Microsoft Azure

Azure / Storage

.NET

Get started with Azure Blob storage using .NET

12/8/2016 • 20 min to read • Contributors all

In this article

.NET

Tip

Manage Azure Blob Storage resources with Microsoft Azure Storage Explorer

Microsoft Azure Storage Explorer is a free, standalone app from Microsoft that enables you to manage Azure Blob Storage resources. Using Microsoft Azure Storage Explorer, you can visually create, read, update, and delete blob containers and blobs, as well as manage access to your blobs containers and blobs.

Overview

Azure Blob storage is a service that stores unstructured data in the cloud as objects/blobs. Blob storage can store any type of text or binary data, such as a document, media file, or application installer. Blob storage is also referred to as object storage.

About this tutorial

This tutorial shows how to write .NET code for some common scenarios using Azure Blob storage. Scenarios covered include uploading, listing, downloading, and deleting blobs.

Prerequisites:

- Microsoft Visual Studio
- Azure Storage Client Library for .NET
- Azure Configuration Manager for .NET
- An Azure storage account

> **Note**
>
> We recommend that you use the latest version of the Azure Storage Client Library for .NET to complete this tutorial. The latest version of the library is 7.x, available for download on Nuget. The source for the client library is available on GitHub.
>
> If you are using the storage emulator, note that version 7.x of the client library requires at least version 4.3 of the storage emulator

More samples

For additional examples using Blob storage, see Getting Started with Azure Blob Storage in .NET. You can download the sample application and run it, or browse the code on GitHub.

What is Blob Storage?

Azure Blob storage is a service for storing large amounts of unstructured object data, such as text or binary data, that can be accessed from anywhere in the world via HTTP or HTTPS. You can use Blob storage to expose data publicly to the world, or to store application data privately.

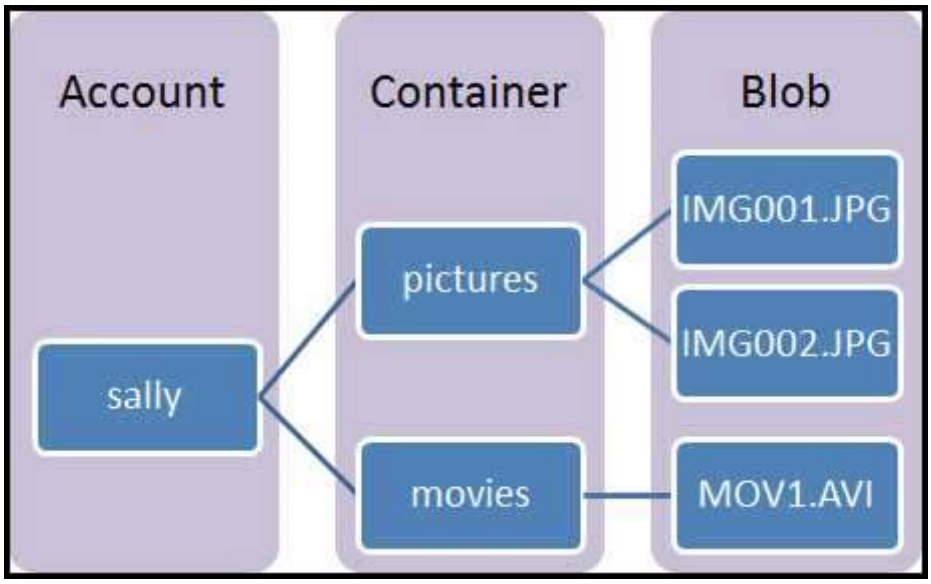
Common uses of Blob storage include:

- Serving images or documents directly to a browser
- Storing files for distributed access

- Streaming video and audio
- Storing data for backup and restore, disaster recovery, and archiving
- Storing data for analysis by an on-premises or Azure-hosted service

Blob service concepts

The Blob service contains the following components:



- **Storage Account:** All access to Azure Storage is done through a storage account. This storage account can be a **General-purpose storage account** or a **Blob storage account** which is specialized for storing objects/blobs. For more information about storage accounts, see Azure storage account.
- **Container:** A container provides a grouping of a set of blobs. All blobs must be in a container. An account can contain an unlimited number of containers. A container can store an unlimited number of blobs. Note that the container name must be lowercase.
- **Blob:** A file of any type and size. Azure Storage offers three types of blobs: block blobs, page blobs, and append blobs.

