Data Analysis using Hadoop: Module 6, Lesson 5  
Getting Started with Event Hubs Hands-On Lab

## Overview

In this lab, you will create an Event Hubs in Azure, and send message to Event Hubs using sender program written by C language.

## Objectives

In this hands-on lab you will learn how to:

* How to provision the Event Hub in Azure portal
* How to send messages to Event Hubs

## Prerequisites

The following are required to complete this hands-on lab:

* A Microsoft Azure subscription
* Microsoft Visual Studio Community Edition for Windows
* You should have completed [Module 6 Lessons 1-5](https://github.com/MSFTImagine/computerscience/tree/master/Complimentary%20Course%20Content/Module6/Lessons) and [Lesson 4 Lab](https://github.com/MSFTImagine/computerscience/tree/master/Complimentary%20Course%20Content/Module6/Labs)

Note: The Azure portal is continually improved and changed. The steps in this exercise reflect the user interface of the Microsoft Azure portal at the time of writing, but may not match the latest design of portal.

## Exercises

This hands-on lab includes the following exercises:

* Exercise 1: Create an Event Hub
* Exercise 2: Send message to Event Hubs
* Exercise 3: Create a Storage Account
* Exercise 4: Receive messages with EventProcessorHost
* Exercise 5: Run the applications

## Introduction

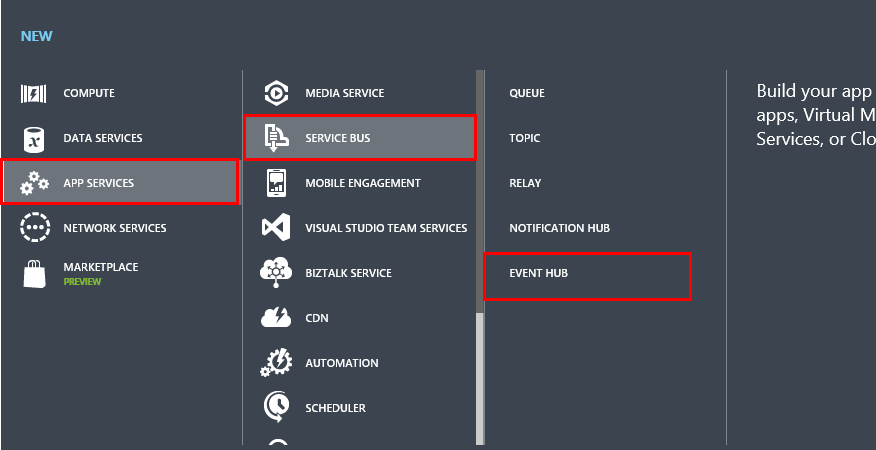
## Event Hubs is a service that processes large amounts of event data (telemetry) from connected devices and applications. After you collect data into Event Hubs, you can store the data using a storage cluster or transform it using a real-time analytics provider. This large-scale event collection and processing capability is a key component of modern application architectures including the Internet of Things (IoT).

## This tutorial shows how to use the Azure classic portal to create an Event Hub. It also shows you how to collect messages into an Event Hub using a console application written in C#, and how to retrieve them in parallel using the C# Event Processor Host library.

