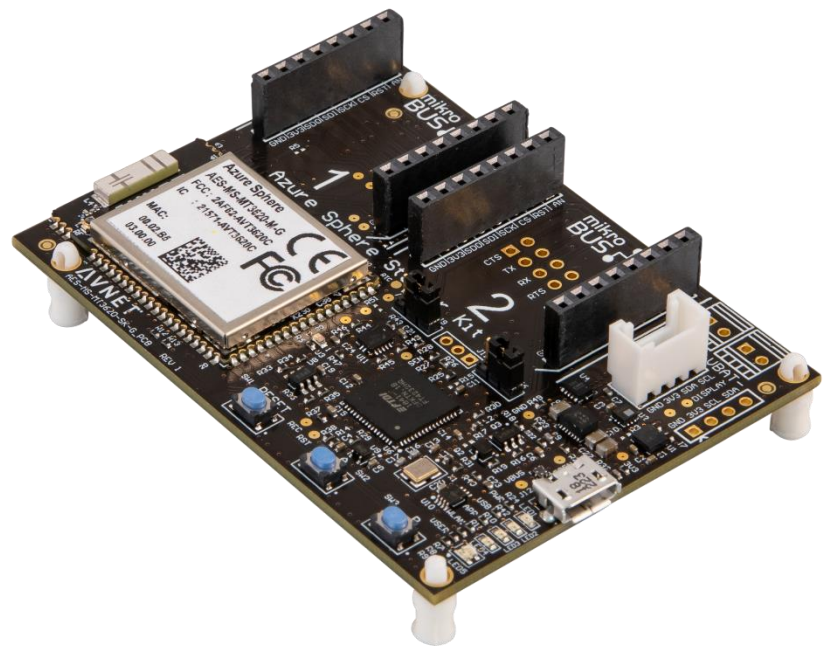


Avnet Technical Training Course

Azure Sphere: Accelerating your Cloud Dashboard Implementation Lab 5



Azure Sphere SDK:	19.11
Training Version:	v3
Date:	14 January 2020

© 2019 Avnet. All rights reserved. All trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

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

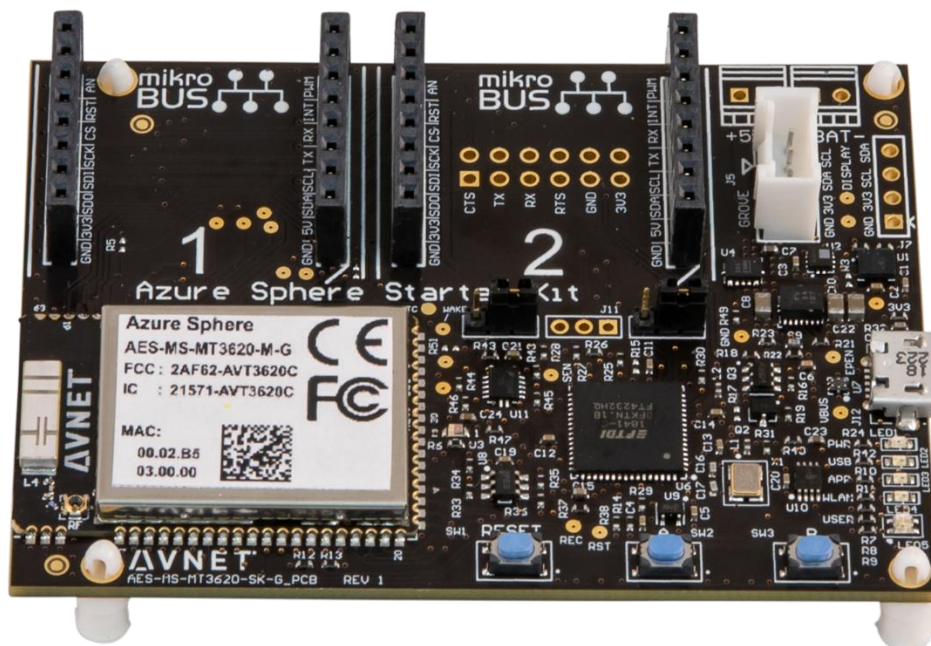
This Lab will walk the student through connecting the example application to an IoT Central application. We'll use a template that I've prepared to make creating the IoT Central application fast and easy. I'll provide a coding assignment to help reinforce some of the Azure Sphere coding that we've covered in the course. Next we'll modify the IoT Central application to add a new LED control, and we'll complete the addition with a coding assignment.

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.



Avnet Azure Sphere Starter Kit

Lab 5: Objectives

The Lab-5 objectives are to teach the student how to stand up an IoT Central application and send telemetry data from the Azure Sphere device to the cloud so we can visualize the telemetry data.

- Learn how to create an IoT Central application from a template
- Learn how to provision our device to the IoT Central application
- Configure the example application for the IoT Central configuration
- Complete a code assignment
- Modify the IoT Central Application to display the new item created in the coding assignment

Lab-5 builds on the previous labs and should not be started until Labs 0-3 have been completed.