  Block blobs are ideal for storing text or binary files, such as documents and media files. *Append blobs* are similar to block blobs in that they are made up of blocks, but they are optimized for append operations, so they are useful for logging scenarios. A single block blob can contain up to 50,000 blocks of up to 100 MB each, for a total size of slightly more than 4.75 TB (100 MB X 50,000). A single append blob can contain up to 50,000 blocks of up to 4 MB each, for a total size of slightly more than 195 GB (4 MB X 50,000).

Page blobs can be up to 1 TB in size, and are more efficient for frequent read/write operations. Azure Virtual Machines use page blobs as OS and data disks.

For details about naming containers and blobs, see Naming and Referencing Containers, Blobs, and Metadata.

Create an Azure storage account

The easiest way to create your first Azure storage account is by using the Azure Portal. To learn more, see Create a storage account.

You can also create an Azure storage account by using Azure PowerShell, Azure CLI, or the Storage Resource Provider Client Library for .NET.

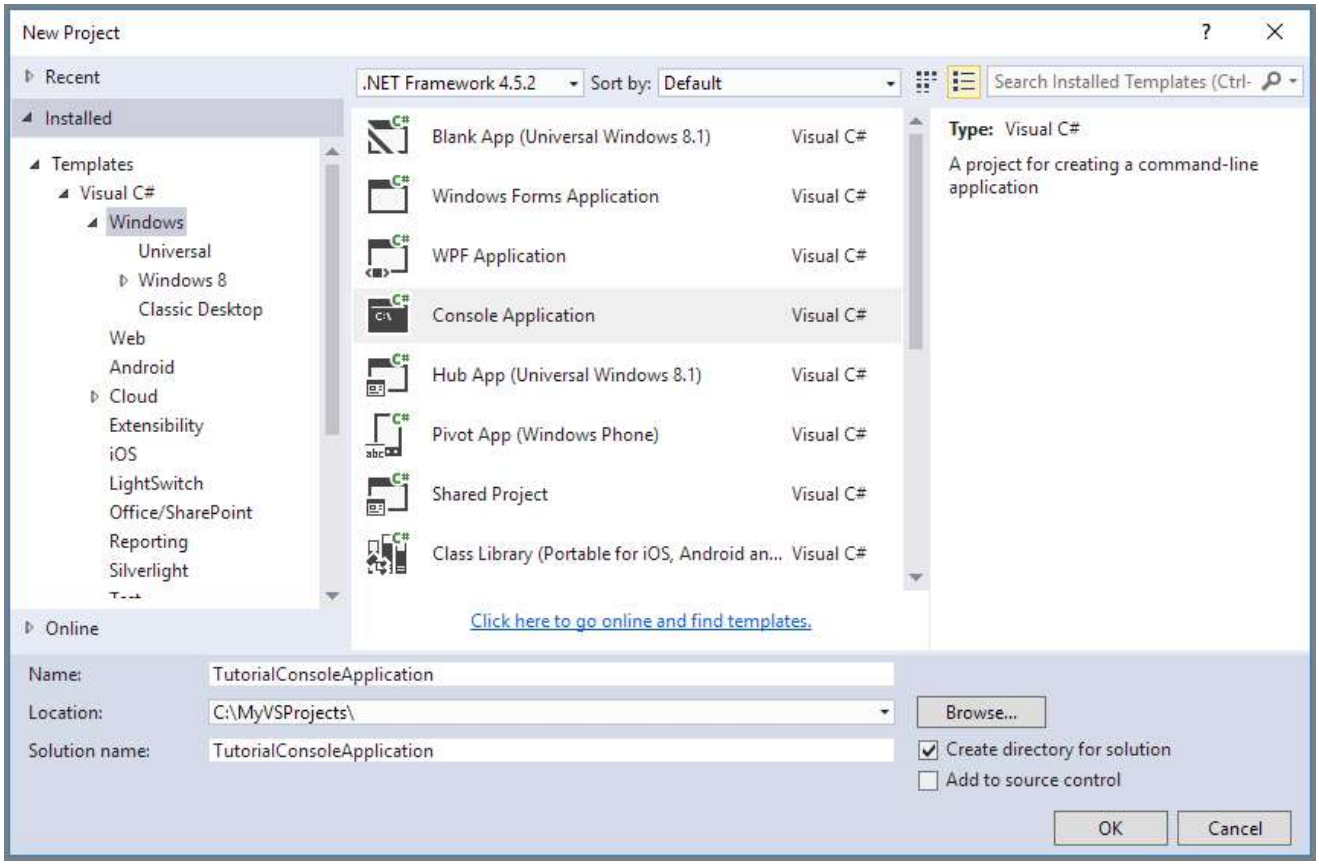
If you prefer not to create a storage account at this time, you can also use the Azure storage emulator to run and test your code in a local environment. For more information, see Use the Azure Storage Emulator for Development and Testing.

