

Renesas Synergy™ Platform

Synergy MQTT/TLS Google Cloud Connectivity Solution

Introduction

This application note describes the IoT Cloud connectivity solution in general and introduces you briefly to the IoT Cloud provider, Google Cloud. It covers the Synergy MQTT/TLS module, its features, and operational flow sequence (Initialization/Data flow). The application example provided in the package uses Google Cloud IoT Core. The detailed steps in this document show first-time Google Cloud IoT Core users how to configure the Google Cloud IoT Core platform to run this application example demonstration.

This application note enables you to effectively use the Synergy MQTT/TLS modules in your own design. Upon completion of this guide, you will be able to add the MQTT/TLS module to your own design, configure it correctly for the target application, and write code using the included application example code as a reference and efficient starting point. References to more detailed API descriptions, and other application projects that demonstrate more advanced uses of the module, are in the *Synergy Software Package (SSP) User's Manual* (see Next Steps section) and serve as valuable resources in creating more complex designs.

Currently, the Synergy MQTT/TLS Connectivity solution is implemented and tested using Google Cloud IoT Core on AE-CLOUD1 and AE-CLOUD2 kit. Support for other Synergy kits and IoT Cloud providers will be provided in upcoming releases.

Required Resources

To build and run the MQTT/TLS application example, you need:

Development tools and software

- e² studio ISDE v7.3.0 or later or IAR Embedded Workbench® for Renesas Synergy™ v8.23.3 or later, available at www.renesas.com/synergy/e2studio
- Synergy Software Package (SSP) 1.6.2 or later (www.renesas.com/synergy/ssp)
- Synergy Standalone Configurator (SSC) 7.3.0 or later (www.renesas.com/synergy/ssc)
- SEGGER J-link® USB driver (www.renesas.com/synergy/jlinksynergy)

Hardware

- Renesas Synergy™ **AE-CLOUD1 kit** (www.renesas.com/synergy/ae-cloud1), which includes a Wi-Fi board and, **AE-CLOUD2 kit** (www.renesas.com/synergy/ae-cloud2), which includes a Pillar board, Wi-Fi board, BG96 Cellular shield.
Note: A CAT-M1, NB-IoT, or EGPRS SIM card should be procured separately for cellular functionality.
- Renesas Synergy™ Application Example kit PMOD based Wi-Fi Module (www.renesas.com/synergy/kits/ae-wifi1).
- PC running Windows® 7 or 10; the Tera Term console, or similar application, and an installed web browser (Google Chrome, Internet Explorer, Microsoft Edge, Mozilla Firefox, or Safari).
- Micro USB cables
- Ethernet cable

Prerequisites and Intended Audience

This application note assumes that you have some experience with the Renesas e² studio ISDE and Synergy Software Package (SSP). Before you perform the procedures in this application note, follow the procedure in the *SSP User Manual* to build and run the Blinky project. Doing so enables you to become familiar with e² studio and SSP and validates that the debug connection to your board functions properly. In addition, this application note assumes that you have some knowledge of MQTT/TLS and its communication protocols.

The intended audience is users who want to develop applications with MQTT/TLS modules using Renesas Synergy™ S5 or S7 MCU Series.

Contents

1. Introduction to Cloud Connectivity4

1.1 Overview..... 4

1.2 Major Components 4

1.3 Cloud Provider Overview..... 5

1.3.1 Google Cloud IoT Core Overview 5

1.4 MQTT Protocol Overview 7

1.5 TLS Protocol Overview..... 8

1.5.1 Device Certificates and Keys 8

1.5.2 Device Security Recommendations 9

2. Synergy MQTT/TLS Cloud Solution.....9

2.1 MQTT Client Overview 9

2.2 Design Considerations 10

2.2.1 Supported Features..... 10

2.2.2 Operational Flow Sequence 10

2.3 TLS Session Overview 11

2.3.1 Design Considerations 11

2.3.2 Supported Features..... 11

2.3.3 Operational Flow Sequence 12

3. MQTT/TLS Application Example.....14

3.1 Application Overview 14

3.2 Software Architecture Overview 15

3.2.1 Console Thread 16

3.2.2 MQTT Thread..... 16

3.2.3 MQTT Rx Thread..... 16

3.3 IoT Cloud Configuration (GCloud)..... 16

3.3.1 Creating a Device on Google Cloud IoT Core..... 17

3.3.2 Generate Device Key and Certificate..... 20

3.3.3 Add Public Key to the Google Cloud IoT Core..... 21

4. Running the MQTT/TLS Application22

4.1 Importing, Building, and Loading the Project 22

4.2 Manually Adding the Board Support Package for AE-CLOUD1/AE-CLOUD2 Kit 22

4.3 Powering up the Board..... 22

4.4 Connect to Google IoT Cloud..... 23

4.4.1 Configuration Wizard Menu..... 24

4.4.2 Dump the Previous Configuration 33

4.4.3 Demo Start/Stop Command 34

4.5 Verifying the Demo..... 35

4.5.1 Running the Synergy Cloud Connectivity Demonstration 35

4.5.2	Monitoring MQTT Messages on Google Cloud Platform	35
4.5.3	Publishing the MQTT Message from Google Cloud Platform	37
4.5.4	Stopping the Synergy Cloud Connectivity Demonstration	38
4.6	Customizing the Demo Delays	38
5.	Next Steps.....	38
6.	MQTT/TLS Reference	39
7.	Known Issues and Limitations	39
	Revision History	41

1. Introduction to Cloud Connectivity

1.1 Overview

Internet of Things (IoT) is a sprawling set of technologies described as connecting everyday objects, like sensors or smart-phones, to the World Wide Web. IoT devices are intelligently linked together to enable new forms of communication between things and people, and among things.

These devices, or things, connect to the network. Using sensors, they provide the information they gather from the environment or allow other systems to reach out and act on the world through actuators. In the process, IoT devices generate massive amounts of data, and cloud computing provides a pathway, enabling data to travel to its destination.

1.2 Major Components

The IoT Cloud Connectivity Solution includes the following major components:

- Devices or Sensors
- Gateway
- IoT Cloud services
- End user application/system

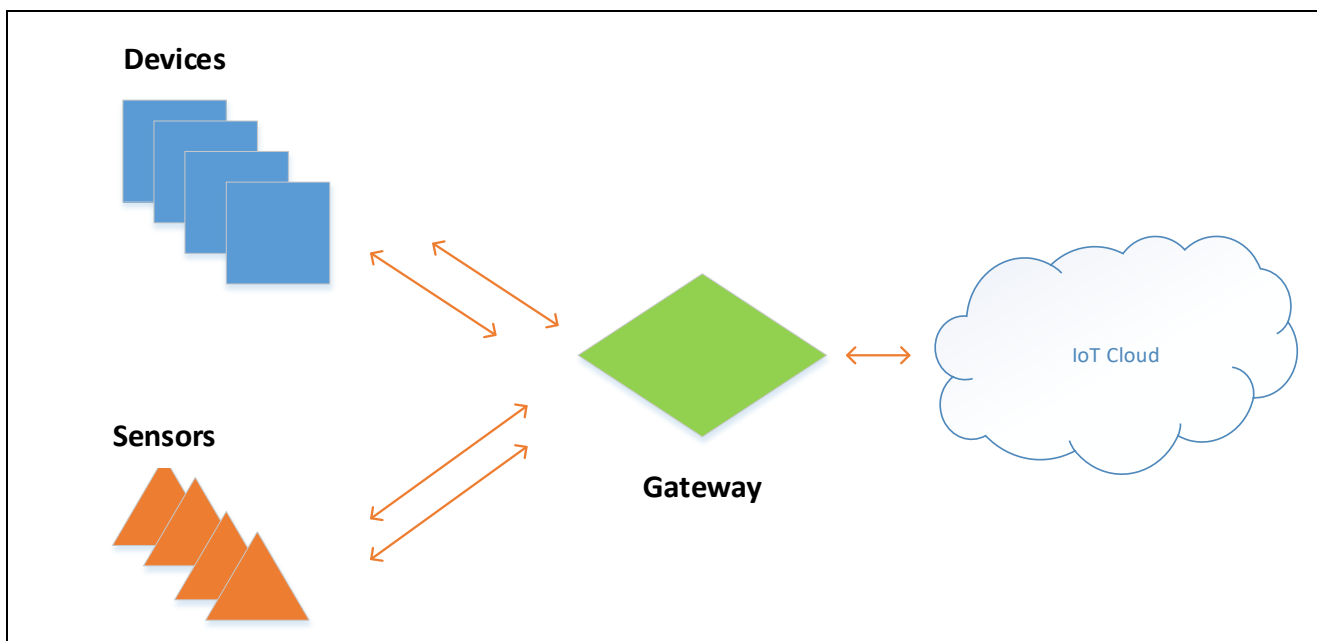


Figure 1. IoT Cloud Connectivity Architecture

Devices or Sensors

A device includes hardware and software that interact directly with the world. Devices connect to a network to communicate with each other, or to centralized applications. Devices may connect to the Internet either directly or indirectly.

Gateway

A gateway enables devices that are not directly connected to the Internet to reach cloud services. The data from each device is sent to the Cloud Platform, where it is processed and combined with data from other devices, and potentially with other business-transactional data. Most of the common communication gateways support one or more communication technologies such as Wi-Fi, Ethernet, or cellular.

IoT Cloud

Many IoT devices produce lots of data. You need an efficient, scalable, affordable way to manage those devices, handle all that information, and make it work for you. When it comes to storing, processing, and analyzing data, especially big data, it is hard to surpass the Cloud.

1.3 Cloud Provider Overview

1.3.1 Google Cloud IoT Core Overview

Google Cloud IoT Core is a fully managed service that allows users to easily and securely connect, manage, and ingest data from millions of globally dispersed devices. Google Cloud IoT Core, in combination with other services on Cloud IoT platform, provides a complete solution for collecting, processing, analyzing, and visualizing IoT data in real time to support improved operational efficiency.

The following diagram summarizes the service components and the flow of data.

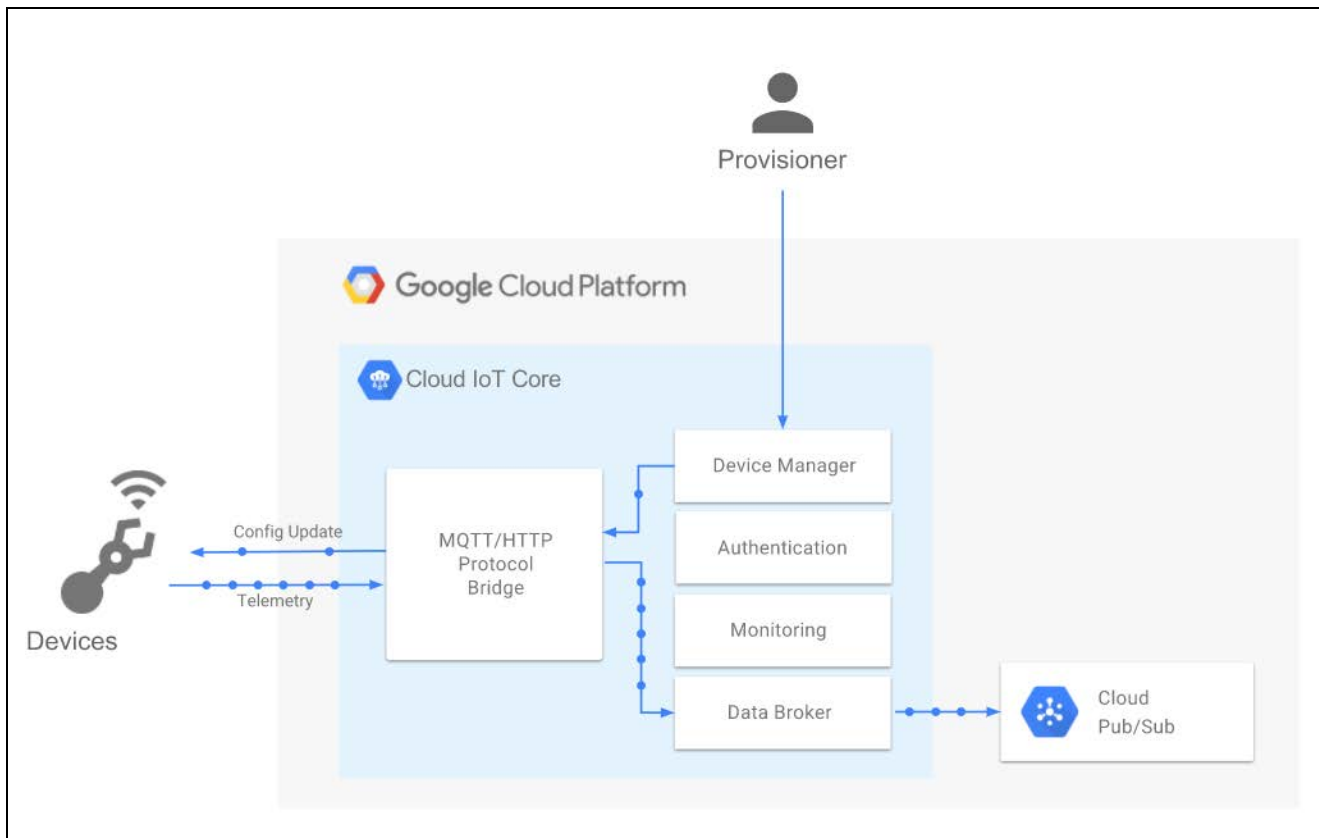


Figure 2. Google IoT Cloud Solution

The main components of Google Cloud IoT Core are the device manager and the protocol bridges:

- **Device Manager**

For a device to connect, it must first be registered with Cloud IoT Core. Registration consists of adding a device to a collection (the registry) and defining some essential properties. You can register a device with Cloud Platform Console, gcloud commands, or the REST-style API.

Collectively, the features that allow you to register, monitor, and configure devices are called the device manager.

- **Protocol bridges (MQTT and HTTP)**

Cloud IoT Core supports two protocols for device connection and communication: MQTT and HTTP. Devices communicate with Cloud IoT Core across a "bridge" — either the MQTT bridge or the HTTP bridge. When you create a device registry, you select protocols to enable MQTT, HTTP, or both.

Device telemetry data is forwarded to a Cloud Pub/Sub topic, which can then be used to trigger Cloud functions. You can also perform streaming analysis with Cloud Dataflow or custom analysis with your own subscribers. With Cloud IoT Core, you can control a device by modifying its configuration. A device configuration is an arbitrary, user-defined blob of data that may or may not be structured. If your devices use MQTT, configurations are automatically propagated to them. If your devices connect over HTTP, they must explicitly request configurations.

1.3.1.1 Key Features

(1) Device Security

Security is a critical concern when deploying and managing IoT devices. Cloud IoT Core offers the following security features:

- Per-device public/private key authentication using JSON Web Tokens (JWTs).
This limits the surface area of an attack, because a compromised key would affect only a single device and not the whole fleet.
- Support for RSA or Elliptic Curve algorithms to verify signatures, with enforcement for strong key sizes.
- Support to rotate keys per device by allowing concurrent keys to be registered, and support for expiration time per credential.
- TLS 1.2 connection, using root certificate authorities (required for MQTT).
- Cloud IoT Core API access is controlled by Cloud Identity and Access Management (IAM) roles and permissions.

(2) Per-device Key Authentication

The following diagram shows authentication in Cloud IoT Core.

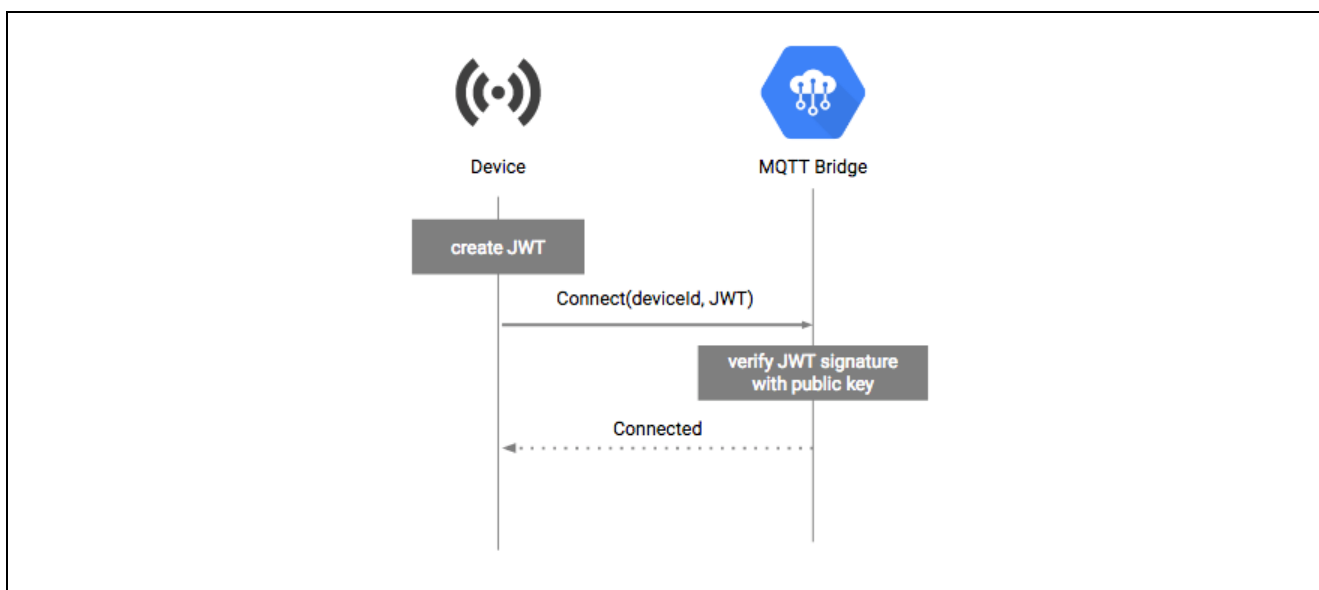


Figure 3. Authentication in Cloud IoT Core

The aspects involved in authentication are as follows:

- The device prepares a JSON Web Token (JWT). The JWT is signed with the private key from the authentication flow.
- When connecting to the MQTT bridge, the device presents the JWT as the password in the MQTT CONNECT message. The username content is ignored. However, some MQTT client libraries will not send the password unless the username is specified. For best results, set the username to an arbitrary value like unused or ignored.
- The MQTT bridge verifies the JWT against the device's public key.
- The MQTT bridge accepts the connection.
- The connection is closed when the JWT expires.

