



Azure RTOS IoT Embedded SDK Sample STM32L4+-DISCO IoT Node using IAR User Guide

Published: November 2020

For the latest information, please see
azure.com/rtos

This document is provided “as-is”. Information and views expressed in this document, including URL and other Internet Web site references, may change without notice.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

Azure RTOS provides OEMs with components to secure communication and to create code and data isolation using underlying MCU/MPU hardware protection mechanisms. It is ultimately the responsibility of the device builder to ensure the device fully meets the evolving security requirements associated with its specific use case.

© 2020 Microsoft. All rights reserved.

Microsoft Azure RTOS, Azure RTOS FileX, Azure RTOS GUIX, Azure RTOS GUIX Studio, Azure RTOS NetX, Azure RTOS NetX Duo, Azure RTOS ThreadX, Azure RTOS TraceX, Azure RTOS Trace, event-chaining, picokernel, and preemption-threshold are trademarks of the Microsoft group of companies. All other trademarks are property of their respective owners.

Revision 6.1

Table of Contents

<i>Overview</i>	<i>4</i>
<i>Prepare Azure Resources</i>	<i>6</i>
Create an IoT Hub	7
Register an IoT Hub device	8
<i>Prepare the device</i>	<i>9</i>
Add configuration	9
Build the project.....	11
Download and run the project.....	11
<i>View device properties</i>	<i>12</i>
Set IoT Hub	12
View device telemetry	13
Update device twin	14
Call a direct method on device	15
Send cloud-to-device message	16
Interact with IoT Plug and Play components	17
<i>Use Device Provisioning Service (DPS)</i>	<i>21</i>
Create a Device Provisioning Service (DPS)	21
Link an IoT Hub for DPS	22
Add enrollment in DPS	22
Add configuration	23
Build the project.....	24
Download and run the project.....	24
View device identity	24
<i>Security monitoring capabilities</i>	<i>26</i>
Before using it	26
Resource requirements	27
Limitation	27
Pricing.....	27
View alerts.....	28
<i>Clean up resources</i>	<i>31</i>
<i>Next steps</i>	<i>32</i>

Overview

The following steps detail how to configure, build and execute the Microsoft Azure IoT integration example using Azure IoT Embedded C SDK (public preview) on the STM32L4S5I Discovery kit for IoT Node, using IAR's EWARM 8.50 or later development tools. A 30-day free trial of IAR's EWARM can be downloaded from this page:

<https://www.iar.com/iar-embedded-workbench/#!?architecture=Arm>

It is also recommended to upgrade the ES-WiFi UART firmware to v3.5.2.5 or above:

<https://www.inventeksys.com/iwin/firmware/>

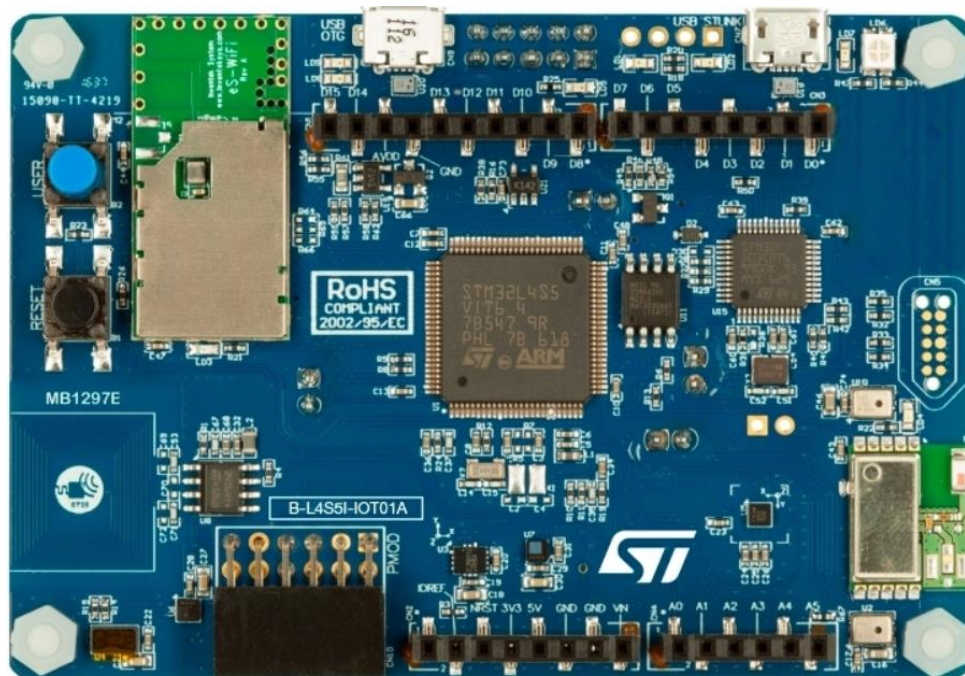


Figure 1 STM32L4S5I Discovery Kit for IoT Node

The sample distribution zip file contains the **azure_rtos.eww** workspace as well as following sub-folders:

Folder	Contents
<i>iar</i>	Contains the following sub-folders as well as the azure_rtos.eww workspace
<i>azure_iot</i>	Azure IoT Middleware for Azure RTOS source code
<i>common_hardware_code</i>	Common code for STM32FL4S5I-DISCO board
<i>docs</i>	User guides
<i>netxduo</i>	NetX Duo source code
<i>sample_azure_iot_embedded_sdk</i>	Sample project to connect to Azure IoT Hub using Azure IoT Middleware for Azure RTOS
<i>sample_azure_iot_embedded_sdk_pnp</i>	Sample project to connect to Azure IoT Hub using Azure IoT Middleware for Azure RTOS via IoT Plug and Play
<i>sample_pnp_temperature_controller</i>	Sample project with IoT Plug and Play using multiple components
<i>stm32l4xx_lib</i>	STM32L4+ drivers
<i>threadx</i>	ThreadX source code

Prepare Azure Resources

To prepare Azure cloud resources and connect a device to Azure, you can use Azure CLI. There are two ways to access the Azure CLI: by using the Azure Cloud Shell, or by installing Azure CLI locally. Azure Cloud Shell lets you run the CLI in a browser, so you don't have to install anything.

Use one of the following options to run Azure CLI.

If you prefer to run Azure CLI locally:

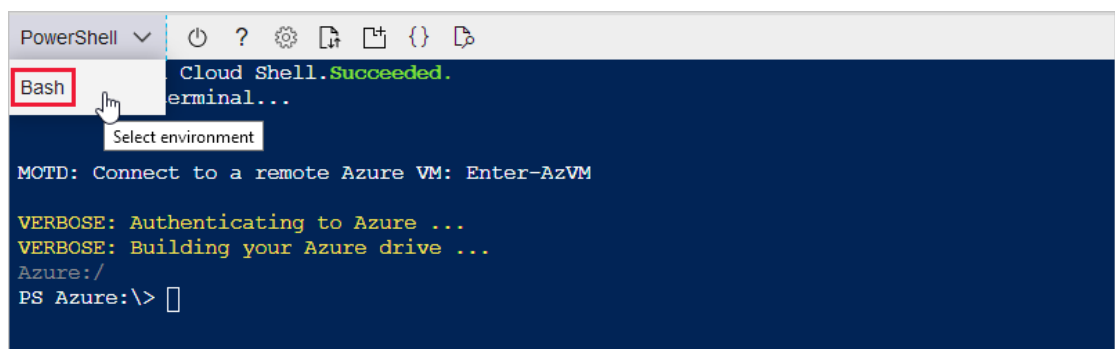
1. If you already have Azure CLI installed locally, run `az --version` to check the version. This tutorial requires Azure CLI 2.5.1 or later.
2. To install or upgrade, see [Install Azure CLI](#). If you install Azure CLI locally, you can run CLI commands in the GCC Command Prompt, Git Bash for Windows, or Powershell.

If you prefer to run Azure CLI in the browser-based Azure Cloud Shell:

1. Use your Azure account credentials to sign into the Azure Cloud shell at <https://shell.azure.com/>.

Note: If this is the first time you've used the Cloud Shell, it prompts you to create storage, which is required to use the Cloud Shell. Select a subscription to create a storage account and Microsoft Azure Files share.

2. Select Bash or Powershell as your preferred CLI environment in the Select environment dropdown. If you plan to use Azure Cloud Shell, keep your browser open to run the Azure CLI commands in this tutorial.



Create an IoT Hub

You can use Azure CLI to create an IoT hub that handles events and messaging for your device.

