Gizwits Platform Standard Protocol

Device and Cloud

(v4.0.5)

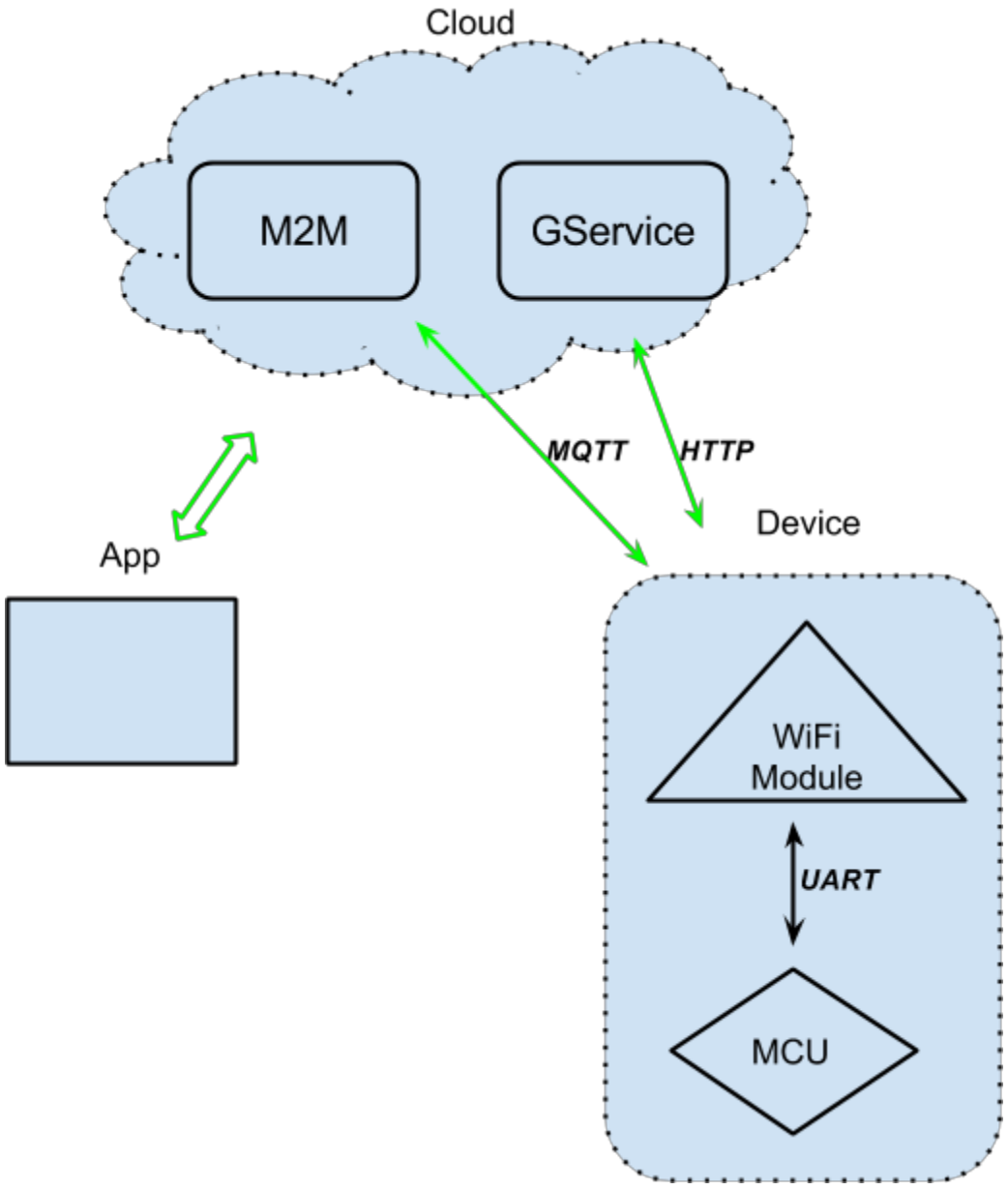
Revision History

Version	Content Modification	By	Date
4.0.0	v4 protocol	Johnson	11/20/2014
4.0.1	1. Add “4.8 Notify Device the Count of Online Apps Changed” (p7) 2. Fix the Device OTA “Device receives OTA push notification” message pattern	Johnson	11/29/2014
4.0.2	1. Add “4.8 Notify Device the Count of Online Apps Changed” (p7) 2. Update the command code of “4.9 Device Requests the Count of Online Apps”, and cloud won’t push the count in interval (p7)	Johnson	12/05/2014
4.0.3	Fix document error: Remove the message field “Business Msg Len” from Business Message Transparent Transmission section.	Johnson	12/15/2014
4.0.4	Add device unregister API. (p4)	Johnson	12/17/2014
4.0.5	1. Add “Business Message Transparent Transmission (with ACK)”. (p7) 2. Add “Device OTA (v4.1)” (p5)	Johnson	04/18/2015

