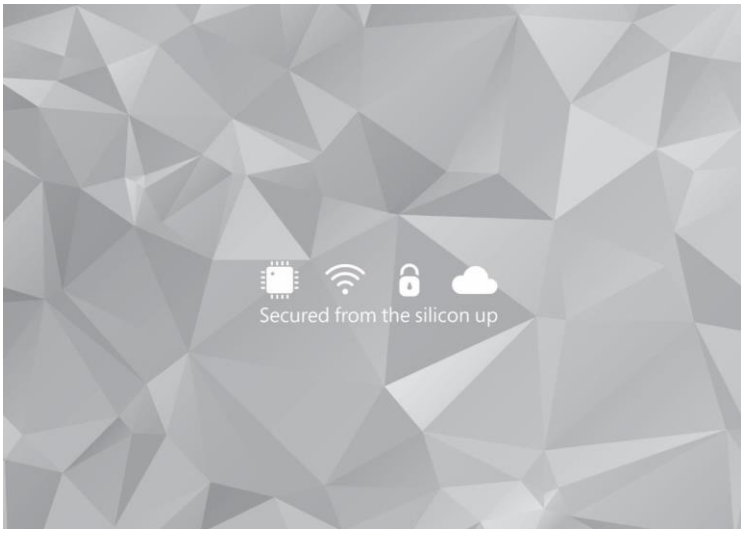


Azure Sphere Boot Camp

Lab: MCU to Mt3620 to Azure

Authors:

- Juergen Schwertl (jschwert@microsoft.com)
- Kevin Saye (kevinsay@microsoft.com)

Version: 1.0

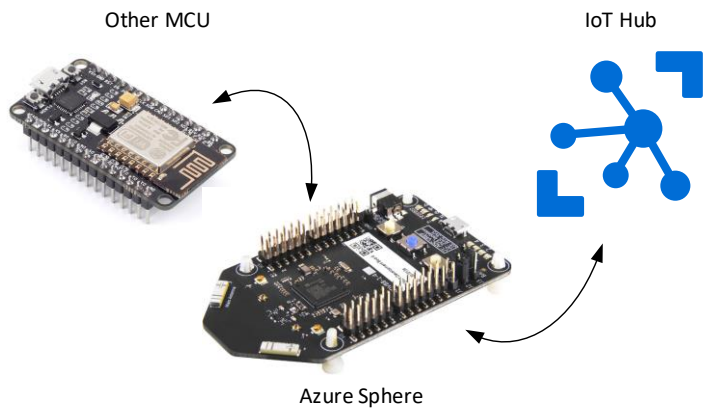
Date: 18 February 2019

CONTENTS

1 LAB OVERVIEW

This lab should be run after Lab 2 has been completed. I

In this lab, we will receive serial (UART) data from another device, convert it to JSON and Azure Sphere will securely send it to Azure IoT Hub. The diagram below illustrates the flow. NOTE, the MT3620 board is a 3.3v board so either use a 3.3v MCU or a Logic Level to convert the volts.



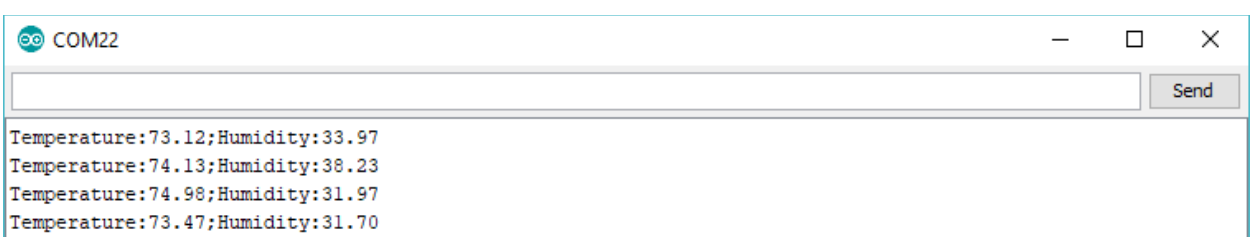
For this lab, an existing MCU is provided that already produces the serial data. For your own lab, the code is as follows:

```
float temp, humidity;
String message;

void setup() {
  Serial.begin(9600);
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  // turn LED on and output randomized temperature and humidity once per second
  digitalWrite(LED_BUILTIN, LOW);
  temp = random(7100, 7500);
  humidity = random(3000, 4000);
  message = "Temperature:";
  message += temp / 100;
  message += ";Humidity:";
  message += humidity / 100;
  Serial.println(message);
  delay(100);
  // turn LED off
  digitalWrite(LED_BUILTIN, HIGH);
  delay(900);
}
```

