WoMaster

User Manual

Advanced tutorial on MQTT and JSON to Modbus gateway usage

Embedded device networking solutions

www.womaster.eu

CATALOGUE

1. summarize

This article describes the use of NPS series devices that support MQTT and JSON. Required to work with version 5.10 of vircom.

MQTT and JSON can be used alone or in combination. JSON supports the conversion of Modbus RTU format to JSON format.

The main features are:

1. Establish a connection with the server using the MQTT-based protocol, and conduct data communication in the form of subscription and publication.
2. Support the independent design and automatic collection of Modbus RTU register.
3. Support to convert specific Modbus register contents into JSON format and send them regularly.
4. The device ID, time, and any string can be added to the JSON format.
5. Support nested writing method in JSON format.
6. The NTP protocol is supported to automatically obtain the time.
7. Support unsigned data and signed data, support decimal point representation, support 4-byte length data.
8. All configurations can be configured on the interface. No customization is required.
9. In addition to MQTT, the protocol can support HTTP POST and GET.

2. JSON Simple example

2.1. Convert Modbus RTU to JSON

Modbus RTU to JSON can realize the automatic collection of Modbus RTU table, and automatically send to the cloud server in JSON format.

Here we illustrate this use with a concrete example.

2.2. Modbus table

Suppose you now have a Modbus table with function code 3 and address 1, with the following register addresses and parameter names. The byte length of 4 indicates that two registers need to be read consecutively.

Register address	Parameter name	Byte	Remarks
0	Current total active power	4	Unsigned, 2 decimal place reserved。
97	A-phase voltage	2	Unsigned, 1 decimal place reserved
98	B-phase voltage	2	
99	C-phase voltage	2	
100	A-phase current	2	Unsigned, 2 decimal place reserved
101	B-phase current	2	
102	C-phase current	2	
119	frequency	2	
358	B phase active power	4	
360	C phase active power	4	
362	Total active power	4	

Signed means that the highest bit of a 2-byte or 4-byte is a sign bit, for example, 0xFFFF will be considered -1. Keeping 2 decimal places means that after converting the data as an integer, the decimal point is moved 2 places from the far right to the left.

2.3. Device configuration

We configure the device as a client.

Using SocketDlgTest (https://www.womaster.eu/download_161_200.htm), to monitor a TCP server on the local computer port 1883.

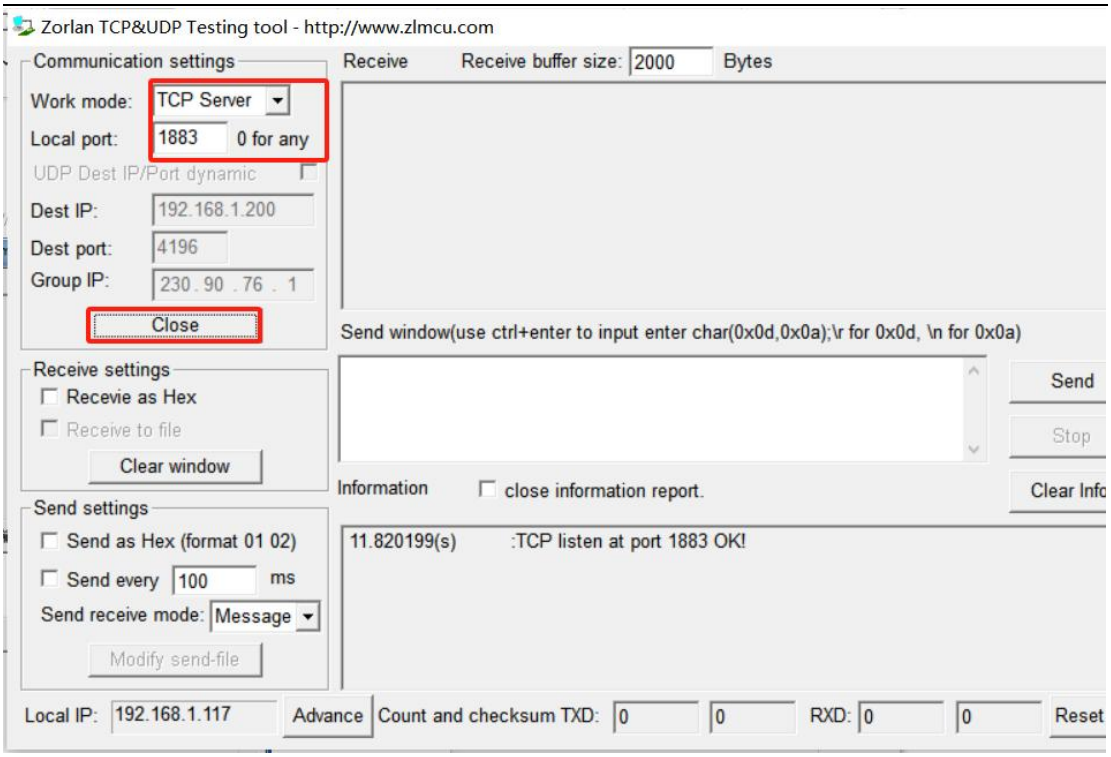


Figure 1Socket simulated server receiving data

Vircom was used to configure the equipment.

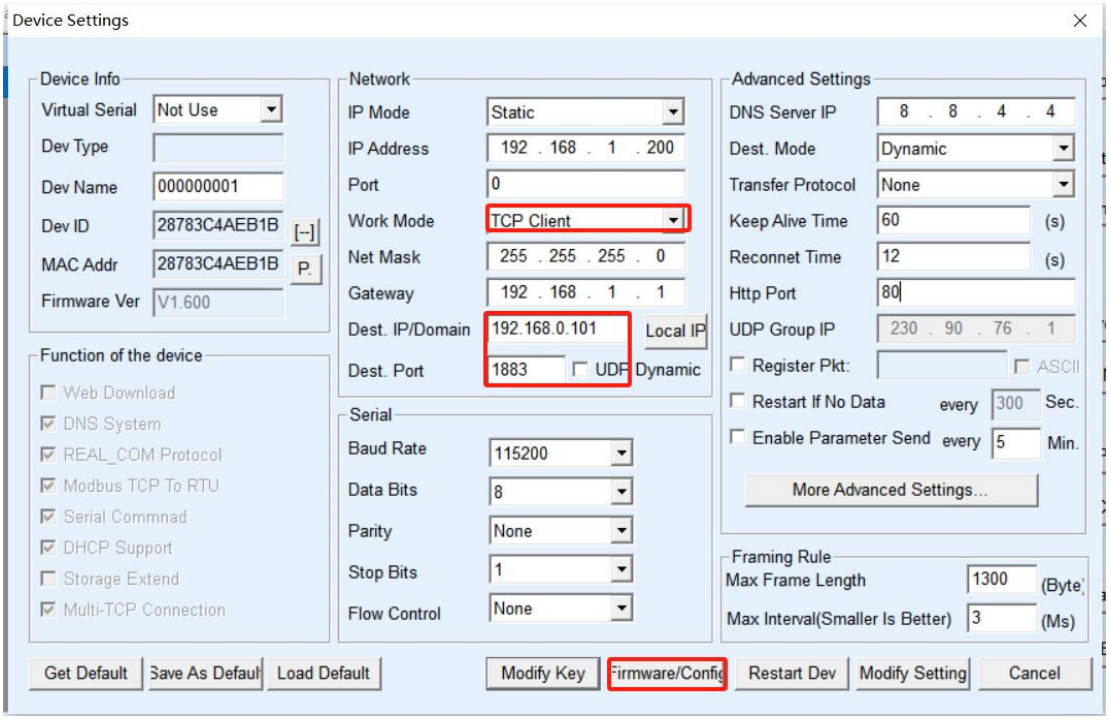


Figure 2 Device configuration

Click Modify Configuration to connect the device to the SocketDlgTest tool. Go to the Device Edit dialog box again. Click the "Firmware and Configuration" button.

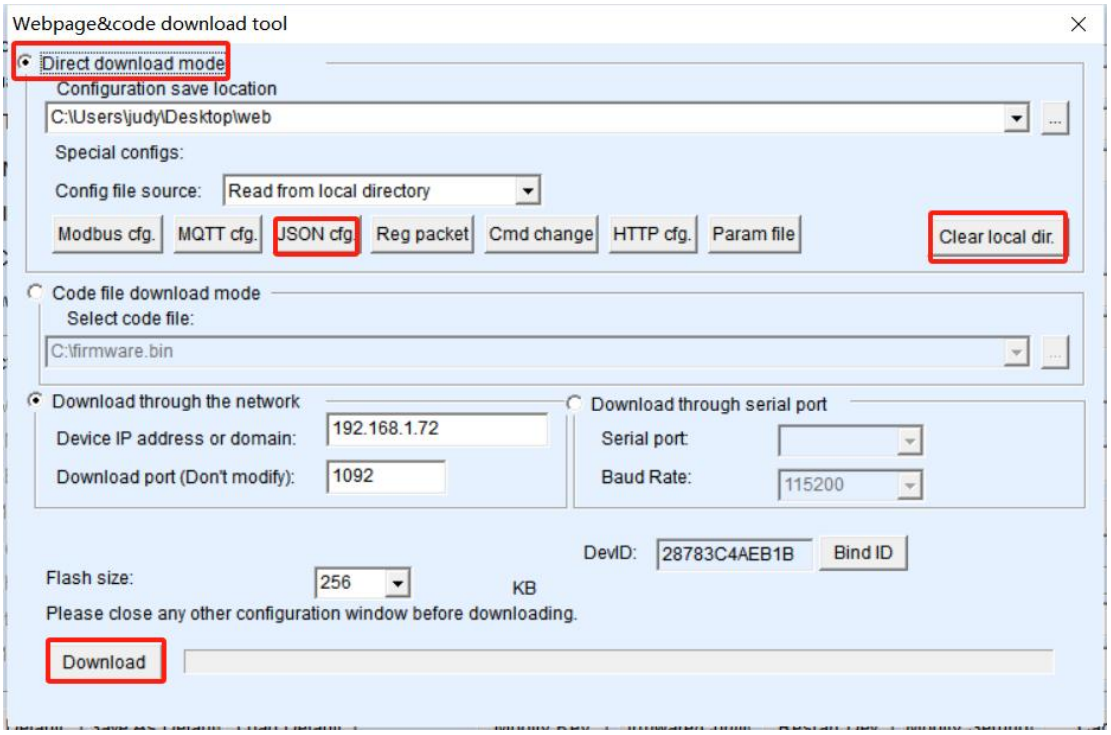


Figure 3 Download interface

First, click "Web Directory Download" to enter the configuration download mode. Then select a new empty directory, such as MQTTHTTPD directory. To prevent the previous design from remaining, please click the "Clear all" button first, so that you can clear the previous design content. The design file will be saved in this directory and can be downloaded to the device later by clicking the "Download" button.

Click the "JSON Configuration" button.

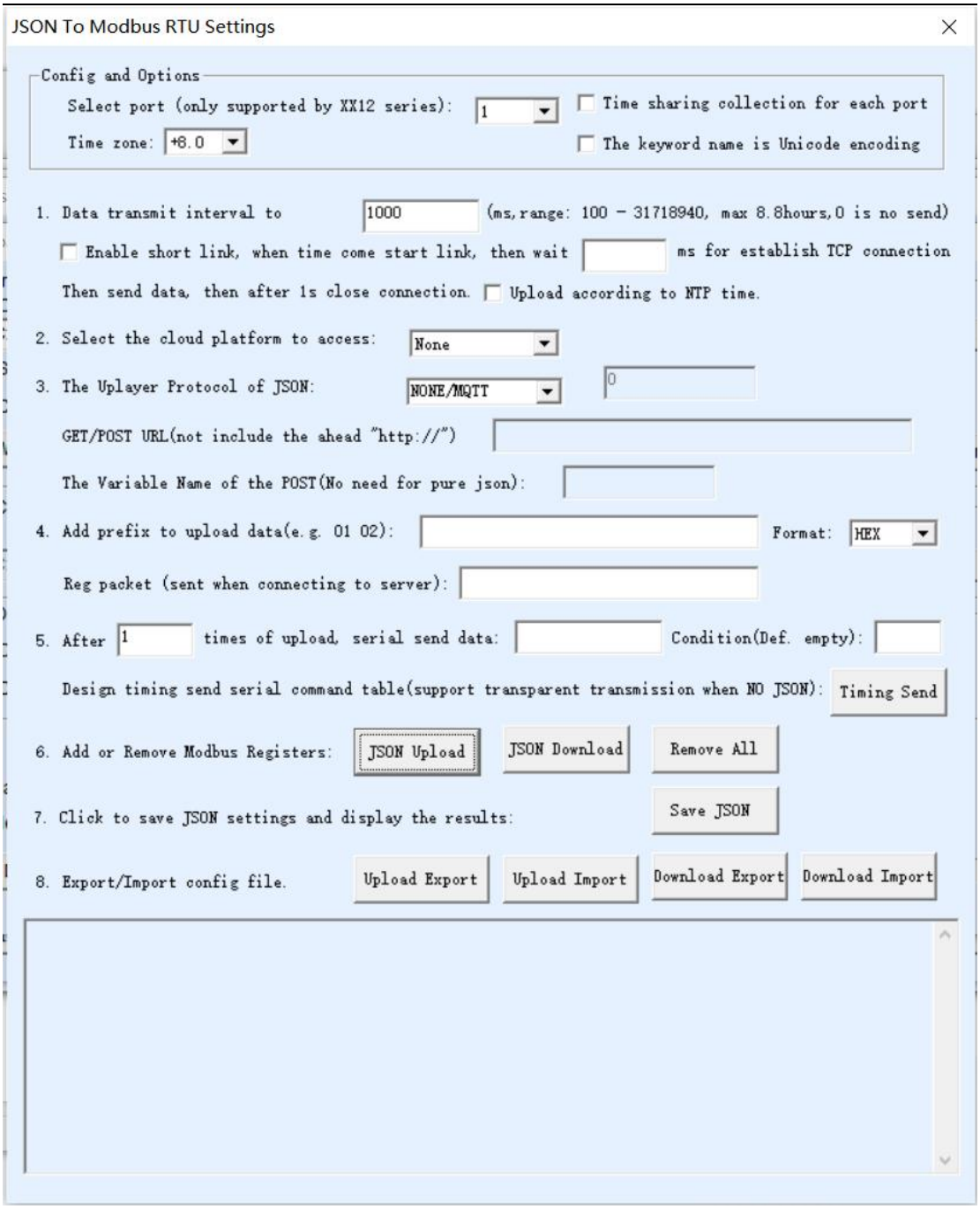


Figure 4JSON configuration main interface

The parameters are described as follows:

1. Sending server time: The default JSON data is sent to the server every time, the server is just set on the device configuration interface of the destination IP address, the unit is milliseconds.

2. Add/view: After clicking, you can design every other JSON node, and you can also view the content already designed.
3. Delete all: Delete all Modbus registers designed by the "Add/View" button, so that you can start the design again.
4. Save JSON Settings: After the design is completed, only by clicking this button can you save the data to the download directory just now, and then download it to the device.

Now click the "Add/View" button. For the first row of the previous Modbus table:

Register address	Parameter name	Byte length	remark
0	Current total active power	4	Unsigned, 2 decimal places reserved.

The corresponding configuration is as follows:

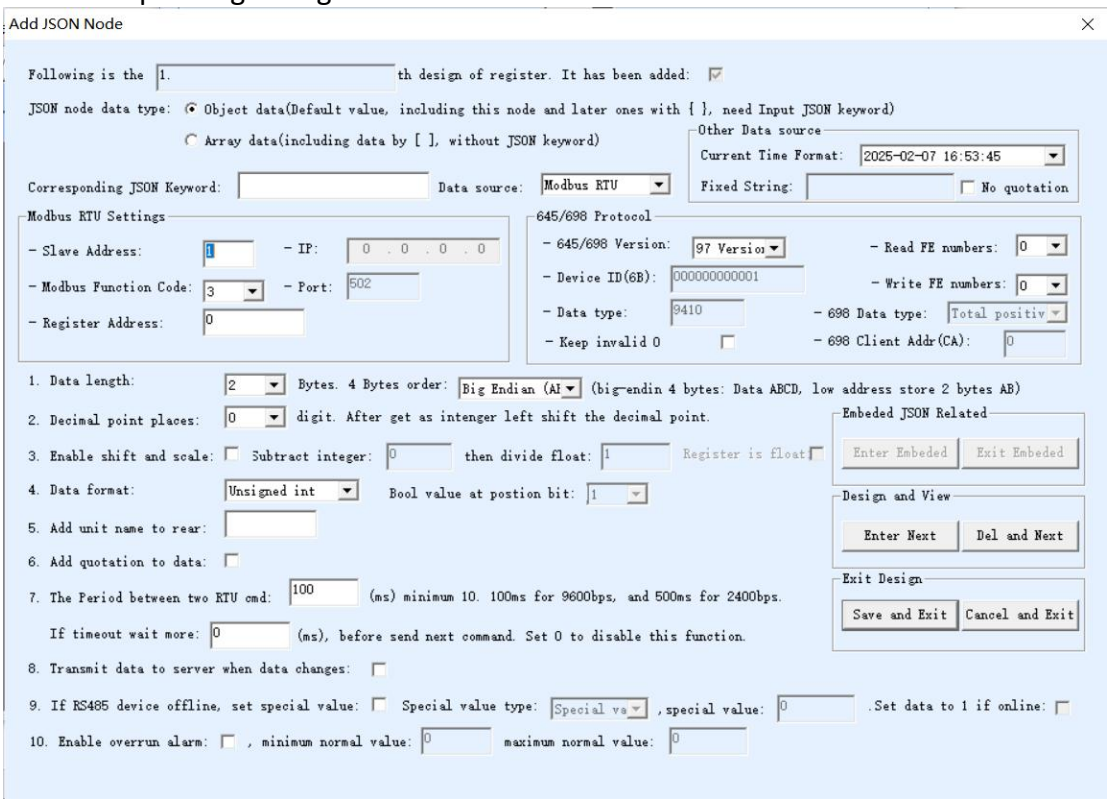


Figure 5 Register Settings

The parameters are described as follows:

1. The following figure is the first JSON keyword: the "1." here represents the current JSON keyword of the design interface, if the second is "2.", if it is the

second JSON under the nested node is "2.1", and so on.

2. It has been added: If a check mark indicates that it has been added, a check mark will be displayed when viewing the configured information, indicating that it is in the editing state. If it is not checked, it is in the added state.
3. Corresponding JSON keyword: Name of the JSON node.
4. Data source: Select the source of JSON data
    a) Modbus RTU: For example, CurrentW:123.45 indicates that data is collected from a Modbus RTU table through the serial port. The left half of the figure shows the parameters related to Modbus RTU design.
    b) Fixed string: for example, in the form of DevName: "MyDev", enter MyDev in the fixed string on the right, and the JSON name is DevName, so that a JSON node with a fixed string can be generated.
    c) Device ID: If the JSON node name is DevID, the string of the node sent is DevID: "285301020304", where 285301020304 is the MAC address or unique number of the device.
    d) d) Current time: If the JSON node name is ColletTime, the string sent is ColletTime: "2019-05-13 22:23:31". The system obtains the time through the NTP protocol.
    e) Nested JSON: If the node name is Alarm, it is sent in the form of Alarm:{temp1: "25.1",temp2: "26.2"}, that is, the contents of the Alarm are still a JSON collection.
5. Modbus related Settings
    a) Slave address: address of the Modbus table.
    b) Modbus function code: Currently, function codes 03 and 04 are supported.
    c) Register address: the corresponding 0 here.
    d) Data length: This corresponds to 4 bytes.
    e) Data format: here corresponds to an unsigned integer.
    f) Keep the decimal point: keep 2 digits here.
    g) Add units after data: For example, if "CurrentW" :25.6W, the W after 25.6 is to add large units. Write the "W" in this box.
    h) Add quotes to data: If this parameter is selected, change

"CurrentW" :25.6W to "CurrentW" :25.6W.

i) Serial port polling time: Set this parameter to 100ms. Refers to the polling interval between this register and the next register, not the polling interval of this instruction.

6. Fixed string: If the source is set to a fixed string, you can enter the string content.
7. Buttons
   a) Nested JSON: When the current node source is selected as "nested JSON" type, you must click this button to enter the design of nested JSON, if the current is "2.", it will enter the design of the "2.1" node.
   b) Return to previous level: If the current node is nested at level N, clicking this button will return to the design of the N-1 node and stay on a new node at level N-1.
   c) Design the next one: Click to enter the next local JSON node. If the next node does not exist in the previous design, the check mark of "already added" will be cancelled, indicating that it is a new node.
   d) Save the design: To complete the design, click "Save Design" at the last design node interface. Then return to the main screen and click "Save JSON configuration".
   e) Cancel the design: cancel all the current design, if you want to view the design content, you can click this button to exit.

Click the "Design Next" button here to continue designing other registers in the Modbus table. After designing all the registers in the table, click "Complete design" and then click "Save JSON configuration" to exit. Then click the "Download button" on the "Download web" page

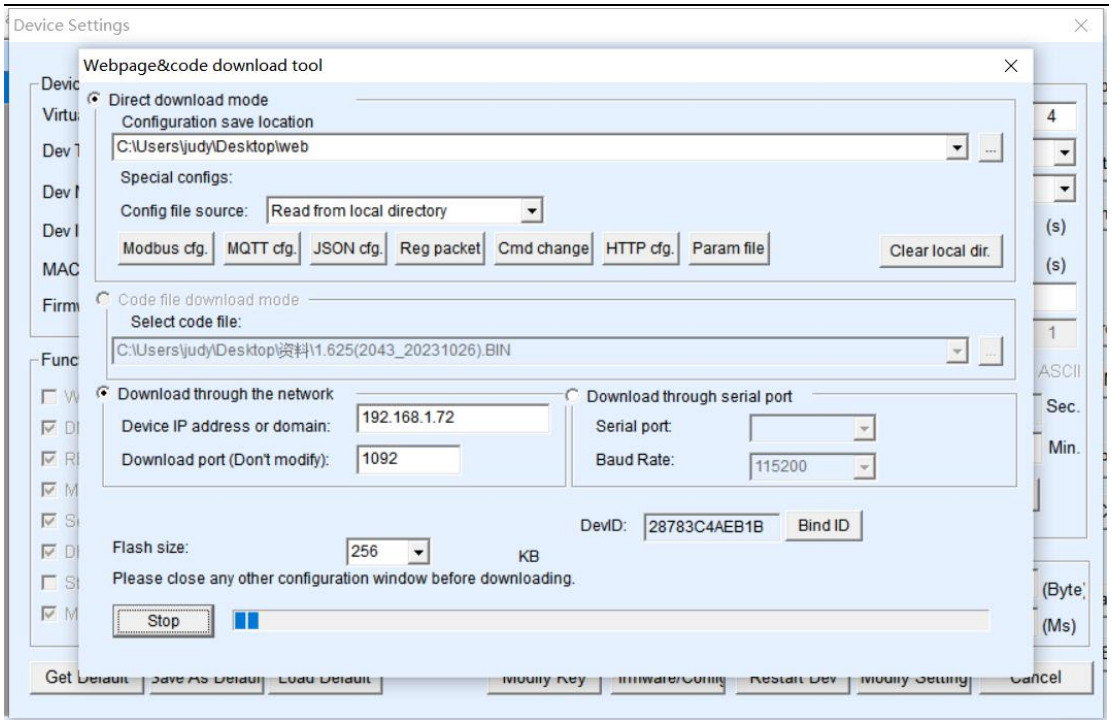


Figure 6 Download

Then click "OK" and the device will automatically restart. If no, restart it manually.

2.4. Create a Modbus analog table.

Here Modbus Slave is used to simulate a table

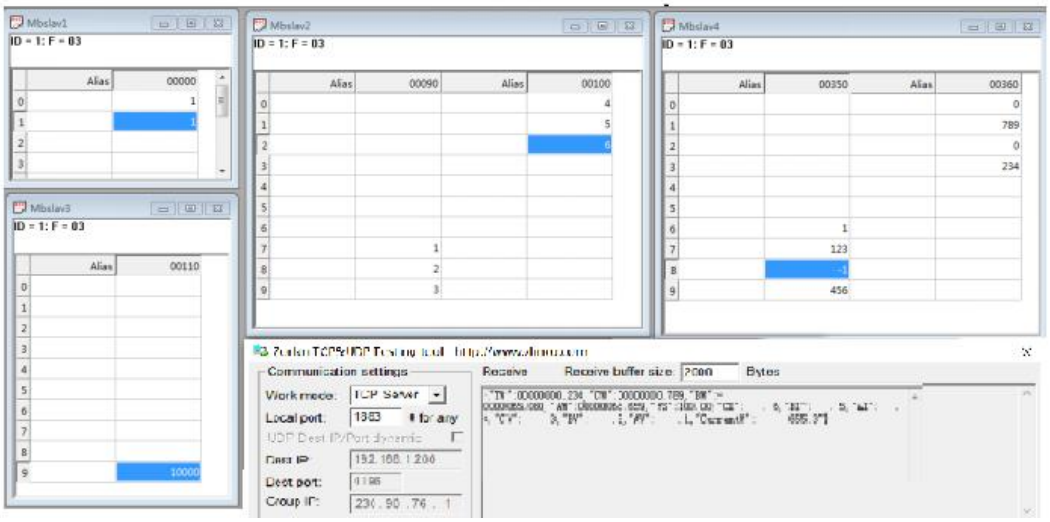


Figure 7 Test results

The test results show that the instrument simulated by Modbus slave tool can be collected by gateway. At the same time, it can be periodically sent to the server software of SocketDlgTest simulation in json format.

3. JSON Complex example

3.1. Enable NTP

To use JSON with time, you must enable the NTP function of the device. The NTP function can obtain the current time over the network.

In the web download directory where httpd.txt is located, create an empty txt file with the following content:

[NTP]
NTP_SERVER1=a1.a2.a3.a4
NTP_SERVER2=b1.b2.b3.b4
NTP_SERVER3=c1.c2.c3.c4
RE_ARUIRE_TIME=0

NTP_SERVER1, NTP_SERVER2, and NTP_SERVER3 are the IP addresses of NTP time servers. Set this parameter based on actual conditions. Up to 3 servers can be set up, but you must write from NTP_SERVER1, if there is only one write NTP_SERVER1, and if there is only 2 write NTP_SERVER1 and NTP_SERVER2.

After saving, ntp.txt is downloaded to the device together with other files.

3.2. Nested JSON design

Suppose we need to design JSON like this:

```
{“header”:{“DEVID”:”285301020304”,
                “time”:” 2019-05-13 22:23:31”},
 “data”:   {“id”:”MyData123456”,
            “alarm”:{“alarm1”:123.4C
                     “alarm2”:567.8C
                  }
```

},

“value”:2345

}

The design steps are as follows:

1. Step 1 keyword is "header", then select "JSON nesting" from the source, and then click the "Design nested JSON" button.
2. Enter the step 1.1, here design "DEVID" : "285301020304", enter the keyword as DEVID, select "device ID" from the source, and click "Design Next".
3. Enter the step 1.2, where "time" is designed: "2019-05-13 22:23:31", enter the keyword as time, and select "current time" as the source. What should be noted here is that although the design of level 1 has been completed, it is still necessary to click "Design Next", and then enter the step 1.3. Click "Go back to previous level". This step 1.3 is abandoned automatically.
4. Go to Step 2. Enter the keyword data here, then select "JSON nested" from the source, and click the "Design nested JSON" button.
5. Enter the step 2.1, here design "id" : "MyData123456", enter the keyword id, select the source as a fixed string, and then enter "MyData123456" in the fixed string box, click "Design next".
6. 6. Go to Step 2.2, where enter the keyword alarm, then select "JSON nested" from the source, and then click the "Design Nested JSON" button.
7. 7. Enter the step 2.2.1 to design "alarm1" :123.4C, which is a Modbus data with unit, function code is 3, register is 0, then the design is shown as follows:

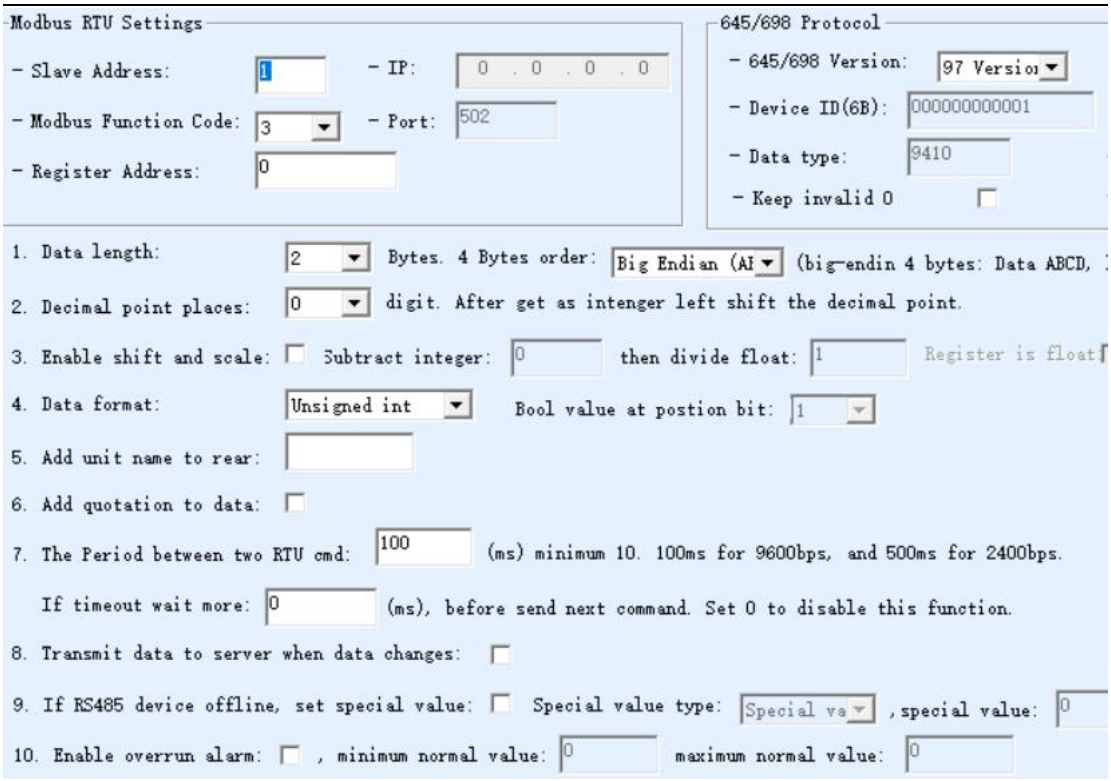


Figure 8 Register design

Then click "Design Next".

8. Go to Step 2.2.2 to design "alarm2" : The method is similar to alarm1, where the register address is set to 1. Similarly, click "Design Next" first, and then in the "2.2.3" step, click "Go back to previous level".
9. Enter 2.3. As 2.3 does not require design, click "Return to previous level" at this time. Because there is still "value" :2345 not designed, otherwise you can directly "save the design".
10. Enter the third., where a Modbus data with the keyword value is designed. Now click "Save Design". Note: If "value" :2345 does not exist here, then you need to directly click to save in the previous step, if you have accidentally clicked "return to the previous level" to enter the third here, but the third does not exist, then you can use the new version of vircom "delete and go to the next" to delete this useless node.
11. Return to the JSON to Modbus RTU Settings screen and click Save JSON Settings. Then download it to the device and use it.

When the TCP connection is established, the following data is received:

{"header":{"DEVID":"2850002F0EEC","time":"2019-05-13 23:41:26"},"data":{"id":"MyData123456","alarm":{"alarm1": 123.4C," alarm2":" 567.8C"}},"value": 2345}

Note that if you are editing the current JSON design, you need to first select the correct directory in the Web download interface, and follow the design steps to click the button in order to fully browse all nodes.

3.3. Read the bits of the byte register

Sometimes the data read by Modbus using 03/04 function code will also use bits to represent specific meanings. For example, the 00 address register read by 03 function code is 0x8183, in which bit16, bit9, bit8, bit2 and bit1 are all 1, and these bits have different meanings. When 1 indicates a different alarm. So you also need to upload with different json keywords. This feature requires 2003 firmware 1.579 and above, designed using vircom version 5.13 and above. Here's how:

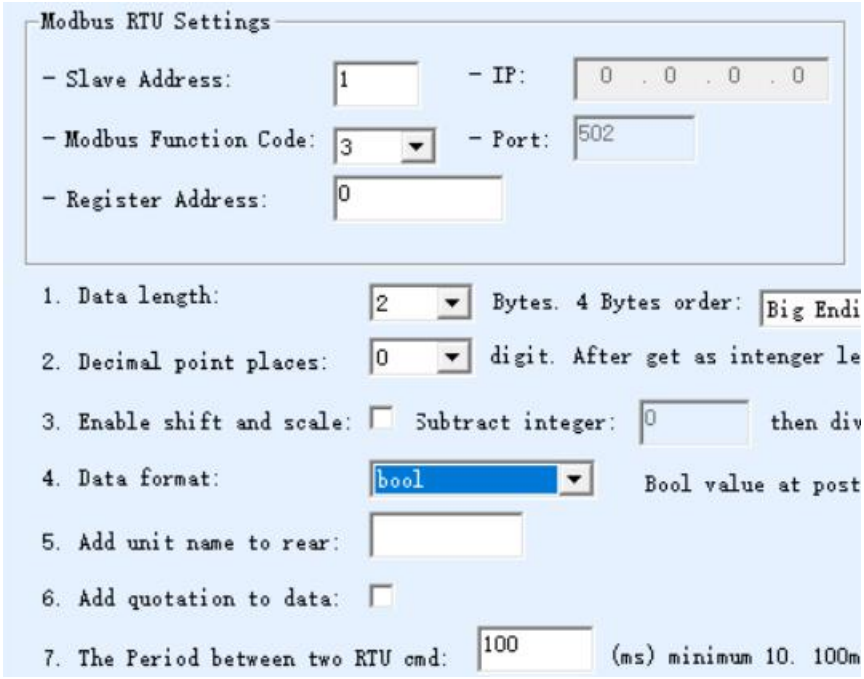


Figure 9 byte register

The design method is basically the same as the previous method, the only thing

to note is that the data format is selected as "Boolean", and then where the Boolean value is located, the bit position of the json variable bit2 is selected in the 03 function code and the bit position of the 00 register. If the register value is 00 20, the 1 here is in the 2 position.

