**W5500-EVB-Pico**

**Getting Started Guide**

**for Azure IoT**

**Version 1.0.0**



<http://www.wiznet.io/>

Table of Contents

[1 Document information 3](#_Toc120718963)

[1.1 Revision history 3](#_Toc120718964)

[2 Introduction 4](#_Toc120718965)

[3 Prerequisites 5](#_Toc120718966)

[3.1 Prepare your development environment 5](#_Toc120718967)

[3.1.1 Tools installation 5](#_Toc120718968)

[3.1.2 Other software required to develop and debug applications for the device 14](#_Toc120718969)

[3.1.3 Additional software references 15](#_Toc120718970)

[3.2 Setup your IoT Hub 15](#_Toc120718971)

[3.3 Provision your device over DPS 15](#_Toc120718972)

[4 Prepare your Device 20](#_Toc120718973)

[4.1 Connect with ethernet cable 20](#_Toc120718974)

[4.2 Connect with 5 pin micro USB cable. 20](#_Toc120718975)

[5 Build SDK and Run Samples 21](#_Toc120718976)

[5.1 Select sample application 21](#_Toc120718977)

[5.2 Enter certificate and DPS related information 21](#_Toc120718978)

[5.3 Build example 24](#_Toc120718979)

[5.4 Upload firmware 24](#_Toc120718980)

[5.5 Run sample application 25](#_Toc120718981)

[6 Integration with Azure IoT Explorer 29](#_Toc120718982)

[6.1 Run Azure IoT explorer 29](#_Toc120718983)

[6.2 Enter IoT Hub connection string and connect to Azure IoT explorer 29](#_Toc120718984)

[6.3 Run device 30](#_Toc120718985)

[7 Additional Links 33](#_Toc120718986)

[8 Troubleshooting 34](#_Toc120718987)

# Document information

## Revision history

|  |  |  |
| --- | --- | --- |
| **Version** | **Date** | **Description of change** |
| V1.0.0 | 2022-11-30 | Initial Release |

# Introduction

This document describes how to connect **W5500-EVB-Pico** running **Windows 10** with Azure IoT SDK. This multi-step process includes:

* Configuring Azure IoT Hub
* Registering your IoT device
* Provisioning your devices on Device Provisioning service
* Build and deploy Azure IoT SDK on device

# Prerequisites

## Prepare your development environment

### Tools installation

**Windows 10** was used during preparation of this guide document. Linux and MacOS user should use compatible software, hardware-wise there is no difference. Please refer to the guide in *section 3.1.3* to find instructions for installing toolchain on Linux and MacOS.

1) Install the Toolchain

To build you will need to install extra tools below.

* [ARM GCC compiler](https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads)
* [CMake](https://cmake.org/download/)
* [Build Tools for Visual Studio](https://visualstudio.microsoft.com/downloads/#build-tools-for-visual-studio-2019)
* [Python 3.9](https://www.python.org/downloads/windows/)
* [Git](https://git-scm.com/download/win)
* [Visual Studio Code](https://code.visualstudio.com/download)

Download the executable installer for each of these from the links above, and then carefully follow the instructions in the following sections to install all six packages on to your Windows computer.

1. Install ARM GCC compiler

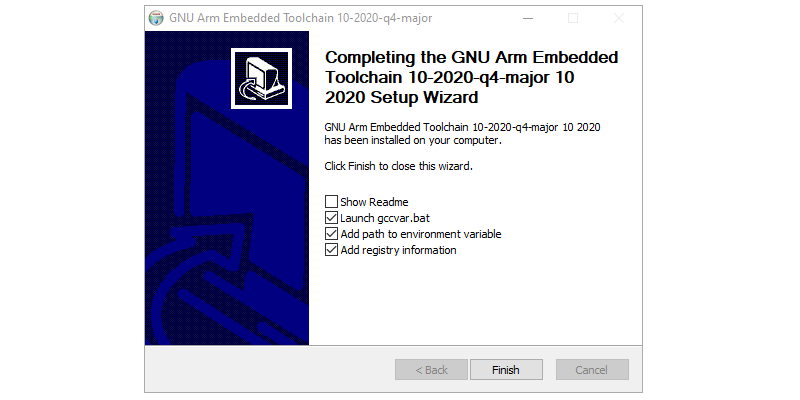


Figure . Install ARM GCC compiler

During installation you should check the box to register the path to the ARM compiler as an environment variable in the Windows shell when prompted to do so.

1. Install CMake

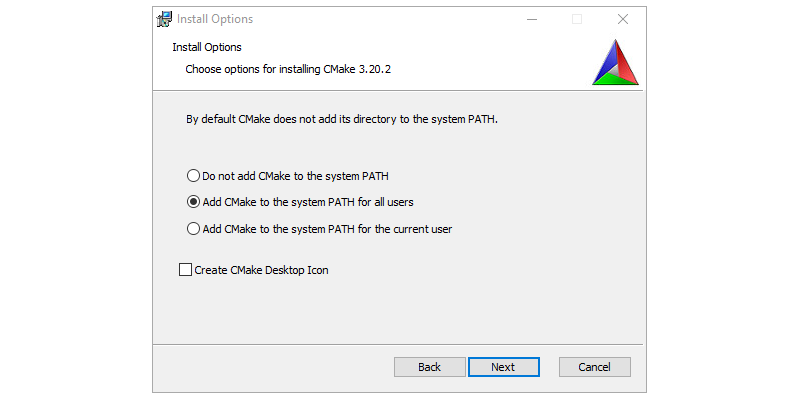


Figure . Install CMake

During the installation add CMake to the system **PATH** for all users when prompted by the installer.

1. Install Build Tools for Visual Studio

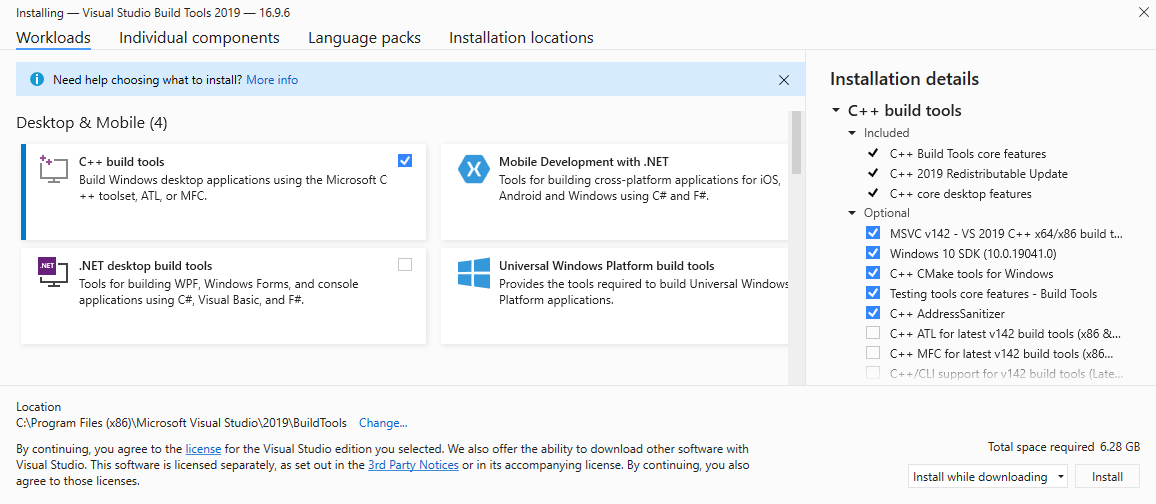


Figure . Install Build Tools for Visual Studio

When prompted by the Build Tools for Visual Studio installer you need to install the C++ build tools only.

