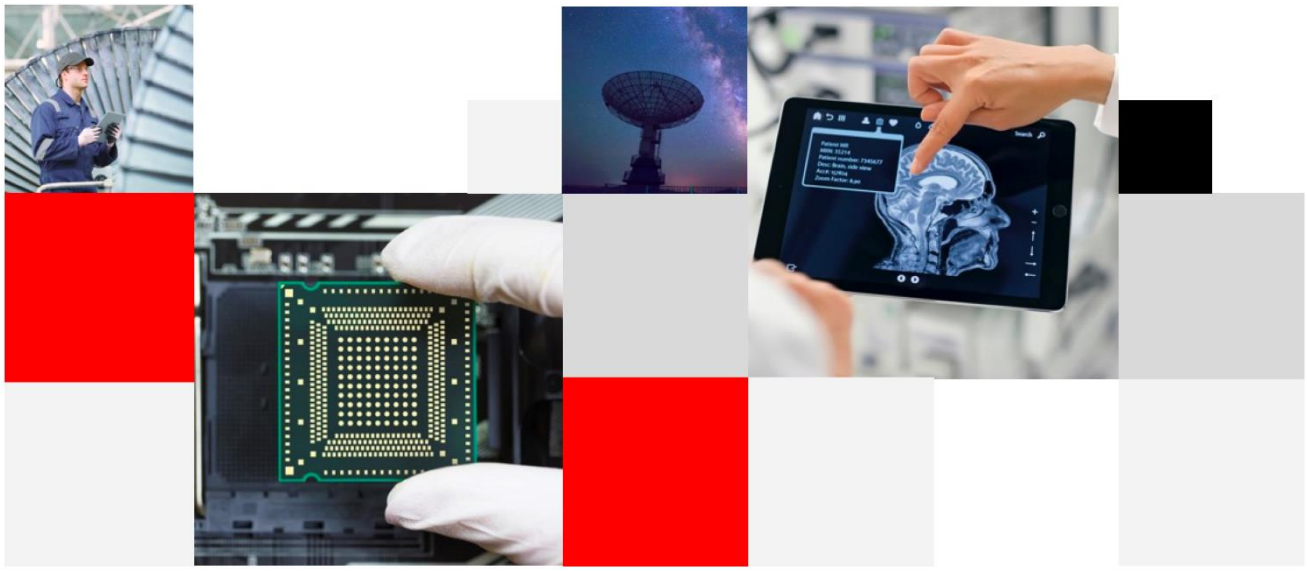
BG95&BG77&BG600L Series
QuecOpen Azure IoT Hub Access Guide

LPWA Module Series

Version: 1.1

Date: 2022-05-16

Status: Released



Build a Smarter World

Our aim is to provide customers with timely and comprehensive service. For any assistance, please contact our company headquarters:

Quectel Wireless Solutions Co., Ltd.
Building 5, Shanghai Business Park Phase III (Area B), No.1016 Tianlin Road, Minhang District, Shanghai 200233, China
Tel: +86 21 5108 6236
Email: info@quectel.com

Or our local office. For more information, visit:
http://www.quectel.com/support/sales.htm.

For technical support, or to report documentation errors, visit:
http://www.quectel.com/support/technical.htm
Or send an email to: support@quectel.com.

General Notes

Quectel offers the information as a service to its customers. The information provided is based upon customers' requirements. Quectel makes every effort to ensure the quality of the information it makes available. Quectel does not make any warranty as to the information contained herein, and does not accept any liability for any injury, loss or damage of any kind incurred by the use of or reliance upon the information. All information supplied herein is subject to change without prior notice.

Disclaimer

While Quectel has made efforts to ensure that the functions and features under development are free from errors, it is possible that these functions and features could contain errors, inaccuracies and omissions. Unless otherwise provided by valid agreement, Quectel makes no warranties of any kind, implied or express, with respect to the use of features and functions under development. To the maximum extent permitted by law, Quectel excludes all liability for any loss or damage suffered in connection with the use of the functions and features under development, regardless of whether such loss or damage may have been foreseeable.

Duty of Confidentiality

The Receiving Party shall keep confidential all documentation and information provided by Quectel, except when the specific permission has been granted by Quectel. The Receiving Party shall not access or use Quectel's documentation and information for any purpose except as expressly provided herein. Furthermore, the Receiving Party shall not disclose any of the Quectel's documentation and information to any third party without prior written consent by Quectel. Quectel reserves the right to take legal action for any noncompliance with the above requirements, unauthorized use, or other illegal or malicious use of the documentation and information.

Copyright

About the Document

Revision History

Version	Date	Author	Description
-	2021-01-20	Billie XING	Creation of the document
1.0	2021-03-26	Billie XING	First official release
1.1	2022-05-16	Terrence YANG	Add Provision a simulated symmetry key device Add Provision a simulated X.509 certificate device Add IoT Central description

Contents

Table Index

Figure Index

1 Introduction

Quectel BG95 series, BG77 and BG600L-M3 modules support QuecOpen® solution. QuecOpen® is an open-source embedded development platform based on ThreadX system. It is intended to simplify the design and development of IoT applications. For more information on QuecOpen®, see *Quectel_BG95&BG77&BG600L_Series_QuecOpen_Application_Note*.

Azure IoT Hub is a cloud-hosted managed service that acts as a central message hub for two-way communication between your IoT application and the devices it manages. You can use Azure IoT Hub to build IoT solutions with reliable and secure communications between millions of IoT devices and a cloud-hosted solution back end.

In order that your equipment connects to Azure IoT hub easily, you can use the Azure IoT Hub Device SDK libraries to build an application that runs on your devices and interacts with the IoT Hub. Quectel BG95 series, BG77 and BG600L-M3 modules are integrated with the Azure IoT Hub Device SDK, and you can use it to build QuecOpen® applications. Azure IoT Hub gives you a secure communication channel for your device to send data, per-device authentication enables each device to connect securely to the IoT Hub and for each device to be managed securely. QuecOpen® facilitates connecting to Azure IoT Hub with two authentication types:

- SAS token-based authentication, which makes you get started with building your IoT solution quickly.
- Device enrollments with Azure Device Provisioning Service.

This document explains how to use these two authentication types in QuecOpen® Azure IoT Hub Device SDK to communicate with the Azure IoT Cloud.

1.1. Applicable Modules

Table 1: Applicable Modules

Module Series	Model	Description
BG95	BG95-M1	Cat M1 only
	BG95-M2	Cat M1/Cat NB2
	BG95-M3	Cat M1/Cat NB2/EGPRS
	BG95-M4	Cat M1/Cat NB2, 450 MHz Supported
	BG95-M5	Cat M1/Cat NB2/EGPRS, Power Class 3
	BG95-M6	Cat M1/Cat NB2, Power Class 3
	BG95-MF	Cat M1/Cat NB2, Wi-Fi Positioning
BG77	BG77	Cat M1/Cat NB2
BG600L	BG600L-M3	Cat M1/Cat NB2/EGPRS

2 Create IoT Hub and Establish Connection with Azure Cloud

2.1. SAS Token-Based Authentication

The Shared Access Signatures (SAS) is a claims-based authorization mechanism using simple tokens. Keys are never passed on the network by using SAS. When you create an IoT device, you can choose a symmetric key, which can easily set up a connection to the Azure Cloud.

2.1.1. Create IoT Hub on Azure Cloud

To create an IoT Hub:

Step 1: Go to https://portal.azure.com and sign in to the Azure portal.

Step 2: From the Azure homepage, select "**IoT Hub**", and then select the "**Add**" button to add a new IoT Hub, e.g., "***quectelTest***".

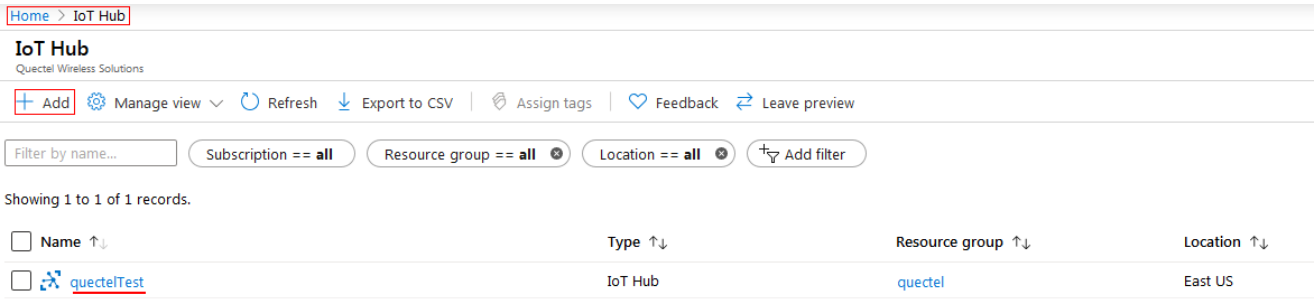


Figure 1: Add a New IoT Hub

Step 3: Click on the newly created IoT Hub e.g., "***quectelTest***" to open the new window.

Step 4: Select "**IoT device**" in the navigation bar, and then click the "**New**" button.