(3) Key Rotation

Cloud IoT Core supports multiple active keys (up to 3 per device) to allow for uninterrupted rotation. The service will try to verify JWTs with each of the active keys, and will accept a connection if any active key matches.

The API allows you to define an **expirationTime** for each device credential (public key). After it expires, the key will be ignored, but it will not be automatically deleted. A 10-minute clock-skew allowance applies. If no expiration time is specified for a key, it will never expire.

1.4 MQTT Protocol Overview

MQTT stands for Message Queuing Telemetry Transport. MQTT is a Client Server publish-subscribe messaging transport protocol. It is an extremely lightweight, open, simple messaging protocol, designed for constrained devices, as well as low-bandwidth, high-latency, or unreliable networks. These characteristics make it ideal for use in many situations, including constrained environments, such as communication in Machine to Machine (M2M) and IoT contexts, where a small code footprint is required, and/or network bandwidth is at a premium.

A MQTT client can publish information to other clients through a broker. A client, if interested in a topic, can subscribe to the topic through the broker. A broker is responsible for authentication, authorization of clients, as well as delivering published messages to any of its clients who subscribe to the topic. In this publisher/subscriber model, multiple clients may publish data with the same topic. A client will receive the messages published if the client subscribes to the same topic.

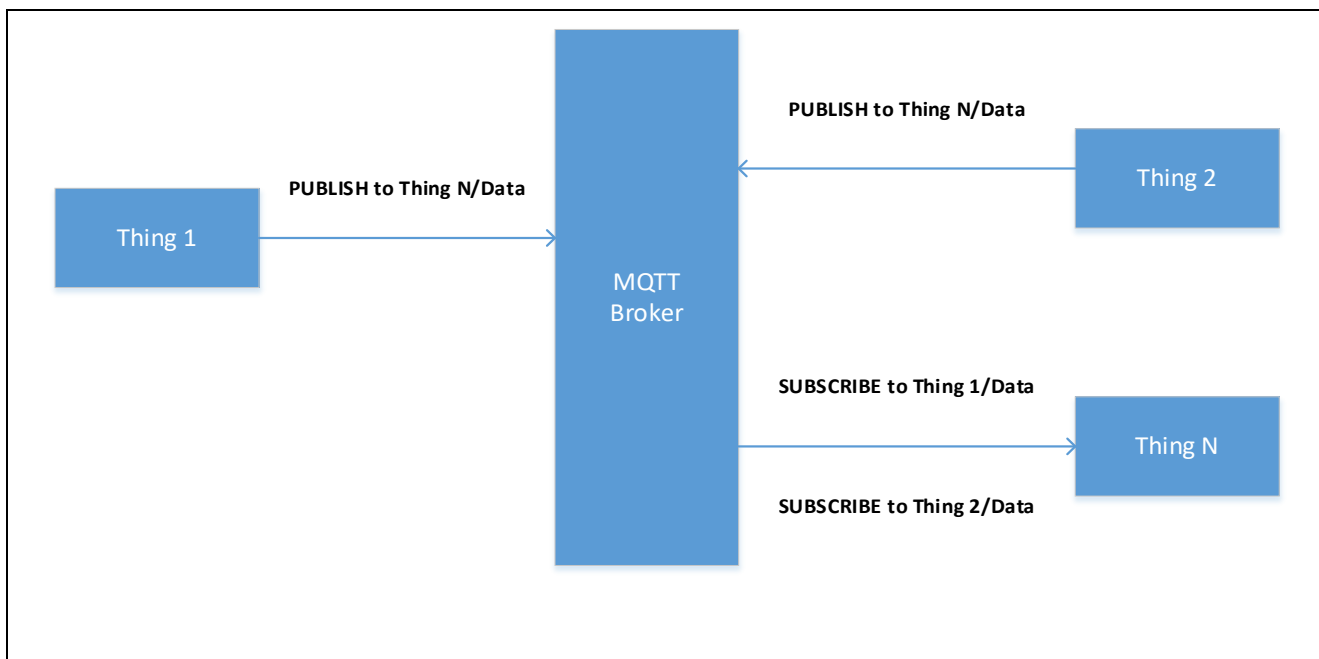


Figure 4. MQTT Client Publish/Subscribe Model

In this model, there is no direct connection between a publisher and the subscriber. To handle the challenges of a pub/sub system, the MQTT generally uses quality of service (QoS) levels. There are three QoS levels in MQTT:

- At most once (0)
- At least once (1)
- Exactly once (2)

At most once (0)

A message will not be acknowledged by the receiver or stored and redelivered by the sender.

At least once (1)

It is guaranteed that a message will be delivered at least once to the receiver. But the message can also be delivered more than once. The sender will store the message until it gets an acknowledgment in form of a PUBACK command message from the receiver.

Exactly once (2)

It is guaranteed that each message is received only once by the counterpart. It is the safest and the slowest quality of service level. The guarantee is provided by two flows, there and back, between sender and receiver.

1.5 TLS Protocol Overview

Transport Layer Security (TLS) protocol and its predecessor, Secure Sockets Layer (SSL), are cryptographic protocols that provide communications security over a computer network.

The TLS/SSL protocol provides privacy and reliability between two communicating applications. It has the following basic properties:

Encryption: The messages exchanged between communicating applications are encrypted to ensure that the connection is private. Symmetric cryptography mechanism such as AES (Advanced Encryption Standard) is used for data encryption.

Authentication: A mechanism to check the peer's identity using certificates.

Integrity: A mechanism to detect message tampering and forgery to ensure that connection is reliable. Message Authentication Code (MAC) such as Secure Hash Algorithm (SHA) is used to ensure message integrity.

TLS/SSL uses TCP but provides secure communication for application layer protocols, such as HTTP and MQTT.

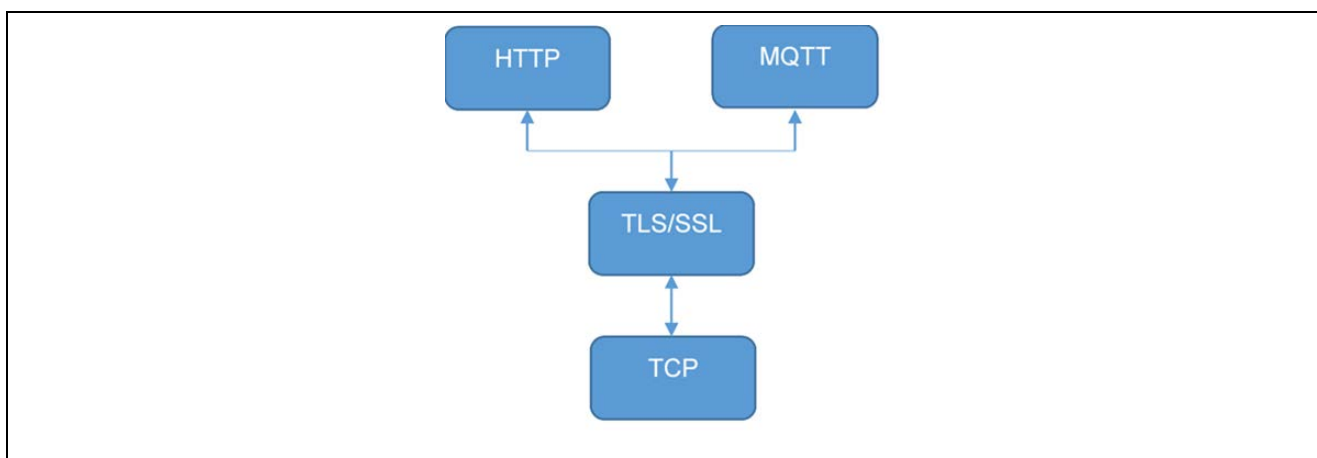


Figure 5. TLS/SSL Hierarchy

1.5.1 Device Certificates and Keys

Devices certificates, public and private keys, and the ways they can be generated, are discussed in this section.

1.5.1.1 Device Certificates

Security is a critical concern when deploying and managing IoT devices. In general, each of the IoT devices needs an identity before they can communicate with the Cloud. Digital certificates are the most common method for authenticating a remote host in TLS. Essentially, a digital certificate is a document with specific formatting that provides identity information for a device.

TLS normally uses a format called X.509, a standard developed by the International Telecommunication Union (ITU), though other formats for certificates may apply if the TLS hosts can agree on the format to use. X.509 defines a specific format for certificates and various encoding that can be used to produce a digital document. Most X.509 certificates used with TLS are encoded using a variant of ASN.1, another telecommunication standard. Within ASN.1 there are various digital encodings, but the most common encoding for TLS certificates is the Distinguished Encoding Rules (DER) standard. DER is a simplified subset of the ASN.1 Basic Encoding Rules (BER) and designed to be unambiguous to make parsing easier.

Though DER-formatted binary certificates are used in the actual TLS protocol, they may be generated and stored in a number of different encodings, with file extensions such as `.pem`, `.crt`, and `.p12`. The most common of the alternative certificate encodings is Privacy-Enhanced Mail (PEM). The PEM format is a base64 encoded version of the DER encoding.

Depending on your application, you may generate your own certificates, be provided certificates by a manufacturer or government organization, or purchase certificates from a commercial certificate authority.

1.5.1.2 Loading Certificates onto your Device

To use a digital certificate in your NetX™ Secure application, you must first convert your certificate into a binary DER format, and optionally, convert the associated private key into a binary format, typically, a PKCS#1-formatted, DER-encoded RSA key. Once converted, it is up to you as to how to load the certificate and the private key on to the device. Possible options include using a flash-based file system or generating a C array from the data (using a tool such as `xxd` from Linux® with the `-i` option) and compiling the certificate and key into your application as constant data.

Once your certificate is loaded on the device, you can use the TLS API to associate your certificate with a TLS session.

1.5.1.3 Generating Self-Signed Certificates

You may also choose to generate a self-signed certificate for testing purposes. The command to generate such a certificate is:

```
openssl req -x509 -newkey rsa:2048 -keyout private.key -out cert.pem -days 365
-nodes -subj "/C=US/ST=Oregon/L=Portland/O=Company
Name/OU=Org/CN=www.example.com"
```

In this situation, a self-signed certificate `www.example.com` is generated. The certificate and private key files are `cert.pem` and `private.key`. You may generate a certificate for **localhost** by replacing **www.example.com** with **localhost**. In that case, pass **localhost** as the first-argument to the installation script.

1.5.2 Device Security Recommendations

The following security recommendations are not enforced by Cloud IoT Core but will help you secure your devices and connections.

- The private key should be kept secret.
- Use TLS 1.2 when communicating with IoT Cloud and verify that the server certificate is valid using root certificate authorities.
- Each device should have a unique public/private key pair. If multiple devices share a single key and one of those devices is compromised, an attacker could impersonate all the devices that have been configured with that one key.
- Keep the public key secure when registering it with Cloud IoT Core. If an attacker can tamper with the public key and trick the provisioner into swapping the public key and registering the wrong public key, the attacker will subsequently be able to authenticate on behalf of the device.
- The key pair is used to authenticate the device to Cloud IoT Core and should not be used for other purposed or protocols.
- Depending on the device's ability to store keys securely, key pairs should be rotated periodically. When practical, all keys should be discarded when the device is reset.
- If your device runs an operating system, make sure you have a way to securely update it. Android Things provides a service for secure updates. For devices that do not have an operating system, ensure that you can securely update the device's software if security vulnerabilities are discovered after deployment.

2. Synergy MQTT/TLS Cloud Solution

2.1 MQTT Client Overview

The NetX Duo MQTT Client module provides high-level APIs for a Message Queuing Telemetry Transport (MQTT) protocol-based client. The MQTT protocol works on top of TCP/IP, and therefore, the MQTT client is implemented on top of the NetX Duo IP and NetX Duo Packet pool. NetX Duo IP attaches itself to the appropriate link layer frameworks, such as Ethernet, Wi-Fi, or cellular.

The NetX Duo MQTT client module can be used in normal or secure mode. In normal mode, the communication between the MQTT client and broker is not secure. In secure mode, the communication between the MQTT client and broker is secured using the TLS protocol.

2.2 Design Considerations

- By default, the MQTT client does not use TLS; communication is not secure between a MQTT client and broker.
- The Synergy MQTT client does not add the NetX Duo TLS session block. It only adds the NetX Duo TLS common block. This block defines/controls the common properties of NetX Secure.
- It is the responsibility of the user/application code to create the TLS session, configure the security parameters, and load the relevant certificates manually under the TLS setup callback provided by `nxd_mqtt_client_secure_connect ()` API.

2.2.1 Supported Features

NetX Duo MQTT Client supports the following features:

- Compliant with OASIS MQTT Version 3.1.1 Oct 29, 2014. The specification can be found at <http://mqtt.org/>.
- Provides an option to enable/disable TLS for secure communication using NetX Secure in SSP.
- Supports QoS and provides the ability to choose levels that can be selected while publishing the message.
- Internally buffers and maintains the queue of received messages.
- Provides a mechanism to register callback when a new message is received.
- Provides a mechanism to register callback when connection with the broker is terminated.

2.2.2 Operational Flow Sequence

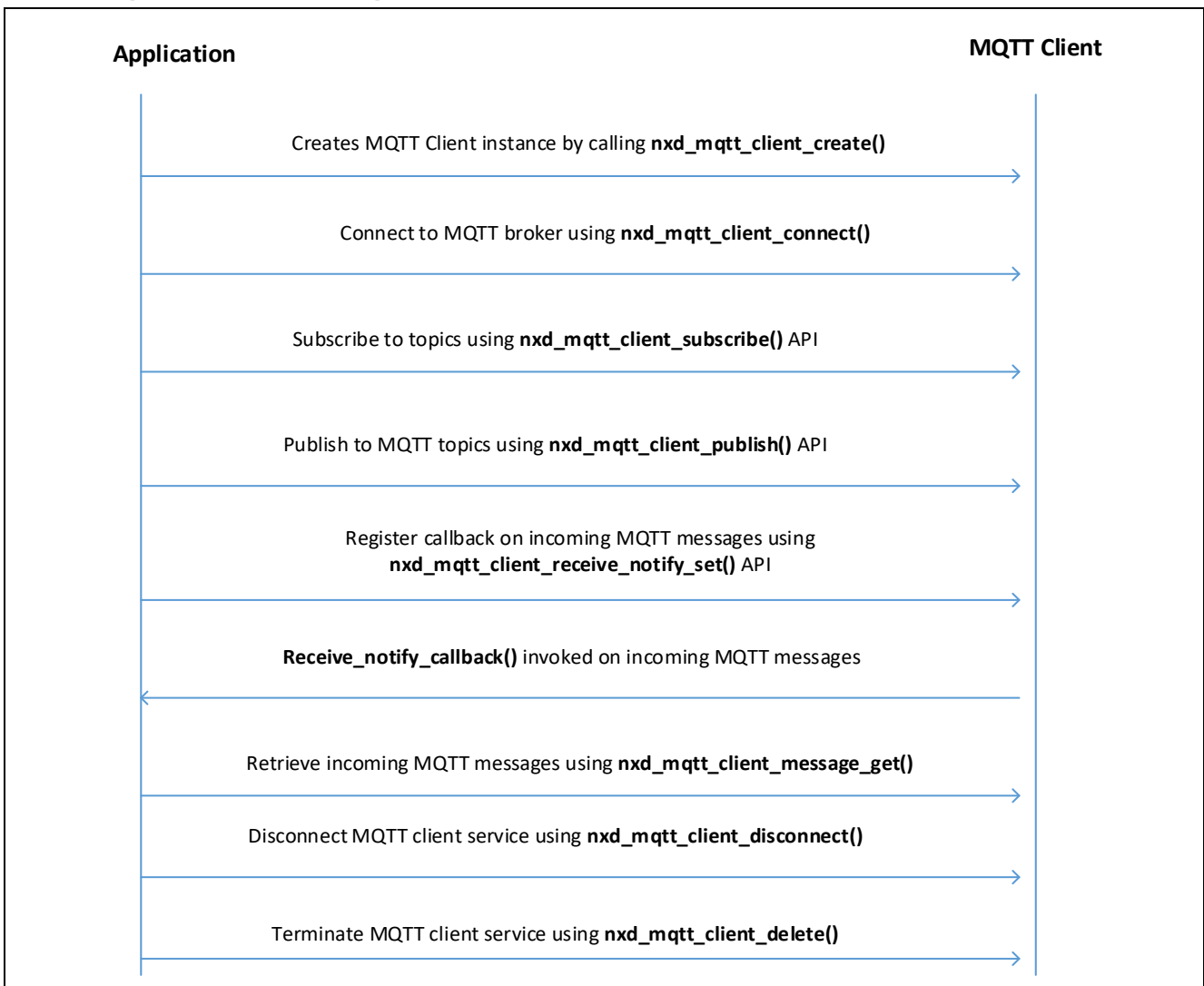


Figure 6. Synergy MQTT Client Flow Sequence

2.3 TLS Session Overview

The NetX Duo TLS session module provides high-level APIs for the TLS protocol-based client. It uses services provided by the Synergy Crypto Engine (SCE) to carry out hardware-accelerated encryption and decryption.

The NetX Duo TLS Session module is based on Express Logic's NetX Secure, which implements the Secure Socket Layer (SSL) and its replacement, TLS protocol, as described in RFC 2246 (version 1.0) and 5246 (version 1.2). NetX Secure also includes routines for the basic X.509 (RFC 5280) format. NetX Secure is intended for applications using ThreadX RTOS in the project.

2.3.1 Design Considerations

- NetX Secure TLS performs only basic path validation on incoming server certificates.
- Once the basic path validation is complete, TLS then invokes the certificate verification callback supplied by the application.
- It is the responsibility of the application to perform any additional validation of the certificate.
- To help with the additional validation, NetX Secure provides X.509 routines for common validation operations, including DNS validation and Certificate Revocation List checking.
- Software-based cryptography is processor intensive.
- NetX Secure software-based cryptographic routines have been optimized for performance, but depending on the power of the target processor, performance may result in very long operations. When hardware-based cryptography is available, it should be used for optimal performance of the NetX Secure TLS.
- Due to the nature of embedded devices, some applications may not have the resources to support the maximum TLS record size of 16 KB.
- NetX Secure can handle 16 KB records on devices with sufficient resources.

