Using AWS IoT Door Lock Reference Application with AT Commands in RAFW Connectivity Devices

This document explains using the reference application for door lock with the AWS IoT Core and a mobile application to control the door lock that is interfaced with an external MCU.

Contents

1. Introduction **4**
2. System Overview **5**
2.1 Software Requirements 5
2.2 Hardware Requirements 5
3. Set Up and Configure Hardware **6**
4. Reference Application Setup **7**
4.1 Prerequisites for RA6W1 Wi-Fi Provisioning 7
4.2 Program Firmware Image 7
4.3 Prepare AWS Account 7
4.4 Prepare External MCU/TTL Scripts 7
4.4.1 Configure AWS Parameters 7
4.4.2 Handle Door Actions 10
5. AWS IoT AT Door Lock Example **13**
5.1 Provisioning Wi-Fi 13
5.1.1 Provision Wi-Fi in RA6W1 13
5.1.2 Provision Wi-Fi in RA6W2 13
5.2 Door Lock Application 15
5.2.1 Door Opening Process 16
5.2.2 Door Closing Process 19
5.2.3 Shadow Update Process 21
5.3 AT Command List 22
6. Revision History **25**

Figures

Figure 1. Architecture of AWS IoT reference application using AT commands 4
Figure 3. AWS configuration using AT commands 8
Figure 4. AT Sequence for door actions and shadow update 10
Figure 5. Door open AT sequence with DPM enabled 11
Figure 6. Renesas Wi-Fi provisioning application steps 14
Figure 7. Open AWS IoT door lock application 15
Figure 8. Application for AWS IoT with AT command 16
Figure 9. Opening door on mobile app with host MCU 17
Figure 10. Closing door on mobile app with host MCU 19
Figure 11. Shadow update on mobile app 21

© 2026 Renesas Electronics

Tables

Table 1. Basic set of AT commands from MCU to RA6W1/RA6W2 22
Table 2. Write TLS certificate from MCU to RA6W1/RA6W2 23
Table 3. Configuration data from MCU to RA6W1/RA6W2 23
Table 4. Command of MCU to RA6W1/RA6W2 23
Table 5. Response From RA6W1/RA6W2 to MCU 24
Table 6. Status from RA6W1/RA6W2 to MCU 24

Terms and Definitions

AP	Access point
Arm	Advanced RISC Machine
AT	Attention (AT commands)
AWS	Amazon Web Services
CA	Certificate Authority
COM	Communications
DPM	Dynamic Power Management
EVK	Evaluation Kit
FSP	Flexible Software Package
GCC	GNU Compiler Collection.
GND	Ground
IDE	Integrated Development Environment
IoT	Internet of Things
MCU	Microcontroller Unit
MQTT	Message Queuing Telemetry Transport
OTA	Over-The-Air
PC	Personal Computer
Soft-AP	Software-enabled Access Point
SSID	Service Set Identifier.
STA	Station
TTL	Tera Term Language
UART	Universal Asynchronous Receiver-Transmitter
URL	Uniform Resource Locator
USB	Universal Serial Bus
Wi-Fi®	Wireless Fidelity

1. Introduction

To start using AWS IoT connectivity on the RA6W1/RA6W2 device with AT command implementation for interfacing with an external MCU, begin with a ready-to-use reference application. The provided application project template, `awsiot_wifi_at_lock_ek_ra6w1` and `awsiot_wifi_ble_at_lock_ek_ra6w2` includes pre-configured project settings, middleware, and example code, which helps save time and ensures that the environment is correctly set up for AWS IoT development along with UART AT interface. This application demonstrates a cloud-connected door lock system in which the application runs on the RA6W1/RA6W2 device, while the actual door lock hardware is controlled by an external MCU.

The RA6W1/RA6W2 device connects to a Wi-Fi network and establishes a secure MQTT connection to AWS IoT Core. A mobile application communicates with AWS IoT Core over the internet to send door lock or unlock commands. These commands are received by the RA6W1/RA6W2 through AWS IoT, and the RA6W1/RA6W2 forwards the corresponding lock or unlock instructions to the external MCU using AT commands over a UART interface. The external MCU processes these AT commands and performs the final door lock or unlock action on the physical hardware.

The following components shown in Figure 1 are the main components of AWS IoT AT reference application:

- RA6W1/RA6W2 EVK
- AWS IoT – ATCMD reference application package
- External MCU EVK (for example: RA6M4)
- Terminal application that can run ttl scripts (if external MCU is not used)
- Router: Connection to internet
- Mobile device: Android/iOS application
- AWS account.

