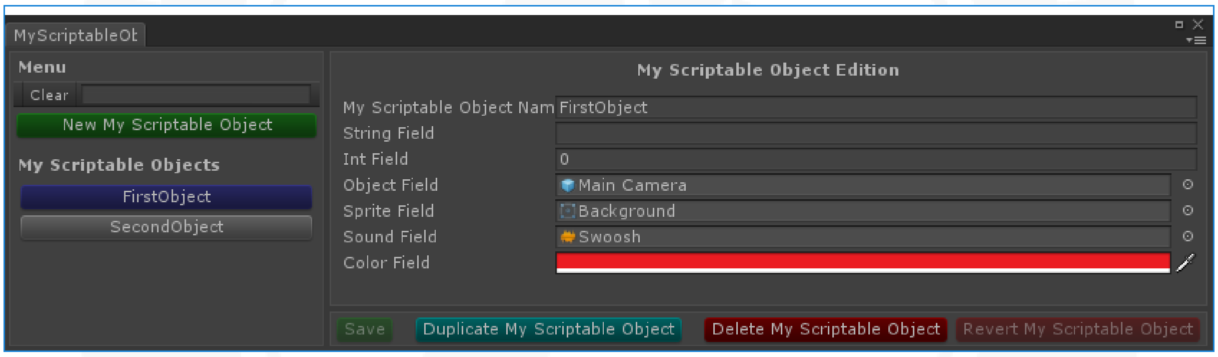
Scriptable Object Suite Documentation

1. What is Scriptable Object Suite?

Scriptable Objects are powerful Unity classes that help you **serialize** static game data (Skins, Level Data...)

They use **Unity's serialization system** so that they can be easily edited **directly inside Unity Editor**. Managing them can be painful without a proper toolchain.

Scriptable Object Suite is a tool that helps you manage, create, edit, delete, and duplicate your Scriptable Objects, in automatically generated windows:


MyScriptableOt
Menu
Clear
New My Scriptable Object
My Scriptable Objects
FirstObject
SecondObject
My Scriptable Object Edition
Save
Duplicate My Scriptable Object
Delete My Scriptable Object
Revert My Scriptable Object

My Scriptable Object Nam FirstObject
String Field
Int Field 0
Object Field Main Camera
Sprite Field Background
Sound Field Swoosh
Color Field 🟥

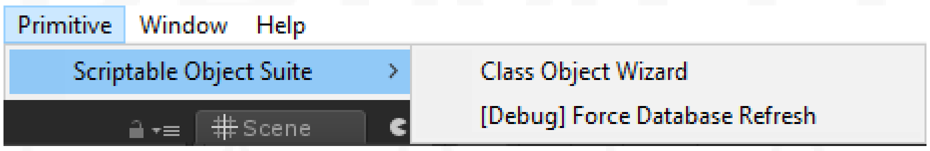
It supports any type Unity can serialize, from simple types to references such as GameObjects, Sounds, Sprites, etc.

Scriptable Object Suite also supports **Load on Demand** features to control loading time and manage memory.

