

# Google Doc Connector

Anyone can comment

---

[Short Overview](#)

[Setup](#)

[Guidelines](#)

[Adding Google Spreadsheet as data source](#)

[Connecting Spreadsheet values using Editor](#)

[Retrieving Spreadsheet data with scripting](#)

[API References](#)

[Tutorials and Use Examples](#)

[Association Google Data with your script fields.](#)

[Association Google Data with array fields](#)

[Localization using SpreadSheets \(dictionary\)](#)

[How to Get Support](#)

# Short Overview

---

The plugin allows you to use Google SpreadSheets as config storage for your game. The data is available via scripting API or you can connected data source to any [MonoBehaviour](#) class field, using Unity editor. You can learn more how to use plugin in Plugin use [Guidelines](#) chapter.

## Supported data types is:

- string, int, float, long
- <supported\_type>[],
- List<supported\_type>
- Dictionary<supported\_type, supported\_type>

## Editor Implementation

The plugin has unity editor implementation. Witch mean you can connect data from spreadsheet cells to any field of your [MonoBehaviour](#) script. Data can be connected to the to any field, if field type is supported. Data retrieval is recursive. For example [Vector3](#) class is serializable class with x,y,z float fields - you can connect data to this fields. The same for your own scripts. Here is [how](#) editor data connection looks like.

And here is [how](#) you can retrieve spreadsheet data using scripting.

## Advantages of keeping data in Google Spreadsheets

1. Game designers and programmers can modify config data without access to the code
2. You may use spreadsheet formulas to calculate dependencies between config value, with is extremely helpful for designing and configuring game balance.
3. The project config can be very user friendly and readable. Instead of XML and JSON or Yaml files. You may use all formatting power of Google Spreadsheets.
4. You will never lose a thing, since the google docs full revision history.

5. You can store the localization in Spreadsheets, so it will be easily accessed and modified with translators.

The requested data is cached, and stored as text files in project [Resources](#). You can find advantages and disadvantages to this approach below.

**Advantages:**

- Once data is cached Editor will not do any WWW calls anymore
- The builded application will not do any WWW calls
- Requested data is delivered immediately, without any time outs
- You can see what document version you are using in plugin setting, so you can decide when update data cash in your project from google doc. So if somebody will spoil some data in the doc, it will have zero effect to your project.

**Disadvantages**

- Data is not realtime. You can not get updated value on the fly. You need to re cash document in order to use updated data.

# Setup

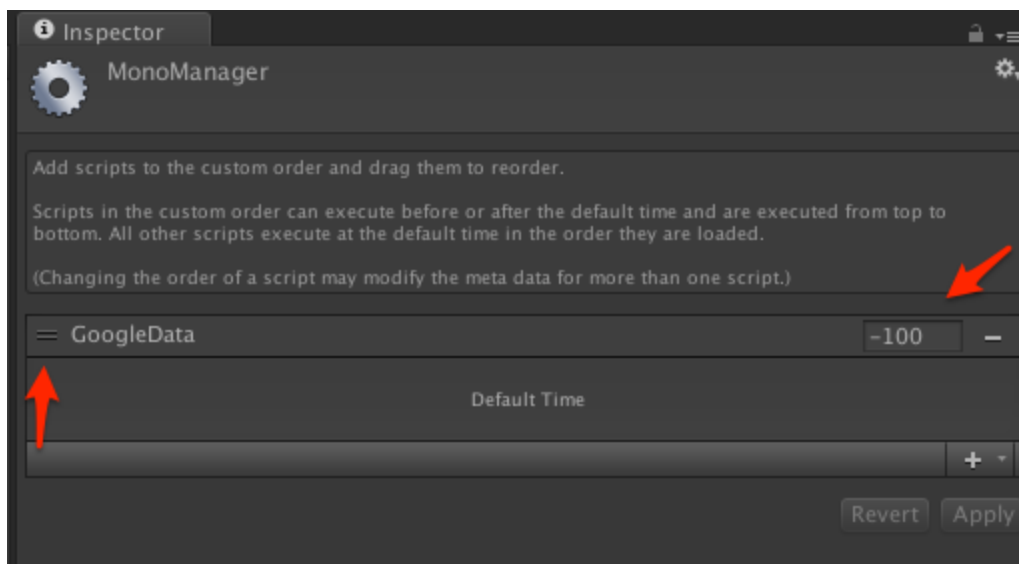
---

The plugin does is requires one set up step.

By default, the Awake, OnEnable and Update functions of different scripts are called in the order the scripts are loaded (which is arbitrary). However, we need [GoogleData.cs](#) script Awake function to be executed before all other scripts. [Script Execution Order Settings](#) help us to achieve this.

1) Open Settings window Edit → Project Settings → Script Execution Order Settings

2) [GoogleData.cs](#) Script can be added to the inspector using the Plus “+” button and dragged to change it relative order. Note that it you should to drag a script either above the **Default Time** bar;



