IN THE UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
TYLER DIVISION

LUCIO DEVELOPMENT LLC,	§	
	§	
Plaintiff,	§	Case No: 6:19-cv-283
	§	
vs.	§	PATENT CASE
	§	
MICROSOFT CORP.,	§	
	§	
Defendant.	§	
	§	

COMPLAINT

Plaintiff Lucio Development LLC ("Plaintiff" or "Lucio") files this Complaint against Microsoft Corp. ("Defendant" or "Microsoft") for infringement of United States Patent No. 7,069,546 (hereinafter "the '546 Patent").

PARTIES AND JURISDICTION

1. This is an action for patent infringement under Title 35 of the United States Code. Plaintiff is seeking injunctive relief as well as damages.

2. Jurisdiction is proper in this Court pursuant to 28 U.S.C. §§ 1331 (Federal Question) and 1338(a) (Patents) because this is a civil action for patent infringement arising under the United States patent statutes.

3. Plaintiff is a Texas limited liability company with its office address at 555 Republic Dr., Suite 200, Plano, Texas 75074.

4. On information and belief, Defendant is a Washington corporation with its principal place of business at One Microsoft Way, Redmond, Washington, 98052. On information and belief, Defendant may be served with process through its registered agent

Corporation Service Company d/b/a CSC, 211 E. 7th St., Suite 620, Austin, Texas 78701.

5. This Court has personal jurisdiction over Defendant because Defendant has committed, and continues to commit, acts of infringement in this District, has conducted business in this District, and/or has engaged in continuous and systematic activities in this District.

6. On information and belief, Defendant's instrumentalities that are alleged herein to infringe were and continue to be used, imported, offered for sale, and/or sold in this District.

VENUE

7. Venue is proper in this District pursuant to 28 U.S.C. §1400(b) because acts of infringement are occurring in this District and Defendant has a regular and established place of business in this District at 2601 Preston Rd #1176, Frisco, TX 75034.

COUNT I
(INFRINGEMENT OF UNITED STATES PATENT NO. 7,069,546)

8. Plaintiff incorporates paragraphs 1 through 7 herein by reference.

9. This cause of action arises under the patent laws of the United States and, in particular, under 35 U.S.C. §§ 271, *et seq*.

10. Plaintiff is the owner by assignment of the '546 Patent with sole rights to enforce the '546 Patent and sue infringers.

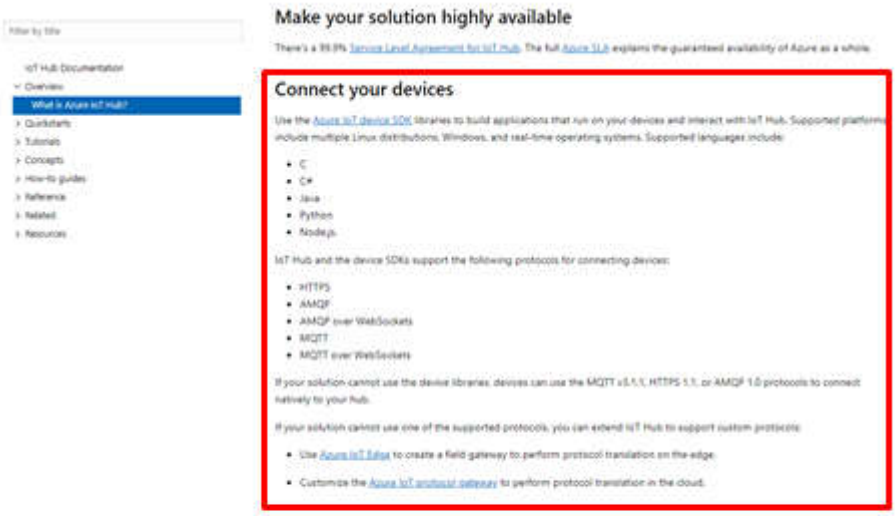
11. A copy of the '546 Patent, titled "Generic Framework for Embedded Software Development," is attached hereto as Exhibit A.

12. The '546 Patent is valid, enforceable, and was duly issued in full compliance with Title 35 of the United States Code.

13. On information and belief, Defendant has infringed and continues to infringe one or more claims, including at least Claim 1, of the '546 Patent by making, using, importing,

selling, and/or offering for sale a software platform for embedded software development, which is covered by at least Claim 1 of the '546 Patent. Defendant has infringed and continues to infringe the '546 Patent directly in violation of 35 U.S.C. § 271.

14. Defendant, sells, offers to sell, and/or uses embedded software development packages including, without limitation, the Azure IoT Hub software developer kit, and any similar products ("Product"), which infringe at least Claim 1 of the '546 Patent. The Product practices a method for producing embedded software. For example, Defendant provides the Azure IoT Hub Software Development Kit (such as IoT Hub Device SDKs and/or IoT Hub Service SDKs) which integrates IoT (Internet of Things) devices running on different operating systems/platforms/frameworks (such as MBED2, Arduino, Windows CE, .Net Standard 1.3, and/or Xamarian iOS, Android and/or UWP). Microsoft and/or its customers specifically use Azure IoT Hub Software Development Kit (such as IoT Hub Device SDKs and/or IoT Hub Service SDKs) which integrates IoT (Internet of Things) devices running on different operating systems/platforms/frameworks (such as MBED2, Arduino, Windows CE, .Net Standard 1.3, and/or Xamarian iOS, Android and/or UWP) to produce embedded software. Certain elements of this limitation are illustrated in the screenshots below and in the screenshots referenced in connection with other elements herein.



Make your solution highly available

There's a 99.9% Service Level Agreement for IoT Hub. The full Azure SLA explains the guaranteed availability of Azure as a whole.

Connect your devices

Use the Azure IoT device SDK libraries to build applications that run on your devices and interact with IoT Hub. Supported platforms include multiple Linux distributions, Windows, and real-time operating systems. Supported languages include:

- C
- C#
- Java
- Python
- Node.js

IoT Hub and the device SDKs support the following protocols for connecting devices:

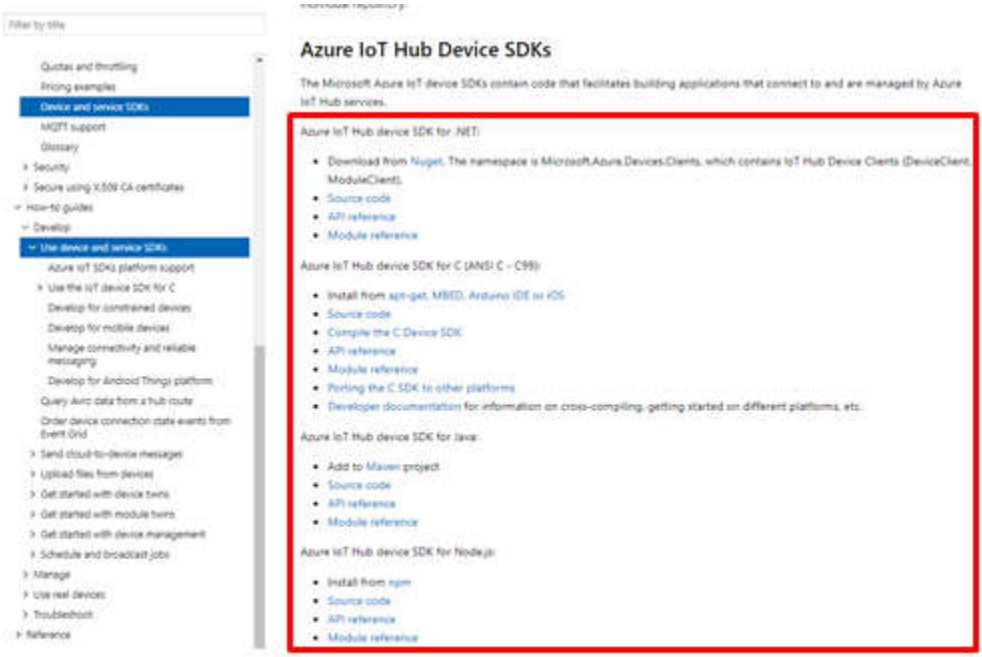
- HTTPS
- AMQP
- AMQP over WebSockets
- MQTT
- MQTT over WebSockets

If your solution cannot use the device libraries, devices can use the MQTT v3.1.1, HTTPS 1.1, or AMQP 1.0 protocols to connect natively to your hub.

