Save Manager Documentation

Summary

Contributors	Valid for Version	Last Updated
Jonathan Carter	1.1.0 & Higher	24/07/2021

Contents

Package Information

The package has 6 main folders & 10 files, all listed with an asterisk and coloured **green** are required files for the asset to work. Those without are in red are not needed for the functionality but are required for some cosmetic features or provide examples of how to use the asset..

- **Editor/Carter Games/Save Manager**
  - ***DataTypeHelper.cs**
  - ***SaveDataEditor.cs**
- **Recourses/Carter Games/Save Manager**
  - LogoSM.png
- **Scenes/Carter Games/Save Manager/Example**
  - SaveManagerExampleScene.unity
- **Scripts/Save**
  - ***SaveData.cs**

> This should be deleted by you when you have generated your save data class. We have to provide one to avoid errors with Unity.

- **Scripts/Carter Games/Save Manager**
  - ***SaveExtraTypes.cs**
  - ***SaveManager.cs**
- **Scripts/Carter Games/Save Manager/Example**
  - SaveManagerExample.cs

Change Log: Shows the changes from previous versions of the asset.

Docs: An offline copy of this pdf.

Supported Platforms

The save manager supports all platforms that support binary file saving. However, we **DO NOT** support **WebGL** saving in this asset. This is purely due to the end user needing to have a webserver to host parts of the asset as well as it being rather complex for new users, which this asset is more geared towards as a target audience.

Installing the asset

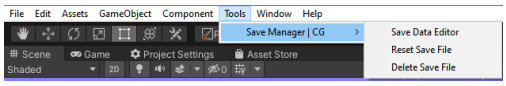
Installing the asset is super simple. If your using an older version of Unity you will be able to get the asset from the asset store within Unity. However if your on a newer version then the asset can be installed via the package manager under the **My Assets** section. Once imported into the project, you'll be ready to use the asset.

Setup & First Use

The setup for the asset is very straight forward as there is a handy editor tool to do all the hard work for you. The tool assists in the creation of the **SaveData.cs** file which a blank version of is provided with the asset to avoid errors in the package. Now you can just write in the file yourself if you know what you are doing and put in your save types.

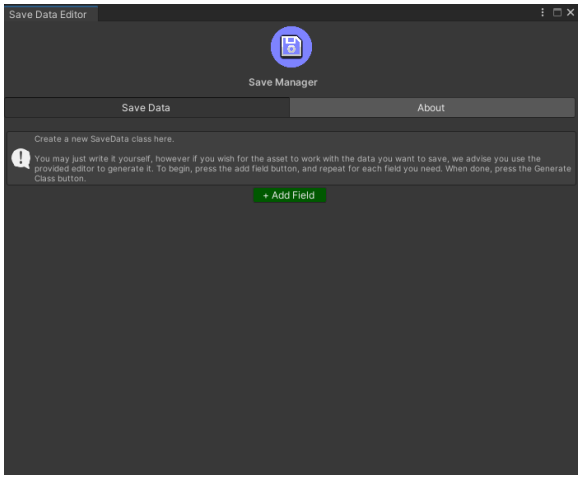
> ⚠ Editing the class yourself has the risk of breaking the editor tool, we have tried to catch all issues, but you may experience issues by editing the class yourself.

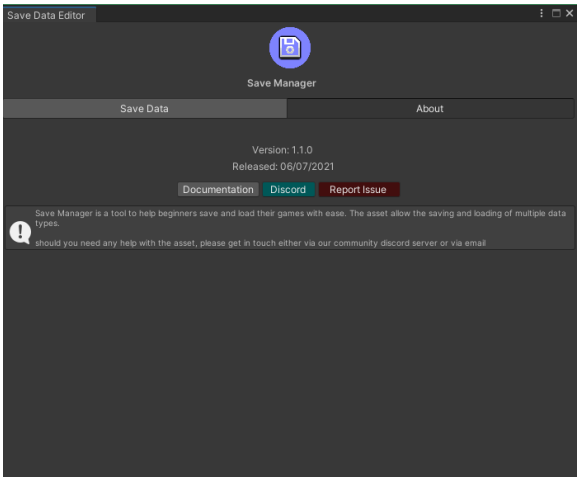
Accessing the editor tool



To access the editor tool, all you have to do is navigate to the **tools** tab on the navigation bar at the top of the Unity window and enter the **Save Manager | CG** tab and press the **Save Data Editor** option. Once pressed an editor window will popup which can be moved, resized and docked as you would with any other tab like the inspector, hierarchy and game view tabs for instance.

