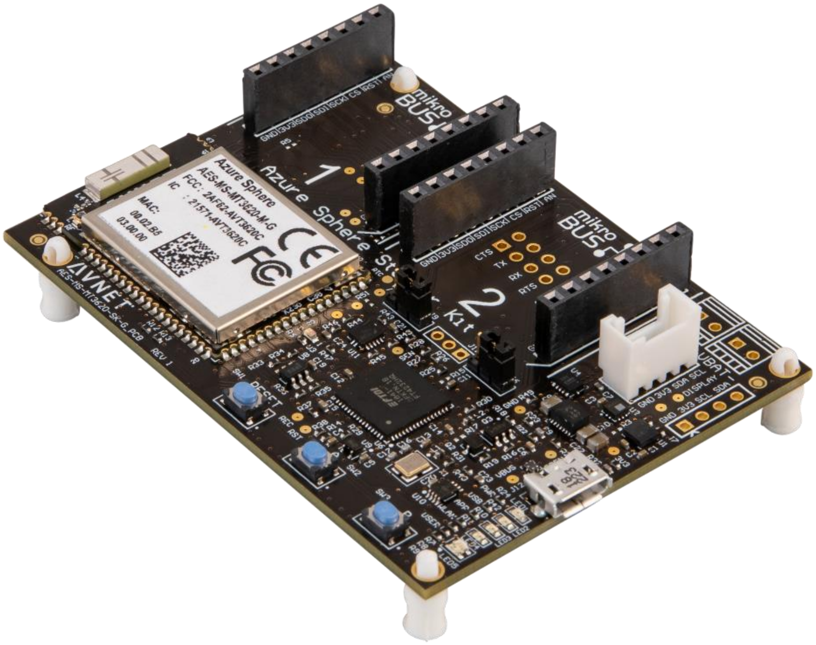
Avnet Starter Kit Plug and Play

Getting Started Guide



Azure Sphere SDK:	20.11
Document Version:	v1
Date:	30th December 2020

NOTICE OF DISCLAIMER: Avnet is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Avnet makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Avnet expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Introduction

This document will walk the reader through all the steps necessary to run the Avnet Starter Kit Plug and Play application on an Avnet Azure Sphere Starter Kit. The document is broken down into sections so that the reader can skip over sections that they are familiar with our may have completed already.

If you are new to Azure Sphere, it's recommended that you work through the entire document. If you're already familiar with Azure Sphere and Azure Sphere development, you should skip to the section titled "Pulling the application source code and configuring your application."

- Installing the Azure Sphere Development tools
- Preparing your device for development
- Pulling the application source code and configuring your application
- Running the application

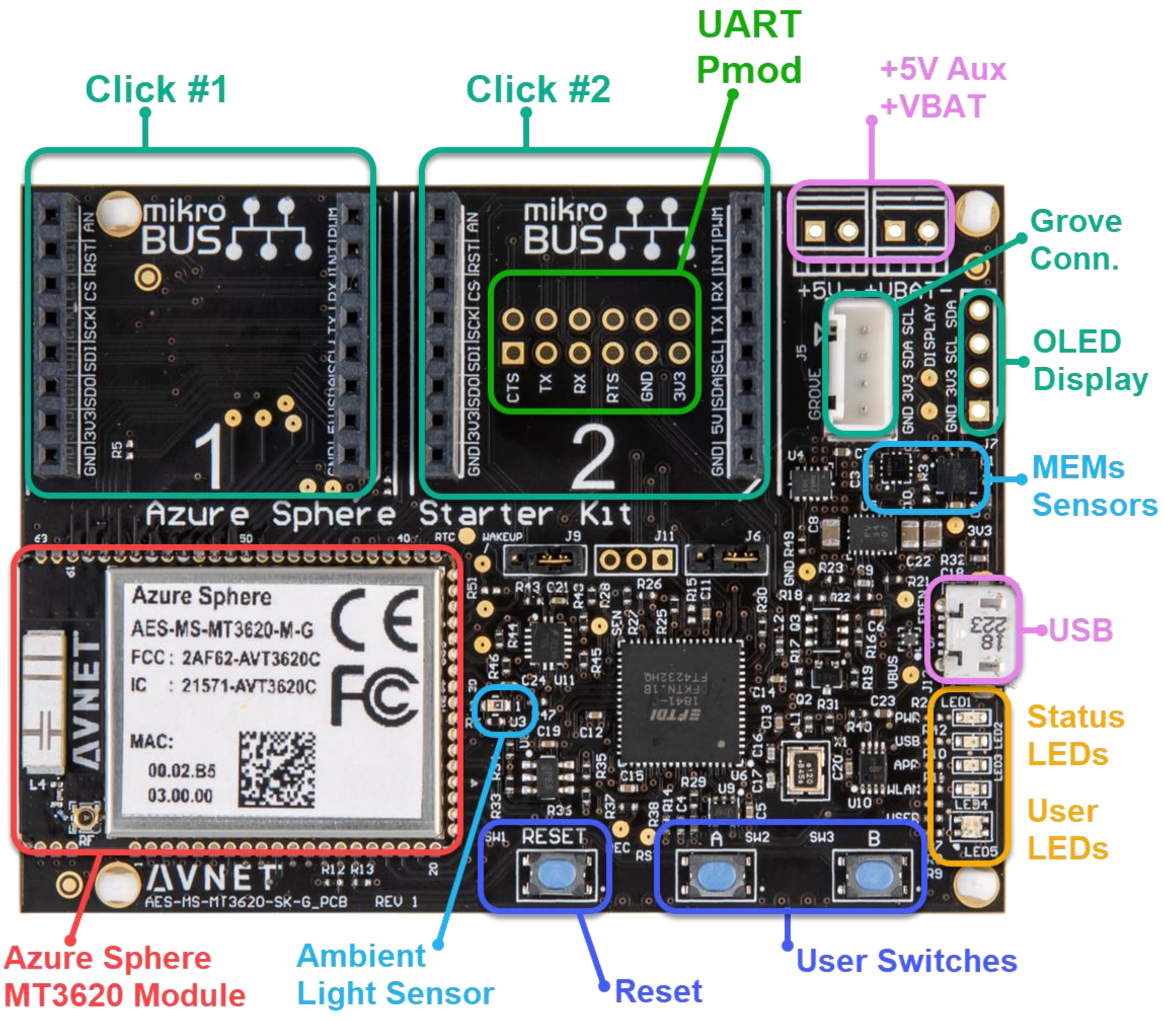
Avnet Azure Sphere Starter Kit Overview

The Avnet Azure Sphere Starter Kit from Avnet Electronics Marketing provides engineers with a complete system for prototyping and evaluating systems based on the MT3620 Azure Sphere device.

The Avnet Azure Sphere MT3620 Starter Kit supports rapid prototyping of highly secure, end-to-end IoT implementations using Microsoft's Azure Sphere. The small form-factor carrier board includes a production-ready MT3620 Sphere module with Wi-Fi connectivity, along with multiple expansion interfaces for easy integration of off-the-shelf sensors, displays, motors, relays, and more.

The Starter Kit includes Avnet's MT3620 Module. Having the module on the Starter Kit means that you can do all your development work for your IoT project on the Starter Kit and then easily migrate your Azure Sphere Application to your custom hardware design using Avnet's MT3620 Module.

Note that there are currently two versions of the Starter Kit Rev1 and Rev2. Either board will work with this example application.



Avnet Azure Sphere Starter Kit

Table of Contents

Installing the Azure Sphere Development tools

Azure Sphere Development Environments (Windows/Linux/Mac)

Starting with the 19.11 SDK release, Azure Sphere development is now supported under Linux, Windows (Visual Studio), and Windows (Visual Studio Code) environments. This document focuses only on the Windows with Visual Studio development environment. The application code will work just as well in the Linux environment.

At this time Azure Sphere development on Mac is not supported, although I have heard of some users setting up an Ubuntu VM on Mac and having success.

Objectives

The objectives of this section are to install the tools required to carry out Azure Sphere Development tasks.

- Install the Azure Sphere SDK
- Install Visual Studio 2019
- Install the Azure Sphere Visual Studio extension
- Update the Azure Sphere OS on your device
- Confirm that you're development PC can communicate with the Azure Sphere board

Requirements

Hardware

- A PC running Windows 10 Anniversary Update or later (Version 1607 or greater)
- An unused USB port on the PC
- An Avnet Azure Sphere Starter Kit
- A micro USB cable to connect the Starter Kit to your PC

Software

- Visual Studio 2019 version 16.4 or later (Enterprise, Professional, or Community version)
- Azure Sphere SDK 20.11 or the current SDK release

Other Requirements

- You will need Administrator rights on your PC to install the tools

Install Azure Sphere SDK

The first thing we need to install is the Azure Sphere SDK.

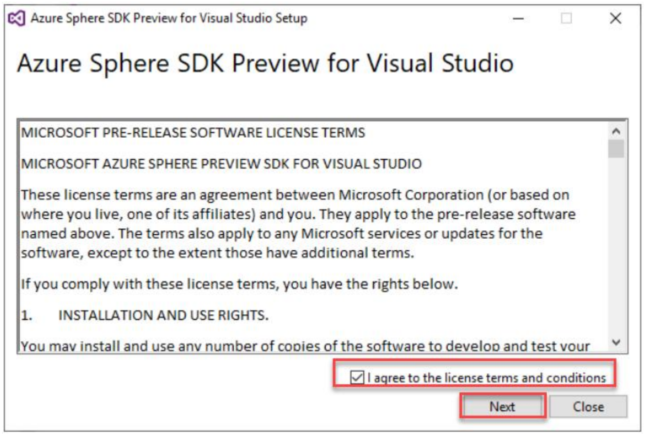
I've included all the details you need to install the SDK in this section, but I also wanted to provide links to the Microsoft SDK installation documentation and their troubleshooting guide in case you run into any issues.

- Microsoft SDK Installation Documentation
- Microsoft Azure Sphere Troubleshooting Guide

The Windows Azure Sphere SDK installer can be downloaded by clicking this link.

The SDK includes:

- The Azure Sphere Developer Command Prompt window, which can be found on the Windows Start Menu inside the Azure Sphere folder
- The azsphere.exe command-line utility for managing devices, images, and deployments
- Libraries for application development



Azure Sphere SDK Preview for Visual Studio

MICROSOFT PRE-RELEASE SOFTWARE LICENSE TERMS

MICROSOFT AZURE SPHERE PREVIEW SDK FOR VISUAL STUDIO

These license terms are an agreement between Microsoft Corporation (or based on where you live, one of its affiliates) and you. They apply to the pre-release software named above. The terms also apply to any Microsoft services or updates for the software, except to the extent those have additional terms.

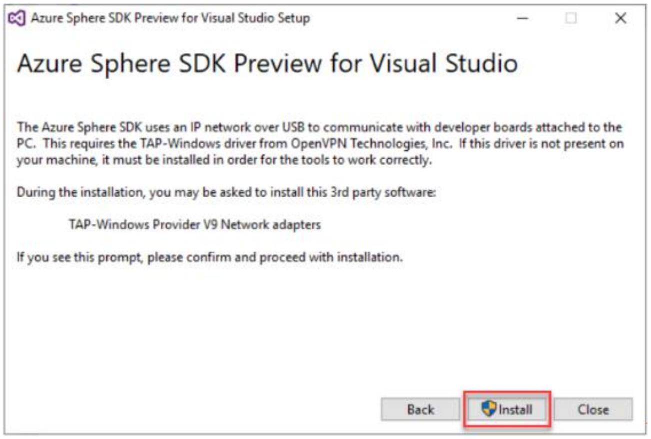
If you comply with these license terms, you have the rights below.

1. INSTALLATION AND USE RIGHTS.

You may install and use any number of copies of the software to develop and test your

☑ I agree to the license terms and conditions

Run the Azure Sphere SDK installer, agree to the license terms and select "Next."



Azure Sphere SDK Preview for Visual Studio

The Azure Sphere SDK uses an IP network over USB to communicate with developer boards attached to the PC. This requires the TAP-Windows driver from OpenVPN Technologies, Inc. If this driver is not present on your machine, it must be installed in order for the tools to work correctly.

During the installation, you may be asked to install this 3rd party software:

TAP-Windows Provider V9 Network adapters

If you see this prompt, please confirm and proceed with installation.

Click "Install" to begin the installation. Note the message on the dialog box about the TAP driver. The TAP driver enables an Ethernet connection over the USB serial port and is used to communicate to the Azure Sphere Starter Kit to/from the PC.

Updating the Azure Sphere SDK

The Azure Sphere team is on a quarterly release cadence. To update your SDK just download the new SDK installer and run it.

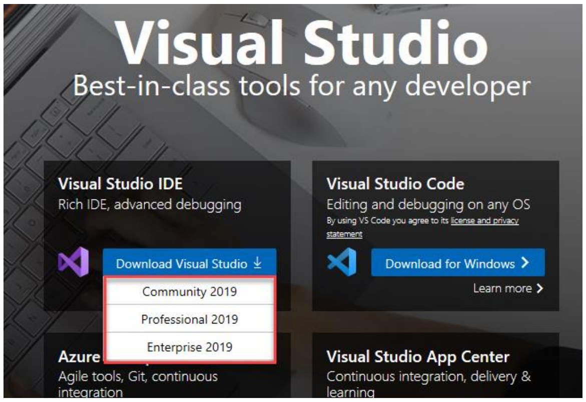
Potential Issues

Potential issues you may run into.

- Installer Hangs
  - If the installer hangs, try to reinstall the SDK. If it continues to fail try to uninstall the SDK.
- Other issues
  - Microsoft has been documenting potential issues and how to work around them. This page is continually being updated.
    - Troubleshooting installation and setup on Windows

Install Visual Studio 2019

Download the Visual Studio installer from Microsoft here. Download the distribution that matches your situation, Community, Professional, or Enterprise. All three will work for Azure Sphere Development.


Visual Studio
Best-in-class tools for any developer
Visual Studio IDE
Rich IDE, advanced debugging
Download Visual Studio
Community 2019
Professional 2019
Enterprise 2019
Visual Studio Code
Editing and debugging on any OS
By using VS Code you agree to its license and privacy statement
Download for Windows
Learn more
Azure
Agile tools, Git, continuous integration
Visual Studio App Center
Continuous integration, delivery & learning

After downloading the installer, go ahead and run the executable. The installer will download more files then present a window asking you which workloads you want to install.

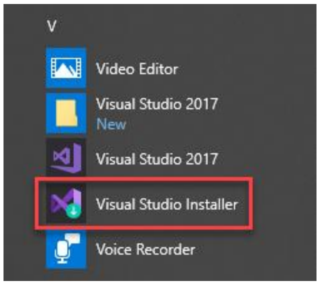
Feel free to install any additional workloads if you wish**, however no additional workloads are required for Azure Sphere development.**

Once you select any additional workloads that you want to install, click on the "Install" button in the bottom right corner of the window.

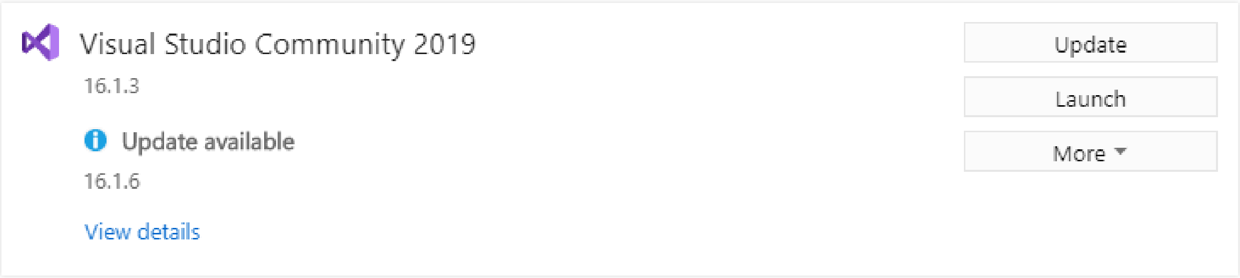
After the installation finishes, Visual Studio will launch and you'll be prompted to "Sign in" with a Microsoft account. You can check/verify that your email address is associated with a Microsoft account here.

Updating your Visual Studio Installation

Over time Microsoft will update the Visual Studio application. To update Visual Studio find the Visual Studio installer on the Start Menu.



Once the Visual Studio Installer if your installation is out of date, you'll see the option to "update."



