RX Family

How to Implement AWS IoT Fleet Provisioning (202406-LTS Version)

Introduction

Connection to AWS IoT, a cloud service provided as part of Amazon Web Services™ (AWS), requires IoT device provisioning. The term "provisioning" used here refers to the process of generating, utilizing, and managing authentication information such as "things", private keys, and device certificates. Provisioning requires prior consideration of methods for writing authentication information to products in the manufacturing process (initial installation) and managing (protecting) and updating key data. These data are stored in the on-chip flash memory of a product from the RX family.
Since subsequent modification of the provisioning method for IoT devices is difficult, consideration of the above points must begin in the development stage of a product and verification must be completed before the mass production phase.

Of the various provisioning methods provided by AWS, descriptions in this document are based on a "fleet provisioning method," in which provisioning is automated in the manufacturing process and in the initial usage of the IoT device.

The fleet provisioning method eliminates the need to spend time and effort on cumbersome provisioning procedures while realizing more secure and convenient provisioning.

What you will learn in this application

- ✓ Outline of the methods of provisioning provided by AWS
- ✓ How to implement fleet provisioning and confirm operation by using the demo. The steps required to run the demo are described in section 4, Running the Fleet Provisioning Demo.

The contents of this document are sufficient to implement provisioning by simply following the methods described, but if this is done without modification, the result will be that important data saved in the provisioning process, such as the private key and device certificate, are stored as "clear text" (text with no encryption) in the on-chip flash memory of the MCU of the RX family. This means that if a user program stored in the MCU has a security hole in that it allows the reading of any desired area of memory, access to the provisioning data in the flash memory will be possible and this may allow an attacker to perform an unauthorized login to the user's AWS account.

Selecting an MCU of the RX family that includes the Trusted Secure IP (TSIP) module and using the module enables the storage of encrypted rather than clear private keys and device certificates, greatly reducing the danger of unauthorized access to the provisioning data. For details on the TSIP module, see the page at the following link.

https://www.renesas.com/software-tool/trusted-secure-ip-driver

We strongly encourage using the TSIP module to boost security.

Improving software quality can reduce the risk of leakage of provisioning data but this approach can never completely eliminate the risk. In particular, if the software of an IoT device has defects that expose it to threats by attackers, we recommend that the firmware update function be used to apply corrections in a timely manner.

Note: This application note shows an implementation example based on the operating environment of the CK-RX65N v2 board with Ethernet communications, but it can also be used with combinations of other boards and communications control modules, such as the RYZ014A PMOD module or DA16600 module (Wi-Fi).
For the combinations of boards and communications control modules, refer to the page at the following link.
[GitHub] iot-reference-rx/Getting_Started_Guide.md at main · renesas/iot-reference-rx (github.com)

Note: Renesas has announced discontinuation of production of the existing LTE module with the part number RYZ014A, and has announced the cessation of shipment of the product.
Accompanying cessation of shipment of the RYZ014A, the CK-RX65N v1 board will also no longer be shipped.
If the RYZ014A is in use for a current design or item in production, the Sequans product GM01Q is a pin and functionally compatible replacement for RYZ014A.
The EOL notice for the RYZ014A is given at the following link.
[Link]
https://www.renesas.com/document/eln/plc-240004-end-life-eol-process-select-part-numbers
[Product page]
https://www.renesas.com/ryz014a

Operating Environment

The operations described in this application note have been confirmed on the following environment.

Integrated development environment	e² studio 2025-12
Board	CK-RX65N v2
Toolchain	CC-RX Compiler v3.07.00 GCC for Renesas RX v14.2.0.202505
Emulator	E2OB (E2 Lite On Board) on CK-RX65N

Before applying the contents of this application note to another MCU, review the product-specific settings to suit the specifications of the MCU and thoroughly evaluate the results.

Related Application Notes

The documents listed below are related to this application note. Refer to them as necessary.

- How to Implement FreeRTOS OTA Using Amazon Web Services (202406-LTS Version) (R01AN7662)

Information about boards, related programs, and development environments that are necessary for developing RX cloud solutions is summarized on the page at the following link.

https://www.renesas.com/rx-cloud

Also, the following information officially released by AWS will be of use.

- Provisioning authentication information for devices in AWS IoT (only available in Japanese)
  Video: https://youtu.be/gcJwNEQ2eLY
  Document: https://pages.awscloud.com/rs/112-TZM-766/images/EV_iot-deepdive-aws2_Sep-2020.pdf
- Document on fleet provisioning templates
  https://docs.aws.amazon.com/iot/latest/developerguide/provision-template.html
- Document on AWS IoT Core policies
  https://docs.aws.amazon.com/iot/latest/developerguide/iot-policies.html
- AWS IoT API reference document: CreateCertificateFromCsr
  https://docs.aws.amazon.com/iot/latest/apireference/API_CreateCertificateFromCsr.html
- Provisioning devices that do not have device certificates using fleet provisioning
  https://docs.aws.amazon.com/iot/latest/developerguide/provision-wo-cert.html
- How to automate onboarding of IoT devices to AWS IoT Core at scale with Fleet Provisioning
  https://aws.amazon.com/blogs/iot/how-to-automate-onboarding-of-iot-devices-to-aws-iot-core-at-scale-with-fleet-provisioning/

Contents

1. Terminology 6

2. Device Provisioning 7
2.1 Methods of Provisioning with AWS IoT 8
2.2 Fleet Provisioning Methods 9
2.3 Provisioning by Claim (Approach Using a Provisioning Claim Certificate) 10
2.3.1 Overview of Provisioning by Claim (Using a Provisioning Claim Certificate) 11
2.3.2 Determining a Unique Name for a Thing 13

3. Preparation 14
3.1 Hardware Environment 14
3.2 Software Environment 14
3.3 Installing and Setting Tera Term 15
3.4 Installing the AWS Command Line Interface (CLI) 17
3.5 FreeRTOS Project 19

4. Running the Fleet Provisioning Demo 20
4.1 Preparing the Execution Environment 20
4.2 Preparing for AWS 21
4.3 AWS Settings for Fleet Provisioning 21
4.3.1 Creating a Policy for the Fleet Provisioning Demo 22
4.3.2 Creating Policies for Things Created through Fleet Provisioning 24
4.3.3 Generating a Claim Certificate and a Claim Key Pair 26
4.3.4 Creating a Role 28
4.3.5 Creating a Fleet Provisioning Template 29
4.4 Creating a Sample Project 32
4.5 Setting Up the Project 38
4.5.1 Modifying the Configuration File 38
4.5.2 Settings for the Individual Connectivity Types 39
4.6 Running the Project 41
4.6.1 Building and Downloading the Project 41
4.6.2 Registering AWS IoT Information 42
4.6.3 Running the Fleet Provisioning Demo and Confirming the Results 44
4.6.4 Confirming the Results of Execution 51
4.6.5 OTA Update Execution through the Fleet Provisioning Project 56

5. Summary 57

6. Web Sites and Support Information 57

7. Supplementary Information 58

7.1 Points to Keep in Mind when Operating Multiple Devices within the Same LAN Environment 58

Revision History 59

1. Terminology

The following shows the terms used in this document.

Table 1-1 Terminology

Term	Meaning
AWS	Cloud computing services provided by Amazon Web Services, Inc.
Fleet provisioning	Function that automates provisioning for IoT devices at the time of their first activation.
FreeRTOS	An open-source real-time operating system for embedded systems.
Provisioning	Device provisioning, which certifies devices for communications with AWS IoT Core.

2. Device Provisioning

IoT device provisioning refers to the process of generating a unique ID (such as an X.509 certificate or a private key) for a device, registering the unique ID with an AWS IoT endpoint, and associating the necessary access privileges (IoT policies, etc.) with the ID to enable secure connection of the device with AWS IoT and other cloud-based applications (see Figure 2-2).

Device provisioning with AWS IoT uses the AWS IoT Core functions such as just-in-time-registration (JITR) and just-in-time-provisioning (JITP) to automate the process of registering the identity of each device in the AWS cloud and associating it with the necessary permissions, and to perform provisioning for multiple devices. However, the process of securely generating a unique ID and writing it to each device must be performed at the user's responsibility and this process incurs time-consuming manual operations for OEM vendors manufacturing large numbers of devices.

Fleet provisioning, which is described in this document, is one way to work around this issue.

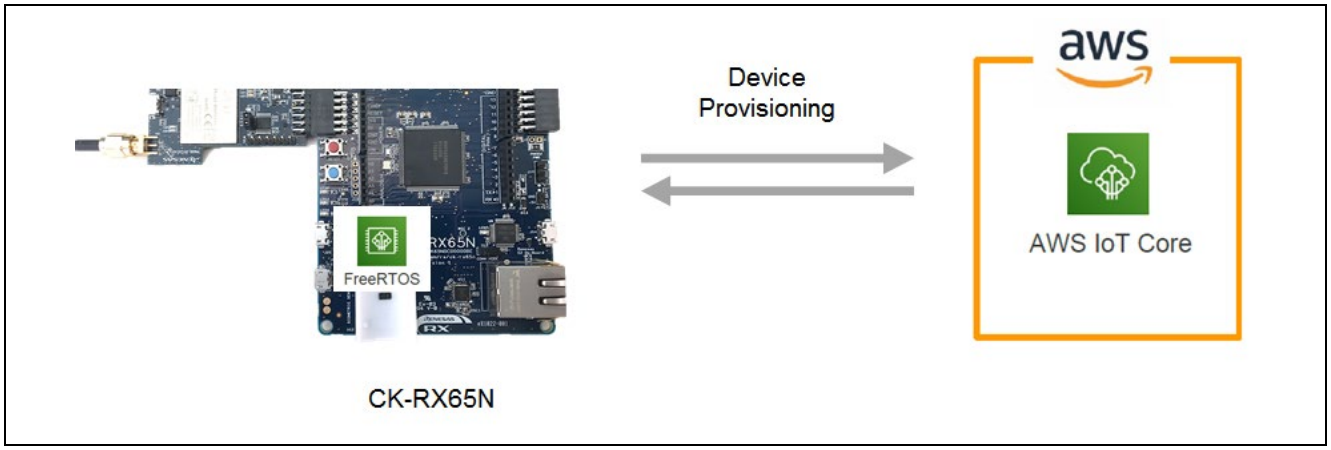


Figure 2-1 Device Provisioning

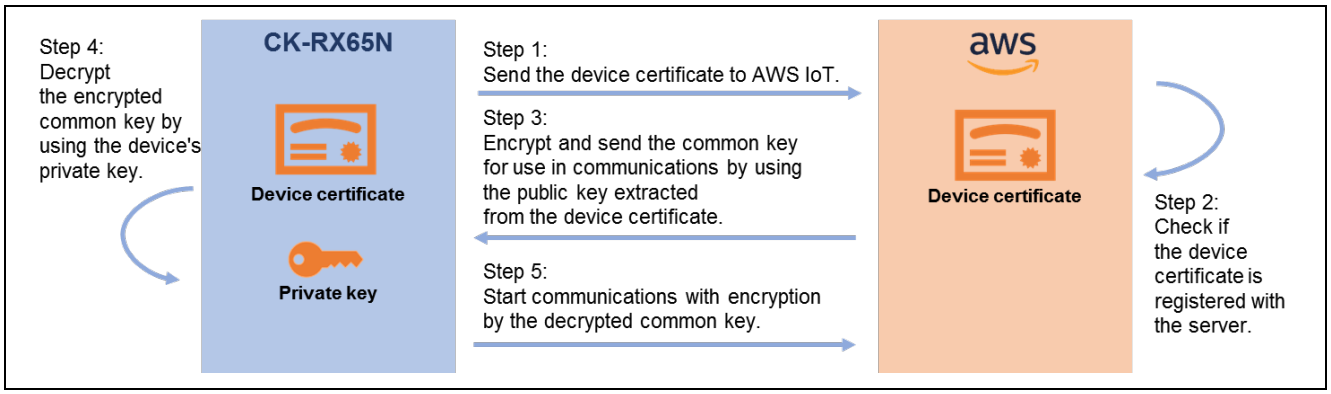


Figure 2-2 Outline of IoT Device Provisioning

2.1 Methods of Provisioning with AWS IoT

Any of the provisioning methods listed below can be selected for use with AWS IoT.

To allow users to select the device provisioning method that best matches their fields of application, AWS provides multiple provisioning methods to suit the market demand and a variety of expected use cases.

[Provisioning Methods in AWS IoT]

1. AWS IoT issuing and pre-registering a private key and a certificate (registration at the time of device kitting)
2. AWS IoT issuing and pre-registering a certificate (registration at the time of device kitting)
3. Registering through fleet provisioning (basis of descriptions in this document)
4. Having your own certification authority issue a certificate and pre-register it with AWS IoT
5. Having your own certification authority issue a certificate and register it with JITR
6. Having your own certification authority issue a certificate and register it with JITP
7. Having a non-registered certification authority issue a certificate and register it (multi-account registration)

To confirm the operation of FreeRTOS before starting to consider its use in a product for mass production, the simplest approach is "1. AWS IoT issuing and pre-registering a private key and a certificate", in which AWS issues a private key and a certificate which are then converted to a form that can be embedded in source code. The resulting code is then embedded in the set of source code to run under FreeRTOS. However, using this method to embed individual device-specific certificates during manufacturing is difficult. For this reason, this document focuses on fleet provisioning, which does not require the use of a certification authority and imposes the lightest workload during mass production.

Note: Some MCUs of the RX family incorporate a Trusted Secure IP (TSIP) module.
When the TSIP is used, an on-chip random number generator is used to generate an RSA or elliptic curve cryptosystem key pair and the public key is extracted and sent to a user-specified certification authority, which attaches a certificate and returns the result. This enables the implementation of JITR or JITP.
This method provides stronger security while reducing the cost of implementation.

2.2 Fleet Provisioning Methods

Fleet provisioning refers to a set of procedures in which provisioning takes place when each IoT device is activated for the first time. Its implementation can be broadly classified as being conducted through either of the following two methods.

1. Provisioning by claim (approach with the use of provisioning claim certificates)
2. Provisioning on the basis of trust (such as of mobile or web application users)

In addition, either of the following two methods can be used to obtain individual device-specific certificates and private keys for use in fleet provisioning.

A) Having the AWS certification authority generate a new device-specific certificate and private key and send them to the device (CreateKeysAndCertificate).
B) Generating a key pair internally on the device and sending a certificate signing request (CSR) to AWS to have them generate only a device-specific certificate and send it to the device (CreateCertificateFromCsr).

This document describes the implementation of a fleet provisioning demo that combines provisioning method 1 and certificate and key method B (see Figure 2-4).

This provisioning method has the following advantages.

Advantages:

- The device's private key is never sent outside the device.
- The manufacturing plant does not need to establish a connection with AWS IoT.
- The steps of issuing and registering individual device-specific certificates are not necessary.

On the other hand, it also has the following disadvantages. If you use this provisioning method, you need to be aware of both its advantages and disadvantages.

Disadvantages:

- The possibility of the provisioning claim certificate being leaked to an unauthorized party must be taken into account.
- Functionality for issuing a provisioning request and receiving a response must be implemented on the device.

2.3 Provisioning by Claim (Approach Using a Provisioning Claim Certificate)

A device can be manufactured with an embedded provisioning claim certificate and private key. If these credentials have been registered with AWS IoT, AWS IoT can exchange them for a unique device certificate that can then be used in the regular operation of the device. This process consists of the steps listed below.

The design of provisioning by claim assumes a scenario in which all devices are manufactured by using a common provisioning claim certificate. Such a provisioning claim certificate only allows each device to do the following.

1. Establish an initial connection to AWS IoT Core.
2. Authenticate its identity.
3. Request an ID to which the necessary permissions have been assigned for use in subsequent communications.

The provisioning claim certificate for use in common by all devices is written to each device along with the initial software at a site such as the manufacturing plant. If the device already contains a device-specific private key, it can send a provisioning claim certificate to be signed by AWS IoT Core along with a certificate signing request (CSR) (see Figure 2-4).

In addition to verification of the provisioning claim certificates presented by individual devices, Lambda-based provisioning hooks can be used in fleet provisioning to verify the attributes of devices. Examples of device attributes include the serial number, MAC ID, and device location. We recommend that you consider using Lambda functions in provisioning transactions as a way to automate acceptance or rejection of the provisioning status of the individual devices based on the custom attributes sent during this process.

Note that Lambda functions are not used in the demo project described in this application note.

Refer to the page at the following link for information on provisioning with the use of AWS Lambda.

https://docs.aws.amazon.com/iot/latest/developerguide/provision-wo-cert.html
"Using pre-provisioning hooks with the AWS CLI"

2.3.1 Overview of Provisioning by Claim (Using a Provisioning Claim Certificate)

Once power is supplied to the device and it is capable of connecting with a network, the following workflow is executed.

Also see Figure 2-3 and Figure 2-4 for the workflows of the CreateKeysAndCertificate method and CreateCertificateFromCsr method, respectively.

In addition, you can confirm the details of the AWS IoT Fleet Provisioning Demo workflow (CreateCertificateFromCsr method) on which the fleet provisioning demo described in this document is based, by visiting the page at the following link.

https://aws.github.io/aws-iot-device-sdk-embedded-C/latest/docs/doxygen/output/html/fleet_provisioning_demo.html

1. The device uses the claim certificate written to the device in advance to connect with AWS IoT Core via a secure TLS 1.2 connection. If the device contains a CSR, it is presented along with the provisioning claim certificate.
2. The certificate is associated with an extremely restrictive policy that only provides access to the IoT topics associated with the fleet provisioning process.
3. The fleet provisioning service returns a token providing "proof of ownership" to securely isolate the transaction and a valid certificate and private key payload. The token will be presented through an MQTT request in step 4 and will be used to start template processing for activating the certificate. If a CSR was presented, a certificate is generated from the CSR.
4. The device sends an MQTT request to AWS IoT Core and presents the ownership token, name of the fleet provisioning template created by the account owner and, optionally, device attributes for use in provisioning verification. The use of Lambda-based provisioning hooks to enable additional verification such as checking the device’s serial number or MAC ID against a pre-approved list is recommended.
5. The provisioning transaction is done by using the fleet provisioning template and the results are returned. Typically, the device attributes are verified by Lambda functions, the certificate is activated, the production policy is attached, and a thing or a group is created (optional).
6. Based on the results of the provisioning transaction, the status of the new certificate is returned. If the transaction was successful, the provisioning claim certificate becomes unnecessary and the device uses the activated device certificate to reconnect with AWS IoT Core. If the transaction was denied, an "access denied" error is returned to the device.

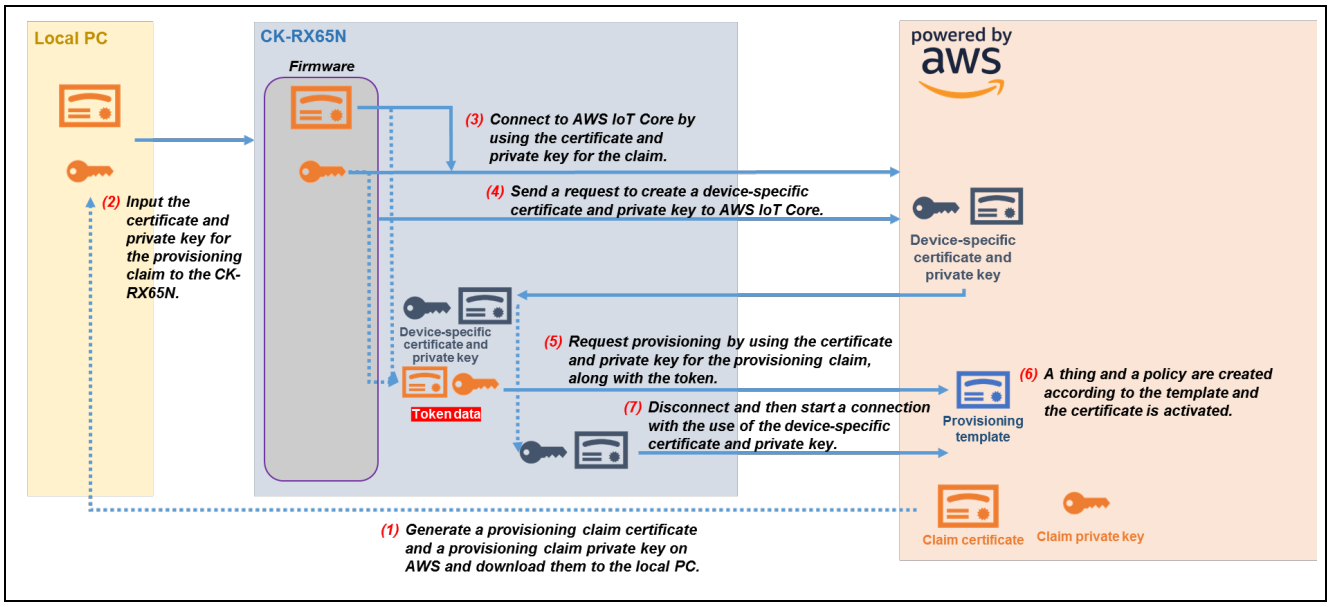


Figure 2-3 Workflow of Provisioning by Claim Using CreateKeysAndCertificate Method

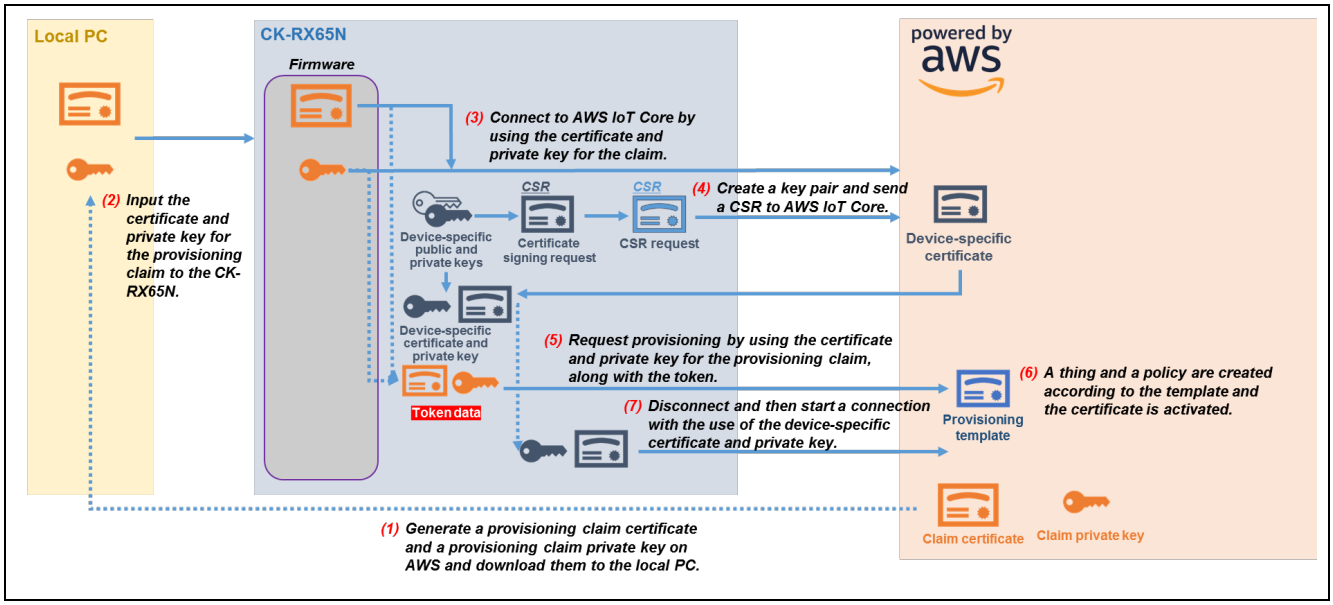


Figure 2-4 Workflow of Provisioning by Claim Using CreateCertificateFromCsr Method

2.3.2 Determining a Unique Name for a Thing

The payload of an MQTT request sent during fleet provisioning can include the serial number of the device to ensure that each device has a unique thing name and not a duplicate of an existing one.

