RX65N グループ

RX65N Cloud kit で Azure RTOS を用いて センサデータを可視化する方法

要旨

本ドキュメントでは、ルネサス製ボード RX65N Cloud Kit とキット付属の Wi-Fi モジュール（SX-ULPGN(Silex Technology 製)）を使用し、マイクロソフト社の Azure RTOS にて Azure クラウドサービス（Azure IoT Hub）と通信するサンプルコードを提供します。また、Wi-Fi 経由で Azure IoT Hub にアップロードした温度データを、web アプリを使って可視化する方法を説明します。

Azure RTOS は、接続、セキュリティ、および無線（OTA）アップデートなどを行うリアルタイムオペレーションシステムです。マイクロソフト社より提供されている Azure RTOS には、Azure RTOS 機能のデモを行うデモアプリケーションも含まれています。このデモアプリケーションは RX65N Cloud Kit 上で動作します。また、RX ファミリはマイクロソフト社より、Azure RTOS の認定ハードウェアとされています。このため、RX ファミリの MCU 上で Azure RTOS を使用する場合は無償でご利用頂けます。詳細はコチラ <https://github.com/azure-rtos/threadx/blob/master/LICENSED-HARDWARE.txt> をご参照ください。

e^2 studio は、オープンソースの Eclipse CDT(C/C++ Development Tooling)をベースとした開発環境で、デバッグのインターフェイスに加え、ビルド（エディタ、コンパイラ、リンカ制御）をサポートしています。

動作確認デバイス

RX65N グループ（RX65N Cloud Kit）

・RXクラウドソリューションの開発に必要なボード、関連プログラム、開発環境に関する情報は以下のリンクにまとめられています。
https://www.renesas.com/rx-cloud

・YouTube 動画
本アプリケーションノートに記載されている内容を動画にて説明しています。

Azure RTOS チュートリアル(1/3) *RX65N* Cloud Kit : ~開発環境セットアップ編～ - YouTube

Azure RTOS チュートリアル(2/3) RX65N Cloud Kit : ~プログラムセットアップ編～ - YouTube

Azure RTOS チュートリアル(3/3) RX65N Cloud Kit : ~Azure クラウド操作編～ - YouTube

・Azure RTOS GitHub Sample Code
Azure RTOS 機能サンプルコード
https://github.com/azure-rtos/samples

Azure RTOS Plug and Play サンプルコード
https://github.com/azure-rtos/samples/tree/PublicPreview/PnP

Azure RTOS ADU サンプルコード
https://github.com/azure-rtos/samples/tree/PublicPreview/ADU

目次

1. システム概要

1.1 システム概要図

以下に、RX65N Cloud Kit のセンサデータを取得してから可視化するまでのシステム図および、Azure IoT Explorer を使用して RX65N Cloud Kit を制御するシステム図を示します。

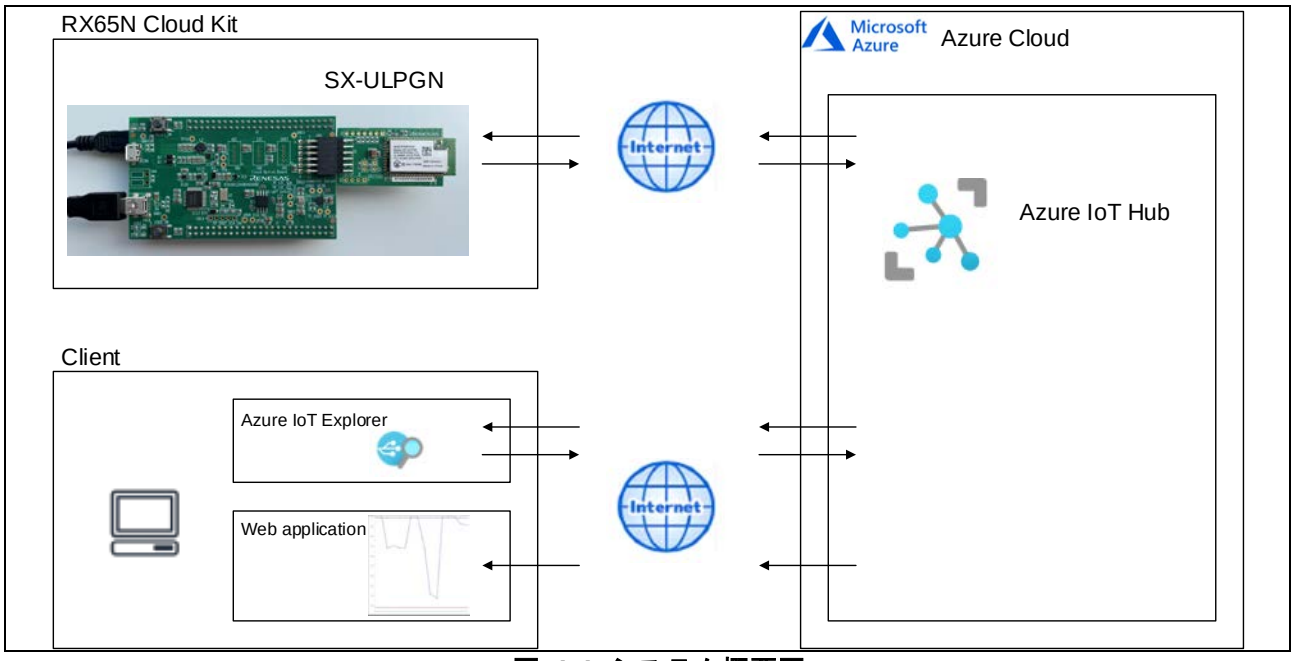

図 1-1 システム概要図

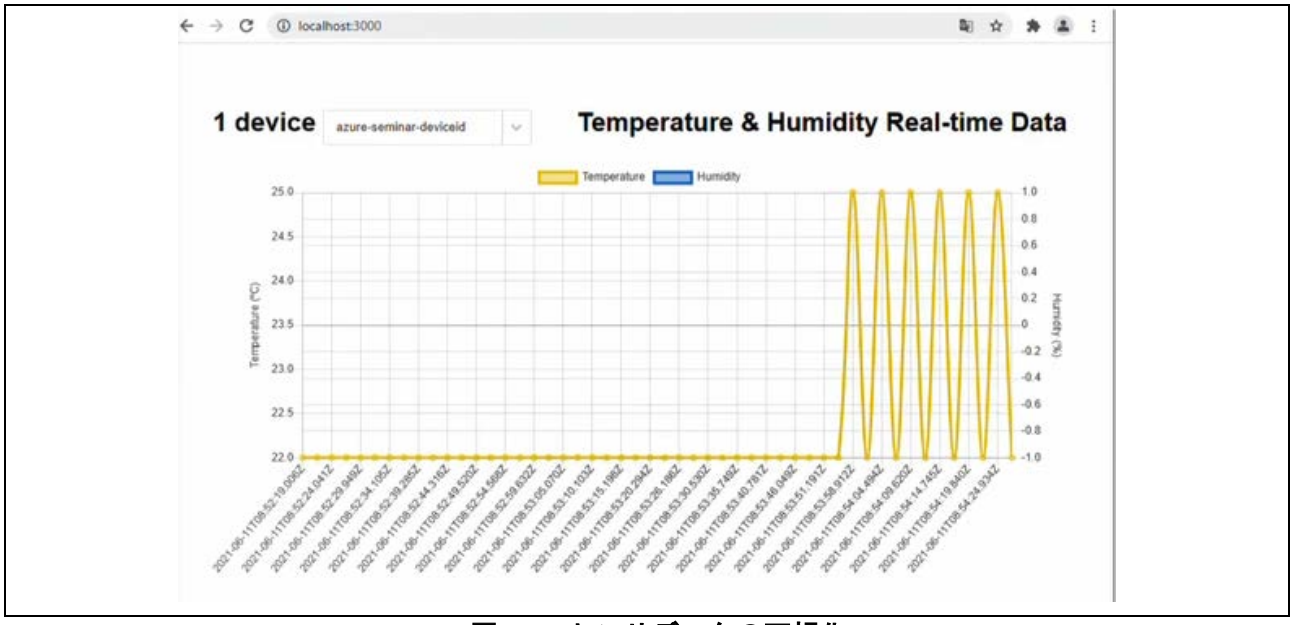

図 1-2 センサデータの可視化

2. 準備

2.1 ハードウェア構成

本システムのハードウェア構成を下表に示します。

表 2-1 ハードウェア構成

Item	Content	Provider	Description
使用ボード (RX65N Cloud Kit 同梱物)	Target Board for RX65N	Renesas Electronics Corporation	RX65N MCU 搭載の評価ボード注
	RX Cloud Option Board		Azure 接続可能なクラウド通信評価ボード注
	Silex Pmod Module		無線 LAN モジュール搭載の通信ボード注
Wi-Fi	無線ルーター	-	無線 LAN 規格：IEEE 802.11b/g/n (2.4GHz) 暗号化方式：AES
PC	Windows 10	-	推奨 OS
	Google Chrome	-	使用するブラウザ
注 Target Board for RX65N と RX65N Cloud Kit Option Board、Silex Pmod Module は、RX65N Cloud Kit に同梱されています。			

RX65N Cloud Kit Web:

https://www.renesas.com/products/microcontrollers-microprocessors/rx-32-bit-performance-efficiency-mcus/rx65n-cloud-kit-renesas-rx65n-cloud-kit

2.2 Azure および Azure RTOS について

Azure とは、Microsoft が提供するクラウドコンピューティングサービスのことです。

Microsoft が提供する、マイコン向けのリアルタイムオペレーティングシステムである Azure RTOS にはマイコンと Azure を接続するためのライブラリ群が用意されており、クラウド接続した IoT 機器の管理や制御が可能です。

2.3 ソフトウェア構成

本システムのソフトウェア構成を下表に示します。

表 2-2 ソフトウェア構成

Item	Content	Version
統合開発環境	e^2 studio	2021-04
コンパイラ	GCC for Renesas RX	8.3.0.202004
通信ソフト	Tera Term	Version4.105
デバイスとの対話ツール	Azure IoT Explorer	Version 0.14.3
web アプリダウンロード	Git	2.31.1
web アプリ実行	node.js (npm)	14.17.0
エミュレータ	E2 Lite (オンボード)	-

本システムで使用するソフトウェアのダウンロードサイトを下表に示します。

注 リンク先は変更になる場合もありますのでご注意ください。

表 2-3 ツールダウンロードサイト

Content	Link
e^2 studio	https://www.renesas.com/kr/software-tool/e-studio
GCC for Renesas RX	https://llvm-gcc-renesas.com/rx-download-toolchains
Tera Term	https://ja.osdn.net/projects/ttssh2
Azure IoT Explorer	https://github.com/Azure/azure-iot-explorer/releases
Git	https://www.git-scm.com/download
node.js (npm)	http://nodejs.org

