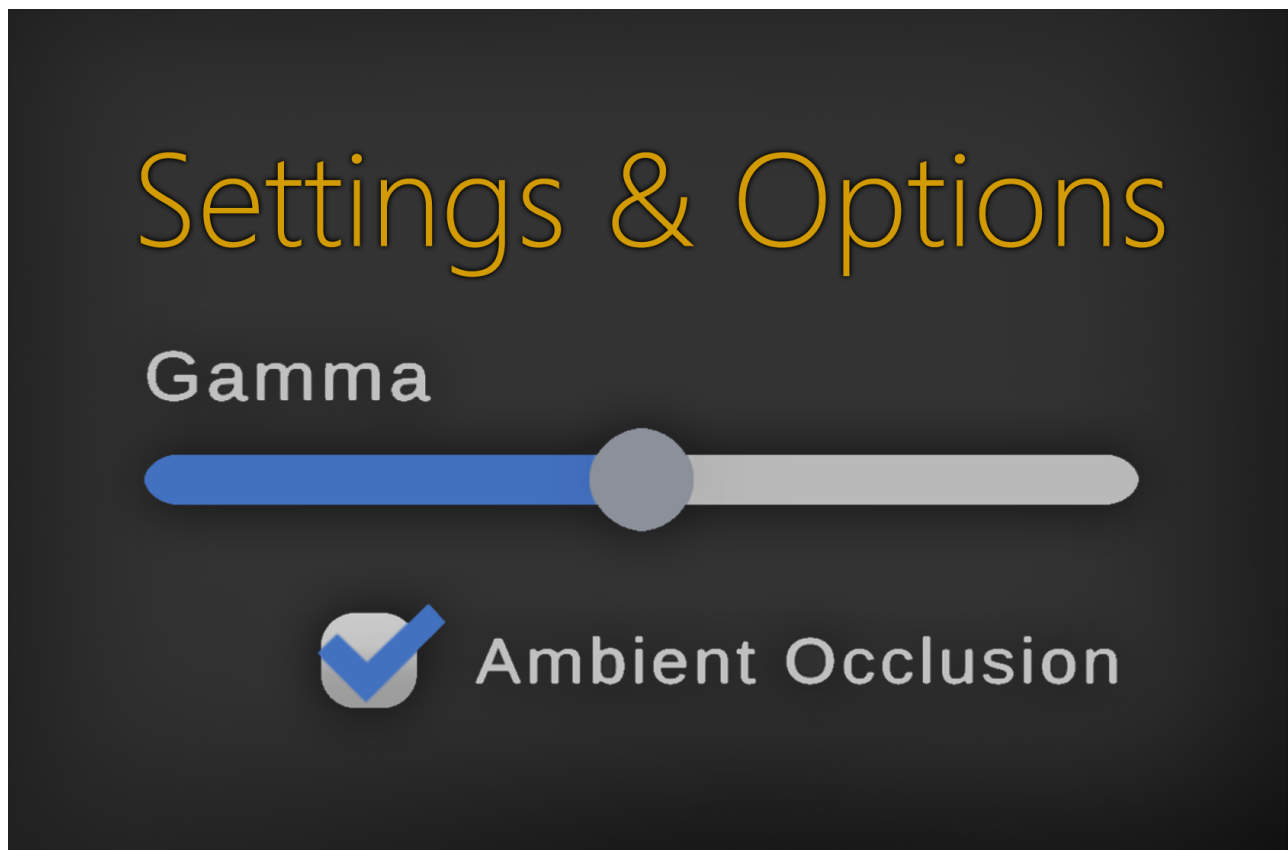


Setting Generator



Generate settings for your game within seconds.

Table of contents

100+ pages of documentation, what the hell?	5
Overview (read this first!)	5
Structure	5
Initialization	6
Accessing Settings	6
Quick Guide	7
Creating Settings and a SettingsProvider	7
Adding UI	8
Make settings do stuff (aka adding code connections)	9
Detailed guide	10
What is it doing exactly?	10
How is it doing it? (aka „How to generate a new setting in UGUI“)	11
What are those IDs?	14
Creating your own Settings and Settings Provider	15
Initializing Settings: When are the settings applied for the first time?	19
Integrating the settings into your own game	21
Step 1: Make sure the settings are initialized (loaded) at the start	22
Settings_INITIALIZER	22

Step 2: Make a copy of the example and clean it up.....	23
Step 3: Rename the resources and scene.....	25
Step 4: Fixing the links inside the provider & choosing a save file key.....	26
Step 5: Integrate the UI into your game.....	27
Step 6: (Optional) Delete unneeded settings.....	33
Combining the Settings System with other Assets.....	34
SettingsApplier.....	35
Settings.....	36
.....	36
Basic Types (Bool, Int, Float, String).....	37
ColorOptions.....	37
Options.....	37
Key Combination (preferred solution for the old InputSystem).....	37
Input Binding (preferred solution for the new Input System).....	37
Setting Resolvers.....	38
Auto-Save.....	39
UI Systems Overview (IMGUI vs UGUI vs UI Toolkit).....	40
IMGUI (not supported).....	41
UGUI (fully supported).....	41
UGUI Components.....	41
Implementations.....	42
Color Picker UGUI & Color UGUI.....	42
Dropdown UGUI.....	43
Headline UGUI.....	44
Input Key UGUI.....	44
Input Binding UGUI.....	44
Options Button UGUI.....	45
Slider UGUI.....	46
Stepper UGUI.....	47
Textfield UGUI.....	48
Toggle UGUI.....	48
UGUI Helpers.....	49
Auto Navigation Overrides.....	49
Selection Event Listener.....	49
Scroll Into View On Select UGUI.....	50
SelectionUGUI.....	50
UI Toolkit (full logic, fewer components, Unity 2021.2+).....	51
UI Toolkit Resolvers.....	52
Linking a Visual Element to a Setting (guide).....	52
Linking a Visual Element to a Localization.....	55
UI Toolkit Components.....	56
DropdownField.....	56
Toggle.....	57
Slider.....	57
TextField.....	57
Helper: UIElementClickEvent.....	58
Helper: UIElementEvents.....	58
Connections.....	59
Ambient Occlusion (SSAO).....	60

Caveats (mostly URP).....	60
Anti Aliasing.....	60
Audio Mixer.....	61
Audio Paused.....	62
Audio Volume.....	62
Audio Source Volume.....	63
How to make sound groups for „effects“, „music“, „voice“, ...?.....	63
Bloom.....	64
Depth Of Field.....	64
Frame Rate.....	64
Full Screen.....	65
GetSetConnection<T>.....	65
Gamma.....	65
InputBindingConnection.....	65
Motion Blur.....	65
Monitor.....	65
Quality.....	66
Refresh Rate.....	66
Resolution.....	66
Shadow.....	67
Shadow Distance.....	67
Shadow Resolution.....	67
Texture Resolution.....	67
Vignette.....	67
V-Sync.....	68
Window Mode.....	68
Input Rebinding (Using the new Input System).....	69
Rebinding Overview.....	69
Rebinding Tutorial.....	70
1) Generating the InputBinding Connections.....	71
2) Link InputBinding Connections to Settings.....	72
3) Configuring the Rebinding UI.....	72
Working with Auto-generated script code for Actions.....	74
Scripting API (Adding custom Settings).....	76
How to get access to the API if you are using Assembly Definitions.....	76
How pull the value from a setting.....	77
How to get notified of a value change in a setting.....	78
Using the GetSetConnection<T>.....	79
Making a custom Connection class.....	81
The IConnectionWithSettingsAccess interface.....	83
Adding custom save & load methods.....	84
Localization.....	86
Third Party Localization Support (example: I2 Localization).....	87
Localizing Key Control Paths.....	88
Visual Scripting (formerly BOLT).....	89
Configuration & Debugging.....	91

Input Validation.....	92
Validation via OnChange events.....	92
Validation via Connection objects.....	93
Upgrading from older versions.....	96
Actions to perform after updating to fix example scenes.....	96
Third Party Integrations.....	97
Rewired Integration.....	98
Rewired Integration Tutorial.....	99
InControl Integration.....	103
DLSS Connection (by Alterego Games).....	104
FSR 2 Connection (by Alterego Games).....	107
Common Issues / FAQ.....	109
Post Processing Settings do nothing.....	109
Post Processing Render Context Null Pointer.....	109
Post Processing not visible on mobile (Built-In render pipeline).....	109
Controller X is not working properly (Old Input System).....	109
The example scenes are all pink after updating the asset (URP or HDRP).....	109
InvalidOperationException: You are trying to read Input using	110
The Post Processing effect is not found.....	111
I have upgraded from an older version but don't see the new examples?.....	113
Rebinding input actions has no effect. Why?.....	113
How to add a „Mouse Sensitivity“ setting?.....	114
There are some errors in URP or HDRP Connections but I am not using URP or HDRP?!.....	115
My settings work in the menu but not in the level?.....	116
DLSS: I have a multi camera setup and the DLSS setting does only apply to the camera tagged with „MainCamera“.....	117
I can't get mouse interaction to work with the settings prefabs?.....	117

100+ pages of documentation, what the hell?

First, thank you for opening the manual. I know it looks daunting.

The reason for this long documentation is that in here you will find explanations for all the options (there are a lot) and pretty much any edge case I have heard of.

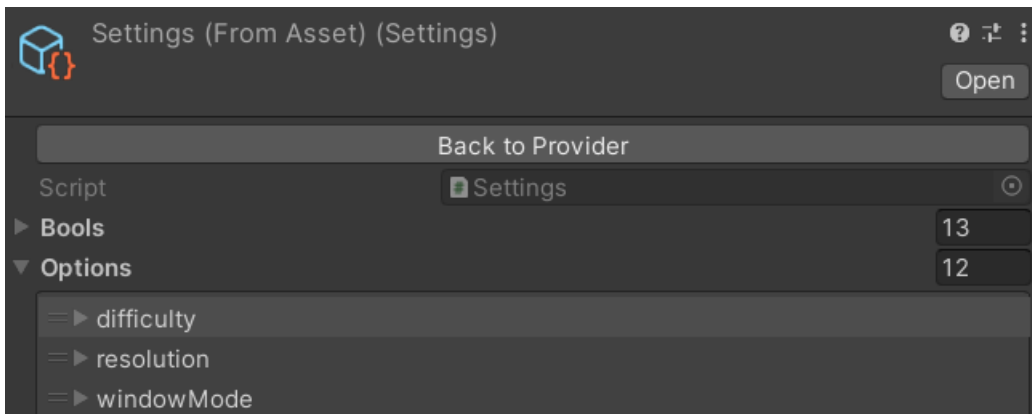
Please read the „Overview“ and the „Quick Guide“ to get started.

Overview (read this first!)

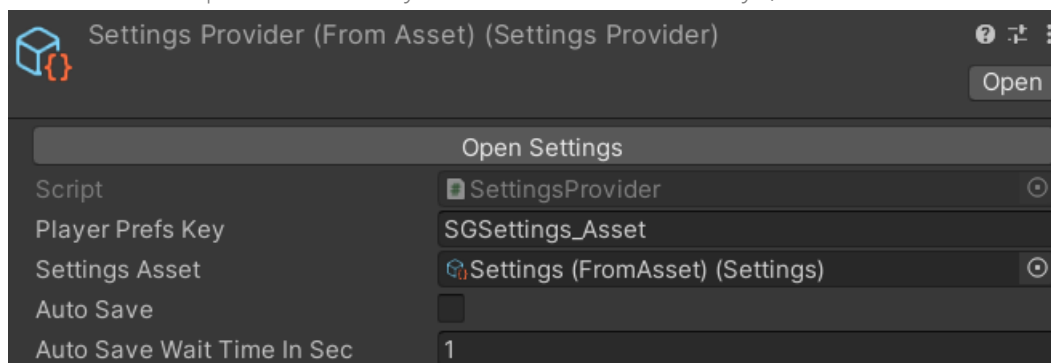
This overview and the following quick guide will get you set up as quickly as possible. This section does not explain all the details. For the details check out the „Detailed Guide“ below.

Structure

Settings are configured in Scriptable Objects (SO). The main one is called **Settings**. It contains a list of all your settings.



Access to this „Settings“ object is managed by another SO called the **SettingsProvider**. There are multiple reasons why we do it this indirect way (more in the detailed guide).



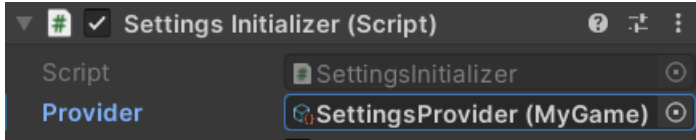
Game ↔ SettingsProvider ↔ Settings (list of settings) ↔ Data (PlayerPrefs/JSON)

Each setting can be either pure data (just a number for example) or it can have a dynamic connection to some code (i.e. changing the value will trigger some code). The connections are again Scriptable Objects and are called „Connections“ (surprise).

Initialization

You initialize the Settings at the very start by using the **SettingsInitializer Prefab**. Settings are saved in the PlayerPrefs by default (JSON or custom methods can be used too).

The initializer requires one reference, which is the SettingsProvider.



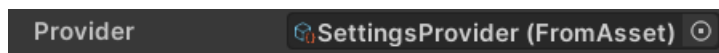
IMPORTANT: **Do NOT access the settings in Awake()**. The settings code needs to wait for Unity's stuff to initialize in Awake() before it can apply the saved settings. **Use Start()** instead.

Accessing Settings

You can access the settings in three ways:

1. The recommended way is to add a public SettingsProvider reference to any MonoBehaviour that needs access to the settings:

```
public SettingsProvider Provider;
```



2. If you use the SettingsInitializer then you can also use the static API:
3. If you know your settings have already been initialized then you can use the Provider's static API. Though, this is the least recommended way.

```
SettingsProvider.LastUsedSettingsProvider.Settings
```

Getting a setting value can be done by pull (you ask for the value) or push (the value is pushed to a method).

Pulling a value from a setting:

```
var settings = SettingsInitializer.Settings;  
SettingFloat volume = settings.GetFloat(id: "volume");  
Debug.Log( volume.GetFloatValue() );
```

Registering a callback to receive value changes:

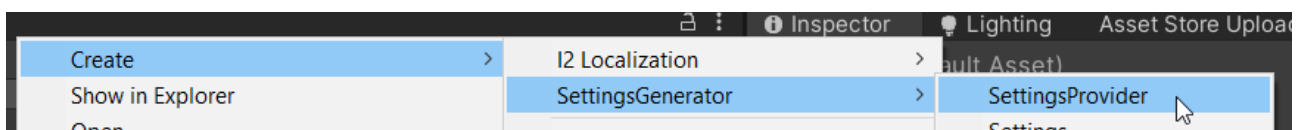
```
Settings settings = SettingsInitializer.Settings;  
settings.GetFloat("volume").OnSettingChanged += (ISetting setting) => {  
    Debug.Log( setting.GetFloatValue() );  
};
```

Quick Guide

You may have noticed that the asset comes with some example scenes. These scenes use premade Settings and SettingsProviders. You can (but shouldn't) use those in your game. Be aware that if you use them you will have to be very careful if you upgrade the asset in the future since these example assets will be overwritten by any update and your changes will be lost. Therefore **I strongly recommend that the first thing you do is to make your own Settings and SettingsProvider.**

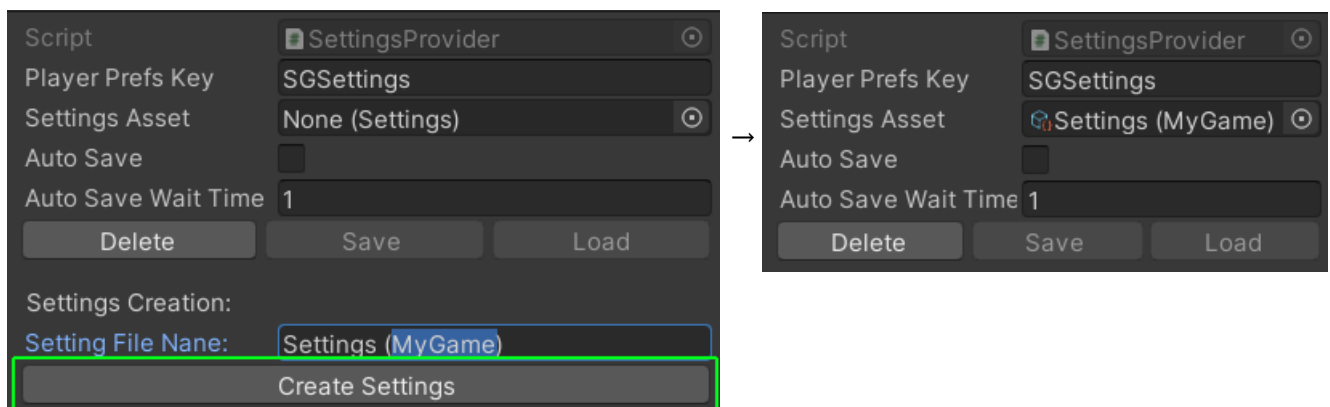
Creating Settings and a SettingsProvider

Right-Click in your project window at the location where you want to store your setting configs. Choose **Create > SettingsGenerator > SettingsProvider**:

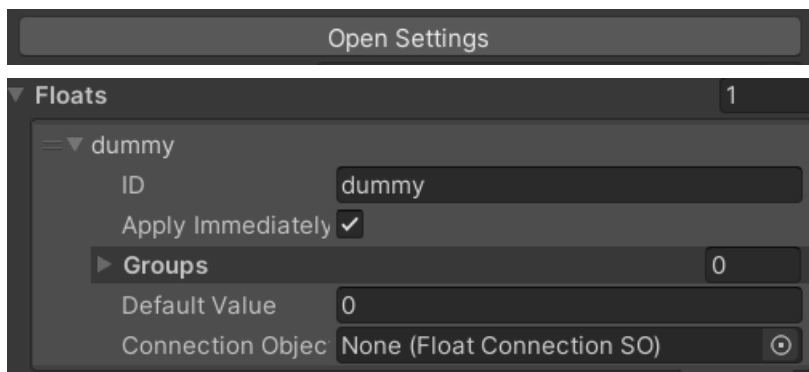


 **SettingsProvider (MyGame)** ← Name your provider so you can find it easily in the future.

By default the provider is empty. You need to add some settings to it.



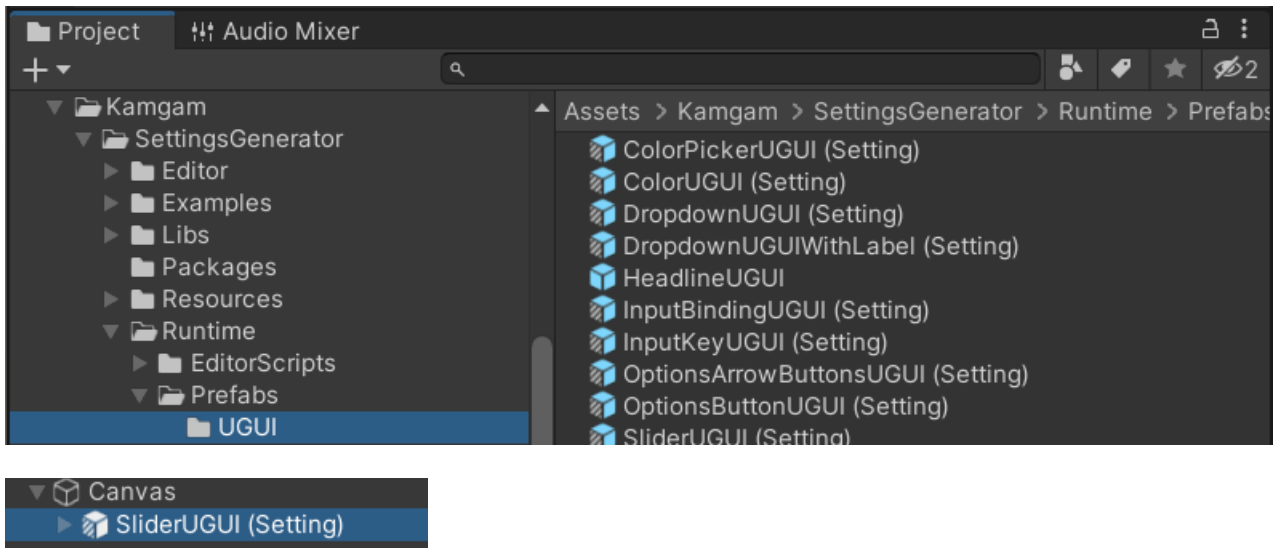
Now if you open the settings you will find a float setting added with the id „dummy“:



Each setting must have a unique ID. The ID is used to reference the setting in the UI:
SettingsProvider + ID = global reference to setting

Adding UI

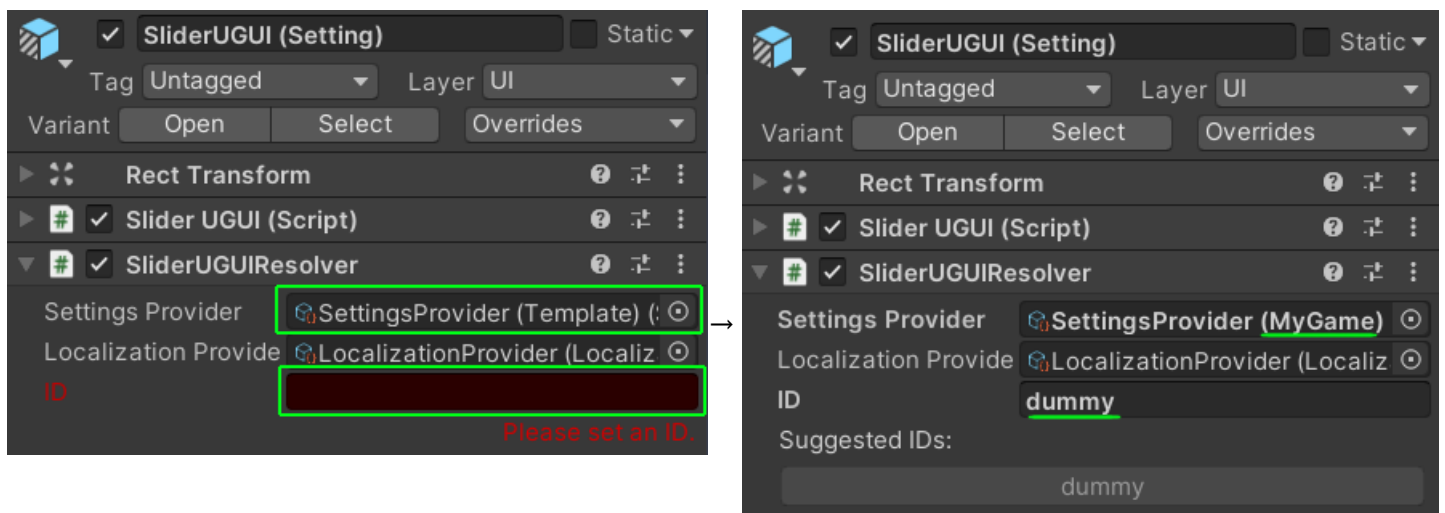
There are some UI prefabs that can be used. Let's try the slider for the „dummy“ setting.



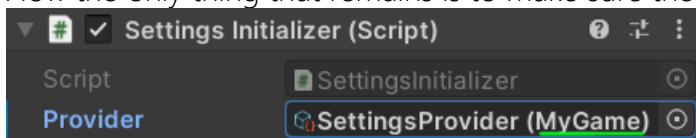
Once you have dragged in the „SliderUGUI (Setting)“ Prefab you have to configure two things to find the setting:

1. The SettingsProvider
2. The Setting ID

Together the Provider and the ID uniquely identify a setting.



Now the only thing that remains is to make sure the Initializer also references the right provider:

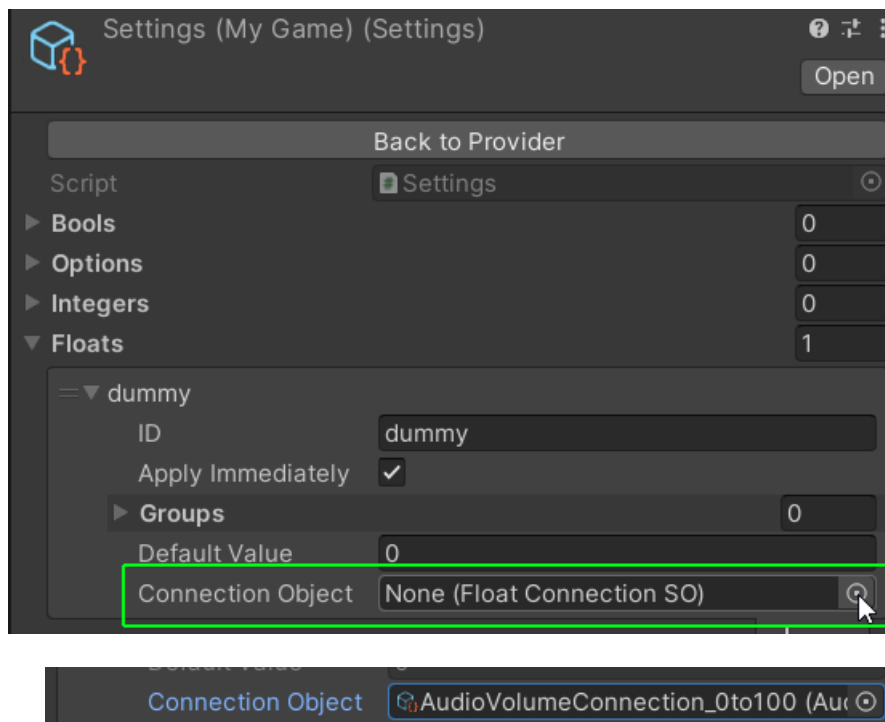


And that's it. Now you can listen for changes in the setting value:

```
var settings = SettingsInitializer.Settings;  
settings.GetFloat(id: "dummy").OnSettingChanged += (ISetting setting) => {  
    Debug.Log(setting.GetFloatValue());  
};
```

Make settings do stuff (aka adding code connections)

The „dummy“ setting we have made before does not yet do much. In order for it to execute some code we need to hook it up with a Connection object.



Once you have added a reference to a connection the setting will not only contain a plain float value but it will also change the volume (or do what ever that connection is doing if the settings value changes).

Check out the connections in the detailed documentation below (there are a lot of them).

Detailed guide

Please read this whole section.

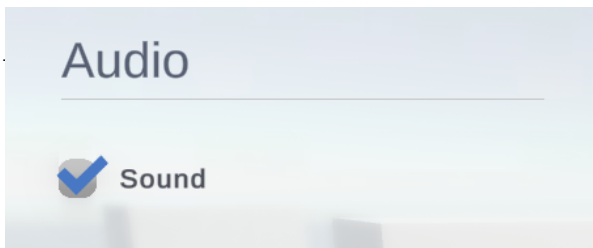
I promise it will be easy to understand and it will help you A LOT with getting into the settings system. Go grab a cup of coffee or a tea. This will take a while :-)

There are multiple parts that play together in making the settings work.

First let's explore what it is doing and then see how it is done.

What is it doing exactly?

The settings system allows you to hook up some code with UI. The code it hooks up with are things that are often used in game settings. Like whether or not the audio is paused for example. A toggle UI might be a good fit for that. It looks like this:



And hooks up to this code:

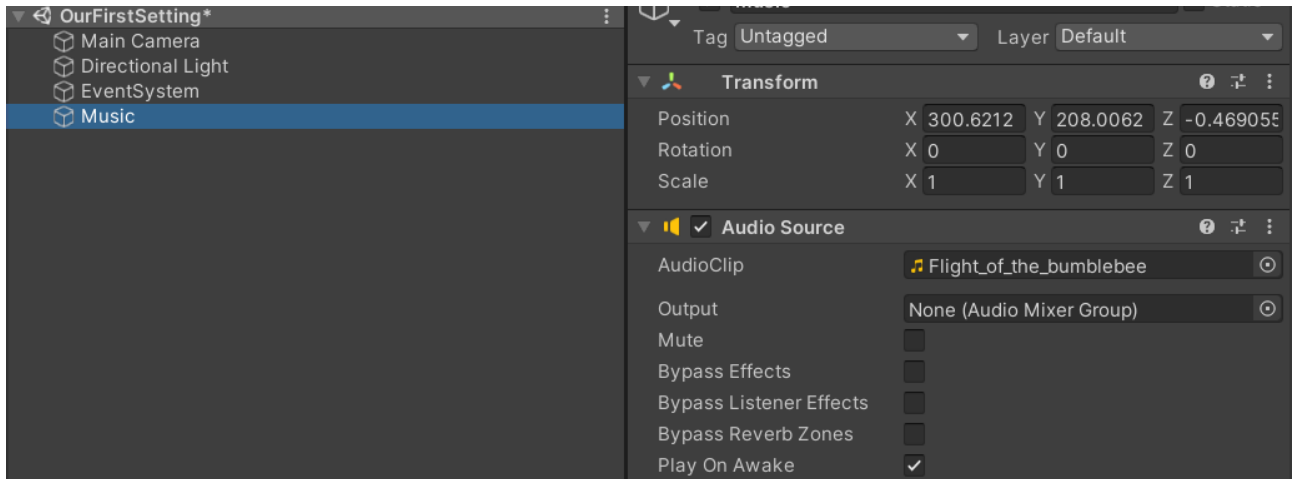
```
public override void Set(bool audioPaused)
{
    AudioListener.pause = audioPaused;
}
```

Don't worry if you are not a programmer. The system has many code assets predefined for you and you can hook them up without ever touching any code.

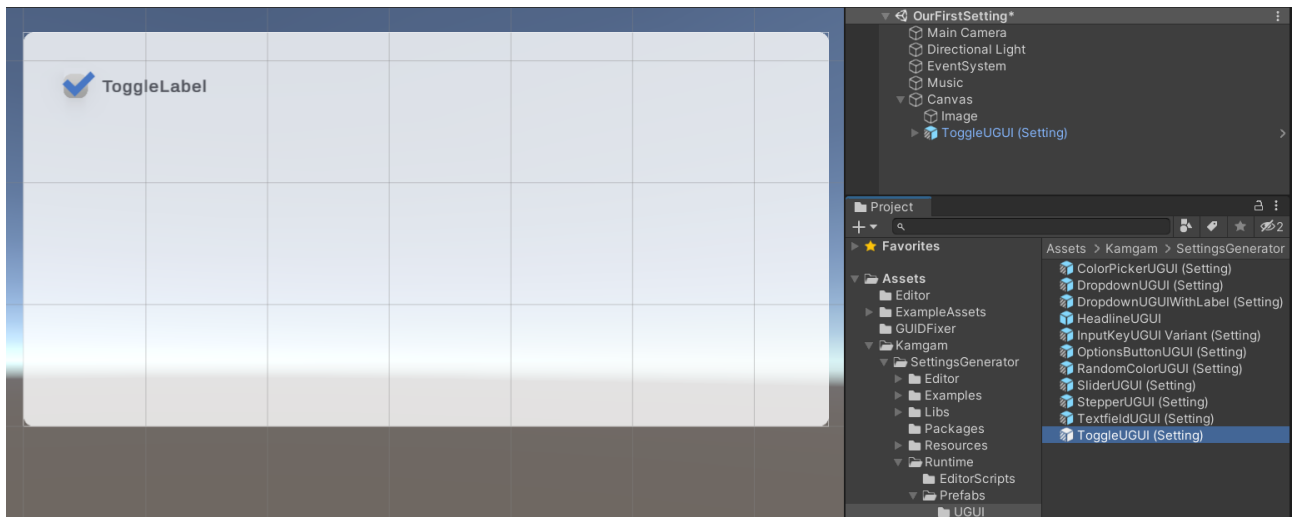
How is it doing it? (aka „How to generate a new setting in UGUI“)

Let's create that audio setting we have just seen (in UGUI).

First we need some music to play so we can hear whether or not our audio setting is working. Let's create a „Music“ game object and add an Audio Source to it. There is a nice recording of the classical „Flight of the bumblebee“ in the examples (make sure „Play On Awake“ is enabled).