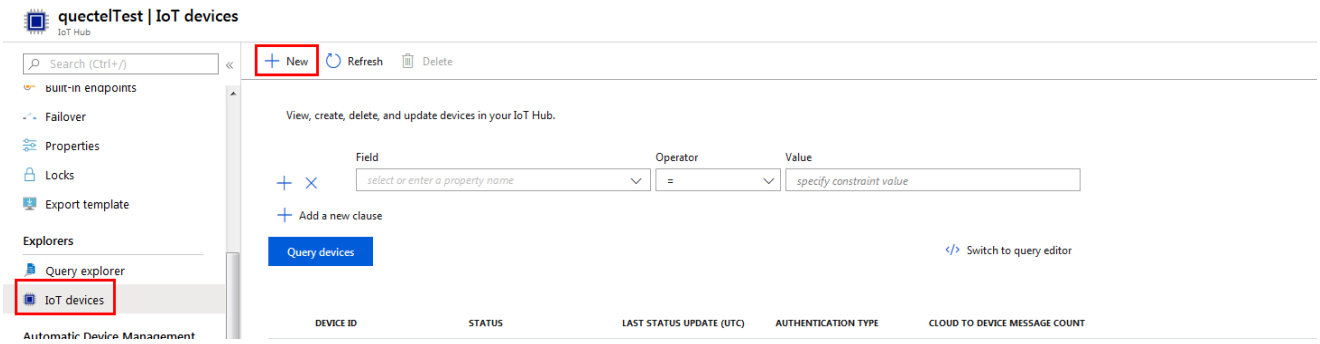


Figure 2: Select "New" Button to Create an IoT Device

Step 5: When you click the "**New**" button, the "**Create a device**" window appears. Set the necessary parameters and click the "**Save**" button to create the IoT device.

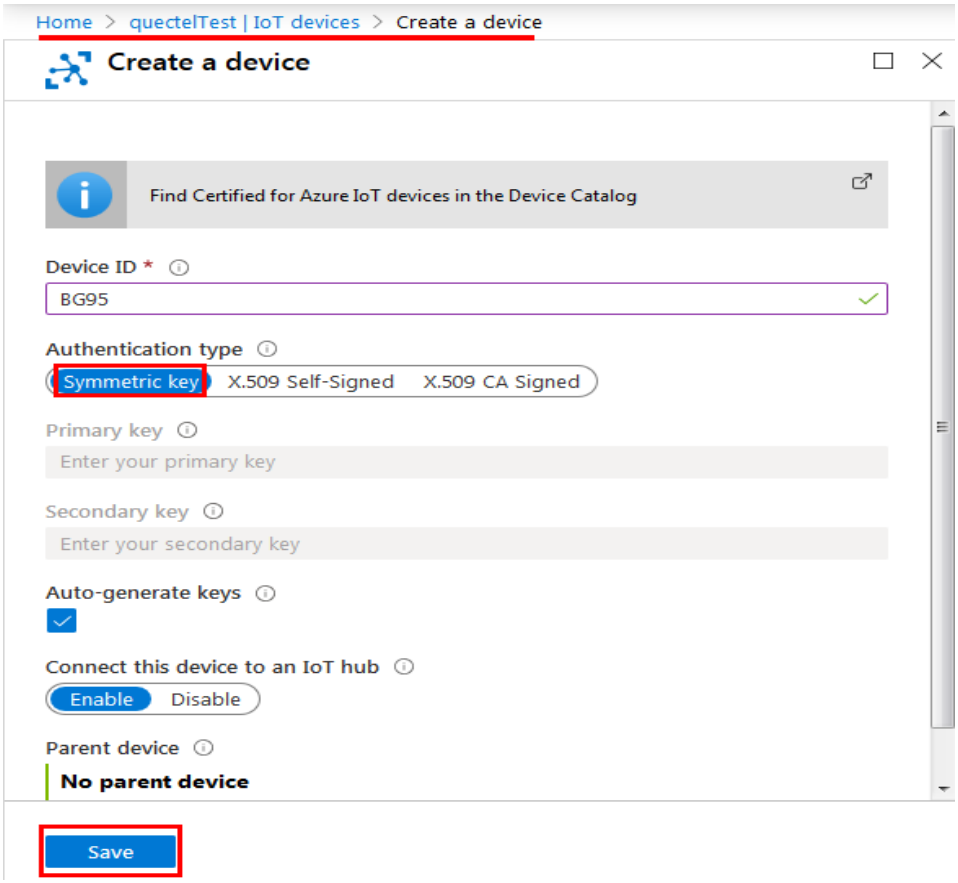


Figure 3: Create an IoT Device

Step 6: Click on the newly created device to get the "**Primary Key**".

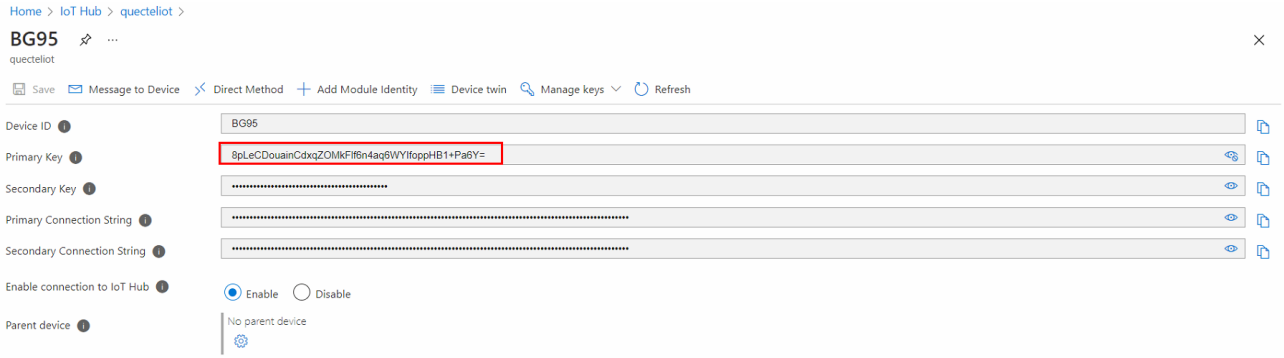


Figure 4: Get the "Primary Connection String"

The Azure IoT device SDK authenticates the connection to Azure IoT Cloud according to the "**Primary Connection String**". For detailed steps, see the official Microsoft documentation at: https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-create-through-portal.

QuecOpen® SDK contains an example on how to use SAS authentication to connect the module to the Azure IoT Cloud. You need to modify below macros in *quectel\example\azure_iot\inc\iot_sample_config.h*:

```
#define    USE_DEVICE_CERTIFICATE    0
#define    USE_DPS                   0
#define    DEVICE_SYMMETRIC_KEY      "8pLeCDouainCdxqZOMkFlf6n4aq6WYlfoppHB1+Pa6Y="
#define    IOT_HUB_HOST_NAME         "quectelTest.azure-devices.net"
#define    DEVICE_ID                 "BG95"
```

The next step is to build a QuecOpen® application and run it in the module. For detailed information, see ***Chapter 3***.

2.2. Device Provisioning Service (DPS)

The IoT Hub Device Provisioning Service (DPS) is a helper service for IoT Hub that enables zero-touch, just-in-time provisioning to the right IoT hub without requiring human intervention. DPS enables the provisioning of millions of devices in a secure and scalable manner. DPS workflow is shown in the figure below. The first step is manual, and all of the following steps are automated.

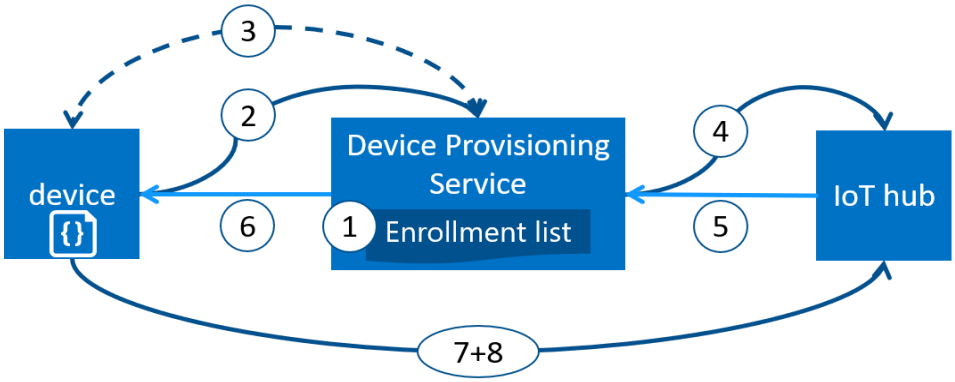


Figure 5: DPS Workflow

1. Device manufacturer adds the device registration information to the enrollment list in the Azure portal.
2. Device contacts the DPS endpoint set at the factory. The device passes the identifying information to DPS to prove its identity.
3. DPS validates the identity of the device by validating the registration ID and key against the enrollment list entry using either a nonce challenge (Trusted Platform Module) or standard X.509 verification (X.509).
4. DPS registers the device with an IoT hub and populates the device's desired twin state.
5. The IoT hub returns device ID information to DPS.
6. DPS returns the IoT hub connection information to the device. The device can now start sending data directly to the IoT hub.
7. The device connects to the IoT hub.
8. The device gets the desired state from its device twin in the IoT hub.

2.2.1. Set up the IoT Hub Device Provisioning Service with the Azure Portal

To create a new IoT Hub Device Provisioning Service:

Step 1: Go to https://portal.azure.com and sign in to the Azure portal.

Step 2: From the Azure homepage, select the "**+ Create a resource**" button.

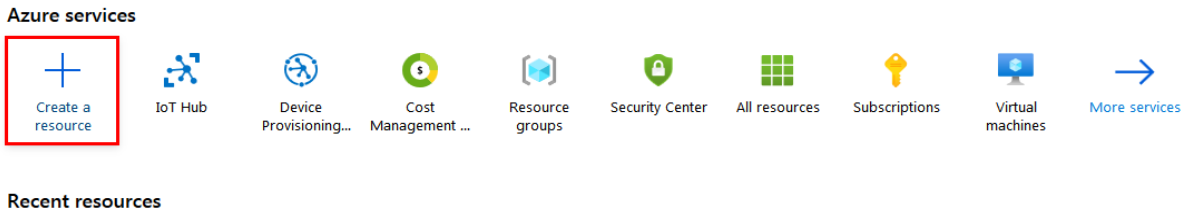


Figure 6: Click "+ Create a resource"

Step 3: Click "**Create**" on the "**IoT Hub Device Provisioning Service**" page.