The output from the other MCU is as follows:



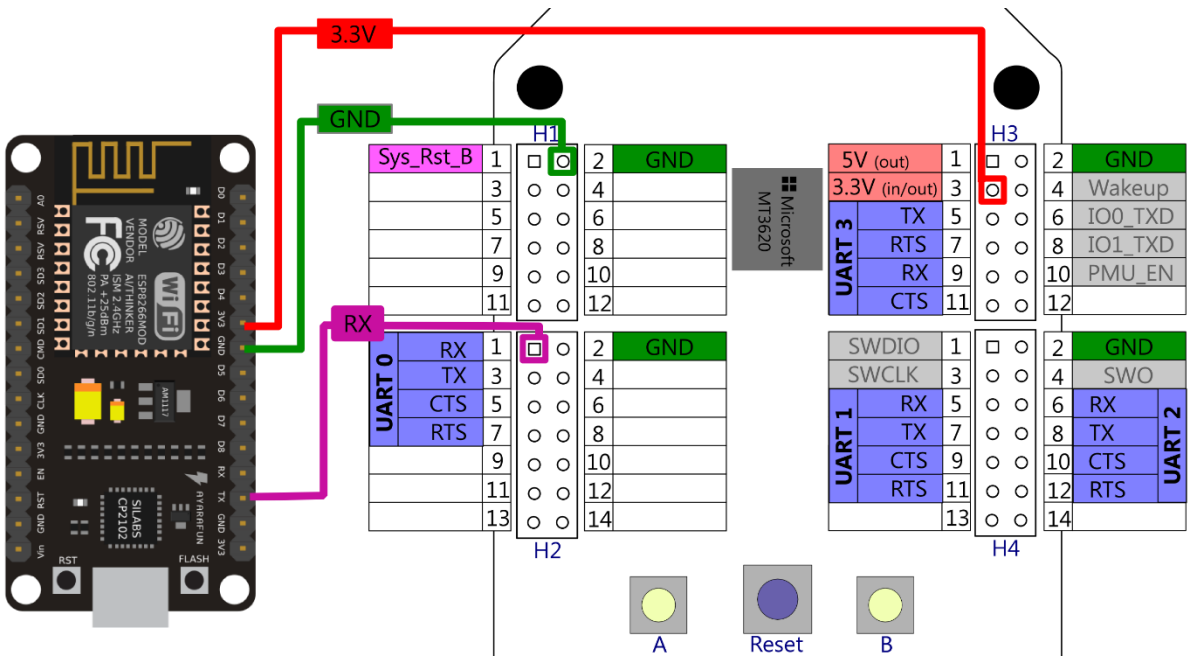
1.1 Wiring the Devices – for Node MCU

This step assumes you are using the smaller NodeMCUs. If using the Arduino R3, refer to section 1.2.

With the Sphere and the other MCU unplugged from power, wire the device as follows:

Purpose	MT3620	Other MCU
3.3V power for NodeMCU	3.3V (Header 3, pin 3)	3.3V (pin varies)
Common ground	GND (Header 1, pin 2)	Gnd (pin varies)
Transmit from MCU to Sphere	RX (Header 2, pin 1)	TX (pin varies)
Receive from Sphere to MCU	*TX (Header 2, pin 4)*	*RX (pin varies)*

Notice that the Transmit on the auxilary MCU is wired to the Receive on Azure Sphere. You can use Azure Sphere 3.3V power out and ground to power the auxiliary MCU.



For information on the pinout of the MT3620 board, see https://github.com/JuergenSchwertl/AzureSphereSamples .