Either of the following two broadly classified methods can be used to determine the serial number.

1. A random value generated by a random number generator or the unique ID value for the device is used as the serial number.
2. A Lambda-based provisioning hook and Amazon S3 or a user-specified database are used to convert a provisionally determined serial number to a unique serial number.

Method 1 is used in the example described in this document; that is, the unique ID assigned to each MCU of the RX family is used to prevent duplication of thing names.

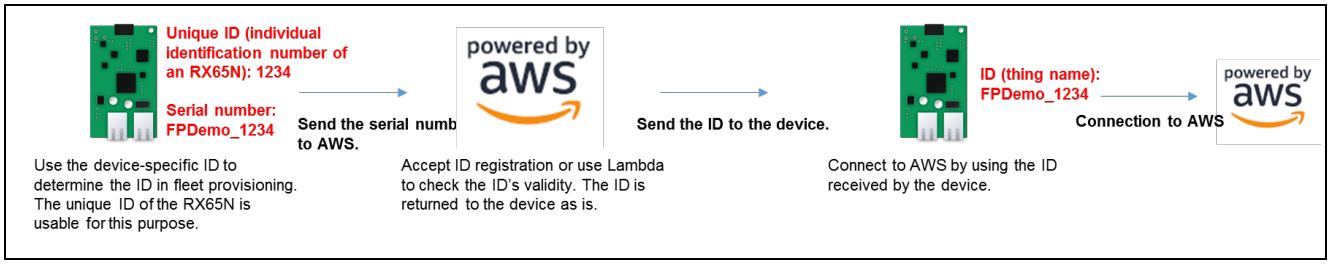


Figure 2-5 Using a Random Value or a Unique ID to Determine a Thing Name

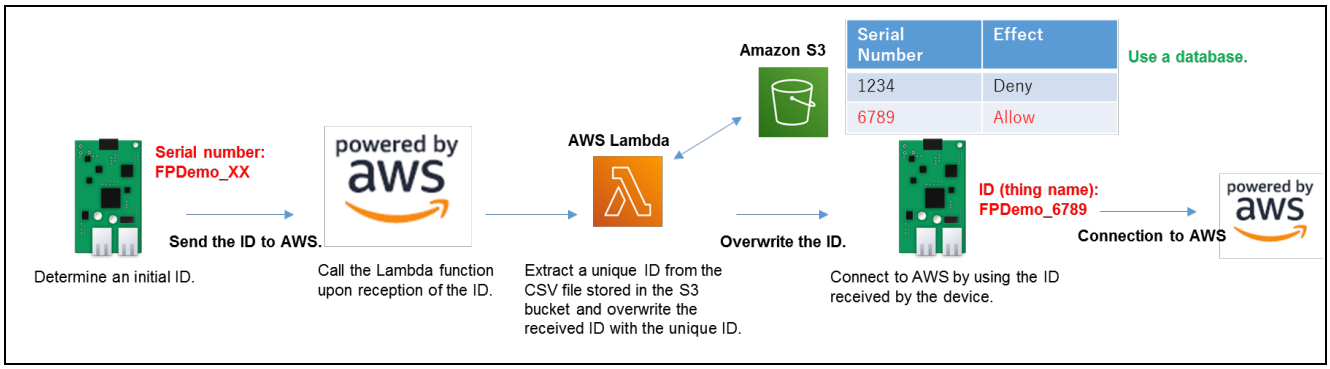


Figure 2-6 Using Amazon S3 or a Database to Determine a Thing Name

3. Preparation

This section and those that follow describe the steps from importing the project accompanying this application note to running the fleet provisioning demo on the CK-RX65N v2 board.

3.1 Hardware Environment

The following table lists the elements of the hardware environment for the demo project.

Table 3-1 Hardware Environment

Item	Content	Provider	Description
Board	CK-RX65N v2	Renesas Electronics Corporation	Cloud kit incorporating the RX65N MCU
PC	Windows11	—	Host PC for demo

3.2 Software Environment

The following table lists the elements of the software environment for the demo project.

Table 3-2 Software Environment

Item	Content	Version	Description
Integrated development environment	e^2 studio	v2025-12	—
Toolchain	CC-RX	v3.07.00	—
	GCC for Renesas RX	v14.2.0.202505	—
FreeRTOS	v202406.04-LTS-rx	V1.2.0	—
Communication software	Tera Term	Ver. 5.5.0	For displaying logs
AWS command line interface (CLI)	AWS Command Line Interface	Version 2	For AWS settings

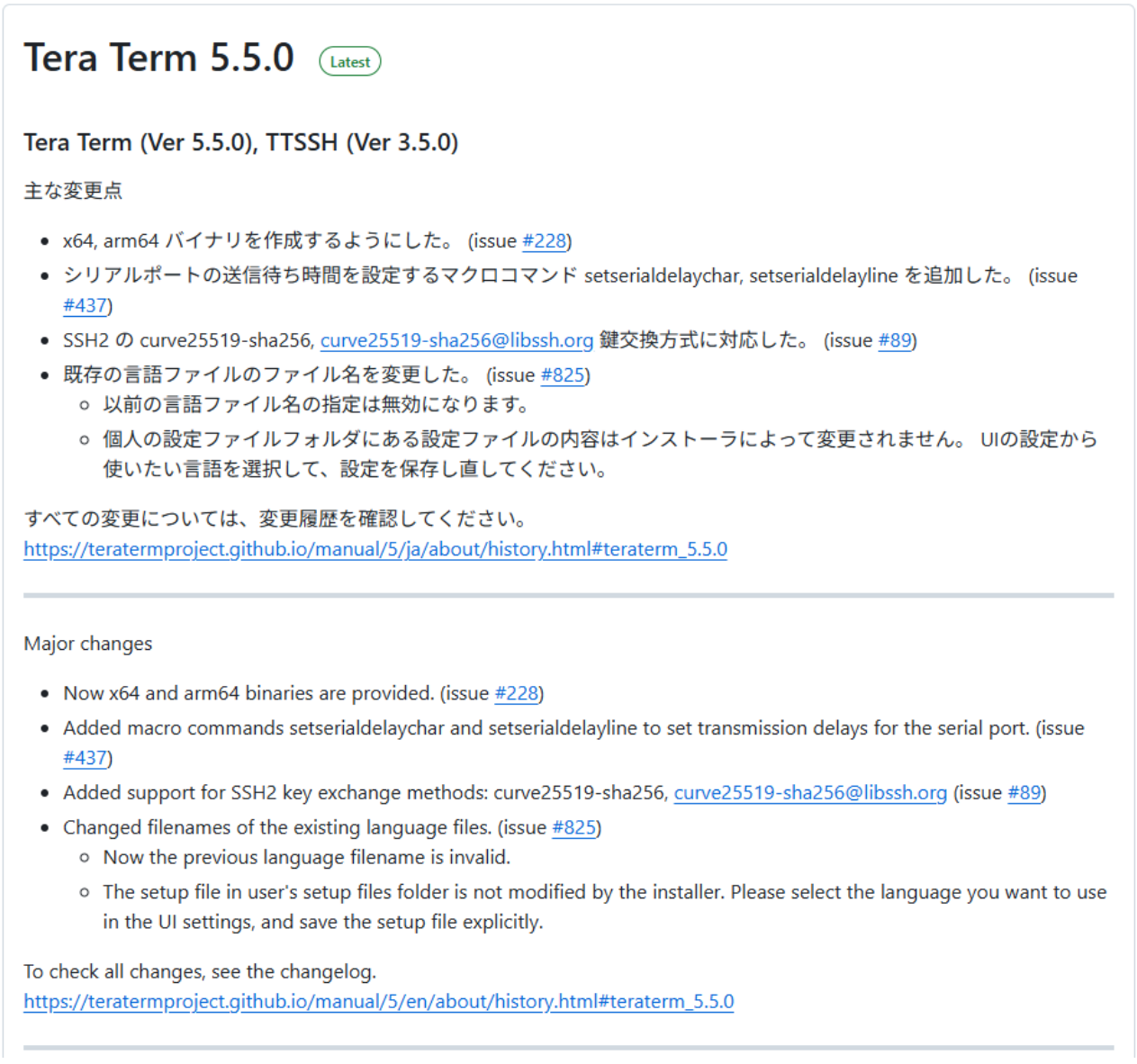
3.3 Installing and Setting Tera Term

The demo uses Tera Term to display log output.

(1) Accessing the Tera Term download site

Access the Tera Term download site at the following link.

Tera Term download site (GitHub)



Tera Term 5.5.0 Latest

Tera Term (Ver 5.5.0), TTSSH (Ver 3.5.0)

主な変更点

- x64, arm64 バイナリを作成するようにした。 (issue #228)
- シリアルポートの送信待ち時間を設定するマクロコマンド setserialdelaychar, setserialdelayline を追加した。 (issue #437)
- SSH2 の curve25519-sha256, curve25519-sha256@libssh.org 鍵交換方式に対応した。 (issue #89)
- 既存の言語ファイルのファイル名を変更した。 (issue #825)
  - 以前の言語ファイル名の指定は無効になります。
  - 個人の設定ファイルフォルダにある設定ファイルの内容はインストーラによって変更されません。 UIの設定から使いたい言語を選択して、設定を保存し直してください。

すべての変更については、変更履歴を確認してください。
https://teratermproject.github.io/manual/5/ja/about/history.html#teraterm_5.5.0

Major changes

- Now x64 and arm64 binaries are provided. (issue #228)
- Added macro commands setserialdelaychar and setserialdelayline to set transmission delays for the serial port. (issue #437)
- Added support for SSH2 key exchange methods: curve25519-sha256, curve25519-sha256@libssh.org (issue #89)
- Changed filenames of the existing language files. (issue #825)
  - Now the previous language filename is invalid.
  - The setup file in user's setup files folder is not modified by the installer. Please select the language you want to use in the UI settings, and save the setup file explicitly.

To check all changes, see the changelog.
https://teratermproject.github.io/manual/5/en/about/history.html#teraterm_5.5.0

Figure 3-1 Tera Term to be Downloaded

(2) Downloading the Tera Term installer

Specify the exe file of the latest version of Tera Term that is compatible with your OS and download it.

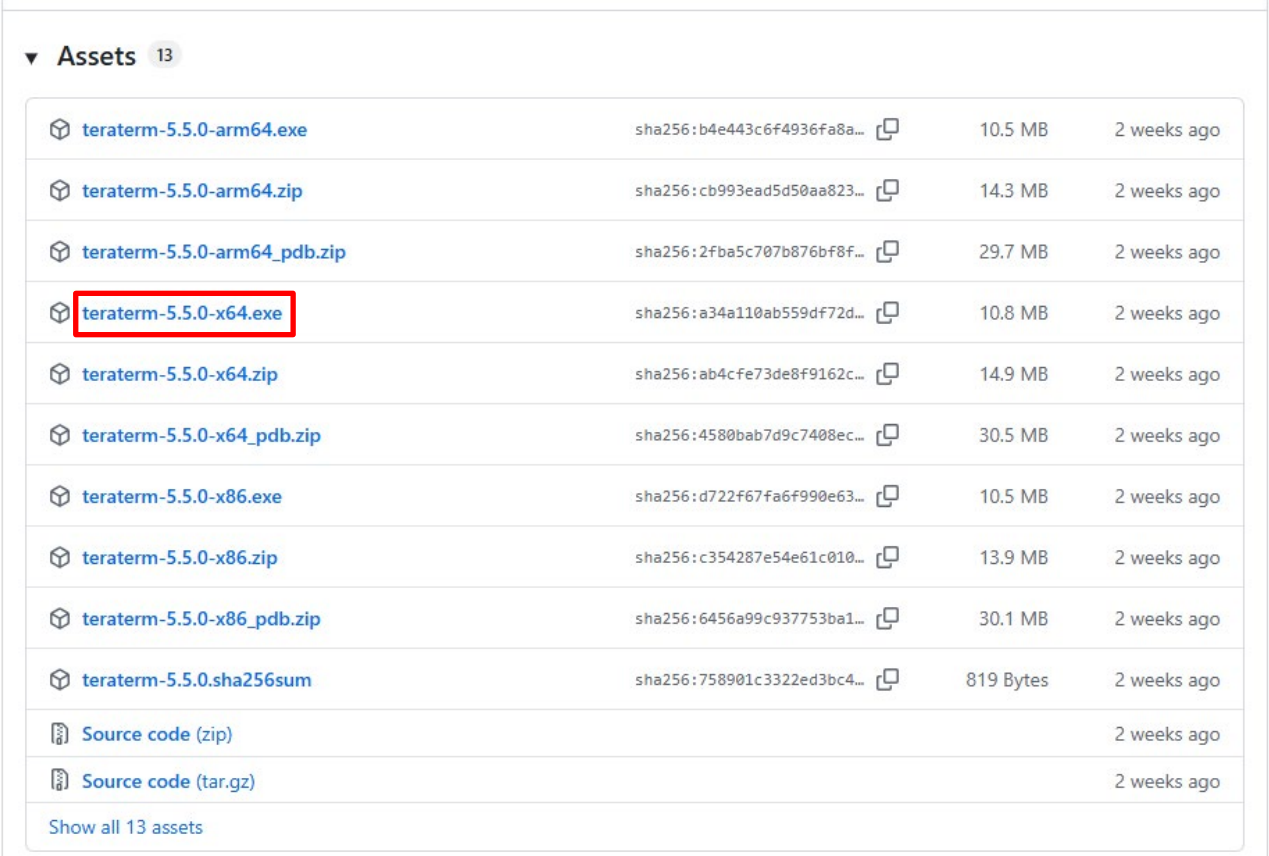


Figure 3-2 Selecting the File to be Downloaded

(3) Running the installer

Run the installer and follow the instructions that appear to install Tera Term.
Execute the installer with administrative permissions.

(4) Confirming that Tera Term starts up

Click on the Tera Term icon in the Start menu and confirm that Tera Term starts.

3.4 Installing the AWS Command Line Interface (CLI)

The AWS command line interface (CLI) is a tool for managing and operating the various Amazon Web Services (AWS) with the use of commands from a command line shell.
This application note provides guidance on making AWS settings with operations through the AWS CLI.

Note: AWS CLI version 1 and AWS CLI version 2 are not compatible with each other. Be sure to install version 2.
If version 1 has already been installed, update it to version 2 with reference to the following Web page:
Migration from AWS CLI version 1 to AWS CLI version 2

(1) Executing the command line shell

Execute the Windows PowerShell or Windows command prompt. The PowerShell is used for the display examples in this application note.
The command line shell is referred to as the command prompt in this application note.

Note: When using PowerShell, it is recommended to use Version 7 or later.

(2) Installing the AWS CLI

Enter the following command at the command prompt. This leads to the automatic downloading and execution of the AWS CLI v2 installer.

```
> msiexec.exe /i https://awscli.amazonaws.com/AWSCLIV2.msi
```

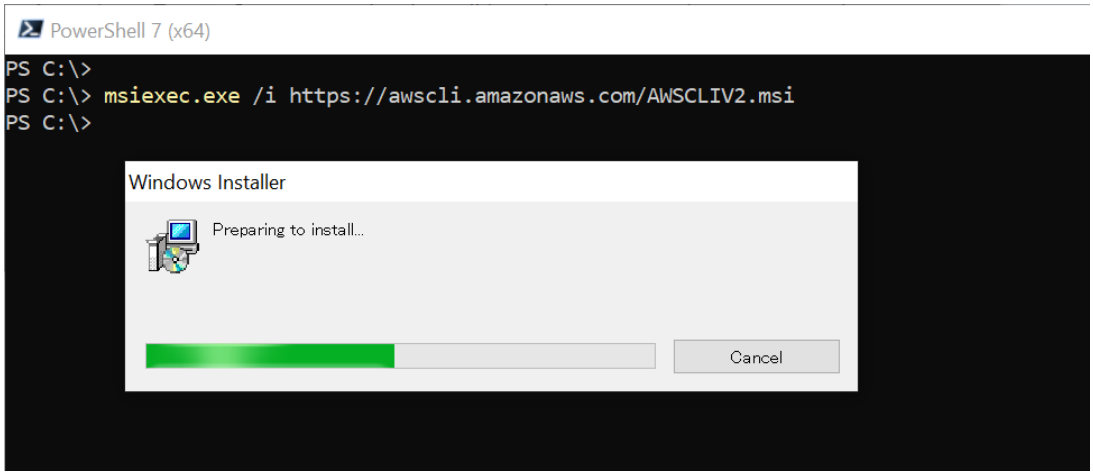


Figure 3-3 Downloading AWS CLI and Running the Installer

The steps for downloading and installation can be referred to on the following Web page.

AWS CLI install and update instructions

(3) Running the installer

Run the installer and wait for a while until the [Next] button becomes selectable.
After clicking on the [Next] button, follow the instructions that appear to install the AWS CLI.
After installation is completed, restart the command prompt window.

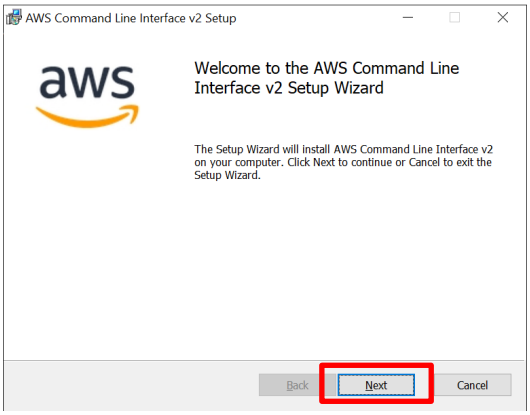


Figure 3-4 Completing Installation of the AWS CLI

(4) Confirming the installation

After installation is completed, enter the following command at the command prompt. The version of the installed AWS CLI is displayed.
After you have been able to confirm that the installed version is 2.xx.x, the installation procedure is complete.

```
> aws --version
```

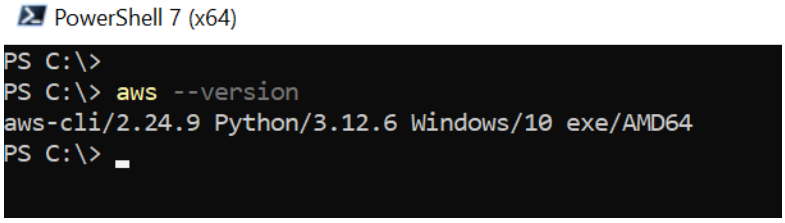


Figure 3-5 Confirming the Version of the AWS CLI

Note: The version of Python displayed here is the version of the Python library included in the executable file of the AWS CLI. It may be different from the version of Python installed in your system but this will not cause a problem.

3.5 FreeRTOS Project

Figure 3-6 shows the software components of the demo project.