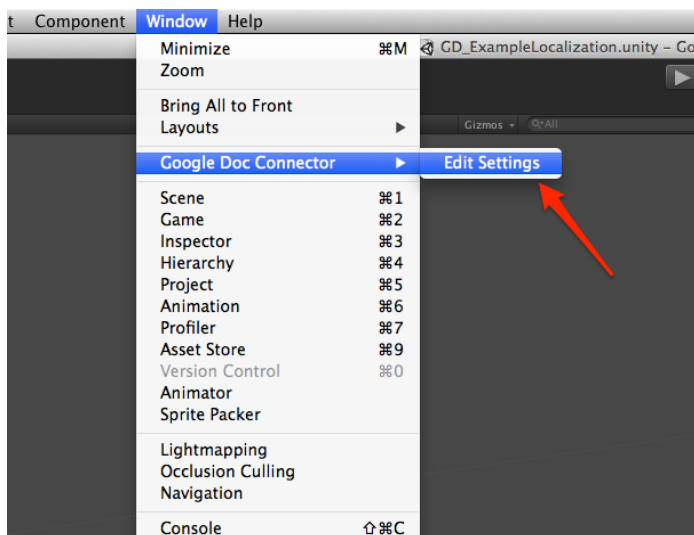
# Guidelines

---

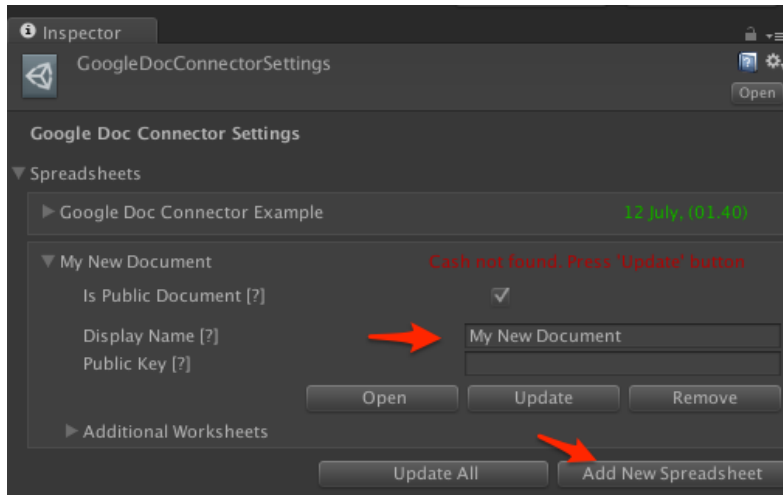
## Adding Google Spreadsheet as data source

You can use any number of documents as data source for your project. Here is few simple steps with allow you to add a document

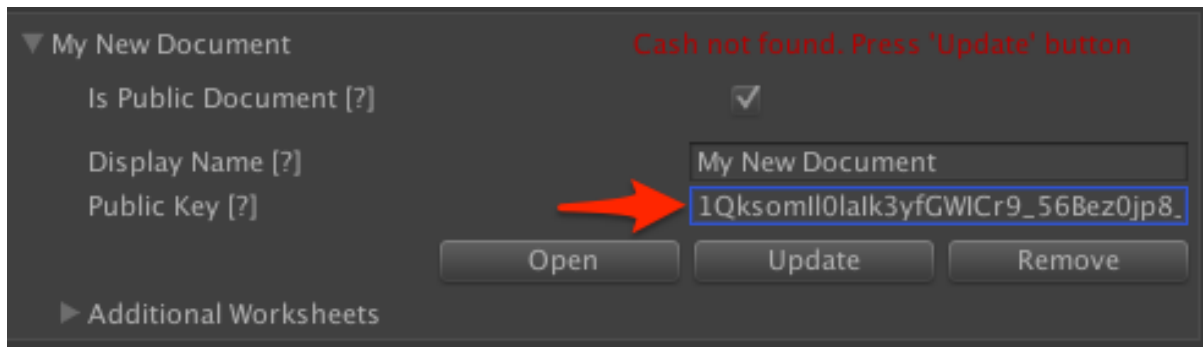
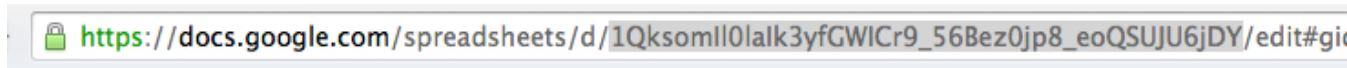
1) Open plugin settings from Unity editor menu



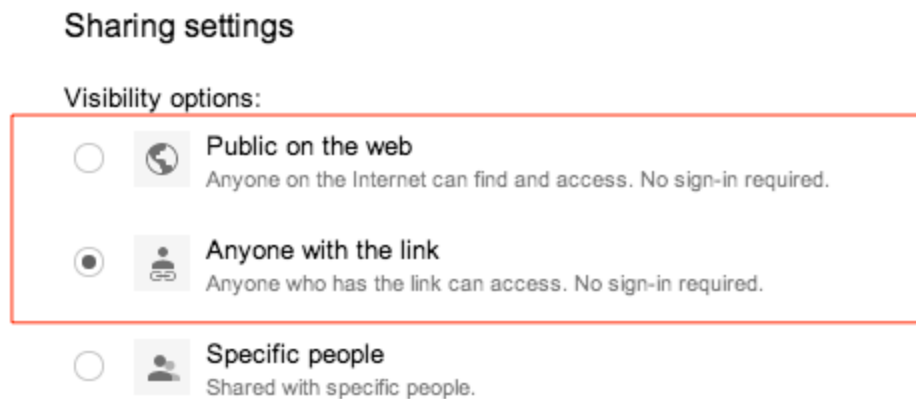
2) Create Google Spreadsheet document by clicking “Add new Spreadsheet”.  
Enter document name. This name will be used as document id in your project. it doesn't have to much with real document name.



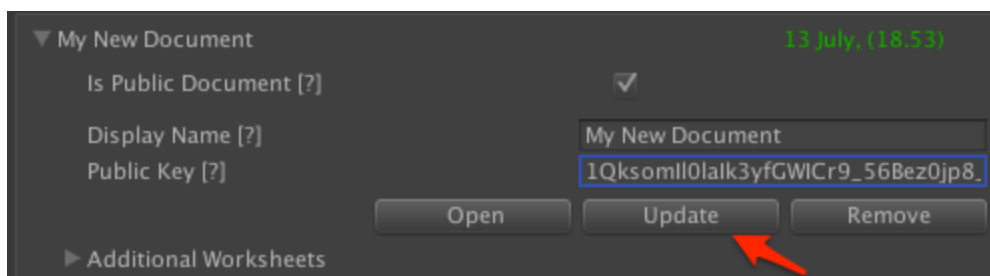
3) Fill Spreadsheet public key field key.



4) Make sure that Spreadsheet visibility options under the document sharing settings is **Public on the web** or **Anyone with the link**.

