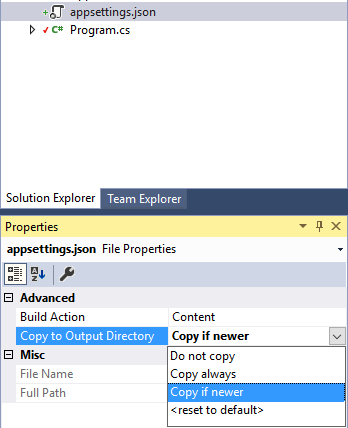
**Reading appsettings.json file in .NET Core Console Application.**

All that’s required is to add the following NuGet packages and an appsettings.json file.

* Microsoft.Extensions.Configuration
* Microsoft.Extensions.Configuration.FileExtensions
* Microsoft.Extensions.Configuration.Json

The **appsettings.json** files “**Copy to Output Directory**” property should also be set to “**Copy if newer**” so that the application is able to access it when published.

[](https://blog.bitscry.com/wp-content/uploads/2017/05/appsettings-copy.png)

appsettings.json

{

"StorageConnectionString": "DefaultEndpointsProtocol=https;AccountName=dssdemo1;AccountKey=pLp+3F+cNsno0FFGFKEZRxSZoPpUmvbB8ZbZbXHHUvCYvqBESP4U2nkQbB/o8gTZ5B/iyali1Au9LQdnQJm/YQ==;EndpointSuffix=core.windows.net"

}

class Program

{

static void Main()

{

var builder = new ConfigurationBuilder()

.SetBasePath(Directory.GetCurrentDirectory())

.AddJsonFile("appsettings.json");

IConfiguration config = builder.Build();

Console.WriteLine($" Hello { config["StorageConnectionString"] } ");

}

}

**Programming BLOB in C#**

1. Create a Console Application.
2. Add reference to System.Configurtion
3. Tools 🡪 NuGet Package Manager 🡪 Manage NuGet Packages for Solution 🡪 Search "**Azure Storage** " 🡪 Install.

Note: This add reference to **Azure.Storage.Blob**.

using System;

using Azure.Storage.Blobs;

using Azure.Storage.Blobs.Models;

using System.IO;

using Azure.Storage.Sas;

using Azure.Storage.Blobs.Specialized;

using System.Collections.Generic;

using Azure;

class Program

{

static BlobServiceClient blobServiceClient;

static void Main(string[] args)

{

string connectionString = "DefaultEndpointsProtocol=https;AccountName=mydssdemostorage;AccountKey=Sb2JyXWwIbAA/UxWxXVruJU53LwzGGxdl2EPhVs6zq59ZcH1fW+FRp5yjCMtCwweFSGSMbGwfHPLvsX623upRA==;EndpointSuffix=core.windows.net";

blobServiceClient = new BlobServiceClient(connectionString);

string file1 = "d:\\images\\shoes-1.png";

string file2 = "d:\\images\\shoes-2.png";

string file3 = "d:\\images\\shoes-3.png";

BlobContainerClient con1 = CreateContainer("con1", false);

BlobContainerClient con2 = CreateContainer("con2", true);

UploadBlob(con1, file1);

UploadBlob(con1, file2);

UploadBlob(con2, file1);

UploadBlob(con2, file3);

DownloadBlob(con2, file3);

ListBlobs(con1);

ListBlobs(con2);

ListBlobsAsAnonymousUser("con1");

ListBlobsAsAnonymousUser("con2");

BlobClient blobClient = con1.GetBlobClient("shoes-2.png");

blobClient.Delete();

LeaseDemo(con1);

}

static BlobContainerClient CreateContainer(string containerName, bool isPublic)

{

BlobContainerClient container = blobServiceClient.GetBlobContainerClient(containerName);

if (!container.Exists())

{

container.CreateIfNotExists();

Console.WriteLine($"{containerName} is Created\n");

if (isPublic)