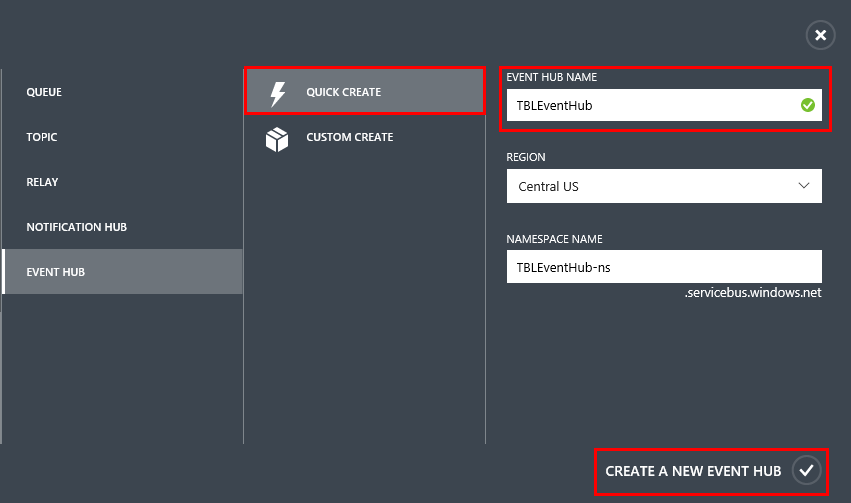
## Exercise 1: Create an Event Hub

The first task you have to perform is to creation of Event Hub.

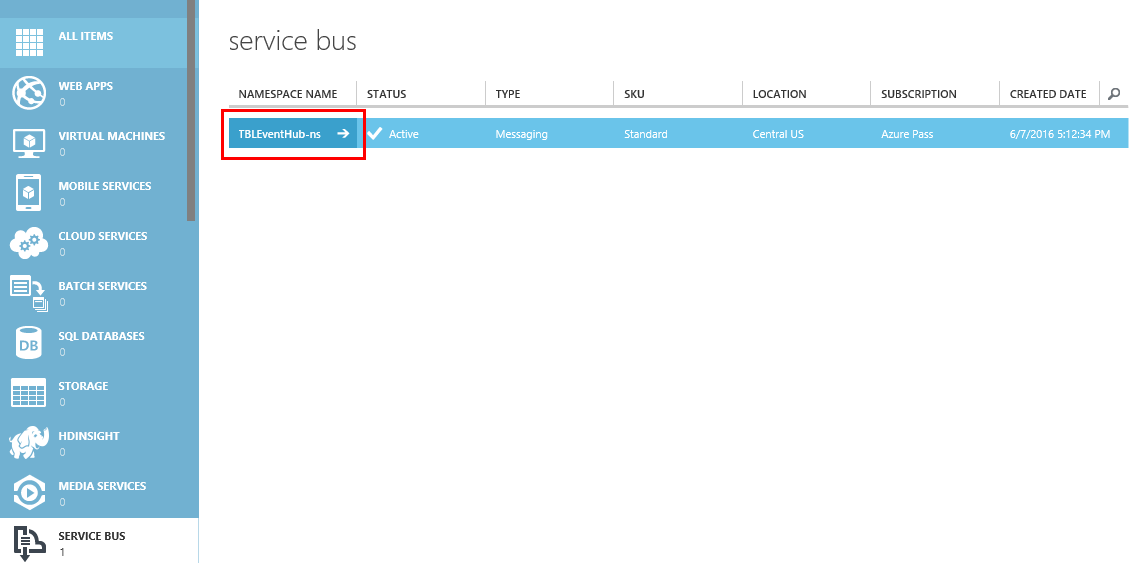
1. In a web browser, navigate to Azure classic portal, https://manage.windowsazure.com/. Sign into the portal using your subscription, and click NEW at the bottom to create an Event Hub.

