

Abstract

End-node 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 to be used with either the MDK-Middleware network stack or with WiFi devices. This application note describes the cloud connectors that are available and explains their basic operation.

Contents

Abstract	1
Introduction.....	1
MQTT	2
MDK-Packs.....	2
cJSON	2
IoT_Socket.....	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)

MQTT

Most of these IoT clients use the [MQTT](#) protocol. It is 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:

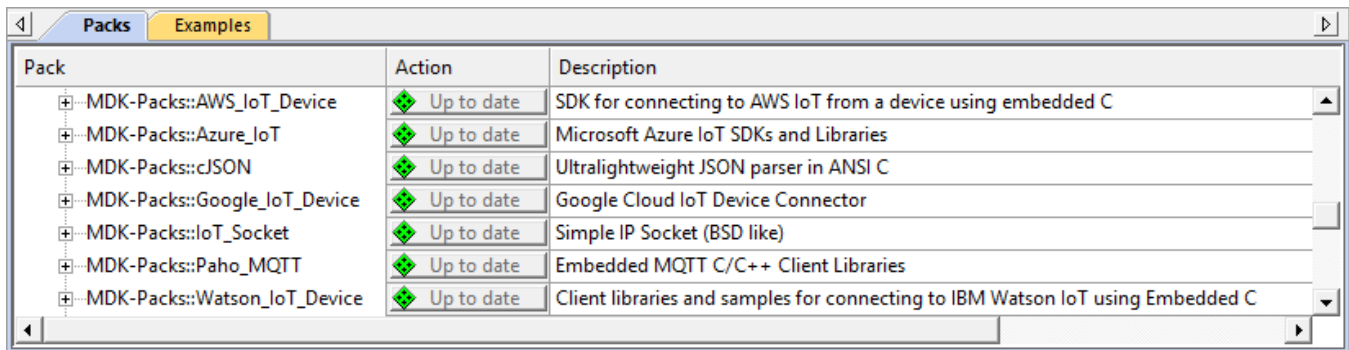
- Connect (with authentication: client ID, username and password)
- Publish (message to topic)
- Subscribe (to topic)
- Disconnect

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.

MDK-Packs

The following MDK-Packs (available from the Pack Installer) provide the basic building blocks that are required to connect to the cloud providers:



Pack	Action	Description
MDK-Packs::AWS_IoT_Device	Up to date	SDK for connecting to AWS IoT from a device using embedded C
MDK-Packs::Azure_IoT	Up to date	Microsoft Azure IoT SDKs and Libraries
MDK-Packs::cJSON	Up to date	Ultralightweight JSON parser in ANSI C
MDK-Packs::Google_IoT_Device	Up to date	Google Cloud IoT Device Connector
MDK-Packs::IoT_Socket	Up to date	Simple IP Socket (BSD like)
MDK-Packs::Paho_MQTT	Up to date	Embedded MQTT C/C++ Client Libraries
MDK-Packs::Watson_IoT_Device	Up to date	Client libraries and samples for connecting to IBM Watson IoT using Embedded C

cJSON

This pack contains the ultra-light weight JSON parser based on the [cJSON GitHub project](#).

