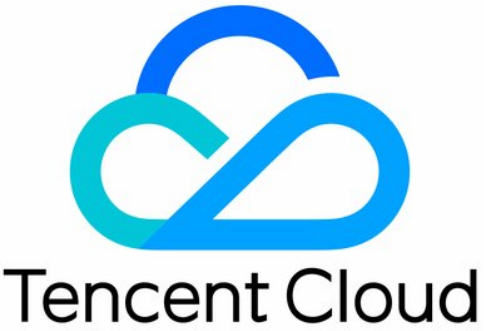
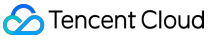
Internet of Things Hub
Device Connection Manual



Copyright Notice

Trademark Notice



Service Notice

This document provides an overview of the as-is details of Tencent Cloud's products and services in their entirety or part. The descriptions of certain products and services may be subject to adjustments from time to time.

The commercial contract concluded by you and Tencent Cloud will provide the specific types of Tencent Cloud products and services you purchase and the service standards. Unless otherwise agreed upon by both parties, Tencent Cloud does not make any explicit or implied commitments or warranties regarding the content of this document.

Contact Us

We are committed to providing personalized pre-sales consultation and technical after-sale support. Don't hesitate to contact us at 4009100100 or 95716 for any inquiries or concerns.

Contents

Device Connection Manual
- Device Connection Overview
- Connection Based on SDK for C
  - SDK for C Download
  - SDK for C Cross-Platform Porting
    - Overview
    - FreeRTOS + lwIP Platform Porting Description
    - MCU + Universal TCP_AT Module Porting (FreeRTOS)
    - MCU + Universal TCP_AT Module Porting (nonOS)
  - SDK for C Connection Description
  - SDK for C Use Instructions
    - Usage Overview
    - Compilation Configuration Description
    - Compilation Environment (Linux and Windows)
    - Getting Started with MQTT
    - API and Variable Parameter Description
    - Device Information Storage
- Connection Based on SDK for Android
  - SDK for Android Release Notes
  - SDK for Android Project Configuration
  - SDK for Android Use Instructions
- Connection Based on SDK for Java
  - SDK for Java Release Notes
  - SDK for Java Project Configuration
  - SDK for Java Use Instructions
- Connection Based on SDK for Python
  - Python SDK Release Notes
  - SDK for Python Project Configuration
  - SDK for Python Use Instructions

Device Connection Manual
Device Connection Overview

Last updated: 2025-03-19 14:59:29

Feature Overview

To facilitate the connection of your devices and ensure the security of connection, IoT Hub provides a complete device connection service. To connect a device to IoT Hub, you need to complete device registration/creation first. The process of connection to IoT Hub be completed only after the device registration/creation succeeds.

Device connection service

- The device connection service provides the feature of dynamic device registration, so device registration can be completed by devices themselves.
- The device connection service supports connection over diverse protocols, including MQTT, WebSocket, HTTP/HTTPS, and CoAP.
- The device connection service is capable of connection authentication, so devices need to be authenticated based on the connection protocol to ensure the connection security.
- The device connection service offers device SDKs, based on which devices can be connected easily.

Device connection based on SDK

IoT Hub provides SDKs for C SDK, SDK for Android, SDK for Java, and SDK for Python. The SDKs integrate the features included in the device connection service. Users only need to set the device information in the SDK (for key devices: ProductID, DeviceName, DeviceKey; for certificate devices: ProductID, DeviceName, certificate file, key file, CA certificate) and integrate the corresponding features of the SDK into the device to complete the device connection. In addition to the connection service features, the SDK also includes features such as device shadow, OTA, and RRPC. For details on the interfaces, please refer to the relevant documents in **SDK Use Instructions**.

- SDK for C Use Instructions.
- SDK for Android Use Instructions.
- SDK for Java Use Instructions.
- SDK for Python Use Instructions.

> **Note:**

> IoT Hub supports custom connection. You can connect devices to it in a custom way simply by following the protocols and authentication processes it provides.

Connection Based on SDK for C
SDK for C Download

Last updated: 2025-03-19 14:59:50

Code Hosting

- Since V1.0.0 version, the device SDK code is hosted on Github
  https://github.com/tencentyun/qcloud-iot-sdk-embedded-c
- Download the latest version
  https://github.com/tencentyun/qcloud-iot-sdk-embedded-c/releases

Version V3.3.0

- Release date: 2022/9/26
- Programming language: C
- Development environment: Linux/Windows/FreeRTOS, etc.
- Content:

  1. Fixed several bugs.
  2. Added some features, such as: remote SSH login, WebSocket support, etc. See documentation for details.

Version V3.2.3

- Release date: 2020/11/03
- Programming language: C
- Development environment: Linux/Windows
- Content:

  1. Added a feature interface to synchronize NTP time from the MQTT backend.
  2. Added gateway sub-device OTA example.
  3. Optimized Log/HAL_Printf handling of string pointers.
  4. Fixed bugs.

V3.2.2 version

- Release date: October 14, 2020
- Programming language: C
- Development environment: Linux/Windows
- Content:

1. Added remote configuration function and examples.
2、The gateway now includes a feature to fetch the list of sub-devices.
3、Bug fixes.

Version V3.2.1

- Release date: August 4, 2020
- Programming language: C
- Development environment: Linux/Windows
- Content:
  1. Added rrpc synchronous communication feature and examples.
  2. Added broadcast feature and examples.
  3. The gateway now includes bind/unbind sub-devices feature.

Version V3.2.0

- Release date: April 30, 2020
- Programming language: C
- Development environment: Linux/Windows
- Content:
  1. Merged mtmc branch code, supporting multi-device connection and optimizing multi-threaded interface.
  2. Fixed some potential memory leaks and out-of-bounds issues, as well as cross-platform compilation and running issues.
  3. Used clang-format to format the code, introduced code checking tools clang-tidy and cpplint.

Version V3.1.3

- Release date: March 6, 2020
- Programming language: C
- Development environment: Linux/Windows
- Content:
  1. Optimize ota_mqtt_sample by decoupling OTA process and file operations, and ensure the sample supports resuming download after MQTT disconnect and reconnect.
  2. Optimize gateway_sample and add example code for proxying more than one sub-device.
  3. Add interface to check MQTT topic subscription success.
  4. Optimize and update documentation.
  5. Fix some compilation warnings and bugs.
  6. Unify code indentation style.

Version V3.1.2

- Publish date: 2019 / 11 / 11
- Programming language: C
- Development environment: Linux/Windows
- Content:
  1. Remove code and documentation related to IoT Explorer platform, support IoT Hub only. Optimize documentation description.
  2. Bug fixes: OTA module memory leak, device_info.json file parsing issue, and Windows platform time format issue.
  3. Avoid filename conflict, rename ca.c/h to qcloud_iot_ca.c/h, device.c/h to qcloud_iot_device.c/h.

Version V3.1.0

- Publish date: 2019 / 09 / 19
- Programming language: C
- Development environment: Linux/Windows
- Content:
  C-SDK Refactoring:
  1. Optimize code architecture and directory hierarchy, use English comments, improve document explanation, and enhance availability and portability.
  2. Add cmake compilation method and code extraction method on top of the original makefile compilation, adapting to various compilation environments.
  3. Add Windows platform support, supporting development under Microsoft Visual Studio.
  4. Add AT_socket network layer to support the development and porting of MCU+TCP AT module devices.
  5. Add porting adaptation for the FreeRTOS+lwIP platform.

Version V3.0.3

- Publish date: 2019 / 08 / 26
- Programming language: C
- Development Environment: Linux, GNU Make.
- Content:
  1. Support OTA breakpoint resume: ota_mqtt_sample.c example adds local firmware version information management (version, breakpoint, MD5), firmware download supports range parameter when establishing HTTPS connection.
  2. SDK version number updated to V3.0.3.

Version V3.0.2

- Publish date: 2019 / 07 / 18
- Programming language: C
- Development Environment: Linux, GNU Make.
- Content:
  1. Data template string type supports escape character processing.
  2. Remove device side version management from device shadow.
  3. Optimize data template related examples.

Version V3.0.1

- Publish date: 2019 / 06 / 11
- Programming language: C
- Development Environment: Linux, GNU Make.
- Content:
  1. Optimized log reporting function, dynamically allocates buffer memory, supports larger log segment reporting, suitable for various scenarios.
  2. Added event handler callback for MQTT subscribe, promptly notifies the status change of subscribed topics.
  3. Fixed some code issues, such as improper return value judgment for MQTT API.

Version V3.0.0

- Publish date: 2019–05–17
- Programming language: C
- Development Environment: Linux, GNU Make.
- Content:
  1. Added data template function based on shadow.
  2. Added event reporting function.
  3. Added data template code generation script tool.
  4. Fixed several bugs in JSON processing.
  5. Added data template example, event example, data template smart light scene example.
  6. Adjusted document structure, added document directory docs and platform SDK application instructions.
  7. Versions V3.0.0 and later support both Internet of Things Hub (IoT Hub) and internet of things development platforms.

Version V2.3.5

- Publish date: 2019–05–15
- Programming language: C

- Development Environment: Linux, GNU Make.
- Content:
  1. Added device dynamic registration function.
  2. Added device dynamic registration example.
  3. Added device information read and write hal interface.
  4. Added aes encryption and decryption api.
  5. Changed the method of acquiring sample device information to hal layer interface implementation.

Version v2.3.3

- Publish date: 2019-05-06
- Programming language: C
- Development Environment: Linux, GNU Make.
- Content:
  1. Optimized MQTT keep alive connection mechanism and PING request packet sending strategy.
  2. Modified MQTT subscription/unsubscription topic names to use dynamic memory for storage, making it easier for interface callers to use.
  3. Changed the maximum length of topic names to 128 to align with the cloud backend.
  4. Fixed bugs in HTTPC and MQTT when fetching sys and log messages.
  5. Optimized error code types.

Version v2.3.2

- Publish date: 2019-04-12
- Programming language: C
- Development Environment: Linux, GNU Make.
- Content:
  1. Fixed experience issues: added gateway compilation options in make.settings (default off) and modified the print level for firmware upgrades.
  2. Fixed loss issues in MQTT receive buffer when handling shadow messages downstream: added an error prompt when the receive buffer is insufficient, and changed the default size of the MQTT send and receive buffer to 2048 bytes.
  3. Changed the maximum number of successful subscription to topics to 10.

Version v2.3.1

- Publish date: 2019-03-12
- Programming language: C
- Development Environment: Linux, GNU Make.

- Content:
  1. The SDK now includes a device log reporting function, allowing users to remotely monitor and diagnose device network connectivity through the cloud console. Currently, only MQTT mode is supported.
  2. Streamlined SDK log print content, fixed several bugs, and optimized some code design.
  3. Changed the maximum length of device names to 48 characters, consistent with the IoT Hub cloud console.

Version V2.3.0

- Publish date: 2019 / 02 / 25
- Programming language: C
- Development Environment: Linux, GNU Make.
- Content:
  1. Added gateway functionality, supporting gateway devices to proxy sub-device online/offline and send and receive messages based on MQTT protocol.
  2. For multithreaded applications, optimized thread-safe design, added multithreaded routines and notes and instructions in samples.
  3. Optimized MQTT reconnection mechanism and heartbeat packet timer refresh strategy.
  4. Fixed several bugs, added legality check for some memory operations.
  5. Removed bit field operation method in several structures to reduce cross-platform errors.

Version V2.2.0

- Publish date: 2018 / 07 / 20
- Programming language: C
- Development Environment: Linux, GNU Make.
- Content:
  1. Added NBIoT device access capability.
  2. Adapted TOPIC's wildcards "#" and "+".
  3. Organized the directory structure of third-party libraries.
  4. Fixed several bugs.

Version V2.1.0

- Publish date: 2018-05-02
- Programming language: C
- Development Environment: Linux, GNU Make.
- Content:
  1. Added firmware upgrade (OTA-CoAP channel) capability.

2. Added low-end resource-constrained device hmac-sha1 authentication access capability.
3. Added the ability to obtain background time.

Version V2.0.0

- Publish date: 2018-03-12
- Programming language: C
- Development Environment: Linux, GNU Make.
- Content:
  1. Added firmware upgrade (OTA-MQTT channel) capability.
  2. Fixed the issue where the device shadow heartbeat interval was ineffective.
  3. Fixed the buffer overflow issue caused by MQTT received data length at the critical value.

Version V1.2.2

- Publish date: 2018-02-07
- Programming language: C
- Development Environment: Linux, GNU Make.
- Content:
  1. Added MQTT/CoAP symmetric encryption connection support.
  2. Linux C compilation optimization.

Version V1.2.1

- Release date: 2018-02-02
- Programming language: C
- Development Environment: Linux, GNU Make.
- Content: Fixed the erroneous logic in the message timeout callback for Publish.

V1.2.0 version

- Release date: 2018-01-17
- Programming language: C
- Development Environment: Linux, GNU Make.
- Content:
  1. Transformed the ACK for publish/subscribe messages to be received through callback, preventing blocking of the send thread.
  2. Added capabilities for terminal and backend regarding connections and log correspondence.