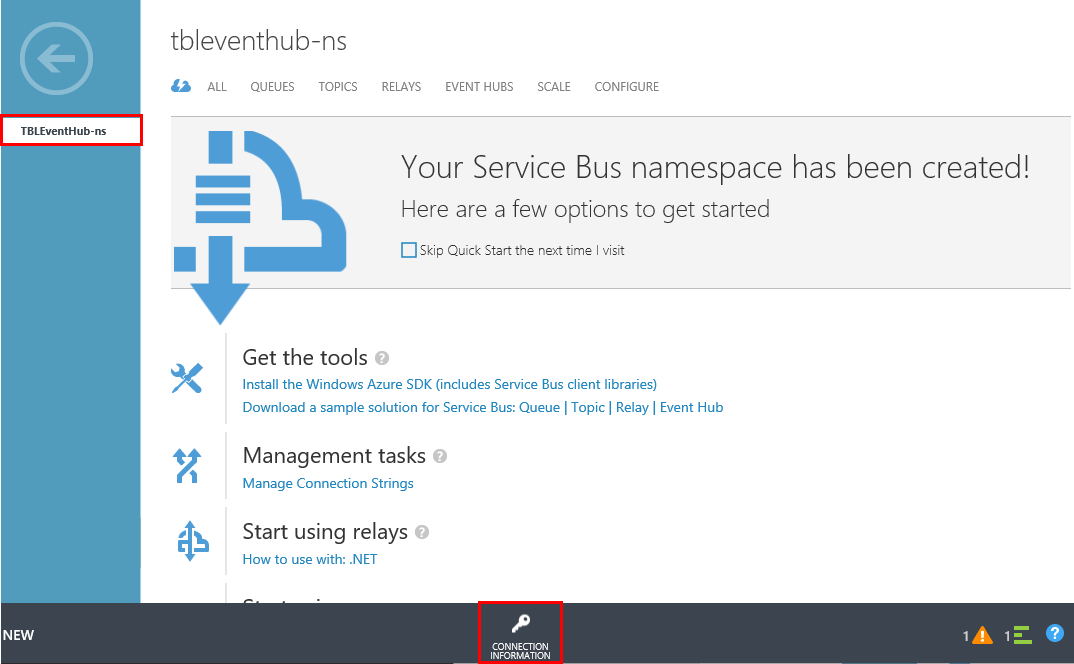


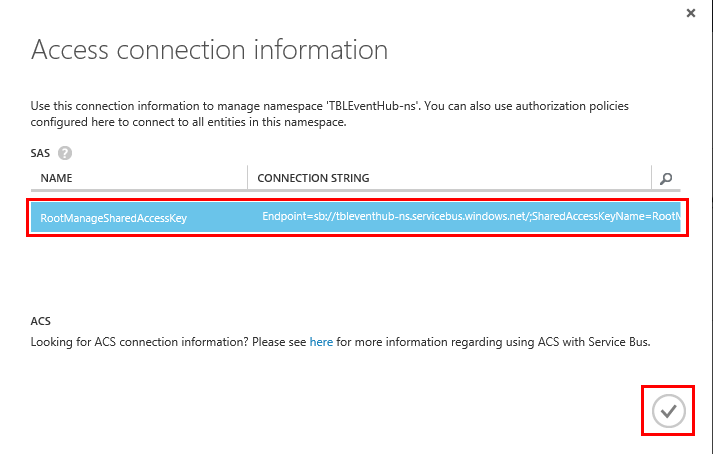
1. Click on App Service, then Service Bus, then Event Hub and Quick Create. This quickly create an Event Hub with default configuration settings.
2. Enter a name for your Event Hub, select your desired region, and the click “Create a new Event Hub”



1. The portal creates a namespace for your Event Hub. Click this namespace (in this lab, TBLEventHub-ns).



1. When the Event Hub has been created successfully, the Bus namespace also has been created. At the bottom of the page, click “Connection Information”. 
2. Show the Access connection information for Event Hub. Click the copy button to copy the RootManageSharedAccessKey connection string to the clipboard. Save this information to use later.

