d2c2d

Lab Workbook Three

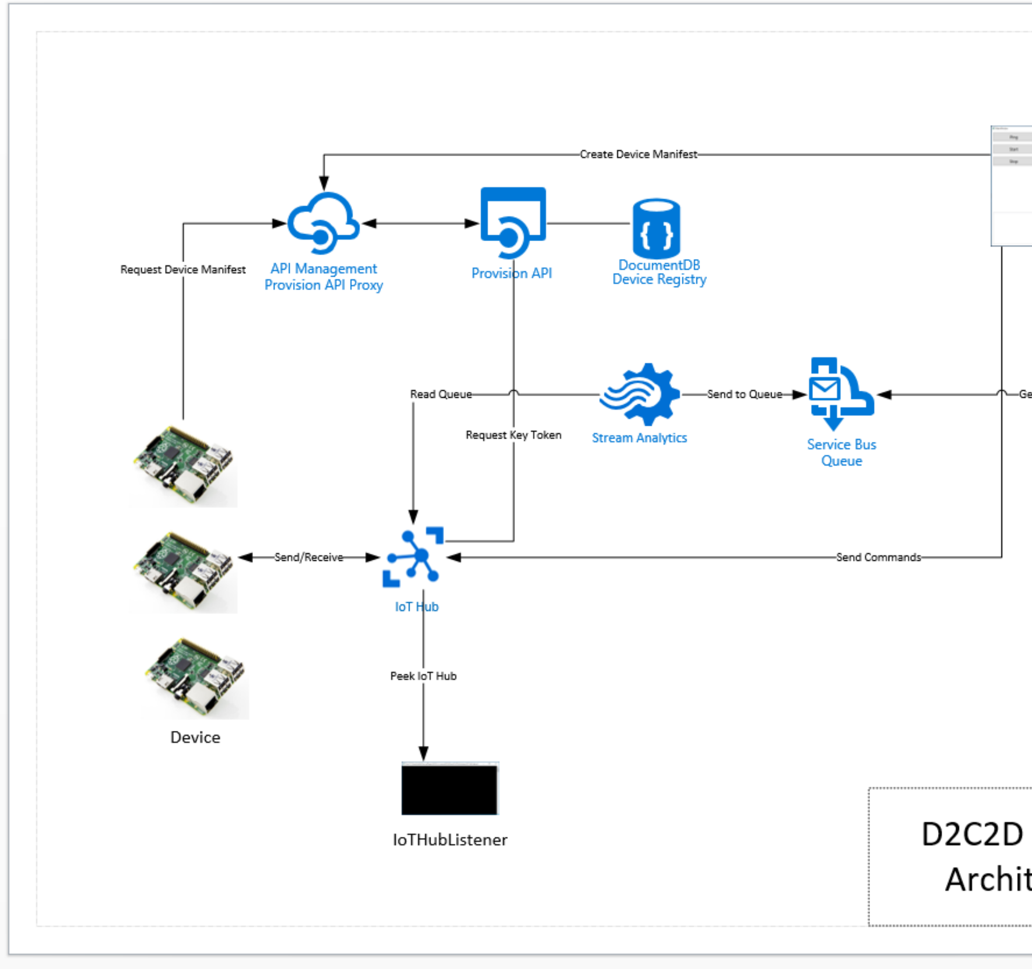
Device to Cloud to Device - a workshop for learning about Windows 10 Cor
development, Azure IoT Hub, Stream Analytics and automating Azure using

Workshop Overview

This training program provides foundational knowledge in how to architect and implem
solution using Windows 10 Core IoT hardware devices and Azure IoT Hub and Stream A
Device-to-Cloud and Cloud-to-Device communication patterns are discussed, designed
implemented using best practices.

At the conclusion of this workshop you will have provisioned, using PowerShell, an Azu
that contains IoT Hub, Stream Analytics Jobs that identify telemetry events and alarm s
Service Bus Namespace and set of message queues for backend integration.

You will also have developed a Windows 10 Core IoT application ("device") that sends t
receives incoming commands from the cloud as well as a real-time dashboard that can
directionally with the device (e.g., displaying telemetry readings and sending commanc
device).