1.2 Wiring the Devices – for Arduino Uno R3

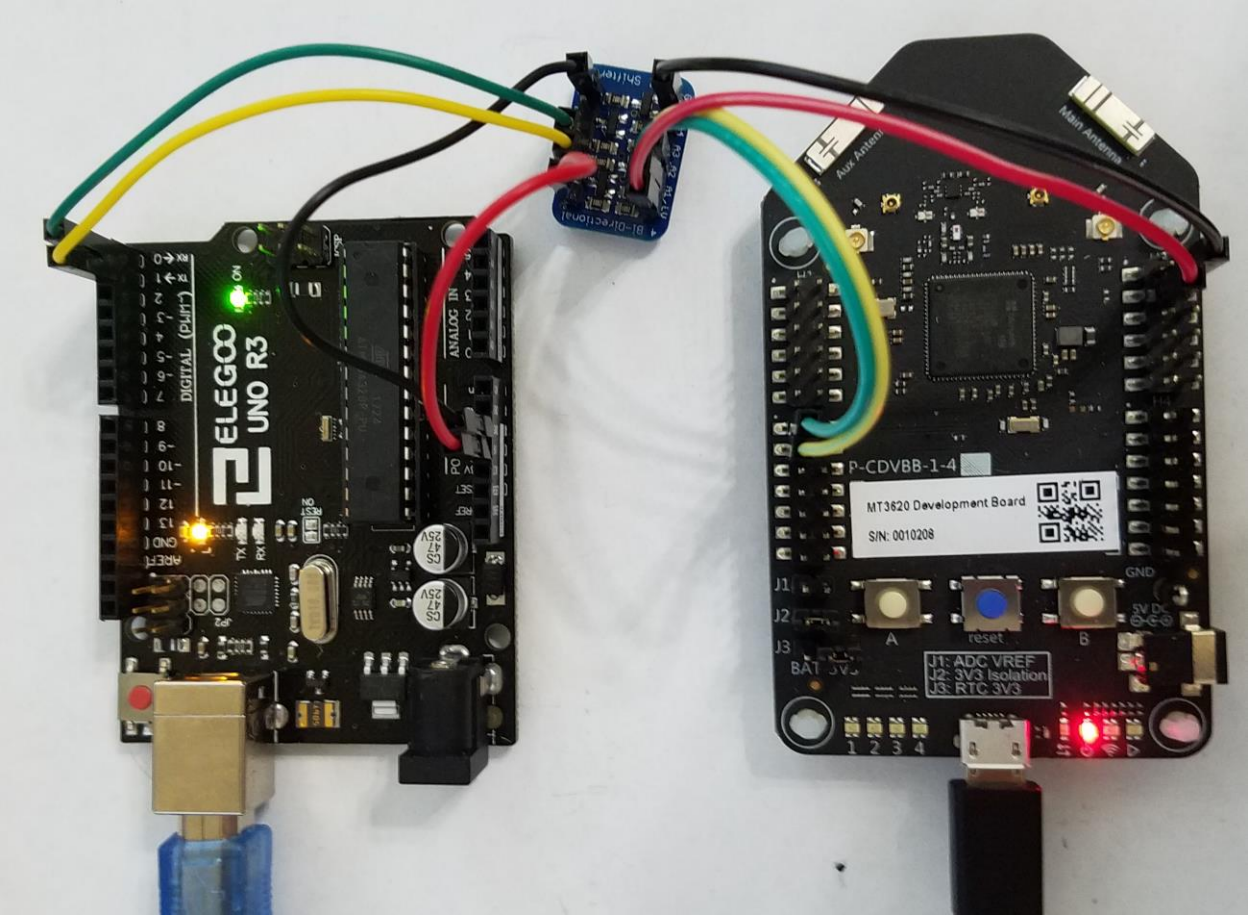
This section is for the Arduino Uno R3. If using the Node MCU, refer to section 1.1.

Because the Arduino Uno R3 is a 5.0 Volt board and the Azure Sphere is a 3.3 volt, we will use a Logic Level Shifter to convert voltages.

