RA6W1/RRQ61001 AWS IoT Core

The RA6W1/RRQ61001 is a highly integrated ultra-low-power Wi-Fi® MCU that allows you to develop a complete Wi-Fi solution on a single chip. This document is a RA6W1/RRQ61001 manual intended to help new or existing developers quickly get started using AWS® IoT Core.

Contents

Figures

Tables

Terms and Definitions

ADC	Analog-to-Digital Converter
AP	Access Point
AWS	Amazon Web Services
CA	Certificate Authority
DPM	Device Power Management
eMMC	Embedded MultiMediaCard
EVB	Evaluation Board
FreeRTOS	Free Real-Time Operating System
ID	Identifier
IDE	Integrated Development Environment
IoT	Internet of Things
IP	Internet Protocol
MCU	Microcontroller Unit
MQTT	Message Queuing Telemetry Transport
MUX	Multiplexer
NVRAM	Non-Volatile Random-Access Memory
OS	Operating System
OTA	Over-the-Air
RF	Radio Frequency
RTC	Real-Time Clock
SD	Secure Digital
SDK	Software Development Kit
TCP	Transmission Control Protocol
UART	Universal Asynchronous Receiver/Transmitter
UI	User Interface
Wi-Fi	Wireless Fidelity

1. Introduction

The RA6W1 Wi-Fi Development Kit provides a quick and easy method to start developing battery powered applications and products using ultra-low power Wi-Fi. The ultra-low-power RA6W1 MCU is the world's lowest power Wi-Fi chip specifically designed to meet the requirements for power sensitive wireless applications.

Benefits

- Ultra-low-power Wi-Fi technology
- More than 1-year battery life for most applications
- Industry leading wireless range
- Fully-integrated Wi-Fi MCU (Microcontroller Unit)
- Comprehensive security capabilities
- Easy-to-use development tools means shorter time to market.

Features

- Highly-integrated ultra-low-power RA6W1 Wi-Fi system module
- Best Radio Frequency performance
- MCU runs full networking OS and TCP/IP stack
- Built-in 4-channel auxiliary ADC for sensor interfaces
- Built-in hardware crypto engines for advanced security features
- Complete software stack
- eMMC/SD expanded memory.

RA6W1 is a full offload MCU for IoT Applications, such as:

- Security systems
- Door locks
- Thermostats
- Garage door openers
- Blinds
- Lighting control
- Sprinkler systems
- Video camera security systems
- Smart appliances
- Video doorbell.

2. Hardware Description

The RA6W1 is a fully-integrated Wi-Fi MCU with an ultra-low power consumption, best RF performance, and easy development environment. For more details, see *RA6W1 Datasheet*.

3. Set Up Development Environment

AWS IoT applications can be developed for the RA6W1 using the RA6W1 FreeRTOS Software Development Kit (SDK) and the Renesas Electronics e^2 studio IDE on either a Windows 10 or Linux-based development system.

To start the software development environment:

1. Install and configure the e^2 studio IDE.
2. Import the RA6W1 SDK into the e^2 studio and build an application.
3. Set up an evaluation board.
4. Download and test the application.
5. Use J-Link debugger to debug the application.

For more information, see the Set Up RA6W1 EVK and the Software Development Kit and the Build Setup sections of *RA6W1 Application Development Guide*.

4. Build and Run Reference Application

To get started with AWS IoT connectivity on the RA6W1 device, begin with a ready-to-use reference application. The provided application project, `awsiot_wifi_lock_ek_ra6w1`, includes pre-configured project settings, middleware, and example code, which helps to save time and ensures that the environment is correctly set up for AWS IoT development. This project shows how you can create an application that implements a cloud-connected door lock system in RA6W1 using the AWS IoT platform. The RA6W1 device connects to a Wi-Fi network and establishes a secure MQTT connection to AWS IoT Core. The actual door lock hardware is expected to be controlled by RA6W1. The mobile application communicates with AWS IoT Core through the internet to remotely control the door lock through RA6W1.

Figure 1 shows the components required to run the AWS IoT reference application in RA6W1 through an Internet connection and AWS IoT server:

- RA6W1 EVK.
- Router: Connection to internet.
- Mobile device: Android/iOS application.
- AWS account.
- AWS IoT reference application package.

Figure 1. Architecture of AWS IoT reference application

4.1 Load AWS-IoT Reference Application in e^2 studio

The project template can be downloaded as a zip file, `r19an0426ee0100-rafw-group-aws-aws.zip`, from the Software and Tools section on the Renesas RA6W1 product page.

For more information on how to load the template, see the Load Project Template in e^2 studio section in *RA6W1 Application Development Guide*.

In the RA6W1 SDK, you can configure the following parameters of AWS IoT core which uniquely identify the AWS IoT account and the device itself:

- Edit Endpoint or Host Name
- Edit Thing Name
- Edit Root CA, Client Certificate, and Client Private key

4.1.1 Edit Endpoint or Host Name

To change the AWS IoT Host Name:

1. In `rm_awsiot_w stack`, go to **Stacks** > **Properties**.
2. In **AWSIOT_W Host Name**, enter the new value.
3. Click **Generate Project Content in Stacks**. This updates the macro AWS_IOT_HOST_NAME in the `rm_awsiot_w_cfg.h` file.

The macro is the following:

```
#define AWS_IOT_HOST_NAME     "(account-specific-prefix).iot.(aws-region).amazonaws.com
```

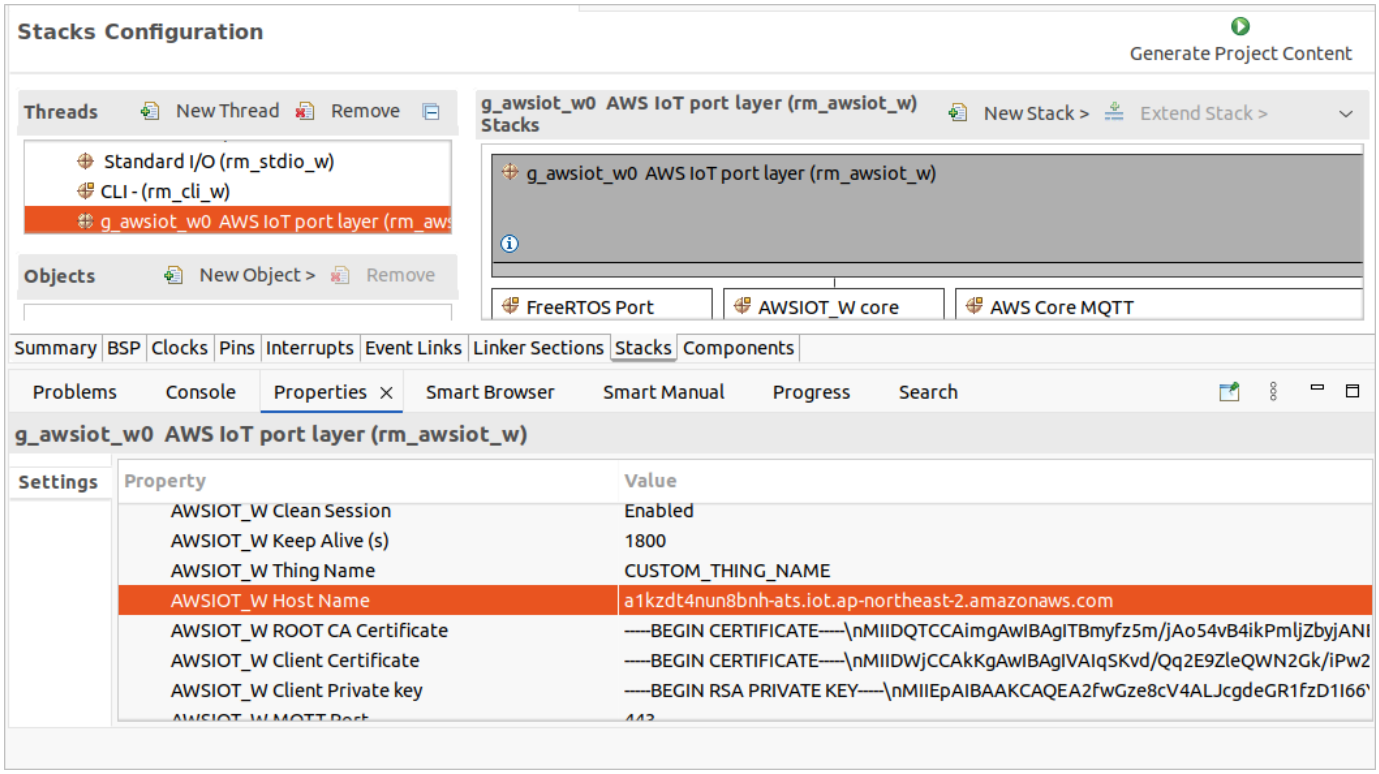

Figure 2. AWS IoT host name change in e² studio

4.1.2 Edit Thing Name

If the Thing Name does not exist in NVRAM, the system uses the predefined name from the configuration header file. To change the AWS IoT Thing Name:

1. In `rm_awsiot_w stack`, go to **Stacks** > **Properties**.
2. In **AWSIOT_W Thing Name**, enter the new value.
3. Click **Generate Project Content**. This updates the macro AWS_IOT_THING_NAME in the `rm_awsiot_w_cfg.h` file, see Figure 3.

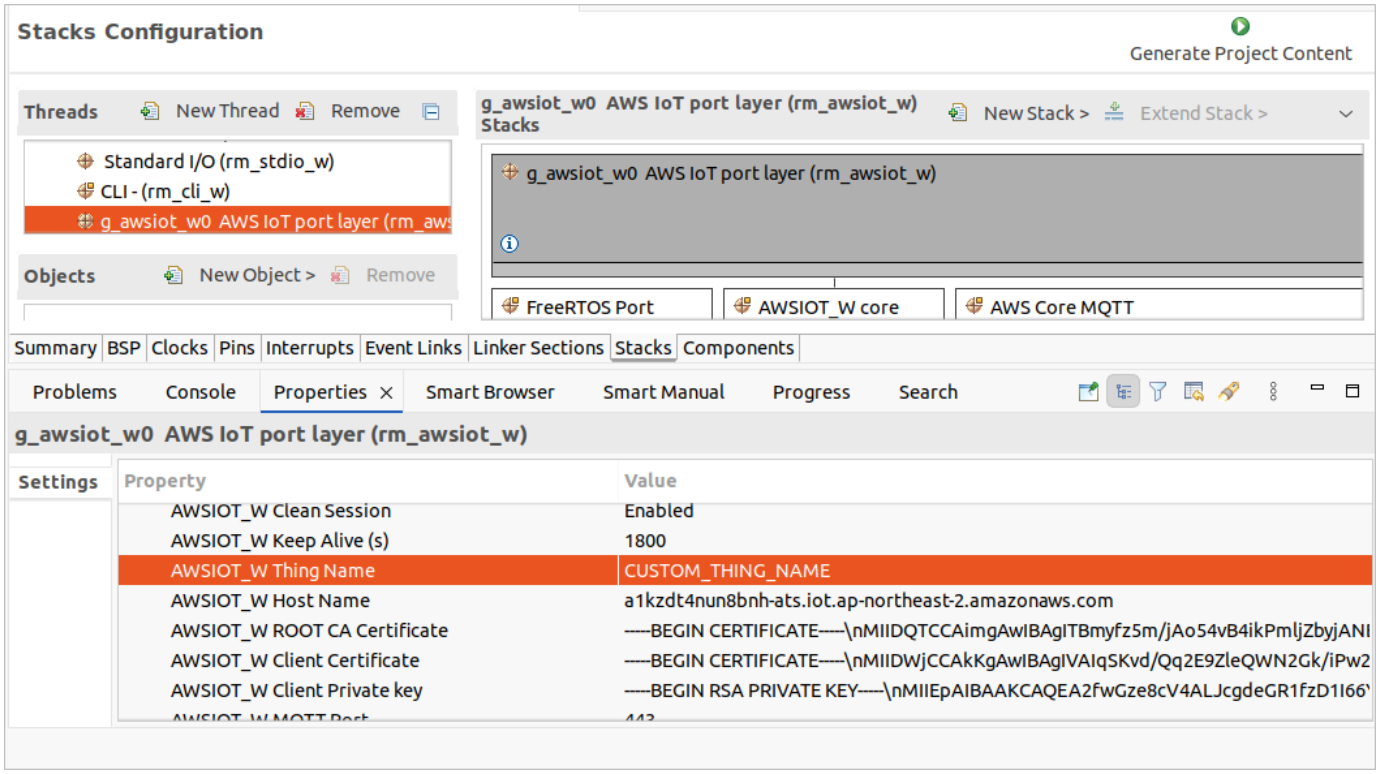

Figure 3. AWS IoT Thing Name change in e² studio

4.1.3 Edit Root CA, Certificate, and Private Key

To authenticate the device with AWS IoT, the device must contain the Root CA, Client Certificate, and Client Private key. For more details, see https://docs.aws.amazon.com/iot/latest/developerguide/iot-security-identity.html.
To update AWSIOT_W ROOT CA Certificate, AWSIOT_W Client Certificate, and AWSIOT_W Client Private key:

NOTE
The Root CA, Client Certificate, and Private key for the reference application are already populated and do not require an update.