Note that you can set the same Modbus station address, function code, and register for different json keywords, and you can obtain different variable contents as long as the location of the Boolean value is different. For example, we designed bit1, bit2, bit8, bit9, bit16, and get the following json data return when the register content is 0x8183: {"bit1":1,"bit2":1,"bit8":1,"bit9":1,"bit16":1}

3.4. 01 and 02 Function codes

You can set a JSON node for the 01/02 function code bit register, but each JSON node needs to set the register address once, so the number of bits read each time is one bit. The bit of the byte register is different from that of the byte register: the bit of the byte register is still read by the 03/04 function code, but only the value of a certain bit of the two bytes is taken; 01/02 function code itself is read bit, because only 1 bit is read at a time, so the position of the Boolean value is generally written 1, if it is 1, it shows' 1 ', otherwise it shows' 0 '.

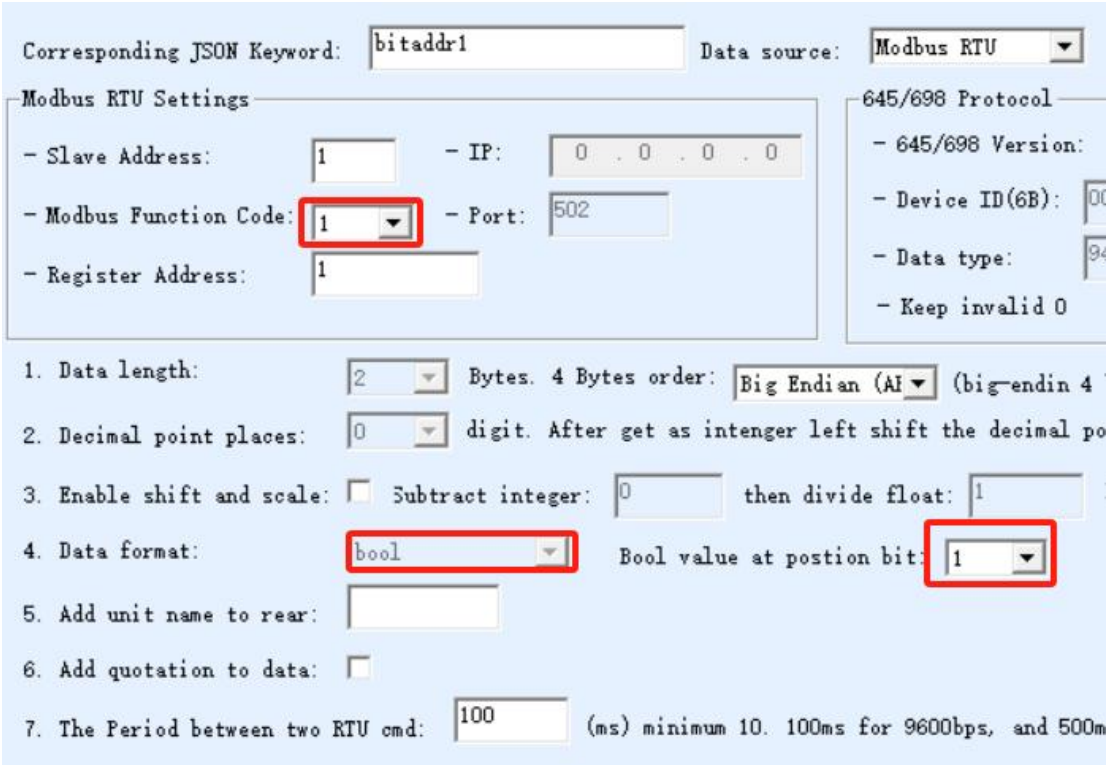


Figure 10 byte register

When the Modbus function code is selected as 01/02, the data length, data format, and reserved decimal places are not optional.

3.5. Display design results

Now click "Save JSON design" after the design, you can see the content of the completed JSON format in the display box, so as to facilitate the overview of the design. At the same time, there is also an index to compare when entering "Add/View".

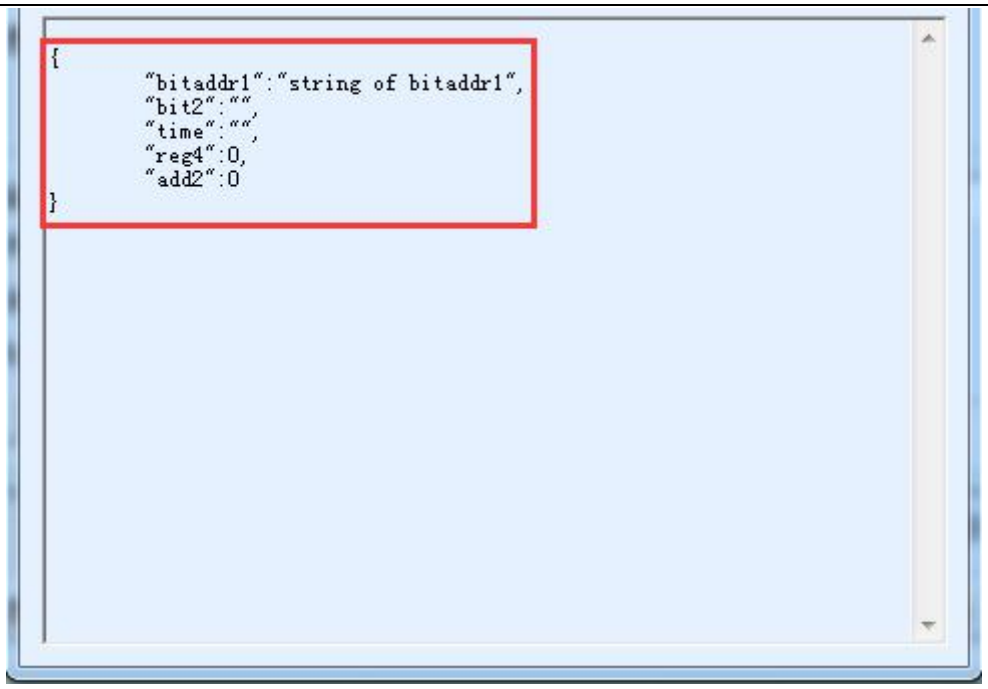


Figure 11 shows the results

3.6. Edit in excel mode

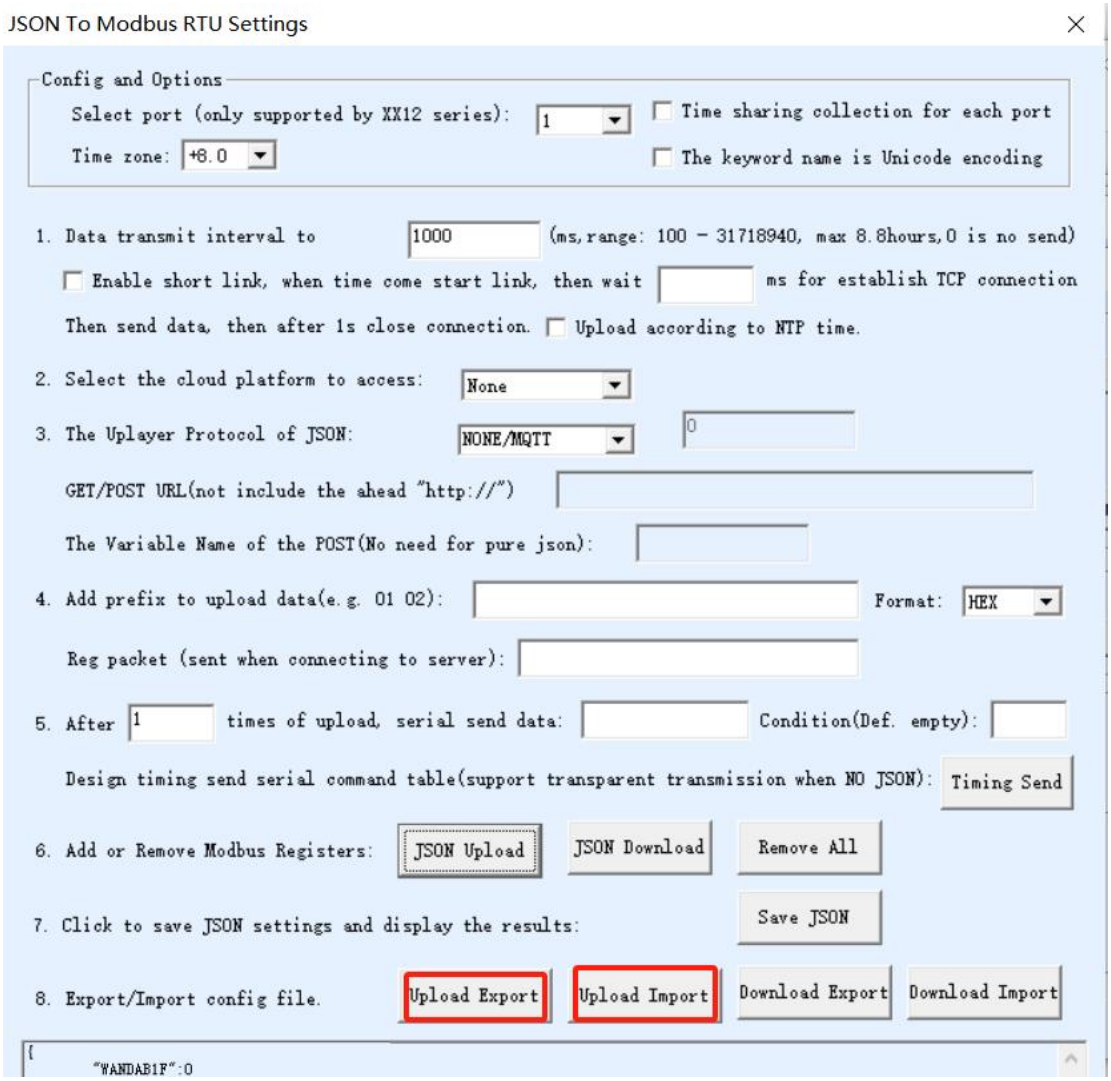


Figure 12 Importing and exporting CSV format

In order to facilitate editing, the content of the design can be exported to CSV format, and then edited with EXCELL, and then saved as CSV, and then imported.

But CSV has a format problem:

序号	JSON关键词	数据来源	固定字符串	Modbus从站	Modbus功能	Modbus寄存	645设备ID	645数据类	数据长度
1	1234	嵌套JSON		1	3	0	2.02E+11	04FF0402	1
1.1.	1235	设备ID		1	3	1	2.02E+11	04FF0402	1
1.2.	1236	当前时间		1	3	2	2.02E+11	04FF0402	1
1.3.	1237	嵌套JSON		1	3	3	2.02E+11	04FF0402	1
1.3.1.	1238	645协议		1	3	4	2.02E+11	2010100	2
1.3.2.	1239	645协议		1	3	5	2.02E+11	2020100	2

Figure 13 Data format error

In this case, you can save the CSV to XLS format for editing. After editing, save as

CSV format and import.

3.7. Brackets and arrays

The new version of Vircom supports JSON array designs. Here's a simple example:

```
{
    "reqBody":
    [
        0,1
    ]
}
```

In this example, there is only one JSON object, reqBody, whose contents are an array with the first element being data 0 and the second element being data 1. Here we treat the array of brackets like nested JSON, except that the nested JSON does not require keyword names and colons. The design steps are as follows:

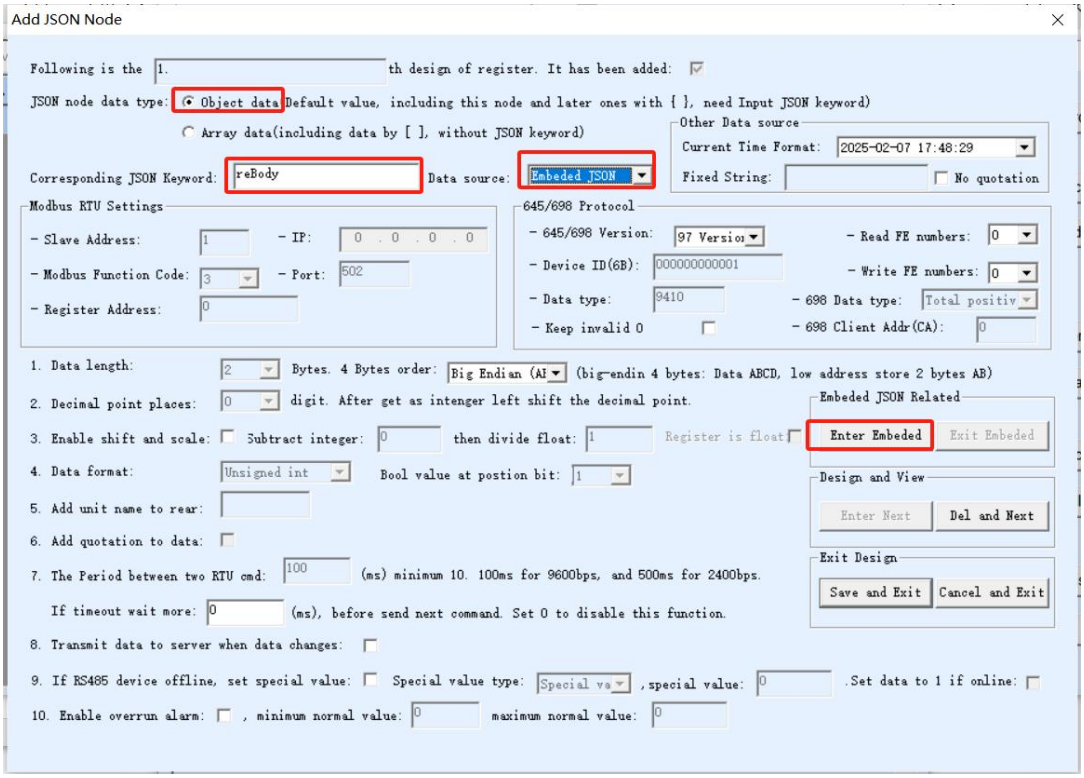


Figure 14 Array objects

Since reqBody itself is an object, not a unit of an array, the node type is chosen for object data, not array data. Since its contents are an array, the source of the data for this reqBody is "nested JSON", following the idea that brackets are braces. Then click "Design Nested JSON". Wherever there are braces and brackets, you need to select the data source as "nested JSON", and whether the node type is "object data" or "array data" depends on whether it is a unit of the array (array element).

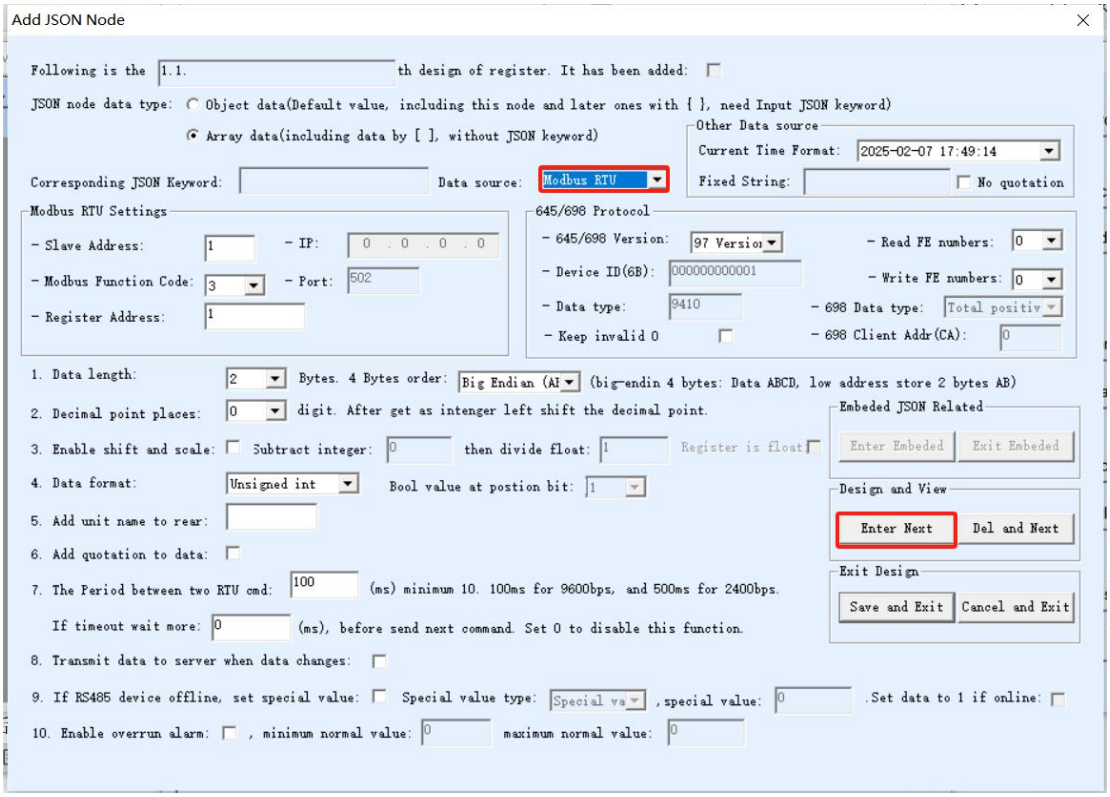


Figure 15 Array contents

Let's design the first contents of the array, because it's the contents of the array. So the node type is array data. The data source is the collection of Modbus RTU, and the relevant parameters of Modbus are filled in. Then click "Go to Next" to design the second element of the array in a similar way. After designing the second element, since there is no more to design, click "Save all and exit" directly. Go back to the previous interface and click "Save JSON Settings", then download it. The data sent is: {"reqBody":[2,3]}.

Now let's look at a more complicated example:

{

```
    "reqBody":
    [
        {
            "workshop_id:"1008",
            "machine_code":"XS114"
        },
        {
            "workshop_id:"1008",
            "machine_code":"XS116"
        }
    ]
}
```

In this case, the contents of the array are not simply data values, but JSON itself.

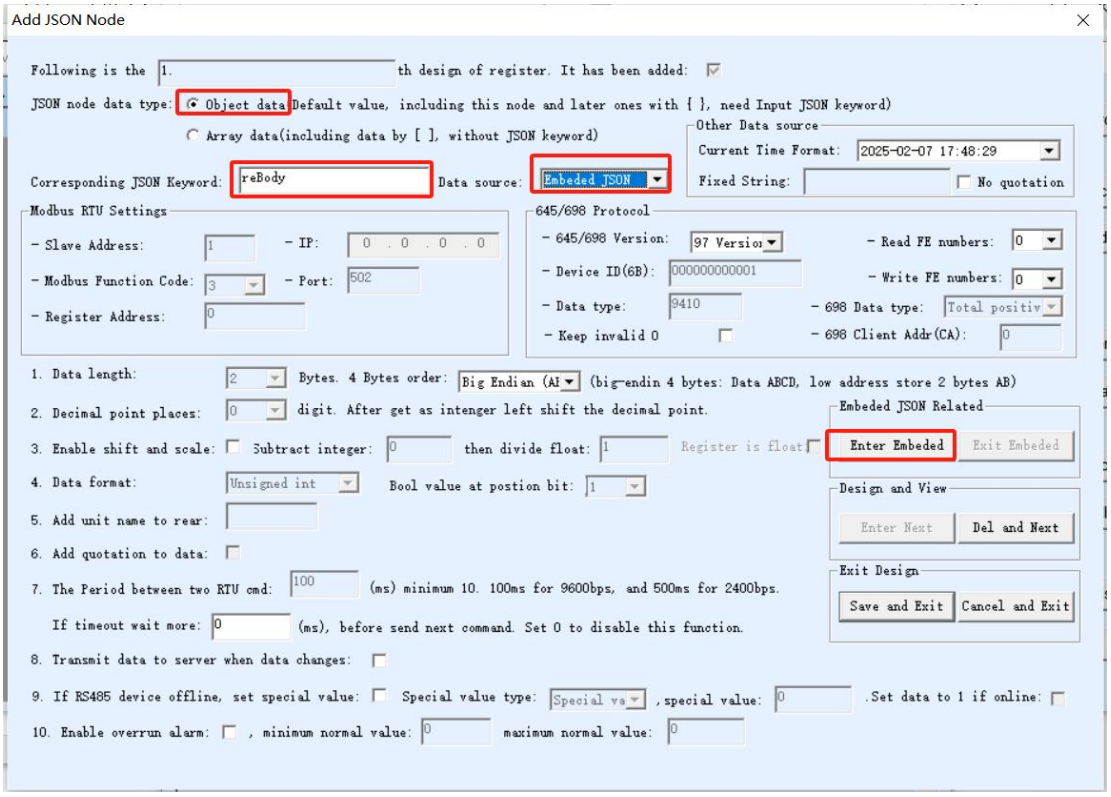


Figure 16 Step 1

Here in 1.1, the previous example 1 is to select the node type as "array data", and then directly design the data in Modbus format, but the data content here is a

JSON object, so you need to select the data source as "nested JSON", and then click "Design nested JSON".

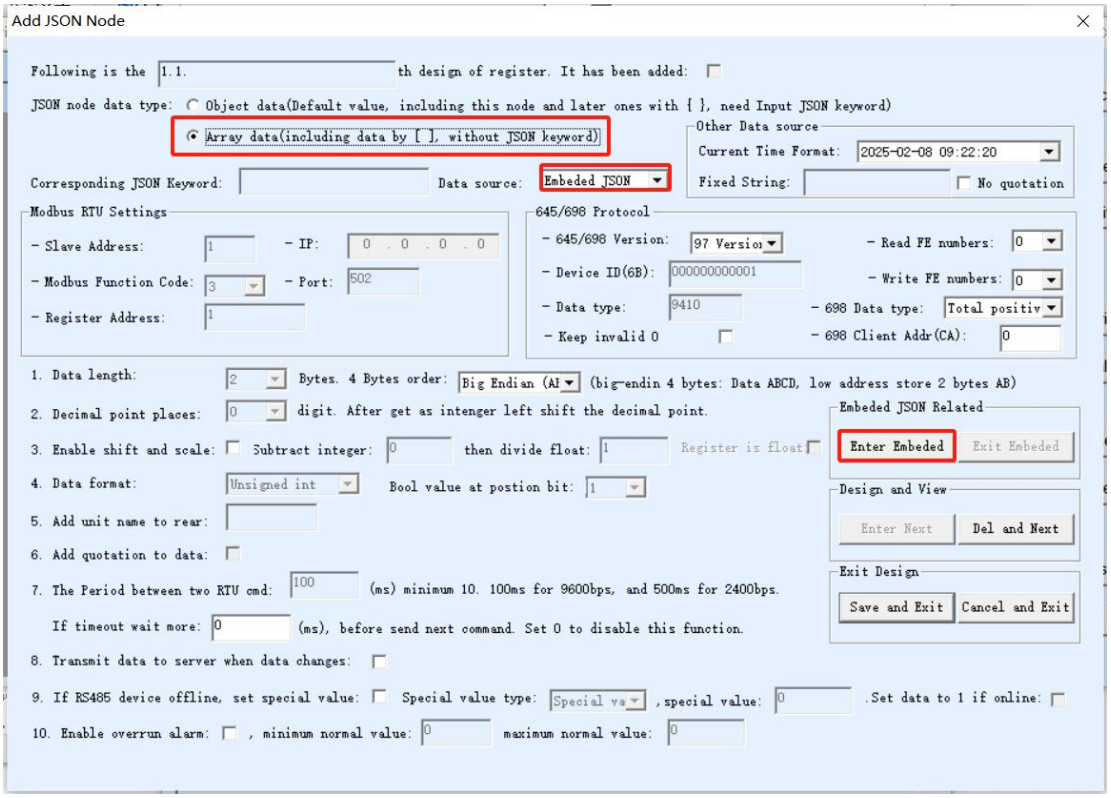


Figure 17 Step 1.1

Then design two object types in steps 1.1.1 and 1.1.2, Modbus source data workshop_id and fixed string source data machine_code.

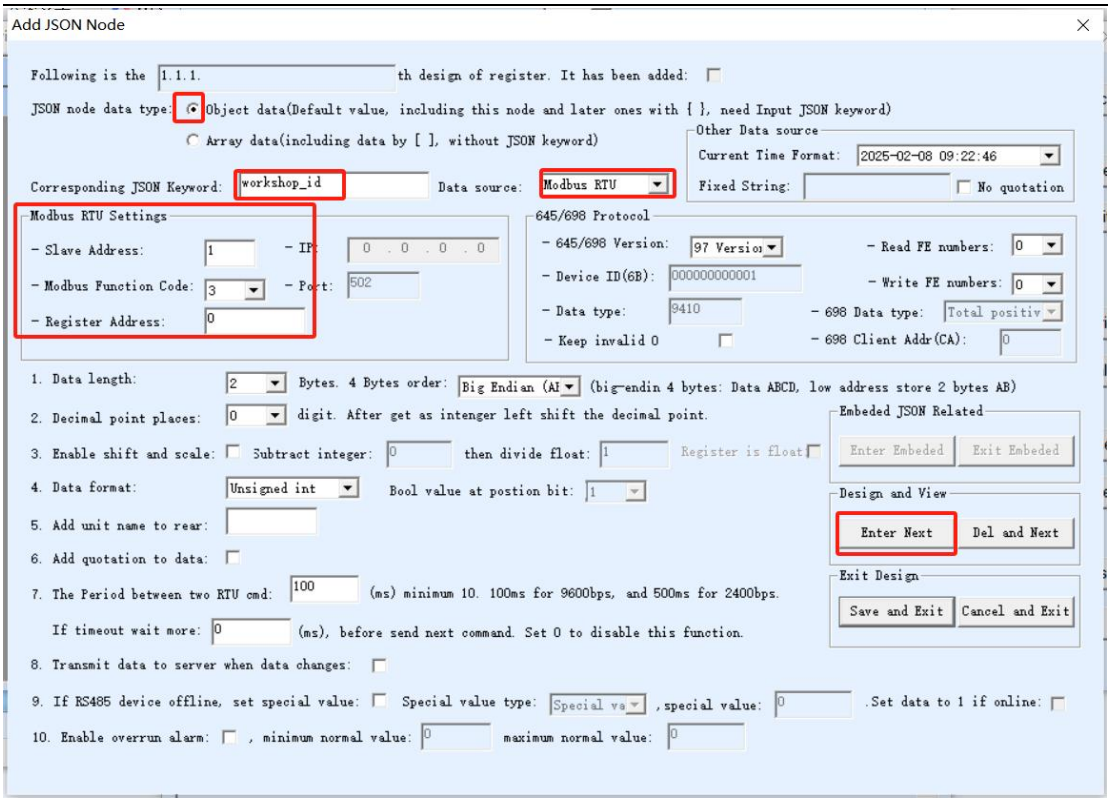


Figure 18 Step 1.1.1

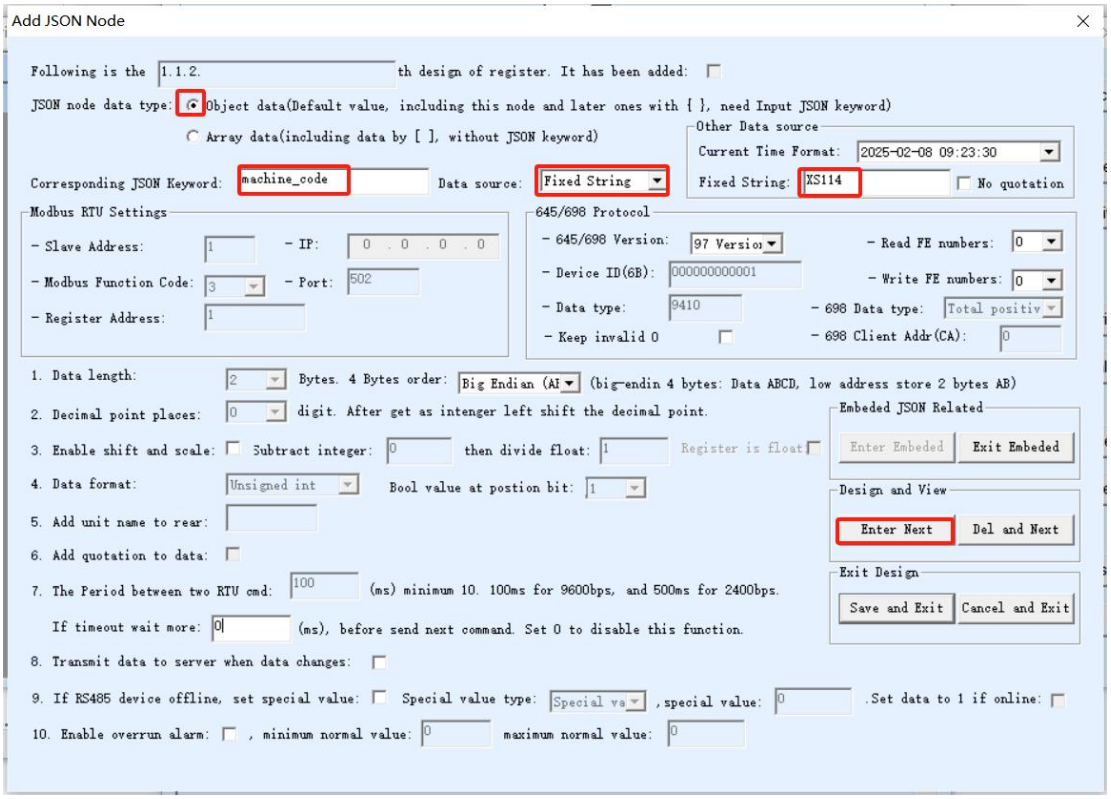


Figure 19 Step 1.1.2

Note here that you need to click "Go to next", in this empty content node (i.e. 1.1.3), click "go back to previous level" to enter "1.2". Here 1.1.3 is actually a non-existent node.

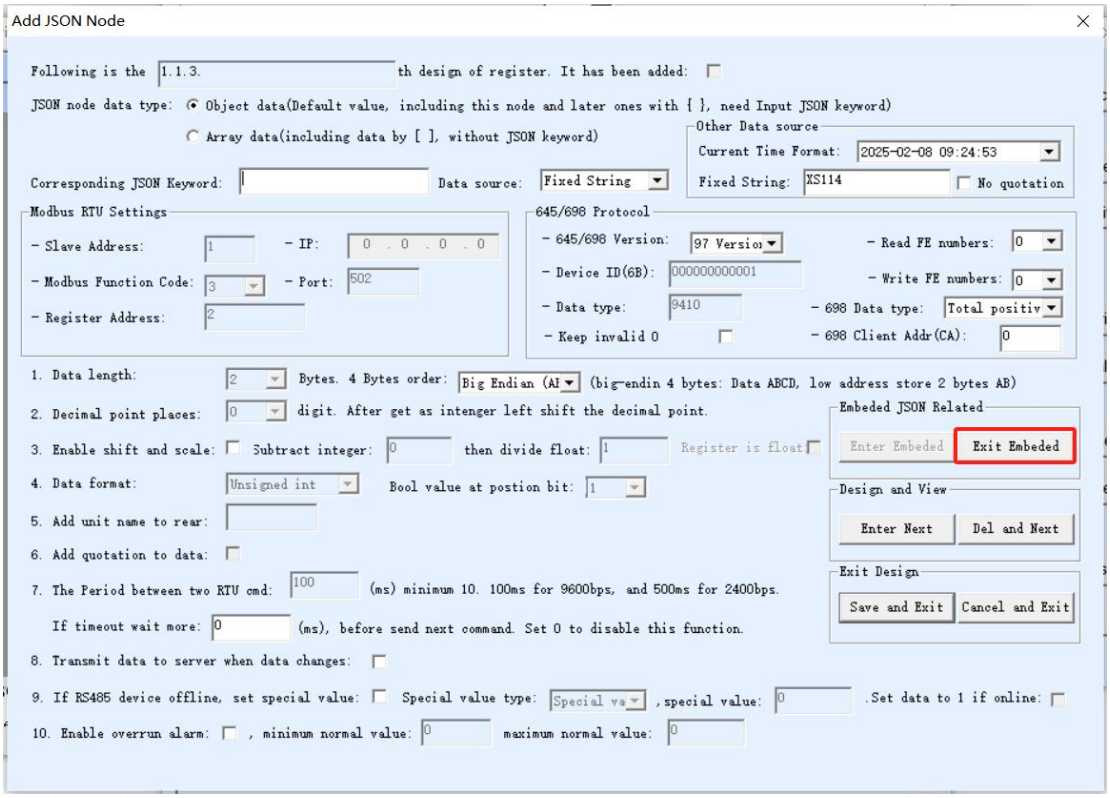


Figure 20 Step 1.1.3

"1.2" is also an array type, the data source is nested JSON, refer to step 1.1, pay attention to select 1. Array data. 2. Nested JSON. Go all the way to "1.2.2" and click "Save All and exit".

The last uploaded data format is as follows, where workshop_id is read from the Modbus register.

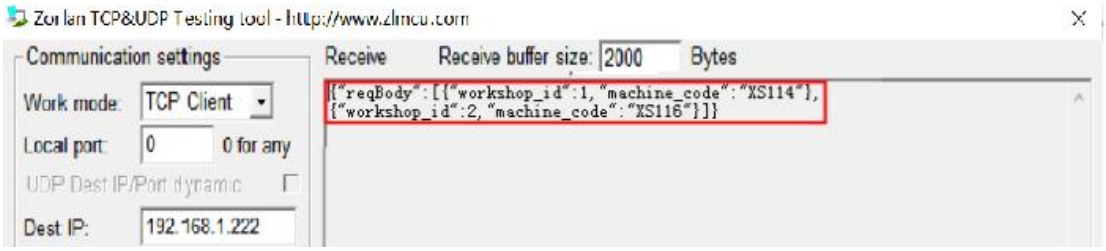


Figure 21 Step 1.1.3

3.8. Small end

Refer to the rest of this article.

