CHAPTER 4

Ingesting Data with Azure IoT Hub

Chapter 2 walked you through generating data with the Raspberry Pi and the Tessel. Chapter 3 introduced you to the Azure IoT Hub, a fully managed communications service which provides secure, reliable, and scalable messaging between IoT devices and IoT solutions. Chapter 3 walked through the creation and configuration of an Azure IoT Hub, but no devices were registered. Chapter 2 created a solution but it didn't send any data to the IoT Hub. Thus, this chapter is going to plug those two together. Essentially, you'll first register your device with the IoT Hub and then modify your project to send data to Azure IoT Hub.

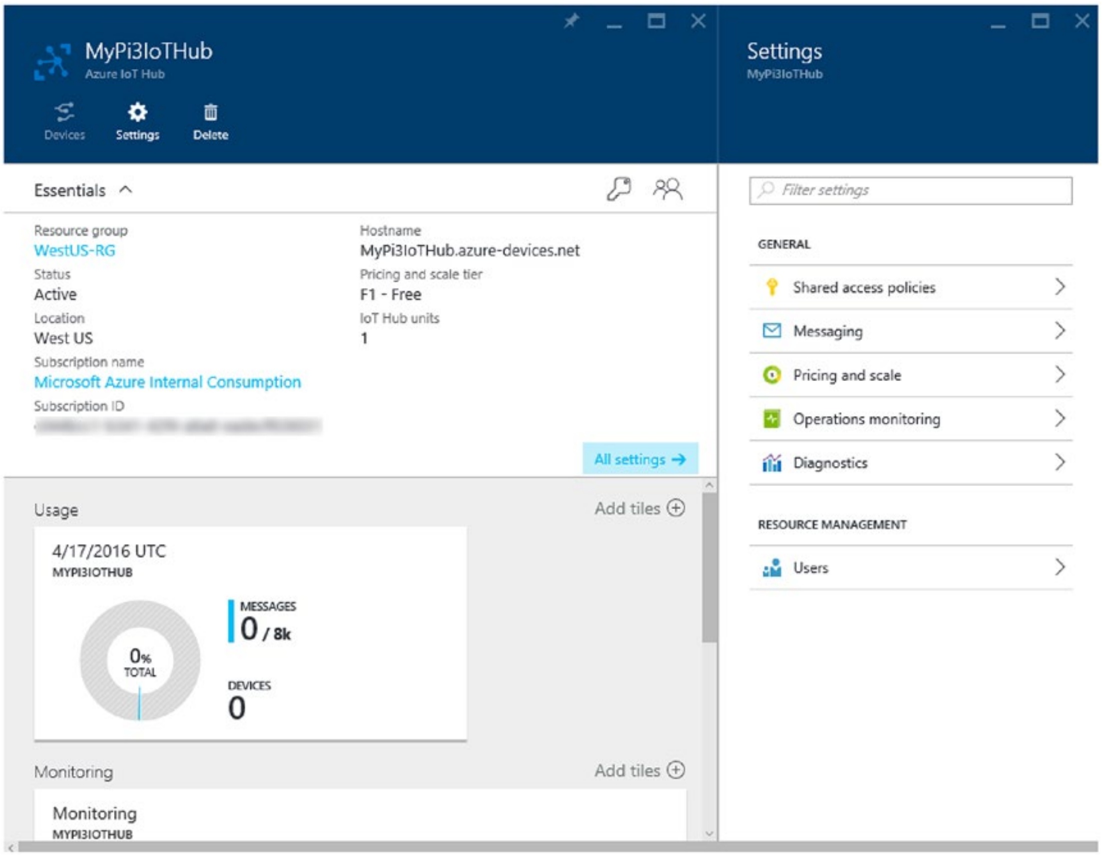
Registering the Device

Currently, the Azure portal does not provide the ability to provision or configure IoT Hub devices. However, as part of the Azure IoT SDK there is an application called Device Explorer which provides the means for managing devices and viewing device-to-cloud and cloud-to-device messages.

Thus, open your favorite browser and go to `https://github.com/Azure/azure-iot-sdk-csharp`. On the web page for the SDK, click the Download ZIP button. Once the zip file is downloaded, extract it to a folder. Within that folder, navigate to the `\tools\DeviceExplorer\` folder. The Device Explorer application is a Visual Studio solution, so open Visual Studio 2015 and open the DeviceExplorer solution.

Before running the solution, compile the solution first because it will need to download all the project dependencies from Nuget. Once the compile succeeds successfully, run the project.

In order to properly register a device in Device Explorer, you first need some information from the Azure portal. Log in to the Azure portal and open the IoT Hub that was created in the previous chapter. Notice in Figure 4-1 that in the Usage section of the Overview blade, there are no devices listed. But what you need from here is the connection string in order to connect the Device Explorer to IoT Hub in order to register any devices.



MyPi3IoTHub
Azure IoT Hub

Devices | Settings | Delete

Essentials

Resource group
WestUS-RG
Status
Active
Location
West US
Subscription name
Microsoft Azure Internal Consumption
Subscription ID
[redacted]

Hostname
MyPi3IoTHub.azure-devices.net
Pricing and scale tier
F1 - Free
IoT Hub units
1

All settings →

Usage — Add tiles

4/17/2016 UTC
MYPI3IOTHUB
0% TOTAL
MESSAGES 0 / 8k
DEVICES 0

Monitoring — Add tiles

Monitoring
MYPI3IOTHUB

Settings
MyPi3IoTHub

Filter settings

GENERAL
- Shared access policies
- Messaging
- Pricing and scale
- Operations monitoring
- Diagnostics