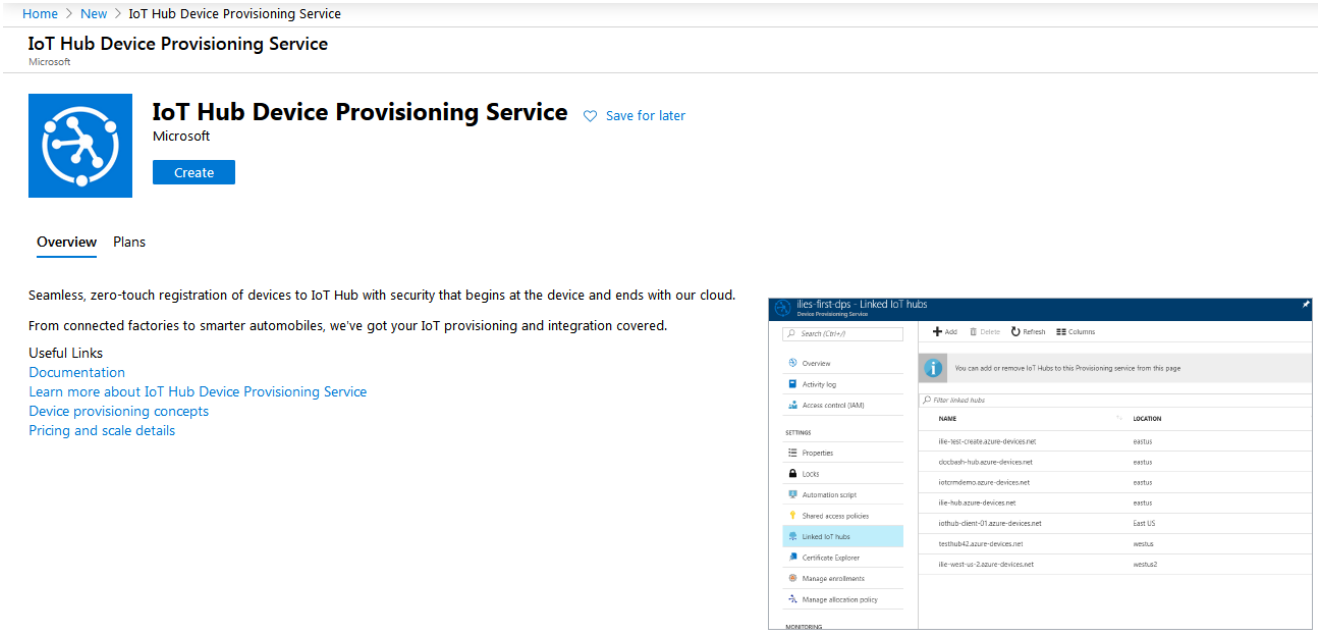


Figure 7: Click "Create" to Create an IoT Hub Device Provisioning Service

Step 4: Fill in the "**Name**", "**Subscription**", "**Resource group**" and "**Location**" fields; then click the "**Create**" button.

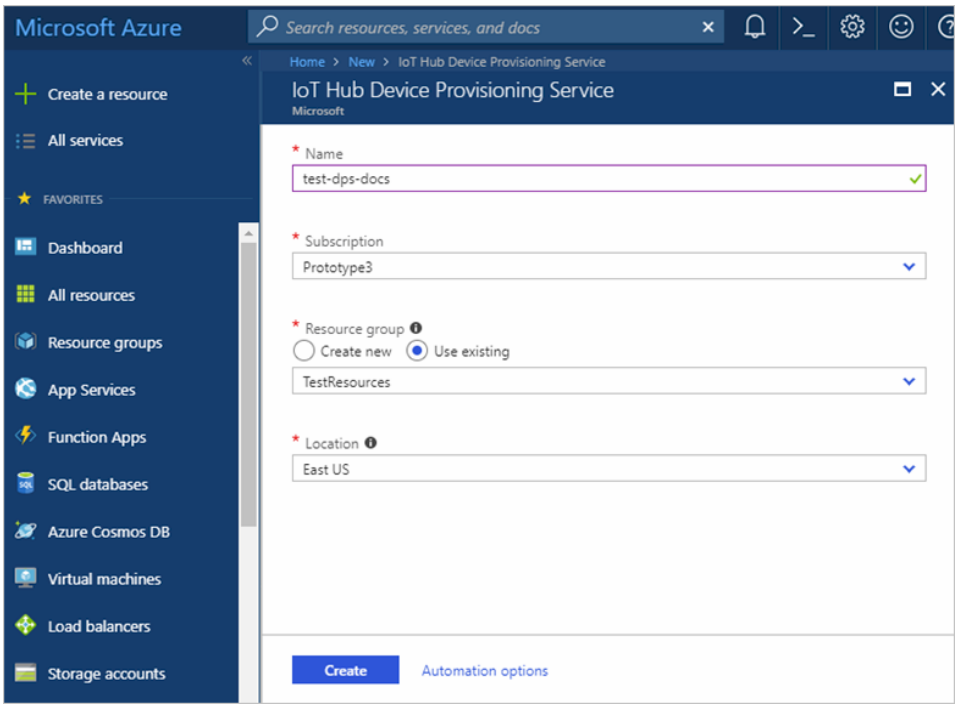


Figure 8: Create an IoT Hub Device Provisioning Service

Step 5: Click the "**Notifications**" button to monitor the creation of the resource instance. Once the service is successfully deployed, click "**Pin to dashboard**", and then click "**Go to resource**".

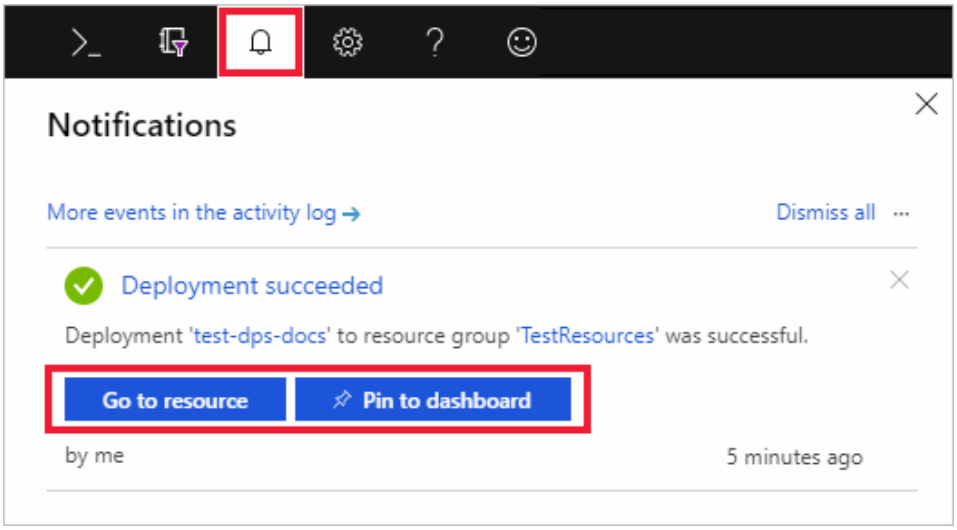


Figure 9: Click "Pin to dashboard" and "Go to resource"

Step 6: To link the IoT Hub and the Device Provisioning Service, select "**Linked IoT hubs**" in the navigation bar, then click the "**+Add**" button and the "**Add link to IoT hub**" window will appear.

Step 7: Fill in the "**Subscription**", "**IoT hub**" and "**Access Policy**" fields, and then click "**Save**" as shown in the figure below.

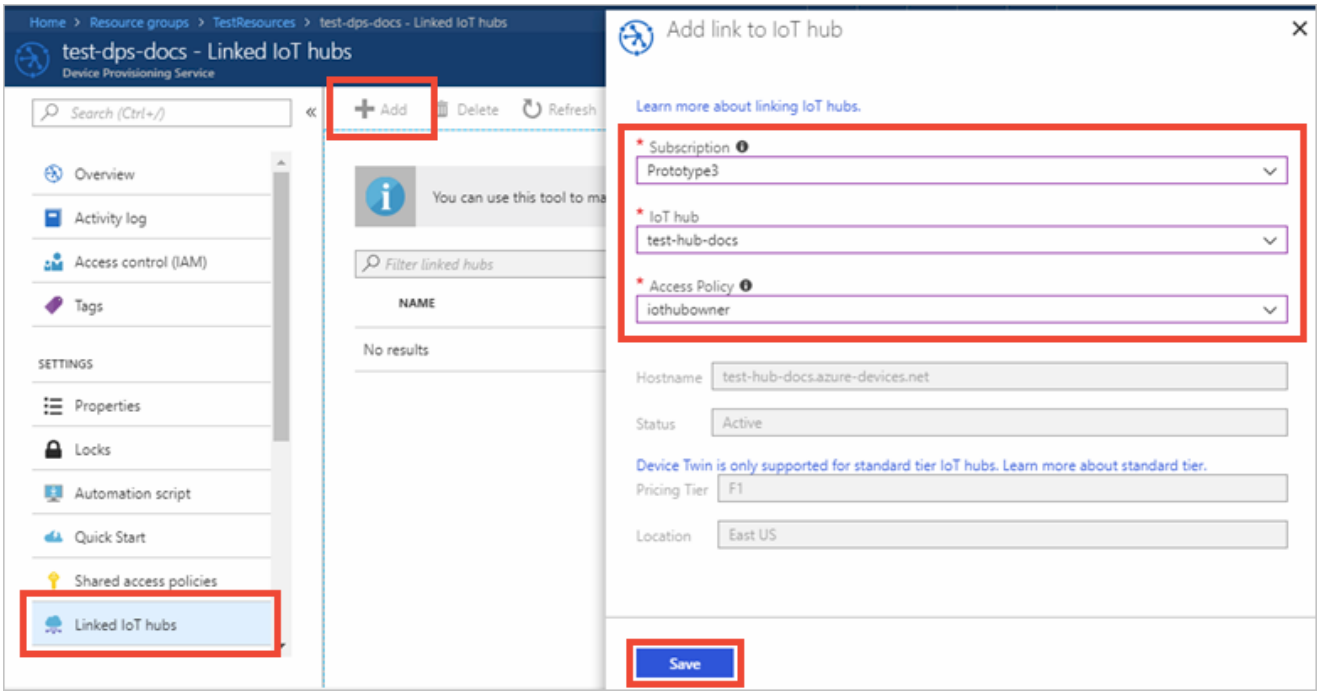


Figure 10: Add Link to IoT Hub

2.2.2. Provision a symmetry key device

To create a device using the symmetric key mechanism for authentication :