1. In `rm_awsiot_w stack`, go to **Stacks** > **Properties**.
2. in **Stacks**, click **Generate Project Content**. This updates the corresponding macros in the `rm_awsiot_w_cfg.h` file, see Figure 4.

```
#define #define AWS_IOT_ROOT_CA     "-----BEGIN CERTIFICATE-----
\nMIIDQTCCAimgAwIBAgITBmyfz5m.....rqXRfboQnoZsG4q5WTP468SQvvG5\n-----END CERTIFICATE-----\n"
#define AWS_IOT_CLIENT_CERT         "-----BEGIN CERTIFICATE-----
\nMIIDWjCCAkKgAwIBAgIVAIqSKvd.....CO4wes8RH5pNIOK2QrKgr9NJkA==\n-----END CERTIFICATE-----\n"
#define AWS_IOT_CLIENT_PRIV_KEY      ""-----BEGIN RSA PRIVATE KEY-----
\nMIIEpAIBAAKCAQEA2fwGze8cV4A.....OcWdbf54nhzcEqjRv1DhCA==\n-----END RSA PRIVATE KEY-----\n"
```

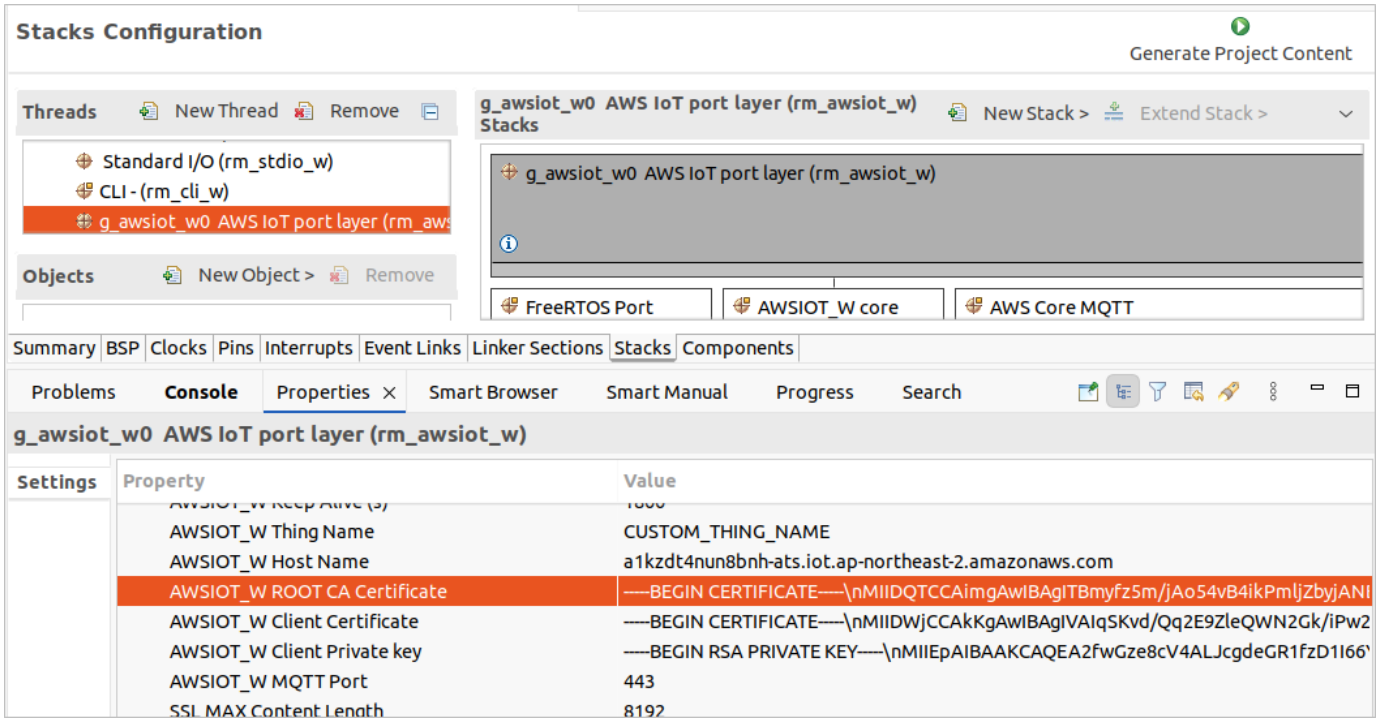

Figure 4. Root CA, Client Certificate, and Client Private key change in e^2 studio

4.2 Build and Flash the Image

To flash the image, see the Software Development Kit and Build Set Up sections of *RA6W1 Application Development Guide*.

4.3 Provision Wi-Fi in RA6W1

Provisioning a Wi-Fi device is the process of setting the device up to connect to a network and function properly. This involves configuring network settings, ensuring secure authentication, and assigning necessary parameters. The provisioning can be done with the Renesas Wi-Fi Provisioning app on either an Android or iOS device.
When RA6W1 initially starts after flashing the image, it is in Station mode.
To change the device into Provisioning mode and to do the Wi-Fi provisioning:

1. Configure factory reset button, see Figure 5.
   If you want to use the BTN1 button for factory reset, connect the pins P0_10 and BTN1 together using a jumper wire.

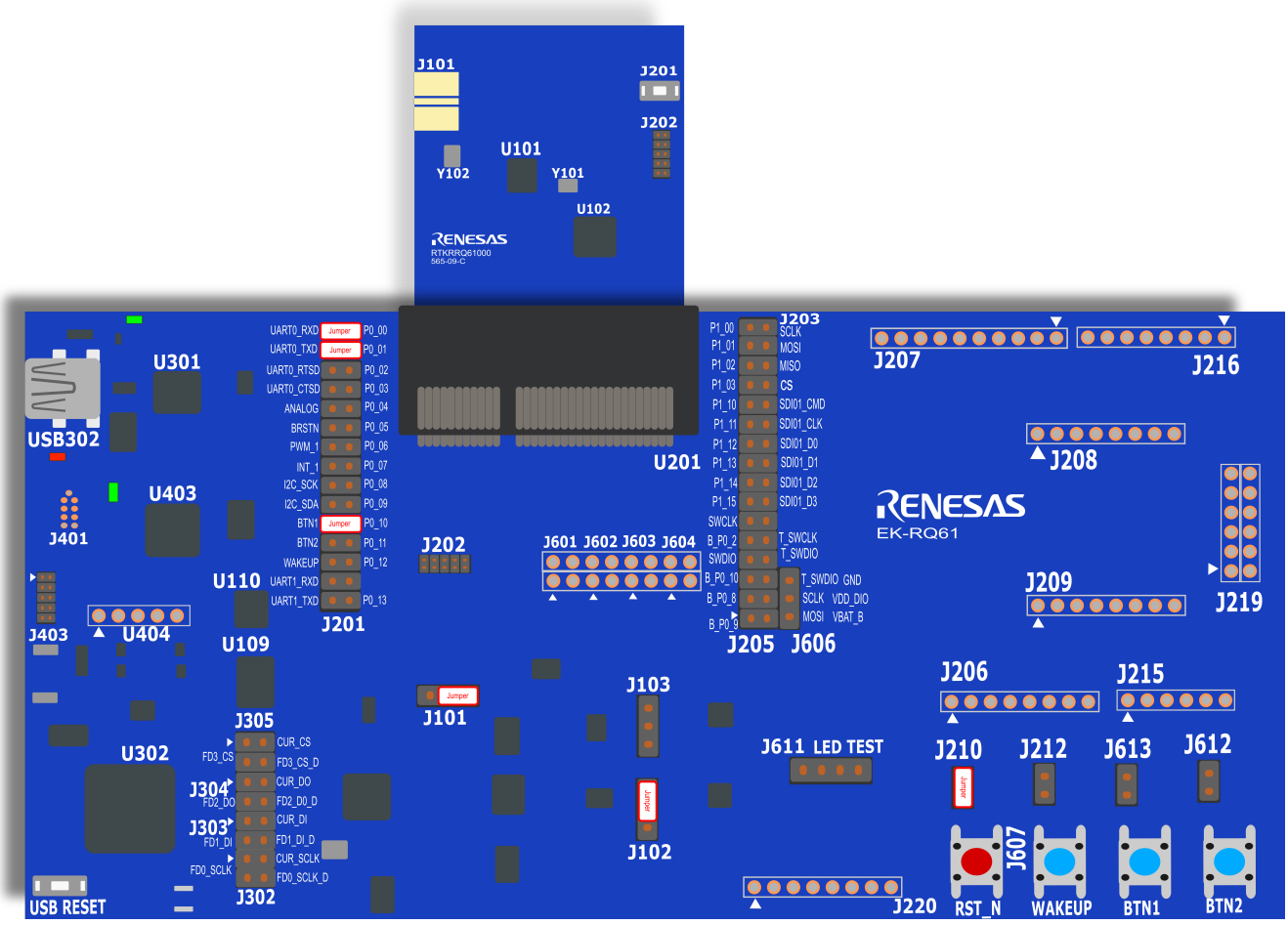

Figure 5. Configure factory reset button in the EVK for Wi-Fi Provisioning mode

2. Press the configured factory reset button using the jumper connection. Now the device does the factory reset and booting in Soft-AP mode and is ready for Wi-Fi provisioning.
4. Download the Renesas Wi-Fi Provisioning application from the Google Play Store or the Apple App Store to connect RA6W1 to Access Point. For Android, see Figure 6.
5. If the location access permissions are asked when you open the application for the first time, open the application and tap **While using the app**.
6. Select **Start DA16200/RRQ61000-based**.
7. Tap **I'm ready** and tap **Connect**.
8. When the application connects to RA6W1, tap **NEXT**.
   Now the application lists all the access points that RA6W1 scanned.
9. Select the needed access point and enter the password.
   This writes the Wi-Fi network details to the device.
10. In the success dialog, tap **OK**.

Wi-Fi provisioning is now completed and RA6W1 reboots and starts as a Station and connects to the access point configured from the application. After the successful connection to the internet, it connects to the AWS server.

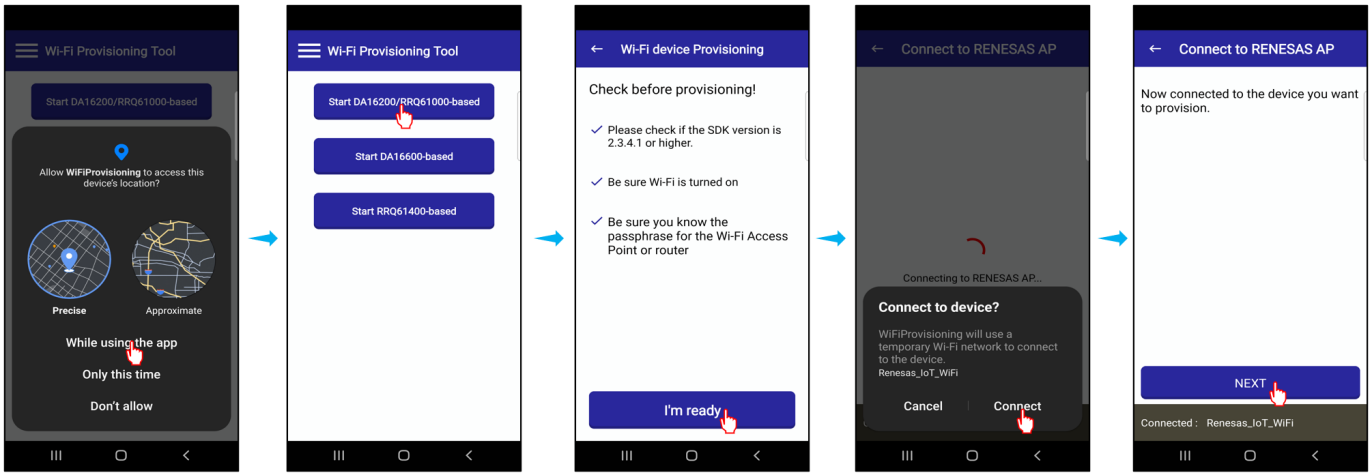

Figure 6. Renesas Wi-Fi provisioning application steps

4.4 Reference Application in RA6W1

When provisioning is completed, to open AWS application on mobile device, select **AWS IoT** by tapping the menu icon, see Figure 7 For details, see Section 6 AWS IoT.

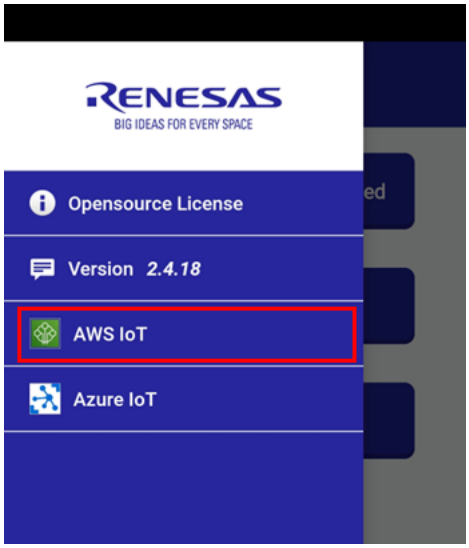

Figure 7. AWS IoT mobile application

In AWS IoT application you can close and open a door, by tapping the Lock icon, see Figure 8.

Figure 8. Open the door

4.4.1 Opening a Door

Figure 9 shows message flow of opening a door.

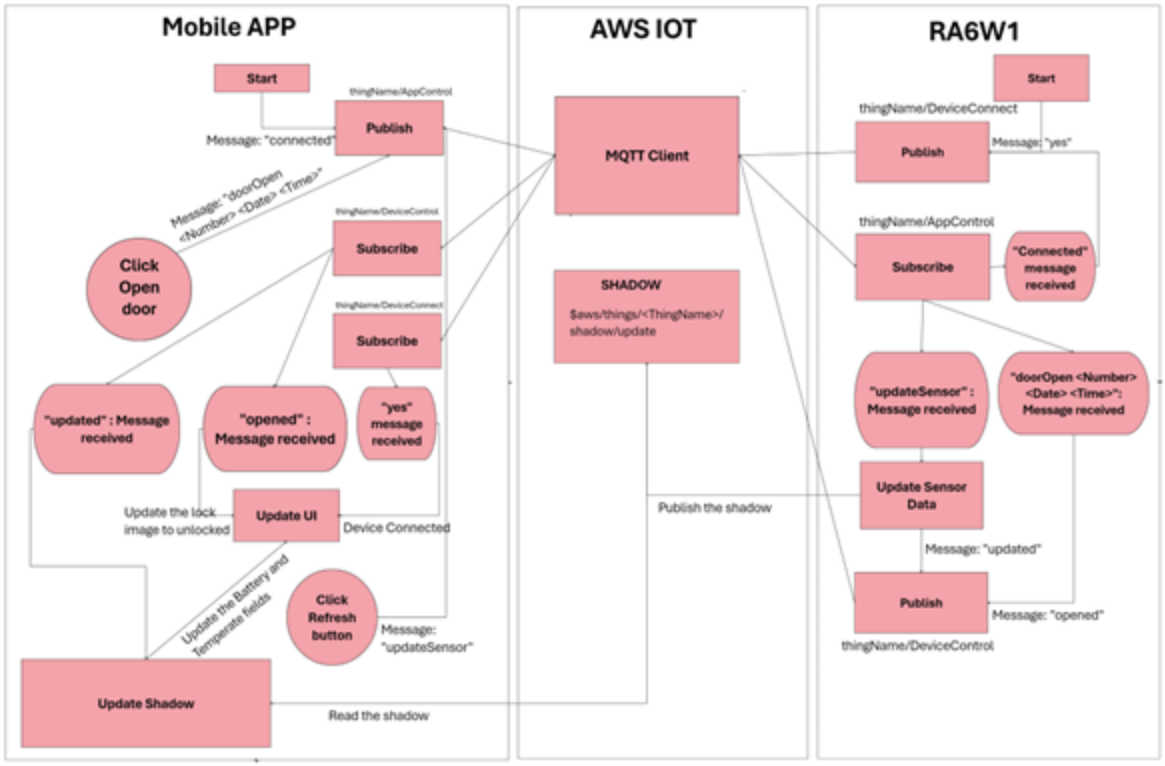

Figure 9. Message flows of opening a door

Figure 10 shows the operation of opening a door on Android app.

