GoogleSheetsForUnity

August, 2016.

Latest version of this guide at:
https://goo.gl/tcvfU6

GSFU Unity3D.com forum thread:
http://goo.gl/fVvttv

Asset Store Page:
http://u3d.as/6Lk

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

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

Developer Guide

Introduction

Principles

Using Google Sheets for Unity enables Unity3D developers to work with one of the most simple and widely available of cloud environment services: Google Spreadsheets. No more extra costs or over complex scenerios!

While offering a simple, flexible and effective solution, avoids introducing any hassle. GSFU is totally unobstrusive to your development process, your game, and your insfraestructure: no need to worry about integrating third party libraries, exotic dependencies, authentications, or users credentials... Just using native Unity API calls, your project will be able to save, retrieve, update and delete objects to Google Spreadsheets at runtime!

Free your project and team from special hosting requirements, and the uncertainties introduced by third party services usually involved in gameplay balancing, localization, backend services, statistics, QA, 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 Spreadsheets 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.

Package Contents

Unboxing GSFU

Included in the Google Sheets For Unity package there are seven files. These are the components of the asset core, as well as the included examples, one for runtime games and apps, and other for in-editor use of the asset.

- **CloudConnector.cs**
  This is the file (along CloudConnectorCore) to be included in the projects using GSFU in runtime, on games or applications.

- **CloudConnectorCore.cs**
  This file is always required if you are going to use GSFU, in editor or runtime mode.

- **CloudConnectorEditor.cs**
  File (along CloudConnectorCore) to be included in the projects using GSFU in the Unity Editor.

- **GSFU_Demo_Runtime.cs**
  File required (along with CloudConnector.cs, CloudConnectorCore.cs and GSFU_Demo_Utils.cs) for the runtime example to work.

- **GSFU_Demo_Editor.cs**
  File required (along with CloudConnectorEditor.cs, CloudConnectorCore.cs and GSFU_Demo_Utils.cs) for the editor example to work.

- **GSFU_Demo_Utils.cs**
  File required by both demos included in the asset.

- **GoogleSheetsForUnityDeveloperGuide.pdf**
  This file. There is also an online version linked at the top of this document, that will always hold the latest updates.

Basic Demo

Performing launch status check

This chapter covers the use of the Google Sheets For Unity Runtime Demo. To understand how to setup Google Sheets For Unity correctly, please read chapters entitled “Deployment”.

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

This demo requires four files CloudConnector.cs, CloudConnectorCore.cs, GSFU_Demo_Runtime.cs and GSFU_Demo_Utils.cs to be in the project.

So in order to play the demo script, create a new project, a new scene, and import the asset. Go to the GSFU folder, and attach the *GSFU_Demo_Runtime* script to any object in the scene. You also have to setup some fields in the *CloudConnector* class as described in the Deployment chapters.

Playing the provided example will look like this :

The demo posible actions.

The possible actions are self explanatory, however is important that the first time they are tested, must be done in top to down order. That is: first create table, then create player, then update player, and so on.

The result of this options will be described in the Unity Console output, and visible in the working spreadsheet linked in the *CloudConnector* script.

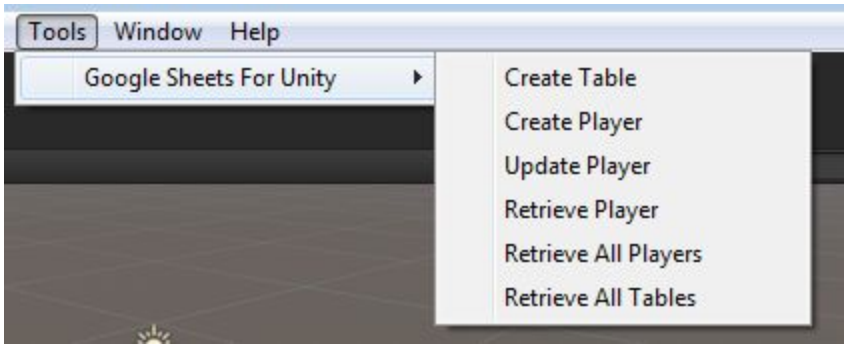
- “**Create Table**”: will create a new worksheet in the spreadsheet.
- “**Create Player**”: will create a new row in the worksheet with player data.
- “**Update Player**”: will search the worksheet for players of a given name and update a value on those found.
- “**Retrieve Player**”: will search the worksheet for players of a given name and Log the first one found to the Unity Console.
- “**Retrieve All Players**”: will get the complete players worksheet data and Log it to the Unity Console.
- “**Retrieve all Tables**”: will get the data from all worksheets on the spreadsheet and log it to the Unity console in Json format.

