Renesas RA FSP™ Platform

RA FSP MQTT/TLS Azure Cloud Connectivity Solution

Introduction

This application note describes IoT Cloud connectivity solutions in general and introduces you briefly to the IoT Cloud provider Microsoft Azure. It covers the RA FSP MQTT/TLS module along with the Azure IoT SDK for embedded C.

This application project is built with the integrated “Azure IoT SDK for Embedded C” package which allows small embedded (IoT) devices like Renesas RA family of MCUs RA6M3/RAM6M4/RAM6M5 to communicate with Azure services.

The application example uses Azure IoT DPS (Device Provisioning Service) to provision, register the IoT device, and send and receive data to and from the development kit.

This application note enables you to effectively use the RA FSP modules in your own design with the Azure IoT SDK. Upon completion of this guide, you will be able to add the FSP modules to your own design, configure it correctly with Azure IoT SDK 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 sample code that demonstrates advanced usage of FSP modules are available in the *RA FSP Software Package (FSP) User’s Manual* (see Next Steps section) and serve as valuable resources in creating more complex designs. Explaining the underlying operation of Azure IoT SDK for Embedded C is beyond the scope of this document. Users should refer to the documentation from Microsoft for education on topics related to Azure IoT SDK section: https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-sdks

In this release, the EK-RA6M5 kit is used for the application project.

Required Resources

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

Development Tools and Software

- e^2 studio version: v2022-04 or later
- RA FSP Software Package (FSP) v3.7.0 or later
- SEGGER J-Link® RTT viewer version: 7.54 or later
- Azure IoT explorer 0.14.5.0 or later (PC tool for validating the Cloud side)
- Azure CLI 2.24 or later (Azure command-line interface is a set of commands used to create and manage Azure resources)
- Access to Azure Cloud Connectivity Portal (https://portal.azure.com/#home) to create IoT Devices (If you are new to Azure IoT)

Hardware

- Renesas RA FSP™ EK-RA6M5 kit
- PC running Windows® 10, 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 (CAT5/6)
- Router with ethernet port or ethernet switch to connecting to the router for Internet connectivity

Prerequisites and Intended Audience

This application note assumes that you have some experience with the Renesas e^2 studio ISDE and RA FSP Software Package (FSP). Before you perform the procedures in this application note, follow the procedure in the *FSP User Manual* to build and run the Blinky project. Doing so enables you to become familiar with the e^2 studio and the FSP, and also validates that the debug connection to your board functions properly. In addition, this application note assumes you have some knowledge of MQTT/TLS and its communication protocols.

The intended audience are users who want to connect to Azure Cloud using the Azure IoT SDK for Embedded C on the Renesas RA RA6 MCU Series.

Note: If you are a first-time user of e^2 studio and FSP, we highly recommend you install e^2 studio and FSP on your system in order to run the Blinky Project and to get familiar with the e^2 studio and FSP development environment before proceeding to the next sections.

Note: If you are new to Azure Internet of Things, we recommend you get started with Introduction the Azure IoT https://docs.microsoft.com/en-us/azure/iot-fundamentals/iot-introduction

Prerequisites

1. Access to online documentation available in the Cloud Connectivity References section
2. Access to latest documentation for identified Renesas Flexible Software Package
3. Prior knowledge of operating e^2 studio and built-in (or standalone) RA Configurator
4. Access to associated hardware documentation such as User Manuals and Schematics

Using this Application Note

Section 1 of this document covers the General Overview of the Cloud Connectivity, Azure IoT Solution using IoT Central, and Azure DPS, MQTT and TLS Protocols and Device certificates and Keys used in the Cloud Connectivity.

Section 2 covers the modules provided by RA FSP to establish connectivity to Cloud service providers and the features supported by the module.

Section 3 covers the architecture of the reference application project, an overview of the software components included, and step-by-step guidelines for recreation using the FSP configurator.

Section 1.1 covers the setup of the Azure Cloud required for the application.

Sections 4 covers building and running the Application project for quick validation.

Note: We recommend that you operate with your own Microsoft Azure Cloud credentials and use your created Cloud configurations to run the application. The default sample configuration detailed in this project is for reference only and may have access issues to Azure since the application is communicating with a test account.

Note: For a quick validation using the provided application project, you can skip sections 1 to 3 and go to section 4 for instructions on importing, building, and running the application project on the EK board. You are still required to provide necessary user credentials for the application as described in section 1.1.

Contents

1. Introduction to Cloud Connectivity

1.1 Cloud Connectivity Overview

Internet of Things (IoT) is a sprawling set of technologies described as connecting everyday objects, like sensors or smartphones, 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.

The IoT Cloud Connectivity Solution includes the following major components:

1. Devices or Sensors
2. Gateway
3. IoT Cloud services
4. End user application/system

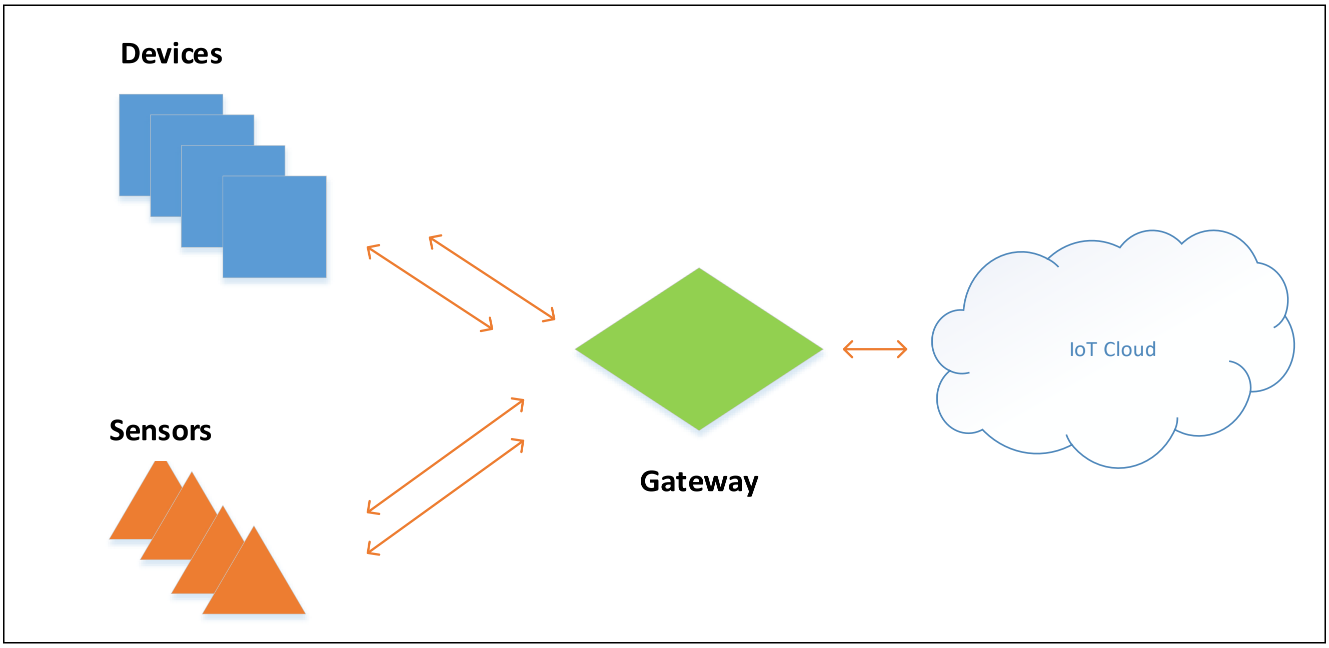


Figure 1. IoT Cloud Connectivity Architecture

Devices or Sensors

A device includes hardware and software that interacts 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 to connect to the IoT Cloud Service provider.

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.

1.2 Microsoft Azure IoT Solution

1.2.1 Overview

Microsoft's end-to-end IoT platform is a complete IoT offering so that enterprises can build and realize value from IoT solutions quickly and efficiently. Azure IoT Central solutions are used with backend support from the Azure IoT Hub Device Provisioning Service.

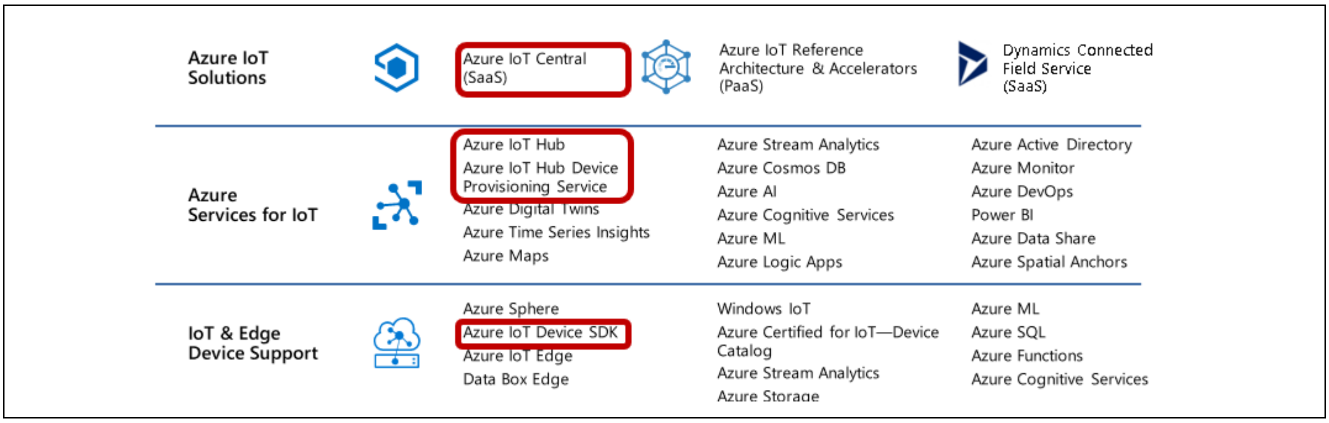


Figure 2. Microsoft Azure IoT Cloud Solution

1.2.2 IoT Hub and Device Provisioning Service

Azure IoT Hub and IoT Hub Device Provisioning Service (DPS)

IoT Hub provides built-in support for the MQTT v3.1.1 protocol. Please review the following webpage for more understanding of the IoT Hub and Device Provisioning Services (DPS).

https://docs.microsoft.com/en-us/azure/iot-dps/

Device Provisioning Service

High-level sequence of events to connect a Device to IoT Hub:

1. After the device is manufactured, the device enrollment information is added to the DPS. This is the only manual step in the process!
2. At some point afterwards, which could be a day or it could be several months, the device goes online and connects to DPS to find its IoT solution home.
3. DPS and the device go through an attestation handshake using the device enrollment info. DPS proves the device's identity.
4. DPS registers the device to IoT hub and populates the initial desired device state.
5. IoT hub returns the connection info for the device.
6. DPS gives the device its IoT Hub connection info.
7. The device now establishes a connection with IoT hub and retrieves its initial configuration from IoT hub and makes any changes/updates, as needed.
8. The device starts sending telemetry to IoT hub.

Embedded C SDK

The Embedded C SDK, the newer addition to the Azure SDKs family, was designed to allow embedded IoT devices to leverage Azure services, like device to cloud telemetry, cloud to device messages, direct methods, device twin, device provisioning, and IoT Plug and play, all while maintaining a minimal footprint.

It allows full control over memory allocation and the flexibility to bring your own MQTT client, TLS, and Socket layers.

Written in C, this version of the SDK is optimized to be used on small and embedded devices with limited capabilities and resources.

The Azure IoT SDK is open source and published on GitHub (https://github.com/Azure/azure-sdk-for-c). This is also distributed with FSP version 3.7.0 and above.

Authentication Methods

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

- **X.509**
  The communication path between devices and Azure IoT Hub, or between gateways and Azure IoT Hub, is secured using the industry-standard Transport Layer Security (TLS) with Azure IoT Hub authenticated using the X.509 standard.
  To protect devices from unsolicited inbound connections, Azure IoT Hub does not open any connection to the device. The device initiates all connections.
  Azure IoT Hub reliably stores messages for devices and waits for the device to connect. These messages are stored for two days, enabling devices connecting sporadically due to power or connectivity concerns to receive these messages. Azure IoT Hub maintains a per-device queue for each device.
  Note: The reference application project attached to this application note doesn't use X509 authentication. Instead it uses the SAS token.
- **Per-Device Key Authentication**
  Figure 3 shows authentication in the IoT Hub using security tokens.

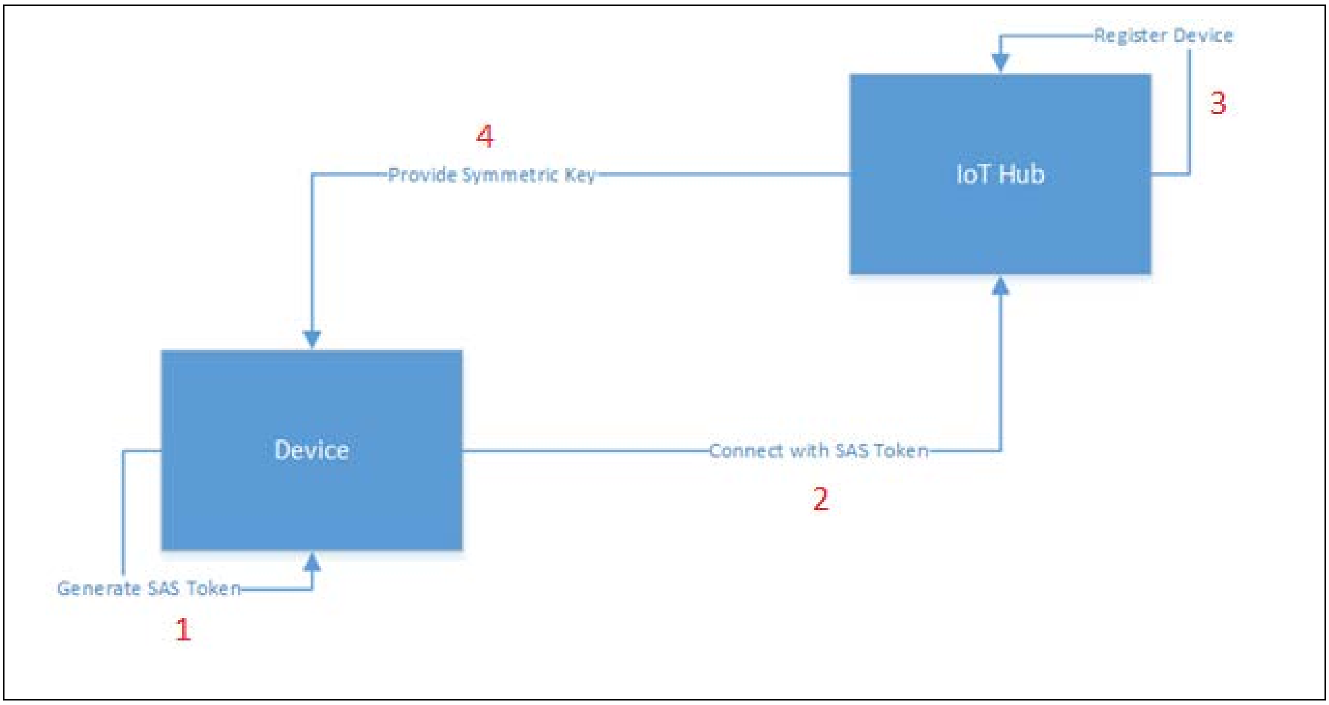


Figure 3. Authentication using Security Tokens

1. The device prepares a shared access signature (SAS) token using the device endpoint, device id, and primary key (generated as part of the device addition to the IoT Hub).
2. When connecting to the IoT Hub, the device presents the SAS token as the password in the MQTT CONNECT message. The username content is the combination of device endpoint and device name along with the additional Azure defined string.
3. The IoT Hub verifies the SAS token and registers the device and connection is established.
4. IoT hub provides Symmetric key for Data encryption.

Note: The connection is closed when the SAS token expires.

1.3 MQTT Protocol Overview

MQTT stands for Message Queuing Telemetry Transport. MQTT is a Client Server publish-subscribe messaging transport protocol. It is an extremely light-weight, 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.

An 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 and 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.

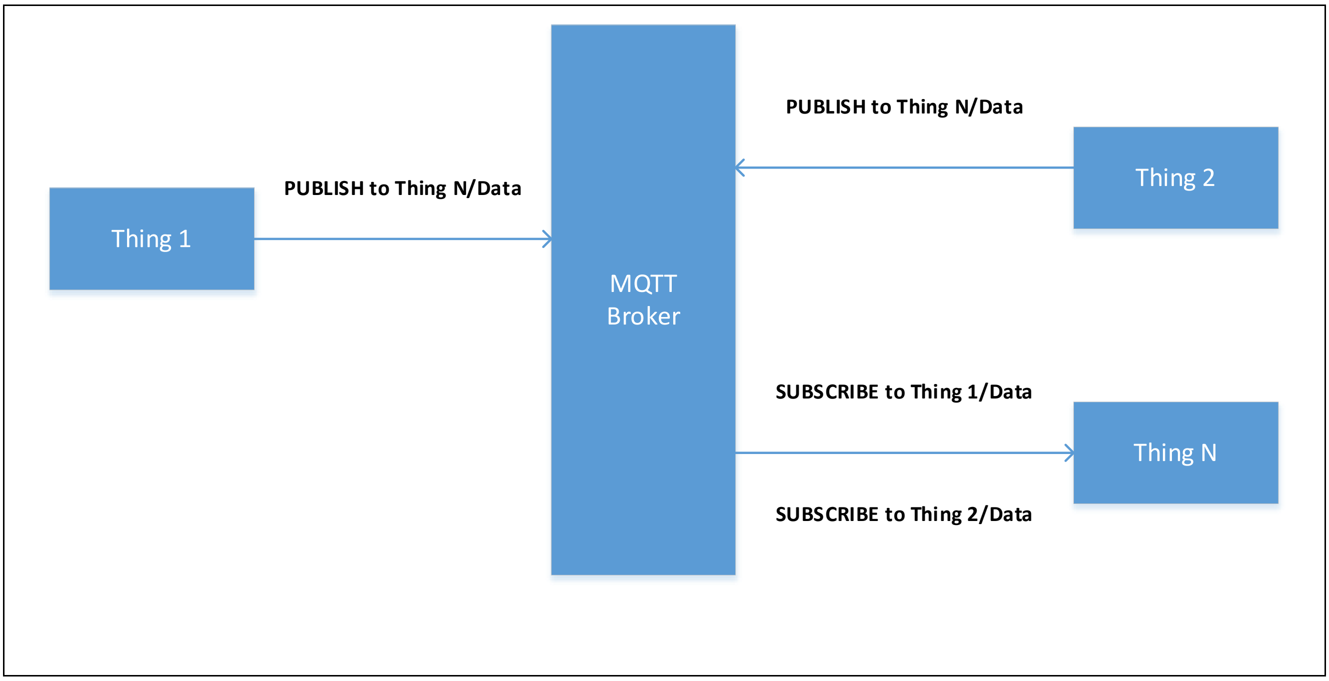


Figure 4. MQTT Client Publish/Subscribe Model

In the Pub/Sub model used by MQTT, 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 guarantees that each message is received only once by the counterpart. It is the safest and the slowest QoS level.

1.4 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 ensures that connection is reliable. A Message Authentication Code (MAC), such as Secure Hash Algorithm (SHA), ensures message integrity.