Device not connected **Device connected** **Publish door open message** **Received door opened message**

Figure 10. Opening a door on Android app

When the operation of opening a door is completed, the console logs of the RA6W1 appear as follows:

```
open comm
[openControl]

DEBUG:   [aws_dpm_app_door_work:1961] previous MQTT result = 0, doorLock CMD (=1: 0-idle, 1-open,
2-close, 3-auto close)
================================================================================
********************************************************************************
last user Timer ID = 5
last doorOpenFlag state: "true"
last FOTA Stat: 0
last FOTA Url: ""

Sleep mode DPM: KA timer interval(=1800 sec)

[DPM_App_Main] DM_FINISH_DEVICE
recv timeout(=19 ms) set OK (socket=0)
[dpm_keepalive_timer_unregister] OK to deregister AWS DPM's timer(5)
[dpm_keepalive_timer_register] OK to delete AWS DPM's timer(5)
[dpm_keepalive_timer_register] OK to register timer: interval = 1770(sec), tid = 5
>>> Start DPM Power-Down !!!
```

4.4.2 Closing a Door

Figure 11 shows message flows of closing a door.

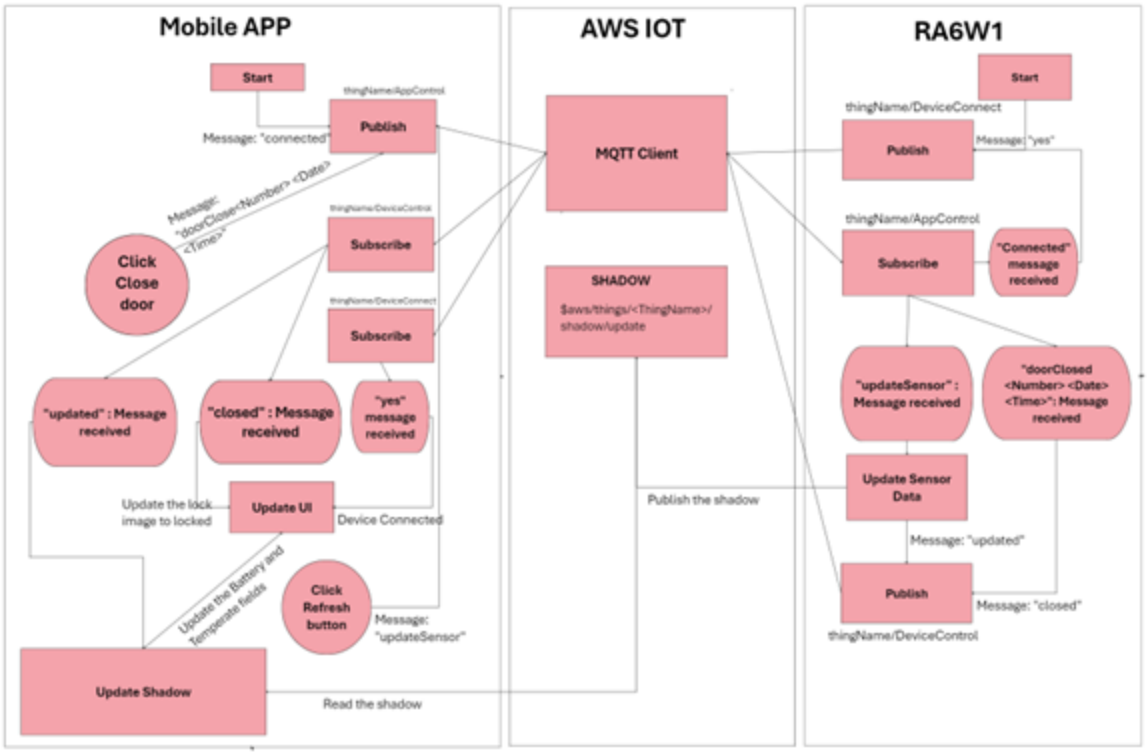

Figure 11. Message flows of closing a door

Figure 12 shows the operation of closing a door on an Android app.

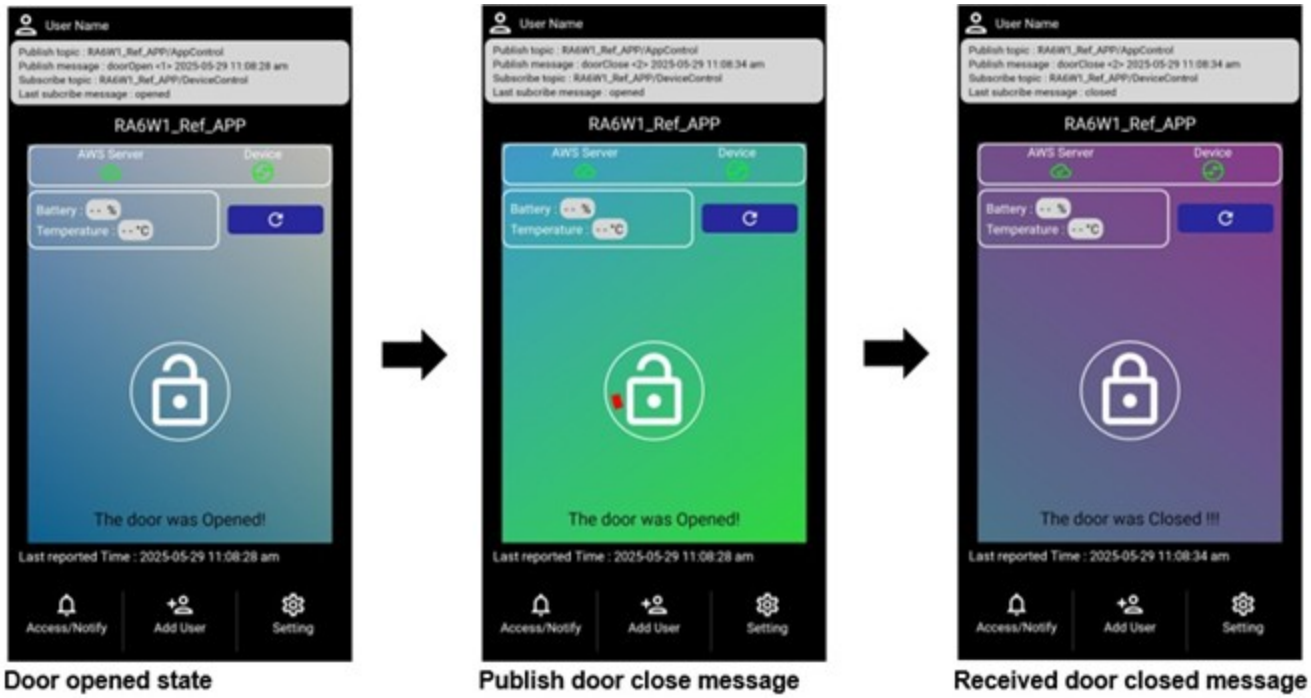

Figure 12. Closing a door on mobile app

When the operation of closing a door is completed, the console logs of the RA6W1 appear as follows:

```
close comm
[closeControl]
DEBUG:   [aws_dpm_app_door_work:1961] previous MQTT result = 0, doorLock CMD (=2: 0-idle, 1-open,
2-close, 3-auto close)
================================================================================
********************************************************************************
last user Timer ID = 5
last doorOpenFlag state: "false"
last FOTA Stat: 0
last FOTA Url: ""
Sleep mode DPM: KA timer interval(=1800 sec)
[DPM_App_Main] DM_FINISH_DEVICE
recv timeout(=19 ms) set OK (socket=0)
[dpm_keepalive_timer_unregister] OK to deregister AWS DPM's timer(5)
[dpm_keepalive_timer_register] OK to delete AWS DPM's timer(5)
[dpm_keepalive_timer_register] OK to register timer: interval = 1770(sec), tid = 5
[aws_dpm_app_finish_loop] break finish_loop...
>>> Start DPM Power-Down !!!
```

5. Reference Application Using AT Commands

To get started with AWS IoT connectivity on the RA6W1 device, which is using AT commands as an interface with external MCU, begin with a ready-to-use reference application. The provided application project,`awsiot_wifi_at_lock_ek_ra6w1`, includes pre-configured project settings, middleware, and example code, which helps to save time and ensures that the environment is correctly set up for AWS IoT development. This project shows how you can create an application that implements a cloud-connected door lock system in RA6W1 where the door lock hardware relates to an external MCU which is interfaced with RA6W1 using UART – AT commands. The mobile application communicates with AWS IoT Core through the internet to remotely control the door lock through RA6W1 and MCU. The mobile application communicates with AWS IoT Core through the internet to remotely control the door lock through RA6W1 and MCU.

If external MCU is not available, you can also use the provided `ttl` scripts to test the reference application. The script replicates the AT commands from external MCU.

Figure 13 shows the components required to run the AWS IoT reference application in RA6W1 through an Internet connection and AWS IoT server:

- AWS IoT - ATCMD reference application package.
- External MCU EVB (RA6M4).
- Terminal application which can run `ttl` scripts (if external MCU is not used).

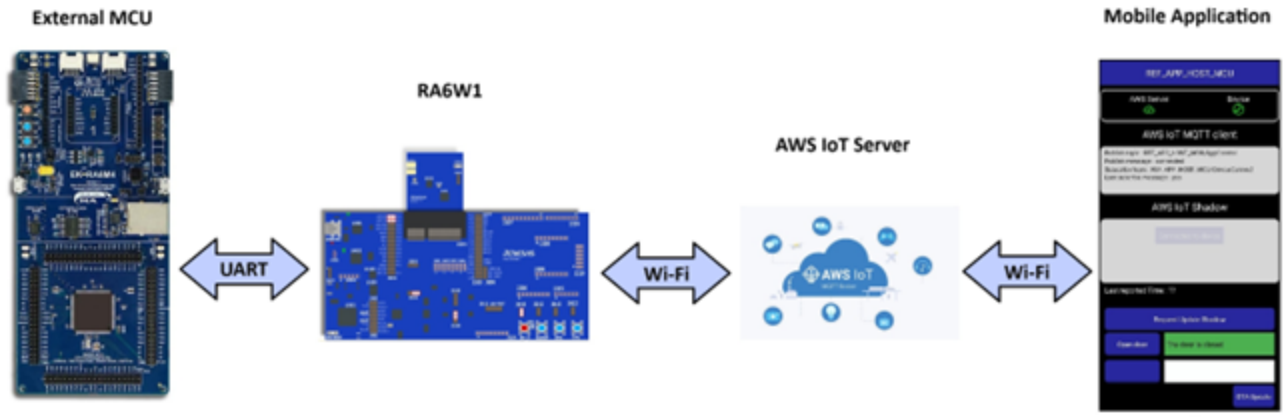

Figure 13. Architecture of AWS IoT reference application using AT commands

Figure 14 shows the jumper connection for AT commands:

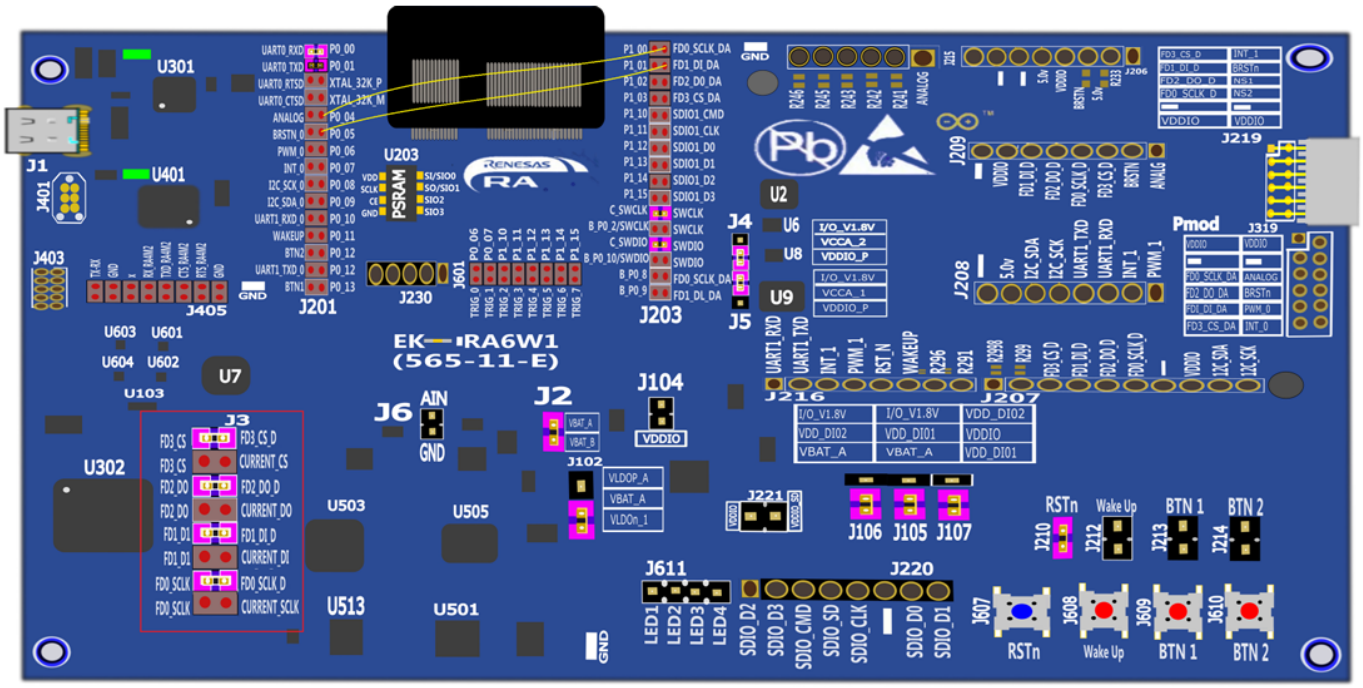

Figure 14. Jumper connection for AT commands

5.1 Load AWS IoT Reference Application in e^2 studio

The project template can be downloaded as a zip file, `r19an0439ee0100-rafw-group-aws-at.zip`, from the Software and Tools section on the Renesas RA6W1 product page.
For more information on how to load the template, see the Load Project Template in e^2 studio section of *RA6W1 Application Development Guide.*

5.2 Build and Flash the Image

For more information on how to flash the image, see the Load Project Template in e^2 studio and the Flash and Debug Using e^2 studio sections of *RA6W1 Application Development Guide*.