2.3.2 Supported Features

- Support for RFC 2246 The TLS Protocol Version 1.0
- Support for RFC 5246 The Transport Layer Security (TLS) Protocol Version 1.2
- Support for RFC 5280 X.509 PKI Certificates (v3)
- Support for RFC 3268 Advanced Encryption Standard (AES) Cipher suites for Transport Layer Security (TLS)
- RFC 3447 Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1
- RFC 2104 HMAC: Keyed-Hashing for Message Authentication
- RFC 6234 US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)
- RFC 4279 Pre-Shared Key Cipher suites for TLS

2.3.3 Operational Flow Sequence

This section describes the TLS handshake operational sequence.

2.3.3.1 TLS Handshake

The following figure shows a typical TLS handshake between the TLS Server and Client.

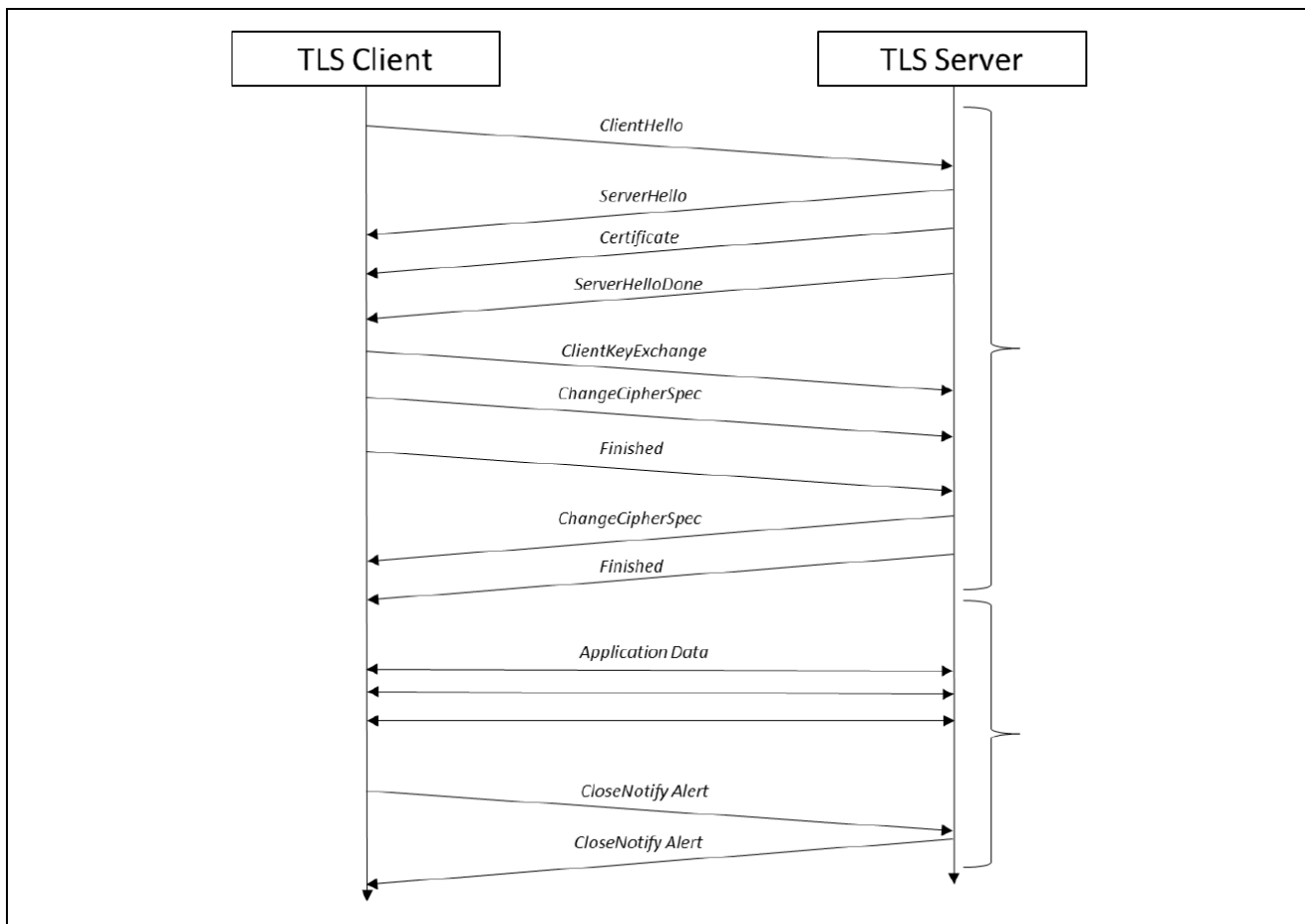


Figure 7. TLS Handshake

- A TLS handshake begins when the TLS client sends a **ClientHello** message to a TLS server, indicating its desire to start a TLS session.
- The message contains information about the encryption that the client would like to use for the session, along with information used to generate the session keys.
- The TLS server responds to the **ClientHello** with a **ServerHello** message, indicating a selection from the encryption options provided by the client.
- It is followed by a Certificate message, in which the server provides a digital certificate to authenticate its identity to the client.
- Finally, the server sends a **ServerHelloDone** message to indicate that it has no more messages to send.
- Once the client has received all the server's messages, it has enough information to generate the session keys. TLS does this by creating a shared bit of random data called the Pre-Master Secret, which is of a fixed size and is used as a seed to generate all the keys needed once encryption is enabled.
- The Pre-Master Secret is encrypted using the public key algorithm (such as RSA) specified in the Hello messages and the public key provided by the server in its certificate.
- The encrypted Pre-Master Secret is sent to the server in the **clientKeyExchange** message. The server, upon receiving the **ClientKeyExchange** message, decrypts the Pre-Master Secret using its private key and proceeds to generate the session keys in parallel with the TLS client.
- Once the session keys are generated, all further messages can be encrypted using the private-key algorithm (such as AES) selected in the Hello messages. One final un-encrypted message called **ChangeCipherSpec** is sent by both the client and server to indicate that all further messages will be encrypted.

- The first encrypted message sent by both the client and server is also the final TLS handshake message, called **Finished**. This message contains a hash of all the handshake messages received and sent. This hash is used to verify that none of the messages in the handshake have been tampered with or corrupted.
- Now, the application begins sending and receiving data. All data — sent by either side — is first hashed using the hash algorithm chosen in the Hello messages, and then encrypted using the chosen private-key algorithm with the generated session keys.
- Finally, a TLS session can only be successfully ended if either the Client or Server chooses to do so. Both the client and server must send and process a **CloseNotify** alert for a successful session shutdown.

2.3.3.2 Initialization Flow Sequence

A typical TLS session initialization flow sequence is given in the following figure.

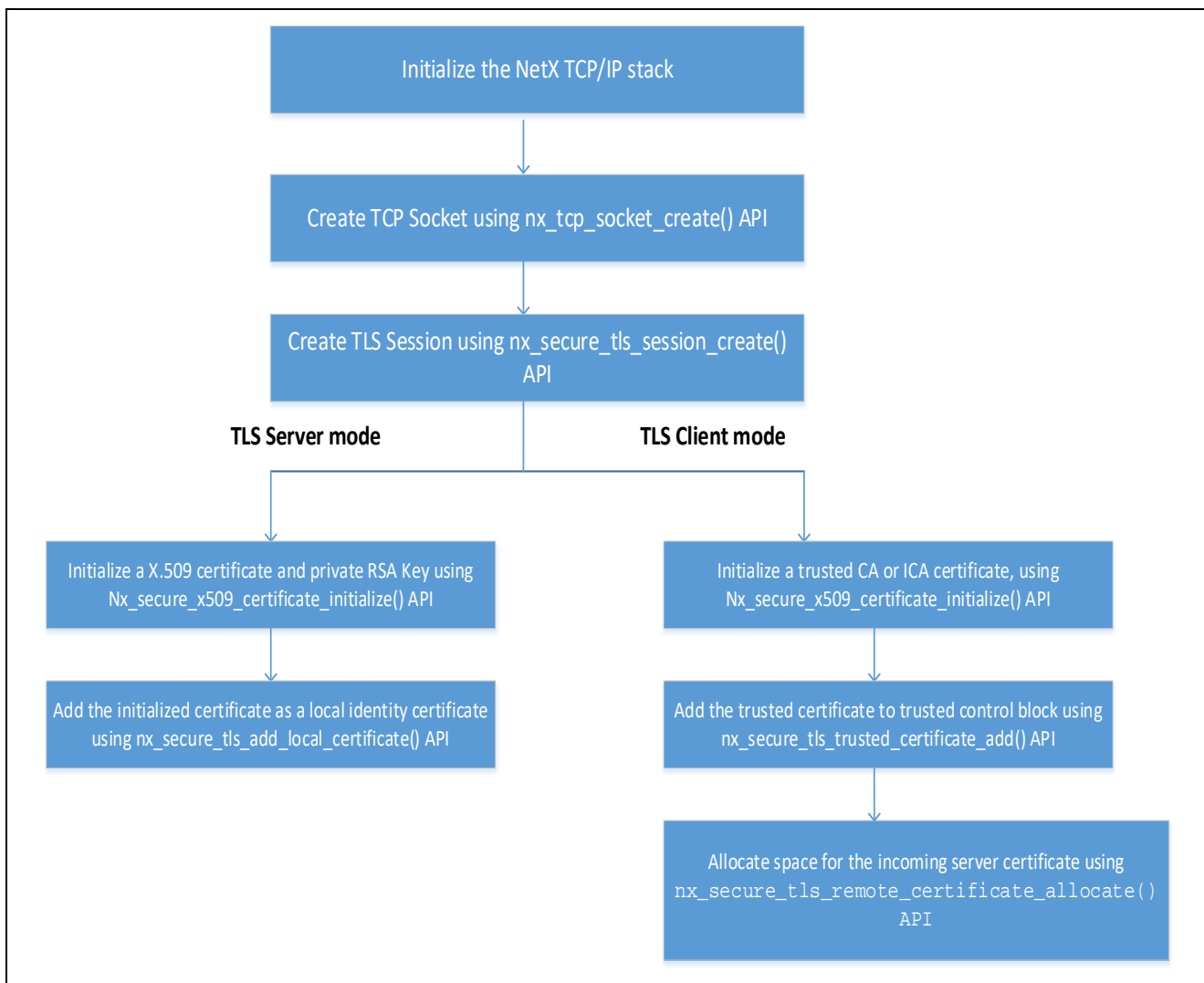


Figure 8. Synergy TLS Session Initialization

2.3.3.3 Data Communication Flow Sequence

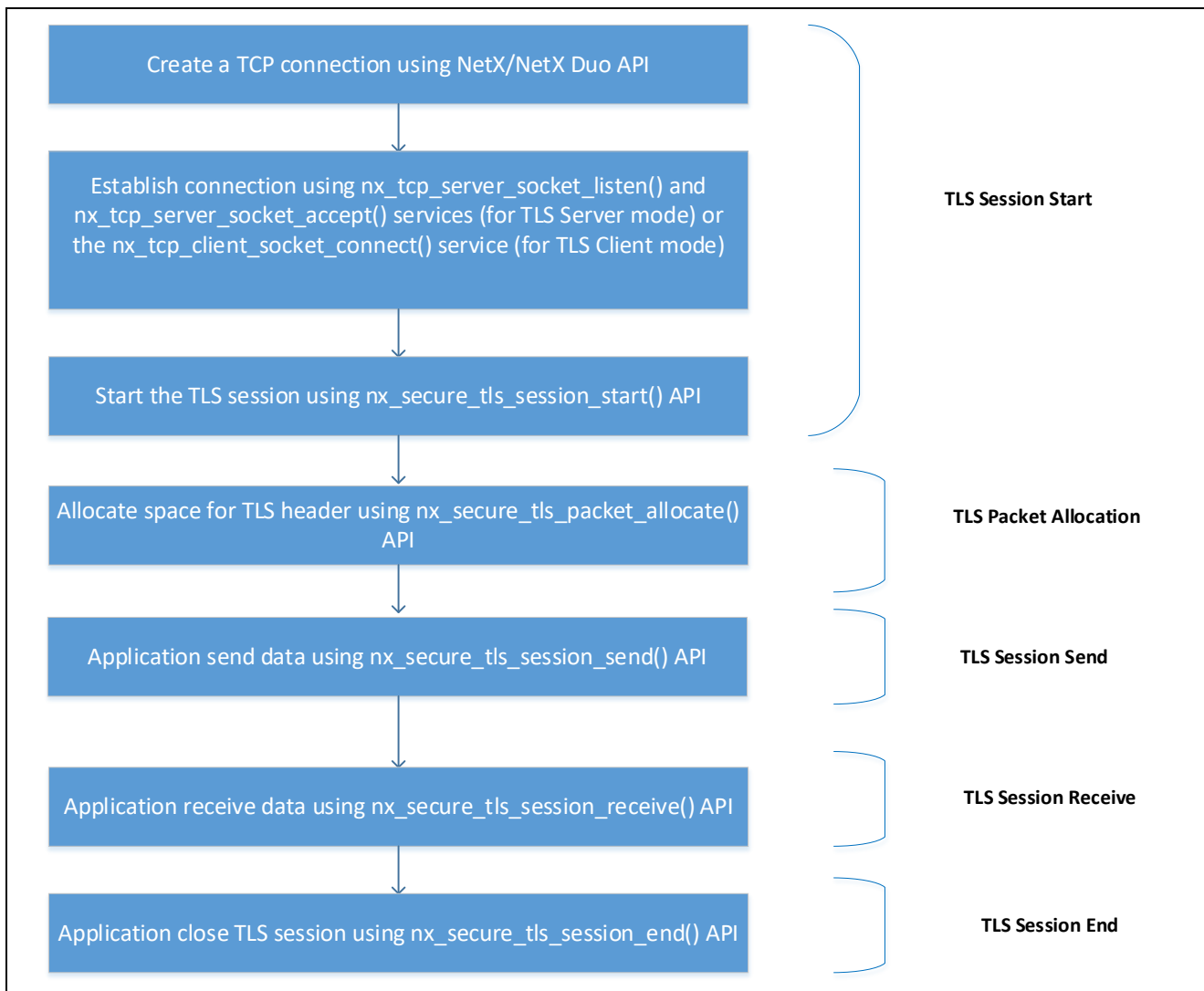


Figure 9. Synergy TLS Session Data Flow Sequence

3. MQTT/TLS Application Example

3.1 Application Overview

This example application project demonstrates the Renesas Synergy™ IoT Cloud Connectivity solution using the onboard Synergy MQTT/TLS modules. For demonstration purposes, this application uses Google Cloud IoT Core as the cloud provider. Ethernet or Wi-Fi or Cellular (supported only on AE-CLOUD2 kit) are used as the primary communication interface between the Thing and Google Cloud IoT Core.

In this example, the AE-CLOUD1/AE-CLOUD2 kit acts as an MQTT node/Thing, connects to the Google Cloud IoT Core, periodically reads the on-board sensor values, and publishes this information to the Google Cloud IoT Core. It also subscribes to its User LED state MQTT topic. You can turn the User LEDs ON/OFF by publishing the LED state remotely. This application reads the updated LED state and turns the User LEDs ON/OFF.

The steps here use the MQTT Subscriber from Google Cloud IoT Core to subscribe to the MQTT topics published by AE-CLOUD1/AE-CLOUD2 kit. Follow the instructions in section 3.3 to setup the MQTT client on Google Cloud IoT Core and run this demonstration. However, you are free to use any known MQTT client to subscribe to the MQTT topics published by AE-CLOUD1/AE-CLOUD2 Synergy MCU kit.

3.2 Software Architecture Overview

The following figure shows the overall software architecture for the Synergy Cloud Connectivity Application Example Project.

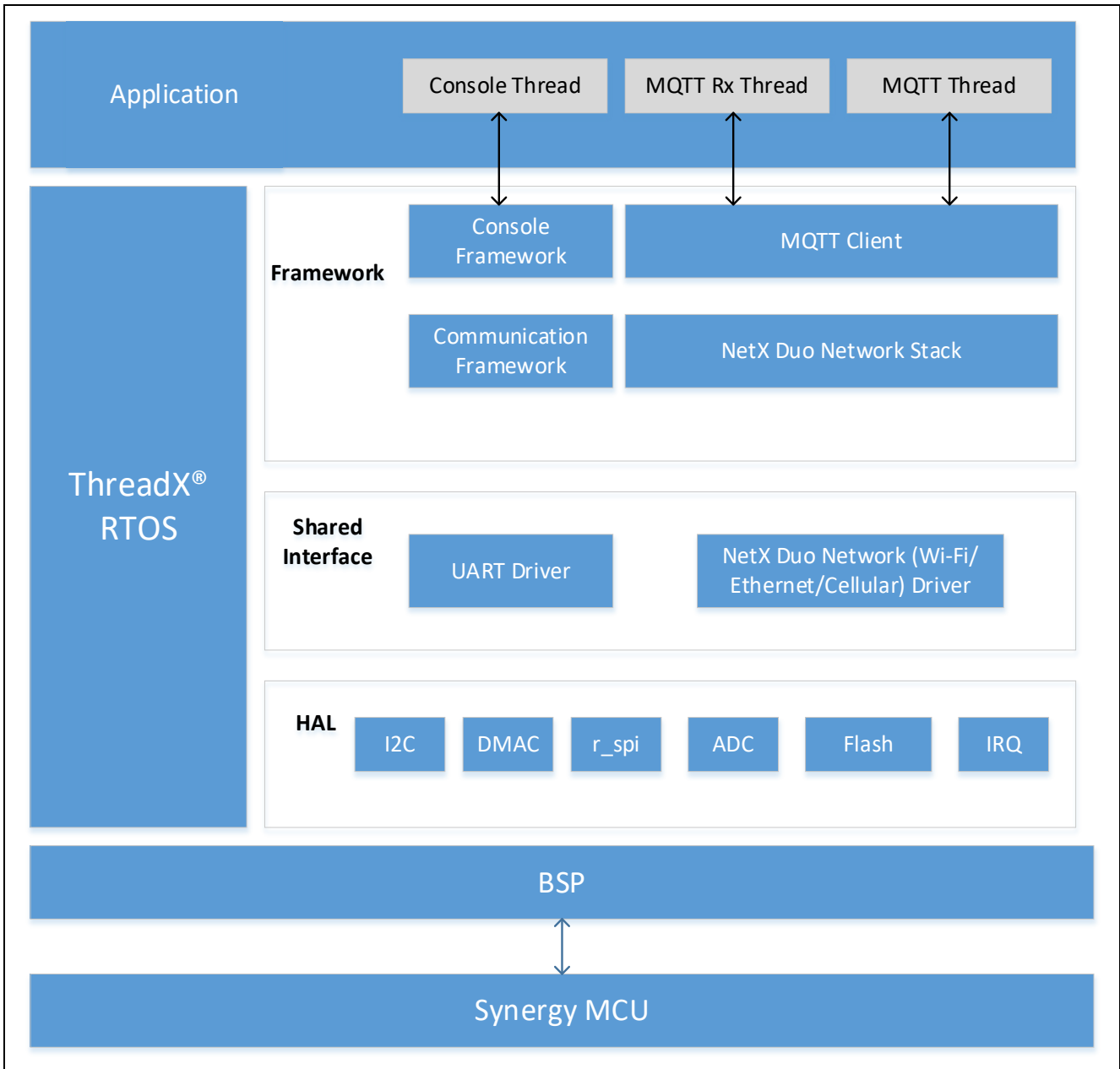


Figure 10. Synergy Cloud Connectivity Application Software Architecture

The main software components of this application are:

- MQTT Client
- NetX Duo IP Stack and its underlying driver components for Ethernet and Wi-Fi
- Console Framework

This application contains the following application threads:

- Console Thread
- MQTT Thread
- MQTT Rx Thread

3.2.1 Console Thread

This thread handles the function related to Common Line Interface (CLI). It uses the console framework, which in-turn uses the communication framework and its underlying USBX CDC device module components.

