|  |  |
| --- | --- |
| Technical Documentation | |
| Project name | iCon |
| Project number | 10665tco21 |
| Date | 2021-12-17 |

Spis treści

[1 Device Provisioning 2](#_Toc99543081)

[1.1 Device registration 2](#_Toc99543082)

[1.1.1 Benefits 2](#_Toc99543083)

[1.1.2 Description 2](#_Toc99543084)

[1.1.3 X.509 Attestation 2](#_Toc99543085)

[1.1.4 Create the device registration 2](#_Toc99543086)

[1.2 Create and register a provisioning client 3](#_Toc99543087)

[2 Connection and message exchange 3](#_Toc99543088)

[2.1 Create a IoT Hub client 3](#_Toc99543089)

[2.2 Cloud to device 3](#_Toc99543090)

[2.3 Device to cloud 3](#_Toc99543091)

[3 Messages (API) 3](#_Toc99543092)

[3.1 The ICoN sends static device details 3](#_Toc99543093)

[3.2 Device status 4](#_Toc99543094)

[*3.2.1* Topic: *$iothub/devices/icon-device-id/messages/events* 4](#_Toc99543095)

[3.3 Device location 5](#_Toc99543096)

[3.3.1 Data needed to display device location 5](#_Toc99543097)

[The ICoN informs the cloud about its location info (send message). The JSON is defined in the section 2.3.3 5](#_Toc99543098)

[3.3.2 Set device location 5](#_Toc99543099)

[The Cloud changes location of the ICoN. The JSON is defined in the section 2.3.3 5](#_Toc99543100)

[3.3.3 JSON data 5](#_Toc99543101)

[3.4 Device schedule. 5](#_Toc99543102)

[3.4.1 Report weekly schedule by the ICoN 5](#_Toc99543103)

[The ICoN sends weekly schedule by the ICoN to the cloud. . The JSON is defined in the section 2.4.3 5](#_Toc99543104)

[3.4.2 Set device (fan) schedule 5](#_Toc99543105)

[3.4.3 JSON data 5](#_Toc99543106)

[3.5 Get device mode (command to the device) 6](#_Toc99543107)

[3.6 Service setting 6](#_Toc99543108)

[3.7 Firmware update 7](#_Toc99543109)

[4 Data types 7](#_Toc99543110)

[5 Document history 8](#_Toc99543111)

# Device Provisioning

## Device registration

The IoT Hub Device Provisioning Service is a helper service for IoT Hub that enables zero-touch, just-in-time provisioning to the right IoTHub without requiring human intervention, enabling customers to provision millions of devices in a secure and scalable manner.

### Benefits

There are many provisioning scenarios in which the Device Provisioning Service is an excellent choice for getting devices connected and configured to IoTHub, such as:

1. Zero-touch provisioning to a single IoT solution without hardcoding IoT Hub connection information at the factory (initial setup).
2. Allows easy, secure and automatic setup of a large number of devices as soon as they are online.
3. Allows secure encrypted configuration communication between the cloud and the device from the first time the device is online.
4. Load balancing devices across multiple hubs.
5. Connecting devices to their owner’s IoT solution based on sales transaction data (multitenancy).
6. Connecting devices to a particular IoT solution depending on use-case (solution isolation).
7. Connecting a device to the IoT hub with the lowest latency (geo-sharding).
8. Reprovisioning based on a change in the device.
9. Rolling the keys used by the device to connect to IoT Hub (when not using X.509 certificates to connect).
10. Building an enrollment list containing the complete record of devices/groups of devices that may at some point register.

### Description

Each manufactured device will register itself to the IoT Hub trough the IoT Hub Device Provisioning Service on initial boot up. To do that Microsoft.Azure.Devices.Provisioning.Transport, Microsoft.Azure.Devices.Provisioning.Service & Microsoft.Azure.Devices.Provisioning.Client SDK’s will be used.

For more information: <https://docs.microsoft.com/en-us/azure/iot-dps/>

### X.509 Attestation

Using X.509 certificates as an attestation mechanism is an excellent way to scale production and simplify device provisioning. In case of ICoN project, choosing the MQTT over websocket causes the authenticate a device to your IoT Hub using two self-signed device certificates. This is sometimes called thumbprint authentication because the certificates contain thumbprints (hash values) that you submit to the IoT hub.

### Create the device registration

The last step is to write a registration application that uses the Device Provisioning Service client SDK to [register the device with the IoT Hub service](https://docs.microsoft.com/en-us/azure/iot-dps/tutorial-set-up-device#create-the-device-registration-software). As a result, the unique **scope\_id** is issued to the device.

