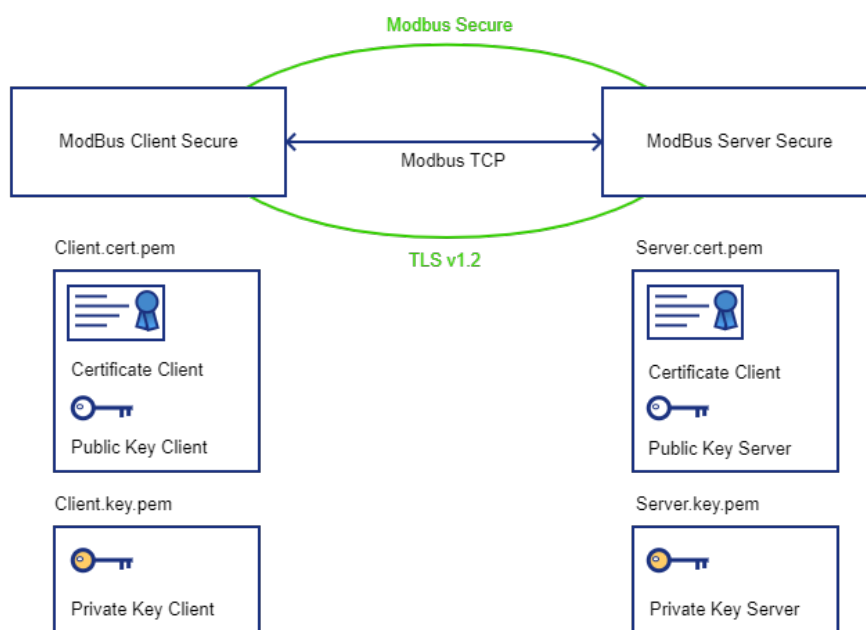


Figure 10.1: Modbus Secure certificate scheme

The standard requires certain extensions to be added to the x.509v3 certificate including an OID (Object Identifier standardized by the International Telecommunications Union) to define the role of the client at the time of authentication. In fact, the specification stipulates that all clients can use read functions but only clients with the role of "operator" can use write functions (of coils or holding registers).

## 10.1 Modbus Secure Specification

The main specifications of the MB-TCP\_Security-v21\_2018-07-24 regulations are given below:

- The protocol version must be  $\geq$  TLS 1.2
- Cipher suite: RSA for key exchange
- Cipher suite: AES128 for packet encryption
- Cipher suite: SHA256SUM for the integrity check of the messages
- Default cipher: TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA256

## 10.2 X509v3 extensions

The following are the contents of the extensions that the standard requires to be added to a standard X509v3 certificate:

X509v3 extensions:

```
X509v3 Subject Key Identifier:
    38:A4:CC:19:6D:6D:E2:AA:7C:82:75:44:A0:59:39:81:47:D3:13:F0
X509v3 Authority Key Identifier:
    keyid:38:A4:CC:19:6D:6D:E2:AA:7C:82:75:44:A0:59:39:81:47:D3:13:F0

X509v3 Basic Constraints:
    CA:FALSE
X509v3 Key Usage: critical
    Digital Signature, Non Repudiation, Key Encipherment
1.3.6.1.4.1.50316.802.1:
    ..Operator
X509v3 Subject Alternative Name:
    IP Address:192.168.1.10
```

## 10.3 Certificate generation

The steps to generate certificates with openssl are as follows (multiple rows are handled so the commands can be copied and pasted directly to the shell console):

Generate certificate and private key for the server (change the ip to the correct server address):

```
# Server
openssl req -x509 -newkey rsa:4096 -sha256 -days 360 \
-keyout server.key.pem -out server.cert.pem \
-nodes -subj "/C=IT/ST=Italy/L=Rovereto/O=ModBusServer/OU=ModBusServer/CN=ModbusSecurityServer" \
-addext "keyUsage=critical,digitalSignature,nonRepudiation,keyEncipherment" \
-addext "subjectAltName=IP:192.168.1.20"
```

Generate one or more certificates/private keys for the clients that will connect to the server (three examples are given below, as before the ip address must be updated with the correct one):

```
# Client 1:
openssl req -x509 -newkey rsa:4096 -sha256 -days 360 \
-keyout client1.key.pem -out client1.cert.pem -nodes \
-subj "/C=IT/ST=Italy/L=Rovereto/O=ModBusClient/OU=ModBusClient/CN=ModbusSecurityClient" \
-addext "keyUsage=critical,digitalSignature,nonRepudiation,keyEncipherment" \
-addext "1.3.6.1.4.1.50316.802.1=ASN1:UTF8String:Operator" \
-addext "subjectAltName=IP:192.168.1.10"
```

On some operating-systems very often the certificate+private key pair is merged into a single password-protected file (.pfx). To merge certificate and key into a single pfx file use the following command:

```
openssl pkcs12 -export -out client1.pfx -inkey client1.key.pem -in client1.cert.pem
```

To view the content of a certificate use the following command:

```
openssl x509 -in client1.cert.pem -text
```

## 11 | Other notes

The maximum number of addresses that can be read with a single command is 123 for TCP requests or 125 for RTU requests.

An error message is displayed if more records are entered.