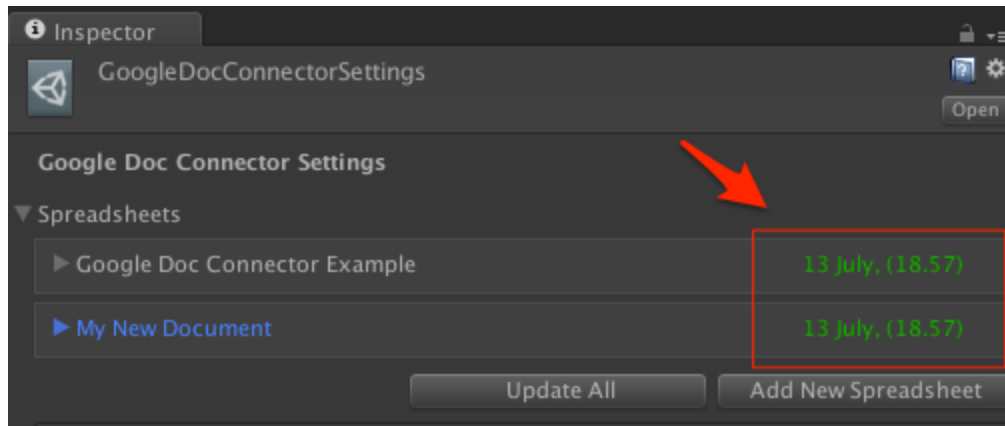


- Click “Update ” button to retrieve document data to the cash file. Every time you want to grab new data from document you can update the cache using Update or Update All data.



That is it. The data now inside your project and can be accessible via [Editor](#) or [Scripting API](#)

**Note:** Every document has the timestamp when it was changed. The timestamp is displayed near the document name.



**Warning:** All document lines should have some data, or Google Request api will simply ignore this rows. Unfortunately this is how the API works. You can just view the [example document](#) to understand what it means.

### Optionally.

You can also use Spreadsheet pages as different data sources. Instead of using different spreadsheets. To register page in associated document follow simple steps below:

- Open page in google spreadsheet you want to add
- Copy worksheet gid [UJU6jDY/edit#gid=99142187](#)
- Click **Add new Worksheet** document settings.
- Paste gid to the **Worksheet ID** field, and enter the **Worksheet Name**. This name will be used as worksheet id in your project. it doesn't have to much with real document name.



## Connecting Spreadsheet values using Editor.

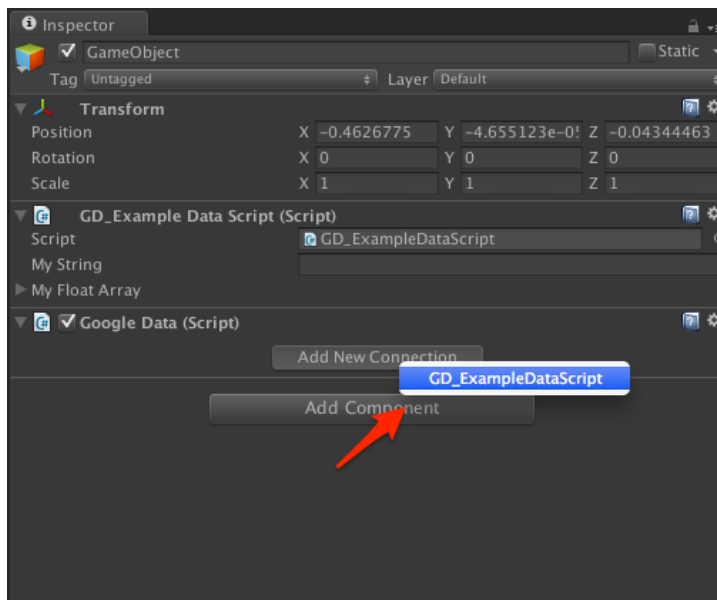
You can connect data from spreadsheet cells to any field of your [MonoBehaviour](#) script. Data can be connected to the to any field, if field with type:

- string, int, float, long
- <supported\_type>[],
- List<supported\_type>
- Dictionary<supported\_type, supported\_type>

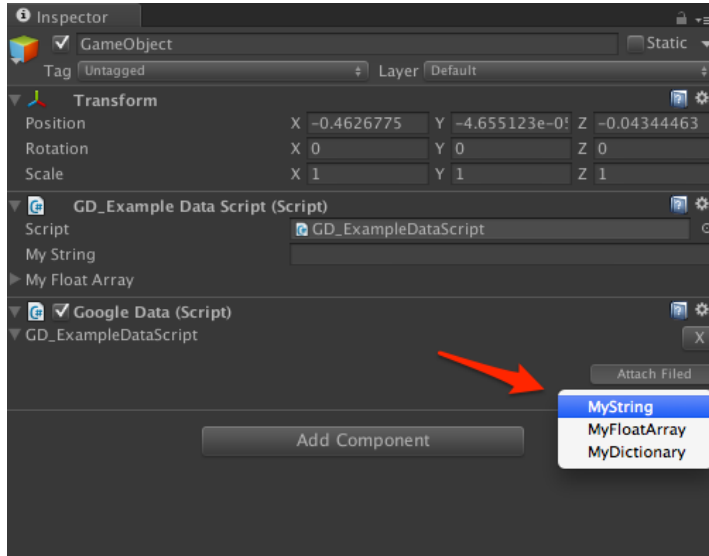
Data retrieval is recursive. For example [Vector3](#) class is serializable class with x,y,z float fields - you can connect data to this fields. The same for your own scripts. Here is how editor data connection looks like:

Here is simple steps, how to connect data to fields with type: [string](#), [int](#), [float](#), [long](#). We will use **GD\_ExampleDataScript** as example. Script contains `public string MyString;` filed. Letch attach **GD\_ExampleDataScript** and **GoogleData** scripts ot the same game object.

1) Press “Add new connection” button and select GD\_ExampleDataScript and worksheet (Optional) .

