RS485 TO ETH (B)
MQTT and JSON Manual

CONTENTS

1. OVERVIEW

MQTT and JSON can be used alone or together. JSON supports Modbusconvert RTU format to JSON format.

The Main FEATURES:

1. Use the MQTT-based protocol to establish a connection with the server, and use the form of subscription to publish data communication.
2. Support independent design and automatic collection of Modbus RTU registers.
3. Support the conversion of specific Modbus register content into JSON format and send it regularly and actively.
4. Support adding device ID, time, and any string in JSON format.
5. Support embeddings in JSON format.
6. Support NTP protocol, get the time automatically.
7. Support unsigned data and signed data, support decimal point representation, and support 4-byte length data.
8. All configurations can be completed in interface configuration, and the user's independent configuration does not need to be customized.
9. In addition to choosing MQTT, the protocol can support HTTP POST and GET methods.

2. JSON SIMPLE EXAMPLE

2.1. MODBUS RTU TO JSON

Modbus RTU to JSON can realize automatic collection of Modbus RTU tables, and is automatically uploaded to the cloud server following the JSON format .

Here we explain this usage through a specific case.

2.2. MODBUS TABLE

Suppose there is a Modbus table with a function code of 3 and an address of 1. Its register addresses and parameter names are as follows. Where the byte length is 4, it means that 2 registers need to be read continuously.

Register address	Parameter name	Byte length	Remarks
0	Current total active energy	4	Unsigned,keep 2 decimal places
97	Phase A voltage	2	Unsigned, 1 decimal place is reserved
98	Phase B voltage	2	
99	Phase C voltage	2	
100	Phase A current	2	Unsigned, 2 decimal place is reserved
101	Phase B current	2	
102	Phase C current	2	
119	Frequency	2	
356	A phase active power	4	Unsigned, 3 decimal place is reserved
358	B phase active power	4	
360	C phase active power	4	
362	Total active power	4	

The so-called signed means that the highest bit of 2 bytes or 4 bytes is the sign bit, for example, 0xFFFF will be recognized as -1. Keeping 2 decimal places means that after the data is converted as an integer, the decimal point moves from the rightmost to the left 2 digits.

2.3. DEVICE CONFIGURATION

We configure the device as a client.

Use the serial port tool to monitor a TCP server on port 1883 of the local computer.

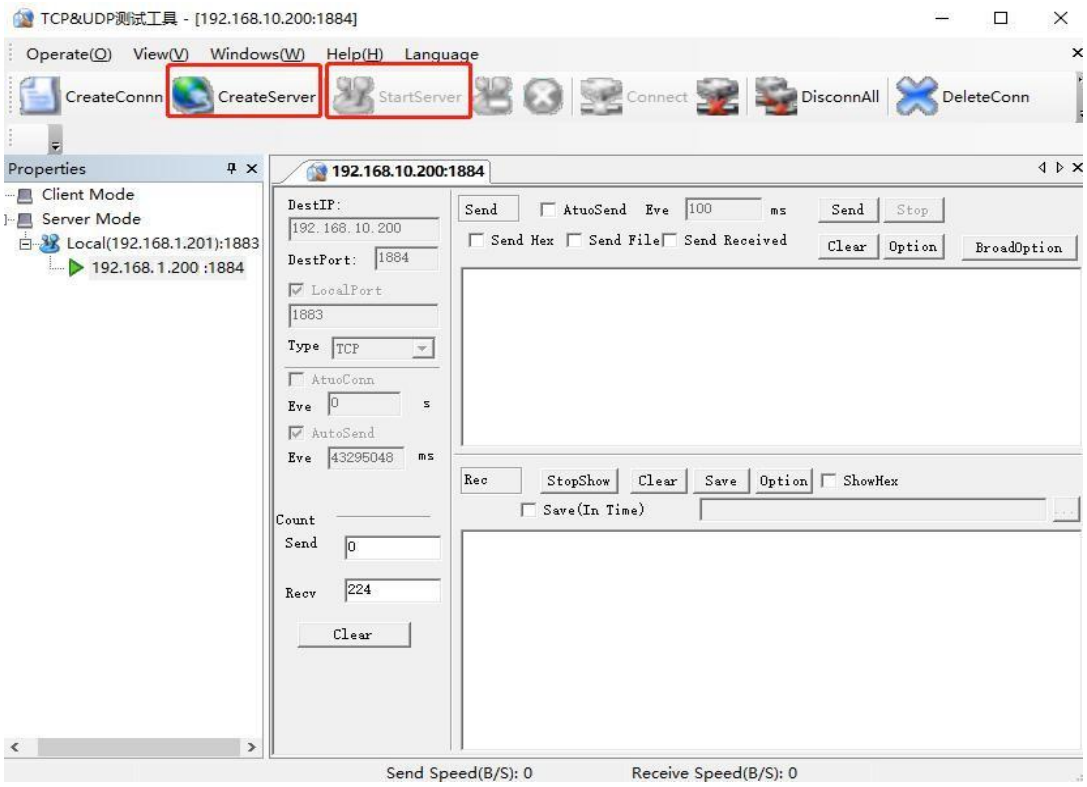

Figure 1 Socket simulated server receiving data

Use Vircom to configure the device.

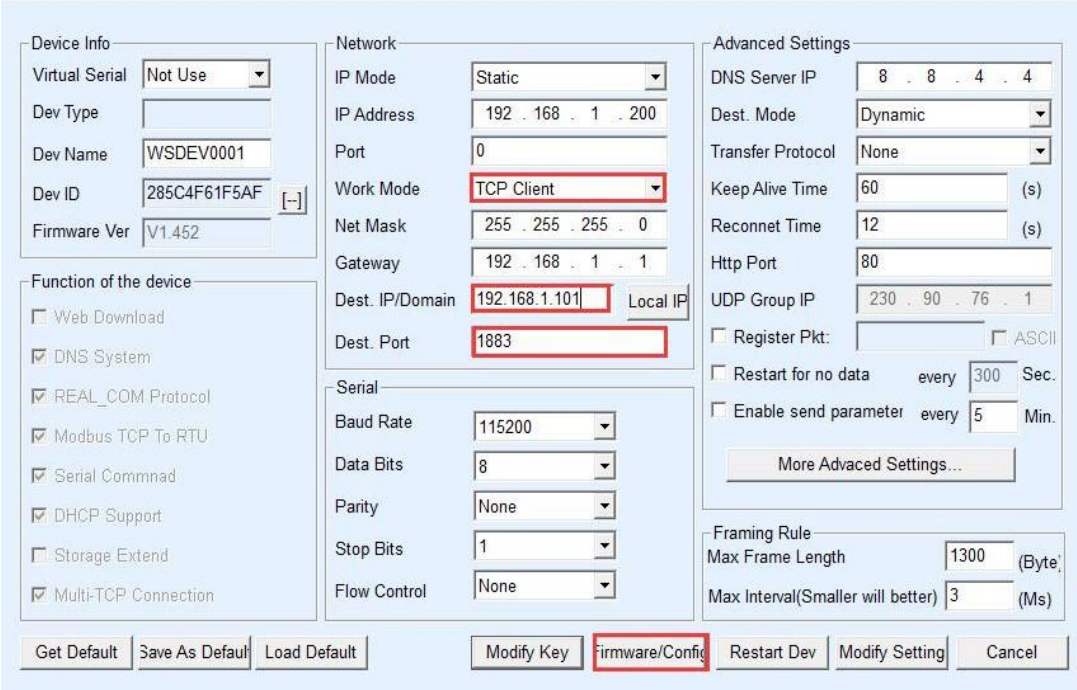

Figure 2 Device configuration

Click Modify configuration to connect the device to the SocketDlgTest tool. Enter device editing again dialog box. Click the "Firmware and Configuration" button.

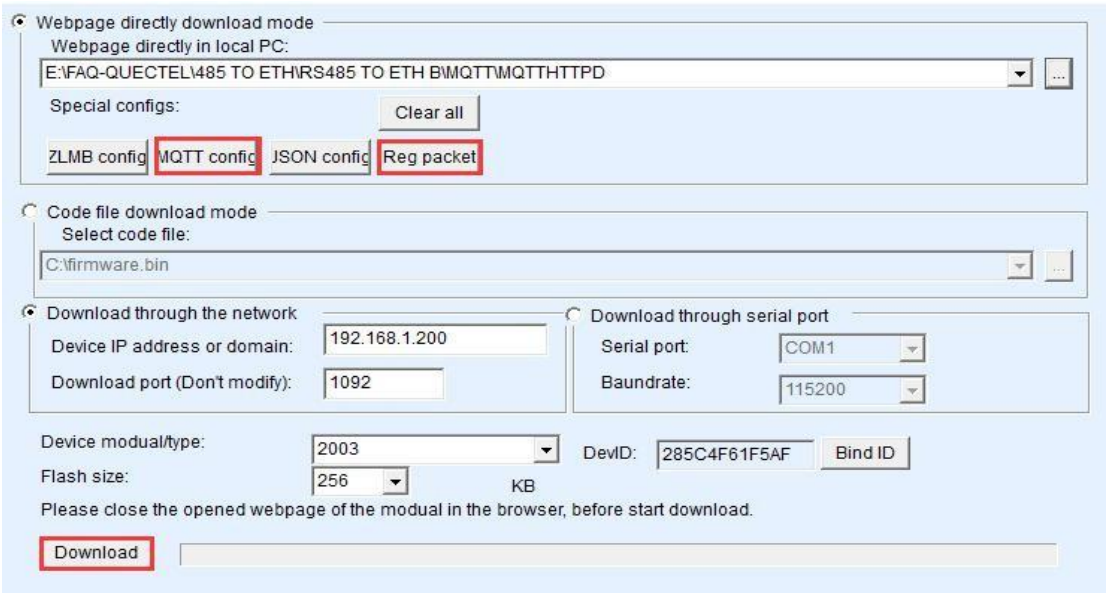

Figure 3 Download interface

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

Click the "JSON Configuration" button.

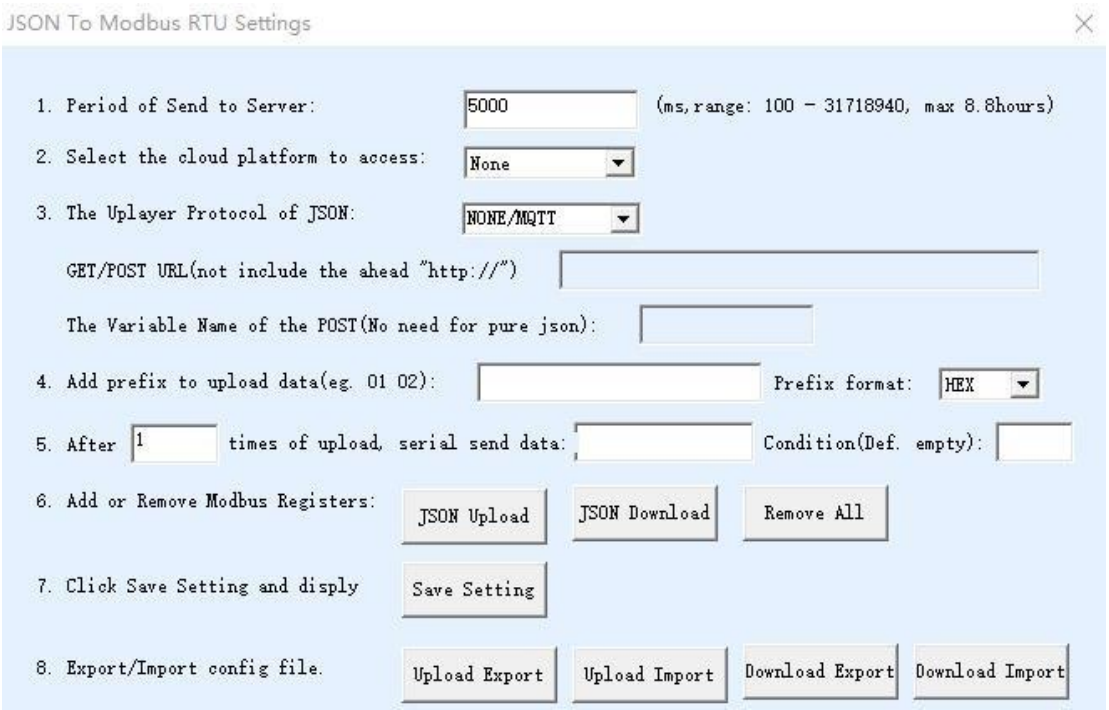

Figure 4 JSON configuration main interface

The parameters here are as follows:

1. Server upload time: The default JSON data is sent to the server every time. The server is the destination IP set in the device configuration interface just now, and the unit is milliseconds.
2. Add/View: After clicking, you can design every JSON node, or you can view the currently designed content.
3. Delete all: Delete all Modbus registers designed by the "Add/View" button to facilitate the restart of the design.
4. Save JSON settings: After the design is completed, only click this button to save the data to the download directory just now, and then download it to the inside of the device.