If your solution cannot use one of the supported protocols, you can extend IoT Hub to support custom protocols:

- Use Azure IoT Edge to create a field gateway to perform protocol translation on the edge.
- Customize the Azure IoT protocol gateway to perform protocol translation in the cloud.

Source: https://docs.microsoft.com/en-us/azure/iot-hub/about-iot-hub



Azure IoT Hub Device SDKs

The Microsoft Azure IoT device SDKs contain code that facilitates building applications that connect to and are managed by Azure IoT Hub services.

Azure IoT Hub device SDK for .NET:

- Download from Nuget. The namespace is Microsoft.Azure.Devices.Clients, which contains IoT Hub Device Clients (DeviceClient, ModuleClient).
- Source code
- API reference
- Module reference

Azure IoT Hub device SDK for C (ANSI C – C99):

- Install from apt-get, MBED, Arduino IDE or iOS
- Source code
- Compile the C Device SDK
- API reference
- Module reference
- Porting the C SDK to other platforms
- Developer documentation for information on cross-compiling, getting started on different platforms, etc.

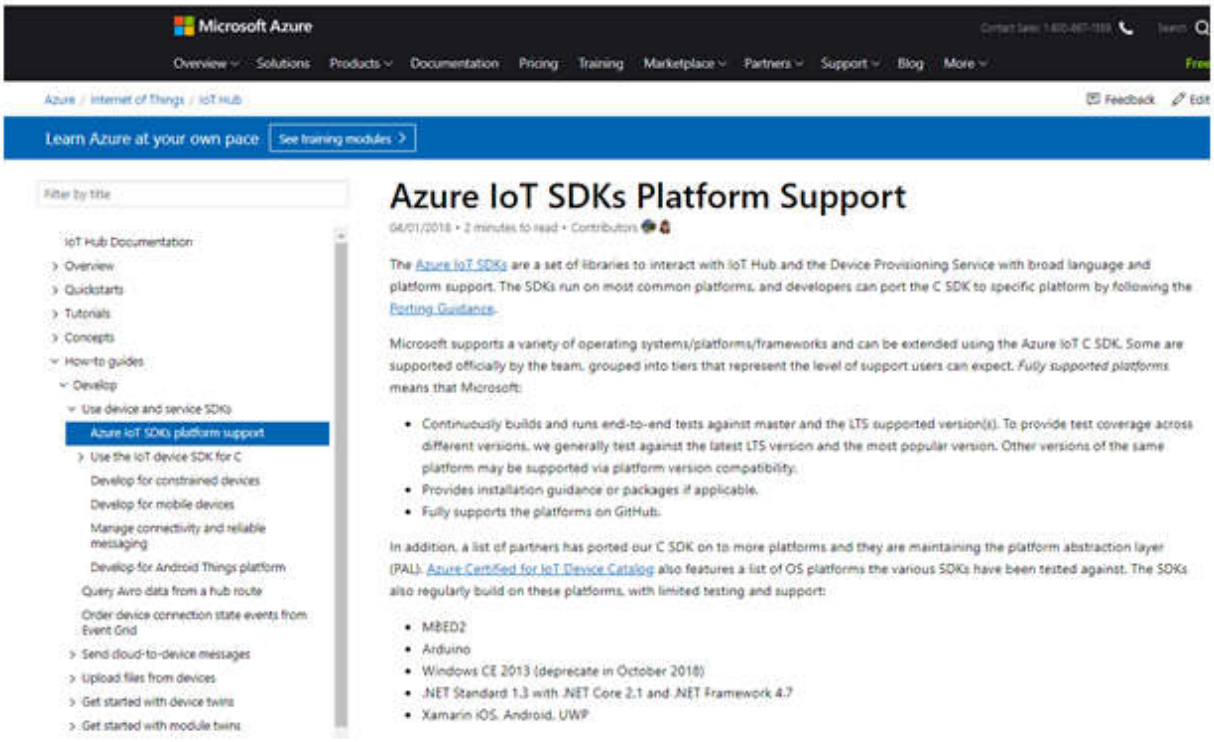
Azure IoT Hub device SDK for Java:

- Add to Maven project
- Source code
- API reference
- Module reference

Azure IoT Hub device SDK for Node.js:

- Install from npm
- Source code
- API reference
- Module reference

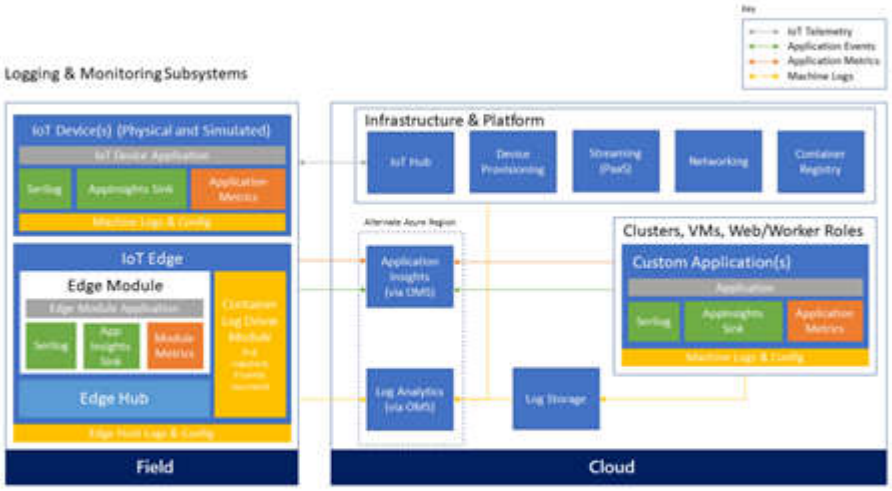
Source: https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-sdks



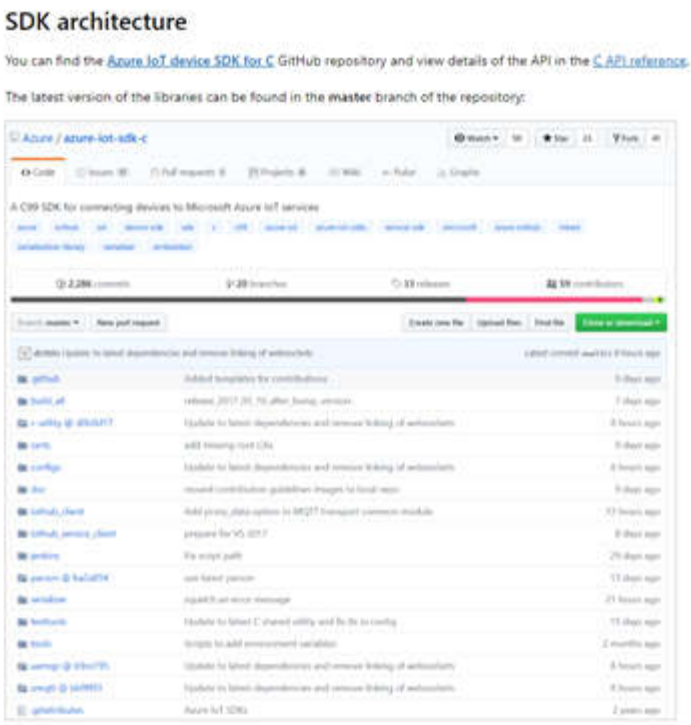
Source: https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-device-sdk-platform-support

Logging and Monitoring Architecture

The following simplified logging and monitoring architecture shows examples of typical IoT solution components and how they leverage the recommended technologies detailed above.

