Milesight UR Series

MQTT API Reference Documentation

Contents

1. Overview

This document introduces the features, message formats, and usage of the Milesight Router Series MQTT API.
Note: The MQTT API features described in this document represent the full feature set supported by the Milesight Router Series. Due to differences in hardware configurations and software versions across models, some APIs, data fields, or functional modules may not be applicable to all devices. For features involving serial ports, DI/DO, GPS, Modbus, DLMS, etc., the device must have the corresponding hardware interface or functional module. Please refer to the specific product datasheet and actual device capabilities for supported features.

1.1 User-Defined System Common Message Structure

Request Message (Client → Router)

```
{
    "id": "1",
    "status": "systeminfo",
    "code": "2",
    "time": "2023-05-10T13:03:17Z",
    "sn": "6219A0780705",
    "data": { }
}
```

Field	Type	Required	Description
id	string	Yes	Random code, used to match request and response
status	string	Yes	Request type, see individual feature sections
sn	string	No	Device SN, used to target a specific device for request processing; if omitted, all devices respond
need_response	int	No	Whether a response is needed: 0=no response needed, 1=response needed (default: 1)

Response Message (Router → Client)

```
{
    "id": "1",
    "status": "systeminfo",
    "code": "2",
    "time": "2023-05-10T13:03:17Z",
    "sn": "6219A0780705",
    "data": { }
}
```

Field	Type	Description
id	string	Same as the id in the request

Field	Type	Description
status	string	Same as the status in the request
code	string	Communication code: "1" = active reporting, "2" = subscription response
time	string	UTC time, format: YYYY-MM-DDTHH:MM:SSZ
sn	string	Device SN (returned only when the request includes the sn field)
data	object	Specific data, see individual feature sections

2. System Status Query & Reporting

Topics are configured by the user in the device interface (req_topic / resp_topic).

Two modes are supported: **Subscription Response** (Client sends request, Router replies) and **Active Reporting** (Router automatically publishes on a periodic basis).

2.1 System Information (systeminfo)

Query basic device identification information, including model, serial number, firmware version, and hardware version. Suitable for initial device identification after going online, or scenarios requiring firmware version confirmation. This information is static and does not change with operating status.

Topic

Direction	Topic	QoS	Retain
Client → Router (Subscribe)	User-configured req_topic	User-configured	-
Router → Client (Publish)	User-configured resp_topic	User-configured	User-configured

Request Example

```
{
    "id": "1",
    "status": "systeminfo"
}
```

Response Example

```
{
    "id": "1",
    "status": "systeminfo",
    "code": "2",
    "time": "2023-05-10T13:03:17Z",
    "sn": "6219A0780705",
    "data": {
        "model": "UR35-L04EU-G-P-W",
        "sn": "6219F3692475",
        "firmware_ver": "35.3.0.12",
        "hardware_ver": "V3.0"
    }
}
```


data Field Description

Field	Type	Description
model	string	Device model
sn	string	Device serial number
firmware_ver	string	Firmware version
hardware_ver	string	Hardware version

2.2 System Status (systemstatus)

Query the current device operating status, including system uptime, CPU usage, CPU temperature, memory usage, Flash usage, and TF card status. Suitable for device health monitoring scenarios; can be combined with active reporting for periodic push notifications to enable remote O&M monitoring. UR4X models do not include TF card related fields.

Request Example


```
{
    "id": "1",
    "status": "systemstatus"
}
```


Response Example


```
{
    "id": "1",
    "status": "systemstatus"
    "code": "2",
    "time": "2023-05-10T13:03:17Z",
    "sn": "6219A0780705",
    "data":
    {
      "uptime": 10613,
```

```
        "cpu_load": "33%",
        "cpu_temp": 55,
        "memory_total": 128,
        "memory_free": 34,  "
        "flash_total": 128,
        "flash_free": 77,
        "tfcard_state": 1,
        "tfcard_total": 7630,
        "tfcard_free": 6000
    }
}
```


data Field Description

Field	Type	Description
uptime	int	System uptime (seconds)
cpu_load	string	CPU usage
cpu_temp	int	CPU temperature (℃)
memory_total	int	Total memory (MB)
memory_free	int	Available memory (MB)
flash_total	int	Total Flash (MB)
flash_free	int	Available Flash (MB)
tfcard_state	int	TF card status (0=not inserted, 1=normal)
tfcard_total	int	Total TF card capacity (MB)
tfcard_free	int	Available TF card space (MB)

Note: The UR4X does not support TF cards, the response does not include the tfcard_state, tfcard_total or tfcard_free fields.

