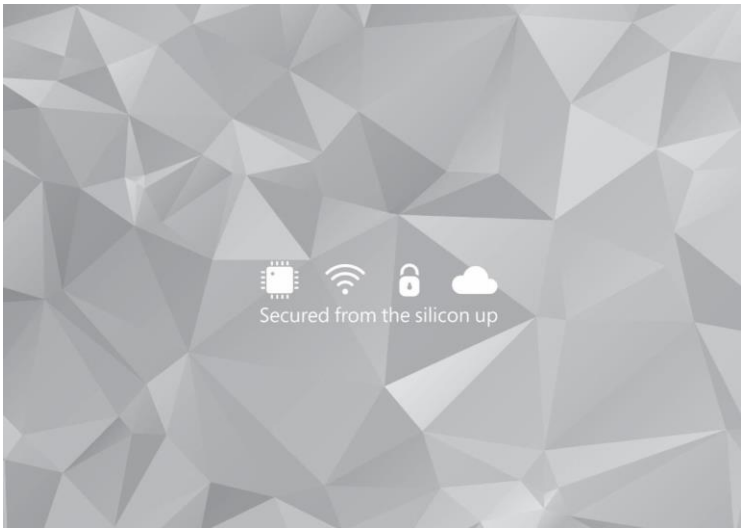


Azure Sphere Boot Camp

Lab MT3620DirectDHT

Authors:

- Kevin Saye (kevinsay@microsoft.com)

Version: 1.0

Date: 14 November 2018

CONTENTS

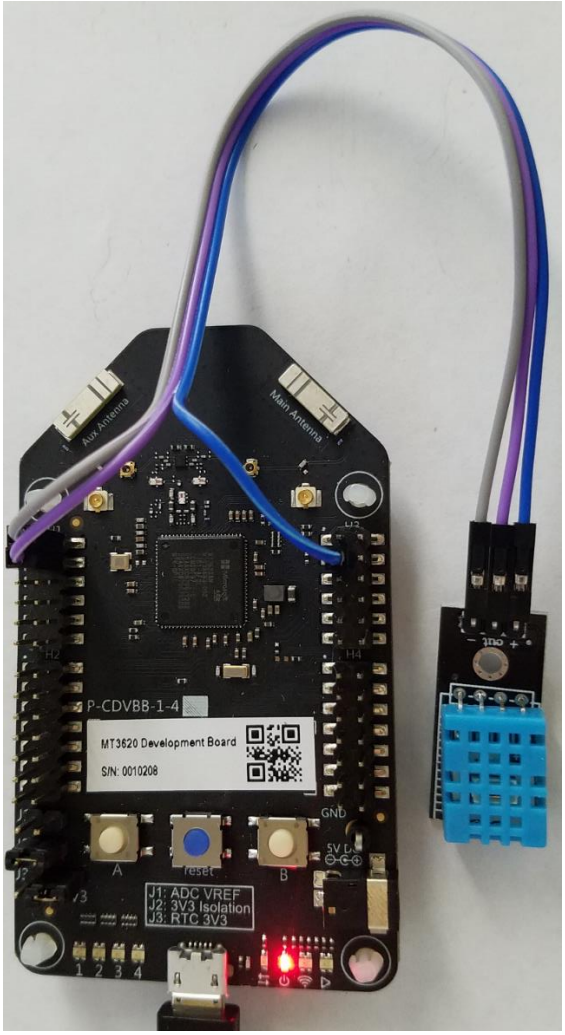
1 Lab Overview

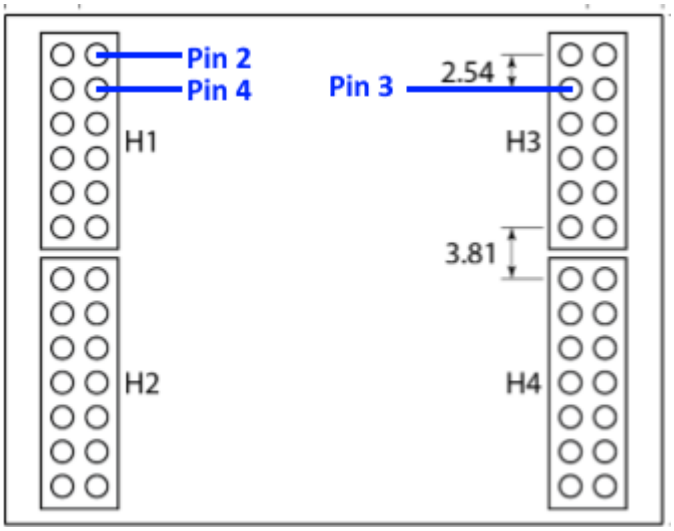
In this lab, we will connect a DHT11 sensor to an Azure Sphere device, which will send in JSON form a message to Azure IoT Hub on a periodic basis or when button B is pressed.

