Azure RTOS and the MXCHIP IoT DevKit

By Sean D. Liming and John R. Malin
Annabooks – www.annabooks.com

May 2023

There are a number of Azure RTOS online guides to get started with different platforms. The MXCHIP IoT DevKit was one of the first platforms that demonstrated connecting to Azure IoT Central. If you follow the quick start online documents, you will be able to build the example application from the command line and get it to run. If you want to use the example applications as a basis for the project, having the ability to step through the code with a debugger is going to be important. In this paper, we will walk through the example but set up the development environment to use Visual Studio Code.

MXCHIP manufactures ARM Core + Wi-Fi modules that are small and fit tightly resource-constrained applications. The MXCHIP IoT DevKit is an example platform demonstrating cloud connectivity. The board documentation is light and gets a little confusing. The actual target ARM core on the board is from ST Microelectronics: STM32F412 - Arm® Cortex®-M4, which is built into the MXCHIP module, but the documentation calls out STM32F103CBT6 - Arm® Cortex®-M3, which is for the STLINK on-chip debugger. Please be aware that there are some little discrepancies like this in this demonstration platform.

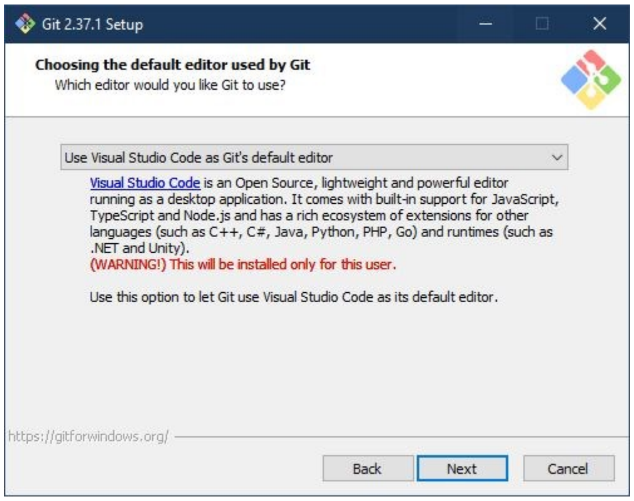
1.1 Tools Setup

For this setup will we need to download and install a few items.

1. Download and install Visual Studio Code: https://code.visualstudio.com/Download .
2. Once Visual Studio Code has been installed, install the following add-ons from the Visual Studio Code marketplace:

    - C/C++ - Visual Studio Marketplace
    - CMake Tools - Visual Studio Marketplace
    - CMake - Visual Studio Marketplace
    - Cortex-Debug - Visual Studio Marketplace

3. Install Git for downloading the Azure RTOS to get started building files: Git - Downloads (git-scm.com).
    a. Accept the license, and click Next.
    b. Leave the install location as is, and click Next.
    c. Leave the Selected Components as they are, and click Next.
    d. Keep the State Menu Folder as is, and click Next.
    e. Set the default editor selection to be “Use Visual Studio Code as Git’s default editor”, and click Next.



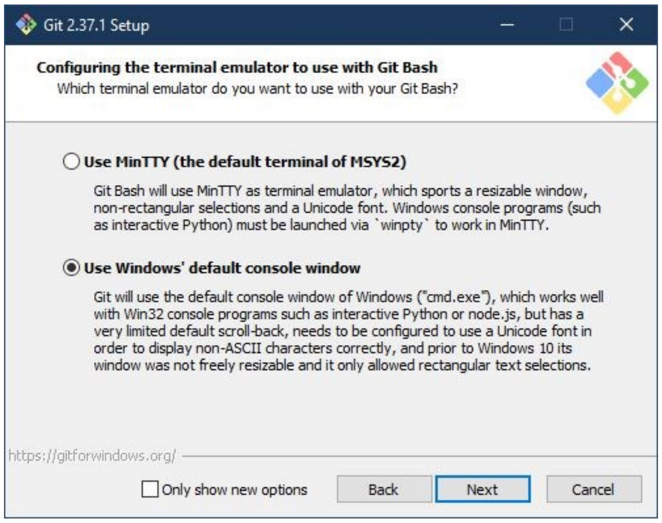
Choosing the default editor used by Git
Which editor would you like Git to use?

Use Visual Studio Code as Git's default editor

Visual Studio Code is an Open Source, lightweight and powerful editor running as a desktop application. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages (such as C++, C#, Java, Python, PHP, Go) and runtimes (such as .NET and Unity).
(WARNING!) This will be installed only for this user.

Use this option to let Git use Visual Studio Code as its default editor.

f. Keep the default for initial branches, and click Next.
g. Keep the default PATH Environment, and click Next.
h. Keep the default OpenSSH selection, and click Next.
i. Select "Use Windows' default console window", and click Next.



Configuring the terminal emulator to use with Git Bash
Which terminal emulator do you want to use with your Git Bash?

○ **Use MinTTY (the default terminal of MSYS2)**

Git Bash will use MinTTY as terminal emulator, which sports a resizable window, non-rectangular selections and a Unicode font. Windows console programs (such as interactive Python) must be launched via `winpty` to work in MinTTY.

