Scriptable Object Architecture Pattern

User Guide

Version 2.4.0

OBVIOUS Game

SOAP:
Scriptable Object Architecture Pattern

Table of Contents

Introduction

Thank you for downloading Soap 😊. The goal of Soap is to help you developing your project. It aims to be easy to understand and to use, and therefore makes it useful to both junior and senior game developers.

Soap leverages the power of **scriptable objects**, which are native objects from the Unity Engine. Scriptable objects are usually used for data storage. However, they can contain code and because they are assets, they can be referenced by other assets and are accessible at runtime and in editor.

Soap is based and build upon the GDC talk of Ryan Hipple: https://www.youtube.com/watch?v=raQ3iHhE_Kk&ab_channel=Unity.

It was developed mostly for and while making Mobile casual games. Because of the nature of these games, the core strength and focus of Soap is **speed** and **simplicity** rather than scalability and complexity. Nevertheless, it can be used in more complex games to solve certain problems, particularly dependencies.

The package contains **5 examples scenes** to quickly show how to use this framework. Each scene has its own specific documentation (in the same folder as the scene). On top of this, there are **6 step-by-step tutorial videos** to help you to create a Rogue like game with Soap: https://www.youtube.com/@obviousgame/

Why use Soap?

<u>Solve dependencies</u>

Avoid coupling your classes by using a reference to a common Scriptable object as a bridge or by using Scriptable Events.

This is particularly true in the Casual mobile market for several reasons:

- Many features are enabled/disabled through independent AB Tests. Therefore, we want to avoid noisy code and hardcoding our features into the core of our game. Having them “hooked” with an independent architecture makes it easy to add or remove them.
- Soap enables you to create reusable features more easily. By using Soap, you can create features as “drag and drop” and this can be a huge time saver over the long run for things that generally don’t change too much across games.

<u>Speed</u>

- Avoid creating new classes for simple behaviors.
- Gain access to key variables directly with a simple reference.
- Modify variable from the editor or code.
- Have persistent data across scene or play sessions already handled.
- Be able to debug and have the game react in real time.

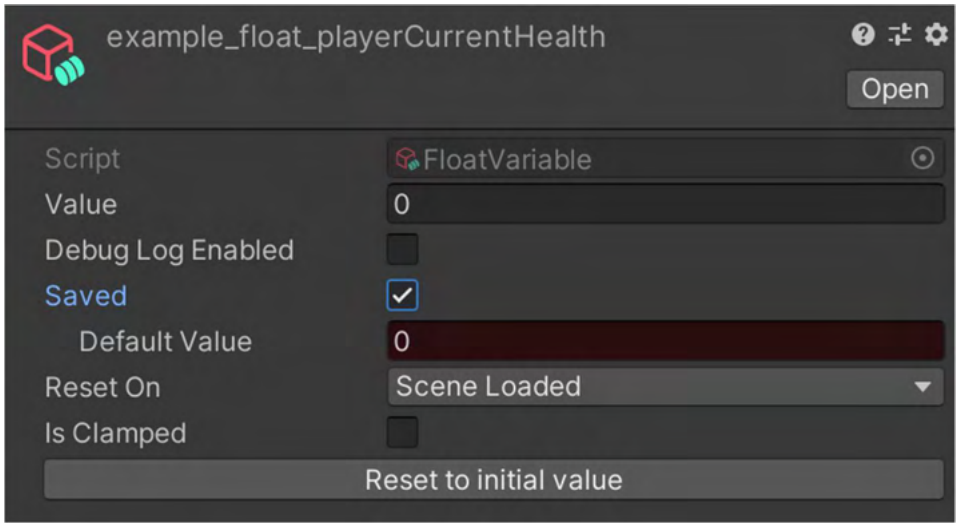
<u>Efficiency / Clarity</u>

- Subscribe to what you need and have each class handle itself.
- Avoid useless managers.
- Reduce code complexity (by preventing spaghetti code)

Finally, because its fun. Once you start developing with Soap, you might realize that the workflow is more enjoyable. It is not for everyone, but people who like this kind of workflow will love Soap.

Scriptable Variables

A scriptable variable is a scriptable object of a certain type containing a value. Here is an example of a FloatVariable:



example_float_playerCurrentHealth

Open

Script	FloatVariable
Value	0
Debug Log Enabled	
Saved	✓
Default Value	0
Reset On	Scene Loaded
Is Clamped	

Reset to initial value

Let's go through its properties:

- **<u>Value</u>**: the current value of the variable. Can be changed in inspector, at runtime, by code or in unity events. Changing the value will trigger an event "OnValueChanged" that can be registered to by code. See examples for practical usage.
- **<u>Debug Log Enabled</u>**: if true, will log in the console whenever this value is changed.
- **<u>Saved</u>**: If true, the value of the variable will be saved to Player Prefs when it changes. (More information in the 5_Save_Documentation).
- **<u>Default Value</u>**: if "Saved" is true, then you can set a default value. This is used the first time you load from PlayerPrefs if there is no save yet.
- **<u>Reset On:</u>** when is this variable reset (or loaded, if saved)?
  - Scene Loaded: whenever a scene is loaded. Ignores Additive scene loading. Use this if you want the variable to be reset if it is only used in a single scene.
  - Application Start: reset once when the game starts. Useful if you want changes made to the variable to persist across scenes.

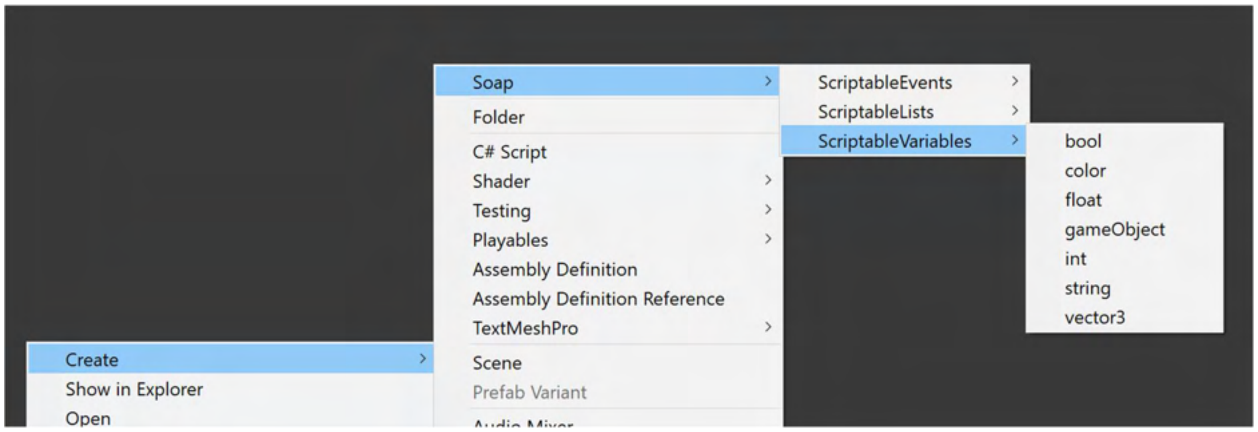
- **Is Clamped:** specific to FloatVariable and IntVariable, gives you the ability to clamp it if you need. (More information in the 1_ScriptableVariables_Documentation).

In the Editor, ScriptableVariables automatically reset to their initial value (the value in the inspector before entering play mode) when exiting play mode.