1.1 Wiring the Device

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

Purpose	MT3620	DHT11/22	Pictured wire below
Ground	Header 1, pin 2	-	grey
Data	Header 1, pin 4	out	purple
3.3 volts	Header 3, pin 3	+	blue



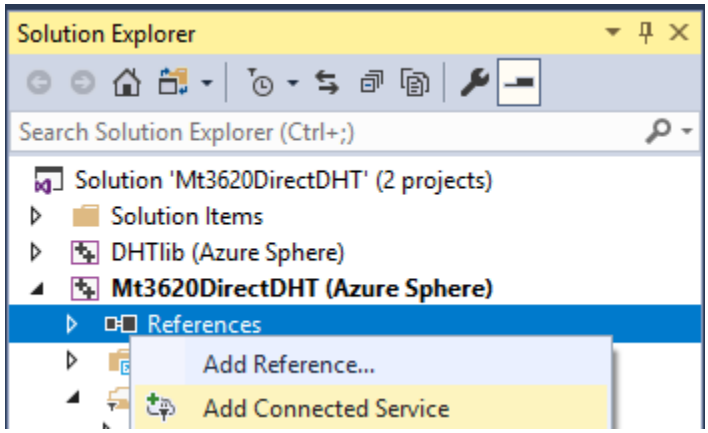


For information on the pinout of the board, see MT3620ReferenceBoardDesignTP4.0.1.pdf.

1.2 Modifying the Code

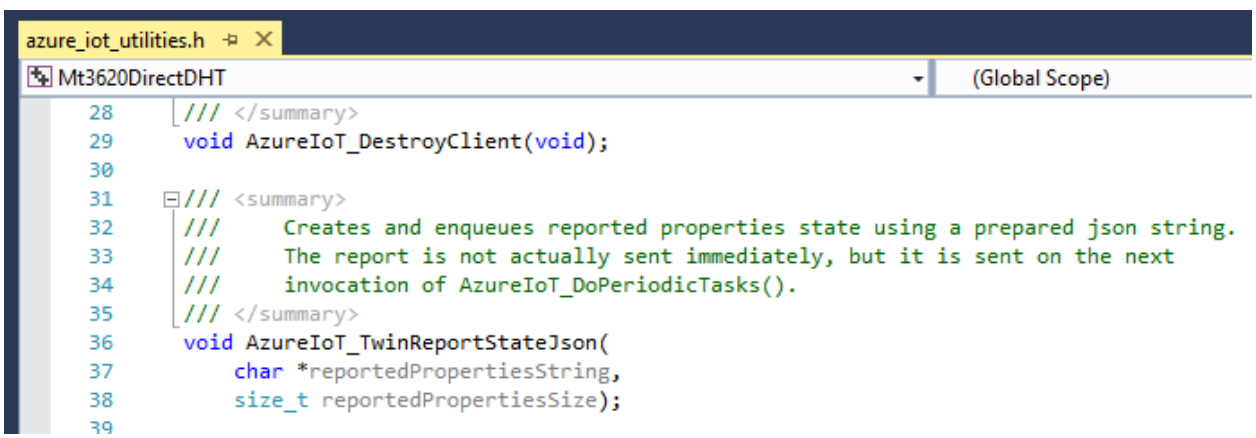
Step 1. In Visual Studio, open `Mt3620DirectDHT\Mt3620DirectDHT.sln` from the zip file provided by the instructor.

