Azure Sphere

Getting Started

TP4.0.1 2-May-2018

Table of Contents

Getting Started with Azure Sphere

This document walks you through setting up your PC and development environment, and using a development board to develop applications. In addition, it demonstrates how to load and build Azure Sphere sample applications with Visual Studio.

Updating from an earlier release

If you have already installed an earlier release, you can skip some of the procedures in this document. You do not need to set up your machine or claim your device again. However, you must complete the following:

- Install Visual Studio 2017 version 15.3 or later
- Install the Visual Studio Tools Preview for Azure Sphere
- Update your device software to the current version
- Add your device to the Microsoft System Software device group
- Configure Wi-Fi to restore your Wi-Fi connections after updating the device software

Set up your machine for Azure Sphere development

To connect to a reference development board (RDB), your development machine requires the following:

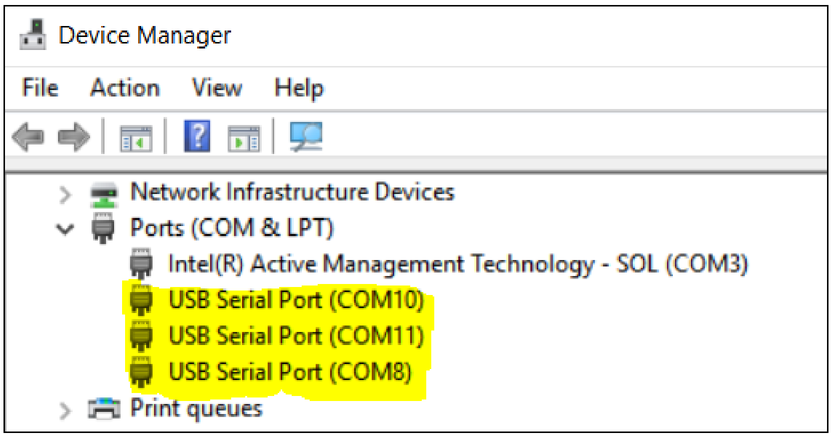
- Windows 10 Anniversary Update or later
- Support for the Visual Studio 2017 System Requirements
- A USB port

Connect the RDB

The RDB connects to a PC through a USB micro-connector. When plugged in, the RDB exposes three COM ports.

The first time you plug in the board, the drivers should be automatically downloaded and installed. Installation can be slow; if the drivers are not installed automatically, right-click on the device name in Device Manager and select **Update driver**. Alternatively, you can download the drivers from Future Technology Devices International (FTDI). Choose the driver that matches your Windows installation (32- or 64-bit).

To verify installation, open **Device Manager** and look for three COM ports:



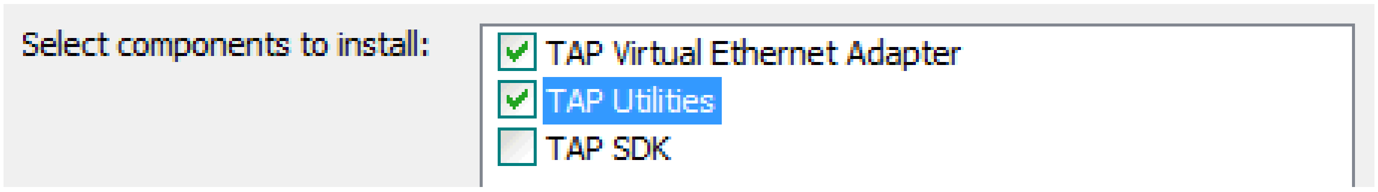
Network Infrastructure Devices
Ports (COM & LPT)
Intel(R) Active Management Technology - SOL (COM3)
USB Serial Port (COM10)
USB Serial Port (COM11)
USB Serial Port (COM8)
Print queues

Install the TAP driver

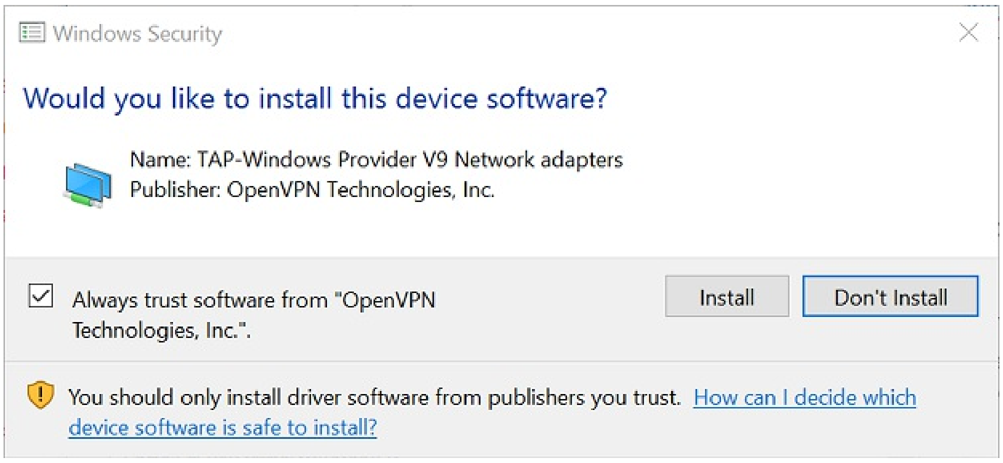
The development board communicates with the PC over serial line internet protocol (SLIP). Tap-Windows provides a network interface driver for SLIP.

To install TAP and enable SLIP communication

1. Install TAP-Windows, which came with your Azure Sphere software development kit (SDK).
2. In the installation options, choose **TAP Virtual Ethernet Adapter** and **TAP Utilities**, but not TAP SDK.


Select components to install:
✓ TAP Virtual Ethernet Adapter
✓ TAP Utilities
☐ TAP SDK

3. If you are asked to authorize installation of the driver, select **Install**.


Windows Security

Would you like to install this device software?

Name: TAP-Windows Provider V9 Network adapters
Publisher: OpenVPN Technologies, Inc.

☑ Always trust software from "OpenVPN Technologies, Inc.".

You should only install driver software from publishers you trust. How can I decide which device software is safe to install?

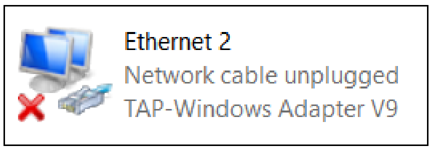
Configure TAP networking

After you install the driver, configure TAP networking for the board.

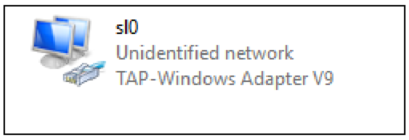
Note: You must have administrator rights to the PC to set the properties in steps 3 and 4.

To configure TAP networking

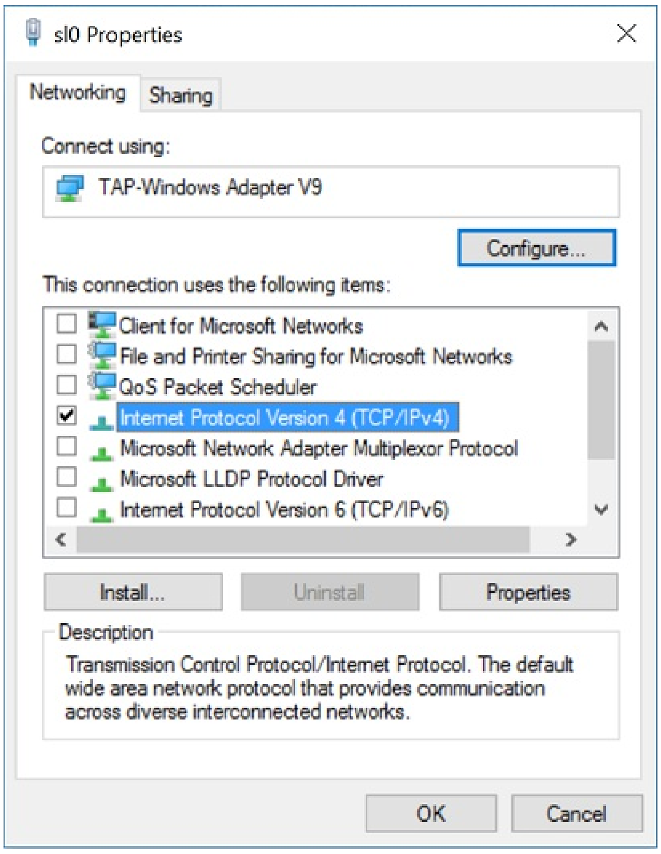
1. In Control Panel, type **View Network Connections** in the search box, and click to open the Network Connections dialog box. Find the TAP-Windows Adapter V9.