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

Register address	Parameter name	Byte length	Remarks
0	Current total active energy	4	Unsigned,keep 2 decimal places

The corresponding configuration is as follows:

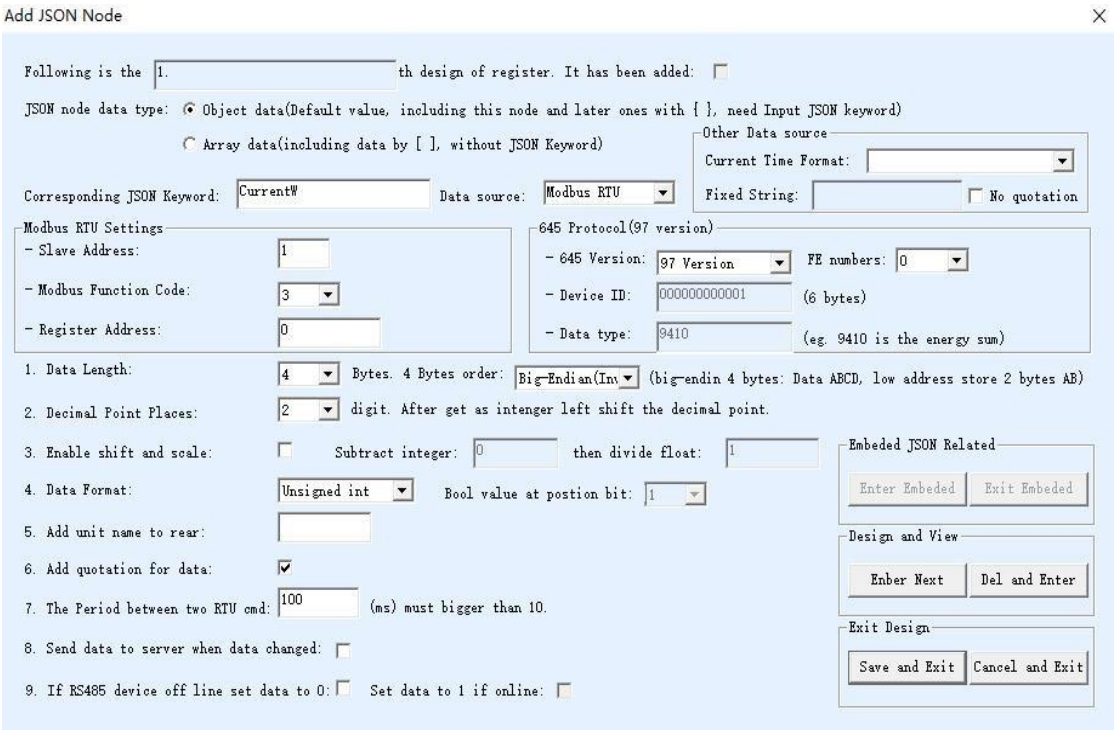

Figure 5 Register settings

The parameters here are as follows:

1. The figure below shows the first JSON keyword: "1." here means the first few JSON keywords of the current design interface, and the second one is "2.". If it is a embedding node under the second JSON, that is "2.1", and so on.

2. Already added: If it is checked, it means it has been added. When viewing the configured information, a check will appear, indicating that it is in the editing state. If there is an unchecked one, it is in the added state.

3. Corresponding JSON keyword: the name of this JSON node.

4. Data source: select the source of JSON data.

a) Modbus RTU: For example, in the form of CurrentW: 123.45, it means that the data comes from a certain Modbus RTU table and is collected through the serial port. The left half of the figure is related to the design of Modbus RTU parameters.

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 fixed string of JSON nodes can be generated.

c) Device ID: If the JSON node name is DevID, the string of the sent node is DevID: "285301020304", where "285301020304" is the MAC address or unique number of the device.

d) Current time: If the JSON node name is ColletTime, the uploaded string is ColletTime: "2019-05-13 22:23:31". The time is obtained by the system through the NTP protocol.

e) Embedding JSON: If the node name is Alarm, the format of its upload has

Alarm: {temp1: "25.1", temp2: "26.2"}, that is, the content of Alarm is still a JSON collection.

5. Modbus related settings

a) Slave address: Modbus table address.

b) Modbus function code: currently supports 03 and 04 function codes.

c) Register address: 0.

d) Data length: 4 bytes.

e) Data format: an unsigned int.

f) Keep the decimal point: 2 digits are reserved here.

g) Add the unit after the data: For example, when "CurrentW": 25.6W, the W behind 25.6 is the large unit to increase. Write "W" into this box.

h) Add quotation marks to the data: if checked, change "CurrentW": 25.6W to "CurrentW": "25.6W" form.

i) Serial port polling time: set it to 100ms. Refers to the polling of this register and the next register instead of the polling interval for this command.

6. Fixed string: When the source is selected as a fixed string, you can enter the content of the string.

7. Button

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 it is currently "2.", it will enter the design of the node "2.1" .

b) Return to the previous level: If the current node is embedded at the Nth level, clicking thisbutton will return to the design of the N-1 level node and stay on a new node at the N-1 level.

c) Design the next one: Click to enter the next local JSON node. If there is no next node in the previous design, the "Already Added" checkbox will be canceled, indicating that it is a new node.

d) Save the design: After completing the design, click "Save Design" at the last design node interface. Then return to the main interface, and then click "Save JSON Configuration".

e) Cancel design: cancel all current designs, if you are viewing the design content, you can click this button to exit.

Here, click the "Design Next" button to continue designing other registers in the Modbus table. After all the registers in the form are designed, click "Finish Design", and then click "Save JSON Configuration" to exit. Then click the "download button" on the "download web" page.

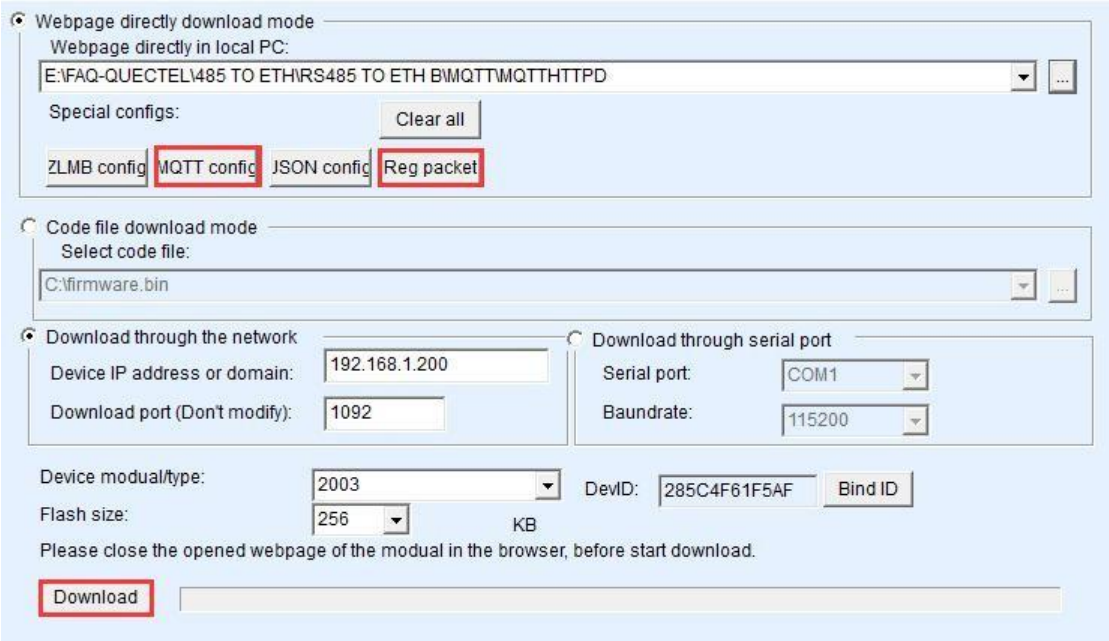

Figure 6 Download

Then click "OK" and the device will automatically restart. If there is no restart, please restart manually.

2.4. CREATE A NEW MODBUS ANALOG METER

Here, Modbus Slave is used to simulate a list.

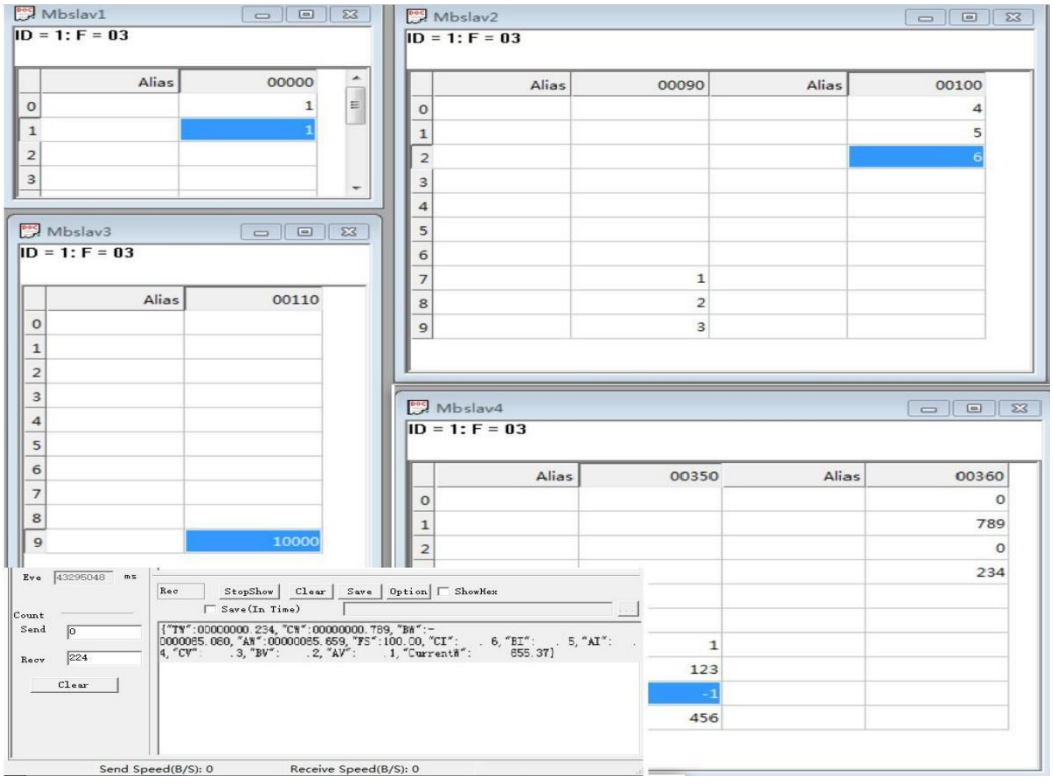

Figure 7 Test results

The test result shows that the meter simulated by the Modbus slave tool can be collected by the gateway. at the same time,it can be sent to the server software simulated by SocketDlgTest according to the json format regularly.

3. JSON complex example

3.1. ENABLE NTP

In order to be able to use JSON with time, you must first enable the NTP function of the device. The NTP function can get the current time through the network.

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

[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, NTP_SERVER3 are the IP of the NTP time server Address, fill in according to the actual situation. Up to 3 servers can be set up, but you must

NTP_SERVER1 starts to write, if there is only one, write NTP_SERVER1, if there are only two, write

NTP_SERVER1 and NTP_SERVER2.

After saving, ntp.txt and other files are downloaded to the inside of the device.

