Google Sheets To Unity

V 1.0

Intro

Google sheets to unity is a 2 way system for importing and exporting data from google sheets to unity in both the editor and runtime (providing an internet connection is established).It can be used as to add a more data driven workflow to your unity project when updating and storing all information on a spreadsheet and then importing this information into Unity.

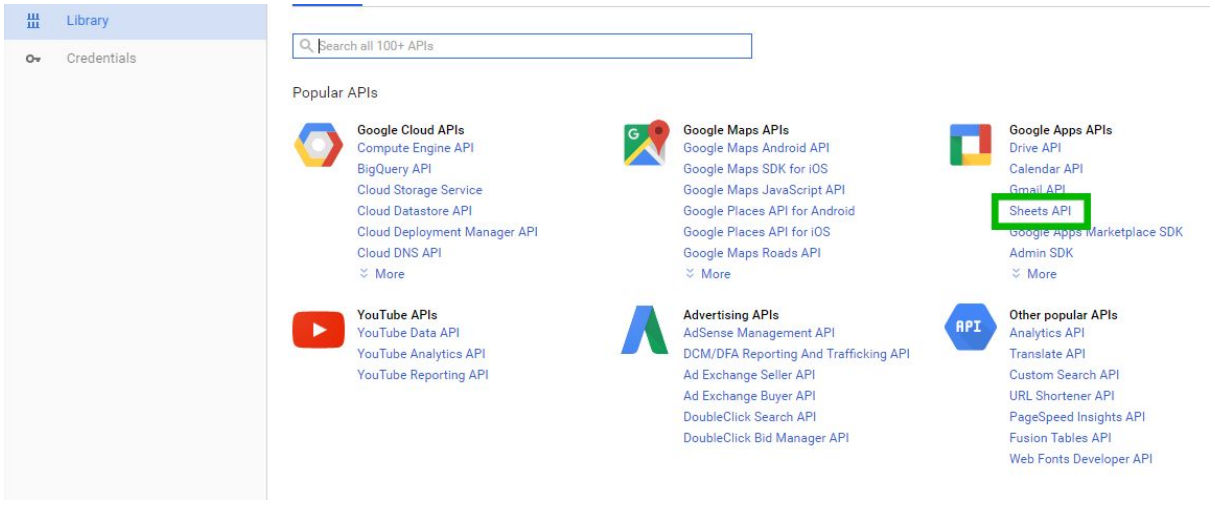
Initial Setup

Private Sheets(Read and Write)

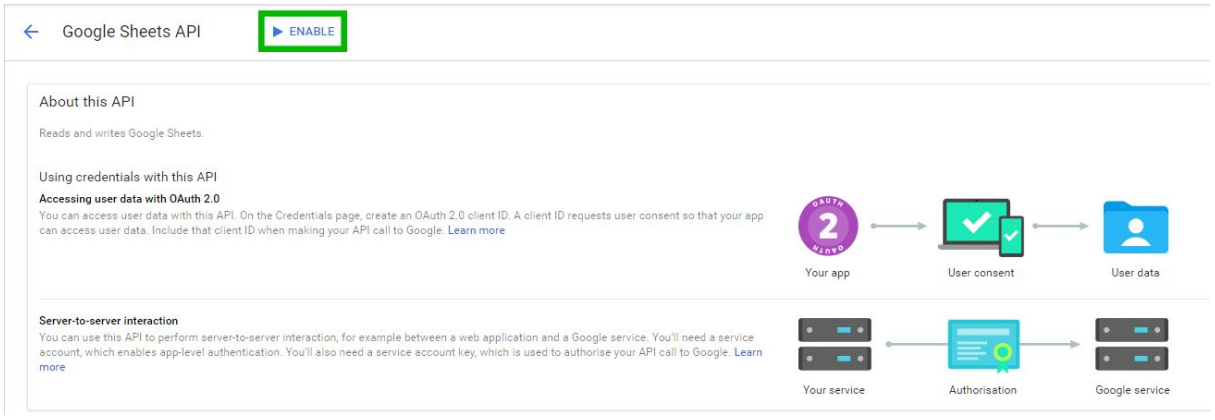
After adding Google Sheets to Unity into unity there are a few steps that need to be set up in order to link your unity project with google.

1. Login or create an account with google developer console (Console.developers.google.com) and create a new project. This can take a few minutes to create the project.

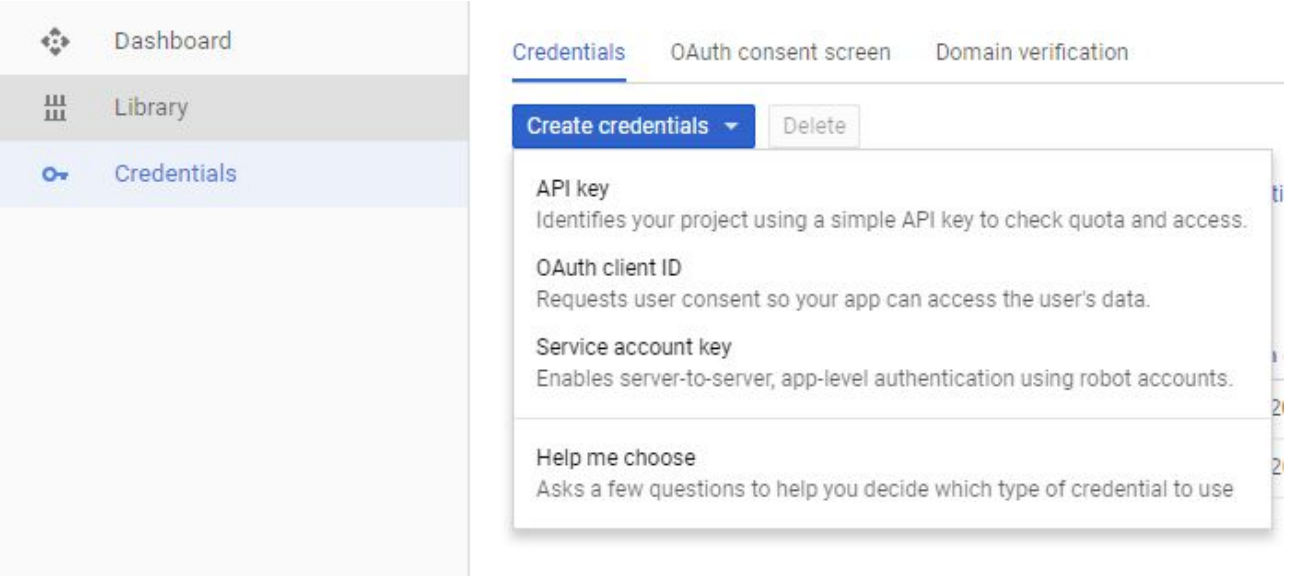
2. Once the project is created click on the “Library” button on the left hand side and either find the “Google Sheets API” link in the “Google Apps APIs” section, or use the search box to location quickly.