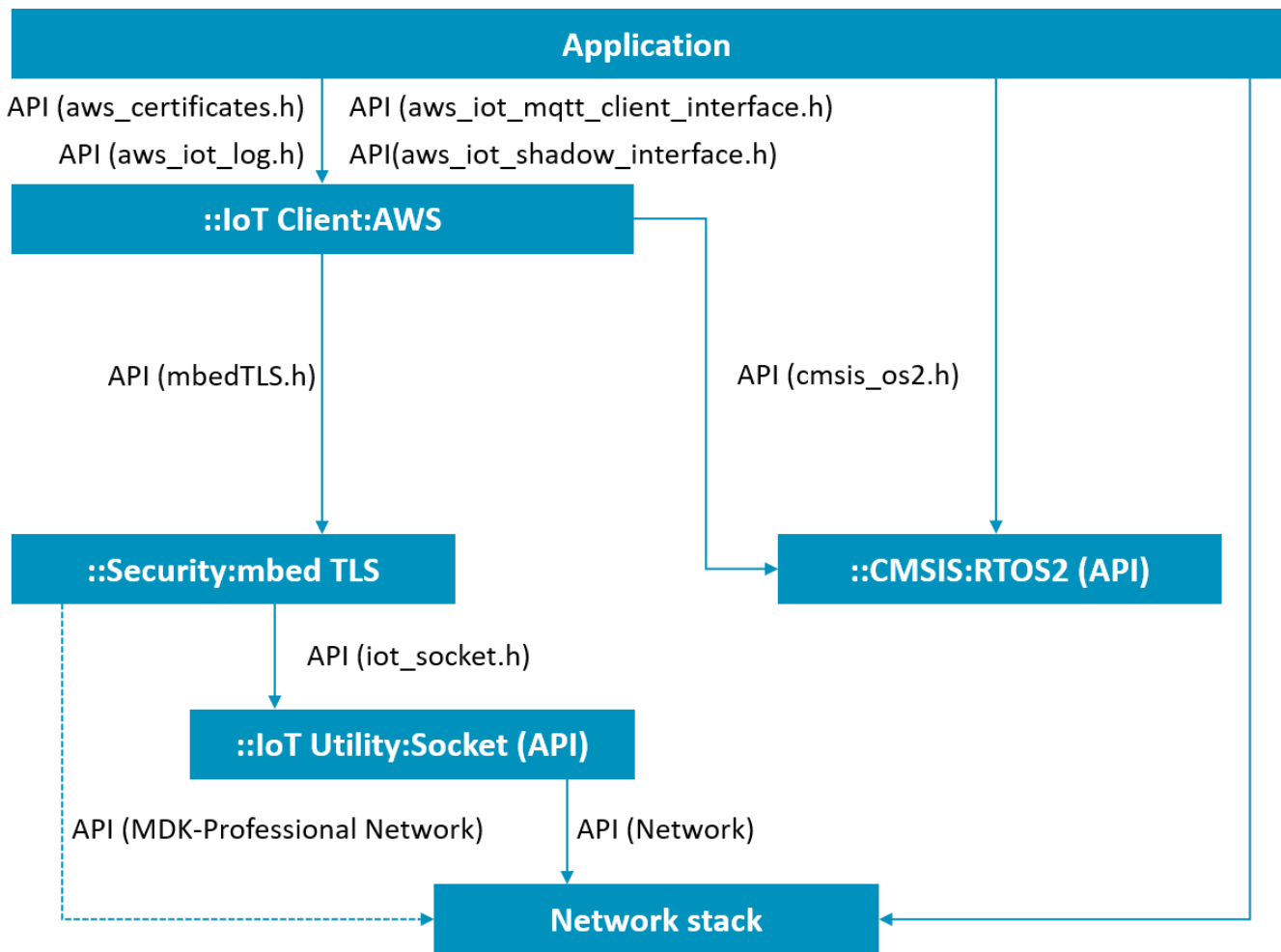
IoT_Socket

The IoT_Socket pack provides a simple IP Socket API (BSD like) and implementation which supports the MDK-Middleware network stack or CMSIS-Driver WiFi implementations). In the IoT client block diagrams, all blocks marked with **Network stack** can either be MDK-Middleware or WiFi:

Socket (API)		Version	Description
MDK Network	<input type="checkbox"/>	1.1.0	Simple IP Socket interface
WiFi	<input type="checkbox"/>	1.1.0	IoT Socket implementation with MDK::Network
	<input type="checkbox"/>	1.0.0	IoT Socket implementation with WiFi Driver

If you are using the socket implementation with WiFi, you require a device that is supported by a [WiFi CMSIS-Driver](#). Other software packs are available that provide these drivers, for example the Arm:CMSIS-Driver pack. It is available via Pack Installer. Note that you also require a CMSIS-Driver for the communication between the WiFi device and your target microcontroller, usually based on SPI or UART.