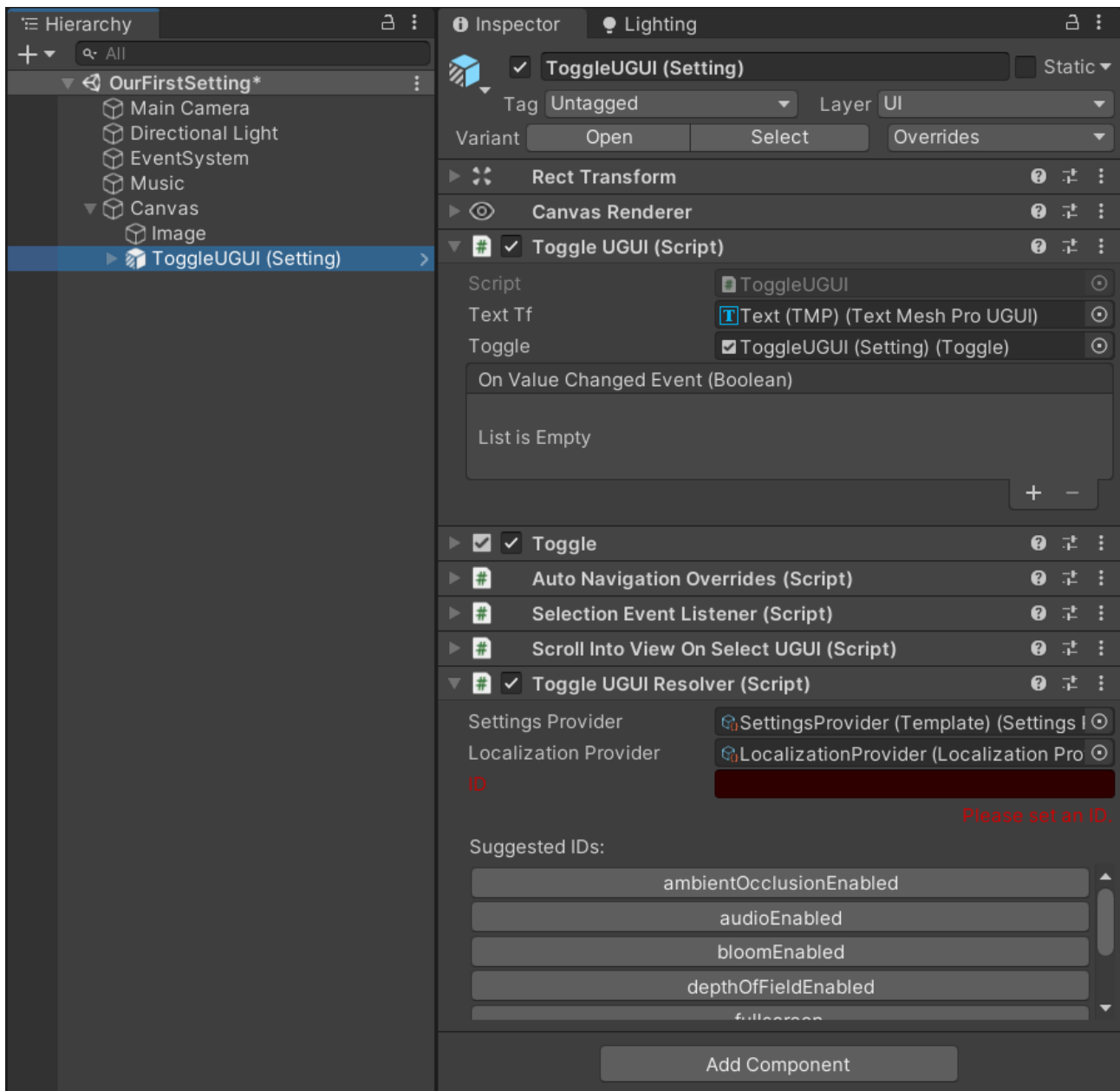


Okay, now first we need a UI. For this demo we will use the „ToggleUGUI (Setting)“ prefab. You can of course use your own UI too. The Prefabs are located under **„Runtime/Prefabs/UGUI“** within the Settings Generator folder.

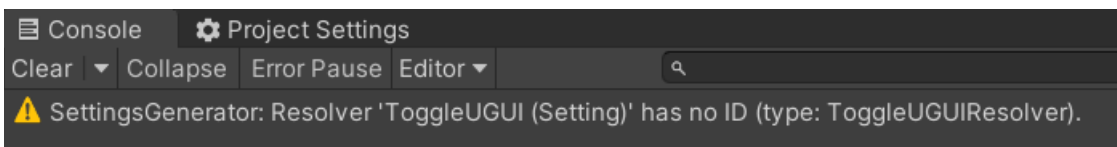


If you select the toggle you will see a LOT of components in the inspector. Most of them are just for convenience and we can ignore them for now. The most important one is:

„Toggle UGUI Resolver“.



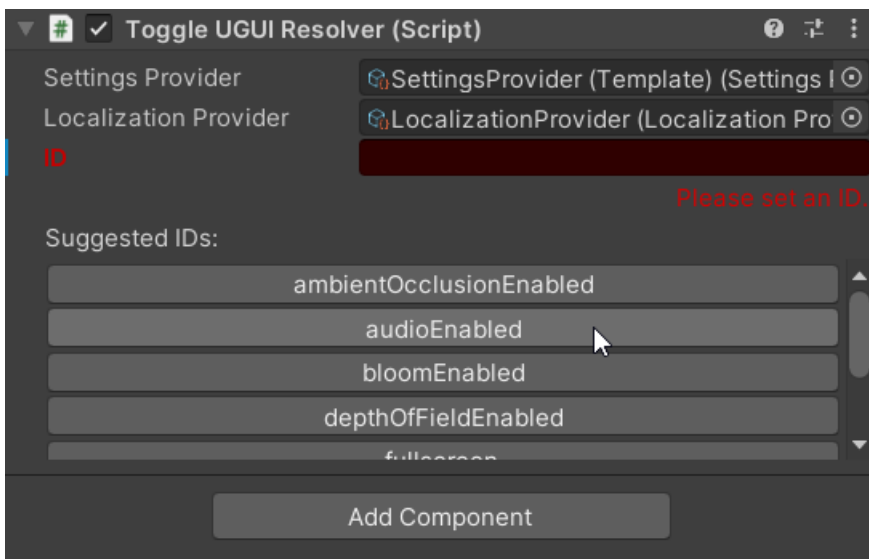
Before we make any changes let's just run our little scene and see what happens.



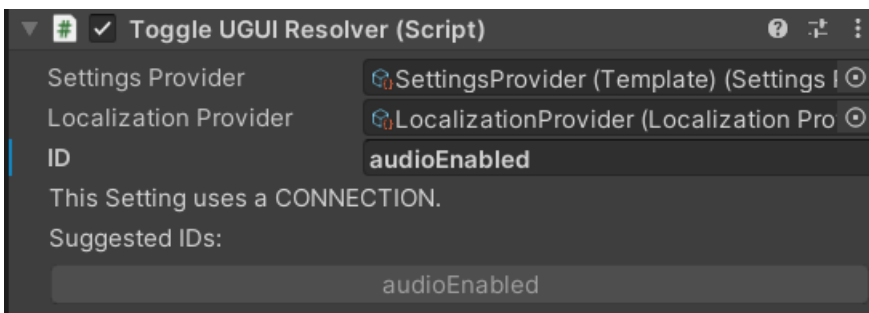
Okay, the settings tool tells us something about our newly added toggle ui is wrong. It complains that it has no ID. Now this is important. **IDs are how the UI finds the setting it belongs to.**

You may have guessed that we need an ID from the glaring **red** warning shown in the Toggle Resolver.

The **Toggle UGUI Resolver** is the one component which connects the UI (the **Toggle UGUI** in our case) to a setting.



But how do we know which ID we need to use? Well, the settings resolver already has a list of suggestions for us. But how is it getting those? What are they? We'll answer these questions later. For now let's just choose the „audioEnabled“ ID and then test our demo again.



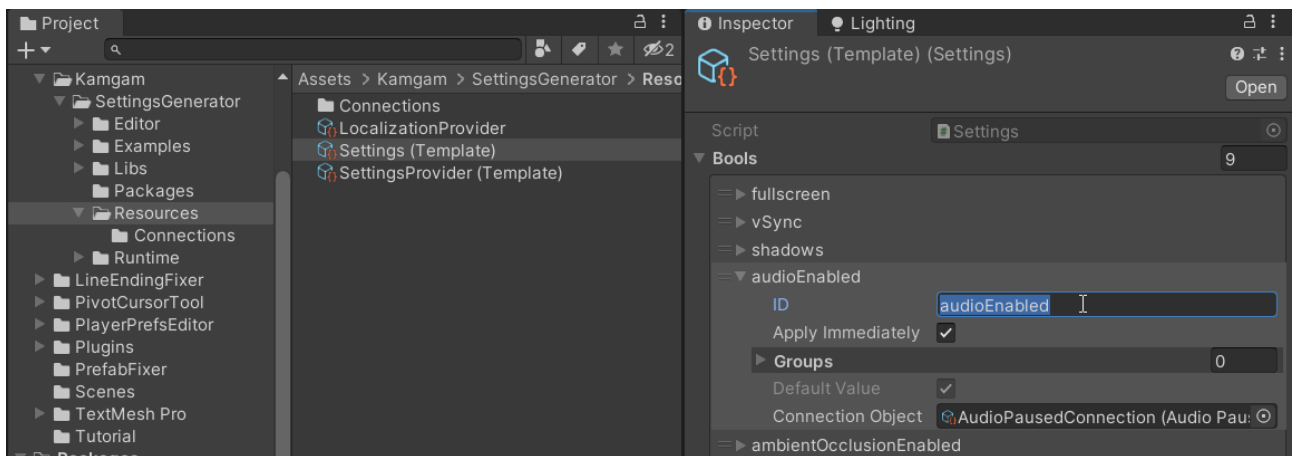
Okay great, the warning is gone and if we hit the toggle it pauses/unpauses the audio playback.

What are those IDs?

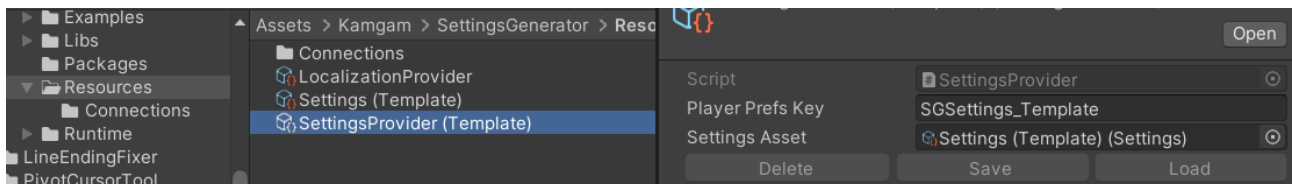
An ID is a unique name for a setting (so we can find it easily). The settings are defined within a **Settings.asset** file (surprise).

There is a „Settings (Template).asset“ which contains all the predefined settings. You can find it within the **Resources** folder. You can either rename and change the template or duplicate it and use the new one. Keeping the template around is the recommended workflow as it allows you to go back and look into it for reference.

The settings are grouped into types. The **audioEnabled** setting can be found under **Bools** as it is a boolean (on/off).

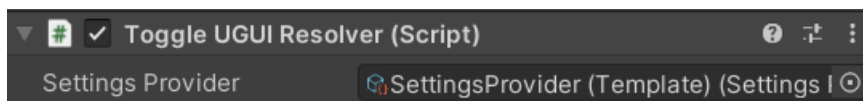


Before we dive into the details of the „audioEnabled“ setting itself let's first talk about the other important asset within the Resources folder, the **SettingsProvider**:

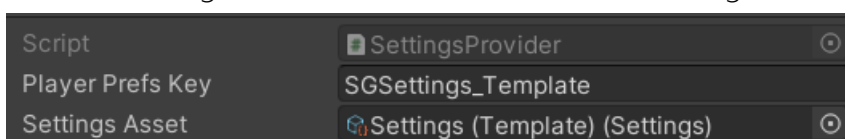


The **SettingsProvider** is the object that connects the settings with any object that is interested in it (it „provides“ the settings to anyone asking for them).

If you look back at the **Toggle UGUI Resolver** from before then you will notice that it has a reference to the SettingsProvider:



And the SettingsResolver has a reference to the Settings.asset

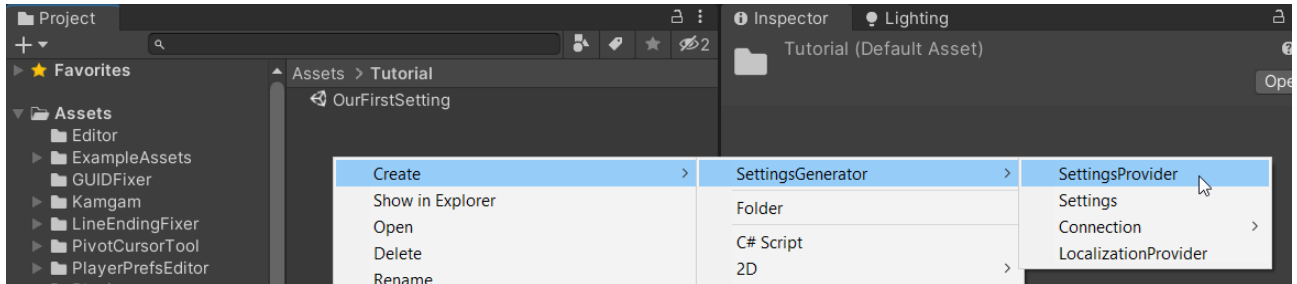


So the connection is: **Toggle UGUI Resolver** > **SettingsProvider** > **Settings** > Setting (ID)

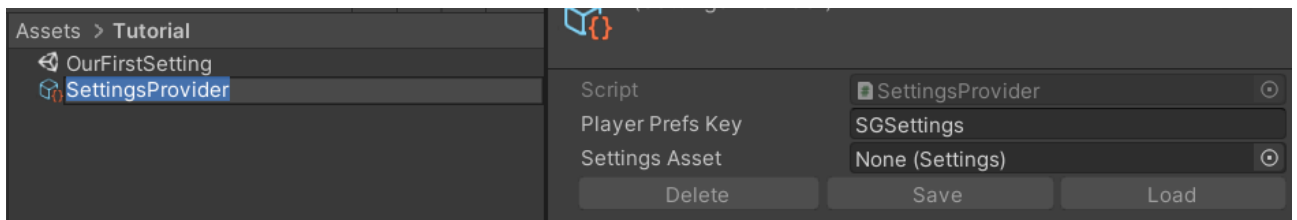
Creating your own Settings and Settings Provider

Usually we would simply copy the „Settings (Template)“ asset and delete the settings we don't need. But to show you how it works we'll do it the manual way (create everything from scratch).

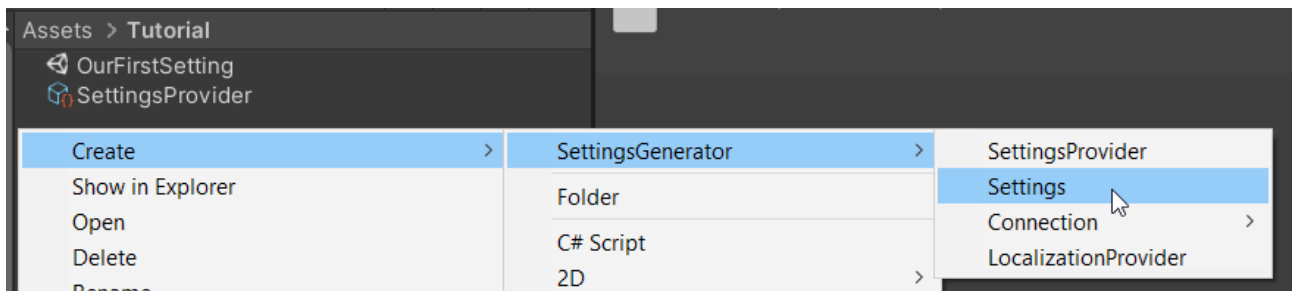
Let's go back to our audio scene and make our very own Settings and SettingsProvider. You can do that via „**Right Click > Create > SettingsGenerator > SettingsProvider**“.



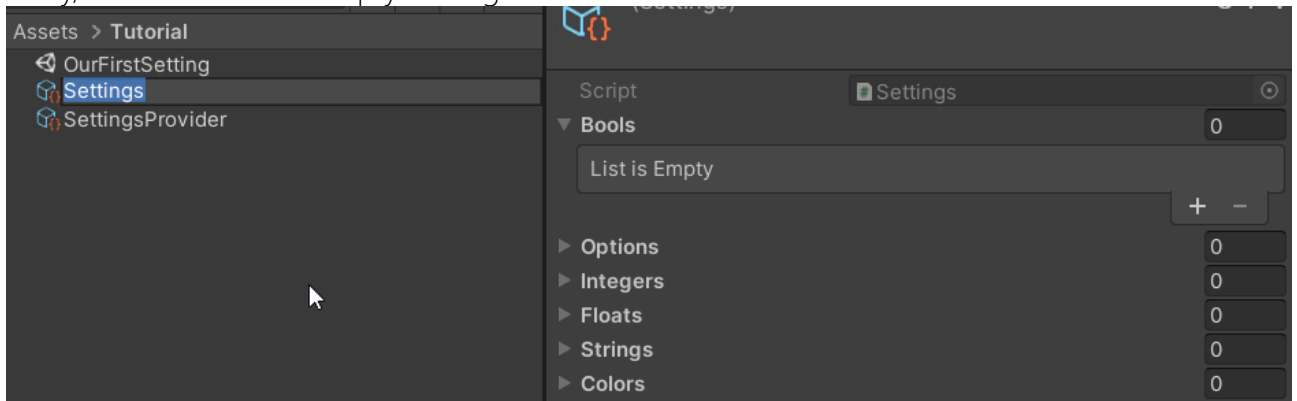
The result will be a new SettingsProvider without a Settings asset:



Now we also create a new Settings object via: „**Right Click > Create > SettingsGenerator > Settings**“.



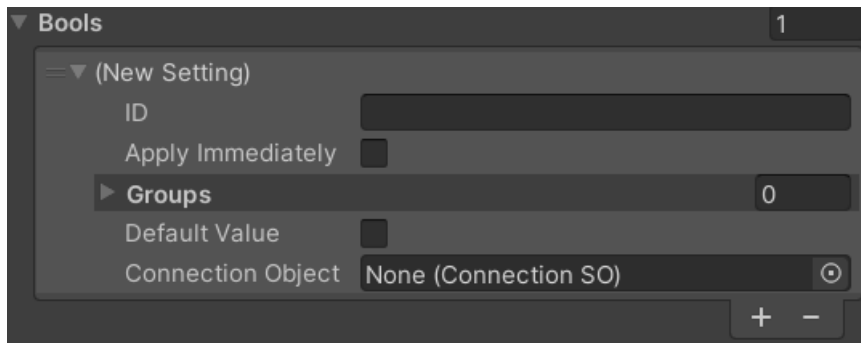
Okay, now we have an empty Settings asset:



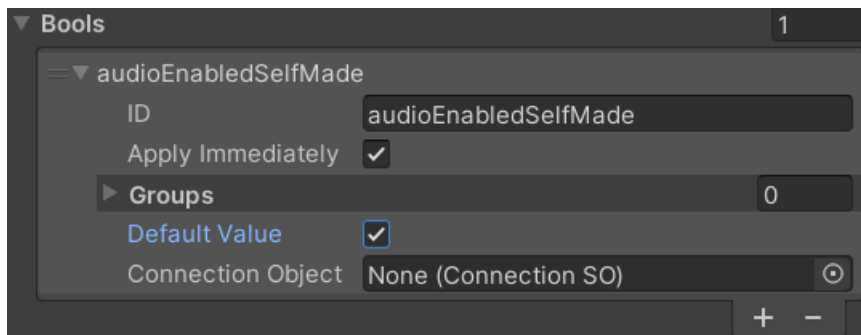
Let's add our own „audioEnabled“ setting to the Settings asset. To do that click on the „+“ at the bottom of the Bools list. It will look like this:



Click on the arrow „>“ left to „(New Setting)“.



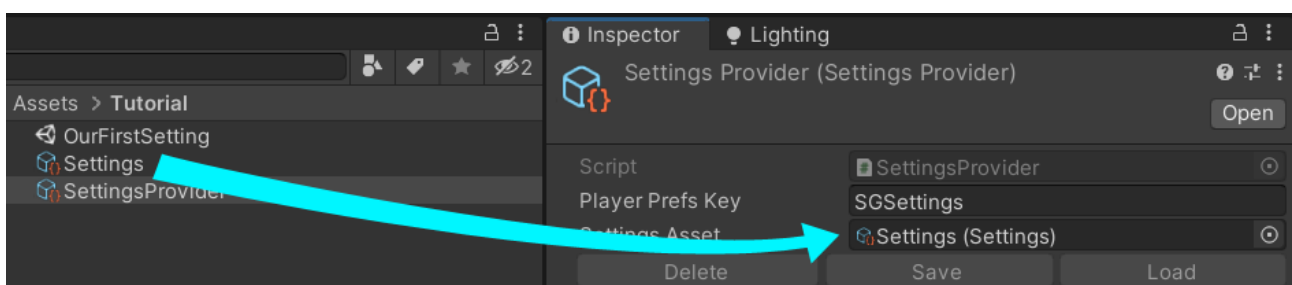
Now we enter our ID „audioEnabledSelfMade“ and check the „Apply Immediately“ and „Default Value“ checkboxes.



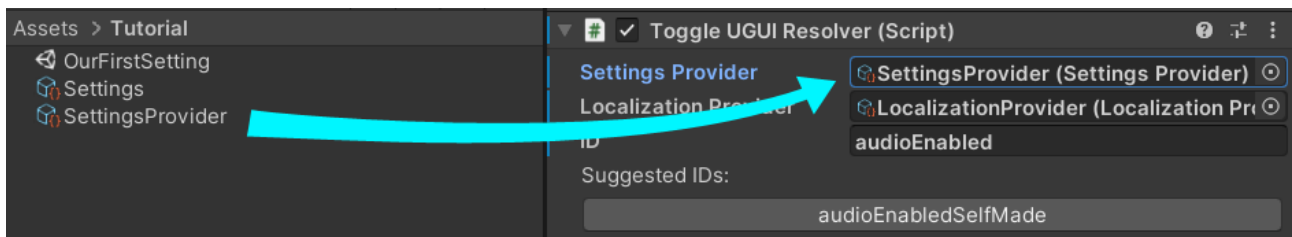
Okay, we have made our first setting (yay).

We still have to hook it up though (remember the „UGUI Resolver > SettingsProvider > Settings > Setting (ID)“ part).

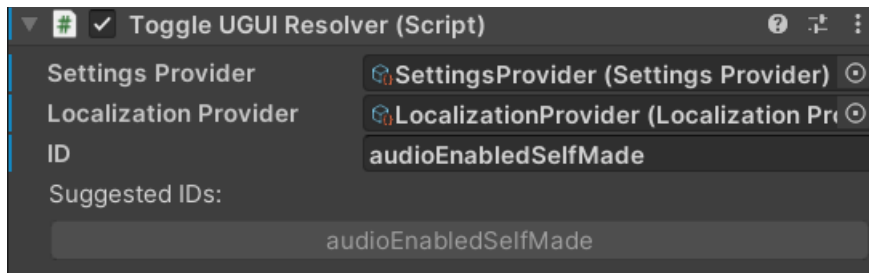
The first thing to do is to hook the Settings up with our SettingsProvider. Simply drag the settings into the „Settings Asset“ field of the provider.



Next we have to let our „ToggleUGUI Resolver“ know that it should use our new SettingsProvider.



Notice how the „Suggested IDs“ change to the ID we have added to our own Settings asset. Update the ID to use our new setting ID.

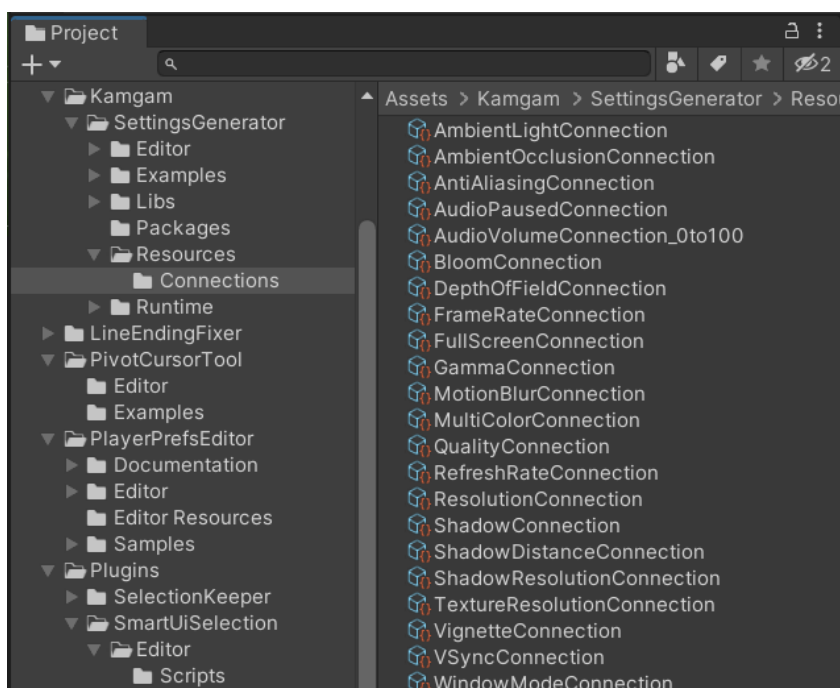


Okay, ready? Let's start our demo scene again.

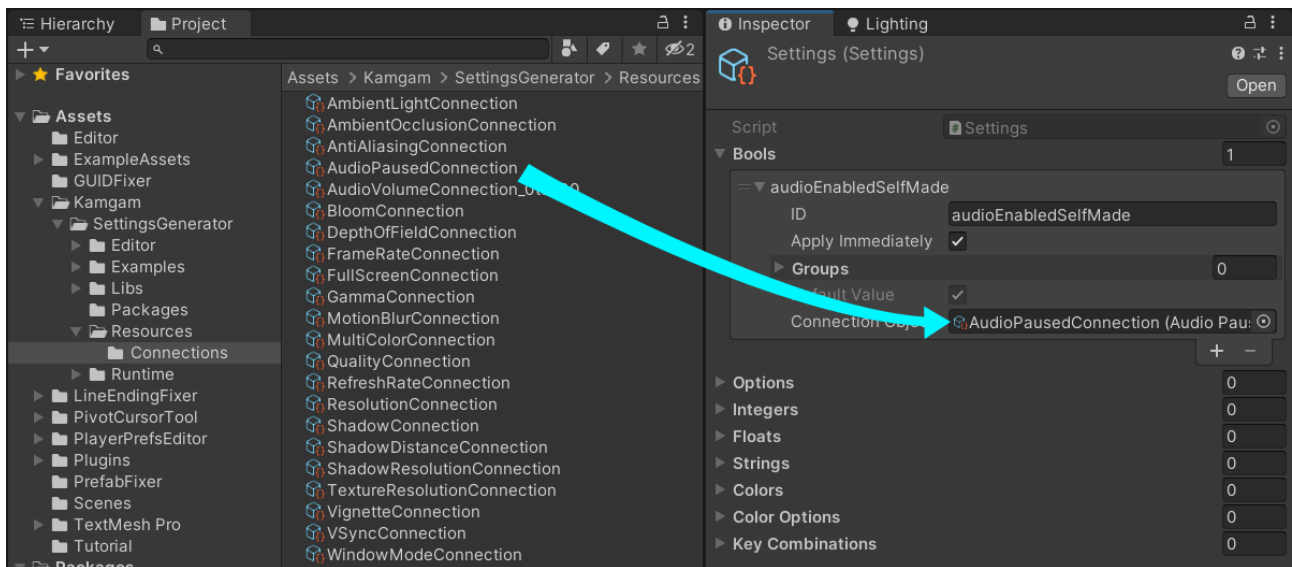
Aaaand it does nothing ?? And there is no warning shown. How disappointing :-/

Well I admit, I tricked you. There is one last step. We have created a new setting „audioEnabledSelfMade“ BUT that settings is not yet hooked up to any code (therefore it does not affect the audio volume). After all, until now it's just a boolean with a name. How was the settings system supposed to know what we wanted it to do?

To hook it up with some code we will use a CONNECTION. Connections are objects representing code. You can assign one connection to every setting. You can find the connections within the „**Resources/Connections**“ folder.



Let's go back to our Settings asset and drag in the „**AudioPausedConnection**“, like this:



Okay, now let's try that demo again.

Ah finally, it's working. Well congratulations, you have created your first setting from scratch.

If you want to see a more sophisticated setup in action then open the „Examples/FromAsset“ demo.

If you are a programmer and you want to do it all in code then look into the „Examples/FromCode“ demo.

If you are interested in the new [InputSystem](#) and how input binding works then look into the „Examples/InputSystemBinding“ demo (please read the „[Input Binding](#)“ section first).

Initializing Settings: When are the settings applied for the first time?

The settings are automatically loaded at the very first use of the Settings object. But when is that exactly? It's when you access the `SettingsProvider.Settings` property for the first time.

But again, when does that happen? It happens whenever you show (set active) one of the UI elements for the first time (when you use one of the UI resolvers to be precise).

But most games don't immediately show the settings at the start so how do we initialize the settings without showing the UI? Well, all you need is just one line of code on a game object in your very first scene. The „`InputSystemBindingDemo.cs`“ is a nice example:

```
public class InputSystemBindingDemo : MonoBehaviour
{
    public SettingsProvider Provider;

    public void Awake()
    {
        // You may think about adding it to Awake(). DON'T (explanation below).

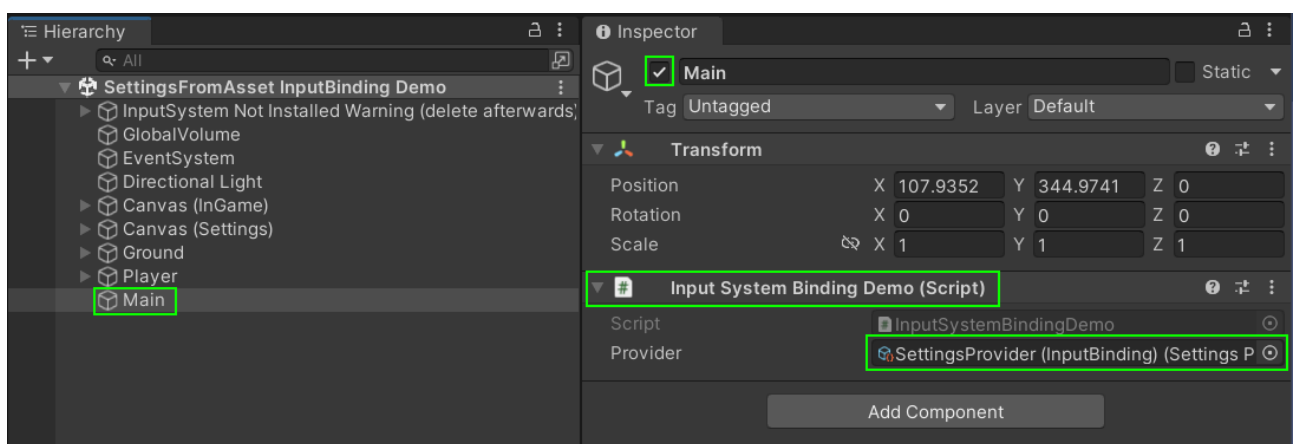
        // If you want some custom SAVE and LOAD methods then you will have to
        // register them BEFORE the settings are initialized. For this Awake() is
        // the right spot to execute.

        // Settings.CustomLoadMethod = customLoad;
        // Settings.CustomSaveMethod = customSave;
        // Settings.CustomDeleteMethod = customDelete;
    }

    public void Start()
    {
        // We have to call the settings system at least once to initialize the load.
        var _ = Provider.Settings;
    }
}
```

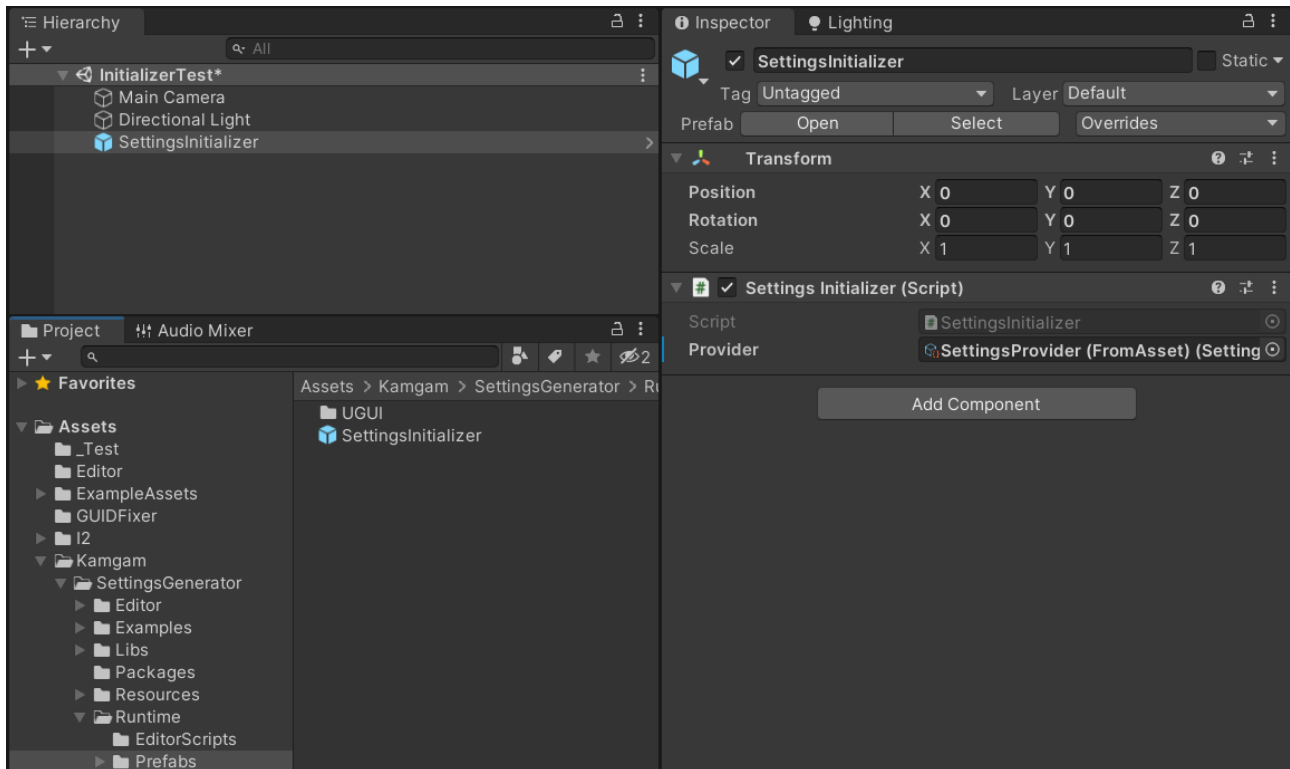
The key code part is the „`Provider.Settings`“. This will init the settings system (if not yet initialized).

The other important part is that the game object you place it on has to be active and a you need to have a SettingsProvider linked, like this:



Be sure to link the right SettingsProvider if you have multiple.

Hint: There is a SettingsInitializer Prefab under ../Runtime/Prefabs.



Do not add the init code to Awake() !**

Here is why:

Some systems, like the post-processing volumes, are initialized in Awake(). If you init the settings in Awake() too then no one knows which one gets executed first (the order is not defined).

If the settings system gets initialized first, then it will not find the Post-Pro volumes (they are not there yet). This will lead to „There was no active '...' PostPro effect" warnings*.

However, if you put your settings init code in Start(). Then the volumes will already be initialized and everyone is happy.

