Contents

Introduction

1. Create Azure account, use following website to subscribe to Azure with http://microsoftazurepass.com
   or https://azure.microsoft.com/en-us/free/students/ or https://azure.microsoft.com/en-us/free/

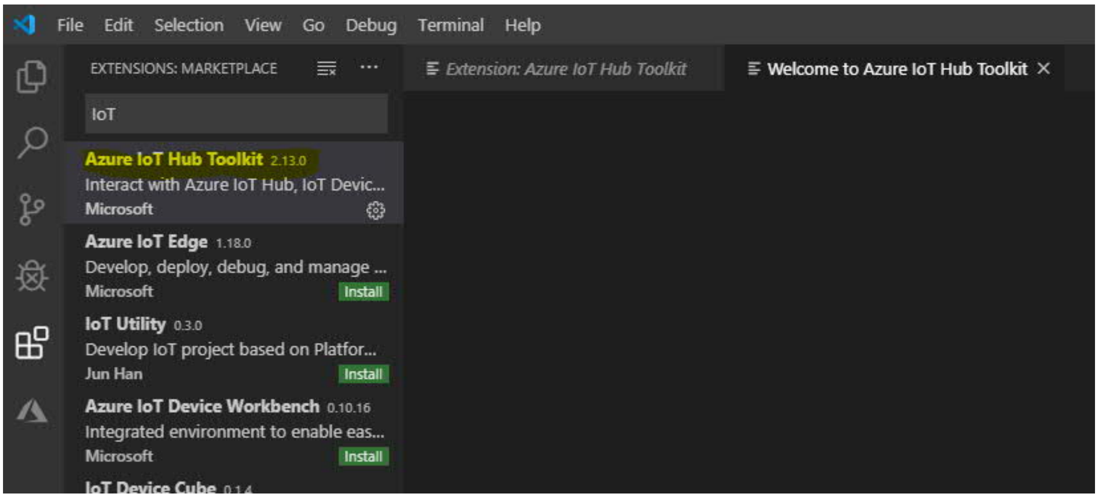
2. Head over to https://shell.azure.com and sign in with your Azure Subscription details,

3. Install Azure CLI (https://docs.microsoft.com/pl-pl/cli/azure/install-azure-cli?view=azure-cli-latest),

4. Install Azure IoT CLI extension (https://github.com/Azure/azure-iot-cli-extension)
   ```
   az extension add --name azure-cli-iot-ext
   ```

5. Install Visual Studio Code (https://code.visualstudio.com/),

6. Run Visual Studio Code and install Azure IoT Toolkit:


File Edit Selection View Go Debug Terminal Help
EXTENSIONS: MARKETPLACE
Extension: Azure IoT Hub Toolkit
Welcome to Azure IoT Hub Toolkit
IoT
Azure IoT Hub Toolkit 2.13.0
Interact with Azure IoT Hub, IoT Devic...
Microsoft
Azure IoT Edge 1.18.0
Develop, deploy, debug, and manage ...
Microsoft
Install
IoT Utility 0.3.0
Develop IoT project based on Platfor...
Jun Han
Install
Azure IoT Device Workbench 0.10.16
Integrated environment to enable eas...
Microsoft
Install
IoT Device Cube 0.1.4

Access to IoT Hub from Azure CLI and Visual Studio Code

Creating IoT Hub

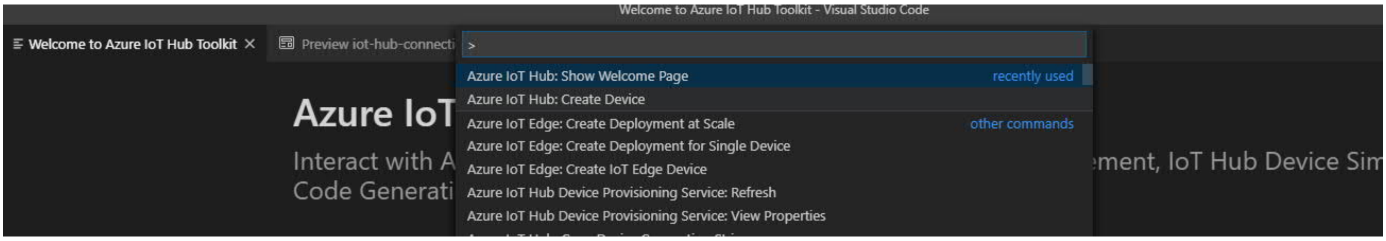
1. Log in to Azure Portal https://portal.azure.com/,

