IoT connectors for MDK-Middleware

AN 312, June 2018, V 1.0

feedback@keil.com

Abstract

End-note devices in the Internet of Things (IoT) collect, process and exchange data. Such devices are frequently connected via the Internet to a cloud service that provides processing power, data analytics, and storage capabilities. A cloud connector client is a software interface which runs in the IoT end-note device and establishes the connection to a cloud service.

Many cloud service providers offer open-source software that implements a cloud connector for an embedded system. Arm adopted these cloud connectors for Keil MDK-Middleware using the reliable networking stack for communication with the cloud service. This application note describes the cloud connectors that are available and explains their basic operation.

Contents

Abstract 1

Introduction 1

Packs 2

MQTT 2

IoT clients 2

- Amazon AWS IoT 3
- Google Cloud IoT 4
- IBM Watson IoT 5
- Microsoft Azure IoT Hub 6
- Paho MQTT 7

Introduction

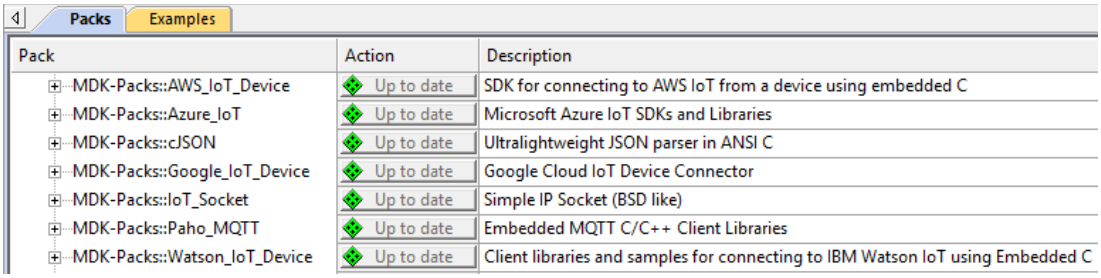
The public MDK-Packs repository on GitHub (https://github.com/MDK-Packs/) contains cloud connector clients for the following cloud providers:

- Amazon AWS IoT
- Google Cloud IoT
- IBM Watson IoT
- Microsoft Azure IoT Hub
- Paho MQTT (Eclipse)

Most of these IoT clients use the MQTT protocol.

Packs

The following MDK-Packs provide the basic building blocks that are required to connect to the cloud providers. These software packs are available from the Pack Installer:



MQTT

MQTT a lightweight messaging protocol for IoT applications. It is designed for networks that are low-bandwidth, high-latency and generally unreliable. The protocol is quite simple and thus suitable for embedded systems. It supports the exchange of messages between a client and a server (here called message broker) and is based on a publish/subscribe model. The major functions are:

- Connect (with authentication: client ID, username and password)
- Publish (message to topic)
- Subscribe (to topic)
- Disconnect

The client connects to a message broker and publishes messages to topics and subscribes to topics to receive messages. The use of topics is not defined in the MQTT protocol itself but is left to the cloud service providers.

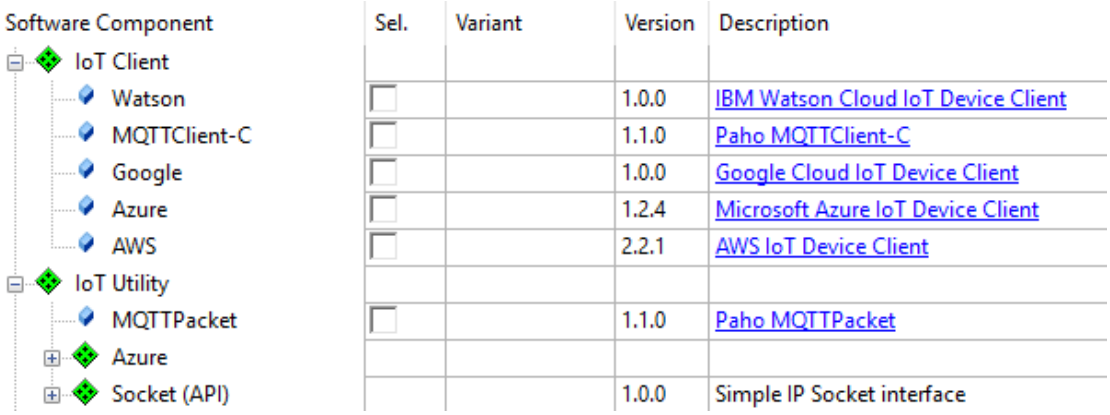
With MQTT, it is possible to use different authentication methods, depending on the cloud service provider. It communicates over TCP/IP using a TCP socket (in case of a non-secure connection) or a TLS socket (in case of a secure connection with encryption). Typically, the cloud service providers only allow secure connections.