Editor Demo

Poking Google from Unity Editor

This chapter covers the use of the Google Sheets For Unity In-Editor Demo. To understand how to setup Google Sheets For Unity correctly, please read chapters entitled “Deployment”.

Besides working with Google Spreadsheets at runtime from a game or app, you can also utilize GSFU as a development tool from the Unity environment itself. To provide an example of this, there is an included demo that will create a new menu group as described in the image below.



In-Editor example in action.

This In-editor example of GSFU depends on four classes to be present on the project:

- CloudConnectorEditor
- CloudConnectorCore
- GSFU_Demo_Editor
- GSFU_Demo_Utils

As long as the required files be included in the project, the demo will be functionality will be available: provides the same options thn the runtime demo described in the provious chapter, but from a Unity Editor menu. The result will be thrown to the console.

You need to setup some fields in the *CloudConnectorEditor* as described in the Deployment chapters.

Usage

Small, simple and clean API

The Google Sheets For Unity principle lies on using a technology mashup to simplify the connection between Unity & Google Spreadsheets. The underlying idea is to provide a foundation where to build upon, to use the asset as a starting point.

But now, since the V.2.0 Google Sheets For Unity also offers an out of the box small API for doing basic CRUD operations from a Unity project connected to a Google Spreadsheet. For anyone not familiar with the CRUD concept, it stands for “Create, Retrieve, Update, Delete”, in relation to data handling.

API Summary

public static void CreateObject (*Dictionary<string, string> fields*, *string objTypeName*, *bool runtime* **)**

Description

Will create a new object (new row in a worksheet) with the specified field values.
Expects a dictionary of field names & values to be stored, as well as the table name.
If not all type fields available in the table are passed, those missing will be filled as "null". Fields passed but not in the table, will be ignored.

Parameters

- ➔ "fields": String dictionary of field names & values to be stored.
- ➔ "objTypeName": Name of the table that will hold the object.
- ➔ "runtime": Indicates if the request was sent from Editor or running game.

public static void CreateObject (*string jsonObject*, *string objTypeName*, *bool runtime* **)**

Description

Will create a new object (new row in a worksheet) with the specified json data.
Expects a json string with the object, as well as the table name).
If not all type fields available in the table are passed, those missing will be filled as "null". Fields passed but not in the table, will be ignored.

Parameters

- ➔ "jsonObject": Json string with the object to be persisted.
- ➔ "objTypeName": Name of the table that will hold the object.
- ➔ "runtime": Indicates if the request was sent from Editor or running game.

public static void CreateTable (*string[] headers,*
string tableTypeName,
bool runtime **)**

Description

Will create a new table (new worksheet in a spreadsheet) with the specified name, and header values.
Expects an array of field names for headers, as well as the table name to be used.

Parameters

- ➔ "headers": String array with the names of the table headers.
- ➔ "tableTypeName": Name of the table to be created.
- ➔ "runtime": Indicates if the request was sent from Editor or running game.

public static void GetObjectsByField(*string objTypeName,*
string searchFieldName,
string searchValue,
bool runtime **)**

Description

Retrieves from the spreadsheet an array of objects found by searching with the specified criteria.
Expects the table name, the name of the field to search by, and the value to search.

