< Previous Unit 7 of 9 ∨ Next >

✓ 100 XP ▶

Exercise - Add a message to the queue

10 minutes

Sandbox activated! Time remaining: **57 min**

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

Now that all the requirements are in place, you can write code that creates a new storage queue and adds a message. We would typically place this code in our front-end apps that generate the data.

Add the client library for Azure Storage

Let's start by adding the **Azure Storage Client Library for .NET** to our app. It can be installed with NuGet (a .NET package manager).

1. Install the `WindowsAzure.Storage` package to the project with the `dotnet add package` command. You can do this in the terminal window *below* the Cloud Shell, or if you are working on your local computer, in a terminal/console window in the same folder as the project.

Azure CLI Copy

```
dotnet add package WindowsAzure.Storage
```

Add a method to send a news alert

Next, let's create a new method to send a news story into a queue.

1. Open the `Program.cs` file in your code editor.

2. At the top of the file, add the following namespaces. We'll be using types from both of these to connect to Azure Storage and then to work with queues.

   - `System.Threading.Tasks`

- `Microsoft.WindowsAzure.Storage`
- `Microsoft.WindowsAzure.Storage.Queue`

3. Create a static method in the `Program` class named `SendArticleAsync` that takes a `string` and returns a `Task`. We'll use this method to send a news article in to our service. Name the input parameter `newsMessage` as shown below.

C# Copy

```
...
using System.Threading.Tasks;
using Microsoft.WindowsAzure.Storage;
using Microsoft.WindowsAzure.Storage.Queue;

class Program
{
    ...

    static async Task SendArticleAsync(string newsMessage)
    {
    }
}
```

4. In your new method, use the static `CloudStorageAccount.Parse` method to parse your connection string (recall we placed it into a constant string). This method returns a `CloudStorageAccount` object that needs to be stored in a variable.

5. Call the `CreateCloudQueueClient()` method on the storage account object to get a client object and store that in a variable.

6. Next, call `GetQueueReference` method on the client object and pass the string "newsqueue" for the queue name. This returns a `CloudQueue` object that we can use to work with the queue. It's OK if the queue does not exist yet.

7. Call `CreateIfNotExistsAsync()` on the `CloudQueue` object to ensure the queue is ready for use. This will create the queue if necessary.

    - Since this is an asynchronous method, use the C# `await` keyword to ensure we work properly with it. That also means you need to decorate the *method* with the `async` keyword.
    - `CreateIfNotExistsAsync` returns a `bool` value that will be `true` if the queue was created and `false` if it already exists. Output a message to the console if we created the queue.
    - Here's an example if you need some help.

C# Copy

```
static async Task SendArticleAsync(string newsMessage)
{
    // Not Shown here - code from prior steps
    ...
    bool createdQueue = await queue.CreateIfNotExistsAsync();
    if (createdQueue)
    {
        Console.WriteLine("The queue of news articles was created.");
    }
}
```

8. Create an instance of a `CloudQueueMessage`.

   - It takes a `string` parameter, pass in the method parameter (`newsMessage`). This will be the *body* of the message. There is also a static method named that can create a binary message payload.

9. Call `AddMessageAsync` on the `CloudQueue` object to add the message to the queue. This is also an asynchronous method and you will need to use the `await` keyword to ensure we properly interact with it.

10. Save the file and build it by typing `dotnet build` into the command window. Fix any errors, you can use the following code to check your work.

C# Copy

```
static async Task SendArticleAsync(string newsMessage)
{
    CloudStorageAccount storageAccount =
CloudStorageAccount.Parse(ConnectionString);

    CloudQueueClient queueClient = storageAccount.CreateCloudQueueClient();

    CloudQueue queue = queueClient.GetQueueReference("newsqueue");
    bool createdQueue = await queue.CreateIfNotExistsAsync();
    if (createdQueue)
    {
        Console.WriteLine("The queue of news articles was created.");
    }

    CloudQueueMessage articleMessage = new CloudQueueMessage(newsMessage);
    await queue.AddMessageAsync(articleMessage);
}
```

Add code to send a message

Let's modify the `Main` method to pass any parameters received into our new method so we can test our new send method.

1. Locate the `Main` method.

2. Check the passed `args` parameter to see if any data was passed to the command line. If so, use `String.Join` to create a single string from all the words using a space as the separator.