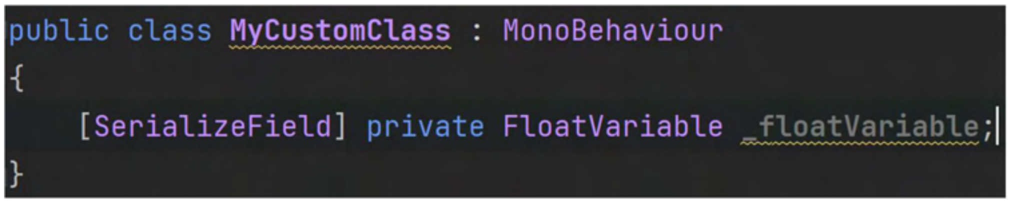
Finally, there is a utility button: "Reset to initial value" which lets you quickly reset the value of the variable to the initial value. It's a quality-of-life button.

To create a new variable, simply right click in the project window and find the scriptable variable you want:

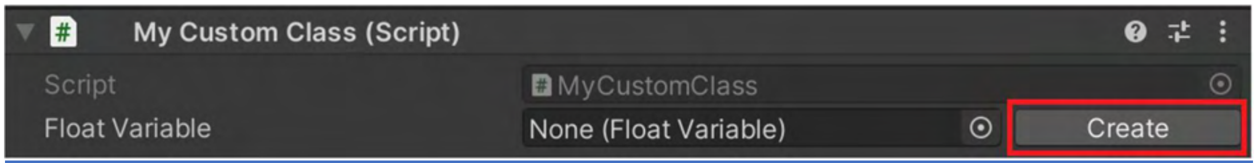
Create/Soap/ScriptableVariables/



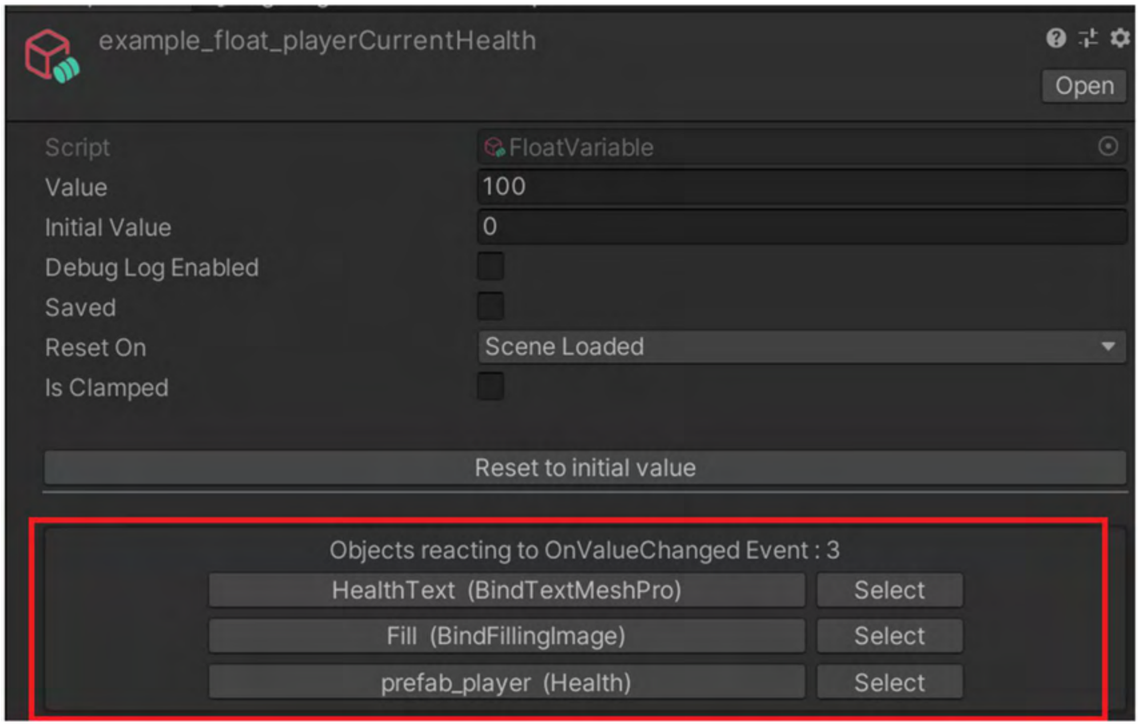
Alternatively, you can also expose a reference to a Scriptable Variable directly in your class, and then click on the create button from the inspector. This will create a new instance of that scriptable variable in the folder you have currently selected in the project window. Example:



```
public class MyCustomClass : MonoBehaviour
{
    [SerializeField] private FloatVariable _floatVariable;
}
```



When you are in **play mode**, the objects (and their component) that have registered to the **OnValueChanged** Event of a scriptable variable are visible in the inspector.



example_float_playerCurrentHealth

Script	FloatVariable
Value	100
Initial Value	0
Debug Log Enabled	
Saved	
Reset On	Scene Loaded
Is Clamped	

Objects reacting to OnValueChanged Event : 3

HealthText (BindTextMeshPro)	Select
Fill (BindFillingImage)	Select
prefab_player (Health)	Select

As you can see in the screenshot above, 3 objects are registered to this variable.

If we inspect the first element that react to this variable, we can find the GameObject named “HealthText” in the hierarchy and, more specially, its component “BindTextMeshPro”.

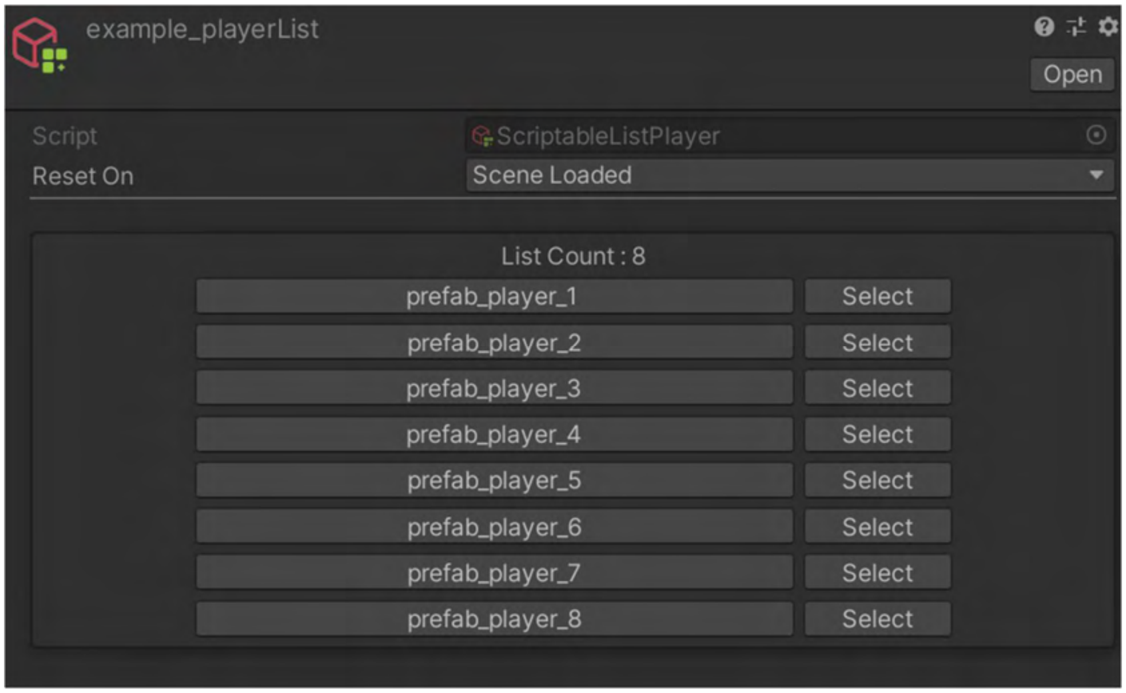
By clicking on the first button, we ping the object in the hierarchy. By clicking on the “Select” button, it will select the object in the hierarchy.