To create an IoT hub:

1. In your CLI console, run the [az extension add](#) command to add the Microsoft Azure IoT Extension for Azure CLI to your CLI shell. The IoT Extension adds IoT Hub, IoT Edge, and IoT Device Provisioning Service (DPS) specific commands to Azure CLI.

```
az extension add --name azure-iot
```

2. Run the [az group create](#) command to create a resource group. The following command creates a resource group named **MyResourceGroup** in the **eastus** region.

Note: Optionally, to set an alternate **location**, run [az account list-locations](#) to see available locations. Then specify the alternate location in the following command in place of eastus.

```
az group create --name MyResourceGroup --location eastus
```

3. Run the [az iot hub create](#) command to create an IoT hub. It might take a few minutes to create an IoT hub.

YourIotHubName. Replace this placeholder below with the name you chose for your IoT hub. An IoT hub name must be globally unique in Azure. This placeholder is used in the rest of this tutorial to represent your unique IoT hub name.

```
az iot hub create --resource-group MyResourceGroup --name {YourIoTHubName}
```

4. After the IoT hub is created, view the JSON output in the console, and copy the **hostName** value to a safe place. You use this value in a later step. The **hostName** value looks like the following example:

```
{Your IoT hub name}.azure-devices.net
```

Register an IoT Hub device

In this section, you create a new device instance and register it with the IoT hub you created. You will use the connection information for the newly registered device to securely connect your physical device in a later section.

To register a device:

1. In your console, run the [az iot hub device-identity create](#) command. This creates the simulated device identity.

YourIoTHubName. Replace this placeholder below with the name you chose for your IoT hub.

MyDevKit. You can use this name directly for the device in CLI commands in this tutorial. Optionally, use a different name.

```
az iot hub device-identity create --device-id MyDevKit --  
hub-name {YourIoTHubName}
```

2. After the device is created, view the JSON output in the console, and copy the ***deviceId*** and ***primaryKey*** values to use in a later step.

Confirm that you have copied the following values from the JSON output to use in the next section:

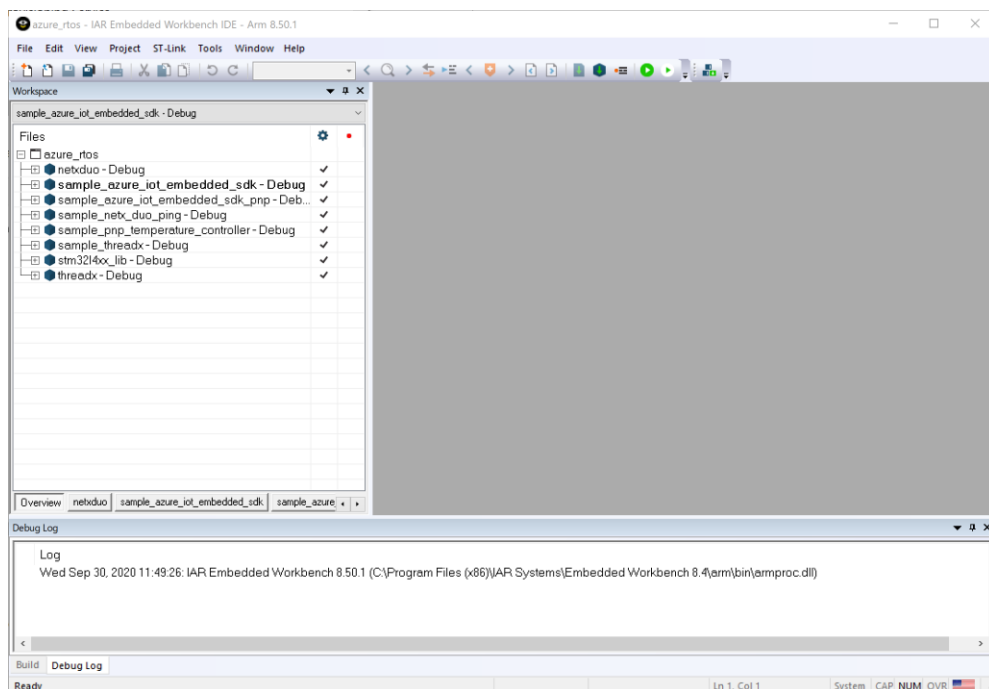
- ***hostName***
- ***deviceId***
- ***primaryKey***

Prepare the device

To connect the device to Azure, you'll modify a configuration file for Azure IoT settings and IAR settings for Wi-Fi, build and flash the image to the device.

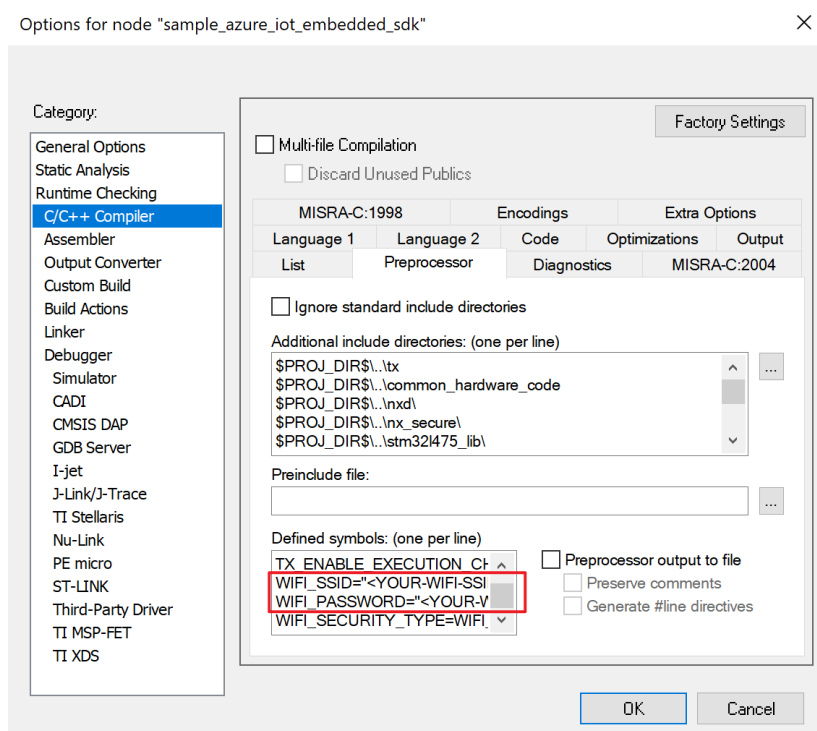
Add configuration

1. Open the **azure_rtos.eww** EWARM Workspace in IAR from the extracted zip file.



2. Select the **sample_azure_iot_embedded_sdk**, right click on it on in the left **Workspace** pane and select **Set as active**.
3. Right click on the active project, select **Options > C/C++ Compiler > Preprocessor**. Replace the values for your WiFi to be used.

Symbol name	Value
WIFI_SSID	{Your Wi-Fi SSID}
WIFI_PASSWORD	{Your Wi-Fi password}



- Expand the sample folder to open **sample_config.h** to set the Azure IoT device information constants to the values that you saved after you created Azure resources.

Constant name	Value
HOST_NAME	{Your IoT hub hostName value}
DEVICE_ID	{Your deviceId value}
DEVICE_SYMMETRIC_KEY	{Your primaryKey value}

The sample project can also run on STM32L475-DISCO IoT Node with just a few configurations:

- Right click on active project, select **Options > General Options**. In **Target** tab, change the device to **ST STM32L475VG**.
- In Options > Linker, replace the configuration file with **common_hardware_code/stm32l475xx_flash.icf**