1. Install Python 3.9

During the installation, ensure that it's installed 'for all users' and add Python 3.9 to the system **PATH** when prompted by the installer. You should additionally disable the **MAX\_PATH** length limit when prompted at the end of the Python installation.

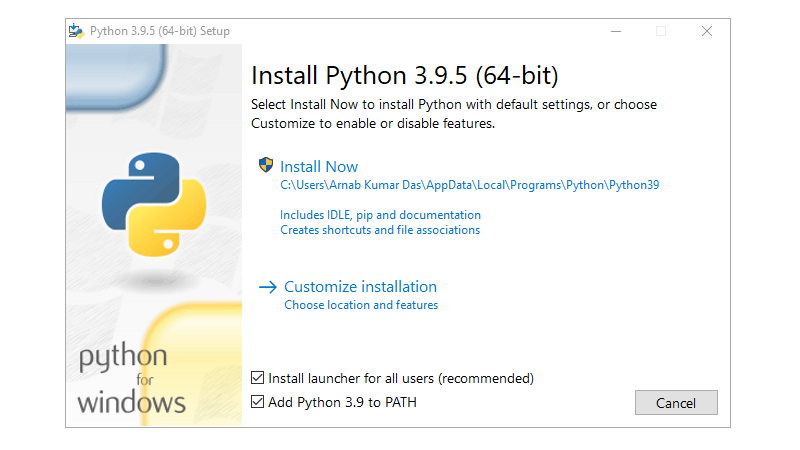


Figure . Install Python

1. Install Git

When installing Git you should ensure that you change the default editor away from vim.

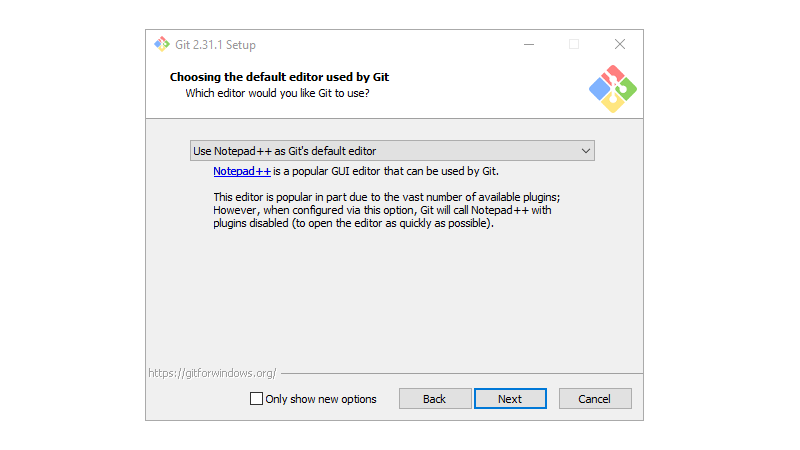


Figure . Install Git

1. Install Visual Studio Code

During the installation add Visual Studio Code to the system **PATH**.

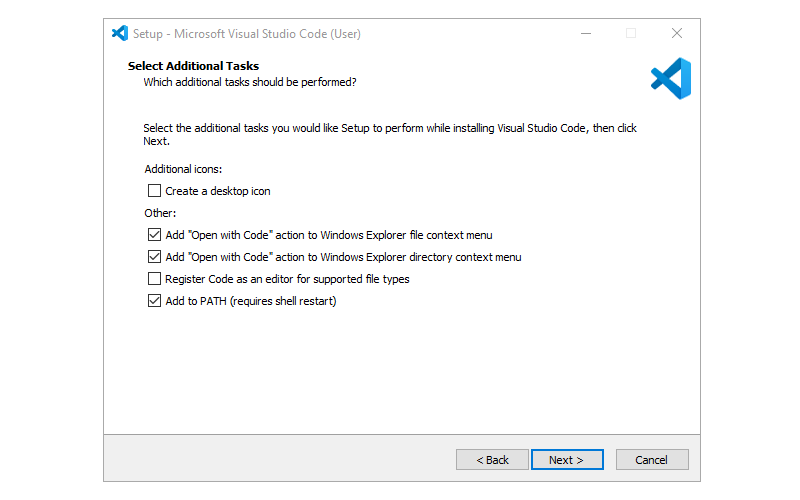


Figure . Install Visual Studio Code

2) Clone the **Raspberry Pi Pico SDK** and WIZnet's **Azure IoT SDK example** using below commands

* Raspberry Pi Pico SDK : <https://github.com/raspberrypi/pico-sdk>
* Azure IoT SDK example : <https://github.com/Wiznet/RP2040-HAT-AZURE-C>

|  |
| --- |
| // create a project directory  D:\>mkdir RP2040  D:\>cd RP2040  // get the SDK  D:\RP2040> git clone -b master https://github.com/raspberrypi/pico-sdk.git  D:\RP2040> cd pico-sdk  D:\RP2040\pico-sdk> git submodule update --init  // get the example  D:\RP2040\pico-sdk> cd ..  D:\RP2040> git clone -b main https://github.com/Wiznet/RP2040-HAT-AZURE-C.git  D:\RP2040> cd RP2040-HAT-AZURE-C  D:\RP2040\RP2040-HAT-AZURE-C> git submodule update --init |