Ethernet 2
Network cable unplugged
TAP-Windows Adapter V9

2. Select **TAP-Windows Adapter v9** and rename it to **sl0** (lower case S, lower case L, the number zero):


sl0
Unidentified network
TAP-Windows Adapter V9

3. Open **Properties** for **sl0** and disable all services except Internet Protocol Version 4 (TCP/IPv4):


sl0 Properties

Networking | Sharing

Connect using:

TAP-Windows Adapter V9

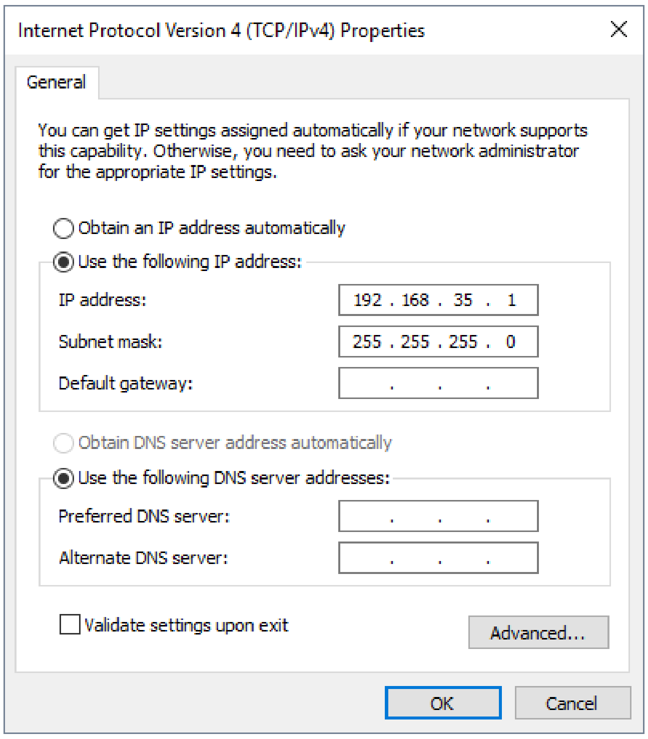
This connection uses the following items:

- [ ] Client for Microsoft Networks
- [ ] File and Printer Sharing for Microsoft Networks
- [ ] QoS Packet Scheduler
- [x] Internet Protocol Version 4 (TCP/IPv4)
- [ ] Microsoft Network Adapter Multiplexor Protocol
- [ ] Microsoft LLDP Protocol Driver
- [ ] Internet Protocol Version 6 (TCP/IPv6)

Description

Transmission Control Protocol/Internet Protocol. The default wide area network protocol that provides communication across diverse interconnected networks.

4. Select **Properties** for TCP/IPv4 and configure it to use the IP address 192.168.35.1, subnet mask 255.255.255.0:


Internet Protocol Version 4 (TCP/IPv4) Properties

General

You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.

○ Obtain an IP address automatically
● Use the following IP address:

IP address:	192 . 168 . 35 . 1
Subnet mask:	255 . 255 . 255 . 0
Default gateway:	. . .

○ Obtain DNS server address automatically
● Use the following DNS server addresses:

Preferred DNS server:	. . .
Alternate DNS server:	. . .

- [ ] Validate settings upon exit

5. Click **OK**.

Install Visual Studio 2017 version 15.3 or later

The Visual Studio Tools for Azure Sphere require Visual Studio Enterprise, Professional, or Community 2017 version 15.3 or later. To verify which version is installed, start the Visual Studio Installer and make sure that the version number is 15.3.0 or later. If the installer prompts you to update the Visual Studio Installer, do so.

If you have installed an earlier version of Visual Studio 2017 15.3.0 (preview 4 or later) from the Visual Studio Preview channel, you can continue to use that version. However, use of the Preview channel is no longer required; you can use the current version from the regular Visual Studio website.

To install Visual Studio, click Download Visual Studio, select the edition to install, and then run the installer. You can choose to install any workloads, or none. The Visual Studio Tools for Azure Sphere installation procedure automatically installs the workloads that the SDK requires.

Install the Visual Studio Tools Preview for Azure Sphere

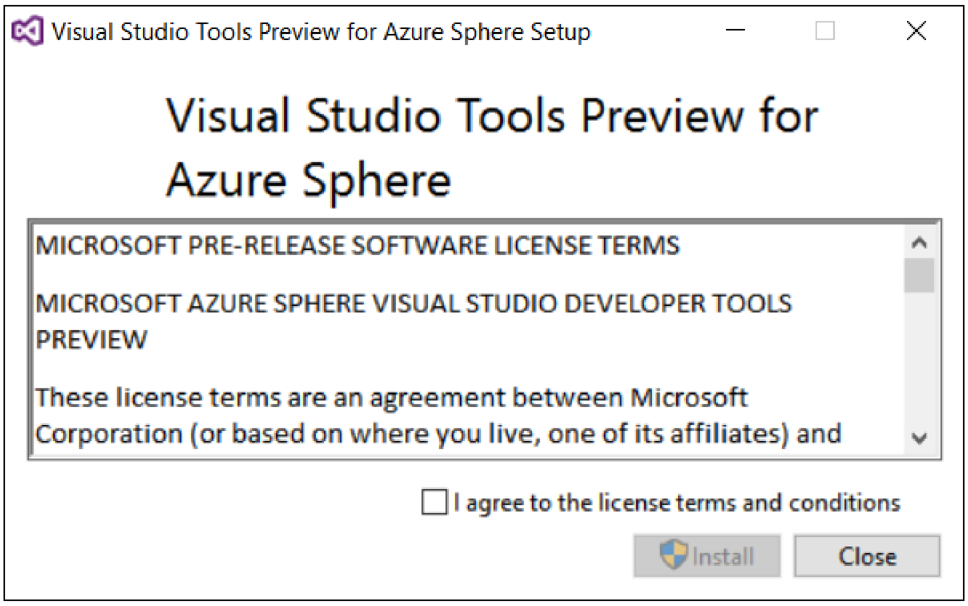
The Visual Studio Tools Preview for Azure Sphere includes:

- A custom Azure Sphere Developer Command Prompt, which is available in the **Start** menu under Azure Sphere
- The GDB debugger for use with the Azure Sphere development board
- Device, cloud, and image utilities
- Libraries for application development
- Visual Studio extensions to support Azure Sphere development

Azure_Sphere_VS_Dev_Tools_Preview.exe installs the complete Azure Sphere software development kit (SDK).

To install the Visual Studio Tools Preview for Azure Sphere

1. Run VS_Tools_Preview_for_Azure_Sphere.exe, which came with your Azure Sphere development kit, to install the developer tools. Agree to the license terms and select **Install**. Accept the elevation prompt if you see one.

   Visual Studio Tools Preview for Azure Sphere Setup

   Visual Studio Tools Preview for Azure Sphere

   MICROSOFT PRE-RELEASE SOFTWARE LICENSE TERMS

   MICROSOFT AZURE SPHERE VISUAL STUDIO DEVELOPER TOOLS PREVIEW

   These license terms are an agreement between Microsoft Corporation (or based on where you live, one of its affiliates) and

   If you have just installed Visual Studio for the first time, you might see the message, "No product to install SDK on." If this occurs, restart your PC and return to this step.

2. In the VSIX Installer window, confirm the Visual Studio version(s) for which to install the tools.

3. Accept the elevation prompt.

4. After installation starts, find the VSIX Installer window and bring it to the front. The installation process displays two installation windows: the Visual Studio Tools Preview for Azure Sphere window and the VSIX Installer window. The former reports progress and errors from the overall installation, and the latter reports information about the Visual Studio extension only. If the VSIX window becomes obscured during installation, you might not see error reports or requests for action.