IoT clients

The following MDK-Pack utilities are required by the IoT clients:

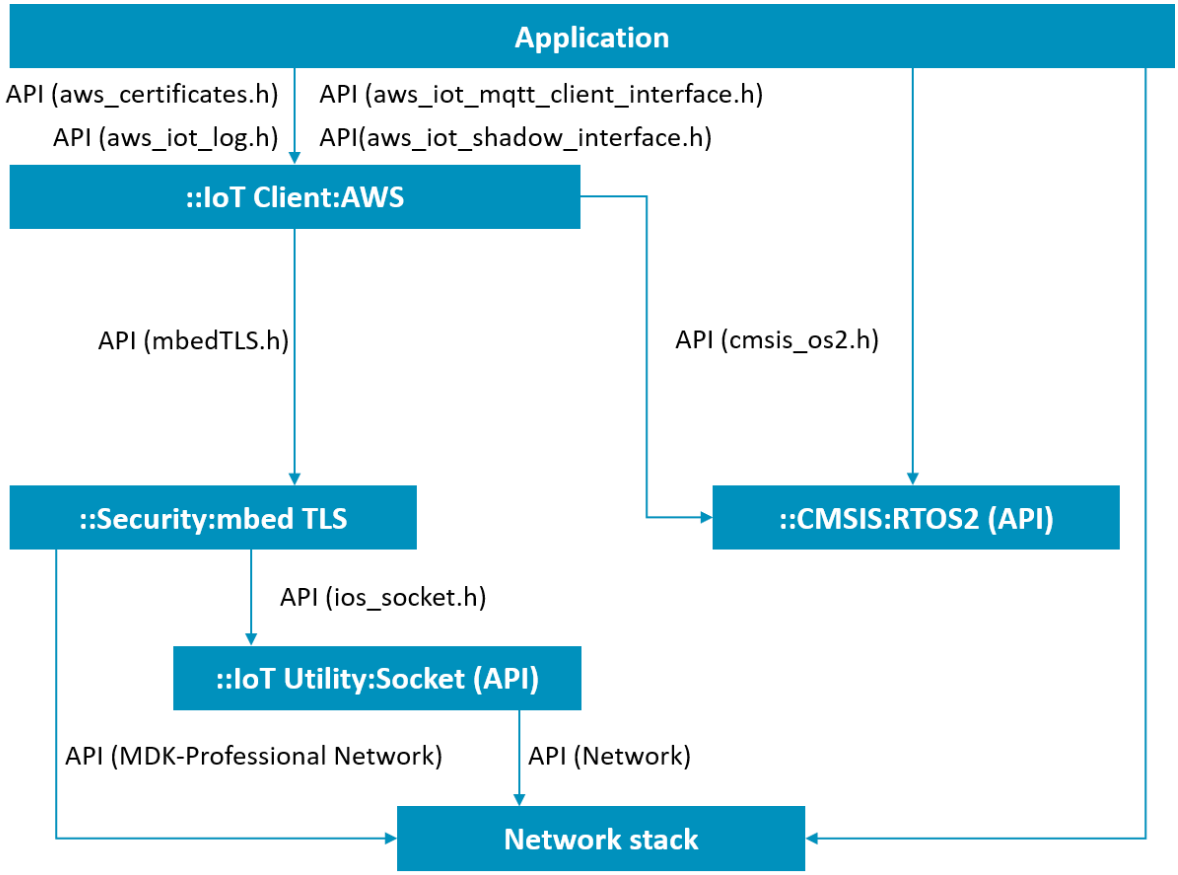
- IoT_Socket: Simple IP Socket API (BSD like) and implementation (which currently only available for the for MDK-Middleware network stack)
- cJSON: Ultra-light weight JSON parser based on the cJSON GitHub project

The available RTE components are:



The following sections provide further details about each IoT client, including the software structure, the APIs used, and component dependencies.

Amazon AWS IoT



Software flow

First, the application initializes and starts the CMSIS-RTOS v2 based real-time operating system. Then, it initializes and prepares the network stack which should be active at that point (for example, an IP address should be acquired via DHCP or static assignment).