3. Azure との接続

RX65N Cloud Kit と Azure を接続させるために必要な準備をします。

3.1 Azure 準備

3.1.1 Azure サインイン

1. Azure にサインインする。

・Azure サインインページ

https://azure.microsoft.com

Azure アカウントを持っていない場合は、画面上の「無料アカウント」から作成してください。

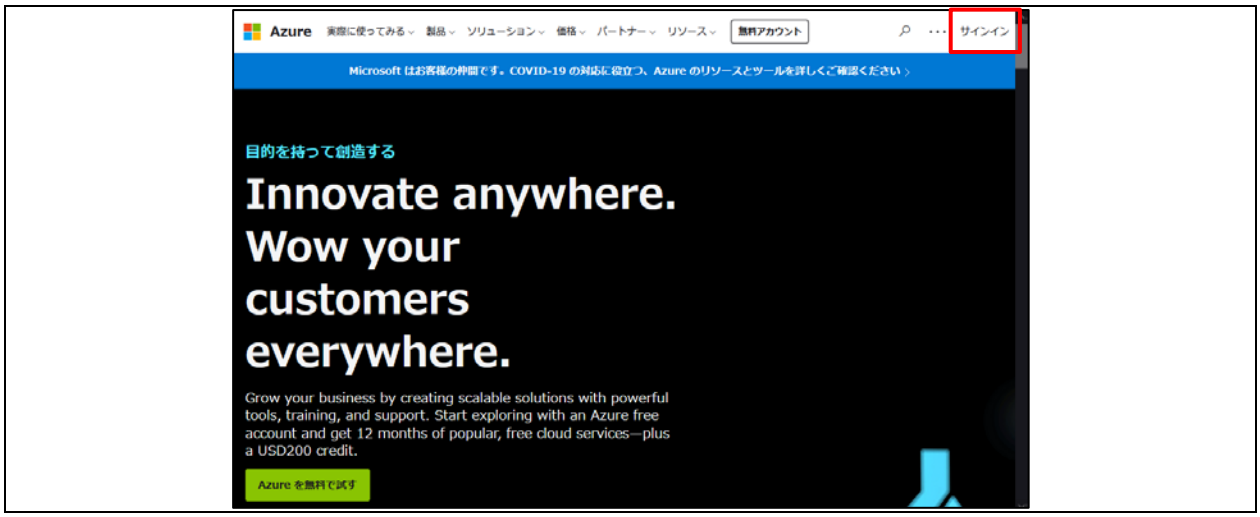

図 3-1 サインイン画面

2. サインイン後、Azure Portal ページに移動する。

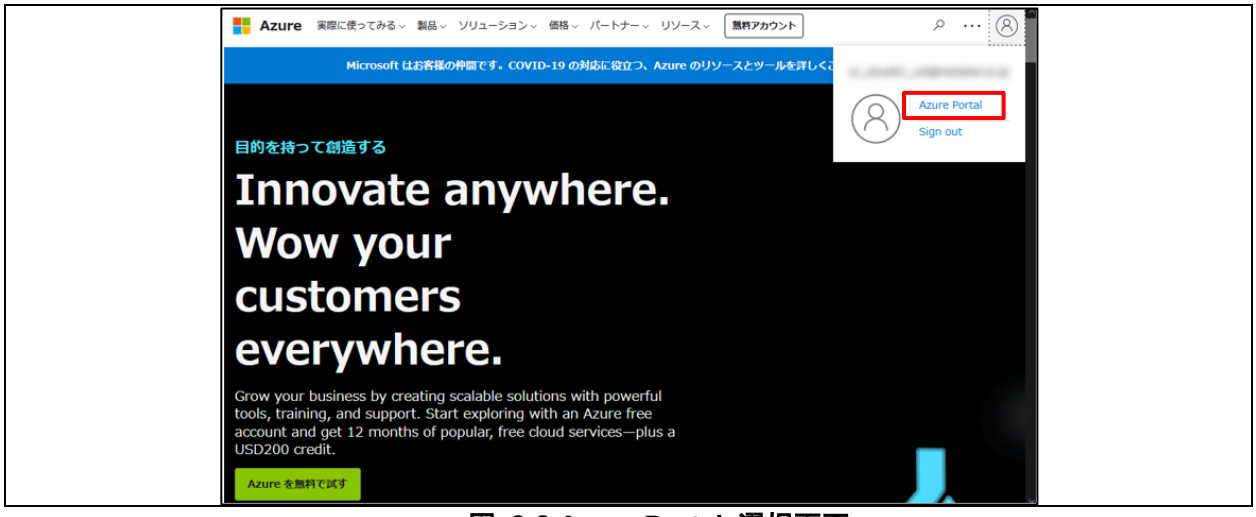

図 3-2 Azure Portal 選択画面

3.1.2 IoT Hub 作成

1. リソースの作成をクリック。

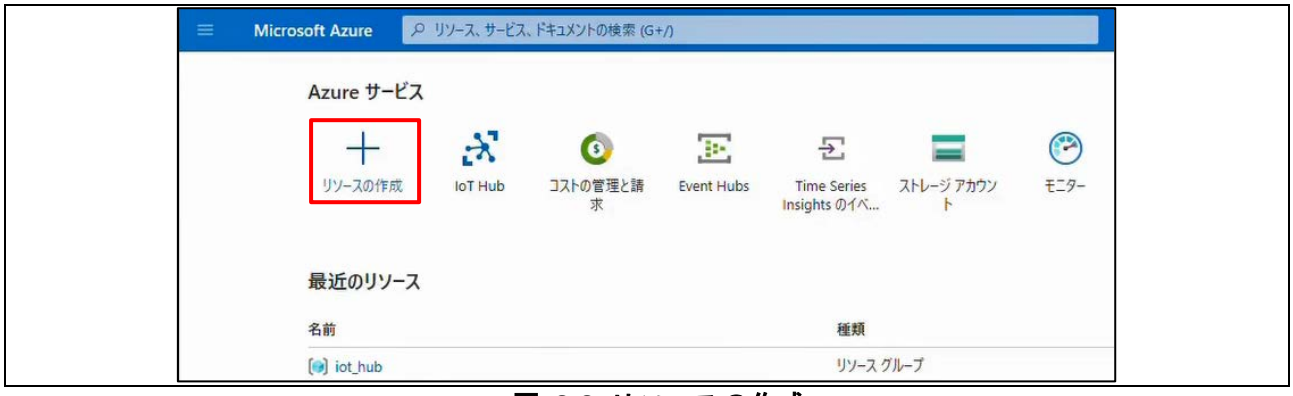

図 3-3 リソースの作成

2. 「IoT Hub」と入力し、 表示された候補から IoT Hub をクリック。

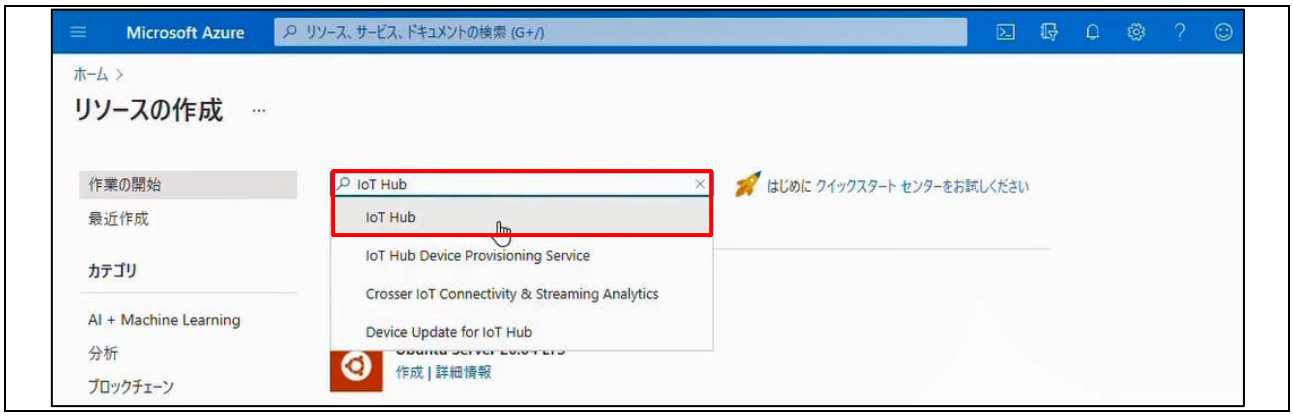

図 3-4 IoT Hub 検索 -> IoT Hub 選択

3. 作成をクリック。

図 3-5 IoT Hub 作成

4. 基本 タブで、使用環境に合わせてサブスクリプションを選択、リソースグループ、IoT Hub 名[注]、領域を入力 -> 次へ: ネットワークをクリック。

注 IoT Hub 名は Azure クラウド全体で一意である必要があります。

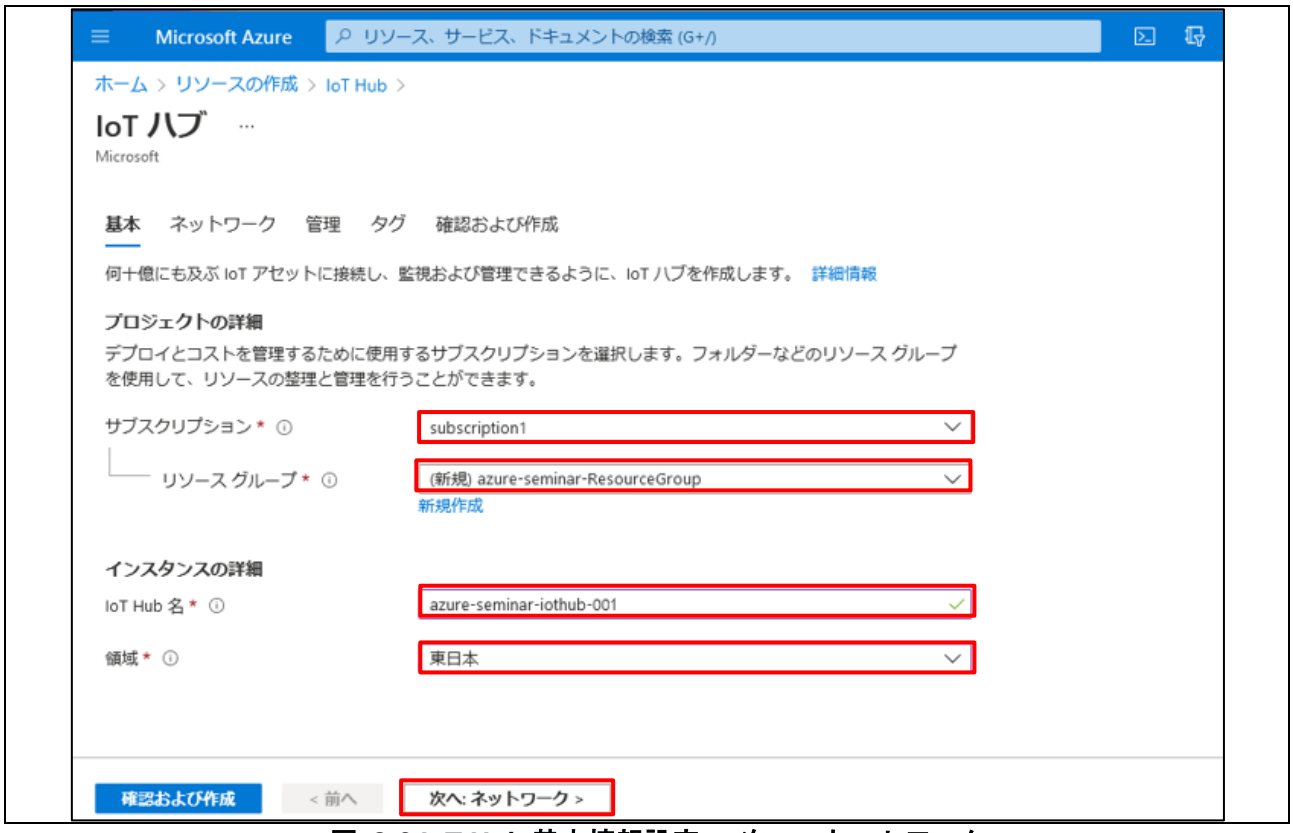

図 3-6 IoT Hub 基本情報設定-> 次へ: ネットワーク

5. ネットワーク タブで、パブリックエンドポイントを選択 -> 次へ: 管理をクリック。

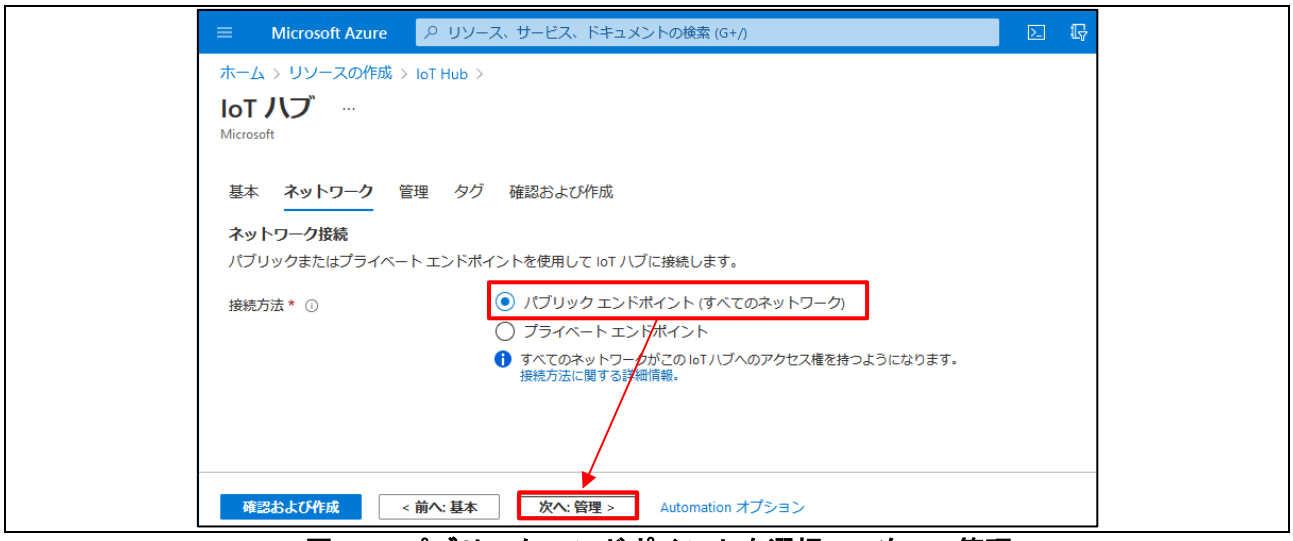

図 3-7 パブリックエンドポイントを選択 -> 次へ: 管理

6. 管理 タブで、使用環境に合わせて 価格とスケールティアを選択 -> 次へ: タグ をクリック。

価格とスケールティアで F1:Free レベル以外を選択する場合、それ以降の設定項目も必要に応じて設定してください。

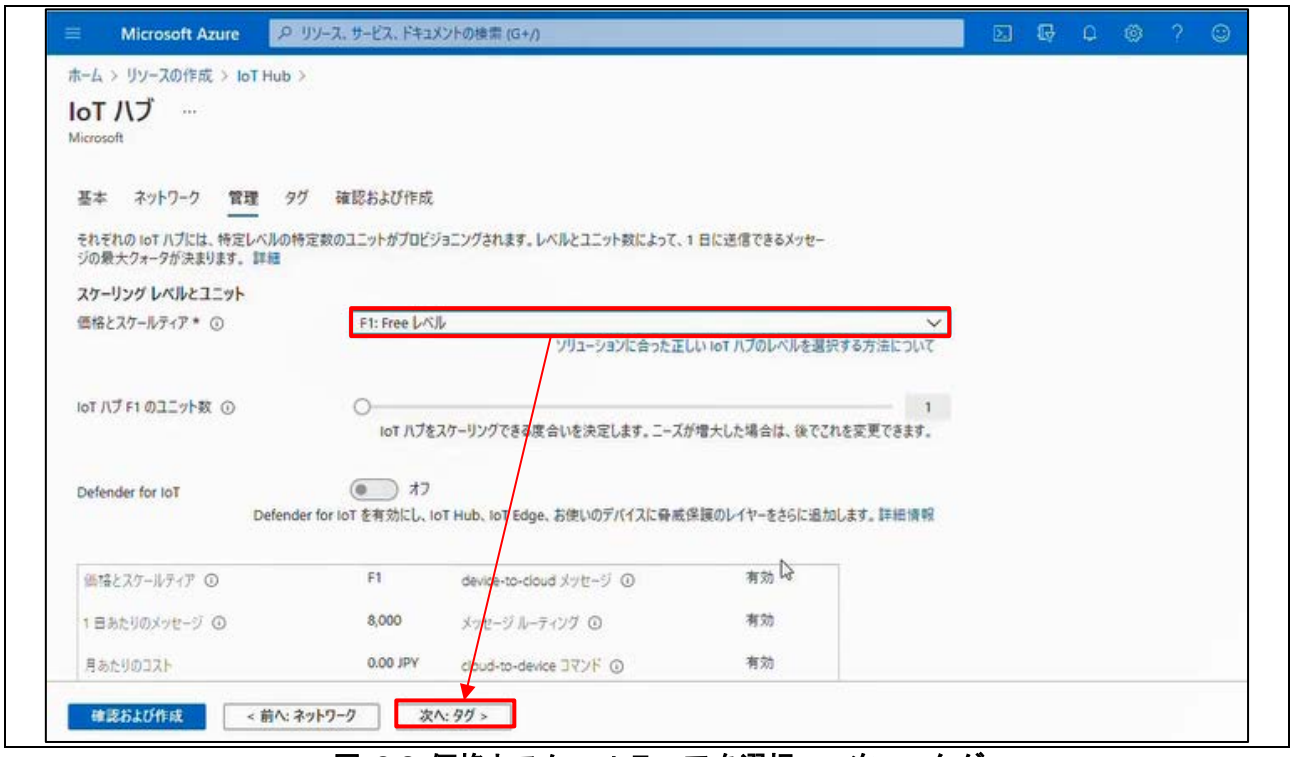

図 3-8 価格とスケールティアを選択 -> 次へ: タグ

7. タグ タブで、必要に応じてタグを設定 -> 次へ: 確認および作成をクリック。

本ドキュメントでは入力不要であるため、何も入力せず 次へ: 確認および作成 をクリックします。

図 3-9 タグを設定 -> 次へ: 確認および作成

8. 確認および作成 タブで、選択内容を確認 -> 問題がなければ 作成をクリック。

IoT Hub の作成には数分かかります。

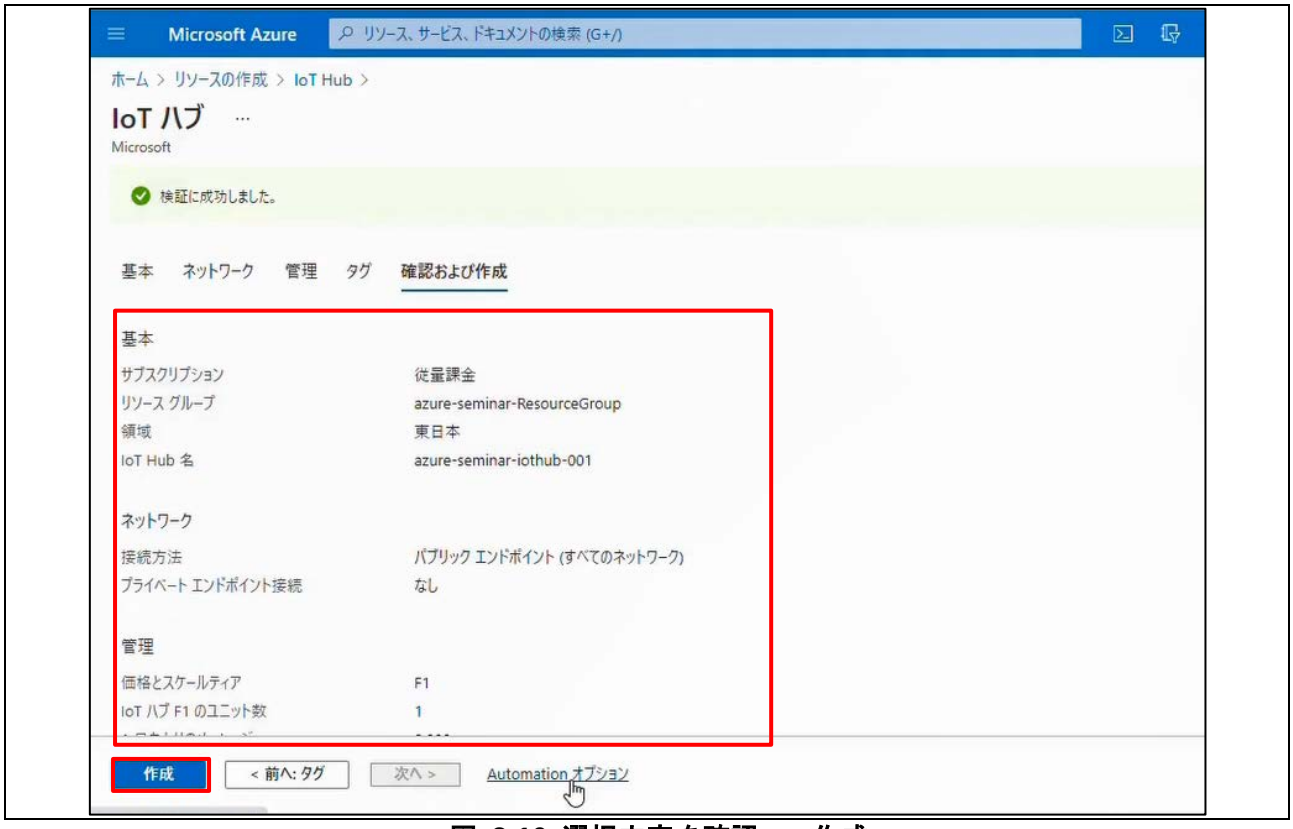

図 3-10 選択内容を確認 -> 作成

9. IoT Hub が作成されたら、リソースに移動をクリック。

図 3-11 リソースに移動

10. ホスト名をテキストエディタ等にメモ。

ホスト名の情報は後で使用します。

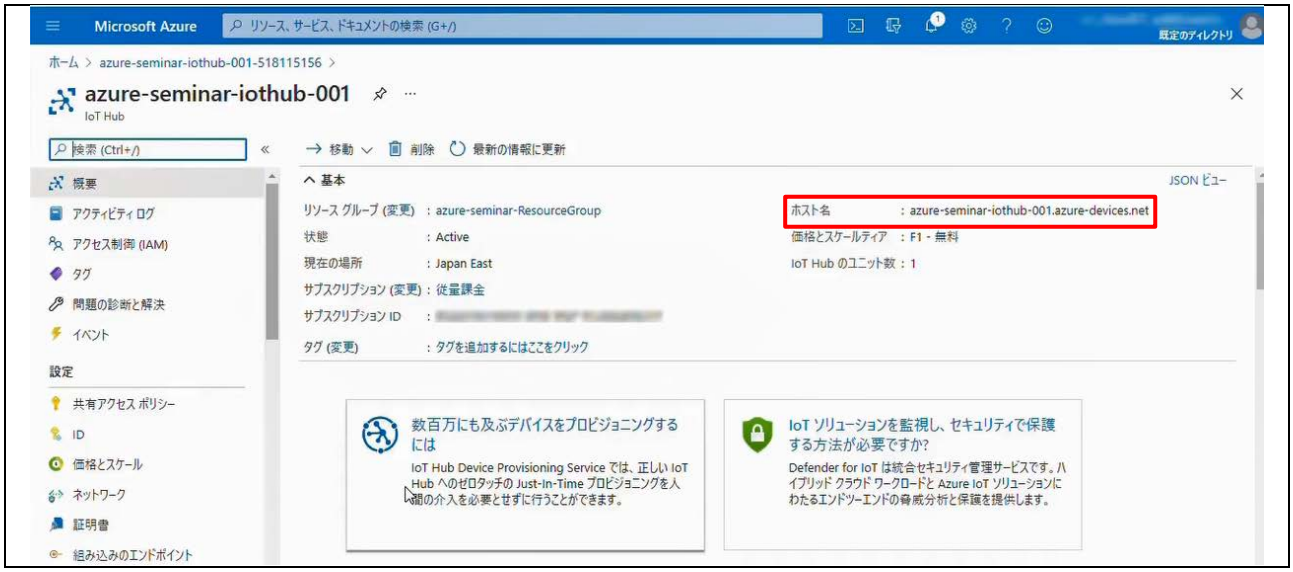

図 3-12 IoT Hub ホスト名 記録

3.1.3 IoT デバイス作成

1. 画面左側の IoT デバイス -> デバイスの追加をクリック。

図 3-13 IoT デバイス -> デバイスの追加

2. デバイスの作成 で、以下の情報を入力および選択 -> 保存をクリック。

・デバイス ID -> 任意の名前を入力
・認証の種類 -> 対称キー
・自動生成キー -> チェックを入れる
・このデバイスを IoT ハブに接続する -> 有効化