2.3 Cellular Network Status (cellular)

Query detailed cellular module information, each APN network connection status, and SIM card traffic statistics. Response data is divided into three parts: modem (module hardware info, signal, registration status), network (IP/DNS/connection duration for apn1/apn2/apn3 channels), and statistics (SIM card traffic and SMS statistics).

Request Example


```
{
    "id": "1",
    "status": "cellular"
}
```

Response Example


```
{
    "id": "1",
    "status": "cellular",
    "code": "2",
    "time": "2023-05-10T13:03:17Z",
    "sn": "6219A0780705",
    "data":
    {
        "modem": {
            "modem_status": "Ready",
            "modem":"EC25"
            "Version": "EC25EUXGAR08A19M1G",
            "cur_sim": "SIM1",
            "signal": "25asu (-63dBm)",
            "band" : "4G(B1)"
            "rsrp": "-86 dBm",
            "rsrq": "-7 dB",
            "sinr": "27 dB",
            "rscp": "0 dBm",
            "signal_quality": "good",
            "register": "Registered (Home network)",
            "imsi": "460013972050982",
            "iccid": "89860125003900082228",
            "net_provider": "CHN-UNICOM",
            "net_type": "FDD LTE",
            "plmnid": "46001",
            "lac": "5f0c",
            "cellid": "e0b70b",
            "imei": "867734081202802",
            "rssi": "-65 dBm"
        },
        "network": {
            "apn1": {
                "status": "Connected",
                "ip": "10.133.144.241",
                "netmask": "255.255.255.252",
                "gate": "10.133.144.242",
                "dns": "218.104.128.106",
                "sec_dns": "58.22.96.66",
                "ipv6": "2408:8448:2003:52a3:100f:56ba:321b:a127/64",
                "gatev6": "2408:8448:2003:52a3:24da:e9c3:ef7c:1916",
                "dnsv6" "2408:8888:0:0:0:0:0:8",
                "sec_dnsv6": "2408:8899:0:0:0:0:0:8",
                "time": "0 days, 02:40:40"
            },
            "apn2":{
               "status": "Disabled",
                "ip": "-",
                "netmask": "-",
                "gate": "-",
                "dns": "-",
```

```
                "sec_dns": "-",
                "ipv6": "-",
                "gatev6": "-",
                "dnsv6": "-",
                "sec_dnsv6": "-",
                "time": "-"
            },
            "apn3":{
                "status": "Disabled",
                "ip": "-",
                "netmask": "-",
                "gate": "-",
                "dns": "-",
                "sec_dns": "-",
                "ipv6": "-",
                "gatev6": "-",
                "dnsv6": "-",
                "sec_dnsv6": "-",
                "time": "-"
            }
        },
        "statistics":
        {
            "sim1": "RX: 0.0 MiB   TX: 0.0 MiB   ALL: 0.0 MiB",
            "sim1_monthly_sms": 0,
            "sim2": "RX: 0.0 MiB   TX: 0.0 MiB   ALL: 0.0 MiB",
            "sim2_monthly_sms": 0
        }
    }
}
```


data.modem Field Description

Field	Type	Description
modem_status	string	Module status
model	string	Module model
version	string	Module firmware version
cur_sim	string	Currently used SIM card
signal	string	Signal strength (including asu and dBm)
band	string	Current frequency band
rsrp	string	Reference Signal Received Power (LTE)
rsrq	string	Reference Signal Received Quality (LTE)
sinr	string	Signal to Noise Ratio (LTE)
rscp	string	Received Signal Code Power (3G)

