



Azure IoT Hub on Talaria TWO Platform

Getting Started Guide

Release: 11-20-2024

InnoPhase IoT, Inc.

2870 Zanker Road, Suite 200,

San Jose, CA 95134

innophaseiot.com

Platform	InnoPhase IoT Talaria TWO SDK
Device	Talaria TWO Evaluation Kit EVB-A (INP3010, INP3011, INP3012, INP3013)
Language	C

Contents

1	Document Information	4
1.1	Naming Conventions	4
1.2	Glossary	4
1.3	Revision History	4
1.4	Figures	5
1.5	Tables.....	5
2	Introduction	6
2.1	Introduction and Features of Talaria TWO Family of Devices	6
3	Step 1: Prerequisites	7
3.1	Prepare the Development Environment (IDE, Toolchain, SDK)	7
3.2	IoT Hub Setup	8
3.3	Setup Device Provisioning Service (DPS).....	8
3.4	Acquire a Talaria TWO INP301x EVB-A Development Kit	8
4	Step 2: Prepare the Device	9
4.1	Setting up Talaria TWO INP301x EVB-A Board.....	9
4.2	Programming the Device.....	9
5	Step 3: Build SDK and Run Samples	9
5.1	Azure IoT Hub Client Sample	10
5.2	Azure IoT HUB Device Twin and Direct Method Sample	10
5.3	Azure IoT Hub Device Provisioning Service Sample	10
6	Step 4: Integration with Azure IoT Explorer	11
6.1	Installing Azure IoT Explorer and Connecting it to a Hub	11
6.2	Interacting with EVB-A using Azure IoT Explorer	13
6.2.1	Monitoring Device Telemetry, Device2Cloud Messages.....	13
6.2.2	Sending Cloud to Device Messages.....	14
6.2.3	Viewing Device Twin.....	16
6.2.4	Updating Device Twin.....	19
6.2.5	Calling a Direct Method.....	22
7	Step 7: Additional Links.....	24
8	Support	25



9 Disclaimers 26

1 Document Information

1.1 Naming Conventions

Talaria TWO	Family of devices using the InnoPhase IoT ultra-low power wireless technology, including the Talaria TWO SoC and Talaria TWO Modules
Talaria TWO SoC	InnoPhase IoT custom wireless platform (INP2045)
Talaria TWO Modules	Modules integrating the Talaria TWO SoC (INP1010, INP1011, INP1012, INP1013)
Talaria TWO EVB-A	Evaluation boards containing the Talaria TWO Modules (INP3010, INP3011, INP3012, INP3013)

1.2 Glossary

API	Application Programming Interface
ELF	Executable and Linkable Format
EVB	Evaluation Board
GPIO	General Purpose Input/Output
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
MQTT	Message Queuing Telemetry Transport
SDK	Software Development Kit

1.3 Revision History

Version	Date	Description of Change
1.0	07-20-2023	First release.
1.1	12-18-2023	Link changes for README files as per new directory structure.
2.0	11-20-2024	Updated sample applications folder paths.

Table 1: Revision History

1.4 Figures

Figure 1: Connecting IoT Explorer to Hub and viewing the devices..... 11

Figure 2: Viewing device identity 12

Figure 3: Monitoring device telemetry. Device2Cloud messages..... 13

Figure 4: Sending Cloud2Device messages 14

Figure 5: Sending Cloud2Device messages, success pop-up in Azure IoT Explorer 14

Figure 6: Sending Cloud2Device messages, Talaria TWO Device Console 15

Figure 7: Default Device Twin JSON 16

Figure 8: Device Twin ‘reported’ properties update success from Talaria TWO Device Console.... 17

Figure 9: Device Twin ‘reported’ properties update success from Azure IoT Explorer 18

Figure 10: Updating Device Twin ‘desired’ properties from Azure IoT Explorer 20

Figure 11: Updating Device Twin ‘desired’ properties, success pop-up in Azure IoT Explorer 20

Figure 12: Updating Device Twin ‘desired’ properties, Talaria TWO Device Console 21

Figure 13: Invoking ‘Direct Method’ from Azure IoT Explorer 22

Figure 14: Invoking ‘Direct Method’, success pop-up in Azure IoT Explorer 22

Figure 15: Invoking ‘Direct Method’, success from Talaria TWO Device Console 23

1.5 Tables

Table 1: Revision History 4

2 Introduction

This document describes the procedure to connect to 'InnoPhase IoT Talaria TWO INP301x EVB-A' devices with Azure IoT Hub using Talaria TWO SDK. This multi-step process includes:

1. Configuring Azure IoT Hub
2. Registering the IoT device
3. Provisioning devices on Device Provisioning service
4. Build and deploy Azure IoT SDK on the device

2.1 Introduction and Features of Talaria TWO Family of Devices

The INP3010/3011/3012/3013 EVB-As are available for evaluating the performance and capability of the Talaria TWO INP1010/1011/1012/1013 modules.

The kits use InnoPhase IoT's award-winning Talaria TWO Multi-Protocol Platform with ultra-low power Wi-Fi + BLE5 for wireless data transfer, an embedded Arm Cortex-M3 for system control and user applications plus advanced security features for device safeguards.

The kits include an Arduino UNO format baseboard with a Talaria TWO module attached and a different antenna option per kit.

The EVB-A can be used in stand-alone mode or attached to an Arduino UNO compatible Host or Shield board. The baseboard has all module GPIOs accessible through either an internal 20-PIN header or the Arduino connectors. Power is supplied from USB, Host Arduino board or battery connector.

Also mounted on the baseboard are environmental sensors for capturing temperature, humidity, pressure and light. It is an ideal platform for developing exciting new battery-based, cloud-connected products such as smart locks, smart sensors, or security and health monitoring devices.

Product brief is available at: https://innophaseiot.com/wp-content/uploads/modules/INP3010_INP3011-EVB-A-Product-Brief.pdf

For more details, refer: <https://innophaseiot.com/talaria-two-modules/>

3 Step 1: Prerequisites

3.1 Prepare the Development Environment (IDE, Toolchain, SDK)

'Talaria TWO Software Development Kit' is used for developing applications on Talaria TWO Platform.

Talaria TWO SDK is available through 'InnoPhase IoT Customer Portal Access' (<https://innophaseiot.com/portal/customer-registration/>) and is available after portal registration, Mutual Non-Disclosure Agreement (MNDA) and Development Tools License Agreement (DTLA).

Talaria TWO SDK is packaged with the following:

1. User Guides for Development Environment Setup on a Host PC
2. SDK API Reference Manual
3. Application Notes
4. Reference Applications and Solution Ready Applications
5. Several Example Applications
6. Documents for the user to start the development targeting different use-cases

A comprehensive guide: `UG_Environment_Setup_for_Linux.pdf` (`sdk_x.y\doc\user_guides\ug_env_setup_linux`) is available covering the procedure of setting up the development environment for using Talaria TWO SDK on an Ubuntu VirtualBox based environment with a Windows 10 Host.

This document details the procedure for installing the toolchain and necessary software packages required for the development, CLI commands for building target executables, programming the target and debugging of an application.

Talaria TWO SDK also supports the development using an Eclipse-based IDE in Windows OS based PC. The details of setting up the development environment in Windows OS using the IDE is provided in user guide: `UG_Eclipse_Setup_Windows.pdf` (`sdk_x.y\doc\user_guides\ug_eclipse_setup_windows`).

`T2-RM001-Talaria TWO SDK API Reference Guide.pdf` provides details about all the SDK APIs available to the user.

All these tools and documents are available through the customer portal after registration through the customer portal link provided in this section.

3.2 IoT Hub Setup

Refer to Azure IoT Hub user guide to create and setup the Azure IoT Hub using the portal:

<https://docs.microsoft.com/azure/iot-hub/iot-hub-create-through-portal>

Creating and registering a new device identity using the portal is also explained in the same link.

Device-specific 'connection string' from this step will be used in a few of the sample application code.

3.3 Setup Device Provisioning Service (DPS)

The following link describes creating a new 'Device Provisioning Service' (DPS) and linking it to the IoT Hub created in previous step: <https://docs.microsoft.com/en-us/azure/iot-dps/quick-setup-auto-provision>

The Device Provisioning Service (DPS) 'Scope ID' and enrollment entity's 'Registration ID' will be used to run the Device Provisioning Service (DPS) sample application. DPS sample applications with both 'Symmetric Key' based attestation and 'X.509 CA' based attestation, are provided.

For 'Symmetric Key' based attestation, the enrollment entity's 'Symmetric Key' is also used for running the Sample Application.

For 'X.509 CA' based attestation, further instructions to generate the certificates and setup the enrollment are provided in the details with the Device Provisioning Service (DPS) Sample Application's document. These steps include uploading the rootCA cert to IoT Hub, performing proof of possession, for X.509 cert and creating enrollment entity on Device Provisioning Service (DPS) Portal.