How to use the editor tool

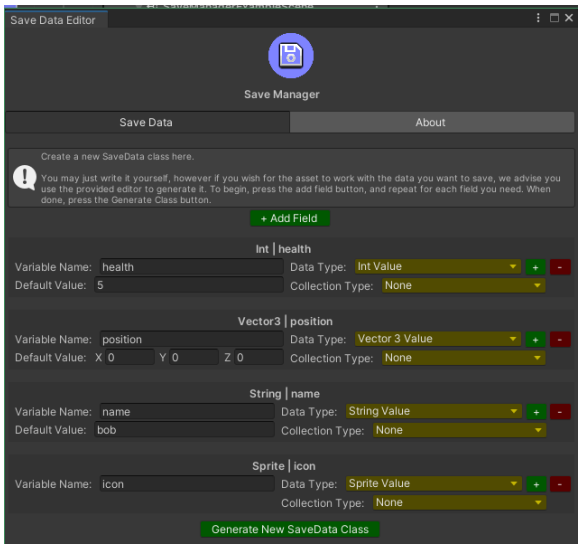




The editor tool comprises of 2 main tabs, the first one allows you to create and edit the **Save Data** class with and the second display information about the asset as well as useful links relevant to the asset. When opening the tool, if you already have an existing Save Data setup which is valid, the editor will update to show the current values for you to adjust and edit. Note that if you edit the class outside of the tool and then open the editor tool, you may find some values will not read correctly.

Creating a save file with the tool

To create a save data class you'll need to setup for variables for the class to contain. To start you'll need to press the add field button to add your first variable.



Creating a variable

Once you have added a field, you be presented with a grouping of fields to edit for this variable like what you can see on the left here. There are 4 options that the field will show by default, but this varies on the setup. Each grouping will always have:

- Variable Name - Which is what the variable will be called.
- Data Type - Which is the data type the variable is defined as.
- Collection Type - Which defines whether this variable is a collection like a list, array, queue, stack or if it is a normal single variable.

Other Options

If the data type is a class value then an extra field will appear for the class name, which is where you'd but the name of the class you want the variable to be of. This field is CaSe SeNiStIvE!

If you have the grouping with no collection type then a field will be available for a default value. This can be left blank or have a value that is valid for the type. The field will adjust based on the data type you have selected.

> ⚠ Note: Sprites & Classes can't have a default value.

You can keep adding more variables with the + Add Field button or the **green + button** next to each grouping and remove a grouping with the **red - button.**

Generating the class for use

Once you have filled in the variables you want to save, you can press the **Generate New SaveData Class button** to make the class with the inputs you gave. At this point the editor will refresh with the class will be placed in the following directory in your project, please delete the SaveData.cs class provided in asset package if you haven't already at this point as it is no longer needed:

> Assets/Scripts/Save/SaveData.cs

What setups are valid?

Save Variants

If your making your save data class yourself or editing the code generated after using the tool there are some important steps to follow to keep the save data valid. You'll note that when you save any Vectors, Colors, Quaternions & Sprites that their datatype has the prefix of **Save** before it. This is out custom struct for saving these types of data. Previously you would've had to use these types in your code, however since 1.1.0 you can now just use the normal version and use the implicit operator to convert them to and from the save variants. So if you are writing your own class for the save manager to use you will need to use the save variants for these types which are listed below for reference:

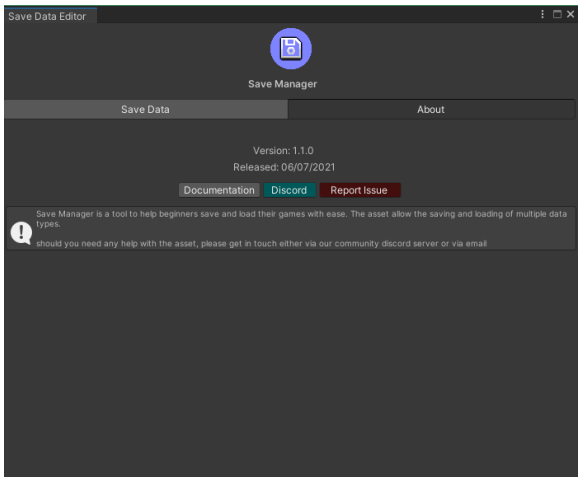
- SaveVector2
- SaveVector3
- SaveVector4
- SaveColor
- SaveQuaternion
- SaveSprite

