SmartGen

MAKING CONTROL SMARTER

# CMM366A/B

# CLOUD MONITORING COMMUNICATION MODULE

# COMMUNICATION PROTOCOL

郑州众智科技股份有限公司
SMARTGEN(ZHENGZHOU)TECHNOLOGY CO.,LTD.

SmartGen **Registered trademark**
**No. 28 Xuemei Street, Zhengzhou, Henan, China**
**Tel:** +86-371-67988888/67981888/67992951
+86-371-67981000(overseas)
**Fax:** +86-371-67992952
**Web:** www.smartgen.com.cn/
www.smartgen.cn/
**Email:** sales@smartgen.cn



**Table 1 Software Version**

| Date | Version | Note |
|---|---|---|
| 2017-09-01 | 1.0 | Original release. |
| 2020-03-10 | 1.1 | Added CMM366B-4G cloud monitoring communication module, same protocol as V1.0. |
| 2023-12-23 | 1.2 | Modified 1.3.12 description in Table 4. |
| | | |

This manual is used for CMM366A-3G/CMM366A-ET/CMM366A-WIFI/CMM365A-2G/CMM366A-4G /CMM366B-4G cloud monitoring module.

Description of symbols used in this file is as below:

**Table 2 Symbols Description**

| Symbol | Instruction |
|---|---|
| NOTE | Highlights an essential element of a procedure to ensure correctness. |
| CAUTION | Indicates a procedure or practice, which, if not strictly observed, could result in damage or destruction of equipment. |
| WARNING | Indicates a procedure or practice, which could result in injury to personnel or loss of life if not followed correctly. |

# CONTENT

# 1 INTRODUCTION

This communication protocol describes the communication process protocol between this module GPRS and cloud server in details. Through this protocol, genset (with SCI) can be connected with Internet. After module logging in the cloud server, cloud server will send corresponding genset controller communication protocol to the module. Cloud monitoring module can obtain genset data information by RS485, USB, LINK, CAN or RS232, and transmit the obtained information to related cloud server by 4G wireless network. Users can monitor genset running state and check genset running records at real time via mobile APP (IOS or Android), PC etc. terminal equipments.

# 2 JSON BASIC RULES

— JSON (JavaScript Object Notation): JSON is a kind of lightweight data change format. Based on the subset of ECMASCript, it saves and indicates data by using text format which is completely independent of programming language.
— JSON data writing format: name/value pair;
— JSON can be number, character string, logic value, array, object, null.

# 3 DATA FRAME FORMAT

Both communication sides have initiator: Server, and Receiver: Device. Communication rule uses json format.

# 4 CLOUD MODEM REGISTER

## 4.1 INTRODUCTION

### 4.1.1 ILLUSTRATION

Cloud monitoring module sends registration message to the server, and the server verifies whether device has been registered based on the Hostid. If registered, update operation will be executed; if not, registration information will be saved. Then the real-time data will be uploaded to the server IP and port; the history data will be uploaded to the server IP and port, and the parameters corresponding to the device will return back to the modem.

### 4.1.2 INTERFACE DIRECTION

Sender: Device; Receiver: Server.

### 4.1.3 COMMUNICATION PROTOCOL

Address port: ServerIp
Protocol: Socket
Method: short connection
Transport format: JSON

### 4.1.4 INTERFACE DEFINITION

**Table 3 Message Request**

| No. | Name | Required | Type | Max Length | Description |
|---|---|---|---|---|---|
| 1 | method | Y | String | 20 | Login |
| 2 | params | Y | String | | Parameter JSON |
| 2.1 | Hostid | Y | String | 24 | Modem ID (24 bit) |
| 2.2 | hostname | Y | String | 20 | Modem name (UTF-8) |
| 2.3 | password | Y | String | 20 | Login password of modem |
| 2.4 | longitude | N | String | 20 | Modem longitude |
| 2.5 | latitude | N | String | 20 | Modem latitude |
| 2.6 | Gpsen | N | Num | 1 | 0: Baidu coordinate;<br>1: gps coordinate. |
| 2.7 | altitude | N | String | 20 | Modem altitude |
| 2.8 | moduletype | Y | String | 30 | Module type (Used to distinguish between different controllers)<br>Details please to see: *GENSET CONTROLLER MODEL LIST* |
| 2.9 | modulePort | Y | Num | 2 | 0: Disable; 1: LINK; 2: RS483; 3: RS232; 4: USB |
| 2.10 | moduleBaud | Y | Num | 0 | 0: 9600; 1: 19200; 2: 115200bit/s |