図 3-14 デバイス ID 情報設定-> 保存

3. デバイス ID 名をテキストエディタ等にメモし、デバイス ID をクリック。

デバイス ID 名の情報は後で使用します。

画面が切り替わらない場合は、「最新の情報に更新」をクリックして更新します。

図 3-15 デバイス ID クリック

4. 主キーの値をテキストエディタ等にメモ。

主キーの情報は後で使用します。

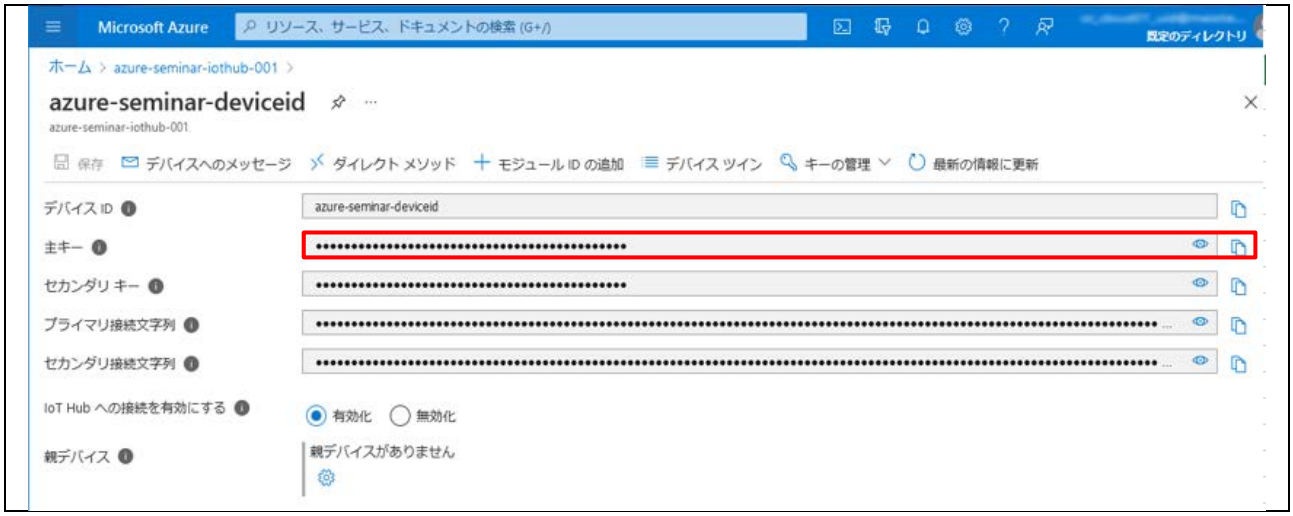

図 3-16 デバイス ID 主キー記録

3.2 ソフトウェア準備

デモプログラムを実行するためのソフトウェアの準備をします。

1. サンプルプロジェクトをダウンロードする。

GitHub の Azure RTOS サンプルページから RX65N Cloud Kit と GCC コンパイラの組み合わせの Azure RTOS プロジェクトをダウンロードします。

・GitHub サンプルページ

https://github.com/azure-rtos/samples

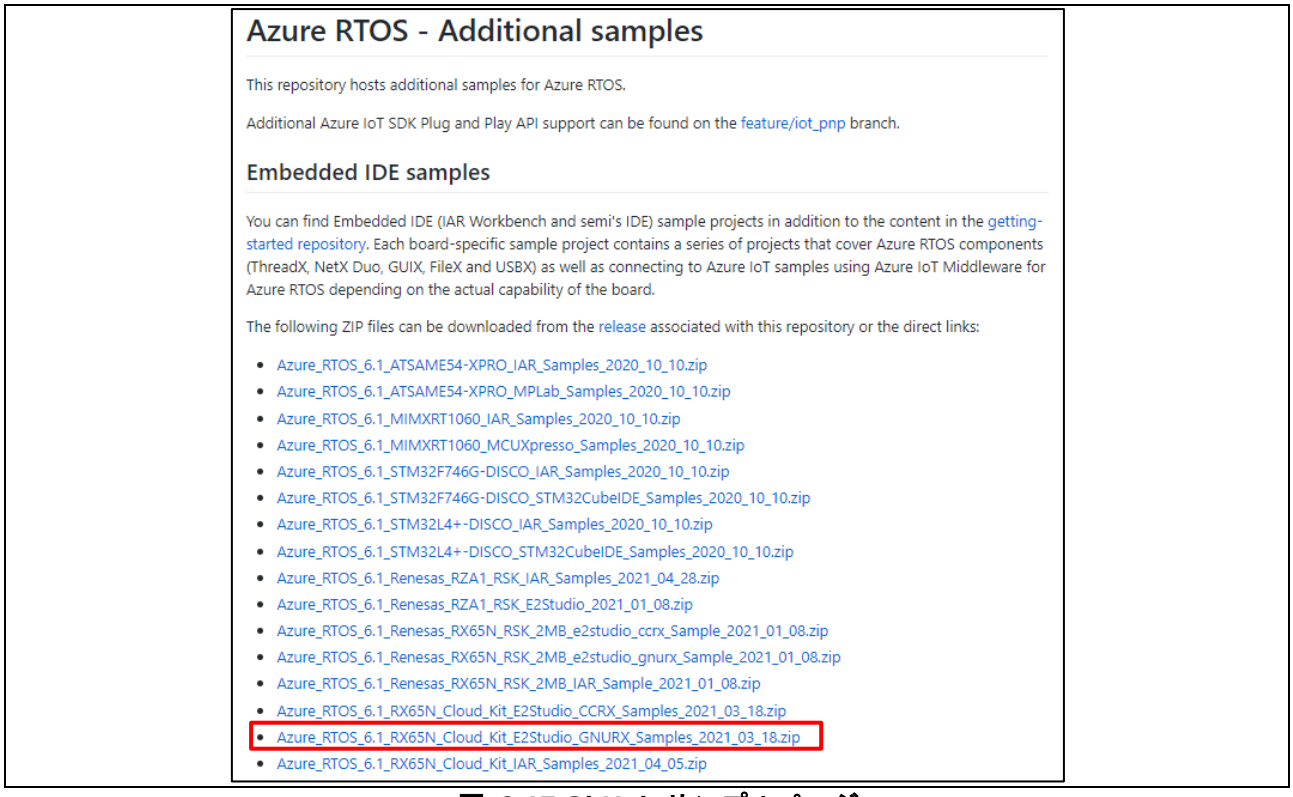

図 3-17 GitHub サンプルページ

2. プロジェクトを解凍し、プロジェクトを配置[注]する。（以下、プロジェクトを配置したルートフォルダを${base_folder}と表記します。）

注 解凍したプロジェクトは¥C ドライブ直下など、フォルダ階層が浅い階層に配置してください。フォルダパスが長い場合、ビルドエラーになる場合があります。

3. e² studio を起動して、ワークスペースを指定する。

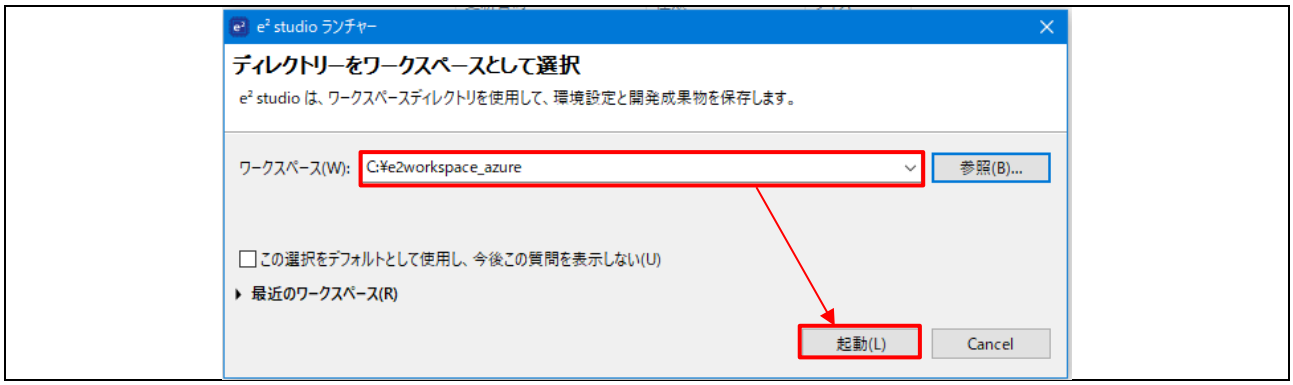

図 3-18 ワークスペース選択画面

4. ファイル -> インポート をクリック。

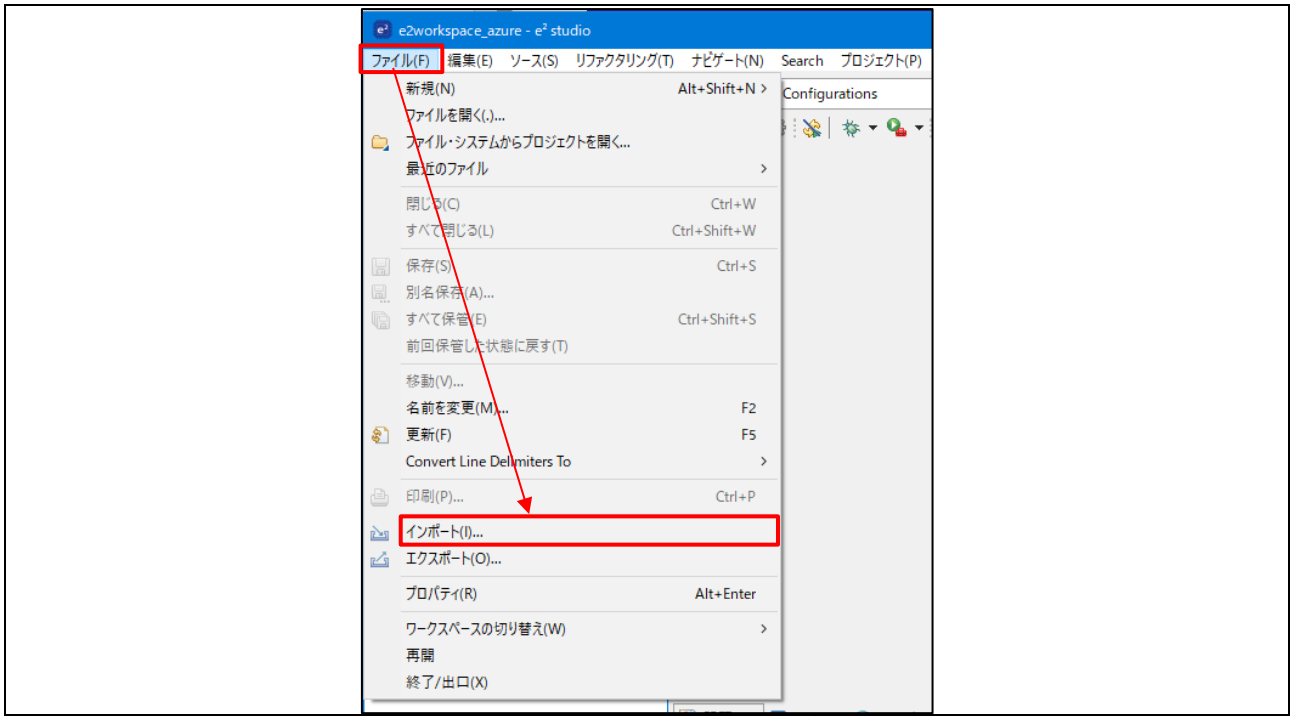

図 3-19 ファイル -> インポート

5. 一般 -> 既存プロジェクトをワークスペースへ -> 次へ をクリック。

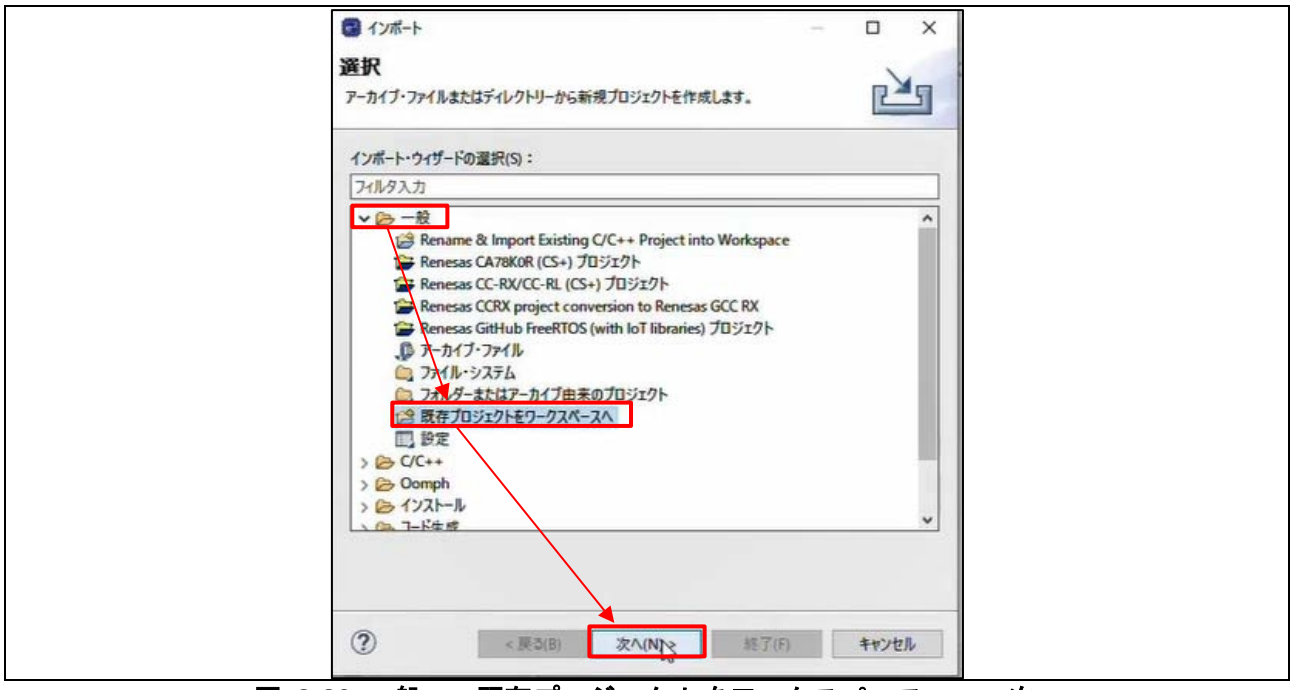

図 3-20 一般 -> 既存プロジェクトをワークスペースへ -> 次へ

6.「参照」で、${base_folder}/rx65n-cloud-kit フォルダを指定 -> 抽出されたすべてのサンプルプロジェクトを選択 -> 終了

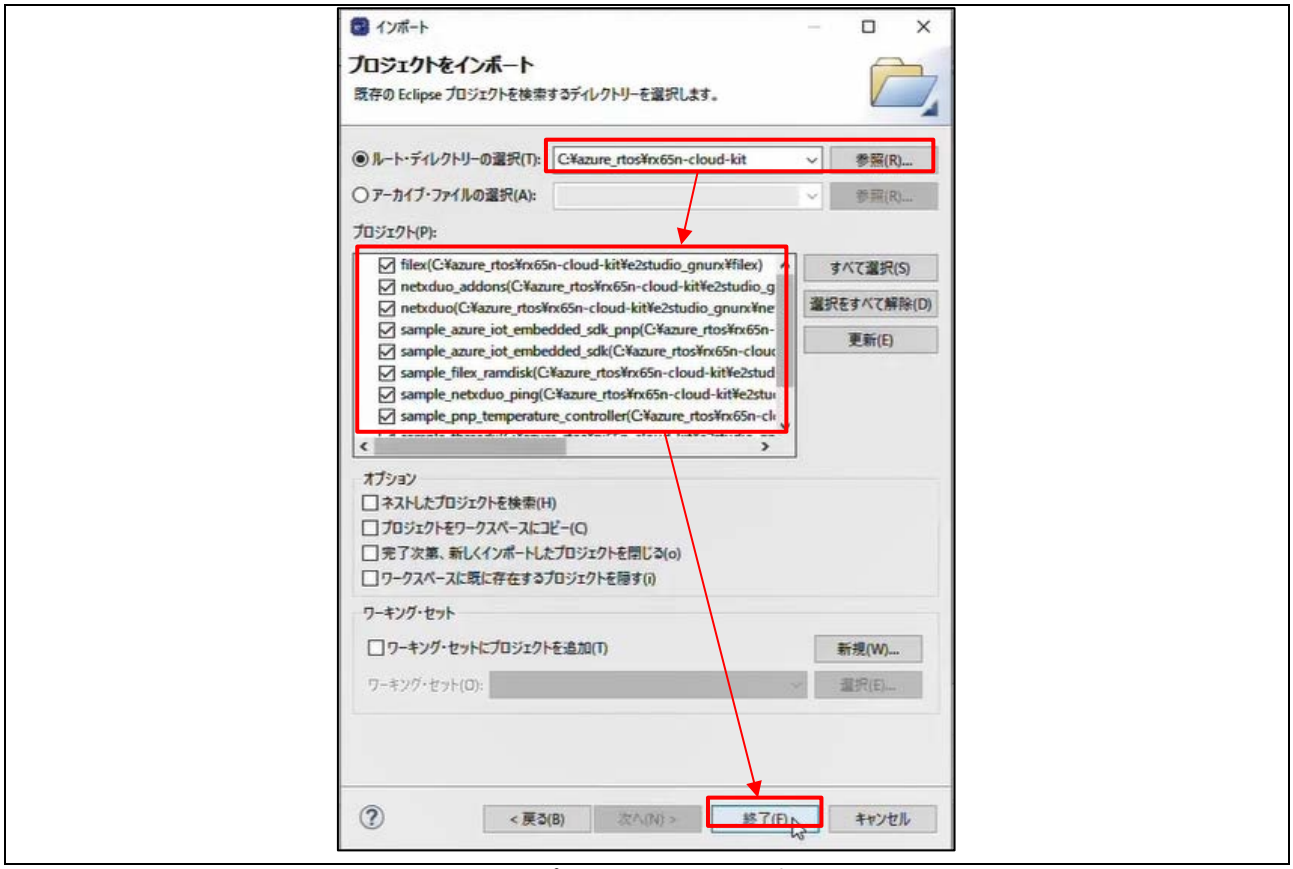

図 3-21 プロジェクトインポート画面

7. {$base_folder}/rx65n-cloud-kit/e2studio_gnurx/sample_pnp_temperature_controller/src/main.c にある 2 つのマクロを設定する。

・WIFI_SSID -> 接続する Wi-Fi アクセスポイントの SSID

・WIFI_PASSWORD -> 接続する Wi-Fi アクセスポイントのパスワード

（上記のマクロは、下図のように、""の中に入力してください。）

```
/*       Copyright (c) Microsoft Corporation. All rights reserved.        */

#include "nx_api.h"
#include "nxd_dns.h"
#include "nx_secure_tls_api.h"
#include <nx_wifi.h>

#include <demo_printf.h>

#include <r_wifi_sx_ulpgn_if.h>

/* Include the sample.  */
extern VOID sample_entry(NX_IP* ip_ptr, NX_PACKET_POOL* pool_ptr, NX_DNS* dns_ptr, UINT (*unix_time_callback)(ULONG *unix_time));

/* Define the Wi-Fi network parameters. Only WPA2 security is supported.  */
#ifndef WIFI_SSID
#define WIFI_SSID                       "SeminarAP"
#endif /* WIFI_SSID  */

#ifndef WIFI_PASSWORD
#define WIFI_PASSWORD                   "SeminarPassword"
#endif /* WIFI_PASSWORD  */
```