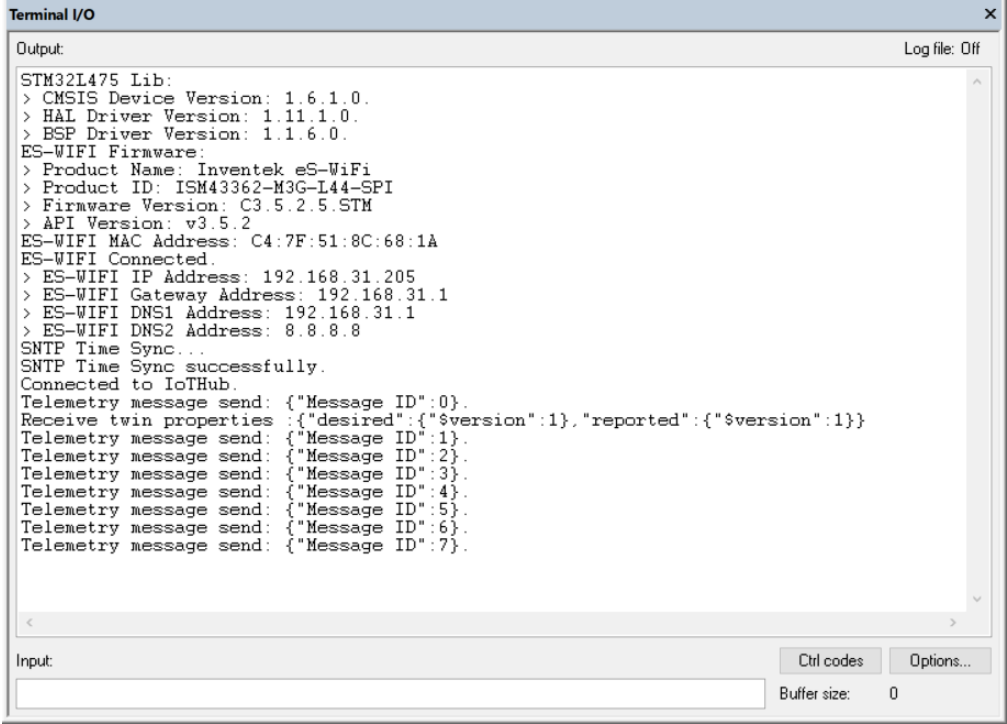
Build the project

In IAR, select **Project > Batch Build** and choose **build_all** and select **Make** to build all projects. You will observe compilation and linking of all sample projects.

Download and run the project

1. Use the Micro USB cable to connect the **USB STLink** port on the STM DevKit to your computer.
2. In IAR, press the green **Download and Debug** button in the toolbar to download the program and run it. Then press **Go**.
3. As the project runs, the demo prints out status information to the Terminal IO window (**View > Terminal I/O**). The demo also publishes the message to IoT Hub every few seconds. Check the Terminal I/O to verify that messages have been successfully sent to the Azure IoT hub.

NOTE: The terminal output content varies depending on which sample you choose to build and run.



The screenshot shows the 'Terminal I/O' window with the following output:

```
Output:
STM32L475 Lib:
> CMSIS Device Version: 1.6.1.0.
> HAL Driver Version: 1.11.1.0.
> BSP Driver Version: 1.1.6.0.
ES-WIFI Firmware:
> Product Name: Inventek eS-WiFi
> Product ID: ISM43362-M3G-L44-SPI
> Firmware Version: C3.5.2.5.STM
> API Version: v3.5.2
ES-WIFI MAC Address: C4:7F:51:8C:68:1A
ES-WIFI Connected.
> ES-WIFI IP Address: 192.168.31.205
> ES-WIFI Gateway Address: 192.168.31.1
> ES-WIFI DNS1 Address: 192.168.31.1
> ES-WIFI DNS2 Address: 8.8.8.8
SNTP Time Sync...
SNTP Time Sync successfully.
Connected to IoTHub.
Telemetry message send: {"Message ID":0}.
Receive twin properties: {"desired":{"$version":1},"reported":{"$version":1}}
Telemetry message send: {"Message ID":1}.
Telemetry message send: {"Message ID":2}.
Telemetry message send: {"Message ID":3}.
Telemetry message send: {"Message ID":4}.
Telemetry message send: {"Message ID":5}.
Telemetry message send: {"Message ID":6}.
Telemetry message send: {"Message ID":7}.
```

At the bottom of the window, there is an 'Input:' field, 'Ctrl codes' and 'Options...' buttons, and a 'Buffer size: 0' indicator.

Keep **Terminal I/O** window open to monitor device output in subsequent steps.

View device properties

You can use the Azure IoT Explorer to view and manage the properties of your devices. In the following steps, you'll add a connection to your IoT hub in IoT Explorer. With the connection, you can view properties for devices associated with the IoT hub.

Download and install latest (above v0.11.1) Azure IoT Explorer from:
<https://github.com/Azure/azure-iot-explorer/releases>

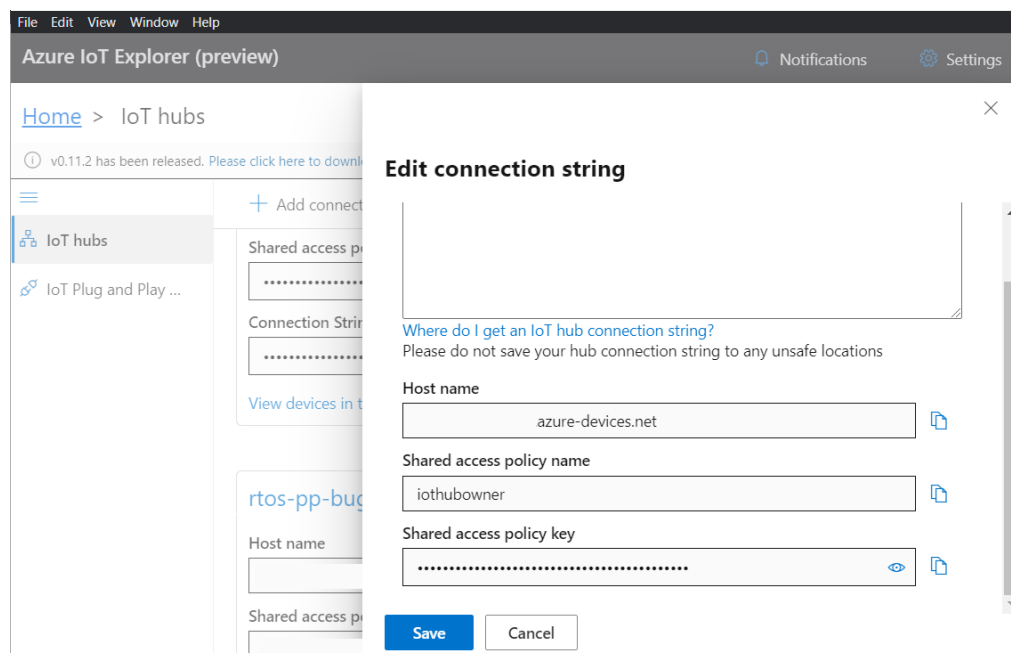
Set IoT Hub

To add a connection to your IoT hub:

1. In your CLI console, run the [az iot hub show-connection-string](#) command to get the connection string for your IoT hub.

```
az iot hub show-connection-string --name {YourIoTHubName}
```

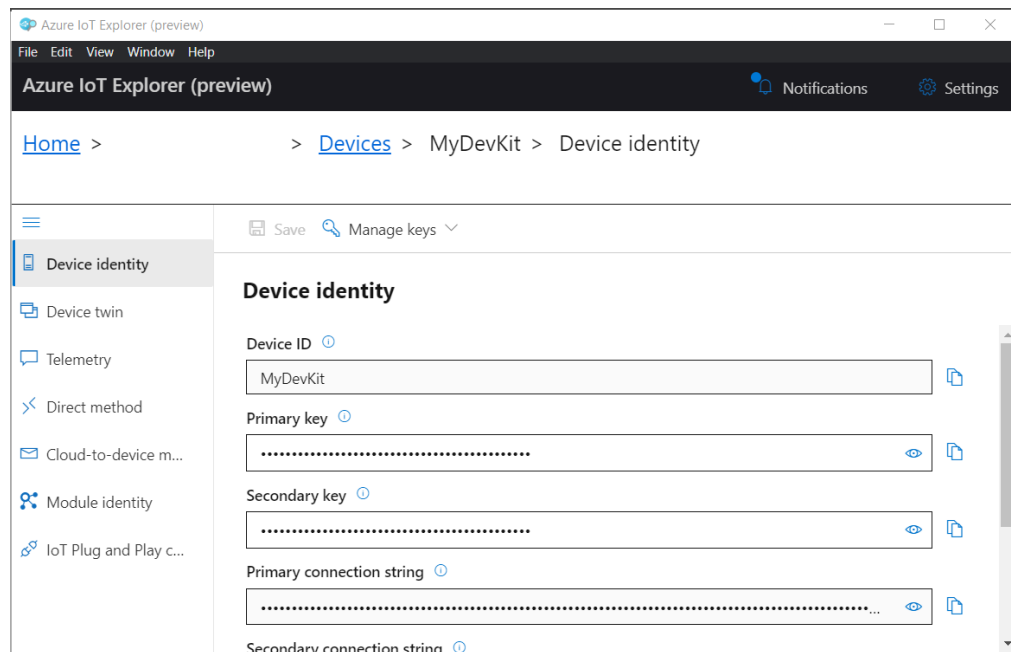
2. Copy the connection string without the surrounding quotation characters.
3. In Azure IoT Explorer, select **IoT hubs > Add connection**.
4. Paste the connection string into the **Connection string** box.
5. Select **Save**.