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:

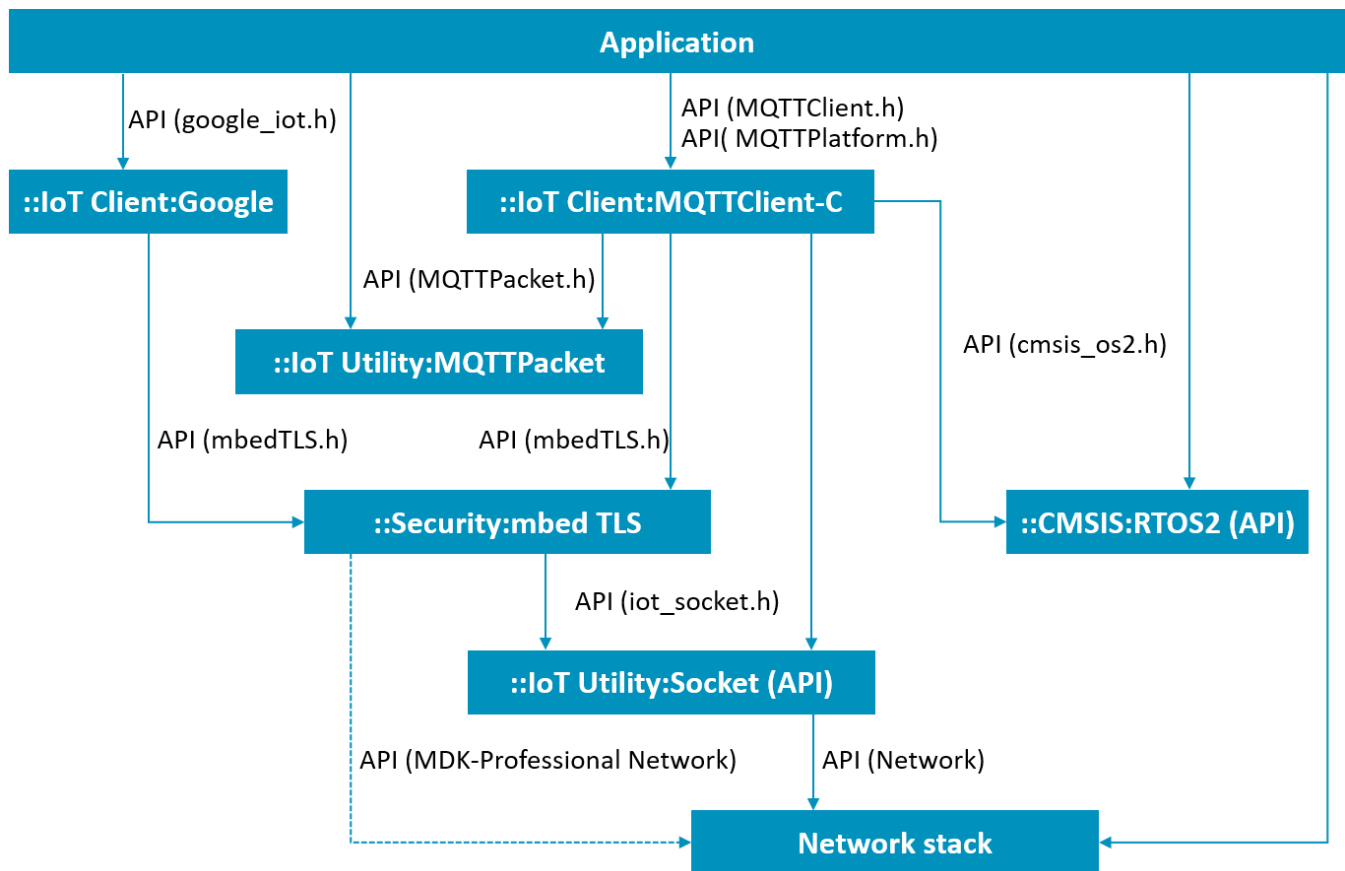
::IoT Client:AWS:Subscribe and publish

::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)