This thread reads the user inputs and stores them in the internal data flash. The stored information is read later by the MQTT Thread when it tries to run the Synergy Cloud connectivity demo.

This thread presents you with the following CLI command options:

- `Cwiz`
- `Demo start/stop`

Cwiz command option

Using this command option, you can select the following configurations:

- Network interface such as Ethernet, Wi-Fi and its associated IP mode (DHCP/Static).
- IoT Cloud selection (GCloud).
- Dump the existing configuration from flash.
- Exit the menu.

Demo start/stop command option

Using this command option, you can run/stop the Synergy Cloud Connectivity Demonstration.

3.2.2 MQTT Thread

The MQTT thread is the main control thread which handles the following major functions:

- Initializes communication interface (Ethernet/Wi-Fi).
- Initializes IoT Cloud interface.
- Reads sensor data and publishes the data periodically on MQTT topics.
- Updates the user LED state based on the type of MQTT message received.

On wakeup, this thread periodically checks (every 5 seconds) for user input event flag state set once you enter the demonstration start/stop command on the CLI. If the `demo start` command is issued from the CLI, this thread will read the pre-configured user information from internal flash and checks its validity. If the content is valid, it then starts the Synergy Cloud connectivity demonstration. If a `demo stop` command is issued, it de-initializes the IoT Cloud interface.

3.2.3 MQTT Rx Thread

This thread handles the incoming MQTT messages from the MQTT broker. On receiving the new MQTT message, the user callback `receive_notify_callback()` will be invoked by the MQTT thread. This callback in turn sets the semaphore on which the MQTT Rx Thread is polling periodically.

On receiving the new MQTT message, it uses the `nxd_mqtt_client_message_get()` API to read the message, parse it, and act on it based on the type of the message received.

3.3 IoT Cloud Configuration (GCloud)

Device Registry

A container of devices with shared properties. You “register” a device with a service (like Cloud IoT Core) so that you can manage it.

Device

A “Thing” in the “Internet of Things”; a processing unit that can connect to the internet and exchange data with the Cloud. Devices are often called “smart devices” or “connected devices”. They communicate two types of data: telemetry and state.

Device configuration

An arbitrary, user-defined blob of data used to modify a device's settings. Configuration data can be structured or unstructured, and flows only in the cloud-to-device direction.

Device state

An arbitrary, user-defined blob of data that describes the status of the device. Device state data can be structured or unstructured, and flows only in the device-to-cloud direction.

JSON

JSON is an open standard, lightweight, data-interchange format. As a text document, it is easy for users to read and write, and for machines to parse and generate.

JSON is completely language independent, using conventions that are familiar to C-family programmers, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. An example JSON script used to turn on a LED is given as follows:

```
{
  "state": {
    "desired": {
      "LED_value": "On"
    }
  }
}
```

Google Cloud Signup

Google Cloud offers a free account (12 months) for each user. It is expected that each user has created an account on the Google Cloud IoT service before continuing to the next section.

Each user is expected to have a Gmail account that will be used to login to the Google Cloud IoT core account. To link your Gmail account to a Google Cloud IoT core account, open to the following link in your web browser: <https://console.cloud.google.com/>.

Fill in the required details and create a user account.

3.3.1 Creating a Device on Google Cloud IoT Core

The following steps show you how to create a project, registry, and how to add a device on the Google Cloud IoT Core user account. It is assumed that you created a user account in the Google Cloud IoT Core and have followed the Google Cloud signup procedure.

Note: While you are creating the project, registry and device, the screenshots may look slightly different from what is shown in the document and users need to navigate in the Google Cloud IoT core environment to find the corresponding attribute while working on this project.

3.3.1.1 Create a Project

1. After you sign up for the Google Cloud free tier, open the following link to enter the Google Cloud Platform dashboard: <https://console.cloud.google.com/>
2. To the right of “Google Cloud Platform” at the top of the window is a drop-down menu as shown in the following figure. Select **NEW PROJECT**.

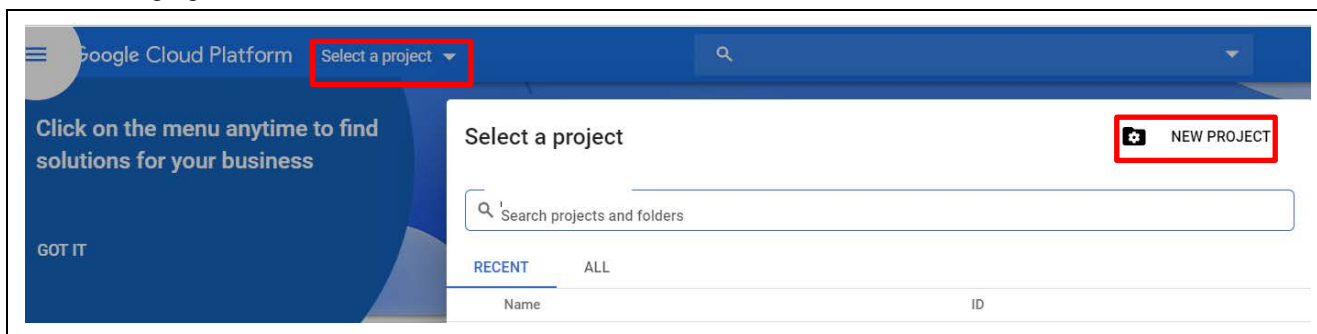


Figure 11. Project Selection

3. Enter the project name as shown and make a note of the project ID as highlighted in the following screenshot. It is a longer string with a unique number.

Note: Remember to store this project ID. This information is passed to the firmware using the serial console as part of the configuration.

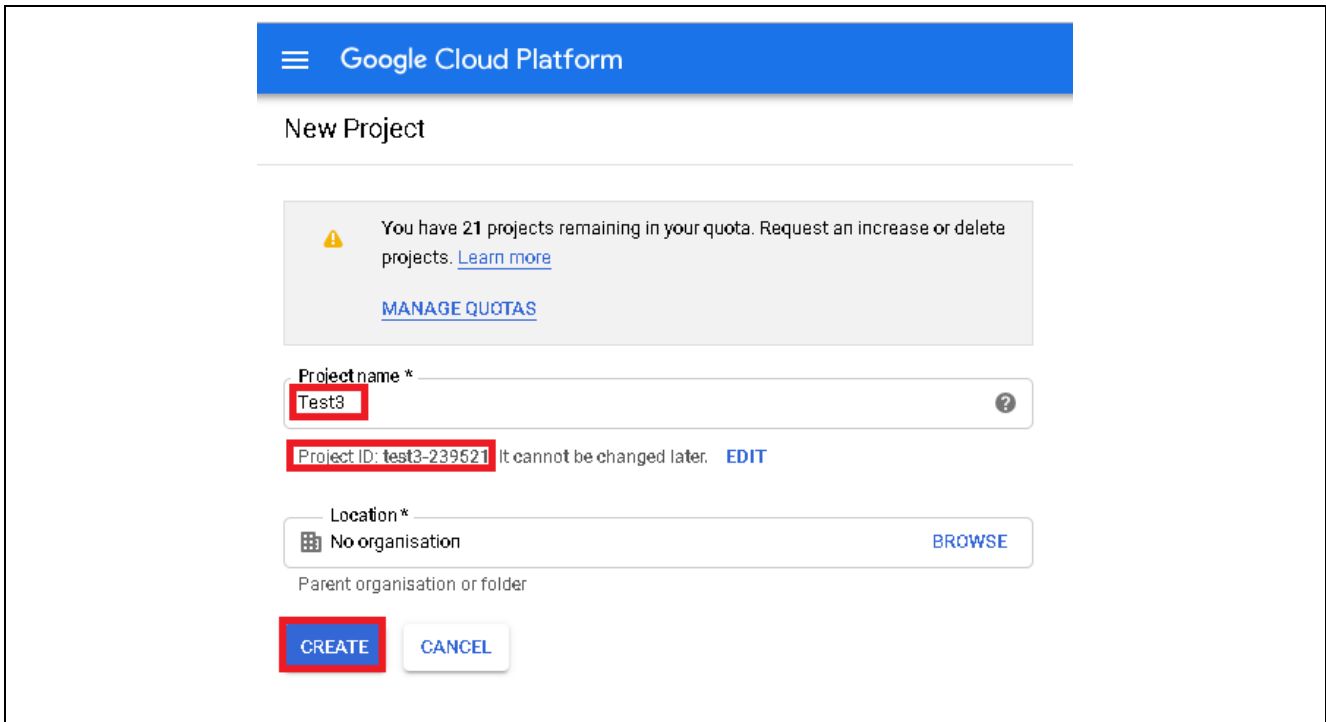


Figure 12. Project Name

3.3.1.2 Create a Device Registry

1. Go to the Google Cloud IoT Core page (<https://console.cloud.google.com/>) in Google Cloud Platform (GCP) Console.
2. Go to the IoT Core page as shown in the following screenshot.

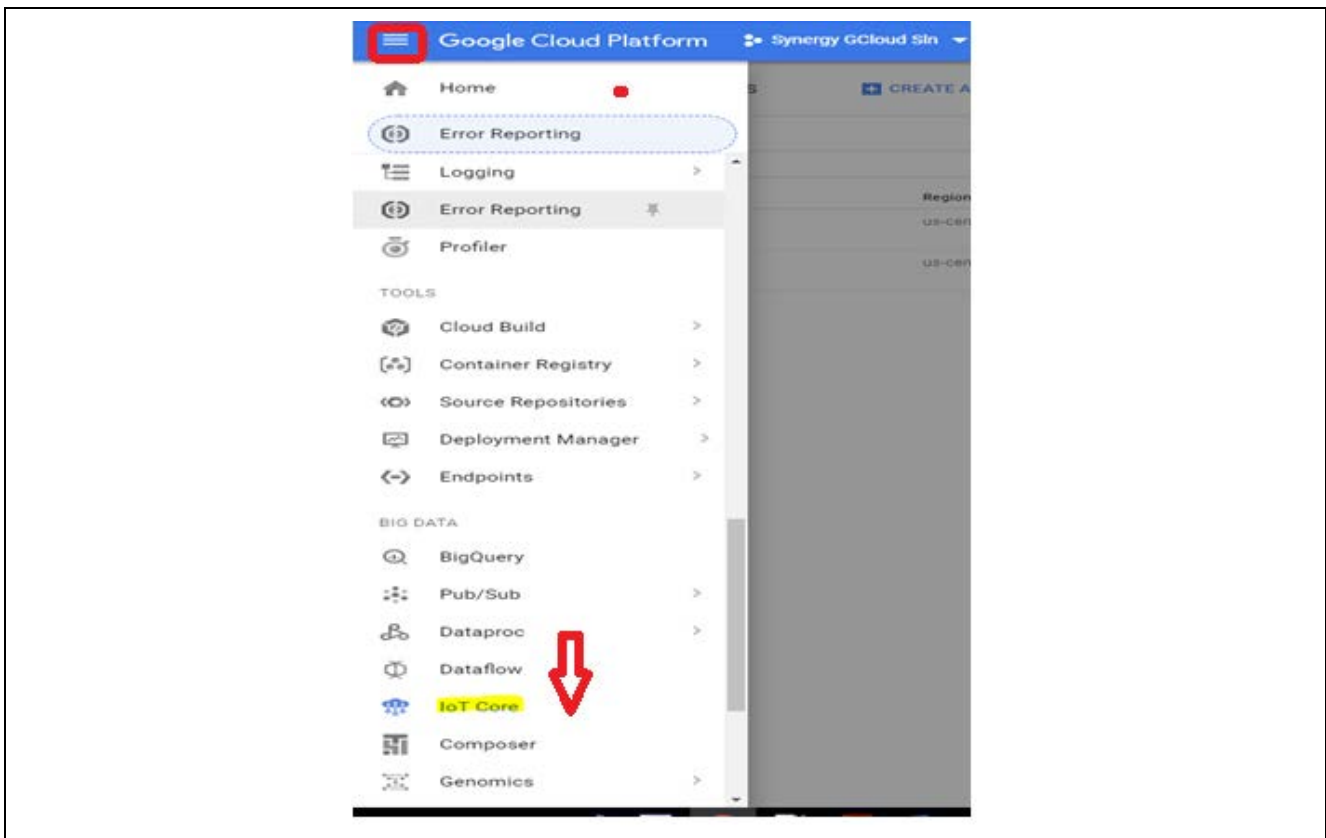


Figure 13. Device Registry Creation

- Click **Enable API** (If the message pops up on the screen)
- 3. Click **Create a device registry**.
- 4. Enter *<my-registry>* for the **Registry ID**.
- 5. If you are in the US, select **us-central1** for the **Cloud region**. If you are outside the US, select your preferred region (**Europe-west1** or **asia-east1**).
- 6. Select **MQTT** for the **Protocol**.
- 7. In the **Default telemetry** topic, keep the default setting.
- 8. In the **Device state** topic dropdown list, select **Create a topic**.
- 9. In the **Create a topic** dialog, enter *<my-device-events>* in the **Name** field.
- 10. Click **Create** in the **Create a topic** dialog. The created topic can be seen in the dropdown menu.
- 11. The **Device state topic** and **Certificate value** fields are optional, so leave them blank.
- 12. Choose **Stackdriver** Logging as **None**.
- 13. Click **Create** on page.
- 14. Note the registry ID.
- 15. Note the cloud region.

You have just created a device registry with a Cloud Pub/Sub topic for publishing device telemetry events.

3.3.1.3 Add a Device to the Registry

- 1. From the created device registry page, you will see the Devices tab under IoT Core; select the **Devices** tab and Click **+ CREATE A DEVICE** to add a device.
- 2. Enter *<my-device>* for the **Device ID**.
- 3. Select **Allow** for **Device communication**.
- 4. The Authentication fields are optional, so temporarily leave them blank or use the default values. The Authentication fields accept a public key that you'll create and implement in the next section.
- 5. The Device metadata field is also optional; leave it blank.
- 6. Use the default settings for the **Stackdriver** Logging.
- 7. Click **Create**.
- 8. Note the device ID.
- 9. Click the **Add public key**. You can visit this later.

You have just added a device to your registry.

3.3.1.4 Create Subscription to the Topic

Open a new browser tab. In the Google Cloud Platform dashboard, scroll down and go to the **Pub/Sub** tab (found under Big Data section); it will take you to the **Pub/Sub** page as shown in the following screenshot. If the **Enable API** message pops up, click it. The telemetry topics created by the user will be displayed under the **Topics** page. Click **Enable API** (if you see the popup screen).

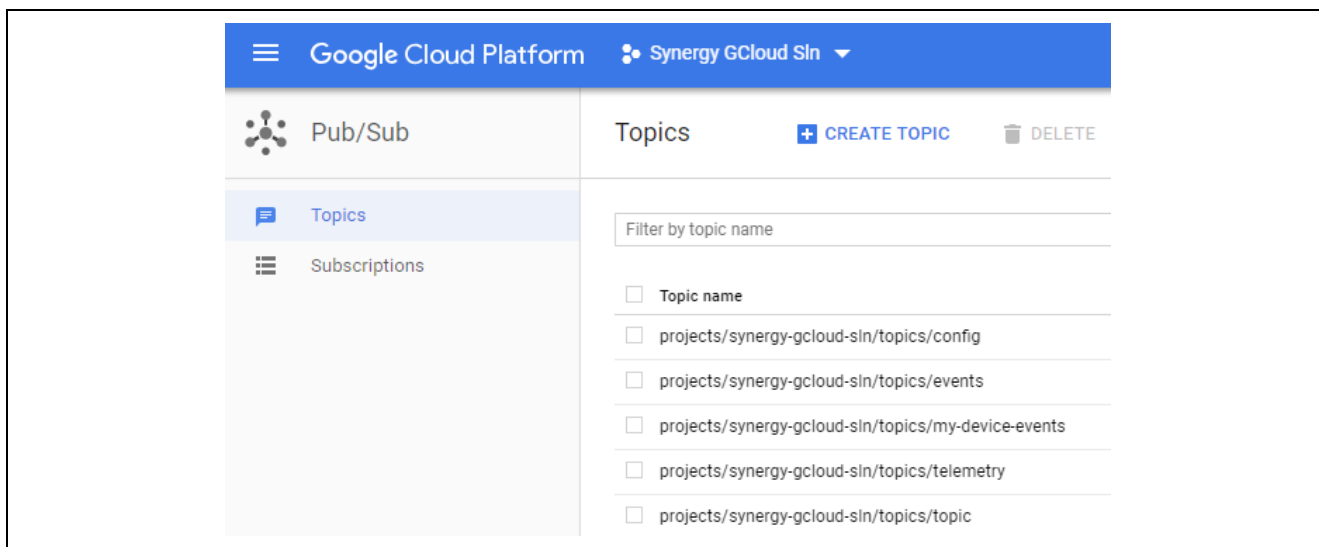


Figure 14. User Created Topics

The steps to create a subscription to the topic are as follows:

1. Find your topic from the list and click on the topic to enter **Topic details** page.
2. Now, you need to create a subscription to capture the stream of messages published to your topic.
3. Click **CREATE SUBSCRIPTION** in the **Topic details** page as shown in the following screenshot, to create a subscription.