3.4 Acquire a Talaria TWO INP301x EVB-A Development Kit

The Talaria TWO INP301x Dev Kits can be procured from the following distributors:

1. <https://www.mouser.com/manufacturer/innophase/>
2. <https://www.richardsonrfd.com/Products/Search?searchBox=innophase&instockonly=false>

4 Step 2: Prepare the Device

4.1 Setting up Talaria TWO INP301x EVB-A Board

A User Guide for setting up Talaria TWO EVB-A can be found here:

<https://innophaseiot.com/wp-content/uploads/modules/User-Guide-for-Talaria-TWO-EVB-A-Evaluation-Board.pdf>

This document includes information regarding a successful setup, including description of components, power supply requirements, details of jumpers and the driver needed and so on.

4.2 Programming the Device

Talaria TWO Download Tool is used for programming the EVB-A and using Debug Console. This tool is available for Windows and Linux platforms.

User Guide for this tool can be found here: <https://innophaseiot.com/wp-content/uploads/modules/Talaria-TWO-Download-Tool-User-Guide.pdf>

This tool can be downloaded from: <https://innophaseiot.com/talaria-two-modules#eval-software>

The Download Tool is found in the following folder in the Evaluation Software download from above link:

I-CUBE-T2-STW.zip\STM32CubeExpansion_T2-HostAPI-lib_V1.0\Utilities\PC_Software\TalariaTwo_DownloadTool\Tool_GUI

5 Step 3: Build SDK and Run Samples

Detailed instructions for cloning the repo for downloading sample device code, setting up the development environment, compiling, programming and running sample applications, are available here: https://github.com/InnoPhaseIoT/talaria_two_azure/blob/main/README.md

Sample applications covering the Device to Cloud, Cloud to Device, Device Twin, Direct Methods and Device Provisioning Service are available in the repository.

To connect the Talaria TWO EVB-A DevKit to Azure IoT Hub, the sample applications need to be modified in certain places for Azure IoT settings, rebuild the image, and provide the Wi-Fi AP setting as a boot argument while flashing the sample application image to the DevKit using the Download Tool.

Detailed instructions for building, programming and running individual sample applications are described in the subsequent sections.

5.1 Azure IoT Hub Client Sample

Sample application 'iothub_client_sample_mqtt' demonstrates Device to Cloud (D2C) and Cloud to Device (C2D) functionality using Azure IoT Hub.

Detailed instructions are provided in the following link for setting this up and running the sample on EVB-A to achieve these functionalities:

https://github.com/InnoPhaseIoT/talaria_two_azure/blob/main/examples/sdk_2.x/iothub_client_sample_mqtt/README.md

5.2 Azure IoT HUB Device Twin and Direct Method Sample

Sample Application 'iothub_client_device_twin_and_methods_sample' demonstrates Device Twin and Direct Methods functionalities of Azure IoT Hub.

Detailed instructions are provided in the following link for setting this up and running the sample on EVB-A to achieve these functionalities:

https://github.com/InnoPhaseIoT/talaria_two_azure/blob/main/examples/sdk_2.x/iothub_device_twin_and_methods_sample/README.md

5.3 Azure IoT Hub Device Provisioning Service Sample

Sample Application 'prov_dev_client_ll_sample' demonstrates the working of Talaria TWO devices with Device Provisioning Service (DPS) for Azure IoT Hub.

Detailed instructions are provided in the link below for setting this up and running the sample on EVB-A to achieve the provisioning using Device Provisioning Service (DPS):

https://github.com/InnoPhaseIoT/talaria_two_azure/blob/main/examples/sdk_2.x/prov_dev_client_ll_sample/README.md

6 Step 4: Integration with Azure IoT Explorer

6.1 Installing Azure IoT Explorer and Connecting it to a Hub

1. Go to Azure IoT explorer releases and expand the list of assets for the most recent release: <https://github.com/Azure/azure-iot-explorer/releases>
2. Download and install the most recent version of the application.
3. Connect Azure IoT Explorer to IoT Hub by providing your IoT Hub's connection string. This can be found from the Azure Portal: Click on your IoT Hub > Shared access policies > iothubowner > connection string-primary key > Copy to clipboard
4. On launching Azure IoT Explorer, click on "Add connection" > paste the Connection String > Save.
5. Once the IoT Hub connection has been added, go to the specific IoT Hub and click on "View devices in this hub". Choose the appropriate device.

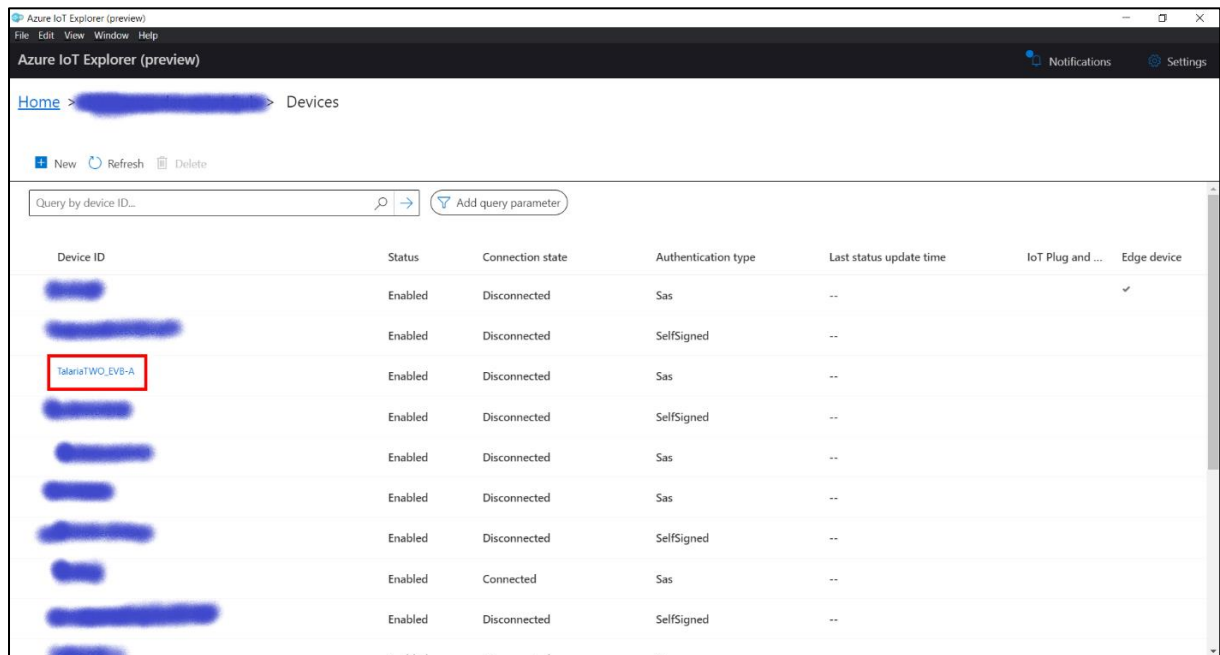


Figure 1: Connecting IoT Explorer to Hub and viewing the devices

6. Select the device and click on Device identity which provides information on the device.

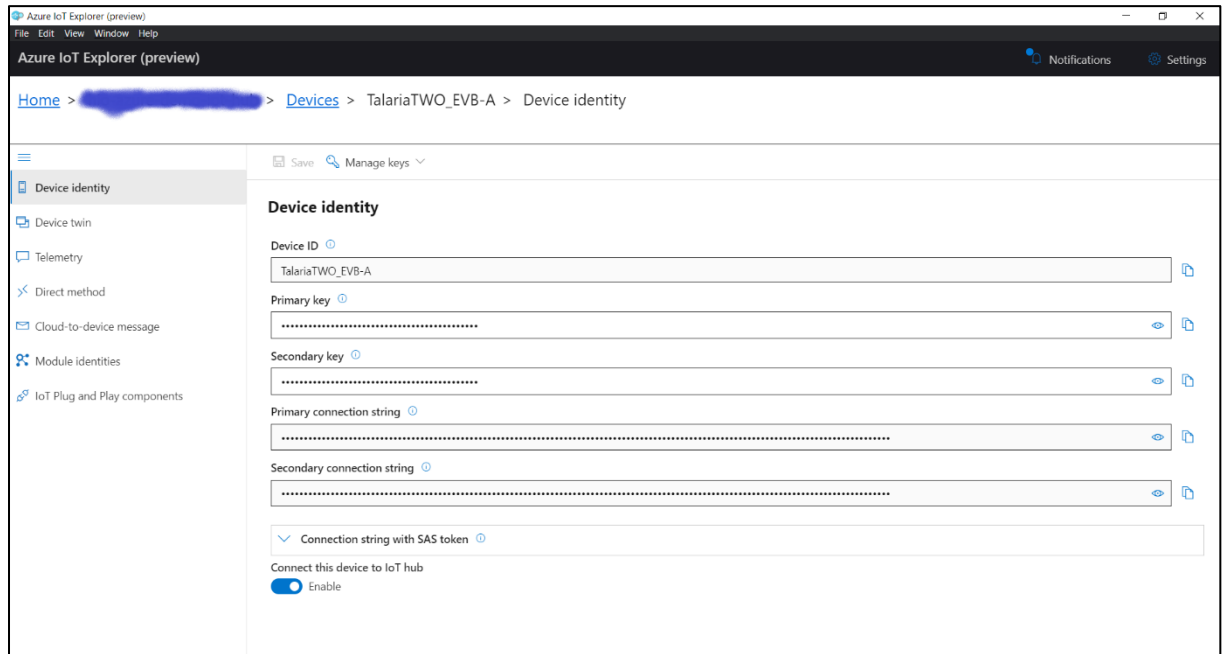


Figure 2: Viewing device identity

6.2 Interacting with EVB-A using Azure IoT Explorer

Azure IoT Explorer can be used to monitor and interact with the device using various options available on the left navigation panel. All the applicable options working with Talaria TWO EVB-A are explained in detail in the subsequent sections.