3.9. Read failed to clear

When a register cannot be read, a data value of 0 can be used to indicate that the data is not read.

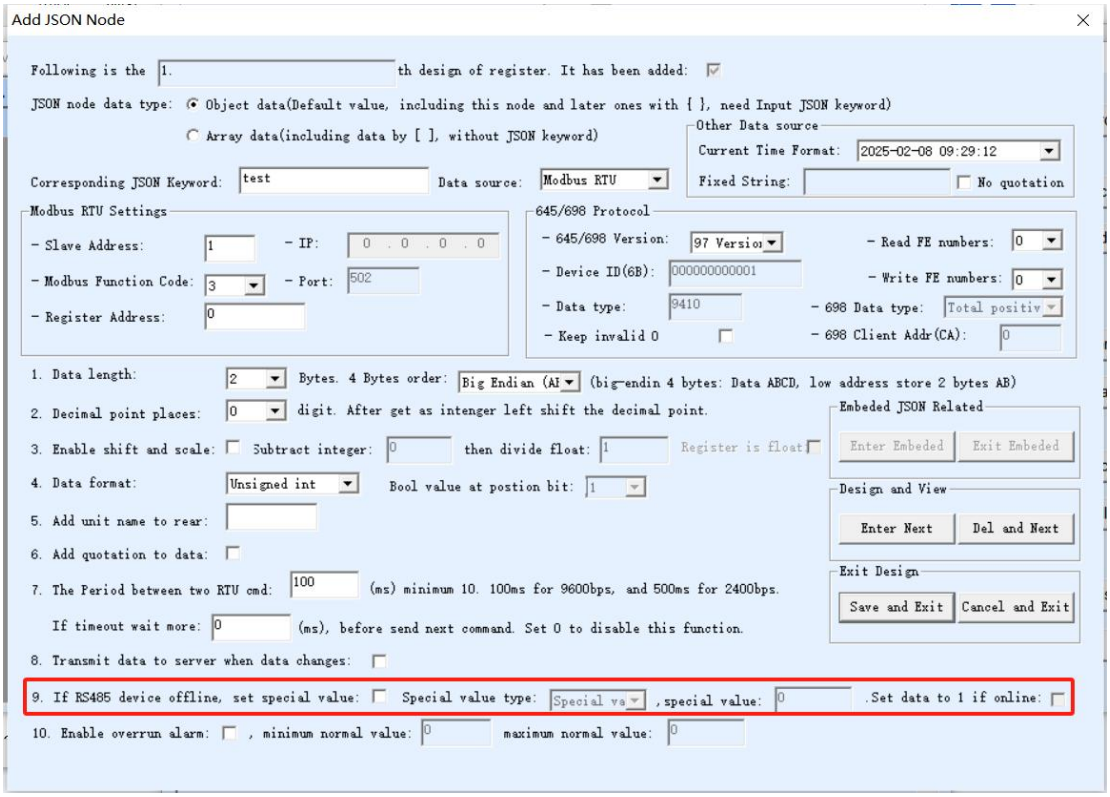


Figure 22 Data clearing

Here we design a test JSON, pay attention to check the RS485 device offline data clearing. When the register can be read, the data sent is {"test":123}, where 123 is the actual register contents. If the device is offline or cannot read data, {"test":0} is sent. This way you can avoid saying that the device is offline and the data still exists, giving people a misunderstanding.

3.10. The device is offline

If no data is read, the data is 0. In some cases, the device cannot be detected

offline. For example, if the data content itself is 0, it is impossible to determine whether the device is offline or the data is 0. In addition, sometimes the device has multiple registers to read, so a separate JSON keyword such as {online1:0} should be 0 or 1 to indicate that the device is online. To do this, you can design online1 as follows. First design all the registers, then add a JSON node:

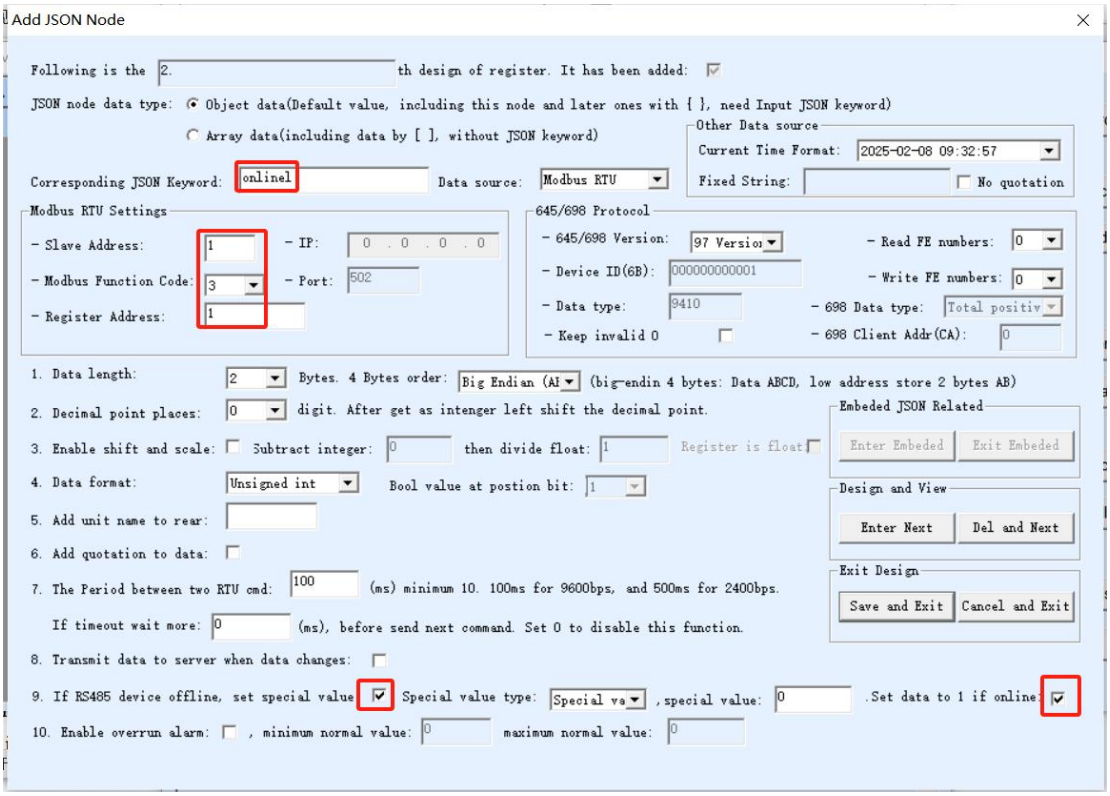


Figure 23 Online device

In this case, the name is online1, and you can choose anything you want. Register number You can select any existing register. The most important thing is to check the Rs485 device offline data zero and if the device is online, regardless of the register content, forcibly set to 1. In this way, even if the read data is 0, as long as the read data is 1. That is, online1 only has 0 and 1 data. 1 indicates online, 0 indicates offline.

3.11. Pan and zoom

Translation and scaling subtract a 2-byte signed integer from the contents of the data read by the Modbus register, and then divide by a single precision floating-point

number to obtain a single precision floating-point number, as shown below

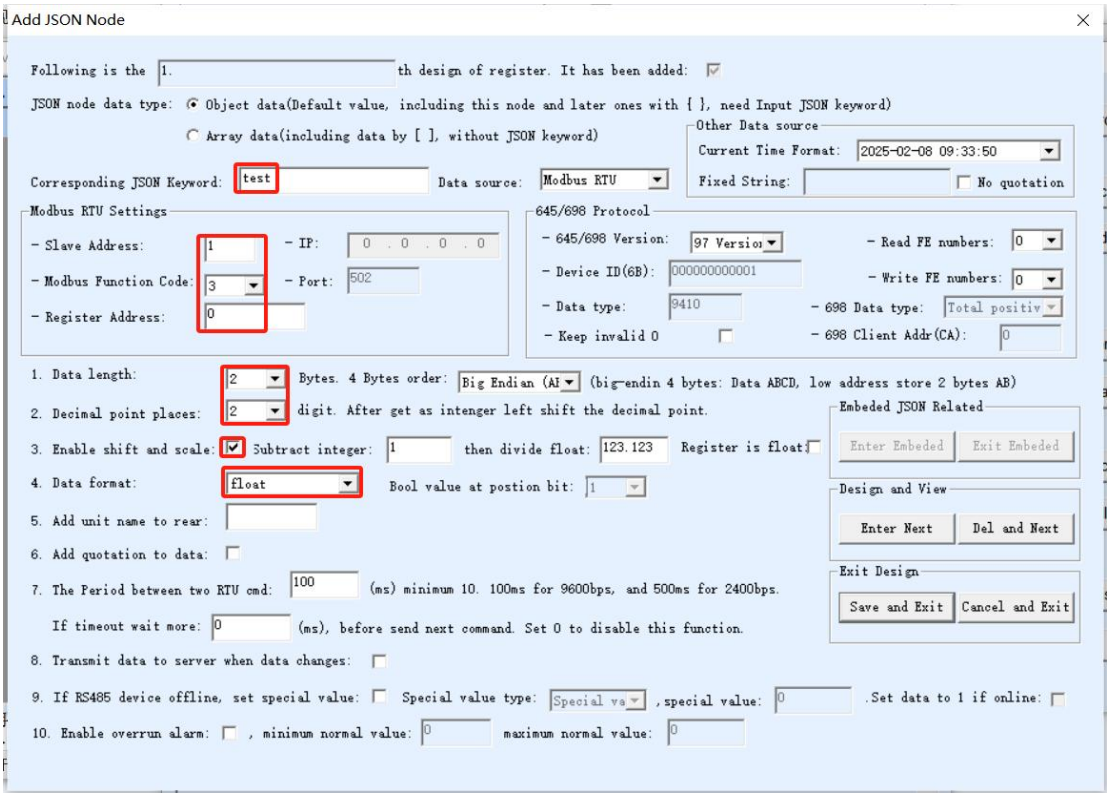


Figure 24 Pan and zoom

As shown in the figure, design a JSON keyword test, read station address 1, register 0 of 1 register (2 bytes), then subtract 1, and then divide by 123.123. This is equivalent to shifting the data read by the register down by 2 units and then shrinking it by a factor of 123.123. When the register content is 124, the data sent is {"test":0.99 "}, (124-1)/123.123=0.99.

Note that only Modbus RTU data sources, 2/4-byte data lengths, 03 or 04 function codes, and floating-point output are supported. The decimal places can be 0 to 4 digits. The main thing is to check the Enable Pan and zoom options. Enter a signed integer ranging from -32768 to 32767 in the Subtracted integer field, and enter a floating-point number in the divided by floating-point field.

Note that as long as one data in the design needs translation and scaling, all other data also need translation and scaling, but the scaling ratio and the translation ratio can be freely set. Data that does not need scaling can be set to subtract 0 and then scale 1.00001. The reason why it is set to 1.00001 and not 1.0 is that if it is set

to 1.0, it automatically assumes that without scaling it is impossible to select a floating-point format for 2-byte data. For an integer value, such as 123, when scaled, the translation is 0, and the divided value is 0.99999, so since the result only takes 0 decimal points, the accuracy is not affected, still 123. So increasing translation scaling has no effect on integers.

Support for various data formats:

1. Supports translation and scaling of 2-byte shaping data.
2. Supports translation and scaling of 4-byte shaping data. Support ABCD, CDAB, two small and small end formats. Older versions do not support byte swapping. Newer versions refer to "Mixing translation scaling parameters".
3. After the data source is set to floating-point, the floating-point format, ABCD and CDAB, is supported.
4. Double precision floating point translation scaling is not supported.

3.12. Mixing pan scaling parameters

Use XX07 firmware 1.511 and above; XX06 model firmware 1.480 and above; XX12 requires 1.480 and above, with Vircom6.71 and above version, can realize the mixed use of translation and scaling parameters.

The previous panning and zooming function had the following limitations:

1. If Pan Zoom is selected for one node, Pan zoom must be selected for all nodes.
2. The parameters of translation and scaling are: 2/4-byte selection, size end/size end exchange 4 combination selection, source data is integer or floating point selection. Previously, if different parameters exist, only the parameters of the first node are selected, and independent parameters of multiple nodes cannot be implemented.

Versions of mixed use parameters:

1. In order to be compatible, one node still needs to check Pan zoom, then all nodes need to check Pan zoom. You can set Pan to 0 and zoom to 0.99999 to set the actual no pan zoom.
2. The parameters of translation and scaling can be set independently.

Vircom6.71 and above versions do compatibility processing, if the translation and scaling parameters are the same, the old configuration file format is still used, so the old firmware can still be configured with the new vircom. If there are inconsistent panning and scaling parameters, the new format will be adopted, and the old device will not work properly, and the new firmware device will get all the class parameters.

There is no special place in the configuration, different nodes can choose different parameters, such as:

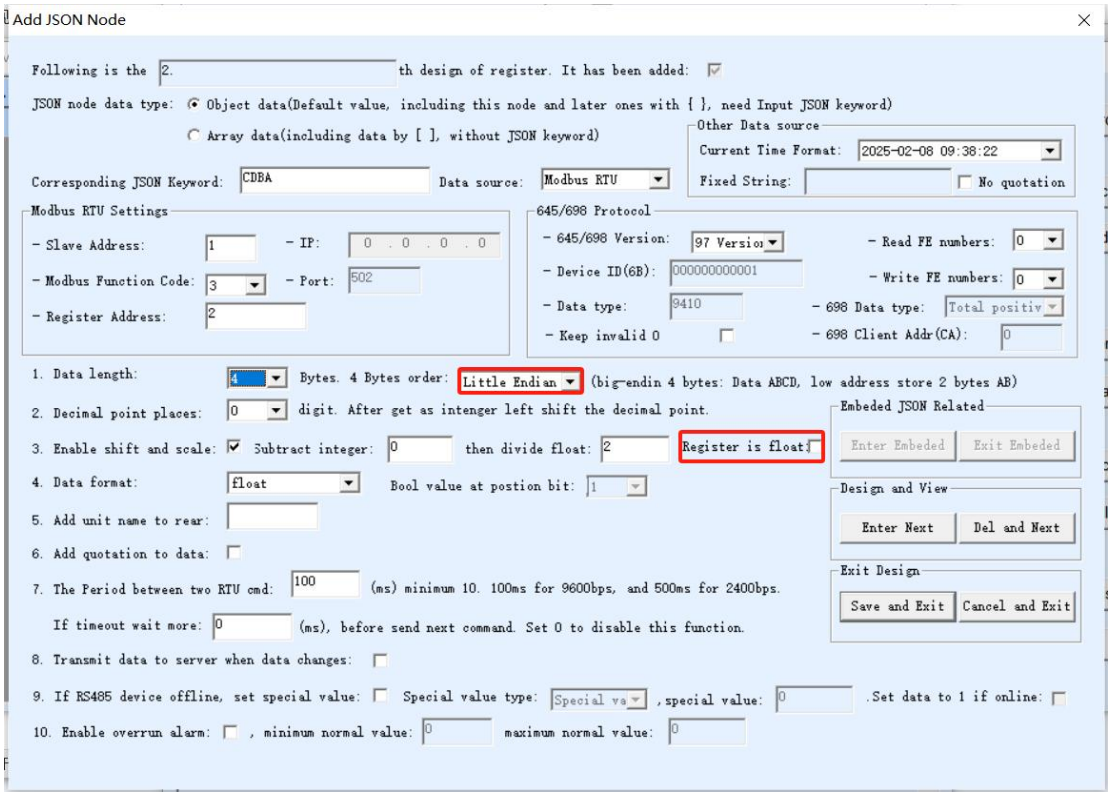


Figure 25 LONG integer configuration in small-endian format

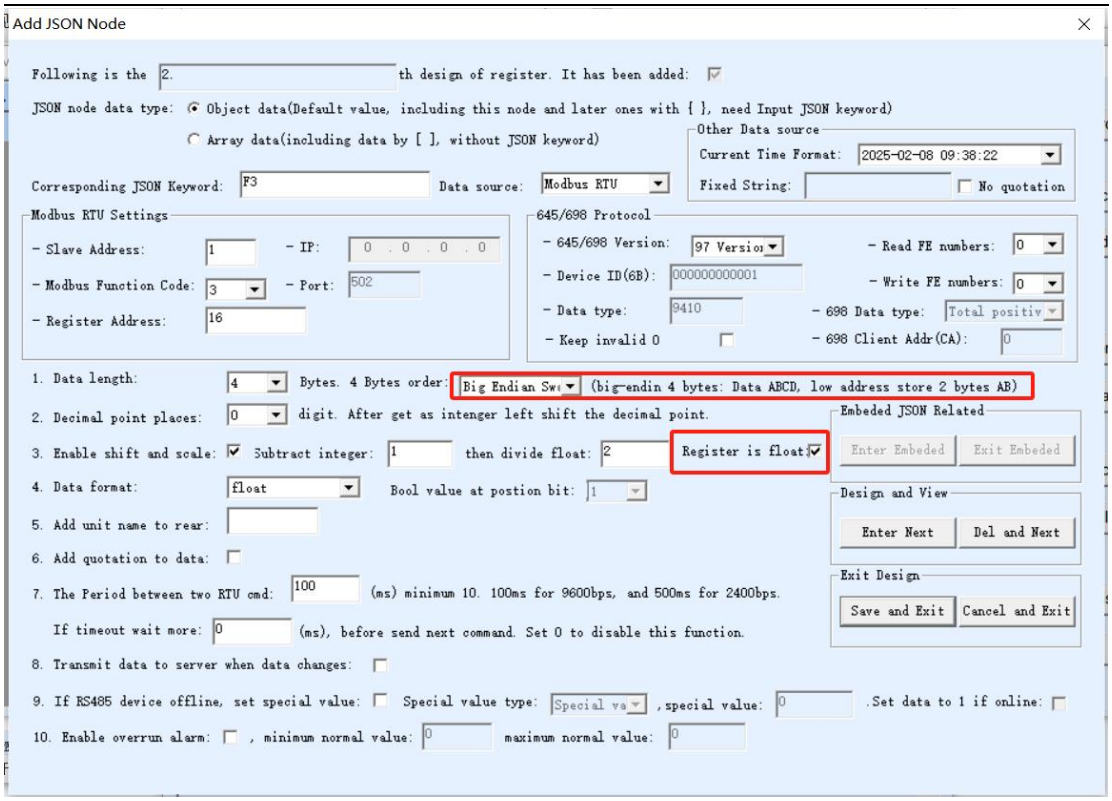


Figure 26 Big-end switch FLOAT configuration

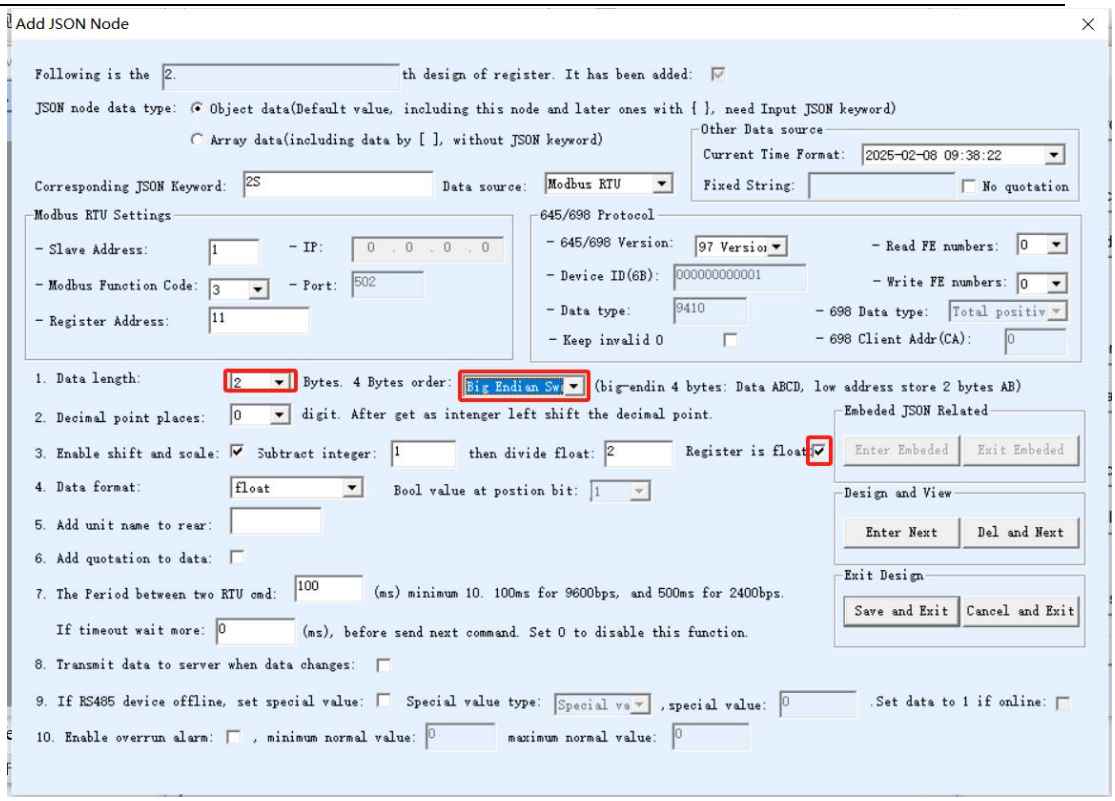


Figure 27 2-byte integer configuration for byte swapping

The three configurations in the preceding figure, byte length, 4-byte order, and floating-point data source, are different and can be configured independently. All translation scaling is the original data minus 1, divided by 2. If the original data is 123, the result is 61.

After the configuration is complete, download the configuration. The test result is as follows:

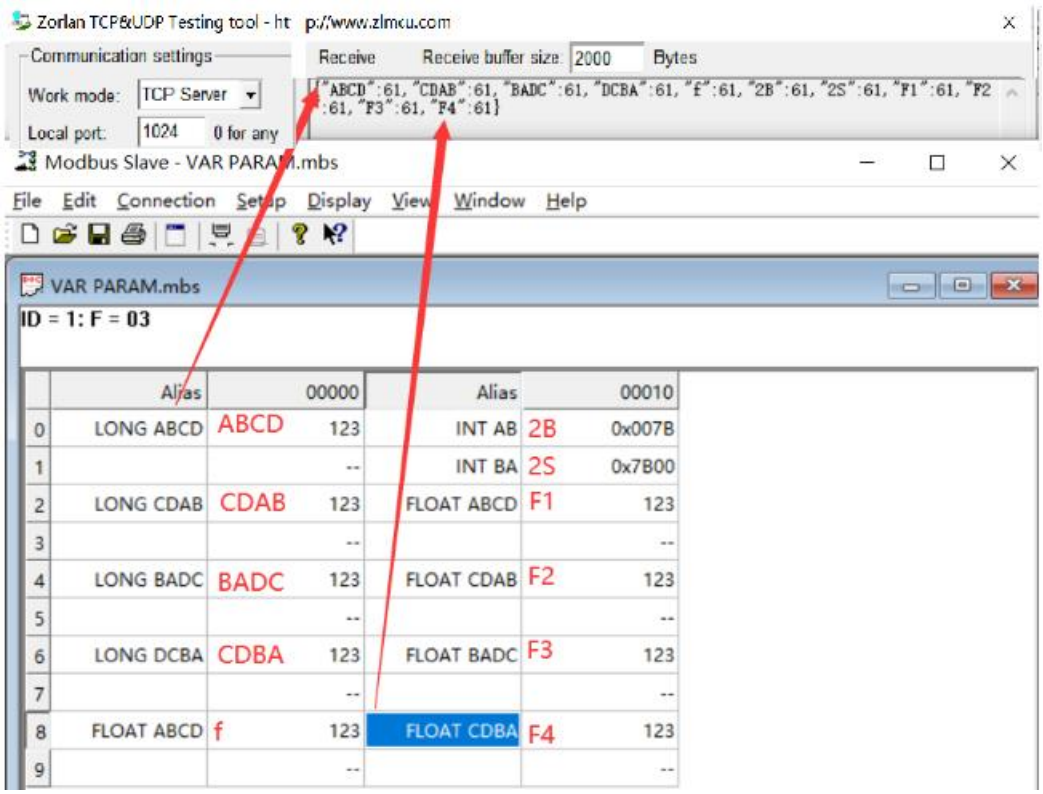


Figure 28 2-byte integer configuration for byte swapping

The top half of the graph is the data received by the network: ：{"ABCD":61,"CDAB":61,"BADC":61,"DCBA":61,"f":61,"2B":61,"2S":61,"F1":61,"F2":61,"F3":61,"F4":61}，There are 15 registers:

Json Node name	Number of bytes	Floating point type	Big and small end	Upload results
ABCD	4	N	ABCD	accord
CDAB	4	N	CDAB	accord
BADC	4	N	BADC	accord
CDBA	4	N	CDBA	accord
f	4	N	ABCD	accord
2B	2	N	AB	accord
2S	2	Y	BA	accord
F1	4	Y	ABCD	accord
F2	4	Y	CDAB	accord
F3	4	Y	BADC	accord

F4	4	Y	CDBA	accord

The Modbus Slave configuration results are displayed in the lower part of the page. Select the format and data type of the Modbus Slave for different configurations. The test shows that all 15 registers, configured with different parameter combinations, can each acquire the correct value.

The data source to be delivered can be floating-point and byte exchange. New vircom and firmware are also required to support it.

3.13. Report data changes

In some cases, in order to reduce data traffic, it is not necessary to upload data frequently. Just upload it when the collected data changes. At present, Vircom5.30 and above versions, combined with 1.589 firmware can achieve this function. In practice, the upload time period can be set to a large maximum of 8.8 hours. In terms of traffic, it is equivalent to not uploading data at this time. However, if data change upload is selected, every time the data of a node changes, all data will be uploaded.

The data changes here can be configured for each JSON node, and can be used to trigger an upload when the device is offline (when the data content changes, it can also be uploaded).

As a typical case, we collect DI status and upload it when it changes.

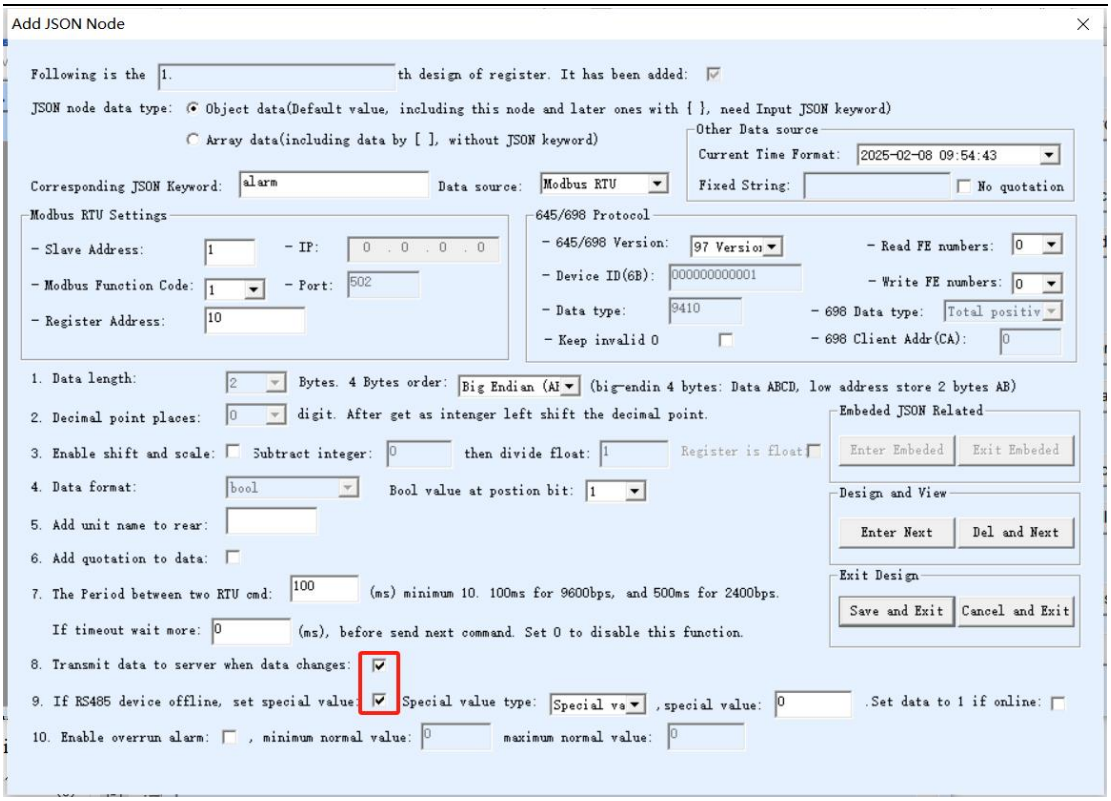


Figure 29 Data change upload

You need to select Data change to upload data and clear the RS485 device offline. If this parameter is selected, the device will report when it is offline. However, this offline report is limited to the case where the original data value is 1. To report any offline data, you need to add a separate JSON node. For details, see section "Device Offline".

Add another alarm2 node with IP address 11, but do not select upload data changes.

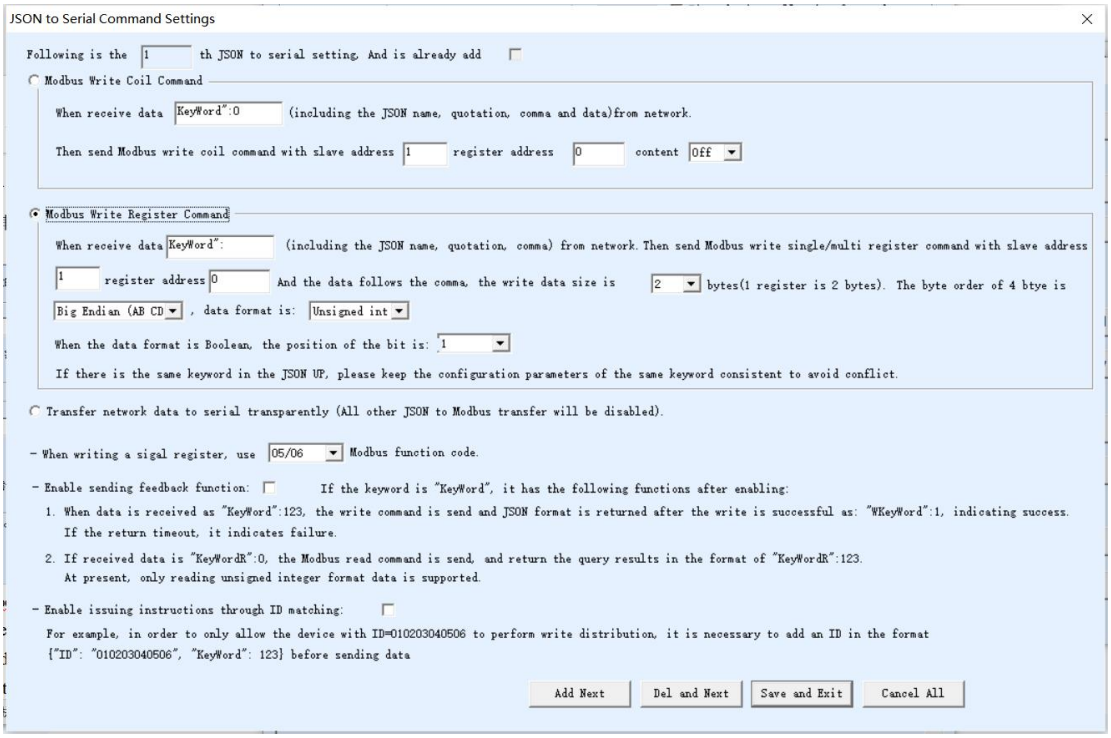
Normal data is uploaded every 10 seconds. If there is a bit change (alarm) at address 10, the report will be triggered immediately (actually due to the rotation of the data change takes several hundred milliseconds, the upload will be slightly delayed). However, if the bit whose address is 11 changes, the change reporting option is not selected.

When the alarm is 1, the device is also reported immediately if it is offline.

3.14. JSON issue

Introduction to use

JSON delivery is to achieve the matching of a single JSON keyword and the corresponding Modubs instruction output. This does not require that the sent strings match exactly, as long as there is a corresponding JSON keyword and related data in the sent data.



ModbusFigure 30JSON transfer to Modbus

Click the "JSON Release" button of "JSON to Modbus RTU Settings" to open the above dialog box. It is divided into three categories: Modbus set coil instruction, Modbus write register instruction, transparent transmission instruction.

Modbus set coil instruction: is 05 instruction, here you can specify the slave station address, register address and set to On and Off. If the data corresponds to the JSON of "alarm" : "1", you need to write the complete "alarm" : "1" to the box, including quotation marks and colons.

Modbus write register instruction: is 06 instruction, currently supports 2 to 8 bytes, that is, 1, 2, 4 register write. Data must be shaped (without decimal points).

Here, you can set the address of the delivery station, the address of the register, and the number of bytes. If it is 4 or 8 bytes, you can choose the large-endian format or small-endian format. If the corresponding method is temp: 1234, enter temp: in the input box. The characters before 1234, including quotation marks, must be entered.

Supports 8 bytes of double precision json delivery, including four small and small end formats.

In transparent transmission mode, the sent command is transparently transmitted to the serial port without parsing. Once the transparent transmission mode is selected, all JSON delivery parsing is disabled.

Multibyte delivery

JSON to Modbus RTU supports the 05/06/16 command. If you need to set more than one coil with the 15 instruction, use the 05 instruction several times.

Depending on the length of the number of bytes, the system will automatically select 06 or 16 instructions to send. Here is an example of setting the coil and setting the register.

If you receive JSON data from {alert: "on"}, you need to use the 05 command to set the coil at station address 02, starting with register 03. Click "JSON Delivery" on the JSON to Modbus screen.

