GlobalLogic®

An Introduction to Windows Azure Blob Storage

Tanveer Ahmed Siddiqui
Senior Consultant Engineer

This document introduces the Windows Azure Blob storage service and demonstrates how to create or set up a storage account on Windows Azure.

This document also provides code samples for performing different operations on Windows Azure Storage for binary large objects ("blobs"), including creating, deleting, copying, listing, setting access control, adding properties, adding metadata, etc.

The code examples are written in C# and use the Windows Azure Storage Client Library for .NET (Microsoft.Windows Azure.Storage Client).

www.globallogic.com

Introduction

The term “blob” is commonly understood to mean “Binary Large OBject.” Many of us are familiar with this term from its usage in a database, where “blob data” might be data stored in our database that does not conform to an established data type as defined by the database. Such data is usually persisted as plain binary data.

Windows Azure Blob storage offers a way to store content and make it available across the web. It’s like a storage drive on the cloud that is used to store large amounts of data. You can store files and folders of any format and type – just like the drive on your computer – and share it across the web.

If you go by Microsoft’s definition, “Windows Azure Blob storage is a service for storing large amounts of unstructured data that can be accessed from anywhere in the world via HTTP or HTTPS.”

In the below example, myaccount is the name of the storage account, mycontainer is the container name, and myblob is your actual blob.

https://myaccount.blob.core.windows.net/mycontainer/myblob

A blob’s size can be hundreds of GBs, and a storage account can contain up to 200TB of blobs.

Using Blob Storage

According to Microsoft, Azure Blob storage might include (but are definitely not limited to):

- Serving images or documents directly to a browser
- Storing files for distributed access
- Streaming video and audio
- Performing secure backup and disaster recovery
- Storing data for analysis by an on-premise or Windows Azure-hosted service

Blob Storage Components

Storage Account

A storage account is the top-level namespace for accessing blob content, and all storage takes place here. An account can contain an unlimited number of containers, as long as their total size is under 200TB. Windows Azure Storage provides a way to store three types of data:

- Files, inside Blob Storage
- Data, inside Table Storage
- Messages, inside Queue Storage

Container

A container is a grouping for a set of blobs, since blobs cannot be stored directly in a storage account. A container can store an unlimited number of blobs.

Blob

Blob storage is the usual way of storing file-based information in Azure. A blob file can be any type and size. There are two types of blobs that can be stored in Windows Azure Storage: block blobs and page blobs. The below table provides more details about each type of blob.

Block Blob	**Page Blob**
Max Size 200 GB	Max Size 1 TB
Each block can be 4 MB	Each page can be 512 Bytes
A block blob provides the ability to upload a large amount of data very fast, as the upload itself can be done in parallel.	A page blob is best optimized for random access (i.e., specifying an offset in the blob for reading/writing).

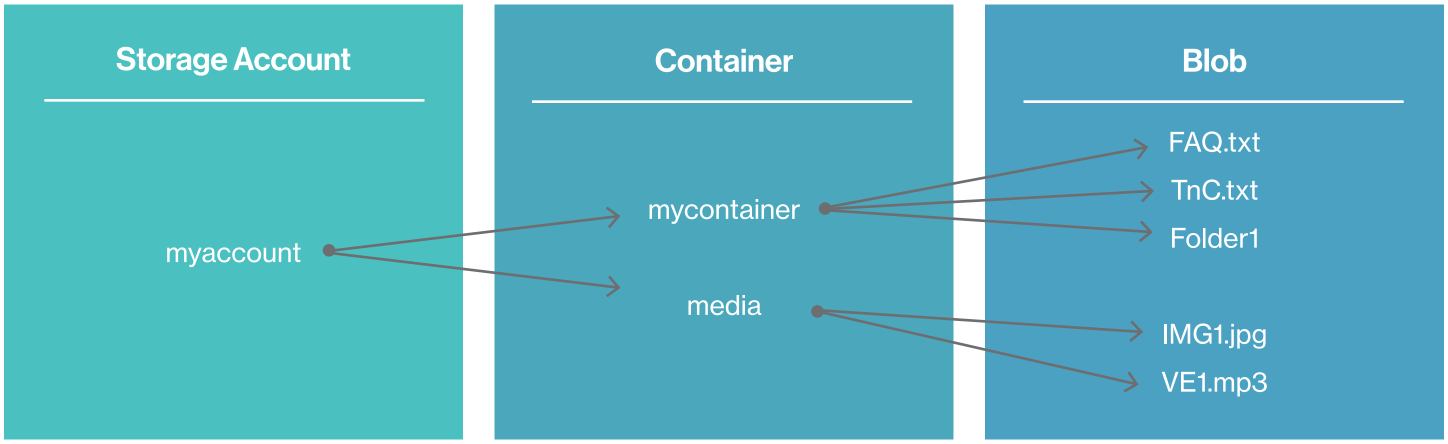


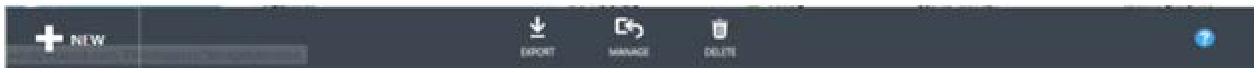
Figure 1: Blob Storage Components

Setting Up a Storage Account on Windows Azure

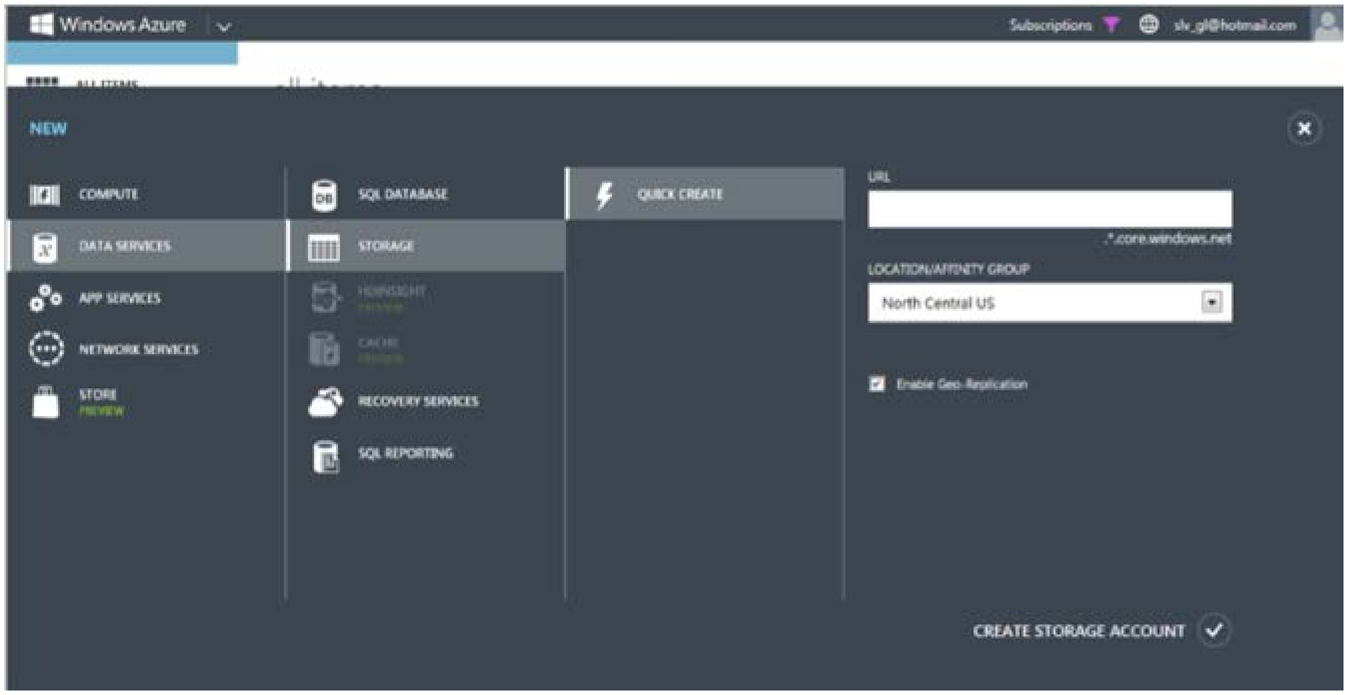
Before going further, we first need to set up a Windows Azure storage account to execute any storage operation. You can set up a storage account by using the Windows Azure Service Management Rest API or via the Windows Azure Management portal. Below is an example of using this second approach.

1. Log into http://manage.windowsazure.com/

2. Click the New on the bottom navigation pane.



3. You will see an open window as shown in figure below.

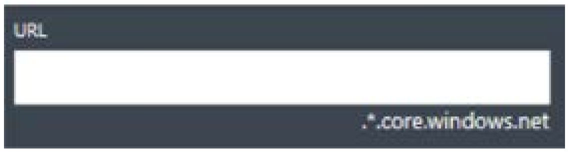


4. Click Data Services.

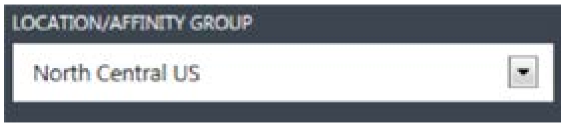
5. Click Storage.

