



Rapid Sheet Data

Version 1.0.0

Documentation

TABLE OF CONTENTS

1. Overview	2
1.1. Features.....	2
1.2 Web service	2
1.3 Unity3D client library	2
2. Installation	3
2.1 Setup and deploy the web service.....	3
2.2. Install the Unity3D client library	6
3. Project setup	7
3.1. Setup a Google sheet for the game data	7
3.2. Setup the data classes.....	8
3.3. Create and configure the 'Rapid Sheet Data Asset'	9
3.4. Using the library.....	12
4. API.....	13
4.1. RSXObject attribute	13
4.2. RSDataAsset	13
4.3. IRSDataSerializer interface.....	14
4.4. IRSDataConverter interface.....	15
4.5. Data types supported.....	15

1. OVERVIEW

Rapid Sheet Data is a system comprised of a web service and a lightweight Unity3D library that allows you to pull game data and content from Google sheets, deserializing them to C# classes in the editor and in run time. This allows for quick iteration during development and prototyping, enabling a more data driven approach.

1.1. Features

- Web service, written in Node.js, to get data from private Google sheets, only using WWW calls from the Unity side
- Support for basic C# types and list of types (bool, int, float, string, enum) with JIT and AOT platform support
- Unity3D asset (Rapid Sheet Data Asset) created from the standard Unity's create menu, that can be easily configured in the inspector and can store a version of the data
- Simple interface to query the data either using a unique ID specified in the Google sheet or by index
- Interface to pull data from Google sheets in runtime, reducing the need to rebuild the game when tweaking values

1.2 Web service

The Rapid Sheet Data web service (backend) is responsible for handling requests coming from the client library, pulling data from your desired Google sheets, returning a JSON formatted string on success. It is written in Node.js and can be deployed to your private server or in any platform that supports Node.js (Google App Engine, Heroku, etc). In the example provided in the Unity3D client library, the web service is running on a single [Heroku](#) dyno (free version).

Setting up the web service is the most time-consuming part of the Rapid Sheet Data system and it assumes some basic knowledge of deploying Node.js applications, but it allows you to pull data from private Google sheets using simple Unity3D WWW calls, reducing the dependencies on client-side libraries and the need to store your service secret keys in the client.

The web service needs only to be setup and installed once ([2.1 Setup and deploy the web service](#)).

1.3 Unity3D client library

The Unity3D client library sends WWW requests to the web service and deserializes the returned JSON string into your desired C# classes. The core functionality of the library is contained within the Rapid Sheet Data Asset that can be created using Unity's create menu, much like all other assets, and can be easily configured in the inspector ([3.3. Create and configure the 'Rapid Sheet Data Asset'](#)).

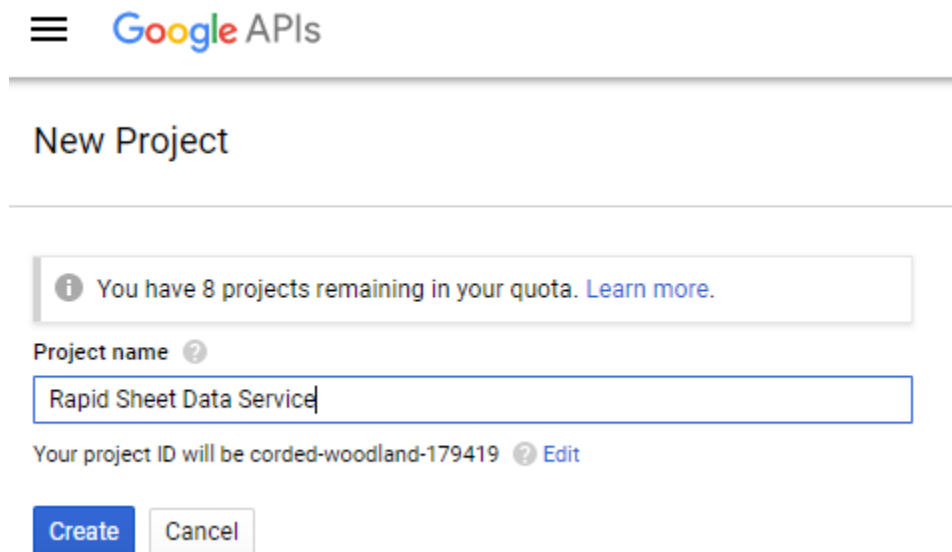
2. INSTALLATION

2.1 Setup and deploy the web service

First you need to grab the latest version of the web service from the [RapidSheetData_service](#) GitHub repo. Clone the repo and open up the directory.

In order for the service to be able to pull data from private sheets it needs to be authenticated to the Google Sheet API using OAuth 2.0. In our case since we need a service to communicate with the Google APIs, we'll need to setup a 'service account' as detailed in [Using OAuth 2.0 for Server to Server Applications](#) under 'Creating a service account'.

The process is quite straightforward: once you've logged into the [Google developer console](#) you'll need to select or create a new project and then create a service account making sure the option 'Furnish a new private key' is selected. Once the service account has been created, you will download a .json file that contains the service secret key used to authenticate the service with the Google APIs.



The screenshot shows the 'New Project' form in the Google APIs console. At the top, there is a hamburger menu icon and the text 'Google APIs'. Below this is a horizontal line, followed by the title 'New Project'. Another horizontal line follows. Below that is a message box with an information icon and the text 'You have 8 projects remaining in your quota. [Learn more.](#)'. Below the message box is the label 'Project name' with a help icon. A text input field contains the text 'Rapid Sheet Data Service'. Below the input field is the text 'Your project ID will be corded-woodland-179419' with a help icon and an 'Edit' link. At the bottom are two buttons: 'Create' (blue) and 'Cancel' (grey).

