



Azure RTOS IoT Embedded SDK Sample Microchip SAM E54 Xplained Pro using MPLAB X IDE User Guide

Published: November 2022

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.

© 2022 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 6.2

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.....	10
Download and run the project.....	10
<i>View device properties</i>	<i>13</i>
Set IoT Hub	13
View device telemetry	14
Update device twin	15
Call a direct method on device	16
Send cloud-to-device message	17
<i>Use Device Provisioning Service (DPS)</i>	<i>18</i>
Create a Device Provisioning Service (DPS)	18
Link an IoT Hub for DPS	19
Add enrollment in DPS	19
Add configuration	20
Build the project.....	20
Download and run the project.....	21
View device identity	22
<i>Security monitoring capabilities</i>	<i>23</i>
Before using it	23
Resource requirements	24
Pricing.....	24
View alerts.....	25
<i>Clean up resources</i>	<i>28</i>
<i>Next steps</i>	<i>29</i>

Overview

The following steps detail how to configure, build and execute the Microsoft Azure IoT integration example using Azure IoT Embedded C on the Microchip SAM E54 Xplained Pro, using MPLAB X IDE 6.0.0 development tools. It can be downloaded from this page:

<https://www.microchip.com/mplab/mplab-x-ide>

You will also need MPLAB XC32/32++ Compiler 4.1.0 or later to build the sample projects. It can be downloaded from this page:

<https://www.microchip.com/mplab/compilers>

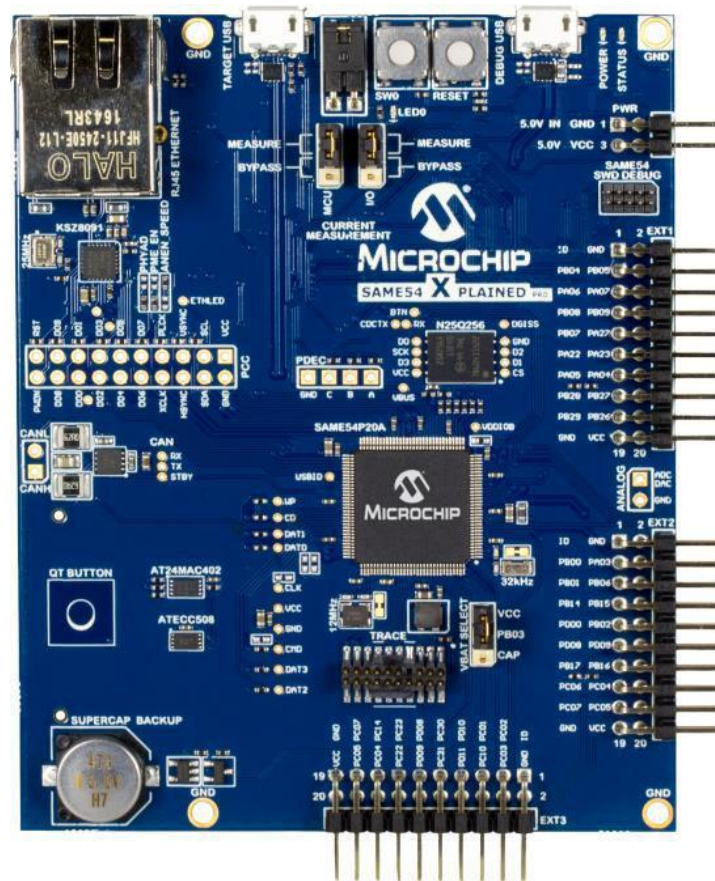


Figure 1 Microchip SAM E54 Xplained Pro

The sample distribution zip file contains the following sub-folders:

Folder	Contents
<i>azure_iot</i>	Azure IoT Middleware for Azure RTOS source code
<i>common_hardware_code</i>	Common code for SAM E54 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>same54_lib</i>	SAM E54 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:

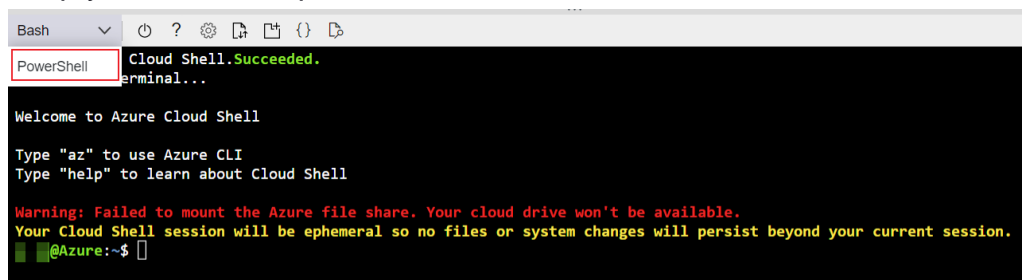
1. If you already have Azure CLI installed locally, run `az --version` to check the version. This tutorial requires Azure CLI 2.36.0 or later.
2. To install or upgrade, see [How to install the Azure CLI](#). If you install Azure CLI locally, you can run CLI commands in the Command Prompt, Git Bash for 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.



```
Bash [v] [?] [⚙] [📁] [🔍] [🔌] [🔌]
PowerShell Cloud Shell.Succeeded.
terminal...

Welcome to Azure Cloud Shell

Type "az" to use Azure CLI
Type "help" to learn about Cloud Shell

Warning: Failed to mount the Azure file share. Your cloud drive won't be available.
Your Cloud Shell session will be ephemeral so no files or system changes will persist beyond your current session.
@Azure:~$
```

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
```

TIP: You can get host name again by using:

```
az iot hub list --query "[].{hostname:
properties.hostName}" --output table
```

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***

TIP: You can get **primaryKey** again by using:

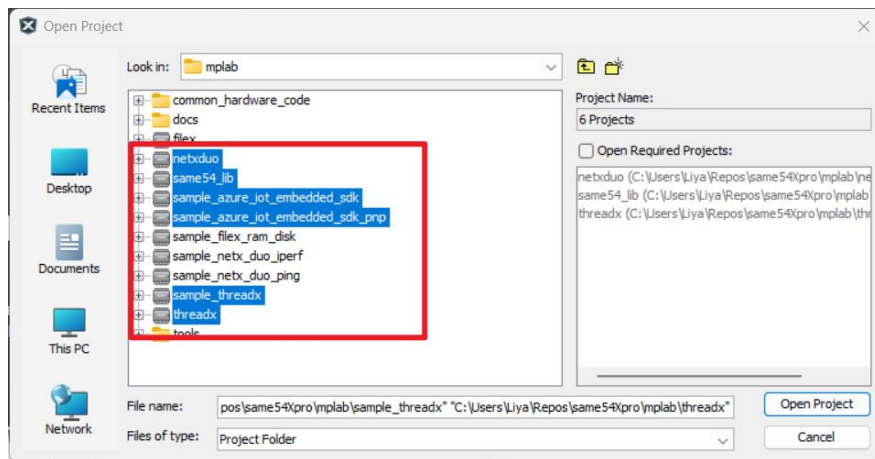
```
az iot hub device-identity show --hub-name {YourIoTHubName} --
device-id MyDevKit
```


Prepare the device

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

Add configuration

1. Open MPLab and select **File > Open Project** and select the following projects from the extracted zip file. You can hold the **Ctrl** key to select multiple projects in the window.
 - threadx
 - netxduo
 - same54_lib
 - sample_azure_iot_embedded_sdk
 - sample_azure_iot_embedded_sdk_pnp

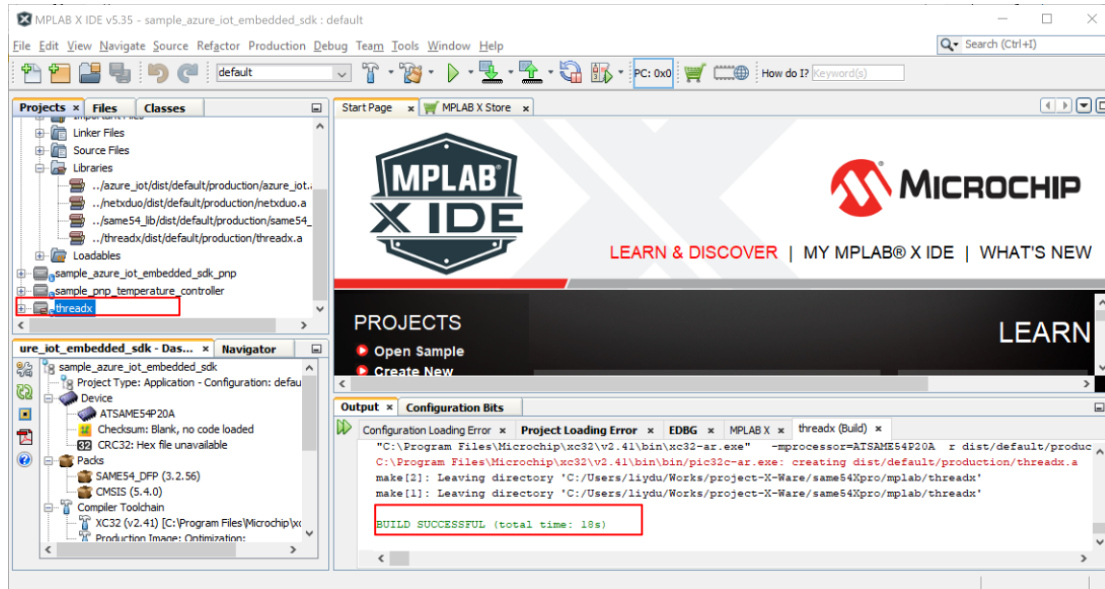


2. Select the **sample_azure_iot_embedded_sdk** project, right click on it on in the left **Projects** pane and select **Set as main project**.
3. Expand the sample folder to open **sample_config.h** to set the Azure IoT device information constants to the values that you saved after you created Azure resources.

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

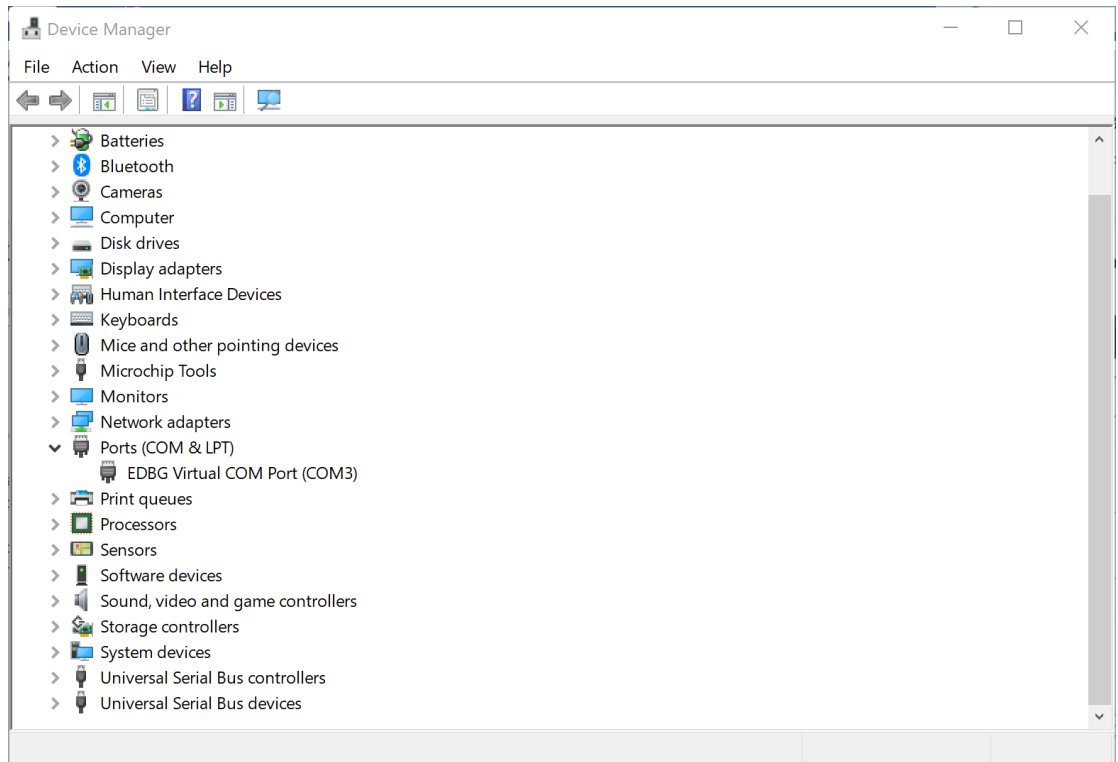
Build the project

Make sure all sample project's dependent libraries (**azure_iot**, **threadx**, **netxduo**, **same54_lib**) are built by select the project in the **Projects** pane, right click on it and select **Build**.



Download and run the project

1. Use the Micro USB cable to connect the **Debug USB** port on the Microchip SAM E54, and then connect it to your computer.
2. Use the Ethernet cable to connect the Microchip SAM E54 to an Ethernet port.
3. In MPLAB, select **Debug > Debug Main Project**.
4. Select the connect SAM E54 board.
5. Verify the serial port in your OS's device manager. It should show up as a COM port.



6. Open your favorite serial terminal program such as [Putty](#) and connect to the COM port discovered above. Configure the following values for the serial ports:

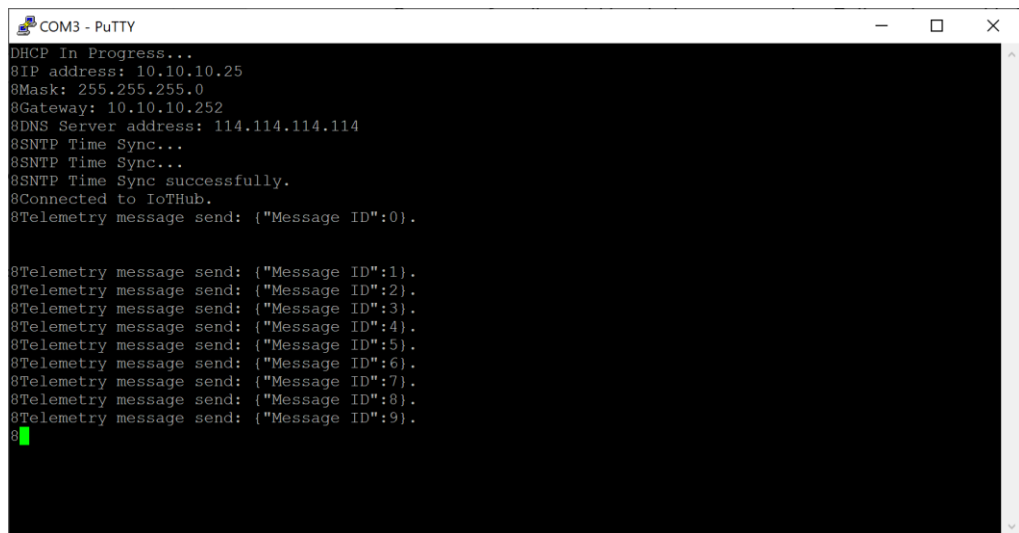
Baud rate: **115200**

Data bits: **8**

Stop bits: **1**

7. As the project runs, the demo prints out status information to the terminal output window. The demo also publishes the message to IoT Hub every few seconds. Check the terminal output 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.



