

# ESave Documentation

---

v1.0.1

May 20, 2024

Stylish Esper

Document Version: 1.0.1

## Table of Contents

Online Documentation .....	3
Installation .....	3
Asset Store .....	3
Github .....	3
Example Setup .....	3
Components .....	3
File Structure.....	3
Create Menu Items .....	4
Save File Setup .....	4
Save Storage .....	5
Saving.....	6
1. Getting the Save File.....	6
2. Saving Data .....	6
Loading.....	7
1. Getting the Save File .....	7
2. Saving Data.....	7
GetData .....	7
Special Methods .....	7
Example.....	8

## Online Documentation

ESave's documentation may be updated occasionally. To find the latest version of the documentation, use [this link](#).

## Installation

### Asset Store

You can find the latest version of ESave in the asset store with [this link](#).

Installation Steps:

1. Get ESave from the asset store.
2. Open your Unity project, go into Window -> Package Manager, switch to My Assets from the packages dropdown at the top-left, and then type ESave in the search bar.
3. Download and install the package.
4. Once it's complete, a package installer window will popup. Click Install Newtonsoft JSON.

### Github

You can find the latest version of ESave on GitHub with [this link](#).

Installation Steps:

1. Download one of the .unitypackage files from the [releases page](#).
2. Open your Unity project and double click the downloaded Unity package.
3. Click import.
4. Once it's complete, a package installer window will popup. Click Install Newtonsoft JSON.

Once you've got it installed, familiarize yourself with the tool by reading the startup guide.

## Example Setup

You can find the example scene that works with any render pipeline in Assets/ESave/Examples/Any RP Example.

If you'd like to try the URP example, you can import the assets from the URP\_Example.unitypackage in Assets/ESave/Examples.

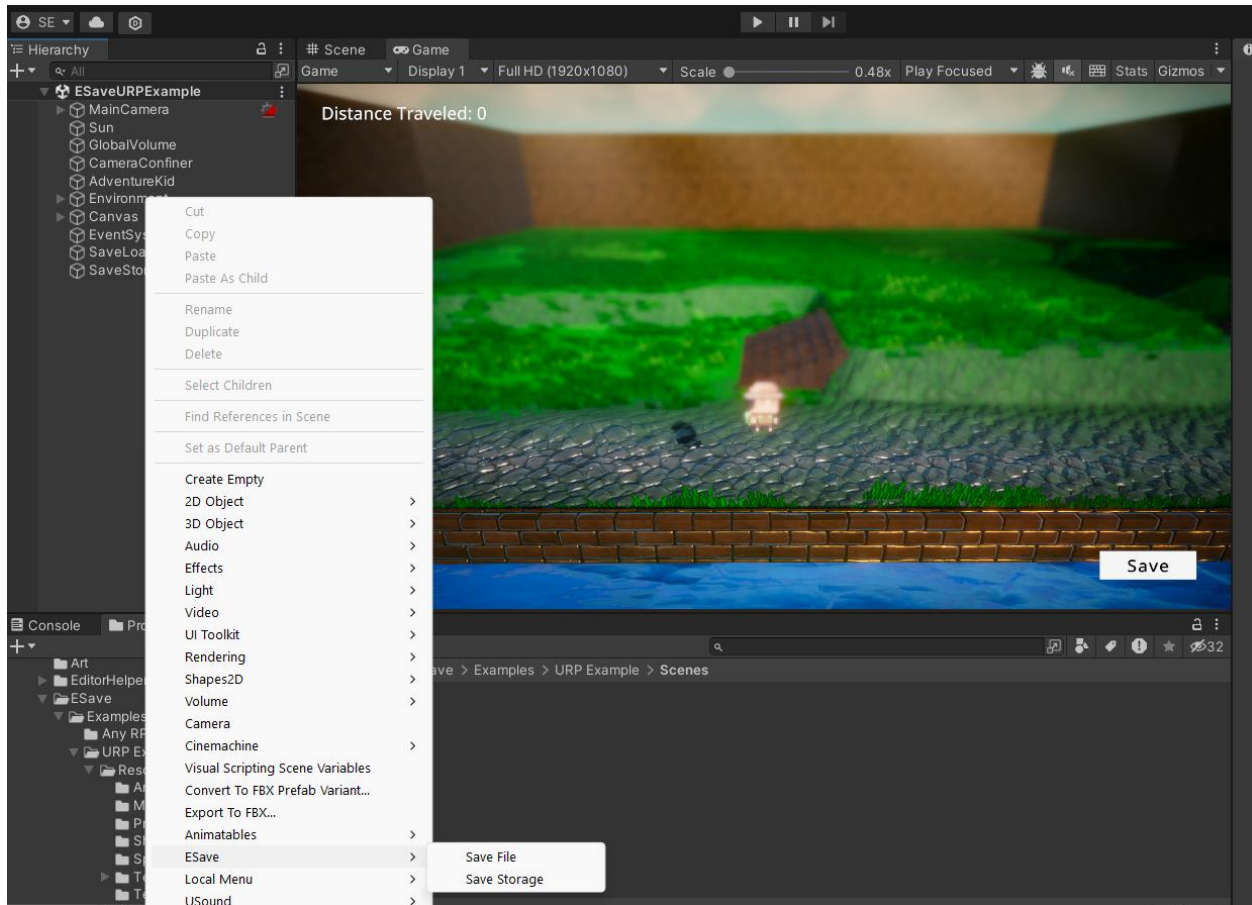
## Components

### File Structure

Save data is typically stored in separate files, with one for each save state and one for the player's settings. ESave is structured around this idea, which is why it's an important concept to remember.

