7) Development of PoC

In order to investigate the feasibility for SMEs to develop IoT products and devices taking into account the criteria of chapter 5.5, this chapter of the research project describes the development of a PoC.

7.1) PoC description

As hardware for the PoC development, an MS Azure Sphere MediaTek 3620 (hereinafter: MT3620) development kit (see Figure 1 - caption 1) manufactured by Seeed Technology Co. Ltd. is used. In addition, an MS Azure Sphere Grove Kit (caption 2), also by Seeed Technology Co. Ltd., is attached, which allows connecting a humidity and temperature sensor (caption 3), a display (caption 4), a button with an indication LED (caption 5), and a relay (caption 6) via an I2C interface to the MT3620. MS Azure Security Services is used for updating the MT3620 through over-the-air (OTA) updates, which is mandatory for the MT3620 in combination with MS. A dashboard and some controls are realised with an MS Azure IoT Central application (see Figure 2 below) for this PoC. The MS Azure Sphere cloud services and MT3620 architecture are explained in chapters 7.2 and 7.3.

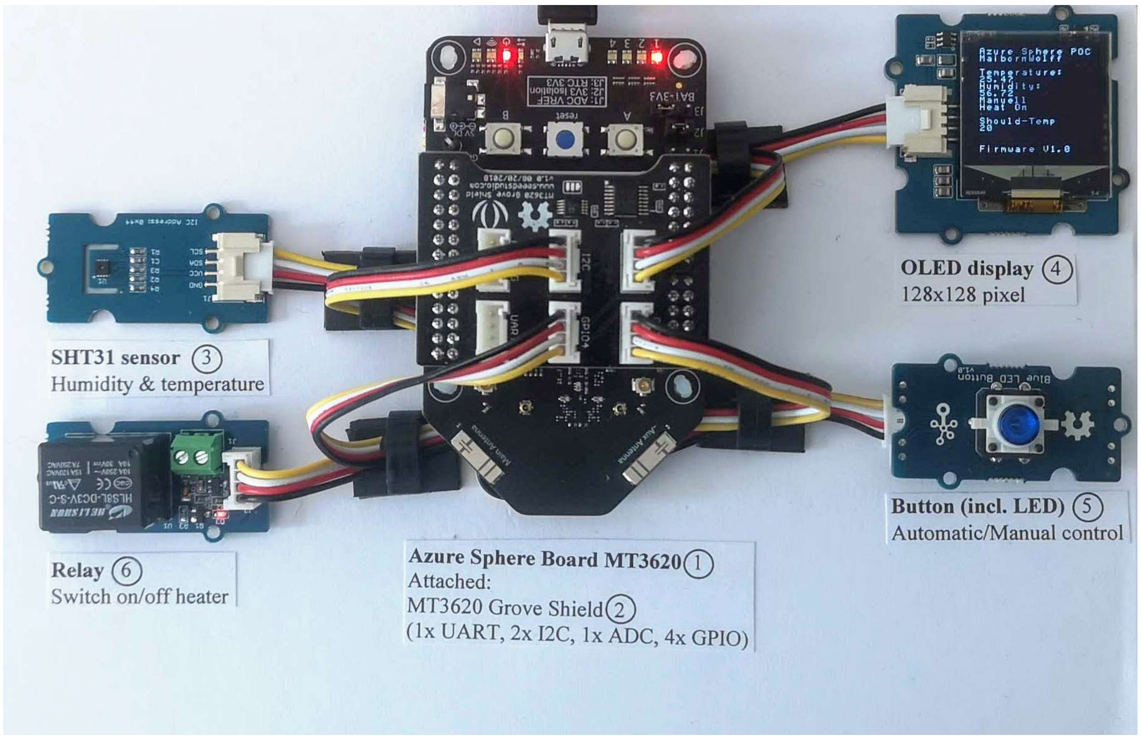
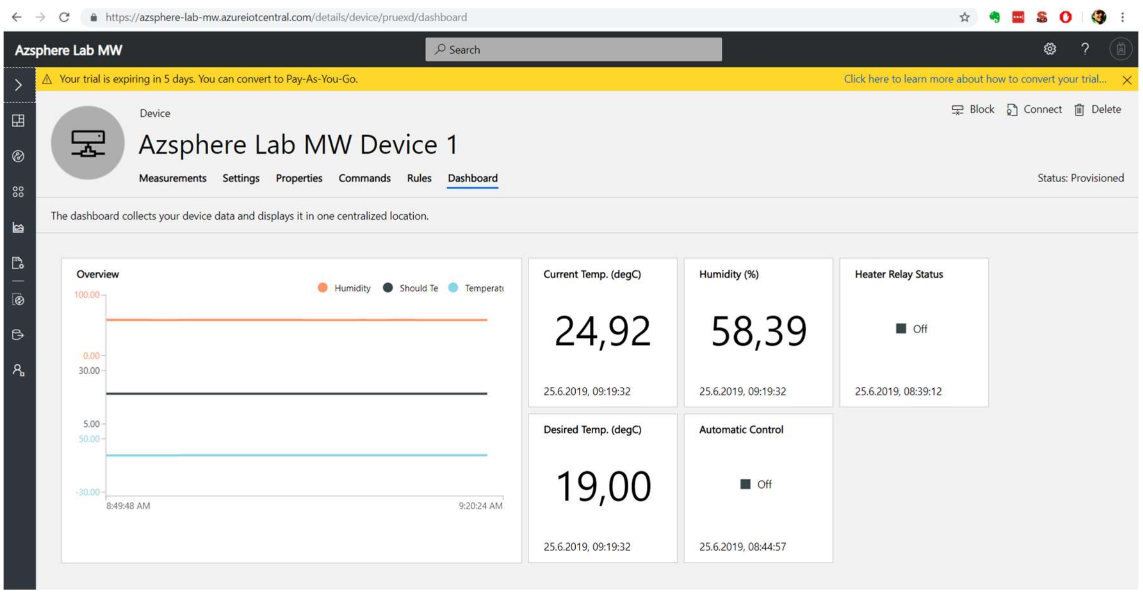