2. Create Resource Group for IoT Hub related resources (more on resource groups: https://docs.microsoft.com/pl-pl/azure/azure-resource-manager/resource-group-overview),

3. Create new Azure IoT Hub and assign it to the resource group created in the previous step, select F1: Free tier in the “Pricing and scale tier” (more on scaling options for IoT Hub: https://docs.microsoft.com/pl-pl/azure/iot-hub/iot-hub-scaling)

Creating IoT Device using Visual Studio Code Extension

1. Open Visual Studio Code and run Azure IoT Hub: Show Welcome Page command:

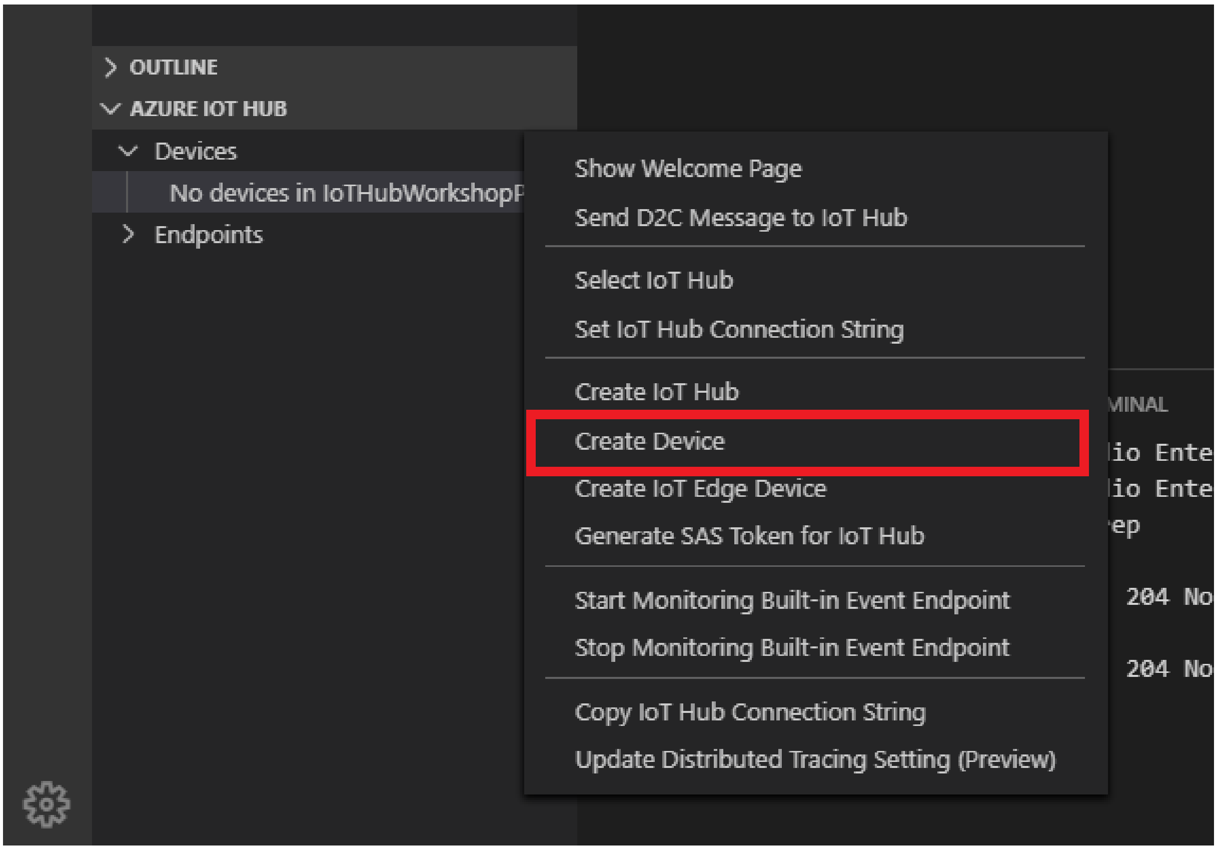

Welcome to Azure IoT Hub Toolkit - Visual Studio Code
Welcome to Azure IoT Hub Toolkit
Preview iot-hub-connecti
>
Azure IoT Hub: Show Welcome Page
recently used
Azure IoT Hub: Create Device
Azure IoT Edge: Create Deployment at Scale
other commands
Azure IoT Edge: Create Deployment for Single Device
Azure IoT Edge: Create IoT Edge Device
Azure IoT Hub Device Provisioning Service: Refresh
Azure IoT Hub Device Provisioning Service: View Properties
Azure IoT
Interact with A
Code Generati
ment, IoT Hub Device Sim

Note: To access Commands Palette go to View->Command Palette... or press Ctrl+Shift+P key combination.

2. Press "Select IoT Hub" (log in to Azure), select subscription and IoT Hub created earlier,

3. Expand Azure IoT Hub panel to see devices created in our IoT Hub (there should be none in this step),

4. Create IoT Device by using menu option from context menu available in the Azure IoT hub panel (provide name of the device):


OUTLINE
AZURE IOT HUB
Devices
No devices in IoTHubWorkshopP
Endpoints
Show Welcome Page
Send D2C Message to IoT Hub
Select IoT Hub
Set IoT Hub Connection String
Create IoT Hub
Create Device
Create IoT Edge Device
Generate SAS Token for IoT Hub
Start Monitoring Built-in Event Endpoint
Stop Monitoring Built-in Event Endpoint
Copy IoT Hub Connection String