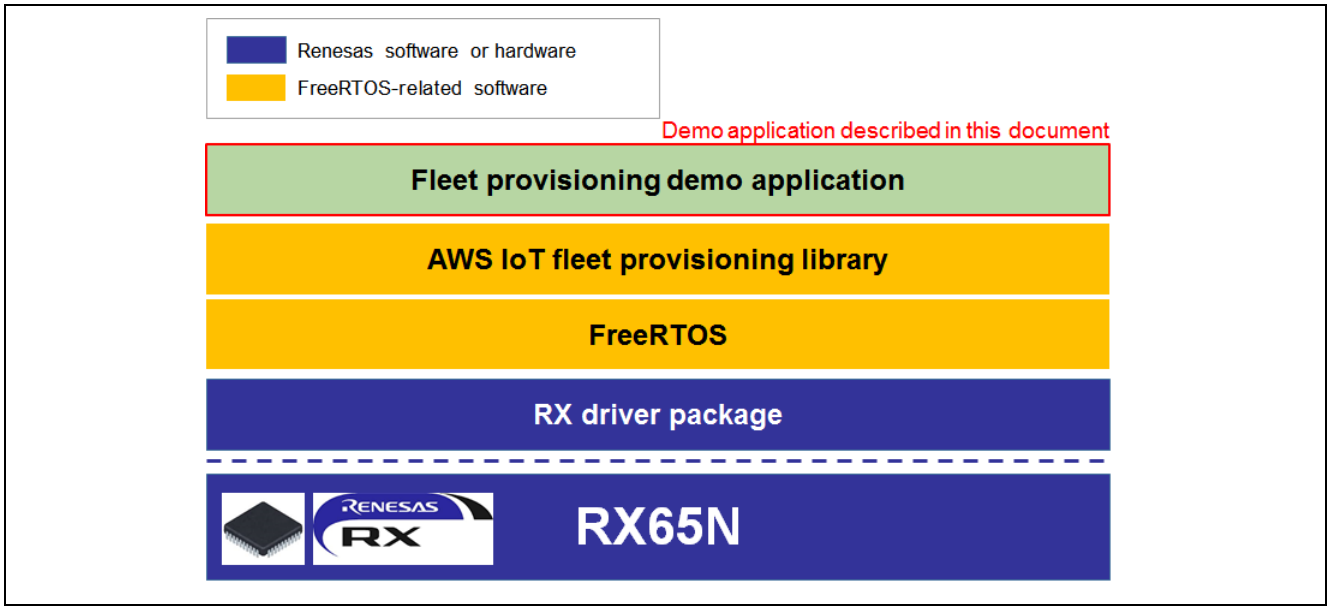


Figure 3-6 Components of the Demo Project Included with This Application Note

The AWS IoT fleet provisioning library for FreeRTOS is used to implement fleet provisioning functionality. The RX Driver Package, FreeRTOS, AWS IoT Fleet Provisioning Library, and the demo application are located in the GitHub repositories listed below.
For information on how to obtain the demo application set, see section 4.4, Creating a Sample Project. The downloaded demo application set contains the RX driver package, FreeRTOS, and AWS IoT fleet provisioning library.

Demo application: https://github.com/renesas/iot-reference-rx

4. Running the Fleet Provisioning Demo

This section describes how to run the fleet provisioning demo application.

4.1 Preparing the Execution Environment

Prepare the environment in which the demo is to run.

Figure 4-1 shows an example with the use of the CK-RX65N v2 board.

Either a wired (Ethernet) or wireless (cellular or Wi-Fi) communications interface can be used to connect to AWS.

The example described in this application note is with the use of an Ethernet connection.

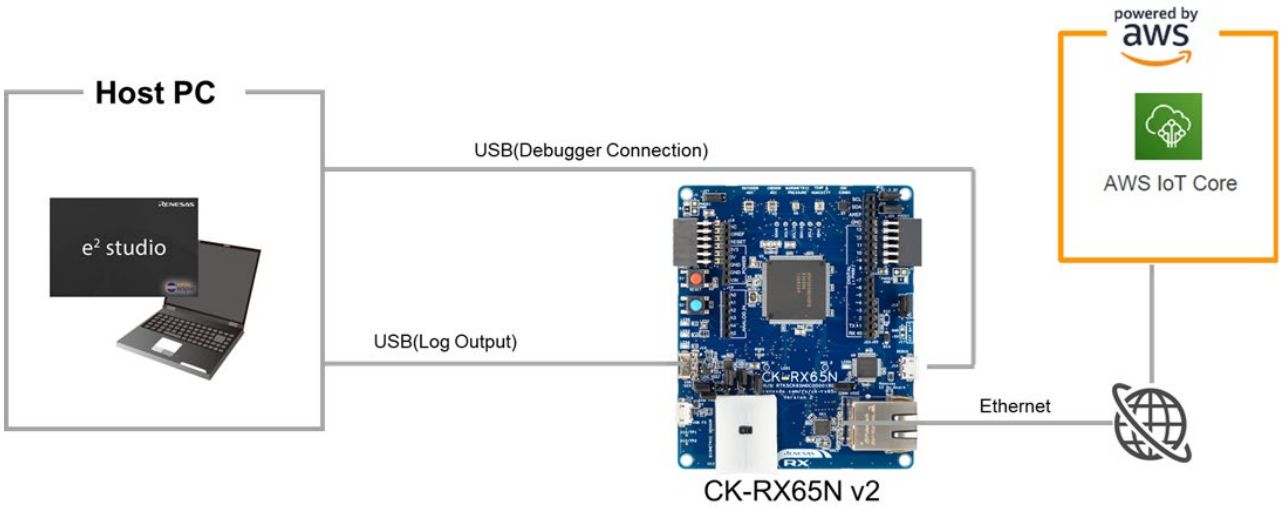


Figure 4-1 Environment for Running the Demo

Note: The board shown in Figure 4-1 is an example in which Ethernet is used.
For the methods of connection for the individual connectivity types, refer to section 2.6, Connecting the CK-RX65N v2, in the application note *RX Family: How to Implement FreeRTOS OTA Using Amazon Web Services (202406-LTS Version)* (R01AN7662).

4.2 Preparing for AWS

This section describes the procedure for setting up AWS to run the fleet provisioning demo application.

A procedure for using the AWS command line interface (CLI) with the single sign-on (SSO) technique as an IAM Identity Center user (workforce user) is explained.

For the procedure from the creation of an AWS account to the setup and initial settings for the AWS CLI, refer to section 3.1, Setting Up an Environment for Signing In to AWS, in the application note *RX Family: How to Implement FreeRTOS OTA Using Amazon Web Services (202406-LTS Version)* (R01AN7662).

As described in the above application note, pay attention to the following points when using the AWS CLI.

(a) Creation of a work folder

Create a desired folder for performing steps with the AWS CLI with reference to section 3.1.5, Creating a Work Folder, in the R01AN7662 application note.

The configuration files and other files to be used in CLI commands are to be stored in this folder.

The work folder is created as "C:\aws" in the examples used for explanations in this application note.

(b) Profile name for use in entering AWS CLI commands

Create a CLI profile with reference to section 3.1.4 (2), Create an SSO profile, in the R01AN7662 application note.

When entering a CLI command, add "--profile *<PROFILE_NAME>*" to the end of the command to specify a profile.

The profile name created in this step should be specified as *<PROFILE_NAME>*.

The profile name is specified as "Renesas" in the examples used for explanations in this application note.

4.3 AWS Settings for Fleet Provisioning

It is necessary to configure AWS settings to run the fleet provisioning demo.

1. Setting policies
2. Generating a claim certificate and a claim key pair
3. Creating a fleet provisioning template

The AWS CLI is used for the AWS settings.
Do an SSO login through the AWS CLI beforehand with reference to section 3.2, SSO Login to AWS through the CLI, in the application note *RX Family: How to Implement FreeRTOS OTA Using Amazon Web Services (202406-LTS Version)* (R01AN7662).

4.3.1 Creating a Policy for the Fleet Provisioning Demo

Create a policy to be used in executing the fleet provisioning demo.

(1) Creating a json file for setting a policy

Create a configuration file of policy. With a text editor, create a file with the name "policy_fp.json" in the work folder that was created in section 4.2, Preparing AWS, and enter the following code.

- policy_fp.json

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:Connect",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Receive",
        "iot:RetainPublish"
      ],
      "Resource": [
        "arn:aws:iot:<region name>:<account id>:topic/$aws/certificates/create-from-
csr/*",
        "arn:aws:iot:<region name>:<account id>:topic/$aws/provisioning-templates/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:Subscribe",
      "Resource": [
        "arn:aws:iot:<region name>:<account id>:topicfilter/$aws/certificates/create-
from-csr/*",
        "arn:aws:iot:<region name>:<account id>:topicfilter/$aws/provisioning-
templates/*"
      ]
    }
  ]
}
```

Figure 4-2 Policy Document for Fleet Provisioning

Enter the following name and account according to your environment in the above code.

— <region name>: Enter the region of usage.
— <account id>: Enter your account ID.

Notes: 1. Enter the character string of the region selected through the AWS management console (Web-based). For example, the region name for "Asia Pacific (Tokyo)" is "ap-northeast-1".
2. The account ID can be obtained by clicking on the account name in the upper-right region of the AWS management console (Web-based) and deleting the hyphens from the displayed 12-digit number.

(2) Creating a policy

Enter the following command and create a policy.

```
> aws iot create-policy --policy-name <DEMO_POLICY_NAME> --policy-document
file://policy_fp.json --profile <PROFILE_NAME>
```

- Any desired policy name can be entered as <DEMO_POLICY_NAME> (example: fp_demo_policy).

After the command has been executed, the display in the window will be as shown below.

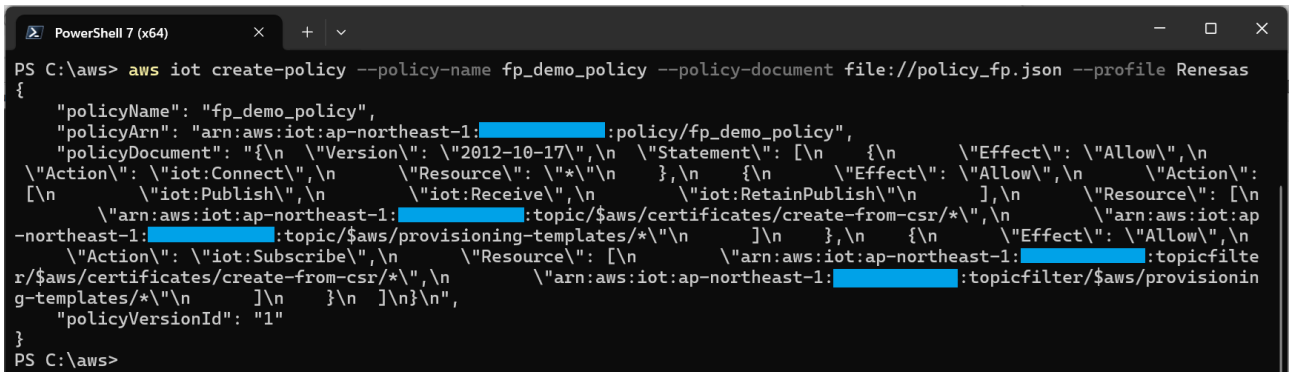


Figure 4-3 Creating the Policy for Fleet Provisioning Demo

Make a note of the policy name that was created. You will need it in a later process.

4.3.2 Creating Policies for Things Created through Fleet Provisioning

Create policies that will be attached to the things created through the execution of fleet provisioning (hereafter called fleet-provisioned things).
Set the following policies for the device to be connected in this application note.

- iot:Connect: Connects to AWS IoT.
- iot:Publish: Publishes (transmits) a topic.
- iot:Subscribe: Subscribes to (receives) a topic.
- iot:Receive: Receives messages from AWS IoT.

Note: The policies created here are the same as those created in section 3.3.1, Setting Policies, of the application note *RX Family: How to Implement FreeRTOS OTA Using Amazon Web Services (202406-LTS Version)* (R01AN7662).
If the policies have been created during OTA update execution, the steps described below can be omitted. In such a case, make a note of the names of the created policies.

(1) Creating a json file for setting the policies for things

Create a configuration file of policies.
With a text editor, create a file with the name "policy.json" in the work folder that was created in section 4.2, Preparing AWS, and enter the following code.

- policy.json

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:Connect",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Publish",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Subscribe",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Receive",
            "Resource": "*"
        }
    ]
}
```

Figure 4-4 Policy Document for Fleet-Provisioned Things

The text strings in blue are the names of access permissions to be assigned.

(2) Creating policies for things

Enter the following command and create policies.

```
> aws iot create-policy --policy-name <THING_POLICY_NAME> --policy-document
file://policy.json --profile <PROFILE_NAME>
```

- Any desired policy name can be entered as *<THING_POLICY_NAME>* (example: fp_thing_policy).

After the command has been executed, the display in the window will be as shown below.

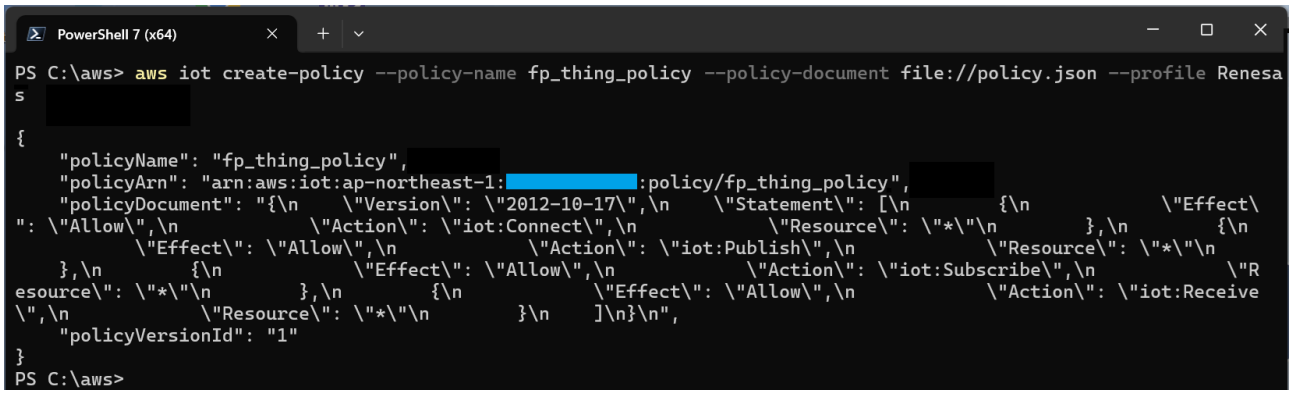


Figure 4-5 Creating the Policies for Fleet-Provisioned Things

Make a note of the policy name for things that was created. You will need it in a later process.

4.3.3 Generating a Claim Certificate and a Claim Key Pair

Generate a certificate and a key pair for the provisioning claim to be used in fleet provisioning.

(1) Creating and downloading a certificate and a key pair for the provisioning claim

Create a provisioning claim certificate and make it active, then download the files of the certificate and key pair (public and private keys) for the provisioning claim.
Enter the following command.

```
> aws iot create-keys-and-certificate --set-as-active --certificate-pem-
outfile "fp-certificate.pem.crt" --public-key-outfile "fp-public.pem.key" --
private-key-outfile "fp-private.pem.key" --profile <PROFILE_NAME>
```

Execution of the command leads to generation of a certificate and a key pair for the provisioning claim and the display in the window will be as shown below (the lengths of the character strings in the example of the display are abbreviated).

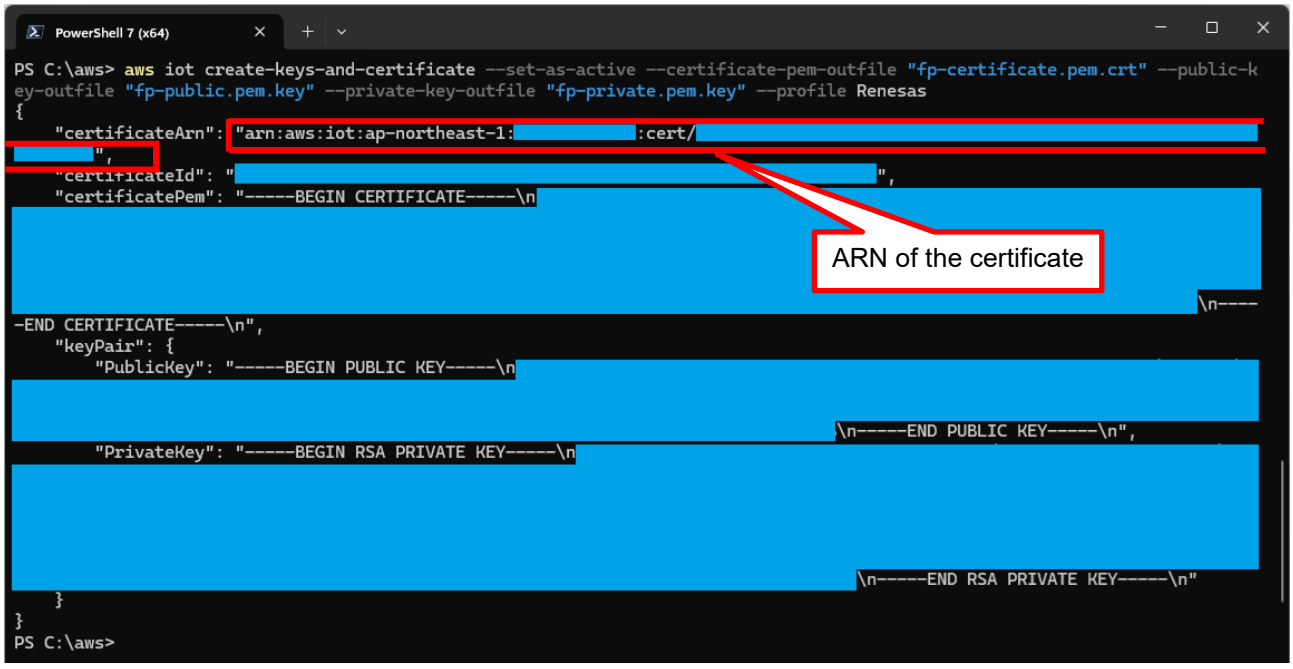


Figure 4-6 Generating and Downloading a Certificate and a Key Pair for the Provisioning Claim

Make a note of the ARN of the provisioning claim certificate ("certificateArn": text inside the red frame in the above figure) which is displayed. You will need it in a later process.

After execution of the command for creating a certificate, the following three files are downloaded to the work folder that was created in section 4.2, Preparing AWS.

Table 4-1 Downloaded Certificate and Key Files for the Provisioning Claim

File Name	Description
fp-certificate.pem.crt	Certificate for provisioning claim
fp-public.pem.key	Public key for provisioning claim
fp-private.pem.key	Private key for provisioning claim

The certificate and private key in fleet provisioning are equivalent to passwords. Registering a certificate and a private key in the device allows their use in establishing connection between the device and AWS.

Note: The certificate, public key, and private key for the provisioning claim can only be downloaded at the time the certificate is created.
These files should be stored in a safe location to prevent leakage of the information.

(2) Attaching a policy to the claim certificate

Attach the policy for the fleet provisioning demo to the created provisioning claim certificate.
Enter the following command.

```
> aws iot attach-policy --policy-name <DEMO_POLICY_NAME> --target
<CERTIFICATE_ARN> --profile <PROFILE_NAME>
```

- Enter the name of the policy for the fleet provisioning demo that was created in section 4.3.1(2), Creating a policy, as *<DEMO_POLICY_NAME>*.
- Enter the ARN of the provisioning claim certificate that was created in step (1), Creating and downloading a certificate and a key pair for the provisioning claim, as *<CERTIFICATE_ARN>*.

After the command has been executed, the display in the window will be as shown below (if the operation is successful, nothing will be displayed as the result of execution).

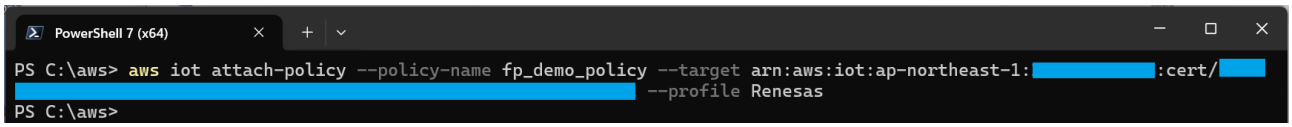


Figure 4-7 Attaching a Policy to the Provisioning Claim Certificate

The settings related to generation of the certificate and key pair for the provisioning claim are complete at this point.

4.3.4 Creating a Role

Create a role to be used in the execution of fleet provisioning.

(1) Creating a json file for setting a role

Create a configuration file of role.
With a text editor, create a file with the name "trust-policy_fp.json" in the work folder that was created in section 4.2, Preparing AWS, and enter the following code.

- trust-policy_fp.json

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "iot.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Figure 4-8 Role Document

(2) Creating a role

Enter the following command and create a role.

```
> aws iam create-role --role-name <ROLE_NAME> --assume-role-policy-document
file://trust-policy_fp.json --path /service-role/ --profile <PROFILE_NAME>
```

- Any desired policy name can be entered as *<ROLE_NAME>* (example: role_fleet).

After the command has been executed, the display in the window will be as shown below.

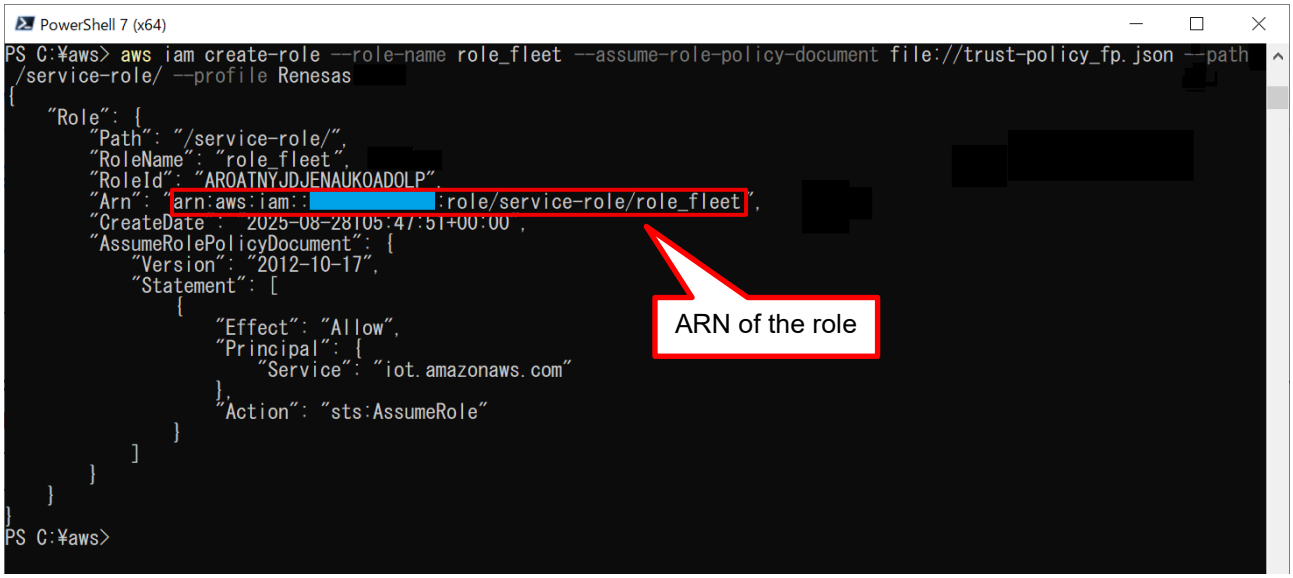


Figure 4-9 Creating a Role

Make a note of the ARN of the role ("Arn": text inside the red frame in the above figure) which is displayed. You will need it in a later process. Also, make a note of the role name which is displayed.