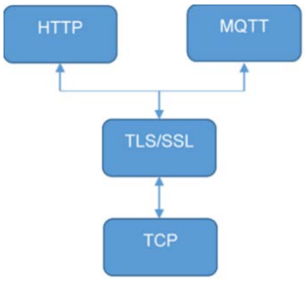


Figure 5. SSL/TLS Hierarchy

1.4.1 Device Certificates and Keys

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

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 TLS hosts can agree on a format to use. X.509 defines a specific format for certificates and various encodings that can be used to produce a digital document. Most X.509 certificates used with TLS are encoded using a variant of ASN.1, which is 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, making 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 base-64 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.

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 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.

1.4.2 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 of the device should be kept secret.
- Use the latest version of TLS (v1.2 or above) when communicating with IoT Cloud and verify that the server certificate is valid using trusted 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 purpose 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 don't have an operating system, ensure that you can securely update the device's software if security vulnerabilities are discovered after deployment.

2. RA FSP MQTT/TLS Cloud Solution

2.1 MQTT Client Module Introduction

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 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 in 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.

2.1.1 Design Considerations

- By default, the MQTT client does not use TLS; communication is not secure between a MQTT client and broker.
- The RA FSP MQTT client does not add the NetX Duo TLS session block. It only adds 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 the `nxd_mqtt_client_secure_connect()` API.

2.1.2 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 FSP.
- Supports QoS and provides the ability to choose the 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 TLS Session Module Introduction

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

The NetX Duo TLS Session module is based on Azure RTOS 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.

2.2.1 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 capabilities 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.

2.2.2 Supported Features

- Support for RFC 2246 Transport Layer Security (TLS) Protocol Version 1.0
- Support for RFC 5246 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 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 Azure IoT Device SDK Module Introduction

The Azure IoT device SDK is a set of libraries designed to simplify the process of developing IoT applications for Azure Cloud to make sending and receiving messages easy from the Azure IoT Hub service. There are different variations of the SDK, each targeting a specific platform, but in this app note we will be describing the Azure IoT device SDK for C.

The Azure IoT device SDK for C is written in ANSI C (C99) to maximize portability. This feature makes the libraries well suited to operate on multiple platforms and devices, especially where minimizing disk and memory footprint is a priority.

In this app note we will cover how to initialize the device library, send data to IoT Hub, and receive messages from it.

More details on the Azure IoT Device SDK can be found in the reference link The Azure IoT device SDK for C | Microsoft Docs.

2.3.1 Design Considerations

The Azure IoT Device SDK is integrated with FSP and is available for the customers to use. To add the SDK to the application, users are required to use the **Stacks** tab and select **Networking→Azure RTOS NetX Duo IOT Middleware**.

When the components are selected using the **Stacks** tab, and the project is created, the SDK and libraries can be seen under the `ra/microsoft/azure-rtos/netxduo/addons/azure_iot` and `ra/microsoft/azure-rtos/netxduo/addons/cloud` folders.

Note: In the later sections, step by step procedure of adding the Azure IoT middleware is explained in detail

2.3.2 Supported Features

Table 1. IoT SDK Supported features

Features	Descriptions
Send device-to-cloud messages	Send device-to-cloud messages to IoT Hub with the option to add custom message properties.
Receive cloud-to-device messages	Receive cloud-to-device messages and associated properties from IoT Hub
Device twins	IoT Hub persists a device twin for each device that you connect to IoT Hub. The device can perform operations like get twin document and subscribe to desired property updates.
Direct methods	IoT Hub gives you the ability to invoke direct methods on devices from the cloud.
Device Provisioning Service (DPS)	This SDK supports connecting your device to the Device Provisioning Service via, for example, individual enrollment using an X.509 leaf certificate.
Protocol	The Azure SDK for Embedded C supports only MQTT.
Retry policies	The Azure SDK for Embedded C provides guidelines for retries, but actual retries should be handled by the application.
IoT plug and play	IoT Plug and Play enables solution builders to integrate smart devices with their solutions without any manual configuration.

3. MQTT/TLS Application Example

3.1 Application Overview

This application project demonstrates the Renesas RA IoT Cloud Connectivity solution using the FSP and uses Microsoft® Azure as the cloud provider. Ethernet is used as the primary communication interface between the MQTT device and the Azure IoT Services.

The EK-RA6M5 kit acts as an MQTT node, connects to the Azure IoT service using MQTT/TLS protocol over the Ethernet interface. The application periodically reads the on-chip temperature sensor values and publishes this information to the Azure IoT Hub. It also subscribes to a 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.

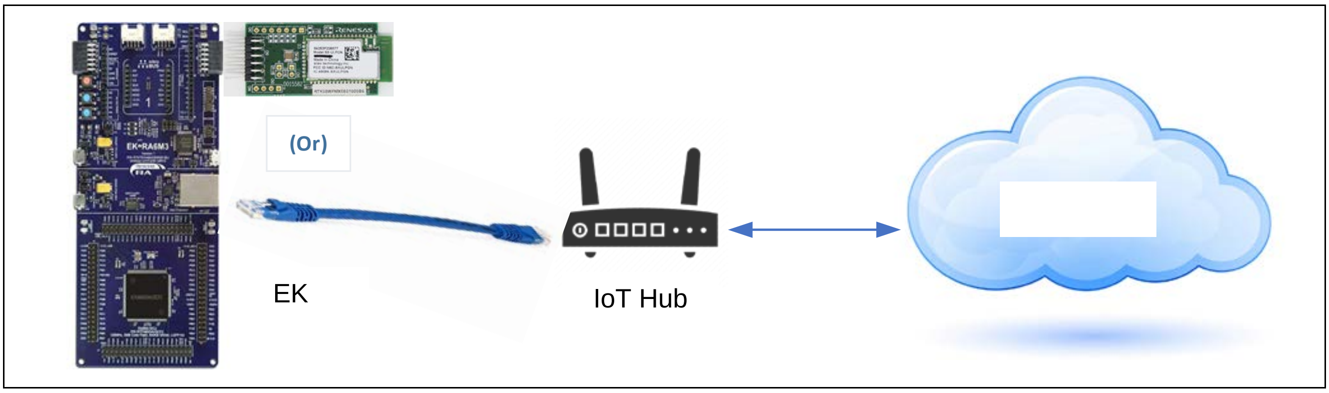


Figure 6. RA MQTT/TLS Application Overview

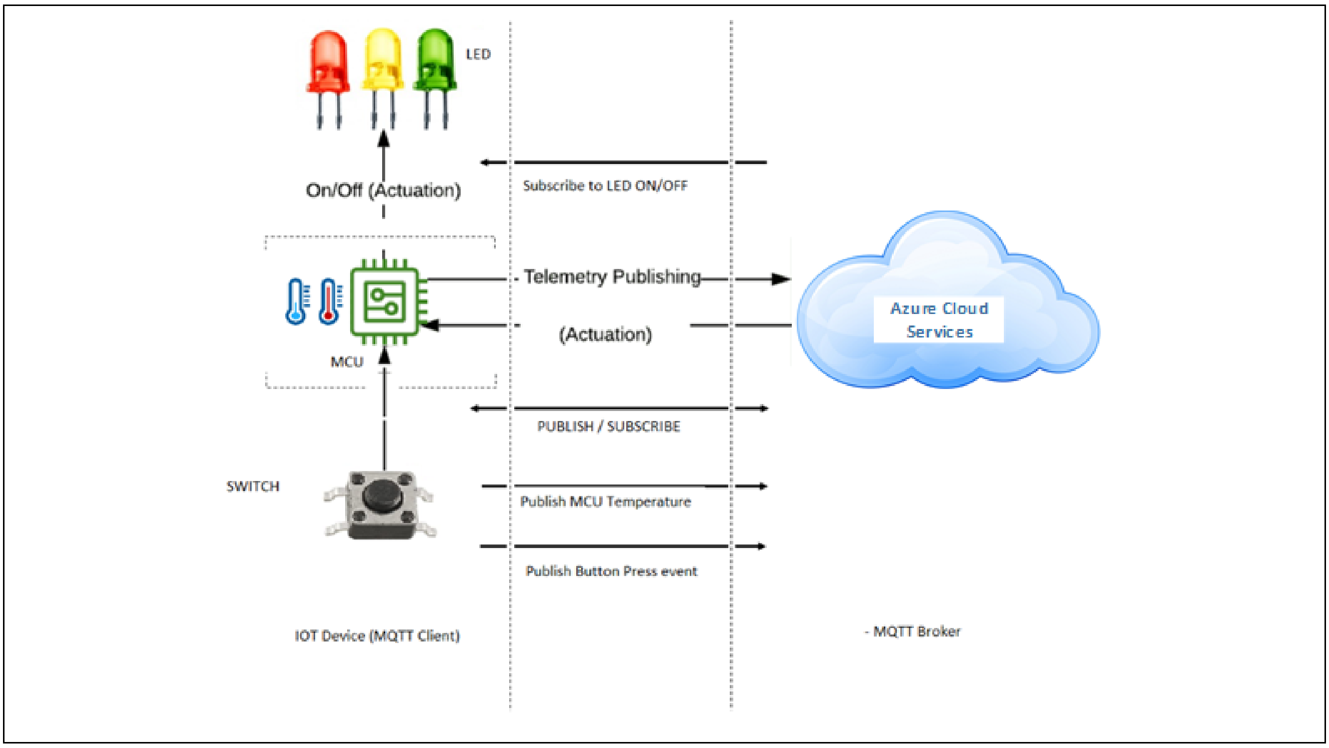


Figure 7. MQTT Publish/Subscribe to/from Azure IoT Centrale

The following files from this application project serve as a reference.

Table 2. Files Used in Application Project

No.	Filename	Purpose
1	src/application_thread_entry.c	Contains data structures functions and main thread used in Cloud Connectivity application.
2	src/common_utils.h	Contains macros, data structures, and functions commonly used across the project.
3	src/hal_entry.c	Unused file automatically generated by FSP. This file is used for non-RTOS based projects.
4	src/usr_hal.c	Contains data structures and functions used for the Hardware Abstraction Layer initialization and associated utilities.
5	src/usr_hal.h	Accompanying header for exposing functionality provided by usr_hal.c.
6	src/usr_app.c	Contains data structures and functions used to operate the user application functions.
7	src/usr_app.h	Accompanying header for exposing functionality provided by usr_app.c.
8	src/usr_network.c	Contains data structures and functions used to operate the NetX Duo TCP/IP and Ethernet Module. This file is for Ethernet-specific usage.
9	src/usr_network.h	Accompanying header for exposing functionality provided by usr_network.c. This file is for Ethernet-specific usage.
10	src/c2d_thread_entry.c	Cloud to Device handling thread
11	src/nx_azure_iot_cert.c	Azure IoT Interface code. These have the reference to the working sample implementation and other features such as Device Twin and Direct Method. These files can be used as reference for developing the application
12	src/nx_azure_iot_cert.h	
13	src/nx_azure_iot_ciphersuites.c	
14	src/nx_azure_iot_ciphersuites.h	
15	src/sample_azure_iot_embedded_sdk.c	
16	src/sample_config.h	
17	src/sample_device_identity.c	
18	src/SEGGER_RTT/SEGGER_RTT.c	Implementation of SEGGER real-time transfer (RTT) which allows real-time communication on targets which support debugger memory accesses while the CPU is running.
19	src/SEGGER_RTT/SEGGER_RTT.h	
20	src/SEGGER_RTT/SEGGER_RTT_Conf.h	
21	src/SEGGER_RTT/SEGGER_RTT_printf.c	

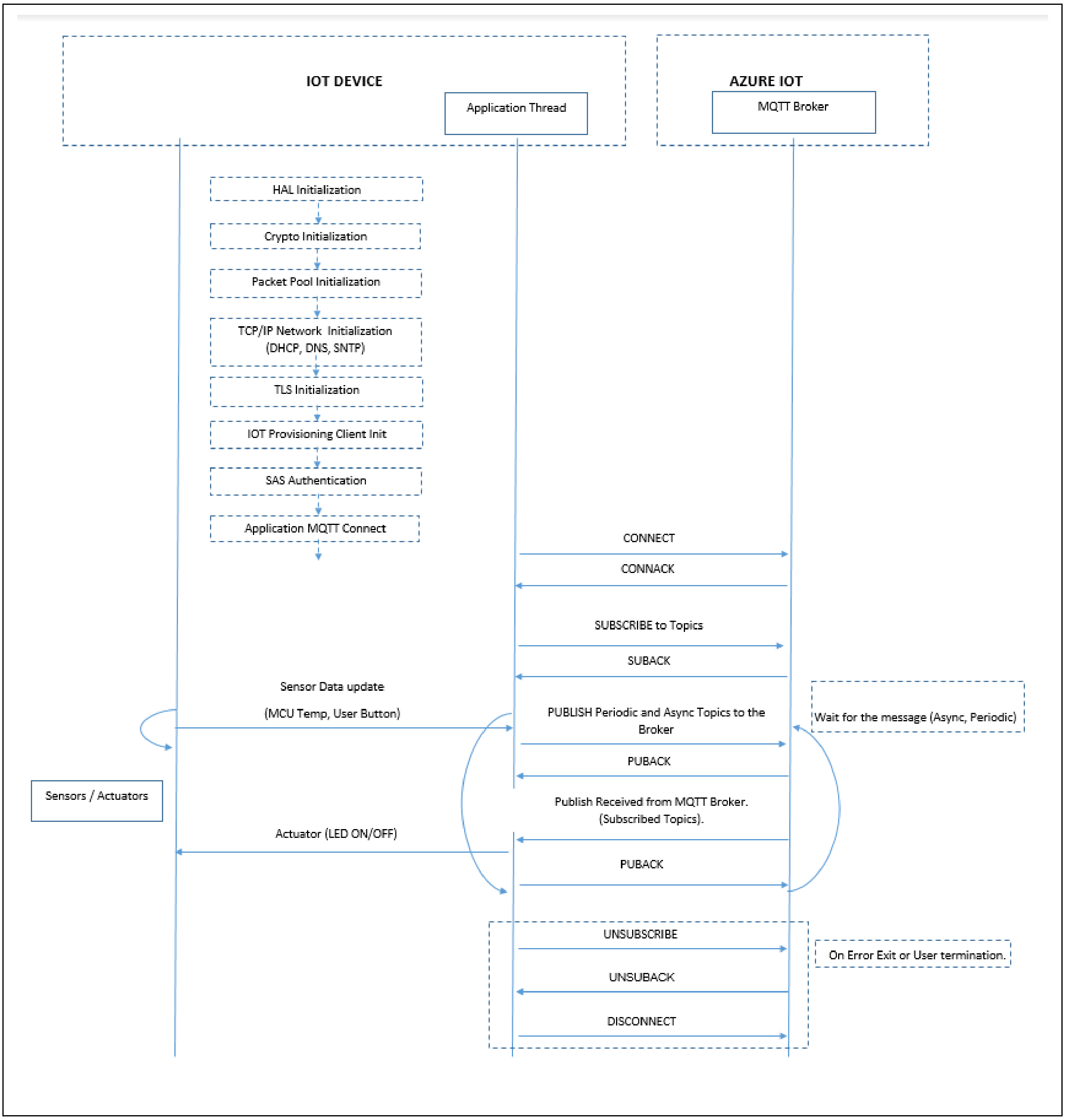


Figure 8. Application Example Implementation Details

3.2 Creating the Application Project using the FSP configurator

Complete steps to create the project from the start using the e^2 studio and FSP configurator. The table below shows the step-by-step process in creating the project. It is assumed that the user is familiar with the e^2 studio and FSP configurator. Launch the installed e^2 studio for the FSP.

Table 3. Step-by-step Details for Creating the Application Project

	Steps	Intermediate Steps
1	Project Creation:	**File → New → Renesas C/C++ Project → Renesas RA**
2	Project Template**:** Templates for Renesas RA Project	**Renesas RA C/C++ Project → Next**

	Steps	Intermediate Steps
3	e^2 studio - Project Configuration: **Renesas RA C/C++ Project** **Project Name and Location**	**Project Name** (Name for the project of your choice) → **Next**
4	**Device and Tools Selection**	
	Device Selection	**FSP Version: 3.7.0**
		Board: EK-RA6M5
		Device: R7FA6M5BH3CFC
		RTOS: Azure RTOS ThreadX
		Language: C
5	**Toolchains**	**Toolchain: GNU ARM Embedded** (Default)
		Toolchain version: 10.3.1.20210824
		Debugger: J-Link ARM → **Next**
6	**Project Type Selection**	**Flat (Non-TrustZone) Project** → **Next**
7	**Build Artifact and RTOS Selection**	**Build Artifact Selection: Executable** **RTOS Selection: Azure RTOS ThreadX (v6.1.10+fsp3.7.0)** → **Next**
8	**Project Template Selection**	**Azure RTOS ThreadX – Minimal** → **Finish**
9	**Stacks** Tab (Part of the **FSP Configurator**)	**Threads** → **New Thread**
10	Configure **Properties** → **Thread**	**Symbol: application_thread**
		Name: Application Thread
		Stack size (bytes): 4096
		Priority: 2
		Auto start: Enabled
		Time slicing interval (ticks): 1
		Note: The stack size of the application thread needs to be a minimum of 4096 or greater. This is the requirement for the NetX Duo Crypto use.
11	Adding the NetX DHCP, IoT Middleware, SNTP Clients and Packet Pool to the Application Thread Keep the default names **g_dhcp_client0**, **g_dns0**, **g_sntp_client0**. The default configuration provided by FSP configurator is used, so there is no need to change any of the specific configuration in the **Property** window.	
	Adding DHCP Client	
	New Stack	**Networking** → **Azure RTOS NetX Duo DHCP IPv4 Client**
	Adding Packet Pool for the DHCP Client	Click on **Add NetX Duo Packet Pool** → **Use**→ **g_packet_pool0 NetX Duo Packet Pool Instance**
	Adding NetX Duo Network Driver	Click on **Add NetX Duo Network Driver** → **New** → **NetX Duo Ethernet Driver**
	Modifying the **BSP** tab → **Properties** → **RA Common** for Main stack and Heap Settings)	
	Property settings for **RA Common**	**Main stack size(bytes): 0x1000**
		Heap size (bytes): 0x1000
	Adding Azure RTOS NetX Duo IoT Middleware	
	New Stack	**Networking** → **Azure RTOS NetX Duo IoT Middleware**
	Adding NetX Duo IP instance for DNS Client	Click on **Add NetX Duo IP Instance** → **Use** → **g_ip0 NetX Duo IP Instance**
	Adding Packet Pool for the DNS Client	Click on **Add NetX Duo Packet Pool** →**Use** → **g_packet_pool0 NetX Duo Packet Pool Instance**