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

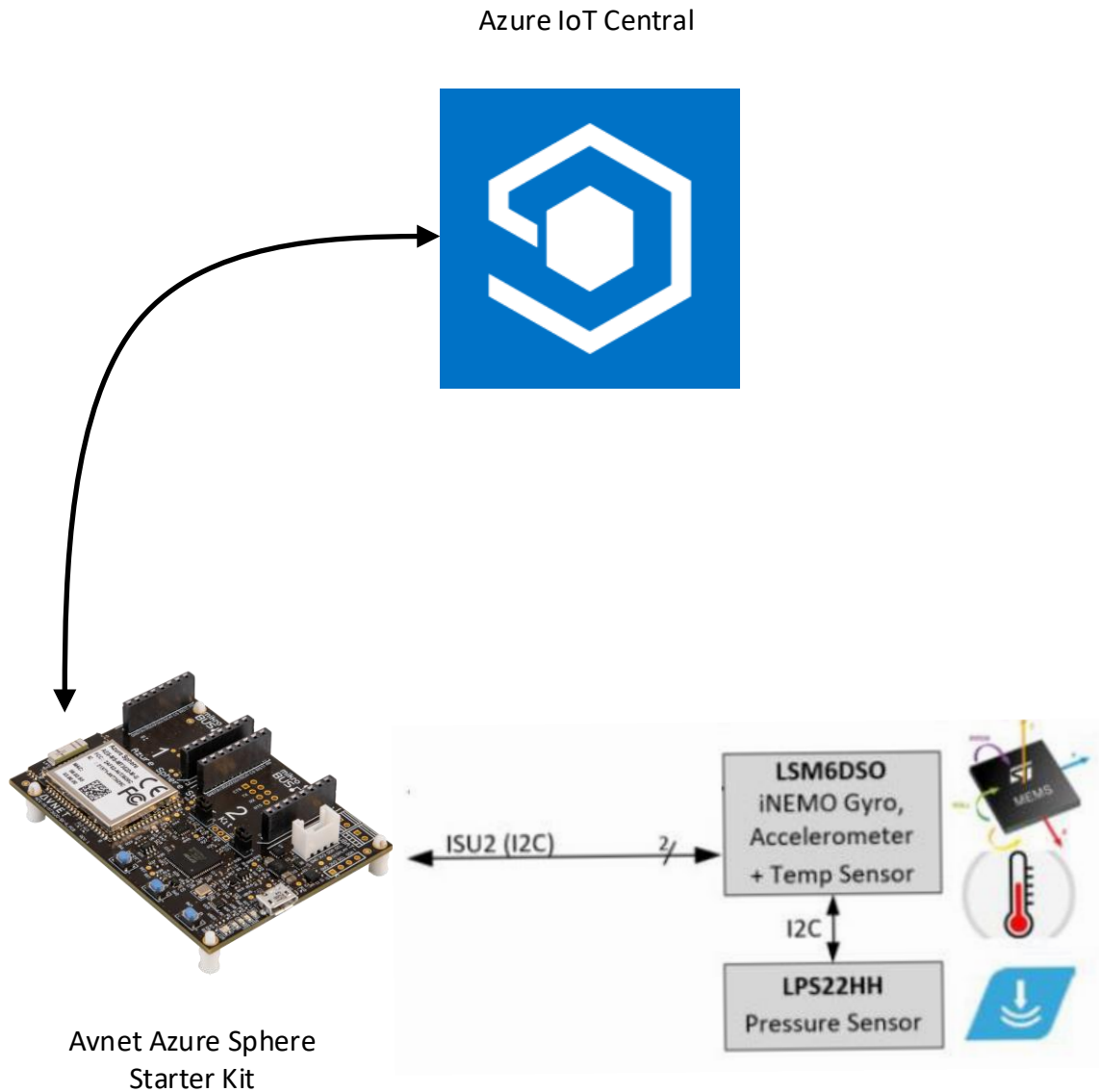
- Visual Studio 2019 Enterprise, Professional, or Community version 16.04 or later; or Visual Studio 2017 version 15.9 or later **installed**
- Azure Sphere SDK 19.11 or the current SDK release **installed**

Other

- Your Azure Sphere device must be connected to a Wi-Fi access point or hotspot with access to the internet.
- Labs 4 and 5 both require an Azure Account. You can sign up for a free Azure account with a \$200 credit [here](#).
 - Lab 5 does not need an Azure account if you select the Free 7-Day IoT Central Trial

The Big Picture

The diagram below shows the system we'll be building out in this lab. Starting at the bottom right of the graphic, we'll use the on-board I2C sensors to read accelerometer data. That data will be sent as telemetry to an IoT Central application.



Create an IoT Central Application

Azure IoT Central

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 the IoT Central Application from a template
- Configure DPS for our application
- Configure our Azure Sphere application to connect to the IoT Central Application

Create IoT Central Application

Recently Microsoft released a new IoT Central feature called “Application Template Export” for IoT central. This is a really nice feature that allows you to basically clone an IoT Central application that someone else created. We’ll use this feature for the lab, but I also wanted to provide a link that includes all the details for creating an IoT Central application from scratch. The process is very simple, and is fully documented [here](#). Additionally, there is a blog on Element14.com that includes all the details for creating the IoT Central Application that I used to create the template that we’ll be using today: [here](#).

Open the Template

The template that we’ll use is [here](#). When you click on the link, a browser window will open like the one shown below. Fill in the form and click “Create”

Build > New application

Hackster.io Azure Sphere TTC

This template provides a starting point for the Avnet Hackster.io Technical Training Content part #6

About your app

Application name *

URL *

 .azureiotcentral.com

☒ 7 day free trial
Supports up to five devices

Contact info

First name *

Last name *

Email

Phone number *

Country/Region *

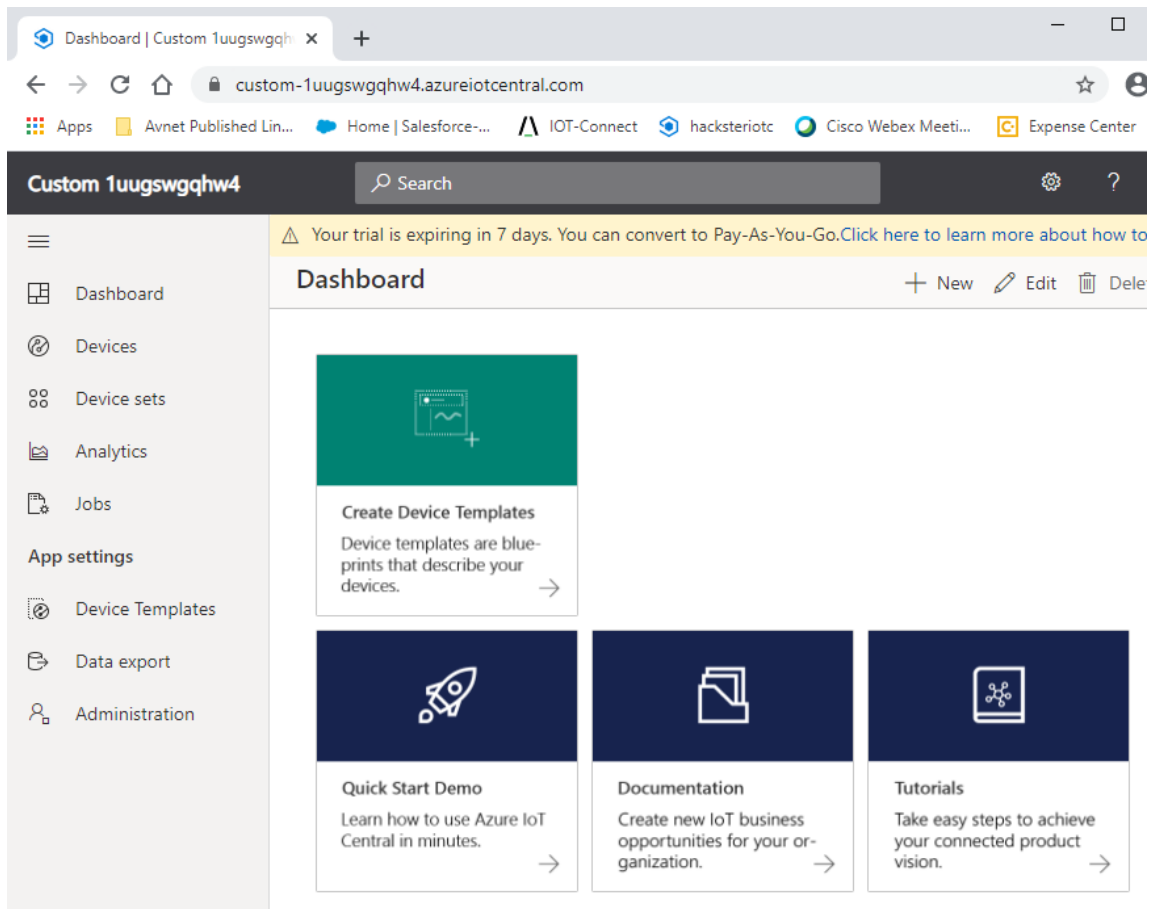
* Required

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 "Trial". "Pay-As-You-Go" requires an Azure subscription, and you acknowledge that this service is licensed to you under the terms applicable to your [Azure Subscription](#).