Install Visual Studio Extension for the Azure Sphere SDK

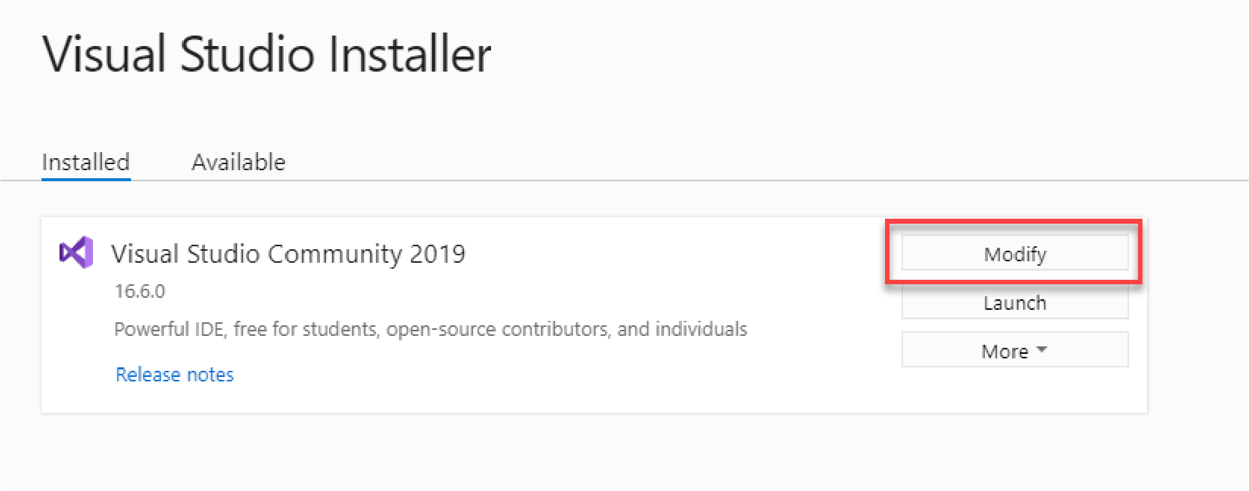
Beginning with the 20.04 release Microsoft moved the Visual Studio Azure Sphere SDK extensions out of the Azure Sphere SDK installer and into a Visual Studio Extension that must be installed separately. This was a positive change for Azure Sphere as now you can install the Azure Sphere SDK without any developer tools and access the azsphere command line tool.

Download the Extension

- Download the extension from here
- Once it's been downloaded find the VSIX installer file and double click on it
    - **Note:** We have seen this installer get downloaded as a *.zip file. If this happens to you simply replace the .zip extension with .vsix, then double click on the file to run the installer.

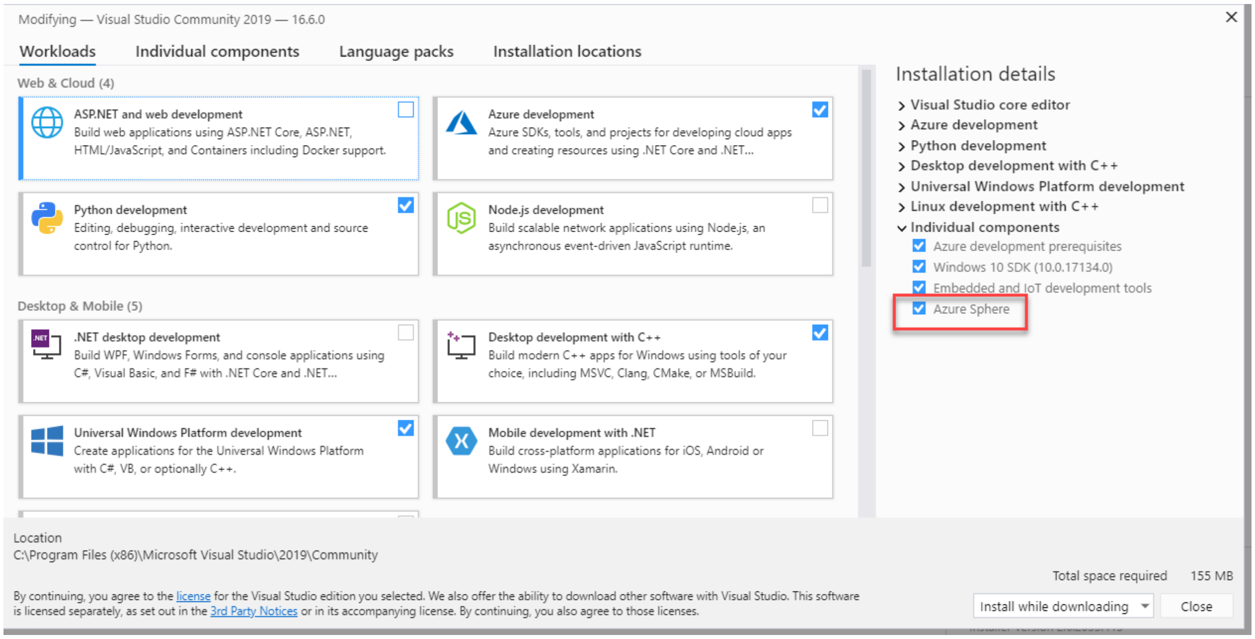
You can verify that the extension is correctly installed by running the Visual Studio Installer

- Click on the Modify button



Powerful IDE, free for students, open-source contributors, and individuals

- Verify that “Azure Sphere” is listed under the “Individual components” heading



Modifying — Visual Studio Community 2019 — 16.6.0

Web & Cloud (4)

□ ASP.NET and web development
Build web applications using ASP.NET Core, ASP.NET, HTML/JavaScript, and Containers including Docker support.

☑ Azure development
Azure SDKs, tools, and projects for developing cloud apps and creating resources using .NET Core and .NET...

☑ Python development
Editing, debugging, interactive development and source control for Python.

□ Node.js development
Build scalable network applications using Node.js, an asynchronous event-driven JavaScript runtime.

Desktop & Mobile (5)

□ .NET desktop development
Build WPF, Windows Forms, and console applications using C#, Visual Basic, and F# with .NET Core and .NET...

☑ Desktop development with C++
Build modern C++ apps for Windows using tools of your choice, including MSVC, Clang, CMake, or MSBuild.

☑ Universal Windows Platform development
Create applications for the Universal Windows Platform with C#, VB, or optionally C++.

□ Mobile development with .NET
Build cross-platform applications for iOS, Android or Windows using Xamarin.

Installation details

- › Visual Studio core editor
- › Azure development
- › Python development
- › Desktop development with C++
- › Universal Windows Platform development
- › Linux development with C++
- ⌄ Individual components
  - ☑ Azure development prerequisites
  - ☑ Windows 10 SDK (10.0.17134.0)
  - ☑ Embedded and IoT development tools
  - ☑ Azure Sphere

Location
C:\Program Files (x86)\Microsoft Visual Studio\2019\Community

By continuing, you agree to the license for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the 3rd Party Notices or in its accompanying license. By continuing, you also agree to those licenses.

Total space required 155 MB

Updating the Visual Studio Extensions

To update the extensions, just download the updated installer and run it. I’ve noticed, that this extension does not automatically get updated when updating Visual Studio. It’s recommended to check for a new extension installer on each Azure Sphere SDK release, as some new features may depend on the updated extension.

Update the Sphere OS

It is important that you start by updating your Starter Kit to the latest Azure Sphere OS.

Manually updating the OS will not be required in a deployment situation. When devices are deployed, all software updates, including the OS are performed Over-the-Air (OTA). Even in a development environment, we don’t have to manually update the OS. Anytime a device is connected to the internet, it will check in with the Azure Sphere Security Service to confirm that it’s running the correct OS version and if required the device *will* receive the latest OS version automatically OTA. (i.e., one could just configure the Wi-Fi on your device and wait for the OS update to happen, but we are going to manually force this update)

Note that when we manually update the OS, this is called a “recover” process, all applications and any Wi-Fi configurations on the device will be removed/erased. Not a big deal, but something to be aware of.

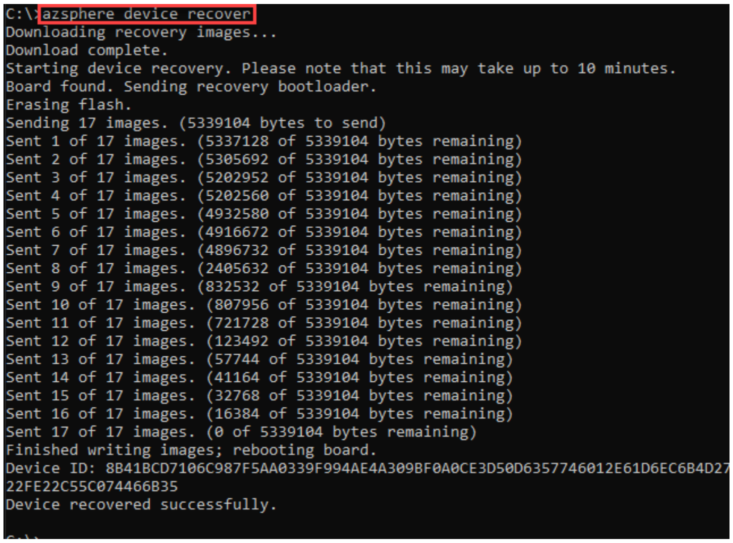
This is an easy process and takes just a few minutes.

1. Open the Azure Sphere CLI from your start menu.
   Start → Azure Sphere → Azure Sphere Developer Command Prompt

2. A terminal window will open with a prompt (This **azsphere.exe** command line utility is part of your SDK installation). The azsphere CLI is used to configure things on your Azure Sphere tenant and your Sphere device. You can learn more about this tool from the on-line documentation here, or you can just enter the **azsphere** command and see what the available options are. Feel free to explore!

From the command line type the following command: **azsphere device recover,** and hit the enter key. The recover command will start the process of downloading the current Sphere OS files to your PC and flashing each OS component to your Sphere device. Your output should look similar to the screenshot below.

When you perform this **recover** command, any applications on the device will be erased from Flash memory and all Wi-Fi networks that were previously configured will be removed.



```
C:\>azsphere device recover
Downloading recovery images...
Download complete.
Starting device recovery. Please note that this may take up to 10 minutes.
Board found. Sending recovery bootloader.
Erasing flash.
Sending 17 images. (5339104 bytes to send)
Sent 1 of 17 images. (5337128 of 5339104 bytes remaining)
Sent 2 of 17 images. (5305692 of 5339104 bytes remaining)
Sent 3 of 17 images. (5202952 of 5339104 bytes remaining)
Sent 4 of 17 images. (5202560 of 5339104 bytes remaining)
Sent 5 of 17 images. (4932580 of 5339104 bytes remaining)
Sent 6 of 17 images. (4916672 of 5339104 bytes remaining)
Sent 7 of 17 images. (4896732 of 5339104 bytes remaining)
Sent 8 of 17 images. (2405632 of 5339104 bytes remaining)
Sent 9 of 17 images. (832532 of 5339104 bytes remaining)
Sent 10 of 17 images. (807956 of 5339104 bytes remaining)
Sent 11 of 17 images. (721728 of 5339104 bytes remaining)
Sent 12 of 17 images. (123492 of 5339104 bytes remaining)
Sent 13 of 17 images. (57744 of 5339104 bytes remaining)
Sent 14 of 17 images. (41164 of 5339104 bytes remaining)
Sent 15 of 17 images. (32768 of 5339104 bytes remaining)
Sent 16 of 17 images. (16384 of 5339104 bytes remaining)
Sent 17 of 17 images. (0 of 5339104 bytes remaining)
Finished writing images; rebooting board.
Device ID: 8B41BCD7106C987F5AA0339F994AE4A309BF0A0CE3D50D6357746012E61D6EC6B4D27
22FE22C55C074466B35
Device recovered successfully.
```

Note: **If you encounter a timeout after the device reboots** then your serial ports are not correctly configured, or the TAP driver installed by the SDK installer is not working correctly. You'll need to resolve this issue before moving on. This is usually just a matter of updating the serial port drivers, see the appendix at the end of this document for details. If updating the serial port drivers does not resolve your issue, then see the troubleshooting guide by following the link below.

Microsoft has been documenting potential issues and how to work around them. This page is continually being updated.

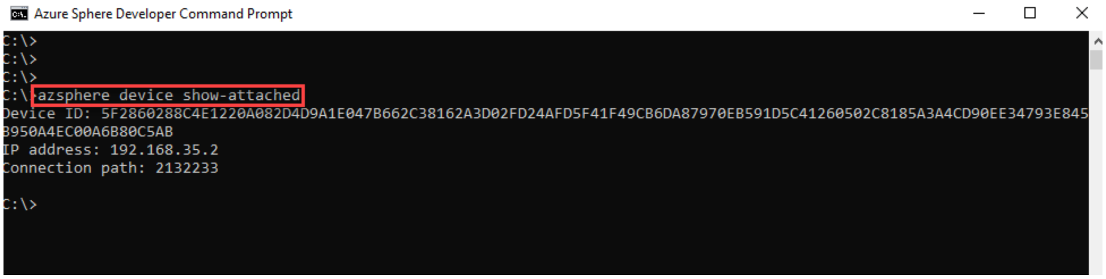
- Troubleshooting installation and setup on Windows

Verify Device to PC Connectivity

To verify that the tools are installed and working properly we need to exercise the communication path between the PC and the Azure Sphere device. This is easy to accomplish by just reading the device ID from the device.

- Using the Azure Sphere Command Prompt window execute the command
    - **azsphere device show-attached**

You should see output similar to the screenshot below. If your command times out, then see the appendix and troubleshooting web page here.



```
C:\>
C:\>
C:\>
C:\>azsphere device show-attached
Device ID: 5F2860288C4E1220A082D4D9A1E047B662C38162A3D02FD24AFD5F41F49CB6DA87970EB591D5C41260502C8185A3A4CD90EE34793E845
B950A4EC00A6B80C5AB
IP address: 192.168.35.2
Connection path: 2132233

C:\>
```

Wrap Up

If you're able to read the device ID from your device, then you're ready to move on to the next section.

Preparing your device for development

Objectives

The objectives of this section are to perform all the tasks required to begin Azure Sphere Development

- Determine if you have access to an Azure Sphere Tenant. If not, then setup a new Tenant
- Claim the Azure Sphere device to your Azure Sphere Tenant
- Configure the device Wi-Fi settings to connect to your local Wi-Fi access point or hotspot
- Unlock our Starter Kit for development

Requirements

Software

- Visual Studio 2019 version 16.4 or later (Enterprise, Professional, or Community version) **installed**
- Azure Sphere SDK 20.11 or the current SDK release **installed**

Azure Sphere Tenant

To manage your Azure Sphere devices, you need access to an Azure Sphere "Tenant", this is the infrastructure that Microsoft has provided to isolate and manage Azure Sphere devices (only used for Azure Sphere). It's important to get setup with a tenant that can be used for the long term. If a tenant is available through your work, it is recommended to use that. If you need to set up your own tenant, that's fine too, but plan on keeping it for long term. You will claim your Starter Kit device to this tenant, which is a one-time process, as once claimed, **it can't be moved to another tenant** (i.e. Be sure this the tenant you want to use for all your sphere development). Microsoft has good Azure Sphere Tenant documentation here.

Note that the Azure Sphere Tenant is **not** associated with an Azure Subscription, it's a stand-alone entity that Microsoft provides for the life of your Azure Sphere devices.

Login and/or Create an Azure Sphere Tenant

If you don't have access to an Azure Sphere Tenant or if have an Azure Sphere login but have just updated your SDK to 19.10 or later for the first time, then you need to use the **--newuser** parameter to create a new login.

You must use an email account associated with a Microsoft account, for example an email address associated with an Office365 account. You can check/verify that your email address is associated with a Microsoft account here. If not, either associate your email as a Microsoft account, or setup a new one.

- **azsphere login --newuser <email-address>**

If you already have access to an Azure Sphere Tenant, then sign in with your Microsoft associated email account credentials:

- **azsphere login**

After you're logged in you'll fall into one of three situations:

1. You are already assigned to one and only one tenant. That tenant will become the "current" tenant
2. You have access to more than one tenant, so you need to select one:
    - **azsphere tenant list**
    - **azsphere tenant select -i <tenant GUID>**
3. You don't have access to any tenants and none exist for you to use, so you need to create one:

    - **azsphere tenant create --name <tenant name> -f**

You can always show the currently selected tenant using the **azsphere tenant show-selected** command

4. You can also add users to your tenant
    - **azsphere register-user -u <email_address>**
    - **azsphere role add -r Contributor|Administrator|Reader -u <email_address>**

Claim your Azure Sphere Device

Now you will claim your Starter Kit device to your Azure Sphere Tenant. This process adds an entry in an Azure Sphere Security Service database, to associate your device (using the device ID) with your tenant.

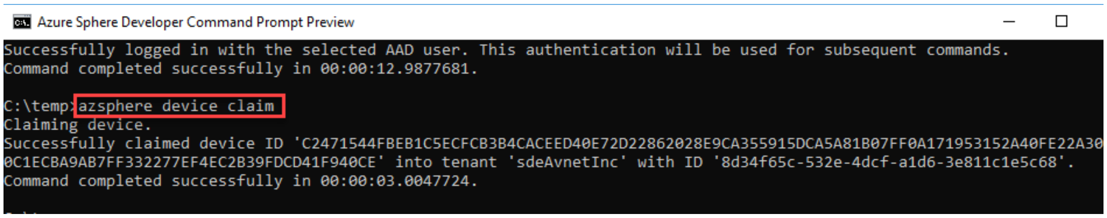
This allows you to manage your devices and can logically tie your device to other Azure services such as a Device Provisioning Service (DPS).