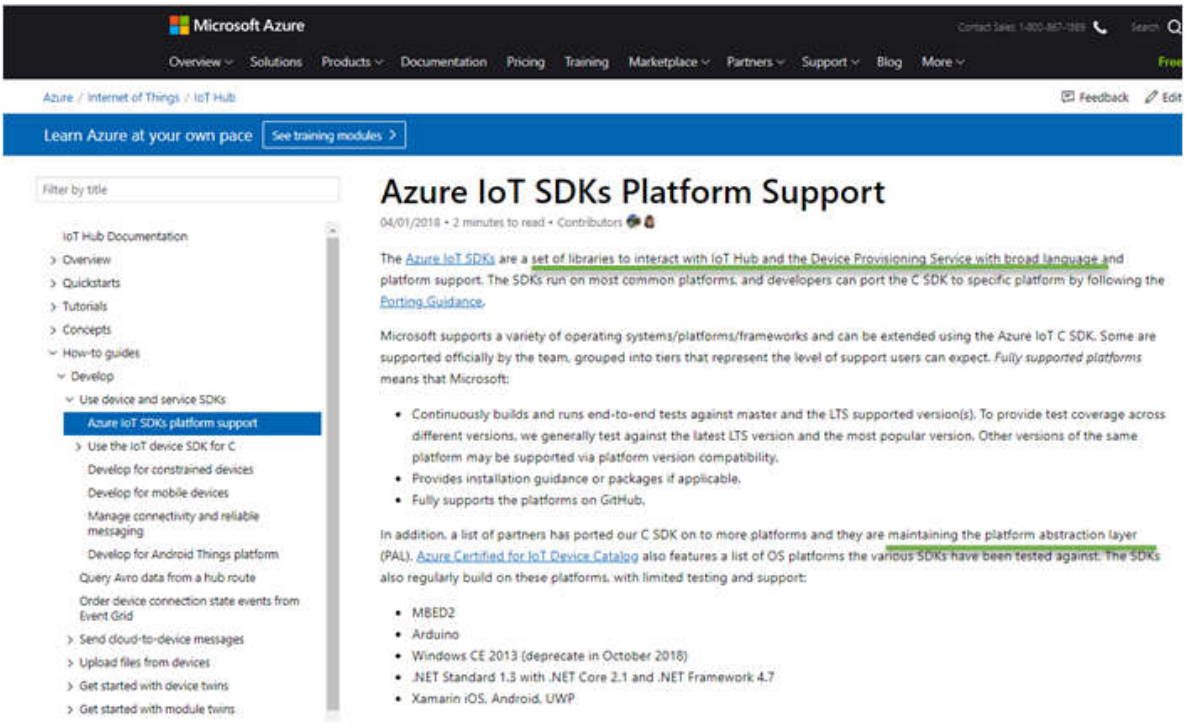


Source: http://download.microsoft.com/download/A/4/D/A4DAD253-BC21-41D3-B9D9-87D2AE6F0719/Microsoft_Azure_IoT_Reference_Architecture.pdf, page 42

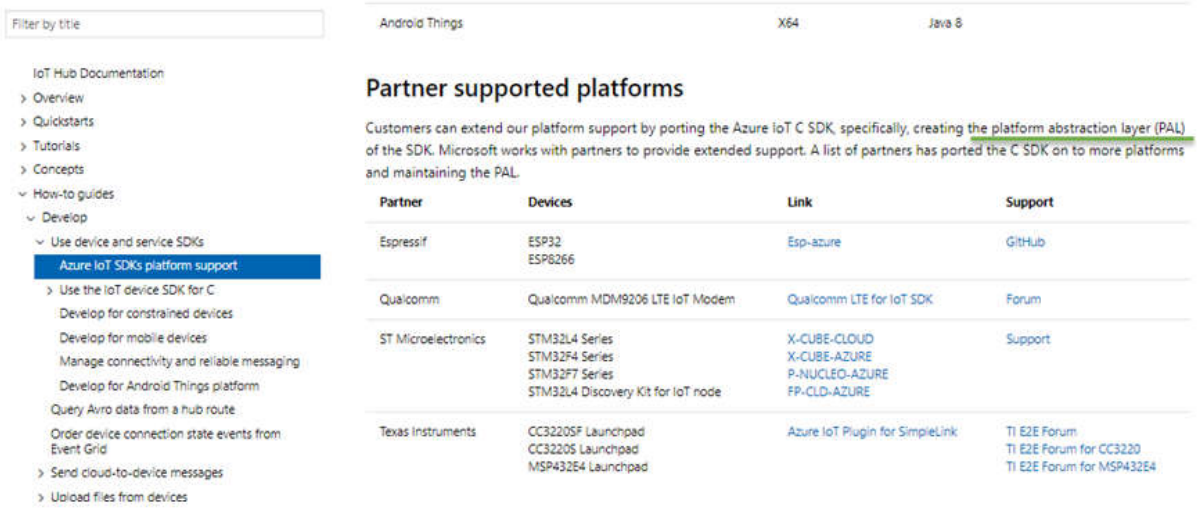


Source: https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-device-sdk-c-intro

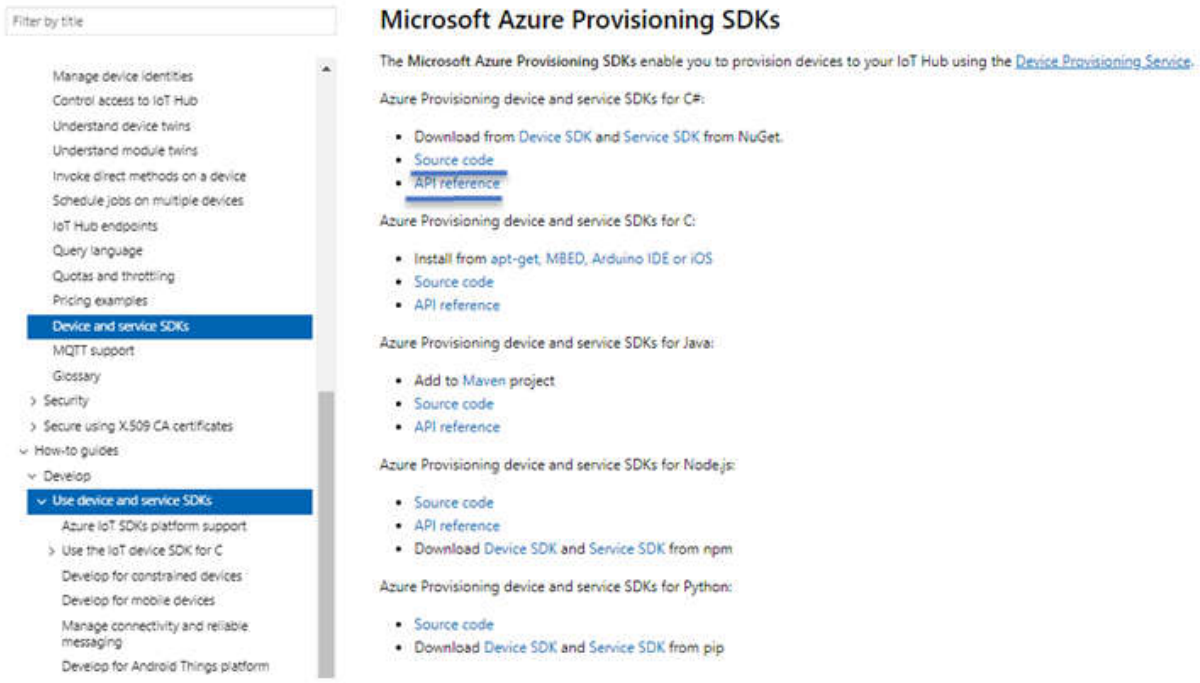
15. The Product practices providing one or more generic application handler programs. Each program has code for performing generic functions common to multiple hardware modules used in a communication system. For example, the Azure IoT Hub Software Development Kit includes generic application handler programs including drivers, libraries, and Abstraction Layers (such as Platform Abstraction Layer (PAL)) that provide multiple generic Application Programming Interfaces (APIs). The generic code provides common and generic functions to multiple hardware modules (such as IoT device) used in a communication system. Certain elements of this limitation are illustrated in the screenshots below and in the screenshots referenced in connection with other elements herein.