[✓ **Use Windows' default console window**]

Git will use the default console window of Windows ("cmd.exe"), which works well with Win32 console programs such as interactive Python or node.js, but has a very limited default scroll-back, needs to be configured to use a Unicode font in order to display non-ASCII characters correctly, and prior to Windows 10 its window was not freely resizable and it only allowed rectangular text selections.

j. Keep the defaults for the next question, and click Next.
k. Keep the defaults for the next question, and click Next.
l. Keep the defaults for the next question, and click Next.
m. Keep the defaults for the extra options, and click Next.
n. Keep the defaults for the experimental options, and click Install.
o. Click Finish once the install completes.

4. Download and install the STM32 Cube Programmer from ST Microelectronics: STM32CubeProg.

5. Download and install the ARM GNU Toolchain: https://developer.arm.com/-/media/Files/downloads/gnu-rm/10.3-2021.10/gcc-arm-none-eabi-10.3-2021.10-win32.exe

Note: Per the description page: Arm GNU Toolchain, this version of the toolchain has been deprecated. The newer tools are found here: GNU Arm Embedded Toolchain Downloads, but these tools will not work with the example software.

6. Download and install ABCOMTERM from Annabooks.com. This will be the terminal program to see the standard output from the devices.
7. Download the latest OpenOCD for Windows (gnutoolchains.com).
8. 7zip will be needed to extract the files.
9. Open Control Panel->System->Advanced system settings
10. Click on the Environment variables button, and edit the Path under System variables, to add the fully qualified path to the OpenOCD \bin folder.
11. Reboot the computer.

1.2 Download the Getting Started Files from GitHub

Now we need to get the getting started repository that contains the Azure RTOS build example and the ports to the MXCHIP IoT DevKit and other development kits.

1. Create a directory called \Azure-RTOS_MXCHIP
2. Open PowerShell
3. Change the directory to the newly created folder:

   ```
   cd \Azure-RTOS_MXCHIP
   ```

4. Run the following

   ```
   git clone --recursive https://github.com/azure-rtos/getting-started.git
   ```

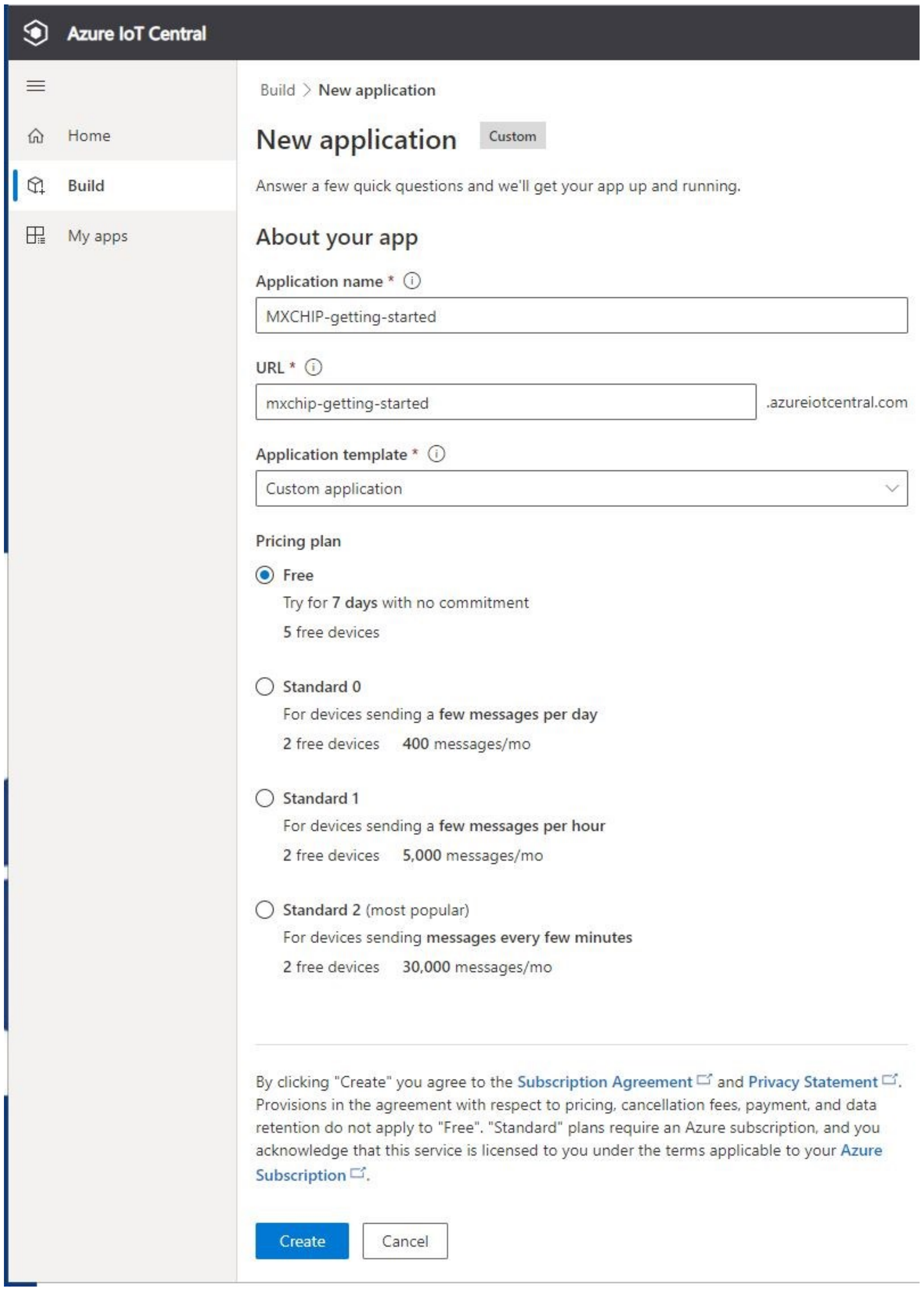
1.3 Create Azure IoT Central Application