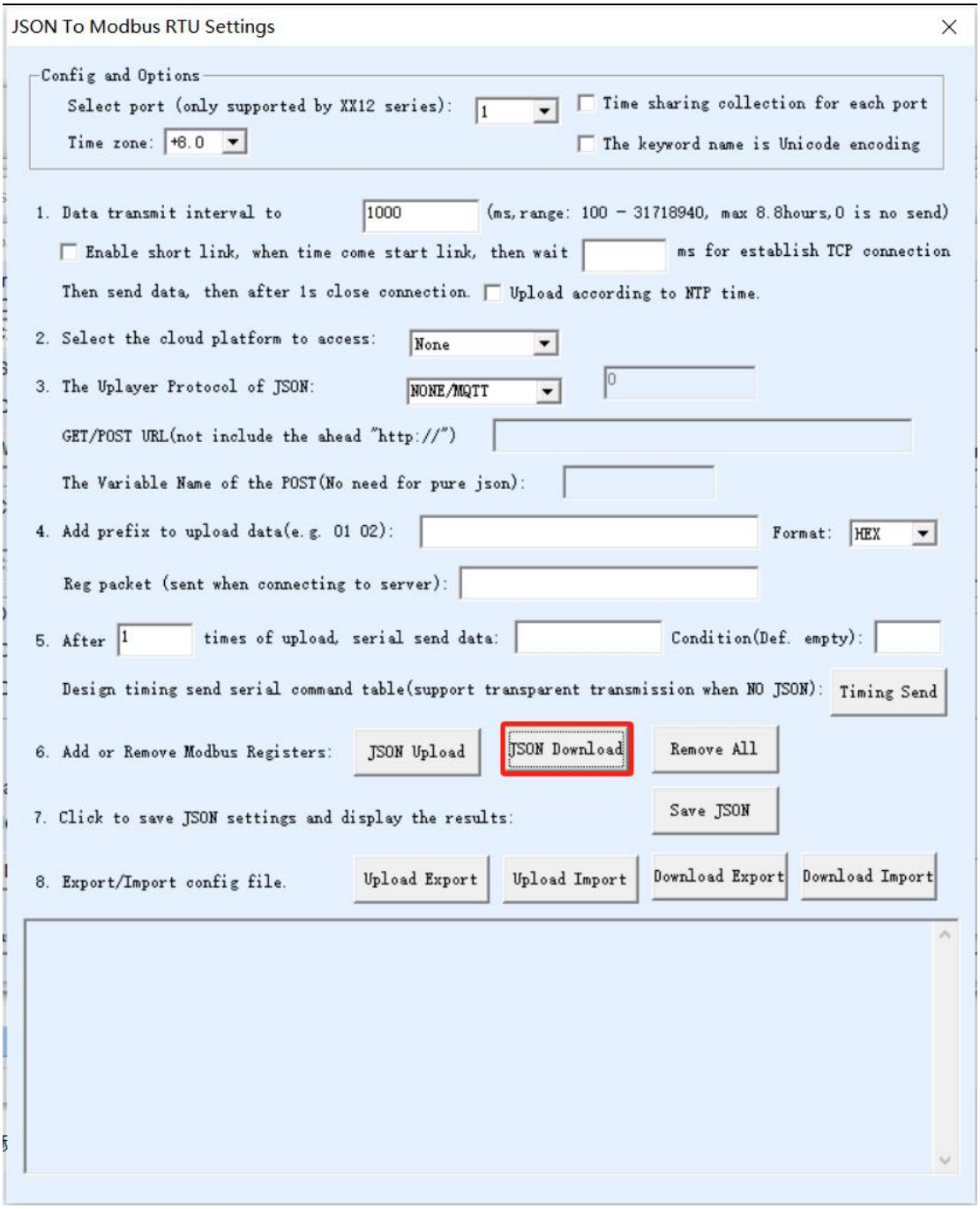


Figure 31 Entering JSON delivery

The configuration screen is as follows: Notice the alert: "on" set need to write part is OK.

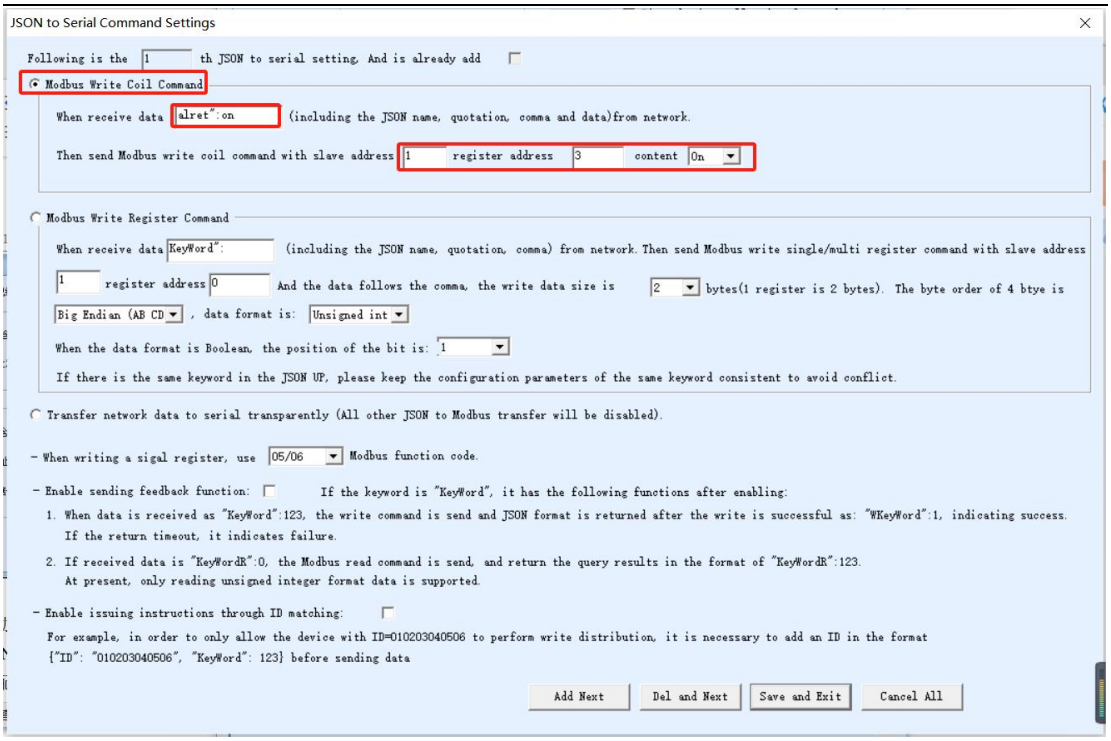


Figure 32 Configuring the coil

Click "Next" to add another delivery conversion, otherwise click "Save all and exit". Back on the main screen, click "Save JSON Settings" and then click "Return". Then click "Download" on the download screen. This completes the configuration.

If you are sending {power: "12345"} down, you need to set the value of power 12345 to the station address 2, register 3. The Settings are as follows:

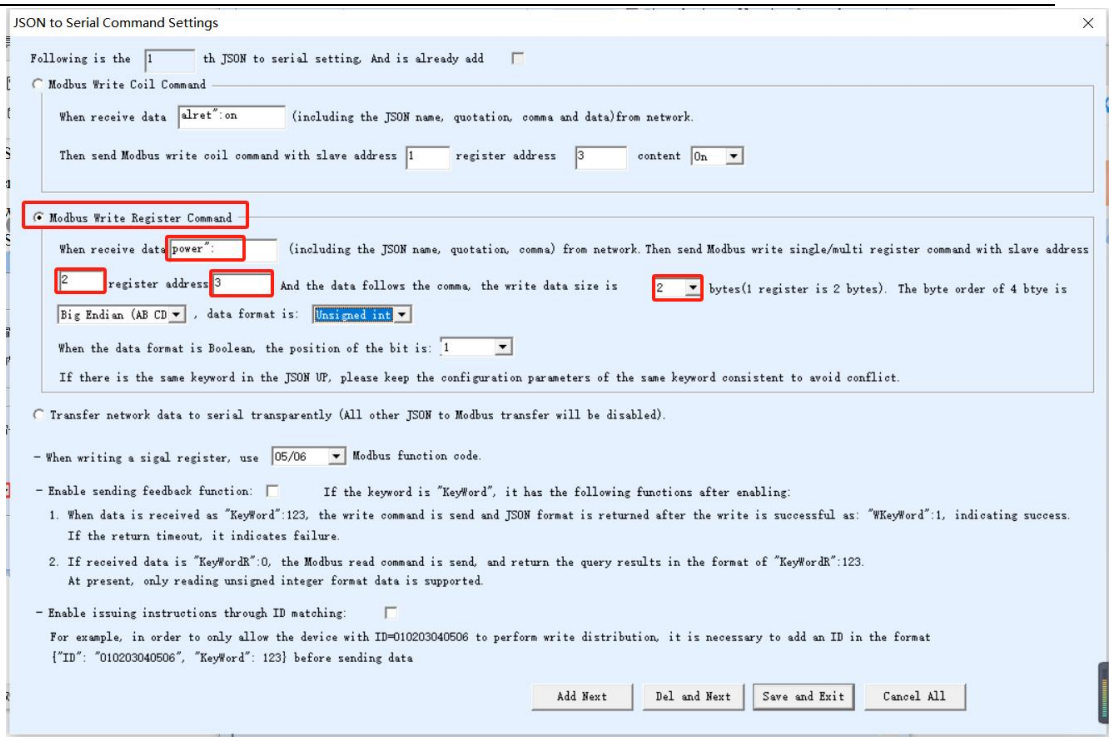


Figure 33JSON setup register

Note that the keyword here only needs to enter power: ", do not need to enter the following 12345, because this value is changing, but you need to enter a colon, if the quotes in the delivered data also need to enter quotes.

Use the 15/16 command

In some cases, the device does not support the 05/06 function code, so you need to use the 15/16 instruction function code even for a single byte. In this case, you need to modify the default delivery rules. The default delivery rule is: use the 05/06 function code for a single coil/register setting, otherwise use the 15/16 function code.

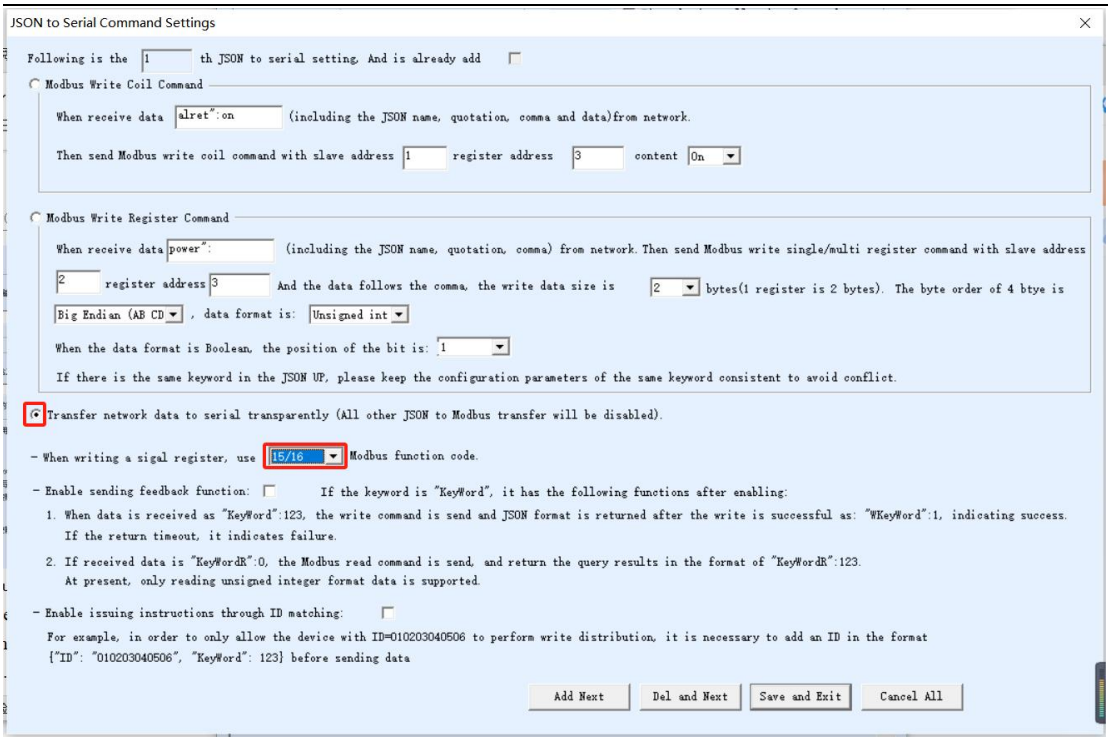


Figure 34 Using the 15/16 instruction

Vircom6.73 and above support manual selection of 15/16 function code, as shown in the figure above. This option is valid only for one node, not for all nodes. It works on coils and registers. After the 15/16 function code is selected, the 15/16 function code is also used if it is a single coil/register setting.

The sending and sending matches

JSON sends more parameters to set, but sends less. Sometimes the sent and delivered JSON keywords are the same. In this case, if the following circumstances, the delivery can use some parameters sent.

1. When Mdobus TCP exists as the data source: IP and port will use the parameters sent above.
2. When there is translation scaling: translation value, scaling value, data length, size end, source data is floating point, the parameters sent above will be used. Note that the register position still needs to be set correctly, but the data type sent can be either integer or floating point.
3. 03 Function code bool Type: The function code is set to 03, the data type is set to

bool, and the position of the bool value is the parameter set in the json file.

4. Floating point data (excluding translation and scaling) : The data type is floating point, the data length is set to json upload, and the size end is set to json upload.

03 Function code BOOL Type delivered

If 03 function code BOOL keyword is sent, for example, {" a ":1}. Then, when the name "a" is set to the same, the location of the BOOL bit can be identified. If the upper transmission does not exist and there is a separate transmission, the previous Vircom does not have a parameter indicating the position of the BOOL bit.

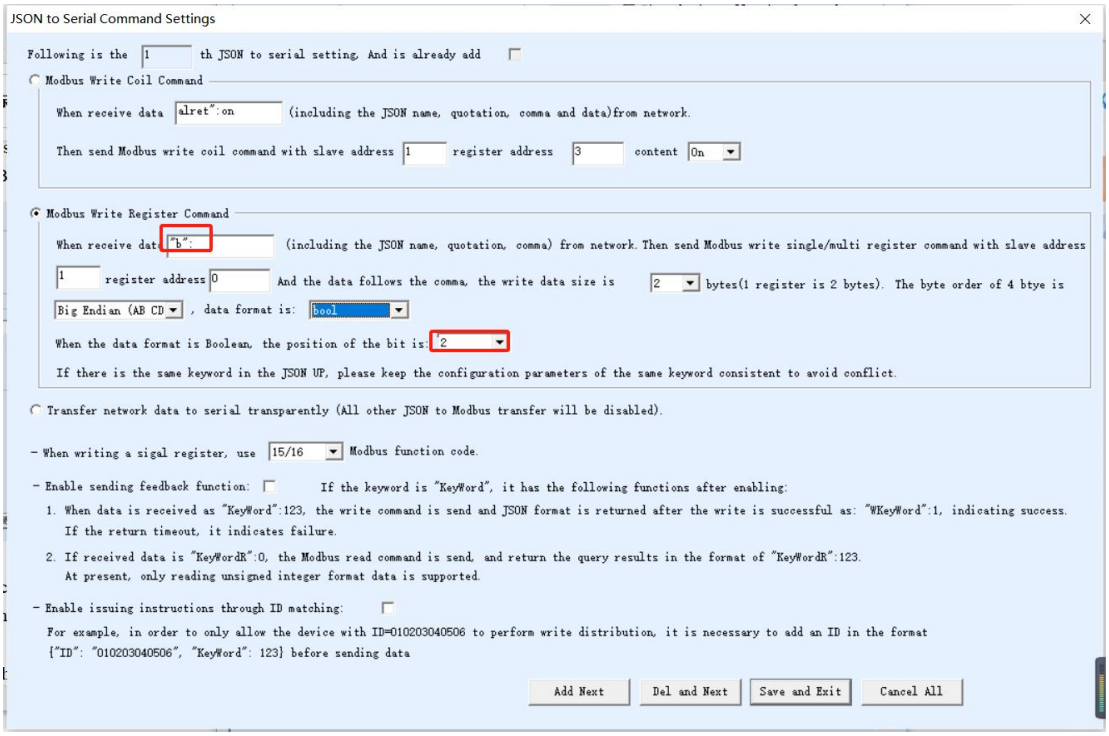


Figure 34 Position Settings for delivering a BOOL bit

Vircom6.73 and above support, issue a separate set of bit location, even if it is a keyword that does not exist on the issue. Notice The data format must be set to Boolean to select the bit where the Boolean value resides.

After setting, send "b" :1, the setting will change the second bit, that is, the register value is changed to 0x0002, instead of the previous non-bool type 0x0001.

3.15. Batch JSON Delivery

When JSON delivers the setting register, multiple variables may be delivered at the same time, for example, {" a1 ":1," a2 ":2," a3 ":3}. Previous versions also support this delivery, but the sequence of a1, a2, and a3 must be consistent with the configuration sequence, and there must be no gaps between them. However, the actual delivery may have a few fewer variables, or the order is different.

Upgrading to a later version after 2023/11/12 can resolve this issue. You can then send any combination of JSON variables that can be set at once.

3.16. 645 Protocol Timing

The timing of the 645 protocol is 68 99 99 99 99 99 99 99 99 68 08 06 SS MM HH DD MM YY CS 16, where SS MM HH DD MM YY are second, minute, hour, day, month and year respectively.

With Vircom5.32, the timing configuration can be realized. Copy the following string "68 99 99 99 99 99 99 99 99 68 08 06 TIME[25...30] CRC[3] 16" to the serial port output command at the same time. Copy "TIME[0]" to the output condition.

Note that at least one upload JSON data must be set for timing delivery. In addition, the TCP connection must be established before the time can be delivered, because the time is sent on the network, if the TCP is not established, it will not be sent, and there is no time.

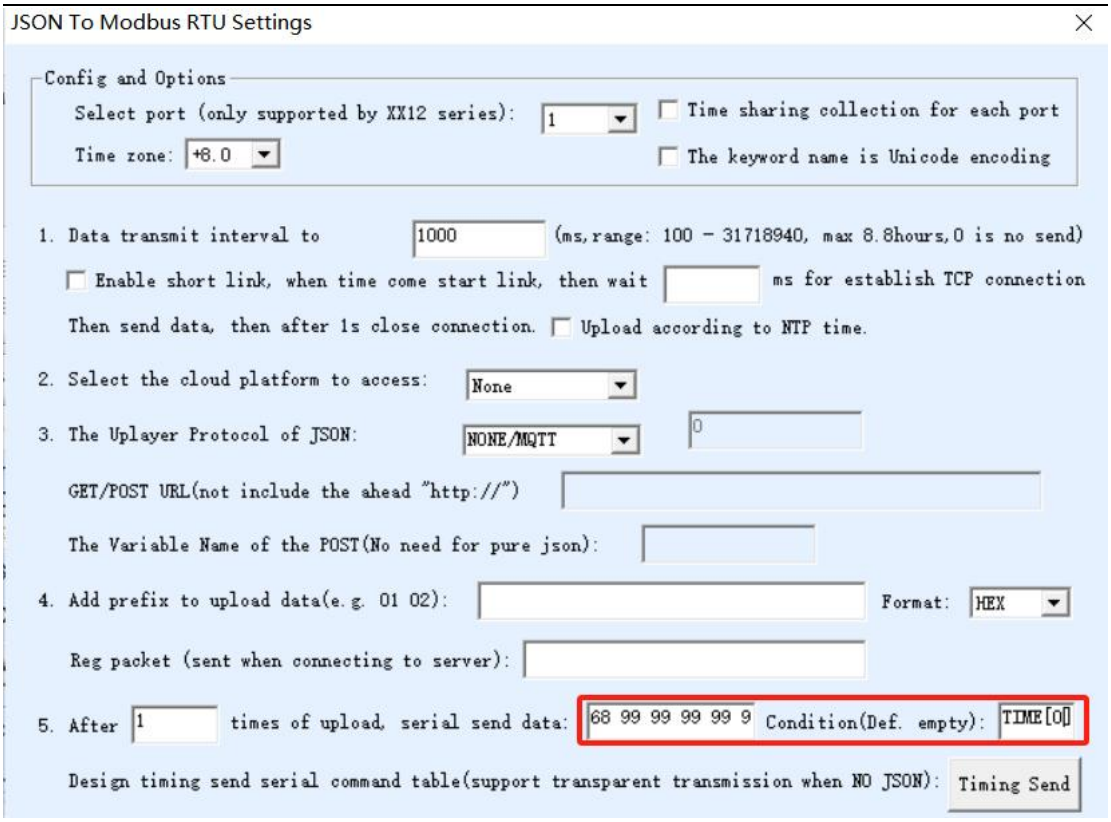


Figure 35 645 timing configuration

"TIME[0]" here is optional, if filled, the device will be sent after the valid time. If it is not filled in, the timing will be sent in any case, and problems will arise if the timing is incorrect.

3.17. JSON Sent back

3.18. Sending data on JSON in UDP mode

3.19. Out-of-limit alarm

The over-limit alarm function provides the function of reporting data immediately when the data collected by Modbus exceeds a certain range. Localization of data computation and processing. Assume that the collected data is c after being converted to the desired value, the minimum normal value of c is a, and the maximum normal value of c is b. When a<=c<=b is normal, otherwise it is abnormal.

When the device finds that the collected data changes from the normal state to

the abnormal state or recovers from the abnormal state to the normal state, it immediately sends a JSON packet to the server.

The speed at which the system detects the normal state or the abnormal state is related to the polling time, which can be set through the "Serial port polling Interval" in JSON to Modbus, and also depends on the number of Modbus collection points set. The maximum time generally does not exceed the number of collection points multiplied by the polling time +1 second.

At present, the overlimit alarm must be used together with the pantograph function, because the general upper/lower limit is a floating point, and the analog value collected is generally an integer, and the collected data needs to be converted to floating point for comparison. Translation scaling is a good way to convert to floating point data. Panning and scaling supports data sources in 2-byte, 4-byte, and sider-end formats, as well as single-precision floating-point data sources.

Since panning and scaling are required to be enabled for all acquired data, panning and scaling are required for all data when over-limit alarm is used. For data that does not require panning and scaling, when panning and scaling is enabled, keep panning and scaling at 0 and default 0.99999, so that the data will not be affected.

Here is a concrete example of how to use it.

Assume a 4~20mA instrument, when 4mA corresponds to the flow of 0, when 20mA corresponds to the flow of 100. Let the current be I and the flow rate be F, and the formula of current conversion flow is F=(I-4)/(20-4)*(100-0)+0= 6.25* i-25.

Use a device to collect analog signals.. If the value read by the register is R, the real current value is calculated by the formula I=R/1024*25(mA). Enter the above formula to get:

F I - 25 = = 6.25 * 6.25 * (R) / 1024 x 25-25 = (R - 163.84) / 6.5536

Translation can only be integers, so take 164, and scaling can only keep 6 decimal places, so 6.5536. To wit:

F= (R-164)/6.5536

If we're in the normal range of 12.1 to 80.123. You need to change 80.123 to 80.12, because after removing the decimal point, the value of the method cannot be

greater than 65535 (80123 is greater than 65535), and the negative upper and lower limits are not supported. Assuming the analog signal of the first channel of this device is collected, the function code is 4 and the register address is 0. The settings are as follows:

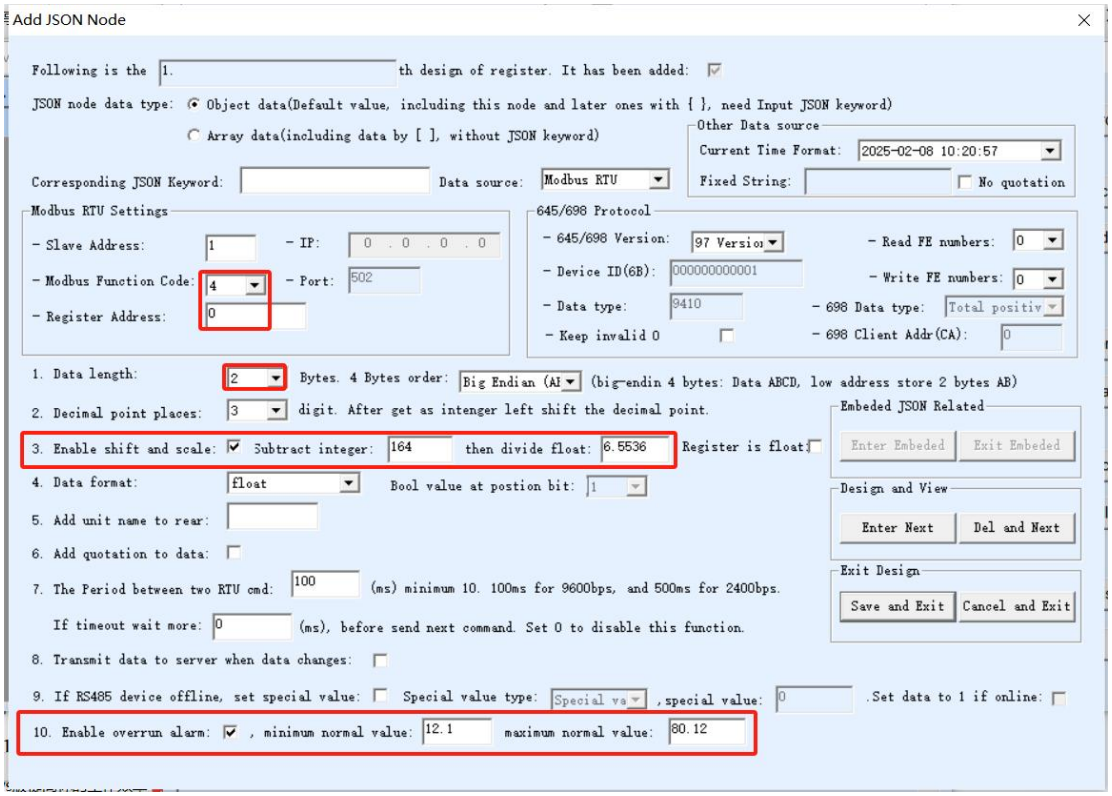


Figure 36 Over-limit alarm

By backward inference, register R=243 or so is the lower limit, R=689 is the upper limit. When Reg=243 and 244 changes, {"Flow":11.902} or {"Flow":12.207} is sent immediately, indicating that the lower limit and recovery are reported. When R=690 and 689 changes, {"Flow":80.261} or {"Flow":80.109} is sent immediately, indicating that the upper limit and recovery are reported.

Different translation scaling and upper and lower limits can be set for multiple registers.

3.20. Data is reported for the TCP short connection

This function can automatically switch from the TCP Server mode to the TCP

Client mode when data packets need to be sent. The destination IP address and port have been preset in the parameters. Wait for T milliseconds, wait for the connection to be established, start sending, send 1 second after the completion of the TCP Server mode again. Here T can be set, the default is 1000.

As shown in the following figure, the device parameters need to be modified in advance for short connections: Switch to the TCP client mode first, set the destination IP address (192.168.0.101) and port (1024), and then switch to the TCP server mode and save the Settings.

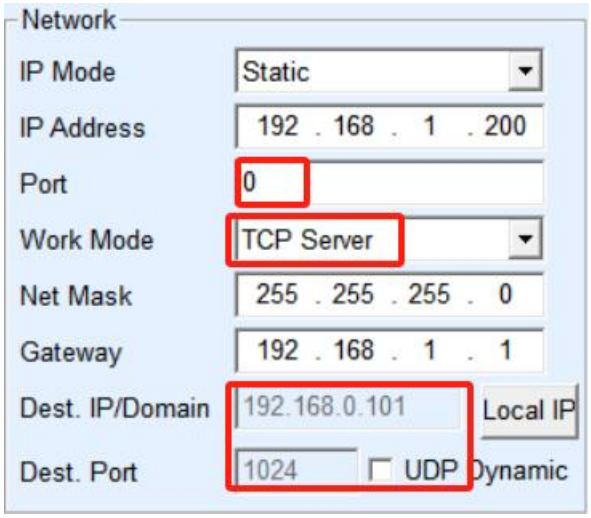


Figure 37 Setting parameters for short connections

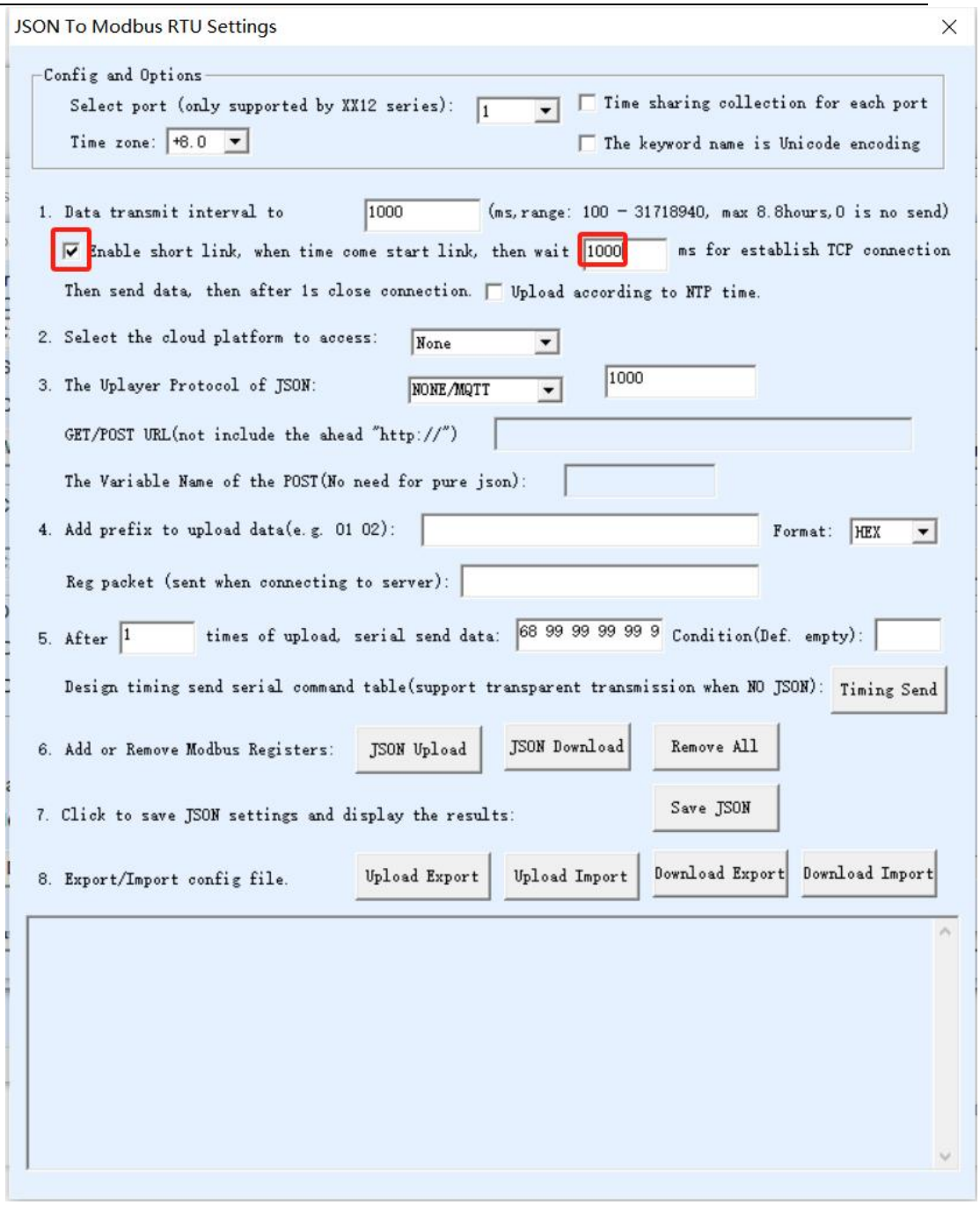


Figure 38 Enabling short connections

Go to the JSON to Modbus RTU Settings dialog box, select the Short connection mode, and set the time for waiting for the server to connect, generally 1000ms. If the connection cannot be established within 1000ms, the sent data will be lost. Therefore, this time needs to set a time to ensure that the connection is established.

However, if the setting is too long, the data reporting time may be delayed.

The other Settings are the same as those used for long connections. If the NTP time is used to report the accurate time, select Report by NTP time. Data change reporting can be enabled simultaneously; The out-of-limit alarm function can be used simultaneously.

JSON TCP short connection report description:

1. TCP short connection can be used at the same time with MQTT protocol, and this function can be used to solve the case that the device will be disconnected from the MQTT server for a long time.
2. Sending server time: indicates the interval between closing the connection and re-establishing the connection. The connection time is independent of the Disconnection and Reconnection time set on the device home screen.
3. The time of "How many ms to wait for the connection to be established before sending data" in the figure above: for MQTT, the connection generally takes 3 seconds, and it can be set to 3000.
4. For JSON data content changes are reported and short connections can be used together. When the data changes, the device tries to establish a connection and send data. Data can coexist with data sent on the cycle.

3.21. Modbus TCP TO JSON

Suppose we need to collect the Modbus TCP slave device at port 502 of 192.168.0.102 and port 502 of 192.168.0.103, and collect the data of the Modbus RTU device at the same time. The data is sent to port 1024 at 192.168.0.102.

Settings The Settings are as follows: Set conversion protocol to None.

