

Meta Sheets

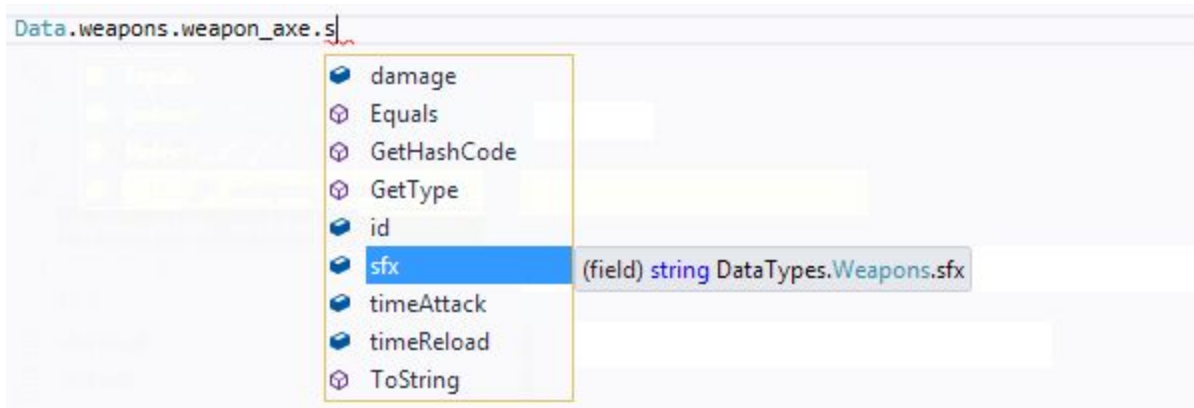
[Website](#) | [Blog](#) | [Support](#)

Overview

Meta Sheets is an Editor extension for Unity that converts google spreadsheets into **type safe** C# code. Setting up a spreadsheet is easy. Optional annotations are intuitive and enable some seriously powerful features.

| | A | B | C | D | E |
|---|----------------|------------|------------|--------|------------------|
| 1 | id | timeReload | timeAttack | damage | sfx |
| 2 | weapon_axe | 0 | 1 | 0.8 | sfx_weapon_slash |
| 3 | weapon_rpg | 3 | 0.3 | 1 | sfx_weapon_rpg |
| 4 | weapon_sword | 0 | 0.75 | 0.6 | sfx_weapon_slash |
| 5 | weapon_shotgun | 1.5 | 0.1 | 0.75 | sfx_weapon_gun |

You can access the data from any script



Once your project is running in the editor or on any platform you can reload the data straight from the spreadsheet and tweak values. For some great ideas visit the [blog](#).

Features

- **Easy** to setup and use
- Automatically **detects data types**
- Clever and optional **annotation syntax**
- **PRO** Extend your data with **custom methods** and **code**
- **PRO** Supports advanced types such as **Color**, **Dates**, **Arrays** or **eNums**
- **PRO** **Reload at runtime** and tweak values on existing builds

Pro and Free Version

For Meta Sheets there is a free version and a Pro version. The free version is very capable but doesn't support all object types and lacks a few advanced features. If you want to give Meta Sheets a try then the free version is a great start.

Meta Sheets Free

- 6 supported data Types

Meta Sheets Pro

- 18 supported data Types
- Reload at runtime
- Extend data with custom base classes
- Fully commented code

Support

Questions, feedback and issues can be raised at metasheets.helpspace.com . You can also post your comments at the **Unity forums**.



MetaSheets comes with 5 sample projects that demonstrate basic and advanced features. Examples include: Basics, an Audio Manager, Reloading and text Translations.

Data Types

By default MetaSheets assumes that your data is stored as strings unless specify a type, extend it to a base class or the values all match a certain type. **Basic Types**

The following types are supported both in the free and pro version of Meta Sheets.