Set up your development environment

Next, set up your development environment in Visual Studio so that you are ready to try the code examples provided in this guide.

Create a Windows console application project

In Visual Studio, create a new Windows console application, as shown:



All of the code examples in this tutorial can be added to the **Main()** method in `program.cs` in your console application.

Note that you can use the Azure Storage Client Library from any type of .NET application, including an Azure cloud service, an Azure web app, a desktop application, or a mobile application. In this guide, we use a console application for simplicity.

Use NuGet to install the required packages

There are two packages that you'll need to install to your project to complete this tutorial:

- Microsoft Azure Storage Client Library for .NET: This package provides programmatic access to data resources in your storage account.
- Microsoft Azure Configuration Manager library for .NET: This package provides a class for parsing a connection string from a configuration file, regardless of where your application is running.

You can use NuGet to obtain both packages. Follow these steps:

1. Right-click your project in **Solution Explorer** and choose **Manage NuGet Packages**.
2. Search online for "WindowsAzure.Storage" and click **Install** to install the Storage Client Library and its dependencies.

3. Search online for "ConfigurationManager" and click **Install** to install the Azure Configuration Manager.

> **Note**
>
> The Storage Client Library package is also included in the Azure SDK for .NET. However, we recommend that you also install the Storage Client Library from NuGet to ensure that you always have the latest version of the client library.
>
> The ODataLib dependencies in the Storage Client Library for .NET are resolved through the ODataLib (version 5.0.2 and greater) packages available through NuGet, and not through WCF Data Services. The ODataLib libraries can be downloaded directly or referenced by your code project through NuGet. The specific ODataLib packages used by the Storage Client Library are OData, Edm, and Spatial. While these libraries are used by the Azure Table storage classes, they are required dependencies for programming with the Storage Client Library.

Determine your target environment

You have two environment options for running the examples in this guide:

- You can run your code against an Azure Storage account in the cloud.
- You can run your code against the Azure storage emulator. The storage emulator is a local environment that emulates an Azure Storage account in the cloud. The emulator is a free option for testing and debugging your code while your application is under development. The emulator uses a well-known account and key. For more details, see Use the Azure Storage Emulator for Development and Testing

If you are targeting a storage account in the cloud, copy the primary access key for your storage account from the Azure Portal. For more information, see View and copy storage access keys.

> **Note**
>
> You can target the storage emulator to avoid incurring any costs associated with Azure Storage. However, if you do choose to target an Azure storage account in the cloud, costs for performing this tutorial will be negligible.

Configure your storage connection string

The Azure Storage Client Library for .NET supports using a storage connection string to configure endpoints and credentials for accessing storage services. The best way to maintain your storage connection string is in a configuration file.

For more information about connection strings, see Configure a Connection String to Azure Storage.

> **Note**
>
> Your storage account key is similar to the root password for your storage account. Always be careful to protect your storage account key. Avoid distributing it to other users, hard-coding it, or saving it in a plain-text file that is accessible to others. Regenerate your key using the Azure Portal if you believe it may have been compromised.