Separating your saved data is recommended. Having all of the player's data in one file is possible and very common, but it's not a requirement. For example, you could have one save file that stores the player's inventory data, one for storing the player's progress, and another for the environmental impacts they made.

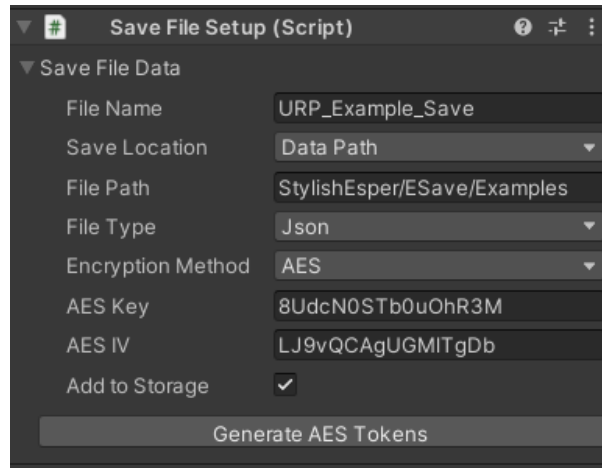
## Create Menu Items



You can easily create ESave's save file setup and save storage components by right-clicking on the hierarchy and navigating to ESave.

## Save File Setup

Save file setup is the main component you will be working with. This component will work exactly as it's named; it will create a save file in the user's system that you can save and load data from.



### File Name

The name of the file that will be saved in the player's system.

### Save Location

There are 2 common locations that you can store player files.

1. **Persistent Data Path:** a directory path where data expected to be kept between runs can be stored.
2. **Data Path:** the path to the game data folder on the target device.

### File Path

This is the extra path after the save location path. For example, in the above image, the file `URP_Example_Save` will be stored in the path `StylishEsper/ESave/Example` in the game's data folder.

### File Type

The file type determines the saved data format and the file extension. Currently, only the JSON format is supported.

### Encryption Method

Only the AES encryption algorithm is supported at this time. The default option is no encryption.

### AES Key and IV

The key and IV are used for the AES algorithm. You can click the 'Generate AES Tokens' button to generate a random token for both.

### Add to Storage

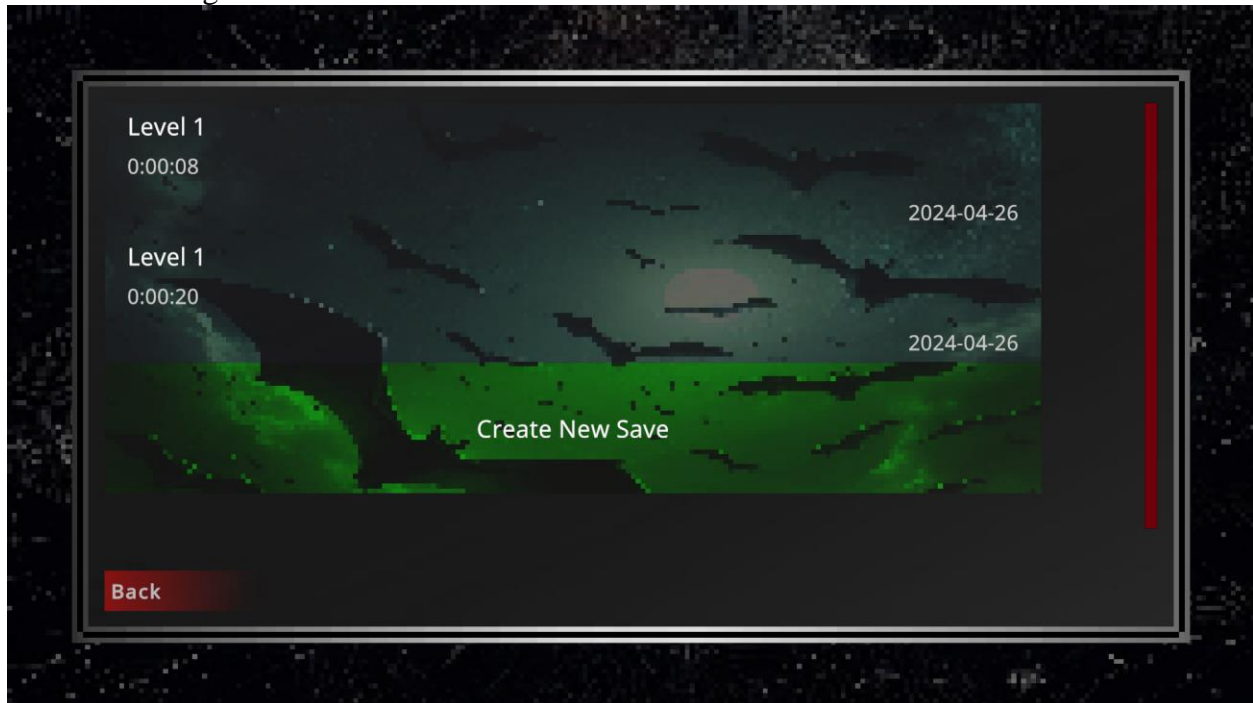
If add to storage is checked, the save file will be added to the save storage.