6.2.1 Monitoring Device Telemetry, Device2Cloud Messages

With Azure IoT Explorer, the flow of telemetry from the device to the cloud can be viewed. To do this, first prepare and boot the [Azure IoT Hub Client Sample](#) application.

In Azure IoT Explorer select 'Telemetry' option on left panel and confirm that 'Use built-in event hub' option is set to yes. Then click 'Start' to see telemetry as the device sends messages to the cloud.

This is shown in the screenshot below:

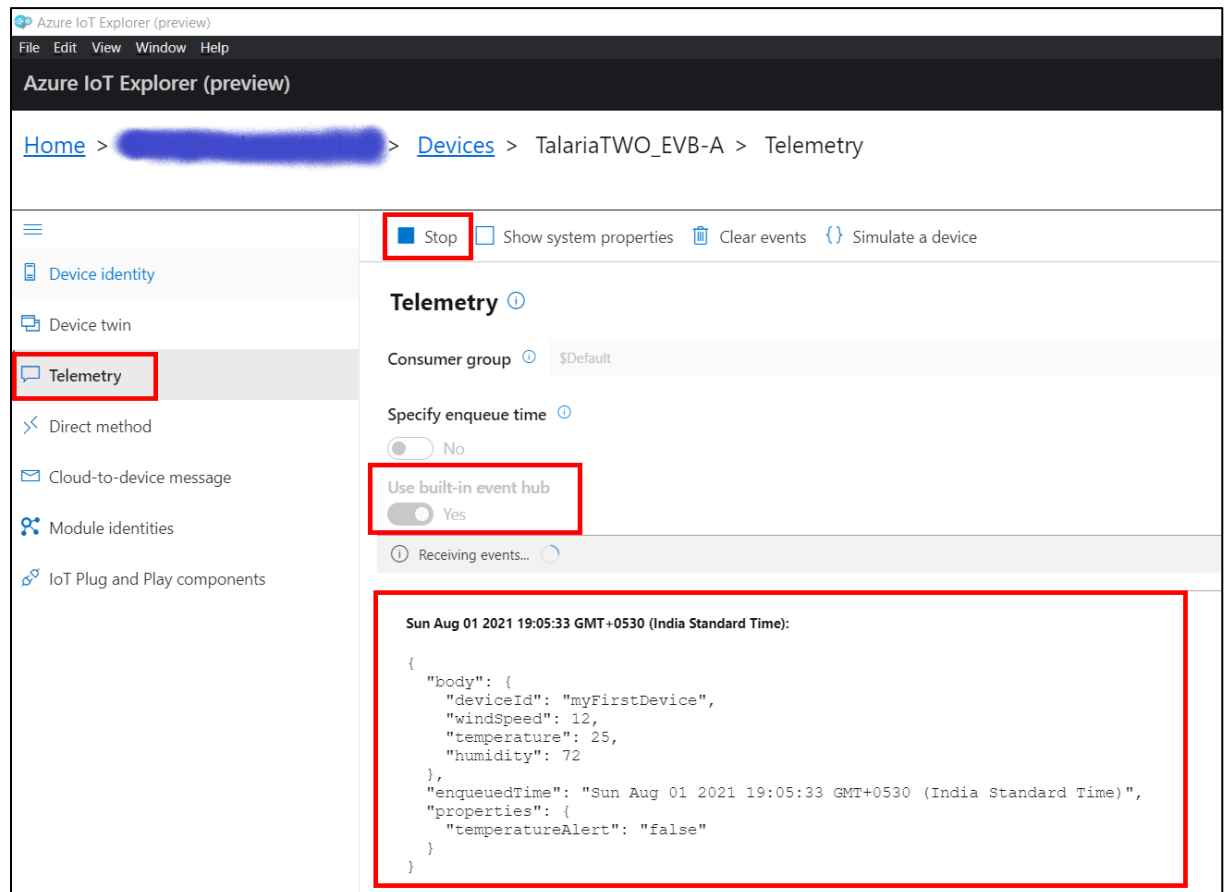


Figure 3: Monitoring device telemetry. Device2Cloud messages

6.2.2 Sending Cloud to Device Messages

Prepare and boot the [Azure IoT Hub Client Sample](#) application.

In Azure IoT Explorer select 'Cloud-to-device message' option on the left panel and give a message body of your choice.

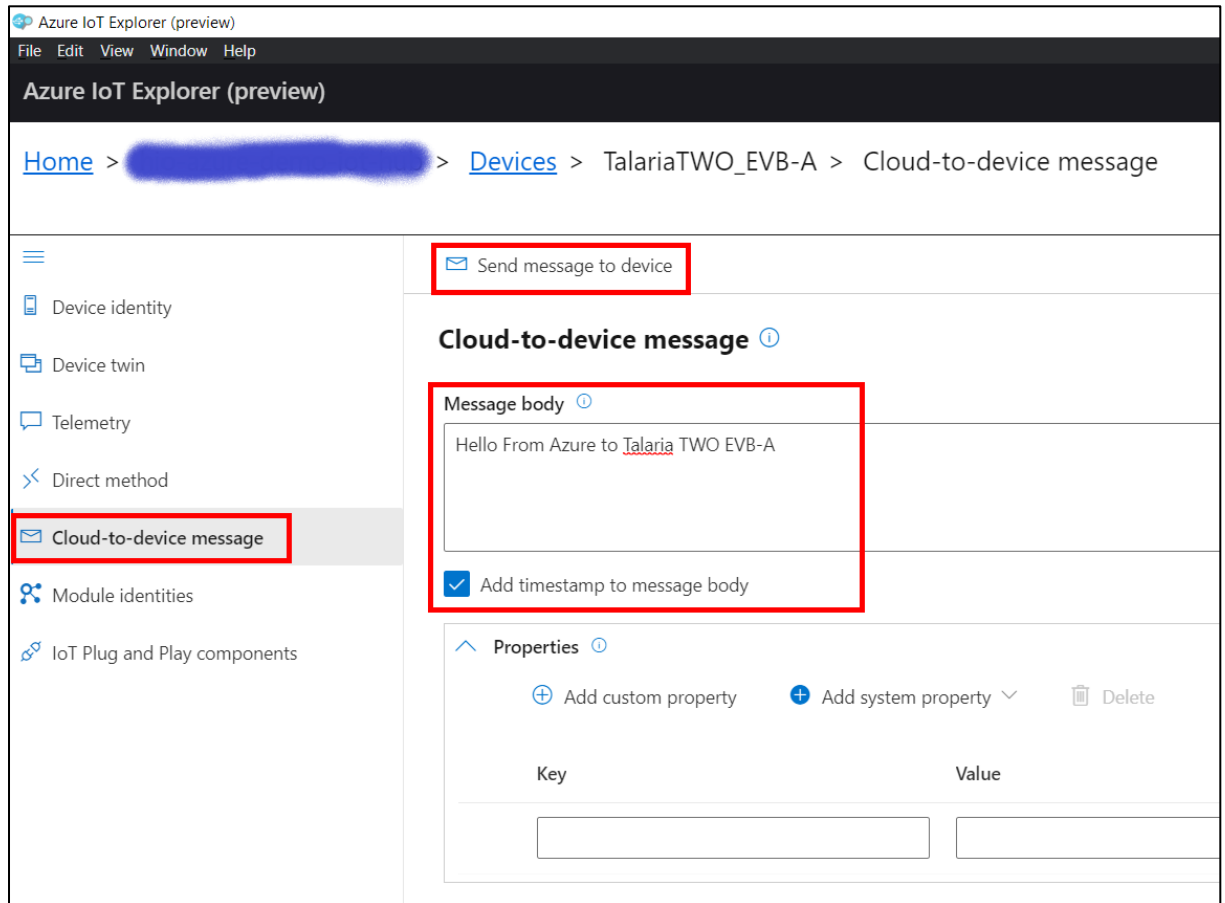


Figure 4: Sending Cloud2Device messages

Then press 'Send message to device' button. The following notification is received.