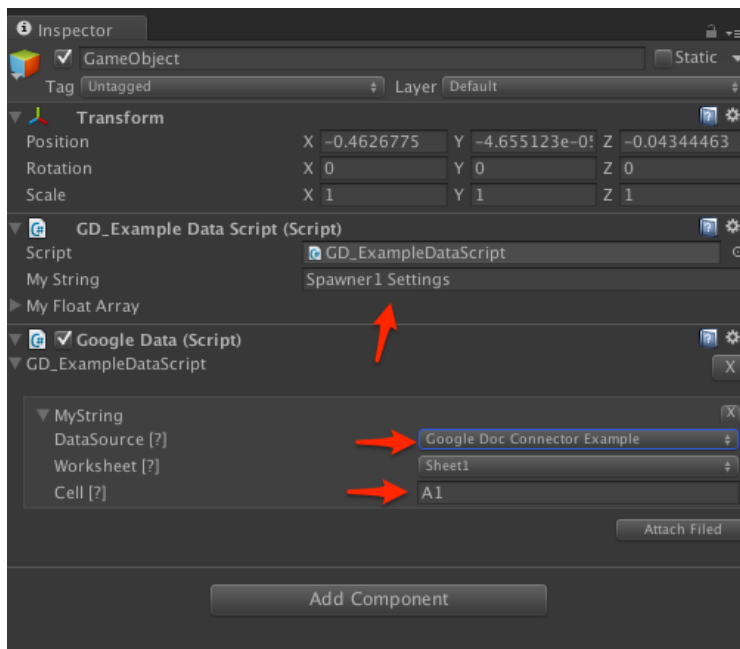


2) Press “Attach Filed” and choose **MyString** filed.



3) In field data delegate settings choose data source.

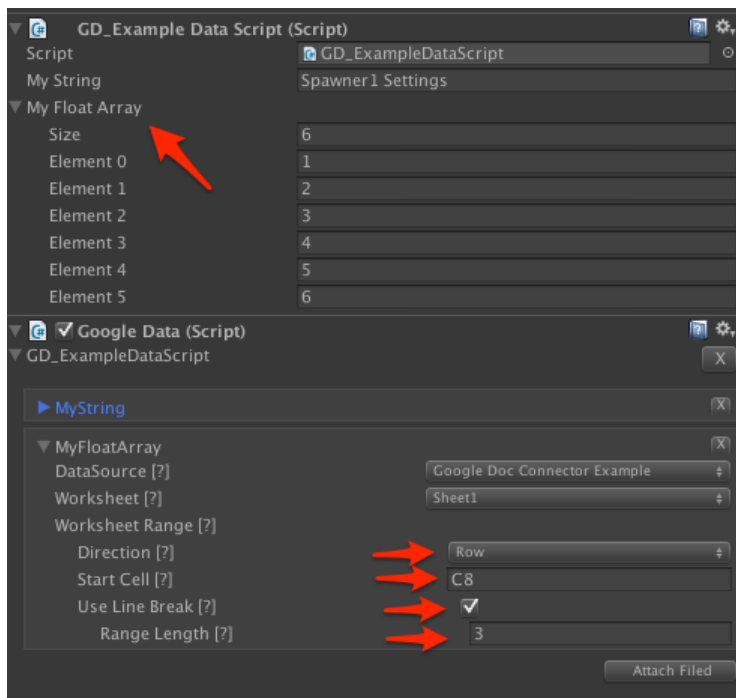
4) Specify Cell key in Worksheet, e.g. 'A1' .



The same actions for data types: [string](#), [int](#), [float](#), [long](#).

Retrieving arrays is similar. Let's use the same gameobject and script. Script contains `public float[] MyFloatArray` filed. To fill the array press "Attach Filed" and choose **MyFloatArray** filed. The data delegate setting is a bit different. Here is explanations for each of it

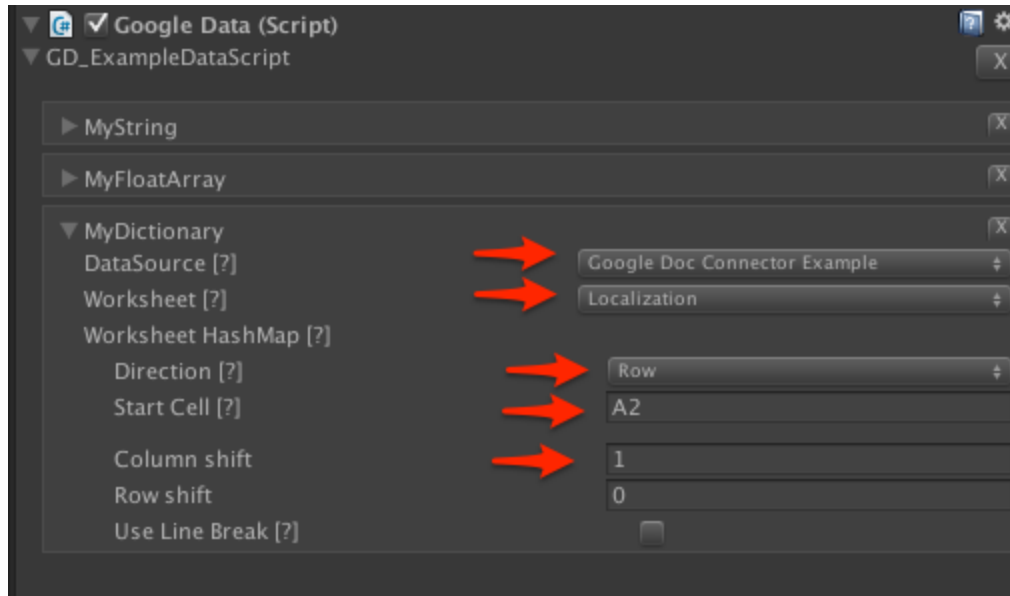
1. **Direction.** Array data direction. we can choose between row or col direction.
2. **Start Cell.** This is cell key with first element of our array.
3. **Use link break.** Enables fixed line length.
4. **Range Length.** Quantity of element before we will jump on next row / col



The same actions for data types: `string[]`, `int[]`, `float[]`, `long[]`, `List<string>`, `List<int>`, `List<float>`, `List<float>`.