| Type | Example | Default Value | Detection rules | | | | | | | | | | |
|-------------|---|---------------|-----------------|--|--|----------|--|-------|--|-------------|--|--------------|---|
| string | <table><tr><td>string</td><td>item</td></tr><tr><td colspan="2"></td></tr><tr><td colspan="2">A string</td></tr><tr><td colspan="2">Hello</td></tr><tr><td colspan="2">more text.</td></tr></table> | string | item | | | A string | | Hello | | more text. | | "" | Last resort if nothing else could be detected. |
| string | item | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| A string | | | | | | | | | | | | | |
| Hello | | | | | | | | | | | | | |
| more text. | | | | | | | | | | | | | |
| int | <table><tr><td>int</td><td>item</td></tr><tr><td colspan="2"></td></tr><tr><td colspan="2">4</td></tr><tr><td colspan="2">71</td></tr><tr><td colspan="2">-913</td></tr></table> | int | item | | | 4 | | 71 | | -913 | | 0 | Parsing as an integer succeeds. |
| int | item | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| 4 | | | | | | | | | | | | | |
| 71 | | | | | | | | | | | | | |
| -913 | | | | | | | | | | | | | |
| float | <table><tr><td>float</td><td>item</td></tr><tr><td colspan="2"></td></tr><tr><td colspan="2">0.65</td></tr><tr><td colspan="2">0.01</td></tr><tr><td colspan="2">2</td></tr></table> | float | item | | | 0.65 | | 0.01 | | 2 | | 0f | parsing as a float succeeds. |
| float | item | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| 0.65 | | | | | | | | | | | | | |
| 0.01 | | | | | | | | | | | | | |
| 2 | | | | | | | | | | | | | |
| bool | <table><tr><td>bool</td><td>item</td></tr><tr><td colspan="2"></td></tr><tr><td colspan="2">false</td></tr><tr><td colspan="2">FALSE</td></tr><tr><td colspan="2">TRUE</td></tr></table> | bool | item | | | false | | FALSE | | TRUE | | false | Parsing as a boolean succeeds. Valid boolean values are true, false, 1 and 0. |
| bool | item | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| false | | | | | | | | | | | | | |
| FALSE | | | | | | | | | | | | | |
| TRUE | | | | | | | | | | | | | |
| Vector2 | <table><tr><td>Vector2</td><td>item</td></tr><tr><td colspan="2"></td></tr><tr><td colspan="2">22.5, 48</td></tr><tr><td colspan="2">1,0</td></tr><tr><td colspan="2">0,0.55</td></tr></table> | Vector2 | item | | | 22.5, 48 | | 1,0 | | 0,0.55 | | Vector2.zero | Parsing as float[] or int[] succeeds but the array length is 2. |
| Vector2 | item | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| 22.5, 48 | | | | | | | | | | | | | |
| 1,0 | | | | | | | | | | | | | |
| 0,0.55 | | | | | | | | | | | | | |
| Vector3 | <table><tr><td>Vector3</td><td>item</td></tr><tr><td colspan="2"></td></tr><tr><td colspan="2">0,1,0</td></tr><tr><td colspan="2">0,0,0</td></tr><tr><td colspan="2">45.2,0,39.9</td></tr></table> | Vector3 | item | | | 0,1,0 | | 0,0,0 | | 45.2,0,39.9 | | Vector3.zero | Parsing as float[] or int[] succeeds but the array length is 3. |
| Vector3 | item | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| 0,1,0 | | | | | | | | | | | | | |
| 0,0,0 | | | | | | | | | | | | | |
| 45.2,0,39.9 | | | | | | | | | | | | | |

PRO Pro Types

In addition of all the basic types the pro version of Meta Sheets supports these additional types.

| Type | Example | Default Value | Detection rules | | | | | | | | |
|------------------------------|--|---------------|-----------------|------------------------------|--|---------------------|--|--------------------|--|---------------|--|
| eNum | <table><tr><td>eNum</td><td>item</td></tr><tr><td>gun</td><td></td></tr><tr><td>gun</td><td></td></tr><tr><td>sword</td><td></td></tr></table> | eNum | item | gun | | gun | | sword | | ERROR | A series of string entries each of maximum length of 24 characters that have at least 2 duplicates all together and at least 1 unique duplicate. Items can not contain odd characters. |
| eNum | item | | | | | | | | | | |
| gun | | | | | | | | | | | |
| gun | | | | | | | | | | | |
| sword | | | | | | | | | | | |
| eNum[] | <table><tr><td>eNum[]</td><td>items</td></tr><tr><td>apple, pear, orange</td><td></td></tr><tr><td>orange</td><td></td></tr><tr><td>apple, banana</td><td></td></tr></table> | eNum[] | items | apple, pear, orange | | orange | | apple, banana | | new eNum[0] | A series of individual string entries or comma separated entries each with a maximum length of 24 characters that have at least 2 duplicates altogether. |
| eNum[] | items | | | | | | | | | | |
| apple, pear, orange | | | | | | | | | | | |
| orange | | | | | | | | | | | |
| apple, banana | | | | | | | | | | | |
| Color | <table><tr><td>Color</td><td>item</td></tr><tr><td>#FF0000</td><td></td></tr><tr><td>#FF3333</td><td></td></tr><tr><td>#31748f</td><td></td></tr></table> | Color | item | #FF0000 | | #FF3333 | | #31748f | | Color.Black | A # character followed by 3 RGB or 4 ARGB values stored as hexadecimal values. |
| Color | item | | | | | | | | | | |
| #FF0000 | | | | | | | | | | | |
| #FF3333 | | | | | | | | | | | |
| #31748f | | | | | | | | | | | |
| int[] | <table><tr><td>int[]</td><td>item</td></tr><tr><td>0, 11, 1, 3, 15, 5, 6, 1, 86</td><td></td></tr><tr><td>44, 561, 235, 48</td><td></td></tr><tr><td>78, 4568, 48, 8, 7</td><td></td></tr></table> | int[] | item | 0, 11, 1, 3, 15, 5, 6, 1, 86 | | 44, 561, 235, 48 | | 78, 4568, 48, 8, 7 | | new int[0] | Parses as all comma separated integers without '.' characters succesfull. |
| int[] | item | | | | | | | | | | |
| 0, 11, 1, 3, 15, 5, 6, 1, 86 | | | | | | | | | | | |
| 44, 561, 235, 48 | | | | | | | | | | | |
| 78, 4568, 48, 8, 7 | | | | | | | | | | | |
| float[] | <table><tr><td>float[]</td><td>item</td></tr><tr><td>0, 1, 4, 6, 8</td><td></td></tr><tr><td>0.5, 0.8, 0.4, 5, 6</td><td></td></tr><tr><td>0, 0, 1.5, 4.5, 6</td><td></td></tr></table> | float[] | item | 0, 1, 4, 6, 8 | | 0.5, 0.8, 0.4, 5, 6 | | 0, 0, 1.5, 4.5, 6 | | new float[0] | Parsing as all comma separated floats succeeds. |
| float[] | item | | | | | | | | | | |
| 0, 1, 4, 6, 8 | | | | | | | | | | | |
| 0.5, 0.8, 0.4, 5, 6 | | | | | | | | | | | |
| 0, 0, 1.5, 4.5, 6 | | | | | | | | | | | |
| bool[] | <table><tr><td>bool[]</td><td>states</td></tr><tr><td>false, true</td><td></td></tr><tr><td>TRUE</td><td></td></tr><tr><td>false, 1, 0, true</td><td></td></tr></table> | bool[] | states | false, true | | TRUE | | false, 1, 0, true | | new bool[0] | A series of booleans separated by comma ','. Each parsing as an boolean successful. Valid boolean values are true, false, 1 or 0. |
| bool[] | states | | | | | | | | | | |
| false, true | | | | | | | | | | | |
| TRUE | | | | | | | | | | | |
| false, 1, 0, true | | | | | | | | | | | |
| string[] | <table><tr><td>string[]</td><td>item</td></tr><tr><td>kindle, apple, android</td><td></td></tr><tr><td>a, b, c, d, e, ,, a</td><td></td></tr><tr><td>must, can, do, eat</td><td></td></tr></table> | string[] | item | kindle, apple, android | | a, b, c, d, e, ,, a | | must, can, do, eat | | new string[0] | A series of strings separated by comma ',' of which each string does not contain the following characters: tab, newline, quotation signs. |
| string[] | item | | | | | | | | | | |
| kindle, apple, android | | | | | | | | | | | |
| a, b, c, d, e, ,, a | | | | | | | | | | | |
| must, can, do, eat | | | | | | | | | | | |