Step 1: Go to https://portal.azure.com and sign in to the Azure portal.
Step 2: On the left-hand menu or on the portal page, select All resources.
Step 3: Select your Device Provisioning Service.
Step 4: Select **"Manage enrollments"** in the navigation bar, and then select **"Add individual enrollment"** at the top of the page.

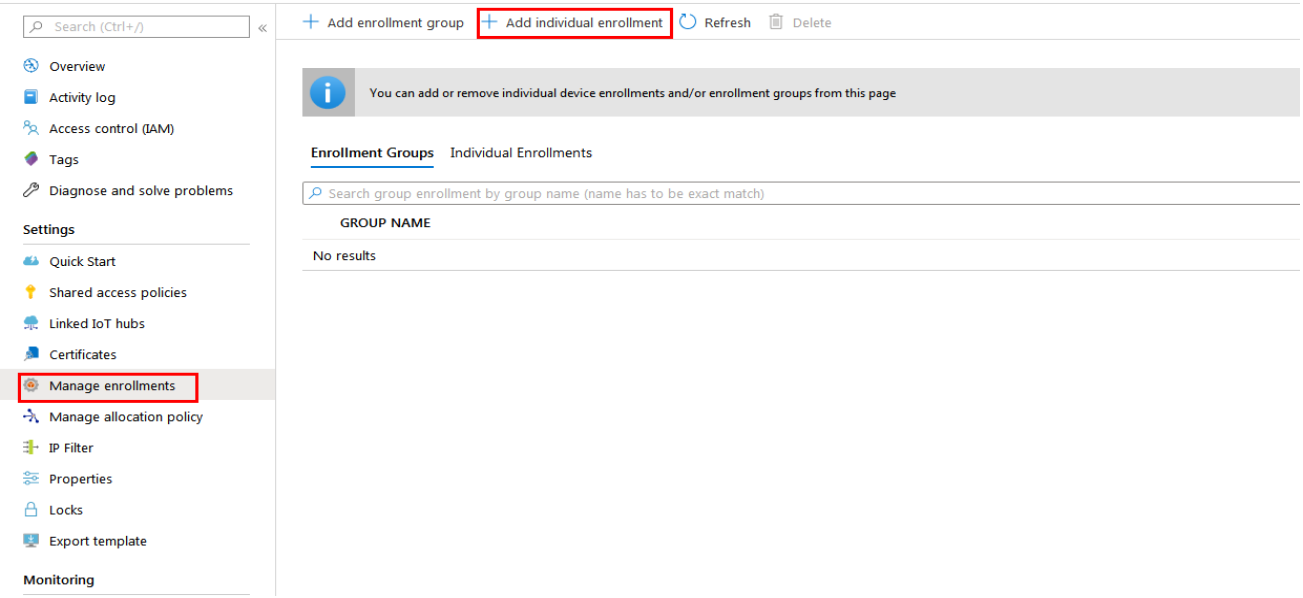


Figure 11: Add Individual Enrollment

Step 5: When you select **"Add individual enrollment"** the **"Add Enrollment"** window appears. Fill in the **"Mechanism", "Auto-generate keys"**, **"Registration ID"**, and **"IoT Hub Device ID"** fields, and then click the **"Save"** button.

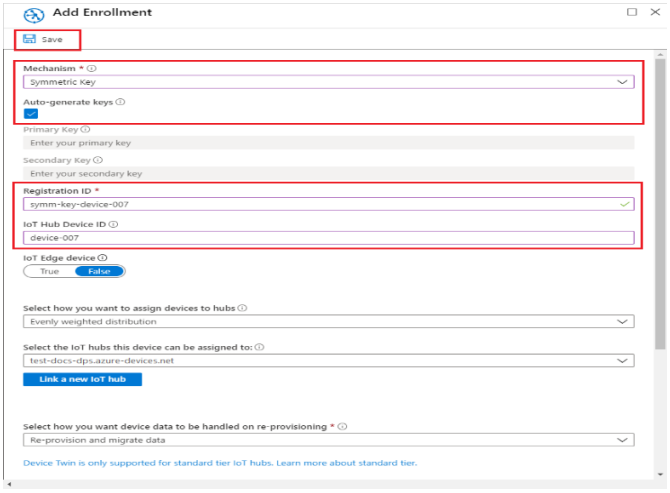


Figure 12: Add Enrollment

Step 6: Once the enrollment is saved, the **Primary Key** and **Secondary Key** are generated and added to the enrollment entry. The symmetric key device enrollment appears as "**symm-key-device007"** under the "***Registration ID***" column in the "***Individual Enrollments***" tab. Open the enrollment and copy the value of the generated "**Primary Key"**.

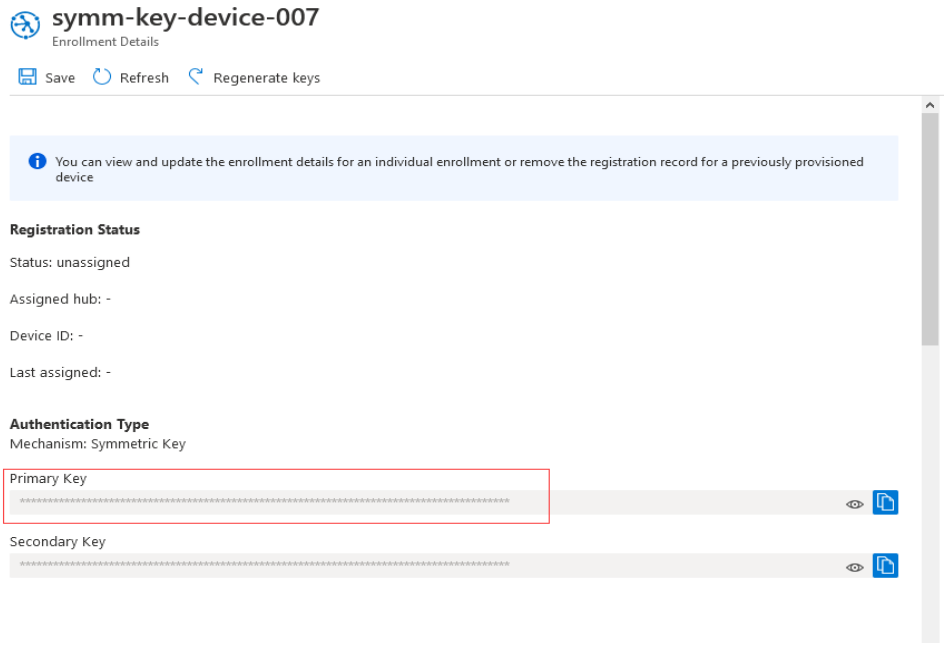


Figure 13: Get the Primary Key

Step 7: Go back to the Azure homepage, select "**Overview"** tab for the Device Provisioning Service and get the "**ID Scope**" value as shown below.

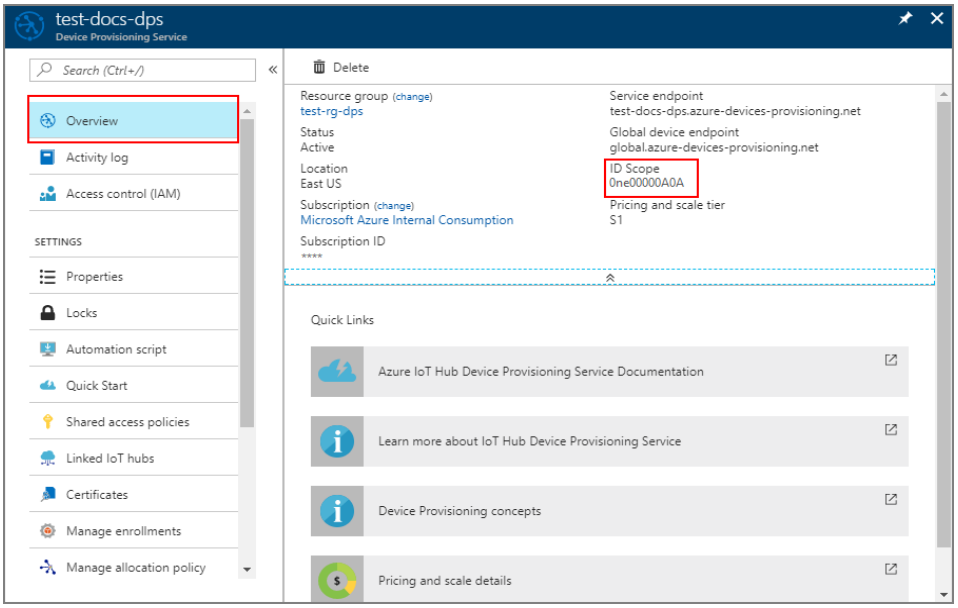


Figure 14: Get the ID Scope

For detailed information about provision a simulated symmetric key device in the Azure portal, see: https://docs.microsoft.com/en-us/azure/iot-dps/quick-create-simulated-device-symm-key?pivots=programming-language-ansi-c.