And the last one, is Dictionaries. Let's use the same gameobject and script. Script contains `public Dictionary<string, long> MyDictionary` filed. To fill the dictionary press "Attach Filed" and choose **MyDictionary** filed. The data delegate setting is described below.

1. **Direction, Start Cell, Use link break, Range Length** - similar to array.
2. **Column Shift.** Value column shift regarding key column.
3. **Row Shift.** Value row shift regarding key row.



The same actions for data types:

Dictionary<string, string>, Dictionary<string, int>, Dictionary<string, long>, Dictionary<string, float>.

Dictionary<int, string>, Dictionary<int, int>, Dictionary<int, long>, Dictionary<int, float>.

Dictionary<long, string>, Dictionary<long, int>, Dictionary<long, long>, Dictionary<long, float>.

Dictionary<float, string>, Dictionary<float, int>, Dictionary<float, long>, Dictionary<float, float>.

# Retrieving Spreadsheet data with scription.

The chapter will describe how to retrieve data from spreadsheets using C# scripting in unity. You can use this api in Editor and in Runtime as well. All described code samples can be found in [DocDataRertivalExamle](#) script.

Here is code example if we need to retrieve string data from A1 cell. We will use our [GoogleDocConnectorExample](#) spreadsheet as data source. The A1 cell contains 'Spawner1 Settings' value. We can specify data cell in few different ways, the code example is below.

```
string val;

string DOC_NAME = "Google Doc Connector Example";


//Using cell col / row indexes
val = GoogleDocAPI.GetValue<string>(DOC_NAME, 1, 1);
Debug.Log(val);

//Using GDCell class with col / row indexes
GDCell cell = new GDCell(1, 1);
val = GoogleDocAPI.GetValue<string>(DOC_NAME, cell);
Debug.Log(val);

//Using GDCell class with key representation
cell = new GDCell("A1");
val = GoogleDocAPI.GetValue<string>(DOC_NAME, cell);
Debug.Log(val);
```

The Code example how to retrieve selected range from [GoogleDocConnectorExample](#) spreadsheet as int List.

<i>f<sub>x</sub></i>	Array Pattern 1				
	A	B	C	D	E
1	Spawner1 Settings				
2		Spawn Range	3	6	
3		Spawn Delay	1	0.1	
4					
5	Spawner2 Settings				
6	Spawn Start Pos:	X	-2		
7		Y	2		
8		Array Pattern 1	1	2	3
9			4	5	6
10					
11		Array Pattern 2	4	15	6
12			10	3	8
13					



```
//Retriving data range into list<int>

List<int> array;

GDCellRange range = new GDCellRange();

range.StartCell = new GDCell("C11");

range.useLinebreak = true;

range.LineLength = 3;

array = GoogleDocAPI.GetList<int>(DOC_NAME, range);

//Printing list values

foreach (int data in array) {

    Debug.Log("List Data: " + data);

}
```

# API References

## GoogleDocAPI class.

---

### API methods:

Getting the cell value. you can use row / col or [GDCell](#) class for cell representation. Use worksheet number or name to specify worksheet. First worksheet will be used by default.

```
public static T GetValue<T>(string docName, int row, int col, int workSheetNumber = 0)
public static T GetValue<T>(string docName, GDCell cell, int workSheetNumber = 0)
public static List<T> GetList<T>(string docName, GDCellRange range, string workSheetName)
```

Getting the Generic List. use [GDCellRange](#) class as range representation. Use worksheet number or name to specify worksheet. First worksheet will be used by default.

```
public static List<T> GetList<T>(string docName, GDCellRange range, string workSheetName)
public static List<T> GetList<T>(string docName, GDCellRange range, int workSheetNumber = 0)
```

Getting the Array. use [GDCellRange](#) class as range representation. Use worksheet number or name to specify worksheet. First worksheet will be used by default.

```
public static T[] GetArray<T>(string docName, GDCellRange range, int workSheetNumber = 0)
public static T[] GetArray<T>(string docName, GDCellRange range, string workSheetName)
```

Getting the Dictionary. use [GDCellDictionaryRange](#) class as map representation. Use worksheet number or name to specify worksheet. First worksheet will be used by default.

```
public static Dictionary<K, V> GetDictionary<K, V>(string docName, GDCellDictionaryRange dictionaryRange, string workSheetName)
public static Dictionary<K, V> GetDictionary<K, V>(string docName, GDCellDictionaryRange dictionaryRange, int workSheetNumber = 0)
```

*Checking type of cell value.*

```
public static bool IsValueType<T>(string docName, int row, int col, int workSheetNumber = 0)
public static bool IsValueType<T>(string docName, GDCell cell, int workSheetNumber = 0)
```

## GDCell class.

---

### Constructors

*Create cell representation with row index = 1, col index = 1 and key = "A1"*

```
public GDCell()
```

*Create cell representation using col / row indexes*

```
public GDCell(int _col, int _row)
```

*Create cell representation using cell key*

```
public GDCell(string CellKey)
```

### Get / Set:

*cell col index*

```
public int col;
```

*cell row index*

```
public int row;
```

*cell key, e.g 'A1'*

```
private string key
```



## GDCellRange class.

---

### Constructors

*Create range representation with cell key = "A1" and range direction = GDRanageDirection.Row*

```
public GDCellRange()
```

*Create range representation with cell key and range direction = GDRanageDirection.Row*

```
public GDCellRange(string StartCellKey)
```

*Create range representation with cell key and range direction*

```
public GDCellRange(string StartCellKey, GDRanageDirection rangeDirection)
```

