Lab 6 Properties, Methods, Tags and Jobs

By: Kevin Saye

IoT Solution Architect

December, 2017

This lab assumes you have completed Lab 5.

If you have any issues or concerns, please email: virtualbootcamphelp@microsoft.com.

Execution Time: 30 minutes.

Required Hardware:

- Windows 10 PC
- IoT Hardware kit: https://www.adafruit.com/product/3605 or similar hardware.
- Access to a WiFi network (without a captive portal aka web page login)

Required Operating System:

- Windows 10

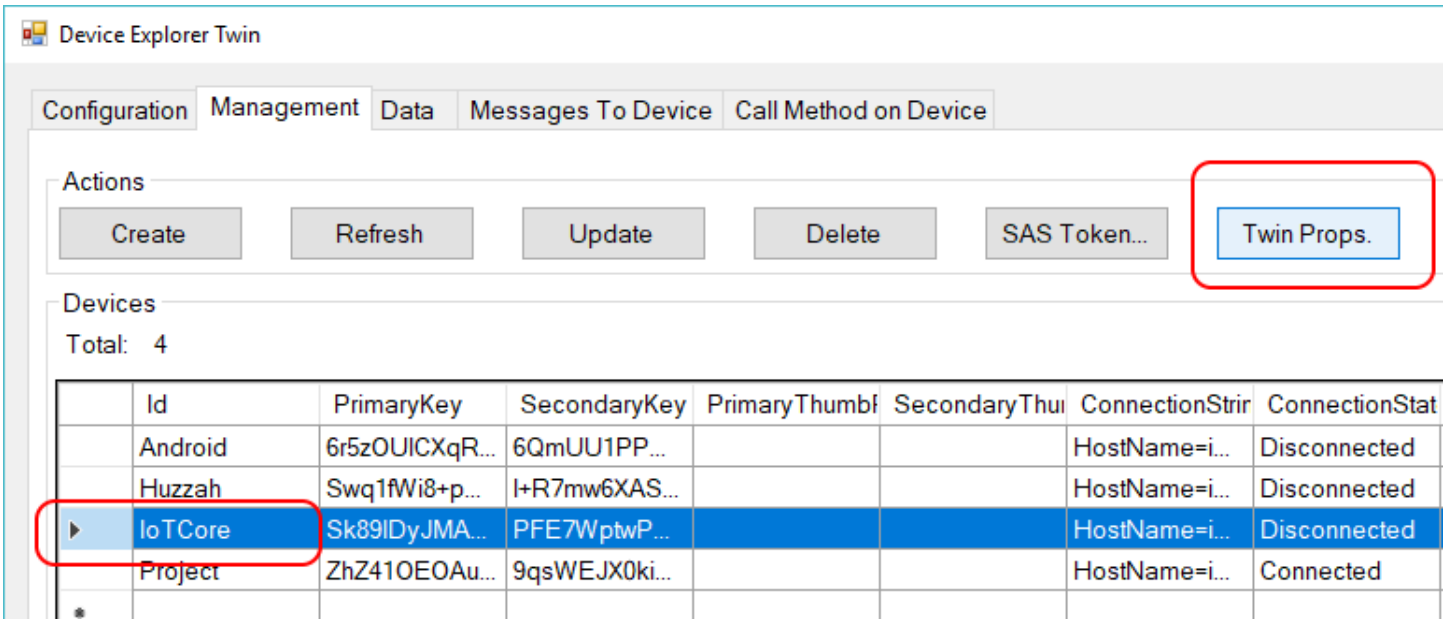
Other Requirements:

- Azure Subscription

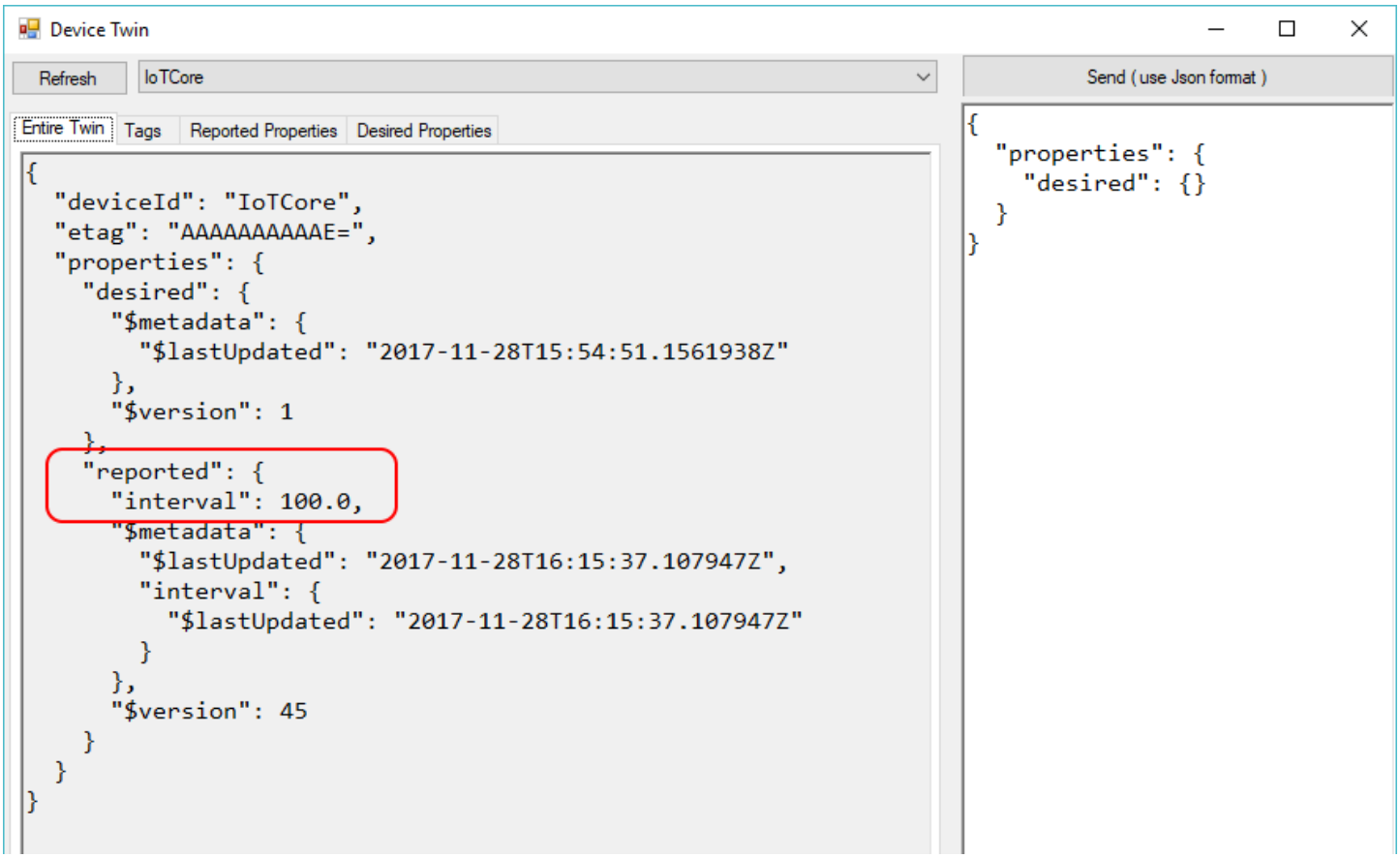
Required Software:

- Device Explorer (installed earlier)