6. Click Quick Create.

7. In the URL textbox, enter a storage account name to use in the URI. The name should be 3-24 lowercase letters and numbers. This value becomes the host name within the URI that is used to address blob, queue, or table resources for the subscription.



8. Under the Location/Affinity Group dropdown menu, select a Region/Affinity Group in which to locate the storage. If you will be using storage from your Windows Azure application, select the same region where you will deploy your application.


LOCATION/AFFINITY GROUP
North Central US

9. Optionally, you can Enable Geo-Replication.

✓ Enable Geo-Replication

10. Click on Create Storage Account.

Geo-Replication for Azure Storage

With geo-replication, Windows Azure Storage now keeps your data durable in two locations. In both locations, Windows Azure Storage constantly maintains multiple healthy replicas of your data.

The location where you read, create, update, or delete data is referred to as the primary location. The primary location exists in the region you choose at the time you create an account via the Azure Portal (e.g., North Central US). The location where your data is geo-replicated is referred to as the secondary location. The secondary location is automatically determined based on the location of the primary; it is in the other data center that is in the same region as the primary location.

In the future, Microsoft plans to support the ability for customers to choose their secondary location, as well as the ability to swap their primary and secondary locations for a storage account.

Geo-replication is included in the current pricing for Azure Storage, and it is called Geo Redundant Storage. If you do not want your data geo-replicated, you can disable geo-replication for your account. This is called Locally Redundant Storage, and it is a 23% to 34% discounted price (depending on how much data is stored) over geo-replicated storage.

When you turn geo-replication off, the data will be deleted from the secondary location. If you decide to turn geo-replication on again after you have turned it off, there is a re-bootstrap egress bandwidth charge (based on the data transfer rates) for copying your existing data from the primary to the secondary location to kick start geo-replication for the storage account. This charge will be applied only when you turn geo-replication on after you have turned it off. There is no additional charge for continuing geo-replication after the re-bootstrap is done.

Currently all storage accounts are bootstrapped and in geo-replication mode between primary and secondary storage locations.

Geo-Location Pairings

Primary	Secondary
North Central US	South Central US
South Central US	North Central US
East US	West US
West US	East US
North Europe	West Europe
West Europe	North Europe
South East Asia	East Asia
East Asia	South East Asia

How Geo-Replication Works

When you create, update, or delete data to your storage account, the transaction is fully replicated on three different storage nodes across three fault domains and update domains. Windows Azure then syncs this same data with the other associated nodes and returns success back to the client.

In the background, the primary location asynchronously replicates the recently committed transaction to the secondary location. That transaction is then made durable by fully replicating it across three different storage nodes in different fault and upgrade domains at the secondary location. Because the updates are asynchronously geo-replicated, there is no change in existing performance for your storage account.

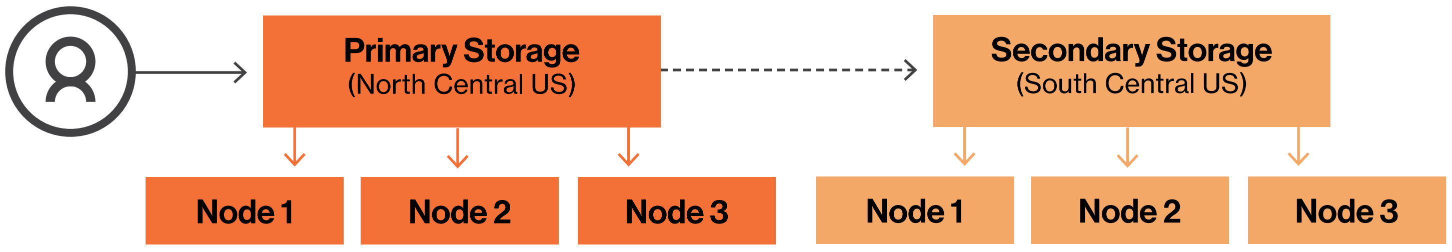


Figure 2: How Geo-Replication Works

How Geo-Failover Works

In the event of a major disaster that affects the primary location, the geo-failover process will first try to restore the primary location. If the primary location cannot be restored, a geo-failover will need to be performed.

As part of the failover, the customer's DNS entry would be updated to point from the primary location to the secondary location. Once this DNS change is propagated, the existing blob and table URIs will work. This means that you do not need to change your application's URIs – all existing URIs will work the same after a geo-failover.

The location that is accepting traffic is considered the new primary location for the storage account. This location will remain as the primary location unless another geo-failover occurs.

Naming and Referencing Containers, Blobs, and Metadata

Resource Name

The URI to reference a container or a blob must be unique. Because every account name is unique, two accounts can have containers with the same name. However, within a given storage account, every container must have a unique name. Every blob within a given container must also have a unique name within that container. If you attempt to create a container or blob with a name that violates the naming rules, the request will fail with the status code 400 (Bad Request).

Container Name

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 they 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 3 to 63 characters long.

Blob Name

A blob name can contain any combination of characters, but reserved URL characters must be properly escaped. A blob name must be at least one character long and cannot be more than 1,024 characters long. Blob names are case-sensitive.

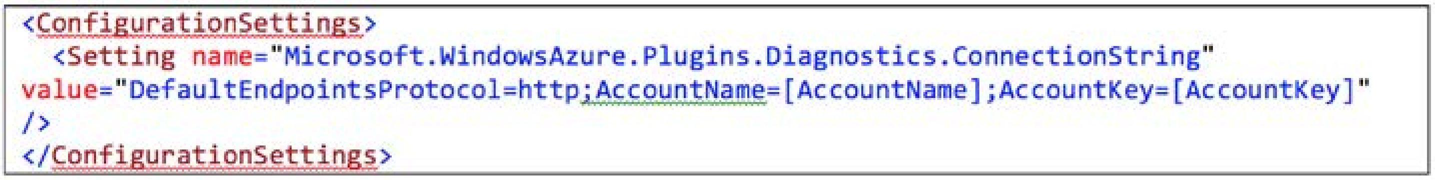
Metadata Name

Metadata for a container or blob resource is stored as name-value pairs associated with the resource. Metadata names must adhere to the naming rules for C# identifiers.

Note that metadata names preserve the case with which they were created, but they are case-insensitive when set or read. If two or more metadata headers with the same name are submitted for a resource, the blob service returns the status code 400 (Bad Request).

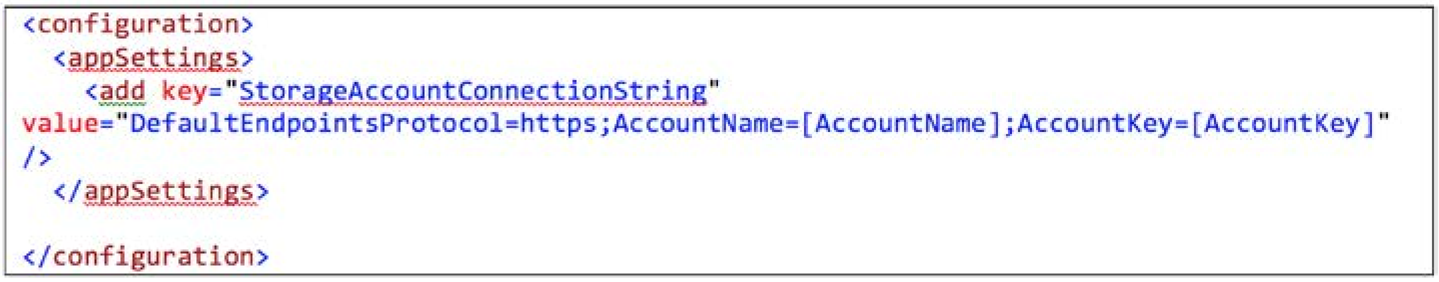
Code Samples for Blob Operations

Before working with blobs, you first need to configure the storage account connection string in your application. When using Windows Azure Cloud Services, it is recommended that you store your connection string using the Windows Azure service configuration system (*.csdef and *.cscfg files).



```
<ConfigurationSettings>
  <Setting name="Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString"
value="DefaultEndpointsProtocol=http;AccountName=[AccountName];AccountKey=[AccountKey]"
/>
</ConfigurationSettings>
```

When using Windows Azure Web Sites, Windows Azure Virtual Machines, or building .NET applications that are intended to run outside of Windows Azure, it is recommended that you store your connection string using the .NET configuration system (e.g. web.config or app.config file).



```
<configuration>
  <appSettings>
    <add key="StorageAccountConnectionString"
value="DefaultEndpointsProtocol=https;AccountName=[AccountName];AccountKey=[AccountKey]"
/>
  </appSettings>

</configuration>
```

Now you need to add assembly reference of Microsoft.WindowsAzure.Storage.dll in your project reference.
You can use NuGet to obtain the Microsoft.WindowsAzure.Storage.dll assembly. Right-click your project in Solution Explorer and choose Manage NuGet Packages. Search online for “WindowsAzure.Storage” and click Install to install the Windows Azure Storage package and dependencies.