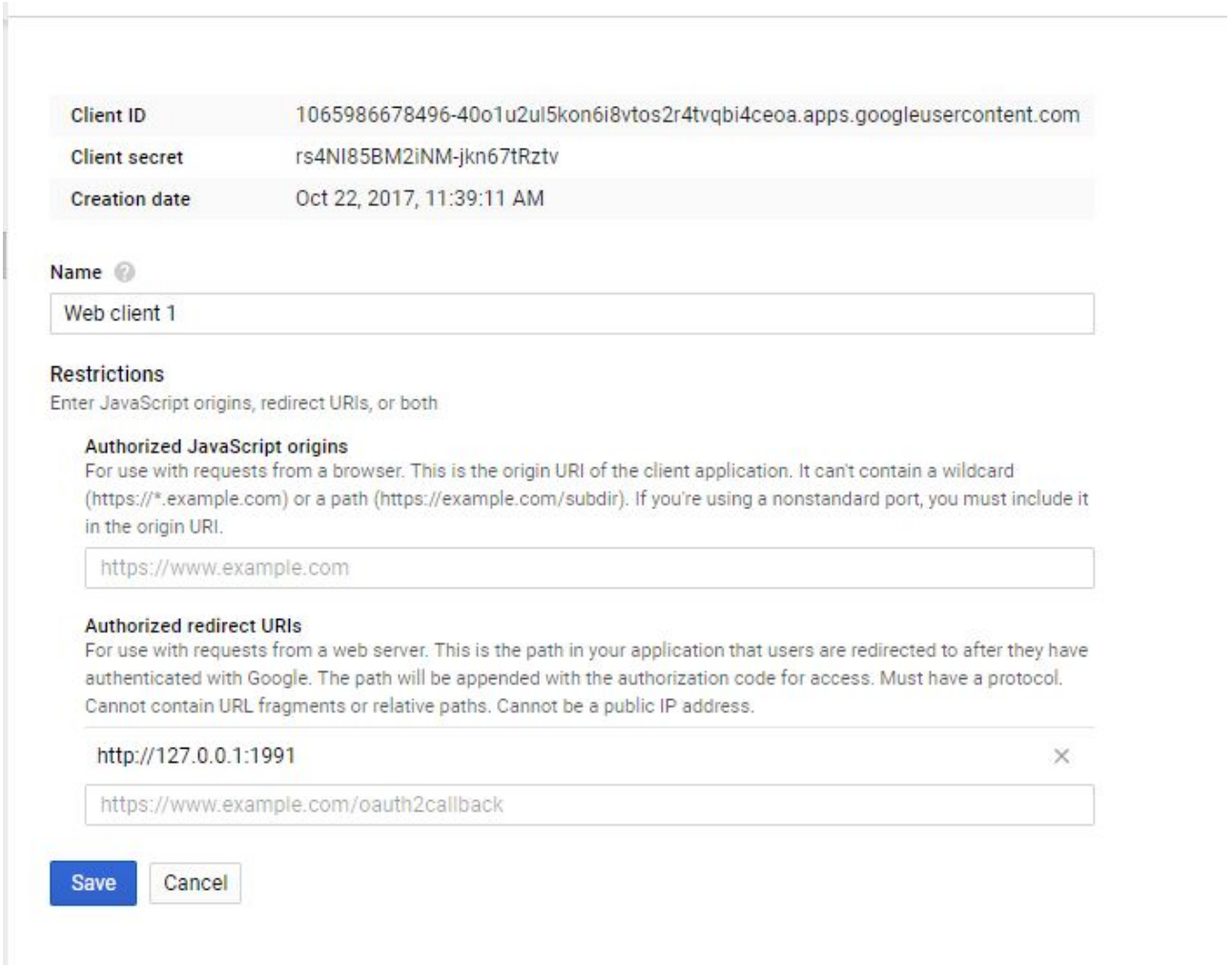
3. On the google Sheets API page click “ENABLE”.



4. Next Navigate to the Credential tab and click “Create Credentials” and select “OAuth client ID” from the dropdown

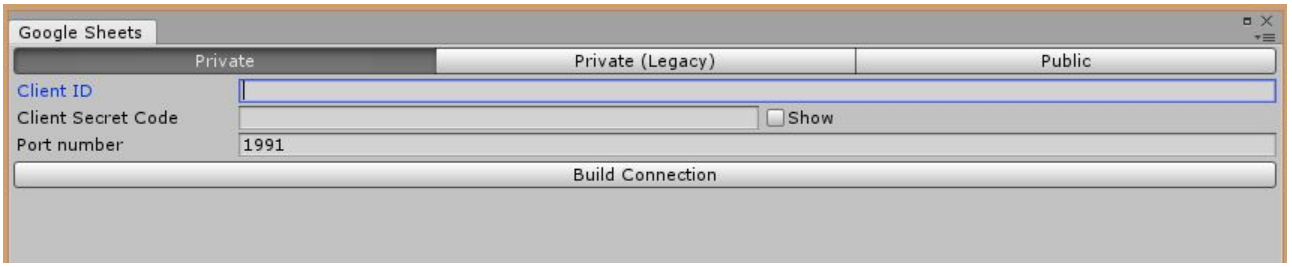


5. Next you need to name your application and in the authorized redirect URI’s enter “http://127.0.0.1:XXXX” replacting the “XXXX” with a 4 random digits (this will be the port number used so make sure its one that is open).