Table of Content

1. Communication Model

Devices communicate with cloud through HTTP and MQTT protocol. The HTTP service is for Device Registration, Provisioning and OTA Firmware Upgrade etc. MQTT is for devices report status and receive remote control commands. MQTT is a machine-to-machine (M2M)/"Internet of Things" connectivity protocol, which is based on TCP. More details please to MQTT_V3.1_Protocol_Specific.pdf.



GService provides HTTP service and M2M provides MQTT real-time communication.

2. Working Flow

The working flow between device and cloud mainly include the following processes.

- **Device registration.** On the device very first power on, device must activate(register) itself and get the DID(Device ID) from cloud.
- **Device provision.** Before connecting to M2M server, device must firstly get the M2M server IP and port through HTTP provision API.
- **Device OTA**. On every startup, device should check out if there is new firmware that it shoud upgrade to.

- **Device report status and receiv remote control commands.** Device connect and keep the MQTT connection with M2M server, report status to cloud in interval or after status changed, and receive remote control commands from App or cloud.

3. Convention

3.1. Reading Instructions

- **Variable Len**: Variable length, represents the number of bytes remaining within the current message, could be 1 to 4 bytes. Details please refer to MQTT_V3.1_Protocol_Specific.pdf page 6 “Remaining Length”.
- **Len**: Length, indicates the field value length. 1 byte or 2 bytes. If is 2 bytes, endianness is big-endian.

3.2. Cloud GService Address and Port(HTTP)

- Address: api.gizwits.com
- Port: 80

3.3. After connecting with M2M server, device should subscribe the following topics to receive messages from cloud.

- Receive message response: *ser2cli_res/<DID>* (<DID> stands for device DID)
- Receive push notification: *ser2cli_noti/<ProductKey>* (<ProductKey> stands for ProductKey of product)
- Receive remote control commands: *app2dev/<DID>/#* (<DID> stands for device DID)

4. Protocol

4.1. Device Register

Device must register(activate) itself before working with Cloud.

Device register, Device ⇒ GService, HTTP. Device post ProductKey, MAC and the dynamic generating Passcode to GService and GService returns DID(deivce ID). Device saves the DID and Passcode.
Please refer to http://docs.gizwits.apiary.io/reference/api-for-devices/devices/create-device.

4.2. Device Unregister

Once the device is reset, it must unregister(disable) the old DID.

Device unregister, Device ⇒ GService, HTTP.
Please refer to http://docs.gizwits.apiary.io/#reference/api-for-devices/devices/disable-device.

4.3. Device Provision

Before connecting to M2M server, device must firstly get the M2M server IP and port through HTTP provision API.

Device provision, Device ⇒ GService, HTTP.
Please refer to
http://docs.gizwits.apiary.io/reference/api-for-devices/device-provision/device-provision.

4.4. Device OTA (v4.0)

On every startup, device should check out if there is new firmware that it should upgrade to.

Device checkouts latest firmware id, Device ⇒ GService, HTTP. Cloud will return the latest firmware id and the download url. Device checks if the local saving current firmware id is different from the latest firmware id, if yes then download the latest firmware and do the OTA upgrade.
Please refer to
http://docs.gizwits.apiary.io/reference/api-for-devices/ota-get-target-fid/get-target-fid.

When device is online and working, M2M server would push OTA notification to it. Device does the OTA upgrade if the target firmware id is not matched with the local saving current firmware id.
Device receives OTA push notification, M2M Server ⇒ Device, TCP, MQTT PUBLISH. Topic is *ser2cli_res/<DID>* (<DID> stands for device DID).

Order	Field Name	Byte Len(B)	Description
1	Fixed Header	4	0x00000003
2	Command	2	0x020E
3	Firmware Type	1	1 for WiFi, 2 for MCU
4	Target Firmware ID	4	Device does the OTA upgrade if the target firmware id is not matched with the local saving current firmware id
5	Download Url Len	2	len(Download Url)
6	Download Url	max 512	Firmware download link

4.5. Device OTA (v4.1)

On every startup, device should check out if there is new firmware that it should upgrade to.