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



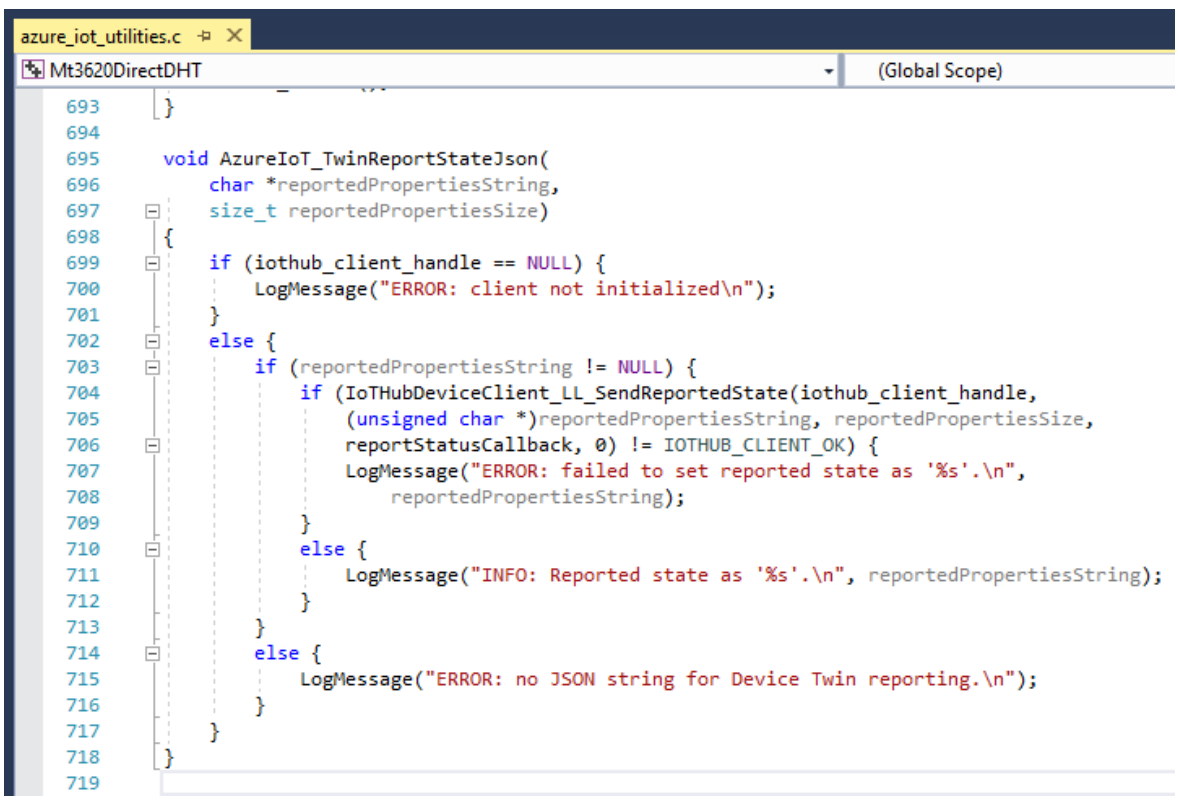
Step 3. Open `azure_iot_utilities.h` on or about line #31 and add the following code as shown below

```
/// <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);
```



Step 4. Open `azure_iot_utilities.c` and at the end of the file, on or about line 695 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");
            }
    }
}
```



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 15 seconds as shown below:

```
Output
Show output from: Device Output
Remote debugging from host 192.168.35.1
MT3620 direct DHT sensor application starting
Open MT3620_RDB_BUTTON_B
Open RGB LED 0
Open RGB LED 1
Open RGB LED 2
[Azure IoT] IoTHubDeviceClient_CreateWithAzureSphereDeviceAuthProvisioning returned 'AZURE_SPHERE_PROV_RESULT_OK'.
[Azure IoT Hub client] INFO: AzureIoT_DoPeriodicTasks calls in progress...
[Azure IoT] INFO: connection to the IoT Hub has been established (IOTHUB_CLIENT_CONNECTION_OK).
[Azure IoT Hub client] INFO: AzureIoT_DoPeriodicTasks calls in progress...
[Azure IoT Hub client] INFO: AzureIoT_DoPeriodicTasks calls in progress...
DHT_ReadData() Humidity = 40.0 % Temperature = 26.0 *C (78.8 *F)
[Azure IoT] INFO: Reported state as '{"Temp_C":"26.00","Temp_F":"78.80","Humidity":"40.00"}'.
[Azure IoT] INFO: IoTHubClient accepted the message for delivery
INFO: SendMessageToIoTHub {"success":true,"Temp_C":"26.00","Temp_F":"78.80","Humidity":"40.00"}
[Azure IoT] INFO: Reported state accepted by IoT Hub. Result is: 204
[Azure IoT] INFO: Message received by IoT Hub. Result is: 0
```

Step 7. Pressing the B button should send the temperature instantly.

Note, using an inexpensive sensor like the DHT11 has limited accuracy and stability.

1.3 Reviewing the Code (main.c)

Line 18 includes the DHT Library, not part of the Azure Sphere SDK.