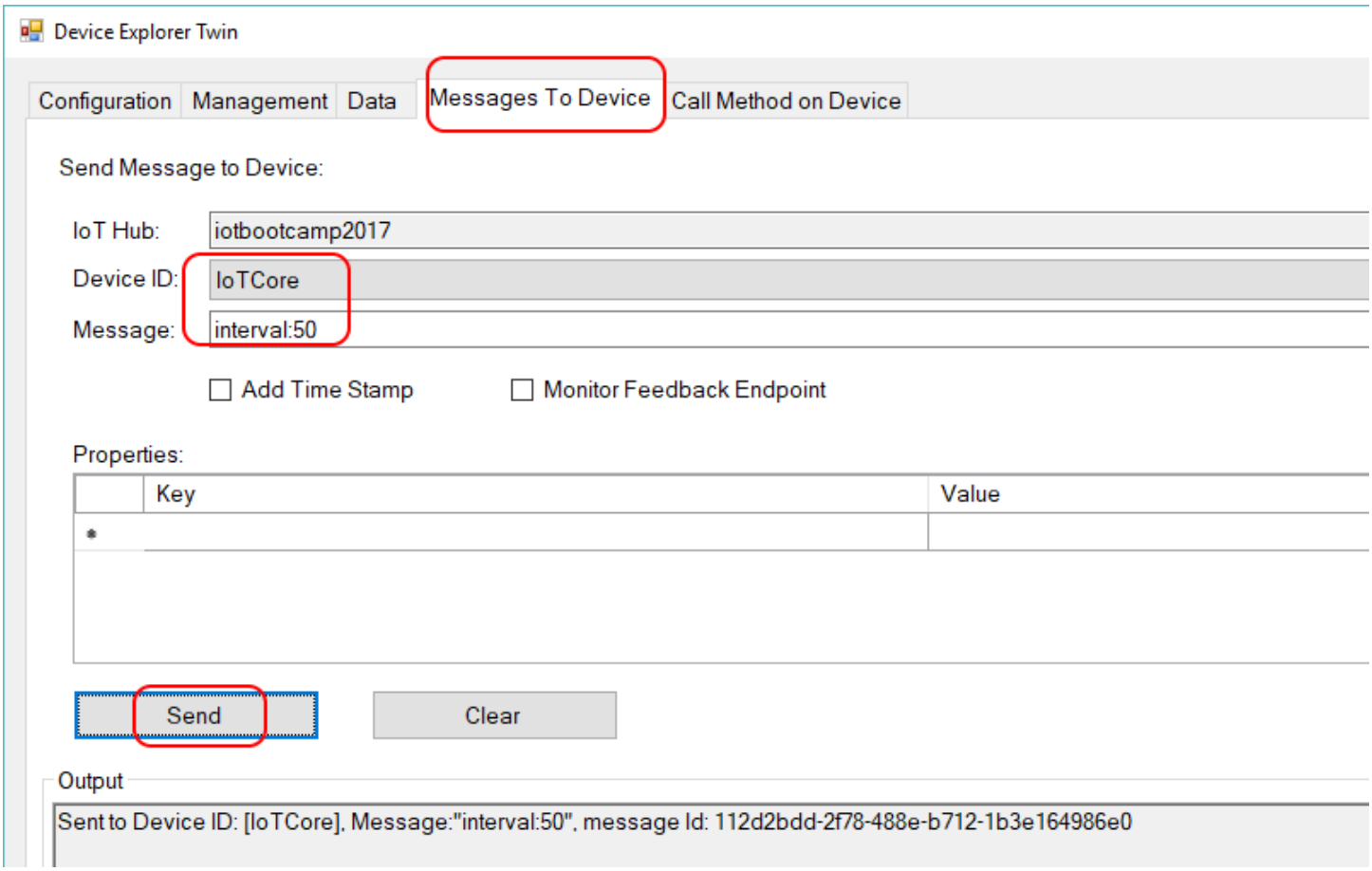
Step 1. Open the Azure Device Explorer, select your device and click Twin Props, as shown below:

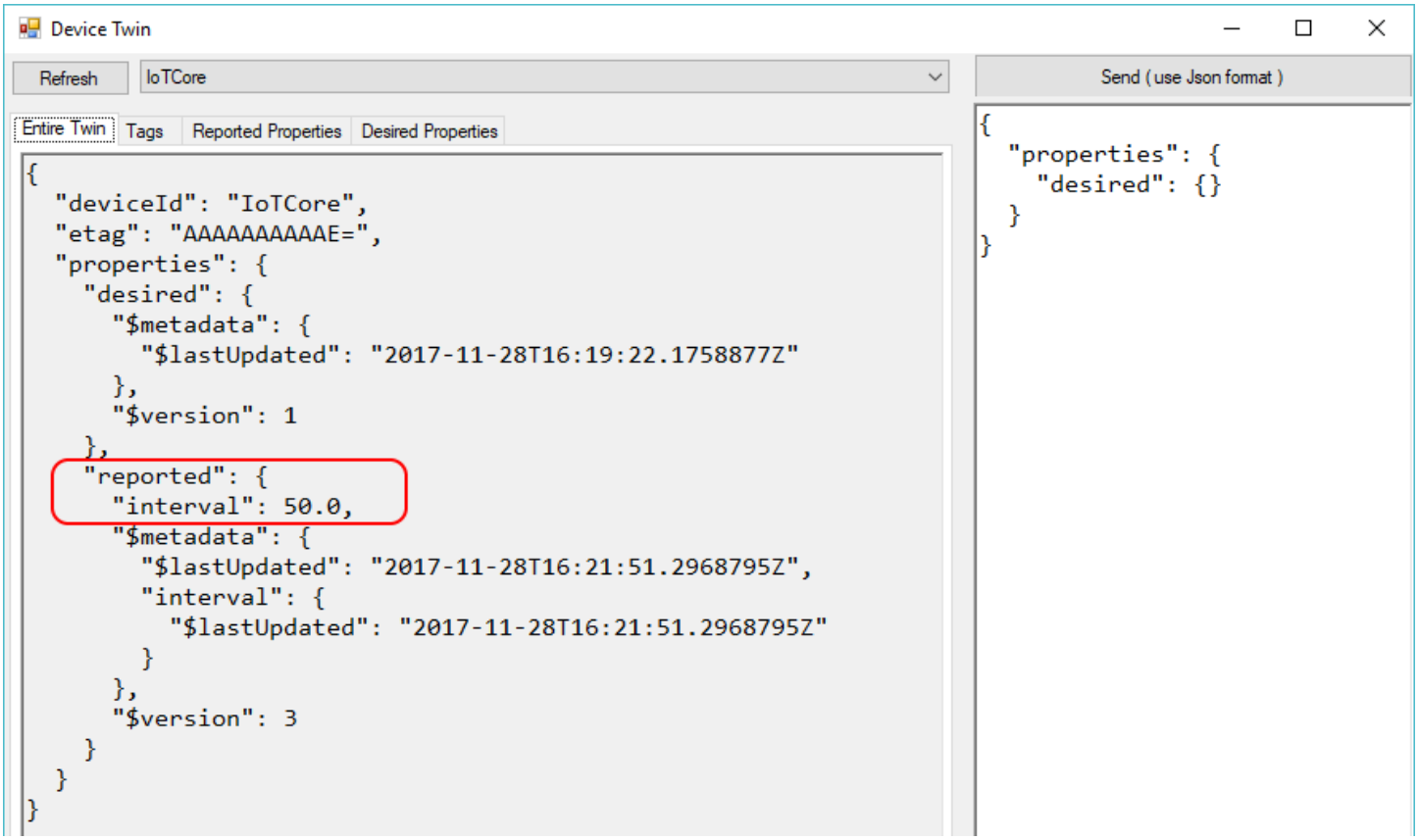


Step 2. Notice how the Reported Property states whatever you left it to the last time, as shown below:

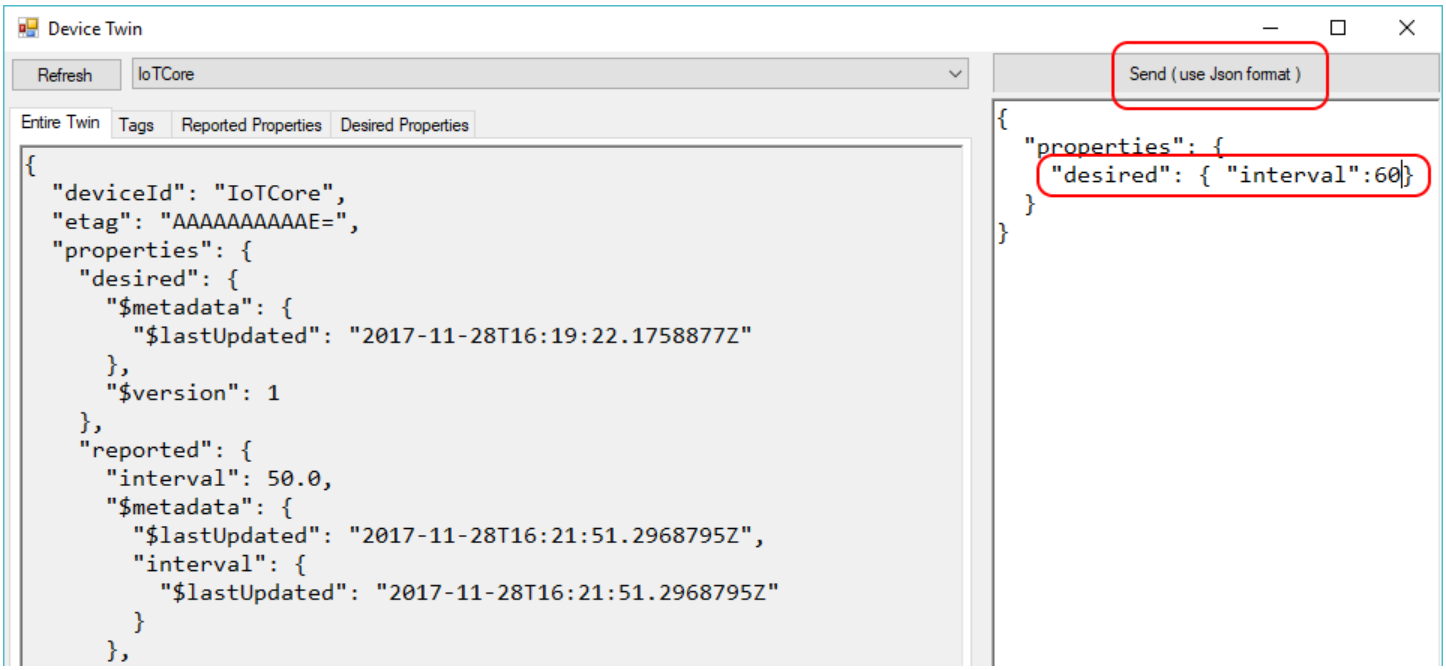


Step 3. Send the command interval:50, watch the IoT Core change the interval and refresh the properties to see the reported property change.