Device checkouts latest firmware soft version, Device ⇒ GService, HTTP. Cloud will return the latest firmware soft version and the download url. Device checks if the local saving current soft version is different from the latest soft version, if yes then download the latest firmware and do the OTA upgrade.

When device is online and working, M2M server would push OTA notification to it. Device checkouts the latest firmware soft version and do the OTA upgrade.
Device receives OTA push notification, M2M Server ⇒ Device, TCP, MQTT PUBLISH. Topic is *ser2cli_res/<DID>* (<DID> stands for device DID).

Order	Field Name	Byte Len(B)	Description
1	Fixed Header	4	0x00000003
2	Command	2	0x0211
3	Firmware Type	1	1 for WiFi, 2 for MCU

4.6. Device Login Cloud(MQTT)

After device has connected to the M2M server, it needs to send the MQTT CONNECT message to login M2M server. The M2M server address and port are fetch from the Device Provision API.

Device login M2M Server, Device ⇒ M2M server, MQTT CONNECT. The request parameter values should refer to the following table.

MQTT Parameter	Content
Client Identifier	Device DID
User Name	Device DID
Password	Device Passcode
Keep Alive timer	Max 600 seconds, recommend 120 seconds. If exceed the max value, server will treat the value as max value

Details please refer to MQTT_V3.1_Protocol_Specific.pdf page 15 “3.1. CONNECT”.

M2M Server response. M2M server ⇒ Device, MQTT CONNACK.
Please refer to MQTT_V3.1_Protocol_Specific.pdf page 18 “3.2. CONNACK”.

4.7. Heartbeat

After device(we call it “client” below) connected with M2M server, it must send ping to the server in interval. Server replies with ping response.

Client sends ping, Client ⇒ M2M server, MQTT PINGREQ.
Please refer to MQTT_V3.1_Protocol_Specific.pdf page 33, “3.12. PINGREQ”

Server response, M2M server ⇒ Client, MQTT PINGRESP.
Please refer to MQTT_V3.1_Protocol_Specific.pdf page 33, “3.13. PINGRESP”

The interval that client sends ping to server must be equal or less than 180 times within 180 seconds. If the ping frequency is quick than this, server would close the MQTT connection. Recommend 1 ping per 60 seconds.

If client did not receive the ping response for 2 times, it could consider the connection with the server is broken and could start reconnecting.

About the reconnection policy. After the connection with the server is broken, client will try to reconnect the server. If the first trying is failed, it should wait 10 seconds. If the second trying is failed again, it should wait 20 seconds, and so on. Each reconnection trying, the client should wait 10 more seconds. If the client reboots, the waiting starts from 0 seconds.

4.8. Business Message Transparent Transmission

After device login M2M server, business commands could be transmitted transparently between App and device MCU pass-through cloud or WiFi module. The business commands is specified to the device specification.

Device receive business message. M2M server ⇒ Device, MQTT PUBLISH. Topic is *app2dev/<DID>/<AppClientId>* (<DID> stands for device DID, <AppClientId> stands for the MQTT Client Identifier of App which sends the message, <AppClientId> will be used when device replies the message, please see below).

Order	Field Name	Byte Len(B)	Description
1	Fixed Header	4	0x00000003
2	Variable Len	1~4	len(Flag...Business Msg)
3	Flags	1	0x00
4	Command	2	0x0090

5	Business Msg	max 65535	Business Message

Device replies the App for the business message. Device ⇒ M2M server, MQTT PUBLISH. Topic is *dev2app/<DID>/<AppClientId>*. If device wants to send business message to all online Apps that have bound this device, the topic is *dev2app/<DID>* (<DID> stands for device DID, <AppClientId> stands for the MQTT Client Identifier of App which will receive the message, <AppClientId> is carried in the request message topic, please see above).

Order	Field Name	Byte Len(B)	Description
1	Fixed Header	4	0x00000003
2	Variable Len	1~4	len(Flag...Business Message)
3	Flags	1	0x00
4	Command	2	0x0091
5	Business Msg	max 65535	Business Message

4.9. Business Message Transparent Transmission (with ACK)

After device login M2M server, business commands could be transmitted transparently between App and device MCU pass-through cloud or WiFi module. The business commands is specified to the device specification.

Send business message that need an ACK. M2M server ⇒ Device or Device ⇒ M2M server, MQTT PUBLISH. Topic is *app2dev/<DID>/<AppClientId>* if send from M2M server to Device or *dev2app/<DID>* or *dev2app/<DID>/<AppClientId>* if send from Device to M2M server (<DID> stands for device DID, <AppClientId> stands for the MQTT Client Identifier of App which sends or receives the message).