5.3 Provision Wi-Fi and Configure AWS IoT Parameters

1. Configure factory reset button, see Figure 15.
   If you want to use the BTN1 button for factory reset, connect the pins P0_10 and BTN1 together using a jumper wire.

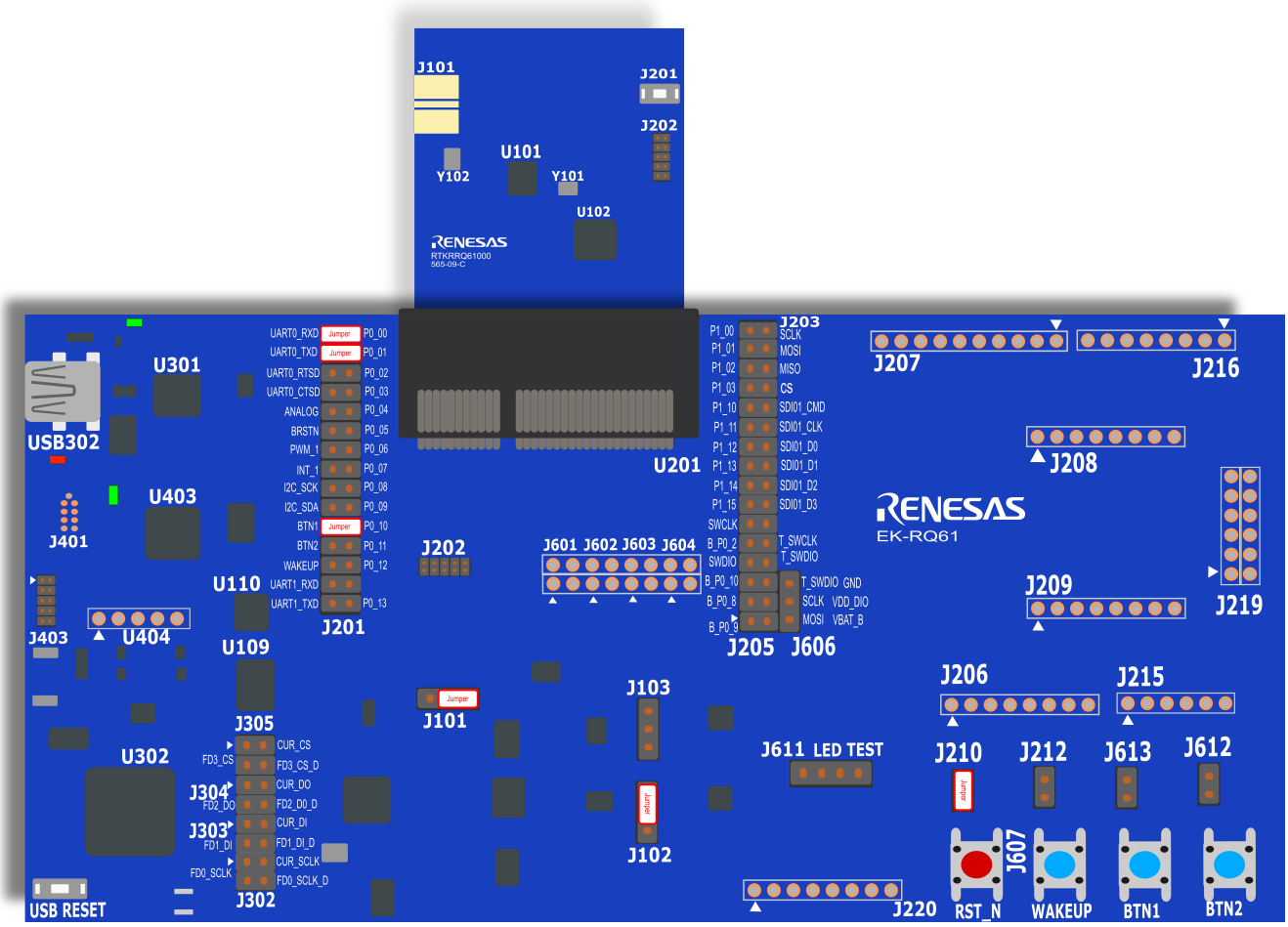

Figure 15. Configure factory reset button in the EVK for Wi-Fi Provisioning mode

2. Press the configured factory reset button using the jumper connection. Now the device does the factory reset and booting in Soft-AP mode and is ready for Wi-Fi provisioning.
3. Configure the following AWS IoT parameters:
   - Unique Thing Name
   - AWS Broker URL
   - Publish and subscribe topic
   - Root CA, Certificate, and Private Key
   - Thing attributes.

   These parameters can be set using the following `ttl` script.

NOTE
If external MCU is used instead of the `ttl` script, the MCU should use the same AT commands shown in the source code.

```
; Wait for entering in provisioning mode
wait '+ATPROV=STATUS 1'
; Configure Thing name
sendln 'AT+AWS=SET,APP_THINGNAME,RA6W1_THINGNAME'
```

```
; Wait for ‘OK’ response
wait 'OK'
; Configure Broker URL
sendln 'AT+AWS=SET,AWS_BROKER,a1kzdt4nun8bnh-ats.iot.ap-northeast-2.amazonaws.com'
wait 'OK'
; Configure Subscribe topic
sendln 'AT+AWS=SET,APP_SUBTOPIC,AppControl'
wait 'OK'
; Configure Publish topic
sendln 'AT+AWS=SET,APP_PUBTOPIC,DeviceControl'
wait 'OK'
; Configure Attributes
; Attribute app_door is used in the publish message from application
sendln 'AT+AWS=CFG 0 app_door 1 2'
wait 'OK'
; Attribute app_shadow is used in the shadow update publish message from application
sendln 'AT+AWS=CFG 1 app_shadow 1 2'
wait 'OK'
; Attribute doorStat is used in the device shadow update
sendln 'AT+AWS=CFG 2 doorStat 1 1'
wait 'OK'
; Attribute battery is used in the device shadow update
sendln 'AT+AWS=CFG 3 battery 2 1'
wait 'OK'
; Attribute DeviceControl is used to publish message to topic, DeviceControl
sendln ‘AT+AWS=CFG 4 DeviceControl 1 0’
wait 'OK'
; Enable DPM
sendln 'AT+AWS=SET,USE_DPM,1'
wait 'OK'
;; Configure RootCA Certificate
sendln #27 'C0,-----BEGIN CERTIFICATE-----'
sendln 'MIIDQTCCAimgAwIBAgITBmyfz5m/jAo54vB4ikPmljZbyjANBgkqhkiG9w0BAQsF'
sendln 'ADA5MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQDExBBbWF6'
sendln 'b24gUm9vdCBDQSAxMB4XDTE1MDUyNjAwMDAwMFoXDTM4MDExNzAwMDAwMFowOTEL'
sendln 'MAkGA1UEBhMCVVMxDzANBgNVBAoTBkFtYXpvbjEZMBcGA1UEAxMQQW1hem9uIFJv'
sendln 'b3QgQ0EgMTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBALJ4gHHKeNXj'
sendln 'ca9HgFB0fW7Y14h29Jlo91ghYPl0hAEvrAIthtOgQ3pOsqTQNroBvo3bSMgHFzZM'
sendln '9O6II8c+6zf1tRn4SWiw3te5djgdYZ6k/oI2peVKVuRF4fn9tBb6dNqcmzU5L/qw'
sendln 'IFAGbHrQgLKm+a/sRxmPUDgH3KKHOVj4utWp+UhnMJbulHheb4mjUcAwhmahRWa6'
sendln 'VOujw5H5SNz/0egwLX0tdHA114gk957EWW67c4cX8jJGKLhD+rcdqsq08p8kDi1L'
sendln '93FcXmn/6pUCyziKrlA4b9v7LWIbxcceVOF34GfID5yHI9Y/QCB/IIDEgEw+OyQm'
sendln 'jgSubJrIqg0CAwEAAaNCMEAwDwYDVR0TAQH/BAUwAwEB/zAOBgNVHQ8BAf8EBAMC'
sendln 'AYYwHQYDVR0OBBYEFIQYzIU07LwMlJQuCFmcx7IQTgoIMA0GCSqGSIb3DQEBCwUA'
sendln 'A4IBAQCY8jdaQZChGsV2USggNiMOruYou6r4lK5IpDB/G/wkjUu0yKGX9rbxenDI'
sendln 'U5PMCCjjmCXPI6T53iHTfIUJrU6adTrCC2qJeHZERxhlbI1Bjjt/msv0tadQ1wUs'
sendln 'N+gDS63pYaACbvXy8MWy7Vu33PqUXHeeE6V/Uq2V8viTO96LXFvKWlJbYK8U90vv'
sendln 'o/ufQJVtMVT8QtPHRh8jrdkPSHCa2XV4cdFyQzR1bldZwgJcJmApzyMZFo6IQ6XU'
sendln '5MsI+yMRQ+hDKXJioaldXgjUkK642M4UwtBV8ob2xJNDd2ZhwLnoQdeXeGADbkpy'
sendln 'rqXRfboQnoZsG4q5WTP468SQvvG5'
sendln '-----END CERTIFICATE-----'
send #3
waitln 'OK'
MPause 500
;; Configure Client Certificate
sendln #27 'C1,-----BEGIN CERTIFICATE-----'
sendln 'MIIDWjCCAkKgAwIBAgIVAKQVbt137DPvbOJzpLZM84OFuaFhMA0GCSqGSIb3DQEB'
sendln 'CwUAME0xSzBJBgNVBAsMQkFtYXpvbiBXZWIgU2VydmljZXMgTz1BbWF6b24uY29t'
sendln 'IEluYy4gTD1TZWF0dGxlIFNUPVdhc2hpbmd0b24gQz1VUzAeFw0yNDAyMjIwNzM3'
sendln 'NDRaFw00OTEyMzEyMzU5NTlaMB4xHDAaBgNVBAMME0FXUyBJb1QgQ2VydGlmaWNh'
```

```
sendln 'dGUwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCx5HvUzqA8YjeWNCGS'
sendln 'VR2jWBqZBWNiEQQ0fgP+bG3eThhPBjcnOCW2CorMvYV1IKaptjjQLKwnEsV5veyM'
sendln 'TGiSrlGNFC0mE0QI3L0LLpIvMSzmU4Qq2AuSz+cx/2xt6RBvZAKFxGxyS6hFfQel'
sendln 'ZGQP1dxJI511pdznaNuGWtyAz+7o3DhpCPX4t1yNnLqRIAYQjqpML+IR5GO3+kSj'
sendln '9brg5a7OgeQfOTm5OLlkQNrW6IQxdvKJgXkAHYzDcZEfK5DnAYTzlGI0pxYrGH0h'
sendln 'CcUPBTTcb3vblAayEtJAu358QeDHA+cI2iuMaPDnNm//SNah7VrNwUQA5WG4bzrx'
sendln 'TNvdAgMBAAGjYDBeMB8GA1UdIwQYMBaAFBvxrsDla8i5VP0dJEmokUXpv+ogMB0G'
sendln 'A1UdDgQWBBQhxX2yhzF7N037pEGUdq1Jlmb6vTAMBgNVHRMBAf8EAjAAMA4GA1Ud'
sendln 'DwEB/wQEAwIHgDANBgkqhkiG9w0BAQsFAAOCAQEAoRPWtx2IwxaY8BFlmfPgTqTd'
sendln 'mLU60Qi+ytOnFuTm7Sgqg2SmjQcsxSGDwMzN0gVi9Qh8WBodsVGZ8/lhvtmvtdZU'
sendln 'P0H6gQheXAeA2ervKqRMqLkfsuYgpQLmTLtF90b22SZe0jMCIO+Q5Yk3Un6knt4H'
sendln 'l/oSi5tuA+eaLavH7Xr+QxqORu6l4qQfi7P1rR8807+Dn9Bf+fs65hFO3EL5OY5H'
sendln '61oLU3Wj6KITyeZAcflnnL6taDYoNjsptJtPlGukwLgbgu60HwBanWnBCP6oskjU'
sendln 'wcESdbbU8mdKapYP3szjT1o97fji/Sh/Fhk7TB5vA561JED/8SvhmQk5HykFug=='
sendln '-----END CERTIFICATE-----'
send #3
waitln 'OK'
MPause 500
;; Configure Client Private Key
sendln #27 'C2,-----BEGIN RSA PRIVATE KEY-----'
sendln 'MIIEpAIBAAKCAQEAseR71M6gPGI3ljQhklUdo1gamQVjYhEENH4D/mxt3k4YTwY3'
sendln 'JzgltgqKzL2FdSCmqbY40CysJxLFeb3sjExokq5RjRQtJhNECNy9Cy6SLzEs5lOE'
sendln 'KtgLks/nMf9sbekQb2QChcRsckuoRX0HpWRkD9XcSSOddaXc52jbhlrcgM/u6Nw4'
sendln 'aQj1+LdcjZy6kSAGEI6qTC/iEeRjt/pEo/W64OWuzoHkHzk5uTi5ZEDa1uiEMXby'
sendln 'iYF5AB2Mw3GRHyuQ5wGE85RiNKcWKxh9IQnFDwU03G9725QGshLSQLt+fEHgxwPn'
sendln 'CNorjGjw5zZv/0jWoe1azcFEAOVhuG868Uzb3QIDAQABAoIBAQCGN9TyeD/Pr28v'
sendln 'Rf/6b3vgLW5L/BPC/wHENTwAL4Ep1JRks1TTo3pYglW9KlVSTSNSMulkr//nrzMN'
sendln 'qq2aIWBZI2rmXc2+x5Ryd1Gu6SFkt96fnW3AAaehynkucQA3YTyEPtnkYZJNHiZE'
sendln '35PwRvF0PTBqgpOgG1ezxZq0IUPzbyy2OeoJFhGT52kD0XCJ0SqnwmuuzGiilrIC'
sendln '/5icgrSjJWYo5rCYsicdVRpIsUs6ku1Umm7hn0z7QdD9n83bmzmE7WQIEU56YmeE'
sendln 'edmkJ9j7vWZpvelu9gnezjS8h2qhiGWr18oCfj4v5mkSXxFJgNngM3aJGrW7Faek'
sendln 'wVVe+dllAoGBAOMWUrW+Qv+D7cxVFAYt65SA6PPqHC0YrskOaQc79KLpOrDkb3yy'
sendln 'KOj4jb31CTs8k7D2m2/CSM4sCWp5TaH2D8r4qTGRTGmGiGW6vZgr63hSYs5DjU2U'
sendln 'U5kOkL+/avPZ4fF+RK1GdtoG28AuK2uA/7VoCQl/anm2B783V1BRoV0vAoGBAMiK'
sendln 'tbPAsCcUx8HFbkiB8u7rI7Q2c/Hc/dowXqT/XXHgZAqE+Ju62nZqJ53Itq3TUmMa'
sendln 'babs8SKYpmFCIyuw5F7jIay9tbYCTYCknhreR2gPNFt4dOdHLT9Vhx4H9XZcQks4'
sendln 'gOcLw5Ni1GirX9VMs8uAM9YlepOHSEr9rIjG7YyzAoGBAMIoufsOe/Y/9Lf6Hi7S'
sendln 'YFQOjB7Qkdq0+eyJ158J0jbgNwA7sF7rbTMUIQzDT2tIdfaeQ3Qgp2MwH7Tbllbe'
sendln 'Lc6bIP5yfuTS1Bg2vBg5pRCxUC2PcGdeZMO+wmBP58ArEJYMIegNEV2E86qzTwiR'
sendln 'uRB/rQpj2MPLsX/6bzsLMG6dAoGAQhbkds7Dvr1sb4F/LMoWo4I+i/9+CnFH/4X9'
sendln 'SucVhpfqoETeIYRcxrWJRMiG25ZPCnYFOQPRHBOukVLYgGeVe2fjCyiiH892dzTJ'
sendln 'HhWu9q48FQEHlcixMrQfCViaHl2dQ2jloj8QPxxM4AnKVWMnxaMEckh7su7cdkpP'
sendln 'd+wHEV0CgYA5r/hkhVKqNDQpF0ct3B4Idofz08nVv+MmvvUQrv9edXFgw3VAZhNR'
sendln 'gQIH3r8BFEGstTRlXRadT3zIZGSViJ/QnWP53476HiLtTG5oVF6HkY0mBvPRZa0g'
sendln 'wXQBS62JONf9Vog2f6Z1yjspl5W7YmO8xwyEduRkRoY0zzS77QS06w=='
sendln '-----END RSA PRIVATE KEY-----'
send #3
waitln 'OK'
```

