SAVE MANAGER

🟦 Welcome

Video Tutorial: https://www.youtube.com/watch?v=D8PHlFH3eek

Save Manager is a game saving solution to let you save and load your games data on any platform. With useful tools such as profiles of save states that can be loaded in the editor & a full save editor to edit your save data in the editor. As well as modularity with save objects which you can make as many or as few as you need.

🌟 Thanks

Thank you for deciding to use my asset for your project. If you like my asset, feel free to leave a ⭐⭐⭐⭐⭐ review! If you find that our asset is not up to scratch or find and issue, please do let me know either via our email: hello@carter.games and I will do my best to help you with the issues you are facing. I can't read minds, so if you don't speak up, it won't get fixed 😆

🟦 How do I make my save data?

Video Tutorial: https://www.youtube.com/watch?v=NSC6mHsJusQ

Unlike version 1.x the save data in this version is modular. To define data you'll need to make your own class for it. You can make as many of these as you need to hold your data, splitting up elements of your save as needed.

For ease of use you can make your own either by hand or using a little tool in the asset, below I'll go over both methods:

🔷 Using Tool
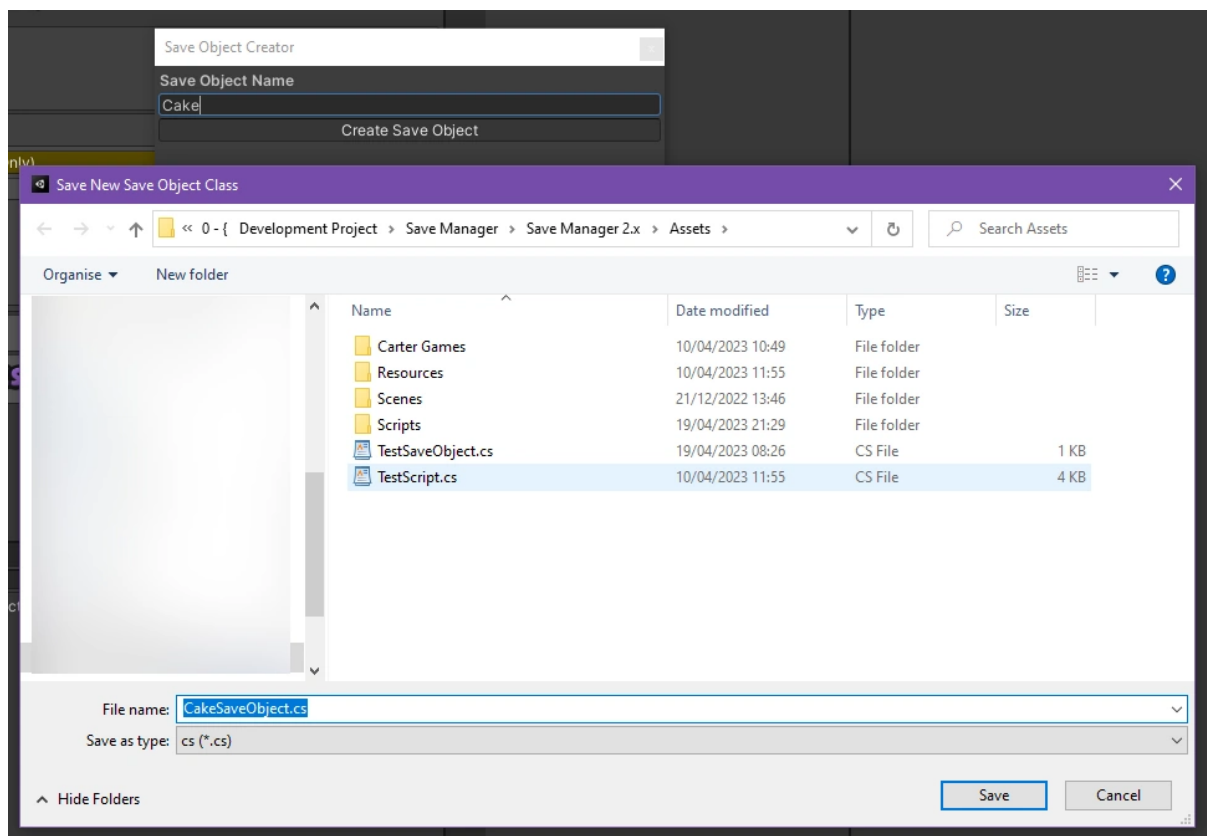
The tool can be accessed via the following path:

```
Tools > Carter Games > Save Manager > Save Object Creator
```
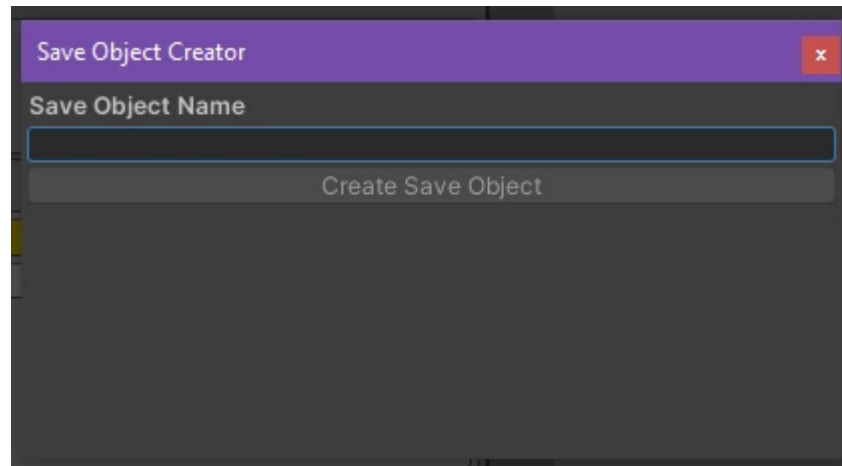