Figure 1: Developed PoC including all hardware

The above-mentioned hard- and software combination is chosen because of the likelihood that it will provide a secure and little resource-demanding possibility for SMEs to realise an IoT system. MS Azure Sphere is advertised to be game changer for security and IoT. Since within the scope of this research project, only one PoC can be developed and investigated, the MT3620 is chosen because of the ambitions in security and integrated development environment (Microsoft Corp., 2019i). Another system providing similar features is AWS Greengrass (Amazon Web Services Inc., 2019b).


https://azsphere-lab-mw.azureiotcentral.com/details/device/pruexd/dashboard

Your trial is expiring in 5 days. You can convert to Pay-As-You-Go.

Click here to learn more about how to convert your trial...

The dashboard collects your device data and displays it in one centralized location.

Status: Provisioned

Current Temp. (degC) 24,92 25.6.2019, 09:19:32

Humidity (%) 58,39 25.6.2019, 09:19:32

Heater Relay Status Off 25.6.2019, 08:39:12

Desired Temp. (degC) 19,00 25.6.2019, 09:19:32

Automatic Control Off 25.6.2019, 08:44:57

Figure 2: Dashboard and controls of MS Azure IoT Central

The development of the PoC demonstrates all necessary tasks to set up an MS Azure Sphere MT3620 IoT system to communicate with the MS Azure IoT Central cloud application. This also automatically includes a device twin, which ensures that controls to an IoT device get delivered even if the device is currently offline. The device twin saves settings and sends it to the IoT device once it is online again. After development, the PoC and development steps receive feedback and get evaluated by an expert in chapter nine.

The PoC demonstrates a control unit for an automatically controlled heater. The following are the basic features and functionality to be achieved for the PoC:

- Receiving, updating and sending data using JavaScript Object Notation (JSON) files and a device twin
- Connecting to MS Azure IoT Central application
- Measuring temperature and humidity and sending it to MS Azure IoT Central
- Displaying information about current temperature and humidity, heat switch status, pre-set temperature, firmware version, automatic or manual control
- Button for choosing between automatic or manual control
- Controlling an attached relay for a heater
- Adjusting a desired temperature with two buttons (up and down) starting at 19 degC
- Switching On/Off the attached relay for the heater
- Receiving data from the IoT device and providing all information in a dashboard

The source code of the developed PoC is available on GitHub Inc. (2019a). The project file can be found under GitHub Inc. (2019b). The developed code includes libraries from two examples of MS "Azure IoT" and "UART", which can be accessed under GitHub Inc. (2019c). In addition, libraries were used for the MS Azure Sphere Grove Kit, which can be accessed under GitHub Inc. (2019d). This is also noted in the newly developed source code for the PoC.

7.2) MS Azure Sphere

MS Azure Sphere is a software and cloud product of Microsoft Corporation, which aims to be a solution for a secure and easy-to-establish IoT system. It consists of an OS, special hardware for IoT devices and cloud components. Microsoft promises four main features for MS Azure Sphere (Microsoft Corp., 2019t):

- Defense in depth
- Deployment flexibility
- Over-the-air (OTA) updates
- Error reporting and automatic security updates

To ensure all the above-mentioned features and security, MS Azure Sphere makes use of certified chips of Microsoft hardware partners. In addition, the MS Azure Sphere OS automatically receives ongoing security updates using the MS Azure Sphere Security Service, which guarantees a secure device-to-cloud communication (ibid.). The interaction of the different services and update paths is explained in chapter 7.4.

