
AZURE-IOT-SDK User Manual

RT-THREAD Document Center

Copyright ©2019 Shanghai Ruisaide Electronic Technology Co., Ltd.



WWW.RT-THREAD.ORG

Friday 28th September, 2018

Versions and Revisions

Date	Version	Author	Note
2018-07-27	v0.1	SummerGift	Initial Version

Table of contents

Versions and Revisions	i
Table of contents	ii
1 Software Package Introduction	1
1.1 Package directory structure.	1
1.2 Azure Overview.	2
1.3 Function Introduction.	2
1.4 Platform and Hardware Compatibility . . .	4
2 Working Principle	5
2.1 IoT client framework.	5
3. Usage Guide	8
3.1 Device-to-cloud communication.	8
3.2 Cloud-to-device communication.	9
3.3 References.	10
4 API Description	11
4.1 Programming model selection.	11
4.2 IoTHubClient Common APIs.	12
4.2.1 Initialize the IoT Center Client.	12
4.2.2 IoT Center Client Resource Release.	12
4.2.3 Set the send event callback function.	13
4.2.4 Set the callback function for receiving messages.	13
4.2.5 Setting up the Physics Center Client.	14

4.2.6 Setting the connection status callback	14
4.2.7 Complete (send/receive) work.	15
5. Sample Program	16
5.1 Introduction.	16
5.2 Preparation	16
5.3 Get the Azure software package.	17
5.4 Introduction to communication protocol.	18
5.5 Create an IoT hub.	18
5.6 Registering a device.	21
5.7 Function Example 1: Device sends telemetry data to IoT Center.	26
5.7.1 Example files.	26
5.7.2 Cloud Monitoring Device Data	26
5.7.3 Modify the device connection string in the sample code.	27
5.7.4 Run the sample program.	29
5.8 Function Example 2: The device monitors data sent from the cloud.	31
5.8.1 Example files.	31
5.8.2 Modify the device connection string in the sample code.	31
5.8.3 Run the sample program on the device.	32
5.8.4 The server sends data to the device.	32

Chapter 1

Software Package Introduction

Azure is a software package ported by RT-Thread for connecting to Microsoft Azure IoT Center. The original SDK is: [azure-iot-sdk-c](#). RT-Thread devices can be easily connected to Azure IoT Center.

Azure IoT Hub, a service hosted in the cloud, acts as a central message hub for bidirectional communication between IoT applications and the devices they manage. Azure IoT Hub enables reliable and secure communication between millions of IoT devices and cloud-hosted solution backends, enabling you to build IoT solutions. You can connect virtually any device to IoT Hub.

Using Azure software packages to connect to IoT Hub enables the following functions:

- Easily connect to Azure IoT Hub and establish reliable communication with Azure IoT
- Set identity and credentials for each connected device and help maintain the confidentiality of cloud-to-device and device-to-cloud messages
- Administrators can remotely maintain, update, and manage IoT devices at scale in the cloud
- Receive telemetry data from devices at scale
- Routing data from devices to stream event processors
- Receive file uploads from devices
- Send cloud-to-device messages to specific devices

You can use Azure IoT Hub to implement your own solution backend. In addition, IoT Hub also includes identity registration Table that you can use to provision devices, their security credentials, and their permissions to connect to IoT Hub.

1.1 Software Package Directory Structure

```

azure
+---azure                                // Azure Cloud Platform SDK
+---azure-port                          // Migrate files
+---docs
    +---figures | api.md                // Document using images
    | introduction.md                  // API usage instructions
    | principle.md | README.md |       // Introduction document
    samples.md | user-guide.md         // Implementation principle
                                        // Document structure description
                                        // Package example
                                        // Instructions
    +---version.md                     // Edition
||||||| +---                           // Example code
    | iotHub_Ill_telemetry_sample // Device upload telemetry data example
    +---iotHub_Ill_c2d_sample // Package // Example of device receiving cloud data
    LICENSE                           License
samples ||||                           // Software package instructions
README.md +---SConscript               //RT-Thread default build script

```

1.2 Azure Overview

Microsoft Cloud in China is provided by 21Vianet to provide services in China, while the international version of Microsoft Cloud provides services worldwide. 21Vianet's locally operated Microsoft Azure provides organizations and individuals with different needs with services covering infrastructure, data. The comprehensive cloud capabilities in the fields of database, Web application, artificial intelligence, Internet of Things, etc. fully meet the needs of different industries, different scales, Diverse business needs of users with different IT levels.

Microsoft Azure, operated by 21Vianet, is the first commercially available service in China and complies with relevant Chinese government regulations. The service is operated by 21Vianet, a Chinese company that has obtained a public cloud business license in China. Compared with other international public cloud service providers that have landed in China, Azure has some striking advantages: stable and high availability, Provide up to 95% service level agreement guarantee.

Microsoft Azure, operated by 21Vianet, uses physically isolated cloud service instances located in China. It uses Microsoft's Azure technology that serves the world, and can provide customers with global consistent service quality assurance. domain, mainly using Microsoft Cloud's Azure IoT Hub to meet the cloud connection needs of devices.

1.3 Function Introduction

The architecture diagram of Azure IoT Hub is as follows:

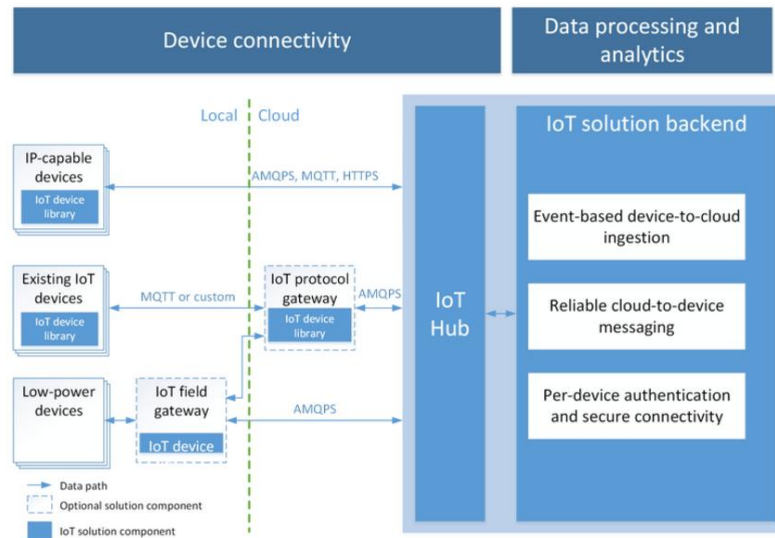


Figure 1.1: Azure IoT Hub architecture diagram

Azure IoT Hub provides the following features:

- Establish two-way communication with millions of IoT devices

Rely on Azure IoT Hub to easily and securely connect Internet of Things (IoT) assets. Reliably send commands and notifications to connected devices with cloud-to-device messaging, and track message delivery with confirmation receipts. Send device messages using durable methods to accommodate intermittently connected devices.

- Platforms and protocols

Add new devices and connect existing ones using open-source device SDKs for multiple platforms, including Linux, Windows, and real-time operating systems. Use standard and custom protocols, including **MQTT** v3.1.1, **HTTPS 1.1**, or **AMQP 1.0**.

- [IoT security](#)

Set up identity and credentials for each connected device and help maintain confidentiality of cloud-to-device and device-to-cloud messages. To maintain system integrity, selectively revoke access permissions for specific devices as needed. You can set up unique security keys for each device to connect to IoT Hub. [IoT Hub Identity Registry](#) Device identities and keys are stored within the solution. The solution backend can add individual devices to allow or deny lists, providing complete control over device access.

Additional connection security features include:

- The communication path between devices and Azure IoT Hub, or between gateways and Azure IoT Hub, will be secured using industry-standard Transport Layer Security (TLS) with Azure IoT Hub using X.509 protocol authentication .

- To protect devices from unsolicited inbound connections, Azure IoT Hub does not open any device Connect. The device will initiate all connections.

Azure IoT Hub permanently stores messages from devices and waits for connected devices. These commands are stored for two days, allowing devices to occasionally connect to receive them based on power or connectivity issues. Azure IoT Hub maintains a device queue for each device.

- Manage IoT devices at scale through device management

With the new device management capabilities of Azure IoT Hub, administrators can remotely maintain, update, and manage IoT devices at scale in the cloud. Device twins: Device twins can be used to store, synchronize, and query device metadata and state information. A device twin is a JSON document that stores device state information, such as metadata, configuration, and conditions. IoT Hub maintains a device twin for every device connected to the IoT Hub, which can scale to millions of devices.

- Leverage edge intelligence with the Azure IoT Gateway SDK

The IoT Gateway SDK provides a powerful framework for building, configuring, and deploying edge logic, enabling you to do more with Azure IoT.

1.4 Platform and Hardware Compatibility

The device platform requirements for connecting to Azure IoT Hub are as follows:

- Capable of establishing IP connectivity: Only devices that support IP can communicate directly with Azure IoT Hub
- Support for TLS: Required to establish a secure communication channel with Azure IoT Hub
- Support for SHA-256 (optional): Required to generate security tokens used to authenticate devices with services.

Different authentication methods, not all require SHA-256

- Have a real-time clock or implementation code to connect to an NTP server: establish a TLS connection and generate a security token for Required for authentication
- Have at least 64KB of RAM: The SDK's memory footprint depends on the SDK and protocol used, as well as the target Platform. Minimum footprint for microcontrollers

Chapter 2

How it works

2.1 IoT Client Framework

Azure IoT Hub provides a variety of connection protocols, such as MQTT and HTTP, to facilitate device connectivity. However, Azure IoT Hub only supports secure connections. Connections to the IoT Hub are established by device clients. Each device connected to the IoT Hub creates an IoT Hub client instance. When the connection is closed, this instance is simply released.