	Steps	Intermediate Steps
	Note: After the Azure IoT Middleware is added, the configurator reports following errors when you hover over the red Blocks. **Error: NetX Duo Azure IoT Middleware Requires NetX Secure to be enabled.** **Error: NetX Duo Azure IoT Middleware Requires IP Packet Filter to be enabled.** **Error: NetX Duo Azure IoT Middleware Requires MQTT Cloud to be enabled.** **Error: A NetX Crypto Implementation must be added.** Note: To fix these errors, enable them as explained in the following steps	
	Enable the NetX Secure	**g_dns0 Azure RTOS NetX Duo DNS Client →Property → Common → MQTT → Client → NX Secure: Enable**
	Enable MQTT Cloud	**g_dns0 Azure RTOS NetX Duo DNS Client →Property → Common → MQTT → Client → Cloud Enable: Enable**
	Enable IP Packet Filter	**g_dns0 Azure RTOS NetX Duo DNS Client →Property → Common → Common → IP Packet Filter: Enabled**
	Add NetX Crypto Implementation	**Click on Add NetX SW Only or HW/SW Implementation → New → Azure RTOS NetX Crypto HW Acceleration**
	Enable the Extended Notify Support	**g_dns0 Azure RTOS NetX Duo DNS Client →Property → Common → Common →Extended Notify Support: Enabled**
12	NetX Secure Component is added from the HW Crypto perspective. IoT SDK also works with SW crypto. But in this application the HW Crypto Accelerators are used.	
	Configure NetX Secure property values (Only values which changed from the default are shown here)	
	PSK Cipher Suite	**Enable**
	ECC Cipher Suite	**Enable**
	TLSv1.0	**Enable**
	TLSv1.1 Legacy Mode	**Enable**
	TLSV1.1	**Enable**
	TLSV1.3	**Enable**
	Server Mode	**Disable**
	Configure Azure RTOS NetX Crypto HW Acceleration property values (Only values which changed from the default are shown here)	
	Common→Hardware Acceleration→ Public Key Cryptography (PKC)→ RSA→RSA 3072 Verify/Encryption (HW)	**Enabled**
	Common→Hardware Acceleration → Public Key Cryptography (PKC) → RSA → RSA 4096 Verify/Encryption (HW)	**Enabled**
	Common-> Standalone Usage	**Use with TLS**
	Note: Increase the Stack size in the BSP Tab to get rid of the error in configurator for NetX Crypto HW Acceleration	Refer to the Modifying the **BSP** tab → **Properties** → **RA Common** for (Main stack and Heap Settings) section in step 11 of this table Note: For crypto operation it is recommended to have astack size of 4K or more**.**
	Adding SNTP Client	
	New Stack	**Networking → Azure RTOS NetX Duo SNTP Client**
	Adding NetX Duo IP instance for SNTP Client	Click on **Add NetX Duo IP Instance →Use → g_ip0 NetX Duo IP Instance**
	Adding Packet Pool for the SNTP Client	Click on **Add NetX Duo Packet Pool →Use → g_packet_pool0 NetX Duo Packet Pool Instance**

	Steps	Intermediate Steps
	Increase the **Number of Packets in Pool**	
		Click on **g_packet_pool0 NetX Duo Packet Pool Instance → Properties window → Number of Packets in Pool**. Change from **16** to **50** (To allow enough buffer for the packets). This can be tuned based on the frequency and size
	Note: After adding the SNTP the configurator reports the ollowing errors when you hover over the red Blocks. **Error: Maximum time adjustment (milliseconds) should be greater than unicast poll interval (seconds).** Note: To fix these errors, enable them as explained in the following steps	
	Reduce the starting poll interval for unicast update request (seconds)	**g_sntp_client0 Azure RTOS NetX Duo SNTP Client → Property → Common → SNTP → Client →Starting poll interval for unicast update request (seconds): 36**
13	Add Cloud to Device Processing Thread to the Application	
	Stacks tab (Part of the **FSP Configurator**)	**Threads → New Thread**
	Configure Thread Properties	
	Symbol	**c2d_thread**
	Name	**Cloud2Device Thread**
	Stack size	**2048 Bytes**
	Priority	**2**
	Auto start	**Disabled**
	Time slicing interval (ticks)	**1**
14	Adding the HAL Modules as required for the Application Project: Here, ADC, Timer0, External IRQ are used for MCU temperature, 30-second periodic timer, and push button switches, respectively.	
	HAL/Common Stacks → New Stack	**Input → External IRQ Driver on r_icu**
	Property Settings for r_icu	**Name: pushButtonS1**
		Channel: 10
		Trigger: Rising
		Digital Filtering: Enabled
		Digital Filtering Sample Clock: PCLK/64
		Pin Interrupt Priority: Priority 10
		Callback: pb_callback
	HAL/Common Stacks → New Stack	**Driver → Input → External IRQ Driver on r_icu**
	Property Settings for r_icu	Name: **pushButtonS2**
		Channel: **9**
		Trigger: **Rising**
		Digital Filtering: **Enabled**
		Digital Filtering Sample Clock: **PCLK/64**
		Pin Interrupt Priority: **Priority 10**
		Callback: **pb_callback**
	HAL/Common Stacks → New Stack	**Timers → Timer Driver on r_gpt**
	Property Settings for r_gpt → **General**	**Name: gpt**
		Channel: 0
		Mode: Periodic
		Period: 30
		Period Unit: Seconds
	Interrupts:	**Callback: g_gpt_timer_cb**
		Overflow/Crest Interrupt Priority: Priority 10
	HAL/Common Stacks → New Stack	**Analog → ADC Driver on r_adc**
	Property Settings for r_adc → General	**Name: adc**

	Steps	Intermediate Steps
		Unit: 0
		Resolution: 12-bit
		Alignment: Right
		Clear after read: On
		Mode: Continuous Scan
		Double-trigger: Disabled
	Property Settings for r_adc → **Input**	**Channel Scan Mask: Temperature Sensor**
16	Adding Azure RTOS Objects for the Application (Topic Queue needs to be created for the application – Message Queue)	
	Stacks Tab → Objects	**New Object → Queue**
	Property Settings for the Queue	**Name: Topic Queue**
		Symbol: g_topic_queue
		Message Size (Words): 16
		Queue Size (Bytes): 64

The above configuration is a prerequisite to generate the required stack and features for the cloud connectivity application provided with this app note. Once the **Generate Project Content** button is clicked, e^2 studio generates the source code for the project. The generated source code contains the required drivers, stacks, and middleware. The user application files must be added into the `src` folder.

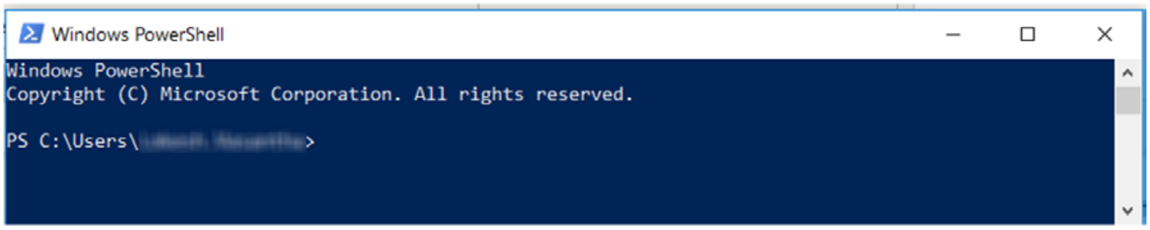
For the validation of the created project, the same source files listed in the section MQTT/TLS Application SW Architecture Overview (Table 2) may be added. This is the quickest way to create and build the application without writing the code for the configuration created in the above section.

Note: After you follow instructions in section 3.2 to recreate the Application project using FSP configurator and add the src code to the project, the project is ready for building.

3.3 Install Azure CLI

To prepare Azure cloud resources and connect a device to Azure, you can use Azure CLI. Azure CLI can be installed locally on your PC.

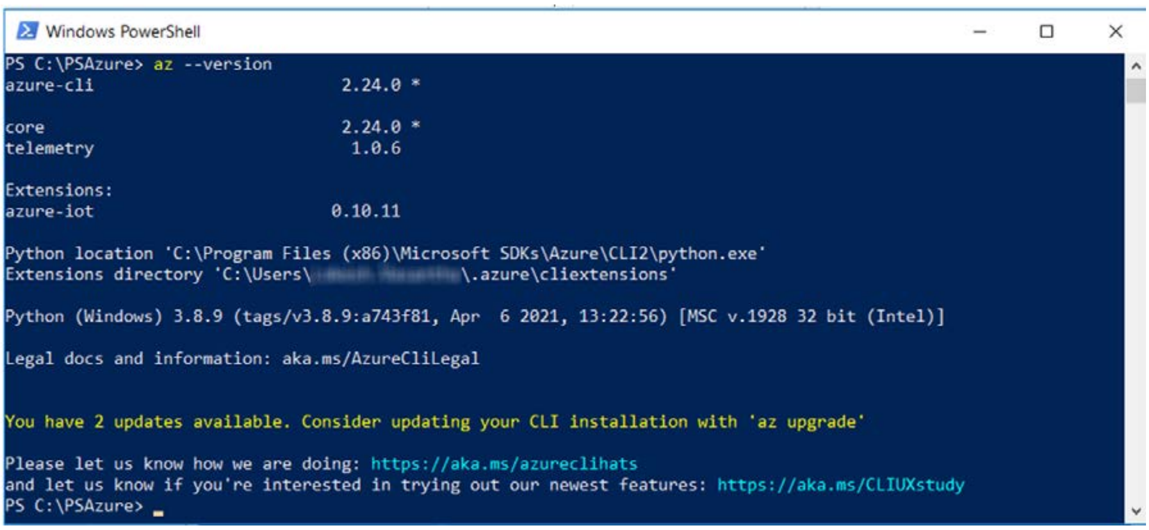
1. Azure CLI can be downloaded from the Microsoft site (https://docs.microsoft.com/en-us/cli/azure/install-azure-cli)
2. Note: The installer name will be similar to `azure-cli-2.24.x.msi`. or later. Click on the installer and the install shield will guide you through the installation process.
3. Install the current release of the Azure CLI. After the installation is complete, you will need to close and reopen any active Windows Command Prompt or PowerShell windows to use the Azure CLI.
4. After the Azure CLI installation is successful, open and launch the Windows PowerShell to use the Azure CLI. A screenshot of the Windows PowerShell is shown below.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\[redacted]>
```

Figure 9. Windows Power Shell

5. If you already have Azure CLI installed locally, run `az --version` to check the version. This app note requires Azure CLI 2.24.0 or later.



```
PS C:\PSAzure> az --version
azure-cli                         2.24.0 *

core                              2.24.0 *
telemetry                          1.0.6

Extensions:
azure-iot                         0.10.11

Python location 'C:\Program Files (x86)\Microsoft SDKs\Azure\CLI2\python.exe'
Extensions directory 'C:\Users\[redacted]\.azure\cliextensions'

Python (Windows) 3.8.9 (tags/v3.8.9:a743f81, Apr  6 2021, 13:22:56) [MSC v.1928 32 bit (Intel)]

Legal docs and information: aka.ms/AzureCliLegal


You have 2 updates available. Consider updating your CLI installation with 'az upgrade'

Please let us know how we are doing: https://aka.ms/azureclihats
and let us know if you're interested in trying out our newest features: https://aka.ms/CLIUXstudy
PS C:\PSAzure>
```

Figure 10. Azure CLI Version

3.4 Create an IoT Hub

You can use Azure CLI to create an IoT hub that handles events and messaging for your device.

Note 1: Before you start creating the IoT Hub you are required to have a login to your Azure Portal via web browser. If not, then you may notice an error that you are not logged in while creating the IoT hub, you may notice an error that you are not logged in.

https://portal.azure.com/

Note 2: If you do not have the Azure Account, you can create one which is valid for 12 months with limited features from the following link

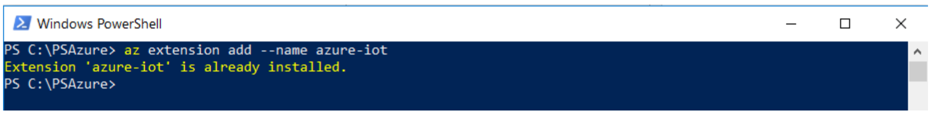
https://azure.microsoft.com/en-us/free/

To create an IoT hub:

Note 3: Some of the user parameters while creating the IoT Hub needs to be unique. Users are required to take care of this while creating the IoT Hub credentials.

1. In your CLI console, run the `az extension add` command to add the Microsoft Azure IoT Extension for Azure CLI to your CLI shell. The IoT Extension adds IoT Hub, IoT Edge, and IoT Device Provisioning Service (DPS) specific commands to Azure CLI.
   — `az extension add --name azure-iot`

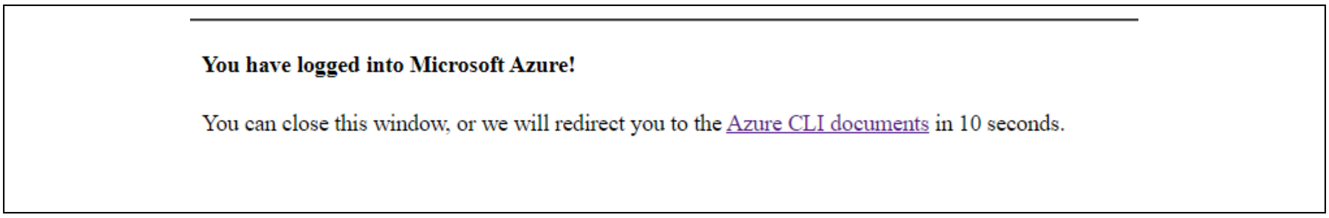
Note 4: When you run the command for the first time you may not notice output on the console as shown below. It just accepts the command.



```
PS C:\PSAzure> az extension add --name azure-iot
Extension 'azure-iot' is already installed.
PS C:\PSAzure>
```

Figure 11. Add Extension for Azure CLI

2. Run the `az login` command to login to the Azure account. Running the `az login` command opens the browser for login. You can enter the login credentials to login to the Azure account. You will notice a similar message on the browser on successful login.
   Note: You can find more info on the Azure CLI at [Overview of the Azure CLI | Microsoft Docs]

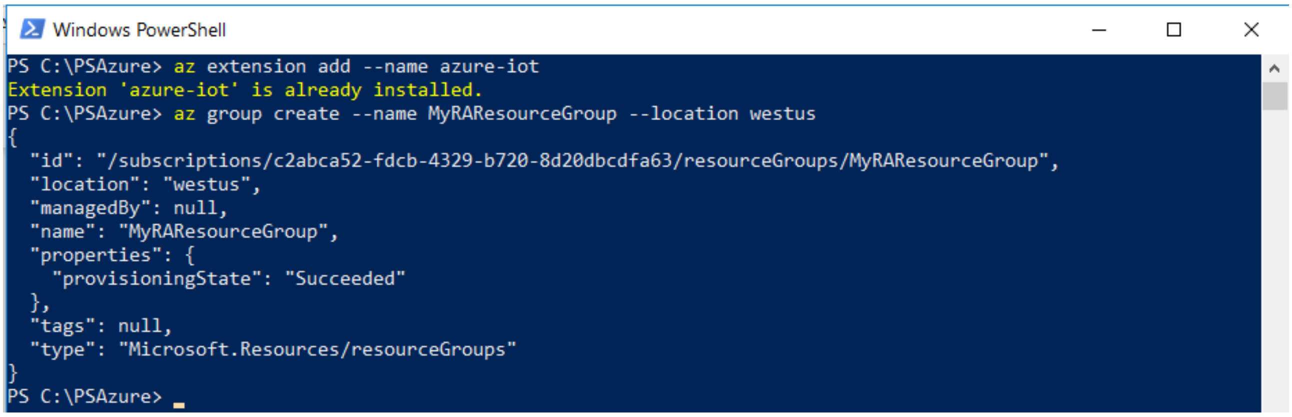


You have logged into Microsoft Azure!

You can close this window, or we will redirect you to the Azure CLI documents in 10 seconds.

Figure 12. Successful Login to the Azure account

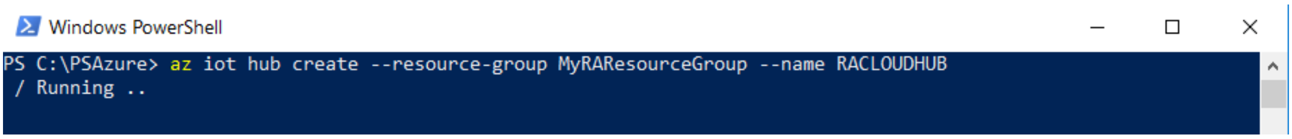
3. Run the `az group create` command to create a resource group. The following command creates a resource group named `MyRAResourceGroup` in the `westus` region.
4. Note: Optionally, to set an alternate `location`, run `az account list-locations` to see available locations. Then specify the alternate location in the following command in place of `westus`.
   — `az group create --name MyRAResourceGroup --location westus`



```
PS C:\PSAzure> az extension add --name azure-iot
Extension 'azure-iot' is already installed.
PS C:\PSAzure> az group create --name MyRAResourceGroup --location westus
{
  "id": "/subscriptions/c2abca52-fdcb-4329-b720-8d20dbcdfa63/resourceGroups/MyRAResourceGroup",
  "location": "westus",
  "managedBy": null,
  "name": "MyRAResourceGroup",
  "properties": {
    "provisioningState": "Succeeded"
  },
  "tags": null,
  "type": "Microsoft.Resources/resourceGroups"
}
PS C:\PSAzure> _
```

Figure 13. Create Resource Group

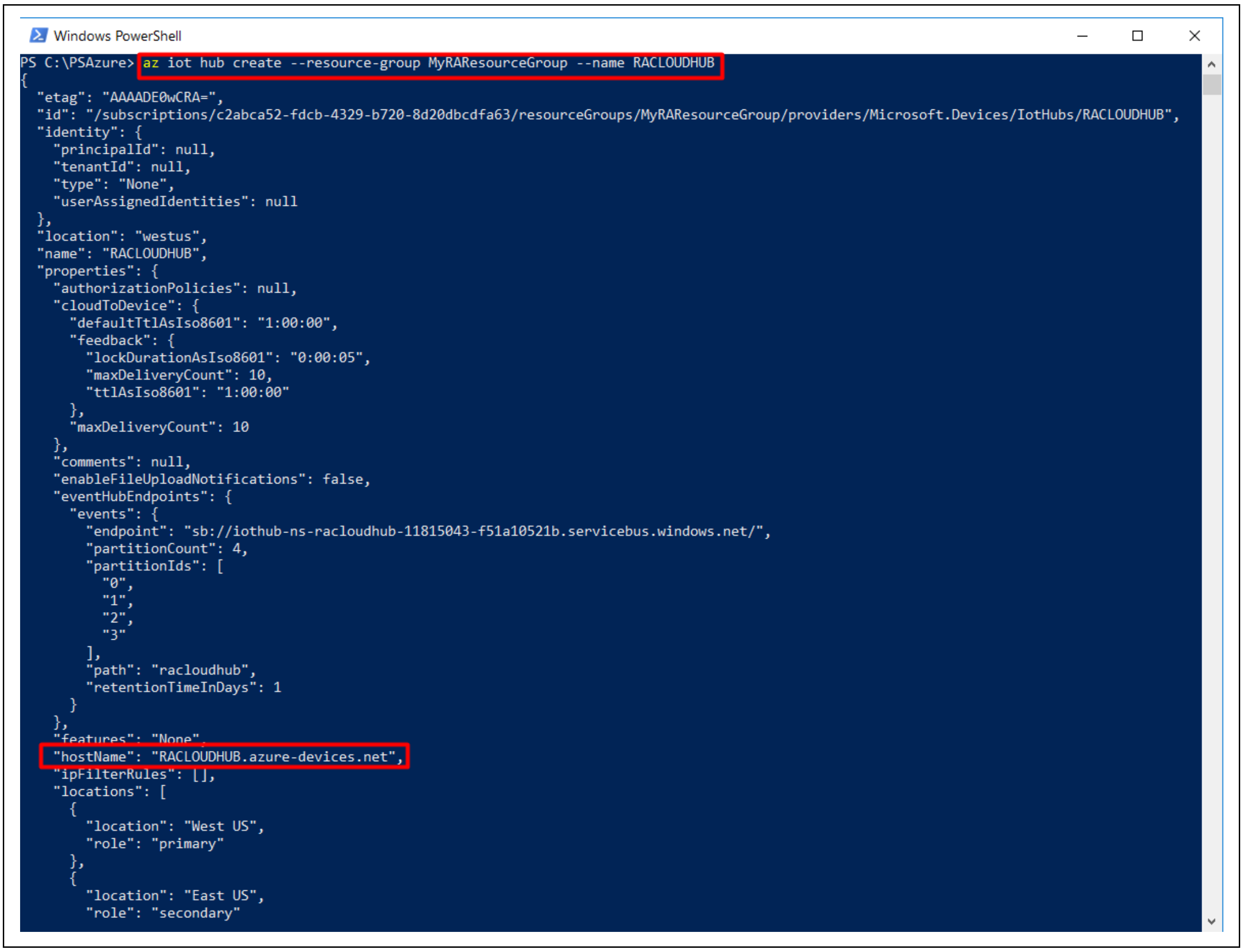
5. Run the `az iot hub create` command to create an IoT hub. It might take a few minutes to create an IoT hub.
Replace the *`YourIotHubName`* placeholder below with the name you chose for your IoT hub. An IoT hub name must be globally unique in Azure. This placeholder is used in the rest of this tutorial to represent your unique IoT hub name.
— `az iot hub create --resource-group MyRAResourceGroup --name {YourIoTHubName}`
Note: It may take few minutes to create the IoT Hub. In this case the IoTHub name used is `RACLOUDHUB`.