If the connection succeeds, the Azure IoT Explorer switches to a Devices view and lists your device.

To view device properties using Azure IoT Explorer:

1. Select the link for your device identity. IoT Explorer displays details for the device.

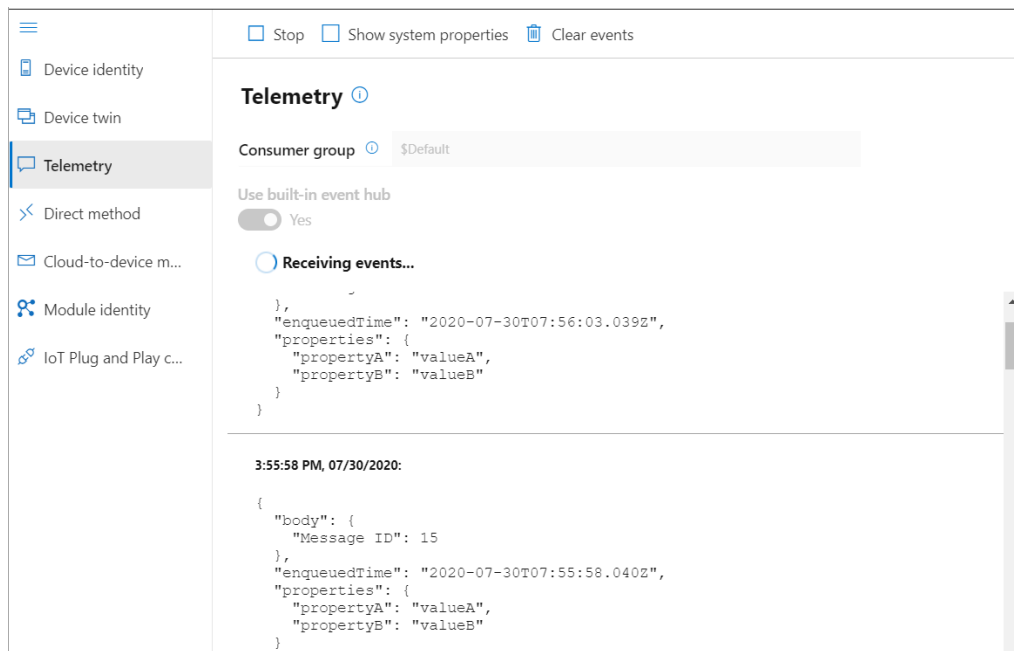


2. Inspect the properties for your device in the **Device identity** panel.

View device telemetry

With Azure IoT Explorer, you can view the flow of telemetry from your device to the cloud. To view telemetry in Azure IoT Explorer:

1. In IoT Explorer select **Telemetry**. Confirm that **Use built-in event hub** is set to Yes.
2. Select **Start**.
3. View the telemetry as the device sends messages to the cloud.



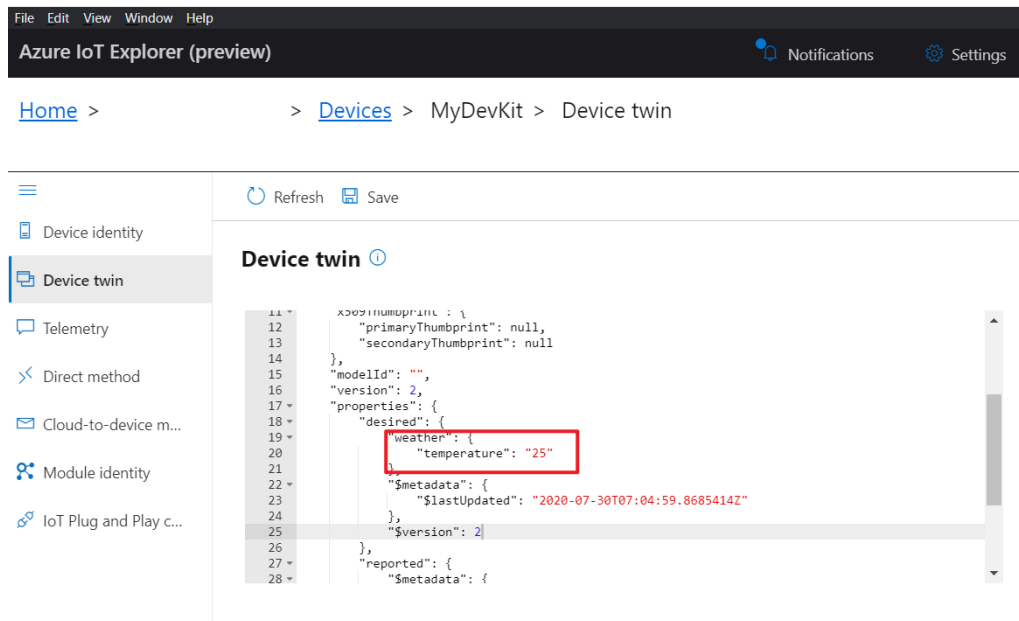
Update device twin

To update device twin in Azure IoT Explorer:

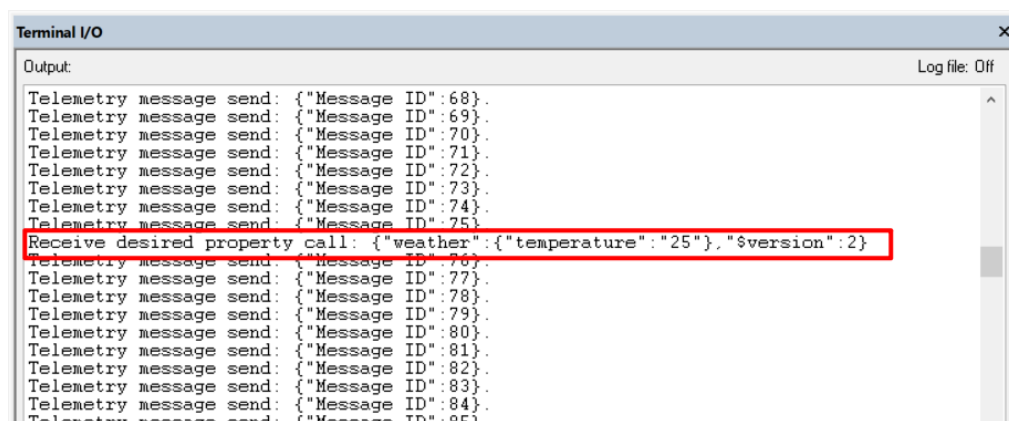
1. In IoT Explorer select **Device twin**.
2. Modify the **desired** section of the Device twin, you can add a custom twin:

```
"weather": {  
  "temperature": "25"  
},
```

3. Select **Save**.



4. View the notification for the device twin update status.
5. In IAR Workbench Terminal I/O window, you can view the desired device twin properties are received.



Call a direct method on device

You can also use Azure IoT Explorer to call a direct method that you have implemented on your device. Direct methods have a name, and can optionally have a JSON payload, configurable connection, and method timeout. To call a direct method in Azure IoT Explorer:

1. In IoT Explorer select **Direct method**.
2. Send a direct method to mimic the device reboot with payload. The device will receive and output the payload as dummy data.

Method name:

reboot

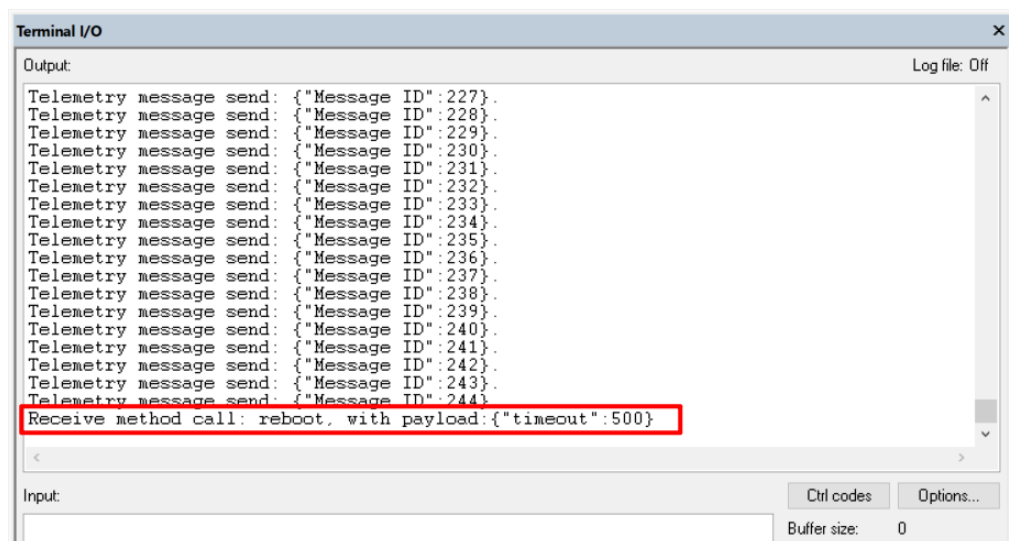
Payload:

```
{"timeout": 500}
```

3. Select **Invoke method**.

The screenshot shows the 'Invoke method' dialog box. On the left is a sidebar with navigation options: Device identity, Device twin, Telemetry, Direct method (selected), Cloud-to-device m..., Module identity, and IoT Plug and Play c... The main area is titled 'Invoke method' and has a 'Direct method' tab. Below the tab, there is a 'Method name' field with the text 'reboot' and a 'Payload' field with the text '{\"timeout\": 500}'. At the bottom, there is a 'Connection timeout in seconds' slider set to 10.