Exposure & Default Values

Every field you add must be public for the tool to work correctly, you'll want this too as there isn't much point of having private values in a save file. The only other issue you'll find is with default values. By default we only support default values when defining values as a single variable and not a collection (list, array etc.). We also don't support default values for sprites and classes. This is mostly due to the amount of backend work that would be need to make it work and the lack of instances where you'd want to use it. If you want to edit the default values outside of the editor you'll have to note that they should be written in their raw form, for example you might use **Vector2.One** as the default value. This won't work with our editor, so you should write it as **new Vector2(1f,1f)** instead.

Extra Setup For Saving Sprites

To save sprites you need to have you sprite asset import settings setup up in a particular way. Thankfully its nice and simple, in the import settings you need to enable the read/write toggle as well as not using any compression on the sprite you plan to save. Because of the lack of compression we advice you to use Power of 2 (256×256, 512×512, 1024×1024 etc) where possible and compress the art outside of Unity to help mitigate this issue if art file size is a problem for your product.



The About Tab

The about tab provides a little more info on the asset as well as the version number, release data and some useful links for the asset. These include a button to this documentation page, a button to open the invite to our discord server and a button to our report form to report issues with the asset and any of our other products to us.

Methods & Properties

Namespace

All code for this asset is under the following namespace, when using the manager you will need to add this namespace to your using defines.

```
// Asset namespace
CarterGames.Assets.SaveManager

// Using line for the asset
using CarterGames.Assets.SaveManager
```

Save Manager Methods

Below are all the public methods and properties that are provided in the save manager asset that you use. Note that some are excluded as they are only for the editor side of the asset to work.

```
// Method - Saves the data provided into the method into the save file....
SaveGame(SaveData data);

// Method - Loads the game data from the file or returns a warning if not data was found...
```

```
// Returns: SaveData containing and save data that was in the save file...
LoadGame();

// Method - Checks to see if a save file exists and returns it for use...
// Returns:Bool
HasSaveFile();

// Method - Resets the save file to the default values...
ResetSaveFile();

// Method - Deletes the current save file...
DeleteSaveFile();
```

Saving & Loading My Game

Saving and loading your game is somewhat subjective as each game requires a different setup. However, most games will want to load their data on the start of the game, edit it when levels are complete and save it when progress is made. We recommend having a static or persistent object that holds a copy of the save data class as a variable which can be edited on the fly and using in the saving and loading process. Below are some examples of how to go about each stage. These can also be seen in the example script in the asset which is used in the

Loading

Here we're assuming that you have an empty GameObject or similar which holds this script to load the game when the scene starts. This is one of the better ways of loading the game if you need the data in each scene.

```
// Gets the game save from the file....
SaveData loadData = SaveManager.LoadGame();

// Applies the values to the
player.name = loadData.playerName;
player.health = loadData.playerHealth.ToString();
player.trandform.position = loadData.playerPosition.ToString();
player.sprite = loadData.playerSprite;
```

Saving

Saving can be done whenever, however it is recommended that you only save when necessary to avoid an impact to your games performance.

```
// Makes a new save data class instance for use...
SaveData saveData = new SaveData();

// Applies the values to the class for saving...
saveData.playerName = player.name;
saveData.playerHealth = player.health;
saveData.playerPosition = player.transform.position;
saveData.playerSprite = player.sprite;

// Saves the game with the class creates/edited above...
SaveManager.SaveGame(saveData);
```

Error Messages & Common Problems

If you run into a problem or get an error and are unsure, feel free to drop me an email at (hello@carter.games) and I'll do my best to help you out.

My SaveData.cs class has an error after I generated/changed the class from the editor: Please make sure you didn't do any of the following:

- Set a class field that was not spelt correctly
- Have another script in the same namespace called SaveData.cs

I can't save a Vector or Color:

Please make sure the ExtraSaveTypes.cs class is imported into the project.

Why can't I find the Save Manager in my code?

Please make sure that you are using the namespace and have imported th eporject correctly.

Where do I put the save/load code?

This is up to you. You put it where you need it! We recommend that you put it in Awake/Start Methods or in a custom method that only gets called once. If you are going to put the loading code in Update we recommend you have a boolean that is used to check if the game has loaded, once run you set it to true it running more than once.