Field	Type	Description
signal_quality	string	Signal quality description
register	string	Registration status
imsi	string	SIM card IMSI
iccid	string	SIM card ICCID
net_provider	string	Network operator name
net_type	string	Network type (e.g., FDD LTE)
plmnid	string	PLMN ID
lac	string	Location Area Code (hexadecimal)
cellid	string	Cell ID (hexadecimal)
imei	string	Module IMEI
rssi	string	Received Signal Strength (dBm)

data.network.apnX Field Description

Field	Type	Description
status	string	Connection status (Connected / Disabled)
ip	string	IPv4 address
netmask	string	Subnet mask
gate	string	IPv4 gateway
dns	string	Primary DNS
sec_dns	string	Secondary DNS
ipv6	string	IPv6 address (with prefix length)
gatev6	string	IPv6 gateway
dnsv6	string	Primary IPv6 DNS
sec_dnsv6	string	Secondary IPv6 DNS
time	string	Connection duration

data.statistics Field Description

Field	Type	Description
sim1	string	SIM1 traffic statistics (format: RX: x MiB TX: x MiB ALL: x MiB)

Field	Type	Description
sim1_monthly_sms	int	SIM1 monthly SMS count
sim2	string	SIM2 traffic statistics
sim2_monthly_sms	int	SIM2 monthly SMS count

Note: The UR4X has only one SIM card, in the statistics, the "sim1" field has been renamed to "sim", and the "sim2" and “sim2_monthly_sms” fields have been removed. The “sim1_monthly_sms” field name remains unchanged.

2.4 Ethernet Status (ethernet)

Query the network status of the router WAN and LAN ports, including IP address, MAC address, connection duration, and number of connected devices. UR3X returns both wan and lan; UR4X only returns lan.

Request Example

```
{
    "id": "1",
    "status": "ethernet"
}
```

Response Example

```
{
    "id": "1",
    "status": "ethernet",
    "code": "2",
    "time": "2023-05-10T13:03:17Z",
    "sn": "6219A0780705",
    "data": {
        "wan": {
            "status": 1,
            "mac": "24:e1:24:ff:d8:c1",
            "ip": "192.168.50.56/24",
            "ipv6": "fe80::26e1:24ff:feff:d8c1/64",
            "time": "0 days, 02:41:13"
        },
        "lan": {
            "ip": "192.168.1.1/24",
            "ipv6": "fe80::26e1:24ff:feff:d8bf/64",
            "connected": 4
        }
    }
}
```

data.wan Field Description

Field	Type	Description
status	int	Connection status: 1=up, 0=down
mac	string	MAC address
ip	string	IP address (with subnet mask, e.g., 192.168.1.1/24)
ipv6	string	IPv6 address
time	string	Connection duration

data.lan Field Description

Field	Type	Description
ip	string	LAN IP address (with subnet mask)
ipv6	string	IPv6 address
connected	int	Number of connected devices

Note: UR3X includes the "wan" field, UR4X does not include the "wan" field and returns only "lan".

2.5 GPS Location Information (gps)

Query parsed GPS positioning data, including latitude/longitude, speed, altitude, number of satellites, etc. Unlike the GPS raw data reporting in Section 3.3, this API returns structured data already parsed by the router, ready for direct use.

Request Example

```
{
    "id": "1",
    "status": "gps"
}
```

Response Example

```
{
    "id": "1",
    "status": "gps",
    "code": "2",
    "time": "2023-05-10T13:03:17Z",
    "sn": "6219A0780705",
    "data": {
        "status": "53",
        "time": "2023-05-10T13:03:17Z",
        "latitude": "22.543099",
        "longitude": "114.057868",
        "speed": "10.101206 km/h",
        "height": "5.8 m",
```

```
        "sats": 8,
        "sativ": 12
    }
}
```

data Field Description

Field	Type	Description
status	string	GPS status code: 50=no GPS module, 51=GPS function not enabled, 52=poor signal, 53=normal positioning
time	string	GPS positioning time (UTC)
latitude	string	Latitude (decimal degrees)
longitude	string	Longitude (decimal degrees)
speed	string	Speed (km/h)
height	string	Altitude (unit varies by actual distance)
sats	int	Number of satellites currently used
sativ	int	Total visible satellites

Note: The request response or active reporting here provides parsed GPS location information, while in the industrial interface data section below, MQTT forwards the raw GPS data returned by the module.

2.6 Digital Input Status Query (di)

Query the current level status of the Digital Input (DI) port. In the response, data.value is 0 for low level and 1 for high level. Suitable for scenarios requiring real-time acquisition of DI port current status. For automatic push notifications on DI status changes, use the DI event reporting API in Section 3.5.