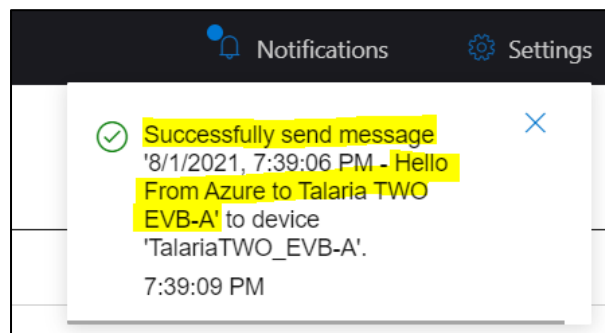


Figure 5: Sending Cloud2Device messages, success pop-up in Azure IoT Explorer

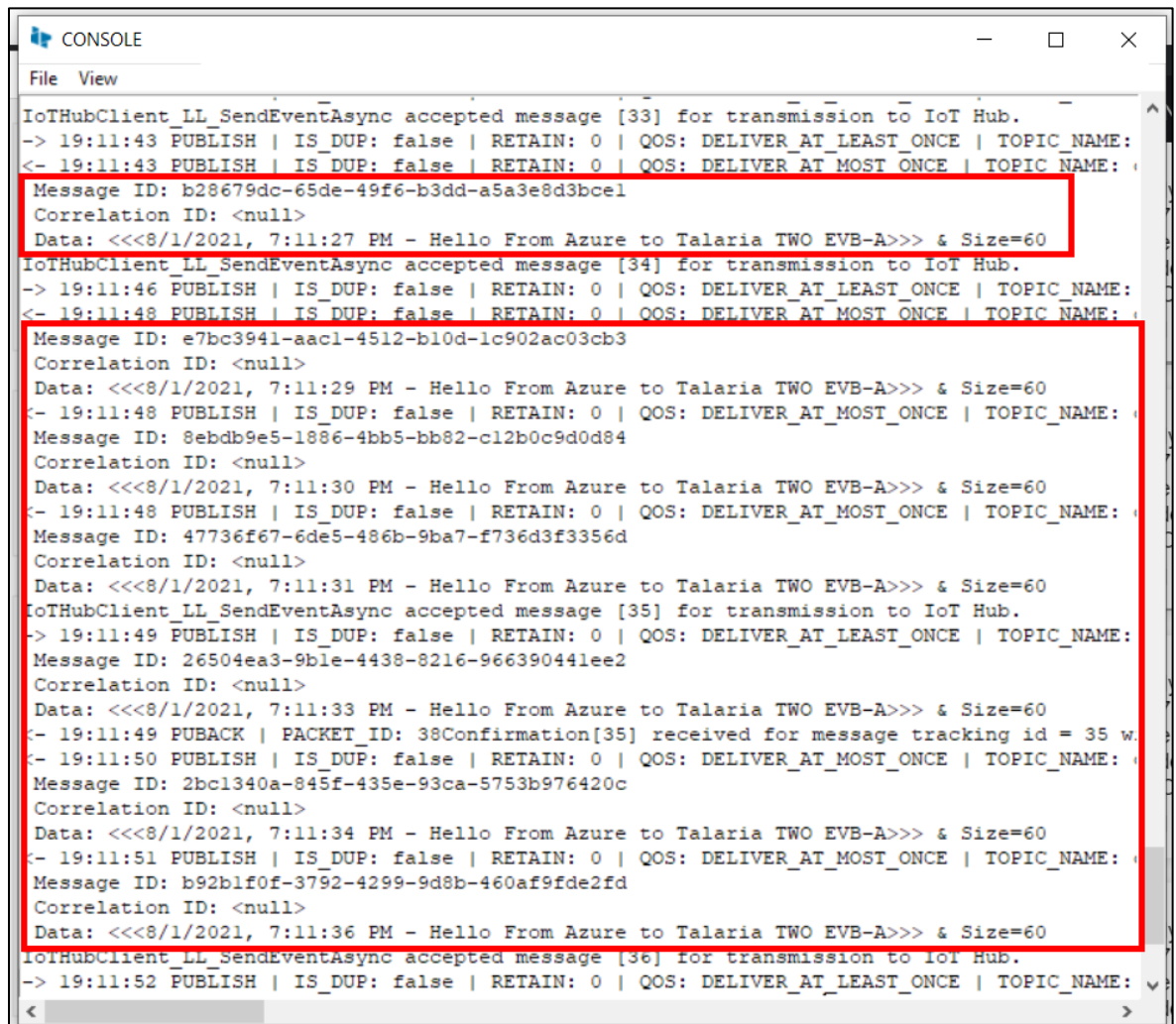
Expected output on Talaria TWO EVB-A console:

```
--
Message ID: 26504ea3-9b1e-4438-8216-966390441ee2

Correlation ID: <null>

Data: <<<8/1/2021, 7:11:33 PM - Hello From Azure to Talaria TWO EVB-A>>> & Size=60 --
```

After sending the messages multiple times from the IoT Explorer, the screenshot from the device console is shown below:



```
CONSOLE
File View
IoTHubClient_LL_SendEventAsync accepted message [33] for transmission to IoT Hub.
-> 19:11:43 PUBLISH | IS_DUP: false | RETAIN: 0 | QOS: DELIVER_AT_LEAST_ONCE | TOPIC_NAME:
<- 19:11:43 PUBLISH | IS_DUP: false | RETAIN: 0 | QOS: DELIVER_AT_MOST_ONCE | TOPIC_NAME:
Message ID: b28679dc-65de-49f6-b3dd-a5a3e8d3bce1
Correlation ID: <null>
Data: <<<8/1/2021, 7:11:27 PM - Hello From Azure to Talaria TWO EVB-A>>> & Size=60
IoTHubClient_LL_SendEventAsync accepted message [34] for transmission to IoT Hub.
-> 19:11:46 PUBLISH | IS_DUP: false | RETAIN: 0 | QOS: DELIVER_AT_LEAST_ONCE | TOPIC_NAME:
<- 19:11:48 PUBLISH | IS_DUP: false | RETAIN: 0 | QOS: DELIVER_AT_MOST_ONCE | TOPIC NAME:
Message ID: e7bc3941-aac1-4512-b10d-1c902ac03cb3
Correlation ID: <null>
Data: <<<8/1/2021, 7:11:29 PM - Hello From Azure to Talaria TWO EVB-A>>> & Size=60
<- 19:11:48 PUBLISH | IS_DUP: false | RETAIN: 0 | QOS: DELIVER_AT_MOST_ONCE | TOPIC_NAME:
Message ID: 8ebdb9e5-1886-4bb5-bb82-c12b0c9d0d84
Correlation ID: <null>
Data: <<<8/1/2021, 7:11:30 PM - Hello From Azure to Talaria TWO EVB-A>>> & Size=60
<- 19:11:48 PUBLISH | IS_DUP: false | RETAIN: 0 | QOS: DELIVER_AT_MOST_ONCE | TOPIC NAME:
Message ID: 47736f67-6de5-486b-9ba7-f736d3f3356d
Correlation ID: <null>
Data: <<<8/1/2021, 7:11:31 PM - Hello From Azure to Talaria TWO EVB-A>>> & Size=60
IoTHubClient_LL_SendEventAsync accepted message [35] for transmission to IoT Hub.
-> 19:11:49 PUBLISH | IS_DUP: false | RETAIN: 0 | QOS: DELIVER_AT_LEAST_ONCE | TOPIC NAME:
Message ID: 26504ea3-9b1e-4438-8216-966390441ee2
Correlation ID: <null>
Data: <<<8/1/2021, 7:11:33 PM - Hello From Azure to Talaria TWO EVB-A>>> & Size=60
<- 19:11:49 PUBACK | PACKET_ID: 38Confirmation[35] received for message tracking id = 35 w
<- 19:11:50 PUBLISH | IS_DUP: false | RETAIN: 0 | QOS: DELIVER_AT_MOST_ONCE | TOPIC NAME:
Message ID: 2bc1340a-845f-435e-93ca-5753b976420c
Correlation ID: <null>
Data: <<<8/1/2021, 7:11:34 PM - Hello From Azure to Talaria TWO EVB-A>>> & Size=60
<- 19:11:51 PUBLISH | IS_DUP: false | RETAIN: 0 | QOS: DELIVER_AT_MOST_ONCE | TOPIC NAME:
Message ID: b92b1f0f-3792-4299-9d8b-460af9fde2fd
Correlation ID: <null>
Data: <<<8/1/2021, 7:11:36 PM - Hello From Azure to Talaria TWO EVB-A>>> & Size=60
IoTHubClient_LL_SendEventAsync accepted message [36] for transmission to IoT Hub.
-> 19:11:52 PUBLISH | IS_DUP: false | RETAIN: 0 | QOS: DELIVER_AT_LEAST_ONCE | TOPIC NAME:
```

Figure 6: Sending Cloud2Device messages, Talaria TWO Device Console

6.2.3 Viewing Device Twin

In Azure IoT Explorer, select 'Device Twin' option on the left panel.