☐ I would like information, tips, and offers about Azure and other Microsoft products and services. [Privacy Statement](#)

- 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.
- Contact Info
 - Fill this section out with your contact information
- 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.



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. Remember in Lab-4 we had to prove to the DPS that we owned the Azure Sphere Tenant, we need to complete a similar exercise here for the IoT Central application.

Download the public tenant CA certificate

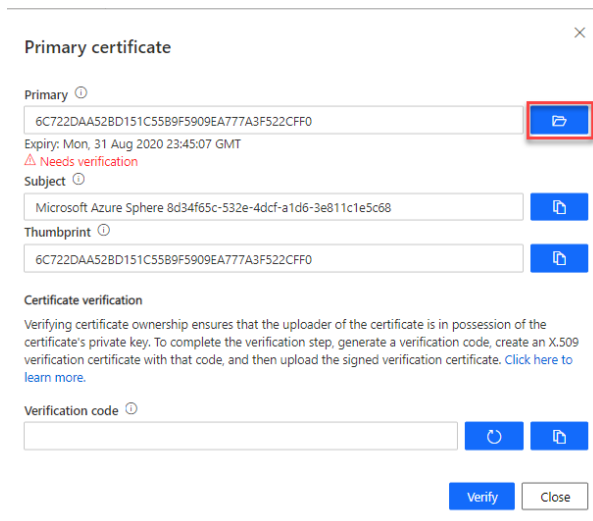
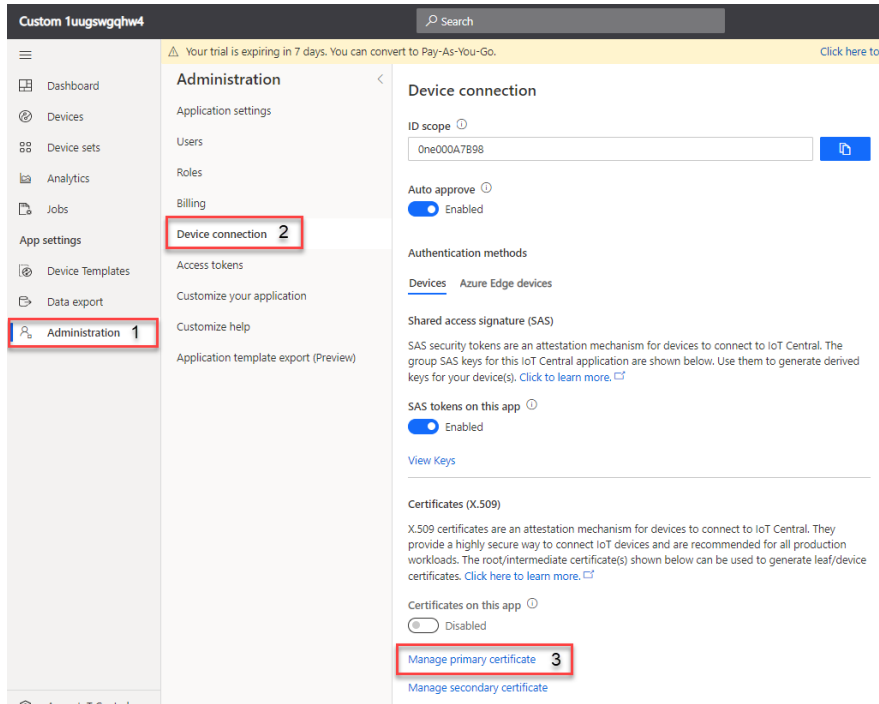
If you completed Lab-4, then you already have this certificate. If not, follow the instructions below.

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

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


In your new IoT Central application

- Select the Administration → Device connection
- Select the “Manage primary certificate” link under the “Certificates (X.509)” section
- Select the Primary folder icon and browse to the tenant CA certificate you downloaded from your tenant. Make sure the view filter is set to All files (*) if you don't see your certificate.



• Your certificate is uploaded and a notification is displayed stating that the certificate “Needs verification.”

• The “Subject” and “Thumbprint” fields are automatically filled out from the details included in your certificate.


- Click on the  icon to generate the “verification code”
- A “verification code is generated and displayed



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 tenant download-validation-certificate --output ValidationCertification.cer --verificationcode <code>`

Back in your IoT Central Application

- Copy the verification code, there's a copy link to the right of the box

Verification code ⓘ
A1CAD02CA6FD6DAB97253D83090C873DBC8F81F541CC58268 





- Back in the command window, replace the <code> text with your verification code and execute the command
- Execute the command

The Azure Sphere Security Service generates the validation certificate and includes the verification code inside the certificate. Since AS3 generated 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 verification code is correct.

```
C:\Users\...\Documents>azsphere tenant download-validation-certificate --output c:\temp\12-24-ValidationCertification.cer --verificationcode A1CAD02CA6FD6DAB97253D83090C873DBC8F81F541CC58268
Saving the validation certificate to 'c:\temp\12-24-ValidationCertification.cer'.
Saved the validation certificate to 'c:\temp\12-24-ValidationCertification.cer'.
```





- 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 ⓘ
6C722DAA52BD151C55B9F5909EA777A3F522CFF0 
Expiry: Mon, 31 Aug 2020 23:45:07 GMT

Subject ⓘ
Microsoft Azure Sphere 8d34f65c-532e-4dcf-a1d6-3e811c1e5c68 
Thumbprint ⓘ
6C722DAA52BD151C55B9F5909EA777A3F522CFF0 

Certificate verification

Verifying certificate ownership ensures that the uploader of the certificate is in possession of the certificate's private key. To complete the verification step, generate a verification code, create an X.509 verification certificate with that code, and then upload the signed verification certificate. [Click here to learn more.](#)

Verification code ⓘ
 
 

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

Microsoft is still working to seamlessly provide IoT Central connection details for Azure Sphere devices. There have been a couple of iterations, and they are working to make it easy and seamless. However, we're not quite there yet. Hopefully, I'll be updating this lab again soon with a final seamless solution. Until then, I'll document two different ways to connect our application/device to our new IoT Central application. Configuring our application/device to connect to our application involves finding some key details about our IoT Connect Application and our Tenant.

Once we have the required details, 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 application.

