# 1. Blinky and the Cloud

Let’s finish up by experimenting with writing a piece of data to the cloud every time the LED on your breadboard flashes.

Create a new event hub

Using the CreateResources.docx to guide you create another event hub for the Blinky and the Cloud lab. Some example setup variables below:

**Event Hub Name:** blinkyandthecloud<yourinitials>

**Region:** North Europe

**Create a new namespace called:** iottrack (or select the previously setup event hub namespace)

**Event hub number of partitions:** 8

**Message Retention:** 1 day

Once created, on the configure tab create a **new shared access policy**

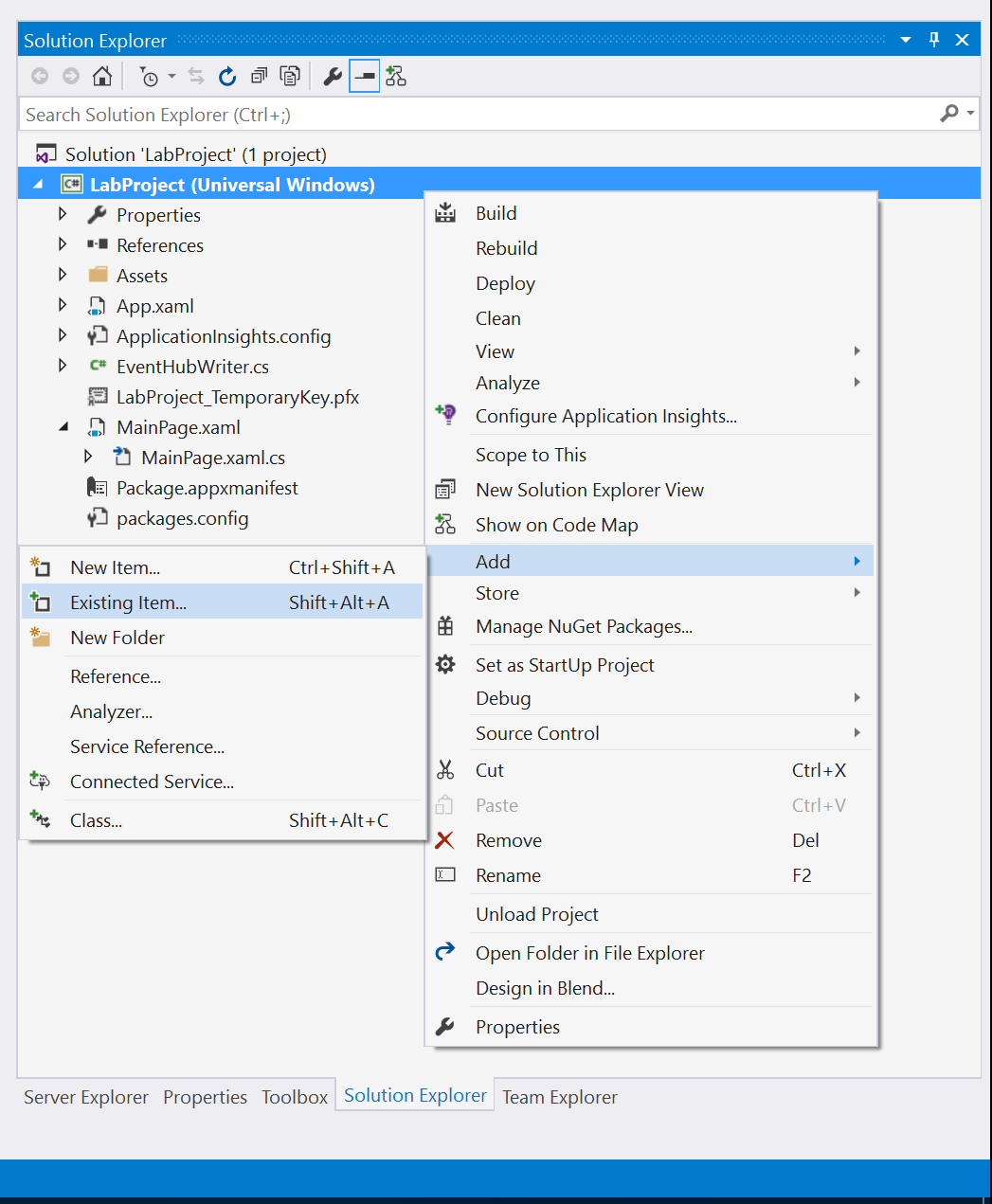
**Name:** sender

**Permissions:** manage, send, listen

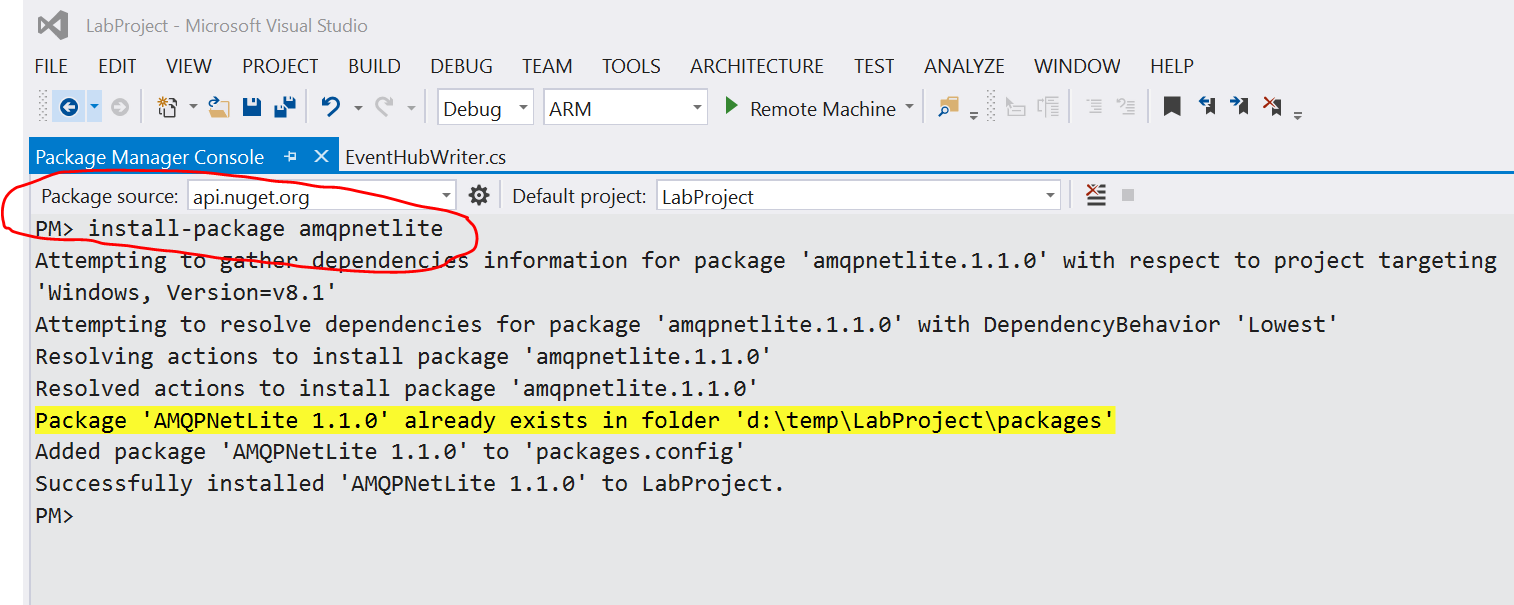
Finally choose Save

The code will post some data into your event hub. In order to allow this, a shared access security policy named **sender** has been created which allows write access to the event hub and you will download a library which makes it easy to write data to this event hub.

1. Locate the EventHubWriter.cs file within the IoT Track folder and 1.BlinkyandtheCloud
2. Add the **EventHubWriter.cs** code file to your project using the **Add->Existing Item** right mouse menu;



1. Inspect the code file – note that it contains a class in the **com.taulty.EventHubLibrary** namespace named **EventHubWriter** and note also that this code is dependent on a library named **Amqp** NET Lite which wraps up some of the details of using the Advanced Message Queueing Protocol to talk to Azure Event Hubs. Note finally that the code has spaces were you will have to add your event hub credentials in order for the application to send data to the cloud (see instructions shortly).