Parameters

- ➔ "objTypeName": Name of the table to search.
- ➔ "searchFieldName": Name of the field to search by.
- ➔ "searchValue": Value to search for.
- ➔ "runtime": Indicates if the request was sent from Editor or running game.

public static void GetTable (*string tableTypeName,*
bool runtime **)**

Description

Retrieves from the spreadsheet an array of all the objects found in the specified table.
Expects the table name.

Parameters

- ➔ “tableTypeName": The name of the table to be retrieved.
- ➔ "runtime": Indicates if the request was sent from Editor or running game.

public static void GetAllTables (*bool runtime*)

Description

Retrieves from the spreadsheet the data from all tables, in the form of one or more array of objects (each table will be an array) as a json string.

Parameters

➔ "runtime": Indicates if the request was sent from Editor or running game.

**public static void UpdateObjects (*string objTypeName,*
string searchFieldName,
string searchValue,
string fieldNameToUpdate,
string updateValue,
bool runtime)**

Description

Updates a field in one or more objects found by searching with the specified criteria. Expects the table name, the name of the field to search by, the value to search, as well as the field name and value to be updated in the matching objects.

Parameters

➔ "objTypeName": Name of the table to search.
➔ "searchFieldName": Name of the field to search by.
➔ "searchValue": Value to search for.
➔ "fieldNameToUpdate": Name of the field to update.
➔ "updateValue": New value to be set.
➔ "runtime": Indicates if the request was sent from Editor or running game.

**public static void DeleteObjects (*string objTypeName,*
string searchFieldName,
string searchValue)**

Description

Deletes one or more objects (rows) matching the specified criteria, from a specified table (worksheet).

Parameters

➔ "objTypeName": Name of the table to search.
➔ "searchFieldName": Name of the field to search by.
➔ "searchValue": Value to search for.
➔ "runtime": Indicates if the request was sent from Editor or running game.

Deployment

Client Side

In order to setup Google Sheets For Unity on a given Unity project, you need to follow a few very simple steps.

First, you need to include the CloudConnector (or CloudConnectorEditor if want the in-editor features) and CloudConnectorCore classes into the desired project. For that purpose, simply copy `CloudConnector.cs` (and/or `CloudConnectorEditor.cs`) and `CloudConnectorCore.cs` files into the Assets folder (or any subfolder) of your Unity project

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

In CloudConnector / CloudConnectorEditor:

- **webServiceUrl** - string value indicating the url of the published webapp. See chapter Deployment, Server Side, for more details.
- **spreadsheetId** - string value to be set to the ID key of the spreadsheet to be used.

> To get the spreadsheet id, simply open the spreadsheet and watch at the URL. It will look like the example below, in which the key is "*abc123456*".
>
> *https://docs.google.com/spreadsheets/d/**abc123456**/edit#gid=671966417*

- **servicePassword** - string value with the password established in the web service script.
- **timeOutLimit** - float value that indicates, the number of seconds allowed for the connection to perform its complete cycle, before canceling it.
- **usePOST** - bool value, true by default, indicating the type of request to be created (POST or GET). Can be also set from the Editor inspector.

In CloudConnectorCore:

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

> Google Sheets For Unity uses UnityWebRequest class, wich received some important bugs fixes on Unity 5.3.4p5 so that is the minium required version of Unity 5.

Deployment
Server Side

First step in setting up Google Sheets For Unity, 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:

> http://goo.gl/ooYNL4
> *(Clicking on the link should automatically create a copy of the script).*

Second step is setting up your webservice password (is set to "*passcode*" by default).

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.