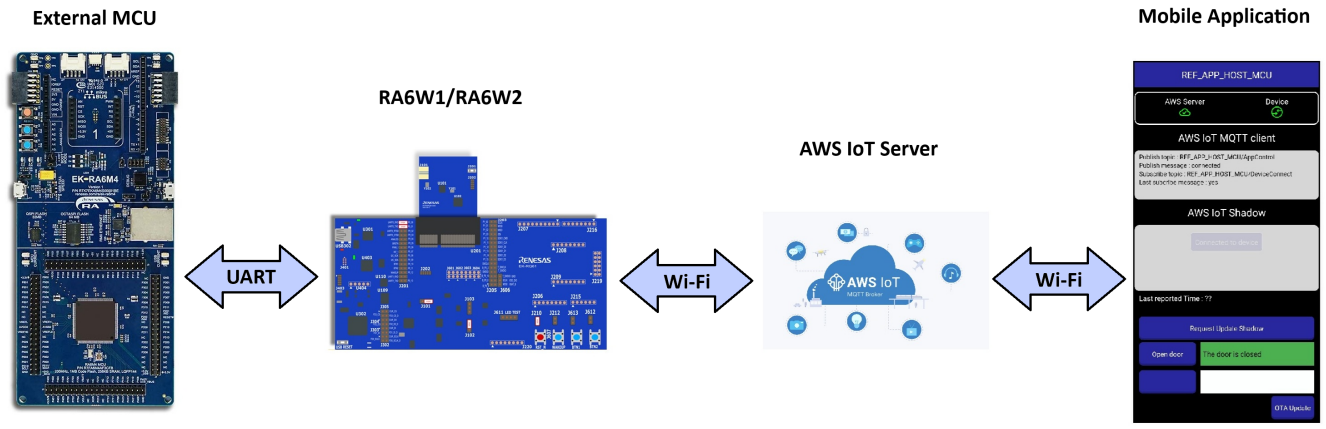


Figure 1. Architecture of AWS IoT reference application using AT commands

2. System Overview

The example project showcases creating an application with the RA6W1/RA6W2 using the AWS IoT platform with an AT UART interface. It implements a cloud-connected door lock system. The mobile application communicates with AWS IoT Core through the internet using MQTT to subscribe and publish to MQTT topics for remotely controlling the door lock using RA6W1/RA6W2 and MCU, which the door lock hardware is interfaced. The application in RA6W1/RA6W2 also uses the subscribe and publish mechanism to give the status or information of the door lock and to get commands from the application.
This example also uses the Wi-Fi provisioning method to assign the AP details for internet access to RA6W1/RA6W2 using the same mobile application. In this same process, the mobile application collects the AWS application mode (AT/ non-AT) with the Thing Name of the device that is configured by the user.
For provisioning in RA6W1, users must configure it in Provisioning mode, which is explained in the following sections. In Provisioning mode, the RA6W1 temporarily creates a Wi-Fi AP with a predefined SSID. Mobile applications connect to this access point and access a provisioning interface to configure network credentials. After provisioning is complete, the device reboots itself in STA mode, connects to the configured AP, and starts the normal functionality of the door lock application.
For provisioning in RA6W2, the application uses Bluetooth LE for a connection with the mobile application, which is explained in the following sections. In Provisioning mode, the RA6W2 creates a Bluetooth LE advertisement with a predefined name. Mobile applications connect to this Bluetooth LE and access a provisioning interface to configure network credentials. After provisioning is complete, the device reboots itself, connects to the configured AP, and starts the normal functionality of the door lock application.

2.1 Software Requirements

The following software tools and packages are required to complete the project:

- Renesas Electronics Flexible Software Package (FSP): Version 2.2.0 or later.
- e^2 studio IDE: Version 2026-04.3 or later.
- GCC Arm Embedded Toolchain: Version 13.3.1.arm-13-24.
- Serial Terminal Application: Tera Term or similar tools for logging and monitoring output.

2.2 Hardware Requirements

- Renesas RA6W1/RA6W2 Evaluation Kit (EVK).
- RA6W1/RA6W2 MCUwith ultra-low power consumption.
- USB Type-C Cable: For programming and debugging.
- Jumper Wire (Female to Female): For making proper motherboard configurations for the application.
- Host PC: For programming, viewing console logs and giving AT commandsFor programming, viewing console logsFor programming, viewing console logs, opening MQTT test client and giving AT commands.
- Wi-Fi Access Point (AP): With network connectivity.

3. Set Up and Configure Hardware

To connect hardware:

1. Connect the RA6W1/RA6W2 MCU to the Evaluation Kit (EVK) using the extension socket.
2. Connect the USB Debug port on the EVK to the host PC for programming and debugging using a USB Type-C cable.
3. Ensure the following jumper settings are configured correctly:
   - Follow the jumper connection instructions for AT commands as shown in Figure 2.

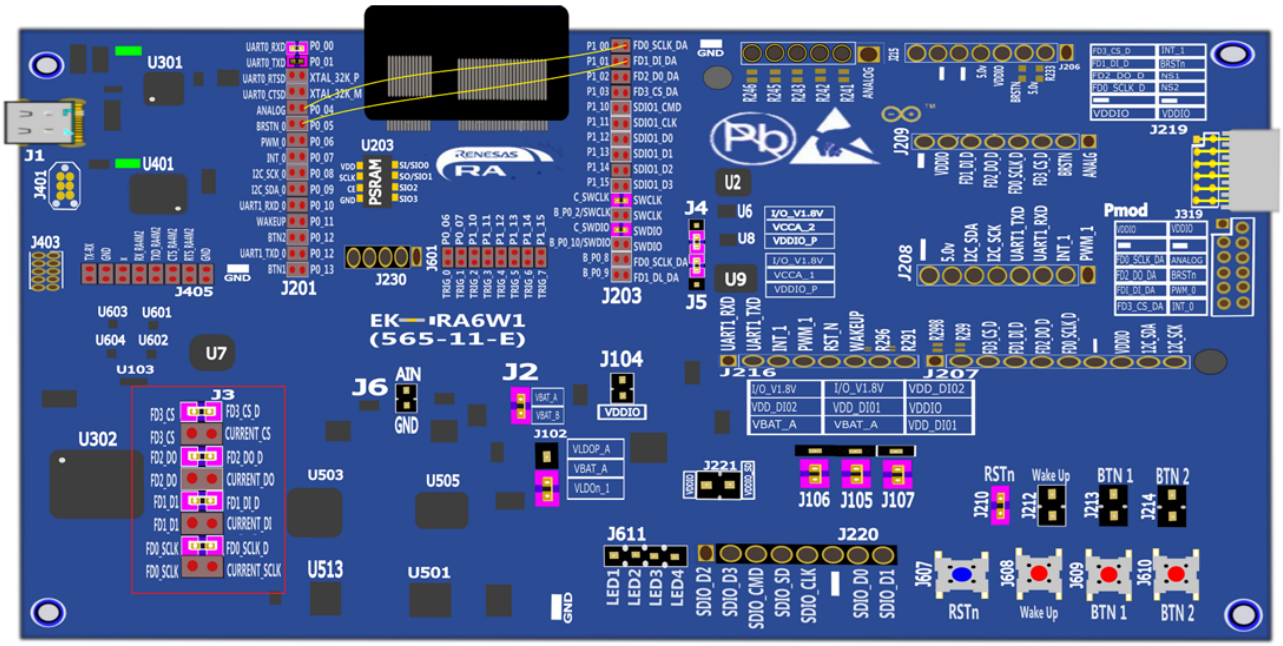


Figure 2. AT commands jumper connections

- Additionally for RA6W1, connect BTN1 and P0_10. This configuration is used for the Provisioning mode.

4. Reference Application Setup

4.1 Prerequisites for RA6W1 Wi-Fi Provisioning

Follow the steps in the Prerequisites for Soft AP-Based Wi-Fi Provisioning section in *RA6W1 Wi-Fi Provisioning Application Note*.

4.2 Program Firmware Image

The firmware image for the reference application is generated and flashed to the EK-RA6W1 using e^2 studio with J-Link support. e^2 studio provides an integrated build and flash environment, eliminating the need for external flash programming scripts.
Flashing procedure:

1. Open e² studio and connect the EK-RA6W1 to the PC through USB (using the inbuilt SEGGER J-Link).
2. Load the reference application to e^2 studio. The path for AWS IoT Door Lock reference application is `application_projects/r19an0439/AWS_AT_Reference_Application/`.
   For more information on how to load the template, see the Load Project Template in e^2 studio section in *RA6W1/RRQ61001 Application Development* or *RA6W2/RRQ61051 Application Development*.
3. Flash the image using debugger.
   For more information, see the Flash and Debug using e^2 studio section in *RA6W1/RRQ61001 Application Development* or *RA6W2/RRQ61051 Application Development*.
   When flashing is complete, the device starts running the firmware.

4.3 Prepare AWS Account

Follow the steps in the Connect Devices to AWS IoT section in *RA6W1/RRQ61001 AWS IoT Core Quick Start Guide* or *RA6W2/RRQ61051 AWS IoT Core Quick Start Guide*.