Having the device associated to your tenant is also a security feature. For example, if someone was able to get hold of one of your applications and they tried to load it onto a device that was NOT in your tenant, then that device would not be authorized to use the DPS associated with your tenant, and thus would not be able to connect to your Azure IoT Hub.

Claiming a device is a simple process.

1. Make sure your device is connected to your PC, then execute the command

    a. **azsphere device claim**



Azure Sphere Developer Command Prompt Preview

```
Successfully logged in with the selected AAD user. This authentication will be used for subsequent commands.
Command completed successfully in 00:00:12.9877681.

C:\temp>azsphere device claim
Claiming device.
Successfully claimed device ID 'C2471544FBEB1C5ECFCB3B4CACEED40E72D22862028E9CA355915DCA5A81B07FF0A171953152A40FE22A30
0C1ECBA9AB7FF332277EF4EC2B39FDCD41F940CE' into tenant 'sdeAvnetInc' with ID '8d34f65c-532e-4dcf-a1d6-3e811c1e5c68'.
Command completed successfully in 00:00:03.0047724.
```

    b. When you execute the claim command the tool will read the device ID from your device, send the device to the Azure Sphere Tenant where the ID will be stored in a database.

    c. Note that this command can also be executed without the device connected using the **-i <device Id>** arguments. So if you needed to claim 100,000 devices and you had all the device IDs in a text file you could script the process.

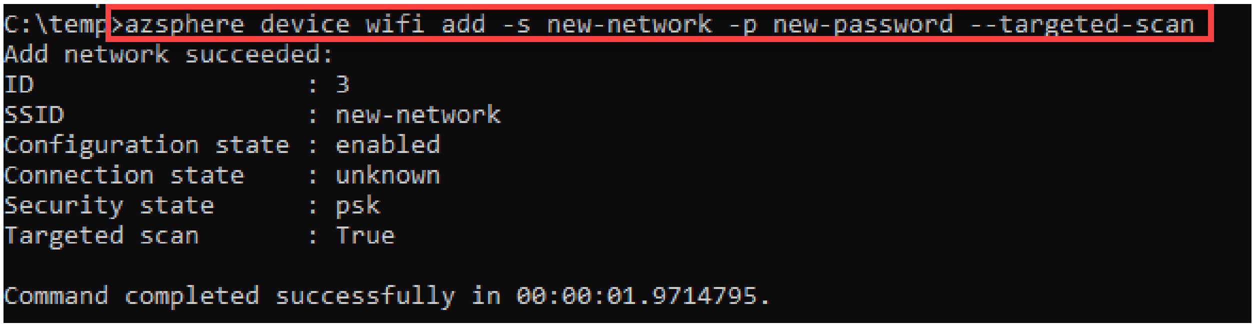
Configure the Wi-Fi

Next we need to configure the Wi-Fi on our device to connect to a Wi-Fi access point or hotspot.

1. Make sure your device is still connected to your PC

2. Execute the command **azsphere device wifi add --ssid <ssid> --psk <password> --targeted-scan**
   where <ssid> and <password> are your SSID and your SSID password

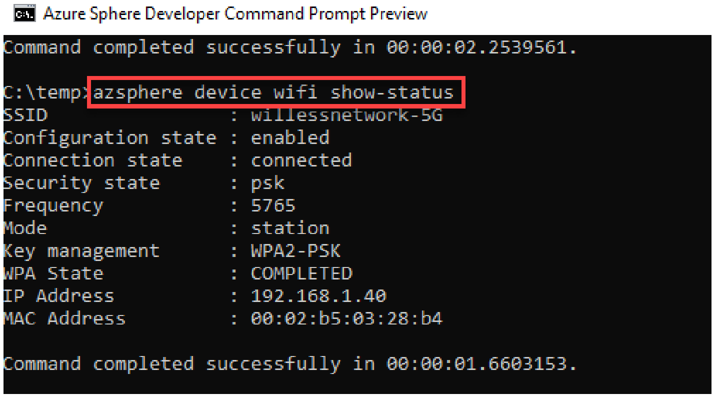
   a. The **--targeted-scan** option was added in the 19.09 release and is major improvement to Azure Sphere Wi-Fi.



```
C:\temp>azsphere device wifi add -s new-network -p new-password --targeted-scan
Add network succeeded:
ID                    : 3
SSID                  : new-network
Configuration state : enabled
Connection state      : unknown
Security state        : psk
Targeted scan         : True

Command completed successfully in 00:00:01.9714795.
```

3. You can check the status of your Wi-Fi connection by executing **azsphere device wifi show-status**

4. You can check the status of all Wi-Fi networks configured on the device by executing **azsphere device wifi list**



```
Command completed successfully in 00:00:02.2539561.

C:\temp>azsphere device wifi show-status
SSID                  : willessnetwork-5G
Configuration state : enabled
Connection state      : connected
Security state        : psk
Frequency             : 5765
Mode                  : station
Key management        : WPA2-PSK
WPA State             : COMPLETED
IP Address            : 192.168.1.40
MAC Address           : 00:02:b5:03:28:b4

Command completed successfully in 00:00:01.6603153.
```

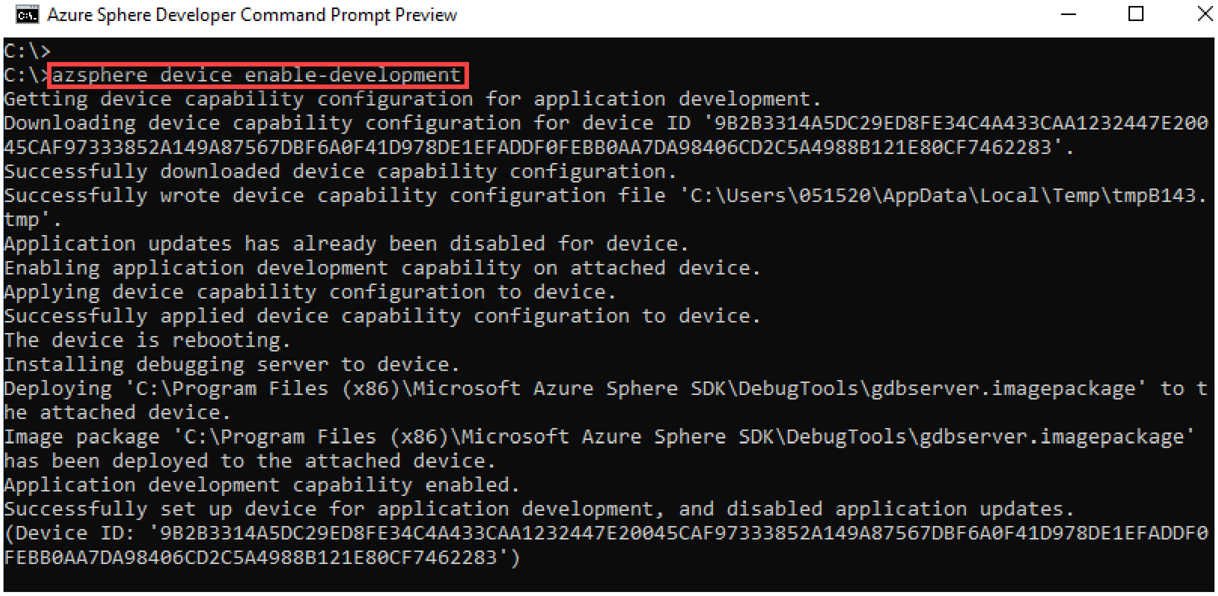
Unlock the Starter Kit for Development

By default Azure Sphere is a locked down device. You can’t load any applications onto the device even with a physical connection, if you could, bad actors who had physical access to the device could load whatever application they wanted to onto the device. So when Azure Sphere devices are deployed, they are put into enable-cloud-test mode that adds the device into a device group associated with a product. Once added to a product and device group the Azure Sphere Security Service (AS3) in the cloud, knows what OS SKU and user application to send to the device over-the-air. Once in enable-cloud-test mode the ONLY way to change software on the device is OTA via the AS3. You can read more about OTA software deployments here.

Devices ship in a locked state, so we need to “unlock” it. This accomplished using an **azsphere** command. We should still be logged into to our tenant. Since we want to change the accessibility of our device, the AS3 will first authenticate us to the tenant, then when we execute the command to change the device from enable-cloud-test to enable-development mode, AS3 will confirm that our device belongs (or has been claimed) to our tenant. If not, then we won’t be able to change the accessibility of the device. So if someone was to steal a deployed device, they could not do anything with the device outside its main purpose in life.

1. Confirm that your device is connected to your PC using the supplied USB cable

2. Next, put the device into enable-development mode by entering the command
   azsphere device enable-development
   You should see output similar to the graphic below



```
C:\>
C:\>azsphere device enable-development
Getting device capability configuration for application development.
Downloading device capability configuration for device ID '9B2B3314A5DC29ED8FE34C4A433CAA1232447E200
45CAF97333852A149A87567DBF6A0F41D978DE1EFADDF0FEBB0AA7DA98406CD2C5A4988B121E80CF7462283'.
Successfully downloaded device capability configuration.
Successfully wrote device capability configuration file 'C:\Users\051520\AppData\Local\Temp\tmpB143.
tmp'.
Application updates has already been disabled for device.
Enabling application development capability on attached device.
Applying device capability configuration to device.
Successfully applied device capability configuration to device.
The device is rebooting.
Installing debugging server to device.
Deploying 'C:\Program Files (x86)\Microsoft Azure Sphere SDK\DebugTools\gdbserver.imagepackage' to t
he attached device.
Image package 'C:\Program Files (x86)\Microsoft Azure Sphere SDK\DebugTools\gdbserver.imagepackage'
has been deployed to the attached device.
Application development capability enabled.
Successfully set up device for application development, and disabled application updates.
(Device ID: '9B2B3314A5DC29ED8FE34C4A433CAA1232447E20045CAF97333852A149A87567DBF6A0F41D978DE1EFADDF0
FEBB0AA7DA98406CD2C5A4988B121E80CF7462283')
```

Pulling the application source code and configuring your application

Objectives

- Pull an Azure Sphere project down from GitHub
- Review the different build options in the project
- Build and run the project

Plug and Play Application

We’ve created an example application that reads the Starter Kit on-board sensors and interacts with Azure. The application implements the following features.

Supports multiple Azure Connectivity options

- No Azure Connection (reads on-board sensors and outputs data to debug terminal)
- IoT Hub Connection using the Azure Device Provisioning Service
- IoT Hub Connection using the direct connection method
- IoT Hub Connection with Azure Plug and Play (PnP) functionality
- Avnet IoTConnect Platform Connection

Sensors

- Reads the Starter Kit on-board sensors
- LSM6DSO: 3D Accelerometer and 3D Gyro sensor
- LPS22HH: Barometric pressure sensor
- ALS-PT19: Ambient light sensor

Button Features

- The Avnet Starter Kit includes two user buttons, ButtonA and ButtonB
- When using the optional OLED display
  - Button A: Move to the last OLED screen
  - Button B: Move to the next OLED screen

Connected Features

- Sends sensor readings up as telemetry
- Implements Device Twins
  - Control all user LEDs
  - Control an optional Relay Click board in Click Socket #1
  - Configure custom message to display on an optional OLED display
- Implements two direct methods
  - setSensorPollTime: Modifies the period (in seconds) between reading the on-board sensors and sending telemetry messages
  - rebootDevice: Forces the device to execute a reboot
- Sends button press events as telemetry

Optional Hardware

The application supports the following optional hardware to enhance the example

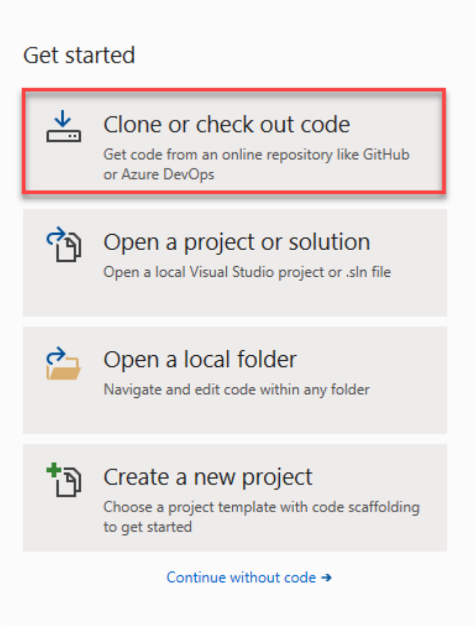
- 0.96" I2C OLED LEC Display
    - Verify that the pinout for your display matches the pinout on the Starter Kit
- Relay Click Board

Pull Azure Sphere project down from GitHub

Let's get started by pulling the source from GitHub.

- Open Visual Studio 2019

- When the application opens you should see "**Get started**" options. Select "**Clone or checkout code**"



Get started

Clone or check out code
Get code from an online repository like GitHub or Azure DevOps

Open a project or solution
Open a local Visual Studio project or .sln file

Open a local folder
Navigate and edit code within any folder

Create a new project
Choose a project template with code scaffolding to get started

Continue without code →

- For the **Repository location** enter https://github.com/Avnet/azure-sphere-samples.git

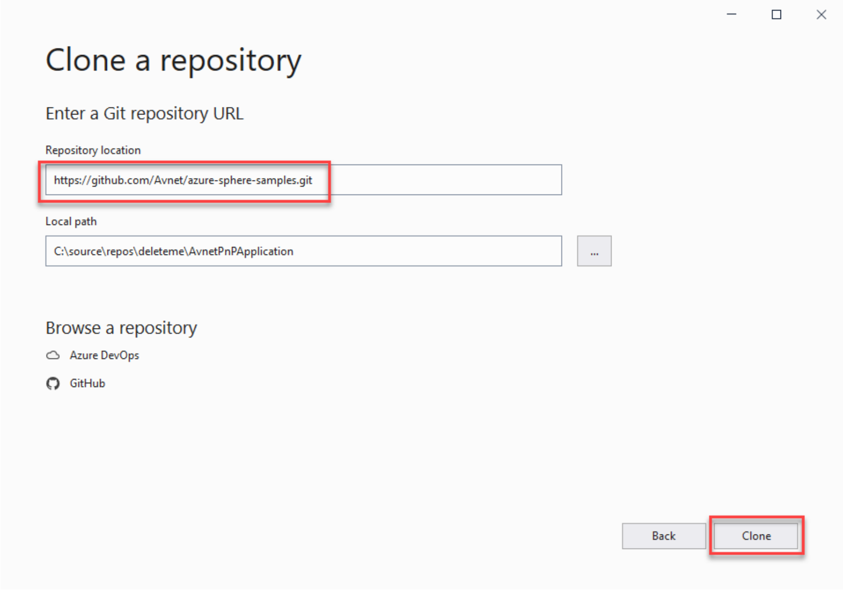
- Confirm the **Local path**, or change this if you wish

- Click the **Clone** button

- The project will download and the Solution Explorer will display the contents of the project.

- Wait for the entire repository to clone, then **close Visual Studio**

  a. The repository contains multiple projects. We'll close Visual Studio and open a new instance using just the Plug and Play project.