3. Introduced a CoAP channel, based on UDP, using DTLS asymmetric encryption, consuming less energy in pure data reporting scenarios.

V1.0.0 version

- Release date: 2017-11-15
- Programming language: C
- Development Environment: Linux, GNU Make.
- Content:
  1. MQTT Protocol support: Enables devices to quickly and easily connect to the IoT Hub cloud server, see MQTT Protocol Description.
  2. Device shadow function support: For details, see Device Shadow Details.
  3. Provides support for both symmetric and asymmetric encryption methods.

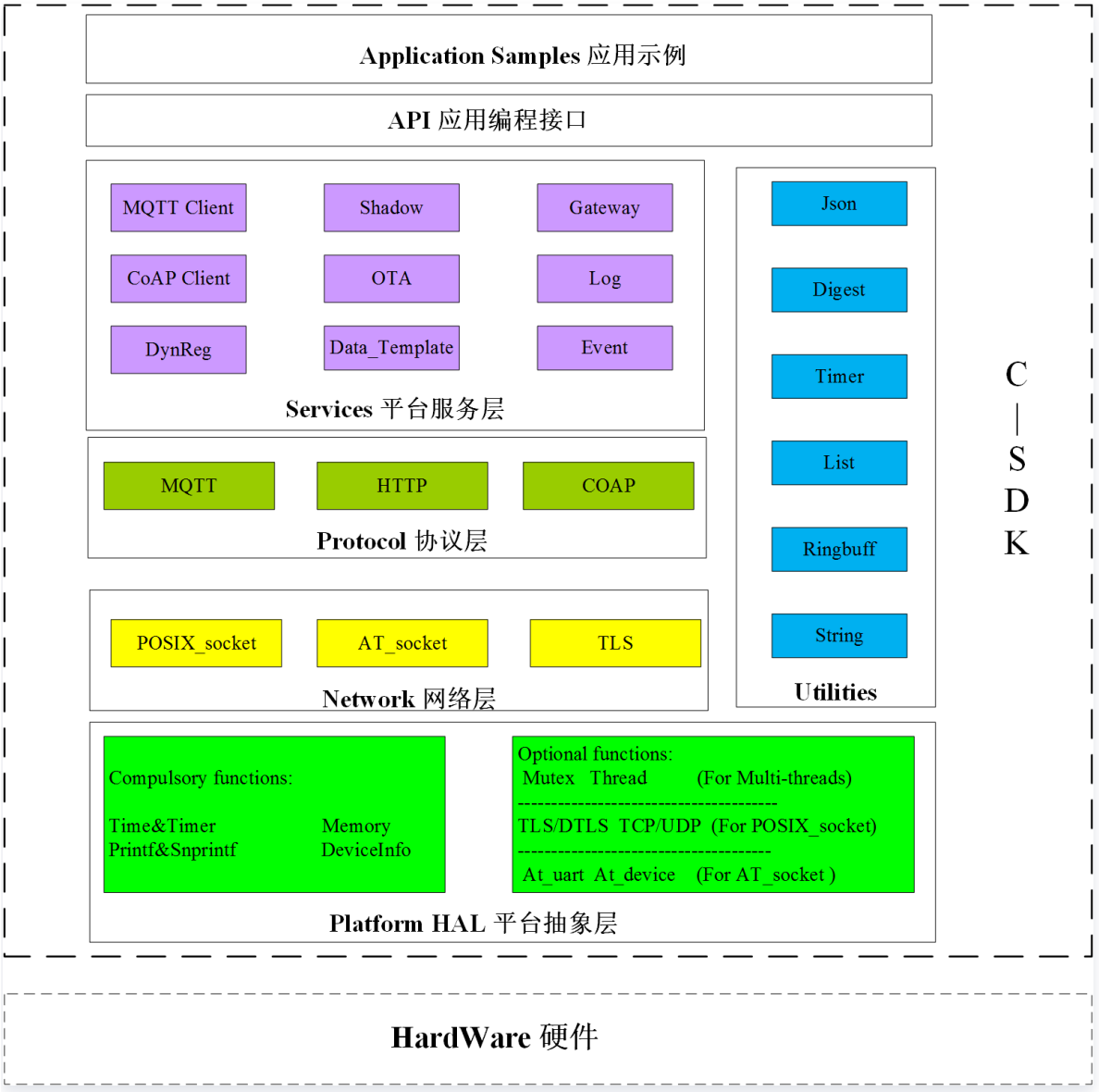
SDK for C Cross-Platform Porting Overview

Last updated: 2025-03-19 15:00:13

This document describes how to port the device C-SDK to the target hardware platform. C-SDK adopts modular design to separate the core protocol service and hardware abstraction layer (HAL). When porting across platforms, you generally only need to modify and adapt the HAL.

C-SDK Architecture

Architecture Diagram



Architecture Description

The SDK is designed into four layers from top to bottom: platform service layer, core protocol layer, network layer, and hardware abstraction layer.

- **Service layer**
  This layer is above the network protocol layer and implements features such as device connection authentication, device shadow, gateway, dynamic registration, log reporting, and OTA.
- **Protocol layer**
  The network protocols over which devices can interact with the IoT Hub platform include MQTT, CoAP, and HTTP.
- **Network layer**
  This layer implements network protocol stacks based on TLS/SSL (TLS/DTLS), POSIX_socket (TCP/UDP), and AT_socket. Different services can use different protocol stack API functions as needed.
- **Hardware abstraction layer**
  This layer abstracts underlying operations for different hardware platforms. Porting is required for specific software and hardware platforms, divided into mandatory and optional HAL interface implementations.

HAL porting

HAL mainly has several major parts for porting, including those related to the OS, network and TLS, time and print, and device information.
In the **platform/os** directory, the SDK demonstrates the implementation of HAL in four scenarios: Linux, Windows, FreeRTOS, and nonOS. You can refer to the corresponding directory to port for the target platform.

OS relevant interfaces

Serial number	Function Name	Description
1	HAL_Malloc	Dynamically applies for memory block.
2	HAL_Free	Releases memory block.
3	HAL_ThreadCreate	Thread creation.
4	HAL_ThreadDestroy	Thread destruction.
5	HAL_MutexCreate	Creates mutex lock.

6	HAL_MutexDestroy	Terminates mutex lock.
7	HAL_MutexLock	Mutex locking.
8	HAL_MutexUnlock	Mutex unlocking.
9	HAL_SemaphoreCreate	Creates semaphore.
10	HAL_SemaphoreDestroy	Terminates semaphore.
11	HAL_SemaphoreWait	Waits for semaphore.
12	HAL_SemaphorePost	Releases semaphore.
13	HAL_SleepMs	Sleeps.

Network and TLS HAL APIs

Network-related interfaces provide two options for adaptation and porting. For devices with network communication capabilities and integrated TCP/IP network protocol stack, you need to implement the POSIX_socket network HAL interface. If using TLS/SSL encrypted communication, you also need to implement the TLS-related HAL interface. For **MCU + Universal TCP_AT Module** devices, you can choose the AT_Socket framework provided by the SDK and implement the related AT module interface.

POSIX_socket-based HAL interface

Among them, TCP/UDP APIs are implemented based on POSIX socket functions. TLS APIs are dependent on the **mbedtls** library. Before porting, you must ensure that the **mbedtls** library is available on the system. If you use other TLS/SSL libraries, please refer to the relevant implementation of **platform/tls/mbedtls** for porting and adapting.

UDP/DTLS functions need to be ported only when **CoAP** communication is enabled.

Serial number	Function Name	Description
1	HAL_TCP_Connect	Establishes TCP connection.
2	HAL_TCP_Disconnect	Closes TCP connection.
3	HAL_TCP_Write	TCP write.
4	HAL_TCP_Read	TCP read.
5	HAL_TLS_Connect	Establishes TLS connection.
6	HAL_TLS_Disconnect	Closes TLS connection.

7	HAL_TLS_Write	TLS write.
8	HAL_TLS_Read	TLS read.
9	HAL_UDP_Connect	Establishes UDP connection.
10	HAL_UDP_Disconnect	Closes UDP connection.
11	HAL_UDP_Write	UDP write.
12	HAL_UDP_Read	UDP read.
13	HAL_DTLS_Connect	Establishes DTLS connection.
14	HAL_DTLS_Disconnect	Closes DTLS connection.
15	HAL_DTLS_Write	DTLS write.
16	HAL_DTLS_Read	DTLS read.

HAL APIs based on AT_socket

By enabling the compilation macro **AT_TCP_ENABLED** to select AT_socket, the SDK will call the `at_socket` API of `network_at_tcp.c`. You don't need to port the at_socket layer, but you need to implement the AT serial port driver and AT module driver. For the AT module driver, you only need to implement the driver API of the driver structure `at_device_op_t` in at_device of the AT framework. You can refer to the supported modules in the at_device directory. The AT serial port driver needs to implement interrupt reception of the serial port, and then call the callback function `at_client_uart_rx_isr_cb` in the interrupt service routine. You can refer to `HAL_AT_UART_freertos.c` to port to the target platform.

Serial number	Function Name	Description
1	HAL_AT_Uart_Init	Initializes AT serial port.
2	HAL_AT_Uart_Deinit	Deinitializes AT serial port.
3	HAL_AT_Uart_Send	Sends data over AT serial port.
4	HAL_AT_UART_IRQHandler	Handles AT serial port receipt interruption.

Time and print HAL APIs

Serial number	Function Name	Description
1	HAL_Printf	Writes formatted data to standard output stream.
2	HAL_Snprintf	Writes formatted data to string.
3	HAL_UptimeMs	Retrieves the number of milliseconds that elapsed since the system has started.
4	HAL_DelayMs	Blocking delay in milliseconds.

Device information HAL APIs

To connect a device to the IoT Hub platform, you need to create product and device information on the platform and save such information in a non-volatile storage medium on the device. You can refer to `platform/os/linux/HAL_Device_linux.c` for implementation.

Serial number	Function Name	Description
1	HAL_GetDevInfo	Reads device information.
2	HAL_SetDevInfo	Saves device information.

FreeRTOS + lwIP Platform Porting Description

Last updated: 2025-03-19 15:00:26

This document describes how to port IoT Hub C-SDK to the **FreeRTOS + lwIP** platform.

FreeRTOS Porting Overview

As a micro-kernel system, FreeRTOS mainly provides core OS mechanisms such as task creation and scheduling and inter-task communication. Different device platforms also should be equipped with different software components before they can form a complete embedded operating platform, including C runtime libraries (such as Newlib or ARM CMSIS library) and TCP/IP network protocol stacks (such as lwIP). In addition, the compilation and development environments vary by device platform, so when porting C-SDK, you need to adapt it according to the specific conditions of different devices.

> **Note:**
> The SDK provides a reference implementation based on **FreeRTOS + lwIP + Newlib** in **platform/os/freertos**, which has been verified and tested on the Espressif ESP8266 platform.

Code Extraction

Because different RTOS-based platforms have different compilation methods, it is generally impossible to directly use the SDK's CMake or Make to compile. Therefore, the SDK provides the code extraction feature. It allows you to extract the relevant code into a separate folder based on your needs. The code hierarchy in the folder is concise, making it easy for you to copy and integrate it into your own development environment.

1. Modify the CMakeLists.txt configuration to the FreeRTOS platform and enable the code extraction feature:

```
set(BUILD_TYPE                  "release")
set(PLATFORM                    "freertos")
set(EXTRACT_SRC ON)
set(FEATURE_AT_TCP_ENABLED OFF)
```

2. Run the following command on Linux:

```
mkdir build
```

```
cd build
cmake ..
```

3. You can find the relevant code files in output/qcloud_iot_c_sdk, with the directory structure as follows:

```
qcloud_iot_c_sdk
├── include
│   ├── config.h
│   ├── exports
├── platform
└── sdk_src
 └── internal_inc
```

> **Note:**
> - include directory: SDK APIs and variable parameters for user use, with config.h being the compilation macro generated by the compilation options. For a detailed introduction of the API, please refer to API and Variable Parameter Description.
> - platform directory: Platform-related code can be modified and adapted according to the specific situation of the device. For specific function descriptions, please refer to C-SDK_Porting Cross-Platform Porting Overview.
> - sdk_src: The core logic and protocol-related code of the SDK generally do not need modification. The internal_inc directory contains header files used internally by the SDK.

4. Users can copy qcloud_iot_c_sdk to their target platform's compilation development environment and modify the compilation options according to the specific situation.

Porting Sample

Build a demo project based on Espressif's ESP8266 RTOS platform in the Linux development environment.

1. Please refer to ESP8266_RTOS_SDK to obtain the RTOS_SDK and cross compiler, and create a project.
2. Copy the extracted qcloud_iot_c_sdk directory to components/qcloud_iot.
3. In components/qcloud_iot, create a new compilation configuration file named component.mk with the following content:

```
#
```

```
# Component Makefile
#
COMPONENT_ADD_INCLUDEDIRS := \
qcloud_iot_c_sdk/include \
qcloud_iot_c_sdk/include/exports \
qcloud_iot_c_sdk/sdk_src/internal_inc
COMPONENT_SRCDIRS := \
qcloud_iot_c_sdk/sdk_src \
qcloud_iot_c_sdk/platform
```

At this point, you can compile qcloud_iot_c_sdk as a component, and then you can call the IoT Hub C-SDK interface in the user code to connect and send and receive messages.