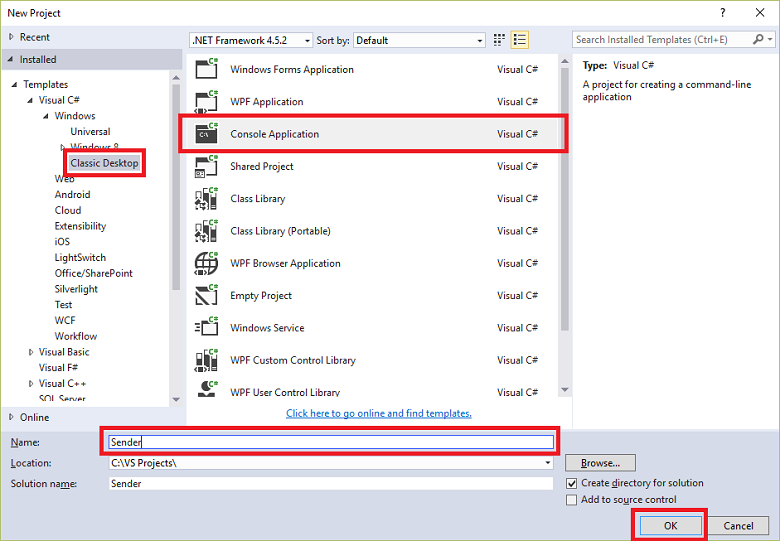


Now, students have the connection strings for send and receive events

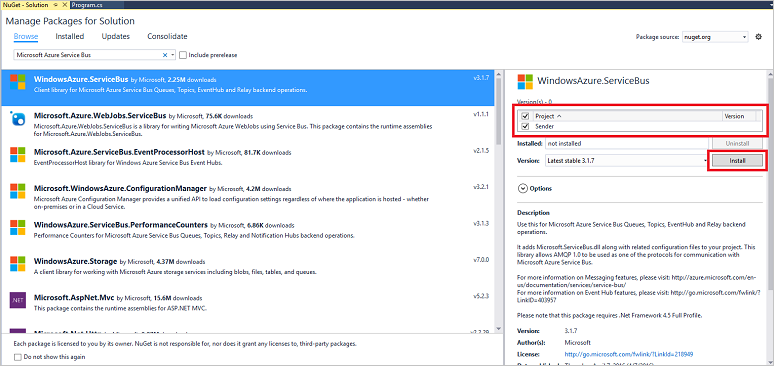
## Exercise 2: Send messages to Event Hubs.

In this section, you'll write a Windows console app that sends events to your Event Hub.

1. In Visual Studio, create a new Visual C# Desktop App project using the Console Application project template. Name the project Sender.



1. In Solution Explorer, right-click the solution, and then click Manage NuGet Packages for Solution.
2. Click the Browse tab, then search for Microsoft Azure Service Bus. Ensure that the project name (Sender) is specified in the Version(s) box. Click Install, and accept the terms of use.



This downloads, installs, and adds a reference to the Azure Service Bus library NuGet package.

1. Add the following using statements at the top of the Program.cs file:

using System.Threading;

using Microsoft.ServiceBus.Messaging;

1. Add the following fields to the Program class, substituting the placeholder values with the name of the Event Hub you created in the previous section, and the namespace-level connection string you saved previously.

static string eventHubName = "{Event Hub name}";

static string connectionString = "{send connection string}";

The Event Hub Name is TBLEventHub and the connection string is the RootManageSharedAccessKey connection string that students saved earlier.

1. Add the following method to the Program class:

static void SendingRandomMessages()

{

var eventHubClient = EventHubClient.CreateFromConnectionString(connectionString, eventHubName);

while (true)

{

try

{

var message = Guid.NewGuid().ToString();

Console.WriteLine("{0} > Sending message: {1}", DateTime.Now, message);

eventHubClient.Send(new EventData(Encoding.UTF8.GetBytes(message)));

}

catch (Exception exception)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("{0} > Exception: {1}", DateTime.Now, exception.Message);

Console.ResetColor();

}

Thread.Sleep(200);

}

}

This method continuously sends events to your Event Hub with a 200ms delay.

1. Finally, add the following lines to the Main method:

Console.WriteLine("Press Ctrl-C to stop the sender process");

Console.WriteLine("Press Enter to start now");