2. In order to compile your project, you will now need to add a reference to the NuGet package named **amqpnetlite**. The easiest way to do this is to use the **Tools->NuGet Package Manager->Package Manager Console** menu and then at the console install the package as below;



1. Alter the code such that it includes a **using** statement for the **com.mtaulty.EventHubLibrary** namespace and then create an instance of the **EventHubWriter** class when the app starts up. Construct it with the right event hub name which is **“mteventhub2”.**
2. Whenever the light flashes, use the instance of the **EventHubWriter** class to **WriteAsync** a custom string of your choosing to the cloud – see the code example below for the complete code at this point: **EventHubWriter.cs**
3. The highlighted code below indicates where you need to substitute in your own values for the event hub

namespace com.mtaulty.EventHubLibrary

{

using Amqp;

using Amqp.Framing;

using System;

using System.Net;

using System.Text;

using System.Threading.Tasks;

public class EventHubWriter

{

public EventHubWriter(string eventHubName)

{

if (string.IsNullOrEmpty(eventHubName))

{

throw new InvalidOperationException("No event hub name specified");

}

this.eventHubName = eventHubName;

this.address = new Lazy<Address>(() =>

{

return (new Address(

string.Format(AmqpAddressFormatString, EventHubPolicyName, WebUtility.UrlEncode(EventHubSasKey))));

});

this.connection = new Lazy<Connection>(() =>

{

return (new Connection(this.address.Value));

});

this.session = new Lazy<Session>(() =>

{

return (new Session(this.connection.Value));

});

this.senderLink = new Lazy<SenderLink>(() =>

{

// directly addresses partition 0

return (

new SenderLink(

this.session.Value,

"send-link:" + this.eventHubName,

this.eventHubName + "/Partitions/" + PartitionKey));