* There is an FAQ about how to set up the post-pro volumes. It also includes infos about some circumstances when the warning will be shown even if everything is properly set up (when you load your scenes asynchronously).

** In the past the recommendation was to init the settings in Awake(). Some of the demos still do.

If for some reason you must initialize the settings in Awake() then please use the [DefaultExecutionOrder(int.MaxValue)] attribute to make it execute LAST.

Integrating the settings into your own game.

Above you have learned how to create your own settings from scratch. While it is important to know how to do this in order to understand how the settings system works it is also a bit tedious.

In this section you will learn how to integrate the examples into your existing project.

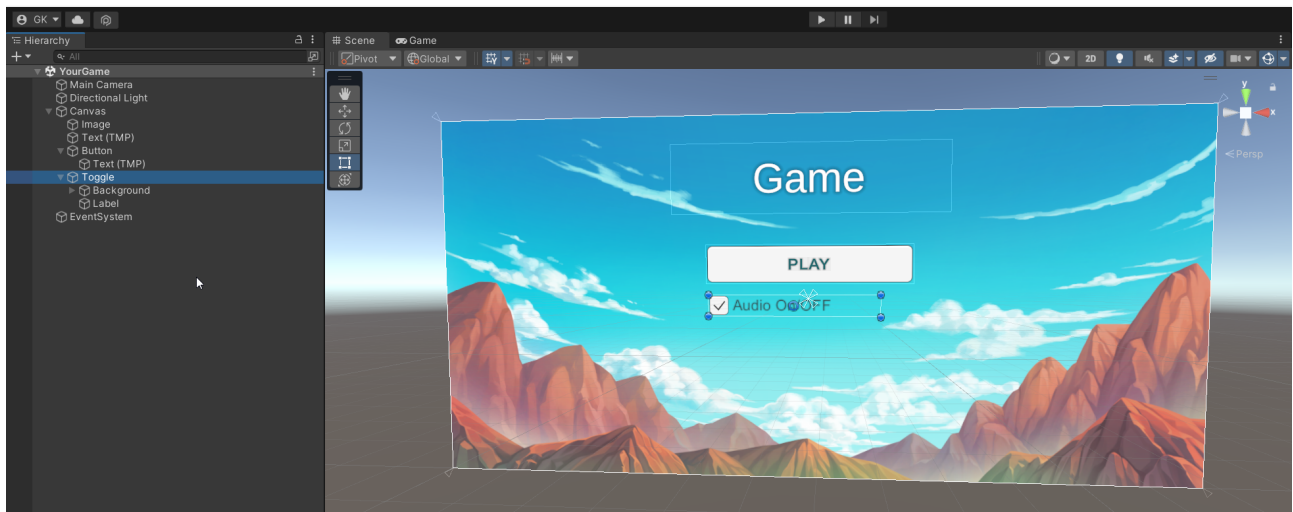
NOTICE: It is not recommended to directly use the examples (except for testing).

Why? → Because if you UPDATE the asset in the future then the example assets will be reset (overwritten). This happens even if you move them out of the examples folder.

That's because of the UGUI system Unity uses to track asset files. Each file has a unique GUID and upon import Unity will match these and replace the files based on their GUIDs, not matter where in the Assets folder they are.

Let's assume you already have some UI in your game and you want to replace parts of it with the settings system UI.

Your game:



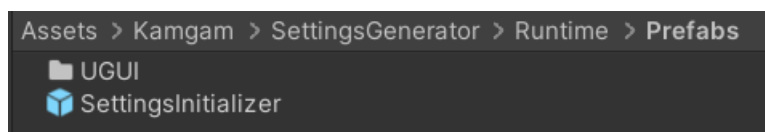
Notice the EventSystem in the scene. Without an EventSystem and a GraphicsRaycaster on your canvas UI mouse clicks will not be registered. I mention this here because the demo scenes are loading the UI additively and the EventSystem is actually in the main scene. This means if you copy the demo UI into your scene you will have to make sure your scene has an EventSystem in it or else mouse click would not work. - Adding an EventSystem is something experience Unity devs do automatically but I have found that for newcomers it sometimes is confusing.

Step 1: Make sure the settings are initialized (loaded) at the start.

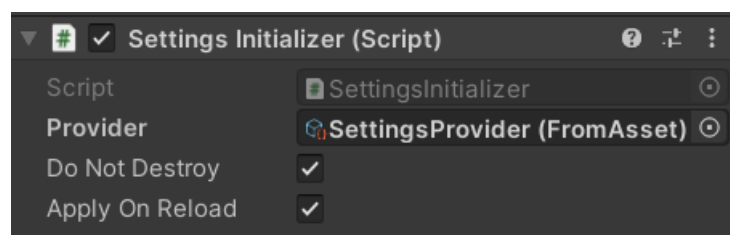
The settings system initializes itself whenever it is used (when a setting UI is displayed for the first time).

However, usually games do not start with the settings screen and thus you should initialize the settings manually to ensure they are loaded right at the start.

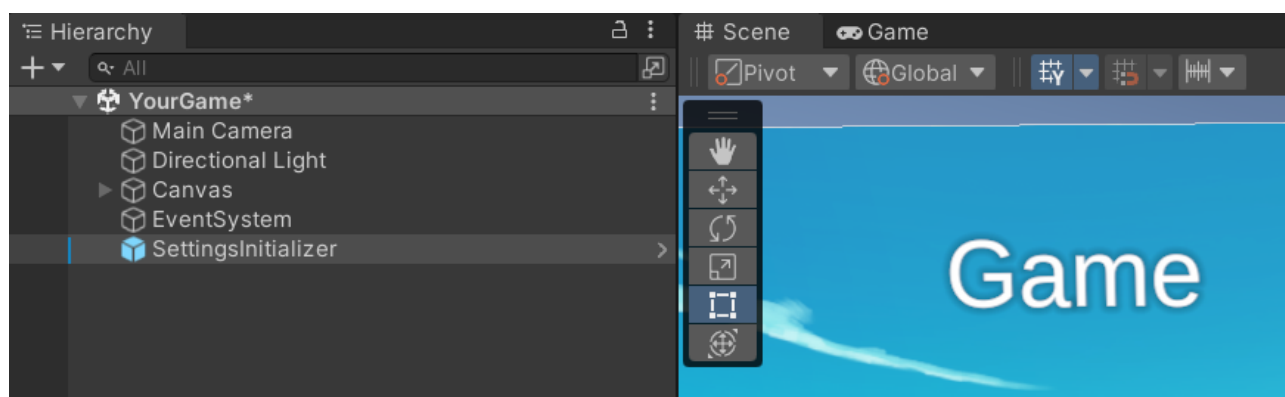
To do this please drag in the „SettingsInitializer“ prefab into your very first scene. You can find it under: **Assets/Kamgam/SettingsGenerator/Runtime/Prefabs**



Settings Initializer



Please do not forget to assign the correct PROVIDER once you have copied the provider asset (we'll get to that later).

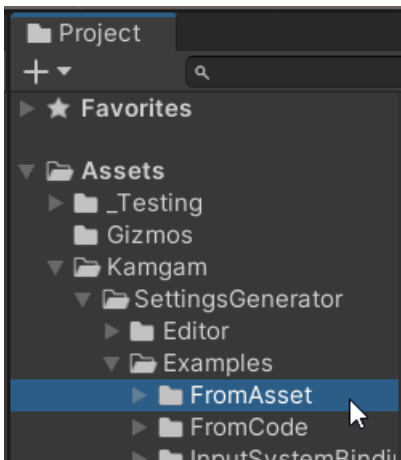


Do Not Destroy: Enable if you are unloading the scene that contains the initializer. This will avoid unloading the initializer. If you can then disable this and use additive scene loading instead.

Apply OnReload: If enabled then it will re-apply the settings in Start() after reloading this scene. If DoNotDestry is disabled then this option is ignored completely.

Step 2: Make a copy of the example and clean it up

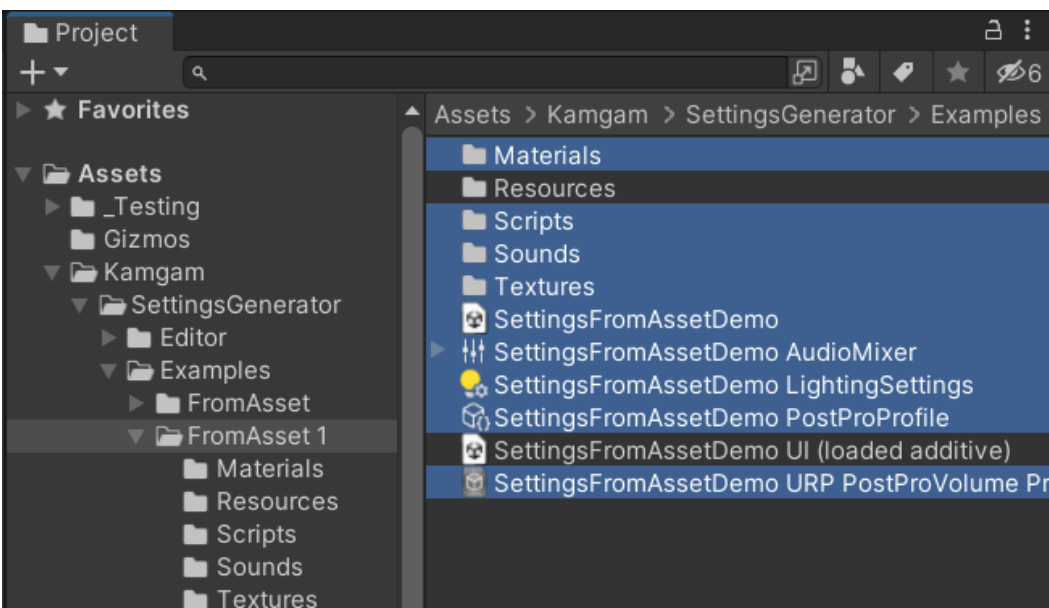
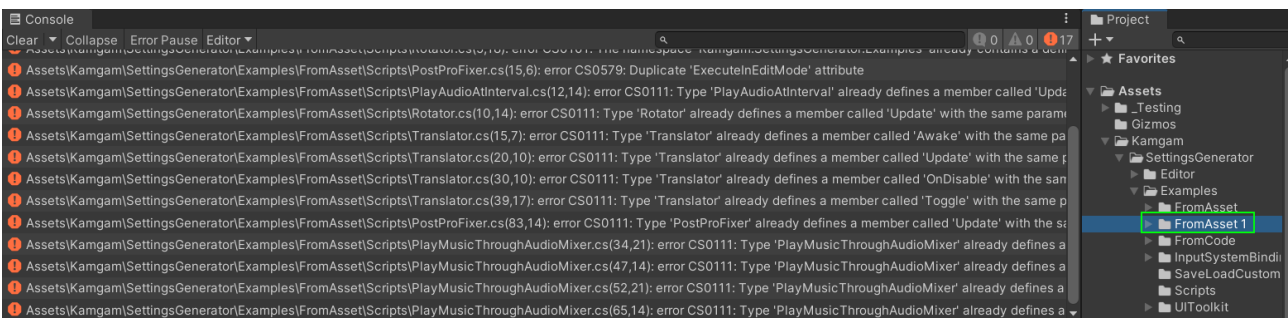
We do not want to use the examples directly (explanation above) so we make a copy of it. The easiest way is to select the folder of the example and press CTRL (or CMD) + D.



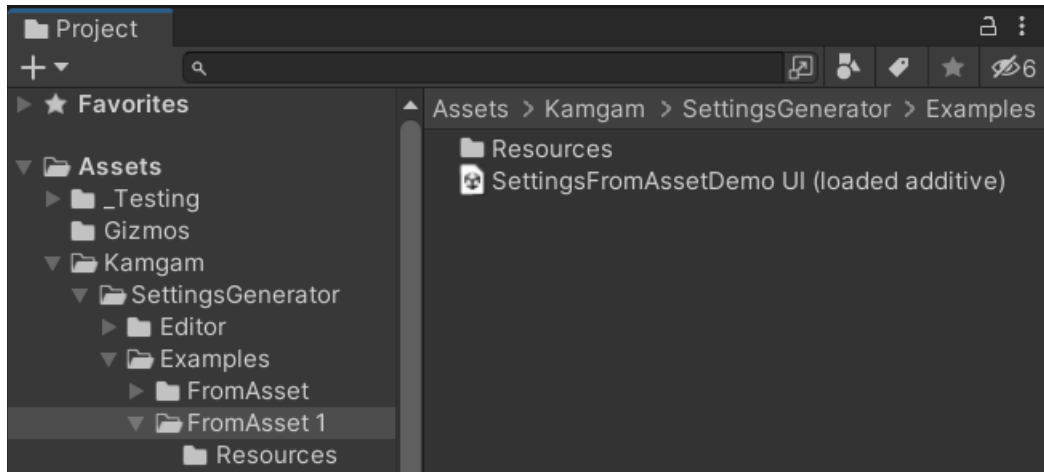
This will create another folder called „FromAsset 1“ and cause A LOT of compile errors.

The errors are shown because a lot of code is now duplicated. We'll fix those right away by deleting all the unneeded duplicate stuff.

Delete everything except the „Resources“ and the „.. UI (loaded additive)“ scene file.



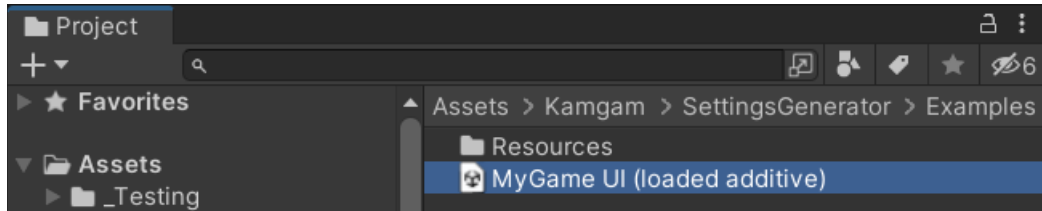
Once you have deleted them, all the errors should be gone and you are left with this:



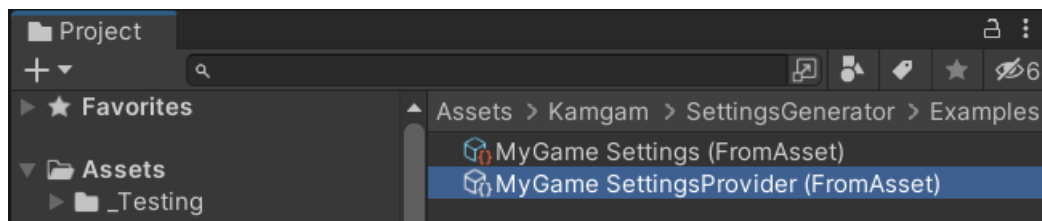
Step 3: Rename the resources and scene

Renaming the resources is important as they are a copy of the example resources and thus share the same name. This will make it hard to differentiate them from the example res. Therefore we should rename them.

Like this:

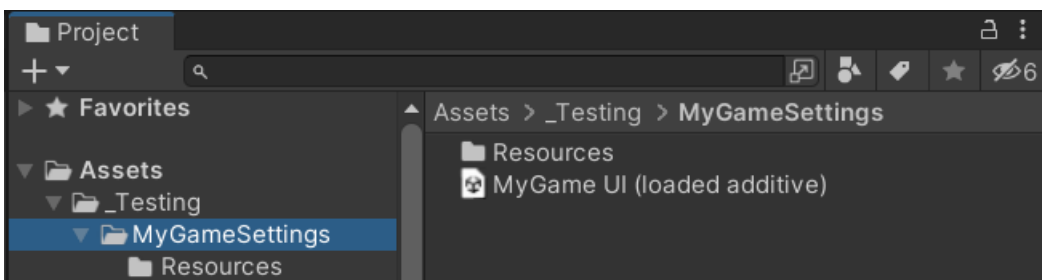


Resources:



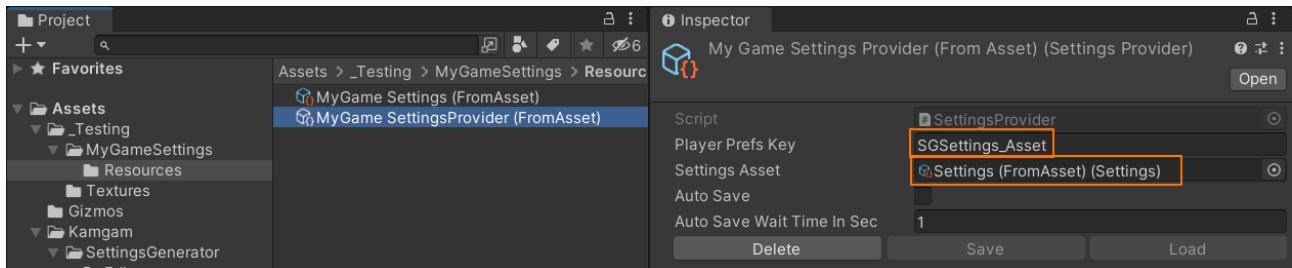
Let's move the folder out of the examples. It does not matter where you move it to. You can also move only the Resources folder and the scene file separately if you wish.

In the example below I kept the parent folder, moved it to „Assets/_Testing“ and called it „MyGameSettings“

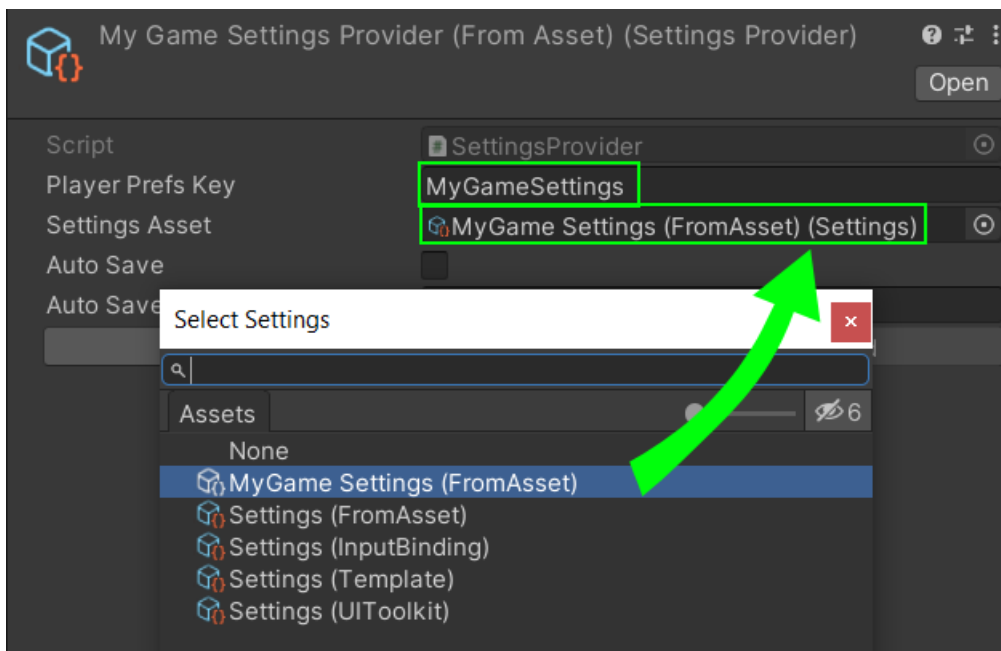


Step 4: Fixing the links inside the provider & choosing a save file key.

Sadly Unity is not very smart when copying objects. We have to tell it to use our new settings asset and we also need to pick a new key for saving to avoid mixing up saved settings with the examples.

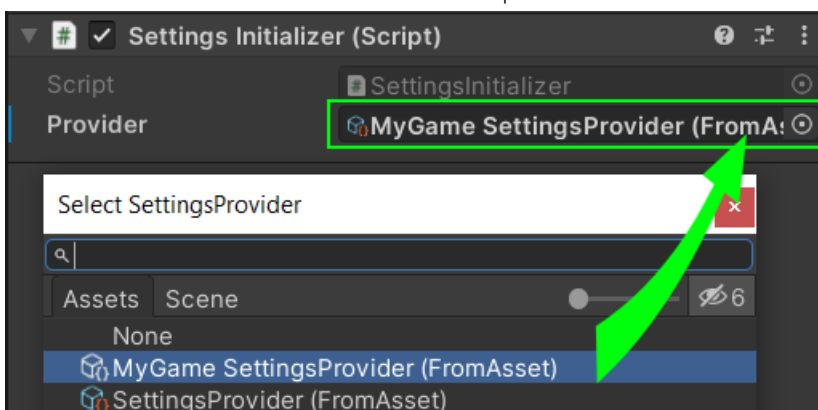


For the „PlayerPrefsKey“ you can choose whatever name you like. I chose „MyGameSettings“. For the „Settings Asset“ choose the copy (now you see why it is good that we renamed it).



Remember: the SettingsInitializer also needs a reference to the provider.

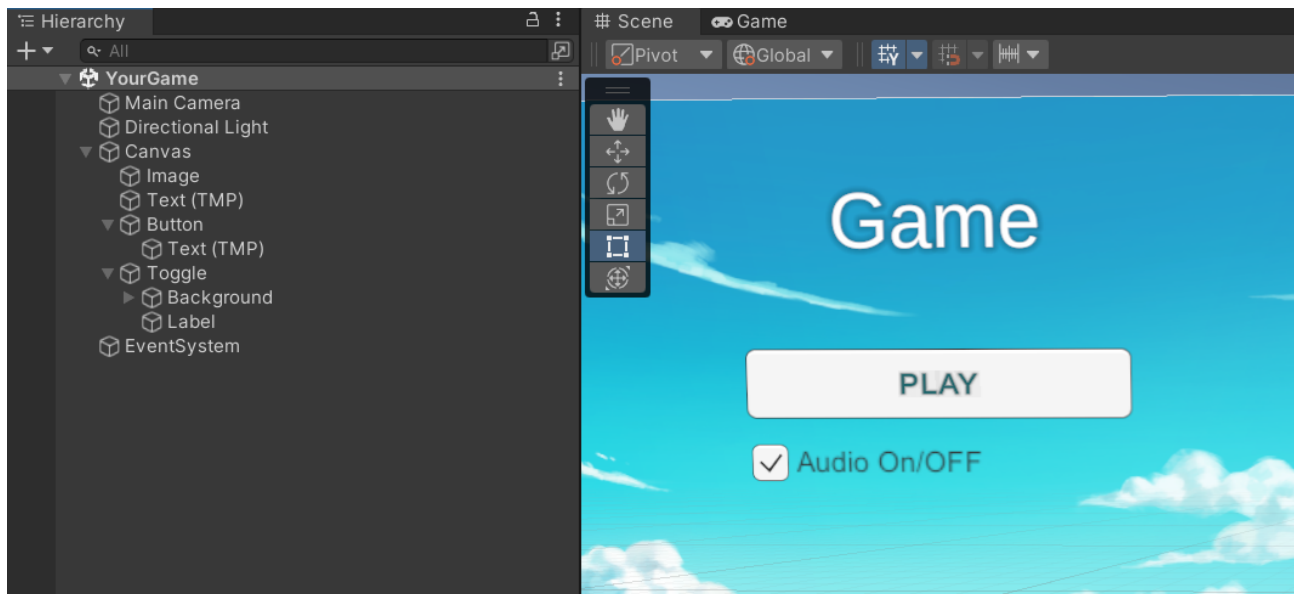
Let's make sure it references our new provider:



Okay, phew, now we have the setup part done. Next: integrate it into your own UI.

Step 5: Integrate the UI into your game.

Let's say your game UI looks like this:



And you want to replace the Audio On/Off trigger with the settings. You have to make a choice here.

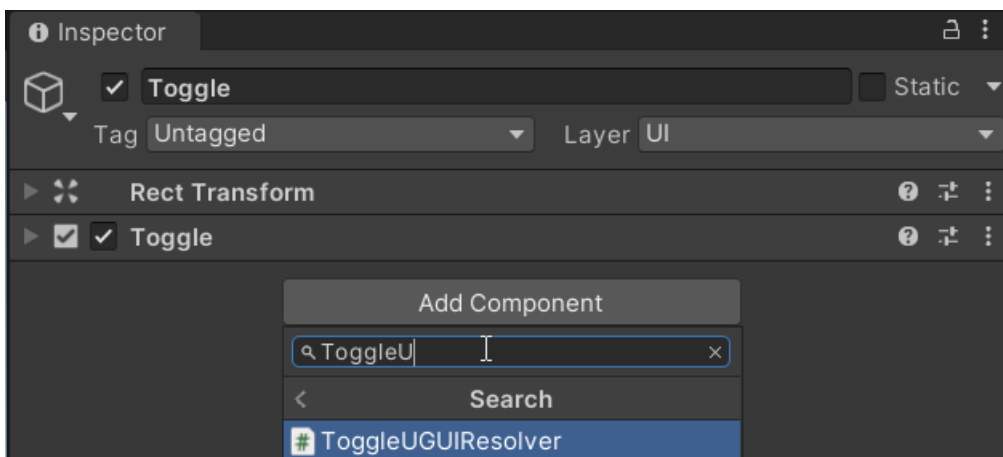
Do you want to:

- (A) Keep using your own UI and simply hook it up with the settings system.
- (B) Copy in the UI from the settings scene.

We'll cover both scenarios below.

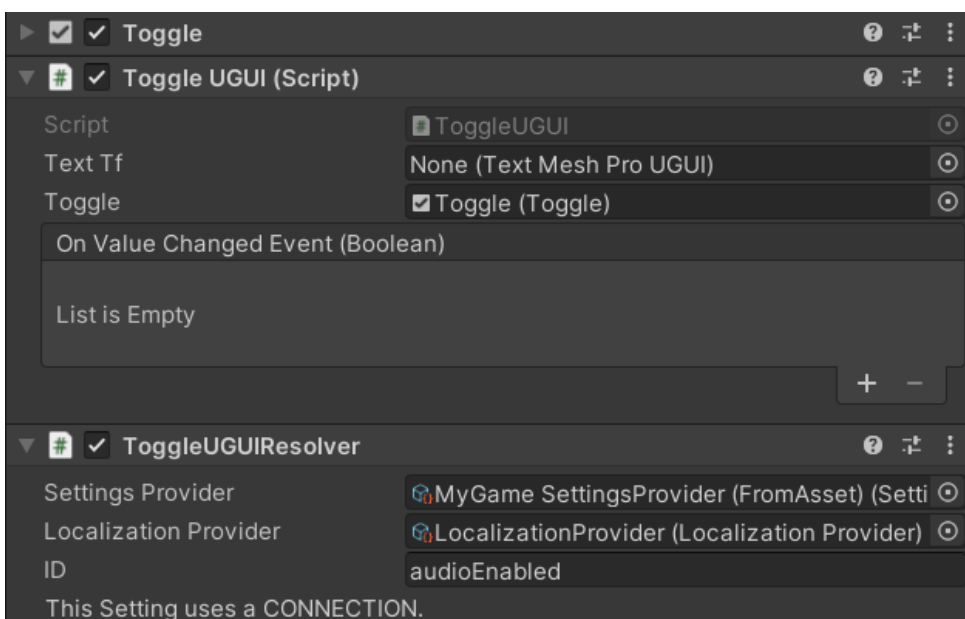
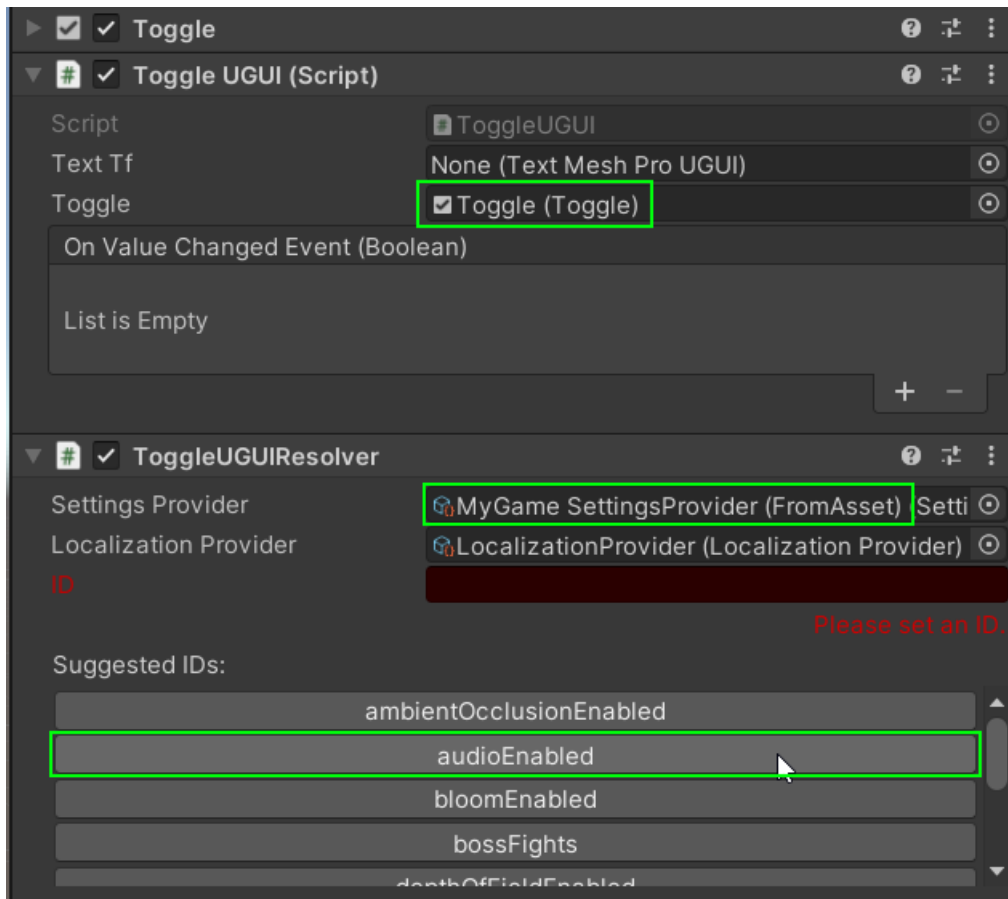
A: Hook your UI up with the settings system

For this we have to add a „ToggleUGUIResolver“ component.



You may have noticed that the resolver also added a „Toggle UGUI“ component.

1. Make sure the „Toggle“ property is hooked up with the Toggle (usually it does this automatically).
2. We also have to configure it to use the proper settings provider („MyGame SettingsProvider (FromAsset)“).
3. And finally we have to choose one of the predefined settings. In this case we use the „audioEnabled“ settings as this will toggle the audio.



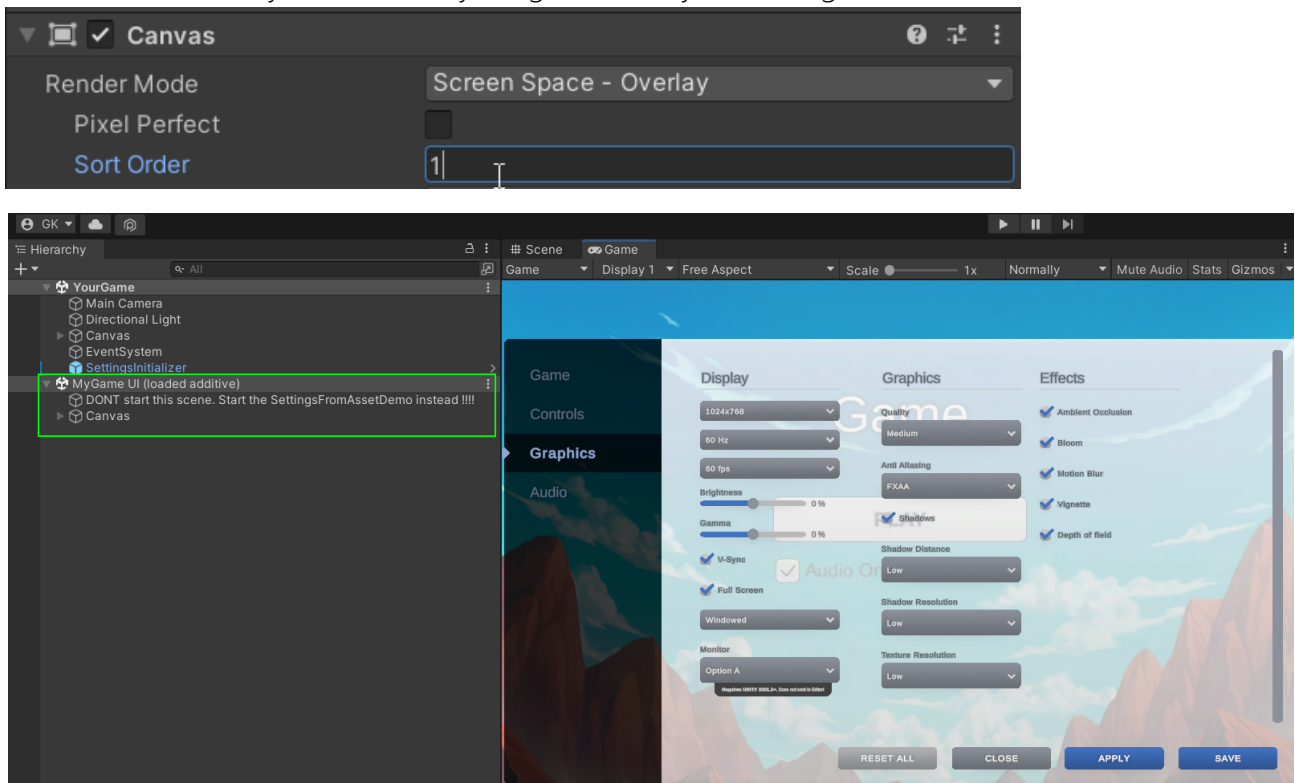
And that's it. It works the same for any other UI element.

B: Copy in the UI from the settings.