4. Download the Renesas Wi-Fi Provisioning application from the Google Play Store or the Apple App Store to connect RA6W1 to Access Point. For Android, see Figure 16.
5. If the location access permissions are asked when you open the application for the first time, open the application and tap **While using the app**.
6. Select **Start DA16200/RRQ61000-based**.
7. Tap **I'm ready** and tap **Connect**.
8. When the application connects to RA6W1, tap **NEXT**.
   Now the application lists all the access points that RA6W1 scanned.
9. Select the needed access point and enter the password.
   This writes the Wi-Fi network details to the device.
10. In the success dialog, tap **OK**.

Wi-Fi provisioning is now completed and RA6W1 reboots and starts as a Station and connects to the access point configured from the application. After the successful connection to the internet, it connects to the AWS server.

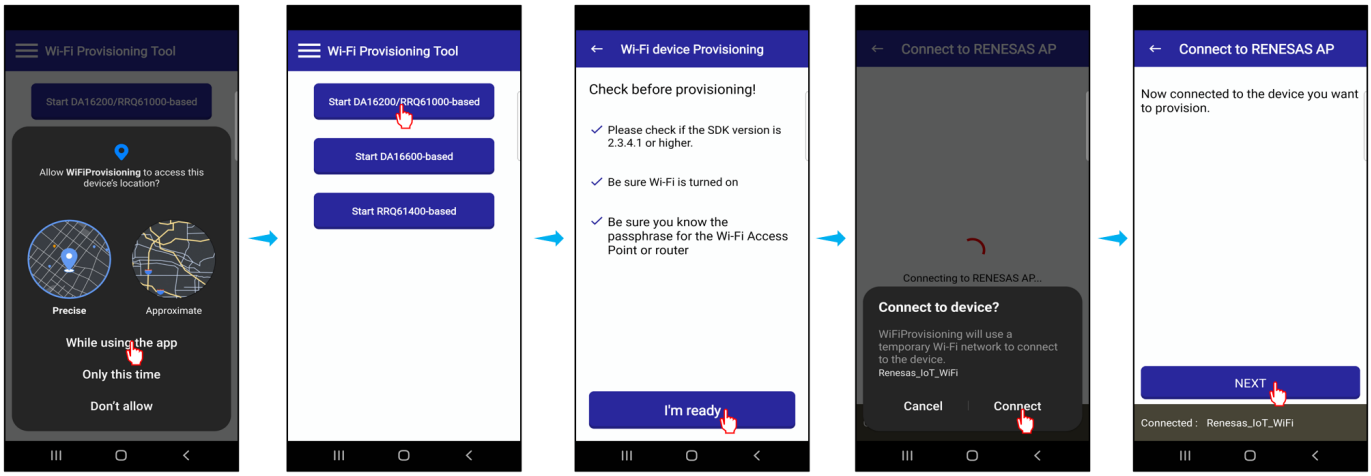

Figure 16. Renesas Wi-Fi provisioning application steps

5.4 AT Command List

Table 1. Basic set of AT commands from MCU to RA6W1

Head	Main	Sub	Parameters
AT+AWS=	SET	APP_ THINGNAME	Set the device thing name. Used to choose a device by its thing name during provisioning.
		AWS_BROKER	Set the broker address.
		APP_LPORT	Set the local port. Default is 443.
		APP_SUBTOPIC	Set subscriber topic name, and the default is "AppControl".
		APP_PUBTOPIC	Set sub-topic name, and the default is "DeviceControl".
		USE_DPM	Define the operation of Sleep mode 3. 0 – DPM Disabled. 1 - DPM Enabled
		SLEEP_MODE	Set Sleep mode. This command is not used currently.
		RTC_TIME	This command is not implemented. It can be used to set the wake-up time for not connected sleep. Where RA6W1 waken up by RTC.
		DPM_KEEP_ ALIVE	This command is not implemented. It can be used to set the keep-alive time between the IoT device and the AP.
		USE_WAKE_UP	This command is not implemented. It can be used to set the wake-up time for Full-boot mode.
		TIM_WAKE_UP	This command is not implemented. It can be used to set the period to check a beacon frame from the AP.
		AWS_USE_FP	This command is not implemented. It can be used to enable/disable fleet provisioning when fleet provisioning feature is implemented.
		APP_BRD_ FEATURE	This command is not implemented. It can be used for board features and Pin MUX.
		UART_CFG	This command is not implemented. It can be used to configure the UART for AT commands.
		APP_MCU_WU_ PORT	This command is not implemented. Set RA6W1 port to wake up external MCU.
		APP_MCU_WU_ PIN	This command is not implemented. Set RA6W1 pin to wake up external MCU.

Example:
`AT+AWS=SET APP_THINGNAME <Assigned Thing Name>`
`AT+AWS=SET AWS_BROKER a1kzdt4nun8bnh-ats.iot.ap-northeast-2.amazonaws.com`			

Table 2. Write TLS certificate from MCU to RA6W1

Start code	Sub code	Type	End code
0x1B (Esc character)	C0,	Root CA. Self-Signed, well known. Has root certificate public key. Signed by root certificate private key.	0x03 (End of Text or Ctrl+C)
	C1	Certificate key. Has own public key. Signed by root certificate private key. Use root certificate public key to prove authenticity.	
	C2	Private key. Has own public key. Signed by certificate private key. Use certificate 1 public key to prove authenticity.	
Example: send "0x1B" over UART or press Esc. send "C0,-----BEGIN CERTIFICATE-----\n" " MIIDQTCCAimgAwIB ….. -----END CERTIFICATE-----" over UART or copy paste the certificate after C0, send "0x03" or press Ctrl+C			

Table 3. Configuration data from MCU to RA6W1

Head	Main	Sub	Parameters
AT+AWS=	CFG	[number] [name] [value-type] [MQTT-type]	**Number**: ■ Index to identify the saved attribute. ■ Increase by 1 when setting a new attribute. ■ Maximum value is 10 (total supported attributes is 10). **Name**: String specifying the attribute name. **Value-type**: 0 - Integer type 1 - String type 2 - Float type **MQTT-type**: ■ 0 - Publish: The prompt command is used to send a value from the MCU to the phone. For example, door state = true/false. ■ Shadow: The value is sent to the device twin and updated on the phone the next time it is connected. ■ Subscribe: The prompt command is used to send a value from the phone to the MCU. For example, door open command.

Example:
`AT+AWS=CFG 0 app_door 1 2`
`AT+AWS=CFG 1 app_shadow 1 2`
`AT+AWS=CFG 2 doorStat 1 1`			

Table 4. Command of MCU to RA6W1

Head	Main	Sub	Description
AT+AWS=	CMD	MCU_DATA	Used by the MCU to set a CFG parameter in the RA6W1.The value must be the same format as defined by the CFG setting. Parameters: [number] [name] [value].
		RESTART	Reboot the device keeping the current mode and status.
		GET_STATUS	Get the current AWS IoT status. The MCU can read the current status from RA6W1 at any time.
		RESET_TO_AP	Not implemented currently.
		FACTORY_RESET	Not implemented currently.

Example:
`AT+AWS=CMD MCU_DATA 2 doorStat closed 3 battery 80`
`AT+AWS=CMD RESTART`			

Table 5. Command of RA6W1 to MCU

Head	Main	Parameters	Description
+AWSIOT	SERVER_DATA	[number] [name] [value]	Used by RA6W1 to set a CFG parameter in the MCU. The value must be the same format as defined by the CFG setting.
	CMD_TO_MCU	update	Used by RA6W1 to request the status of devices such as sensors, batteries, and doors from the MCU. RA6W1 maintains the values obtained from the MCU and forwards them when requested by an external phone app or by an MQTT wake-up event.
Example: `+AWSIOT SERVER_DATA 0 door_control open+AWSIOT CMD_TO_MCU update`			

Table 6. Status from RA6W1 to MCU

Status	Value	Parameters
IDLE	-1	Initial state of AWS-IoT application. Sent when a system error occurs. For example, network connection failure.
Done factory reset	0	Sent after completes factory reset.
Boot Ready	1	Sent when entering AWS IoT application mode.
Need configuration	5	Sent if there is no setting. MCU should set and configure with the SET and CFG command.
Start AP mode	10	Sent when being started to AP mode.
Network OK	15	Sent when it is OK to connect AP without problem.
Network fail	16	Sent when it fails to connect AP with any problem.
Start STA	20	Sent when being started to STA mode.
Done STA	25	Sent when entering Sleep mode for DPM.
MCUOTA	30	Sent when MCU OTA starts processing.
Example: `+AWSIOT STATUS 15`		

6. AWS IoT

The RA6W1 is an MCU for IoT applications such as security systems, door locks, and smart applications. This section provides procedures on how to configure AWS IoT for communicating with the RA6W1 IoT device.
To connect a device to the AWS IoT server, you need an AWS account.
To create an AWS account:

1. Go to the AWS website and create a free account:https://portal.aws.amazon.com.
2. Create an administrative user for performing daily administrative tasks.
3. Open the AWS IoT console to get started with AWS IoT.

NOTE
While creating the project, certificates and policies, the windows (dialogs) may look slightly different from what is shown in the document, and you need to use navigation in the AWS IoT core environment to find corresponding attributes.

6.1 Connect Devices to AWS IoT

You can configure and manage the thing objects, certificates, rules, jobs, policies, and other elements of IoT solutions through the AWS IoT console. Before sending data to and receiving data from AWS IoT server, you should register a device.