Clone a repository

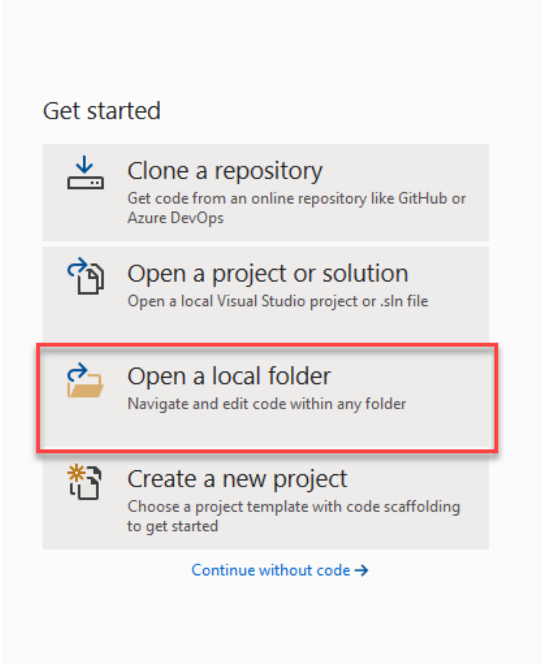
Enter a Git repository URL

Repository location: https://github.com/Avnet/azure-sphere-samples.git

Local path: C:\source\repos\deleteme\AvnetPnPApplication

Open the CMAKE Project

- Open a fresh instance of Visual Studio
- This time select the “Open a local folder” option



Get started

Clone a repository
Get code from an online repository like GitHub or Azure DevOps

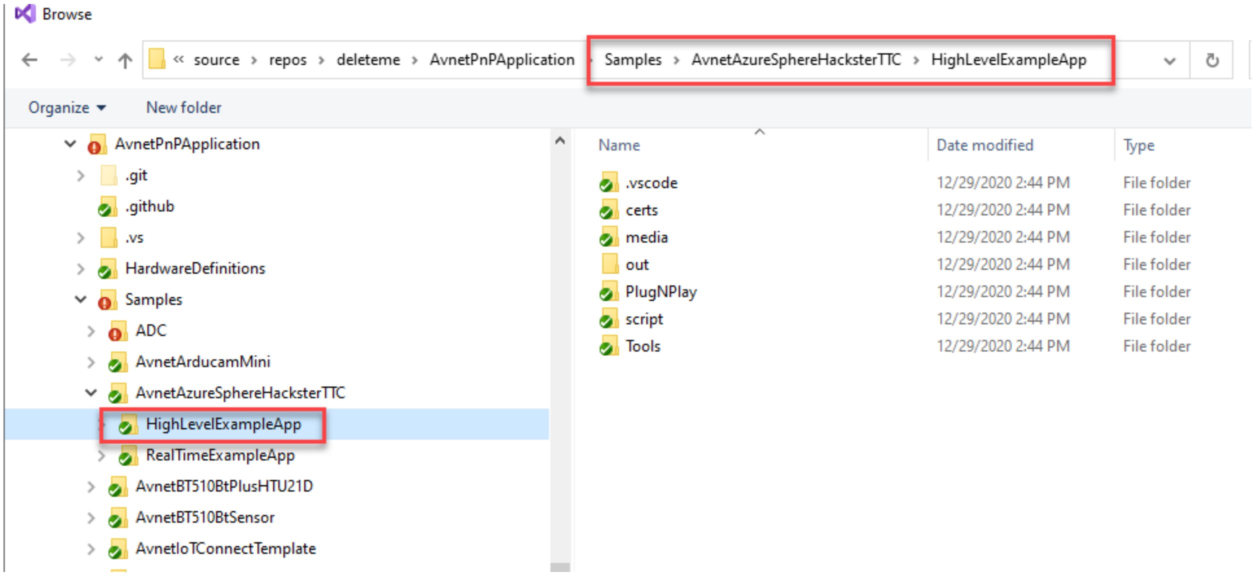
Open a project or solution
Open a local Visual Studio project or .sln file

Open a local folder
Navigate and edit code within any folder

Create a new project
Choose a project template with code scaffolding to get started

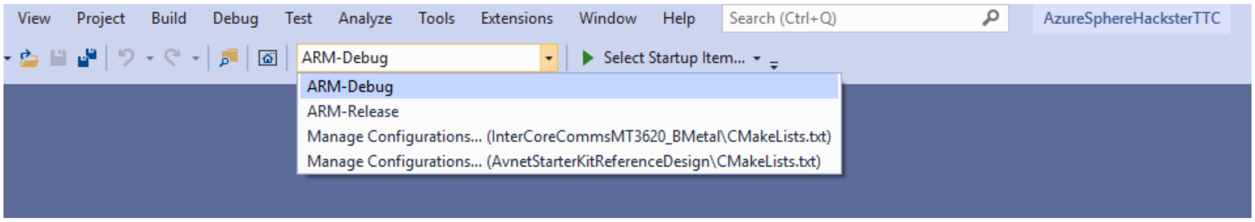
Continue without code →

- A file explorer dialog opens
- Navigate to the folder where the repo was cloned
- Select the <your repo>/Samples/AvnetAzureSphereHacksterTTC/HighLevelApp folder
- Click on the “Select Folder” button

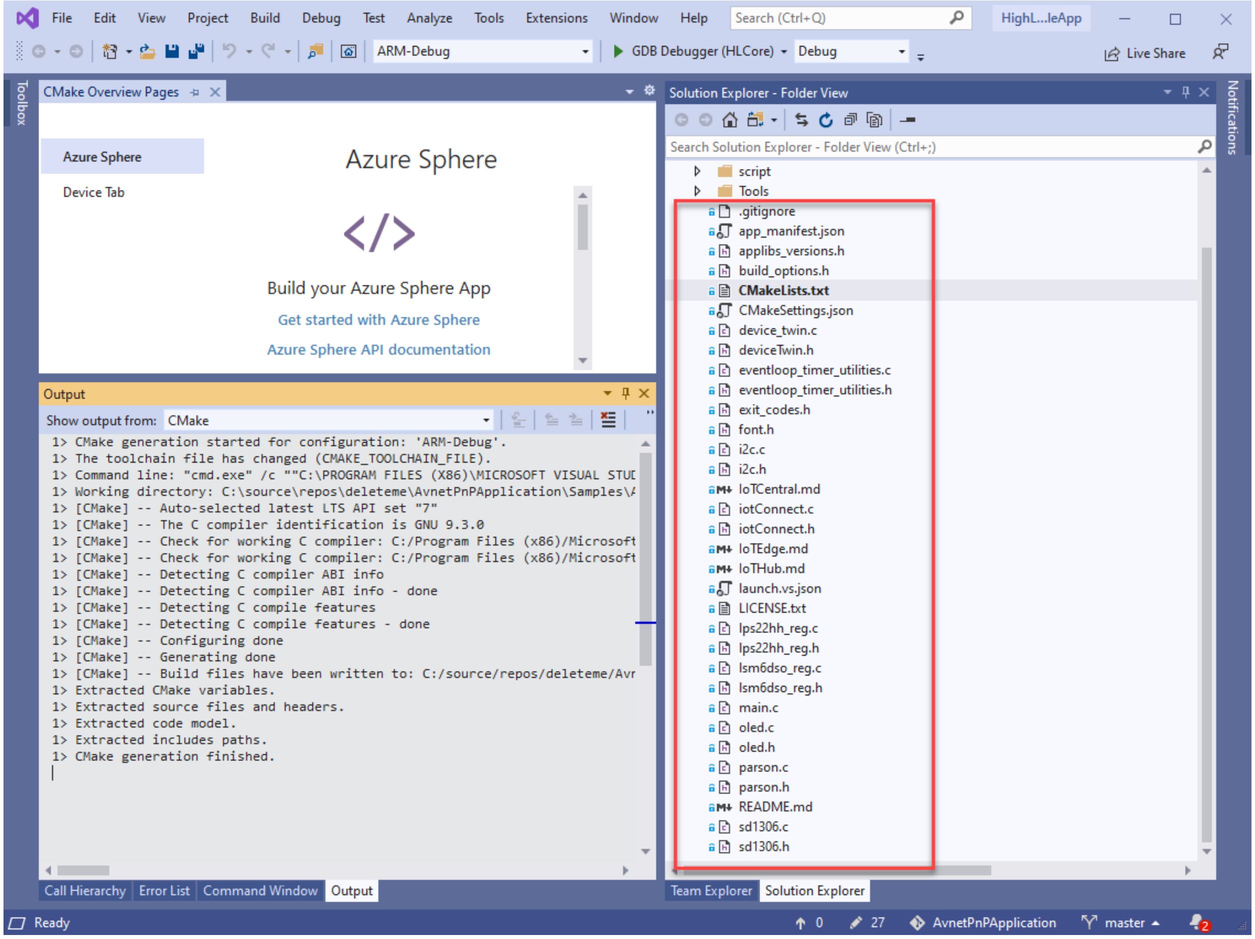


Name	Date modified	Type
.vscode	12/29/2020 2:44 PM	File folder
certs	12/29/2020 2:44 PM	File folder
media	12/29/2020 2:44 PM	File folder
out	12/29/2020 2:44 PM	File folder
PlugNPlay	12/29/2020 2:44 PM	File folder
script	12/29/2020 2:44 PM	File folder
Tools	12/29/2020 2:44 PM	File folder

- The CMAKE project should open automatically. You'll know this is the case if you see the two different build targets listed in the target selection control (ARM-Debug and ARM-Release).



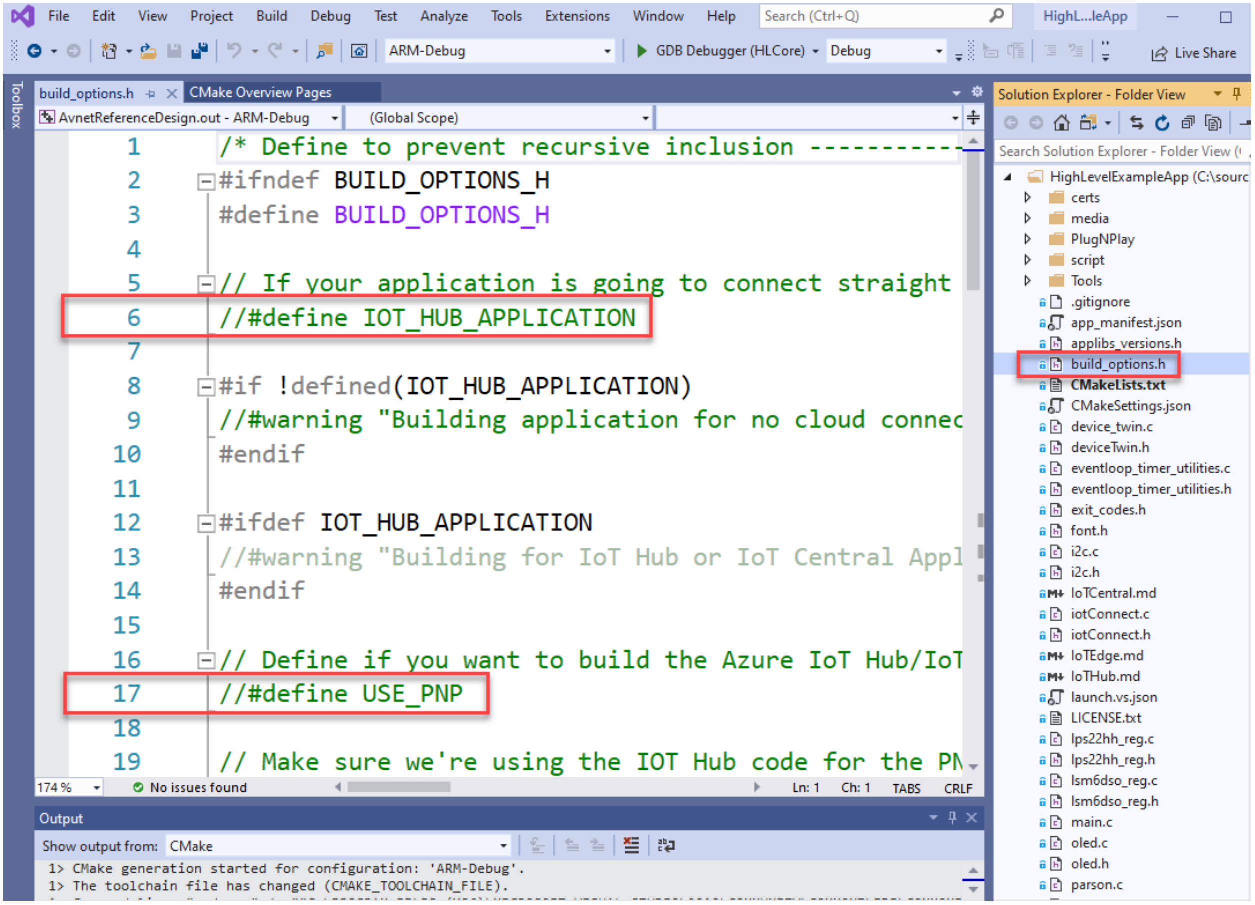
- The open project will list all the source files in the Solution Explorer window



```
1> CMake generation started for configuration: 'ARM-Debug'.
1> The toolchain file has changed (CMAKE_TOOLCHAIN_FILE).
1> Command line: "cmd.exe" /c ""C:\PROGRAM FILES (X86)\MICROSOFT VISUAL STUI
1> Working directory: C:\source\repos\deleteme\AvnetPnPApplication\Samples\A
1> [CMake] -- Auto-selected latest LTS API set "7"
1> [CMake] -- The C compiler identification is GNU 9.3.0
1> [CMake] -- Check for working C compiler: C:/Program Files (x86)/Microsoft
1> [CMake] -- Check for working C compiler: C:/Program Files (x86)/Microsoft
1> [CMake] -- Detecting C compiler ABI info
1> [CMake] -- Detecting C compiler ABI info - done
1> [CMake] -- Detecting C compile features
1> [CMake] -- Detecting C compile features - done
1> [CMake] -- Configuring done
1> [CMake] -- Generating done
1> [CMake] -- Build files have been written to: C:/source/repos/deleteme/Avr
1> Extracted CMake variables.
1> Extracted source files and headers.
1> Extracted code model.
1> Extracted includes paths.
1> CMake generation finished.
```

Review the different build options in the project

The project has multiple different build options that are configured using #define pre-processor directives. The build is configured in the **build_options.h** header file. Open **build_options.h** to see the different options. Let's review the three most common build configurations.



```
/* Define to prevent recursive inclusion ----------
#ifndef BUILD_OPTIONS_H
#define BUILD_OPTIONS_H

// If your application is going to connect straight
//#define IOT_HUB_APPLICATION

#if !defined(IOT_HUB_APPLICATION)
//#warning "Building application for no cloud connec
#endif

#ifdef IOT_HUB_APPLICATION
//#warning "Building for IoT Hub or IoT Central Appl
#endif

// Define if you want to build the Azure IoT Hub/IoT
//#define USE_PNP

// Make sure we're using the IOT Hub code for the PN
```

```
1> CMake generation started for configuration: 'ARM-Debug'.
1> The toolchain file has changed (CMAKE_TOOLCHAIN_FILE).
```

- **No cloud connectivity**: This is the simplest configuration. The application runs and you can monitor sensor data from the Visual Studio debug window.
  - IOT_HUB_APPLICATION (not defined)
  - USE_PNP (not defined)

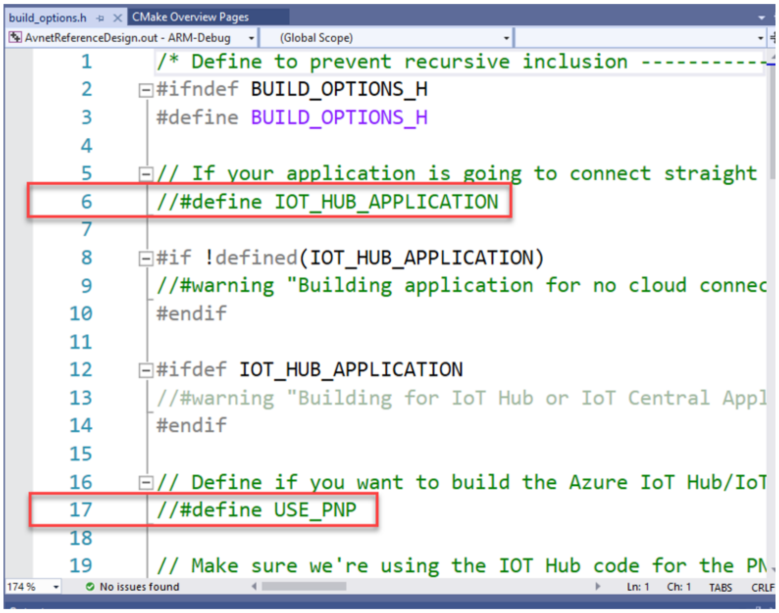
- **IoT Hub connectivity**: This configuration requires an Azure IoT Hub and a Device Provisioning Service (DPS) in Azure or an IoT Central application that will provide an IoT Hub and DPS.
  - IOT_HUB_APPLICATION (**defined**)
  - USE_PNP (not defined)

- **Azure IoT Plug and Play**: This configuration implements all the bits required for the application to register with the Azure IoT Hub as a Plug and Play device and send the correct device twin reported JSON objects.

  - IOT_HUB_APPLICATION (not defined)
  - USE_PNP (**defined**)

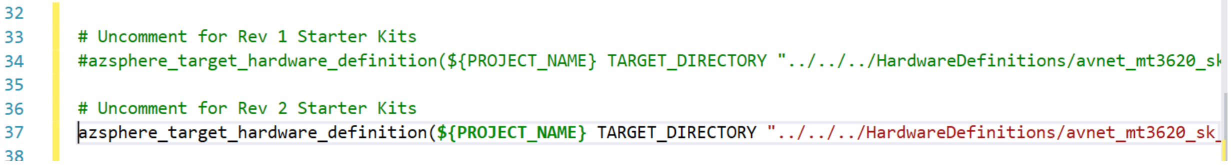
Just to make sure everything is working correctly, we’ll build and run the non-connected configuration.

