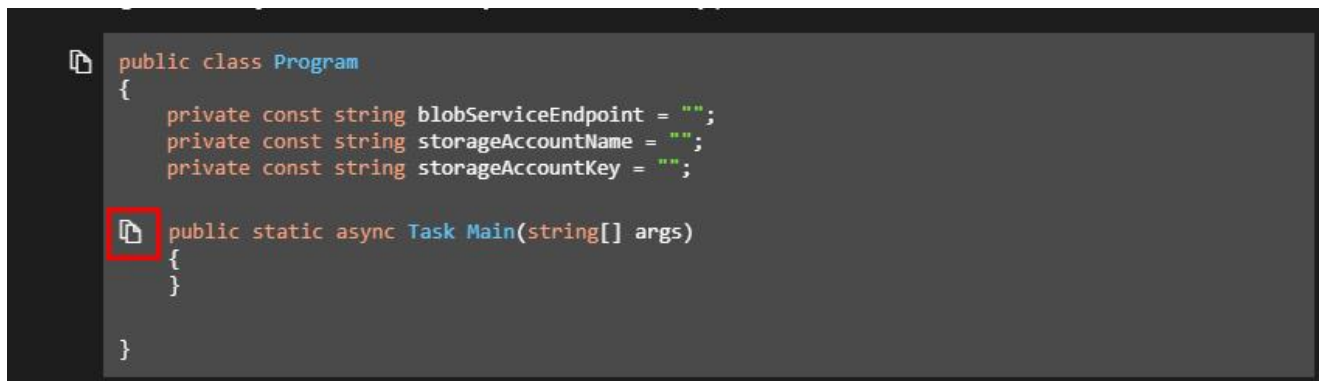


Do not click nested Typetext icons. Doing so will produce the wrong output.



```
public class Program
{
    private const string blobServiceEndpoint = "";
    private const string storageAccountName = "";
    private const string storageAccountKey = "";

    public static async Task Main(string[] args)
    {
    }
}
```

At the end of this lab, you can skip the **Clean Up** exercise directing you to remove the resources from your Subscription or Resource Group(s). The clean up is handled automatically, after ending your lab.

Lab: Asynchronously processing messages by using Azure Queue Storage

Student lab manual

Lab scenario

You're studying various ways to communicate between isolated service components in Microsoft Azure, and you have decided to evaluate the Azure Storage service and its Queue service offering. As part of this evaluation, you'll build a prototype application in .NET that can send and receive messages so that you can measure the complexity involved in using this service. To help you with your evaluation, you've also decided to use Azure Storage Explorer as the queue message producer/consumer throughout your tests.

Objectives

After you complete this lab, you will be able to:

- Add **Azure.Storage** libraries from NuGet.
- Create a queue in .NET.
- Produce a new message in the queue by using .NET.
- Consume a message from the queue by using .NET.
- Manage a queue by using Storage Explorer.

Lab setup

- Estimated time: **45 minutes**

Instructions

Before you start

Sign in to the lab virtual machine

Ensure that you're signed in to your Windows 10 virtual machine by using the following credentials:

- Username: **Admin**
- Password: **Pa55w.rd**

Review the installed applications

Find the taskbar on your Windows 10 desktop. The taskbar contains the icons for the applications that you'll use in this lab:

- Microsoft Edge
- Visual Studio Code
- Storage Explorer

Exercise 1: Create Azure resources

Task 1: Open the Azure portal

1. Sign in to the Azure portal (<https://portal.azure.com>).
2. If this is your first time signing in to the Azure portal, you'll notice a dialog box offering a tour of the portal. Select **Get Started** to skip the tour.

Task 2: Create a Storage account

1. Create a new storage account with the following details:
 - New resource group: **AsyncProcessor**
 - Name: **asyncstor*[yourname]***
 - Location: **(US) East US**
 - Performance: **Standard**
 - Account kind: **StorageV2 (general purpose v2)**
 - Replication: **Locally-redundant storage (LRS)**
 - Access tier: **Hot**

Note: Wait for Azure to finish creating the storage account before you move forward with the lab. You'll receive a notification when the account is created.

2. Find the **Access Keys** blade of your newly created storage account instance.
3. Record the value of the **Connection string** text box. You'll use this value later in this lab.