});

}

public Task WriteAsync(string messageToSend)

{

return (Task.Run(() =>

{

// not sure where the async went?

var message = new Message()

{

BodySection = new Data()

{

Binary = Encoding.UTF8.GetBytes(messageToSend)

}

};

this.senderLink.Value.Send(message);

}));

}

Lazy<Connection> connection;

Lazy<Session> session;

Lazy<Address> address;

Lazy<SenderLink> senderLink;

//Continued .. .. ..

static readonly string AmqpAddressFormatString = "amqps://{0}:{1}@<eventhubnamespace>.servicebus.windows.net";

static readonly string EventHubSasKey = "<connection SAS key>";

static readonly string EventHubPolicyName = "<policy name>";

static readonly string PartitionKey = "0";

string eventHubName;

}

}

MainPage.xaml.cs full code file

// Copyright (c) Microsoft. All rights reserved.

using System;

using Windows.Devices.Gpio;

using Windows.UI.Xaml;

using Windows.UI.Xaml.Controls;

using Windows.UI.Xaml.Controls.Primitives;

using Windows.UI.Xaml.Media;

using com.mtaulty.EventHubLibrary;

namespace Blinky

{

public sealed partial class MainPage : Page

{

private const int LED\_PIN = 5;

private GpioPin pin;

private GpioPinValue pinValue;

private DispatcherTimer timer;

//Brush colours depending on the LED used – add your own colour brush

private SolidColorBrush redBrush = new SolidColorBrush(Windows.UI.Colors.Red);

private SolidColorBrush yellowBrush = new SolidColorBrush(Windows.UI.Colors.Yellow);

private SolidColorBrush greenBrush = new SolidColorBrush(Windows.UI.Colors.Green);

private SolidColorBrush blueBrush = new SolidColorBrush(Windows.UI.Colors.Blue);

//Brush colour for the OFF light on the app

private SolidColorBrush whiteBrush = new SolidColorBrush(Windows.UI.Colors.White);

private SolidColorBrush grayBrush = new SolidColorBrush(Windows.UI.Colors.LightGray);

private EventHubWriter writer;

public MainPage()

{

InitializeComponent();

writer = new EventHubWriter("<eventhubname>");

timer = new DispatcherTimer();

timer.Interval = TimeSpan.FromMilliseconds(2000);

timer.Tick += Timer\_Tick;

InitGPIO();

if (pin != null)

{

timer.Start();

}

}

private void InitGPIO()

{

var gpio = GpioController.GetDefault();

// Show an error if there is no GPIO controller

if (gpio == null)

{

pin = null;

GpioStatus.Text = "There is no GPIO controller on this device.";

return;

}

pin = gpio.OpenPin(LED\_PIN);

pinValue = GpioPinValue.High;

pin.Write(pinValue);

pin.SetDriveMode(GpioPinDriveMode.Output);

GpioStatus.Text = "GPIO pin initialized correctly.";

}

private void Timer\_Tick(object sender, object e)

{

if (pinValue == GpioPinValue.High)

{

// add a message to the cloud

writer.WriteAsync("Message to the Cloud from Blinky: Light On,");

pinValue = GpioPinValue.Low;

pin.Write(pinValue);

//Update brush colour depending on LED

LED.Fill = blueBrush;

}

else

{

// add a message to the cloud

writer.WriteAsync("Message to the Cloud from Blinky: Light Off,");

pinValue = GpioPinValue.High;

pin.Write(pinValue);

LED.Fill = whiteBrush;

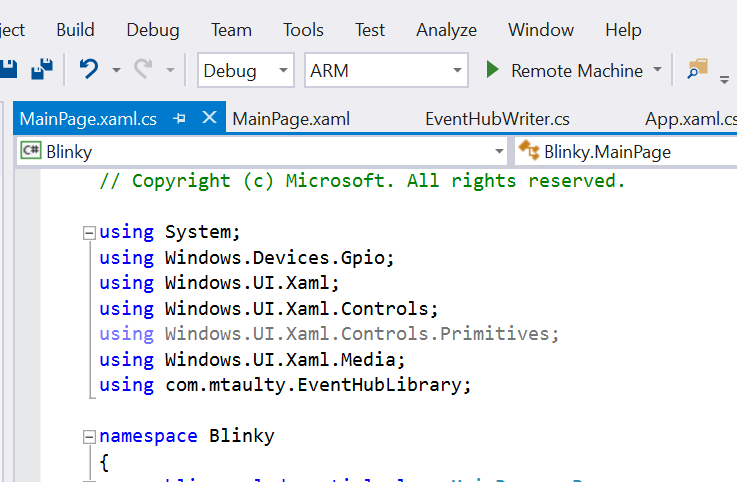
}

}

}

}

Once the changes above have been made, build and deploy the application by selecting the “remote machine” button



Once running your LED should light up for 2 seconds and turn off for 2 seconds. The setup of your RPI2 and breadboard should look like the picture below