Color[]

| |
|---------------------------|
| Color[] item |
| #ff0000, #1122ee, #00ff00 |
| #00ff00, #ff2244 |
| must, can, do, eat |

new Color[0]

Parses all comma separated #hexadecimal values successful. A single hexadecimal starts with a # character followed by either 6 or 8 characters of the hexadecimal character set.

Class

| |
|--------------|
| people who |
| item_00 |
| item_02 |
| item_07 |
| people items |

null

The header needs to start with any of the sheet names followed by any variable name. Any of the entered values need to start with the same sheet name followed by a '.' and a identifier of that referenced sheet.

Class[]

| |
|---------------------------|
| people[] who |
| item_00, item_01, item_02 |
| item_02 |
| item_07, item_09 |
| people items |

new class[0]

The same rules as the 'Class' type apply here except that everything is treated as an array. Each entered array value must contain the sheet class name reference followed by a '.' and an identifier of that referenced sheet.

DateTime

| |
|---------------|
| DateTime item |
| 11/4/2015 |
| 1/1/2014 |
| 3/8/1982 |

DateTime.Today

3 digit pairs separated by / characters. Either as YYYY/MM/DD or DD/MM/YYYY order.

TimeSpan

| |
|---------------|
| TimeSpan item |
| 24:49 |
| 12:41:00 |
| 12:00:05 |

new
TimeSpan(0,0,0)

2 or 3 pairs of integers below 60 separated by : characters. Expects either a hh:mm:ss or mm:ss order.

Documentation

This manual should get you started using Meta Sheets. The following part of the documentation applies to both the Free and Pro version of Meta Sheets. For Meta Sheets pro specific documentation scroll further down.

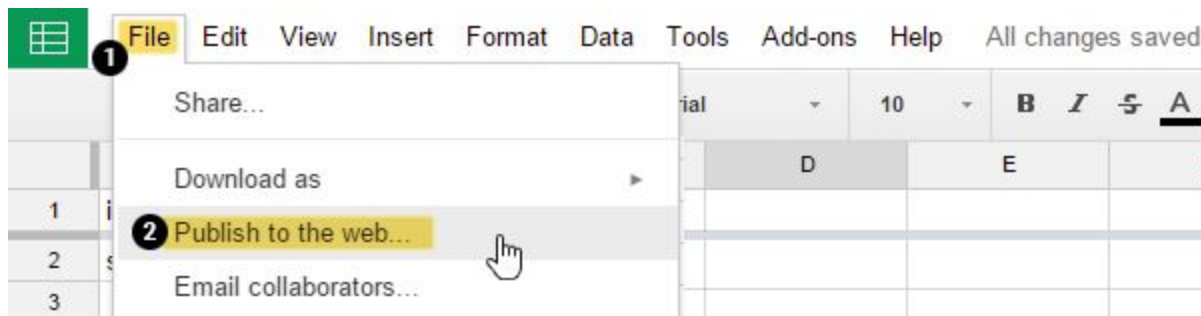
Setting up a Google sheet

- Create a [new google sheet](#) and consider the following:
 1. The sheet name (tab name) describes the category.
 2. The first row is used to describe your **variables**.
 3. Starting with the second row each represents an **object** that Meta Sheets will parse against the properties of the first row.

| | A | B | C | D | E | F |
|---|--------------------|---------|--------|-------------|---|---|
| 1 | B id | color | health | has powerup | | |
| 2 | C speedstar | #ff0000 | 1 | TRUE | | |
| 3 | C rocket | #2244ff | 0.8 | FALSE | | |