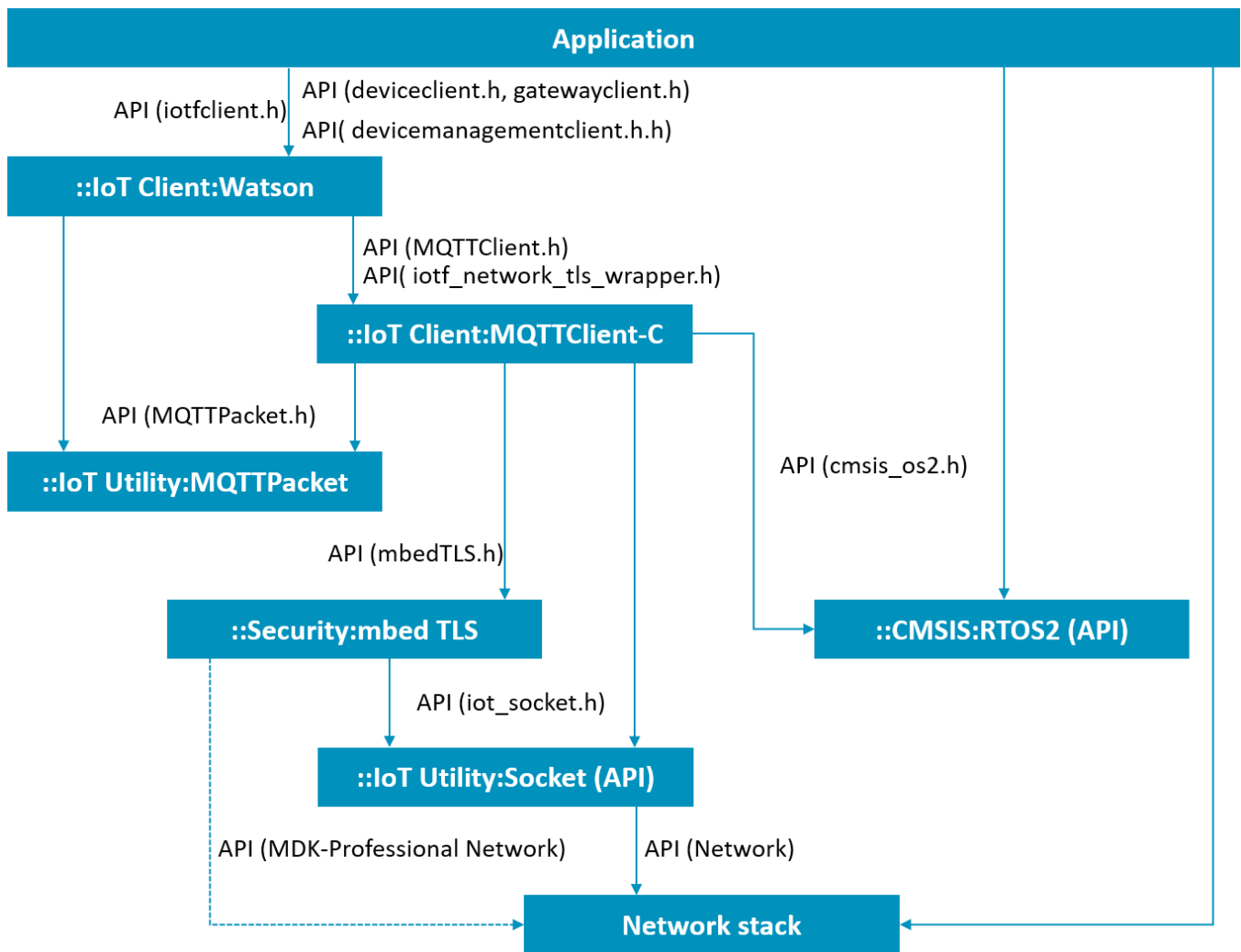
The user code template **::IoT Client:Google:Simple Demo** provides a starting point for your own implementations. A step-by-step guide is available in the software pack's Doc folder: **Google_IoT_MDK.html**.