図 3-22 main.c

8. {$base_folder}/rx65n-cloud-kit/e2studio_gnurx/sample_pnp_temperature_controller/src/sample_config.h にある 3 つのマクロを設定する。

・HOST_NAME -> 「3.1 Azure 準備」で確認したホスト名

・DEVICE_ID -> 「3.1 Azure 準備」で作成したデバイス ID

・DEVICE_SYMMETRIC_KEY -> 「3.1 Azure 準備」で確認した主キー

（上記のマクロは、下図のように、""の中に入力してください。）

```
/* Required when DPS is not used.  */
/* These values can be picked from device connection string which is of format : HostName=<host1>;DeviceId=<device1>;SharedAccessKey=<k
   HOST_NAME can be set to <host1>,
   DEVICE_ID can be set to <device1>,
   DEVICE_SYMMETRIC_KEY can be set to <key1>.  */
#ifndef HOST_NAME
#define HOST_NAME                                   "azure-seminar-iothub-001.azure-devices.net"
#endif /* HOST_NAME */

#ifndef DEVICE_ID
#define DEVICE_ID                                   "azure-seminar-deviceid"
#endif /* DEVICE_ID */

#else /* !ENABLE_DPS_SAMPLE */

/* Required when DPS is used.  */
#ifndef ENDPOINT
#define ENDPOINT                                    ""
#endif /* ENDPOINT */

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

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

#endif /* ENABLE_DPS_SAMPLE */

/* Optional SYMMETRIC KEY.  */
#ifndef DEVICE_SYMMETRIC_KEY
#define DEVICE_SYMMETRIC_KEY
#endif /* DEVICE_SYMMETRIC_KEY */
```

図 3-23 sample_config.h

9.変更を保存し、プロジェクト -> 「クリーン」 をクリック。

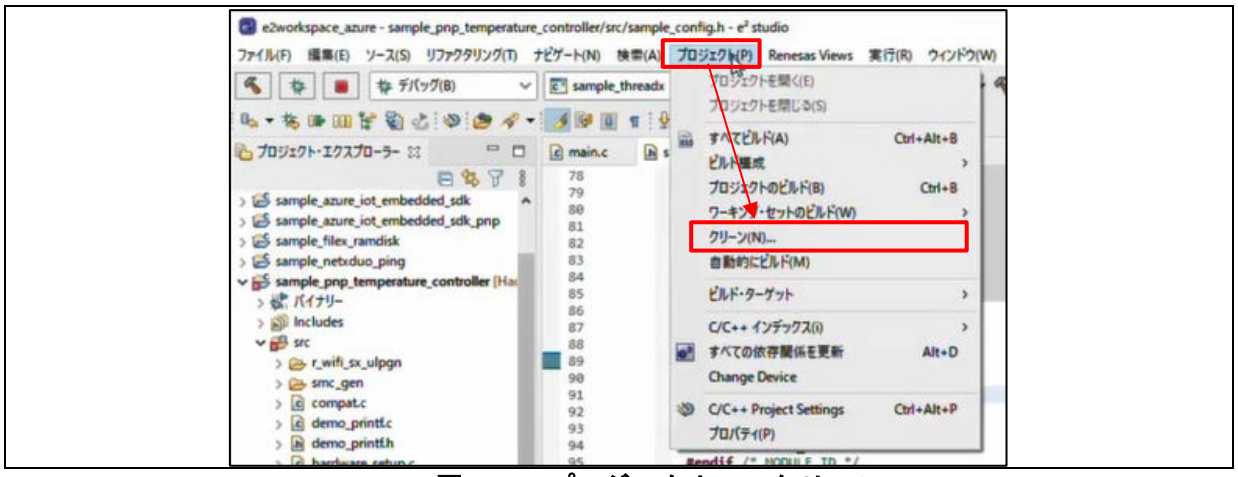

図 3-24 プロジェクト -> クリーン

10.「全てのプロジェクトをクリーン」のチェックを外し、sample_pnp_temperature_controller にのみチェックを入れて「クリーン」 をクリックし、0error であることを確認する。

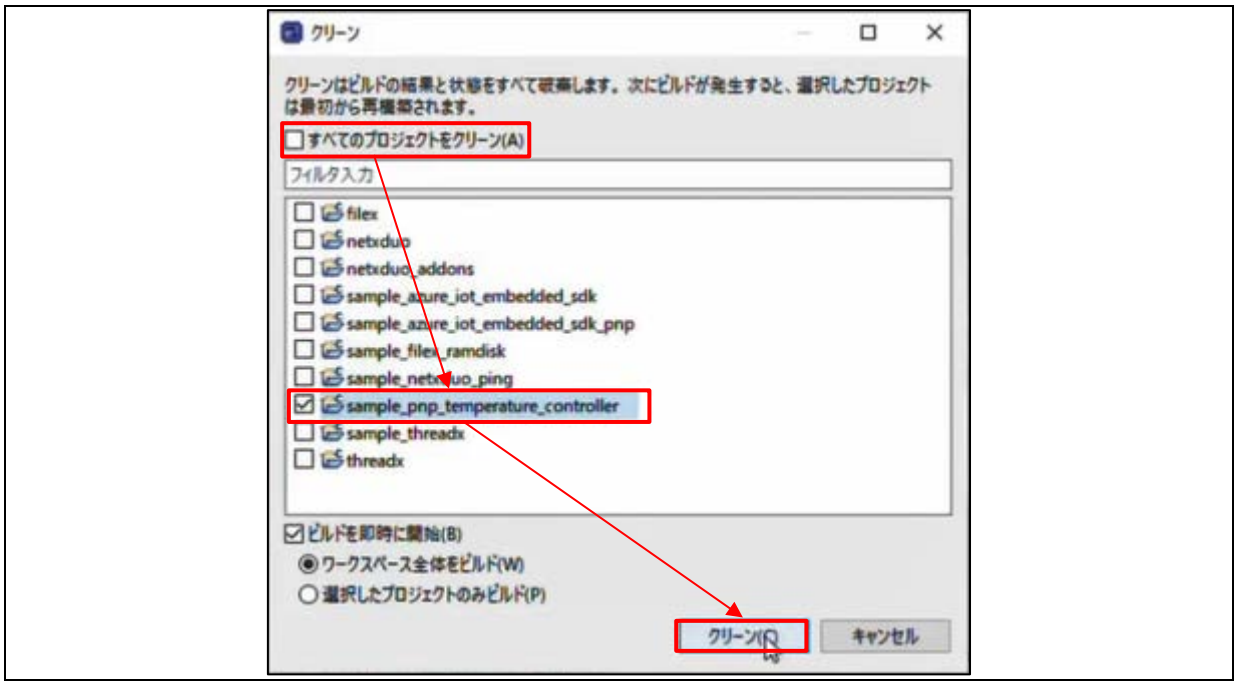

図 3-25 クリーン設定画面

```
rx-elf-size --format=berkeley "sample_pnp_temperature_controller.elf"
   text    data     bss     dec     hex filename
 505648    7048  186288  698984   aaa68 sample_pnp_temperature_controller.elf

15:52:09 Build Finished. 0 errors, 22 warnings. (took 15s.654ms)
```

図 3-26 クリーン完了画面

3.3 実行準備

デモプログラムを実行するための準備をします。

3.3.1 ハードウェア準備

1. Target Board(下段ボード) の EJ2 ジャンパピンを外す。
2. Target Board(下段ボード) の ECN1 コネクタと PC を USB ケーブルで接続する。
3. Cloud Option Board(上段ボード) の CN18 コネクタと PC を USB ケーブルで接続する。

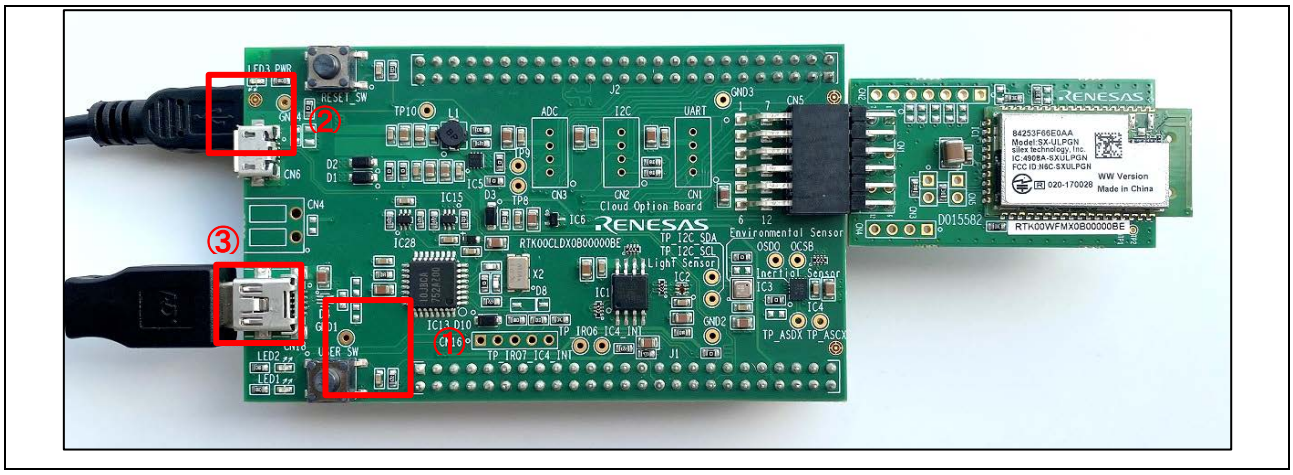

図 3-27 RX65N Cloud Kit（上段）

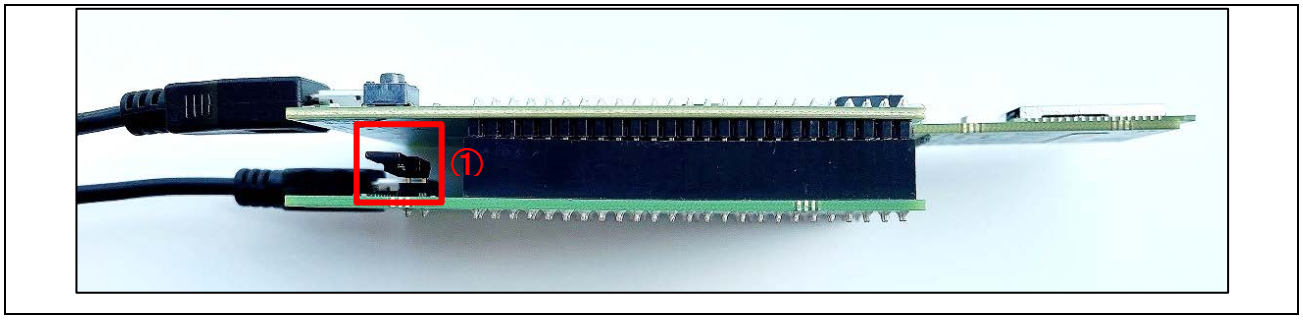

図 3-28 RX65N Cloud Kit（下段）

3.3.2 Tera Term 準備

Tera Term を起動し、下表の通り設定します。

表 3-1 Tera Term 設定

Item	Setting
ボー・レート	115200
データ長	8bit
パリティ	none
ストップビット	1bit
フロー制御	none

3.4 デモプログラム実行

デモプログラムを実行する手順を以下に示します。

1. e² studio 画面左上のドロップダウンリストから「sample_pnp_temperature_controller HardwareDebug」をクリック。