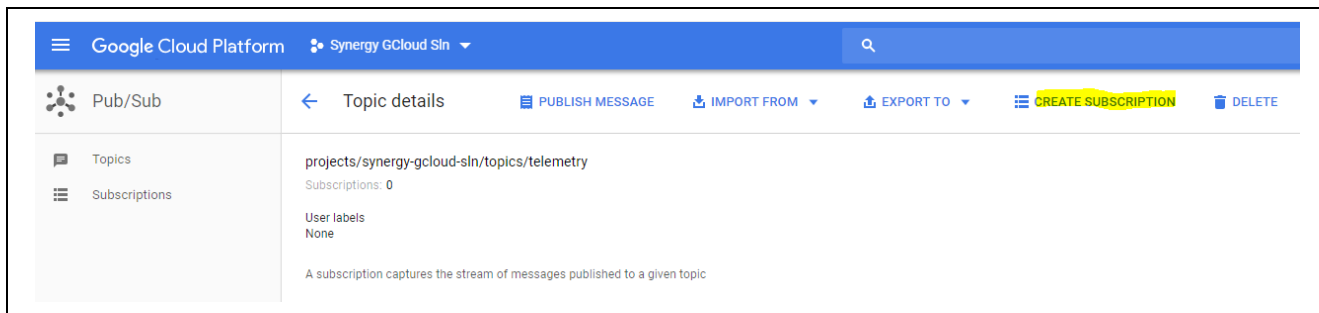


Figure 15. Creating a Subscription

4. Enter the subscription name and choose the **Delivery Type** as **Pull**.
5. Click the **Create** button to create subscription for your topic.

3.3.1.5 Set Permissions for the New Topic

In the Google Cloud Platform dashboard, go to the **Pub/Sub** page as shown in the following screenshot. The telemetry topics created by the user will be displayed under the **Topics** page.

The steps to set permissions for the new topic are as follows:

1. Select the checkbox to the left of the topic name, then click the **Permissions** button.

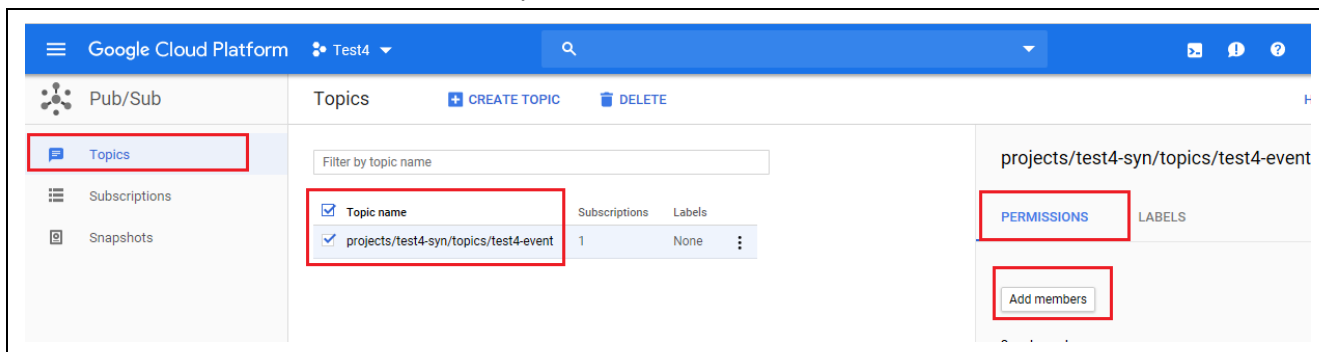


Figure 16. Setting Permissions

2. For testing purpose, enter the gmail address you used for the google cloud registration in the **Add members** box to grant access to all users. In real implementation, the access has to be properly given to certain group of members.
3. Select **Pub/Sub Publisher** from the drop-down menu.
4. Finally, click **Save**.

3.3.2 Generate Device Key and Certificate

Now, you can generate device certificates and keys for the Google Cloud IoT Thing (Thing) created.

1. Make sure you install the `openssl` library in your test PC.
 Note: You may have to create a `config` folder in the `openssl` folder if it does not exist.
2. Open a terminal window in your Windows PC and run the following command to create an RS256 key from the folder where the keys are to be created:

```
openssl req -x509 -newkey rsa:2048 -keyout rsa_private_key.pem -nodes -out
rsa_devcert.pem -subj "/CN=unused"
```

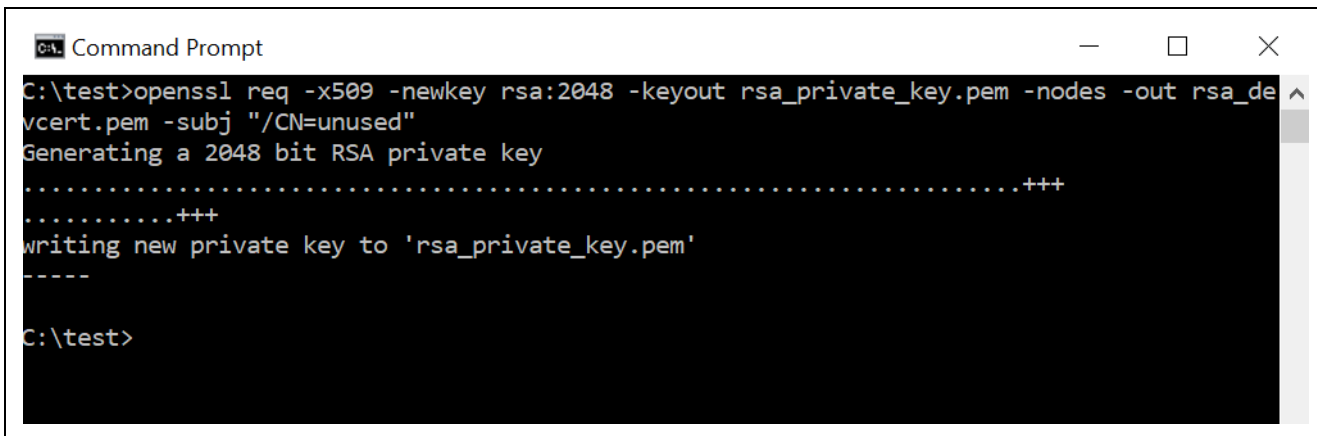


Figure 17. Creating an RS256 Key

In the above example, the keys are created in a `test` folder.

The device private key needs to be converted into PKCS#1 format before passing it to the firmware using serial console as part of configuration. To convert `rsa_private.pem` to PKCS#1 format, run the following command in your command window:

```
openssl rsa -in rsa_private_key.pem -out rsa_private_pkcs1.pem
```



Figure 18. Converting the Key to PKCS#1 Format

Note: This is the private key which will be used for the CLI “`rsa_private_pkcs1.pem`”

3.3.3 Add Public Key to the Google Cloud IoT Core

At this point, it is assumed that you followed the steps described in section 3.3.1, created the registry, and added a device to the registry. Go to the IoT core (section 3.3.1.3, step 8), or the **Devices** tab and click on the created Device ID. This will take you to the **Add public key**.

1. Copy the contents of `rsa_devcert.pem` to the clipboard. Make sure to include lines that say `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----`
2. On the **Device details** page for the device you created in the preceding section, click **Add public key**.
3. Select **RS256_X509** for the **Public key format**.
4. Paste your public key in the **Public key value** box.
5. Click **Add**.
6. An **RS256_X509** key appears on the **Device details** page for your device.

4. Running the MQTT/TLS Application

4.1 Importing, Building, and Loading the Project

See the *Renesas Synergy™ Project Import Guide* (r11an0023eu0121-synergy-ssp-import-guide.pdf), included in this package, for instructions to import the project into e² studio, build, and run the project.

4.2 Manually Adding the Board Support Package for AE-CLOUD1/AE-CLOUD2 Kit

1. From the project bundle, locate the BSP file,
 Renesas.S5D9_PILLAR_ARDUINO_MODULE.1.6.2.pack for AE-CLOUD2 and
 Renesas.S5D9_IOT_BOARD.1.6.2.pack for AE-CLOUD1.
2. e² studio Users
 Copy the file shown in the following figure to the e² studio packs folder location
 C:\Renesas\e2studio_v7.3.0\internal\projectgen\arm\packs.



Figure 19. Loading BSP Pack for AE-CLOUD1/AE-CLOUD2 Kit

3. IAR Users
 Copy the file to SCC \packs folder location
 C:\Renesas\Synergy\ssc_v7.3.0_ssp_v1.6.0\internal\projectgen\arm\packs

Note: If e² studio and IAR SSC are installed in any other location, the same information needs to be provided to copy the pack.

4.3 Powering up the Board

To connect power to the board, connect the SEGGER J-Link® debugger to the PC, connect the board to the PC USB port, and run the debug application, using the following instructions:

1. For AE-CLOUD2, connect the micro USB end of the supplied USB cable to the AE-CLOUD2 board's J6 connector (DEBUG_USB).
 Connect the other end of the USB cable to the USB port on your workstation.
 Note: The kit contains a SEGGER J-Link® On-board (OB). J-Link provides full debug and programming for the AE-CLOUD2 board.
2. On the AE-CLOUD1 board, connect the J-Link Lite supplied with the kit to the J2 connector on the AE-CLOUD1 and to the 10-pin header on the J-Link lite using the supplied 10-pin, flat-ribbon cable.
3. Attach the PMOD-based GT-202 Wi-Fi module in the PMOD A connector.
4. For the AE-CLOUD2 kit, connect the BG96 Cellular shield on the AE-CLOUD2 Arduino Connector. Next, attach the Cellular antenna to the LTE antenna connector, and then the GPS antenna to the GNSS antenna connector on the BG96 shield and attach the PMOD-based GT-202 Wi-Fi module in the PMOD connector all the time in spite of running the demo using ethernet/cellular interface. *This redundant connection is necessary as a workaround to a known issue in SSP v1.6.3 and v1.7.0.*
5. Connect the second micro USB cable as follows, depending on your kit:
 —AE-CLOUD2/AE-CLOUD1 board's J9 connector

Connect the other end of the USB cable to the USB port on your workstation. This connection is necessary for the serial console.

4.4 Connect to Google IoT Cloud

The following instructions show how to run the Synergy Cloud connectivity application project and connect to the Google IoT Cloud.

Note: At this stage, it is assumed that you completed the instructions in section 3.3 to create a Google IoT account, set up your device on the Google IoT Core, and downloaded the device certificates and keys.

- Section 4.4 shows how the command line interface can be used to configure the boards depending upon the desired interface for the cloud connectivity.
- While running the application on these boards, connectivity is done using one interface at a time (Ethernet or Wi-Fi, or Cellular). Users are required to configure only the desired interface to run the application. For example, if you use Ethernet, then Wi-Fi or Cellular does not need to be configured and vice-versa.
- Note that the CLI screenshots shown, in some cases, may not apply to some boards, such as Cellular not being applicable AE-CLOUD1 boards.

Table 1. Kit Connectivity Options (only one interface supported at a time)

Board	Ethernet	Wi-Fi	Cellular
AE-CLOUD1	Supported	Supported	Not Supported
AE-CLOUD2	Supported	Supported	Supported

1. Connect the USB Device port of the kit to the test PC and it will be automatically detected as a USB Serial device in case of Windows 10 PC. In case of Windows® 7/8 PC, refer to the following installation guide to load the Synergy USB CDC driver:

- www.renesas.com/en-us/products/synergy/software/add-ons/usb-cdc-drivers.html
- <https://en-support.renesas.com/knowledgeBase/16977397>

Open the serial console application, such as Tera Term, to connect it to the AE-CLOUD1/AE-CLOUD2 kit. The default Tera Term settings are **8-N-1**, and the baud rate is **9600**.

2. Press **Enter**. The following command prompt and CLI (for AE-CLOUD2 only) information appears on the serial console.

```

*****
*      Renesas Synergy Google IoT Cloud Connectivity Application
*
*      FW version 1.3.0 - Jun 20 2019, 17:57:47
*
*      Synergy Software Package Version: 1.6.2
*****
    
```

Figure 20. Command Prompt (CLI info is for AE-Cloud2 only)

3. Press the ? key on your keyboard to display the available CLI command options shown in the following figure.

```

*****
>?
Help Menu
  cwiz : Network/Cloud Configuration Menu
        Usage: cwiz

  demo : Start/Stop Synergy Cloud Connectivity Demo
        Usage: demo <start>/<stop>
    
```

Figure 21. Help Menu

4.4.1 Configuration Wizard Menu

Enter command `cwiz` and press the enter key in the serial console to enter the configuration menu. This command is used to configure the Network interfaces, Google Cloud IoT Core Service, and dump previous configuration stored in the internal flash.

```
>
>cwiz
##### Main Menu #####
1. Network Interface Selection
2. GCloud IoT Core Configuration
3. Dump previous configuration from flash
4. Exit
Please Enter Your Choice:>
```

Figure 22. Configuration Menu

4.4.1.1 Network Interface Selection

From the configuration menu, press the **1** key to configure the Network Interface. It lists the available network interface options in this application project. Currently this application supports Ethernet, Wi-Fi, Cellular (in case of AE-CLOUD2 kit) communication interfaces.

Note: The user can select only one network interface at a time. For instance, when Ethernet is selected, Wi-Fi and Cellular are not available and vice-versa. To change the network interface, the demo should be stopped, and a new interface should be selected through `cwiz`.

Note: If the user uses the same network interface between power cycles, then the network interface selection can be skipped as the interface and credentials are stored in the flash.

For example, if the user selects Ethernet as the interface for cloud connectivity, the user can then move directly to Google IoT Core Configuration. The same applies for Wi-Fi and Cellular as well.

```
>
>cwiz
##### Main Menu #####
1. Network Interface Selection
2. GCloud IoT Core Configuration
3. Dump previous configuration from flash
4. Exit
Please Enter Your Choice:>1
Network Interface Selection:
1. Ethernet
2. Wi-Fi
3. Cellular
4. Exit
Please Enter Your Choice:>
```

Figure 23. Network Interface Selection Menu

(1) Ethernet Network Interface Configuration

From the **Network Interface Selection** menu, press **1** key to select the Ethernet Network Configuration.



Figure 24. Ethernet Network Interface Configuration Menu

You see the submenu where you choose the IP Address Configuration mode from the available options (DHCP/Static). Choose the **IP Address Configuration** mode. The selected Ethernet configuration setting is stored in internal flash; it is used at a later stage, when communication is initialized.

Note: The Ethernet Static IP configuration does not work with the default project. User needs to add the NetX Duo source code to the project. This issue will be fixed in future releases.

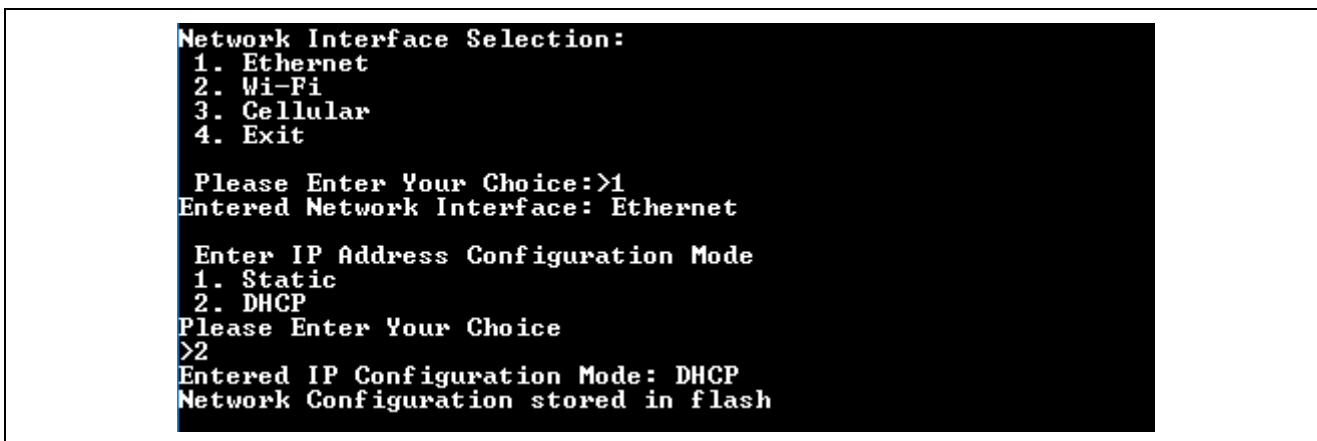


Figure 25. Ethernet Network Interface Configuration Menu – DHCP Configuration

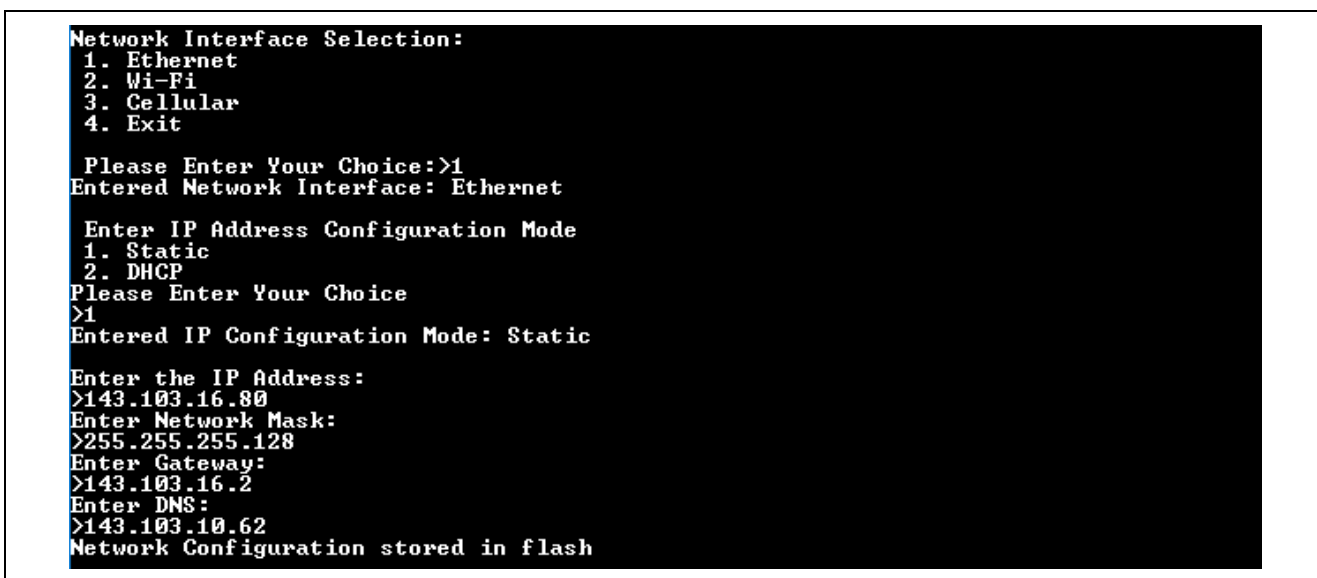


Figure 26. Ethernet Network Interface Configuration Menu – Static IP Configuration

(2) Wi-Fi Network Interface Configuration

From the **Network Interface Selection** menu, press **2** to select the Wi-Fi Network Configuration.