```
COM3 - PuTTY
DHCP In Progress...
8IP address: 10.10.10.25
8Mask: 255.255.255.0
8Gateway: 10.10.10.252
8DNS Server address: 114.114.114.114
8SNTP Time Sync...
8SNTP Time Sync...
8SNTP Time Sync successfully.
8Connected to IoTHub.
8Telemetry message send: {"Message ID":0}.

8Telemetry message send: {"Message ID":1}.
8Telemetry message send: {"Message ID":2}.
8Telemetry message send: {"Message ID":3}.
8Telemetry message send: {"Message ID":4}.
8Telemetry message send: {"Message ID":5}.
8Telemetry message send: {"Message ID":6}.
8Telemetry message send: {"Message ID":7}.
8Telemetry message send: {"Message ID":8}.
8Telemetry message send: {"Message ID":9}.
8
```

Keep terminal 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 v0.16.0) Azure IoT Explorer from: <https://github.com/Azure/azure-iot-explorer/releases>

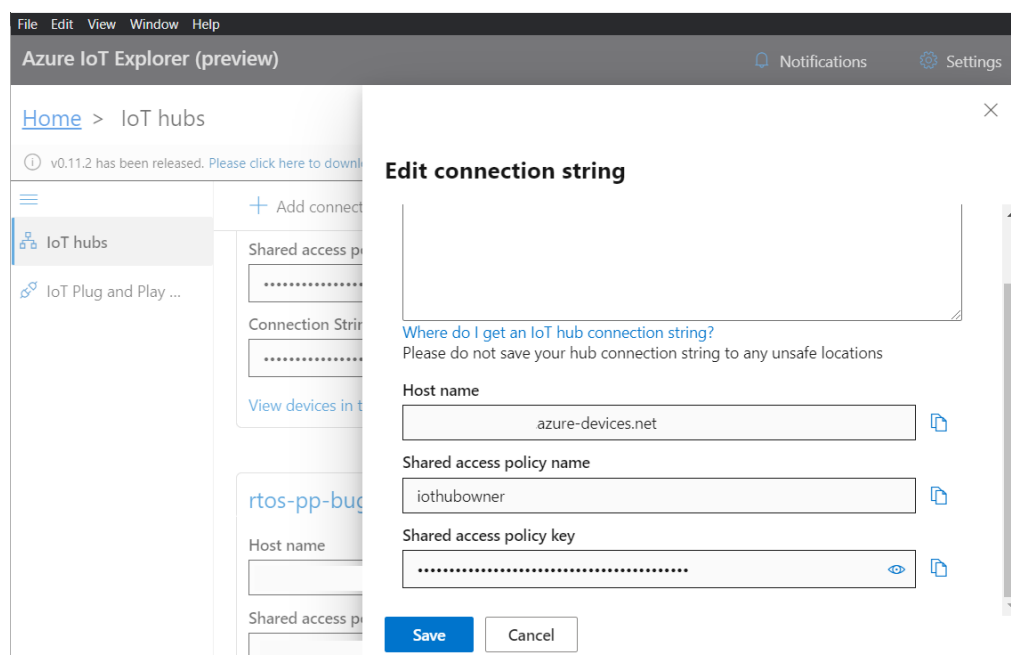
Set IoT Hub

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}
```

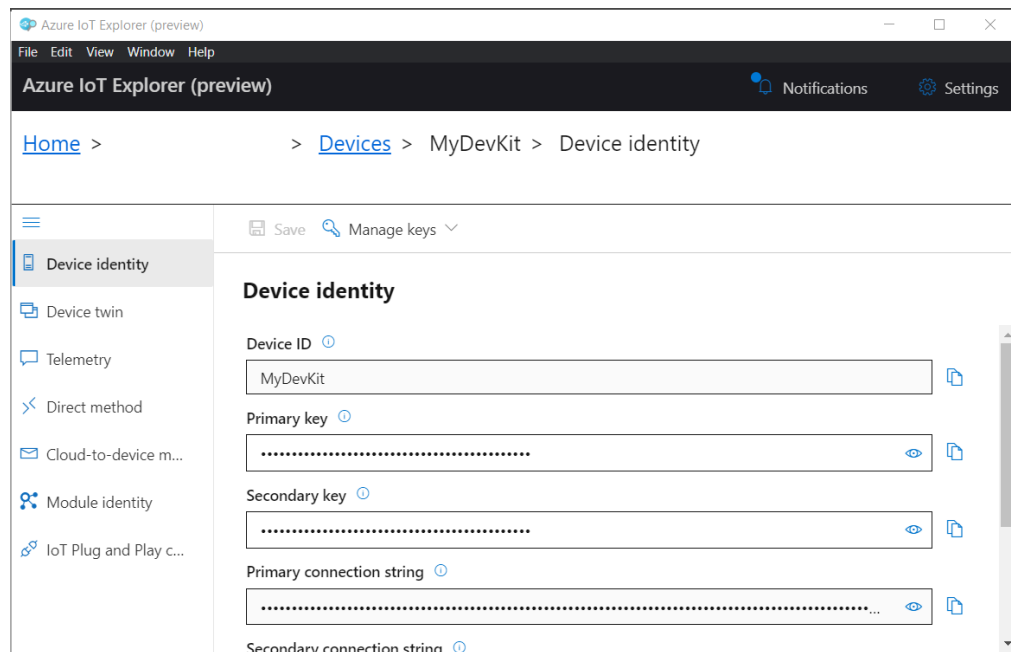
2. Copy the connection string without the surrounding quotation characters.
3. In Azure IoT Explorer, select **IoT hubs > Add connection**.
4. Paste the connection string into the **Connection string** box.
5. Select **Save**.



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.