5. When setup completes, select **Close** in both the VSIX Installer window and the Visual Studio Tools Preview for Azure Sphere setup window.

If the installer returns errors, try uninstalling and then reinstalling the tools. To uninstall the tools, use **Add and Remove Programs** in **Control Panel**.

Update your device software

This release includes updated Azure Sphere device software. If your device does not have the latest software installed, you must update it before proceeding with application development.

Follow the instructions in DeviceUpdate*Release*.pdf to check the version of your device software and then to update your device software to the current version and to assign it to the correct device group.

Important: After update, you must assign your device to the Microsoft device group that delivers only the most recent Azure Sphere OS before you reconfigure its Wi-Fi credentials.

Claim your device

After you install Visual Studio and the SDK, you must *claim* your device. Claim your device only once.

Every device has a unique and immutable Azure Sphere device ID that the Azure Sphere security service uses to identify and authenticate it. Before you update the device software or develop applications for the device, you must claim the device. Claiming the device associates it with your Azure Sphere tenant. See HowToCreateATenant.pdf for information about creating a tenant.

Before you claim a device, be sure that you are logged in to the tenant that you plan to use with Azure Sphere services. See the **cutil login** command in the *Command-line Utilities Reference* for more information.

To claim your device

1. Connect your board to the PC by USB.

2. Open an Azure Sphere Developer Command Prompt. To find the Azure Sphere Developer Command Prompt, click the **Start** button and type **Azure Sphere**.

3. Run the **cutil** cloud utility with the **device** command as follows:

   ```
   cutil device claim --attached
   ```

   This command reads the Azure Sphere device ID from the board and associates it with your current tenant. If you are prompted to log in to Microsoft Azure, do so using your Azure Sphere credentials.

   You should see output like this:

   ```
   Claiming device.
   Claiming attached device ID
   '94B27082513B529C45098884F882B2CA6D832587CAAE1A90B1CEC4A376EA2F22A96C4E7E1FC4D2AFF5633B68DB68FF4420A5588B
   420851EE4F3F1A7DC51399ED' into tenant ID 'd343c263-4aa3-4558-adbb-d3fc34631800'.
   Successfully claimed device ID
   '94B27082513B529C45098884F882B2CA6D832587CAAE1A90B1CEC4A376EA2F22A96C4E7E1FC4D2AFF5633B68DB68FF4420A5588B
   420851EE4F3F1A7DC51399ED' into tenant ID 'd343c263-4aa3-4558-adbb-d3fc34631800'.
   Command completed successfully in 00:00:05.5459143.
   ```

Troubleshooting:

If **cutil** returns the following message, your device might not be correctly installed or configured:

```
An error occurred. Please check your device is connected and your PC has been configured correctly, then retry.
```

This message can appear in several circumstances:

- The board is not connected by USB to the PC.
- The TAP driver is not installed.
- The IP address is not set correctly.
- The Azure Sphere Device Communication Service has not yet started.

Try the following solutions, in order:

1. Ensure that the device is connected by USB to the PC.
2. If the device is connected, press the Reset button on the device. Wait ten seconds or so for the device to restart, and then issue the failed command again.
3. If the command again reports that it cannot find the device, unplug the device from the USB connector, plug it in again, and wait for it to restart.
4. If the error recurs after restart, use **View Network Connections** in **Control Panel** to check that the sl0 device exists and is configured to use IP address 192.168.35.1.

Add your device to the Microsoft System Software device group

Before you configure Wi-Fi on your device, you must add it to the Microsoft-created System Software device group. The group is named System Software and has the following device group ID:

```
63bbe6ea-14be-4d1a-a6e7-03591d882b42
```

Use the following command to add your device to the group:

```
cutil device setdg --attached --devicegroupid 63bbe6ea-14be-4d1a-a6e7-03591d882b42
Successfully moved device
'94B27082513B529C45098884F882B2CA6D832587CAAE1A90B1CEC4A376EA2F22A96C4E7E1FC4D2AFF5633B68DB68FF4420A5588B42085
1EE4F3F1A7DC51399ED' to device group '63bbe6ea-14be-4d1a-a6e7-03591d882b42' in the cloud.
Command completed successfully in 00:00:03.4984740.
```

For more information about device groups, see the *Deployment Guide*.

Configure Wi-Fi

You must configure the device for Wi-Fi before it can communicate with the Azure IoT Hub or receive over-the-air (OTA) updates.

To set up Wi-Fi on your device

1. If your device is not connected to your PC, connect it now.

2. Open an Azure Sphere Developer Command Prompt and issue a command in the following form:

   ```
   dutil wifi add -s <yourSSID> -k <YourNetworkKey>
   ```

   The -s flag specifies the network SSID, and the -k flag specifies the WPA2 key. Network SSIDs are case-sensitive and can include only ASCII characters. To add an open Wi-Fi access point, omit the -k flag.

   ```
   dutil wifi add -s MYOPENSSID
   ```

   You should see:

   ```
   Add network succeeded:
   ID : 0
   SSID : MYOPENSSID
   Configuration state : enabled
   Connection state : unknown
   Security state : open
   ```

   If your network SSID or key has embedded spaces, enclose the SSID or key in quotation marks. If the SSID or key includes a quotation mark, use a backslash to escape the quotation mark. Backslashes do not require escape if they are part of a value. For example:

   ```
   dutil wifi add -s "New SSID" -k "key \"value\" with quotes"
   ```

   It generally takes 45-60 seconds for networking to be ready on the board.

3. Use the **dutil wifi status** command to check the status of the connection. The following example shows successful results from a secure WPA-2 connection:

   ```
   dutil wifi status

   SSID : NETGEAR21
   Configuration state : enabled
   Connection state : connected
   Security state : psk
   Frequency : 2442
   Mode : station
   Key management : WPA2-PSK
   WPA State : COMPLETED
   IP Address : 192.168.1.15
   MAC Address : 52:cf:ff:3a:76:1b
   ```

The **dutil wifi** command supports several additional options. Type **dutil wifi** for a complete list, or **dutil wifi** *option* **-h** for help on an individual option.

Wi-Fi troubleshooting

If you encounter Wi-Fi problems, first ensure that your Wi-Fi network uses 802.11b/g/n; Azure Sphere devices do not support 802.11a. In addition, use only ASCII characters in the network SSID.

If both the wireless protocol and the network SSID are valid, please try the following steps from a PC connected to the router:

1. Open a command prompt window, issue the following command, and save the output in a text file:

   ```
   netsh wlan show networks mode=bssid
   ```

2. Open your router settings and change the cipher type. Note that access to this setting varies by router model and manufacturer. Some manufacturers call it “cipher type” and others call it “encryption.” Here are some guidelines:

   - If your network type is WPA2 PSK, you might need to use the Windows settings. Right-click on the Wi-Fi icon, select Open Network and Sharing Center, and then click on the wireless network connection. In the Wi-Fi Status dialog box, click Properties and then open the Security tab.
   - Find the current encryption type or cipher setting. Options are typically TKIP, AES, or AUTO—most likely AUTO.
   - Choose TKIP or AES and try the Wi-Fi connection again.
   - If changing the setting seems to solve the problem, verify the solution by reverting to the previous setting. Wi-Fi should now fail again.

3. If possible, get a Wireshark trace of a connection failure.

Collect the output from step 1 and the trace from step 3 and send them to us together with the manufacturer and model of the router—and, if possible, the firmware version—so that we can try to reproduce the problem.

Run the sample applications

After you set up the device, Visual Studio, and Wi-Fi connection as described in the previous sections, you can build an Azure Sphere application.

The Azure Sphere SDK includes the following samples and templates:

- Blink Sample shows how to access GPIOs and LEDs.
- UART Sample shows how to access the universal asynchronous receiver/transmitter (UART).
- Azure IoT Hub Sample communicates with an Azure IoT Hub to send and receive messages, to update a device twin, and to handle direct method calls.
- Blank Application is a blank Azure Sphere Application project for use as a template.