This will open this little window, from here all you need to do is enter a name for the class you'd like to make. By default the class name will be what you enter followed the "**SaveObject**", so if I entered **Cake** it would default the name to **CakeSaveObject.cs**

When you press the create button it will ask you to choose save location for the new class. Note this needs to be in the projects Assets/ folder. You can rename the file if you are not happy with the generated name. Once you press save the file wil be generated.
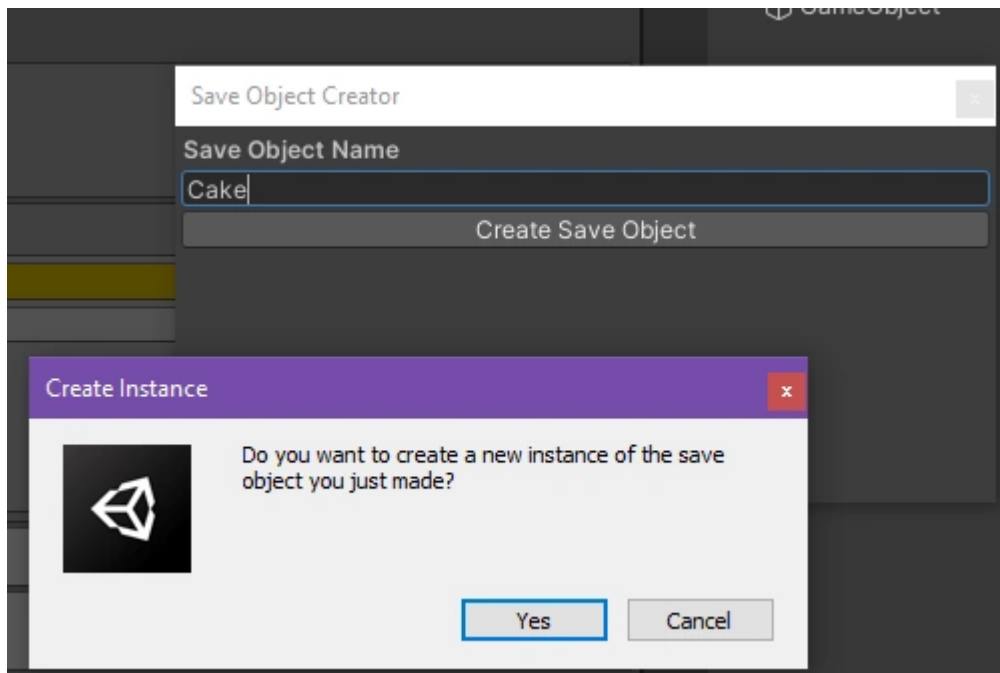
Once the script has been generated and the project has recompiled, you will be prompted with the option to make a new instance of the save object you just made, if you press cancel nothing will happen and you're all set. If you press yes a new save object (scriptable object) will be generated at the same path the class was just created to so you are all ready to use the new object in your project.