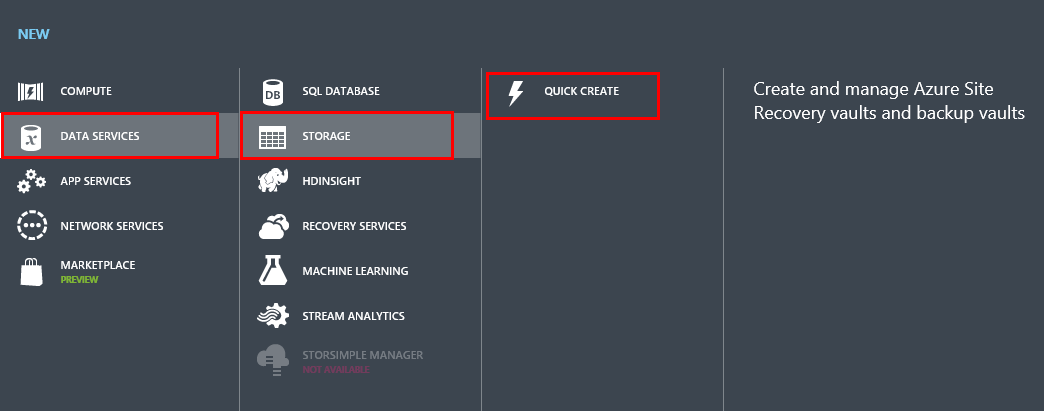
Console.ReadLine();

SendingRandomMessages();

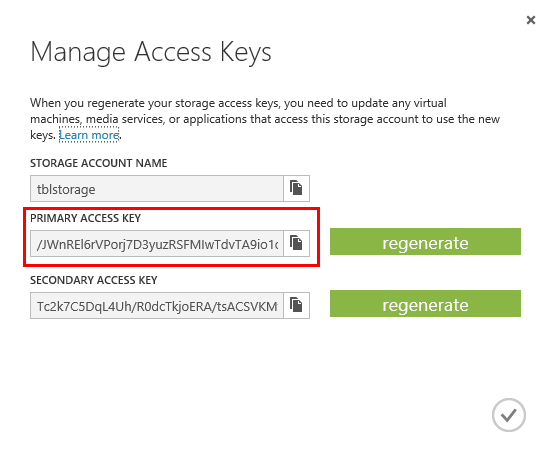
## Exercise 3: Create a Storage Account

In order to receive events from the Event Hub, students need to a storage account.

1. Click NEW at the bottom of page, and click “Data Service, then Storage, then “Quick Create”, and then type a name for your storage account. Select your region, and then click “Create Storage Account”



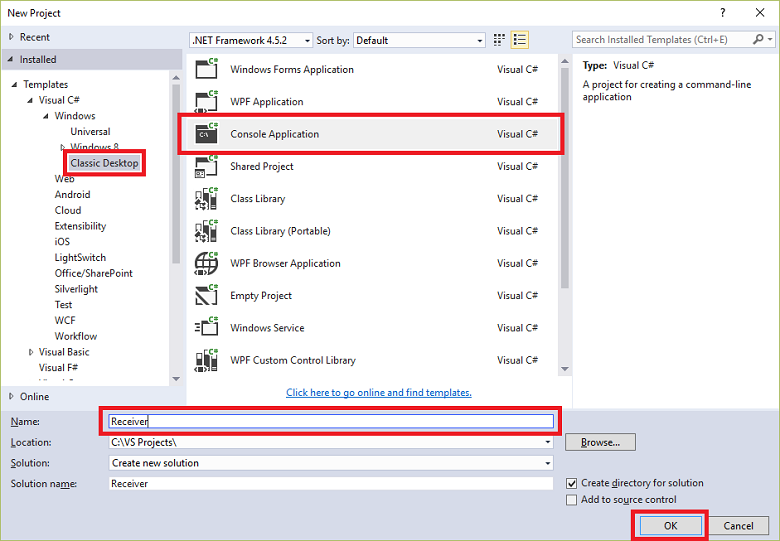
1. Click the newly created storage account, and then click “Manage Access Keys”. Copy the primary access key to use later in this lab.