Image 2.1.1.1: Create a project, if needed

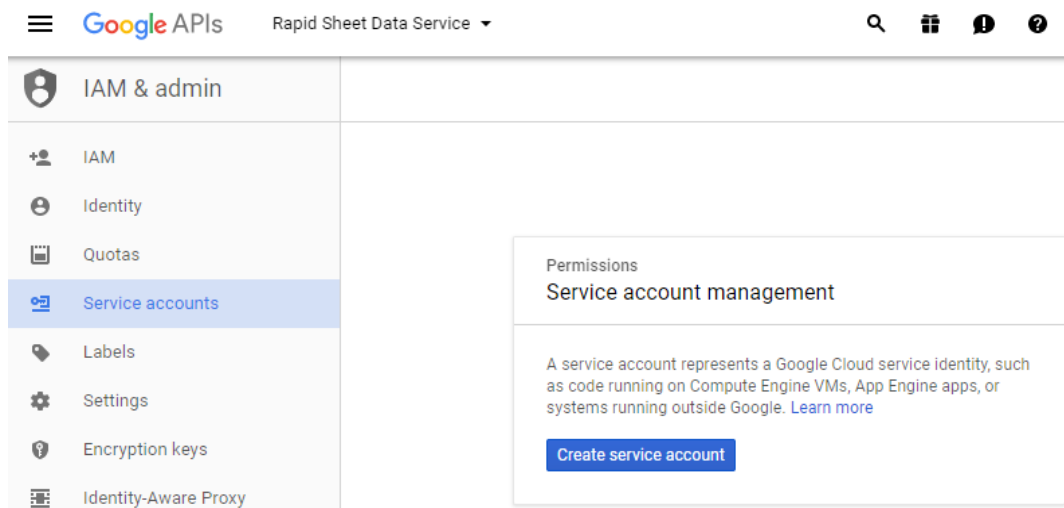


Image 2.1.2: Create a new service account

Create service account

Service account name [?] Role [?]

Service account ID

☒ **Furnish a new private key**
Downloads a file that contains the private key. Store the file securely because this key can't be recovered if lost.

Key type

☒ **JSON**
Recommended

☐ **P12**
For backward compatibility with code using the P12 format

☐ **Enable G Suite Domain-wide Delegation**
Allows this service account to be authorized to access all users' data on a G Suite domain without manual authorization on their part. [Learn more](#)

Image 2.1.3: Select 'Furnish new private key'

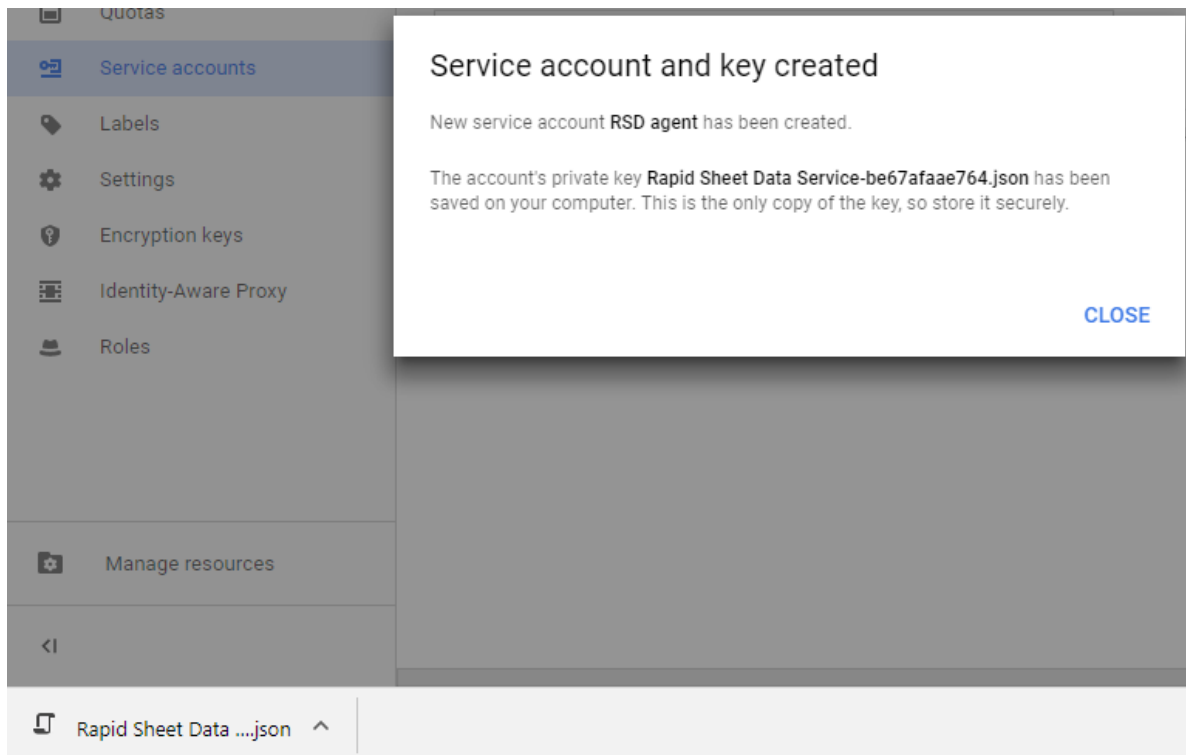


Image 2.1.4: Download the private key .json file

Before you exit the developer console make sure you have enabled the Google Sheets API for the project you just created the service account for. Type 'Google Sheets API' in the search box and click 'ENABLE'.