Wire the UNO to the Level Shifter to the MT3620 as shown below. The following table explains the wiring:

UNO	Wire Color	Level Shifter		Wire Color	MT 3620
Pin 0 (RX)	Green	B1	A1	Yellow	H2 Pin 3
Pin 1 (TX)	Yellow	B2	A2	Green	H2 Pin 1
GND	Black	GND	GND	Black	H3 Pin 2
5-volt power	Red	HV	LV	Red	H3 Pin 3

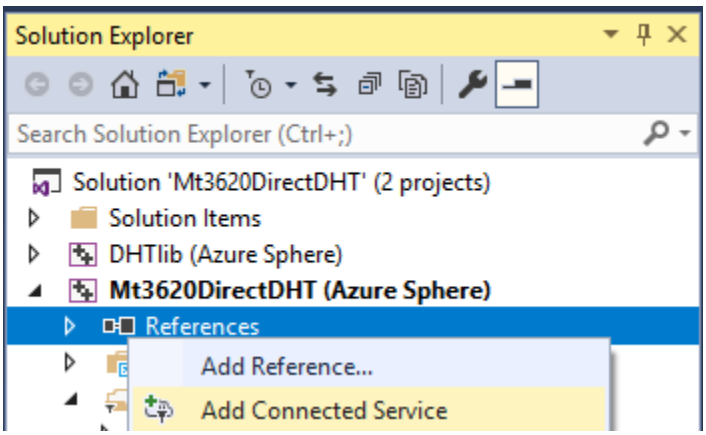
This picture shows the boards wired up:



1.3 Modifying the Code

Step 1. In Visual Studio, open `Mt3620DirectDHT\Mt3620DirectDHT.sln` from the zip file provided by the instructor.

Step 2. In the Solution Explorer, under the `Mt3620DirectDHT` solution, right click on Reference and "Add Connected Service" as shown below:



Select your Azure Subscription, Connection Type: "Device Provisioning Service" and your previously created Device provisioning service from the list and press [Add].
Make sure that the output shows updates to both `AllowedConnections` and `DeviceAuthentication` properties in app_manifest.json

```
[11.02.2019 18:03:35.153] Adding Device Connectivity with Azure IoT to the project.
[11.02.2019 18:03:35.329] The following hostnames have been added to the AllowedConnections
attribute of app_manifest.json: global.azure-devices-provisioning.net, JS-MS-Iot-Hub.azure-
devices.net
[11.02.2019 18:03:35.341] The Azure Sphere tenant ID 'c0b88764-9273-46ab-bab2-effecf13f91c'
has been added to the DeviceAuthentication attribute of app_manifest.json .
[11.02.2019 18:03:36.441] Azure Sphere Device Provisioning Service scope id:'0ne0002304B'
[11.02.2019 18:03:36.449] Successfully added Device Connectivity with Azure IoT to the
project.
```

Step 3. Open `azure_iot_utilities.h` on or about line **#41** and add the following code as shown below (you can copy these lines also from `azure_iot_utilities-snippets.txt`)

```
/// <summary>
///     Creates and enqueues reported properties state using a prepared json string.
///     The report is not actually sent immediately, but it is sent on the next
///     invocation of AzureIoT_DoPeriodicTasks().
/// </summary>
void AzureIoT_TwinReportStateJson(
    char *reportedPropertiesString,
    size_t reportedPropertiesSize);
```

```
azure_iot_utilities.h   azure_iot_utilities-snippets.txt
Mt3620DirectDHT                                          (Global Scope)
39    void AzureIoT_TwinReportState(const char *propertyName, size_t propertyValue);
40
41    /// <summary>
42    ///     Creates and enqueues reported properties state using a prepared json string.
43    ///     The report is not actually sent immediately, but it is sent on the next
44    ///     invocation of AzureIoT_DoPeriodicTasks().
45    /// </summary>
46    void AzureIoT_TwinReportStateJson(
47        char *reportedPropertiesString,
48        size_t reportedPropertiesSize);
49
```

Step 4. Open `azure_iot_utilities.c` and at the end of the file, on or about line **#463** add the following code, as shown below

```
void AzureIoT_TwinReportStateJson(
    char *reportedPropertiesString,
    size_t reportedPropertiesSize)
{
    if (iothub_client_handle == NULL) {
            LogMessage("ERROR: client not initialized\n");
    }
    else {
            if (reportedPropertiesString != NULL) {
                    if (IoTHubDeviceClient_LL_SendReportedState(iothub_client_handle,
                            (unsigned char *)reportedPropertiesString,
reportedPropertiesSize,
                            reportStatusCallback, 0) != IOTHUB_CLIENT_OK) {
                            LogMessage("ERROR: failed to set reported state as
'%s'.\n",
                                    reportedPropertiesString);
                    }
                    else {
                            LogMessage("INFO: Reported state as '%s'.\n",
reportedPropertiesString);
                    }
            }
            else {
                    LogMessage("ERROR: no JSON string for Device Twin reporting.\n");
            }
    }
}
```