The creator uses a template file provided with the asset which I'll show below, it has a comment for where you should put your save values. You can also change the namespace and **createassetmenu** setup if you wish.

```
using CarterGames.Assets.SaveManager;
using UnityEngine;

namespace Save
{
    [CreateAssetMenu(fileName = "%SaveObjectName%")]
    public class %SaveObjectName% : SaveObject
    {
        // Enter your save values here...
    }
}
```

🔷 Manual Creation

You'll need to make a new C# class first. Once you've made a new class you'll need to inherit from **SaveObject** like so:

```
public class MyClass : SaveObject
{

}
```

If your code is in its own assembly definition you'll need to reference the Save Manager Runtime assembly for the inheritance to work.

From here you now have a save object to add values to. Note that this is a scriptable object, so you'll need to add the [**CreateAssetMenu**] attribute to it to allow you to make an instance in your project.

```
[CreateAssetMenu]
public class MyClass : SaveObject
{

}
```

Now all there is to do is to add the data you want to save. Note that whatever you want to save has to be serializable. Most common are, but if you are saving custom classes etc. you'll need to make sure this is the case. The Save Manager using a JSON based save setup so elements such as vectors will work just fine. The asset also comes with a basic serializable dictionary class which can be used to save dictionaries should you wish.

To make a new save value use the **SaveValue<T>** generic class and pass in the type you want to save. If you want to customise the default value or key you can do so by initializing the field on declaration. Note that the save keys **MUST** be unique for the save to work. If left without a declaration it will have a random save key & the default as the type default.

🟦 Asset settings & data assets

The save manager creates several different scriptable objects when installed in a project. These all do different things which I'm going to go over on this page and what all their options mean.

🔷 Runtime Settings

As of release, this is the first asset to have two different settings assets. One for editor only settings such as tab positions & colours while the other is for runtime settings to do with the assets actual functionality. By default you can't edit either of these directly on the assets and only the runtime settings can be edited via the settings provider window in **Project Settings > Carter Games > Save Manager**.

🔷 Runtime Settings

There are a few settings that are not exposed to you the user in the settings provider, these are either automated settings or settings that if edited can cause errors that are very hard to fix. These being the save paths & the save data reference. The save paths do have some logic to allow for some customization but changing them can cause a load of edge cases, too many for me to fix in the initial release (2.0.0).

| Setting	Type	Description
Encryption Option	EncryptionOption (Enum)	Sets the encryption setting that is used for the save. The options are: **Disabled** Doesn't use encryption, the save will be in plain text, handy for debugging but not recommended for your release builds. **Enabled (AES)** Uses AES encryption on the save file when saving & loading the game. The key is in a separate asset stored in the project. It'll deter your common player from editing the save, but this will not work for those who are determined or have the skills to decompile your game as the key itself is pretty available for those will the skills. Best used for single player games, for multiplayer you may have to look elsewhere sadly.
Auto Load	bool	Should the save auto load on starting up?
Auto Save	bool	Should the game auto save when exiting, note this won't apply changes that have not been sent to save objects beforehand.
Pretty Save Formatting?	bool	Sets if the JSON is formatted in a more readable format in the save. **Note:** this only works in the disabled mode, when encryption is in use it doesn't use this setting for your save.
Show Log Messages	bool	Defines if the asset throws any log messages to aid with potential issues. Disable this if you want a cleaner console.

◆ Editor Settings

The editor settings cannot be edited directly by you the user and are stored in your editor prefs. In versions before **2.0.13** they were stored in an asset like the runtime settings. In **2.0.13** this was changed and the old asset should auto delete itself on installing any version from that one and above. As editor settings are per user settings liek tab positions in the editor it made sense to keep these out of version control with this change. You can reset these settings via the menu item: **Tools/Carter Games/Save Manager/Reset User Editor Settings**