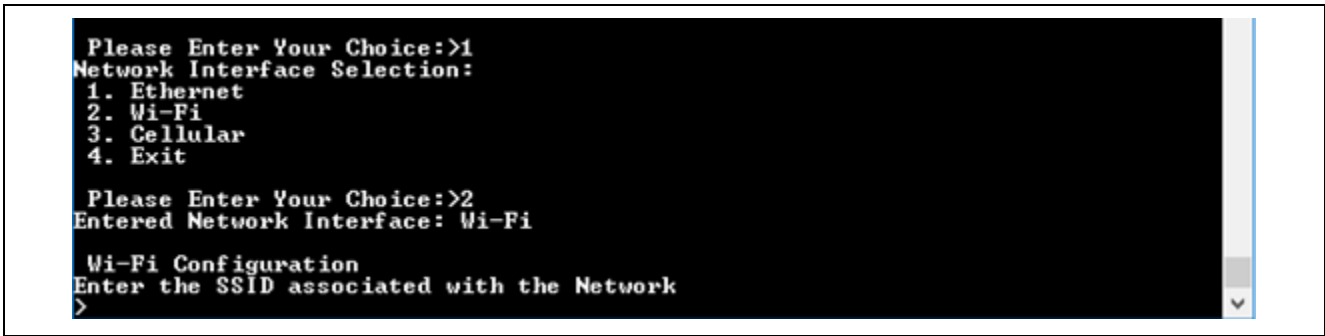


Figure 27. Wi-Fi Network Interface Configuration Menu

You are given the option to enter Wi-Fi Configuration settings, such as **SSID**, **Pass key**, **Security type**, and **IP Address Configuration mode**.

The selected Wi-Fi configuration setting is stored in the internal flash to be used at a later stage, when the communication is initialized.

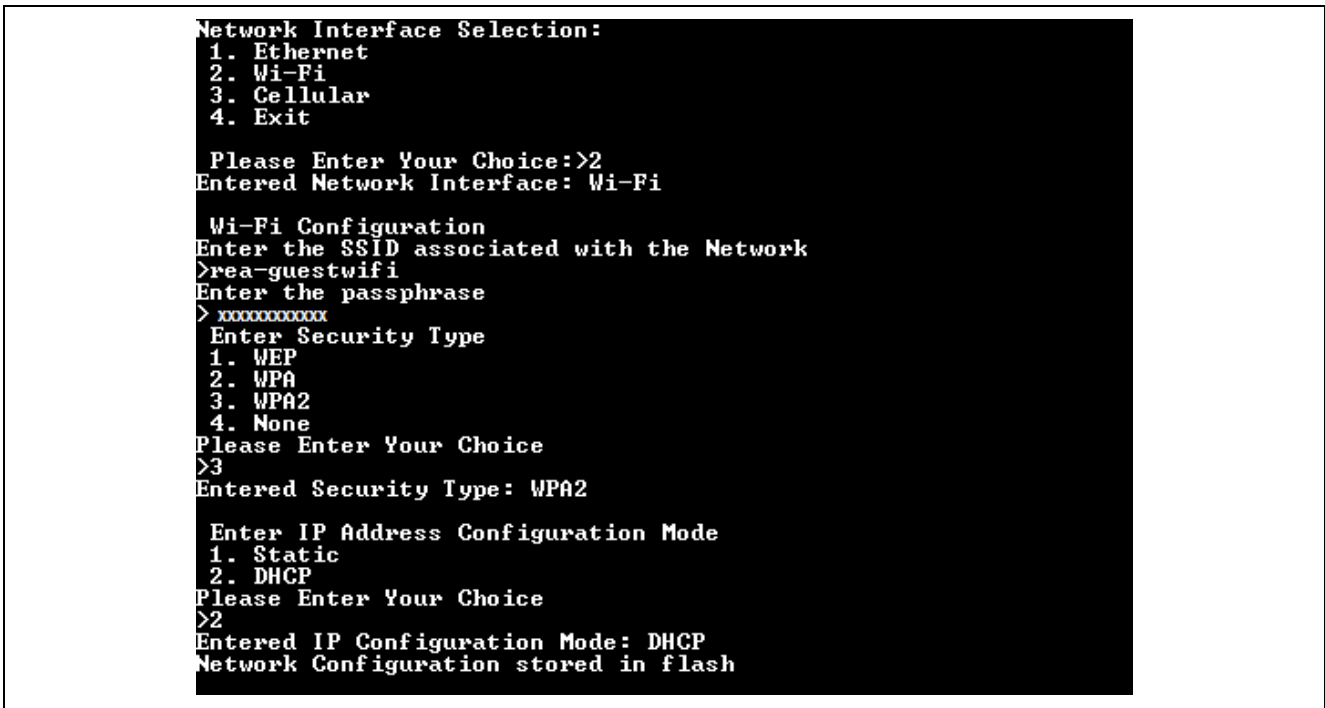


Figure 28. Wi-Fi Configuration – DHCP Configuration Mode

```

Network Interface Selection:
 1. Ethernet
 2. Wi-Fi
 3. Cellular
 4. Exit

Please Enter Your Choice:>2
Entered Network Interface: Wi-Fi

Wi-Fi Configuration
Enter the SSID associated with the Network
>rea-guestwifi
Enter the passphrase
>XXXXXXXXXX
Enter Security Type
 1. WEP
 2. WPA
 3. WPA2
 4. None
Please Enter Your Choice
>3
Entered Security Type: WPA2

Enter IP Address Configuration Mode
 1. Static
 2. DHCP
Please Enter Your Choice
>1
Entered IP Configuration Mode: Static

Enter the IP Address:
>192.168.1.62
Enter Network Mask:
>255.255.255.0
Enter Gateway:
>192.168.1.254
Enter DNS:
> 4.2.2.4
Network Configuration stored in flash

```

Figure 29. Wi-Fi Configuration – Static IP Configuration

(3) Cellular Network Interface Configuration

When the user wants to choose Cellular as the interface, from the **Network Interface Selection** menu, press **3** to select the Cellular Network Configuration.

You will be given two choices as shown in the following figure:

- Option 1: Enter **1** in case of SIM provisioning. In this case, it is assumed that you already pre-configured the SIM card. You can only enter the APN, context ID, and PDP type of the SIM.
Note: If option 1 is used, then option 2 (Start SIM Configuration) can be skipped.
- Option 2: Enter **2** in case of SIM configuration. This option is ideal for users who wish to configure the SIM card using the AT shell interface. For example, setting the Scan Mode, IoT OpMode and so forth, on the SIM.

Note: After cellular connects to the cloud using Option 1, user cannot return to the at shell configuration window using Option 2 anymore.

```

Please Enter Your Choice:>1
Network Interface Selection:
 1. Ethernet
 2. Wi-Fi
 3. Cellular
 4. Exit

Please Enter Your Choice:>3
Entered Network Interface: Cellular

##### Cellular Modem Config Menu #####

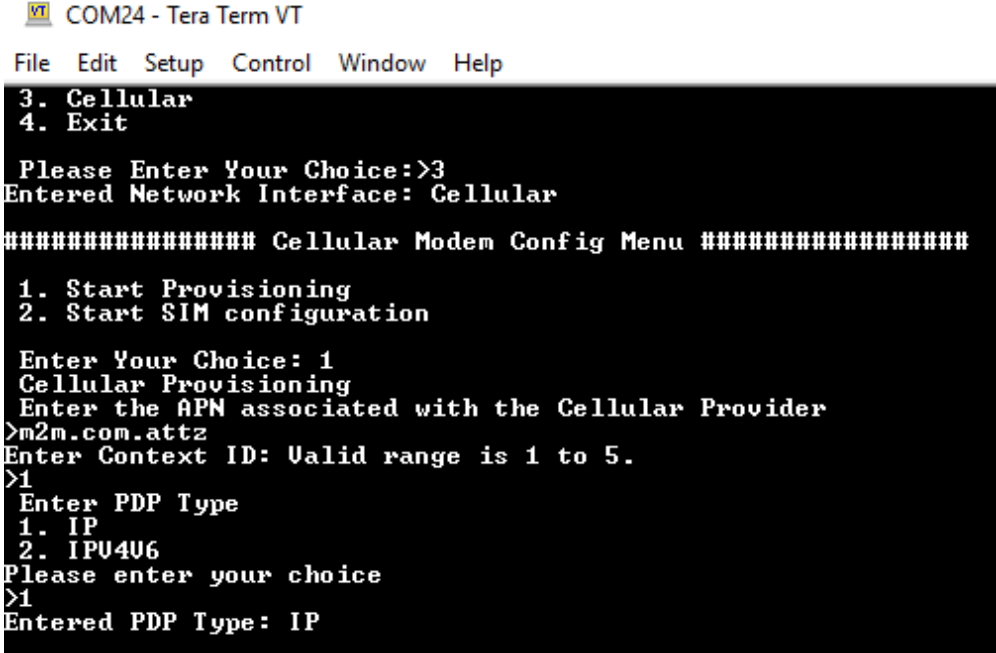
 1. Start Provisioning
 2. Start SIM configuration

```

Figure 30. Cellular Configuration

(a) Start Provisioning Option

In the **cellular modem configuration menu**, choose option 1 to enter the Start provisioning sub-menu as shown in the following screen shot.



```
COM24 - Tera Term VT
File Edit Setup Control Window Help
3. Cellular
4. Exit
Please Enter Your Choice:>3
Entered Network Interface: Cellular
##### Cellular Modem Config Menu #####
1. Start Provisioning
2. Start SIM configuration
Enter Your Choice: 1
Cellular Provisioning
Enter the APN associated with the Cellular Provider
>m2m.com.attz
Enter Context ID: Valid range is 1 to 5.
>1
Enter PDP Type
1. IP
2. IPV4V6
Please enter your choice
>1
Entered PDP Type: IP
```

Figure 31. Cellular Modem Provisioning Menu

The screen shot shows the Cellular configuration settings used for AT&T (USA Carrier) SIM card.

Note: The cellular settings may differ depending upon the SIM card and service provider.

You are given the option to enter Cellular Configuration settings, such as **APN**, **Context ID**, **PDP Type**.

Note: The **APN** name, **Context ID**, and **PDP Type** are provided by the Cellular Service provider. If you already know this information, start entering them using CLI.

The selected Cellular configuration setting is stored in the internal flash to be used at a later stage, when the communication is initialized.

Note: If you have configured your SIM and provisioned the Cellular Modem, you can skip section (b).

(b) Start SIM Configuration Option

Note: User uses this menu item to exercise configuration of a new SIM card and identify the proper settings. This menu item cannot connect the device to the cloud even if `demo start` is issued. After finishing SIM card configuration using this menu item, user needs to go back to the main menu and choose option 1, **Start Provisioning** to connect the device to cloud.

Once the provisioning information is stored, the CLI goes back to the Main Menu. Choose **Network Interface Selection > Cellular**. In the **Cellular Modem Configuration Menu**, choose option 2 to enter the **Start SIM configuration** submenu shown in the following figure.

Note: It will take a few seconds to enter the **Cellular Configuration Menu** since the firmware is opening the cellular framework instance in the background.

```
Please Enter Your Choice:>1
Network Interface Selection:
1. Ethernet
2. Wi-Fi
3. Cellular
4. Exit

Please Enter Your Choice:>3
Entered Network Interface: Cellular

##### Cellular Modem Config Menu #####

1. Start Provisioning
2. Start SIM configuration

Enter Your Choice: 2
Opening Cellular module instance....done

##### Cellular Configuration Menu #####
1. Manual Config using AT cmd shell
2. Auto Config from Pre-stored AT cmd list

Enter your choice: █
```

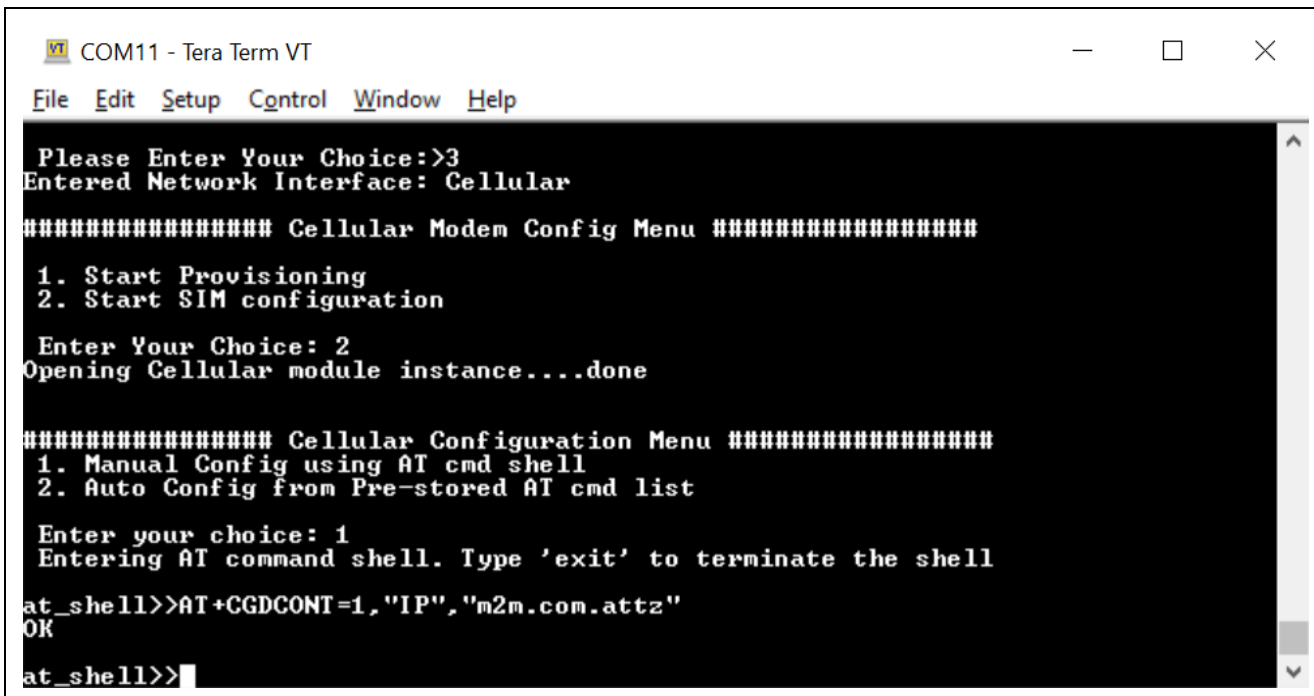
Figure 32. Cellular Configuration Menu

In this mode, option 1 opens the AT command shell mode where you can enter individual AT commands and verify them. You can also store the required AT commands in flash, based on the sequence needed for the Cellular service provider. Option 2 retrieves the pre-stored AT command list.

Manual Configuration using AT Command Shell

In case of option 1, enter the AT command shell shown as follows. You can experiment with various AT commands to configure the SIM cards.

See the following knowledge base article provided by Renesas as a baseline for provisioning the SIM card using the BG96 Cellular modem: <https://en.na4.teamsupport.com/knowledgeBase/18027787>.



```
COM11 - Tera Term VT
File Edit Setup Control Window Help

Please Enter Your Choice:>3
Entered Network Interface: Cellular

##### Cellular Modem Config Menu #####

1. Start Provisioning
2. Start SIM configuration

Enter Your Choice: 2
Opening Cellular module instance....done

##### Cellular Configuration Menu #####
1. Manual Config using AT cmd shell
2. Auto Config from Pre-stored AT cmd list

Enter your choice: 1
Entering AT command shell. Type 'exit' to terminate the shell

at_shell>>AT+CGDCONT=1,"IP", "m2m.com.attz"
OK
at_shell>>█
```

Figure 33. AT Command Shell

To exit the AT command shell, enter the command **exit** or **EXIT**. You will be asked whether you want to save the AT command as shown in the following figure.

```

COM24 - Tera Term VT
File Edit Setup Control Window Help
1. Ethernet
2. Wi-Fi
3. Cellular
4. Exit

Please Enter Your Choice:>3
Entered Network Interface: Cellular

##### Cellular Modem Config Menu #####

1. Start Provisioning
2. Start SIM configuration

Enter Your Choice: 2

##### Cellular Configuration Menu #####
1. Manual Config using AT cmd shell
2. Auto Config from Pre-stored AT cmd list

Enter your choice: 1
Entering AT command shell. Type 'exit' to terminate the shell
at_shell>>exit
Do you wish to store the AT commands for your carrier? [Y/N]: █
    
```

Figure 34. Exiting the AT Command Shell

If you choose to save the AT commands, which can be used later to auto configure the new SIM cards, enter 'Y'. In that case, you will be asked to enter the AT command details as shown in the following figure.