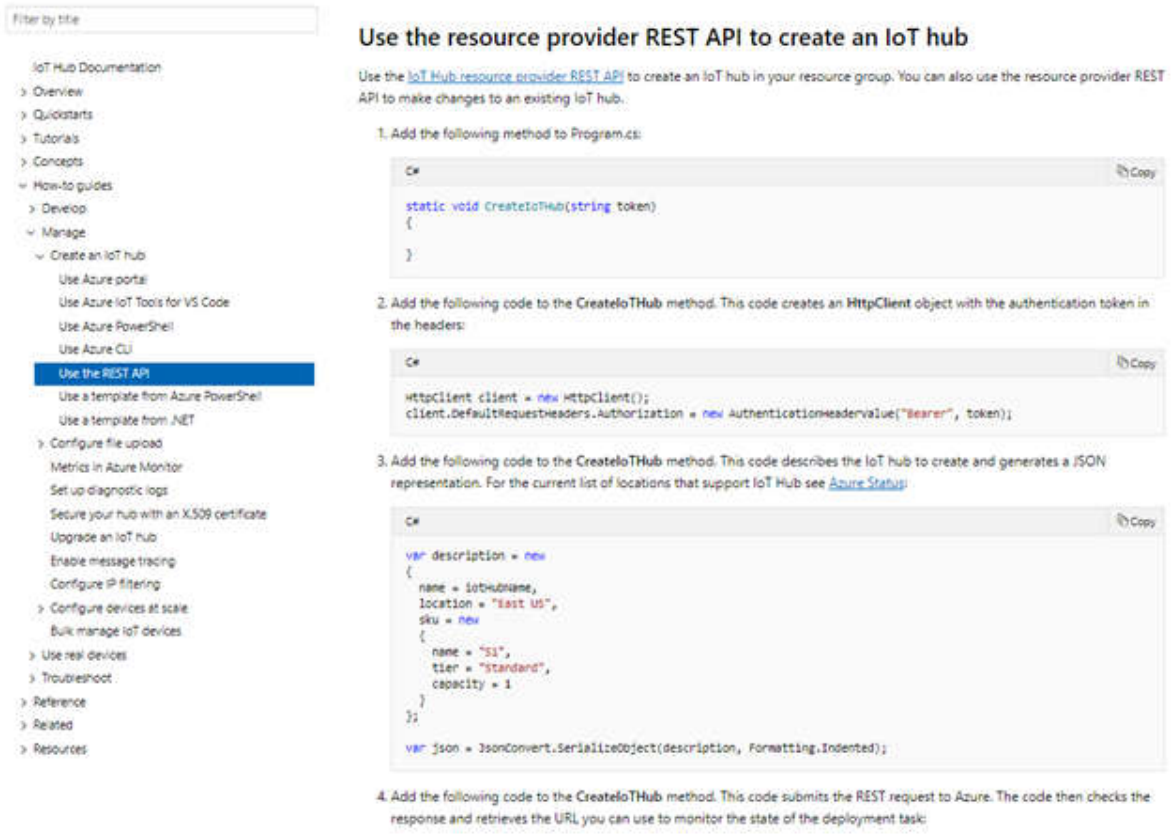
Source: https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-device-sdk-platform-support



Source: https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-device-sdk-platform-support



Source: https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-sdks



Filter by title

IoT Hub Documentation
- Overview
- Quickstarts
- Tutorials
- Concepts
- How-to guides
  - Develop
  - Manage
  - Create an IoT hub
    - Use Azure portal
    - Use Azure IoT Tools for VS Code
    - Use Azure PowerShell
    - Use Azure CLI
    - Use the REST API
    - Use a template from Azure PowerShell
    - Use a template from .NET
  - Configure file upload
  - Metrics in Azure Monitor
  - Set up diagnostic logs
  - Secure your hub with an X.509 certificate
  - Upgrade an IoT hub
  - Enable message tracing
  - Configure IP filtering
  - Configure devices at scale
  - Bulk manage IoT devices
- Use real devices
- Troubleshoot
- Reference
- Related
- Resources

Use the resource provider REST API to create an IoT hub

Use the IoT Hub resource provider REST API to create an IoT hub in your resource group. You can also use the resource provider REST API to make changes to an existing IoT hub.

1. Add the following method to Program.cs:

```
C#
static void CreateIoTHub(string token)
{

}
```

2. Add the following code to the **CreateIoTHub** method. This code creates an **HttpClient** object with the authentication token in the headers:

```
C#
HttpClient client = new HttpClient();
client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", token);
```

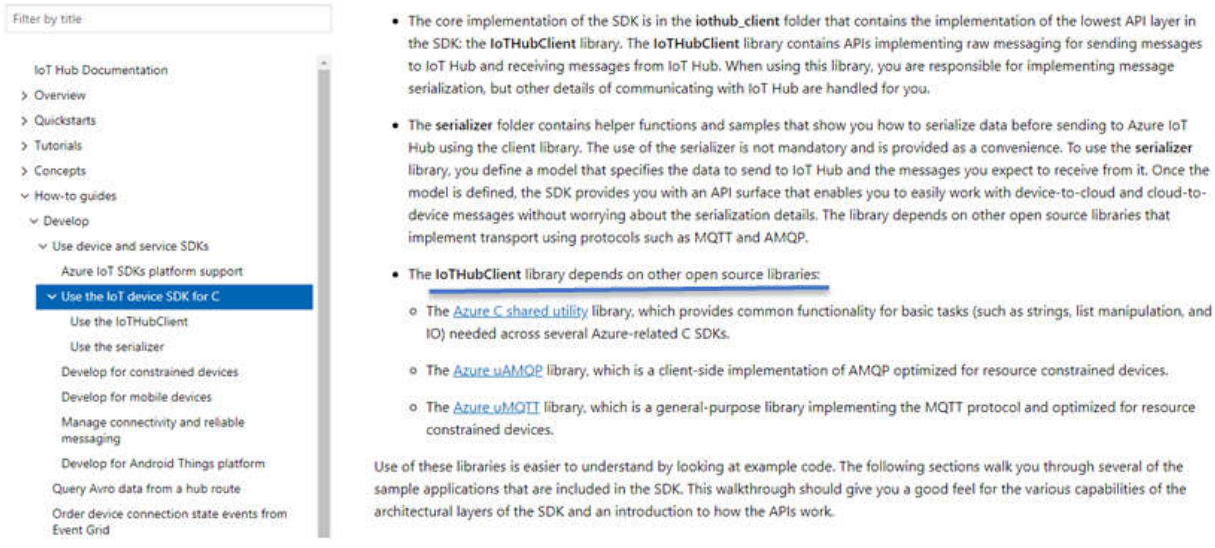
3. Add the following code to the **CreateIoTHub** method. This code describes the IoT hub to create and generates a JSON representation. For the current list of locations that support IoT Hub see Azure Status:

```
C#
var description = new
{
  name = iotHubName,
  location = "East US",
  sku = new
  {
    name = "S1",
    tier = "Standard",
    capacity = 1
  }
};

var json = JsonConvert.SerializeObject(description, Formatting.Indented);
```

4. Add the following code to the **CreateIoTHub** method. This code submits the REST request to Azure. The code then checks the response and retrieves the URL you can use to monitor the state of the deployment task:

Source: https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-rm-rest



Filter by title

IoT Hub Documentation
- Overview
- Quickstarts
- Tutorials
- Concepts
- How-to guides
  - Develop
    - Use device and service SDKs
      - Azure IoT SDKs platform support
      - Use the IoT device SDK for C
        - Use the IoTHubClient
        - Use the serializer
      - Develop for constrained devices
      - Develop for mobile devices
      - Manage connectivity and reliable messaging
      - Develop for Android Things platform
    - Query Avro data from a hub route
    - Order device connection state events from Event Grid