Namespace Declaration

Add the following namespace declarations to the top of any C# file in which you wish to programmatically access Windows Azure Storage.

```
using Microsoft.WindowsAzure.Storage;
using Microsoft.WindowsAzure.Storage.Auth;
using Microsoft.WindowsAzure.Storage.Blob;
```

Retrieve Storage Connection String

You can use the CloudStorageAccount type to represent your storage account information. If you are using a Windows Azure project, you can you use the CloudConfigurationManager type to retrieve your storage connection string and storage account information from the Windows Azure service configuration.

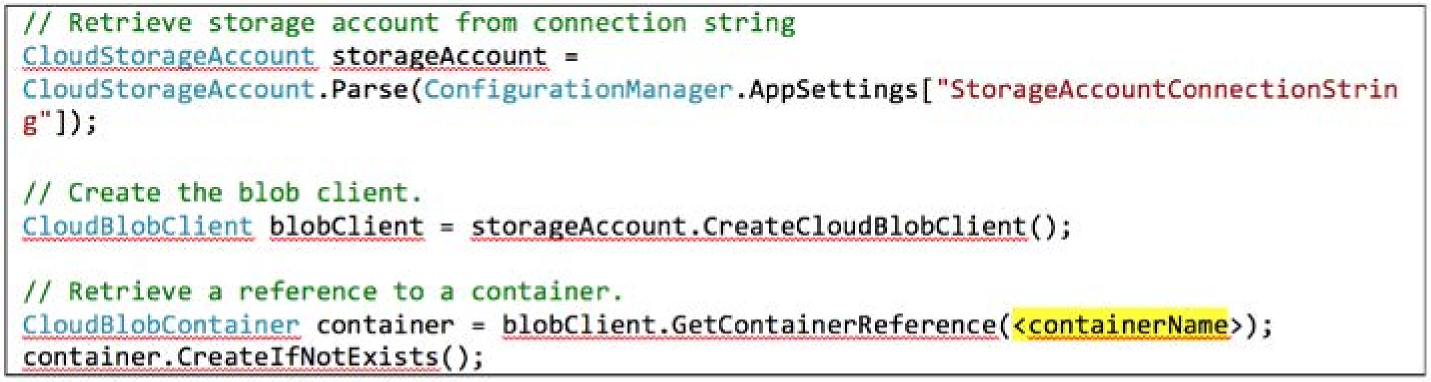
```
// Retrieve storage account from connection string
CloudStorageAccount storageAccount =
CloudStorageAccount.Parse(CloudConfigurationManager.GetSetting("StorageAccountConnectionS
tring"));

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

If you are using Windows Azure Web Sites, Windows Azure Virtual Machines, or building .NET applications that are intended to run outside of Windows Azure, and your connection string is located in the web.config or app.config (as show above), then you can use ConfigurationManager to retrieve the connection string. You will need to add a reference to System.Configuration.dll to your project, and add another namespace declaration for it.

```
// Retrieve storage account from connection string
CloudStorageAccount storageAccount =
CloudStorageAccount.Parse(ConfigurationManager.AppSettings["StorageAccountConnectionStrin
g"]);

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

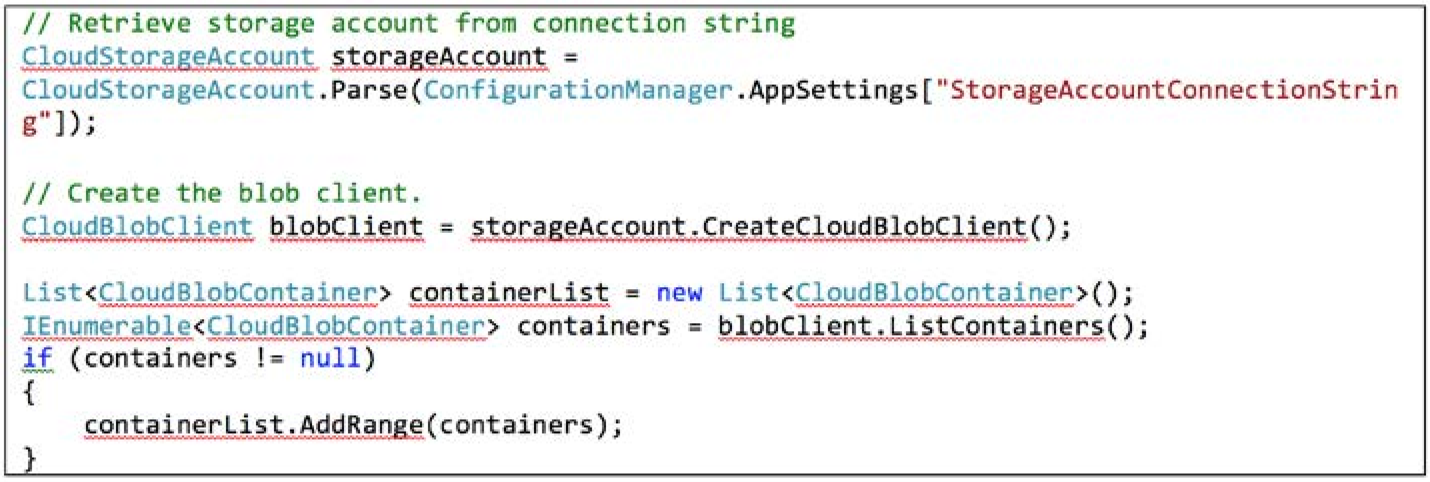
A CloudBlobClient type allows you to retrieve objects that represent containers and blobs stored within the Blob Storage Service. The below diagrams contain code snippets for the following functions.

Create Container



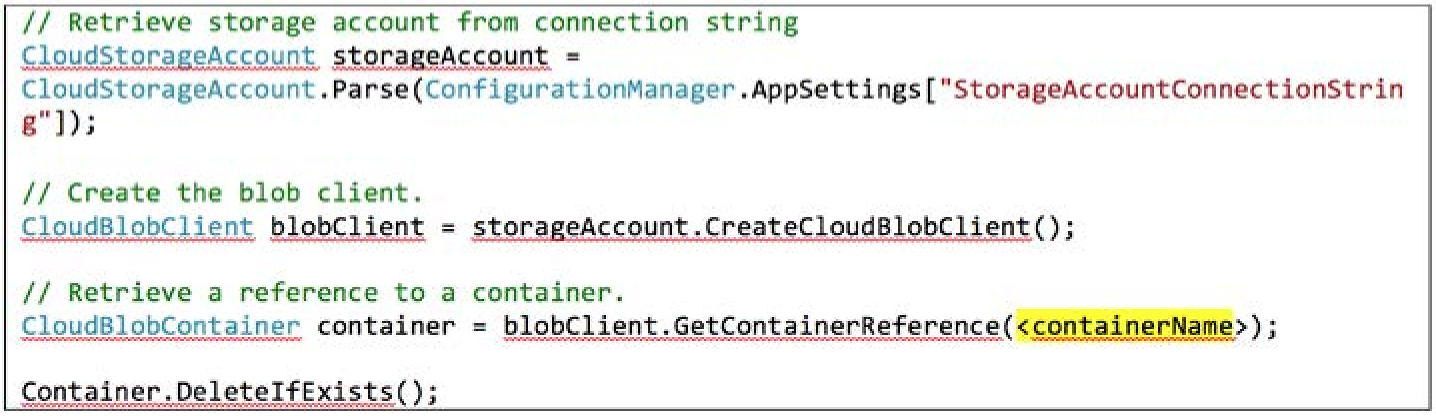
```
// Retrieve storage account from connection string
CloudStorageAccount storageAccount =
CloudStorageAccount.Parse(ConfigurationManager.AppSettings["StorageAccountConnectionStrin
g"]);

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

// Retrieve a reference to a container.
CloudBlobContainer container = blobClient.GetContainerReference(<containerName>);
container.CreateIfNotExists();
```