Update Distributed Tracing Setting (Preview)
MINAL
lio Ente
lio Ente
ep
204 No
204 No

Creating Azure IoT Device using Azure CLI

1. Run command line and log in to Azure using following command: az login

2. In order to create new device it's required to use following command:

   ```
   az iot hub device-identity create -d %NAME_OF_THE_DEVICE% -n %IOT_HUB_NAME%
   e.g. az iot hub device-identity create -d test-device-02 -n IoTHubForDevices
   ```

Sending device to cloud messages (without real device) using VS code and Azure CLI

1. (VS Code) From context menu for a device in Azure IoT Hub Panel select "Start monitoring Built-in Event Endpoint",

2. (VS Code) From context menu for a device in Azure IoT Hub Panel select "Send D2C Message to IoT Hub",

3. (VS Code) Provide information on number of messages, interval between messages and message, press Send,

4. (Azure CLI) Run command line and log in to Azure using following command: az login (if not done previously)

5. (Azure CLI) Following command allow to send message to the IoT Hub:

```
az iot device send-d2c-message -n %IOT_HUB_NAME% -d %NAME_OF _DEVICE%  --data "Hello from CLI"
```

6. (Azure CLI) To monitor messages that are coming to the IoT Hub the following command can be used (from another command window):

   ```
   az iot hub monitor-events -n %IOT_HUB_NAME%
   ```

Sending cloud to device messages (without real device) using VS code and Azure CLI

1. (VS Code) From context menu for a device in Azure IoT Hub Panel select "Start Receiving C2D Messages",

2. (VS Code) From context menu for a device in Azure IoT Hub Panel select "Send C2D Message to Device",

3. (VS Code) Provide message and confirm, message to device should appear in the output panel,

4. (Azure CLI) Run command line and log in to Azure using following command: az login (if not done previously)

5. (Azure CLI) Following command allow to send message to device:

   ```
   az iot device c2d-message send -n %IOT_HUB_NAME% -d %NAME_OF _DEVICE% --data
   "Hello from Azure CLI" (Azure CLI)
   ```
6. To monitor messages that are coming to the IoT Hub the following command can be used (from another command window):

   ```
       az iot device c2d-message receive -n %IOT_HUB_NAME% -d %NAME_OF _DEVICE%
   ```

Simulating a Device (can listen to incoming messages)

1. (Azure CLI) az iot device simulate -d %NAME_OF _DEVICE% -n %IOT_HUB_NAME% --msg-interval 10 --data "Hello from device",

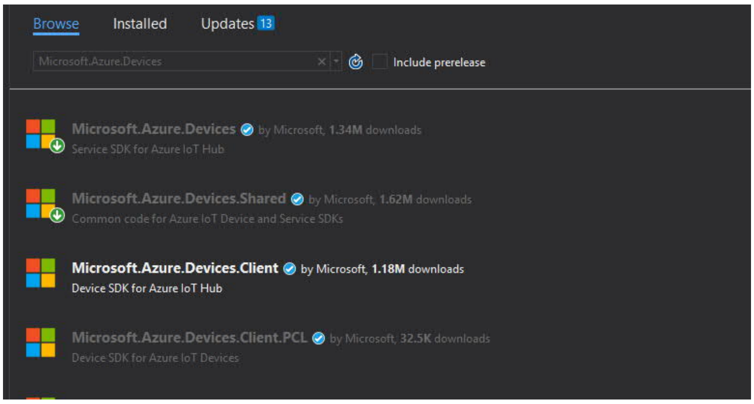
2. (VS Code) Send C2D message and see what’s happening on the console when above command is still running,

Building sample .NET application to interact with Azure IoT Hub

Creating .Net Core device client project in Visual Studio

1. Create Console App (.NET Core) project in Visual Studio,

2. Add Nuget package Microsoft.Azure.Devices.Client,



Microsoft.Azure.Devices by Microsoft, 1.34M downloads
Service SDK for Azure IoT Hub

Microsoft.Azure.Devices.Shared by Microsoft, 1.62M downloads
Common code for Azure IoT Device and Service SDKs

Microsoft.Azure.Devices.Client by Microsoft, 1.18M downloads
Device SDK for Azure IoT Hub

Microsoft.Azure.Devices.Client.PCL by Microsoft, 32.5K downloads
Device SDK for Azure IoT Devices

Creating .Net Core device client project in Visual Studio Code

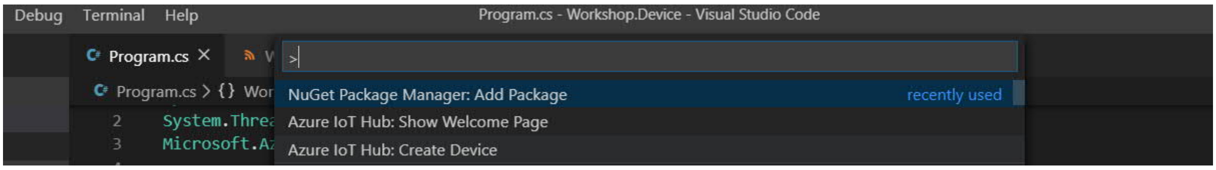
1. Install .NET Core SDK (https://dotnet.microsoft.com/download),

2. Add NuGet Package Manager and C# extensions,

3. Switch to terminal, create folder for device C# project, once in the folder run following command to get project created: dotnet new console,