6. Download the JSON and place it in your streamingassets folder of the project

7. Open the Json and copy the “Client_id” and “client_Secret” into the fields on the spreadsheet config screen and enter the 4 digits set in the above step into the port number

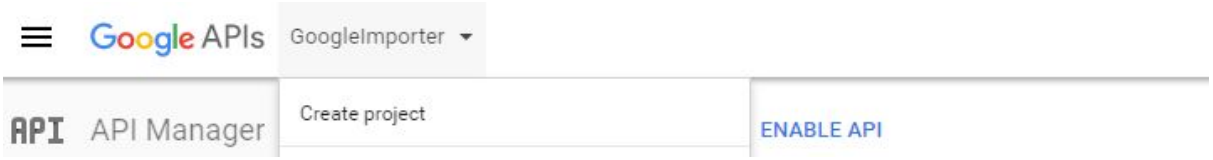


8. Click “Build Connection” and a browser will open asking you to log into google (log into the account that the sheet you want to access is available on).
9. After login in and following the steps on google you will receive a screen that says “Google Sheets and Unity are now linked, you may close this window”.
10. Google sheets and unity are now linked.
11.

12.

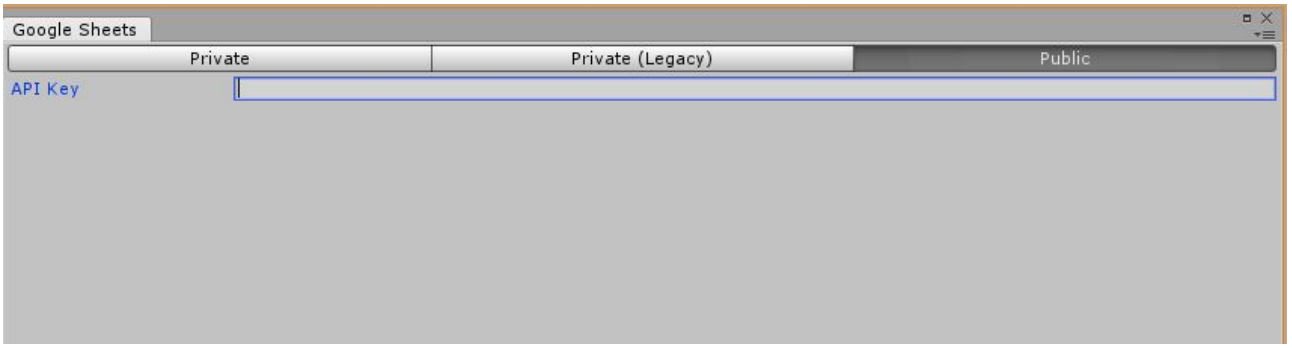
Public Sheets (Read only)

1. After setting you sheet to be public you need to visit the Google Developer Console(Console.developers.google.com) and create a new project. This can take a few minutes to create the project.

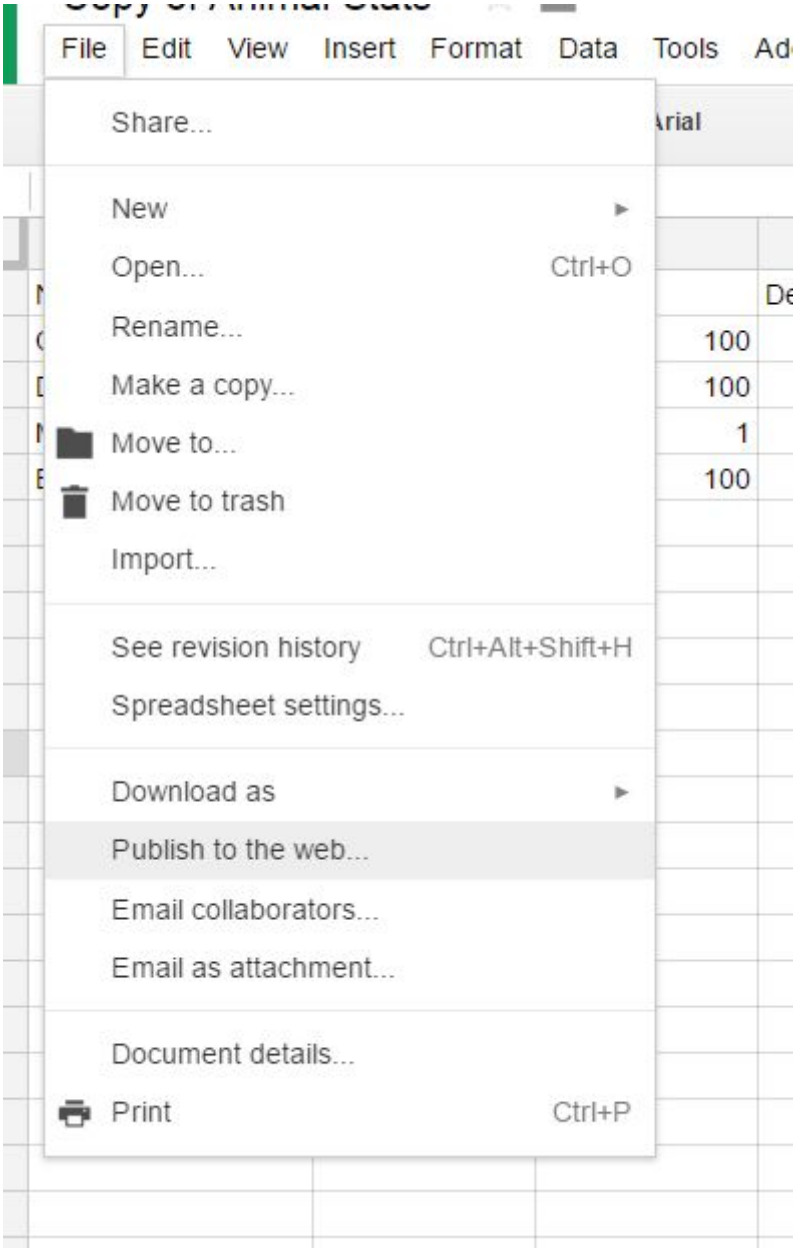


2. Click on the Credentials button on the left hand side and click Create credentials. From the dropdown list select API key.