Note: The AWS client always uses TLS sockets (for a secure connection) through the **::Security:mbed TLS** interface. The latter also uses the **::IoT Utility:Socket (API)** for underlying TCP sockets or the native interface in case of the network component of MDK-Professional.

The certification authority's (CA) certificate for server verification, as well as the client certificate and the private key for client authentication need to be provided. The user code template **::IoT Client:AWS:Certificates** shows a dummy implementation.

More information about the AWS client is available on GitHub:
https://github.com/aws/aws-iot-device-sdk-embedded-C

The following user code templates are available as a starting point:

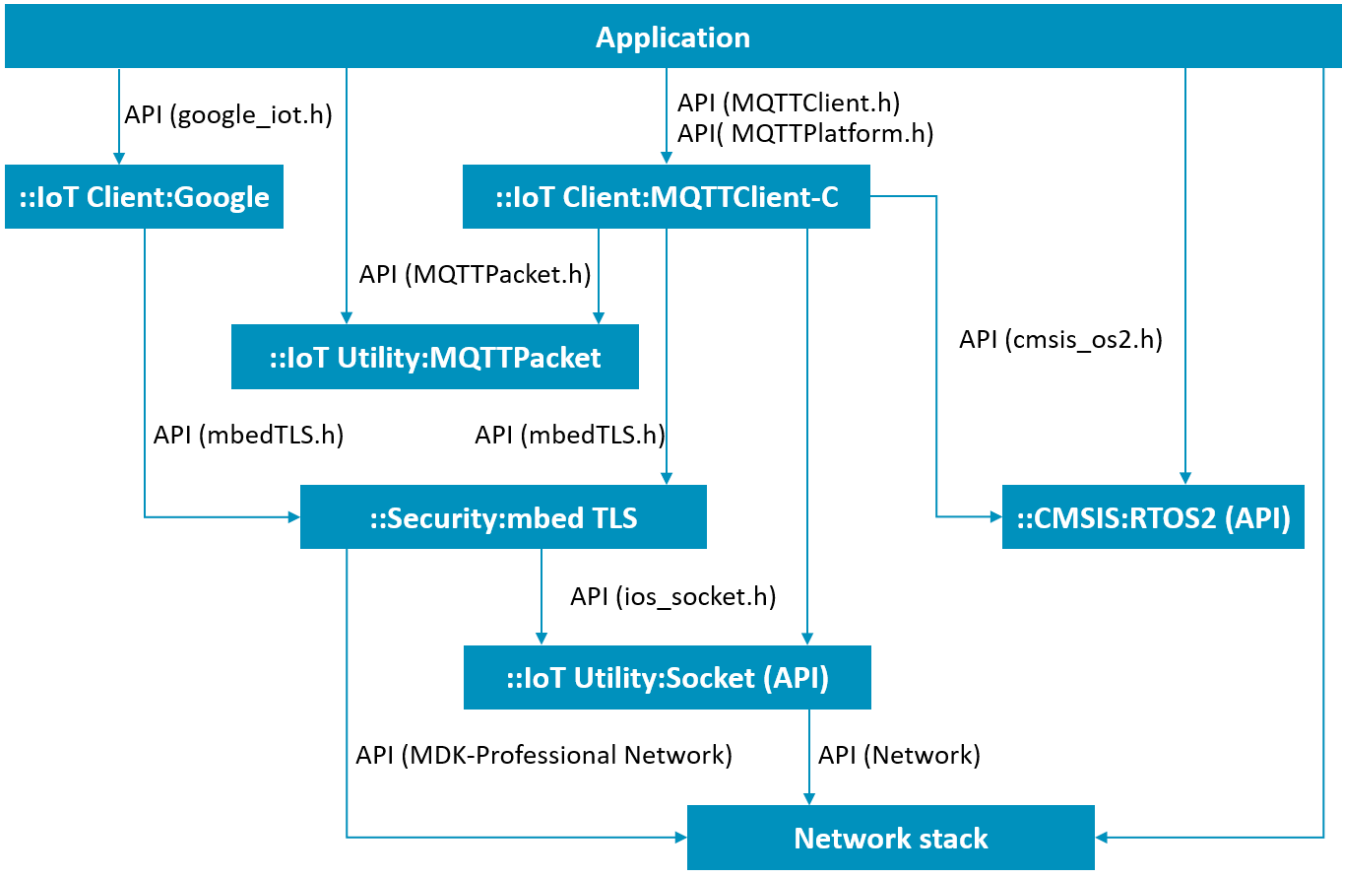
::IoT Client:AWS:Subscribe and publish

::IoT Client:AWS:Shadow

::IoT Client:AWS:Shadow (console echo)

A step-by-step guide on how to create an application in MDK is available in the software pack's **Doc** folder: **AWS_IoT_MDK.html**.