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

- The DPS scope ID
- Our 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

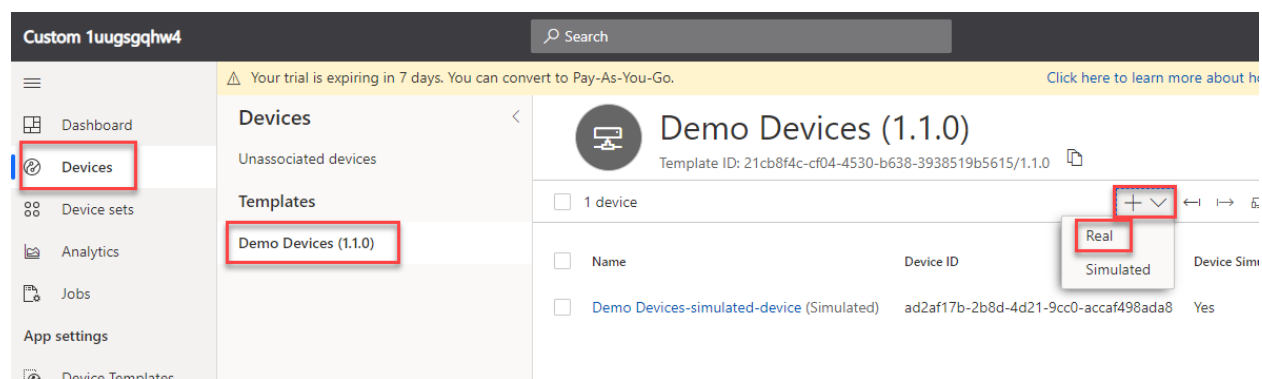
Method #1: ShowIoTCentralConfig.exe

Microsoft provides a utility to help us find the required information, however it only works with IoT Central Applications built using older templates. Fortunately for us, the template we used above was built using an older template. If you're using a newer template, skip to the section titled "Method #2 Find the details ourselves" a few pages down.

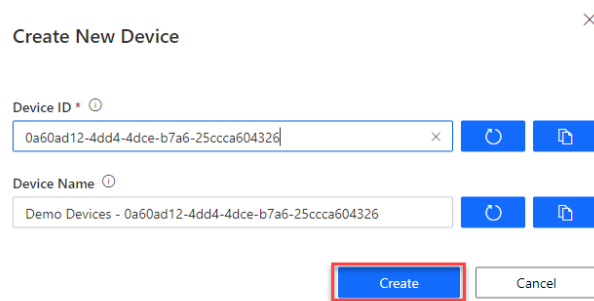
Create a "real" device

Before we can use the utility, we need to create a device in the application. The ShowIoTCentralConfig.exe utility will not pull the information from our application without it.

1. Open your IoT Central Application
2. Select Devices → Demo Devices (1.1.0) → "+" → Real

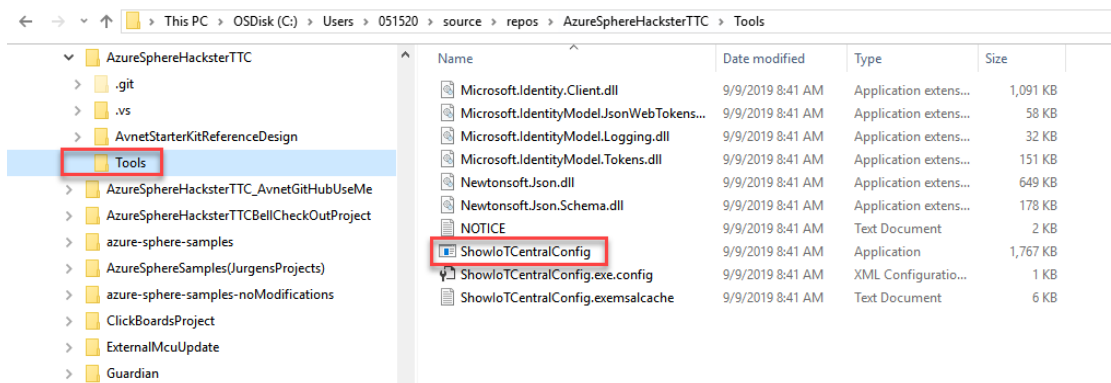


3. The “Create New Device” dialog will open

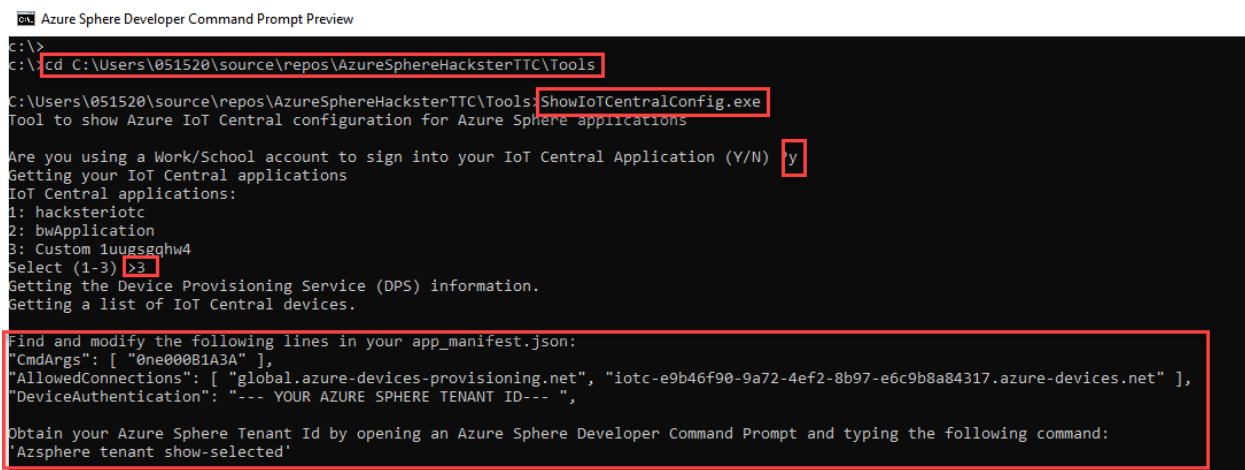


4. Keep the default values and click the “Create” button

Now we’re ready to run the utility. The ShowIoTCentralConfig.exe utility was included in the GitHub project pulled for the lab.



5. In your Azure Sphere Developer Command Prompt Preview window, change to the Tools directory, then run the utility
6. Answer “y” when asked if you’re are using a Work/School account, **DO NOT PRESS ENTER** after entering the “y”. This is a known bug with this utility, if you press “y”, then enter the application will exit after displaying the IoT Central applications in your subscription.
7. You’ll be asked to login to your subscription, do so using your azure account credentials.



8. The utility will output details for your app_manifest.json file.
9. Execute the `azsphere tenant show-selected` command to find your tenant Id for the "DeviceAuthentication" field.
10. You can skip the next section, find the "Modify the Azure Sphere source code" section and continue working on the lab.