LIST Container



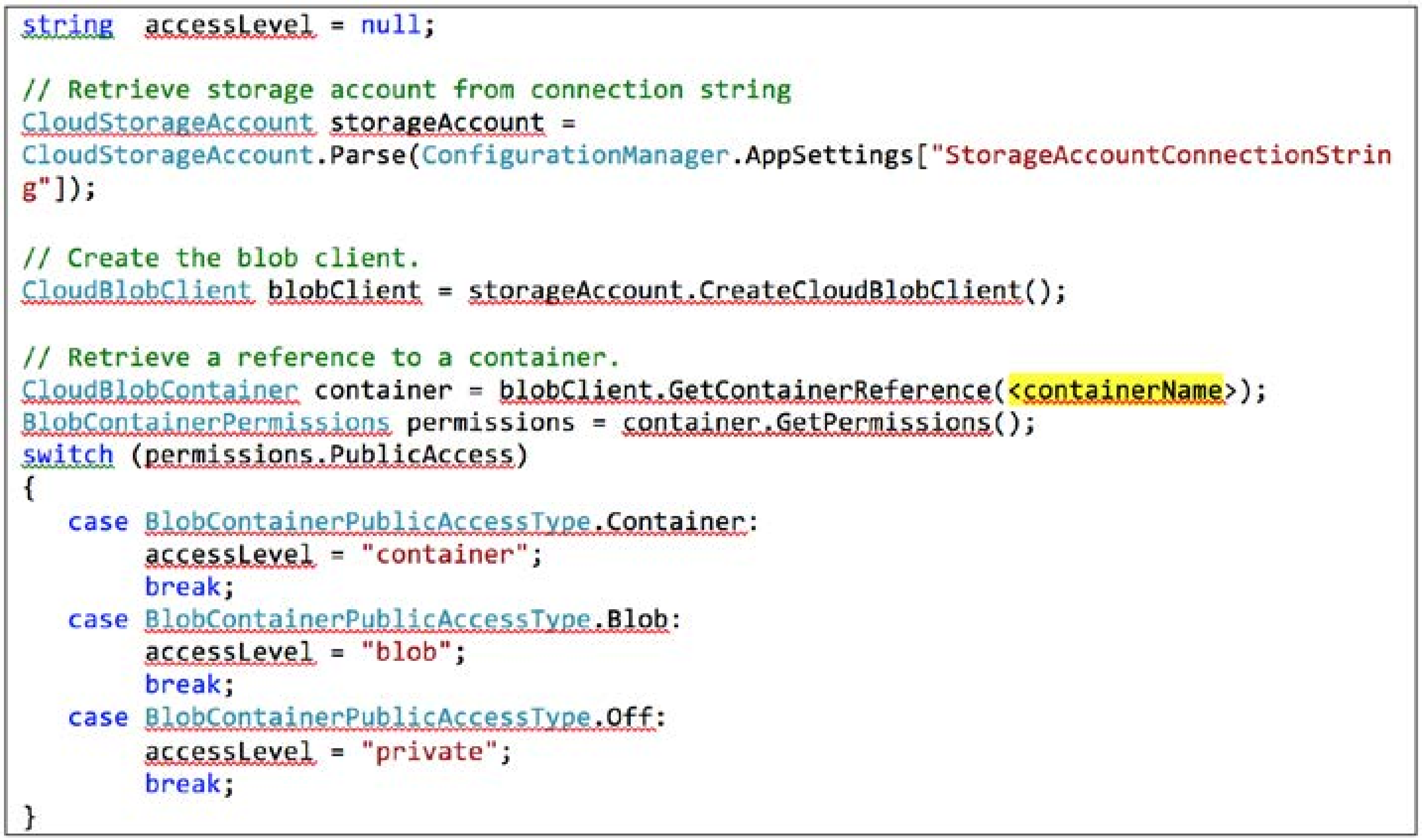
```
// Retrieve storage account from connection string
CloudStorageAccount storageAccount =
CloudStorageAccount.Parse(ConfigurationManager.AppSettings["StorageAccountConnectionStrin
g"]);

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

List<CloudBlobContainer> containerList = new List<CloudBlobContainer>();
IEnumerable<CloudBlobContainer> containers = blobClient.ListContainers();
if (containers != null)
{
    containerList.AddRange(containers);
}
```

Delete Container



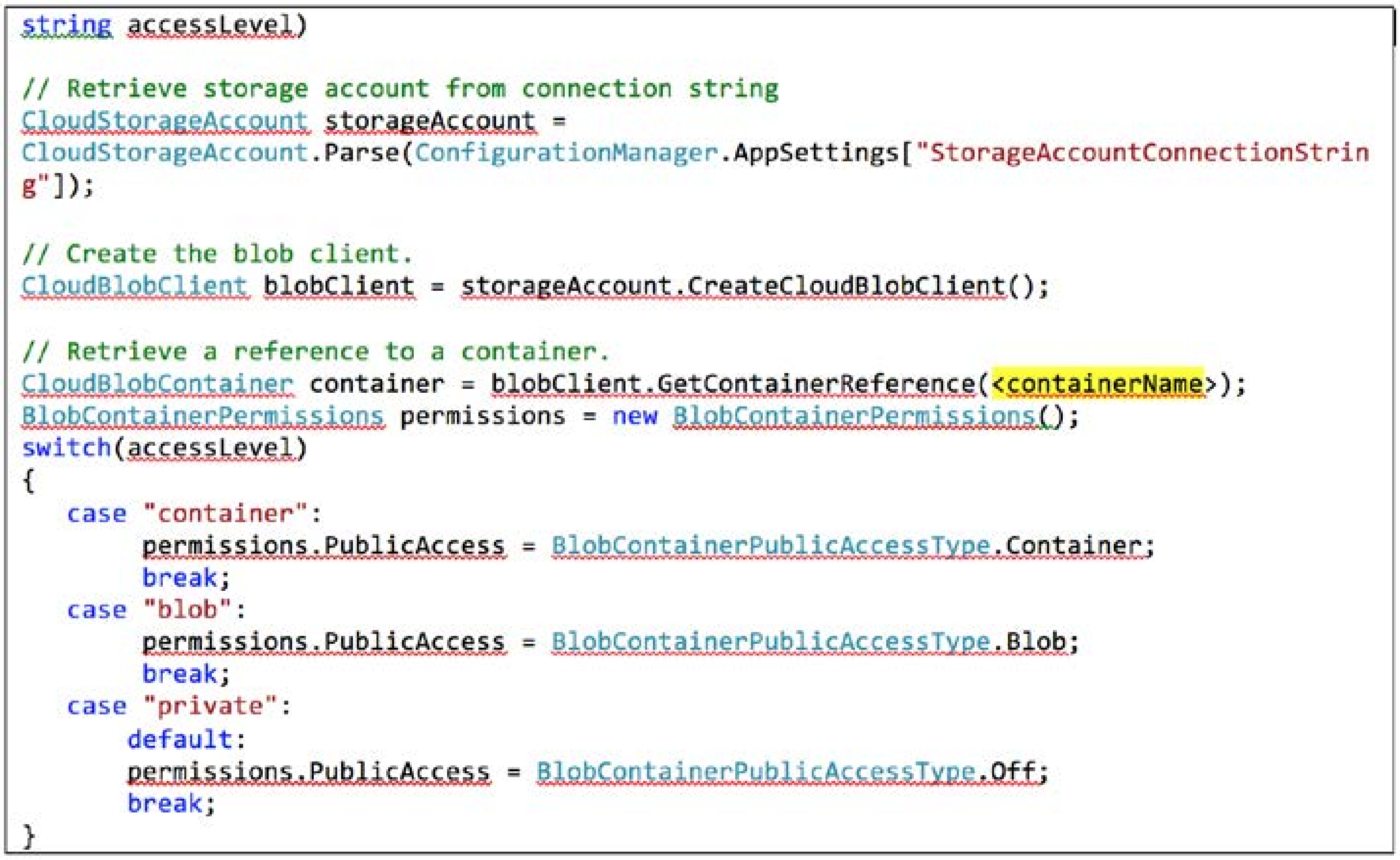
```
// Retrieve storage account from connection string
CloudStorageAccount storageAccount =
CloudStorageAccount.Parse(ConfigurationManager.AppSettings["StorageAccountConnectionStrin
g"]);

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

// Retrieve a reference to a container.
CloudBlobContainer container = blobClient.GetContainerReference(<containerName>);

Container.DeleteIfExists();
```

Get Container Access Control



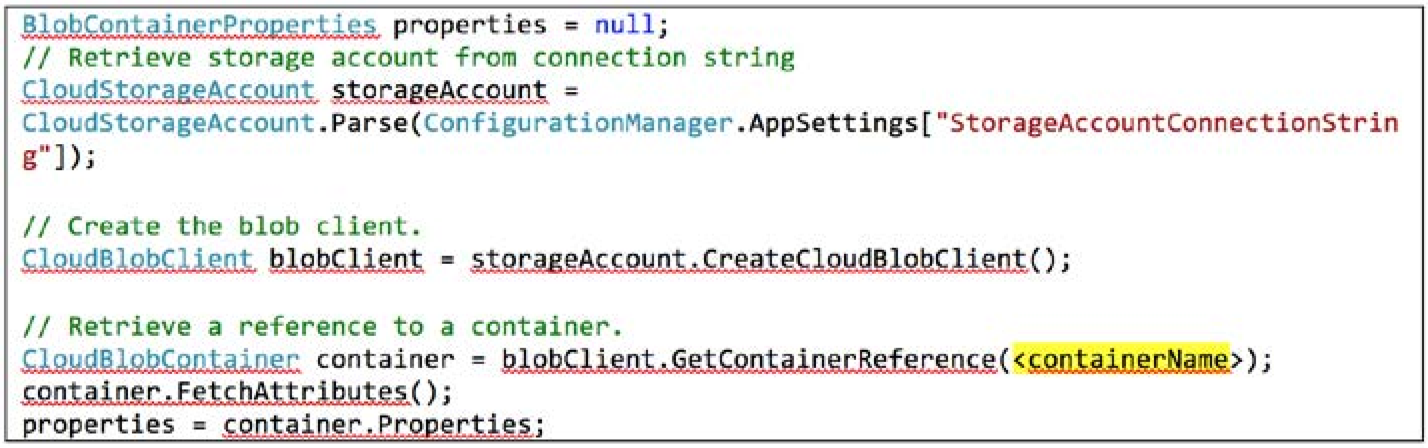
```
string  accessLevel = null;

// Retrieve storage account from connection string
CloudStorageAccount storageAccount =
CloudStorageAccount.Parse(ConfigurationManager.AppSettings["StorageAccountConnectionStrin
g"]);

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

// Retrieve a reference to a container.
CloudBlobContainer container = blobClient.GetContainerReference(<containerName>);
BlobContainerPermissions permissions = container.GetPermissions();
switch (permissions.PublicAccess)
{
    case BlobContainerPublicAccessType.Container:
        accessLevel = "container";
        break;
    case BlobContainerPublicAccessType.Blob:
        accessLevel = "blob";
        break;
    case BlobContainerPublicAccessType.Off:
        accessLevel = "private";
        break;
}
```