Solution Architecture

The solution you will build and deploy consists of the following components:

- **Device** – a Windows 10 IoT Core IoT solution that dynamically connects to IoT h
heartbeat and climate telemetry as well as responds to command from a dashb
application can run on your local machine or be deployed to a Windows 10 Cor
as a Raspberry Pi.
- **Dashboard** – a Windows 10 WPF application that lists registered devices, maps
Bing Maps, and displays incoming device telemetry and alarms.
- **Provision API** – a ReST API that provides endpoints for device and device manif

- **IoT Hub –** IoT Hub provides device registration, incoming telemetry at scale, an message services
- **DocumentDb** – DocumentDb is a NoSQL database service that is used for mana Manifests, i.e., a Device Registry
- **Stream Analytics –** the solution leverages two Stream Analytics jobs, one that l incoming messages and another that identifies alarm states and routes those n second queue.

Lab Three Overview

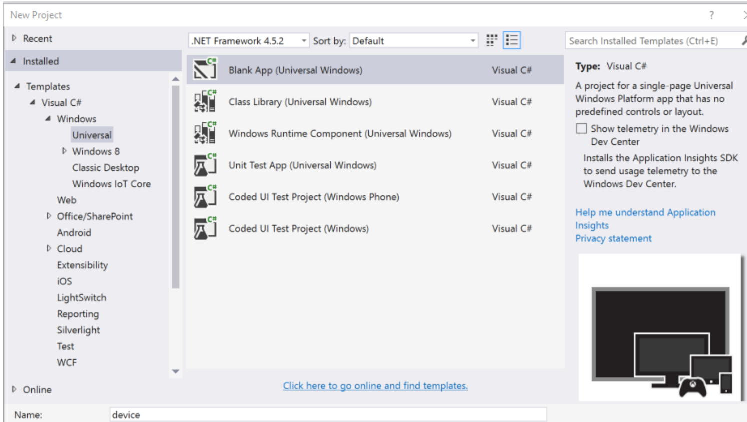
In Lab three you will begin the implementation of your device. The device must be able the Provision API to retrieve its manifest and then use the key stored there to connect that is working, the device can start to send a heartbeat message using the Ping Class y 1. Next we will update the Dashboard to receive these Ping Messages.

Lab

Step	*Details*
1	**Create the Device Solution**

The Device solution will be built on a .NET Core Universal App foundation. We Windows 10 Core IoT libraries and then reference the IoT and Message Mode needed for our implementation.

- Create a solution call 'device' in the device folder of the repo using th Universal Blank App template



Blank App (Universal Windows) Visual C#
Class Library (Universal Windows) Visual C#
Windows Runtime Component (Universal Windows) Visual C#
Unit Test App (Universal Windows) Visual C#
Coded UI Test Project (Windows Phone) Visual C#
Coded UI Test Project (Windows) Visual C#

