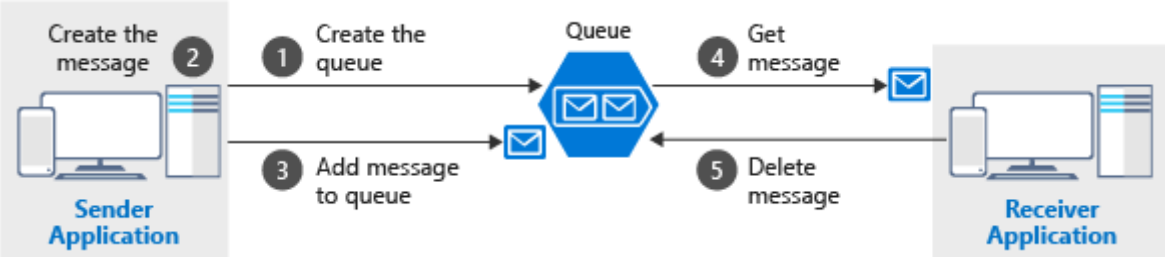
100 XP

Programmatically access a queue

5 minutes

Queues hold messages - packets of data whose shape is known to the sender application and receiver application. The sender creates the queue and adds a message. The receiver retrieves a message, processes it, and then deletes the message from the queue. The following illustration shows a typical flow of this process.



Notice that `get` and `delete` are separate operations. This arrangement handles potential failures in the receiver and implements a concept called *at-least-once delivery*. After the receiver gets a message, that message remains in the queue but is invisible for 30 seconds. If the receiver crashes or experiences a power failure during processing, then it will never delete the message from the queue. After 30 seconds, the message will reappear in the queue and another instance of the receiver can process it to completion.

The Azure Storage Client Library for .NET

The **Azure Storage Client Library for .NET** provides types to represent each of the objects you need to interact with:

- `CloudStorageAccount` represents your Azure storage account.
- `CloudQueueClient` represents Azure Queue storage.
- `CloudQueue` represents one of your queue instances.
- `CloudQueueMessage` represents a message.

You will use these classes to get programmatic access to your queue. The library has both synchronous and asynchronous methods; you should prefer to use the asynchronous versions to avoid blocking the client app.

> **Note**

The Azure Storage Client Library for .NET is available in the **WindowsAzure.Storage** NuGet package. You can install it through an IDE, Azure CLI, or through PowerShell `Install-Package WindowsAzure.Storage`.

How to connect to a queue

To connect to a queue, you first create a `CloudStorageAccount` with your connection string. The resulting object can then create a `CloudQueueClient`, which in turn can open a `CloudQueue` instance. The basic code flow is shown below.

C# Copy

```
CloudStorageAccount account = CloudStorageAccount.Parse(connectionString);

CloudQueueClient client = account.CreateCloudQueueClient();

CloudQueue queue = client.GetQueueReference("myqueue");
```

Creating a `CloudQueue` doesn't necessarily mean the *actual* storage queue exists. However, you can use this object to create, delete, and check for an existing queue. As mentioned above, all methods support both synchronous and asynchronous versions, but we will only be using the `Task`-based asynchronous versions.

How to create a queue

You will use a common pattern for queue creation: the sender application should always be responsible for creating the queue. This keeps your application more self-contained and less dependent on administrative set-up.

To make the creation simple, the client library exposes a `CreateIfNotExistsAsync` method that will create the queue if necessary, or return `false` if the queue already exists.

Typical code is shown below.

C# Copy

```
CloudQueue queue;
//...

await queue.CreateIfNotExistsAsync();
```

Note

You must have `Write` or `Create` permissions for the storage account to use this API. This is always true if you use the **Access Key** security model, but you can lock down permissions to the account with other approaches that will only allow read operations against the queue.

How to send a message

To send a message, you instantiate a `CloudQueueMessage` object. The class has a few overloaded constructors that load your data into the message. We will use the constructor that takes a `string`. After creating the message, you use a `CloudQueue` object to send it.

Here's a typical example:

C# Copy

```
var message = new CloudQueueMessage("your message here");

CloudQueue queue;
//...

await queue.AddMessageAsync(message);
```

Note

While the total queue size can be up to 500 TB, the individual messages in it can only be up to 64 KB in size (48 KB when using Base64 encoding). If you need a larger payload you can combine queues and blobs – passing the URL to the actual data (stored as a Blob) in the message. This approach would allow you to enqueue up to 200 GB for a single item.

How to receive and delete a message

In the receiver, you get the next message, process it, and then delete it after processing succeeds. Here's a simple example:

C# Copy

```
CloudQueue queue;
//...

CloudQueueMessage message = await queue.GetMessageAsync();

if (message != null)
{
```

```
    // Process the message
    //...

    await queue.DeleteMessageAsync(message);
}
```

Let's now apply this new knowledge to our application!

Next unit: Exercise - Add a message to the queue

Continue >