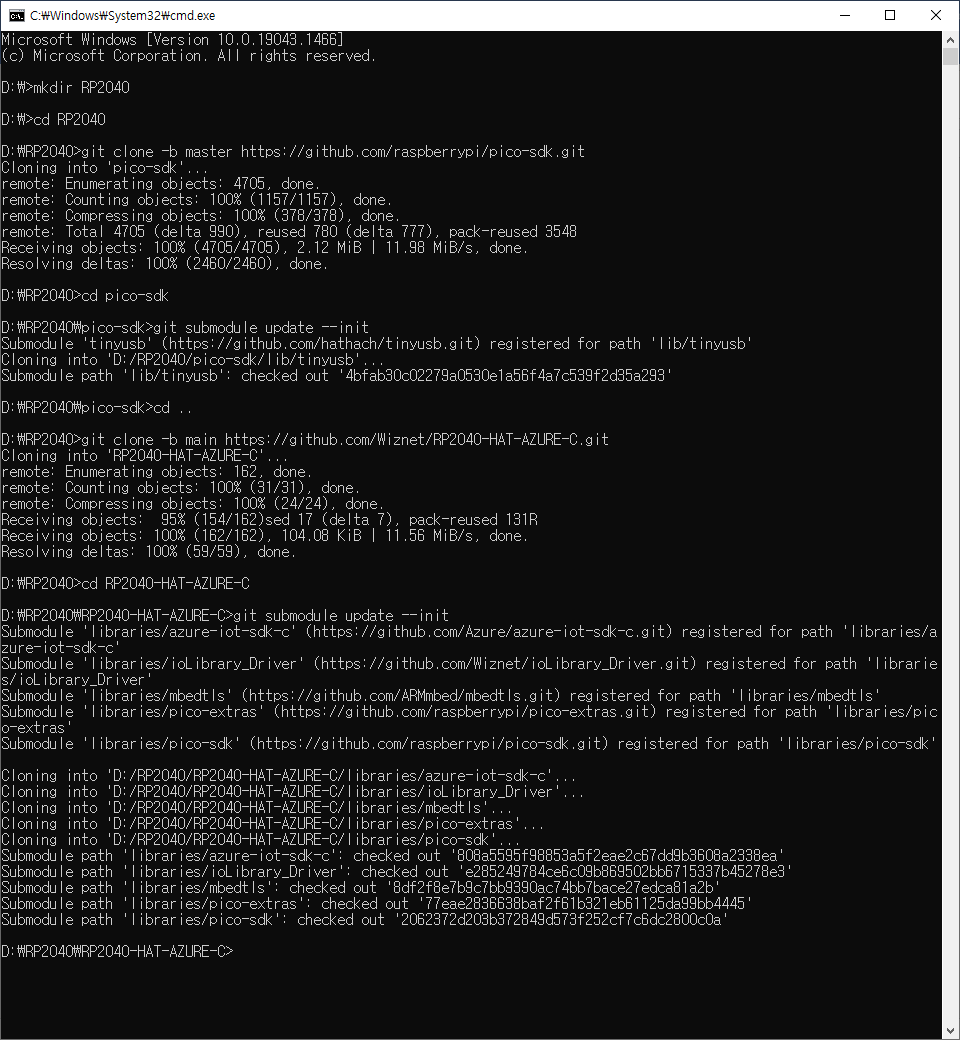


Figure . Get the SDK and example

3) Set up Visual Studio Code

1. Open a new Visual Studio 2019 Developer Command Prompt
2. Run the below command to open Visual Studio Code

|  |
| --- |
| D:> code -n |

1. Opening Visual Studio Code from Developer Command Prompt
2. Open Extensions

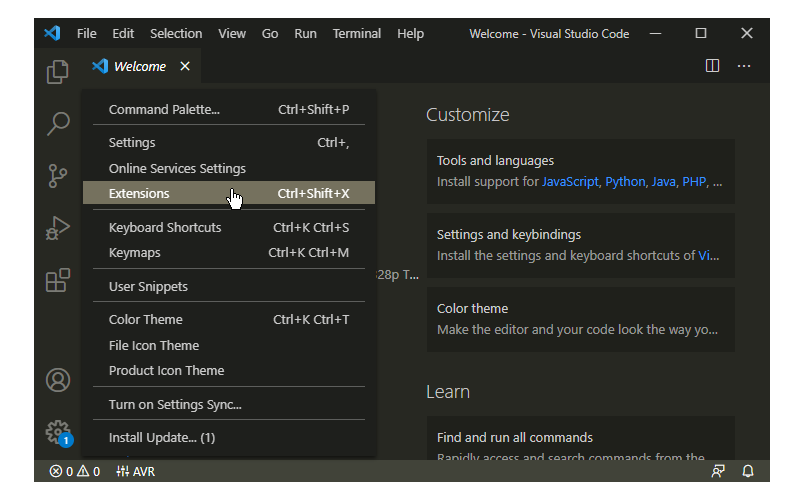


Figure . Install Extensions in Visual Studio Code

1. Install some tools
   * CMake Tools
   * C/C++
2. Open CMake Tools Extension Settings
3. Scroll down and set up some items
   * Add Cmake: Configure Environment Item as **PICO\_SDK\_PATH**
   * Add Cmake: Configure Environment Value as **D:\RP2040\pico-sdk**
   * Add Cmake: Generator as **NMake Makefiles**

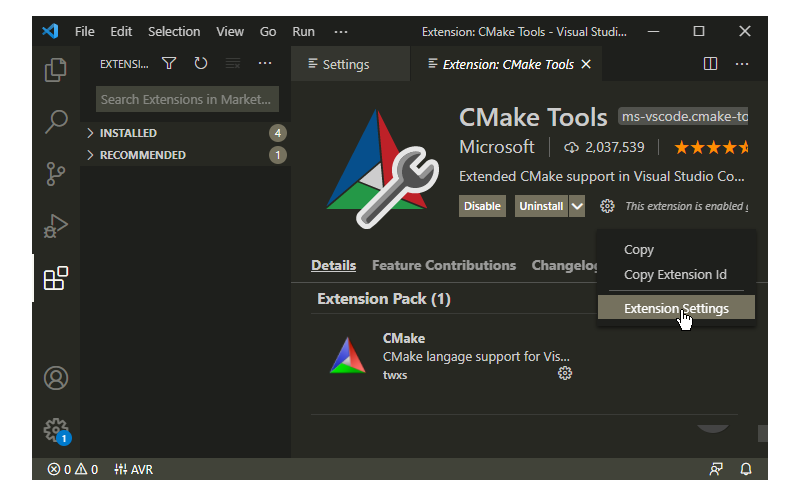


Figure . CMake Tools Extension Settings in Visual Studio Code

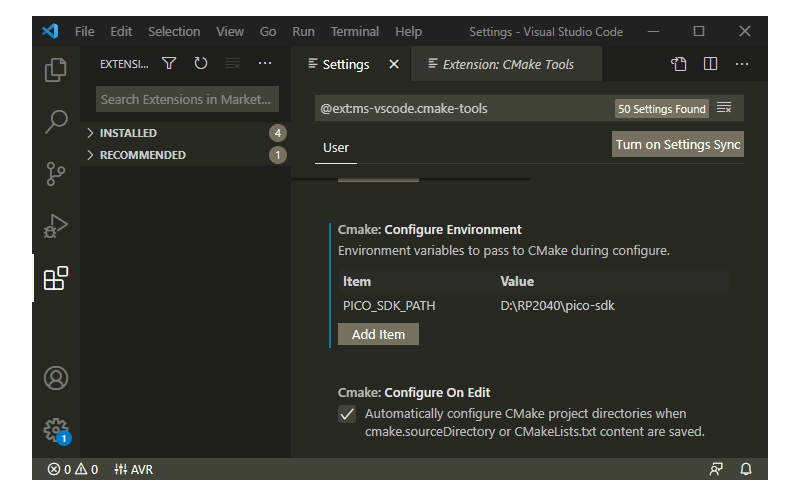


Figure . Add CMake Configure Environment path

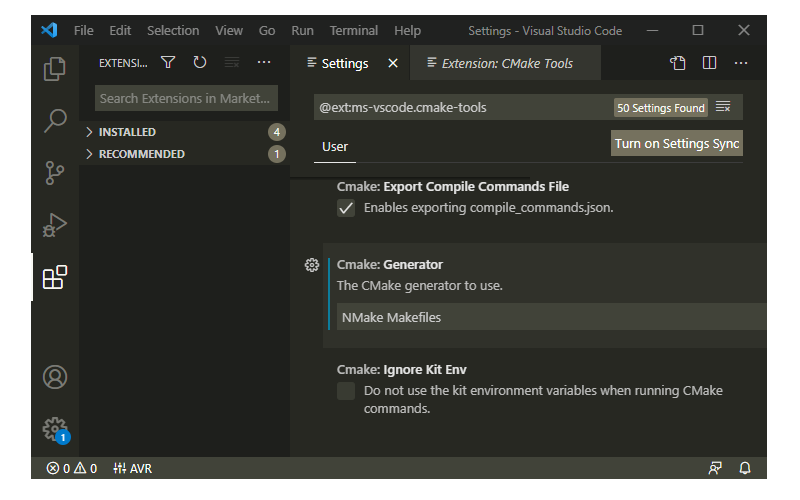


Figure . Add CMake Generator name

1. Add folder RP2040-HAT-AZURE-C to Visual Studio Code
2. Visual Studio Code will scan for kits
3. Select 'Yes' when asked: Would you like to configure project RP2040-HAT-AZURE-C?
4. Select 'Yes' if you like to configure the project upon opening
5. Click CMake in the bottom bar to select the kit RP2040-HAT-AZURE-C
6. Select Debug / Release based on your preference
7. Visual Studio Code will save all file and configure the project
8. Click on Build to build all examples, if no error Build will finish with exit code 0

### Other software required to develop and debug applications for the device

Serial terminal program is required for operation check and debugging.

