✓ 100 XP

Exercise - Retrieve a message from the queue

10 minutes

Sandbox activated! Time remaining: **22 min**

You have used 3 of 10 sandboxes for today. More sandboxes will be available tomorrow.

Now we want to complete the application by writing code to read the next message in the queue, process it, and delete it from the queue.

We're going to place this code into the same application and execute it when you don't pass any parameters, however in our news service scenario, we would really place this code into our middle-tier servers to process the stories.

Dequeue a message

Let's add a new method that retrieves the next message from the queue.

1. Open the `Program.cs` source file in your editor.
2. Create a static method in the `Program` class named `ReceiveArticleAsync` that takes no parameters and returns a `Task<string>`. We'll use this method to pull a news article from the queue and return it.
   - Go ahead and add the `async` keyword to the method since we'll be using some asynchronous `Task`-based methods.

   C# Copy

   ```
   static async Task<string> ReceiveArticleAsync()
   {
   }
   ```

3. All of the setup code to get a `CloudQueue` will be identical to what we did in the last exercise. Code duplication is a bad habit, even in samples so go ahead and, refactor the

code that obtains the `CloudQueue` to a new method named `GetQueue` and change the `SendArticleAsync` to use your new method.

- Make sure to leave the code that *creates* the queue in the `SendArticleAsync` method; remember only the **publisher** should create the queue.

C# Copy

```csharp
const string ConnectionString = ...;
// ...

static CloudQueue GetQueue()
{
    CloudStorageAccount storageAccount =
CloudStorageAccount.Parse(ConnectionString);

    CloudQueueClient queueClient = storageAccount.CreateCloudQueueClient();
    return queueClient.GetQueueReference("newsqueue");
}
```

4. In your `ReceiveArticleAsync` method, call the new `GetQueue` method to retrieve your queue reference and assign it to a variable.

5. Next, call the `ExistsAsync` method on the `CloudQueue` object; this will return whether the queue has been created. If we attempt to retrieve a message from a non-existent queue, the API will throw an exception.

   - This method is asynchronous so use `await` to get the return value.
   - You should already have the `async` keyword on the `ReceiveArticleAsync` method, but if not add it now.

6. Add an `if` block that uses the return value from `ExistsAsync`. We'll add our code to read a value from the queue into the block. Add a final return string to the method that indicates no value was read. Your method should be looking something like this:

C# Copy

```csharp
static async Task<string> ReceiveArticleAsync()
{
    CloudQueue queue = GetQueue();
    bool exists = await queue.ExistsAsync();
    if (exists)
    {
    }

    return "<queue empty or not created>";
}
```

1. Call `GetMessageAsync` on the `CloudQueue` object to get the first `CloudQueueMessage` from the queue. The return value will be `null` if the queue is empty.

2. If it's non-null, use the `AsString` property on the `CloudQueueMessage` object to get the contents of the message.

3. Call `DeleteMessageAsync` on the `CloudQueue` object to delete the message from the queue.

The final method implementation should resemble:

C# Copy

```csharp
static async Task<string> ReceiveArticleAsync()
{
    CloudQueue queue = GetQueue();
    bool exists = await queue.ExistsAsync();
    if (exists)
    {
        CloudQueueMessage retrievedArticle = await queue.GetMessageAsync();
        if (retrievedArticle != null)
        {
            string newsMessage = retrievedArticle.AsString;
            await queue.DeleteMessageAsync(retrievedArticle);
            return newsMessage;
        }
    }

    return "<queue empty or not created>";
}
```

Call the ReceiveArticleAsync method

Finally, let's add support to invoke our new method. We'll do this when we don't pass any parameters into the program.

1. Locate the `Main` method and specifically the `if` block you added earlier to look for parameters.

2. Add an `else` condition and call the `ReceiveArticleAsync` method.

3. Since it's asynchronous, use the `await` keyword to retrieve the result and print it to the console window. If you didn't convert your app to C# 7.1, you can get the value using the `Result` property from the returning task.

Your code should look something like:

C# Copy

```csharp
if (args.Length > 0)
{
    // ...
}
else
{
    string value = await ReceiveArticleAsync();
    Console.WriteLine($"Received {value}");
}
```

Execute the application

The code is now complete. It can now send and retrieve messages.

⚠ **Warning**

Make sure you have saved all the files in the online editor before you build and run the program.

To test it, use `dotnet run` and pass parameters to send messages, and leave off parameters to read a single message.

If you want to test when a queue doesn't exist, you can delete the queue (and all the data) with the Azure CLI. Make sure to replace the `<connection-string>` parameter (or set the environment variable).

Azure CLI Copy

```
az storage queue delete --name newsqueue --connection-string <connection-string>
```

The next time you add a message, the queue should be re-created.

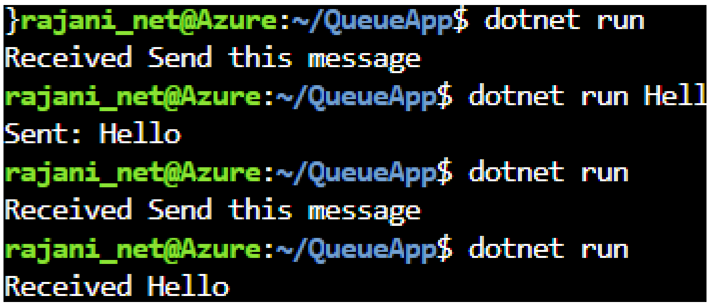
ⓘ **Note**

The delete operation actually occurs asynchronously. If it has not completed you may get an exception when you attempt to re-create the queue.

Next unit: Summary

Continue >

Azure Cloud Shell



```
}rajani_net@Azure:~/QueueApp$ dotnet run
Received Send this message
rajani_net@Azure:~/QueueApp$ dotnet run Hell
Sent: Hello
rajani_net@Azure:~/QueueApp$ dotnet run
Received Send this message
rajani_net@Azure:~/QueueApp$ dotnet run
Received Hello
```