Now we need to set up the application on Azure IoT Central.

1. In a browser, open https://apps.azureiotcentral.com/home
2. Sign into the account or create an account
3. Click on Build App
4. In the Custom app tile, click Create app.

   Application Name: MXCHIP-getting-started.
   Pricing Plan: Free.

5. Click Create.



New application Custom

Answer a few quick questions and we'll get your app up and running.

About your app

Application name *: MXCHIP-getting-started

URL *: mxchip-getting-started .azureiotcentral.com

Application template *: Custom application

Pricing plan

● Free
Try for **7 days** with no commitment
5 free devices

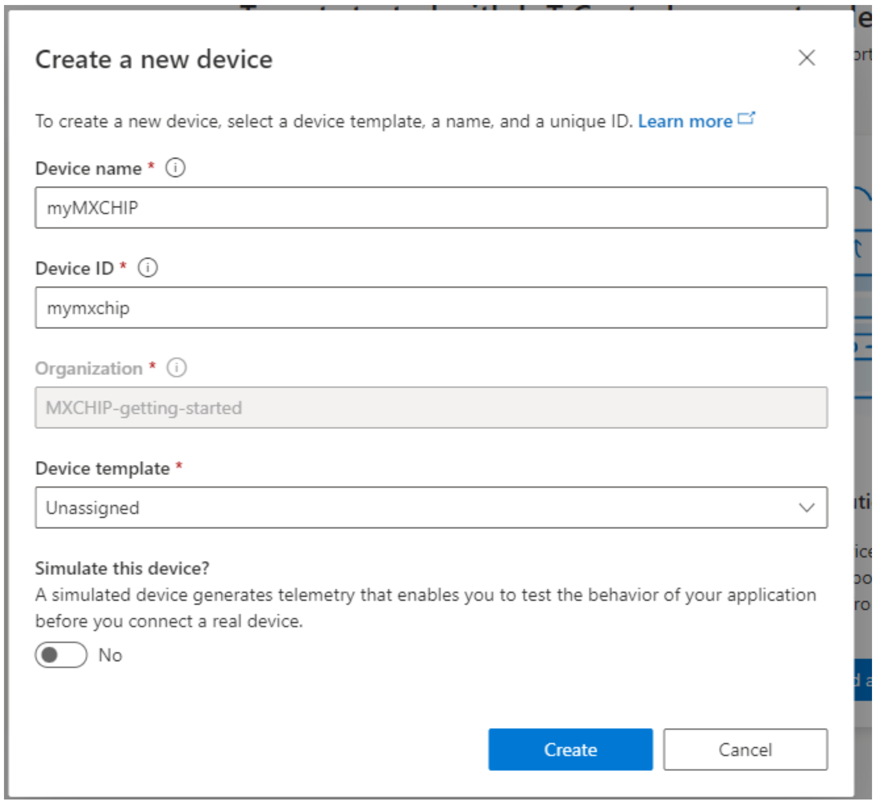
○ Standard 0
For devices sending a **few messages per day**
2 free devices **400** messages/mo

○ Standard 1
For devices sending a **few messages per hour**
2 free devices **5,000** messages/mo

○ Standard 2 (most popular)
For devices sending **messages every few minutes**
2 free devices **30,000** messages/mo

By clicking "Create" you agree to the Subscription Agreement and Privacy Statement. Provisions in the agreement with respect to pricing, cancellation fees, payment, and data retention do not apply to "Free". "Standard" plans require an Azure subscription, and you acknowledge that this service is licensed to you under the terms applicable to your Azure Subscription.

6. Now, we need to add a device to the application. Click on the +New button that is above the All Devices section.
7. Enter the following:
    a. Device Name: myMXCHIP
    b. Device ID mymxchip
8. Click Create.