3.2. NESTED JSON DESIGN

Suppose we need to design the following JSON:

```
{"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. The keyword in step 1. is "header", then select "JSON Embedding" as the source, and thenclick the "Design Nested JSON" button.
2. Enter step 1.1, here design "DEVID": "285301020304", enter the keyword as DEVID, select "Device ID" as the source, and click "Design next".
3. Enter step 1.2, here design "time": "2019-05-13 22:23:31", enter the key word as time, select "current time" as the source, it should be noted that although the first level has been designed Finished, but you still need to click "Design Next". After entering step 1.3, click "Return to the previous level". This 1.3 step will be automatically abandoned.
4. Go to step 2. Enter the keyword data here, then select "JSON Nesting" as the source, and then click the "Design Nested JSON" button.
5. Go to step 2.1, here design "id": "MyData123456", enter the keyword id, select the source as a fixed string, then enter "MyData123456" in the fixed string box, and click "Design next".
6. Go to step 2.2, enter the keyword alarm here, select "JSON Nesting" as the source, and then click the "Design Nested JSON" button.
7. Enter step 2.2.1 to design "alarm1": 123.4C, this is a Modbus data with unit, the function code is 3, the register is 0, then the design is shown in the figure:

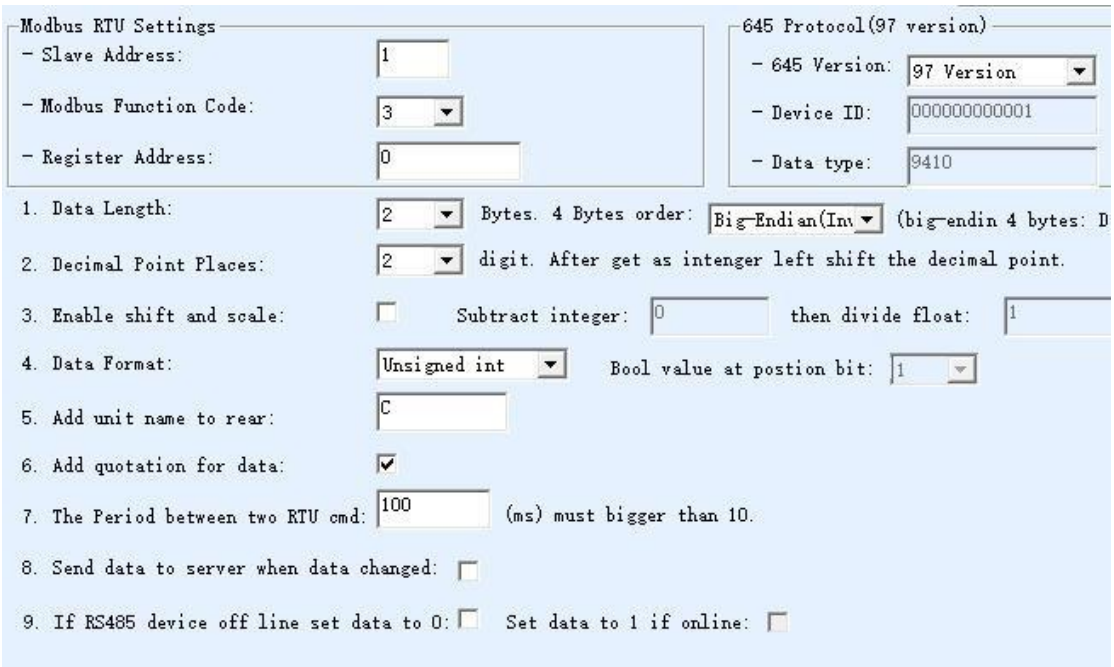

Figure 8 Register design

Then click "Design Next".

8. Go to step 2.2.2 to design "alarm2": The method is similar to alarm1, and the register address is set to 1. Similarly, first click "Design Next", and then in the "2.2.3" step, click "Return to the previous level".

9. Enter section 2.3, because 2.3 does not need to be designed, click "return to previous level" at this time. Because there is still "value": 2345 is not designed, otherwise you can directly "save the design".

10. Enter section 3. Here, design a Modbus data with the keyword value. Now click "Save Design". Note: If the "value": 2345 does not exist here, then you need to click save directly 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, you can use it at this time The new version of vircom "Delete and go to the next" to delete this useless node.

11. Back to the JSON to Modbus RTU setting interface, click "Save JSON Settings". Then download it to the inside of the device and use it.

    When the TCP connection is established, the 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 select the correct destination on the web download interface first.In addition, click the button in accordance with the design-time steps to fully browse all nodes.

3.3. READ THE BITS OF THE BYTE REGISTER

Sometimes the data read by Modbus using the 03/04 function code will also use bits to express specific meanings. For example, the register of the 00 00 address read by the 03 function code is 0x8183, then bit16, bit9, bit8, bit2 and bit1 are included. All are 1. These bits have different meanings. When they are 1, they indicate different alarms. So you also need to upload with different json keywords. Methods as below:

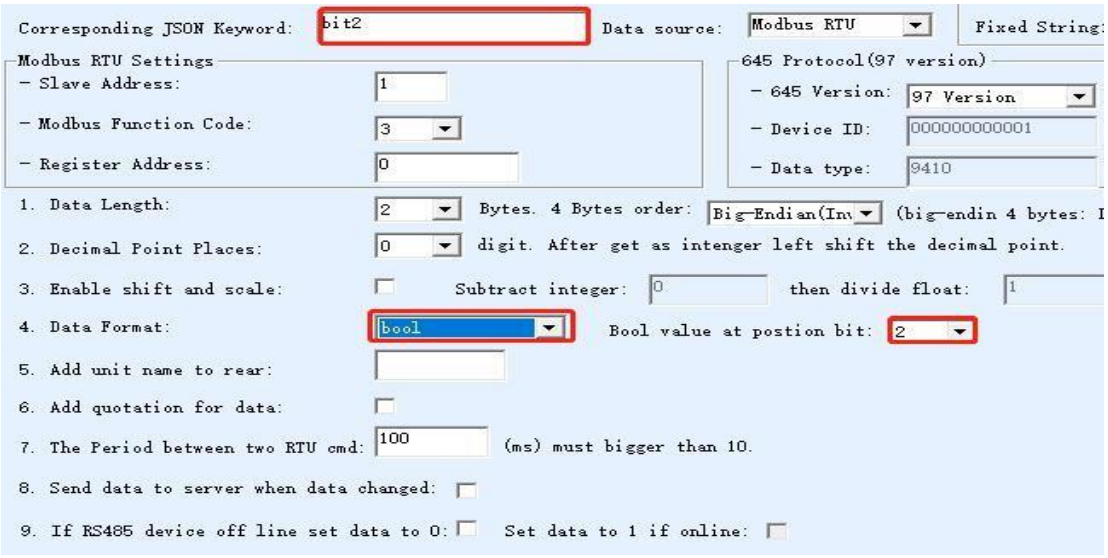

Figure 9 Byte register

The design method is basically the same as the previous method. The only thing to pay attention to is that the data format is selected as "Boolean", and then where the Boolean value is located, bit2 is the json variable in the 03 function code and the bit position in the 00 register. If the register value is 00 20, where 1 is in position 2, please note that you can set the same Modbus station address, function code, and register for different json keywords. Different variable contents can be obtained as long as the position of the Boolean value is different. For example, we designed bit1, bit2, bit8, bit9, bit16, when the register content is 0x8183, we get the following json data return: {"bit1":1,"bit2":1,"bit8":1,"bit9" :1,"bit16":1}