Topic: Same as system status query; user-configured req_topic / resp_topic.

Request Example

```
{
    "id": "1",
    "status": "di"
}
```

Response Example

```
{
    "id": "1",
    "status": "di",
    "code": "2",
    "time": "2023-05-10T13:03:17Z",
    "sn": "6219A0780705",
    "data": {
       "value": 1
    }
}
```


data Field Description

Field	Type	Description
value	int	DI current level: 0=low level, 1=high level

2.7 Digital Output Control & Query (do)

Trigger the Digital Output (DO) port to execute an output action, and simultaneously return the current DO configuration status. These two operations are performed independently:

- **Trigger:** After receiving the request, the router writes a trigger signal to an internal communication file. The DO output process detects the signal within 1 second and drives the hardware to execute (the DO port must be enabled in the interface).
- **Response:** The router reads the current DO configuration and returns it. The response reflects the configuration status, not the execution result of the hardware action.

The request does not need to carry a data field; the DO operating mode and parameters are pre-configured in the device interface. In the response, result=1 indicates DO is enabled; result=2 indicates DO is disabled.

Topic: Same as system status query; user-configured req_topic / resp_topic.

Request Example


```
{
    "id": "1",
    "status": "do"
}
```


Response Example (high/low mode)

```
{
    "id": "1",
    "status": "do",
    "code": "2",
    "time": "2023-05-10T13:03:17Z",
    "sn": "6219A0780705",
    "result": 2,
    "data": {
        "enable": 0,
        "duration": 100,
        "mode": 0
    }
}
```


Response Example (pulse mode)


```
{
    "id": "1",
    "status": "do",
    "code": "2",
    "time": "2023-05-10T13:03:17Z",
    "sn": "6219A0780705",
    "result": 1,
    "data": {
       "enable": 1,
       "init_status": 0,
       "duration_of_high": 500,
       "duration_of_low": 500,
       "amount_of_pulse": 10,
       "mode": 2,
       "phone_group": "group1"
        }
}
```


Top-Level Field Description

Field	Type	Description
result	int	Execution result: 1=DO enabled, 2=DO disabled

data Field Description

Field	Type	Description
enable	int	Enable status: 0=disabled, 1=enabled
mode	int	Output mode: 0=high, 1=low, 2=pulse, 3=custom
duration	int	Duration (seconds), valid in high/low mode
init_status	int	Initial status: 0=low, 1=high, valid in pulse/custom mode

Field	Type	Description
duration_of_high	int	Pulse high-level duration (ms), valid in pulse mode
duration_of_low	int	Pulse low-level duration (ms), valid in pulse mode
amount_of_pulse	int	Number of pulses, valid in pulse mode; 0=continuous output
init_status	int	Initial status: 0=low, 1=high, valid in pulse/custom mode
phone_group	string	Phone group ID (present only if a phone group is configured)

Note: Upon receiving a DO request, the router triggers the DO output action and returns the current DO configuration status. When need_response=0, the router does not reply.

3. Industrial Interface Data Active Reporting

All APIs in this chapter are **actively published** by the router (Router → Client); no request from the Client is needed. The router automatically pushes data to the configured MQTT topic after collecting it.

3.1 Serial Data Passthrough (DTU)

The router directly forwards raw data received from serial ports (RS232/RS485) to MQTT without parsing. Suitable for scenarios requiring serial device data to be connected to cloud platforms, such as industrial instruments, PLCs, barcode scanners, etc. The data content and format are entirely determined by the serial device output; the router performs no processing.

Topic: web.

Direction: Router → Client()

Payload: Serial data is encapsulated in JSON format; the data field contains the raw content received from the serial port (string).

Payload Example

```
{"data": "..."}
```

Note: The contents of the "data" field depend entirely on the output from the serial device; the router does not perform any parsing.

MQTT Configuration Parameters (Serial Port Web Config)

Parameter	Description
mqtt_name	MQTT client name
topic	Publish topic
qos	QoS level (0/1/2)
retain_flag	Message retain flag (0/1)

Parameter	Description
type	Message type flag

3.2 Modbus Client Data Collection Reporting

The router acts as a Modbus Client, polling slave registers at the configured collection interval and publishing the results to MQTT in JSON format. Suitable for scenarios requiring periodic reporting of data from Modbus devices such as PLCs and instruments. Each Modbus channel can independently configure the collection address, register range, and reporting topic.

