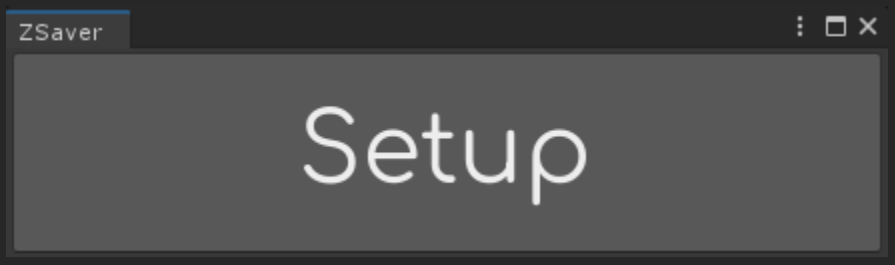
ZSerializer

Quick Start Guide

Contents

Setup

When you first install ZSerializer you will be greeted by this big Setup button. When you click it, it'll generate the necessary files you need to start using the tool.



If it's your first time updating to a version beyond 2.0, you will be prompted to Reset all the project ZUIDs. You are allowed and should do it as long as

Serializing Components

Serializing GameObjects

Saving and Loading.

Serialization Groups

Scene Groups

Component Settings

Advanced Serialization.

Asynchronous Serialization

The ZSerializer Menu

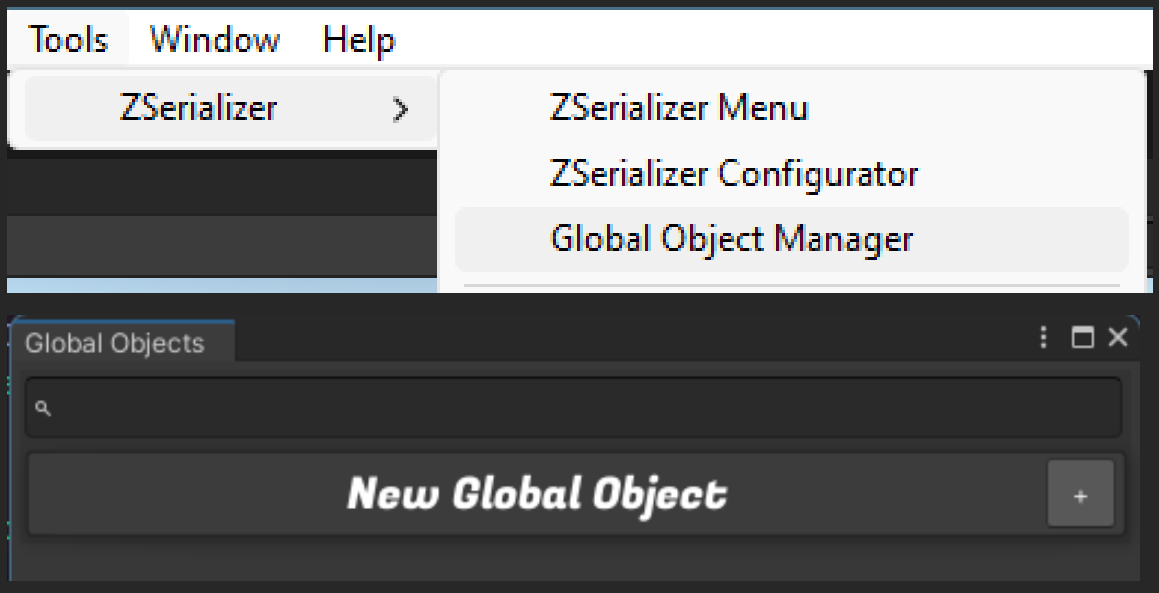
Serializing data Globally

One of the biggest hiccups of the main workflow of ZSerializer is that it's Scene based, so, in a production environment, things like saving settings and any kind of data that you need in several scenes is a really big hassle.

This is where the Global Objects System come in handy:

Your First Global Object

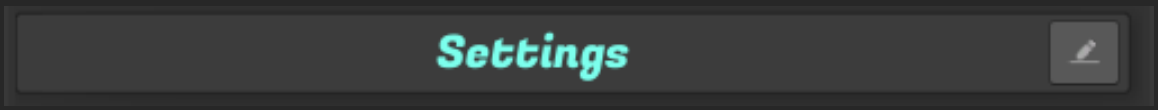
To create a global object, go to Tools/ZSerializer/Global Object Manager



When you press the + button, you'll create a new global object with the name you prefer. This object is a class where you can write whatever

variables you want, and they will get serialized with the same conditions as Persistent MonoBehaviours' variables.

Let's say you want to save your game's Settings, we'll create a Global Object called Settings:



Upon creation, you can click on the pencil icon, and your IDE will open the object's file.

```
[Serializable, SerializeGlobalData(GlobalDataType.PerSaveFile)]
1 asset usage  3 usages  Alberto Fernández Ruiz  1 exposing API
public partial class Settings : GlobalObject
{
    //  Add serializable variables to this object to be able to serialize and access them.
}
```

In here you can do several things:

- Change the Serialize Global Data Type attribute to save your global object per save file or globally. In the case of Settings, because we want it to be the same no matter the save file, we can change it to Globally, but for another object like an Inventory, you'd want it to be saved per Save File.
- Write variables that will get stored and any other functionality that you need, as if you were writing any other class.

```
[Serializable, SerializeGlobalData(GlobalDataType.Globally)]
1 asset usage  3 usages  Alberto Fernández Ruiz +1 *  1 exposing API
public partial class Settings : GlobalObject
{
    //  Add serializable variables to this object to be able to serialize and access them.
    public float musicVolume;  Unchanged
    public float soundFXVolume;  Unchanged

    public bool assistMode;  Unchanged
    public bool rtxOn;  Unchanged


    new *
    public void ResetToDefaultSettings(){...}
}
```

Any variable that would be serialized by Unity's default Inspector, will get serialized by this system when you save and load them.

Saving and Loading

To save and load your global objects, it's as easy as calling the Save and Load functions inside each class:

```
Event function  new *
void Start()
{
    Settings.Load();
}

Event function  new *
private void Update()
{
    if (Input.GetKeyDown(KeyCode.S))
    {
        Settings.Save();
    }
}
```

Referencing Global Objects' Variables

These variables are not static, so we can't do the same thing as we did with the save and load functions.

Because there can only be one object of type Settings, there's a singleton reference we can use. You can just call Settings.Instance, and you'll have access to all of your variables from there.

```
{
    Settings.Instance.assistMode = false;
    Settings.Instance.soundFXVolume = .25f;
}
```