3. Copy the new generated API key into the api key text box on the google sheets to unity config menu.



4. You need to enable your spreadsheet to be published. This is done by clicking File > Publish to the Web and then clicking “Publish” in the opening popup. By Doing this the sheet can be readable without authenticating yourself but can only be used to read data.

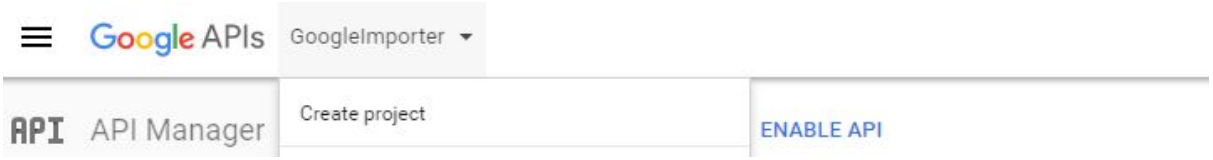


Each Spreadsheet has a unique ID that can be found in the web url for that sheet as shown below. This is the spreadsheet id needed to import the data.

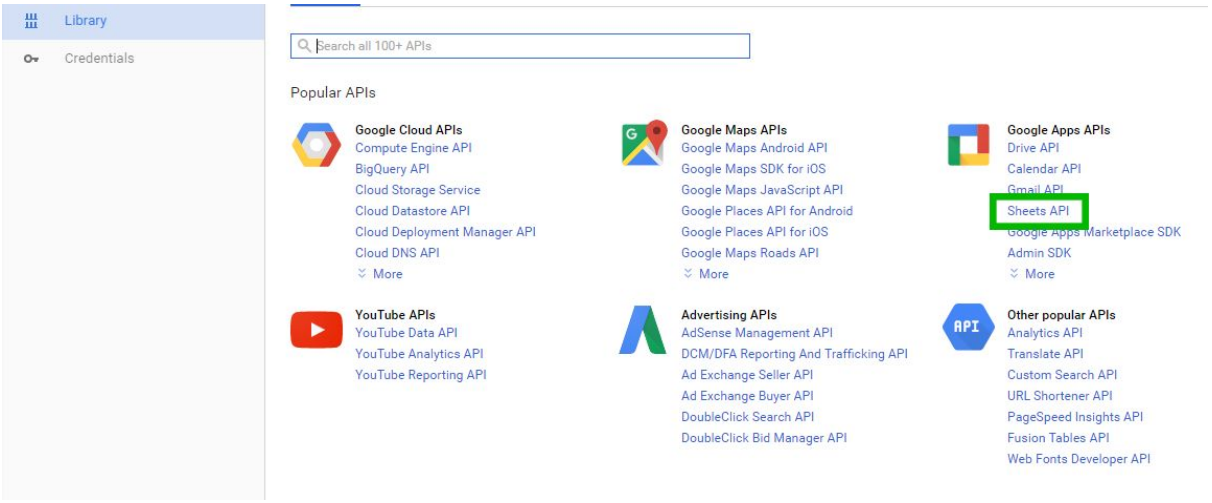
Legacy Private Sheets(Read and Write)

After adding Google Sheets to Unity into unity there are a few steps that need to be set up in order to link your unity project with google.

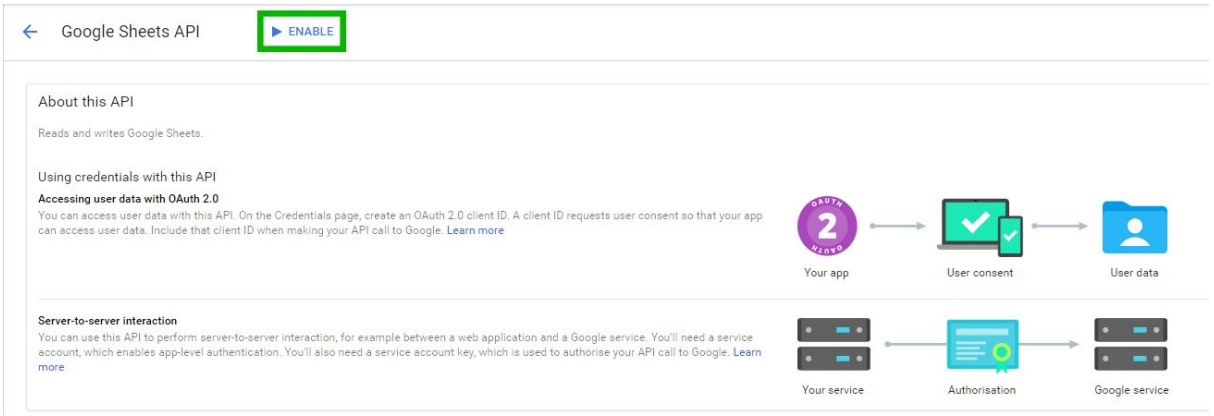
13. Login or create an account with google developer console (Console.developers.google.com) and create a new project. This can take a few minutes to create the project.