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>



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/>

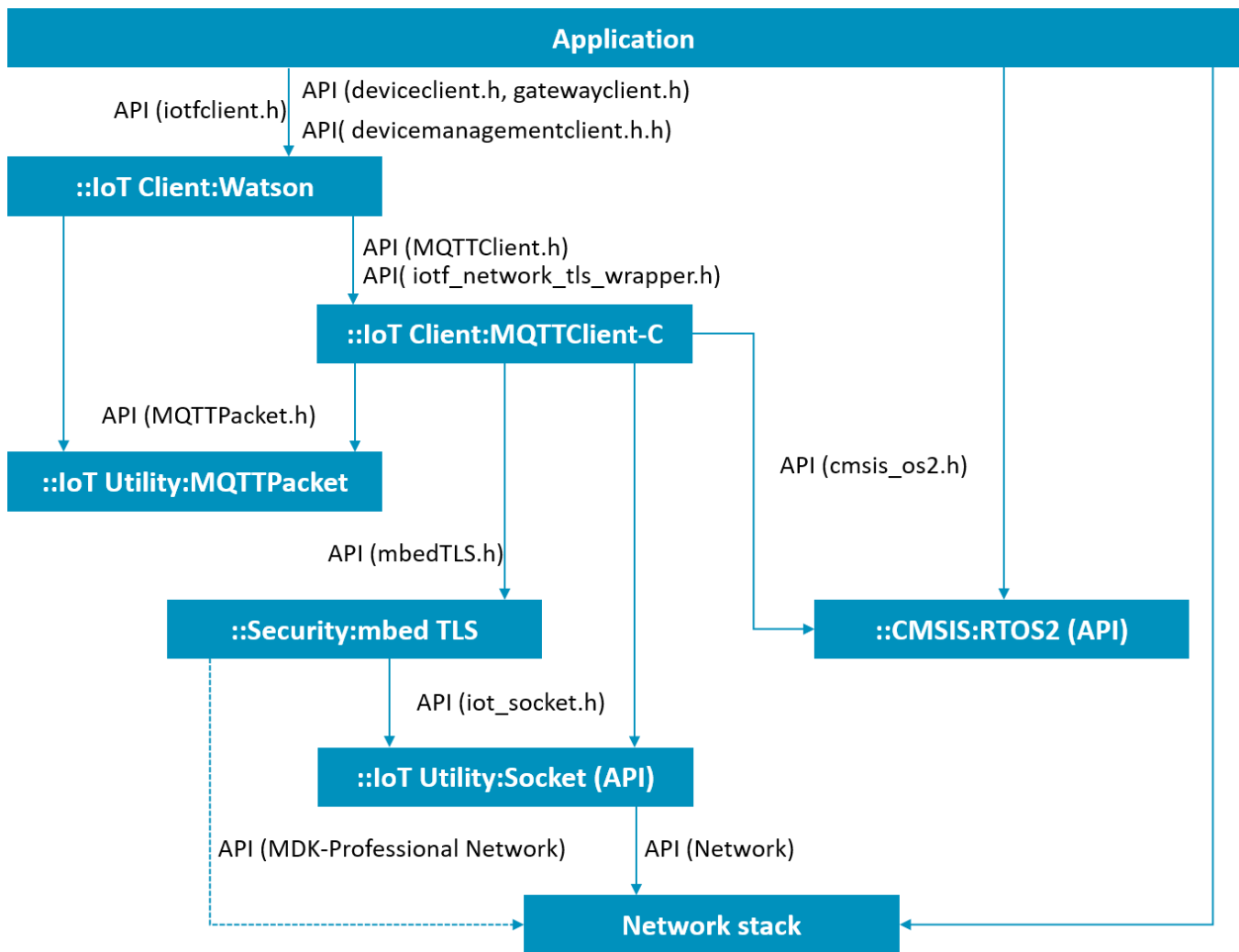
The user code templates