Note: Only the commands you entered after you choose 'Y' for the above query will be saved. See following example.

```

at_shell>>exit
Do you wish to store the AT commands for your carrier? [Y/N]: Y
**** Start Inserting AT Commands. Type exit to terminate!!! ****

AT Command: at+cgdcont=1,"IP","m2m.com.attz"
Response <case sensitive>: OK
Response Wait time in MilliSeconds: 1000
Retry Count: 5
Retry Delay in milli-seconds : 100

#####
AT Command: at+cgdcont=1,"IP","m2m.com.attz"
Response string: OK
Response Wait time: 1000
Retry Count: 5
Retry Delay: 100

#####
Do you Want to save this AT Command ? [y/n]: Y
AT Command: exit█
    
```

Figure 35. Saving the AT Commands for Later Use

Auto Configuration from Pre-stored AT Command List

From the Cellular Configuration menu, choose option **2** to enter **Auto configuration from pre-stored AT command** list menu as shown in the following figure.

```
##### Cellular Modem Config Menu #####
1. Start Provisioning
2. Start SIM configuration

Enter Your Choice: 2
Opening Cellular module instance....done

##### Cellular Configuration Menu #####
1. Manual Config using AT cmd shell
2. Auto Config from Pre-stored AT cmd list

Enter your choice: 2

#####

Command: at+cgdcont=1,"IP","m2m.com.attz"

Resp:
OK
```

Figure 36. Auto Configuration from Pre-stored AT Command List

The pre-stored AT commands will be sent to the cellular modem and their responses will be displayed in the console window.

Note: In case of repeated failures to register to the network, increase the AT command retry count and set the appropriate network scan sequence in the Synergy Configurator of the project, generate, and rebuild the project.

4.4.1.2 Google IoT Core Configuration

At this stage, it is assumed that you followed the instructions mentioned in section 3.3 to create a device in Google Cloud Platform. If not, complete the steps mentioned in section 3.3 before proceeding.

From the **Main Menu**, press **2** and the **Enter** key to configure the Google Cloud IoT Core service as shown in the following figure.

```
VT Tera Term - [disconnected] VT
File Edit Setup Control Window Help

##### Main Menu #####
1. Network Interface Selection
2. GCloud IoT Core Configuration
3. Dump previous configuration from flash
4. Exit

Please Enter Your Choice:>2
1. Google IoT Core Setting Menu
2. Device Certificate/Keys Setting Menu
3. Exit

Please Enter Your Choice:>1
```

Figure 37. Google IoT Core Configuration Menu

(1) Google IoT Core Setting Menu

From the **Google IoT Core configuration menu**, press **1** and the **Enter** key to configure the Google IoT Core settings. In the Google IoT Core Configuration menu, you have the option to enter the following information as listed in the following window.

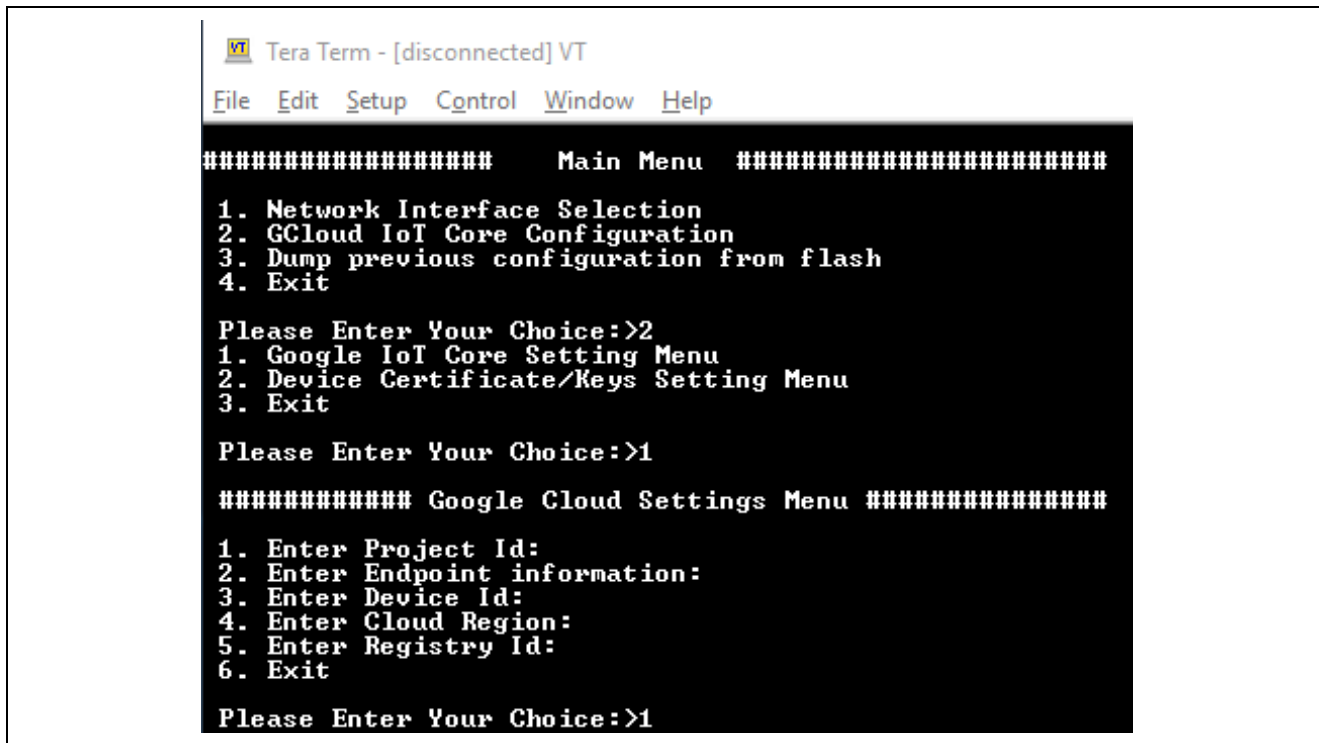


Figure 38. Google Cloud Settings Menu

Note: In case of option 2 (Enter Endpoint Information), enter **mqtt.googleapis.com**.

Note: The Project ID (created in section 3.3.1.1), Device ID Registry ID (Created in section 3.3.1.3) and Cloud region (section 3.3.1.2) info are noted as part of the Gcloud settings.

(2) Device Certificate/Keys Setting Menu

From the **Google IoT Core configuration menu**, press **2** and the **Enter** key to configure the Device Certificate/Keys settings.

In the **Device Certificate/Keys settings** menu, you have the option of entering the root CA, thing certificates, and thing private key in **.pem** format.

Open these certificates in a text editor, then copy and paste them in the serial console. Press **Enter**.

The root CA certificate (*gcloud_rootCA.pem*) for Google Cloud is enclosed as part of the package.

Thing certificate is the *rsa_devcert.pem* file created in section 3.3.2.

Thing Private Key is the *rsa_private_pkcsi.pem* file generated in section 3.3.2.

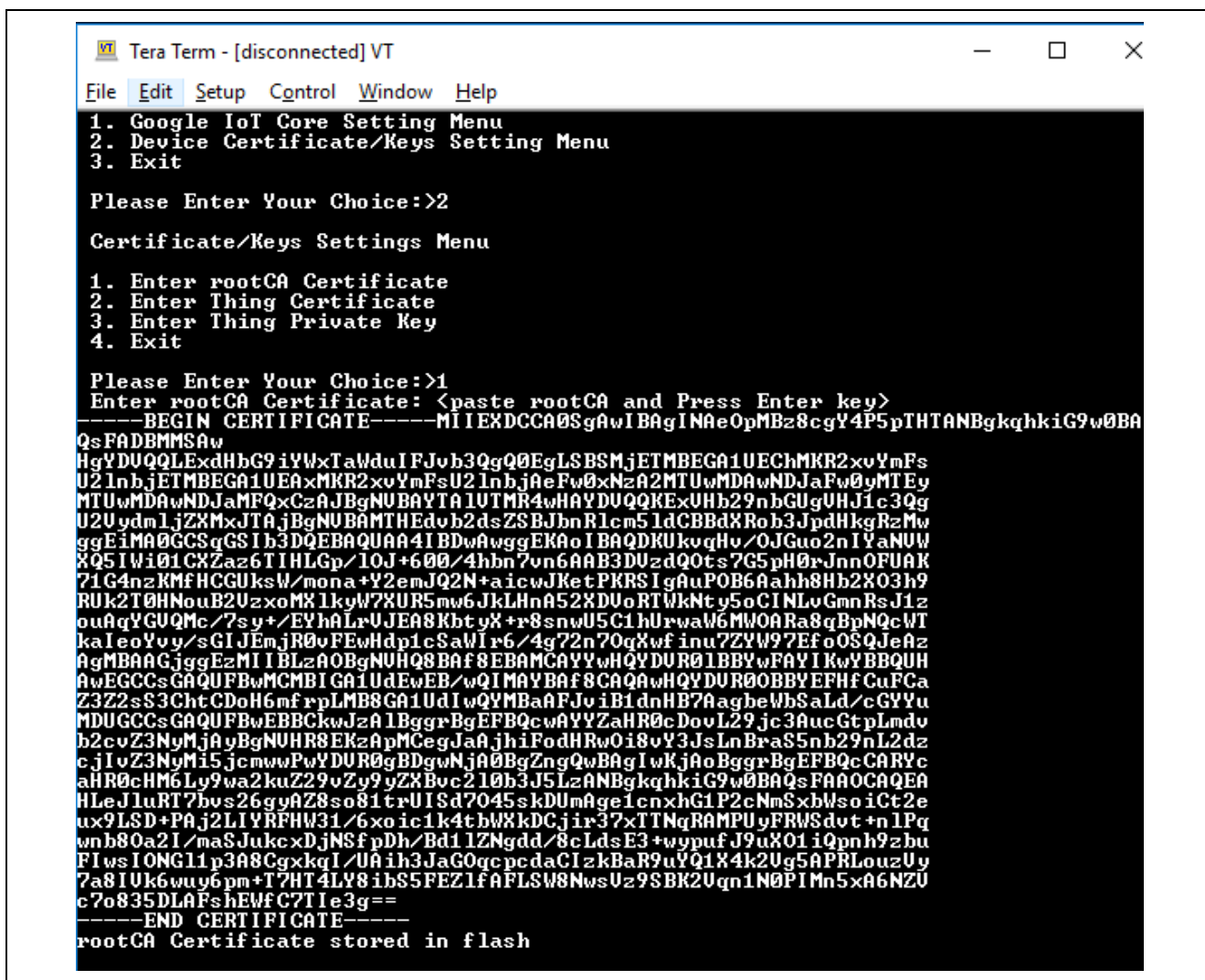


Figure 39. Certificates/Keys Setting Menu

The selected configuration setting is stored in the internal flash; it is used at a later stage during the Google IoT Core connection.

4.4.2 Dump the Previous Configuration

From the **Main Menu**, choose **Synergy MQTT TLS Cloud Connectivity App Notes 3** to display the pre-selected network, the Google Cloud IoT core Service Configuration options you made from the internal flash, as shown in the following figure.

Note: Make sure that all the Cloud information is present in the dump from the flash. Any empty space in the Cloud information indicates that the Cloud information is stored incorrectly.

Note: When the project is flashed onto the board the first-time, since there is no information stored in flash, dumping the data using the `cwiz` command may result in garbage being displayed on the CLI. Power cycle the board and configure before proceeding to dump the data.

```

Network Configuration stored in flash
##### Main Menu #####
1. Network Interface Selection
2. GCloud IoT Core Configuration
3. Dump previous configuration from flash
4. Exit

Please Enter Your Choice:>3

##### Flash Dump Start#####

Network Interface selected: WiFi
IP Mode: DHCP

WiFi Configuration
SSID      : visitor
Key       : Renesas123@
Security  : WPA2

GCloud Endpoint: mqtt.googleapis.com
GCloud Project ID: test4-syn
GCloud Device ID: test4-device0
GCloud Cloud Region: us-central1
GCloud Registry Id: test4-registry
rootCA len: 1122
deviceCert len: 795
devicePrivKey len: 1194

##### Flash Dump End #####

##### Main Menu #####

1. Network Interface Selection
2. GCloud IoT Core Configuration
3. Dump previous configuration from flash
4. Exit

Please Enter Your Choice:>
    
```

Figure 40. Dump Configuration Menu

4.4.3 Demo Start/Stop Command

From the CLI console, enter `demo start` command to start the Synergy Cloud Connectivity Application Demonstration.

```

>?
Help Menu
cwiz : Network/Cloud Configuration Menu
Usage: cwiz

demo : Start/Stop Synergy Cloud Connectivity Demo
Usage: demo <start>/<stop>
    
```

Figure 41. Help Menu

The application framework reads your pre-configured selection options for the network interface, the IoT Service from the internal flash, and checks for its validity. If the content is valid, it then initializes the network interface and establishes a MQTT connection with the Google IoT Core.

This application wakes up periodically (every 5 seconds) and checks for your input event flag state. The flag state is set once you have entered the `demo start/stop` command on the CLI. This application performs the following functions periodically until you enter the `demo stop` command:

1. Initializes communication interface (Ethernet, Wi-Fi, or Cellular).
2. Initializes IoT Cloud interface.
3. Reads sensor data and publishes them periodically on MQTT topics.
4. Updates your LED state based on the type of MQTT message received.

If the `demo stop` command is issued, it de-initializes the IoT Cloud interface, which in turn stops MQTT messages from publishing and clears any pending MQTT messages from its internal queue.

Note: Once the demo starts running, that is `demo start` command is issued, `cwiz` command should not be used until the demo is stopped using `demo stop` command.

Note: With this release of the application project, swapping Ethernet and Wi-Fi interface after `demo stop` is not supported.

4.5 Verifying the Demo

Use the following instructions to verify the functionality of this Synergy Cloud Connectivity Application Project.

4.5.1 Running the Synergy Cloud Connectivity Demonstration

Run this application demonstration using the `demo start` command from the serial console.

Once you run `demo start`, it begins to configure the network interface, establishes a connection with Google Cloud IoT Core, and starts publishing sensor data periodically (every 5 seconds).

4.5.2 Monitoring MQTT Messages on Google Cloud Platform

Once the demo is running, sensor data is periodically published to the Google Cloud IoT Core. To view the data published, open the Google Cloud shell by clicking **Activate Cloud shell** button as highlighted in the following graphic.

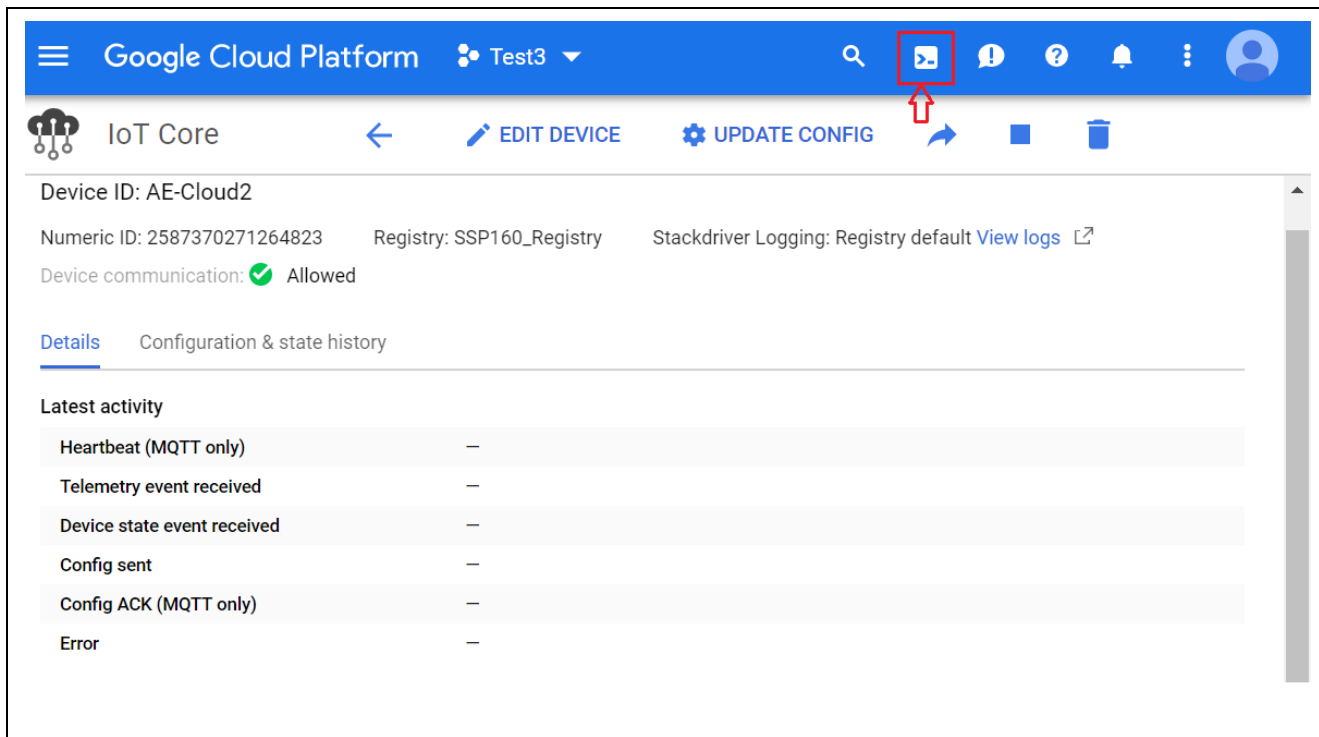


Figure 42. Activating Cloud Shell Button

Now, find out the subscription name created for your topic. It can be found in the **Pub/Subpage** under **Subscriptions** as shown in the following graphic.