6.1.1 Register Device

In the Thing Registry, the devices connected to the AWS IoT server are **Things**. The Thing Registry allows keeping records of all devices that are connected to an AWS IoT account.
To register a device in the Thing Registry:

1. In the AWS IoT console, go to **Registry** > **All devices** > **Things**.
2. Click **Create things**, see Figure 17.

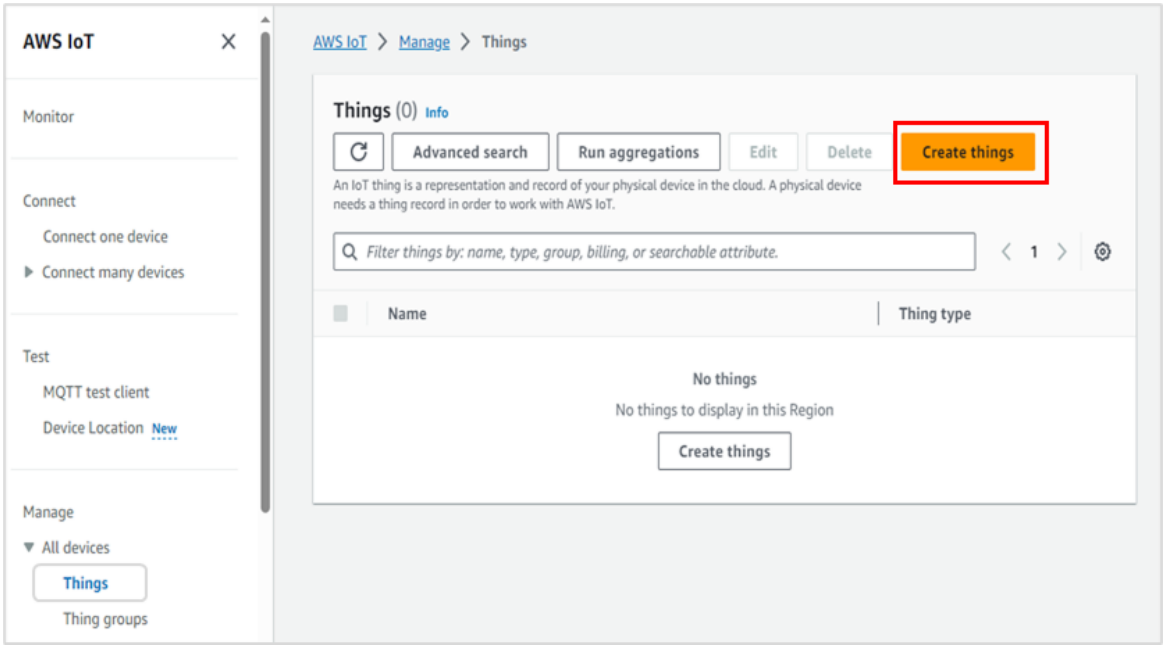

Figure 17. Create things

3. Select **Create single thing**.

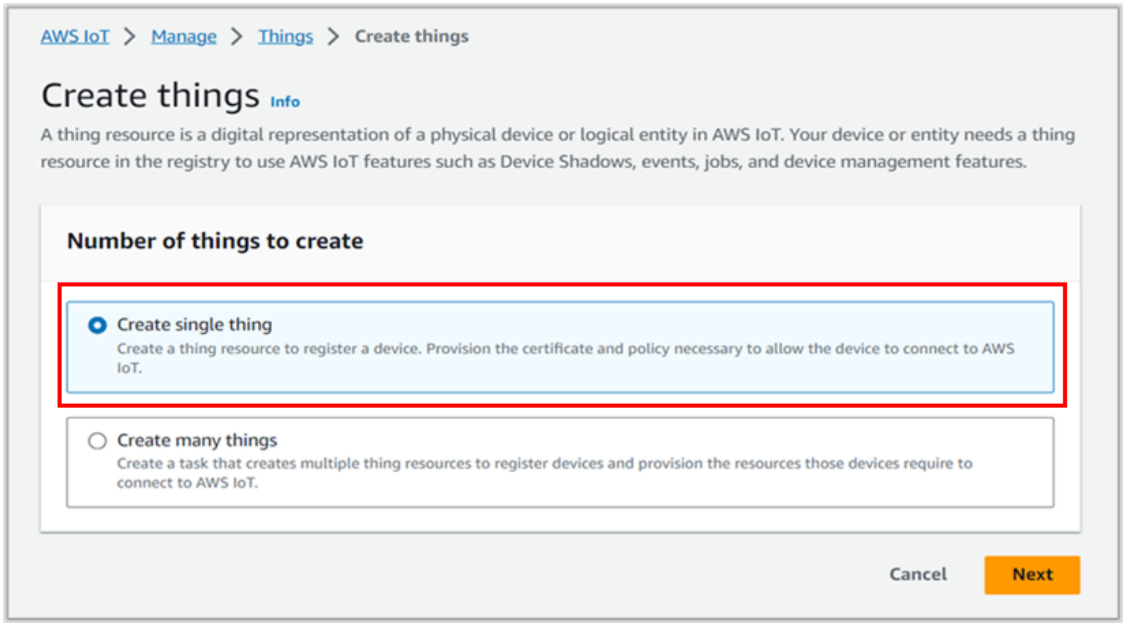

Figure 18. Create single thing

a. In **Thing name**, enter a device name, for example, MyTestDoorLock.
b. In **Device Shadow**, select **Unnamed shadow (classic)**, and click **Next**, see Figure 19

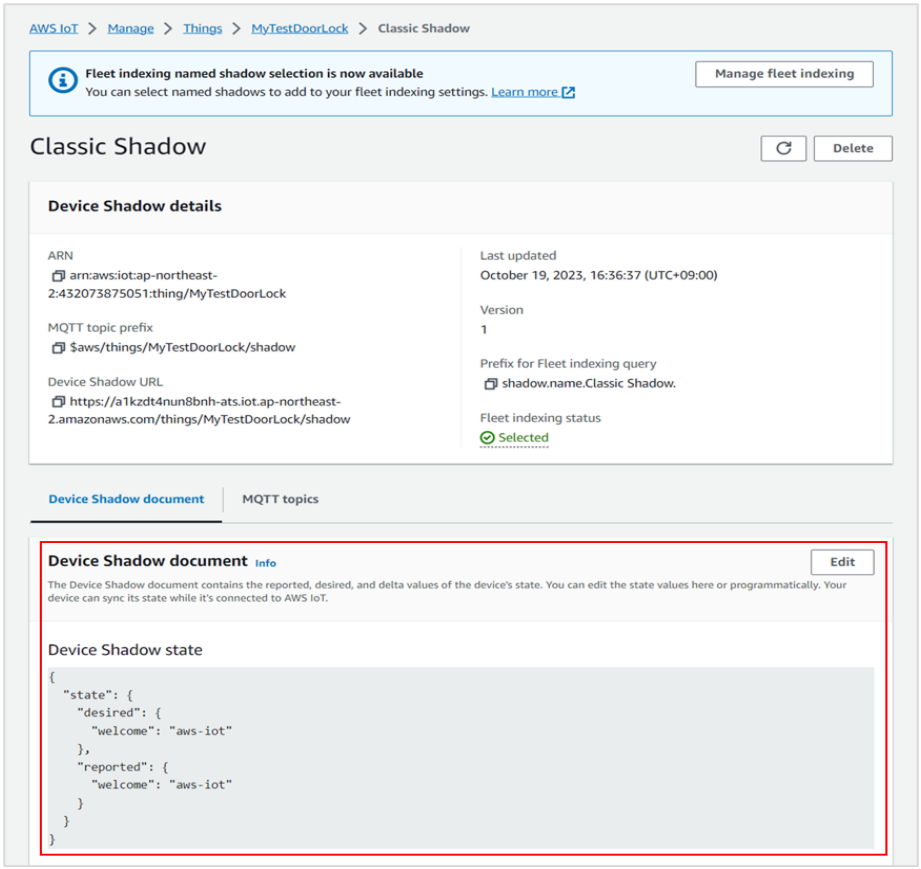

Figure 19. Thing properties

4. Select **Skip creating a certificate at this time** and click **Create thing**.
Now you have the thing created to perform the test and it is named **MyTestDoorLock**.

5. In the **Things**, select the created thing and the thing details appear.
6. For the shadow function of the thing, select the **Device Shadows** and select **Classic shadow**.

Figure 20 shows the device shadow document.

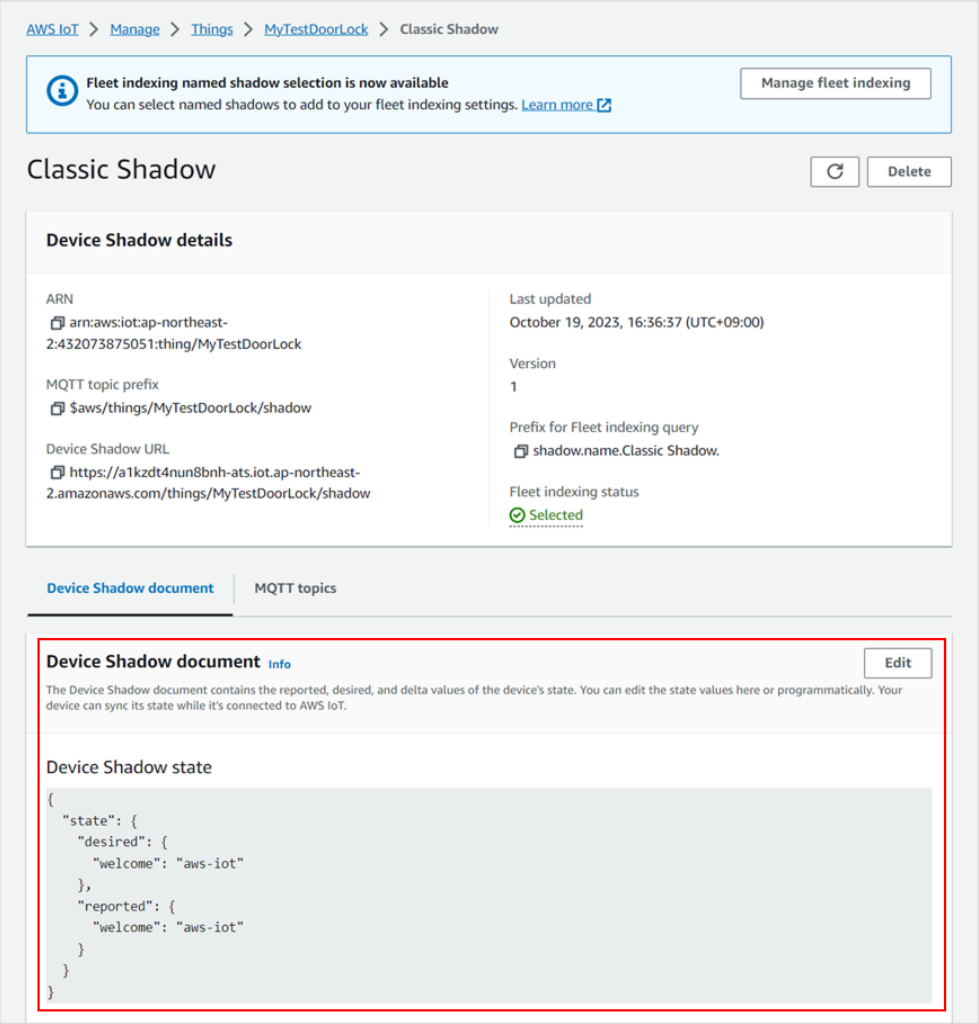

Figure 20. Device shadow document

For more information on device shadows for AWS IoT, visit AWS IoT Device Shadow service: https://docs.aws.amazon.com/iot/latest/developerguide/iot-device-shadows.html.

6.1.2 Create and Activate Device Certificate

The communication between the device and the AWS IoT web service is protected by X.509 certificates. You can let the AWS IoT generate a certificate or you can use your own X.509 certificate. This section shows that AWS IoT generates the X.509 certificate. You should activate the certificate before using it.
To create and activate a device certificate:

1. On the navigation pane, expand **Security** and click **Certificates** > **Add certificate** > **Create certificate**, see Figure 21.

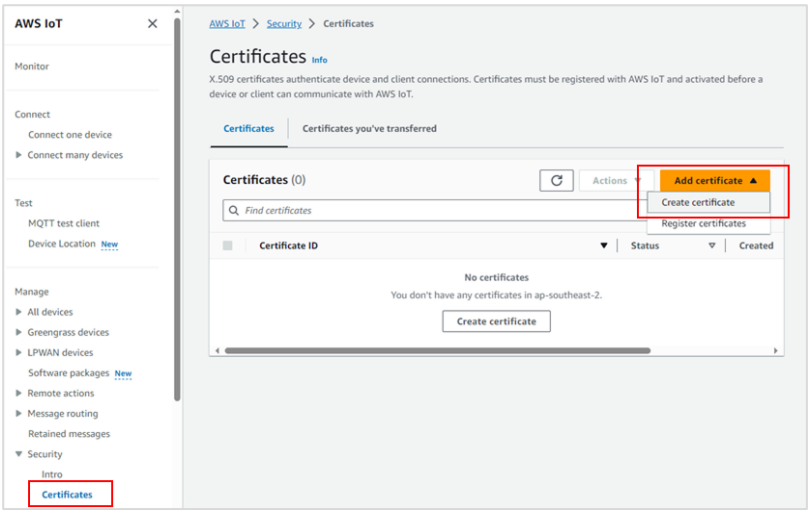

Figure 21. Create certificates

2. On the **Create certificate** page, select the **Auto-generate new certificate (recommended)** and **Active** options, and then click **Create**.

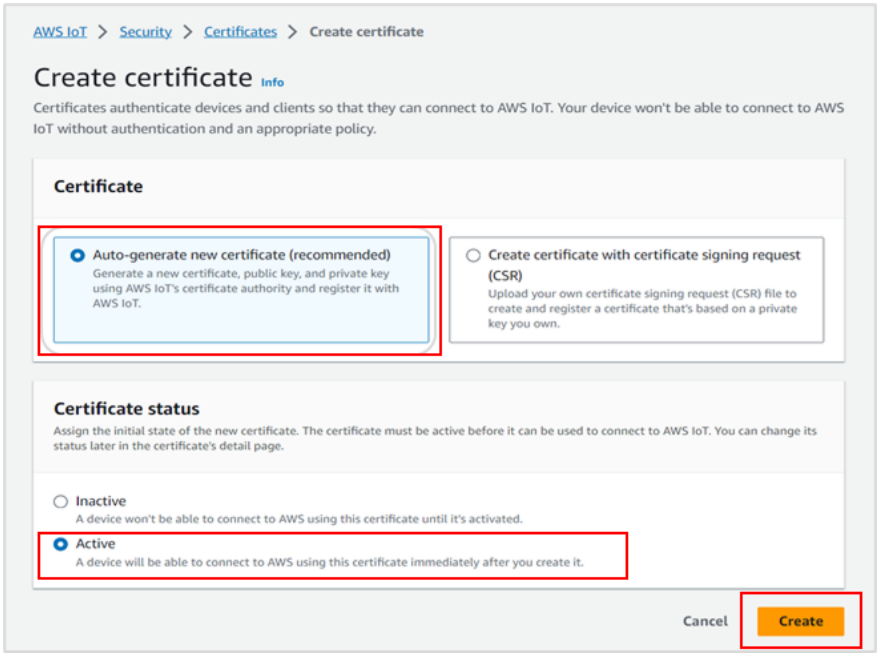

Figure 22. Create certificates (continued)

3. Download and save the following certificates for AWS IoT:
   - Device certificate
   - Private key
   - Root CA certificates.