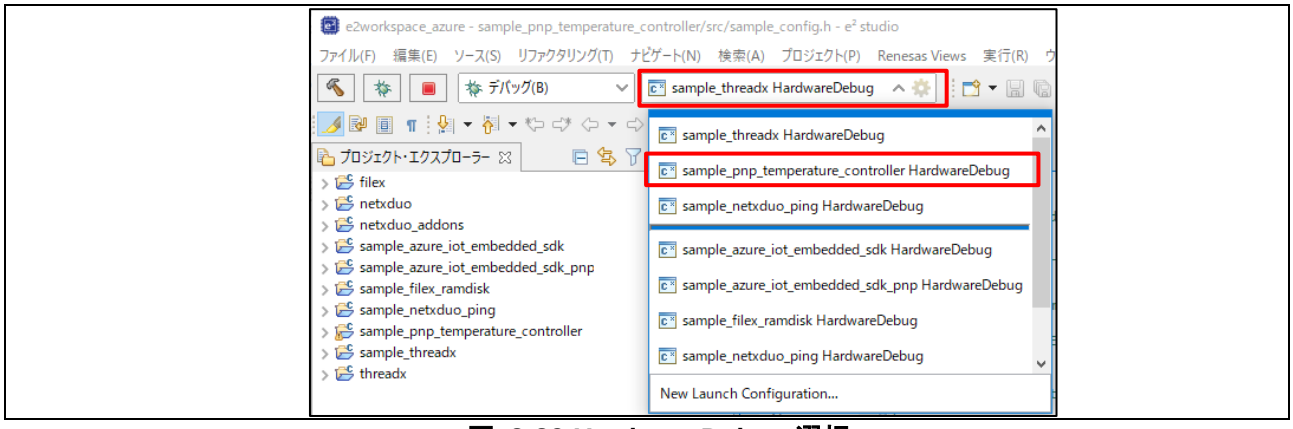

図 3-29 HardwareDebug 選択

2. 虫アイコンをクリック。

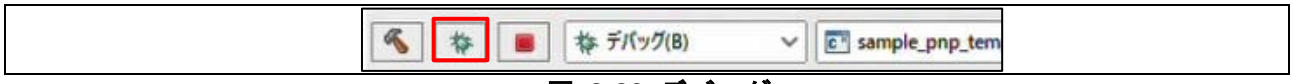

図 3-30 デバッグ

3. パースペクティブ切り替えの確認が出現したら、「切り替え」をクリック。

4. restart ボタン をクリックする。しばらくして main 関数に止まるので再開ボタンをクリック。

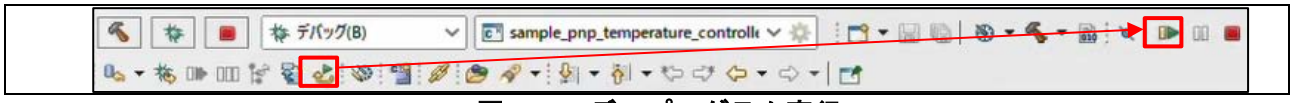

図 3-31 デモプログラム実行

5. Tera Term 画面に実行ログが出力されるのを確認する。

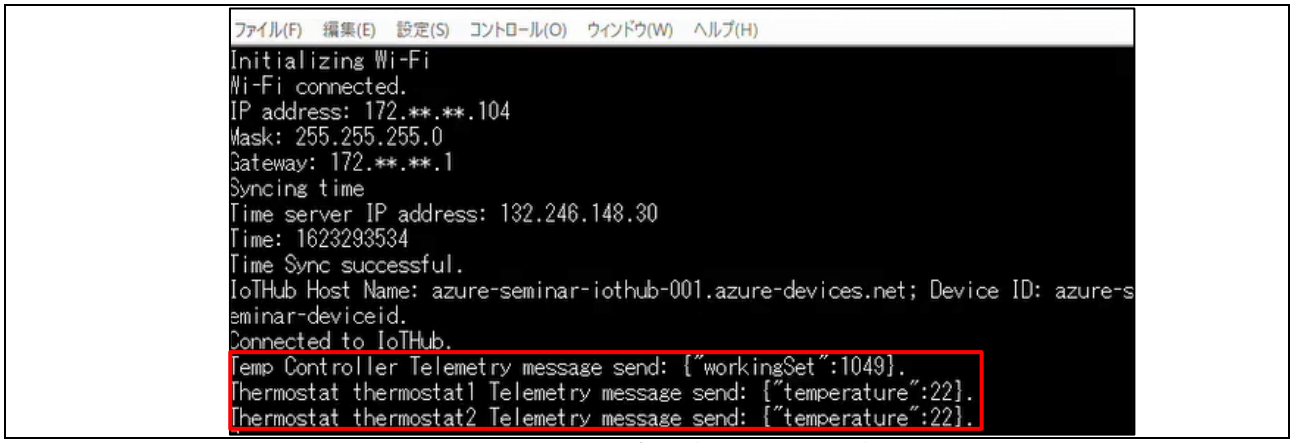

図 3-32 デモプログラム実行ログ

3.5 Azure IoT Explorer による通信確認

アップロードしたデータが Azure クラウドに送信されていることを確認します。

1. Azure Portal の作成した IoT Hub ページにアクセスし、共有アクセスポリシー -> iothubowner -> プライマリ接続文字列 を取得。

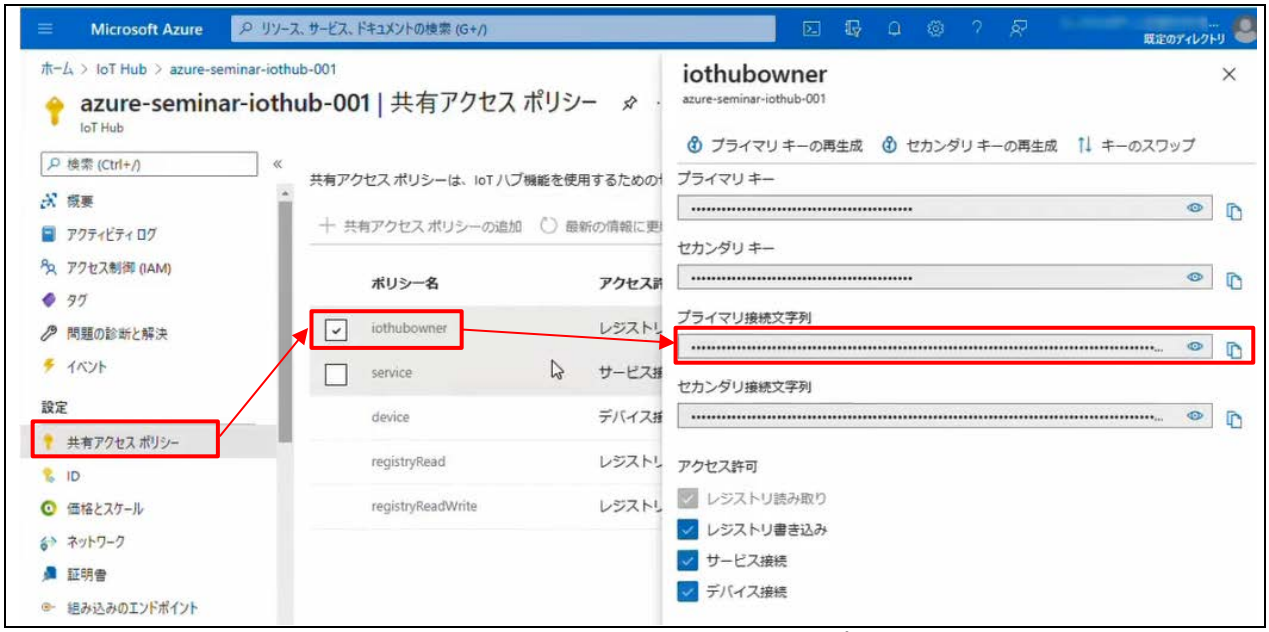

図 3-33 共有アクセスポリシー -> iothubowner -> プライマリ接続文字列

2. Azure IoT Explorer を起動し、Add connection をクリック。

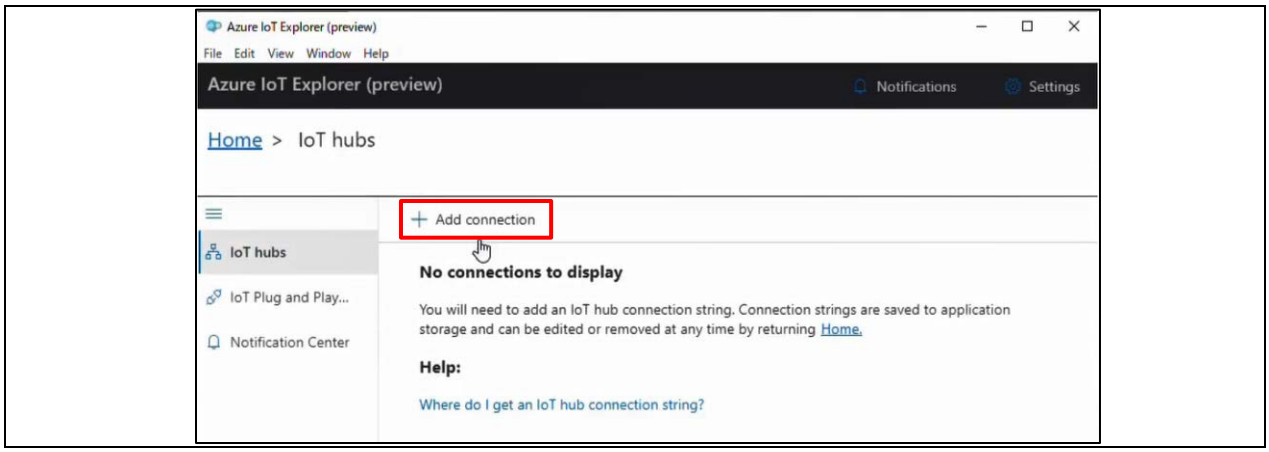

図 3-34 Azure IoT Explorer

3. 「Connection string」に取得したプライマリ接続文字列を入力 -> Save をクリック。

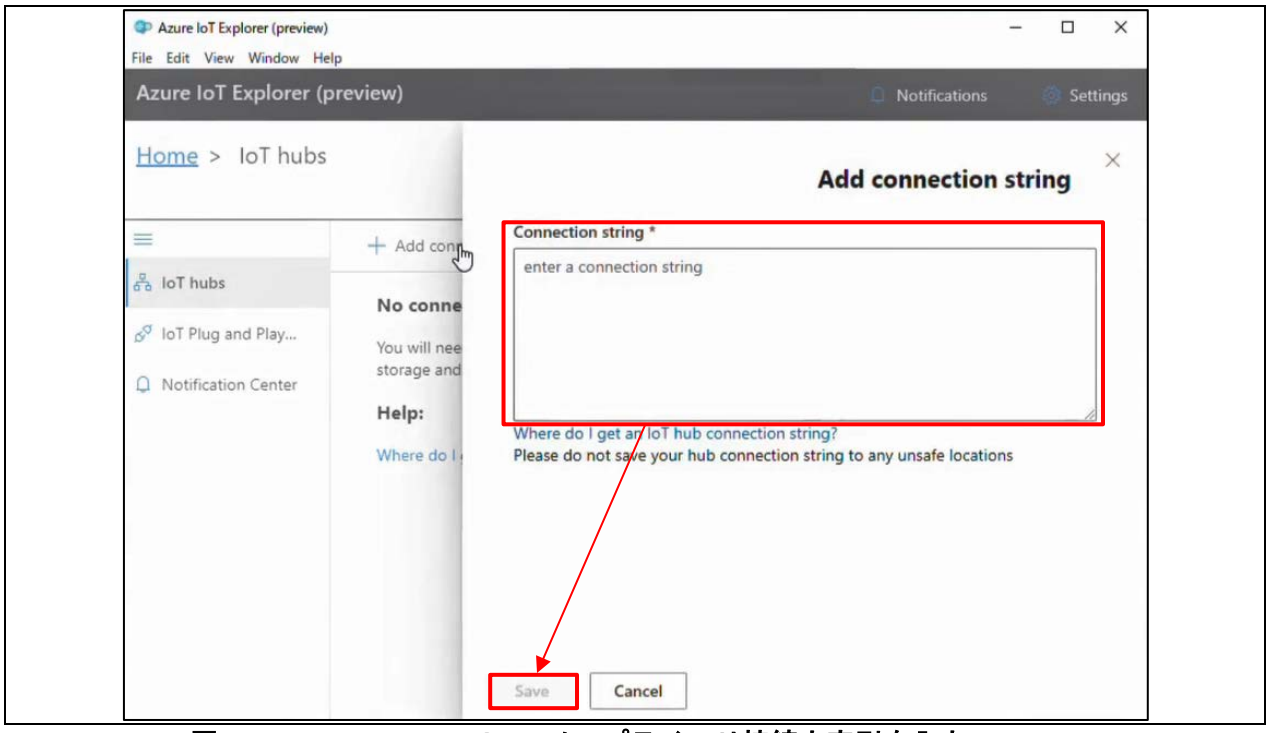

図 3-35 Azure IoT Explorer に プライマリ接続文字列を入力 -> Save

4. ローカルフォルダーに「models」という名前のフォルダを作成。

5. IoT プラグアンドプレイ のクイックスタート web ページにアクセスし、2 つのモデルファイルを「models」フォルダにダウンロード。

   ・クイックスタート web ページ

   https://docs.microsoft.com/azure/iot-develop/set-up-environment

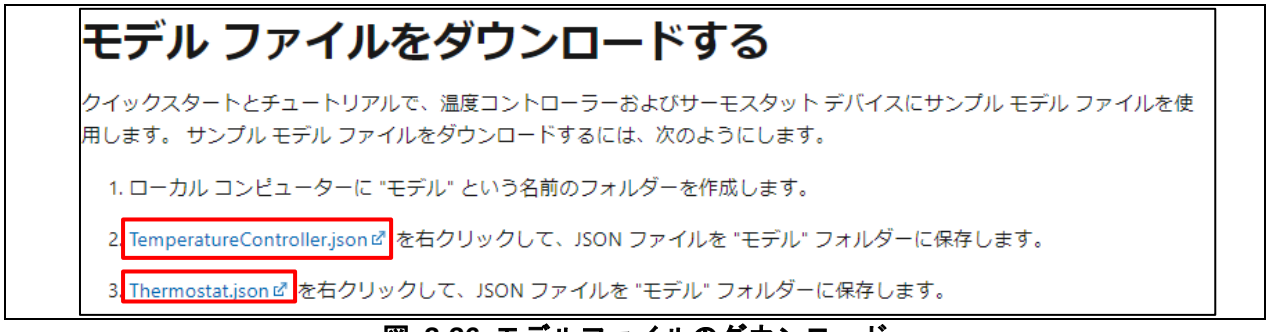

図 3-36 モデルファイルのダウンロード

6. Azure IoT Explorer の IoT Plug and Play components -> Add をクリック -> プルダウンの Local folder をクリック。

