CloudImages

July, 2016.

Latest version of this guide at:
https://goo.gl/stKP4M

CloudImages forum thread:
http://goo.gl/BBiaHY

Asset Store Page:
http://u3d.as/xoU

Asset Store Publisher Page:
https://goo.gl/9Ec0Tx

CYBERIANSOFTWARE
http://CyberianSoftware.com.ar
support@cyberiansoftware.com.ar

Developer Guide

Introduction

Principles

Using *CloudImages* enables Unity3D developers to save and get images from Google Drive, one of the most simple and widely available cloud environment services: no more extra costs or over complex scenerios!

Liberate your project and team from special hosting requirements, external libraries or exotic third party services! *CloudImages* offers a simple, flexible and effective solution, while avoids introducing any hassle.

By simply incorporating a small c# class to your project, based entirely on Unity native API calls, you will have a powerful tool for saving screenshots to the cloud, getting texture updates to your game, saving player profile pictures, and whatnot.

From the Unity Editor or virtually any Unity build target(1), connect your app or game with a web service hosted on your Google Drive account, and open the field of cloud services bringing the power of Google Drive to the table!

The web service is a simple script developed using a javascript based language, making use of online Google APIs. The source code for both extremes of the connection, Unity and Google, are included, which means optimal flexibility.

(1): Exceptions are only web build targets: WebGL coming on future versions. WebPlayer, deprecated by Unity, will not be available.

Basic Demo

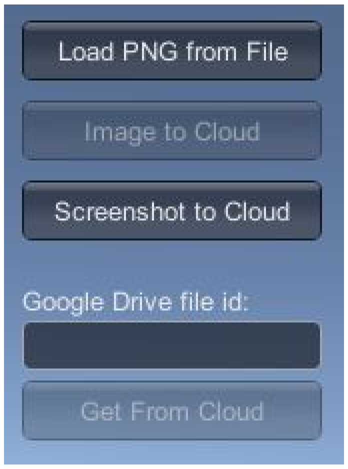
Performing launch status check

This chapter covers the use of the *CloudImages* Demo. To understand how to properly setup the asset, please read chapters entitled “Deployment”.

Using the demo in an empty Unity project is always recommended, as it also works as indicator that everything has been correctly setup up, and that any potential issues are not related to other parts of the project.

So in order to play the demo script, create a new project, a new scene, and import the asset from the Unity Asset Store. Go to the CloudImages folder, and attach the *CloudImages_Demo* script to any object in the scene. You also have to setup some fields in the *CloudImagesConnector* class as described in the Deployment chapters.

Playing the provided example will look like this :



The demo options.

The possible actions are self explanatory.

The result of this options will be described in the Unity Console output, and visible in the Texture2D to the left of the buttons.

Also, of course you will be able to verify the result of any images sent to the cloud, in your Google Drive account. Depending on the performed action, you will be able to see new images on your Google Drive, named “TextureFile” and/or “Screenshot.”

Remember that Google Drive enables multiple files of the same name.

- ❏ “**Load PNG from File**”: will load and display an image included in the Assets/CloudImages folder.
- ❏ “**Image to Cloud**”: will send any images visible on the demo, to your Drive.
- ❏ “**Screenshot to Cloud**”: will take an screenshot of your game and upload it to the Google Drive account.
- ❏ “**Get From Cloud**”: will download an image file from your Google Drive account, based on the file id specified in the input box above the button.

CloudImages will return the name and id of any file saved on the cloud. It is up to you to decide how to handle that data for further use (ie: save it to a Google Spreadsheet using ***Google Sheets For Unity***: http://u3d.as/6Lk).

Usage

Small, simple and clean API

CloudImages offers a very concise and clean API for making the cloud calls.

API Summary

public static void RequestImage (*string imgId* **)**

Description

Will retrieve an image from the Google Drive account where the webapp has been deployed. The image file is specified by Id.

Parameters

➔ "imgId": String value holding the Id of the Google Drive image file to retrieve.

public static void PersistImage (*Texture2D texture,*
string name,
bool usePNG = true **)**

Description

Will upload a texture to the cloud, creating a new file in the Google Drive account where the webapp has been deployed.

Parameters

➔ "texture": Texture2D to be persisted to the cloud.
➔ "name": Name to be used for the GoogleDrive file.
➔ "usePNG": determines the image encoding: PNG or JPG. Will default to PNG.

The JPG quality can be set on the CloudImagesConnector class corresponding field.

Deployment

Client Side

In order to setup *CloudImages* on a given Unity project, you need to follow a few very simple steps.

First, you need to include the CloudImagesConnector class into the desired project. For that purpose, simply copy `CloudImagesConnector.cs` file into the Assets folder (or any subfolder) of your Unity project

After that, you need to complete some field members of that class.

❏ **webServiceUrl** - string value indicating the url of the published webapp. See chapter Deployment, Server Side, for more details.

❏ **timeOutLimit** - float value that indicates, the number of seconds allowed for the connection to perform its complete cycle, before canceling it.

❏ **debugMode** - bool value for activating a more verbose mode on for the connection procedure. Is inactive by default, to prevent console clogging.