Create a new device

To create a new device, select a device template, a name, and a unique ID. Learn more

Device name * : myMXCHIP

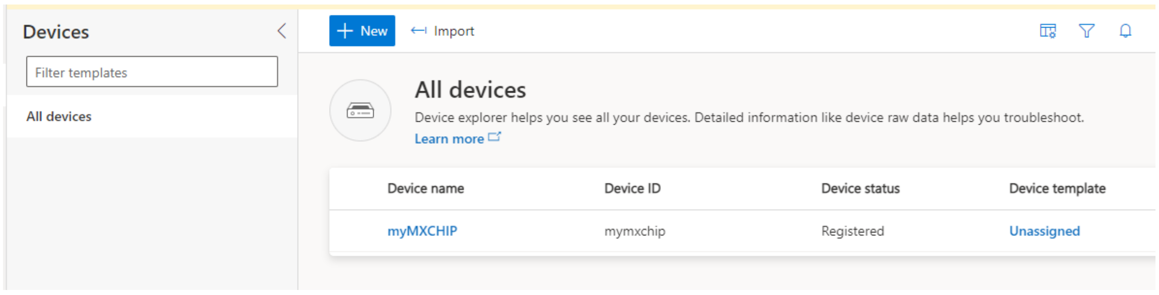
Device ID * : mymxchip

Organization * : MXCHIP-getting-started

Device template * : Unassigned

Simulate this device?
A simulated device generates telemetry that enables you to test the behavior of your application before you connect a real device.
No

9. The device will be created and listed under all devices

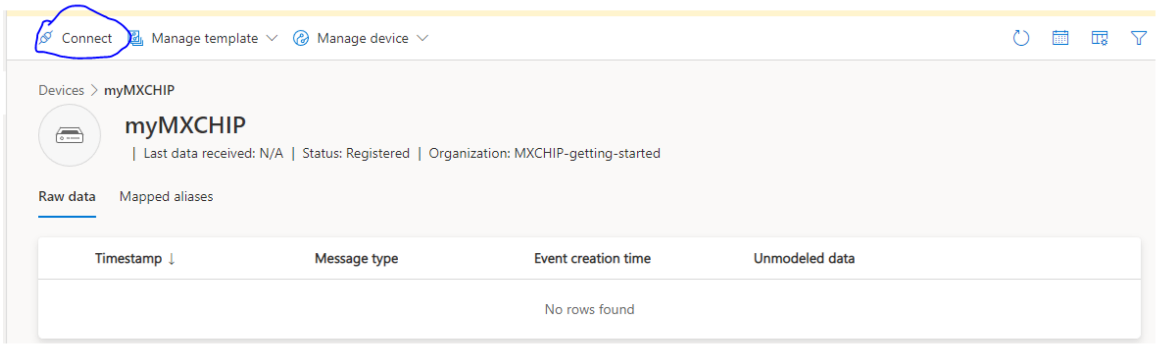


All devices

Device explorer helps you see all your devices. Detailed information like device raw data helps you troubleshoot. Learn more

Device name	Device ID	Device status	Device template
myMXCHIP	mymxchip	Registered	Unassigned

10. Click on myMXCHIP. This will be the view of the data coming in.
11. Click on Connect at the top of the bar



myMXCHIP

| Last data received: N/A | Status: Registered | Organization: MXCHIP-getting-started

Timestamp ↓	Message type	Event creation time	Unmodeled data
No rows found			

12. A Device Connections group box appears. Copy the following information and paste it into a Notepad or Notepad++ document for future use:

- ID scope
- Device ID
- Primary Key

13. Save the Notepad or Notepad++ document to be used later.
14. Close the dialog when finished.

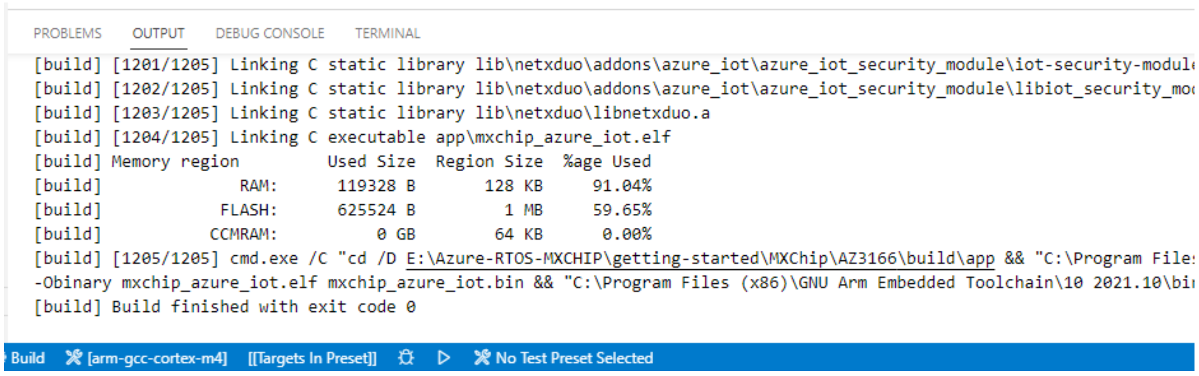
1.4 Building the Sample App

With the application created in Azure IoT Central and the device information collected to make the connection, we are ready to build the example.

