Delta Controls IoT Gateway

Google Cloud Platform – MQTT Bridge

Application Guide

Document version: 0.5 – (compatible with V1.2.4 and later of the module)

Contents

1 Google IoT Gateway

Documentation for the use of the python module that provides connectivity to Google IoT Core using the preferred Google connection techniques of a Gateway and proxy devices. The gateway supports proxy devices that use the BACnet or MODbus TCP protocols.

1.1 Overview

This module is an application of IoT Connectivetly using the MQTT library within the Version 4.9 (and later) firmware. The concept that it is designed to fulfil is described by Google in their how-to guides for Google Cloud (Using gateways | Cloud IoT Core Documentation | Google Cloud)

This method presumes that a gateway device establishes a secure connection to the Google MQTT Bridge using TLS and a JSON Web Token created with a Private/Public encryption key pair using openssl. This is implemented in V4.9.5002 and V4.11.5002 firmware, so that the device creates the key pair from a pre-loaded root certificate and only exposes the public key to allow connection to GCP IoT Core.

The connection to IoT Core requires a Google Cloud Platform Project, with a registry (in the appropriate cloud region) and then one or more gateway(s) and one or more device(s) which the physical gateway will proxy for in the cloud connection. The only setup on the physical gateway is the connection to the IoT Core, which is defined in any CSV Object that must be named exactly (including case capitalisation) as:

1.1.1 Google Cloud IoT Gateway Configuration

The object must contain a JSON block that defines the key: value pairs as shown in the table and example below

Key	Description	Default	Optional / Required	Example
hostname	Name of IP host		Required	"mqtt.googleapis.com"
tcpPort	TCP port at host		Required	8883
project	GCP Project		Required	"bacnet-gateway"
location	Cloud region of GCP Project registry		Required	"europe-west1"

auth-type	Authentication method to use for MQTT connection "jwt" = GCP "none" = MQTT3.1.1 with no authentication "simple" = MQTT3.1.1 with simple username / password authentication	jwt	Optional	"jwt" / "none" / "simple"
device	Device name of the Gateway within the GCP Project registry		Required	"GAT-100"
publish-topic	Pointset telemetry topic name for publish		Optional	"/devices/{{}}/events/pointset"
state-topic	State telemetry topic name for public		Optional	"/devices/{{}}/state"
debug	Diagnostic information level for debug info in the CSV.Description property 0 = Debug Off 1 = Minimal Info 2 = Normal Info 3 = Detailed Info 4 = Full Diagnostic Info	0	Optional	0-4
maxupdatems	Maximum update rate between MQTT requests (in milliseconds)	1000	Optional	1000 = 1 second
poll-interval	Polling interval between publish of proxy device telemetry data (in milliseconds)	900000	Optional	900000 = 900 seconds = 15 minutes
rpm	Use BACnet Read Property Multiple to collect proxy device telemetry data	1	Optional	1 = Use RPM 0 = Do not use RPM
cloud-write	Allow Cloud-to-Device commands (BACnet Write)	0	Optional	1 = Allow C2D 0 - Do not allow C2D
write-priority	BACnet priority level for Cloud write back	9	Optional	1-16
relinquish-always	When off (0) only Cloud-to-Device write backs to individual points that are subsequently cleared will be relinquished. When on all points are relinquished regardless of any prior write backs	0	Optional	0 = Relinquish all 1 = Relinquish changed
jwt_exp_mins	Interval between required refresh of JSON Web Tokens for connection (minutes)	20	Optional	20 = 20 minutes

Example of minimally specified configuration file

```
{
  "hostName": "mqtt.googleapis.com",
  "tcpPort": 8883,
  "location": "us-central1",
  "project": "bacnet-gateway",
  "registry": "registrar-us",
  "device": "GAT-100",
}
```

Example of fully specified configuration file

```
{
  "hostName": "mqtt.googleapis.com",
  "tcpPort": 8883,
  "location": "us-central1",
  "project": "bacnet-gateway",
  "registry": "registrar-us",
  "auth-type": "jwt",
  "device": "GAT-100",
  "publish-topic": "/devices/{{}}/events/pointset",
  "state-topic": "/devices/{{}}/state",
  "debug": 4,
  "maxupdatems": 2000,
  "poll-interval": 60000,
  "rpm": 0,
  "cloud-write": 1,
  "write-priority": 7,
  "relinquish-always": 1,
  "jwt_exp_mins": 30
}
```

1.2 Security

In order to connect to the Google IOT Core three pieces are needed:

Device Private Key: self-generated on the device. The Python module will generate this on the device if the file does not exist. This file is not exposed externally to maintain security.

Device Public Key: self-generated on the device. The Python module will generate this on the device when it discovers any FIL object named "**Google Cloud IoT Gateway Public Key**" (exactly). This can be downloaded from the device from the FIL dialogue in enteliWEB and uploaded to the RSA key of the Gateway device in the GCP Project Registry. It is also copied into the FIL.Description property for copy/paste.

Google Root Certificate: A new Google Root certificate can be uploaded to any FIL object created on the device and exactly named "**Google Cloud IoT Gateway Roots**". Loading a new root certificate will cause the Python module to recreate the private / public key pair.

2 Usage

Once the connection to GCP IoT Core has been established then IoT Core will publish to a topic named "**devices/[gateway device]/config**", the Python module will have already subscribed to this and will be expecting to receive the payload defined in the UDMI Schema (UDMI Gateway Config JSON Block). This will define the devices that the gateway is to proxy for and the names must match names of devices created in the GCP Project Registry and that have been 'bound' to the Gateway.

Following receipt of a valid gateway config message on the MQTT topic the device will publish its status to a topic named "**devices/[gateway device]/status**".