7.3) MT3620 microcontroller

As discussed, the MS Azure Sphere environment can only be used in full by using a certified chipset for the IoT system. The first chipset fulfilling all hardware requirements is the MT3620. This chip is a specially designed microchip and allows running the OS with its security features and the Wi-Fi network isolated from a customizable high-level application. The high-level application is the programme that will be developed by the SMEs to support all functions for their products, such as displays, buttons, etc. Only hardware features, such as I2C, UART, etc. of the chip that have explicitly been enabled by the MS Azure Sphere OS and configuration files of the high-level application software image can be used during operation of the device. This results in a lower risk of manipulations. The in-built hardware separation of the MT3620 guarantees that security features and Wi-Fi can only be accessed and managed by the MS Azure Sphere APIs and security is managed by Microsoft, but not by the high-level application. Therefore, the high-level application is fully customizable by the engineers for the product without risking security issues (MediaTek Co., 2019). This makes it easy for SME developers to develop firmware for products without risking security issues. A scheme of the functional blocks of the MT3620 is given in Figure 3.

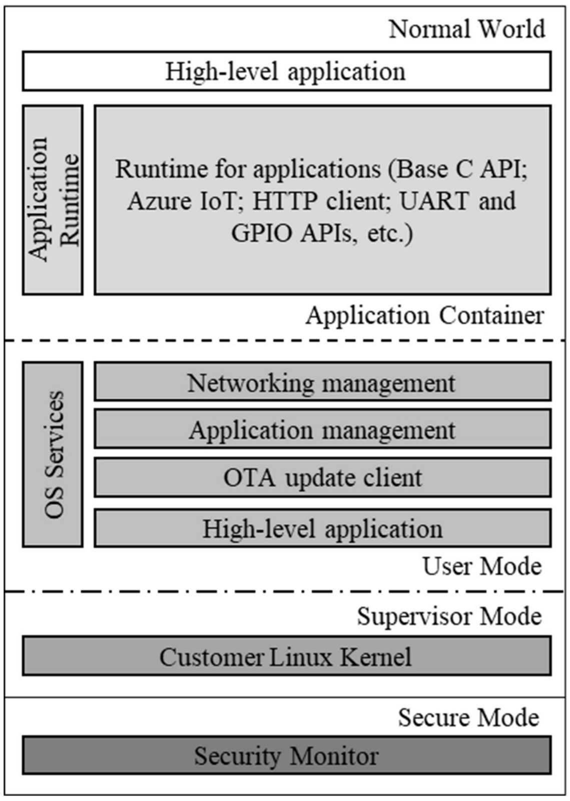
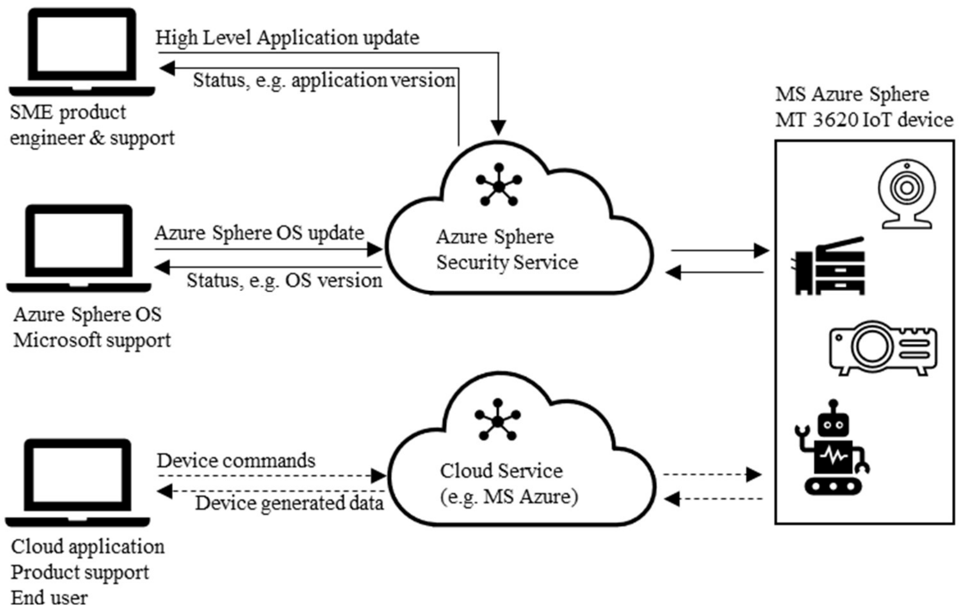


Figure 3: Software architecture of the MS Azure Sphere MT3620. Own elaboration based on Microsoft Corp. (2019i).

7.4) IoT system with MS Azure Sphere and MT3620

MS Azure Sphere might support more devices and services in the future, but for this research project, the first fully supported and certified chipset is used, which is the MT3620 by MediaTek Co. as an IoT device in combination with MS Azure Sphere. Figure 4 illustrates the basic functionality and connection between an MT3620, MS Azure Sphere Security Service and a cloud service such as MS Azure IoT Central.