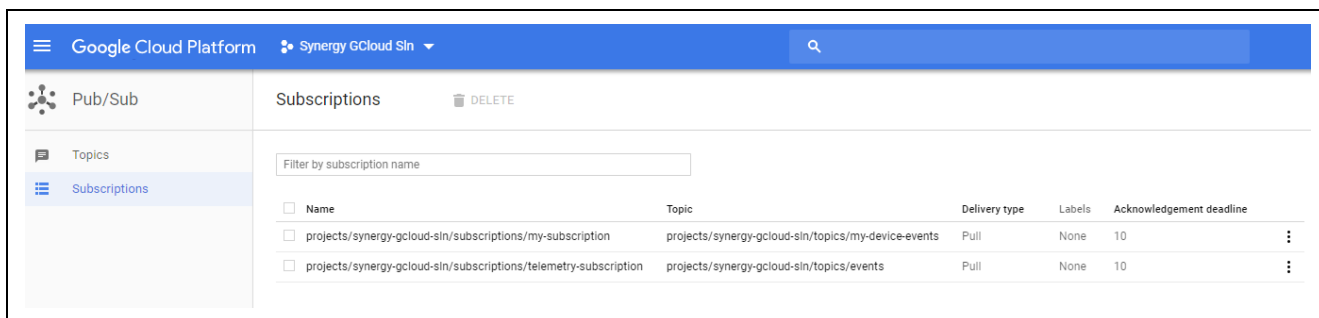


Figure 43. Subscription Name

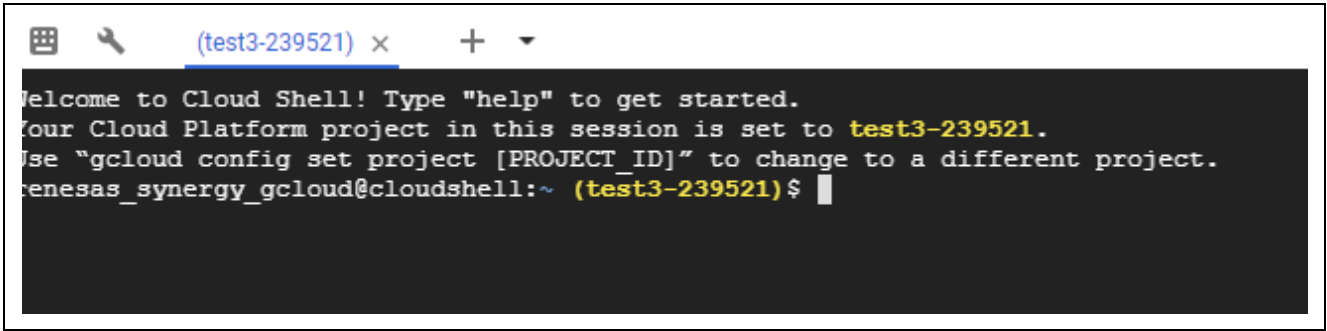


Figure 44. Cloud Shell

Once the shell is activated, enter the following commands to pull the published telemetry data to the MQTT topic.

```
gcloud pubsub subscriptions pull --limit 100 --auto-ack <subscription name of your publish topic>
```

Note: The messages published are in FIFO order. To view the latest message, the buffer must be flushed. The number 100 in the above command can be increased to publish more messages.

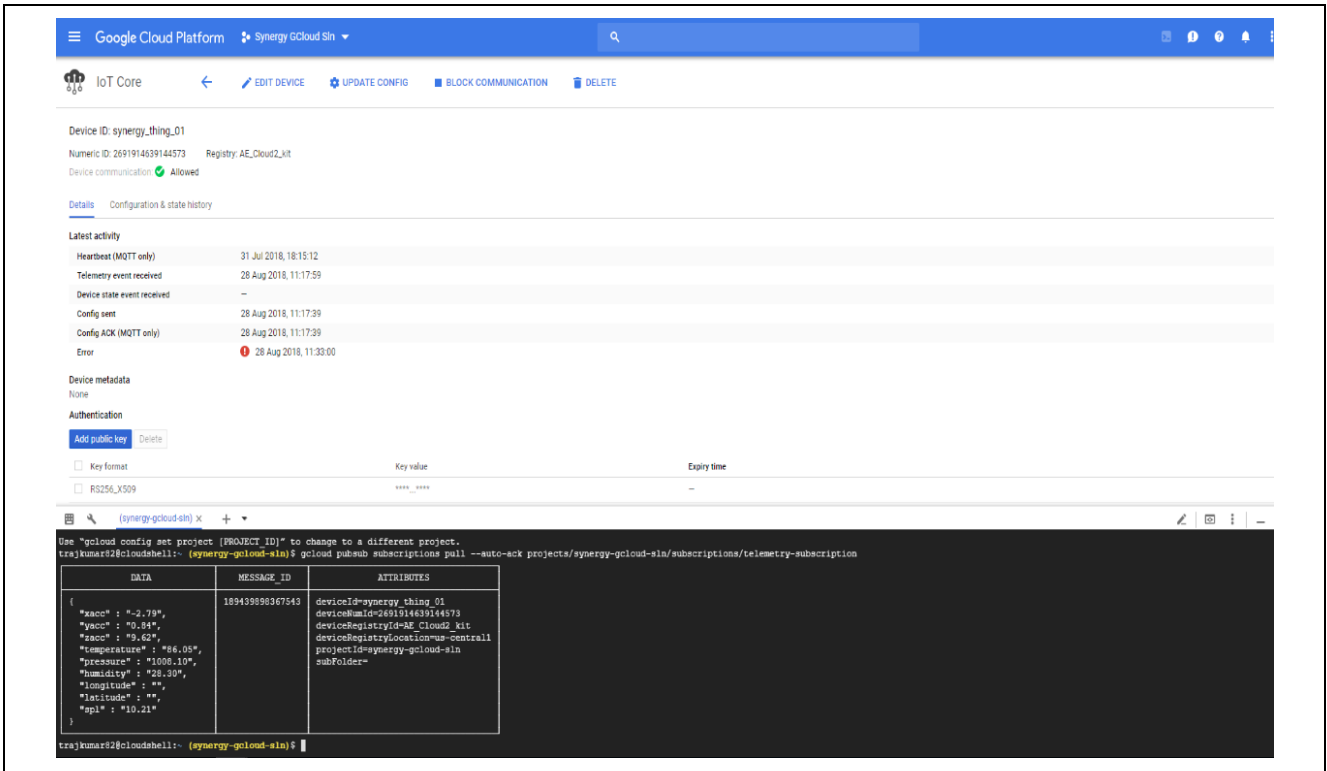


Figure 45. Viewing Published Messages

4.5.3 Publishing the MQTT Message from Google Cloud Platform

You can publish the MQTT message to turn your LEDs ON/OFF on the AE-CLOUD1/AE-CLOUD2 kit by using messages from the following table. These messages indicate the states for red, green, and yellow LEDs.

Note: The Messages under Message column are **case sensitive**, users need to take care of this while using them to turn the LEDs ON/OFF.

Note: Only one message can be entered at a time. Copy the message as-is and do not include any extra spaces.

Table 2. Turning the User LED ON/OFF on your AE-CLOUD1/AE-CLOUD2 Kit

LED State	Message
RED LED ON	<code>{"state":{"desired":{"Red_LED":"ON"}}</code>
RED LED OFF	<code>{"state":{"desired":{"Red_LED":"OFF"}}</code>
YELLOW LED ON	<code>{"state":{"desired":{"Yellow_LED":"ON"}}</code>
YELLOW LED OFF	<code>{"state":{"desired":{"Yellow_LED":"OFF"}}</code>
GREEN LED ON	<code>{"state":{"desired":{"Green_LED":"ON"}}</code>
GREEN LED OFF	<code>{"state":{"desired":{"Green_LED":"OFF"}}</code>

To publish the message, use the following steps:

1. Go to the **IoT Core** page.
2. The registry you created will be listed in the **IoT Core** page. Locate the registry from the list and enter the corresponding registry details page.
3. The device added under this registry will be listed. Locate the device from the list and enter the corresponding device page.
4. Click the **UPDATE CONFIG** button in the device page as shown in the following figure.

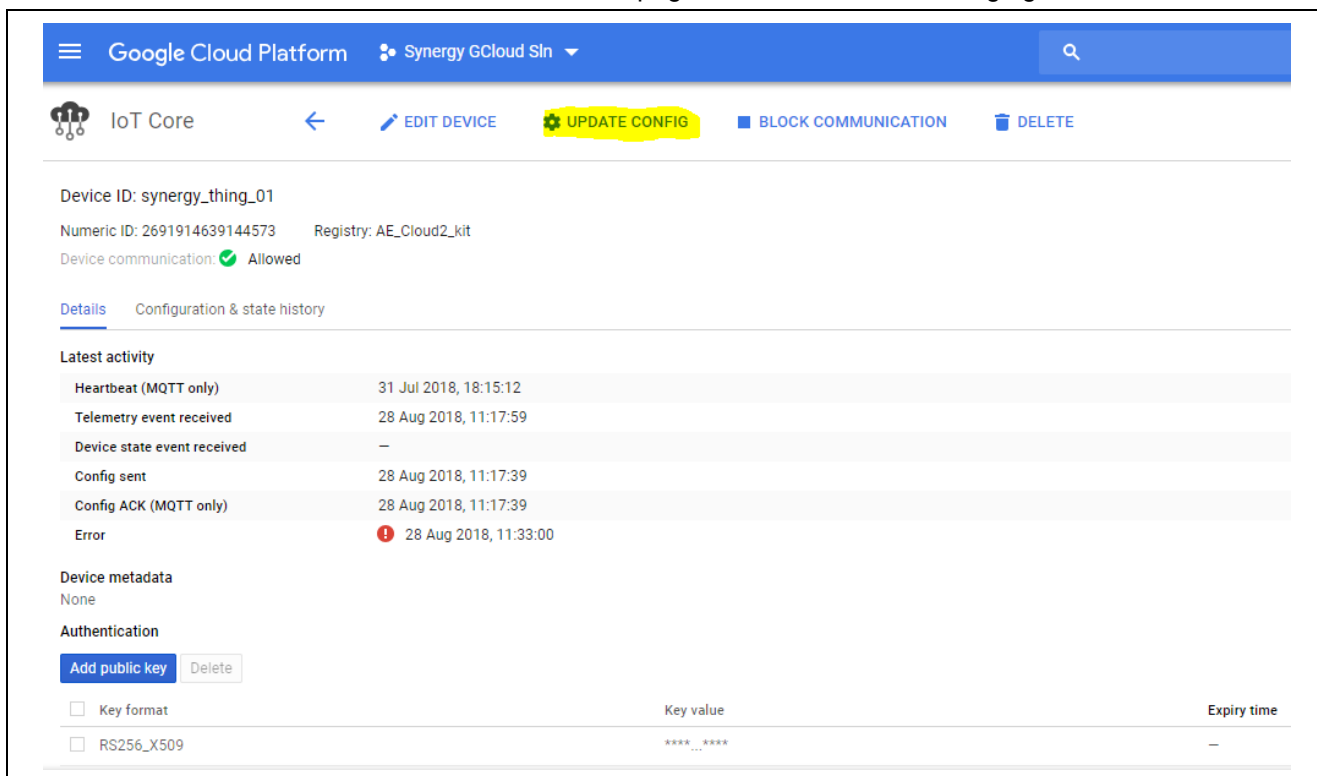


Figure 46. Updating the Device Configuration

A pop-up window appears as shown in the following figure. Choose the **Message format** as Text and paste the message that needs to be send to the MQTT device.

- Click the **SEND TO DEVICE** button to send the message to the MQTT device. On the AE-CLOUD2/AE-CLOUD1 kit, once the message is received, the corresponding user LED toggles.

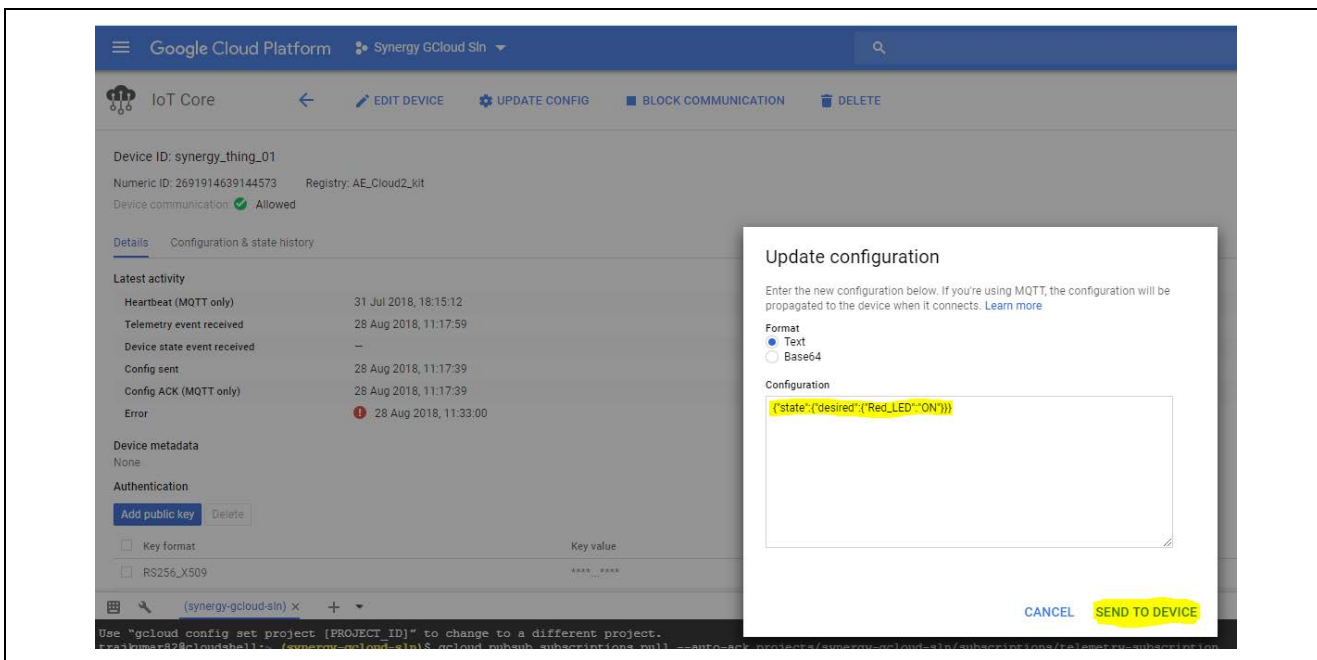


Figure 47. Sending the Message to the Device

4.5.4 Stopping the Synergy Cloud Connectivity Demonstration

To stop the demonstration, enter the `demo stop` command. Issuing this command de-initializes the IoT Cloud interface, stops it from publishing MQTT messages, and clears any pending MQTT messages from its internal queue. The demo can be restarted by typing `demo start` command.



Figure 48. Application Demo Stop Sequence

4.6 Customizing the Demo Delays

This application supports failure recovery for a period specified by certain macros. To increase or reduce the time for recovery, modify the following in the source code (`MQTT_Config.h`):

Table 3. Delay settings

Macro	Purpose	Set to
MQTT_UPDATE_DELAY	Delay between the updates pushed to the cloud.	5 seconds
IOT_NW_RETRY_DELAY	Retry for Network failures	5 seconds
NETWORK_RETRY_CNT	Retry count for network connectivity in case of network failures.	10 seconds
IOT_SERVICE_RETRY_DELAY	Delay between retries for IoT Service connectivity in case of failures.	Currently set to 100
IOT_SERVICE_RETRY_CNT	Retry count for IoT Service.	5 seconds

5. Next Steps

Visit www.renesas.com/synergy/tools to learn more about development tools and utilities.

Visit www.renesas.com/synergy/gallery to download development tools and utilities.

Renesas Synergy Module Guides collateral link: www.renesas.com/synergy/applicationprojects

6. MQTT/TLS Reference

- SSP 1.6.0 User's Manual can be downloaded from the Renesas Synergy™ Gallery (www.renesas.com/synergy/ssp)
- Google Cloud IoT Core documentation (<https://cloud.google.com/iot/docs/?authuser=0>)

7. Known Issues and Limitations

1. If a user running this demo using Ethernet connection in some corporate network tries to stop the demo using `demo stop` command and then restarts the demo using `demo start` command, the demo fails to reconnect to the Google Cloud IoT MQTT broker.
2. Occasional outages in Cloud connectivity can be noticed during the demo due to changes in the Cloud server. Please contact Renesas support team for questions.

Website and Support

Visit the following vanity URLs to learn about key elements of the Synergy Platform, download components and related documentation, and get support.

Synergy Software	www.renesas.com/synergy/software
Synergy Software Package	www.renesas.com/synergy/ssp
Software add-ons	www.renesas.com/synergy/addons
Software glossary	www.renesas.com/synergy/softwareglossary
Development tools	www.renesas.com/synergy/tools
Synergy Hardware	www.renesas.com/synergy/hardware
Microcontrollers	www.renesas.com/synergy/mcus
MCU glossary	www.renesas.com/synergy/mcuglossary
Parametric search	www.renesas.com/synergy/parametric
Kits	www.renesas.com/synergy/kits
Synergy Solutions Gallery	www.renesas.com/synergy/solutionsgallery
Partner projects	www.renesas.com/synergy/partnerprojects
Application projects	www.renesas.com/synergy/applicationprojects
Self-service support resources:	
Documentation	www.renesas.com/synergy/docs
Knowledgebase	www.renesas.com/synergy/knowledgebase
Forums	www.renesas.com/synergy/forum
Training	www.renesas.com/synergy/training
Videos	www.renesas.com/synergy/videos
Chat and web ticket	www.renesas.com/synergy/resourcelibrary

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Sep.24.18	-	Initial version
1.01	Feb.04.19	-	Updated links and section 4, Running the MQTT/TLS Application.
1.02	Apr.03.19	-	Fixed Cellular at command shell bug.
1.03	Jun.27.19	-	Updated for latest tools
1.04	Oct.22.19	-	Updated for latest tools
1.05	Nov.01.19	-	Updated section 4.3, step 4

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.
 - "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
 - "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user's manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user's manuals, application notes, "General Notes for Handling and Using Semiconductor Devices" in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.