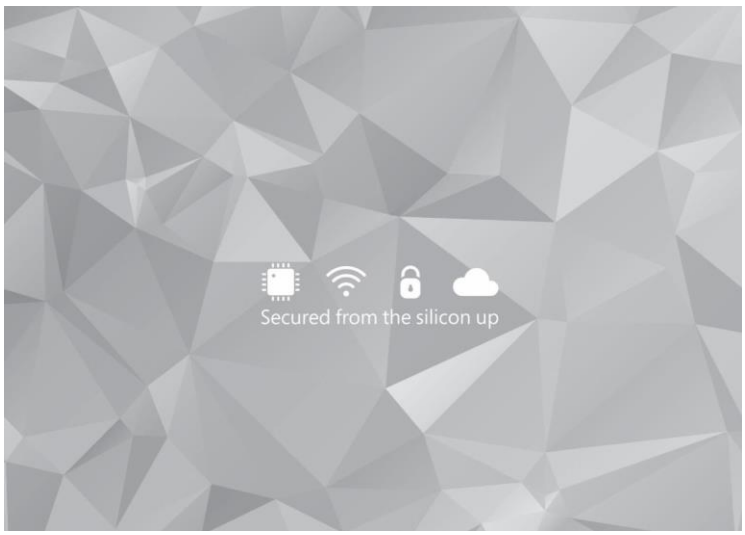


Azure Sphere Boot Camp

Lab 4

Authors:

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

Version: 1.0

Date: 15 February 2019

Microsoft

CONTENTS

© 2018 Microsoft – Microsoft

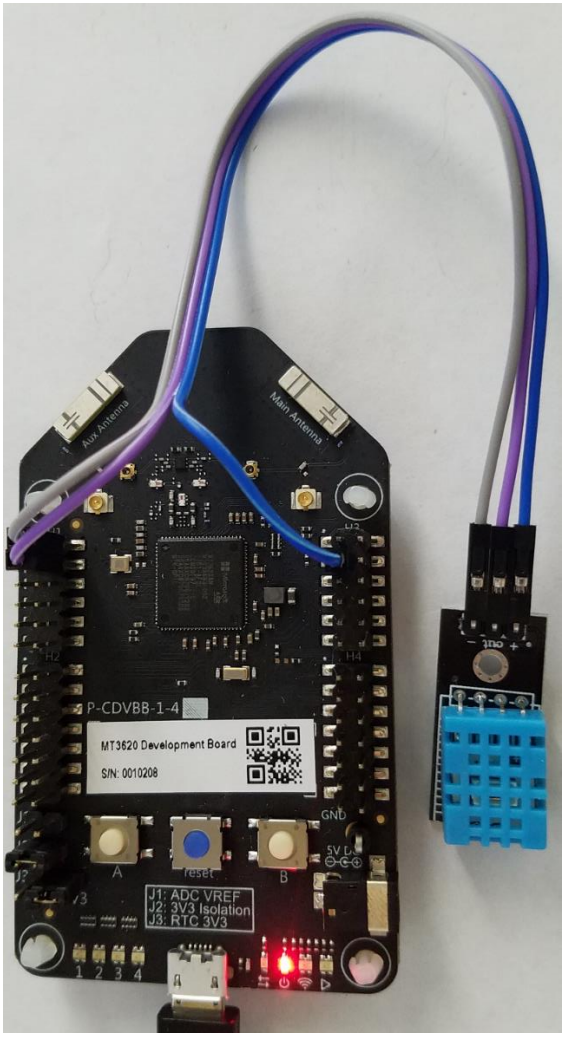
1 Lab Overview

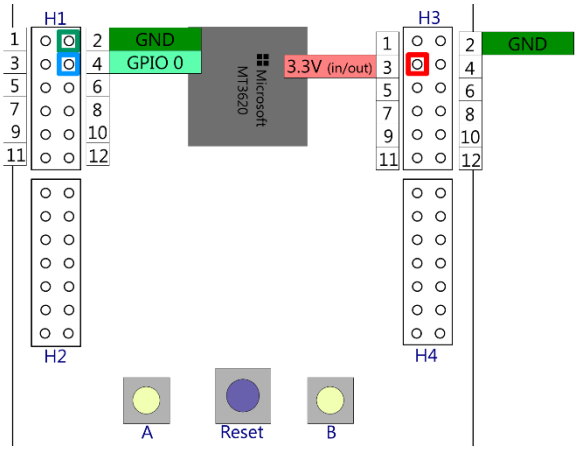
In this lab, we will connect a DHT11 sensor to an Azure Sphere device, which will send in JSON form a message to Azure IoT Hub on a periodic basis or when button B is pressed.