If the device was created using the portal, by default the Device Twin JSON will look like as shown in Figure 7 (without any values in reported and desired properties):

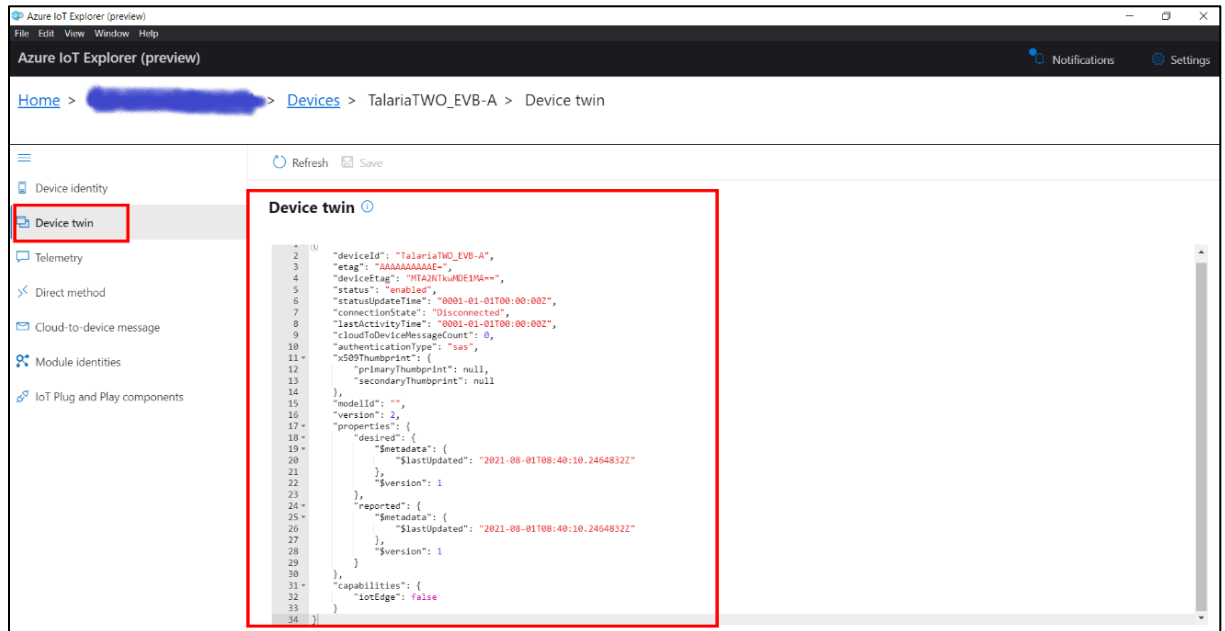


Figure 7: Default Device Twin JSON

Now, prepare and boot the [Azure IoT HUB Device Twin and Direct Method Sample](#) application.

Expected output:

```

--
'Device Twin reported properties update completed with result: 204'
--
    
```

```
CONSOLE
File View

Y-BOOT 208ef13 2019-07-22 12:26:54 -0500 790dal-b-7
ROM yoda-h0-rom-16-0-gd5a8e586
FLASH:PNWWWWAEBuild $Patch: git-97ea9fecf $ $Id: git-bfddb0922 $
ssid=[REDACTED] passphrase=[REDACTED] vm.ways=8
[0.141,859] No VM flash location is specified, using default 0x40000

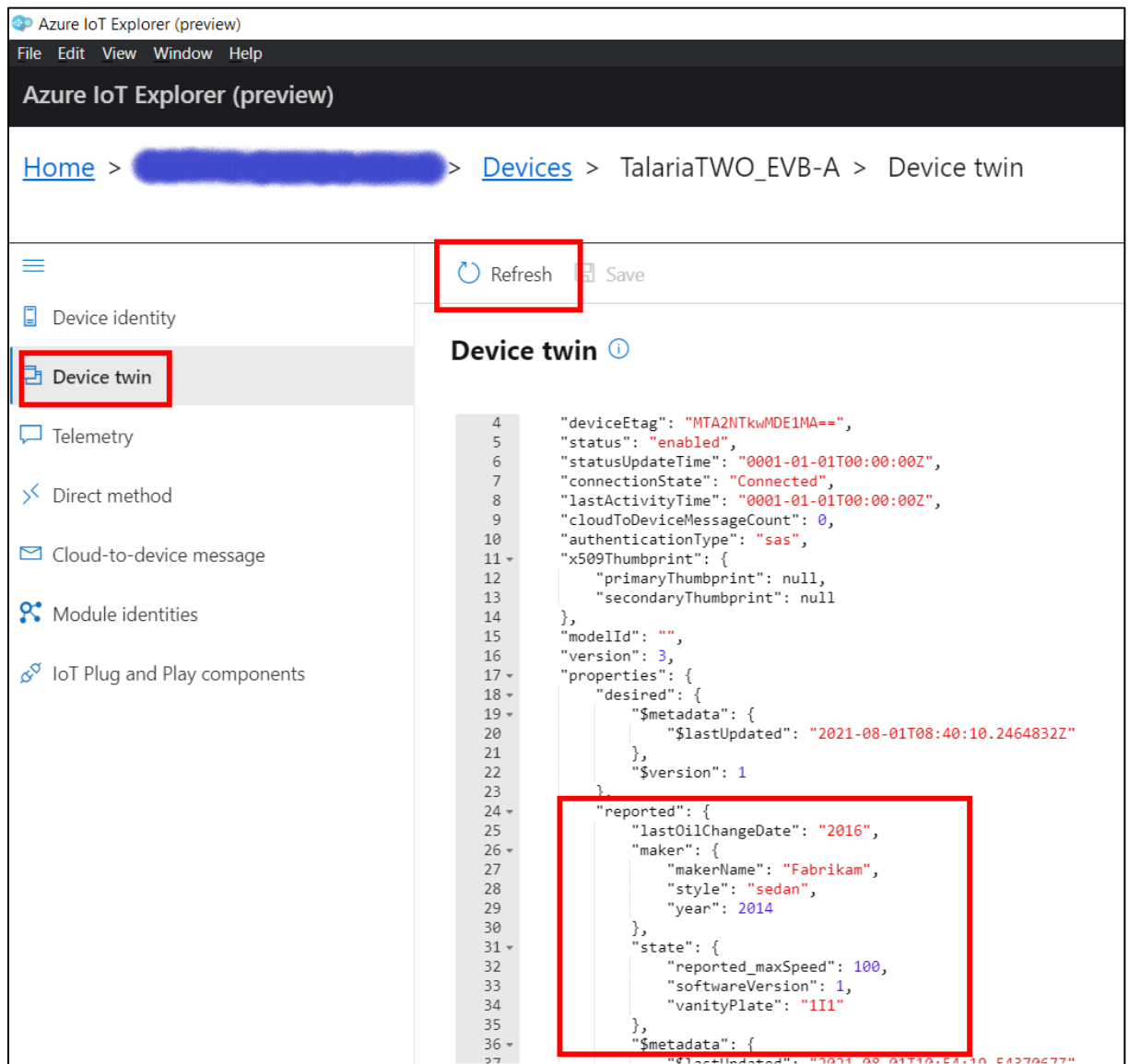
WiFi Details SSID: novid20, PASSWORD: shxp3496
Calibrating.....Done
addr 02:03:04:30:ae:dd
Connecting to WiFi...
add network status: 0
added network successfully, will try connecting..
connecting to network status: 0
[6.233,946] CONNECT:d6:8a:39:2d:db:50 Channel:11 rssi:-47 dBm
wcm_notify_cb to App Layer - WCM_NOTIFY_MSG_LINK_UP
wcm_notify_cb to App Layer - WCM_NOTIFY_MSG_ADDRESS
[8.875,414] MYIP 192.168.43.59
[8.875,494] IPv6 [fe80::3:4ff:fe30:aedd]-link
wcm_notify_cb to App Layer - WCM_NOTIFY_MSG_CONNECTED
starting app_thread

sntp_process: Sun Aug 1 16:24:16 202

Root Done[0]Loading the client cert. and key. size tls:4
. Connecting to hio-azure-demo-iot-hub.azure-devices.net/8883... ok
. Setting up the SSL/TLS structure... This certificate has no flags
This certificate has no flags
This certificate has no flags
SSL/TLS handshake. DONE ..ret:0
ok
[ Protocol is TLSv1.2 ]
[ Ciphersuite is TLS-ECDHE-RSA-WITH-AES-128-CBC-SHA256 ]
[ Record expansion is 69 ]
. Verifying peer X.509 certificate...
ok
-> 16:24:32 CONNECT | VER: 4 | KEEPALIVE: 240 | FLAGS: 192 | USERNAME: [REDACTED]
Device Twin reported properties update completed with result: 204
<- 16:24:33 SUBACK | PACKET_ID: 5 | RETURN_CODE: 0
```

