Connector for Azure Storage Containers

By Fast Forward

Version 1.0.1
Released March 2023

Contents

Intro

This collection of scripts is designed to help with connecting to Azure Blob Storage via Shared Key Authorization. It was developed based on Microsoft's documentation, but is in no way associated with or sponsored by Microsoft. To be concise and avoid repeating Microsoft's existing documentation we will link to their documentation when relevant.

Getting Started

1. Before you can do anything with this plugin you need a Microsoft/Azure account. Learn more here: https://azure.microsoft.com/en-us/get-started/azure-portal
2. Then you will need to create an **Azure Storage Account** by following Microsoft's documentation here: https://learn.microsoft.com/en-us/azure/storage/common/storage-account-create
3. Finally you need to locate the **storage account access key** (will be referred to as "access key" from now on), by following these directions: https://learn.microsoft.com/en-us/azure/storage/common/storage-account-keys-manage

Demo Scene Guide

The demo scene contains a test script which runs through all the functionality and dumps information to the debug log. This test script isn't intended to be used in a final application. It exists to showcase the functionality and allow the user to verify they have everything set up correctly. The various calls are sequential but are written so they don't depend on each other, so even if one step fails the others will still run.

Important Note

If the scene runs correctly there is likely to be one warning log reading "List Containers failed. HTTP/1.1 403 Forbidden". This is because creating a new container causes an issue on Azure's side which makes the next call to that Storage Account fail (even if it is accessing a different container). To control for this in the test script we simply added an extra list containers call which we expect to fail. Because of this bug we don't recommend using scripts to create new containers but if it is a necessary functionality then we recommend bundling in a call you don't care about immediately after to clear out the error.

Getting the Demo Scene Working

1. First make sure you have an Azure Storage account and the access key. See Getting Started for more information.
2. Then open the DemoScene and find the "TestScript" object.
3. In the "Account Name" field put your Azure Storage Account name.

4. In the “Account Key” put your Storage Account Access Key.
5. For “Upload Test Image” you can optionally put a reference to your own Texture2D or leave it with the promo banner included.
6. For best results leave “Download Test Image” as the promo banner, as this will result in your downloaded image replacing the banner image on your screen.
7. Set “Container” to the name of a container that already exists in your storage account which will be used to read and write to.
8. Set “Download Image Blob Name” to the name (including file extension) of an image file (ideally png or jpg) which you have uploaded to the container named in step 7.
9. Set “Download Text Blob Name” to the name (including file extension) of a .txt file which you have uploaded to the container named in step 7.
10. Make sure you don’t have a container called “script-test” in your storage account (as that is the container the script will attempt to create).
11. To make the test script easier to modify the various test calls aren’t hardcoded into a specific sequence as would happen if each call had to call the next. This way you can easily comment out a call or add more. However this means on a slow connection calls could overlap, to avoid this you can modify the “delayBetweenCalls” value (the default is 3 seconds).

Running the Demo Scene

1. Once everything is set up simply press play on the scene.
2. The script will log information to the console as it goes through and with the default delayBetweenCalls value will take around 30 seconds to complete.
3. As mentioned in the note above if the scene runs correctly there is likely to be one warning log reading “List Containers failed. HTTP/1.1 403 Forbidden”. See note for more details.

Resetting the Demo Scene

1. When we download the image from the blob we store it by overwriting an existing texture in Unity’s memory. Because it isn’t written to disk the image will remain unchanged in your file system, to get the texture to reset find the file in your project view and right click to “Reimport” it. (It will also reset if you close and reopen unity.)
2. Make sure to delete the “script-test” container created by the script via the Azure Portal otherwise that step will fail as the container already exists. Also give it a minute or so for the deletion to process to avoid an error due to trying to create a container which is in the process of being deleted. (The rest of the script will still run normally even if the container creation fails.)

Scripting Guide

The primary script you will be interacting with is **AzureConnector** in the namespace **FastForward.CAS**. This script is designed as a singleton which creates itself upon first access (it will also mark that it shouldn't be destroyed on scene load) so you don't need to add it to your scene manually simply call functions on it via **AzureConnector.Instance**.