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





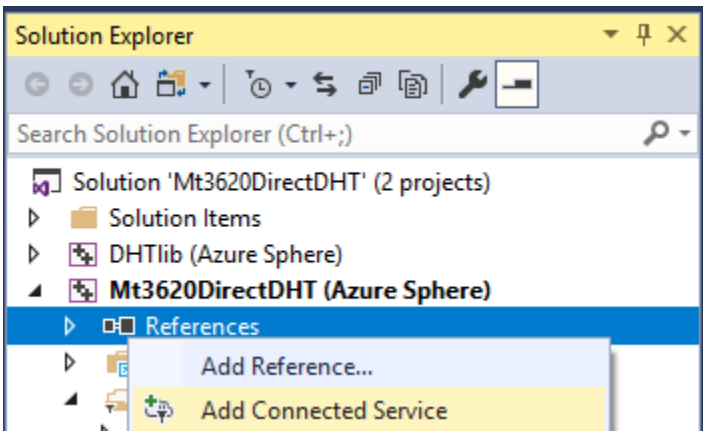
For information on the pinout of the board, see MT3620ReferenceBoardDesignTP4.0.1.pdf.

© 2018 Microsoft – Microsoft

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:



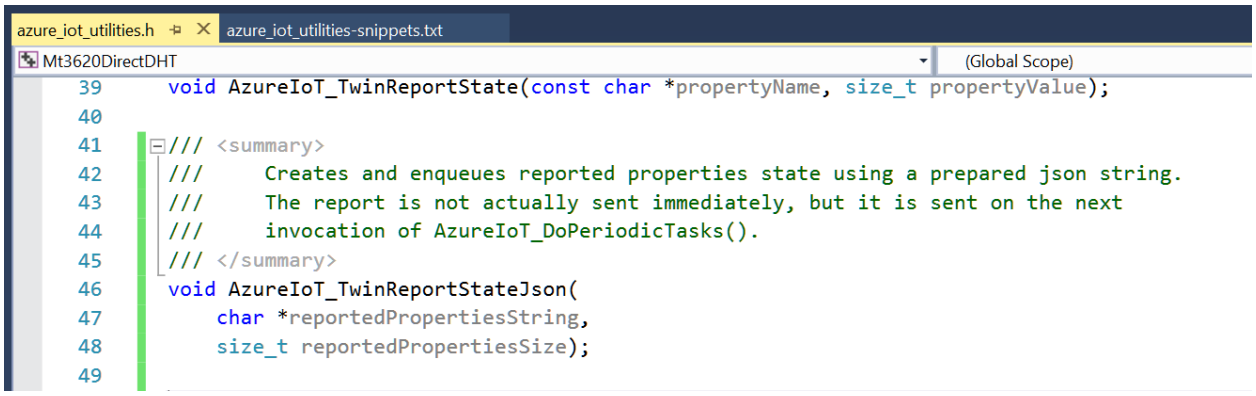
Select your Azure Subscription, Connection Type: "Device Provisioning Service" and your previously created Device provisioning service from the list and press [Add].
Make sure that the output shows updates to both `AllowedConnections` and `DeviceAuthentication` properties in app_manifest.json

```
[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);
```