Figure 8: Device Twin 'reported' properties update success from Talaria TWO Device Console

In Azure IoT Explorer, refreshing the Device Twin, the 'reported' values from running the application are now visible in the Device Twin JSON:



The screenshot shows the Azure IoT Explorer interface. The breadcrumb navigation is: Home > [redacted] > Devices > TalariaTWO_EVB-A > Device twin. On the left sidebar, 'Device twin' is selected. In the top right of the main area, a 'Refresh' button is highlighted with a red box. The main area displays the Device twin JSON. A red box highlights the 'reported' properties section, which contains the following data:

```

{
  "lastOilChangeDate": "2016",
  "maker": {
    "makerName": "Fabrikam",
    "style": "sedan",
    "year": 2014
  },
  "state": {
    "reported_maxSpeed": 100,
    "softwareVersion": 1,
    "vanityPlate": "1I1"
  },
  "$metadata": {
    "$lastUpdated": "2021-08-01T08:40:10.2464832Z"
  }
}

```

Figure 9: Device Twin 'reported' properties update success from Azure IoT Explorer

6.2.4 Updating Device Twin

The desired section of the Device Twin JSON will be empty. In Azure IoT Explorer, copy and paste the parts from the following JSON blob under `desired` property:

```
--  
  
    "desired": {  
        "changeOilReminder": "LOW_OIL",  
        "settings": {  
            "desired_maxSpeed": 126,  
            "location": {  
                "longitude": 72000000,  
                "latitude": 26000000  
            }  
        },  
        "$metadata": {  
            "$lastUpdated": "2021-06-26T00:06:52.9307518Z"  
        },  
        "$version": 1  
    },  
  
--
```

Note: The `$metadata` part does not need to be copied, this is just for reference as for how the final 'desired' section will look like.

The final desired part should look like as shown in Figure 10:

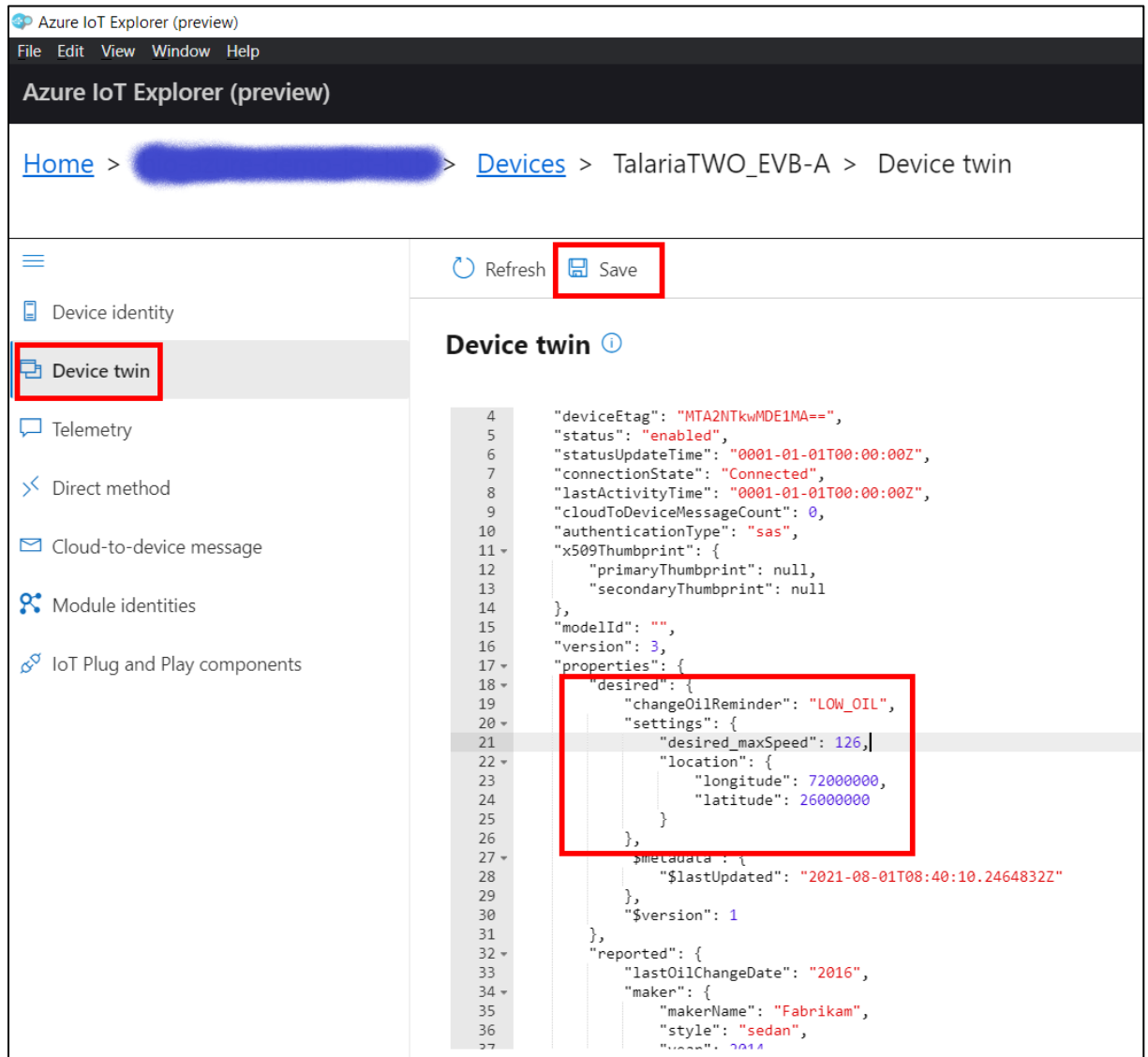


Figure 10: Updating Device Twin 'desired' properties from Azure IoT Explorer

Click 'Save' which will provide the following notification:

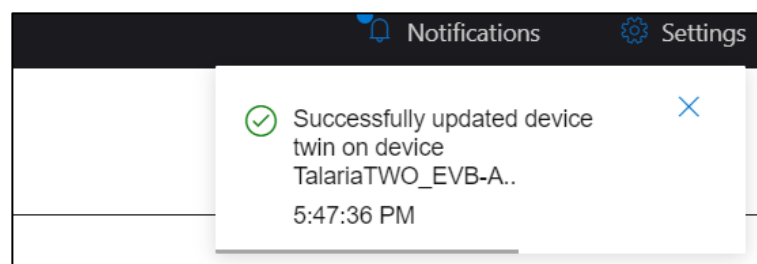
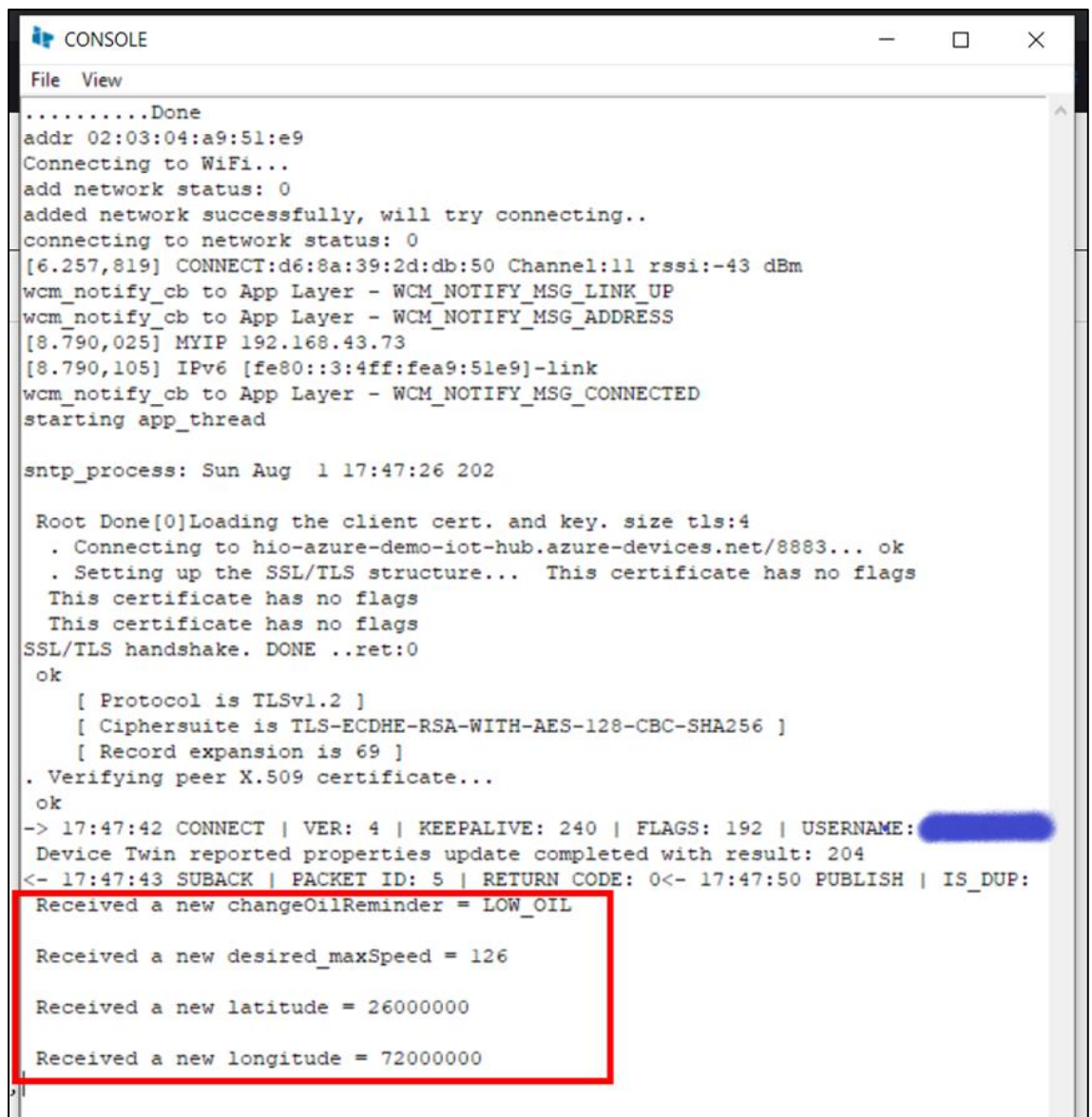