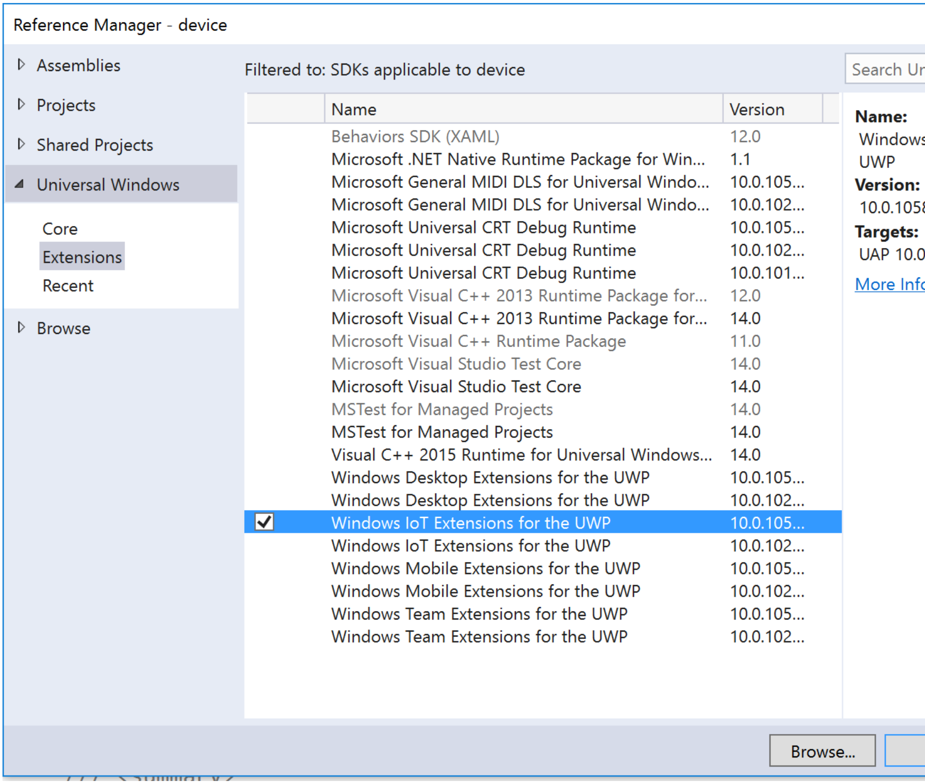
Type: Visual C#

A project for a single-page Universal Windows Platform app that has no predefined controls or layout.

□ Show telemetry in the Windows Dev Center

Installs the Application Insights SDK to send usage telemetry to the Windows Dev Center.

Name: device



Reference Manager - device

Filtered to: SDKs applicable to device

	Name	Version
	Behaviors SDK (XAML)	12.0
	Microsoft .NET Native Runtime Package for Win...	1.1
	Microsoft General MIDI DLS for Universal Windo...	10.0.105...
	Microsoft General MIDI DLS for Universal Windo...	10.0.102...
	Microsoft Universal CRT Debug Runtime	10.0.105...
	Microsoft Universal CRT Debug Runtime	10.0.102...
	Microsoft Universal CRT Debug Runtime	10.0.101...
	Microsoft Visual C++ 2013 Runtime Package for...	12.0
	Microsoft Visual C++ 2013 Runtime Package for...	14.0
	Microsoft Visual C++ Runtime Package	11.0
	Microsoft Visual Studio Test Core	14.0
	Microsoft Visual Studio Test Core	14.0
	MSTest for Managed Projects	14.0
	MSTest for Managed Projects	14.0
	Visual C++ 2015 Runtime for Universal Windows...	14.0
	Windows Desktop Extensions for the UWP	10.0.105...
	Windows Desktop Extensions for the UWP	10.0.102...
☑	Windows IoT Extensions for the UWP	10.0.105...
	Windows IoT Extensions for the UWP	10.0.102...
	Windows Mobile Extensions for the UWP	10.0.105...
	Windows Mobile Extensions for the UWP	10.0.102...
	Windows Team Extensions for the UWP	10.0.105...
	Windows Team Extensions for the UWP	10.0.102...

Name:
Windows
UWP
Version:
10.0.1058
Targets:
UAP 10.0
More Info

- Replace the entire contents of MainPage.xaml file with the markup be

```
<Page
     x:Class="device.MainPage"
     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
     xmlns:local="using:device"
     xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
     xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
     mc:Ignorable="d"
     Loaded="MainPage_OnLoaded">

     <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
         <Grid.RowDefinitions>
             <RowDefinition Height="50*"/>
             <RowDefinition Height="50*"/>
         </Grid.RowDefinitions>

         <TextBlock Grid.Row="0"
                    HorizontalAlignment="Center"
                    VerticalAlignment="Center"
                    FontSize="24">Windows 10 IoT Device</TextBlock>
         <StackPanel Grid.Row="1" Margin="10,10,10,10">
             <TextBox x:Name="Status" Margin="10" IsReadOnly="True" TextAlignmer
         </StackPanel>
```

    - search for Microsoft Azure Devices Client (be sure to select the Br the top of the NuGet window), and install that package.
    - search for Newtonsoft JSON and install that package.
- With the source set to the d2c2d NuGets folder location, add a refere MessageModelsNet5 NuGet package

Open the MainPage.xaml.cs file and add the following using statements:

```
using Looksfamiliar.d2c2d.MessageModels;
using Microsoft.Azure.Devices.Client;
using Newtonsoft.Json;
using System.Text;
using System.Net.Http;
using System.Threading.Tasks;
```