2. Basic Use

Importing the package

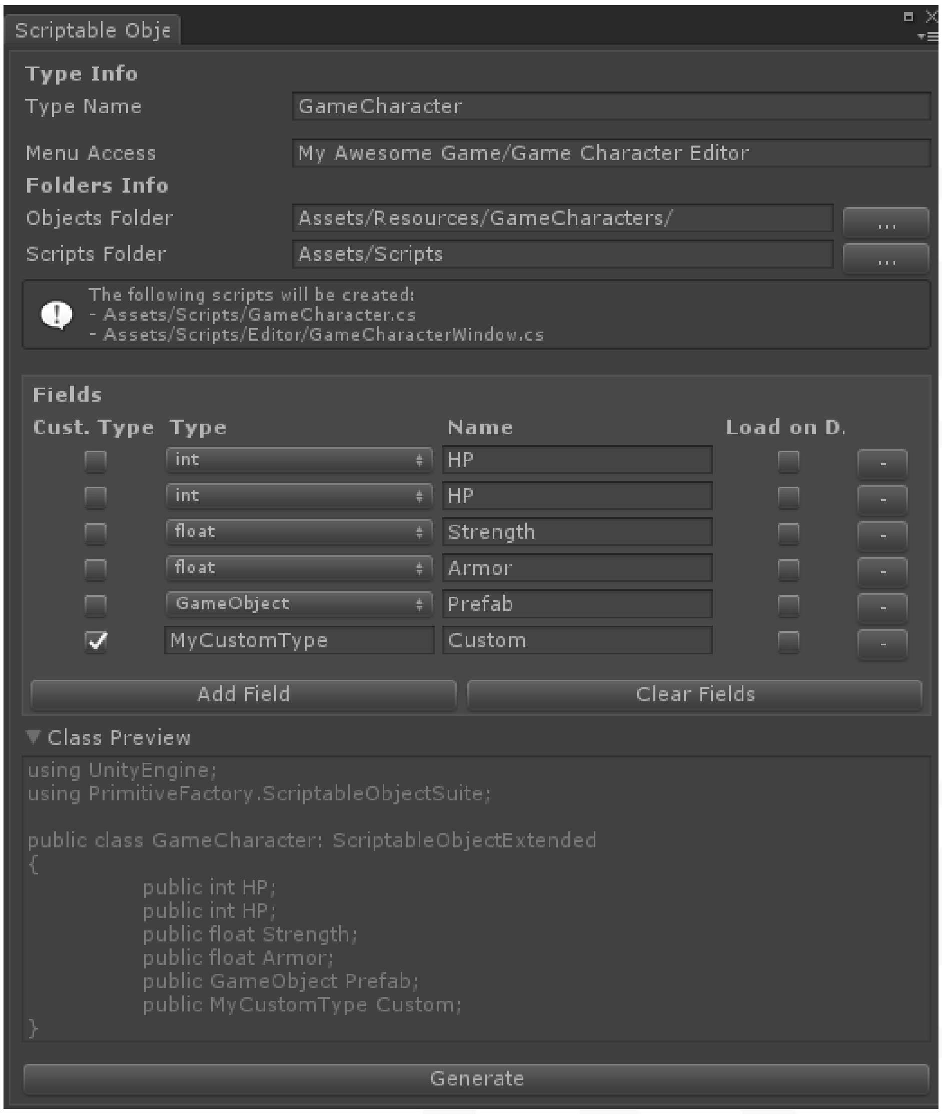
Get the package directly on the Unity Store. Import your package inside your project. Nothing more, you're good to go!
If everything went fine, you should have a new menu in the Unity top menu:



You're now ready to use Scriptable Object Suite!

Using the Wizard

In the top menu, go to Primitive > Scriptable Object Suite > Class Object Wizard :



Scriptable Obje

Type Info

Type Name: GameCharacter

Menu Access: My Awesome Game/Game Character Editor

Folders Info

Objects Folder: Assets/Resources/GameCharacters/

Scripts Folder: Assets/Scripts

The following scripts will be created:
- Assets/Scripts/GameCharacter.cs
- Assets/Scripts/Editor/GameCharacterWindow.cs

Fields

Cust. Type	Type	Name	Load on D.	
☐	int	HP	☐	-
☐	int	HP	☐	-
☐	float	Strength	☐	-
☐	float	Armor	☐	-
☐	GameObject	Prefab	☐	-
✓	MyCustomType	Custom	☐	-

```
using UnityEngine;
using PrimitiveFactory.ScriptableObjectSuite;

public class GameCharacter: ScriptableObjectExtended
{
		public int HP;
		public int HP;
		public float Strength;
		public float Armor;
		public GameObject Prefab;
		public MyCustomType Custom;
}
```

Let's go over each field in detail:

- **Type Name**: Name of your class
- **Menu Access**: Defines the path in the Unity top menu to access the window in which you will edit the data
- **Objects Folder**: Relative path to your project root where your different objects of this type will be stored
- **Scripts Folder**: Relative path to your project root where the classes will be generated
- **Fields**: This is where you define each field that you type contains
  - Click the Add Field button to add a new field
  - For each field, input the type of your field and the name of your field.
    The type can be selected inside the dropdown list. If you need another type not present in this list, check the "Cust Type" toggle. The Load on Demand toggle is defined in the advanced use section, later in the documentation
  - You can remove the field using the "-" button
  - Click the Clear Fields button to remove all the fields
- **Class Preview**: You can open this to check the class that will be generated

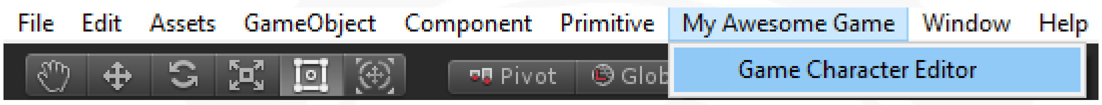
Hit the "Generate" button once you're done. This will create 2 files:

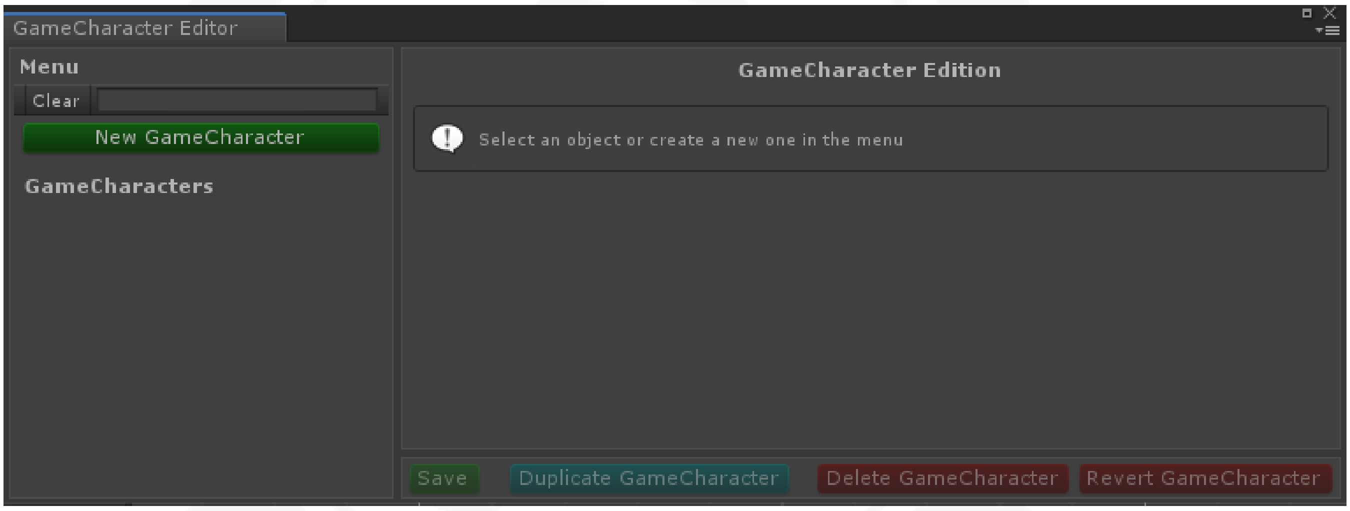
- A class that contains the definition of a new type with all the fields you've defined above
- An editor class that allows you to open a window to create and edit objects of that type