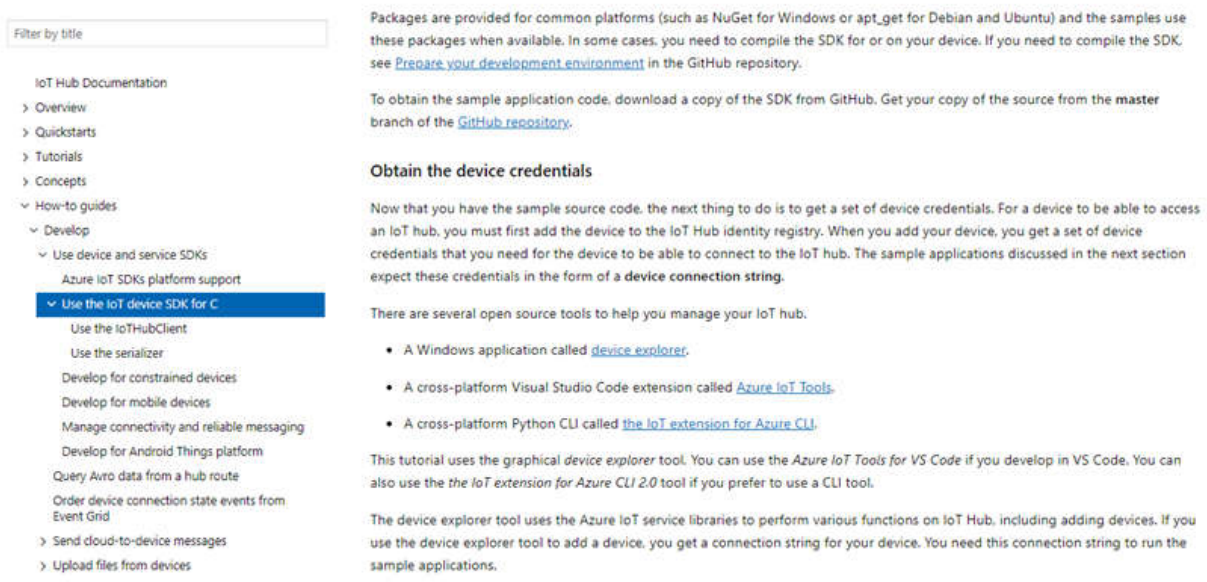
- The core implementation of the SDK is in the **iothub_client** folder that contains the implementation of the lowest API layer in the SDK: the **IoTHubClient** library. The **IoTHubClient** library contains APIs implementing raw messaging for sending messages to IoT Hub and receiving messages from IoT Hub. When using this library, you are responsible for implementing message serialization, but other details of communicating with IoT Hub are handled for you.
- The **serializer** folder contains helper functions and samples that show you how to serialize data before sending to Azure IoT Hub using the client library. The use of the serializer is not mandatory and is provided as a convenience. To use the **serializer** library, you define a model that specifies the data to send to IoT Hub and the messages you expect to receive from it. Once the model is defined, the SDK provides you with an API surface that enables you to easily work with device-to-cloud and cloud-to-device messages without worrying about the serialization details. The library depends on other open source libraries that implement transport using protocols such as MQTT and AMQP.
- The **IoTHubClient** library depends on other open source libraries:
  - The Azure C shared utility library, which provides common functionality for basic tasks (such as strings, list manipulation, and IO) needed across several Azure-related C SDKs.
  - The Azure uAMQP library, which is a client-side implementation of AMQP optimized for resource constrained devices.
  - The Azure uMQTT library, which is a general-purpose library implementing the MQTT protocol and optimized for resource constrained devices.

Use of these libraries is easier to understand by looking at example code. The following sections walk you through several of the sample applications that are included in the SDK. This walkthrough should give you a good feel for the various capabilities of the architectural layers of the SDK and an introduction to how the APIs work.

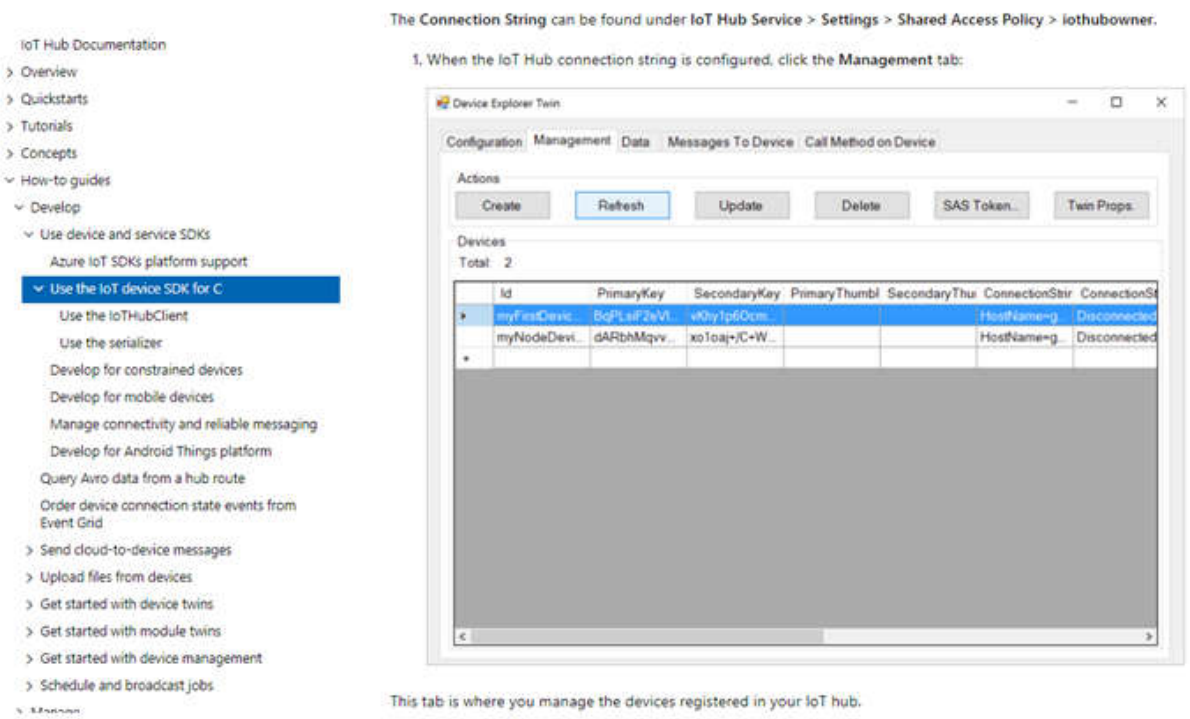
Source: https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-device-sdk-c-intro

16. The Product practices generating specific application handler code to associate

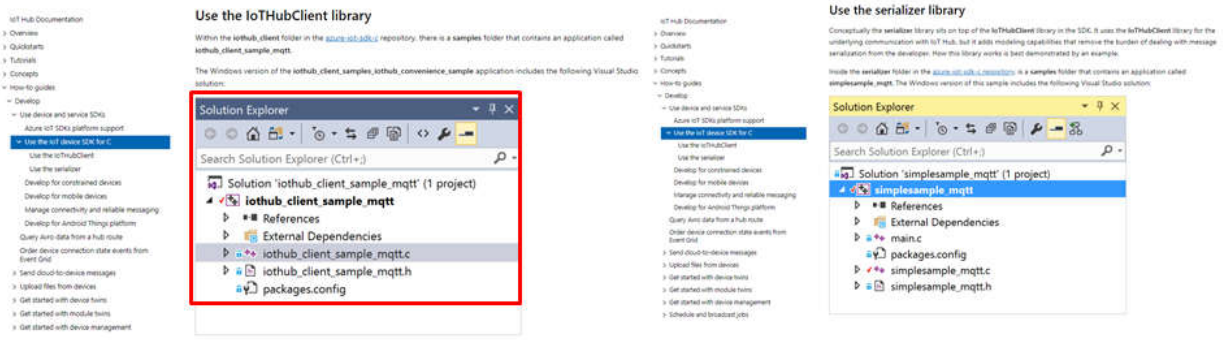
the generic application functions with specific functions of a device driver for at least one of the types of the hardware modules. For example, in addition to the Platform Abstraction Layer (PAL), Azure IoT Hub SDK also include processor-specific application handler code that are specific to particular IoT device running on different operating systems/platforms/frameworks (such as MBED2, Arduino, Windows CE, .Net Standard 1.3, and/or Xamarian iOS, Android and/or UWP). Certain elements of this limitation are illustrated in the screenshots below and in the screenshots referenced in connection with other elements herein.