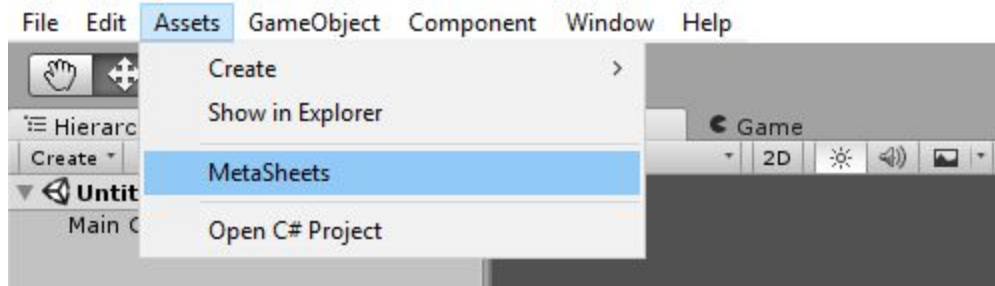
+ cars **A**

- Next select
 1. File
 2. Publish to the web...
 3. Choose the "**Entire Document**" and then press **Publish**. This will make the document readable from outside but keep your document unlisted from the web.

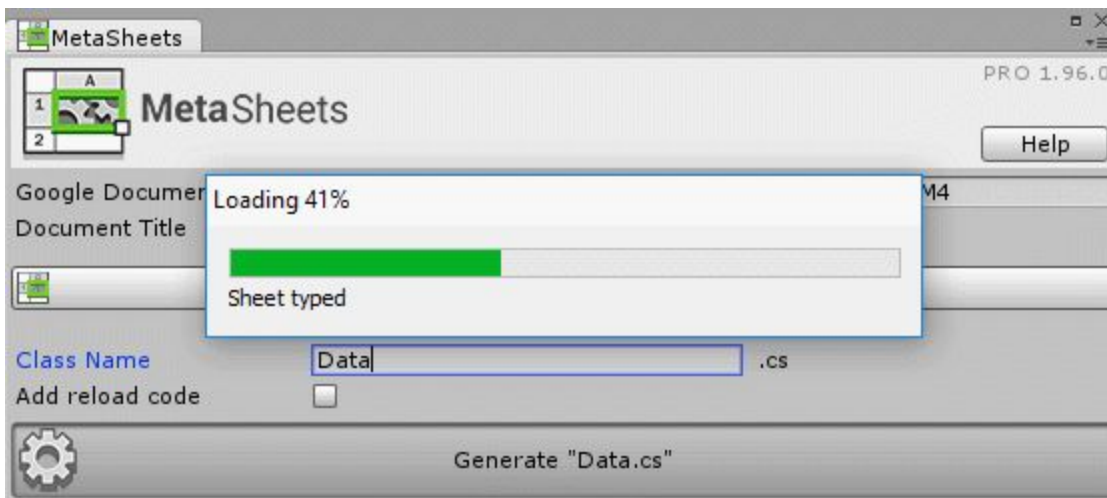


Generating Code

- Open Meta Sheets from the Unity menu: **Assets > Meta Sheets**



- **Copy** and **paste** the URL of the google document (make sure it is published), Meta Sheets should automatically extract the correct key.
- Enter a **class name**, for example "**Data**"
- Click **Generate "Data.cs"**.



Tip: The class name determines how you access code later from your C# scripts. Essentially the class name will be your entry point to access all of the data.

How to access the data

When converting a spreadsheet into C# code the following translates into parts of the generated code

| | A | B | C | D | E | F |
|---|-----------|---------|--------|-------------|---|---|
| 1 | id | color | health | has powerup | | |
| 2 | speedstar | #ff0000 | 1 | TRUE | | |
| 3 | rocket | #2244ff | 0.8 | FALSE | | |

+

☰

cars

- A. **Sheets** (tabs) turn into **classes** each with a static singleton.
- B. **Rows** turn into **objects**; all objects together into an array.
- C. **Columns** turn into **properties**.

| | A | B | C | D | E | |
|---|----------------|------------|------------|--------|------------------|--|
| 1 | id | timeReload | timeAttack | damage | sfx | |
| 2 | weapon_axe | 0 | 1 | 0.8 | sfx_weapon_slash | |
| 3 | weapon_rpg | 3 | 0.3 | 1 | sfx_weapon_rpg | |
| 4 | weapon_sword | 0 | 0.75 | 0.6 | sfx_weapon_slash | |
| 5 | weapon_shotgun | 1.5 | 0.1 | 0.75 | sfx_weapon_gun | |

+

☰

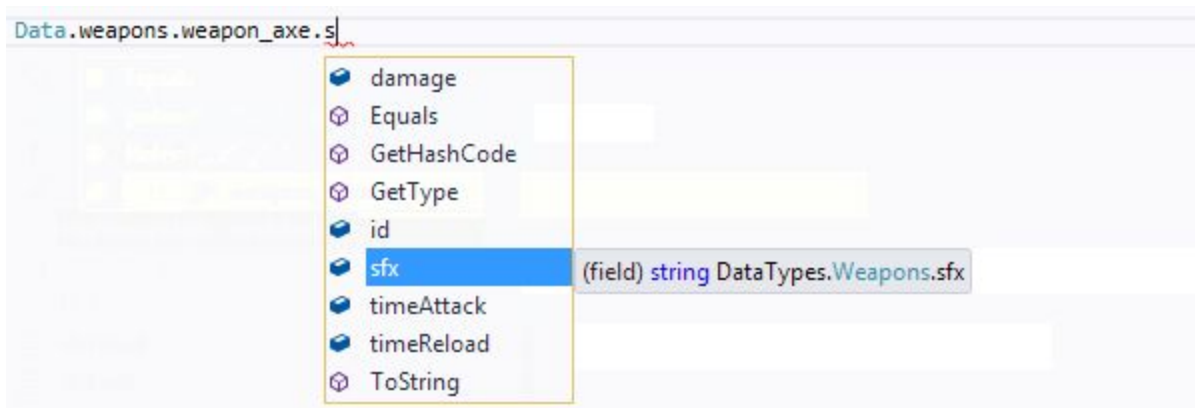
weapons

potions

Tip: When your first column consists of strings Meta Sheets will create id accessors that you can access from code. Otherwise you have to access it as an array[] item or from within a foreach loop.