- Open the build_options.h file and confirm that both the #defines for IOT_HUB_APPLICATION and USE_PNP are both **commented out**. Look around lines 6-17 for the #define pre-processor directives. This will build the no cloud connectivity configuration.



```
1   /* Define to prevent recursive inclusion -----------
2   #ifndef BUILD_OPTIONS_H
3   #define BUILD_OPTIONS_H
4
5   // If your application is going to connect straight
6   //#define IOT_HUB_APPLICATION
7
8   #if !defined(IOT_HUB_APPLICATION)
9   //#warning "Building application for no cloud connec
10  #endif
11
12  #ifdef IOT_HUB_APPLICATION
13  //#warning "Building for IoT Hub or IoT Central Appl
14  #endif
15
16  // Define if you want to build the Azure IoT Hub/IoT
17  //#define USE_PNP
18
19  // Make sure we're using the IOT Hub code for the PN
```

- Configure the application for our Starter Kit (rev1 or rev2)
    - Look at your device, under the Azure Sphere Module printed on the PCB you’ll see a revision either “REV 1” or “REV 2”
    - Open the CMakeLists.txt file
    - Look at lines ~33 – 37
    - Depending on which revision you have, uncomment the line for your kit
    - I’m using a Rev 2 kit, my file is shown below



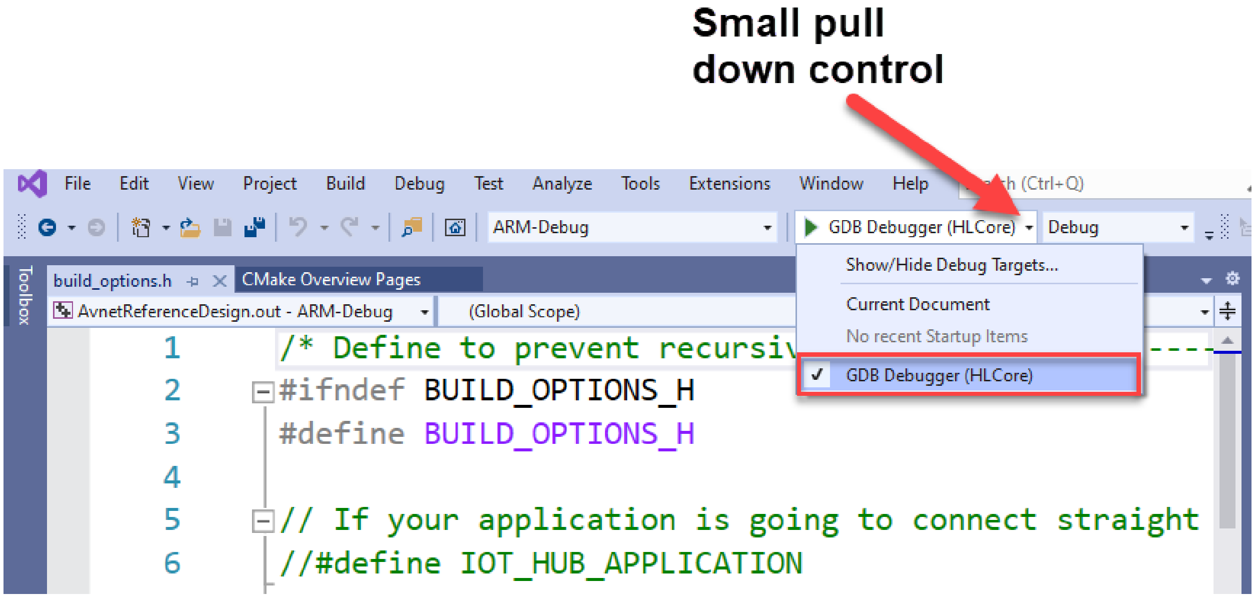
```
32
33  # Uncomment for Rev 1 Starter Kits
34  #azsphere_target_hardware_definition(${PROJECT_NAME} TARGET_DIRECTORY "../../../HardwareDefinitions/avnet_mt3620_sk
35
36  # Uncomment for Rev 2 Starter Kits
37  azsphere_target_hardware_definition(${PROJECT_NAME} TARGET_DIRECTORY "../../../HardwareDefinitions/avnet_mt3620_sk_
38
```

- Make sure that only one line is enabled
- Save the file, CMake regenerates the build scripts

Build and run the project

Now we can run the application.

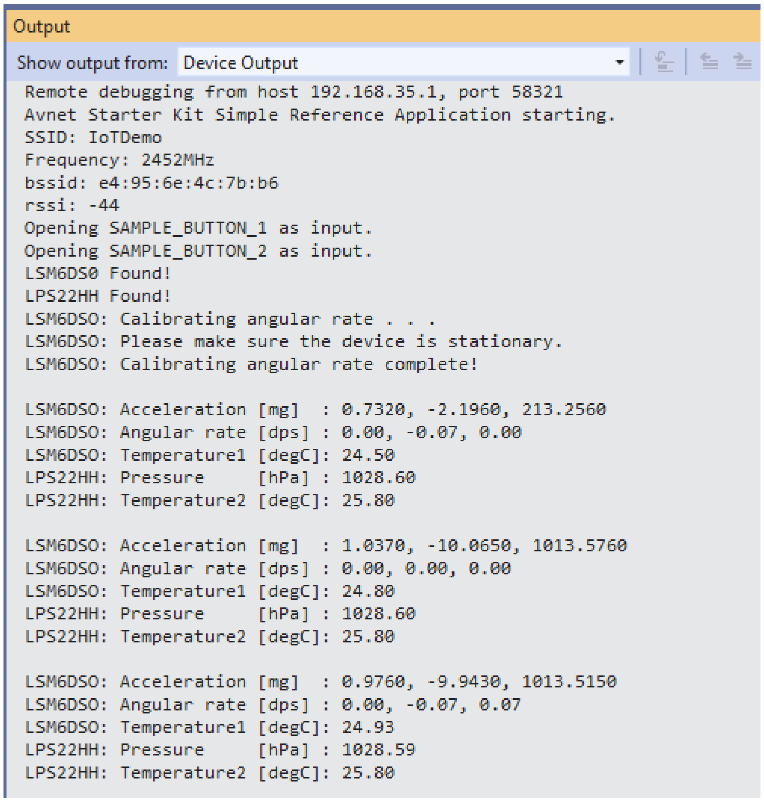
- Confirm that your Avnet Starter Kit is connected to your PC
- Select "GDB Debugger (HL Core)"
- Click on the "GDB Debugger (HL Core)" button at the top of the VS application. If prompted to save your files click "OK."



Small pull down control

```
/* Define to prevent recursiv
#ifndef BUILD_OPTIONS_H
#define BUILD_OPTIONS_H

// If your application is going to connect straight
//#define IOT_HUB_APPLICATION
```

- VS will build, link, side load, and run your application on the Sphere Starter Kit, all with just one click!
- You should see your Output window open and debug will start to scroll in the window. If not open the Output window with Ctrl + Alt + O
- Pick-up your Starter Kit up and slowly rotate in different planes to observe the Acceleration and Angular rates change as you move the device
- If you press and release the A and B user buttons on the Starter Kit the application will detect the button events and output applicable debug messages.
  Note! Be careful not to press the RESET button! (i.e., the button closest to the Sphere module)
- That's all there is to running an Azure Sphere application from Visual Studio
- If your sensor readings are "nan," then power cycle your starter kit and try again



```
Remote debugging from host 192.168.35.1, port 58321
Avnet Starter Kit Simple Reference Application starting.
SSID: IoTDemo
Frequency: 2452MHz
bssid: e4:95:6e:4c:7b:b6
rssi: -44
Opening SAMPLE_BUTTON_1 as input.
Opening SAMPLE_BUTTON_2 as input.
LSM6DS0 Found!
LPS22HH Found!
LSM6DSO: Calibrating angular rate . . .
LSM6DSO: Please make sure the device is stationary.
LSM6DSO: Calibrating angular rate complete!

LSM6DSO: Acceleration [mg]  : 0.7320, -2.1960, 213.2560
LSM6DSO: Angular rate [dps] : 0.00, -0.07, 0.00
LSM6DSO: Temperature1 [degC]: 24.50
LPS22HH: Pressure     [hPa] : 1028.60
LPS22HH: Temperature2 [degC]: 25.80

LSM6DSO: Acceleration [mg]  : 1.0370, -10.0650, 1013.5760
LSM6DSO: Angular rate [dps] : 0.00, 0.00, 0.00
LSM6DSO: Temperature1 [degC]: 24.80
LPS22HH: Pressure     [hPa] : 1028.60
LPS22HH: Temperature2 [degC]: 25.80

LSM6DSO: Acceleration [mg]  : 0.9760, -9.9430, 1013.5150
LSM6DSO: Angular rate [dps] : 0.00, -0.07, 0.07
LSM6DSO: Temperature1 [degC]: 24.93
LPS22HH: Pressure     [hPa] : 1028.59
LPS22HH: Temperature2 [degC]: 25.80
```

Configure our Application to connect to a cloud service

The benefit of Plug and Play certified applications is that cloud solutions that support PnP will automatically provision and associate PnP devices when they first connect to the solution. Before we can configure our application to connect to a solution, we need a solution to connect to. This next section will walk the reader through standing up a new IoT Central Application.

Microsoft provides test IoTCentral applications for free. There is a device limit, but since we’re just connecting a single device, this will be free.

IoT Central Application

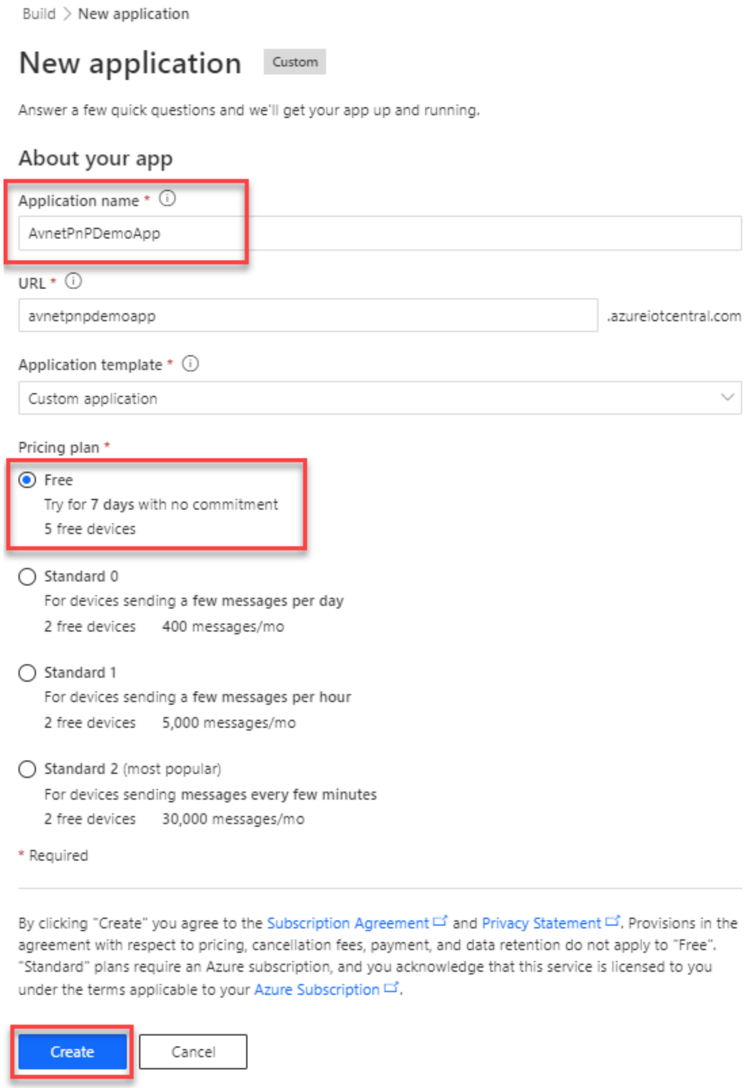
IoT Central is a Software-as-a-Service (SaaS) offering from Microsoft. Microsoft has made it easy to connect IoT devices and display telemetry data from connected devices. Additionally, there are some cloud to device (C2D) controls available. IoT Central Applications are created on-line without any coding necessary; all IoT Central configuration is driven from the web interface.

The steps we need to complete are listed below

- Create a new IoT Central Application or open an existing IoT Central Application if you have one
- Configure DPS for our application
- Configure our Azure Sphere application to connect to the IoT Central Application

Create IoT Central Application

- Open the Azure IoT Central Web Page: Link
- Click on the “Build a solution” blue button
- Scroll down to the bottom and click on the “Create a custom app” blue button
- The New application dialog opens



Build > New application

New application Custom

Answer a few quick questions and we'll get your app up and running.

About your app

Application name *

AvnetPnPDemoApp

URL *

avnetpnpdemoapp .azureiotcentral.com

Application template *

Custom application

Pricing plan *

● Free
Try for 7 days with no commitment
5 free devices

○ Standard 0
For devices sending a few messages per day
2 free devices 400 messages/mo

○ Standard 1
For devices sending a few messages per hour
2 free devices 5,000 messages/mo

○ Standard 2 (most popular)
For devices sending messages every few minutes
2 free devices 30,000 messages/mo

* Required

By clicking "Create" you agree to the Subscription Agreement and Privacy Statement. Provisions in the agreement with respect to pricing, cancellation fees, payment, and data retention do not apply to "Free". "Standard" plans require an Azure subscription, and you acknowledge that this service is licensed to you under the terms applicable to your Azure Subscription.

Application Name and URL

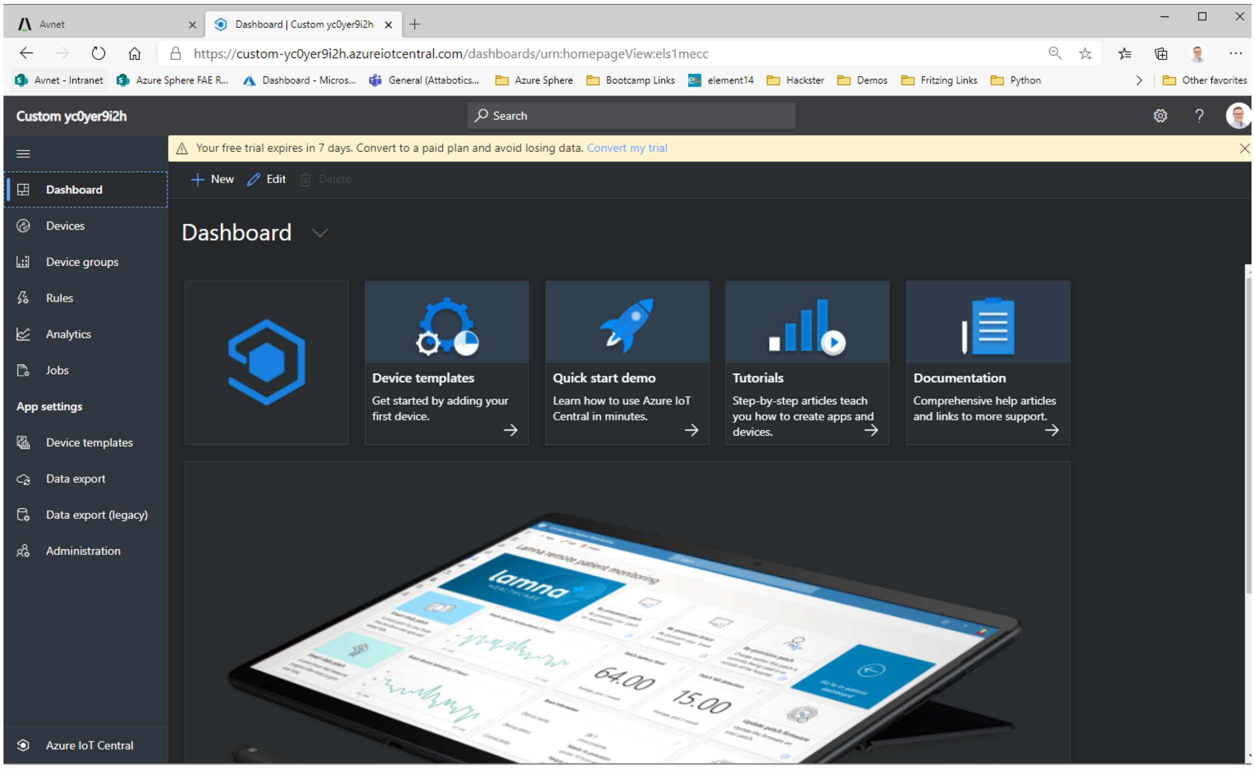
- Feel free to modify these as you wish. Note that the URL will need to be unique across of Azure since this will generate a FQDN.

7 day free trial (Recommended)

- Select this option to use the IoT Central application free for 7 days. At the end of the 7 day trial you’ll be prompted to update the application to a pay-as-you-go or some other paid Azure subscription.

Click on the Create button

The IoT Central application will be provisioned and will open in your browser. Bookmark this page or copy the URL in case you close the window.



https://custom-yc0yer9i2h.azureiotcentral.com/dashboards/urn:homepageView:els1mecc

Your free trial expires in 7 days. Convert to a paid plan and avoid losing data. Convert my trial

Device templates: Get started by adding your first device.