◆ Save Data
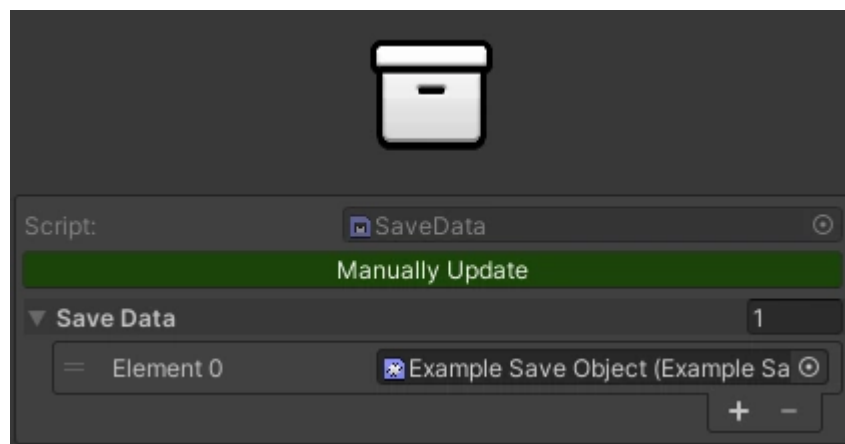
The save data asset stores the a collection of every save object instance in your project. This should update for you when you make new asset, however if it doesn't you can update it manually via the "Manually Update" button on the save data editor.

🔷 Encryption Key Asset
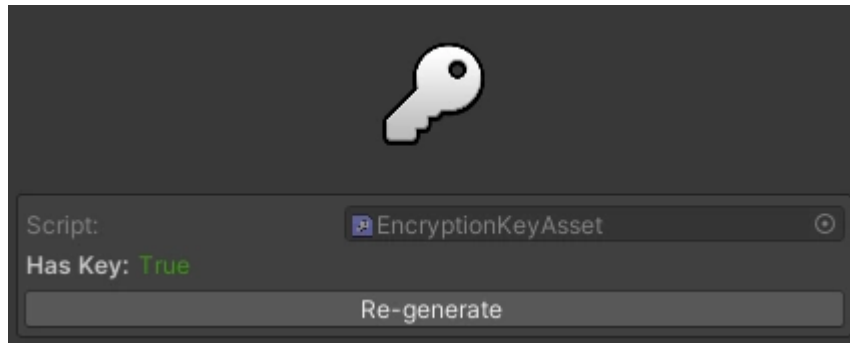
The encryption key asset stores the key used to save you game data. You can regenerate a new key by pressing the button in the inspector. **Note:** doing this will mean old saves will not decrypt. So this should only be done when in development.

🔷 Save Profiles
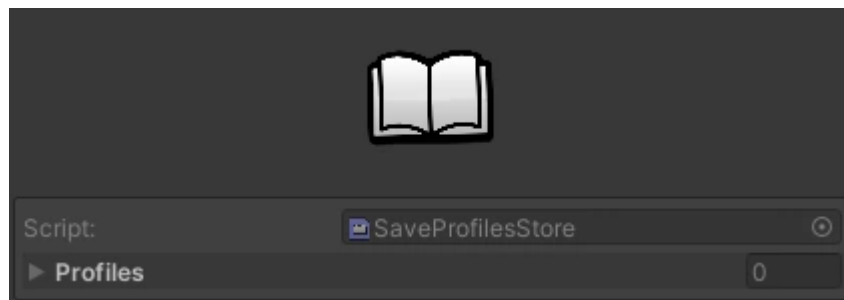
The save profiles asset just stores a reference to all the profiles you have made and will update based on changes to the profiles in the editor window for the asset.

🟦 How to save & load the game

The game will auto load and/or save all save objects on the game opening or closing if the relevant settings are enabled. To save the game manually you just need to call the load or save methods from the Save Manager class like so:

```
// Saves all save objects to the game save file.
SaveManager.Save();

// Loads all the save objects to the latest values in the save file.
SaveManager.Load();
```

It's advised to not call these methods too frequently as they can cause a performance hit. It's best to only call to save when you really need to like when changing levels or area's etc.

🟦 What types can I save?

The save manager supports any type that is serializable by Unity by default. This means your common types like `bool`, `string`, `int` etc. as well as `Array` & `Lists`. You can read more on the serialization here: https://docs.unity3d.com/Manual/script-Serialization.html

The only addition in this asset is the support to save a dictionary using a custom `SerializableDictionary` class. The class just stores a list of a custom key pair class that is serializable. The asset uses this for its lookups and you can use it in your code should you wish.

🟦 How to save in WebGL builds?

Note: The asset uses a deprecated method to make the WebGL save work. It has been tested in the version asset was developed in (2020.3.x), but may not work in higher versions.

🔷 TL:DR;

There is no additional setup on your end to use web saves, it will automatically use it if your game is in a build on the web platform, when in the editor it will use the standard setup.