When using DPS in QuecOpen®, you need to modify below macros in *quectel\example\azure_iot\inc\iot_sample_config.h*:

```
#define    USE_DEVICE_CERTIFICATE    0
#define    USE_DPS                   1
#define    DEVICE_SYMMETRIC_KEY
"U+qM4ZB+UvcY8M9Nn8cLtjjLwgZspaRVi+tg9amQleDj1BlxCiNjiTnII/Zk8fh0qr9iHTatg9GGvRwnY1o3xA=="
#define    ID_SCOPE                  "0ne00000A0A"
#define    REGISTRATION_ID           "symm-key-device-007"
```

The next step is to build the QuecOpen® Application and run it in the module. For more information see ***Chapter 3***.

2.2.3. Provision a X.509 certificate device

To create a device using the X.509 certificate mechanism for authentication :

Step 1: Generate the device test certificate, please see: https://github.com/Azure/azure-iot-sdk-c/blob/master/tools/CACertificates/CACertificateOverview.md. The device test certificate is named *new-device.cert.pem,* the private key is named *new-device.key.pem*. Place these files in the */datatx* directory.

Step 2: Go to https://portal.azure.com and sign in to the Azure portal.

Step 3: On the left-hand menu or on the portal page, select All resources.

Step 4: Select your Device Provisioning Service.

Step 5: Select "**Manage enrollments**" in the navigation bar, and then select "**Add individual enrollment**" at the top of the page.

Step 6: When you select "**Add individual enrollment**" the "**Add Enrollment**" window appears. Fill in the "**Mechanism**", "**Primary certificate .pem or .cer file**", and "**IoT Hub Device ID**" fields, and then click the "**Save**" button.

For detailed information about provision a X.509 certificate simulated device in the Azure portal, see: https://docs.microsoft.com/en-us/azure/iot-dps/quick-create-simulated-device-x509?tabs=windows&pivots=programming-language-ansi-c.

When using DPS in QuecOpen®, you need to modify below macros in *quectel\example\azure_iot\inc\iot_sample_config.h*:

```
#define    USE_DEVICE_CERTIFICATE    1
#define    USE_DPS                   1
#define    ID_SCOPE                  "0ne00000A0A"
```

The next step is to build the QuecOpen® Application and run it in the module. For more information see ***Chapter 3***.

2.3. IoT Central

IoT Central is an IoT application platform as a service(aPaaS)that reduces the burden and cost of developing, managing, and maintaining enterprise-grade IoT solutions. If you choose to build with IoT Central, you'll have the opportunity to focus time, money, and energy on transforming your business with IoT data, rather than just maintaining and updating a complex and continually evolving IoT infrastructure.

2.3.1. Create an IoT Central application

To create an IoT Central application:

Step 1: Go to https://apps.azureiotcentral.com and sign in.
Step 2: On the left-hand menu select **Build**.
Step 3: Select **Create app**.
Step 4: Add **Application name** and **URL**.
Step 5: Select **Create**.

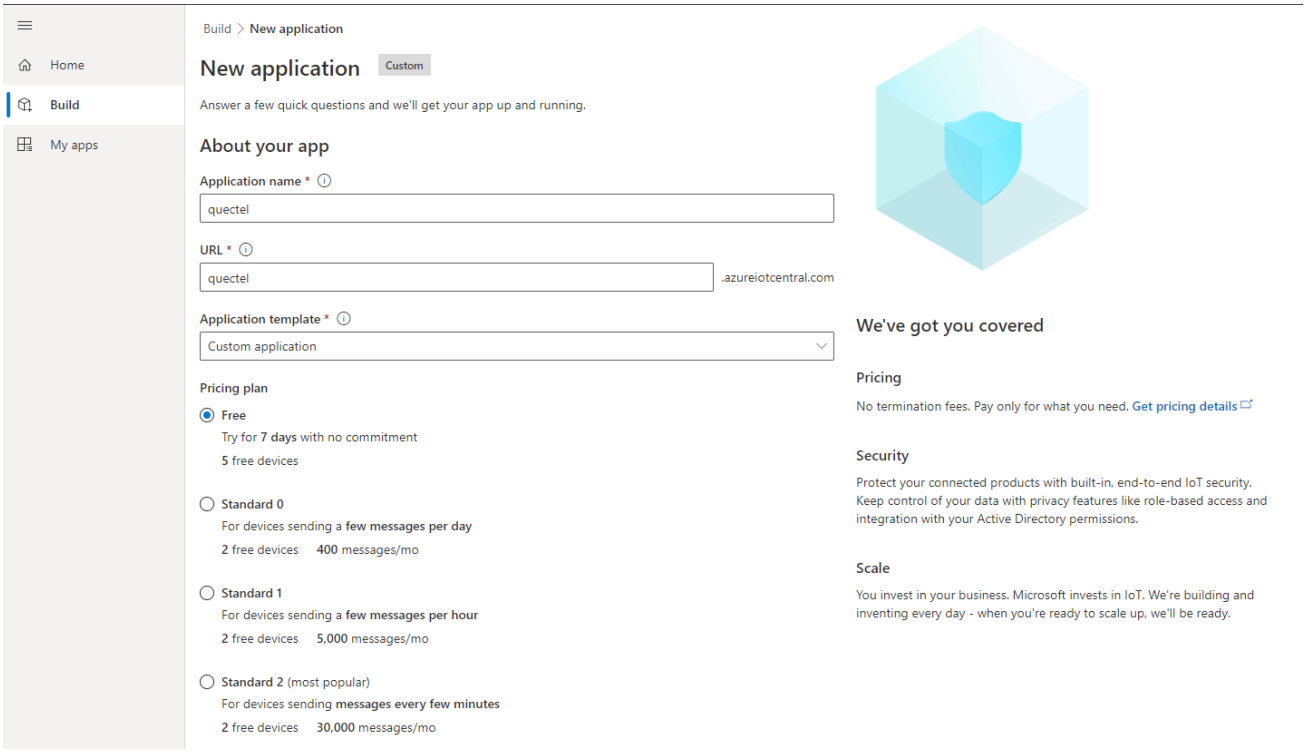


Figure 15: Create an IoT Central application

2.3.2. Create and connect a client application to Azure IoT Central application

The application simulates the behavior of a tracker device. When the applications connect to IoT Central,

it sends the model ID of the tracker device model, IoT Central uses the model ID to retrieve the device model and create a template.

To create and connect a client application:

Step 1: In the application, navigate to **Permissions > Device connection groups**, make a note of the **ID scope** value.

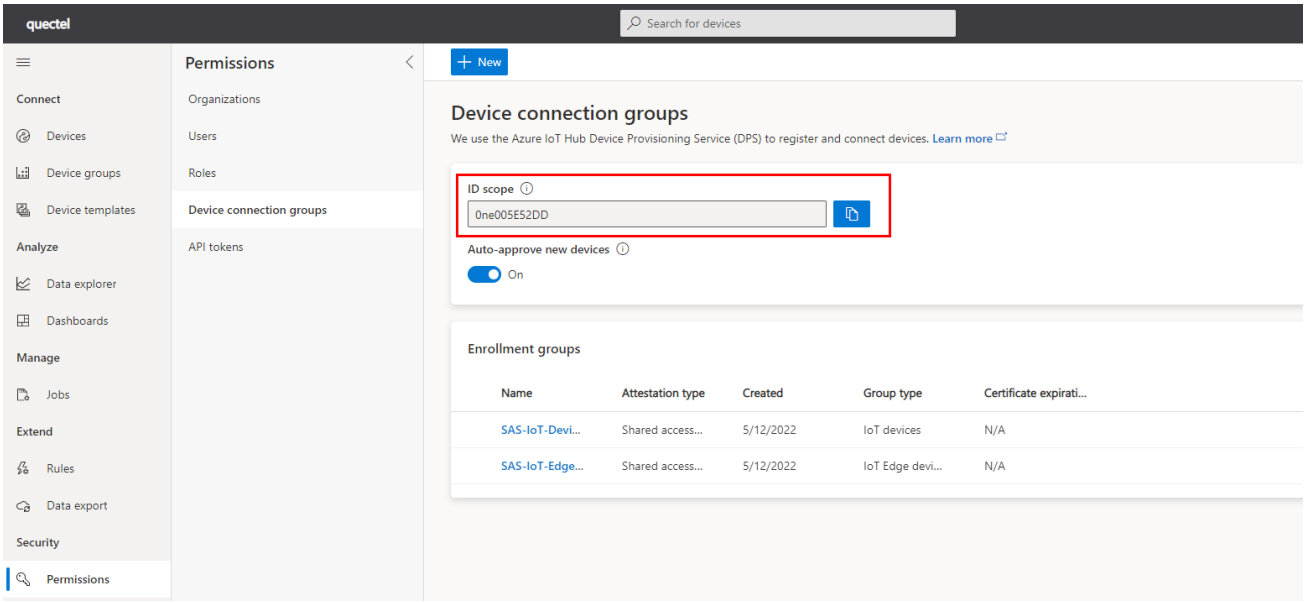


Figure 16: Get ID scope value

Step 2: navigate to **Permissions > Device connection groups > SAS-IoT-devices**. Make a note of the shares access signature **Primary key** value.