❏ **jpgQuality** - int value detailing the level of compression for the JPG encondig. Ranges from 1 to 100 and is set to 90 by default.

CloudImages uses UnityWebRequest class, wich received critical bugs fixes on recent patches, so make sure to always be using the latest Unity3D stable version.

Deployment

Server Side

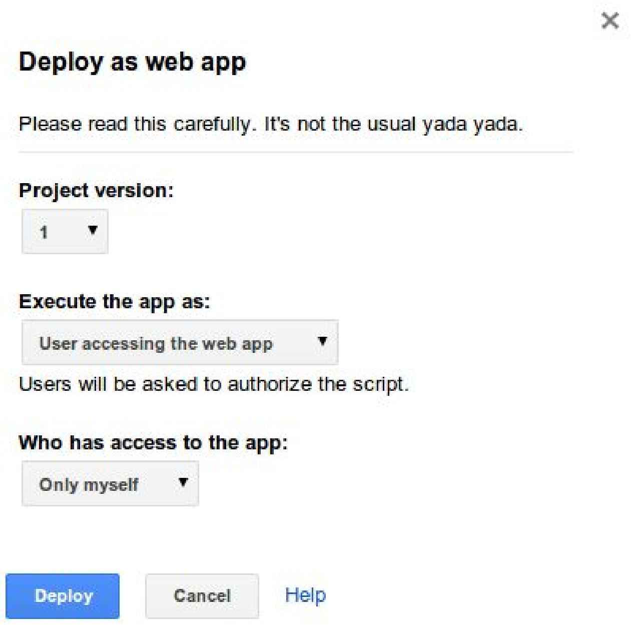
First step in setting up CloudImages, is getting a copy of the web service script. It was developed using Google Apps Script, and is conveniently stored and deployed from your Google Drive. You can grab your copy from this link:

https://goo.gl/mUlYw0

Deploying the Web Service

Second step is the script deployment as webapp. To publish the script as a webapp, follow these steps:

1. Open your script file, by double clicking on the file at Google Drive. If it's the first time, it will guide you to associate the script files to the Drive Script Editor.
2. From the Script Editor, save a new version of the script by selecting **File > Manage Versions**, then **Save New Version**.
3. Select **Publish > Deploy as web app**.
4. Under **Project version**, select the version you just saved.
5. Under **Execute the app as**, select your account (the developer's).
6. Under **Who has access to the app**, select "*Anyone, even anonymous*".
7. Click **Deploy**.



Deploy as web app

Please read this carefully. It's not the usual yada yada.

Project version: 1

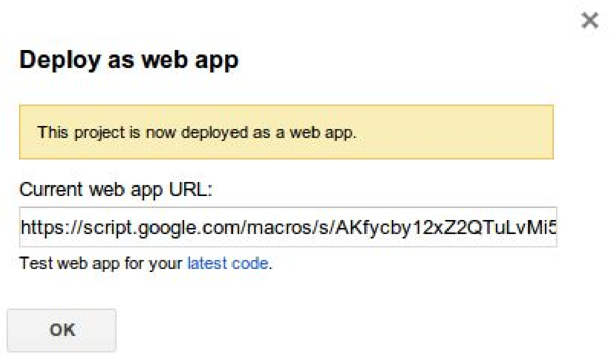
Execute the app as: User accessing the web app

Users will be asked to authorize the script.

Who has access to the app: Only myself

If you found that there is no "*Anyone, even anonymous*" option, you need to check your Google Apps Administrator panel, there is a policy option for sharing the docs outside the organization. You can read more about Google Apps sharing policy here: https://support.google.com/a/answer/60781

Once you click **Deploy**, you'll see a new dialog with a message indicating that your scriptt has been successfully deployed as a web app.


Deploy as web app

This project is now deployed as a web app.

Current web app URL:

https://script.google.com/macros/s/AKfycby12xZ2QTuLvMi5

Test web app for your latest code.

The above dialog provides the URL for your app, to be used in the **webServiceUrl** field of the CloudConnector class:

- Is labeled **Current web app URL** and is the address of the the published version of your app, based on the last version you saved and deployed.

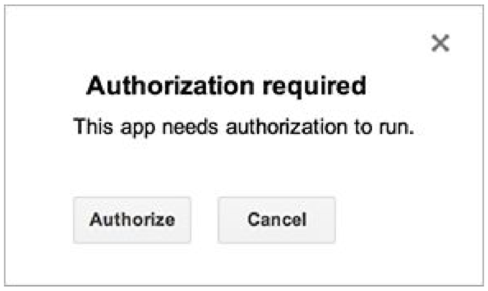
If you make changes to your script, you will have to repeat the revision and deployment process then, or your changes will not go live. For further info, see https://developers.google.com/apps-script

Granting access rights

In order for the web app to use the Google Script API, you need to grant it permissions to run. Usually, the authorization request appears at the moment of deploying your script (previous step).

If this was not the case, you can provoke the auth dialog to show up by follwing the steps described below.

To get the authorization dialog, open the "Run" menu and click on the first function, "doGet". You can **ignore/dismiss any error** produced by this one-time in-editor script run.


Authorization required

This app needs authorization to run.

From the moment you authorize it, the script will be ready to receive calls.