### Get / Set:

*col or row range direction*

```
public GDRenageDirection direction;
```

*range start cell*

```
public GDCell StartCell;
```

*col specify if range has linebreak*

```
public bool useLinebreak = false;
```

*col range line length*

```
public int LineLength = 1;
```

## GDCellDictionaryRange class.

---

### Constructors

Create dictionary range representation with the default [GDCellRange](#) and no shifts

```
public GDCellDictionaryRange ()
```

Create dictionary range representation with the default [GDCellRange](#) and shifts values

```
public GDCellDictionaryRange (GDCellRange range, int colShiftValue, int rowShiftValue)
```

### Get / Set:

*keys range*

```
public GDCellRange cellRange;
```

*shifts for column and row.*

```
public int rowShift = 0;
```

```
public int columnShift = 0;
```

# Tutorials and Use Examples

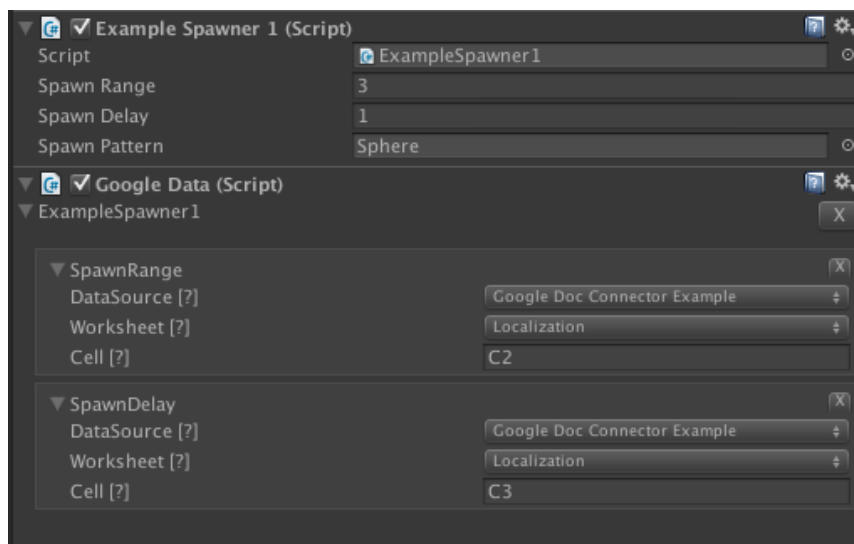
## Association Google Data with your script fields.

**Note:** Tutorial implementation can be founded under **GD\_Example** example scene

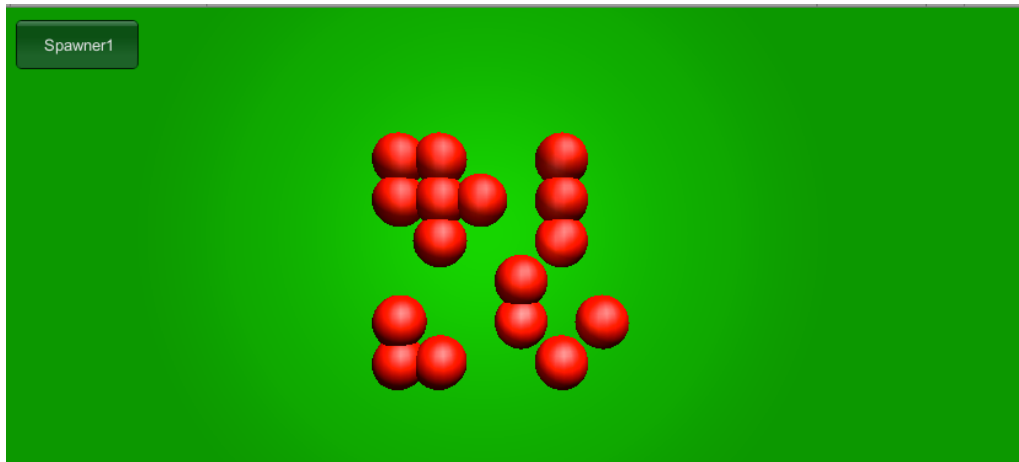
Let's learn how we can use google data on a simple game example. The example document with [Google Doc Connector Example](#) name, should be already created, in the plugin settings.

Example Scene contains simple [ExampleSpawner1](#) script. It will spawn spheres after **Spawner1** button is pressed in the play mode. Script using [public int](#) SpawnRange and [public float](#) SpawnDelay as spawn settings.

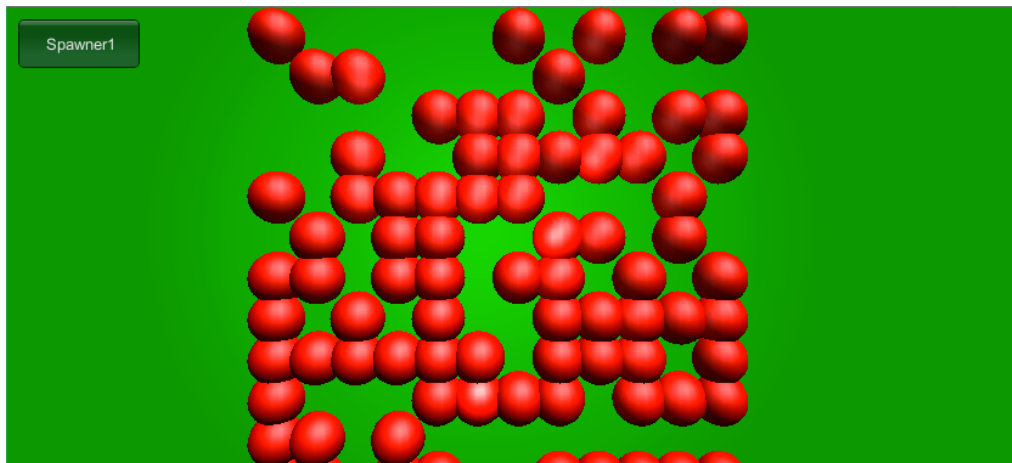
To attach data to the script, we need to add [GoogleData](#) script to the same gameobject. Lets attach **SpawnRange** to **C2** cell and **SpawnDelay** to **C3** cell.