(3) Attaching the permission policy to the role

Enter the following command and attach a policy to the role.

```
> aws iam attach-role-policy --role-name <ROLE_NAME> --policy-arn
arn:aws:iam::aws:policy/service-role/AWSIoTThingsRegistration --profile
<PROFILE_NAME>
```

- Enter the role name that was created in section 4.3.4(2), Creating a role, as *<ROLE_NAME>*.

After the command has been executed, the display in the window will be as shown below (if the operation is successful, nothing will be displayed as the result of execution).

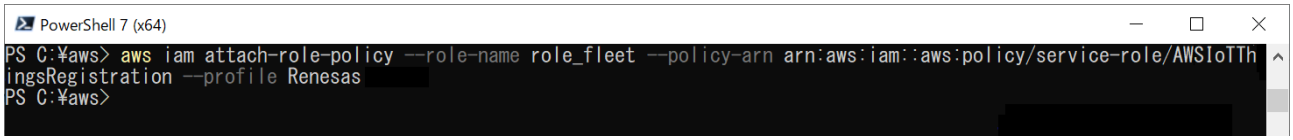


Figure 4-10 Attaching the Permission Policy to the Role

4.3.5 Creating a Fleet Provisioning Template

Create a provisioning template for storage of the settings to be used in automatically creating and assigning the necessary AWS resources when the IoT device is initially connected.

(1) Creating a json file for a template

Create a configuration file of template.
With a text editor, create a file with the name "template.json" in the work folder that was created in section 4.2, Preparing AWS, and enter the following code.

- template.json

```
{
  "Parameters": {
    "SerialNumber": {
      "Type": "String"
    },
    "AWS::IoT::Certificate::Id": {
      "Type": "String"
    }
  },
  "Resources": {
    "policy_<THING_POLICY_NAME>": {
      "Type": "AWS::IoT::Policy",
      "Properties": {
        "PolicyName": "<THING_POLICY_NAME>"
      }
    },
    "certificate": {
      "Type": "AWS::IoT::Certificate",
      "Properties": {
        "CertificateId": {
          "Ref": "AWS::IoT::Certificate::Id"
        },
        "Status": "Active"
      }
    },
    "thing": {
      "Type": "AWS::IoT::Thing",
      "OverrideSettings": {
        "AttributePayload": "MERGE",
        "ThingGroups": "DO_NOTHING",
        "ThingTypeName": "REPLACE"
      },
      "Properties": {
        "AttributePayload": {},
        "ThingGroups": [],
        "ThingName": {
          "Fn::Join": [
            "",
            [
              "<THING_NAME_PREFIX>",
              {
                "Ref": "SerialNumber"
              }
            ]
          ]
        }
      }
    }
  }
}
```

Figure 4-11 Provisioning Template Document

- Enter the policy name for fleet-provisioned things that was created in section 4.3.1(2), Creating a policy, as *<THING_POLICY_NAME>* (at two locations).
- Any desired character string can be entered as *<THING_NAME_PREFIX>*. The names of the things to be registered with AWS will be generated from this character string and the unique ID for the MCU (example of a prefix: fp_thing_).

(2) Creating a Provisioning Template

Enter the following command and create a provisioning template.

```
> aws iot create-provisioning-template --template-name <TEMPLATE_NAME> --
provisioning-role-arn <ROLE_ARN> --template-body file://template.json --
enabled --profile <PROFILE_NAME>
```

- Any desired template name can be entered as <TEMPLATE_NAME> (example: fp_template).
- Enter the ARN of the role that was created in step 4.3.4(2), Creating a role, as <ROLE_ARN>.

After the command has been executed, the display in the window will be as shown below.

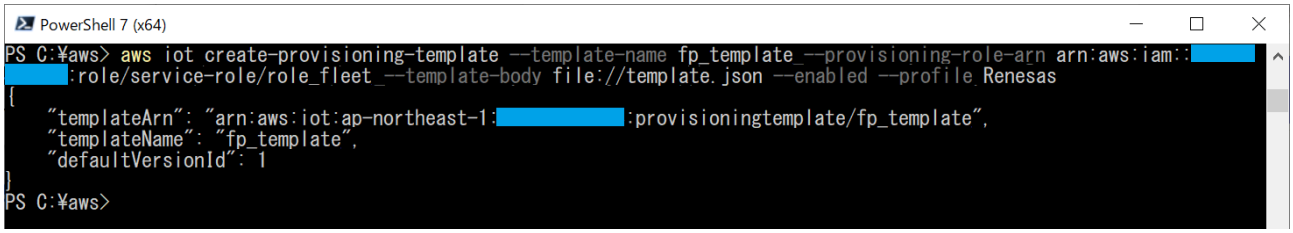


Figure 4-12 Creating a Provisioning Template

Make a note of the provisioning template name which is displayed. You will need it in a later process.

The AWS settings related to fleet provisioning are complete at this point.

4.4 Creating a Sample Project

This section describes the procedure for creating a sample project to perform provisioning for IoT devices with the use of Amazon Web Services.

(1) Creating an e² studio workspace

Launch the e² studio and create a new workspace.

Use short names for the workspace and project files. If the full path to the files at the lowest level of the directory structure exceeds 259 bytes, an error will occur.
Make sure that the names you enter only have alphanumeric characters. An error will occur if other characters are used.

Example: Creating a workspace in the directory "C:\workspace"

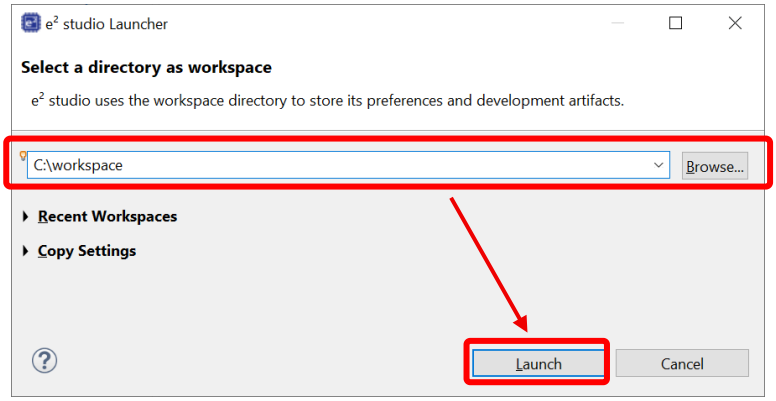


Figure 4-13 Dialog Box for Creating a Workspace

(2) Creating a folder for importing a project

Create a folder where a project is to be imported.

Create a folder with a desired name under the created workspace folder.
The folder is created as "iot-reference-rx" in the example in this application note.

(3) Importing files

Select [File] → [Import] to open the [Import] dialog box.

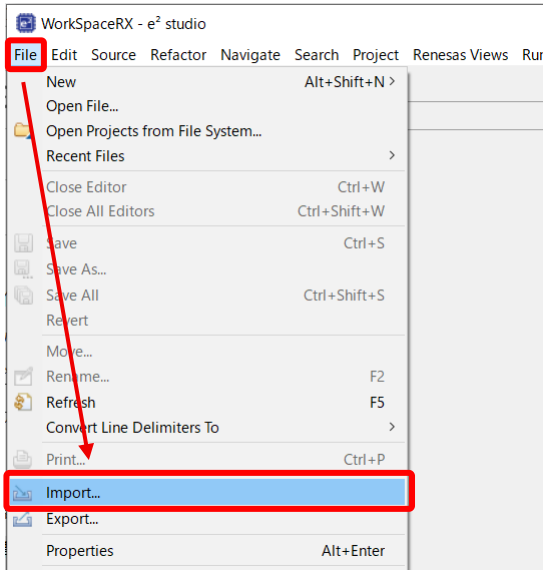


Figure 4-14 Importing a Project

(4) Selecting the method of importing

Click on [General] in the [Import] tree to expand that branch, select [Renesas GitHub FreeRTOS (with IoT libraries) Project], and press the [Next] button.

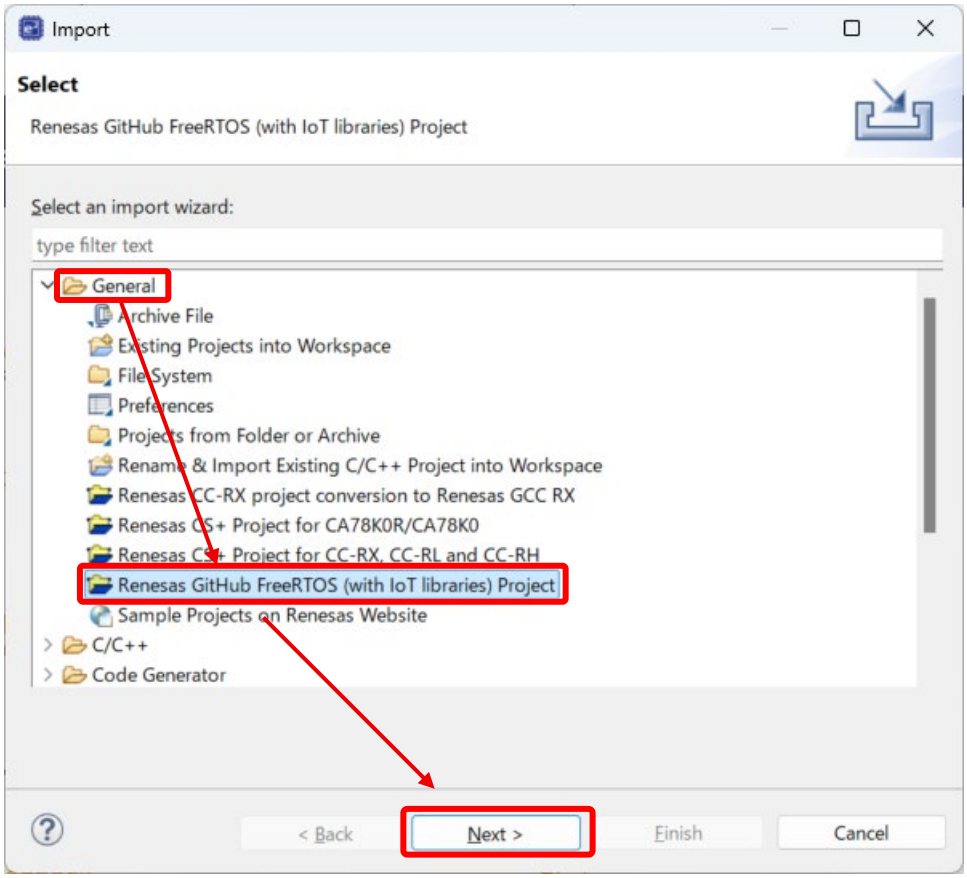


Figure 4-15 Selecting the Method of Importing

(5) Selecting the version of FreeRTOS to be imported

The dialog box for selecting one from among the RTOS versions that are available for importing appears.

Click on the [Browse] button and specify the folder for importing a project that was created in section 4.4(2), The project will be downloaded from GitHub to the specified folder.

Select "v202406.04-LTS-rx-1.2.0" under [RTOS version setting] and click on the [Next] button. The project will be copied to the specified folder.

Note: Among the versions of the RTOS registered with the e^2 studio, those that have been downloaded to the download folder are listed here.

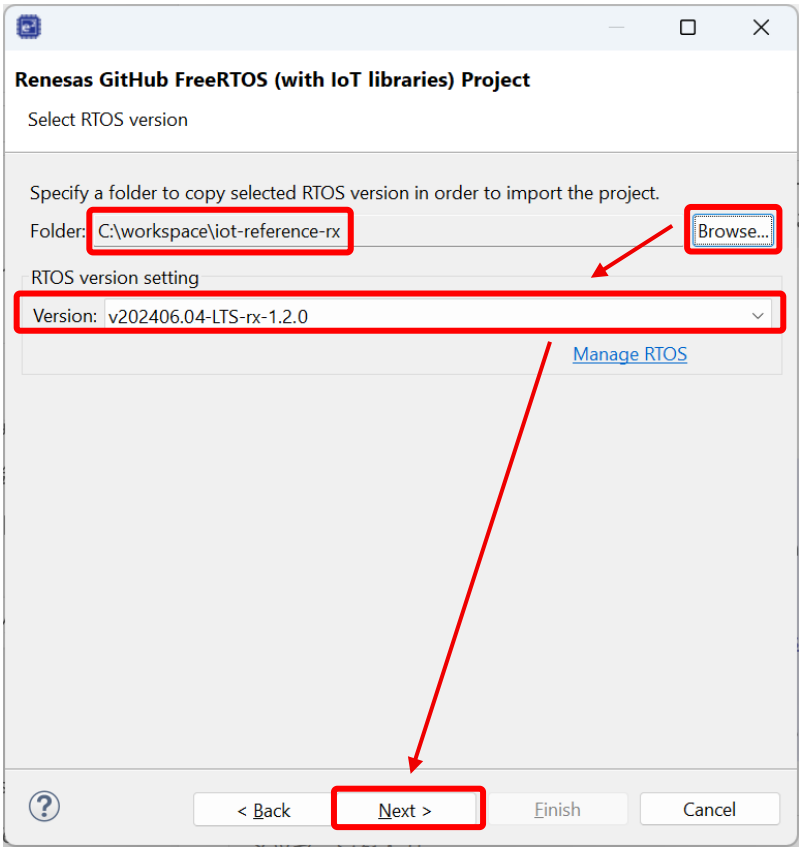


Figure 4-16 Selecting the Version of FreeRTOS to be Imported

If the destination folder for importing is not empty, the dialog box for confirming the overwriting of files will open. To overwrite the files, click on the [Yes] button.

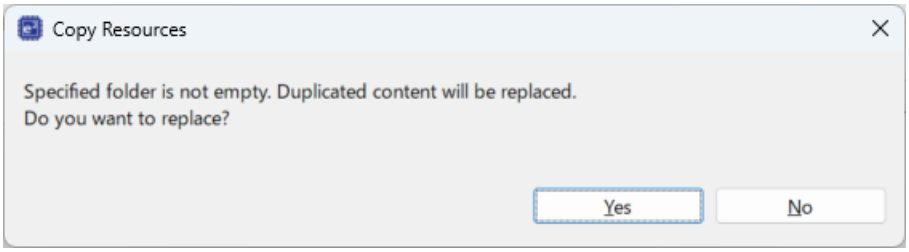


Figure 4-17 Confirming the Overwriting of Files

If you are running the e² studio for the first time or if the desired version does not appear in the [Version] list, you will have to download FreeRTOS from GitHub.
Click on [Manage RTOS Versions...] to display the [Download Confirmation] dialog box, select "FreeRTOS (with IoT libraries) for RX", and click on the [OK] button.

Figure 4-18 Selecting FreeRTOS

The [FreeRTOS (with IoT libraries) Module Download] dialog box will appear. Select the desired version and click on the [Download] button to download it.

Note: Specify a path name no longer than ten characters, such as "c:\afr", in [Module Folder Path] to avoid problems in the build process due to the limitation on the length of the full path name.

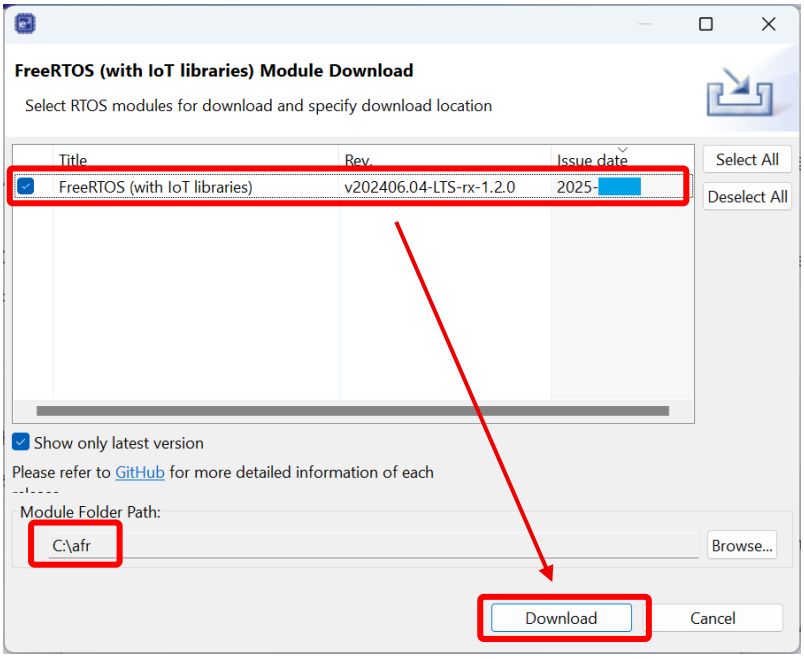


Figure 4-19 Version of FreeRTOS to be Downloaded

After downloading is complete, the dialog box for selecting the project to be imported shown in section 4.4(6), Selecting the project to be imported, will appear.

Note: The downloaded project refers to the GitHub repository at the following link.
https://github.com/renesas/iot-reference-rx

(6) Selecting the project to be imported

After a version of FreeRTOS has been selected, the [Import Projects] dialog box will appear.
Select the following project and click on the [Finish] button.

Confirm the compiler in the path name in parentheses when selecting the project.

- aws_ether_ck_rx65n_v2 (CC-RX)

Note: This application note explains the steps for an Ethernet-connected CC-RX compiler project as an example.

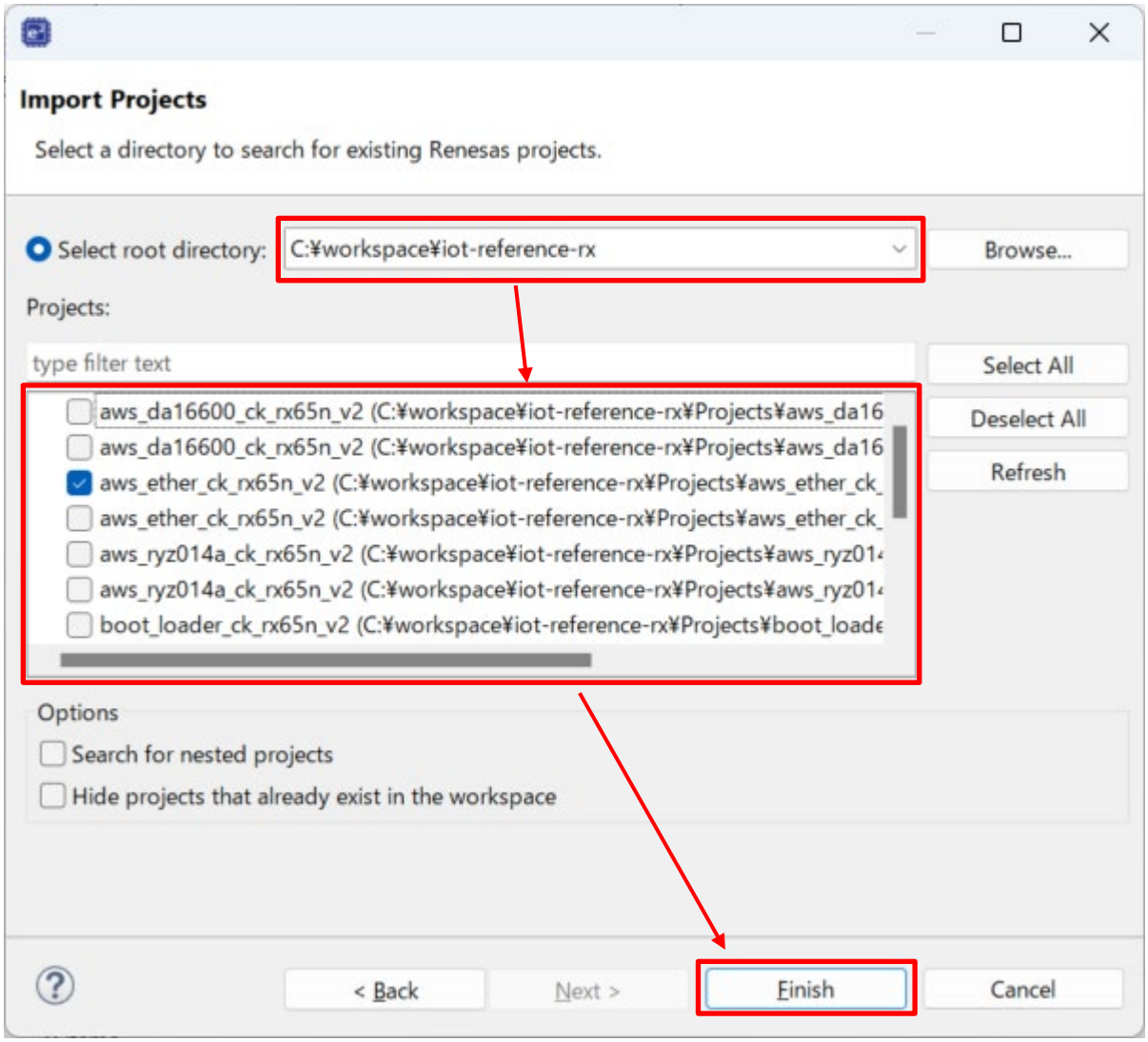


Figure 4-20 Selecting the Project to be Imported

(7) Confirming the imported project

On completion of importing, the project is registered with the e² studio as shown below.