4.4 Prepare External MCU/TTL Scripts

Simulate the functions of external MCU using TTL scripts that can be loaded to terminal applications. TTL scripts can be written to send AT commands as response for the AT commands from RA6W1/RA6W2. If an external MCU must be used, it should flash with the code which functions like the TTL scripts.
The TTL script supports the following two functionalities:

- AWS Parameters (Section 4.4.1 Configure AWS Parameters)
- Door Actions (Section 4.4.2 Handle Door Actions).

4.4.1 Configure AWS Parameters

The following parameters need to be configured:

- Unique Thing Name (created in the Connect Devices to AWS IoT section in *RA6W1/RRQ61001 AWS IoT Core Quick Start Guide* or *RA6W2/RRQ61051 AWS IoT Core Quick Start Guide*).
  In AWS IoT Core, a Thing Name is the unique identifier assigned to each IoT device (called a Thing) registered in the cloud. AWS IoT uses the Thing Name to identify and manage a specific device. If two devices use the same Thing Name, AWS IoT cannot reliably distinguish between them. So, each RA6W1/RA6W2 under the test must have a unique Thing Name.
- AWS Broker URL (Get the endpoint from IoT Core in AWS account).
- MQTT topic which the application Publishes and Subscribes to.
- Root CA, Certificate and Private Key (see the Create and Activate Device Certificate section in *RA6W1IRRQ61001 AWS IoT Core Quick Start Guide* or *RA6W2/RRQ61051 AWS IoT Core Quick Start Guide*).
- Thing attributes (shown in TTL script).

These parameters can be configured using the AT commands. Figure 3 explains the AT commands for the configurations and its responses. See Section 5.3 AT Command List.

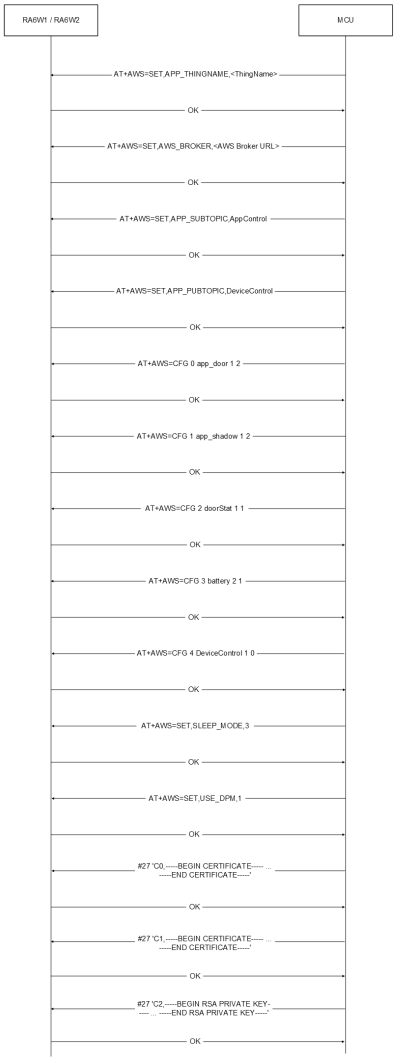


Figure 3. AWS configuration using AT commands

The following shows a TTL script that can be used for configuration.

```
; Wait for entering in provisioning mode
wait '+ATPROV=STATUS 1'
; Configure Thing name
sendln 'AT+AWS=SET,APP_THINGNAME,RA6Wx_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'
; Set Sleep Mode (currently only mode 3 is supported)
sendln 'AT+AWS=SET,SLEEP_MODE,3'
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 'rqXRfboQnoZsG4q5WTP468SQvvG5'
sendln '-----END CERTIFICATE-----'
send #3
waitln 'OK'
MPause 500
;; Configure Client Certificate
sendln #27 'C1,-----BEGIN CERTIFICATE-----'
sendln 'MIIDWjCCAkKgAwIBAgIVAKQVbt137DPvbOJzpLZM84OFuaFhMA0GCSqGSIb3DQEB'
……..
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 'wXQBS62JONf9Vog2f6Z1yjspl5W7YmO8xwyEduRkRoY0zzS77QS06w=='
sendln '-----END RSA PRIVATE KEY-----'
send #3
waitln 'OK'
```

4.4.2 Handle Door Actions

Figure 4 explains the AT commands sequence for handling the door actions and shadow update on both the RA6W1/RA6W2 and an external MCU side. See Section 5.3 AT Command List.

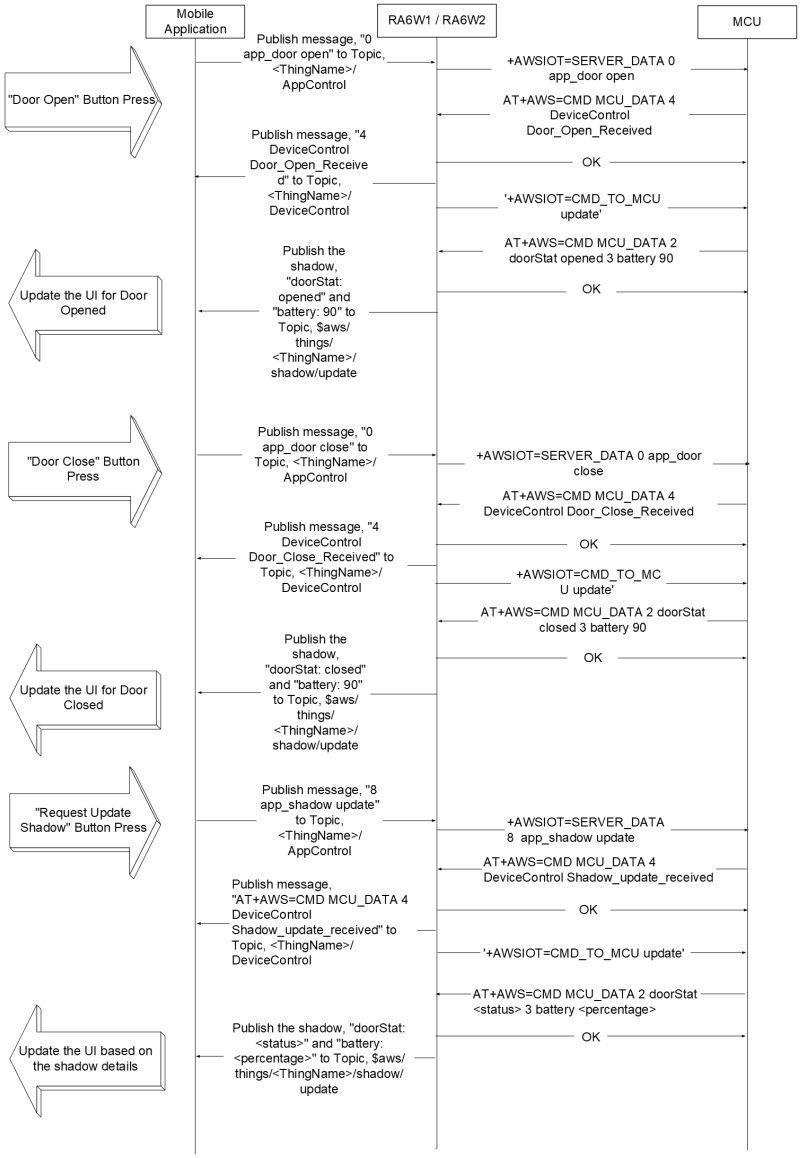


Figure 4. AT Sequence for door actions and shadow update

If the DPM is enabled, additional AT commands for handling DPM must be included. Figure 5 explains the AT command sequence for the Door Opening action with DPM enabled. For other actions, a similar sequence must be followed. For more information on the AT commands for DPM, see the Power Manager Commands section in the *AT Commands Manual*.