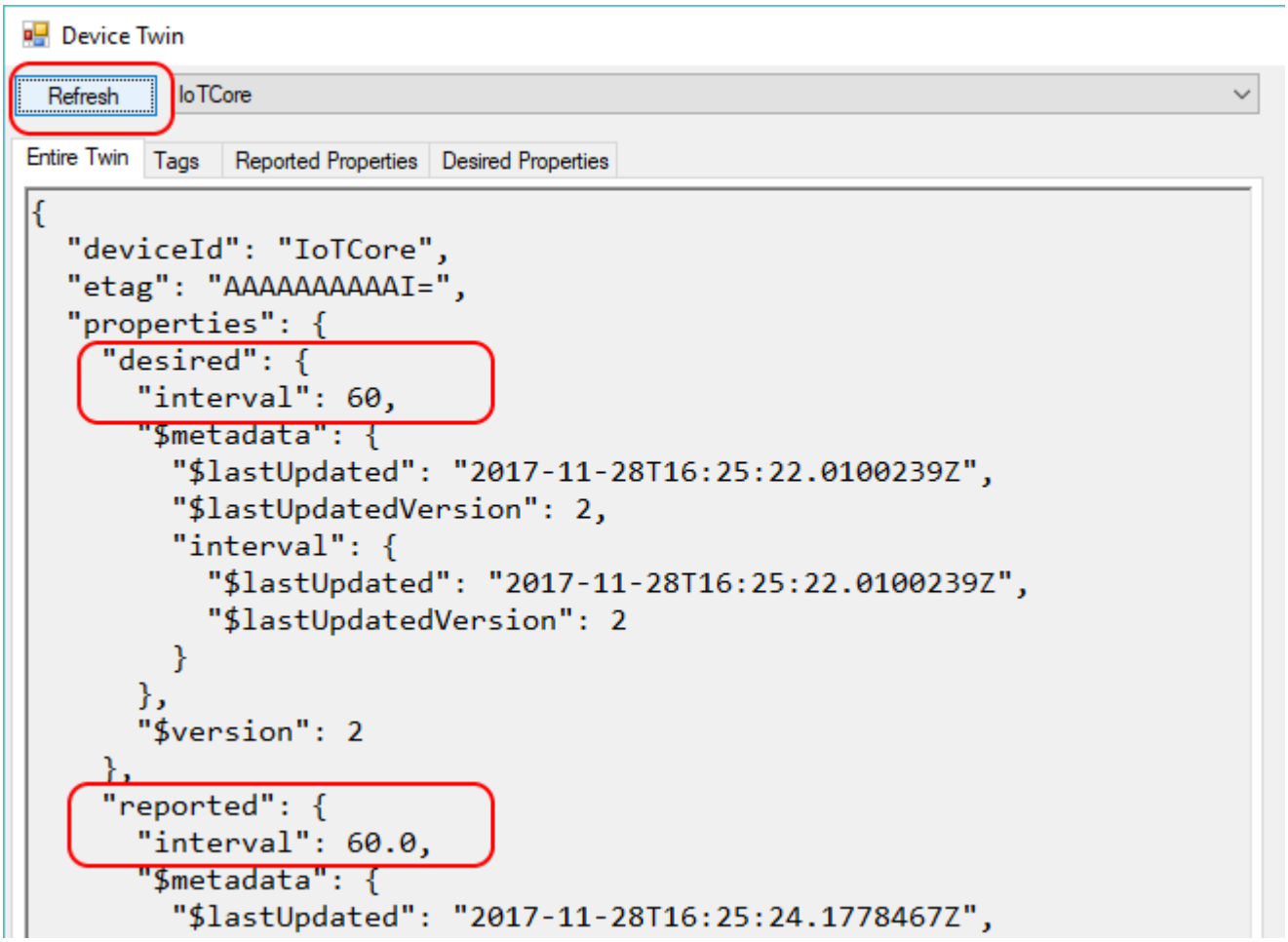




Step 4. Next, se the desired property, via JSON as shown below and watch the device interval change:



Step 5. Once you refresh the property (TWIN), you should see that the desired match the reported, as shown below:



Step 6. If we review the code in IoT Blinky, we see:

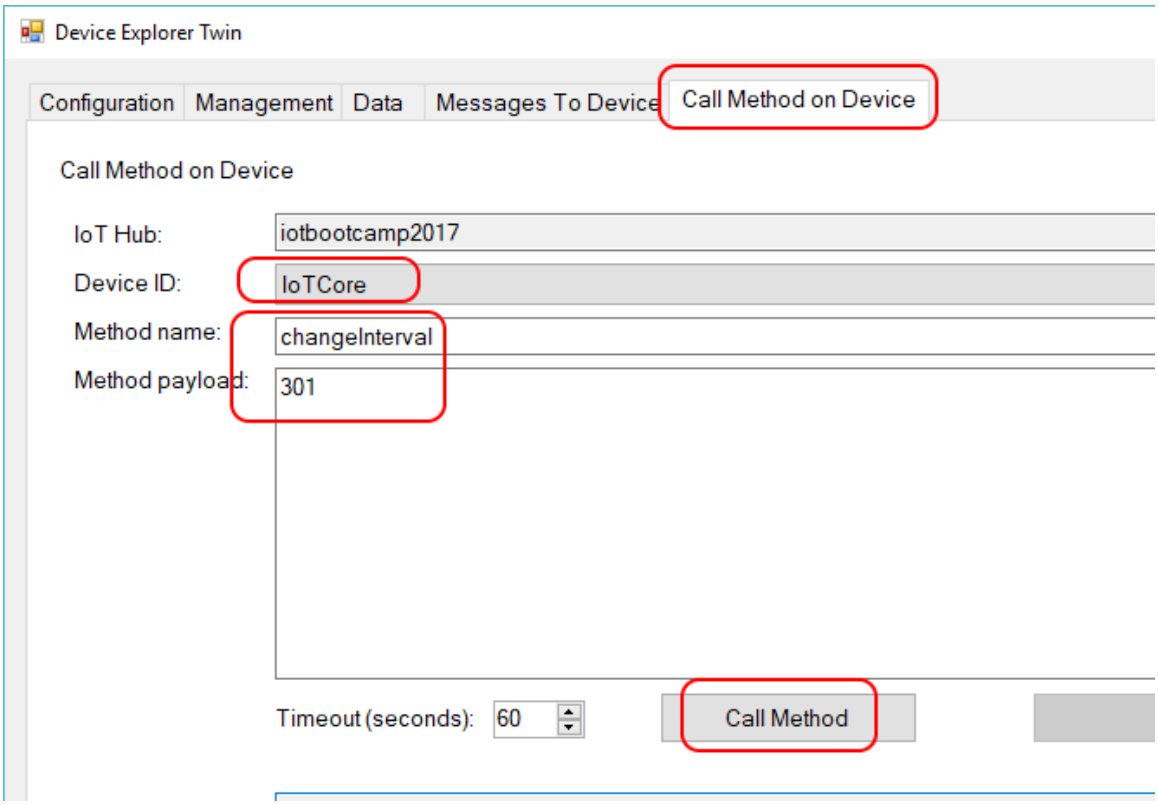
- Line 88 defines that we are listening for Property Changes
- Line 112 parses the desired properties
- Line 113 says if the interval is not null
- Lines 117 – 120 stops the timer, applies the new setting, updates the screen and restart the timer
- Line 123 updates the "reported" property, by calling the method on line 127 - 133

MainPage.xaml.cs | BlinkyIoT

BlinkyIoT | Blinky.MainPage

```
82
83      private async void connectToIoTHub()
84          {
85              deviceClient = DeviceClient.CreateFromConnectionString(IoTDeviceConnectionString, TransportType.Mqtt);
86              await deviceClient.OpenAsync();
87              await deviceClient.SetMethodHandlerAsync("changeInterval", changeInterval, null);
88              await deviceClient.SetDesiredPropertyUpdateCallbackAsync(OnDesiredPropertyChanged, null);
89
90          IoTSTatus.Text = "Connected to IoT Hub";
91      }
92
93      private async Task<MethodResponse> changeInterval(MethodRequest methodRequest, object userContext)...
109
110     private async Task OnDesiredPropertyChanged(TwinCollection desiredProperties, object userContext)
111     {
112         dynamic desiredProperty =  JObject.Parse(desiredProperties.ToJson());
113         if (desiredProperty.interval != null)
114         {
115             // Because we are in a Task, we have to use a dispatcher to get on the same Thread
116             await dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal, () => {
117                 timer.Stop();
118                 timer.Interval = TimeSpan.FromMilliseconds(Convert.ToDouble(desiredProperty.interval.Value));
119                 DelayText.Text = desiredProperty.interval.Value + "ms";
120                 timer.Start();
121             });
122
123             await UpdateTwin("interval", Convert.ToDouble(desiredProperty.interval.Value));
124         };
125     }
126
127     private async Task UpdateTwin(string property, double value)
128     {
129         TwinCollection reportedProperties = new TwinCollection();
130         reportedProperties[property] = value;
131
132         await deviceClient.UpdateReportedPropertiesAsync(reportedProperties);
133     }
```