**Message request example:**

```
{
    "method": "login",
    "params":
    {
        "hostid": "AAAAA",
        "hostname": "Smartgen",
        "password": "980318",
        "longitude": "54",
        "latitude": "0",
        “gpsen”: 0,
        "moduletype": "HGM6120"
    }
}
```

**Table 4 Message Response**

| No. | Name | Type | Description | |
|---|---|---|---|---|
| 1.1 | Method | String | login | |
| 1.2 | Retcode | String | Reference to the return status code | |
| 1.3 | Result | | | |
| 1.3.1 | register | Int | Registration status:<br>1 success, 0 failure | |
| 1.3.2 | historic | String | History data upload address port | |
| 1.3.3 | liveData | String | Real-time data upload address port | |
| 1.3.4 | realtime | String | Return time UTC | |
| 1.3.6 | para_command | String | Commands of monitoring items, multiple commands are separated by semicolons(;) | Controller model, which corresponds to different codes. |
| 1.3.7 | con_command | String | Read configuration commands, multiple commands are separated by semicolons(;) | Please to see 1.3.6 |
| 1.3.8 | online_rate | Int | Client send rate on-line | Discussion result: e.g. send 3000 |
| 1.3.9 | offline_rate | Int | Client send rate off-line | Discussion result: same with online_rate |
| 1.3.11 | moduletype | String | Module model | Send controller model |
| 1.3.12 | data_mode | Int | Data upload mode | 0: Save flow mode (see: 4.5)<br>1: Full send mode (see: 4.4) |
| 1.3.13 | modulePort | Int | Num | 0: Disable; 1: LINK; 2: RS483; 3: RS232; 4: USB |
| 1.3.14 | moduleBaud | Int | Num | 0: 9600; 1: 19200; 2: 115200bit/s |

**Message response example:**


```
{
    "method": "login",
    "result":
    {
        "register": 1,
        "historic": "192.168.0.194:81",
        "liveData": "192.168.0.194:81",
        "realTime": "UTC",

        "para_command": ”01030000005045F6; 01030000005045F6”,
        "con_command": ”01030000005045F6; 01030000005045F6”,
        "online_rate": 3000,
        "offline_rate": 3000,
```

“modulePort”:2,
“moduleBaud”:0,
}
"retcode": "000000"
}


## 4.2 MODEM SENDING LONG CONNECTION

### 4.2.1 INTRODUCTION

After registration, modem will send long connection requirements actively to the server. After the connection is created, the server returns the status of the successful connection.

### 4.2.2 INTERFACE DIRECTION

Sender: Device; Receiver: Server.

### 4.2.3 COMMUNICATION PROTOCOL

Address port: ServerIp
Protocol: Socket
Method: long connection
Transport format: JSON

### 4.2.4 INTERFACE DEFINITION

**Table 5 Message Request**

| No. | Name | Required | Type | Max Length | Description |
|---|---|---|---|---|---|
| 1 | method | Y | String | 20 | LongCon |
| 2 | hostid | Y | String | 24 | Modem ID(24-bit) |

**Message request example:**

{"method": "LongCon ","hostid":"AAAAAAAAA"}

**Table 6 Message Response**

| No. | Name | Type | Description |
|---|---|---|---|
| 1 | retcode | String | Status |
| 2 | message | String | Message |
| 3 | method | String | LongCon |

**Message response example:**

{"method": "LongCon", "message": "OK","retcode": "000000"}

### 4.3 MODEM SENDING HEARTBEAT DATA PROTOCOL

#### 4.3.1 INTRODUCTION

After registration, modem regularly send heartbeat data to the server to check network connection status.

#### 4.3.2 INTERFACE DIRECTION

Sender: Device; Receiver: Server.

#### 4.3.3 COMMUNICATION PROTOCOL

Address port: ServerIp
Protocol: Socket
Method: long connection
Transport format: JSON

#### 4.3.4 INTERFACE DEFINITION

**Table 7 Message Request**

| No. | Name | Required | Type | Max Length | Description |
|---|---|---|---|---|---|
| 1 | method | Y | String | 20 | HB |
| 2 | params | Y | String | | Null |
| 3 | hostid | Y | String | 24 | Modem ID (24-bit) |

**Message request example:**

```
{
    "method": "HB",
    "hostid":"AAAAAAAAA",
    "params":""
}
```

**Table 8 Message Response**

| No. | Name | Type | Description |
|---|---|---|---|
| 1 | retcode | String | Status |
| 2 | message | String | Message |
| 2 | method | String | HB |

**Message response example:**

```
{"method": "HB","message": "OK","retcode": "000000"}
```

### 4.4 REAL-TIME DATA UPLOAD MONITORING DATA (FULL AMOUNT)

#### 4.4.1 INTRODUCTION

Modem sets different data upload frequencies based on whether the current client is online or not. And request results are saved to both the memory database and the history database.