* [Tera Term](https://osdn.net/projects/ttssh2/releases/)

You may use your preferred serial terminal program.

### Additional software references

Refer to the '**9.2. Building on MS Windows**' section of '**Getting started with Raspberry Pi Pico**' guide document below to set up the development environment.

* [Getting started with Raspberry Pi Pico](https://datasheets.raspberrypi.com/pico/getting-started-with-pico.pdf)

## Setup your IoT Hub

Refer to the '**Create an IoT hub**' section of '**Quickstart: Set up the IoT Hub Device Provisioning Service with the Azure portal**' guide below to set up the IoT Hub.

* [Quickstart: Set up the IoT Hub Device Provisioning Service with the Azure portal](https://docs.microsoft.com/en-us/azure/iot-dps/quick-setup-auto-provision)

## Provision your device over DPS

Refer to the '**Create a new IoT Hub Device Provisioning Service**' section of '**Quickstart: Set up the IoT Hub Device Provisioning Service with the Azure portal**' guide below to set up the DPS.

* [Quickstart: Set up the IoT Hub Device Provisioning Service with the Azure portal](https://docs.microsoft.com/en-us/azure/iot-dps/quick-setup-auto-provision)

After completing the DPS settings according to the guide document above, some additional settings are required.

1) Generate the certificate in X.509 format required when provisioning the device

Generate the certificate in X.509 format that is required when provisioning the device using OpenSSL.

For the OpenSSL commands to generate the certificate in X.509 format, refer to the following.

|  |
| --- |
| /\* Generate the certificate \*/  OpenSSL> genpkey -out [key name].key -algorithm RSA -pkeyopt rsa\_keygen\_bits:2048  OpenSSL> req -new -key [key name].key -out [csr name].csr  OpenSSL> x509 -req -days 365 -in [csr name].csr -signkey [key name].key -out [crt name].crt  OpenSSL> x509 -in [crt name].crt -out [pem name].pem -outform PEM  // e.g.  OpenSSL> genpkey -out prov\_device1.key -algorithm RSA -pkeyopt rsa\_keygen\_bits:2048  OpenSSL> req -new -key prov\_device1.key -out prov\_device1.csr  OpenSSL> x509 -req -days 365 -in prov\_device1.csr -signkey prov\_device1.key -out prov\_device1.crt  OpenSSL> x509 -in prov\_device1.crt -out prov\_device1.pem -outform PEM |

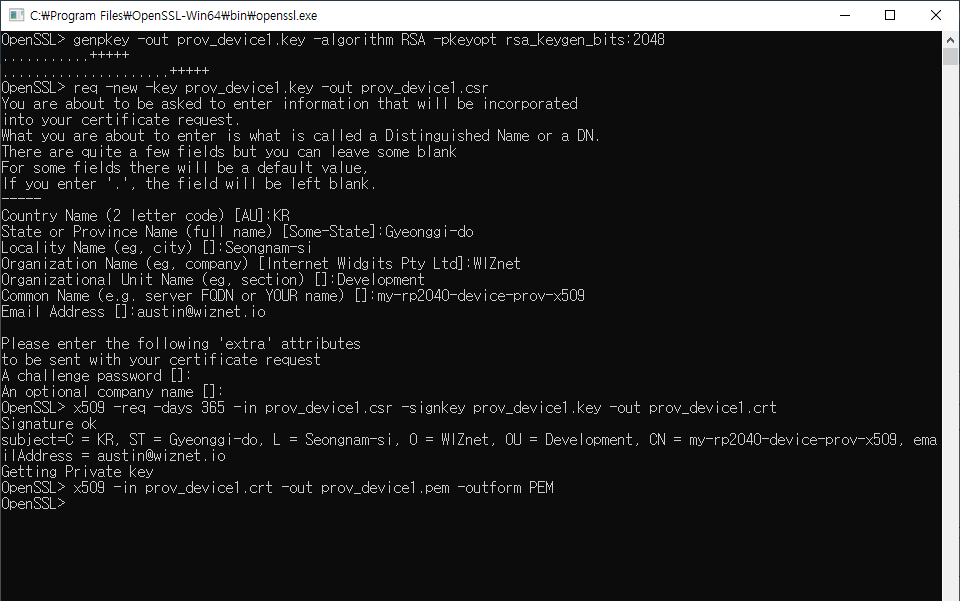
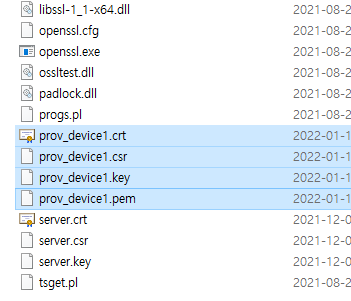


Figure . generate certificate in X.509



If OpenSSL is not installed, you can download it from the link below and install it, or you can use another preferred program to generate the certificate in X.509 format.

* [OpenSSL](https://www.openssl.org/source/)