Inside your scripts you can then start typing the names of the sheets followed by their id's and then their properties. Like

`{Data}.{sheet}.{id}.{property}`



Apart from accessing data from their static variables you can also Access items as Array like list of items

//Static id Access

```
Debug.loop("Sfx of axe: "+Data.weapons.weapon_axe.sfx);
```

//Access as string ID

```
Debug.loop("Sfx of axe: "+Data.weapons["weapon_axe"].sfx);
```

//Access in a for i loop

```
for(int i=0; i < Data.weapons.Length; i++){  
    Debug.Log("Sfx: "+Data.weapons[i].sfx);  
}
```

//Access in a foreach loop

```
foreach(DataTypes.weapon weapon in Data.weapons){  
    Debug.Log("Sfx: "+weapon.sfx);  
}
```

Entering values

Most of the time you can name columns, rows and sheets however you want. In some cases however MetaSheets will automatically convert names to c# safe object names. This may happen when some of your identifiers contain unsafe characters such as `\#-.?<> []{}()` or are equal to unsafe keywords such as **abstract**, **as**, **base**, **bool**, **break**, **byte**, ...

For the values certain types of values expect a certain way of entering the values in the spreadsheet. Most of the time you write the values as if you would write them in code. In most cases Meta Sheets will automatically figure out how to convert your data. Should it fail however Meta Sheets or the compiler will tell you in detail what type of data was expected where.

A complete list of the supported object types and their syntax can be found at the Data Types section further below.

| string item | bool item | eNum item | Color item |
|-------------|-----------|-----------|------------|
| A string | false | gun | #FF0000 |
| Hello | FALSE | gun | #FF3333 |
| more text. | TRUE | sword | #31748f |

Ignoring elements

Empty rows are ignored when building or reloading sheet data in Meta Sheets. You can mark sheets, columns or rows to ignore in Meta Sheets by adding a `//` in front of their names.

| | A | B | C | D | E | F |
|---|----------------|------|---------|------------|-----------|--------------|
| 1 | id | time | //color | offset | usesClips | colorOutline |
| 2 | weapon_axe | 0 | #fe32d5 | 0.50,0.75 | FALSE | #fe32d5 |
| 3 | weapon_rpg | 3 | #7afe32 | 0.25,0 | TRUE | #7afe32 |
| 4 | // unkown | 1 | | 0.25,7 | FALSE | #7afe32 |
| 5 | weapon_sword | 0 | #3b7eed | 0,1 | FALSE | #3b7eed |
| 6 | weapon_shotgun | 1.5 | #fe32d5 | 0.75, 0.35 | TRUE | #fe32d5 |

Above: The item `// unkown` will be ignored as well as the property `//color`. You can also ignore entire tabs by adding `//` in front of the tab name:



Enforcing types

Meta Sheets can detect object types automatically if the majority of all values match a certain criteria. Sometimes however you might want to enforce a certain variable type. To enforce a specific variable type write the object type in front of the column variable name. The way you write this follows the same convention as you would do in C#.

| | A | B | C | D | E | |
|---|----------------|------------------|-------------|----------------|----------------|--|
| 1 | id | float timeAttack | Color color | Vector2 offset | bool usesClips | |
| 2 | weapon_axe | 0 | #fe32d5 | 0.50,0.75 | FALSE | |
| 3 | weapon_rpg | 3 | #7afe32 | 0.25,0 | TRUE | |
| 4 | weapon_sword | 0 | #3b7eed | 0,1 | FALSE | |
| 5 | weapon_shotgun | 1.5 | #fe32d5 | 0.75, 0.35 | TRUE | |

Note that some object types are only supported in the pro version of Meta Sheets. A complete list of the supported object types and their syntax can be found at the Data Types section.

Below are a few example types that can be forced in your spreadsheet.

| string item | bool item | eNum item | Color item | Vector2 item |
|-------------|-----------|-----------|------------|--------------|
| A string | false | gun | #FF0000 | 22.5, 48 |
| Hello | FALSE | gun | #FF3333 | 1,0 |
| more text. | TRUE | sword | #31748f | 0,0.55 |

PRO eNums

Meta Sheets automatically detects enums when a series of strings have at least 2 duplicates. You can however also enforce a eNum type by writing **eNum** in front of your column label. eEnumerators can also be used as **eNum[]** arrays with comma seperated values.

| | A | B | C | |
|---------------|----------------|------------|-------------------|--|
| 1 | id | eNum type | eNum[] materials | |
| 2 | weapon_axe | swing | wood, metal | |
| 3 | weapon_rpg | projectile | metal, explosives | |
| 4 | weapon_sword | swing | metal | |
| 5 | weapon_shotgun | projectile | metal, wood | |
| | | | | |
| + ☰ weapons ▾ | | | | |

Meta Sheets will automatically generate an eNum structure with all of the unique items found from your spreadsheet. All of the data types are stored in a special name space called **{Data}Types** where {Data} is the name of your data class in MetaSheets unless you extended to a base class.

```
//Comparing weapon type
if(Data.weapons.weapon_axe.type == DataTypes.weapons.eType.swing){
    Debug.Log("This weapon is of type 'swing'");
}
//Log type and material values
Debug.Log("Type: "+ Data.weapons.weapon_sword.type.ToString() );
Debug.Log("Materials: "+Data.weapons.weapon_axe.materials[0].ToString() );
```


Items from other sheets (tabs) can be referenced as **Class** types. This will store a reference to the generated MetaSheets object and it's properties. This type reference is type safe as there are no string or array references used.