RESOURCE MANAGEMENT
- Users

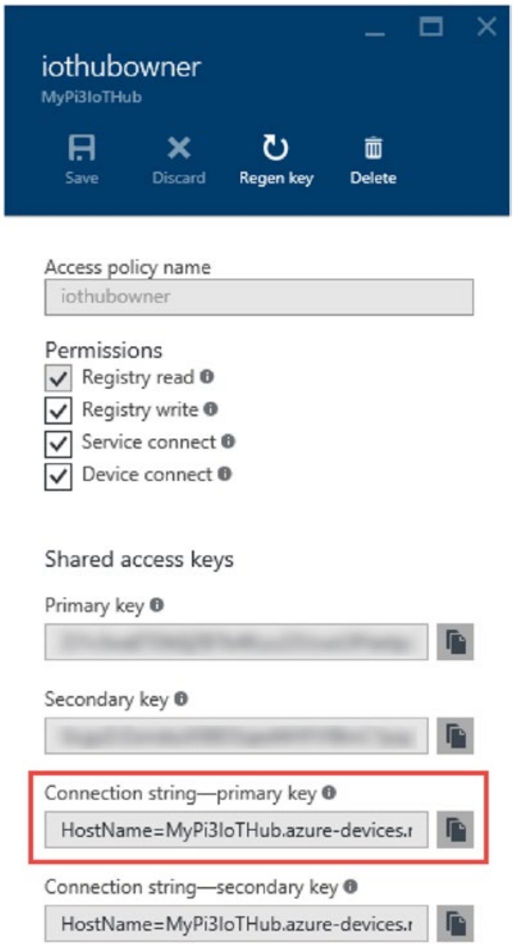
Figure 4-1. *IoT Hub overview*

In the Settings blade, click the “Shared access policies” option, and then in the Shared access policies blade, click the iothubowner policy. This will open the Access permissions blade for the iothubowner policy. In the Policy blade, the piece of information needed is the primary key connection string, highlighted in Figure 4-2. Click the Click to Copy icon to the right of the connection string field to copy the value to the clipboard.

The Device Explorer tool uses a set of Azure IoT service libraries to execute a number of tasks such as adding devices or viewing messages. It uses the connection string in order to connect to Azure IoT Hub via the service libraries (API calls).

The connection string is made of three pieces:

- **HostName**: The URI of the Azure IoT Hub, thus, MyPi3IoTHub.azure-devices.net
- **SharedAccessKeyName**: The policy name, in this case iothubowner
- **SharedAccessKey**: A unique key used to authenticate to Azure IoT Hub



iothubowner
MyPi3IoTHub

Access policy name: iothubowner

Permissions
- ✓ Registry read
- ✓ Registry write
- ✓ Service connect
- ✓ Device connect

Shared access keys

Primary key

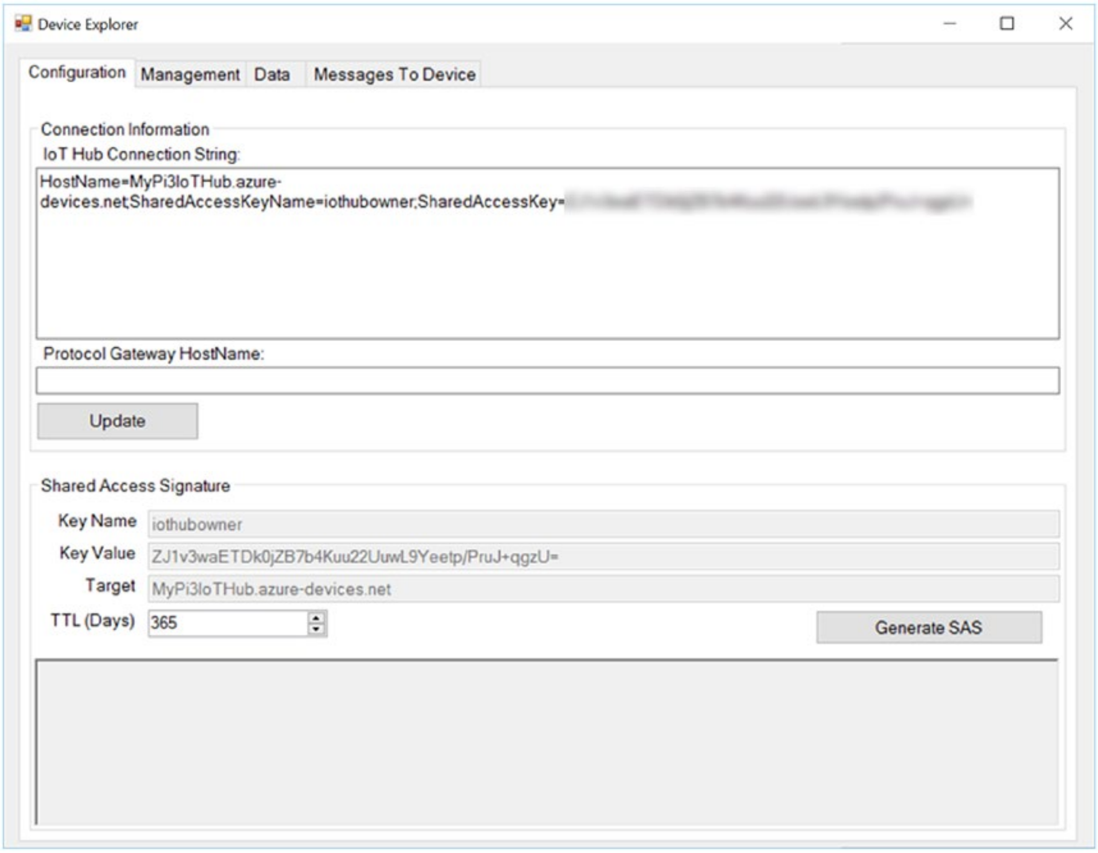
Secondary key

Connection string—primary key: HostName=MyPi3IoTHub.azure-devices.r

Connection string—secondary key: HostName=MyPi3IoTHub.azure-devices.r

Figure 4-2. *Getting the IoT Hub primary connection string*

With Device Manager running, ensure that the Configuration tab is selected and paste the connection string into the "IoT Hub Connection String" text box. Click the Update button to save the configuration. You will then receive a message confirming the update of the setting and configuring the tool to communicate with IoT Hub. Device Manager should now look like Figure 4-3.