Google Cloud IoT



Software flow

First, the application initializes and starts the CMSIS-RTOS v2 based real-time operating system. Then, it initializes and prepares the network stack which should be active at that point.

Note: The MQTT client in this case uses TLS sockets (for a secure connection) through the **::Security:mbed TLS** interface. The latter also uses the **::IoT Utility:Socket (API)** for underlying TCP sockets or the native interface in case of the network component of MDK-Professional.

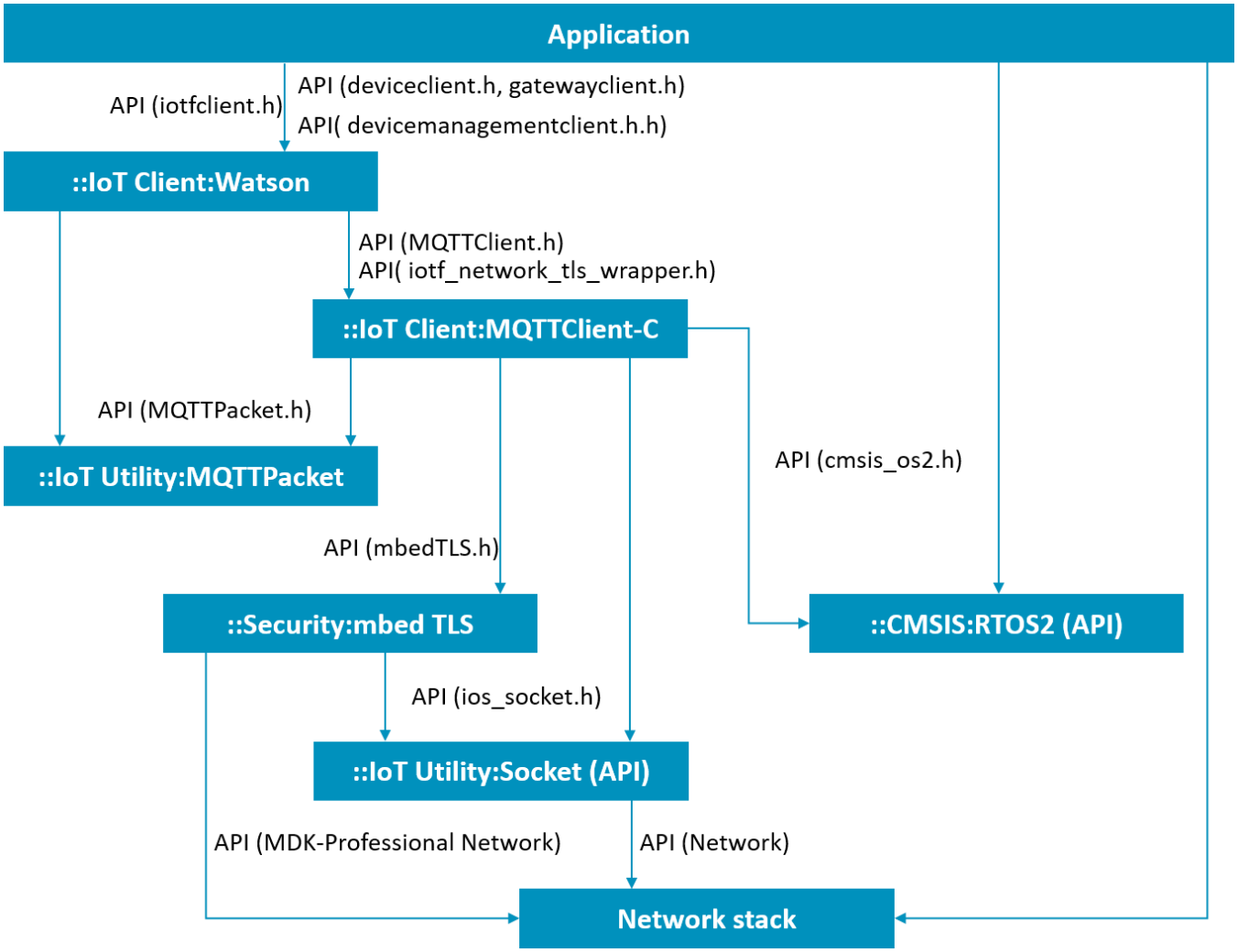
The MQTT client is then used through its API: MQTTClient.h, MQTTPlatform.h, and MQTTPacket.h:

1. The client connects to the server on socket level by calling the functions NetworkInit and NetworkConnectTLS (for a secure connection) (from MQTTPlatform.h). The CA certificate for server verification can be provided and optionally a client certificate and private key for client authentication (if requested by the server) (refer to the user code template **::IoT Client:MQTTClient-C:Certificates**).
2. Provide a private key for the Jason Web Tokens (JWT) (template **::IoT Client:Google:Device Private Key**).
3. The JWT is generated with the function google_iot_jwt (from google_iot.h)
4. The MQTT client connects on MQTT level using MQTTClientInit and MQTTConnect (MQTTClient.h), using the GOOGLE_IOT_CLIENT_ID macro for the client ID and the JWT as the password.
5. The MQTT client exchanges messages using MQTTPublish and MQTTSubscribe (MQTTClient.h)
6. Disconnect the MQTT from the server on socket level by calling NetworkDisconnect (MQTTPlatform.h)

The user code template **::IoT Client:Google:Simple Demo** provides a starting point for your own implementations. A step-by-step guide is available in the software pack's **Doc** folder: **Google_IoT_MDK.html**.

The Google IoT cloud defines specific topics. More information is available here:
https://cloud.google.com/iot/docs/how-tos/config/configuring-devices
https://cloud.google.com/iot/docs/how-tos/config/getting-state
https://cloud.google.com/iot/docs/how-tos/mqtt-bridge

IBM Watson IoT



Software flow

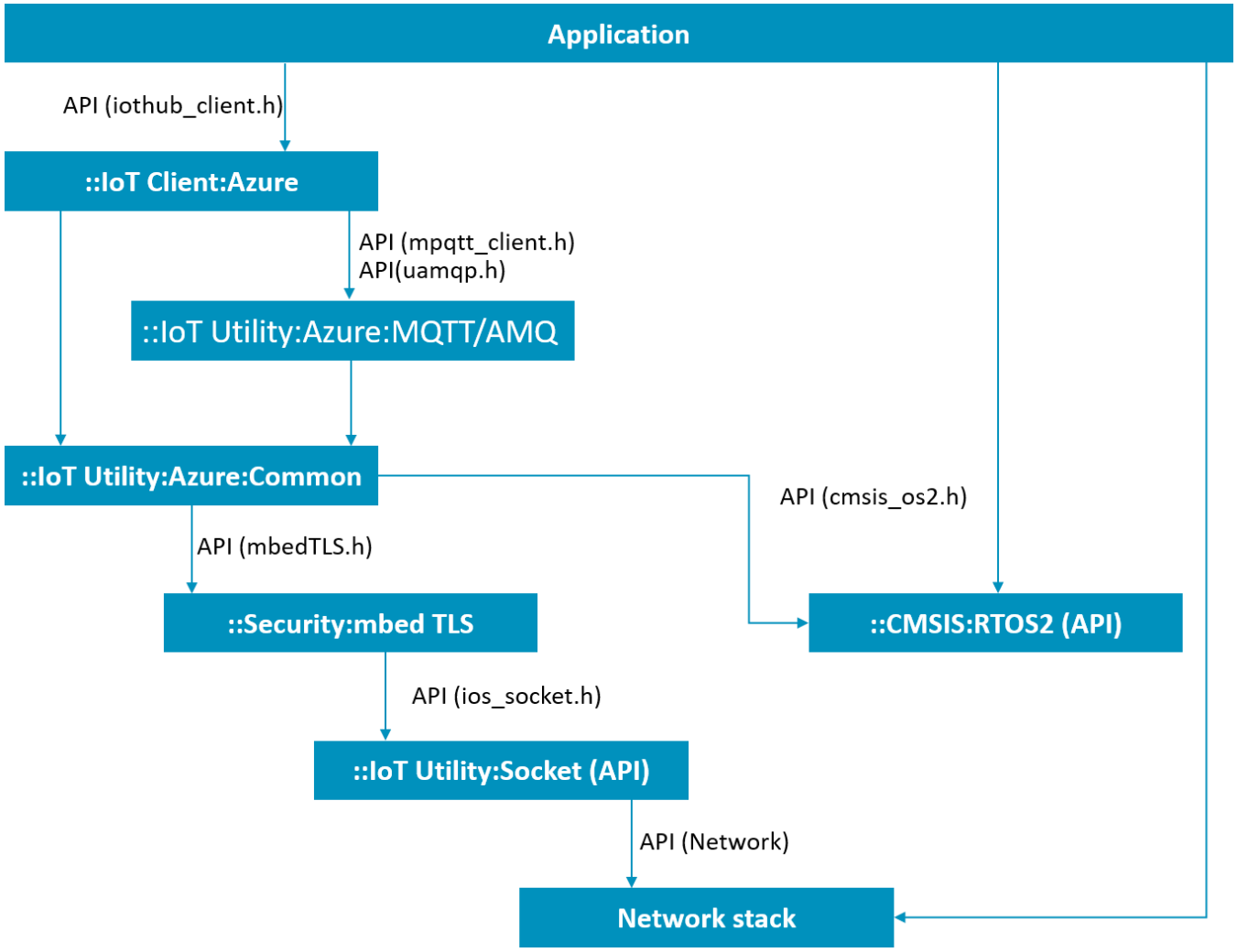
First, the application initializes and starts the CMSIS-RTOS v2 based real-time operating system. Then, it initializes and prepares the network stack which should be active at that point.