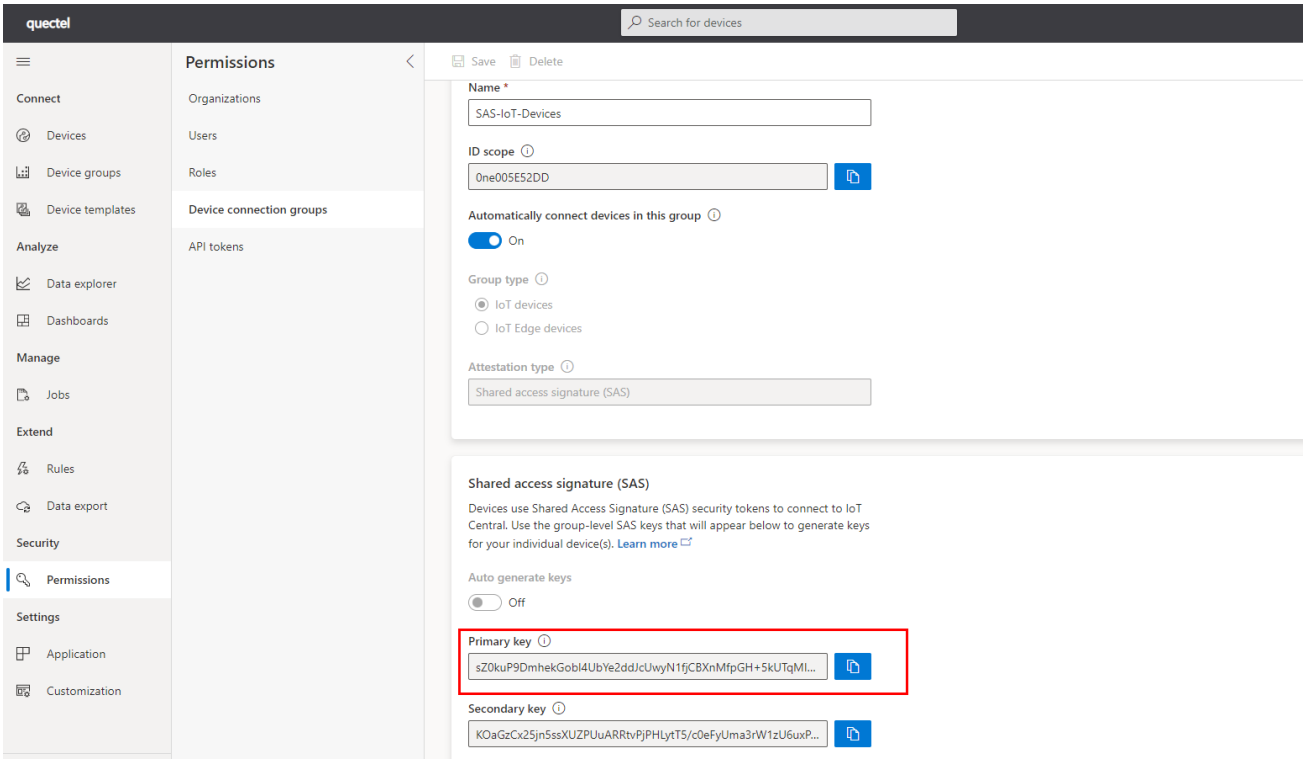


Figure 17: Get primary value

Step 3: Use the Cloud Shell(get from https://docs.microsoft.com/en-us/cli/azure/install-azure-cli) to generate a device key from the group primary key retrieved.

For detailed information about create and connect a client application to Azure IoT Central application, see: https://docs.microsoft.com/en-us/azure/iot-central/core/tutorial-connect-device?pivots=programming-language-ansi-c.

When using tracker example in QuecOpen®, you need to modify below macros in *quectel\example\azure_iot\inc\iot_sample_config.h*:

```
#define    USE_DEVICE_CERTIFICATE    0
#define    USE_DPS                   1
#define    ID_SCOPE                  "0ne005E52DD"
#define    DEVICE_SYMMETRIC_KEY      "7Ic2z0VabexsicoJz8iA0RFt9NKTAHHKgnU3mHBXuqw="
#define    REGISTRATION_ID           "sample-device-01"   // It's the –device-id used in Cloud Shell
#define    MODEL_ID                  "dtmi:quectel:tracker;1 "
```

The next step is to build the QuecOpen® Application and run it in the module. For more information see ***Chapter 3***.

Step 4: Select the device from the device list to view telemetry as the device send messages to the cloud in the **Overview** Tab.

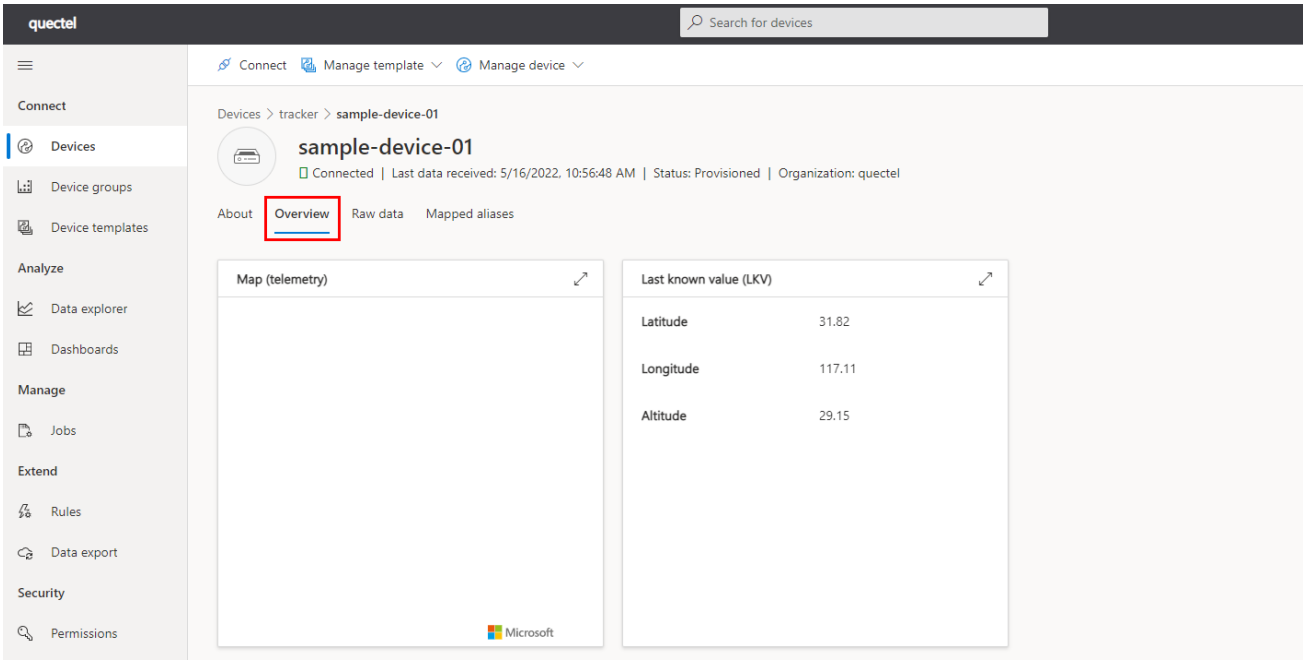


Figure 18: Telemetry information

Step 5: Select **About** tab to view device information.