1. In \Azure-RTOS-MXCHIP\getting-started\MXChip\AZ3166, double-click on AZ3166.code-workspace. This will open Visual Studio Code.
2. When asked for a toolchain at the top, accept arm-gcc-cortex-m4.
3. Under AZ3166\App, open Azure_config.h and fill in the information gathered from the Azure IoT Central application, as well as, your Wi-Fi connection settings:

Constant name	Value
IOT_DPS_ID_SCOPE	ID scope value
IOT_DPS_REGISTRATION_ID	Device ID value
IOT_DEVICE_SAS_KEY	Primary key value
WIFI_SSID	Your Wi-Fi SSID
WIFI_PASSWORD	Your Wi-Fi password
WIFI_MODE	WEP, WPA_PSK_TKIP, or WPA2_PSK_AES

4. Save the file.
5. At the bottom, click on Build. It will take a few minutes, but the build should complete successfully

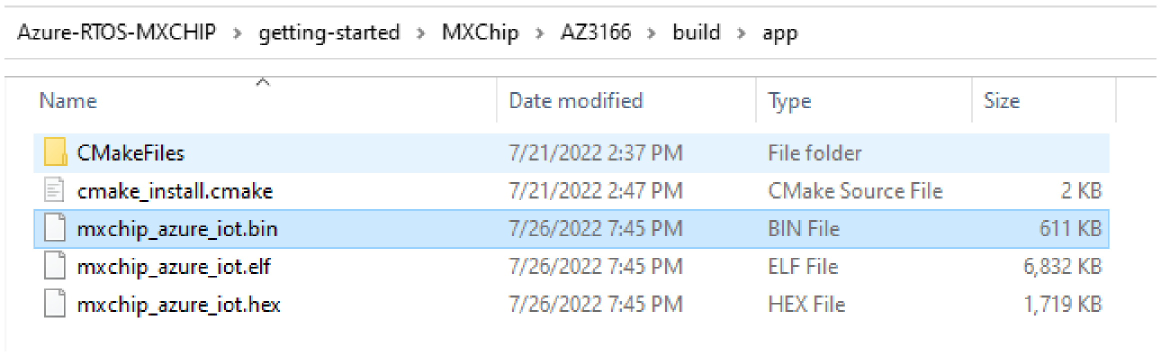


```
[build] [1201/1205] Linking C static library lib\netxduo\addons\azure_iot\azure_iot_security_module\iot-security-modul
[build] [1202/1205] Linking C static library lib\netxduo\addons\azure_iot\azure_iot_security_module\libiot_security_mo
[build] [1203/1205] Linking C static library lib\netxduo\libnetxduo.a
[build] [1204/1205] Linking C executable app\mxchip_azure_iot.elf
[build] Memory region         Used Size  Region Size  %age Used
[build]              RAM:      119328 B       128 KB     91.04%
[build]            FLASH:      625524 B         1 MB     59.65%
[build]           CCMRAM:          0 GB        64 KB      0.00%
[build] [1205/1205] cmd.exe /C "cd /D E:\Azure-RTOS-MXCHIP\getting-started\MXChip\AZ3166\build\app && "C:\Program File
-Obinary mxchip_azure_iot.elf mxchip_azure_iot.bin && "C:\Program Files (x86)\GNU Arm Embedded Toolchain\10 2021.10\bi
[build] Build finished with exit code 0
```

1.5 Program the MXCHIP IoT DevKit Board

With the mxchip_azure_iot.bin build, programming the board is a simple copy and paste.

1. Open File Explorer
2. Navigate to the \Azure-RTOS-MXCHIP\getting-started\MXChip\AZ3166\build\app folder. The newly created mxchip_azure_iot.bin file should be present.


Azure-RTOS-MXCHIP > getting-started > MXChip > AZ3166 > build > app

Name	Date modified	Type	Size
CMakeFiles	7/21/2022 2:37 PM	File folder	
cmake_install.cmake	7/21/2022 2:47 PM	CMake Source File	2 KB
mxchip_azure_iot.bin	7/26/2022 7:45 PM	BIN File	611 KB
mxchip_azure_iot.elf	7/26/2022 7:45 PM	ELF File	6,832 KB
mxchip_azure_iot.hex	7/26/2022 7:45 PM	HEX File	1,719 KB