4. Open .csproj file in the editor,

5. Run NuGet Package Manager: Add Package command and search for Microsoft.Azure.Devices.Client package,

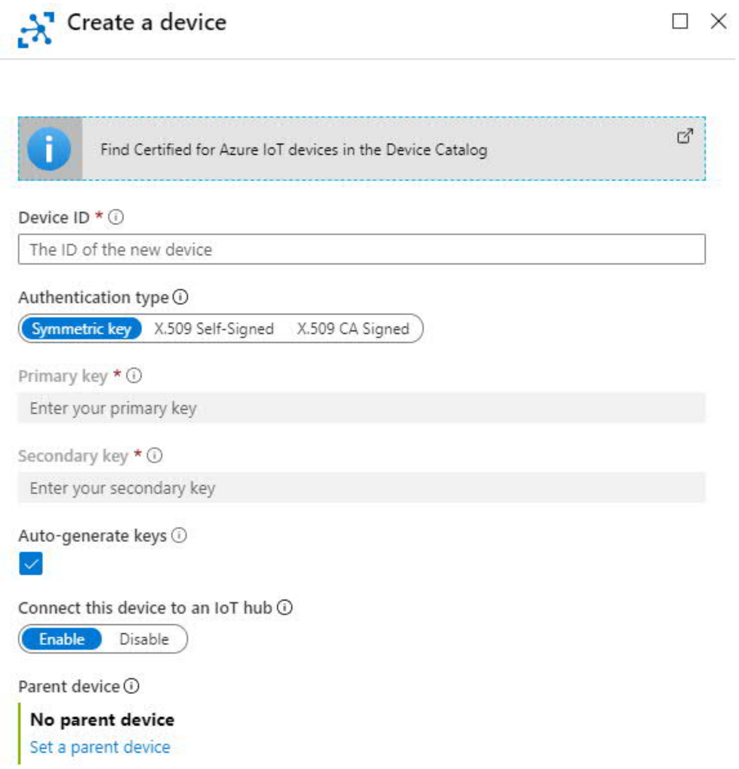

Debug
Terminal
Help
Program.cs - Workshop.Device - Visual Studio Code
Program.cs
Program.cs > {} Wor
>
NuGet Package Manager: Add Package
recently used
Azure IoT Hub: Show Welcome Page
Azure IoT Hub: Create Device

```
System.Threa
Microsoft.Az
```

Sending device to cloud messages

1. Switch Main method to be async,
   `static async Task Main(string[] args)`

2. Create IoT Device (using Azure Portal or Azure CLI or VS Code Azure IoT Hub Toolkit), set authentication type to “Symmetric key”



Create a device

Find Certified for Azure IoT devices in the Device Catalog

Device ID *

The ID of the new device

Authentication type

[✓ Symmetric key] X.509 Self-Signed X.509 CA Signed

Primary key *

Enter your primary key

Secondary key *

Enter your secondary key

Auto-generate keys

☑

Connect this device to an IoT hub

[✓ Enable] Disable

Parent device

No parent device

Set a parent device

3. Copy Primary connection string to device once it’s created,

4. Connection string is required to connect device to IoT Hub, following commands need to be executed to instantiate device in C# code:

```
var device = DeviceClient.CreateFromConnectionString(DeviceConnectionstring);
await device.OpenAsync();
```

5. Microsoft.Azure.Devices.Client API exposes Message class which can be used to wrap the data to be send to cloud:

```
 var message = new Message(Encoding.ASCII.GetBytes("Hello from device!"));
```

6. SendEventAsync method allows to send message to the cloud:

```
await device.SendEventAsync(message);
```

7. Create complex object to transfer device data:

```
Public class DeviceData
{
    public string Message {get;set;}
    public int StatusCode {get;set;}
}
```

8. Object can be serialized using JsonConvert.SerizalizeObject method:

```
var data = new DeviceData()
{
    Message = "Hello device",
    StatusCode = count
};

var dataJson = JsonConvert.SerializeObject(data);
```

Device Twins -> changing reported properties (https://docs.microsoft.com/pl-pl/azure/iot-hub/iot-hub-devguide-device-twins)

1. Add namespace to Program.cs: using Microsoft.Azure.Devices.Shared;

2. There is a calss called TwinCollection exposed through Microsoft.Azure.Devices.Client API which allows to update Reported Properties for the device.

```
var twinProperties = new TwinCollection();
twinProperties["connectionType"] = "wi-fi";
twinProperties["connectionStrength"] = "weak";
```