4. In IAR Workbench Terminal I/O window, you can view the method is invoked on the IoT Device.



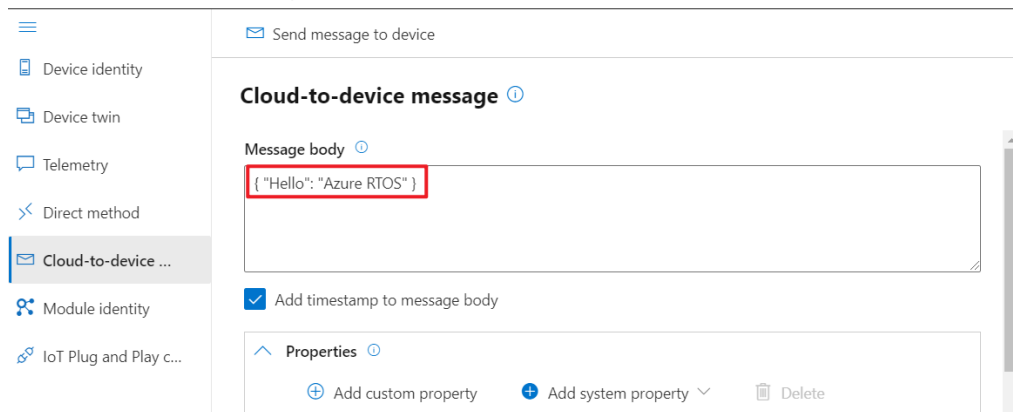
Send cloud-to-device message

To send a cloud-to-device message in Azure IoT Explorer:

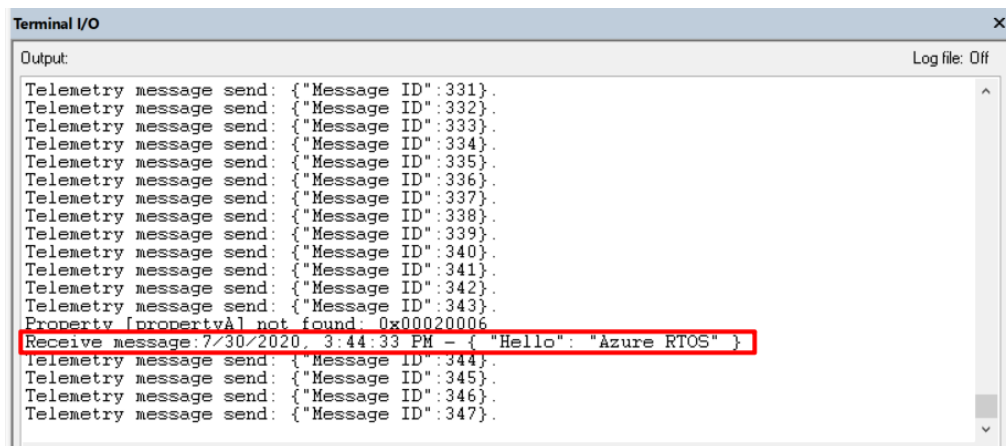
1. In IoT Explorer select **Cloud-to-device message**.
2. Enter the message in the **Message body**.

```
{ \"Hello\": \"Azure RTOS\" }
```


3. Check **Add timestamp to message body**.
4. Select Send message to device.



5. In IAR Workbench Terminal I/O window, you can view the message is received by the IoT Device.



Interact with IoT Plug and Play components

If you are running **sample_azure_iot_embedded_sdk_pnp** or **sample_pnp_temperature_controller** samples, you can use Azure IoT Explorer to interact with IoT Plug and Play components.

Azure IoT explorer needs a local copy of the model file that matches the **Model ID** your device sends. The model file lets Azure IoT explorer display the telemetry, properties, and commands that your device implements.

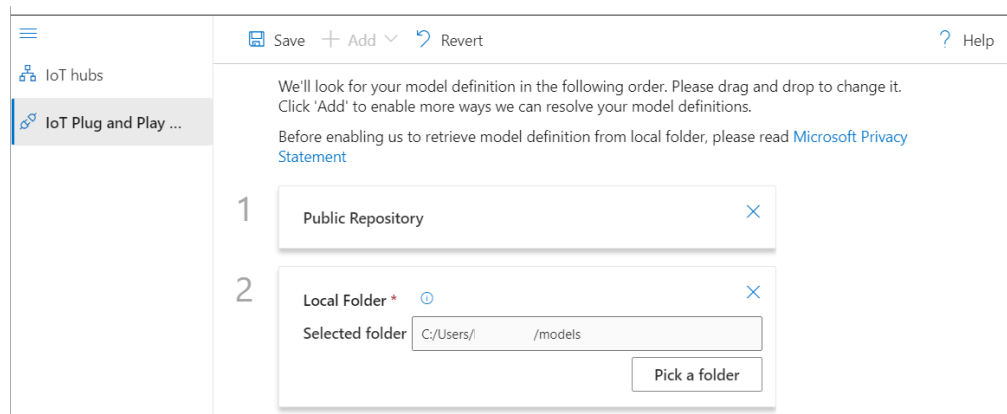
If you haven't already downloaded the sample model files:

1. Create a folder called **models** on your local machine.
2. Right-click [TemperatureController.json](#) and save the JSON file to the models folder.

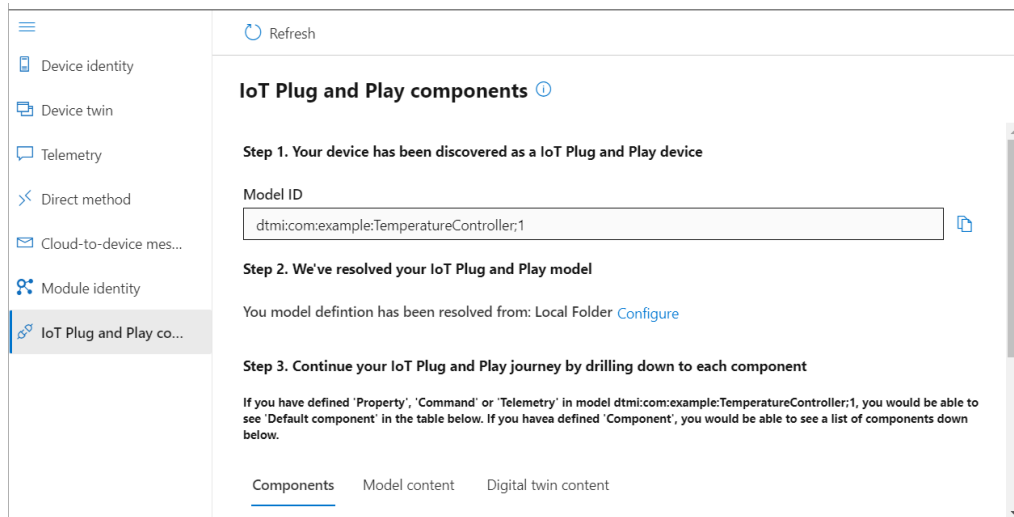
3. Right-click [Thermostat.json](#) and save the JSON file to the models folder.
4. Right-click [DeviceInformation.json](#) and save the JSON file to the models folder.

To use the Azure IoT explorer tool to verify the IoT Plug and Play device application is working:

1. In IoT Explorer, select the **IoT Plug and Play Settings** page.
2. Select **Add**.
3. In **Local folder** section and select **Pick a folder** and open the local models folder where you saved your model files. Then select **Save**.