The IoT Hub client calls the LL layer to complete its work. The LL layer connects to the transport layer of different communication protocols, which in turn connects to the communication protocol implementation layer. The following two figures illustrate the call hierarchy when the IoT Hub client completes its functions:

- IoT client framework HTTP/MQTT function call relationship diagram:

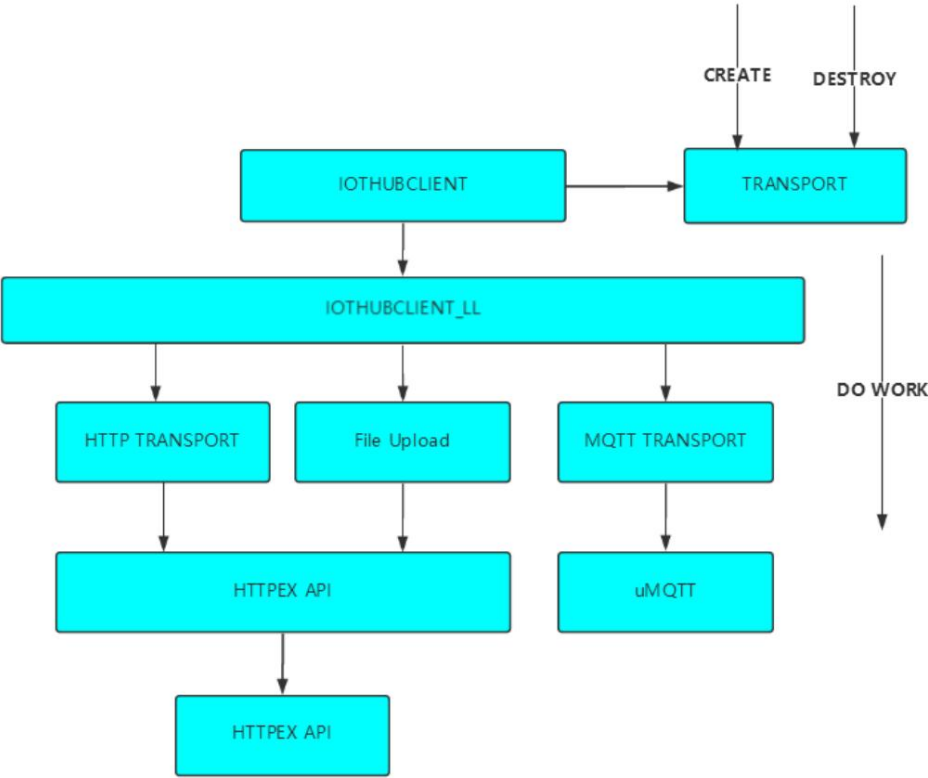


Figure 2.1: HTTP/MQTT Function call relationship diagram

- The following figure shows the API call using HTTP protocol as an example:

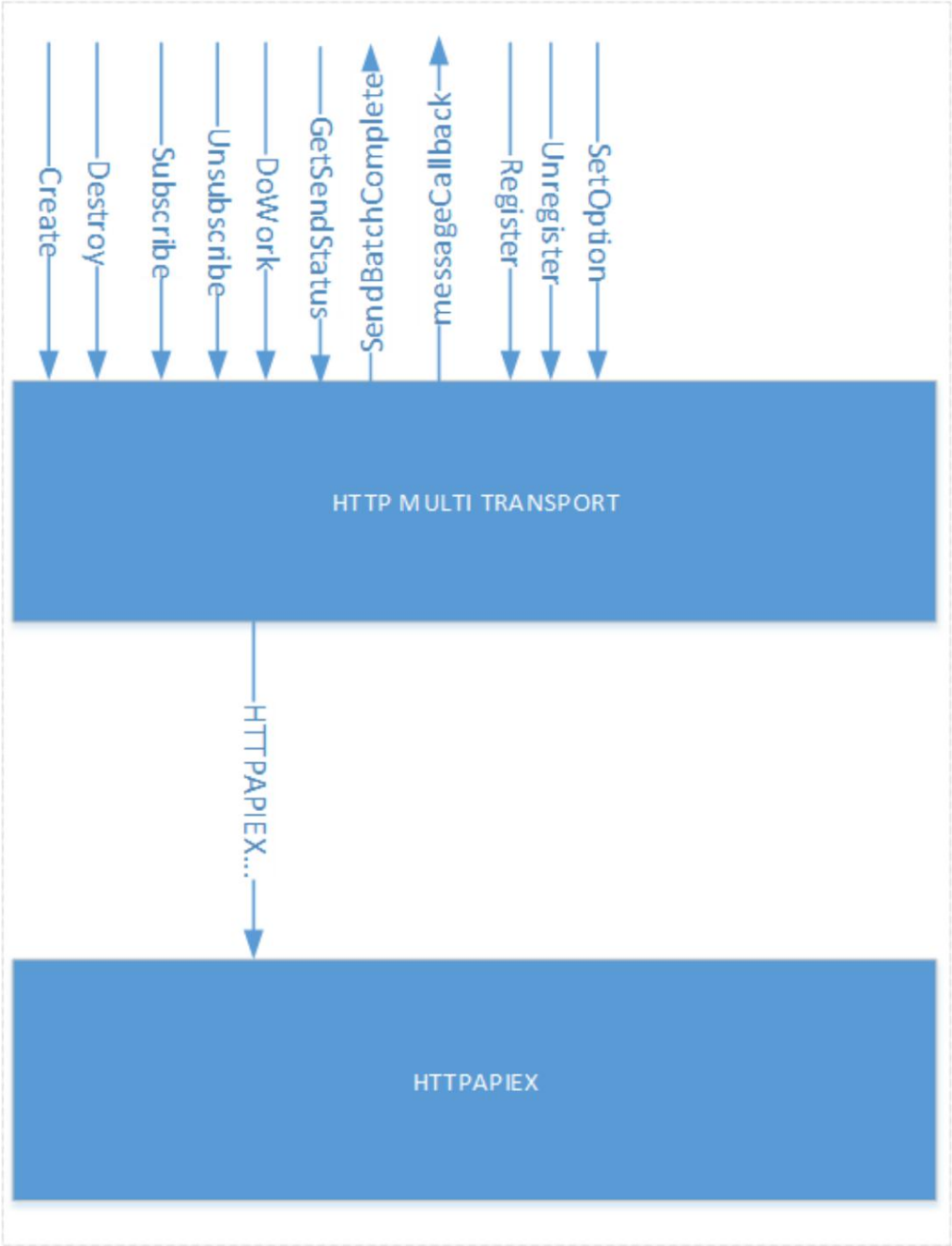


Figure 2.2: HTTP protocol API Call

Chapter 3

Usage Guidelines

Azure IoT Hub is a fully managed service that helps connect millions of devices and a single solution backend.

Azure IoT Hub provides the following features:

- Secure communications with per-device security credentials and access controls
- Multiple device-to-cloud and cloud-to-device communication options for ultra-large-scale communication
- Queryable storage of per-device state information and metadata
- Easily establish device connections through device libraries for the most popular languages and platforms

3.1 Device-to-Cloud Communication

IoT Hub exposes three options when sending information from a device app to a solution backend:

- Device-to-cloud messages for time-series telemetry and alerts
- Device twin reporting properties for reporting device state information such as available capabilities, conditions, or long-running jobs
 - The status of the workflow. For example, configuration and software updates
- File upload, for media files and large files uploaded by devices with intermittent connections or compressed to save bandwidth
 - Telemetry batch files

Here's a detailed comparison of various device-to-cloud communication options:

	设备到云的消息	设备克隆的报告属性	文件上传
方案	遥测时序和警报。例如，每隔 5 分钟发送的 256KB 传感器数据批。	可用功能和条件。例如，当前设备连接模式，诸如手机网络或 WiFi。同步长时间运行的工作流，如配置和软件更新。	媒体文件。大型（通常为压缩的）遥测批。
存储和检索	通过 IoT 中心临时进行存储，最多存储 7 天。仅顺序读取。	通过 IoT 中心存储在设备孪生中。可使用 IoT 中心查询语言检索 。	存储在用户提供的 Azure 存储帐户中。
大小	消息大小最大为 256-KB。	报告属性大小最大为 8 KB。	Azure Blob 存储支持的最大文件大小。
频率	高。有关详细信息，请参阅 IoT 中心限制 。	中。有关详细信息，请参阅 IoT 中心限制 。	低。有关详细信息，请参阅 IoT 中心限制 。
协议	在所有协议上可用。	使用 MQTT 或 AMQP 时可用。	在使用任何协议时可用，但设备上必须具备 HTTPS。

Figure 3.1: Device-to-cloud communication

An application might need to both send information as telemetry series or alerts and make it available in the device twin.

In this case, you can choose one of the following options:

- The device app sends a device-to-cloud message and reports the property changes • The solution backend, upon receiving the message, can store the information in a tag on the device twin

Because device-to-cloud messages allow for much higher throughput than device twin updates, it is sometimes desirable to avoid sending a single message for each device-to-cloud message.

Cloud messaging updates device twins.