- ::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/>

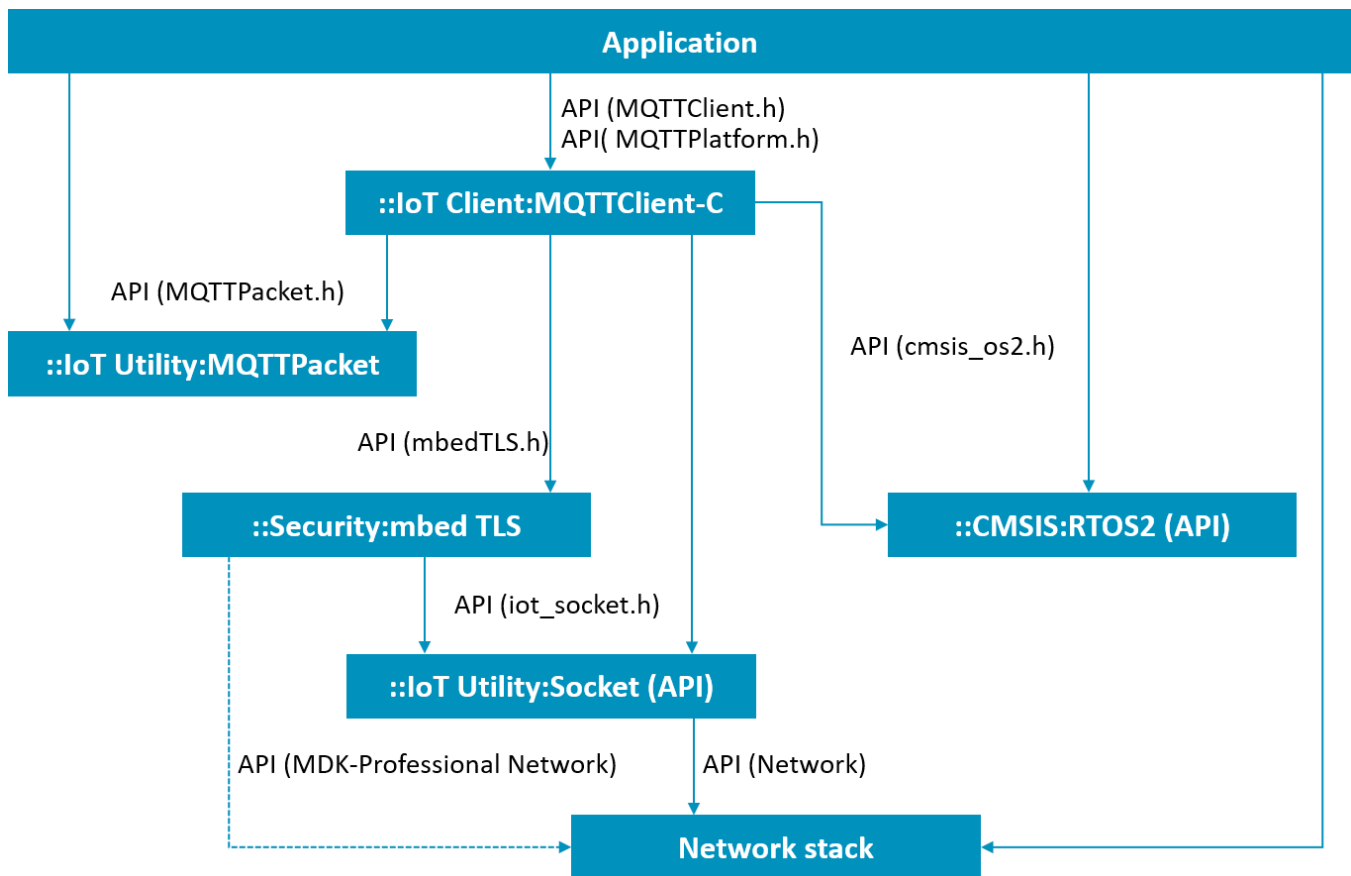
The following template is available as a starting point:

“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**.