To configure your connection string, open the `app.config` file from Solution Explorer in Visual Studio. Add the contents of the `<appSettings>` element shown below. Replace `account-name` with the name of your storage account, and `account-key` with your account access key:

xml Copy

```
<configuration>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5.2" />
    </startup>
      <appSettings>
        <add key="StorageConnectionString" value="DefaultEndpointsProtocol=https;Accoun
      </appSettings>
</configuration>
```

For example, your configuration setting will be similar to:

xml Copy

```
<add key="StorageConnectionString" value="DefaultEndpointsProtocol=https;AccountName=s
```

To target the storage emulator, you can use a shortcut that maps to the well-known account name and key. In that case, your connection string setting will be:

xml Copy

```
<add key="StorageConnectionString" value="UseDevelopmentStorage=true;" />
```

Add namespace declarations

Add the following **using** statements to the top of the `program.cs` file:

C# Copy

```
using Microsoft.Azure; // Namespace for CloudConfigurationManager
using Microsoft.WindowsAzure.Storage; // Namespace for CloudStorageAccount
using Microsoft.WindowsAzure.Storage.Blob; // Namespace for Blob storage types
```

Parse the connection string

The Microsoft Azure Configuration Manager Library for .NET provides a class for parsing a connection string from a configuration file. The CloudConfigurationManager class parses configuration settings regardless of whether the client application is running on the desktop, on a mobile device, in an Azure virtual machine, or in an Azure cloud service.

To reference the CloudConfigurationManager package, add the following `using` directive:

C# Copy

```
using Microsoft.Azure;    //Namespace for CloudConfigurationManager
```

Here's an example that shows how to retrieve a connection string from a configuration file:

C# Copy

```
// Parse the connection string and return a reference to the storage account.
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
    CloudConfigurationManager.GetSetting("StorageConnectionString"));
```

Using the Azure Configuration Manager is optional. You can also use an API like the .NET Framework's ConfigurationManager class.

Create the Blob service client

The **CloudBlobClient** class enables you to retrieve containers and blobs stored in Blob

+ Options

ervice client:

C# Copy

```
CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();
```

Now you are ready to write code that reads data from and writes data to Blob storage.

Create a container

Every blob in Azure storage must reside in a container. The container forms part of the blob name. For example, `mycontainer` is the name of the container in these sample blob URIs:

Copy

```
https://storagesample.blob.core.windows.net/mycontainer/blob1.txt
https://storagesample.blob.core.windows.net/mycontainer/photos/myphoto.jpg
```

A container name must be a valid DNS name, conforming to the following naming rules:

1. Container names must start with a letter or number, and can contain only letters, numbers, and the dash (-) character.
2. Every dash (-) character must be immediately preceded and followed by a letter or number; consecutive dashes are not permitted in container names.
3. All letters in a container name must be lowercase.
4. Container names must be from 3 through 63 characters long.

> **Important**
>
> Note that the name of a container must always be lowercase. If you include an upper-case letter in a container name, or otherwise violate the container naming rules, you may receive a 400 error (Bad Request).

This example shows how to create a container if it does not already exist:

C# Copy

```
// Retrieve storage account from connection string.
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
    CloudConfigurationManager.GetSetting("StorageConnectionString"));

// Create the blob client.
CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();
```

+ Options

```
ainer.
CloudBlobContainer container = blobClient.GetContainerReference("mycontainer");

// Create the container if it doesn't already exist.
container.CreateIfNotExists();
```

By default, the new container is private, meaning that you must specify your storage access key to download blobs from this container. If you want to make the files within the container

available to everyone, you can set the container to be public using the following code:

C# Copy

```
container.SetPermissions(
    new BlobContainerPermissions { PublicAccess = BlobContainerPublicAccessType.Blob }
```

Anyone on the Internet can see blobs in a public container, but you can modify or delete them only if you have the appropriate account access key or a shared access signature.

Upload a blob into a container

Azure Blob Storage supports block blobs and page blobs. In the majority of cases, block blob is the recommended type to use.

To upload a file to a block blob, get a container reference and use it to get a block blob reference. Once you have a blob reference, you can upload any stream of data to it by calling the **UploadFromStream** method. This operation will create the blob if it didn't previously exist, or overwrite it if it does exist.

The following example shows how to upload a blob into a container and assumes that the container was already created.