3.4. 01 AND 02 FUNCTION CODES

You can set the JSON node for the bit register of the 01/02 function code, but each JSON node needs to set the register address once, so the number of bits read each time is one bit. The difference with the bit of the byte register is: it is still read through the 03/04 function code, but only the value of one of the 2 bytes is taken. As only 1 bit is read at a time with 01/02 function code, the Boolean value is generally with 1, if it is 1, it will display '1', otherwiseit will display '0'.

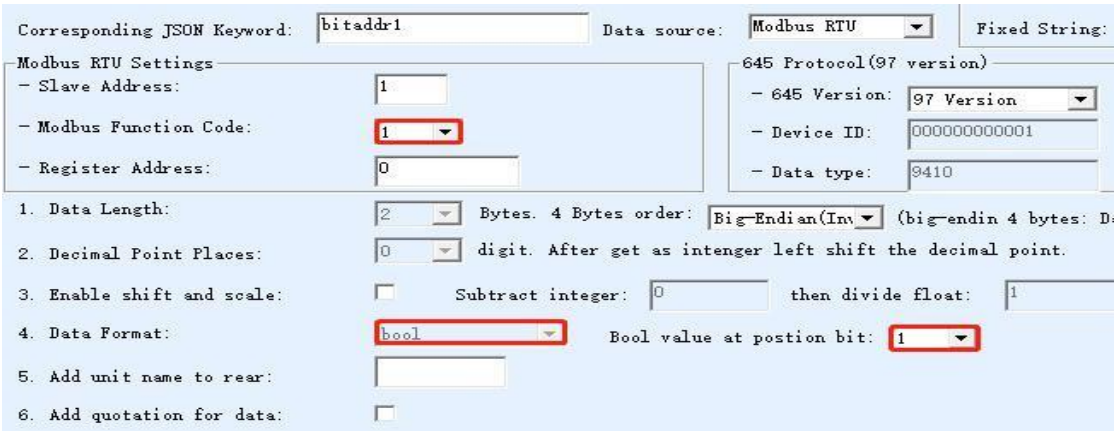

Figure 10 Byte register

When the Modbus function code is 01/02, the data length, data format, and the number of reserved decimal places cannot be selected.

3.5. SHOW DESIGN RESULTS

Now click "Save JSON Design" to design and you can see the content of the designed JSON format in the display box, which is convenient for overview of the design. At the same time, there is an index for comparison when entering "Add/View".

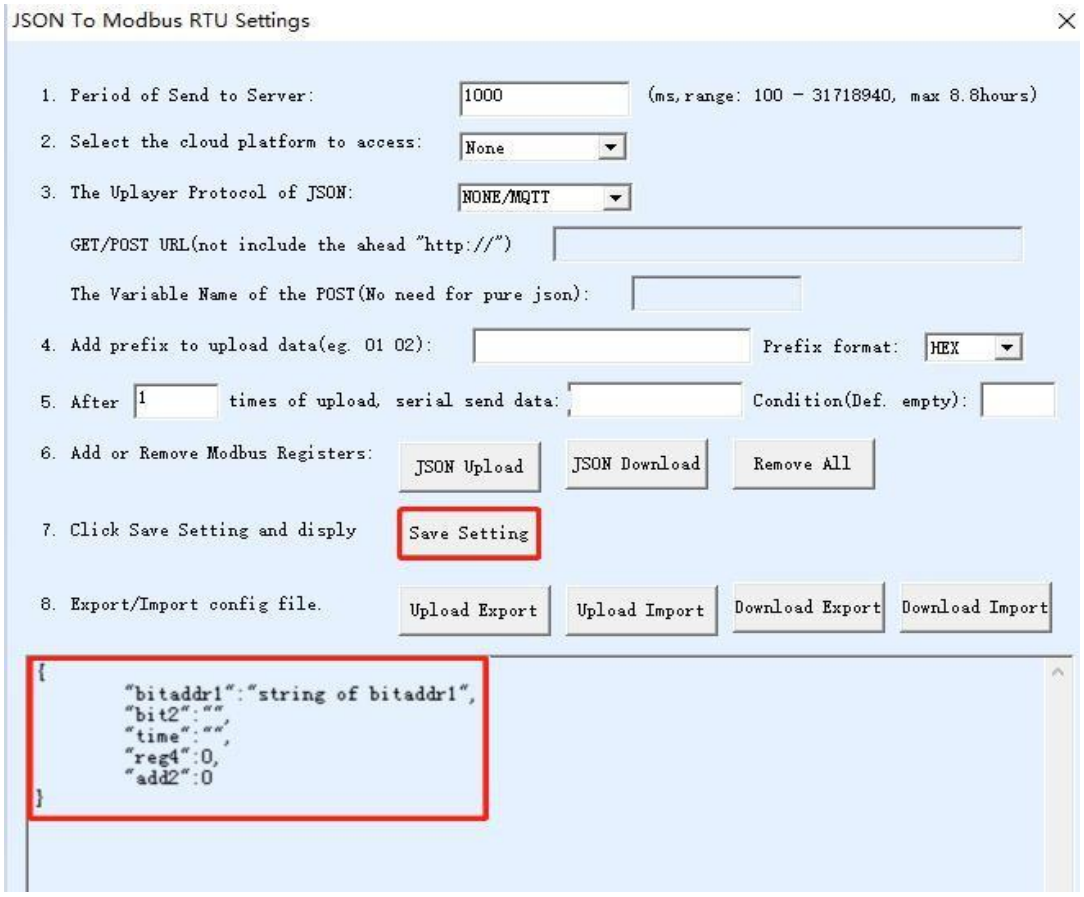

Figure 11 shows the results

3.6. EXCEL EDITING

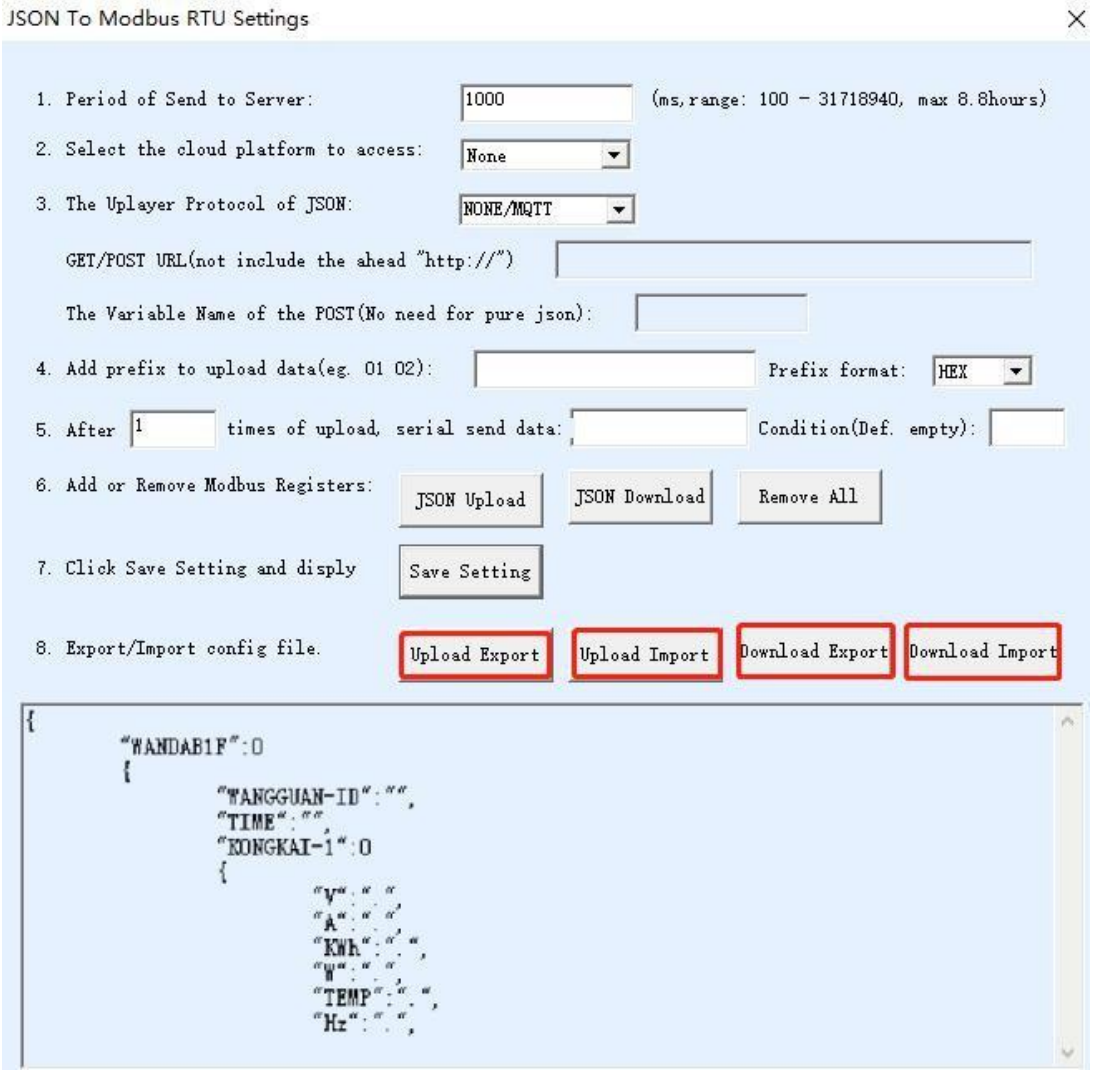

Figure 12 Import and export CSV format

In order to facilitate editing, you can export the content of the design to CSV format, then edit with EXCELL, then save it as CSV, and then import it.

However, CSV has formatting issues:

Num	JSON Keywords	ata Source	ixed strin	is slave ad	us function	is register ad	045 device ID	045 data	ta Form	byte ord	ite Form
1	CW	Modbus RTU		1	3	0	1	9410	2	0	gned int
2	BW	Modbus RTU		1	3	1	1	9410	2	0	gned int
3	BV	Modbus RTU		1	3	100	1	9410	2	0	gned int
4	AV	Modbus RTU		1	3	101	1	9410	2	0	gned int
5	CV	Modbus RTU		1	3	102	1	9410	2	0	gned int

Figure 13 Data format error

At this time, you can save the CSV as 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 design. Give a simple example:

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

In this example, there is only one JSON object reqBody, its content is an array, the first element of the array is data 0, and the second element is data 1. Here we treat the bracketed array similarly to nested JSON, but this nested JSON does not require keyword names and colons. The design steps are as follows:

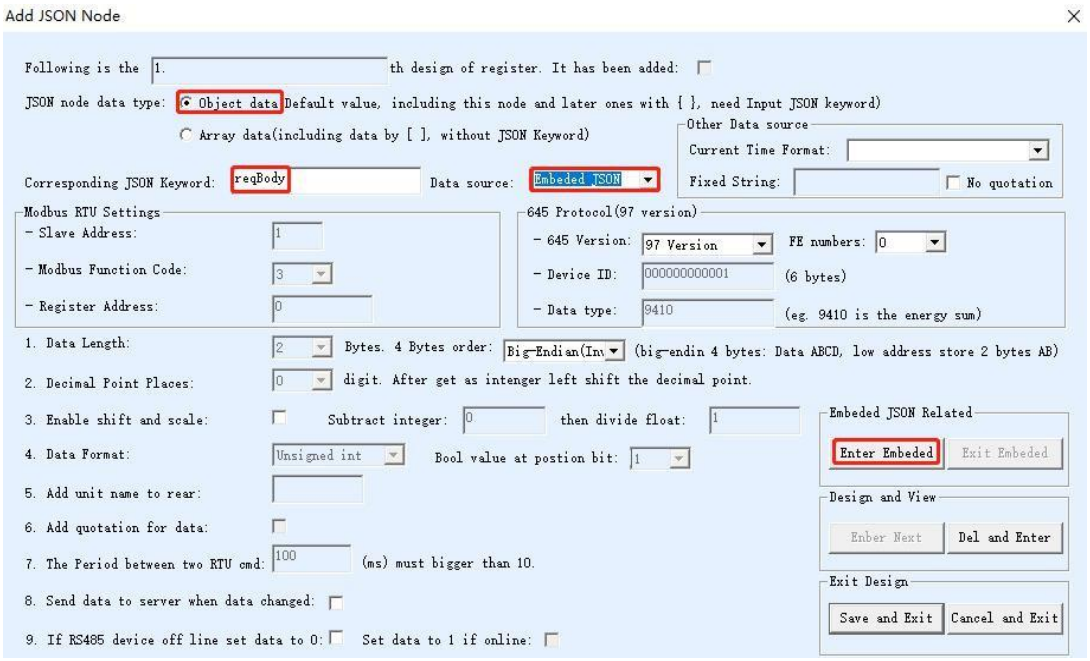

Figure 14 Array object

As reqBody is an object itself rather than an array unit, we should choose the node type as the object data instead of the array data. Hence, the reqBody data is from the “nested JSON”. Please click on “Design the nested JSON”.

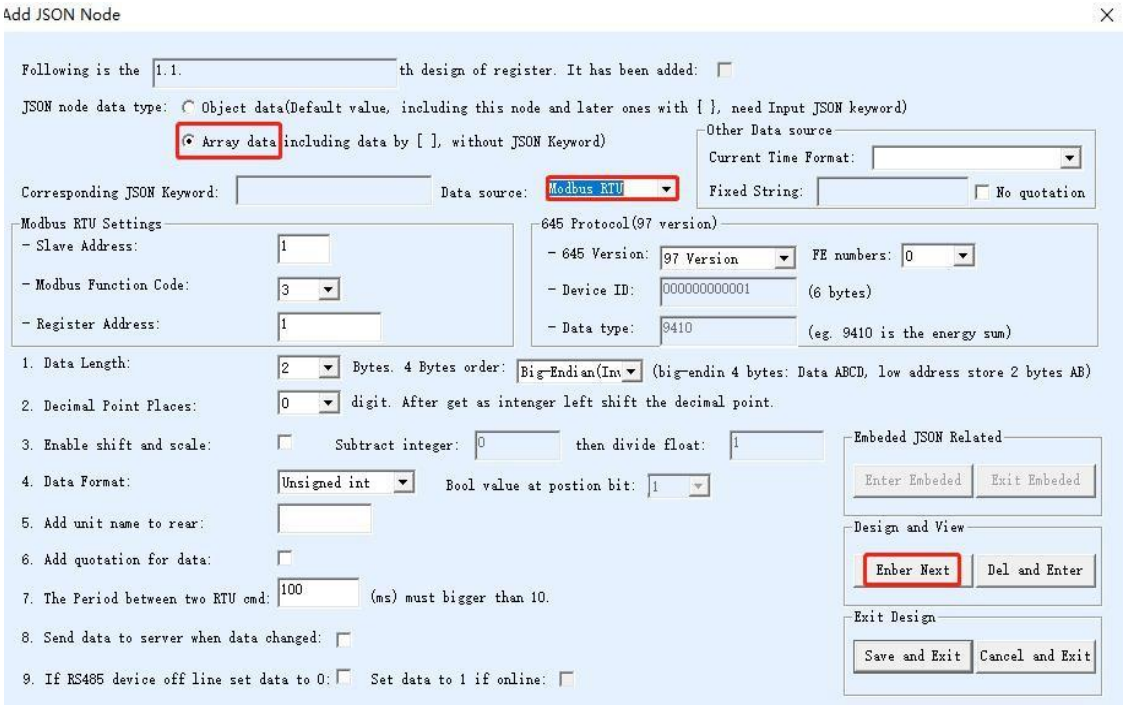

Figure 15 Array content

Next, design the first array content, and the node type is the array content.

The data source is Modbus RTU collection, fill in the relevant Modbus registers and other parameters. Then click "Go to Next" to design the second element of the array, and the method is similar. After designing the second element, since there is no more design, click "Save all and exit". Go back to the previous interface and click"Save JSON Settings", then download it. The data posted is: {"reqBody":[2,3]}.

Now look at a more complicated example:

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

]

}

The content of the array here is not a simple data value, it is a JSON itself.

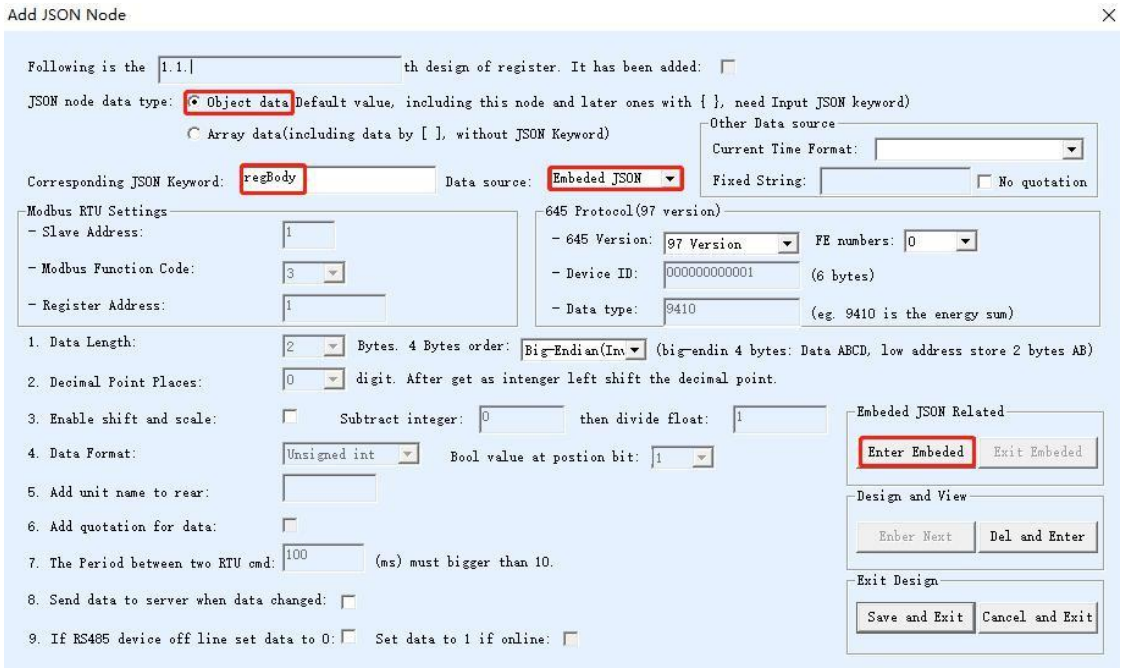

Figure 16 Step 1

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 "Enter Embedded".

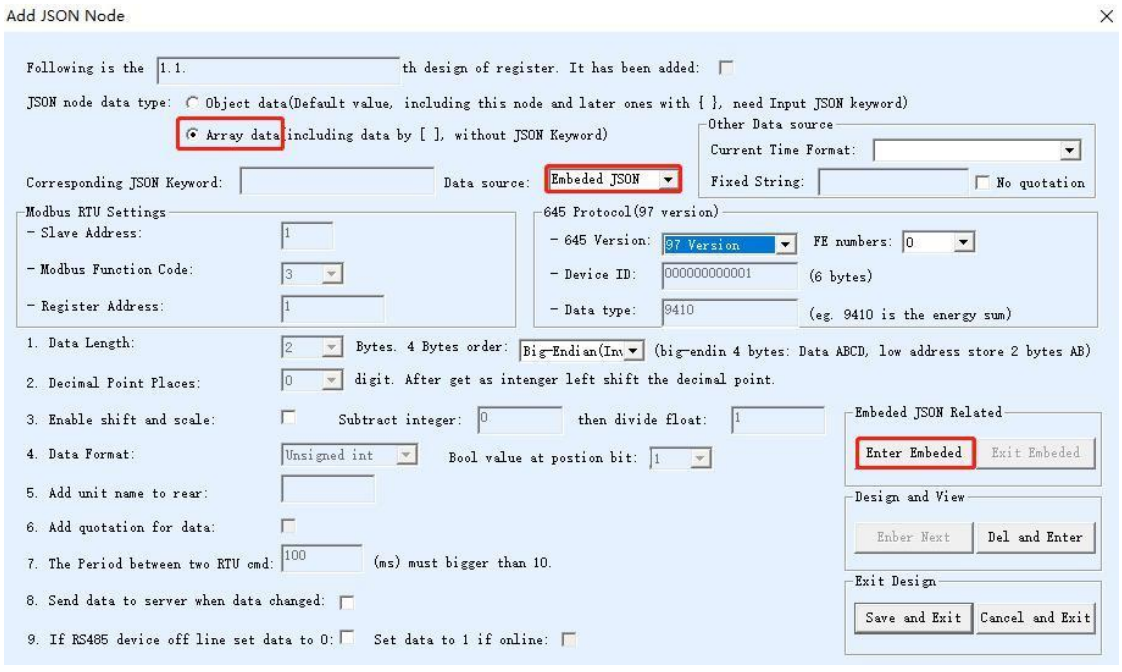

Figure 17 Step 1.1

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.

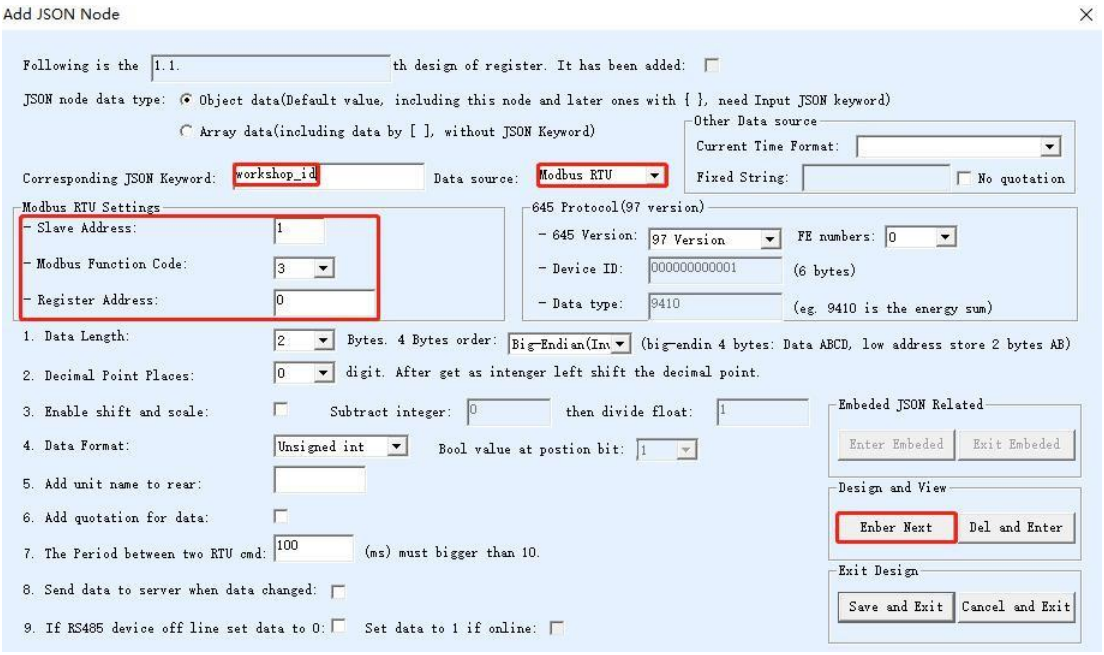

Figure 18 Step 1.1.1

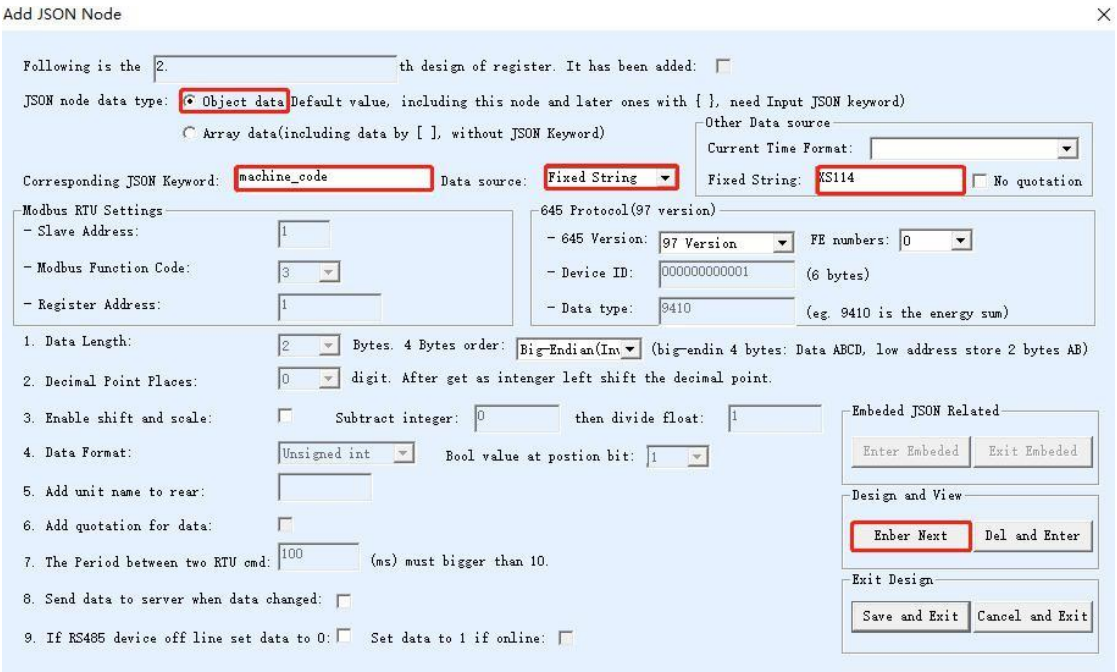

Figure 19 Step 1.1.2