High Level Application update
Status, e.g. application version
SME product engineer & support
MS Azure Sphere MT 3620 IoT device
Azure Sphere OS update
Status, e.g. OS version
Azure Sphere Security Service
Azure Sphere OS Microsoft support
Device commands
Device generated data
Cloud Service (e.g. MS Azure)
Cloud application Product support End user

Figure 4: Overview of connections between clients, clouds and MT3620. Own elaboration based on Microsoft Corp. (2019i).

To ensure security, three different links between the IoT device and cloud services must be considered. The MS Azure Sphere Security Service is mandatory for every device in this environment. Another cloud service is needed to actually make use of the collected data or to send commands to the IoT device. This can be another MS Azure service or another non-MS Azure third party service (Microsoft Corp., 2019i). As mentioned, this is MS Azure IoT Central for the current PoC development.

The MS Azure Sphere OS is maintained and updated by Microsoft engineers and the Microsoft support team. Microsoft ensures the security of the whole system and pushes regular system updates for the MS Azure Sphere OS to provide latest security patches for the base system. The IoT device can only pull these updates – with no exception – from the MS Azure Sphere Security Service cloud. The MT3620 automatically checks for OS updates regularly or after a reset of the device. This structure ensures a secure and updated OS for the IoT device without the interaction of the company selling the device. Microsoft may receive information back about the version and status of the IoT device. This cannot be finally answered from the documentation (ibid.).

The high-level application holds the programme structure for the physical IoT device. Not only can the MT3620 connect and transfer data between devices and clouds as gateway, it is also a fully capable microcontroller. Products like washing machines, dish washers and small industrial machines can be controlled by the MT3620 in addition to transceiving data to a cloud. Product engineers can develop a high-level application by using MS Visual Studio and the MS Azure Sphere developer kit. When the high-level application is successfully developed, the engineers upload the update file to the MS Azure Sphere Security Service. The MT3620 checks for new updates every 24 hours, while checking for OS updates at the same time or after a reset of the device, and automatically downloads and installs an update of the high-level application if available. Devices are tied to certain update feeds to ensure that every device only gets the updates foreseen for that specific device. More explanation on update feeds is given in chapter 7.5.

The collected data and control commands are sent by the high-level application. This part of the system is completely separated from the MS Azure Sphere OS and other security-relevant systems. While the updates of the MS Azure Sphere OS and the updates of the high-level application can only be received by the device via the MS Azure Sphere Security Service, the high-level application can connect to other cloud services outside the MS Azure environment. For a fully integrated solution, it is of course possible to also use MS Azure as a cloud service for the IoT device, as in the described PoC.

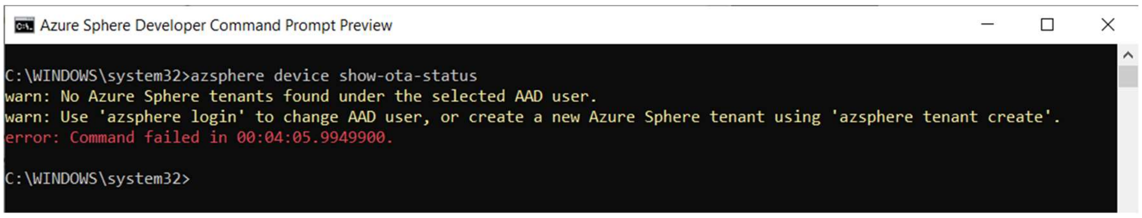
7.5) Development steps

In this section, a description is given of all necessary development steps to build the PoC as described in chapter 7.1. While giving in-depth insights into the overall process, it is not meant to be a tutorial. The description focuses on the steps to connect the MT3620 to the cloud for updates and transmit data to the MS Azure IoT Central application. This explanation excludes the programme code of the high-level application. The high-level application can be downloaded and includes detailed comments in the source code.

In this chapter, the term “step” designates a development step that may include more than one action in order to complete a certain milestone for the system. The PoC makes use of the previously described MS Azure Sphere environment in combination with an MT3620 and an MS Azure IoT Central application. The first of eight development steps involves setting up the integrated development environment (IDE) and updating the factory-new MT3620.