Quick start demo: Learn how to use Azure IoT Central in minutes.

Tutorials: Step-by-step articles teach you how to create apps and devices.

Documentation: Comprehensive help articles and links to more support.

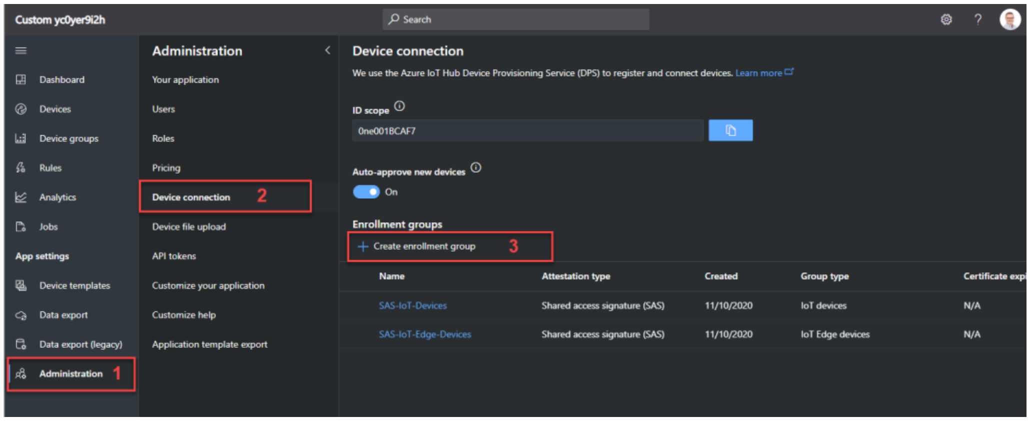
Configure DPS for our Application

The next thing we need to do is to configure the IoT Central application so that devices from our Azure Sphere Tenant are allowed to connect to our application. IoT Central uses a Device Provisioning Service (DPS) to provision devices.

Upload the public tenant CA certificate to Azure IoT Central and generate a verification code

In your new IoT Central application

- Select the Administration (1)
- Select Device connection (2)
- Select the "+ Create enrollment group" (3)



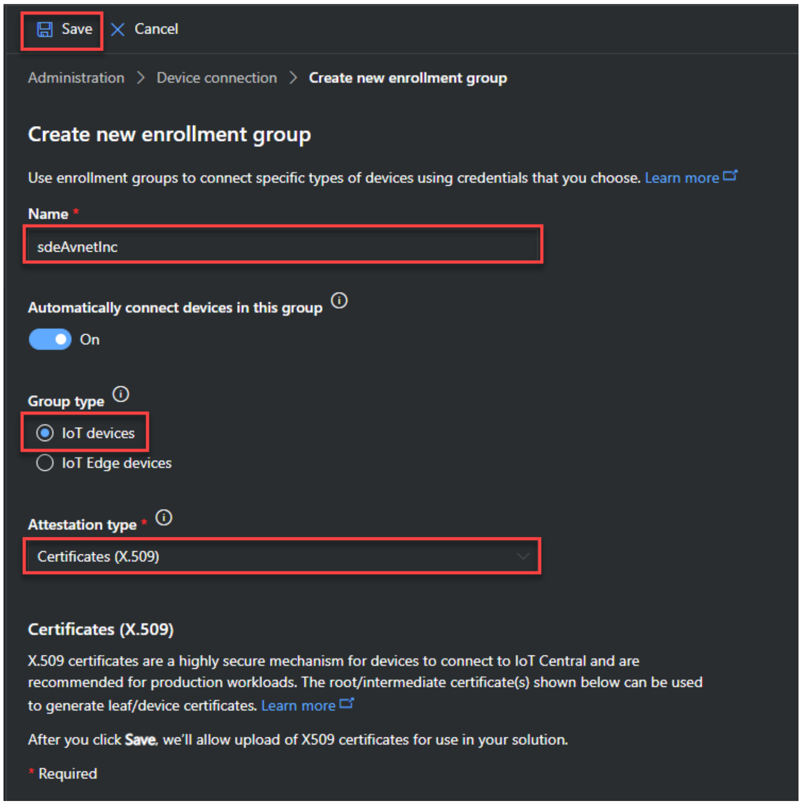
We use the Azure IoT Hub Device Provisioning Service (DPS) to register and connect devices. Learn more

ID scope: 0ne001BCAF7

Auto-approve new devices: On

Enrollment groups

Name	Attestation type	Created	Group type	Certificate exp
SAS-IoT-Devices	Shared access signature (SAS)	11/10/2020	IoT devices	N/A
SAS-IoT-Edge-Devices	Shared access signature (SAS)	11/10/2020	IoT Edge devices	N/A



Save
Cancel
Administration > Device connection > Create new enrollment group

Create new enrollment group

Use enrollment groups to connect specific types of devices using credentials that you choose. Learn more

Name *

sdeAvnetInc

Automatically connect devices in this group

On

Group type

● IoT devices

○ IoT Edge devices

Attestation type *

Certificates (X.509)

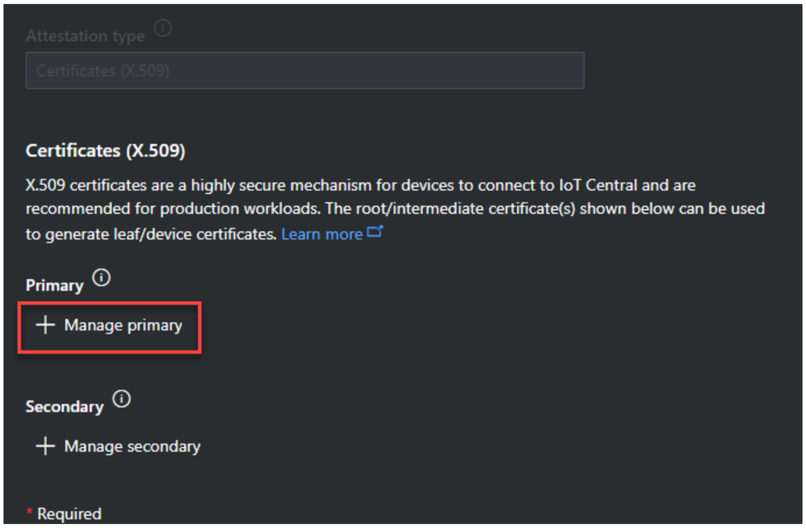
Certificates (X.509)

X.509 certificates are a highly secure mechanism for devices to connect to IoT Central and are recommended for production workloads. The root/intermediate certificate(s) shown below can be used to generate leaf/device certificates. Learn more

After you click **Save**, we'll allow upload of X509 certificates for use in your solution.

* Required

- Give your enrollment group a name. I recommend naming the group after your Azure Sphere Tenant, since we're setting up the group for devices in our Tenent
- Make sure that the Group type is set to "IoT devices"
- Select "Certificates (X.509) for the Attestation type
- Click the "Save" link at the top of the form



Attestation type

Certificates (X.509)

Certificates (X.509)

X.509 certificates are a highly secure mechanism for devices to connect to IoT Central and are recommended for production workloads. The root/intermediate certificate(s) shown below can be used to generate leaf/device certificates. Learn more

Primary

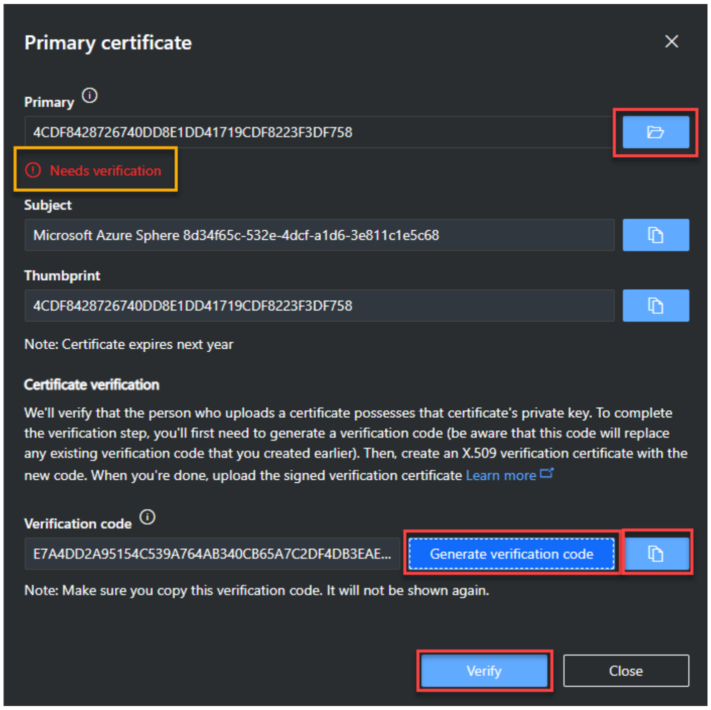
Secondary

* Required

- Click on the "+ Manage primary" link to add the primary X.509 certificate

Download the public tenant CA certificate

- Go back to your “Azure Sphere Developer Command Prompt” application
    - Start → Azure Sphere → Azure Sphere Developer Command Prompt
- Make sure you’re logged into your tenant:
    - `azsphere login`
- Copy and paste in the command:
    - `azsphere ca download --output CAcertificate.cer`
        - Note the output file must have the .cer extension



Primary ⓘ

4CDF8428726740DD8E1DD41719CDF8223F3DF758

Needs verification

Subject

Microsoft Azure Sphere 8d34f65c-532e-4dcf-a1d6-3e811c1e5c68

Thumbprint

4CDF8428726740DD8E1DD41719CDF8223F3DF758

Note: Certificate expires next year

Certificate verification

We'll verify that the person who uploads a certificate possesses that certificate's private key. To complete the verification step, you'll first need to generate a verification code (be aware that this code will replace any existing verification code that you created earlier). Then, create an X.509 verification certificate with the new code. When you're done, upload the signed verification certificate Learn more

Verification code ⓘ

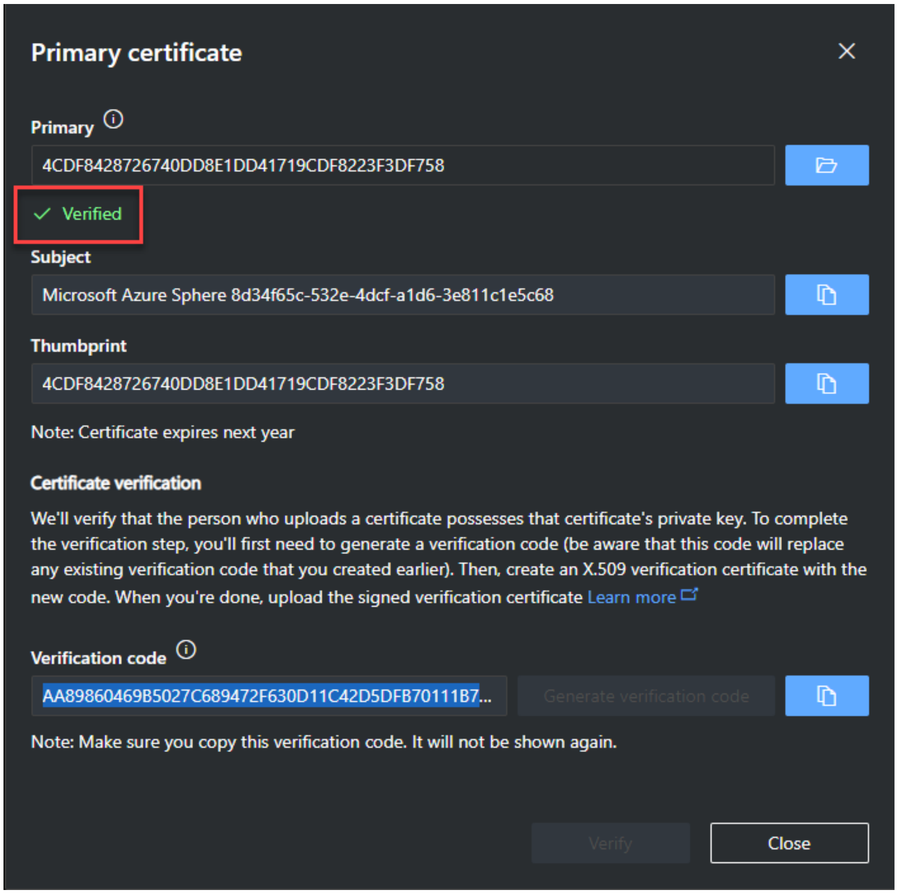
E7A4DD2A95154C539A764AB340CB65A7C2DF4DB3EAE...

Note: Make sure you copy this verification code. It will not be shown again.

- Use the folder link to browse to and select the CAcertificate downloaded in the previous step.
- Return to the Azure Sphere Developer Command Prompt application
- Copy and paste the following command into the application, **don’t execute the command yet**, we need to get the validation code first
- `azsphere ca download-proof --output ValidationCertification.cer --verificationcode <code>`
- Click on the “Generate verification code” button to generate a Verification code. Copy the verification code and paste it at the end of your azsphere command (replace the <code> text with your verification code)

The Azure Sphere Security Service generates the validation certificate and includes the verification code inside the certificate. Since AS3 generated/encrypted this certificate using the private key, and we provided the public key to DPS, DPS can decrypt the validation certificate and confirm/verify that the validation code is the same code that was generated in the interface.

- Back in your IoT Central application, click on the Verify button at the bottom of the form
- Upload the new Verification Certificate by browsing to the file you just downloaded from the tenant
- The Primary Certificate window will update and show the status as Verified



Primary certificate

Primary ⓘ

4CDF8428726740DD8E1DD41719CDF8223F3DF758

✓ Verified

Subject

Microsoft Azure Sphere 8d34f65c-532e-4dcf-a1d6-3e811c1e5c68

Thumbprint

4CDF8428726740DD8E1DD41719CDF8223F3DF758

Note: Certificate expires next year

Certificate verification

We'll verify that the person who uploads a certificate possesses that certificate's private key. To complete the verification step, you'll first need to generate a verification code (be aware that this code will replace any existing verification code that you created earlier). Then, create an X.509 verification certificate with the new code. When you're done, upload the signed verification certificate Learn more

Verification code ⓘ

AA89860469B5027C689472F630D11C42D5DFB70111B7...

Note: Make sure you copy this verification code. It will not be shown again.

- Click on the Save button (top of dialog)
- Click on the Close button (bottom of dialog)

Now that we have the IoT Central side ready, let’s go back to our application code and configure the application to connect to IoT Central.

Configure our application to connect to the IoT Central Application

We need to find key pieces of information from the IoT Central interface to include in our Azure Sphere application so that it will connect to our new IoT Central application.

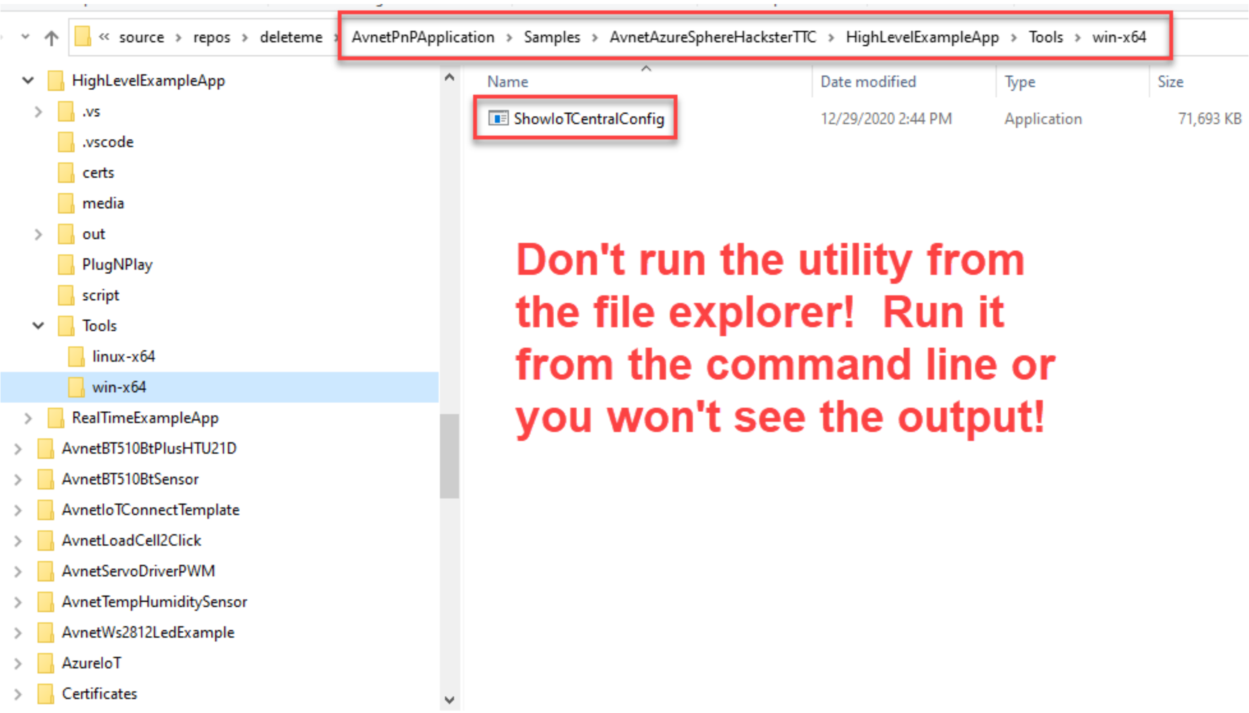
Once we have the required details integrated into our application, we can load our application onto as many Azure Sphere devices as we want to, and as long as they are all “claimed” to our Azure Sphere Tenant, the first time they come on-line they will connect and auto-provision into our IoT Central Application.