This visual debug can be useful to keep track of what is reacting to the changes of your variable.

Please check the example scene 1_ScriptableVariables to understand concrete use of scriptable variables.

Scriptable Lists

Lists are useful to solve dependencies by avoiding the need to have a "manager" that holds that list.



example_playerList

Script: ScriptableListPlayer
Reset On: Scene Loaded

List Count : 8

prefab_player_1	Select
prefab_player_2	Select
prefab_player_3	Select
prefab_player_4	Select
prefab_player_5	Select
prefab_player_6	Select
prefab_player_7	Select
prefab_player_8	Select

Let's go through its properties:

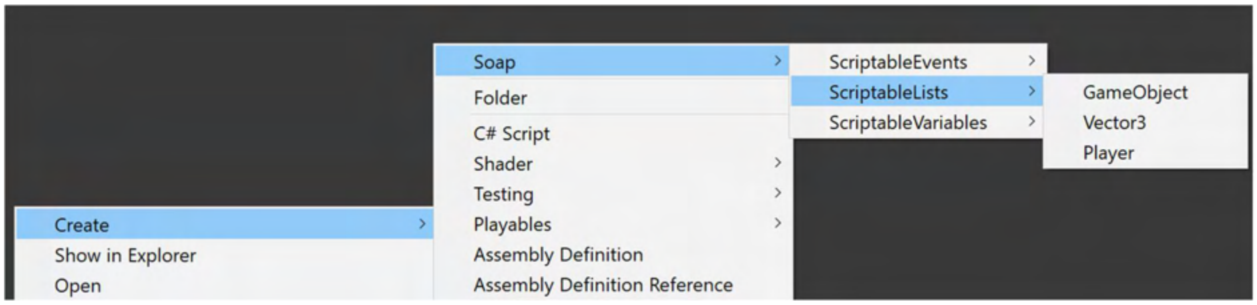
- **Reset On:** when is this list cleared?
    - *Scene Loaded:* whenever a scene is loaded. Ignores Additive scene loading.
    - *Application Start:* reset once when the game starts.

- **List content**: in play mode, you can see all the elements that populate the list. By clicking on the first button (with the name of the object), it pings the object in the hierarchy. By clicking on the "Select" button, it will select the object in the hierarchy.

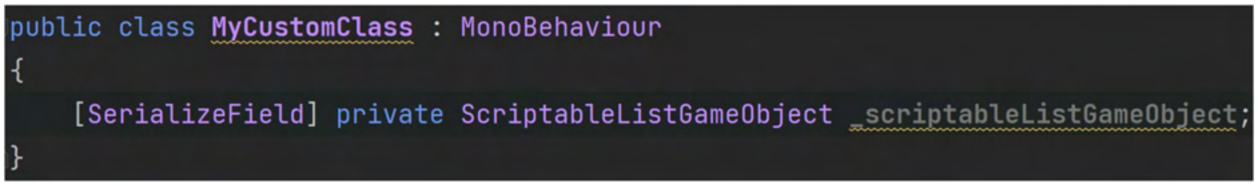
In the Editor, ScriptableLists automatically clear themselves when exiting play mode.

To create a new list, simply right click in the project window and find the scriptable list:

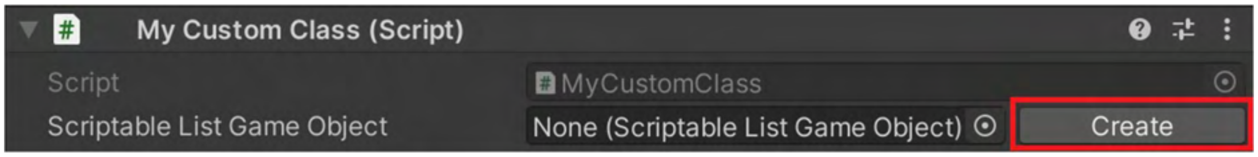
Create/Soap/ScriptableLists/



Alternatively, you can also expose a reference to a ScriptableList directly in your class, and then click on the create button from the inspector. This will create a new instance of that scriptable list in the folder you have currently selected in the project window. Example:



```
public class MyCustomClass : MonoBehaviour
{
    [SerializeField] private ScriptableListGameObject _scriptableListGameObject;
}
```

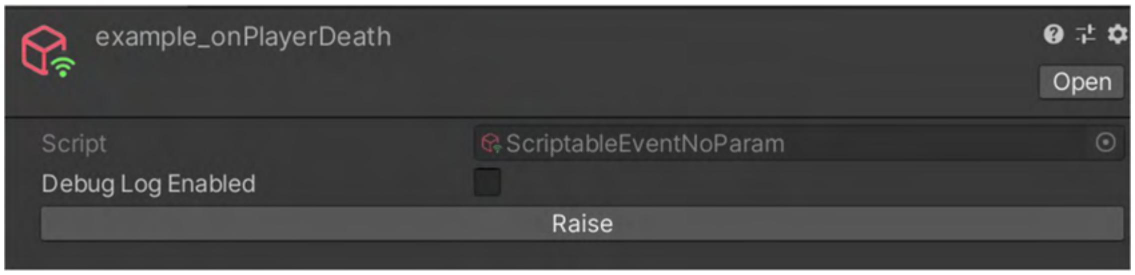

My Custom Class (Script)
Script
MyCustomClass
Scriptable List Game Object
None (Scriptable List Game Object)
Create

Please check the example scene 3_Lists to understand concrete use of scriptable lists.

Scriptable Events

There are two types of scriptable events:

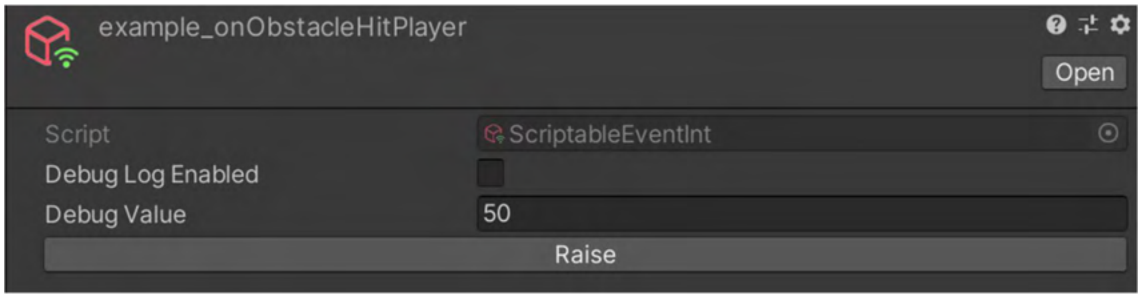
- Event without parameters



example_onPlayerDeath

Script	ScriptableEventNoParam
Debug Log Enabled	

- Event with parameter (int, bool, float, string, Vector2, Vector3, Color)