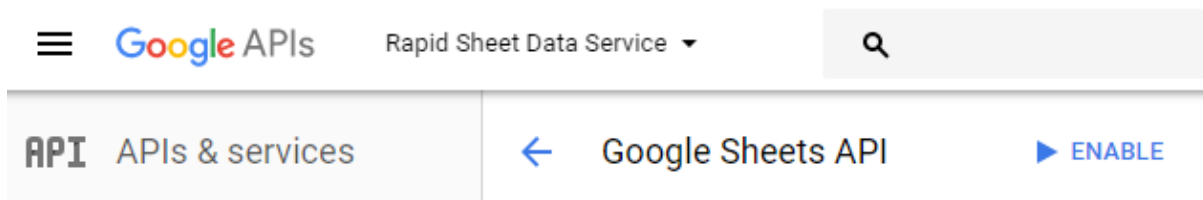


Image 2.1.5: Enable Google Sheets API

Finally, take a note of the service account ID which can be found under 'IAM & admin -> Service accounts' (in our case rapid-sheet-data@rapid-sheet-data-service.iam.gserviceaccount.com). For every sheet you want the service to have access to, you'll have to share it with this email address.



Image 2.1.6: Take a note of the service account ID

Now you need to copy the service secret .json file you just downloaded in the root directory of the Rapid Sheet Data service project. Then open up the rsdModule.js file and in line 30 paste the full filename including the .json extension in 'CLIENT_SECRET'.

```

28 var TOKEN_DIR = (process.env.HOME || process.env.HOMEPATH || process.env.USERPROFILE) + '/.credentials/';
29 var TOKEN_PATH = TOKEN_DIR + 'sheets.googleapis.com-nodejs-rsdAuth.json';
30 var CLIENT_SECRET = 'Rapid Sheet Data Service-be67afaae764.json'; // filename eg.: "rsd_example_secret.json"

```

Image 2.1.7: Copy the secret key filename to 'CLIENT_SECRET'

The Rapid Sheet Data service is now ready to be deployed to the platform of your choice. You can also run it locally with the command 'node app.js', which will start the service at <http://localhost:5000/>.

2.2. Install the Unity3D client library

Grab the latest library version from [RapidSheetData library](#) GitHub repo. You can simply copy the 'RapidSheetData' folder under 'Assets' to your project. You can also import the library from the .unitypackage located under Builds (/Builds/RapidSheetData_v1.0.0.unitypackage).