Note here that you need to click "Enter Next", in the empty content node (that is, 1.1.3), click "Return to the previous level" to enter "1.2". 1.1.3 here is actually a non-existentnode.

"1.2" is also an array type, and the data source is nested JSON, please refer to step 1.1. After that until "1.2.2", click "Save all and exit" directly.

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

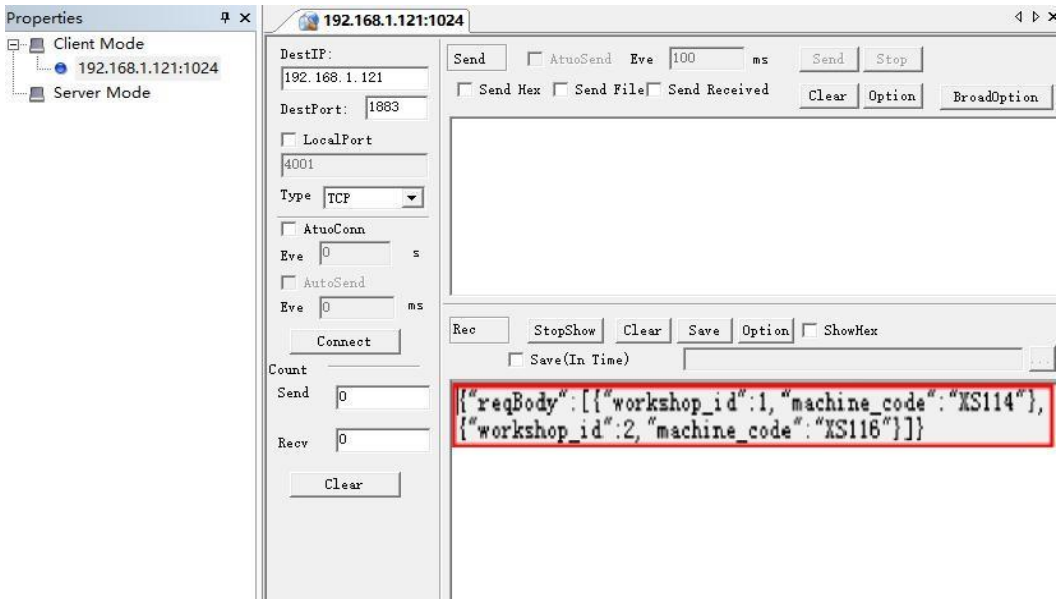

Figure 21 Step 1.1.3

3.8. CANNOT READ AND CLEAR

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

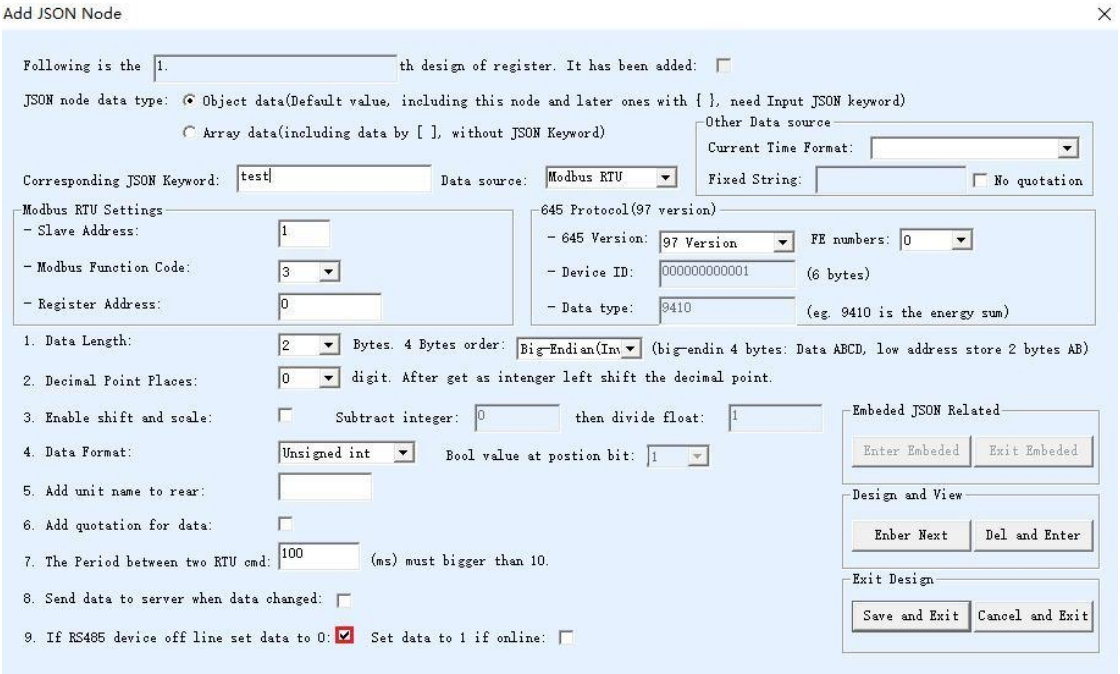

Figure 22 Clear data

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 content. Once the device is offline or the data cannot be read, it will be uploaded as {"test":0}. This way can avoid saying that the data still exists after the device is offline, which can give people a misunderstanding.

3.9. DEVICE OFFLINE

If the data cannot be read, the data is 0. In some cases, it is impossible to judge that the device is offline. For example, if the data content itself is 0, it is impossible to judge whether the device is offline or the data is 0. In addition, sometimes the device has multiple registers that need to be read, so a separate JSON keyword should be used, such as {online1:0} as 0 or 1 to indicate that the device is online. For this, you can design online1 as follows. First design all the registers, and then add a JSON node:

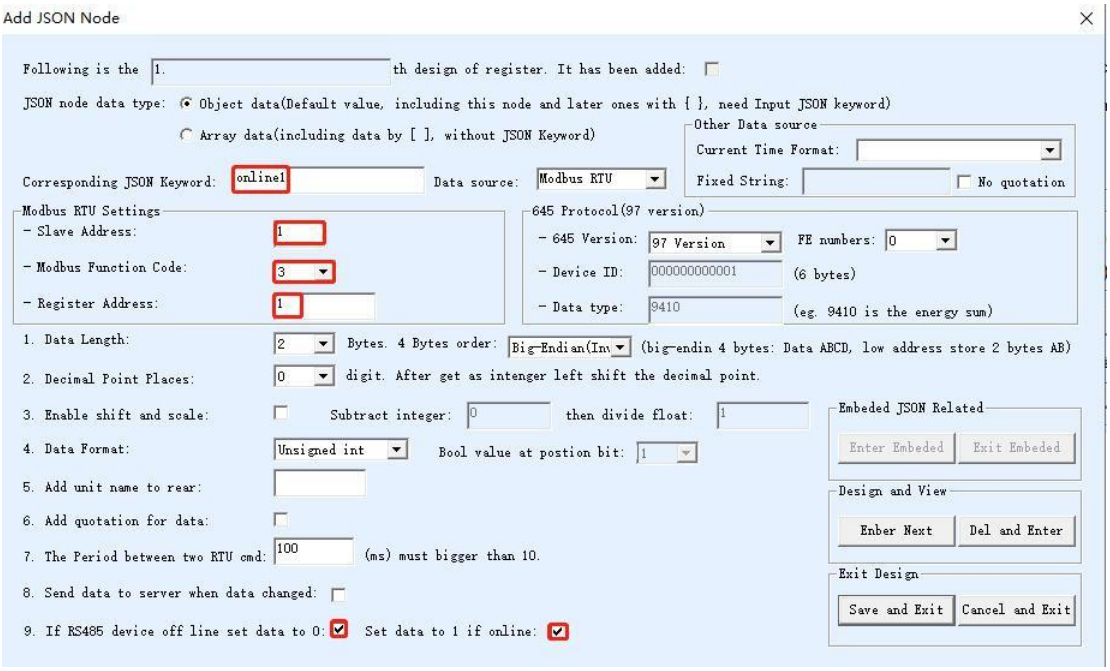

Figure 23 Device online

The name here is online1, which can be taken at will. The register number can select an arbitrary existing register. The most important thing is to check the Rs485 device offline data clearing and if the device is online, regardless of the register content, it is forcibly set to 1. In this way, even if the data read is 0, it will be 1 as long as the data read. That is, online1 has only two data, 0 and 1. 1 means online, 0 means offline.

3.10. PAN AND ZOOM

Pan and zoom are the data content read by Modbus register minus a 2-byte integer, and then divide by a single-precision floating-point number to get a single-precision floating-point number, as shown in the figure below:

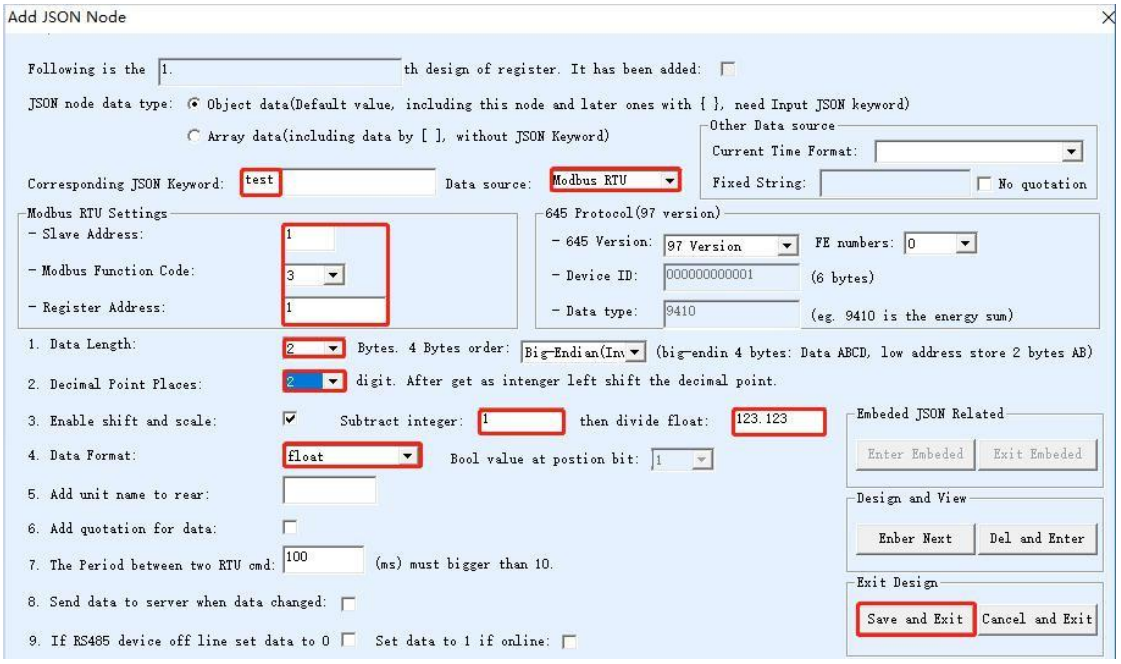

Figure 24 Pan and zoom

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

Note that this only supports Modbus RTU data source, the data length is 2, 03 or 04 function code and the floating point output. The number of decimal points can be selected from 0 to 4. The most important thing is to check the option to enable pan and zoom. Enter an integer in the minus integer, the range is -32768 to 32767. Enter a floating-point number in the single-precision floating-point number.

3.11. DATA CHANGE REPORTING

In some cases, in order to reduce data traffic, it is not necessary to upload data frequently. Only need to upload the collected data when there is a change. The actual method is to set the upload time period to be very large, the maximum is 8.8 hours. From the traffic point of view, this is equivalent to not uploading data at this time, but if you check the data change upload, every time a node data changes, it will trigger the upload of all data. The data change here can be configured for each JSON node, and can be used to

trigger upload after the device is offline (at this time, the data content changes so it can also be uploaded). As a typical case, we collect DI status and upload it when it changes.

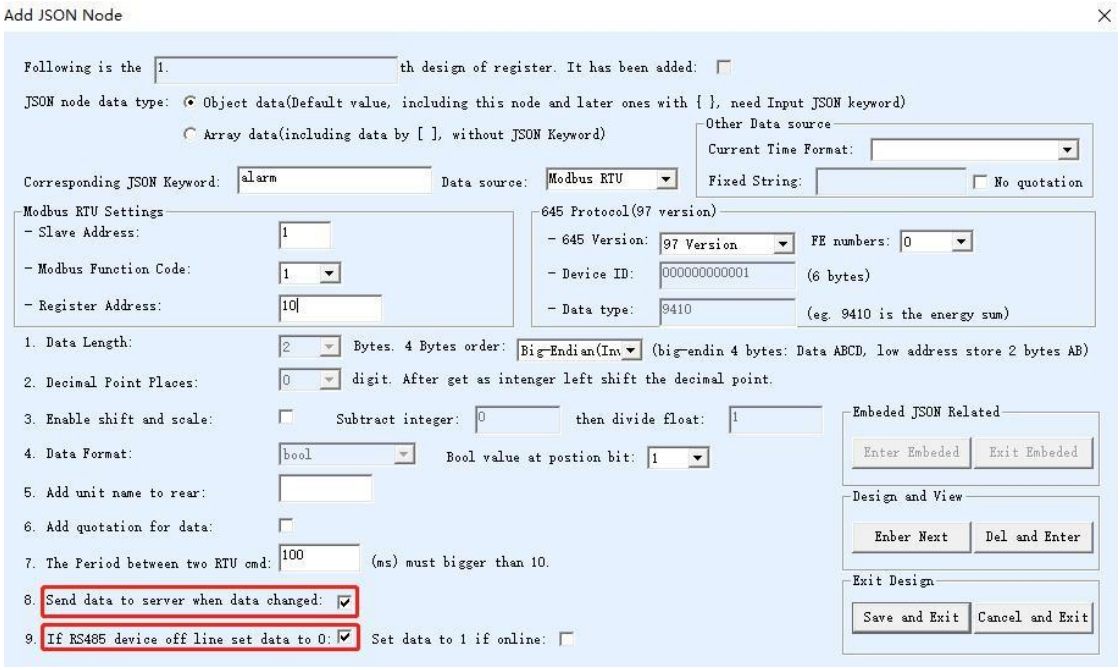

Figure 25 Data change upload

Here you need to check “Send data to server when data changed”. “If the RS485 device off line set data to 0” is checked, it will report when the device is offline. However, this offline report is limited to the original data value of 1. If any offline report is required, a separate JSON nodeneeds to be added. Please refer to the introduction in the "Device Offline" section.