Memory database storage format is monitor_hostid: {id,ymid,hostid,data,mask,status,itime, deviceNo}

When client requests real-time monitoring data, server will read the data from memory database directly.

For the storage of historical data, it is necessary to compare the time difference between the time when the historical data was last saved and the current time and the historical data storage frequency, and save the operation when the time difference is greater than the storage time of the historical data. And record the current time as the retention time of the most recent historical data.

#### 4.4.2 INTERFACE DIRECTION

Sender: Device; Receiver: Server.

#### 4.4.3 COMMUNICATION PROTOCOL

Address Port: ServerIp
Protocol: Socket
Method: long link
Transport format: JSON, command content is ModBus

#### 4.4.4 INTERFACE DEFINITION

**Table 9 Message Response**

| No. | Name | Type | Description | |
|---|---|---|---|---|
| 1 | method | String | Interface type, reqdata request data | |
| 2 | params | Object | ModBus content of data, multiple data need to be divided by semicolon (;). | |
| 3 | hostid | String | Modem ID | |

**Message response example:**

```
{
    "method": "reqdata ",
    "params": "",
    "hostid": "AAAAAAAA"
}
```

**Table 10 Message Request**

| No. | Name | Type | Description |
|---|---|---|---|
| 1 | retcode | String | Status |
| 2 | message | String | Message |
| 3 | method | String | Interface type, reqdata request data. |

**Message request example:**

```
{"method": "reqdata","message": "OK","retcode": "000000"}
```

### 4.5 REAL-TIME DATA UPLOAD MONITORING DATA (SEND CHANGEABLE DATA)

#### 4.5.1 INTRODUCTION

The modem uploads the changed data to the cloud server according to the data upload frequency set by the current client.

#### 4.5.2 INTERFACE DIRECTION

Sender: Device; Receiver: Server.

#### 4.5.3 COMMUNICATION PROTOCOL

Address port: ServerIp
Protocol: Socket
Method: long connect
Transport format: JSON, command content is MODBus

### 4.5.4 INTERFACE DEFINITION

**Table 11 Message Response**

| No. | Name | Type | Description | |
|---|---|---|---|---|
| 1 | method | String | Interface type, reqdatachange request data | |
| 2 | params | Object | Only upload changeable data parameters | |
| 3 | hostid | String | Modem ID | |
| 4 | AccessAlarm | Int | Access Alarm Input<br>0: Inactive; 1: Active | |
| 5 | FireAlarm | Int | Fire Alarm Input<br>0: Inactive; 1: Active | |
| 6 | AlarmInput | Int | Alarm Input<br>0: Inactive; 1: Active | |
| 7 | InhbitRemote | Int | Inhibit Remote Control Input<br>0: Inactive; 1: Active | |

**Message response example:**

```
{
    "method": "reqdatachange",
    "params": {
        "03": [
            {
                "0000": "0805"
            }
        ],
        "01": [
            {
                "0000": "0805"
            }
        ]
    },
    "hostid": "AAAAAAAA"
},
```

**Table 12 Message Request**

| No. | Name | Type | Description |
|---|---|---|---|
| 1 | retcode | String | Status |
| 2 | message | String | Message |
| 3 | method | String | Interface type, reqdata request data |

**Message request example:**

{"method": "reqdata","message": "OK","retcode": "000000"}

### 4.6 WRITE CONFIGURATION DATA

#### 4.6.1 INTRODUCTION

After receiving command of writing configuration, modem sends a write configuration command to the controller. When the modem receives the return result from the controller, it splices the command and execution status (separate by comma, like: 0000000000,1).

#### 4.6.2 INTERFACE DIRECTION

Sender: Server; Receiver: Device.