## Save Storage

Save storage is a singleton component that has no inspector properties. Save storage automatically stores the paths of every save file that has 'Add to Storage' checked. It requires a save file setup component to store the paths.

Ensure the 'Add to Storage' box is unchecked for Save Storage's save file setup component.

With this component, you can find any save (that has the 'Add to Storage' value checked) by its file name through code.



Save storage isn't a mandatory component, but having it can significantly simplify tasks like accessing all of a player's saves, particularly useful for games featuring a save list screen.

## Saving

### 1. Getting the Save File

The first step to saving data is getting the save file. This can be done by getting the save file setup component first, which will give you access to the save file with

`SaveFileSetup.GetSaveFile.`

```
saveFileSetup = GetComponent<SaveFileSetup>();  
saveFile = saveFileSetup.GetSaveFile();
```

It's recommended to use this code in Awake or Start.

### 2. Saving Data

Each piece of data in a save file has an ID (string). This makes it possible to retrieve specific data by ID. Saving data can be done with a single line with `SaveFile.AddOrUpdateData.`

```
saveFile.AddOrUpdateData("DataID", data);
```

The first parameter is the ID of the data, and the second parameter is the data itself. Any serializable data type can be saved. However, most Unity types are not savable. ESave can save common Unity data types, including (but not limited to) Vector2, Vector3, Quaternion, Color, and Transform.

## Loading

### 1. Getting the Save File

Just like saving, you need a reference to a save file to load data.

```
saveFileSetup = GetComponent<SaveFileSetup>();  
saveFile = saveFileSetup.GetSaveFile();
```

### 2. Saving Data

Each ESave method that loads data only accepts one parameter, which is the ID of the data.

#### GetData

The main method to load data is `SaveFile.GetData`. This method accepts any type parameter.

```
// Where T is the data type  
T data = saveFile.GetData<T>("DataID");
```

You can also retrieve a list of data of the same type:

```
// Where T is the data type  
List<T> dataList = saveFile.GetData<T>("DataID", "DataID2",  
"DataID3");
```

#### Special Methods

ESave features special methods to retrieve data for some Unity types.

##### Vector2

```
Vector2 v2 = saveFile.GetVector2("DataID");
```

##### Vector3

```
Vector3 v3 = saveFile.GetVector3("DataID");
```

##### Quaternion

```
Quaternion q = saveFile.GetQuaternion("DataID");
```

##### Color

```
Color c = saveFile.GetColor("DataID");
```

## Transform

```
// Returns a SavableTransform
SavableTransform st = saveFile.GetVector2("DataID");

// Use Transform.CopyTransformValues to set values from a
// SavableTransform to a Transform
transform.CopyTransformValues(st);
```

## Example

This script saves the player's transform values when the game is exited and loads the transform values on start.

**When saving a transform with ESave, only the position, rotation, and scale properties are saved.**

```
using UnityEngine;

public class SaveLoadExample : MonoBehaviour
{
    // Const data ID
    private const string playerTransformDataKey = "PlayerTransform";

    [SerializeField]
    private Transform playerTransform;

    private SaveFileSetup saveFileSetup;
    private SaveFile saveFile;

    private void Start()
    {
        // Get save file component attached to this object
        saveFileSetup = GetComponent<SaveFileSetup>();
        saveFile = saveFileSetup.GetSaveFile();

        // Load game
        LoadGame();
    }

    /// <summary>
    /// Loads the game.
    /// </summary>
    public void LoadGame()
    {
        // Check if the data exists in the file
        if (saveFile.HasData(playerPositionDataKey))
        {
            // Get Vector3 from a special method because Vector3 is
            not savable data
            var savableTransform =
            saveFile.GetTransform(playerPositionDataKey);
```



```

        playerTransform.CopyTransformValues(savableTransform);
    }

    Debug.Log("Loaded game.");
}

/// <summary>
/// Saves the game.
/// </summary>
public void SaveGame()
{
    saveFile.AddOrUpdateData(playerPositionDataKey,
playerTransform);
    saveFile.Save();

    Debug.Log("Saved game.");
}

private void OnApplicationQuit()
{
    SaveGame();
}
}

```