```
PS C:\PSAzure> az iot hub create --resource-group MyRAResourceGroup --name RACLOUDHUB
 / Running ..
```

Figure 14. IoT Hub Creation in Progress

6. After the IoT hub is created, view the JSON output in the console, and copy the `hostName` value to a safe place. You use this value in a later step. The `hostName` value looks like the following example:
   - `{Your IoT hub name}.azure-devices.net`



```
PS C:\PSAzure> az iot hub create --resource-group MyRAResourceGroup --name RACLOUDHUB
{
  "etag": "AAAADE0wCRA=",
  "id": "/subscriptions/c2abca52-fdcb-4329-b720-8d20dbcdfa63/resourceGroups/MyRAResourceGroup/providers/Microsoft.Devices/IotHubs/RACLOUDHUB",
  "identity": {
    "principalId": null,
    "tenantId": null,
    "type": "None",
    "userAssignedIdentities": null
  },
  "location": "westus",
  "name": "RACLOUDHUB",
  "properties": {
    "authorizationPolicies": null,
    "cloudToDevice": {
      "defaultTtlAsIso8601": "1:00:00",
      "feedback": {
        "lockDurationAsIso8601": "0:00:05",
        "maxDeliveryCount": 10,
        "ttlAsIso8601": "1:00:00"
      },
      "maxDeliveryCount": 10
    },
    "comments": null,
    "enableFileUploadNotifications": false,
    "eventHubEndpoints": {
      "events": {
        "endpoint": "sb://iothub-ns-racloudhub-11815043-f51a10521b.servicebus.windows.net/",
        "partitionCount": 4,
        "partitionIds": [
          "0",
          "1",
          "2",
          "3"
        ],
        "path": "racloudhub",
        "retentionTimeInDays": 1
      }
    },
    "features": "None",
    "hostName": "RACLOUDHUB.azure-devices.net",
    "ipFilterRules": [],
    "locations": [
      {
        "location": "West US",
        "role": "primary"
      },
      {
        "location": "East US",
        "role": "secondary"
```

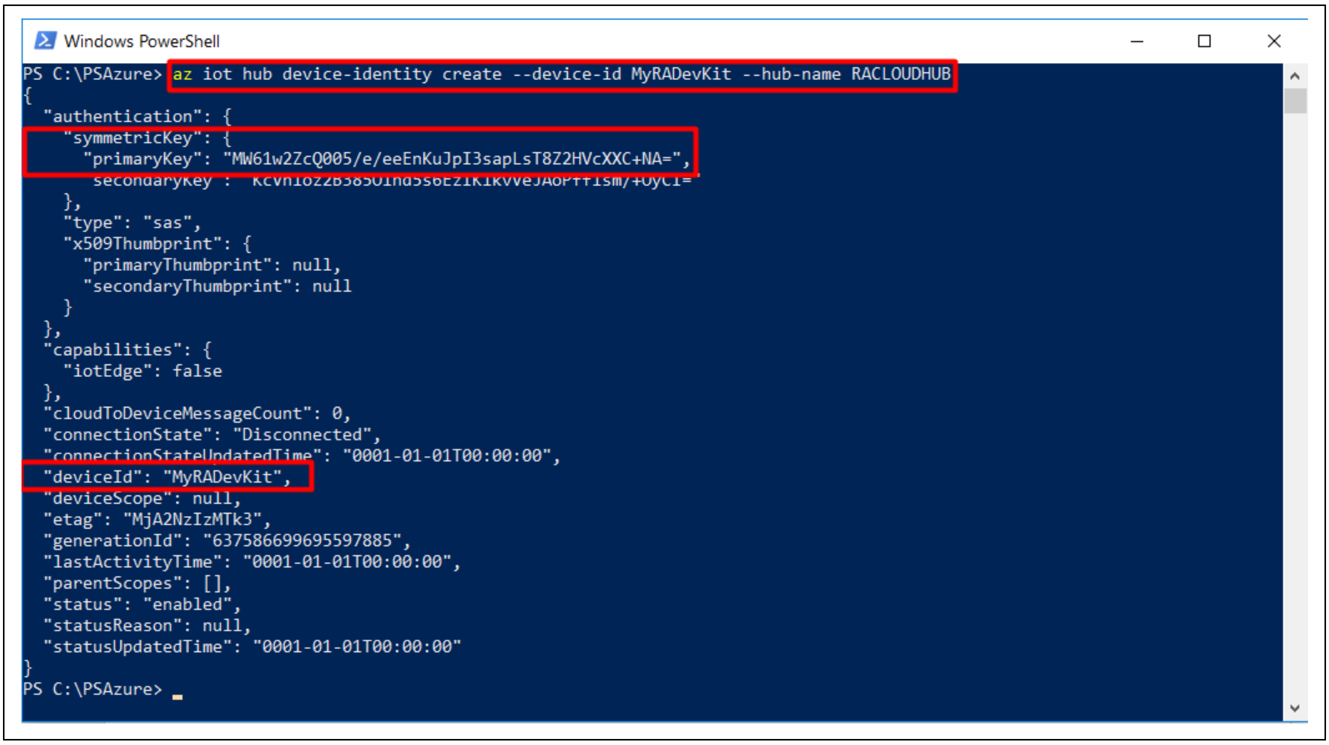
Figure 15. JSON Output after IoT Hub Creation

3.5 Register an IoT Hub Device

In this section, you create a new device instance and register it with the IoT Hub you created. You will use the connection information for the newly registered device to securely connect your physical device in a later section.

To register a device:

1. In your console, run the `az iot hub device-identity create` command. This creates the simulated device identity.
2. Replace the *`YourIotHubName`* placeholder below with the name you chose for your IoT hub.
3. You can use the *`MyRADevKit`* name directly for the device in CLI commands in this tutorial. Optionally, use a different name.
   - `az iot hub device-identity create --device-id MyRADevKit --hub-name {YourIoTHubName}`



```
PS C:\PSAzure> az iot hub device-identity create --device-id MyRADevKit --hub-name RACLOUDHUB
{
  "authentication": {
    "symmetricKey": {
      "primaryKey": "MW61w2ZcQ005/e/eeEnKuJpI3sapLsT8Z2HVcXXC+NA=",
      "secondaryKey": "KCVn1oZ2B385O1nd5s6EZ1K1kVveJAoPf+1sm/+0yC1="
    },
    "type": "sas",
    "x509Thumbprint": {
      "primaryThumbprint": null,
      "secondaryThumbprint": null
    }
  },
  "capabilities": {
    "iotEdge": false
  },
  "cloudToDeviceMessageCount": 0,
  "connectionState": "Disconnected",
  "connectionStateUpdatedTime": "0001-01-01T00:00:00",
  "deviceId": "MyRADevKit",
  "deviceScope": null,
  "etag": "MjA2NzIzMTk3",
  "generationId": "637586699695597885",
  "lastActivityTime": "0001-01-01T00:00:00",
  "parentScopes": [],
  "status": "enabled",
  "statusReason": null,
  "statusUpdatedTime": "0001-01-01T00:00:00"
}
PS C:\PSAzure> _
```

Figure 16. IoT Hub Creation in Progress

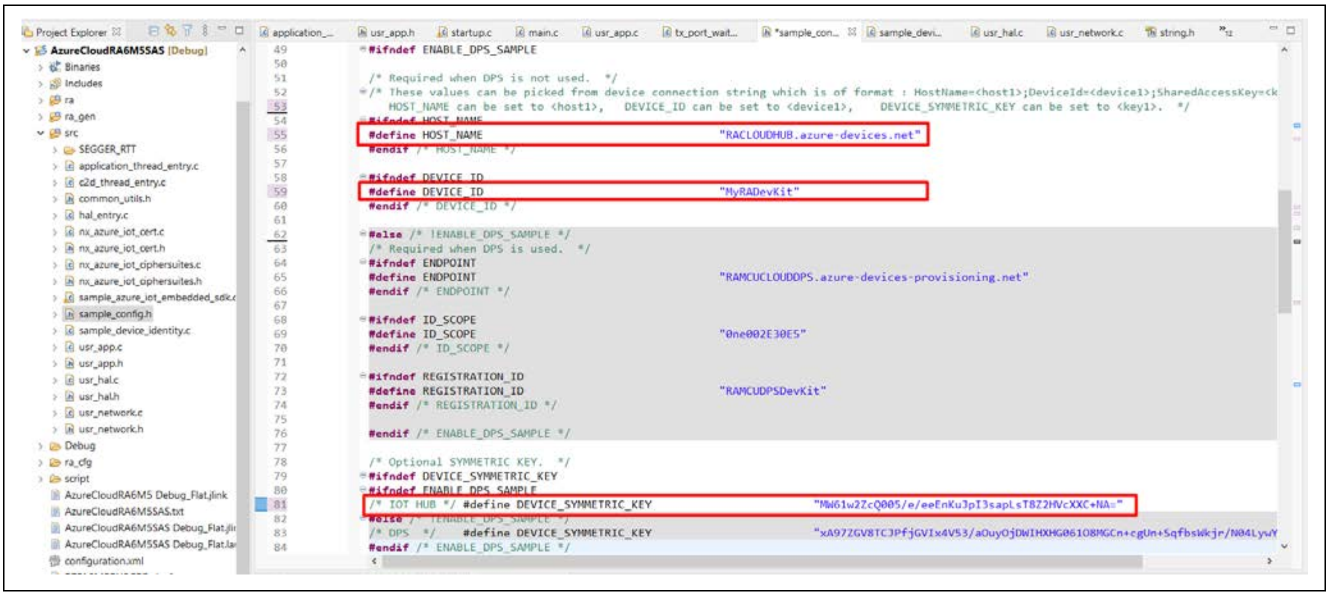
4. After the device is created, view the JSON output in the console, and copy the `deviceId` and `primaryKey` values for use in a later step.
5. Confirm that you have saved or copied the following values from the JSON outputs from the previous sections to use in the next section
   - `Hostname`
   - `deviceId`
   - `primaryKey`

3.6 Prepare the Device

To connect the device to Azure, modify a configuration file for Azure IoT settings, build and flash the image to the device.

Add configuration

1. Import the application project into an empty e^2 studio. pen `sample_config.h` and make the changes to the configuration as shown in the snapshot with your `Hostname`, `deviceId` and `primaryKey`.



```
#ifndef ENABLE_DPS_SAMPLE

/* Required when DPS is not used.  */
/* These values can be picked from device connection string which is of format : HostName=<host1>;DeviceId=<device1>;SharedAccessKey=<k
   HOST_NAME can be set to <host1>,   DEVICE_ID can be set to <device1>,    DEVICE_SYMMETRIC_KEY can be set to <key1>.  */
#ifndef HOST_NAME
#define HOST_NAME                                    "RACLOUDHUB.azure-devices.net"
#endif /* HOST_NAME */

#ifndef DEVICE_ID
#define DEVICE_ID                                    "MyRADevKit"
#endif /* DEVICE_ID */

#else /* !ENABLE_DPS_SAMPLE */
/* Required when DPS is used.  */
#ifndef ENDPOINT
#define ENDPOINT                                     "RAMCUCLOUDDPS.azure-devices-provisioning.net"
#endif /* ENDPOINT */

#ifndef ID_SCOPE
#define ID_SCOPE                                     "0ne002E30E5"
#endif /* ID_SCOPE */

#ifndef REGISTRATION_ID
#define REGISTRATION_ID                              "RAMCUDPSDevKit"
#endif /* REGISTRATION_ID */

#endif /* ENABLE_DPS_SAMPLE */

/* Optional SYMMETRIC KEY.  */
#ifndef DEVICE_SYMMETRIC_KEY
#ifndef ENABLE_DPS_SAMPLE
/* IOT HUB */ #define DEVICE_SYMMETRIC_KEY                   "MW61w2ZcQ005/e/eeEnKu3pI3sapLsTBZ2HVcXXC+NA="
#else /* !ENABLE_DPS_SAMPLE */
/* DPS  */    #define DEVICE_SYMMETRIC_KEY                   "xA97ZGV8TCJPfjGVIx4V53/aOuyOjDWIHXHG06108MGCn+cgUn+SqfbsWkjr/N04LywY
#endif /* ENABLE_DPS_SAMPLE */
```

Figure 17. Configuration Changes to sample_config.h

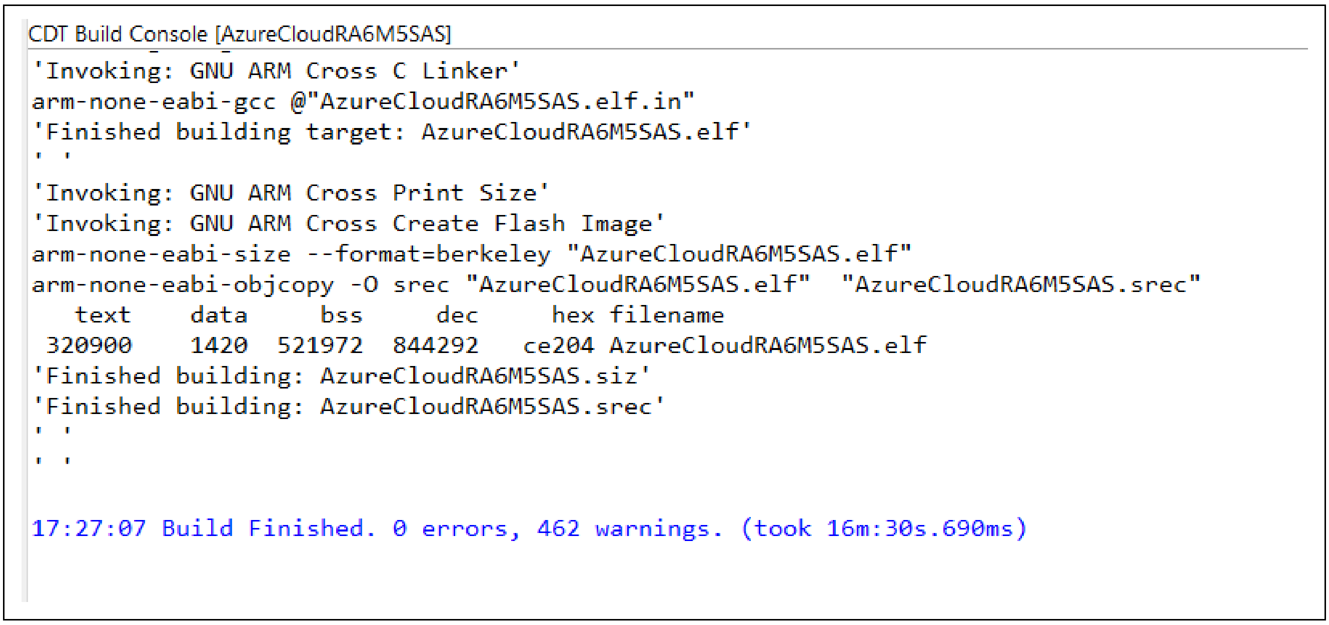
Constant name	Value
HOST_NAME	{Your IoT hub hostName value}
DEVICE_ID	{Your deviceID value}
DEVICE_SYMMETRIC_KEY	{Your primaryKey value}

3.7 Building and Running the Application

The project is now ready to compile. Press the **Build** (hammer icon) to start building the project.

Figure 18. Starting to Build the Project

The toolchain will report compilation and build status to the console pane in the lower-right corner of e^2 studio. When the build has completed, confirm that there are zero errors and few warnings. Warnings, if any, may result from highly restrictive compilation warnings settings being applied by e^2 studio to third party code.

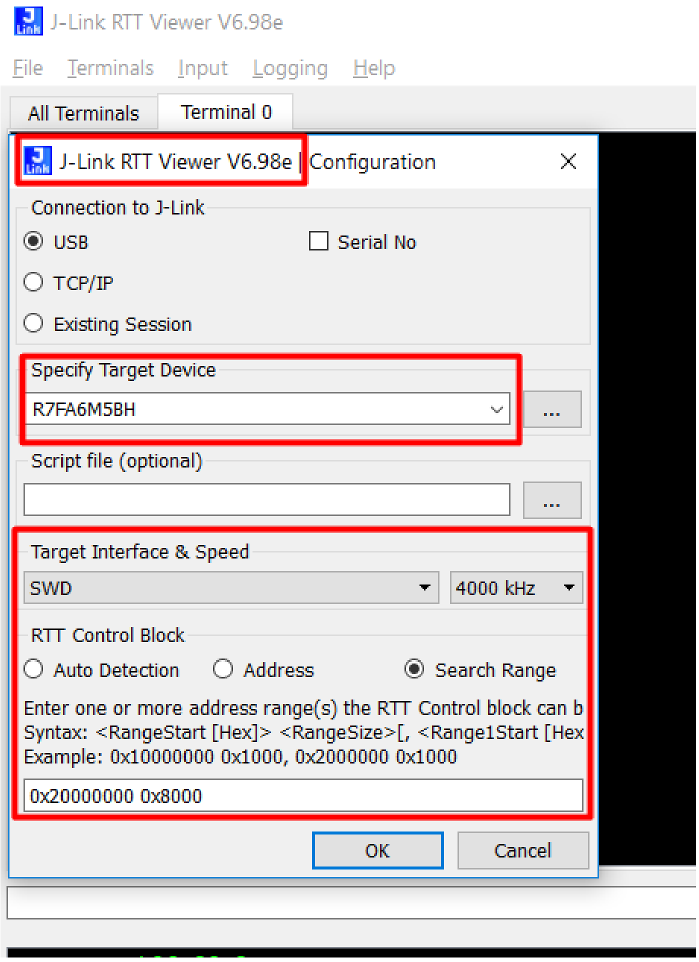