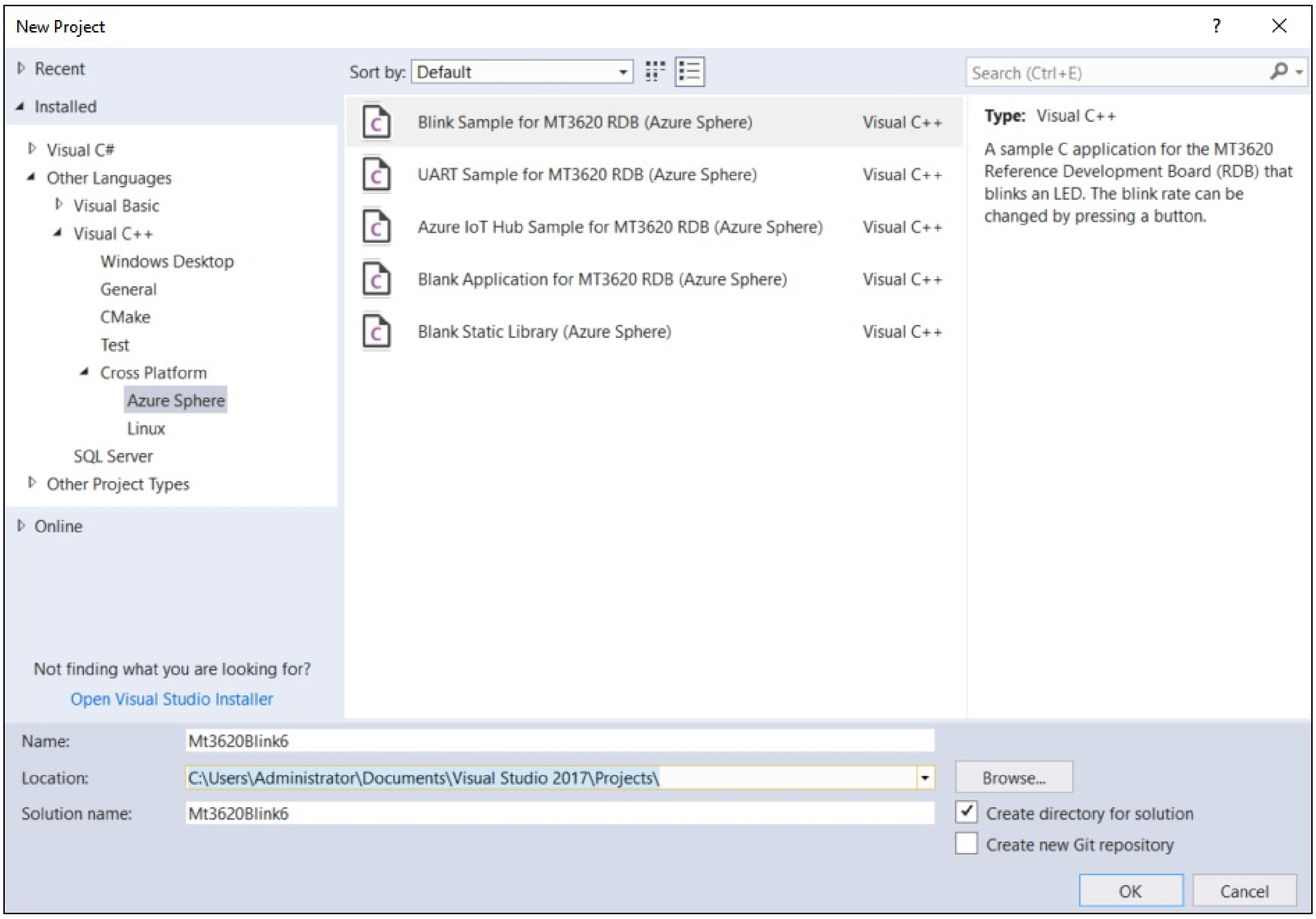
This section walks you through using the Blink sample to create your first Azure Sphere application, set a breakpoint, and sideload the application to the development board. Next, it goes through the UART sample. Finally, it shows how an Azure Sphere application can use an Azure IoT Hub.

Blink sample application

The Blink sample application shows how to access GPIOs and LEDs on the development board.

To run the Blink sample

1. Start Visual Studio 2017 and go to **File>New>Project**. The templates for Azure Sphere are available in **Visual C++>Cross Platform>Azure Sphere**. Select **Blink Sample for MT3620 RDB (Azure Sphere)**.



Blink Sample for MT3620 RDB (Azure Sphere)	Visual C++
UART Sample for MT3620 RDB (Azure Sphere)	Visual C++
Azure IoT Hub Sample for MT3620 RDB (Azure Sphere)	Visual C++
Blank Application for MT3620 RDB (Azure Sphere)	Visual C++
Blank Static Library (Azure Sphere)	Visual C++

Type: Visual C++

A sample C application for the MT3620 Reference Development Board (RDB) that blinks an LED. The blink rate can be changed by pressing a button.

Name: Mt3620Blink6

Location: C:\Users\Administrator\Documents\Visual Studio 2017\Projects\

Solution name: Mt3620Blink6

✓ Create directory for solution

☐ Create new Git repository

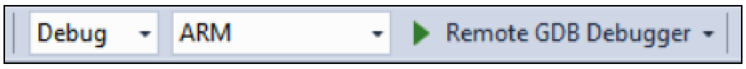
2. Enter a name and location for the project and select **OK**. The project opens with main.c in the editor. The project is

configured to use the Windows cross compilers and to automatically select the COM port to which to deploy the binary.

3. Navigate to the line that tests the value of newGpioButtonState and press **F9** to set a breakpoint:

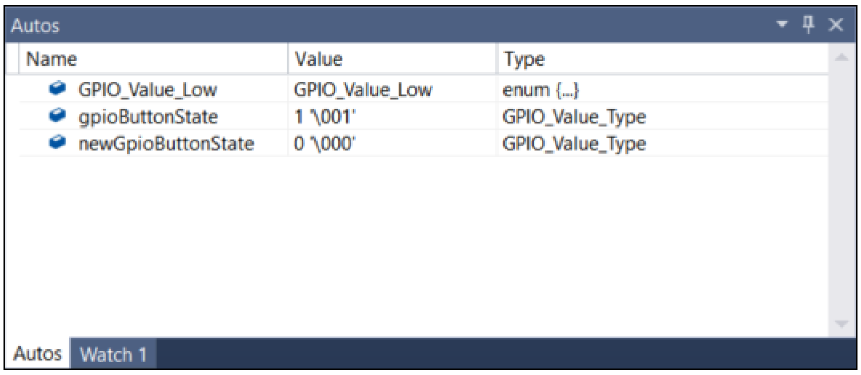
   ```
   if (newGpioButtonState == GPIO_Value_Low) {
   ```

4. Ensure that the board is connected to your PC by USB. Then select **Remote GDB Debugger** from the menu bar or press **F5**.

5. If you are prompted to build the project, select **Yes**. Visual Studio compiles the application, sideloads it to the board, and starts it in debug mode.

6. Press button A. Visual Studio stops at the breakpoint that you set. Open **Debug>Windows** and select Autos to display the variables that are used in the current and previous statements. Visual Studio populates the call stack and the auto variables after program execution stops at the breakpoint. In the following example, the value of the newGpioButtonState variable is 0, equal to GPIO_Value_Low, which indicates a button press.

Name	Value	Type
GPIO_Value_Low	GPIO_Value_Low	enum {...}
gpioButtonState	1 '\001'	GPIO_Value_Type
newGpioButtonState	0 '\000'	GPIO_Value_Type

7. By default, the Output window shows output from **Device Output**. To see messages from the debugger, change the contents of the Output window to **Debug**. You can also inspect the program disassembly, registers, or memory through the **Debug>Windows** menu.

8. Select **Continue**. Execution pauses again at this breakpoint. Now the value of newButtonState variable is 1, equal to GPIO_Value_High, which indicates a button release. Each time you press and release Button A, the blink rate changes.

9. If you find stepping through the application slow, set additional breakpoints in the code and select **Continue** to resume execution to the next line in the source code you want to inspect.

10. When you are done debugging, select the **Stop** icon in the menu bar or press **Shift+F5**. You can also optionally clear any breakpoints you previously set.

