“Advanced Editor V0.1” for Databases - Unity (English).

Summary

1.Introduction. What is its purpose?

This project originated from the need to reduce the development time for entities editor windows. Before, I spent a lot of time to develop the editors.

Previously, for each window I needed 1 to 3 hours to create, depending on its complexity, including extensive testing. To streamline this process, I created an editor capable of adapting to all entities, saving time in the long run.

Before, each entity required five separate files for editing, displaying, creating, listing, and managing the database. With the Advanced Editor, this complexity is reduced to **just creating the necessary database files.**

Now, you can easily configure the data you want to display using the "Editor" property for each entity, which automates the window creation process.

The Advanced Editor **currently supports** basic data types like integers, enums, strings, Unity objects such as Sprites, popups, arrays, and dictionaries. **Future plans include methods for creating tables and their own methods and windows**

2.How it works

2.1.Editor

I used C# Reflection methods. It takes the properties defined in **“Editor” property** of each entity.

The Advanced Editor is made up with 4 files in the folder Assets>Scripts>Windows>Core.

Each file corresponds to:

- ListWindow: **List the elements** of the DB.
- WindowAction: An **Enum with the actions** that the editor can perform.
- Window: It executes when you press the element for creating, modifying or displaying. **It generate the window** with the objective of performing the action chosen in WindowAction.
- WindowInterface: The v**iew of the windows**, after the window is managed in “Window”.
- Display: It will be used in “WindowInterface” to **display all data that aren’t** common to all entities and that must adapt to entity. For example, **It won’t create a “Confirm” button;** “WindowInterface” will handle that.
- AdvancedEditor: This is where all functions for **the editor’s fields are located**.

2.2.DataBase.

Additionally, in Assets>Scripts>Windows, there is a file named “Windows”, where **entities with editor must be defined.**

There is a class called “CoreDB<T>” located in Assets>Scripts>Data>Database>Core. This class **creates the DB.** You need to create a new file named “EntityDB.cs” with a class of the same name. It should be capable of serialization into a .asset file, and include “CreateAssetMenu” annotation. The class of editor must extend “CoreDB<Entity>”.

2.3.Others

There are some classes, like **SerializableDictionary(it doesn’t created by me)** that can be useful. This class allows you to create serializable dictionaries that do not exits in Unity. It means that normal dictionary won’t be saved in a database.

3.How to use

3.1.Make a new Entity and DB

To create an entity and DB, you must create, a new file for each entity and DB. The entity needs **to extend “BaseEntity.cs”**, it has a ID and name fields. The DB files **must extend “CoreDB<T>.cs”** where T is the name of entity that extends BaseEntity. This class contains everything needed to manage the data. The entity must should have **a constructor for creating a new instance with an ID and another for cloning.**

In summary, create a T.cs file for the entity T, which extends BaseEntity, and create a “Ts.cs” file for the DB Ts, extending CoreDB<T>.

3.1.1.Entity

Create a new file with class. Ensure that all fields are serializable, as in the class. Create a constructor for new item, including an ID, and **another for cloning**. The entity must have an “Editor” property where the configuration of editor windows is set.aqñ

3.1.2.CoreDB – DataBase

Create a new file with class and extend it with CoreDB<T>. It must include the [CreateAssetMenu(menuName=”Database/Entities”)] annotation to enable the creation of a new .asset file by right-clicking in the Database>Entities section.

4.Using the Editor

4.1.Make Windows for the editor

Create a new class in Windows>Windows.cs **(all Windows classes can be in the same file).**

Extend the class ListWindow<Entity, CoreDB<Entity>> and add the **annotation** [CustomEditor(typeof(CoreDB<T>))]. This step generates the windows, but they only contain fields for ID, Name and Confirm button.

4.2.“Editor” property in entity.

Finally, all setting **for editing** the data, are found here. Overwrite the “Editor” property of BaseEntity in the new entity.

This property defines the properties to be displayed, along with their constraints and attributes. “Editor” is of type “EditorParams” in the Utils folder.

EditorParams has a constructor: EditorParams(**parametters**, configurations, labels, columns, maxV, maxH, minV, minH, extensionOf). Only “parametters” is mandatory.

- Parametters: An array of strings used to set **names of properties** (recommendable to be get-set properties), that can edit edited in the editor.
- Configurations: A Dictionary<parameter/property, array with configuration>. The key represents the parameter with restrictions, while the value is an array that specifies those restrictions. For example, {“ID”, new[]{“readOnly”}} sets the ID field as readonly in all cases. I recommend using “AdvancedEditor.Restrictions” for basic restrictions; these values are strings with exact keyswords, and the restriction checks utilize them. Additionally, there are restrictions that uses Tuples, typically related to restrictions that require data, such as size limits on lists or selection lists that aren’t enums. You should use the “EditorParams” methods to create the Tuples.
- Labels: Dictionary<parameter, Label>, how **the name of parameter** will be displayed.
- Columns: **Not implemented.** It will make columns.
- Max/Min: **Max and Min sizes** (Vertical and Horizontal) for the editor window.
- ExtensionOf: **Adds all attributes** and properties from another field “EditorParam”. It’s recommended to use it with base.Editor to include the parent’s property fields.

If this property is well defined, it should work.

5.Annotations

- There is an example called “EditorExample”.
- AdvancedEditor functionalities **can be used outside the central system;** they are public methods and have descriptions.
- The minimum in the dictionary field in AdvancedEditor only works if the size is larger than the minimum at least once.

- The editor is under development, so there may be bugs. Please, **notify me of any errors** that you find. Thanks you.
- Please, c**<u>redit me as creator of the code</u>** if you use the editor. Provide the Git link and a contact. Thanks you.

6.Links

- SerializableDictionary: https://discussions.unity.com/t/solved-how-to-serialize-dictionary-with-unity-serialization-system/71474/4