MCU + Universal TCP_AT Module Porting (FreeRTOS)

Last updated: 2025-03-19 15:00:38

For MCUs without network communication capability, the MCU+Communication Module method is generally adopted. The communication module (including Wi-Fi/2G/4G/NB-IoT) usually provides a serial AT command protocol for the MCU to perform network communications. For this scenario, the C-SDK encapsulates the AT-socket network layer. The core protocol and service layer above the network layer do not need to be ported. This document describes how to port the C-SDK and connect to the Tencent Cloud IoT Platform for an MCU (FreeRTOS) + General TCP AT module target environment.

SDK Download

Download the latest version of the C-SDK for the device side.

SDK Feature Configuration

Use the general TCP module to compile and configure the options as follows:

Name	Configuration	Description
BUILD_TYPE	debug/release	Set as needed.
EXTRACT_SRC	ON	Enable code extraction.
COMPILE_TOOLS	gcc/MSVC	Set as needed and ignore in case of IDE.
PLATFORM	Linux/Windows	Set as needed and ignore in case of IDE.
FEATURE_OTA_COMM_ENABLED	ON/OFF	Set as needed.
FEATURE_AUTH_MODE	KEY	Key authentication is recommended for resource-constrained devices.
FEATURE_AUTH_WITH_NOTLS	ON/OFF	Enable TLS as needed.
FEATURE_EVENT_POST_ENABLED	ON/OFF	Enable event reporting as needed.

FEATURE_AT_TCP_ENABLED	ON	Whether to enable TCP feature in AT module.
FEATURE_AT_UART_RECV_IRQ	ON	Whether to enable receipt interruption feature in AT module.
FEATURE_AT_OS_USED	ON	Whether to enable multithreaded feature in AT module.
FEATURE_AT_DEBUG	OFF	The AT module debugging feature is disabled by default, and it needs to be enabled during debugging.

Code Extraction

1. Run the following command on Linux:

```
mkdir build
cd build
cmake ..
```

2. You can find the relevant code files in output/qcloud_iot_c_sdk, with the directory structure as follows:

```
qcloud_iot_c_sdk
├── include
│   ├── config.h
│   ├── exports
├── platform
└── sdk_src
 └── internal_inc
```

Note:

- include directory: APIs and variable parameters provided by the SDK for users; config.h is generated based on compilation options.
- platform directory: Platform-related code that can be modified for specific device adaptation.
- sdk_src: Core logic and protocol-related code of the SDK, which generally do not need modification. internal_inc contains header files for internal SDK use.

3. Users can copy qcloud_iot_c_sdk to their target platform's compilation development environment and modify the compilation options according to the specific situation.

HAL Porting

Please refer to C-SDK_Porting Cross-Platform Porting Overview for porting first.
For network-related HAL interfaces, the `AT_Socket` framework provided by SDK has been chosen through the above compilation options. The SDK will call the `network_at_tcp.c` 's `at_socket` interface. The `at_socket` layer does not need to be ported, but the implementation of AT serial port driver and AT module driver is required. For the AT module driver, only the driver structure `at_device` in the AT framework and the driver interface `at_device_op_t` need to be implemented. You can refer to the supported modules in the `at_device` directory.
Currently, the SDK provides the lower-level interface implementation for the widely used Wi-Fi module ESP8266 in IoT, serving as a reference for porting to other communication modules.

Business Logic Development

You can refer to the examples in the SDK samples directory for development.

MCU + Universal TCP_AT Module Porting (nonOS)

Last updated: 2025-03-19 15:00:50

For MCUs without network communication capability, the MCU + communication module approach is generally used. Communication modules (including Wi-Fi/2G/4G/NB-IoT) usually provide AT command protocol based on serial ports for MCU to communicate with the network. For this scenario, the C-SDK encapsulates the AT-socket network layer, and the core protocol and service layer above the network layer do not need to be ported. This article explains how to port the C-SDK and connect to the Tencent Cloud IoT Platform for the target environment of MCU (no OS) + generic TCP AT module.
Compared with scenarios with RTOS, there will be differences in the processing of network data reception by the at_socket. The application layer needs to periodically call **IOT_MQTT_Yield** to receive server downlink data. If the reception window is missed, data loss may occur. Therefore, it is recommended to use RTOS in scenarios with complex business logic and configure FEATURE_AT_OS_USED = OFF to select the non-OS mode.

SDK Download

Download the latest version of the C-SDK for the device side.

SDK Feature Configuration

Use the general TCP module to compile and configure the options for nonOS as follows:

Name	Configuration	Description
BUILD_TYPE	debug/release	Set as needed.
EXTRACT_SRC	ON	Enable code extraction.
COMPILE_TOOLS	gcc/MSVC	Set as needed and ignore in case of IDE.
PLATFORM	Linux/Windows	Set as needed and ignore in case of IDE.
FEATURE_OTA_COMM_ENABLED	ON/OFF	Set as needed.
FEATURE_AUTH_MODE	KEY	Key authentication is recommended for resource-

		constrained devices.
FEATURE_AUTH_WITH_NOTLS	ON/OFF	Enable TLS as needed.
FEATURE_EVENT_POST_ENABLED	ON/OFF	Enable event reporting as needed.
FEATURE_AT_TCP_ENABLED	ON	Enable at_socket Component.
FEATURE_AT_UART_RECV_IRQ	ON	Enable AT serial port receipt interruption.
FEATURE_AT_OS_USED	OFF	The at_socket component is used in a non-RTOS environment.
FEATURE_AT_DEBUG	OFF	The AT module debugging feature is disabled by default, and it needs to be enabled during debugging.

Code Extraction

1. Run the following command on Linux:

```
mkdir build
cd build
cmake ..
```

2. You can find the relevant code files in output/qcloud_iot_c_sdk, with the directory structure as follows:

```
qcloud_iot_c_sdk
├── include
│   ├── config.h
│   ├── exports
├── platform
└── sdk_src
 └── internal_inc
```

Notes:

- include directory: SDK provides APIs and variable parameters for users. Among them, config.h is a compilation macro generated according to the compilation

options.

- platform directory: Platform-related code that can be modified for specific device adaptation.
- sdk_src: Core logic and protocol-related code of the SDK, which generally do not need modification. internal_inc contains header files for internal SDK use.

3. Users can copy qcloud_iot_c_sdk to their target platform's compilation development environment and modify the compilation options according to the specific situation.

HAL Porting

Please refer to C-SDK_Porting Cross-Platform Porting Overview first.
For network-related HAL interfaces, the SDK provided AT_Socket framework has been selected through the above compilation options. The SDK will call the `network_at_tcp.c` 's `at_socket` interface. The `at_socket` layer does not require porting. It is necessary to implement the AT serial port driver and AT module driver. For the AT module driver, only the driver structure body `at_device` in the AT framework and the driver interface `at_device_op_t` need to be implemented. You can refer to the supported modules under the `at_device` directory. The AT serial driver needs to implement interrupt reception for the serial port, and then call the callback function `at_client_uart_rx_isr_cb` in the interrupt service program, which can be referred to the `HAL_OS_nonos.c` for porting to the target platform.

Business Logic Development

You can refer to the examples in the SDK samples directory for development.

SDK for C Connection Description

Last updated: 2025-03-19 15:01:03

To ensure security, the Internet of Things platform verifies the legitimacy of each access device and provides various authentication methods to meet different usage scenarios and device access for various resources. You can watch the following video to learn about Tencent Cloud IoT Hub device access instructions.
Watch video

Device Identity Information

Based on the form of device key, devices are divided into certificate devices and key devices. The certificate method is more secure but requires more software and hardware resources.

- Certificate-authenticated devices must carry the following four pieces of information before it can pass the authentication by the platform: product ID (ProductId), device name (DeviceName), device certificate (DeviceCert), and device private key (DevicePrivateKey), among which, the certificate and private key files are generated by the platform and correspond to each other.
- Key-authenticated devices must carry the following three pieces of information before it can pass the authentication by the platform: product ID (ProductId), device name (DeviceName), and device key (DeviceSecret), among which, the device key is generated by the platform.

The device key is determined by setting the authentication method when creating a product, as shown in the figure below:

Device Identity Information Burning

Device information burning is divided into preset burning and dynamic burning, which differ in terms of convenience and security.

Preset burning

After a product is created, you can create devices one by one in the IoT Hub console or through TencentCloud API, get their corresponding device information, and burn the above three or four pieces of information into a non-volatile medium in a specific step of device production, so that the device SDK can read the stored device information during running for device authentication.

Dynamic burning

- Preset burning: this involves performing personalized production actions in the mass production process and thus affects the production efficiency. To improve the ease of use, the platform supports dynamic burning. This feature is implemented as follows: after a

product is created, its dynamic registration feature can be enabled to generate a product key (ProductSecret). Unified product information can be burned for all devices under it in the production process, i.e., product ID (ProductId) and product key (ProductSecret). After the devices are shipped, the device identity information can be obtained through dynamic registration and then saved, and then obtained three or four pieces of information can be used for device authentication.

- Device name (DeviceName) generation for dynamic burning: if automatic device creation is enabled during dynamic registration, device names can be generated by devices themselves, which are generally device IMEIs or MAC addresses but must be unique under the same product ID (ProductId). If automatic device creation is not enabled during dynamic registration, device names should be entered on the platform in advance, and the platform will verify whether the requested device names are validly entered during dynamic device registration. This can reduce the security risks in case of product key leakage.

> ⚠ **Note:**
> For dynamic registration, you should ensure the security of the product key (ProductSecret); otherwise, major security risks may arise.

Pre-burn device authentication programming

Writes device information

Certificate device implements the following HAL API:

HAL_API	Description
HAL_SetProductID	Set product ID, which must be stored in non-volatile storage.
HAL_SetDevName	Set device name, which must be stored in non-volatile storage.
HAL_SetDevCertName	Set device certificate file name, and the certificate file needs to be placed in the certs directory.
HAL_SetDevPrivateKeyName	Set device certificate private key file name, and the private key file needs to be placed in the certs directory.

Key device implements the following HAL API:

HAL_API	Description
HAL_SetProductID	Set product ID, which must be stored in non-volatile storage.

HAL_SetDevName	Set the device name, which must be stored in non-volatile storage medium.
HAL_SetDevSec	Set the device key, which must be stored in non-volatile storage medium, and it is recommended to use encryption and scrambling.

Device information retrieval

Certificate device implements the following HAL API:

HAL_API	Description
HAL_GetProductID	Get Product ID.
HAL_GetDevName	Get device name.
HAL_GetDevCertName	Get device certificate file name.
HAL_GetDevPrivateKeyName	Get device certificate private key file name.

Key device implements the following HAL API:

HAL_API	Description
HAL_GetProductID	Get Product ID.
HAL_GetDevName	Get device name.
HAL_GetDevSec	Get device key, if encrypted and scrambled when written, it needs to be decrypted and descrambled when read.