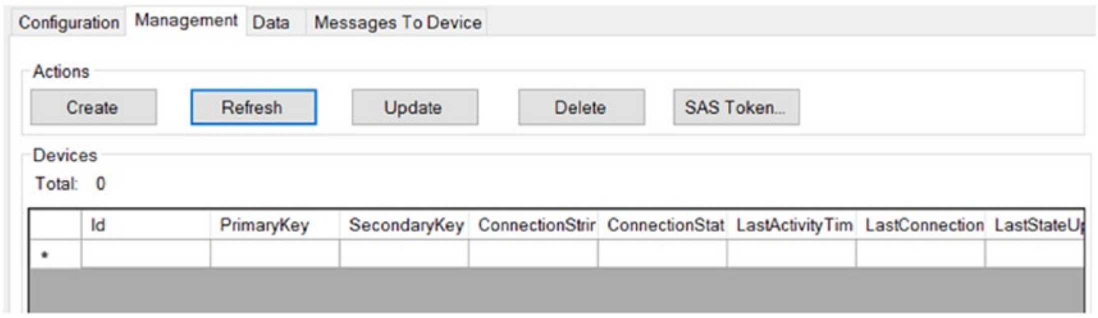
IoT Hub Connection String:
HostName=MyPi3IoTHub.azure-devices.net;SharedAccessKeyName=iothubowner;SharedAccessKey=[redacted]

Protocol Gateway HostName:

Key Name: iothubowner
Key Value: ZJ1v3waETDk0jZB7b4Kuu22UuwL9Yeetp/PruJ+qgzU=
Target: MyPi3IoTHub.azure-devices.net
TTL (Days): 365

Figure 4-3. *Configuring the Device Explorer*

Next, click the Management tab. This tab is where devices are registered with IoT Hub. As shown in Figure 4-4, you can create, delete, and update devices with IoT Hub.

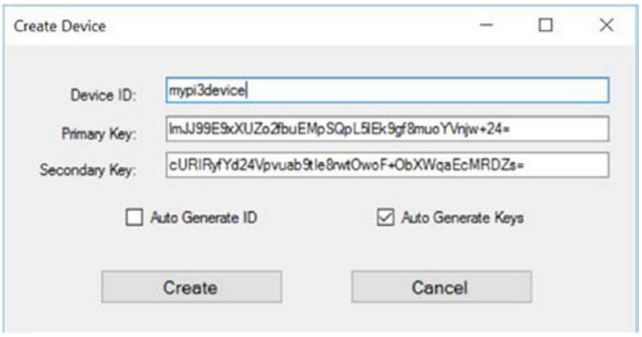


Total: 0

	Id	PrimaryKey	SecondaryKey	ConnectionStrir	ConnectionStat	LastActivityTim	LastConnection	LastStateUp
*								

Figure 4-4. *Managing devices in the Device Explorer*

Click the Create button. This will open a dialog that will prompt you to provide a device name for the device you want to add, as seen in Figure 4-5. Associated with this device ID are two values, a primary key and a secondary key. These values will be prepopulated unless you specify to not have them autogenerated by unchecking the Auto Generate Keys checkbox.



Device ID: mypi3device

Primary Key: ImJJ99E9xXUZo2fbuEMpSQpL5lEk9gf8muoYVnjw+24=

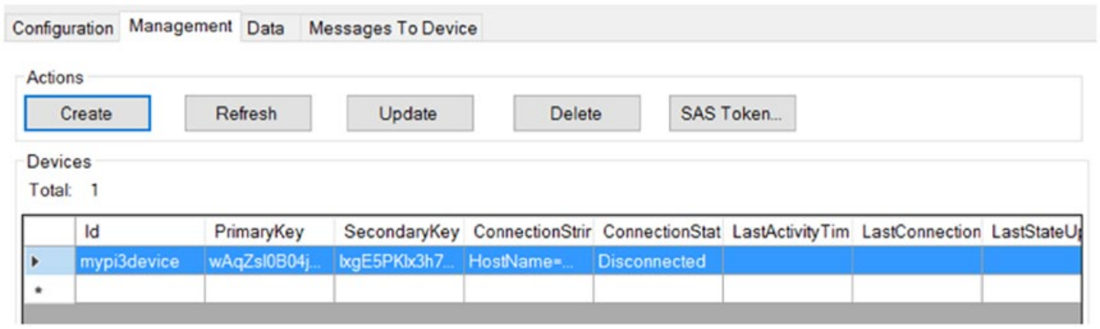
Secondary Key: cURIRyfYd24Vpvuab9tle8rwtOwoF+ObXWqaEcMRDZs=

☐ Auto Generate ID   ☑ Auto Generate Keys

Figure 4-5. *Creating a new device*

Provide a device ID by entering a unique name and then click Create. For this example, I named my device "mypi3device." Obviously you will want to have a more unique and descriptive name, especially if you are dealing with more than one device.

Clicking Create on the dialog will then create the device and register it with Azure IoT Hub. Device Explorer will be updated with a list of all registered devices, as shown in Figure 4-6.