Set Container Access Control



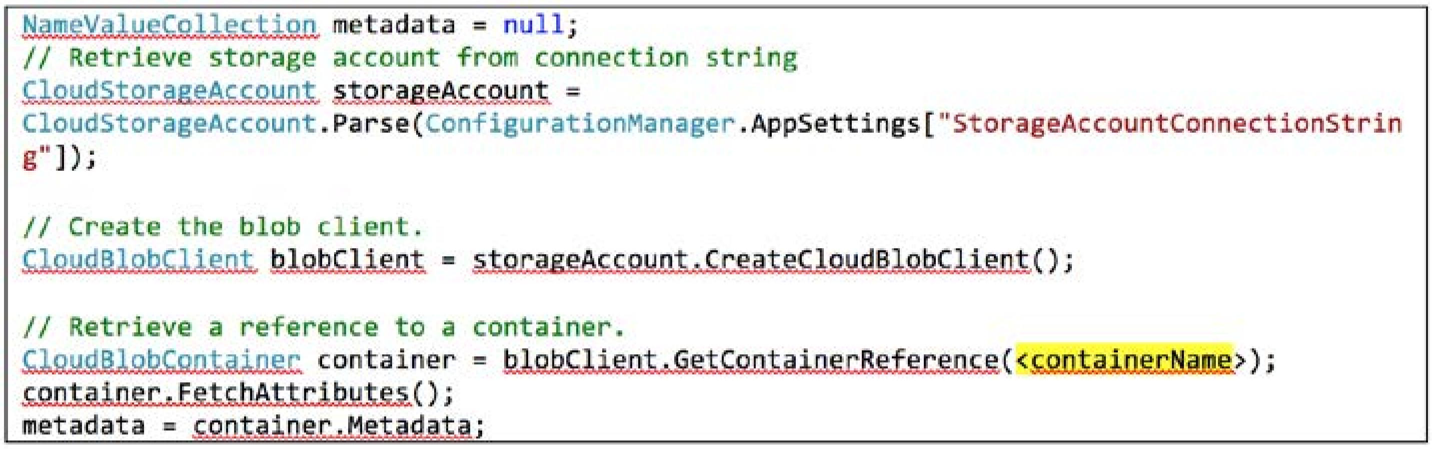
```
string accessLevel)

// Retrieve storage account from connection string
CloudStorageAccount storageAccount =
CloudStorageAccount.Parse(ConfigurationManager.AppSettings["StorageAccountConnectionStrin
g"]);

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

// Retrieve a reference to a container.
CloudBlobContainer container = blobClient.GetContainerReference(<containerName>);
BlobContainerPermissions permissions = new BlobContainerPermissions();
switch(accessLevel)
{
   case "container":
        permissions.PublicAccess = BlobContainerPublicAccessType.Container;
        break;
   case "blob":
        permissions.PublicAccess = BlobContainerPublicAccessType.Blob;
        break;
   case "private":
       default:
       permissions.PublicAccess = BlobContainerPublicAccessType.Off;
       break;
}
```

Get Container Properties



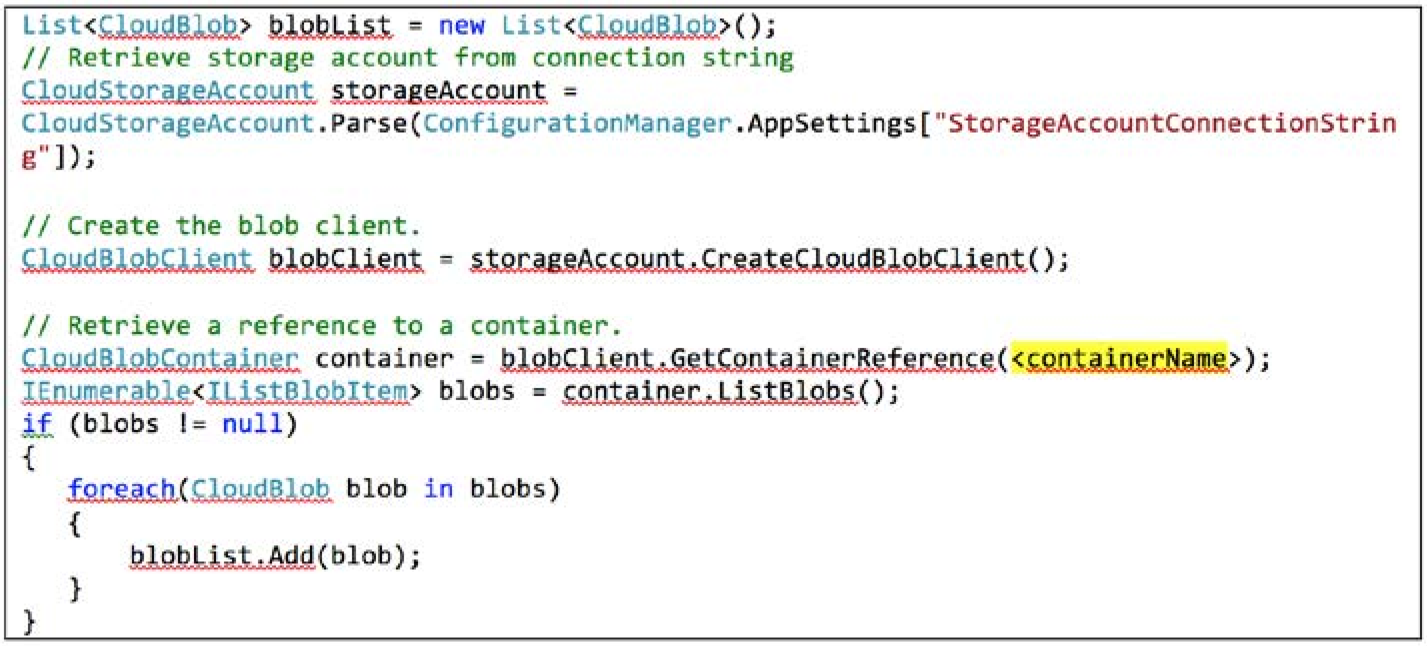
```
BlobContainerProperties properties = null;
// Retrieve storage account from connection string
CloudStorageAccount storageAccount =
CloudStorageAccount.Parse(ConfigurationManager.AppSettings["StorageAccountConnectionStrin
g"]);

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

// Retrieve a reference to a container.
CloudBlobContainer container = blobClient.GetContainerReference(<containerName>);
container.FetchAttributes();
properties = container.Properties;
```