```
CDT Build Console [AzureCloudRA6M5SAS]
'Invoking: GNU ARM Cross C Linker'
arm-none-eabi-gcc @"AzureCloudRA6M5SAS.elf.in"
'Finished building target: AzureCloudRA6M5SAS.elf'
' '
'Invoking: GNU ARM Cross Print Size'
'Invoking: GNU ARM Cross Create Flash Image'
arm-none-eabi-size --format=berkeley "AzureCloudRA6M5SAS.elf"
arm-none-eabi-objcopy -O srec "AzureCloudRA6M5SAS.elf"  "AzureCloudRA6M5SAS.srec"
   text    data     bss     dec     hex filename
 320900    1420  521972  844292   ce204 AzureCloudRA6M5SAS.elf
'Finished building: AzureCloudRA6M5SAS.siz'
'Finished building: AzureCloudRA6M5SAS.srec'
' '
' '


17:27:07 Build Finished. 0 errors, 462 warnings. (took 16m:30s.690ms)
```

Figure 19. Compilation and Build Status Report

3.8 Download and Run the Project

1. Connect the micro USB cable to the DEBUG port (J10) of the EK-RA6M5 Cloud Kit and other end to the host computer.
2. Make sure the Ethernet Cable is connected to the RJ-45 connector of the board and other end to the router/switch as applicable for the internet access.
3. In e^2 studio, open the **Debug Configurations** dialog and launch the **AzureCloudRA6M5SAS Debug_Flat** debug configuration.
4. Open your RTT viewer terminal (version 6.98 or later). Configure the following values for the RTT terminal:



J-Link RTT Viewer V6.98e | Configuration

Connection to J-Link

● USB ☐ Serial No

○ TCP/IP

○ Existing Session

Specify Target Device

R7FA6M5BH

Script file (optional)

Target Interface & Speed

SWD 4000 kHz

RTT Control Block

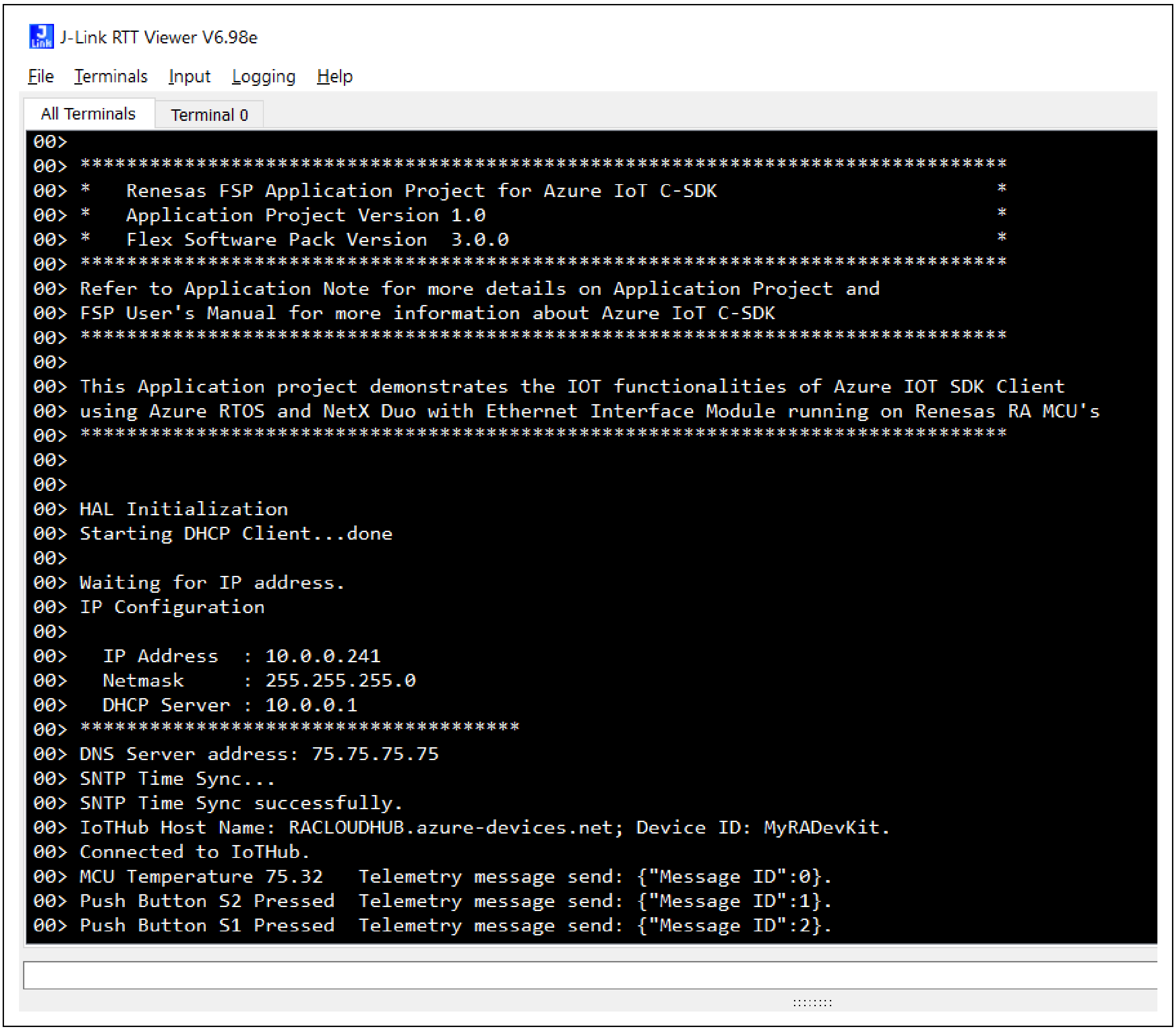
○ Auto Detection ○ Address ● Search Range

Enter one or more address range(s) the RTT Control block can b
Syntax: <RangeStart [Hex]> <RangeSize>[, <Range1Start [Hex
Example: 0x10000000 0x1000, 0x2000000 0x1000

0x20000000 0x8000

Figure 20. RTT Terminal

5. As the project runs, the demo prints out status information to the terminal output window. The demo also publishes the MCU Temperature message to IoT Hub every 30 seconds. Check the terminal output to verify that messages have been successfully sent to the Azure IoT hub:



```
00>
00> ********************************************************************************
00> *   Renesas FSP Application Project for Azure IoT C-SDK                        *
00> *   Application Project Version 1.0                                            *
00> *   Flex Software Pack Version  3.0.0                                          *
00> ********************************************************************************
00> Refer to Application Note for more details on Application Project and
00> FSP User's Manual for more information about Azure IoT C-SDK
00> ********************************************************************************
00>
00> This Application project demonstrates the IOT functionalities of Azure IOT SDK Client
00> using Azure RTOS and NetX Duo with Ethernet Interface Module running on Renesas RA MCU's
00> ********************************************************************************
00>
00>
00> HAL Initialization
00> Starting DHCP Client...done
00>
00> Waiting for IP address.
00> IP Configuration
00>
00>   IP Address  : 10.0.0.241
00>   Netmask     : 255.255.255.0
00>   DHCP Server : 10.0.0.1
00> **************************************
00> DNS Server address: 75.75.75.75
00> SNTP Time Sync...
00> SNTP Time Sync successfully.
00> IoTHub Host Name: RACLOUDHUB.azure-devices.net; Device ID: MyRADevKit.
00> Connected to IoTHub.
00> MCU Temperature 75.32   Telemetry message send: {"Message ID":0}.
00> Push Button S2 Pressed  Telemetry message send: {"Message ID":1}.
00> Push Button S1 Pressed  Telemetry message send: {"Message ID":2}.
```

Figure 21. RTT Terminal Output

Keep the terminal window open to monitor device output in subsequent steps.

3.9 View Device Properties

You can use the Azure IoT Explorer (https://docs.microsoft.com/en-us/azure/iot-pnp/howto-use-iot-explorer) to view and manage the properties of your devices. In the following steps, you'll add a connection to your IoT hub in IoT Explorer. With the connection, you can view properties for devices associated with the IoT hub.

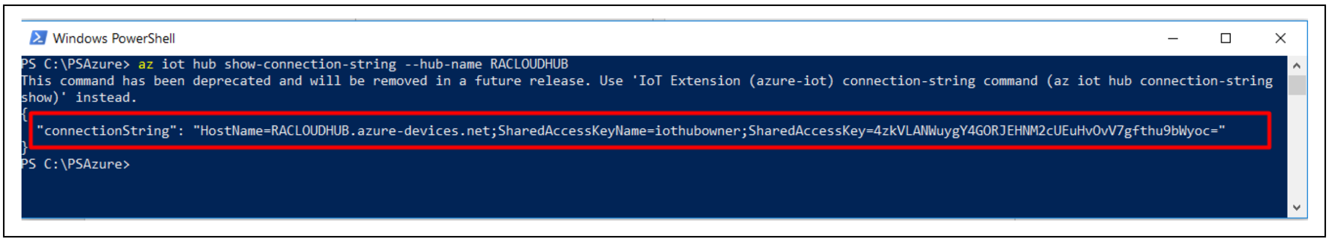
Download and install latest (above v0.14.2.0) Azure IoT Explorer from: https://github.com/Azure/azure-iot-explorer/releases

Note: Click and install the downloaded msi file `Azure.IoT.Explorer.preview.0.14.2.msi` or newer version of the downloaded file. The install shield guides you through the installation process.

3.10 Set IoT Hub

To add a connection to your IoT hub:

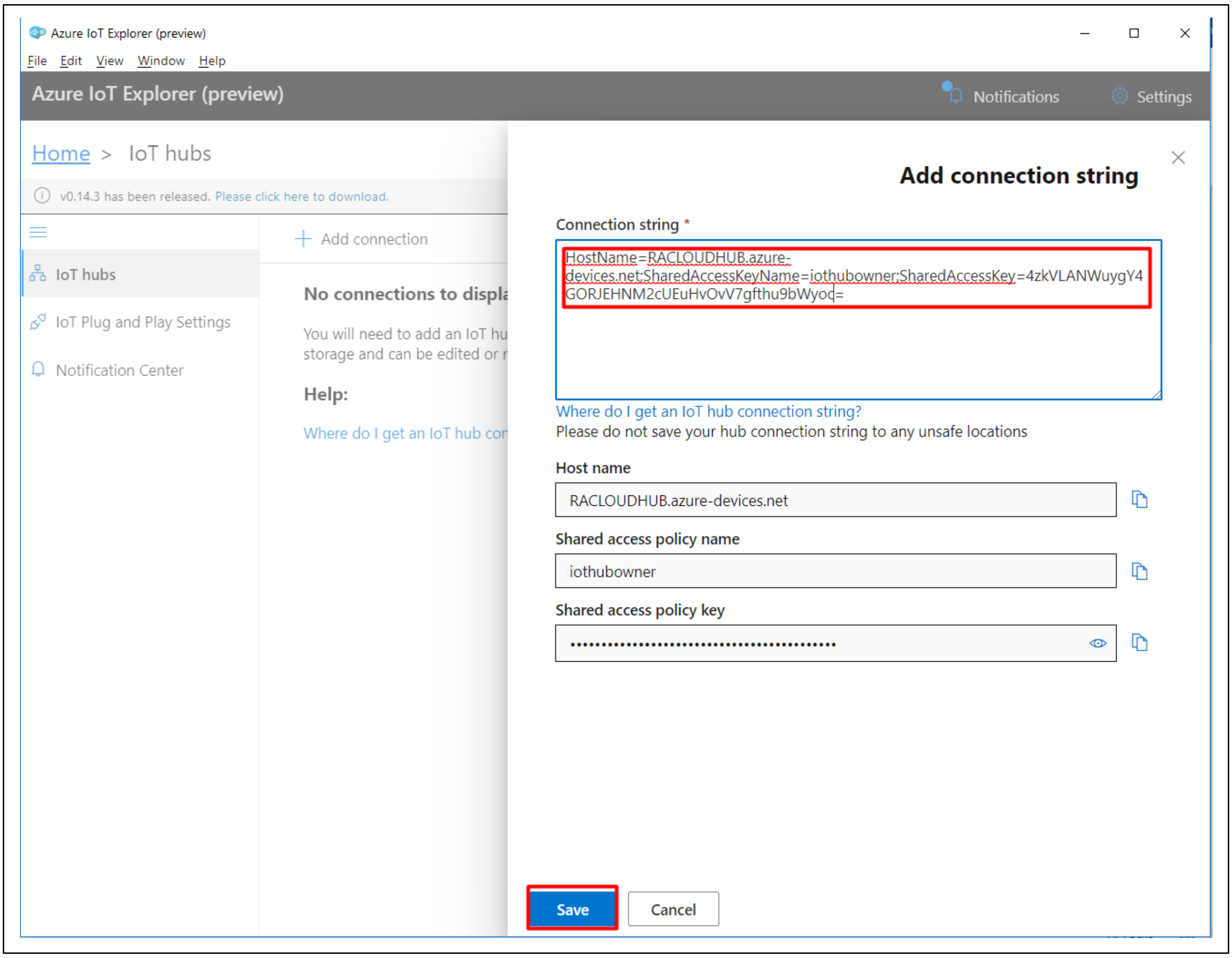
1. In your CLI console, run the `az iot hub show-connection-string` command to get the connection string for your IoT hub.
   - `az iot hub show-connection-string --name {YourIoTHubName}`