Application Example

- Initialize the connection parameters

```
static DeviceInfo sg_devInfo;

static int _setup_connect_init_params(MQTTInitParams* initParams)
{
    int ret;

    ret = HAL_GetDevInfo((void *)&sg_devInfo);
    if(QCLOUD_ERR_SUCCESS != ret){
```

```
        return ret;
    }

    initParams->device_name = sg_devInfo.device_name;
    initParams->product_id = sg_devInfo.product_id;
     ......
}
```

- Device information retrieval

```
int HAL_GetDevInfo(void *pdevInfo)
{
    int ret;
    DeviceInfo *devInfo = (DeviceInfo *)pdevInfo;

    memset((char *)devInfo, 0, sizeof(DeviceInfo));
    ret = HAL_GetProductID(devInfo->product_id,
MAX_SIZE_OF_PRODUCT_ID);
    ret |= HAL_GetDevName(devInfo->device_name,
MAX_SIZE_OF_DEVICE_NAME);

#ifdef  AUTH_MODE_CERT
    ret |= HAL_GetDevCertName(devInfo->devCertFileName,
MAX_SIZE_OF_DEVICE_CERT_FILE_NAME);
    ret |= HAL_GetDevPrivateKeyName(devInfo->devPrivateKeyFileName,
MAX_SIZE_OF_DEVICE_KEY_FILE_NAME);
#else
    ret |= HAL_GetDevSec(devInfo->devSerc, MAX_SIZE_OF_DEVICE_SERC);
#endif

    if(QCLOUD_ERR_SUCCESS != ret){
        Log_e("Get device info err");
        ret = QCLOUD_ERR_DEV_INFO;
    }

    return ret;
}
```

- Authentication parameter generation

```
static int _serialize_connect_packet(unsigned char *buf, size_t
buf_len, MQTTConnectParams *options, uint32_t *serialized_len) {
```

```
            ......
            ......
    int username_len = strlen(options->client_id) +
strlen(QCLOUD_IOT_DEVICE_SDK_APPID) + MAX_CONN_ID_LEN +
cur_timesec_len + 4;
    options->username = (char*)HAL_Malloc(username_len);
    get_next_conn_id(options->conn_id);
    HAL_Snprintf(options->username, username_len, "%s;%s;%s;%ld",
options->client_id, QCLOUD_IOT_DEVICE_SDK_APPID, options->conn_id,
cur_timesec);

#if defined(AUTH_WITH_NOTLS) && defined(AUTH_MODE_KEY)
     if (options->device_secret != NULL && options->username != NULL)
{
         char                sign[41]    = {0};
         utils_hmac_sha1(options->username, strlen(options->username),
sign, options->device_secret, options->device_secret_len);
         options->password = (char*) HAL_Malloc (51);
         if (options->password == NULL)
IOT_FUNC_EXIT_RC(QCLOUD_ERR_INVAL);
         HAL_Snprintf(options->password, 51, "%s;hmacsha1", sign);
     }
#endif
            ......
}
```

Dynamic burning device authentication programming

- Determine whether to initiate a dynamic application

```
int main(int argc, char **argv) {
            ......
    memset((char *)&sDevInfo, 0, sizeof(DeviceInfo));
    ret = HAL_GetProductID(sDevInfo.product_id,
MAX_SIZE_OF_PRODUCT_ID);
    ret |= HAL_GetProductKey(sDevInfo.product_key,
MAX_SIZE_OF_PRODUCT_KEY);
    ret |= HAL_GetDevName(sDevInfo.device_name,
MAX_SIZE_OF_DEVICE_NAME);  // For dynamic registration, it is
recommended to use the unique identifier of the device as the device
name, such as chip ID or IMEI

#ifdef  AUTH_MODE_CERT
```

```
    ret |= HAL_GetDevCertName(sDevInfo.devCertFileName,
MAX_SIZE_OF_DEVICE_CERT_FILE_NAME);
    ret |= HAL_GetDevPrivateKeyName(sDevInfo.devPrivateKeyFileName,
MAX_SIZE_OF_DEVICE_KEY_FILE_NAME);
    if(QCLOUD_ERR_SUCCESS != ret){
        Log_e("Get device info err");
        return QCLOUD_ERR_FAILURE;
    }
    /* Users need to modify the logic of empty device information
according to their product situation, this is just an example */
    if(!strcmp(sDevInfo.devCertFileName,
QCLOUD_IOT_NULL_CERT_FILENAME)
        ||!strcmp(sDevInfo.devPrivateKeyFileName,
QCLOUD_IOT_NULL_KEY_FILENAME)){
        Log_d("dev Cert not exist!");
        infoNullFlag = true;
    }else{
        Log_d("dev Cert exist");
    }
#else
    ret |= HAL_GetDevSec(sDevInfo.devSerc, MAX_SIZE_OF_PRODUCT_KEY);
    if(QCLOUD_ERR_SUCCESS != ret){
        Log_e("Get device info err");
        return QCLOUD_ERR_FAILURE;
    }
    /* Users need to modify the logic of empty device information
according to their product situation, this is just an example */
    if(!strcmp(sDevInfo.devSerc, QCLOUD_IOT_NULL_DEVICE_SECRET)){
        Log_d("dev psk not exist!");
        infoNullFlag = true;
    }else{
        Log_d("dev psk exist");
    }
#endif
            ......

}
```

- Initiate a dynamic application and save the device information.

```
    /*Device information is empty, initiate device registration. Note:
Once registration is successful and a connection is made, registration
```

```
cannot be initiated again. Please ensure to save the device
information.*/
    if(infoNullFlag){
        if(QCLOUD_ERR_SUCCESS == qcloud_iot_dyn_reg_dev(&sDevInfo)){

            ret = HAL_SetDevName(sDevInfo.device_name);
#ifdef  AUTH_MODE_CERT
            ret |= HAL_SetDevCertName(sDevInfo.devCertFileName);
            ret |=
HAL_SetDevPrivateKeyName(sDevInfo.devPrivateKeyFileName);
#else
            ret |= HAL_SetDevSec(sDevInfo.devSerc);
#endif
            if(QCLOUD_ERR_SUCCESS != ret){
                Log_e("devices info save fail");
            }else{
#ifdef  AUTH_MODE_CERT
                Log_d("dynamic register success, productID: %s,
devName: %s, CertFile: %s, KeyFile: %s", \
                        sDevInfo.product_id, sDevInfo.device_name,
sDevInfo.devCertFileName, sDevInfo.devPrivateKeyFileName);
#else
                Log_d("dynamic register success,productID: %s,
devName: %s, devSerc: %s", \
                        sDevInfo.product_id, sDevInfo.device_name,
sDevInfo.devSerc);
#endif
            }
        }else{
            Log_e("%s dynamic register fail", sDevInfo.device_name);
        }
    }
```

After the device information is successfully applied dynamically, the preset burning feature is completed. The subsequent authentication process is consistent with the preset burning process.

SDK for C Use Instructions
Usage Overview

Last updated: 2025-03-19 15:01:24

Tencent Cloud IoT device C SDK relies on a secure and powerful data channel to provide IoT developers with the ability to quickly connect devices to the cloud for two-way communication.

> **Note:**
> In version v3.1.0 and beyond, the SDK has undergone refactoring and optimization of the compilation environment, code, and directory structure, improving availability and portability.

C SDK Scope

C SDK adopts a modular design, separating core protocol service and hardware abstraction layer (HAL), and offers flexible configuration options and various compilation methods, suitable for different device development platforms and usage environments.

Devices with network communication capabilities and using Linux/Windows operating systems

- For devices with network communication capabilities and using standard Linux/Windows systems, such as PCs/servers/gateway devices, and more advanced embedded devices like Raspberry Pi, the SDK can be compiled and run directly on these devices.
- For embedded Linux devices that require cross-compilation, if the development environment toolchain has glibc or similar libraries and can provide system calls including socket communication, select synchronous IO, dynamic memory allocation, acquisition time/hibernate/random number/print function, and critical data protection such as mutex mechanisms (only when multi-threading is needed), simple adaptation (e.g., modifying cross-compiler settings in CMakeLists.txt or make.settings) is all that's required to compile and run the SDK.

Devices with network communication capabilities using RTOS systems

- For IoT devices with network communication capabilities using RTOS, the C SDK needs to be ported and adapted for different RTOS platforms. Currently, the C SDK has been adapted to multiple IoT-targeted RTOS platforms, including FreeRTOS, RT-Thread, and TencentOS tiny.

- When porting the SDK to RTOS devices, if the platform provides a C runtime library similar to newlib and an embedded TCP/IP protocol stack similar to lwIP, the porting and adaptation work can be easily completed.

MCU + Communication Module Devices

- For MCUs without network communication capabilities, the MCU + communication module method is generally adopted. The communication module (including Wi-Fi, 2G, 4G, NB-IoT) usually provides an AT Command Protocol based on the serial port for the MCU to perform network communication. For this scenario, the C SDK encapsulates the AT-socket network layer, and the core protocol and service layers above the network layer do not need to be ported. It also provides HAL implementations for both FreeRTOS and non-OS.
- In addition, Tencent Cloud IoT provides a dedicated AT Instruction Set. If the communication module implements this instruction set, device access and communication become simpler and require less code. For this scenario, refer to the dedicated MCU AT SDK tailored for Tencent Cloud.

SDK Directory Structure Introduction

The directory structure and top-level file overview are as follows:

Name	Description
CMakeLists.txt	CMake compilation description file.
CMakeSettings.json	CMake configuration file under Visual Studio.
cmake_build.sh	Compilation script using CMake under Linux.
make.settings	Configuration file for direct compilation using Makefile under Linux.
Makefile	Direct compilation using Makefile under Linux.
device_info.json	Device information file. When DEBUG_DEV_INFO_USED=OFF, device information will be parsed from this file.
docs	Document directory, SDK usage instructions on different platforms.
external_libs	Third-party software package components, such as mbedtls.
samples	Application examples.
include	External header files provided for user use.

platform	Platform-related source files, currently provided for different OS (Linux/Windows/FreeRTOS/nonOS), TLS (mbedtls), and AT module implementations.
sdk_src	SDK core communication protocols and service code.
tools	SDK supporting compilation and code generation scripting tools.

SDK Compilation Method Description

The SDK for C supports three compilation methods:

- cmake.
- Makefile.
- Code Extraction Method.

For detailed explanations of compilation methods and configuration options, please refer to Compilation Configuration Instructions and Compilation Environment Description.

SDK Samples

The samples directory of the SDK for C contains examples using various features. For detailed descriptions on running examples, please refer to all documents under the SDK documentation directory.

For a quick experience of device-side MQTT access and message sending and receiving on the IoT Hub platform, please refer to Getting Started with MQTT.

Must-Knows

OTA Upgrade API Changes

Starting from SDK version 3.0.3, OTA upgrade supports breakpoint resume. If the firmware download is interrupted due to network issues or other reasons, the partially downloaded firmware section can be saved, and the download can resume from the last downloaded section.

With this new feature, the usage method of the OTA-related API has changed. Users upgrading from version 3.0.2 and earlier need to modify their user logic code; otherwise, the firmware download will fail. Please refer to `samples/ota/ota_mqtt_sample.c` for the modification.

Code Naming Changes

To improve code readability and ensure naming conventions, SDK version 3.1.0 has made changes to some variables, functions, and macro naming. Users upgrading from version 3.0.3 or earlier can execute the `tools/update_from_old_SDK.sh` script in a Linux environment to

perform name replacement in their code. After the replacement is complete, you can directly use the new version of the SDK.

Legacy Naming	New Naming
QCLOUD_ERR_SUCCESS	QCLOUD_RET_SUCCESS
QCLOUD_ERR_MQTT_RECONNECTED	QCLOUD_RET_MQTT_RECONNECTED
QCLOUD_ERR_MQTT_MANUALLY_DISCONNECTED	QCLOUD_RET_MQTT_MANUALLY_DISCONNECTED
QCLOUD_ERR_MQTT_CONNACK_CONNECTION_ACCEPTED	QCLOUD_RET_MQTT_CONNACK_CONNECTION_ACCEPTED
QCLOUD_ERR_MQTT_ALREADY_CONNECTED	QCLOUD_RET_MQTT_ALREADY_CONNECTED
MAX_SIZE_OF_DEVICE_SERC	MAX_SIZE_OF_DEVICE_SECRET
devCertFileName	dev_cert_file_name
devPrivateKeyFileName	dev_key_file_name
devSerc	device_secret
MAX_SIZE_OF_PRODUCT_KEY	MAX_SIZE_OF_PRODUCT_SECRET
product_key	product_secret.
DEBUG	eLOG_DEBUG
INFO	eLOG_INFO
WARN	eLOG_WARN
ERROR	eLOG_ERROR
DISABLE	eLOG_DISABLE
Log_writter	IOT_Log_Gen
qcloud_iot_dyn_reg_dev	IOT_DynReg_Device
IOT_SYSTEM_GET_TIME	IOT_Get_SysTime

Compilation Configuration Description

Last updated: 2025-03-19 15:01:37

This document describes the compilation methods and compilation configuration options of the SDK for C, as well as the compilation environment setup and compilation samples in the Linux and Windows development environments.

SDK for C Compilation Method Description

The SDK for C supports the following compilation methods.

CMake Method

- We recommend you use CMake, a cross-platform compilation tool, for compilation in the Linux and Windows development environments.
- The CMake method uses CMakeLists.txt as the compilation configuration options input file.

Makefile

- For environments that don't support CMake, Makefile can be used for compilation.
- The Makefile method is consistent with SDK version 3.0.3 and earlier versions. It uses make.settings as the compilation configuration options input file. After making modifications, simply execute 'make'.

Code Extraction Method

- This method allows you to select features based on your needs and extract the relevant code into a separate folder. The code hierarchy in the folder is concise, making it easy for you to copy and integrate it into your own development environment.
- This method relies on the cmake tool. Configure the relevant feature module switches in the CMakeLists.txt file and set EXTRACT_SRC to ON. Then run the following command in a Linux environment:

```
mkdir build
cd build
cmake ..
```

- You can find the related code files in the output/qcloud_iot_c_sdk directory. The directory structure is as follows:

```
qcloud_iot_c_sdk
├── include
│   ├── config.h
│   ├── exports
├── platform
└── sdk_src
 └── internal_inc
```

- The include directory contains the API and variable parameters provided by the SDK for the user. The file config.h contains the compilation macros generated based on the compilation options.
- The platform directory contains platform-related code, which can be modified and adapted according to the specific conditions of the device.
- The sdk_src directory contains the core logic and protocol-related code of the SDK, which generally does not require modification. The internal_inc directory contains header files used internally by the SDK.

Note:
Users can copy qcloud_iot_c_sdk to their target platform's compilation development environment and modify the compilation options according to the specific situation.

SDK for C Compilation Option Description

Compilation configuration options

Most of the following configuration options apply to both cmake and make. The ON value in setting.cmake corresponds to y in make.setting, and OFF corresponds to n.