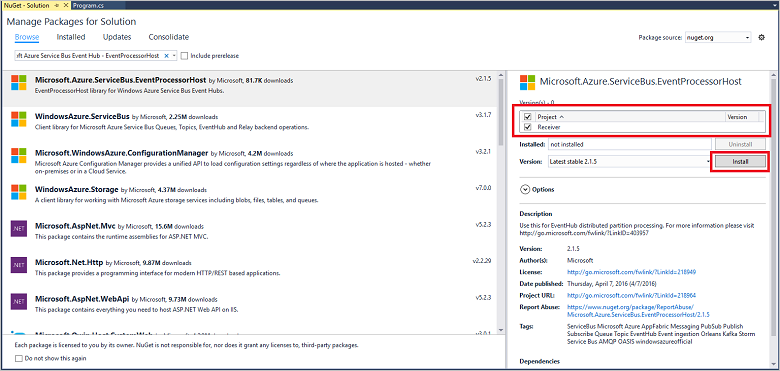
## Exercise 3: Receive Messages with EventProcessorHost

EventProcessorHost is a .NET class that simplifies receiving events from Event Hubs by managing persistent checkpoints and parallel receives from those Event Hubs. Using EventProcessorHost, you can split events across multiple receivers, even when hosted in different nodes. This example shows how to use EventProcessorHost for a single receiver.

1. In Visual Studio, create a new Visual C# Desktop App project using the Console Application project template. Name the project Receiver.

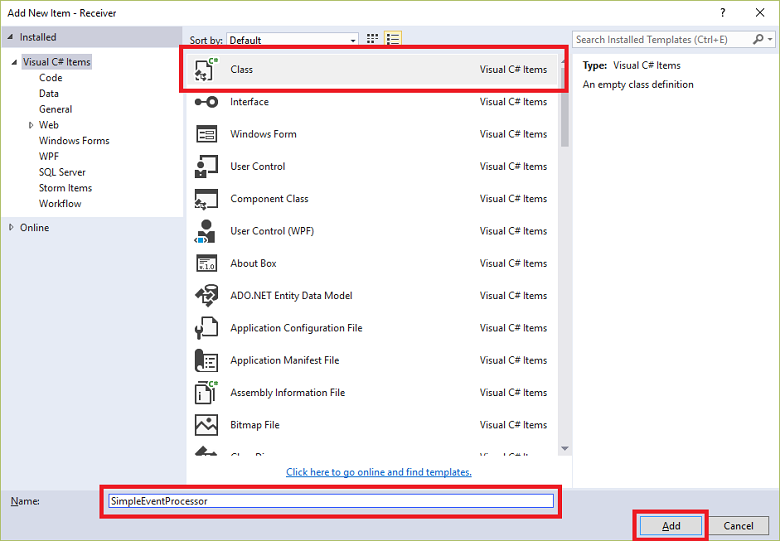


1. In Solution Explorer, right-click the solution, and then click Manage NuGet Packages for Solution.
2. Click the Browse tab, then search for Microsoft Azure Service Bus Event Hub - EventProcessorHost. Ensure that the project name (Receiver) is specified in the Version(s) box. Click Install, and accept the terms of use.



This downloads, installs, and adds a reference to the Azure Service Bus Event Hub - EventProcessorHost NuGet package, with all its dependencies.

1. Right-click the Receiver project, click Add, and then click Class. Name the new class SimpleEventProcessor, and then click Add to create the class.



1. Add the following statements at the top of the SimpleEventProcessor.cs file:

using Microsoft.ServiceBus.Messaging;

using System.Diagnostics;

1. Substitute the following code for the body of the class:

class SimpleEventProcessor : IEventProcessor

{

Stopwatch checkpointStopWatch;

async Task IEventProcessor.CloseAsync(PartitionContext context, CloseReason reason)

{

Console.WriteLine("Processor Shutting Down. Partition '{0}', Reason: '{1}'.", context.Lease.PartitionId, reason);

if (reason == CloseReason.Shutdown)

{

await context.CheckpointAsync();

}

}

Task IEventProcessor.OpenAsync(PartitionContext context)