```
PS C:\PSAzure> az iot hub show-connection-string --hub-name RACLOUDHUB
This command has been deprecated and will be removed in a future release. Use 'IoT Extension (azure-iot) connection-string command (az iot hub connection-string
show)' instead.
{
  "connectionString": "HostName=RACLOUDHUB.azure-devices.net;SharedAccessKeyName=iothubowner;SharedAccessKey=4zkVLANWuygY4GORJEHNM2cUEuHvOvV7gfthu9bWyoc="
}
PS C:\PSAzure>
```

Figure 22. Connection String

2. Copy the connection string.
3. Open the Azure IoT Explorer and select **IoT hubs > Add connection**.
4. Paste the connection string into the **Connection string** box.
5. Select **Save**.



Azure IoT Explorer (preview)
File Edit View Window Help
Azure IoT Explorer (preview)
Notifications
Settings
Home > IoT hubs
IoT hubs
IoT Plug and Play Settings
Notification Center
Add connection
Save
Cancel

v0.14.3 has been released. Please click here to download.

No connections to displa

You will need to add an IoT hu storage and can be edited or r

Help:

Where do I get an IoT hub cor

Add connection string

Connection string *

HostName=RACLOUDHUB.azure-devices.net;SharedAccessKeyName=iothubowner;SharedAccessKey=4zkVLANWuygY4GORJEHNM2cUEuHvOvV7gfthu9bWyoc=

Where do I get an IoT hub connection string?

Please do not save your hub connection string to any unsafe locations

Host name: RACLOUDHUB.azure-devices.net

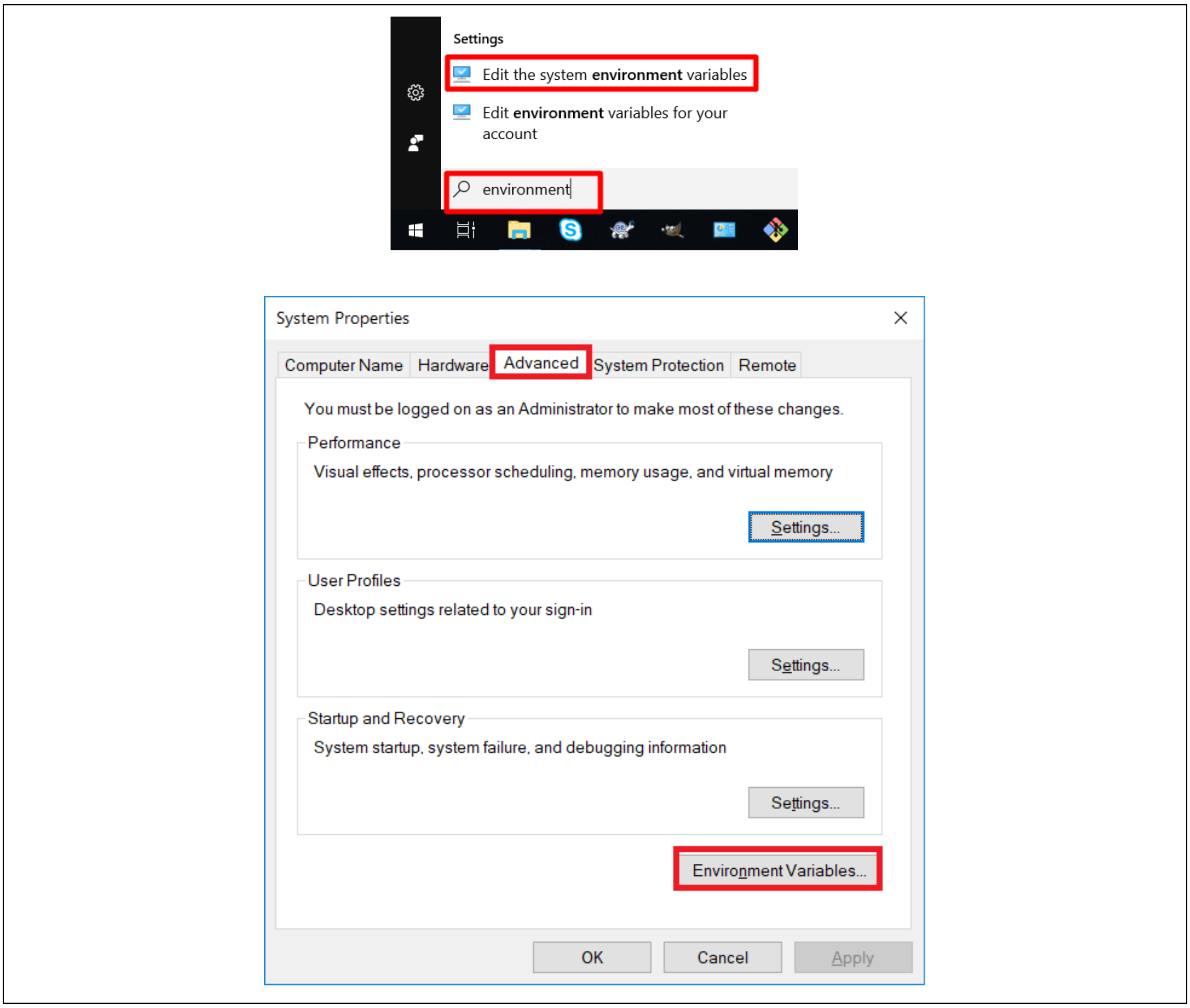
Shared access policy name: iothubowner

Shared access policy key: ••

Figure 23. Adding Connection String

Note: In some cases, Azure IoT Explorer may report an error that the default port that IoT Explorer is trying to use is being used by another application. In order to overcome this error, you can add a different port number for the Azure IoT Explorer as shown below.

Go to your PC, edit the system environmental variables similarly to the screenshots shown below.



Settings

Edit the system **environment** variables

Edit **environment** variables for your account

environment

System Properties

You must be logged on as an Administrator to make most of these changes.

Performance

Visual effects, processor scheduling, memory usage, and virtual memory

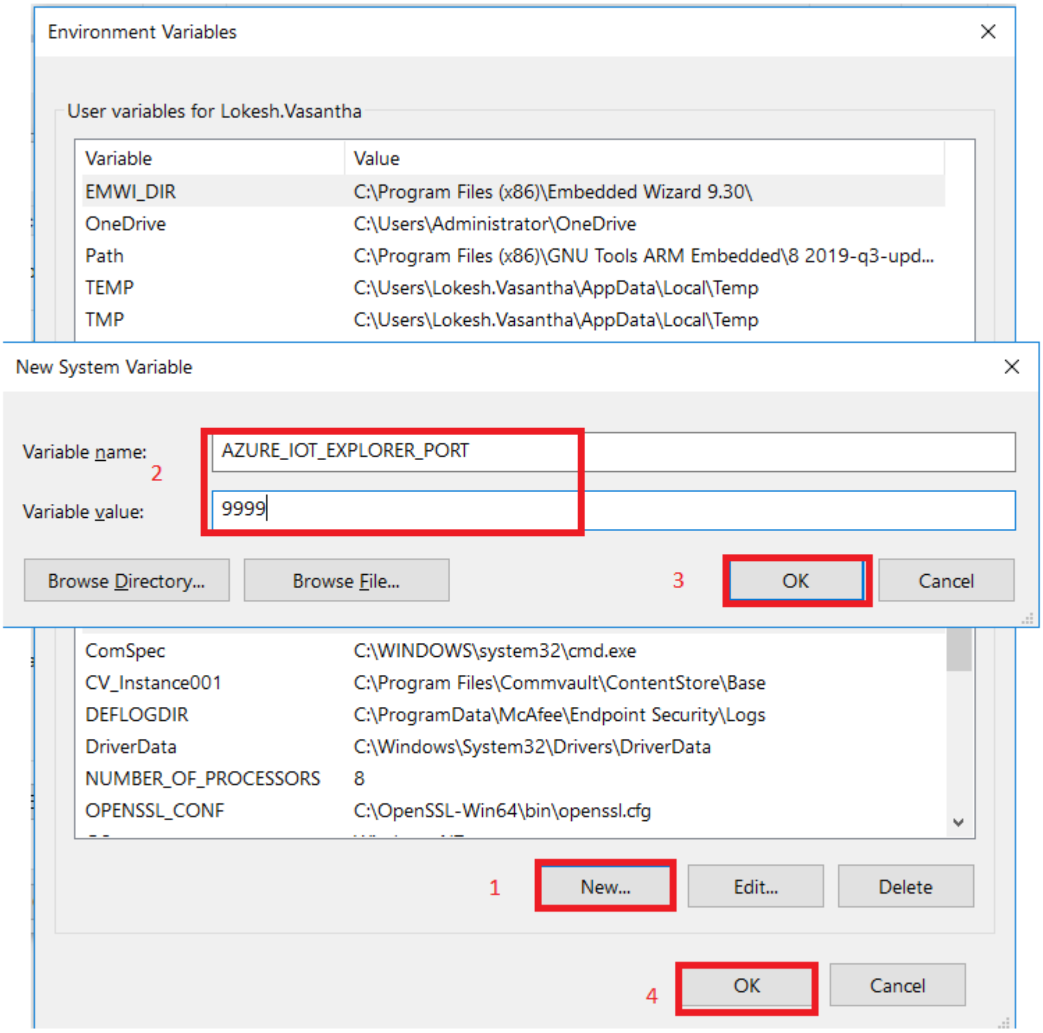
User Profiles

Desktop settings related to your sign-in

Startup and Recovery

System startup, system failure, and debugging information

Figure 24. Editing System Environment Variable



Environment Variables

User variables for Lokesh.Vasantha

Variable	Value
EMWI_DIR	C:\Program Files (x86)\Embedded Wizard 9.30\
OneDrive	C:\Users\Administrator\OneDrive
Path	C:\Program Files (x86)\GNU Tools ARM Embedded\8 2019-q3-upd...
TEMP	C:\Users\Lokesh.Vasantha\AppData\Local\Temp
TMP	C:\Users\Lokesh.Vasantha\AppData\Local\Temp

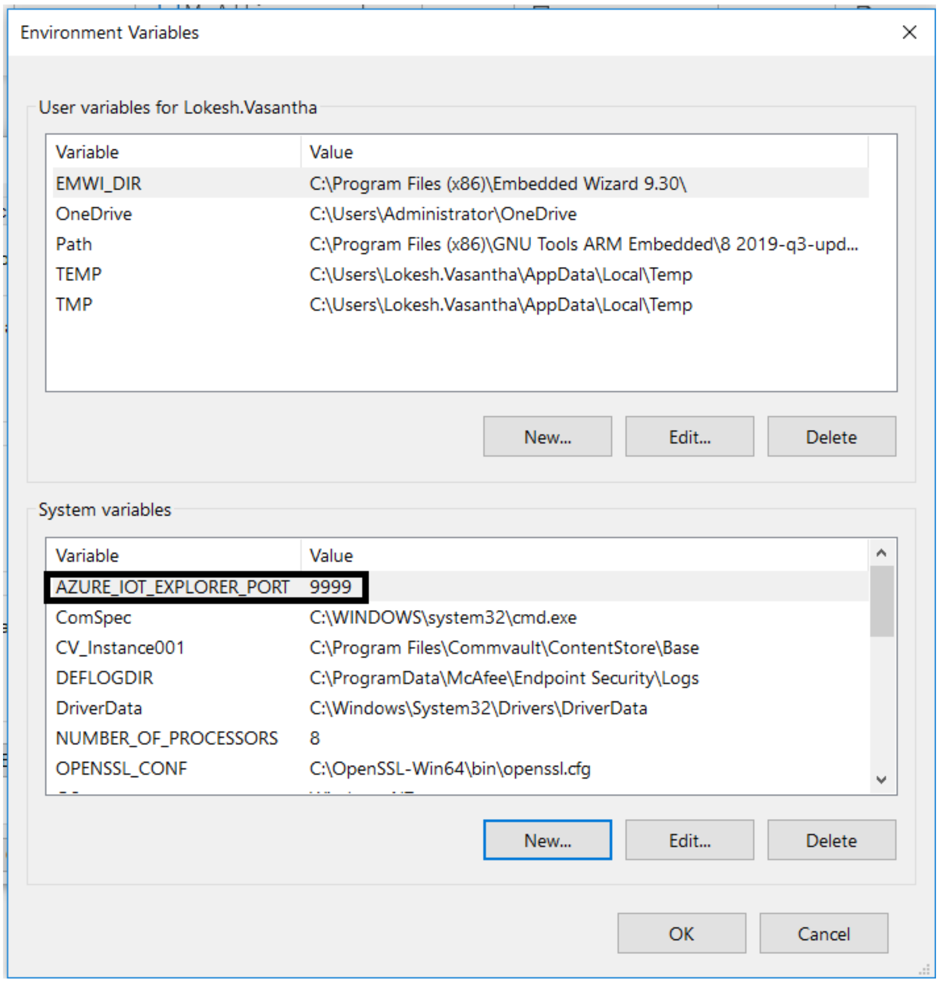
New System Variable

Variable name: AZURE_IOT_EXPLORER_PORT

Variable value: 9999

ComSpec	C:\WINDOWS\system32\cmd.exe
CV_Instance001	C:\Program Files\Commvault\ContentStore\Base
DEFLOGDIR	C:\ProgramData\McAfee\Endpoint Security\Logs
DriverData	C:\Windows\System32\Drivers\DriverData
NUMBER_OF_PROCESSORS	8
OPENSSL_CONF	C:\OpenSSL-Win64\bin\openssl.cfg

Figure 25. Adding System Environment Variable for Alternate Port - Azure IoT Explorer



Environment Variables

User variables for Lokesh.Vasantha

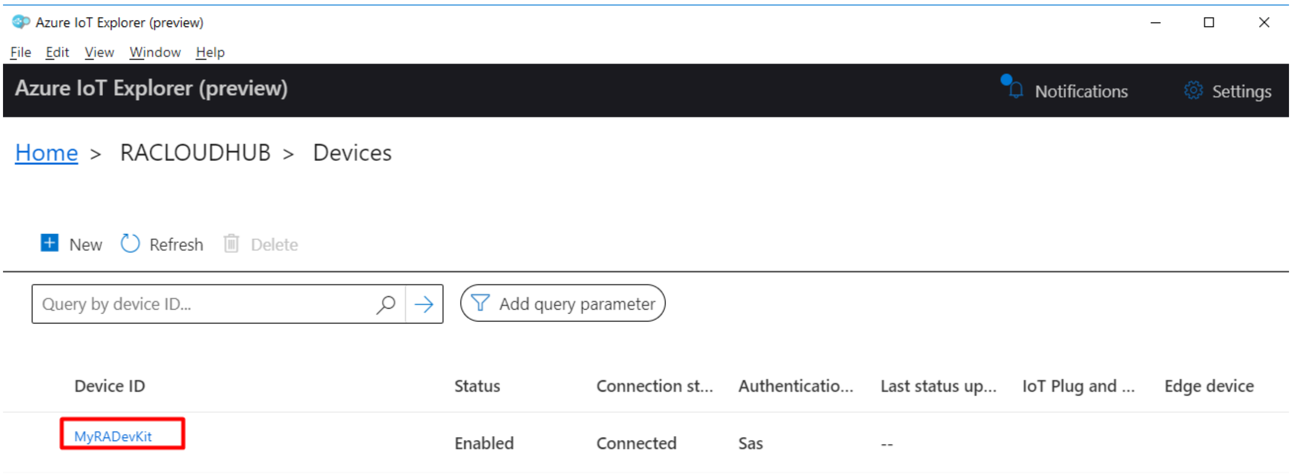
Variable	Value
EMWI_DIR	C:\Program Files (x86)\Embedded Wizard 9.30\
OneDrive	C:\Users\Administrator\OneDrive
Path	C:\Program Files (x86)\GNU Tools ARM Embedded\8 2019-q3-upd...
TEMP	C:\Users\Lokesh.Vasantha\AppData\Local\Temp
TMP	C:\Users\Lokesh.Vasantha\AppData\Local\Temp

System variables

Variable	Value
AZURE_IOT_EXPLORER_PORT	9999
ComSpec	C:\WINDOWS\system32\cmd.exe
CV_Instance001	C:\Program Files\Commvault\ContentStore\Base
DEFLOGDIR	C:\ProgramData\McAfee\Endpoint Security\Logs
DriverData	C:\Windows\System32\Drivers\DriverData
NUMBER_OF_PROCESSORS	8
OPENSSL_CONF	C:\OpenSSL-Win64\bin\openssl.cfg

Figure 26. Added Alternate Port for Azure IoT Explorer

If the connection succeeds, the Azure IoT Explorer switches to a **Devices** view and lists your device.



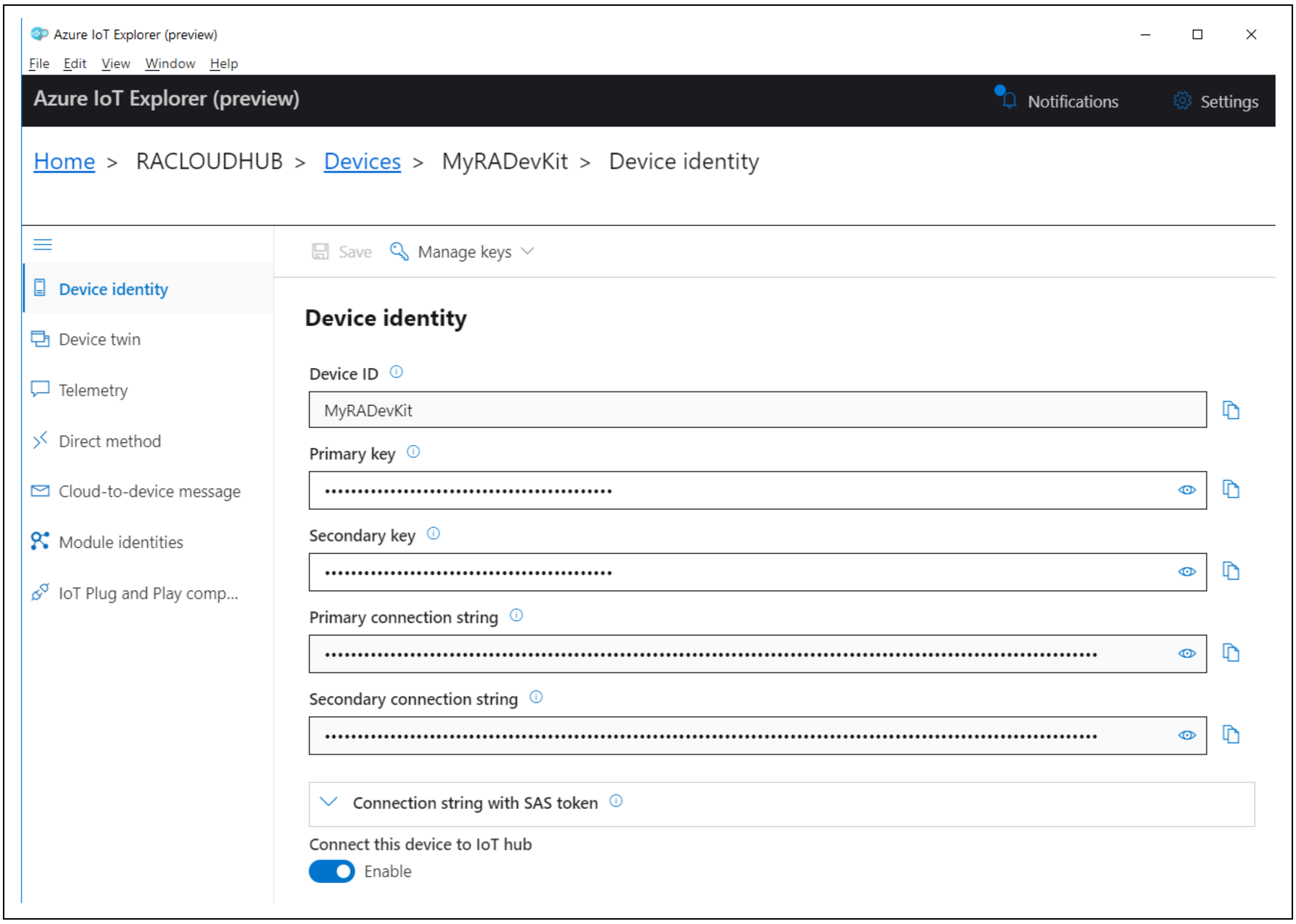
Azure IoT Explorer (preview)
File Edit View Window Help
Azure IoT Explorer (preview)
Notifications
Settings
Home > RACLOUDHUB > Devices
New
Refresh
Delete
Query by device ID...
Add query parameter

Device ID	Status	Connection st...	Authenticatio...	Last status up...	IoT Plug and ...	Edge device
MyRADevKit	Enabled	Connected	Sas	--		

Figure 27. Listed Devices

To view device properties using Azure IoT Explorer:

1. Click the link for your device. IoT Explorer displays details for the device.
2. Inspect the properties for your device in the **Device identity** panel


Azure IoT Explorer (preview)
File Edit View Window Help
Azure IoT Explorer (preview)
Notifications
Settings
Home > RACLOUDHUB > Devices > MyRADevKit > Device identity
Device identity
Device twin
Telemetry
Direct method
Cloud-to-device message
Module identities
IoT Plug and Play comp...
Save
Manage keys

Device identity

Device ID: MyRADevKit

Primary key: ••

Secondary key: ••

Primary connection string: ••

Secondary connection string: ••

Connection string with SAS token

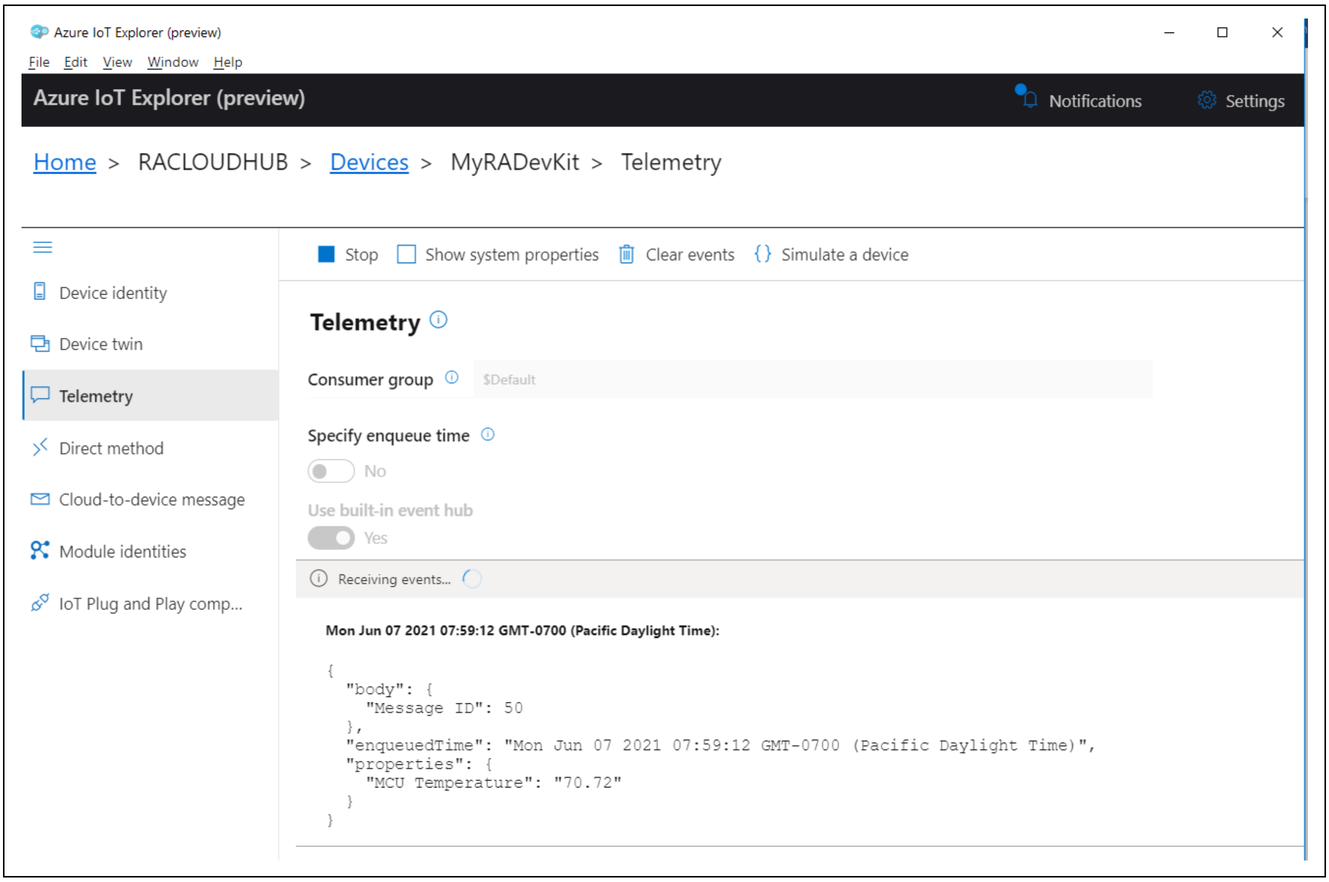
Connect this device to IoT hub: Enable

Figure 28. Device Details

3.11 View Device Telemetry

With Azure IoT Explorer, you can view the flow of telemetry from your device to the cloud. To view telemetry in Azure IoT Explorer:

1. In IoT Explorer select **Telemetry**. Confirm that **Use built-in event hub** is set to **Yes**.
2. Select **Start**.
3. View the telemetry as the device sends messages to the cloud.



Mon Jun 07 2021 07:59:12 GMT-0700 (Pacific Daylight Time):