Method #2: Find the details ourselves

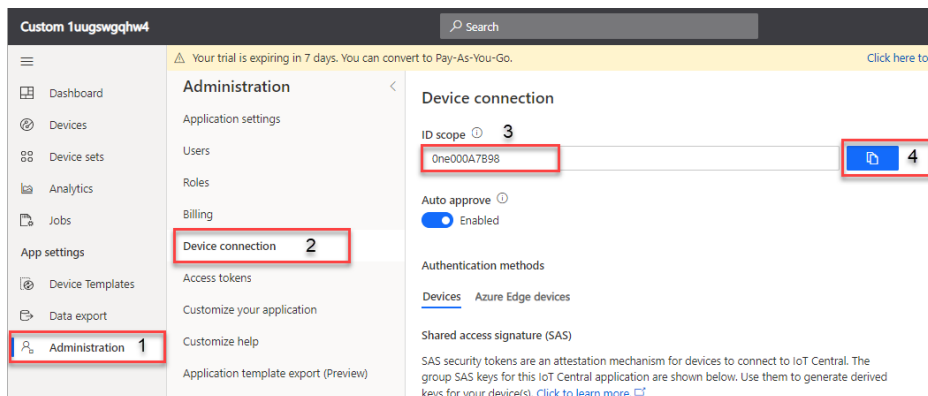
If you find yourself creating your own IoT Central Application using the newer implementation, you may not be able to use the ShowIoTCentralConfig.exe utility. But don't worry, we can still find all the required information we need, it's just a little messy. Some of this information we have easy access to, however the FQDN for the IoT Hub is buried in the implementation and we need to use a command line utility that Microsoft provides to get that information.

As a reminder, to configure our application we need to find the following pieces of information:

- The DPS scope ID
- Our 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

DPS scope ID

The DPS scope ID can be found in the IoT Central Application Administration → Device connection → ID scope



Azure Sphere Tenant GUID

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

- azsphere tenant show-selected

```
C:\Users>azsphere tenant show-selected
Default Azure Sphere tenant ID is 8d34f65c-532e-4dcf-a1d6-3e8e5c68 .
```

FQDN for the global DPS endpoint

This is the global DPS server. All IoT Hub DPS connections initially connect to this address. Once the server is contacted, the DPS Scope ID is used to route the request to our specific DPS instance.

- This will be the same for all Azure Sphere Applications
- `global.azure-devices-provisioning.net`

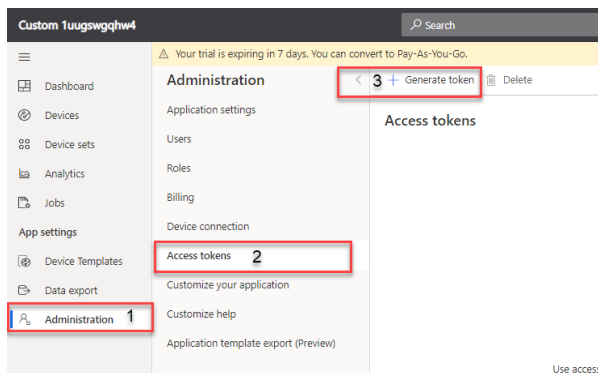
FQDN(s) for the IoT Hub

This is the messy one. Since our Azure Sphere application needs explicit permission to access any server, and our application connects to Azure using an IoT Hub, we need to determine the IoT Hub hostname(s). This/these hostname(s) are added to our “AllowedConnection” entry in the app_manifest.json file. The IoT Hub hostname is not available in the IoT Central application so we’ll use a tool called iotc-reserved-hubs to uncover the data.

1. Install the [iotc-reserved-hubs utility](#)
 - a. Open your Windows Powershell application
 - b. Enter and execute the command: “npm i -g iotc-reserved-hubs” to install the utility
 - i. Note this requires NodeJS Version 10 or higher
 1. NodeJS [Installation instructions](#) if needed
2. Generate an IoT Central Access Token

The iotc-reserved-hubs utility requires an access token for the IoT Central application

- a. Open your IoT Central Application
- b. Navigate to Administration → Access tokens → + Generate token



- c. Give your token a name, I used “MyToken” and click the generate link

- d. The token is generated and displayed. **Note, this is the only time you'll be able to see the token, make a copy of it.**

Token successfully generated

Make sure you copy this token. It will not be shown again.



3. Find your IoT Central URL

- a. We need to pass in the URL for our application into the iotc-reserved-hubs tool. Since I have my IoT Central application open, I just copy the URL from my web browser. We only need the base URL, that is all text up to and including "https://*.azureiotcentral.com."

4. Using the iotc-reserved-hubs tool, dump all the IoT Hub hostnames associated with our application

- a. From the Windows Powershell command line execute the tool using the format

- i. iotc-reserved-hubs -u <IoT Central URL> -t "<access token>"

1. Make sure to include ""s around your access token, otherwise there will be spaces inserted into the string and the command will fail.

```
PS C:\> iotc-reserved-hubs -u https://custom-1uugswgqhw4.azureiotcentral.com -t "SharedAccessSignature sr=74eb064d-8603-4b2c-a332-0e4ae6dc7bf8&sig=aOz1fs%2Fr*Y5AnvUadbnS0E1HQJG0P1xOK1pR7JMBDJz0%3D&skn=MyToken&se=1608828568307"

<=====>

***** IoTHub FQDNs associated with the provided IoT application *****

<=====>

HubName1: iotc-41b42b08-3928-4450-9f33-79eaa9db7b41.azure-devices.net
HubName2: iotc-e2a382b5-696f-4f3a-a0d1-b2f49ea2c385.azure-devices.net
HubName3: iotc-747fff07-0474-4bb1-995f-143677862f81.azure-devices.net
HubName4: iotc-bcf631f4-996d-4d43-8392-6aea7e8d339b.azure-devices.net
HubName5: iotc-a1f5f81a-fcdd-4586-a016-5c2f2de062d6.azure-devices.net
HubName6: iotc-faa9beb3-5695-45f2-8d3f-fd77df1d0508.azure-devices.net
HubName7: iotc-5a46dae3-e6fb-42cb-b415-096334a34325.azure-devices.net
HubName8: iotc-daafc8f6-4e1b-4808-8df7-660597afef32.azure-devices.net
HubName9: iotc-bb82b811-8f86-4642-a881-bff83e6e01a6.azure-devices.net
HubName10: iotc-dc8bc7c-e199-4979-8e4a-485569e40197.azure-devices.net

<=====>
```

- b. Note that there are 10 different IoT Hubs associated with my application! Keep this data handy, we'll use it when we update our application code.

Modify the Azure Sphere source code

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

- Launch the Visual Studio application and open the “**AvnetStarterKitReferenceDesign**” project. Visual Studio keeps a list of recent projects, your project should be found in that list.