Get Container Metadata



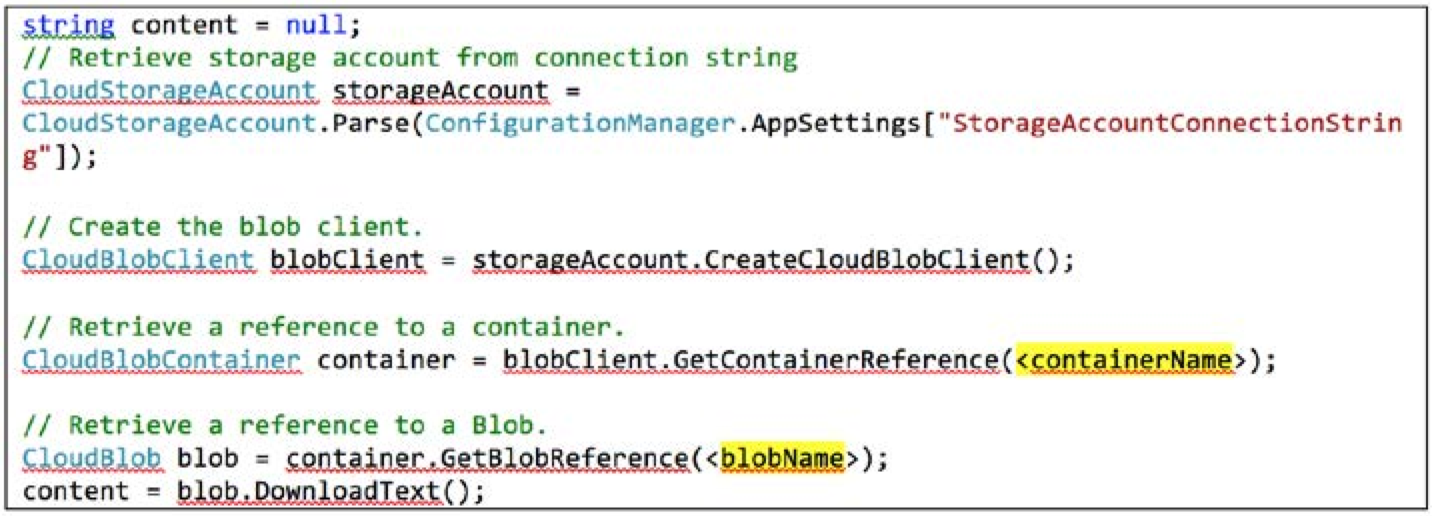
```
NameValueCollection metadata = null;
// Retrieve storage account from connection string
CloudStorageAccount storageAccount =
CloudStorageAccount.Parse(ConfigurationManager.AppSettings["StorageAccountConnectionStrin
g"]);

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

// Retrieve a reference to a container.
CloudBlobContainer container = blobClient.GetContainerReference(<containerName>);
container.FetchAttributes();
metadata = container.Metadata;
```

List Blob



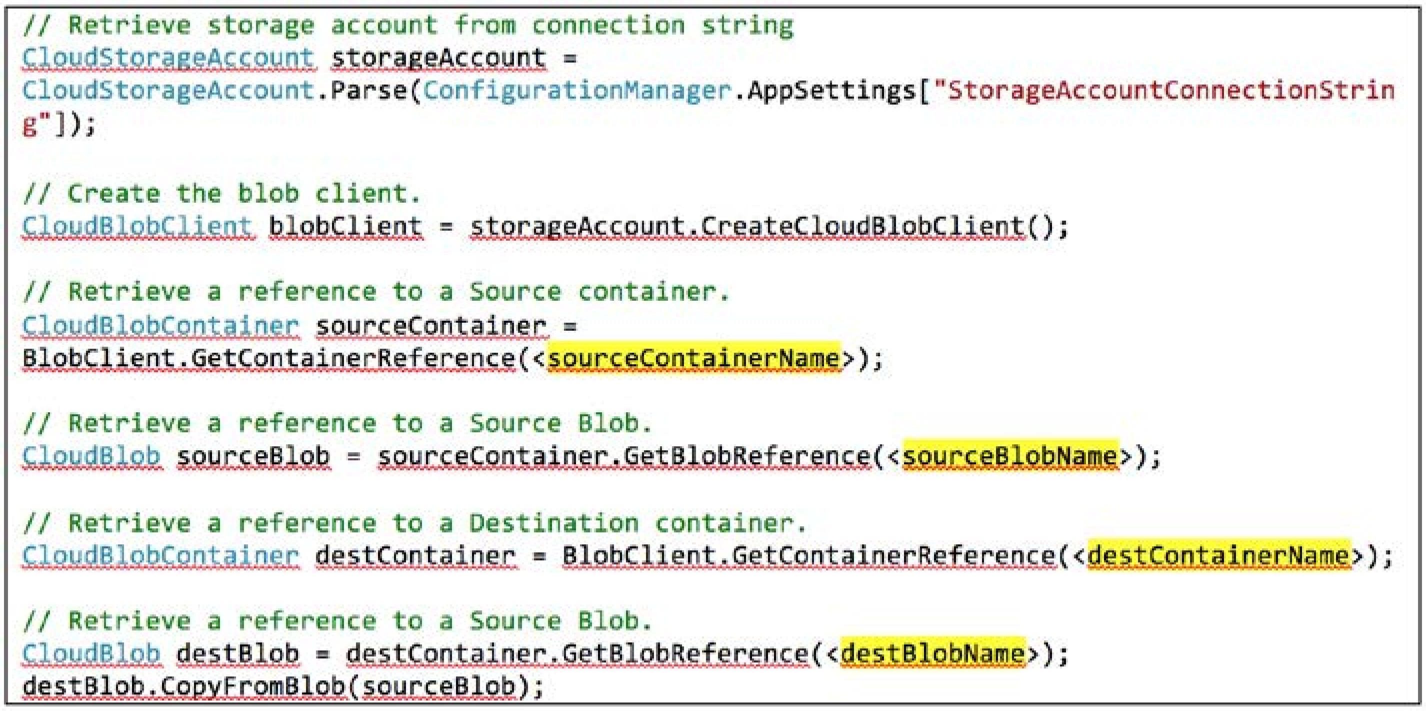
```
List<CloudBlob> blobList = new List<CloudBlob>();
// Retrieve storage account from connection string
CloudStorageAccount storageAccount =
CloudStorageAccount.Parse(ConfigurationManager.AppSettings["StorageAccountConnectionStrin
g"]);

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

// Retrieve a reference to a container.
CloudBlobContainer container = blobClient.GetContainerReference(<containerName>);
IEnumerable<IListBlobItem> blobs = container.ListBlobs();
if (blobs != null)
{
    foreach(CloudBlob blob in blobs)
    {
        blobList.Add(blob);
    }
}
```

Get Blob



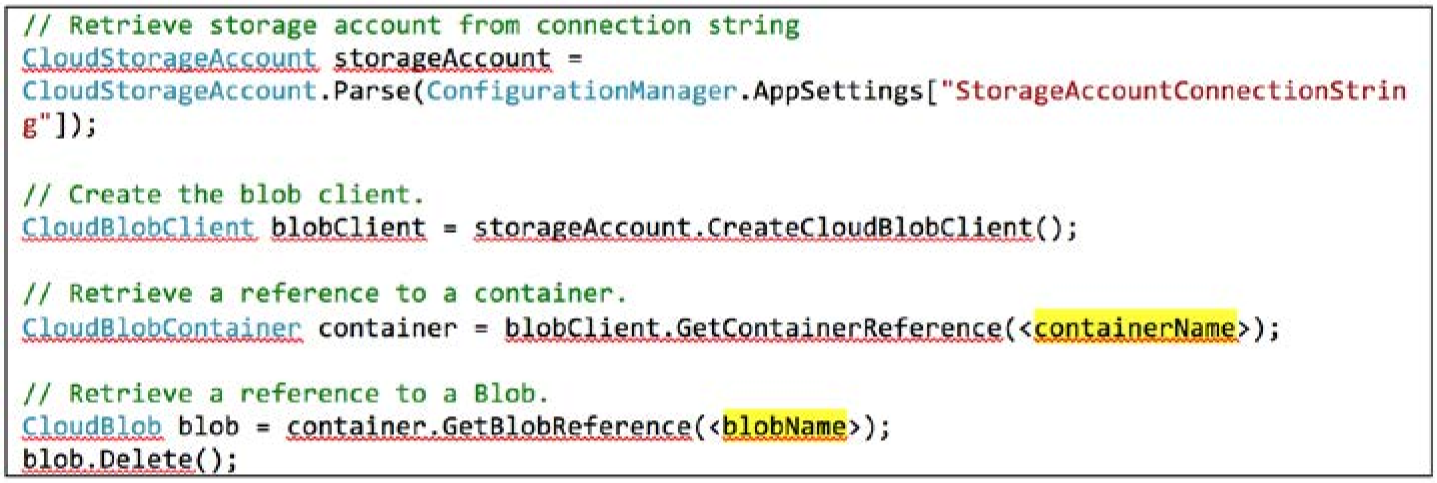
```
string content = null;
// Retrieve storage account from connection string
CloudStorageAccount storageAccount =
CloudStorageAccount.Parse(ConfigurationManager.AppSettings["StorageAccountConnectionStrin
g"]);

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

// Retrieve a reference to a container.
CloudBlobContainer container = blobClient.GetContainerReference(<containerName>);

// Retrieve a reference to a Blob.
CloudBlob blob = container.GetBlobReference(<blobName>);
content = blob.DownloadText();
```

Copy Blob



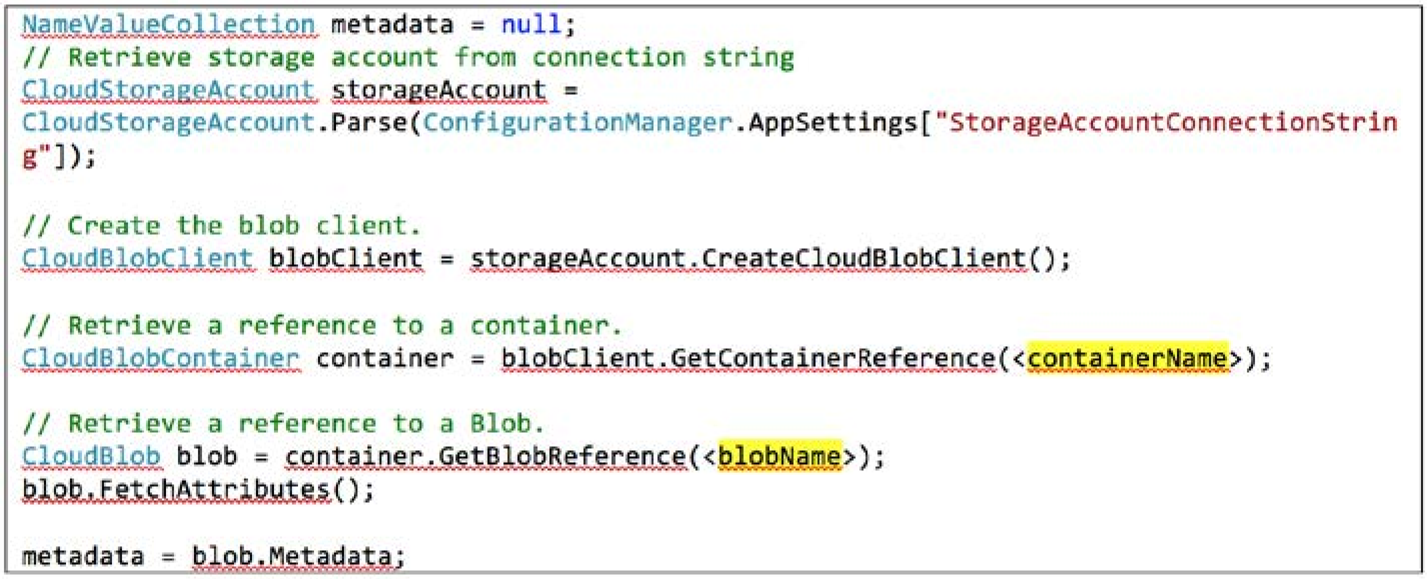
```
// Retrieve storage account from connection string
CloudStorageAccount storageAccount =
CloudStorageAccount.Parse(ConfigurationManager.AppSettings["StorageAccountConnectionStrin
g"]);

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

// Retrieve a reference to a Source container.
CloudBlobContainer sourceContainer =
BlobClient.GetContainerReference(<sourceContainerName>);

// Retrieve a reference to a Source Blob.
CloudBlob sourceBlob = sourceContainer.GetBlobReference(<sourceBlobName>);

// Retrieve a reference to a Destination container.
CloudBlobContainer destContainer = BlobClient.GetContainerReference(<destContainerName>);

// Retrieve a reference to a Source Blob.
CloudBlob destBlob = destContainer.GetBlobReference(<destBlobName>);
destBlob.CopyFromBlob(sourceBlob);
```

Delete Blob



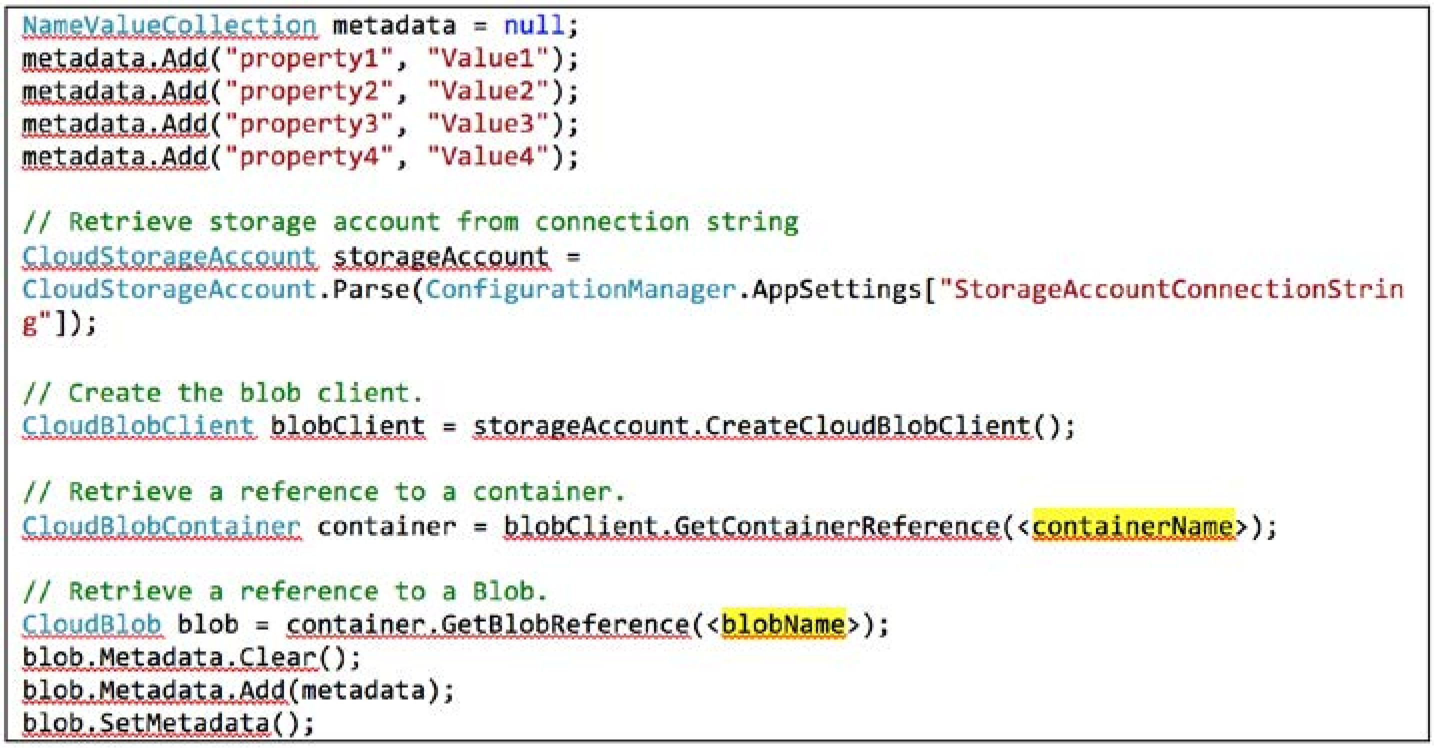
```
// Retrieve storage account from connection string
CloudStorageAccount storageAccount =
CloudStorageAccount.Parse(ConfigurationManager.AppSettings["StorageAccountConnectionStrin
g"]);

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

// Retrieve a reference to a container.
CloudBlobContainer container = blobClient.GetContainerReference(<containerName>);

// Retrieve a reference to a Blob.
CloudBlob blob = container.GetBlobReference(<blobName>);
blob.Delete();
```

Get Blob Metadata



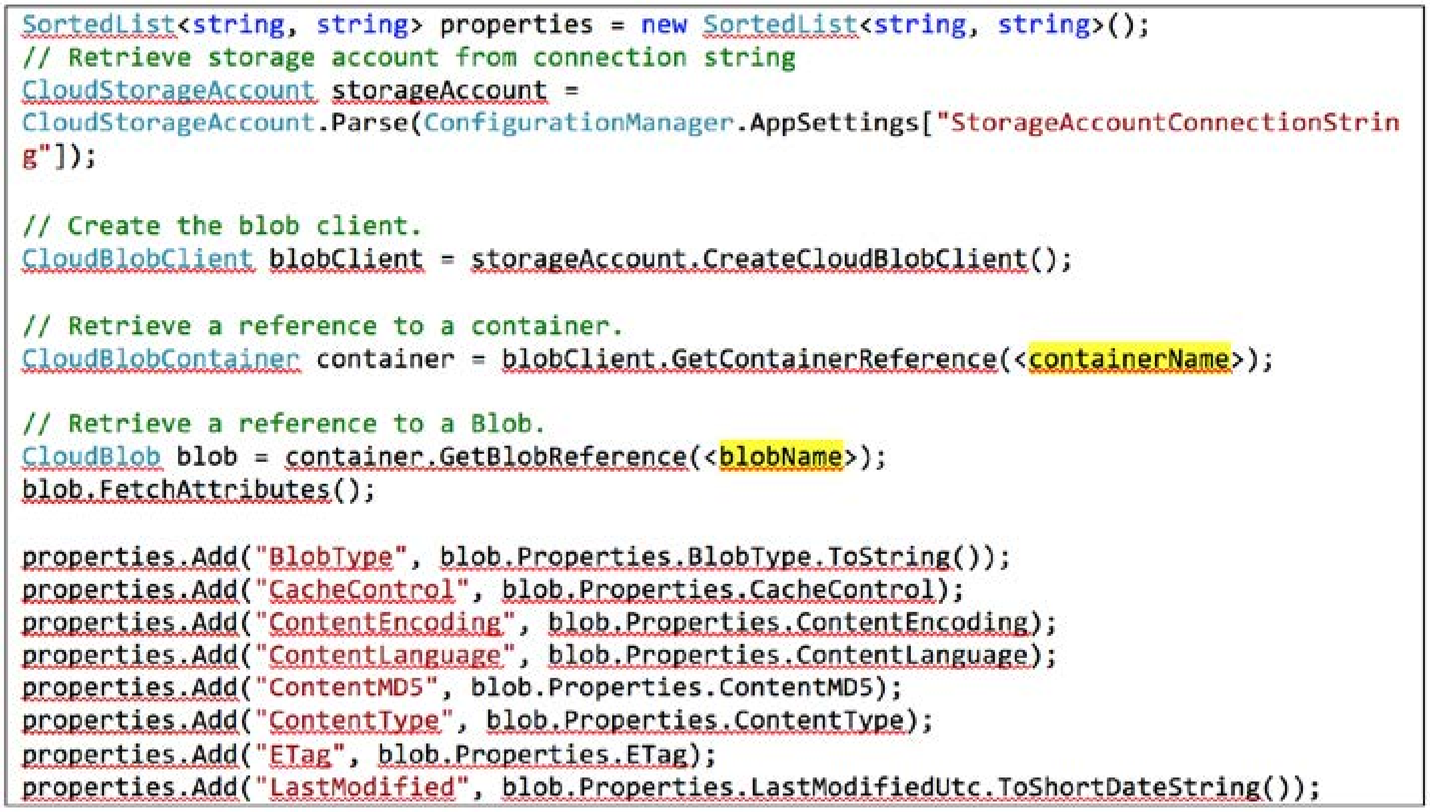
```
NameValueCollection metadata = null;
// Retrieve storage account from connection string
CloudStorageAccount storageAccount =
CloudStorageAccount.Parse(ConfigurationManager.AppSettings["StorageAccountConnectionStrin
g"]);

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

// Retrieve a reference to a container.
CloudBlobContainer container = blobClient.GetContainerReference(<containerName>);

// Retrieve a reference to a Blob.
CloudBlob blob = container.GetBlobReference(<blobName>);
blob.FetchAttributes();

metadata = blob.Metadata;
```