© 2018 Microsoft – Microsoft



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");
            }
    }
}
```


© 2018 Microsoft – Microsoft

azure_iot_utilities.c | azure_iot_utilities-snippets.txt

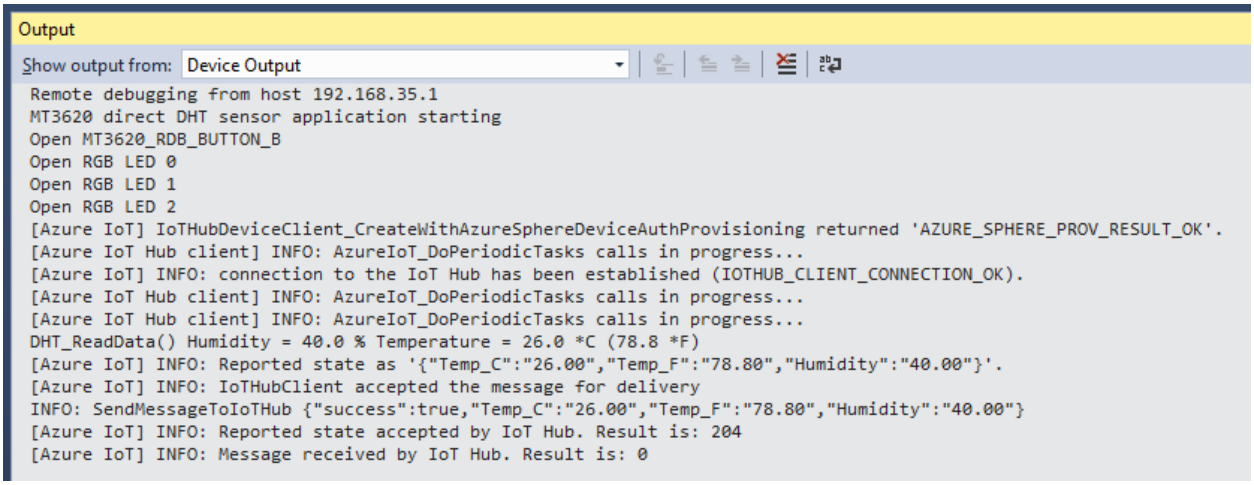
Mt3620DirectDHT | (Global Scope)

```
463  /// <summary>
464  ///     Creates and enqueues reported properties state using a prepared json string.
465  ///     The report is not actually sent immediately, but it is sent on the next
466  ///     invocation of AzureIoT_DoPeriodicTasks().
467  /// </summary>
468  void AzureIoT_TwinReportStateJson(
469      char *reportedPropertiesString,
470      size_t reportedPropertiesSize)
471  {
472      if (iothubClientHandle == NULL) {
473          LogMessage("ERROR: client not initialized\n");
474      }
475      else {
476          if (reportedPropertiesString != NULL) {
477              if (IoTHubDeviceClient_LL_SendReportedState(iothubClientHandle,
478                  (unsigned char *)reportedPropertiesString, reportedPropertiesSize,
479                  reportStatusCallback, 0) != IOTHUB_CLIENT_OK) {
480                  LogMessage("ERROR: failed to set reported state as '%s'.\n",
481                      reportedPropertiesString);
482              }
483              else {
484                  LogMessage("INFO: Reported state as '%s'.\n", reportedPropertiesString);
485              }
486          }
487          else {
488              LogMessage("ERROR: no JSON string for Device Twin reporting.\n");
489          }
490      }
491  }
```


© 2018 Microsoft – Microsoft

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:



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

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

© 2018 Microsoft – Microsoft

1.3 Reviewing the Code (main.c)

Line 20 includes the DHT Library (not part of the Azure Sphere SDK).

Line 22ff are now commented out as the *Add Connected Service* wizard added a definition for `AZURE_IOT_HUB_CONFIGURED` to the project settings (RightClick Mt3620DirectDHT-Project->Properties and go to Configuration Properties->C/C++->All Options and check "Additional Options" to contain "-D AZURE_IOT_HUB_CONFIGURED".

```
18  #include "mt3620_rdb.h"
19  #include "rgbled_utility.h"
20  #include "..\DHTlib\Inc\Public\DHTlib.h"
21
22  #ifndef AZURE_IOT_HUB_CONFIGURED
23  #error \
24      "WARNING: Please add a project reference to the Connected Service first \
25  (right-click References -> Add Connected Service)."
26  #endif
```

Lines 100ff define message format and temperature reading intervals.

```
105 // json format strings
106 static const char cstrJsonData[] = "{\"Temp_C\":\"%.2f\",\"Temp_F\":\"%.2f\",\"Humidity\":\"%.2f\"}";
107 static const char cstrJsonSuccessAndData[] = "{\"success\":true,\"Temp_C\":\"%.2f\",\"Temp_F\":\"%.2f\",\"Humidity\":\"%.2f\"}";
108 static const char cstrJsonErrorNoData[] = "{\"success\":false,\"message\":\"could not read DHT sensor data\"}";
109 static const char cstrJsonMethodNotFound[] = "{\"success\":false,\"message\":\"method not found '%s'\"}";
```

Lines 176ff reads the sensor data from the hard coded GPIO (GPIO0) and converts to json format.