UART sample application

The UART sample demonstrates the use of communication over UART. A simple way to test UART is to loop back a UART on the board. On header 2 (marked H2) on the lower left side of the board:

- Connect pins 1 and 3 (ISU0 RDX and ISU0 TDX) of H2 with a jumper header.

To run the UART Sample

1. Start Visual Studio 2017 and go to **File>New>Project**. The templates for Azure Sphere are available in **Other Languages>Visual C++>Cross Platform>Azure Sphere**. Select **UART Sample for MT3620 RDB (Azure Sphere)**, specify a name and location, and select **OK**.

2. Ensure that you have installed the jumper header in the correct location for your device. Connect the device to your PC by USB, then press **F5** or select **Remote GDB Debugger** on the menu bar. If you are prompted to build the project, select **Yes**.

3. Press button A on the board. This sends 13 bytes over the UART connection and displays the sent and received text in the Visual Studio Device Output window:

   ```
   Sent 13 bytes over UART in 1 calls
   UART received 12 bytes: 'Hello world!'
   UART received 1 bytes: '
   '
   ```

   All the received text might not appear at once, and it might not appear immediately.

   LED 2 on the development board toggles on and off each time you press the button.

Azure IoT Hub sample application

The Azure IoT Hub sample application shows how to connect to and communicate with an Azure IoT Hub. It also demonstrates several features of the Azure Sphere WifiConfig API.

The Azure IoT Hub sample uses the Connected Service for Azure Sphere, which is installed with the Azure Sphere SDK. This Visual Studio extension provides a template for connecting your device to an IoT Hub and communicating with the hub. It is similar to the Visual Studio Connected Service for Azure IoT Hub but provides Azure Sphere-specific features. Applications that use the Connected Service functionality can send messages to and receive messages from an IoT Hub, maintain a device twin, and respond to direct method calls from cloud service applications.

Set up Microsoft Azure credentials

To run this sample, you must have a Microsoft Azure subscription and an IoT Hub. If your organization does not already have them, follow these instructions to set up a free trial subscription to Microsoft Azure.

After you set up the subscription, you can create a hub. Log into the Azure Portal and follow these instructions to set up your hub.

Add your device to the IoT Hub

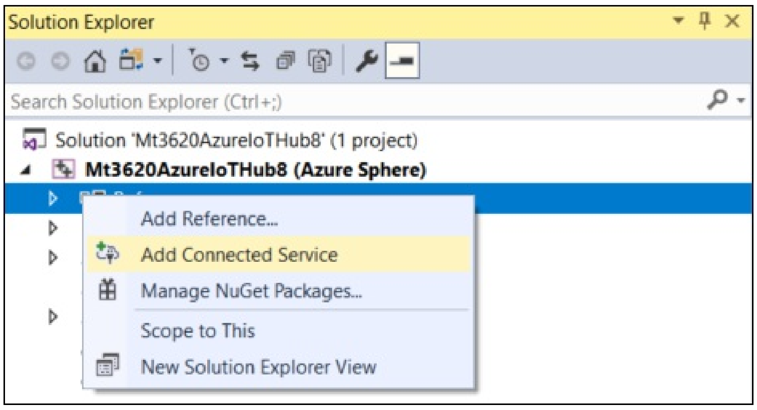
To use an IoT Hub in an Azure Sphere application, you identify the IoT Hub that you plan to use and then add your device to that hub. The connected service retrieves the IoT Hub connection string and records it in a file named azure_iot_hub.c. The connected service then adds azure_iot_hub.c and the header file azure_iot_hub.h to your application.

Important

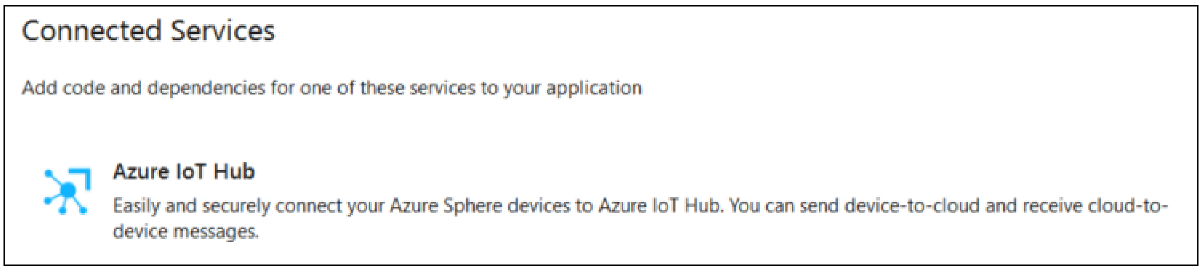
Connect your Azure Sphere device to your PC and to a Wi-Fi network before you run this sample.

1. Start Visual Studio 2017 and go to **File>New>Project**. The templates for Azure Sphere are available in **Visual C++>Cross Platform>Azure Sphere**. Select **Azure IoT Hub Sample for MT3620 RDB (Azure Sphere)**, specify a name and location, and select **OK**.

2. In Solution Explorer, right-click **References** and then select **Add Connected Service**.


Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'Mt3620AzureIoTHub8' (1 project)
Mt3620AzureIoTHub8 (Azure Sphere)
Add Reference...
Add Connected Service
Manage NuGet Packages...
Scope to This
New Solution Explorer View

3. Select **Azure IoT Hub (Azure Sphere)** from the list of connected services.

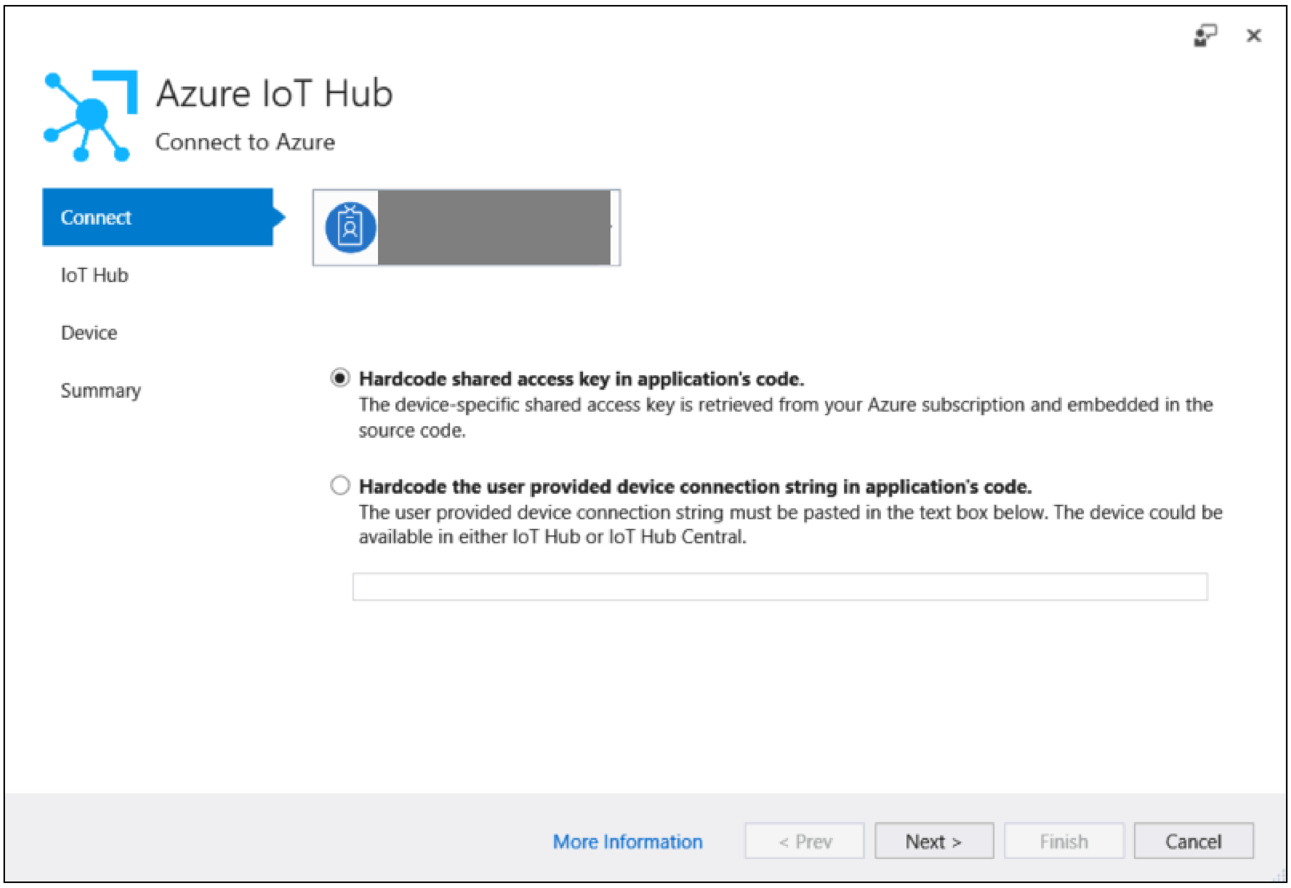

Connected Services

Add code and dependencies for one of these services to your application

Azure IoT Hub

Easily and securely connect your Azure Sphere devices to Azure IoT Hub. You can send device-to-cloud and receive cloud-to-device messages.