The Gateway will then construct a list of devices that it is to proxy for from the data in the gateway config message. Note that the gateway can proxy for itself and so act as both a BAS Controller and an IoT Gateway. The module will then iterate through the list and publish to the topic "**devices/[proxy device]/attach**", upon a successful attachment IoT Core will publish to a topic named "**devices/[proxy device]/config**", the Python module will have already subscribed to this and will be expecting to receive the payload defined in the UDMI Schema (UDMI Device Config JSON Block). This will define the BACnet address of the proxy device in ['localnet']['subsystem']['bacnet']['localid'] and the points data for telemetry in the ['pointset']

The device will publish the status for each proxy to a topic named "**devices/[proxy device]/status**" in accordance with the UDMI Schema Proxy Status.

Once attached and then at the relevant poll interval the Python module will collect the BACnet data into one JSON pointset payload defined in the UDMI Schema (UDMI Device Event Pointset Telemetry JSON Block) and publish this to the topic "**devices/[proxy device]/events/pointset**"

2.1 Gateway Status Telemetry

The JSON payload that is published to the "**devices/[gateway device]/status**" defaults to:

```
{
  "version": 1,
  "timestamp": "<TIME ISO-8601 UTC>",
  "system": {
    "make_model": "<BACNET DEV.Model_Name>",
    "firmware": {
      "version": "<BACNET DEV.Application_Software_Version>"
    },
    "serial_no": "<BACNET DEV.Serial_Number>",
    "last_config": "",
    "operational": "<BACNET DEV.System_Status>"
  }
  "gateway": {
    "error_ids": []
  }
}
```

This can be modified using a template that resides in a CSV Object that must be named exactly (including case capitalisation) as:

2.1.1 Google Cloud IoT Gateway State Telemetry

The object must contain a JSON block that defines the key: value pairs that depict a frame for the telemetry. The device includes a pre-processor that will parse the template prior to publishing and replace any references to <BACnet > properties or < Time> values.

2.2 Proxy Devices Status Telemetry

The JSON payload that is published to the **"devices/[proxy device]/status"** defaults to:

```
{
  "version": 1,
  "timestamp": "<TIME ISO-8601 UTC>",
  "system": {
    "make_model": "<BACNET DEV.Model_Name>",
    "firmware": {
      "version": "<BACNET DEV.Application_Software_Version>"
    },
    "serial_no": "N/A",
    "last_config": "",
    "operational": "<BACNET DEV.System_Status>"
  },
  "pointset": {
    "points": {}
  }
}
```

This can be modified using a template that resides in a CSV Object that must be named exactly (including case capitalisation) as:

2.2.1 Google Cloud IoT State Telemetry

The object must contain a JSON block that defines the key: value pairs that depict a frame for the telemetry. The device includes a pre-processor that will parse the template prior to publishing and replace any references to <BACnet > properties or <Time> values.

3 Limits

There will be real world limits associated with both the amount of data can be collected in one Read Property Multiple request from the BACnet Server on the device and with the limits for the amount of point data can be defined within a single pointset of the UDMI.

The UDMI limit has not been set in stone, but the figure of 150 has been postulated by Google themselves, which is well within the constraints of RPM. We need to be mindful of this and any use of this gateway should always consider both these physical limits and also any logical limits that could be run into that may depend on the network architecture of the site, constraints of the physical devices themselves and network traffic management. Good network practice must be adhered to as well.

4 Cloud-Side Configuration

There will be occasions when the device is logically placed behind cyber security curtains and so (re)configuration and setup will be difficult. In order to make this process easier and possible to do without the need for a physical visit to site support for a number of GCP 'Commands' in the device gateway shadow on GCP have been implemented in V0.10.1

4.1 Load-Module

Format

```
{
  "load-module": "delta-google-iot-gw.py",
  "loader-repo": "[enter url to new module here]"
  "signature": "[enter file digital signature]"
}
```

Use this gateway command to send a command that will trigger a pull of a new python module. Use this if the module needs to be updated to a later version. The command must include a pre-generated electronic signature, contact Delta Controls for information on how to obtain this.

Note that this command will first rename the current version of the module to *[filename.bak]* (overwriting any exisiting file of that name) and then replace with the new version downloaded from the url supplied.

Further note that Python modules run from memory and so the new version will not take effect until it is reloaded, e.g. by a device reset. Alternatively and as the Python module is designed to exit following a successful load then it will restart itself after a predefined interval as specified in the Python Module Loader module (typically within one minute)

4.2 Reboot

Format

```
{"reboot": true}
```

Use this gateway command to force a reset of the physical device.

4.3 Activate Network Change

Format

```
{"activate-network-change": true}
```

Use this gateway command to force an activate network change on the physical device.

4.4 Restore-Module

Format

```
{"restore-module": "delta-google-iot-gw.py"}
```

Use this gateway command to restore the backup. The current version will be replaced with the backup from *[filename.bak]*

Note once this command is executed the old backup is lost.

Further note that Python modules run from memory and so the new version will not take effect until it is reloaded, e.g. by a device reset.

4.5 Write-Property

Format

```
{
  "write-property": {
    "CSV9991.Relinquish_Default":
"{'hostName':'mqtt.googleapis.com','tcpPort':8883,'location':'us-
central1','project':'bacnet-gateway','registry':'US-MTV-918R','device':'GAT-
1','debug':2,'cloud-write':1}"
  }
}
```

Use this gateway command to write to a property of a BACnet object in the gateway device, the example above shows a possible use case for this in modifying the GCP Connection parameters stored in the "Google Cloud IoT Gateway Configuration" CSV Object.

Thus a gateway device connection could be 'moved' from a test GCP registry to a live registry (or vice-versa) without the need for any on-premesis configuration, which may be difficult or require the physical attendance of a Delta Tech.