Source: https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-device-sdk-c-intro



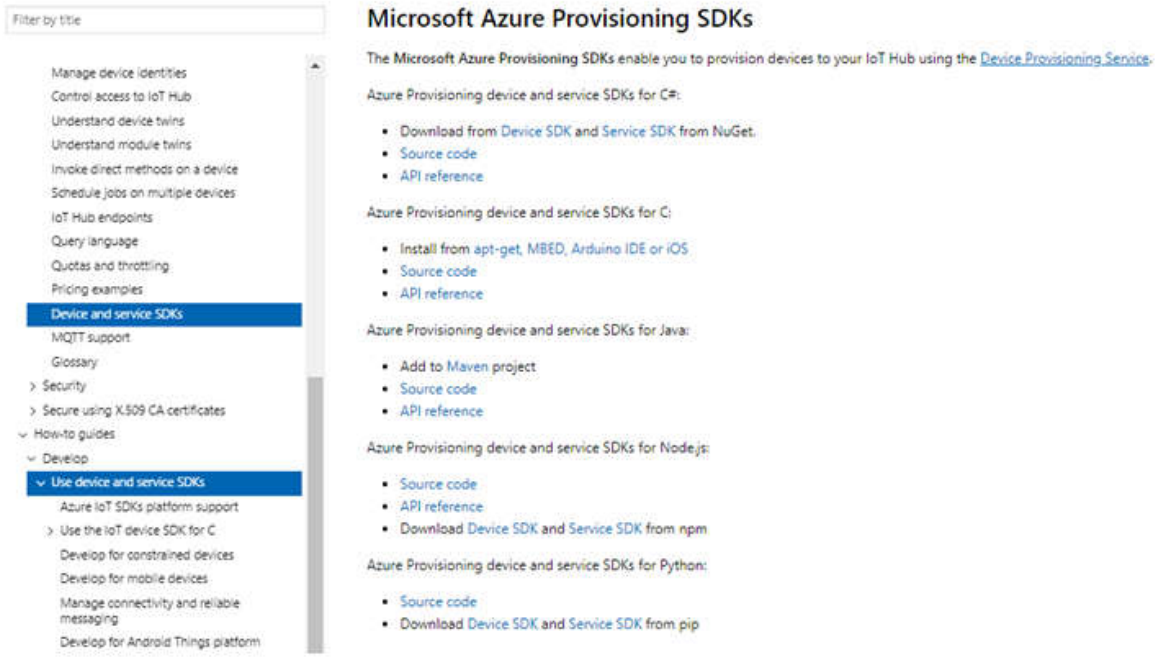
Source: https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-device-sdk-c-intro



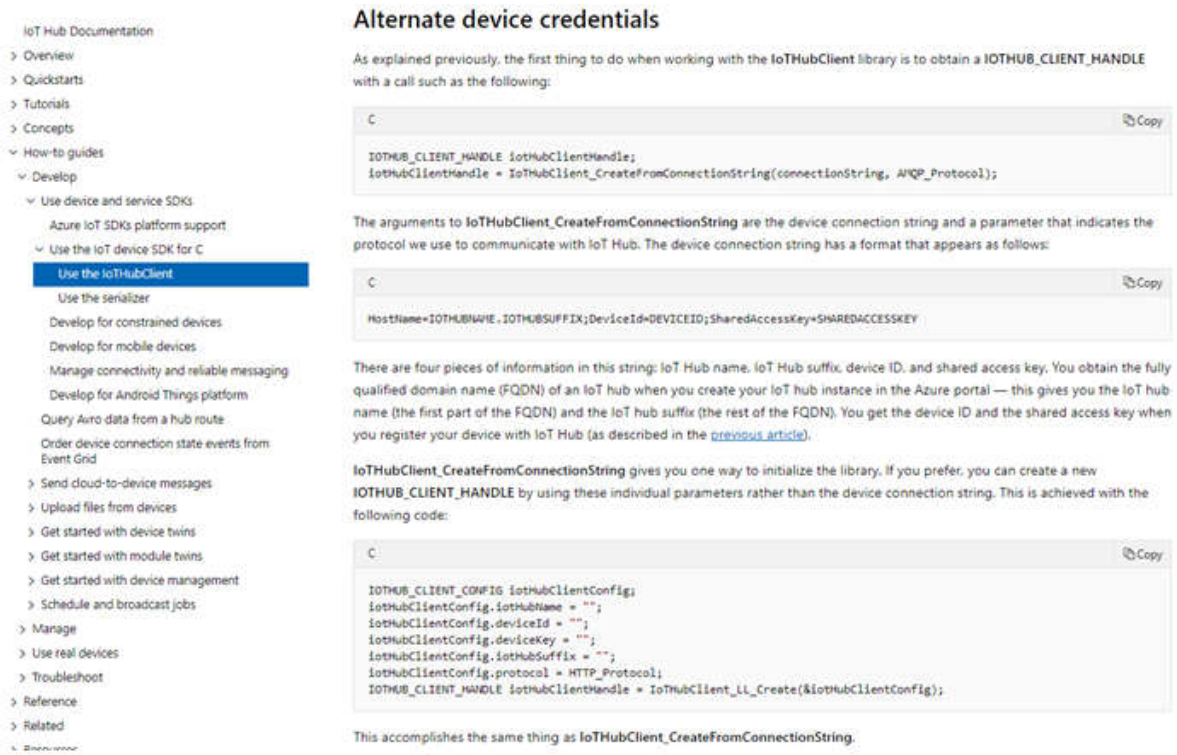
Source: https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-device-sdk-c-intro

17. The Product practices defining a specific element in the specific application handler code to be handled by one of the generic application functions for the at least one of the types of the hardware modules, and registering one of the specific functions of the device driver for use in handing the defined specific element. For example, the Azure IoT Hub SDK

generates system-specific application handler code by defining a specific element such as data structures and functions that are handled by one or more generic application functions in the Platform Abstraction Layer (PAL). Certain elements of this limitation are illustrated in the screenshots below and in the screenshots referenced in connection with other elements herein.



Source: https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-sdks



IoT Hub Documentation
- Overview
- Quickstarts
- Tutorials
- Concepts
- How-to guides
  - Develop
    - Use device and service SDKs
      - Azure IoT SDKs platform support
      - Use the IoT device SDK for C
        - Use the IoTHubClient
        - Use the serializer
      - Develop for constrained devices
      - Develop for mobile devices
      - Manage connectivity and reliable messaging
      - Develop for Android Things platform
    - Query Avro data from a hub route
    - Order device connection state events from Event Grid
    - Send cloud-to-device messages
    - Upload files from devices
    - Get started with device twins
    - Get started with module twins
    - Get started with device management
    - Schedule and broadcast jobs
  - Manage
  - Use real devices
  - Troubleshoot
- Reference
- Related

Alternate device credentials

As explained previously, the first thing to do when working with the **IoTHubClient** library is to obtain a **IOTHUB_CLIENT_HANDLE** with a call such as the following:

```
C
IOTHUB_CLIENT_HANDLE iotHubClientHandle;
iotHubClientHandle = IoTHubClient_CreateFromConnectionString(connectionString, AMQP_Protocol);
```

The arguments to **IoTHubClient_CreateFromConnectionString** are the device connection string and a parameter that indicates the protocol we use to communicate with IoT Hub. The device connection string has a format that appears as follows:

```
C
HostName=IOTHUBNAME.IOTHUBSUFFIX;DeviceId=DEVICEID;SharedAccessKey=SHAREDACCESSKEY
```