Name	CMake Value	Description
BUILD_TYPE	release/debug	• release: Do not enable IOT_DEBUG information, compile and output to the release directory. • debug: Enable IOT_DEBUG information, compile and output to the debug directory.
EXTRACT_SRC	ON/OFF	Whether to enable code extraction, which takes effect only for CMake.

COMPILE_TOOLS	gcc	Supports gcc and msvc, cross-compilers can also be used. For example, arm-none-linux-gnueabi-gcc.
PLATFORM	Linux	Includes Linux/Windows/FreeRTOS/nonOS.
FEATURE_MQTT_COMM_ENABLED	ON/OFF	Whether to enable MQTT channel.
FEATURE_MQTT_DEVICE_SHADOW	ON/OFF	Whether to enable device shadow.
FEATURE_COAP_COMM_ENABLED	ON/OFF	Whether to enable CoAP channel.
FEATURE_GATEWAY_ENABLED	ON/OFF	Whether to enable gateway feature.
FEATURE_OTA_COMM_ENABLED	ON/OFF	Whether to enable OTA firmware update.
FEATURE_OTA_SIGNAL_CHANNEL	MQTT/COAP	OTA signaling channel type.
FEATURE_AUTH_MODE	KEY/CERT	Access Authentication Method.
FEATURE_AUTH_WITH_NOTLS	ON/OFF	• OFF: TLS enabled. • ON: TLS disabled.
FEATURE_DEV_DYN_REG_ENABLED	ON/OFF	Whether to enable dynamic device registration.
FEATURE_LOG_UPLOAD_ENABLED	ON/OFF	Log Reporting Switch.
FEATURE_EVENT_POST_ENABLED	ON/OFF	Event Reporting Switch.
FEATURE_DEBUG_DEV_INFO_USED	ON/OFF	Whether to enable device information source acquisition.
FEATURE_SYSTEM_COMM_ENABLED	ON/OFF	Backend Time Acquisition Switch.

FEATURE_AT_TCP_ENABLED	ON/OFF	Whether to enable TCP feature in AT module.
FEATURE_AT_UART_RECV_IRQ	ON/OFF	Whether to enable receipt interruption feature in AT module.
FEATURE_AT_OS_USED	ON/OFF	Whether to enable multithreaded feature in AT module.
FEATURE_AT_DEBUG	ON/OFF	Whether to enable debugging feature in AT module.
FEATURE_MULTITHREAD_TEST_ENABLED	ON/OFF	Whether to compile the Linux multithreaded test routine.

There is a dependency relationship between the configuration options. A configuration option is valid only when the value of its dependent option is valid as shown below:

Name	Dependent option	Effective value
FEATURE_MQTT_DEVICE_SHADOW	FEATURE_MQTT_COMM_ENABLED	ON
FEATURE_GATEWAY_ENABLED	FEATURE_MQTT_COMM_ENABLED	ON
FEATURE_OTA_SIGNAL_CHANNEL(MQTT)	FEATURE_OTA_COMM_ENABLED FEATURE_MQTT_COMM_ENABLED	ON ON
FEATURE_OTA_SIGNAL_CHANNEL(COAP)	FEATURE_OTA_COMM_ENABLED FEATURE_COAP_COMM_ENABLED	ON ON
FEATURE_AUTH_WITH_NOTLS	FEATURE_AUTH_MODE	KEY
FEATURE_AT_UART_RECV_IRQ	FEATURE_AT_TCP_ENABLED	ON
FEATURE_AT_OS_USED	FEATURE_AT_TCP_ENABLED	ON
FEATURE_AT_DEBUG	FEATURE_AT_TCP_ENABLED	ON

Device information options

After creating a device in the Tencent Cloud IoT Console, you need to configure the device information (ProductID/DeviceName/DeviceSecret/Cert/Key file) in the SDK to ensure proper operation. In the development phase, the SDK provides two methods of storing device information:

- Stored in code (compilation option DEBUG_DEV_INFO_USED = ON), then modify the device information in `platform/os/xxx/HAL_Device_xxx.c`. This method can be used on platforms without a file system.
- Stored in configuration files (compilation option DEBUG_DEV_INFO_USED = OFF), then modify the device information in the `device_info.json` file. This method does not require recompiling the SDK to change the device information and is recommended for development on Linux/Windows platforms.

Compilation Environment (Linux and Windows)

Last updated: 2025-03-19 15:01:52

Linux (Ubuntu)

> **Note:**
> The Ubuntu version used for demonstration in this document is v16.04.

1. **Installing necessary software**
   The SDK requires CMake v3.5 or above. The CMake version installed by default is low. If compilation fails, please click download and refer to the Installation Instructions to download and install the specific version of CMake.

```
$ sudo apt-get install -y build-essential make git gcc cmake
```

2. **Modifying Configuration**
   Modify the CMakeLists.txt file in the root directory of the SDK and ensure the following options exist (using key authentication device as an example):

```
set(BUILD_TYPE                   "release")
set(COMPILE_TOOLS                "gcc")
set(PLATFORM                    "linux")
set(FEATURE_MQTT_COMM_ENABLED ON)
set(FEATURE_AUTH_MODE "KEY")
set(FEATURE_AUTH_WITH_NOTLS OFF)
set(FEATURE_DEBUG_DEV_INFO_USED  OFF)
```

3. **Run the script for compilation**

   3.1 Below is a complete compilation library and demo:

```
./cmake_build.sh
```

   3.2 The output library files, header files, and samples are in the output/release folder. After a complete compilation, if you only need to compile the demo, then run the following code:

```
./cmake_build.sh samples
```

4. **Enter the device information**
   Enter the device information created on the Tencent Cloud IoT platform (using key authentication device as an example) into the SDK root directory's device_info.json. Sample code is as follows:

```
"auth_mode":"KEY",
"productId":"S3EUVBQAZW",
"deviceName":"test_device",
"key_deviceinfo":{
    "deviceSecret":"vX6PQqazsGsMyf5SMfs6OA6y"
}
```

5. **Run Example**
   The example output is located in the `output/release/bin` folder. For example, to run the data_template_sample example, enter `./output/release/bin/data_template_sample`.

Windows Environment

Getting and installing Visual Studio 2019

1. Visit the Visual Studio download site, download and install Visual Studio 2019. The version downloaded and installed in this document is v16.2 Community.

2. Select "**Desktop development with C++**" and make sure "C++ CMAKE tools for Windows" is checked.

Compilation and running

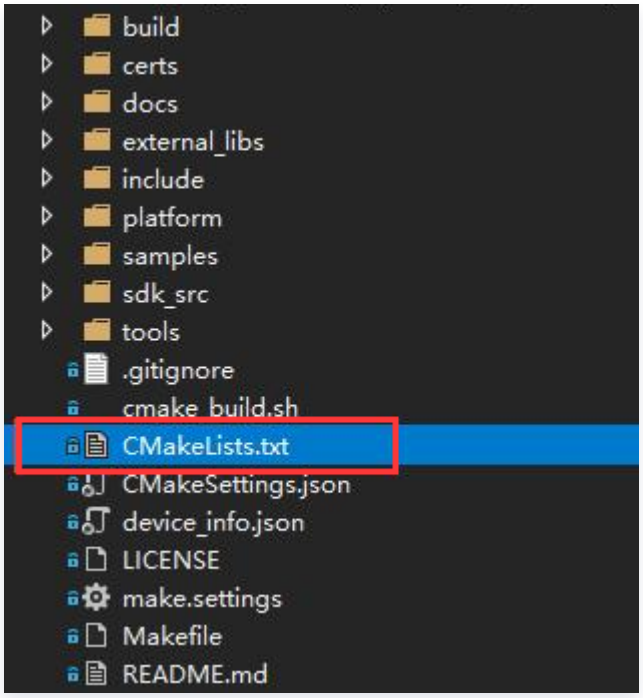
1. Run Visual Studio, choose **Open a local folder**, and select the downloaded C SDK directory.

2. Fill in the device information of the device created in the Tencent Cloud IoT Communication Console (using a key-authenticated device as an example) into device_info.json. The sample code is as follows:

```
"auth_mode":"KEY",
"productId":"S3EUVBQAZW",
"deviceName":"test_device",
"key_deviceinfo":{
    "deviceSecret":"vX6PQqazsGsMyf5SMfs6OA6y"
```

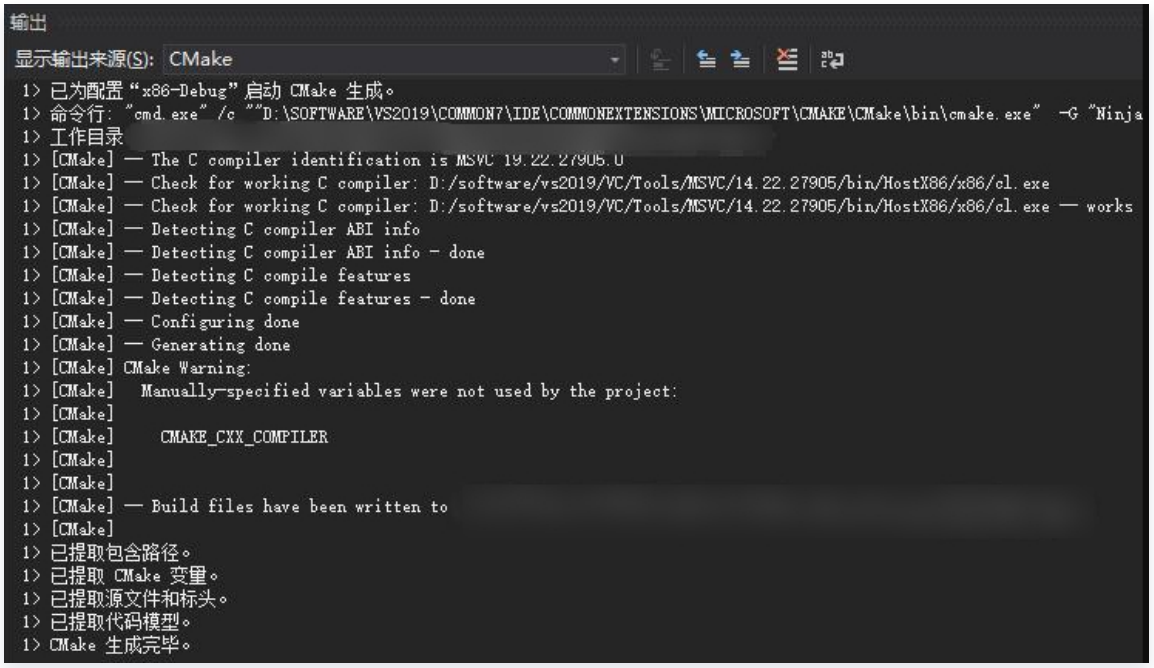
```
}
```

3. Double-click to open the CMakeLists.txt in the root directory and ensure the platform set in the compilation toolchain is **Windows** and the compiler is **MSVC**.



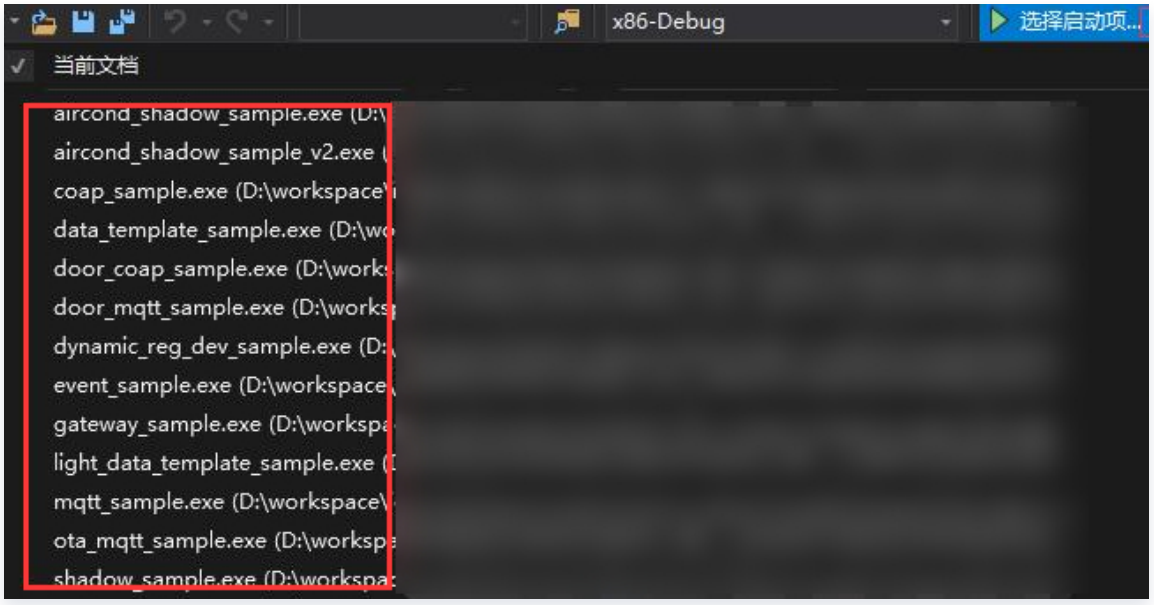

```
# Compilation toolchain
   #set(COMPILE_TOOLS "gcc")
   #set(PLATFORM       "linux")

   set(COMPILE_TOOLS "MSVC")
   set(PLATFORM       "windows")
```

4. Visual Studio will automatically build the CMake cache. Just wait for the build to complete.



5. After the cache build is complete, select **Build** > **All Generation**.

6. Choose the appropriate example to run, which should correspond to the user information.



Getting Started with MQTT

Last updated：2025-03-19 15:02:05

This document describes how to create devices and permissions in the Tencent Cloud IoT Hub console and quickly try out device connection to IoT Hub over the MQTT protocol for message sending and receiving based on the mqtt_sample of the C-SDK.