Review

In this exercise, you created a new Storage account that you'll use through the remainder of the lab.

Exercise 2: Configure the Azure Storage SDK in a .NET project

Task 1: Create a .NET project

1. Using Visual Studio Code, open the **Allfiles (F):\Allfiles\Labs\11\Starter\MessageProcessor** folder.
2. Using a terminal, create a new .NET project named **MessageProcessor** in the current folder:

```
dotnet new console --name MessageProcessor --output .
```

Note: The **dotnet new** command will create a new **console** project in a folder with the same name as the project.

3. Using the same terminal, import version 12.0.0 of **Azure.Storage.Queues** from NuGet:

```
dotnet add package Azure.Storage.Queues --version 12.0.0
```

Note: The **dotnet add package** command will add the **Azure.Storage.Queues** package from NuGet. For more information, go to [Azure.Storage.Queues](#).

4. Using the same terminal, build the .NET web application:

```
dotnet build
```

5. Close the current terminal.

Task 2: Write code to access Azure Storage

1. Open the **Program.cs** file in Visual Studio Code.
2. Delete all existing code in the **Program.cs** file.
3. Add the following **using** directives for libraries that will be referenced by the application:

```
using Azure;  
using Azure.Storage.Queues;  
using Azure.Storage.Queues.Models;  
using System;  
using System.Threading.Tasks;
```

4. Create a new **Program** class with two constant string properties named **storageConnectionString** and **messagequeue**, and then create an asynchronous **Main** entry point method:

```
public class Program  
{  
    private const string storageConnectionString = "<storage-connection-st  
    private const string queueName = "messagequeue";  
  
    public static async Task Main(string[] args)  
    {  
    }  
}
```

5. Update the **storageConnectionString** string constant by setting its value to the **Connection string** of the storage account that you recorded earlier in this lab.

Task 3: Validate Azure Storage access

1. In the **Main** method, add the following block of code to connect to the storage account and to asynchronously create the queue if it doesn't already exist:

```
QueueClient client = new QueueClient(storageConnectionString, queueName);  
await client.CreateAsync();
```

2. Still in the **Main** method, add the following block of code to render the Uniform Resource Identifier (URI) of the queue endpoint:

```
Console.WriteLine($"---Account Metadata---");  
Console.WriteLine($"Account Uri:\t{client.Uri}");
```

3. Save the **Program.cs** file.
4. Using a terminal, run the ASP.NET web application project:

```
dotnet run
```

Note: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\11\Solution\MessageProcessor** folder.

5. Observe the output from the currently running console application. The output contains metadata for the queue endpoint.
6. Close the current terminal.

Review

In this exercise, you configured your .NET project to access the Storage service and manipulate a queue made available through the service.

Exercise 3: Add messages to the queue

Task 1: Write code to access queue messages

1. In the **Main** method, perform the following actions:
 - a. Render a header by using the **Console.WriteLine** static method:

```
Console.WriteLine($"---Existing Messages---");
```

- b. Add the following block of code to create variables that will be used when retrieving queue messages:

```
int batchSize = 10;  
TimeSpan visibilityTimeout = TimeSpan.FromSeconds(2.5d);
```

- c. Add the following block of code to retrieve a batch of messages asynchronously from the queue service and iterate over the messages:

```
Response<QueueMessage[]> messages = await client.ReceiveMessagesAsync
foreach(QueueMessage message in messages?.Value)
{
}
```

- d. Within the **foreach** block, render the **MessageId** and **MessageText** properties of the **QueueMessage** instance:

```
Console.WriteLine($"[{message.MessageId}]\t{message.MessageText}");
```

2. Save the **Program.cs** file.
3. Using a terminal, build the ASP.NET web application project:

```
dotnet build
```

Note: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\11\Solution\MessageProcessor** folder.

4. Close the current terminal.

Task 2: Test message queue access

1. Using a terminal, run the ASP.NET web application project:

```
dotnet run
```

Note: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\11\Solution\MessageProcessor** folder.