4. On the **IoT hubs** page, click on the name of the hub you want to work with. You see a list of devices registered to the IoT hub.
5. Click on the **Device ID** of the device you created previously.
6. The menu on the left shows the different types of information available for the device.
7. Select **IoT Plug and Play components** to view the model information for your device.



8. You can view the different components of the device. The default component and any additional ones. Select a component to work with.
9. Select the **Telemetry** page and then select Start to view the telemetry data the device is sending for this component.
10. Select the **Properties (read-only)** page to view the read-only properties reported for this component.
11. Select the **Properties (writable)** page to view the writable properties you can update for this component.
12. Select a property by its **name**, enter a new value for it, and select **Update desired value**.
13. To see the new value show up select the **Refresh** button.
14. Select the **Commands** page to view all the commands for this component.
15. Select the command you want to test set the parameter if any. Select **Send command** to call the command on the device. You can see your device respond to the command in the command prompt window where the sample code is running.

Device identity

Device twin

Telemetry

Direct method

Cloud-to-device mes...

Module identity

IoT Plug and Play co...

Interface

Properties (read-only)

Properties (writable)

Commands

Telemetry

Refresh

Back

You model defintion has been resolved from: Local Folder [Configure](#)

Interface Id

dtmi:com:example:Thermostat;1

Name

Thermostat

Description

Reports current temperature and provides desired temperature control.

1 {

2 "@context": "dtmi:dtdl:context;2",

3 "@id": "dtmi:com:example:Thermostat;1",

4 "@type": "Interface",

5 "displayname": "Thermostat",

6 "description": "Reports current temperature and provides desired temperature control.",

7 "contents": [

8

Use Device Provisioning Service (DPS)

The [IoT Hub Device Provisioning Service \(DPS\)](#) is a helper service for IoT Hub that enables zero-touch, just-in-time provisioning devices to the right IoT hub in a secure and scalable manner. In the following steps, you will enroll the board in DPS using Symmetric Key and provision it automatically in IoT Hub when connecting to the Internet.

Create a Device Provisioning Service (DPS)

You can use Azure CLI to create a DPS to provision the device in IoT Hub automatically.

1. Run the [az iot dps create](#) command to create a DPS. It might take a few minutes to create it.

YourDPSName. Replace this placeholder below with the name you chose for your DPS. An IoT hub name must be globally unique in Azure. This placeholder is used in the rest of this tutorial to represent your unique DPS name.

```
az iot dps create --resource-group MyResourceGroup --name {YourDPSName}
```

2. After the DPS is created, view the JSON output in the console, and copy the ***serviceOperationsHostName*** and ***idScope*** values to a safe place. You use this value in a later step. The ***serviceOperationsHostName*** and ***idScope*** values looks like the following example:

serviceOperationsHostName

```
{Your DPS name}.azure-devices-provisioning.net
```

idScope

```
0nexxxxxxxx
```

3. You can also run the [az iot dps show](#) command to view the values again:

```
az iot dps show --resource-group MyResourceGroup --name {YourDPSName}
```

Link an IoT Hub for DPS

To make the DPS provision the device in IoT Hub, you need to link an IoT Hub for it.

1. Run the [az iot hub show-connection-string](#) command to get the IoT Hub connection string:

```
az iot hub show-connection-string --name {YourIoTHubName}
```

2. Run the [az iot dps linked-hub create](#) command to create a linked IoT Hub in DPS, replace the **YourIoTHubConnectionString** with the actual one you get:

```
az iot dps linked-hub create --dps-name {YourDPSName} --  
resource-group MyResourceGroup --connection-string  
{YourIoTHubConnectionString}
```

Add enrollment in DPS

Now you need to add an individual enrollment record in DPS that later your device can use it to connect to DPS and perform the provisioning in IoT Hub.

1. Run the [az iot dps enrollment create](#) command to create a device enrollment in DPS:

```
az iot dps enrollment create --dps-name {YourDPSName} --  
resource-group MyResourceGroup --attestation-type  
symmetricKey --enrollment-id {MyDPSDevKit}
```

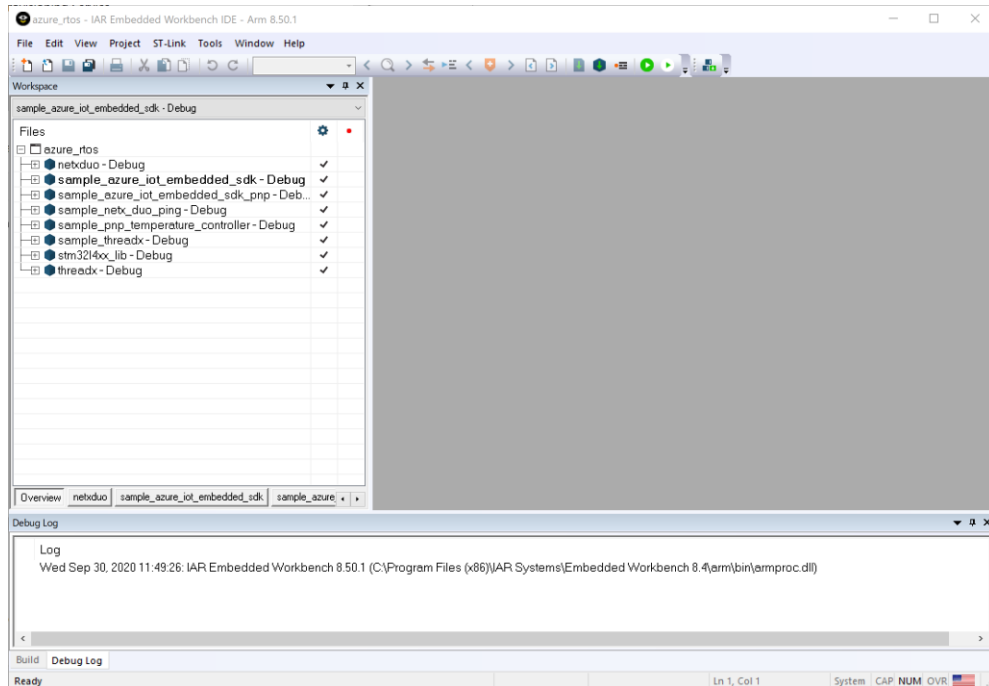
2. After the device is created, view the JSON output in the console, and copy the **registrationId** and **primaryKey** values to use in a later step.

Confirm that you have the copied the following values from the JSON output to use in the next section:

- **serviceOperationsHostName**
- **idScope**
- **registrationId**
- **primaryKey**

Add configuration

1. Open the **azure_rtos.eww** EWARM Workspace in IAR from the extracted zip file.



2. Select the **sample_azure_iot_embedded_sdk**, right click on it on in the left **Workspace** pane and select **Set as active**.
3. Expand the sample folder to open **sample_config.h** to enable the DPS by uncomment the line:

```
#define ENABLE_DPS_SAMPLE
```

4. Further set the Azure IoT DPS constants to the values that you saved after you created Azure IoT DPS resources.

Constant name	Value
ENDPOINT	{Your serviceOperationsHostName value}
ID_SCOPE	{Your idScope value}
REGISTRATION_ID	{Your registrationId value}
DEVICE_SYMMETRIC_KEY	{Your primaryKey value}

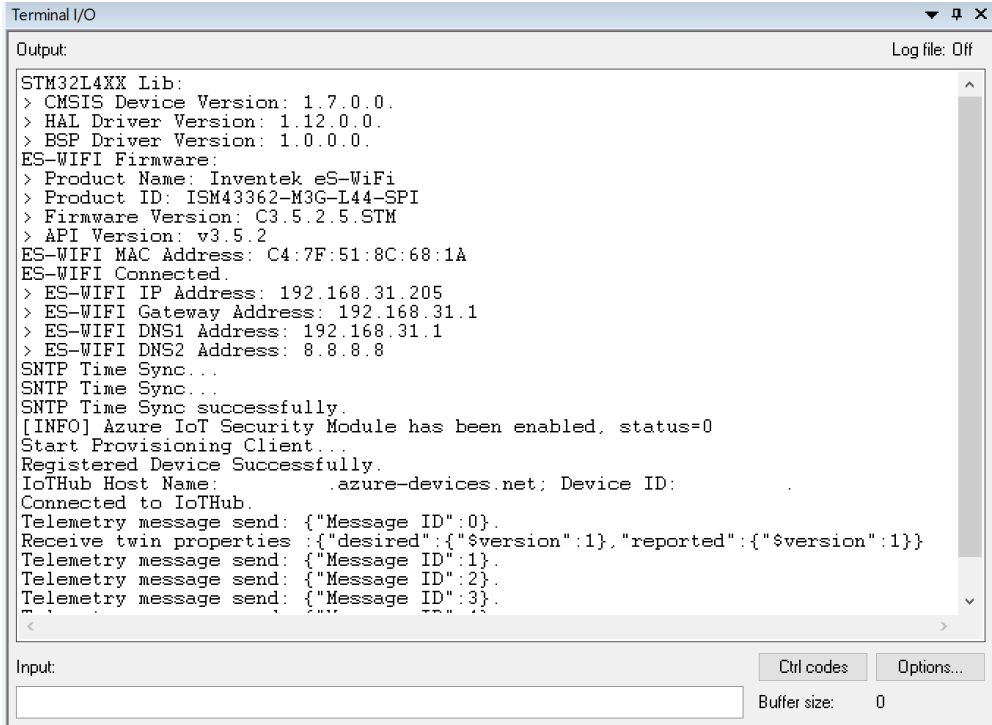
Build the project

In IAR, select **Project > Batch Build** and choose **build_all** and select **Make** to build all projects. You will observe compilation and linking of all sample projects.