{

Console.WriteLine("SimpleEventProcessor initialized. Partition: '{0}', Offset: '{1}'", context.Lease.PartitionId, context.Lease.Offset);

this.checkpointStopWatch = new Stopwatch();

this.checkpointStopWatch.Start();

return Task.FromResult<object>(null);

}

async Task IEventProcessor.ProcessEventsAsync(PartitionContext context, IEnumerable<EventData> messages)

{

foreach (EventData eventData in messages)

{

string data = Encoding.UTF8.GetString(eventData.GetBytes());

Console.WriteLine(string.Format("Message received. Partition: '{0}', Data: '{1}'",

context.Lease.PartitionId, data));

}

//Call checkpoint every 5 minutes, so that worker can resume processing from 5 minutes back if it restarts.

if (this.checkpointStopWatch.Elapsed > TimeSpan.FromMinutes(5))

{

await context.CheckpointAsync();

this.checkpointStopWatch.Restart();

}

}

}

This class will be called by the EventProcessorHost to process events received from the Event Hub. Note that the SimpleEventProcessor class uses a stopwatch to periodically call the checkpoint method on the EventProcessorHost context. This ensures that, if the receiver is restarted, it will lose no more than five minutes of processing work.

1. In the Program class, add the following using statement at the top of the file:

using Microsoft.ServiceBus.Messaging;

Then, replace the Main method in the Program class with the following code, substituting the Event Hub name and the namespace-level connection string that you saved previously, and the storage account and key that you copied in the previous sections.

static void Main(string[] args)

{

string eventHubConnectionString = "{Event Hub connection string}";

string eventHubName = "{Event Hub name}";

string storageAccountName = "{storage account name}";

string storageAccountKey = "{storage account key}";

string storageConnectionString = string.Format("DefaultEndpointsProtocol=https;AccountName={0};AccountKey={1}", storageAccountName, storageAccountKey);

string eventProcessorHostName = Guid.NewGuid().ToString();

EventProcessorHost eventProcessorHost = new EventProcessorHost(eventProcessorHostName, eventHubName, EventHubConsumerGroup.DefaultGroupName, eventHubConnectionString, storageConnectionString);

Console.WriteLine("Registering EventProcessor...");

var options = new EventProcessorOptions();

options.ExceptionReceived += (sender, e) => { Console.WriteLine(e.Exception); };

eventProcessorHost.RegisterEventProcessorAsync<SimpleEventProcessor>(options).Wait();

Console.WriteLine("Receiving. Press enter key to stop worker.");

Console.ReadLine();

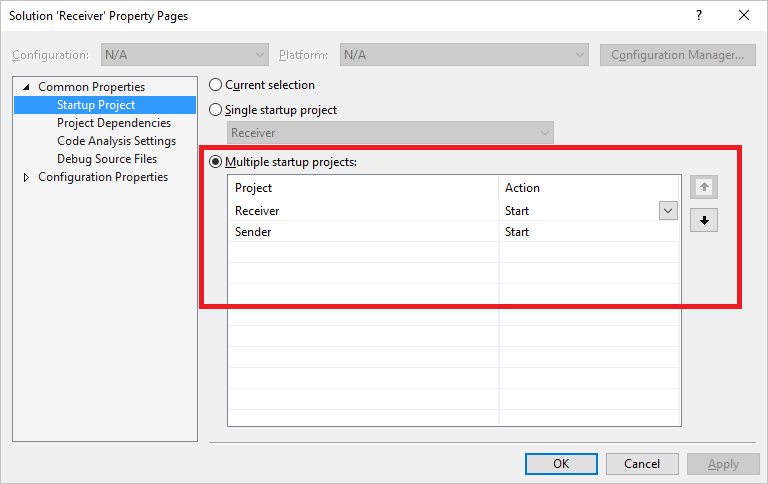
eventProcessorHost.UnregisterEventProcessorAsync().Wait();