NOTE
You must save the certificate files before leaving the page. After leaving this page in the console, you no longer have access to the certificate files.

The certificate status should be Active in the list of certificates, see Figure 23.

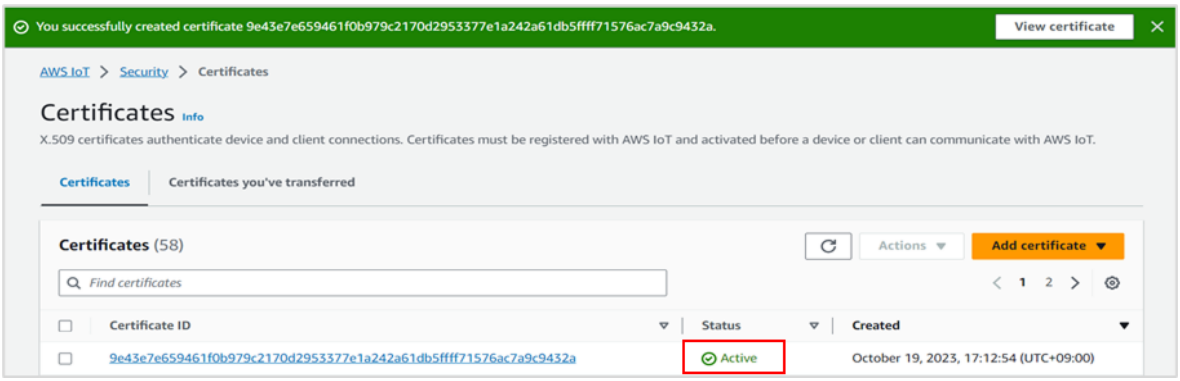

Figure 23. Active certificate

6.1.3 Create and Attach AWS IoT Policy

The X.509 certificates are used to authenticate the device with the AWS IoT. The AWS IoT policies are used to authorize the device for AWS IoT operations, such as subscribing or publishing to MQTT topics. The device displays its certificate only while connecting to the AWS IoT.
To allow the device for AWS IoT operations, you should create an AWS IoT policy and attach that policy to the device certificate.
To create an AWS IoT policy:

1. On the navigation pane, go to **Security** > **Policies** > **Create policy**.

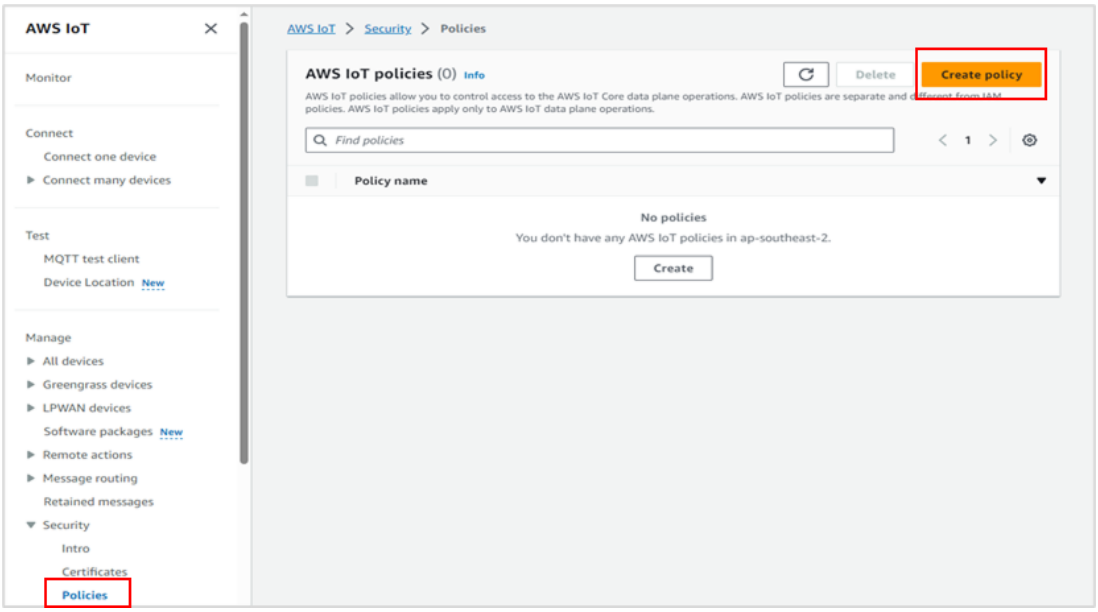

Figure 24. Create policy

2. On the **Create policy** page, do the following:
   a. In the **Policy name** field, under **Policy properties**, enter a name for the policy (for example, MyTestPolicy). Renesas Electronics recommends not using personally identifiable information in policy names, see Figure 25.

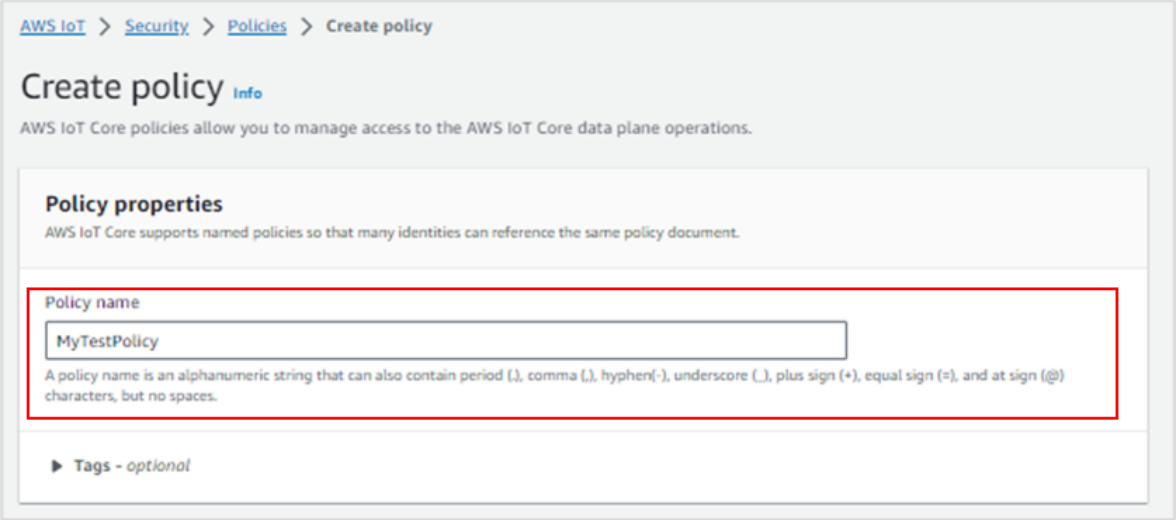

Figure 25. Add policy name

   b. Set **Action** for the policy to **iot:***.
   c. Set **Resource ARN** to *.
   d. After entering the required information, click **Create**.

NOTE
The examples in this document are intended only for development environments. All devices in your production fleet must have credentials with privileges that authorize only intended actions on specific resources. The specific permission policies may vary depending on use cases. Identify the permission policies that best meet the business and security requirements. For more information, see Example Policies and Security Best practices in AWS IoT.

3. To view the created policies, expand **Security** and click **Policies**, and select the created policy.

After an AWS IoT policy is created, you must attach that policy to the device certificate. The attachment of an AWS IoT policy to a certificate gives the device the permissions that are specified in the policy.

To attach the AWS IoT Policy to a device certificate:

1. Go to the certificate created by you, select **Policies**, and click **Attach policies**.
2. Select the created policy and click **Attach policies**.

NOTE
A device should have a certificate, private key, and root CA certificate to authenticate with the AWS IoT. Renesas recommends that you attach the device certificate to the thing that represents the device in AWS IoT. This allows you to create AWS IoT policies that grant permissions based on certificates attached to things.

3. Go to the certificate created by you, select **Things**.
4. Selected created thing, and then click **Attach to things**.
5. To activate the certificate, in the **Actions**, click **Activate**.

6.2 AWS Core APIs

The AWS Core MQTT is documented online. Table 7 lists APIs provided by AWS Core MQTT that are used as a part of the Application Example.

Table 7. MQTT module APIs

API	Description
MQTT_Init	Initializes an MQTT context.
MQTT_Connect	Establishes an MQTT session.
MQTT_Subscribe	Sends MQTT SUBSCRIBE for the given list of topic filters to the broker.
MQTT_Publish	Publishes a message to the given topic name.
MQTT_Ping	Sends an MQTT PINGREQ to broker.
MQTT_Unsubscribe	Sends MQTT UNSUBSCRIBE for the given list of topic filters to the broker.
MQTT_Disconnect	Disconnects an MQTT session.
MQTT_ProcessLoop	Loop to receive packets from the transport interface. Handles keep-alive.
MQTT_ReceiveLoop	Loop to receive packets from the transport interface. Does not handle keep-alive.
MQTT_ GetSubAckStatusCodes	Parses the payload of an MQTT SUBACK packet that contains status codes corresponding to topic filter subscription requests from the original subscribe packet.
MQTT_Status_strerror	Error code to string conversion for MQTT statuses.
MQTT_PublishToResend	Gets the packet ID of next pending publish to be resent.

6.3 FSP APIs

6.3.1 AWS IoT Port Layer (rm_awsiot_w)

The only two APIs in FSP module rm_awsiot_w are not implemented currently: RM_AWSIOT_W_Open and RM_AWSIOT_W_Close. Therefore, these APIs are not used in the reference application.

6.3.2 AWS LWIP Sockets Wrapper (rm_aws_lwip_sock_wrap_w)

This section lists the APIs for the rm_aws_lwip_sock_wrap_w (Networking) Module. The module provides interface to control, access, and write to lwIP. Table 8 lists the APIs used to manage the DPM for AWS functionality.

Table 8. AWS LWIP sockets wrapper FSP APIs

API	Description
app_dpm_get_sleep_flag	Retrieves the flag indicating whether the API enters or exits DPM.
app_dpm_set_sleep_flag	Sets the flag indicating whether the system should enter or exit DPM.
app_dpm_info_get	Gets the DPM information from RTM if it exits, or allocates a new one if it does not.
app_dpm_get_client_socket_port	Gets the saved client binding port.
app_dpm_set_send_pub_flag	Sets the flag whether the device publishes to server or not.
app_dpm_get_send_pub_flag	Gets the flag whether the device publishes to server or not.
app_dpm_set_wait_job_next_flag	Sets the flag whether the device needs to wait for the flag for the next job or not.
app_dpm_get_wait_job_next_flag	Gets the flag whether the device needs to wait for the flag for the next job or not.
app_get_peer_ip_str	Gets the IP string of server to connect.
app_get_random_local_port	Retrieves a randomly generated local port number for socket binding.
app_get_count_of_reconnection	Gets the number of reconnection attempts on DPM Wake-up mode.
app_socket_set_port_n_filter	Registers a binded local port for DPM mode and sets a port filter for DPM wake-up.
app_set_persistent_session	Sets the flag depending on whether the persistent session exists or not.
app_is_persistent_session	Retrieves the flag indicating whether a persistent session is enabled.
app_is_reconnected	Gets the flag whether reconnection happened or not.
app_set_reconnect_flag	Sets the reconnection flag to the input parameter.
app_dpm_get_recv_timeout_flag	Gets the receive timeout flag.
app_dpm_set_recv_timeout_flag	Sets the receive timeout flag to the input parameter.
app_dpm_get_unknown_uc_flag	Gets the unknown DPM wake-up cause flag.
app_dpm_set_unknown_uc_flag	Sets the unknown DPM wake-up cause flag to the input parameter.
app_set_ka_value	Sets the keep-alive time interval for connection peer to the input parameter by seconds.
app_get_ka_value	Gets the keep-alive time interval for communication with peer.
app_dpm_set_rcv_ready	Notifies the DPM module that the receiver is ready after DPM wake-up.
persistant_read_callback_set	Sets the read callback function to get server IP address from nvram.
persistant_write_callback_set	Sets the write callback function to get server IP address to nvram.
awsiot_app_print_elapse_time_ms	Checks passed time.

The reference applications, `awsiot_wifi_lock_ek_ra6w1` and `awsiot_wifi_at_lock_ek_ra6w1`, show how to implement the APIs for AWS IoT applications.
For more information about the APIs, see the FSP documentation on the Renesas website.

7. Reference Application for AWS Fleet Provisioning

AWS provides several different ways to provision a device and install unique client certificates on it.

- **Manual Provisioning**: This method creates Certificate and Thing manually in AWS and attaches policy with the certificate. This certificate is flashed into the device.
- **Fleet Provisioning by Claim**: All devices share one claim certificate. Initial connection is established using the claim certificate. AWS issues a unique device certificate, and the device switches to the unique certificate.
- **Fleet Provisioning by Trusted User**: The device connects to AWS IoT for the first time when a trusted user, such as an end user or installation technician, uses a mobile app to configure the device in its deployed location.
- **Just-In-Time Provisioning (JITP)**: A device connects with its pre-signed certificate (signed by a registered Certificate Authority). AWS IoT verifies the CA, triggers a registration process, creates an IoT "Thing", attaches a policy, and activates the certificate, allowing the device to connect and function securely.
- **Just-In-Time Registration (JITR)**: This method uses an AWS IoT rule and Lambda for custom logic, while JITP uses a provisioning template for simpler, template-based automation, both scaling efficiently for IoT fleets.

In the Fleet Provisioning reference application, the **Fleet Provisioning by Claim** method is used. This method can be further differentiated by the methods in which it obtains a permanent certificate and private key from AWS.

- **CreateKeysAndCertificate**: Calls `CreateKeysAndCertificate` to create a new certificate and private key using the AWS certificate authority.
- **CreateCertificateFromCsr**: Calls `CreateCertificateFromCsr` to generate a certificate from a certificate signing request that keeps its private key secure.

In the reference application, the `CreateCertificateFromCsr` method is used.