example_onObstacleHitPlayer

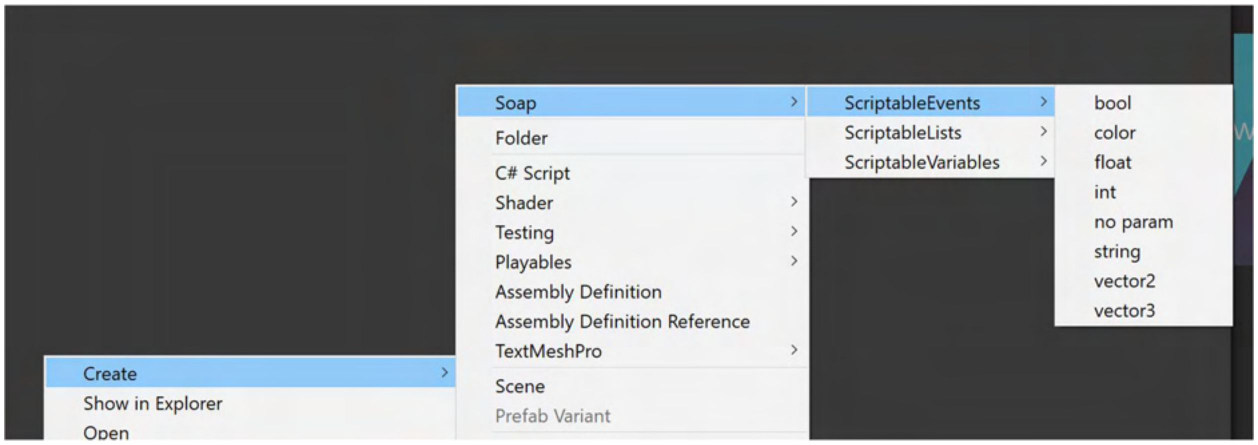
Script	ScriptableEventInt
Debug Log Enabled	
Debug Value	50

Events can be triggered by code, unity actions or the inspector (via the raise button).

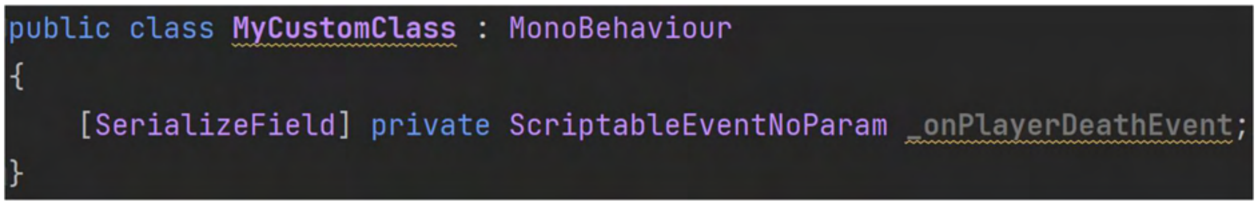
Raising events in the inspector can be useful to quickly debug your game.

To create an event:

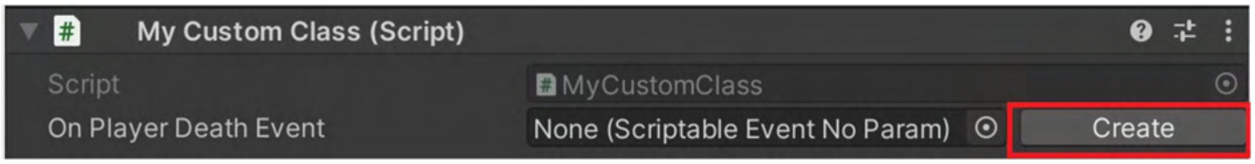
Create/Soap/ScriptableEvents/



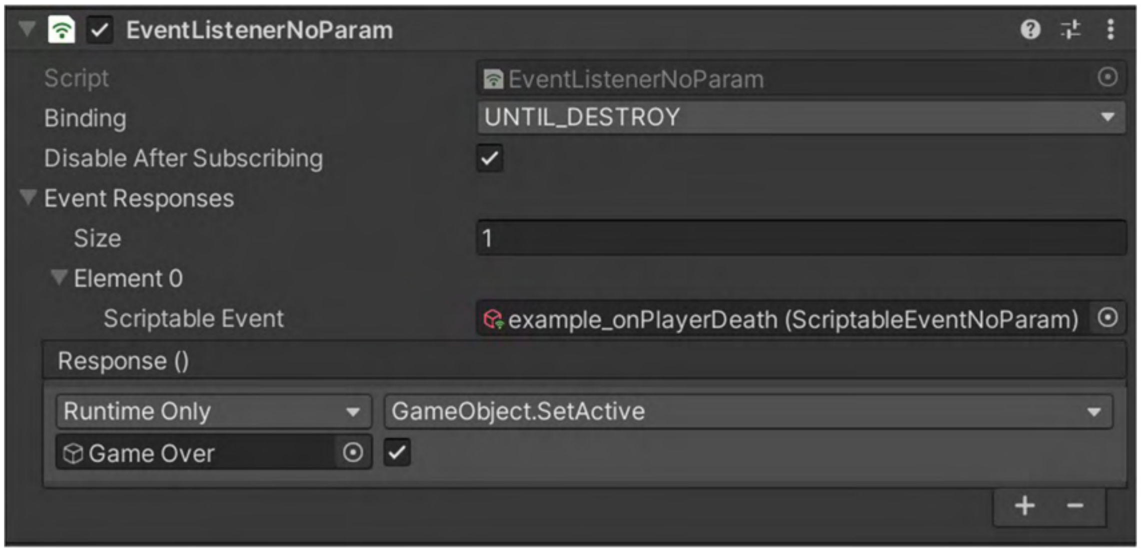
Alternatively, you can also expose a reference to a ScriptableEvent directly in your class, and then click on the create button from the inspector. This will create a new instance of that scriptable event in the folder you have currently selected in the project window. Example:



```
public class MyCustomClass : MonoBehaviour
{
    [SerializeField] private ScriptableEventNoParam _onPlayerDeathEvent;
}
```



To listen to these events when they are fired, you need to attach an **Event Listener** component (of the same type) to your GameObjects.



EventListenerNoParam

Script	EventListenerNoParam
Binding	UNTIL_DESTROY
Disable After Subscribing	✓
Event Responses	
Size	1
Element 0	
Scriptable Event	example_onPlayerDeath (ScriptableEventNoParam)

Response ()

Runtime Only	GameObject.SetActive
Game Over	✓

- **Binding**:
  - Until_Destroy: will register in Awake() and unsubscribe OnDestroy().
  - Until_Disable: will register OnEnable() and unsubscribe OnDisable().

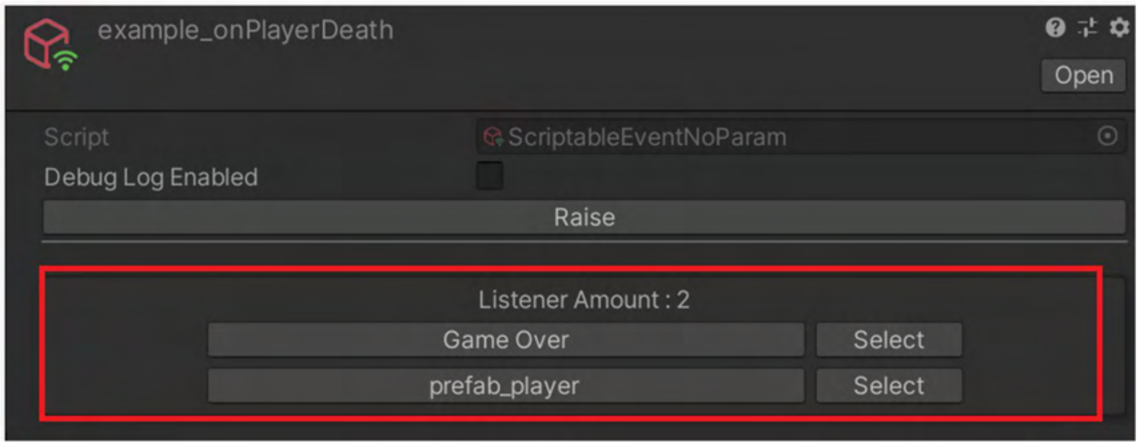
- **Disable after subscribing**: if true, will deactivate the GameObject after registering to the event. Useful for UI elements.