C# Copy

```
// Retrieve storage account from connection string.
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
    CloudConfigurationManager.GetSetting("StorageConnectionString"));

// Create the blob client.
CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();

// Retrieve reference to a previously created container.
CloudBlobContainer container = blobClient.GetContainerReference("mycontainer");

                                    amed "myblob".
+ Options                           ner.GetBlockBlobReference("myblob");

// Create or overwrite the "myblob" blob with contents from a local file.
using (var fileStream = System.IO.File.OpenRead(@"path\myfile"))
{
    blockBlob.UploadFromStream(fileStream);
}
```

List the blobs in a container

To list the blobs in a container, first get a container reference. You can then use the container's **ListBlobs** method to retrieve the blobs and/or directories within it. To access the rich set of properties and methods for a returned **IListBlobItem**, you must cast it to a **CloudBlockBlob**, **CloudPageBlob**, or **CloudBlobDirectory** object. If the type is unknown, you can use a type check to determine which to cast it to. The following code demonstrates how to retrieve and output the URI of each item in the *photos* container:

C# Copy

```
// Retrieve storage account from connection string.
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
    CloudConfigurationManager.GetSetting("StorageConnectionString"));

// Create the blob client.
CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();

// Retrieve reference to a previously created container.
CloudBlobContainer container = blobClient.GetContainerReference("photos");

// Loop over items within the container and output the length and URI.
foreach (IListBlobItem item in container.ListBlobs(null, false))
{
    if (item.GetType() == typeof(CloudBlockBlob))
    {
        CloudBlockBlob blob = (CloudBlockBlob)item;

        Console.WriteLine("Block blob of length {0}: {1}", blob.Properties.Length, blo

    }
    else if (item.GetType() == typeof(CloudPageBlob))
    {
        CloudPageBlob pageBlob = (CloudPageBlob)item;

        Console.WriteLine("Page blob of length {0}: {1}", pageBlob.Properties.Length,

    }
    else if (item.GetType() == typeof(CloudBlobDirectory))
    {
        CloudBlobDirectory directory = (CloudBlobDirectory)item;

        + Options                                    ory: {0}", directory.Uri);

    }
}
```

As shown above, you can name blobs with path information in their names. This creates a virtual directory structure that you can organize and traverse as you would a traditional file system. Note that the directory structure is virtual only - the only resources available in Blob

storage are containers and blobs. However, the storage client library offers a **CloudBlobDirectory** object to refer to a virtual directory and simplify the process of working with blobs that are organized in this way.

For example, consider the following set of block blobs in a container named *photos*:

Copy

```
photo1.jpg
2010/architecture/description.txt
2010/architecture/photo3.jpg
2010/architecture/photo4.jpg
2011/architecture/photo5.jpg
2011/architecture/photo6.jpg
2011/architecture/description.txt
2011/photo7.jpg
```

When you call **ListBlobs** on the *photos* container (as in the above sample), a hierarchical listing is returned. It contains both **CloudBlobDirectory** and **CloudBlockBlob** objects, representing the directories and blobs in the container, respectively. The resulting output looks like:

Copy

```
Directory: https://<accountname>.blob.core.windows.net/photos/2010/
Directory: https://<accountname>.blob.core.windows.net/photos/2011/
Block blob of length 505623: https://<accountname>.blob.core.windows.net/photos/photo1
```

Optionally, you can set the **UseFlatBlobListing** parameter of the **ListBlobs** method to **true**. In this case, every blob in the container is returned as a **CloudBlockBlob** object. The call to **ListBlobs** to return a flat listing looks like this:

C# Copy

```
// Loop over items within the container and output the length and URI.
                                    ntainer.ListBlobs(null, true))
```

+ Options

```
    ...
}
```

and the results look like this:

Copy

```
Block blob of length 4: https://<accountname>.blob.core.windows.net/photos/2010/archit
Block blob of length 314618: https://<accountname>.blob.core.windows.net/photos/2010/a
Block blob of length 522713: https://<accountname>.blob.core.windows.net/photos/2010/a
Block blob of length 4: https://<accountname>.blob.core.windows.net/photos/2011/archit
Block blob of length 419048: https://<accountname>.blob.core.windows.net/photos/2011/a
Block blob of length 506388: https://<accountname>.blob.core.windows.net/photos/2011/a
Block blob of length 399751: https://<accountname>.blob.core.windows.net/photos/2011/p
Block blob of length 505623: https://<accountname>.blob.core.windows.net/photos/photo1
```

Download blobs

To download blobs, first retrieve a blob reference and then call the **DownloadToStream** method. The following example uses the **DownloadToStream** method to transfer the blob contents to a stream object that you can then persist to a local file.