3. By using UpdateReportedPropertiesAsync method on DeviceClient object we can update device reported properties.

```
await device.UpdateReportedPropertiesAsync(twinProperties);
```

4. Following Azure CLI command allows to see device twins:

   az iot hub device-twin show -n %IOT_HUB_NAME% -d %NAME_OF _DEVICE%

Message processing

1. Create new .NET Core console application (instruction how to achieve that is available in previous steps),

2. Change Main method to be async,

3. Add Microsoft.Azure.EventHubs.Processor NuGet package to the project,

4. Following properties are required to create EventProcessorHost object:
   - Event hub name that is connected to IoT Hub,
   - Event hub connection string,
   - Storage container name where event processor can store it's data,
   - Storage account connection string,
   - Consumer group name which event processor should get assigned to, in this case it should stay as a default consumer group, the name of default consumer group can be retrieved from PartitionReceiver.DefaultConsumerGroupName property,

```
var eventHubName = "";
var iotEventHubConnectionString = "";
var storageConnectionString = "";
var storageContainerName = "";
var consumerGroupName = PartitionReceiver.DefaultConsumerGroupName;
```

5. Create instance of EventProcessorHost object which will host event processor,

```
var processor = new EventProcessorHost(
               eventHubName, consumerGroupName,iotEventHubConnectionString,
   storageConnectionString, storageContainerName);
```

6. Implement event processor class, it needs to implement IEventProcessor interface:

```
public class LoggingEventProcessor: IEventProcessor
```

7. Provide implementation for LoggingEventProcessor methods:

```
public Task CloseAsync(PartitionContext context, CloseReason reason)
{
    Console.WriteLine($"Logging event processor closing, partition {context.Pa
rtitionId}, reason: {reason}");
    return Task.CompletedTask;
}

public Task OpenAsync(PartitionContext context)
{
    Console.WriteLine($"Logging event processor opened, partition {context.Par
titionId}");
    return Task.CompletedTask;
}

public Task ProcessErrorAsync(PartitionContext context, Exception error)
{
    Console.WriteLine($"Logging event processor error, partition {context.Part
itionId}, error: {error.Message}");
    return Task.CompletedTask;
}

public Task ProcessEventsAsync(PartitionContext context, IEnumerable<EventData
> messages)
{

    return Task.CompletedTask;
}
```

8. Change implementation of ProcessEventAsync method to make it visualize more details of the messages that are being processed:

```
public Task ProcessEventsAsync(PartitionContext context, IEnumerable<EventData>
messages)
    {
      Console.WriteLine($"batch of events received on partition '{context.PartitionId}'");

      foreach (var message in messages)
      {
        var payload = Encoding.ASCII.GetString(message.Body.Array, message.Body.Offset,
message.Body.Count);

        var deviceId = message.SystemProperties["iothub-connection-device-id"];

        Console.WriteLine($"Message received on partition '{context.PartitionId}', " +
        $"device Id: {deviceId}, payload '{payload}'");
      }
      return context.CheckpointAsync();
    }
```

9. Register newly created even processor and make sure it gets unregistered before application closes:

```
await processor.RegisterEventProcessorAsync<LoggingEventProcessor>();

Console.WriteLine("Event processor started, press enter to exit...");
Console.ReadLine();

await processor.UnregisterEventProcessorAsync();
```

Receiving cloud to device messages

1. Switch back to the Device console project created in the first step,

2. Add new async method that will process messages from the cloud, method needs to get DeviceClient as an input e.g.:

```
static async Task ReceiveEvents(DeviceClient device)
```

3. ReceiveAsync method allows to query cloud for the new messages:

```
var message = await device.ReceiveAsync();
```

4. IoT Hub expects to either confirmation if message is accepted, rejected or abandoned, CompleteAsync method means the message is accepted:

```
while (true)
{
    var message = await device.ReceiveAsync();