Initialization

Before using the AzureConnector you need to call AzureConnector.Instance.Init and provide it with your account name and access key. See the Getting Started section for more information. If you need to change which account you are accessing you can call Init again with new values. Note: there is currently no protection against doing that while a call is running (which could cause unexpected behavior) or support for talking to multiple Azure accounts at the same time.

Writing to Azure Storage

UploadImage

This function uploads a byte array representing an image to Azure as a JPEG.

Parameters

- byte[] data: The image represented as a jpeg encoded byte array. If your image is stored as a Texture2D you can convert to a byte array via the ImageConversion.EncodeToJPG function.
- string containerName: The name of the Azure storage container you want the image stored in.
- string fileName: The desired file name minus the file extension. Names must abide by Azure's rules for blob names.
- bool useMillis: Optional argument, defaults to false. If true the upload time in milliseconds will be appended to the filename in the format {fileName}-{milliseconds}.jpg.
- AzureUploadCallback uploadCallback: Optional a callback which will be called upon process completion. This callback will be called and given a success boolean, a string which will contain any errors if applicable, and the uri of the uploaded file if successful.

UploadText

This function uploads a string as a plain text file.

Parameters

- string text: the text which will become the contents of the text file.
- string containerName: The name of the Azure storage container you want the text file stored in.

- string fileName: The desired file name minus the file extension. Names must abide by Azure's rules for blob names.
- bool useMillis: Optional argument, defaults to false. If true the upload time in milliseconds will be appended to the filename in the format {fileName}-{milliseconds}.jpg.
- AzureUploadCallback uploadCallback: Optional a callback which will be called upon process completion. This callback will be called and given a success boolean, a string which will contain any errors if applicable, and the uri of the uploaded file if successful.

CreateContainer

Creates a new container with the provided name on the Azure Storage Account.

Warning: For some reason this causes the next call to the account to fail validation even if it is reading or writing to a different container. As a result we don't recommend using this function, but if you absolutely need this functionality we recommend calling a function you don't care about right after to clear the error.

Parameters

- string containerName: The name of the new container. The name must abide by Azure's rules regarding container names.
- AzureUploadCallback uploadCallback: Optional a callback which will be called upon process completion. This callback will be called and given a success boolean, a string which will contain any errors if applicable, and the uri of the new container if successful.

Reading from Azure Storage

ListContainers

Gets a list of all containers within the storage account which is returned via callback as an xml string.

Parameters

- AzureTextDownloadCallback callback: Required callback which will be called upon process completion. This callback will be called and given a success boolean, a string which will contain any errors if applicable, and xml as a string in the "text" argument.

ListBlobs

Gets a list of blobs in the container and returns it as an xml string via the callback.

Warning: This function does not currently handle paging so will truncate at ~5,000 items.

Parameters

- string container: The name of the container to list the contents of.

- AzureTextDownloadCallback callback: Required callback which will be called upon process completion. This callback will be called and given a success boolean, a string which will contain any errors if applicable, and xml as a string in the “text” argument.

GetBlob

Gets the specified blob from azure storage and returns it via callback as a byte array. It is up to the caller to determine how to decode the byte array. Functions like ImageConversion.LoadImage and System.Text.Encoding.UTF8.GetString may be of use.

Parameters

- string container: The name of the container which contains the file.
- string blobName: The full name of the blob including file extension.
- AzureDataDownloadCallback callback: Required callback which will be called upon process completion. This callback will be called and given a success boolean, a string which will contain any errors if applicable, and a byte array of data.

About Fast Forward

Fast Forward is an interactive software agency, creating unique user experiences with cutting-edge technology. We communicate, engage and entertain users with a consistent message across multiple channels. Experts in web development, native mobile applications, data visualization and event installations, we are always looking for new methods of storytelling and interaction. Most importantly, we love to tackle complex problems and provide simple elegant solutions to help our clients exceed their goals.

Sometimes we find new, better, and quicker ways to build things. We want to share those ways with you to make it faster and easier for you to build things, too.

Interested in having us work on a project for you? Reach out at: newbiz@fastforward.sh

Interested in working for us? Reach out at: careers@fastforward.sh

Learn more on our website at: https://fastforward.sh/