In addition, add another node of alarm2 with address 11, but uncheck the data change upload.

Normal data is uploaded once in 10 seconds. If the bit of address 10 changes (alarm) at this time, the report will be triggered immediately (actually, it takes several hundred milliseconds for the data to change due to the rotation, and the upload will be slightly delayed). But if the bit of address 11 changes, the report will not be triggered because the change report option is not checked.

When the alarm is 1, the report will be triggered immediately if the device is offline.

3.12. JSON Download

JSON download realizes the matching of a single JSON keyword and the output of the corresponding Modubs command. This does not require that the issued string matches exactly, but as long as there are corresponding JSON keywords and related data in the issued data.

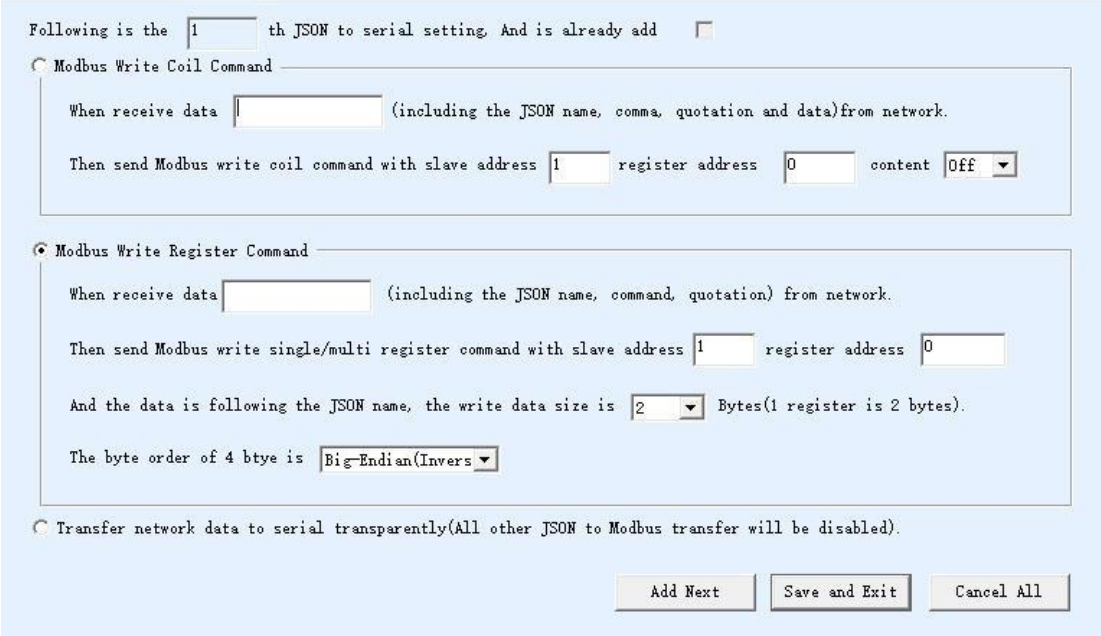

Figure 26 Sending JSON to Modbus

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

Modbus setting coil command: It is the 05 command, where you can specify the slave address, register address and set it to On and Off. In terms of data, if it corresponds to the JSON of "alarm":"1", you need to write the complete ""alarm":"1"" into the box, including quotation marks and colons.

Modbus register write instruction: it is the 06 instruction, currently supports 2 to 4 bytes, that is, 1 and 2 register writes. The data must be integer (without decimal point). Here you can set the sending station address, register address, and the number of bytes. If it is 4 bytes, you can select the big-endian format or the little-endian format. For data,if the corresponding method is "temp":"1234", you need to enter ""temp":"" in the input box. Here, you need to enter the characters before 1234, including quotation marks.

For the transparent transmission method, the issued command will be transparently transmitted to the serial port without analysis. Once the transparent transmission is selected, all JSON delivery and parsing will be disabled.

3.13. 645 TIMING OF THE AGREEMENT

The time service of the 645 protocol is 68 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 seconds, minutes, hours, days, months, and years.

Cooperate with Vircom and 4.99 (SN7044)/5.92 (SN2043) firmware to realize timing configuration. Copy the following character string "68 99 99 99 99 99 99 68 08 06 TIME[25...30] CRC[3] 16" to the serial port for simultaneous output command when sending; copy "TIME[0]" to the output condition .

Note that you must set at least one JSON data to be sent in order to implement timing and delivery. In addition, the time service can only be done after the TCP connection is established, because the time service occurs when the network is sent. If the TCP is not established, it will not be sent, and there is no time service.

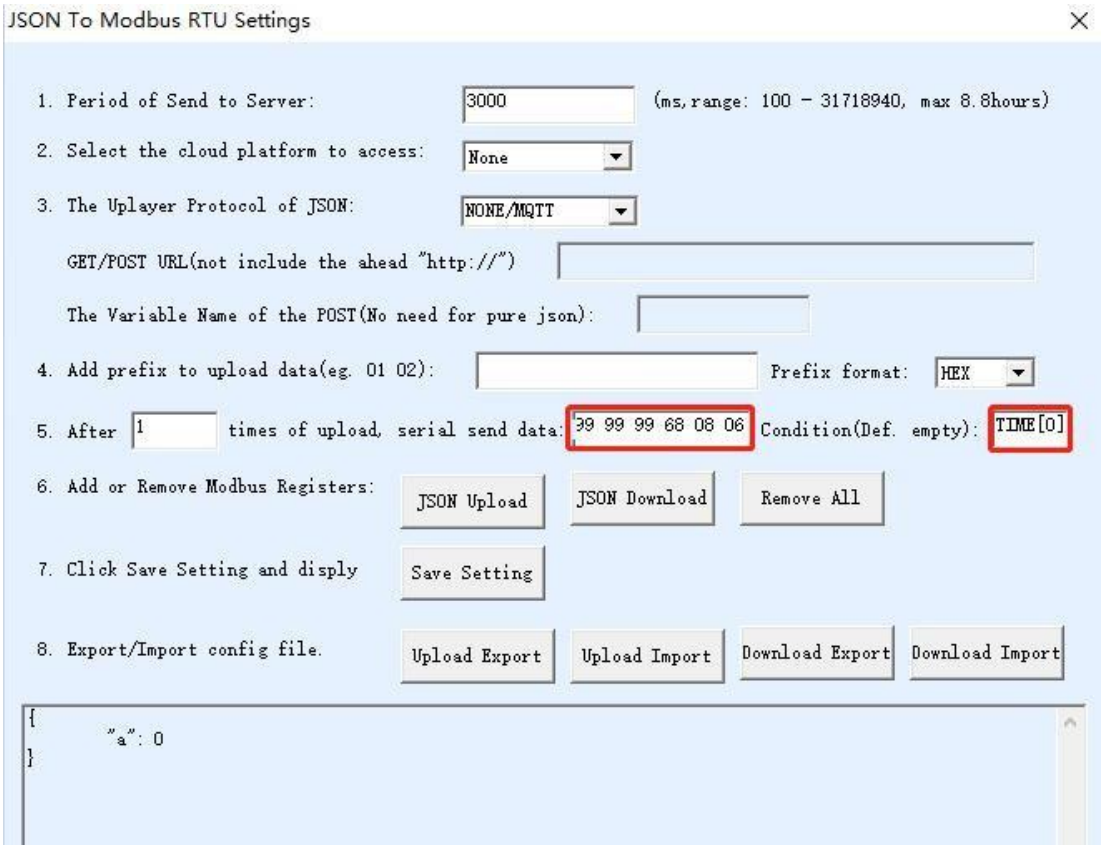

Figure 27 645 timing configuration

The "TIME[0]" here is optional. If it is filled in, the time service will be sent after the device obtains the effective time. If you do not fill in, the time service will be sent under any circumstances. If the time service is incorrect, it will cause problems.

4. JSON TO MODBUS RTU

JSON to Modbus RTU supports 05/06/16 command. If you need to use the 15 command to set multiple coils, please use the 05 command multiple times.

According to the length of the number of bytes, the system will automatically select the 06 or 16 command to send. Here are examples of setting coils and setting registers.

If you receive {alert:"on"} JSON data, you need to use the 05 command to set the station address 02 and the coil starting with register 03. Then in: JSON to Modbus interface, click "JSON Download".