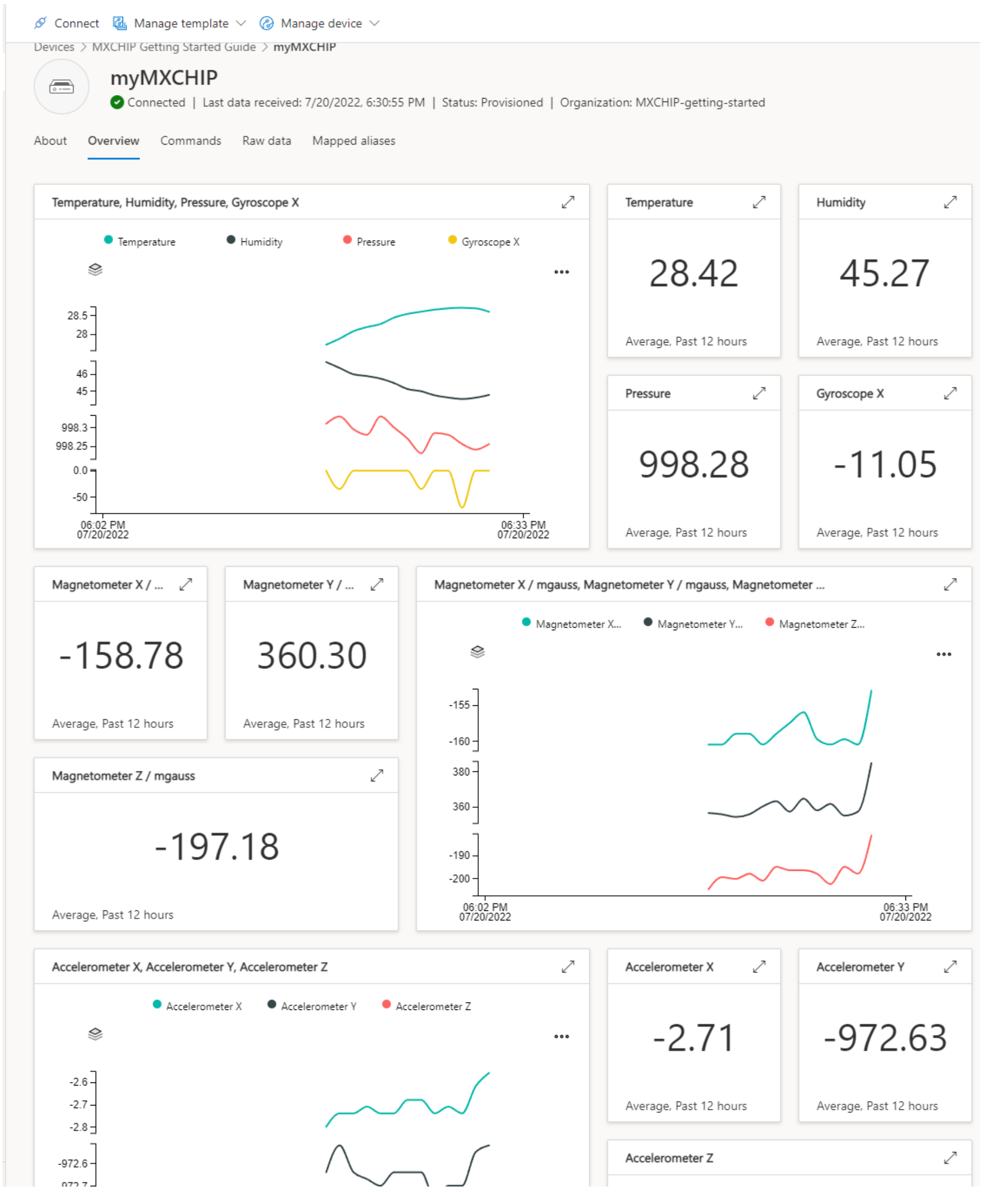
3. Connect the USB cable from the AZ3166 to your development computer.
4. Copy and paste the mxchip_azure_iot.bin into the <drive letter>AZ3166 folder. Programming starts automatically. The Red LED will be lit and go off when completed.
5. Open a serial terminal program and connect to the AZ3166 COM port, and set the baud rate to 115200. ABCOMTERM sets the baud rate to 115200 by default.
6. Hit the reset button of the AZ3166.

If all goes well, you will see the terminal output with something similar to the following:

```
Initializing Wi-Fi
       MAC address: C7:92:46:85:97:FB
SUCCESS: Wi-Fi initialized


Connecting Wi-Fi
       Connecting to SSID 'mywifi'
       Attempt 1...
SUCCESS: Wi-Fi connected

Initializing DHCP
       IP address: 192.168.1.25
       Mask: 255.255.255.0
       Gateway: 192.168.1.1
SUCCESS: DHCP initialized

Initializing DNS client
       DNS address: 192.168.1.1
SUCCESS: DNS client initialized

Initializing SNTP time sync
       SNTP server 0.pool.ntp.org
       SNTP time update: Jul 21, 2022 1:17:46.215 UTC
SUCCESS: SNTP initialized

Initializing Azure IoT DPS client

       DPS endpoint: global.azure-devices-provisioning.net
       DPS ID scope: 0ne006B73CD
       Registration ID: mymxchip
SUCCESS: Azure IoT DPS client initialized

Initializing Azure IoT Hub client
       Hub hostname: iotc-15b8066c-0815-48c5-9117-efc4e092b770.azure-devices.net
       Device id: mymxchip
       Model id: dtmi:azurertos:devkit:gsgmxchip;2
```

```
SUCCESS: Connected to IoT Hub

Receive properties: {"desired":{"$version":1},"reported":{"$version":1}}
Sending property:
$iothub/twin/PATCH/properties/reported/?$rid=3{"deviceInformation":{"__t":"c","manufact
urer":"MXCHIP","model":"AZ3166","swVersion":"1.0.0","osName":"Azure
RTOS","processorArchitecture":"Arm Cortex
M4","processorManufacturer":"STMicroelectronics","totalStorage":1024,"totalMemory":128}
}
Sending property: $iothub/twin/PATCH/properties/reported/?$rid=5{"ledState":false}
Sending property:
$iothub/twin/PATCH/properties/reported/?$rid=7{"telemetryInterval":{"ac":200,"av":1,"va
lue":10}}

Starting Main loop
Telemetry message sent: {"humidity":46.7,"temperature":27.62,"pressure":998.29}.
Telemetry message sent: {"magnetometerX":-162,"magnetometerY":360,"magnetometerZ":-
207}.
Telemetry message sent: {"accelerometerX":-2.8,"accelerometerY":-
972.64,"accelerometerZ":285.66}.
```