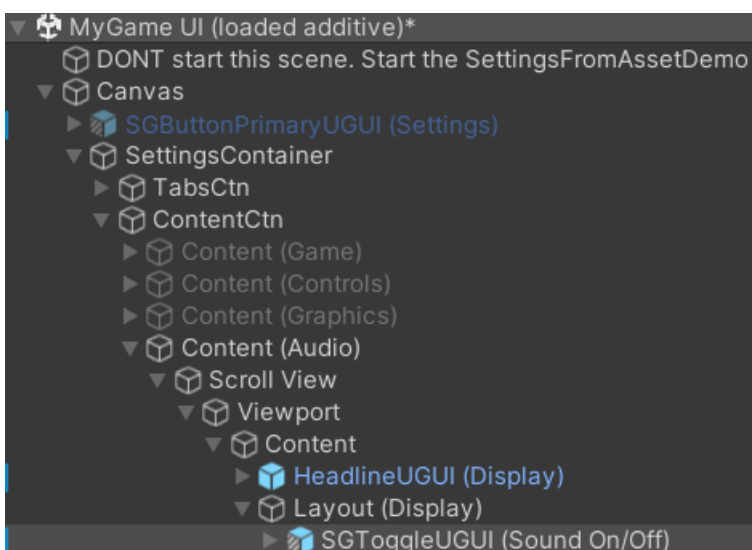
If you want to use the premade UI from the example then first drag in the scene you copied into your game scene. In the end we will not use that scene file but it is convenient to keep it around so we can copy from it.

Once dragged in it may look a bit messy. Like this:

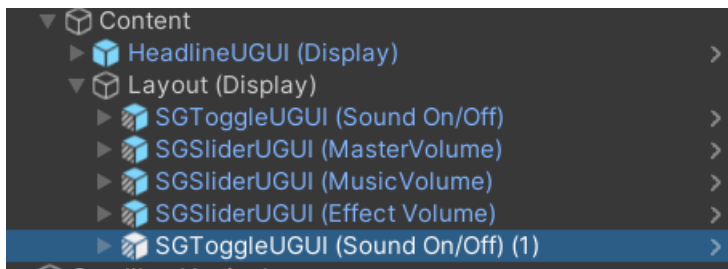
Hint: The canvas may hide behind your game UI. Try increasing the SortOrder to make it visible.



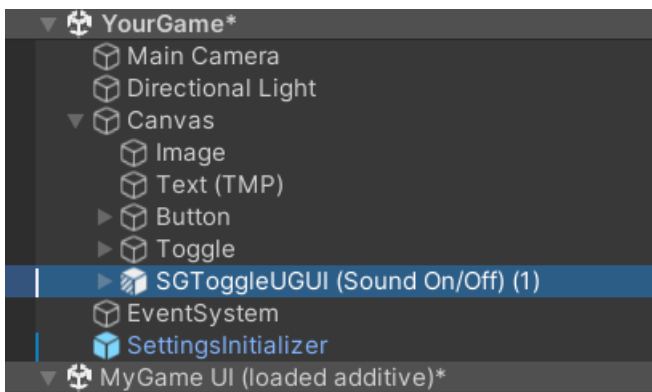
Don't worry about about any changes to the demo UI. We will not save it and remove it from your game once we are done. Now all we need to do is find the „Sound toggle“:



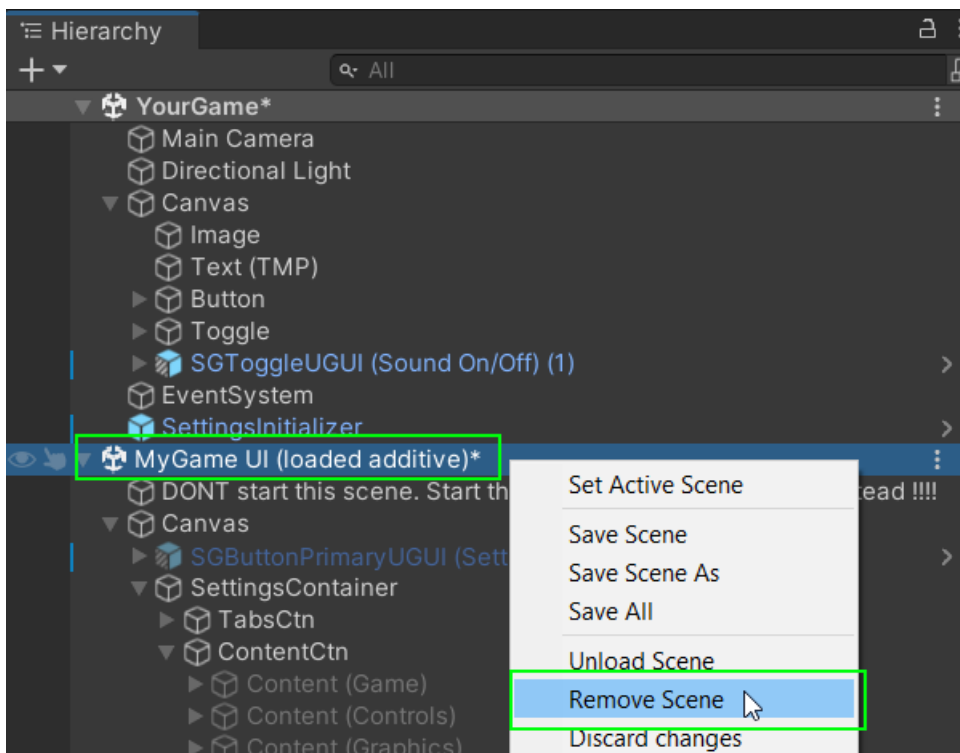
Once we have it let's duplicate it (CTRL/CMD + D):



And move it to our game UI:

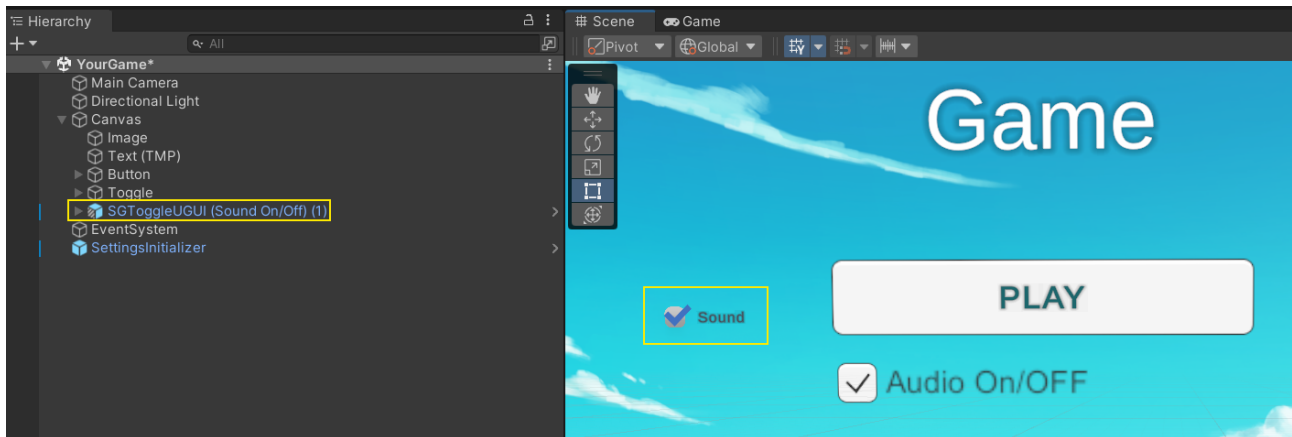


Now that we have what we wanted we can remove the scene. We do not need it anymore. Don't save it.



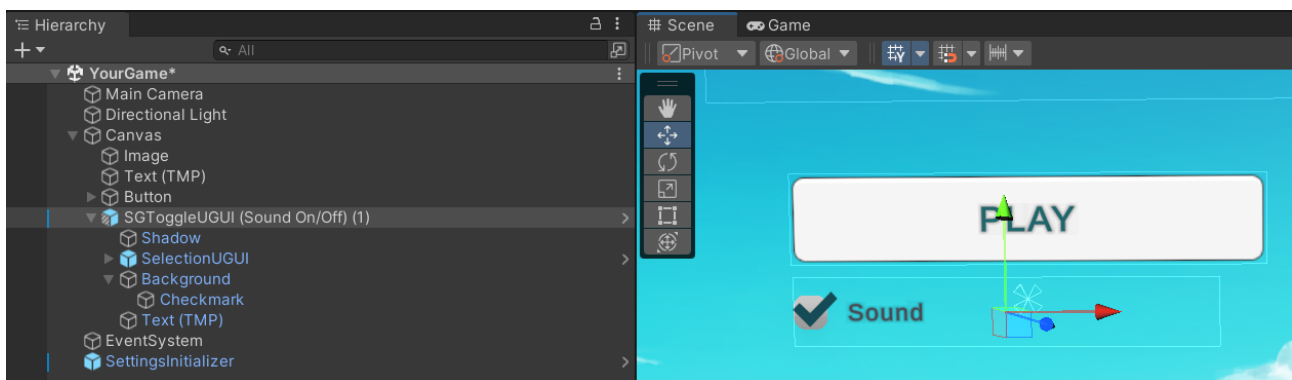
We have successfully copied the toggle to your game UI.

It should look something like this (new parts marked yellow):

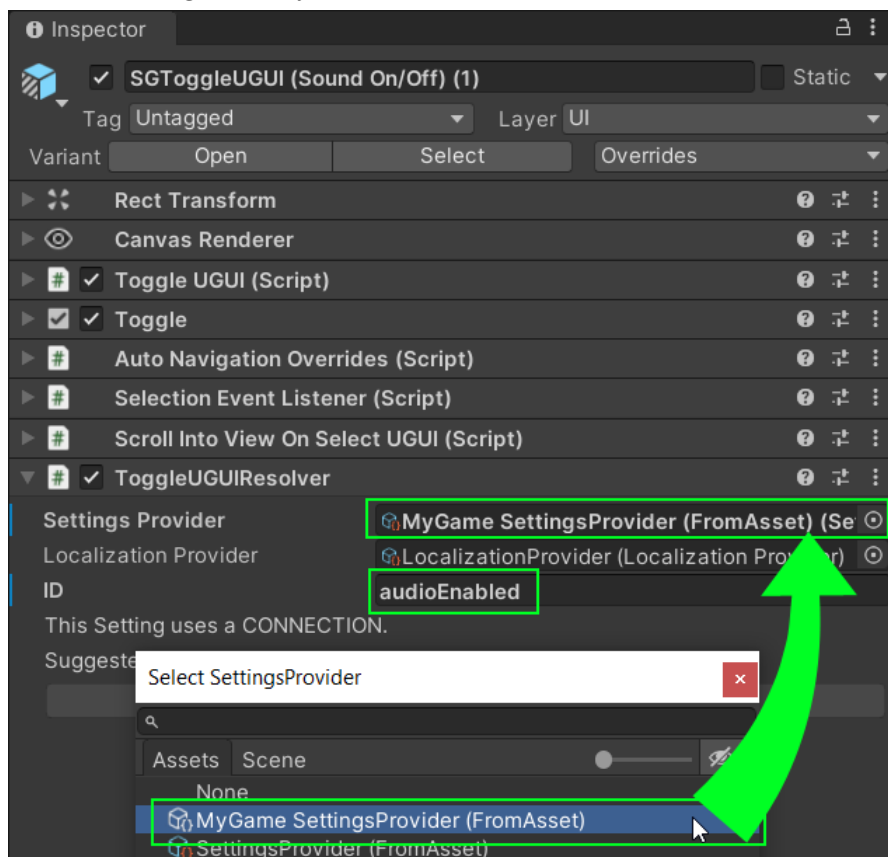


We can now delete our old toggle and position the new one.

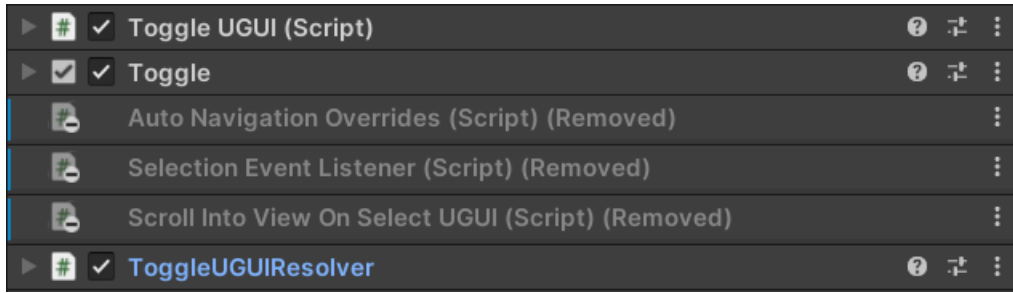
Finally we need to make sure the toggle uses our new settings provider and is hooked up to the



correct setting, namely „audioEnabled“.



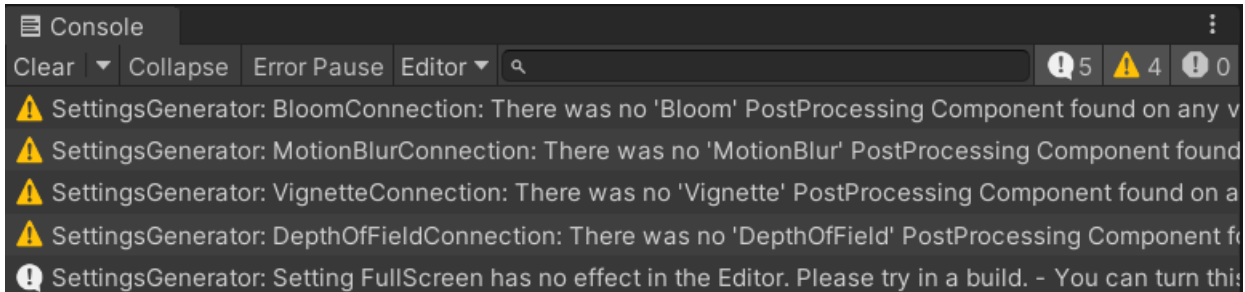
If you want you can remove all the unnecessary helper components too.



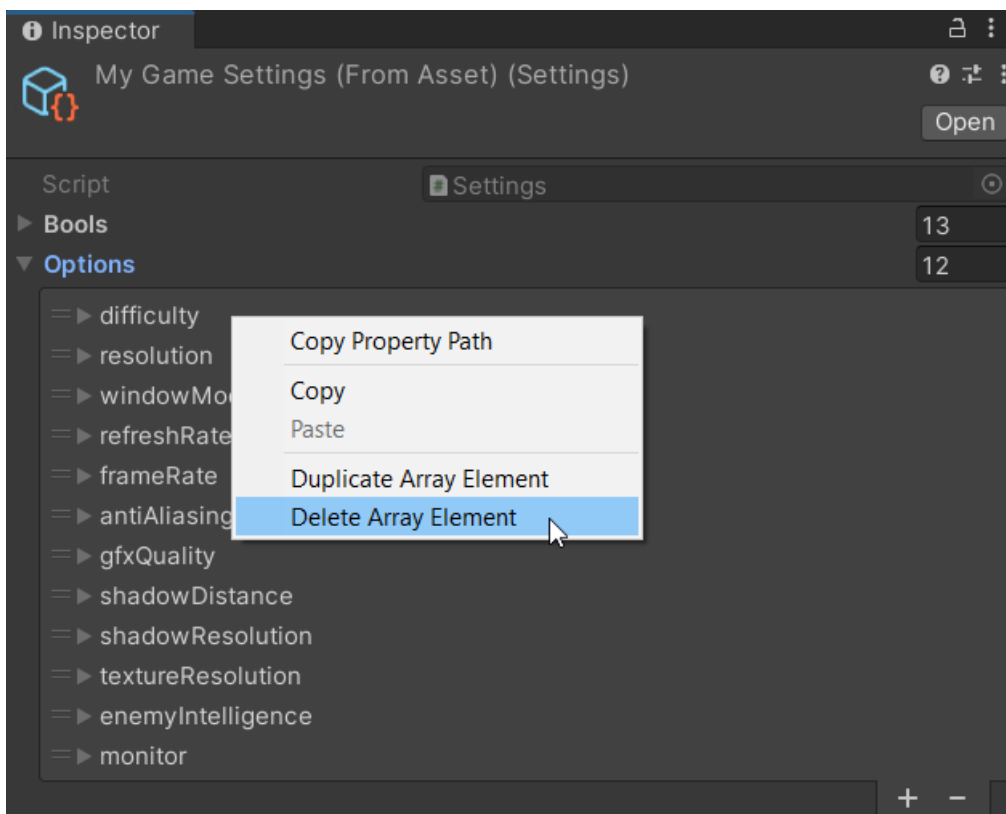
And that's it.

Step 6: (Optional) Delete unneeded settings.

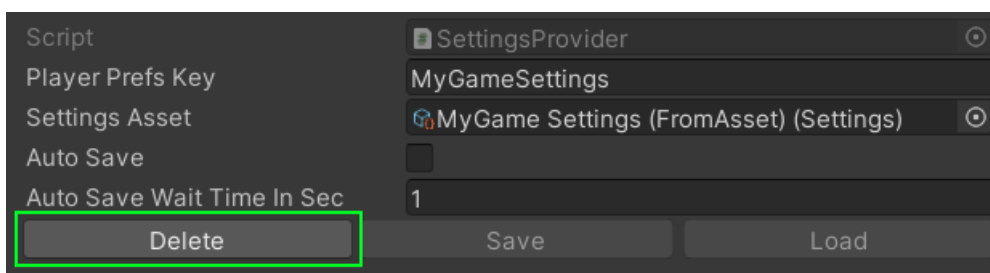
You may have noticed some logs in play mode when using the settings. These are there because the example asset includes ALL settings and some are only working if set up correctly (post-pro for example).



In case you are not using them you can clean up your settings asset to only use those that you are using. For this open the settings asset of your game and remove whatever you don't need.



After deletion please also delete any stored settings, or else the system will try to restore them.



Thanks for reading it all. I wish you all the best for your game!

Combining the Settings System with other Assets

You may have noticed that the settings are only applied at certain points in time (i.e. when the menu is used or whenever you call „Settings.Apply(changedOnly: false);“.

If you are using other assets in your game it may happen that these will override your settings.

Most common are audio assets setting the master volume on your AudioListener. This may lead to the situation that you have muted the master audio in the settings menu, yet in your level the audio still plays as loudly as ever.

You may have a setup like this:



First Frame

SettingsSystem

Sets the Volume at level start to 0

Every Frame in the Level

AudioManager(s) - could be multiple if you use multiple Assets
Sets the Volume to something else.



Repeats every frame.

As you can see in the image above the third-party assets are continually setting the master volume, effectively negating any effect of your settings. Since the settings system can not know what code or assets you use it can't prevent that.

The proper way to fix this is to forward the settings value to whatever third-party system you are using.

The control flow should be like this:

SETTINGS SYSTEM → THIRD-PARTY-ASSET → YOUR-LEVEL

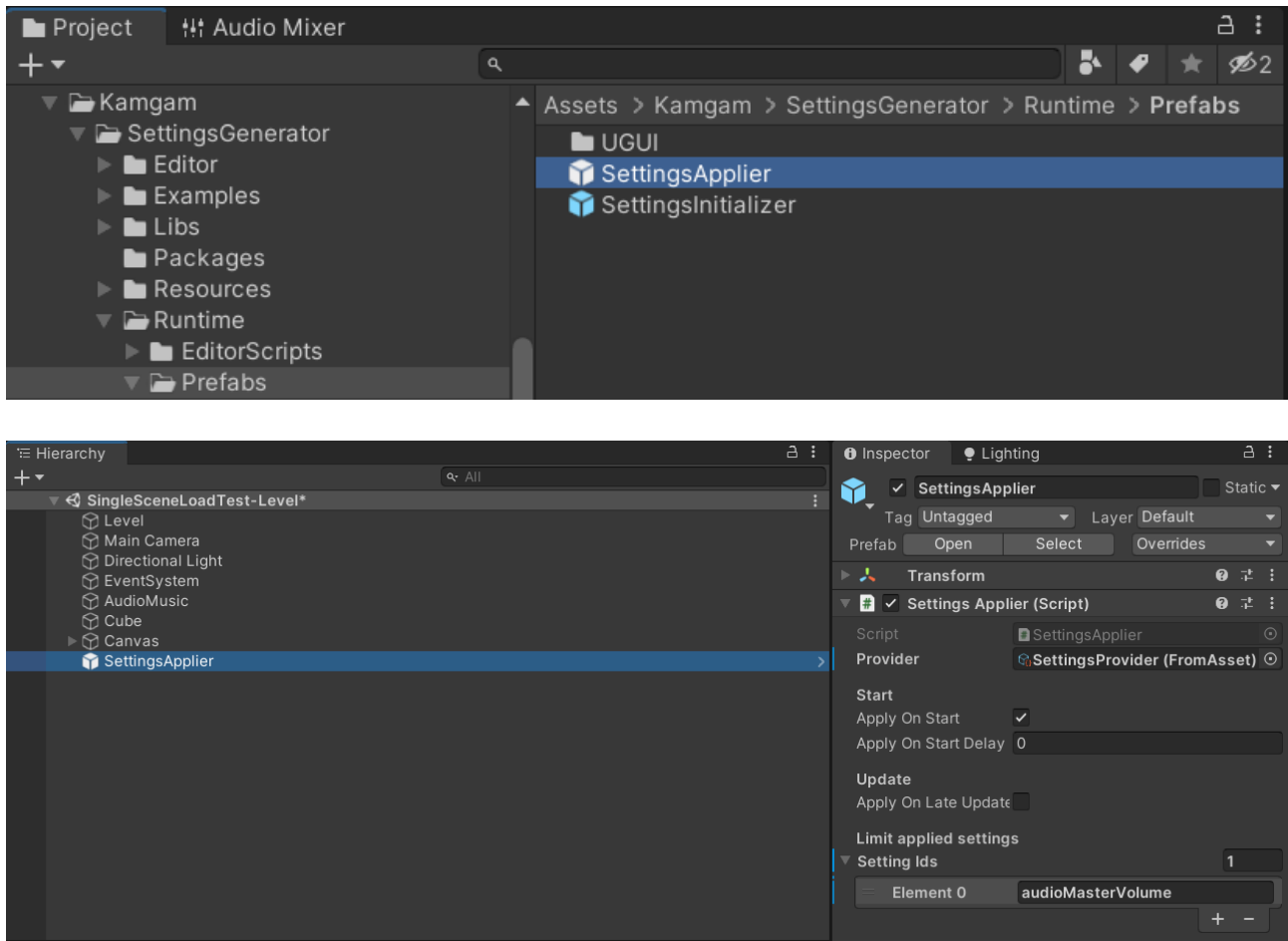
If you have both the settings system and some third party asset control a value at the same time then these two systems will fight over who takes priority. At worst the results are random applications of the setting values. Sometimes the settings system will win, sometimes the third-party asset. It's impossible to know.

The not-so-proper (really bad) way is to use the „SettingsApplier“. Really, you should fix this properly.

SettingsApplier

If you are not loading your scene additively (meaning you unload your menu scene) then you probably have to re-apply your settings on level load. There is a prefab called „SettingsApplier“ to make this step easier.

You can find it under **Kamgam > SettingsGenerator > Runtime > Prefabs**:



Provider: Don't forget to set the correct provider.

ApplyOnStart: By default the applier is configured to apply all settings on Start().

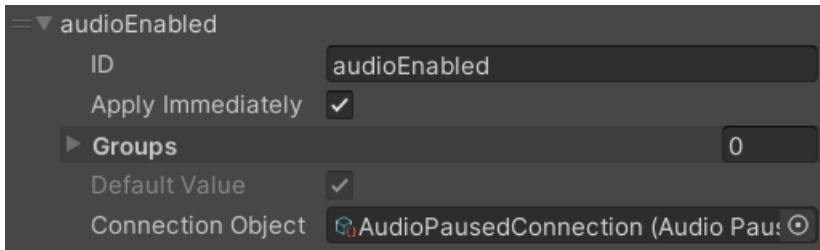
ApplyOnStartDelay: If you know that some other scripts are also being executed in Start() then you may want to delay the execution a little bit. Be aware though: timing based fixed are always bad.

ApplyOnLateUpdate: I strongly suggest you only use this for debugging to find out if another system is also setting the same values as the settings system. Please refer to the „Combining the Settings System with other Assets“ section for more details on why using Update() is a bad idea.

Settings Ids: Instead of applying ALL settings you can also apply only a subset. In the example above we only apply the „audioMaterVolume“ and nothing else.

Settings

Most Settings are based on simple types. There are some predefined ones but you can also invent your own.



ID:

Each settings has a unique ID which identifies it globally.

Apply Immediately:

Each setting has an „Apply Immediately“ switch. If enabled then any changes to the setting will be propagated to the connection immediately (if there is a connection). Usually this should be set to enabled. Though for some settings it is better to disable it („resolution“ dropdown for example). If disabled then the user will have to hit the „apply“ button to actually apply the changes.

Groups:

Each settings can be part of one or more groups. Groups can be used to perform actions only on a subset of settings (resetting only the key-bindings for example).

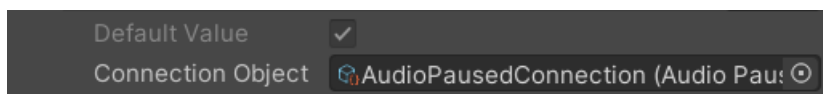
Default Value:

Each setting can have a default value (that's the value used for reset and at first boot). If the settings has a connection (meaning it gets its values from a dynamic code source) then the default value will be ignored (it is pulled from the connection instead).

Connection Object:

Each settings can have one connection. A connection object is just a wrapper for some code that should be executed if the setting value changes or if the setting is applied (see „Apply Immediately“ above).

All these things can also be configured via the scripting API. You can find a fully scripted example in the „Examples/FromCode“ directory.



Basic Types (Bool, Int, Float, String)

These are generic settings to create custom settings from. You can use a `GetSetConnection<T>` to hook them up to any method in your game.

Colors

Save and load any color value.

ColorOptions

Let the user pick from a list of colors.

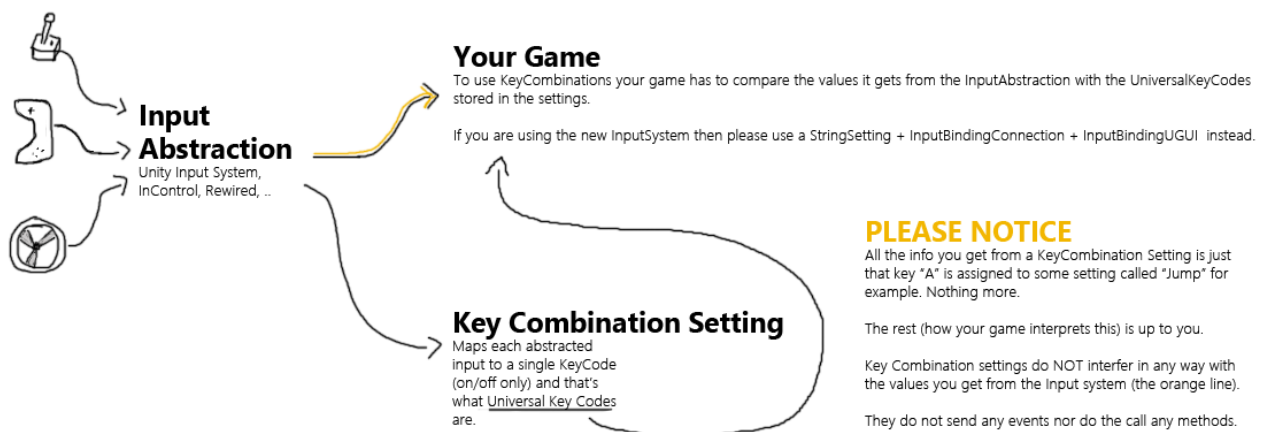
Options

If you want the player to choose only from a limited set. You give it a list of names (A, B, C) and it tells you with an index (0,1,2) which option the player selected.

Key Combination (preferred solution for the old InputSystem)

Used for key code storage for example. Supports old and new key codes and key combinations.

NOTICE: A **KeyCombination Settings** does **NOT** do any input binding. It simply receives and stores the pressed keys as a `UniversalKeyCode` (enum). It does not interact with your `InputActions` at all. It does not send any events or messages.



HINT: There is a static class „`Kamgam.UGUIComponentsForSettings.InputUtils`“ that contains some useful methods for dealing with input detection.

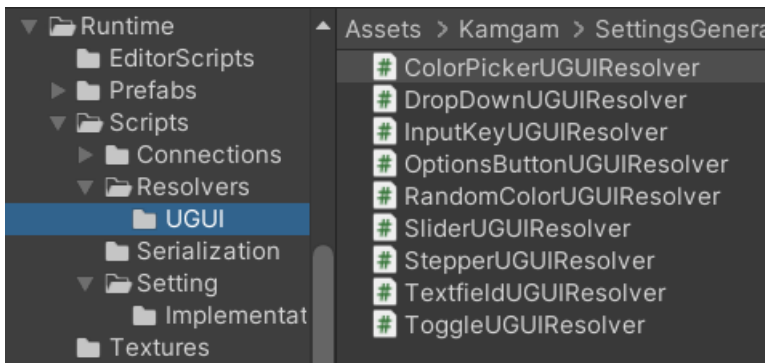
Input Binding (preferred solution for the new Input System)

If you need rebinding of an action in the new InputSystem then please use a **String Setting** in combination with an **InputBindingConnection** and the **InputBindingUGUI (Setting)** Prefab. The [Input Rebinding](#) section contains a detailed explanation on how to do that. You can find a demo of this under `Asset/Kamgam/SettingsGenerator/InputSystemBinding`.

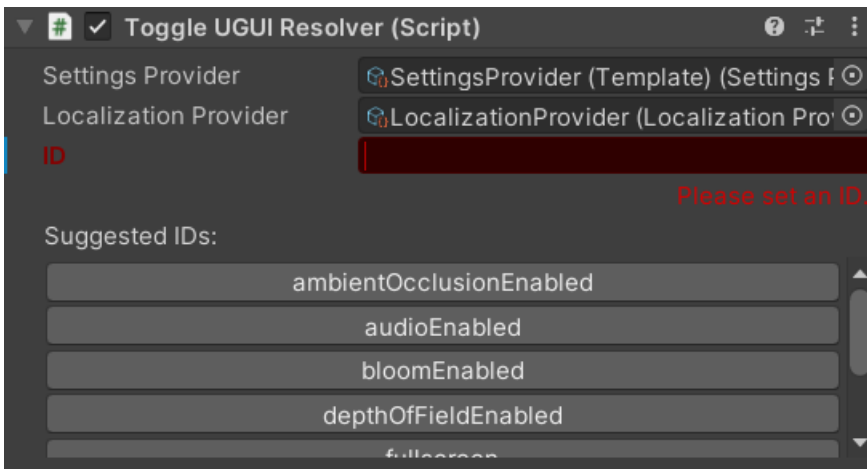
Setting Resolvers

If you want to hook up a UI with the settings system you need something to make that connection. Any object can be a resolver as long as it implements the **ISettingResolver** interface. It does not have to be a UI.

There are some predefined UI Resolvers (see folder „Runtime/Scripts/Resolvers“). These are implementations for the UGUI components (see above).



A typical resolver looks like this:



The „Settings Provider“ is a reference to the settings provider asset (see section „Overview“).

The „Localization Provider“ is a reference to the localization provider (see section „Localization“).

The „ID“ is where you will have to enter the ID of the setting you want to connect to.

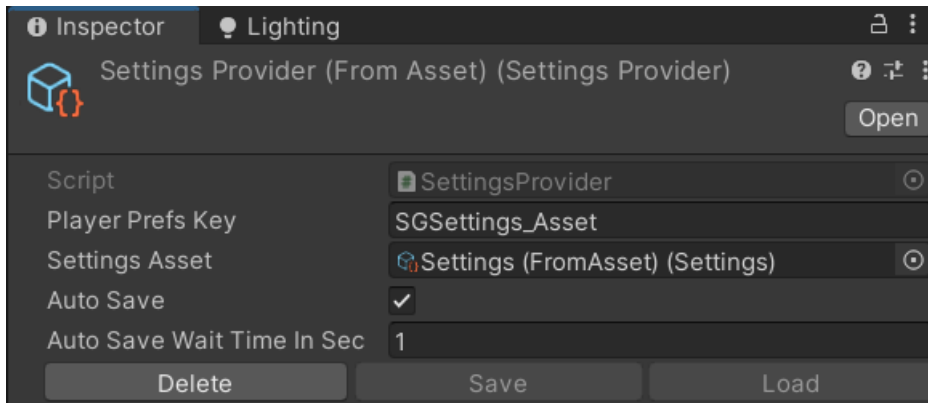
NOTICE: Not every setting is compatible with every resolver. The „Suggested IDs“ will only list compatible IDs. If you enter an incompatible ID then you will receive a „type mismatch“ error like this:

SettingsGenerator: SGSettingResolver: Type mismatch for ID 'resolution'. Got 'Kamgam.SettingsGenerator.SettingOption' but can only handle types: Bool.

Some resolvers can take multiple setting types. A slider for example can take float or int settings.

Auto-Save

Auto save is disabled by default. To enable it you have to check the „Auto Save“ checkbox on the SettingsProvider. Please make sure you change this on the right one (the one used by your resolvers on the UI).



Auto save will NOT save immediately after a settings has changed. It will wait for „Auto Save Wait Time in Sec“ and then save. This is done to avoid saving too often, for example if a slider is dragged. If a setting is changed within the wait time then the wait will restart.

UI Systems Overview (ImGui vs UGUI vs UI Toolkit)

As you may know there are three UI Systems. Here they are listed from OLD to NEW.

ImGui:

Immediate GUI (used primarily in the Editor but you can also make game ui with it). It's the oldest system and you really should not use it anymore except in rare occasions.

UGUI:

The current standard. This is the most widely used and supported gui system. If an asset in the store says „ui“ on it then in 99% of cases it's means UGUI. I recommend you keep using UGUI until UI Toolkit is ready for prime time (which in my opinion will be in about 1-2 years).