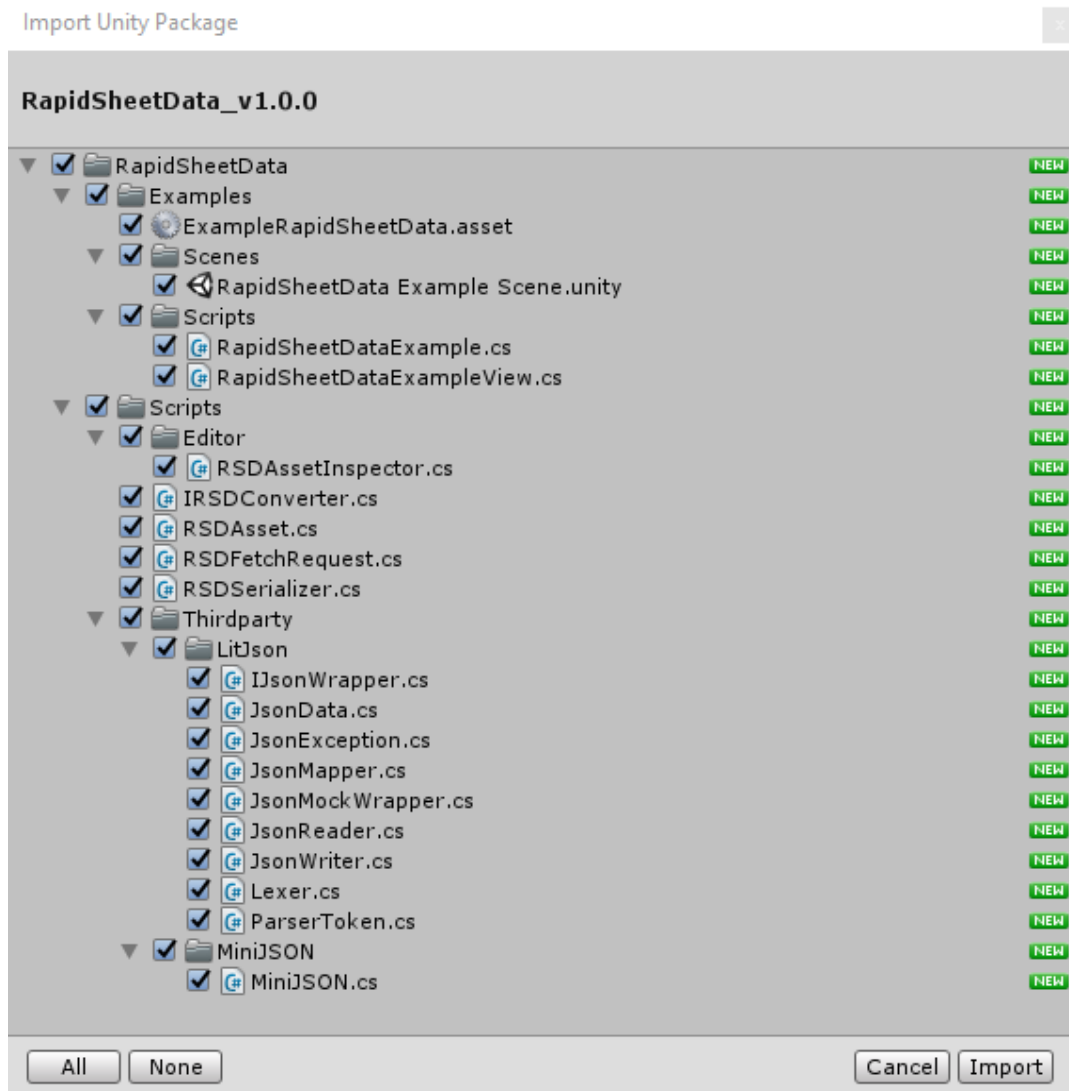


Image 2.2.1: Import the Rapid Sheet Data unitypackage

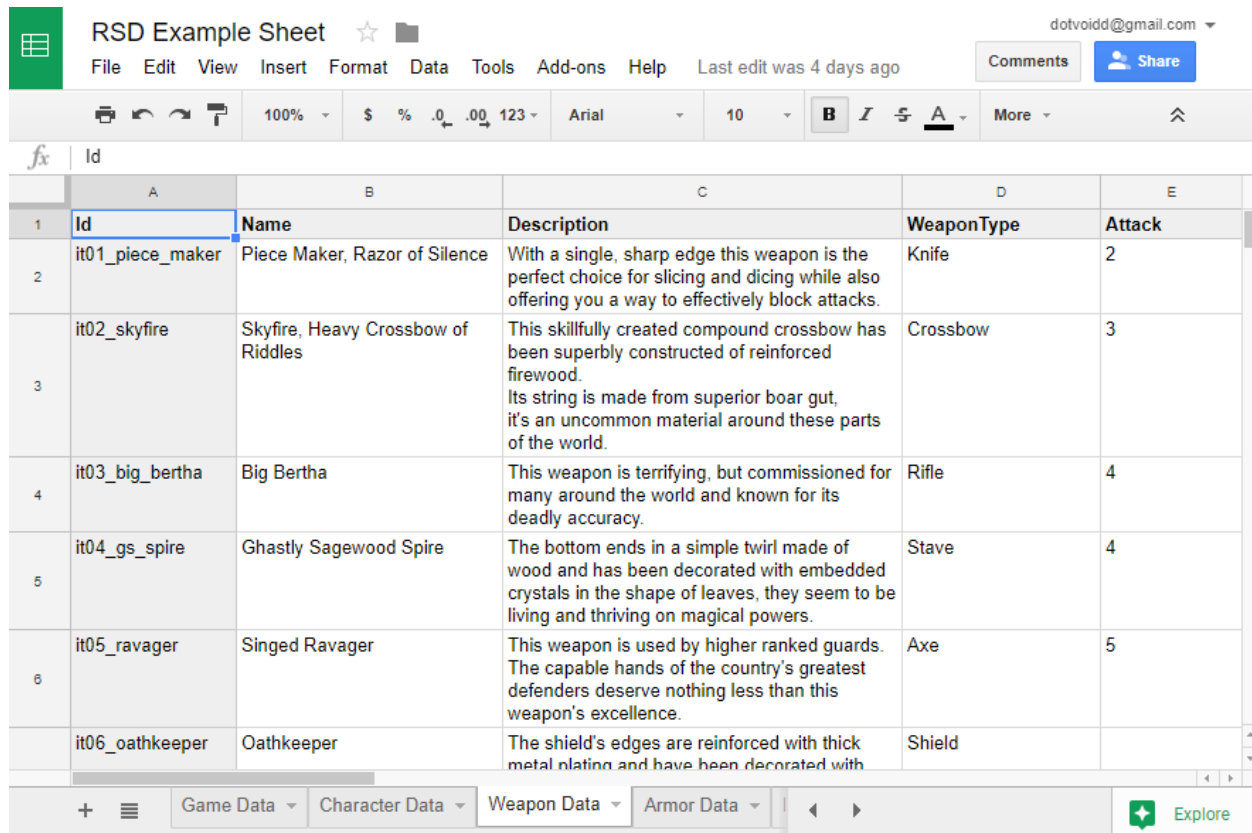
3. PROJECT SETUP

3.1. Setup a Google sheet for the game data

First you need to create a new Google. You can now add some data to pull in your game. Take a look at how the [Rapid Sheet Data - Example Sheet](#) is set up for reference.

The first row defines the names of the fields. These will be deserialized into C# class variables or property fields. If there are spaces in the name they will be removed. So, for example a field called 'Weapon Type' will look for a variable or property called 'WeaponType' in the target C# class.

The cell A1 can be defined as ID (case insensitive) which allows you to look up the data by its ID. The ID of each field should be unique (a simple sheet function can be used to do so). If an ID field does not exist (see "Example Data" sheet), you can look up the sheet data by index. Check section [4. API](#) for more information.



RSD Example Sheet

File Edit View Insert Format Data Tools Add-ons Help Last edit was 4 days ago

Comments Share

100% \$ % .0 .00 123 Arial 10 B I A More

	A	B	C	D	E
1	Id	Name	Description	WeaponType	Attack
2	it01_piece_maker	Piece Maker, Razor of Silence	With a single, sharp edge this weapon is the perfect choice for slicing and dicing while also offering you a way to effectively block attacks.	Knife	2
3	it02_skyfire	Skyfire, Heavy Crossbow of Riddles	This skillfully created compound crossbow has been superbly constructed of reinforced firewood. Its string is made from superior boar gut, it's an uncommon material around these parts of the world.	Crossbow	3
4	it03_big_bertha	Big Bertha	This weapon is terrifying, but commissioned for many around the world and known for its deadly accuracy.	Rifle	4
5	it04_gs_spire	Ghostly Sagewood Spire	The bottom ends in a simple twirl made of wood and has been decorated with embedded crystals in the shape of leaves, they seem to be living and thriving on magical powers.	Stave	4
6	it05_ravager	Singed Ravager	This weapon is used by higher ranked guards. The capable hands of the country's greatest defenders deserve nothing less than this weapon's excellence.	Axe	5
	it06_oathkeeper	Oathkeeper	The shield's edges are reinforced with thick metal plating and have been decorated with	Shield	

+ ≡ Game Data Character Data Weapon Data Armor Data Explore

Image 3.1.1: Rapid Sheed Data example sheet

The example sheet (Image 3.1.1) is public so everyone can see it for demonstration purposes, but the motivation behind Rapid Sheet Data was to be able to pull data from private sheets, which is the main reason why you had to setup the web service. To allow the web service to pull data from your private sheets, you need to share it with the service account ID you created earlier (image 2.1.6).