Step 7. In Device Explorer, click Call Method, select set the Method Name to changeInterval with a payload of 301. Click Call Method, as shown below and watch the device interval change:



Step 8. In Device Explorer, we see that the TWIN interval has changed.


```
{
  "deviceId": "IoTCore",
  "etag": "AAAAAAAAAAM=",
  "version": 10,
  "properties": {
    "desired": {
      "interval": 60,
      "$metadata": {
        "$lastUpdated": "2017-11-30T20:41:03.3877419Z",
        "$lastUpdatedVersion": 3,
        "interval": {
          "$lastUpdated": "2017-11-30T20:41:03.3877419Z",
          "$lastUpdatedVersion": 3
        }
      },
      "$version": 3
    },
    "reported": {
      "interval": 301.0,
      "$metadata": {
        "$lastUpdated": "2017-11-30T20:44:27.9129543Z",
        "interval": {
          "$lastUpdated": "2017-11-30T20:44:27.9129543Z"
        }
      },
      "$version": 7
```

Step 9. If we review the code in IoT Blinky, we see:

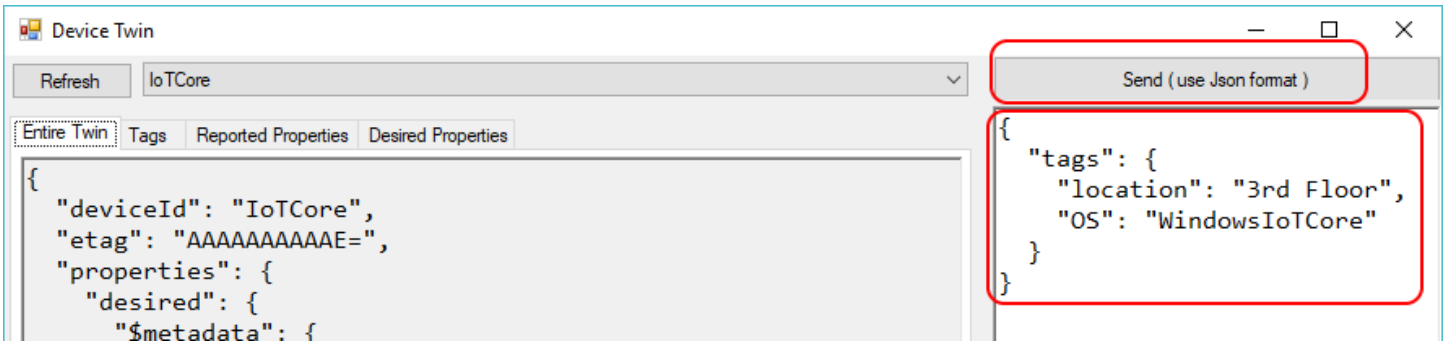
- Line 87 defines that we are listening for a Method named changeInterval
- Line 95 says if the method is not null
- Lines 98 – 104 defines that on the correct thread:
  - Stop the timer
  - Set the timer interval
  - Update the UI
  - Start the timer
- Line 406 updates the "reported" property, by calling the method on line 127 - 133