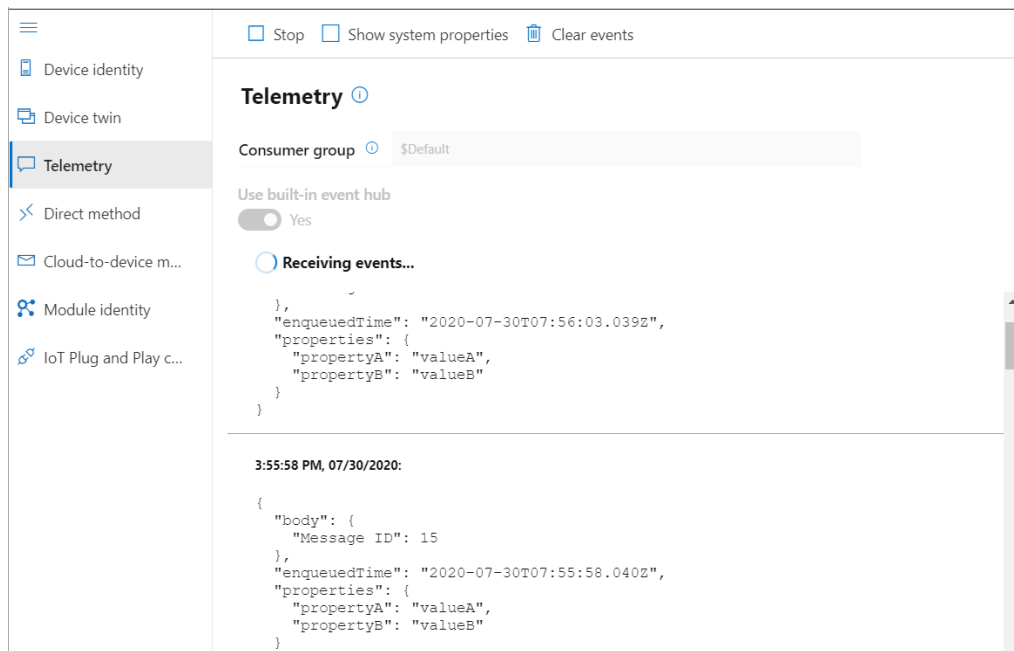


2. Inspect the properties for your device in the **Device identity** panel.

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.



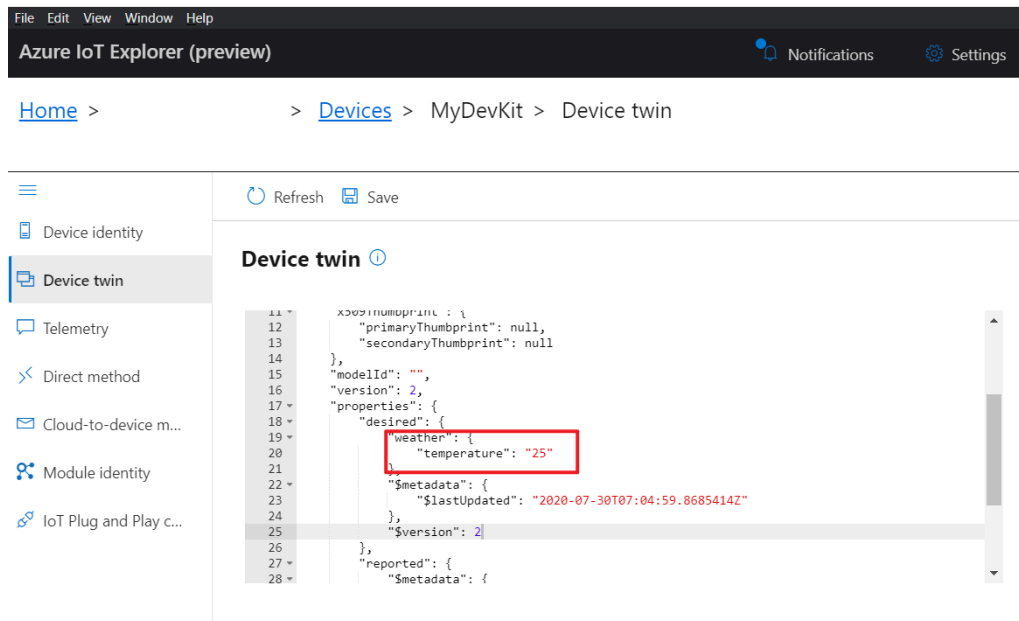
Update device twin

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 terminal output 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**.