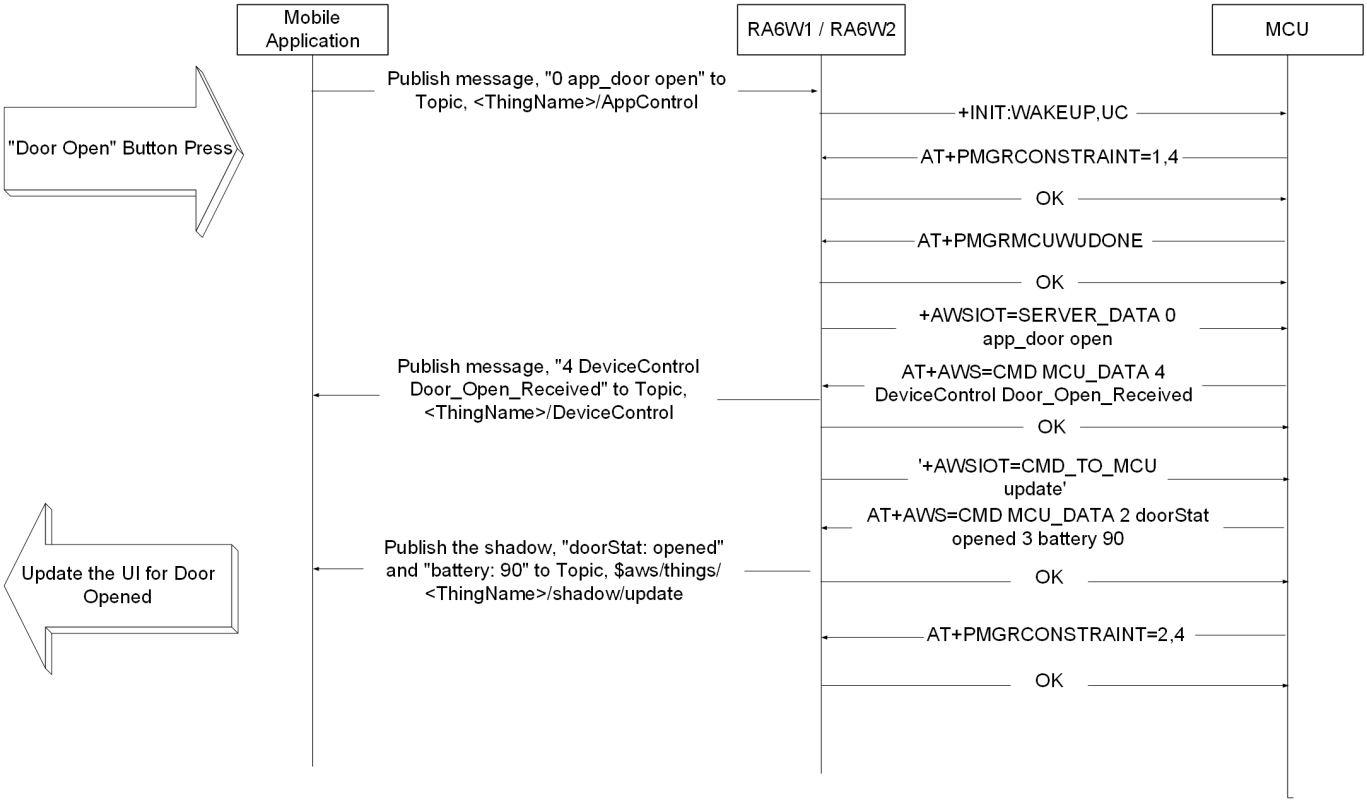


Figure 5. Door open AT sequence with DPM enabled

The following TTL script is a good reference for how the external MCU application should handle the AT commands for door open, close, and shadow update actions.

```
; Handles door open/close and resends last command on shadow update
; Initialize variable
strlastCommand = 'AT+AWS=CMD MCU_DATA 2 doorStat closed 3 battery 98'
doorLockCmdRcvd = 0
:loop
wait '+AWSIOT=SERVER_DATA 0 app_door close' '+AWSIOT=SERVER_DATA 0 app_door open' '+AWSIOT=SERVER_
DATA 8 app_shadow update' '+AWSIOT=CMD_TO_MCU
update' '+INIT:WAKEUP,UC' '+INIT:WAKEUP,DEAUTH' '+INIT:WAKEUP,NOACK'
if result = 1 then
    ; Door closed
    strlastCommand = 'AT+AWS=CMD MCU_DATA 2 doorStat closed 3 battery 80'
    doorLockCmdRcvd = 1
    sendln 'AT+AWS=CMD MCU_DATA 4 DeviceControl Door_Close_Received'
endif
if result = 2 then
    ; Door opened
    strlastCommand = 'AT+AWS=CMD MCU_DATA 2 doorStat opened 3 battery 90'
    doorLockCmdRcvd = 1
    sendln 'AT+AWS=CMD MCU_DATA 4 DeviceControl Door_Open_Received'
endif
if result = 3 then
    ; Resend last known command
    doorLockCmdRcvd = 1
    sendln 'AT+AWS=CMD MCU_DATA 4 DeviceControl Shadow_update_received'
endif
if result = 4 then
    ; Resend last known command
```

```
    sendln strlastCommand
    waitln 'OK'
    if doorLockCmdRcvd = 1 then
        sendln 'AT+PMGRCONSTRAINT=2,4'
        waitln 'OK'
        doorLockCmdRcvd = 0
    endif
endif
if (result = 5) || (result = 6) then
    ; Add constraint and notify MCU ready for AT
    sendln 'AT+PMGRCONSTRAINT=1,4'
    ;waitln 'OK'
    sendln 'AT+PMGRMCUWUDONE'
    waitln 'OK'
    sendln strlastCommand
    waitln 'OK'
endif
if result = 7 then
    sendln 'AT+PMGRMCUWUDONE'
    waitln 'OK'
    sendln strlastCommand
    waitln 'OK'
endif
goto loop
```

5. AWS IoT AT Door Lock Example

5.1 Provisioning Wi-Fi

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 application using either an Android or an iOS device.