To configure our application we need to find the following pieces of information:

- The DPS “scope ID” or “ID scope”
- Our Azure Sphere tenant GUID/ID
- The FQDN for the global DPS endpoint
- The FQDN(s) for the IoT Hub(s) that exists behind the IoT Central SaaS implementation

Run the ShowIoTCentralConfig Utility

Microsoft provides a utility to help us find most of the required information. The utility is called ShowIoTCentralConfig.exe. This utility is included in the GitHub project in the HighLevelExampleApp/Tools/win-x64 folder.



Name	Date modified	Type	Size
ShowIoTCentralConfig	12/29/2020 2:44 PM	Application	71,693 KB

Don't run the utility from the file explorer! Run it from the command line or you won't see the output!

1. In your Azure Sphere Developer Command Prompt Preview window (or any shell you prefer), change to the Tools directory, then run the utility
   a. Cd <your repo location>/ Samples\AvnetAzureSphereHacksterTTC\HighLevelExampleApp\Tools\win-x64
   b. **Do NOT run this from the file explorer!** The application will exit at the end and you won’t be able to see the output.

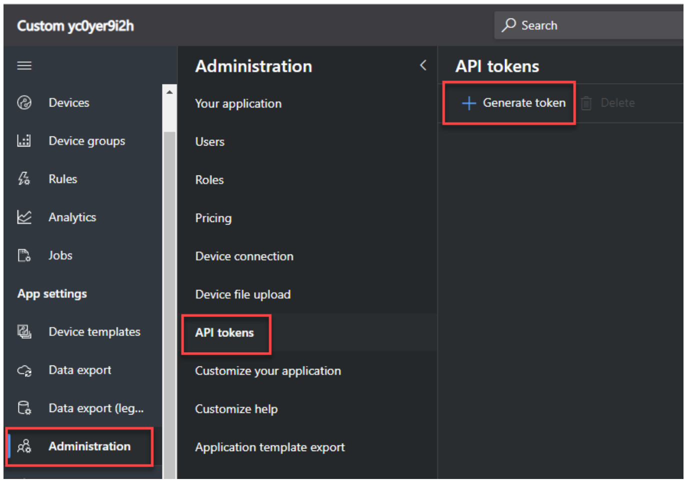
2. When asked if you're using a legacy (2018) IoT Central application (Y/N): Answer **N**

3. Next you'll be prompted for the URL for your IoT Central application. Copy the URL from your browser window and paste it into the command prompt window. You only need the URL up to and including the *azureiotcentral.com.

4. Next you'll be asked for an Azure IoT Central application API Token

   a. Go back to your IoT Central Application

   b. Select Administration → API tokens → "+ Generate token"

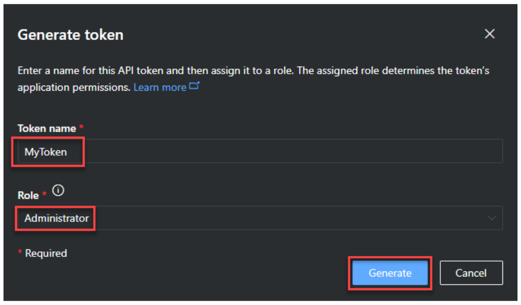


   c. The Generate token dialog is opened

      i. Type in a token name (any name is fine)

      ii. Verify that "Administrator" is selected for the Role

      iii. Click the Generate button



Enter a name for this API token and then assign it to a role. The assigned role determines the token's application permissions. Learn more

Token name *: MyToken

Role *: Administrator

* Required

   d. The token is generated and displayed. Copy the token and paste it into the command prompt window

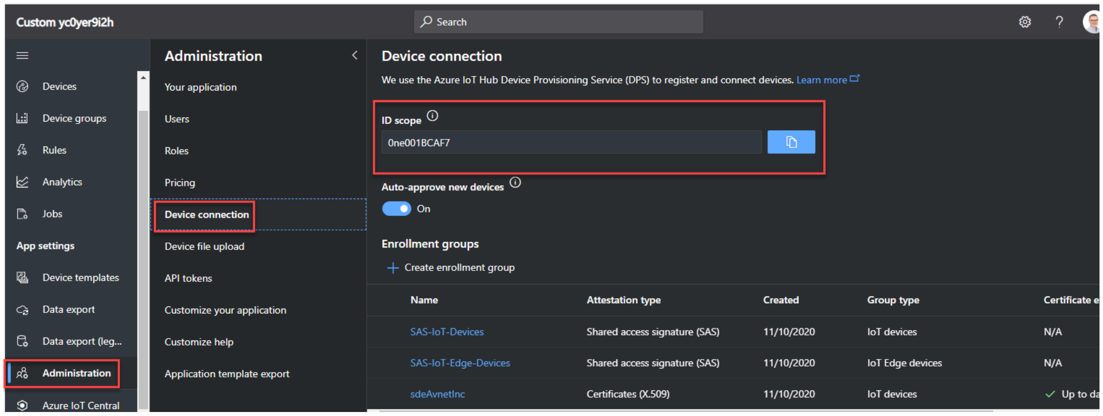
5. Next we need to find the ID Scope and provide that detail to the utility

    a. The ID Scope is for the Device Provisioning Service that's deployed for your IoT Central application

    b. Back in your IoT Central Application select Administration → Device Connection

    c. Copy the ID scope using the copy button

    d. Paste the ID scope into the command prompt window



Device connection

We use the Azure IoT Hub Device Provisioning Service (DPS) to register and connect devices. Learn more

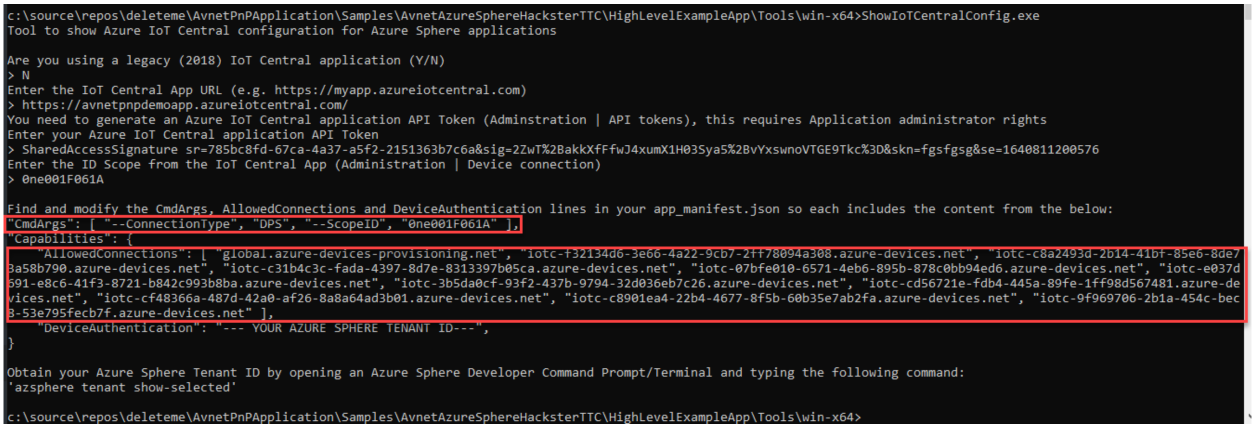
ID scope ⓘ
0ne001BCAF7

Auto-approve new devices ⓘ
On

Enrollment groups
+ Create enrollment group

Name	Attestation type	Created	Group type	Certificate e
SAS-IoT-Devices	Shared access signature (SAS)	11/10/2020	IoT devices	N/A
SAS-IoT-Edge-Devices	Shared access signature (SAS)	11/10/2020	IoT Edge devices	N/A
sdeAvnetInc	Certificates (X.509)	11/10/2020	IoT devices	✓ Up to da

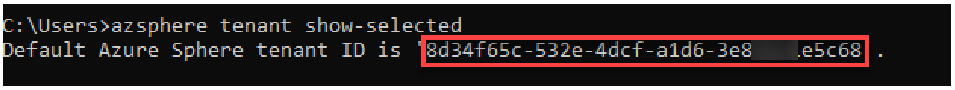
6. The utility will generate entries for your Azure Sphere applications app_manifest.json file. See the graphic below for the items we need to copy into our app_manifest.json file.



```
c:\source\repos\deleteme\AvnetPnPApplication\Samples\AvnetAzureSphereHacksterTTC\HighLevelExampleApp\Tools\win-x64>ShowIoTCentralConfig.exe
Tool to show Azure IoT Central configuration for Azure Sphere applications

Are you using a legacy (2018) IoT Central application (Y/N)
> N
Enter the IoT Central App URL (e.g. https://myapp.azureiotcentral.com)
> https://avnetpnpdemoapp.azureiotcentral.com/
You need to generate an Azure IoT Central application API Token (Adminstration | API tokens), this requires Application administrator rights
Enter your Azure IoT Central application API Token
> SharedAccessSignature sr=785bc8fd-67ca-4a37-a5f2-2151363b7c6a&sig=2ZwT%2BakkXfFfwJ4xumX1H03Sya5%2BvYxswnoVTGE9Tkc%3D&skn=fgsfgsg&se=1640811200576
Enter the ID Scope from the IoT Central App (Administration | Device connection)
> 0ne001F061A

Find and modify the CmdArgs, AllowedConnections and DeviceAuthentication lines in your app_manifest.json so each includes the content from the below:
"CmdArgs": [ "--ConnectionType", "DPS", "--ScopeID", "0ne001F061A" ],
"Capabilities": {
    "AllowedConnections": [ "global.azure-devices-provisioning.net", "iotc-f32134d6-3e66-4a22-9cb7-2ff78094a308.azure-devices.net", "iotc-c8a2493d-2b14-41bf-85e6-8de7
3a58b790.azure-devices.net", "iotc-c31b4c3c-fada-4397-8d7e-8313397b05ca.azure-devices.net", "iotc-07bfe010-6571-4eb6-895b-878c0bb94ed6.azure-devices.net", "iotc-e037d
691-e8c6-41f3-8721-b842c993b8ba.azure-devices.net", "iotc-3b5da0cf-93f2-437b-9794-32d036eb7c26.azure-devices.net", "iotc-cd56721e-fdb4-445a-89fe-1ff98d567481.azure-de
vices.net", "iotc-cf48366a-487d-42a0-af26-8a8a64ad3b01.azure-devices.net", "iotc-c8901ea4-22b4-4677-8f5b-60b35e7ab2fa.azure-devices.net", "iotc-9f969706-2b1a-454c-bec
8-53e795fecb7f.azure-devices.net" ],
    "DeviceAuthentication": "--- YOUR AZURE SPHERE TENANT ID---",
}

Obtain your Azure Sphere Tenant ID by opening an Azure Sphere Developer Command Prompt/Terminal and typing the following command:
'azsphere tenant show-selected'

c:\source\repos\deleteme\AvnetPnPApplication\Samples\AvnetAzureSphereHacksterTTC\HighLevelExampleApp\Tools\win-x64>
```

7. The Azure Sphere Tenant GUID can be obtained from the azsphere command line tool

    a. **azsphere tenant show-selected**

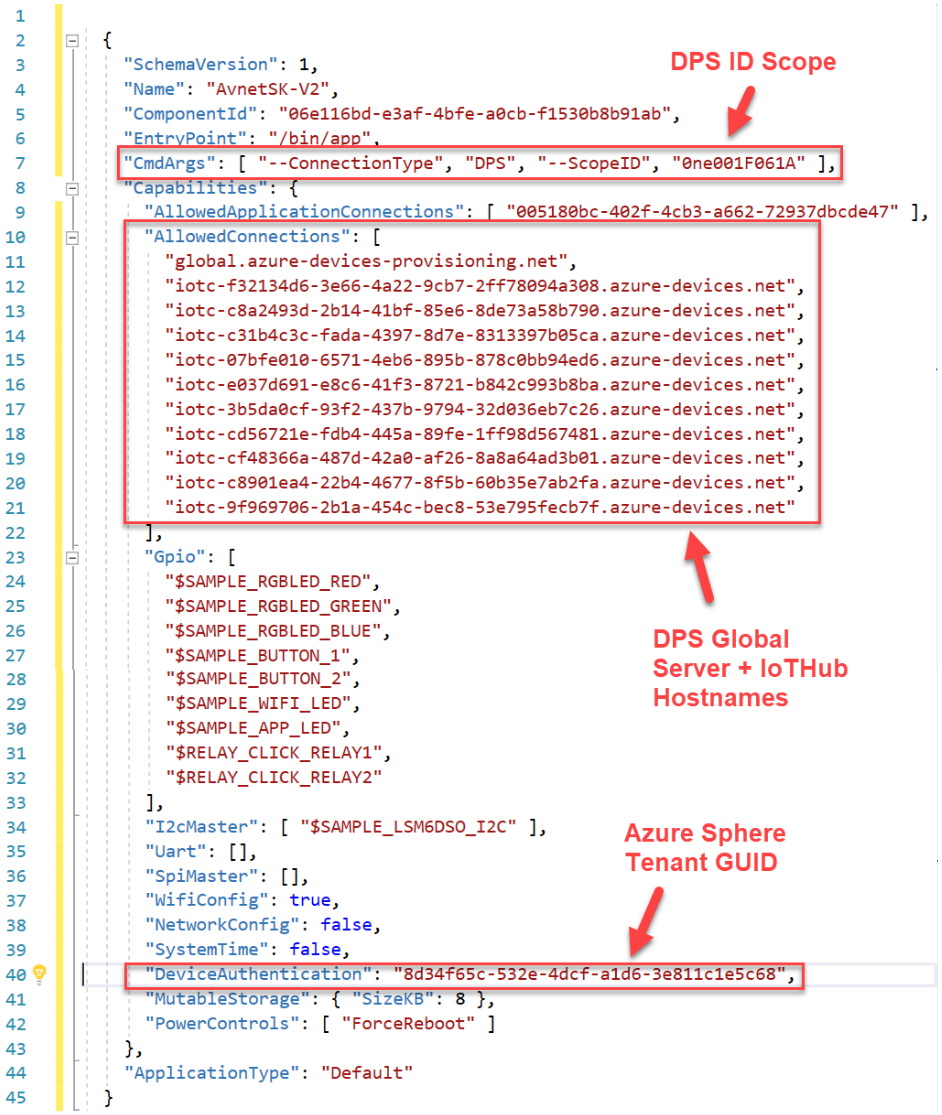
    b. 

```
C:\Users>azsphere tenant show-selected
Default Azure Sphere tenant ID is '8d34f65c-532e-4dcf-a1d6-3e8[redacted]e5c68 .
```

Modify the Azure Sphere source code

Once we have all the information we need, we can update our application!