Operations in Console

Creating product and device

1. Log in to the IoT Hub console, click Products on the left sidebar.
2. On the product list page, click Create New Product.
3. In the pop-up Add New Product page, select the node type and product type, enter the product name, choose the authentication method and data format, and enter the product description, then click Confirm. (For ordinary direct connect devices, you can choose as shown in the figure.)

4. After the product is created, click Device List on the generated product page.
5. On the device list page, click Add New Device.
   - If the authentication method is certificate authentication, after entering the device name successfully, remember to click the Download button in the pop-up window to download the device key file and device certificate from the package for device connection authentication with IoT Hub.
   - If the authentication method is key authentication, after the device name is entered, the key of the added device will be displayed in the pop-up window.

Creating Topic

1. On the generated product page, click Permission List.
2. Enter the permission list page and click **Adding topic permission**.
3. In the pop-up Topic Permissions page, enter data and set the operation permissions to **Publishing and Subscribing**, then click **OK**.

4. Then, the `productID/\${deviceName}/data` topic will be created, and you can view all permissions of the product in the permission list on the product page.

Compiling and Running Demo

The following describes the compilation and running of the mqtt_sample example in the Linux environment (using key authentication equipment as an example).

1. Compile the SDK

1.1 Modify CMakeLists.txt to ensure the following options exist:

```
set(BUILD_TYPE                  "release")
set(COMPILE_TOOLS               "gcc")
set(PLATFORM                   "linux")
```

```
set(FEATURE_MQTT_COMM_ENABLED ON)
set(FEATURE_AUTH_MODE "KEY")
set(FEATURE_AUTH_WITH_NOTLS OFF)
set(FEATURE_DEBUG_DEV_INFO_USED  OFF)
```

1.2 Run the following script for compilation.

```
./cmake_build.sh
```

1.3 The demo output is in the `output/release/bin` folder.

2. Enter the device information

Fill in the device information created on Tencent Cloud IoT Hub into device_info.json.

```
"auth_mode":"KEY",
"productId":"S3EUVBRJLB",
"deviceName":"test_device",
"key_deviceinfo":{
"deviceSecret":"vX6PQqazsGsMyf5SMfs6OA6y"
    }
```

3. Run the mqtt_sample demo.

```
./output/release/bin/mqtt_sample
INF|2019-09-12 21:28:20|device.c|iot_device_info_set(67): SDK_Ver:
3.1.0, Product_ID: S3EUVBRJLB, Device_Name: test_device
DBG|2019-09-12 21:28:20|HAL_TLS_mbedtls.c|HAL_TLS_Connect(204):
Setting up the SSL/TLS structure...
DBG|2019-09-12 21:28:20|HAL_TLS_mbedtls.c|HAL_TLS_Connect(246):
Performing the SSL/TLS handshake...
DBG|2019-09-12 21:28:20|HAL_TLS_mbedtls.c|HAL_TLS_Connect(247):
Connecting to /S3EUVBRJLB.iotcloud.tencentdevices.com/8883...
INF|2019-09-12 21:28:20|HAL_TLS_mbedtls.c|HAL_TLS_Connect(269):
connected with /S3EUVBRJLB.iotcloud.tencentdevices.com/8883...
INF|2019-09-12 21:28:20|mqtt_client.c|IOT_MQTT_Construct(125): mqtt
connect with id: p8t0W success
INF|2019-09-12 21:28:20|mqtt_sample.c|main(303): Cloud Device
Construct Success
DBG|2019-09-12
21:28:20|mqtt_client_subscribe.c|qcloud_iot_mqtt_subscribe(138):
topicName=$sys/operation/result/S3EUVBRJLB/test_device|packet_id=1932
```

```
INF|2019-09-12 21:28:20|mqtt_sample.c|_mqtt_event_handler(71):
subscribe success, packet-id=1932
DBG|2019-09-12
21:28:20|system_mqtt.c|_system_mqtt_sub_event_handler(80): mqtt sys
topic subscribe success
DBG|2019-09-12
21:28:20|mqtt_client_publish.c|qcloud_iot_mqtt_publish(337): publish
packetID=0|topicName=$sys/operation/S3EUVBRJLB/test_device|payload=
{"type": "get", "resource": ["time"]}
DBG|2019-09-12
21:28:20|system_mqtt.c|_system_mqtt_message_callback(63): Recv Msg
Topic:$sys/operation/result/S3EUVBRJLB/test_device, payload:
{"type":"get","time":1568294900}
INF|2019-09-12 21:28:21|mqtt_sample.c|main(316): system time is
1568294900
DBG|2019-09-12
21:28:21|mqtt_client_subscribe.c|qcloud_iot_mqtt_subscribe(138):
topicName=S3EUVBRJLB/test_device/data|packet_id=1933
INF|2019-09-12 21:28:21|mqtt_sample.c|_mqtt_event_handler(71):
subscribe success, packet-id=1933
DBG|2019-09-12
21:28:21|mqtt_client_publish.c|qcloud_iot_mqtt_publish(329): publish
topic seq=1934|topicName=S3EUVBRJLB/test_device/data|payload=
{"action": "publish_test", "count": "0"}
INF|2019-09-12 21:28:21|mqtt_sample.c|_mqtt_event_handler(98): publish
success, packet-id=1934
INF|2019-09-12 21:28:21|mqtt_sample.c|on_message_callback(195):
Receive Message With topicName:S3EUVBRJLB/test_device/data, payload:
{"action": "publish_test", "count": "0"}
INF|2019-09-12
21:28:22|mqtt_client_connect.c|qcloud_iot_mqtt_disconnect(437): mqtt
disconnect!
INF|2019-09-12
21:28:22|system_mqtt.c|_system_mqtt_sub_event_handler(98): mqtt client
has been destroyed
INF|2019-09-12 21:28:22|mqtt_client.c|IOT_MQTT_Destroy(186): mqtt
release!
```

4. Observe message sending

The following log information shows that the sample program reports data to `/{productID}/{deviceName}/data` through an MQTT Publish message, and the server has received and successfully processed the message.

```
INF|2019-09-12 21:28:21|mqtt_sample.c|_mqtt_event_handler(98): publish
success, packet-id=1934
```

5. Observe message receiving

The following log information shows that as the message reached the subscribed topic, it was pushed to the demo as-is by the server and entered the corresponding callback function.

```
INF|2019-09-12 21:28:21|mqtt_sample.c|on_message_callback(195):
Receive Message With topicName:S3EUVBRJLB/test_device/data, payload:
{"action": "publish_test", "count": "0"}
```

6. Observe logs in the console

Log in to the IoT Hub console, click the product name, and click **Cloud Logs** in the top menu to view the recently reported message.

API and Variable Parameter Description

Last updated: 2025-03-19 15:02:19

The header files for API function declarations, constants, and variable parameter definitions for the device-side C SDK that can be called by users are located in the include directory. This document mainly describes the variable parameters and API functions in that directory.

Variable Parameter Configuration

You can configure corresponding parameters in the SDK for C based on the needs in specific scenarios to ensure the smooth operations of your businesses. Variable connection parameters include:

- Timeout period of blocking MQTT calls (including connection, subscribing, and publishing) in milliseconds. 5000 ms is recommended.
- Size of the buffer for message sending and receiving over the MQTT protocol, which is 2,048 bytes by default and can be up to 16 KB.
- Size of the buffer for message sending and receiving over the CoAP protocol, which is 512 bytes by default and can be up to 1 KB.
- MQTT heartbeat message sending interval in milliseconds, which can be up to 690s.
- Maximum waiting time for reconnection in milliseconds. When a device is reconnected after disconnection, the waiting time will double if reconnection fails, and reconnection will stop when the maximum waiting time is exceeded.

Modify the macro definitions in the `include/qcloud_iot_export_variables.h` file as follows to change the configuration of the corresponding access parameters.
After modification, the SDK needs to be recompiled. Example code is as follows:

```
/* default MQTT/CoAP timeout value when connect/pub/sub (unit: ms) */
#define QCLOUD_IOT_MQTT_COMMAND_TIMEOUT                                 (5 *
1000)

/* default MQTT keep alive interval (unit: ms) */
#define QCLOUD_IOT_MQTT_KEEP_ALIVE_INTERNAL                             (240
* 1000)

/* default MQTT Tx buffer size, MAX: 16*1024 */
#define QCLOUD_IOT_MQTT_TX_BUF_LEN
(2048)
```

```
/* default MQTT Rx buffer size, MAX: 16*1024 */
#define QCLOUD_IOT_MQTT_RX_BUF_LEN
(2048)

/* default COAP Tx buffer size, MAX: 1*1024 */
#define COAP_SENDMSG_MAX_BUFLEN
(512)

/* default COAP Rx buffer size, MAX: 1*1024 */
#define COAP_RECVMSG_MAX_BUFLEN
(512)

/* MAX MQTT reconnect interval (unit: ms) */
#define MAX_RECONNECT_WAIT_INTERVAL                                (60
* 1000)
```

API Function Description

Below are the main features and corresponding API interface descriptions provided by the C SDK v3.1.0 version. They are used for customers to write business logic. For more detailed descriptions such as interface parameters and return values, please refer to comments in the SDK code `include/exports/qcloud_iot_export_*.h` and other header files.

MQTT APIs

Serial number	Function Name	Description
1	IOT_MQTT_Construct	Constructs an MQTTClient and connects to the MQTT cloud service.
2	IOT_MQTT_Destroy	Closes MQTT connection and destroys MQTTClient.
3	IOT_MQTT_Yield	Performs tasks such as reading MQTT messages, processing messages, timing out requests, and managing heartbeat packets and reconnection status in the current thread context.
4	IOT_MQTT_Publish	Publishes MQTT message.
5	IOT_MQTT_Subscribe	Subscribes to MQTT topic.

6	IOT_MQTT_Unsubscribe	Unsubscribes from subscribed MQTT topic.
7	IOT_MQTT_IsConnected	Queries whether MQTT is currently connected.
8	IOT_MQTT_GetErrCode	Gets the error code for IOT_MQTT_Construct failure.

Notes on use in multithreaded environment

To use MQTT APIs in a multithreaded environment, you need to pay attention to the following:

- Multi-threaded calls to IOT_MQTT_Yield, IOT_MQTT_Construct, and IOT_MQTT_Destroy are not allowed.
- Multi-threaded calls to IOT_MQTT_Publish, IOT_MQTT_Subscribe, and IOT_MQTT_Unsubscribe are allowed.
- IOT_MQTT_Yield, as a function to read and process MQTT messages from the socket, should ensure certain execution time, avoiding long suspension or preemption.

Device Shadow APIs

For an introduction to the Device Shadow feature, please refer to Device Shadow Details.

Serial number	Function Name	Description
1	IOT_Shadow_Construct	Constructs the device shadow client ShadowClient and connects to MQTT cloud service.
2	IOT_Shadow_Publish	The shadow client publishes an MQTT message.
3	IOT_Shadow_Subscribe	The shadow client subscribes to an MQTT topic.
4	IOT_Shadow_Unsubscribe	The shadow client unsubscribes from a subscribed MQTT topic.
5	IOT_Shadow_IsConnected	Queries whether MQTT of the shadow client is currently connected.
6	IOT_Shadow_Destroy	Closes Shadow MQTT connection and destroys ShadowClient.

7	IOT_Shadow_Yield	Performs tasks such as reading MQTT messages, processing messages, timing out requests, and managing heartbeat packets and reconnection status in the current thread context.
8	IOT_Shadow_Update	Updates device shadow document asynchronously.
9	IOT_Shadow_Update_Sync	Updates device shadow document synchronously.
10	IOT_Shadow_Get	Gets device shadow document asynchronously.
11	IOT_Shadow_Get_Sync	Gets device shadow document synchronously.
12	IOT_Shadow_Register_Property	Registers the attribute of the current device.
13	IOT_Shadow_UnRegister_Property	Deletes registered device attribute.
14	IOT_Shadow_JSON_ConstructReport	Add the reported field to the JSON document without updating.
15	IOT_Shadow_JSON_Construct_OverwriteReport	Add the reported field to the JSON document and update.
16	IOT_Shadow_JSON_ConstructReportAndDesireAllNull	Add the reported field to the JSON document and clear the desired field.
17	IOT_Shadow_JSON_ConstructDesireAllNull	Add the "desired": null field to the JSON document.