The following example casts the type **people** for the variable **who**. As you can see a sheet by the name **people** actually exists and will be referenced here in the generated code.

| | A | B | |
|---|--------------|------------|--|
| 1 | id | people who | |
| 2 | weapon_axe | item_00 | |
| 3 | weapon_rpg | item_02 | |
| 4 | weapon_sword | item_07 | |

+

☰

people ▼

weapons ▼

Important: You can only reference Class items if the sheet or tab you reference uses string ID's in the first column. This can be the ID column for example.

| | A | B | |
|---|---------|---|--|
| 1 | id | | |
| 2 | item_00 | | |
| 3 | item_01 | | |
| 4 | item_02 | | |

+

☰

people ▾

weapons ▾

In the script the property "who" will be casted of the type "people"

```
//Access a referenced object of type "people"
Debug.Log("Axe is owned by: "+Data.weapons.weapon_axe.who.id);
```

```
//A more detailed break down
DataTypes.people object = Data.weapons.weapon_axe.who;
Debug.Log("Id of 'who' object inside 'weapon_axe': "+ object.id);
```

Class[] array references

Apart from referencing just a single item you can also reference Class array items

| | A | B | |
|------------------------|--------------|------------------------------------|--|
| 1 | id | people[] who | |
| 2 | weapon_axe | item_00, item_02 | |
| 3 | weapon_rpg | item_03 | |
| 4 | weapon_sword | item_01, item_00, item_02, item_04 | |
| | | | |
| + people weapons | | | |

In the script the property "who" will be cast as an array of the type "people"

```
//Determine length
Debug.Log("People count: "+Data.weapons.weapon_axe.who.Length);

//Access the first item
Debug.Log("First person: "+Data.weapons.weapon_axe.who[0]);

//Access in a for i loop
for(int i=0; i < Data.weapons.weapon_axe.who.Length; i++){
    Debug.Log("Person["+i+"] = "+Data.weapons.weapon_axe.who[i]);
}

//Access in a foreach loop
foreach(DataTypes.person person in Data.weapons.weapon_axe.who){
    Debug.Log("Person: "+person);
}
```

Important: Sometimes when referencing other sheet classes from other tabs it can cause the C# to return some null values. This is caused by a deterministic initialization order of static members inside C# and causes some values to have their default values - which in this case are null values.

This can be fixed by re-arranging the tabs in your sheet in a particular order and not referencing oneself. Tabs with referenced elements should go to the left followed by tabs that reference other previous tabs each after another sorted by their dependencies.

PRO Base Classes

A base class is a class that Meta Sheets will extend on a particular generated sheet code. Just like in C# and any OOP languages: base classes allow you to extend Meta Sheets generated code from the base and add for example additional variables, methods, getters, settings and alike.

Let's take this simple class for example as our base class

```
public class Base {  
    public enum eGender {  
        male,  
        female  
    }  
  
    public string name;  
    public int age;  
    public eGender type;  
  
    public void LogInfo(){  
        Debug.Log(string.Format("name:{0}, age:{1}, gender:{2}", name, age, type.ToString()));  
    }  
}
```

By writing this **{sheetName}:{baseClassName}** for the sheet name we can tell MetaSheets to extend the data of that sheet on our base class.

| | A | B | C | D | |
|------------------|------------|-------|-----|--------|--|
| 1 | id | name | age | gender | |
| 2 | user_john | John | 35 | male | |
| 3 | user_maria | Maria | 29 | female | |
| 4 | user_alex | Alex | 21 | male | |
| | | | | | |
| + ≡ users:Base ▾ | | | | | |

All public variables that match between the base class and the spreadsheet will be used and referenced in the generated data class. Variables that can not be found in the base class will be added dynamically to the data class that Meta Sheets generates.

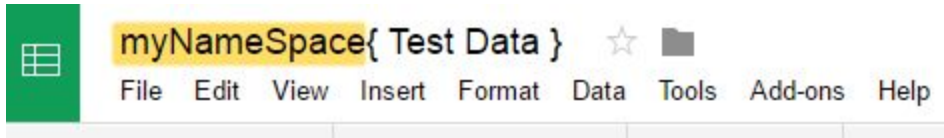
This technique allows to access any of the base class features directly from a Meta Sheets data item including methods. For example:

```
//Access base class features  
Data.users.user_john.LogInfo();
```

To get more ideas as to for what this is useful for head over to the [Blog](#) or have a look at the MetaSheets bundled examples.

Custom Namespace

Namespaces are optional and allow you to nest the generated data into a custom namespace by changing the title of your sheet to the following convention:



This will generate the sheet data in a namespace of '**myNameSpace**'. The namespace element will go in front of the curly brackets and the title must include the '{' and '}' characters around the document title.