- **Event responses:** here you can add multiple events and trigger things with unity events when they are fired.

You can also register to events directly from code. For more details about this method, please refer to documentation and the scene 4_ScriptableEvents.

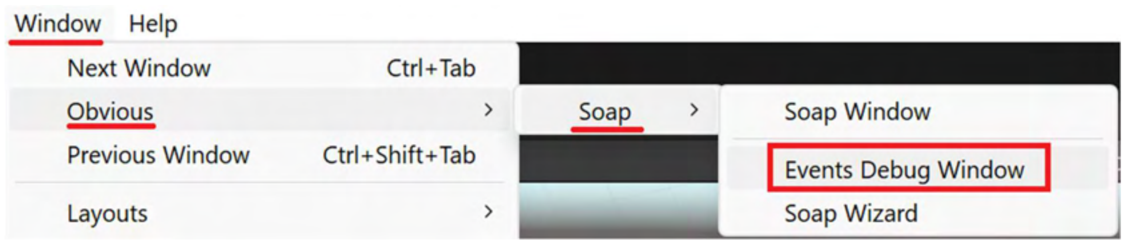
Scriptable events also have their own play mode custom inspector.

When you hit play, you can see all objects that have registered to that event:



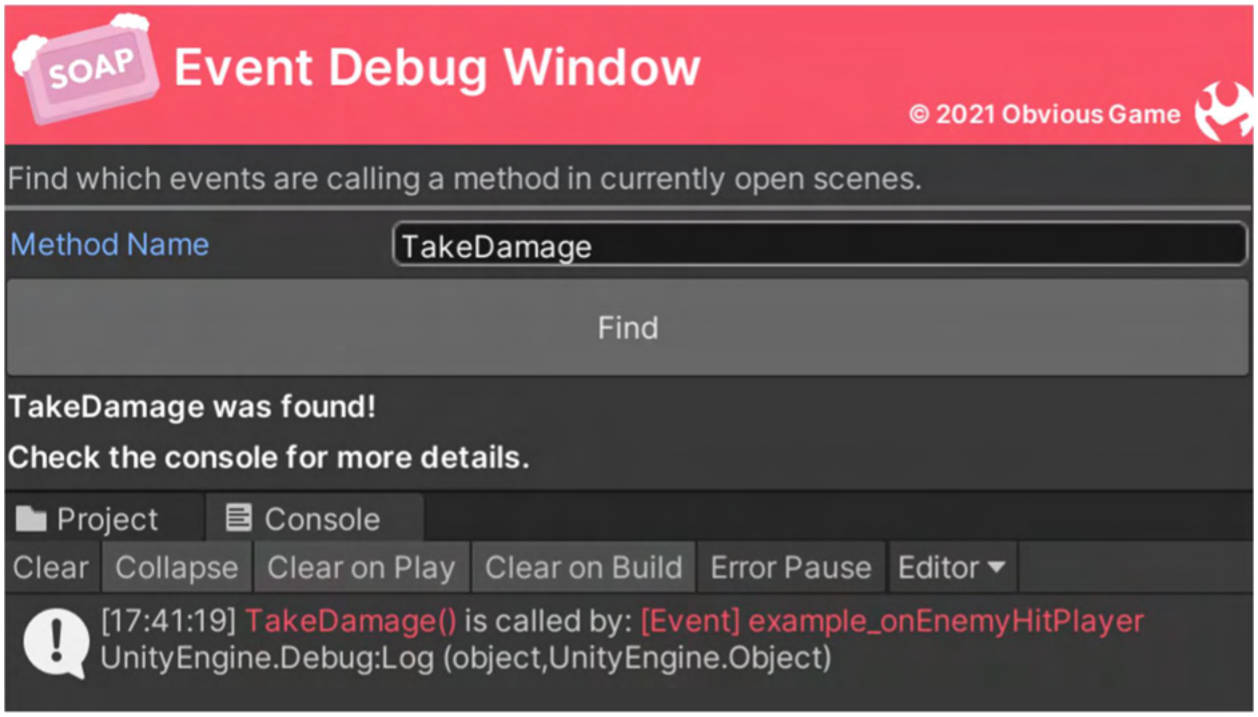
If you have a method in one of your scripts, but you don't remember which event calls that method, you can use the **Events Debug Window.** You can access it through the menu:

Window/Obvious/Soap/Event Debug Window



By typing the name of the method in the event debug window, it will search all objects in your scene and find which event calls that method.

Note: it is case sensitive.



Find which events are calling a method in currently open scenes.

Method Name: TakeDamage

TakeDamage was found!
Check the console for more details.

```
[17:41:19] TakeDamage() is called by: [Event] example_onEnemyHitPlayer
UnityEngine.Debug:Log (object,UnityEngine.Object)
```

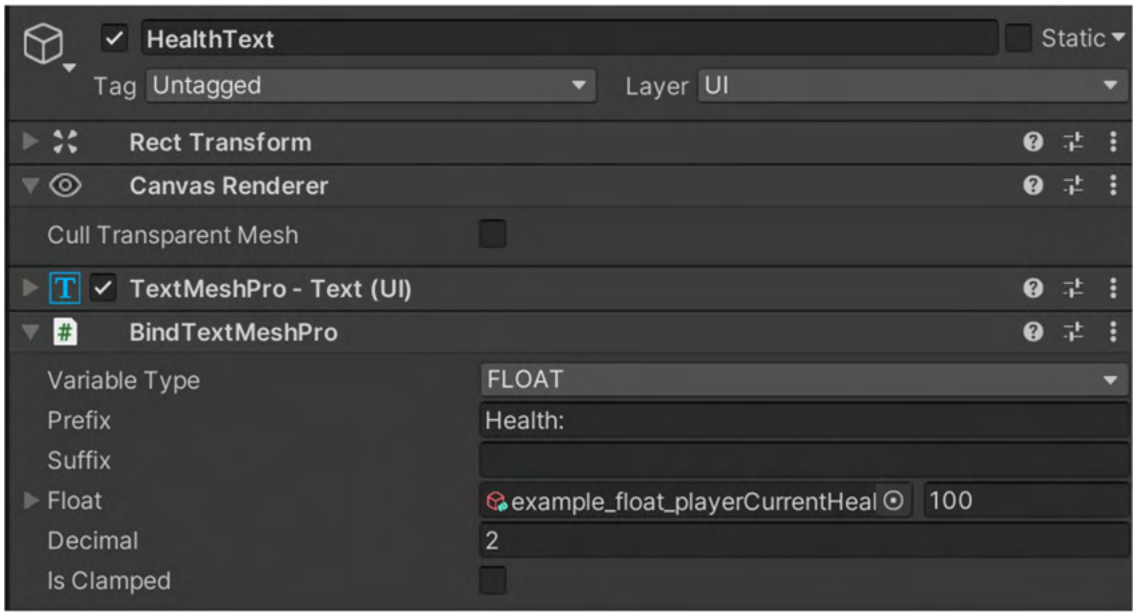
Once the method if found (or not), a message is logged in the console and if you click on it, it will ping the corresponding object in the hierarchy. You can also click on the debug message and check the stack trace for additional information.

Please check the example scene to understand concrete use of scriptable events and how to raise them.