CoAP APIs

Serial number	Function Name	Description
1	IOT_COAP_Construct	Construct CoAPClient and complete the CoAP connection.
2	IOT_COAP_Destroy	Close the CoAP connection and destroy CoAPClient.

3	IOT_COAP_Yield	Perform tasks such as reading CoAP messages and processing messages in the current thread context.
4	IOT_COAP_SendMessage	Publish CoAP messages.
5	IOT_COAP_GetMessageId	Get COAP Response message msgId.
6	IOT_COAP_GetMessagePayload	Get COAP Response message content.
7	IOT_COAP_GetMessageCode	Get COAP Response message error code.

OTA interface

For more information on the OTA firmware download feature, please see Device Firmware Upgrade.

Serial number	Function Name	Description
1	IOT_OTA_Init	Initialize the OTA module. The client needs to initialize MQTT/CoAP before calling this API.
2	IOT_OTA_Destroy	Releases resources related to the OTA module.
3	IOT_OTA_ReportVersion	Reports local firmware version information to the OTA server.
4	IOT_OTA_IsFetching	Checks whether the firmware is being downloaded.
5	IOT_OTA_IsFetchFinish	Checks whether the firmware has been downloaded.
6	IOT_OTA_FetchYield	Gets firmware from a remote server with a specific timeout value.
7	IOT_OTA_Ioctl	Gets the specified OTA information.
8	IOT_OTA_GetLastError	Gets the last error code.
9	IOT_OTA_StartDownload	Establishes HTTP connection with firmware server according to obtained firmware update

		address and local firmware information offset (whether to support checkpoint restart).
10	IOT_OTA_UpdateClientMd5	Calculates the MD5 of local firmware before checkpoint restart.
11	IOT_OTA_ReportUpgradeBegin	Reports the status of impending update to the server before firmware update.
12	IOT_OTA_ReportUpgradeSuccess	Reports the status of update success to the server after successful firmware update.
13	IOT_OTA_ReportUpgradeFail	Reports the status of update failure to the server after failed firmware update.

Log APIs

For detailed information on the device log reporting Cloud feature, refer to the SDK docs directory under the Internet of Things Hub platform documentation device log reporting feature section.

Serial number	Function Name	Description
1	IOT_Log_Set_Level	Sets the printout level of SDK logs.
2	IOT_Log_Get_Level	Returns the printout level of SDK logs.
3	IOT_Log_Set_MessageHandler	Sets log callback function to redirect SDK logs to another output method.
4	IOT_Log_Init_Uploader	Enables SDK log reporting to the cloud and initializes resources.
5	IOT_Log_Fini_Uploader	Disables SDK log reporting to the cloud and releases resources.
6	IOT_Log_Upload	Reports SDK execution logs to the cloud.
7	IOT_Log_Set_Upload_Level	Sets the reporting level of SDK logs.
8	IOT_Log_Get_Upload_Level	Returns the reporting level of SDK logs.
9	Log_d/i/w/e	Prints SDK logs by level.

System time APIs

Serial number	Function Name	Description
1	IOT_Get_SysTime	Gets IoT Hub's backend time. Currently, the time sync feature is supported only for the MQTT channel.

Gateway feature APIs

For an introduction to the gateway feature, refer to the IoT Hub documentation under the Internet of Things Hub platform section in the SDK docs directory.

Serial number	Function Name	Description
1	IOT_Gateway_Construct	Constructs gateway client and completes MQTT connection.
2	IOT_Gateway_Destroy	Closes MQTT connection and terminates gateway client.
3	IOT_Gateway_Subdev_Online	Sub-device connection.
4	IOT_Gateway_Subdev_Offline	Sub-device decommission.
5	IOT_Gateway_Yield	Performs tasks such as reading MQTT messages, processing messages, timing out requests, and managing heartbeat packets and reconnection status in the current thread context.
6	IOT_Gateway_Publish	Publishes MQTT message.
7	IOT_Gateway_Subscribe	Subscribes to MQTT topic.
8	IOT_Gateway_Unsubscribe	Unsubscribes from subscribed MQTT topic.

Device Information Storage

Last updated: 2025-03-19 15:02:32

Overview

Tencent Cloud IoT Platform assigns a unique identifier ProductID to each created product. Users can customize DeviceName to identify devices and use product identifier + device identifier + device certificate/key to verify the validity of the device. The device side needs to store this device identity information. The C-SDK provides interfaces and reference implementations for device information read and write, which can be adapted according to actual conditions.

Device Identity Information

- Certificate-authenticated devices must carry the following four pieces of information before it can pass the authentication by the platform: product ID (ProductId), device name (DeviceName), device certificate (DeviceCert), and device private key (DevicePrivateKey), among which, the certificate and private key files are generated by the platform and correspond to each other.
- Key-authenticated devices must carry the following three pieces of information before it can pass the authentication by the platform: product ID (ProductId), device name (DeviceName), and device key (DeviceSecret), among which, the device key is generated by the platform.

Device Identity Information Burning

Device information burning is divided into preset burning and dynamic burning, which differ in terms of convenience and security.

Preset burning

After a product is created, you can create devices one by one in the IoT Hub console or through TencentCloud API, get their corresponding device information, and burn the above three or four pieces of information into a non-volatile medium in a specific step of device production, so that the device SDK can read the stored device information during running for device authentication.

Dynamic burning

- Preset burning: this involves performing personalized production actions in the mass production process and thus affects the production efficiency. To improve the ease of use, the platform supports dynamic burning. This feature is implemented as follows: after a

product is created, its dynamic registration feature can be enabled to generate a product key (ProductSecret). Unified product information can be burned for all devices under it in the production process, i.e., product ID (ProductId) and product key (ProductSecret). After the devices are shipped, the device identity information can be obtained through dynamic registration and then saved, and then obtained three or four pieces of information can be used for device authentication.

- Device name (DeviceName) generation for dynamic burning: if automatic device creation is enabled during dynamic registration, device names can be generated by devices themselves, which are generally device IMEIs or MAC addresses but must be unique under the same product ID (ProductId). If automatic device creation is not enabled during dynamic registration, device names should be entered on the platform in advance, and the platform will verify whether the requested device names are validly entered during dynamic device registration. This can reduce the security risks in case of product key leakage.

> ⚠ **Note:**
> For dynamic registration, you should ensure the security of the product key (ProductSecret); otherwise, major security risks may arise.

Device Information Read/Write HAL APIs

The SDK provides HAL interfaces for device information read and write, which must be implemented. You can refer to the implementation of device information read and write in HAL_Device_Linux.c on the Linux platform.
Device information HAL APIs:

HAL_API	Description
HAL_SetDevInfo	Writes device information.
HAL_GetDevInfo	Reads device information.

Device Information Configuration in Development Phase

After a device is created, you need to configure its information (`ProductID/DeviceName/DeviceSecret/Cert/Key` file) in the SDK first before the demo can run properly. In the development phase, the SDK provides two methods of storing the device information.

1. Stored in code (compilation option DEBUG_DEV_INFO_USED = ON), then modify the device information in `platform/os/xxx/HAL_Device_xxx.c` . This method can be used on platforms without a file system.

```
/* product Id  */
static char sg_product_id[MAX_SIZE_OF_PRODUCT_ID + 1]     =
"PRODUCT_ID";

/* device name */
static char sg_device_name[MAX_SIZE_OF_DEVICE_NAME + 1]   =
"YOUR_DEV_NAME";

#ifdef DEV_DYN_REG_ENABLED
/* product secret for device dynamic Registration  */
static char sg_product_secret[MAX_SIZE_OF_PRODUCT_SECRET + 1]   =
"YOUR_PRODUCT_SECRET";
#endif

#ifdef AUTH_MODE_CERT
/* public cert file name of certificate device */
static char sg_device_cert_file_name[MAX_SIZE_OF_DEVICE_CERT_FILE_NAME
+ 1]       = "YOUR_DEVICE_NAME_cert.crt";
/* private key file name of certificate device */
static char
sg_device_privatekey_file_name[MAX_SIZE_OF_DEVICE_SECRET_FILE_NAME +
1] = "YOUR_DEVICE_NAME_private.key";
#else
/* device secret of PSK device */
static char sg_device_secret[MAX_SIZE_OF_DEVICE_SECRET + 1] =
"YOUR_IOT_PSK";
#endif
```

2. Stored in configuration files (compilation option DEBUG_DEV_INFO_USED = OFF), then modify the device information in the `device_info.json` file. This method does not require recompiling the SDK to change the device information and is recommended for development on Linux/Windows platforms.

```
{
    "auth_mode":"KEY/CERT",

    "productId":"PRODUCT_ID",
    "productSecret":"YOUR_PRODUCT_SECRET",
    "deviceName":"YOUR_DEV_NAME",

    "key_deviceinfo":{
        "deviceSecret":"YOUR_IOT_PSK"
```

```
    }

    "cert_deviceinfo":{
        "devCertFile":"YOUR_DEVICE_CERT_FILE_NAME",
        "devPrivateKeyFile":"YOUR_DEVICE_PRIVATE_KEY_FILE_NAME"
    }

    "subDev":{
        "sub_productId":"YOUR_SUBDEV_PRODUCT_ID",
        "sub_devName":"YOUR_SUBDEV_DEVICE_NAME"
    }
}
```

Application Example

- Initialize the connection parameters

```
static DeviceInfo sg_devInfo;

static int _setup_connect_init_params(MQTTInitParams* initParams)
{
    int ret;

    ret = HAL_GetDevInfo((void *)&sg_devInfo);
    if(QCLOUD_ERR_SUCCESS != ret){
        return ret;
    }

    initParams->device_name = sg_devInfo.device_name;
    initParams->product_id = sg_devInfo.product_id;
     ......
}
```

- Generate the parameters for authenticating a key-authenticated device

```
static int _serialize_connect_packet(unsigned char *buf, size_t
buf_len, MQTTConnectParams *options, uint32_t *serialized_len) {
            ......
            ......
    int username_len = strlen(options->client_id) +
strlen(QCLOUD_IOT_DEVICE_SDK_APPID) + MAX_CONN_ID_LEN +
cur_timesec_len + 4;
```

```
    options->username = (char*)HAL_Malloc(username_len);
    get_next_conn_id(options->conn_id);
    HAL_Snprintf(options->username, username_len, "%s;%s;%s;%ld",
options->client_id, QCLOUD_IOT_DEVICE_SDK_APPID, options->conn_id,
cur_timesec);

#if defined(AUTH_WITH_NOTLS) && defined(AUTH_MODE_KEY)
     if (options->device_secret != NULL && options->username != NULL)
{
         char                sign[41]    = {0};
         utils_hmac_sha1(options->username, strlen(options->username),
sign, options->device_secret, options->device_secret_len);
         options->password = (char*) HAL_Malloc (51);
         if (options->password == NULL)
IOT_FUNC_EXIT_RC(QCLOUD_ERR_INVAL);
         HAL_Snprintf(options->password, 51, "%s;hmacsha1", sign);
     }
#endif
            ......
}
```

Connection Based on SDK for Android
SDK for Android Release Notes

Last updated: 2025-03-19 15:02:52

Code Hosting

The code of the device SDK for Android has been hosted on GitHub since v1.0.0.

Version Information

For the version iteration information of the device SDK for Android since v1.0.0, please visit GitHub.

SDK for Android Project Configuration

Last updated: 2025-03-19 15:03:07

IoT Hub device SDK for Android relies on a secure and powerful data channel to enable IoT developers to quickly connect devices to the cloud for two-way communication. Developers only need to complete the corresponding project configuration to connect devices.

Prerequisites

The product and device have been created according to the Device Access Preparation.

Reference Method

Integrated SDKs

- Dependencies: Maven Remote Construction, as shown below:

```
dependencies {
        implementation 'com.tencent.iot.hub:hub-device-android:x.x.x'
}
```

> **Note:**
> You can set the above x.x.x to the latest version according to Version Explanation.

- Depend on local SDK source code to build:
  Modify the application's build.gradle so that the application module depends on the source code. The example is as follows:

```
dependencies {
        implementation project(':hub:hub-device-android')
}
```

Connection Authentication

Edit the configuration information in the app-config.json file, you can read the corresponding data in IoTMqttFragment.java:

```
{
```

```
            "PRODUCT_ID":         "",
            "DEVICE_NAME":        "",
            "DEVICE_PSK":         "",
            "SUB_PRODUCT_ID":     "",
            "SUB_DEV_NAME":       "",
            "SUB_PRODUCT_KEY":    "",
            "TEST_TOPIC":         "",
            "SHADOW_TEST_TOPIC":  "",
            "PRODUCT_KEY":        ""
}
```

The SDK provides two authentication methods: Certificate Authentication and Key Authentication. You need to select and configure according to the authentication type of the created product.