2) Set up Enrollment and register the generated the certificate in X.509 format

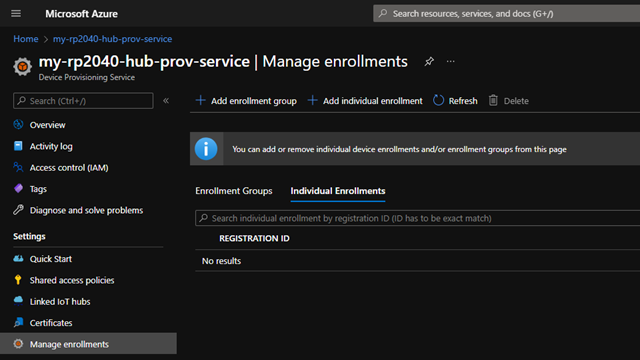


Figure . Add Individual Enrollment

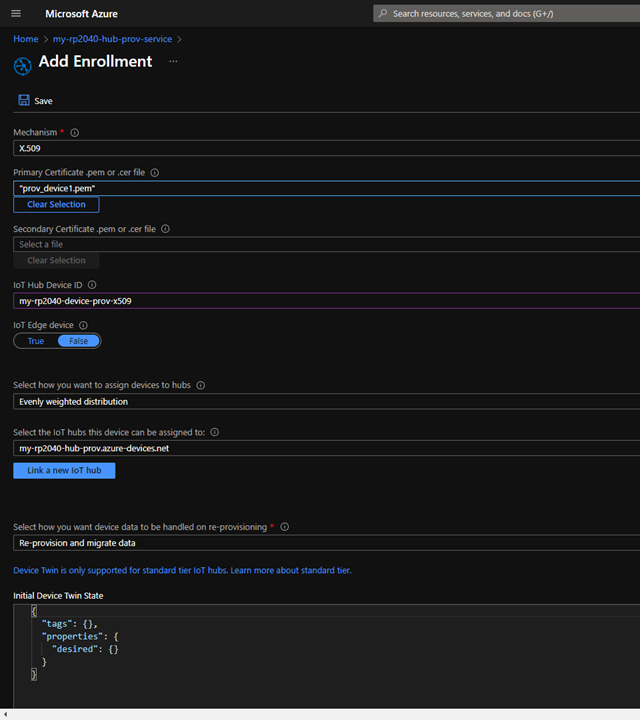


Figure . Set up enrollment items

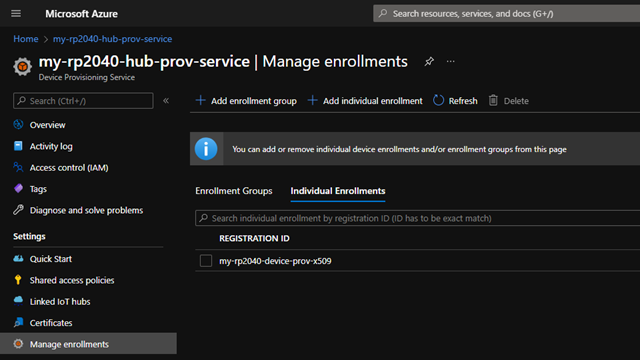


Figure . Registered Individual Enrollment

# Prepare your Device

## Connect with ethernet cable

Connect ethernet cable to device ethernet port.

## Connect with 5 pin micro USB cable.

Connect device to desktop or laptop using 5 pin micro USB cable.

# Build SDK and Run Samples

## Select sample application

Since you need to use the **prov\_dev\_client\_ll\_sample** sample application, uncomment the following in '**main.c**' in the '**RP2040-HAT-AZURE-C\examples\**' directory to configure the sample application.

|  |
| --- |
| /\*\*  \* -----------------------------------------------------------------------  \* Macros  \* -----------------------------------------------------------------------  \*/  // The application you wish to use should be uncommented  //  //#define APP\_TELEMETRY  //#define APP\_C2D  //#define APP\_CLI\_X509  #define APP\_PROV\_X509 |

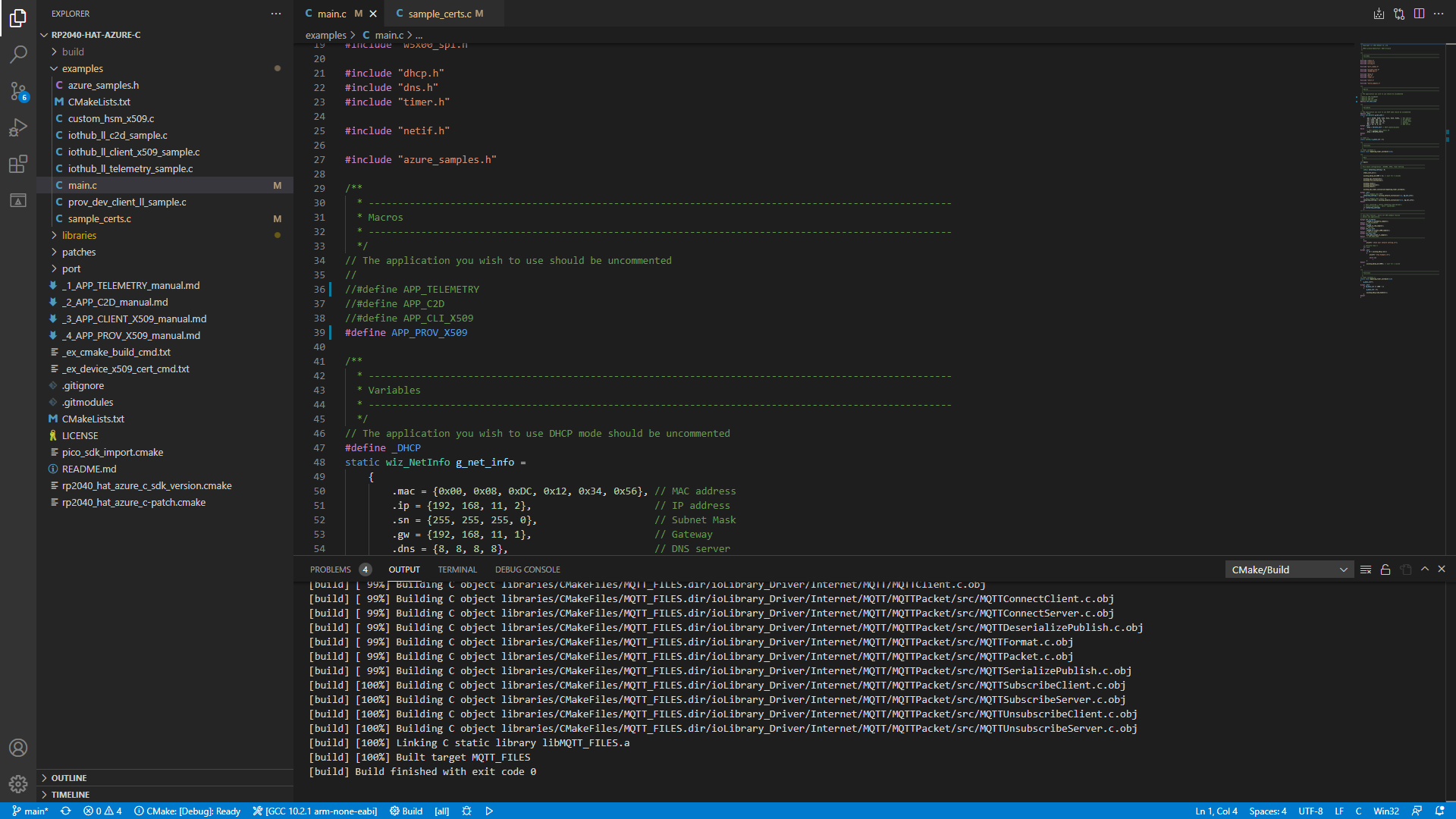


Figure . Set up sample application

## Enter certificate and DPS related information

For provisioning the device to the IoT Hub through DPS and connecting to IoT Hub, enter **ID scope**, **common name**, and **certificate in X.509 format** generated and registered in *section 3.3*.

The ID scope can be checked in the Device Provisioning Service set in the Azure Portal.

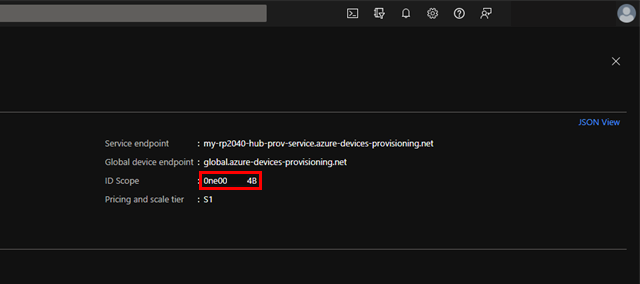


Figure . ID scope

The common name is one of the information entered when generating the certificate in X.509 format in OpenSSL.