Step 1	**Commissioning of MS Azure Sphere MT3620 and IDE installation**
1.1	Installing MS Visual Studio (free, Community Version)
1.2	Installing MS Azure Sphere Development SDK
1.3	Command: *“azsphere device show-ota-status”*
1.4	Command: *“azsphere device recover”*

Microsoft provides all tools for the development of the PoC for free. First, MS Visual Studio must be installed. In addition, the freely available software development kit (SDK) for MS Azure Sphere and MT3620 need to be installed (Microsoft Corp., 2019j). After the successful installation, a designated command prompt is available: “Azure Sphere Developer Command Prompt”. The commands in 1.3 and 1.4 are needed to step up and configurate the MT3620 after all drivers have been installed automatically by MS Windows 10. With the command *“azsphere* *device show-ota-status"*, the current software version of MS Azure Sphere OS on the MT3620 is shown. The OS is preloaded to the chip but might be outdated. Using the command *"azsphere device recovery"* will sideload the latest OS version in the MT3620 via a USB connection (Microsoft Corp., 2019k). In order to get a better idea of this process, Figure 5 shows the Azure Sphere Developer Command Prompt with the command of step 1.3:



```
C:\WINDOWS\system32>azsphere device show-ota-status
warn: No Azure Sphere tenants found under the selected AAD user.
warn: Use 'azsphere login' to change AAD user, or create a new Azure Sphere tenant using 'azsphere tenant create'.
error: Command failed in 00:04:05.9949900.

C:\WINDOWS\system32>
```

Figure 5: Screenshot of the Azure Sphere Developer Command Prompt (step 1.3)

After these initial steps the MT3620 is up to date and all necessary IDEs and SDKs are installed, and the development can start.

Step 2 MS Azure tenant and claiming of a device

2.1	Command: *"azsphere login"*
2.2a	Command: *"azsphere tenant create --name <tenant_name>"*
2.2b	Command: *"azsphere tenant create --force --name <tenant_name>"*
2.3	Command: *"azsphere device claim"*

Before the MT3620 can be used as an IoT device, it is necessary to claim the device to a tenant. A tenant is part of an MS Azure account. One tenant can contain many devices, but a device can only be claimed to one tenant. The claiming to a tenant is also a one-time step and cannot be revoked or repeated. Once a MT3620 chip is claimed to a tenant, it is permanent. An MS tenant requires an MS Azure account (Microsoft Corp., 2019l). An MS Azure account can have more than one tenant, but it is recommended by Microsoft to only have one tenant per account. The reason for this recommendation remains unclear from the documentation. With this step, the ownership and all administrative rights are bound to this account with its tenant (Microsoft Corp., 2019m). Step 2.1 logs into the dedicated MS Azure account (Microsoft Corp., 2019l). Within this account, a new tenant can be created with the command of 2.2a. The command expects a tenant name instead of the placeholder "*<tenant_name>"*. If one tenant in the account already exists but another one shall be created, the command of step 2.2b is needed. With the additional parameter *"--force"* a second tenant in the account can be created. The last step is the claiming of the device to a tenant through step 2.3. If more than one tenant in this MS Azure account is available, the Azure Sphere Developer Command Prompt provides an option to choose (Microsoft Corp., 2019m).

Step 3 Connection with a wireless network

3.1	Command: *"azsphere device wifi show-status"*
3.2	Command: *"azsphere device wifi add --ssid <SAMPLEssid> --key <NetworkKey>"*

Step three connects the IoT device with a Wi-Fi network. In step 3.1, the command will list Wi-Fi and network details, while in step 3.2 the command will add a Wi-Fi. It is possible to add and remove Wi-Fi networks. The MT3620 can save more than one set of Wi-Fi credentials and the MS Azure Sphere OS will automatically choose the most suitable one. The MT3620 is now connected to a Wi-Fi and has internet access (Microsoft Corp., 2019n)

Step 4 Developing and debugging of high-level MS Azure Sphere MT3620 applications

4.1 Command: *"azsphere device prep-debug –EnableRTCoreDebugging"*

In step four, the MT3620 is prepared for development and debugging. This means that the device-specific high-level application gets sideloaded via a USB connection from the developer´s workstation into the MT3620 and OTA updates via the MS Azure Security Service are disabled. By default, the MT3620 is locked and cannot receive data from a connected PC (Microsoft Corp., 2019o). Also, part of this process is that the MT3620 gets erased from any formerly installed high-level application. Disabling OTA updates and erasing the chip ensures the development of a new high-level application on a clean device. After erasure, the MS Azure Sphere OS remains on the chip and the chip is still claimed to a tenant.