---

Creating and editing objects

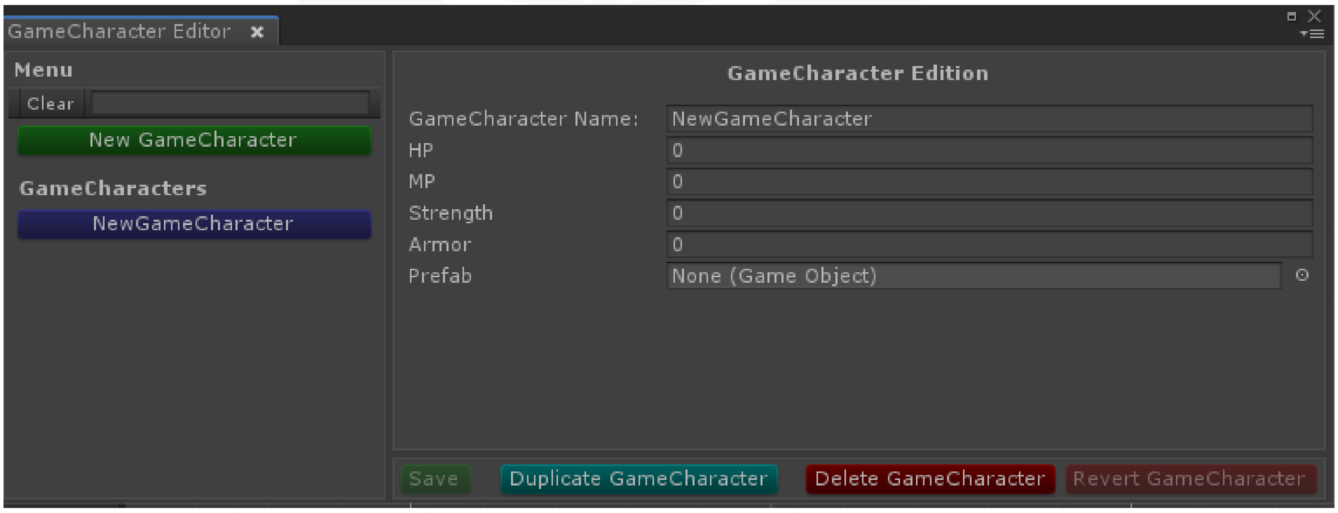
- You can now open the edition window using the Menu Access you've defined in the wizard:





Select an object or create a new one in the menu

- You can create a new object using the "New <YourType>" button:

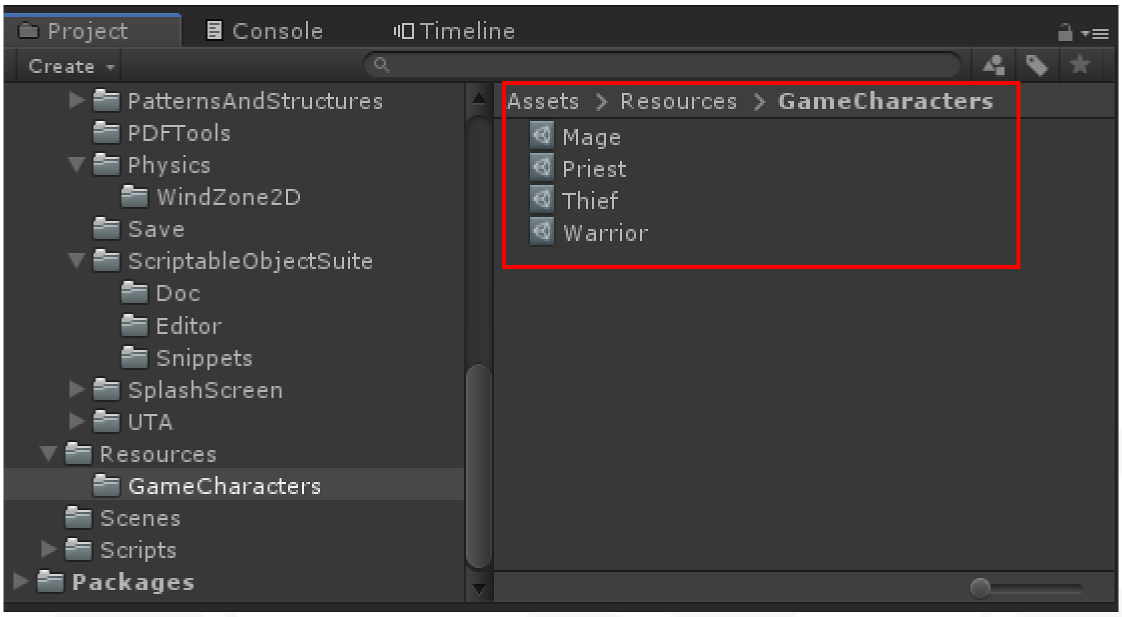


GameCharacter Name: NewGameCharacter
HP 0
MP 0
Strength 0
Armor 0
Prefab None (Game Object)

- You can now edit all the data for this object
    - Save your data using the Save button
    - Duplicate or Delete your object using the Duplicate / Delete buttons
    - The Revert button will revert your object to its values before your last Save

- Use the filter field above the "New <YourType>" button to filter your objects by name

- Objects are stored inside the directory you defined in the wizard:



Manually editing your classes

- There's no dark magic behind Scriptable Object Suite, you can simply edit you classes after they've been generated by the wizard. Go to the Scripts Directory you defined in the wizard to find your object class. Here's the example of our GameCharacter class we created in the previous chapter:

```
public class GameCharacter: ScriptableObjectExtended
{
        public int HP;
        public int MP;
        public float Strength;
        public float Armor;
        public GameObject Prefab;
}
```

- Feel free to add / edit your fields at will
    - Warning: Changing a field type of name will erase all data regarding this field on all objects of this type

3. Advanced use

Bypassing the Wizard

The Wizard is here to help non-programmers get a hang of the suite. You are not forced to use it: once you've got a grasp on how everything works, feel free to simply create your own classes.

Once the Wizard has created the files, you are free to edit them.

Using more complex types

- **"Using" clauses**: the wizard only adds the UnityEngine namespace to your scriptable object type. You are free to add any other namespace that you might require to compose your object:
    - System.Collections.Generic;
    - UnityEngine.UI
    - etc.
- **Custom methods**: define any method you would like in the file, without any restriction
- **Classes / Structs**: as long as your field type is tagged with the [System.Serializable] attribute, you can add a field of this type inside your object. For instance, the following type is completely valid as a field for your object:

```
[Serializable]
public class UnlockCondition
{
    public UnlockConditionType ConditionType;
    public int ConditionParameter1;
    public int ConditionParameter2;
}
```

Load on demand / Managing your memory

- By default, Unity cascade loads any serialized object.
  Let's take an example: I'm designing a card game; each card is a scriptable object with the following type:

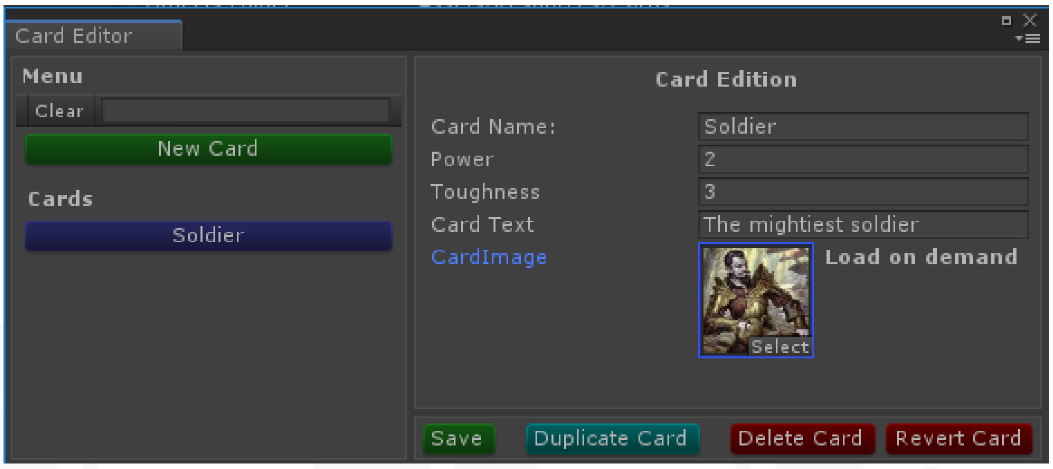
```
public class Card: ScriptableObjectExtended
{
        public int Power;
        public int Toughness;
        public string CardText;
        public Texture2D CardImage;
}
```