Note: The MQTT client in this case uses TLS sockets (for a secure connection) through the **::Security:mbed TLS** interface. The latter also uses the **::IoT Utility:Socket (API)** for underlying TCP sockets or the native interface in case of the network component of MDK-Professional. Non-secure connections can only be used when connecting to IBM Quickstart.

The Watson IoT client provides APIs for devices, gateways and managed devices. More information is available here: https://github.com/ibm-watson-iot/iot-embeddedc/
The user code templates
::IoT Client:Watson:Hello World
::IoT Client:Watson:Device
::IoT Client:Watson:Device Management
::IoT Client:Watson:Device Management Actions
::IoT Client:Watson:Device Management Firmware Actions
::IoT Client:Watson:Gateway
provide a starting point for your own implementations.

A step-by-step guide on how to create an application in MDK is available in the software pack's **Doc** folder: **Watson_IoT_MDK.html**.

Microsoft Azure IoT Hub



Software flow

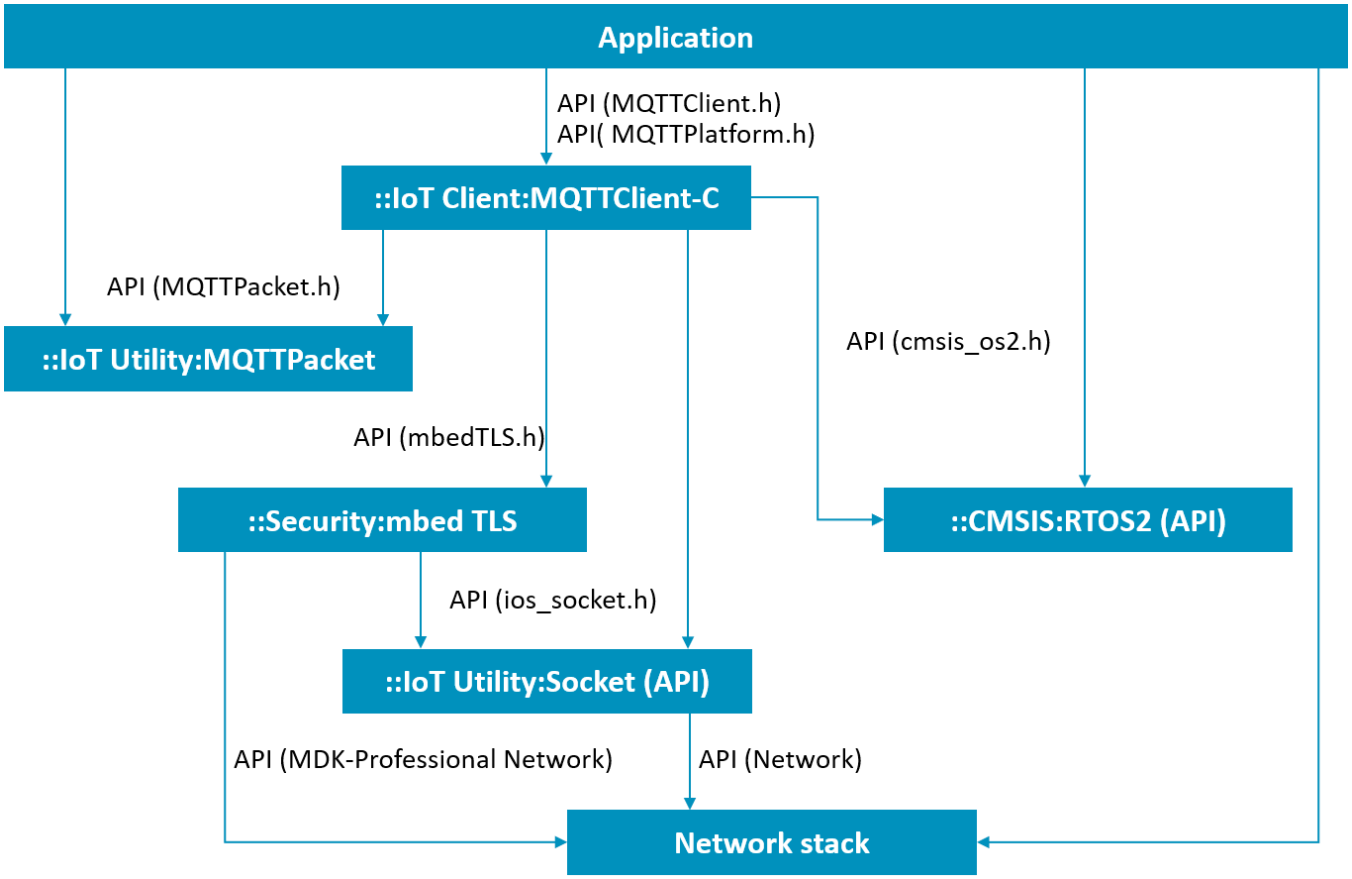
First, the application initializes and starts CMSIS-RTOS2. Then, it initializes and prepares the Network stack which should be active at that point (ex: IP address acquired through DHCP).

Note: The Azure client in this case uses TLS sockets (for a secure connection) through the **::Security:mbed TLS** interface. The latter uses the **::IoT Utility:Socket (API)** for underlying TCP sockets.

More information about Azure IoT SDK is available here: https://github.com/Azure/azure-iot-sdk-c/

The following template is available as a starting point:
"IoT Client:Azure:Telemetry sample"
A step-by-step guide on how to create an application in MDK is available in the software pack's **Doc** folder: **Azure_IoT_MDK.html**.