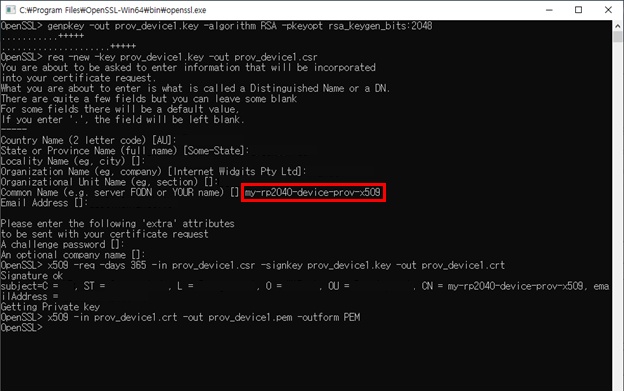


Figure . common name

Set up the above information in '**sample\_certs.c**' in the '**RP2040-HAT-AZURE-C\examples\**' directory.

|  |
| --- |
| const char pico\_az\_id\_scope[] = "[ID Scope]";  const char pico\_az\_COMMON\_NAME[] = "[custom-hsm-device]";  const char pico\_az\_CERTIFICATE[] =  "-----BEGIN CERTIFICATE-----""\n"  "-----END CERTIFICATE-----";  const char pico\_az\_PRIVATE\_KEY[] =  "-----BEGIN PRIVATE KEY-----""\n"  "-----END PRIVATE KEY-----"; |

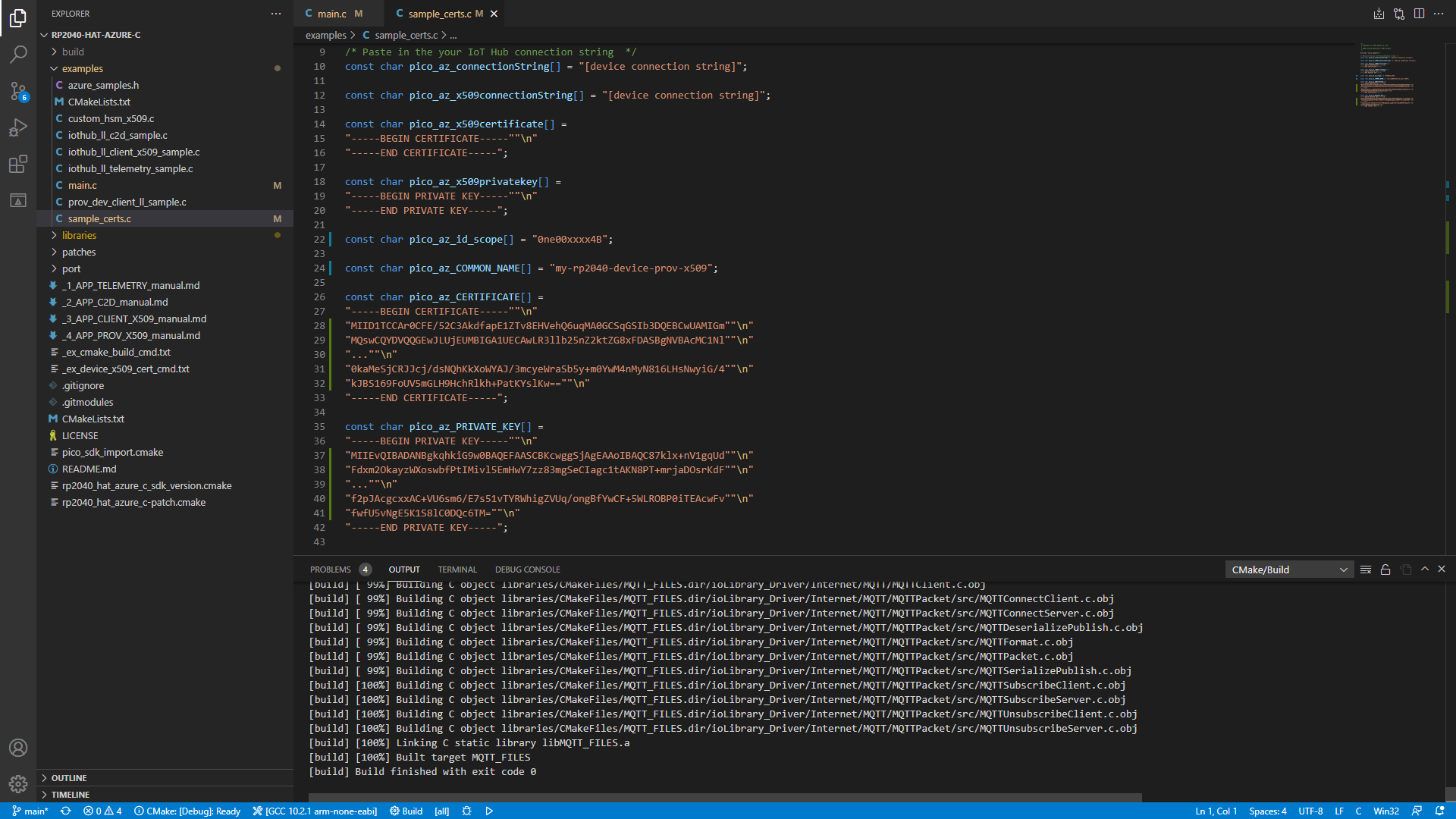


Figure . Set up 'sample\_certs.c'

## Build example

1) After completing the Azure IoT SDK example configuration, click 'build' in the status bar at the bottom of Visual Studio Code or press the 'F7' button on the keyboard to build.

2) When the build is completed, '**main.uf2**' is generated in '**RP2040-HAT-AZURE-C\build\examples\**' directory.

## Upload firmware

1) While pressing the **BOOTSEL** button of the device power on the board, the USB mass storage '**RPI-RP2**' is automatically mounted.

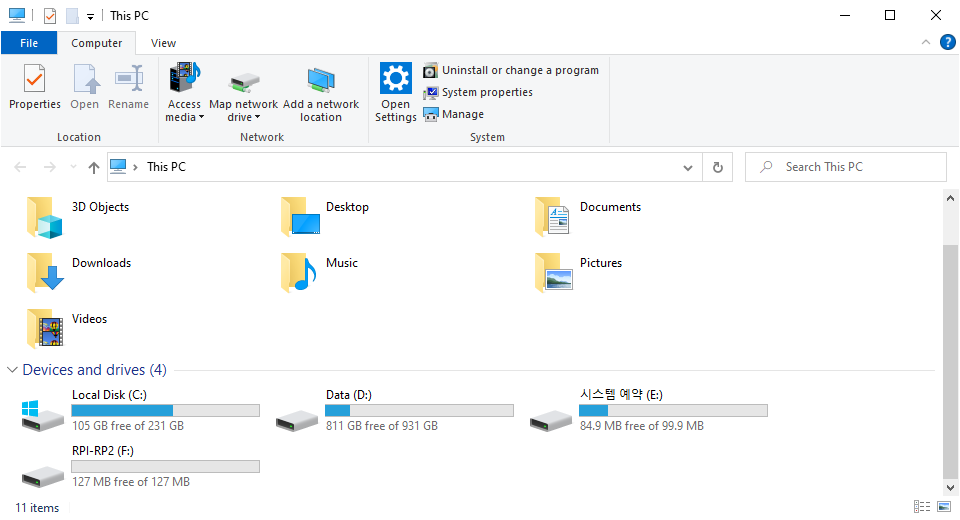


Figure . Automatically mounted USB mass storage 'RPI-RP2'

2) Drag and drop '**main.uf2**' onto the USB mass storage device '**RPI-RP2**'.