Here is what we can see after pressing **Spawner1** button in play mode.



After Changing **C2** to **D2** and **C3** to **D3**, You will see that script values changes immediately. the same as script work result.



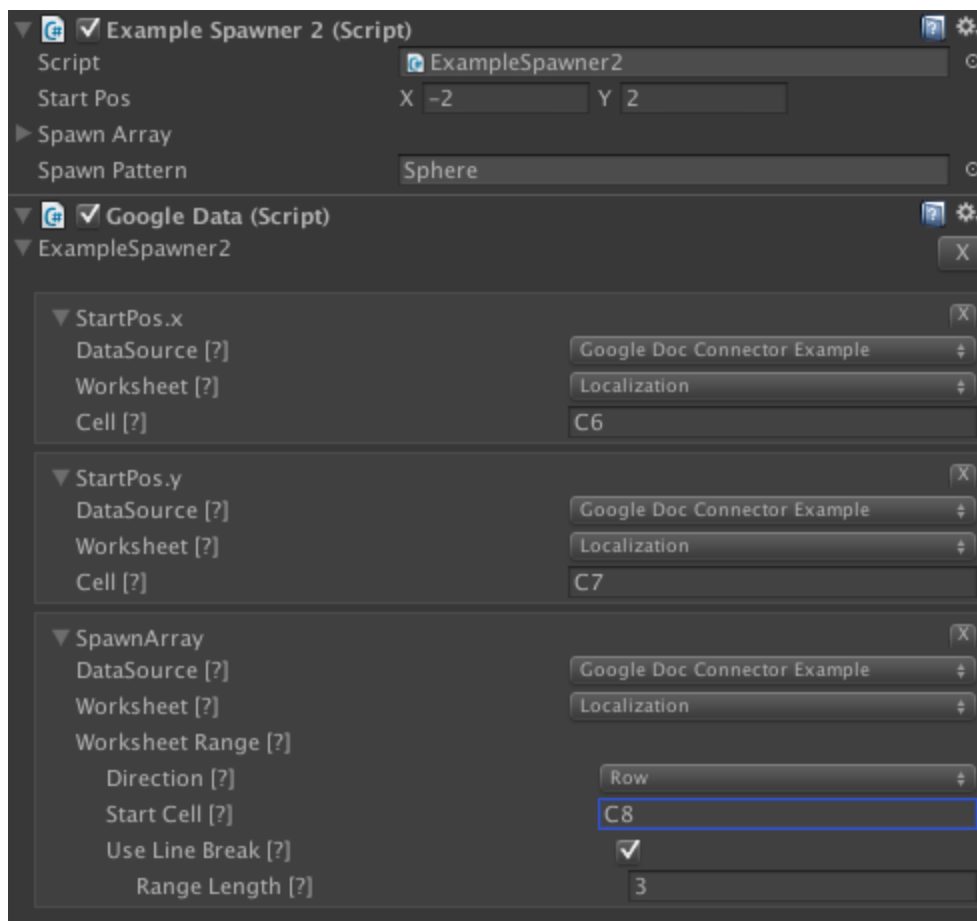
That is how easily you can bound simple value types ([string](#), [int](#), [float](#), [long](#)) to the spreadsheet cell.

## Association Google Data with array fields

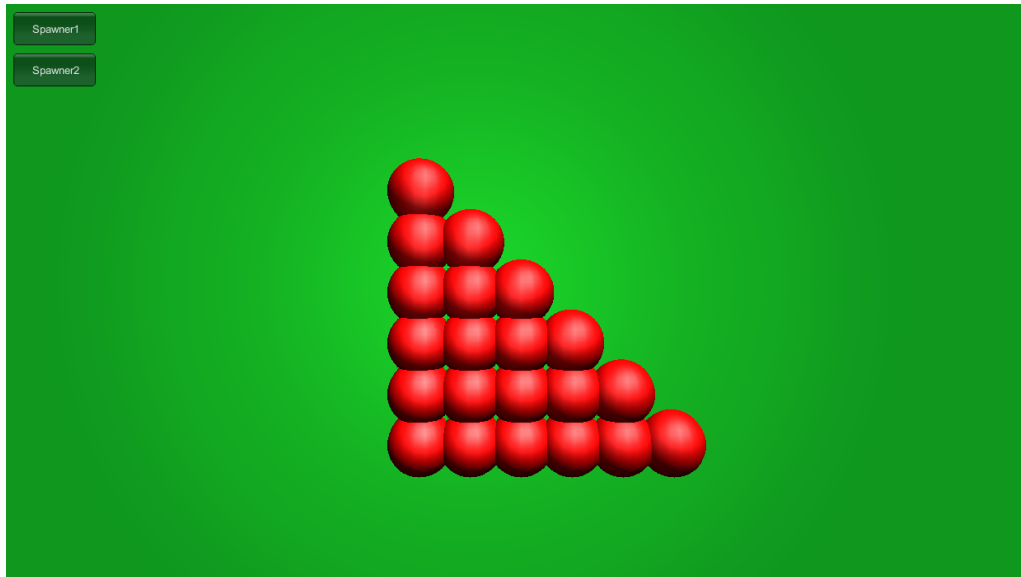
**Note:** Tutorial implementation can be founded under **GD\_Example** example scene

Same Example Scene contains simple [ExampleSpawner2](#) script. It will spawn spheres after **Spawner2** button is pressed in the play mode. Script using [public Vector2](#) StartPos and [public int\[\]](#) SpawnArray as spawn settings.