Set Blob Metadata



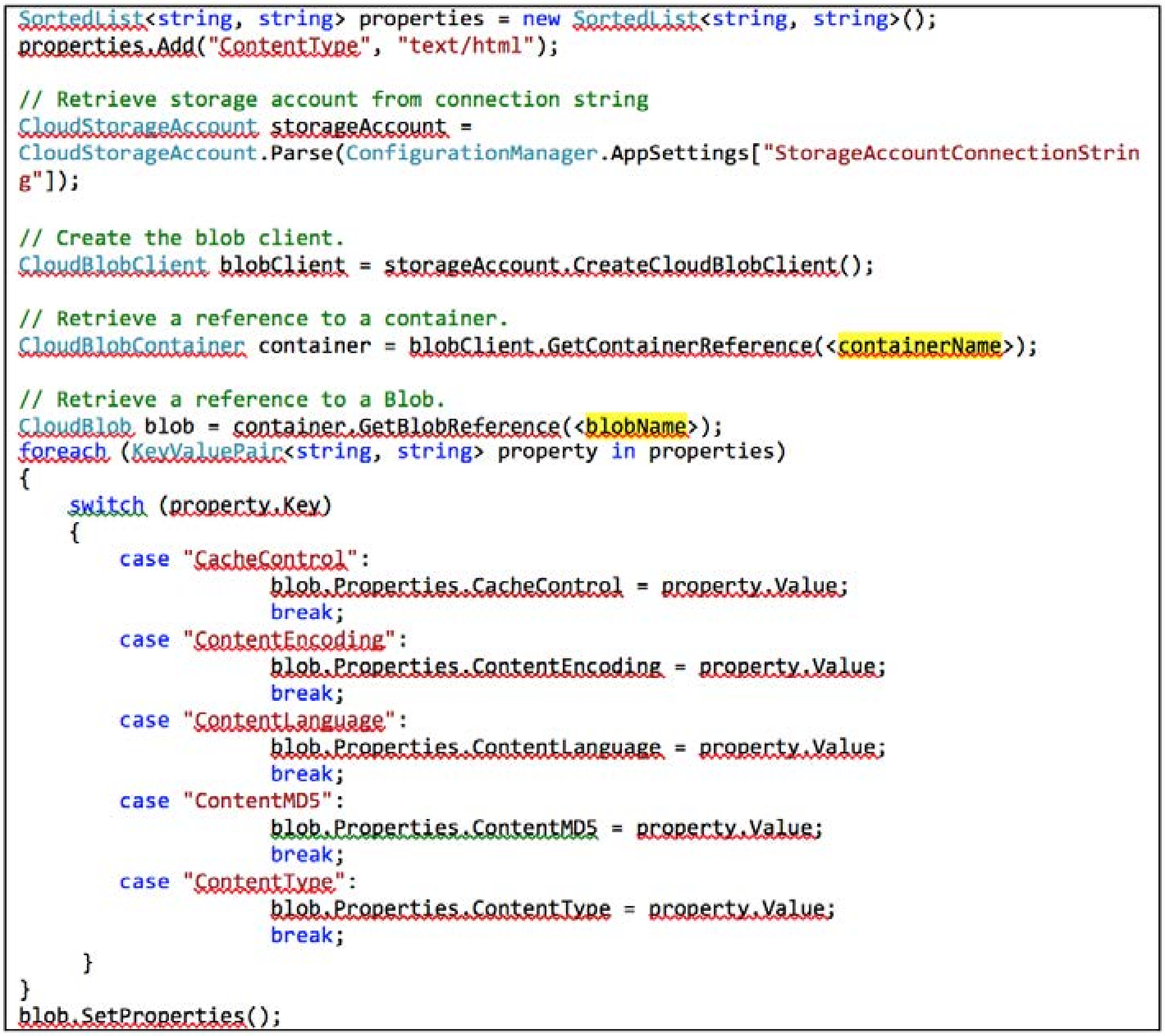
```
NameValueCollection metadata = null;
metadata.Add("property1", "Value1");
metadata.Add("property2", "Value2");
metadata.Add("property3", "Value3");
metadata.Add("property4", "Value4");

// Retrieve storage account from connection string
CloudStorageAccount storageAccount =
CloudStorageAccount.Parse(ConfigurationManager.AppSettings["StorageAccountConnectionStrin
g"]);

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

// Retrieve a reference to a container.
CloudBlobContainer container = blobClient.GetContainerReference(<containerName>);

// Retrieve a reference to a Blob.
CloudBlob blob = container.GetBlobReference(<blobName>);
blob.Metadata.Clear();
blob.Metadata.Add(metadata);
blob.SetMetadata();
```

Get Blob Properties



```
SortedList<string, string> properties = new SortedList<string, string>();
// Retrieve storage account from connection string
CloudStorageAccount storageAccount =
CloudStorageAccount.Parse(ConfigurationManager.AppSettings["StorageAccountConnectionStrin
g"]);

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

// Retrieve a reference to a container.
CloudBlobContainer container = blobClient.GetContainerReference(<containerName>);

// Retrieve a reference to a Blob.
CloudBlob blob = container.GetBlobReference(<blobName>);
blob.FetchAttributes();

properties.Add("BlobType", blob.Properties.BlobType.ToString());
properties.Add("CacheControl", blob.Properties.CacheControl);
properties.Add("ContentEncoding", blob.Properties.ContentEncoding);
properties.Add("ContentLanguage", blob.Properties.ContentLanguage);
properties.Add("ContentMD5", blob.Properties.ContentMD5);
properties.Add("ContentType", blob.Properties.ContentType);
properties.Add("ETag", blob.Properties.ETag);
properties.Add("LastModified", blob.Properties.LastModifiedUtc.ToShortDateString());
```

Set Blob Properties



```
SortedList<string, string> properties = new SortedList<string, string>();
properties.Add("ContentType", "text/html");

// Retrieve storage account from connection string
CloudStorageAccount storageAccount =
CloudStorageAccount.Parse(ConfigurationManager.AppSettings["StorageAccountConnectionStrin
g"]);

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

// Retrieve a reference to a container.
CloudBlobContainer container = blobClient.GetContainerReference(<containerName>);

// Retrieve a reference to a Blob.
CloudBlob blob = container.GetBlobReference(<blobName>);
foreach (KeyValuePair<string, string> property in properties)
{
    switch (property.Key)
    {
        case "CacheControl":
                    blob.Properties.CacheControl = property.Value;
                    break;
        case "ContentEncoding":
                    blob.Properties.ContentEncoding = property.Value;
                    break;
        case "ContentLanguage":
                    blob.Properties.ContentLanguage = property.Value;
                    break;
        case "ContentMD5":
                    blob.Properties.ContentMD5 = property.Value;
                    break;
        case "ContentType":
                    blob.Properties.ContentType = property.Value;
                    break;
     }
}
blob.SetProperties();
```

Summary

Windows Azure blob storage is a great tool for easily storing and accessing large amounts of data. With the above tutorials and code samples, you can quickly set up a storage account and begin using this helpful resource immediately.

About the Author

Tanveer Ahmed Siddiqui is a Senior Consultant Engineer at Globallogic India. With over 13 years of experience in Microsoft technologies across multiple domains, he specializes in Windows Azure Cloud.