Figure 11: Updating Device Twin 'desired' properties, success pop-up in Azure IoT Explorer

Expected output on Talaria TWO EVB-A console:

```
--
Received a new changeOilReminder = LOW_OIL
Received a new desired_maxSpeed = 126
Received a new latitude = 26000000
Received a new longitude = 72000000
--
```



```
.....Done
addr 02:03:04:a9:51:e9
Connecting to WiFi...
add network status: 0
added network successfully, will try connecting..
connecting to network status: 0
[6.257,819] CONNECT:d6:8a:39:2d:db:50 Channel:11 rssi:-43 dBm
wcm_notify_cb to App Layer - WCM_NOTIFY_MSG_LINK_UP
wcm_notify_cb to App Layer - WCM_NOTIFY_MSG_ADDRESS
[8.790,025] MYIP 192.168.43.73
[8.790,105] IPv6 [fe80::3:4ff:fea9:51e9]-link
wcm_notify_cb to App Layer - WCM_NOTIFY_MSG_CONNECTED
starting app_thread

sntp_process: Sun Aug  1 17:47:26 202

Root Done[0]Loading the client cert. and key. size tls:4
. Connecting to hio-azure-demo-iot-hub.azure-devices.net/8883... ok
. Setting up the SSL/TLS structure... This certificate has no flags
This certificate has no flags
This certificate has no flags
SSL/TLS handshake. DONE ..ret:0
ok
[ Protocol is TLSv1.2 ]
[ Ciphersuite is TLS-ECDHE-RSA-WITH-AES-128-CBC-SHA256 ]
[ Record expansion is 69 ]
. Verifying peer X.509 certificate...
ok
-> 17:47:42 CONNECT | VER: 4 | KEEPALIVE: 240 | FLAGS: 192 | USERNAME: [REDACTED]
Device Twin reported properties update completed with result: 204
<- 17:47:43 SUBACK | PACKET ID: 5 | RETURN CODE: 0<- 17:47:50 PUBLISH | IS_DUP:
Received a new changeOilReminder = LOW_OIL
Received a new desired_maxSpeed = 126
Received a new latitude = 26000000
Received a new longitude = 72000000
```

Figure 12: Updating Device Twin 'desired' properties, Talaria TWO Device Console

6.2.5 Calling a Direct Method

Prepare and boot the [Azure IoT HUB Device Twin and Direct Method Sample](#) application.

In Azure IoT Explorer, select 'Direct Method' option on left device panel, and use method name 'getCarVIN' and choose a payload string.

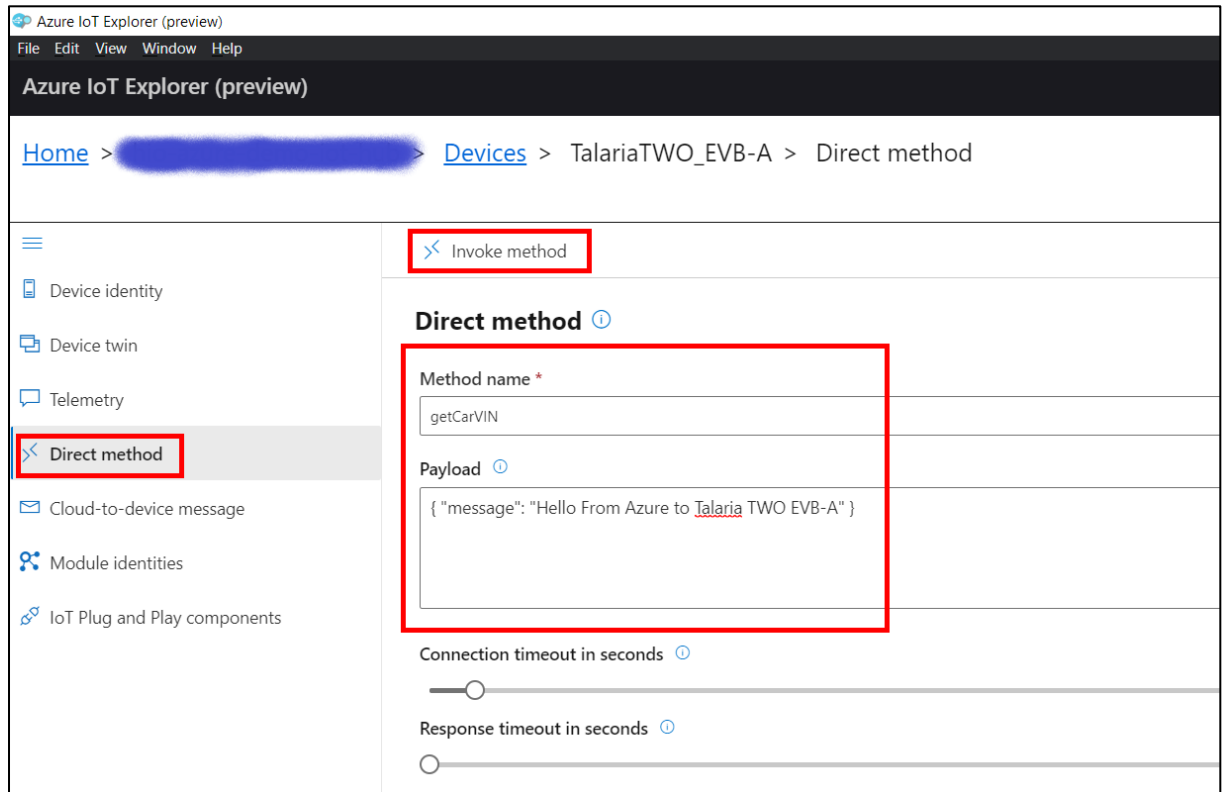


Figure 13: Invoking 'Direct Method' from Azure IoT Explorer

Then press 'Invoke method' which will provide the following notification:

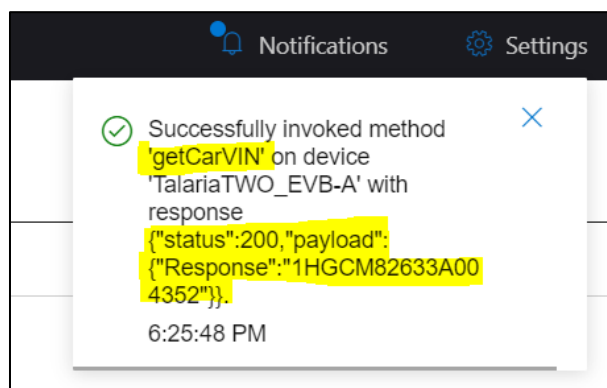
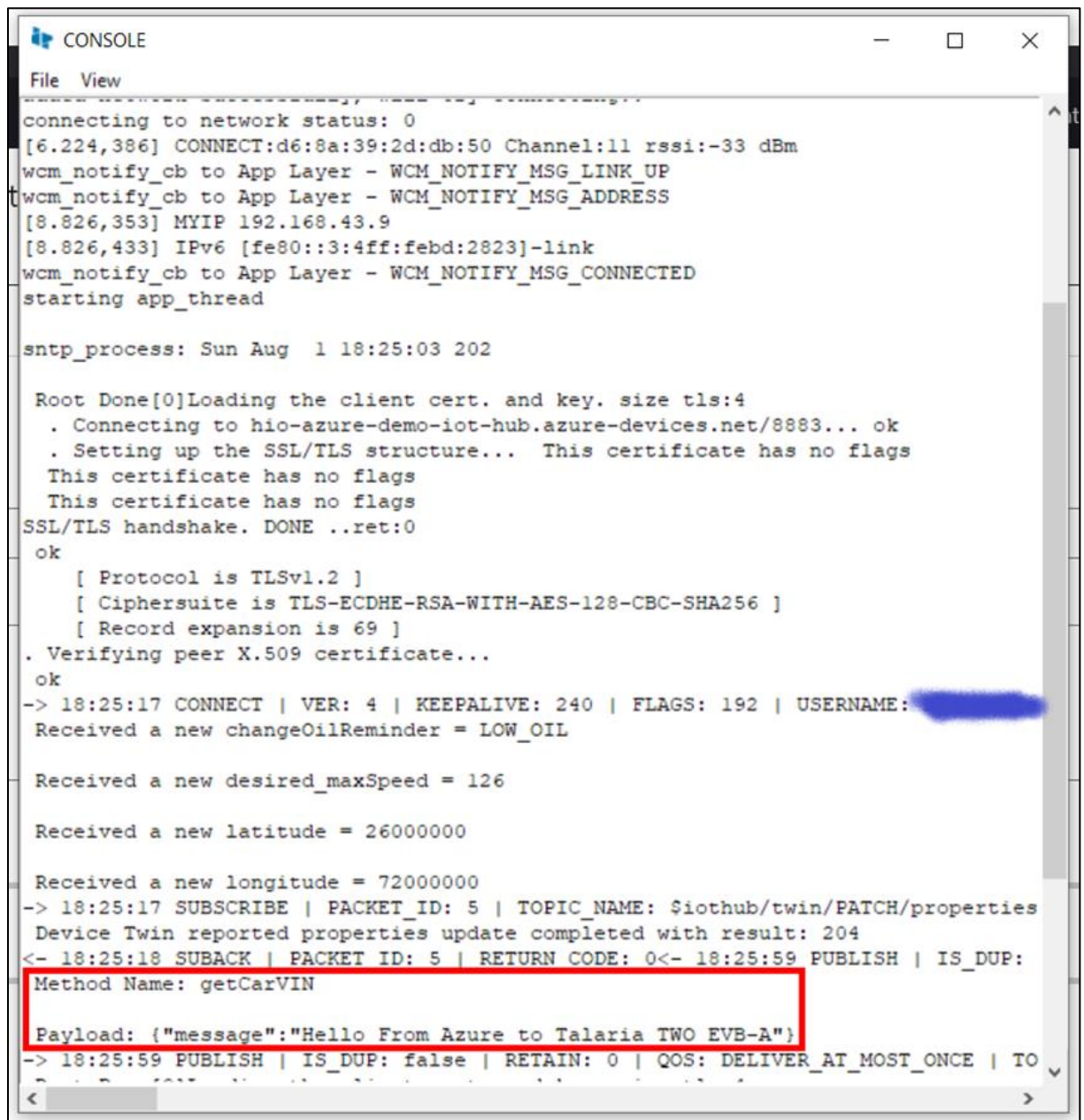


Figure 14: Invoking 'Direct Method', success pop-up in Azure IoT Explorer

Expected output on Talaria TWO EVB-A console:

```
--
Method Name: getCarVIN
Payload: {"message":"Hello From Azure to Talaria TWO EVB-A"}
--
```



```

CONSOLE
File View
connecting to network status: 0
[6.224,386] CONNECT:d6:8a:39:2d:db:50 Channel:11 rssi:-33 dBm
wcm_notify_cb to App Layer - WCM_NOTIFY_MSG_LINK_UP
wcm_notify_cb to App Layer - WCM_NOTIFY_MSG_ADDRESS
[8.826,353] MYIP 192.168.43.9
[8.826,433] IPv6 [fe80::3:4ff:febd:2823]-link
wcm_notify_cb to App Layer - WCM_NOTIFY_MSG_CONNECTED
starting app_thread

sntp_process: Sun Aug  1 18:25:03 202

Root Done[0]Loading the client cert. and key. size tls:4
. Connecting to hio-azure-demo-iot-hub.azure-devices.net/8883... ok
. Setting up the SSL/TLS structure... This certificate has no flags
This certificate has no flags
This certificate has no flags
SSL/TLS handshake. DONE ..ret:0
ok
[ Protocol is TLSv1.2 ]
[ Ciphersuite is TLS-ECDHE-RSA-WITH-AES-128-CBC-SHA256 ]
[ Record expansion is 69 ]
. Verifying peer X.509 certificate...
ok
-> 18:25:17 CONNECT | VER: 4 | KEEPALIVE: 240 | FLAGS: 192 | USERNAME: [REDACTED]
Received a new changeOilReminder = LOW_OIL

Received a new desired_maxSpeed = 126

Received a new latitude = 26000000

Received a new longitude = 72000000
-> 18:25:17 SUBSCRIBE | PACKET_ID: 5 | TOPIC_NAME: $iothub/twin/PATCH/properties
Device Twin reported properties update completed with result: 204
<- 18:25:18 SUBACK | PACKET ID: 5 | RETURN CODE: 0<- 18:25:59 PUBLISH | IS_DUP:
Method Name: getCarVIN
Payload: {"message":"Hello From Azure to Talaria TWO EVB-A"}
-> 18:25:59 PUBLISH | IS_DUP: false | RETAIN: 0 | QOS: DELIVER_AT_MOST_ONCE | TO

```

Figure 15: Invoking 'Direct Method', success from Talaria TWO Device Console

7 Step 7: Additional Links

- Manage cloud device messaging with Azure-IoT-Explorer
<https://github.com/Azure/azure-iot-explorer/releases>
- How to use IoT Explorer to interact with the device
<https://docs.microsoft.com/en-us/azure/iot-pnp/howto-use-iot-explorer#install-azure-iot-explorer>
- Schematics of the kit can be downloaded from the link below:
https://innophaseiot.com/wp-content/uploads/modules/INP3010_3011-Schematic.pdf
- Additional documentation is available on the Talaria TWO modules webpage:
<https://innophaseiot.com/talaria-two-modules#doc>
- Data Sheet for the modules used in the kit can be downloaded from the link below:
https://innophaseiot.com/wp-content/uploads/modules/INP101_INP1011-Talaria-TWO-Modules-Datasheet.pdf

8 Support

1. Sales Support: Contact an InnoPhase IoT sales representative via email – sales@innophaseiot.com
2. Technical Support:
 - a. Visit: <https://innophaseiot.com/support/>
 - b. Also Visit: <https://innophaseiot.com/talaria-two-modules>
 - c. Contact: support@innophaseiot.com

InnoPhase IoT is working diligently to provide outstanding support to all customers.

9 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, InnoPhase IoT Incorporated does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and assumes no liability associated with the use of such information. InnoPhase IoT Incorporated takes no responsibility for the content in this document if provided by an information source outside of InnoPhase IoT Incorporated.

InnoPhase IoT Incorporated disclaims liability for any indirect, incidental, punitive, special or consequential damages associated with the use of this document, applications and any products associated with information in this document, whether or not such damages are based on tort (including negligence), warranty, including warranty of merchantability, warranty of fitness for a particular purpose, breach of contract or any other legal theory. Further, InnoPhase IoT Incorporated accepts no liability and makes no warranty, express or implied, for any assistance given with respect to any applications described herein or customer product design, or the application or use by any customer's third-party customer(s).

Notwithstanding any damages that a customer might incur for any reason whatsoever, InnoPhase IoT Incorporated' aggregate and cumulative liability for the products described herein shall be limited in accordance with the Terms and Conditions of identified in the commercial sale documentation for such InnoPhase IoT Incorporated products.

Right to make changes — InnoPhase IoT Incorporated reserves the right to make changes to information published in this document, including, without limitation, changes to any specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — InnoPhase IoT Incorporated products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an InnoPhase IoT Incorporated product can reasonably be expected to result in personal injury, death or severe property or environmental damage. InnoPhase IoT Incorporated and its suppliers accept no liability for inclusion and/or use of InnoPhase IoT Incorporated products in such equipment or applications and such inclusion and/or use is at the customer's own risk.'

All trademarks, trade names and registered trademarks mentioned in this document are property of their respective owners and are hereby acknowledged.