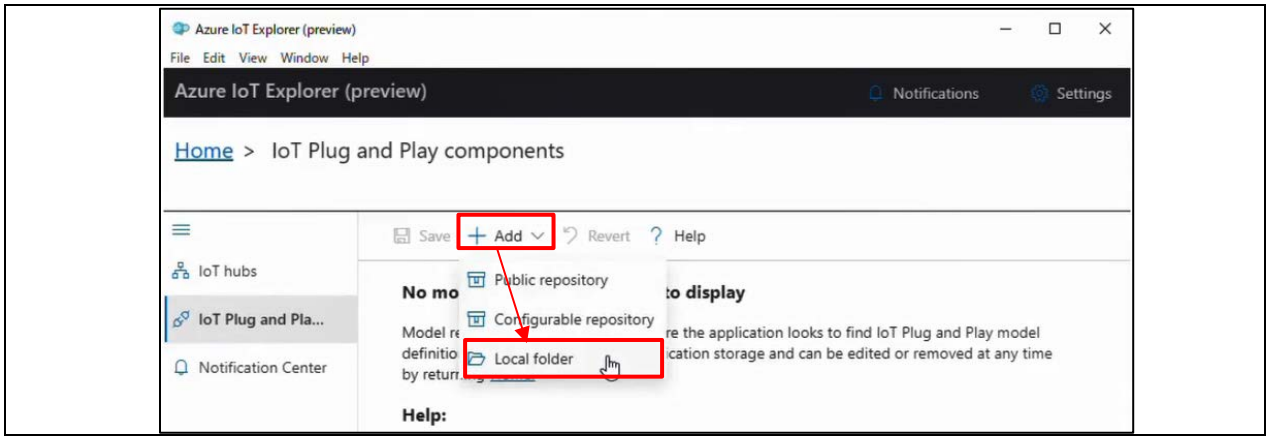

図 3-37 IoT Plug and Play components -> Add クリック -> Local folder

7. Pick a folder -> モデルファイルを保存した「models」フォルダを選択 -> Save をクリック。

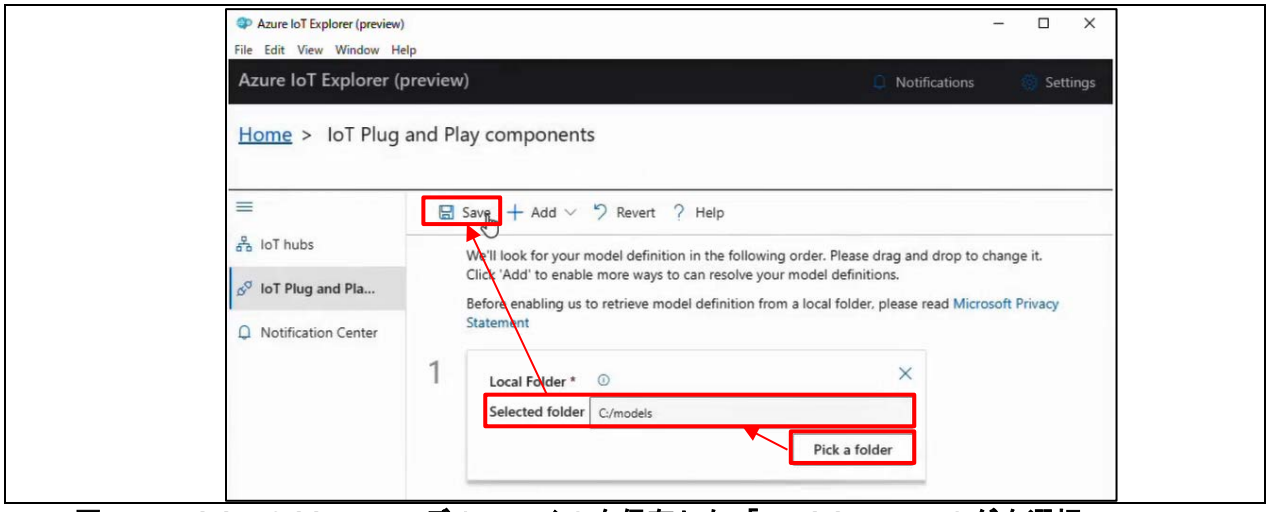

図 3-38 Pick a folder -> モデルファイルを保存した「models」フォルダを選択 -> Save

8. 画面左の IoT hubs -> 作成した IoT Hub をクリック。

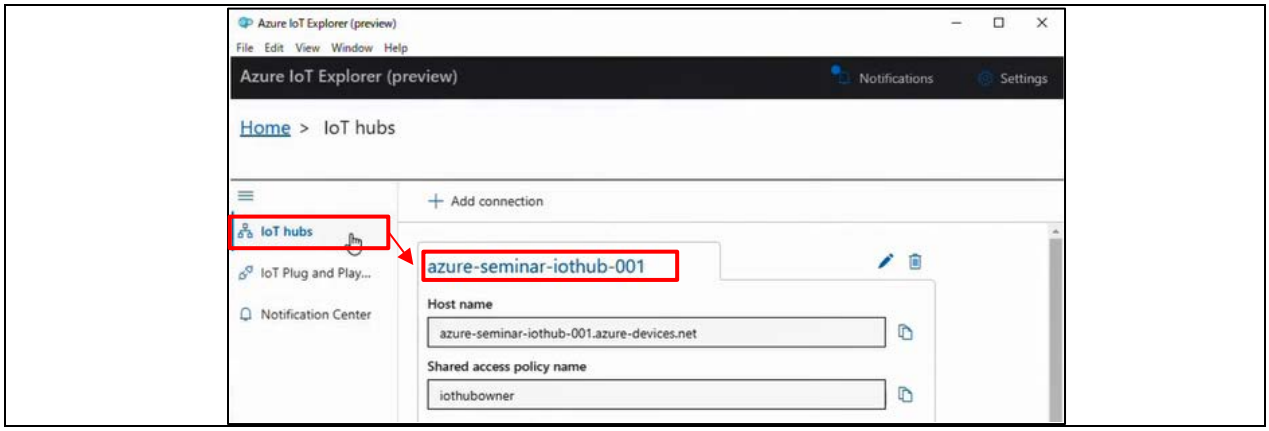

図 3-39 IoT Hub 選択

9. デバイス ID 名をクリック。

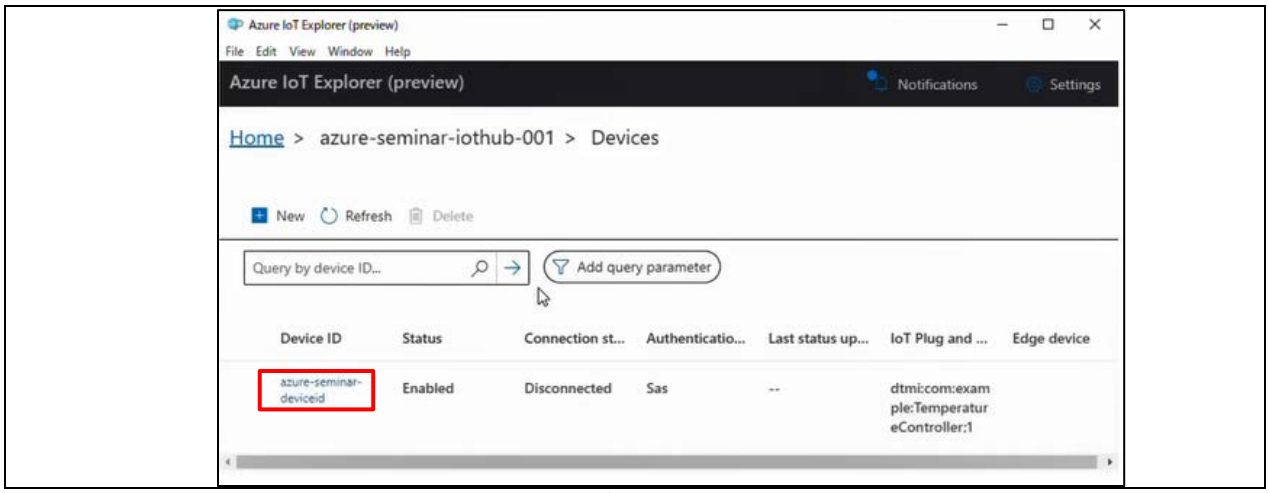

図 3-40 デバイス ID 選択

10. IoT Plug and Play components をクリック。

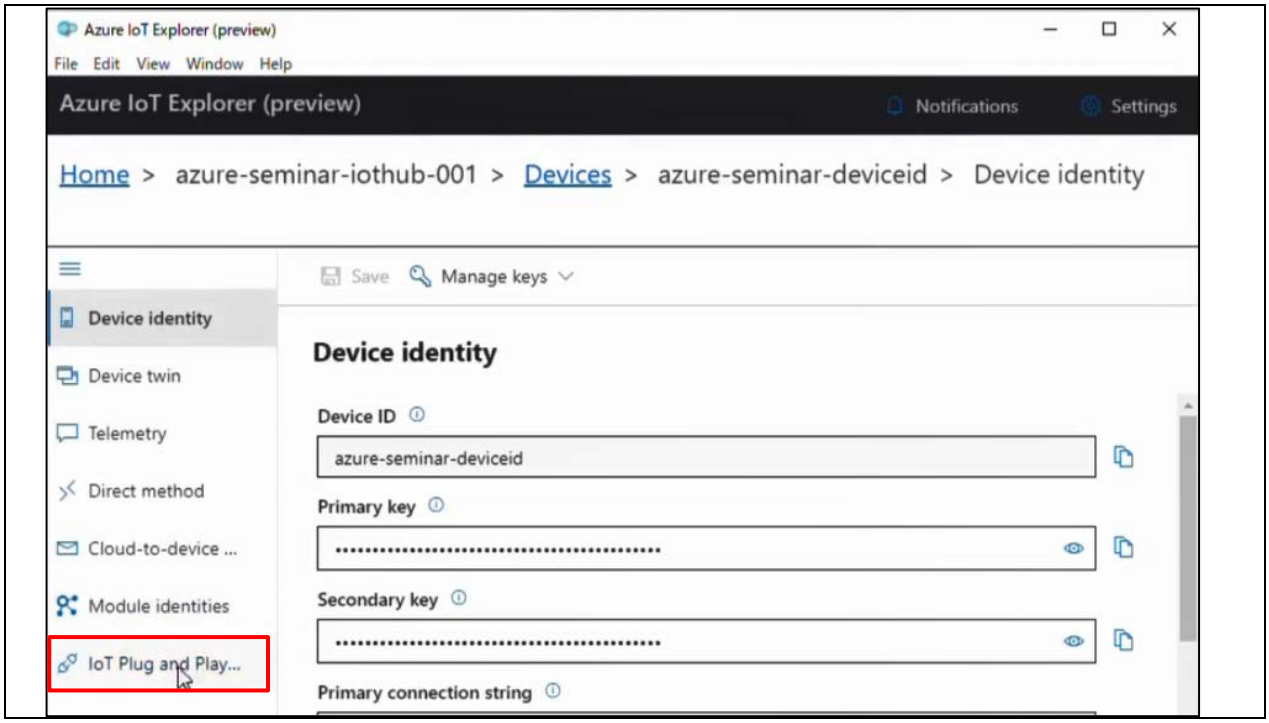

図 3-41 IoT Plug and Play components 選択

11. ページを下にスクロールし、Components から「thermostat1」をクリック。

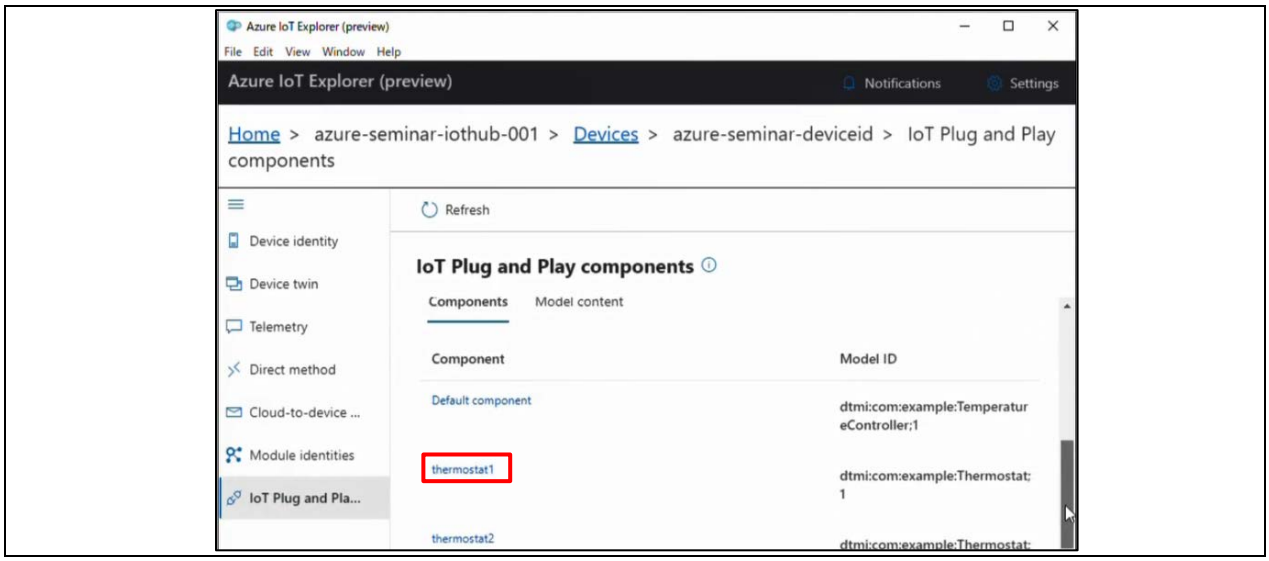

図 3-42 thermostat1 選択

12. Telemetry をクリック。

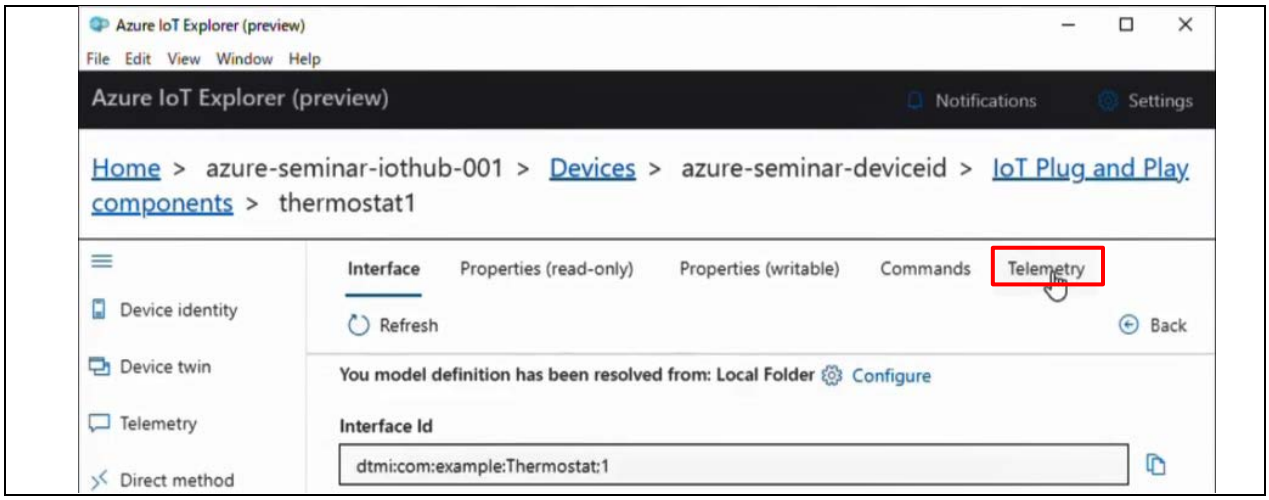

図 3-43 Telemetry 選択

13. Start をクリック。

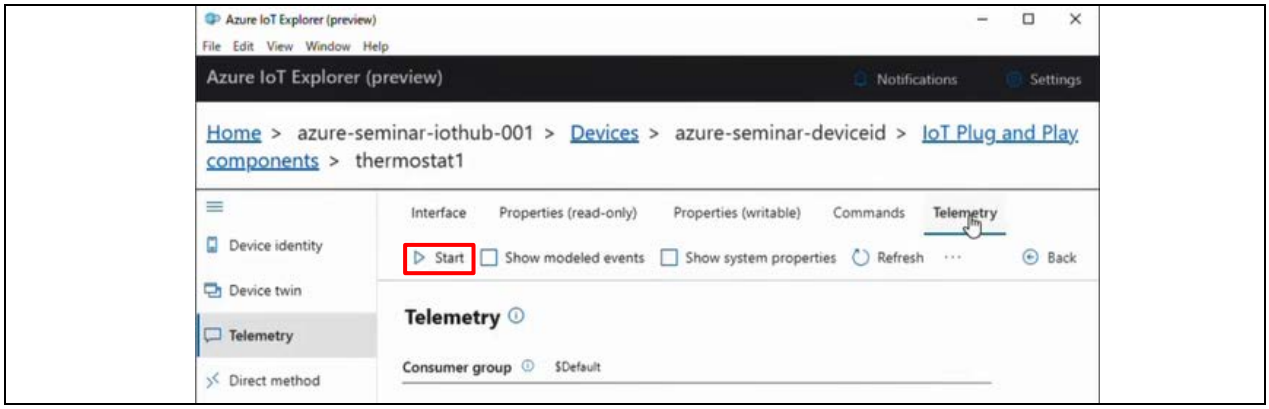

図 3-44 Telemetry 開始

14.「Receiving events」欄にアップロードしたデータが表示されるのを確認 -> Stop をクリック。

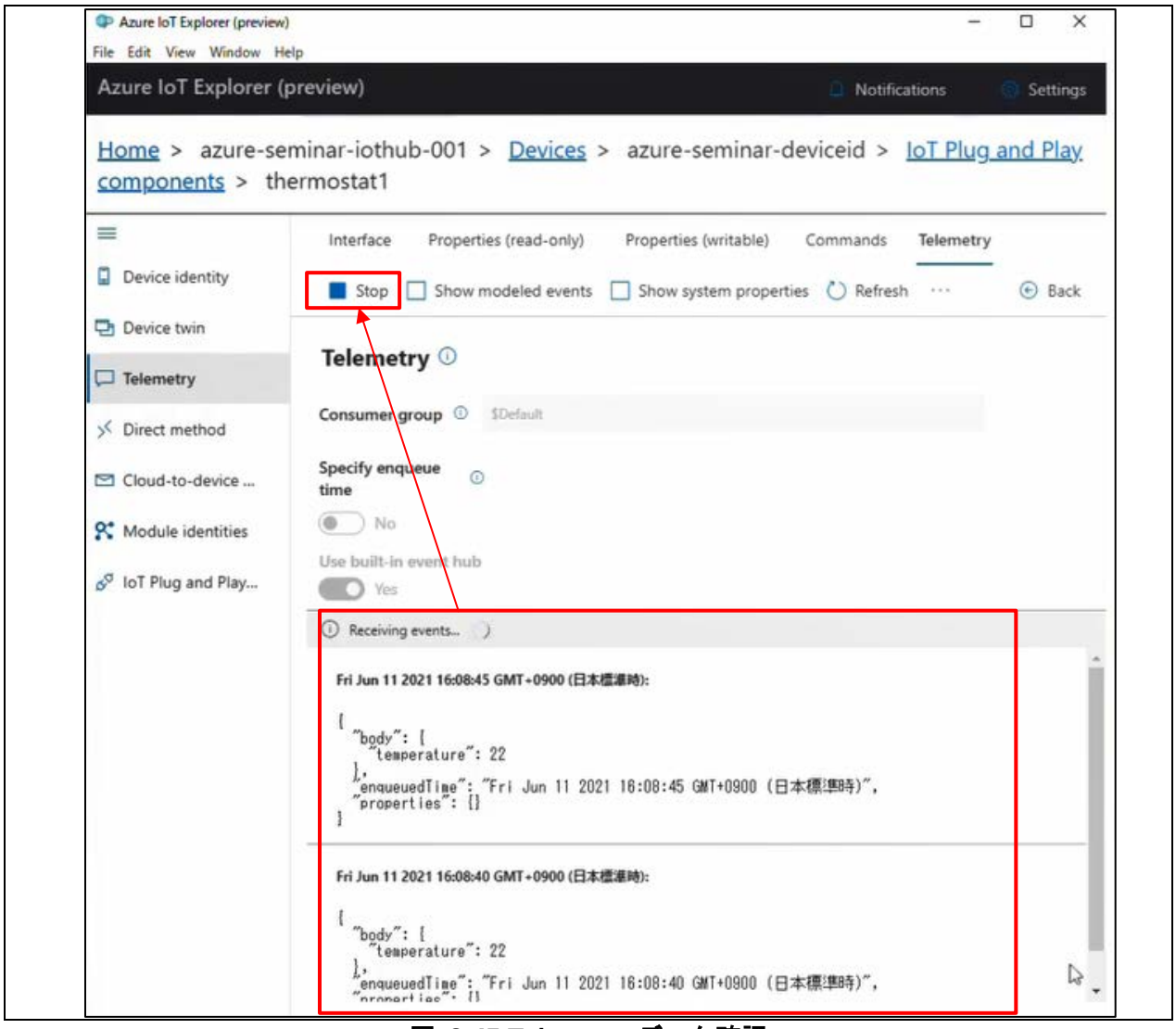

図 3-45 Telemetry データ確認

15. e^2 studio 画面上の停止ボタンをクリック。

図 3-46 デモプログラム停止

4. Azure IoT Explorer による LED ON/OFF 操作

Azure クラウドから RX65N Cloud Kit に搭載されている LED1 の点灯・消灯を操作する手順を以下に示します。

LED1 を操作するには、「3.2 ソフトウェア準備」で GitHub の Azure RTOS サンプルページからダウンロードしたサンプルプロジェクトに変更を加える必要があります。変更内容は「8.付録」をご確認ください。