```
176 /// <summary>
177 ///     Helper function to read the DHT sensor values and create response json if jsonBuffer and cstrJsonFormat is available.
178 /// </summary>
179 /// <param name="jsonBuffer">pointer to string buffer for json result.</param>
180 /// <param name="jsonBufferSize">length of pre-allocated json string buffer</param>
181 /// <returns>True if successful, false if an error occurred.</returns>
182 bool GetSensorDataJson(char * jsonBuffer, size_t jsonBufferSize, const char * cstrJsonFormat )
183 {
184     if ((jsonBuffer == NULL) || (cstrJsonFormat==NULL))
185     {
186         return false;
187     }
188
189     DHT_SensorData * pDHT = DHT_ReadData(MT3620_GPIO0);
190     if (pDHT == NULL)
191     {
192         strncpy(jsonBuffer, cstrJsonErrorNoData, jsonBufferSize);
193         return false;
194     }
195
196     // prepare json data to be sent
197     snprintf(jsonBuffer, jsonBufferSize, cstrJsonFormat,
198                 pDHT->TemperatureCelsius, pDHT->TemperatureFahrenheit, pDHT->Humidity);
199     return true;
200 }
```

Lines 202ff allocate a jsonBuffer and populate the buffer with the DHT sensor data to send the telemetry to Azure IoT Hub.

© 2018 Microsoft – Microsoft

```
202 /// <summary>
203 ///     Sends a message to Azure IoT Hub.
204 /// </summary>
205 static void SendMessageToIotHub(void)
206 {
207     if (connectedToIoTHub) {
208         char * jsonBuffer = (char *)malloc(JSON_BUFFER_SIZE);
209         if (GetSensorDataJson(jsonBuffer, JSON_BUFFER_SIZE, cstrJsonSuccessAndData)) {
210 
211             // Send a message
212             AzureIoT_SendMessage(jsonBuffer);
213             Log_Debug("INFO: SendMessageToIoTHub %s\n", jsonBuffer);
214             // Set the send/receive LED to blink once immediately to indicate the message has been queued
215             BlinkLedOnce(&ledSendMessage, timerFdSendMessageLed, RgbLedUtility_Colors_Green);
216         }
217         else
218         {
219             // Send/receive LED to blink once red to indicate sensor failure
220             BlinkLedOnce(&ledSendMessage, timerFdSendMessageLed, RgbLedUtility_Colors_Red);
221         }
222         free(jsonBuffer);
223     } else {
224         Log_Debug("[SendMessageToIoTHub]: Cannot send message: not connected to the IoT Hub\n");
225     }
226 }
```

An interesting change you'll find to the `event_data_t` declaration in `epoll_timerfd_utilities.h` at line #30. It has been extended to contain an additional `void * ptr` for additional event context.

```
15 /// <summary>
16 /// Data structure for context data for epoll events.
17 /// When registering event handlers, a pointer to this struct must be provided;
18 /// this pointer's liveness must be maintained while the event is active.
19 /// In other words, do not use a local function variable for this data structure.
20 /// </summary>
21 typedef struct event_data {
22     /// <summary>
23     /// The event handler
24     /// </summary>
25     event_handler_t eventHandler;
26     /// <summary>
27     /// The file descriptor that generated the event
28     /// </summary>
29     int fd;
30     /// <summary>
31     /// a pointer to additional event context
32     /// </summary>
33     void * ptr;
34 } event_data_t;
35 
```

This allows to e.g. use a generic handler for all Led-timers since it now contains the required context information, what Led should be affected by the timer.

```
385 /// <summary>
386 ///     Handle the blinking for LEDs.
387 /// </summary>
388 static void LedUpdateHandler(event_data_t *eventData)
389 {
390 
391     if (ConsumeTimerFdEvent(eventData->fd) != 0) {
392         terminationRequired = true;
393         return;
394     }
395 
396     // Clear the send/receive LED2.
397     RgbLedUtility_SetLed((RgbLed *) eventData->ptr, RgbLedUtility_Colors_Off);
398 }
```


© 2018 Microsoft – Microsoft