If [Project Explorer] is not visible, click on [C/C++] as the perspective selection in the upper-right region of the window and then select [Window] → [Show View] → [Project Explorer].

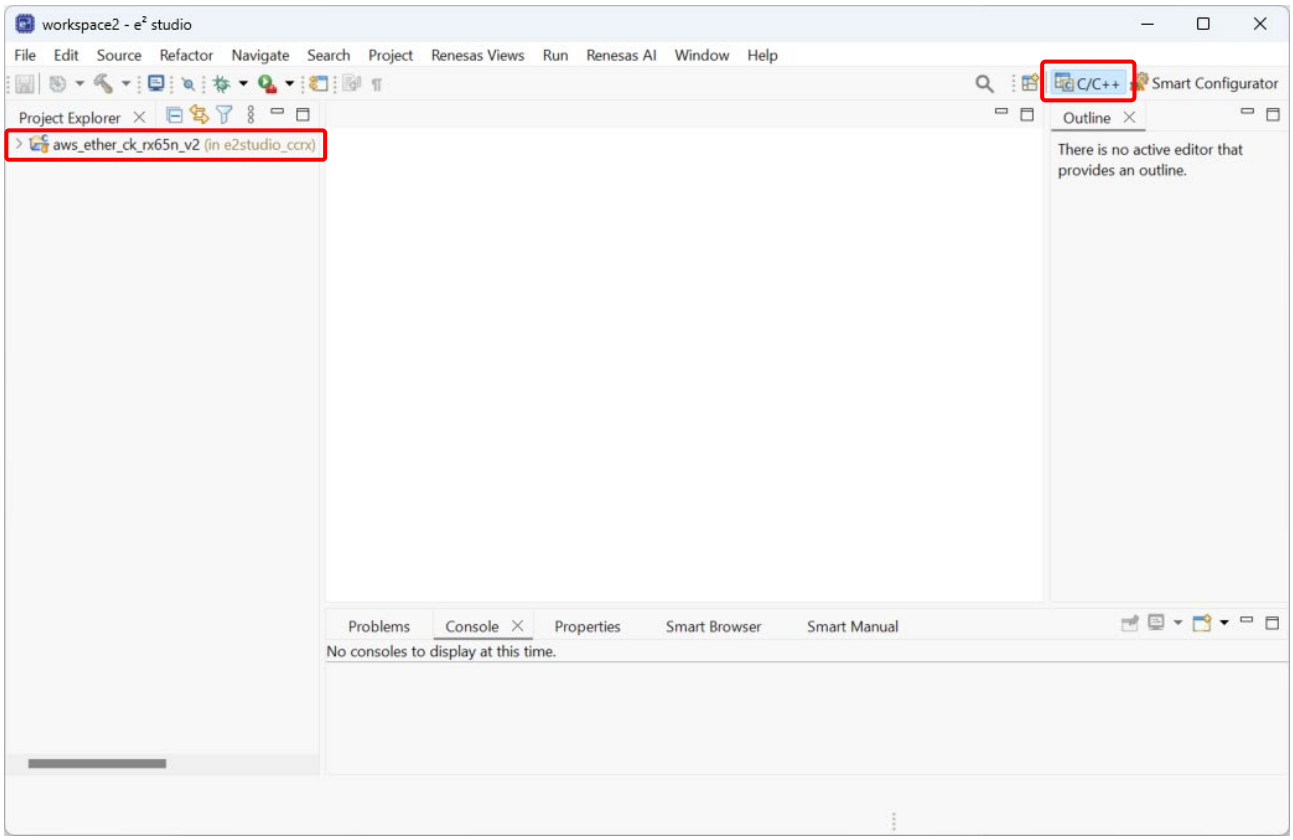


Figure 4-21 Window Displayed after Importing

The table below is a list of projects that can be imported.
You can select a project with the desired connectivity and compiler.

Table 4-1 Importable project

Path	Project	Compiler
C:\workspace\iot-reference-rx \Projects\aws_ether_ck_rx65n_v2\e2studio_ccrx	Demo project: Ethernet	CC-RX
C:\workspace\iot-reference-rx \Projects\aws_ether_ck_rx65n_v2\e2studio_gcc		GCC
C:\workspace\iot-reference-rx \Projects\aws_ryz014a _ck_rx65n_v2\e2studio_ccrx	Demo project: Cellular (RYZ014A)	CC-RX
C:\workspace\iot-reference-rx \Projects\aws_ryz014a _ck_rx65n_v2\e2studio_gcc		GCC
C:\workspace\iot-reference-rx \Projects\aws_da16600_ck_rx65n_v2\e2studio_ccrx	Demo project: Wi-Fi（DA16600）	CC-RX
C:\workspace\iot-reference-rx \Projects\aws_da16600_ck_rx65n_v2\e2studio_gcc		GCC
C:\workspace\iot-reference-rx \Projects\boot_loader_ck_rx65n_v2\e2studio_ccrx	Boot loader project	CC-RX
C:\workspace\iot-reference-rx \Projects\boot_loader_ck_rx65n_v2\e2studio_gcc		GCC

Note: Projects with the same connectivity for CC-RX and GCC cannot be imported at the same time.

4.5 Setting Up the Project

The program requires modification before the fleet provisioning demo is run.

4.5.1 Modifying the Configuration File

From the [Project Explorer] in the e^2 studio, open "aws_ether_ck_rx65n_v2/src/frtos_config/demo_config.h" and change the value for "ENABLE_FLEET_PROVISIONING_DEMO" to "1".

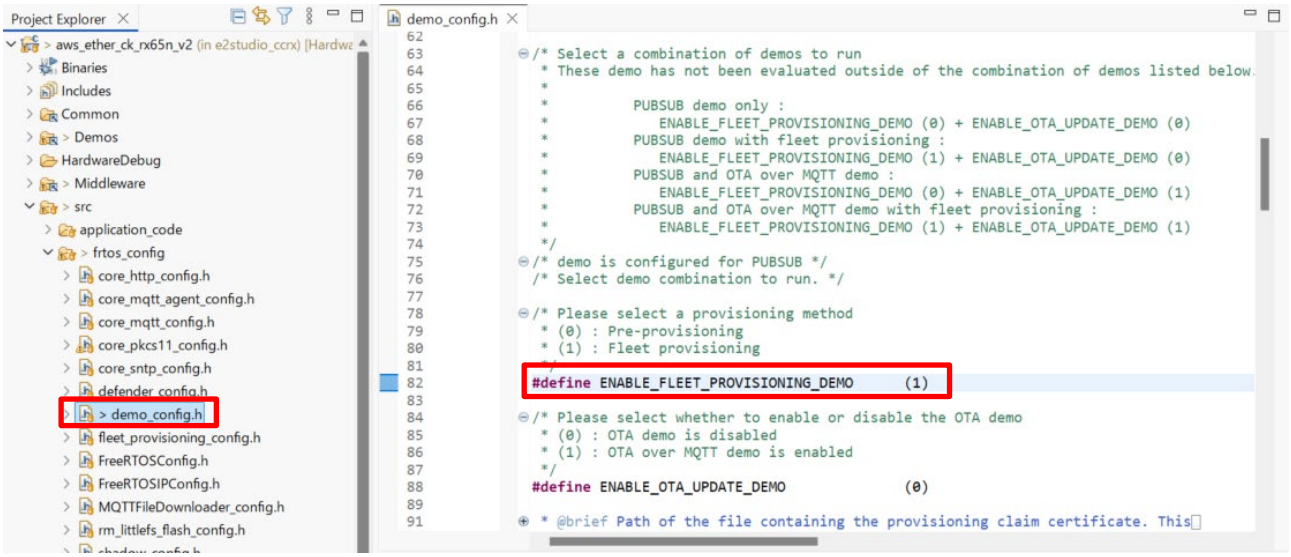


Figure 4-22 Location for Modification in demo_config.h

4.5.2 Settings for the Individual Connectivity Types

For a cellular or Wi-Fi connection, set up the corresponding connectivity module.

(1) Setting up the RYZ014A cellular module

When a cellular module is to be used for the AWS connection, set up the FIT module (r_cellular) for the RYZ014A cellular module.

Open "aws_ryz014a_ck_rx65n_v2.scfg" and click on the [Components] tab. Make settings for the following items for [r_cellular] to match the settings in the target SIM card.

- [Access point name]: Enter the access point name.
- [Access point login ID]: Enter the user ID for login.
- [Access point password]: Enter the password for login.
- [Authentication protocol type]: Enter "1" (PAP) or "2" (CHAP) as the type of authentication protocol.

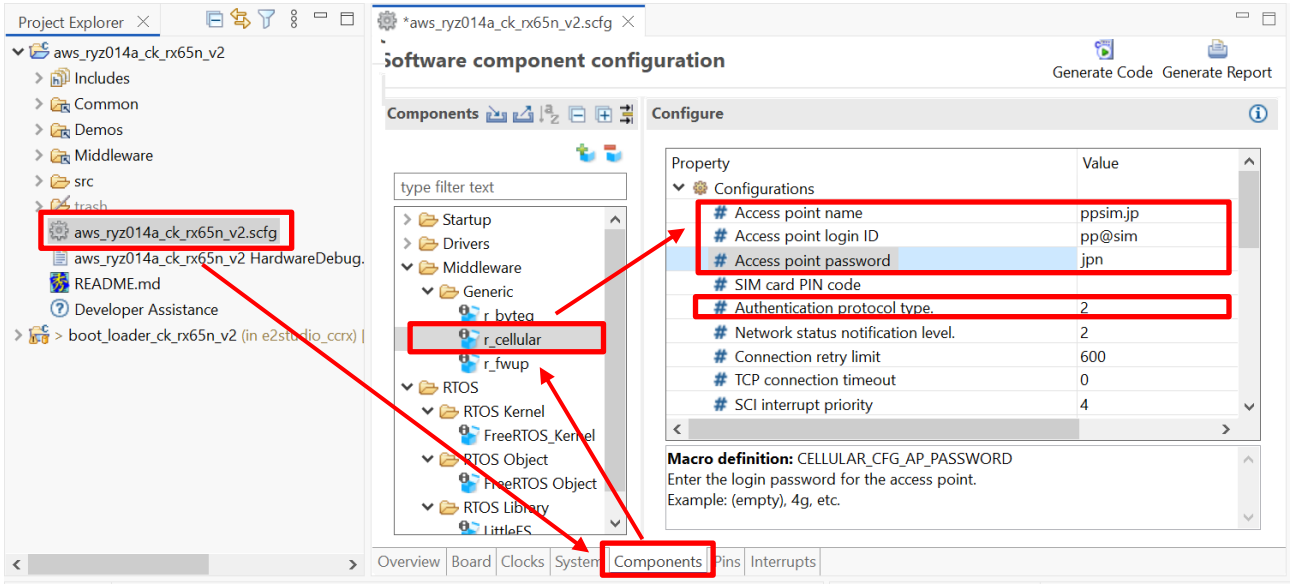


Figure 4-23 Setting up the FIT Module for the RYZ014A Cellular Module

(2) Setting up the DA16600 Wi-Fi module

When a Wi-Fi module is to be used for the AWS connection, set up the "aws_clientcredential.h" configuration file and the FIT module (r_wifi_da16xxx) for the DA16xxx Wi-Fi module.

(a) Setting the country code and timezone

Open "aws_da16600_ck_rx65n_v2.scfg" and click on the [Components] tab. Make settings for the following items for [r_wifi_da16xxx].

- [Timezone offset in hours]: Enter a value from -12 to 12 as the offset of the timezone from GMT.
- [Country code]: Enter the country code defined by the ISO 3166-1 alpha-2 standard such as US, JP, or CH.

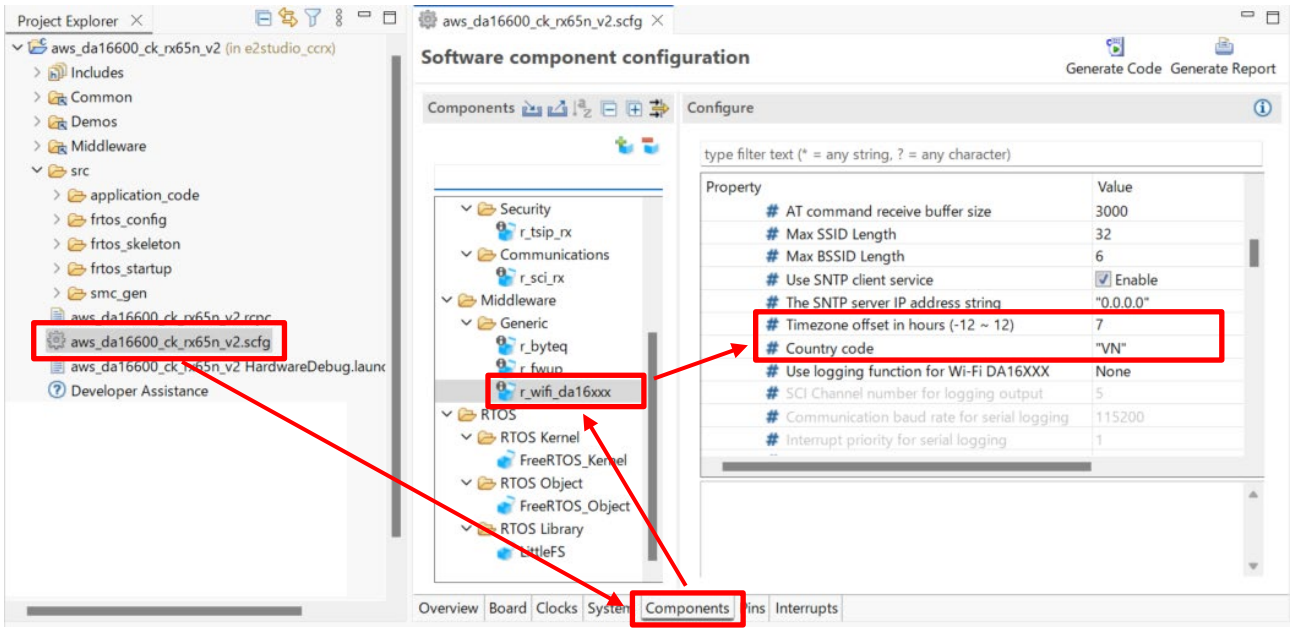


Figure 4-24 Setting up the FIT Module for the Wi-Fi DA16xxx Module

(b) Setting the Wi-Fi network

Set the Wi-Fi network for connection.
Make settings of the following macros in "src\application_code\include\aws_clientcredential.h".

- clientcredentialWIFI_SSID: Enter the access point name (SSID) for the Wi-Fi network.
- clientcredentialWIFI_PASSWORD: Enter the password for the Wi-Fi network.

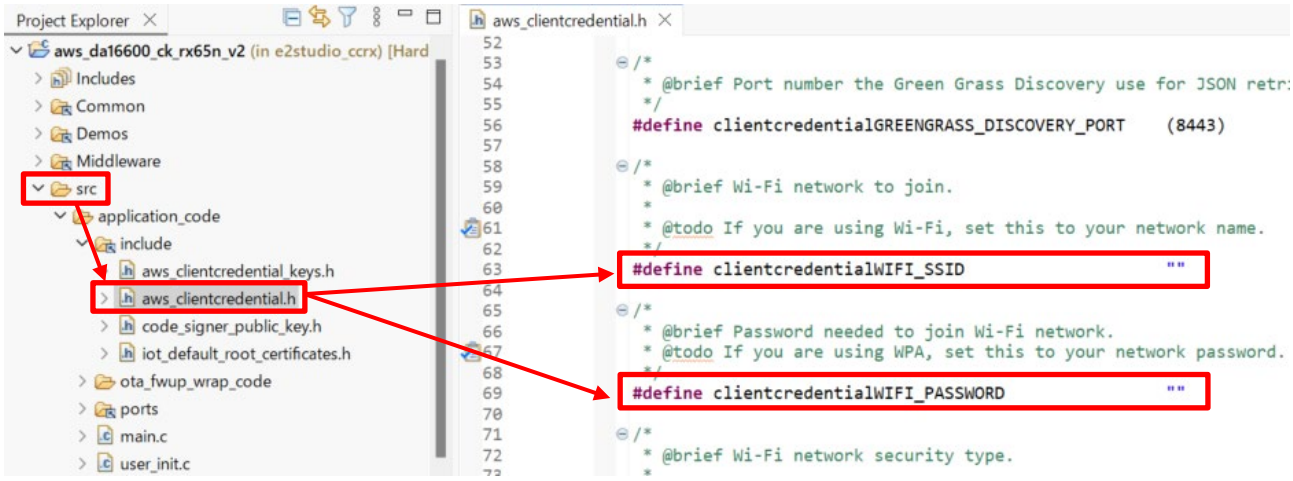


Figure 4-25 Setting the Wi-Fi Network

Note: Enter the character strings to be set between each of the pairs of double quotation marks.

4.6 Running the Project

4.6.1 Building and Downloading the Project

Build the project, write it to the device, and run the demo.

Firstly, right-click on "aws_ether_ck_rx65n_v2" in [Project Explorer] and select [Build Project] to build the project.

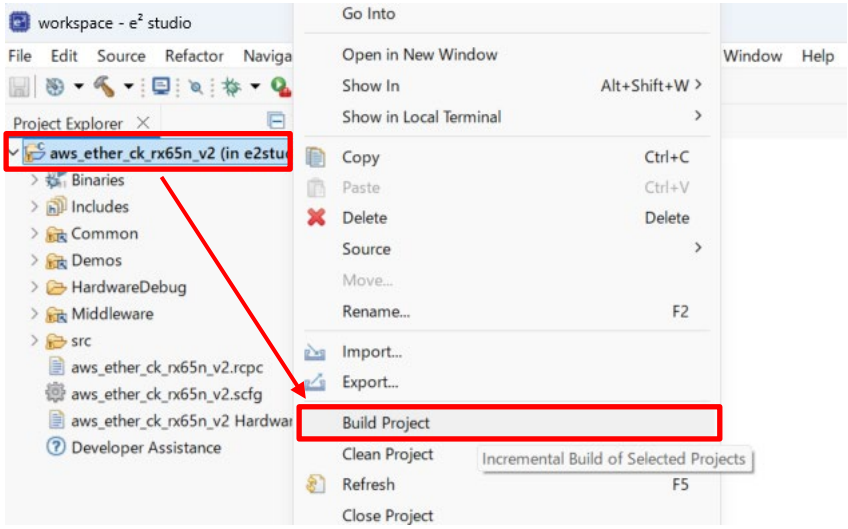


Figure 4-26 Building the Project

After confirming that no build error has occurred, select [Debug] from the [Run] menu of the e² studio to download the executable data generated by building to the device.

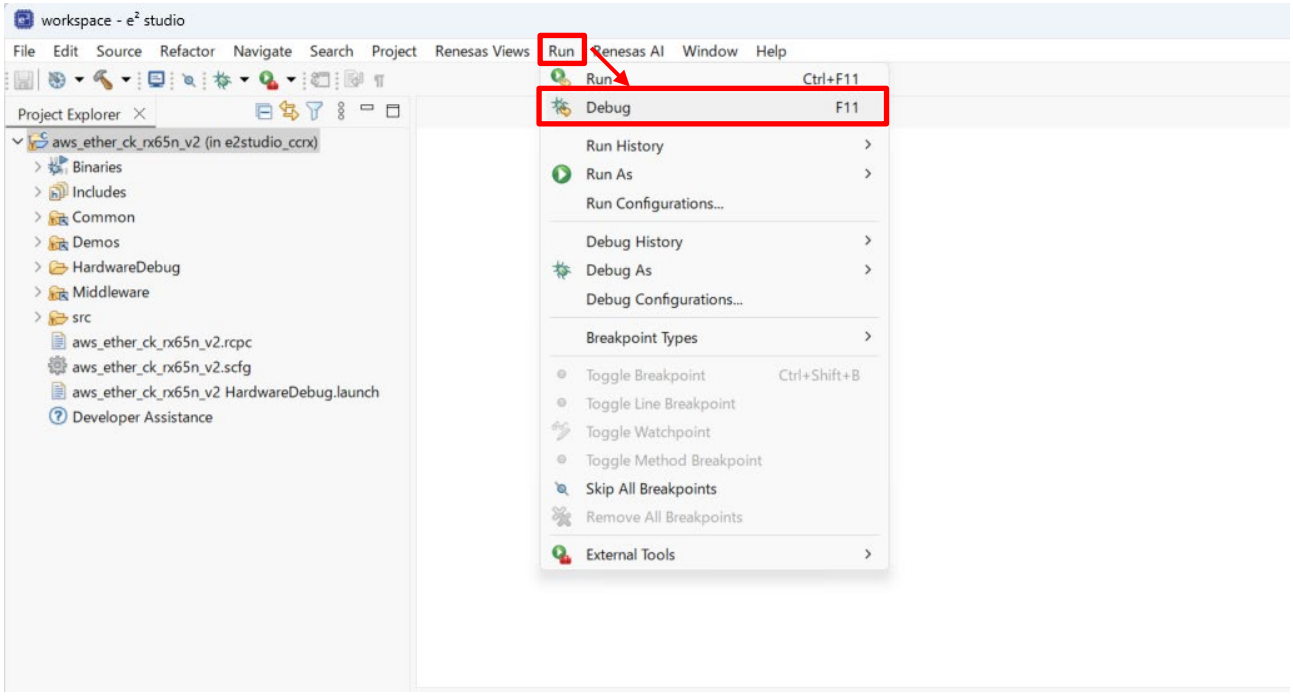


Figure 4-27 Executing [Debug] for the Project

4.6.2 Registering AWS IoT Information

Set AWS IoT information in Tera Term by running the aws_ether_ck_rx65n_v2 project. The information set through this process is written to data flash memory.