4. Log into Microsoft Azure. Select **Hardcode shared access key in application's code**, and then click **Next**.


Azure IoT Hub

Connect to Azure

[redacted]

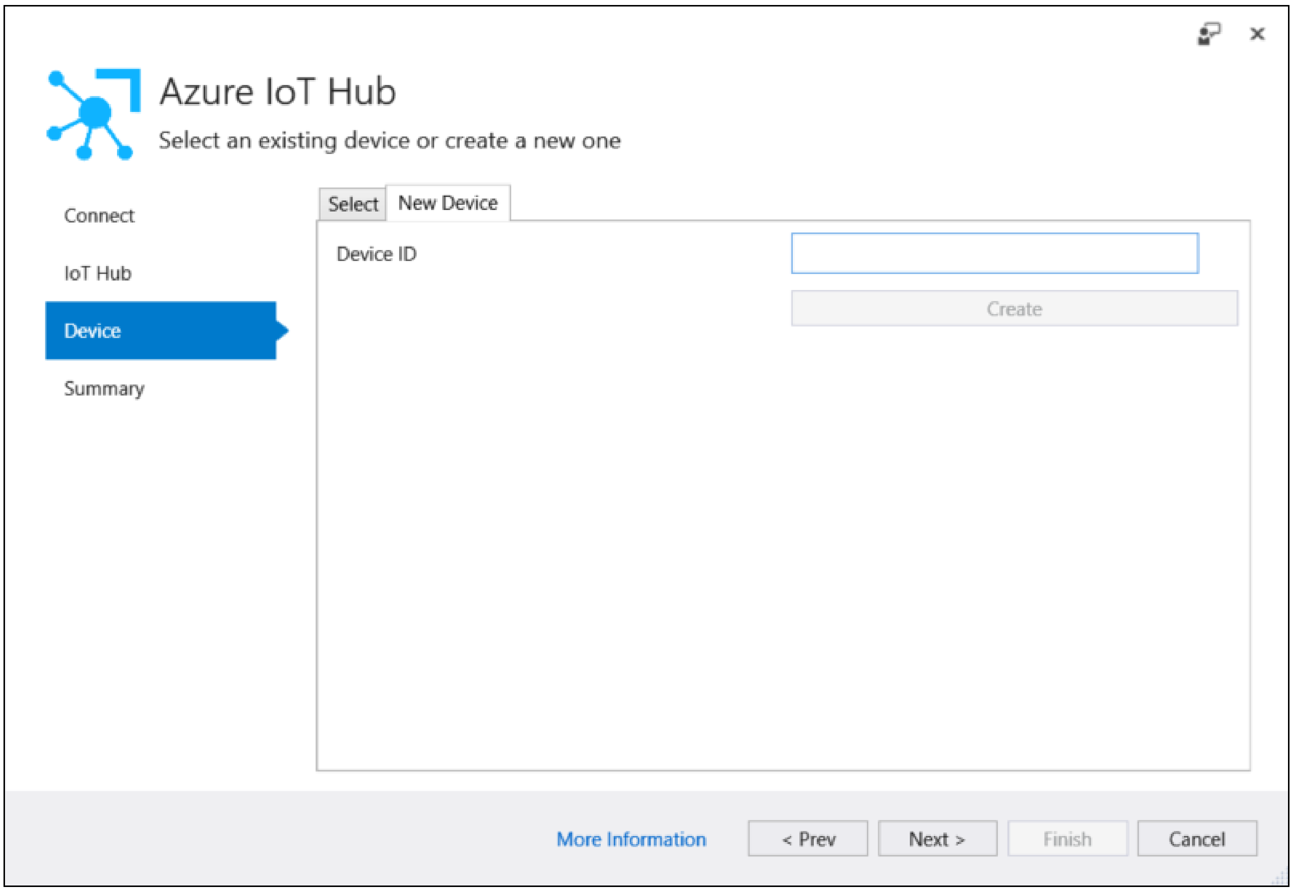
(●) **Hardcode shared access key in application's code.**
The device-specific shared access key is retrieved from your Azure subscription and embedded in the source code.

( ) **Hardcode the user provided device connection string in application's code.**
The user provided device connection string must be pasted in the text box below. The device could be available in either IoT Hub or IoT Hub Central.

5. In the list of hubs, select your hub and click **Next.**

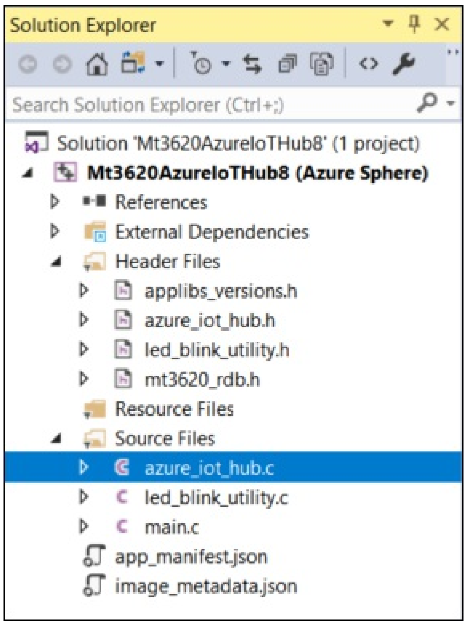
6. In the list of devices, select your device and click **OK**.

   If your device does not appear in the list, open the **New Device** tab. In the **Device ID** box, type a string that you will use to identify your board in the IoT Hub, and click **Create**. This identification string is used only by the IoT Hub and is not related to the Azure Sphere device ID. We suggest a human-readable string such as LabTest1, but if you prefer, you can use the Azure Sphere device ID with which you claimed the board.



7. From the **Summary** screen, copy the IoT Hub Connection String to temporary storage, such as Notepad; you will need it later in this walkthrough to use Device Explorer with your hub. Then click **Finish**.

8. In Solution Explorer, you should now see azure_iot_hub.h and azure_iot_hub.c in your solution.



In addition, the host name for the Azure IoT Hub has been added to the **Capabilities** section of the app_manifest.json file. An application can connect only to the internet hosts that are specified in the **AllowedConnections** field.

Prepare the sample

Now you can build the application and use the Azure IoT Hub. In this walkthrough, we will use existing IoT Hub tools to monitor and communicate with your device.

To prepare the sample code

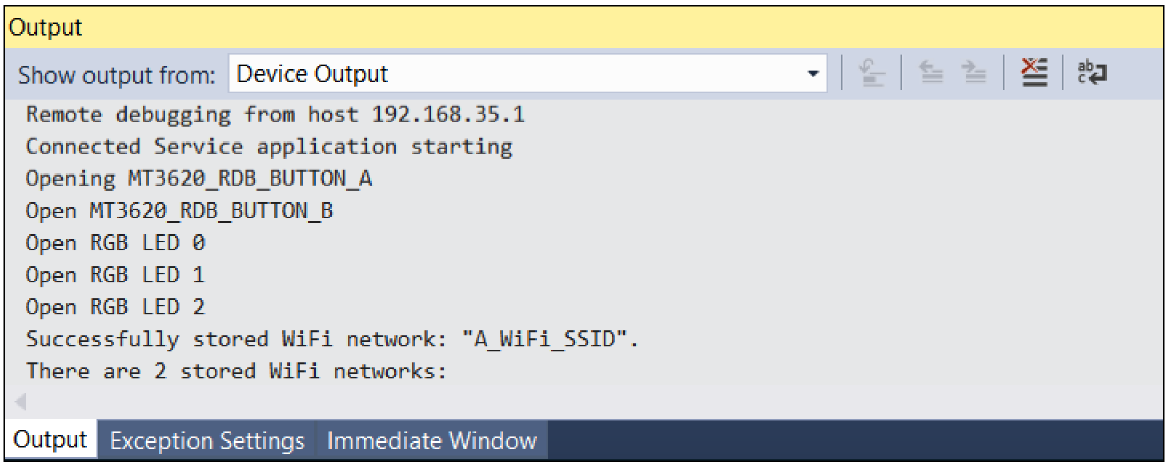
1. Open main.c in the sample application.