1. Azure IoT Explorer の作成したデバイス ID -> Device twin をクリック。

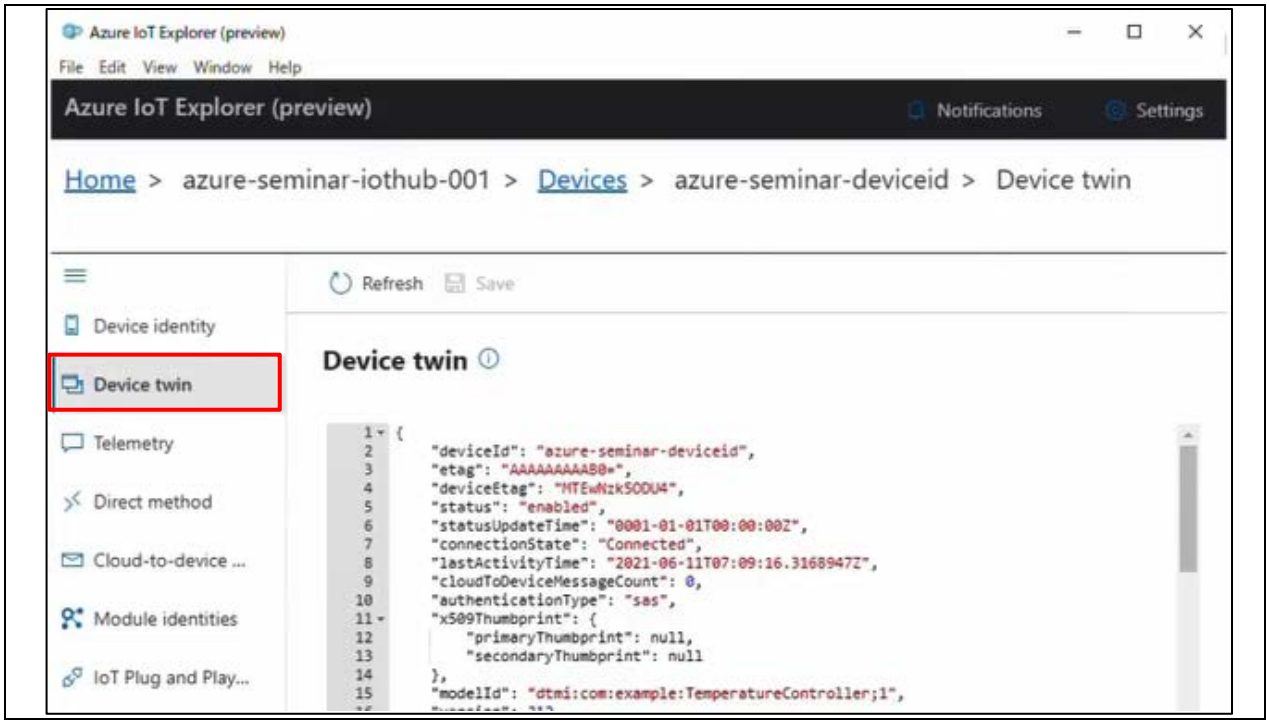

図 4-1 Device twin 画面

2. 以下の情報を desired に追記-> Save をクリック。

```
“LED”: {
    “LED1”: 1
},
```

図 4-2 メッセージペイロード LED1 点灯

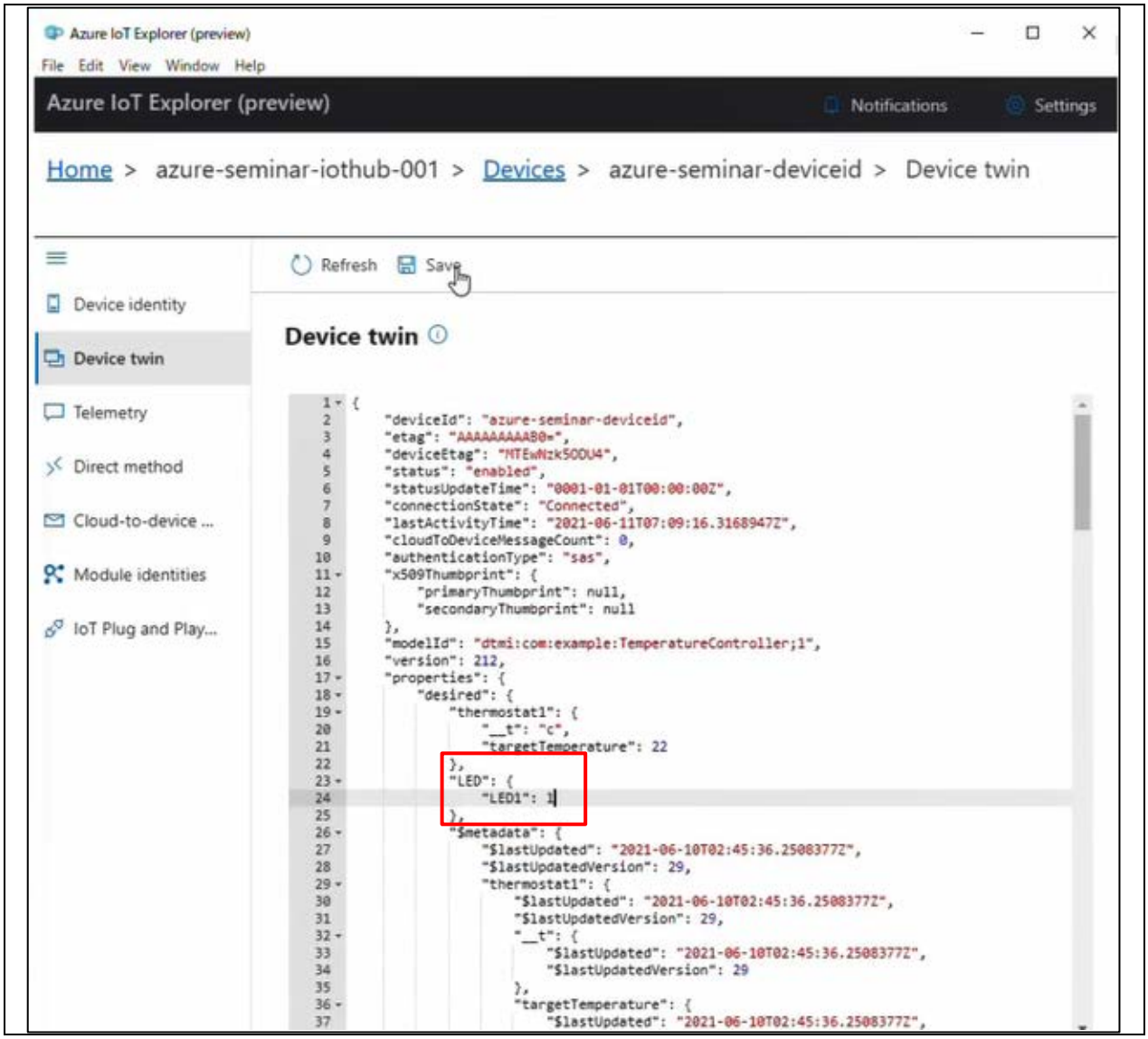

図 4-3 Device twin 設定状態

3. RX65N Cloud Kit LED1 の点灯を確認する。

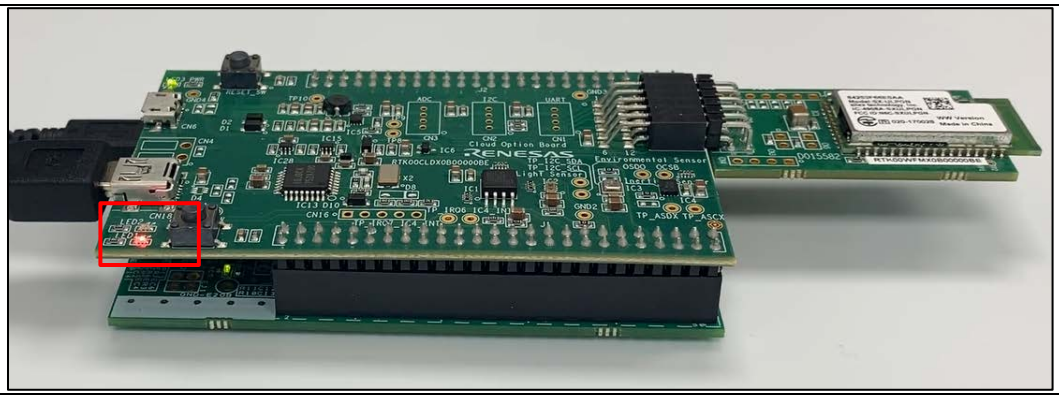

図 4-4 LED1 点灯

4. 以下の情報を desired に追記 -> Save をクリック。

```
“LED”: {
    “LED1”: 0
},
```

図 4-5 メッセージペイロード LED1 消灯

5. LED1 の消灯を確認する。

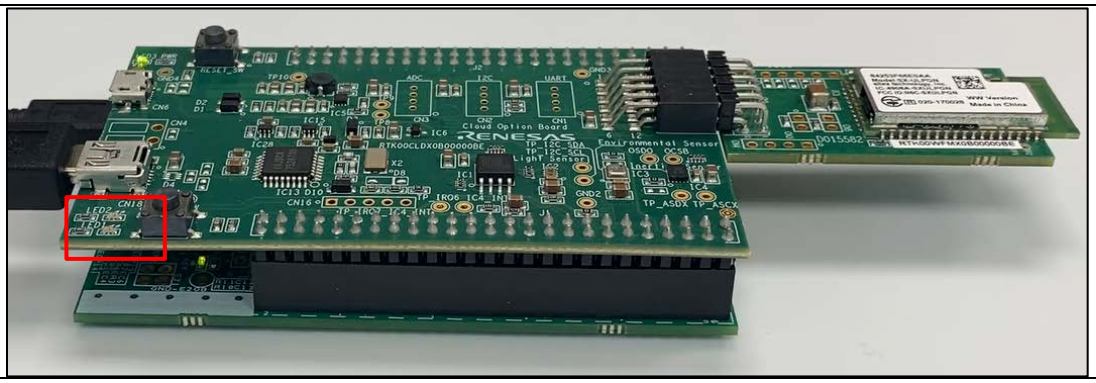

図 4-6 LED1 消灯

5. センサデータの可視化

RX65N Cloud Kit から Azure クラウドにアップロードしたセンサデータ[注]を、web アプリを使って可視化する手順を以下に示します。

注 本ドキュメントのデモプログラムはセンサデータとして固定値をアップロードします。

5.1 ソフトウェア準備

1. Azure Portal の作成した IoT Hub ページにアクセスし、組み込みのエンドポイント -> 新しいコンシューマーグループを作成する に任意の名前を入力 -> 保存 をクリック。

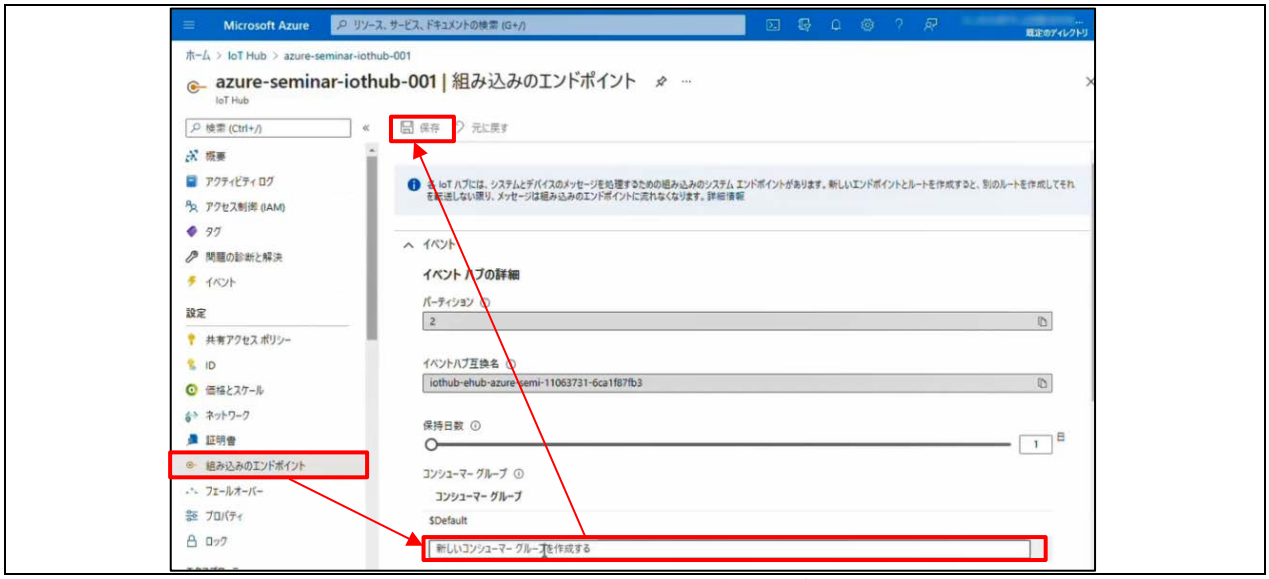

図 5-1 コンシューマーグループ作成

2. コンシューマーグループ名をテキストエディタ等にメモ。

コンシューマーグループ名の情報は後で使用します。

3. 共有アクセスポリシー -> service -> プライマリ接続文字列をテキストエディタ等にメモ。

プライマリ接続文字列の情報は後で使用します。

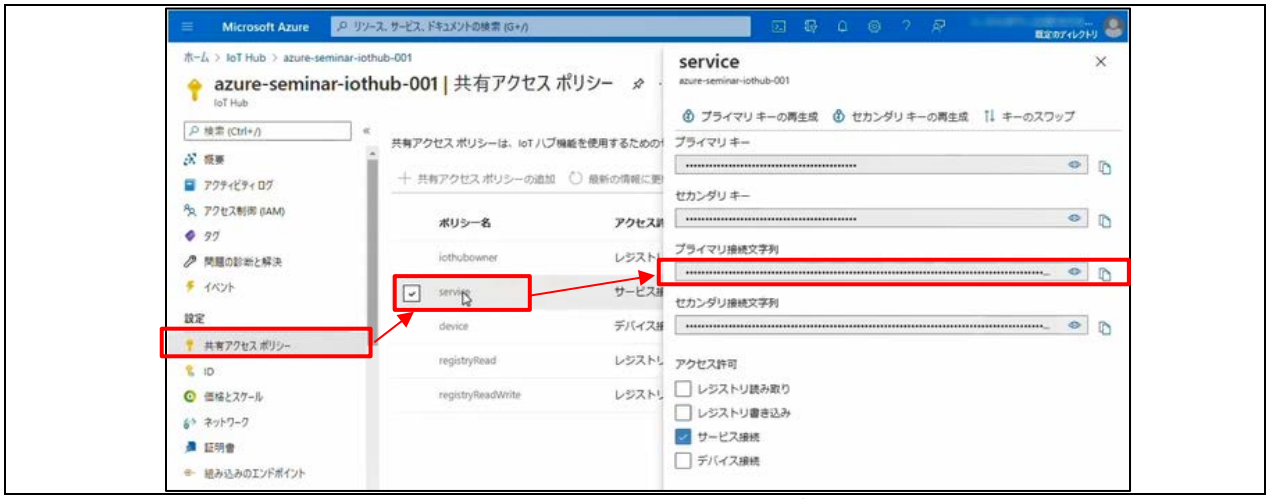

図 5-2 共有アクセスポリシー -> service -> プライマリ接続文字列

5.2 web アプリ実行

1. コマンドプロンプトを起動し、以下の順番でコマンドを実行。

(1)GitHub から可視化用 web アプリのサンプルコードをダウンロード

```
git clone https://github.com/Azure-Samples/web-apps-node-iot-hub-data-visualization.git
```

図 5-3 web アプリ サンプルコードダウンロード

(2)サンプルコードの格納場所へフォルダ移動

```
cd web-apps-node-iot-hub-data-visualization
```

図 5-4 サンプルコードフォルダ移動

(3)環境変数を設定

```
set IotHubConnectionString=[「5.1 ソフトウェア準備の 3.」プライマリ接続文字列]
set EventHubConsumerGroup=[「5.1 ソフトウェア準備の 2.」コンシューマーグループ名]
```