C# Copy

```
// Retrieve storage account from connection string.
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
    CloudConfigurationManager.GetSetting("StorageConnectionString"));

// Create the blob client.
CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();

// Retrieve reference to a previously created container.
CloudBlobContainer container = blobClient.GetContainerReference("mycontainer");

// Retrieve reference to a blob named "photo1.jpg".
CloudBlockBlob blockBlob = container.GetBlockBlobReference("photo1.jpg");

// Save blob contents to a file.
using (var fileStream = System.IO.File.OpenWrite(@"path\myfile"))
{
    blockBlob.DownloadToStream(fileStream);
}
```

You can also use the **DownloadToStream** method to download the contents of a blob as a

+ Options

C# Copy

```
// Retrieve storage account from connection string.
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
    CloudConfigurationManager.GetSetting("StorageConnectionString"));

// Create the blob client.
CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();

// Retrieve reference to a previously created container.
CloudBlobContainer container = blobClient.GetContainerReference("mycontainer");

// Retrieve reference to a blob named "myblob.txt"
CloudBlockBlob blockBlob2 = container.GetBlockBlobReference("myblob.txt");

string text;
using (var memoryStream = new MemoryStream())
{
    blockBlob2.DownloadToStream(memoryStream);
    text = System.Text.Encoding.UTF8.GetString(memoryStream.ToArray());
}
```

Delete blobs

To delete a blob, first get a blob reference and then call the **Delete** method on it.

C# Copy

```
// Retrieve storage account from connection string.
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
    CloudConfigurationManager.GetSetting("StorageConnectionString"));

// Create the blob client.
CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();

// Retrieve reference to a previously created container.
CloudBlobContainer container = blobClient.GetContainerReference("mycontainer");

// Retrieve reference to a blob named "myblob.txt".
CloudBlockBlob blockBlob = container.GetBlockBlobReference("myblob.txt");

// Delete the blob.
```

+ Options

List blobs in pages asynchronously

If you are listing a large number of blobs, or you want to control the number of results you return in one listing operation, you can list blobs in pages of results. This example shows

how to return results in pages asynchronously, so that execution is not blocked while waiting to return a large set of results.

This example shows a flat blob listing, but you can also perform a hierarchical listing, by setting the *useFlatBlobListing* parameter of the **ListBlobsSegmentedAsync** method to *false*.

Because the sample method calls an asynchronous method, it must be prefaced with the *async* keyword, and it must return a **Task** object. The await keyword specified for the **ListBlobsSegmentedAsync** method suspends execution of the sample method until the listing task completes.

C# Copy

```
async public static Task ListBlobsSegmentedInFlatListing(CloudBlobContainer container)
{
    //List blobs to the console window, with paging.
    Console.WriteLine("List blobs in pages:");

    int i = 0;
    BlobContinuationToken continuationToken = null;
    BlobResultSegment resultSegment = null;

    //Call ListBlobsSegmentedAsync and enumerate the result segment returned, while th
    //When the continuation token is null, the last page has been returned and executi
    do
    {
        //This overload allows control of the page size. You can return all remaining
        //or by calling a different overload.
        resultSegment = await container.ListBlobsSegmentedAsync("", true, BlobListingD
        if (resultSegment.Results.Count<IListBlobItem>() > 0) { Console.WriteLine("Pag
        foreach (var blobItem in resultSegment.Results)
        {
            Console.WriteLine("\t{0}", blobItem.StorageUri.PrimaryUri);
        }
        Console.WriteLine();

        //Get the continuation token.
        continuationToken = resultSegment.ContinuationToken;
    }
    while (continuationToken != null);
}
```

+ Options

Writing to an append blob

An append blob is a new type of blob, introduced with version 5.x of the Azure storage client library for .NET. An append blob is optimized for append operations, such as logging. Like a block blob, an append blob is comprised of blocks, but when you add a new block to an

append blob, it is always appended to the end of the blob. You cannot update or delete an existing block in an append blob. The block IDs for an append blob are not exposed as they are for a block blob.

Each block in an append blob can be a different size, up to a maximum of 4 MB, and an append blob can include a maximum of 50,000 blocks. The maximum size of an append blob is therefore slightly more than 195 GB (4 MB X 50,000 blocks).

The example below creates a new append blob and appends some data to it, simulating a simple logging operation.