🔷 Breakdown

In earlier versions of this asset, saving to web wasn't possible. However in this version it is. This has mainly been tested on itch.io but should work on any build in theory. Web saves use a slightly different setup to the standard save setup with a different path and some additional logic to make it work.

In the engine, the save will still use the default setup for standalone builds as the web setup will only work in builds. When in a build the save will be placed in the IndexDB of the users browser and will remain there until their clear their browser data. The path is idbfs/{YOUR_GAME_NAME_IN_PROJECT}-{COMPNAY_NAME_IN_PROJECT}/save.sf, this is the persistent data path with a consistent path. If the asset used the default persistent path it would be unique for each play so the user wouldn't have a save if they left the game and came back at a later date.

🟦 Where is my game save file saved to?

The save file is always created at the application persistent data path. This is done to ensure it will work for all platforms. You can access the save file quickly via the yellow button is the project settings for the asset:

Pressing this will open the save file location one directory back from the file with the folder to continue highlighted. Just enter that folder to find the save.sf file with your data. Note in builds this will not be the save file for the game unless you are building for PC/Mac/Linux. To find out where it is stored on build please see the persistent data path information here: https://docs.unity3d.com/ScriptReference/Application-persistentDataPath.html

🟦 Save Profiles

Video Tutorial: https://www.youtube.com/watch?v=udm_Ar6yX8c
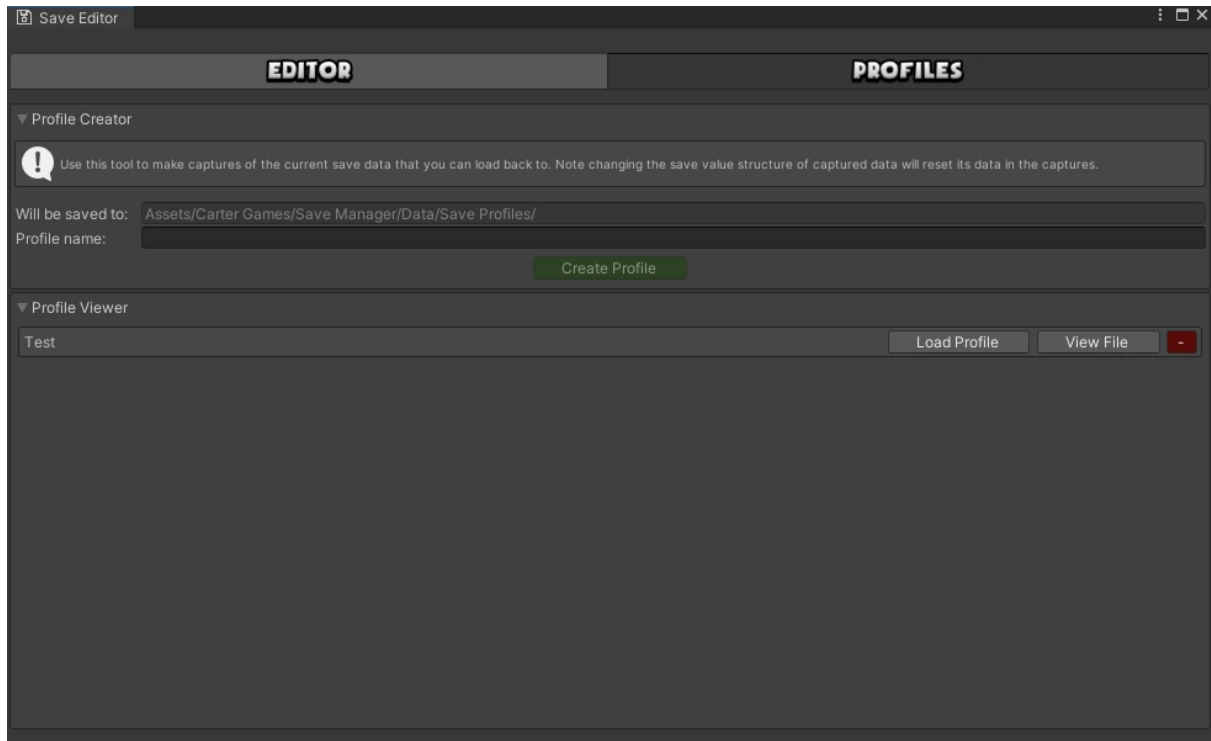
🔷 Profile Tab

The profile tab lets you create captures of your games save at its current state. You can then load the states to a previous save from the editor. Note that this only saves what values you currently save in their current type to a text file so any changes to types etc will not load from an older profile. You cannot load or edit save profiles at runtime.