(1) Starting Tera Term and setting the serial port

Start Tera Term and specify the serial port to be connected to the target board.
Select [New Connection] from the [File] menu.
In the dialog box that appears, select [Serial] and the port connected to J10 on the CK-RX65N v2 board, then click on [OK].

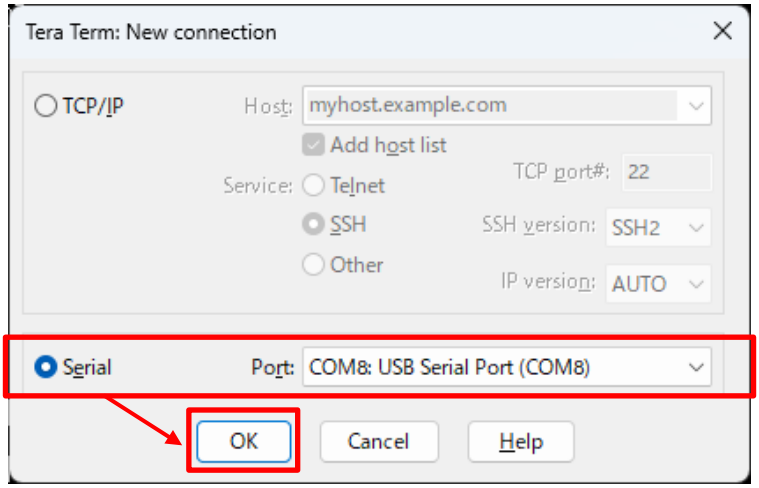


Figure 4-28 Selecting the Serial Port

(2) Setting the new-line code

Specify the new-line code for use in transmission and reception through the serial port.
Select [Terminal] from the [Setup] menu. Under [New-line] on the dialog box that appears, select "AUTO" for [Receive] and "CR+LF" for [Transmit] and then click on [OK].

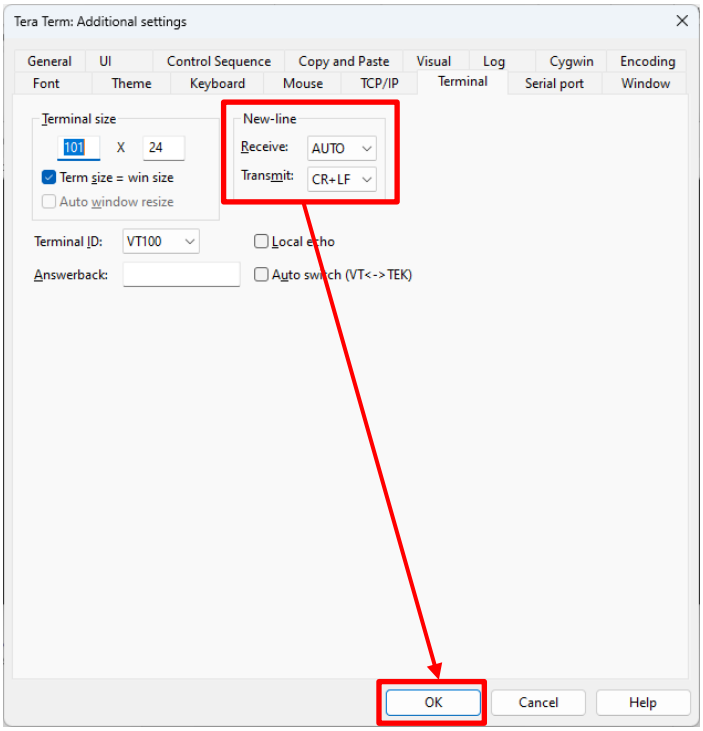


Figure 4-29 Setting the New-line Code

(3) Setting the serial communications speed

Select [Serial port…] from the [Setup] menu. In the dialog box that appears, select "115200" for [Speed], then click on [OK].

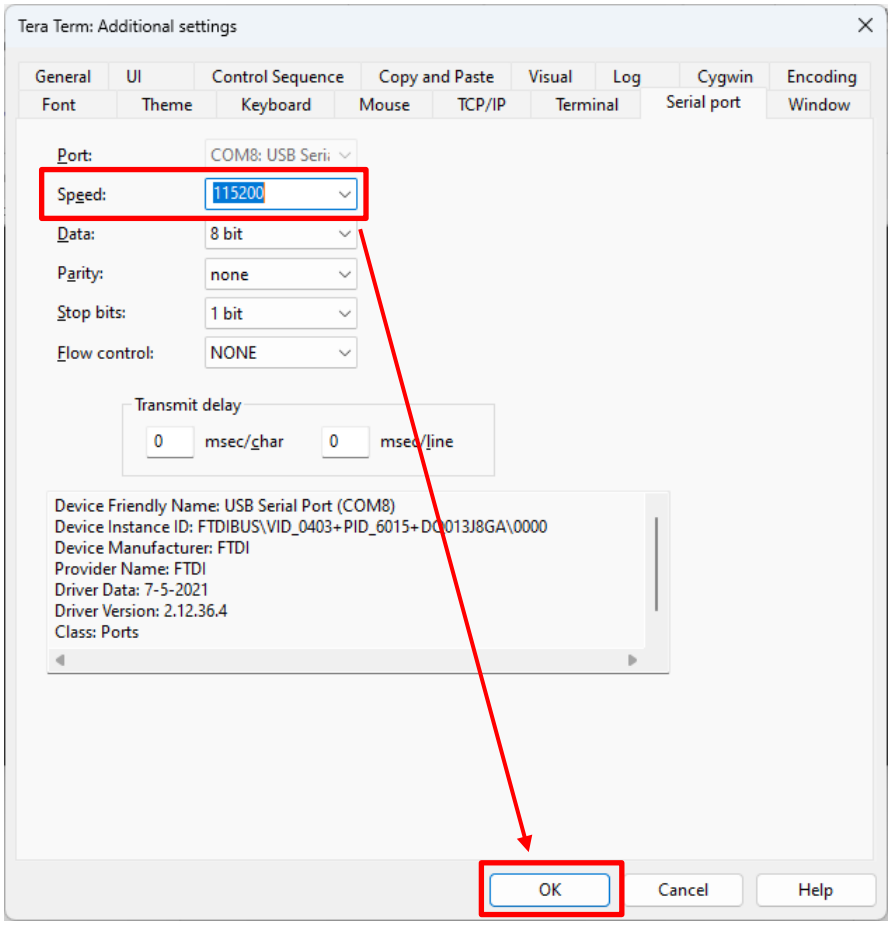


Figure 4-30 Setting the Serial Communications Speed

4.6.3 Running the Fleet Provisioning Demo and Confirming the Results

(1) Setting the MQTT test client

Setting the MQTT test client in the AWS management console enables monitoring of the MQTT data sent by the demo.

On [AWS IoT Core] of the AWS management console, select [MQTT test client], enter "#" under [Topic filter], and click on [Additional configuration].
Under [MQTT payload display], select "Display payloads as strings (more accurate)" and click on [Subscribe].

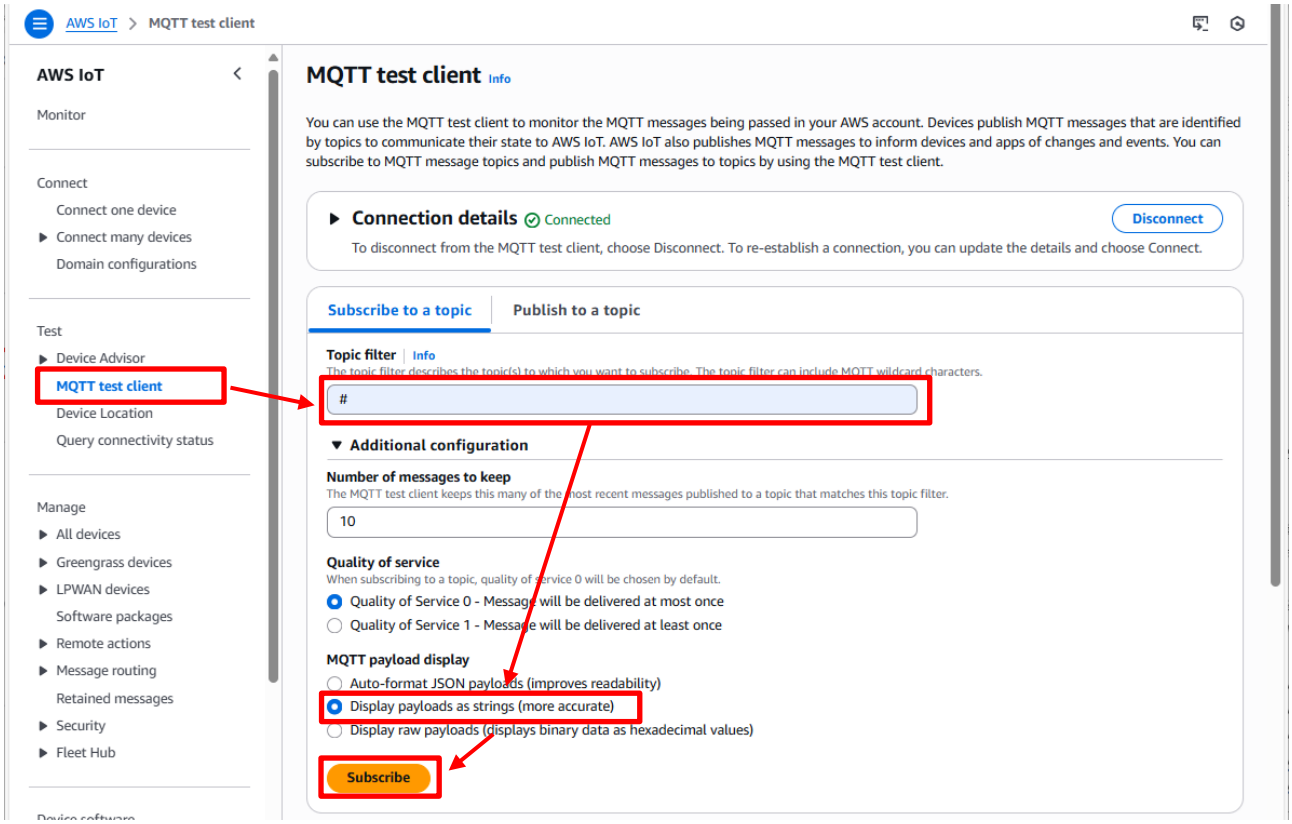


Figure 4-31 Setting the MQTT Test Client

(2) Running the fleet provisioning demo

In the e² studio, press the [Resume (F8)] button and the demo project will run.

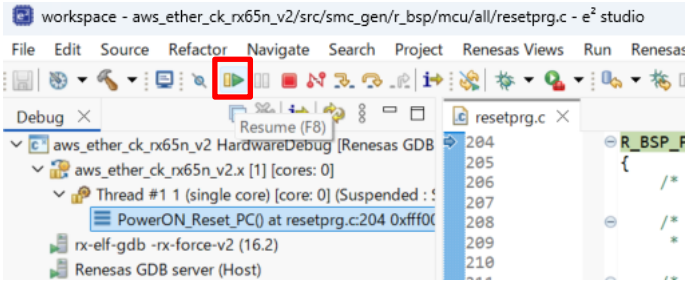


Figure 4-32 Running the Fleet Provisioning Demo

Execution of the demo project produces a menu in the Tera Term window as shown below. Within ten seconds of this, enter "CLI" and press the Enter key. Execution will switch to CLI mode.

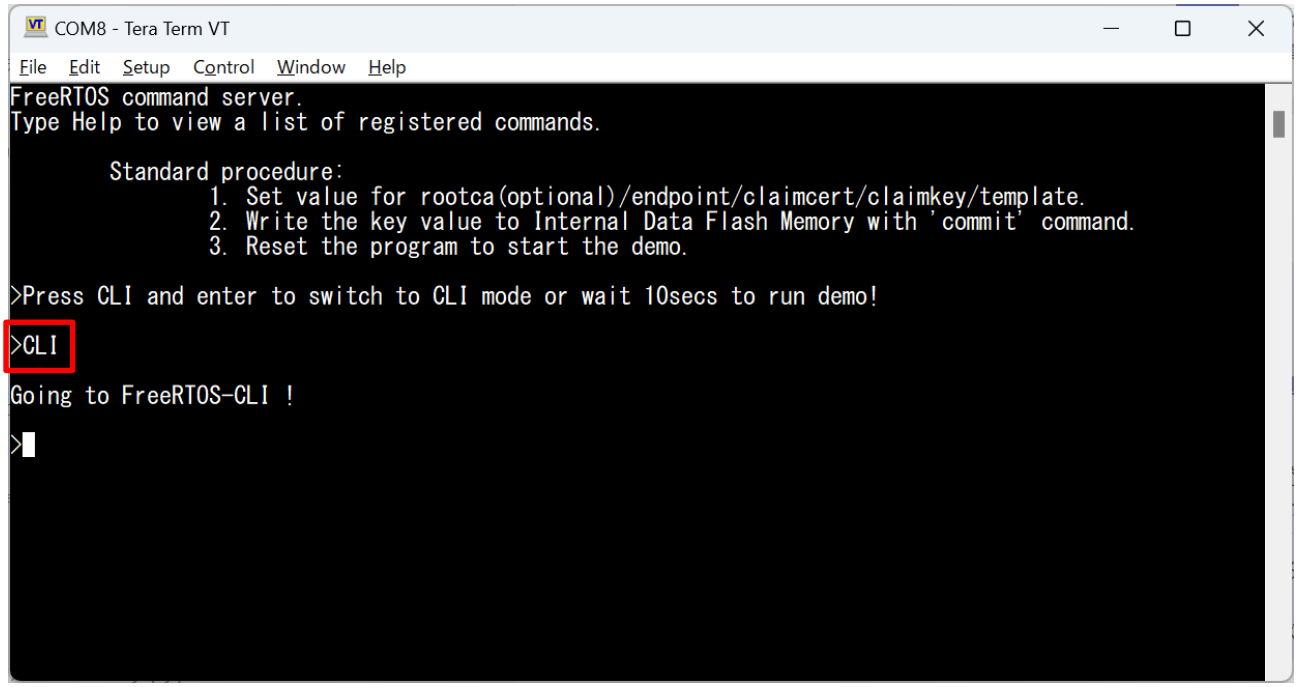


Figure 4-33 Switching to CLI Mode

(3) Registering AWS authentication information

Register AWS authentication information by command input in Tera Term.
The information entered here is recorded in data flash memory of the MCU on the CK-RX65N v2 board.

(a) Erasing existing credential information

As old information for previous execution of the demo may currently be stored, enter "format" in Tera Term and press the Enter key.
"Format OK !" will be displayed after all information has been erased from data flash memory.

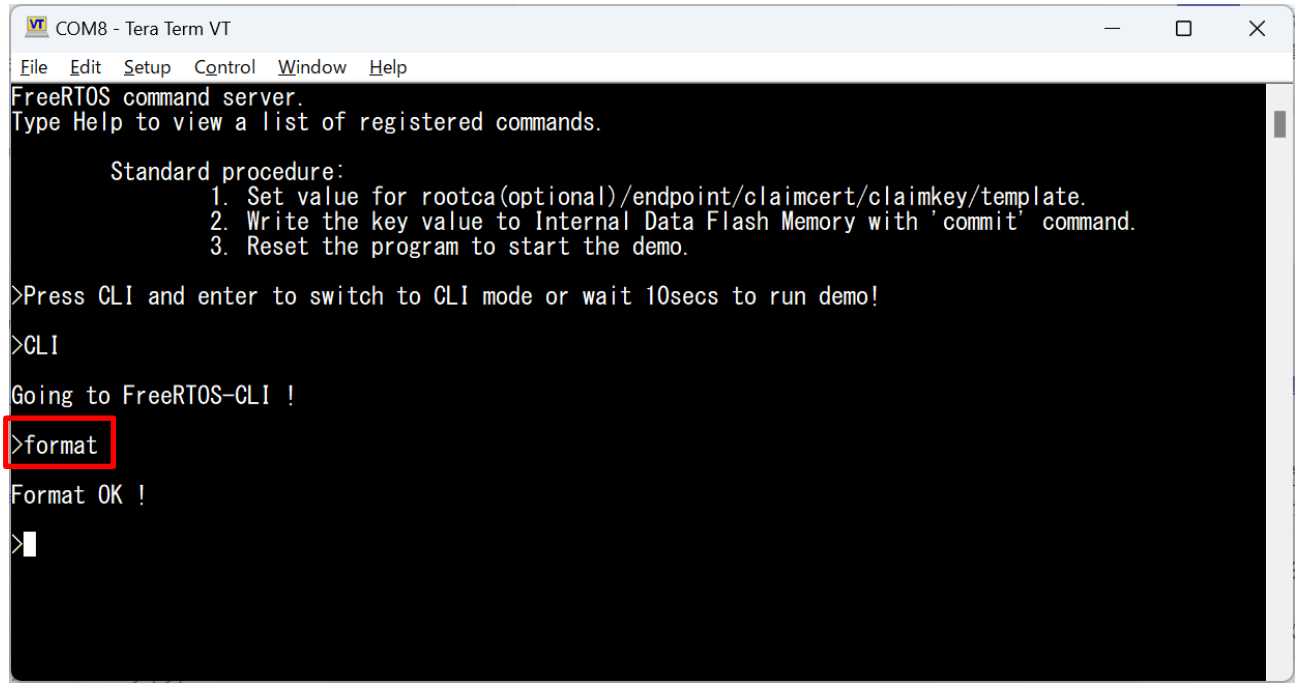


Figure 4-34 Formatting Data Flash Memory

(b) Checking the endpoint (domain)

The endpoint is equivalent to the destination (URL) for connection of the device. Registering the endpoint in the device will lead to connection of the device with the specified endpoint.
Enter the following command through the AWS CLI to check the endpoint.

```
> aws iot describe-endpoint --endpoint-type iot:Data-ATS --profile
<PROFILE_NAME>
```

After the command has been executed, the display in the window will be as shown below.

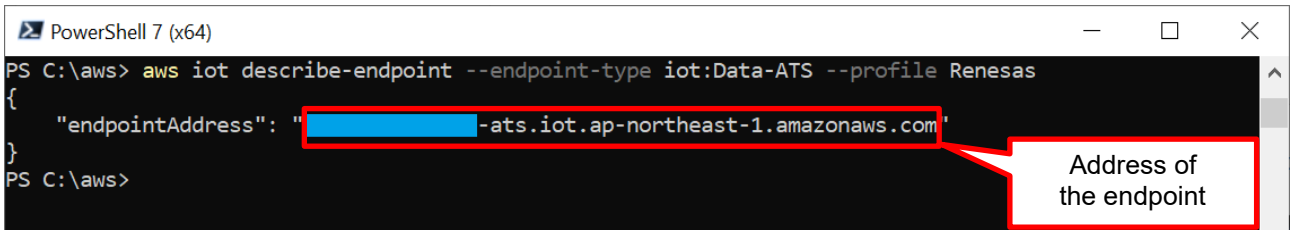


Figure 4-35 Checking the Endpoint

Make a note of the address of the endpoint (endpointAddress: text inside the red frame in the above figure) which is displayed. You will need it in a later process.

(c) Entering the endpoint name

Execute the following command in Tera Term to register the name of the endpoint with the target board.

```
conf set endpoint <ENDPOINT_NAME>
```

- Enter the name of the endpoint checked in section 4.6.3(3)(b) as *<ENDPOINT_NAME>*.

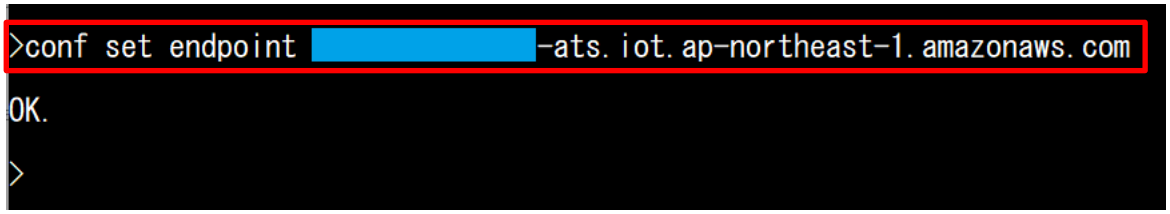


Figure 4-36 Entering the Name of the Endpoint

(d) Entering the provisioning template

Execute the following command in Tera Term to register the name of the provisioning template with the target board.

```
conf set template <TEMPLATE_NAME>
```

- Enter the name of the provisioning template created in section 4.3.5(2) as *<TEMPLATE_NAME>*.

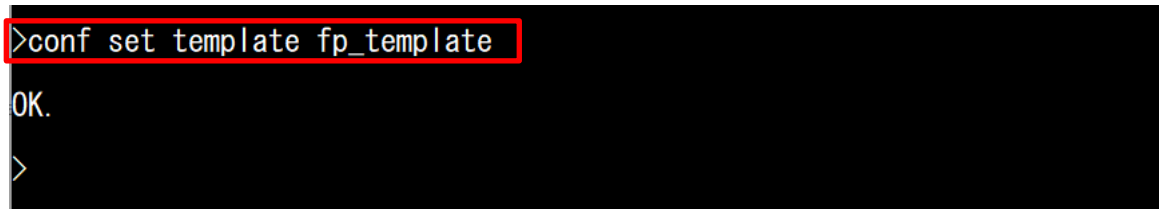


Figure 4-37 Entering the Name of the Provisioning Template

(e) Entering the provisioning claim certificate

Register the provisioning claim certificate (fp-certificate.pem.crt) that was created in section 4.3.3(1), Creating and downloading a certificate and a key pair for the provisioning claim, with the target board.

- Enter "`conf set claimcert `" in Tera Term, then drag and drop the claim certificate (fp-certificate.pem.crt) to the Tera Term window. Be sure to enter a space after "`claimcert`".
- Select "Binary" under [Send File] in the [Tera Term: File Drag and Drop] dialog box and click on the [OK] button.
- Finally, press the Enter key while Tera Term is in focus.

Note: Change the new-line codes in the claim certificate file to LF before dragging and dropping the file.

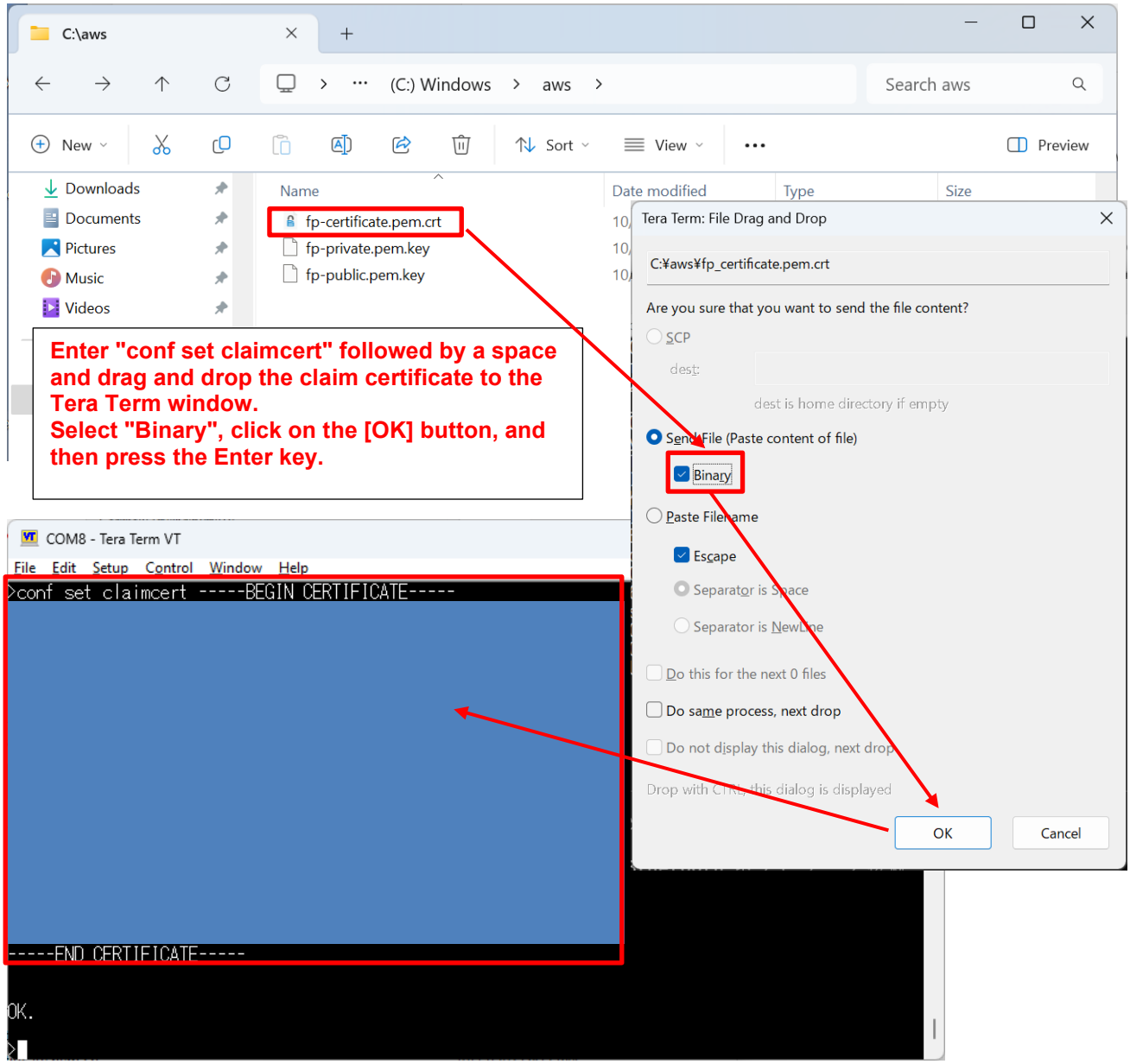


Figure 4-38 Entering the Provisioning Claim Certificate

(f) Entering the provisioning claim private key

Register the provisioning claim private key (fp_private.pem.key) that was created in section 4.3.3(1), Creating and downloading a certificate and a key pair for the provisioning claim, with the target board.

- Enter "`conf set claimkey `" in Tera Term, then drag and drop the claim private key (fp_private.pem.key) to the Tera Term window. Be sure to enter a space after "`claimkey`".
- Select "Binary" under [Send File] in the [Tera Term: File Drag and Drop] dialog box and click on the [OK] button.
- Finally, press the Enter key while Tera Term is in focus.

Note: Change the new-line codes in the claim private key file to LF before dragging and dropping the file.

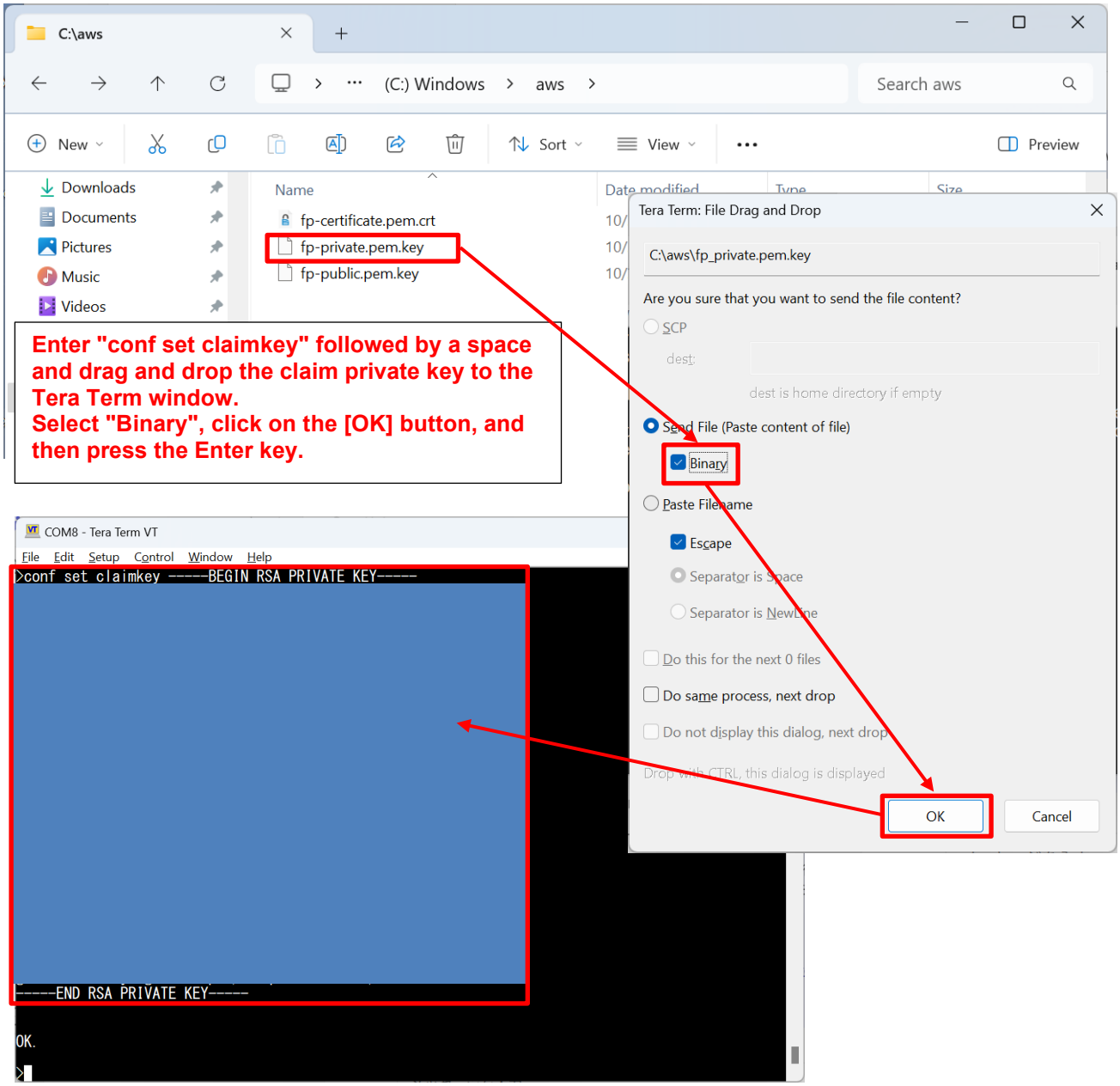


Figure 4-39 Entering the Claim Private Key

(g) Writing the settings to data flash memory

Execute the following command in Tera Term to commit the AWS IoT settings (write the settings to data flash memory)

```
> conf commit
```

On completion of writing, "Configuration save xxxx bytes..." will be displayed as shown below.

```
>conf commit

Configuration save 3024 bytes to Data Flash.  Total used size is 3024 bytes .
```

Figure 4-40 Writing the Settings to Data Flash Memory

(h) Applying a reset

Execute the following command in Tera Term to apply a software reset and restart the firmware.

```
> reset
```

The demo will start if nothing is entered in Tera Term for ten seconds after the reset. Confirm that Tera Term displays a communications log as shown below.

```
>reset

FreeRTOS command server.
Type Help to view a list of registered commands.

        Standard procedure:
                1. Set value for rootca(optional)/endpoint/claimcert/claimkey/template.
                2. Write the key value to Internal Data Flash Memory with 'commit' command.
                3. Reset the program to start the demo.

>Press CLI and enter to switch to CLI mode or wait 10secs to run demo!

>0 10000 [MAIN_TASK] FreeRTOS_AddEndPoint: MAC: 79-03 IPv4: c0a80b0cip

1 10000 [IP-Task] vIPSetDHCP_RATimerEnableState: Off

2 10000 [IP-Task] prvCloseDHCPSocket[79-03]: closed, user count 0

3 10002 [ETHER_RECEI] Deferred Interrupt Handler Task started
```

Figure 4-41 Resetting the Demo

4.6.4 Confirming the Results of Execution

A log of execution will be displayed in the Tera Term window after the fleet provisioning demo is reset as described in section 4.6.3(3)(h), Applying a reset.
In this demo, a thing is registered and a device-specific certificate is attached to it through fleet provisioning. After that, the PubSub (MQTT communications) demo is run.

(1) Confirming the results of the fleet provisioning demo

After the start of the demo, fleet provisioning is first executed.
If "Demo completed successfully." and "Fleet Provisioning Task Finished" are displayed in the log, the fleet provisioning demo was successfully completed.

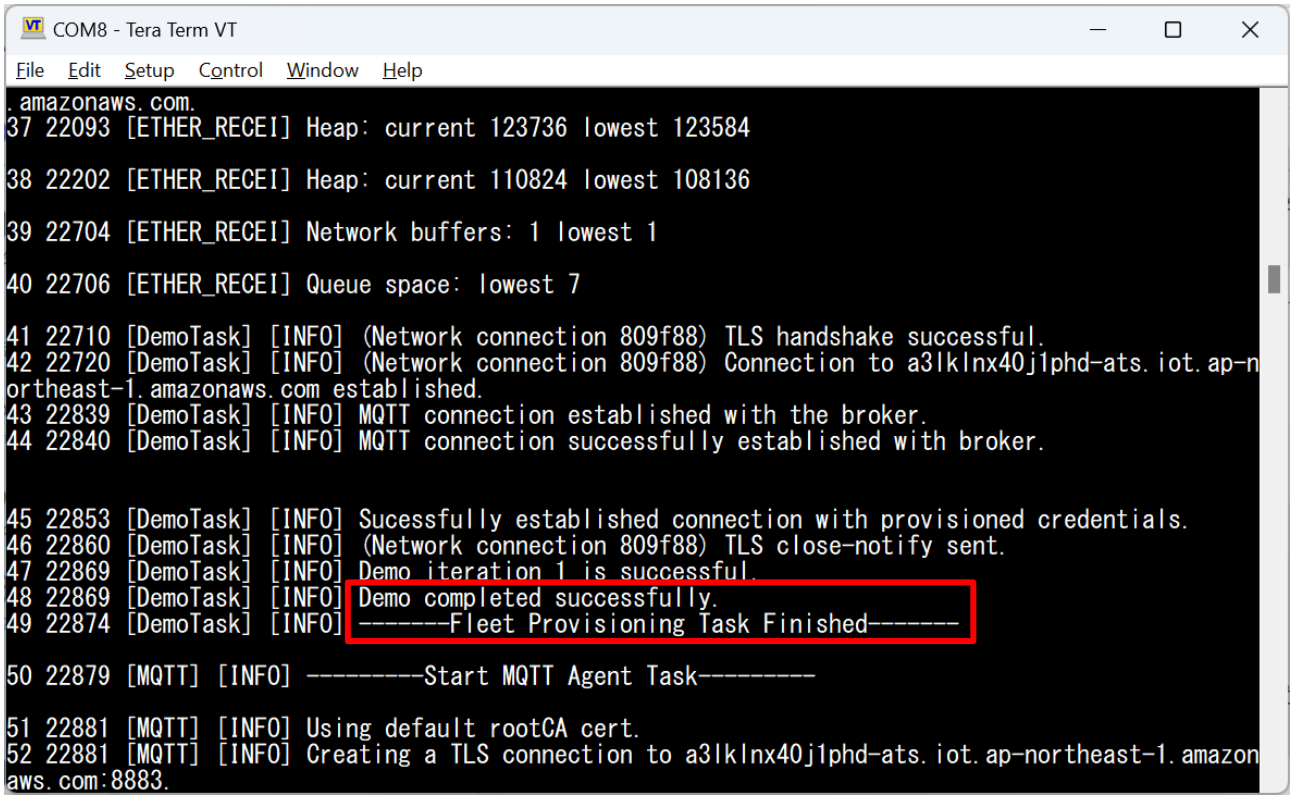


Figure 4-42 Log of Execution when the Fleet Provisioning Demo was Successful

(2) Running the PubSub demo

After the fleet provisioning demo, the PubSub demo runs with the use of the device-specific certificate and private key obtained from AWS.
The PubSub demo publishes and subscribes to messages through MQTT communications.
You can see that "Successfully sent QoS 0 publish to topic:" and "Received incoming publish message Task..." appear in the Tera Term log as shown below.

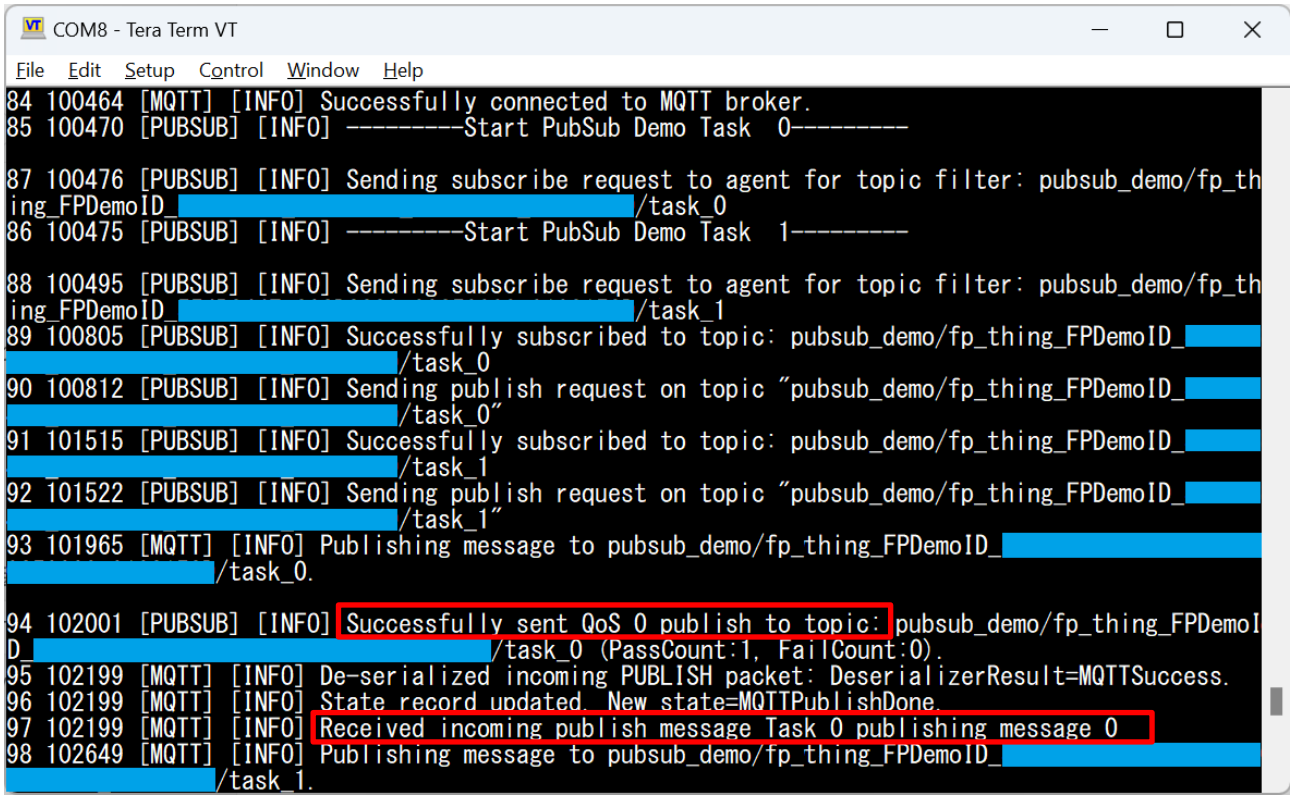


Figure 4-43 Execution Log when the PubSub Demo was Successful

The PubSub demo sends ten messages, message 0 to message 9, through each of PubSub demo task 0 and task 1, that is, the demo sends a total of 20 messages.
After message 9 is sent through task 1 and "Unsubscribe successfully from task 1" is displayed at the end of the log, the PubSub demo is completed.

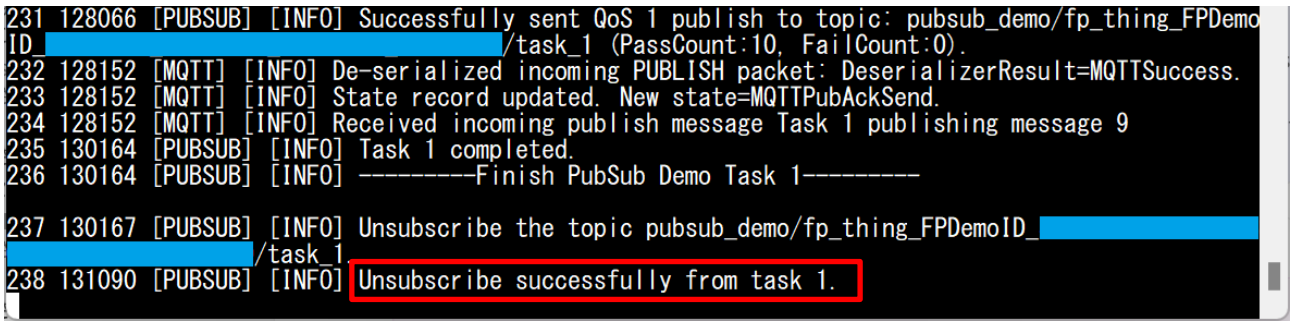


Figure 4-44 Completion of the PubSub Demo

You can also check the MQTT messages sent from the CK-RX65N v2 board to AWS in the [MQTT test client] window that was set on [AWS IoT Core] of the AWS management console in section 4.6.3(1), Setting the MQTT.