With a namespace you may need to include it to your code when accessing the data, for example:

```
//Importing the namespace to the header of your C# Document
using myNameSpace;

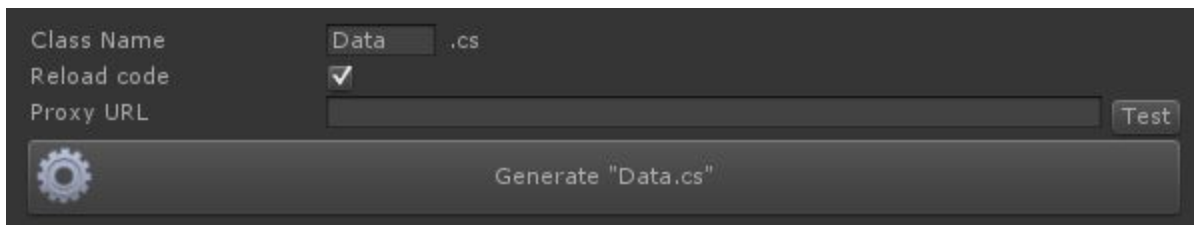
...
Debug.Log("Data: "+ Data.sheetName.Length );
//Accessing it without importing
Debug.Log("Data: "+ myNameSpace.Data.sheetName.Length );
```

Namespaces are useful when you want to name different sets of Data with the same name. By adding namespaces you can better organise more complex projects.

PRO Reload at runtime

With the pro version of MetaSheets you can reload changes from the spreadsheet at runtime- even on your target platform like Android, iOS or the web player.

Inside Meta Sheets Window make sure you enable the "Reload code" tick which will generate the runtime reload code inside your data class.



From within anywhere in your code you can then call your data **Reload()** method.

```
//Reload Data
Data.Reload();
```


You can also pass a `System.Action()` to get signal when the data has been loaded.

```
//Reload Data with a signal  
Data.Reload(OnDataLoaded);  
void OnDataLoaded(){  
    Debug.Log("Data has been loaded");  
}
```

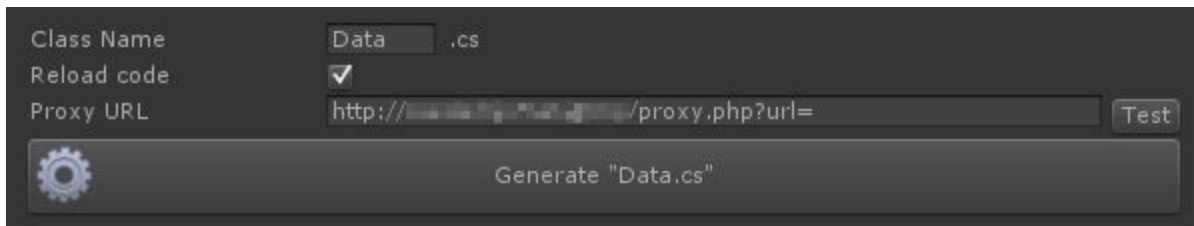
Sometimes your spreadsheet might contain a lot of individual sheets. It is often faster to load just individual sheets instead. The same options are available when loading individual sheets.

```
//Reload individual sheets  
Data.sheetA.Reload();  
Data.sheetB.Reload();
```

It's not recommended to use the reload functionality for a production build of your project mainly because of two reasons: Your runtime could break because of altered spreadsheet data that is not compatible anymore and because a lot of users might hit your document that you published causing a lot of traffic on Google's free service.

PRO Using a proxy server

Certain target platforms such as the Unity plugin or WebGL require cross domain permissions from the Google hosted domains which are not available. In order to still reload on these restricted platforms one can bypass the loading requests through a proxy server.

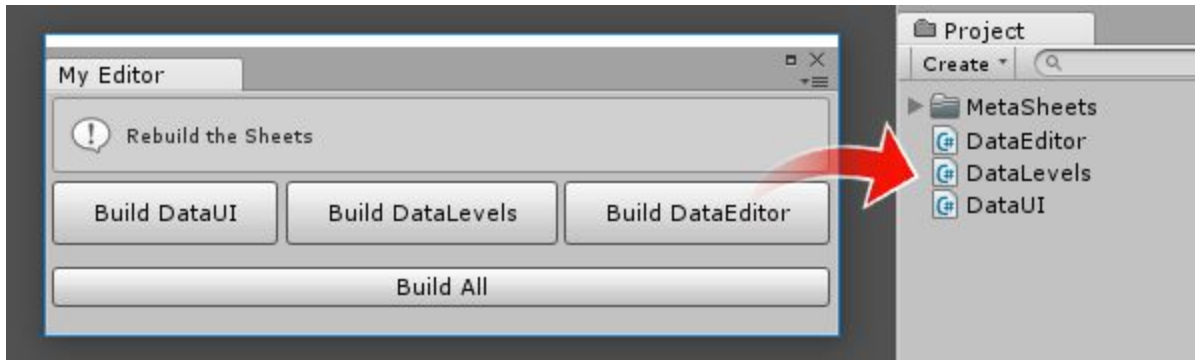


The screenshot shows the Meta Sheets Unity plugin interface. It has a dark theme. At the top, there are three fields: 'Class Name' with the value 'Data' and a '.cs' suffix, 'Reload code' with a checked checkbox, and 'Proxy URL' with the value 'http://.../proxy.php?url='. To the right of the Proxy URL field is a 'Test' button. Below these fields is a large button with a gear icon on the left and the text 'Generate "Data.cs"' in the center.

