

[< Previous](#)Unit 6 of 10 [Next >](#)✓ 100 XP 

Exercise: Send and receive message from a Service Bus queue by using .NET.

30 minutes

In this exercise you learn how to:

- Create a Service Bus namespace, and queue, using the Azure CLI.
- Create a .NET console application to send and receive messages from the queue.

Prerequisites

- An **Azure account** with an active subscription. If you don't already have one, you can sign up for a free trial at <https://azure.com/free> .
- [Visual Studio Code](#) on one of the [supported platforms](#) .
- The [C# extension](#) for Visual Studio Code.
- [.NET 6](#) is the target framework for the exercise.

Sign in to Azure

In this section, you open your terminal and create some variables used throughout the rest of the exercise to make command entry, and unique resource name creation, a bit easier.

1. Launch the [Azure Cloud Shell](#) and select **Bash** and the environment.
2. Create variables used in the Azure CLI commands. Replace `<myLocation>` with a region near you.

```
Bash
```

```
myLocation=<myLocation>  
myNamespaceName=az204svcbus$RANDOM
```

Create Azure resources

1. Create a resource group to hold the Azure resources you're creating.

Bash

```
az group create --name az204-svcbus-rg --location $myLocation
```

2. Create a Service Bus messaging namespace. The following command creates a namespace using the variable you created earlier. The operation takes a few minutes to complete.

Bash

```
az servicebus namespace create \  
  --resource-group az204-svcbus-rg \  
  --name $myNamespaceName \  
  --location $myLocation
```

3. Create a Service Bus queue

Bash

```
az servicebus queue create --resource-group az204-svcbus-rg \  
  --namespace-name $myNamespaceName \  
  --name az204-queue
```

Retrieve the connection string for the Service Bus Namespace

1. Open the Azure portal and navigate to the **az204-svcbus-rg** resource group.
2. Select the **az204svcbus** resource you created.

3. Select **Shared access policies** in the **Settings** section, then select the **RootManageSharedAccessKey** policy.
4. Copy the **Primary Connection String** from the dialog box that opens up and save it to a file, or leave the portal open and copy the key when needed.

Create console app to send messages to the queue

1. Open a local terminal and create, and change in to, a directory named *az204svcbus* and then run the command to launch Visual Studio Code.

```
Bash
```

```
code .
```

2. Open the terminal in Visual Studio Code by selecting **Terminal > New Terminal** in the menu bar and run the following commands to create the console app and add the **Azure.Messaging.ServiceBus** package.

```
Bash
```

```
dotnet new console  
dotnet add package Azure.Messaging.ServiceBus
```

3. In *Program.cs*, add the following `using` statements at the top of the file after the current `using` statement.

```
C#
```

```
using Azure.Messaging.ServiceBus;
```

4. Add the following variables to the code and set the `connectionString` variable to the connection string that you obtained earlier.

```
C#
```

```
// connection string to your Service Bus namespace
string connectionString = "<CONNECTION STRING>";

// name of your Service Bus topic
string queueName = "az204-queue";
```

5. Add the following code below the variables you just added. See code comments for details.

C#

```
// the client that owns the connection and can be used to create
senders and receivers
ServiceBusClient client;

// the sender used to publish messages to the queue
ServiceBusSender sender;

// Create the clients that we'll use for sending and processing mes-
sages.
client = new ServiceBusClient(connectionString);
sender = client.CreateSender(queueName);

// create a batch
using ServiceBusMessageBatch messageBatch = await sender.CreateMes-
sageBatchAsync();

for (int i = 1; i <= 3; i++)
{
    // try adding a message to the batch
    if (!messageBatch.TryAddMessage(new ServiceBusMessage($"Message
{i}")))
    {
        // if an exception occurs
        throw new Exception($"Exception {i} has occurred.");
    }
}

try
{
    // Use the producer client to send the batch of messages to the
Service Bus queue
    await sender.SendMessagesAsync(messageBatch);
    Console.WriteLine($"A batch of three messages has been published
to the queue.");
}
finally
```

```
{  
    // Calling DisposeAsync on client types is required to ensure  
    that network  
    // resources and other unmanaged objects are properly cleaned  
    up.  
    await sender.DisposeAsync();  
    await client.DisposeAsync();  
}  
  
Console.WriteLine("Follow the directions in the exercise to review  
the results in the Azure portal.");  
Console.WriteLine("Press any key to continue");  
Console.ReadKey();
```