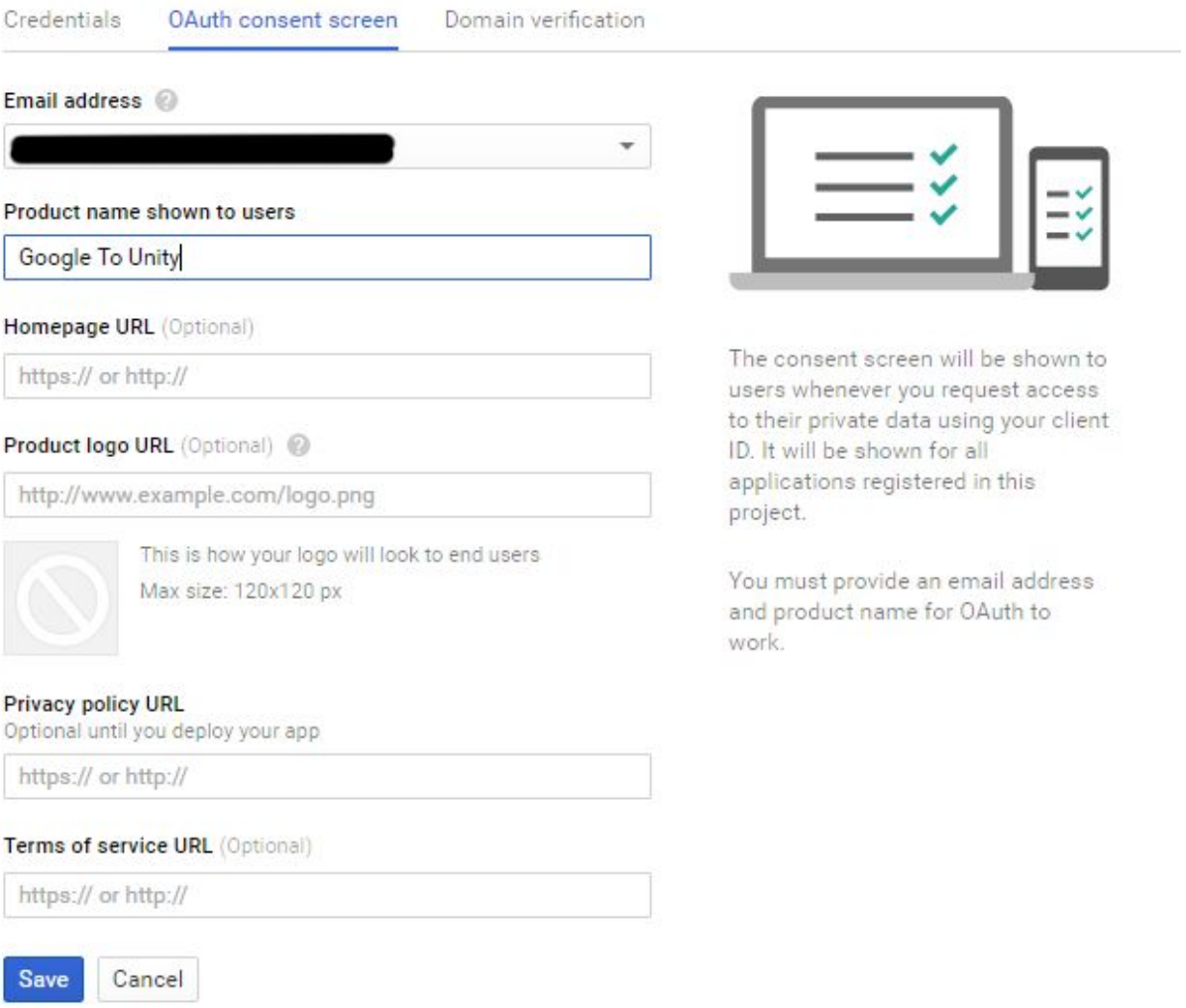
14. Once the project is created click on the “Library” button on the left hand side and either find the “Google Sheets API” link in the “Google Apps APIs” section, or use the search box to location quickly.



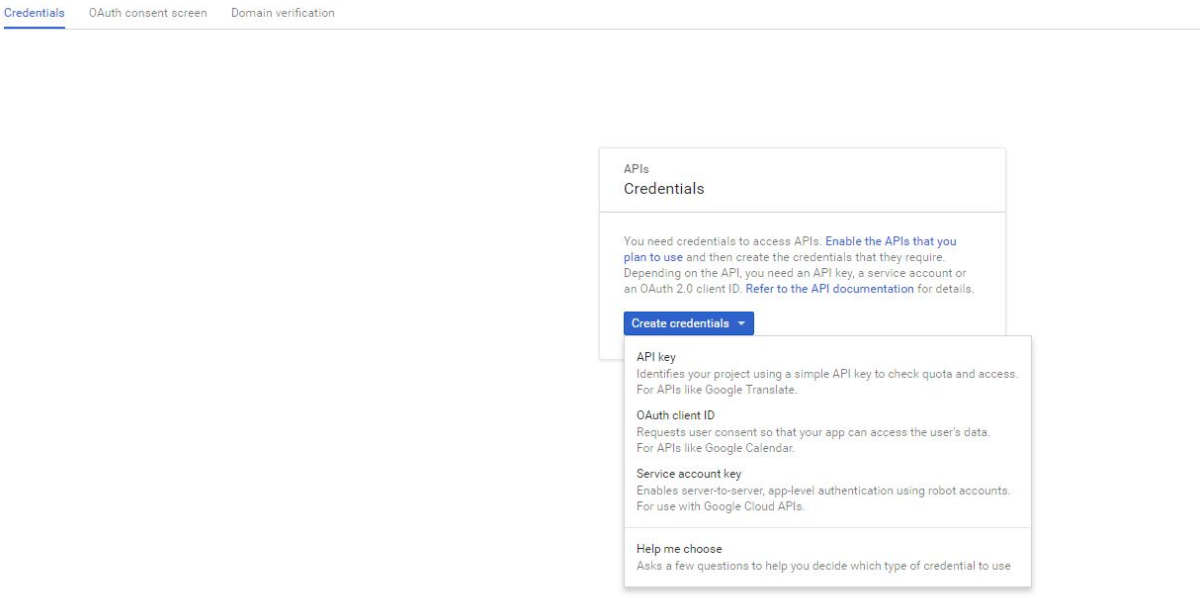
15. On the google Sheets API page click “ENABLE”.