3.2 Cloud-to-Device Communication

IoT Hub provides three options for allowing device apps to expose functionality to backend apps:

- Direct methods are suitable for communications that require immediate confirmation of results. Direct methods are often used to control devices interactively.
For example, turn on the fan
- Desired properties of a twin, for long-running commands designed to put a device into a desired state. For example,
Telemetry sending interval is set to 30 minutes
- Cloud-to-device messaging, suitable for providing one-way notifications to device applications

Here's a detailed comparison of various cloud-to-device communication options:

	直接方法	克隆的所需属性	云到设备的消息
方案	需要立即确认的命令，例如打开风扇。	旨在将设备置于某个所需状态的长时间运行命令。例如，将遥测发送间隔设置为 30 分钟。	提供给设备应用的单向通知。
数据流	双向。设备应用可以立即响应方法。解决方案后端根据上下文接收请求结果。	单向。设备应用接收更改了属性的通知。	单向。设备应用接收消息
持续性	不联系已断开连接的设备。通知解决方案后端：设备未连接。	设备孪生会保留属性值。设备会在下次重新连接时读取属性值。属性值可通过 IoT 中心查询语言 检索。	IoT 中心可保留消息长达 48 小时。
目标	通过 deviceld 与单个设备通信，或通过 作业 与多个设备通信。	通过 deviceld 与单个设备通信，或通过 作业 与多个设备通信。	通过 deviceld 与单个设备通信。
大小	直接方法有效负载的最大大小为 128 KB。	所需属性大小最大为 8 KB。	最多 64 KB 消息。
频率	高。有关详细信息，请参阅 IoT 中心限制 。	中。有关详细信息，请参阅 IoT 中心限制 。	低。有关详细信息，请参阅 IoT 中心限制 。
协议	使用 MQTT 或 AMQP 时可用。	使用 MQTT 或 AMQP 时可用。	在所有协议上可用。使用 HTTPS 时，设备必须轮询。

Figure 3.2: Cloud-to-device communication

Learn how to use direct methods, required properties, and cloud-to-device messaging in the following tutorials:

- Using direct methods: for direct methods • Configuring devices
- with desired properties: for desired properties of device twins • Sending messages from the cloud to the device: for messages from the cloud to the device

3.3 References

The Azure Documentation Center provides a wealth of development resources. For more detailed information, please refer to the following address:

- [Azure official website](#)
- [IoT Hub documentation](#)
- [Azure Development Guide](#)

Chapter 4

API Description

When writing applications in Azure, you can easily send and receive messages using the **IoTHubClient** library. This chapter introduces the common APIs provided by the **IoTHubClient** library. For more detailed APIs, see the Azure SDK [C API Reference](#).

4.1 Programming Model Selection

The IoTHubClient library provides two sets of easy-to-use APIs: one with "LL" in its name and one without. The "LL" APIs are lower-level APIs. Whether you choose the "LL" API or the "LL" API, be consistent. If you first call **IoTHubClient_LL_CreateFromConnectionString**, be sure to use only the corresponding lower-level API for any subsequent work :

- IoTHubClient_LL_SendEventAsync •
- IoTHubClient_LL_SetMessageCallback •
- IoTHubClient_LL_Destroy
- IoTHubClient_LL_DoWork

The opposite is also true. If you call **IoTHubClient_CreateFromConnectionString** first, Please use non-LL APIs for any other processing.

These functions have "LL" in their API names. In addition, each of these functions has parameters that are identical to their non-LL counterparts. However, there is an important difference in the behavior of these functions.

When you call **IoTHubClient_CreateFromConnectionString**, the underlying library creates a new thread that runs in the background. This thread sends events to and receives messages from IoT Hub. When using the "LL" API, no such thread is created. The background thread is created for developer convenience. You don't need to worry about explicitly sending and receiving messages to and from IoT Hub; this happens automatically behind the scenes. In contrast, with the "LL" API, you can explicitly control communication with IoT Hub as needed.

4.2 IoT Hub Client Common APIs

The following introduces commonly used APIs using the APIs whose names contain "LL" as an example.

4.2.1 IoT Center Client Initialization

```
IOTHUB_CLIENT_LL_HANDLE  
  
IoT Hub Client_LL_CreateFromConnectionString(  
  
    const char* connectionString,  
  
    IOTHUB_CLIENT_TRANSPORT_PROVIDER protocol)  
  
Creates an IoT Hub client using the specified connection string parameters to communicate with an existing IoT Hub.
```

parameter	describe
connectionString	Device connection string
protocol	Function pointers implemented by the protocol
return	describe
IOTHUB_CLIENT_LL_HANDLE A non-null iotHubClientHandle value, which is used in calling IoT	
Used when other functions of the client are used	
NULL	fail

4.2.2 IoT Center Client Resource Release

```
void  
  
IoT Hub Device Client_LL_Destroy(  
  
    IOTHUB_DEVICE_CLIENT_LL_HANDLE iotHubClientHandle)  
  
Releases resources allocated by the IoT Hub Client. This is a blocking call.
```

parameter	describe
iotHubClientHandle	Device connection string
return	describe
none	none

4.2.3 Set the send event callback function

IOTHUB_CLIENT_RESULT

IoTHubClient_LL_SendEventAsync(

IOTHUB_CLIENT_LL_HANDLE IoTHubClientHandle,

IOTHUB_MESSAGE_HANDLE eventMessageHandle,

IOTHUB_CLIENT_EVENT_CONFIRMATION_CALLBACK eventConfirmation-

Callback,

void* userContextCallback)

The asynchronous call sends the message specified by eventMessageHandle.

parameter	describe
IoTHubClientHandle	The handle of the created IoT hub client
eventMessageHandle	IoT Center message handler
eventConfirmation Callback	The callback specified by the device to receive the delivery confirmation of the IoT center message. This callback can be called by the same message: iothubclientllsendeventasync function, attempts to retry sending a failed The user can specify a NULL value here to indicate that No callback required
userContextCallback	The user specified context will be provided to the callback, can be empty
return	describe
IOTHUB_CLIENT_OK	Setup successful
error code	fail

4.2.4 Set the callback function for receiving messages

IOTHUB_CLIENT_RESULT

IoTHubDeviceClient_LL_SetMessageCallback(

IOTHUB_DEVICE_CLIENT_LL_HANDLE IoTHubClientHandle,

IOTHUB_CLIENT_MESSAGE_CALLBACK_ASYNC messageCallback,

void* userContextCallback)

Sets the message callback to be called when the IoT hub sends a message to the device. This is a blocking call.

parameter	describe
iotHubClientHandle	The handle of the created IoT hub client
messageCallback	A callback specified by the device to receive messages from the IoT hub.
userContextCallback	The user specified context will be provided to the callback, can be empty
return	describe
IOTHUB_CLIENT_RESULT Initialization of IoT client succeeded	
NULL	fail

4.2.5 Setting up the Physics Center Client

IOTHUB_CLIENT_RESULT

IoTHubDeviceClient_LL_SetOption

IOTHUB_DEVICE_CLIENT_LL_HANDLE iotHubClientHandle,

const char* optionName, const void* value)

This API sets a runtime option identified by the parameter `optionName`, the option name and data type value point to. Each option is specific.

parameter	describe
iotHubClientHandle	The handle of the created IoT hub client
optionName	Option Name
value	Specific value
return	describe
IOTHUB_CLIENT_OK Setup successful	
error code	fail

4.2.6 Setting connection status callback

IOTHUB_CLIENT_RESULT

IoTHubDeviceClient_LL_SetConnectionStatusCallback(

IOTHUB_DEVICE_CLIENT_LL_HANDLE iotHubClientHandle,

IOTHUB_CLIENT_CONNECTION_STATUS_CALLBACK connectionStatusCallback,

void * userContextCallback)

Sets the connection status callback to be called to indicate the status of the connection to the IoT hub. This is a blocking call.

parameter	describe
iotHubClientHandle	The handle of the created IoT hub client
connectionStatusCallback	Device-specified callback for receiving status information about the connection to the IoT Hub.
	Status Update
userContextCallback	The user specified context will be provided to the callback, can be empty
return	describe
IOTHUB_CLIENT_OK	Setup successful
error code	fail

4.2.7 Completing (Sending/Receiving) Work

void

IoTHubDeviceClient_LL_DoWork(

IOTHUB_DEVICE_CLIENT_LL_HANDLE iotHubClientHandle)

This function will be called by the user when work (send/receive) can be completed by IoTHubClient.

IoTHubClient interactions (regarding network traffic and/or user-level callbacks) are the result of calling this function.

DoWork is performed synchronously.

parameter	describe
iotHubClientHandle	The handle of the created IoT hub client
return	describe
none	none

Chapter 5

Sample Program

5.1 Introduction

IoT Hub is an Azure service that brings large amounts of telemetry data from IoT devices into the cloud for storage or processing.

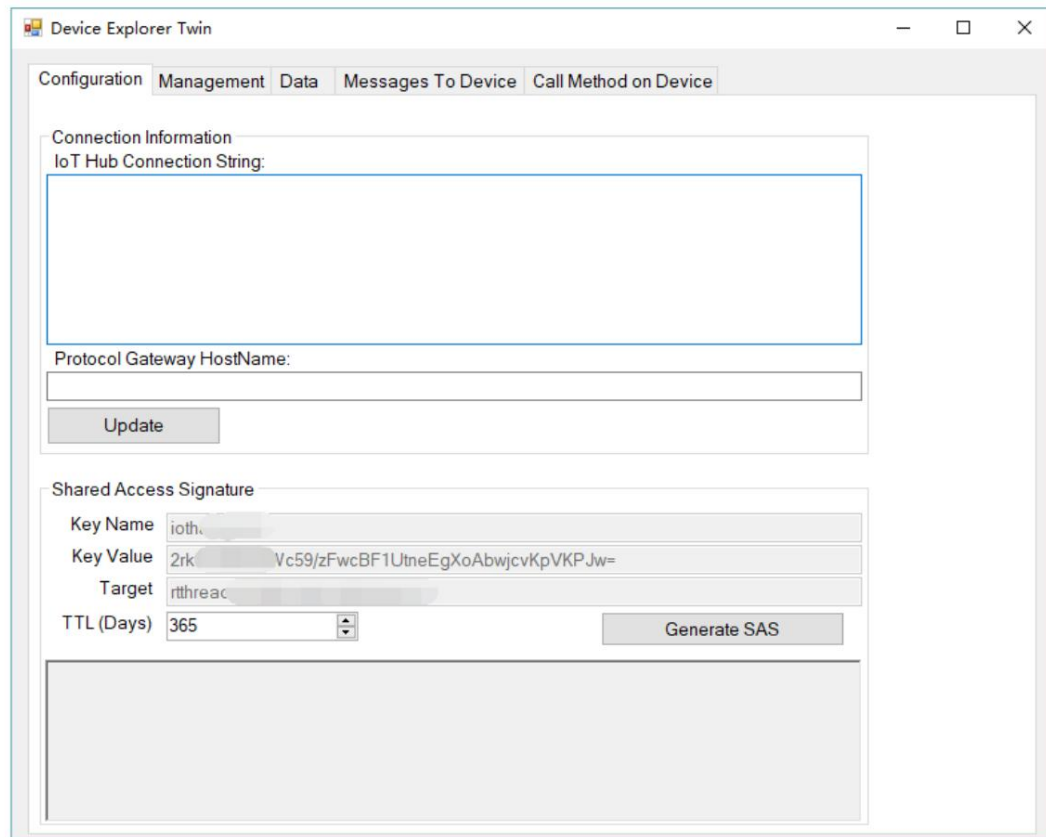
This sample program will demonstrate the data exchange function between the device and the IoT center.