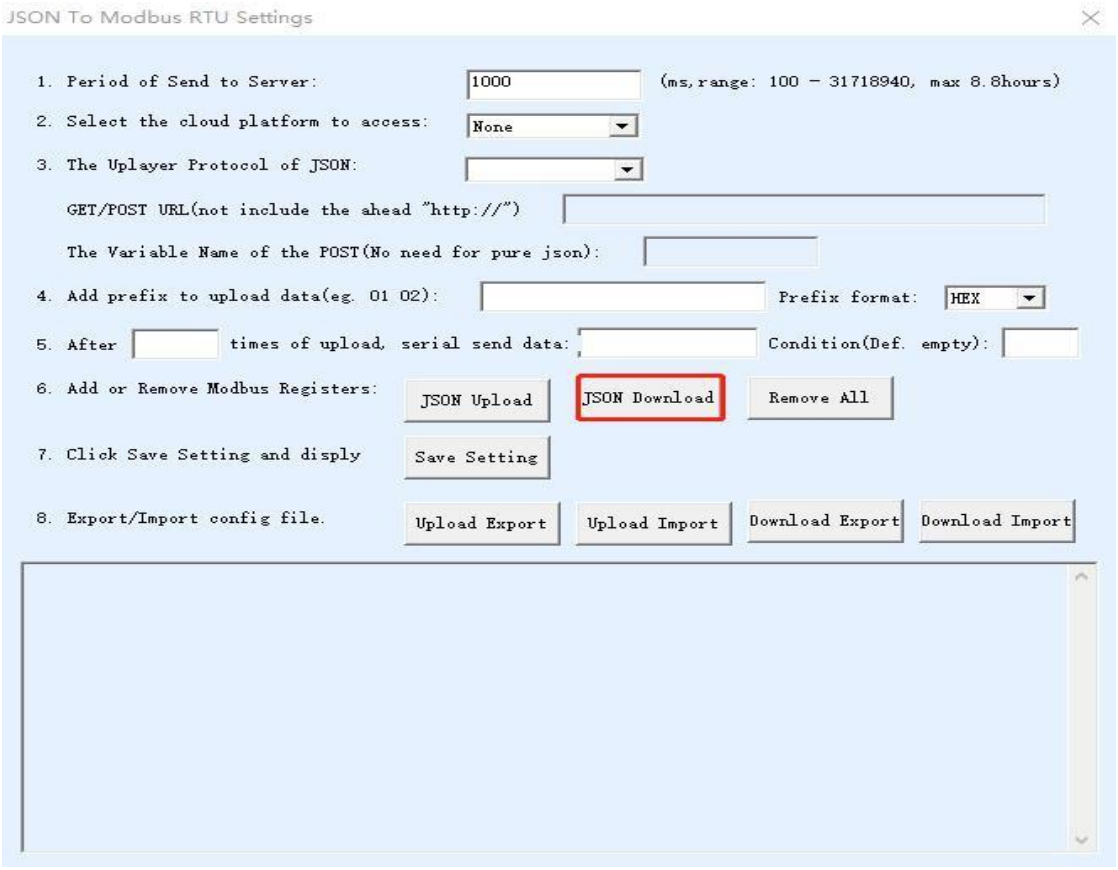

Figure 28 Enter JSON to send

The configuration interface is as follows: pay attention to the alert: "on setting that needs to be written.

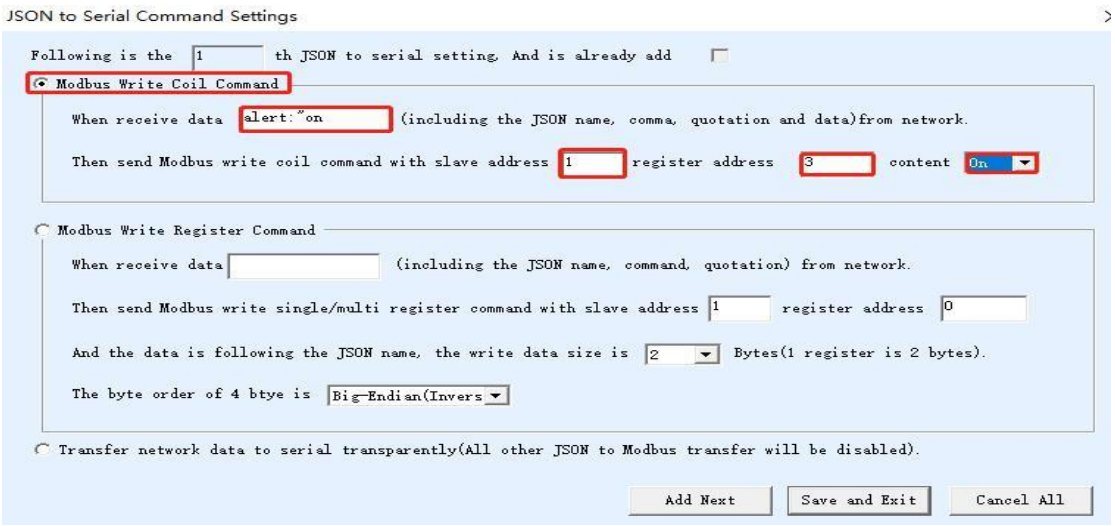

Figure 29 Configure coil

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

If {power:"12345"} is sent now, the power value 12345 needs to be set to station address 2 and register 3. The settings are as follows:

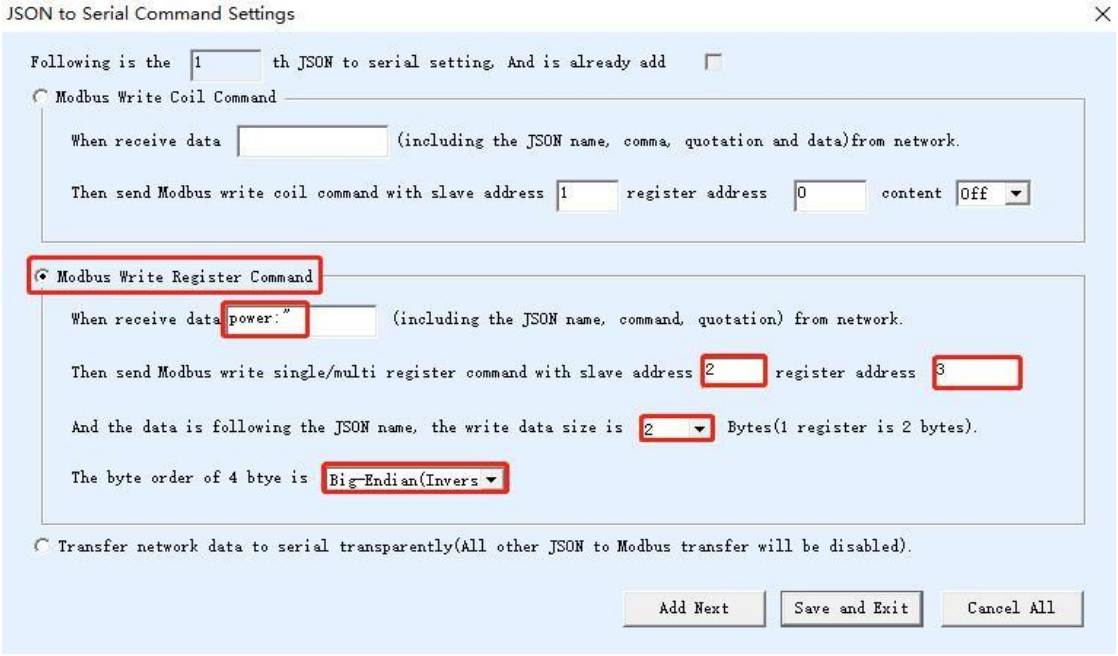

Figure 30 JSON setting register

Note that the keyword here only needs to enter power:", and you do not need to enter the following 12345, because this value is changed, but you need to enter a colon. If the quotation marks are in the issued data, you also need to enter quotation marks.

5. MQTT

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

5.1. DEVICE CONFIGURATION

First search for the device, then click Edit Device:

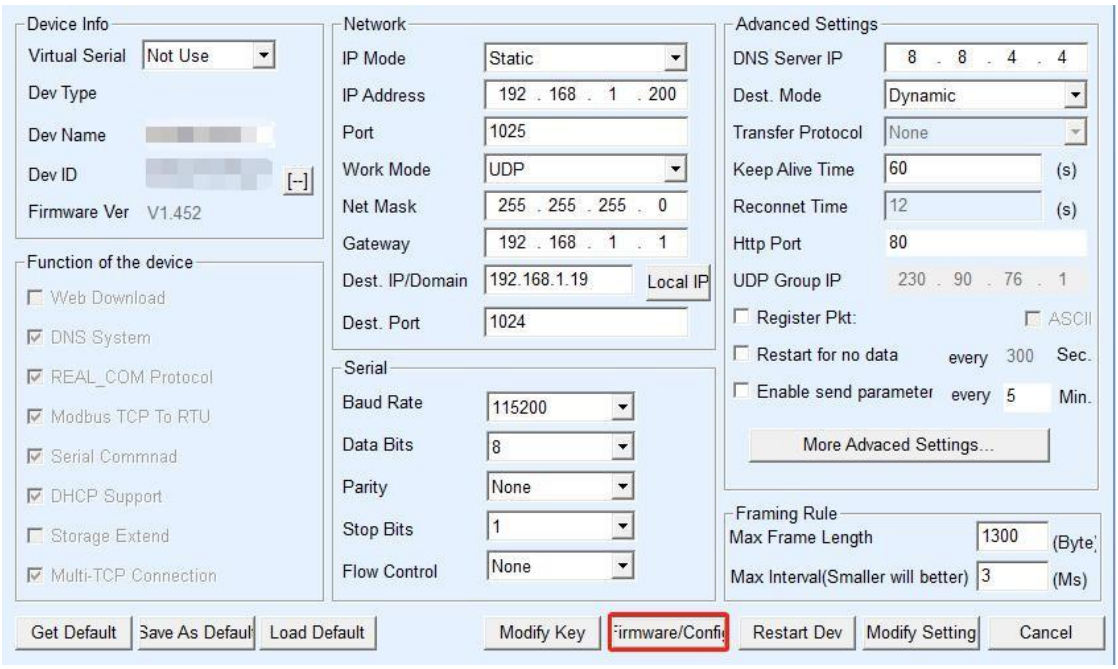

Figure 31 MQTT configuration 1

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

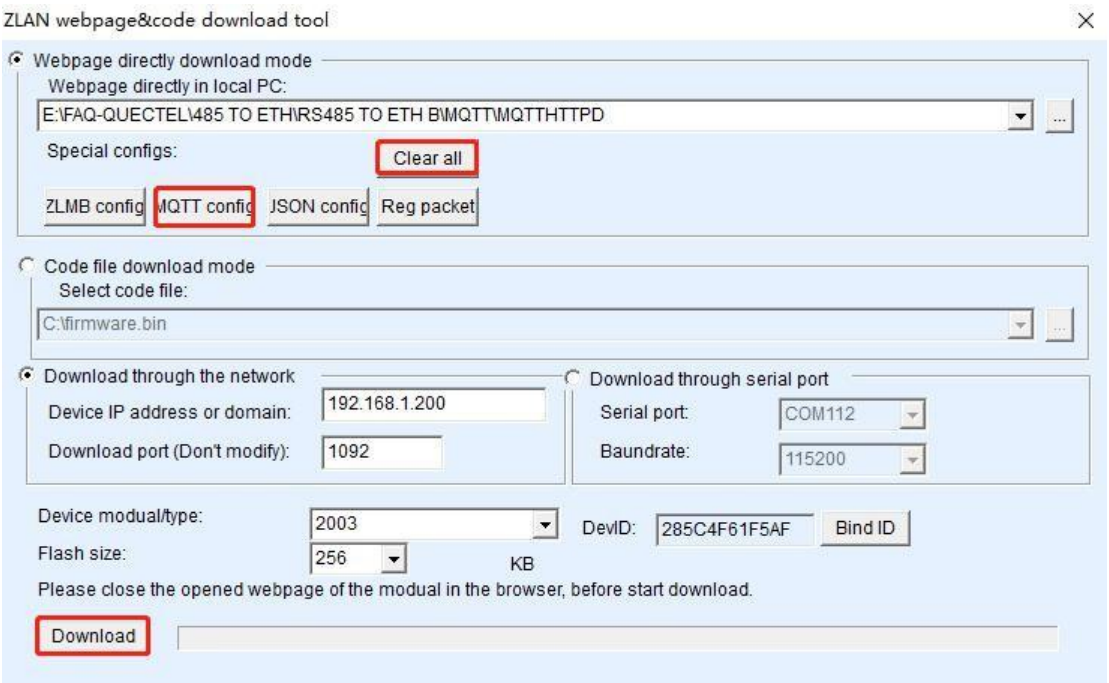

Figure 32 MQTT configuration 2

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

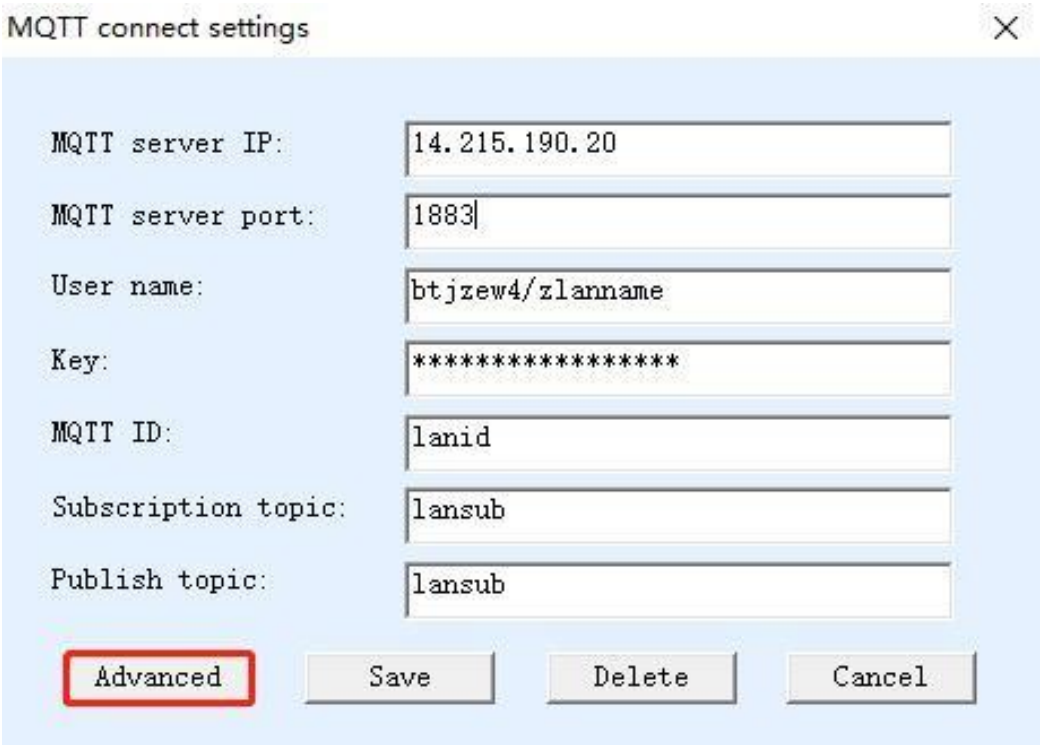

Figure 33 MQTT configuration 3

The configuration instructions here are as follows:

1 Server domain name or IP: here is the IP of the MQTT server, the maximum length is 30 characters.

2 Username: is the username of the MQTT server.

3 Password: is the login password of this user.

4 Client ID: It is the client ID of MQTT.

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

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

7 MQTT advanced parameters: used to configure advanced parameters.

8 Save MQTT settings: After designing, click this button to save, and then click the "download button" in the web page download directory to download.

Now click "MQTT advanced parameters" (generally no need to configure advanced parameters):

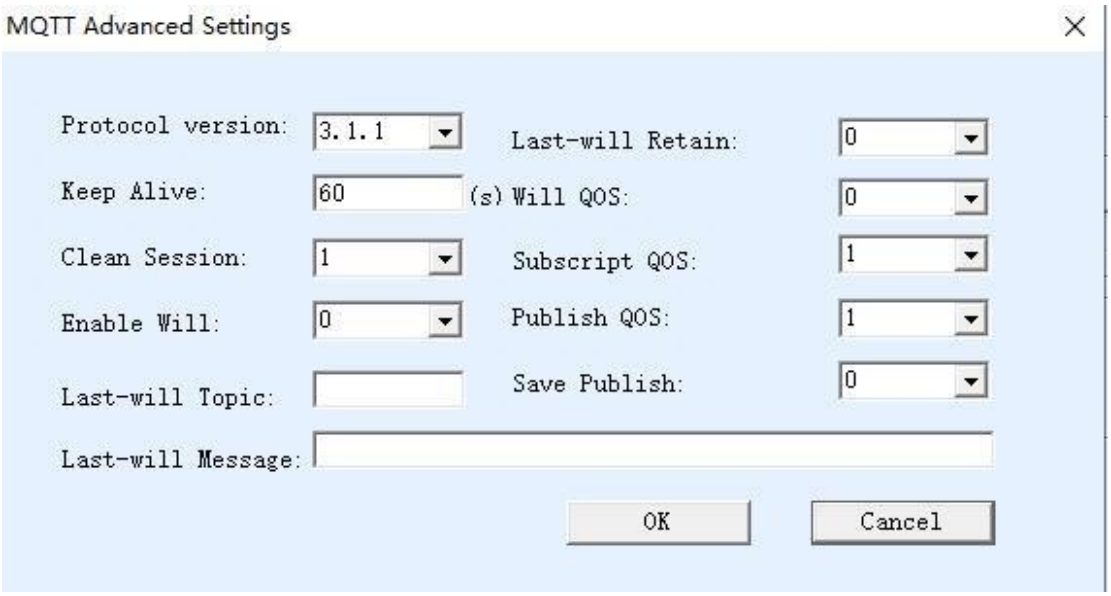

Figure 34 MQTT advanced parameter configuration

Described as follows:

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

2 Keep-alive time: The heartbeat time of MQTT, the minimum is 10 seconds, and the default is 60 seconds.

3 Server clear subscription: Whether the server clears the subscription information after the client is disconnected.

4 Whether to enable the last wish: whether there is a last wish.

5 Last will topic: Last wish topic.

6 Last will message: information about the last will.

7 Last will retain: whether the server needs to keep the last wish messagesent to the client when the client is abnormally offline.

8 Will QOS: the delivery quality level of the last will message sent by the server.

9 Subscript QOS: The delivery quality level of the subscription. In some cases, it needs to be set to 0 to prevent disconnection caused by retransmission.

10 Publish QOS: The delivery quality level of the message published by the client. In some cases, it needs to be set to 0 to prevent disconnection caused by retransmission.

11 Whether to save the publication: whether the server keeps the last message (if there is a new client subscription, it will be sent to the client)

We will not modify the advanced parameters here. Click "Save MQTT Settings" directly. Then click "download":

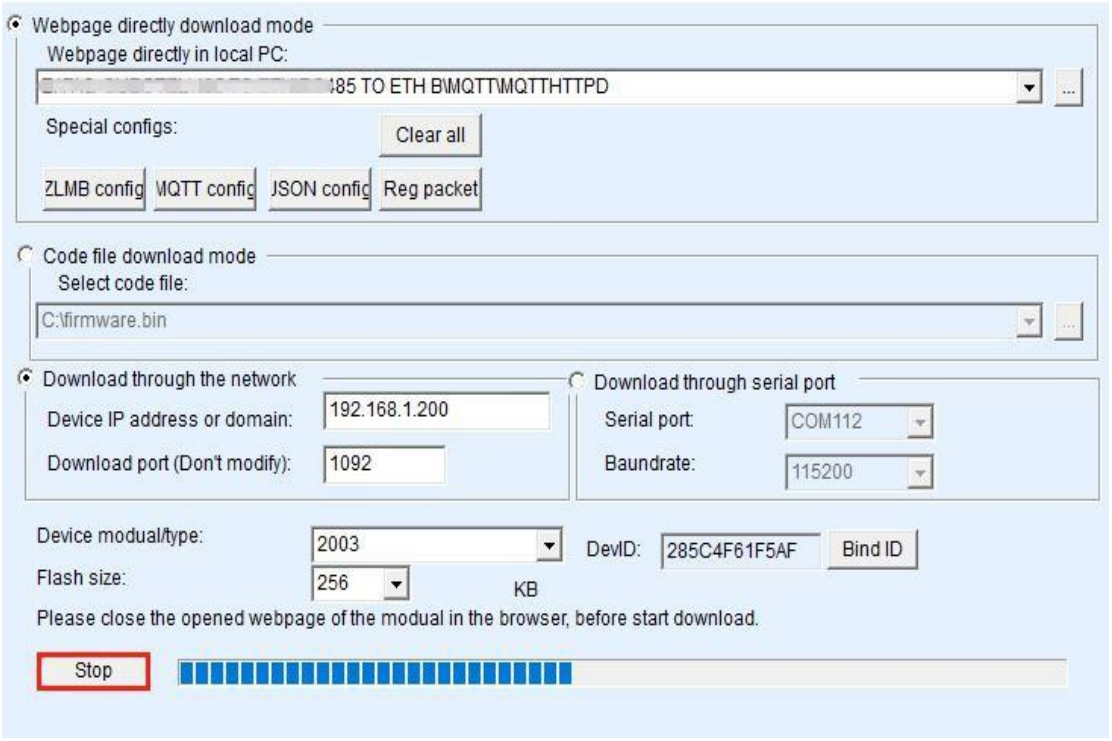

Figure 35 Download

After downloading, click OK, and you will return to the device management dialog box. You can see that the device's destination IP, working mode, and destination port have been automatically modified to the MQTT settings:

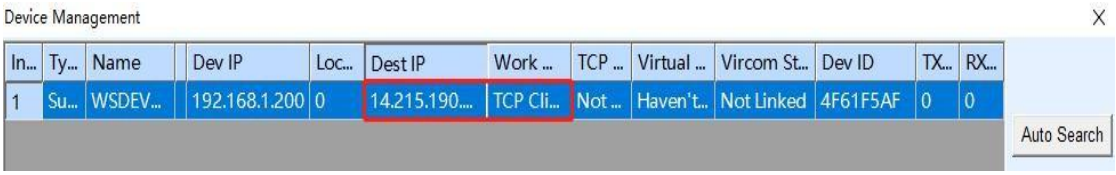

Figure 36 Automatic modification

If there is no automatic modification, you need to set the destination IP, working mode, and destination port in the device edit dialog box. Then click "Edit Settings".