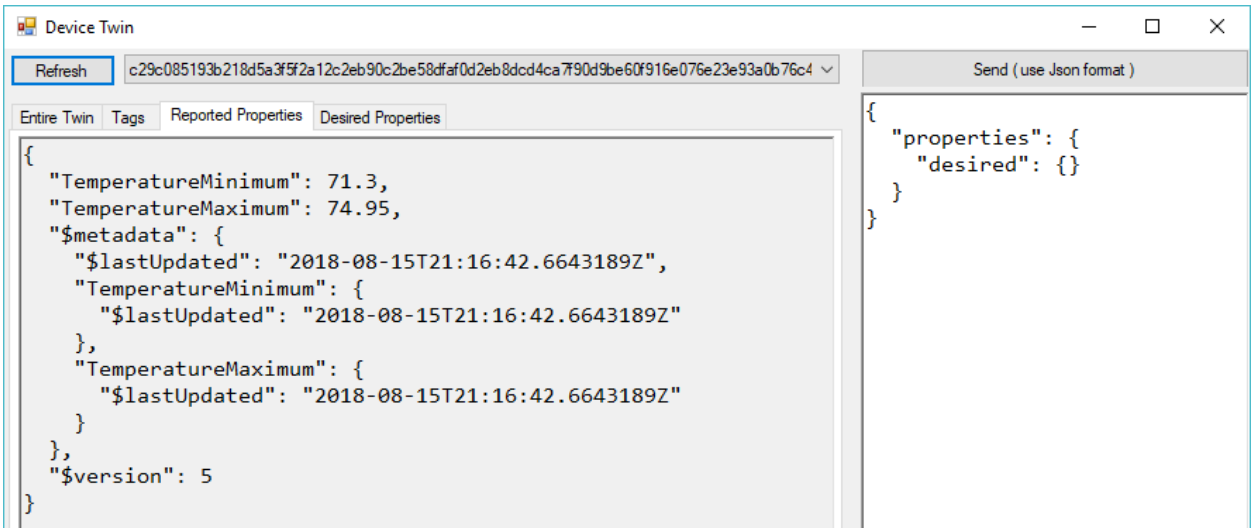
```c
/// <summary>
///     Creates and enqueues reported properties state using a prepared json string.
///     The report is not actually sent immediately, but it is sent on the next
///     invocation of AzureIoT_DoPeriodicTasks().
/// </summary>
void AzureIoT_TwinReportStateJson(
    char *reportedPropertiesString,
    size_t reportedPropertiesSize)
{
    if (iothubClientHandle == NULL) {
        LogMessage("ERROR: client not initialized\n");
    }
    else {
        if (reportedPropertiesString != NULL) {
            if (IoTHubDeviceClient_LL_SendReportedState(iothubClientHandle,
                (unsigned char *)reportedPropertiesString, reportedPropertiesSize,
                reportStatusCallback, 0) != IOTHUB_CLIENT_OK) {
                LogMessage("ERROR: failed to set reported state as '%s'.\n",
                    reportedPropertiesString);
            }
            else {
                LogMessage("INFO: Reported state as '%s'.\n", reportedPropertiesString);
            }
        }
        else {
            LogMessage("ERROR: no JSON string for Device Twin reporting.\n");
        }
    }
}
```

Step 5. In Visual Studio, click “Remote GDB Debugger” to compile, deploy, run and debug the code on the device.