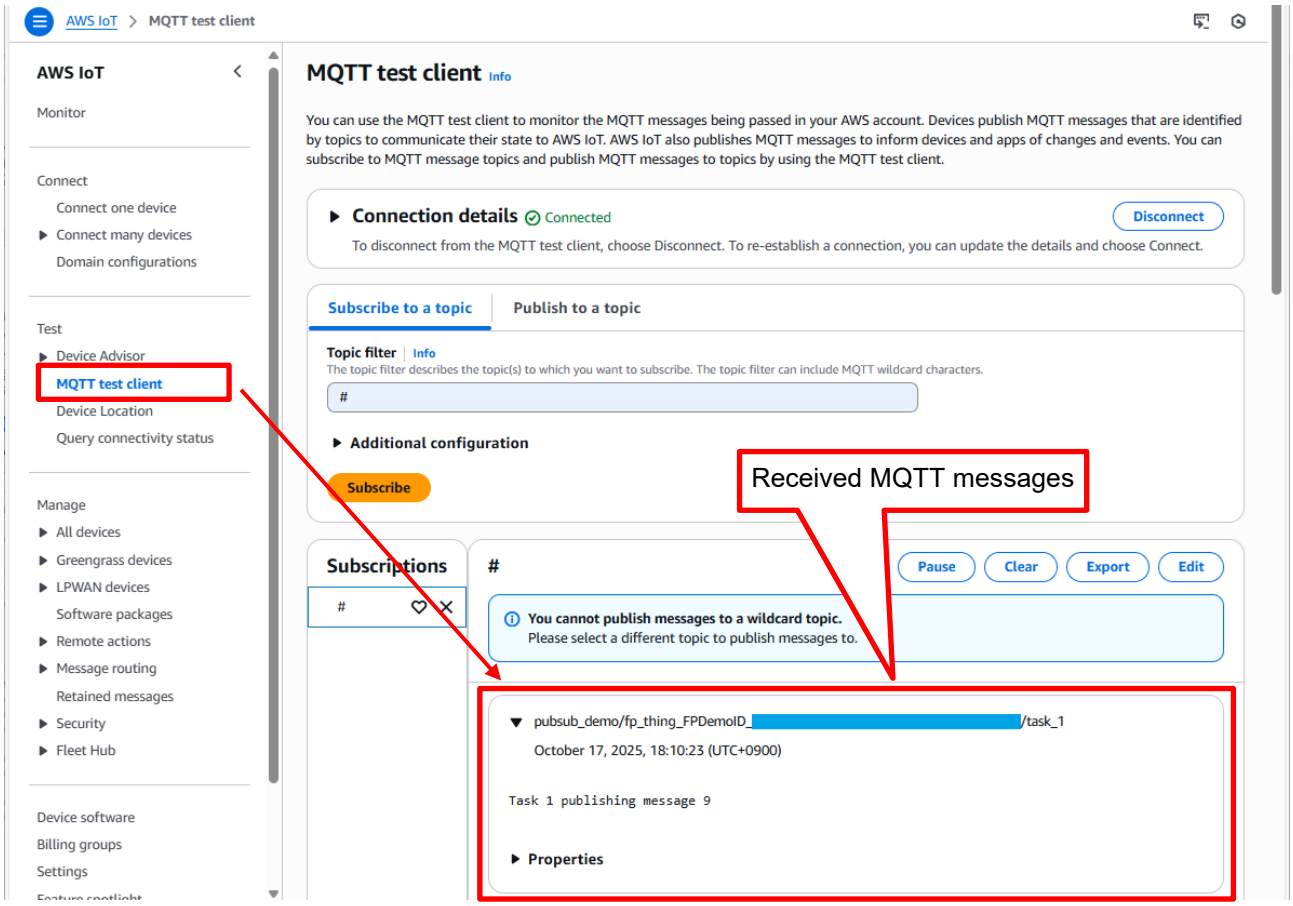


Figure 4-45 MQTT Test Client after the PubSub Demo was Successful

(3) Checking the thing name

You can check the name of the thing registered by the fleet provisioning demo in "Received AWS IoT Thing name" in the Tera Term log that was produced during the fleet provisioning demo.
This thing name is required in OTA update execution with the use of fleet provisioning.

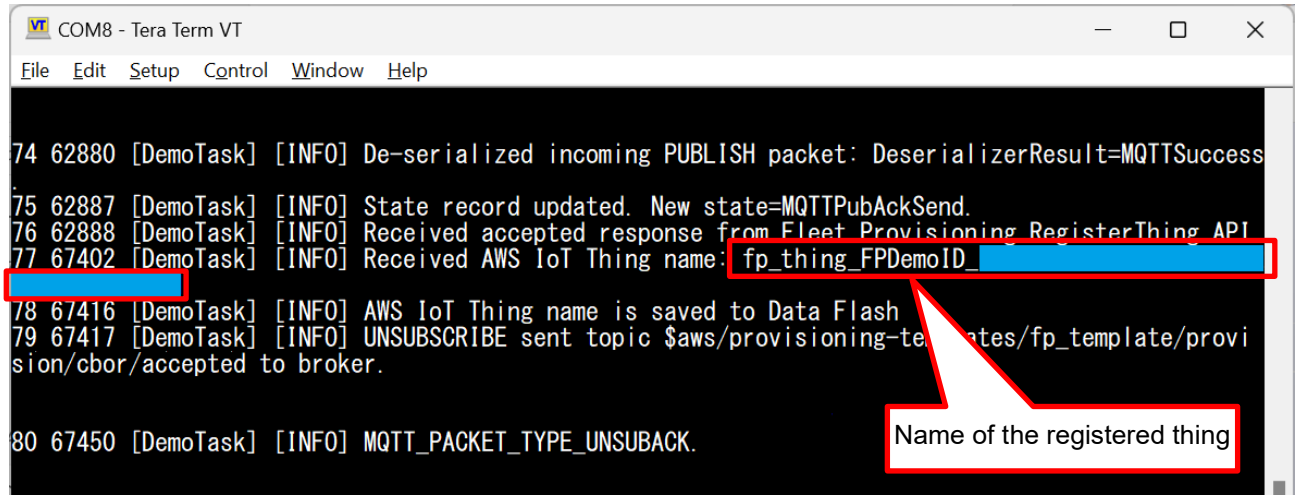


Figure 4-46 Log for Reception of the Thing Name

The following rule is used to create the name of a thing.

<THING_NAME_PREFIX>FPDemoID_xxxxxxxx_xxxxxxxx_xxxxxxxx_xxxxxxxx

(1) (2) (3)

(1) *<THING_NAME_PREFIX>*: This is the character string of the prefix specified in the json file for the provisioning template as described in section 4.3.5(1), Creating a json file for a template.
(2) Fixed character string: This is fixed to "FPDemoID".
(3) Unique MCU ID: This is the ID specific to the MCU on your CK-RX65N v2 board.

The name of the registered thing can be checked in the window for [AWS IoT Core] of the AWS management console. Select [All devices] → [Things], find the thing name that matches the name in the Tera Term log, and click on it.
Make note of the ARN of the thing displayed here if an OTA update is to be executed.

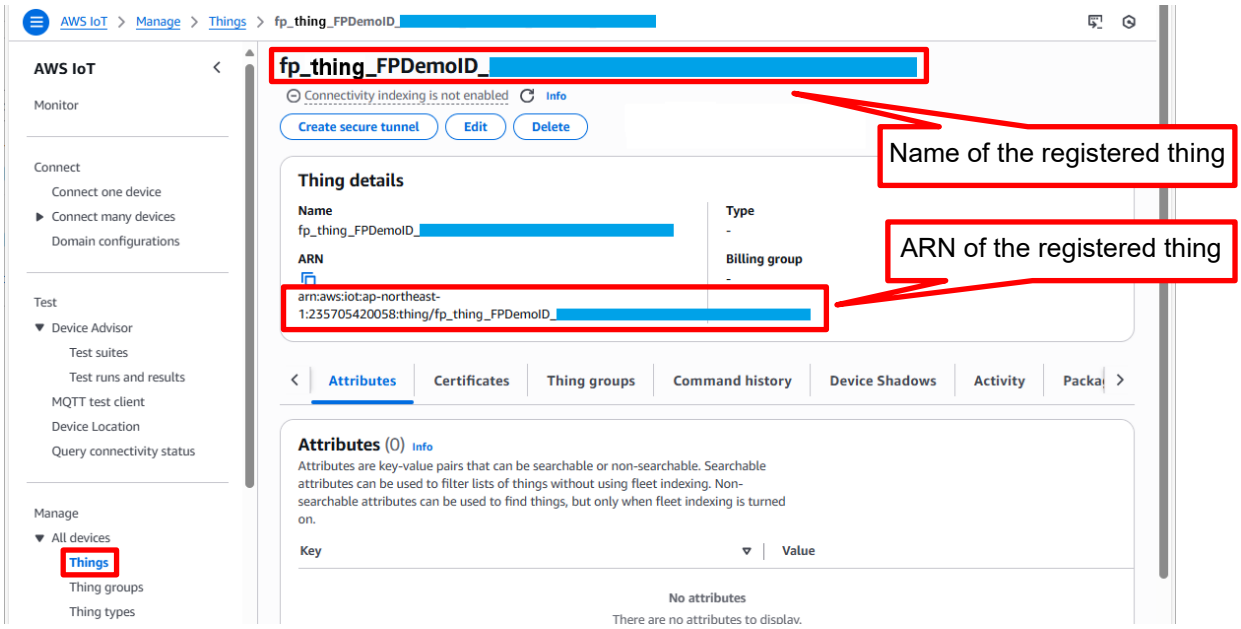


Figure 4-47 ARN of the Thing

(4) Checking the device certificate

By checking the registered thing, you can confirm that the device-specific certificate generated and assigned by fleet provisioning has been attached and activated. It is shown next to "Received certificate with Id:".

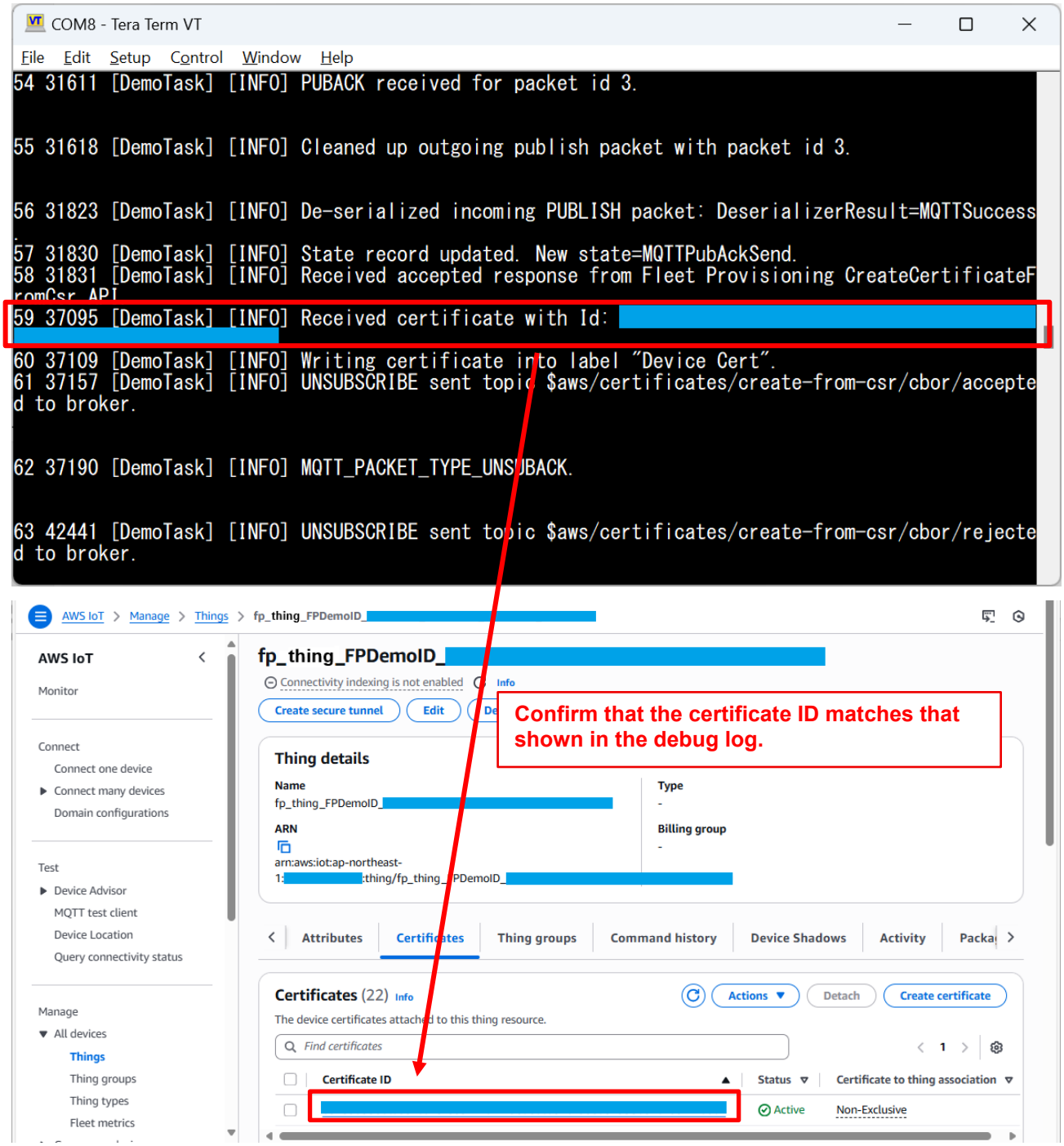


Figure 4-48 Checking the Device Certificate

4.6.5 OTA Update Execution through the Fleet Provisioning Project

The fleet provisioning demo application can be executed alongside the OTA update application.

For the basic procedure for an OTA update, refer to the application note *RX Family: How to Implement FreeRTOS OTA Using Amazon Web Services (202406-LTS Version)* (R01AN7662).

When executing an OTA update alongside the fleet provisioning application, please note the following steps, which differ from the procedure described in the application note above.

(a) Set up the tool environment according to the steps in “Section 2 Preparations in Advance” of R01AN7662.
The steps described in “2.1, Installing Tera Term”, and “2.5, Installing the AWS Command Line Interface (CLI)”, are not required, as they have already been completed in this application note.

(b) Ensure that AWS CLI can run in the development environment.
If the steps described in “4.2, Preparing AWS” in this application note have already been completed, you may skip the steps in “Section 3 Setting Up AWS” of R01AN7662 .

(c) Configure the policy by performing “Section 3.3.1 Setting Policies” of R01AN7662 to create the OTA policy.
Alternatively, you may reuse the policy created in “Section 4.3.2 Creating Policies for Things Created through Fleet Provisioning” of this application note.

(d) Perform the steps in “Section 3.3.2 Creating an Amazon S3 Bucket”, and “Section 3.3.3 Allocating Permission for OTA Execution to an IAM User” described in R01AN7662, to configure AWS settings for OTA.

(e) You may skip the steps in “Section 3.3.4 Registering a Device as a Thing in AWS IoT” of R01AN7662.
However, if the steps in “Section 4.3 AWS Settings for Fleet Provisioning” in this application note have not been performed, please perform them.

(f) Perform the steps in “Section 4.1 Generating a Key Pair and Certificate” of R01AN7662 to create the key pair and certificate for OTA.

(g) Perform the steps in “Section 4.4 Creating a Sample Project” described in this application note to import the Fleet Provisioning demo project.
When selecting the project in “Section 4.4(6) Selecting the project to be imported”, make sure to also check and import the bootloader project (boot_loader_ck_rx65n_v2).
You may skip the procedure described in “Section 4.2.1 Creating the Sample Projects, in R01AN7662”.

(h) Perform the steps from “Section 4.2.2 Setting Up Projects” to “Section 4.2.4 Creating the Initial Firmware” of R01AN7662 to create the initial firmware for OTA Fleet Provisioning.

Note: When performing the steps in “Section 4.2.2(2)”, make sure to also set the Fleet Provisioning demo setting to “1” as shown below:

```
/* Please select a provisioning method
 * (0) : Pre-provisioning
 * (1) : Fleet provisioning
 */
#define ENABLE_FLEET_PROVISIONING_DEMO      (1)

/* Please select whether to enable or disable the OTA demo
 * (0) : OTA demo is disabled
 * (1) : OTA over MQTT demo is enabled
 */
#define ENABLE_OTA_UPDATE_DEMO              (1)
```

(i) Execute “Section 4.2.5 Registering AWS IoT Information” of R01AN7662 to register AWS credential information in the demo application.
Since the steps for registering credential information to the CLI from “Section 4.2.5(6) Register the device certificate” onwards differ slightly, replace them with the following operations:

1. Enter the endpoint (see section 4.6.3(3)(c) in this application note)
2. Enter the provisioning template (see section 4.6.3(3)(d) in this application note)
3. Enter the provisioning claim certificate (see section 4.6.3(3)(e) in this application note)
4. Enter the provisioning claim private key (see section 4.6.3(3)(f) in this application note)
5. Enter the certificate for code-signing verification (see section 4.2.5(9) in R01AN7662)
6. Write the configuration values to data flash (see section 4.2.5(10) in R01AN7662)
7. Perform a reset (see section 4.2.5(11) in R01AN7662)

Note: The operations in “Section 4.2.5(6) Register the device certificate”, and “Section 4.2.5(7) Register the private key” in R01AN7662 are not required.

(j) Perform the steps in “Section 5 Updating the Firmware” of R01AN7662 to create the update firmware and execute OTA.

Note: In “Section 5.2.3, Executing an OTA Job”, enter the target ARN (the Thing for OTA execution) in the JSON file using the Thing name assigned by Fleet Provisioning, which was confirmed in “Section 4.6.4(3) Checking the thing name”, in this application note.

5. Summary

As mentioned earlier, there are multiple provisioning methods and the methods provide various levels of security.

Selecting and deploying a provisioning method that is appropriate in terms of the actual application in the target market, the scale of the system (number of devices), and the required level of security is essential.

However, owning, managing, and operating a secure manufacturing facility in-house in order to implement provisioning functionality is not a simple matter. This is why the fleet provisioning method had gained so much attention as an approach to the device provisioning process and this is probably the reason for the rapid growth of the market demand for this method.

The method of provisioning described in this document is only one example, so it will not satisfy the requirements of all users. Nevertheless, the information presented here will help deepen the user’s understanding of the advantages and disadvantages of the deployment of fleet provisioning. We hope that this document will help users in the smooth building of production lines.

6. Web Sites and Support Information

AWS re:Post: https://repost.aws

Renesas FreeRTOS GitHub: https://github.com/renesas/iot-reference-rx

7. Supplementary Information

7.1 Points to Keep in Mind when Operating Multiple Devices within the Same LAN Environment

Addresses assigned from the vendor ID of Renesas Electronics Corporation are used as the MAC addresses in the sample code.

When using the sample program to operate multiple devices at the same time within the same LAN environment, the MAC addresses must be changed to avoid duplication.

The sample program may not operate properly if the same MAC addresses are duplicated among multiple devices.

The procedure for changing the MAC addresses is described below.

Open "aws_ether_ck_rx65n_v2.scfg" in the Smart Configurator and select the [Components] tab.

In the tree, select [RTOS] → [RTOS Kernel] → [FreeRTOS_Kernel], and then under [Property] set hexadecimal values of your choice as "Value" for "MAC address 0" to "MAC address 5".

Enter values in the format 0xXX (where XX represents a hexadecimal value of your choice).

When creating commercial products for sale, be sure to use MAC addresses that have been submitted to the IEEE.

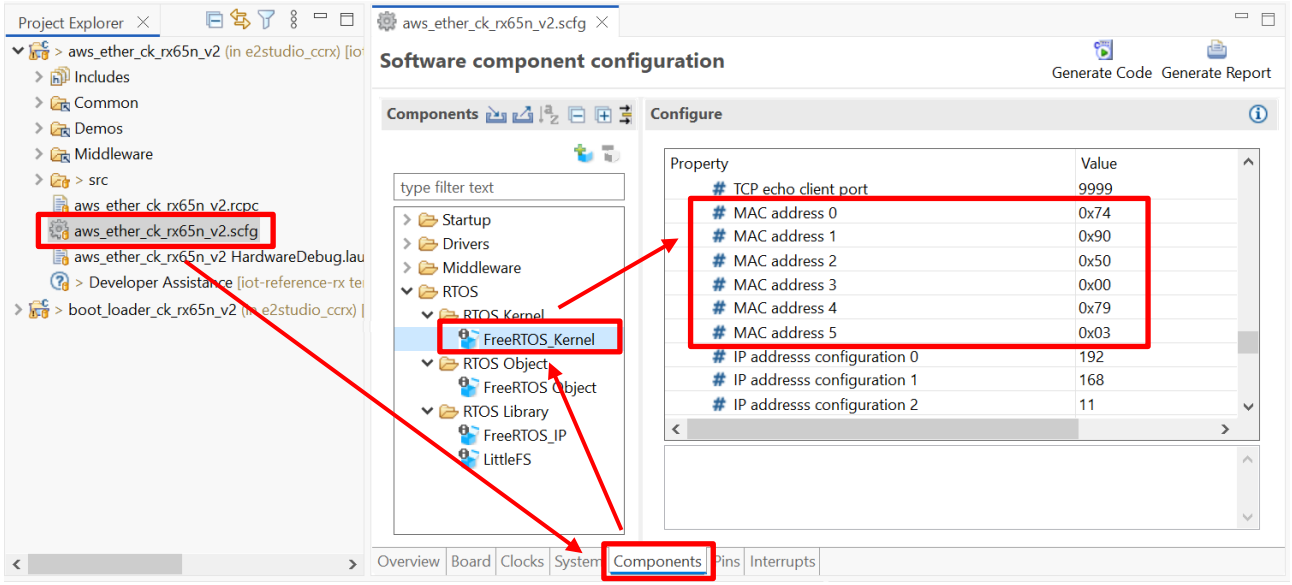


Figure 7-1 Setting MAC Addresses

After making the above changes, click on the [Generate Code] button in the upper-right region of the window to have the code reflect the changes made in the Smart Configurator.

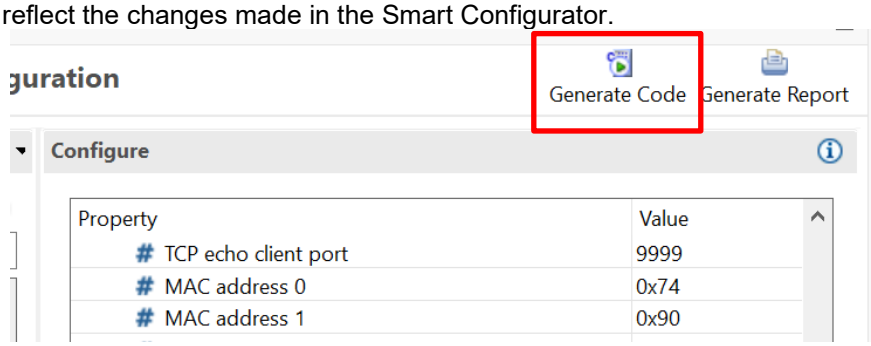


Figure 7-2 Generating Code

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Mar. 10, 2026	—	First edition issued.

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins

   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

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

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

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

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

(Rev.5.0-1 October 2020)

Corporate Headquarters

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

Trademarks

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

Contact information

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