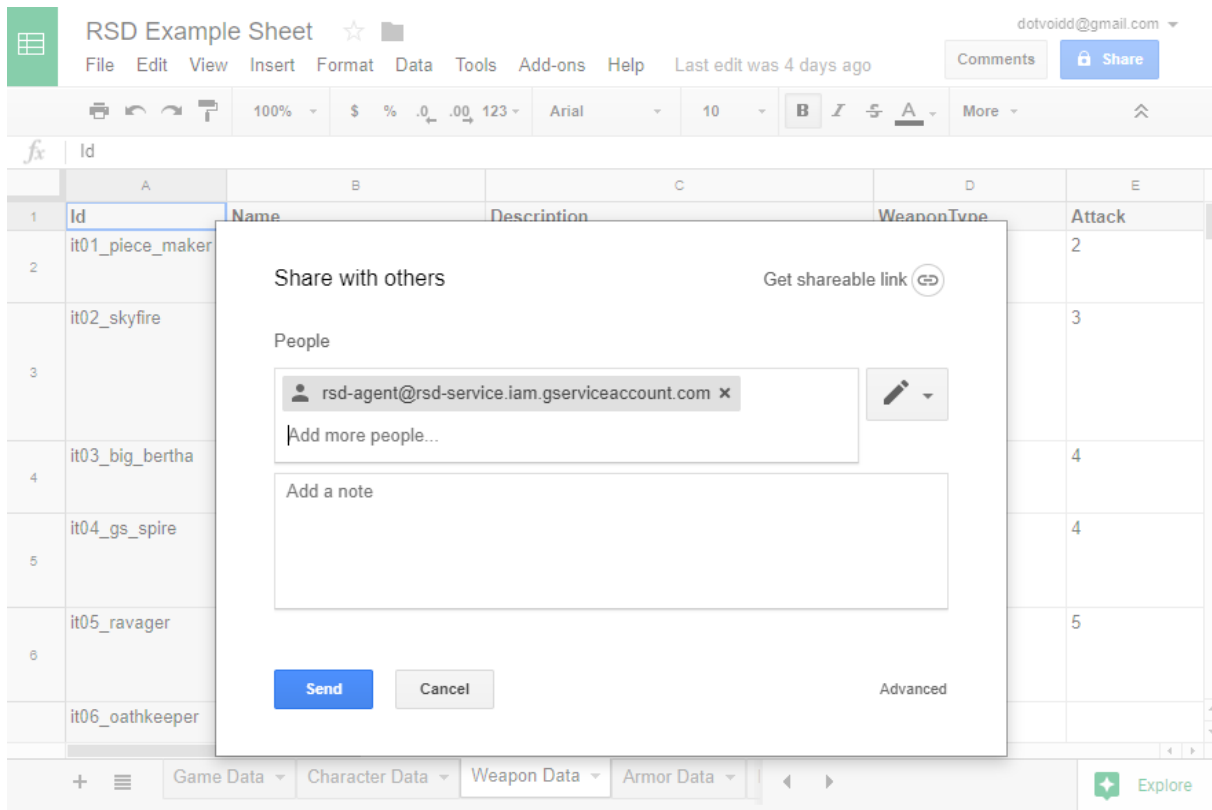


Image 3.1.2: Share the private sheet with the service account ID

Finally, take a note of the spreadsheet ID which you will use in the next step. This can be found in the URL field of the browser.

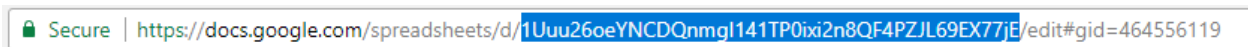


Image 3.1.3: Take a note of the Google spreadsheet ID

3.2. Setup the data classes

Before you can pull any data from the spreadsheet, you'll need to create some C# classes where the data will be deserialized to. Create a new class, add the attribute [RSDObject] and declare some public variables or properties with the same name as the fields in your spreadsheet. That's all you need to do for the Rapid Sheet Data asset to pick up the target classes. Take a look at section [4. API](#) for more details on how to use the Rapid Sheet Data API.

```

153 [RSDObject]
154 public class GameDOB
155 {
156     public string ID { get; private set; }
157     public int PlayerInitGold { get; private set; }
158     public int NumberOfStartingPlayers { get; private set; }
159     public float DaytimeLength { get; private set; }
160     public float NighttimeLength { get; private set; }
161     public string StartingLevel { get; private set; }
162
163     /// <summary>
164     ///
165     /// </summary>
166     /// <returns></returns>
167     public override string ToString()...
180 }

```

Image 3.2.1: Create a Rapid Sheet Data Object

3.3. Create and configure the 'Rapid Sheet Data Asset'

Now you can configure the Rapid Sheet Data Asset. You can create a new asset by using Unity's create menu in your desired asset folder.