{

container.SetAccessPolicy(PublicAccessType.Blob);

}

}

return container;

}

static BlobClient UploadBlob(BlobContainerClient container, string path)

{

FileInfo fi = new FileInfo(path);

BlobClient blobClient = container.GetBlobClient(fi.Name);

blobClient.Upload(path);

Console.WriteLine($"Access blob here - {blobClient.Uri.AbsoluteUri}");

return blobClient;

}

static BlobClient DownloadBlob(BlobContainerClient container, string path)

{

FileInfo fi = new FileInfo(path);

BlobClient blobClient = container.GetBlobClient(fi.Name);

blobClient.DownloadTo("d:\\demo.png");

return blobClient;

}

static void ListBlobs(BlobContainerClient container)

{

Console.WriteLine($"List of Blobs in {container.Name} and {container.GetAccessPolicy().Value}");

foreach (var blob in container.GetBlobs())

{

BlobClient blobClient = container.GetBlobClient(blob.Name);

Console.WriteLine($"{blob.Name} - {blobClient.Uri}");

}

Console.WriteLine("");

}

static void DeleteBlobs(BlobContainerClient container)

{

foreach (var blob in container.GetBlobs())

{

container.DeleteBlob(blob.Name);

}

}

static void DeleteContainer(BlobContainerClient container)

{

container.Delete();

}

static string CreateServiceSASforBlob(BlobClient blobClient, string storedPolicyName = null)

{

if (blobClient.CanGenerateSasUri)

{

// Create a SAS token that's valid for one hour.

BlobSasBuilder sasBuilder = new BlobSasBuilder()

{

BlobContainerName = blobClient.GetParentBlobContainerClient().Name,

BlobName = blobClient.Name,

Resource = "b"

};

if (storedPolicyName == null)

{

sasBuilder.ExpiresOn = DateTimeOffset.UtcNow.AddHours(1);

sasBuilder.SetPermissions(BlobSasPermissions.Read | BlobSasPermissions.Write);

}

else

{

sasBuilder.Identifier = storedPolicyName;

}

Uri sasUri = blobClient.GenerateSasUri(sasBuilder);

Console.WriteLine("SAS URI for blob is: {0}", sasUri);

Console.WriteLine();

return sasUri.ToString();

}

else

{

Console.WriteLine(@"BlobClient must be authorized with Shared Key

credentials to create a service SAS.");

return null;

}

}

static string CreateServiceSASforBlobContainer(BlobContainerClient container, string policyName = null)

{

// Check whether this BlobContainerClient object has been authorized with Shared Key.

if (container.CanGenerateSasUri)

{

// Create a SAS token that's valid for one hour.

BlobSasBuilder sasBuilder = new BlobSasBuilder()

{

BlobContainerName = container.Name,

Resource = "c"

};

if (policyName == null)

{

sasBuilder.ExpiresOn = DateTimeOffset.UtcNow.AddHours(1);

sasBuilder.SetPermissions(BlobContainerSasPermissions.Read);

}

else

{

sasBuilder.Identifier = policyName;

}

Uri sasUri = container.GenerateSasUri(sasBuilder);

Console.WriteLine("SAS URI for blob container is: {0}", sasUri);

return sasUri.ToString();

}

else

{

Console.WriteLine(@"BlobContainerClient must be authorized with Shared Key credentials to create a service SAS.");

return null;

}

}

static void CreateSharedAccessPolicy(BlobContainerClient container, string policyName)

{

try

{

List<BlobSignedIdentifier> signedIdentifiers = new List<BlobSignedIdentifier>();

signedIdentifiers.Add(

new BlobSignedIdentifier

{

Id = policyName,

AccessPolicy = new BlobAccessPolicy

{

StartsOn = DateTimeOffset.UtcNow.AddHours(-1),

ExpiresOn = DateTimeOffset.UtcNow.AddDays(1),

Permissions = "rw"

}

}

);

// Set the container's access policy.