Bindings

Bindings are components that you attach to GameObjects to bind them to a variable so that they execute a simple behavior when that variable value changes. They were created to save time and to not have to create a script for small things like changing the color or text / image and others.

Please check out the Example scene “2_Bindings_Examples_Scene” and its documentation, as it is the best way to understand how to use Bindings and how useful they are.

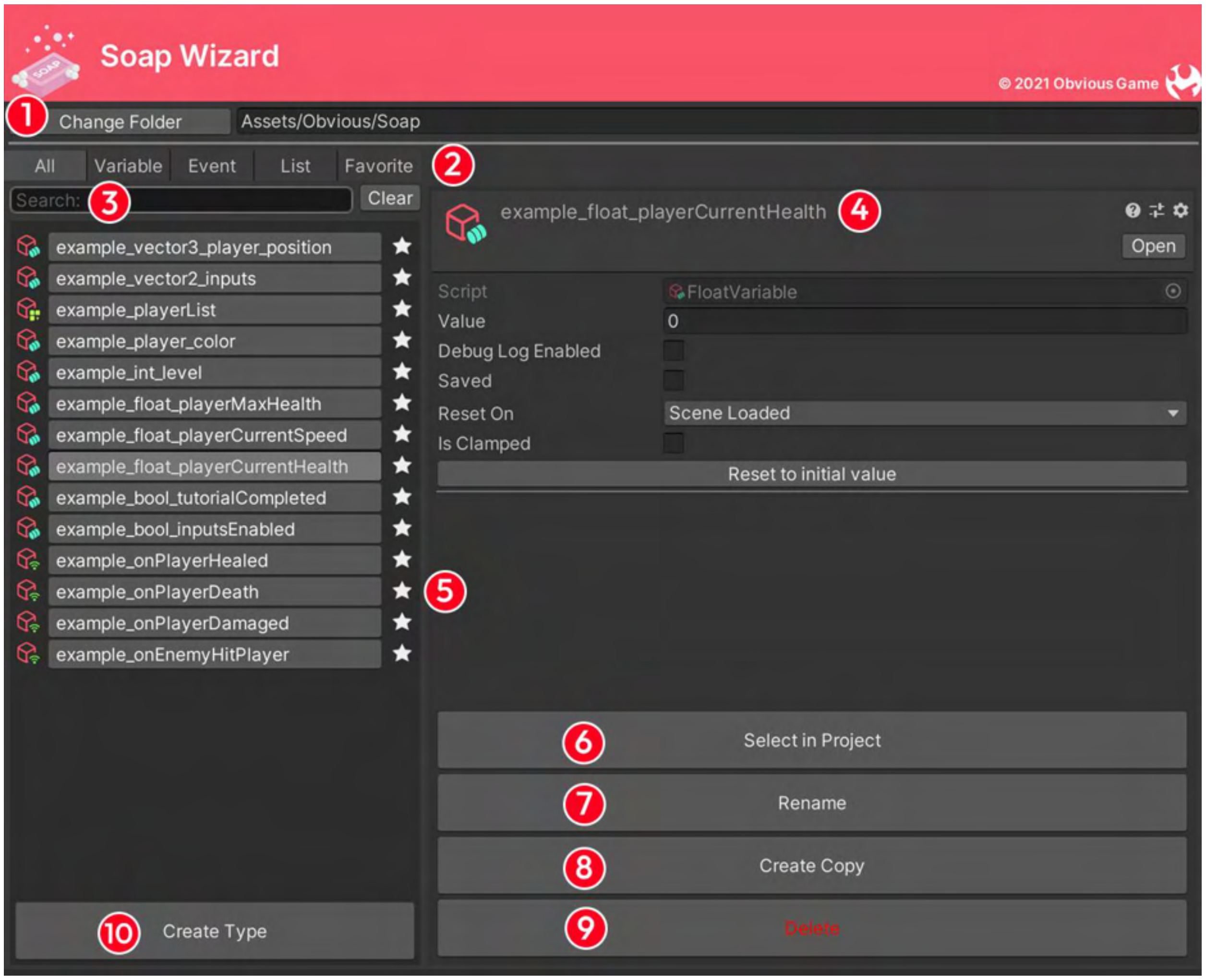


HealthText — Static
Tag: Untagged — Layer: UI
Rect Transform
Canvas Renderer
Cull Transparent Mesh: ☐
TextMeshPro - Text (UI)
BindTextMeshPro
Variable Type: FLOAT
Prefix: Health:
Suffix:
Float: example_float_playerCurrentHeal — 100
Decimal: 2
Is Clamped: ☐

Soap Wizard

The Soap Wizard is a custom window that helps you manage all the Scriptable Objects from Soap in one place. It also provides convenient quality of life features. You can open this wizard from the menu:

Window/Obvious/Soap/Soap Wizard



Assets/Obvious/Soap

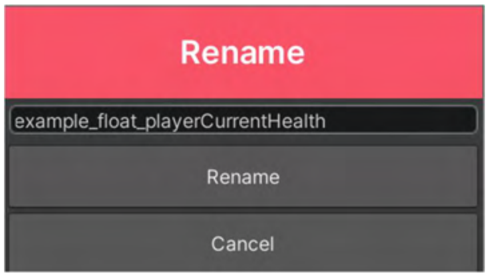
example_vector3_player_position
example_vector2_inputs
example_playerList
example_player_color
example_int_level
example_float_playerMaxHealth
example_float_playerCurrentSpeed
example_float_playerCurrentHealth
example_bool_tutorialCompleted
example_bool_inputsEnabled
example_onPlayerHealed
example_onPlayerDeath
example_onPlayerDamaged
example_onEnemyHitPlayer

example_float_playerCurrentHealth

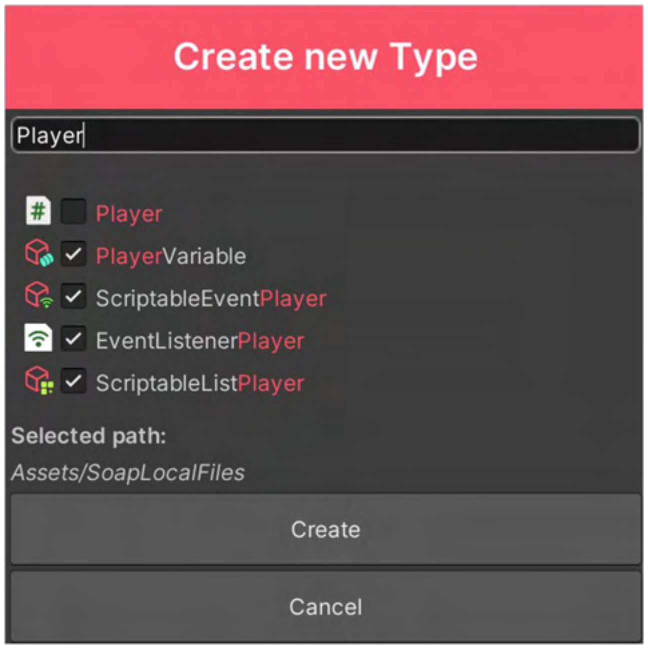
Script FloatVariable
Value 0
Debug Log Enabled
Saved
Reset On Scene Loaded
Is Clamped
Reset to initial value

1. Change Folder: this is where you can set the root folder from which the wizard will find all the scriptable variables, events, and lists. By default, it is set to “Asset”, which means it will populate the wizard with all the Soap scriptable objects in your project.
2. Tabs: click on a specific tab to display the type of scriptable object selected. There are 4 categories: All, Variable, Event and List.