MainPage.xaml.cs | BlinkyIoT

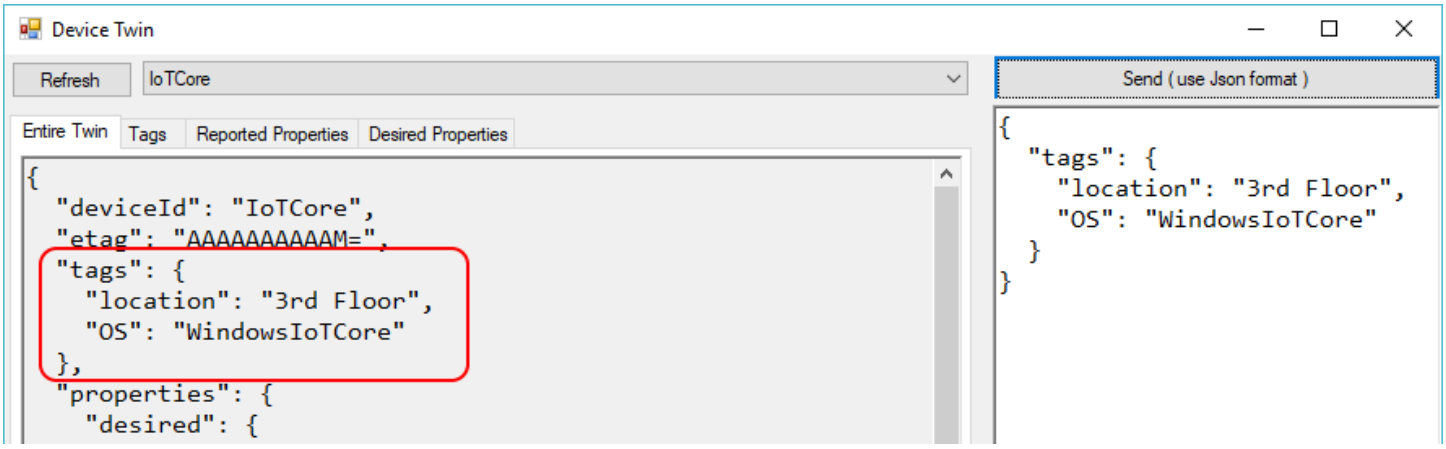
BlinkyIoT | Blinky.MainPage

```
82
83      private async void connectToIoTHub()
84          {
85              deviceClient = DeviceClient.CreateFromConnectionString(IoTDeviceConnectionString, TransportType.Mqtt);
86              await deviceClient.OpenAsync();
87              await deviceClient.SetMethodHandlerAsync("changeInterval", changeInterval, null);
88              await deviceClient.SetDesiredPropertyUpdateCallbackAsync(OnDesiredPropertyChanged, null);
89
90          IoTSTatus.Text = "Connected to IoT Hub";
91      }
92
93      private async Task<MethodResponse> changeInterval(MethodRequest methodRequest, object userContext)
94      {
95          if (methodRequest.DataAsJson != null)
96          {
97              // Because we are in a Task, we have to use a dispatcher to get on the same Thread
98              await dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal, ()=> {
99                  timer.Stop();
100                 timer.Interval = TimeSpan.FromMilliseconds(Convert.ToDouble(methodRequest.DataAsJson));
101                 DelayText.Text = methodRequest.DataAsJson + "ms";
102                 timer.Start();
103             });
104         };
105
106         await UpdateTwin("interval", Convert.ToDouble(methodRequest.DataAsJson.ToString()));
107         return null;
108     }
109
110     private async Task OnDesiredPropertyChanged(TwinCollection desiredProperties, object userContext)...
126
127     private async Task UpdateTwin(string property, double value)
128     {
129         TwinCollection reportedProperties = new TwinCollection();
130         reportedProperties[property] = value;
131
132         await deviceClient.UpdateReportedPropertiesAsync(reportedProperties);
133     }
```

Step 10. In the device properties (TWIN) of the device, make the following change shown below:



Step 11. Click Refresh to see the applied tags, as shown below:



Step 12. (Discussion Only) Now that we understand properties, tags and direct methods, we can better understand Jobs, as discussed below:

Jobs

Jobs are actions, such as a TWIN update, that you can schedule. We need to understand a few things before we can discuss Jobs vs. Direct Methods. Azure IoT is a public cloud offering, and as such has built in quotas to keep Customer A from consuming all the resources, causing Customer B to suffer.

When we look at the IoT Hub Quotas: https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-quotas-throttling we see that you can only update, for an S1 Hub, 1.67 devices per second. With this, imposed limit, we need to sometimes "schedule an update" to many devices. This is called a Job.

Lets say you want to update the heartbeat property on all devices in Las Colinas on the 3rd floor. To accomplish this, we will create a TWIN that represents our changes, create a job that applies this TWIN to a query based on existing TWIN properties. The following C# code shows this:

```
using Microsoft.Azure.Devices;
using Microsoft.Azure.Devices.Shared;

updateTwinsDesiredProperty("STARTSWITH(deviceId,'loadTest1') AND tags.location = 'Las
Colinas' AND tags.floor = '3rd'", "heartbeat", "16").Wait();

static async Task updateTwinsDesiredProperty(string query, string property, string value)
{
   jobClient = JobClient.CreateFromConnectionString(iotHubConnectionString);
   var twin = new Twin();
   twin.Properties.Desired[property] = value;
   JobResponse result = await
jobClient.ScheduleTwinUpdateAsync(Guid.NewGuid().ToString(),
                                                          query,
                                                          twin,
                                                          DateTime.Now,
                                                          100);
   Console.WriteLine("Started Twin Update Job for devices = '" + query + "' result = " +
result.Status);
   // close the connection
   jobClient.CloseAsync().Wait();
}
```

This concludes the lab.