C# Copy

```
//Parse the connection string for the storage account.
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
    Microsoft.Azure.CloudConfigurationManager.GetSetting("StorageConnectionString"));

//Create service client for credentialed access to the Blob service.
CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();

//Get a reference to a container.
CloudBlobContainer container = blobClient.GetContainerReference("my-append-blobs");

//Create the container if it does not already exist.
container.CreateIfNotExists();

//Get a reference to an append blob.
CloudAppendBlob appendBlob = container.GetAppendBlobReference("append-blob.log");

//Create the append blob. Note that if the blob already exists, the CreateOrReplace()
//You can check whether the blob exists to avoid overwriting it by using CloudAppendBl
appendBlob.CreateOrReplace();

int numBlocks = 10;

//Generate an array of random bytes.
Random rnd = new Random();
byte[] bytes = new byte[numBlocks];
rnd.NextBytes(bytes);

//Simulate a logging operation by writing text data and byte data to the end of the ap
                                    +)
        + Options
    appendBlob.AppendText(String.Format("Timestamp: {0:u} \tLog Entry: {1}{2}",
        DateTime.UtcNow, bytes[i], Environment.NewLine));
}

//Read the append blob to the console window.
Console.WriteLine(appendBlob.DownloadText());
```

See Understanding Block Blobs, Page Blobs, and Append Blobs for more information about the differences between the three types of blobs.

Managing security for blobs

By default, Azure Storage keeps your data secure by limiting access to the account owner, who is in possession of the account access keys. When you need to share blob data in your storage account, it is important to do so without compromising the security of your account access keys. Additionally, you can encrypt blob data to ensure that it is secure going over the wire and in Azure Storage.

> **Important**
>
> Your storage account key is similar to the root password for your storage account. Always be careful to protect your account key. Avoid distributing it to other users, hard-coding it, or saving it in a plain-text file that is accessible to others. Regenerate your account key using the Azure Portal if you believe it may have been compromised. To learn how to regenerate your account key, see How to create, manage, or delete a storage account in the Azure Portal.

Controlling access to blob data

By default, the blob data in your storage account is accessible only to storage account owner. Authenticating requests against Blob storage requires the account access key by default. However, you may wish to make certain blob data available to other users. You have two options:

- **Anonymous access:** You can make a container or its blobs publicly available for anonymous access. See Manage anonymous read access to containers and blobs for more information.
- **Shared access signatures:** You can provide clients with a shared access signature (SAS), which provides delegated access to a resource in your storage account, with permissions that you specify and over an interval that you specify. See Using Shared Access Signatures

+ Options

Encrypting blob data

Azure Storage supports encrypting blob data both at the client and on the server:

- **Client-side encryption:** The Storage Client Library for .NET supports encrypting data within client applications before uploading to Azure Storage, and decrypting data while

downloading to the client. The library also supports integration with Azure Key Vault for storage account key management. See Client-Side Encryption with .NET for Microsoft Azure Storage for more information. Also see Tutorial: Encrypt and decrypt blobs in Microsoft Azure Storage using Azure Key Vault.

- **Server-side encryption**: Azure Storage now supports server-side encryption. See Azure Storage Service Encryption for Data at Rest (Preview).

Next steps

Now that you've learned the basics of Blob storage, follow these links to learn more.

Microsoft Azure Storage Explorer

- Microsoft Azure Storage Explorer (MASE) is a free, standalone app from Microsoft that enables you to work visually with Azure Storage data on Windows, OS X, and Linux.

Blob storage samples

- Getting Started with Azure Blob Storage in .NET

Blob storage reference

- Storage Client Library for .NET reference
- REST API reference

Conceptual guides

- Transfer data with the AzCopy command-line utility
- Get started with File storage for .NET
- How to use Azure blob storage with the WebJobs SDK

+ Follow

Post comment as...

Kristomb Nov 19, 2016

CreateIfNotExists and SetPermissions doesn't seem to exist in .net core. It needs to be CreateIfNotExistsAsync and SetPermissionsAsync.

Like Reply

Seguler-Msft 11 days ago

@Kristomb Thanks. You are correct. Our .NET Core implementation is missing sync methods at the moment but we have a backlog item to add these methods in the future.

Like Reply

ShipraTrivedi Nov 17, 2016

Hi, I want to open my excel files stored in azure blob in Excel Web App. How can I do that ?

Like Reply

Seguler-Msft 11 days ago

@ShipraTrivedi You can download the blob using the Storage Explorer, or via a SAS token using any browser and then open it in excel.

Like Reply

English