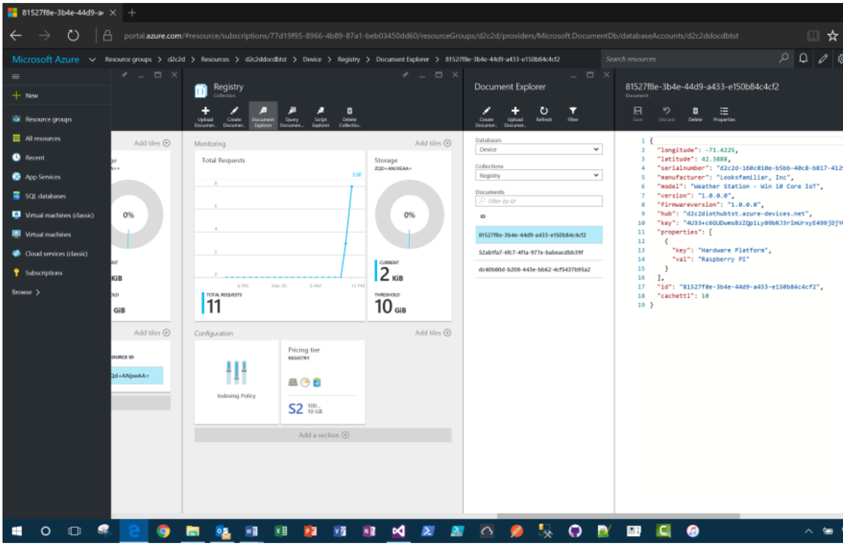
The device will need to know how to connect to the Provision API in order to r manifest.

- Add the following class member variables (within the MainPage class)

```
private const string DeviceSerialNumber = "[device-serial-number]";
private const string ProvisionApi = "[provision-api]" + DeviceSerialNum
private const string AckMessage = "Windows 10 Core IoT Device is Alive"

private static DeviceManifest _deviceManifest;
private static DeviceClient _deviceClient;

private static Task _pingTask;
private static Task _listenTask;
private static Task _telemetryTask;
private static bool _sendingTelemetry = false;
```

To retrieve the Device Serial Number for the lab, use the Azure Portal to navig DocumentDb and lookup the device you provisioned in Lab 2

- Click on DocumentDb in your Resource Group
- Click on the Device Database
- Click on the Registry Collection
- Click Document Explorer in the menu bar



Databases: Device

Collections: Registry

Documents

ID
81527f8e-3b4e-44d9-a433-e150b84c4cf2
52ab1fa7-6fc7-4f1a-977e-babeacdbb39f
dc40b80d-b208-443e-bb62-4cf5437b95a2

81527f8e-3b4e-44d9-a433-e150b84c4cf2

Document

```
{
  "longitude": -71.4225,
  "latitude": 42.3888,
  "serialnumber": "d2c2d-160c010e-b5bb-40c8-b817-412
  "manufacturer": "Looksfamiliar, Inc",
  "model": "Weather Station - Win 10 Core IoT",
  "version": "1.0.0.0",
  "firmwareversion": "1.0.0.0",
  "hub": "d2c2diothubtst.azure-devices.net",
  "key": "4U33+c6GUDwmsBzZQp1Ly09bNJ3rImUrxyE499jDjY
  "properties": [
    {
      "key": "Hardware Platform",
      "val": "Raspberry PI"
    }
  ],
  "id": "81527f8e-3b4e-44d9-a433-e150b84c4cf2",
  "cachettl": 10
}
```

- Copy the 'serialnumber' property and paste that into your source cod `serial-number]` initialization value; be sure to remove the surrounding
- Replace `[provision-api]` with the Provision microservice endpoint:

```
https://<prefix>provisiontapi<suffix>.azurewebsites.net/provision/dev
```

To invoke a ReST API from a Windows 10 Core IoT Device, you will use the Htt

- Add the method called GetDeviceManifest() to the MainPage class to Provision API and retrieve the device

```
private static async Task<DeviceManifest> GetDeviceManifest()
{
    var client = new HttpClient();
    var uriBuilder = new UriBuilder(ProvisionApi);
    var json = await client.GetStringAsync(uriBuilder.Uri);
    return JsonConvert.DeserializeObject<DeviceManifest>(json);
}
```