Paho MQTT



Software flow

First, the application initializes and starts the CMSIS-RTOS v2 based real-time operating system. Then, it initializes and prepares the network stack which should be active at that point (for example, an IP address should be acquired via DHCP or static assignment).

Note: The MQTT client uses TCP sockets (for a non-secure connection) through the **::IoT Utility:Socket (API)** interface or TLS sockets (for a secure connection) through the **::Security:mbed TLS** interface. The latter also uses the **::IoT Utility:Socket (API)** for underlying TCP sockets or the native interface in case of the network component of MDK-Professional.

The MQTT client is then used through its API: MQTTClient.h, MQTTPlatform.h, and MQTTPacket.h:

1. The client connects to the server on socket level by calling the functions NetworkInit, NetworkConnect (for a non-secure connection) or NetworkConnectTLS (for a secure connection) (from MQTTPlatform.h). When secure connection is used, the CA certificate for server verification can be provided and optionally a client certificate and private key for client authentication (if requested by the server) (refer to the user code template **::IoT Client:MQTTClient-C:Certificates**).
2. The MQTT client connects on MQTT level using MQTTClientInit and MQTTConnect (MQTTClient.h)
3. The MQTT client exchanges messages using MQTTPublish and MQTTSubscribe (MQTTClient.h)
4. Disconnect the MQTT from the server on socket level by calling NetworkDisconnect (MQTTPlatform.h):

The user code template **::IoT Client:MQTTClient-C:MQTT Echo** provides a starting point for your own implementations.

A step-by-step guide on how to create an application in MDK is available in the software pack's **Doc** folder: **Paho_MQTT_MDK.html**.