5.1.1 Provision Wi-Fi in RA6W1

When RA6W1 initially starts after flashing the image, it is in Station mode.
To change the device to Provisioning mode and to provision Wi-Fi :

1. Press the factory reset button configured using the jumper connection in Section 3 Set Up and Configure Hardware. Now the device does the factory reset and bootup in Soft-AP mode and is ready for Wi-Fi provisioning
2. Configure the device with the AWS parameters from Section 4.4.1 Configure AWS Parameters.
3. To complete Wi-Fi provisioning, follow the Provision Wi-Fi through Soft AP Mode in the *RA6W1 Wi-Fi Provisioning Application Note* .

Wi-Fi Provisioning is now completed and RA6W1 reboots and starts in Station mode and connects to the Access Point configured from the application. After the successful connection to the internet, it connects to the AWS server.

5.1.2 Provision Wi-Fi in RA6W2

Wi-Fi Provisioning in RA6W2 is performed using Bluetooth LE and you do not need to change the device from Station mode to SoftAP mode. If Wi-Fi is not provisioned, Bluetooth LE is active and the Renesas Wi-Fi provisioning application can connect to it and configure RA6W2 with the Wi-Fi Access Point details while staying in Station mode.
To provision Wi-Fi:

1. Configure the device with the AWS parameters as mentioned in Section 4.4.1 Configure AWS Parameters.
2. Open the same Renesas Wi-Fi Provisioning app mentioned in the in *RA6W1 Wi-Fi Provisioning Application Note* .
3. Select **Start provisioning over Bluetooth Low Energy**.
4. Select the device **RRQ-DEVICE** from the scanned Bluetooth LE devices
   Next, the application reads the AWS IoT Thing Name and Operation mode (AT or non-AT), and it connects to the Bluetooth LE device.
5. Select **Scan Wi-Fi network**.
6. Select the required Access Point from the scanned list.
7. Enter the password and click **OK**.
8. Select **Connect to <AP Name>**.
   After the connection attempt, the application provides a network check result.
9. Review the details of network check result, and select **Complete**.
   Finally, the device reboots and connects to the configured Access Point.

Figure 6 explains the process with mobile application screenshots.



Figure 6. Renesas Wi-Fi provisioning application steps

5.2 Door Lock Application

After provisioning is completed, load the TTL script shown in Section 4.4.2 Handle Door Actions on the AT UART COM port
of RA6W1/RA6W2. If external MCU is used instead, ensure the interfacing with RA6W1/RA6W2 is correct.

- To open AWS application on a mobile device, select **AWS IoT** by clicking the hamburger menu icon (see Figure 7).

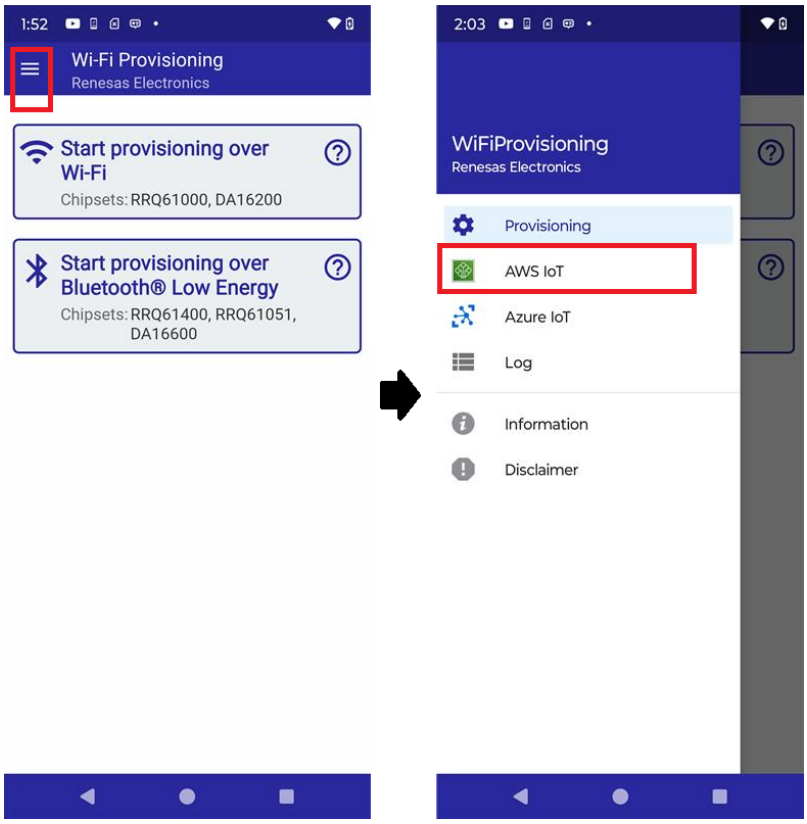


Figure 7. Open AWS IoT door lock application

- To open and close the door, click the **Open door** or **Close door** button (see Figure 8).
  The text in the button and the text box next to changes according to the state of the door. The left side of Figure 8 shows the closed door state and the right side shows the opened door state.
- To update the shadow information, click the **Request Update Shadow** button. The shadow information from RA6W1/RA6W2 appears on the window above the button.

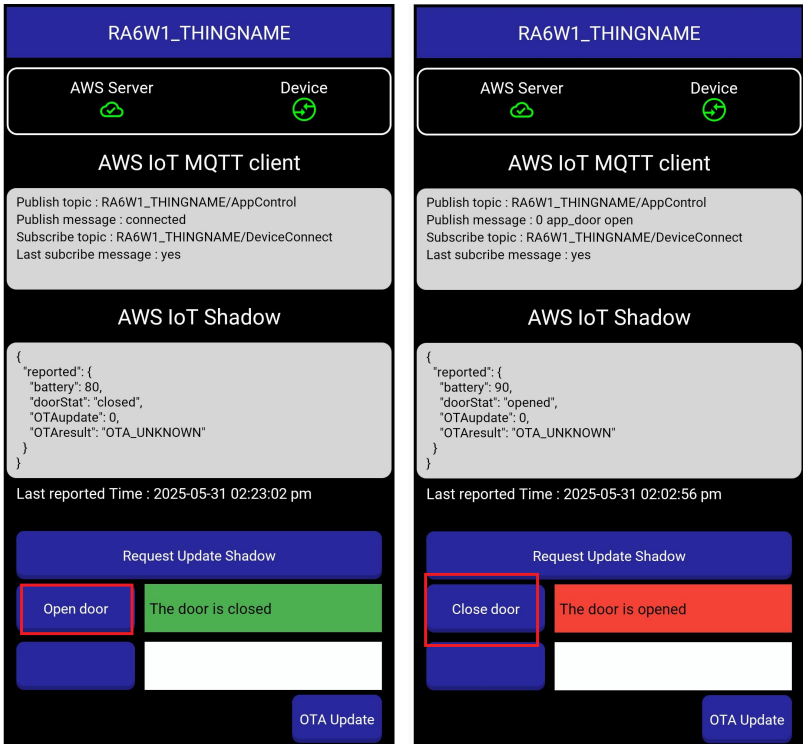


Figure 8. Application for AWS IoT with AT command

When RA6W1/RA6W2 boots up, it connects to the provisioned Wi-Fi Access Point. After successful connection to internet, the RA6W1/RA6W2 application connects to the AWS server after fetching the configured AWS IoT parameters.