- Add a MainPage_OnLoaded() script that will call the GetDeviceManife then use the DeviceClient to connect to IoT Hub.

```
private async void MainPage_OnLoaded(object sender, RoutedEventArgs e)
{
    Status.Text = "Main Page Loaded";

    try
```

```
            Status.Text = connectionErr.Message;
        }

        StartPingTask(Status);
    }
```

Each activity that we want our device to provide is implemented as a backgrou
way the device can be doing multiple tasks at the same time such as issuing h
messages, sending telemetry, and receiving commands.

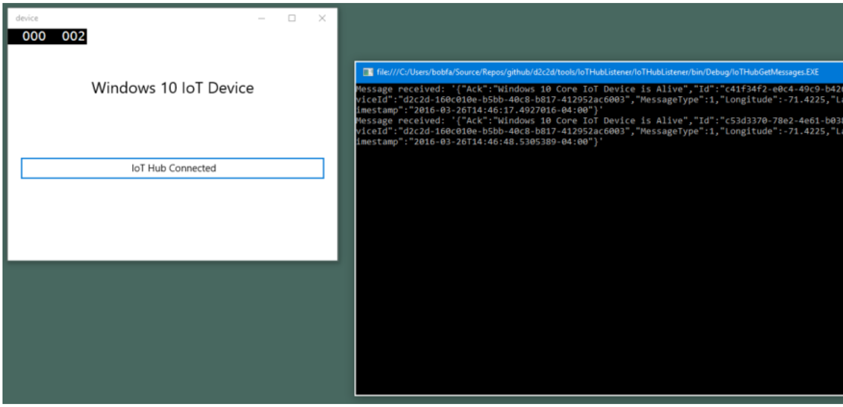
- Add the following implementation for the StartPingTask() method. Th
  off a background thread which is a forever loop that sends Ping messa
  every 10 seconds.

```
private static void StartPingTask(TextBox status)
{
    _pingTask = Task.Factory.StartNew(async () =>
    {
        while (true)
        {
            var ping = new Ping
            {
                Ack = AckMessage,
                Longitude = _deviceManifest.longitude,
                Latitude = _deviceManifest.latitude,
                DeviceId = _deviceManifest.serialnumber
            };

            var json = JsonConvert.SerializeObject(ping);

            var message = new Message(Encoding.ASCII.GetBytes(json));

            try
            {
                await _deviceClient.SendEventAsync(message);
            }
            catch (Exception err)
            {
                var errMessage = err.Message;
                status.Text = errMessage;
            }

            await Task.Delay(10000);
        }
    });
}
```

- Compile and run your code on the local machine; you should see a sir
  open, with the heading “Windows 10 IoT Device.” When the device is
  the IoT Hub in Azure, you should see a message populate the blue box

- Run IoT Hub Listener application by double-clicking IoTHubGetM
  This utility will show you the messages arriving at IoT Hub from
  application you just built.

  Recall, the interval is set via the last line of the MainPage class t


device
000 002
Windows 10 IoT Device
IoT Hub Connected
file:///C:/Users/bobfa/Source/Repos/github/d2c2d/tools/IoTHubListener/IoTHubListener/bin/Debug/IoTHubGetMessages.EXE

```
Message received: '{"Ack":"Windows 10 Core IoT Device is Alive","Id":"c41f34f2-e0c4-49c9-b42
viceId":"d2c2d-160c010e-b5bb-40c8-b817-412952ac6003","MessageType":1,"Longitude":-71.4225,"La
imestamp":"2016-03-26T14:46:17.4927016-04:00"}'
Message received: '{"Ack":"Windows 10 Core IoT Device is Alive","Id":"c53d3370-78e2-4e61-b03
viceId":"d2c2d-160c010e-b5bb-40c8-b817-412952ac6003","MessageType":1,"Longitude":-71.4225,"La
imestamp":"2016-03-26T14:46:48.5305389-04:00"}'
```

2 Provision a Stream Analytics Job

The messages coming from the device are first sent to IoT Hub. There they are
minimum of one day and a maximum of seven days. In order for the message
Dashboard, the messages need to be forwarded from IoT Hub to some type o
processing and/or storage location.