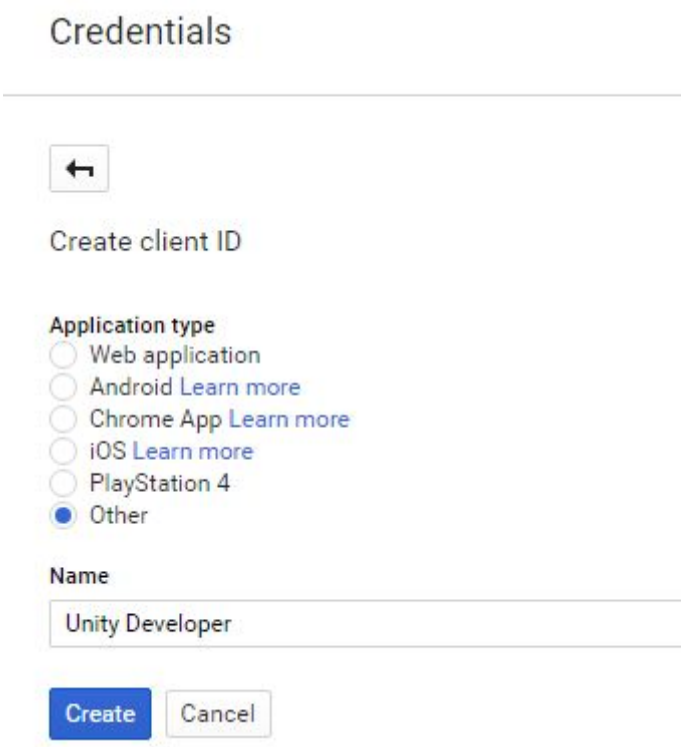
16. Navigate to the “Credentials” page on the left hand side and then to the “OAuth consent Screen” and select an email address for your product and enter a product name and click save.



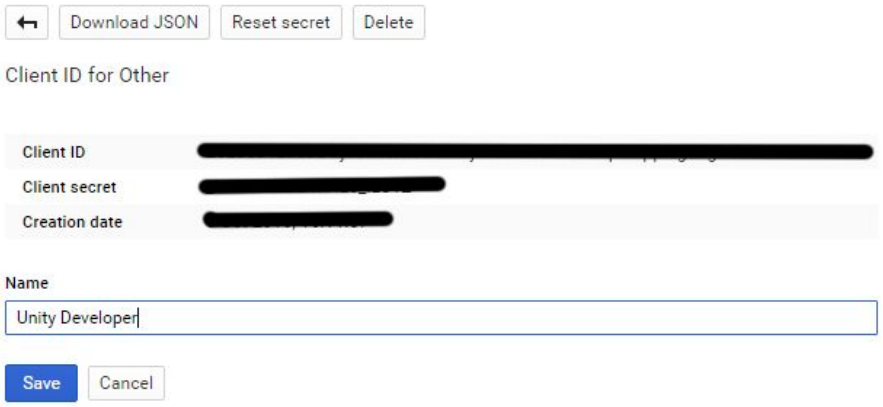
17. Still on the “Credentials” page now navigate to the “Credentials” tab, click on the “Create credentials” dropdown followed by selecting “OAuth client ID”



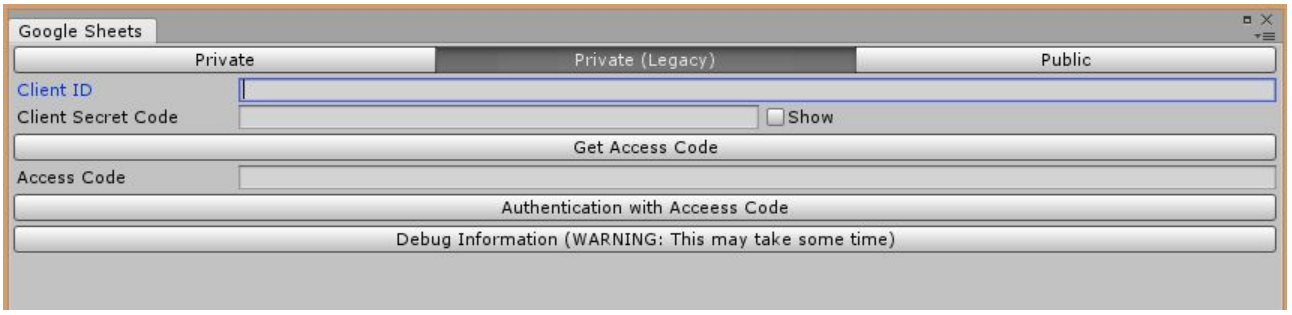
18. Select “Other” from the list and end “Unity Developer” in the name field that appears and press create



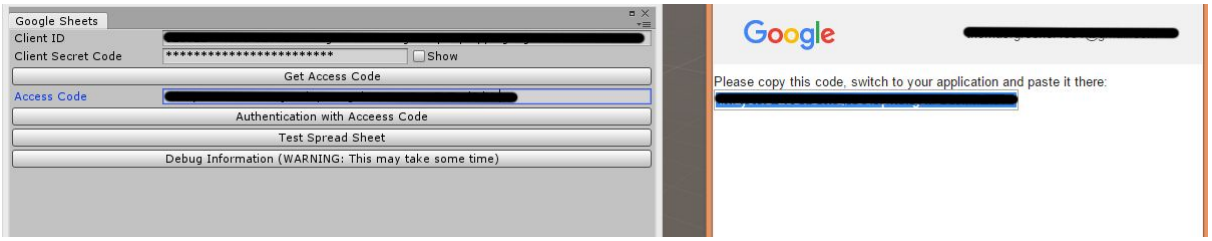
19. Your client ID and client Secret will now appear. (You can click on the created credential to get these in the future if needed)



20. In Unity open the Google Sheets to Unity window (Window > Google Sheets To Unity)



21. Copy the Client ID and the Client Secret to the relevant fields and click get access code.
22. A web page will now open where you will be asked to log into your Google account that the spreadsheet is available on (note documents that are shared with your account will still show). Follow the steps on the webpage and this will present you with an access Code, copy this code into the unity project “Access Code” field and click “Authentication with Access Code”. Unity will log out Validation successful if validation was a success.



23. If you wish to view all sheets and worksheets that your login has provided you access to click the “Debug Information” button. This will take a few minutes but will display all sheets on your google account has access to (the more sheets the longer this will take)
24. You are now set up with Google Sheets to Unity. Refer to the usage section for code references.

Example Projet (Animal Stats)

Intro

GS2U comes with a quick example project to demonstrate import and exporting data from a spreadsheet. This example will show you how to update assets with values from a spreadsheet and how to send new data to the spreadsheet.