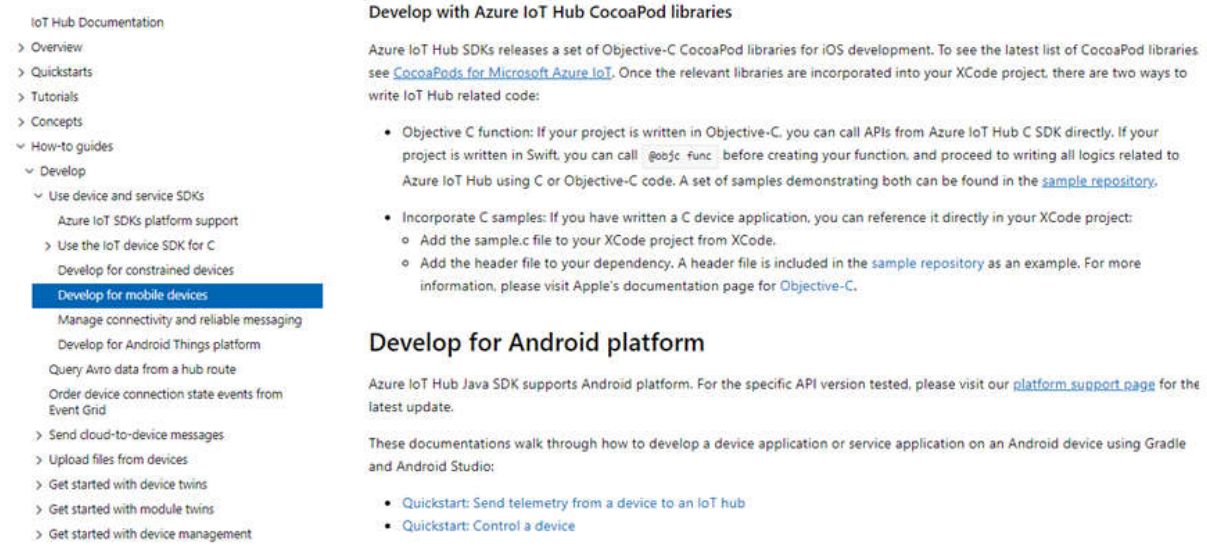
There are four pieces of information in this string: IoT Hub name, IoT Hub suffix, device ID, and shared access key. You obtain the fully qualified domain name (FQDN) of an IoT hub when you create your IoT hub instance in the Azure portal — this gives you the IoT hub name (the first part of the FQDN) and the IoT hub suffix (the rest of the FQDN). You get the device ID and the shared access key when you register your device with IoT Hub (as described in the previous article).

IoTHubClient_CreateFromConnectionString gives you one way to initialize the library. If you prefer, you can create a new **IOTHUB_CLIENT_HANDLE** by using these individual parameters rather than the device connection string. This is achieved with the following code:

```
C
IOTHUB_CLIENT_CONFIG iotHubClientConfig;
iotHubClientConfig.iotHubName = "";
iotHubClientConfig.deviceId = "";
iotHubClientConfig.deviceKey = "";
iotHubClientConfig.iotHubSuffix = "";
iotHubClientConfig.protocol = HTTP_Protocol;
IOTHUB_CLIENT_HANDLE iotHubClientHandle = IoTHubClient_LL_Create(&iotHubClientConfig);
```

This accomplishes the same thing as **IoTHubClient_CreateFromConnectionString**.

Source: https://www.arm.com/files/pdf/20160830_06_ARM_Keil_MDK_FTD.pdf, page 15



IoT Hub Documentation
- Overview
- Quickstarts
- Tutorials
- Concepts
- How-to guides
  - Develop
    - Use device and service SDKs
      - Azure IoT SDKs platform support
      - Use the IoT device SDK for C
      - Develop for constrained devices
      - Develop for mobile devices
      - Manage connectivity and reliable messaging
      - Develop for Android Things platform
    - Query Avro data from a hub route
    - Order device connection state events from Event Grid
    - Send cloud-to-device messages
    - Upload files from devices
    - Get started with device twins
    - Get started with module twins
    - Get started with device management

Develop with Azure IoT Hub CocoaPod libraries

Azure IoT Hub SDKs releases a set of Objective-C CocoaPod libraries for iOS development. To see the latest list of CocoaPod libraries see CocoaPods for Microsoft Azure IoT. Once the relevant libraries are incorporated into your XCode project, there are two ways to write IoT Hub related code:

- Objective C function: If your project is written in Objective-C, you can call APIs from Azure IoT Hub C SDK directly. If your project is written in Swift, you can call `@objc func` before creating your function, and proceed to writing all logics related to Azure IoT Hub using C or Objective-C code. A set of samples demonstrating both can be found in the sample repository.
- Incorporate C samples: If you have written a C device application, you can reference it directly in your XCode project:
  - Add the sample.c file to your XCode project from XCode.
  - Add the header file to your dependency. A header file is included in the sample repository as an example. For more information, please visit Apple's documentation page for Objective-C.

Develop for Android platform

Azure IoT Hub Java SDK supports Android platform. For the specific API version tested, please visit our platform support page for the latest update.

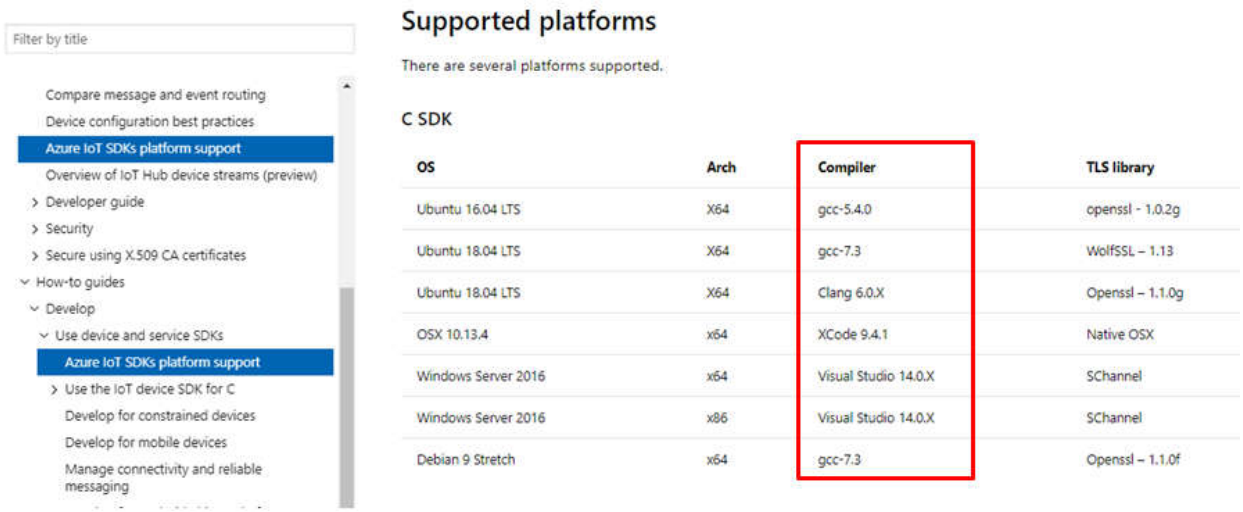
These documentations walk through how to develop a device application or service application on an Android device using Gradle and Android Studio:

- Quickstart: Send telemetry from a device to an IoT hub
- Quickstart: Control a device

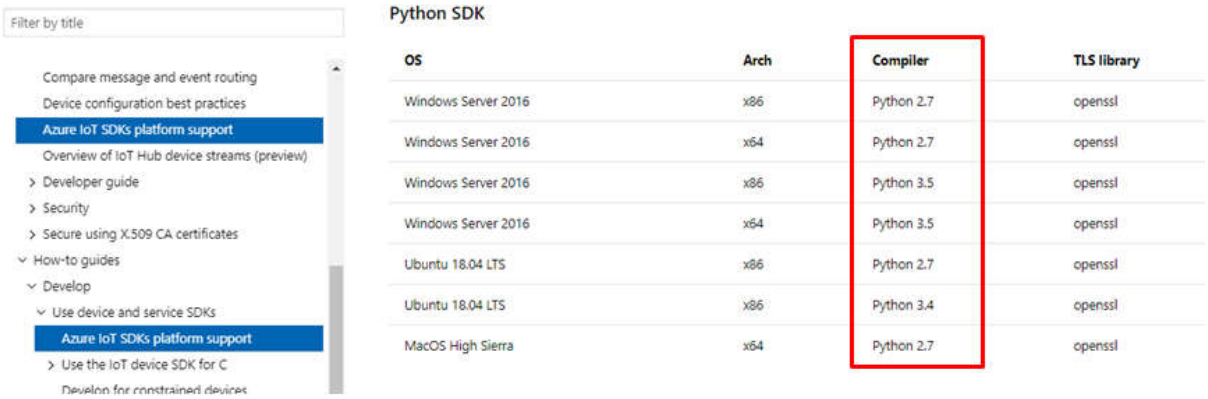
Sourcehttps://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-how-to-develop-for-mobile-devices

18. The Product practices compiling the generic application handler programs together with the specific application handler code to produce machine-readable code to be executed by an embedded processor in the at least one of the types of the hardware modules.