Order	Field Name	Byte Len(B)	Description
1	Fixed Header	4	0x00000003
2	Variable Len	1~4	len(Flag...Business Msg)
3	Flags	1	0x00
4	Command	2	0x0093
5	SN	4	The serial number of this message. The Sn is used for message ACK

6	Business Msg	max 65535	Business Message

The receiver acknowledges the business message (ACK). Device ⇒ M2M server or M2M server ⇒ Device, MQTT PUBLISH. Topic is *app2dev/<DID>/<AppClientId>* if send from M2M server to Device or *dev2app/<DID>* or *dev2app/<DID>/<AppClientId>* if send from Device to M2M server (<DID> stands for device DID, <AppClientId> stands for the MQTT Client Identifier of App which sends or receives the message).

Order	Field Name	Byte Len(B)	Description
1	Fixed Header	4	0x00000003
2	Variable Len	1~4	len(Flag...Business Message)
3	Flags	1	0x00
4	Command	2	0x0094
5	SN	4	The SN is used to indicate which message to ACK
6	Business Msg	max 65535	Business Message

Note that, M2M server will never ACK any message, the responsibility of message ACK is receiver which is Device or App.

4.10. Set Device Log Level and LED Switches

App could set device log level and LED switches.

Device receives the setting request. M2M server ⇒ Device, MQTT PUBLISH. Topic is *app2dev/<DID>/<AppClientId>* (<DID> stands for device DID, <AppClientId> stands for the MQTT Client Identifier of App which sends the message, <AppClientId> will be used when device replies the message, please see below).

Order	Field Name	Byte Len(B)	Description
1	Fixed Header	4	0x00000003
2	Variable Len	1~4	len(Flag...Values(bit0~bit7))
3	Flags	1	0x00
4	Command	2	0x0010
5	Values(bit8~bit15)	1	The order of bit8 ~ bit15 is from low bit to high bit. For the function definition please refer to the next field.

6	Values(bit0~bit7)	1	The order of bit0 ~ bit7 is from low bit to high bit. The bit0 ~ bit15 function definition is described as below: ● bit0: Error log level, 0 is off, 1 is on ● bit1: Warning log level, 0 is off, 1 is on ● bit2: Info log level, 0 is off, 1 is on ● bit3: WiFi module LEDs switches, 0 is off, 1 is on ● bit4~bit15: reserved

Device response. Device ⇒ M2M server, MQTT PUBLISH. Topic is *dev2app/<DID>/<AppClientId>* (<DID> stands for device DID, <AppClientId> stands for the MQTT Client Identifier of App which will receive the message, <AppClientId> is carried in the request message topic, please see above).

Order	Field Name	Byte Len(B)	Description
1	Fixed Header	4	0x00000003
2	Variable Len	1~4	len(Flag...Command)
3	Flags	1	0x00
4	Command	2	0x0011

4.11. Device Requests the Count of Online Apps

Device could request the count of online Apps after connecting to the M2M server.

Device requests the count of online Apps. Device ⇒ M2M server, MQTT PUBLISH. Topic is *ser2cli_req*.

Order	Field Name	Byte Len(B)	Description
1	Fixed Header	4	0x00000003
2	Command	2	0x020F

4.12. Notify Device the Count of Online Apps Changed

Cloud sends the count of online Apps to devices if device requests it or whenever the count is changed.

Device receives the count of online Apps. M2M server ⇒ Device, MQTT PUBLISH. Topic is *ser2cli_res/<DID>* (<DID> stands for device DID).

Order	Field Name	Byte Len(B)	Description

1	Fixed Header	4	0x00000003
2	Command	2	0x0210
3	Count of Online Apps	2	Nonnegative integer, big-endian

4.13. Device Send Logs

If some log levels are turned on, device will send corresponding level logs to M2M server, which would send the logs to online Apps that have bound to this device.

Device send logs. Device ⇒ M2M server, MQTT PUBLISH. Topic is *dev2app/<DID>* (<DID> stands for device DID).

Order	Field Name	Byte Len(B)	Description
1	Fixed Header	4	0x00000003
2	Variable Len	1~4	len(Flag...Log Content)
3	Flags	1	0x00
4	Command	2	0x0012
5	Log Level	1	1 is Error, 2 is Warning, 3 is Info
6	Tags Len	2	len(Tags)
7	Tags	max 256	Tags of the log
8	Source Len	2	len(Source)
9	Source	max 32	Indicates where the log is from
10	Log Content Len	2	len(Log Content)
11	Log Content	max 65535	UTF8 encoding string