🔷 Profile Creator

You can create a profile by just entering a new name for that profile and then press the create button. This will add a capture of your current save data, add it to the profiles store asset & display it in the profile viewer.

🔷 Profile Viewer

Here you can view all the profiles you have created and manage them. You have three options here:

- **Load Profile:** loads the profile to the save, overriding any current data.
- **View File:** Highlights the text asset for the profile.
- **-:** Deletes the profile from the system & removes the text asset.

14

🟦 Save Editor Window
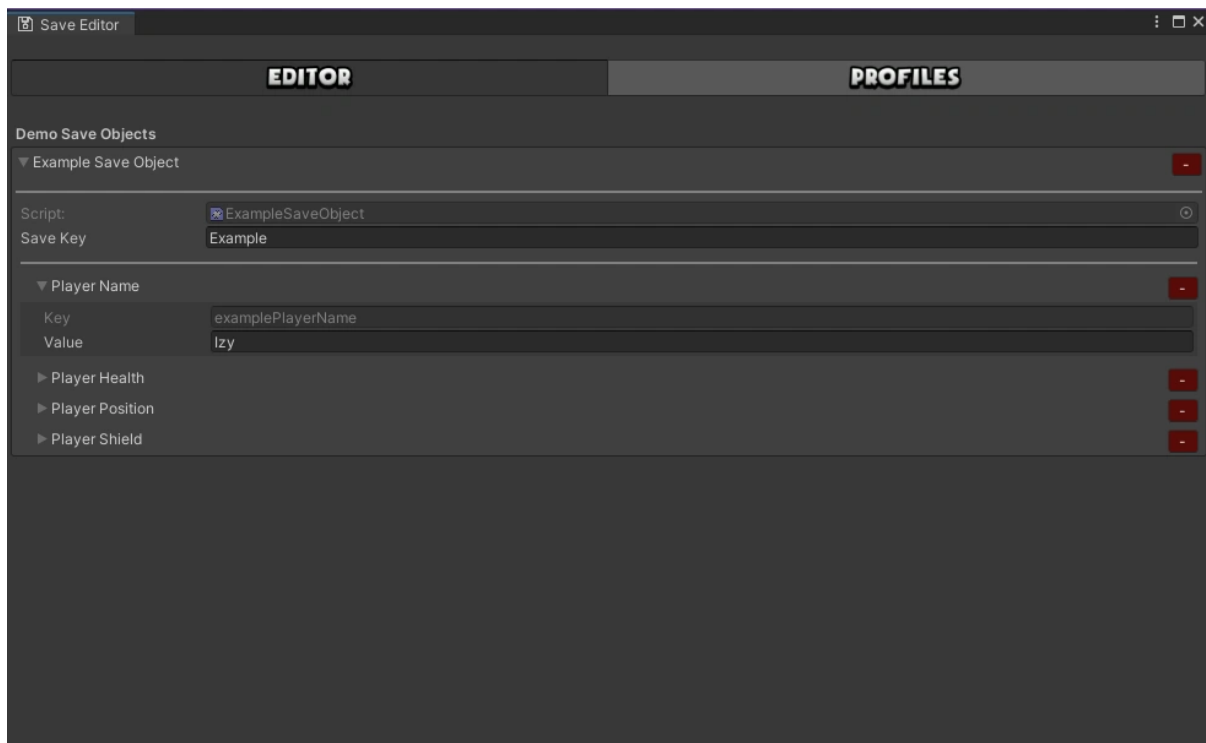
The editor tab will show all the save objects in your project in one place. You can expand/collapse any section or field at will and any edits made to any save data will apply to the save file so you can use it to debug & test parts of your game. Note that you can only edit the save while in edit mode. If in play mode the editor will be disabled.

If you want to reset a value you can press the red – next to the save value, this will reset the value to its type default. You can reset all on an object by pressing the same button next to the object dropdown.