3. Pass that to the new `SendArticleAsync` method.

4. Once it returns, use `Console.WriteLine` to output the message we sent.

5. Save the file and build the program (`dotnet build`), you can use the code below to check your work.

> **Note**
>
> Since our method is technically asynchronous, we will want to use the `await` keyword, however that feature isn't available on your `Main` method unless you are using C# 7.1 or later. Just call `Wait()` on the method to actually block waiting for the method to return, we'll fix that in a minute.

C#

Copy

```csharp
static void Main(string[] args)
{
    if (args.Length > 0)
    {
        string value = String.Join(" ", args);
        SendArticleAsync(value).Wait();
        Console.WriteLine($"Sent: {value}");
    }
}
```

Execute the application

The application can now send messages. To test it, you can run it from the command line with the `dotnet run` command. Any additional strings are passed as parameters to the application.

> ⚠ **Warning**

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

As an example, you can type:

Azure CLI

Copy

```
dotnet run Send this message
```

This should add the string `"Send this message"` into the queue.

Check your results

You can check queues in the Azure portal using the **Storage Explorer**, if you open that product it will let you explore the data in each storage account you own.

Alternatively, you can use the Azure CLI or PowerShell. Try this command in the shell, replacing the `<connection-string>` value with your specific connection string:

Azure CLI

Copy

```
az storage message peek --queue-name newsqueue --connection-string <connection-
string>
```

This should dump the information for your message, which will look something like this:

JSON

Copy

```
[
  {
    "content": "U2VuZCB0aGlzIG1lc3NhZ2U=",
    "dequeueCount": 0,
    "expirationTime": "2018-09-05T02:43:40+00:00",
    "id": "b47cbe9f-a246-4a81-86ae-fa02ea8d98bc",
    "insertionTime": "2018-09-24T02:43:40+00:00",
    "popReceipt": null,
    "timeNextVisible": null
  }
]
```

There are several other commands available that you can try with the tools - check out both `az storage queue --help` and `az storage message --help` to explore them.

Tip

Put your connection string value into an environment variable named `AZURE_STORAGE_CONNECTION_STRING` to save yourself from having to type the `--connection-string` parameter every time!

Optional - Use the async versions of the methods

We used `Wait()` on the send method above but that's not an efficient use of our computing resources. Instead, we want to use the C# `async` and `await` methods. However, we will need to be using *at least* C# 7.1 to be able to apply these keywords to our **Main** method.

Switch to C# 7.1

C#'s `async` and `await` keywords were not valid keywords in **Main** methods until C# 7.1. We can easily switch to that compiler through a flag in the **.csproj** file.

1. Open the **QueueApp.csproj** file in the editor.

2. Add `<LangVersion>7.1</LangVersion>` into the first `PropertyGroup` in the build file. It should look like the following when you are finished.

   XML Copy

   ```xml
   <Project Sdk="Microsoft.NET.Sdk">

     <PropertyGroup>
       <OutputType>Exe</OutputType>
       <TargetFramework>netcoreapp2.2</TargetFramework>
       <LangVersion>7.1</LangVersion>
     </PropertyGroup>
   ...
   ```

Apply the async keyword

Next, apply the `async` keyword to the **Main** method. We will have to do three things.

1. Add the `async` keyword onto the **Main** method signature.
2. Change the return type from `void` to `Task`.
3. Remove the call to `Wait()` on the call to `SendArticleAsync` and replace it with `await`.

Try running the app again - it should still work exactly as before, but now the code is able to release the thread back to the .NET threadpool while we wait for the message to be queued.

Now that we have sent a message, the last step is to add support to *receive* the message.

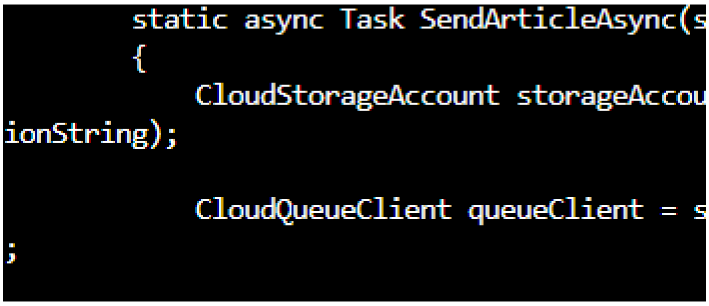
Next unit: Exercise - Retrieve a message from the queue

Continue >

English (United States) Theme

Previous Version Docs Blog Contribute Privacy & Cookies Terms of Use Trademarks

Azure Cloud Shell



```
        static async Task SendArticleAsync(s
        {
            CloudStorageAccount storageAccou
ionString);

            CloudQueueClient queueClient = s
;
```