If the mobile application successfully connects to the AWS server, the AWS server icon becomes solid green. When the application connects to AWS cloud, it publishes the message "connected" to the topic, <Thingname>/AppControl while RA6W1/RA6W2 is subscribing to it. Therefore, when RA6W1/RA6W2 receives this message it publishes a message "yes" to the topic <Thingname>/DeviceConnect, which is subscribed by the application. When the message "yes" is received on the application, the Device icon becomes solid green.

The initial connection of MCU, RA6W1/RA6W2, and Mobile application is now completed. If the device does not respond with the shadow update, the application notifies you that the device is not responding.

5.2.1 Door Opening Process

Figure 9 shows the operation of opening door on an Android application.

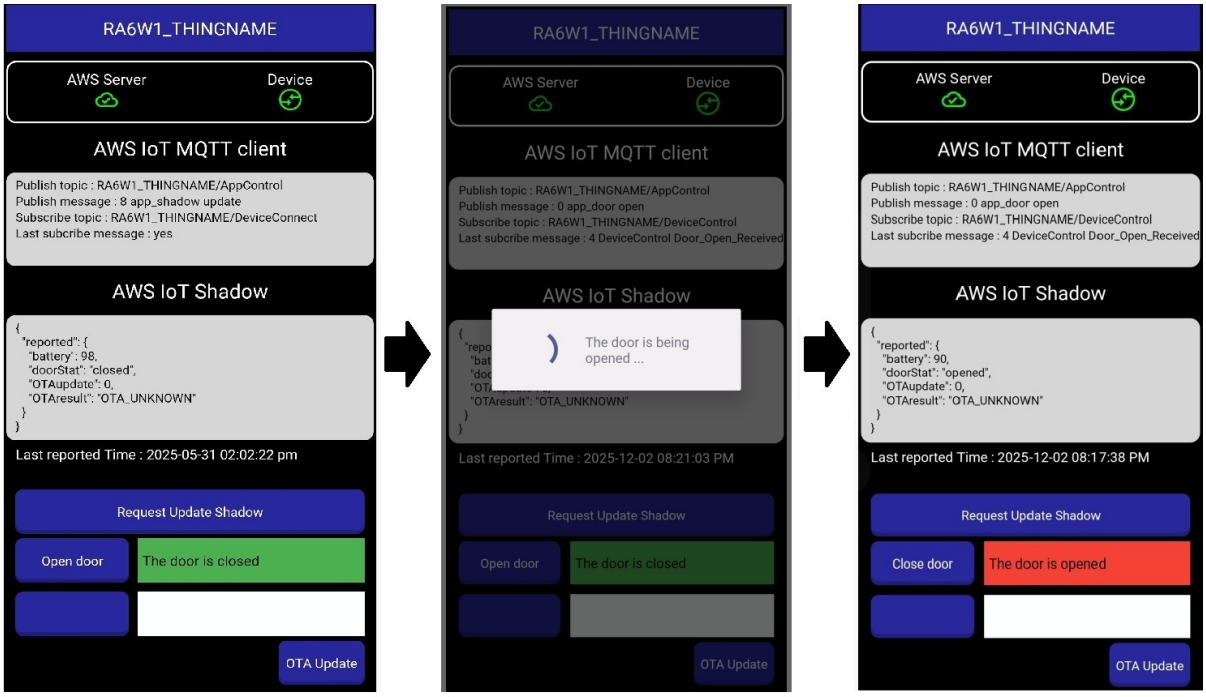


Figure 9. Opening door on mobile app with host MCU

Click the **Open door** button and this happens:

- Application publish message, 0 app_door open on the topic <Thingname>/AppControl.
- If the DPM is enabled, when RA6W1/RA6W2 receives the message, it sends the AT command +INIT:WAKEUP,UC. The MCU should keep the DPM from going to sleep by sending the command, AT+PMGRMCUWUDONE and AT+PMGRCONSTRAINT=1,4.
  Next, RA6W1/RA6W2 sends the command +AWSIOT=SERVER_DATA 0 app_door open, and it waits for a response until timeout.
- If the DPM is not enabled, RA6W1/RA6W2 directly sends the command +AWSIOT=SERVER_DATA 0 app_door open, and it waits for a response until timeout.
- The MCU responds with the command: AT+AWS=CMD MCU_DATA 4 DeviceControl <message>. The message in the command is updated in the application, because the application is subscribing to the topic <Thingname>/DeviceControl.
- Next, the MCU processes the command and the door opening. RA6W1/RA6W2 again sends the command +AWSIOT=CMD_TO_MCU update to get the status from the MCU, and it waits for a response until timeout.
- The MCU updates the shadow with the command AT+AWS=CMD MCU_DATA 2 doorStat opened 3 battery 90.
- RA6W1/RA6W2 parses this and publishes the shadow to the application, and the box under AWS IoT Shadow is updated according to the received values. Also, the state of door is changed in the application with the update to the shadow.

When the operation of opening door is completed, the console logs of the RA6W1/RA6W2 appear as follows:

```
pal_app_event_cb 0 app_door opene
dynamic subscription command ("app_door")

send "SERVER_DATA 0 app_door open" to MCU

=========================================================
argc num = 2
argv[0]: AT+AWS
argv[1]: CMD MCU_DATA 4 DeviceControl Door_Open_Received
```

```
=========================================================
topicCount : 1


Count : 0, cmdNum = 4
mqtttype = 0
data type(publish)=1
call publish: 4 DeviceControl Door_Open_Received
publish (contents update) OK - topic : RA6Wx_THINGNAME/DeviceControl payload: 4 DeviceControl Door_
Open_Received

release response

*****************************************************************************************

send "CMD_TO_MCU update" to MCU

=========================================================
argc num = 2
argv[0]: AT+AWS
argv[1]: CMD MCU_DATA 2 doorStat opened 3 battery 90

=========================================================
topicCount : 2


Count : 0, cmdNum = 2
mqtttype = 1
index(=1) matched
data type(shadow) = 1
call update sensor(need to be set variable): doorStat = opened


Count : 1, cmdNum = 3
mqtttype = 1
index(=0) matched
data type(shadow) = 2
call update sensor(need to be set variable): battery = 90.000000

release response


*****************************************************************************************
publish (shadow sensor update) OK -  payload: "{"state":{"reported":
{"battery":90.000000,"doorStat":"opened","OTAupdate":0,"OTAresult":"OTA_
UNKNOWN"}},"clientToken":"RA6Wx_THINGNAME-0"}"
last user Timer ID = 0
last doorOpenFlag state: "false"
last FOTA Stat: 0
last FOTA Url: ""
```

AT console log for door opening is the following:

```
+AWSIOT=SERVER_DATA 0 app_door open
>>>>> command: <AT+AWS=CMD MCU_DATA 4 DeviceControl Door_Open_Received> is given here <<<<<
OK
+AWSIOT=CMD_TO_MCU update
```

```
>>>>> command: <AT+AWS=CMD MCU_DATA 2 doorStat closed 3 battery 80> is given here <<<<<
OK
```

5.2.2 Door Closing Process

Figure 10 shows the operation of door close in the mobile application.