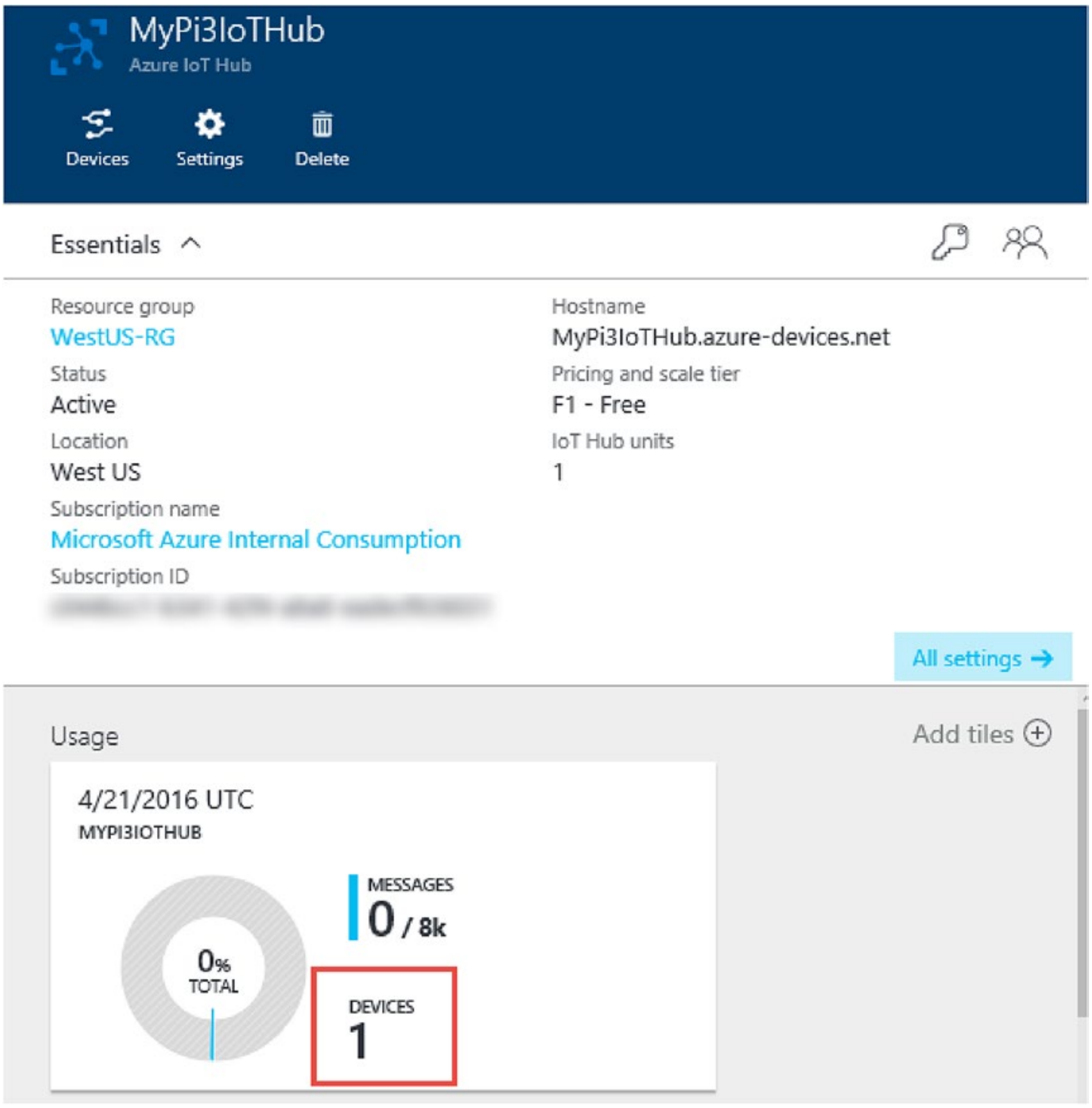
Total: 1

	Id	PrimaryKey	SecondaryKey	ConnectionStrir	ConnectionStat	LastActivityTim	LastConnection	LastStateU
▸	mypi3device	wAqZsl0B04j...	lxgE5PKlx3h7...	HostName=...	Disconnected			
*								

Figure 4-6. *Newly created device*

The information that should stand out in Figure 4-6 is the connection status, shown as Disconnected. Since your device isn't connected to the Azure IoT Hub, this status makes sense. You haven't added the code to your Visual Studio project to wire it up to the IoT Hub. You'll do that momentarily. However, in order to connect the app to the Azure IoT Hub you will need the connection string. This tool makes it easy: you simply right-click the row for the device and select "Copy connection string for selected device" from the context menu.

Before you modify the application, first go back to the portal and refresh the Overview blade. The first thing to point out is that in the usage section it now lists one device, which is the device you just added. The next thing to point out is that on the header of the Overview tab the Devices button is now enabled. Clicking the Devices button will open the Device Explorer blade, shown in Figure 4-7. This blade lists all the registered devices and their status. In this case, you only have one device so the list isn't long. However, this blade does provide a great initial view into the devices associated and connected to the corresponding IoT Hub.