Step 5 High-level application configuration for MS Azure IoT Central

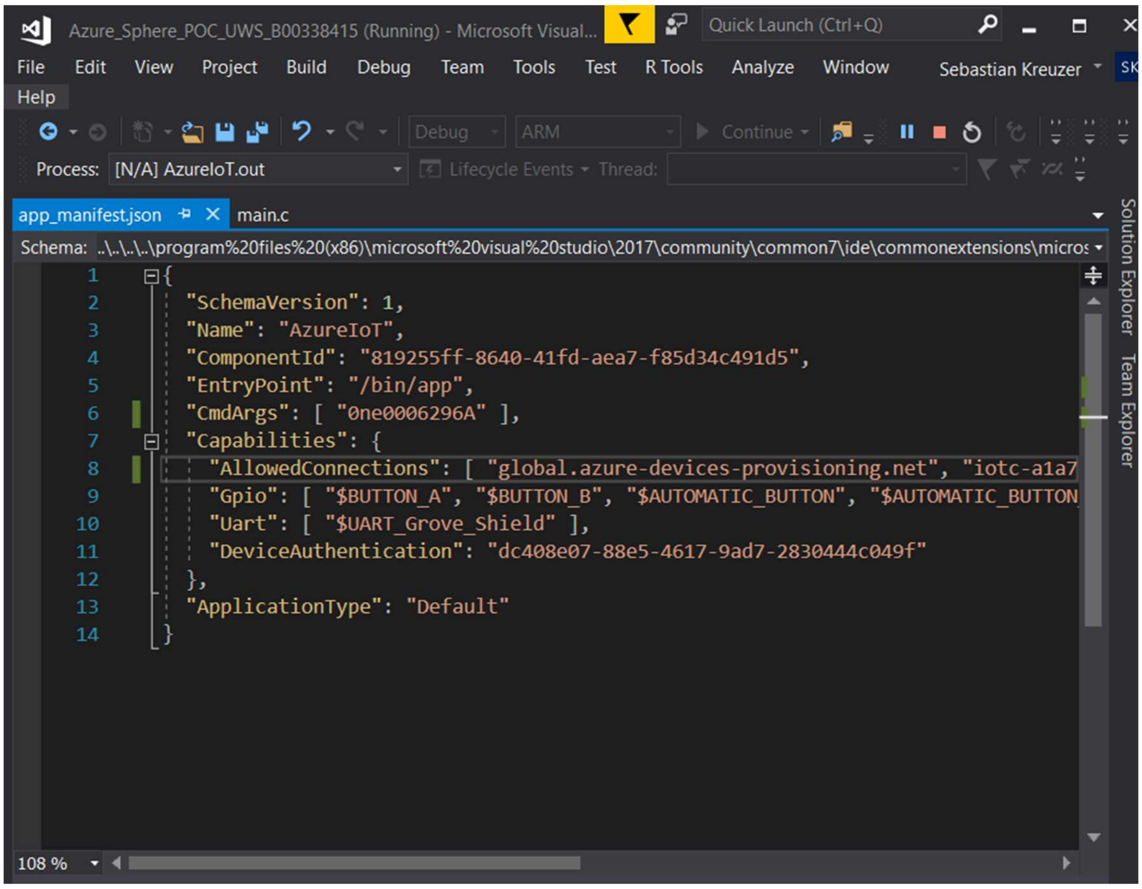
5.1 Command: *"azsphere tenant show-selected"*

5.2 ShowIoTCentralConfig.exe

5.3 In MS Visual Studio High Application adjust „app_manifest.json“ with the parameters

Step five is responsible for preparing the high-level application to connect to an MS Azure IoT Central application. After this step, it is still mandatory to establish trust between the tenant the device belongs to and the IoT Central application. This will be part of step six, but first the high-level application for the MT3620 will be prepared for the connection.

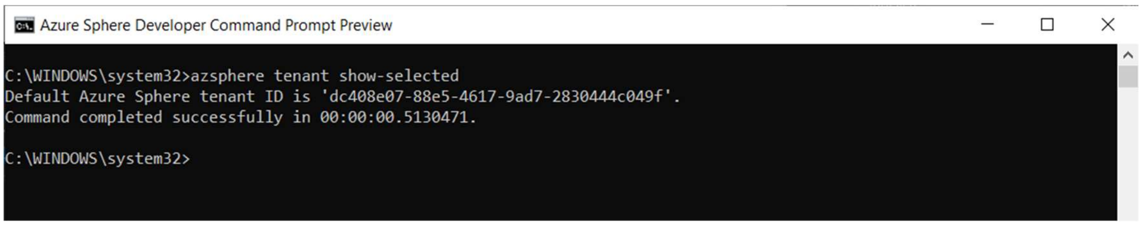
The project of the high-level application in MS Visual Studio contains a file named "manifest.json", which holds important parameters for the operation of the MT3620 IoT device (Figure 6). Some of these parameters need to be adjusted to connect to the MS Azure IoT Central application (Microsoft Corp., 2019p).