Next, we'll run two functional examples provided by the Azure software package: one demonstrating how a device can send telemetry data to the cloud, and the other demonstrating how to receive data from an IoT hub. Before running these two examples, you need to create an IoT hub and register a device with it.

5.2 Preparation

Before running this sample program, you need to prepare the following:

1. Sign up for a Microsoft Azure account. If you don't have an Azure subscription, create a trial account before you begin.
2. Install the DeviceExplorer tool, which is essential for testing Azure software package functionality on Windows. The installation package for this tool is SetupDeviceExplorer.msi in the tools directory. Follow the prompts to install it. A successful installation will display the following interface.

Figure 5.1: *DeviceExplorer* Tool interface

5.3 Get the Azure software package

- Add the software package to the project and run the system on a development board with sufficient resources

Open the env tool provided by RT-Thread. In the bsp directory, use the menuconfig command to open the configuration options. Select the [Azure: Microsoft Azure Cloud SDK for RT-Thread](#) package and enable the required functional examples. Next, use the `pkgs --update` command to add the Azure package to the system. The env tool will automatically add the package's dependencies.

```
RT-Thread online packages --->
  IoT - internet of things --->
    IoT Cloud --->
      [*] Azure: Microsoft azure cloud SDK for RT-Thread --->
        Choose Protocol (Using MQTT Protocol) ---> #Select communication protocol
        [*] Enable Azure iotHub telemetry example #device sends telemetry data
            Example
        [*] Enable Azure iotHub cloud to device example #Send data from the cloud
            Example
```

Version (latest) ---> Book

#Select the software package version

5.4 Introduction to Communication Protocol

Currently, the sample code provided by the RT-Thread Azure package supports the MQTT and HTTP communication protocols. To choose which protocol to use, just select the corresponding protocol from the options above. When selecting the device-side communication protocol, you need to pay attention to the following points:

1. When sending cloud-to-device data, HTTPS does not have an efficient method for implementing server push. Therefore, when using the HTTPS protocol, the device polls the IoT hub for messages from the cloud to the device. This method is inefficient for both the device and the IoT hub. According to current HTTPS guidelines, each device should poll for messages every 25 minutes or longer. MQTT supports server push upon receipt of cloud-to-device messages. This enables direct message push from the IoT hub to the device. If delivery latency is a concern, MQTT is preferred. HTTPS is also suitable for devices that rarely connect.

2. When using HTTPS, each device should poll for messages from the cloud to the device every 25 minutes or more. However, during development, polling can be performed at a higher frequency of less than 25 minutes.

3. For more detailed protocol selection documentation, please refer to the Azure [official document "Choosing a Communication Protocol"](#).

5.5 Create IoT Hub

The first thing to do is to create an IoT hub in your subscription using the Azure portal. The IoT hub is used to bring large amounts of telemetry data from many devices into the cloud. The hub then allows one or more backend services running in the cloud to read and process that telemetry data.

1. Log in to [Azure](#) 2. Select Create a resource > Internet of Things > IoT Hub.

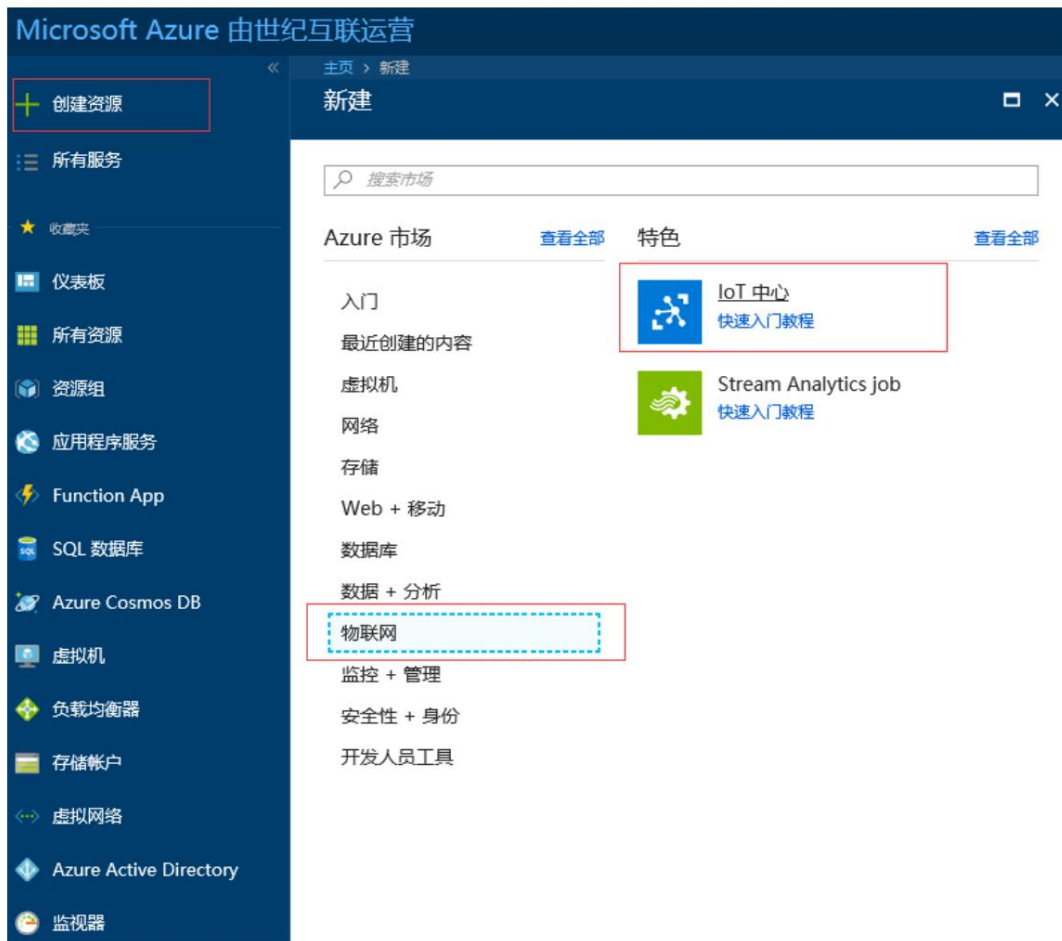


Figure 5.2: Creating an IoT Hub

3. In the IoT Hub pane, enter the following information for your IoT hub:

- **Subscription:** Select the subscription that needs to be used to create this IoT hub.
- **Resource Group:** Create a resource group to host your IoT Hub, or use an existing one
For more information, see [Using Resource Groups](#).
[Manage Azure resources](#).
- **Region:** Select the nearest location.
- **IoT Hub Name:** Create a name for the IoT hub. This name needs to be unique.
If the name you entered is available, a green check mark appears.

主页 > 新建 > IoT 中心

IoT 中心
Microsoft

Basics **Size and scale** Review + create

Create an IoT Hub to help you connect, monitor, and manage billions of your IoT assets. [Learn More](#)