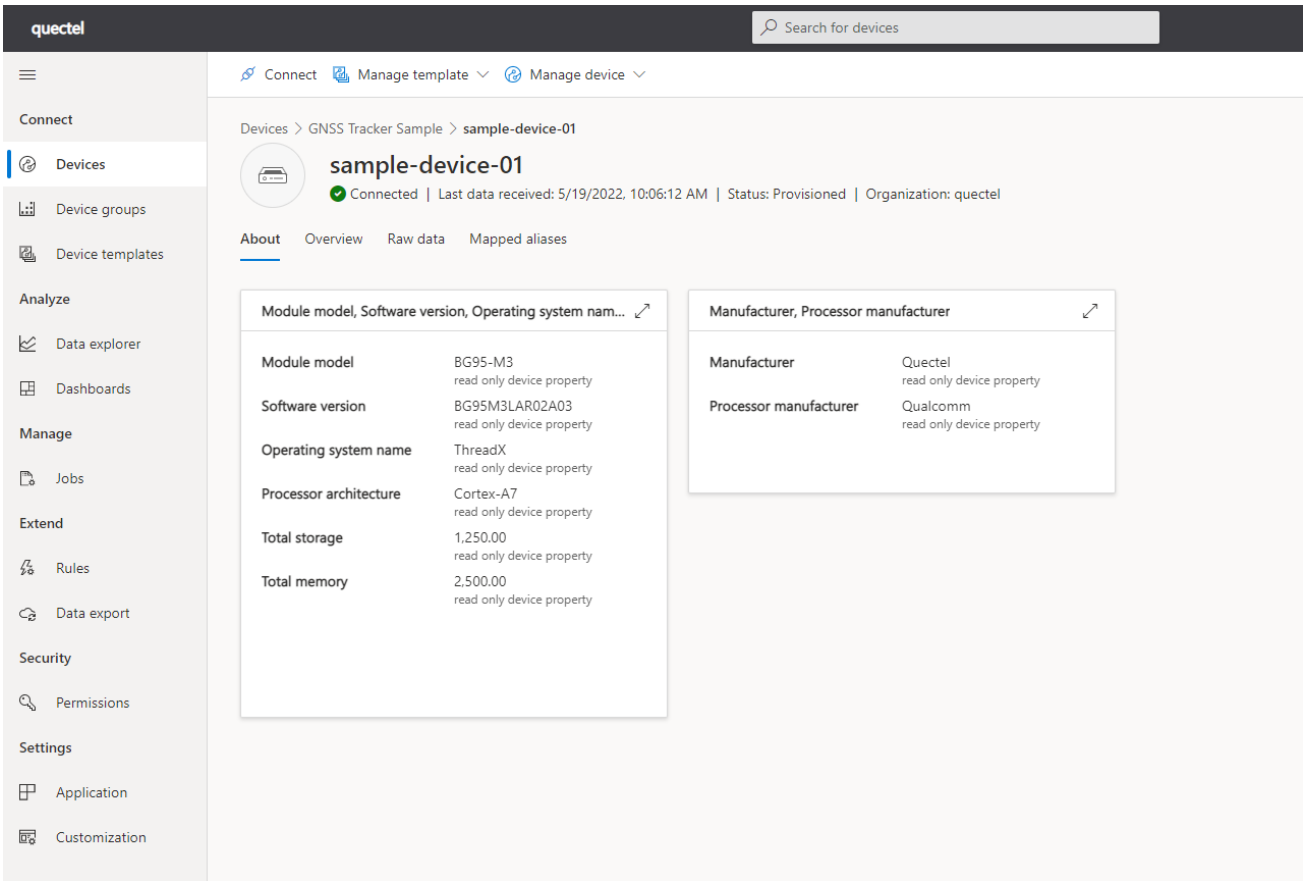


Figure 19: Device information

3 Build and Run Azure IoT Hub Applications

3.1. QuecOpen® SDK Package

Folder and file structure of QuecOpen SDK (use *Quectel_BG95_QuecOpen_SDK_Package* for illustration in this document) is presented below.

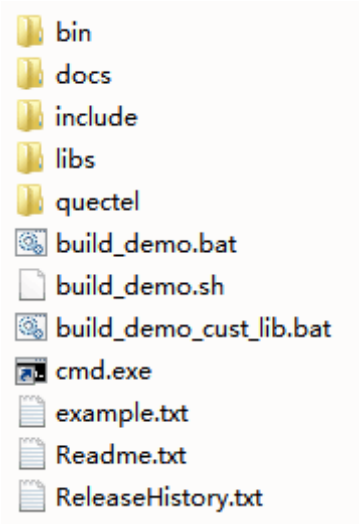


Figure 20: Folder and File Structure of BG95 QuecOpen® SDK Package

Table 2: Description of BG95 QuecOpen® SDK Package Directories and Files

Directories	Description/Function
bin	Applications are created in this folder after successful compilation
docs	QuecOpen® related documents
include	Header files needed for compilation provided by Quectel
libs	Required libraries should be copied here
quectel	Quectel example source code files

Files	Description/Function
build_demo.bat	Batch script for building Quectel examples
build_demo.sh	Shell script for building Quectel examples
build_demo_cust_lib.bat	Batch script including examples for building private libraries
example.txt	Configuration file for compilation options of examples
Readme.txt	Information for QuecOpen® SDK package
ReleaseHistory.txt	SDK package release history

3.2. Build Azure IoT Hub Applications

Before application building, see *Quectel_BG95&BG77&BG600L_Series_QuecOpen_Application_Note* for details about the preparations.

- **In *build_demo.bat,* you can find the lines about Azure SDK building, as shown below**:

```
set AZURE_SDK_INC_PATH=include\azure_api
set AZURE_SDK_PORT_INC_PATH=include\porting_laye

set AZURE_SDK_LIBNAME=azure_sdk.lib
set AZURE_SDK_PORT_LIBNAME=azure_sdk_port.lib

set DAM_INCPATHS=%DAM_INCPATHS% -I    %AZURE_SDK_PORT_INC_PATH%\inc -
I %AZURE_SDK_INC_PATH%\c-utility\inc -I %AZURE_SDK_INC_PATH%\c-utility\pal\generic -
I %AZURE_SDK_INC_PATH%\serializer\inc -I %AZURE_SDK_INC_PATH%\iothub_client\inc -
I %AZURE_SDK_INC_PATH%\deps\parson -I %AZURE_SDK_INC_PATH%\umqtt\inc -
I %AZURE_SDK_INC_PATH%\deps\umock-c\inc -I %AZURE_SDK_INC_PATH%\deps\azure-macro-
utils-c\inc      -I %AZURE_SDK_INC_PATH%\umqtt\inc
```

- **Run build commands for Azure IoT application compilation**

The following commands can only be executed in Windows:

The build commands for basic example:

```
build_demo.bat llvm azure_iot basic_example          //New build

build_demo.bat llvm –c                               //Clean build
```

The build commands for pnp example:

```
build_demo.bat llvm azure_iot pnp_example            //New build

build_demo.bat llvm –c                               //Clean build
```

The build commands for pnp tracker example:

```
build_demo.bat llvm azure_iot pnp_tracker_example        //New build

build_demo.bat llvm –c                                   //Clean build
```

Once the build process is completed, the application binary image (e.g., *quectel_demo_azure_iot.bin*) is created under */bin* folder.

To run the binary image, upload it into the alternate file system of the module. See ***Chapter 3.4*** in *Quectel_BG95&BG77&BG600L_Series_QuecOpen_Application_Note* for details.

3.3. Connection with Azure IoT Hub

Use Azure IoT tool in the Visual Studio Code to debug Azure IoT Hub. For more information on how to install and use the Azure IoT tool see: https://marketplace.visualstudio.com/items?itemName=vsciot-vscode.azure-iot-tools.

When the device connects Azure IoT Hub with SAS, you see the following in Visual Studio Code:

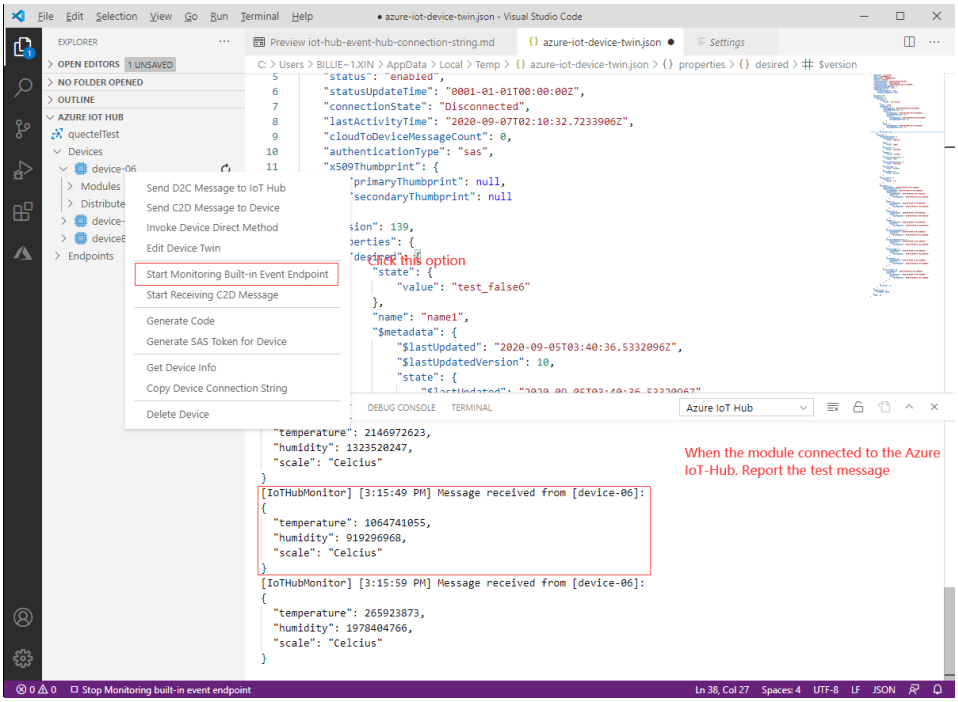


Figure 21: Azure IoT Hub with SAS Connection

The Azure IoT Hub connection with DPS is shown below:

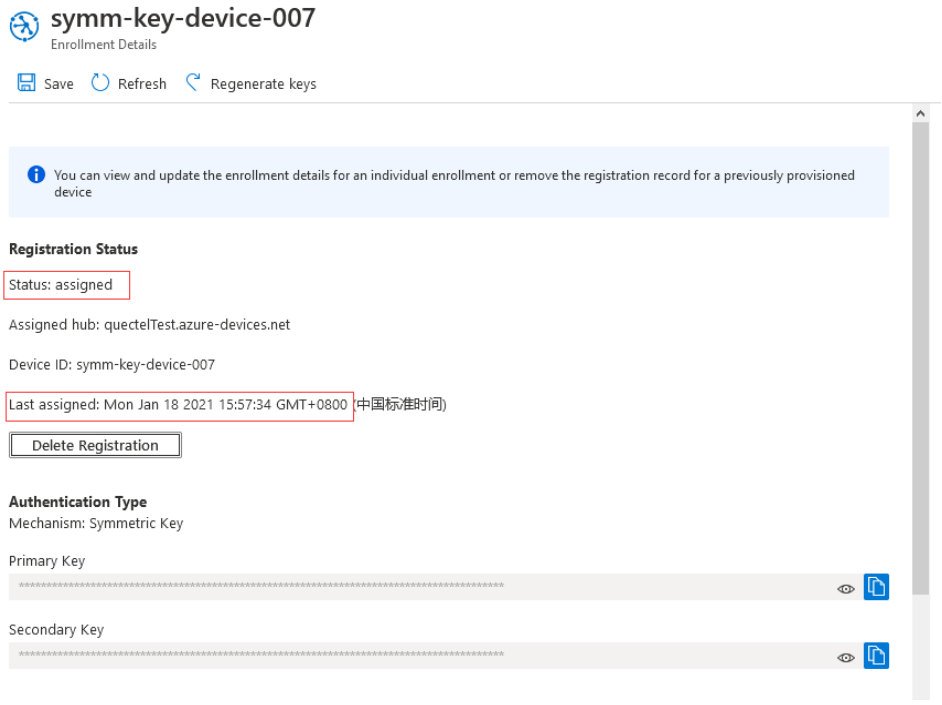


Figure 22: Azure IoT Hub with DPS Connection

3.4. Interact with Azure IoT Hub

The device can interact with Azure IoT Hub after making a successful connection.