UI Toolkit (aka UIElements):

This is the newcomer system. It works quite differently than ugui since it uses XML (UXML) and a CSS subset (USS) to define your UI. The most notable change is that you no longer have a hierarchy in the scene and thus you can not add any components (MonoBehaviours) to the individual elements.

The settings system has rudimentary support for UI Elements. The code aspects all work (all connections are supported) but there are much fewer ready-made UI components. Please read the „UI Toolkit“ section for more details.

NOTICE: There was a significant change in the UI Toolkit in **Unity 2021.2**. At that point the UI Toolkit switched from being a package (com.unity.ui) to an internal module. This means that you do not have to install the UI Toolkit in Unity 2021.2+, it is already there included.

This also means that the old package, and with it Unity versions older than 2021.2, have dropped out of UI Toolkit support.

The old UI Toolkit package does not work very well and is no longer Supported by Unity. **It is therefore strongly suggested to only use UI Toolkit in Unity 2021.2+.** The settings system may drop UIToolkit support for Unity below 2021.2 in the future (as Unity did).

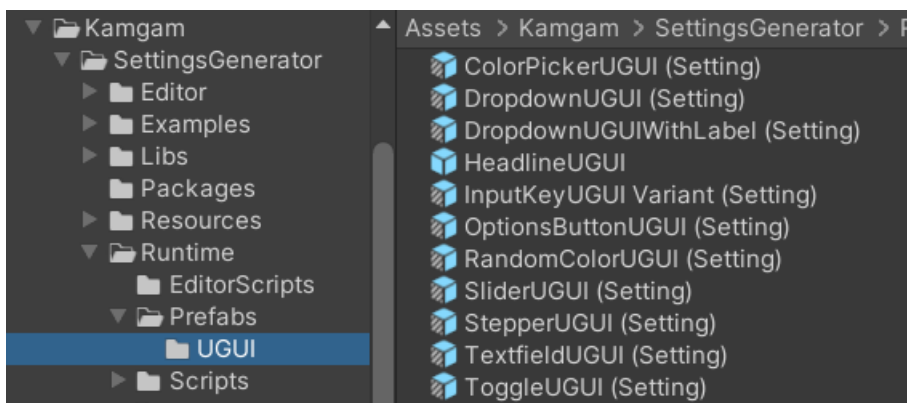
IMGUI (not supported)

IMGUI is the old legacy UI system. There are no read-made UI components or resolvers for it. However the code parts can be hooked up if you are a programmer. To do soe you will have to implement the resolver interface for your IMGUI elements.

UGUI (fully supported)

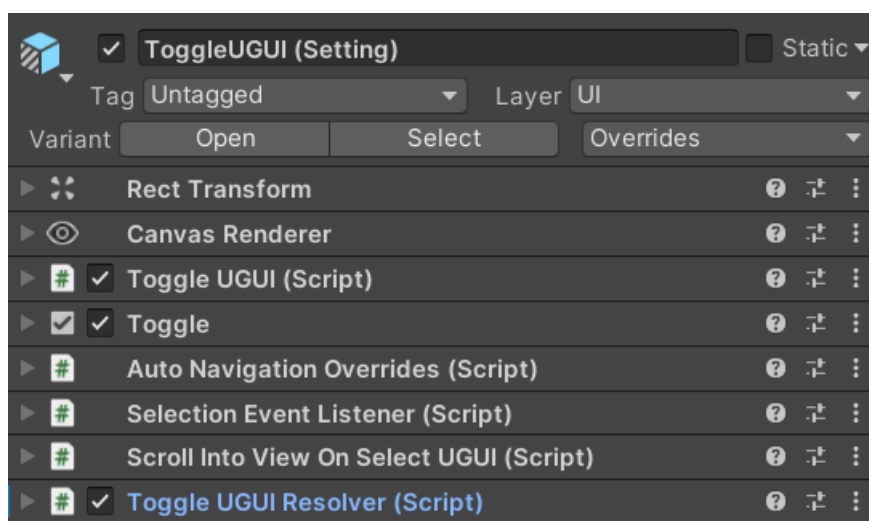
UGUI Components

The settings generator contains a set of UGUI prefabs that you can use to create your UI. They are the visual representation of the settings. These components are located in the „Runtime/Prefabs/UGUI“ folder.



You can of course use your own UI components. Though you probably would have to write UI Resolvers for them (more on UI Resolvers below).

Each UI prefab has some components attached. The most important one is the „UI Resolver“ (details below). The others are mainly convenience components (see „Helpers“ section).



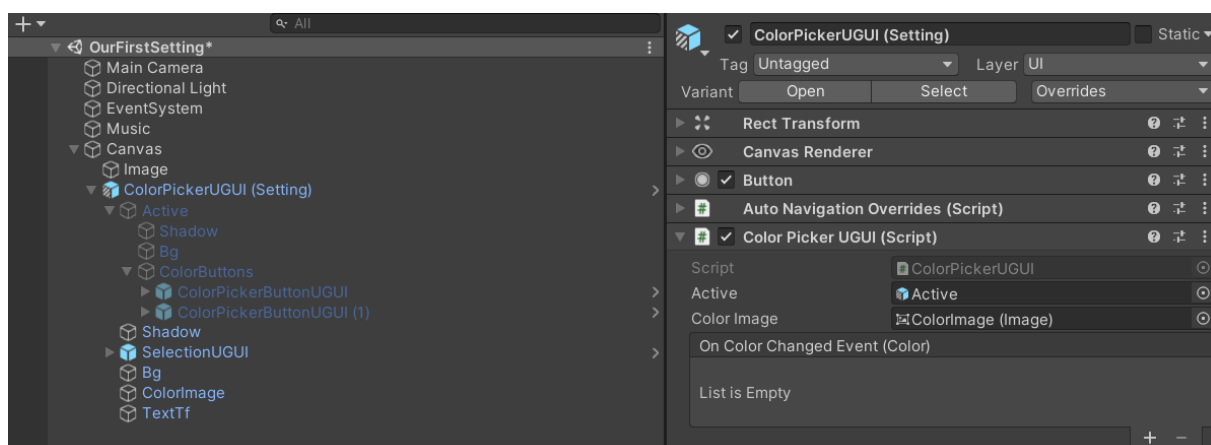
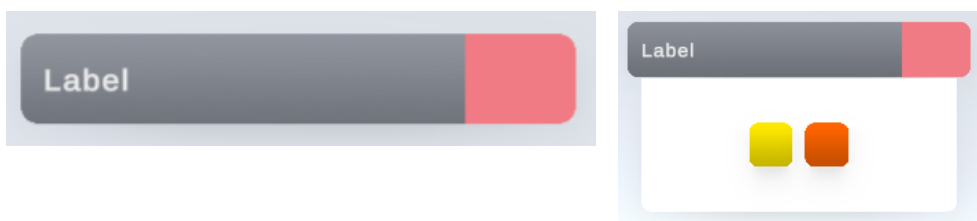
Implementations

When searching for component prefabs make sure you are not accidentally using the components from the „Libs/UGUIComponentsForSettings“ library. These are the base prefabs.

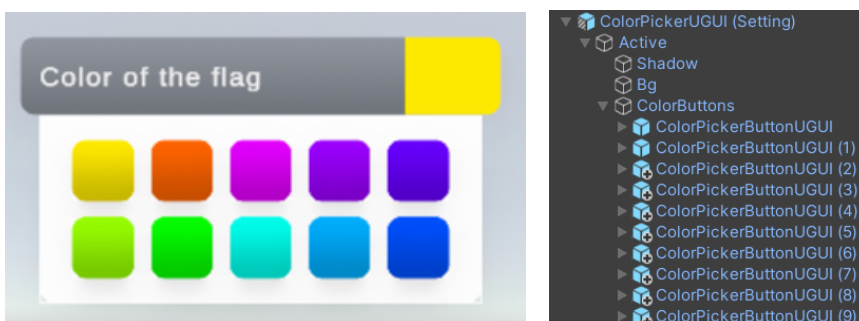
You should use the prefabs from the „Runtime/Prefabs/UGUI“ folder. The major difference between the two is that the latter are already prepared for use as settings UI while the former are just UI components with no special purpose.

Color Picker UGUI & Color UGUI

The Color picker allows the user to pick from a list of colors. It is supposed to be customized by you to represent the colors you want. By default it only contains two colors (ColorPickerButtonUGUI).



If you need more colors then duplicate the buttons and specify the color within each button.



The ColorPickerUGUI Resolver does not care how the UI looks as long as it can find some ColorPickerButtonUGUI components within the ColorButtons game object. You do not have to specify the colors in the UI, you can also specify them in the Settings asset or via a dynamic connection.

Dropdown UGUI

Shows a list of options from which the user can pick one.

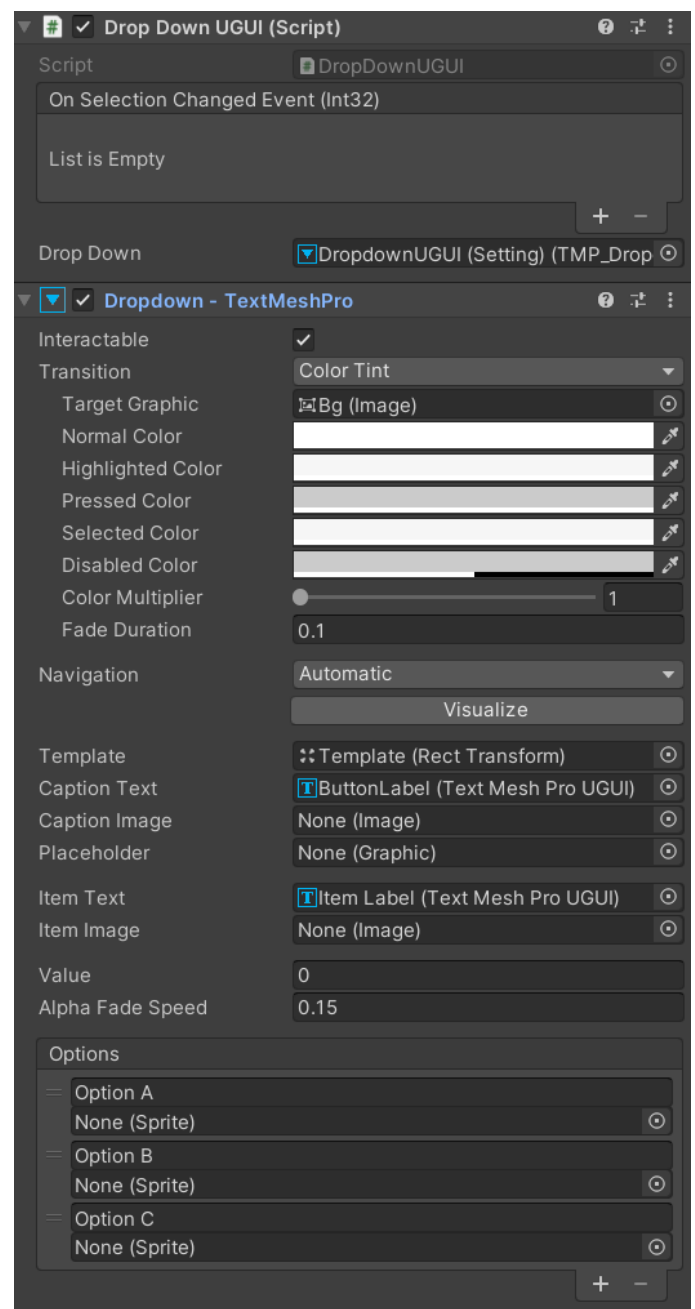


The Dropdown options can be specified in three places:

- A) Within the „Options“ of the TextMeshPro Dropdown.
- B) In the Settings asset
- C) Dynamically through a connection.

If C is present then it will override A and B.

If B is present then it will override A.



There are some variations of prefabs available. For example there is a „DropdownUGUIWithLabel (Setting)“ prefab which adds a label on top of the dropdown.



Headline UGUI

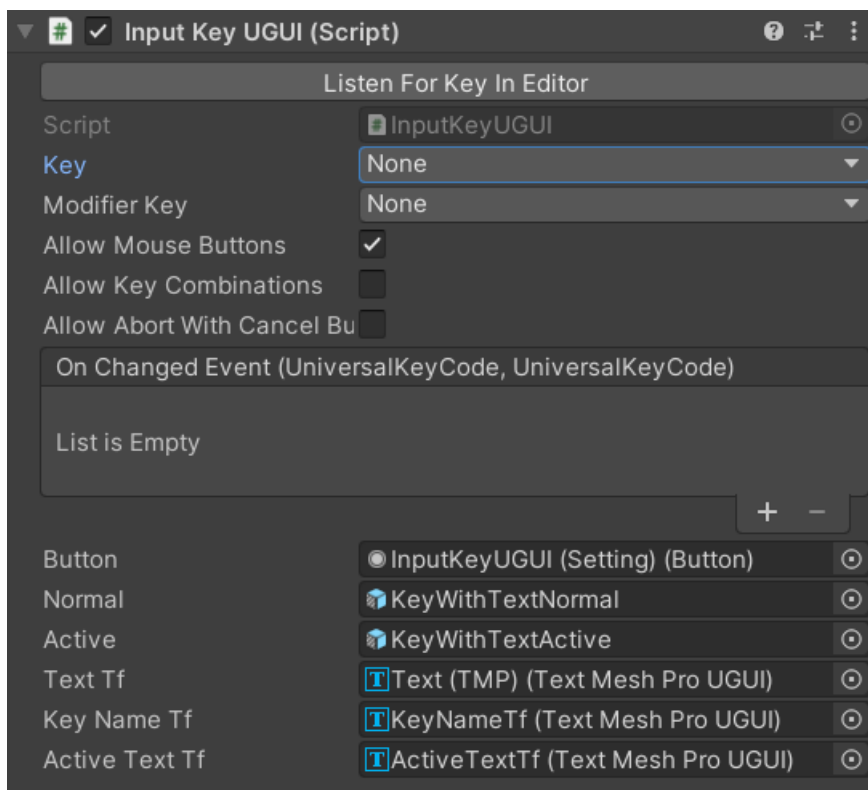
Just a label with a line beneath it. No settings related stuff here.

A light blue rectangular box with the word "Controls" in a dark blue, sans-serif font. A thin horizontal line is positioned just below the text.

Input Key UGUI

Key-Binding UI. It has an active and an inactive state. It listens for key, mouse and button inputs.

NOTICE: The InputKey does NOT perform any input binding. Use the InputBindingUGUI instead.



Input Binding UGUI

Key-Binding UI. The look and interaction is identical to the KeyCombination UI.



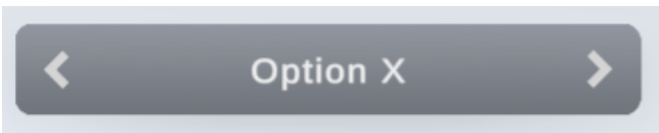
The difference is in the used setting (a simple String Setting) and the Connection (InputBindingConnection). It performs an interactive rebind using the new InputSystem. However to do this it requires a bit of setup. Please read the „[Input Binding](#)“ section before use.

Options Button UGUI

Similar to Options but it simply iterates through the options and shows the current one in the center. It is best suited for mobile devices or very short lists where a dropdown would be to cumbersome.

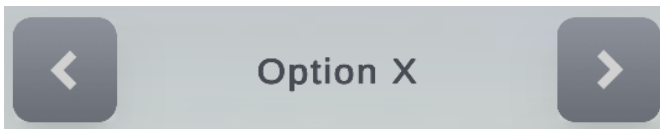
There are two variations of that Prefab:

OptionsButtonUGUI (Setting)



Despite showing two arrows this is just one button which cycles through the options.

OptionsArrowButtonsUGUI (Setting)



This one has two buttons (previous and next).

Slider UGUI

The slider is built based on the default Unity Slider.

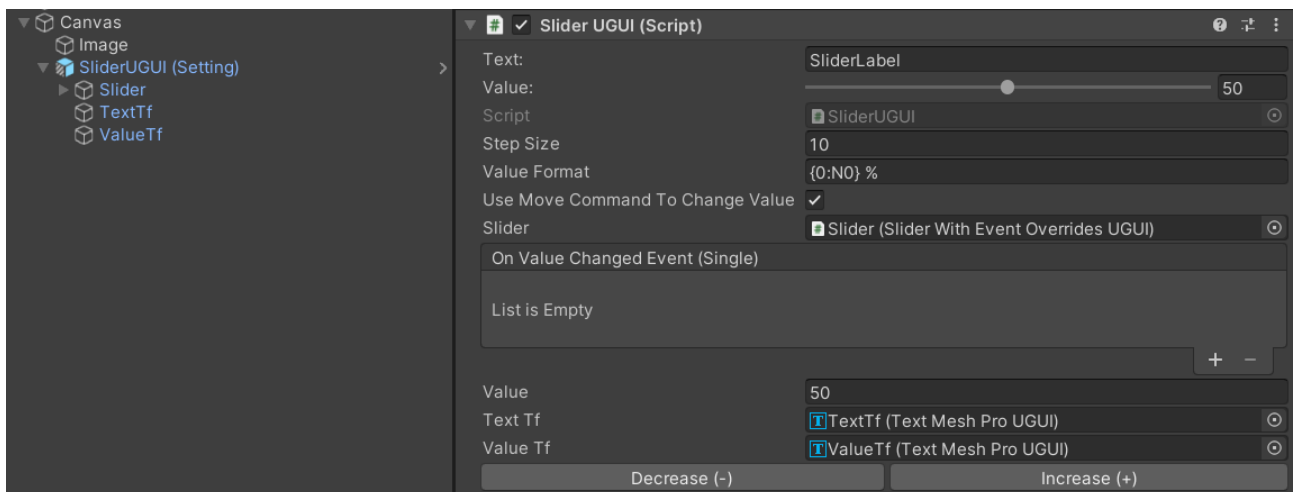
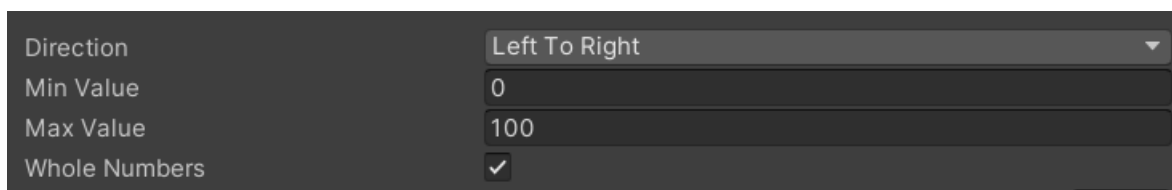


You can specify a value format which is based on the c# [Standard Numeric Format](#).

You can also specify a step size (default is 10 for a range from 0 to 100).

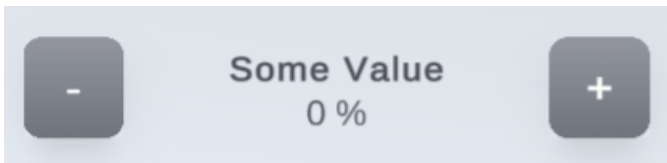
Keep in mind that controllers and gamepads usually only move the slider by one step size increment per button press. Don't make the step size too small or else console players will complain.

Some values (min, max, int/float) are configured directly on the Unity Slider.

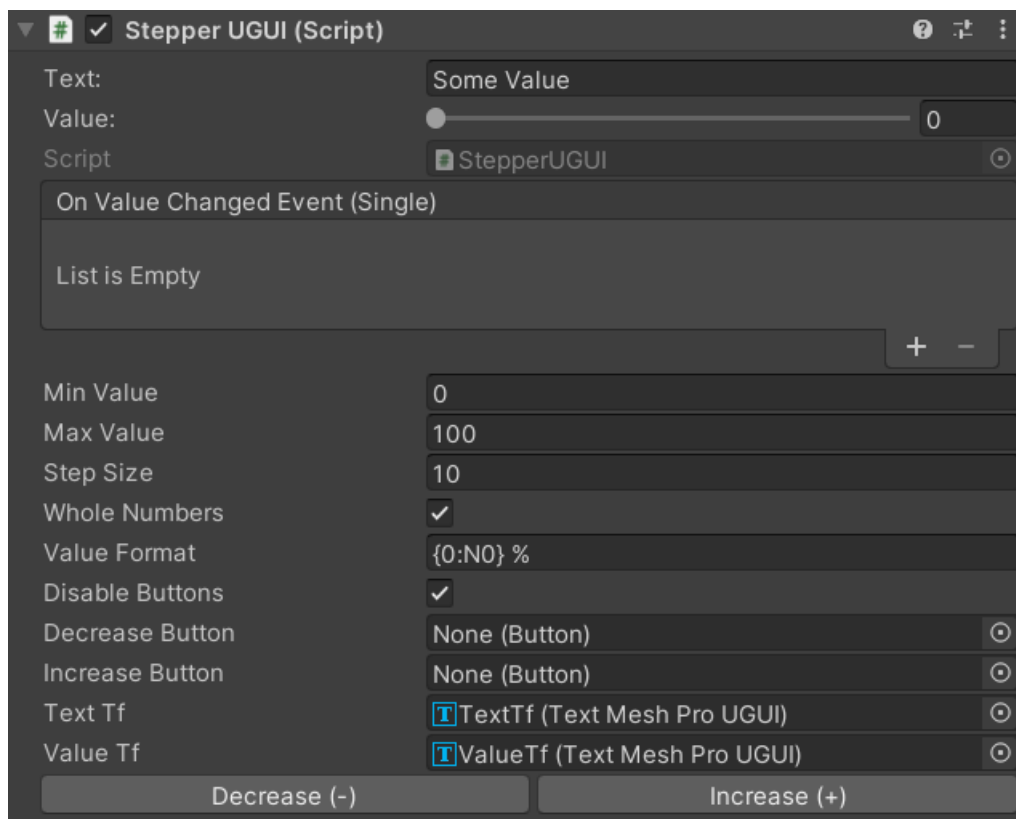


Stepper UGUI

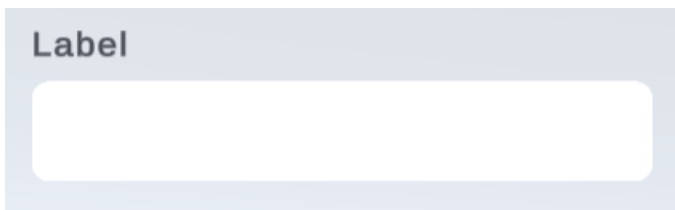
The stepper is similar to a slider but it makes the steps more explicit. This one is useful for touch interfaces or for options with few values.



You can specify a value format which is based on the c# [Standard Numeric Format](#). You can also specify a step size, min/max value and whether or not it should round to whole numbers.



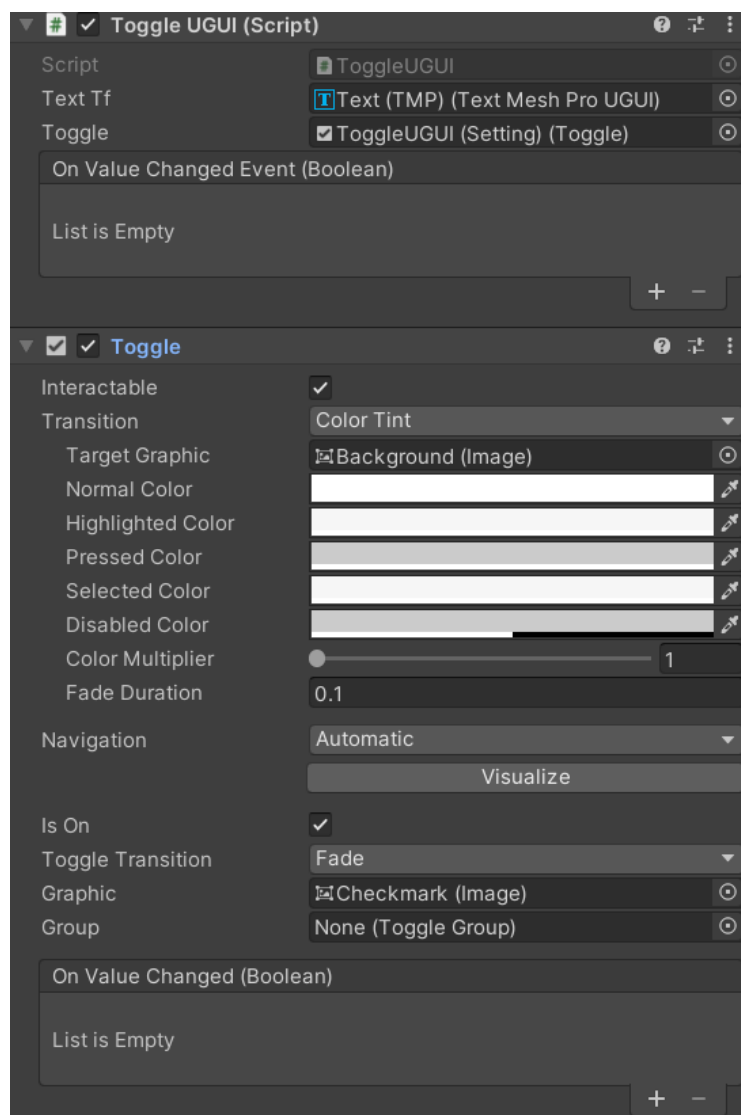
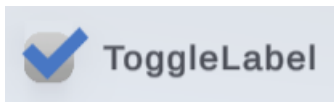
Textfield UGUI



Allows the player to enter a text.

Toggle UGUI

Use this for anything that needs to be either on or off. It is based on the Unity default Toggle.



UGUI Helpers

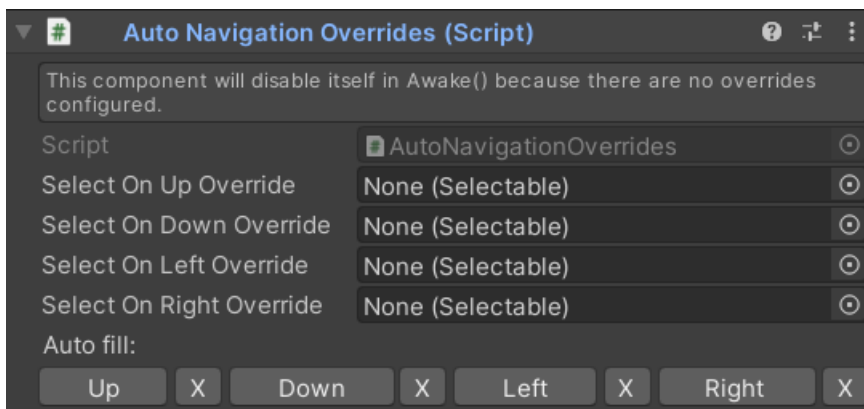
Various components are attached to the UGUI components to help out with common tasks.

Auto Navigation Overrides

TLDR: Use this instead of the "explicit" navigation mode.

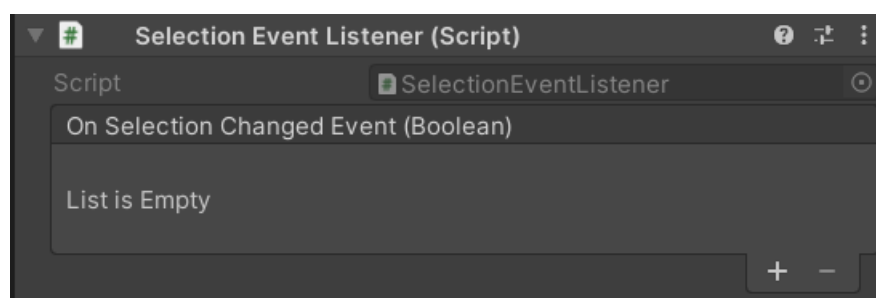
Usually auto navigation does a fine job, but sometimes it is better to set the selection by hand. The simple approach would be to use "explicit" mode instead. Sadly in explicit mode the selection will just do nothing if the target is disabled.

It would be more desirable to fall back on dynamic selection if the explicit selection does not find a valid target. This is what the overrides are for. Think of them as explicit values which fall back on auto select if the specified target can not be selected.



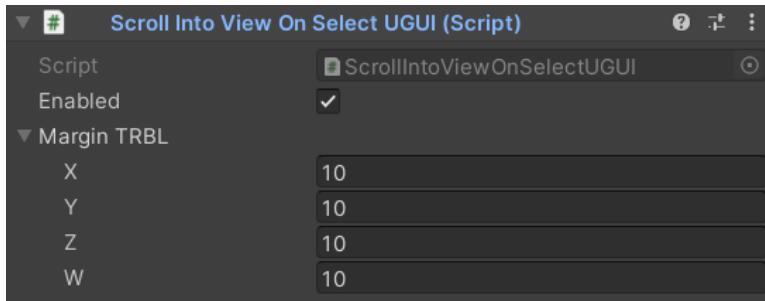
Selection Event Listener

The selection event listener allows to register callbacks for selection events. It's just a forwarder of `ISelectHandler`, `IDeselectHandler` method calls.



Scroll Into View On Select UGUI

If a UI component is selected within a scroll view then it is desirable for that elements to scroll into view (become visible). This is especially handy if a controller or gamepad is used as these can select off-screen elements too.

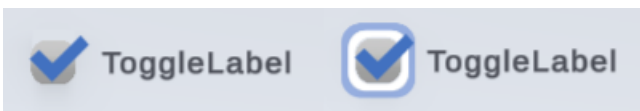


SelectionUGUI

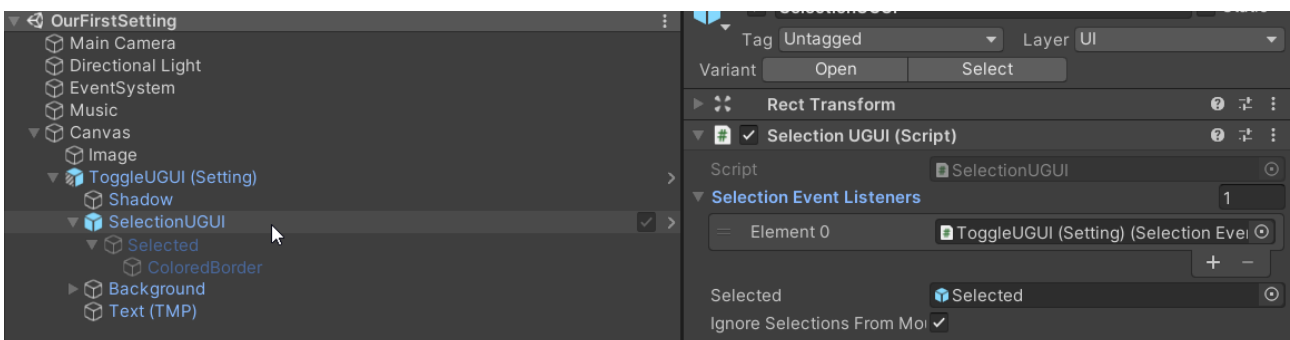
You will find a SelectionUGUI object within all UGUI component prefabs. These are the selection highlights for controller & gamepad input.

They look like this:

not selected selected (notice the border)



You can customize them for all elements by editing the „SelectionUGUI“ prefab.



UI Toolkit (full logic, fewer components, Unity 2021.2+)

UI Toolkit is the newcomer system and (possible) future of UI making in Unity.

Terms linked to it are „UI Elements“, „Visual Elements“, „Visual Tree“, „Unity Style Sheets“, „UXML“, „USS“, „com.unity.ui“ (old), „UnityEngine.UIElementsModule“ (new).

It works quite differently than ugui since it uses an XML file (.uxml) and a CSS subset (.uss) to define your UI. The most notable changes are:

A) you no longer have a hierarchy in the scene and

B) you can not add any components (MonoBehaviours) to UI elements.

The settings system has rudimentary support for UI Elements. The code aspects all work (all connections are supported) but there are much fewer ready-made UI components.

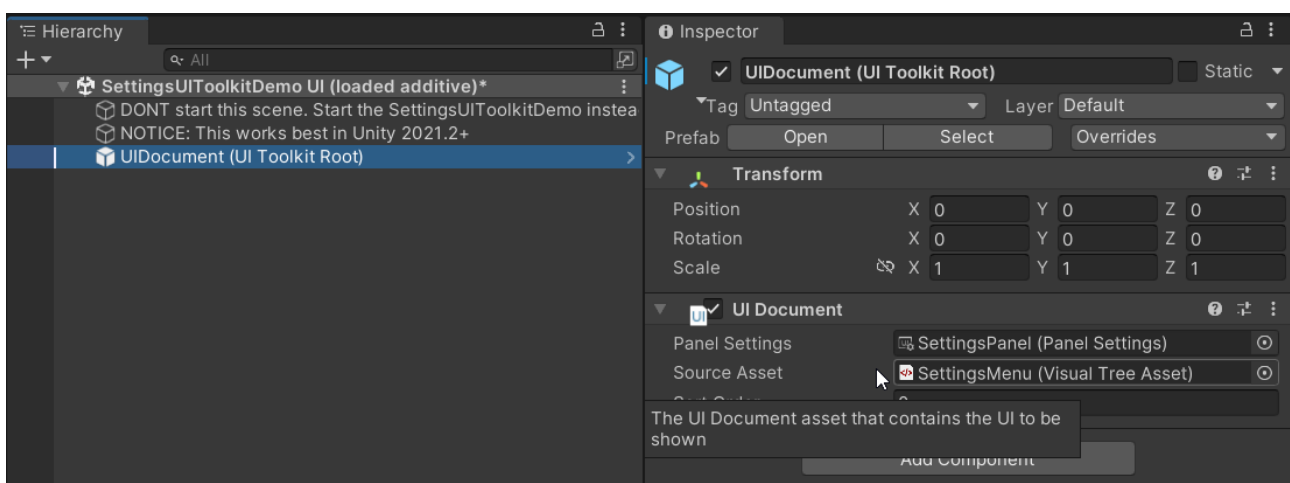
NOTICE: There was a significant change in the UI Toolkit in **Unity 2021.2**. At that point the UI Toolkit switched from being a package (com.unity.ui) to an internal module. This means that you do not have to install the UI Toolkit in Unity 2021.2+, it is already always there. The min version of the package that does support UIToolkit is **com.unity.ui 1.0.0-preview.18 (used in Unity 2020.3.30f1+)**

This also means that the old package, and with it Unity versions older than 2021.2, have dropped out of UI Toolkit support.