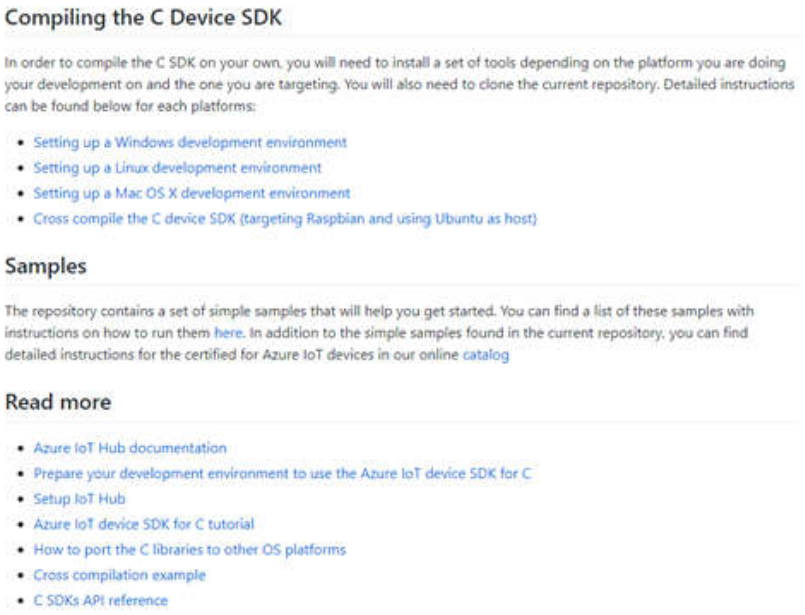
For example, when a specific application is needed for a particular hardware, the generic functions and the specific functions are compiled together to yield a machine readable code. Microsoft and/or its customers compile the generic functions and the specific functions using Azure IoT Hub SDK or any other IDE/compiler (such as GCC, Clang, XCode, Visual Studio and/or Python) supported by Microsoft. Certain elements of this limitation are illustrated in the screenshots below and in the screenshots referenced in connection with other elements herein.



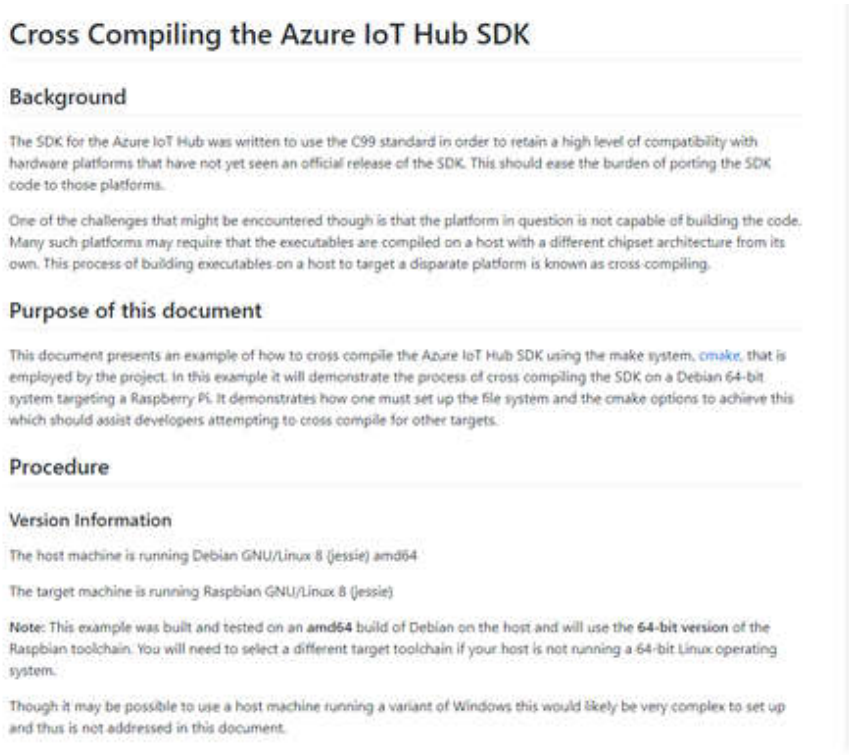
Source: https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-device-sdk-platform-support



Source: https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-device-sdk-platform-support



Source: https://github.com/Azure/azure-iot-sdk-c/blob/master/iothub_client/readme.md#compiling-the-c-device-sdk



Cross Compiling the Azure IoT Hub SDK

Background

The SDK for the Azure IoT Hub was written to use the C99 standard in order to retain a high level of compatibility with hardware platforms that have not yet seen an official release of the SDK. This should ease the burden of porting the SDK code to those platforms.

One of the challenges that might be encountered though is that the platform in question is not capable of building the code. Many such platforms may require that the executables are compiled on a host with a different chipset architecture from its own. This process of building executables on a host to target a disparate platform is known as cross compiling.

Purpose of this document

This document presents an example of how to cross compile the Azure IoT Hub SDK using the make system, cmake, that is employed by the project. In this example it will demonstrate the process of cross compiling the SDK on a Debian 64-bit system targeting a Raspberry Pi. It demonstrates how one must set up the file system and the cmake options to achieve this which should assist developers attempting to cross compile for other targets.

Procedure

Version Information

The host machine is running Debian GNU/Linux 8 (jessie) amd64

The target machine is running Raspbian GNU/Linux 8 (jessie)

Note: This example was built and tested on an **amd64** build of Debian on the host and will use the **64-bit version** of the Raspbian toolchain. You will need to select a different target toolchain if your host is not running a 64-bit Linux operating system.

Though it may be possible to use a host machine running a variant of Windows this would likely be very complex to set up and thus is not addressed in this document.

Source: https://github.com/Azure/azure-iot-sdk-c/blob/master/doc/SDK_cross_compile_example.md

19. Defendant's actions complained of herein will continue unless Defendant is enjoined by this court.

20. Defendant's actions complained of herein are causing irreparable harm and monetary damage to Plaintiff and will continue to do so unless and until Defendant is enjoined and restrained by this Court.

21. Plaintiff is in compliance with 35 U.S.C. § 287.

PRAYER FOR RELIEF

WHEREFORE, Plaintiff asks the Court to:

(a) Enter judgment for Plaintiff on this Complaint on all causes of action asserted herein;

(b) Enter an Order enjoining Defendant, its agents, officers, servants, employees,

attorneys, and all persons in active concert or participation with Defendant who receive notice of the order from further infringement of United States Patent No. 7,069,546 (or, in the alternative, awarding Plaintiff a running royalty from the time of judgment going forward);

(c) Award Plaintiff damages resulting from Defendant's infringement in accordance with 35 U.S.C. § 284;

(d) Award Plaintiff pre-judgment and post-judgment interest and costs; and

(e) Award Plaintiff such further relief to which the Court finds Plaintiff entitled under law or equity.

Dated: June 27, 2019

Respectfully submitted,

/s/ Jay Johnson

JAY JOHNSON
State Bar No. 24067322
D. BRADLEY KIZZIA
State Bar No. 11547550
KIZZIA JOHNSON, PLLC
1910 Pacific Ave., Suite 13000
Dallas, Texas 75201
(214) 451-0164
Fax: (214) 451-0165
jay@kjpllc.com
bkizzia@kjpllc.com

ATTORNEYS FOR PLAINTIFF

EXHIBIT A