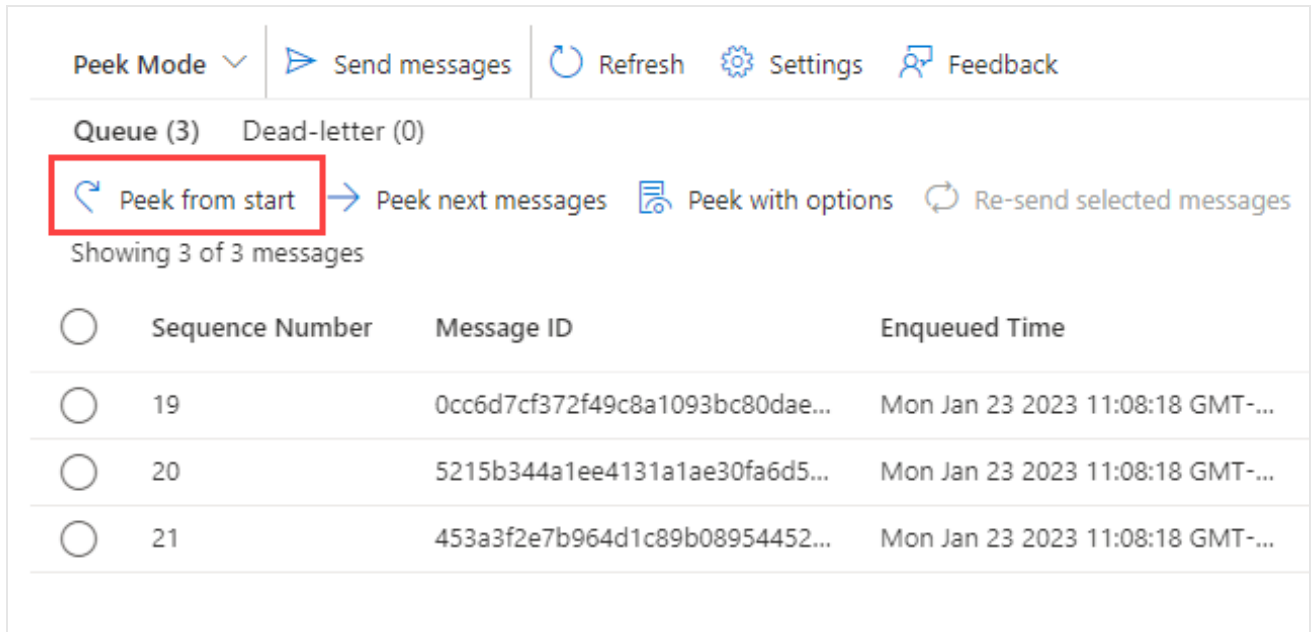
6. Save the file and run the `dotnet build` command to ensure there are no errors.
7. Run the program using the `dotnet run` command and wait for the following confirmation message. Then press any key to exit the program.

Bash

A batch of three messages has been published to the queue.

Review results

1. Sign in to the Azure portal and navigate to your Service Bus namespace. Select the **Service Bus Explorer** in the Service Bus Queue navigation pane.
2. Select **Peek from start** and the three messages that were sent appear.



Peek Mode ▾ Send messages Refresh Settings Feedback

Queue (3) Dead-letter (0)

Peek from start Peek next messages Peek with options Re-send selected messages

Showing 3 of 3 messages

	Sequence Number	Message ID	Enqueued Time
<input type="radio"/>	19	0cc6d7cf372f49c8a1093bc80dae...	Mon Jan 23 2023 11:08:18 GMT-...
<input type="radio"/>	20	5215b344a1ee4131a1ae30fa6d5...	Mon Jan 23 2023 11:08:18 GMT-...
<input type="radio"/>	21	453a3f2e7b964d1c89b08954452...	Mon Jan 23 2023 11:08:18 GMT-...

Update project to receive messages to the queue

In this section, you update the program to receive messages from the queue.

1. Add the following code at the end of the existing code. See code comments for details.

C#

```
ServiceBusProcessor processor;
client = new ServiceBusClient(connectionString);

// create a processor that we can use to process the messages
processor = client.CreateProcessor(queueName, new ServiceBusProcessorOptions());

try
{
    // add handler to process messages
    processor.ProcessMessageAsync += MessageHandler;

    // add handler to process any errors
    processor.ProcessErrorAsync += ErrorHandler;

    // start processing
    await processor.StartProcessingAsync();
}
```

```
        Console.WriteLine("Wait for a minute and then press any key to  
end the processing");  
        Console.ReadKey();  
  
        // stop processing  
        Console.WriteLine("\nStopping the receiver...");  
        await processor.StopProcessingAsync();  
        Console.WriteLine("Stopped receiving messages");  
    }  
    finally  
    {  
        // Calling DisposeAsync on client types is required to ensure  
        // that network  
        // resources and other unmanaged objects are properly cleaned  
        // up.  
        await processor.DisposeAsync();  
        await client.DisposeAsync();  
    }  
  
    // handle received messages  
    async Task MessageHandler(ProcessMessageEventArgs args)  
    {  
        string body = args.Message.Body.ToString();  
        Console.WriteLine($"Received: {body}");  
  
        // complete the message. messages is deleted from the queue.  
        await args.CompleteMessageAsync(args.Message);  
    }  
  
    // handle any errors when receiving messages  
    Task ErrorHandler(ProcessErrorEventArgs args)  
    {  
        Console.WriteLine(args.Exception.ToString());  
        return Task.CompletedTask;  
    }  
}
```

2. Use the `dotnet build` command to ensure there are no errors.
3. Use the `dotnet run` command to run the application. It sends three more messages to the queue and then retrieve all six messages. Press any key to stop the receiver and the application.

Bash

```
Wait for a minute and then press any key to end the processing  
Received: Message 1
```

```
Received: Message 2
Received: Message 3
Received: Message 1
Received: Message 2
Received: Message 3

Stopping the receiver...
Stopped receiving messages
```

⚠ Note

Since the application sent two batches of messages before retrieving them, you should see two batches of three messages represented in the output.

4. Return to the portal and select **Peek from start** again. Notice that no messages appear in the queue since we've retrieved them all.

Clean up resources

When the resources are no longer needed, you can use the `az group delete` command in the Azure Cloud Shell to remove the resource group.

Bash

```
az group delete --name az204-svcbus-rg --no-wait
```

Next unit: Explore Azure Queue Storage

[Continue >](#)