MyPi3IoTHub
Azure IoT Hub

Resource group
WestUS-RG
Status
Active
Location
West US
Subscription name
Microsoft Azure Internal Consumption
Subscription ID
[redacted]

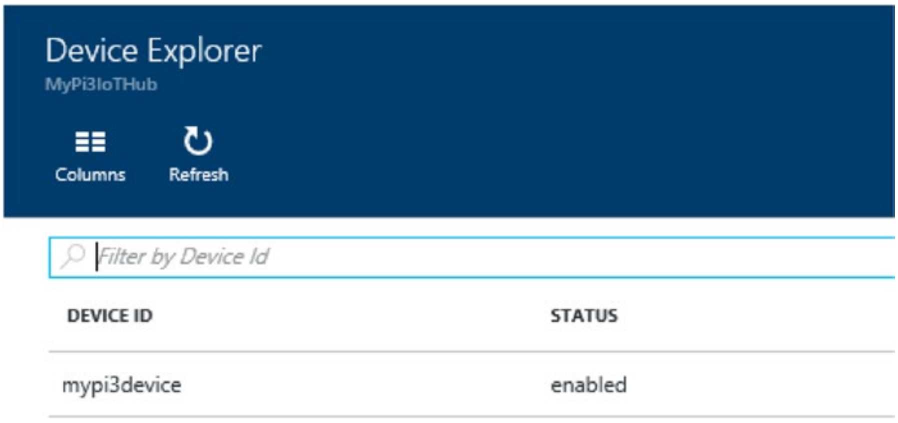
Hostname
MyPi3IoTHub.azure-devices.net
Pricing and scale tier
F1 - Free
IoT Hub units
1

Usage
4/21/2016 UTC
MYPI3IOTHUB
0% TOTAL
MESSAGES
0 / 8k
DEVICES
1

Figure 4-7. *Updated Overview blade showing new device*

There are several things that are nice about the Device Explorer blade. First, you can add additional columns to the list of devices. By default, it only shows the device name and status. However, you can add three additional columns: Status Reason, Last Status Update, and Last Activity. All three of these columns can provide real-time insight into the status of the device to help troubleshoot any connectivity issues the device might be having.

Second, it allows you to filter the list of devices, which will come in handy if you have many, many devices to manage. Clicking a specific device will open a Details blade, which provides key and connection string information as well as the ability to enable and disable the connection to the IoT Hub, as shown in Figure 4-8.



DEVICE ID	STATUS
mypi3device	enabled

Figure 4-8. Azure Portal Device Explorer

Honestly, registering the device with Azure IoT Hub was pretty easy. At some point this functionality will be available in the portal, but for now you need to use the Device Explorer tool. This isn't necessarily bad because it does provide some great functionality. The next step is to update the application to connect to IoT Hub and start sending messages. You can then come back to the Device Explorer tool and refresh the list to show it in a connected state.

Updating the Application

This section will walk through updating both the Raspberry Pi and Tessel applications to add code to send the data to Azure and the IoT Hub. We'll begin with the Raspberry Pi.

Raspberry Pi

As it currently sits, the application created in Chapter 3 generates good data, but it does not send any data to Azure IoT Hub. The changes necessary to implement this functionality are actually quite minor and easy. The first thing to do is add the following line of code below the class declaration section with the other declared variables:

```
private const string IOTHUBCONNECTIONSTRING = "HostName=<IoTHubName>.azure-devices.net;Devic
eId=<DeviceID>;SharedAccessKey=<SharedAccessKey>";
```

Next, replace the connection string with the connection string copied from the Device Explorer:

```
private const string IOTHUBCONNECTIONSTRING = "HostName=MyPi3IoTHub.azure-devices.net;Device
Id=mypi3device;SharedAccessKey=<SharedAccessKey>";
```

The connection string has three components:

- **HostName**: The name of the Azure IoTHub hostname. You can get this from the Azure portal for your IoT Hub Overview blade, shown in Figure 4-7.
- **DeviceID**: The name given to the device when registering the device with IoT Hub, shown in Figure 4-5.
- **SharedAccessKey**: The primary symmetric shared access key stored in base64 format. This is the same key found on the Device Details pane in the Primary Key field, discussed earlier.

The next step is to modify the code in the `readSensor()` method as follows:

```
DhtReading reading = await dht.GetReadingAsync().AsTask();
if (reading.IsValid)
{
    double temp = ConvertTemp.ConvertCelsiusToFahrenheit(reading.Temperature);

    var payload = new
    {
        Device = "Pi2",
        Sensor = "DHT11",
        Temp = temp.ToString(),
        Humidity = reading.Humidity.ToString(),
        Time = DateTime.Now
    };

    string json = JsonConvert.SerializeObject(payload);

    //send to listbox
    listBox.Items.Add(json.ToString());

    //send to azure IoT Hub
    Message eventMessage = new Message(Encoding.UTF8.GetBytes(json));
    DeviceClient deviceClient = DeviceClient.CreateFromConnectionString
    (IOTHUBCONNECTIONSTRING);
    await deviceClient.SendEventAsync(eventMessage);

}
```

I'll point out some of the changes in the code. First, you're still converting the temperature to Fahrenheit, but the next statement creates an object not only with the temperature and humidity, but you also want to track which specific device and sensor the temperature and humidity is coming from, as well as the date and time of the generated data.

You then use the `JsonConvert.SerializeObject` method to serialize the specific object to a JSON string. That JSON is then sent to the ListBox. Next, with the Microsoft.Azure.Devices.Client package installed from Nuget, you can add the code to send the temperature data to Azure IoT Hub. Thus, the three lines of code immediately following the ListBox statement do just that. The first line creates a new instance of the `Message` class, which the serialized JSON object is added to. The second statement creates an instance of the `DeviceClient` class in which a connection is made to Azure IoT Hub using the connection string added above. Lastly, the message is sent to Azure IoT Hub using the `SendEventAsync` method.