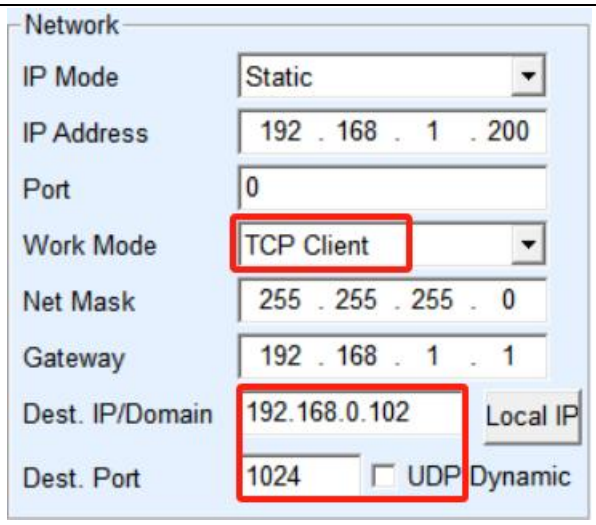


Figure 39 Setting up the primary socket connection

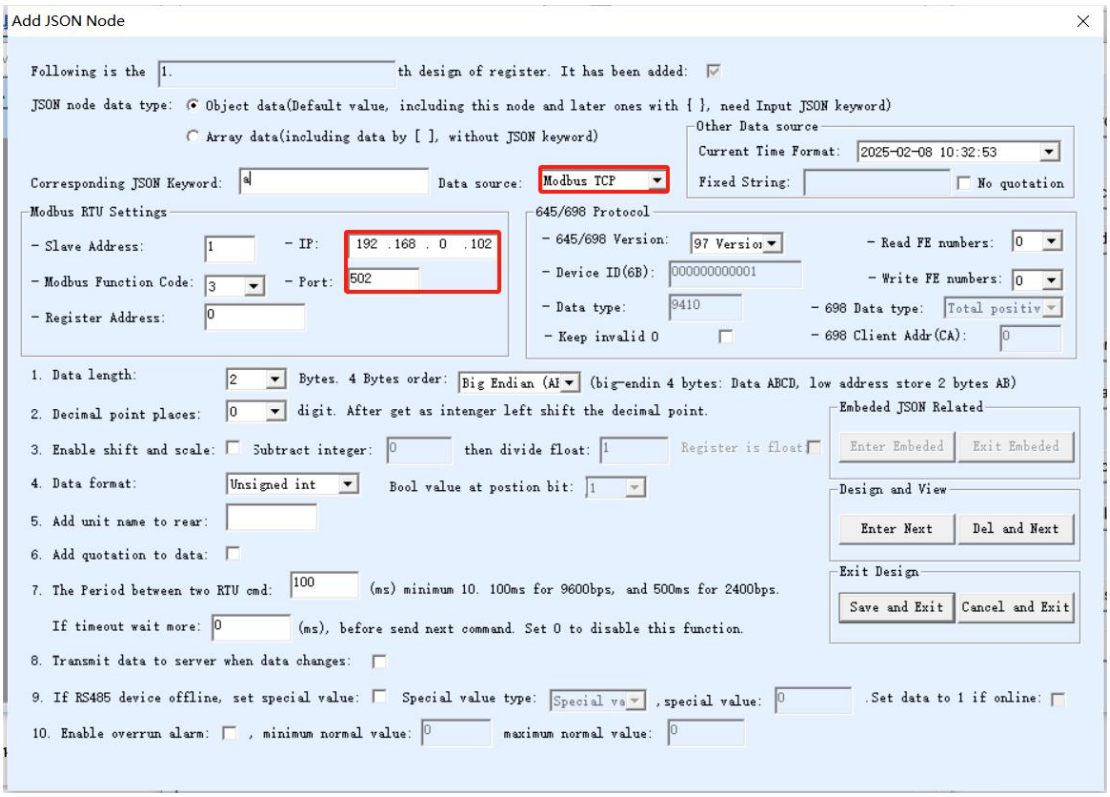


Figure 40 Setting up the primary socket connection

To use Modbus TCP, select Modbus TCP as the data source and enter the IP address and port number. You can set a maximum of six IP addresses and ports.

Can be mixed with Modbus RTU equipment.

Other uses are compatible with Modbus RTU.

Short connection Modbus TCP

If Short connection mode is selected in JSON to Modbus RTU, the collection may fail if Modbus TCP is used.

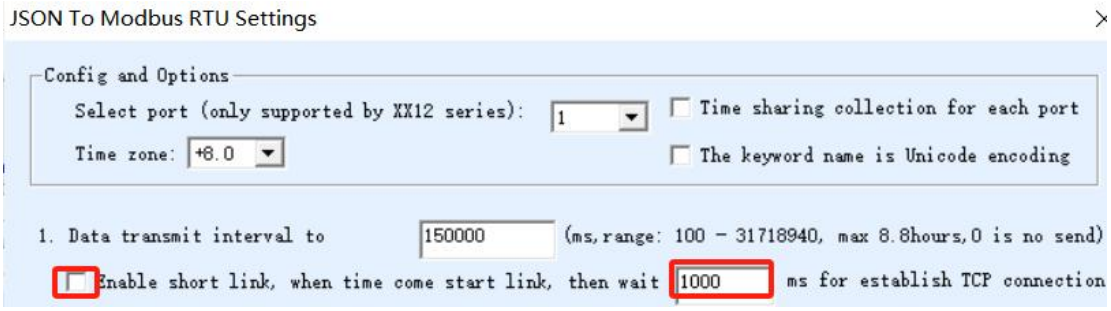


Figure 41 Short connection startup

The reason is that Modbus TCP data can be collected only when the device is in TCP Client mode. But a short connection leaves the device in TCP Server mode most of the time and in TCP Client mode for just over a second. Therefore, you need to change the disconnection time to 0. Otherwise, the data cannot be collected.

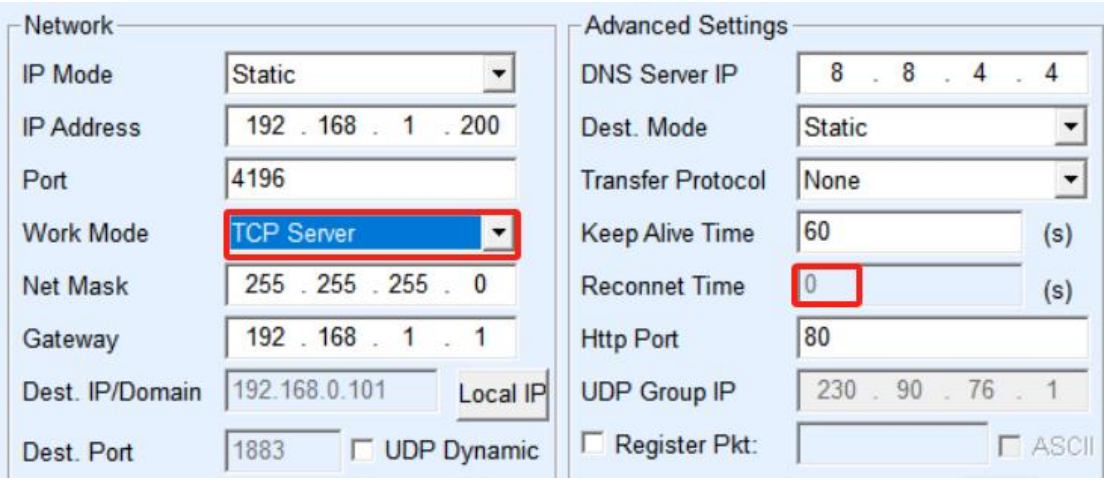


Figure 42 The disconnected line is reconnected to 0

However, "Short connection" + "Modbus TCP" cannot be used with "out-of-limit alarm" :

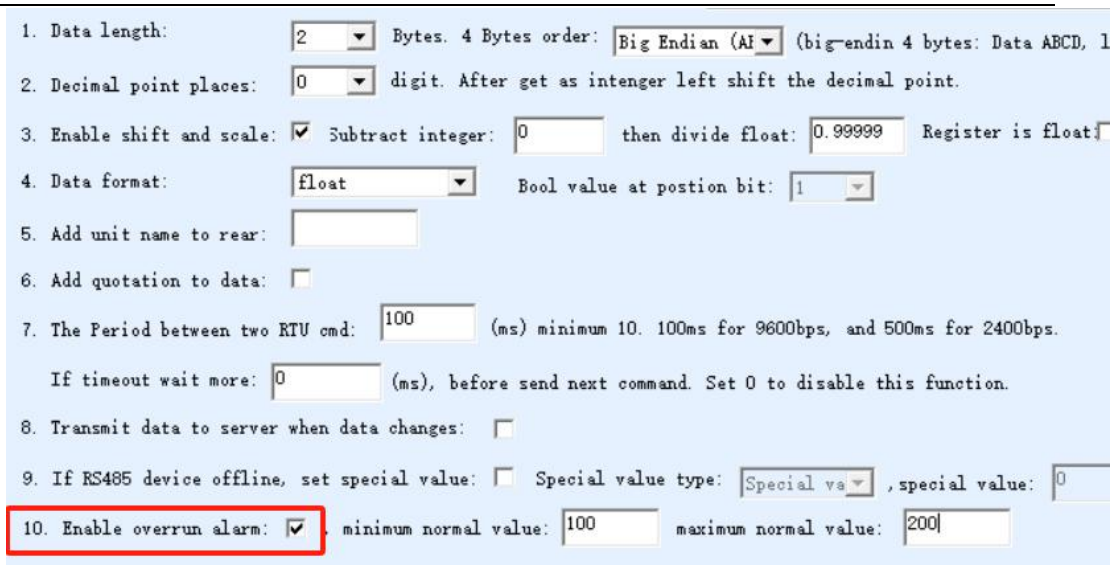


Figure 43 Over-limit alarm

Because the over-limit alarm requires the device to continuously collect the register data of Modbus TCP, it needs to send JSON data if it is found to exceed the range. However, since the device is in Modbus TCP mode most of the time, it cannot collect data, so it is not known whether the data has changed. When the TCP Client mode is changed to collect data, it is also the time to send JSON data, and it is meaningless to send JSON alarm data in advance.

3.22. JSON TO Modbus TCP

Set coil

If the sending Settings of Modbus TCP to JSON are as follows:

keyword	IP	port	Slave address	Function code	address
a	192.168.0.100	502	1	1	0
b	192.168.0.100	503	1	1	1

If the delivery is from JSON to Modbus RTU, you can set any delivery keywords such as aoff, aon, boff, and bon to indicate the delivery control, as shown in the following figure. Because the address of the slave station, the address of the register, and the content of the off/on have fully expressed the information of the instruction.

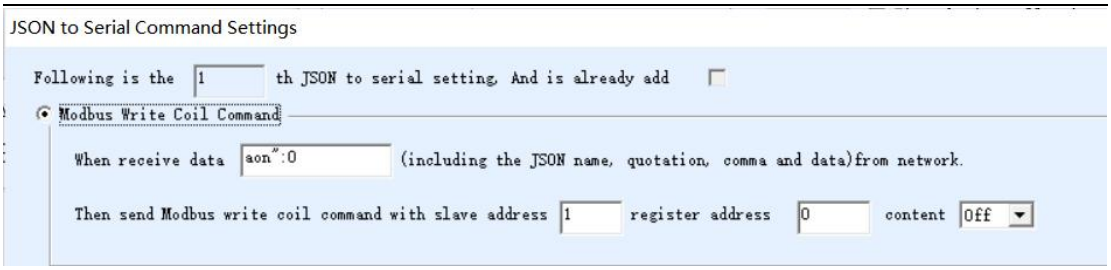


Figure 44 Coil control of common JSON to Modbus RTU

However, when using JSON to Modbus TCP, you also need IP and port information. In this case, you must use the same name as the previous one to find the IP address and port after matching.

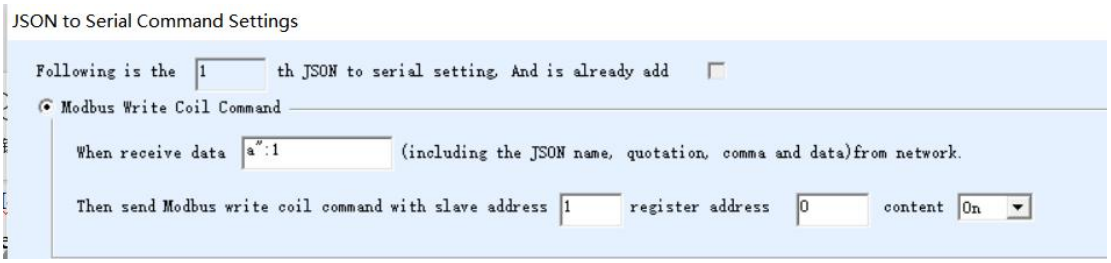


Figure 45 Coil On control of JSON to Modbus TCP

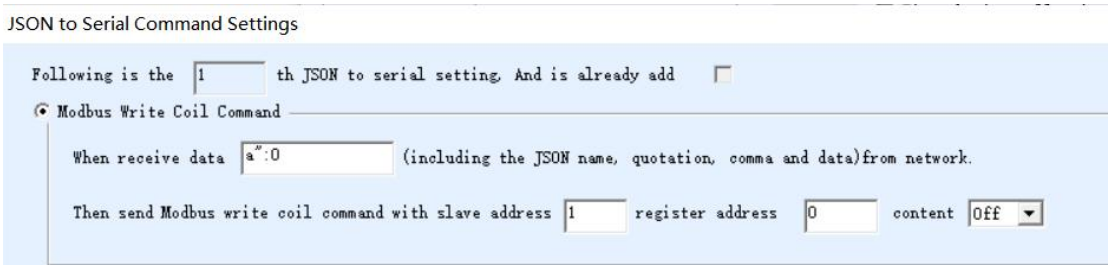


Figure 46 Coil Off control of JSON to Modbus TCP

In the figure, a ":1" and a ":0 are used to indicate the valid part of the delivered data. The actual delivered data can be {" a" :1}. Note that the keyword name must be "a", the same as the preceding text. In addition, the register position and Off/On also need to be set correctly in the above dialog box.

JSON server issued by {" a ": 0}, {" a" : 1}, {" b ": 0}, {" b" : 1} can set the following corresponding coil of the Modbus TCP.

Set register

If the sending Settings of Modbus TCP to JSON are as follows:

keyword	IP	port	Slave address	Function code	address

aReg	192.168.0.100	502	1	3	0
bReg	192.168.0.100	503	1	3	1

Do not choose a Boolean data format, generally choose unsigned integers.

Similar to the setting of the collar, the setting register also needs to issue the JSON keyword name and the above issue match, here set to bReg ":. However, the following 0 and 1 do not need to be written, and the device will set the value according to the actual received shaping data.

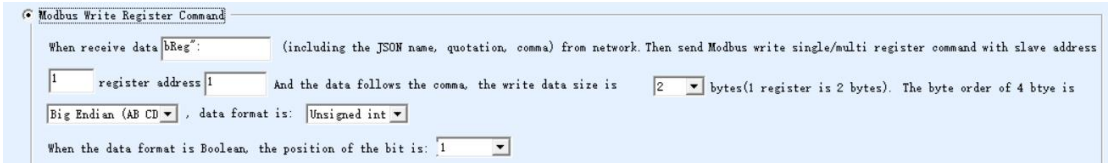


Figure 47 Register Settings of JSON to Modbus TCP

Note that the register address must be the same as the one sent above, set it to 1 here. If it is different, the one set here is the main one. The upsending and sending are not the same register.

Floating point data

When the Modbus floating point type is used: The data type is set to floating point, the data length is set to 4, and the decimal number is set based on actual requirements.

In this case, the length of the delivered byte must also be set to 4 bytes. Issue such as {" bReg ":123.123} to set the floating-point register.

Send back

JSON is delivered to Modbus TCP. Currently, the function of reading register contents by JSON and delivering return by JSON is not supported.

3.23. 645 Protocol symbol

The symbols of the returned data of the 645 protocol are handled in three ways:

1. When selecting an unsigned integer: all data is considered unsigned, for example 8012 will be considered 0x8012.

2. When selecting signed shaping: the highest digit of 1 is considered as a negative symbol, for example, 80, 12 will be considered as -12.
3. When floating-point type is selected: the highest bit of 1 is considered a negative symbol, but the symbol is not displayed. For example, 80 12 is considered -12, but the display is directly 12, that is, only the absolute value is displayed, and the symbol is not displayed.

3.24. Sending hexadecimal data

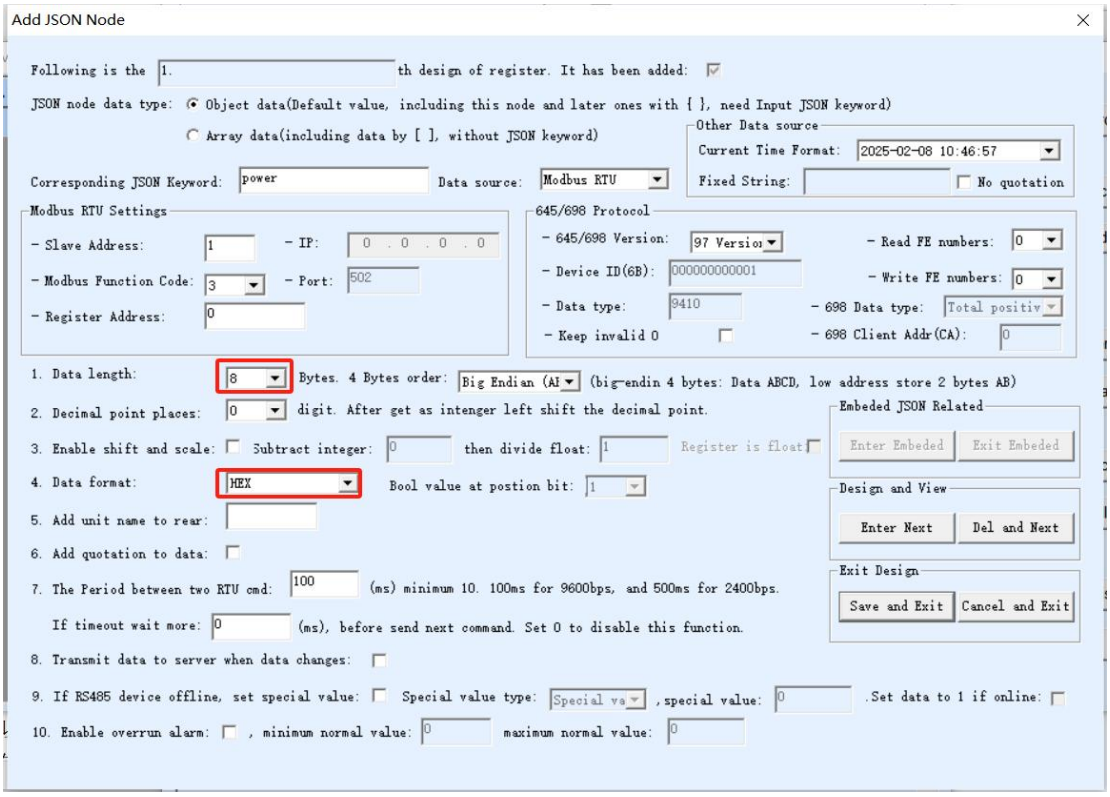


Figure 48 Hexadecimal reporting Settings

If the data format is set to Hexadecimal, the data format can be reported as a hexadecimal string. In this case, the data length can be a value ranging from 2 to 8.

For example, the content of register 0 is 0x1234, register 1 is 0x5678, register 2 is 0x90AB, and register 3 is 0xCDEF.

The following JSON data is sent: {power=0x1234567890ABCDEF}

3.25. Time Zone Settings

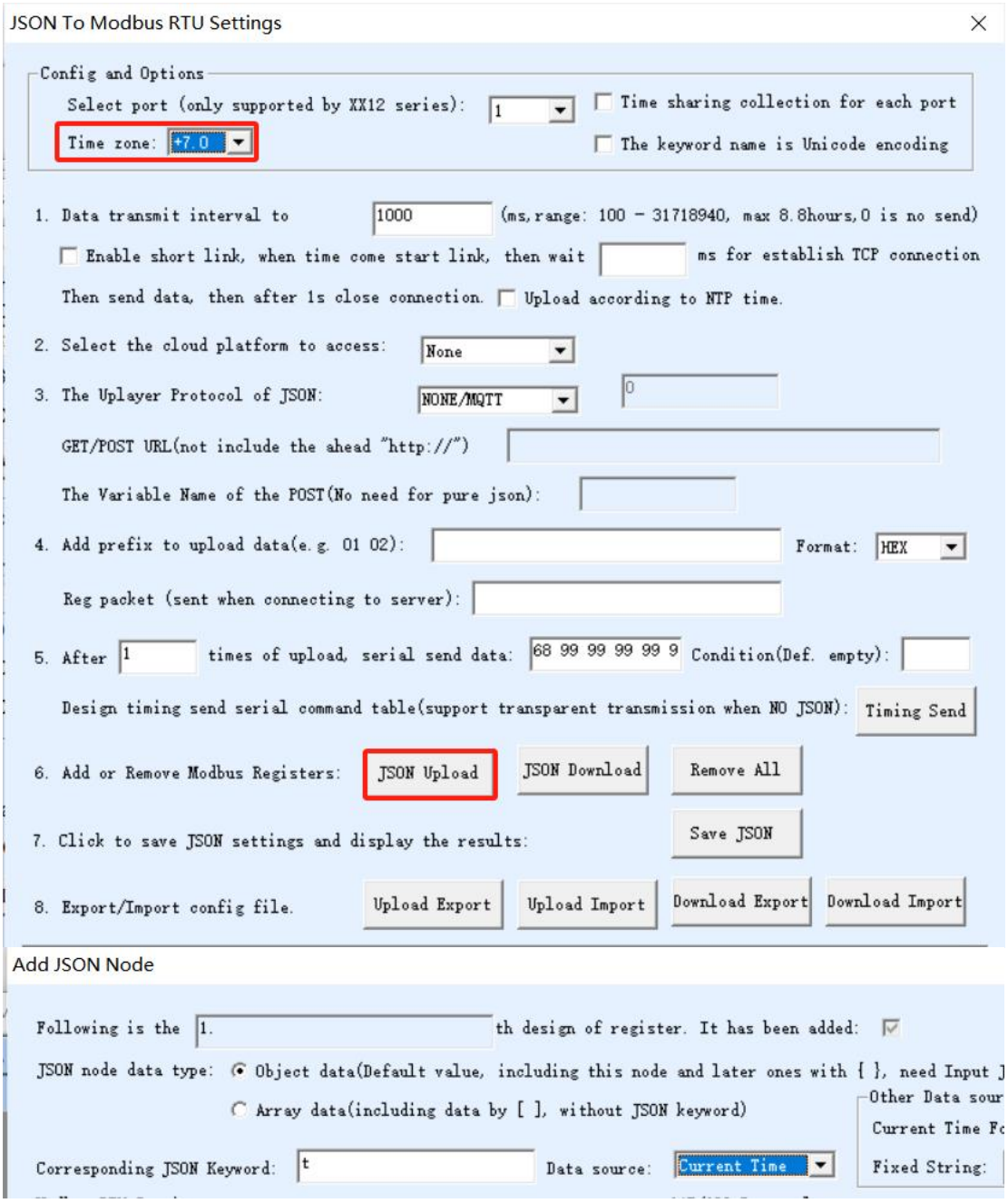


Figure 49 Hexadecimal reporting Settings

Set a sending time t on the JSON, and then select 7 time zone, then the sending time is Beijing time minus 1 hour.

There are 48 zones in total, each zone is 0.5 hour apart. Beijing is East 8 District.

3.26. Number of FE received by DTL-645

When the number of FE received by DTL-645 is certain, you can select it according to the following figure. If you are not sure, you can select all FE. However, selecting "all" will slow down the comparison, so try to choose a definite value.

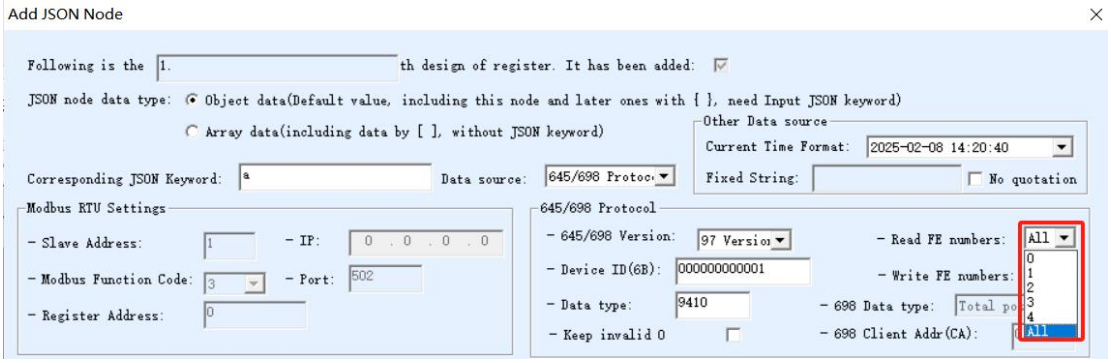


Figure 50 Hexadecimal reporting Settings

3.27. Periodic command delivery

For details, see Timing Command Delivery.

3.28. JSON and registration package

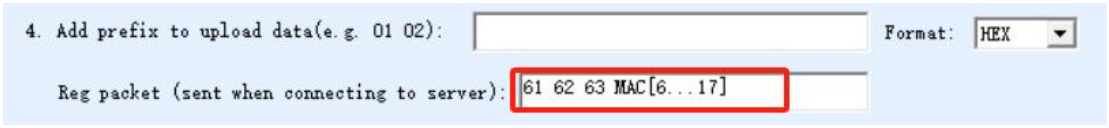


Figure 51 Setting up the registration package in the JSON dialog

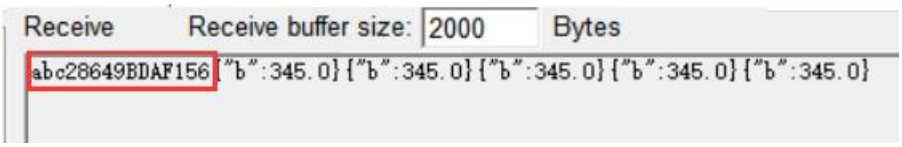


Figure 52 The server receives the registration package

In addition to sending on JSON, it also supports TCP connection establishment and sending registration packets. The default registration package format is hexadecimal, such as 61, 62, 63 with Spaces. You can also add MAC addresses in MAC[6... 17] format. If it is a character, select Ascii mode on the right.

3.29. 698 Converting the protocol to JSON

See "645 meters in json format to send method"

3.30. The device sends NULL messages offline

For Mobus to JSON, if the data bit type is integer or floating point, NULL can be sent instead of 0 when the device is offline. Because sometimes there are zeros in the actual data, it is impossible to distinguish between zero data and the device offline. Currently, the Boolean type, DLT-645, and DLT-698 do not support this function.

vircom6.38 or later is required to support this feature. New firmware support is required. For example, 5144J uses firmware 1.526(7044).bin or later.

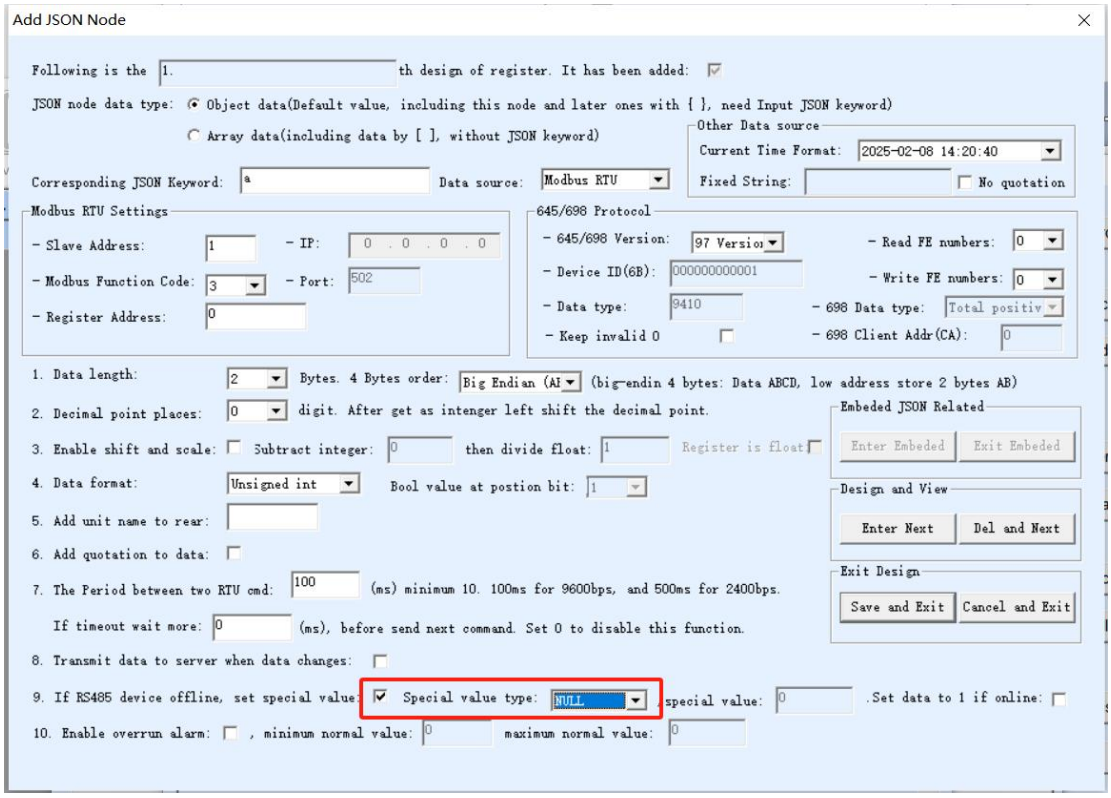


Figure 53 Setting for sending NULL offline

As shown in the figure, when you need to send NULL offline, first check the "RS458 device offline set special value" function, and then select NULL in the special value list, the default is 0.

Figure 54 Effect of sending NULL offline

The figure above shows the effect of the device sending NULL offline.

In the exported CSV file, if bit5 of the value of "Function Options" is 1, it means that the function of sending NULL offline is included. As shown in the picture below:

	Q	R	S	T
	Serial port polling	645 type	Functional option	indicates the number of JSON
	100	97	33	0

Figure 55 Setting NULL to be sent offline in the configuration file

33 of them have bit5 of 1.

Note:

1. If the data is an integer and Keep decimal place is not 0, the data sent is NULL. A decimal point is added to the end. In other cases and in floating-point mode, the upper transmission is directly NULL.
2. Supports various formats.
3. Support the combined query of continuous registers.

3.31. Upload only when changing

If Upload when data changes is selected, upload when data changes is enabled. However, periodic reporting needs to be canceled. Select 0 in the periodic reporting time (vircom6.53 or later is required). This allows uploading only when the data changes. Can be used to indicate whether the delivered Settings are set successfully.

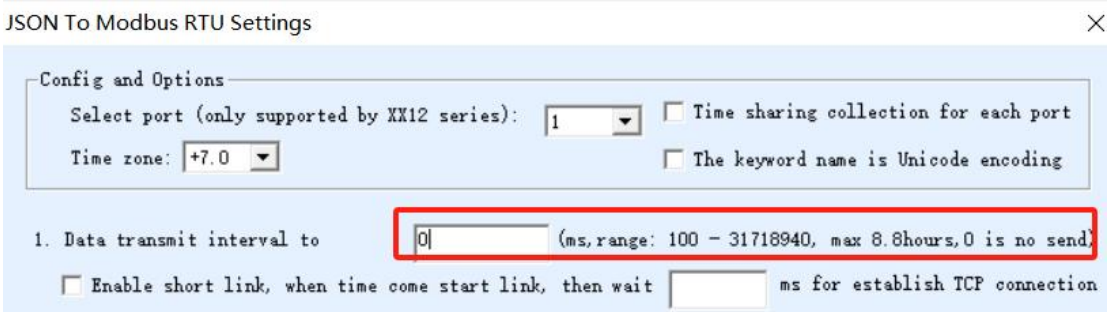


Figure 56 Cancel periodic reporting

4. Various data formats

Data format Currently supports unsigned integer, signed integer, floating point, Boolean, hexadecimal and other data formats. Here are the details.

If the value is 4 bytes or more, the system supports ABCD, CDAB, BACD, and CDBA, where A, B, C, and D are all 8bit (1 byte) hexadecimal numbers, such as 0xE1 and 0x2F.

4.1. Unsigned integer

An unsigned integer is a number that reads values greater than or equal to 0.

A simple unsigned integer is set as follows:

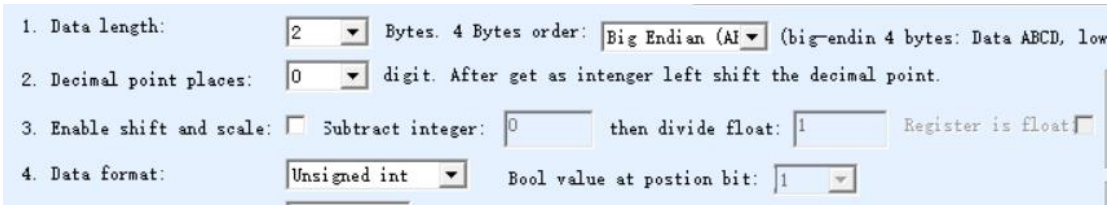


Figure 57 Unsigned integer 2 bytes without decimal point

If two registers are read, the contents of the registers are 123 and 456. The preceding data is: {"uint1":123,"uint2":456}

If the data needs to be shifted, the number of bits to be shifted ranges from 1 to 5. Set "Keep decimal places" in the figure above. If it is set to 4, the uploaded data is {"uint1":0.0123,"uint2":0.0456}. If the option 3 is: {" uint1 ": 0.123," uint2 ": 0.456}, if option 2 is: {" uint1" : 1.23, "uint2" : 4.56}

Small and small end: The two-byte exchange format, that is, the BA format. In this case, you can select the big end exchange format, but not the small end exchange format.

Now change the data length to 4 bytes. reg1 is still 123 and reg2 is still 456. Note that the starting address of the second register may be changed to increment by 2 instead of increment by 1. The preceding data is: {"uint1":123,"uint2":456}

Based on the actual data situation, you can select four formats: big -, small -, big -, and small-end exchange. It also supports the mixed use of four formats. Only data that is greater than or equal to 4 bytes can be selected in the small-endian format, unless the data is in the 2-byte format, that is, the BA format. In this case, the big-endian format can be selected, but the small-endian format cannot be selected.

Unsigned integers of 8 bytes are not supported, and 8 bytes are only available for floating-point numbers.

Supports setting to 0 or NULL when the device is offline or the register is not returned. Setting to NULL can distinguish between a real value of 0 and an offline value of 0. Each node can separately set the non-offline setting value (retain the original value), the offline setting to 0, and the offline upload NULL.