```
17  #include "led_blink_utility.h"
18  #include "..\DHTlib\Inc\Public\DHTlib.h"
19
20  #include "azure_iot_utilities.h"
```

Lines 74 – 80 define message format and temperature reading intervals.

```
73  // json format string for reported properties
74  static const char cstrReportedPropertiesJson[] = "{\"Temp_C\":\"%.2f\",\"Temp_F\":\"%.2f\",\"Humidity\":\"%.2f\"}";
75  static const char cstrJsonErrorNoData[] = "{ \"success\" : false, \"message\" : \"could not read DHT sensor data\" }";
76  static const char noMethodFound[] = "\"method not found '%s'\"";
77  static const char cstrJsonSuccessAndData[] = "{\"success\":true,\"Temp_C\":\"%.2f\",\"Temp_F\":\"%.2f\",\"Humidity\":\"%.2f\"}";
78
79  // how often we automatically send the temperature.
80  static int sendTempIntervalSeconds = 15;
```

Lines 121 – 146 read the sensor data from the hard coded GPIO (GPIO0).

```
121 bool GetAndReportSensorData(char * jsonBuffer, size_t jsonBufferSize )
122 {
123     DHT_SensorData * pDHT = DHT_ReadData(MT3620_GPIO0);
124     if (pDHT != NULL)
125     {
126
127         char *jsonPropertyBuffer = (char *)malloc(JSON_BUFFER_SIZE);
128         if (jsonPropertyBuffer != NULL)
129         {
130             snprintf(jsonPropertyBuffer, JSON_BUFFER_SIZE, cstrReportedPropertiesJson, pDHT->TemperatureCelsius, pDHT->TemperatureFahrenheit, pDHT->Humidity);
131             AzureIoT_TwinReportStateJson(jsonPropertyBuffer, strlen(jsonPropertyBuffer));
132             free(jsonPropertyBuffer);
133         }
134         else {
135             Log_Debug("ERROR: failed to allocate buffer for reported state.\n");
136         }
137
138
139         if (jsonBuffer != NULL) {
140             // prepare data to be sent via AzureIoT_SendMessage
141             snprintf(jsonBuffer, jsonBufferSize, cstrJsonSuccessAndData, pDHT->TemperatureCelsius, pDHT->TemperatureFahrenheit, pDHT->Humidity);
142         }
143         return true;
144     }
145     return false;
146 }
```

Lines 154 – 158 allocate a jsonBuffer and populate the buffer with the DHT sensor data to send to Azure IoT Hub.

```
148 /// <summary>
149 ///     Sends a message to the IoT Hub.
150 /// </summary>
151 static void SendMessageToIotHub()
152 {
153     if (connectedToIoTHub) {
154         char * jsonBuffer = (char *)malloc(JSON_BUFFER_SIZE);
155         if (GetAndReportSensorData(jsonBuffer, JSON_BUFFER_SIZE)) {
156
157             // Send a message
158             AzureIoT_SendMessage(jsonBuffer);
159             Log_Debug("INFO: SendMessageToIoTHub %s\n", jsonBuffer);
160             // Set the send/receive LED to blink once immediately to indicate the message has been queued
161             LedBlinkUtility_BlinkNow(&ledMessageSent, LedBlinkUtility_Colors_Green);
162         }
163     } else {
164         // Send/receive LED to blink once red to indicate sensor failure
165         LedBlinkUtility_BlinkNow(&ledMessageSent, LedBlinkUtility_Colors_Red);
166         Log_Debug("WARNING: Cannot send message: not connected to the IoT Hub\n");
167     }
168 }
```

On or about line 389 we see that that we check to see if the button is pressed.

Lines 395 – 401 check the sendTempIntervalSeconds for our periodic sending of messages.

```
385    // AzureIoT_DoPeriodicTasks() needs to be called frequently in order to keep active
386    // the flow of data with the Azure IoT Hub
387    AzureIoT_DoPeriodicTasks();
388
389    if (CheckForButtonPresses() != 0) {
390        break;
391    }
392
393    // once we are connected and we have the correct time
394    // we send the temperature every sendTempIntervalSeconds
395    if (connectedToIoTHub) {
396        time_t now = time(0);
397        if (lastSentTemperature < now - sendTempIntervalSeconds) {
398            SendMessageToIotHub();
399            lastSentTemperature = time(0);
400        }
401    }
```