In the top section of the script, last of the global vars, you will found this line:

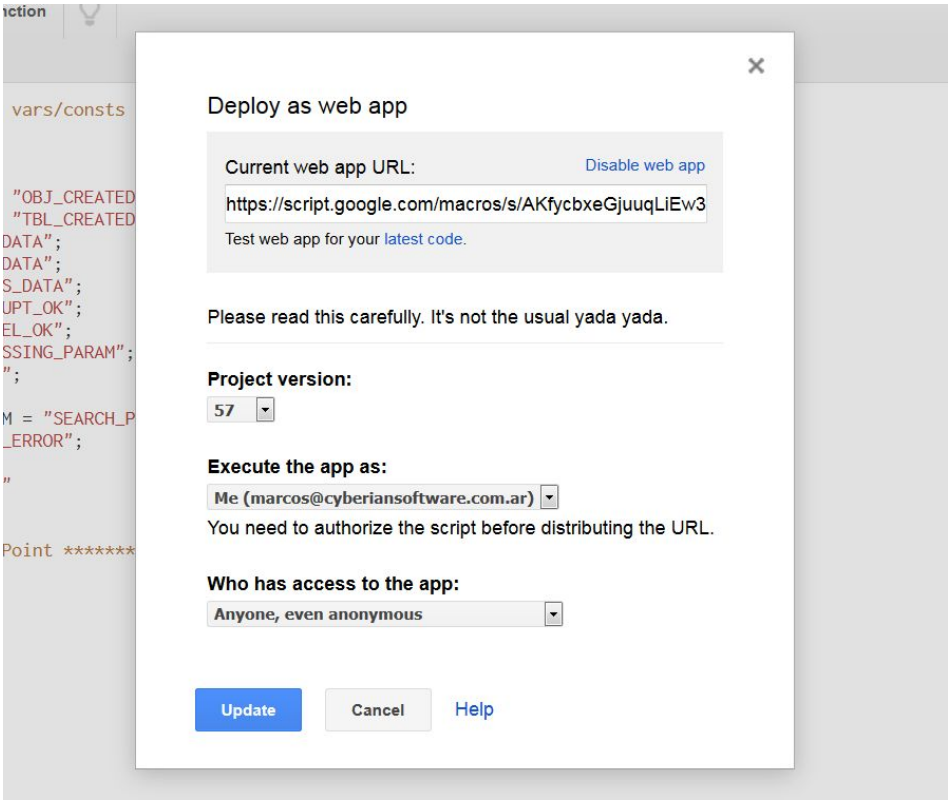
```
var PASSWORD = "passcode";
```

Change "*passcode*" to whichever password you want. Rememer to also change the ***servicePassword*** field in the CloudConnector and/or the CloudConnectorEditor scripts in Unity to the same value used here.

Deploying the Web Service

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

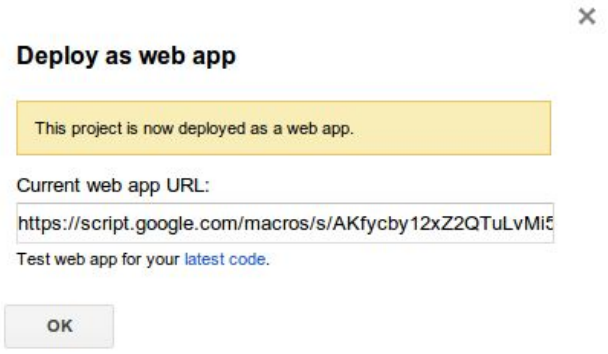
1. If you haven't, open your script file, by double clicking on the file at Google Drive. If it's the first time, it will guide you to associate the script files to the Drive Script Editor.
2. From the Script Editor, save a new version of the script by selecting **File > Manage Versions**, then **Save 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** (In further deploys, the button will say "Update") .



Deploy Dialog.

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.



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.

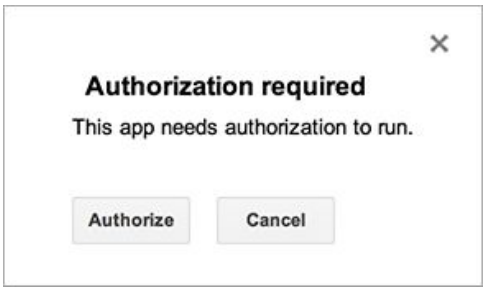
If you make changes to your script, you will have to repeat the revision and deployment process then, or your changes will not go live. For further info, see https://developers.google.com/apps-script

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 function, "doPost". You can **ignore/dismiss any error** produced by this one-time in-editor script run.



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