The old UI Toolkit package does not work very well. **It is therefore strongly suggested to only use UI Toolkit in Unity 2021.2+.** Actually, based on my experience I would suggest using it only in Unity 2023 LTS (yes, 2023).

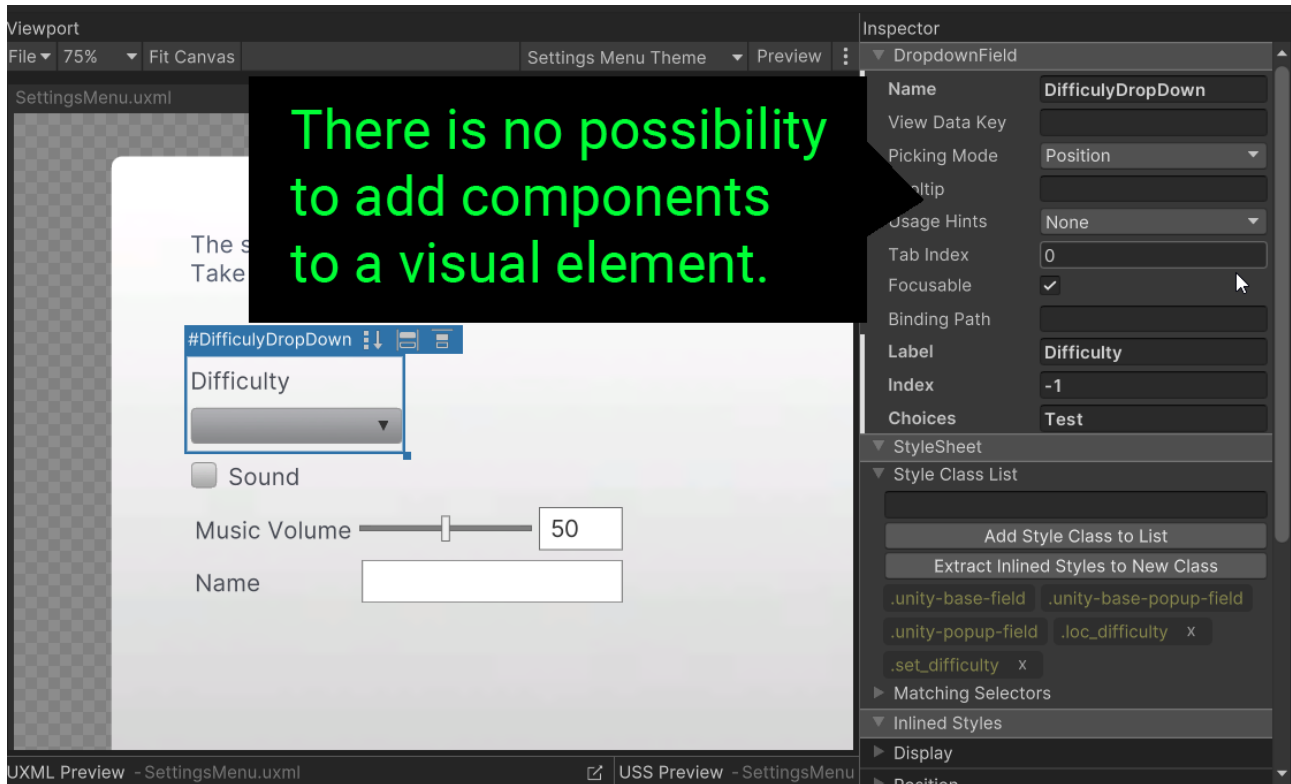
The settings system ~~may~~ will drop UIToolkit support for Unity versions below 2021.2 versions in the future (as Unity did).

Here is how a typical UI Toolkit interface looks like in the hierarchy (not much to see).



UI Toolkit Resolvers

One of the key differences between UGUI and UI Toolkit is that in UI Toolkit there is no scene hierarchy (instead there is the „Visual tree“ based on .uxml files). You may have noticed that we can not add any MonoBehaviour Components to the elements in the UI Builder.



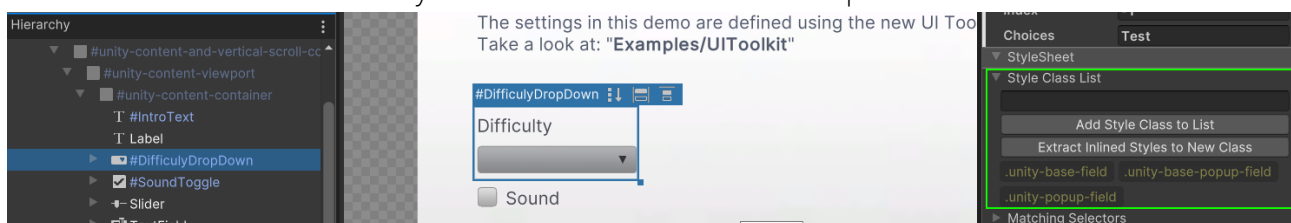
Therefore (sadly) we have to generate our own means of adding custom code.

The settings generator works around this limitation by adding components to the scene which are linked to visual elements via **class names***.

*This may be changed in the future to use Bindings but at the time of writing the implementation was still a [work-in-progress](#). Same goes for [localizations](#).

Linking a Visual Element to a Setting (guide)

First let's look at what we usually have at the start. Here is a DropdownField in the UI Builder:



Notice the marked area on the right. There we can add custom class names to the element. You can find infos on what a „class name“ is in the [Unity Manual](#) (basically it's just a string).

Now to establish a link to a visual element we first need a way to find it inside the UIToolkit Hierarchy (aka Visual Tree). Note: this is NOT the scene hierarchy!

Since [binding for runtime](#) is still a [work-in-progress](#) we need some other way of identifying the elements. To do this we will use the „class names“. We will add a unique class name to every element that should be used by the settings system.

Step 1: Add a unique „set_“ class name

Let's add the class name. **Make sure that it starts with „set_“**. This is the signal to the settings system that the class name is meant for a setting (example: „set_whatEver“). Also **make sure the class name is unique** or else the settings system will have trouble differentiating it from other classes. Hint: Using the setting id after the „_“ is a nice way of keeping it unique.

Step 1: Select the element and type in the class name (set_[your-setting-id])



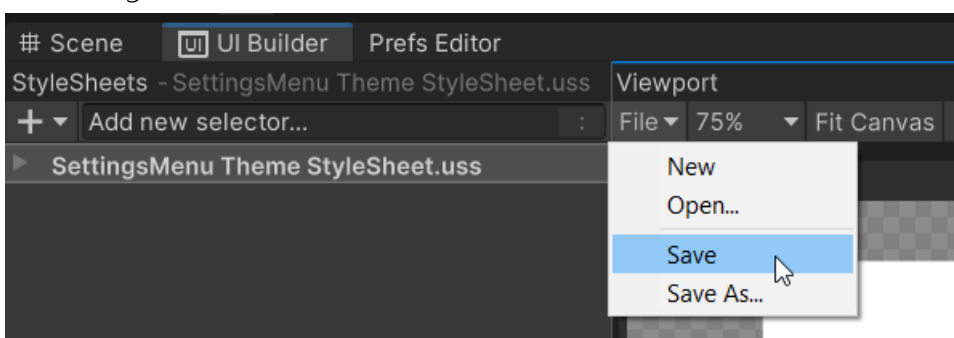
Step 2: Press the „Add Style Class“ button to add the class.



Step 2: Check if the class was added. Sadly if you have a typo in it you will have to delete and reenter (hopefully Unity will improve that workflow some day).

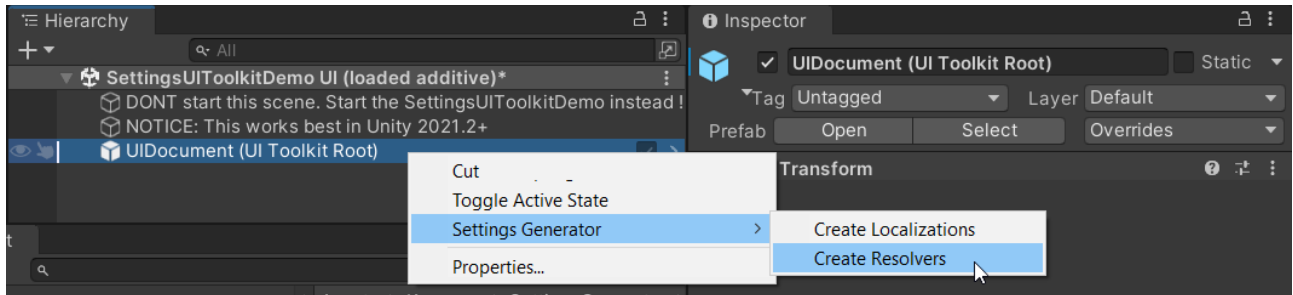


Don't forget to hit „Save“ in the UI Builder afterwards:

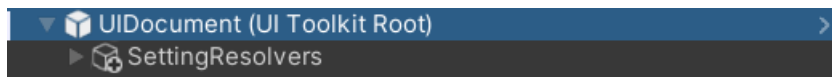


Step 2: Generate the resolver components

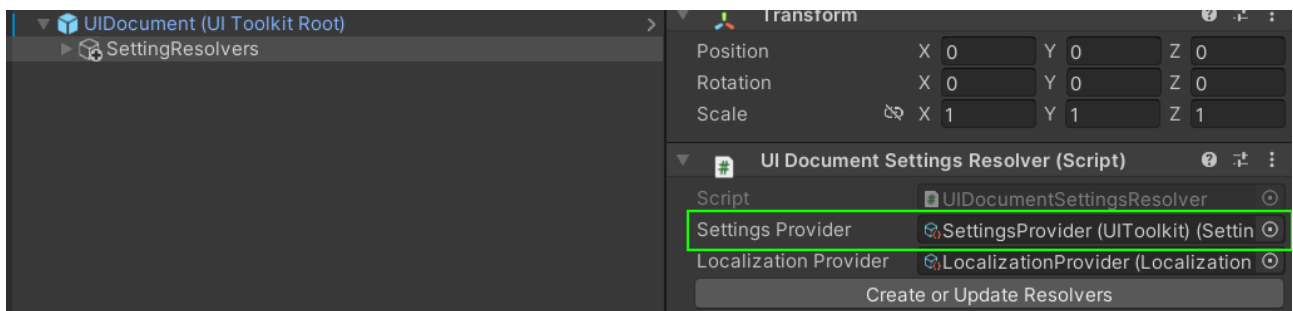
Okay now that we have added some class names we can generate those components we need (the setting resolvers). To do that **Right-Click** on the UIDocument in your scene and choose **Settings Generator > Create Resolvers**.



This will generate a child object inside the UIDocument.

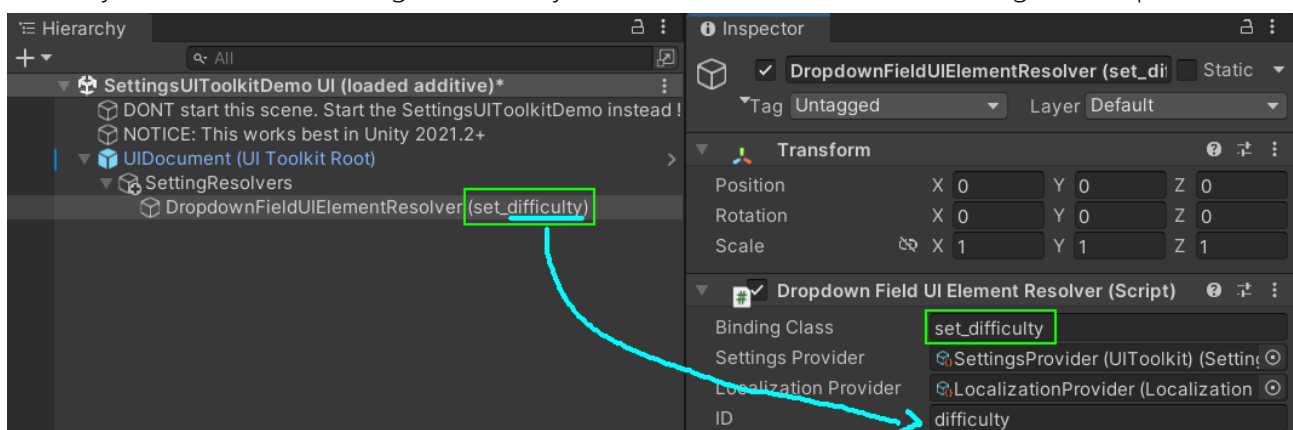


Select the „SettingResolvers“ object and check in the inspector that it has a SettingsProvider assigned. By default it will use the first one it finds but maybe you want a different one.



Once you have the right provider assigned hit the „Create or Update Resolvers“ button.

Now if you unfold the „SettingResolvers„ you should find a resolver matching the drop down:



The class name you have set should be listed in the „Binding Class“ field. This tells the resolver which visual element (DropDownField) it belongs to.

You may have also noticed that the ID has been prefilled too. That was also done based on your class name. The class name without the „set_“ part is the ID (set_**difficulty**).

However, you can change both the BindingClass and ID here at any time.

Don't forget to save the scene after generating the resolvers!

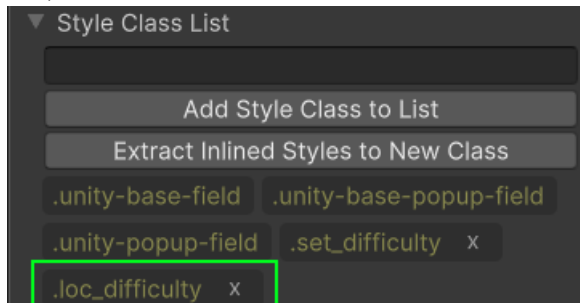
Now you can keep adding „set_“ class names to your ui elements and press that „Create or Update Resolvers“ button. NOTICE: it will always regenerate all the setting resolvers from scratch.

If you choose your class names well you will never have to touch the resolver components.

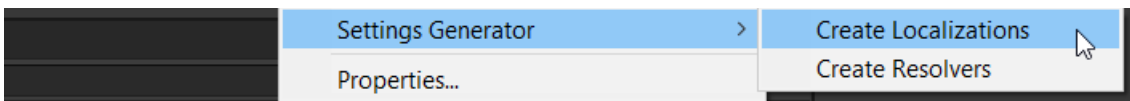
Linking a Visual Element to a Localization

It's the same workflow as with the settings. The only difference is that instead of the „set_“ prefix you have to use „loc_“ in your class names.

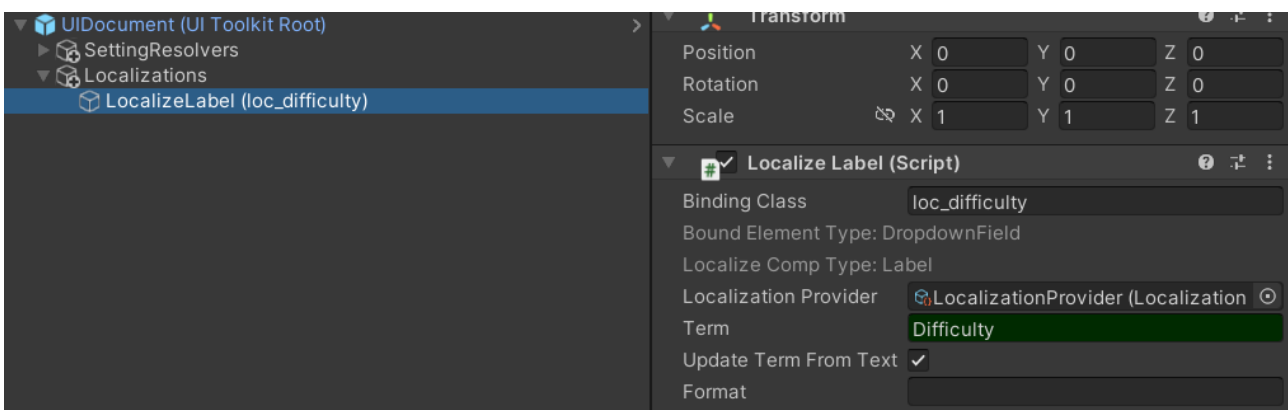
Step 1:



Step 2:



Result:

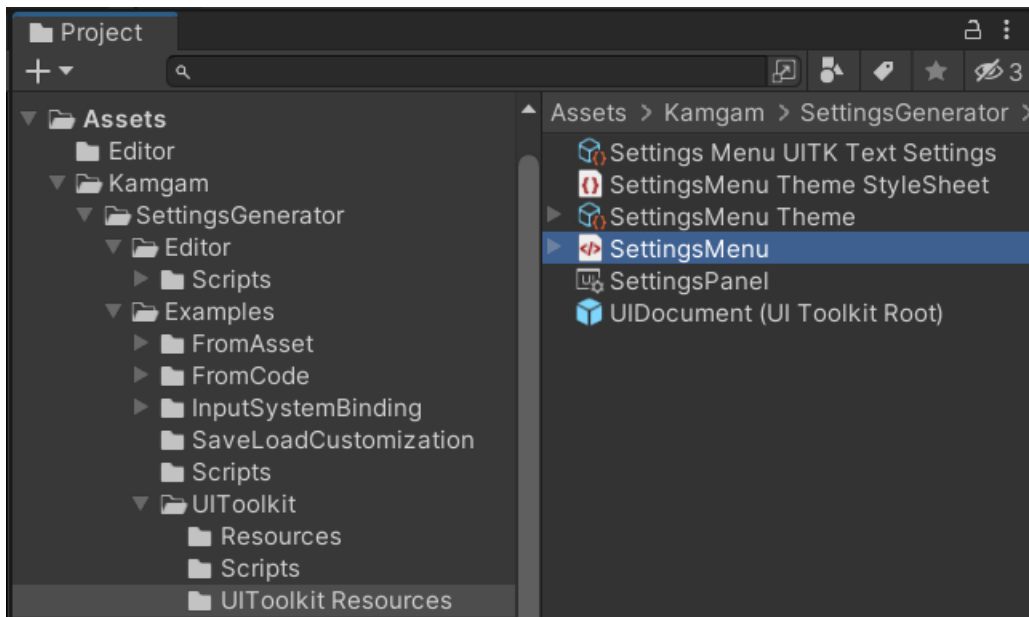


UI Toolkit Components

At the moment there are no custom made components for UI Toolkit. However there is a range of resolvers and helpers supporting Unity's existing UIElement components. You can find them used in the demo under:

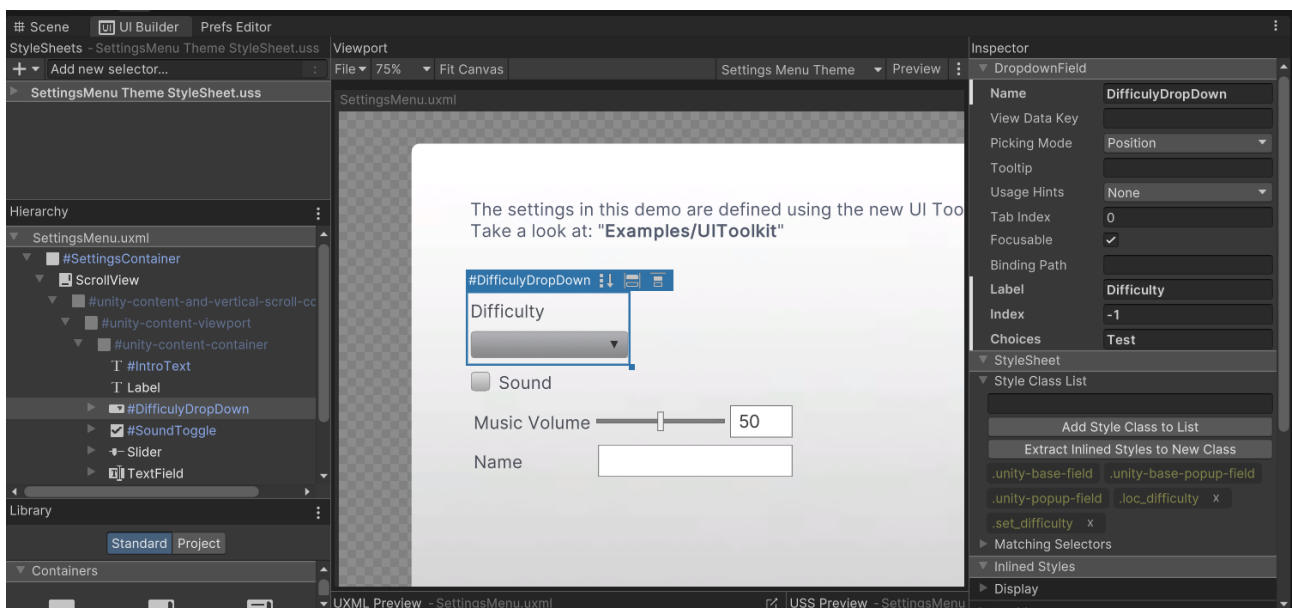
Assets/Kamgam/SettingsGenerator/Examples/UIToolkit/SettingsUIToolkitDemo.

To inspect them open the SettingsMenu.uxml in the UIBuilder window.



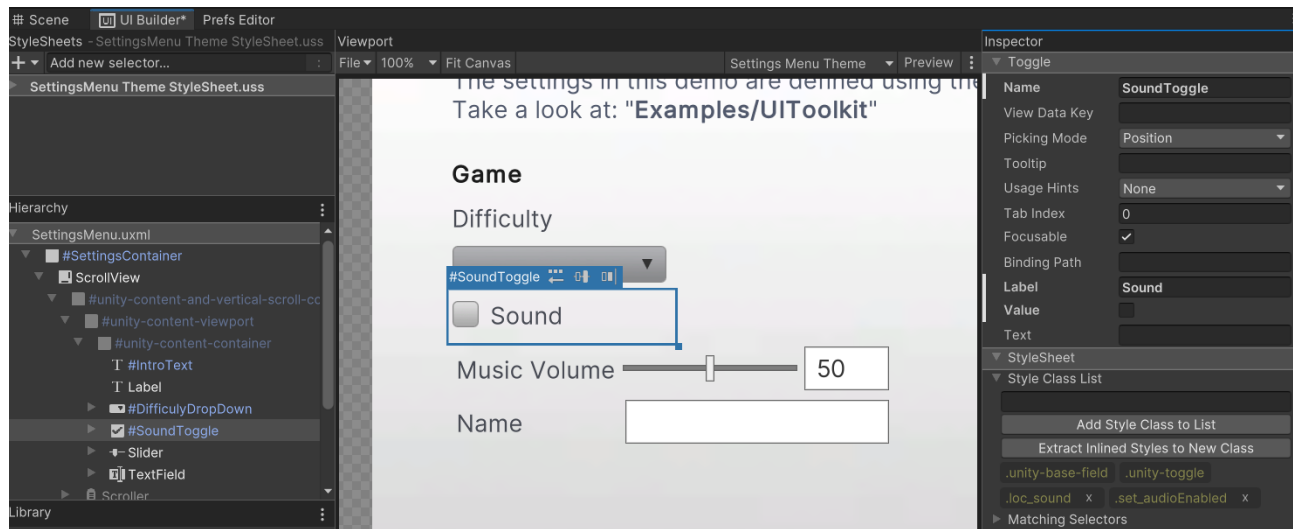
DropDownField

Use the default Unity DropDownField from the Library in the UI Builder. It can be connected to an option setting via a „**DropDownFieldUIElementResolver**“.



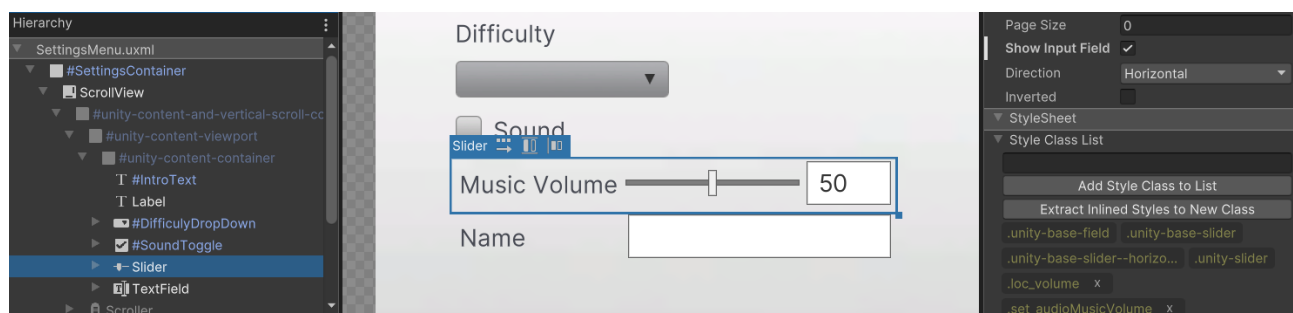
Toggle

Use the default Unity Toggle from the Library in the UI Builder. It can be connected to an boolean setting via a „**ToggleUIElementResolver**“.



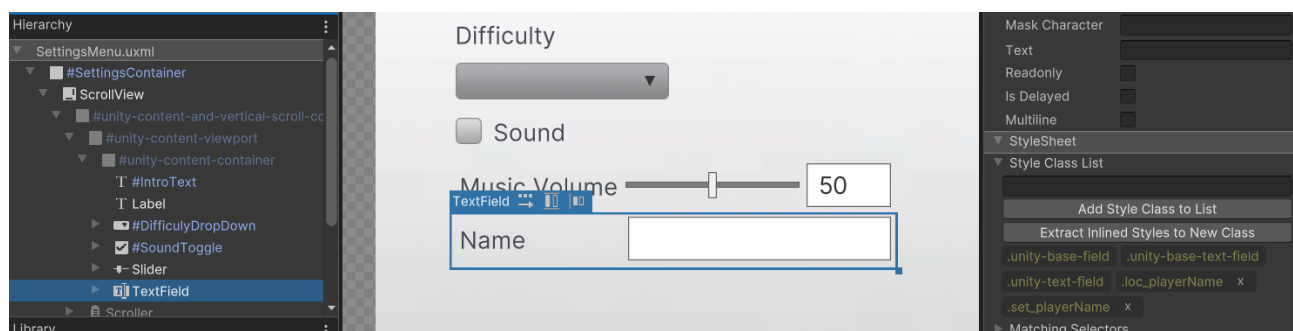
Slider

Use the default Unity Toggle from the Library in the UI Builder. It can be connected to an boolean setting via a „**SliderUIElementResolver**“.



TextField

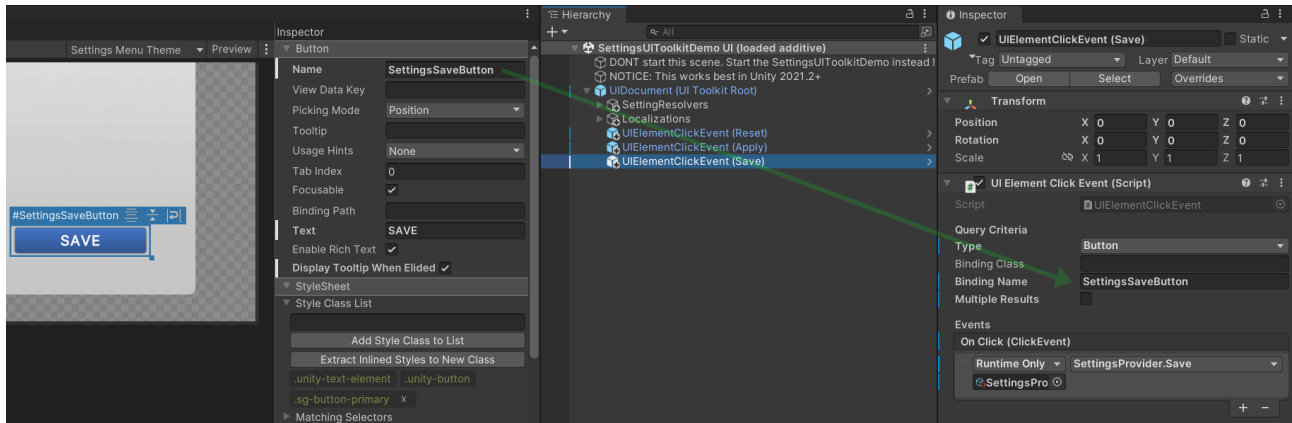
Use the default Unity Toggle from the Library in the UI Builder. It can be connected to an boolean setting via a „**TextFieldUIElementResolver**“.



Helper: UIElementClickEvent

This is more of a general helper than a setting related component. You can add it as a CHILD to the UIDocument and specify some query criteria to target one (or more) elements. It then allows you to add UnityEvents for clicking. You can find the prefab under:

Assets/Kamgam/SettingsGenerator/Libs/UIToolkitComponentsForSettings/Runtime/Prefabs/UIElementClickEvent



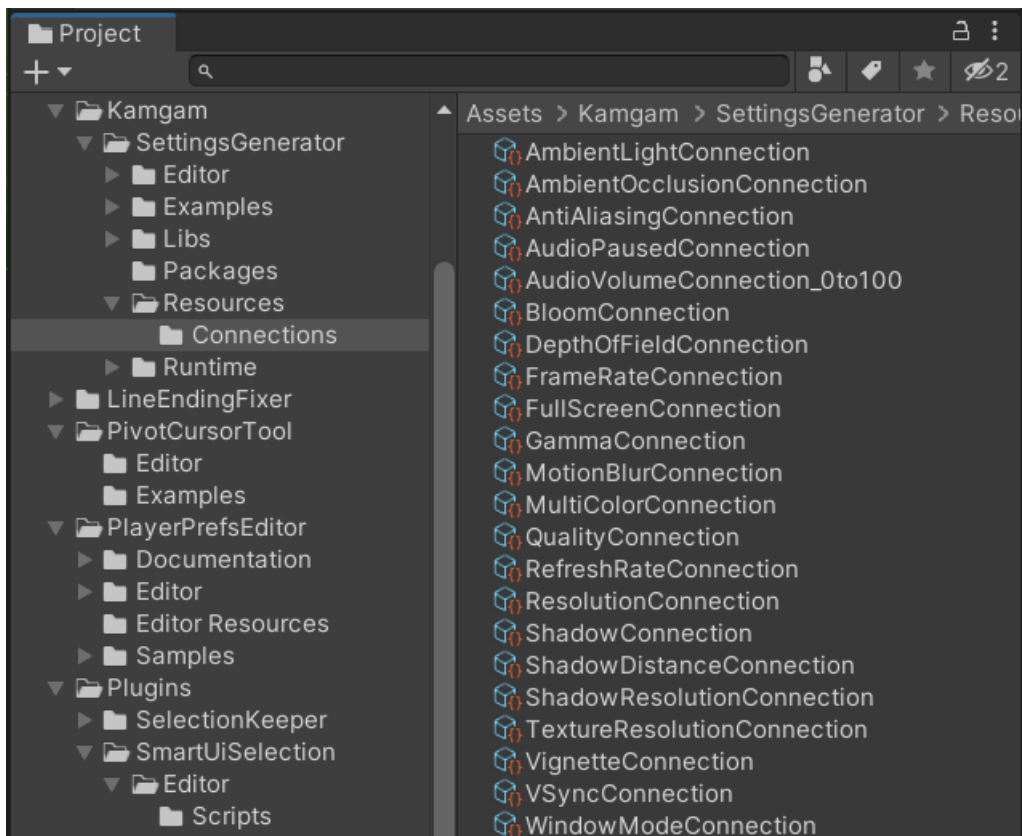
Helper: UIElementEvents

Same as UIElementClickEvent just with many more events (pointer, key, drag, focus, blur, change).

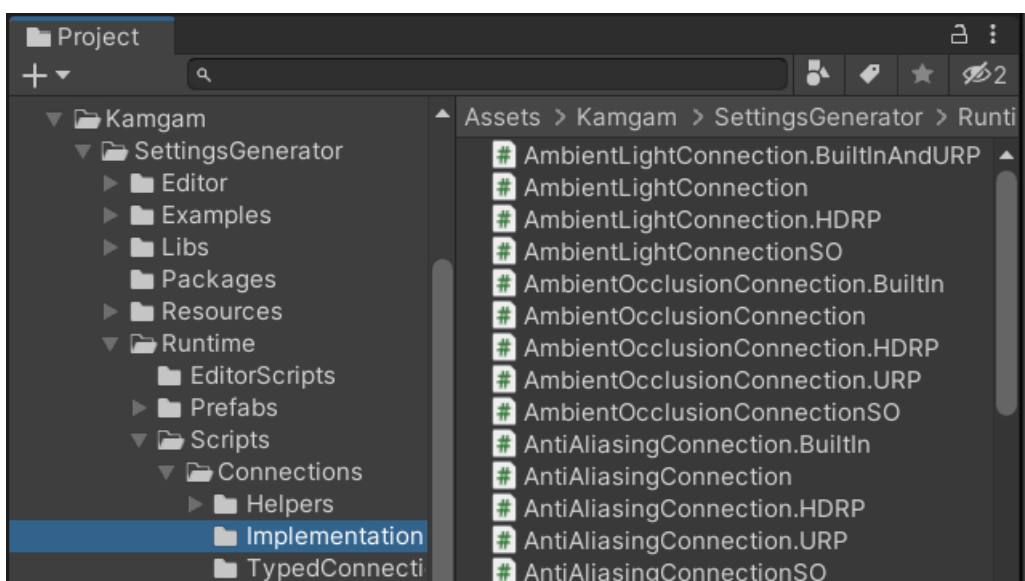
Connections

There is an extensive list of predefined connections. All of them support HDRP, URP and Built-in renderers.

ScriptableObjects: As explained in the „Overview“ section connections can be hooked up with settings to drive the value from code. You can find them under Resources/Connections.



CODE: Most of the connections are split into three partial classes (Built-In, HDRP, URP). You can find the implementations within the Runtime/Scripts/Connections folder.