Azure_Sphere_POC_UWS_B00338415 (Running) - Microsoft Visual...
Quick Launch (Ctrl+Q)
File
Edit
View
Project
Build
Debug
Team
Tools
Test
R Tools
Analyze
Window
Sebastian Kreuzer
Help
Debug
ARM
Continue
Process: [N/A] AzureIoT.out
Lifecycle Events
Thread:
app_manifest.json
main.c
Schema: ..\..\..\program%20files%20(x86)\microsoft%20visual%20studio\2017\community\common7\ide\commonextensions\micros
Solution Explorer
Team Explorer
108 %

```
{
  "SchemaVersion": 1,
  "Name": "AzureIoT",
  "ComponentId": "819255ff-8640-41fd-aea7-f85d34c491d5",
  "EntryPoint": "/bin/app",
  "CmdArgs": [ "0ne0006296A" ],
  "Capabilities": {
    "AllowedConnections": [ "global.azure-devices-provisioning.net", "iotc-a1a7
    "Gpio": [ "$BUTTON_A", "$BUTTON_B", "$AUTOMATIC_BUTTON", "$AUTOMATIC_BUTTON
    "Uart": [ "$UART_Grove_Shield" ],
    "DeviceAuthentication": "dc408e07-88e5-4617-9ad7-2830444c049f"
  },
  "ApplicationType": "Default"
}
```

Figure 6: Screenshot of the "app_manifest.json" in MS Visual Studio

By using the command of step 5.1 in the Azure Sphere Developer Command Prompt we receive in return the tenant ID (Figure 7), which needs to be inserted in the above-mentioned file for the *"DeviceAuthentication"* (ibid.).

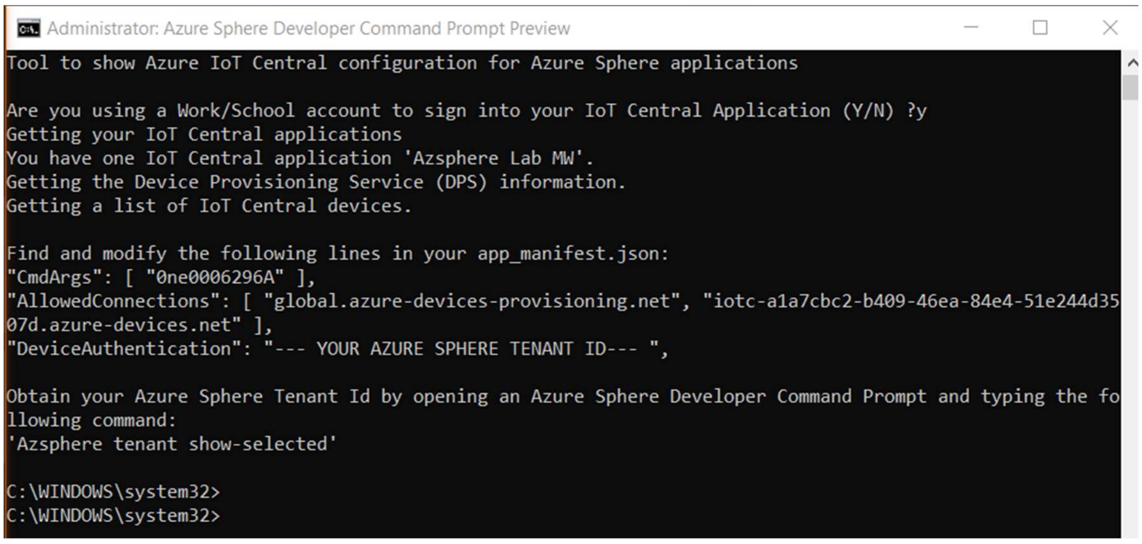


```
C:\WINDOWS\system32>azsphere tenant show-selected
Default Azure Sphere tenant ID is 'dc408e07-88e5-4617-9ad7-2830444c049f'.
Command completed successfully in 00:00:00.5130471.

C:\WINDOWS\system32>
```

Figure 7: Screenshot of the Azure Sphere Developer Command Prompt (step 5.1)

The other important parameters must be deduced using a tool that is part of the "AzureIoT" sample on GitHub. In step 5.2 *"ShowIoTCentralconfig.exe"* will return all other necessary parameters that need to be included in *"manifest.json"* file (Figure 8). After adjusting the "manifest.json" file it must be saved in the MS Visual Studio project, so that the changes remain (ibid.).


Administrator: Azure Sphere Developer Command Prompt Preview

```
Tool to show Azure IoT Central configuration for Azure Sphere applications

Are you using a Work/School account to sign into your IoT Central Application (Y/N) ?y
Getting your IoT Central applications
You have one IoT Central application 'Azsphere Lab MW'.
Getting the Device Provisioning Service (DPS) information.
Getting a list of IoT Central devices.

Find and modify the following lines in your app_manifest.json:
"CmdArgs": [ "0ne0006296A" ],
"AllowedConnections": [ "global.azure-devices-provisioning.net", "iotc-a1a7cbc2-b409-46ea-84e4-51e244d35
07d.azure-devices.net" ],
"DeviceAuthentication": "--- YOUR AZURE SPHERE TENANT ID--- ",

Obtain your Azure Sphere Tenant Id by opening an Azure Sphere Developer Command Prompt and typing the fo
llowing command:
'Azsphere tenant show-selected'

C:\WINDOWS\system32>
C:\WINDOWS\system32>
```

Figure 8: Screenshot of the Azure Sphere Developer Command Prompt (step 5.2)

The programme routine to send data from a sensor and receive commands for the IoT device is shown in the AzureIoT example (ibid.), because the present research project focuses on the cloud connection and the overall operation of the device. The provided source code for this PoC includes comments regarding sending single data sets (GitHub Inc., 2019e).