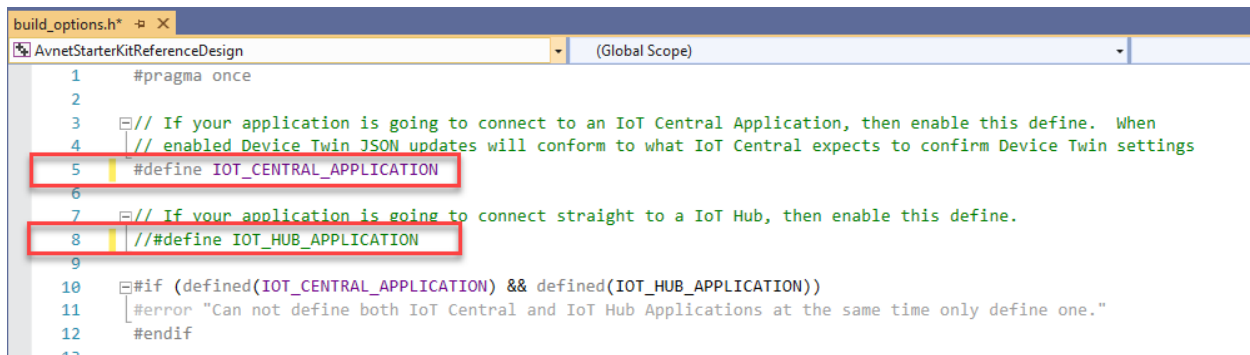
If you completed Lab-4 then you already have entries for the CmdArgs, AllowedConnections, and DeviceAuthentication in your app_manifest.json file. If you want to save these settings, make a backup copy of your app_manifest.json file.

- Open the app_manifest.json file from the Solution Explorer
- Update the following items with the data we just collected
 - "CmdArgs": ["<your DPS scope ID>"],
- "AllowedConnections": ["global.azure-devices-provisioning.net", + **the hostname(s) for your application**],
- "DeviceAuthentication": "<Your Azure Sphere Tenant GUID>",
- Save the file

Configure the application for the IoT Central build

Next we'll update the build configuration to specify that we're connecting to and IoT Central application.

- Open the build_options.h file
- On line #5 remove the “//”s to enable the IOT_CENTRAL_APPLICATION build option
- Confirm that line #8 is commented out

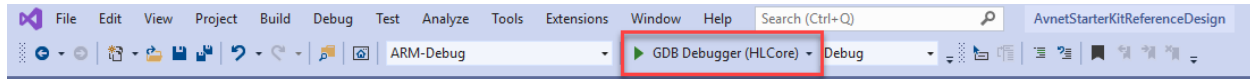


```
1  #pragma once
2
3  // If your application is going to connect to an IoT Central Application, then enable this define. When
4  // enabled Device Twin JSON updates will conform to what IoT Central expects to confirm Device Twin settings
5  #define IOT_CENTRAL_APPLICATION
6
7  // If your application is going to connect straight to a IoT Hub, then enable this define.
8  //#define IOT_HUB_APPLICATION
9
10 #if (defined(IOT_CENTRAL_APPLICATION) && defined(IOT_HUB_APPLICATION))
11     #error "Can not define both IoT Central and IoT Hub Applications at the same time only define one."
12 #endif
13
```

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 will display an “AZURE_SPHERE_PROV_RESULT_PROV_DEVICE_ERROR” message. This is because the device has connected to the IoT Central application, but the device has not been “Approved.”

```
Output
Show output from: Device Output
Remote debugging from host 192.168.35.1, port 60668

Setting Azure Scope ID 0ne000A7B98
Avnet Starter Kit Simple Reference Application starting.
LSM6DSO Found!
LPS22HH Found!

LSM6DSO: Calibrating angular rate . . .

LSM6DSO: Please make sure the device is stationary.

LSM6DSO: Calibrating angular rate complete!

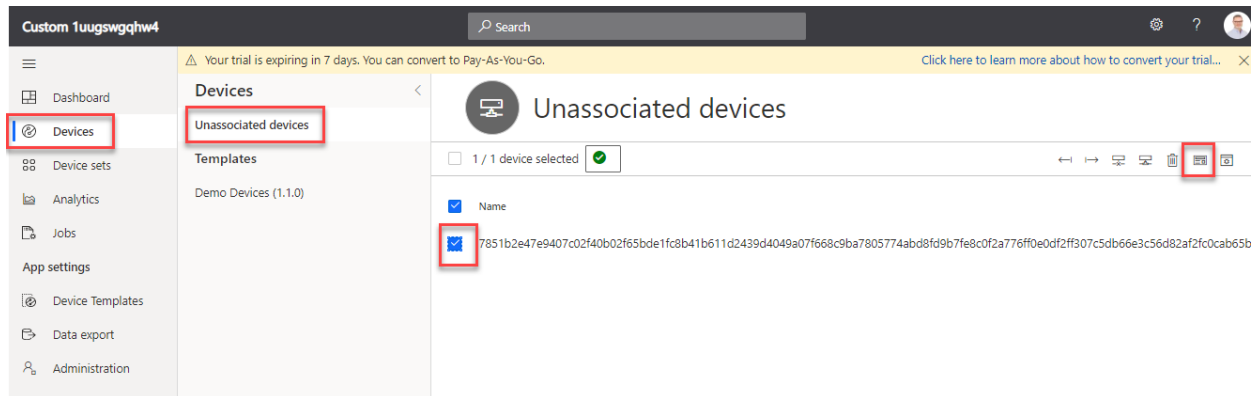
Opening Starter Kit Button A as input.
Opening Starter Kit Button B as input.
[Azure IoT] Using HSM cert at /run/daa/8d34f65c-532e-4dcf-a1d6-3e811c1e5c68

[Azure IoT Hub client]
IoHubDeviceClient_CreateWithAzureSphereDeviceAuthProvisioning returned 'AZURE_SPHERE_PROV_RESULT_PROV_DEVICE_ERROR'.
ERROR: Failed to set up IoT Hub client
SSID: willessnetwork
Frequency: 2417MHz
bssid: 10:da:43:73:db:3b
[MCU] Updating device twin: {"ssid": {"value": "willessnetwork", "status": "completed", "desiredVersion": 0 }}
[Azure IoT Hub client] ERROR: client not initialized
[MCU] Updating device twin: {"freq": 2417}
[Azure IoT Hub client] ERROR: client not initialized
[MCU] Updating device twin: {"bssid": "10:da:43:73:db:3b"}
[Azure IoT Hub client] ERROR: client not initialized
[Azure IoT Hub client] INFO: AzureIoT_DoPeriodicTasks calls in progress...

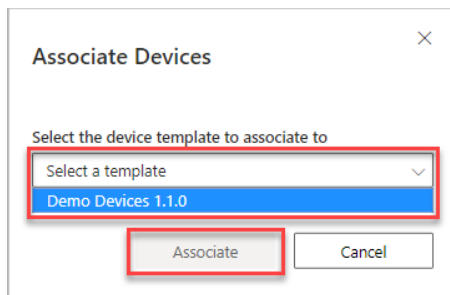
[Azure IoT Hub client] IoHubDeviceClient_CreateWithAzureSphereDeviceAuthProvisioning returned 'AZURE_SPHERE_PROV_RESULT_PROV_DEVICE_ERROR'.
ERROR: Failed to set up IoT Hub client
```


To approve the device

- Go back to your IoT Central application
- Navigate to Devices → Unassociated devices
- You should see an unassociated device listed, this should be your device's device ID
- Select the check box next to your device, and then click on the link in the upper right to associate the device with a device template