Ambient Light

Controls the ambient light intensity.

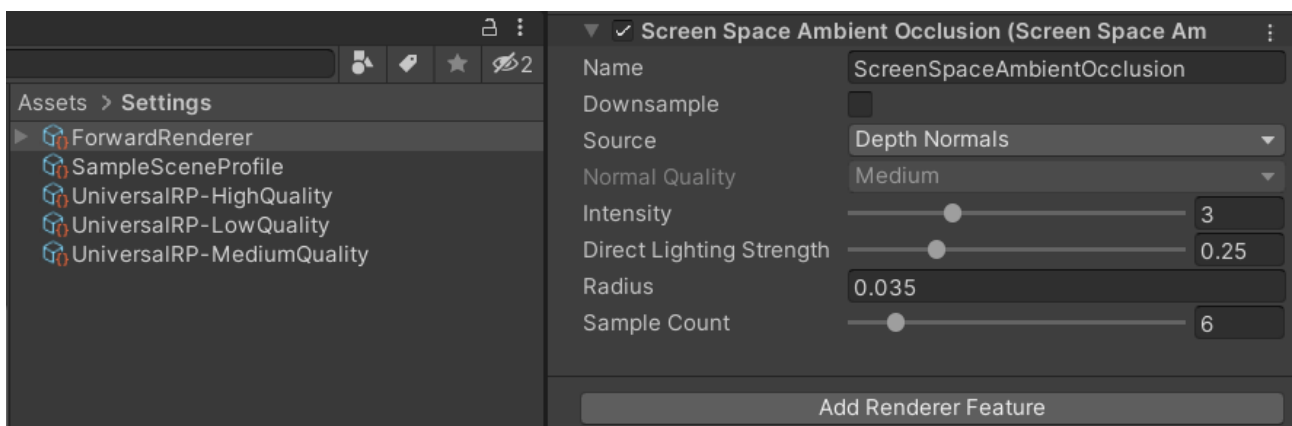
Ambient Occlusion (SSAO)

On/Off for Screen Space Ambient Occlusion. There are some caveats with this one.

Caveats (mostly URP)

In URP Ambient Occlusion is only supported in URP 10+ (Unity 2020.2).

In URP SSAO has to be added as a `RendererFeature` to the `ForwardRenderer`. It is NOT part of the normal post processing stack.



In URP the performance cost of SSAO is always paid since it is always enabled. To really disable it you will have to set the static property **AmbientOcclusionConnection.UseActiveStateToDisable** to true.

NOTICE: If you enable it then the functionality of SSAO depends on the availability of the disabled shader variants. These may be stripped from builds. You will have to disable shader stripping under `ProjectSettings > Graphics > URP Global Settings > Shader Stripping` (both "post pro" and "unused").

See: <https://forum.unity.com/threads/turn-urp-ssao-on-and-off-at-runtime.1066961/#post-8613702>

In HDRP disabling the shadows will also disable SSAO within the shadows. If anyone finds a workaround for this then please let us know.

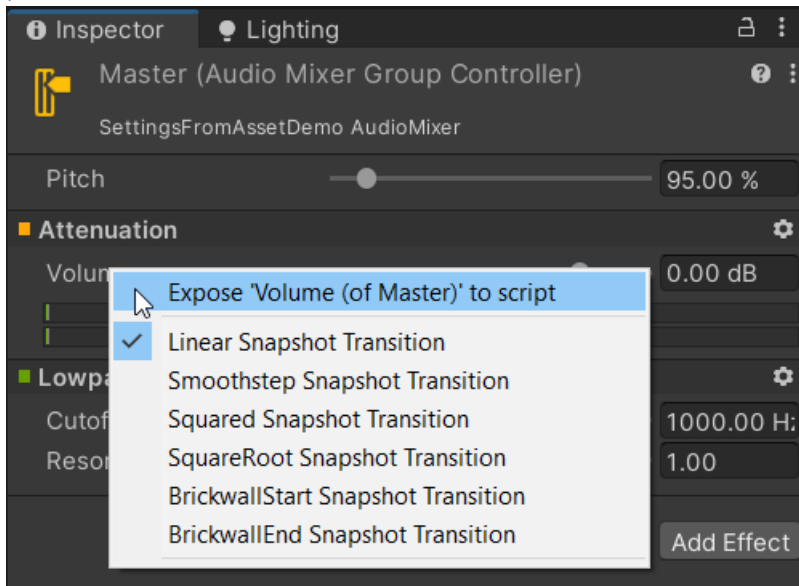
Anti Aliasing

Selectable options are based on your renderer settings.

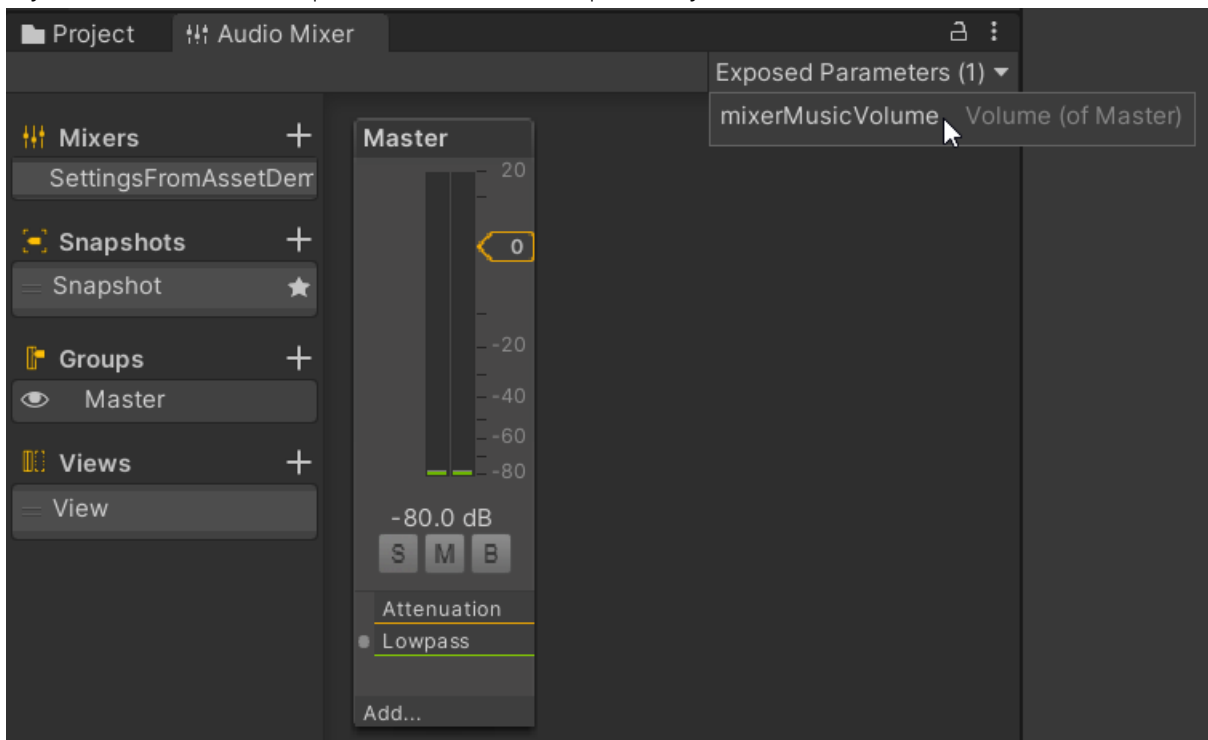
Audio Mixer

Allows to control exposed parameters of [AudioMixers](#).

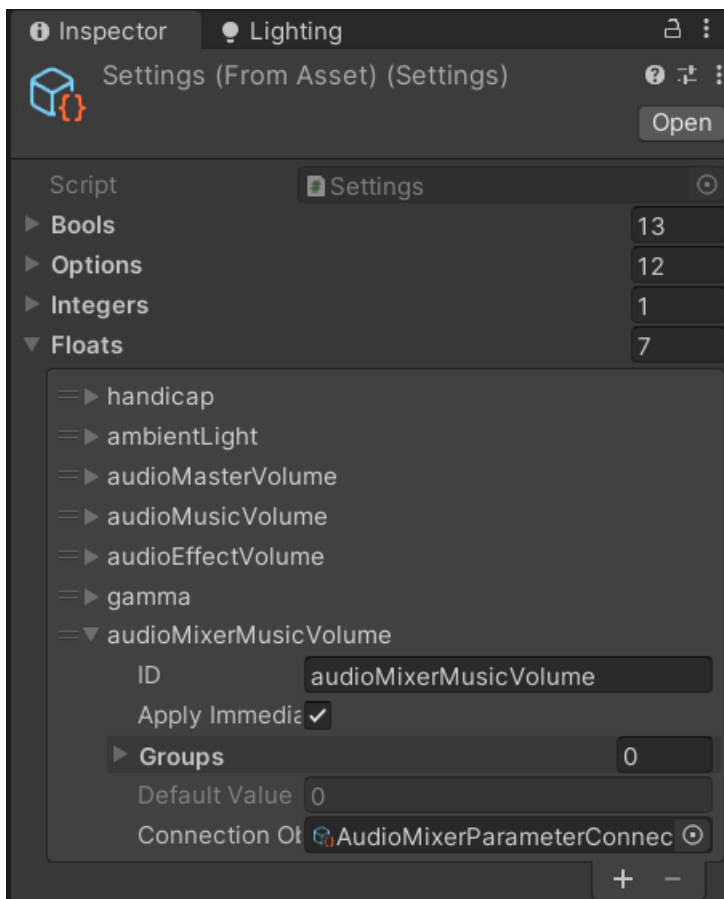
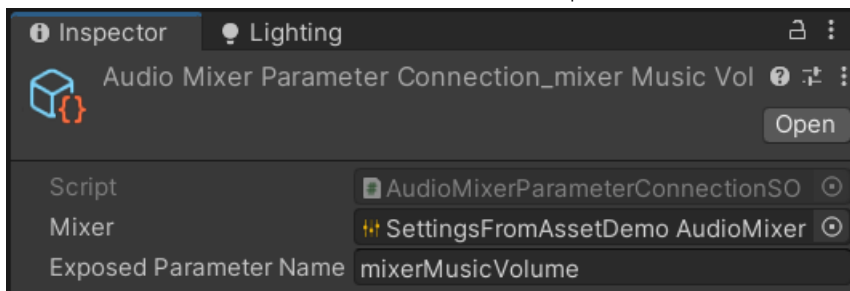
It requires you to expose the parameters for modification and you will need one Connection per parameter.



If you double-click the parameter in the dropdown you can rename it.



The connection will need the name of the parameter so it can find it.



Be careful with the values you plugin into it. Don't start off with 20dB Volume! Your audio equipement may not like that too much.

Audio Paused

On/Off for Audio: Pauses the Audio Listener (useful for global audio on/off).

Audio Volume

Controls the global Audio Listener volume (it changes the „AudioListener.volume“ value). Useful for a master volume setting. For volume controls of individual audio sources please use the „AudioSourceVolume“.

Audio Source Volume

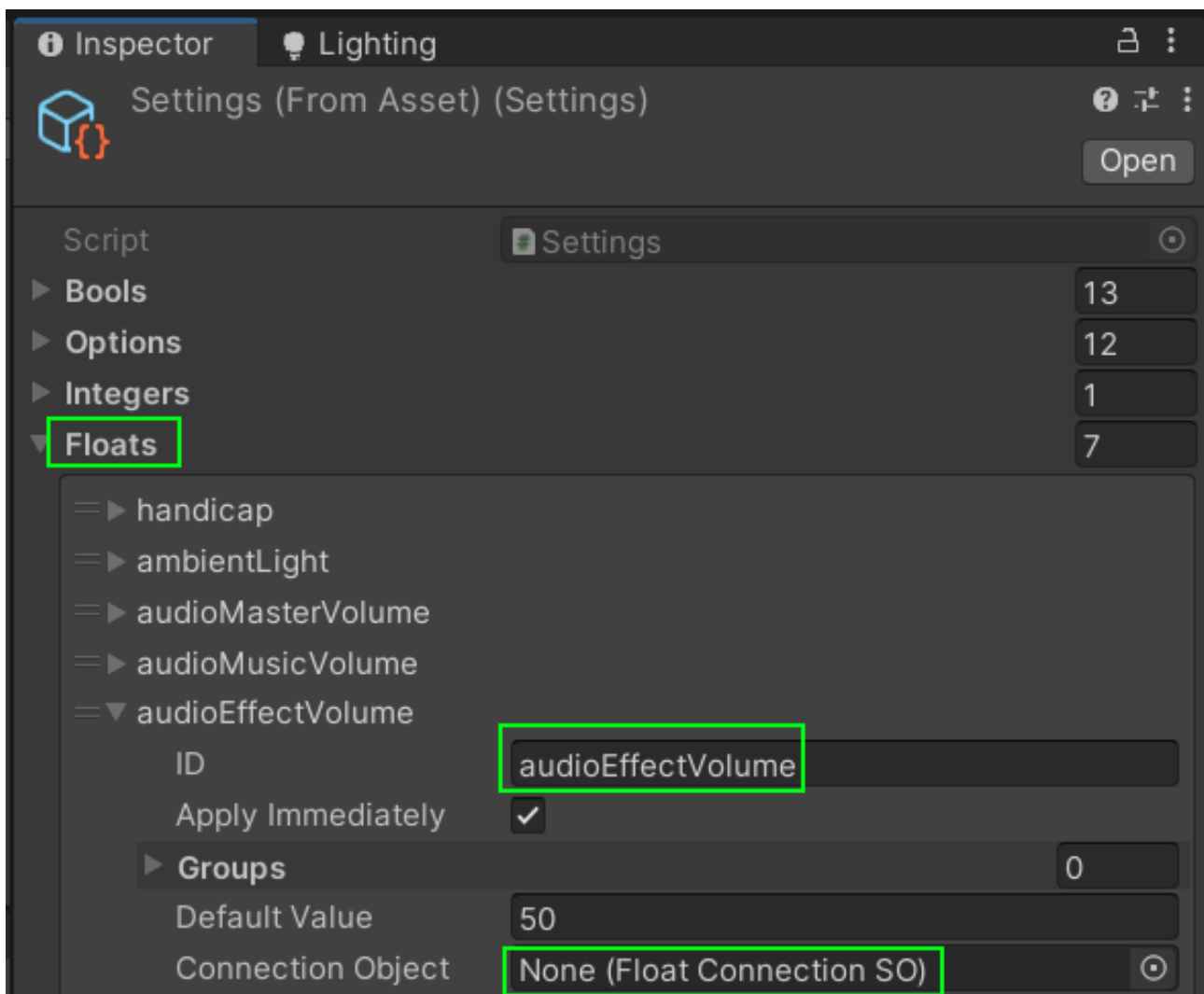
Controls the volume of one or more specific AudioSource Components. It does NOT use a connection object in the setting.

How to make sound groups for „effects“, „music“, „voice“, ...?

TLDR:

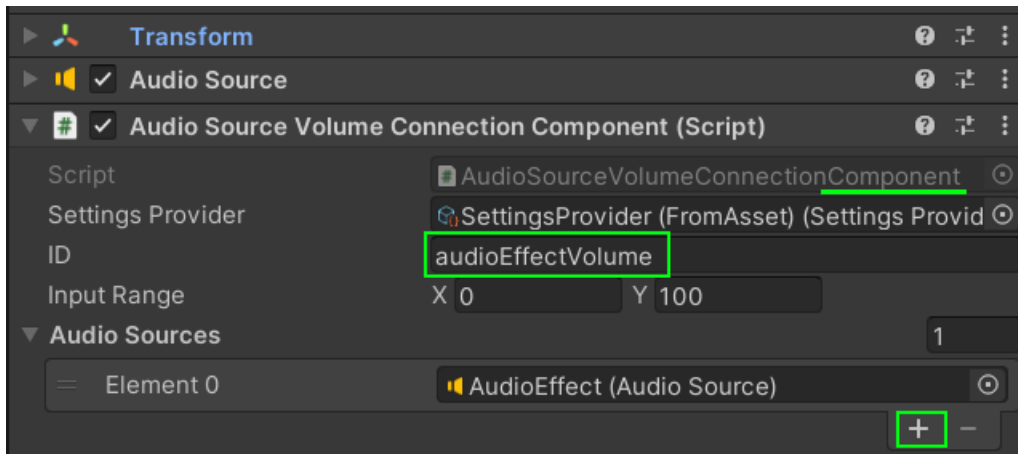
- 1) Create a float setting for reach group (do NOT add a Connection, leave it empty).
- 2) Add AudioSourceVolumeConnectionComponents to AudioSources and specify the id.

To control volumes by "group" please add one float setting for each (like musicVolume, effectVolume, ...). Do NOT add a VolumeConnection.



Since AudioSources live on GameObjects in the scene they require you to add an AudioSourceVolumeConnectionComponent.

Like this:



In the example above the `AudioSourceVolumeConnectionComponent` controls only one `AudioSource` but you can add more. You can also have multiple components referencing the same ID.

NOTICE: The audio effect volume in the demo is not controlled by a `VolumeConnection` setting but a normal **float** setting.

The `AudioSourceVolumeConnectionComponent` will create a `VolumeConnection` on-demand in `Start()` and use that. But this part is automatic and hidden from the user so you may not even notice it.

The important part is that **you can use as many `AudioSourceVolumeConnectionComponents` with as many different (or equal) setting ids as you want.** There is no dedicated place to define groups like music/fx/voices (except for the float settings section).

Bloom

On/Off for the bloom post processing effect.

Depth Of Field

On/Off for the depth of field post processing effects.

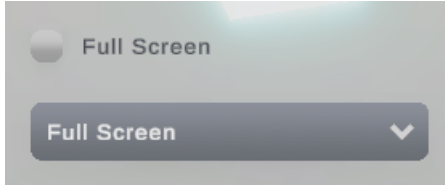
Frame Rate

Options to control the target frame rate. Default options are: 30, 60 and 120.

Full Screen

On/Off of for full screen mode.

NOTICE: The „FullScreen“ connection and the „Window Mode“ connection should NOT be used at the same time as they can contradict each other, like this:



Is fullscreen now ON or OFF?

What will happen in this case is that first the FullScreen will be disabled by the FullScreenConnection (checkbox) but afterwards it will be enabled by the WindowModeConnection (drop down). This will result in an ugly window bar being visible for one or two frames. That's why you should only use one OR the other. In the examples both are added only for demonstration purposes.

GetSetConnection<T>

A generic connection which you can use to forward a value directly to a callback function.

Gamma

Range for the gamma correction post processing effect (-1 to +1).

InputBindingConnection

This connection stores the bound key for an action as a string.
For more details please refer to the **Input Rebinding** section.

Motion Blur

On/Off for the motion blur post processing effect. **NOTICE:** URP does not have a per-object motion blur. This means for example that an object rotating in place will not be blurred. More on that in the [Unity manual](#).

Monitor

Allows you to switch the display monitor of the main window. **⚠ NOTICE:** This requires Unity 2021.2 or higher since that's when Unity added the monitor switching API.

Quality

Changes the global quality setting. Options are taken from the graphics quality settings automatically.

Refresh Rate

Options for monitor refresh rates. Options are taken from the refresh rates supported by the monitor.

NOTICE: Whether or not the actual refresh rate is changed depends on the OS and the hardware. Usually refresh rates can only be changed on exclusive window mode on Windows. The refresh rate has no effect in the Editor.

If you want to control the fps of your application use the „[Application.targetFrameRate](#)“ API in combination with and int setting or the „FrameRateConnection“.

The refresh rate connection as some options:

CacheRefreshRates (default: true)

If enabled then the refresh rates are queried only once from the system and are then reused.

LimitToCurrentResolution (default: false)

If enabled then only those refresh rates which are usable with the current resolution are listed. That list may be much shorter than the full list (often just one).

MinRate (default: 0)

Refresh rate in Hz. Refresh rates below this are ignored. Equal rates are still included.

MaxRate (default: 1000)

Refresh rate in Hz. Refresh rates above this are ignored. Equal rates are still included.

Resolution

Options for the game resolution. Options are taken from the resolutions supported by the monitor.

NOTICE: The resolution has no effect in the Editor.

The resolution connection as some options:

CacheResolutions (default: true)

Disable if the resolutions change very often.

LimitToCurrentRefreshRate (default: false)

If enabled then only those resolution options which match the current resolution refresh rate are listed. That list may be much shorter than the full list.

LimitToUniqueResolutions (default: true)

If enabled then only one resolution per frequency will be listed.

For example there may be two resolutions: 640x480 @60Hz and 640x480 @75Hz

If enabled then only one of these will be in the list. It will choose the one which has the closest frequency to the currently used frequency.

SkipRefreshRatesWith59Hz (default: true)

If enabled then resolutions with a refresh rate of 59 Hz will be skipped if (and only if) there is an alternative with 60 Hz.

AddRefreshRateToLabels (default: false)

Should the refresh rate (frequency) be added to the labels.

Example without: 1024x768

Example with: 1024x768 (60Hz)

Shadow

On/Off for shadows (and contact shadows if you use HDRP).

Shadow Distance

Options for max shadow distance. The options are based on your graphics quality settings.

Shadow Resolution

Options for max shadow map resolutions. The options are based on your graphics quality settings. Notice: the shadow resolution is spread out over the shadow distance. Thus the highest shadow resolution with the lowest shadow distance gives the best quality (that's a little counter intuitive).

Texture Resolution

Options for texture resolutions. The default options are: full-res, 1/2 res, 1/4 res, 1/8 res.

Vignette

On/Off for the vignette post processing effect.

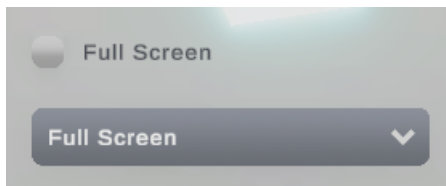
V-Sync

On/Off for vertical sync.

Window Mode

Use this if you need more control than just "full screen on/off". Otherwise use the "Full Screen" setting. You rarely need this.

NOTICE: The „FullScreen“ connection and the „Window Mode“ connection should NOT be used at the same time as they can contradict each other, like this:



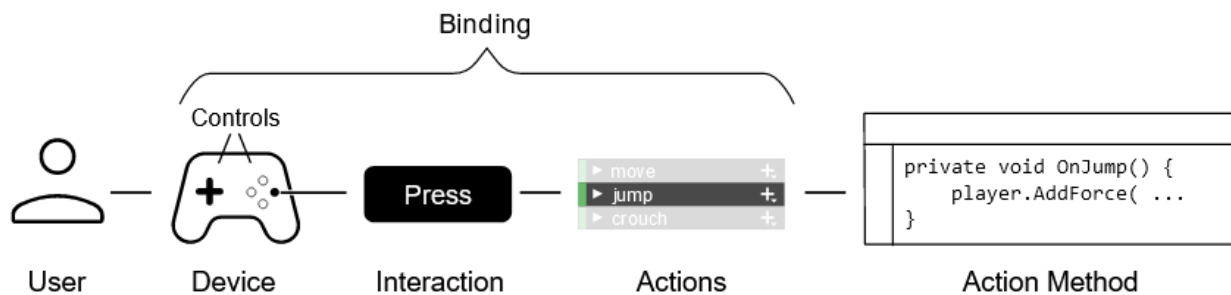
Is fullscreen now ON or OFF?

What will happen in this case is that first the FullScreen will be disabled by the FullScreenConnection (checkbox) but afterwards it will be enabled by the WindowModeConnection (drop down). This will result in an ugly window bar being visible for one or two frames. That's why you should only use one OR the other. In the examples both are added only for demonstration purposes.

Input Rebinding (Using the new Input System)

Rebinding Overview

With Unity 2019 a new [InputSystem](#) was introduced. Among other things it also features the possibility of binding inputs to actions.



Source: <https://docs.unity3d.com/Packages/com.unity.inputsystem@1.5/manual/Concepts.html>

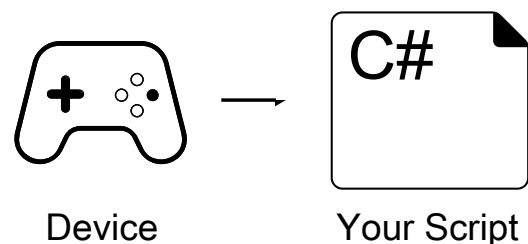
There are four major ways of how your game can interact with the InputSystem.

From the Unity Manual:

[Directly Reading Device States](#)

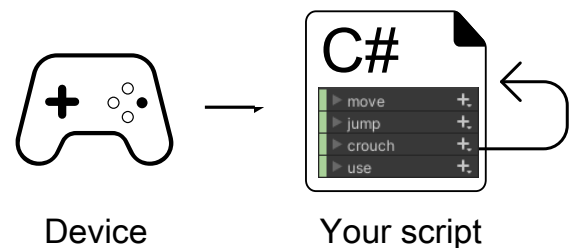
Your script explicitly refers to device controls and reads the values directly.

Can be the fastest way to set up input for one device, but it is the least flexible workflow. [Read more](#)



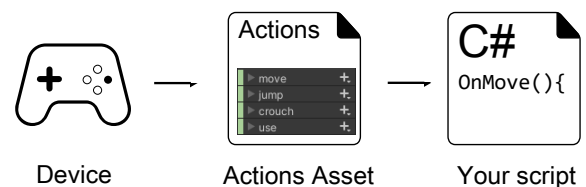
[Using Embedded Actions](#)

Your script uses the InputAction class directly. The actions display in your script's inspector, and allow you to configure them in the editor. [Read more](#)



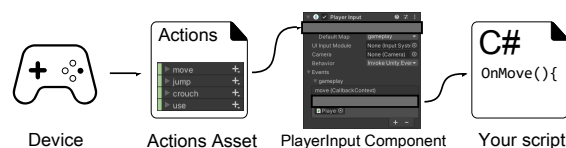
[Using an Actions Asset](#)

Your script does not define actions directly. Instead your script references an Input Actions asset which defines your actions. The Input Actions window provides a UI to define, configure, and organize all your Actions into useful groupings. [Read more](#)



Using an Actions Asset and a PlayerInput component

In addition to using an Actions Asset, the PlayerInput component provides a UI in the inspector to connect actions to event handlers in your script, removing the need for any intermediary code between the Input System and your Action Methods. [Read more](#)



Source: <https://docs.unity3d.com/Packages/com.unity.inputsystem@1.5/manual/Workflows.html>

NOTICE:

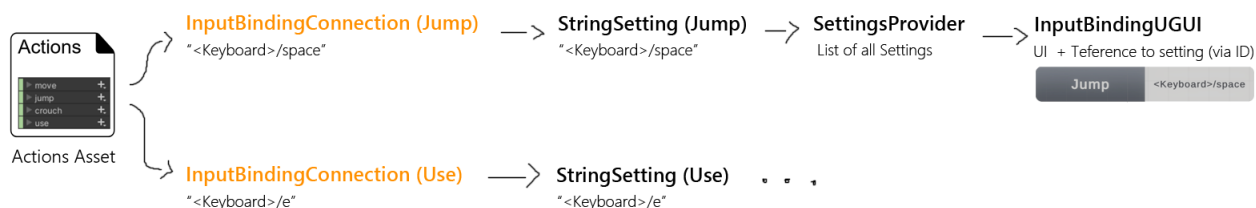
To support Input rebinding the **Settings System requires an Actions Asset**. Therefore it only supports the „Using an Actions Asset“ and „Using an Actions Asset and a PlayerInput component“ workflows.

Embedded Actions and direct Input access are not supported since there simply is no way for the Settings System to know where in your code the actions are and how to access them.

However if you are a programmer then you can forward the binding path via the `AddChangeListener()` API of the connection.

Rebinding Tutorial

The Settings System needs to know which Input Actions (Bindings) do exist so it can connect the Settings and UI with it.



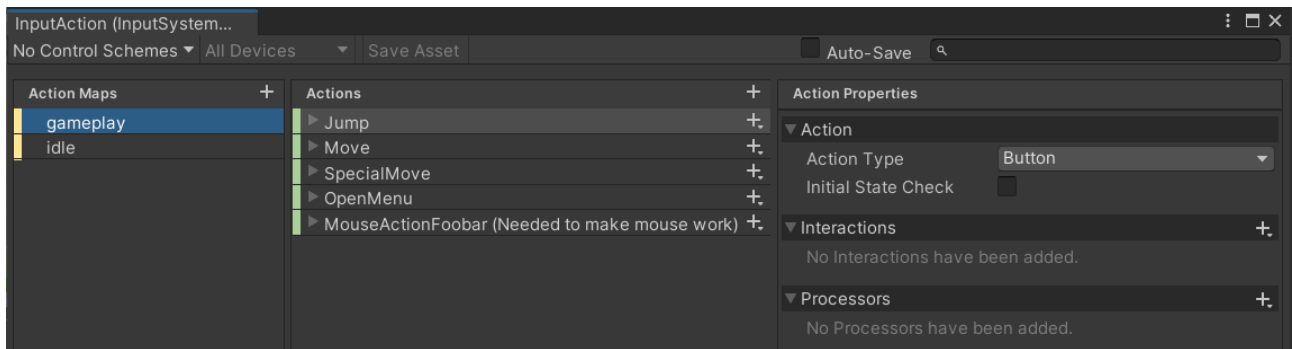
To do that we will have to generate an **InputBindingConnection** object for each binding in your Actions Asset. The following section will teach you how to set this up.

If you want to skip to the end then you can find a complete example under:

Assets/Kamgam/SettingsGenerator/Examples/InputSystemBinding.

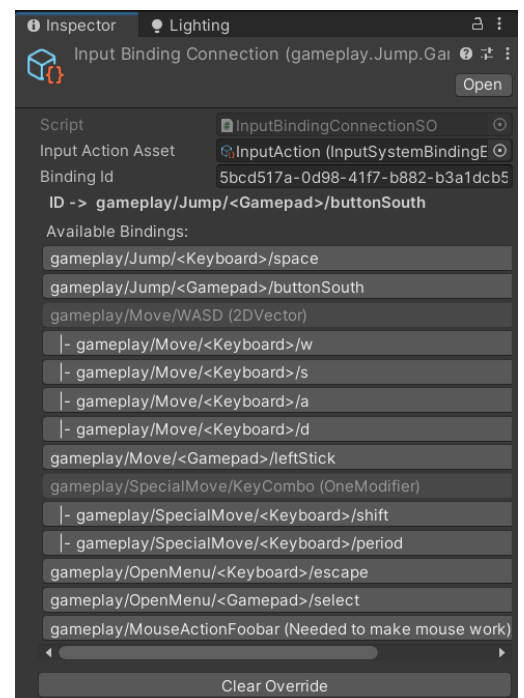
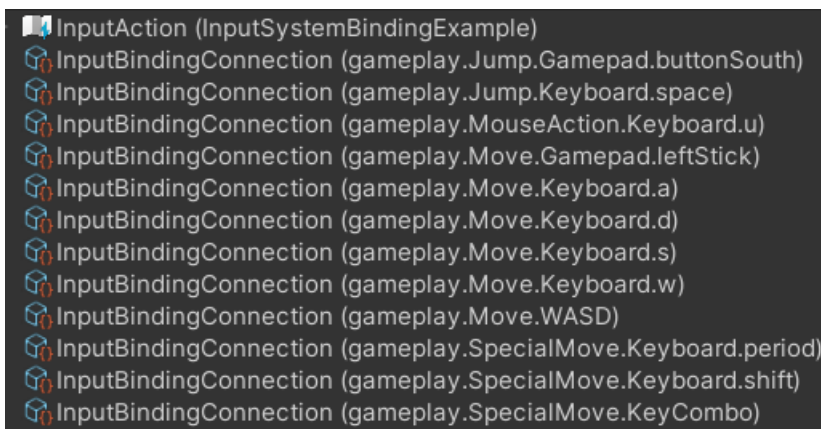
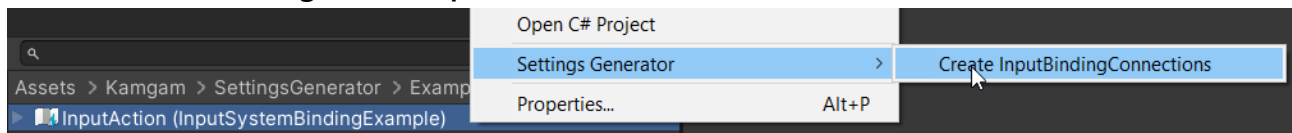
1) Generating the InputBinding Connections

Let's assume you have an [InputActions Asset](#) like this:



To generate the InputBinding Connections from it you can right-click on the asset and then choose **SettingsGenerator > Create InputBindingConnections**.