Topic: web Modbus Client .

Direction: Router → Client()

Payload

```
{"channel_name": "hex_data"}
```

Where channel_name is the user-configured channel name, and hex_data is the complete Modbus response frame.

Example

```
{"6": "0603020010"}
```

The example above shows channel name 6 with a Modbus response frame of 0603020010.

MQTT Configuration Parameters (Modbus Client Web Config)

Parameter	Description
mqtt_name	MQTT client name
topic	Publish topic (response_topic)
qos	QoS level
retain_flag	Message retain flag

3.3 GPS Raw Data Reporting (NMEA)

The router directly publishes the raw sentences output by the GPS module to MQTT without parsing. Suitable for scenarios requiring self-parsing of GPS raw data in the cloud. Unlike the GPS location information API in Section 2.5, this API reports unparsed raw messages (such as $GPRMC, $GPGGA, etc.), with accuracy and fields entirely determined by the GPS module. Users can selectively enable sentence types such as RMC/GGA/GSA/GSV as needed, supporting simultaneous push to up to 5 MQTT connections.

Topic: GPS MQTT .

Direction : Router → Client

Payload: NMEA , ()

NMEA Sentence	Config Field	Description
$GPRMC	rmc	Recommended Minimum Navigation Information (position, speed, time)
$GPGGA	gga	GPS Fix Data (position, altitude, number of satellites)
$GPGSA	gsa	GPS DOP and Active Satellites
$GPGSV	gsv	Visible Satellites Information

Payload Example

```
$GPRMC,130317.00,A,2232.58594,N,11403.47208,E,0.000,,100523,,,A*6E
$GPGGA,130317.00,2232.58594,N,11403.47208,E,1,08,1.0,50.0,M,0.0,M,,*47
```

3.4 DLMS Data Collection Reporting

The router communicates with electricity meters via the DLMS/COSEM protocol, collects meter data (COSEM objects such as energy, voltage, current, etc.), and publishes the results to MQTT.

Topic: DLMS (mqtt_name + topic + qos + retain).

Direction: Router → Client (active)

Payload : Published in the following JSON format by default; fields are optional (web configuration).


```
{
    "name": "group_1",
    "timestamp": 1683720197,
    "date": "10/05/2023 13:03:17",
    "date_iso_8601": "2023-05-10T13:03:17",
    "sn": "6219A0780705",
    "size": 128,
        "data": {
            "cosem_value_name": {
                "server_name": {
                    "1": "0.0.1.0.0.255",
                    "2": "01\/00\/1900 00:00:00 UTC+00:00",
                    "3": "0",
                    "4": "0",
                    "5": "01\/00\/1900 00:00:00 UTC+00:00",
                    "6": "01\/00\/1900 00:00:00 UTC+00:00",
                    "7": "0",
                    "8": "0",
                    "9": "0",
                }
            },
             "cosem_value_name2": {
                "server_name2": {
                    "1": "0.0.1.0.0.255",
                    "2": "01\/00\/1900 00:00:00 UTC+00:00",
```

```
                    "3": "0",
                    "4": "0",
                    "5": "01\/00\/1900 00:00:00 UTC+00:00",
                    "6": "01\/00\/1900 00:00:00 UTC+00:00",
                    "7": "0",
                    "8": "0",
                    "9": "0",
                }
            }
        }
}
```


Field Description

Field	Type	json_flags	Description
name	string	bit0 (FLAG_NAME)	COSEM collection group name
timestamp	int	bit1 (FLAG_TIMESTAMP)	Unix timestamp (seconds)
date	string	bit2 (FLAG_DATE)	Local time, format: dd/MM/YYYY HH:MM:SS
date_iso_8601	string	bit3 (FLAG_DATE_ISO_8601)	ISO 8601 format time
sn	string	bit4 (FLAG_SN)	Device SN
size	int	bit5 (FLAG_SIZE)	Data size in bytes
data	object	Bit6 (FLAG_DATA)	COSEM object data; key is the OBIS code, value is the collected value

The web interface support to configure custom reporting formats.