Invoke method

Direct method

Method name *

reboot

Payload

{\"timeout\": 500}

Connection timeout in seconds

10

4. In terminal output 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.

Send message to device

Cloud-to-device message

Message body

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

☒ Add timestamp to message body

Properties

+ Add custom property + Add system property Delete

5. In terminal output window, you can view the message is received by the IoT Device.

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}
```

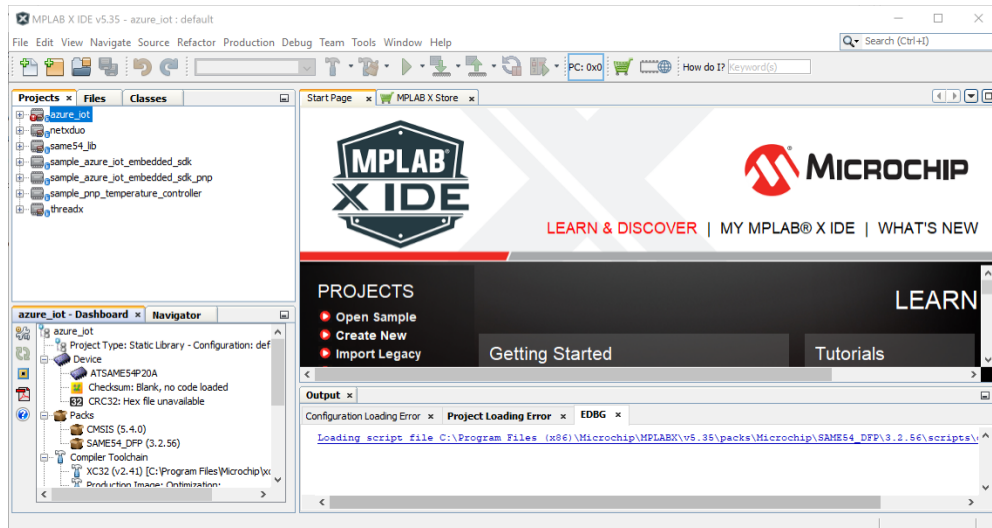
2. After the device is created, view the JSON output in the console, and copy the **registrationId** and **primaryKey** values to use in a later step.

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 MPLab and select **File > Open Project** and select all projects from the extracted zip file.



2. Select the **sample_azure_iot_embedded_sdk** project, right click on it on in the left **Projects** pane and select **Set as main project**.
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.

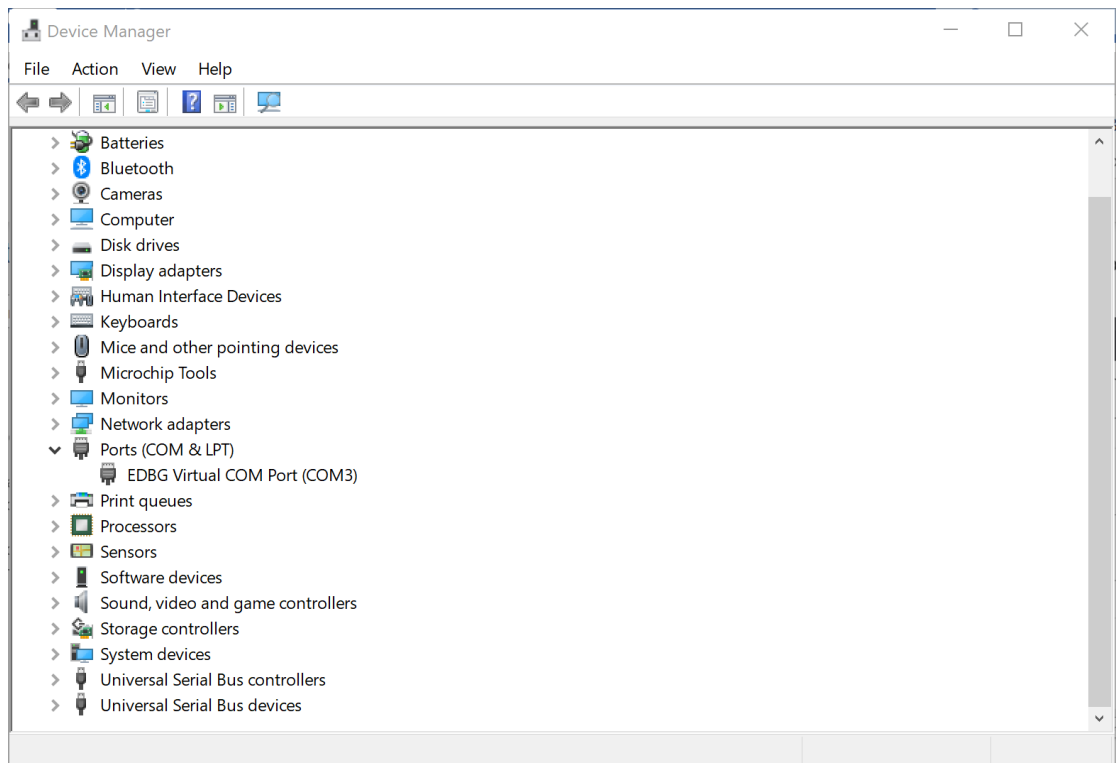
Constant name	Value
ENDPOINT	{Your serviceOperationsHostName value}
ID_SCOPE	{Your idScope value}
REGISTRATION_ID	{Your registrationId value}
DEVICE_SYMMETRIC_KEY	{Your primaryKey value}

Build the project