## Run sample application

First, connect to the serial COM port of the device with terminal program.

When connecting to the serial COM port of the device, use following settings to set up the serial port.

* Baud rate : 115,200
* Data bit : 8
* Parity bit : none
* Stop bit : 1
* Flow control : none

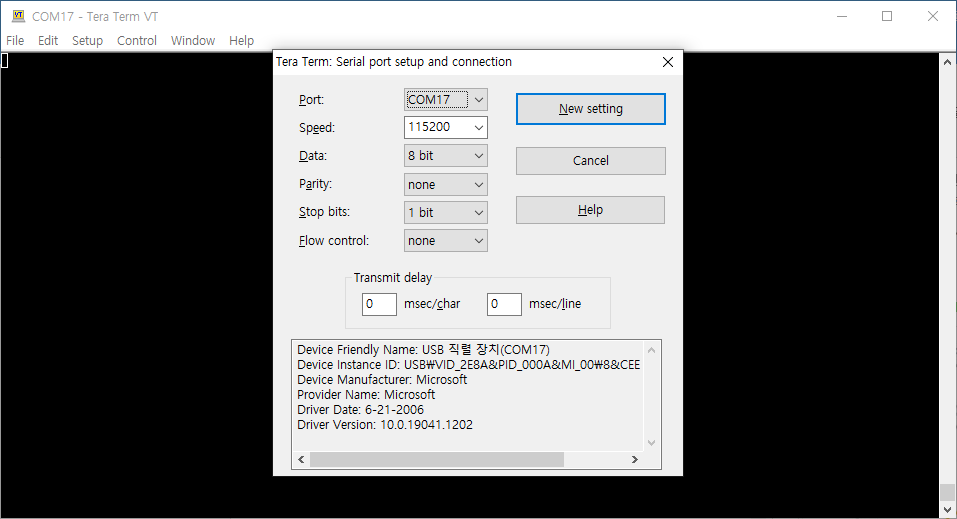


Figure . set up serial port

If sample application is running normally, in terminal you should be able to see the device network information, registration to the IoT Hub through DPS, and sending data from the device to the IoT Hub.