PROJECT DETAILS

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

* Subscription ⓘ 1 元人民币的试用订阅

* Resource Group ⓘ ☒ 新建 ☐ 使用现有项

* Region ⓘ 中国北部

* IoT Hub Name ⓘ *Name your IoT Hub*

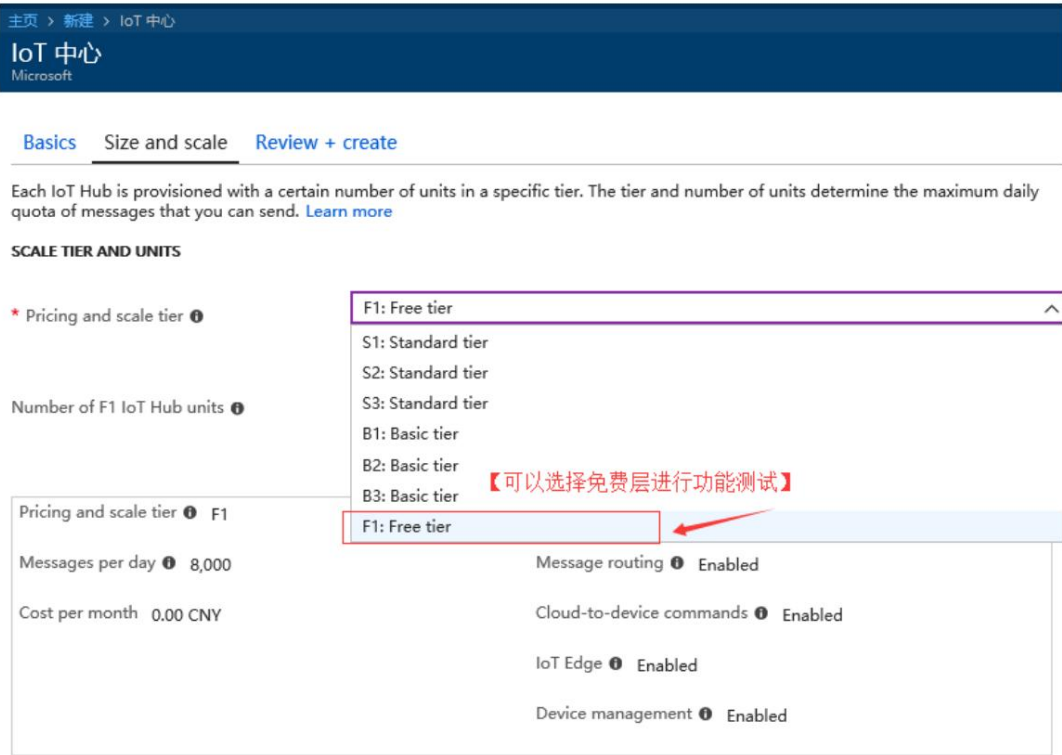
【创建 IoT 中心的名称，需要是唯一的】

Figure 5.3: fill in IoT Center Information

4. Select Next: **Size and scale** to continue creating the IoT hub.

5. Select **"Pricing and scale tier"**. For testing purposes, select **"F1:Free tier (F1 - Free)"** tier (provided this tier is still available on the subscription). For more information, see [Pricing and](#)

[Put the layer.](#)



6. Select “Review + create ” .
7. Review the IoT hub information and click Create. Creating the IoT hub may take several minutes.
You can monitor the progress in the Notifications pane. Once the creation is successful, you can proceed to the next step of registering the device.
8. To facilitate subsequent searches, you can manually add successfully created resources to the dashboard.

5.6 Registering a Device

To run device-related examples, you need to register the device information in the IoT center first, and then the device can connect to Connect to the IoT Hub. In this example, you can use the DeviceExplorer tool to register the device.

- Get the IoT Hub's shared access key (i.e., IoT Hub connection string)

1. After the IoT center is created, in the Settings column, click the Shared Access Policy option to open the IoT center Access permission settings: Open iothubowner and obtain the shared access key of the IoT hub in the properties box that pops up on the right.

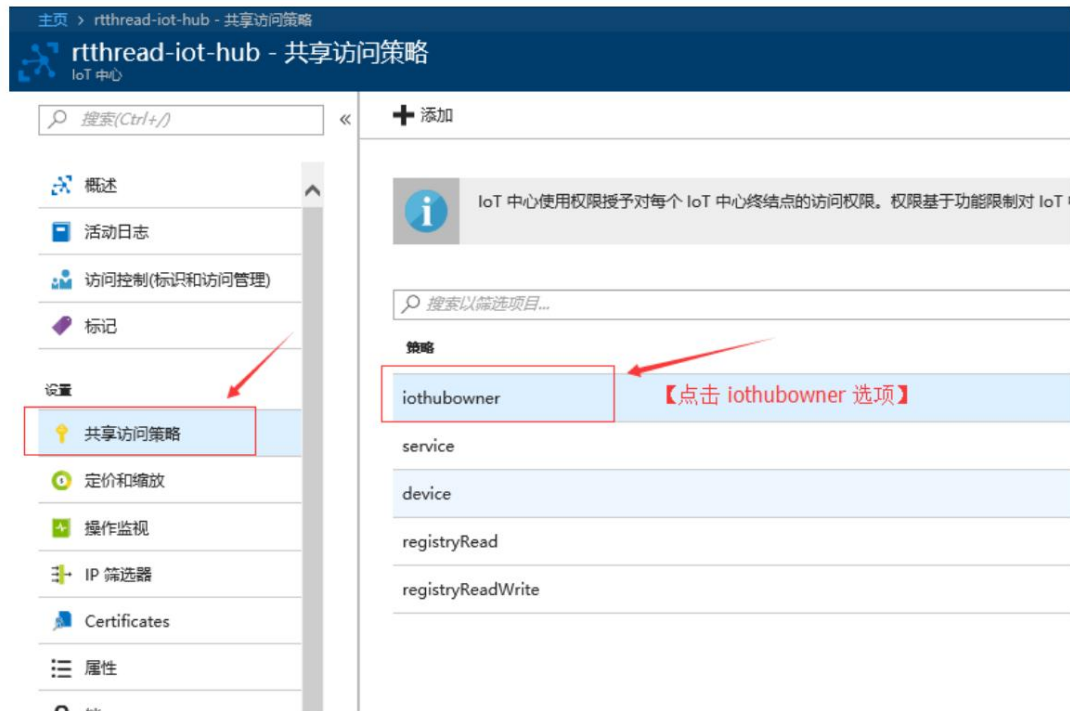


Figure 5.5: View IoT Hub shared access policies

2. Get the IoT Hub connection string in the properties box that pops up on the right:

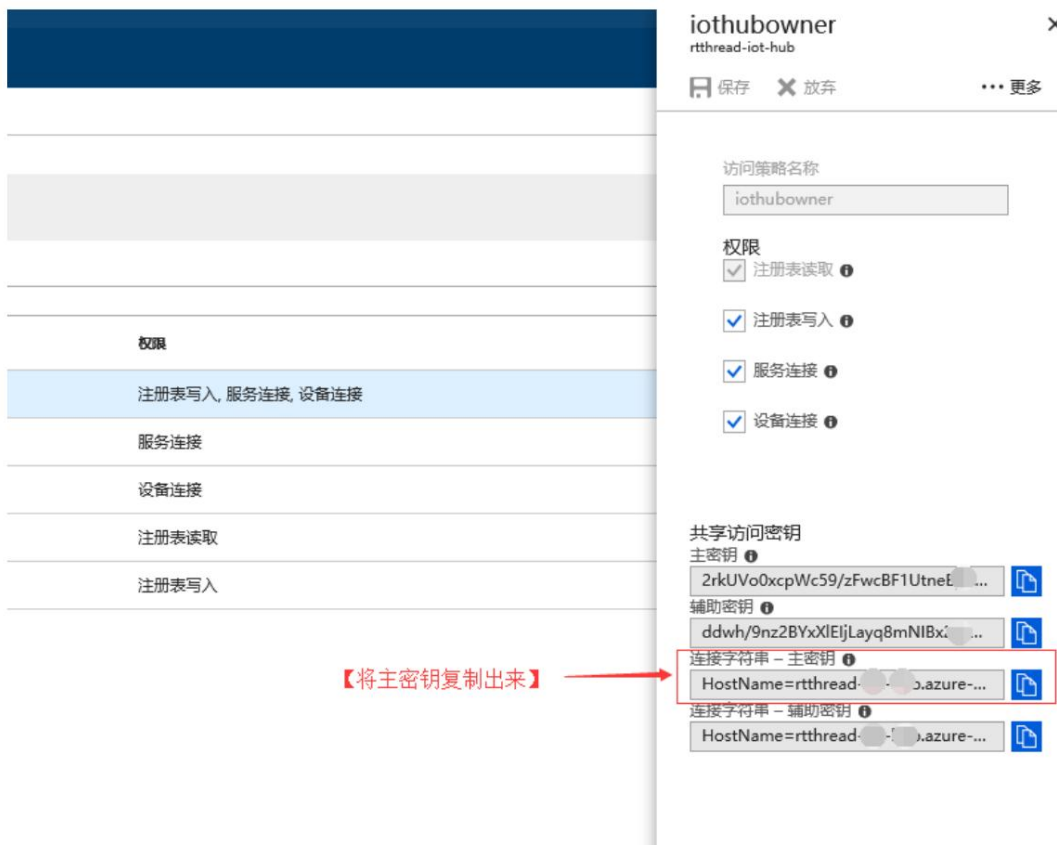


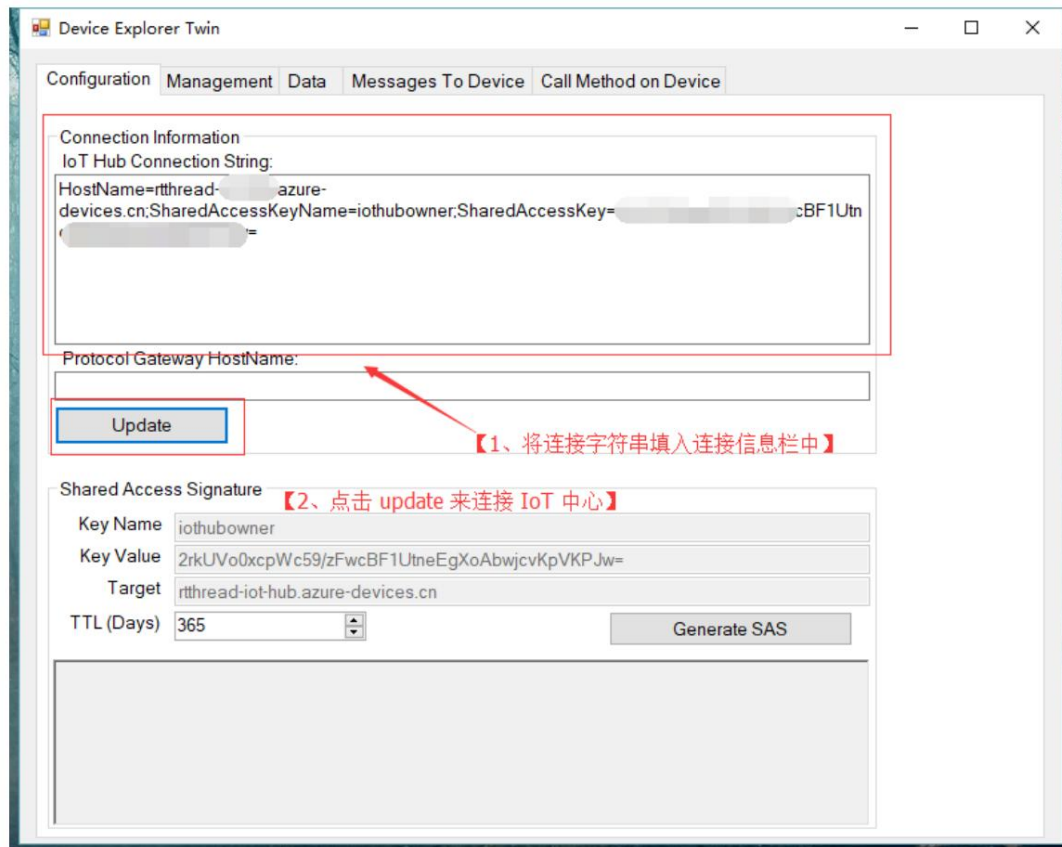
Figure 5.6: Copy the access master key

• Create a device

1. With the connection string, we can use the DeviceExplorer tool to create a device and test IoT

Center function. Open the test tool and fill in the connection string in the configuration options. Click the [update](#) button to update this

The configuration of connecting to the IoT center is ready for the next step of creating a test device.