3.5 Digital Input Event Reporting (DI)

When the DI port triggers the configured condition, the router actively publishes an event message to the configured MQTT topic. Unlike the DI status query in Section 2.6, this API operates in **event-driven** mode: no Client polling is needed; the router automatically pushes when the DI status meets the trigger condition. Supports four modes: high-level trigger (HIGH), low-level trigger (LOW), counter trigger (COUNTER), and switch trigger (SWITCH), each carrying different fields. Trigger conditions and MQTT targets are configured in the device industrial interface settings.

The topic is configured in the device I/O settings.

Direction: Router → Client (active)

Payload

```
{
    "data": {
        "enable": 1,
        "mode": 0,
        "duration": 5
    }
}
```

data Field Description

Field	Type	Description
enable	int	Trigger enable: 1=triggered
mode	int	Trigger mode, see table below
duration	int	Duration (seconds), valid in HIGH/LOW/SWITCH mode
condition	int	Count condition (COUNTER mode): 0=rising edge (L → H), 1=falling edge (H → L)
counter	int	Counter value (COUNTER mode)
level_switch	int	Switch level (SWITCH mode): 0=low, 1=high

mode Value Description

mode Value	Mode	Description
0	HIGH	High-level trigger
1	LOW	Low-level trigger
3	COUNTER	Counter trigger
4	SWITCH	Switch trigger

Payload Examples by Mode

HIGH Mode:

```
{"data": {"enable": 1, "mode": 0, "duration": 5}}
```

LOW Mode:

```
{"data": {"enable": 1, "mode": 1, "duration": 5}}
```

COUNTER Mode:

```
{"data": {"enable": 1, "mode": 3, "condition": 0, "counter": 10}}
```

SWITCH Mode:

{"data": {"enable": 1, "mode": 4, "level_switch": 1, "duration": 3}}

4. Modbus Client Control (Request/Response)

Send Modbus read/write commands to the router via MQTT. The router acts as a Modbus Client, executes the command, and returns the result. Unlike the periodic collection reporting in Section 3.2, this API operates in **on-demand control** mode: the Client sends a request, and the router immediately executes one Modbus operation and replies with the result.

Supports both RTU (serial) and TCP links, and all standard function codes including Read Coils, Read Registers, Write Coils, and Write Registers. Suitable for scenarios requiring real-time reading or modification of slave register values.

Topic

Direction	Topic	Description
Client → Router (Subscribe)	User-configured request_topic	Router subscribes to this topic to receive requests
Router → Client (Publish)	User-configured response_topic	Router publishes the execution result

4.1 Request Message

Request Example (Read Holding Registers)

```
{
    "id": 1,
    "link_type": "rtu",
    "server_id": 1,
    "modbus_function": 3,
    "first_register": 0,
    "register_count": 5
}
```

Request Example (Write Single Register)

```
{
    "id": 3,
    "link_type": "rtu",
    "server_id": 1,
    "modbus_function": 6,
    "first_register": 10,
    "register_value": "100"
}
```

Request Example (Write Multiple Registers)

```
{
    "id": 4,
    "link_type": "rtu",
    "server_id": 1,
    "modbus_function": 16,
    "first_register": 0,
    "register_count": 3,
    "register_value": "100,200,300"
}
```

Request Field Description

Field	Type	Required	Description
id	int	Yes	Request ID, used to match the response
link_type	string	Yes	Link type: "rtu"=serial, "tcp"=TCP
server_id	int	Yes	Modbus slave address (1-247)
ip	string	Required in TCP mode	Slave IP address
port	int	Required in TCP mode	Slave port number
modbus_function	int	Yes	Modbus function code, see table below
first_register	int	Yes	Starting register address
register_count	int	Required for read operations / write multiple	Register count
register_value	string	Required for write operations	Write value(s), multiple values separated by commas (e.g., "100,200,300")

4.2 Function Code Description

	Operation	register_count	register_value
1	Read Coils	Required	-
2	Read Discrete Inputs	Required	-
3	Read Holding Registers	Required	-
4	Read Input Registers	Required	-
5	Write Single Coil	-	Required, value is "0" or "1"
6	Write Single Register	-	Required, integer string
15	Write Multiple Coils	Required	Required, comma-separated "0" or "1"
16	Write Multiple Registers	Required	Required, comma-separated integers