“Azure IoT” will appear on the little screen, and in the browser refresh the screen to see the myMXCHIP device data.

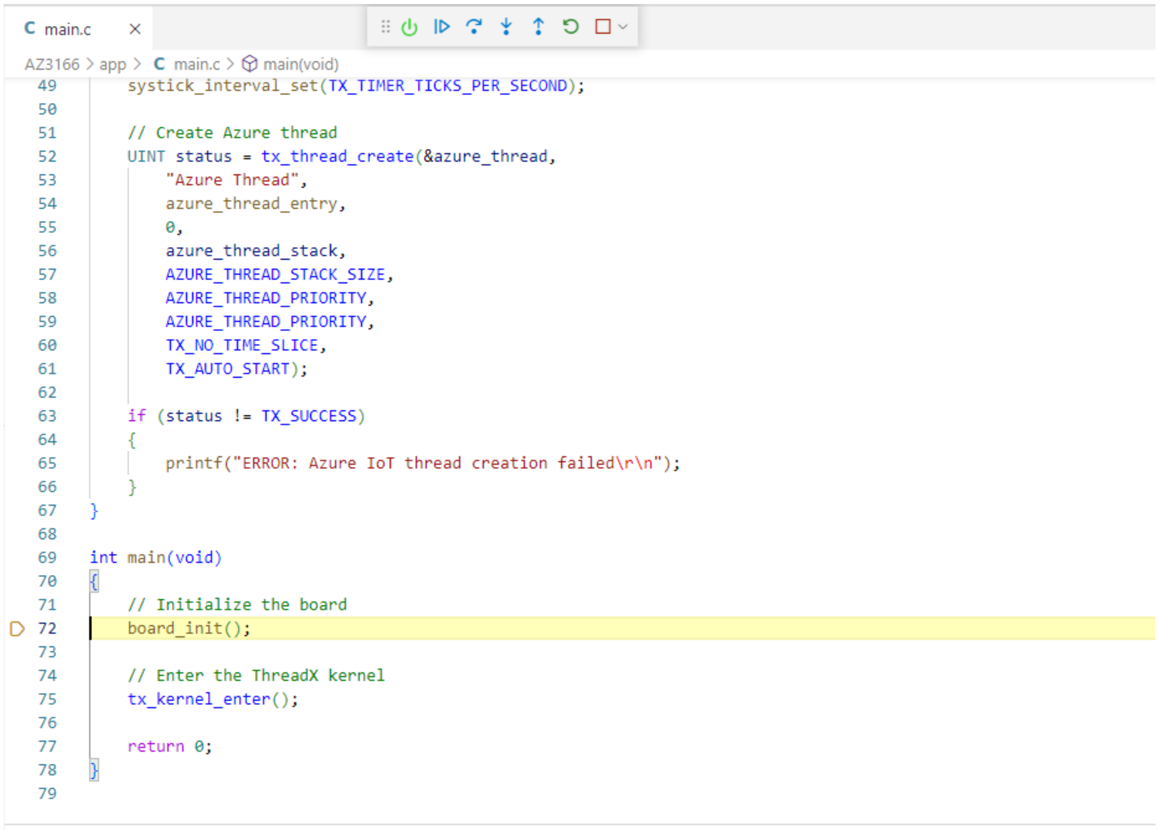

Connect
Manage template
Manage device
Devices > MXCHIP Getting Started Guide > myMXCHIP
myMXCHIP
Connected | Last data received: 7/20/2022, 6:30:55 PM | Status: Provisioned | Organization: MXCHIP-getting-started
About
Overview
Commands
Raw data
Mapped aliases
Temperature, Humidity, Pressure, Gyroscope X
Temperature
Humidity
Pressure
Gyroscope X
06:02 PM 07/20/2022
06:33 PM 07/20/2022
Magnetometer X / mgauss, Magnetometer Y / mgauss, Magnetometer ...
Accelerometer X, Accelerometer Y, Accelerometer Z
Accelerometer Z

Tile	Value	Period
Temperature	28.42	Average, Past 12 hours
Humidity	45.27	Average, Past 12 hours
Pressure	998.28	Average, Past 12 hours
Gyroscope X	-11.05	Average, Past 12 hours
Magnetometer X / ...	-158.78	Average, Past 12 hours
Magnetometer Y / ...	360.30	Average, Past 12 hours
Magnetometer Z / mgauss	-197.18	Average, Past 12 hours
Accelerometer X	-2.71	Average, Past 12 hours
Accelerometer Y	-972.63	Average, Past 12 hours

1.6 *Debugging the application*

Now, we will step through the code to see how it works.