Figure 5.7: Configuration *DeviceExplorer* tool

2. Open the Management tab and follow the steps shown below to create a test device. After the device is successfully created,

You can now run the functional examples of the device.

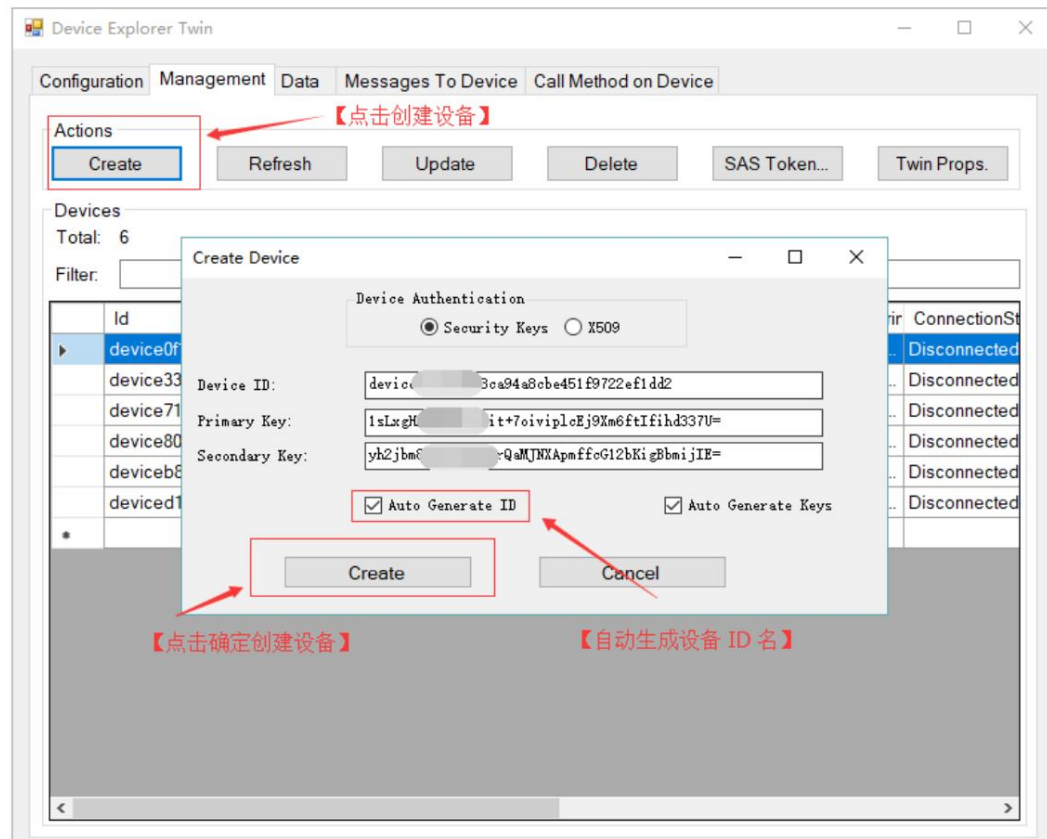


Figure 5.8: Create a device

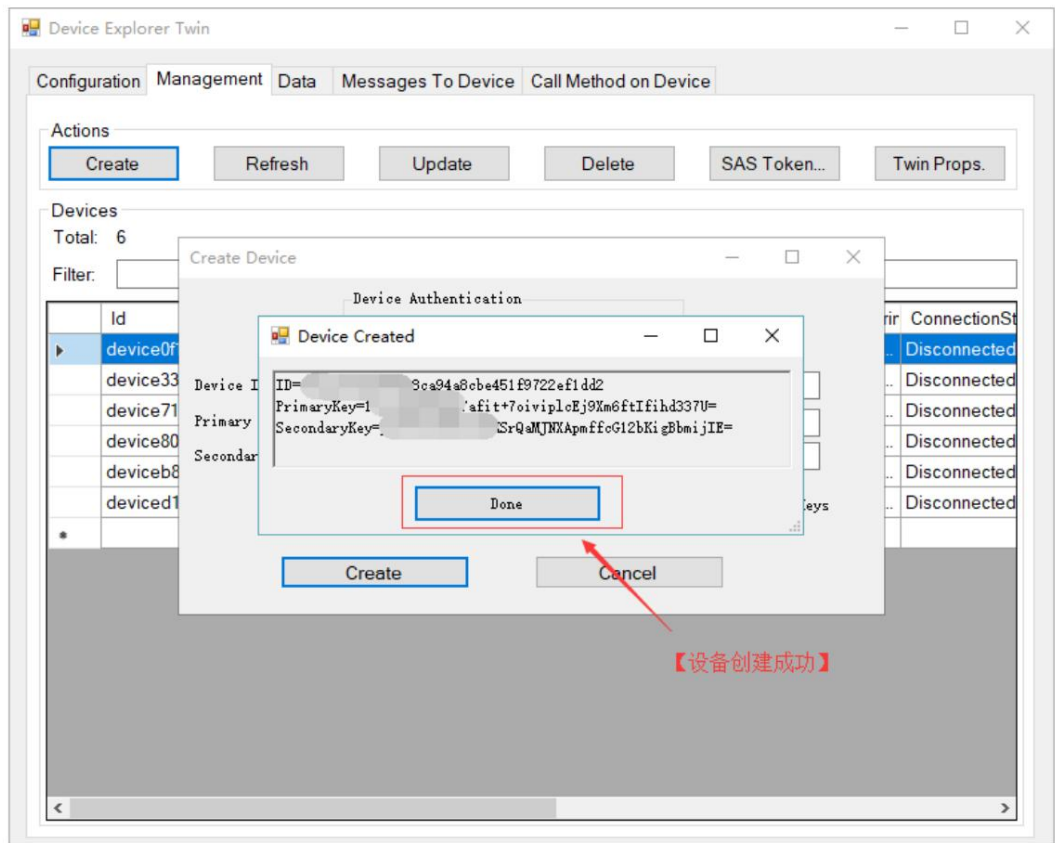


Figure 5.9: Device created successfully

5.7 Function Example 1: Device Sending Telemetry Data to IoT Center

5.7.1 Example File

Sample program path	illustrate
<code>samples/iothub_ll_telemetry_sample.c</code>	Send telemetry data from devices to Azure IoT center

5.7.2 Cloud Monitoring Device Data

- Open the Data tab of the test tool, select the device to be monitored, and start monitoring:

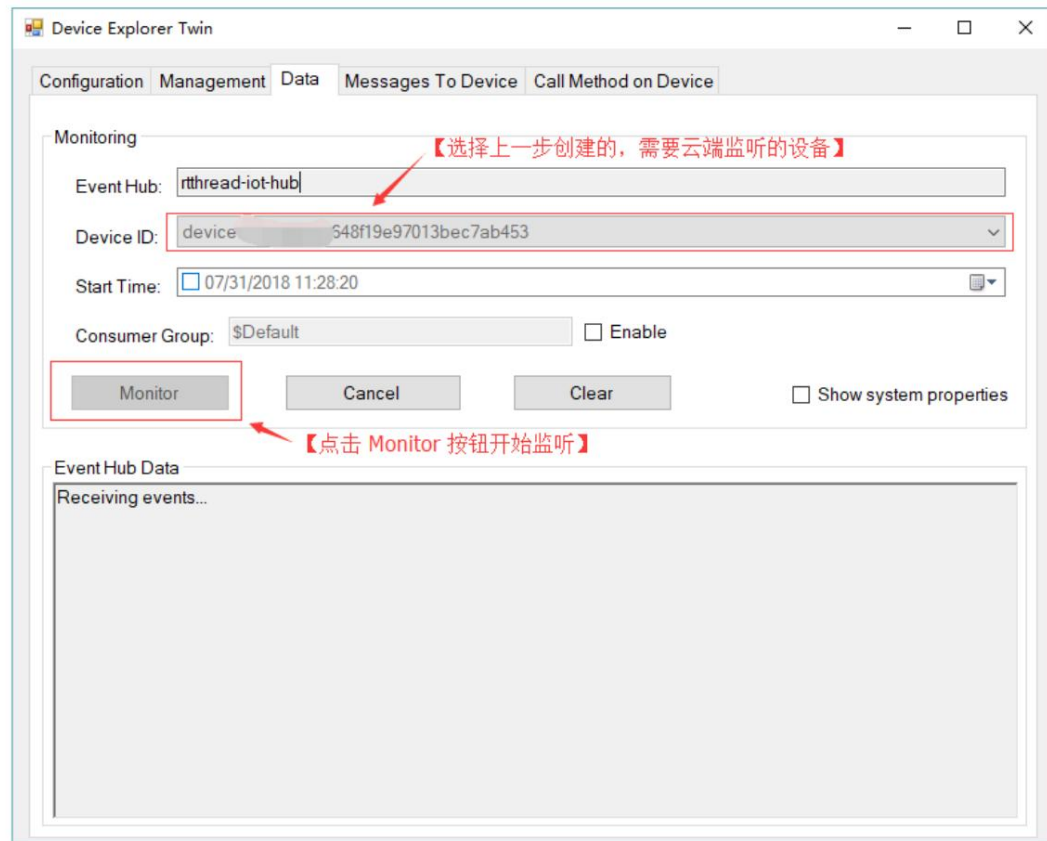


Figure 5.10: Monitoring device telemetry data

5.7.3 Modify the device connection string in the sample code

1. Before running the test example, you need to obtain the device connection string.

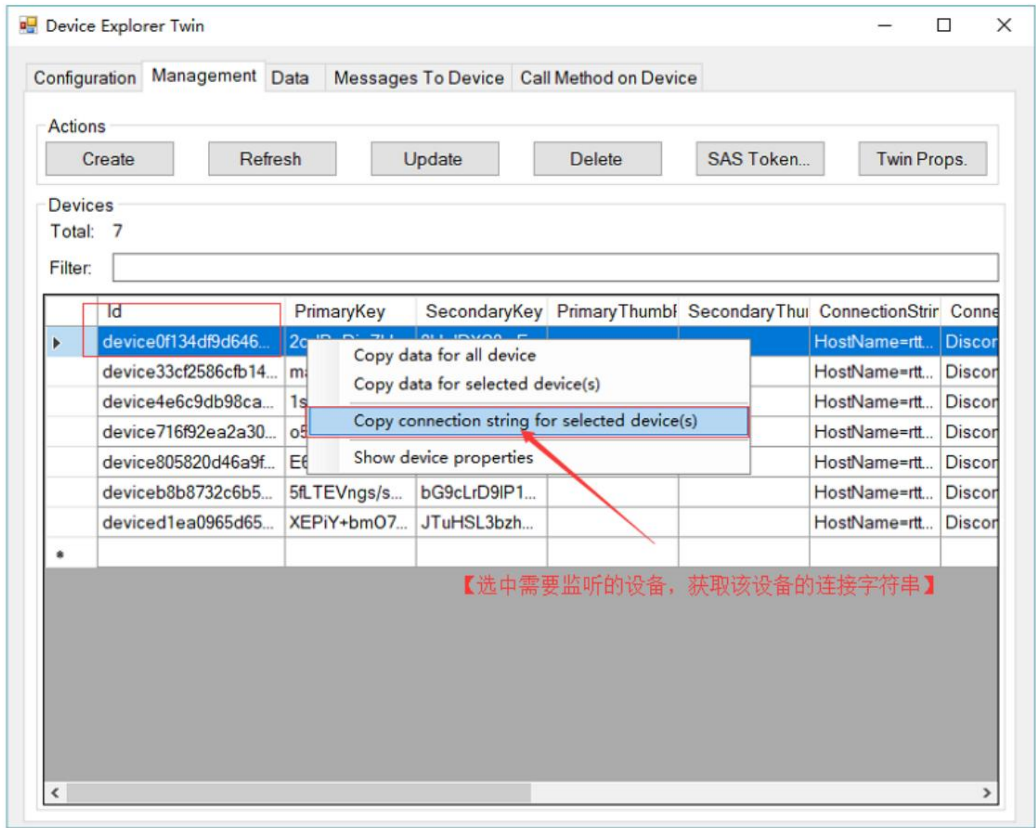


Figure 5.11: Get the device connection string

2. Fill in the connection string in the test example, recompile the program, and download

to the development board.

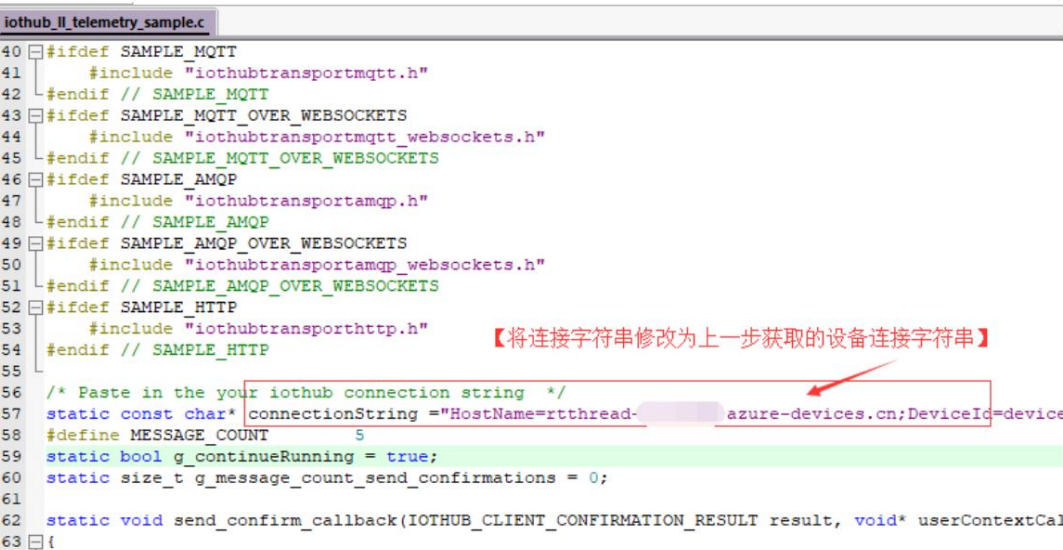


Figure 5.12: Fill in the device connection string

5.7.4 Running the Sample Program

1. Run the `azure_telemetry_sample` sample program in msh:

```
msh />azure_telemetry_sample
msh />
ntp init
Creating IoT Hub Device handle
Sending message 1 to IoT Hub
-> 11:46:58 CONNECT | VER: 4 | KEEPALIVE: 240 | FLAGS: 192 |
USERNAME:
xxxxxxxxx.azuredevices.cn/devicexxxxxxxx9d64648f19e97013bec7ab453
/?api-version=2017-xx-xx-preview&
DeviceClientType=iothubclient%2f1.2.8%20
(native%3b%20xxxxxxxx%3b%20xxxxxx) | PWD: XXXX | CLEAN: 0 <- 11:46:59
CONNACK | SESSION_PRESENT: true | RETURN_CODE: 0x0
The device client is connected to iothub
Sending message 2 to IoT Hub
Sending message 3 to IoT Hub
-> 11:47:03 PUBLISH | IS_DUP: false | RETAIN: 0 | QOS:
    DELIVER_AT_LEAST_ONCE |
    TOPIC_NAME: devices/device0f134df9d64648f19e97013bec7ab453/messages/
        events
    /hello=RT-Thread | PACKET_ID: 2 | PAYLOAD_LEN: 12
-> 11:47:03 PUBLISH | IS_DUP: false | RETAIN: 0 | QOS:
    DELIVER_AT_LEAST_ONCE |
    TOPIC_NAME: devices/device0f134df9d64648f19e97013bec7ab453/messages/
        events
    /hello=RT-Thread | PACKET_ID: 3 | PAYLOAD_LEN: 12
-> 11:47:03 PUBLISH | IS_DUP: false | RETAIN: 0 | QOS:
    DELIVER_AT_LEAST_ONCE |
    TOPIC_NAME: devices/device0f134df9d64648f19e97013bec7ab453/messages/
        events
    /hello=RT-Thread | PACKET_ID: 4 | PAYLOAD_LEN: 12
<- 11:47:04 PUBACK | PACKET_ID: 2
Confirmation callback received for message 1 with result
    IOTHUB_CLIENT_CONFIRMATION_OK
<- 11:47:04 PUBACK | PACKET_ID: 3
Confirmation callback received for message 2 with result
    IOTHUB_CLIENT_CONFIRMATION_OK
<- 11:47:04 PUBACK | PACKET_ID: 4
Confirmation callback received for message 3 with result
    IOTHUB_CLIENT_CONFIRMATION_OK
Sending message 4 to IoT Hub
-> 11:47:06 PUBLISH | IS_DUP: false | RETAIN: 0 | QOS:
```

```
DELIVER_AT_LEAST_ONCE |
  TOPIC_NAME: devices/device0f134df9d64648f19e97013bec7ab453/messages/
    events
    /hello=RT-Thread | PACKET_ID: 5 | PAYLOAD_LEN: 12
<- 11:47:07 PUBACK | PACKET_ID: 5
Confirmation callback received for message 4 with result
  IOTHUB_CLIENT_CONFIRMATION_OK
Sending message 5 to IoT Hub
-> 11:47:09 PUBLISH | IS_DUP: false | RETAIN: 0 | QOS:
  DELIVER_AT_LEAST_ONCE |
  TOPIC_NAME: devices/device0f134df9d64648f19e97013bec7ab453/messages/
    events
    /hello=RT-Thread | PACKET_ID: 6 | PAYLOAD_LEN: 12
<- 11:47:10 PUBACK | PACKET_ID: 6
Confirmation callback received for message 5 with result
  IOTHUB_CLIENT_CONFIRMATION_OK
-> 11:47:14 DISCONNECT
Error: Time:Tue Jul 31 11:47:14 2018 File:packages\azure\azure-port\pal\
  src\
socketio_berkeley.c Func:socketio_send Line:853 Failure: socket state
is not opened.
The device client has been disconnected
Azure Sample Exit
```