container.SetAccessPolicy(permissions: signedIdentifiers);

}

catch (RequestFailedException e)

{

Console.WriteLine(e.ErrorCode);

Console.WriteLine(e.Message);

}

}

}

**Programming Storage Tables in C#**

1. Create a Console Application.
2. Tools 🡪 NuGet Package Manager 🡪 Manage NuGet Packages for Solution 🡪 Search **"Microsoft.Azure.Cosmos.Table"** 🡪 Install.
3. Open Configuration file 🡪Add the following.
4. Open Program.cs and add the following code.

using Microsoft.Azure.Storage;

using Microsoft.Azure.Cosmos.Table;

using Microsoft.Extensions.Configuration;

using System;

class Employee : TableEntity

{

public string Email { get; set; }

public int PhoneNumber { get; set; }

public Employee() { }

public Employee(string deptName, string empName, string email, string phoneNumber)

{

this.PartitionKey = deptName;

this.RowKey = empName;

this.Email = email;

this.PhoneNumber = phoneNumber;

}

}

class Program

{

static void Main(string[] args)

{

//string cs = System.Configuration.ConfigurationManager.AppSettings["StorageConnectionString"];

var builder = new ConfigurationBuilder().SetBasePath(Directory.GetCurrentDirectory()).AddJsonFile("appsettings.json");

IConfiguration config = builder.Build();

string cs = config["StorageConnectionString"];

CloudStorageAccount storageAccount = CloudStorageAccount.Parse(cs);

CloudTableClient tableClient = storageAccount.CreateCloud**Table**Client();

CloudTable tableEmployee = tableClient.**GetTableReference**("employee");

tableEmployee.CreateIfNotExists();

Employee emp1 = new Employee("Training", "Emp1", "emp1@test.com", 11111 );

Employee emp2 = new Employee("Training", "Emp2", "emp2@test.com", 22222 );

Employee emp3 = new Employee("Development", "Emp3", "[emp3@test.com](mailto:emp3@test.com)" , 33333);

TableBatchOperation batchOperation = new TableBatchOperation();

batchOperation.InsertOrReplace(emp1);

batchOperation.InsertOrReplace(emp2);

tableEmployee.ExecuteBatch(batchOperation);

batchOperation = new TableBatchOperation();

batchOperation.InsertOrReplace(emp3);

tableEmployee.ExecuteBatch(batchOperation);

TableOperation operation = TableOperation.**Retrieve**<Employee>("Development", "Emp3");

TableResult result = tableEmployee.Execute(operation);

emp3 = result.Result as Employee;

//emp3.ETag = "\*";

batchOperation = new TableBatchOperation();

batchOperation.Delete(emp3);

tableEmployee.ExecuteBatch(batchOperation);

TableQuery<Employee> query = new TableQuery<Employee>();

string filter = TableQuery.GenerateFilterCondition("Email", QueryComparisons.Equal, "emp1@test.com");

query = query.Where(filter);

var emps = tableEmployee.ExecuteQuery(query);

foreach (Employee emp in emps)

{

Console.WriteLine(emp.PartitionKey + " " + emp.RowKey + " " + emp.Email);

}

//tableEmployee.DeleteIfExists();

}

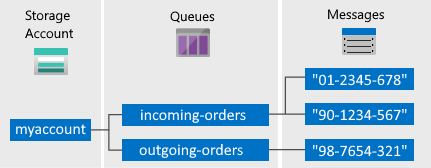
}

1. Build and Run the application.

**Programming Storage Queue in C#**

## Queue service concepts

The Azure Queue service contains the following components:



* **Storage Account:** All access to Azure Storage is done through a storage account. For more information about storage accounts, see [Storage account overview](https://docs.microsoft.com/en-us/azure/storage/common/storage-account-overview).
* **Queue:** A queue contains a set of messages. All messages must be in a queue. Note that the queue name must be all lowercase. For information on naming queues, see [Naming Queues and Metadata](https://docs.microsoft.com/en-us/rest/api/storageservices/Naming-Queues-and-Metadata).
* **Message:** A message, in any format, of up to 64 KB. The maximum time that a message can remain in the queue is 7 days. For version 2017-07-29 or later, the maximum time-to-live can be any positive number, or -1 indicating that the message doesn't expire. If this parameter is omitted, the default time-to-live is seven days.
* **URL format:** Queues are addressable using the following URL format: http://<storage account>.queue.core.windows.net/<queue>

The following URL addresses a queue:

<http://myaccount.queue.core.windows.net/incoming-orders>

**Create the Sender Application to Post messages in to a Queue**

**Add the NuGet Package** Azure.Storage.Queues

using System;

using System.IO;

using Azure.Storage.Queues;

using Azure.Storage.Queues.Models;

using Microsoft.Extensions.Configuration;

class Program

{

static QueueClient queueClient;

static void Main(string[] args)

{

var builder = new ConfigurationBuilder()

.SetBasePath(Directory.GetCurrentDirectory())

.AddJsonFile("appsettings.json");

IConfiguration config = builder.Build();

string connectionString = config["StorageConnectionString"];

string queueName = "demoqueue";

QueueClient queueClient = new QueueClient(connectionString, queueName);

queueClient.CreateIfNotExists();

while (true)

{

Console.Write("Enter a message to be sent to myqueue:");

var msg = Console.ReadLine();

if (msg == "exit")

break;

**queueClient.SendMessage(msg);**

}

//If you don't pass a value for the maxMessages parameter, the default is to peek at one message.

PeekedMessage[] peekedMessages = queueClient.PeekMessages(maxMessages: 1);

if (peekedMessages.Length != 0)

Console.WriteLine($"Peeked message: '{peekedMessages[0].Body}'");

// The following code updates the queue message with new contents, and sets the visibility timeout to extend another 60 seconds.

QueueMessage[] messages = queueClient.ReceiveMessages(); //Fetches only one message from queue (visibilitytimeout=30)

queueClient.**UpdateMessage**(messages[0].MessageId,

message[0].PopReceipt,

"Updated contents",

TimeSpan.FromSeconds(60) // Make it invisible for another 60 seconds; Default

);

**QueueProperties** properties = queueClient.GetProperties();

int cachedMessagesCount = properties.**ApproximateMessagesCount**;

//This number is not lower than the actual number of messages in the queue, but could be higher.

}

}

Build and Run the application

**Create the Consumer Application to read messages from the queue**

1. Start a New Instance of Visual Studio and Create a Console Based Application

using Azure.Storage.Queues;

using Azure.Storage.Queues.Models;

using Microsoft.Extensions.Configuration;

using System;

using System.IO;

class Program2

{

static QueueClient queueClient;

static void Main(string[] args)

{

var builder = new ConfigurationBuilder()

.SetBasePath(Directory.GetCurrentDirectory())

.AddJsonFile("appsettings.json");

IConfiguration config = builder.Build();

string connectionString = config["StorageConnectionString"];

string queueName = "demoqueue";

QueueClient queueClient = new QueueClient(connectionString, queueName);

queueClient.CreateIfNotExists();

while (true)

{

QueueMessage[] retrievedMessages = queueClient.ReceiveMessages(1); //Fetches only one message from queue (visibilitytimeout=30)

if (retrievedMessages.Length ==0)

Console.WriteLine("No Messages in last 5 secs...");

foreach (var msg in retrievedMessages)

{

Console.WriteLine($"Dequeued message: '{msg.Body}'");

queueClient.DeleteMessage(msg.MessageId, msg.PopReceipt);

}

System.Threading.Thread.Sleep(5000);

}

}

}

1. Right click on the Sender project 🡪Debug🡪Start new instance.
2. Right click on the Received project 🡪Debug🡪Start new instance.
3. Send messages from the Producer and see them received by the Consumer.