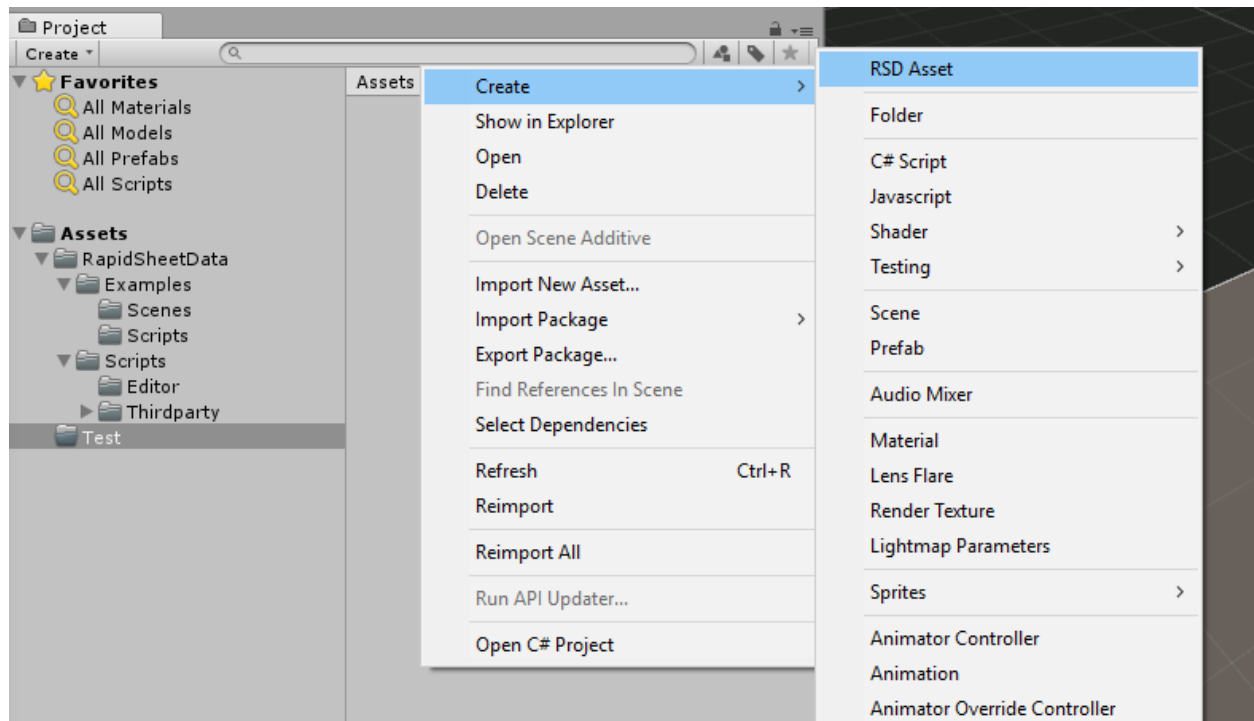


Image 3.3.1: Create a new Rapid Sheet Data Asset from the create menu.

This will create a new asset. Select it and go over to the inspector to configure it.

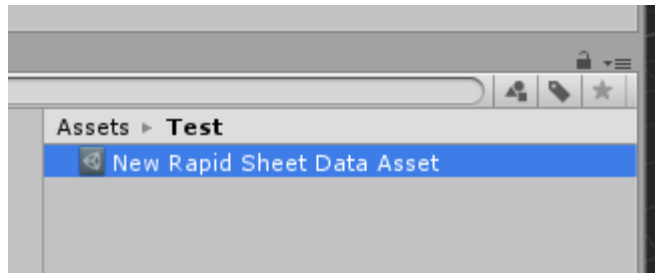


Image 3.3.2: Select the Rapid Sheet Data asset

In the inspector tab, paste the URL of the server where the Rapid Sheet Data web service runs (localhost or remote) and the spreadsheet ID you copied earlier (image 3.1.3) in the right fields.

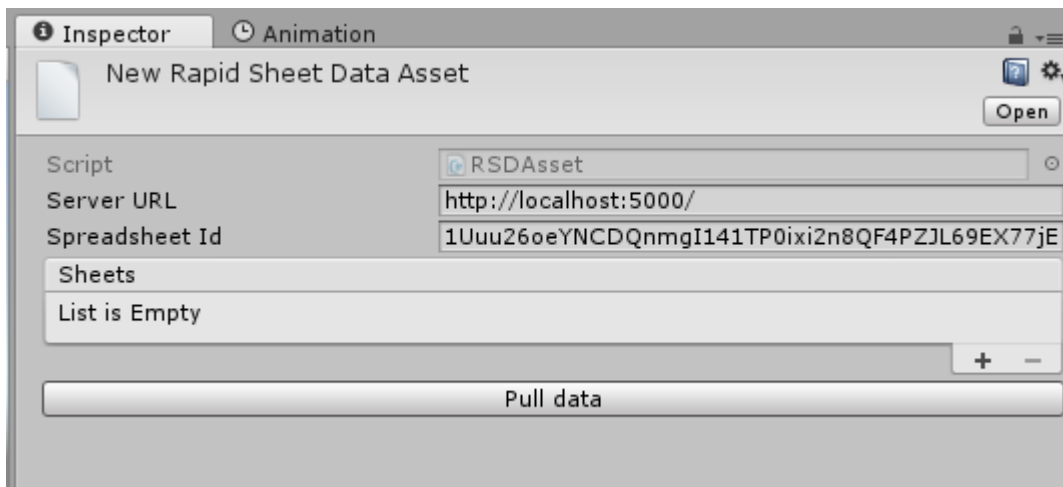


Image 3.3.3: Setup the server URL and spreadsheet ID

Next you need to configure what data the system will pull from which sheets within the spreadsheet and how it will treat them. Click the “+” sign to add a new sheet configuration. Put the name of the sheet you want to pull data from, exactly as you see it in Google sheets.

Next select the C# class the data should be deserialized to. If you followed the steps in section [3.2 Setup the data classes](#), you should see the available classes in the drop-down menu.

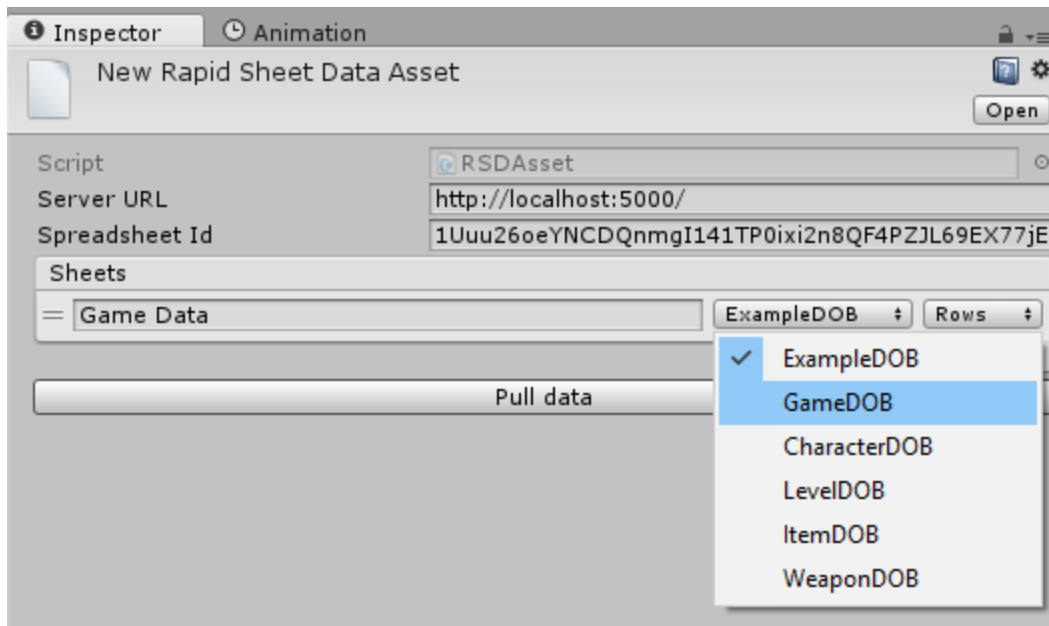


Image 3.3.4: Add the sheet name and target class in the sheet configuration

Finally, select whether the sheet data will be read row by row or column by column. Take a look at the 'Game Data' and 'Character Data' sheets in [Rapid Sheet Data - Example Sheet](#) to see the difference.

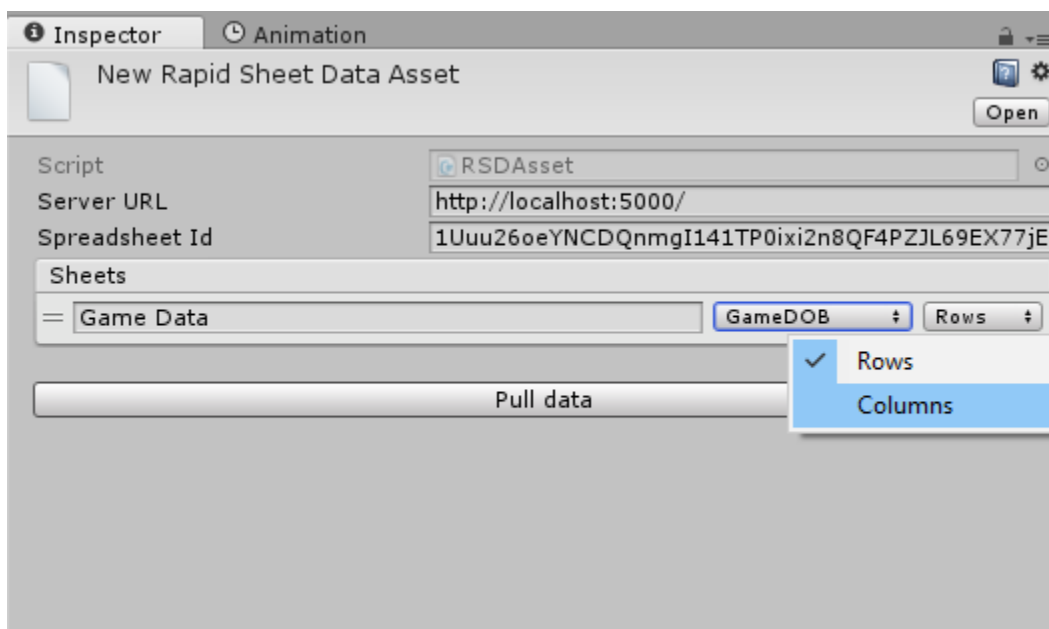


Image 3.3.5: Select how to read the data from the sheet, rows or columns.

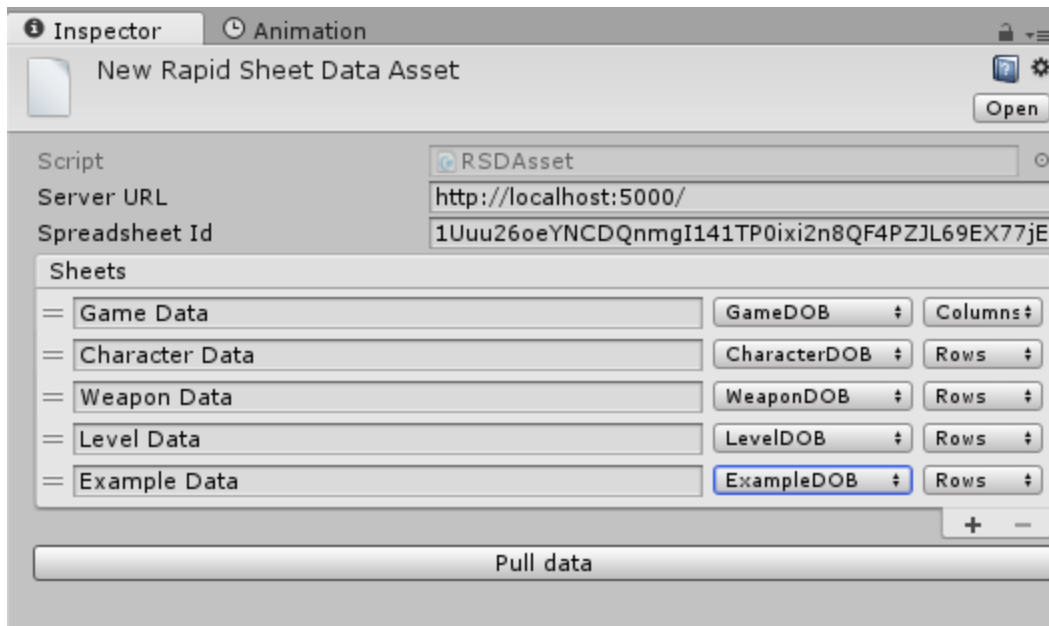


Image 3.3.6: Rapid Sheet Data Asset configured

Add as many sheets as you like, as long as they are part of the same spreadsheet. Now your Rapid Sheet Data Asset should be ready and configured. Hit the 'Pull data' button. This will send a request to the web service and you should see a message (success or failure) in your Unity console.

If the request has succeeded, your data should be saved in the asset and can be used in your game. If you have selected 'Force Text' for your asset serialization in Unity3D editor preferences, you can open up the asset you created on any text editor and see its contents.

3.4. Using the library

For an example of how to use the Rapid Sheet Data asset to either access the cached data or pull data in runtime, open up the 'RapidSheetDataExample.cs' file and look at the TestRSDAsset() function.

Essentially, all you need to do at this point, is have a reference of the asset in your script:

```
[SerializeField]
private RSDAsset _dataAsset = null;
```

And call its Init() function:

```
// Called from a MonoBehaviour script. Performs an asynchronous request
_dataAsset.Init(this);

// Called outside MonoBehaviour. Performs synchronous request
_dataAsset.Init();
```

4. API

4.1. RSDObject attribute

Use the RSDObject attribute to define a C# class where Google sheet data will be deserialized to. For Rapid Sheet Data to deserialize the data properly, the class variable or property field names should be the same ones used in the sheet's first row or column without any whitespace.

```
[RSDObject]
public class ItemDOB
{
    public string Id { get; private set; }
    public string Name { get; private set; }
    public string Description { get; private set; }
}
```

4.2. RSDAsset

An RSDAsset can be created from the create menu in the Unity3D editor and can be configured to point to the web service (Server URL) and pull data from a specific spreadsheet (Spreadsheet Id). Take a look at section [3.3. Create and configure the 'Rapid Sheet Data Asset'](#) to see how to setup the asset.

You can initialize the RSDAsset by calling its Init() function.

```
public bool Init(MonoBehaviour parent = null, IRSDSerializer serializer = null);
```

The RSDAsset can be initialized either from inside a MonoBehaviour script or from a plain C# class. If the asset is initialized without a MonoBehaviour reference then the WWW operation will be synchronous. If it a MonoBehaviour reference is passed, then the operation will run inside a coroutine, making it asynchronous.

```
RSDAsset dataAsset = null;

// 1. Initialize the asset from a plain C# class. Performs synchronous requests
dataAsset.Init();

// 2. Initialize the asset from inside a MonoBehaviour. Performs asynchronous requests
dataAsset.Init(this);

// 3. Initialize the asset from inside a MonoBehaviour, passing a custom
// IRSDSerialized and IRSDConverter
dataAsset.Init(this, new RSDSerializerDefaultLit(new RSDConverterAOT()));
```

You can get data from sheets in run time using the PullData() function. A System.Action can be passed to get a notification when the data have been downloaded and deserialized.

```
public bool PullData(Action<bool> onCompleted = null);
```

```
// Pull data from the web service and get a callback on completion
dataAsset.PullData((bool success) =>
{
    if (success)
    {
        // Do stuff
    }
});
```

You can get all the data from a sheet as a list of C# class instances using the `GetSheet<T>()` function.

```
public List<T> GetSheet<T>(string sheet);
```

```
// Get the all the data from the "Example Data" sheet as a list
List<ExampleDOB> exampleData = _dataAsset.GetSheet<ExampleDOB>("Example Data");
foreach (var data in exampleData)
{
    Debug.LogFormat("{0}\n\n", data);
}
```

You can also get specific data by ID (if an `Id` field has been specified in the sheet) or by index using the `GetFromSheet<T>()` function.

```
public T GetFromSheet<T>(int index, string sheet);
```

```
// Get the data of the first row from "Game Data"
GameDOB releaseGameData = _dataAsset.GetFromSheet<GameDOB>(0, "Game Data");
Debug.Log(releaseGameData);
```

```
public T GetFromSheet<T>(string id, string sheet);
```

```
// Get the data of the row with the id "Development" from "Game Data"
GameDOB developmentGameData = _dataAsset.GetFromSheet<GameDOB>("Development", "Game Data");
```

4.3. IRSDSerializer interface

The `IRSDSerializer` interface can be implemented to provide a custom deserializer. Implement the `Deserialize()` function.

```
///
/// Interface:      IRSDSerializer
/// Description:    The serializer / deserializer interface used by the RSD library
///
public interface IRSDSerializer
```

```

{
    /// <summary>
    /// /// <param name="sheetDesc">A List of one or more SheetSerializeDesc
    defining how the sheets should be deserialized</param>
    /// <param name="szData">The JSON formatted serialized data</param>
    /// </summary>
    /// <param name="sheetDesc"></param>
    /// <param name="szData"></param>
    /// <returns></returns>
    Dictionary<string, Dictionary<string, object>>
    Deserialize(List<SheetSerializeDesc> sheetDesc, string szData);
}

```

4.4. IRSDConverter interface

The IRSDConverter interface can be implemented to provide a custom data converter. Implement the Convert() function.

```

///
/// Interface:      IRSDConverter
/// Description:    Provides the converted interface used to convert a string value
to the desired type
///
public interface IRSDConverter
{
    /// <summary>
    ///
    /// </summary>
    /// <param name="value"></param>
    /// <param name="type"></param>
    /// <returns></returns>
    object Convert(string value, Type type);
}

```

4.5. Data types supported

The Rapid Sheet Data serializer and converter currently supports most of the basic C# types (bool, int, uint, long, ulong, float, double, enums, strings) as well as lists and arrays of those types. You can define a list or an array on your Google sheet as a comma separated list of values.

IntListField	FloatListField	String List Field	BoolListField
1,1,2,3,5,8,13,21	1,1,2,3,5,8,13,21	one, two, three,	false, true, 0
5		one, two, three,	-1, 0, 1, false, true, invalid
		, ,	true, false
2.1, 3, 3.4, 7, -1		,	false, true

Table 4.5.1: Arrays of basic C# types in Rapid Sheet Data – Example Sheet