## Create and register a provisioning client

The ICoN device to communicate to the Azure IoTHub must be created as a [Provisioning Client](https://docs.microsoft.com/en-us/azure/iot-hub/iot-c-sdk-ref/prov-device-ll-client-h/prov-device-ll-create). The creation step requires three parameters:

1. IotHub URI: (*example iconhubue.azure-devices.net*)
2. Scope\_id (generated as described in 1.1.4).
3. Protocol: *Prov\_Device\_MQTT\_WS\_Protocol*

In case of successful creation ,the device must be [registered](https://docs.microsoft.com/en-us/azure/iot-hub/iot-c-sdk-ref/prov-device-ll-client-h/prov-device-ll-register-device).

# Connection and message exchange

## Create a IoT Hub client

When the device is successfully registered in provisioning client procedure (1.2) a client to communicate with the existing IoT Hub must be created, using the device auth module (<https://docs.microsoft.com/en-us/azure/iot-hub/iot-c-sdk-ref/iothub-client-ll-h/iothubclient-ll-createfromdeviceauth>). The parameters:

1. IoTHub hostname received in the registration process
2. device Id received in the registration process
3. Protocol: *Prov\_Device\_MQTT\_WS\_Protocol*

The device to cloud (and vice-versa) communication varies depends on the reason of the communication (see [device-to-cloud](https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-d2c-guidance) and [cloud-to-device](https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-c2d-guidance) for details). Thus, the different schemas is planned to be used.

## Cloud to device

[Direct methods](https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-direct-methods) for communications that require immediate confirmation of the result. Direct methods are often used for interactive control of devices such as turning on the air cleaner fan.

## Device to cloud

[Device-to-cloud messages](https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-messages-d2c) for reporting the ICoN state (e.g. increase/decrease fan speed) telemetry and raising alerts (errors).

# Messages (API)

## The ICoN sends static device details

Topic: *$iothub/devices/icon-device-id/messages/events*

Send device id, device name, HW version and FW version.

* **JSON**

{  
 "MessageName": "deviceInfo",

"DeviceId": “ICON\_M\_2022.02.0001”,

"HwVersion": "1.0",  
 "FwVersion": "1.0.0"  
}

The message is sent after the ICoN in two cases:

1. Initial message (after successful connection to the cloud)
2. After successfully upgrade

## Device status

The ICoN sends device status information (direct method)

### Topic: *$iothub/devices/icon-device-id/messages/events*

* **JSON**

{  
 "MessageName": "deviceStatus",

"DeviceId": "ICON\_M\_2022.02.0001",

"Timestamp": 1628763832, // when the change of schedule is done  
 "DeviceMode": 0, # 0 - manual, 1 - automatic  
 "TotalOn": 2, // time **when** the device **is** turned **on** [**in** hours]  
 "TimUv1": 5, // percent of lamp life (in comparison with lifespan) "TimUv2": 5, // percent of lamp life (in comparison with lifespan)  
 "TimHepa": 10, // percent of HEPA life

"Rtc": 1628763832, // unix timestamp  
 "FanLevel": 5, // 0 - off ,[1-5] – fan speed level  
 "EcoMode": false, // false - off, true - on  
 "AlarmCodes": [12, 101],

“EthernetOn”: false, // true if ICoN communicates to the cloud over ethernet, otherwise false  
 "TouchLock": false**,** // true if the touchpanel is on and operate

"WifiOn": true**,** //false if the Wifi is off by switch "DeviceReset": true**,**

"ResetReason" **:** 1  
}

The Timestamp filed is mandatory in this message.

There are two types of alarm (to be sent in the AlarmCode): Errors and Warnings. For details see: 

Field “Device Reset” is sent after the ICoN device is reset to the manufactured default values (once after reset).

After restart the ResetReason get the last reset reason. The list is:

|  |  |
| --- | --- |
| value | Reason |
| 0 | Reset reason can not be determined. |
| 1 | Reset due to power-on event. |
| 2 | Reset by external pin (not applicable for ESP32) |
| 3 | Software reset via esp\_restart. |
| 4 | Software reset due to exception/panic. |
| 5 | Reset (software or hardware) due to interrupt watchdog. |
| 6 | Reset due to task watchdog. |
| 7 | Reset due to other watchdogs. |
| 8 | Reset after exiting deep sleep mode. |
| 9 | Brownout reset (software or hardware) |
| 10 | Reset over SDIO |