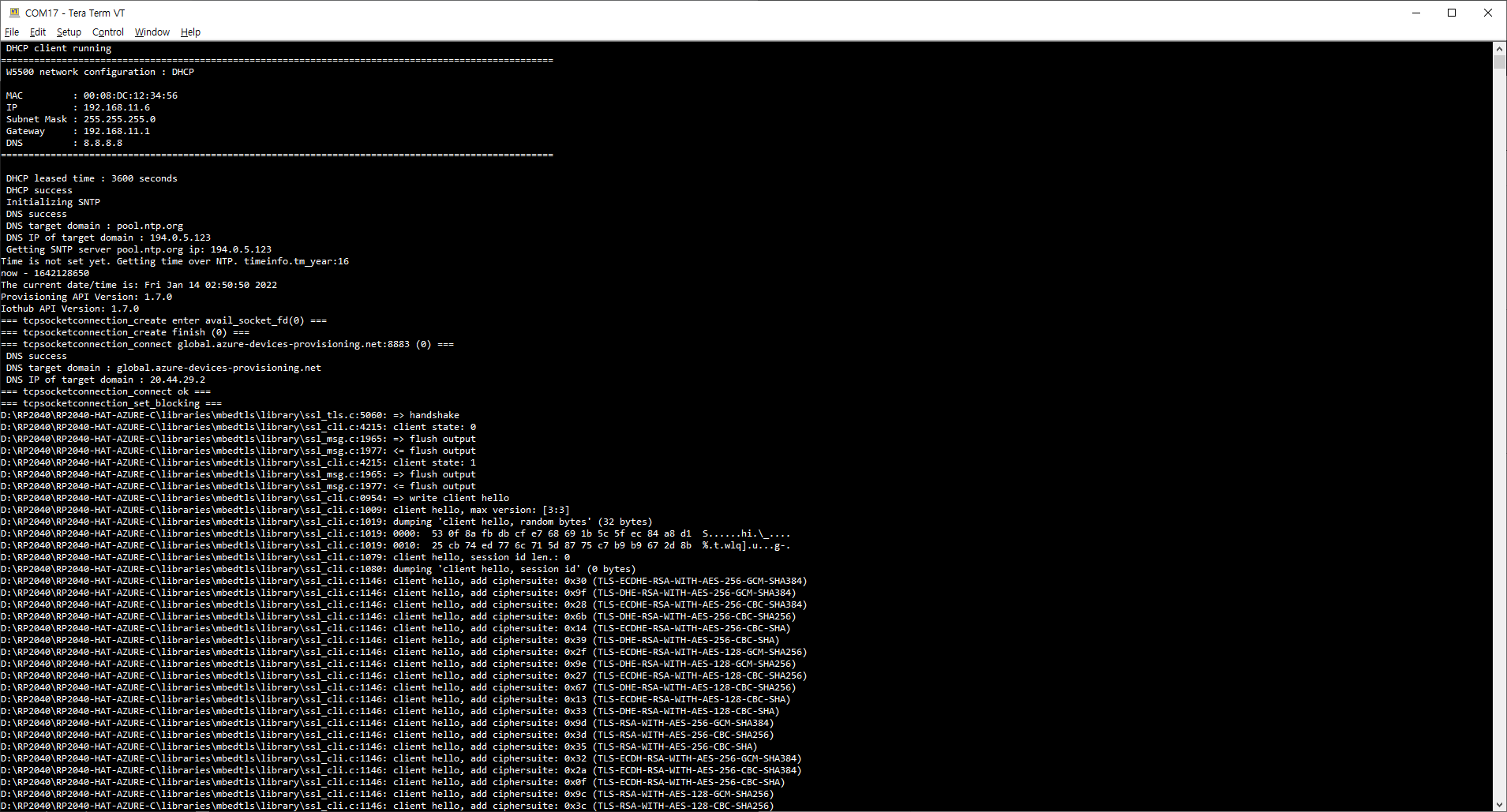


Figure . Device network information

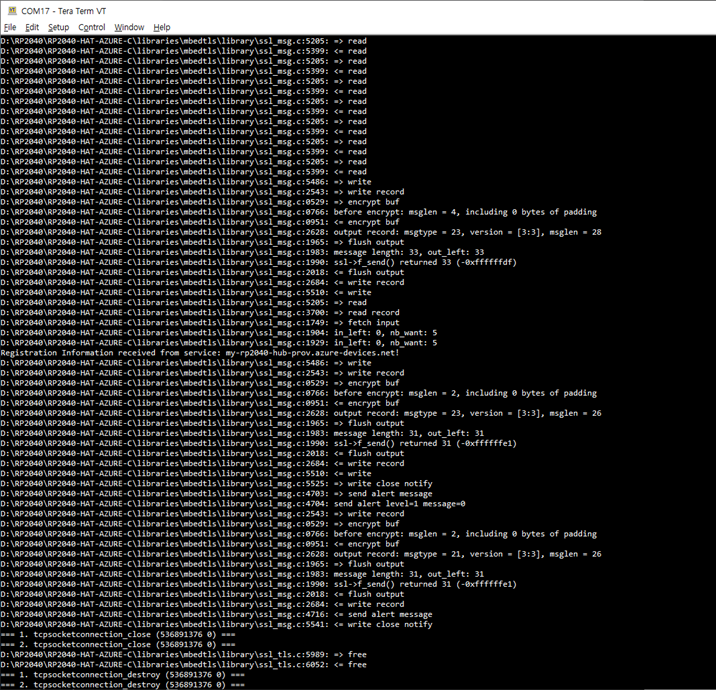


Figure . Registration to IoT Hub through DPS

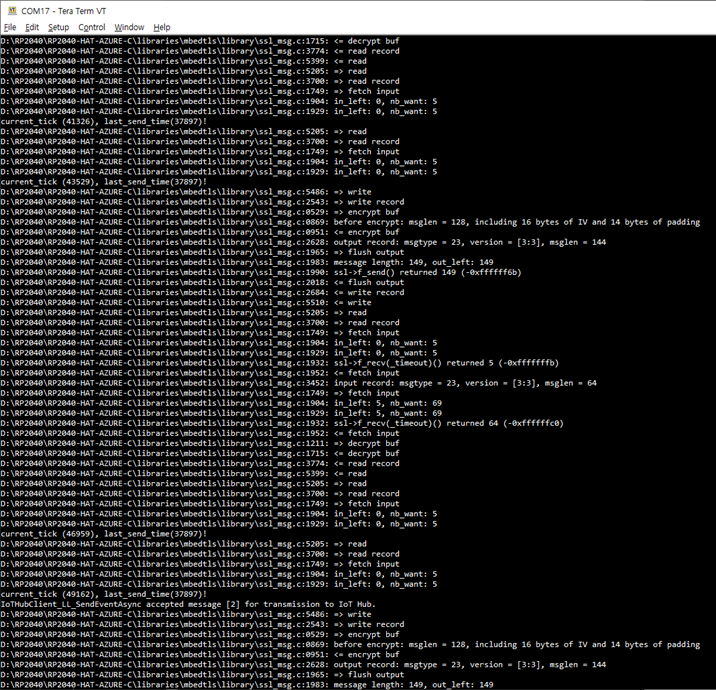


Figure . Sending data from device to IoT Hub

# Integration with Azure IoT Explorer

## Run Azure IoT explorer

Run Azure IoT explorer.

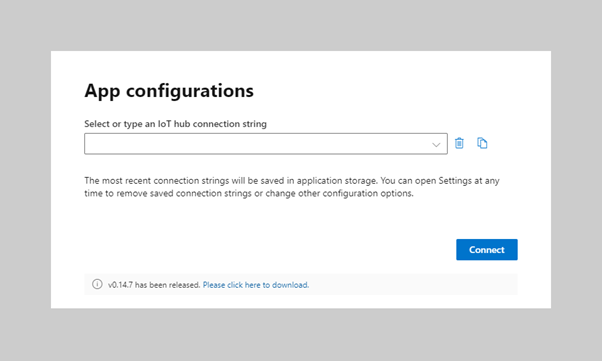


Figure . Run Azure IoT explorer

If Azure IoT explorer is not installed, download the latest version Azure IoT explorer from the link below and install it.

* [Azure IoT explorer](https://github.com/Azure/azure-iot-explorer/releases)

## Enter IoT Hub connection string and connect to Azure IoT explorer

Enter the connection string of the IoT Hub set up in *section 3.2* and press the connect button to connect to Azure IoT explorer.

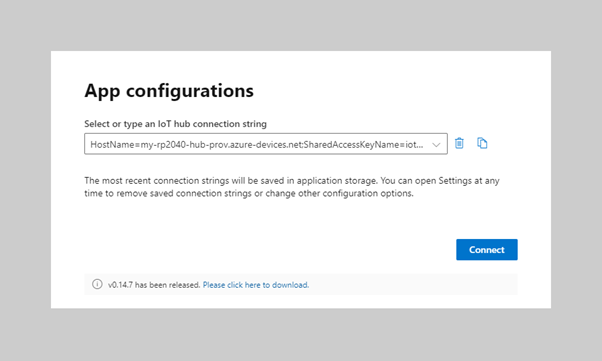


Figure . Enter connection string of IoT Hub

## Run device

If the device is running normally, in Azure IoT explorer you should be able to see the device registering to the IoT Hub through DPS, and receiving data sent from the device to the IoT Hub.

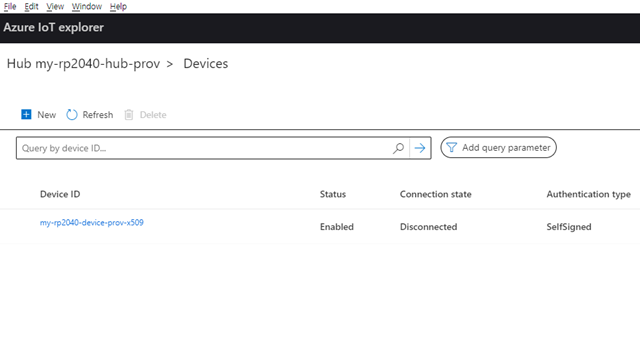


Figure . Registered device

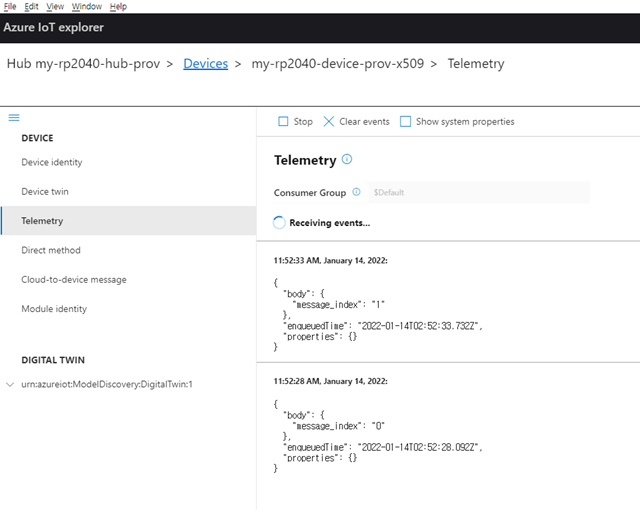


Figure . Receive data sent from device to IoT Hub

# Additional Links

Basic example of the W5500-EVB-Pico is also provided. If necessary, please refer to the example of the link below.

* [RP2040-HAT-C](https://github.com/Wiznet/RP2040-HAT-C)

# Troubleshooting

If you have any questions or problems while testing W5500-EVB-Pico examples, please post them at the links below.

* [WIZnet Developer Forum](https://forum.wiznet.io/)
* [RP2040-HAT-C Issues](https://github.com/Wiznet/RP2040-HAT-C/issues)
* [RP2040-HAT-AZURE-C Issues](https://github.com/Wiznet/RP2040-HAT-AZURE-C/issues)