Stream Analytics is the service in Azure that defines scalable jobs to read mes
Hub, apply business rules and/or transformations, and route payloads to stora
application integration. The storage locations that Stream Analytics supports i
DocumentDb, SQL Database, Service Bus Queues and Topics, Event Hubs, Pow
Table storage.

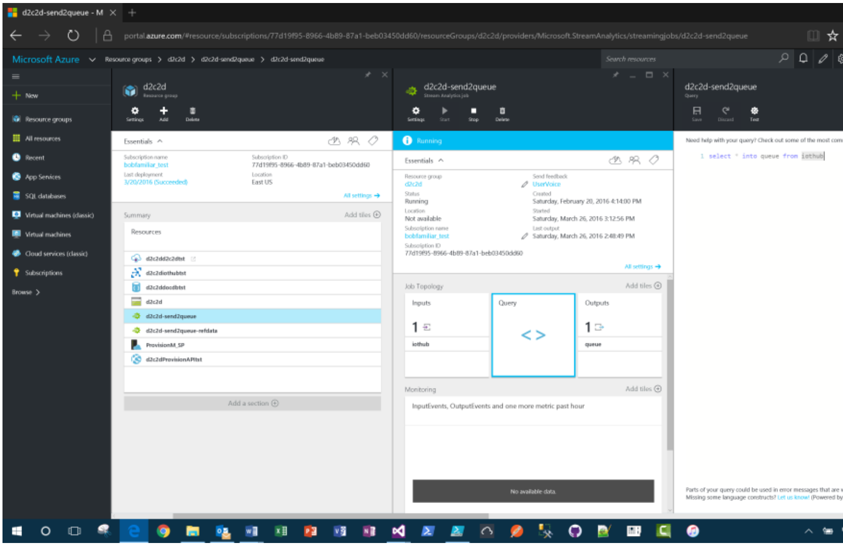
A Stream Analytics Job has one or more inputs and outputs and a query. Your
Analytics Job will be used to route all incoming messages to a Service Bus Que
step, you'll add a more sophisticated job that catches alarm states.

The PowerShell script that your will use has all this information already define

- The Input for the Stream Analytics Job is IoT Hub
- The Output for the Stream Analytics Job is the *messagedrop* Service B

Azure Location: [East US for example]
Prefix: [a unique prefix to be used in the naming of service componen
Suffix: [dev | tst | stg | prd]

This script will deploy a Stream Analytics job called ‘d2c2d-messages-queue’.
script provisions the Stream Analytics job by navigating to the Azure Portal Re
screen and clicking on that resource.



Subscription name: bobfamiliar_test
Subscription ID: 77d19f95-8966-4b89-87a1-beb03450dd60
Last deployment: 3/20/2016 (Succeeded)
Location: East US

Resources:
d2c2dd2c2dtst
d2c2diothubtst
d2c2ddocdbtst
d2c2d
d2c2d-send2queue
d2c2d-send2queue-refdata
ProvisionM_SP
d2c2dProvisionAPItst

Resource group: d2c2d
Send feedback: UserVoice
Status: Running
Created: Saturday, February 20, 2016 4:14:00 PM
Location: Not available
Started: Saturday, March 26, 2016 3:12:56 PM
Subscription name: bobfamiliar_test
Last output: Saturday, March 26, 2016 2:48:49 PM
Subscription ID: 77d19f95-8966-4b89-87a1-beb03450dd60

Inputs: 1 iothub
Outputs: 1 queue

Monitoring: InputEvents, OutputEvents and one more metric past hour
No available data.