Make sure all sample project's dependent libraries (**azure_iot**, **threadx**, **netxduo**, **same54_lib**) are built by select the project in the **Projects** pane, right click on it and select **Build**.

Download and run the project

1. Use the Micro USB cable to connect the **Debug USB** port on the Microchip SAM E54, and then connect it to your computer.
2. Use the Ethernet cable to connect the Microchip SAM E54 to an Ethernet port.
3. In MPLAB, select **Debug > Debug Main Project**.
4. Select the connect SAM E54 board.
5. Verify the serial port in your OS's device manager. It should show up as a COM port.



6. Open your favorite serial terminal program such as [Putty](#) and connect to the COM port discovered above. Configure the following values for the serial ports:

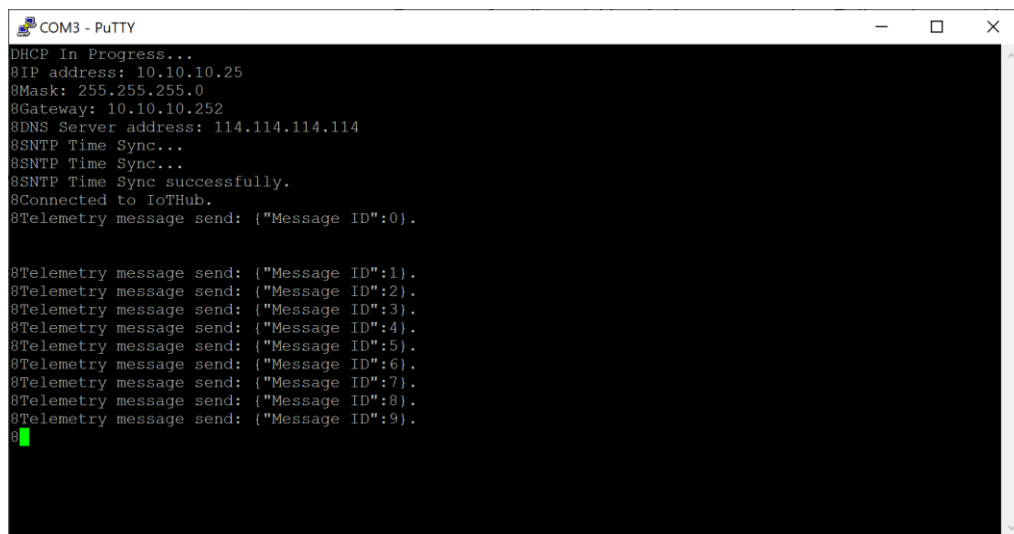
Baud rate: **115200**

Data bits: **8**

Stop bits: **1**

7. As the project runs, the demo prints out status information to the terminal output window. The demo also publishes the message to IoT Hub every few seconds. Check the terminal output 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.



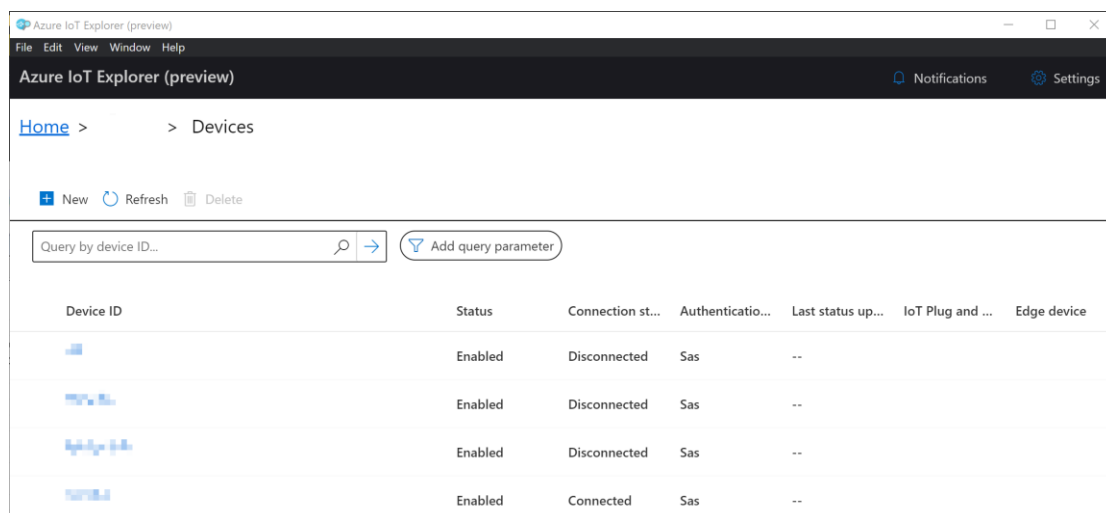
```
COM3 - PuTTY
DHCP In Progress...
8IP address: 10.10.10.25
8Mask: 255.255.255.0
8Gateway: 10.10.10.252
8DNS Server address: 114.114.114.114
8SNTP Time Sync...
8SNTP Time Sync...
8SNTP Time Sync successfully.
8Connected to IoTHub.
8Telemetry message send: {"Message ID":0}.





8Telemetry message send: {"Message ID":1}.
8Telemetry message send: {"Message ID":2}.
8Telemetry message send: {"Message ID":3}.
8Telemetry message send: {"Message ID":4}.
8Telemetry message send: {"Message ID":5}.
8Telemetry message send: {"Message ID":6}.
8Telemetry message send: {"Message ID":7}.
8Telemetry message send: {"Message ID":8}.
8Telemetry message send: {"Message ID":9}.
8
```

Keep terminal 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.



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](#).

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

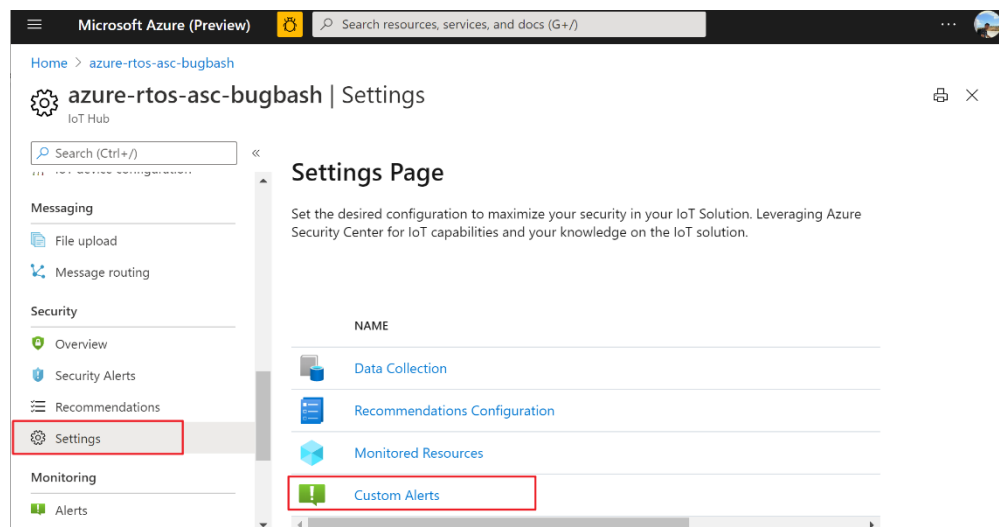
To view the captured alerts by the security group in Azure Security Center:

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

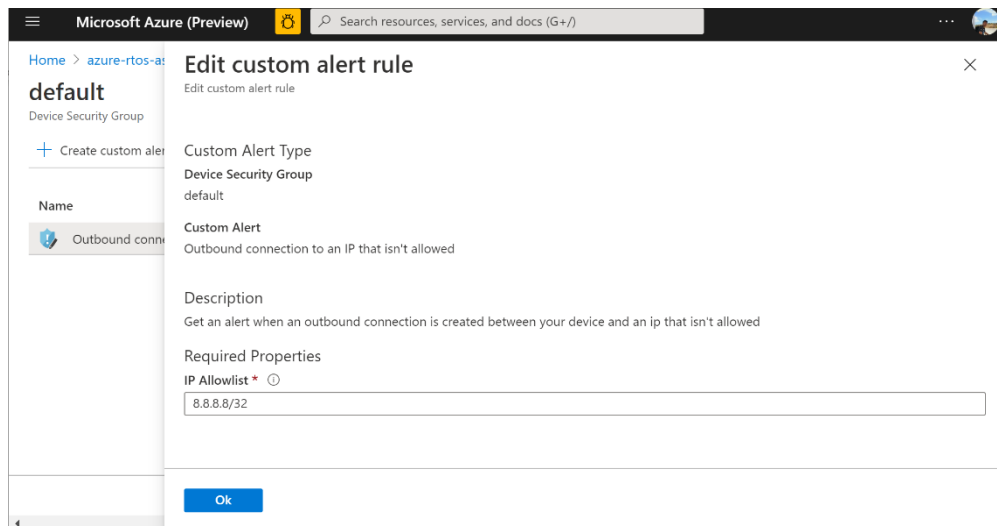
1. Ensure **sample_azure_iot_embedded_sdk** sample application is up and running. From the terminal output, you can see the Azure IoT Security Module is enabled by default.