LFigure 58 supports offline clearing to 0 or NULL

4.2. Signed integers

The use of signed and unsigned integers is almost the same. However, when the highest digit of the read data is 1, it indicates a negative number.

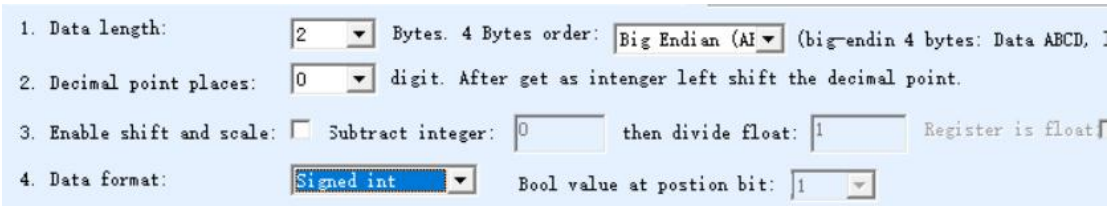


Figure 59 Signed integer 2 bytes without decimal point

The data returned similar: {" int1 ": - 123," int2 ": 0.456," int3 ": 0.00789}

4.3. Floating point type

Floating-point means that the value read is 4 bytes of floating-point data or 8 bytes of double precision data.

A simple floating-point type setting is as follows:

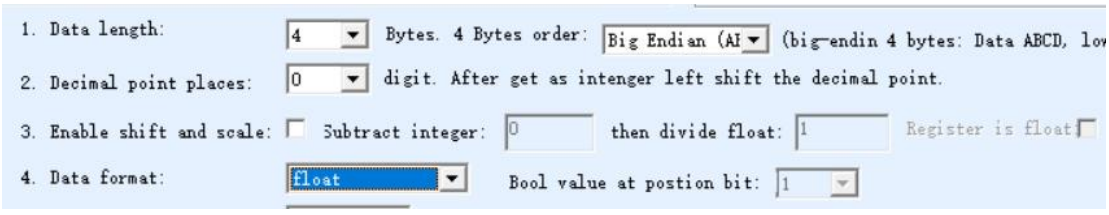


Figure 60 Unsigned integer 2 bytes without decimal point

If you read three registers, the register contents are 1.234567, 2.345678, and 3.456789. If the decimal places are 0, 3, and 5, the preceding data is: {"float1":1,"float2":2.346,"float3":-3.4568}. The original value of the first register is 1.234567, which retains 0 decimal points, so it is 1; Second number, keep 3 decimals

so it is 2.346 (the third decimal here is rounded); The third number is set to 5, but there can only be a maximum of 4 decimal places here, so only 4 decimal points are kept. The maximum floating point number is 4294967296.0.

Based on the actual data situation, you can select four formats: big -, small -, big -, and small-end exchange. It also supports the mixed use of four formats.

Supports setting to 0 or NULL when the device is offline or the register is not returned. Setting to NULL can distinguish between a real value of 0 and an offline value of 0. Each node can separately set the non-offline setting value (retain the original value), the offline setting to 0, and the offline upload NULL.

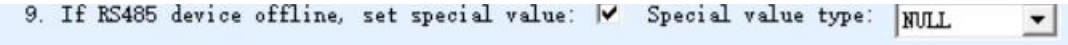


Figure 61 supports offline clearing to 0 or NULL

Supports 8 bytes of double precision floating point type. When the data is 8 bytes, the decimal point of the uploaded data is at most 4 digits.

4.4. It is in hexadecimal format

When choosing hexadecimal, the data is uploaded in hexadecimal according to the data returned by Modbus, and the analysis of floating point and integer formats is not done, which is suitable for uploading some non-standard formats.

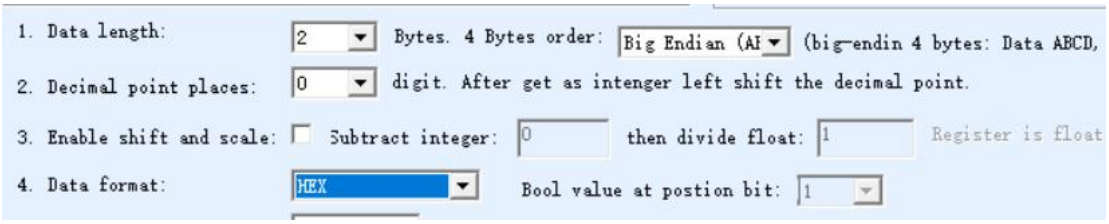


Figure 62 Hexadecimal 2 bytes

When the register content is 3FF3, {"hex":0x3FF3} is uploaded. Supports 2, 4, and 8 bytes of data. For 4 bytes, the format is {"hex":0x11112222}, where 1111 indicates the contents of the lower-level register. 2222 is the contents of the low register after +1. And so on for 8 bytes.

4.5. Boolean type

Boolean type: indicates whether a bit of the 2-byte data read by the 03 and 04

function codes is 1. If it is 1, 1 is uploaded; otherwise, 0 is uploaded. When the 01 or 02 function code is used, the data type is automatically Boolean.

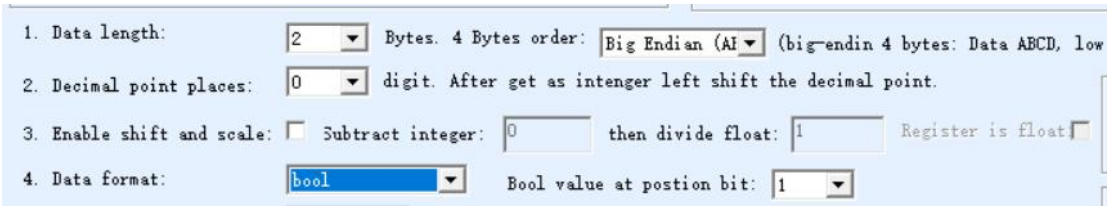


Figure 63 Boolean type 2 bytes

4.6. Number of JSON nodes

The new version (after 2023/10/27) has 1600nodes for 8308 and 2700 nodes for 8305.

5. Deliver the data format

By default, if the json name of the sent data is the same as that of the sent data, the sent parameters are used, and the delivered parameters are invalid. You can set more parameters, including decimal shift, translation scaling, 03 function code BOOL type.

Currently, only data length, format, and type can be configured for delivery.

5.1. Modbus Sets the coil

The way to set it to Off:

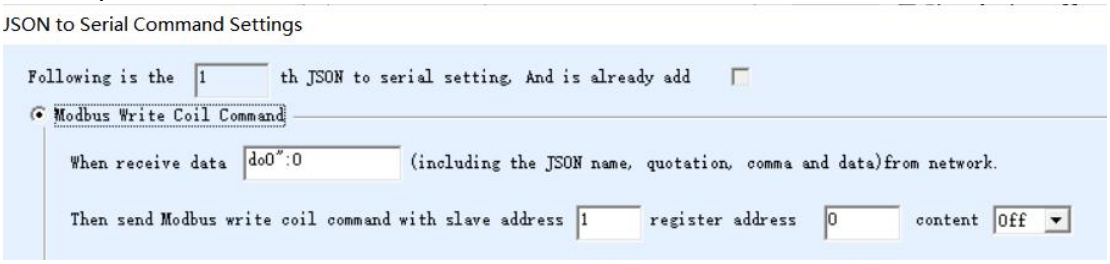


Figure 64 Setting customs

The way to set it to ON:

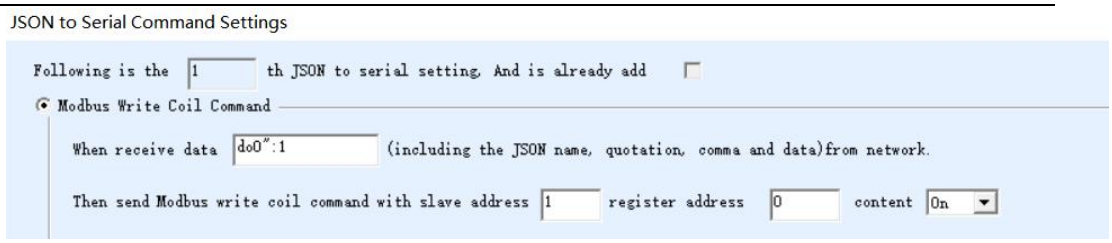


Figure 65 Setup on

5.2. Modbus Set register

The basic mode of setting registers is shown below:

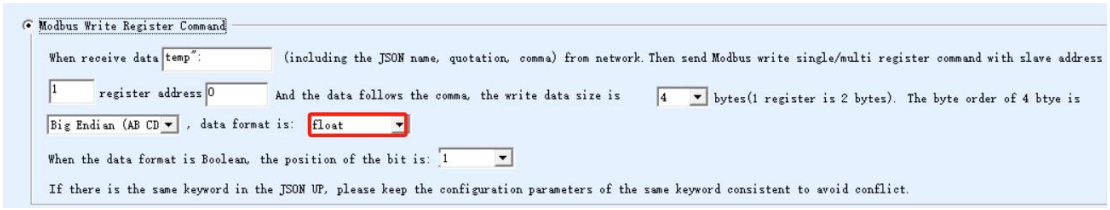


Figure 66 Setting registers

The list of other supported modes is as follows:

Number of bytes	Big and small end	Data format	Instructions
2	/	unsigned integer	Send-1 is also supported, and the actual value is 65535
2	/	signed integer	No different than an unsigned integer
4	AB CD	unsigned integer	No different than an unsigned integer
4	CD AB	unsigned integer	No different than an unsigned integer
4	BA DC	unsigned integer	No different than an unsigned integer
4	DC BA	unsigned integer	No different than an unsigned integer
4	AB CD	Floating point type	
4	CD AB	Floating point type	

4	BA DC	Floating point type	
4	DC BA	Floating point type	
8	AB CD	Floating point type	
8	CD AB	Floating point type	
8	BA DC	Floating point type	
8	DC BA	Floating point type	
/	/	HEX	The HEX type is invalid. It is actually a signed integer
/	/	bool	Boolean type is invalid, it is actually a signed integer

6. MQTT

MQTT can be used alone or in conjunction with JSON functionality. When used alone, the MQTT function transparently transmits serial data to the MQTT server. That is, the data received by the serial port is used as the payload of MQTT. At the same time, the MQTT load will be transparently transmitted from the serial port output. Implement serial port to MQTT.

6.1. Device configuration

First search for the device, then click Edit Device:

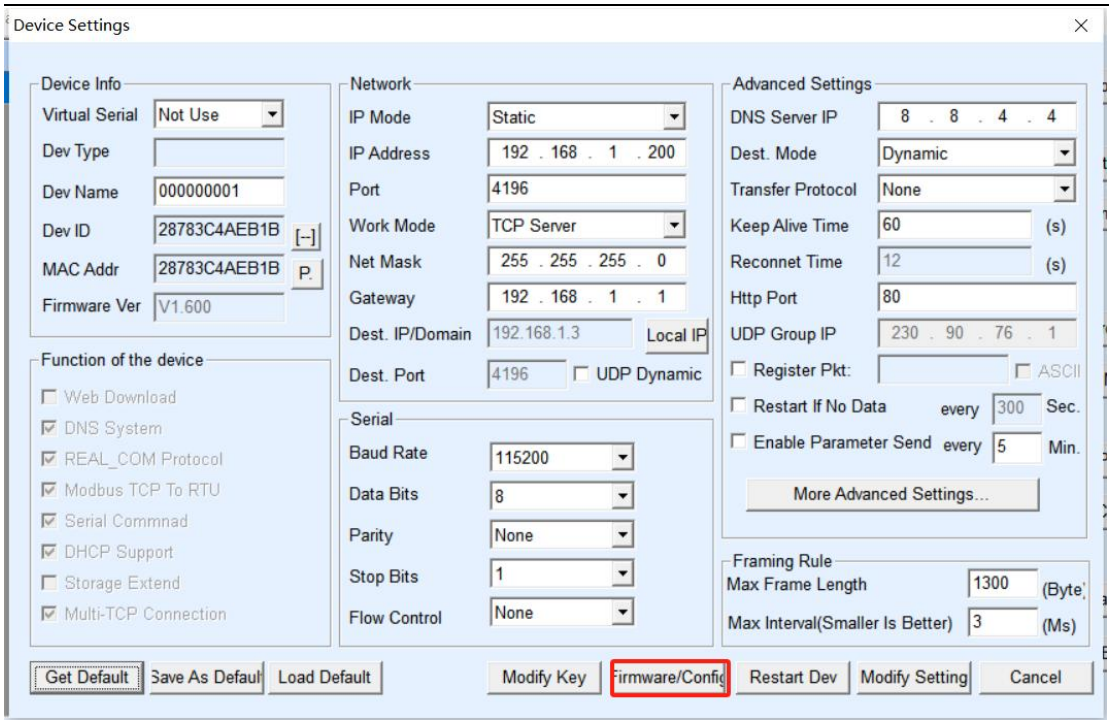


Figure 67MQTT configuration 1

Click "Firmware and Configuration", the configuration download and design dialog box will pop up:

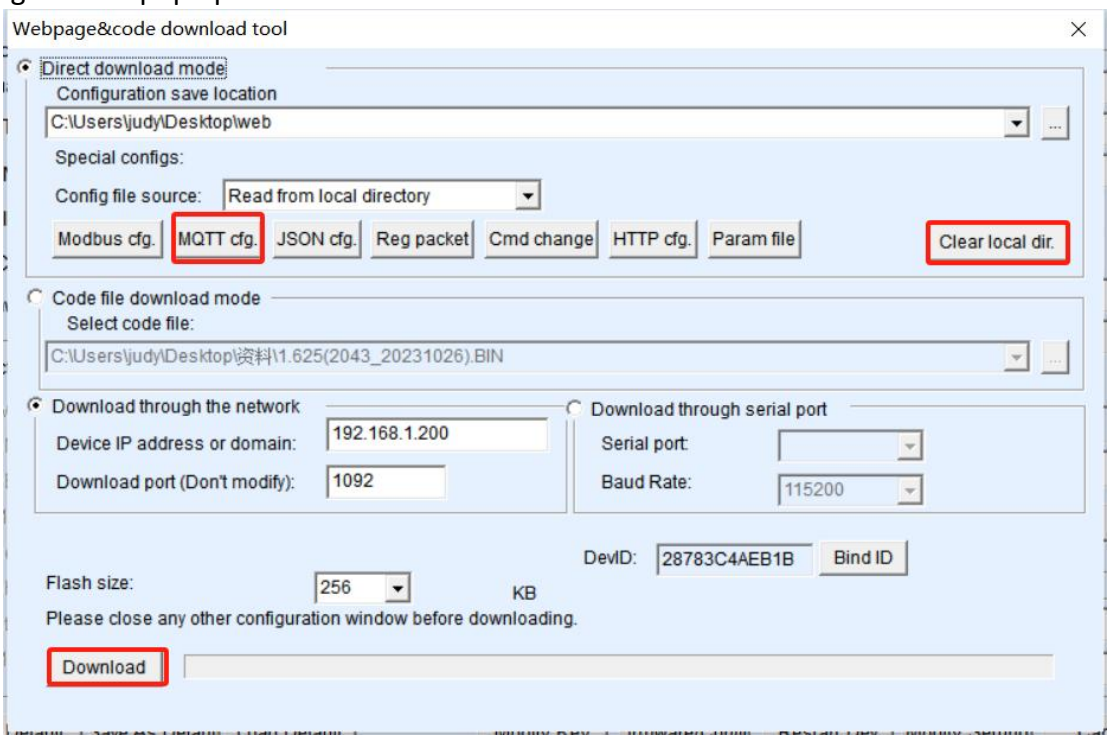


Figure 68MQTT Configuration 2

Here select "Web directory Download", then select an empty directory, such as MQTTHTTPD directory, and then click "Clear all" to clear the previous design (note that if the previous design was designed with JSON, do not clear all, otherwise the previous JSON design will be cleared). Then click MQTT Configuration.

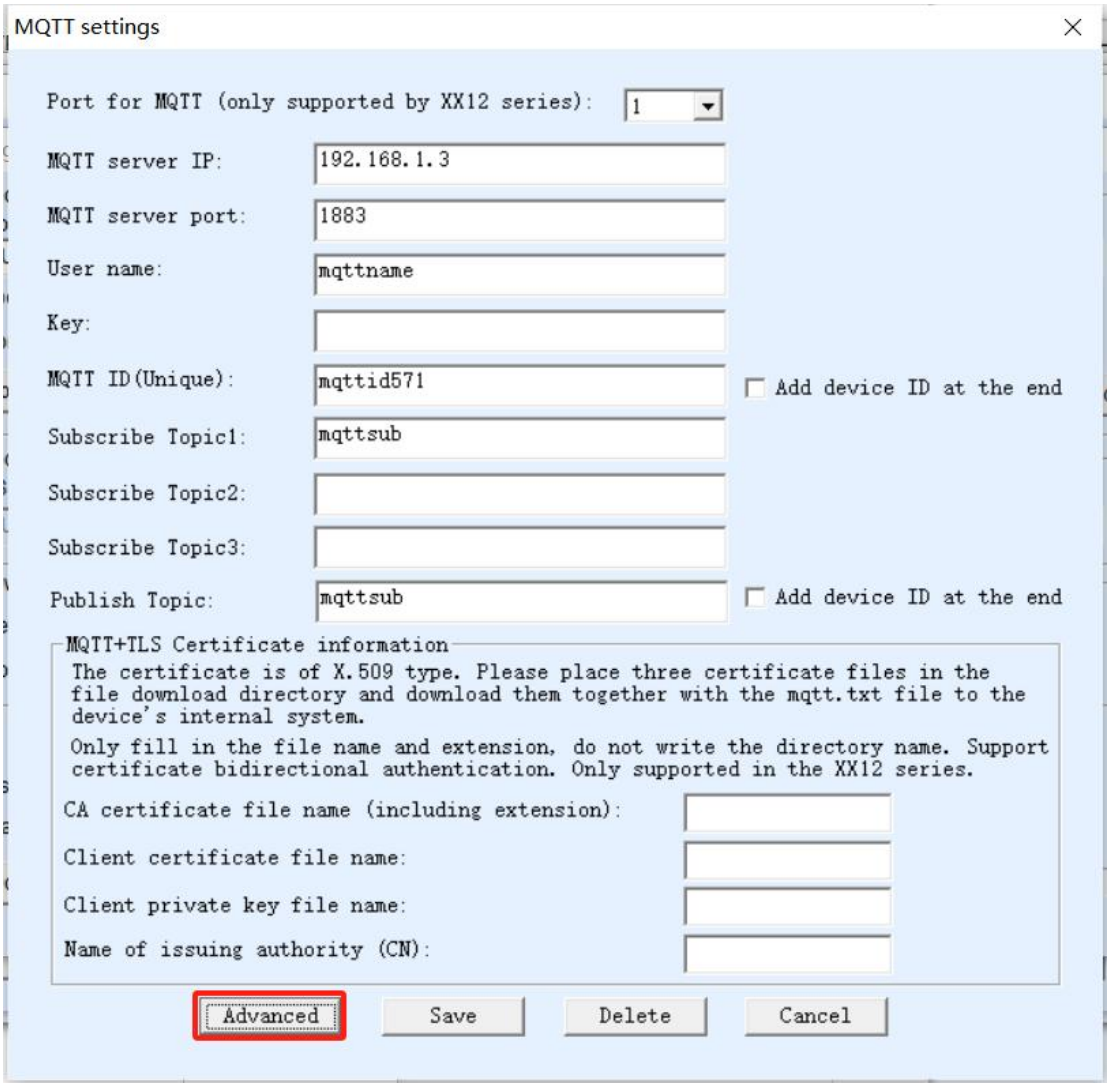


Figure 69MQTT configuration 3

这里配置说明如下：

1 Server domain name or IP: Enter the IP address of the MQTT server. The maximum length is 30 characters.

2 User name: indicates the user name of the MQTT server.

3 Password: specifies the login password of the user.

4 client ID: indicates the client ID of the MQTT.

5 Subscription theme: This is the theme subscribed by this device. When other devices publish this theme, the server will send it to this device. If you are only publishing, you do not need to fill in this field.

6 Publish topic: The topic of the data sent to the server when the serial port of the device is converted to MQTT.

7 MQTT Advanced parameters: Used to configure advanced parameters.

8 Save MQTT Settings: Click this button to save the design, and then click the "Download button" in the web download directory to download.

Now first click on "MQTT Advanced parameters" (generally do not need to configure advanced parameters) :

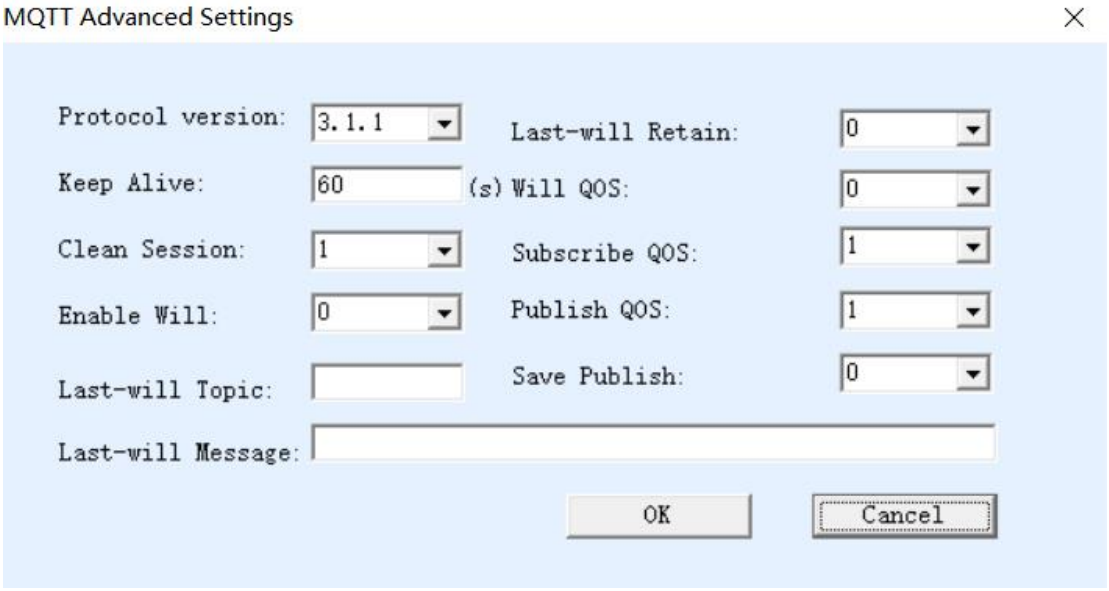


Figure 70MQTT advanced parameter configuration

The explanation is as follows:

1 Protocol Version: The current mainstream is 3.1.1 version, if you need to select 3.1 version, please select here.

2 Keepalive time: The heartbeat time of MQTT, at least 10 seconds, the default is 60 seconds.

3 Server Clear subscription: Specifies whether the server clears subscription information after the client disconnects.

4 Whether to Enable Last Wishes: Whether there is a last wish.

5 Last Wish theme: Last wish theme.

6 Last Wish information: Information about last wishes.

7 Save Wishes: Specifies whether the server needs to send a wish message to the client when the client is offline.

8 Wish Quality: The delivery quality level of the last wish message sent by the server.

9 Subscription Quality: The delivery quality level of the subscription. In some cases, the value must be set to 0 to prevent disconnection caused by retransmission.

10 Release Quality: The delivery quality level of the client's published messages. In some cases, the value must be set to 0 to prevent disconnection caused by retransmission.

11 Whether to save publishing: Whether the server keeps the last message (sent to the client if there is a new client subscription).

We do not modify advanced parameters here. Click "Save MQTT Settings" directly. Then click "Download"

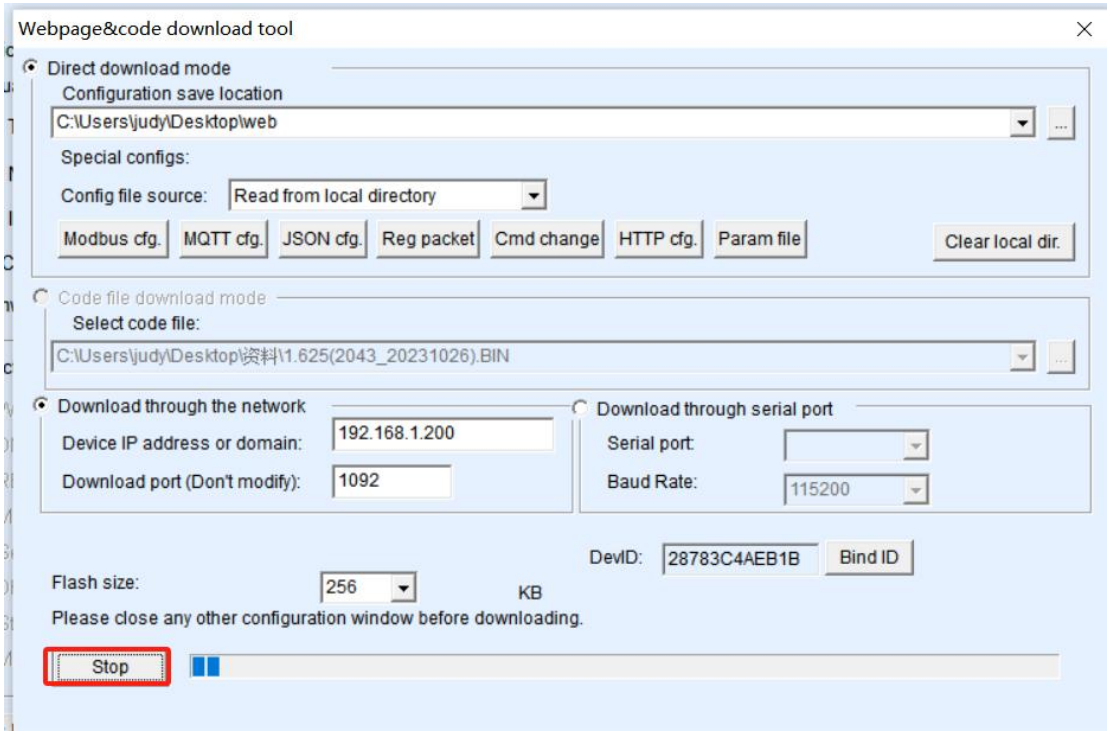


Figure 71 Download

After downloading, click OK. At this time, you will return to the device management dialog box and see that the destination IP, working mode and destination port of the device have been automatically modified to MQTT Settings:

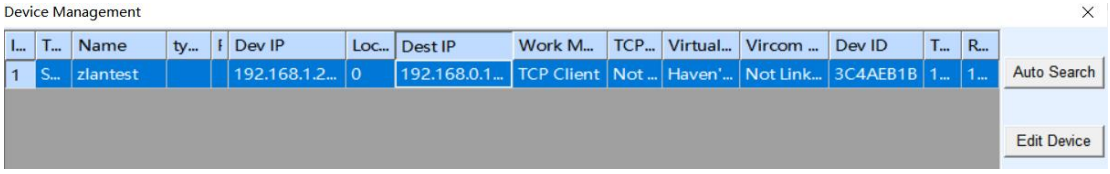


Figure 72 Automatic modification

If no, you need to set the destination IP address, working mode, and destination port in the device Edit dialog box. Then click "Modify Settings".

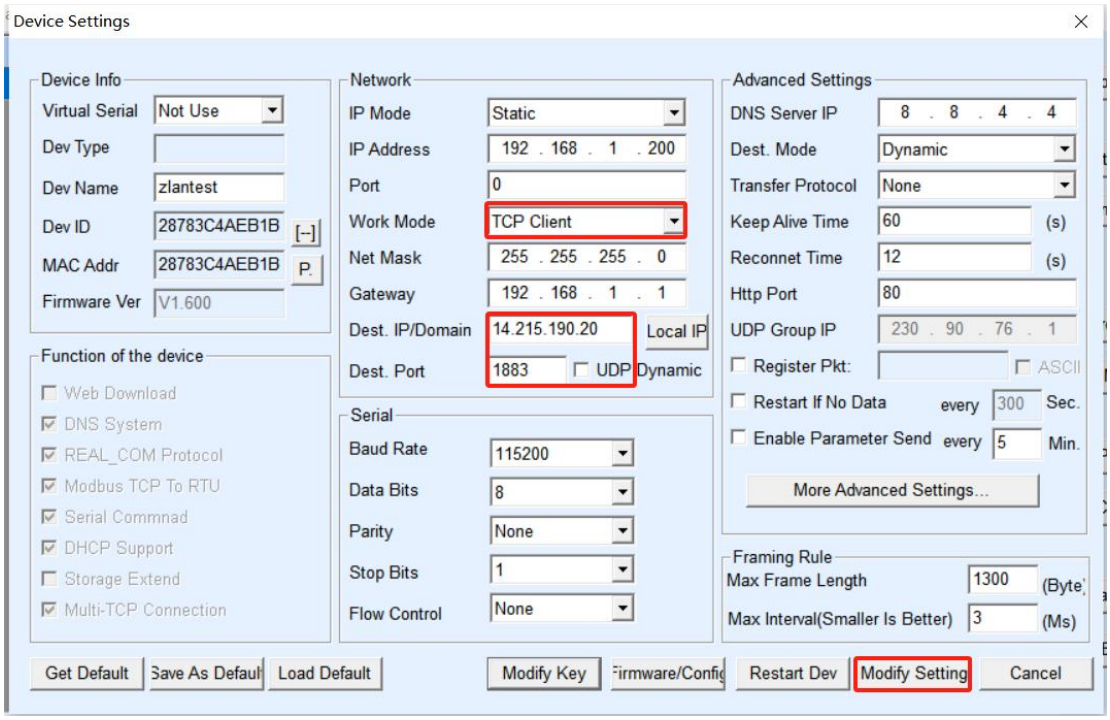


Figure 73IP configuration

This completes the configuration.

6.2. Data testing

After the connection is complete, the LINK light of the device (usually the blue light in the middle) turns on. The device is connected to the MQTT server.

Now open the serial tool:

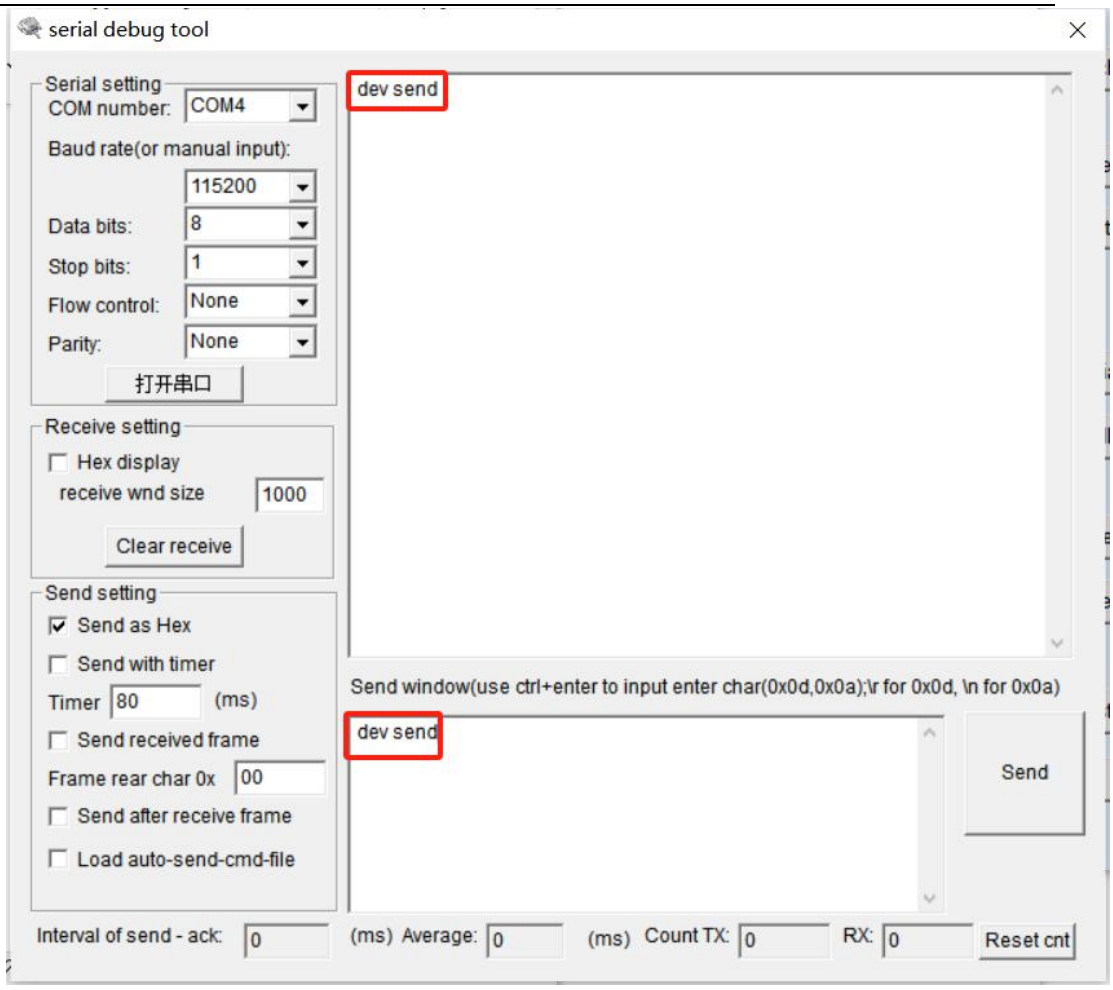


Figure 74 Serial port sending and receiving

Open the serial port using the same baud rate as the device, and send dev send. In the receiving window, dev send is displayed. This is due to the fact that dev send this information to the MQTT server under the subject of Womastersub. But our device also subscribed to the Womastersub theme, so the server immediately sent us a subscription message, the content of the subscription message is dev send. This information is downloaded as MQTT payload and output from the serial port through transparent transmission.

If other devices send messages, this device can also receive data.

In general, the user can directly transmit the serial port instructions (such as Modbus RTU) directly to the MQTT server. In addition, you can also use JSON function, using automatic Modbus RTU format collection, timed JSON format on the

form. In addition, you can also find Womaster to customize some non-standard instruments and host computer protocol formats.