NOTE: Prior to running the example scene you must have Linked Google sheets to your Unity as described in the initial setup section.

After you have your project linked with Google open the “Animal Stats Google Sheets Link” and select File > Make A Copy and rename the copy to “Animal Sheets”. This will create you a copy of the google sheet used in the example and allow you to access and modify it.

All Example code is available under the “Animal.cs” script

Example 1 - Updating Stats

Navigate you the example files (Default : Assets > Google Sheets To Unity > Animal Example) and open up the “Animal Stats” folder In here are 3 created animal scriptable objects with some basic information for health, attack and defence. By clicking on one of these items you can see all the information that the animal holds. Each of these animals have a custom inspector with various functions attached to them. By clicking “pull data Method one/Two” this will bring the data from the spreadsheet and match the information to the animal name. In The example the Badger holds 3 items, to handle merged cell data click the “Pull Data With merged Cells” this will collect the values of all items for the badger that in the example has 3 rows merged together.

If you look at the AnimalManager.cs script in this scene it shows an example of how the information can be updated at runtime or in a final build. All functions have a callback parameter that will be called upon the request being completed. The example will update all the animals with the latest stats that are entered into the spreadsheet when pressing play from either a public or a private spreadsheet depending on the dropdown toggle.

```
SpreadsheetManager.Read(new GSTU_Search(SheetId,worksheetName), callback);

internal void Callback(GstuSpreadSheet ss)
{
     items.Clear();
     health = int.Parse(ss[name, "Health"].value);
     attack = int.Parse(ss[name, "Attack"].value);
     defence = int.Parse(ss[name, "Defence"].value);
     items.Add(ss[name, "Items"].value.ToString());
}
```

Example 2 - Updating the data on Google Sheets (private only)

Cells are able to update themselves on google sheets and a demo of this is shown by using either the “Update Sheet information” or “update health only” buttons on the animal. These take the values that are entered into the fields and updates the animals information on google sheets. In the case of just updating health the example below will update just the one cell.

```
private void UpdateAnimalHealth()
    {
        SpreadsheetManager.Read(new GSTU_Search(animal.associatedSheet,
animal.associatedWorksheet), UpdateAnimalHealth);
    }
    private void UpdateAnimalHealth(GstuSpreadSheet ss)
    {
        ss[animal.name, "Name"].UpdateCellValue(animal.associatedSheet,
animal.associatedWorksheet, animal.health.ToString());
    }
```

If you wish to update more than one cell at once a batch update can be performed

```
    void UpdateAnimalInformationOnSheet()
    {
        SpreadsheetManager.Read(new GSTU_Search(animal.associatedSheet,
animal.associatedWorksheet), UpdateAnimalInformation);
    }
    private void UpdateAnimalInformation(GstuSpreadSheet ss)
    {
        BatchRequestBody updateRequest = new BatchRequestBody();
        updateRequest.Add(ss[animal.name, "Health"].AddCellToBatchUpdate(animal.associatedSheet,
animal.associatedWorksheet, animal.health.ToString()));
        updateRequest.Add(ss[animal.name, "Defence"].AddCellToBatchUpdate(animal.associatedSheet,
animal.associatedWorksheet, animal.health.ToString()));
        updateRequest.Add(ss[animal.name, "Attack"].AddCellToBatchUpdate(animal.associatedSheet,
animal.associatedWorksheet, animal.health.ToString()));
        updateRequest.Send(animal.associatedSheet, animal.associatedWorksheet, null);
}
```

It is recommended to cache the sheet information when first read and then to update this information as needed. By Doing this you can reference the sheet without having to load the sheet again as in the above example.

Example 3 - Sending Data To Google Sheets (private only)

GS2U can also add and append new information into a sheet. To append data use the below example. This creates a new row of data and adds it to the next available space in the sheet.

```
        List<string> list = new List<string>() {
        animal.name,
        animal.health.ToString(),
        animal.attack.ToString(),
        animal.defence.ToString()
        };
        SpreadsheetManager.Append(new GSTU_Search(animal.associatedSheet,
animal.associatedWorksheet), new ValueRange(list), null);
```

If you wish you append multiple rows the use the following, this will append multiple lines one after another.

```
 List<string> list1 = new List<string>();
 List<string> list2 = new List<string>();
//Add data to the 2 lists here
        ValueRange values = new ValueRange();
        values.Add(list1);
        values.Add(list2);


        SpreadsheetManager.Append(new GSTU_Search(animal.associatedSheet,
animal.associatedWorksheet),values , null);
```

Note: the process adds the values to the cells in a row, if you require a blank cell if the row then you need to add an empty string to the list. IE if i did not want to update the attack but i wanted to update the rest i would use

```
        List<string> list = new List<string>() {
        animal.name,
        animal.health.ToString(),
        “”,
        animal.defence.ToString()
        };
```

GS2U can also add to a set cell reference or range using Write instead of Append. In the below example this will add all the data in starting at cell “G10”

```
       SpreadsheetManager.Write(new GSTU_Search(animal.associatedSheet,
animal.associatedWorksheet, "G10"), new ValueRange(list), null);
```

Useage

Both private and public spreadsheets work in the same way to read and write to google sheets. The only limitation is that public sheets are only able to read. All functions have a callback parameter as part of their function with will be called upon the request being completed.