Now, good programming practice dictates that you create a class called `DeviceMessage` with the appropriate properties and then use a generic in Newtonsoft to serialize and deserialize the class. The code above is a quick solution to start sending data. However, to do it right, right-click the Temperature project and select Add ➤ Class. In the Add New Item dialog, ensure the Class template is selected and rename the class to DeviceMessage and click OK. Change the class `DeviceMessage` to public and then add the properties as follows:

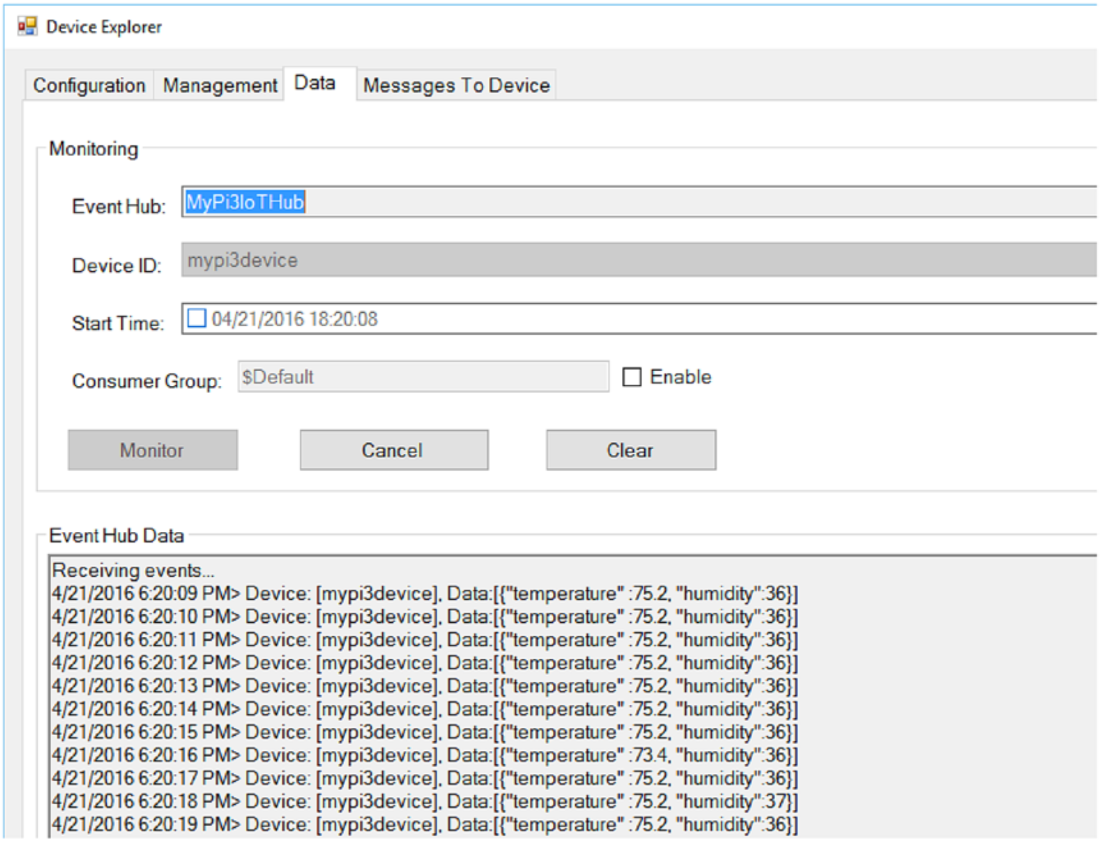
```
public class DeviceMessage
{
    public string Device { get; set; }
    public string Sensor { get; set; }
    public string Temp { get; set; }
    public string Humidity { get; set; }
    public DateTime Time { get; set; }
};
```

Then, back in the `readSensor()` method on the MainPage, replace `the var payload` and `string json` statements with the following:

```
DeviceMessage msg = new DeviceMessage
{
    Device = "Pi2",
    Sensor = "DHT11",
    Temp = temp.ToString(),
    Humidity = reading.Humidity.ToString(),
    Time = DateTime.Now
};
string json = JsonConvert.SerializeObject(msg, Formatting.Indented);
```

So now you are following good coding practice (well, technically, you would also add some error handling). At this point, the solution is ready to be compiled and run. Recompile the solution and when that is done, deploy the solution to the Raspberry Pi by clicking the green Play button next to Remote Machine on the toolbar again. Or, simply press F5.

If the ListBox line of code is uncommented, you'll see data written to the ListBox. However, the important part is seeing it in Azure IoT Hub. Thus, go back to the Device Manager tool and select the Data tab. On the Data tab, click the Monitor button. If the IoT Hub is indeed receiving messages, you'll see them showing up in the Event Hub Data ListBox, as shown in Figure 4-9. The Cancel button stops messages from being written to the ListBox; it does not stop messages from being received by IoT Hub. The Clear button clears the ListBox.