2. You can now view the telemetry data sent by the device to the cloud in the Data column of the DeviceExplorer tool:

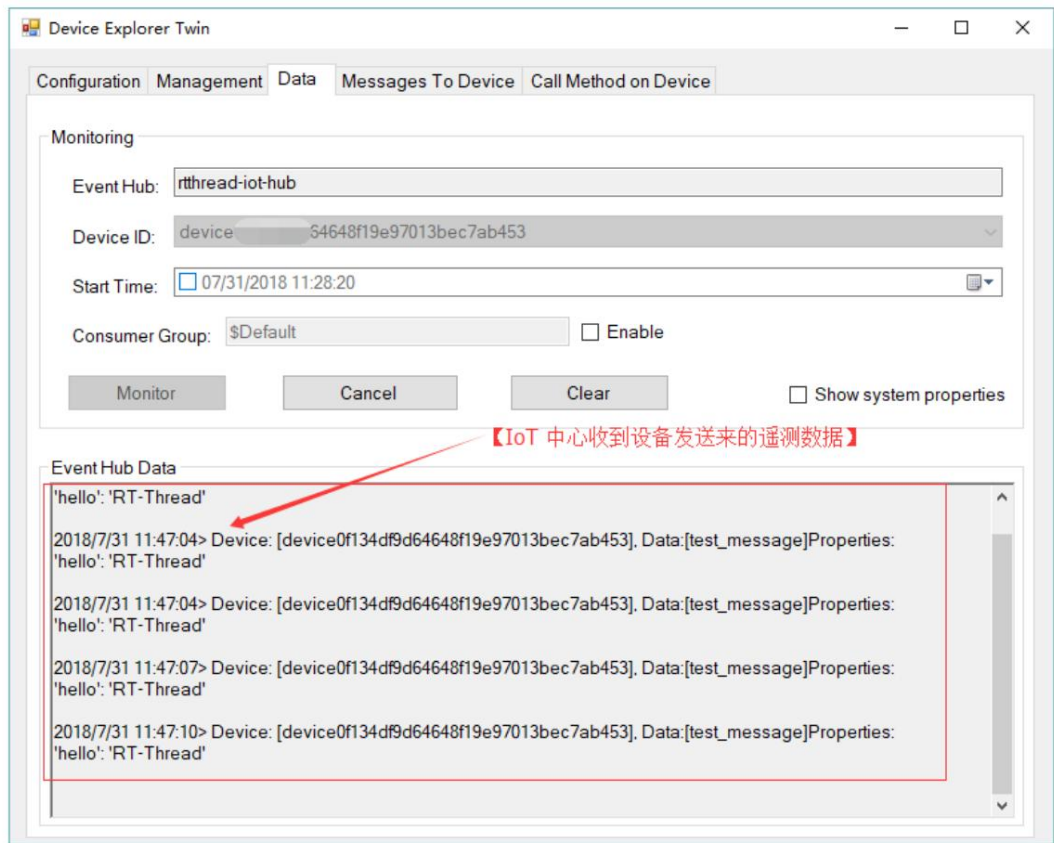


Figure 5.13: Receive telemetry data

The example runs successfully, and in the DeviceExplorer tool, you can see 5 telemetry data sent by the device to the IoT center.

according to

5.8 Function Example 2: Device monitoring data sent from the cloud

5.8.1 Example File

Sample program path	illustrate
samples/iotHub_Ill_c2d_sample.c	Listen to Azure IoT Center on the device side data.

5.8.2 Modify the device connection string in the sample code

- Similar to the above example, this example program also requires you to fill in the correct device connection string.

Compile the program and download it to the development board. The modifications are as follows:

```

34 #ifndef SAMPLE_MQTT_OVER_WEBSOCKETS
35     #include "iothubtransportmqtt_websockets.h"
36 #endif // SAMPLE_MQTT_OVER_WEBSOCKETS
37 #ifndef SAMPLE_AMQP
38     #include "iothubtransportamqp.h"
39 #endif // SAMPLE_AMQP
40 #ifndef SAMPLE_AMQP_OVER_WEBSOCKETS
41     #include "iothubtransportamqp_websockets.h"
42 #endif // SAMPLE_AMQP_OVER_WEBSOCKETS
43 #ifndef SAMPLE_HTTP
44     #include "iothubtransporthttp.h"
45 #endif // SAMPLE_HTTP
46
47 #ifndef SET_TRUSTED_CERT_IN_SAMPLES
48     #include "certs/certs.h"
49 #endif // SET_TRUSTED_CERT_IN_SAMPLES
50
51 /* Paste in the your iothub connection string */
52 static const char* connectionString = "HostName=rtthread-iot-hub.azure-devices.cn;DeviceId
53
54 #define MESSAGE_COUNT 3
55 static bool g_continueRunning = true;
56 static size_t g_message_recv_count = 0;

```

Figure 5.14: Modify the device connection string

5.8.3 Running the sample program on the device

- Run the `azure_telemetry_sample` sample program in msh. After the sample program runs, the device will wait and receive data sent from the cloud:

```

msh /> azure_c2d_sample
msh />
ntp init
Creating IoT Hub Device handle                                #Wait for data to be sent from IoT center
Waiting for message to be sent to device (will quit after 3 messages)

```

5.8.4 The server sends data to the device

1. Open the Messages To Device tab of the Device Explorer tool to send data to the specified device:

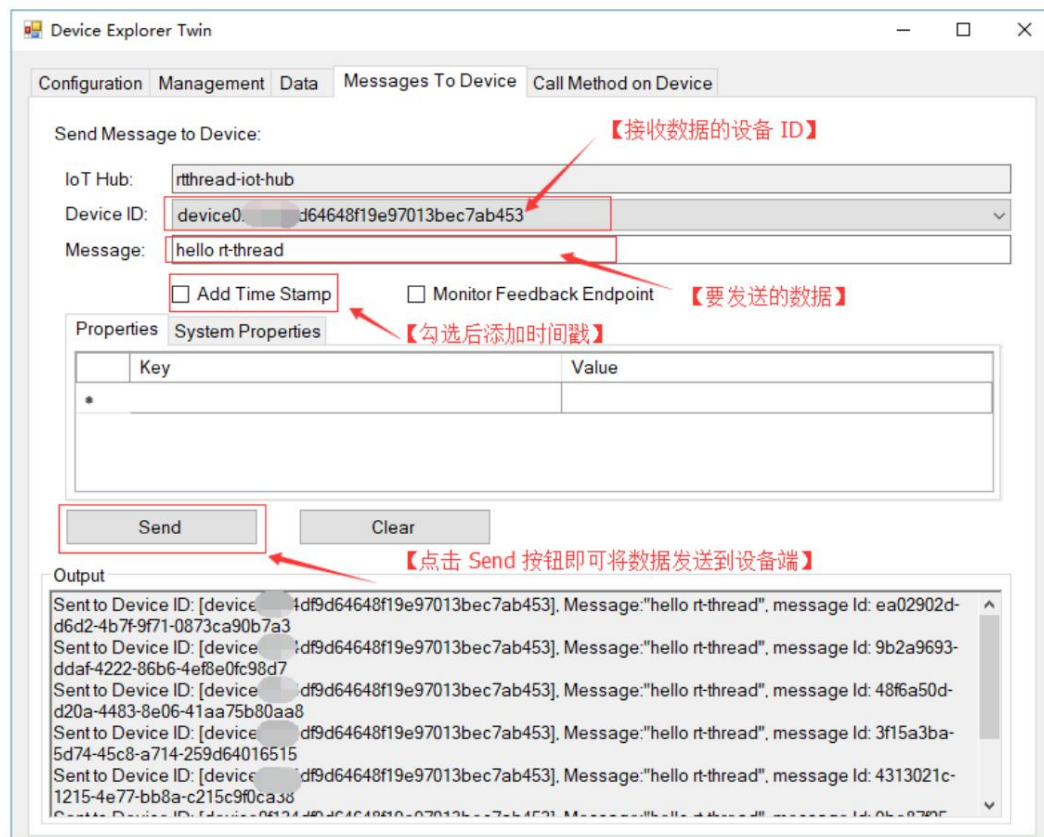


Figure 5.15: The server sends data to the device

2. Now check the data sent from the IoT center to the device on the device side:

```
msh />azure_c2d_sample
msh />
ntp init
Creating IoTHub Device handle
Waiting for message to be sent to device (will quit after 3 messages) #Receive binary data
Received Binary message
Message ID: ea02902d-d6d2-4b7f-9f71-0873ca90b7a3
Correlation ID: <unavailable>
Data: <<<hello rt-thread>>> & Size=15
Received Binary message
Message ID: 9b2a9693-ddaf-4222-86b6-4ef8e0fc98d7
Correlation ID: <unavailable>
Data: <<<hello rt-thread>>> & Size=15
Received Binary message
Message ID: 48f6a50d-d20a-4483-8e06-41aa75b80aa8
Correlation ID: <unavailable>
Data: <<<hello rt-thread>>> & Size=15
Received Binary message
```

```
Message ID: 3f15a3ba-5d74-45c8-a714-259d64016515
Correlation ID: <unavailable>
Data: <<<hello rt-thread>>> & Size=15
Received Binary message
Message ID: 4313021c-1215-4e77-bb8a-c215c9f0ca38
Correlation ID: <unavailable>
Data: <<<hello rt-thread>>> & Size=15
Received Binary message
Message ID: 9be87f25-2a6f-46b5-a413-0fb2f93f85d6
Correlation ID: <unavailable>
Data: <<<hello rt-thread>>> & Size=15
Error: Time:Tue Jul 31 13:54:14 2018
File:packages\azure\azure-port\pal\src\socketio_berkeley.c Func:socketio_send
Line:853 Failure: socket state is not opened.
Azure Sample Exit #After receiving a certain amount of data, the functional sample automatically exits.
```