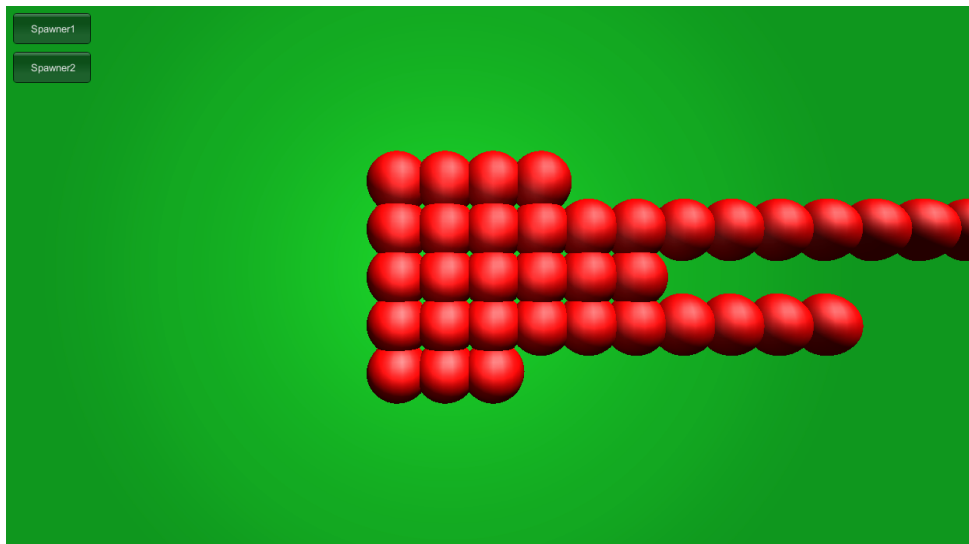
To attach data to the script, we need to add [GoogleData](#) script to the same gameobject. Lets attach **StartPos.x** to **C6** cell and **StartPos.y** to **C7** cell. For **SpawnArray** configuration let's use Start Cell **C8** and **Range Length** 3.



Here is what we can see after pressing **Spawner2** button in play mode.



After changing **SpawnArray** Start Cell from **C8** to **C11**. Script work result will be changed as on the picture below.



# Localization using SpreadSheets (dictionary)

**Note:** Tutorial implementation can be founded under **GD\_ExampleLocalization** example scene

Let's create the simple localization script using spreadsheet as data source. As example data source will use [Localization](#) worksheet of [Google Doc Connector Example doc](#).

*f<sub>x</sub>* |

	A	B	C	D	E	F
1	Tokcen	en	ru	zh	es	
2	banana	<i>Banana</i>	<i>Банан</i>	<i>香蕉</i>	<i>Plátano</i>	
3	watermelon	<i>Watermelon</i>	<i>Арбуз</i>	<i>西瓜</i>	<i>Sandía</i>	
4	melon	<i>Melon</i>	<i>Дыня</i>	<i>瓜</i>	<i>Melón</i>	
5	plum	<i>Plum</i>	<i>Слива</i>	<i>李子</i>	<i>Ciruela</i>	
6	apricot	<i>Apricot</i>	<i>Абрикос</i>	<i>杏</i>	<i>Albaricoque</i>	
7	grapes	<i>Grapes</i>	<i>Виноград</i>	<i>葡萄</i>	<i>Uvas</i>	
8						
9						
10						

Instead of first two tutorial here we going to use Scripting API instead of editor implementation. The implementation can be found in.

Create [LocalizationFromDocExample](#) script and attach it to any Gameobject on the scene.

We will need Dictionary where we will store our localization data and List of available languages.

```
private List<string> LangCodes;  
  
private Dictionary<string, Dictionary<string, string>> languages = new  
Dictionary<string, Dictionary<string, string>>();
```

Also let's define constant values

```
private const string DOC_NAME = "Google Doc Connector Example";  
private const string LOCALIZATION_WORK_SHEET_NAME = "Localization";
```

Now on **Awake** function we can retrieve all data for our localization Dictionary.  
Code example below.

```
//Getting available languages codes:  
  
GDCellRange range = new GDCellRange("B1");  
  
LangCodes = GoogleDocAPI.GetList<string>(DOC_NAME, range,  
LOCALIZATION_WORK_SHEET_NAME);  
  
//creating range of keys (tokens);  
  
GDCellRange keysRange = new GDCellRange("A2", GDRangeDirection.Column);  
  
foreach(string lang in LangCodes) {  
    //creating dictionary range for current language  
    //the row shift is 0  
    //the coll shift is lang index in LangCodes list  
    GDCellDictionaryRange dict = new GDCellDictionaryRange(keysRange,  
LangCodes.IndexOf(lang) + 1, 0);  
    //filling languages dictionary  
    languages.Add(lang, GoogleDocAPI.GetDictionary<string,  
string>(DOC_NAME, dict, LOCALIZATION_WORK_SHEET_NAME));  
}  
  
//Setting default language code and retrieving all available tokens  
//this section is only for demo GUI redner  
  
tokens = GoogleDocAPI.GetList<string>(DOC_NAME, keysRange,  
LOCALIZATION_WORK_SHEET_NAME);  
  
currentLangCode = LangCodes[0];
```



## Function that will return localized string by the token

```
private string GetLocalizedString(string langCode, string token) {
    if(languages.ContainsKey(langCode)) {
        if(languages[langCode].ContainsKey(token)) {
            return languages[langCode][token];
        }
    }
    return string.Empty;
}
```

Now the only thing left if to print our localized strings and implement ability to switch between available languages. We will use Unity native GUI for this.

```
private List<string> tokens;
private string currentLangCode;
void OnGUI() {

    int x = 10;
    int y = 10;
    foreach(string lang in LangCodes) {
        if(GUI.Button( new Rect(x, y, 140, 60), lang)) {
            currentLangCode = lang;
        }
        x += 170;
    }

    x = 10;
    y = 100;

    foreach(string token in tokens) {
        GUI.Label(new Rect(x, y, 140, 20), "Token: " + token,
        GUILayoutStyle);
        GUI.Label(new Rect(x + 300, y, 140, 20), "Localized String: " +
        GetLocalizedString(currentLangCode, token), GUILayoutStyle);
        y += 25;
    }
}
```

As example we will have GUI with printed localization and ability to swith between languages:

