Azure Sphere

Self Hands-on

2019/5/13

Takashi Matsuoka

Azure Sphere Self Hands-on

Lab#1 The Blink Sample

Lab#2 Create Digital I/O App from the Blank App

Lab#3 To use Static Library

Lab#4 The Azure IoT Hub Sample

Lab#5 Create Telemetry App from the Blank App

Lab#6 Send to Azure IoT Central w/o DPS

Lab#7 Send to Azure IoT Central w DPS

Prerequisites

- MT3620 Development Kit

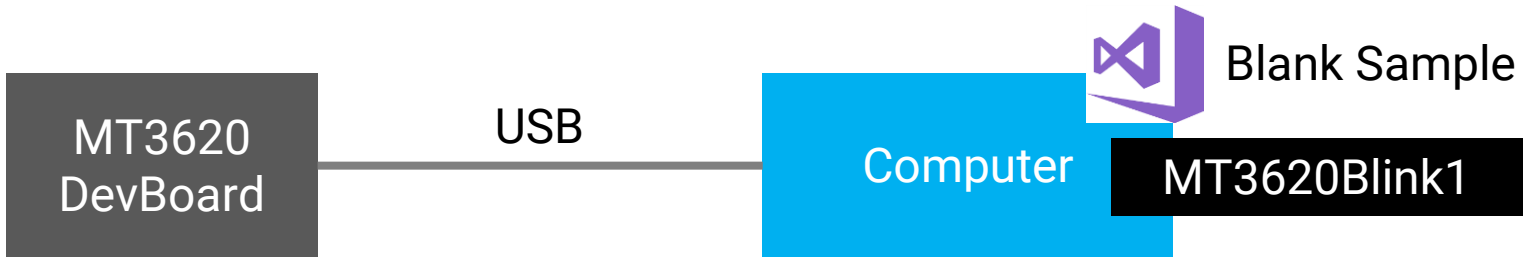
- Windows 10 1803~
- Visual Studio 2017 15.7~
- Azure Sphere SDK Preview for Visual Studio
- Git for Windows

- Microsoft Azure Account

- Update the OS
- Set up an account for Azure Sphere
- Claim your device
- Prepare your device for development and debugging

Lab#1

Lab#1 The Blink Sample



得ること

- MT3620DevBoardとVisual Studioの関係
- アプリケーションの作成~実行~停止
- デバイス上のアプリケーションを削除

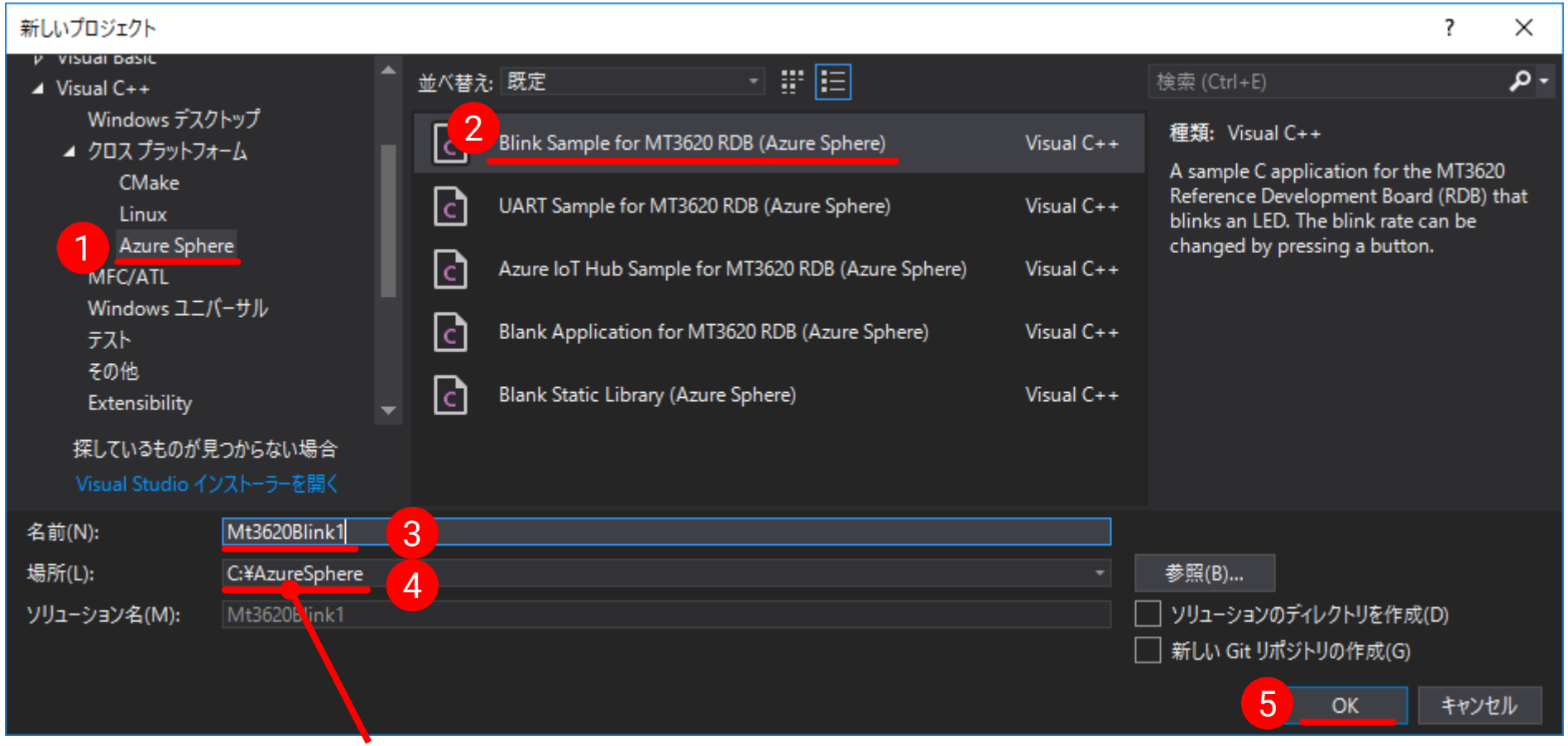
やること

1. Blink Sampleを新規作成
2. Blink Sampleをビルド
3. Blink Sampleをデバイスへデプロイ、実行、停止
4. デバイスのBlink Sampleを削除

Lab#1.1 Blink Sampleを新規作成

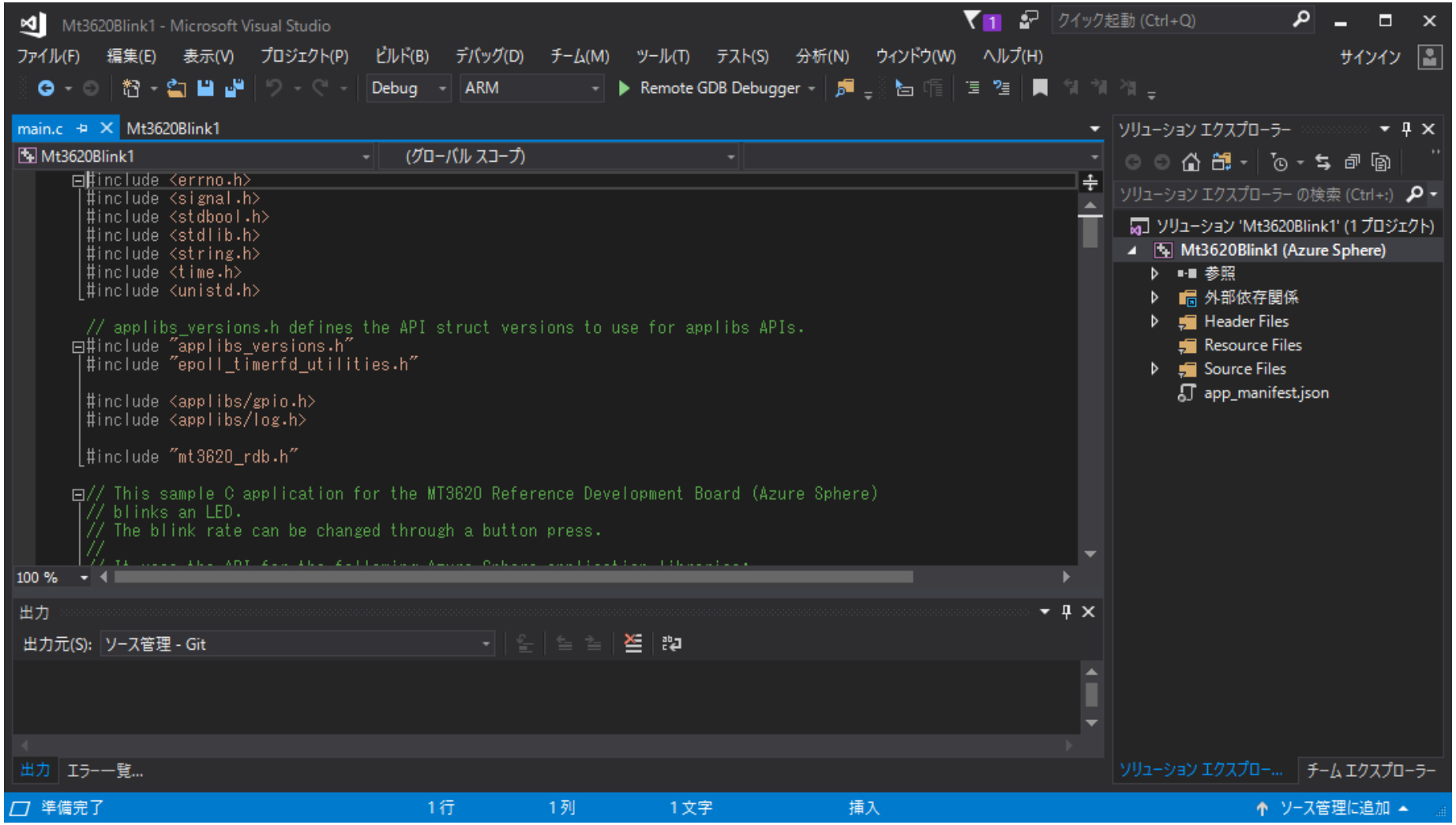
Visual C++ > クロスプラットフォーム > Azure Sphere

Blink Sample for MT3620 RDB



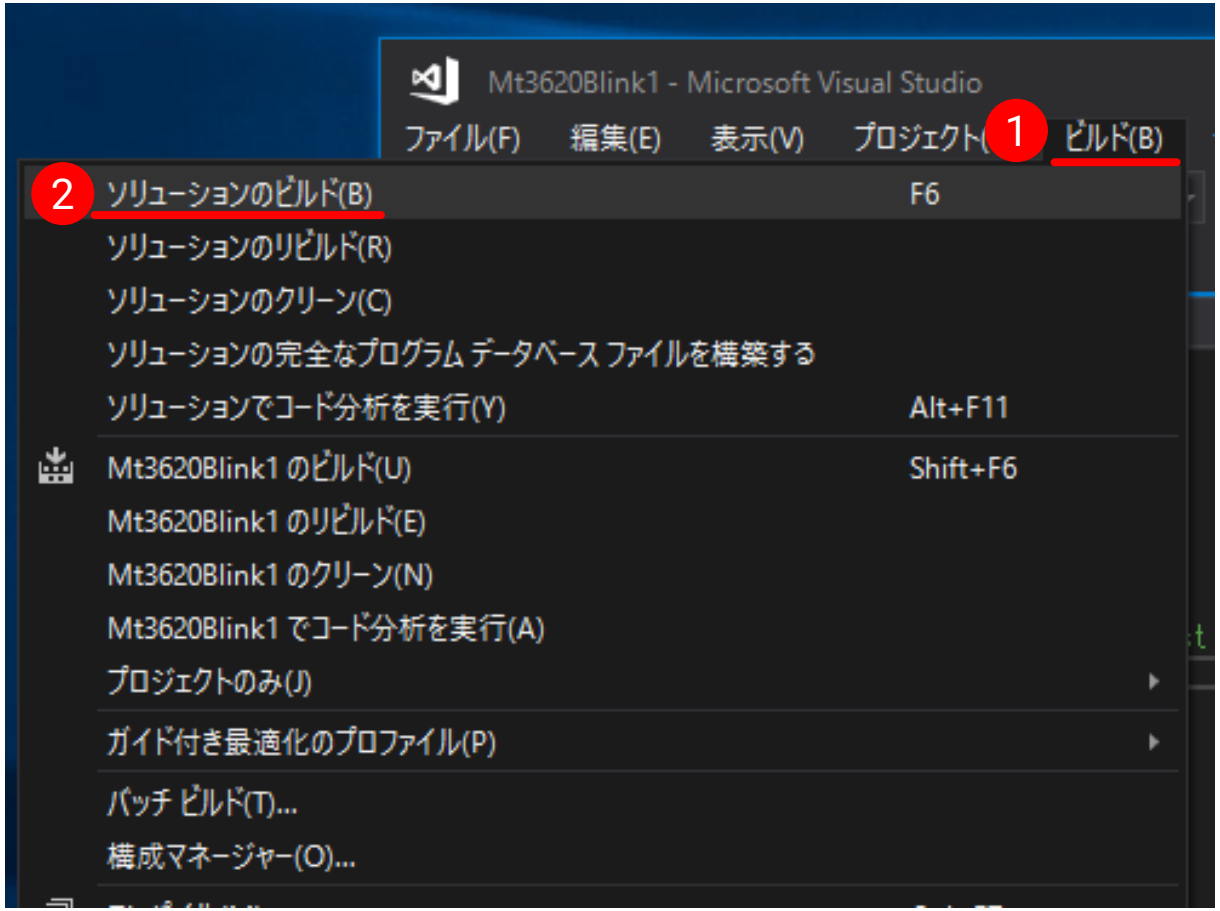
本ハンズオンでは、C:¥AzureSphere を指定してください

Lab#1.1 Blink Sampleを新規作成



Lab#1.2 Blink Sampleをビルド

ビルド > ソリューションのビルド



Lab#1.2 Blink Sampleをビルド

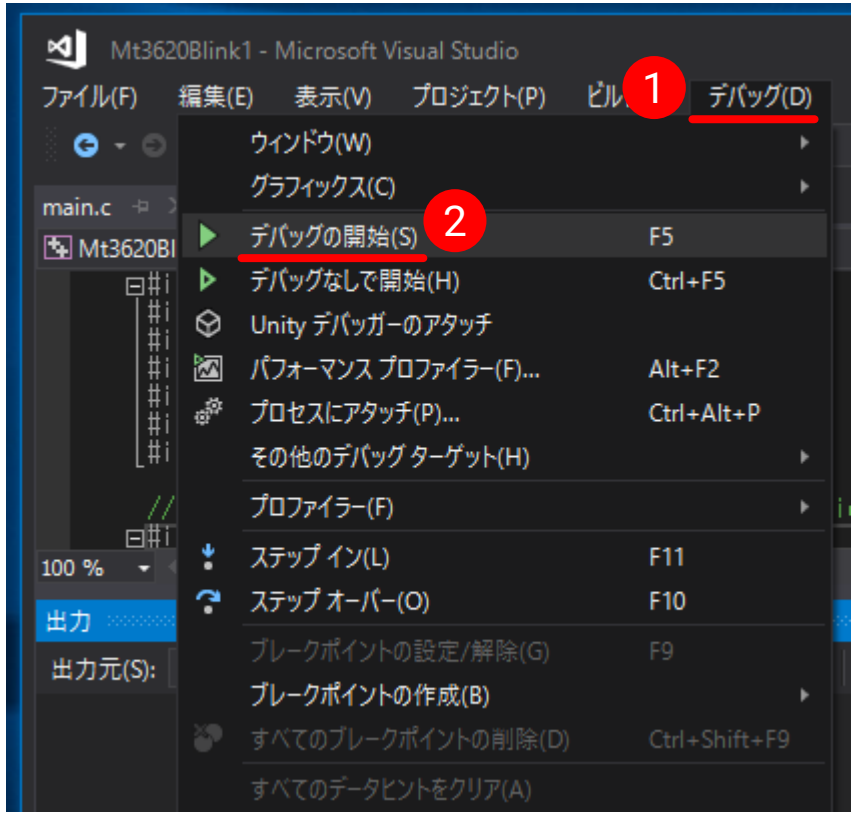
出力 : ビルド

```
1>------ ビルド開始: プロジェクト: Mt3620Blink1, 構成: Debug ARM ------
1>Azure Sphere Utility version 18.11.3.20146
1>Copyright (C) Microsoft Corporation. All rights reserved.
1>
1>Start time (UTC): Friday, 04 January 2019 08:31:45
1>verbose: Creating image package.
1>verbose: Azure Sphere application image package written.
1>verbose: Appending metadata.
1>verbose: Wrote metadata:
1>  Section: Identity
1>    Image Type:           Application
1>    Component ID:         fe9414aa-6e21-4b27-9dd2-7b161506f3e1
1>    Image ID:             47014730-82a0-42d3-9faa-f8eee37780f0
1>  Section: Signature
1>    Signing Type:         ECDsa256
1>    Cert:                 a8d5cc6958f48710140d7a26160fc1cfc31f5df0
1>  Section: Debug
1>    Image Name:           Mt3620Blink1
1>    Built On (UTC):       2019/01/04 8:31:46
1>    Built On (Local):     2019/01/04 17:31:46
1>  Section: Temporary Image
1>    Remove image at boot: False
1>    Under development:    True
1>  Section: ABI Depends
1>    Depends on:           ApplicationRuntime, version 1
1>
1>verbose: Packaging completed successfully.
1>verbose: Output file is at: C:¥AzureSphere¥Mt3620Blink1¥bin¥ARM¥Debug¥Mt3620Blink1.imagepackage
1>Command completed successfully in 00:00:00.9505361.
========== ビルド: 1 正常終了、0 失敗、0 更新不要、0 スキップ ==========
```

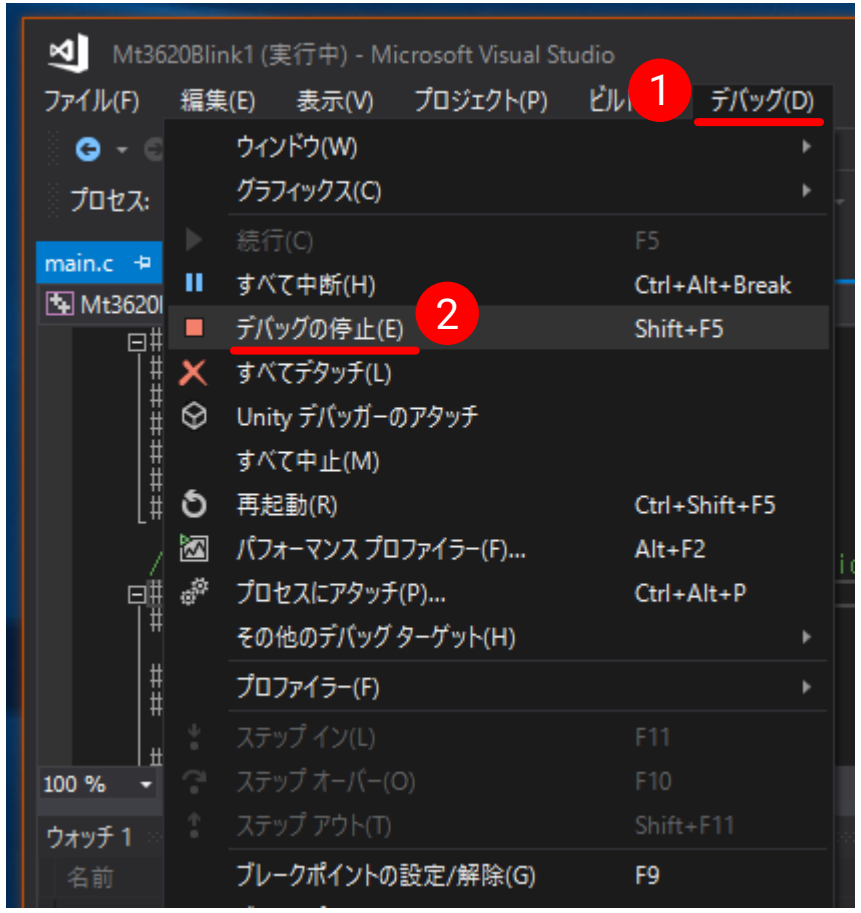
Mt3620Blink1.imagepackage

Lab#1.3 Blink Sampleをデバイスへ デプロイ、実行、停止

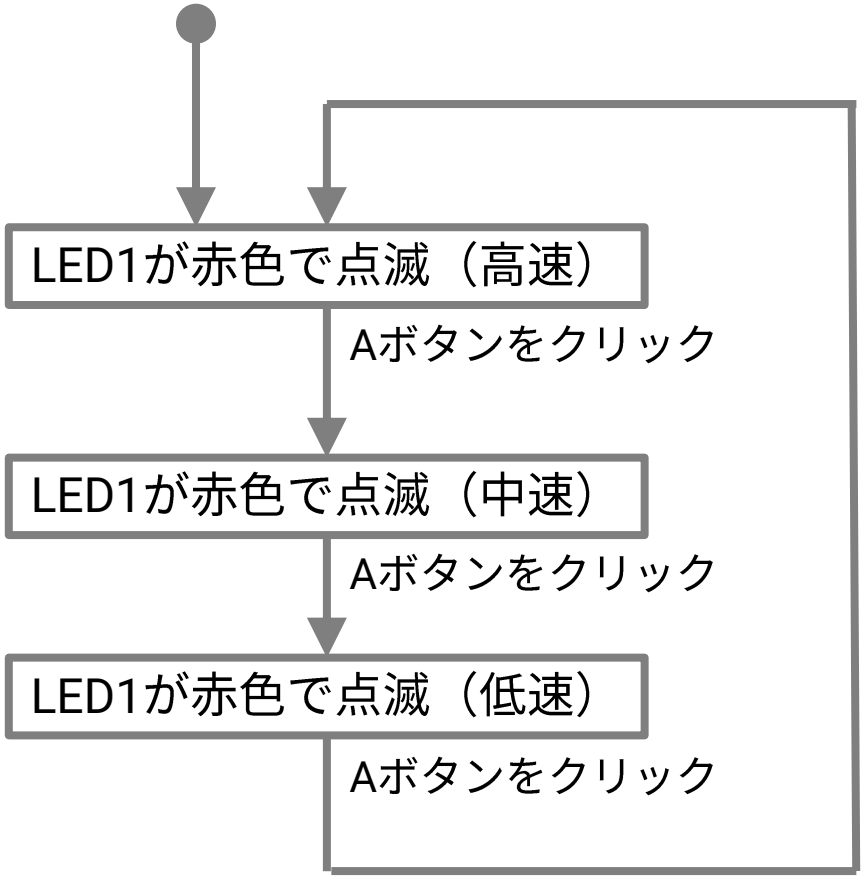
デバッグ > デバッグの開始

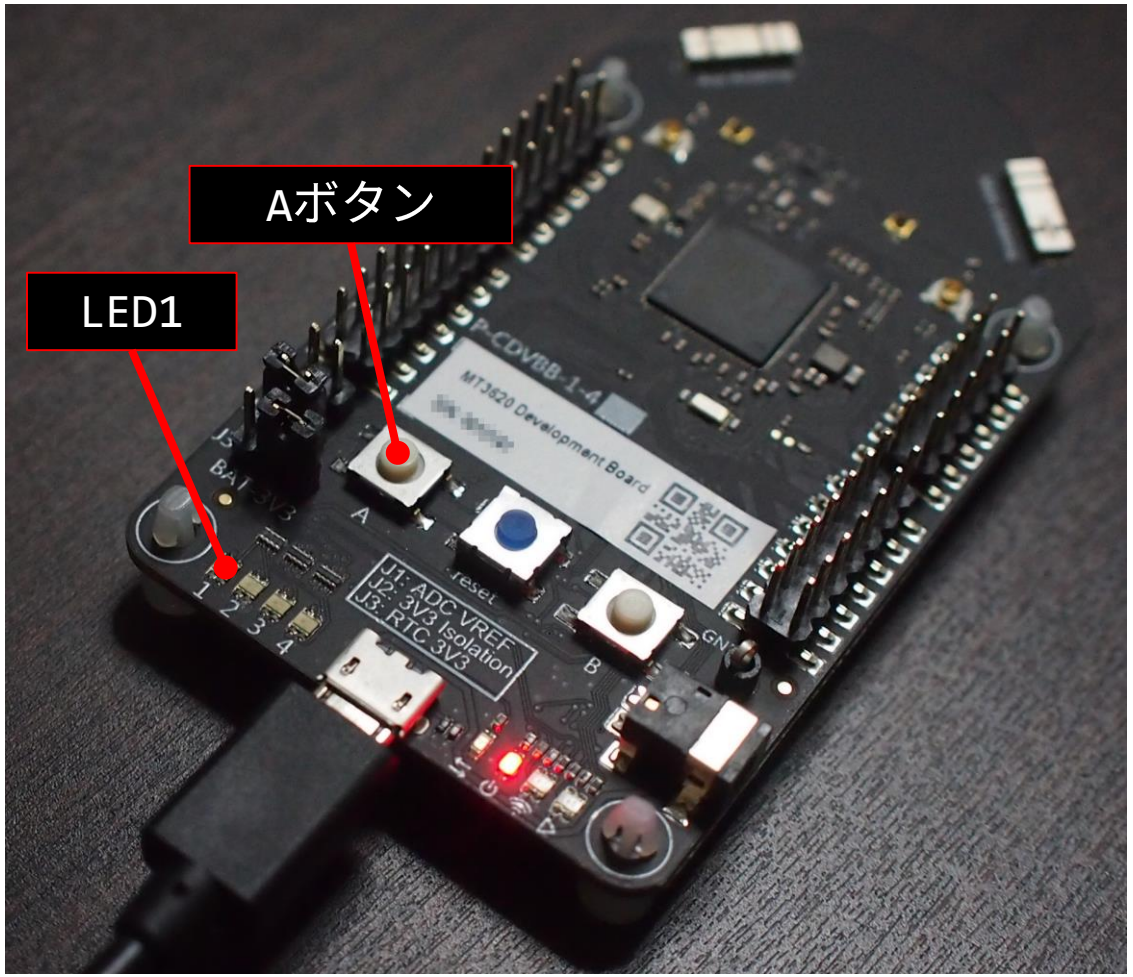


デバッグ > デバッグの停止



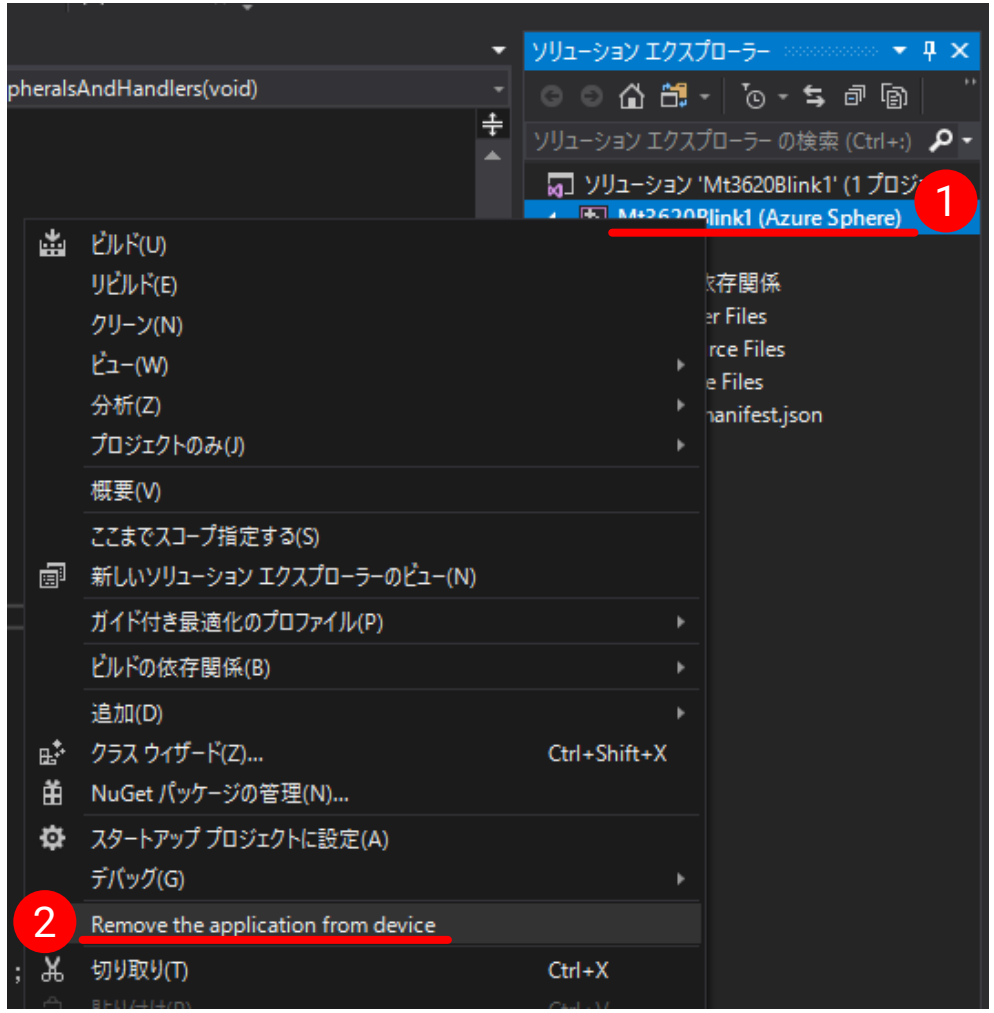
Lab#1.3 Blink Sampleをデバイスへ デプロイ、実行、停止





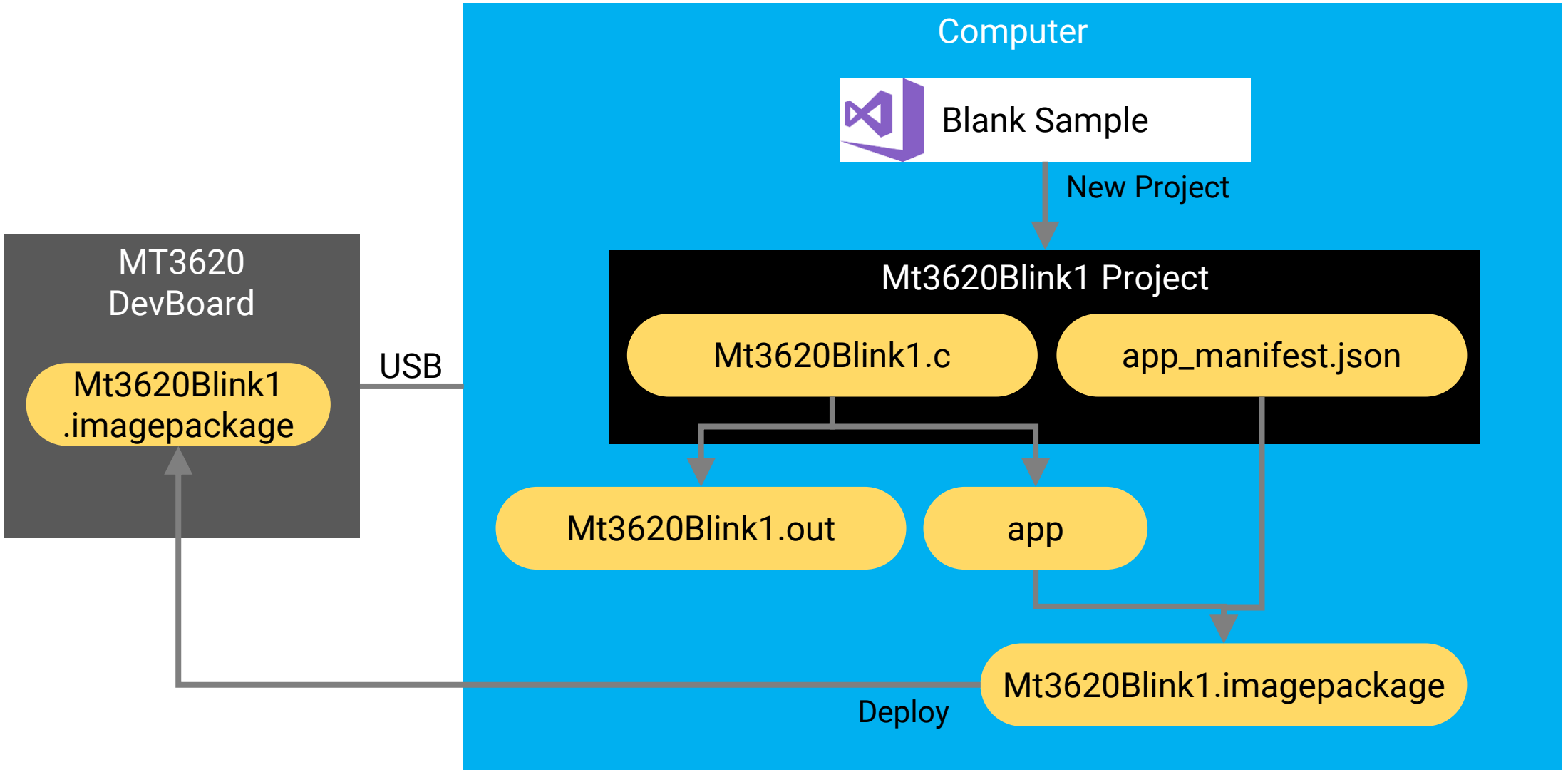
Lab#1.4 デバイスのBlink Sampleを削除

プロジェクト（右クリック） > Remove the application from device



Lab#1 Appendix

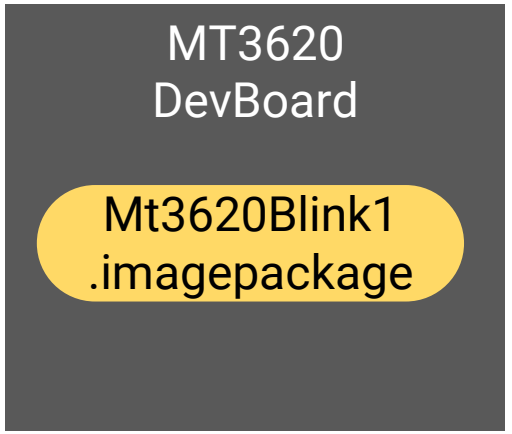
Azure Sphere Build and Deploy



Show Installed Images in MT3620

azsphere device image list-installed

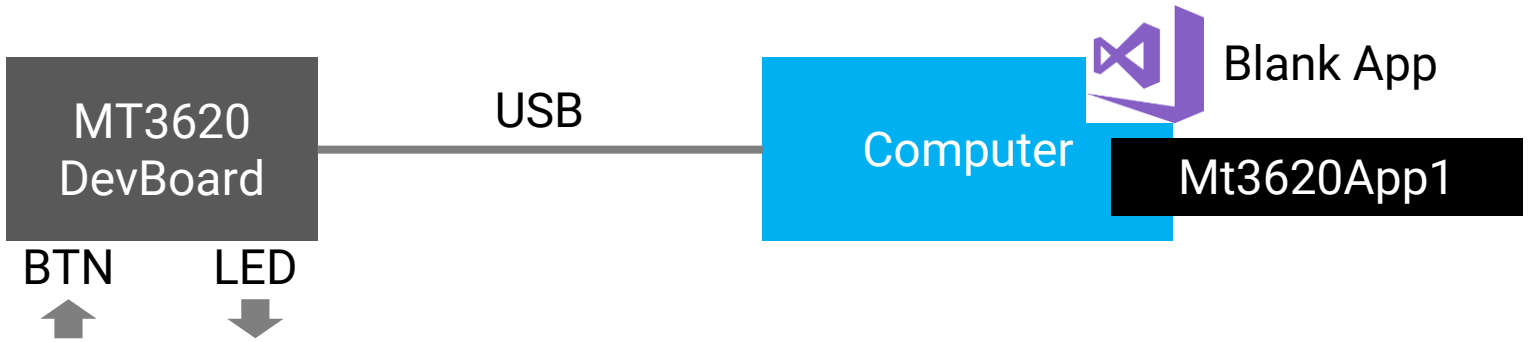
```
C:¥>azsphere device image list-installed
Installed images:
 --> gdbserver
   --> Image type:   Application
   --> Component ID: 8548b129-b16f-4f84-8dbe-d2c847862e78
   --> Image ID:     43d2707f-0bc7-4956-92c1-4a3d0ad91a74
 --> Mt3620Blink1
   --> Image type:   Application
   --> Component ID: fe9414aa-6e21-4b27-9dd2-7b161506f3e1
   --> Image ID:     47014730-82a0-42d3-9faa-f8eee37780f0
Command completed successfully in 00:00:01.4073832.

C:¥>
```



https://docs.microsoft.com/ja-jp/azure-sphere/app-development/app-manifest

Lab#2

Lab#2 Create Digital I/O App from the Blank App



得ること

- Blank Appの構造
- デジタル入力、デジタル出力

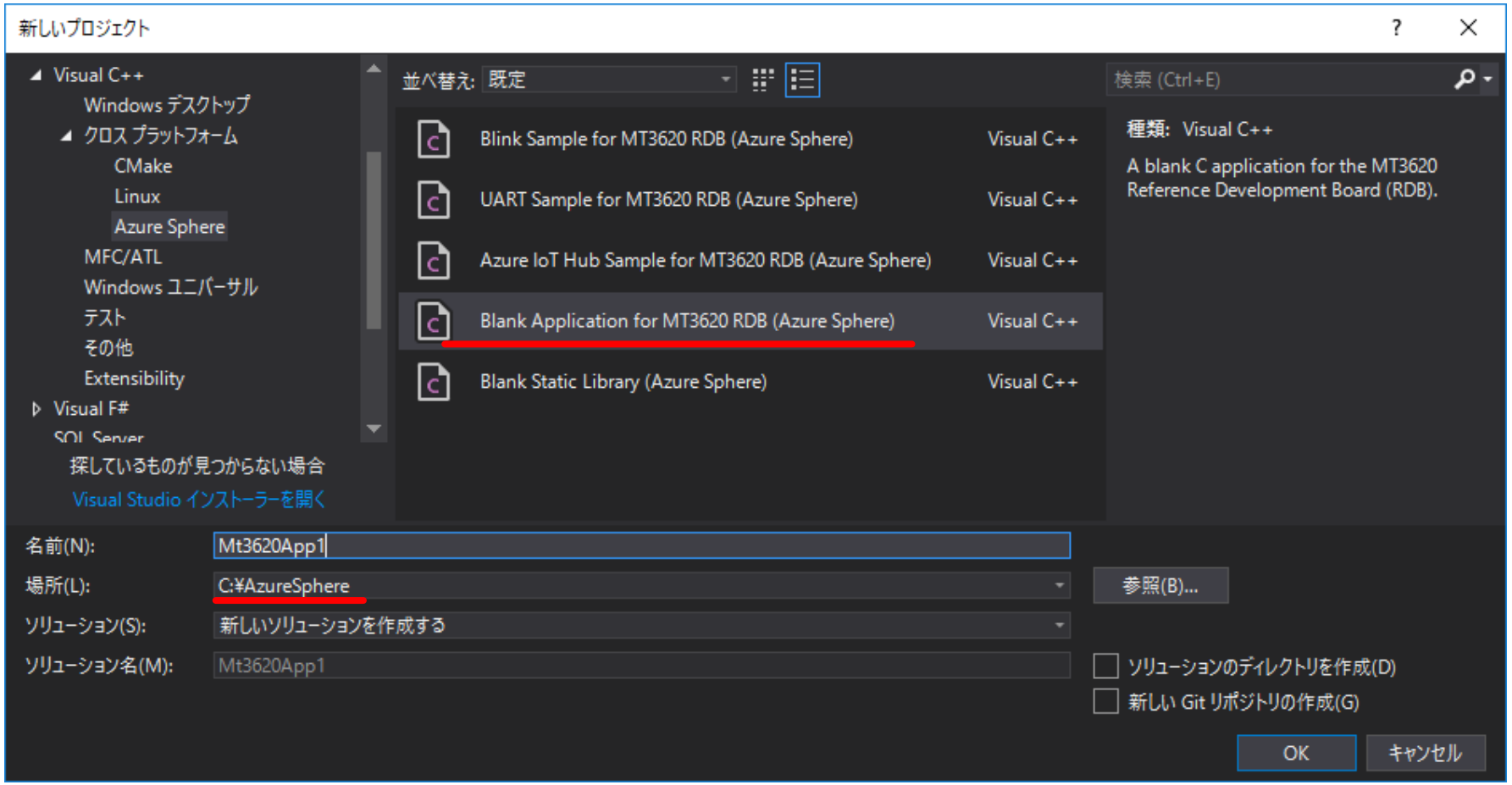
やること

1. Blank Appを新規作成
2. Blank Appのコードを読む
3. Aボタンをログに出力
4. AボタンをLED1に出力
5. 永久ループの周期を変更

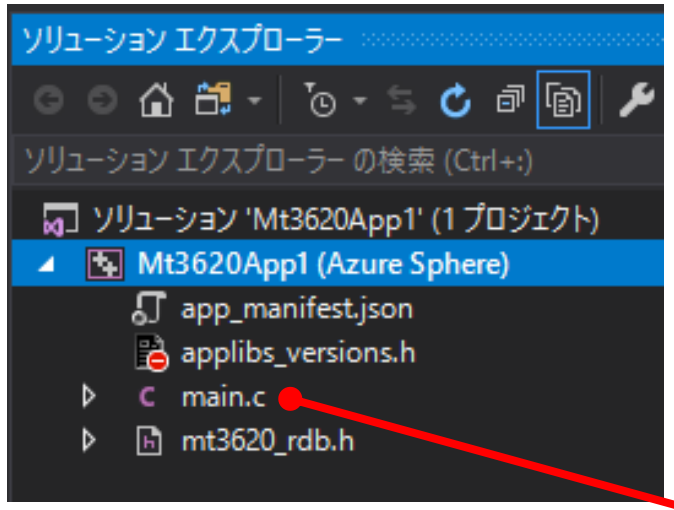
Lab#2.1 Blank Appを新規作成

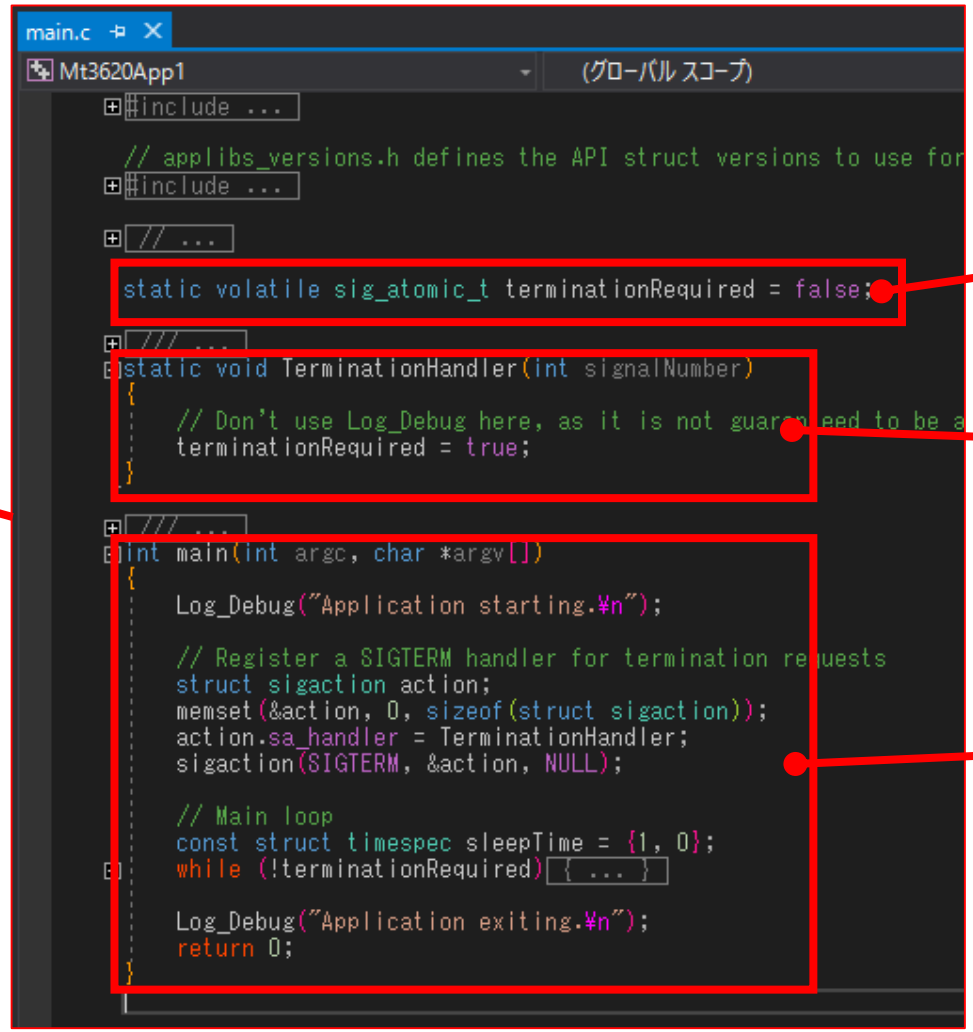
Visual C++ > クロスプラットフォーム > Azure Sphere

Blank Application for MT3620 RDB



Lab#2.2 Blank Appのコードを読む





通知用の変数

プロセス終了時に
呼ばれる関数

最初に呼ばれる関数

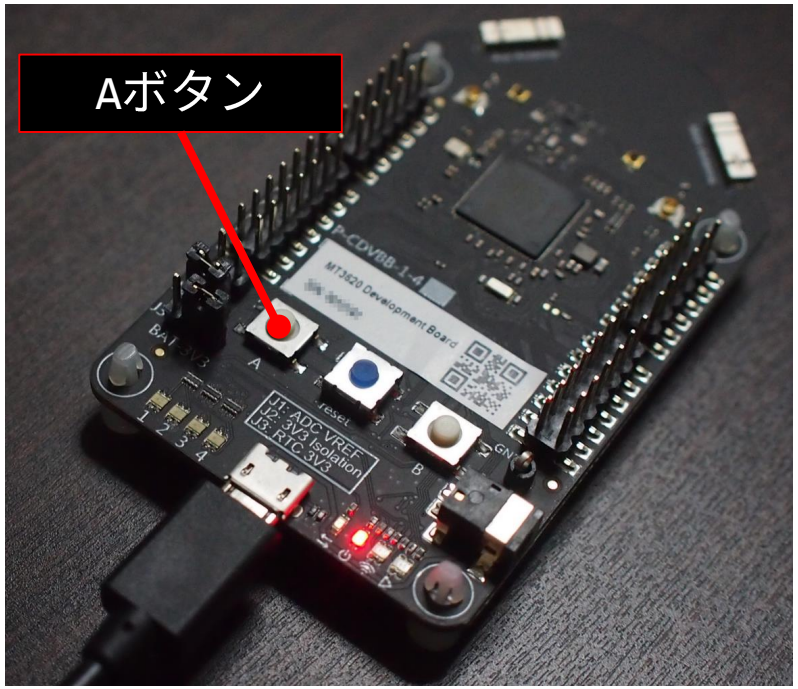
Lab#2.2 Blank Appのコードを読む

```
int main(int argc, char *argv[])
{
    Log_Debug("Application starting.¥n");

    // Register a SIGTERM handler for termination requests
    struct sigaction action;
    memset(&action, 0, sizeof(struct sigaction));
    action.sa_handler = TerminationHandler;
    sigaction(SIGTERM, &action, NULL);

    // Main loop
    const struct timespec sleepTime = {1, 0};
    while (!terminationRequired) {
        Log_Debug("Hello world!¥n");
        nanosleep(&sleepTime, NULL);
    }

    Log_Debug("Application exiting.¥n");
    return 0;
}
```

プロセス終了時にTerminationHandler関数を呼ぶよう設定

terminationRequiredがfalseの間、永久ループ

Lab#2.3 Aボタンをログに出力



The two user buttons (A and B) are connected to the GPIO pins listed in the following table. Note that these GPIO inputs are pulled high via 4.7K resistors. Therefore, the default input state of these GPIOs is high; when a user presses a button, the GPIO input is low.

通常はHIGHで、ボタンを押すとLOW

https://docs.microsoft.com/ja-jp/azure-sphere/hardware/mt3620-user-guide#user-buttons

Lab#2.3 Aボタンをログに出力

main.c: インクルードファイル

```
#include <assert.h>
#include <applibs/gpio.h>
```

main.c: GPIO初期化

```
int buttonFd = GPIO_OpenAsInput(MT3620_RDB_BUTTON_A);
assert(buttonFd >= 0);
```

永久ループの外側

main.c: Aボタンの状態をログに出力

```
GPIO_Value_Type buttonValue;
int ret = GPIO_GetValue(buttonFd, &buttonValue);
assert(ret == 0);
if (buttonValue == GPIO_Value_High)
        Log_Debug("Button A is OFF¥n");
else
        Log_Debug("Button A is ON¥n");

nanosleep(&sleepTime, NULL);
```

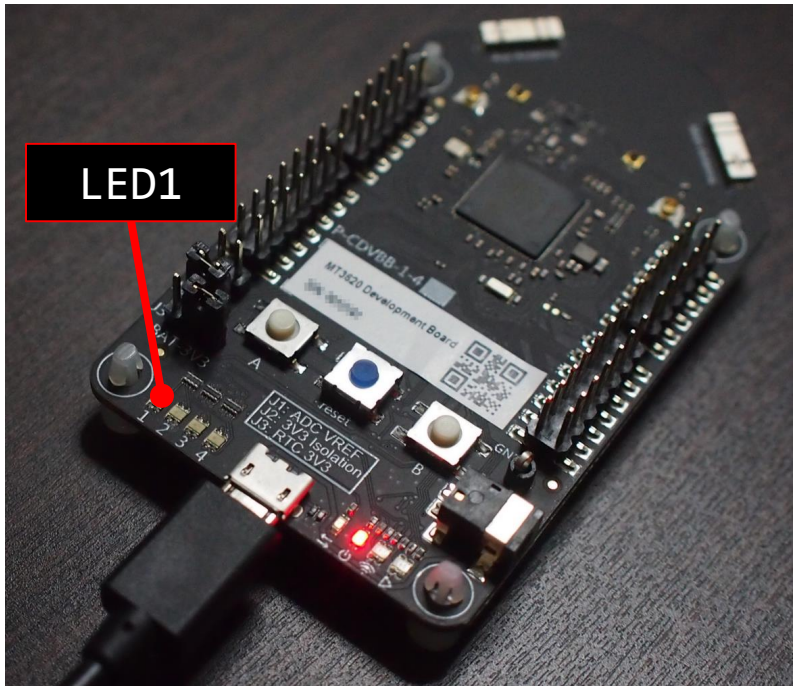
永久ループの内側

Lab#2.3 Aボタンをログに出力

app_manifest.json

```
{
  "SchemaVersion": 1,
  "Name" : "Mt3620App1",
  "ComponentId" : “xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
  "EntryPoint": "/bin/app",
  "CmdArgs": [],
  "Capabilities": {
    "AllowedConnections": [],
    "AllowedTcpServerPorts": [],
    "AllowedUdpServerPorts": [],
    "Gpio": [ 12 ],
    "Uart": [],
    "WifiConfig": false,
    "NetworkConfig": false,
    "SystemTime": false
  }
}
```

Lab#2.4 AボタンをLED1に出力



The development board includes four RGB user LEDs, labelled 1-4. The LEDs connect to MT3620 GPIOs as listed in the table. The common anode of each RGB LED is tied high; therefore, driving the corresponding GPIO low illuminates the LED.

GPIOをLOWにするとLEDが点灯

https://docs.microsoft.com/ja-jp/azure-sphere/hardware/mt3620-user-guide#user-leds

Lab#2.4 AボタンをLED1に出力

main.c: GPIO初期化

```
int ledFd = GPIO_OpenAsOutput(MT3620_RDB_LED1_RED, GPIO_OutputMode_PushPull,
                                                    GPIO_Value_High);
assert(ledFd >= 0);
```

main.c: Aボタンの状態をLED1に出力

```
int ret2 = GPIO_SetValue(ledFd,
               buttonValue == GPIO_Value_High ? GPIO_Value_High : GPIO_Value_Low);
assert(ret2 == 0);
```

app_manifest.json

```
"Gpio": [ 12, 8 ],
```

Lab#2.4 永久ループの周期を変更

```c
struct timespec {
    time_t tv_sec;
    long tv_nsec;
};
```

秒

ナノ秒（10^{-9}）

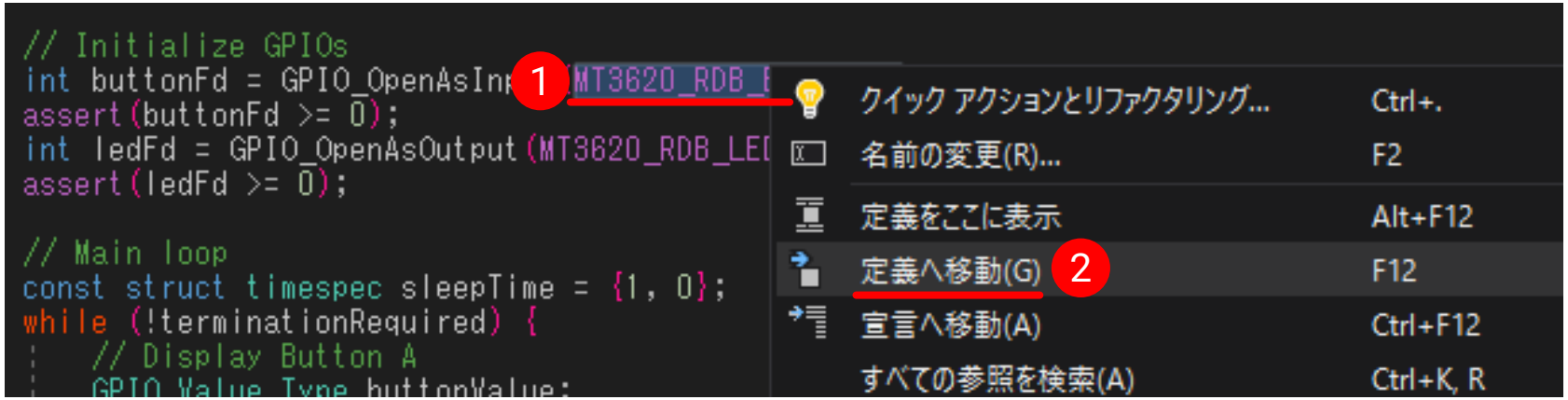
```c
int main(int argc, char *argv[])
{
    ...
    const struct timespec sleepTime = {1, 0};
    while (!terminationRequired) {
        ...

        nanosleep(&sleepTime, NULL);
    }
    ...
}
```

https://linuxjm.osdn.jp/html/LDP_man-pages/man2/nanosleep.2.html

Lab#2 Appendix

How to find GPIO number ?

定義へ移動



main.c

```
int buttonFd = GPIO_OpenAsInput(MT3620_RDB_BUTTON_A);
assert(buttonFd >= 0);
```

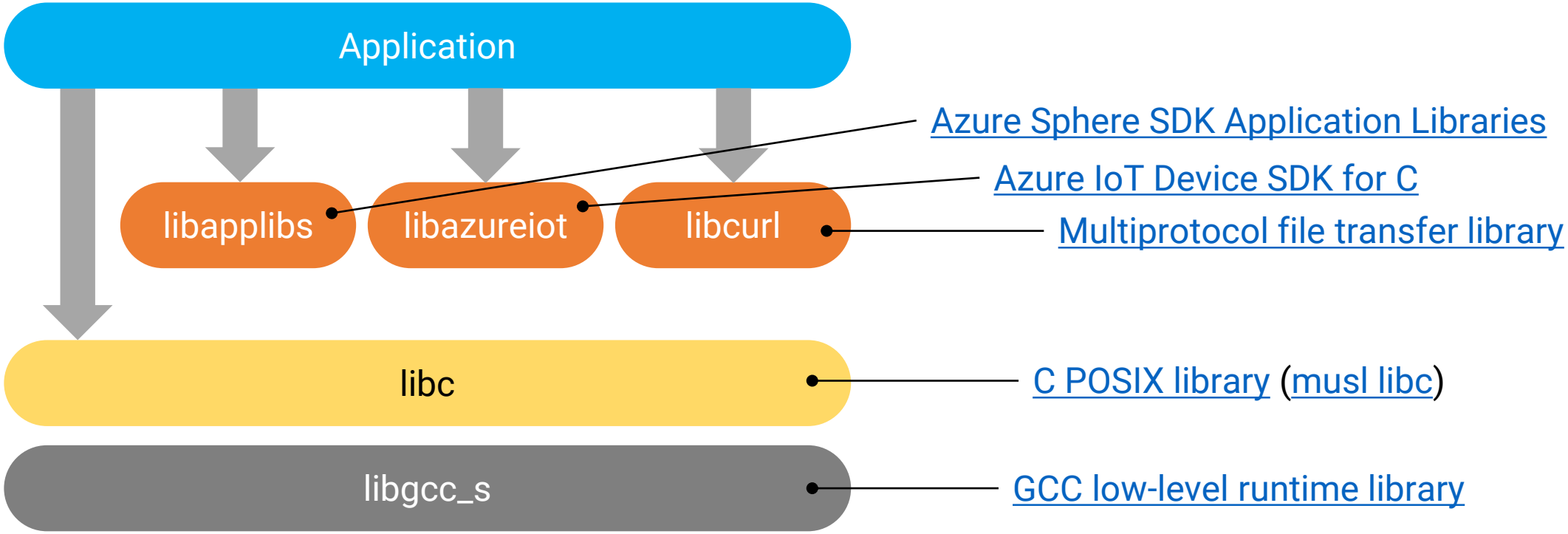
mt3620_rdb.h

```
/// <summary>Button A is GPIO12.</summary>
#define MT3620_RDB_BUTTON_A MT3620_GPIO12
```

mt3620_gpios.h

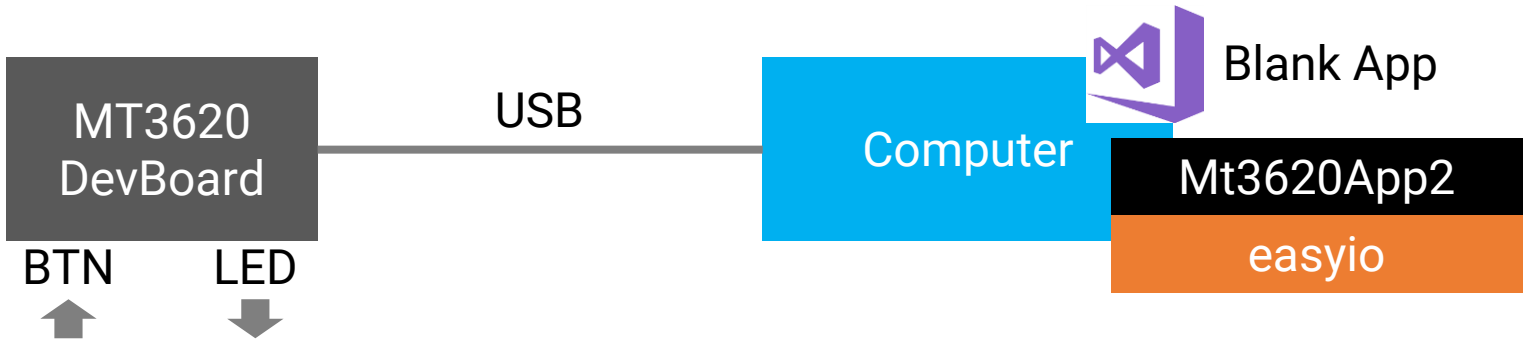
```
#define MT3620_GPIO12 ((GPIO_Id)12)
```

Libraries in Azure Sphere SDK



Lab#3

Lab#3 To use Static Library



得ること

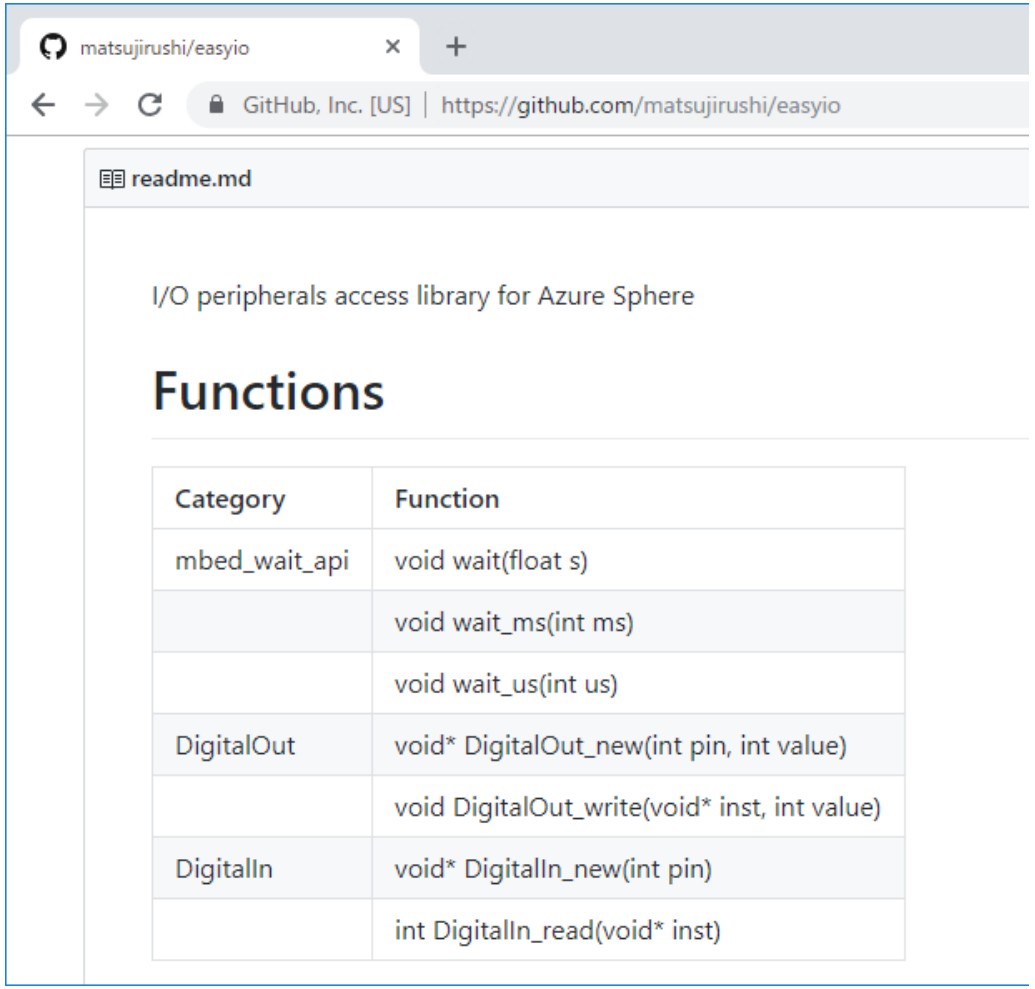
- ライブラリの利用

やること

1. easyioをクローン
2. Blank Appを新規作成
3. easyioプロジェクトを追加
4. easyioプロジェクトを参照
5. AボタンをLED1に出力

Lab#3 easyio

https://github.com/matsujirushi/easyio

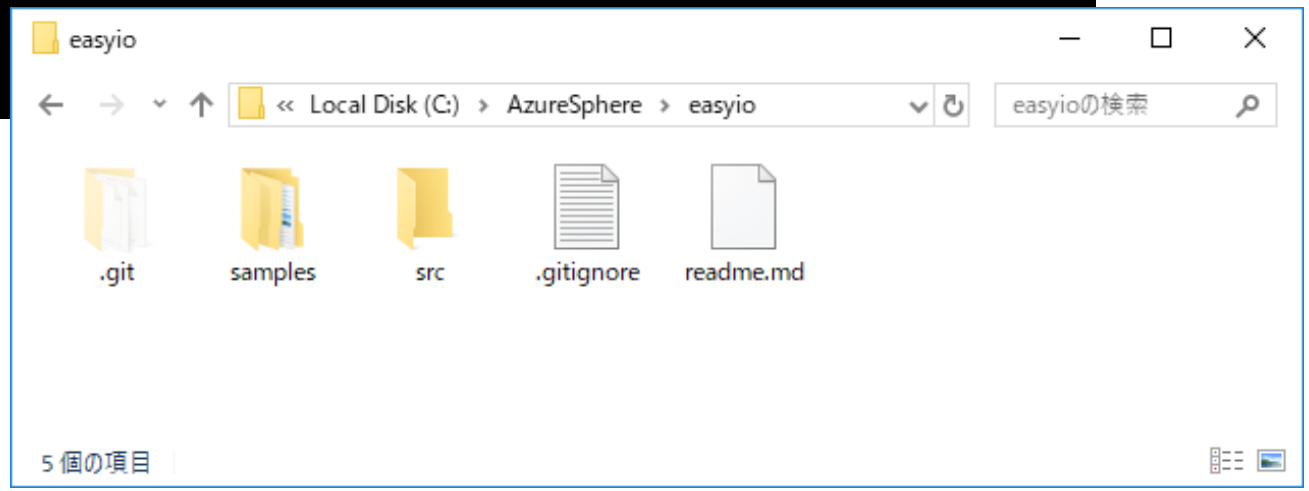


I/O操作のコーディングの手間を減らすライブラリ

Lab#3.1 easyioをクローン

cd C:¥AzureSphere
git clone https://github.com/matsujirushi/easyio.git

```
C:¥AzureSphere>git clone https://github.com/matsujirushi/easyio.git
Cloning into 'easyio'...
remote: Enumerating objects: 25, done.
remote: Counting objects: 100% (25/25), done.
remote: Compressing objects: 100% (21/21), done.
remote: Total 25 (delta 2), reused 25 (delta 2), pack-reused 0
Unpacking objects: 100% (25/25), done.

C:¥AzureSphere>
```

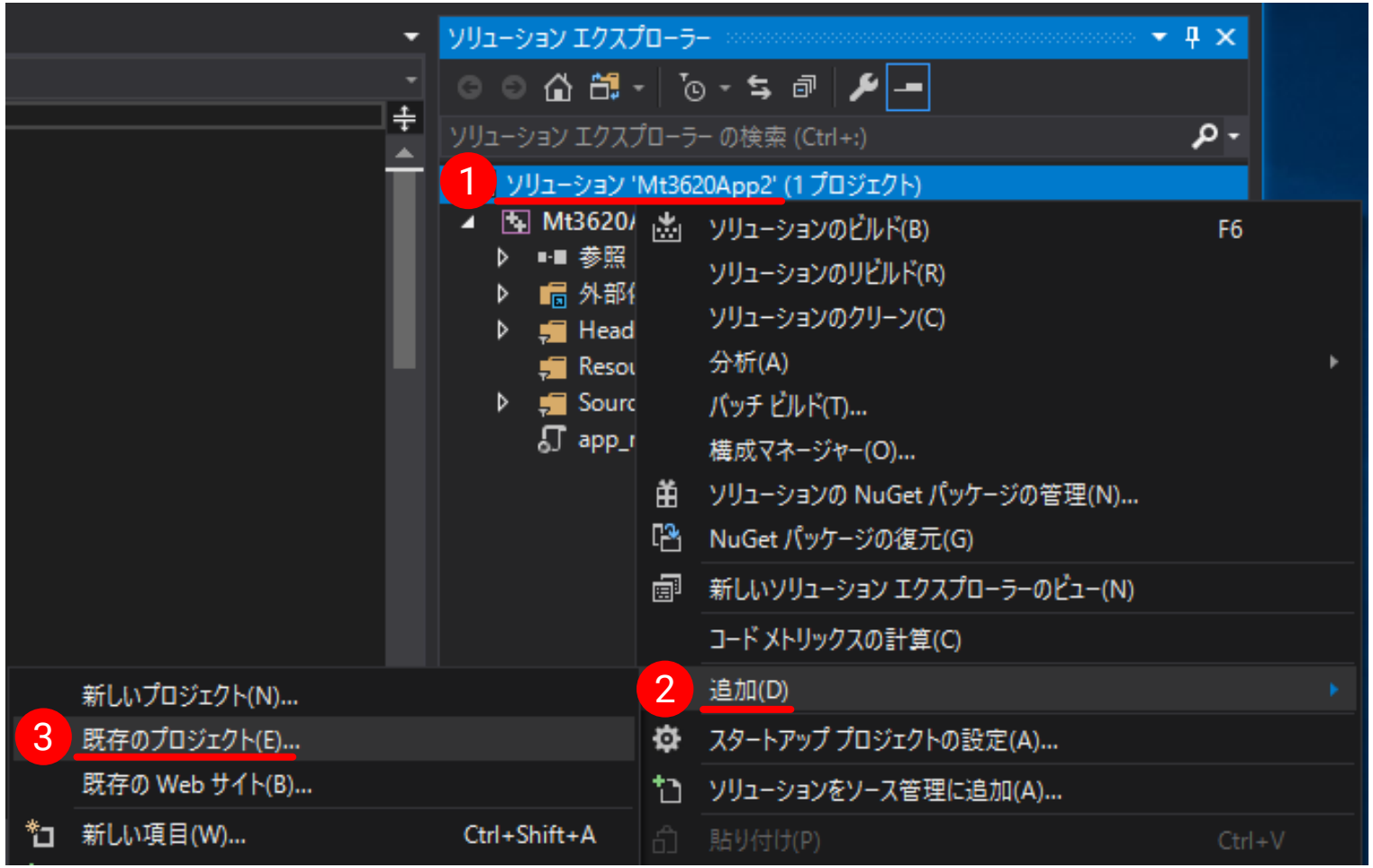


Lab#3.2 Blank Appを新規作成

（省略）

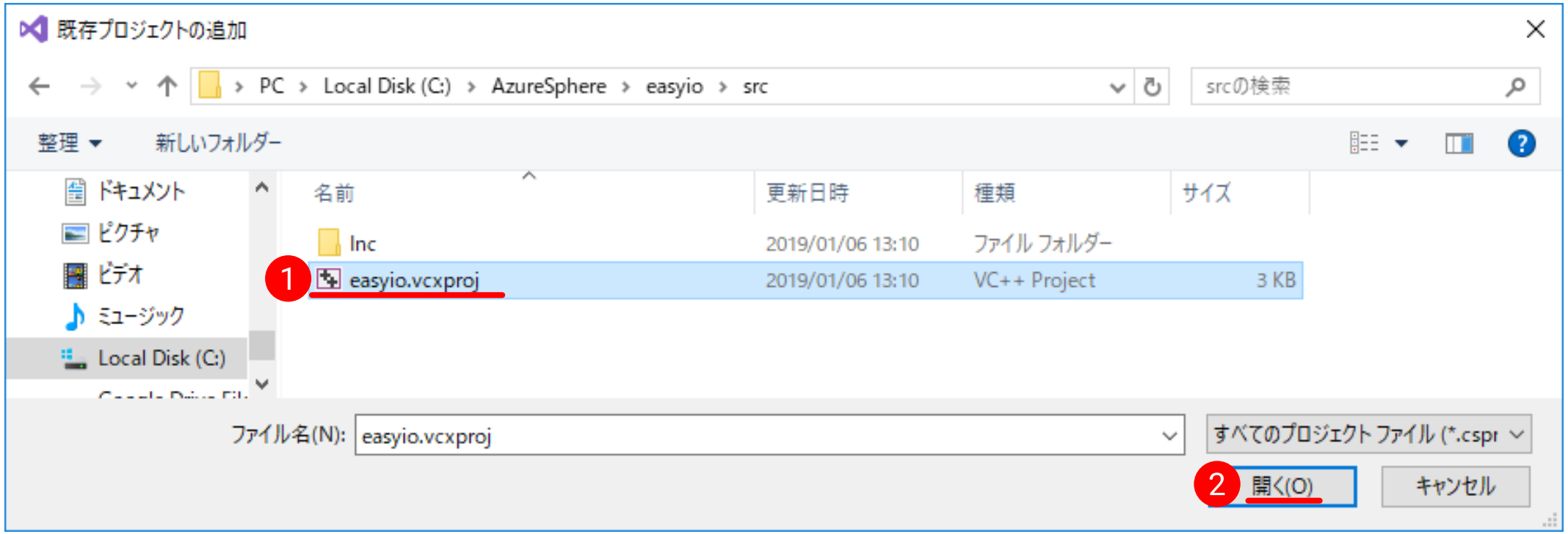
Lab#3.3 easyioプロジェクトを追加

ソリューション（右クリック） > 追加 > 既存のプロジェクト



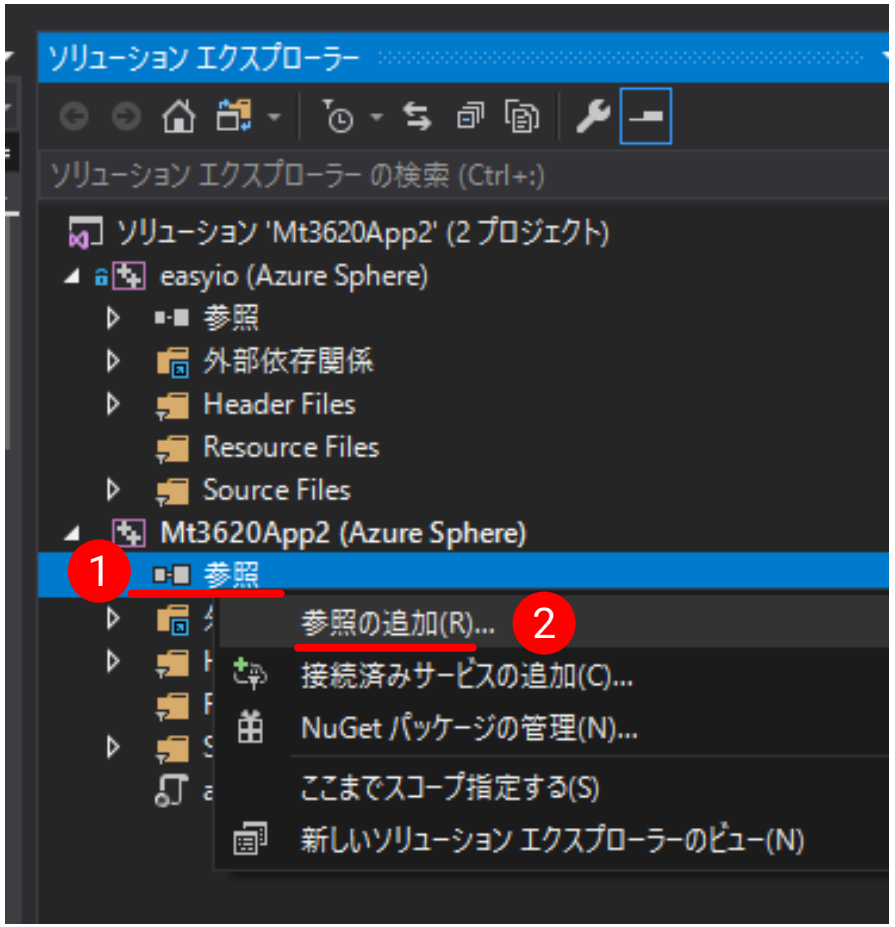
Lab#3.3 easyioプロジェクトを追加

easyio/src/easyio.vcxproj



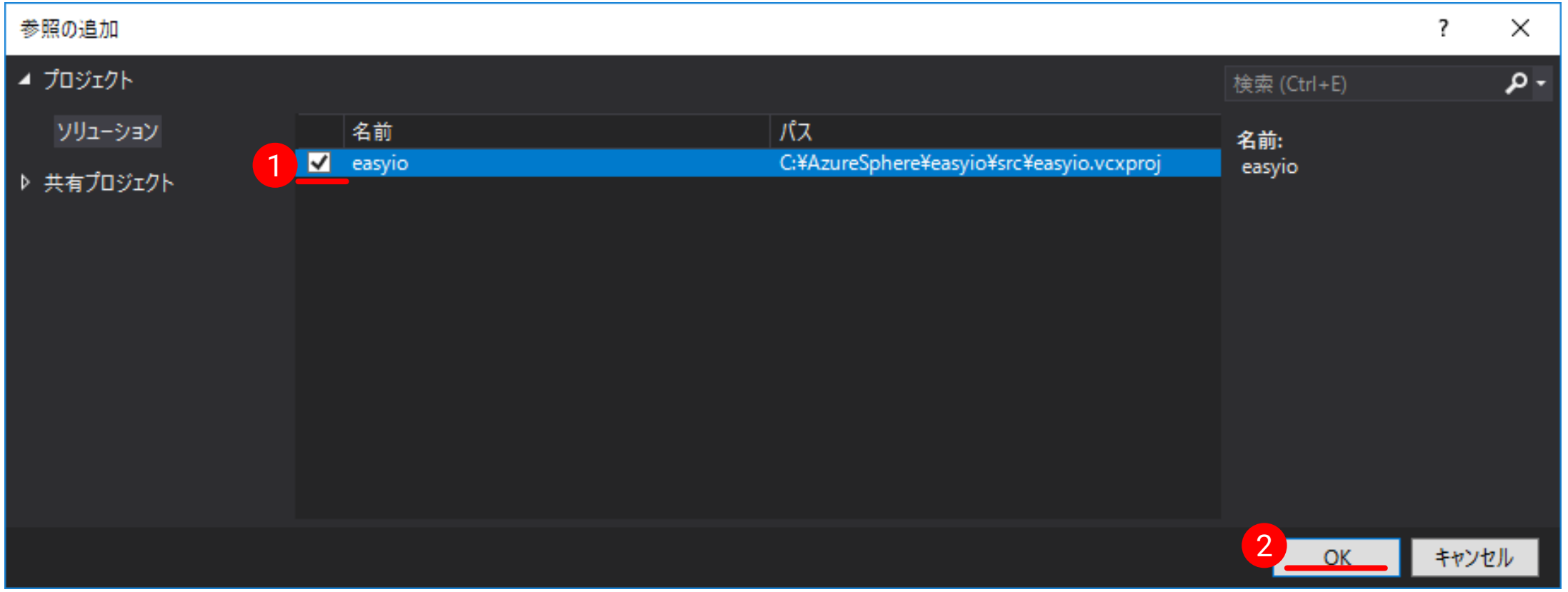
Lab#3.4 easyioプロジェクトを参照

Mt3620App2の参照（右クリック） > 参照の追加



Lab#3.4 easyioプロジェクトを参照

easyioをチェック



Lab#3.5 AボタンをLED1に出力

main.c: インクルードファイル

```
#include <easyio.h>
```

main.c: GPIO初期化

```
void* button = DigitalIn_new(MT3620_RDB_BUTTON_A);
void* led = DigitalOut_new(MT3620_RDB_LED1_RED, 1);
```

永久ループの外側

main.c: Aボタンの状態をLED1に出力

```
int buttonValue = DigitalIn_read(button);
DigitalOut_write(led, buttonValue == 1 ? 1 : 0);
wait_ms(1000);
```

永久ループの内側

app_manifest.json

（省略）

Lab#3 Appendix

How to create the library ?

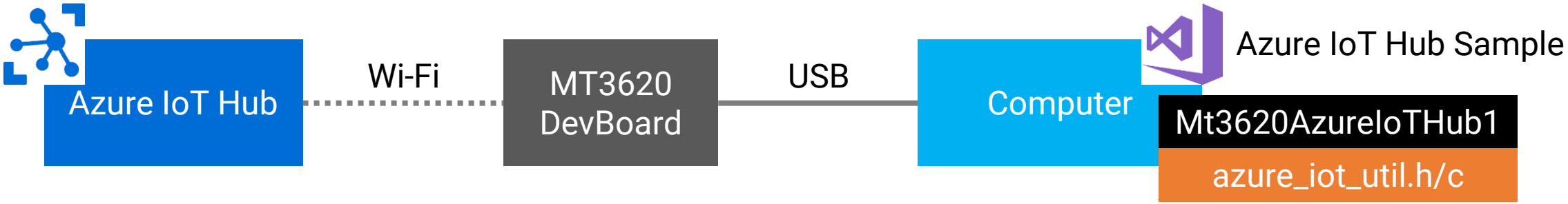
https://qiita.com/matsujirushi/items/dc246c200f11e2c1d485

How to use the library ?

https://qiita.com/matsujirushi/items/2b8af03058b6a84cced2

Lab#4

Lab#4 The Azure IoT Hub Sample



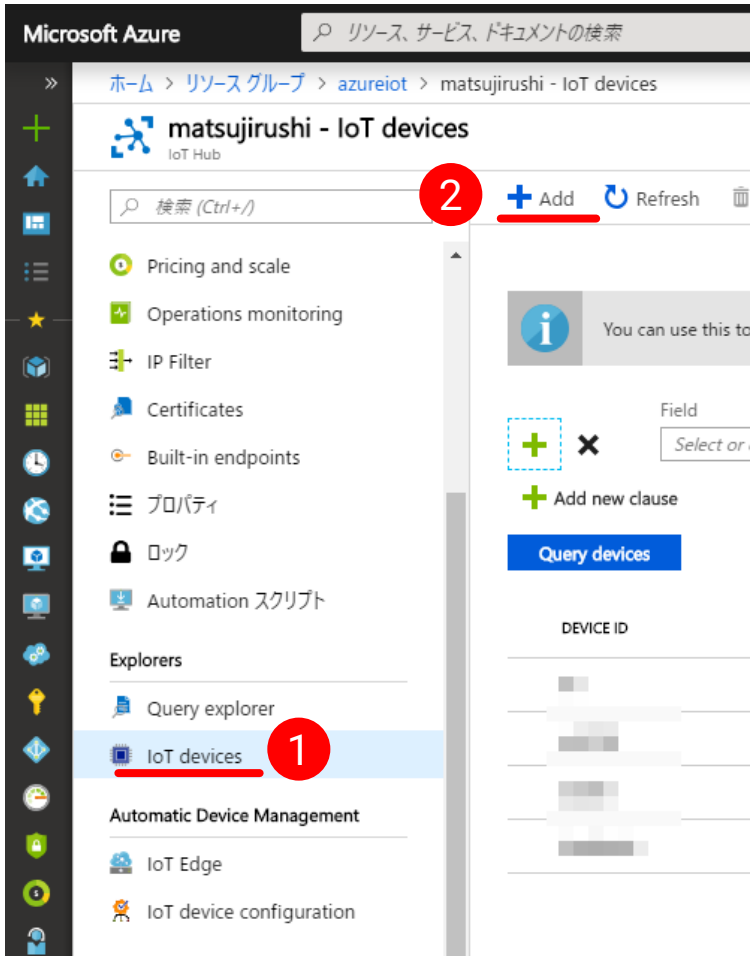
得ること

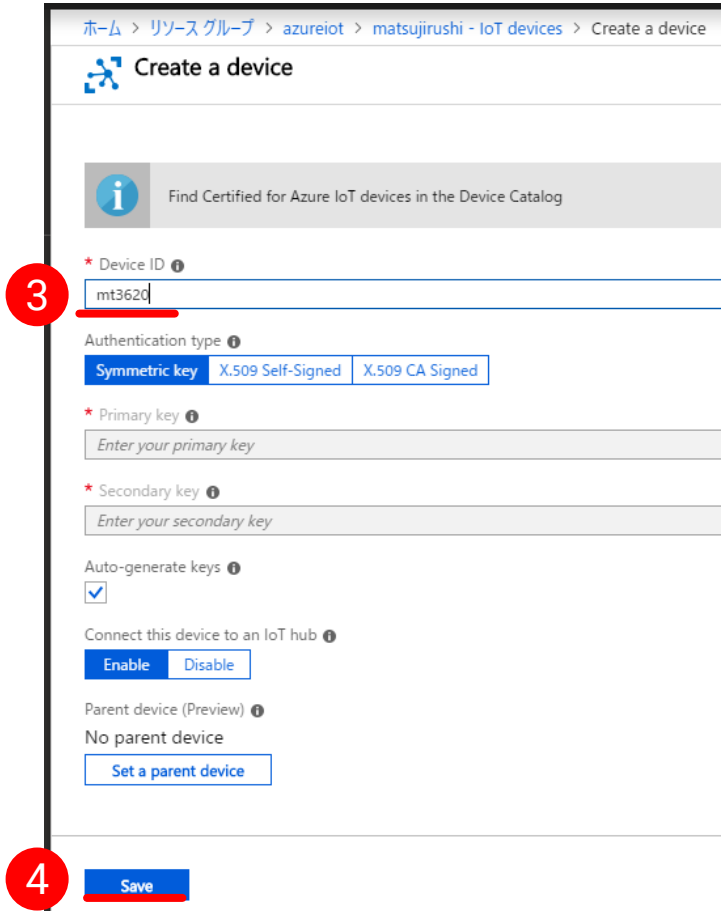
- Azure IoT Hubへの接続方法

やること

1. IoT Hubにデバイスを作成
2. Wi-Fi接続を設定
3. Azure IoT Hub Sampleを新規作成
4. Azure IoT接続コードを生成
5. Azure IoT Hub Sampleを実行

Lab#4.1 IoT Hubにデバイスを作成





IoT Hubの作り方は、下記URLのCreate an IoT Hubを参照してください。
https://docs.microsoft.com/ja-jp/azure/iot-hub/iot-hub-create-through-portal

Lab#4.2 Wi-Fi接続を設定

Wi-Fi接続を追加

azsphere device wifi add -s (SSID) -k (PASSWORD)

Wi-Fi接続の一覧表示

azsphere device wifi list

```
C:¥>azsphere device wifi add -s aterm-3b1234-g -k 123456

C:¥>azsphere device wifi list
Network list:

ID                  : 0
SSID                : aterm-3b1234-g
Configuration state : enabled
Connection state    : connected
Security state      : psk

Command completed successfully in 00:00:01.2361330.

C:¥>
```

Lab#4.3 Azure IoT Hub Sampleを新規作成

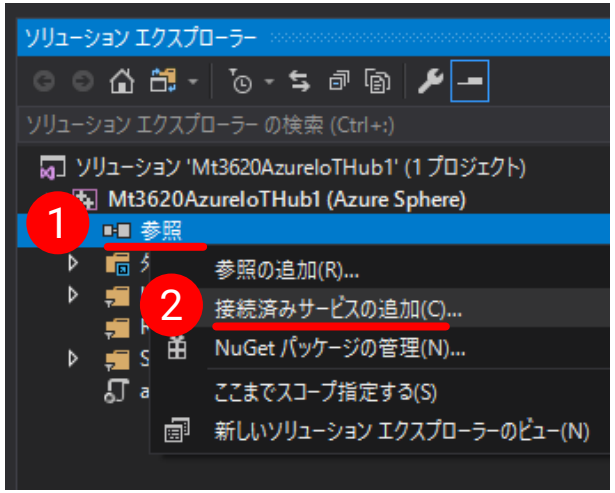
Visual C++ > クロスプラットフォーム > Azure Sphere

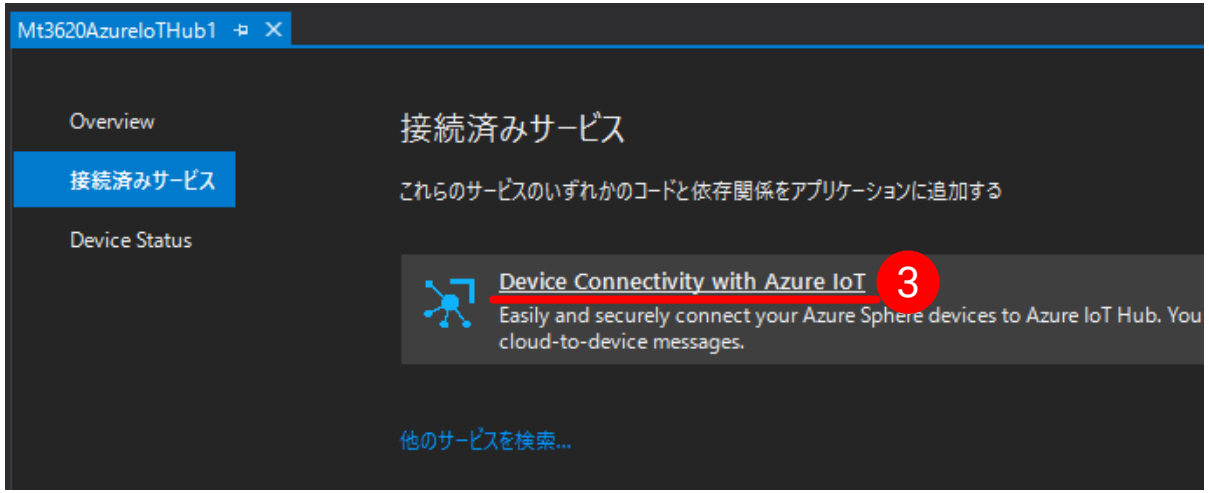
Azure IoT Hub Sample for MT3620 RDB

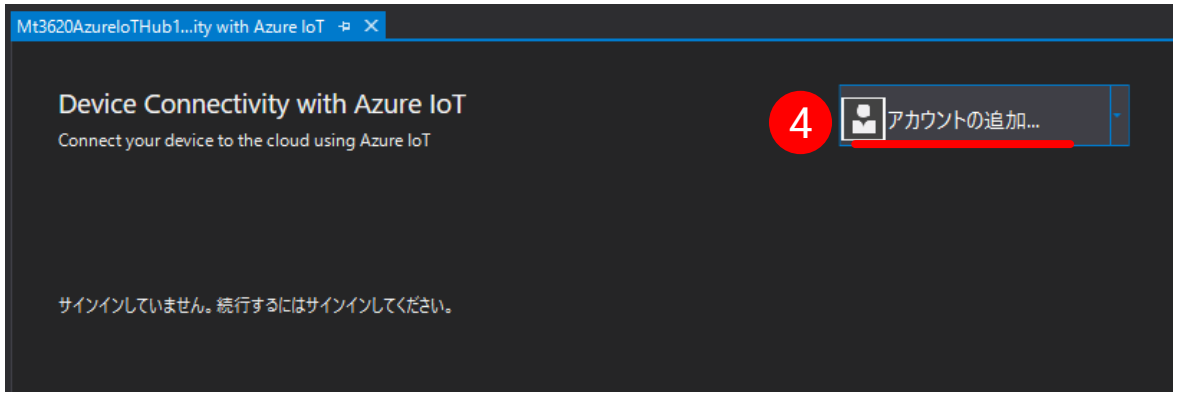
（省略）

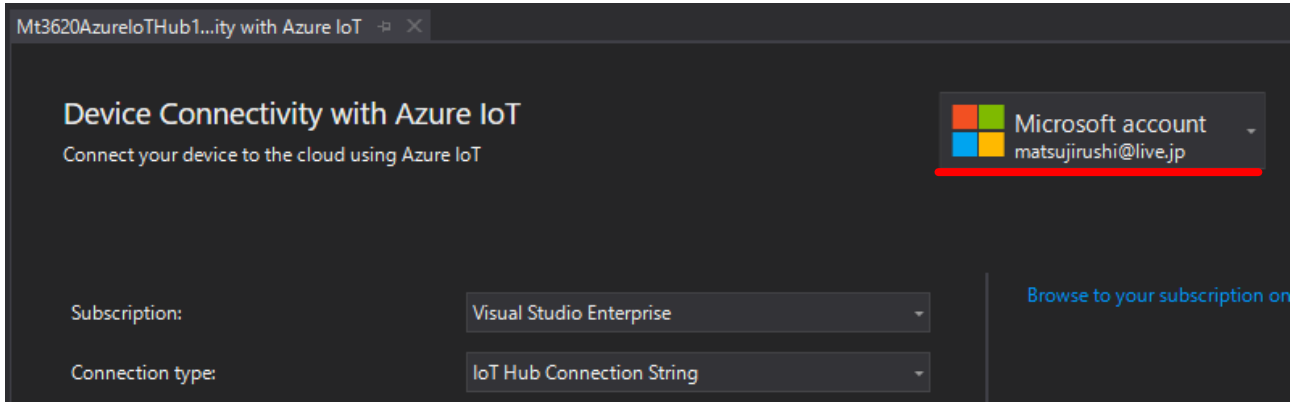
Lab#4.4 Azure IoT接続コードを生成

参照（右クリック） > 接続済みサービスの追加

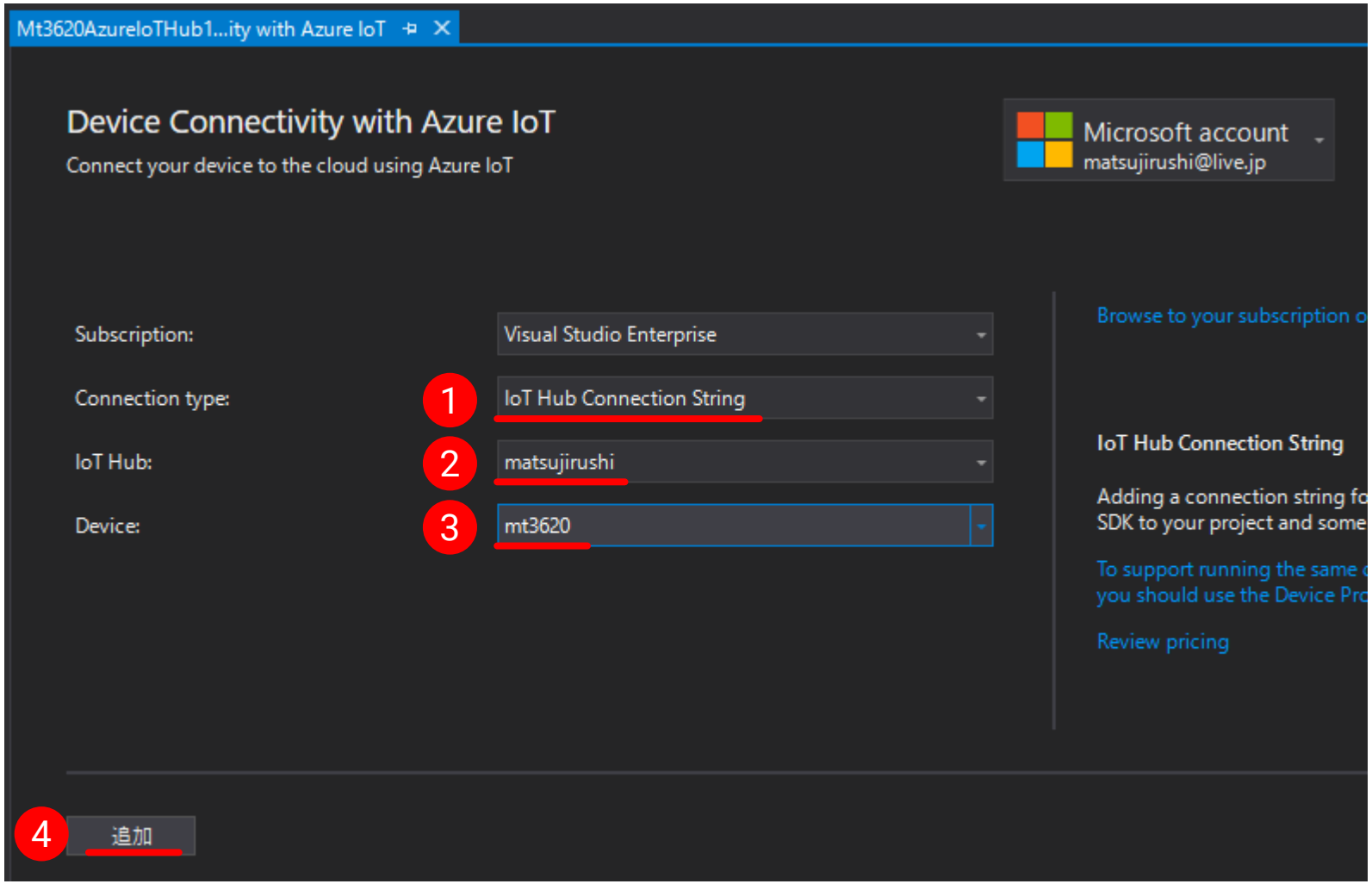


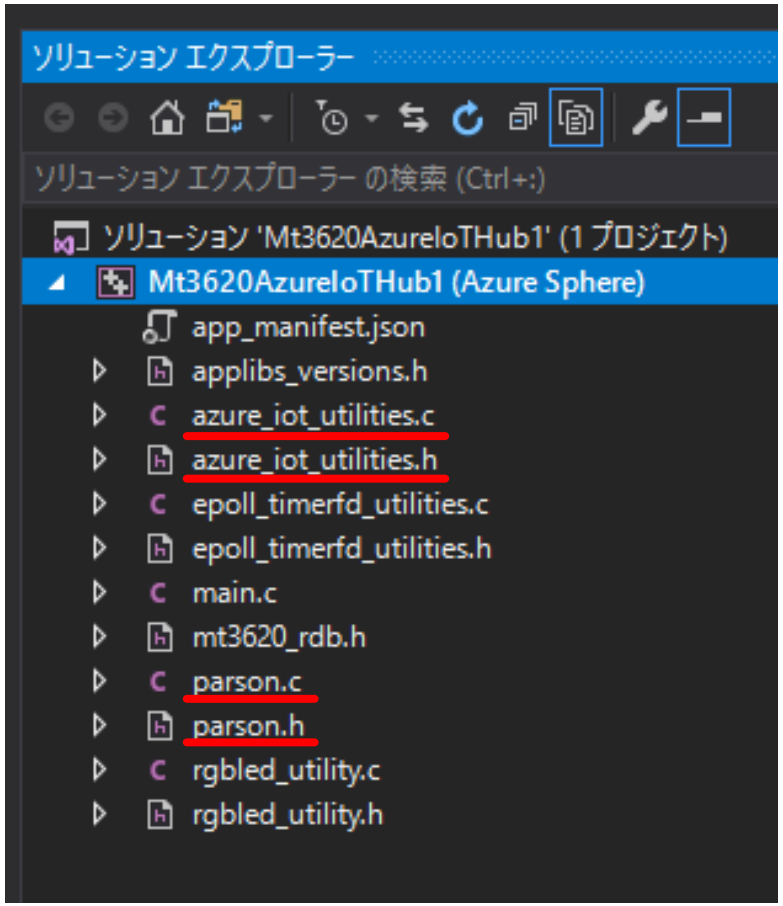






Lab#4.4 Azure IoT接続コードを生成





Lab#4.5 Azure IoT Hub Sampleを実行

Aボタン

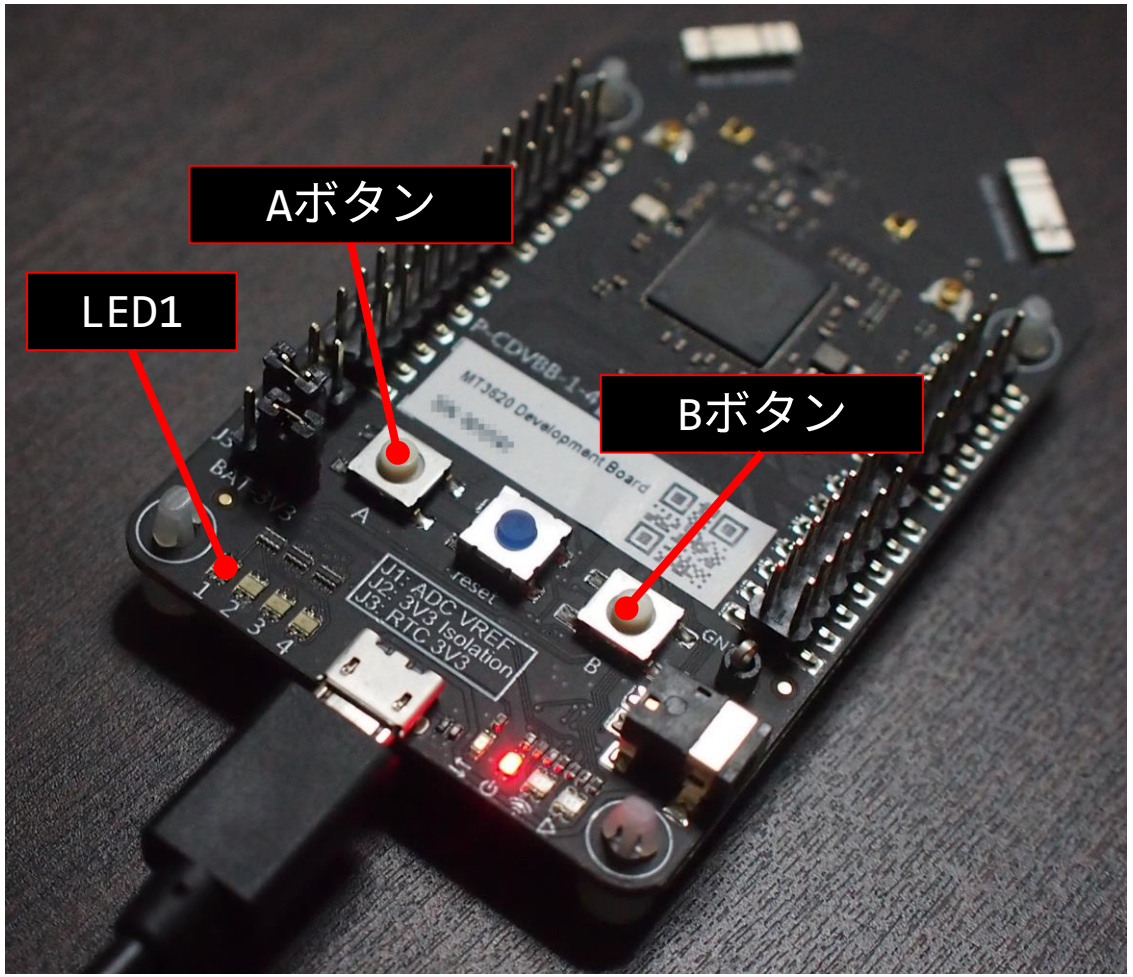
- LED1の点滅スピード変更
- 点滅スピードをDeviceTwinに通知

Bボタン

- D2Cメッセージを送信

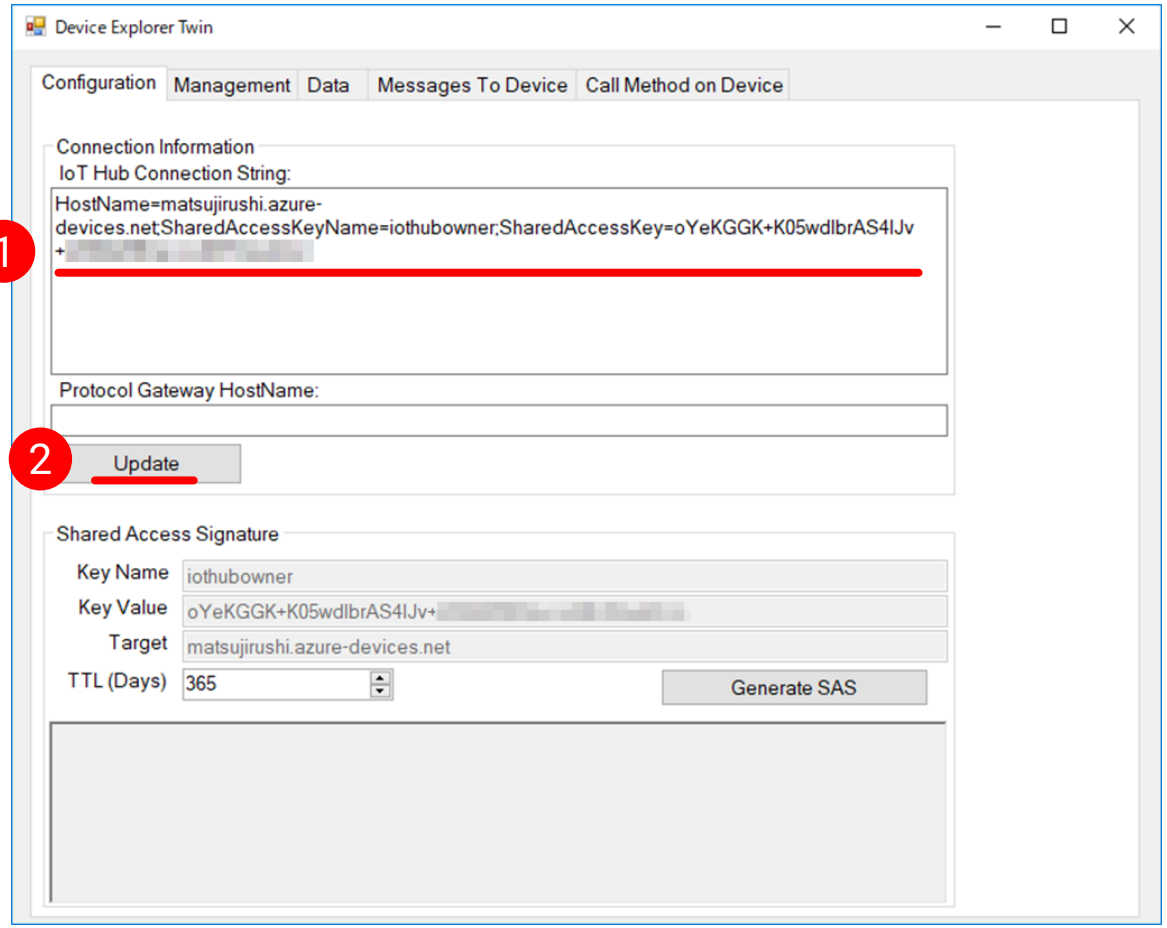
DirectMethod

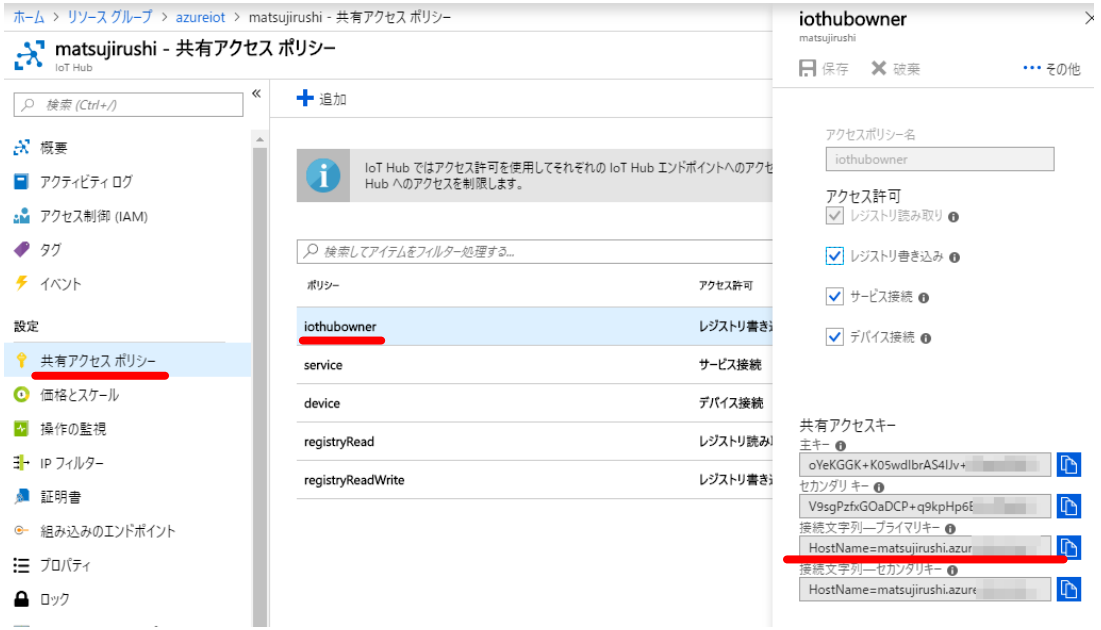
- LED1の色を変更



Lab#4.5 Azure IoT Hub Sampleを実行

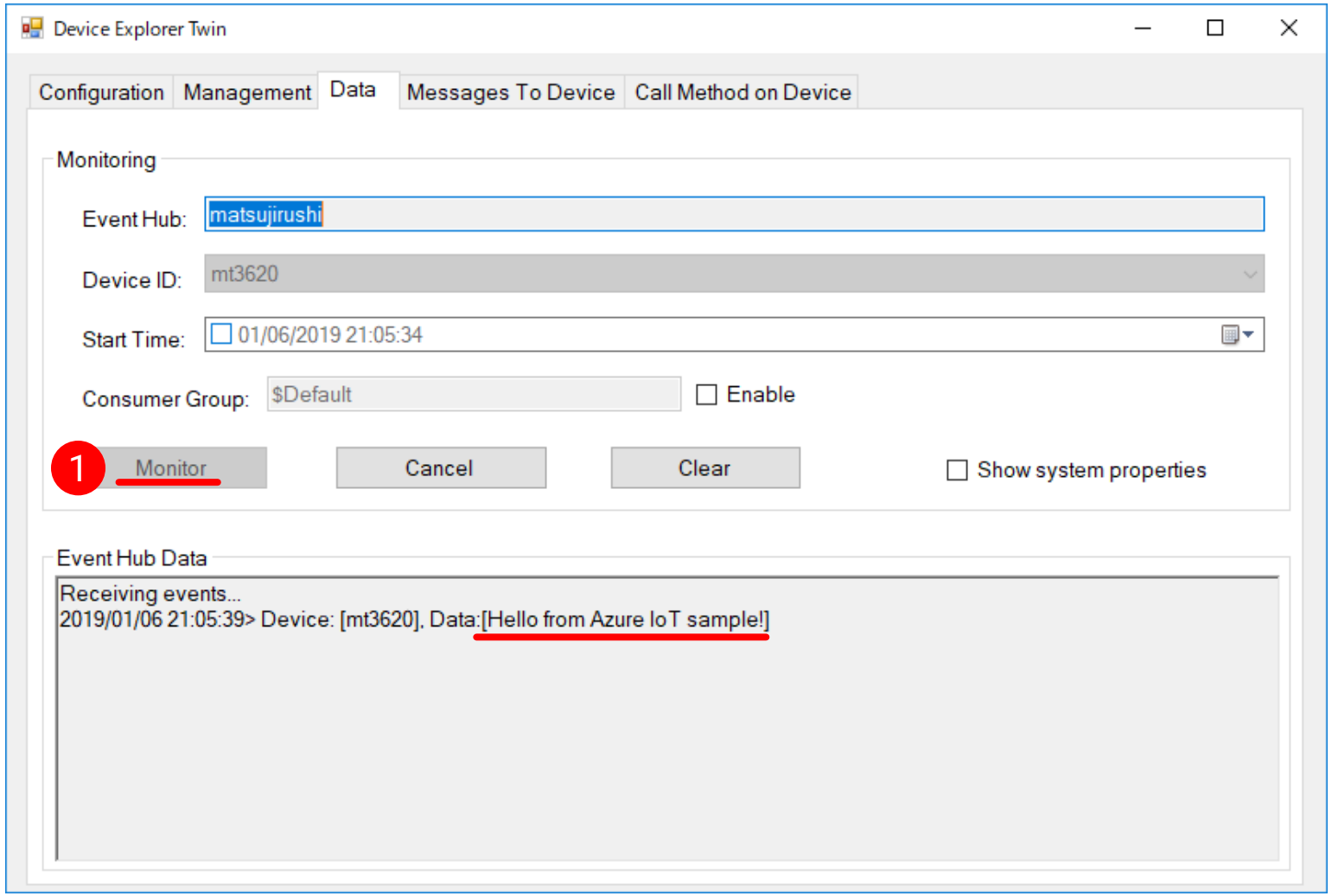
Device Explorer - Configuration





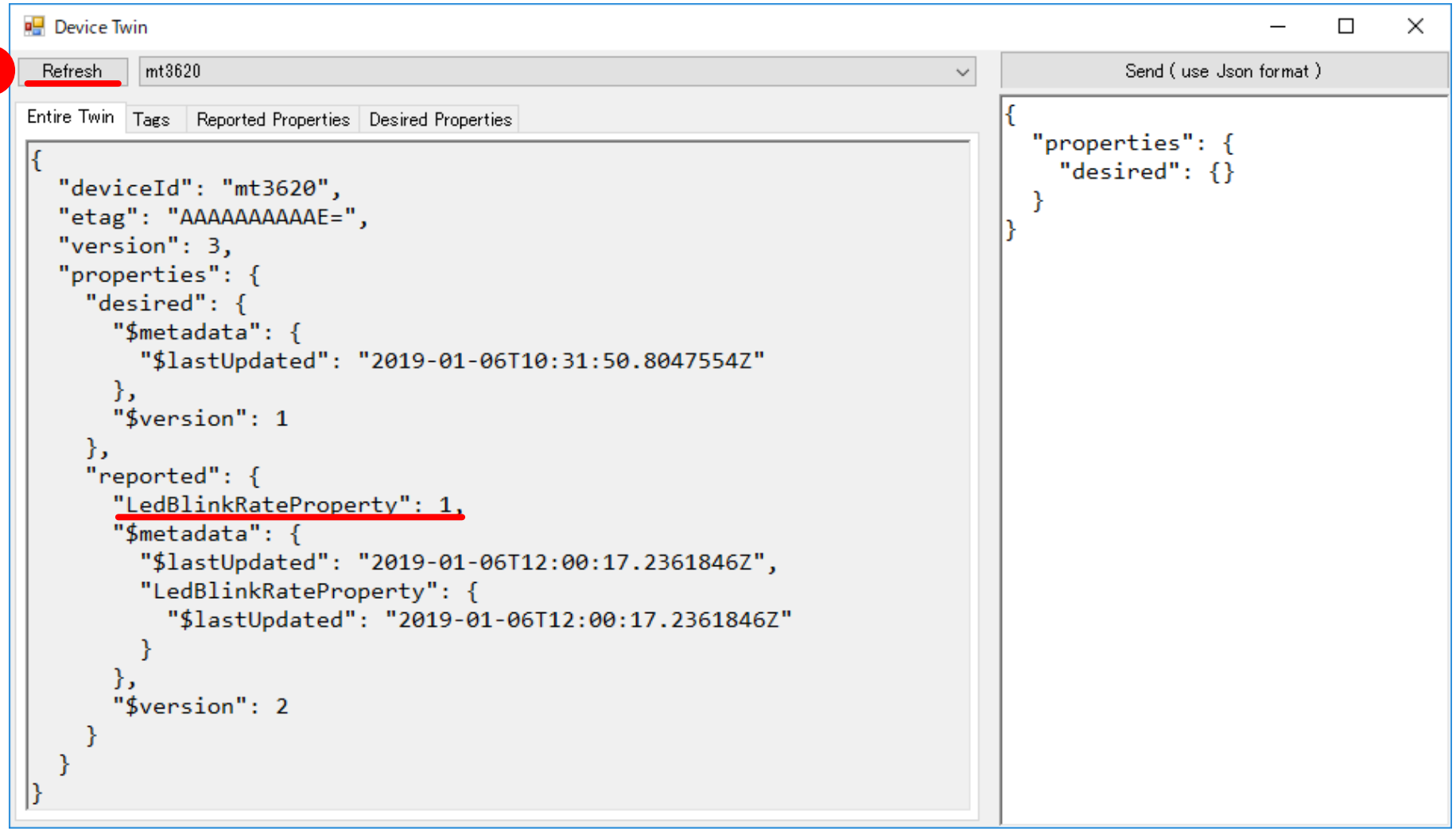
Lab#4.5 Azure IoT Hub Sampleを実行

Bボタン --> D2Cメッセージ



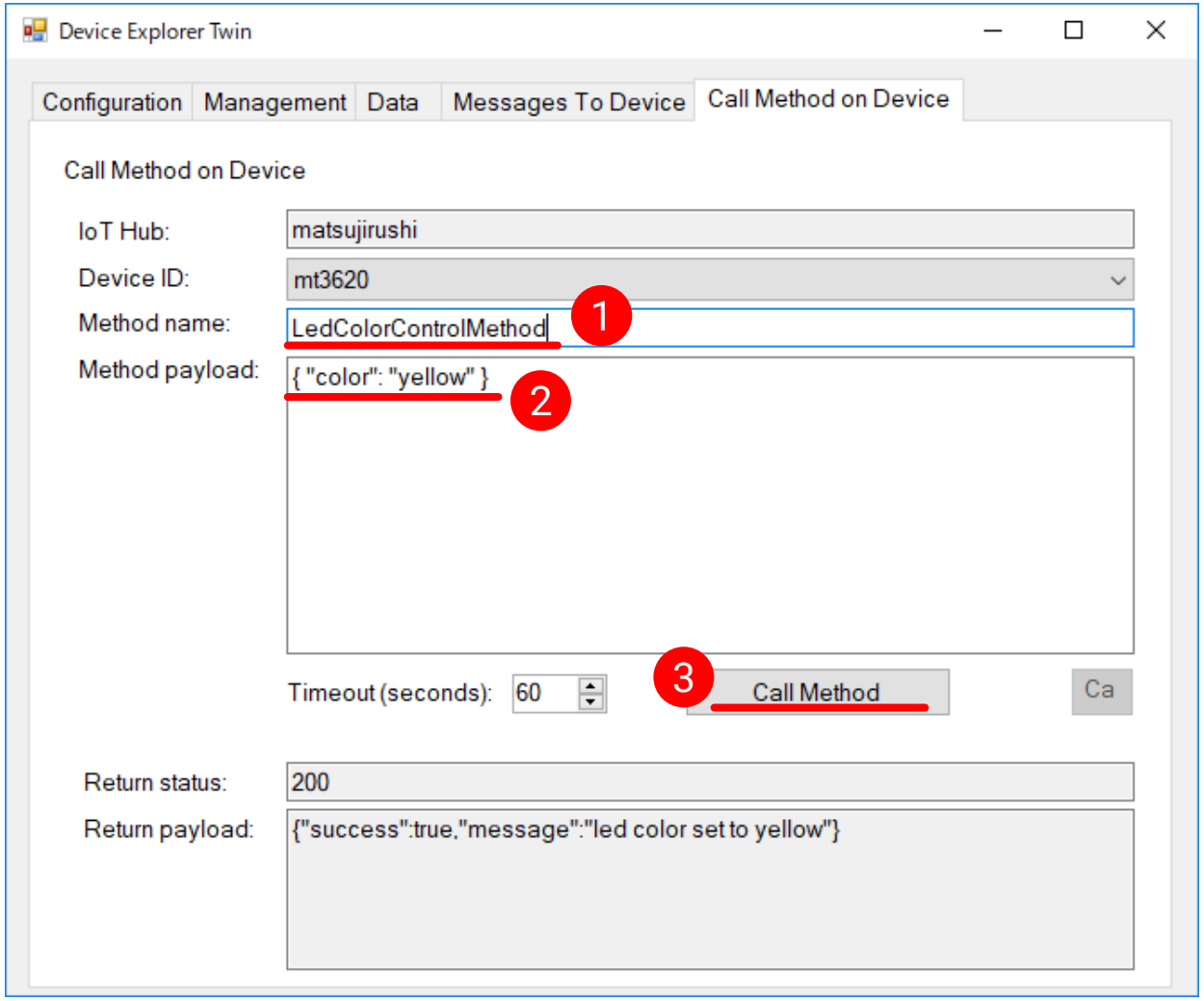
Lab#4.5 Azure IoT Hub Sampleを実行

Aボタン --> DeviceTwin - Reported Property



Lab#4.5 Azure IoT Hub Sampleを実行

DirectMethod “LedColorControlMethod”--> LED1の色



Lab#4 Appendix

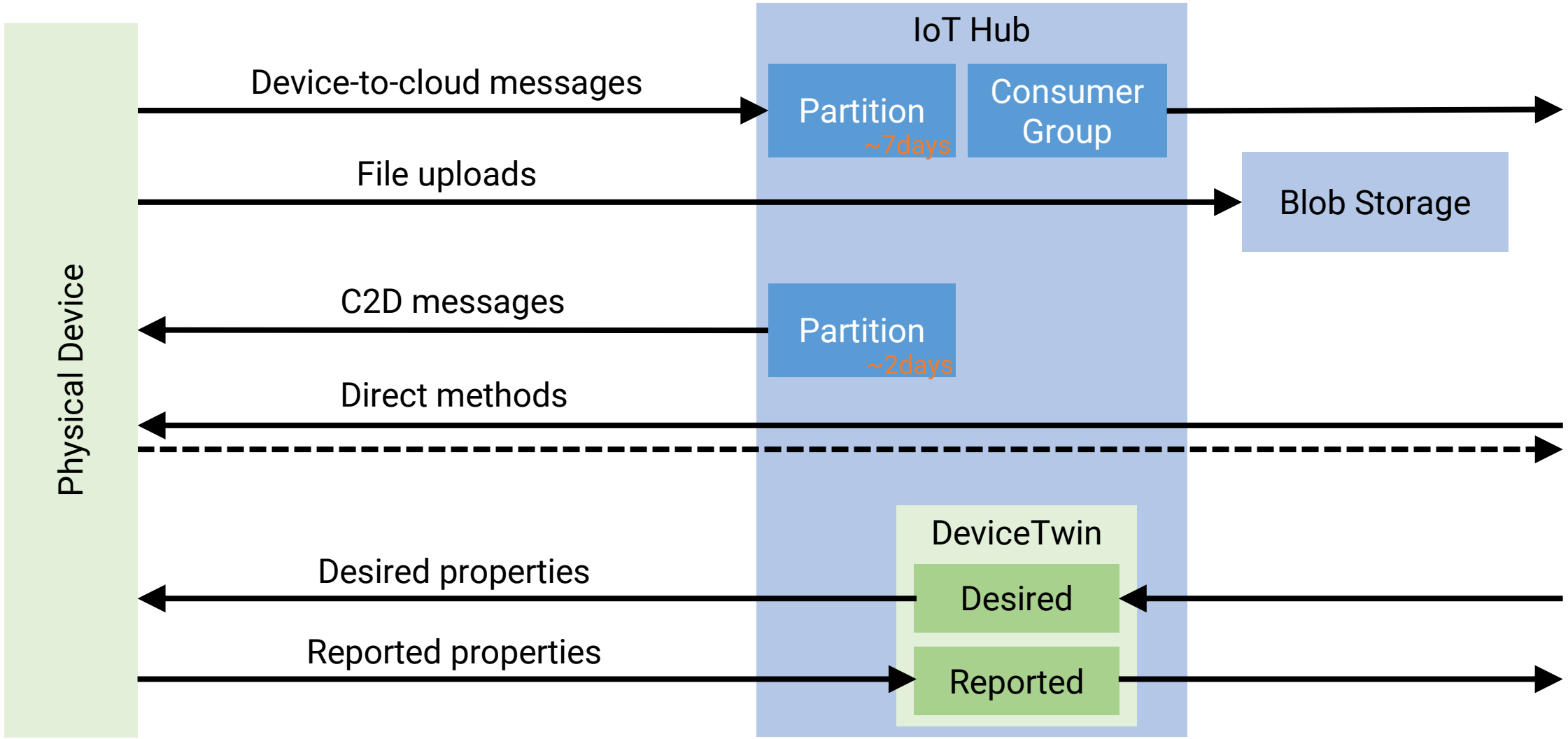
azsphere device wifi

Azure Sphere Utility version 18.11.3.20146

Command	Operations			Required parameters	Summary
azsphere	device	wifi	add	ssid	Add the details of a wireless network to the attached device.
			delete	id	Remove the details of a wireless network from the attached device.
			disable	id	Disable a wireless network on the attached device.
			enable	id	Enable a wireless network on the attached device.
			list		List the current Wi-Fi configuration for the attached device.
			scan		Scan for available networks on the attached device.
			show-status		Show the status of the wireless interface on the attached device.

[azsphere operation list - 20181117]

Communicate a Device and Azure IoT Hub



Manage Devices Connecting to IoT Hub

Device Explorer

https://aka.ms/aziotdevexp

~~iothub-explorer~~

~~https://github.com/Azure/iothub-explorer#iothub-explorer~~

Azure IoT Extension for Azure CLI

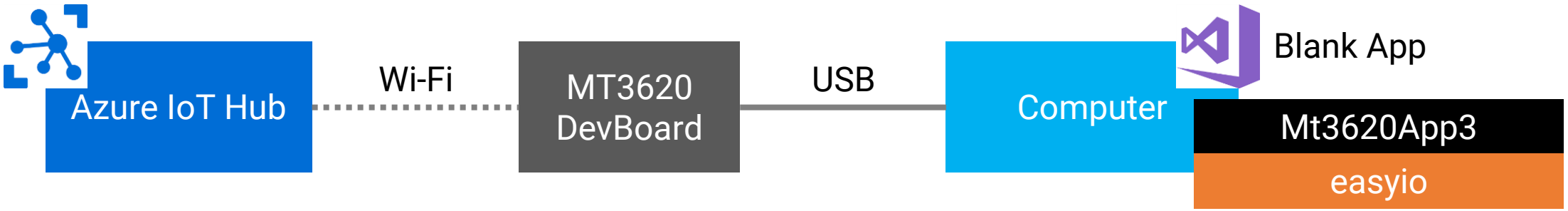
https://github.com/Azure/azure-iot-cli-extension#microsoft-azure-iot-extension-for-azure-cli

Azure IoT Hub Toolkit

https://marketplace.visualstudio.com/items?itemName=vsciot-vscode.azure-iot-toolkit

Lab#5

Lab#5 Create Telemetry App from the Blank App



得ること

- Azure IoT Device SDKの理解

やること

1. IoT Hubにデバイスを作成
2. Wi-Fi接続を設定
3. easyioをクローン
4. Blank Appを新規作成
5. easyioプロジェクトを追加・参照
6. Azure IoT Device SDKを追加
7. コードを追加

Lab#5.1 IoT Hubにデバイスを作成

Lab#5.2 Wi-Fi接続を設定

Lab#5.3 easyioをクローン

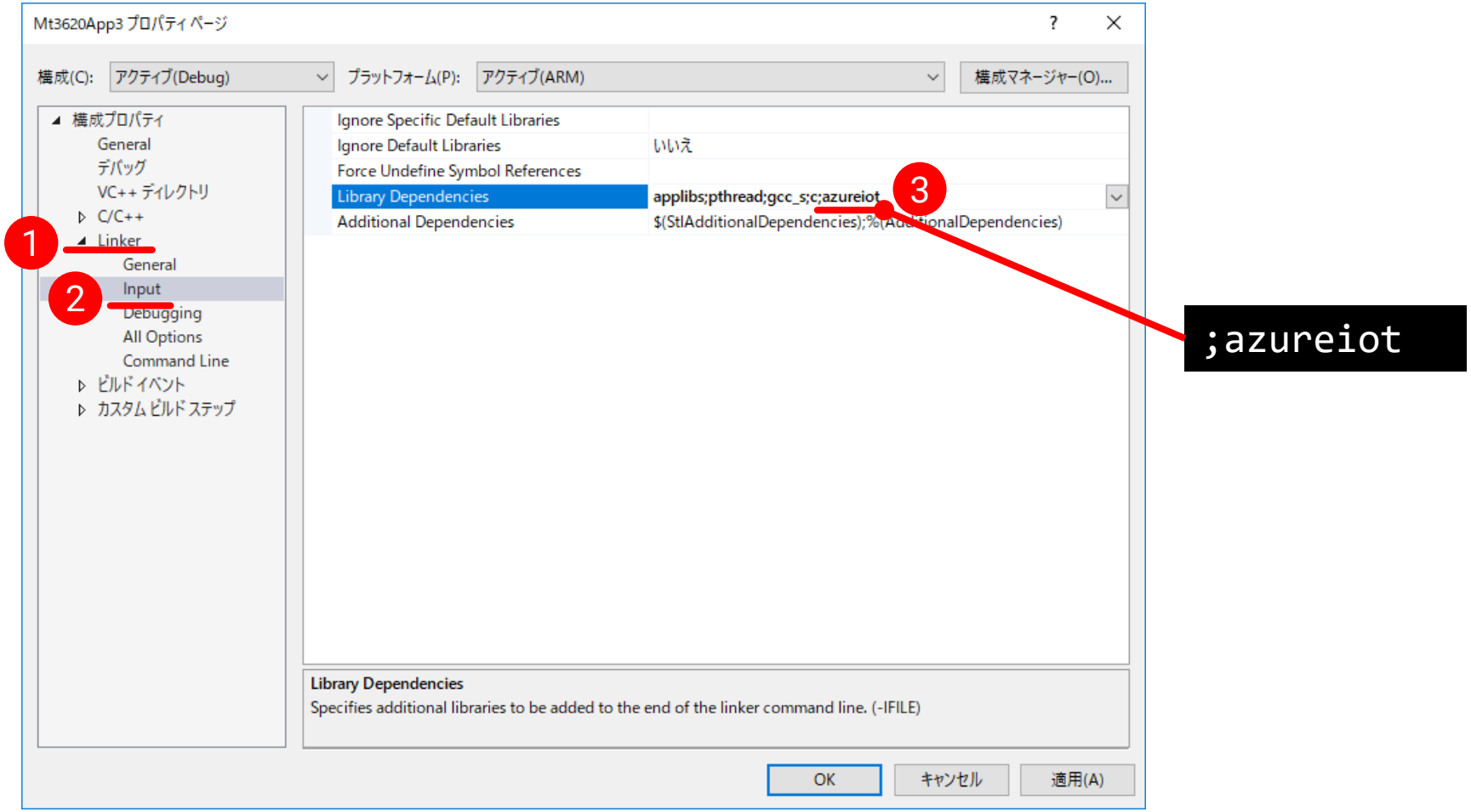
Lab#5.4 Blank Appを新規作成

Lab#5.5 easyioプロジェクトを追加・参照

（省略）

Lab#5.6 Azure IoT Device SDKを追加

プロジェクト（右クリック） > プロパティ
Linker > Input > Library Dependencies



Lab#5.7 コードを追加

main.c: ヘッダファイルをインクルード

```
#include <azureiot/iothubtransportmqtt.h>
#include <azureiot/iothub_device_client_ll.h>
#include <assert.h>
```

main.c: 接続文字列

```
#define CONNECTION_STRING   "具体的な接続文字列"
```

app_manifest.json

```
 "AllowedConnections": [ "名前.azure-devices.net" ],
```

Lab#5.7 コードを追加

main.c: 証明書

```
static const char azureIoTCertificatesX[] =
/* Baltimore */
"-----BEGIN CERTIFICATE-----¥r¥n"
"MIIDdzCCAl+gAwIBAgIEAgAAuTANBgkqhkiG9w0BAQUFADBaMQswCQYDVQQGEwJJ¥r¥n"
"RTESMBAGA1UEChMJQmFsdGltb3JlMRMwEQYDVQQLEwpDeWJlclRydXN0MSIwIAYD¥r¥n"
"VQQDExlCYWx0aW1vcmUgQ3liZXJUcnVzdCBSb290MB4XDTAwMDUxMjE4NDYwMFoX¥r¥n"
"DTI1MDUxMjIzNTkwMFowWjELMAkGA1UEBhMCSUUxEjAQBgNVBAoTCUJhbHRpbW9y¥r¥n"
"ZTETMBEGA1UECxMKQ3liZXJUcnVzdDEiMCAGA1UEAxMZQmFsdGltb3JlIEN5YmVy¥r¥n"
"VHJ1c3QgUm9vdDCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAKMEuyKr¥r¥n"
"mD1X6CZymrV51Cni4eiVgLGw41uOKymaZN+hXe2wCQVt2yguzmKiYv60iNoS6zjr¥r¥n"
"IZ3AQSsBUnuId9Mcj8e6uYi1agnnc+gRQKfRzMpijS3ljwumUNKoUMMo6vWrJYeK¥r¥n"
"mpYcqWe4PwzV9/lSEy/CG9VwcPCPwBLKBsua4dnKM3p31vjsufFoREJIE9LAwqSu¥r¥n"
"XmD+tqYF/LTdB1kC1FkYmGP1pWPgkAx9XbIGevOF6uvUA65ehD5f/xXtabz5OTZy¥r¥n"
"dc93Uk3zyZAsuT3lySNTPx8kmCFcB5kpvcY67Oduhjprl3RjM71oGDHweI12v/ye¥r¥n"
"jl0qhqdNkNwnGjkCAwEAAaNFMEMwHQYDVR0OBBYEFOWdWTCCR1jMrPoIVDaGezq1¥r¥n"
"BE3wMBIGA1UdEwEB/wQIMAYBAf8CAQMwDgYDVR0PAQH/BAQDAgEGMA0GCSqGSIb3¥r¥n"
"DQEBBQUAA4IBAQCFDF2O5G9RaEIFoN27TyclhAO992T9Ldcw46QQF+vaKSm2eT92¥r¥n"
"9hkTI7gQCvlYpNRhcL0EYWoSihfVCr3FvDB81ukMJY2GQE/szKN+OMY3EU/t3Wgx¥r¥n"
"jkzSswF07r51XgdIGn9w/xZchMB5hbgF/X++ZRGjD8ACtPhSNzkE1akxehi/oCr0¥r¥n"
"Epn3o0WC4zxe9Z2etciefC7IpJ5OCBRLbf1wbWsaY71k5h+3zvDyny67G7fyUIhz¥r¥n"
"ksLi4xaNmjICq44Y3ekQEe5+NauQrz4wlHrQMz2nZQ/1/I6eYs9HRCwBXbsdtTLS¥r¥n"
"R9I4LtD+gdwyah617jzV/OeBHRnDJELqYzmp¥r¥n"
"-----END CERTIFICATE-----¥r¥n";
```

Lab#5.7 コードを追加

main.c: クライアントハンドルの作成、証明書を設定、処理を実行、クライアントハンドルの破棄

```
IOTHUB_DEVICE_CLIENT_LL_HANDLE clientHandle =
IoTHubDeviceClient_LL_CreateFromConnectionString(CONNECTION_STRING, MQTT_Protocol);
assert(clientHandle != NULL);
```

```
IOTHUB_CLIENT_RESULT result;
result =
IoTHubDeviceClient_LL_SetOption(clientHandle, "TrustedCerts", azureIoTCertificatesX);
assert(result == IOTHUB_CLIENT_OK);
```

```
while (!terminationRequired)
{
    IoTHubDeviceClient_LL_DoWork(clientHandle);
}
```

```
IoTHubDeviceClient_LL_Destroy(clientHandle);
```

Lab#5.7 コードを追加

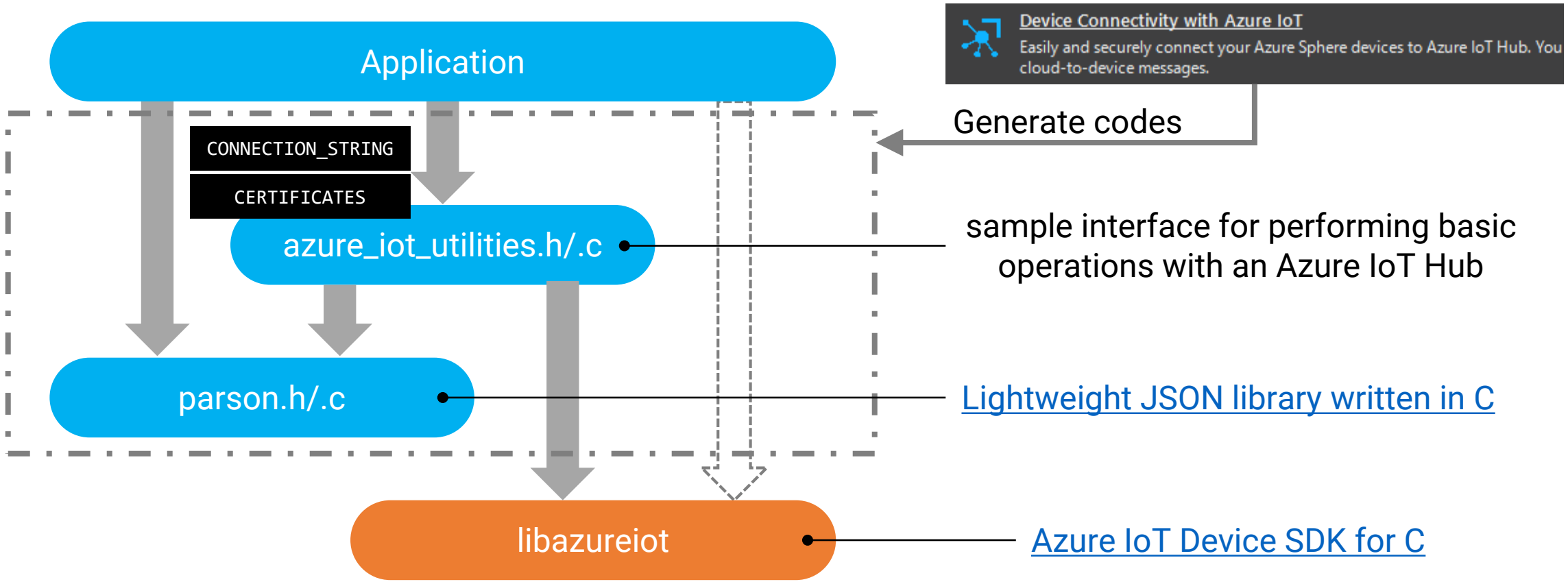
main.c: D2Cメッセージを送信

```
IOTHUB_MESSAGE_HANDLE messageHandle = IoTHubMessage_CreateFromString("Hello world.");
assert(messageHandle != NULL);
result = IoTHubDeviceClient_LL_SendEventAsync(clientHandle, messageHandle, NULL, 0);
assert(result == IOTHUB_CLIENT_OK);
IoTHubMessage_Destroy(messageHandle);
```

「Bボタンを押したときに、D2Cメッセージを送信」するよう、デジタル入力のコードを追加してください。

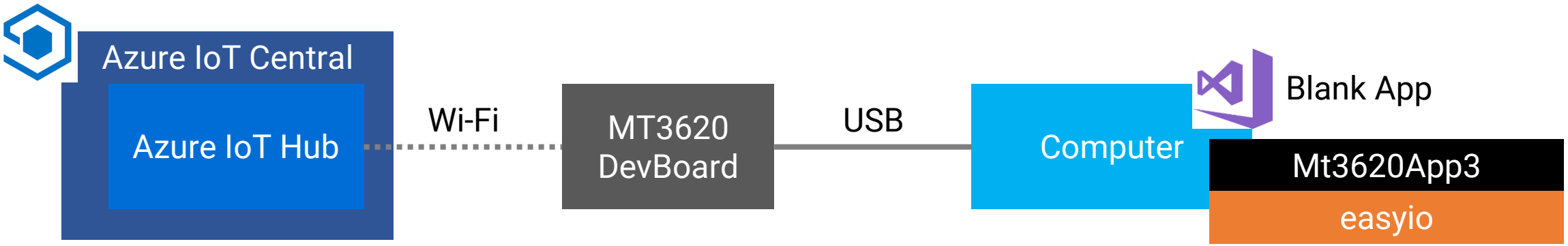
Lab#5 Appendix

Device Connectivity with Azure IoT



Lab#6

Lab#6 Send to Azure IoT Central w/o DPS



得ること

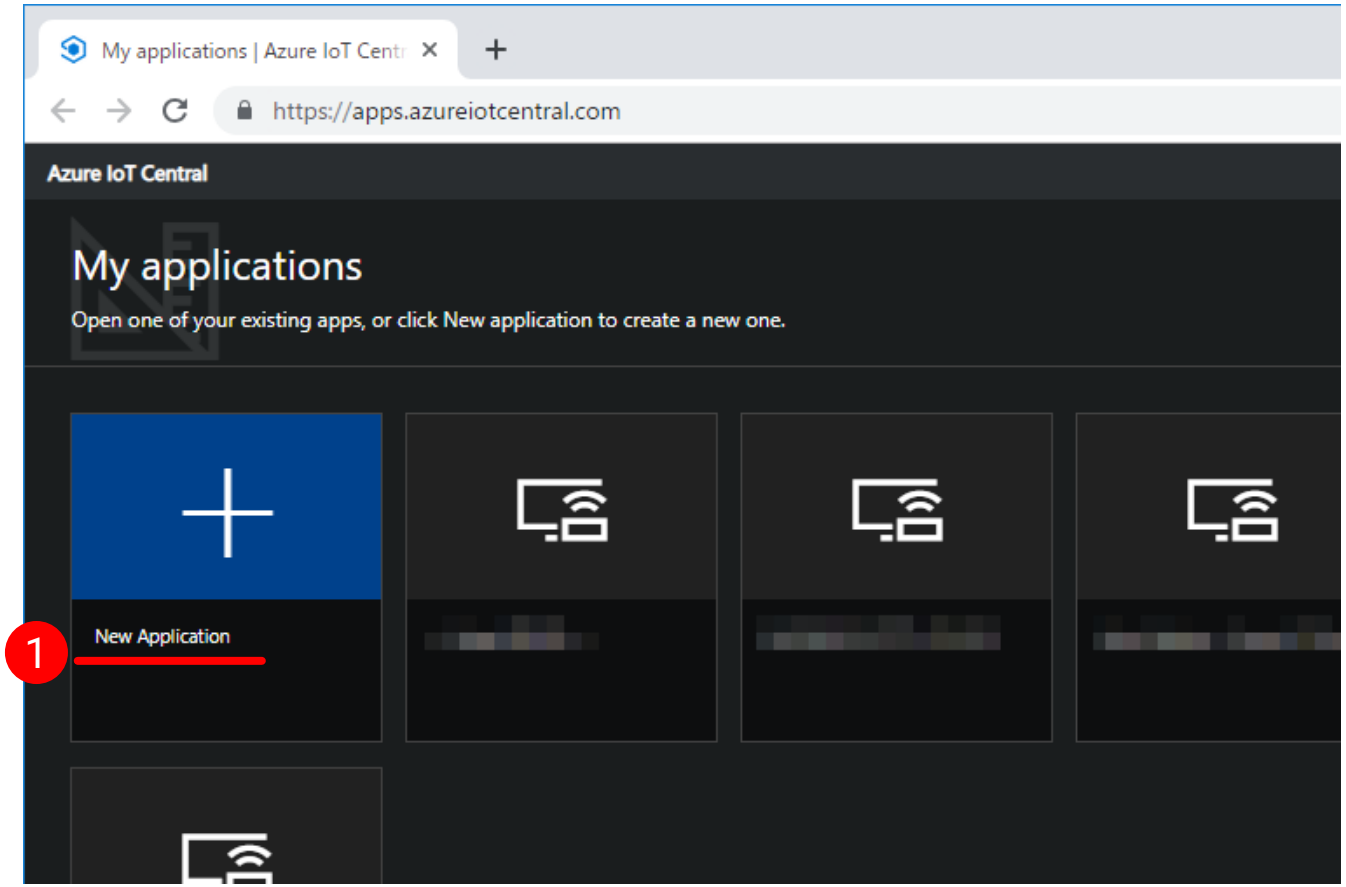
- Azure IoT Centralの使用

やること

1. IoT Centralアプリケーションを作成
2. IoT Centralにデバイステンプレートを追加・設定
3. IoT Centralにデバイスを追加
4. デバイスのIoT Hub接続文字列を取得
5. Lab#5のコードを変更

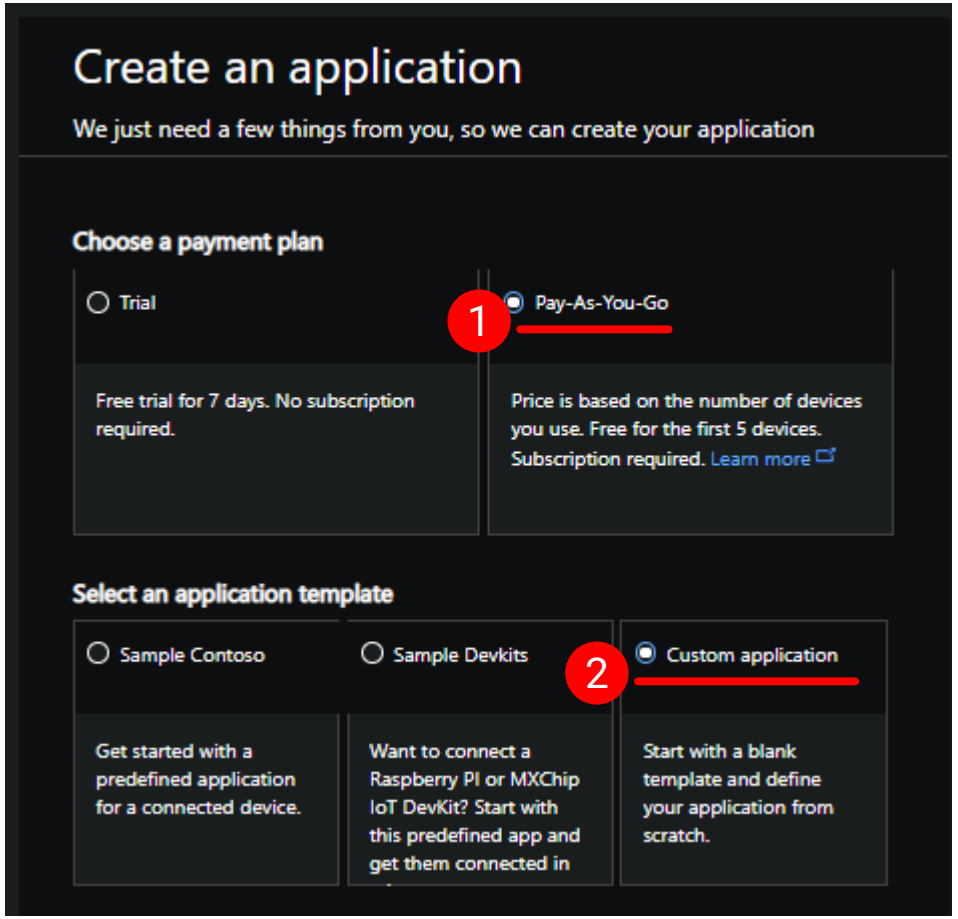
Lab#6.1 IoT Centralアプリケーションを作成

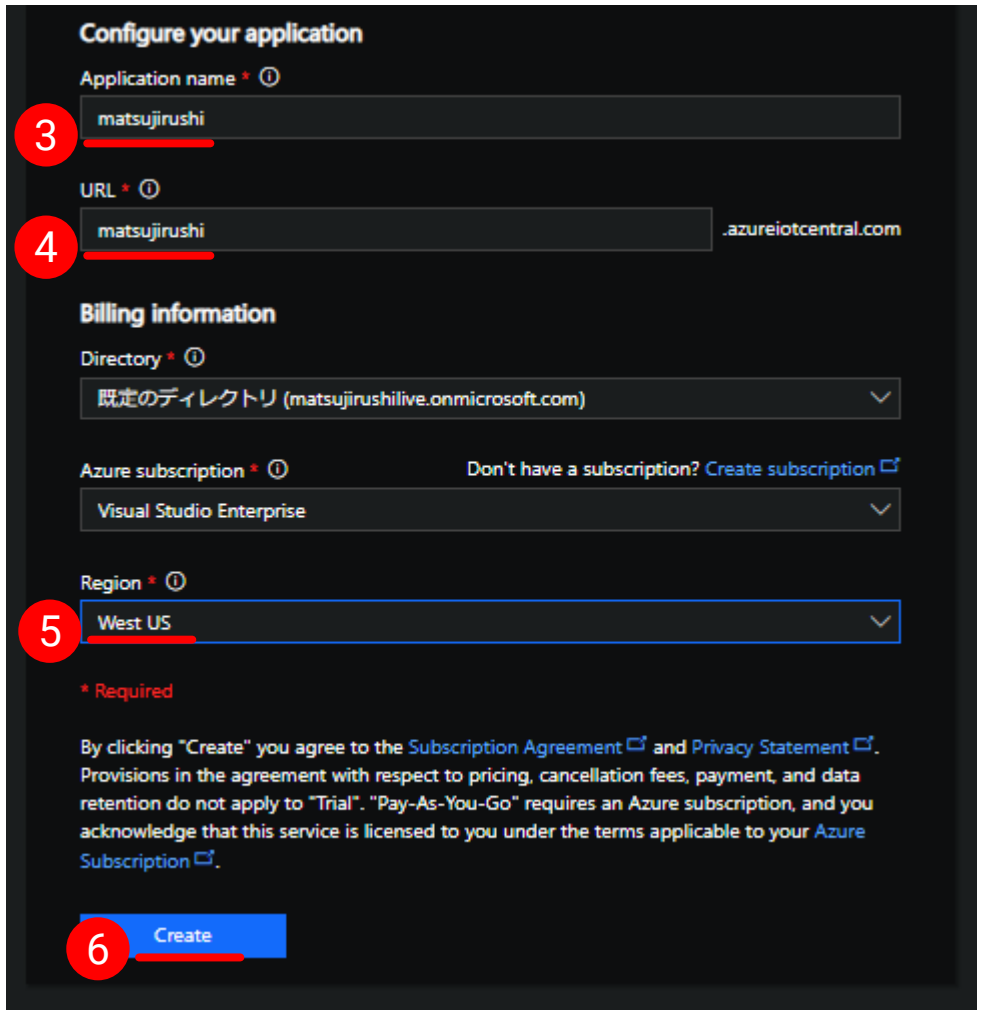
https://apps.azureiotcentral.com/



詳しくは、下記URLを参照してください。
https://docs.microsoft.com/ja-jp/azure/iot-central/quick-deploy-iot-central

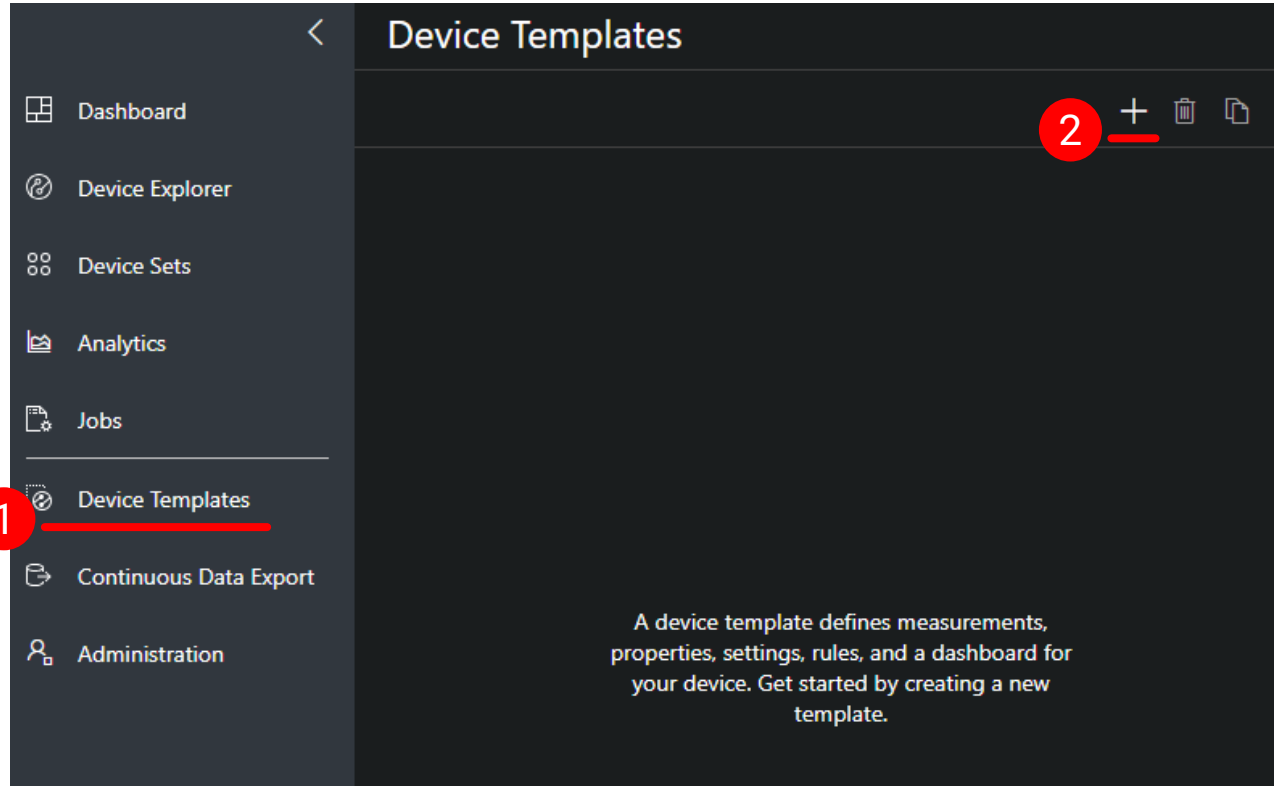
Lab#6.1 IoT Centralアプリケーションを作成

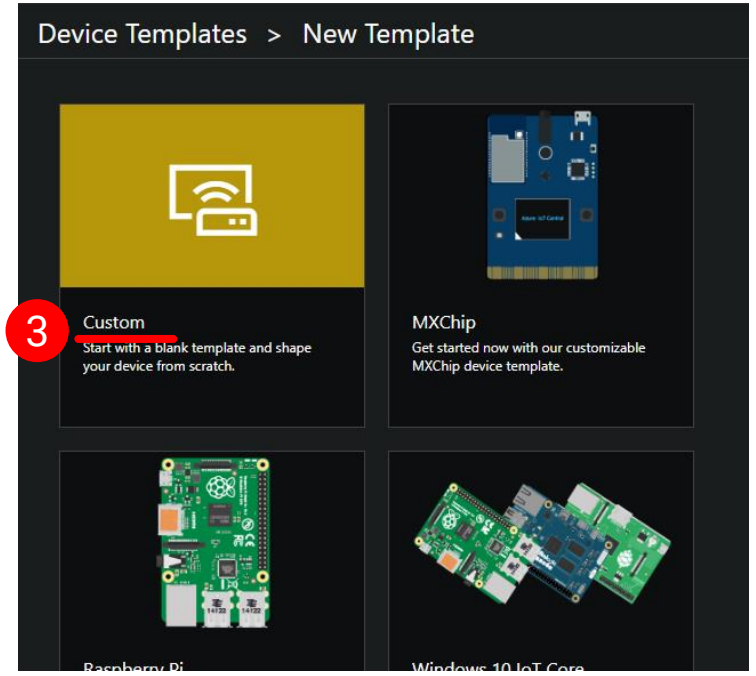


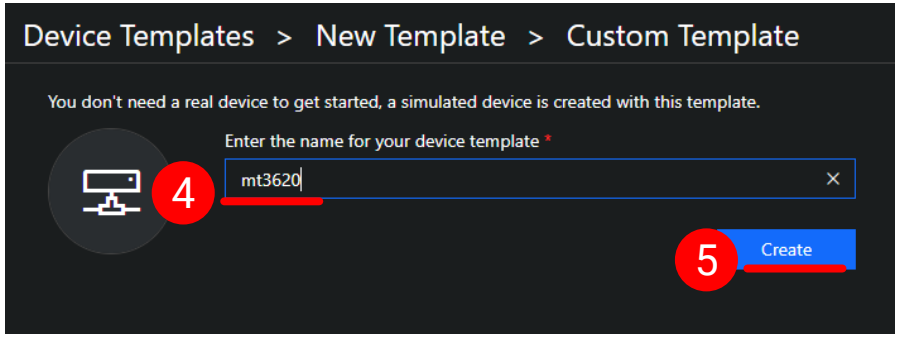


詳しくは、下記URLを参照してください。
https://docs.microsoft.com/ja-jp/azure/iot-central/quick-deploy-iot-central

Lab#6.2 IoT Centralにデバイステンプレートを追加・設定

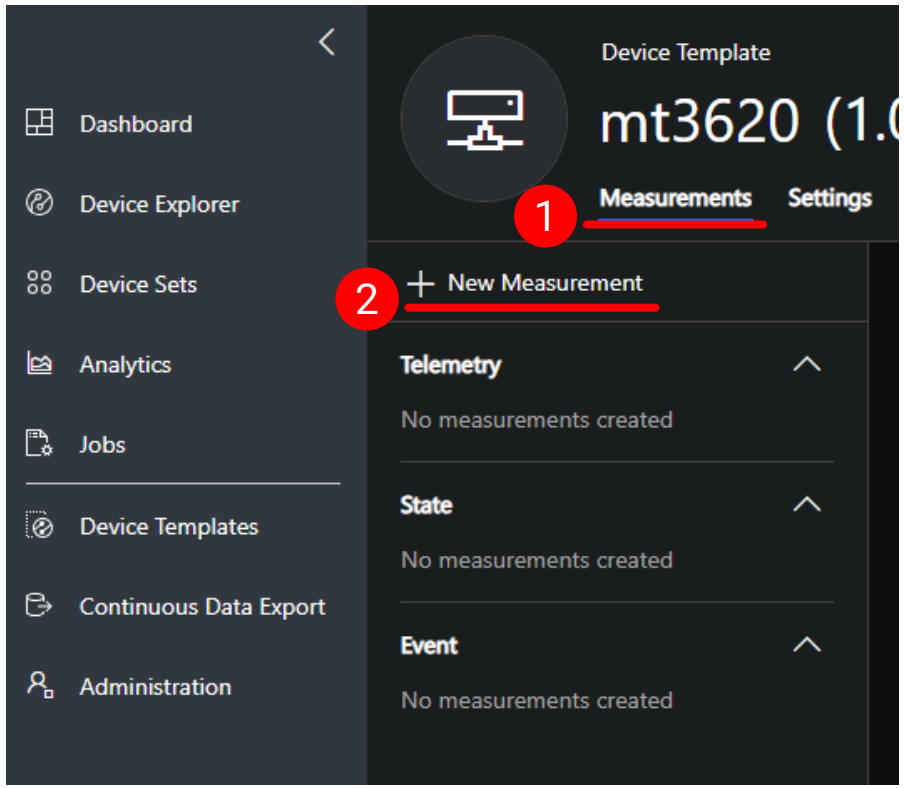


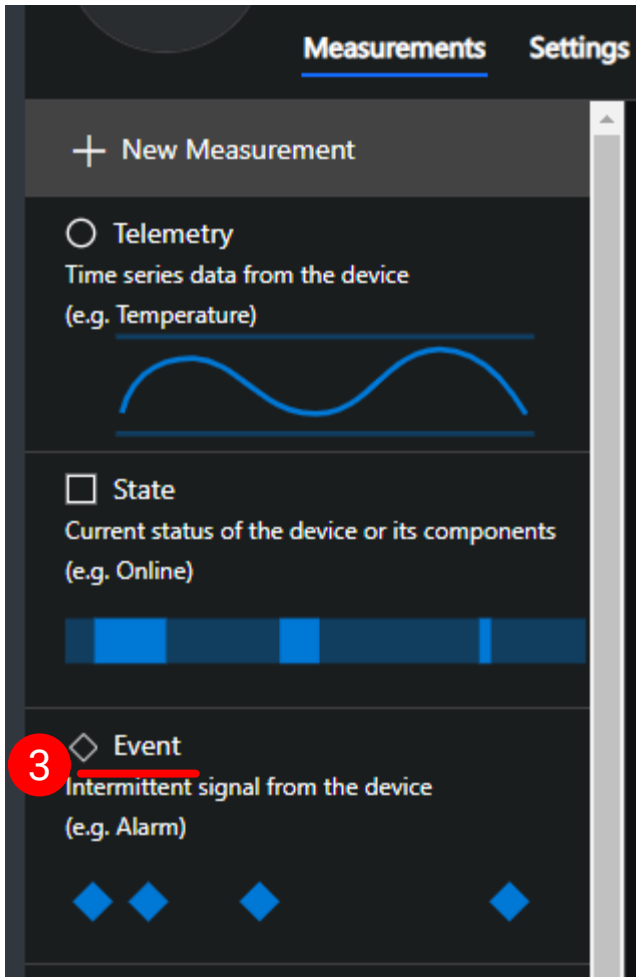


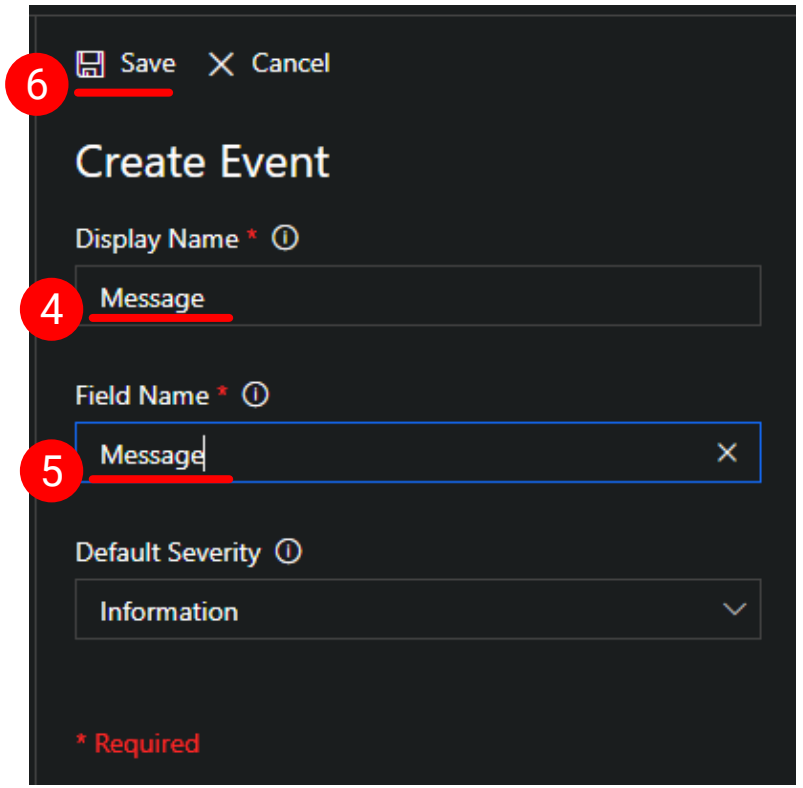


詳しくは、下記URLを参照してください。
https://docs.microsoft.com/ja-jp/azure/iot-central/tutorial-define-device-type#add-a-device-template

Lab#6.2 IoT Centralにデバイステンプレートを追加・設定

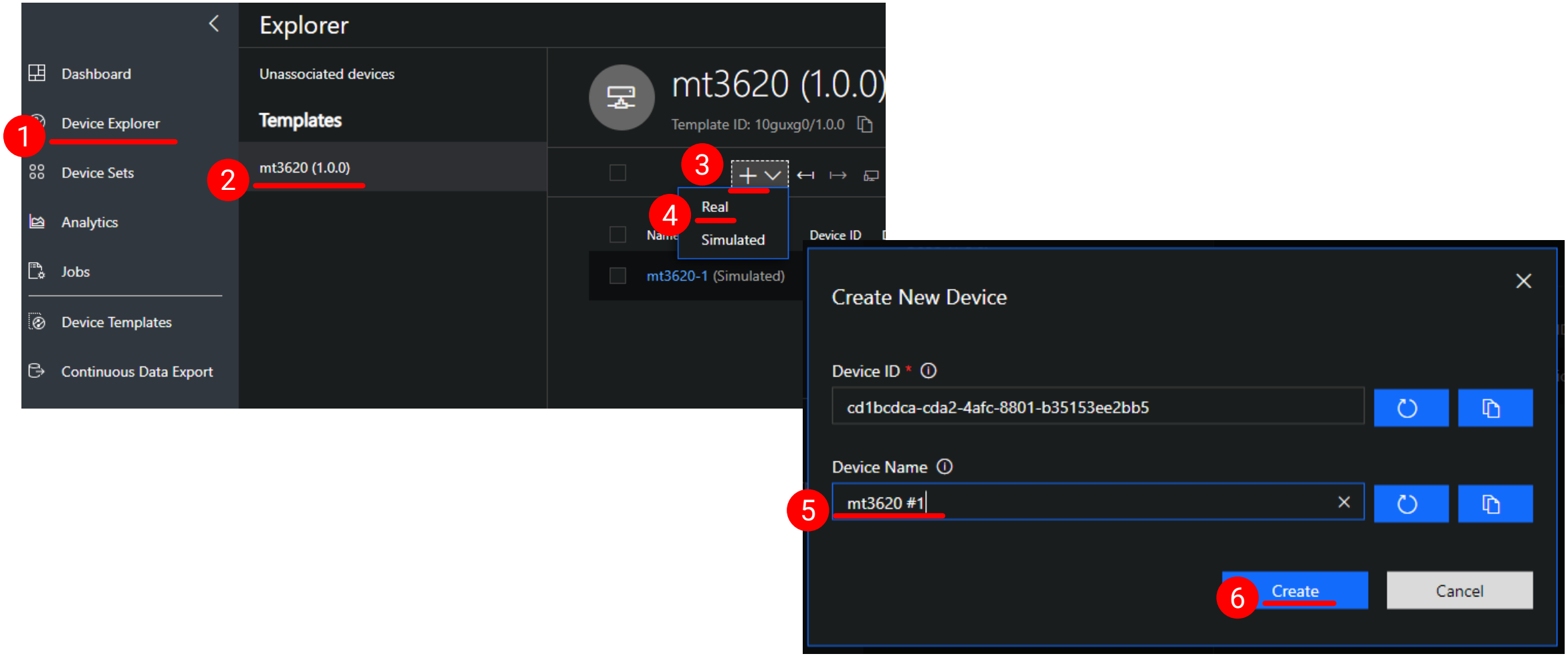






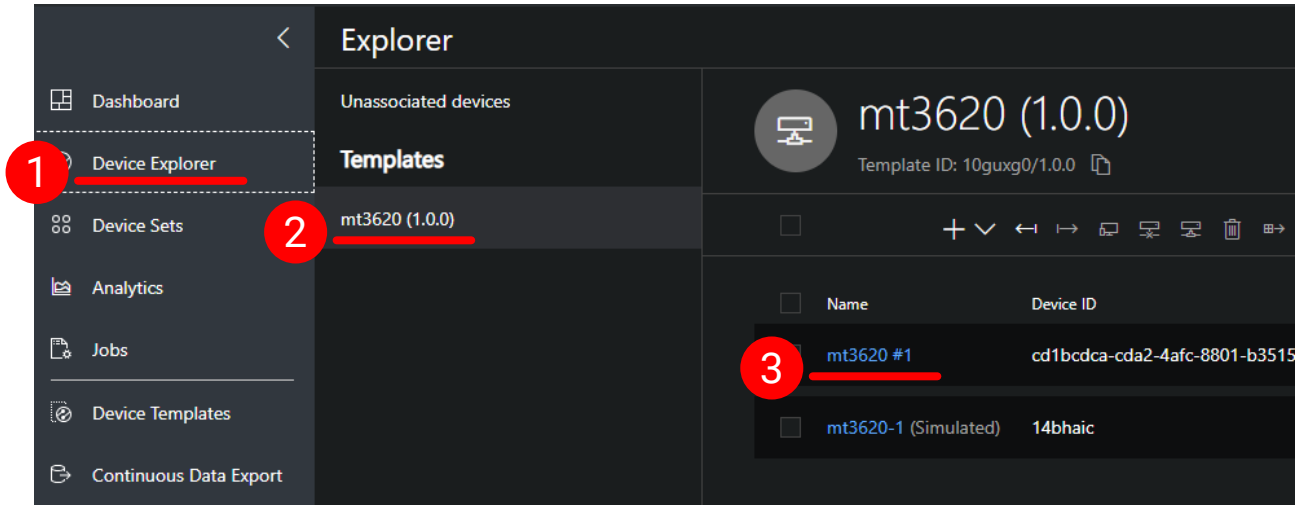
詳しくは、下記URLを参照してください。
https://docs.microsoft.com/ja-jp/azure/iot-central/tutorial-define-device-type#add-a-device-template

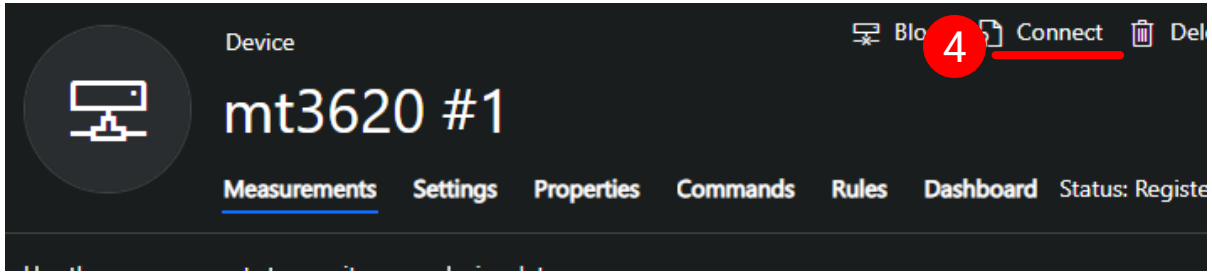
Lab#6.3 IoT Centralにデバイスを追加

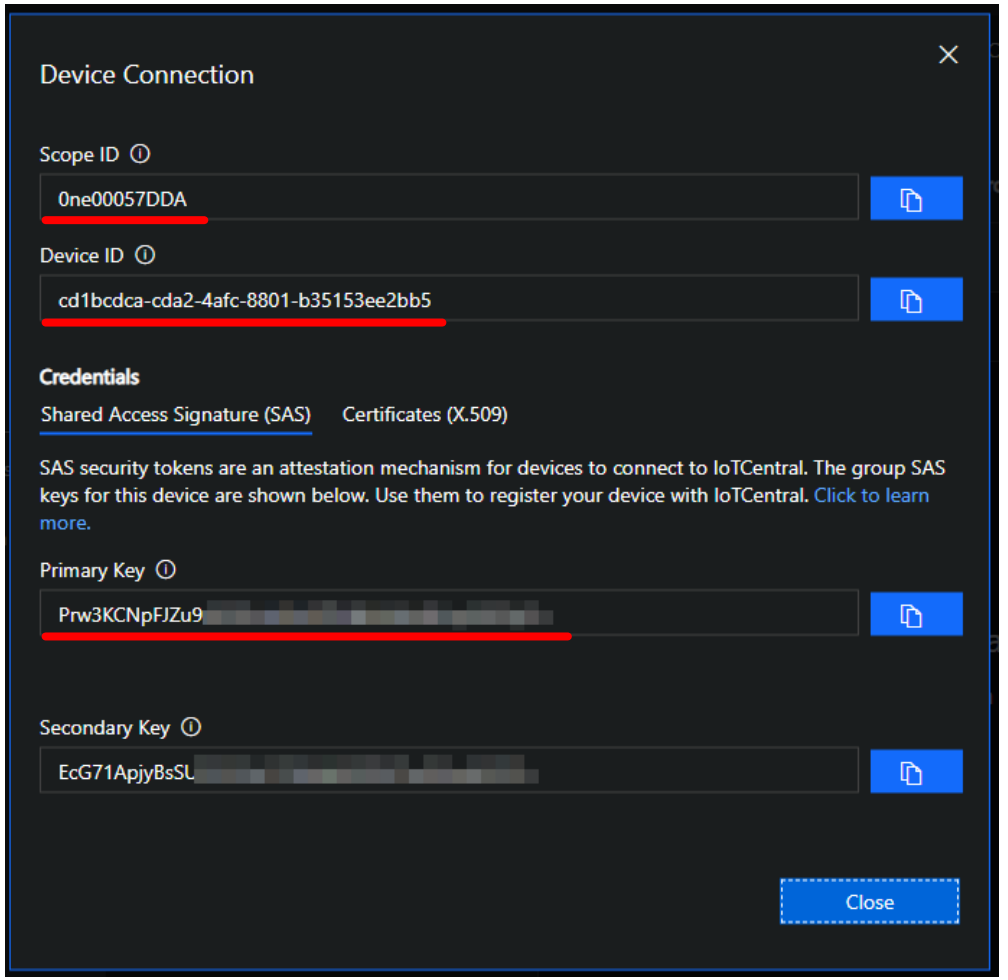


詳しくは、下記URLを参照してください。
https://docs.microsoft.com/ja-jp/azure/iot-central/tutorial-add-device#add-a-real-device

Lab#6.4 デバイスのIoT Hub接続文字列を取得





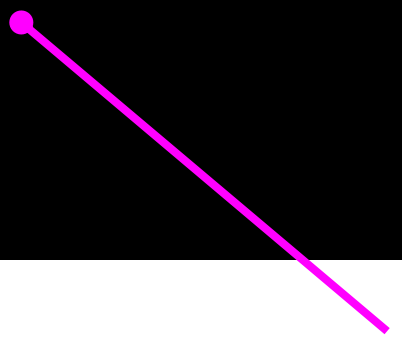


Lab#6.4 デバイスのIoT Hub接続文字列を取得

https://github.com/Azure/dps-keygen/blob/master/bin/windows/dps_cstr.zip からダウンロード

dps_str (Scope ID) (Device ID) (SAS Primary Key)

```
C:¥AzureSphere>dps_cstr 0ne00057DDA cd1bcdca-cda2-4afc-8801-b35153ee2bb5
Prw3KCNpFJZ...=
...
Registration Information received from service: iotc-79abbad4-667d-4870-942e-
8b413003ab64.azure-devices.net!
Connection String:
HostName=iotc-79abbad4-667d-4870-942e-8b413003ab64.azure-
devices.net;DeviceId=cd1bcdca-cda2-4afc-8801-
b35153ee2bb5;SharedAccessKey=Prw3KCNpFJZ...=

C:¥AzureSphere>
```



IoT Hub接続文字列

Lab#6.5 Lab#5のコードを変更

main.c: 接続文字列

```
#define CONNECTION_STRING    "具体的な接続文字列"
```

app_manifest.json

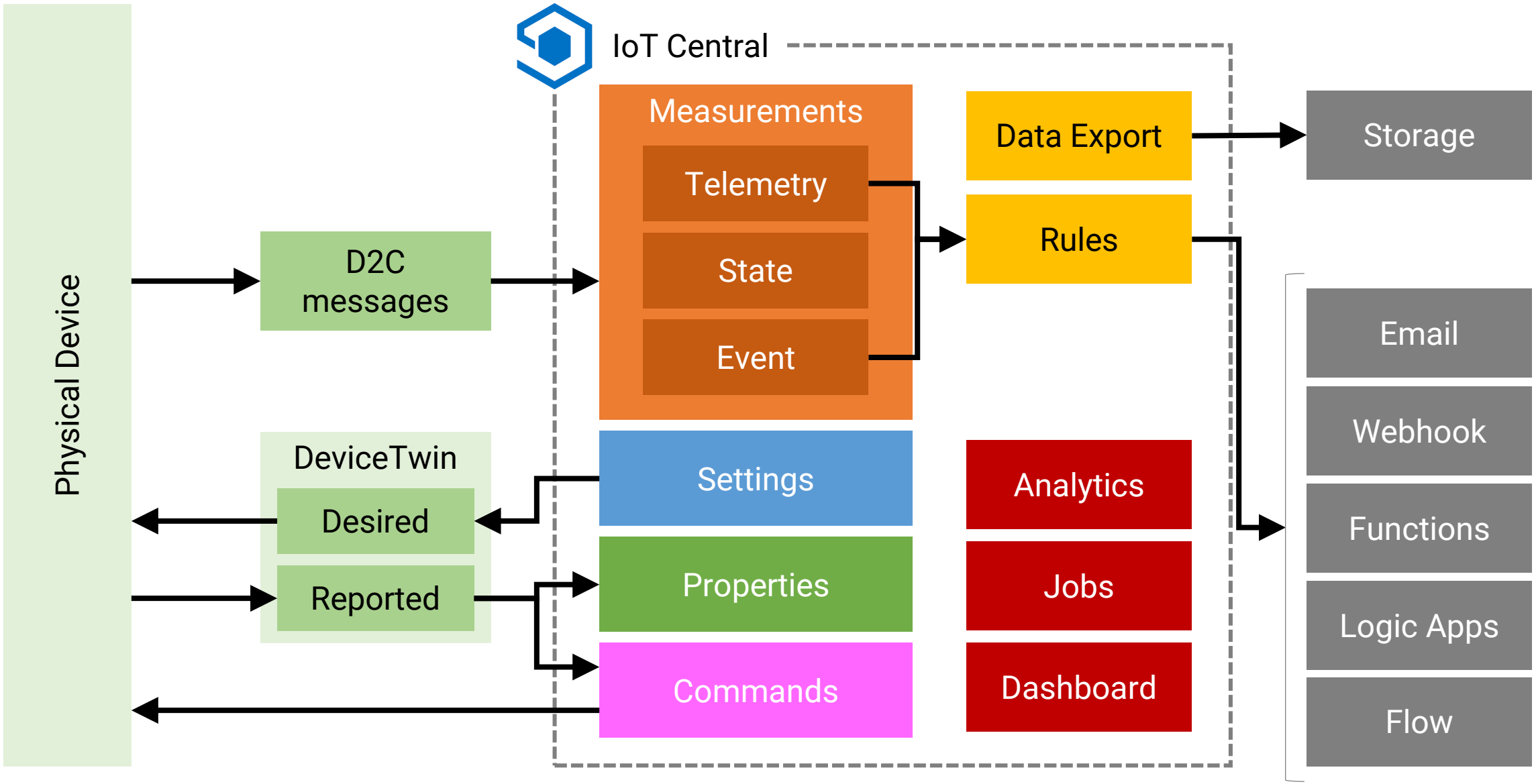
```
 "AllowedConnections": [ "名前.azure-devices.net" ],
```

main.c: D2Cメッセージを送信

```
IOTHUB_MESSAGE_HANDLE messageHandle =
    IoTHubMessage_CreateFromString("{¥"Message¥":¥"Hello world. ¥"}");
assert(messageHandle != NULL);
result = IoTHubDeviceClient_LL_SendEventAsync(clientHandle, messageHandle, NULL, 0);
assert(result == IOTHUB_CLIENT_OK);
IoTHubMessage_Destroy(messageHandle);
```

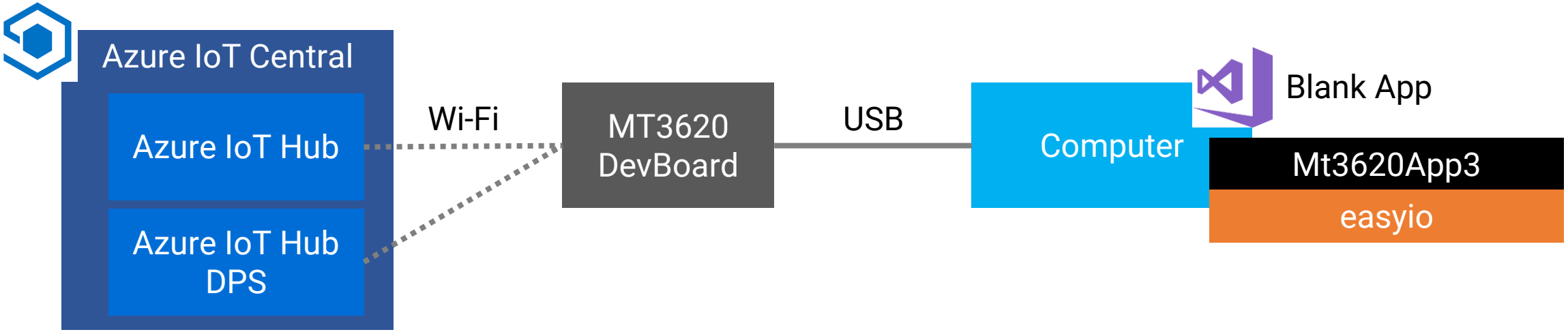
Lab#6 Appendix

Azure IoT Central



Lab#7

Lab#7 Send to Azure IoT Central w DPS



得ること

- Azure IoT Hub DPSの使用

やること

1. Azure SphereテナントとIoT Centralを連携
2. Lab#5のコードを変更
3. IoT Centralにデバイスを追加

Lab#7.1 Azure SphereテナントとIoT Centralを連携

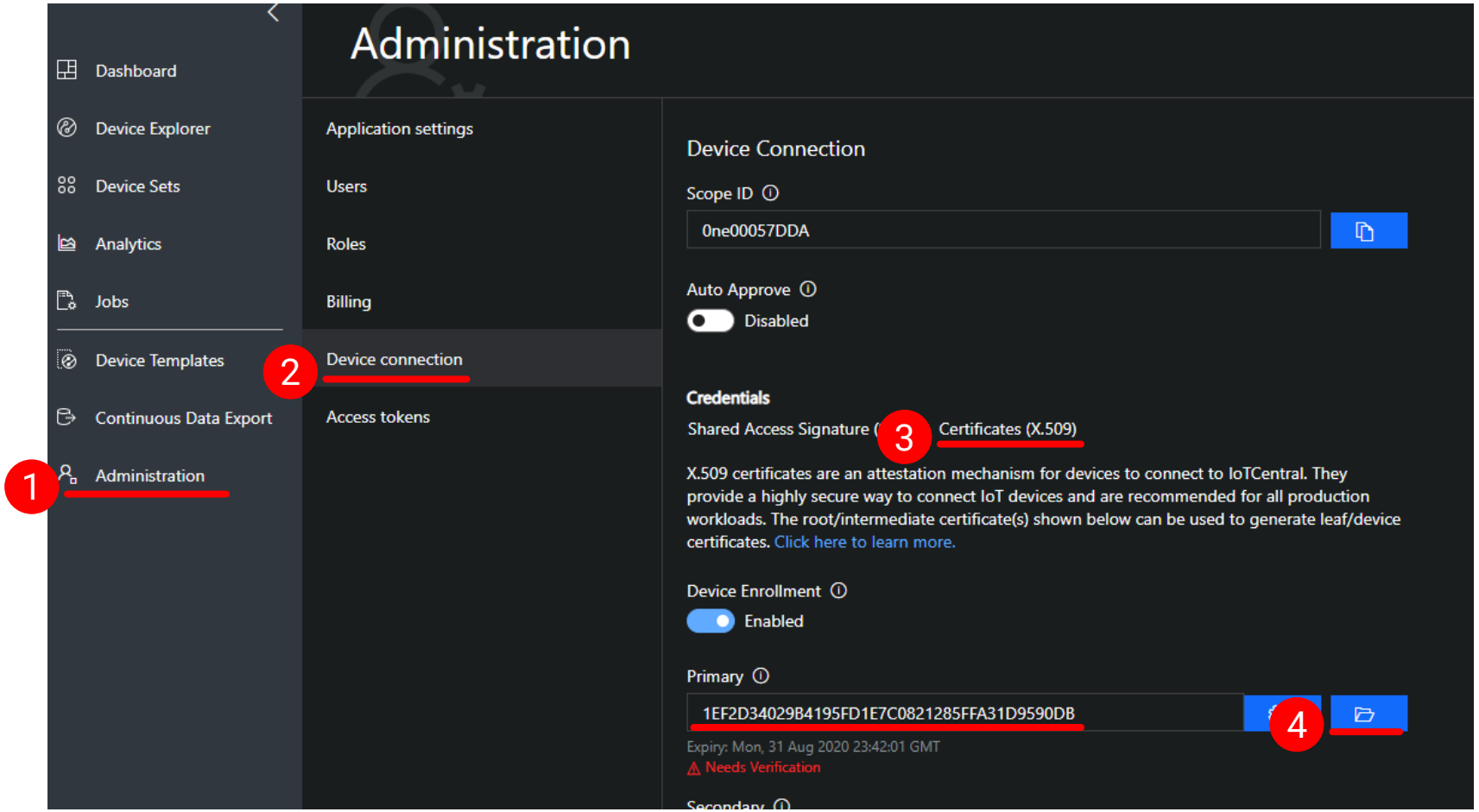
Azure SphereテナントのCA証明書を取得

azsphere tenant download-CA-certificate --output CAcertificate.cer

```
C:¥AzureSphere>azsphere tenant download-CA-certificate --output
CAcertificate.cer
Saving the CA certificate to 'C:¥AzureSphere¥CAcertificate.cer'.
Saved the CA certificate to 'CAcertificate.cer'.
Command completed successfully in 00:00:05.9005793.

C:¥AzureSphere>
```

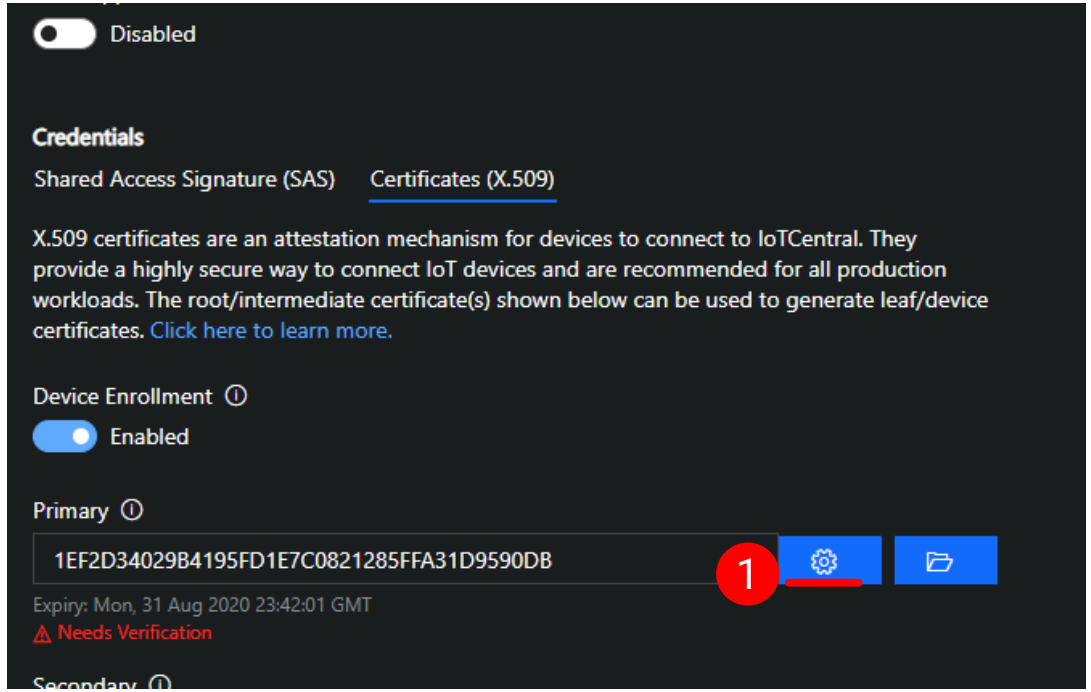
Lab#7.1 Azure SphereテナントとIoT Centralを連携

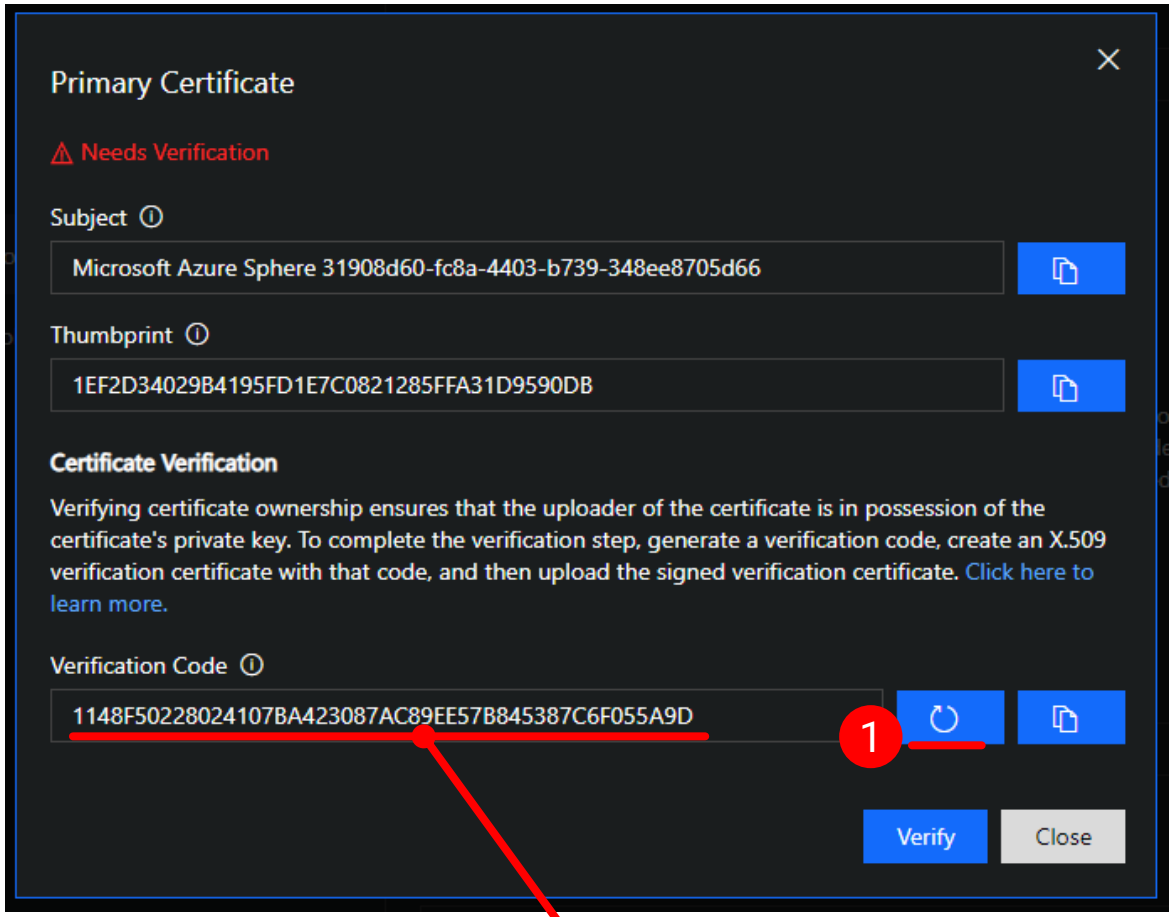
CA証明書を設定



Lab#7.1 Azure SphereテナントとIoT Centralを連携

CA証明書の検証コードを生成





検証コード

Lab#7.1 Azure SphereテナントとIoT Centralを連携

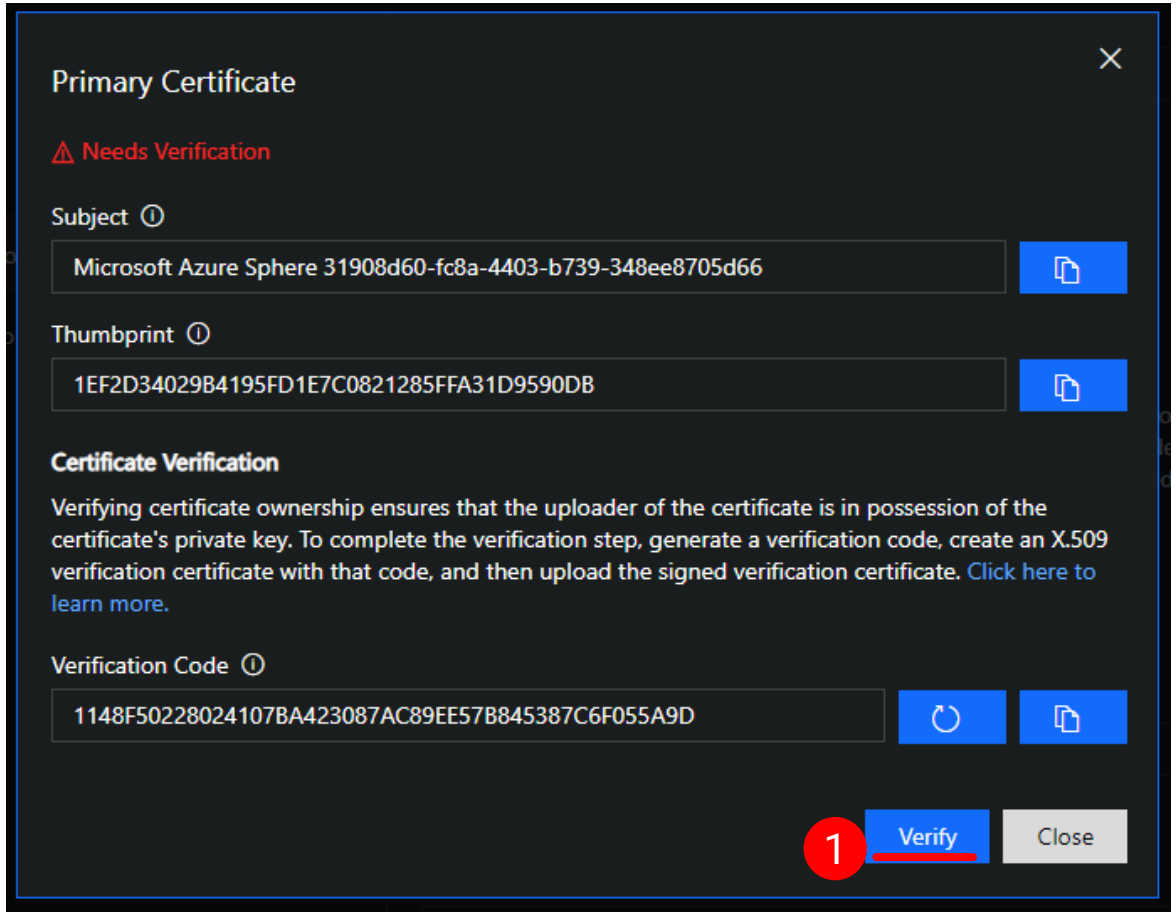
検証コードを使って検証証明書を取得

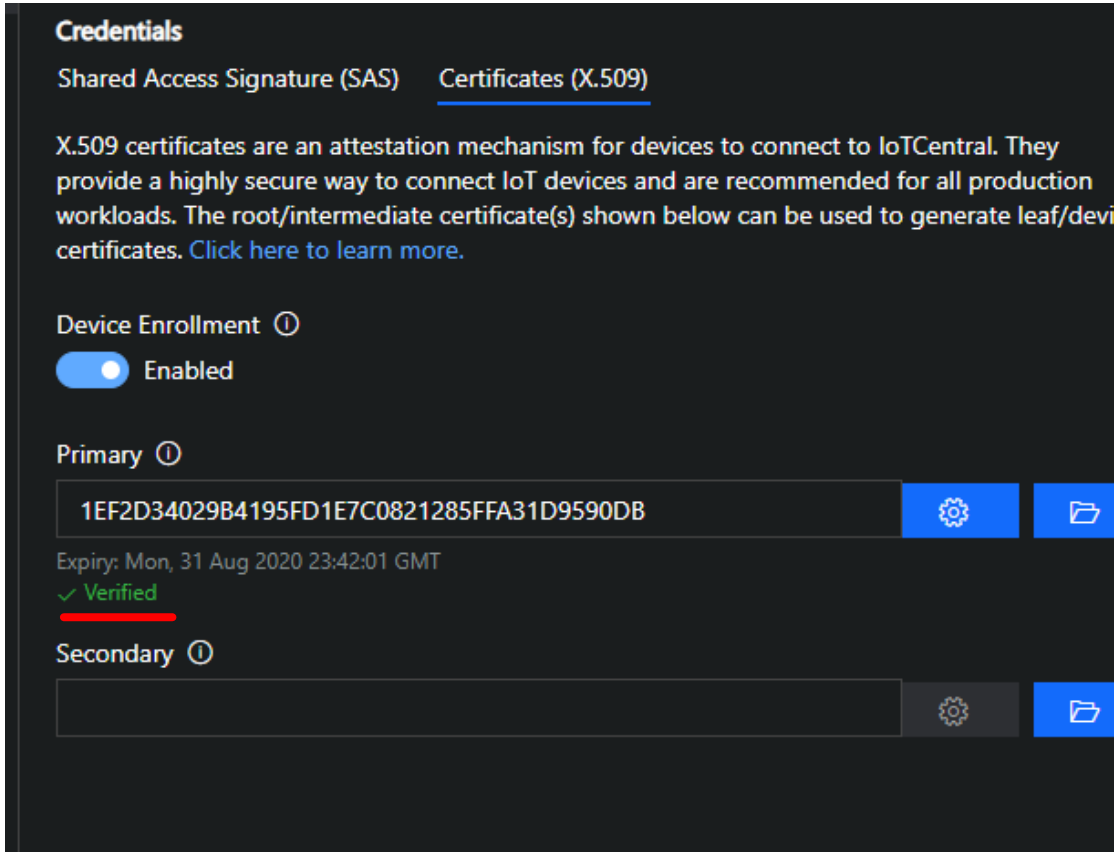
azsphere tenant download-validation-certificate --output ValidationCertification.cer
--verificationcode (Verification Code)

```
C:¥AzureSphere>azsphere tenant download-validation-certificate
         –output ValidationCertification.cer
         --verificationcode 1148F50228024107BA423087AC89EE57B845387C6F055A9D
Saving the validation certificate to 'C:¥AzureSphere¥ValidationCertification.cer'.
Saved the validation certificate to 'ValidationCertification.cer'.
Command completed successfully in 00:00:02.1465368.

C:¥AzureSphere>
```

Lab#7.1 Azure SphereテナントとIoT Centralを連携

検証証明書を設定





Lab#7.2 Lab#5のコードを変更

main.c: ヘッダファイルをインクルード

```
#include <azureiot/azure_sphere_provisioning.h>
```

main.c: クライアントハンドルの作成

```
IOTHUB_DEVICE_CLIENT_LL_HANDLE clientHandle;
AZURE_SPHERE_PROV_RETURN_VALUE provResult =
 IoTHubDeviceClient_LL_CreateWithAzureSphereDeviceAuthProvisioning("Scope ID", 10000,
                                                                   &clientHandle);
```

app_manifest.json

```
 "AllowedConnections": ["global.azure-devices-provisioning.net",
                                                    "名前.azure-devices.net" ],
...
 "SystemTime": false,
 "DeviceAuthentication": "Azure Sphere Tenant ID"
```

azsphere tenant show-selected の実行結果

Lab#7.3 IoT Centralにデバイスを追加

device show-attachedの実行結果の、Device IDを小文字に変換

Lab#7 Appendix

Azure Sphere Device Provisioning