```
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 IoT Hub.
Telemetry message send: {"Message ID":0}.
Receive twin properties : {"desired":{"$version":1},"reported":{"$version":1}}
```

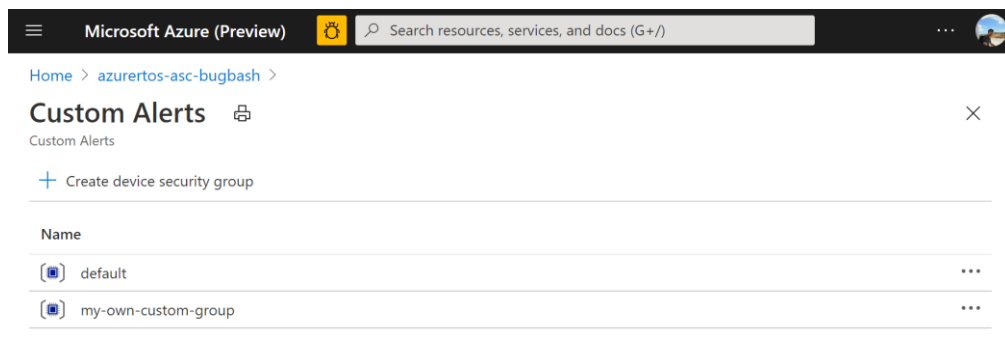
2. In Azure portal, go to the IoT Hub and select **Settings > Custom Alerts** in **Security** tab.



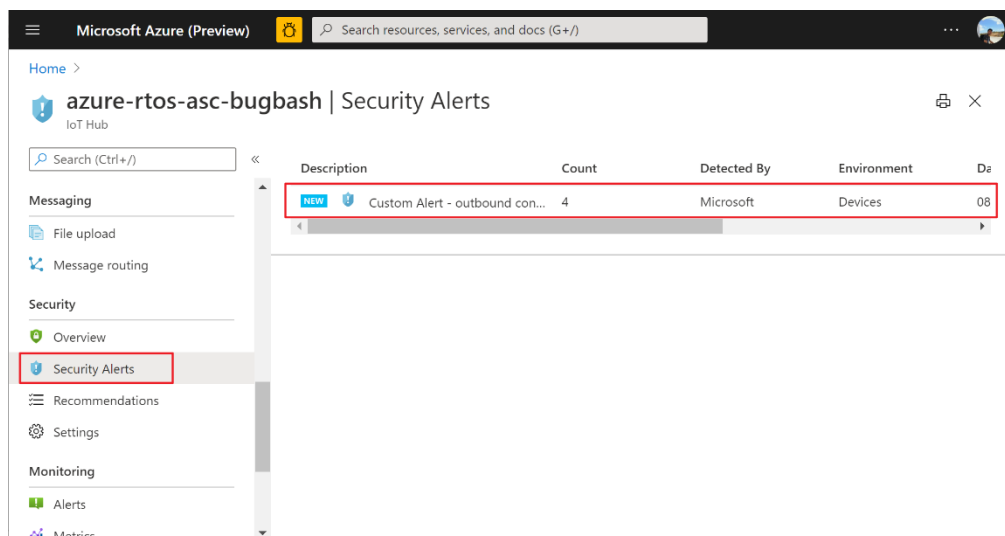
3. In the default security group, you can see there is a custom alert rule set to capture every IP access except for **8.8.8.8**. In the real scenario, you can opt-out the legit IP addresses that can communicate with the device and capture the rest be identified as malicious access. Then they can be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



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



5. 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 summary information: '8/26/2020' for the activity time, 'Low' severity, '27' total alerts, and 'Microsoft' as the detected by. The main content area shows a list of alerts, with the first one expanded to show details. The details panel on the right lists the alert ID, device ID, activity status, and activity time. It also includes a section for 'Additional properties' showing the remote address, port, and protocol.

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

Alert details

ID: 46a64295-3179-4f68-bc58-fc21c5ceac61

- IoT Device
 - Device ID: liya-x86-sim
- Process
- Additional properties
 - RemoteAddress: 10.168.98.110
 - RemotePort: 52033
 - Protocol: TCP

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>.