2. Remove the `#error` directive:

3. Press F5 to build, load, and start the sample.

   The sample opens handles to the buttons and RGB LEDs on the board and stores a test Wi-Fi network on the device. It scans for available Wi-Fi networks, and displays information about the stored Wi-Fi networks, the currently connected Wi-Fi network, and the available networks.

   It then starts the main loop. You should see LED 1 start to blink and see output like the following in the Device Output window:



```
Remote debugging from host 192.168.35.1
Connected Service application starting
Opening MT3620_RDB_BUTTON_A
Open MT3620_RDB_BUTTON_B
Open RGB LED 0
Open RGB LED 1
Open RGB LED 2
Successfully stored WiFi network: "A_WiFi_SSID".
There are 2 stored WiFi networks:
```

This sample does the following:

- Stores a sample Wi-Fi network on the device, displays information about the stored Wi-Fi networks, reports the currently connected Wi-Fi network, scans for available Wi-Fi networks, and displays a list of all the Wi-Fi networks found.

- Blinks LED 1 constantly. Pressing button A changes the rate between three values. The rate is also stored in the device twin, where a cloud service program can change it.

- Sends a message to the IoT Hub when you press button B.

- Lights LED 3 green after start-up to indicate that the device has connected to the IoT Hub and the application has successfully authenticated with the hub.

- Changes the color of LED 1 in response to a direct method call.

The following sections describe how to see device-to-cloud messages, send cloud-to-device messages, request changes to the device twin, and make direct method calls. This walkthrough uses Device Explorer, a utility that is the easiest way to interact with your device from an IoT Hub. Azure Portal and IoT Hub Explorer provide many of the same capabilities.

To install Device Explorer

- On the Azure IoT Hub SDK for C# Releases repository, download and install SetupDeviceExplorer.msi. If you installed an earlier version of Device Explorer, uninstall it through **Add and Remove Programs** on **Control Panel** before installing this version.

N o t e

This tool is called Device Explorer Twin in its window banner.

Send and receive messages

Device Explorer is an interactive application that lets you see messages from your device to the cloud and send messages from the cloud to your device.

To send and receive messages

1. Run Device Explorer. You can find it on the **Start** menu under **Azure IoT Hub**.
2. On the **Configuration** tab, paste the IoT Hub Connection String that you recorded in Add your device to the IoT Hub in the box and click **Update**. If you forgot to copy the string, you can step through the Add Connected Service wizard again until you reach the **Summary** screen, copy the IoT Hub Connection String from the box, and then click **Cancel** to exit.
3. On the **Data** tab, select your device from the Device ID dropdown menu and press **Monitor**.
4. Press button B on your Azure Sphere device to send a message to the cloud. If Device Explorer displays an error box describing an endpoint epoch error, restart Device Explorer.
5. In Device Explorer, check the **Data** tab to see the message from your device. In some situations, a delay may occur before the messages appear, and you'll sometimes see a batch of device-to-cloud messages at once.
6. In Device Explorer, open the **Messages to Device** tab and select your device from the Device ID dropdown menu. Type a message in the Message box, add a timestamp if you want, and press **Send**.
7. In Visual Studio, your message should appear in the **Output** window for Device Output.

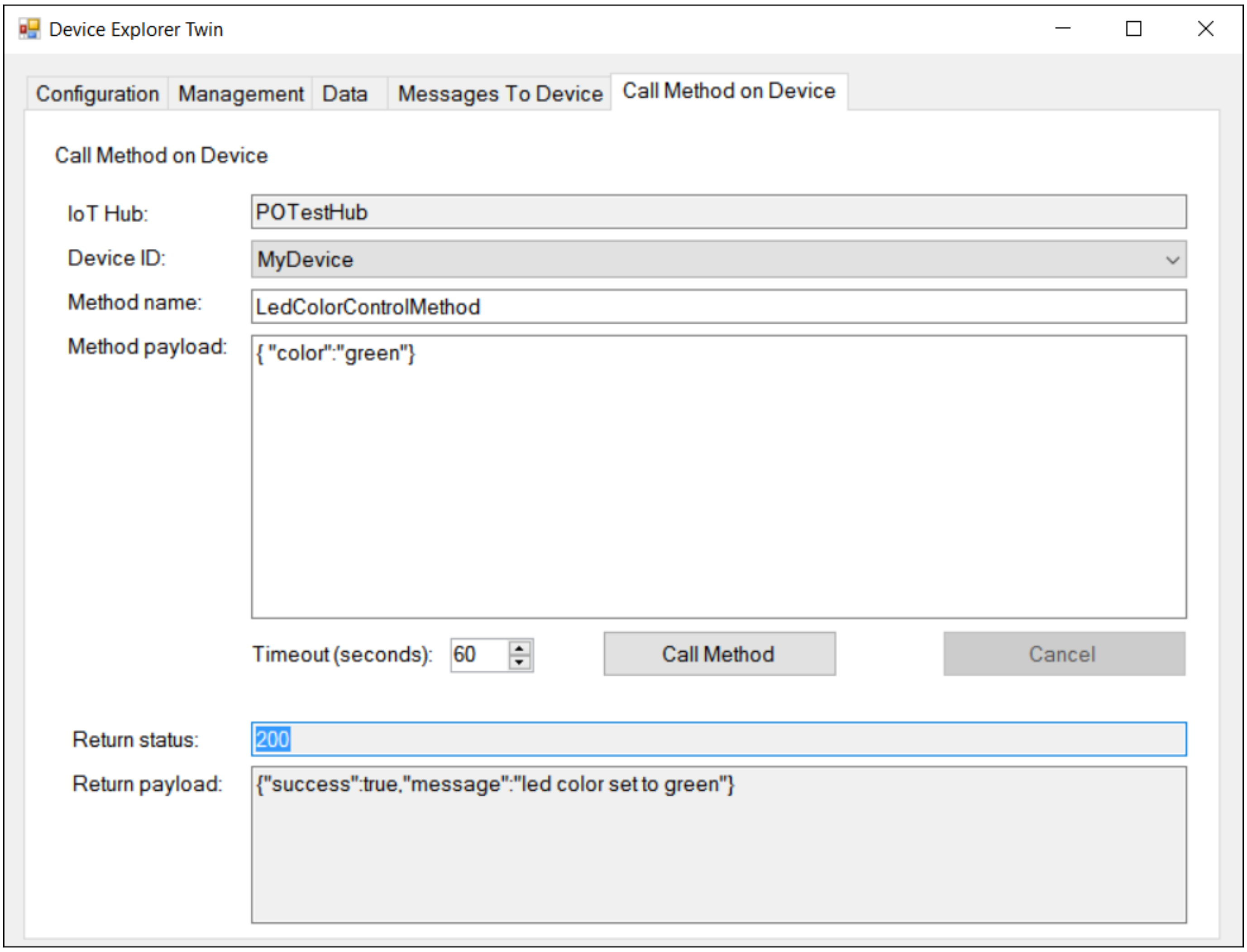
Call a direct method on the device

Applications that use an IoT Hub can respond to direct method calls from applications that run in the cloud. For example, a cloud-service application might collect data from the IoT Hub and then call the application on the device to reset a counter.

The example registers the DirectMethodCall() function to handle direct method calls. When a cloud service application calls a method on the device, this callback parses the method name and payload and responds appropriately.