- Go back to your Visual Studio application
- Open the app_manifest.json file from the Solution Explorer
- Update the following items with the data we just collected from the ShowIoTConfig.exe utility
- `"CmdArgs": [ "--ConnectionType", "DPS", "--ScopeID", "<your ID Scope>" ],`
- "AllowedConnections": ["global.azure-devices-provisioning.net", **+ all the hostnames for your application**],
- "DeviceAuthentication": "<Your Azure Sphere Tenant GUID>",
- Save the file
- My updated file is shown below . . .



```

{
  "SchemaVersion": 1,
  "Name": "AvnetSK-V2",
  "ComponentId": "06e116bd-e3af-4bfe-a0cb-f1530b8b91ab",
  "EntryPoint": "/bin/app",
  "CmdArgs": [ "--ConnectionType", "DPS", "--ScopeID", "0ne001F061A" ],
  "Capabilities": {
    "AllowedApplicationConnections": [ "005180bc-402f-4cb3-a662-72937dbcde47" ],
    "AllowedConnections": [
      "global.azure-devices-provisioning.net",
      "iotc-f32134d6-3e66-4a22-9cb7-2ff78094a308.azure-devices.net",
      "iotc-c8a2493d-2b14-41bf-85e6-8de73a58b790.azure-devices.net",
      "iotc-c31b4c3c-fada-4397-8d7e-8313397b05ca.azure-devices.net",
      "iotc-07bfe010-6571-4eb6-895b-878c0bb94ed6.azure-devices.net",
      "iotc-e037d691-e8c6-41f3-8721-b842c993b8ba.azure-devices.net",
      "iotc-3b5da0cf-93f2-437b-9794-32d036eb7c26.azure-devices.net",
      "iotc-cd56721e-fdb4-445a-89fe-1ff98d567481.azure-devices.net",
      "iotc-cf48366a-487d-42a0-af26-8a8a64ad3b01.azure-devices.net",
      "iotc-c8901ea4-22b4-4677-8f5b-60b35e7ab2fa.azure-devices.net",
      "iotc-9f969706-2b1a-454c-bec8-53e795fecb7f.azure-devices.net"
    ],
    "Gpio": [
      "$SAMPLE_RGBLED_RED",
      "$SAMPLE_RGBLED_GREEN",
      "$SAMPLE_RGBLED_BLUE",
      "$SAMPLE_BUTTON_1",
      "$SAMPLE_BUTTON_2",
      "$SAMPLE_WIFI_LED",
      "$SAMPLE_APP_LED",
      "$RELAY_CLICK_RELAY1",
      "$RELAY_CLICK_RELAY2"
    ],
    "I2cMaster": [ "$SAMPLE_LSM6DSO_I2C" ],
    "Uart": [],
    "SpiMaster": [],
    "WifiConfig": true,
    "NetworkConfig": false,
    "SystemTime": false,
    "DeviceAuthentication": "8d34f65c-532e-4dcf-a1d6-3e811c1e5c68",
    "MutableStorage": { "SizeKB": 8 },
    "PowerControls": [ "ForceReboot" ]
  },
  "ApplicationType": "Default"
}
```

DPS ID Scope

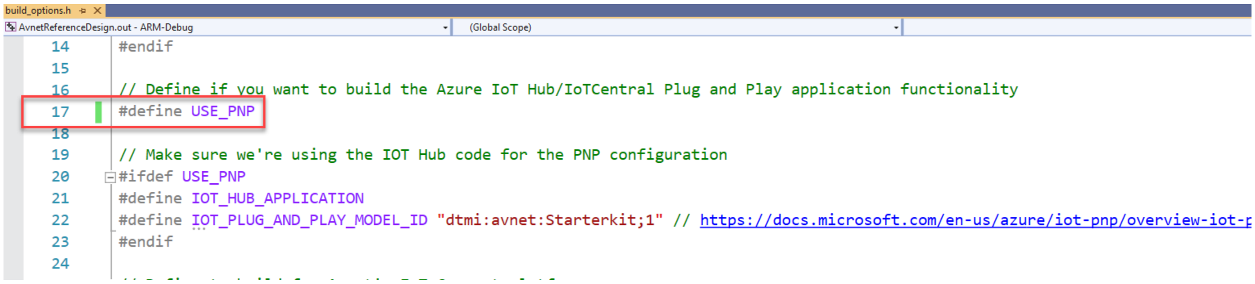
DPS Global Server + IoTHub Hostnames

Azure Sphere Tenant GUID

Configure the application for the Plug and Play build

Next we'll update the build configuration to specify that we're using Plug and Play.

- Open the build_options.h file
- Remove the comment specifiers "//" from line ~#17 to enable the plug and play build

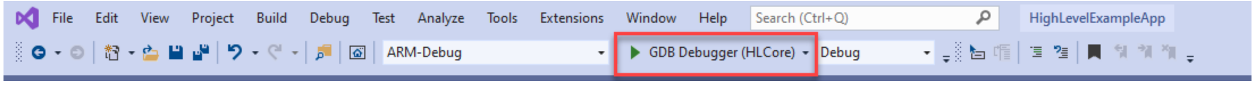


```
14  #endif
15
16  // Define if you want to build the Azure IoT Hub/IoTCentral Plug and Play application functionality
17  #define USE_PNP
18
19  // Make sure we're using the IOT Hub code for the PNP configuration
20  #ifdef USE_PNP
21  #define IOT_HUB_APPLICATION
22  #define IOT_PLUG_AND_PLAY_MODEL_ID "dtmi:avnet:Starterkit;1" // https://docs.microsoft.com/en-us/azure/iot-pnp/overview-iot-p
23  #endif
24
```

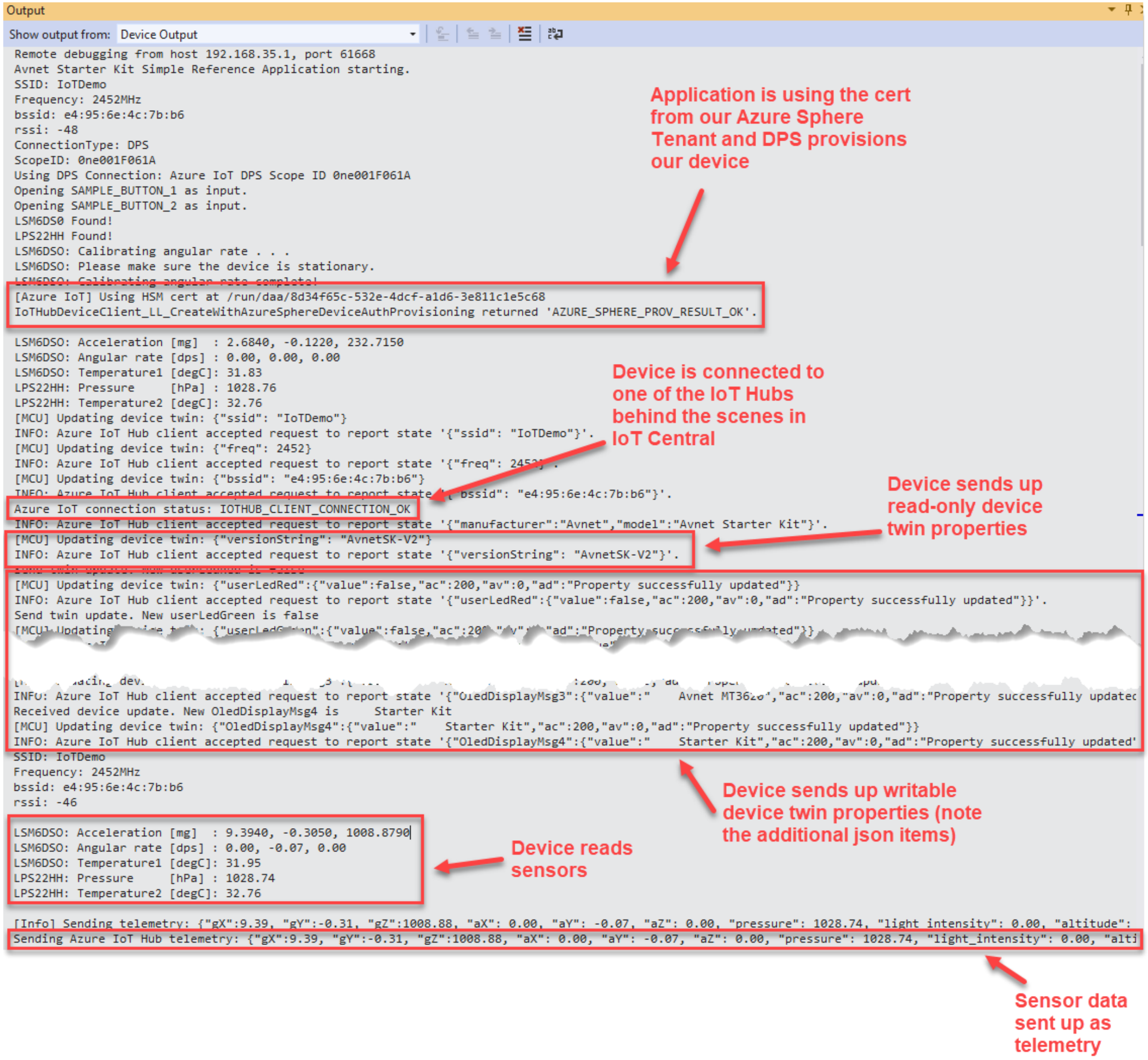
Build and Run the Application

Now let's build and run our application.

- Make sure your device is connected to your PC
- Ensure your device has a Wi-Fi connection
  - **azsphere device wifi show-status**
- Click on the "GDB Debugger (HLCore)" button at the top of the application. Your application will build, link, side-load, and run.



Initially, your application may display an “AZURE_SPHERE_PROV_RESULT_PROV_DEVICE_ERROR” message. Give it a few seconds to see if it connects. When it connects to the DPS and IoT Hub you’ll see debug similar to the graphic below.



```
Remote debugging from host 192.168.35.1, port 61668
Avnet Starter Kit Simple Reference Application starting.
SSID: IoTDemo
Frequency: 2452MHz
bssid: e4:95:6e:4c:7b:b6
rssi: -48
ConnectionType: DPS
ScopeID: 0ne001F061A
Using DPS Connection: Azure IoT DPS Scope ID 0ne001F061A
Opening SAMPLE_BUTTON_1 as input.
Opening SAMPLE_BUTTON_2 as input.
LSM6DS0 Found!
LPS22HH Found!
LSM6DSO: Calibrating angular rate . . .
LSM6DSO: Please make sure the device is stationary.
LSM6DSO: Calibrating angular rate complete!
[Azure IoT] Using HSM cert at /run/daa/8d34f65c-532e-4dcf-a1d6-3e811c1e5c68
IoTHubDeviceClient_LL_CreateWithAzureSphereDeviceAuthProvisioning returned 'AZURE_SPHERE_PROV_RESULT_OK'.

LSM6DSO: Acceleration [mg]  : 2.6840, -0.1220, 232.7150
LSM6DSO: Angular rate [dps] : 0.00, 0.00, 0.00
LSM6DSO: Temperature1 [degC]: 31.83
LPS22HH: Pressure     [hPa] : 1028.76
LPS22HH: Temperature2 [degC]: 32.76
[MCU] Updating device twin: {"ssid": "IoTDemo"}
INFO: Azure IoT Hub client accepted request to report state '{"ssid": "IoTDemo"}'.
[MCU] Updating device twin: {"freq": 2452}
INFO: Azure IoT Hub client accepted request to report state '{"freq": 2452}'.
[MCU] Updating device twin: {"bssid": "e4:95:6e:4c:7b:b6"}
INFO: Azure IoT Hub client accepted request to report state '{"bssid": "e4:95:6e:4c:7b:b6"}'.
Azure IoT connection status: IOTHUB_CLIENT_CONNECTION_OK
INFO: Azure IoT Hub client accepted request to report state '{"manufacturer":"Avnet","model":"Avnet Starter Kit"}'.
[MCU] Updating device twin: {"versionString": "AvnetSK-V2"}
INFO: Azure IoT Hub client accepted request to report state '{"versionString": "AvnetSK-V2"}'.
Send twin update. New userLedRed is false
[MCU] Updating device twin: {"userLedRed":{"value":false,"ac":200,"av":0,"ad":"Property successfully updated"}}
INFO: Azure IoT Hub client accepted request to report state '{"userLedRed":{"value":false,"ac":200,"av":0,"ad":"Property successfully updated"}}'.
Send twin update. New userLedGreen is false
[MCU] Updating device twin: {"userLedGreen":{"value":false,"ac":200,"av":0,"ad":"Property successfully updated"}}
[illegible]
INFO: Azure IoT Hub client accepted request to report state '{"OledDisplayMsg3":{"value":"    Avnet MT3620","ac":200,"av":0,"ad":"Property successfully updatec
Received device update. New OledDisplayMsg4 is     Starter Kit
[MCU] Updating device twin: {"OledDisplayMsg4":{"value":"    Starter Kit","ac":200,"av":0,"ad":"Property successfully updated"}}
INFO: Azure IoT Hub client accepted request to report state '{"OledDisplayMsg4":{"value":"    Starter Kit","ac":200,"av":0,"ad":"Property successfully updated'
SSID: IoTDemo
Frequency: 2452MHz
bssid: e4:95:6e:4c:7b:b6
rssi: -46

LSM6DSO: Acceleration [mg]  : 9.3940, -0.3050, 1008.8790
LSM6DSO: Angular rate [dps] : 0.00, -0.07, 0.00
LSM6DSO: Temperature1 [degC]: 31.95
LPS22HH: Pressure     [hPa] : 1028.74
LPS22HH: Temperature2 [degC]: 32.76

[Info] Sending telemetry: {"gX":9.39, "gY":-0.31, "gZ":1008.88, "aX": 0.00, "aY": -0.07, "aZ": 0.00, "pressure": 1028.74, "light_intensity": 0.00, "altitude":
Sending Azure IoT Hub telemetry: {"gX":9.39, "gY":-0.31, "gZ":1008.88, "aX": 0.00, "aY": -0.07, "aZ": 0.00, "pressure": 1028.74, "light_intensity": 0.00, "alti
```

Application is using the cert from our Azure Sphere Tenant and DPS provisions our device

Device is connected to one of the IoT Hubs behind the scenes in IoT Central

Device sends up read-only device twin properties

Device sends up writable device twin properties (note the additional json items)

Device reads sensors

Sensor data sent up as telemetry

If your device never connects

- Verify the data in the app_manifest.json file
- Verify your device has an internet connection
- Try a different wifi network
- See Azure Sphere network troubleshooting guide here

Conclusion

In this Getting Started Guide you learned . . .

- How to install and maintain the Azure Sphere development tools for Windows
- How to provision and configure a IoT Central Application for Azure Sphere Devices
- How to pull a GitHub project
- How to configure the Avnet Azure Sphere Example application to connect to your IoT Central application
- How to run the Avnet example application

The next step is to learn more about using the IoT Central application to view telemetry data from your device and how to control things on your device, like the LEDs from your IoTCentral application.

Please refer to this online documentation to learn how to use the IoTCentral application: Link

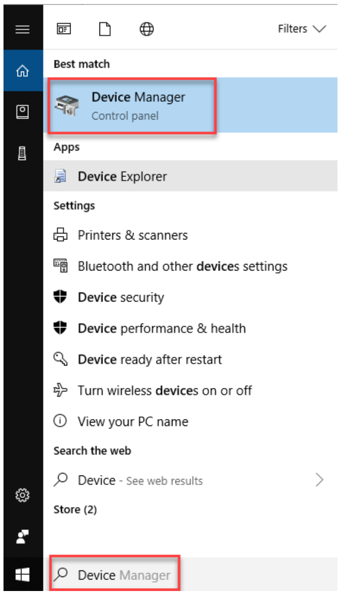
Appendix

Com Ports

If your Azure Sphere board does not communicate with the azsphere.exe utility, you may just need to install/update USB Serial port drivers. See this section for details.

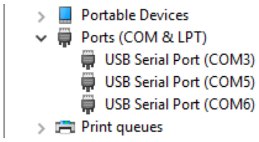
1. Connect your Azure Sphere Starter Kit to your PC using the micro USB cable provided with the kit

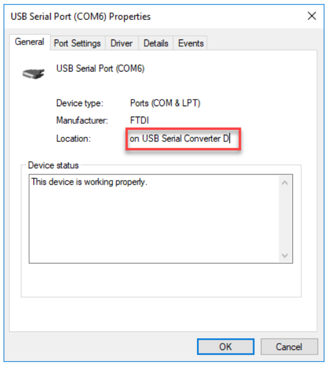
2. Open your Windows Device Manager



a. Type “Device Manager” in the Windows Search bar at the bottom of your desktop. When you see a selection for Device Manager, select it.

3. Once Device Manager is open, find the folder for “Ports (COM & LPT).” You should see three USB Serial Ports enumerated. Your COM numbers will likely be different than mine. These are the three COM ports for the Azure Sphere Starter Kit.





USB Serial Port (COM6)

Device type: Ports (COM & LPT)
Manufacturer: FTDI
Location: on USB Serial Converter D

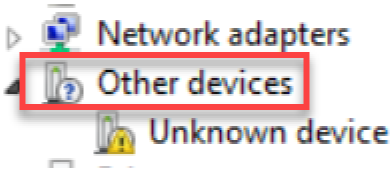
Device status
This device is working properly.

If you right click each COM port and select “**Properties**”, their “**Location**” will list one of the following:

- USB Serial Converter A
- USB Serial Converter B
- USB Serial Converter D

If this seen for the COM ports, then they are configured correctly!
If not, you will need to update the FTDI drivers (See step #4 below)

4. If you don’t see the ports listed under “Ports (COM & LPT), then they may be under a folder called “**Other Devices**”. If this is the case you need to update the drivers.



5. To update the drivers, first download the FTDI drivers from the FTDI website here. Then right-click on the “**Unknown device**” and click “**Update Driver**”. This will open a dialog box where you can browse to the folder where you put your FTDI drivers. You will need to repeat this process for each COM port.

6. Note, I’ve seen this process require two different driver updates. You’ll know the drivers are installed correctly if the COM port properties are as described in step 3.a above.

Revision History

Date	Version	Revision
30 Dec 2020	01	Preliminary release