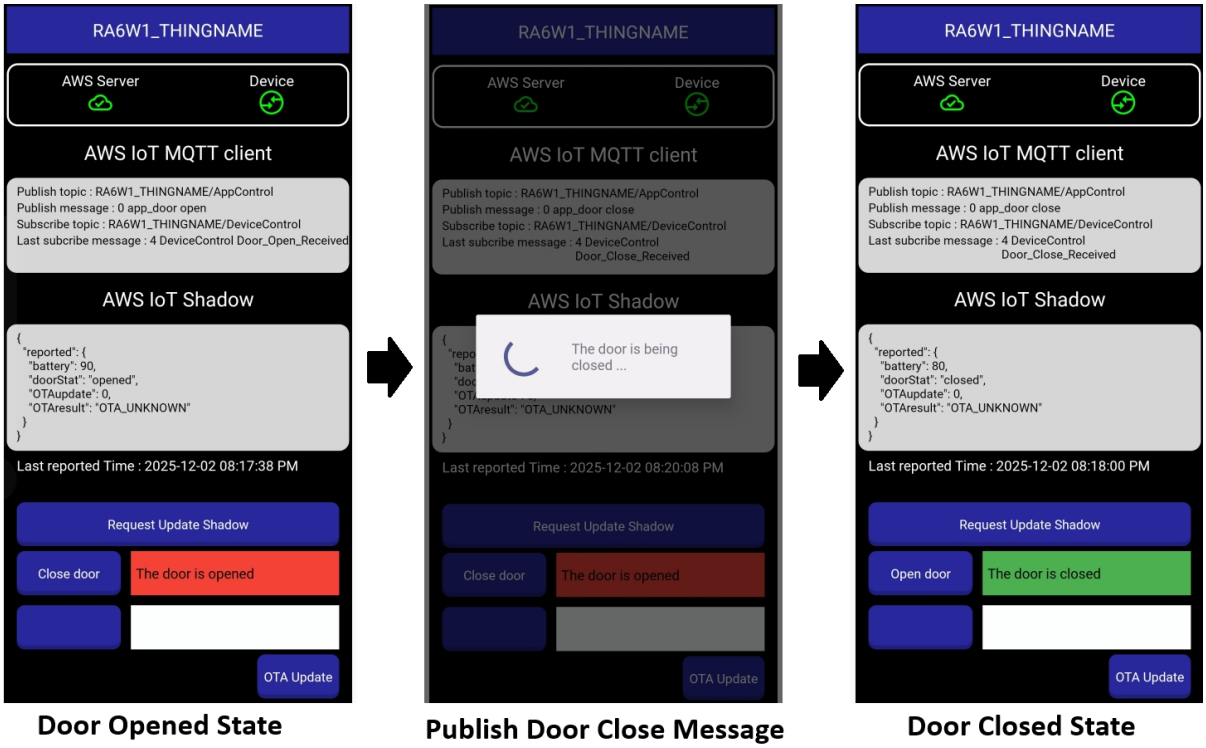


Figure 10. Closing door on mobile app with host MCU

Click the **Close door** button and this happens:

- Application publish message, 0 app_door close on the topic <Thingname>/AppControl.
- If the DPM is enabled, when RA6W1/RA6W2 receives the message, it sends an AT command, +INIT:WAKEUP,UC. MCU should now keep the DPM from going to sleep by sending the command, AT+PMGRMCUWUDONE and AT+PMGRCONSTRAINT=1,4.
  Now RA6W1/RA6W2 sends the command, +AWSIOT=SERVER_DATA 0 app_door close and wait for the response until timeout (Currently the waiting time is configured as 6 seconds. You can modify this time using the macro, MAX_RESPONSE_WAIT_COUNT in the file, `src/atcmd/app_atcommand_pal.c`).
- If the DPM is not enabled, RA6W1/RA6W2 sends the command, +AWSIOT=SERVER_DATA 0 app_door close, and it waits for the response until timeout.
- The MCU should respond with the command AT+AWS=CMD MCU_DATA 4 DeviceControl <message>. The message in the command is updated in the application, because the application is subscribing to the topic <Thingname>/DeviceControl.
- The MCU processes the command and completes the door opening. RA6W1/RA6W2 again sends a command +AWSIOT=CMD_TO_MCU update to get the status from the MCU, and it waits for the response until timeout.
- Next, the MCU updates the shadow with the command AT+AWS=CMD MCU_DATA 2 doorStat closed 3 battery 80.
- RA6W1/RA6W2 parses this and publishes the shadow to the application, and the box under AWS IoT Shadow is updated according to the received values. The state of door is changed in the application with the update to the shadow.

When the operation of closing door is completed, the console logs of the RA6W1/RA6W2 appears:

```
pal_app_event_cb 0 app_door close
dynamic subscription command ("app_door")

send "SERVER_DATA 0 app_door close" to MCU

=========================================================
argc num = 2
argv[0]: AT+AWS
argv[1]: CMD MCU_DATA 4 DeviceControl Door_Close_Received

=========================================================
topicCount : 1


Count : 0, cmdNum = 4
mqtttype = 0
data type(publish)=1
call publish: 4 DeviceControl Door_Close_Received
publish (contents update) OK - topic : RA6Wx_THINGNAME/DeviceControl payload: 4 DeviceControl Door_
Close_Received

release response

*****************************************************************************************

send "CMD_TO_MCU update" to MCU

=========================================================
argc num = 2
argv[0]: AT+AWS
argv[1]: CMD MCU_DATA 2 doorStat closed 3 battery 80

=========================================================
topicCount : 2


Count : 0, cmdNum = 2
mqtttype = 1
index(=1) matched
data type(shadow) = 1
call update sensor(need to be set variable): doorStat = closed


Count : 1, cmdNum = 3
mqtttype = 1
index(=0) matched
data type(shadow) = 2
call update sensor(need to be set variable): battery = 80.000000

release response


*****************************************************************************************
publish (shadow sensor update) OK -  payload: "{"state":{"reported":
{"battery":80.000000,"doorStat":"closed","OTAupdate":0,"OTAresult":"OTA_
UNKNOWN"}},"clientToken":"RA6Wx_THINGNAME-0"}"
last user Timer ID = 0
last doorOpenFlag state: "false"
last FOTA Stat: 0
last FOTA Url: ""
```

5.2.3 Shadow Update Process

Figure 11 shows the operation of shadow update in the mobile application.