1. In Visual Studio Code, hit F5.
2. The binary will be downloaded and a breakpoint will be hit within main.c.



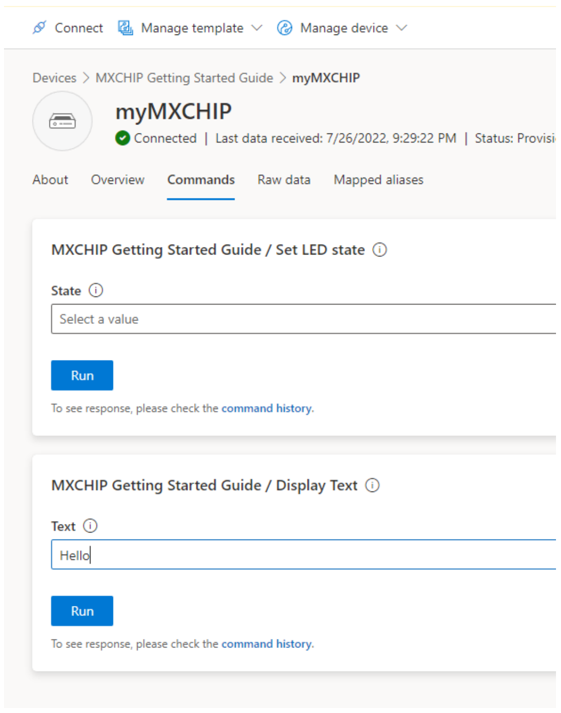
```c
    systick_interval_set(TX_TIMER_TICKS_PER_SECOND);

    // Create Azure thread
    UINT status = tx_thread_create(&azure_thread,
        "Azure Thread",
        azure_thread_entry,
        0,
        azure_thread_stack,
        AZURE_THREAD_STACK_SIZE,
        AZURE_THREAD_PRIORITY,
        AZURE_THREAD_PRIORITY,
        TX_NO_TIME_SLICE,
        TX_AUTO_START);

    if (status != TX_SUCCESS)
    {
        printf("ERROR: Azure IoT thread creation failed\r\n");
    }
}

int main(void)
{
    // Initialize the board
    board_init();

    // Enter the ThreadX kernel
    tx_kernel_enter();

    return 0;
}
```

3. Click Step Over (F10) to move past the board initialization call.
4. Click Step Over (F10) and the application thread will kick off and run.
5. Stop the debugger (Shift+F5).

The files comprise the core functionality of the application are:

- main.c – sets up and runs the thread.
- nx_client.c – creates the callback function to send telemetry and handle receive commands.
- Azure_iot_nx_client.c – this file has the main loop client_run(), which connects to Azure IoT Central and handles communications between the local application and the application on Azure IoT Central.

6. In main.c, set a breakpoint at line 40, which is the call to azure_iot_nx_client_entry.
7. In nx_client.c, set a breakpoint at line 374, which is within the call to azure_iot_nx_client_entry.
8. Also, in nx_client.c, set another breakpoint at line 260, which is the screen_printn call to handle the command to print the text to the screen.
9. In Azure_iot_nx_client.c, set a breakpoint at line 1116, which is in client_run().
10. Hit F5.
11. When the breakpoint hits in Main.c, hit F10 twice.
12. The debugger will break at line 40, Hit F11 to step into the to azure_iot_nx_client_entry call.
13. The debugger opens nx_client.c and hits the breakpoint at line 374.
14. Continue to hit F10, but at Line 414, hit F11 to step into azure_iot_nx_client_dps_run.
15. Continue to hit F10, and at line 1199 at the return, hit F11.
16. The debugger is now in the main loop in Azure_iot_nx_client.c. In Azure IoT Central, click on Command, type “Hello” in the Display Text, and click Run.



myMXCHIP

Connected | Last data received: 7/26/2022, 9:29:22 PM | Status: Provisi

MXCHIP Getting Started Guide / Set LED state

State: Select a value

To see response, please check the command history.

MXCHIP Getting Started Guide / Display Text

Text: Hello

To see response, please check the command history.

17. Go back and continue to hit F10, eventually, you should hit the breakpoint at line 260 in nx_client.c.
18. Hit F5 to continue debugging and the display should show the message.
19. Hit Shift+F5 to stop debugging.

1.7 Conclusion

Sample projects are good starting points to get familiar with the software. The ability to step through the code and see the API calls in operation provides good insight when documentation is lacking. The paper here covered debugging with Visual Studio Code.

References

More information on the Azure IoT SDKs can be found here.

MXCHIP Website: https://www.mxchip.com