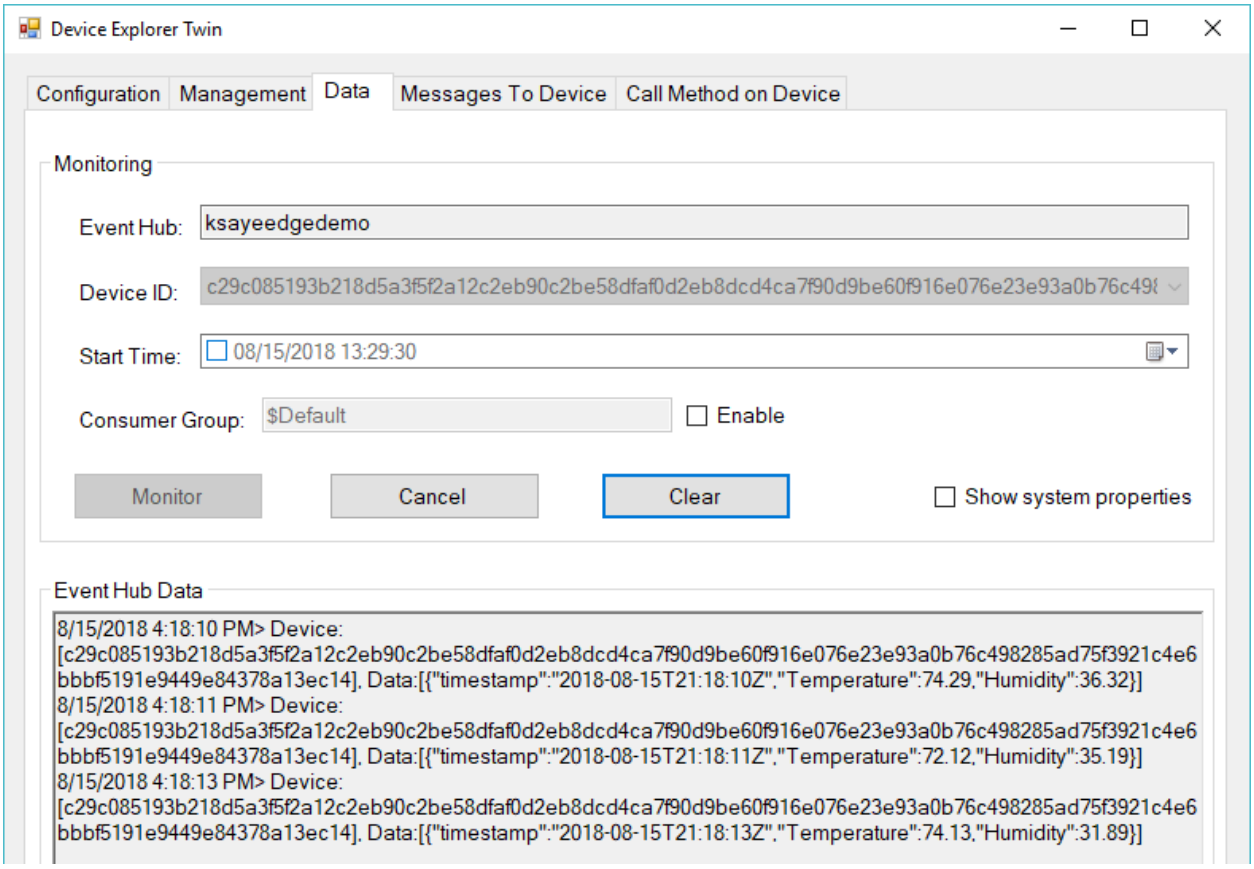
Step 6. Monitoring the output window in Visual Studio, you should see the device send the temperature every second as shown below. Note how we send telemetry for the Temperature and Humidity while updating a TWIN when the maximum and minimum temperature change.

```
Output
Show output from: Device Output
Remote debugging from host 192.168.35.1
MCUtoMt3620toAzure application starting
[Azure IoT] IoTHubDeviceClient_CreateWithAzureSphereDeviceAuthProvisioning returned 'AZURE_SPHERE_PROV_RESULT_OK'.
[Azure IoT Hub client] INFO: AzureIoT_DoPeriodicTasks calls in progress...
[UART] Received line: Temperature:71.58;Humidity:39.54
[MCU] Sending telemetry {"timestamp":"2018-08-15T21:14:29Z","Temperature":71.58,"Humidity":39.54}
[Azure IoT] INFO: IoTHubClient accepted the message for delivery
[MCU] Updating device twin: {"TemperatureMinimum":71.58,"TemperatureMaximum":71.58}
[Azure IoT] INFO: Reported state as '{"TemperatureMinimum":71.58,"TemperatureMaximum":71.58}'.
[Azure IoT] INFO: connection to the IoT Hub has been established (IOTHUB_CLIENT_CONNECTION_OK).
[UART] Received line: Temperature:72.67;Humidity:33.13
[MCU] Updating device twin: {"TemperatureMinimum":71.58,"TemperatureMaximum":72.67}
[Azure IoT] INFO: Reported state as '{"TemperatureMinimum":71.58,"TemperatureMaximum":72.67}'.
```

Step 7. Using Azure Device Explorer, viewing the TWIN properties you should see the min and max temperature received.