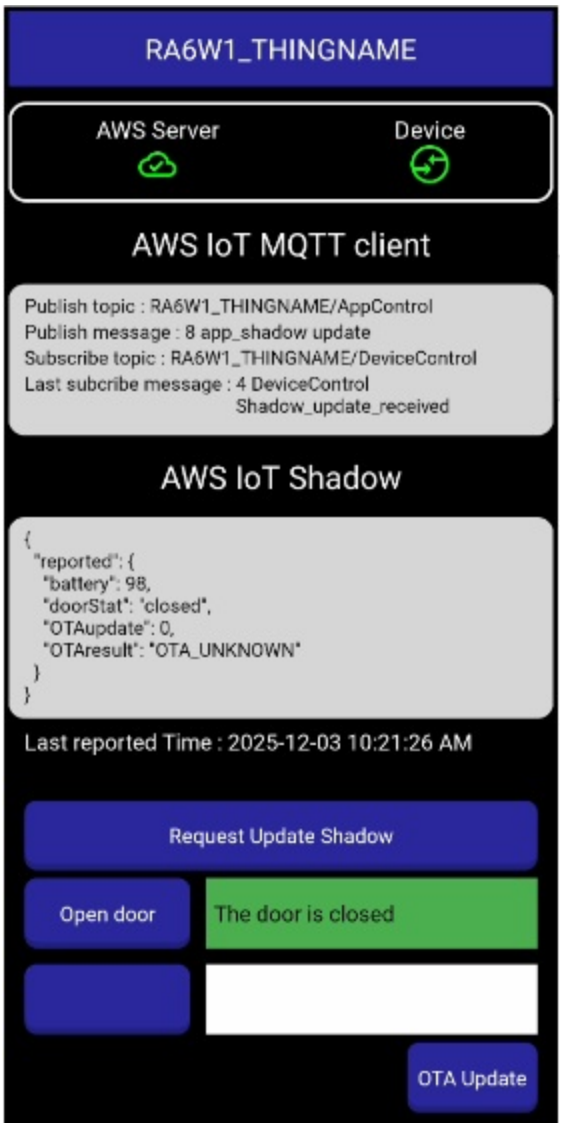


Figure 11. Shadow update on mobile app

Click the **Request Update Shadow** button and this happens:

- Application publish message, 8 app_shadow update on the topic thingName/AppControl.
- If the DPM is enabled, when RA6W1/RA6W2 receives the message, it sends an AT command, +INIT:WAKEUP,UC. The MCU should now keep the DPM from going to sleep by sending the command, AT+PMGRMCUWUDONE and AT+PMGRCONSTRAINT=1,1.
  Now RA6W1/RA6W2 sends the command, +AWSIOT=SERVER_DATA 8 app_shadow update and waits for response until timeout.
- If the DPM is not enabled, RA6W1/RA6W2 sends the command, +AWSIOT=SERVER_DATA 8 app_shadow update and waits for a response until timeout.
- The MCU should respond with the command: AT+AWS=CMD MCU_DATA 4 DeviceControl <message>. The message in the command is updated in the application, because the application is subscribing to the topic <Thingname>/DeviceControl.
- The MCU should now process the command and do the door opening. Next, RA6W1/RA6W2 again sends a command +AWSIOT=CMD_TO_MCU update to get the status from the MCU and wait for the response until timeout.

- The MCU updates the shadow with the command, AT+AWS=CMD MCU_DATA 2 doorStat closed 3 battery 80.
- RA6W1/RA6W2 parses this and publishes the shadow to the application, and the box under AWS IoT Shadow is updated according to the received values. If the doorStat is changed, the state of open door or close door also changes.

AT console for closing door is the follwoing:

```
+AWSIOT=SERVER_DATA 0 app_door close
>>>>> command: <AT+AWS=CMD MCU_DATA 4 DeviceControl Door_Close_Received> is given here <<<<<
OK
+AWSIOT=CMD_TO_MCU update
>>>>> command: <AT+AWS=CMD MCU_DATA 2 doorStat closed 3 battery 80> is given here <<<<<
OK
```

5.3 AT Command List

Table 1. Basic set of AT commands from MCU to RA6W1/RA6W2

Head	Main	Sub	Parameters
AT+AWS=	SET	APP_ THINGNAME	Set the device thing name. It is 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 subs 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. Currently only mode 3 is used.
		RTC_TIME	This command is not implemented. It can be used to set the wake-up time for not connected sleep. Where RA6W1/RA6W2 wakes 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 used currently. It can be used to set RA6W1/RA6W2 port to wakeup external MCU.
		APP_MCU_ WU_PIN	This command is not used currently. It can be used to set RA6W1/RA6W2 pin to wakeup external MCU.
Example: AT+AWS=SET APP_THINGNAME <Assigned Thing Name>			

Head	Main	Sub	Parameters
AT+AWS=SET AWS_BROKER a1kzdt4nun8bnh-ats.iot.ap-northeast-2.amazonaws.com			

Table 2. Write TLS certificate from MCU to RA6W1/RA6W2

Start Code	Sub Code	Type	End Code
0x1B (Esc character)	C0,	Root CA Self-Signed 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 key. send "C0,-----BEGIN CERTIFICATE-----\n“ " MIIDQTCCAimgAwIB ….. -----END CERTIFICATE-----” over UART or copy paste the certificate after C0, send “0x03” or press Ctrl+c key			

Table 3. Configuration data from MCU to RA6W1/RA6W2

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. Max 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 1 - Shadow: The value is sent to the device twin and is updated on the phone the next time it is connected. 2 - 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/RA6W2

Head	Main	Sub	Description
AT+AWS=	CMD	MCU_DATA	Used by the MCU to set a CFG parameter in theRA6W1/RA6W2. The value must be the same format as defined by the CFG setting. Parameters: [number] [name] [value]

Head	Main	Sub	Description
		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/RA6W2 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. Response From RA6W1/RA6W2 to MCU

Head	Main	Parameters	Description
+AWSIOT	SERVER_DATA	[number] [name]	Used by RA6W1/RA6W2 to set a CFG parameter in the MCU. The value must be the same format as defined by the CFG setting.
	CMD_TO_MCU	[value]	
Example: +AWSIOT SERVER_DATA 0 door_control open +AWSIOT CMD_TO_MCU update			

Table 6. Status from RA6W1/RA6W2 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 the factory reset completes.
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 AP mode is being started.
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. Revision History

Revision	Date	Description
1.02	Sep 21, 2026	Updated content for FSP version 2.2.0. Updated reference documents name.
1.01	July 21, 2026	Updated content for FSP version 2.1.0.
1.00	May 29, 2026	First 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/

© 2026 Renesas Electronics Corporation. All rights reserved.