- To avoid this issue, Scriptable Object Suite allows you to have fields load only when you want to. Here's how it works:
    - Put all of your card textures in a folder inside a Resource folder
    - Use a [LoadOnDemand] attribute trick to specify that a field will not be loaded automatically:

```
[NonSerialized]
public Texture2D CardImage;
[SerializeField][LoadOnDemand("CardImage")]
private LoadOnDemandInfo m_CardImageLoDInfo;
```

        - The CardImage field is tagged as NonSerialized, to avoid the cascade loading
        - Add a LoadOnDemandInfo field that is serialized and will hold info as to where the asset is located. The LoadOnDemand string parameter "CardImage" links this path to the actual CardImage Texture2D field

- A slight change appears on the edition window: a "Load on demand" tag appears on the CardImage field, telling that this field will not be loaded by default:



Card Editor

Menu

Card Edition

Card Name: Soldier

Power 2

Toughness 3

Card Text The mightiest soldier

CardImage Load on demand

- Any type of asset can be loaded on demand, as long as it's located **inside a resource folder**

Triggering a load / unload action

- Once you have a reference on a ScriptableObjectExtended object, you can call the **Load** method to load Load on Demand fields:

```
List<Card> m_CardDatabase;
Card myCard = m_CardDatabase[3];
myCard.Load();
```

- The **Load** method takes two optional parameters:

```
public void Load(string fieldName = null, bool async = false);
```

  - The ***fieldName*** parameter allows you to load only a single field of your object, if it contains multiple Load on Demand fields. Set to null to load all fields.
  - The ***async*** parameter allows you to asynchronously load your object. By default, the loading is synchronous

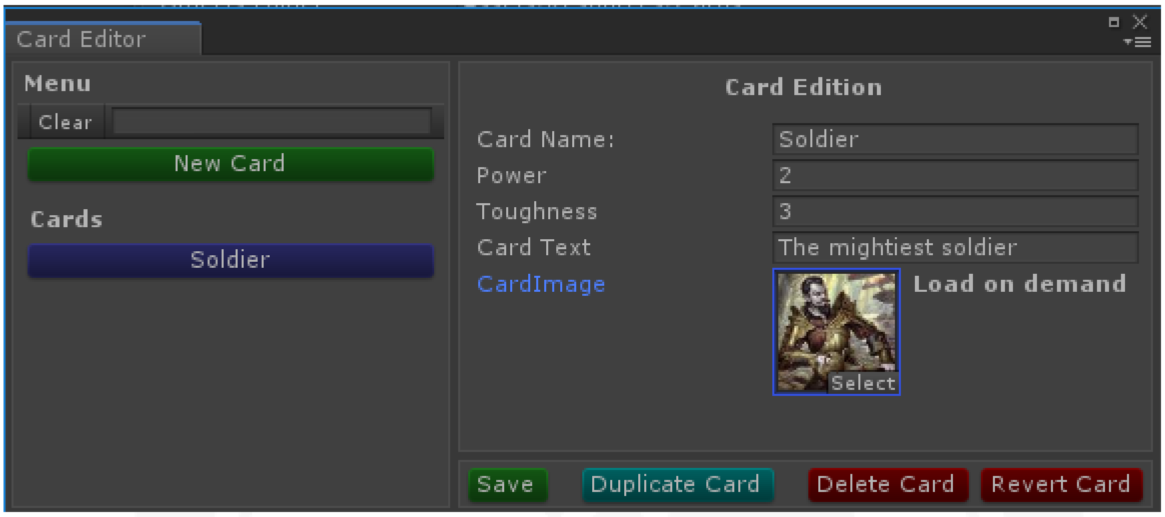
- Similarly, an **Unload** method exists to unload Load on Demand fields:

```
public void Unload(string fieldName = null);
```