Event Hub: MyPi3IoTHub

Device ID: mypi3device

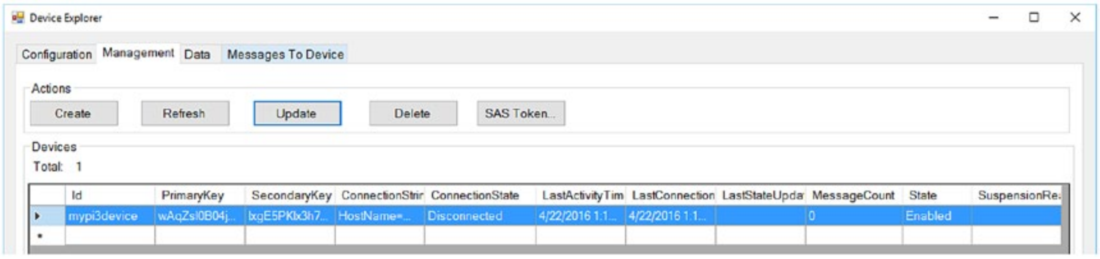
Start Time: 04/21/2016 18:20:08

Consumer Group: $Default ☐ Enable

```
Receiving events...
4/21/2016 6:20:09 PM> Device: [mypi3device], Data:[{"temperature":75.2, "humidity":36}]
4/21/2016 6:20:10 PM> Device: [mypi3device], Data:[{"temperature":75.2, "humidity":36}]
4/21/2016 6:20:11 PM> Device: [mypi3device], Data:[{"temperature":75.2, "humidity":36}]
4/21/2016 6:20:12 PM> Device: [mypi3device], Data:[{"temperature":75.2, "humidity":36}]
4/21/2016 6:20:13 PM> Device: [mypi3device], Data:[{"temperature":75.2, "humidity":36}]
4/21/2016 6:20:14 PM> Device: [mypi3device], Data:[{"temperature":75.2, "humidity":36}]
4/21/2016 6:20:15 PM> Device: [mypi3device], Data:[{"temperature":75.2, "humidity":36}]
4/21/2016 6:20:16 PM> Device: [mypi3device], Data:[{"temperature":73.4, "humidity":36}]
4/21/2016 6:20:17 PM> Device: [mypi3device], Data:[{"temperature":75.2, "humidity":36}]
4/21/2016 6:20:18 PM> Device: [mypi3device], Data:[{"temperature":75.2, "humidity":37}]
4/21/2016 6:20:19 PM> Device: [mypi3device], Data:[{"temperature":75.2, "humidity":36}]
```

Figure 4-9. *Viewing live data in Device Explorer*

Selecting the Management tab and clicking the Refresh button now shows the ConnectionState as Connected, which is seen in Figure 4-10.



Total: 1

	Id	PrimaryKey	SecondaryKey	ConnectionStrir	ConnectionState	LastActivityTim	LastConnection	LastStateUpda	MessageCount	State	SuspensionRe
▸	mypi3device	wAqZsI0B04j...	lxgE5PKlx3h7...	HostName=...	Disconnected	4/22/2016 1:1...	4/22/2016 1:1...		0	Enabled	
*											

Figure 4-10. *Device Explorer connection state*

Now go back to the Azure portal and refresh the Overview blade; you should now see in the Usage section the number of messages ingested into Azure IoT Hub at the time of the refresh. The blade will refresh automatically every minute so you will get an idea of how many messages are being sent and consumed by IoT Hub (see Figure 4-11).

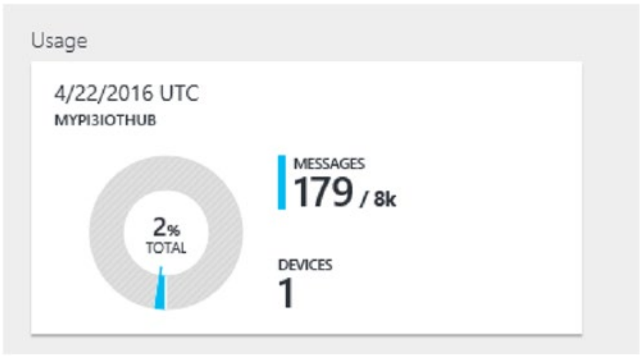


Figure 4-11. *Updated message count in the Azure Portal*

In this example, you simply modified the application to include the connection to Azure IoT Hub and sent the temperature and humidity messages to IoT Hub. The temperature and humidity readings were taken every second, and immediately sent to IoT Hub, which means data was being sent to IoT Hub every second.

Is this the right thing to do? Does data need to be sent to IoT Hub every second? The next section will address this question and others, discussing design considerations and best practices.

Tessel

Several months ago I wrote a blog post which details how to modify the same example to send temperature code to Azure IoT Hub. Instead of rewriting it here, you can read it and follow along on my blog post at `https://blogs.msdn.microsoft.com/sqlscott/2016/10/17/tessel-2-and-microsoft-azure/`. Plus, it will make it easy to copy/paste the code if you are following along and using a Tessel.

Considerations

Before wrapping up this chapter I'd like to discuss a couple of additional items that are relevant to this topic.