Need help with your query? Check out some of the most com

```
1  select * into queue from iothub
```

Parts of your query could be used in error messages that are
Missing some language constructs? Let us know! (Powered by

3 Update the Dashboard to receive Ping Messages

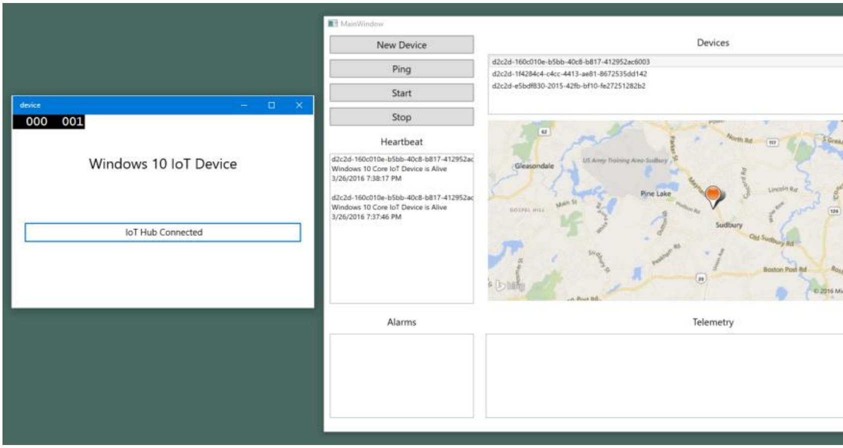
Now that the messages are being delivered to the Service Bus *messagedrop* q
add code to the Dashboard to receive those messages and display them on th

- Open the Dashboard Solution in Visual Studio
- Add the following code to the end of the MainWindow_OnLoaded() e
  background thread that polls for Ping Messages arriving on the *messa*
-

```
var messageTask = Task.Factory.StartNew(() =>
{
    while (true)
    {
        var message = _messageClient.Receive();
        var messageBody = string.Empty;
```

```
                    break;

                case MessageTypeEnum.Ping:

                    var ping = JsonConvert.DeserializeObject<Ping>(

                    Application.Current.Dispatcher.Invoke(
                           DispatcherPriority.Background,
                        new ThreadStart(delegate
                        {
                            // update the map
                            var location = new Map.Location(
                                ping.Latitude, ping.Longitude);
                            var pin = new Map.Pushpin {Location = l

                            MyMap.Children.Add(pin);
                            MyMap.Center = location;
                            MyMap.ZoomLevel = 12;
                            MyMap.SetView(location, 12);
                            MyMap.Focusable = true;
                            MyMap.Focus();

                            // update the Ping message display
                            var currHeartbeat = PingFeed.Text;
                            PingFeed.Text = string.Empty;
                            PingFeed.Text += $"{ping.DeviceId}\r\n"
                            PingFeed.Text += $"{ping.Ack}\r\n";
                            PingFeed.Text += $"{ping.Timestamp}\r\
                            PingFeed.Text += $"{currHeartbeat}";
                        }));

                    break;
             }
             message.Complete();
        }
        catch (Exception err)
        {
            Application.Current.Dispatcher.Invoke(
               DispatcherPriority.Background, new ThreadStart(deleg
            {
                var currTelemetry = TelemetryFeed.Text;
                TelemetryFeed.Text = string.Empty;
                TelemetryFeed.Text += $"{err.Message}\r\n";
                TelemetryFeed.Text += $"{messageBody}\r\n\r\n";
                TelemetryFeed.Text += $"{currTelemetry}\r\n\r\n";
                message.Abandon();
            }));
        }
    }
});
```

- Test the end-to-end message delivery
    - ○ Start the Device Application
    - ○ Start the Dashboard Application



Windows 10 IoT Device

IoT Hub Connected

d2c2d-160c010e-b5bb-40c8-b817-412952ac6003
d2c2d-1f4284c4-c4cc-4413-ae81-8672535dd142
d2c2d-e5bdf830-2015-42fb-bf10-fe27251282b2

d2c2d-160c010e-b5bb-40c8-b817-412952ac
Windows 10 Core IoT Device is Alive
3/26/2016 7:38:17 PM

d2c2d-160c010e-b5bb-40c8-b817-412952ac
Windows 10 Core IoT Device is Alive
3/26/2016 7:37:46 PM

4 **Congratulations! You have competed Lab 3**

Let’s review:

- You created the Device Solution and added the code to initialize the d
  Provision API, connect to IoT Hub, and send Ping messages
- You provisioned a Stream Analytics Job that takes incoming messages
  them on a Service Bus Queue
- You updated the Dashboard to receive the Ping messages