```
{
  "body": {
    "Message ID": 50
  },
  "enqueuedTime": "Mon Jun 07 2021 07:59:12 GMT-0700 (Pacific Daylight Time)",
  "properties": {
    "MCU Temperature": "70.72"
  }
}
```

Figure 29. Device Telemetry Details

3.12 Send Cloud-to-Device Message

To send a cloud-to-device message in Azure IoT Explorer:

1. In IoT Explorer select **Cloud-to-device message**.
2. Enter the message in the **Message body** = "**LED**", **Key** = **LED**, **Value** = **ON**
3. Check **Add timestamp to message body**.
4. Select **Send message to device**.

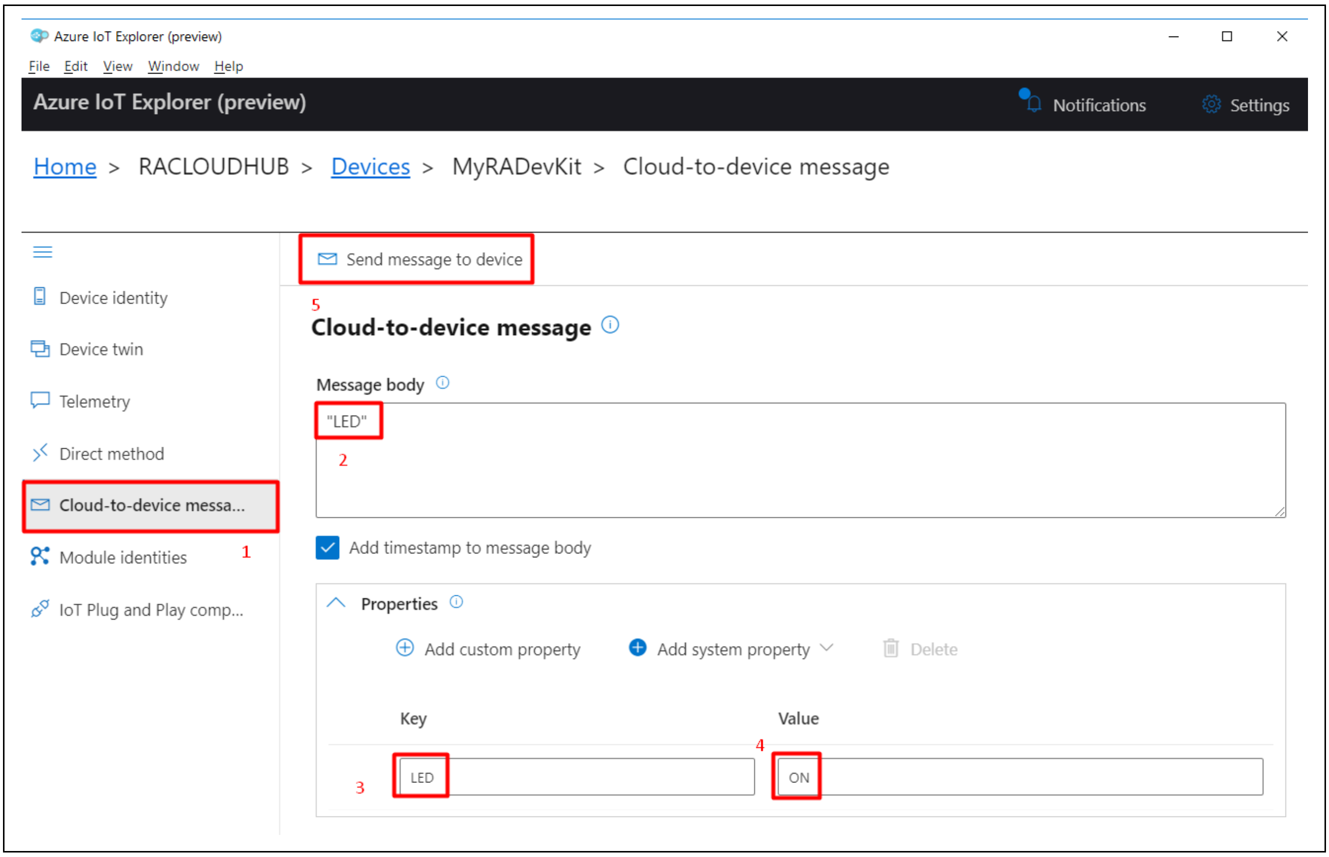
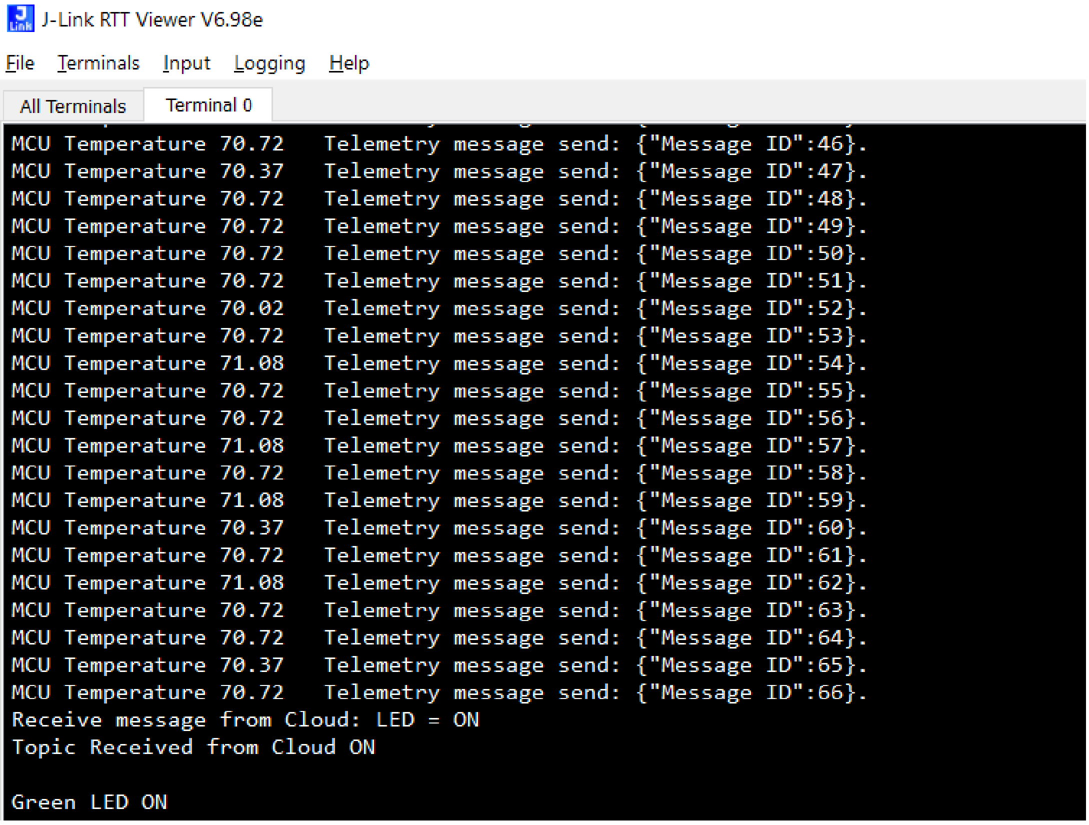


Figure 30. Device Telemetry Details

5. In the terminal window, you can see that the message is received by the IoT Device.

Note: The Cloud-to-device message in this application is sent to turn on the green LED. You can see the green LED on the board is turned ON.



```
MCU Temperature 70.72   Telemetry message send: {"Message ID":46}.
MCU Temperature 70.37   Telemetry message send: {"Message ID":47}.
MCU Temperature 70.72   Telemetry message send: {"Message ID":48}.
MCU Temperature 70.72   Telemetry message send: {"Message ID":49}.
MCU Temperature 70.72   Telemetry message send: {"Message ID":50}.
MCU Temperature 70.72   Telemetry message send: {"Message ID":51}.
MCU Temperature 70.02   Telemetry message send: {"Message ID":52}.
MCU Temperature 70.72   Telemetry message send: {"Message ID":53}.
MCU Temperature 71.08   Telemetry message send: {"Message ID":54}.
MCU Temperature 70.72   Telemetry message send: {"Message ID":55}.
MCU Temperature 70.72   Telemetry message send: {"Message ID":56}.
MCU Temperature 71.08   Telemetry message send: {"Message ID":57}.
MCU Temperature 70.72   Telemetry message send: {"Message ID":58}.
MCU Temperature 71.08   Telemetry message send: {"Message ID":59}.
MCU Temperature 70.37   Telemetry message send: {"Message ID":60}.
MCU Temperature 70.72   Telemetry message send: {"Message ID":61}.
MCU Temperature 71.08   Telemetry message send: {"Message ID":62}.
MCU Temperature 70.72   Telemetry message send: {"Message ID":63}.
MCU Temperature 70.72   Telemetry message send: {"Message ID":64}.
MCU Temperature 70.37   Telemetry message send: {"Message ID":65}.
MCU Temperature 70.72   Telemetry message send: {"Message ID":66}.
Receive message from Cloud: LED = ON
Topic Received from Cloud ON

Green LED ON
```

Figure 31. Console Output

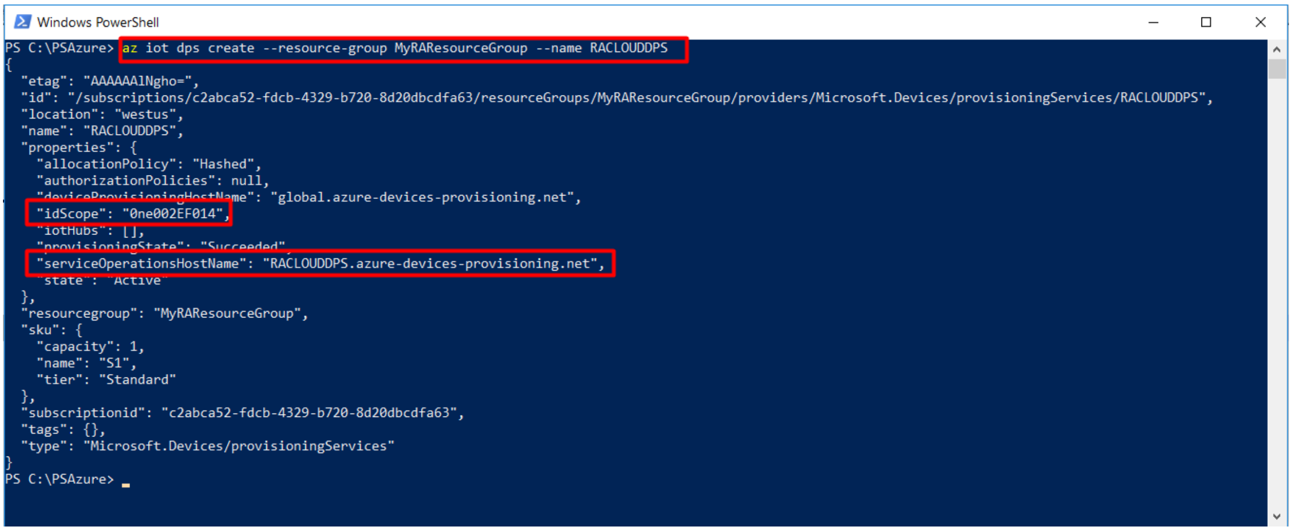
3.13 Use Device Provisioning Service (DPS)

The IoT Hub Device Provisioning Service (DPS) is a helper service for IoT Hub that enables zero-touch, just-in-time provisioning devices to the right IoT hub in a secure and scalable manner. In the following steps, you will enroll the board in DPS using Symmetric Key and provision it automatically in IoT Hub when connecting to the Internet.

Create a Device Provisioning Service (DPS)

You can use Azure CLI to create a DPS to provision the device in IoT Hub automatically.

1. Run the `az iot dps create` command to create a DPS. It might take a few minutes to create it. Replace the *`YourDPSName`* placeholder below with the name you chose for your DPS. An IoT hub name must be globally unique in Azure. This placeholder is used in the rest of this tutorial to represent your unique DPS name.
   - `az iot dps create --resource-group MyRAResourceGroup --name {YourDPSName}`