- The “Associate Devices” dialog will open
- Select the “Demo Devices 1.1.0” template from the list
- Click on the “Associate” button
- The device will be moved from the “Unassociated devices” list to the “Demo Device (1.1.0)” template



- Now go back to observe that your Azure Sphere application has been provisioned and has connected to the IoT Hub

```

Output
Show output from: Device Output
LPS22HH: Temperature [degC]: 32.45

[Info] Sending telemetry: {"gX":"10.2480", "gY":"-27.9380", "gZ":"1011.1360", "aX": "0.07", "aY": "0.00", "aZ": "0.00"}
[Azure IoT Hub client] WARNING: IoT Hub client not initialized

[Azure IoT Hub client] IoTHubDeviceClient_CreateWithAzureSphereDeviceAuthProvisioning returned 'AZURE_SPHERE_PROV_RESULT_OK'.
[MCU] Updating device twin: {"versionString": "AvnetStarterKit-Hackster.io-V1.0"}
[Azure IoT Hub client] INFO: Reported state as '{"versionString": "AvnetStarterKit-Hackster.io-V1.0"}'.

[Azure IoT Hub client] INFO: AzureIoT_DoPeriodicTasks calls in progress...

LSM6DSO: Acceleration [mg] : 10.3700, -28.0600, 1011.2580
LSM6DSO: Angular rate [dps] : 0.07, 0.00, 0.00
LSM6DSO: Temperature [degC]: 33.93
LPS22HH: Pressure [hPa] : 967.28
LPS22HH: Temperature [degC]: 32.44

[Info] Sending telemetry: {"gX":"10.3700", "gY":"-28.0600", "gZ":"1011.2580", "aX": "0.07", "aY": "0.00", "aZ": "0.00"}
[Azure IoT Hub client] INFO: IoTHubClient accepted the message for delivery

LSM6DSO: Acceleration [mg] : 3.1720, -8.2960, 296.2160
LSM6DSO: Angular rate [dps] : 0.00, 0.00, 0.00
LSM6DSO: Temperature [degC]: 33.93
LPS22HH: Pressure [hPa] : 967.26
LPS22HH: Temperature [degC]: 32.45

[Info] Sending telemetry: {"gX":"3.1720", "gY":"-8.2960", "gZ":"296.2160", "aX": "0.00", "aY": "0.00", "aZ": "0.00"}
[Azure IoT Hub client] INFO: IoTHubClient accepted the message for delivery
[Azure IoT Hub client] INFO: connection to the IoT Hub has been established (IOTHUB_CLIENT_CONNECTION_OK).

```

If you have errors . . .

- Confirm your device is connected to a Wi-Fi access point or hot spot that has internet connectivity

Once your application is running, I invite you to go to the IoT Central Application to see the telemetry and play with the settings page. I'll review the application in the lecture material.

Modify the IoT Central Application

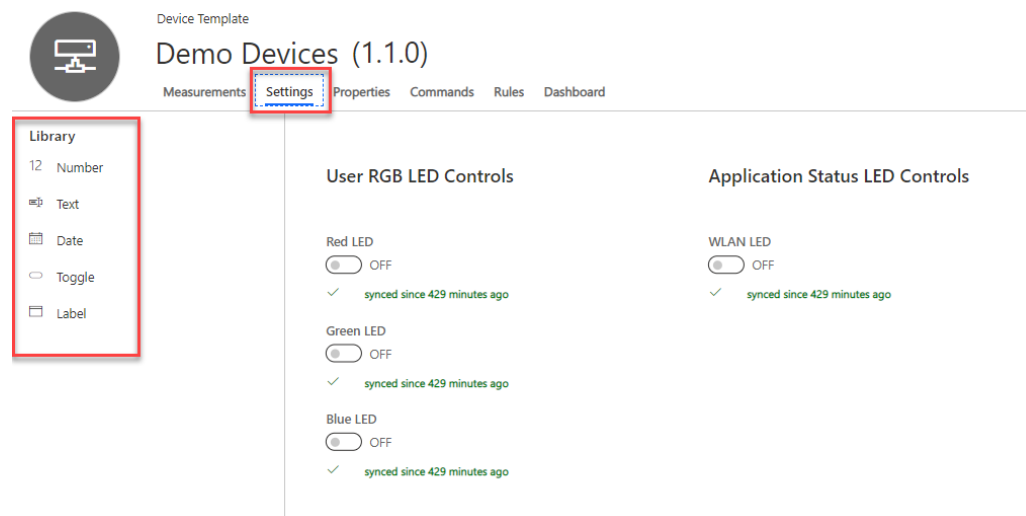
Adding telemetry or settings items to IoT Central is very easy. For all the details on how we built the IoT Central application you should read through [this blog on Element14.com](https://www.element14.com/industry/adding-telemetry-and-settings-to-the-application/). The blog shows how to add additional telemetry and settings to the application.

For this section, I'm going to walk us through adding a new item to the settings tab for our application, Application LED control. The code assignment will ask the student to add the Azure Sphere code to support this change.

- Open the IoT Central Application you created in this Lab
- From the left menu, select "Device Templates", then the only template listed "Demo Devices (1.1.0)".

A screen opens up that shows all the current Measurements, or telemetry data. The data displayed is from a simulated device. See the blog for details on adding telemetry data to this tab. We want to change the Settings Tab.

- Click on the "Settings" tab, you'll see a screen similar to the graphic below
- The items on the right side have already been configured
- The Library at the left side shows items that can be added. The cool thing about the settings entries is that they are all linked to the Device Twin items. So if you have a device twin item implemented in your Azure Sphere application you can control that item using this interface!



- We want to add a new toggle, so click on the Toggle text in the Library, the "Configure Toggle" form opens

Fill in the form as shown here

 Save  Cancel


Configure Toggle

Display Name * 


Application LED

Field Name * 

appLed

ON Display Text 

ON

OFF Display text 

OFF

Initial Value * 

☐ Off

Description 

Turns On/Off the Application LED
on the Avnet Starter Kit

* Required

Display Name: This can be anything you like. It will be displayed over the toggle control.

Field Name: This is the Key from the {"key": value} device twin entry. This entry is case sensitive and must match the key from your device twin exactly.

On Display Text: Text to be displayed with the toggle is in the on position.

Off Display Text: Text to be displayed with the toggle is in the off position.

Once you have the form filled out as shown, click on the "Save" button at the top of the form. Since we added a new item to the template you'll be prompted to create a new template version, version 1.2.0. Click the "Upgrade" button to accept the change.

Upgrade required

Some of the edits impact all the devices for this device template. Create a new template version to continue saving the edits.

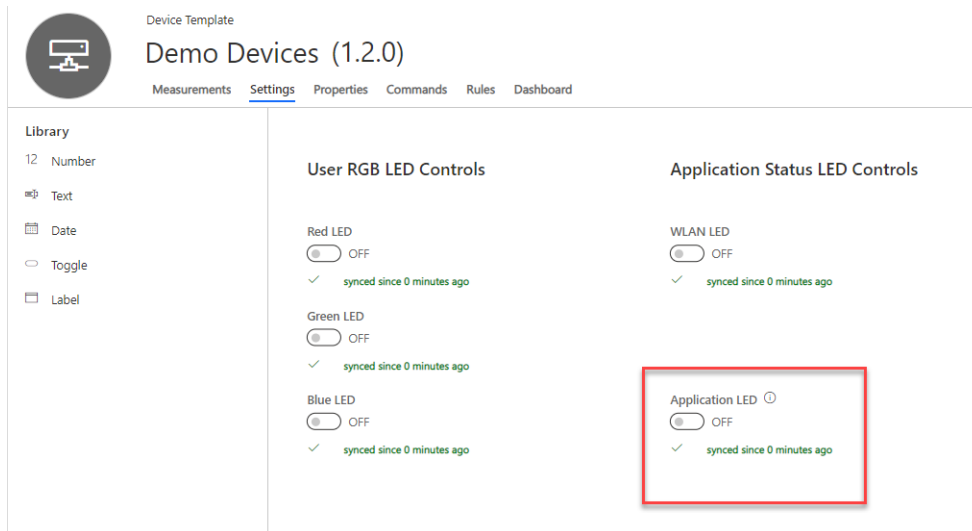
Current Version: 1.1.0

New Version: 1.2.0

Upgrade

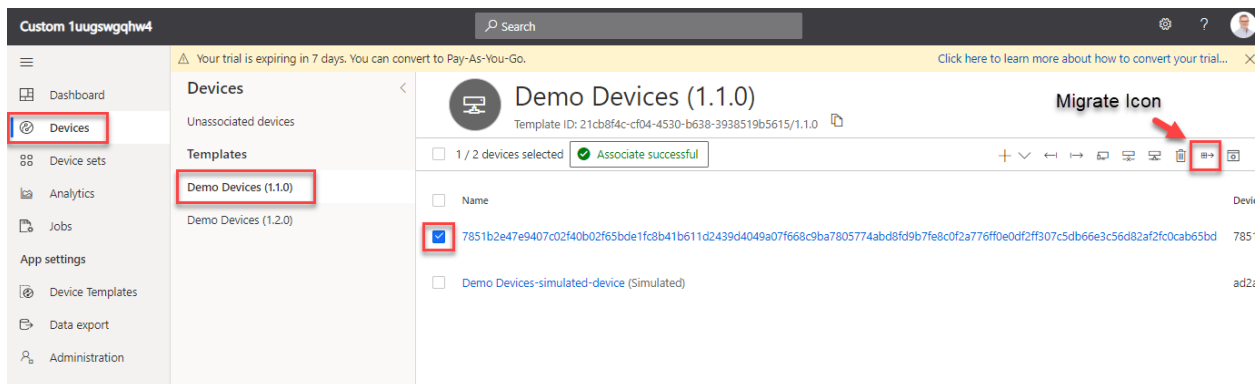
Cancel

The new control is shown on the new template



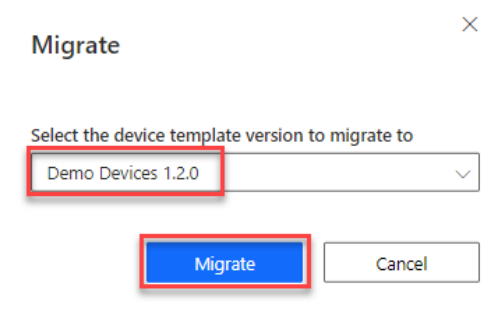
Next we need to move our device to the new template.

- From the left menu select Devices → Demo Devices (1.1.0)
- Check the box next to your device
- Select the migrate icon as shown below

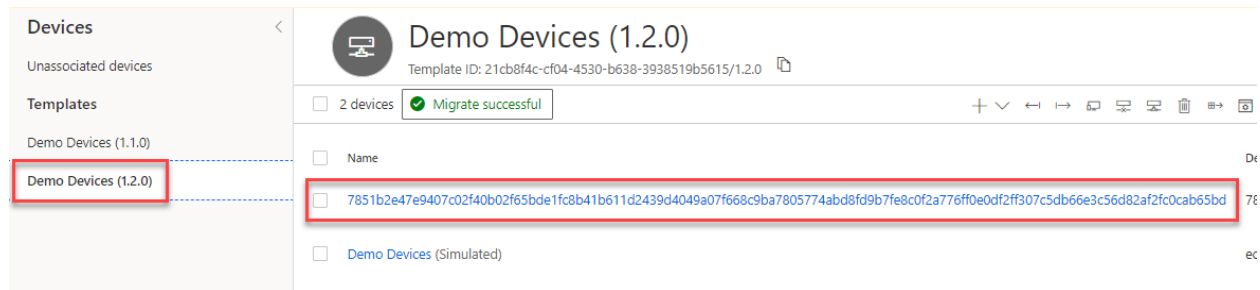


The “Migrate” dialog will open

- Select “Demo Devices 1.2.0 from the list and click on the “Migrate” Button



Your device will be moved from the “Demo Devices (1.1.0)” template and placed into the “Demo Devices (1.2.0)” template where the new setting control for the Application LED will be active.



We have modified our IoT Central Application but the device twin item we just referenced is not implemented in the Azure Sphere application, let's fix that.

Complete a code assignment

The assignment is to add a new Device Twin item to control the Application LED on the Avnet Starter Kit.

In Lab-3/Lecture-4 we reviewed the example application Device Twin implementation. Use that information and the details below to add a new device twin item that can be used to control the App LED on the board. Remember that the Lecture content for Device Twins was different than the Lab content on Device Twins. You can use the Lab content to understand what all has to happen when we process device twins. The lecture content discusses the actual implementation in the example application.

Key: “appLed”

GPIO: `AVNET_MT3620_SK_APP_STATUS_LED_YELLOW`

Active_high: false

Hint: Look at how one of the other LED device twin items is implemented.

What does success look like?

When the new device twin item is implemented and the Azure Sphere application is running, you should be able to go to your device in the IoT Central Application, toggle the “Application LED” control and see the App LED on the Avnet Starter Kit toggle!

Bonus Assignment

As a bonus assignment add the pressure telemetry (from the Lab-4 coding assignment) to the Measurements Tab on your IoT Central Application.

Useful Links

- [Microsoft Azure Sphere Documentation](#)
- [Microsoft Azure Sphere GitHub Examples](#)
- [Azure Sphere Developer Resources](#)

Wrap Up

In this Lab we learned about IoT Central and how to implement changes in the application.

- How to create an IoT Central Application from a Template
- How to add a device to IoT Central
- How to add a new Settings item to IoT Central
- Reinforced making changes in the Azure Sphere Application to implement a new Device Twin item

Revision History

Date	Version	Revision
1 July 19	01	Preliminary release
22 Aug 19	02	Updated broken link for IOTC Template with SURL
24 Dec 19	03	Updated for new CMAKE build process Reworked IoT Central content since it was recently changed by MSFT
14 Jan 20	04	Added details to use the ShowIoTCentralConfig.exe utility