4.3 Response Message

Success Response

```
{
    "id": "1",
    "status": "success",
    "data": "100,200,300,400,500"
}
```

Failure Response

```
{
    "id": "1",
    "status": "error",
    "message": "Connection timed out"
}
```

Response Field Description

Field	Type	Description
id	string	Same as the id in the request
status	string	"success"=success, "error"=failure
data	string	Read data, multiple values separated by commas (on success)
message	string	Error description (on failure)

Note: In the "data" field, coil/discrete inputs return 0 or 1, whilst registers return decimal integers. The multiple values are separated by commas.

5. Event Notification

When the router detects a configured event, it actively pushes a notification message.

Topic: Event Notification (user-configurable via the interface).

Direction : Router → Client (actively pushed upon event trigger)

5.1 Event Log Push (eventslog)

When the router detects configured event conditions, it automatically pushes event notifications to the configured MQTT topic. Each event type can independently configure whether MQTT push is enabled and the push target. The data field always contains three fields: event occurrence time, event type name, and event description text.

```
{
    "id": "6219A0780705",
    "status": "eventslog",
    "time": "2023-05-10T13:03:17Z",
    "code": "1",
    "sn": "6219A0780705",
    "data": {
       "time": "2023-05-10 21:03:17",
       "event_type": "lan up",
       "event_txt": "LAN1 connected"
    }
}
```


Field Description

Field	Type	Description
id	string	Device SN
status	string	Fixed as "eventslog"
time	string	Event occurrence time (UTC), format: YYYY-MM-DDTHH:MM:SSZ
code	string	Fixed as "1" (active reporting)
sn	string	Device SN
data.time	string	Event occurrence local time
data.event_type	string	Event type name
data.event_txt	string	Event description text

Note: The "data" field always contains three fields: "time", "event_type" and "event_txt".

MQTT (configured in the interface event settings)

Parameter	Description
mqtt_name	MQTT client name
topic	Publish topic
qos	QoS level
retain_flag	Message retain flag

6. System Control

6.1 Reboot System (reboot)

Remotely trigger a full router reboot. Upon receiving the request, the router first replies with a "rebooting" response (result=3), then executes the reboot. After the reboot completes and the MQTT Broker connection is re-established, the router automatically publishes a second response (result=1).

Topic: Same as system status query; user-configured req_topic / resp_topic.

Request Example

```
{
    "id": "1",
    "status": "reboot"
}
```

Response Example (First response, before reboot)

```
{
    "id": "1",
    "status": "reboot",
    "code": "2",
    "time": "2023-05-11T03:09:30Z",
    "sn": "6219A0780705",
    "result": 3,
    "resultmsg": "Rebooting system..."
}
```

Response Example (Second response, online after reboot)

```
{
    "id": "1",
    "status": "reboot",
    "code": "2",
    "time": "2023-05-11T03:09:30Z",
    "sn": "6219A0780705",
    "result": 1,
    "resultmsg": "Reboot system success"
}
```

result Field Description

result Value	Description
1	Execution successful
3	About to execute (rebooting in progress)

Note: After receiving the reboot request, the router first replies with a result=3 response, then executes the reboot. Once the reboot is complete and the device is back online, it automatically publishes a result=1 response.

6.2 Reboot Module (rebootmodel)

Remotely trigger a reboot of the router cellular module without affecting the router system itself. Suitable for scenarios requiring module reset such as cellular network anomalies or unresponsive modules. The router executes the module reboot immediately after replying; the entire process has only one response (result=3) with no second response.

Request Example

```
{
    "id": "1"
    "status": "rebootmodel"
}
```

Response Example

```
{
    "id": "1"
    "status": "rebootmodel",
    "code": "2",
    "time": "2023-05-11T03:09:30Z",
    "sn": "6219A0780705",
    "result": 3,
    "resultmsg": "Rebooting model..."
}
```

result Field Description

result Value	Description
3	About to execute (rebooting module)

Note: Module restart is an asynchronous operation. The restart is executed immediately once the router has responded, and no second response is sent.