```
PS C:\PSAzure> az iot dps create --resource-group MyRAResourceGroup --name RACLOUDDPS
{
  "etag": "AAAAAA1Ngho=",
  "id": "/subscriptions/c2abca52-fdcb-4329-b720-8d20dbcdfa63/resourceGroups/MyRAResourceGroup/providers/Microsoft.Devices/provisioningServices/RACLOUDDPS",
  "location": "westus",
  "name": "RACLOUDDPS",
  "properties": {
    "allocationPolicy": "Hashed",
    "authorizationPolicies": null,
    "deviceProvisioningHostName": "global.azure-devices-provisioning.net",
    "idScope": "0ne002EF014",
    "iotHubs": [],
    "provisioningState": "Succeeded",
    "serviceOperationsHostName": "RACLOUDDPS.azure-devices-provisioning.net",
    "state": "Active"
  },
  "resourcegroup": "MyRAResourceGroup",
  "sku": {
    "capacity": 1,
    "name": "S1",
    "tier": "Standard"
  },
  "subscriptionid": "c2abca52-fdcb-4329-b720-8d20dbcdfa63",
  "tags": {},
  "type": "Microsoft.Devices/provisioningServices"
}
PS C:\PSAzure>
```

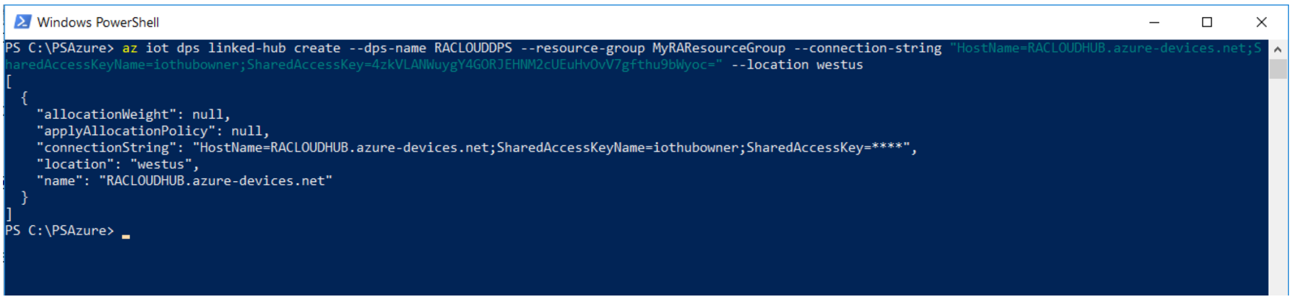
Figure 32. DPS Creation JSON Output

2. After the DPS is created, view the JSON output in the console, and copy the `serviceOperationsHostName` and `idScope` values to a safe place. You use this value in a later step. The `serviceOperationsHostName` and `idScope` values looks like the following example:
   `serviceOperationsHostName`: `{Your DPS name}.azure-devices-provisioning.net`
   `idScope`: 0nexxxxxxxx
3. You can also run the `az iot dps show` command to view the values again:
   - `az iot dps show --resource-group MyRAResourceGroup --name {YourDPSName}`

3.14 Link an IoT Hub for DPS

To make the DPS provision the device in IoT Hub, you need to link an IoT Hub for it.

1. Run the `az iot hub show-connection-string` command to get the IoT Hub connection string:
   - `az iot hub show-connection-string --name {YourIoTHubName}`
2. Run the `az iot dps linked-hub create` command to create a linked IoT Hub in DPS, and replace the `YourIoTHubConnectionString` with the actual one you get:
   - `az iot dps linked-hub create --dps-name {YourDPSName} --resource-group MyRAResourceGroup --connection-string {YourIoTHubConnectionString} --location westus`



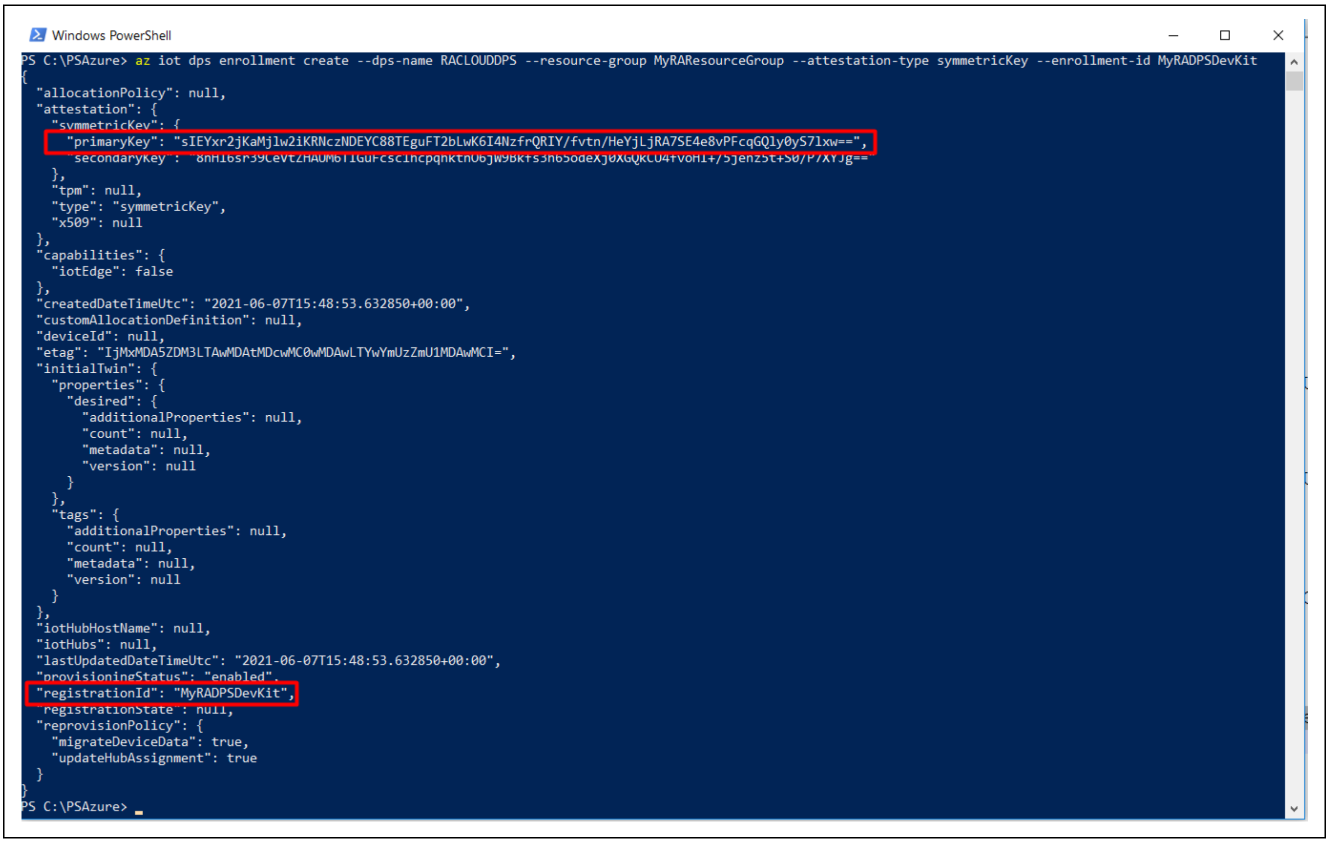
```
PS C:\PSAzure> az iot dps linked-hub create --dps-name RACLOUDDPS --resource-group MyRAResourceGroup --connection-string "HostName=RACLOUDHUB.azure-devices.net;SharedAccessKeyName=iothubowner;SharedAccessKey=4zkVLANWuygY4GORJEHNM2cUEuHvOvV7gfthu9bWyoc=" --location westus
[
  {
    "allocationWeight": null,
    "applyAllocationPolicy": null,
    "connectionString": "HostName=RACLOUDHUB.azure-devices.net;SharedAccessKeyName=iothubowner;SharedAccessKey=****",
    "location": "westus",
    "name": "RACLOUDHUB.azure-devices.net"
  }
]
PS C:\PSAzure>
```

Figure 33. Linking IoT HUB to DPS

3.15 Add Enrollment in DPS

Now you need to add an individual enrollment record in DPS which your device can use later to connect to DPS and perform the provisioning in IoT Hub.

1. Run the `az iot dps enrollment create` command to create a device enrollment in DPS:
   - `az iot dps enrollment create --dps-name {YourDPSName} --resource-group MyRAResourceGroup --attestation-type symmetricKey --enrollment-id {MyDPSDevKit}`



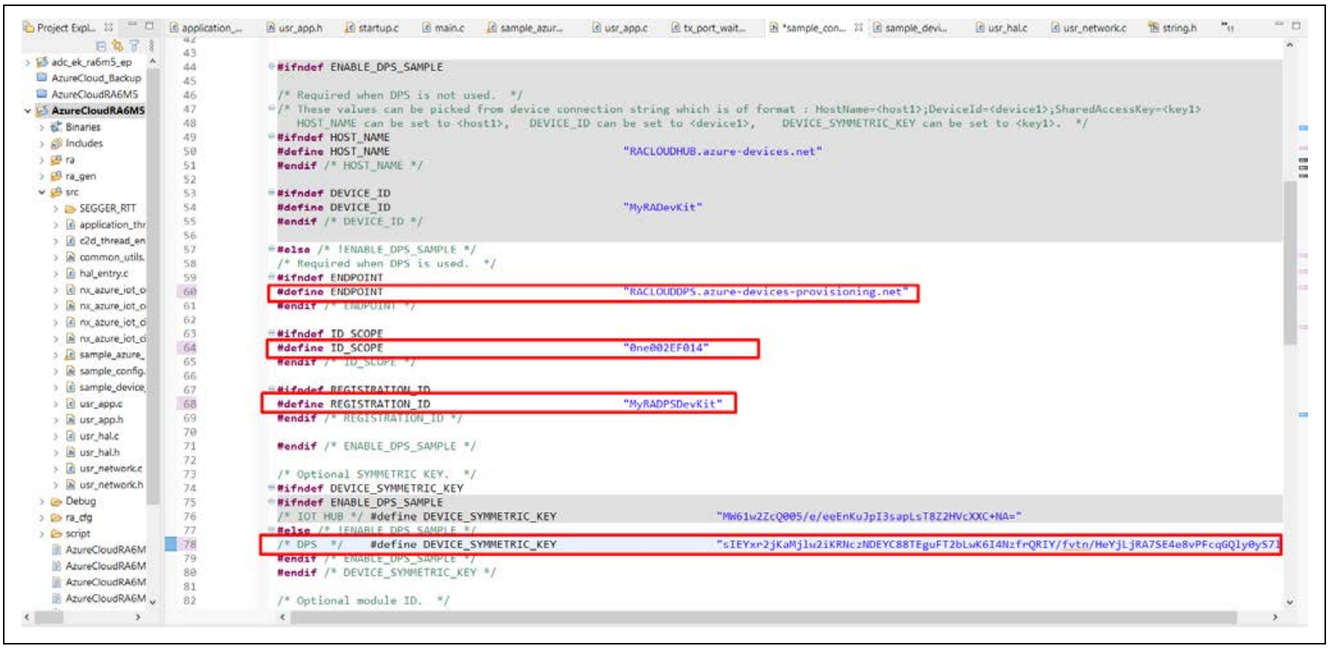
```
PS C:\PSAzure> az iot dps enrollment create --dps-name RACLOUDDPS --resource-group MyRAResourceGroup --attestation-type symmetricKey --enrollment-id MyRADPSDevKit
{
  "allocationPolicy": null,
  "attestation": {
    "symmetricKey": {
      "primaryKey": "sIEYxr2jKaMjlw2iKRNczNDEYC88TEguFT2bLwK6I4NzfrQRIY/fvtn/HeYjLjRA7SE4e8vPFcqGQly0yS7lxw==",
      "secondaryKey": "8nHI6sr39CeVtZHAUM6TIGuFcsclncpqnktnO6jW9Bkfs3h65odeXj0XGQkCU4fvoHI+/5jenz5t+S0/P7XYJg=="
    },
    "tpm": null,
    "type": "symmetricKey",
    "x509": null
  },
  "capabilities": {
    "iotEdge": false
  },
  "createdDateTimeUtc": "2021-06-07T15:48:53.632850+00:00",
  "customAllocationDefinition": null,
  "deviceId": null,
  "etag": "IjMxMDA5ZDM3LTAwMDAtMDcwMC0wMDAwLTYwYmUzZmU1MDAwMCI=",
  "initialTwin": {
    "properties": {
      "desired": {
        "additionalProperties": null,
        "count": null,
        "metadata": null,
        "version": null
      }
    },
    "tags": {
      "additionalProperties": null,
      "count": null,
      "metadata": null,
      "version": null
    }
  },
  "iotHubHostName": null,
  "iotHubs": null,
  "lastUpdatedDateTimeUtc": "2021-06-07T15:48:53.632850+00:00",
  "provisioningStatus": "enabled",
  "registrationId": "MyRADPSDevKit",
  "registrationState": null,
  "reprovisionPolicy": {
    "migrateDeviceData": true,
    "updateHubAssignment": true
  }
}
PS C:\PSAzure> _
```

Figure 34. DPS Enrollment Creation

2. After the device is created, view the JSON output in the console, and copy the `registrationId` and `primaryKey` values to use in a later step.
3. Confirm that you have copied the following values from the JSON output from previous steps to use in the next section:
   - `serviceOperationsHostName`
   - `idScope`
   - `registrationId`
   - `primaryKey`

Add configuration:

1. Open `sample_config.h` and enable the DPS by uncommenting the line `#define ENABLE_DPS_SAMPLE` and make the changes to the configuration as shown in the snapshot with your `serviceOperationsHostName`, `idScope`, `registrationId`, and `PrimaryKey`.



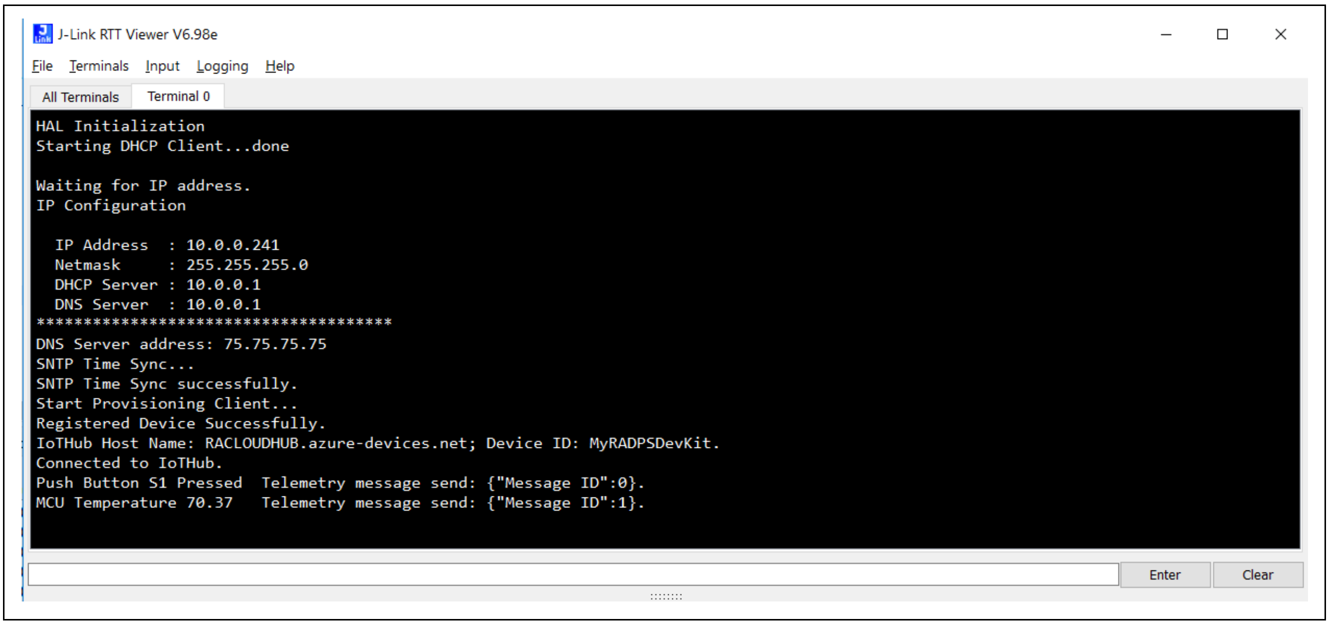
```
#ifndef ENABLE_DPS_SAMPLE

/* Required when DPS is not used.  */
/* These values can be picked from device connection string which is of format : HostName=<host1>;DeviceId=<device1>;SharedAccessKey=<key1>
   HOST_NAME can be set to <host1>,   DEVICE_ID can be set to <device1>,   DEVICE_SYMMETRIC_KEY can be set to <key1>.  */
#ifndef HOST_NAME
#define HOST_NAME                                "RACLOUDHUB.azure-devices.net"
#endif /* HOST_NAME */

#ifndef DEVICE_ID
#define DEVICE_ID                                "MyRADevKit"
#endif /* DEVICE_ID */

#else /* !ENABLE_DPS_SAMPLE */
/* Required when DPS is used.  */
#ifndef ENDPOINT
#define ENDPOINT                                 "RACLOUDDPS.azure-devices-provisioning.net"
#endif /* ENDPOINT */

#ifndef ID_SCOPE
#define ID_SCOPE                                 "0ne002EF014"
#endif /* ID_SCOPE */

#ifndef REGISTRATION_ID
#define REGISTRATION_ID                          "MyRADPSDevKit"
#endif /* REGISTRATION_ID */

#endif /* ENABLE_DPS_SAMPLE */

/* Optional SYMMETRIC KEY.  */
#ifndef DEVICE_SYMMETRIC_KEY
#ifndef ENABLE_DPS_SAMPLE
/* IOT HUB */ #define DEVICE_SYMMETRIC_KEY                "MW61w2ZcQ005/e/eeEnKuJpI3sapLsT8Z2HVcXXC+NA="
#else /* !ENABLE_DPS_SAMPLE */
/* DPS  */    #define DEVICE_SYMMETRIC_KEY                "sIEYxr2jKaMjlw2iKRNczNDEYC88TEguFT2bLwK6I4NzfrQRIY/fvtn/HeYjLjRA7SE4e8vPFcqGQly0yS7l
#endif /* ENABLE_DPS_SAMPLE */
#endif /* DEVICE_SYMMETRIC_KEY */

/* Optional module ID.  */
```

Figure 35. Configuration Changes to sample_config.h

Constant name	Value
ENDPOINT	{Your serviceOperationsHostName value}
ID_SCOPE	{Your idScope value}
REGISTRATION_ID	{Your registrationId value}
DEVICE_SYMMETRIC_KEY	{Your primaryKey value}

2. Build and run the project. As the project runs, the demo prints out status information to the terminal output window. The demo application successfully provisions the device using the DPS and publishes the MCU Temperature message to IoT Hub every 30 seconds. Check the terminal output to verify that messages have been successfully sent to the Azure IoT hub.

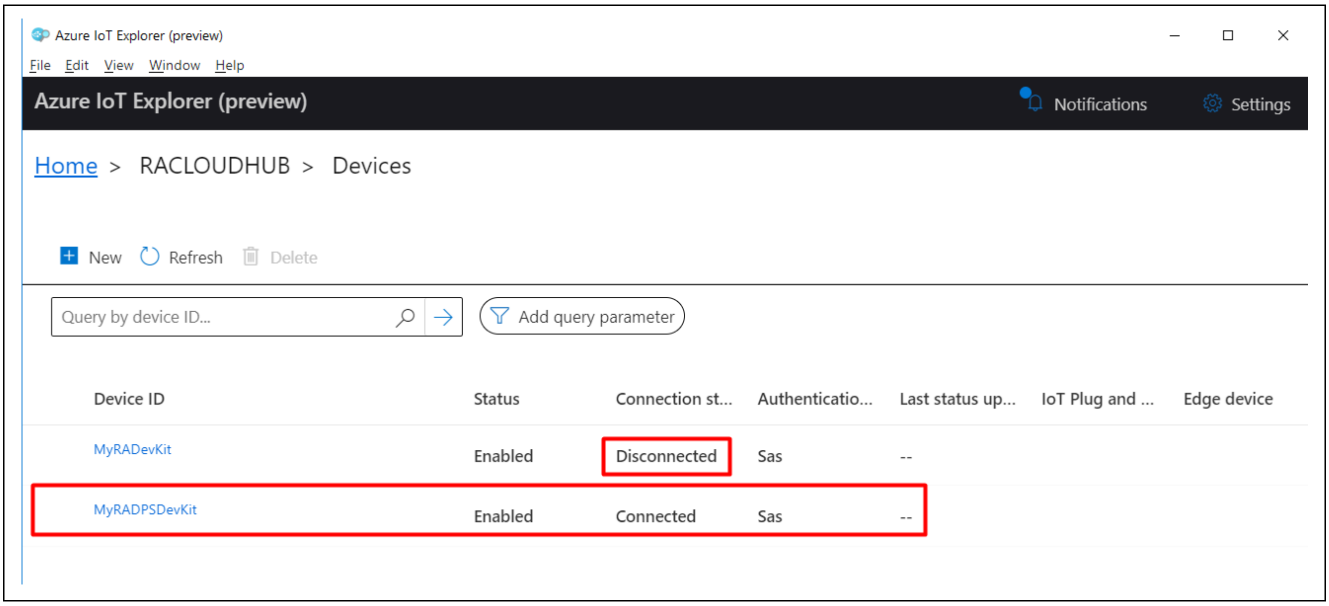
Note: Press the pushbutton S1 or S2 to send the asynchronous data to the cloud.



```
HAL Initialization
Starting DHCP Client...done

Waiting for IP address.
IP Configuration

  IP Address  : 10.0.0.241
  Netmask     : 255.255.255.0
  DHCP Server : 10.0.0.1
  DNS Server  : 10.0.0.1
**************************************
DNS Server address: 75.75.75.75
SNTP Time Sync...
SNTP Time Sync successfully.
Start Provisioning Client...
Registered Device Successfully.
IoTHub Host Name: RACLOUDHUB.azure-devices.net; Device ID: MyRADPSDevKit.
Connected to IoTHub.
Push Button S1 Pressed  Telemetry message send: {"Message ID":0}.
MCU Temperature 70.37   Telemetry message send: {"Message ID":1}.
```

Figure 36. RTT Terminal Output

Now the Device can be seen on the IoT Explorer. You can click on the newly created DPS ID and verify the Device Telemetry and Cloud-to-device messaging as explained in the sections 3.11 and 3.12


Azure IoT Explorer (preview)
File Edit View Window Help
Azure IoT Explorer (preview)
Notifications
Settings
Home > RACLOUDHUB > Devices
New
Refresh
Delete
Query by device ID...
Add query parameter

Device ID	Status	Connection st...	Authenticatio...	Last status up...	IoT Plug and ...	Edge device
MyRADevKit	Enabled	Disconnected	Sas	--		
MyRADPSDevKit	Enabled	Connected	Sas	--		

Figure 37. DPS Config on the IoT Explorer

Note: Users can verify the communication from the Device to the Cloud and Cloud to the Device as well.

4. Importing, Building and Loading the Project

For a quick validation of this application project, import and build the project. The following steps show how to import, build, and download the project.

Note: To run the application project successfully and to communicate to the cloud, follow the instructions for setting up the cloud interface as described in section 3, which details making changes to the credentials and creating your own cloud devices, running and validating the application.

4.1 Importing

The application project bundled as part of this app note can be imported into e^2 studio using instructions provided in the *RA FSP User's Manual*. See Section *Starting Development > e2 studio ISDE User Guide > Importing an Existing Project into e2 studio ISDE.*

4.2 Building the Latest Executable Binary

Upon successfully importing and/or modifying the project into e^2 studio IDE, follow instructions provided in the *RA FSP User's Manual* to build an executable binary/hex/mot/elf file. See Section *Starting Development > e2 studio ISDE User Guide > Tutorial: Your First RA MCU Project > Build the Blinky Project.*

4.3 Loading the Executable Binary into the Target MCU

The executable file may be programmed into the target MCU through any one of three means.

4.3.1 Using a Debugging Interface with e^2 studio

Instructions to program the executable binary are found in the latest *RA FSP User Manual*. See Section *Starting Development > e2 studio ISDE User Guide > Tutorial: Your First RA MCU Project > Debug the Blinky Project.*

This is the preferred method for programming as it allows for additional debugging functionality available through the on-chip debugger.

4.3.2 Using J-Link Tools

SEGGER J-Link Tools such as J-Flash, J-Flash Lite, and J-Link Commander can be used program the executable binary into the target MCU. Refer User Manuals UM08001, and UM08003 on www.segger.com.

4.3.3 Using Renesas Flash Programmer

The Renesas Flash Programmer provides usable and functional support for programming the on-chip flash memory of Renesas microcontrollers in each phase of development and mass production. The software supports all RA MCUs and the software user's manual is available online (Renesas Electronics Corporation, n.d.).

5. Next Steps and References

- Refer to the following GitHub repository for various FSP modules example projects and application projects (https://github.com/renesas/ra-fsp-examples/)
- Refer to *Establishing and Protecting Device Identity using SCE7 and Security MPU* (R11AN0449) on renesas.com
- Refer to *Securing Data at Rest Utilizing the RA Security MPU* (R11AN0416) on renesas.com
- Refer to Azure GitHub link for more details on Azure SDK for Embedded C (https://github.com/Azure/azure-sdk-for-c)

6. MQTT/TLS References

- *FSP v3.0.0 User's Manual* (www.renesas.com/RA/FSP).
- Azure IoT documentation https://docs.microsoft.com/en-us/azure/iot-hub/

7. Known Issues and Limitations

1. Occasional outages in cloud connectivity may be noticed during the demonstration due to changes in the cloud server. Contact the Renesas support team for questions.
2. Currently, there is no support for direct device-to-device communications with Azure IoT Hub.
3. Device will reconnect after 65 minutes due to SAS token refresh. Currently it is under SDK control. Users need to know this when developing the application.

Website and Support

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

RA Product Information	www.renesas.com/ra
RA Product Support Forum	www.renesas.com/ra/forum
RA Flexible Software Package	www.renesas.com/FSP
Renesas Support	www.renesas.com/support

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Jul.19.21	—	Initial version
1.02	Oct.27.21	—	Migrated to FSP 3.3.0 which has the full-fledged Configurator support.
1.03	May. 23.22	—	Updated to FSP 3.7.0

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/.