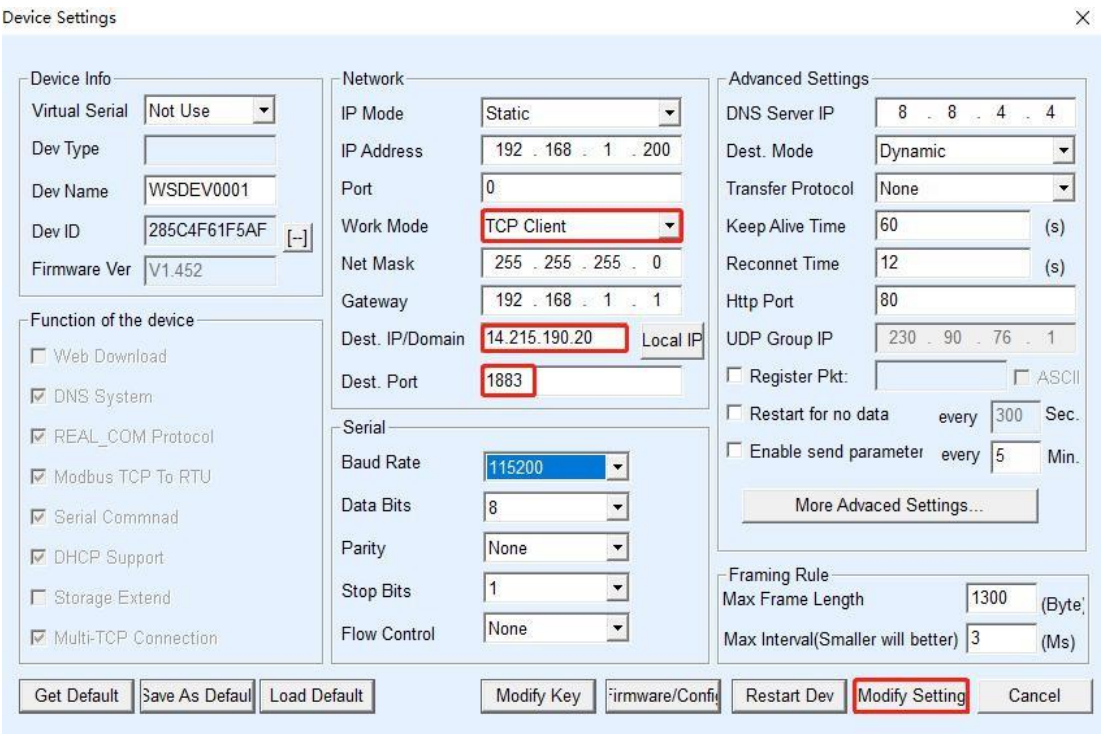

Figure 37 IP configuration

This configuration is complete.

5.2. DATA TEST

After the connection is completed, the LINK light (usually the blue light in the middle) of the device turns on. It indicates that the device is normally connected to the MQTT server.

Now open the serial port tool:

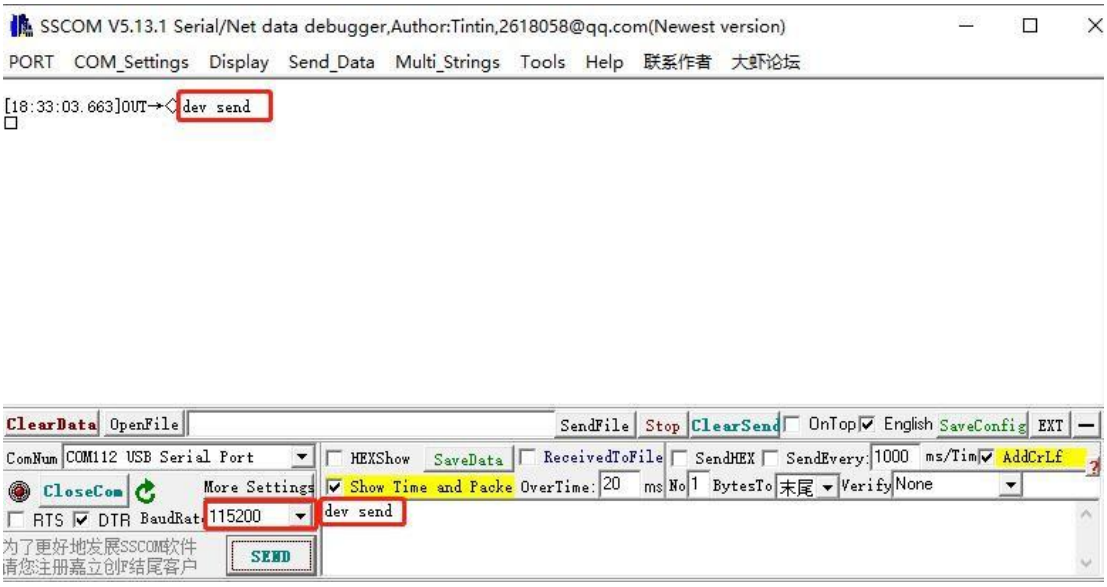

Figure 38 Serial port sending and receiving

Use the same baud rate as the device to open the serial port and send data "dev send", and then see the returned data "dev send" in the receiving window. This is because we publish the dev send message to the MQTT server under the lansub. But at the same time our device is also subscribed to the lansub, so the server will immediately send us a subscription message, and the content of the subscription message is dev send. This information is sent and downloaded as the MQTT payload, and is output from the serial port by the transparent transmission.

If the other devices publishes the information, this device can also receive the data.

Generally speaking, users can directly transparently transmit serial port commands (such as Modbus RTU) to the MQTT server. In addition, you can also use the JSON function such as the automatic Modbus RTU format collection, and the regular JSON format uploading. In addition, you can also find WAVESHARE to customize some non-standard instruments and host computer protocol formats.

6. MQTT+JSON TO MODBUS RTU

Combining the above JSON and MQTT can achieve the following functions:

1. The MQTT-based protocol is used to establish a connection with the server, and data communication is carried out in the form of subscription and publication.
2. Support independent design and automatic collection of Modbus RTU registers.
3. Support the conversion of specific Modbus register content into JSON format and send it regularly and actively.
4. Support adding device ID to JSON format to facilitate cloud identification of 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 type. After the two designs are finished, click the "Download" button together to download the contents of the device.

Generally, you can manually restart the device to load the settings after downloading.

7. HTTP POST/GET+JSON

In addition to MQTT, the host computer protocol can also choose HTTP protocol, and upload data through POST and GET instructions. Let's take the POST command as an example to introduce.

If you need to support the POST/GET+JSON function, choose the Vircom configuration tool to select version 5.17 and above; if you need to support POST command, it needs to be version 5.81 and above (only If you support GETyou can use the ordinary

version firmware that supports JSON).

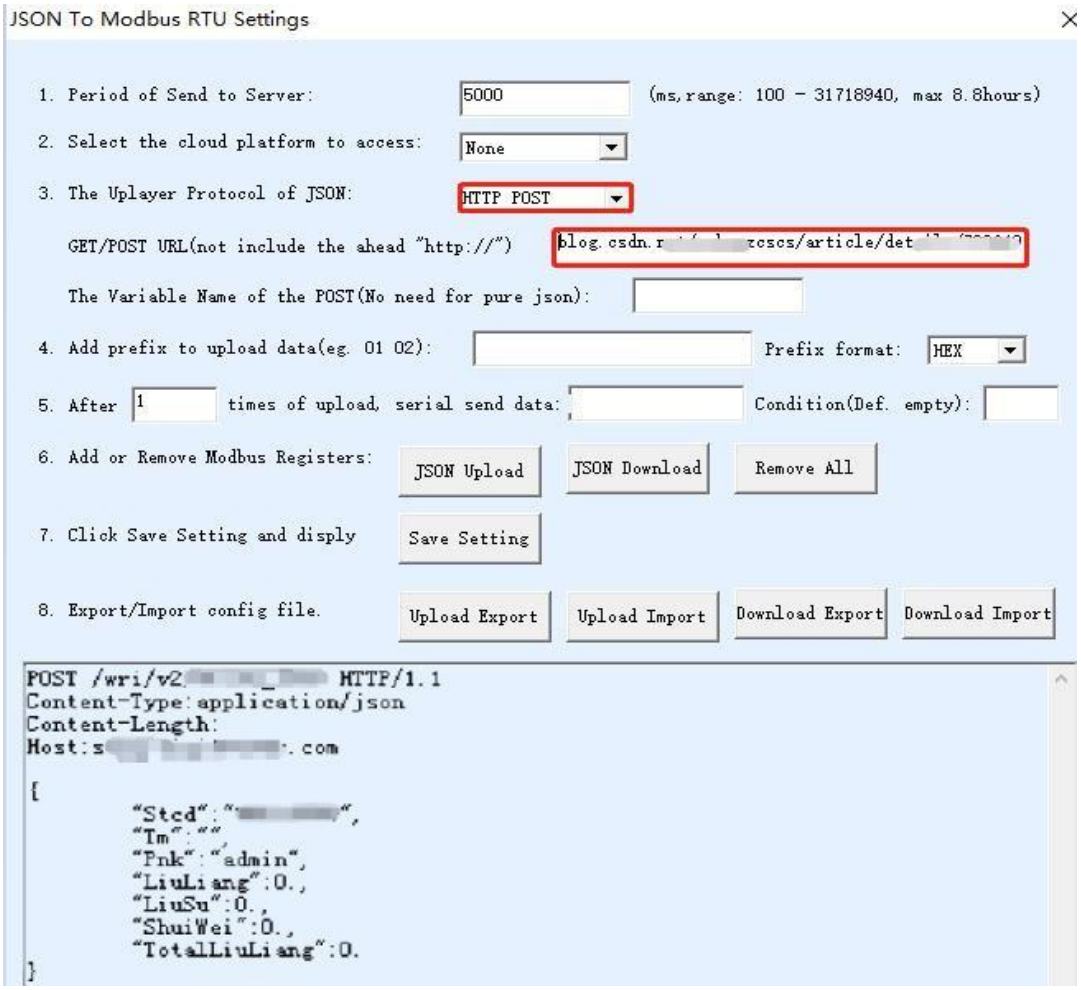

Figure 39 POST+JSON

Vircom version 5.17 adds two options in the JOSN to Modbus RTU settings, as shown in the figure:

1. JSON upper layer protocol: If it is no protocol or MQTT protocol, please select the first item: "NONE/MQTT". If it is HTTP POST, please select the second item "HTTP POST", if it is HTTP GET, please select the third item "HTTP GET".

2. POST/GET URL: When choosing POST or GET, you must fill in the URL. For example, if the URL is http://sacom/wri/v2, remove the http:// in front and enter sacom/wri/v2 directly .

Other JSON structure design methods are the same as the methods introduced before. After clicking the "Save JSON Settings" button, if POST/GET is selected, HTTP header format information will be added in front of the JSON data to support the HTTP transmission protocol.

This POST/GET design method is simple and practical, and it can simply and quickly realize the transmission of Modbus RTU and other instrument data to the server by means of HTTP POST/GET+JSON.