Step 6	**Authentication of MS Azure tenant for MS Azure IoT Central**
6.1	Create a new MS Azure IoT Central account and application
6.2	Command: *"azsphere login"*
6.3	Command: *"azsphere tenant dowload-CA-certification --output <file_name>"*
6.4	IoT Central portal: Administration>>Device Connection>>Certificates (X.509)
6.5	Add certificate
6.6	Press refresh button to generate verification code
6.7	Command: *"azsphere tenant download-validation-certificate --output <file_name> --verificationcode <code>"*
6.8	Click verify and use the validation certification of step 6.7

In step six, a secure connection via certificates is set up between the tenant the device has been claimed to in step 2.3 and the MS Azure IoT Central application. A certificate needs to be downloaded and afterwards validated.

First, an account and a new MS Azure IoT Central application must be created in step 6.1. For the next steps in the MS Azure IoT Central application a certificate is needed, which can be downloaded using the Azure Sphere Developer Command Prompt and the commands in steps 6.2 and 6.3. In steps 6.4 and 6.5, the certificate gets uploaded to MS Azure IoT Central and provides a verification code which is needed in step 6.7 to generate and download a validation certificate using the developer prompt again. In step 6.8 the validation is complete, and a trustful connection is established between the tenant and the MS Azure IoT Central application (Microsoft Corp., 2019q).

Step 7	**Creating MS Azure IoT Central application and adding a device**
7.1	In MS Azure IoT Central application create a new device template
7.2	Add all telemetry data that are sent from an MS Azure Sphere MT3620 application
7.3	Add a real device from the created template
7.4a	Get the device ID by command: *"azsphere device show-attached"*
7.4b	Get the device ID alternative: VisualStudio>Project>Add Connected Service>Device Status
7.5	Insert the device ID for the new real device in MS Azure IoT Central

An MS Azure IoT Central application can include different kinds of IoT devices with specific data sent and received for a device. In order to define a device, a new device template needs to be created in step 7.1. For example, the developed PoC sends, among other data, the temperature and humidity and receives commands to switch on/off a heater or an automatic mode. We can define these data in the IoT Central application through the device template in step 7.2. After a template is defined, a real device can be added in step 7.3. The adding of the real device requires the device ID, which is unique for every MT3620 and can be retrieved via the Azure Sphere Developer Command Prompt or with the IDE, which relates to step 7.4a and 7.4b. After adding the device ID in step 7.5 the data will start to flow into MS Azure IoT Central and will be displayed in the measurements tag (Microsoft Corp., 2019p). Every device that is part of the same tenant can easily be added by just inserting the device ID as shown in this step. All devices claimed to the same tenant benefit from the secure connection established in step six with the created MS IoT Central application.

Step 8	**Over-the-air software update and update feeds**
8.1	Command: *"azsphere device prep-field --newdevicegroupname <unique-dev-group> --newskuname <unique-sku>"*
8.2	Command: *"azsphere feed list"*
8.3a	Command: *"azsphere device link-feed --dependentfeedid <Retail Azure Sphere OS> --imagepath <pathtoimage> --newfeedname <uniquefeedname>"*
8.3b	Command: *"azsphere device prep-field --skuid skuid --devicegroupid groupid"*

Step eight demonstrates the process of automatic OTA updates for the IoT device. Before the IoT device can receive updates, a device group, e.g. printer, dishwasher, light ball, etc. and a unique stock keeping unit (SKU) for the product must be defined with the command in step 8.1. This will determine which update feed the IoT device will listen to. With MS Azure Sphere, different feeds can be created to deliver updates to groups defined by the device group and SKU. Step 8.3b applies when a feed already exists (Microsoft Corp., 2019s).

When a new feed is created the update feed ID of the correct MS Azure Sphere OS version needs to be retrieved with the command in step 8.2. In order to upload a new high-level application and create a new feed, the command in step 8.3a is used. The uploaded high-level application will be bound to an MS Azure OS version and its feed ID in step 8.2. It is recommended to link the application to the stable "*Retail Azure Sphere OS*" version (Microsoft Corp., 2019r). The ID for this version can be retrieved from the Azure Sphere Developer Command Prompt. If an update feed for a device group and SKU already exists, the device can be added to the feed with step 8.3b (Microsoft Corp., 2019s). The command *"azsphere device prep-field"* in steps 8.1 and 8.3b closes the development and debugging mode. Therefore, the IoT device can now receive OTA updates and stops accepting sideloaded images (ibid.).