#### 4.6.3 COMMUNICATION PROTOCOL

Address port: ServerIp
Protocol: Socket
Method: long link
Transport format: JSON, command content is ModBus

#### 4.6.4 INTERFACE DEFINITION

**Table 13 Message Request**

| No. | Name | Type | Description | |
|---|---|---|---|---|
| 1 | result | String | ModBus protocol content | Definite by controller series |
| 2 | datatype | String | ModBus data format | Definite by controller series |
| 3 | uid | String | User id of requester | Arbitrary send |
| 4 | method | String | Request method writeConfig | |
| 5 | modulePort | Num | 0: Disabled; 1: LINK; 2: RS483; 3: RS232; 4: USB | |
| 6 | moduleBaud | Num | 0: 9600; 1: 19200; 2: 115200bit/s | |

**Write configuration message request example:**

```
{
    "result": "000000000000",
    "datatype": 1,
    "uid": "1",
    "method": "writeConfig”
}
```

**Write modem port number or baud rate message request example:**

```
{
    "modulePort": 1,
    "moduleBaud": 0,
    "datatype":1,
```

```
    "uid":"1",
    "method": "writeConfig”
}
```

**Table 14 Message Response**

| No. | Name | Type | Description |
|---|---|---|---|
| 1 | method | String | Interface type, writeConfig request data. |
| 2 | params | Object | Command, status like: (0000000000,1) 1 stands for success, 0 stands for failure |
| 3 | hostid | String | Modem ID |
| 4 | uid | String | User id of requester |
| 5 | modulePort | Num | 0: Disabled; 1: LINK; 2: RS483; 3: RS232; 4: USB |
| 6 | moduleBaud | Num | 0: 9600; 1: 19200; 2: 115200bit/s |

**Write configuration message response example:**

```
{
    "method": "writeConfig",
    "params": "01010101,1",
    "hostid": "AAAAAAAA",
    "uid": "1"
}
```

**Write modem port number or baud rate message response example:**

```
{
    "method ": "writeConfig",
    "modulePort": 1,
    "moduleBaud": 0,
    "hostid": "AAAAAAAA",
    "uid": "1"
}
```

## 4.7 TIME SERVICE INTERFACE

### 4.7.1 INTRODUCTION

Modem periodically requests the server for network timing to ensure that the modem synchronizes with the server clock.

### 4.7.2 INTERFACE DIRECTION

Sender: Device; Receiver: Server.

### 4.7.3 COMMUNICATION PROTOCOL

Address port: ServerIp
Protocol: Socket
Method: long link
Transport format: JSON

### 4.7.4 INTERFACE DEFINITION

**Table 15 Message Response**

| No. | Name | Type | Description |
|---|---|---|---|
| 1 | method | String | Interface type, timing request data. |
| 2 | hostid | String | Modem ID |

**Message response example:**

```
{
    "method": "timing",
    "hostid": "AAAAAAAA",
}
```

**Table 16 Message Request**

| No. | Name | Type | Description |
|---|---|---|---|
| 1 | message | String | Message content |
| 2 | retcode | String | Message status |
| 3 | method | String | Request method: timing |
| 4 | result | String | Current time: UTC long |

**Message request example:**

```
{
    "message": "ok",
    "retcode":"000000",
    "method":" timing",
    "result":" 1497811080077"
}
```

### 4.8 RESET INTERFACE

#### 4.8.1 INTRODUCTION

Configurate modem.

After server sending reset command, modem activates registration function.

#### 4.8.2 INTERFACE DIRECTION

Sender: Server; Receiver: Device.

#### 4.8.3 COMMUNICATION PROTOCOL

Address Port: ServerIp

Protocol: Socket

Method: long link

Transport format: JSON

#### 4.8.4 INTERFACE DEFINITION

**Table 17 Message Request**

| No. | Name | Type | Description |
|---|---|---|---|
| 1 | message | String | Message content |
| 2 | retcode | String | Message status |
| 3 | method | String | Request method: reset |

**Message request example:**

```
{
    "message": "ok",
    "retcode": "000000",
    "method": "reset”,
}
```

**Table 18 Message Response**

| No. | Name | Type | Description |
|---|---|---|---|
| 1 | method | String | Interface type, reset request data |
| 2 | hostid | String | Modem ID |

**Message Response example:**

```
{
    "method": "reset",
    "hostid": "AAAAAAAA",
}
```

### 4.9 DATA PROTOCOL

**Table 19 Data Format Sent to The Server by Modem (Socket)**

| No. | Name | Type | Description |
|---|---|---|---|
| 1 | method | String | Interface type |
| 2 | params | Object | Data object can be strings, or Json object, or ModBus protocol content. |
| 3 | hostid | String | Modem ID |
| 4 | uid | String | User ID |

**Table 20 Data Format Sent to The Modem by Server (Socket)**

| No. | Name | Type | Description |
|---|---|---|---|
| 1 | result | Object | Data object can be strings, or Json object, or ModBus protocol content |
| 2 | retcode | String | |
| 3 | uid | String | User ID(6-bit) |
| 4 | datatype | Int | Result data type, 0: string data; 1: ModBus data; 2 Json object |
| 5 | message | String | Maybe null |
| 6 | packetnum | Int | Parameter packet number |