}

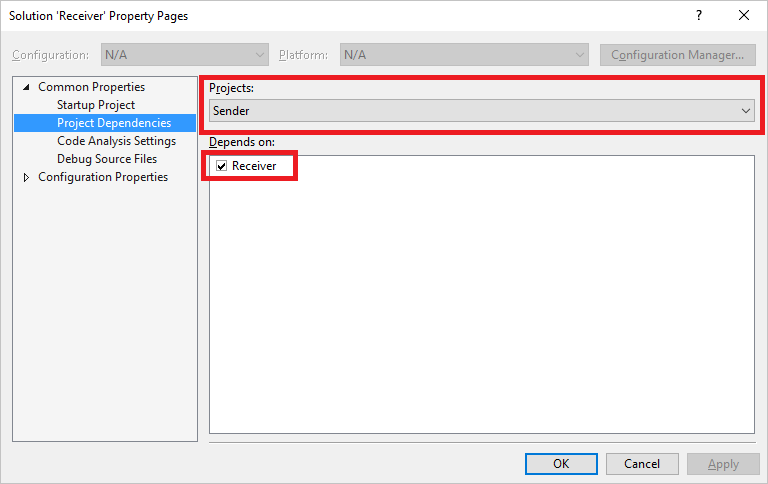
## Exercise 4: Run the applications

Now, run the applications:

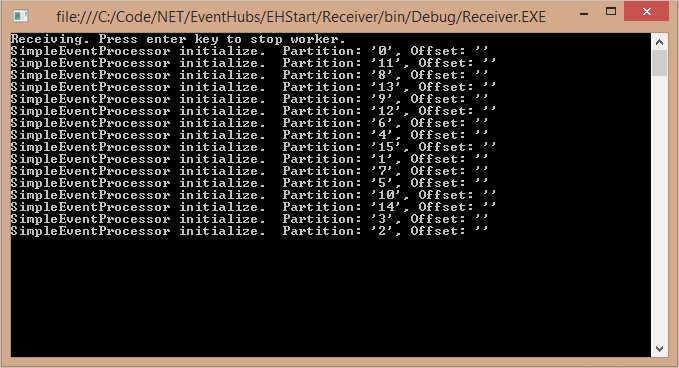
1. From within Visual Studio, open the Receiver project you created earlier.
2. Right-click the Receiver solution, then click Add, and then click Existing Project.
3. Locate the existing Sender.csproj file, then double-click it to add it to the solution.
4. Again, right-click the Receiver solution and then click Properties. The Receiver property page is displayed.
5. Click Startup Project, then click the Multiple startup projects button. Set the Action box for both the Receiver and Sender projects to Start.



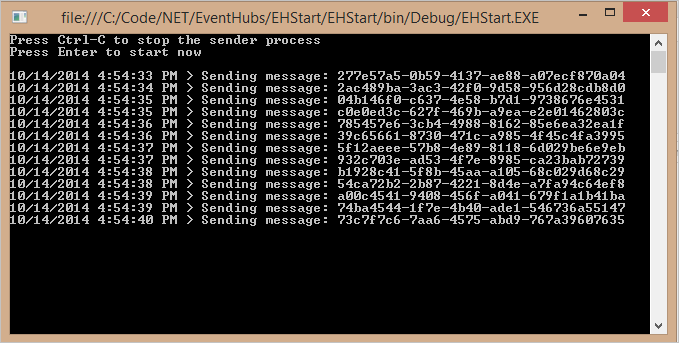
1. Click Project Dependencies. In the Projects box, click Sender. In the Depends on box, make sure Receiver is checked.



1. Click OK to dismiss the Properties dialog.
2. Press F5 to run the Receiver project from within Visual Studio, then wait for it to start the receivers for all the partitions.



1. The Sender project will run automatically. Press Enter in the console window, and see the events appear in the receiver window.



Press Ctrl+C in the Sender window to end the Sender application, then press Enter in the Receiver window to shut down that application.

## Summary

In this hands-on lab, you learned how to:

* How to provision the Event Hub in Azure portal.
* How to send messages to Event Hubs.
* How to provision a Storage Account in Azure portal.
* How to receive messages at the Event Hubs.