2. Observe the output from the currently running console application. The output indicates that no messages are in the queue.
3. Close the current terminal.
4. Find the **asyncstor*[yourname]*** Storage account that you created earlier in this lab.
5. In the **Overview** section of the blade, select **Open in Explorer** to open the Storage account by using Storage Explorer.
6. In the **Azure Storage Explorer** application, sign in to your Azure account.
7. From the **Azure Storage Explorer** application, in the **EXPLORER** pane, find and expand the **asyncstor*[yourname]*** storage account that you created earlier in this lab.
8. Within the **asyncstor*[yourname]*** node for the Storage account that you created earlier in this lab, find and open the **messagequeue** queue.
9. Add a new message to the queue with the following properties:
 - Message text: **Hello World**

- Expires in: **12 Hours**
- Encode message body in Base 64: **No**

10. Return to the Visual Studio Code application.

11. Using a terminal, run the ASP.NET web application project:

```
dotnet run
```

Note: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\11\Solution\MessageProcessor** folder.

12. Observe the output from the currently running console application. The output includes the new message that you created.

13. Close the current terminal.

Task 3: Delete queued messages

1. In the **Visual Studio Code** window, open the **Program.cs** file.
2. Update the **foreach** loop's block of code to invoke the **DeleteMessageAsync** method of the **QueueMessage** class, passing in the **MessageId** and **PopReceipt** properties of the *message* variable:

```
foreach(QueueMessage message in messages?.Value)
{
    Console.WriteLine($"[{message.MessageId}]\t{message.MessageText}");
    await client.DeleteMessageAsync(message.MessageId, message.PopReceipt)
}
```

3. **Save** the **Program.cs** file.

4. Using a terminal, run the ASP.NET web application project:

```
dotnet run
```

Note: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\11\Solution\MessageProcessor** folder.

5. Observe the output from the currently running console application. The message that you created earlier in the lab still exists because it hasn't been deleted previously.

6. Close the current terminal.

7. In the **Azure Storage Explorer** window, within the **asyncstor*[yourname]*** node for the Storage account that you created earlier in this lab, find and open the **messagequeue** queue.

8. Observe the empty list of messages in the queue.

Note: You might need to refresh the queue.

Review

In this exercise, you read and deleted existing messages from the Storage queue by using the .NET library.

Exercise 4: Queue new messages by using .NET

Task 1: Write code to create queue messages

1. In the **Visual Studio Code** window, open the **Program.cs** file.
2. In the **Main** method, perform the following actions:
 - a. Render a header by using the **Console.WriteLine** static method:

```
Console.WriteLine($"---New Messages---");
```

- b. Add the following block of code to create a new string variable named *greeting* with a value of **Hi, Developer!**, and then invoke the **SendMessageAsync** method of the **QueueClient** class by using the *greeting* variable as a parameter:

```
string greeting = "Hi, Developer!";  
await client.SendMessageAsync(greeting);
```

- c. Add the following block of code to render the content of the message that you sent:

```
Console.WriteLine($"Sent Message:\t{greeting}");
```

3. **Save** the **Program.cs** file.
4. Using a terminal, run the ASP.NET web application project:

```
dotnet run
```

Note: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\11\Solution\MessageProcessor** folder.

5. Observe the output from the currently running console application. The content of the new message that you sent should be in the output.
6. Close the current terminal.

Task 2: View queued messages by using Storage Explorer

1. In the **Azure Storage Explorer** window, within the **asyncstor*[yourname]*** node for the Storage account that you created earlier in this lab, find and open the **messagequeue** queue.
2. Observe the single new message in the list of messages in the queue.

Note: You might need to refresh the queue.

Review

In this exercise, you created new messages in the queue by using the .NET library for Storage queues.

Exercise 5: Clean up your subscription

Task 1: Open Azure Cloud Shell and list resource groups

1. In the portal, select the **Cloud Shell** icon to open a new shell instance.
2. If Cloud Shell isn't already configured, configure the shell for Bash by using the default settings.

Task 2: Delete a resource group

1. Enter the following command, and then select Enter to delete the **AsyncProcessor** resource group:

```
az group delete --name AsyncProcessor --no-wait --yes
```

2. Close the Cloud Shell pane in the portal.

Task 3: Close the active application

1. Close the currently running Microsoft Edge application.
2. Close the currently running Visual Studio Code application.
3. Close the currently running Azure Storage Explorer application.

Review

In this exercise, you cleaned up your subscription by removing the resource group that was used in this lab.

Congratulations!

You have successfully completed this exercise. Click **End** to advance to the next exercise.