Download and run the project

1. Use the Micro USB cable to connect the **USB STLink** port on the STM DevKit to your computer.
2. In IAR, press the green **Download and Debug** button in the toolbar to download the program and run it. Then press **Go**.
3. As the project runs, the demo prints out status information to the Terminal IO window (**View > Terminal I/O**). The demo also publishes the message to IoT Hub every few seconds. Check the Terminal I/O to verify that messages have been successfully sent to the Azure IoT hub.

NOTE: The terminal output content varies depending on which sample you choose to build and run.



The screenshot shows the 'Terminal I/O' window with the following output:

```
STM32L4XX Lib:
> CMSIS Device Version: 1.7.0.0.
> HAL Driver Version: 1.12.0.0.
> BSP Driver Version: 1.0.0.0.
ES-WIFI Firmware:
> Product Name: Inventek eS-WiFi
> Product ID: ISM43362-M3G-L44-SPI
> Firmware Version: C3.5.2.5.STM
> API Version: v3.5.2
ES-WIFI MAC Address: C4:7F:51:8C:68:1A
ES-WIFI Connected.
> ES-WIFI IP Address: 192.168.31.205
> ES-WIFI Gateway Address: 192.168.31.1
> ES-WIFI DNS1 Address: 192.168.31.1
> ES-WIFI DNS2 Address: 8.8.8.8
SNTP Time Sync...
SNTP Time Sync...
SNTP Time Sync successfully.
[INFO] Azure IoT Security Module has been enabled, status=0
Start Provisioning Client...
Registered Device Successfully.
IoT Hub Host Name: .azure-devices.net; Device ID: .
Connected to IoT Hub.
Telemetry message send: {"Message ID":0}.
Receive twin properties :{"desired":{"$version":1},"reported":{"$version":1}}
Telemetry message send: {"Message ID":1}.
Telemetry message send: {"Message ID":2}.
Telemetry message send: {"Message ID":3}.
```

At the bottom of the window, there is an 'Input:' field, 'Ctrl codes' and 'Options...' buttons, and a 'Buffer size: 0' indicator.

Keep **Terminal I/O** window open to monitor device output in subsequent steps.

View device identity

With Azure IoT Explorer, you can see the Device ID the same as the Registration ID you created in DPS be listed in the IoT Hub. This is to confirm a IoT Hub device has been provisioned by the DPS.

Azure IoT Explorer (preview)

File Edit View Window Help

Azure IoT Explorer (preview)NotificationsSettings

Home > > Devices

New Refresh Delete

Query by device ID...Add query parameter

Device ID	Status	Connection st...	Authenticatio...	Last status up...	IoT Plug and ...	Edge device
	Enabled	Disconnected	Sas	--		
	Enabled	Disconnected	Sas	--		
	Enabled	Disconnected	Sas	--		
	Enabled	Connected	Sas	--		

Security monitoring capabilities

IoT devices typically perform a set of operations in a predictable way. Meaning, devices send the same message structure at the same time interval to the same IP address range. [Azure Security Center for IoT](#) leverages this learnable behavior and allows you to build a baseline of your common IoT device behavior. Deviation from expected behavior creates an alert about suspicious activity in your device field.

For example, you can set a baseline for the number of messages the device should send to the IoT Hub in each timeframe, or the IP ranges the device should communicate with.

Any deviation from this expected behavior will trigger an alert. In this example, we will demonstrate capture the malicious IP that should not be communicate with the IoT device.

The Azure Security Center for IoT - RTOS security module provides a comprehensive security solution for Azure RTOS devices. Azure RTOS now ships with the ASC for IoT security module built-in as part of the Azure IoT platform and provides coverage for common threats and potential malicious activities.

For more information: <https://docs.microsoft.com/en-us/azure/asc-for-iot/iot-security-azure-rtos>

Before using it

The Azure Security Center for IoT - RTOS security module is part of the [Azure IoT Middleware for Azure RTOS](#) and is enabled by default. It analyzes inbound and outbound network activity on IPv4 and IPv6 Supported protocols:

- TCP
- UDP
- ICMP

And with below data collected:

- Local and remote addresses
- Local and remote port numbers
- Number of Bytes received
- Number of Bytes transmitted

You can disable it by defining the following symbol in the netxduo/nx_port.h file before building the NetX Duo library:

```
#define NX_AZURE_DISABLE_IOT_SECURITY_MODULE
```

Resource requirements

Azure Security Center for IoT leverages existing Azure RTOS resources, and sends security messages in the background, without interfering with the user application, using the same connection to the IoT Hub. The following table shows typical memory footprint (for RAM and flash) on a Cortex M7 device. The RAM and ROM usage may vary depending on the build options.

- Memory Footprint (using default config – 4 unique monitored connection in IPv4 in an hour):

Toolchain	RAM	ROM
IAR Embedded workbench	4Kb	10Kb
GNU Arm Embedded	4Kb	13Kb

- Additional resources implications for each additional connection:

Connection type	RAM	Network bandwidth
IPv4	52bytes	36bytes
IPv6	200bytes	60bytes

Network bandwidth calculation:

Total bandwidth (in bytes) = Metadata (e.g. 300bytes) + IPv4_connections * 36 + IPv6_connections * 60

To fine tune security module you can refer to the [configuration page](#).

Limitation

The STM32L4+ and L4 boards don't support ICMP connections now.

Pricing

There are no additional costs for including the security module in NetX Duo. However, for devices on metered network, data from ASC traffic may incur additional cost.

On IoT hub, associated costs are added only if Azure Security Center for IoT service is enabled in IoT Hub. Refer to the [pricing page](#) for more information.

To learn more about Azure Security Center for IoT, view:
<https://docs.microsoft.com/azure/defender-for-iot>

View alerts

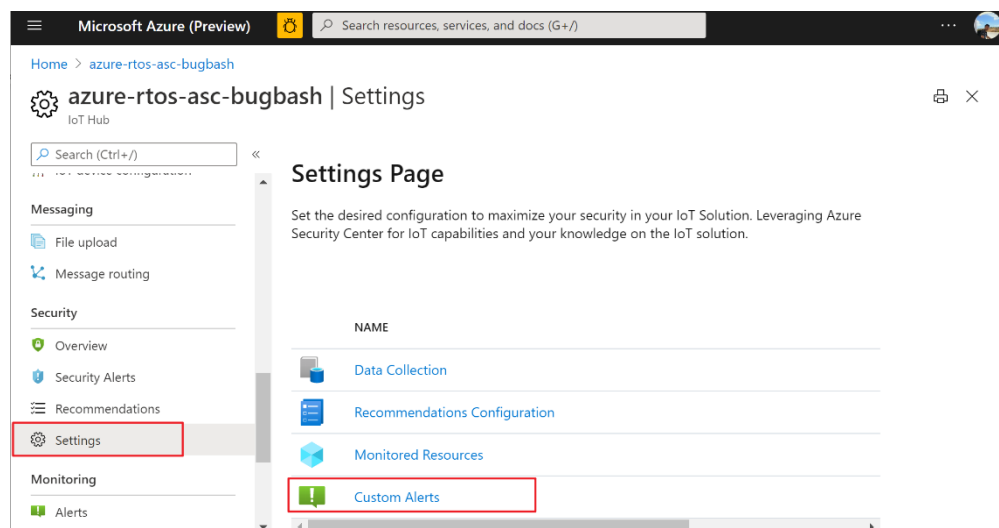
To view the captured alerts by the security group in Azure Security Center:

To view the captured alerts by the security group in Azure Security Center:

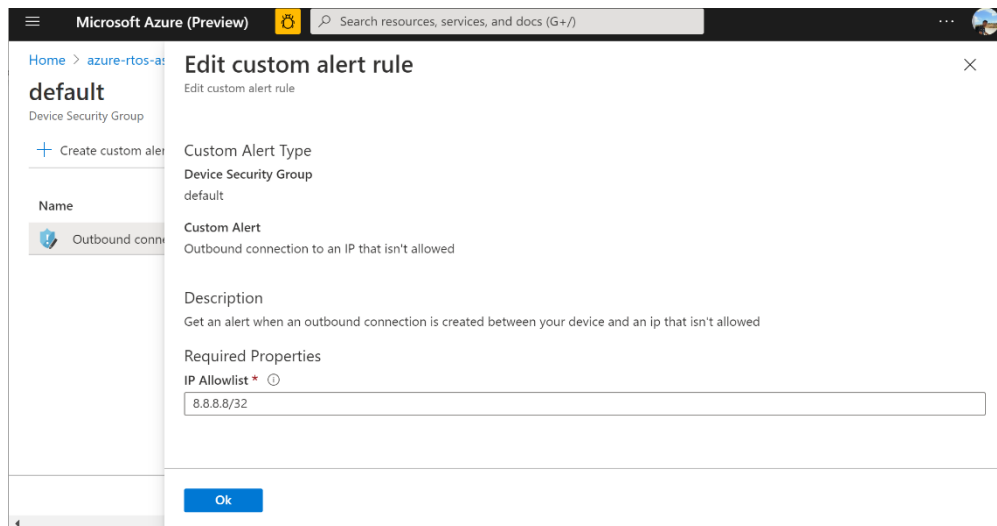
1. Ensure **sample_azure_iot_embedded_sdk** sample application is up and running. From the terminal output, you can see the Azure IoT Security Module is enabled by default.

```
DHCP In Progress...
IP address: 10.94.192.55
Mask: 255.255.240.0
Gateway: 10.94.192.1
DNS Server address: 10.50.50.50
[INFO] Azure IoT Security Module has been enabled, status=0
Connected to IoTHub.
Telemetry message send: {"Message ID":0}.
Receive twin properties :{"desired":{"$version":1},"reported":{"$version":1}}
```

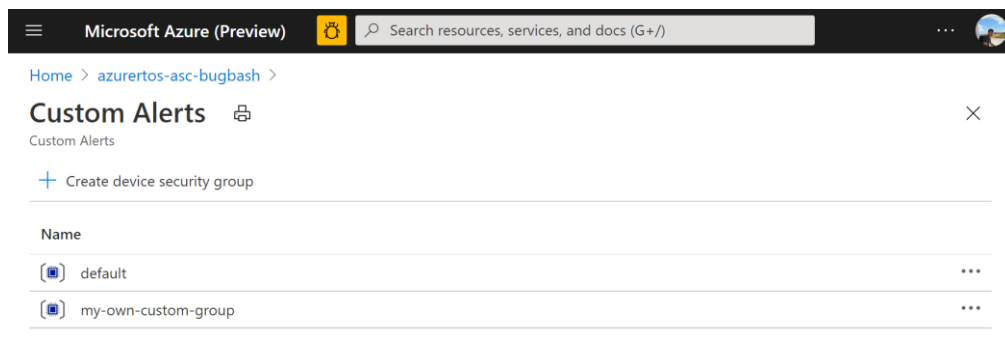
2. In Azure portal, go to the IoT Hub and select **Settings > Custom Alerts** in **Security** tab.



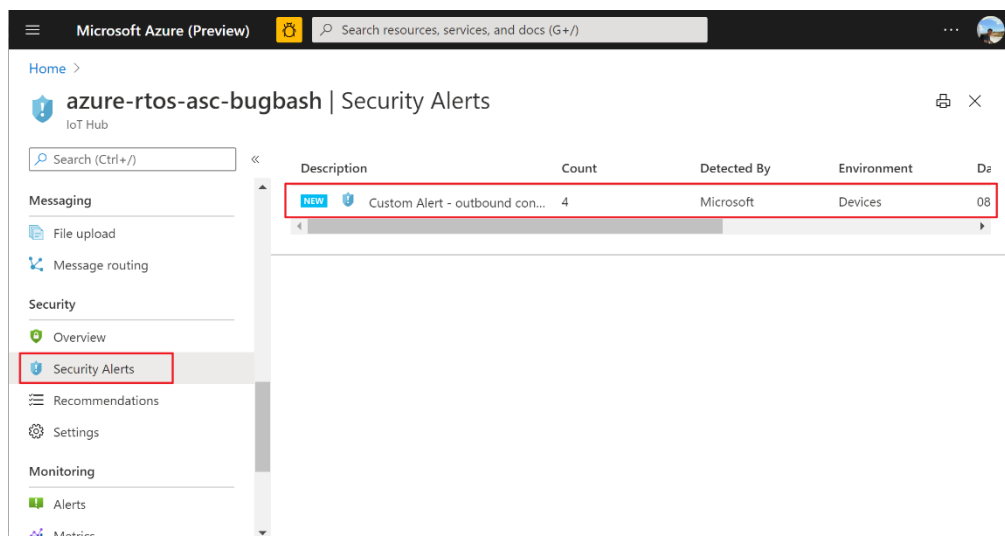
3. In the default security group, you can see there is a custom alert rule set to capture every IP access except for **8.8.8.8**. In the real scenario, you can opt-out the legit IP addresses that can communicate with the device and capture the rest be identified as malicious access. Then they can be reflected in the Security Alert view above.



- Also you can add more customer security groups with [custom alert](#) rules set for it.



- After the security group is configured, select **Security Alerts** in **Security** tab.



NOTE: From the time you start running the IoT device application, it may take up to 45 minutes for the events to be populated into the Security Alerts view.

6. Select from alert list to view the alert details. You can find more details about it from [here](#).

The screenshot displays the Microsoft Azure Security Alerts interface. The top navigation bar shows 'Microsoft Azure (Preview)' and a search bar. The main header indicates the current view is 'Security Alert | Custom Alert - outbound connection created to an IP that isn't allowed'. Below this, a sidebar on the left provides filters for 'Activity time' (8/26/2020), 'Severity' (Low), 'Total Alerts' (27), and 'Detected by' (Microsoft). The main area shows a list of 27 alerts, with columns for 'Alert ID', 'Device ID', 'Activity Status', and 'Activity Time'. An 'Alert details' panel is open on the right, showing details for alert ID 46a64295-3179-4f68-bc58-fc21c5ceac61. The details include the device 'liya-x86-sim', the process 'Process', and additional properties: 'RemoteAddress: 10.168.98.110', 'RemotePort: 52033', and 'Protocol: TCP'.

Alert ID	Device ID	Activity Status	Activity Time
liya-x86-sim (21)			
6c098983-91be-4c...	liya-x86-sim	8/26/2020, ...	8/26/2020, ...
46a64295-3179-4f6...	liya-x86-sim	8/26/2020, ...	8/26/2020, ...
4159f387-7302-426...	liya-x86-sim	8/26/2020, ...	8/26/2020, ...
e2c684e1-527d-42...	liya-x86-sim	8/26/2020, ...	8/26/2020, ...

All event is captured in the Security Alert view from the IoT device since there is a default security group has been created for it.

Clean up resources

If you no longer need the Azure resources created in this tutorial, you can use the Azure CLI to delete the resource group and all the resources you created for this tutorial. Optionally, you can use Azure IoT Explorer to delete individual resources including devices and IoT hubs.

If you continue to another tutorial in this getting started guide, you can keep the resources you've already created and reuse them.

Important: Deleting a resource group is irreversible. The resource group and all the resources contained in it are permanently deleted. Make sure that you do not accidentally delete the wrong resource group or resources.

To delete a resource group by name:

1. Run the [az group delete](#) command. This removes the resource group, the IoT Hub, and the device registration you created.

```
az group delete --name MyResourceGroup
```

2. Run the [az group list](#) command to confirm the resource group is deleted.

```
az group list
```

Next steps

In this tutorial you built and flash the application that contains Azure IoT Middleware for Azure RTOS sample code. You also used the Azure CLI to create Azure resources, connect the device securely to Azure. And eventually view telemetry, and send messages using Azure IoT Explorer.

To learn more about Azure RTOS and how it works with Azure IoT, view <https://azure.com/rtos>.