Meta Sheets provides an example php proxy script that you can host on your server (together with a valid crossdomain policy). Paste the proxy script URL in the Meta Sheets proxy URL field including the '?url=' argument to have all data to be loaded via the proxy.

Editor Scripts

A great way of using MetaSheets is by integrating it into other Editor scripts. You can even access the generated Data in Editor UI's and scripts to automate tasks, unit tests, build settings, etc.



To build a data class from an Editor script you can use the Build method which is part of the MetaSheets main class.

//Build a C# Class from code

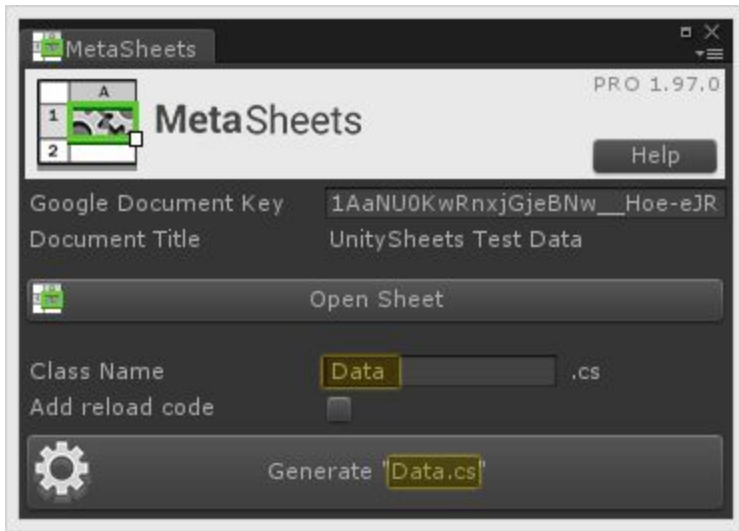
```
MetaSheets.MetaSheets.Build("Data", "1OjHL4ZUSInfkV207WNKi1tHHhqITjUSrdyHJrQpSrl", true);
```

Parameters

| | |
|--------------------------|--|
| className | The name of the C# Class name, e.g. "Data" |
| documentID | The document ID of the spreadsheet. |
| PRO addReloadCode | When true will add additional Reload capabilities to the Data code |

{Data} Class

This is your entry point to the spreadsheet data. The name of the class is defined in the MetaSheets window under 'Class Name'.



Exposes

{Sheet} A static variable that returns the sheet data.

PRO Reload A static method that reloads the data at runtime. An optional System.Action call triggers when it finished loading.

Example Code

```
//Access the spreadsheet Data
Debug.Log(Data);
```


```
//Reload all sheets and log when done loading. The System.Action Parameter is optional
Data.Reload( ()=>{
    Debug.Log("Done loading all sheets");
});
```

{Sheet} Class

This class represents a particular sheet of the spreadsheet. It extends the IEnumerable and behaves just like an array or dictionary. Think of it as the sheet tab of your spreadsheet.



Exposes

| | |
|---|--|
| updated | A System.DateTime object that reflects when this data was last changed. |
| labels | A string[] array with all of the {property} labels of the sheet. |
| Length | The length of this sheet and how many items are stored in here. |
| this [string id] | A bracket accessor to access elements of this sheet by the string id. Important: This is only compiled into the code when your first column of the sheet contains string values. |
| this [int index] | A bracket accessor to access elements by their index. |
| ToArray() | Returns an array of the {Sheet} elements. |
| Random() | Returns a random item of the {Sheet}. |
|  Reload() | Reloads the particular sheet data at runtime. An optional System.Action call triggers when it finished loading. |
| ContainsKey (string key) | Checks if a particular string key exists. Important: This is only compiled into the code when your first column of the sheet contains string values. |
| {item} | Specific static items that are defined if the first column of the sheet consists of strings. Important: This is only compiled into the code when your first column of the sheet contains string values. |

Example Code

```
//Basic info
Debug.Log("Sheet1 length: " + Data.Sheet1.Length);
Debug.Log("Item types: " + Data.Sheet1[0].GetType() );
Debug.Log("ContainsKey('apple')? " + Data.Sheet1.ContainsKey("apple"));

//Accessing items
Debug.Log("Random item: " + Data.Sheet1.Random());
Debug.Log("First item: " + Data.Sheet1.ToArray().First()); //using System.Linq
Debug.Log("Specific item: " + Data.Sheet1.apple);
Debug.Log("Array accessed item: " + Data.Sheet1[0]);
Debug.Log("Key accessed item: " + Data.Sheet1["apple"]);

//Foreach loop access
foreach(DataBasicsTypes.Sheet1 item in Data.Sheet1) {
    Debug.Log("Foreach: " + item.id);
}

//for i++ loop access
for (int i = 0; i < Data.Sheet1.Length; i++) {
    Debug.Log("for i++: ["+i+"] = "+Data.Sheet1[i]);
}

//Reload a specific sheet only
Data.Sheet1.Reload(()=>{
    Debug.Log("Done loading Sheet1");
});

//Access specific item
Debug.Log("Sheet1 item 'apple': "+Data.Sheet1.apple);
```

{Object} Class

This class represents a single object item from a sheet (e.g. a single row from a spreadsheet). With MetaSheets Pro you can extend this class to a custom base class.

Exposes

{variable} A particular variable of the sheet. These represent the column headers in your sheet. Where each column is reflected as its own variable of each row item.

Example Code

```
//Access a property 'age' from apple in Sheet1 of Data
Debug.Log("The age of the apple is: "+Data.Sheet1.apple.age);
```