  - The ***fieldName*** parameter allows you to unload only a single field of your object, if it contains multiple Load on Demand fields. Set to null to unload all fields
  - Unload actions are always synchronous

Customizing the Editor Window

This is the default look of your editor window:



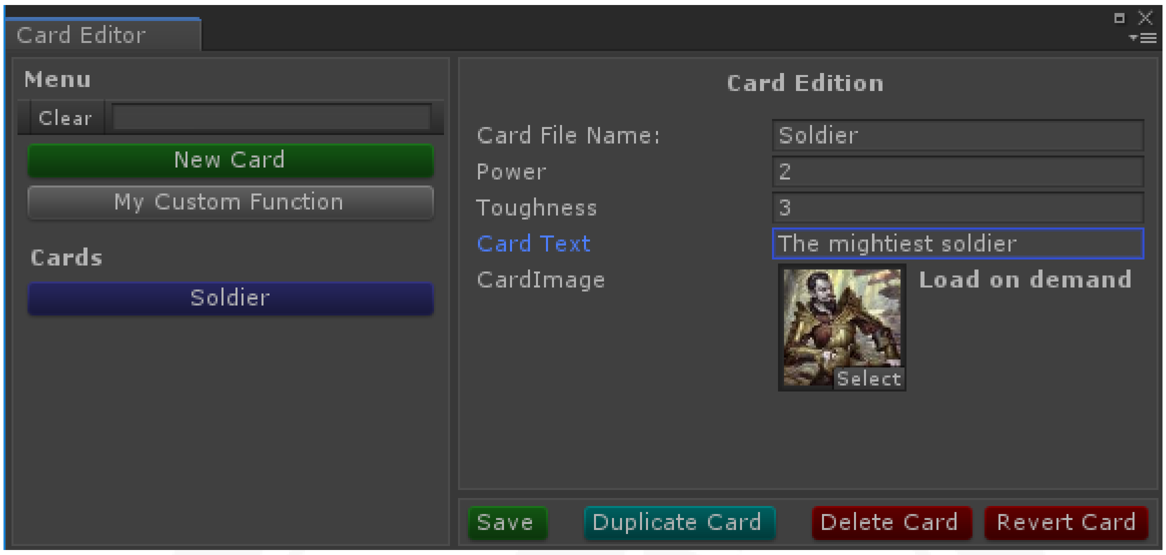
Soldier
2
3
The mightiest soldier

You can customize the object selection / creation / deletion behaviour by overriding the following methods:

```
// Called when a new object is created, obj parameter is the new object
protected override void OnObjectNew(ObjectType obj);

// Called when an object is duplicated, obj parameter is the new object
protected override void OnObjectDuplicated(ObjectType obj);

// Called when an object is saved, obj parameter is the saved object
protected override void OnObjectSaved(ObjectType obj);

// Called when an object is deleted, obj parameter is the deleted object
protected override void OnObjectDeleted(ObjectType obj);

// Called when the selection changes, oldObj parameter is the previous object
(can be null), newObj is the new object
protected override void OnSelectionChanged(ObjectType oldObj, ObjectType newObj);
```

You can also override the **DrawCustomFunctions** method to add custom actions below the "New <YourType>" button. For example:

```
protected override void DrawCustomFunctions()
{
    DrawCustomFunction("My Custom Function", CustomAction);
}

private void CustomAction()
{
    Debug.Log("This is my custom action!")
}
```



Card File Name: Soldier
Power 2
Toughness 3
Card Text The mightiest soldier
CardImage

Finally, you can customize the core draw method of your window by overriding the DrawEditor method of your window:

```
protected override void DrawEditor(Card target)
{
    base.DrawEditor(target);

    // Do something else
}
```

4. Support

- Unity Download Link
- Customer Mail Support
- Online documentation