To call a method on the device

1. Make sure that the application is running, and is not stopped at a breakpoint.
2. In Device Explorer, open the **Call Method on Device** tab. Ensure that the IoT Hub and Device ID are correct.
3. In the **Method name** box, type LedColorControlMethod, and in the Method payload box type {"color”:"green"}. Then press **Call Method.** The sample is hardcoded to check for calls to LedColorControlMethod. When you call LedColorControlMethod with this payload, LED 1 should start blinking green.



Call Method on Device

IoT Hub: POTestHub

Device ID: MyDevice

Method name: LedColorControlMethod

Method payload: { "color":"green"}

Timeout (seconds): 60

Return status: 200

Return payload: {"success":true,"message":"led color set to green"}

4. In Visual Studio, check the Output window for Device Output. You should see:

```
[Azure IoT Hub client]:INFO: Trying to invoke method LedColorControlMethod
INFO: color set to: 'green'.
```

5. In Device Explorer, check the Return status and Return payload boxes. The Return status should be 200 and the return payload should be:

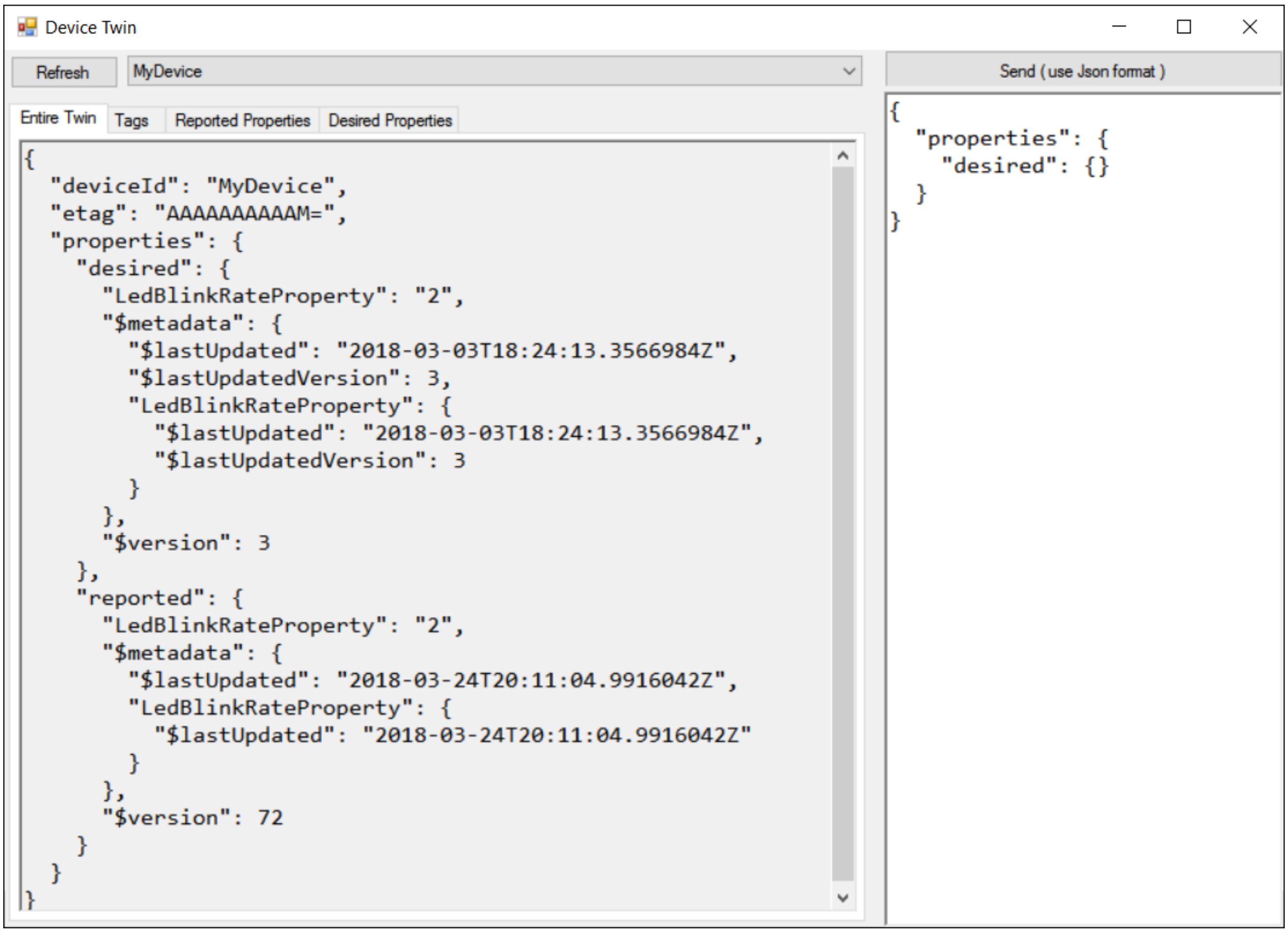
```
{"success":true,"message":"led color set to green”}
```

Manage a device twin

A device twin is a JSON document in the cloud that stores information about a device. Use the device twin to synchronize status information and maintain device properties that should be accessible to cloud service applications as well as to applications that run on the device.

To view and change the device twin

1. While the sample program runs, open the **Management** tab in Device Explorer and then click **Twin Props**. Select your device from the drop-down menu in the bar and click **Refresh**. You should see the **Entire Twin** tab open on the left and an editable window on the right. Note that the twin for your device may have different properties and metadata than the one in the figure.

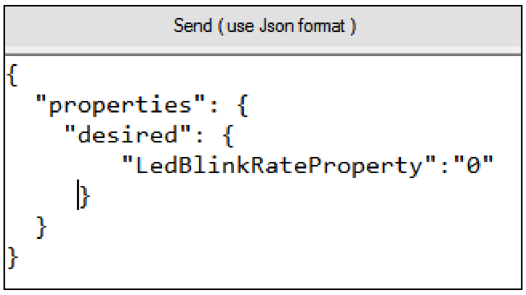


```
{
  "deviceId": "MyDevice",
  "etag": "AAAAAAAAAAM=",
  "properties": {
    "desired": {
      "LedBlinkRateProperty": "2",
      "$metadata": {
        "$lastUpdated": "2018-03-03T18:24:13.3566984Z",
        "$lastUpdatedVersion": 3,
        "LedBlinkRateProperty": {
          "$lastUpdated": "2018-03-03T18:24:13.3566984Z",
          "$lastUpdatedVersion": 3
        }
      },
      "$version": 3
    },
    "reported": {
      "LedBlinkRateProperty": "2",
      "$metadata": {
        "$lastUpdated": "2018-03-24T20:11:04.9916042Z",
        "LedBlinkRateProperty": {
          "$lastUpdated": "2018-03-24T20:11:04.9916042Z"
        }
      },
      "$version": 72
    }
  }
}
```

```
{
  "properties": {
    "desired": {}
  }
}
```

2. When you press button A on the board, the sample changes the blink rate and reports the new rate to the device twin as LedBlinkRateProperty in the "reported" section. To see the new rate in Device Explorer, click **Refresh**. Note that the "reported" version number in the twin also changes.

3. You can also change the blink rate by setting the value of LedBlinkRateProperty in the device twin. Edit the JSON in the window on the right and then press **Send (use JSON format)**. You can set the value to 0, 1, or 2 for different blink rates.



```
{
  "properties": {
    "desired": {
        "LedBlinkRateProperty":"0"
    }
  }
}
```

When you change the value of LedBlinkRateProperty, the sample updates the LED 1 blink rate and reports the new value to the twin in the "reported" section. You should see the LED blink at a different rate.

4. In Visual Studio, check the Output window for Device Output. You should see:

```
Property LedBlinkRateProperty changed, new value is "0"
[Azure IoT Hub client]:INFO: Reported state set
[Azure IoT Hub client]:INFO: Reported state accepted by IoT Hub. Result is: 204
```

Next steps

Now that you've set up your PC, installed the development tools, and run the sample applications, you can create applications of your own. See *Developing Applications* to find out how to create, package, and sideload your applications. For details about the APIs supported by the Azure Sphere SDK, see the *API Reference*.