```

```
     if(message == null)
     {
         continue;
     }

    var messageBody = message.GetBytes();

    var payload = Encoding.ASCII.GetString(messageBody);

    Console.WriteLine($"Received message from cloud: '{payload}'");

    await device.CompleteAsync(message);
}
```

Send Cloud to Device messages from .NET

1. Create new .Net Core console project in (VS Code or Visual Studio),

2. Change Main method to be async,

3. Add NuGet package Microsoft.Azure.Devices to the created project,

4. Add namespace in Program.cs: using Microsoft.Azure.Devices;

5. Communication to the devices requires object of class ServiceClient to be instantiated by executing static method CreateFromConnectionString,

6. Service connection string is available in the Hub configuration, you can find it in the “Shared access policies” section,

7. Execute following code to get service client object created (please replace service connection string with the connection string corresponding to your Hub):

```
var serviceConnectionString = ""
var service = ServiceClient.CreateFromConnectionString(serviceConnectionString);
```

8. Async method SendAsync on ServiceClient object allows to send message to the device, it accepts Device Id and message parameters e.g.

```
Console.WriteLine("Provide a message to the device '{deviceId}':");
var body = Console.ReadLine();
var payload = Encoding.ASCII.GetBytes(body);
var message = new Message(payload);
await service.SendAsync(deviceId, message);
```

Message feedback

1. To get information what happened with the message sent to a device we can use FeedbackReceiver object that is accessible from ServiceClient.

```
var feedbackReceiver = service.GetFeedbackReceiver();
```

2. Application needs to query Hub for new feedbacks by constantly asking using ReceiveAsync() method on FeedbackReceiver object:

```
var feedbackBatch = await feedbackReceiver.ReceiveAsync();
```

3. If there are no feedbacks available Hub returns null object, if feedbacks are available Hub returns they through Records property on FeedbackBatch object (result of feedbackReceiver.ReceiveAsync()) method:

```
foreach (var record in feedbackBatch.Records)
{
   var messageId = record.OriginalMessageId;
   var statusCode = record.StatusCode;
   Console.WriteLine($"Feedback for message {messageId}, status code: {statusCode}");
   }
```

4. IoT Hub expects to get information about the status of processing feedbacks:

```
await feedbackReceiver.CompleteAsync(feedbackBatch);
```

5. The API allows to configure message to a device with the settings which indicate what kind of response is expected by the service, expiration date of a message and set message id which makes it easier later to correlate message with the feedback:

```
message.Ack = DeliveryAcknowledgement.Full;
message.MessageId = Guid.NewGuid().ToString();
message.ExpiryTimeUtc = DateTime.UtcNow.AddSeconds(10);
```

Executing direct methods on a device

1. Azure IoT Hub API allow to execute methods on a device. DeviceClient object expose a method which allow to register function that IoT Hub can execute:

```
await device.SetMethodHandlerAsync("showMessage", ShowMessage, null);
```

2. Second parameter indicates which method to call, when method with the name specified as a first argument is executed,

3. Example implementation of device method:

```
private static Task<MethodResponse> ShowMessage(MethodRequest methodRequest, object userContext)
 {
    Console.WriteLine("***Message received***");
    Console.WriteLine(methodRequest.DataAsJson);
    var responsePayload = Encoding.ASCII.GetBytes("{\"response\": \"Message shown!\"}");
    return Task.FromResult(new MethodResponse(responsePayload, 200));
 }
```

4. Azure CLI allow to execute direct method on a device by using following command:

```
az iot hub invoke-device-method -d %NAME_OF_THE_DEVICE% -n %IOT_HUB_NAME% --mn "showMessage"
```

5. Example implementation of the method on the service side:

```
private static async Task CallDirectMethod(ServiceClient service, string deviceId)
{
   var method = new CloudToDeviceMethod("showMessage");
   method.SetPayloadJson("'Hello from cloud'");
   var response = service.InvokeDeviceMethodAsync(deviceId, method);
   Console.WriteLine($"Response status: {response.Status}, payload: {response.Result.GetPayloadAsJson()}");
}
```