3.4.1. Sending Messages to Device with SAS Connection

IoT Hub gives you the ability to invoke "**Direct Method**" on devices from the cloud. The "**Direct Method**" is a request-reply interaction with a device similar to an HTTP call in that it succeeds or fails immediately. The "**Direct Method**" has payload, configurable connection and method timeouts.

To send a message to the device with SAS connection:

Step 1: Go to https://portal.azure.com and sign in to the Azure portal.

Step 2: In the Azure home page, click "**IoT Hub**", select and open the IoT Hub created in ***Chapter 2.1.1***.

Step 3: Under "**Explorers**", click "**IoT devices**". In the left navigation bar of the "**IoT devices**" detail page, select the device created in ***Chapter 2.1.1***. The use of "**Direct Method**" to send method and payload to a device in the device console page is shown in the figure below.

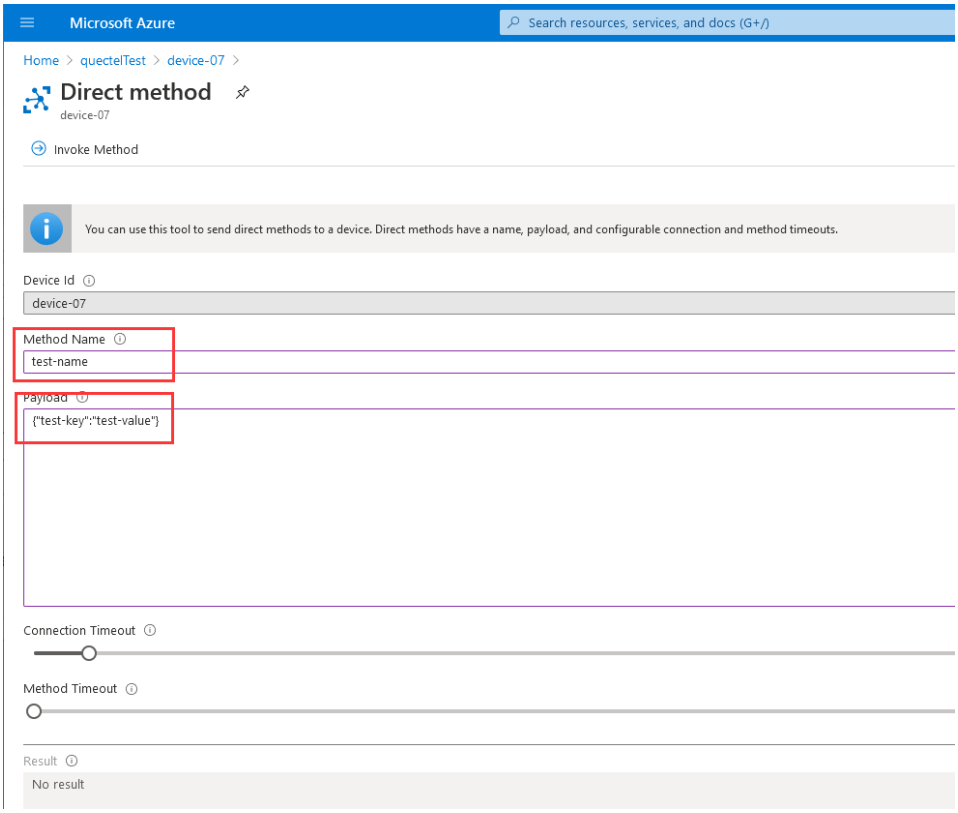


Figure 23: Send Message from the Azure IoT Cloud to a Specific Device

UART logs indicate that the device received direct methods successfully.

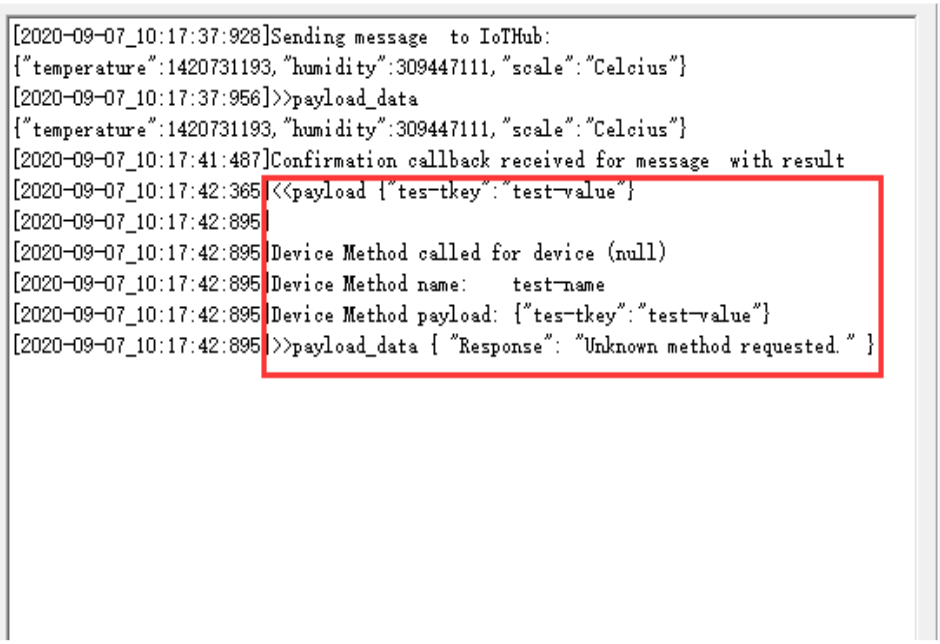


Figure 24: Device Receive Message from the Azure IoT Cloud

3.4.2. Device Report Properties to Device Twins with DPS Connection

Device twins are JSON documents that store device state information including metadata, configurations, and conditions. Azure IoT Hub maintains a device twin for each device. A device twin contains desired properties, reported properties, and tags. A desired property is set by a backend service and read by a device. A reported property is set by a device and read by a backend service. A tag is set by a backend service and is never sent to a device. You can use tags to organize your devices.

The use of reported properties by the backend service to receive state information from a device is shown in the figure below.

Figure 25: Device State Information Reported to Device Twins

3.4.3. Update Device Twin's Reported Properties

When you update the desired properties of device twins, the device application can also receive notifications of changes in the desired properties.

```
"x509Thumbprint": {
    "primaryThumbprint": null,
    "secondaryThumbprint": null
},
"version": 139,
"properties": {
    "desired": {
        "state": {
            "value": "test_false6"
        },
        "name": "name1",
        "$metadata": {
            "$lastUpdated": "2020-09-05T03:40:36.5332096Z",
            "$lastUpdatedVersion": 10,
            "state": {
                "$lastUpdated": "2020-09-05T03:40:36.5332096Z",
                "$lastUpdatedVersion": 10,
                "value": {
                    "$lastUpdated": "2020-09-05T03:40:36.5332096Z",
                    "$lastUpdatedVersion": 10
                }
            },
            "name": {
                "$lastUpdated": "2020-09-05T03:40:36.5332096Z",
                "$lastUpdatedVersion": 10
            }
        },
        "$version": 10
    },
    "reported": {
```


Figure 26: Modify Device Twin's Desired Properties

UART logs that show that the device received notifications of changes in the desired properties are presented in the figure below:


```
[2020-09-05_11:00:06:366]Current nesting 4, string 33554432}},"$iotin:sensor":
{"state":{"value":"ff"}},"$version":55}}
[2020-09-05_11:00:06:368]Current nesting 2, string {"state":
{"value":"ff"}},"$version":55}}
[2020-09-05_11:00:06:368]Current nesting 3, string {"value":"ff"}},"$version":55}}
[2020-09-05_11:00:06:368]Current nesting 4, string "ff"}},"$version":55}}
[2020-09-05_11:00:06:368]Current nesting 2, string 55}}
[2020-09-05_11:00:06:368]propertyName state
[2020-09-05_11:00:06:368]propertyToQuery state.value
[2020-09-05_11:00:06:377]value=h?@
[2020-09-05_11:00:06:377]buf="ff"
[2020-09-05_11:00:06:377]GetPayloadFromProperty "ff"
[2020-09-05_11:00:06:377]DigitalTwin Interface : Invoking property callback for
interface sensor, propertyName=state, propertyCallbackContext=
0x400ed12SENSOR_INTERFACE: Property name <state> is not associated with this
interface
[2020-09-05_11:00:06:384]DigitalTwin Interface: Invoking property callback returned
[2020-09-05_11:00:06:384]<<payload {"desired":{"state":
{"value":"test_false6"},"name":"name1","$version":10},"reported":
{"$iotin:deviceinfo":{"manufact
```


Figure 27: Device Received Notifications of Changes in the Desired Properties

4 Appendix A References

Table 3: Related Document

SN	Document Name	Description
[1]	Quectel_BG95&BG77&BG600L_Series_QuecOpen_Application_Note	Quectel BG95&BG77&BG600L Series QuecOpen® Application Note

Table 4: Terms and Abbreviations

Abbreviation	Description
DPS	Device Provisioning Service
ID	Identifier
IoT	Internet of Things
I2C	Inter-Integrated Circuit
LLVM	Low Level Virtual Machine
RAM	Random Access Memory
SDK	Software Development Kit
SAS	Statistical Analysis System
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver/Transmitter