Step 8. Using Azure Device Explorer, monitoring the data, you should see both temperature and humidity sent as a JSON message.



Step 9. Unique to this lab, we enabled the Uart `ISU0` in the `app_manifest.json` as shown below:

1.4 Reviewing the Code (main.c)

Lines 19 - 20 includes the UART and MCU utilities, not part of the Azure Sphere SDK.

```
16
17  #include "mt3620_rdb.h"
18  #include "azure_iot_utilities.h"
19  #include "UART_utilities.h"
20  #include "MCU_utilities.h"
```

Lines 125 - 134 verify the connectivity and permission to the UART and set a handler.

```
123     sigaction(SIGTERM, &action, NULL);
124
125     epollFd = CreateEpollFd();
126     if (epollFd < 0) {
127         return -1;
128     }
129
130     // Create a UART_Config object, open the UART and set up UART event handler
131     if ((uartFd = UART_InitializeAndAddToEpoll(MT3620_RDB_HEADER2_ISU0_UART, epollFd, &MCU_ParseDataToIotHub)) < 0)
132     {
133         return -1;
134     }
```

Lines 189 – 191 call the event handler.

```
188
189         if (WaitForEventAndCallHandler(epollFd) != 0) {
190             terminationRequired = true;
191         }
192
193         // AzureIoT_DoPeriodicTasks() needs to be called frequently in order to keep active
194         // the flow of data with the Azure IoT Hub
195         AzureIoT_DoPeriodicTasks();
```

1.5 Reviewing the Code (UART_utilities.c)

Lines 113 – 140 initialize and set the settings for the UART.

```
UART_utilities.c   MCU_utilities.c   main.c   app_manifest.json   azure_iot_utilities.c   azure_iot_utilities.h
MCUtoMT3620toAzure                                                    (Global Scope)
112   ///<returns>UART file decriptor or -1 on error</returns>
113   int UART_InitializeAndAddToEpoll(UART_Id uartId, int epollFd, uart_line_received_handler_t handleLineReceived)
114   {
115       m_UartId = uartId;
116       m_HandleLineReceived = handleLineReceived;
117       // Create a UART_Config object, open the UART and set up UART event handler
118       UART_Config uartConfig;
119       UART_InitConfig(&uartConfig);
120       uartConfig.baudRate = 9600;
121       uartConfig.flowControl = UART_FlowControl_None;
122       m_UartFd = UART_Open(uartId, &uartConfig);
123       if (m_UartFd < 0) {
124           Log_Debug("ERROR: Could not open UART: %s (%d).\n", strerror(errno), errno);
125           return -1;
126       }
127
128       struct epoll_event eventToAdd;
129       eventToAdd.data.ptr = &handleUartEvent;
130       eventToAdd.events = EPOLLIN;
131
132       // Register the UART file descriptor on the epoll instance referred by epollFd
133       // and register the eventHandler handler for events in epollEventMask
134       if (epoll_ctl(epollFd, EPOLL_CTL_ADD, m_UartFd, &eventToAdd) == -1) {
135           Log_Debug("ERROR: Could not add event to epoll instance %s (%d)\n", strerror(errno), errno);
136           return -1;
137       }
138
139       return m_UartFd;
140   }
```

Lines 50 reads the data from the UART.

```
UART_utilities.c   MCU_utilities.c   main.c   app_manifest.json   azure_iot_utilities.c   azure_iot_utilities.h
MCUtoMT3620toAzure                                                    (Global Scope)
43    /// </summary>
44    static void handleUartEvent()
45    {
46        char *pszLine = NULL;
47        char *pchSegment = &receiveBuffer[nBytesInBuffer];
48
49        // Poll the UART and store the byte(s) behind already received bytes
50        ssize_t nBytesRead = read(m_UartFd, (void *)pchSegment, RECEIVE_BUFFER_SIZE - nBytesInBuffer);
51
```

You may continue reviewing both `UART_utilities.h/UART_utilities.c` and `MCU_utilities.h/ MCU_utilities.c` as time allows.