## Device location

### Data needed to display device location

Topic*: $iothub/devices/icon-device-id/messages/events*

### The ICoN informs the cloud about its location info (send message). The JSON is defined in the section 2.3.3

### Set device location

Topic*: $iothub/methods/POST/device Location*

### The Cloud changes location of the ICoN. The JSON is defined in the section 2.3.3

### JSON data

{

"MessageName": "deviceLocation",  
 "DeviceId": "ICON\_M\_2022.02.0001",  
 "Location": "Cystersów 19, Kraków",  
 "Room": "16",  
}

## Device schedule.

### Report weekly schedule by the ICoN

Topic: *$iothub/devices/icon-device-id/messages/events*

### The ICoN sends weekly schedule by the ICoN to the cloud. . The JSON is defined in the section 2.4.3

### Set device (fan) schedule

Topic*: $iothub/methods/POST/deviceSchedule*

The ICoN receives weekly schedule by the ICoN to the cloud. . The JSON is defined in the section 2.4.3

### JSON data

{

"MessageName": "deviceSchedule",  
 "DeviceId": "ICON\_M\_2022.02.0001",

"Timestamp": 1628763832, // when the change of schedule is done

"Monday":{  
 "fan":[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],  
 "eco":[false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false]},  
 "Tuesday":{  
 "fan":[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],  
 "eco":[false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false]},  
 "Wednesday":{  
 "fan":[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],  
 "eco":[false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false]},  
 "Thursday":{  
 "fan":[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],  
 "eco":[false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false]},  
 "Friday":{  
 "fan":[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],  
 "eco":[false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false]},  
 "Saturday":{  
 "fan":[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],  
 "eco":[false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false]},  
 "Sunday":{  
 "fan":[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],  
 "eco":[false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false]}  
}

The Timestamp filed is mandatory in this message.

## Get device mode (command to the device)

Topic*: $iothub/methods/POST/deviceMode/*

The ICoN receives the new device mode, fan speed, eco mode or touchpad lock/unlock request.

* **JSON**

{  
 "MessageName": "deviceMode",

"DeviceId": "ICON\_M\_2022.02.0001",

"Timestamp": 1628763832, // when the change is requested  
 "DeviceMode": 0, // 0 - manual, 1 - automatic  
 "FanLevel": 5, // 0 - off ,[1-5]  
 "EcoMode": false, // false - off, true - on  
 "TouchLock": false // false means touch is to be unlocked  
}

The Timestamp filed is mandatory in this message.

## Service setting

Topic: *$iothub/methods/POST/deviceService*

The ICoN receives service request (UV lamp/HEPA filter replacement, RT clock reset request or change of the UV lamp/HEPA filter parameter)

* **JSON**

{  
 "MessageName": "deviceService",

"DeviceId": "ICON\_M\_2022.02.0001",

"DeviceReset": 0, // 0 - no, 1 - yes  
 "TimUv1Reload": 0, // 0 - no, 1 - yes  
 "TimUv2Reload": 0, // 0 - no, 1 - yes  
 "TimHepaReload": 0, // 0 - no, 1 - yes  
 "ScheduleReset": 0, // 0 - no, 1 - yes  
 "RtcSet": 1628763832, // unix timestamp

"HepaLivespan": 9000, // in hours

"UvLivespan": 8000, // in hours

"HepaWarning": 8100, // in hours

"UvWarning": 7200, // in hours

"UtcTimeoffset": 2 // UTC offset (in hours)  
}

## Firmware update

Topic*: $iothub/methods/POST/deviceUpdate*

The ICoN receives the upgrade firmaware request. In case of successful upgrade the ICoN sends the deviceInfo message (section 2.1) with the new FwVersion.

* **JSON**

{

"MessageName": "deviceUpdate",  
 "DeviceId": "ICON\_M\_2022.02.0001",  
 "NewDeviceId": "2",

"FwVersion": "2.1.1",  
 "FwPackageURI": "https://example.com",  
 "FwPackageCheckValue": 3421780262  
}

# Data types

All information in attached spreadsheet

# Document history

|  |  |  |
| --- | --- | --- |
| Version | Data | Author |
| 1.0 | 2021-08-13 | Mateusz Fiołek |
| 1.4 | 2021-11-26 | Artur Łoziński |
| 1.5 | 2021-12-02 | Artur Łoziński |
| 1.7 | 2021-12-14 | Artur Łoziński |
| 1.8 | 2021-12-14 | Artur Łoziński |
| 2.0 | 2021-12-17 | Artur Łoziński |
| 2.1 | 2022-01-04 | Artur Łoziński |
| 2.2 | 2022-03-30 | Artur Łoziński |
| 2.3 | 2022-04-19 | Artur Łoziński |