図 5-5 環境変数設定

(4)web サイトを開始

web サイトを終了する場合は Ctrl-C を入力します。

```
npm install
npm start
```

図 5-6 web サイト起動

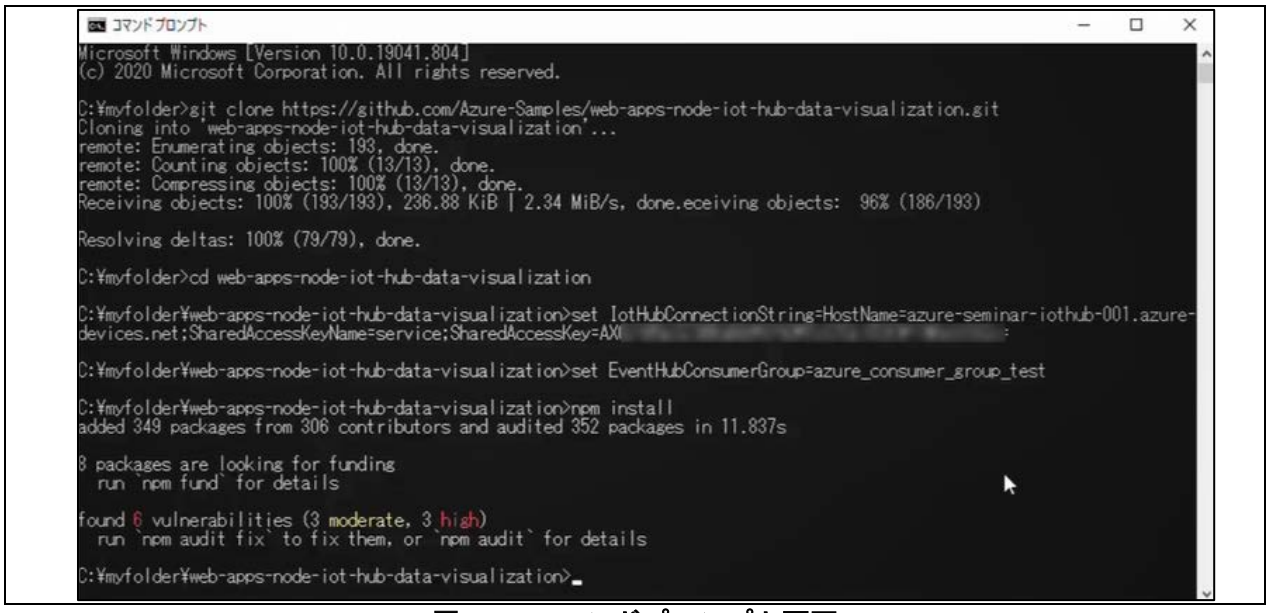

図 5-7 コマンドプロンプト画面

2. 「3.4 デモプログラム実行」の手順でデモプログラムを実行する。

3. ブラウザを起動して http://localhost: 3000 にアクセスし、温度データのグラフを確認する。

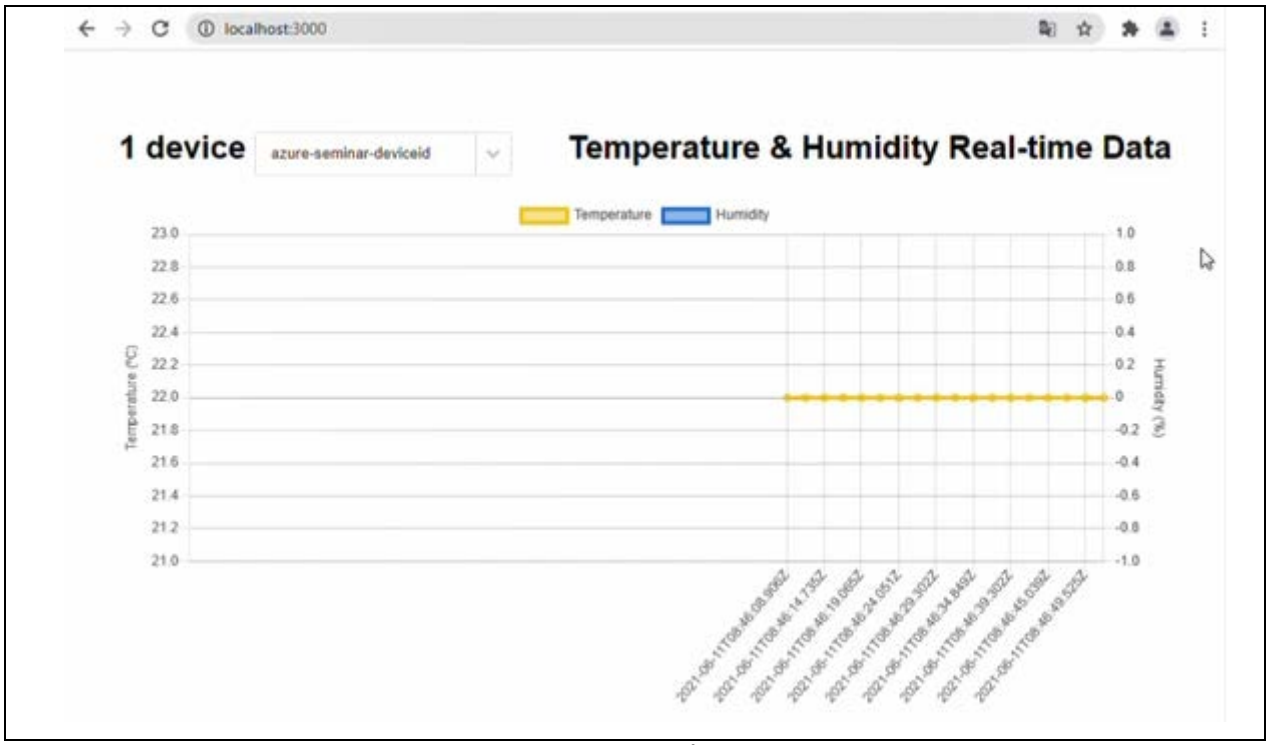

図 5-8 web アプリ画面

5.3 グラフ変化表示

デモプログラムでは、アップロードするセンサデータの値を変更することができます。変更したデータはグラフで確認することができます。

1. Azure IoT Explorer の作成したデバイス ID -> IoT Plug and Play components -> ページを下にスクロールし、Components から「thermostat1」をクリック。

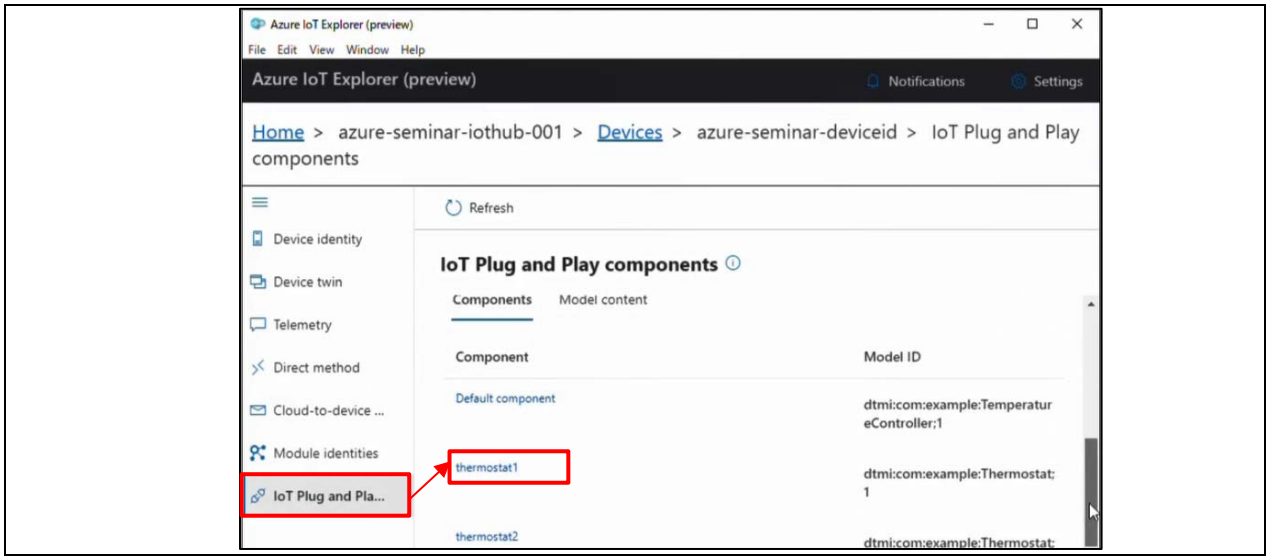

図 5-9 Components 画面

2. Properties(writable) をクリック。

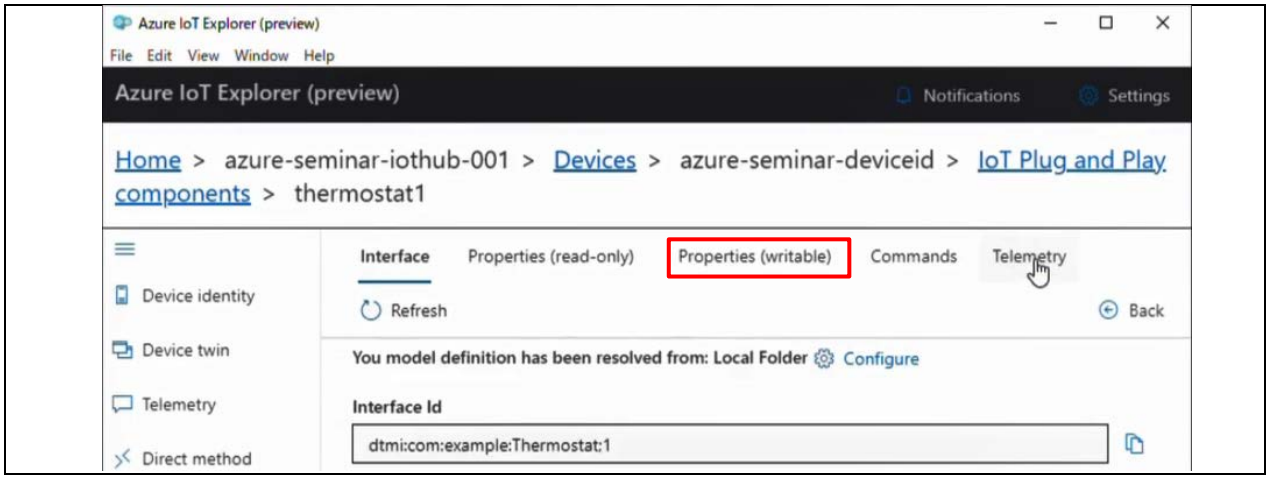

図 5-10 Properties(writable) 選択

3. 「target Temperature」 に 温度データを入力 -> Update desired value をクリック。

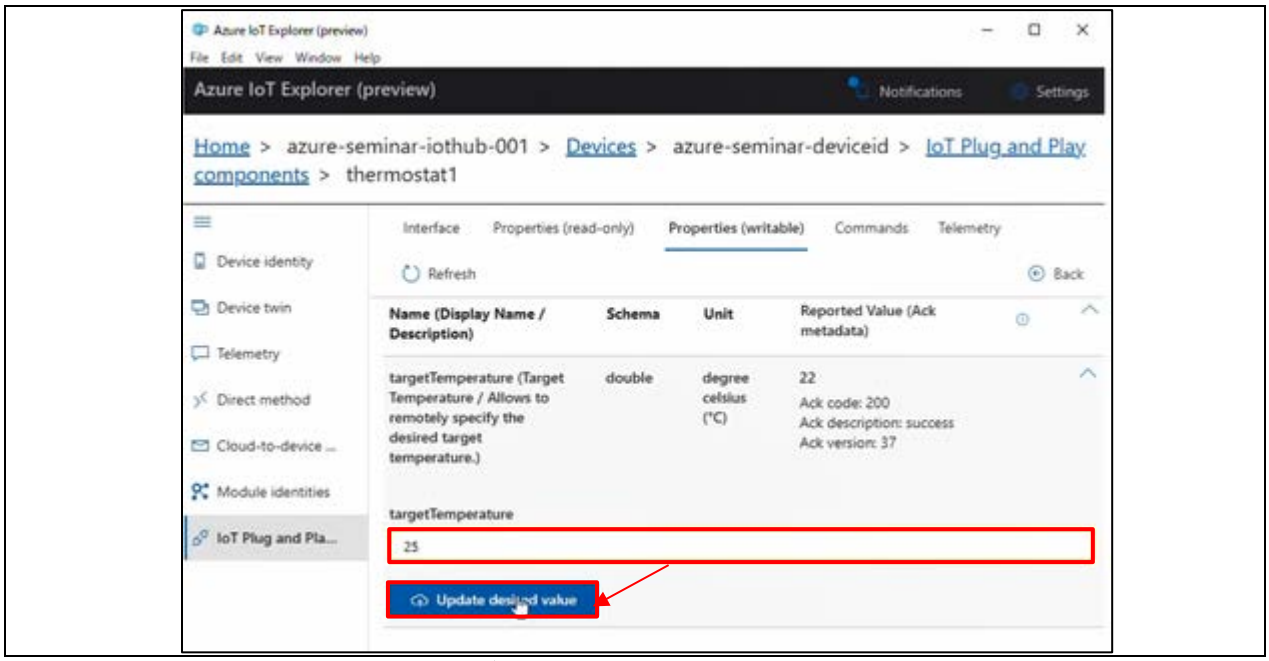

図 5-11 温度データを入力 -> Update desired value

4. http://localhost: 3000 にアクセスし、グラフが変化していることを確認する。

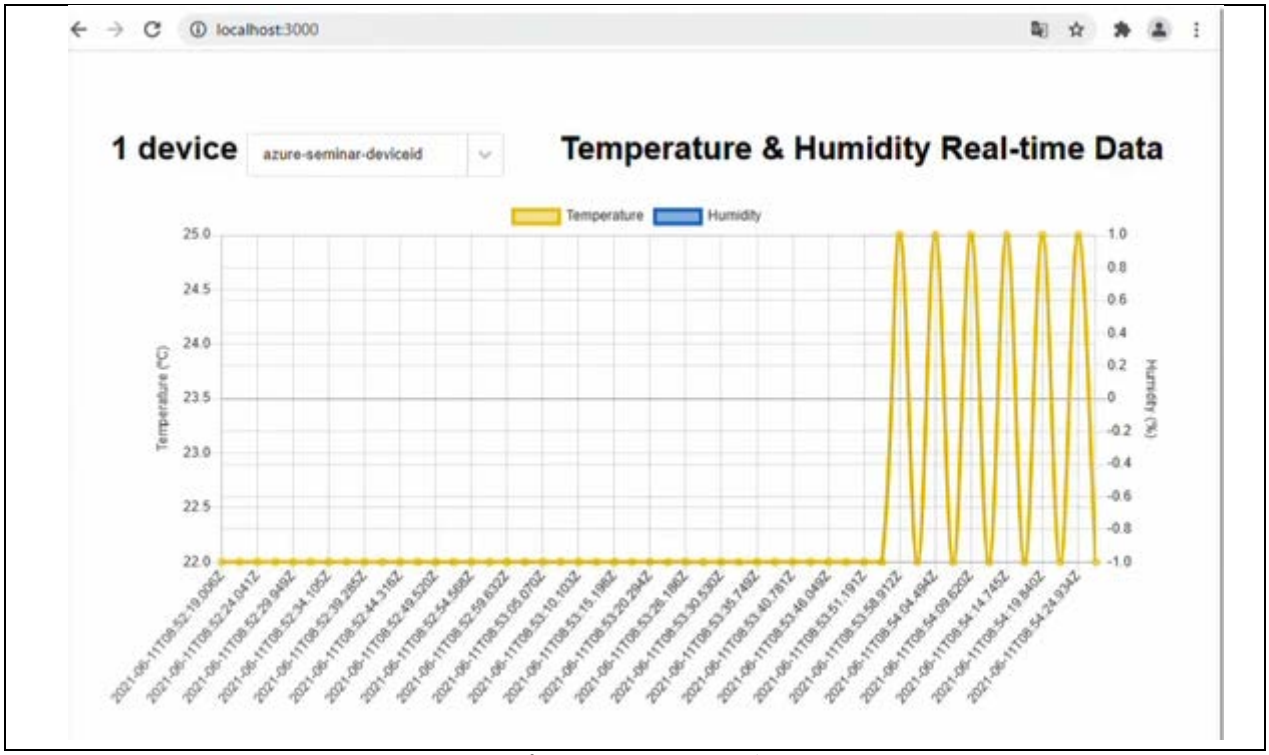

図 5-12 web アプリ画面でセンサデータ変化を確認

5. e^2 studio 画面上の停止ボタンをクリックし、デモの実行を終了する。

6. デモプログラム実行時の注意事項について

Azure のサービスは起動していると料金が発生します。使用しない場合は削除してください。

7. ウェブサイトおよびサポート

Azure RTOS: https://azure.microsoft.com/support/community/

Azure RTOS GitHub: https://github.com/azure-rtos

8. 付録

「4. Azure IoT Explorer による LED ON/OFF 操作」を行うために、サンプルプログラムに対して必要な変更内容を示します。

注 説明文中の行番号は、ダウンロードした時点のサンプルプログラムの行数を示しています。

1. {$base_folder}/rx65n-cloud-kit/e2studio_gnurx/sample_pnp_temperature_controller/src/ sample_pnp_temperature_controller.c に、以下の変更を加えます。

(1) 以下の変数の宣言を追加。

```
l.143   static const CHAR sample_device_info_component[] = "deviceInformation";
        static SAMPLE_PNP_THERMOSTAT_COMPONENT sample_led;
        static const CHAR sample_led_component[] = "LED";
        static double sample_led_last_device_reported;
```

追加

図 8-1 sample_pnp_temperature_controller.c 変更(1)

(2) sample_components 配列に LED 用の要素を追加。

```
l.146   static const CHAR *sample_components[] = { sample_thermostat_1_component,
l.147                                              sample_thermostat_2_component,
                                                   sample_led_component,
l.148                                              sample_device_info_component };
```

追加

図 8-2 sample_pnp_temperature_controller.c 変更(2)

(3) sample_desired_property_callback() 関数の以下の if 文に条件式を追加。

```
l.811   if (component_name_ptr == NULL || component_name_len == 0)
l.812   {
           (略)
l.834   }
        else if (sample_pnp_led_process_property_update(&sample_led,
                                                        (NX_AZURE_IOT_HUB_CLIENT *)userContextCallback,
                                                        component_name_ptr, component_name_len,
                                                        property_name_ptr, property_name_len,
                                                        &property_value_reader, version) == NX_AZURE_IOT_SUCCESS)
        {
              printf("property updated of led1¥r¥n");
        }
l.835   else                                                                                          追加
l.836   {
          (略)
        }
```

図 8-3 sample_pnp_temperature_controller.c 変更(3)

(4) sample_components_init() 関数の以下の if 文に条件式を追加。

```
l.1232  if ((status = sample_pnp_thermostat_init(&sample_thermostat_1,
l.1233                                           (UCHAR *)sample_thermostat_1_component,
l.1234                                           sizeof(sample_thermostat_1_component) - 1,
l.1235                                           SAMPLE_DEFAULT_START_TEMP_CELSIUS)))
l.1236  {
              (略)
l.1247  }
        else if ((status = sample_pnp_thermostat_init(&sample_led,
                                                      (UCHAR *)sample_led_component,
                                                      sizeof(sample_led_component) - 1,
                                                      SAMPLE_DEFAULT_START_TEMP_CELSIUS)))
        {
              printf("Failed to initialize %s: error code = 0x%08x¥r¥n",
                     sample_led_component, status);
        }
l.1248                                                                                                追加
l.1249  sample_thermostat_1_last_device_max_temp_reported = 0;
```

図 8-4 sample_pnp_temperature_controller.c 変更(4)

2. {$base_folder}/rx65n-cloud-kit/e2studio_gnurx/sample_pnp_temperature_controller/src/ sample_pnp_thermostat_component.c に、以下の変更を加えます。

(1) platform.h ファイルのインクルード設定を追加。

```
l.12    #include "sample_pnp_thermostat_component.h"
        #include "platform.h"        追加
```

図 8-5 sample_pnp_thermostat_component.c 変更(1)

(2) Azure と通信するデータのキー名 ”LED1” の定数を追加。

```
l.36    static const CHAR temp_response_description_failed[] = "failed";
        static const CHAR led_property_name[] = "LED1";        追加
```

図 8-6 sample_pnp_thermostat_component.c 変更(2)

(3) Device twin 操作に使用する 2 関数を追加。

```
最終行  static VOID sample_send_led_report(SAMPLE_PNP_THERMOSTAT_COMPONENT *handle,
                                           NX_AZURE_IOT_HUB_CLIENT *iothub_client_ptr, double temp,
                                           INT status_code, UINT version, const CHAR *description)
{
    UINT bytes_copied;
    UINT response_status;
    UINT request_id;
    NX_AZURE_IOT_JSON_WRITER json_writer;
    ULONG reported_property_version;

    /* Build telemetry JSON payload */
    if (nx_azure_iot_json_writer_with_buffer_init(&json_writer, scratch_buffer, sizeof(scratch_buffer)))
    {
        printf("Failed to create json writer¥r¥n");
        return;
    }

    if (nx_azure_iot_pnp_helper_build_reported_property_with_status(
                                handle -> component_name_ptr, handle -> component_name_length,
                                (UCHAR *)led_property_name,
                                sizeof(led_property_name) - 1,
                                append_temp, (VOID *)&temp, status_code,
                                (UCHAR *)description,
                                strlen(description), version, &json_writer))
    {
        printf("Failed to create reported response¥r¥n");
    }
    else
    {
        bytes_copied = nx_azure_iot_json_writer_get_bytes_used(&json_writer);
        if (nx_azure_iot_hub_client_device_twin_reported_properties_send(iothub_client_ptr,
                                                                         scratch_buffer, bytes_copied,
                                                                         &request_id, &response_status,
                                                                         &reported_property_version,
                                                                         (5 * NX_IP_PERIODIC_RATE)))
        {
            printf("Failed to send reported response¥r¥n");
        }
    }
    nx_azure_iot_json_writer_deinit(&json_writer);
}
```

図 8-7 sample_pnp_thermostat_component.c 変更(3)-1

最終行

```
UINT sample_pnp_led_process_property_update(SAMPLE_PNP_THERMOSTAT_COMPONENT *handle,
                                            NX_AZURE_IOT_HUB_CLIENT *iothub_client_ptr,
                                            CHAR *component_name_ptr, UINT component_name_length,
                                            UCHAR *property_name_ptr, UINT property_name_length,
                                            NX_AZURE_IOT_JSON_READER *property_value_reader_ptr, UINT version)
{
    double parsed_value = 0;
    INT status_code;
    const CHAR *description;

    if (handle == NX_NULL)
    {
        return(NX_NOT_SUCCESSFUL);
    }

    if (handle -> component_name_length != component_name_length ||
       strncmp((CHAR *)handle -> component_name_ptr,
                (CHAR *)component_name_ptr, component_name_length) != 0)
    {
        return(NX_NOT_SUCCESSFUL);
    }

    if (property_name_length != (sizeof(led_property_name) - 1) ||
        strncmp((CHAR *)property_name_ptr, (CHAR *)led_property_name, property_name_length) != 0)
    {
        printf("PnP property=%.*s is not supported on thermostat component¥r¥n",
                property_name_length, property_name_ptr);
        status_code = 404;
        description = temp_response_description_failed;
    }
    else if (nx_azure_iot_json_reader_token_double_get(property_value_reader_ptr, &parsed_value))
    {
        status_code = 401;
        description = temp_response_description_failed;
    }
    else
    {
        status_code = 200;
        description = temp_response_description_success;

        if (parsed_value == 1)    // LED ON
        {
            PORTB.PDR.BIT.B0 = 1;
            PORTB.PODR.BIT.B0 = 0;
        }
        else      // LED OFF
        {
            PORTB.PDR.BIT.B0 = 1;
            PORTB.PODR.BIT.B0 = 1;
        }
    }

    sample_send_led_report(handle, iothub_client_ptr, parsed_value,
                                    status_code, version, description);

    return(NX_AZURE_IOT_SUCCESS);
}
```

図 8-8 sample_pnp_thermostat_component.c 変更(3)-2

3. {$base_folder}/rx65n-cloud-kit/e2studio_gnurx/sample_pnp_temperature_controller/src/sample_pnp_thermostat_component.h に、以下の変更を加えます。

(1) sample_pnp_led_process_property_update() 関数の宣言を追加。

```
l.65    UINT sample_pnp_thermostat_process_property_update(SAMPLE_PNP_THERMOSTAT_COMPONENT *handle,
l.66                                                       NX_AZURE_IOT_HUB_CLIENT *iothub_client_ptr,
l.67                                                       UCHAR *component_name_ptr, UINT component_name_length,
l.68                                                       UCHAR *property_name_ptr, UINT property_name_length,
l.69                                                       NX_AZURE_IOT_JSON_READER *property_value_reader_ptr, UINT version);

        UINT sample_pnp_led_process_property_update(SAMPLE_PNP_THERMOSTAT_COMPONENT *handle,
                                                    NX_AZURE_IOT_HUB_CLIENT *iothub_client_ptr,
                                                    UCHAR *component_name_ptr, UINT component_name_length,
                                                    UCHAR *property_name_ptr, UINT property_name_length,
                                                    NX_AZURE_IOT_JSON_READER *property_value_reader_ptr, UINT version);
                                                                                                          追加
```

図 8-9 sample_pnp_thermostat_component.h 変更(1)

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	Aug.31.21	-	初版発行

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 静電気対策
   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。
2. 電源投入時の処置
   電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. 電源オフ時における入力信号
   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。
4. 未使用端子の処理
   未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。
5. クロックについて
   リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
6. 入力端子の印加波形
   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。
7. リザーブアドレス（予約領域）のアクセス禁止
   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
8. 製品間の相違について
   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。回路、ソフトウェアおよびこれらに関連する情報を使用する場合、お客様の責任において、お客様の機器・システムを設計ください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品または本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を組み込んだ製品の輸出入、製造、販売、利用、配布その他の行為を行うにあたり、第三者保有の技術の利用に関するライセンスが必要となる場合、当該ライセンス取得の判断および取得はお客様の責任において行ってください。
5. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
6. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。

   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等

   高品質水準：輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等

   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
7. あらゆる半導体製品は、外部攻撃からの安全性を 100%保証されているわけではありません。当社ハードウェア／ソフトウェア製品にはセキュリティ対策が組み込まれているものもありますが、これによって、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されたあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵入行為（「脆弱性問題」といいます。）によって影響を受けないことを保証しません。当社は、脆弱性問題に起因しまたはこれに関連して生じた損害について、一切責任を負いません。また、法令において認められる限りにおいて、本資料および当社ハードウェア／ソフトウェア製品について、商品性および特定目的との合致に関する保証ならびに第三者の権利を侵害しないことの保証を含め、明示または黙示のいかなる保証も行いません。
8. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
10. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
12. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
13. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
14. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

(Rev.5.0-1　2020.10)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。