🟦 Code Namespace & Assemblies

Assembly
Editor: CarterGames.SaveManager.Editor
Runtime: CarterGames.SaveManager.Runtime
Demo: CarterGames.SaveManager.Demo

Namespaces
Editor: CarterGames.Assets.SaveManager.Editor
Runtime: CarterGames.Assets.SaveManager
Demo: CarterGames.Assets.SaveManager.Demo

🟦 Save Manager

🔷 Assembly & Namespace

Runtime: CarterGames.SaveManager.Runtime

Namespace: CarterGames.Assets.SaveManager

The save manager is the main class you'll use to save & load your game in code. You can also use the class to get save objects via code instead of needing a reference to them directly.

🔷 API

Properties

bool IsInitialized
Gets if the save manager is initialized or not.

bool IsSaving
Gets if the game is currently saving or not.

Events

Evt Saved
Listen to, to get callbacks when the game has been saved.

```
SaveManager.Saved.Add(MyMethod);
```

Evt Loaded
Listen to, to get callbacks when the game has been loaded.

```
SaveManager.Loaded.Add(MyMethod);
```

Methods

void RegisterObject(SaveObject obj)
`SaveObject` → The save object to register.

Registers a save object with the save manager. It is called automatically by the save manager.

```
SaveManager.RegisterObject(MySaveObject);
```

void Save()
`bool` → Optional → Calls any save listeners. Default: True

Saves the game when called.

```
SaveManager.Save();
```

void Load()
Loads the game when called

```
SaveManager.Load();
```

void Clear()
Clears the save data when called.

```
SaveManager.Clear();
```

SaveObject<T> GetSaveObject()

Returns

T → The save object found of the requested type.

Gets the first save object of the requested save object type. The type must inherit from the save object class to be found by this method.

```
MySaveObject obj = SaveManager.GetSaveObject<MySaveObject>();
```

string → The save key for the object to get.

Returns

T → The save object found of the requested type.

Gets the first save object of the requested save object type which has the matching save key. The type must inherit from the save object class to be found by this method.

```
MySaveObject obj = SaveManager.GetSaveObject<MySaveObject>("MySaveObjectKey");
```

bool TryGetSaveObject(SaveObject<T> obj)

Returns

T → The save object found of the requested type.
bool → If a save object was found.
out T → The save object found.

Tries to get the first save object of the requested save object type. The type must inherit from the save object class to be found by this method.

```
bool hasObject = SaveManager.TryGetSaveObject<MySaveObject>(out MySaveObject obj);
```

Method

`string` → The save key for the object to get.

Returns

`T` → The save object found of the requested type.

`bool` → If a save object was found.

`out T` → The save object found.

Tries to get the first save object of the requested save object type which has the matching save key. The type must inherit from the save object class to be found by this method.

```
hasObject = SaveManager.TryGetSaveObject<MySaveObject>("MySaveObjectKey", out MySaveObject obj);
```

🟦 Save Listeners

🔷 Assembly & Namespace

Runtime: CarterGames.SaveManager.Runtime

Namespace: CarterGames.Assets.SaveManager

The save listeners interface lets you hook into the save manager saving & loading directly. You can implement these interfaces to run your own logic when the save system does stuff. You can also just use the Save Manager events should you wish.

🔷 API

🔷 ISaveListener

void OnGameSaveCalled()
Runs when the game has been called to save by any means to the save file.

```csharp
public class Example : MonoBehaviour, ISaveListener
{
    public void OnGameSaveCalled()
    {
        // Your logic here!
    }
}
```

void OnGameSaveCompleted()
Runs when the game has completed saving to the save file, this normally is pretty quick so will run more or less right after the OnGameSaveCalled() is invoked.

```csharp
public class Example : MonoBehaviour, ISaveListener
{
    public void OnGameSaveCompleted()
    {
        // Your logic here!
    }
}
```

🔷 ILoadListener

void OnGameLoadCalled()
Runs when the game has been called to load by any means from the save file.

```
public class Example : MonoBehaviour, ILoadListener
{
    public void OnGameLoadCalled()
    {
        // Your logic here!
    }
}
```

void OnGameLoadCompleted()
Runs when the game has completed loading from the save file, this normally is pretty quick so will run more or less right after the OnGameLoadCalled() is invoked.

```
public class Example : MonoBehaviour, ILoadListener
{
    public void OnGameLoadCompleted()
    {
        // Your logic here!
    }
}
```

🟦 Support

✉ Email

You can email me any time through the support email: hello@carter.games.

💙 Discord

You can join the community discord server and react with the assets 🎨 role in the:
✅ server-info-rules channel to gain access to support channels for each asset.
https://carter.games/discord