3. Search bar: filter and search for specific scriptable objects. Use the clear button to clear your search quickly.
4. Selected Scriptable: this is where you can see the inspector for the selected scriptable object. It is the same as what it displayed in the inspector view when inspecting an object.
5. Favorite: click here to add this scriptable object to the favorite tab.
6. Select in project: this pings the object in the project view.
7. Rename: opens a pop that lets you rename the object.

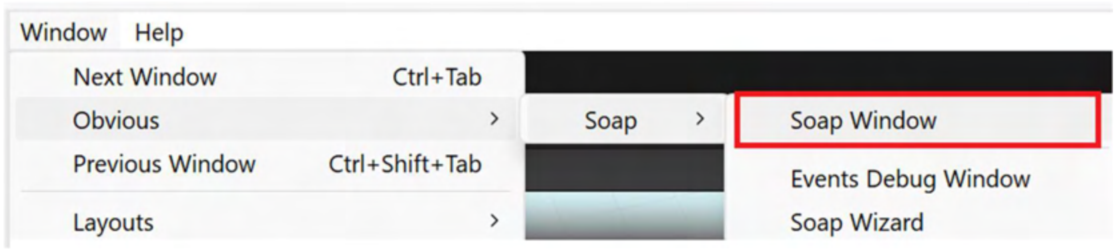


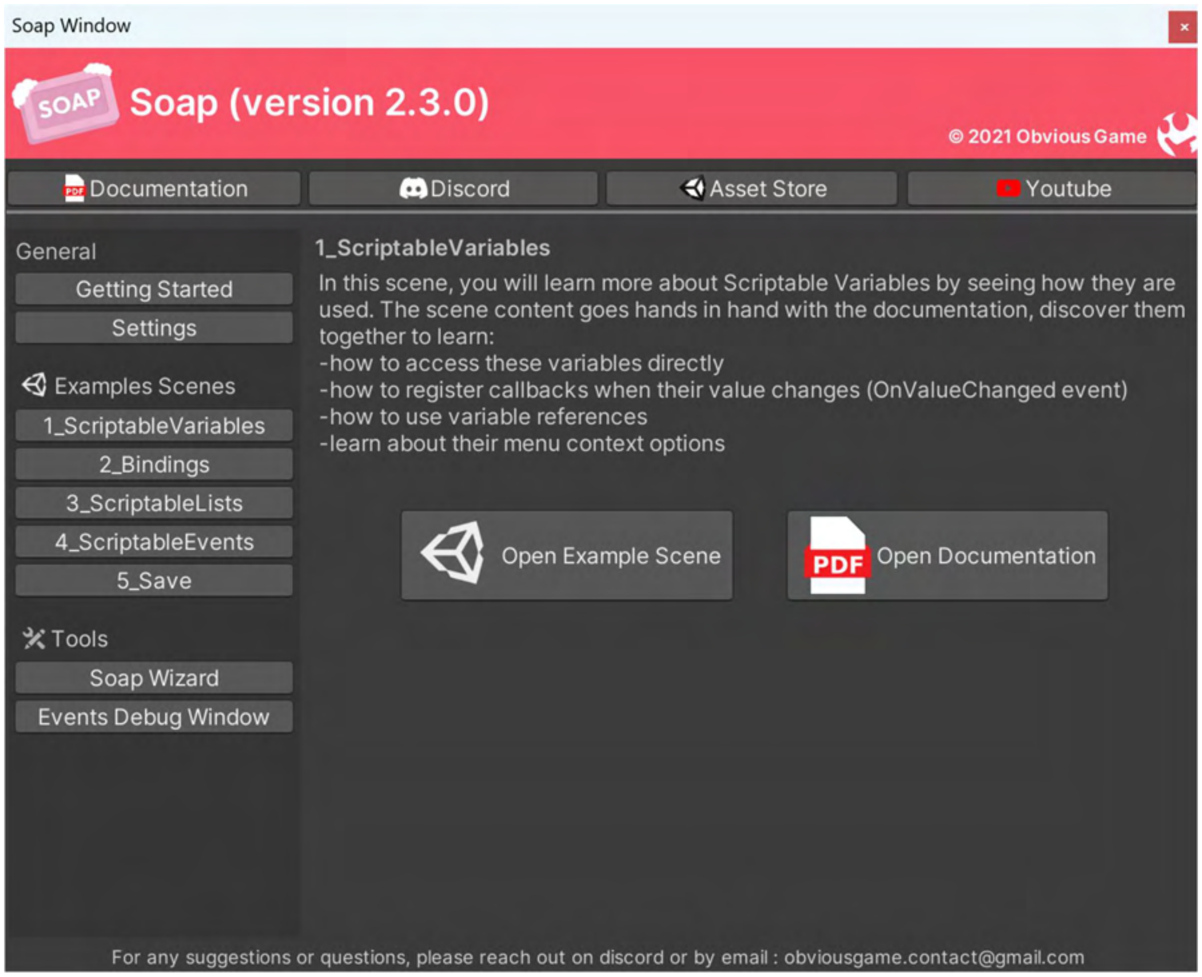
8. Create Copy: this duplicates the selected scriptable object. It is the same as pressing CTRL+D on an object in the project view.
9. Delete: this deletes the selected scriptable object.
10. Create New Type: opens a pop up where you can choose to create a new type of scriptable object. For example, if you want to create a custom variable of type "Player". This will generate the C# classes for you and output them in the selected folder.



Soap Window

The Soap Window appears the first time you import Soap. You can always open it via: *Window/Obvious/Soap/Soap Window*





1_ScriptableVariables

In this scene, you will learn more about Scriptable Variables by seeing how they are used. The scene content goes hands in hand with the documentation, discover them together to learn:
-how to access these variables directly
-how to register callbacks when their value changes (OnValueChanged event)
-how to use variable references
-learn about their menu context options

For any suggestions or questions, please reach out on discord or by email : obviousgame.contact@gmail.com

The Soap Window will provide you with everything useful you need to know about Soap. It works like a Hub to make it easy to find something in one place.

You can access and modify the settings from here.

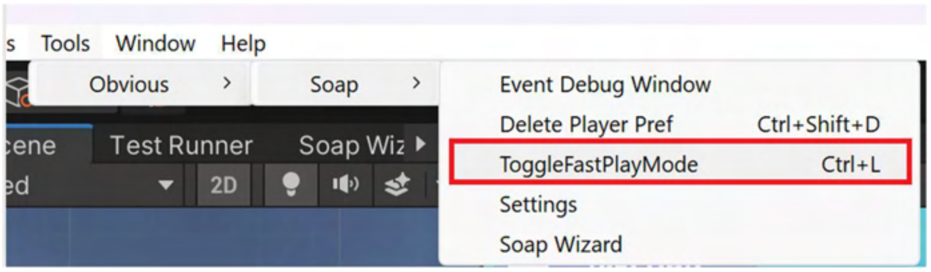
Enabling Editor fast play mode

To enable / disable fast play mode. You can either:

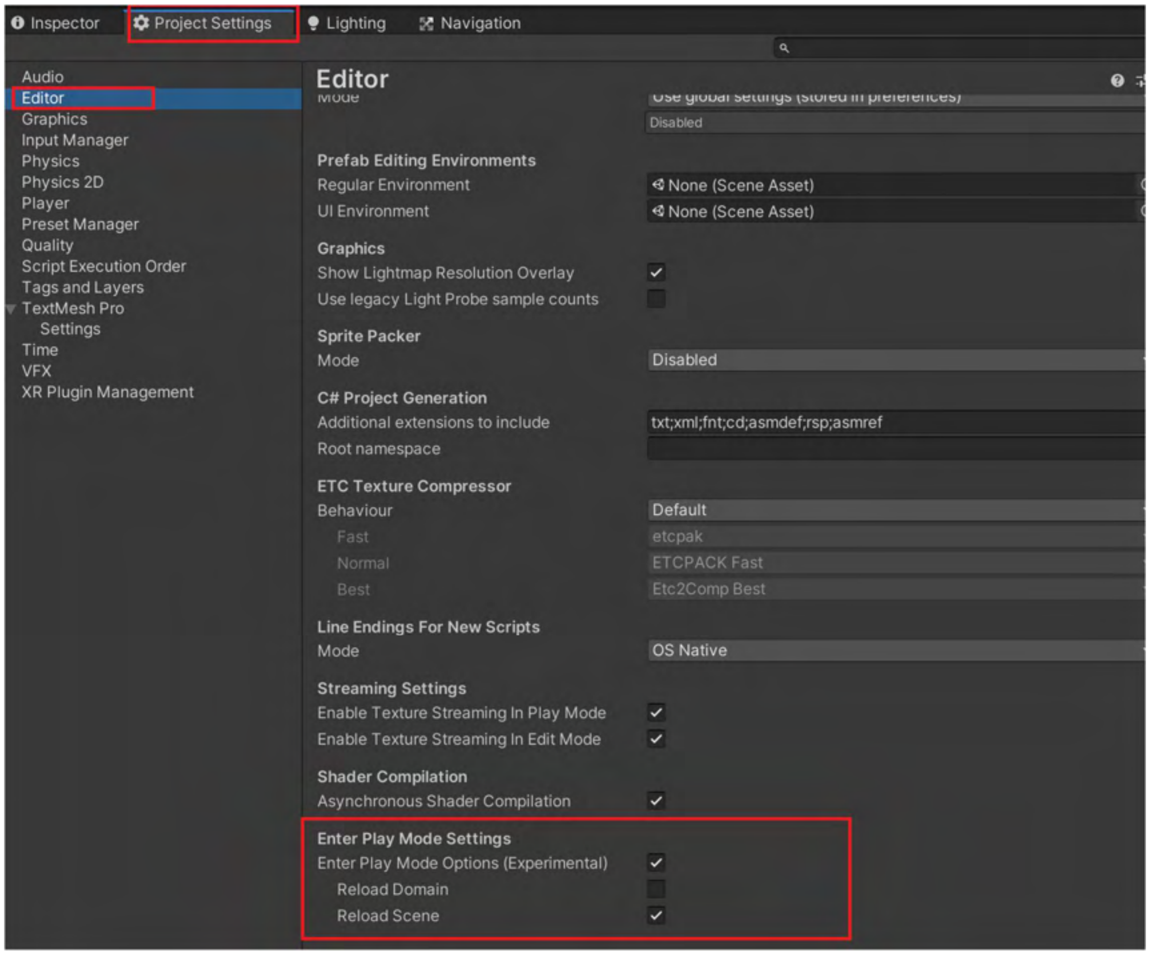
- use the shortcut (CTRL+ L) or Tools menu:

Tools/Obvious/Soap/ToggleFastPlayMode

A message in the console will tell you if its enabled or disabled.



- Manually set it (*Project Settings/Editor/EnterPlayMode Settings*)



Editor

Mode	Use global settings (stored in preferences)
	Disabled
Prefab Editing Environments	
Regular Environment	None (Scene Asset)
UI Environment	None (Scene Asset)
Graphics	
Show Lightmap Resolution Overlay	✓
Use legacy Light Probe sample counts	
Sprite Packer	
Mode	Disabled
C# Project Generation	
Additional extensions to include	txt;xml;fnt;cd;asmdef;rsp;asmref
Root namespace	
ETC Texture Compressor	
Behaviour	Default
Fast	etcpak
Normal	ETCPACK Fast
Best	Etc2Comp Best
Line Endings For New Scripts	
Mode	OS Native
Streaming Settings	
Enable Texture Streaming In Play Mode	✓
Enable Texture Streaming In Edit Mode	✓
Shader Compilation	
Asynchronous Shader Compilation	✓
Enter Play Mode Settings	
Enter Play Mode Options (Experimental)	✓
Reload Domain	
Reload Scene	✓

Make sure to have “Reload Scene” enabled. We want to reload the scene (fast) but not the domain (the slow part).

Note: the choice of toggling and having a shortcut to quickly alternate between fast and normal play mode is because sometimes, some plugins or some components create errors (usually because they are not reset properly). Therefore, by switching quickly to normal play mode and playing once, you can “fix” occasional problems. If something does not work, most of time, simply reloading the domain fixes it. And as soon as it fixed, I usually toggle back to fast play mode.

How to extend this package?

You can extend this SDK by creating your own scriptable variables, lists and events. You can use the Soap Wizard to generate the classes automatically (see Create New Type above) or you can write the classes yourself.

If you go with the second option, simply inherit from these classes, and replace T with your type:

- ScriptableVariable<T>
- ScriptableList<T>
- ScriptableEvent<T>
- EventListener<T>
- VariableReferences<V, T>

Make sure your class is Serializable: add the [System.Serializable] attribute to your class.

You can also create new "Binding" components. I have made a few, but you can be creative and make new bindings that will be useful in your games.

Examples of extending the package can be found in the various example's scripts.

Content

ScriptableVariables
BoolVariable
IntVariable
FloatVariable
StringVariable
Vector2Variable
Vector3Variable
ColorVariable
GameObjectVariable

VariableReferences
BoolReference
IntReference
FloatReference
StringReference
Vector2Reference
Vector3Reference
ColorReference

ScriptableEvents
ScriptableEventNoParam
ScriptableEventInt
ScriptableEventFloat
ScriptableEventString
ScriptableEventVector2
ScriptableEventvector3
ScriptableEventColor
ScriptableEventGameObject

Editor
SoapWizardWindow
EventsDebugWindow
ScriptableVariableGuidGenerator
ShowIfPropertyDrawer
Custom Editors / PropertyDrawers
SoapSettings
SoapWindow
SoapWindowSettings

ScriptableLists
ScriptableListVector3
ScriptableListGameObject
ScriptableListPlayer (Example)

Unit Tests
GuidTest
ScriptableListTest
ScriptableVariableTest
VariableReferenceTest

Bindings
BindText
BindGraphicColor
BindRendererColor
BindToggle
BindSlider
BindComparisonToUnityEvent
BindFillingImage
BindInputField
CacheComponent

Utils
SoapEditorUtils
SoapFileUtils
SoapInspectorUtils
SoapTypeUtils

Compatibility and recommendations

I highly recommend purchasing an asset like FindReference2 in combination with Soap. This will show exactly in which assets any of your scriptable objects is referenced, making it easy to identify where they are accessed or modified. Even without Soap, I highly recommend this kind of asset as it is extremely useful.

Soap is compatible with Odin, Fast Script Reload and most assets. In the future I plan to improve integration with major assets (like Playmaker or Node Canvas).

Note: if you are using NaughtyAttributes, make sure you delete the ObjectEditor.cs script from Soap. NaughtyAttributes has its own script, and it conflicts with the one from Soap. Things should work fine after that 😊 .

Contact

Any questions, suggestions, or feedback?

Feel free to reach me at: obviousgame.contact@gmail.com

Or join the discord server: https://discord.gg/CVhCNDbxF5

Please leave a review on the asset store, as it really helps us to improve the packages and show your support.