7.1 AWS IoT Fleet Provisioning by Claim (CSR method)

The AWS IoT Fleet Provisioning reference application showcases a way to provision a fleet of IoT devices with unique certificates and register them with AWS IoT Core using the Fleet Provisioning feature. It shows how devices with the ability to generate a public-private key-pair on board can utilize a common claim certificate (across the entire fleet of devices) to request unique certificates from AWS IoT Core for their generated key-pairs, and register themselves with AWS IoT Core as AWS IoT thing resources.

For information about configuring AWS account for fleet provisioning, see Section 7.2 AWS Settings for Fleet Provisioning.

The process of fleet provisioning by claim method is the following:

1. Device opens TLS connection to AWS IoT using initial Claim certificate and Claim private key and connects to AWS IoT MQTT broker.
2. Generates a private/public key pair.
3. Keeps private key secure by saving it in the device.
4. Creates a CSR (Certificate Signing Request) and publishes it to: `$aws/certificates/create-from-csr/<payload-format>`. The message payload format is `cbor` or `json`.
   Payload:

```
{
    "certificateSigningRequest": "string"
}
```

5. AWS responds with providing New device certificate, Certificate ID, Certificate ARN, Certificate ownership token in the topic: `$aws/certificates/create-from-csr/<payload-format>/accepted`.
   Payload:

```
{
    "certificateOwnershipToken": "string",
    "certificateId": "string",
    "certificatePem": "string"
}
```

This certificate is unique for a device, and it is saved in the device securely.

- If the request is not accepted, then the response is received in the topic: `$aws/certificates/create-from-csr/<payload-format>/rejected`.
  Payload:

```
{
    "statusCode": int,
    "errorCode": "string",
    "errorMessage": "string"
}
```

6. Device calls RegisterThing, publishes to `$aws/provisioning-templates/<template-name>/provision/<payload-format>`.
   Payload:

```
{
    "certificateOwnershipToken": "string",
    "parameters": {
        "string": "string",
        ...
    }
}
```

- The register thing response is received in the topic: `$aws/provisioning-templates/<templateName>/provision/<payload-format>/accepted`.
  Payload:

```
{
    "deviceConfiguration": {
        "string": "string",
        ...
    },
    "thingName": "string"
}
```

- If the register thing request fails, then error response is received in the topic: `$aws/provisioning-templates/<templateName>/provision/<payload-format>/rejected`.
  Payload:

```
{
    "statusCode": int,
    "errorCode": "string",
    "errorMessage": "string"
}
```

7. AWS creates the Thing, attaches certificate, attaches policy, and activates certificate.
   Now the device is ready to disconnect and connect again using the provisioned unique certificate and key.

7.2 AWS Settings for Fleet Provisioning

To use the Fleet Provisioning feature of AWS IoT Core, you must set up an IAM role and a Provisioning Template in your AWS account. These AWS resources can be set up either through the AWS console or programmatically through the AWS CLI.

To set up of these resources using the AWS CLI:

NOTE
In the following example commands, replace the <aws-region> and <aws-account-id> with the AWS Region and ID relevant to your AWS account.

1. Clone the open-source FreeRTOS github repository: `https://github.com/FreeRTOS/FreeRTOS.git`.
2. In `FreeRTOS-Plus/Demo/AWS/Fleet_Provisioning_Windows_Simulator/CSR_Demo`, navigate to the demo subfolder.
3. Login to the AWS account:

```
aws login
```

4. Create an IAM role:

```
aws iam create-role --role-name "FleetProvisioningDemoRole" --assume-role-policy-document '
{"Version":"2012-10-17","Statement":[{"Action":"sts:AssumeRole","Effect":"Allow","Principal":
{"Service":"iot.amazonaws.com"}}]}'
```

5. Attach the AWSIoTThingsRegistration policy to the IAM role created. This allows the role to register new AWS IoT Things.

```
aws iam attach-role-policy --role-name "FleetProvisioningDemoRole" --policy-arn
arn:aws:iam::aws:policy/service-role/AWSIoTThingsRegistration
```

6. Create an AWS IoT Policy which the Fleet Provisioning Claim attaches to newly created things. An example IoT thing policy which you can modify to work with the demo can be found in the `example_iot_thing_policy.json` file.
   a. Modify the `example_iot_thing_policy.json` file by replacing all occurrences of <aws-region> with the AWS region of your choice (for example, us-west-2) and <aws-account-id> with your AWS account ID.
   b. Run the following command:

```
aws iot create-policy --policy-name FleetProvisioningDemoThingPolicy --policy-document
file://example_iot_thing_policy.json
```

7. Create an IoT Thing Type. This Thing Type is attached to all things created by the Fleet Provisioning demo, which allows for easy cleanup.

```
aws iot create-thing-type --thing-type-name "fp_demo_things"
```

8. Create the template resource which is used for provisioning the demo application.
   This needs to be done only once. An example fleet provisioning template which works with the demo can be found in the `example_fleet_provisioning_template.json` file.

```
aws iot create-provisioning-template --template-name FleetProvisioningDemoTemplate --
provisioning-role-arn arn:aws:iam::<aws-account>:role/FleetProvisioningDemoRole --template-body
file://example_fleet_provisioning_template.json -enabled
```

9. After you made your fleet provisioning template, you can verify it was successfully created using the following CLI command.

```
aws iot describe-provisioning-template --template-name FleetProvisioningDemoTemplate
```

10. Create a claim certificate and private key to use for the Provisioning by Claim workflow in the demo. In the command's output, remember the `certificateId` as it is used in step 12.

```
aws iot create-keys-and-certificate --certificate-pem-outfile "ClaimCertificate.pem" --public-
key-outfile "ClaimPubKey.pem" --private-key-outfile "ClaimPrivateKey.pem" --set-as-active
```

11. Create an IoT policy for the Claim certificate. An example Claim certificate policy can be found in the `example_claim_policy.json` file.
    a. Modify `example_claim_policy.json` by replacing all occurrences of `<aws-region>` with the AWS region of your choice (for example, us-west-2) and `<aws-account-id>` with your AWS account ID.
    b. Run the following command:

```
aws iot create-policy --policy-name FleetProvisioningDemoClaimPolicy --policy-document
file://example_claim_policy.json
```

12. Attach the policy to the claim certificate by replacing `<Claim-Cert-ID>` with the certificate ID of the Claim Certificate that you created in Step 10.

```
aws iot attach-policy --target "arn:aws:iot:<aws-region>:<aws-account-id>:cert/<Claim-Cert-
ID>" --policy-name "FleetProvisioningDemoClaimPolicy"
```

7.3 Load AWS-IoT Fleet Provisioning Reference Application in e^2 studio

The project template can be downloaded as a zip file, `r19an0471ee0100-rafw-group-aws-fleet-provisioning.zip`, from the Software and Tools section on the Renesas RA6W1 product page.
For more information on how to load the template, see the Load Project Template in e^2 studio section of *RA6W1 Application Development Guide*.

7.4 Configure Reference Application Parameters

You can configure the following parameters for Fleet Provisioning reference application using CLI or AT commands:

- **Device ID**: the unique identifier for each device in production. This ID is used to generate CSR and Client Identifier, and also to generate register thing request which after an accepted response from AWS cloud generates the Thing Name.
  CLI Command: `nvram setenv appcfg fp_dev_id <device_id>`
  AT Command: `AT+AWS=SET,DEVICE_ID,<device_id>`
- **Fleet Provisioning Template Name**: the template name registered in AWS cloud used for fleet provisioning.
  CLI Command: `nvram setenv appcfg fp_tmpl_name <template_name>`
  AT Command: `AT+AWS=SET,FP_TMPL_NAME,<template_name>`

- **AWS Broker URL**: the unique MQTT endpoint assigned to the AWS account.
  CLI Command: `nvram setenv appcfg broker_url <broker_url>`
  AT Command: `AT+AWS=SET,AWS_BROKER,<broker_url>`
- **AWS Port**: the port over which MQTT connection is established.
  CLI Command: `nvram setenv appcfg port <port>`
  AT Command: `AT+AWS=SET,PORT,<port>`

7.5 Build and Flash the Image

For more information on how to flash the image, see the Load Project Template in e^2 studio and the Flash and Debug using e^2 studio sections of *RA6W1 Application Development Guide*.

7.6 Run Fleet Provisioning Reference Application

1. During the initial bootup, in Wi-Fi setup menu, modify Wi-Fi details for internet connection.
2. Write the RootCA, claim a certificate, and claim a private key into the NVRAM using CLI commands:

```
net cert write ca6: RootCA
net cert write initcert6: Claim Certificate
net cert write initkey6: Claim Private Key
```

3. Configure reference application parameters: Device ID, Fleet Provisioning Template Name, AWS Broker URL, and MQTT Port. See Section 7.4 Configure Reference Application Parameters.
4. In AWS IoT console in web browser, open the MQTT test client and subscribe to the topic: `$aws/things/<thingname>/shadow/update/delta`.

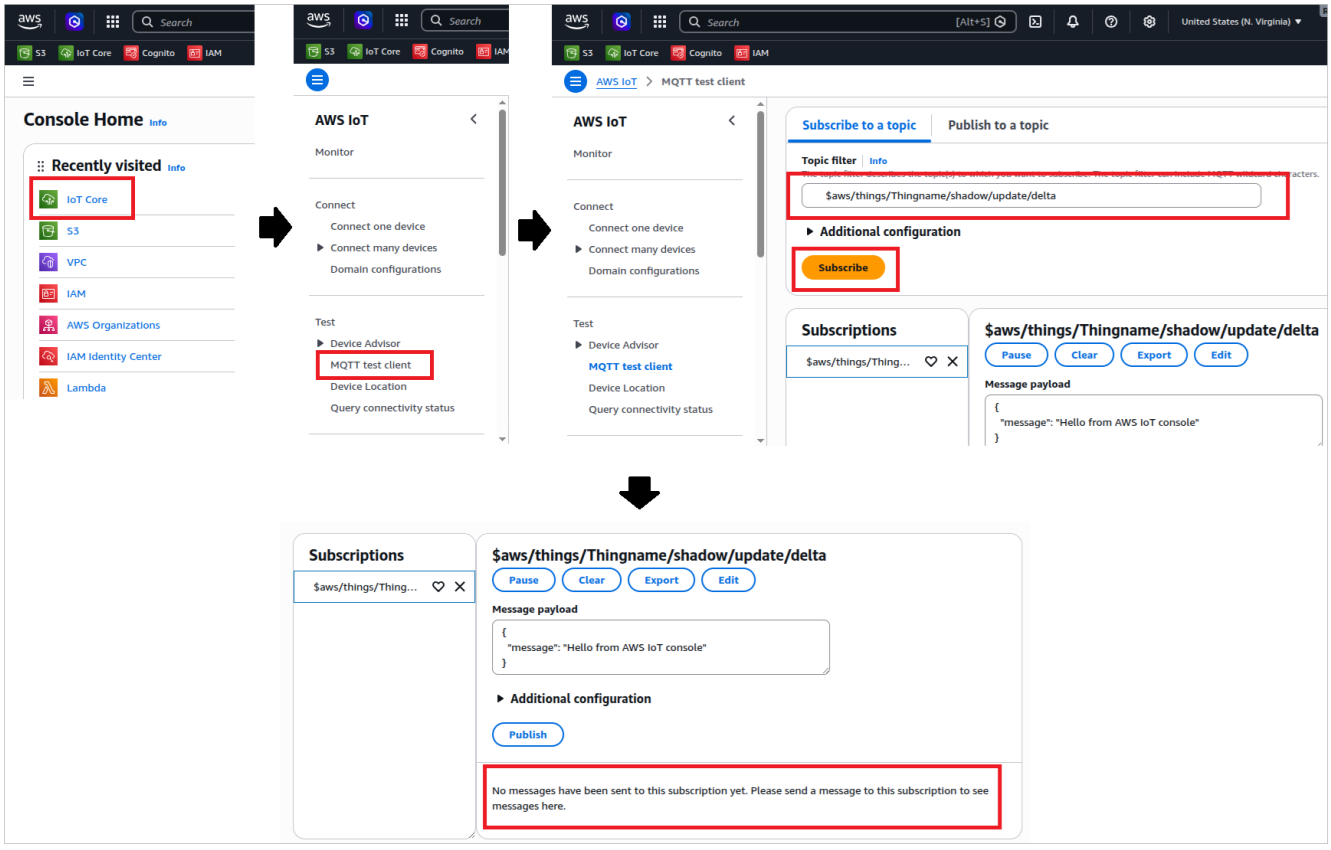

Figure 26. AWS MQTT test client

5. Reboot the device using the `reboot` CLI command or the reset button.
   After reboot, RA6W1 connects to the configured AP.

After connection to AP is successful, RA6W1 checks for the AWS IoT provisioned credentials. If it is available, then RA6W1 goes directly for AWS IoT Core connection using the provisioned credentials.
If not, then RA6W1 initiates the Fleet Provisioning by Claim process. Because this is initial bootup, RA6W1 goes for Fleet Provisioning.
If everything goes well, fleet provisioning is completed successfully.
After Fleet provisioning is completed, the unique certificate received from AWS Core and unique key generated locally are saved in the NVRAM, and RA6W1 disconnects itself from AWS IoT Core and reconnects using the provisioned credentials.
After the Fleet Provisioning is completed, it publishes a message to the topic:
`$aws/things/<thingname>/shadow/update/delta`.
Now, because the fleet provisioning is completed, if RA6W1 is rebooting, it directly connects to AWS IoT Core using the provisioned credentials and publishes to the same topic.

8. Revision History

Revision	Date	Description
1.03	July 21, 2026	Added minor changes based on the latest FSP release such as document title updates.
1.02	June 3, 2026	Added the Reference Application for AWS Fleet Provisioning section and the AWS Settings for Fleet Provisioning section. Modified the Load AWS IoT Reference Application in e^2 studio section and the Provision Wi-Fi and Configure AWS IoT Parameters section.
1.01	Dec 18, 2025	Modified the Build and Run Reference Application section. Added the Reference Application using AT Commands section. Added the FSP APIs section.
1.00	Aug 31, 2025	Initial version.

IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD-PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers who are designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only to develop an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third-party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising from your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Disclaimer Rev.1.01)

CorporateHeadquarters

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-us/