- For Key Authentication, fill in the parameters PRODUCT_ID, DEVICE_NAME, and DEVICE_PSK in the app-config.json configuration. The SDK will automatically generate a signature based on the device configuration information to serve as a credential for accessing the IoT Hub.
- For Certificate Authentication, fill in the PRODUCT_ID, DEVICE_NAME, etc., in the app-config.json configuration and read the contents of the device certificate and device private key files. There are two ways to read:
  - Read through AssetManager, at this time, you need to create an assets directory under the path `hub/hub-android-demo/src/main` and place the device certificate and private key in that directory.
  - Read through InputStream, at this time, you need to pass in the full path information of the device certificate and private key.

1.1 After successfully reading the certificate file and private key file, you need to set the certificate name mDevCertName and private key name mDevKeyName in IoTMqttFragment.java.

```
private String mDevCertName = "YOUR_DEVICE_NAME_cert.crt";
private String mDevKeyName  = "YOUR_DEVICE_NAME_private.key";
```

1.2 After configuration is completed, you can call the relevant interfaces of MQTT connection in the SDK in the project to complete the device connection.

```
 mMqttConnection = new TXGatewayConnection(mContext, mBrokerURL,
mProductID, mDevName, mDevPSK,null,null ,mMqttLogFlag,
mMqttLogCallBack, mMqttActionCallBack);
```

```
mMqttConnection.connect(options, mqttRequest);
```

SDK for Android Use Instructions

Last updated: 2025-03-19 15:03:19

In addition to the device connection feature, the SDK for Android also provides gateway sub-device, device shadow, and OTA features with the following API descriptions.

MQTT APIs

TXMqttConnection

Method Name	Description
connect	Establishes MQTT connection.
reconnect	Reestablishes MQTT connection.
disConnect	Closes MQTT connection.
publish	Publishes MQTT message.
subscribe	Subscribes to MQTT topic.
unSubscribe	Unsubscribes from MQTT topic.
getConnectStatus	Gets MQTT connection status
setBufferOpts	Sets buffer for disconnection status.
initOTA	Initializes OTA feature.
reportCurrentFirmwareVersion	Reports current device version information to the backend server.
reportOTAState	Reports device upgrade status to the backend server.

MQTT Gateway APIs

TXGatewayConnection

Method Name	Description
connect	Gateway connection.
reconnect	Gateway reconnection.

disConnect	Closes gateway MQTT connection.
publish	Publishes MQTT message.
subscribe	Subscribes to MQTT topic.
unSubscribe	Unsubscribes from MQTT topic.
getConnectStatus	Gets MQTT connection status.
setBufferOpts	Sets buffer for disconnection status.
initOTA	Initializes OTA feature.
reportCurrentFirmwareVersion	Reports current device version information to the backend server.
reportOTAState	Reports device upgrade status to the backend server.
addSubDev	Adds sub-device.
removeSubdev	Removes sub-device.
findSubdev	Finds sub-device.
gatewaySubdevOffline	Sub-device decommission.
gatewaySubdevOnline	Sub-device connection.

Device Shadow APIs

TXShadowConnection

Method Name	Description
connect	Shadow connection.
disConnect	Close shadow connection.
getConnectStatus	Gets MQTT connection status.
update	Updates device shadow document.
get	Gets device shadow document.
registerProperty	Registers device attribute.

unRegisterProperty	Unregisters device attribute.
reportNullDesiredInfo	Reports empty desired information after updating delta information.
setBufferOpts	Sets buffer for disconnection status.
getMqttConnection	Gets txmqttconnection instance.

MQTT remote service client.

TXMqttClient

Method Name	Description
setMqttActionCallBack	Sets mqttaction callback interface.
setServiceConnection	Sets remote service connection callback interface.
init	Initialize remote service client.
startRemoteService	Start remote service.
stopRemoteService	Stop remote service.
connect	Establishes MQTT connection.
disConnect	Closes MQTT connection.
subscribe	Subscribes to MQTT topic.
unSubscribe	Unsubscribes from MQTT topic.
publish	Publishes MQTT message.
setBufferOpts	Sets buffer for disconnection status.
clear	Release the resources.

Shadow remote service client.

TXShadowClient

Method Name	Description
setShadowActionCallBack	Set ShadowAction callback.

setServiceConnection	Sets remote service connection callback interface.
init	Initialize remote service client.
startRemoteService	Start remote service.
stopRemoteService	Stop remote service.
connect	Shadow connection.
disConnect	Disconnect Shadow connection.
getMqttClient	Retrieve MQTT client.
get	Retrieve device shadow.
update	Update device shadow.
registerProperty	Registers device attribute.
unRegisterProperty	Unregisters device attribute.
reportNullDesiredInfo	Reports empty desired information after updating delta information.
setBufferOpts	Sets buffer for disconnection status.
clear	Release the resources.

Firmware upgrade via MQTT channel.

TXMqttClient

Method Name	Description
initOTA	Initializes OTA feature.
reportCurrentFirmwareVersion	Reports current device version information to the backend server.
reportOTAState	Reports device upgrade status to the backend server.

Connection Based on SDK for Java SDK for Java Release Notes

Last updated: 2025-03-19 15:03:40

Code Hosting

Since v1.0.0, the code of the device SDK for Java has been hosted on GitHub.

Version Information

For the version iteration information of the device SDK for Java since v1.0.0, please visit GitHub.

SDK for Java Project Configuration

Last updated: 2025-03-19 15:03:52

IoT Hub device SDK for Java relies on a secure and powerful data channel to enable IoT developers to quickly connect devices to the cloud for two-way communication. You only need to complete the corresponding project configuration to connect devices.

Prerequisites

The product and device have been created according to the Device Access Preparation.

Reference Method

- If you need to conduct project development through jar reference, you can add dependencies in `build.gradle` under the module directory as follows:

```
dependencies {
    ...
    implementation 'com.tencent.iot.hub:hub-device-java:x.x.x'
}
```

> **Note:**
> You can set the above x.x.x to the latest version according to Version Explanation.

- If you need to develop a project through code integration, you can download the SDK for Java source code from GitHub.

Connection Authentication

Device identity authentication supports two methods: key authentication and certificate authentication:

- If using key authentication, you need ProductID, DevName, and DevPSK.
- If using certificate authentication, you need ProductID, CertFile, and PrivateKeyFile.

Below is the sample code for connection authentication:

```
private String mProductID = "YOUR_PRODUCT_ID";
private String mDevName = "YOUR_DEVICE_NAME";
private String mDevPSK = "YOUR_DEV_PSK";
private String mCertFilePath = null;
private String mPrivKeyFilePath = null;
```

```
TXMqttConnection mqttconnection = new TXMqttConnection(mProductID,
mDevName, mDevPSK, new callBack());
mqttconnection.connect(options, null);
try {
        Thread.sleep(20000);
} catch (InterruptedException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
mqttconnection.disConnect(null);
```

SDK for Java Use Instructions

Last updated: 2025-03-19 15:04:05

In addition to the device connection feature, the SDK for Java also provides gateway subdevice and device shadow features with the following APIs.

MQTT APIs

MQTT APIs are defined in the txmqttconnection class and support publishing and subscribing. If you want to support the device shadow feature, you need to use the txshadowconnection class and its methods. The txmqttconnection class APIs are detailed below:

Method Name	Description
connect	Establishes MQTT connection.
reconnect	Reestablishes MQTT connection.
disConnect	Closes MQTT connection.
publish	Publishes MQTT message.
subscribe	Subscribes to MQTT topic.
unSubscribe	Unsubscribes from MQTT topic.
getConnectStatus	Gets MQTT connection status.
setBufferOpts	Sets buffer for disconnection status.

MQTT Gateway APIs

- Devices that don't have direct access to the Ethernet can be connected to the network of the local gateway device first and then connected to the IoT Hub platform through the communication feature of the gateway device.
- For the subdevices that join or leave the LAN, they need to be bound or unbound through the platform.

> **Notes:**
> After a subdevice is connected once, as long as the gateway is successfully connected subsequently, the backend will show that the subdevice is online until it is disconnected.

MQTT gateway APIs are defined in the TXGatewayConnection class as detailed below:

Method Name	Description
connect	Establishes gateway MQTT connection.
reconnect	Reestablishes gateway MQTT connection.
disConnect	Closes gateway MQTT connection.
publish	Publishes MQTT message.
subscribe	Subscribes to MQTT topic.
unSubscribe	Unsubscribes from MQTT topic.
getConnectStatus	Gets MQTT connection status.
setBufferOpts	Sets buffer for disconnection status.
gatewaySubdevOffline	Sub-device decommission.
gatewaySubdevOnline	Sub-device connection.
gatewayBindSubdev	Binds subdevice.
gatewayUnbindSubdev	Unbinds subdevice.

Device Shadow APIs

To support the device shadow feature, use the interfaces in the TXShadowConnection class, as described below:

Method Name	Description
connect	Establishes MQTT connection.
reconnect	Reestablishes MQTT connection.
disConnect	Closes MQTT connection.
publish	Publishes MQTT message.
subscribe	Subscribes to MQTT topic.
unSubscribe	Unsubscribes from MQTT topic.
update	Updates device shadow document.

get	Gets device shadow document.
reportNullDesiredInfo	Reports empty desired information after updating delta information.
setBufferOpts	Sets buffer for disconnection status.
getMqttConnection	Gets txmqttconnection instance.
getConnectStatus	Gets MQTT connection status.

Connection Based on SDK for Python
Python SDK Release Notes

Last updated: 2025-03-19 15:04:26

Code Hosting

Since v1.0.0, the code of the device SDK for Python has been hosted on GitHub.

Version Information

For the version iteration information of the device SDK for Python since v1.0.0, see GitHub.

SDK for Python Project Configuration

Last updated: 2025-03-19 15:04:38

IoT Hub device SDK for Python relies on a secure and powerful data channel to enable IoT developers to quickly connect devices to the cloud for two-way communication. You only need to complete the corresponding project configuration to connect devices.

Prerequisites

The product and device have been created according to the Device Access Preparation.

Reference Method

- If you want to develop a project through import, you can install the SDK as follows:

```
pip3 install tencent-iot-device
```

If you need to view the used SDK version, run the following command:

```
pip3 show --files tencent-iot-device
```

If you need to update the SDK version, run the following command:

```
pip3 install --upgrade tencent-iot-device
```

- If you want to develop a project through code integration, you can download the SDK for Python source code from GitHub.

SDK for Python Use Instructions

Last updated: 2025-03-19 15:04:52

In addition to the device connection feature, the SDK for Python also provides gateway subdevice and device shadow features with the following APIs.

MQTT APIs

MQTT APIs are defined in the hub.py class and support publishing and subscribing. If you want to support the device shadow feature, you need to use the shadow.py class and its methods as detailed below:

Method Name	Description
connect	Establishes MQTT connection.
disconnect	Closes MQTT connection.
subscribe	Subscribes to topic over MQTT.
unsubscribe	Unsubscribes from topic over MQTT.
publish	Publishes message over MQTT.
registerMqttCallback	Registers MQTT callback function.
registerUserCallback	Registers user callback function.
isMqttConnected	Checks whether MQTT is normally connected.
getConnectState	Gets MQTT connection status.
setReconnectInterval	Sets MQTT reconnection attempt interval.
setMessageTimout	Sets message sending timeout period.
setKeepaliveInterval	Sets MQTT keepalive interval.

MQTT Gateway APIs

- Devices that don't have direct access to the Ethernet can be connected to the network of the local gateway device first and then connected to the IoT Hub platform through the communication feature of the gateway device.
- For the subdevices that join or leave the LAN, they need to be bound or unbound through the platform.

> **Note:**
> After a subdevice is connected once, as long as the gateway is successfully connected subsequently, the backend will show that the subdevice is online until it is disconnected.

MQTT gateway APIs are defined in gateway.py class as detailed below:

Method Name	Description
gatewayInit	Initializes gateway.
isSubdevStatusOnline	Determine whether sub-devices are online.
updateSubdevStatus	Updates subdevice's connection status.
gatewaySubdevGetConfigList	Gets subdevice list from configuration file.
gatewaySubdevOnline	Proxies subdevice connection.
gatewaySubdevOffline	Agent subdevice offline.
gatewaySubdevBind	Binds subdevice.
gatewaySubdevUnbind	Unbinds subdevice.
gatewaySubdevSubscribe	Proxies subdevice subscription.

Dynamic Registration APIs

If you want to use the dynamic registration feature, you need to use the APIs in the hub.py class as detailed below:

Method Name	Description
dynregDevice	Gets the dynamic registration information of device.

OTA interface

If you want to use the OTA feature, you need to use the APIs in the hub.py class as detailed below:

Method Name	Description
otaInit	OTA initialization.

otaIsFetching	Determines whether the download is in progress.
otaIsFetchFinished	Determines whether the download is completed.
otaReportUpgradeSuccess	Reports update success message.
otaReportUpgradeFail	Reports update failure message.
otaIoctlNumber	Retrieves int type information such as download firmware size.
otaIoctlString	Retrieves string type information such as download firmware MD5.
otaResetMd5	Resets MD5 information.
otaMd5Update	Updates MD5 information.
httpInit	Initializes HTTP.
otaReportVersion	Reports the information of current firmware version.
otaDownloadStart	Starts firmware download.
otaFetchYield	Reads firmware.