100 XP

Write code that uses Service Bus queues

5 minutes

Distributed applications use queues, such as Service Bus queues, as temporary storage locations for messages that are awaiting delivery to a destination component. To send and receive messages through a queue, you must write code in the source and destination components.

Consider the Contoso Slices application. The customer places the order through a website or mobile app. Because websites and mobile apps run on customer devices, there is really no limit to how many orders could come in at once. By having the mobile app and website deposit the orders in a queue, we can allow the back-end component (a web app) to process orders from that queue at its own pace.

The Contoso Slices application actually has several steps to handle a new order. All the steps are dependent on first authorizing payment, so we decide to use a queue. Our receiving component's first job will be processing the payment.

In the mobile app and website, Contoso needs to write code that adds a message to the queue. In the back-end web app, they'll write code that picks up messages from the queue.

Here, you will learn how to write that code.

The Microsoft.Azure.ServiceBus NuGet package

To make it easy to write code that sends and receives messages through Service Bus, Microsoft provides a library of .NET classes, which you can use in any .NET Framework language to interact with a Service Bus queue, topic, or relay. You can include this library in your application by adding the **Microsoft.Azure.ServiceBus** NuGet package.

The most important class in this library for queues is the `QueueClient` class. You must start by instantiating this class both in sending and receiving components.

Connection strings and keys

Source components and destination components both need two pieces of information to connect to a queue in a Service Bus namespace:

- The location of the Service Bus namespace, also known as an **endpoint**. The location is specified as a fully qualified domain name within the **servicebus.windows.net** domain. For example: **pizzaService.servicebus.windows.net**.
- An access key. Service Bus restricts access to queues, topics, and relays by requiring an access key.

Both of these pieces of information are provided to the `QueueClient` object in the form of a connection string. You can obtain the correct connection string for your namespace from the Azure portal.

Calling methods asynchronously

The queue in Azure may be located thousands of miles away from sending and receiving components. Even if it is physically close, slow connections and bandwidth contention may cause delays when a component calls a method on the queue. For this reason, the Service Bus client library makes `async` methods available for interacting with the queues. We'll use these methods to avoid blocking a thread while waiting for calls to complete.

When sending a message to a queue, for example, use the `QueueClient.SendAsync()` method with the `await` keyword.

Write code that sends to queues

In any sending or receiving component, you should add the following `using` statements to any code file that calls a Service Bus queue:

C# Copy

```csharp
using System.Threading;
using System.Threading.Tasks;
using Microsoft.Azure.ServiceBus;
```

Next, create a new `QueueClient` object and pass it the connection string and the name of the queue:

C# Copy

```csharp
queueClient = new QueueClient(TextAppConnectionString, "PrivateMessageQueue");
```

You can send a message to the queue by calling the `QueueClient.SendAsync()` method and passing the message in the form of a UTF-8 encoded string:

C# Copy

```
string message = "Sure would like a large pepperoni!";
var encodedMessage = new Message(Encoding.UTF8.GetBytes(message));
await queueClient.SendAsync(encodedMessage);
```

Receive messages from the queue

To receive messages, you must first register a message handler - this is the method in your code that will be invoked when a message is available on the queue.

C# Copy

```
queueClient.RegisterMessageHandler(MessageHandler, messageHandlerOptions);
```

Do your processing work. Then, within the message handler, call the `QueueClient.CompleteAsync()` method to remove the message from the queue:

C# Copy

```
await queueClient.CompleteAsync(message.SystemProperties.LockToken);
```

Next unit: Exercise - Write code that uses Service Bus queues

Continue >