7. MQTT+JSON TO Modbus RTU

The combination of the above JSON and MQTT can achieve the following functions:

1. Establish a connection with the server using the MQTT-based protocol, and conduct data communication in the form of subscription and publication.
2. Support the independent design and automatic collection of Modbus RTU register.
3. Support to convert specific Modbus register contents into JSON format and send them regularly.
4. Device ids can be added to the JSON format to facilitate the cloud cloud to identify devices. 1. Establish a connection with the server using the MQTT-based protocol, and conduct data communication in the form of subscription and publication.
5. Support the independent design and automatic collection of Modbus RTU register.
6. Support to convert specific Modbus register contents into JSON format and send them regularly.
7. Device ids can be added to the JSON format to facilitate the cloud cloud to identify devices.

If you need MQTT+JSON to Modbus RTU function, you can design MQTT and JSON separately, in no particular order. Do not click the "Clear design" button after designing one, and click the "Download" button together to download the device content after two designs are finished.

Generally, you can manually restart the device and load the Settings.

8. HTTP POST/GET+JSON

The host protocol can also choose HTTP protocol in addition to MQTT, and upload data through POST and GET instructions. The following uses the POST command as an example.

If you need to support POST/GET+JSON function Vircom configuration tool select 5.17 or later version; In addition, if you need to support the POST command, the 2003 firmware needs to be in 5.81 and above (if you only support GET, use the normal version 2003 firmware that supports JSON).

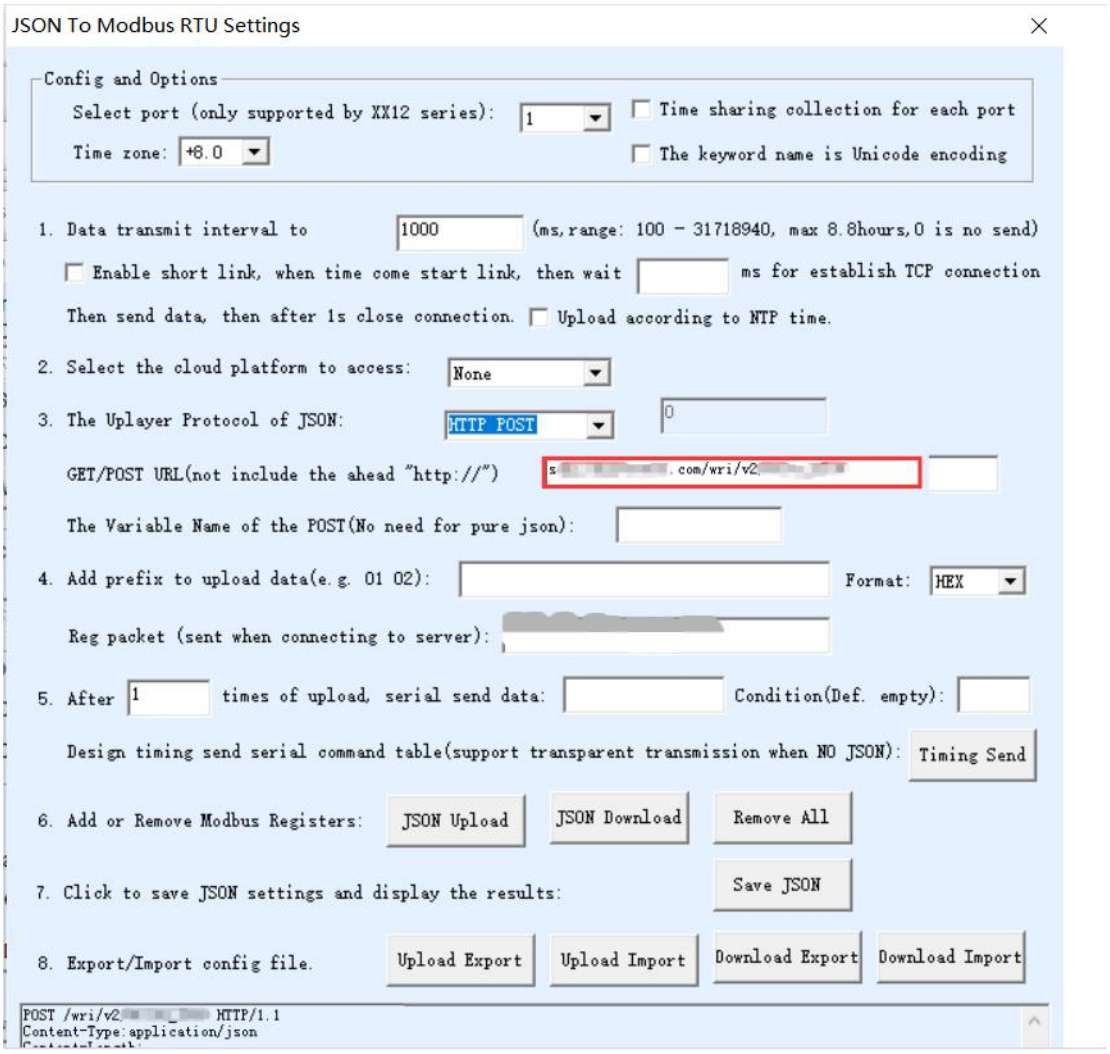


Figure 75 POST+JSON

Vircom5.17 added two options in JOSN to Modbus RTU Settings, as shown in the figure:

1. Upper-layer protocol of JSON: If no protocol or MQTT protocol is used, select the first option: "NONE/MQTT". Select the second item "HTTP POST" for HTTP POST and the third item "HTTP GET" for HTTP GET.
2. URL of POST/GET: You must enter the URL when selecting POST or GET. For example, if the URL is http://s.a.com/wri/v2, delete http:// and enter s.a.com/wri/v2 directly.

The other JSON structure design method is the same as the method introduced before, and then click the "Save JSON Settings" button, if you select the POST/GET will add HTTP header format information in front of the JSON data to support the HTTP transmission protocol.

This POST/GET design method is simple and practical, and can simply and quickly realize the transmission of instrument data such as Modbus RTU to the server in the way of HTTP POST/GET+JSON.

9. Add the prefix +BASE64 encoding to the prefix

Now you need to add a secret field and a timestamp field in front of the normal json format, where secret is a fixed password and timestamp represents a 10-digit timestamp of the current time. All uploaded data is then encoded using Base64. That is, the last upload format is base64Encode(secret+ "" +timestamp+" "+json).

The 5812LD firmware version 1.481(2012L).bin is introduced here with Vircom version 6.73.

First, create an empty local folder directory, and then select this directory in Vircom to store configuration information.

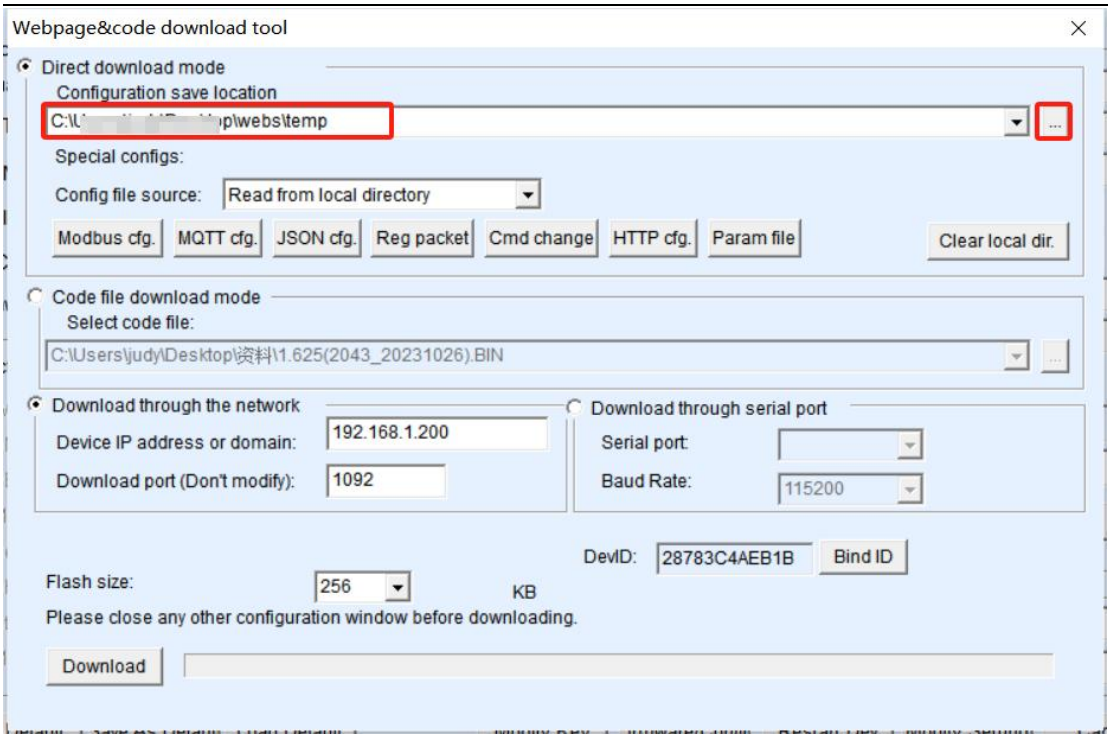


Figure 76 Selecting the configuration directory

9.1. Adding a prefix

Click the "JSON Configuration" in the above picture, and enter the following data in the "Upload data Add frame header" field:

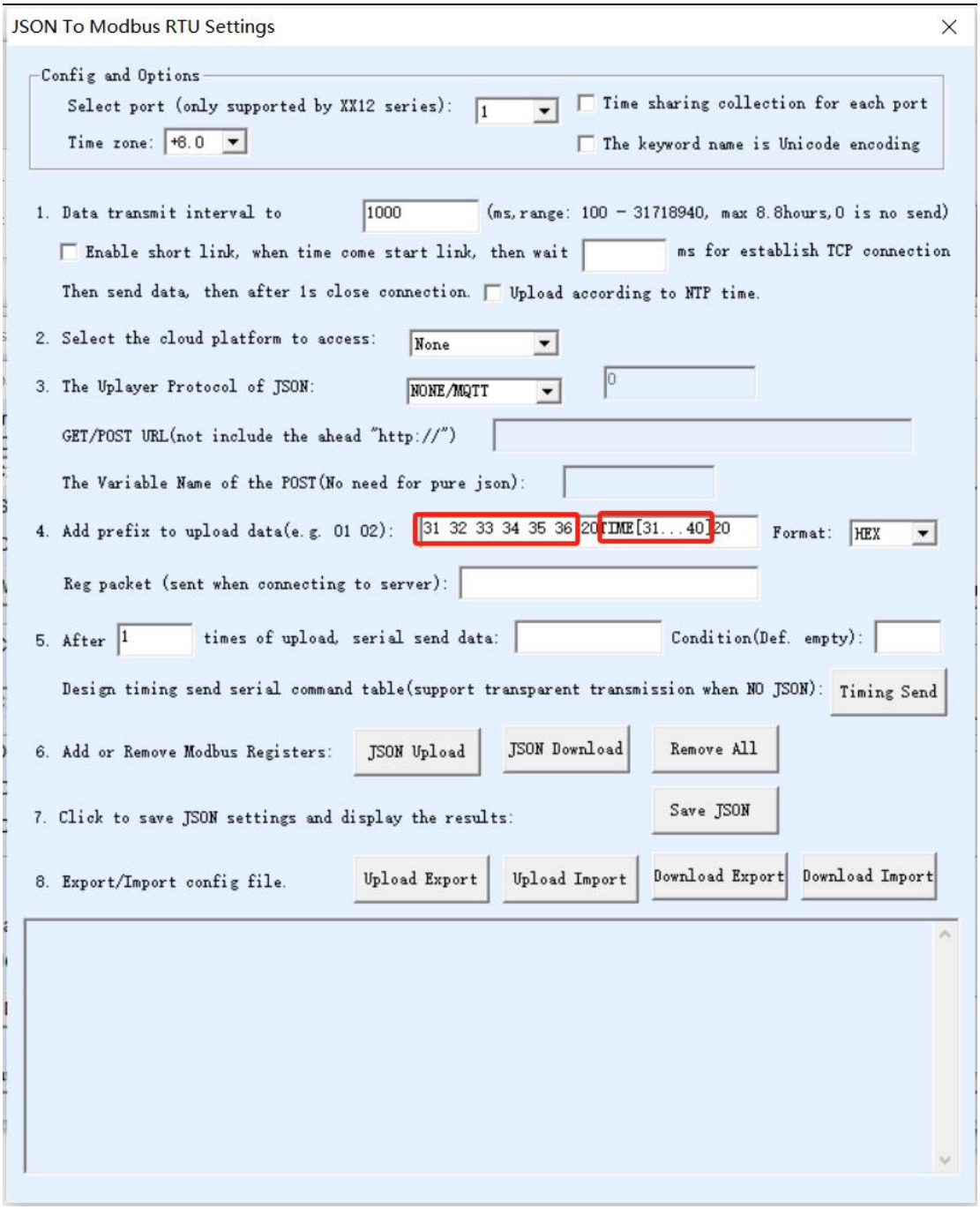


Figure 77 Input frame header

这 Here 31 32 33 34 35 36 20 TIME[31...40] 20 is explained as follows: The yellow part 31 32 33 34 35 36 is the hexadecimal representation of the secret string. In order to be able to input strings directly, the "format" HEX on the right can be changed to ASCII first, and then changed to HEX after entering secrect in string form,

so that the bit HEX form is automatically converted. The green 20 is a space. The blue TIME[31...40] is a 10-bit timestamp (note that Vircom6.73 or higher is required to support this). All fields are separated by Spaces.

Then click "send on JSON" to configure Modbus RTU to JSON, which will not be repeated here. Then go back to the above interface and click "Save JSON Settings", it is OK.

9.2. BASE64 encoding

In order to support BASE64 encoding, on the basis of the above, click the "parameter file" button of the "Configure Web page/program download tool" dialog box above, in the pop-up dialog box, enter BASE64_EN, and enter 1 on the right.

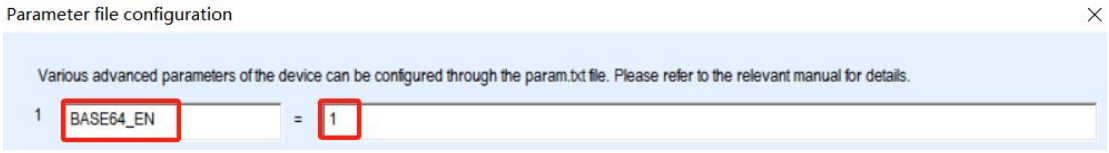


Figure 78 BASE64 encoding

Then click "Save". This configuration will be saved as a param.txt file to the local computer directory and downloaded to the device along with the json configuration to enable base64 encoding.

BASE64_EN=1 will make the device uplink data encoded through base64, including mqtt upload and transparent upload.

This function is not enabled if param.txt does not exist, the BASE64_EN configuration does not exist, or BASE64_EN=0.

9.3. Mqtt Configuration

In order to support the MQTT protocol, on the basis of the above, click the "MQTT configuration" button of the "Configuration page/program download tool" dialog box above, in the pop-up dialog box, enter your own MQTT configuration, the following is just an example:

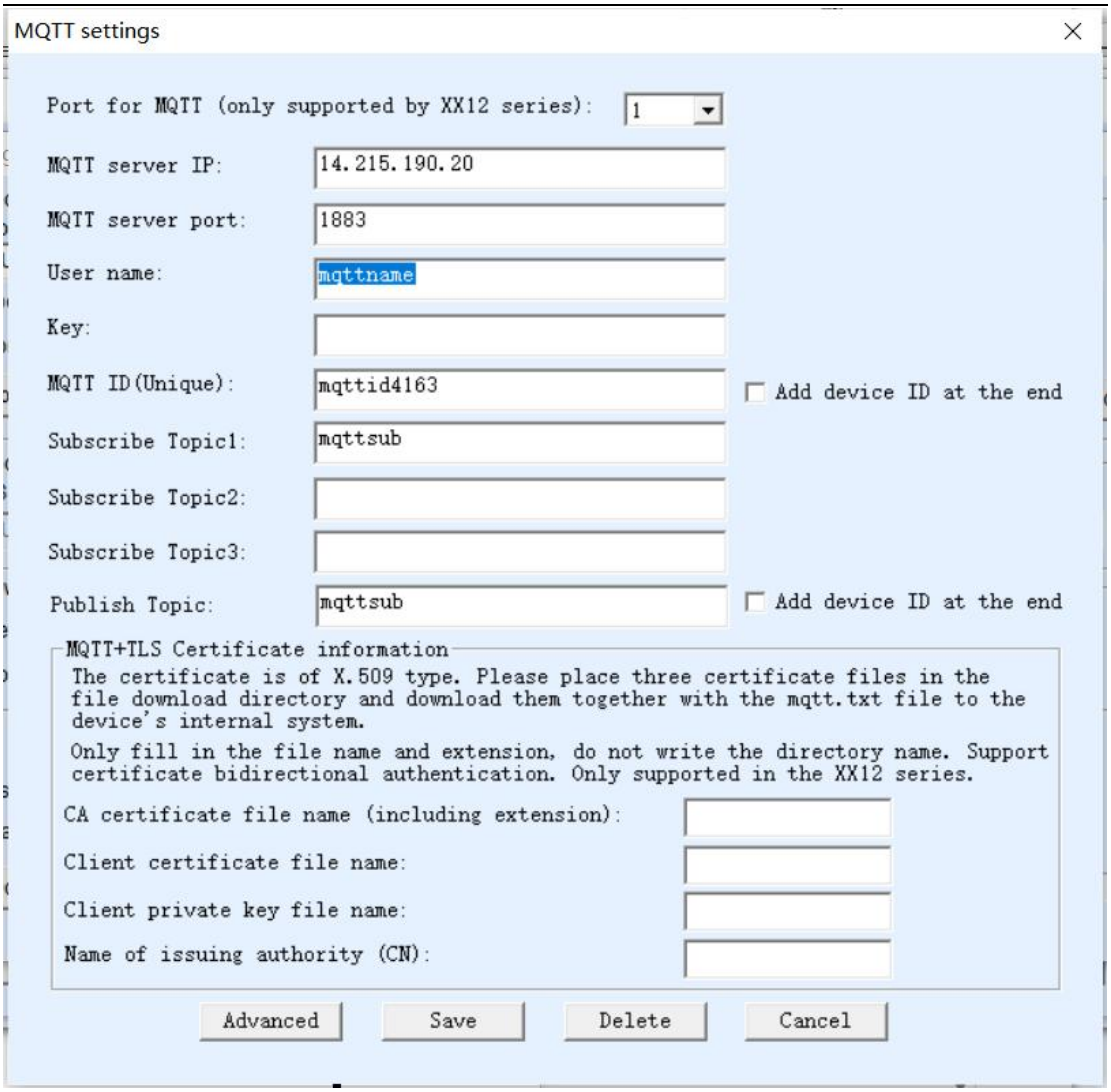


Figure 79 MQTT configuration

Then click "Save MQTT Settings".

9.4. Test

Go back to the "Configure Web/program Download Tool" dialog box and click the "Download" button to download the previous json configuration, base64 configuration, mqtt configuration to the device.

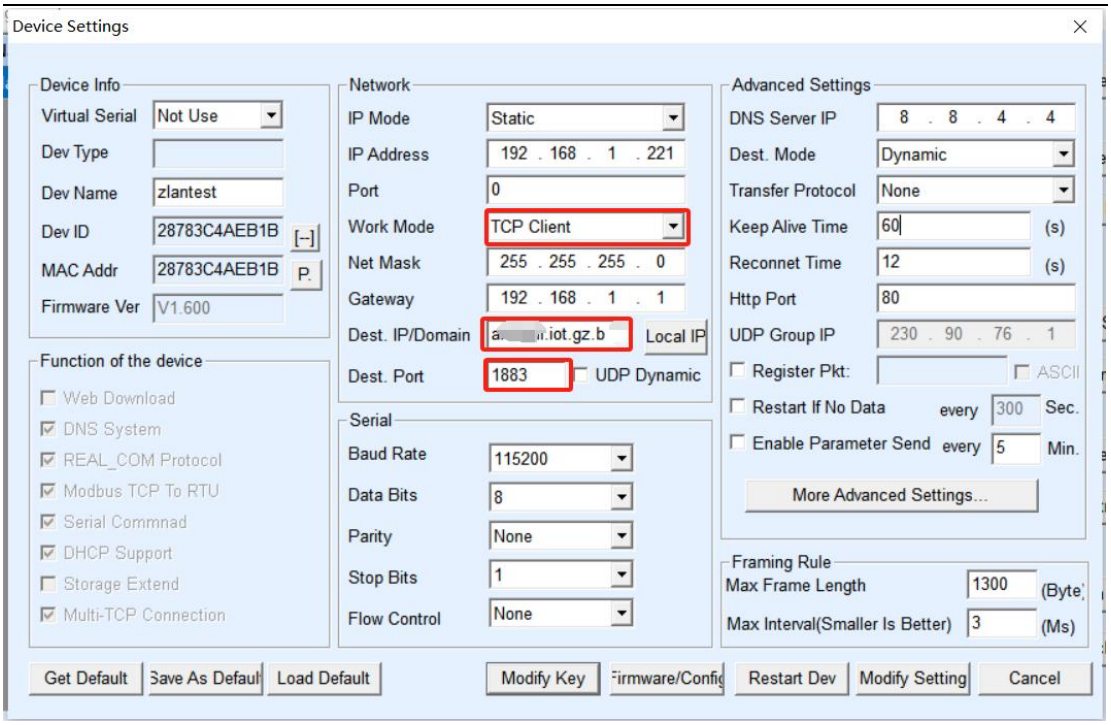


Figure 80 Confirm parameters

Check whether the device works in TCP client mode, whether the destination domain name points to the MQTT server, and whether the destination port is correct. If it is incorrect, modify it.

To subscribe to this topic using MQTT fx software, you can receive the subscription information in base64 encoding format:

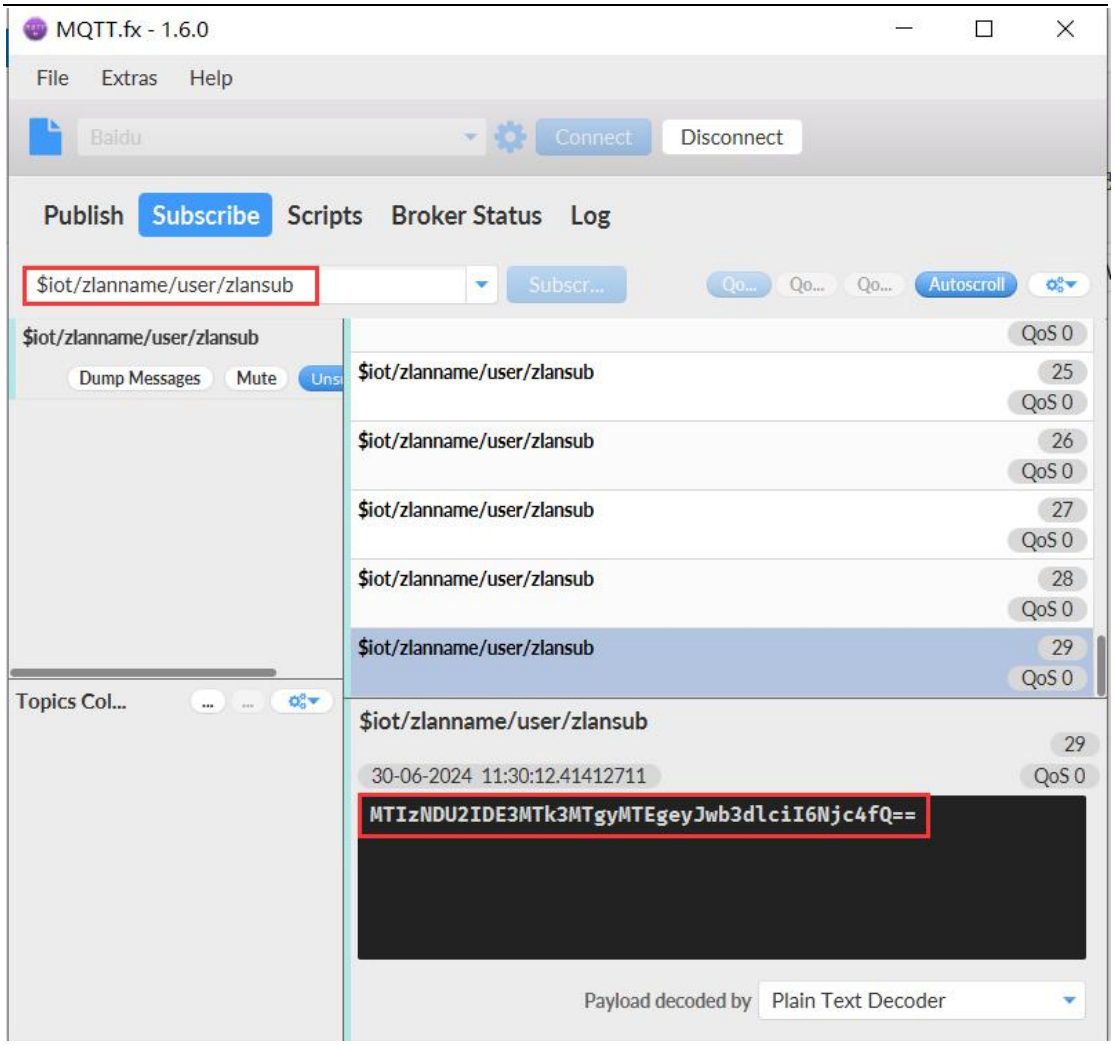


Figure 81 Subscription information

Copy the base64 information and open the online encoding and decoding tool for base64 decoding: Get the required secret+ "" +timestamp+" "+json raw data.

10. Multiple mqtt+json setting methods

This feature is only supported by some series of products and requires firmware of 1.445 or higher. It is configured with Vircom 6.45 and higher versions.

Select any port of this series in Vircom for configuration.

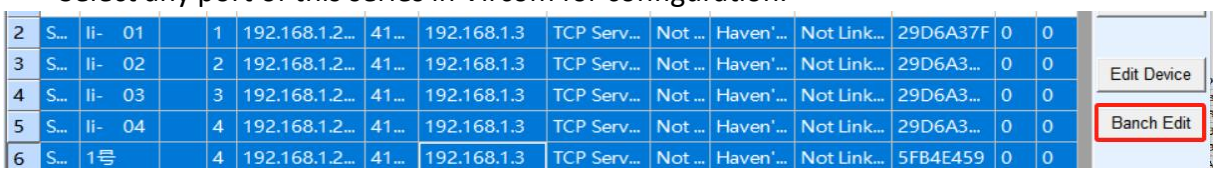


Figure 83 Select a PORT

10.1. Mqtt disposition

Click on "Edit Device", click on "Firmware and Configuration", click on "Configure Web directory Download", then click on "MQTT Settings"

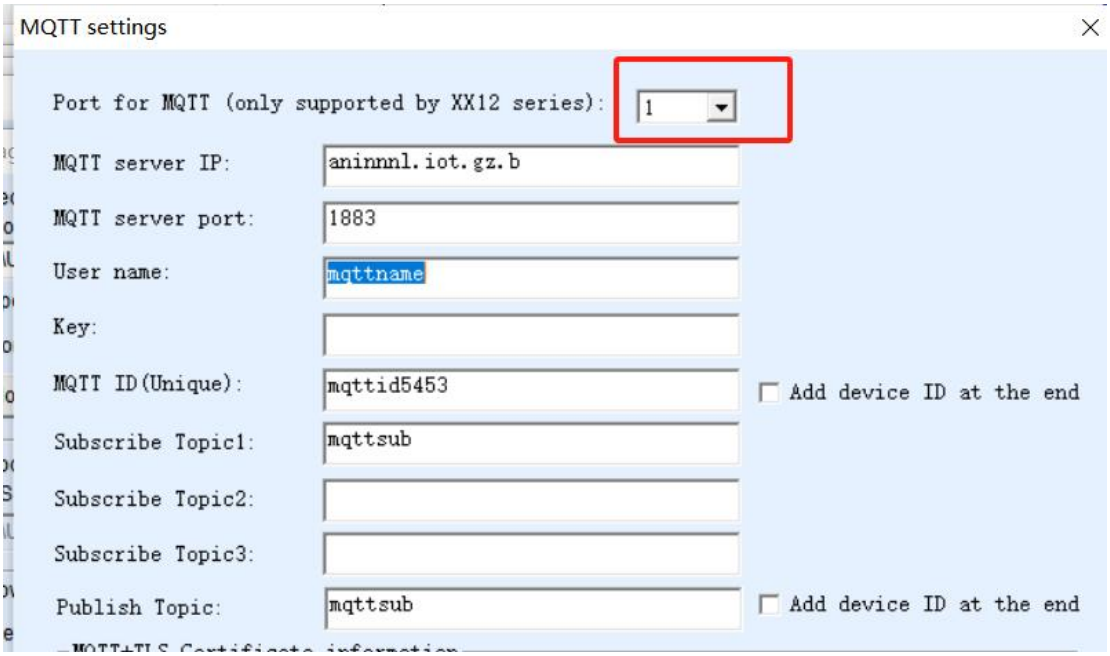


Figure 84 Multiple MQTT Settings

Unlike normal MQTT configurations, a Port PORT can be selected to indicate which port this MQTT is bound to. Click "Save mqtt Settings" to save the disk will be saved as MQtt.txt, mqtt2.txt~mqtt8.txt (depending on the port), these files will be downloaded to the device internal. If there are several mqtt files, there should be several mqtt connections.

Because different ports correspond to different serial ports (serial ports 1 to serial ports 8), they also correspond to different IP or TCP ports. So different mqtt files correspond to different serial ports and different tcp connections.

Note that clicking "Save MQTT Settings" only saves the mqtt configuration of the current PORT, and does not save the mqtt Settings of other ports. If you need to save the MQTT Settings of other ports, please change the PORT of the dialog box to save again.

Also, the client ID for each PORT needs to be kept different.

Three MQTTS are set here, with themes corresponding to mqttsub1 to mqttsub3.

10.2. JSON Disposition

Similarly, in the Configure Page/Program Download dialog box, click "JSON Configuration". The value is set to Modbus 03 slave 01 addr 01 in JSON. The JSON keyword sent is reg1. Then select PORT as 1 and click "Save JSON Settings".

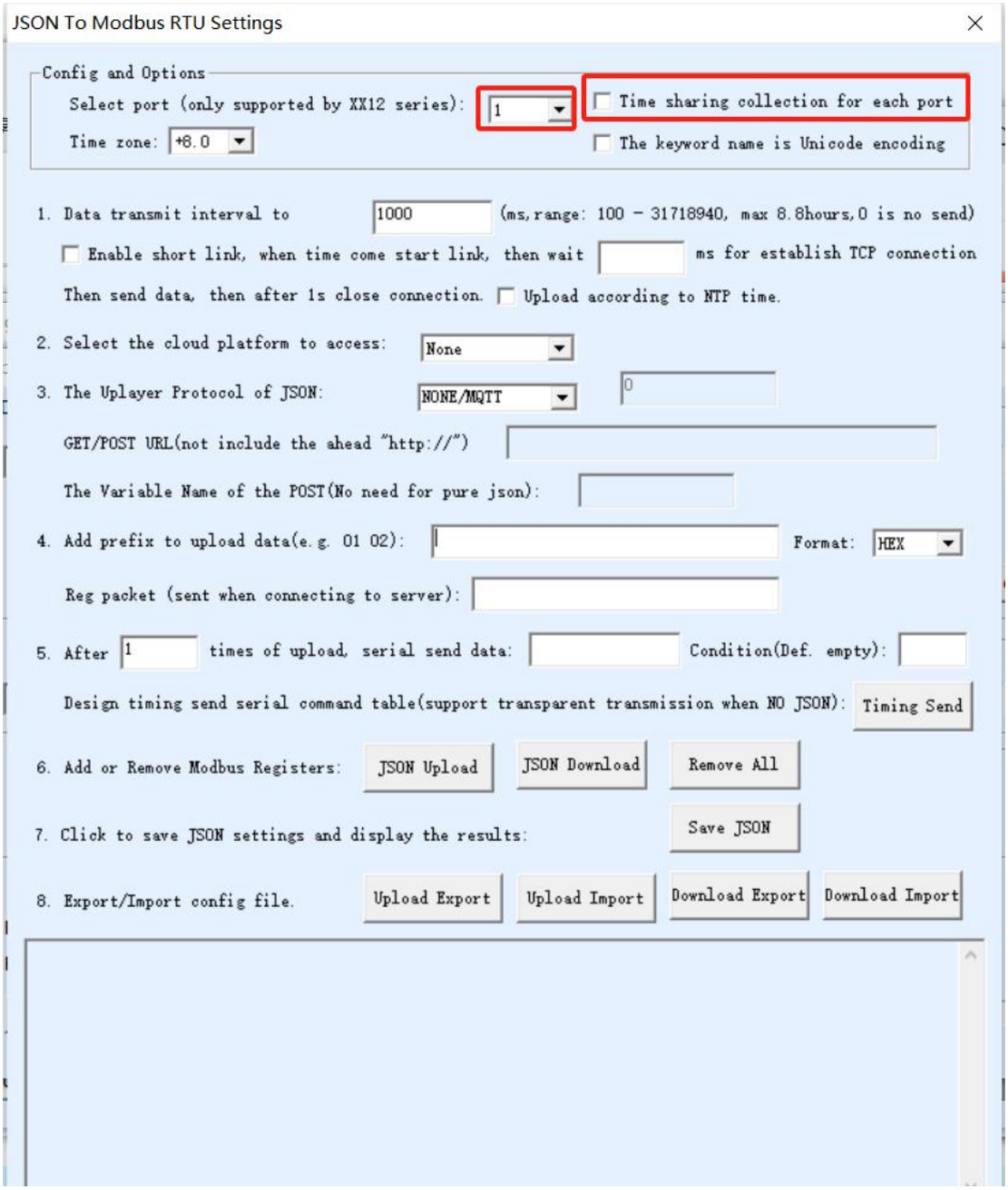


Figure 85 Multi-JSON Settings

Similar to MQTT Settings, the choice of PORT number will cause the JSON to be saved as httpd.txt,httpd2.txt~httpd8.txt, indicating the serial port and TCP connection corresponding to the data collection.