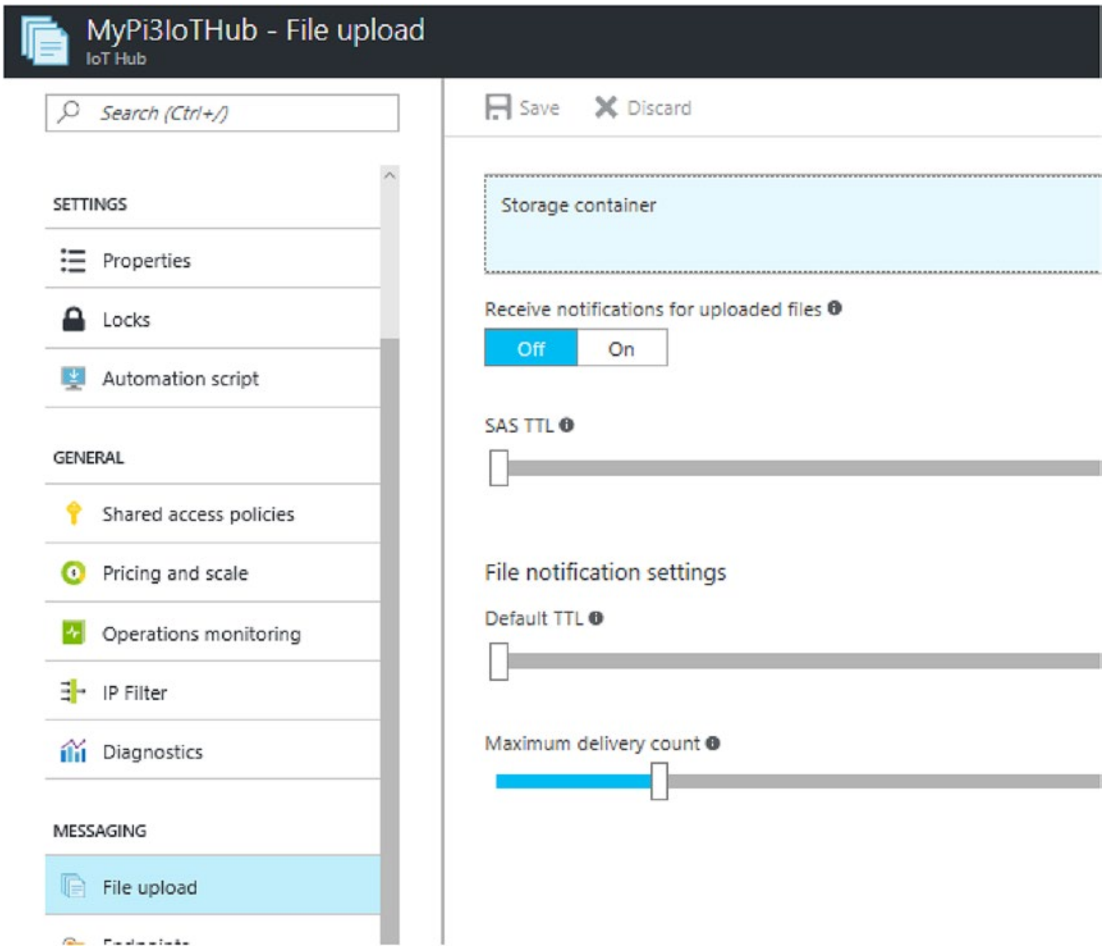
Uploading Files to Azure IoT Hub

This chapter focused on using devices to send messages to Azure IoT Hub. However, there are some scenarios where it is more difficult to map the data from the device into the comparatively small device-to-cloud messages received by IoT Hub. For instance, some applications or devices might be sending files or videos that would typically be processed in batches with services such as Hadoop/HDInsight or Azure Data Factory.

One example is `how-old.net`, a site that captures your picture and uses Azure Machine Learning to guess how old you are. A simple, and silly, example, but it provides a good example of getting real-time analysis of data.

In cases such as this example, files uploaded from a device or application can still use IoT Hub, especially to take advantage of the security and reliability functionality, meaning files are stored in Azure Blob Storage but they are managed via Azure IoT Hub, thus receiving the benefits provided, including notifications of uploaded files.

As shown in Figure 4-12, configuring Azure IoT Hub to work with uploaded files from devices is quite simple. In the Properties page of IoT Hub, select the File Upload link and then simply select the Azure Storage account and container in which the blobs will be uploaded, and then select whether to receive notifications of uploaded blobs and notification settings.



Storage container

Receive notifications for uploaded files: [✓ Off] On

SAS TTL

File notification settings

Default TTL

Maximum delivery count

Figure 4-12. *Configuring IoT Hub file uploads*

The code for uploading files is quite simple. The following method specifies a file, creates a connection to Azure IoT Hub, and then, like the previous example, uses the `UploadToBlobAsync` method on the `DeviceClient` class to upload the file to Azure Blob Storage:

```
private static async void SendToBlobAsync()
{
    string fileName = "<filename>";

    DeviceClient deviceClient = DeviceClient.CreateFromConnectionString
    (IOTHUBCONNECTIONSTRING);

    using (var sourceData = new FileStream(@"<filename>", FileMode.Open))
    {
        await deviceClient.UploadToBlobAsync(fileName, sourceData);
    }
}
```

Similarly, with a few lines of code your application can receive file upload notification messages from Azure IoT Hub. Both of these examples illustrate the ability and flexibility of device-to-cloud and cloud-to-device messaging functionality of IoT Hub.

Other Device Management Solutions

While this chapter focused on device management using Microsoft solutions, there are many open source tools that can be used to manage devices. Search in any browser search engine for "iot device management solutions" and a whole list of companies will be listed, including Opensensors.io, Allegro, Bosch, and Kaa. In fact, this list is just a small sampling. In a recent report, nearly 20 companies provide IoT device management, with Microsoft being one of them.

What makes many of these solutions interesting is that they are open source and allow you to deploy them yourself to virtual machines. With Kaa, for example, you can create a sandbox locally for development and experimentation on a small scale, and the sandbox can be either installed locally or deployed to a virtual machine.

A quick search on the Internet for IoT device management solutions comparison will turn up a few reports to look at, comparing security, protocols, and integration features. Some of the reports require you to pay to view them, but there are a few that are free and they do a decent job in their comparisons.

Most the solutions have HTTP/S and MQTT data collection protocol support, while only a few support AMQP. The majority of them offer REST API integration and also provide some flavor of real-time analytics. The support for real-time analytics is provided by a wide range of solutions including Apache Storm or a rules engine.

While many of these device management solutions will provide security, data collection protocols, and some analytics, the key point is that the Azure IoT Hub tracks events across categories of operations, allows you to monitor the status of operations within the hub, and integrates very well with Azure Stream Analytics for real-time streaming data insight, which, by the way, is the topic of the next chapter.

Summary

This pivotal chapter focused primarily on connecting the application developed in Chapter 2 with the IoT Hub created in Chapter 3. While the application was a simplistic one, you saw how easy it was to make the minor changes needed to the application in order to send messages to Azure IoT Hub.

In the real world, it will probably take a bit more work, but the goal with these chapters was simply to illustrate the process for connecting your IoT device to an IoT solution.

Lastly, this chapter discussed some design considerations and best practices for device-to-cloud communication, including device pull vs. send frequency and others.