Reading

To Read a spreadsheet use either SpreadsheetManager.Read or SpreadsheetManager.ReadPublicSpreadsheet for private and public sheets respectively. Both these functions work in the same way and require a GSTU_Search and a callback. The callback returns a GstuSpreadsheet with all the read data that can then be searched through.

```
SpreadsheetManager.Read(new GSTU_Search(associatedSheet, associatedWorksheet), UpdateAllAnimals);
SpreadsheetManager.ReadPublicSpreadsheet(new GSTU_Search(associatedSheet, associatedWorksheet),
UpdateAllAnimals);

void UpdateAllAnimals(GstuSpreadSheet spreadsheetRef)
{
}
```

Once the data has been loaded there are several ways to access cell data. The animal example sheet looks like below which i will use as a example for viewing the references

	A	B	C	D	E
1	Name	Health	Attack	Defence	Items
2	Cat	100	150	200	A
3	Dog	10	50	10	B
4	Mouse	10	10	10	C
5	Bear	150	100	50	D
6	Fly	10	20	30	E
7					F
8					G
9	Badger	150	50	10	H

At the simplest term you are able to access through a direct cell reference using the following.

```
spreadsheetRef["A2"].value //This will return the value of the cell "A2" in this case “Cat”
```

If the search parameters for title row and column have been set up you can access directly though cross referencing the row and column ID’s

```
spreadsheetRef["Badger", "Health"].value //This will return the value of the Badgers health
foreach (var value in spreadsheetRef["Badger", "Items", true])
{
  Debug.log(value.value.ToString()); //Debug out all the badgers items
}
```

You can also get how row or column data by using either

```
spreadsheetRef.columns["A"] //returns a list of all cells in column A
spreadsheetRef.rows[1] //Returns a list of all cells in row 1
```

And if the information for title rows has been set up then you can use

```
spreadsheetRef.columns["Health"] //returns a list of all cells in the health column
spreadsheetRef.rows[Dog] //Returns a list of all cells in the dogs row
```

Write and Updating

To update row information you can use spreadsheet manager.write. This function requires a GSTU_Search, valueRange and can have a callback for once the data is written. This method will write the information starting at the startCell provided in the search and continue. This method is destructive and will override any cells with data present in them with the new values provided by the valueRange. Value range is a List<List<string>> which translates to Row<Column<Value>> from the start cell. IE if the start cell was “B4” then the first entry in the valueRange will be entered in “B4”. Below is an example

```
        List<string> list1 = new List<string>()
        {
            "A",
            "",
            "C"
        };
        List<string> list2 = new List<string>()
        {
            "",
            "2",
            "3"
        };
        List<List<string>> combined = new List<List<string>>
        {
            list1,
            list2,
        };
        SpreadsheetManager.Write(new GSTU_Search(animal.associatedSheet,
animal.associatedWorksheet, "A2"), new ValueRange(combined), null);
```

This will lead to the below output

	A	B	C
1			
2	A		C
3		2	3

A less destructive way to update cell values is to call the reference on the cell directly after the sheet has been read and use the “UpdateCellValue” method passing the sheetId and worksheet name into the function.

```
spreadSheetRef[name, "Health"].UpdateCellValue(animal.associatedSheet,
animal.associatedWorksheet,health.ToString());
```

If there are multiple cells that need to be updated at once then this can all be done through 1 call instead of through several. To do this create a BatchBodyRequest and add all the updates to this before calling Send to finalise the request.

```
        BatchRequestBody updateRequest = new BatchRequestBody();
        updateRequest.Add(ss[animal.name, "Health"].AddCellToBatchUpdate(animal.associatedSheet,
animal.associatedWorksheet, animal.health.ToString()));
        updateRequest.Add(ss[animal.name, "Defence"].AddCellToBatchUpdate(animal.associatedSheet,
animal.associatedWorksheet, animal.health.ToString()));
        updateRequest.Add(ss[animal.name, "Attack"].AddCellToBatchUpdate(animal.associatedSheet,
animal.associatedWorksheet, animal.health.ToString()));
        updateRequest.Send(animal.associatedSheet, animal.associatedWorksheet, null);
```

Appending

To append onto a sheet use the Append function. This will add the information to the next available space in the sheet starting at the startCell value in the search parameters. This will not override any data that is currently in the sheet and will only insert if there is enough space for all the information. As with the above writing example, multiple rows can be passed through at once to append more than 1 row.

```
        List<string> list1 = new List<string>()
        {
            "A",
            "",
            "C"
        };
        List<string> list2 = new List<string>()
        {
            "",
            "2",
            "3"
        };
        List<List<string>> combined = new List<List<string>>
        {
            list1,
            list2,
        };
        SpreadsheetManager.Append(new GSTU_Search(animal.associatedSheet,
animal.associatedWorksheet, "A2"), new ValueRange(combined), null);
```

GSTU_Search Class

The GSTU_Search class holds all the information needed to access a sheet and where to read. All Searches will require a spreadsheet Id which can be obtained from the spreadsheet url as shown below.

This is how the search will know what spreadsheet to read. By default the system will read from cell A1 to cell Z100 (but will stop early if there is no data past a certain cell). By editing these values you are able to adjust how much information is read, note that larger amounts of information can take longer to return its information. Title row and columns are used to enable the developer to quickly cross-reference titles to access a cell value, these are defaulted to A and 1 but are not required, title row and columns must be included within the start and end cell values

```
        public readonly string sheetId = ""; //The sheet Id to read
        public readonly string worksheetName = "Sheet1"; //the worksheet to read

        public readonly string startCell = "A1"; //the cell to start reading from
        public readonly string endCell = "Z100"; //the cell to read to, this will auto stop
reading if there is no more data

        public readonly string titleColumn = "A"; //column defined as the title column
        public readonly int titleRow = 1; //row defined as the title row.
```

Video Tutorials

Setup : https://youtu.be/uTsOhS50H7Y?list=PLuNz4X8IvmwJNtg4KiTnh9R0aKPJQPGJz

Single Data Use : https://youtu.be/T0R1SFKV_IA?list=PLuNz4X8IvmwJNtg4KiTnh9R0aKPJQPGJz

Multi Data Use : https://youtu.be/6liU2B4dMp4?list=PLuNz4X8IvmwJNtg4KiTnh9R0aKPJQPGJz

Updating in build: https://youtu.be/yqUqXvCIsGY?list=PLuNz4X8IvmwJNtg4KiTnh9R0aKPJQPGJz

Additional Notes

1. It is strongly recommended that V4 scripts described in this documentation are used as v3 script are limited on some platforms. If you experience build problems please remove the v3 scripts. These scripts will be removed in a future version.

Change Log

1.0

- Updated logic to use restful calls
- No hangs while information is loading/writing
- Updated example scenes
- Updated wrapper functions
- Clean up redundant functions
- Marked V3 as obsolete to be removed in future.

9.5.2

- Bug fix where public spreadsheets would not work as intended
- Unified version across unity versions