The "Time-sharing collection for each Port" option coordinates the collection time for each Port, which is described later.

If you need to change the PORT port, click Save JSON Settings again. This will result in multiple httpd.txt files in the directory.

Now download these mqtt files, along with httd files, into the device. And configure the device as a client to connect to the mqtt server.

10.3. Collection of each Port by time

The above mentioned option "time-sharing collection by Port" is to coordinate the collection time of each Port. When multiple servers need to obtain data from the same Modbus RTU instrument, you can merge the RS485 serial ports of multiple ports, connect them to the instrument, and configure them according to the above method. However, multiple ports may send data at the same time, resulting in conflicts and code errors during collection.

At this time, when you need to configure JSON, select "each Port time-sharing collection". After being selected, the httpd.txt file will become an HTTpc. txt file, and multiple httpc files will be coordinated to ensure that the serial port will not output instructions at the same time, but output in a certain order. To realize the purpose of combining multiple ports to collect data on one RS485 bus.

In order to see the effect of time-sharing collected, it is recommended to change the serial port polling time of JSON to a larger 1000ms

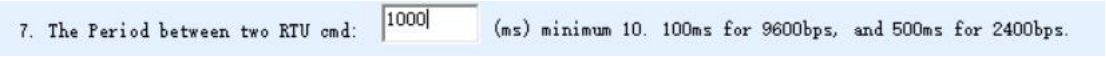


Figure 86 Modifying the polling time

10.4. MQTT Multiple subscription topics

Supports subscribing to 3 topics at the same time. For details, see Identifying JSON Tape Delivery ids

10.5. MQTT topic contains ID

The MQTTclient id sent on the support contains the device id, and the published theme contains the device id. This enables mqtt.txt to be the same for each device, but with different MQTTclient ids and different publishing topics. For details, see Identifying JSON Tape Delivery ids

10.6. JSON Tape delivery ID identification

JSON Delivery ID identification means that when all devices subscribe to a topic at the same time and the data delivered is {" ID ":" 010203040506 ", "Keyword" :123}, only the device with the corresponding ID updates the register corresponding to the Keyword. Other devices that do not match the message ignore this message. This feature allows different devices subscribing to the same topic to execute commands separately, rather than all executing the same command. This feature is supported by all models, as long as you update vircom.

MQTT aspect: In terms of MQTT: Currently, each serial port of the module supports multiple different subscription topics, up to a maximum of 3. At the same time, in order to simplify the configuration steps of client MQTT, the function of adding device IDs and publishing topic suffixes to MQTT client IDs has been updated.

JSON: module is now updated to add the device ID prefix function, support by the device ID to determine whether to execute subsequent JSON delivery commands.

Firmware version: 1.447 (2012) or later.

Figure 87 Firmware version

VIRCOM software version: 6.46 or later.

Figure 88VIRCOM version

MQTT/JSON Disposition

MQTT Settings

Click MQTT Settings and select the first PORT1 port in the upper right corner. Because the MQTT client ID needs to be unique, we can add the device ID option at the end, as shown in the following figure, if the device ID is 28649C1CCA1C. Then the actual client ID of MQTT1 is mqttid28649C1CCA1C. The MQTT1 publishing theme also supports trailing device ids. After this parameter is selected, the publication topic of MQTT1 is mqttsub28649C1CCA1C. MQTT1 subscription topic biggest support three at the same time, this way we set to mqttsub1 respectively, mqttsub3, mqttsub4. Finally, you need to click Save MQTT Settings to make the MQTT parameters of the corresponding PORT take effect.

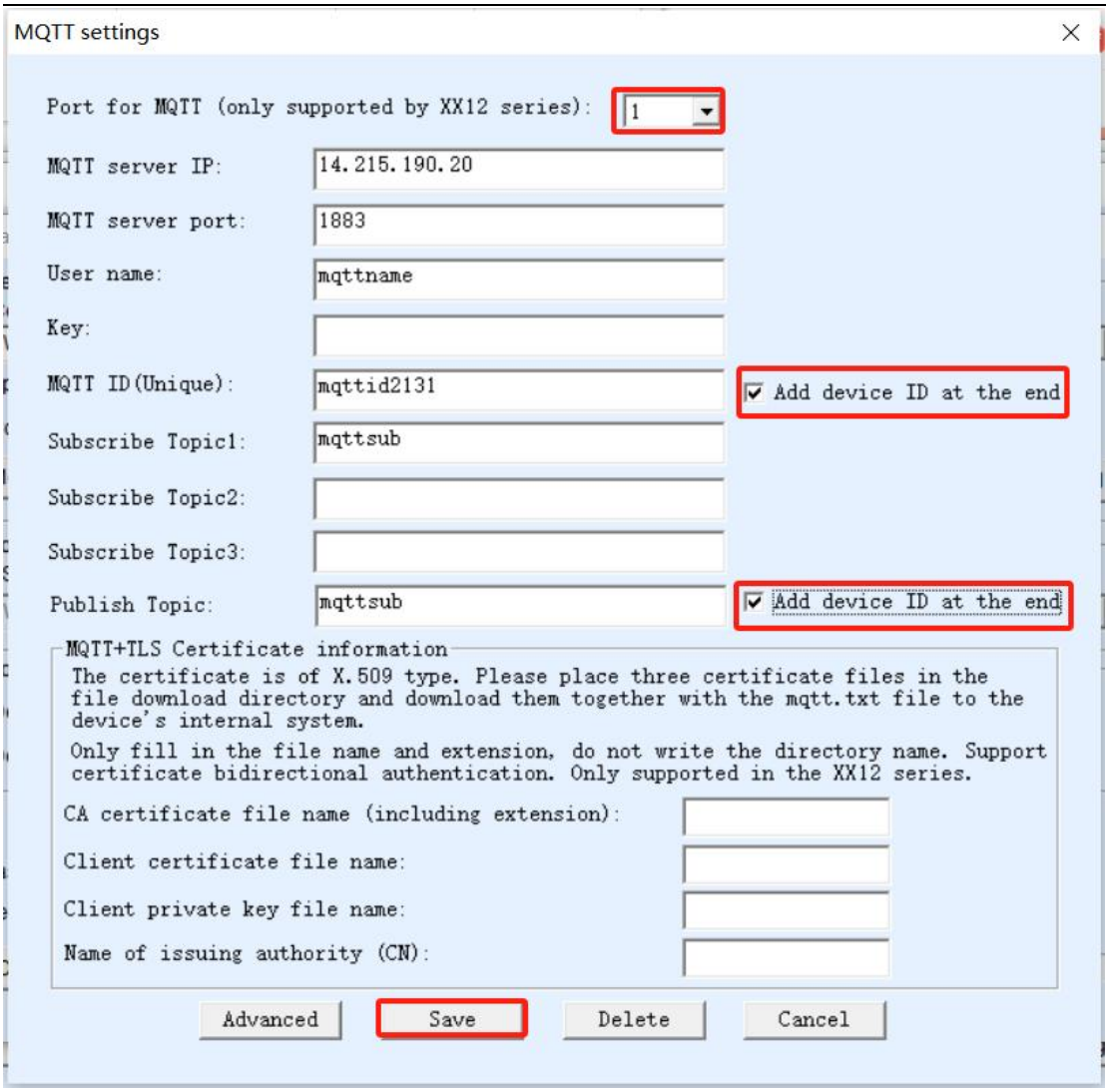


Figure 89MQTT1 Settings

The top right corner to choose two PORT2, other parameters we shall, in accordance with the just MQTT1 set, only the subscribe to the topic of MQTT2 separately set to mqttsub2, mqttsub3, mqttsub4. Where mqttsub3 and mqttsub4 are equal to PORT1 and 2 subscribed at the same time.

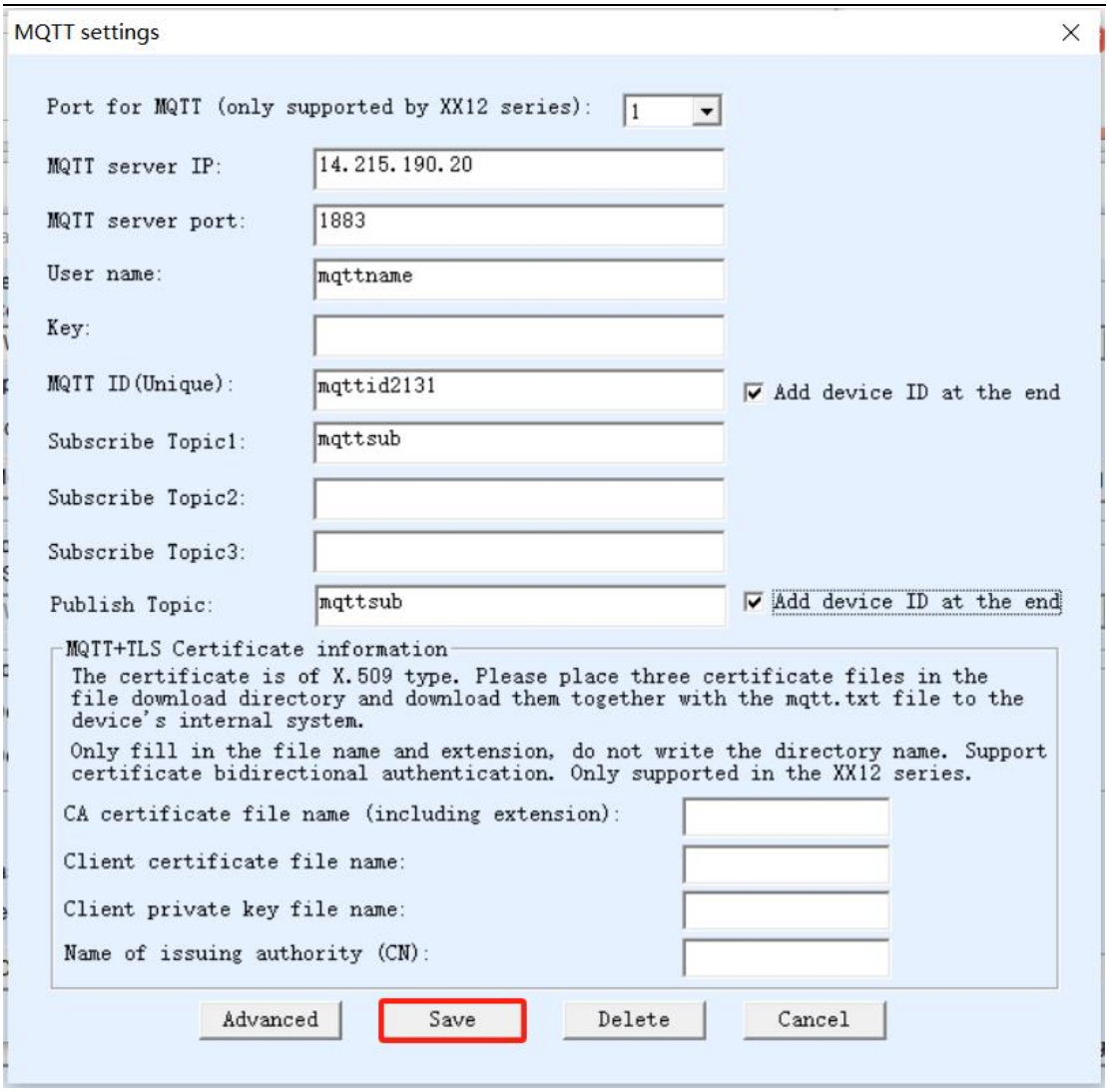


Figure 90MQTT2 Settings

JSON Set

Collect Port select 1, click on JSON to send, add keyword TEMP1, station address 1, function code 3, register address 0.

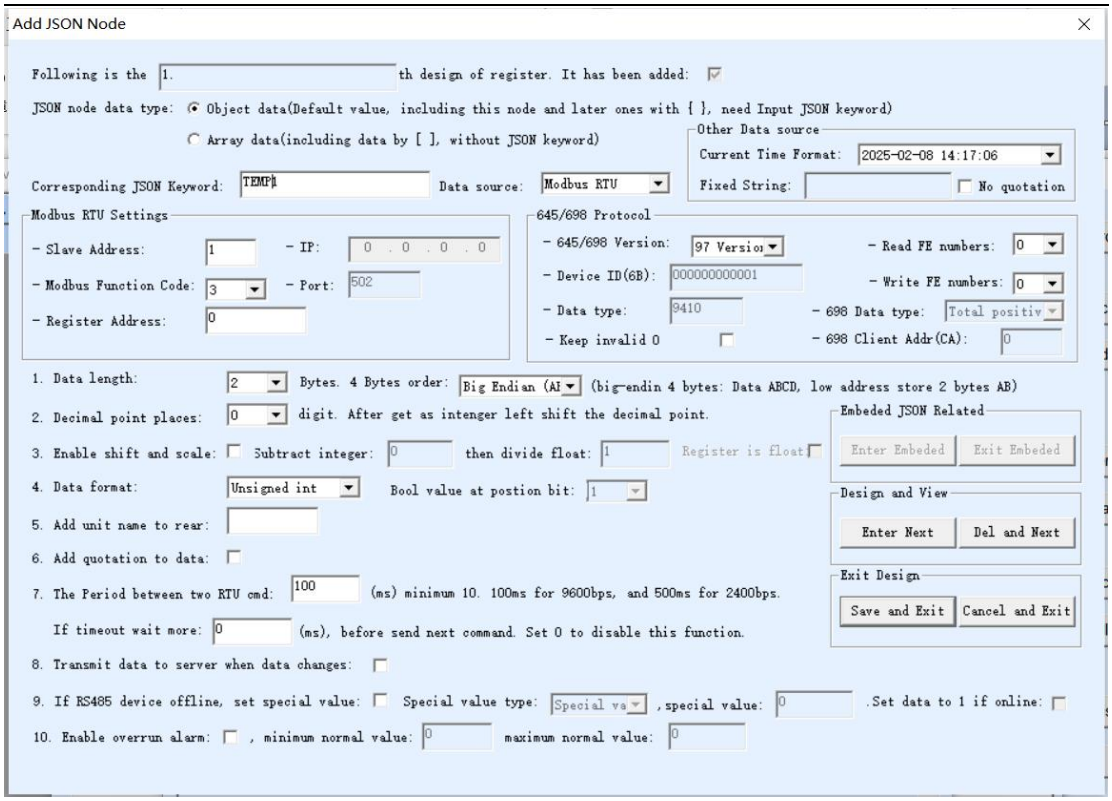


Figure 91JSON1 Upload Settings

JSON1 delivers, enabling commands to be delivered based on ID matching.

Keyword FS_VALUE:

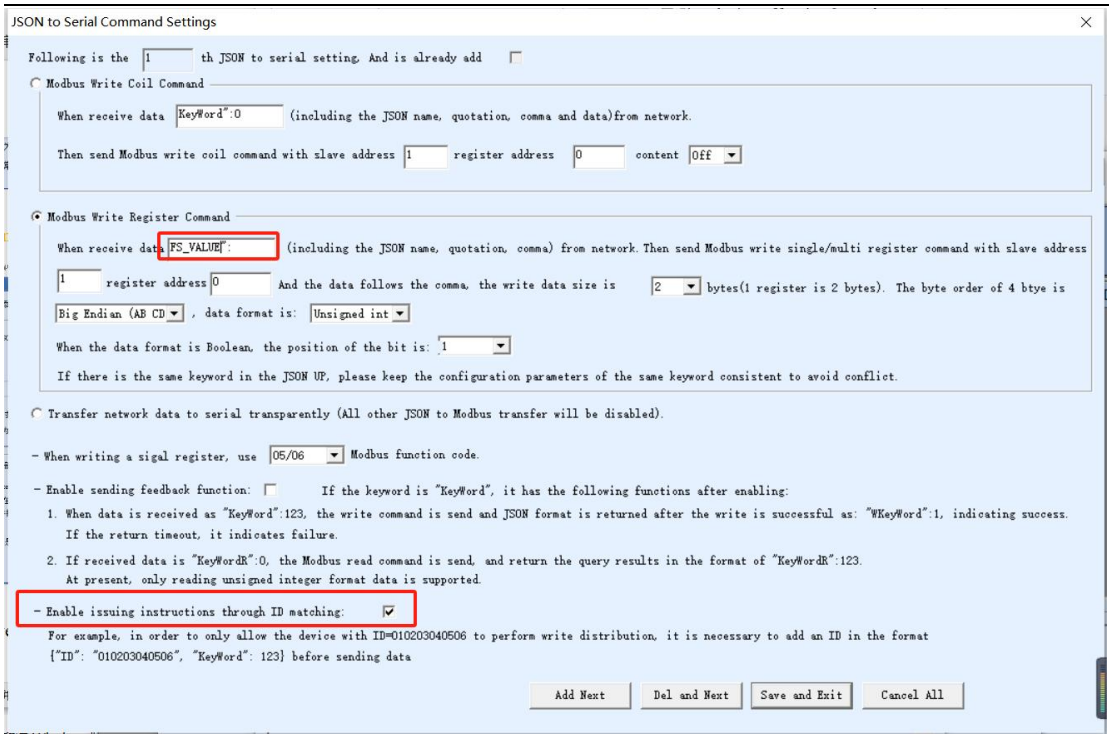


Figure 92JSON1 Delivery Settings

After setting, click Save JSON Settings.

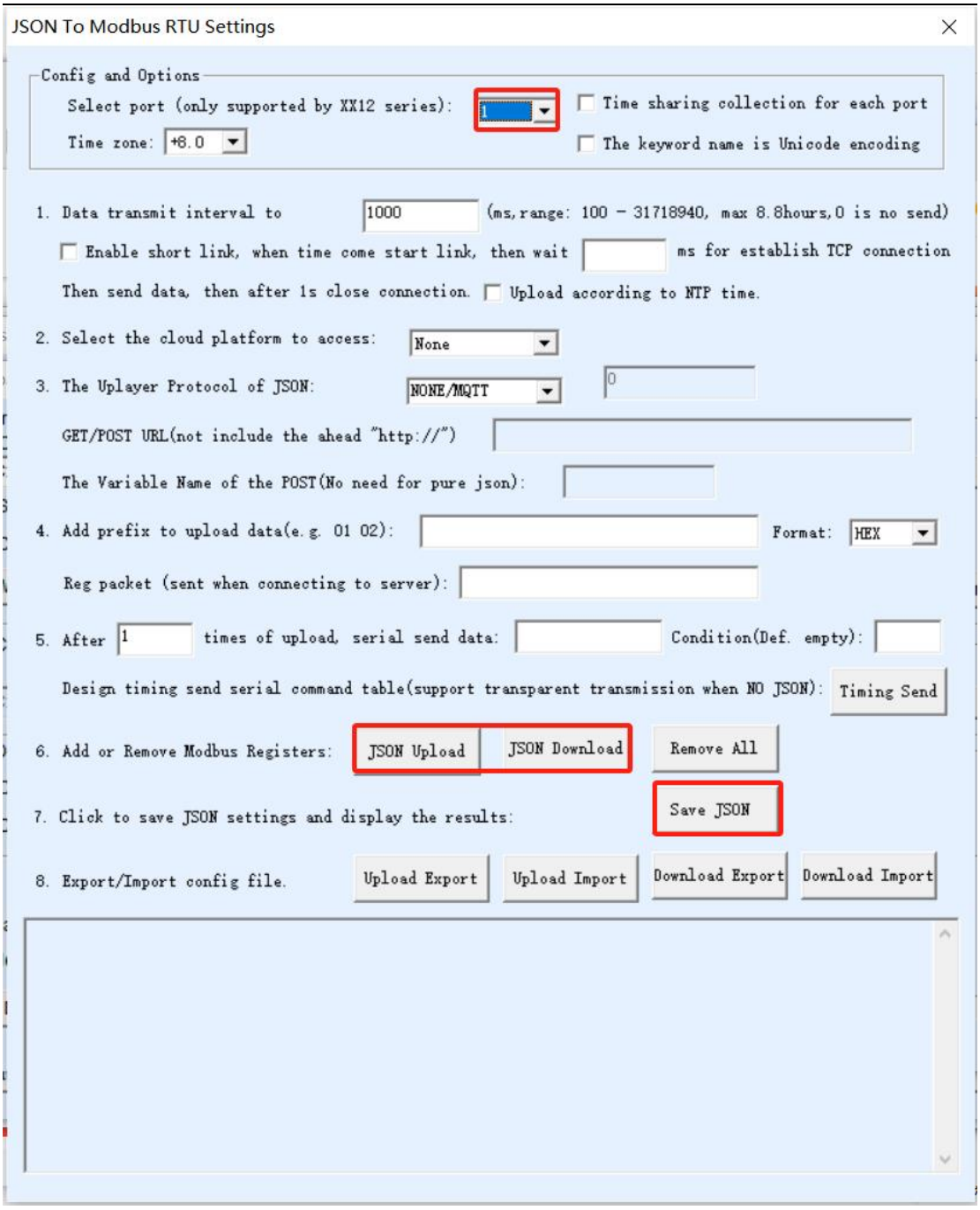


Figure 93JSON1 Settings

JSON2 Settings need to collect port selection 2, and then change the keyword in the JSON to TEMP2, the rest is consistent with the JSON1 Settings.

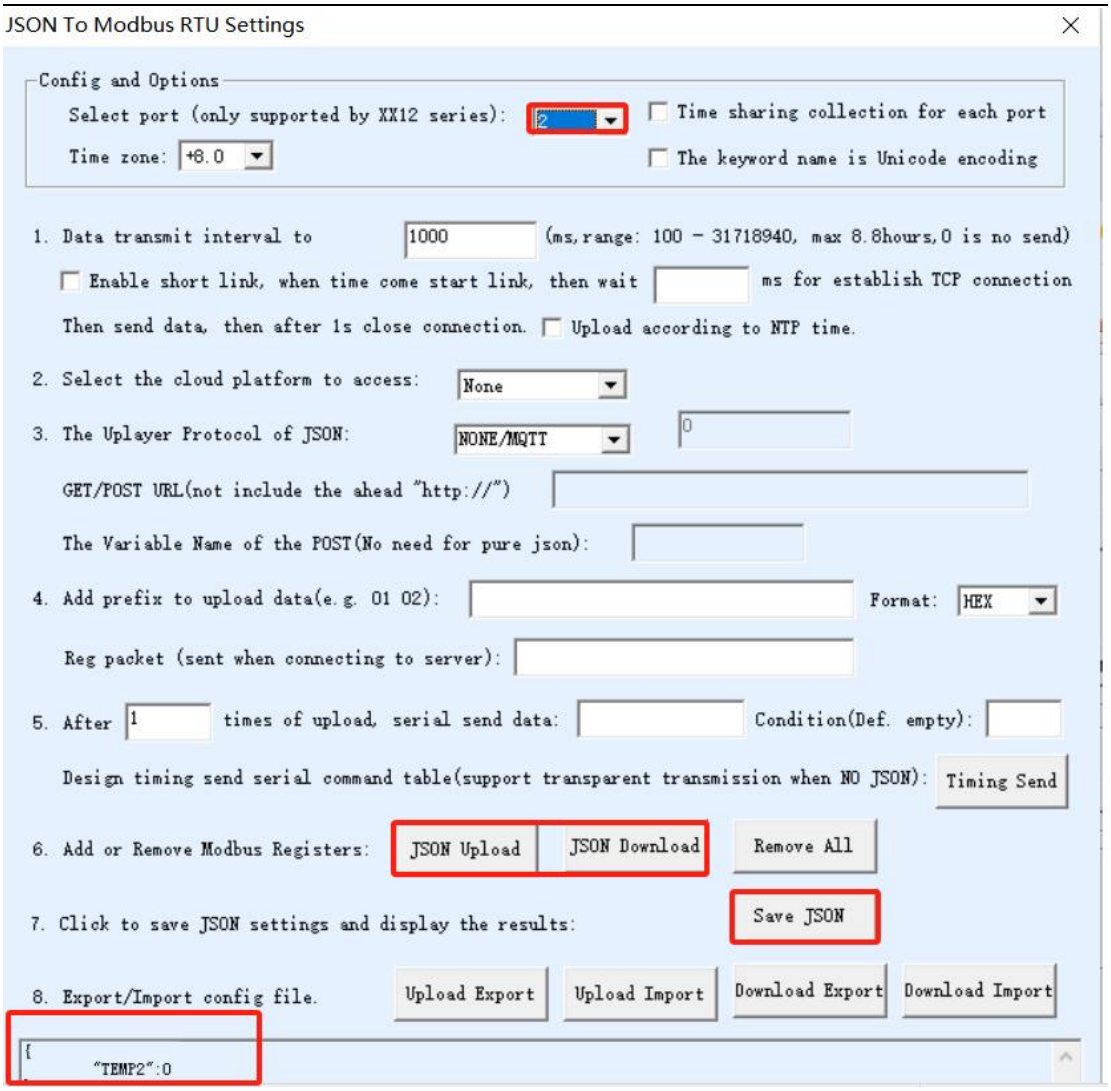


Figure 94JSON2 Settings

Finally, download the edited web file to the device. Note that MQTT and JSON will only work for ports that have been edited and saved, and mqtt and json for the rest of the ports will not work.

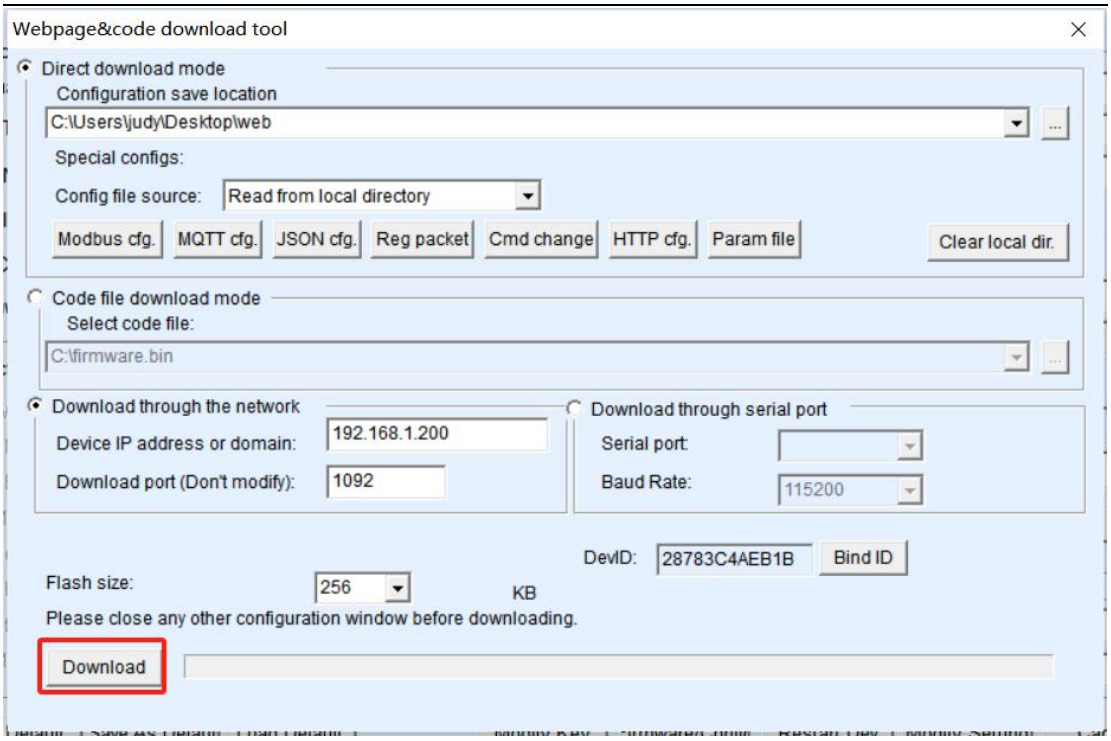


Figure 95 Download

MQTT/JSON TEST

Through the mqtt background, you can see that port1 and port2 just set have been connected to MQTT, and the device ID is the ID of the mqttid+ device edited by yourself in the mqtt parameter, as shown in the following figure: mqttid28649C1CCA1C, mqttid28649C1CCA1D.

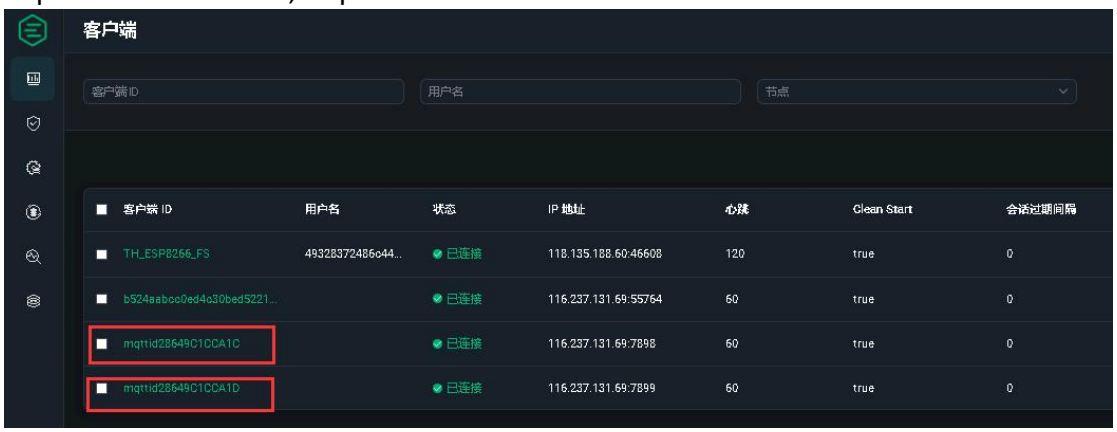


Figure 96 Client ID

As shown in the figure below, the MQTT server background client can see that

the device is indeed subscribed to three topics

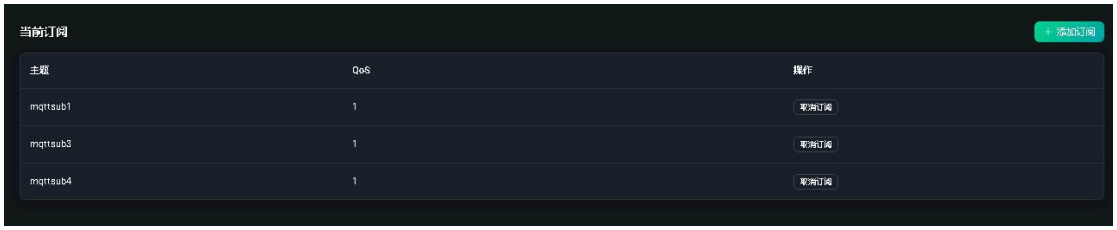


Figure 97 Client subscription topics

When sending JSON, subscribe MQTT1's mqttsub28649C1CCA1C topic and MQTT2mqttsub28649C1CCA1D topic through MQTT, you can see that the device can send normally.

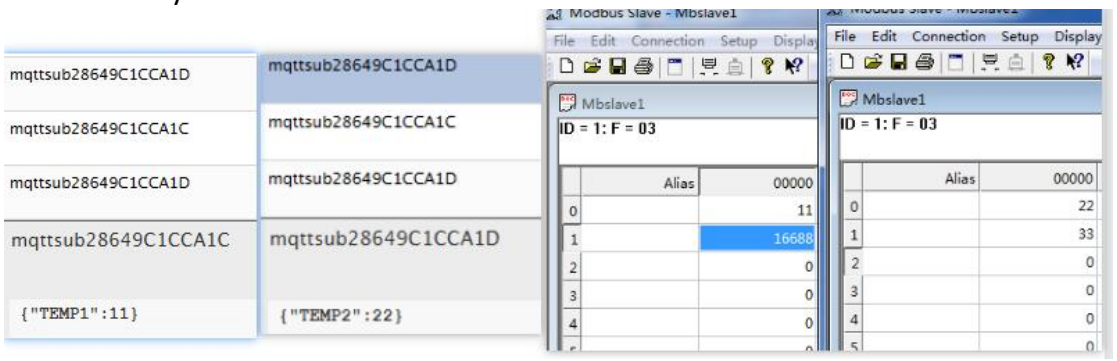


Figure 98 Publish topics

When the JSON is delivered. As shown in the following figure, when the theme is set to mqttsub4. Note that this is subscribed to both MQTT1 and MQTT2. When {" ID":"28649C1CCA1C","FS_VALUE":123} is sent, only the MQTT device with the corresponding ID will send MODBUS data.

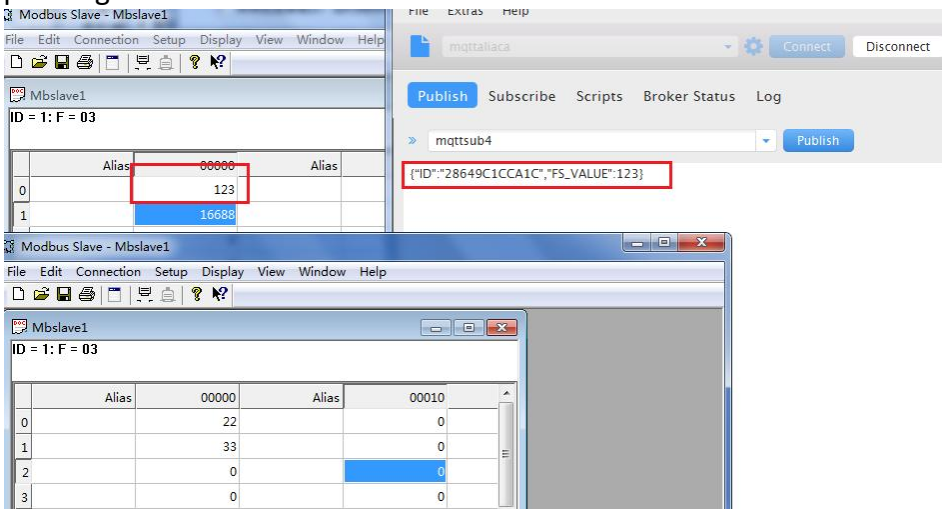


Figure 99 Subscribe to topics