This will generate one InputBindingConnection for each Binding you have in the Input Actions Asset. If you add or remove Input Actions later then you should repeat the process and your Connections will be updated. **Don't delete your connections or you will have to reconnect them with the settings (see step 2).**

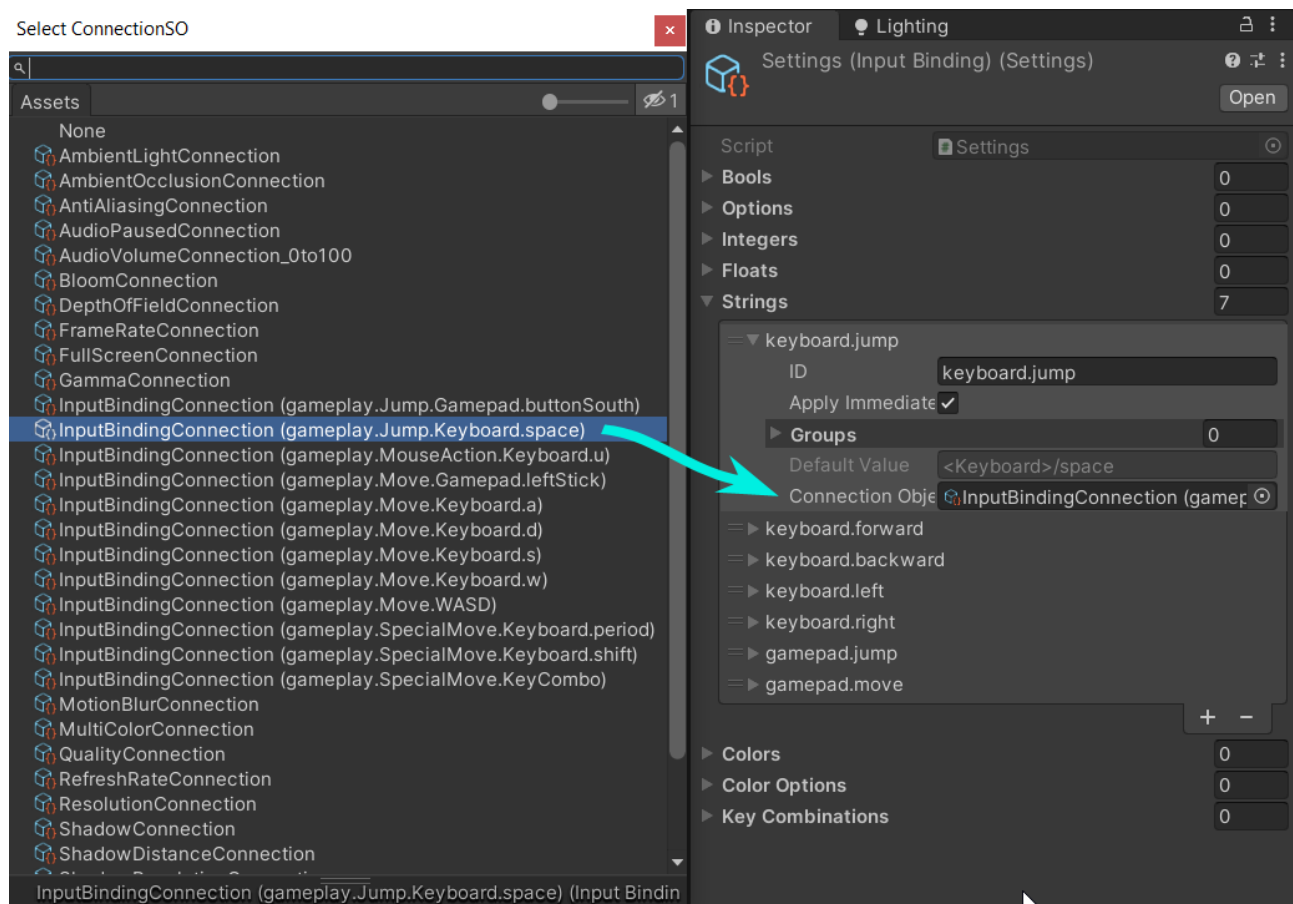


Info: Each Connection is linked to a Binding via two things.

A) the Actions Asset and B) the Binding GUID (see image on the right). Usually you can just leave the connections objects alone after creation. However if needed you can edit the linked GUID manually or via the „Available Bindings“ buttons. The bold „ID →“ line tells you which Binding the current „Binding Id“ belongs to.

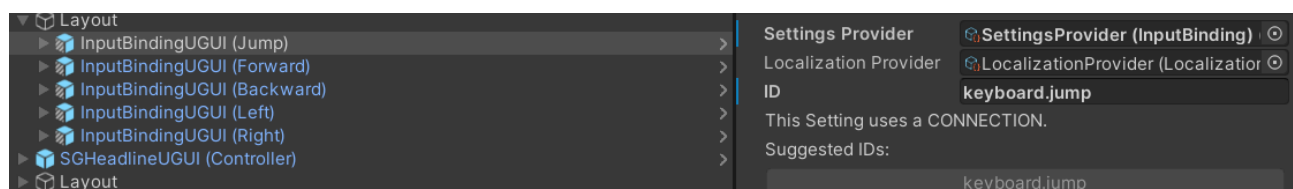
2) Link InputBinding Connections to Settings

You have linked the Input Actions Asset with connections but you still need to link these connections with your settings. For each Input you would like to control you will have to create a StringSetting and drag in the corresponding connection.



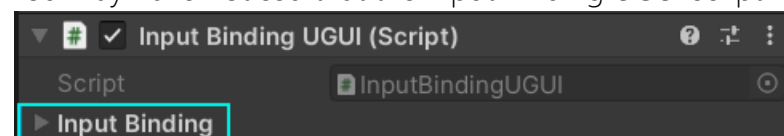
It's a bit cumbersome but you will only have to do this once (as long as you don't delete your connection objects).

Now the rest works just like it does for all the other settings. You drag in the UI and use the Resolvers ID field to link the UI to the Setting. Like this:



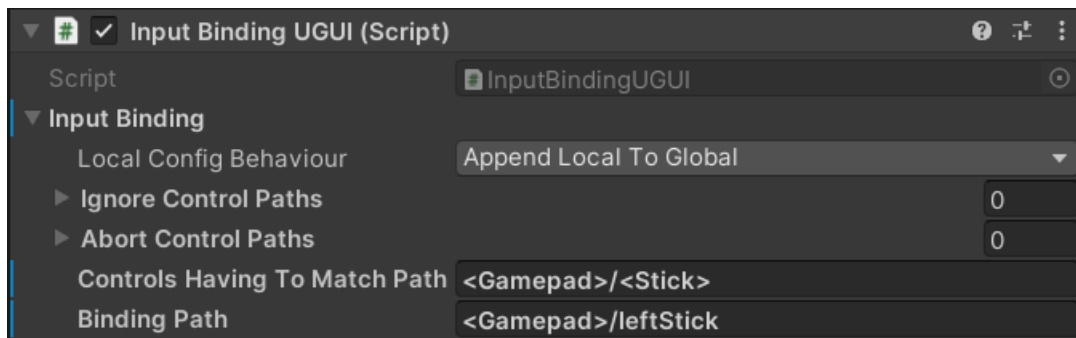
3) Configuring the Rebinding UI

You may have noticed that the Input Binding UGUI script has a special section „Input Binding“.

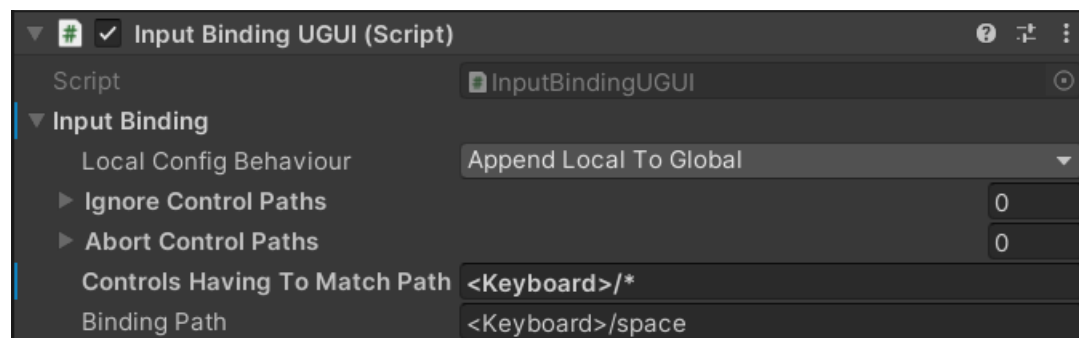


If you open it you will see some configuration options. These are used to LIMIT the inputs this UI will react to. This is useful for example to limit a MOVE action for a controller to only take Thumb Sticks and ignore buttons („<Gamepad>/<Stick>“).

This magic „<Gamepad...“ string is called an „Input Control Path“ and is documented in the [Unity Manual](#).



Another example of restricting a UI to only allow Keyboard input:



The [InputControlPaths](#) are used in the [Interactive Rebinding](#) process.

Ignore Control Paths:

These paths are ignored (see [WithControlsExcluding](#)). By default there already are some paths in this list (global list):

- "<Pointer>/position" // Don't bind to mouse position
- "<Pointer>/delta" // Don't bind to mouse movement deltas
- "<Pointer>/{PrimaryAction}" // Don't bind to controls such as leftButton and taps.
- "<Mouse>/clickCount" // Don't bind to mouse click count events

You can choose to either override or extend them via the „**Local Config Behaviour**“ dropdown.

Abort Control Paths:

These paths behave like the [WithControlsCancelingThrough](#) option. By default there already are some paths in this list (global list):

- "<Keyboard>/escape",
- "<Gamepad>/start"

You can choose to either override or extend them via the „**Local Config Behaviour**“ dropdown.

NOTICE:

There is a quite [nasty error in the 1.3. version of the InputSystem](#). If you can, then please upgrade to version 1.4.1. or higher. The Settings System contains a workaround for this error but it may not cover 100% of all cases.

Control Having To Match Path:

Here you can limit which controls the UI will react to. It works like the [WithControlsHavingToMatch](#) option.

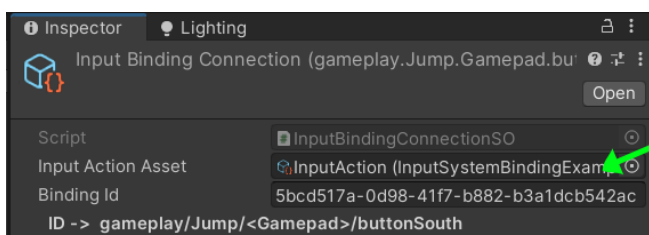
Binding Path:

At runtime this shows the currently bound path. It also is used as a fallback path to display if something fails.

Working with Auto-generated script code for Actions

If you are using an input actions asset then you can choose to generate a [c# actions class](#).

Now, the issue here is that if you use the c# class it will create a new INSTANCE of the actions at runtime and you NEED to tell the settings system about it. Otherwise the InputBindingConnections you use will point to the wrong actions instance.



Will point to the wrong input actions asset if the c# class is being used.

Here is an example of how to set the input actions asset:

```
public class Test : MonoBehaviour
{
    YouInputActionsClass inputControls;
    public SettingsProvider SettingsProvider;

    void Start()
    {
        if (inputControls == null)
        {
            inputControls = new YouInputActionsClass();
        }

        var settings = SettingsProvider.Settings;
        settings.SetInputActionAsset(inputControls.asset);
    }
}
```

Scripting API (Adding custom Settings)

There are three ways to connect a setting to your custom code:

- Fetch the value of a setting manually.
- Using the `GetSetConnection<T>`
- Making a custom `Connection` class

More on those in the sections below.

Getting a setting value can be done by pull (you ask for the value) or push (the value is pushed to a method).

Pulling a value from a setting:

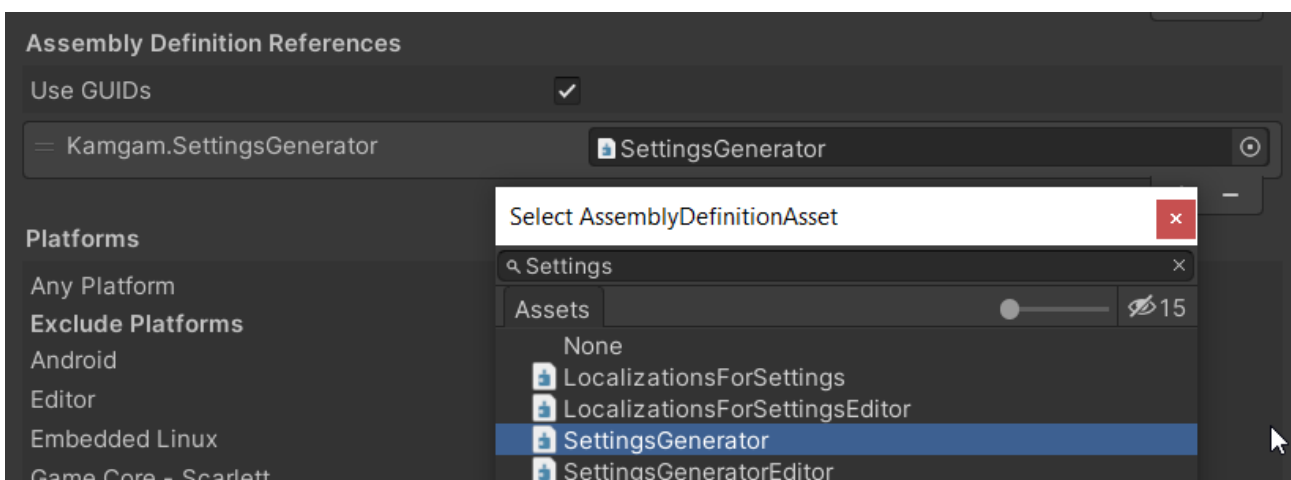
```
var settings = SettingsInitializer.Settings;  
SettingFloat volume = settings.GetFloat(id: "volume");  
Debug.Log( volume.GetFloatValue() );
```

Registering a callback to receive value changes:

```
Settings settings = SettingsInitializer.Settings;  
settings.GetFloat("volume").OnSettingChanged += (ISetting setting) => {  
    Debug.Log( setting.GetFloatValue() );  
};
```

How to get access to the API if you are using Assembly Definitions

You may have noticed that the `Kamgam.SettingsGenerator` namespace is not immediately available in your script if you are using assembly definitions. That's because the settings system uses Unity's [Assembly Definitions](#) system to reduce compile times. This means the code is wrapped in a separate assembly. To use it you will have to add an `asmdef` file to your code and reference the `SettingsGenerator` runtime assembly like this:



How pull the value from a setting

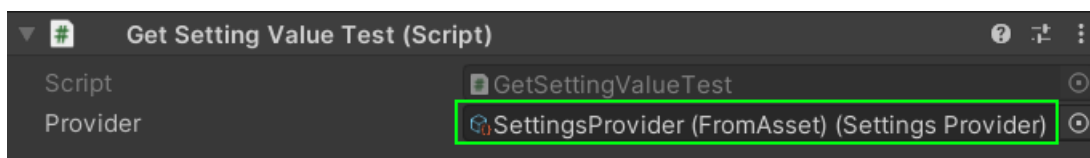
Your entry point to accessing the settings system is always a `SettingsProvider` Asset. The easiest way to get one is to make it a public property of a `MonoBehaviour`, like this:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Kamgam.SettingsGenerator;

public class GetSettingValueTest : MonoBehaviour
{
    public SettingsProvider Provider;
}
```

Notice the use of the „`Kamgam.SettingsGenerator`“ namespace.

Once you have that you need to drag in the `SettingsProvider` asset in the inspector (make sure you use the right one if you have multiple providers).



Now you can fetch any setting object via the `GetOrCreate...()` methods.

To get the actual value use the `GetValue()` method (or the `OnSettingChanged` event, more on that below).

Like this:

```
public class GetSettingValueTest : MonoBehaviour
{
    public SettingsProvider Provider;

    protected SettingFloat _musicVolumeSetting;

    public void Start()
    {
        // I'd recommend caching settings that are used often.
        _musicVolumeSetting = Provider.Settings.GetOrCreateFloat("audioMusicVolume");
    }

    public void Update()
    {
        float volume = _musicVolumeSetting.GetValue();
        Debug.Log($"Music Volume: {volume}");

        // HINT: You can also fetch values directly from the settings object:
        // Provider.Settings.GetValue<float>("audioMusicVolume");
    }
}
```

And that's it. If you want you can explore the API of the settings object. It has many features.

How to get notified of a value change in a setting.

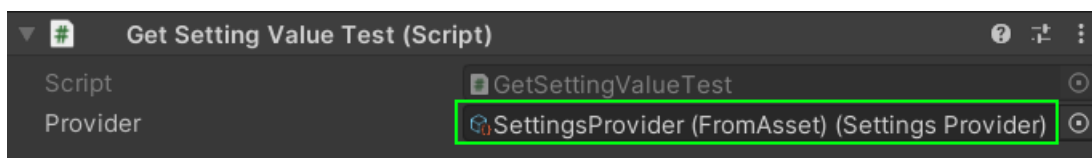
Your entry point to accessing the settings system is always a SettingsProvider Asset. The easiest way to get one is to make it a public property of a MonoBehaviour, like this:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Kamgam.SettingsGenerator;

public class GetSettingValueTest : MonoBehaviour
{
    public SettingsProvider Provider;
}
```

Notice the use of the „Kamgam.SettingsGenerator“ namespace.

Once you have that you need to drag in the SettingProvider asset in the inspector (make sure you use the right one if you have multiple providers).



Now you can fetch any setting object via the **GetOrCreate...()** methods.

To get notified of a change you can use the **OnSettingChanged** event.

Like this:

```
public class GetSettingValueTest : MonoBehaviour
{
    public SettingsProvider Provider;

    public void Start()
    {
        var musicVolumeSetting = Provider.Settings.GetOrCreateFloat("audioMusicVolume");
        musicVolumeSetting.OnValueChanged += onChanged;
    }

    private void onChanged(float value)
    {
        Debug.Log($"Value: {value}");

        // HINT: There is also a OnSettingFloatChanged event that you can use to access
        // the setting object if needed.
    }
}
```

And that's it. If you want you can explore the API of the settings object. It has many handy features.

Using the GetSetConnection<T>

There is a premade Connection called GetSetConnection<T>. You can use it to register two methods. One to get the value from the setting and one to send the value to the setting.

The code would look like this (from the code example in the asset):

```
/// A simple int for a slider from 0 to 100 (the range is defined in the UI).
/// It defines a percentage of how strong the health regeneration should be.
protected int _healthRegeneration;

protected void addHealthRegenerationPercentage(Settings settings)
{
    // This setting will react to changes in the setting and propagate those
    // to the local field "_healthRegeneration".
    //
    // To connect a setting (data) with some logic we use Connection objects.
    // These are very simple. They have a Get() and a Set(value) method.
    //
    // Get() means getting a value from the connection and saving it in the setting (pull).
    // Set(value) means sending a new value to the connection (push).
    // There are many specialized predefined Connections (like the "FrameRateConnection").
    //
    // For this example we use a simple generic GetSetConnection<T> connection.
    // This connection does nothing except forward Get() and Set(value) calls to
    // some other methods (getter and setter).

    var connection = new GetSetConnection < int > (
        getter: getHealthRegeneration, // executed if connection.Get() is called.
        setter: setHealthRegeneration // executed if connection.Set(value) is called.
    );

    // Now that we have our connection we need to hook it up with our setting.
    // In fact at first boot we also have to create our setting. Luckily there
    // is a handy GetOrCreateInt() method so we don't have to worry about that.
    var healthSetting = settings.GetOrCreateInt(

        // Each setting needs an ID.
        id: "healthRegeneration",

        // The default value is the fallback default value used if no connection is
        // set. In this case we are using a connection and the default value is pulled
        // initially from that connection. Therefore we actually don't need to specify it.
        // defaultValue: false

        // We want to use a connection to get/set the values.
        connection: connection
    );

    // If all you need is to listen for changes in a setting then you may not even need a
    // Connection object. There is a healthSetting.AddChangeListener() method which you
    // can use for that.
}
```

```

// This simply returns the current state of "_healthRegeneration".
// This getter is called at the very first use of the setting
// and the return value will be stored as the default value (used
// if you call setting.ResetToDefault()).
//
// It may also be called at any time by the settings system and
// should return the current state of the value in your game.
//
// If this value is changed from outside the settings system, then
// you need to call setting.PullFromConnection() to update the internal
// value of the setting.
//
// "Pull" in this context is meant as from the view point of the setting.
// I.e. "pull the value from the connection into the setting and update the UI".
// During this pull process connection.Get() is called which calls
// this getter.
//
// There is also a setting.PushToConnection() method which does the opposite.
// It pushes the value from the setting into the connection (connection.Set())
protected int getHealthRegeneration()
{
    // This is where you would add your game specific code.
    return _healthRegeneration;
}

// This simply sets the local field and logs the new value.
protected void setHealthRegeneration(int value)
{
    // This is where you would add your game specific code.
    _healthRegeneration = value;
    Debug.Log("Health regeneration has been set to: " + value);
}

```

Making a custom Connection class

The settings system reduces any setting into basic types (bool, Color, float, int, string, ...). To connect a setting to some custom code you have to implement an interface called `IConnection<T>`.

The interface is really simple:

```
public interface IConnection<TValue> : IConnection
{
    public delegate void OnChangedDelegate(TValue value);

    // This is where all the magic happens.
    public TValue Get();
    public void Set(TValue value);

    // Don't worry. There is a base class implementing these for you (see below).
    public void AddChangeListener(OnChangedDelegate listener);
    public void RemoveChangeListener(OnChangedDelegate listener);
}
```

You need to create a new Connection class implementing that interface. All the premade connections are made like this too.

Here is the VSync connection for example:

```
public class VSyncConnection : Connection<bool>
{
    public override bool Get()
    {
        return QualitySettings.vSyncCount != 0;
    }

    public override void Set(bool vSyncEnabled)
    {
        QualitySettings.vSyncCount = vSyncEnabled ? 1 : 0;
        NotifyListenersIfChanged(vSyncEnabled);
    }
}
```

You may have noticed that it does not implement the `IConnection` interface directly but instead extends a generic class `Connection<bool>` (where `bool` is the type of what the `Get()` method returns and what the `Set(value)` takes).

Using the generic base class spares you the work of implementing `AddChangeListener()` etc. . Just override `Get` and `Set(value)` and you have a fully functional connection.

Now this is a complete Connection and you can use it in code. BUT you can not yet use your new Connection in the ScriptableObject Assets (`Settings.asset`). To enable this you will also have

to implement a Scriptable Object for your Connection (and then instantiate it). We can use some of our base classes to spare us typing all the repetitive code.

Here is how the SO class for the VSyncConnection looks like:

```
namespace Kamgam.SettingsGenerator
{
    [CreateAssetMenu(fileName = "VSyncConnection", menuName =
"SettingsGenerator/Connection/VSyncConnection", order = 1)]
    public class VSyncConnectionSO : BoolConnectionSO
    {
        protected VSyncConnection _connection;

        public override IConnection<bool> GetConnection()
        {
            if(_connection == null)
                Create();

            return _connection;
        }

        public void Create()
        {
            _connection = new VSyncConnection();
        }
    }
}
```

Now you can create an SO Asset and use it for any setting matching the type (bool in this case).

If you are not a coder then maybe you are able to do it with the VisualScripting package and the GetSetConnection, though it's cumbersome compared to the code solution.

The IConnectionWithSettingsAccess interface

Add it to a connection if that connection needs to know what settings object it belongs too. If this is added to a connection then the settings system will notice and call `SetSettings(Settings settings)` after load but before `InitializeConnection()`.

An example of a connection using this is the `QualityConnection`.

Adding custom save & load methods

You can override the methods used for save, load and delete on the Settings.

To do this set the static: **Settings.CustomSaveMethod**, **Settings.CustomLoadMethod** and **Settings.CustomDeleteMethod** methods. If you change one of these then you should always change all three.

Under Assets/Kamgam/SettingsGenerator/Examples/SaveLoadCustomization you will find a file named „**SettingsInitializerSaveLoadToFile.cs**“. It contains a demo of how to override the save & load methods (json file).

NOTICE: It is important that you set these BEFORE the Settings are used for the very first time or else it will still use the default method.

That's why the class uses the `[DefaultExecutionOrder(-10)]` Attribute.

SettingsInitializerSaveLoadToFile.cs:

```
using System.IO;
using UnityEngine;

namespace Kamgam.SettingsGenerator.Examples
{
    // Make sure this runs before any normal Awake code.
    [DefaultExecutionOrder(-10)]
    public class SettingsInitializerSaveLoadToFile : MonoBehaviour
    {
        void Awake()
        {
            Settings.CustomLoadMethod = load;
            Settings.CustomSaveMethod = save;
            Settings.CustomDeleteMethod = delete;
        }

        // Let's save the json data into a file instead of Player Prefs.
        string getFilePath(string key)
        {
            return Application.dataPath + "/" + key + ".json";
        }

        void delete(string key, Settings settings)
        {
            var filePath = getFilePath(key);
            if (File.Exists(filePath))
            {
                File.Delete(filePath);
            }

            // Delete .meta file in editor
            #if UNITY_EDITOR
            if (File.Exists(filePath + ".meta"))
            {
                File.Delete(filePath + ".meta");
                UnityEditor.AssetDatabase.Refresh();
            }
            #endif
        }
    }
}
```



```

    }
#endif

}

void save(string key, Settings settings)
{
    var json = SettingsSerializer.ToJson(settings);
    if (!string.IsNullOrEmpty(json))
    {
        var filePath = getFilePath(key);
        // Write to tmp file first.
        File.WriteAllText(filePath + ".tmp", json, System.Text.Encoding.UTF8);

        // Check if tmp file now exists.
        // If you want to be extra sure you could do a validity check of .tmp.
        if (File.Exists(filePath + ".tmp"))
        {
            // Delet existing
            if (File.Exists(filePath))
            {
                File.Delete(filePath);
            }
            // Finally move the tmp file.
            File.Move(filePath + ".tmp", filePath);

            // Log path (for testing)
            Debug.Log("Saved to: " + filePath);
        }
    }
}

#if UNITY_EDITOR
    UnityEditor.AssetDatabase.Refresh();
#endif

}

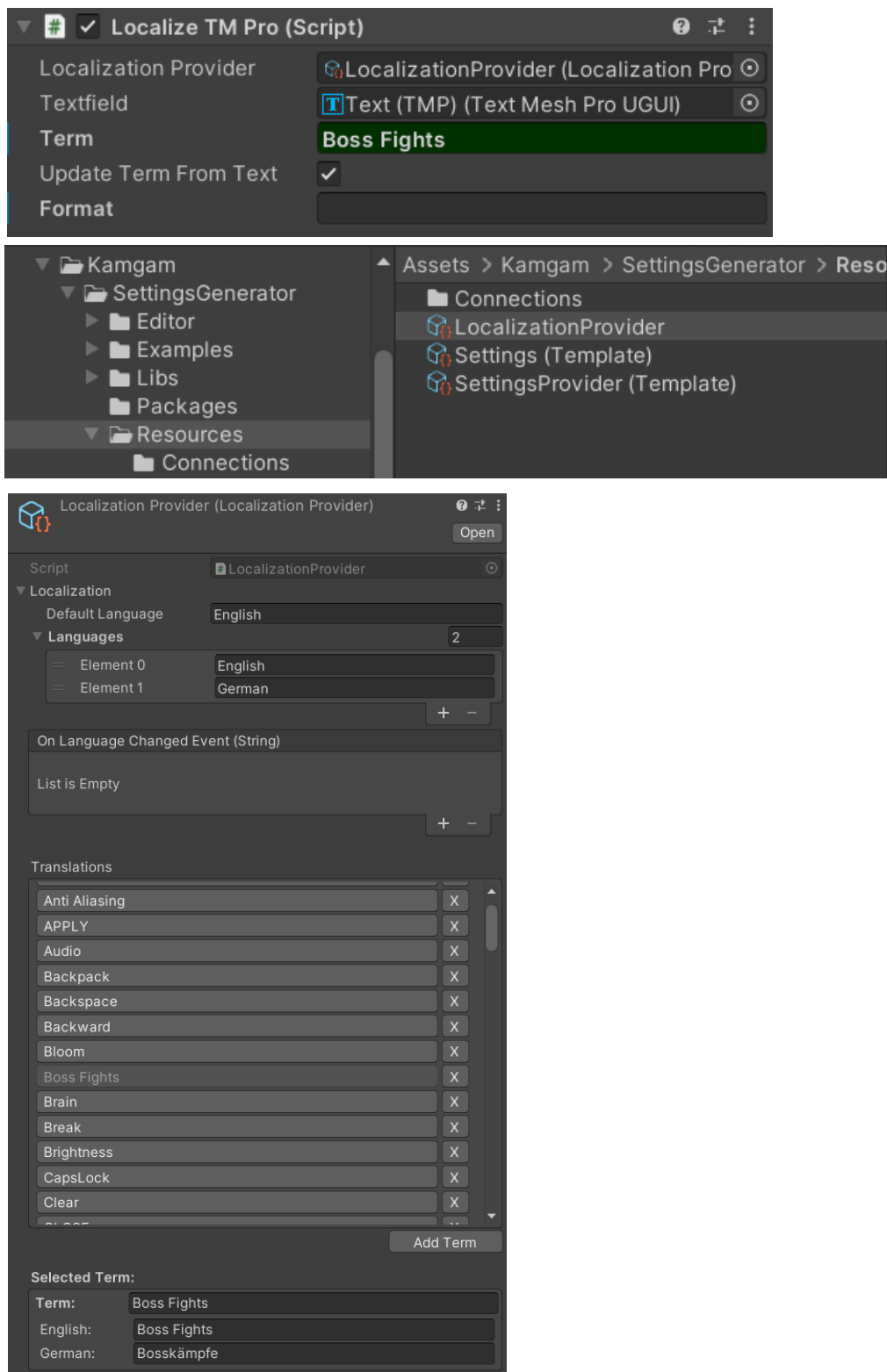
void load(string key, Settings settings)
{
    var filePath = getFilePath(key);
    if (!File.Exists(filePath))
        return;

    var json = File.ReadAllText(filePath, System.Text.Encoding.UTF8);
    if (!string.IsNullOrEmpty(json))
    {
        SettingsSerializer.FromJson(json, settings);
    }
}
}

```

Localization

Each textfield in the UGUI components comes attached with a Localize component. There you can enter a term that will then be looked up in the LocalizationProvider.



It is a fairly rudimentary localization system but for the settings it is sufficient. You can easily replace it with some dedicated localization system like [i2 Localization](#) (see „Localization.SetDynamicLocalizationCallback“)

Third Party Localization Support (example: I2 Localization)

To quickly integrate a third-party localization you can configure the included localization to forward all translation requests to another object.

The example below shows how this can be done for the popular [I2 Localization](#) asset. You can find a script template named „LocalizationInitializerTemplate“ in the „Libs/LocalizationsForSettings/Runtime“ folder.

```
using UnityEngine;
using Kamgam.LocalizationForSettings;

/// <summary>
/// Usually the localization does NOT need and initialization as it is
/// initialized on demand. However, if you want to bind another localization
/// solution to it then this may be a handy template.
/// </summary>
[DefaultExecutionOrder(-11)] // Not strictly necessary. Just to be sure it runs before settings.
public class I2LocalizationSettingsInitializer : MonoBehaviour
{
    /// Don't forget to hook this up with the right provider in the inspector.
    public LocalizationProvider Provider;

    public void Awake()
    {
        var localization = Provider.GetLocalization();

        // Tell the settings localization to ask I2L for translations.
        localization.SetDynamicLocalizationCallback(dynamicLocalization);

        // Hook into I2 Localization and listen for language changes.
        I2.Loc.LocalizationManager.OnLocalizeEvent += onI2LLocalizeEvent;
    }

    protected void onI2LLocalizeEvent()
    {
        // Trigger language change event to force a refresh on all translations.
        var localization = Provider.GetLocalization();
        localization.TriggerLanguageChangeEvent();
    }

    protected string dynamicLocalization(string term, string language)
    {
        string translation;

        if (!I2.Loc.LocalizationManager.TryGetTranslation(term, out translation))
        {
            var localization = Provider.GetLocalization();
            // Get translation without dynamic (to avoid infinite loop).
            translation = localization.Get(term, ignoreDynamic: true);
        }

        return translation;
    }
}
```

Localizing Key Control Paths

To localize the text that is used for the key controls you can use the control paths as keys for the localization system. The system automatically searches in the localization before it tries to auto convert the control paths to readable text.

The screenshot shows the Unity Inspector window with the Localization Provider (Localization Provider) selected. The Inspector displays the following settings:

- Script: LocalizationProvider
- Localization
 - Default Language: English
 - Auto Detect Language: ☒
 - Languages: 2
 - Element 0: English
 - Element 1: German
- On Language Changed Event (String): List is Empty
- Current Language: German
- Translations
 - Shout: X
 - Sound: X
 - Space: X
 - Tab: X
 - Take: X
 - Texture Resolution: X
 - Ultra: X
 - UpArrow: X
 - Very High: X
 - Very Low: X
 - Vignette: X
 - V-Sync: X
 - Window: X
 - <Keyboard>/space: X

Below the Inspector, the Project window shows the Assets folder structure. The LocalizationProvider script is highlighted in the Project window. A green arrow points to the LocalizationProvider script in the Project window, with the text "Find the localization provider." below it.

Below the Inspector, the Selected Term section shows the following translations for the term "<Keyboard>/space":

Term	English	German
<Keyboard>/space	<style=AllKeys/space>	<style=AllKeys/space>

Green arrows point to the English and German translation fields, with the text "Add a translation for each control path." above them and "Translate them with whatever text you need." below them.

Visual Scripting (formerly BOLT)

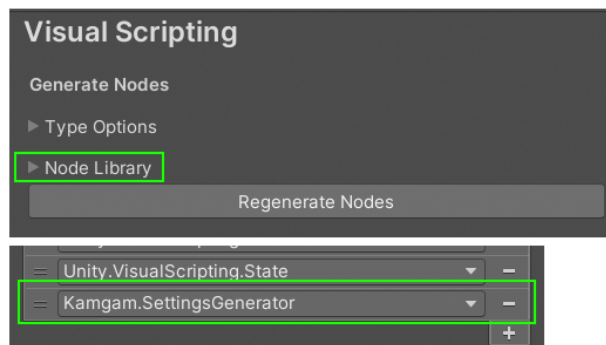
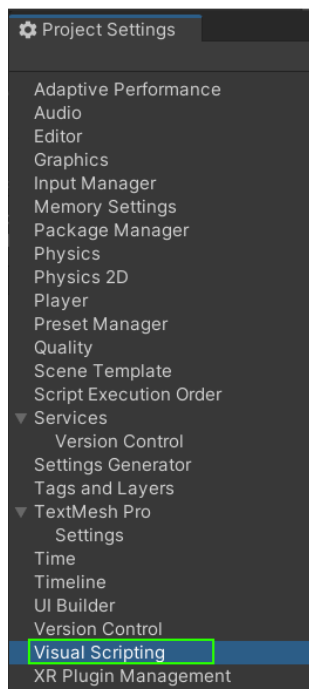
Visual Scripting requires a manual setup (let BOLT know the settings exist). Don't worry this will take only a minute.

Here are the steps:

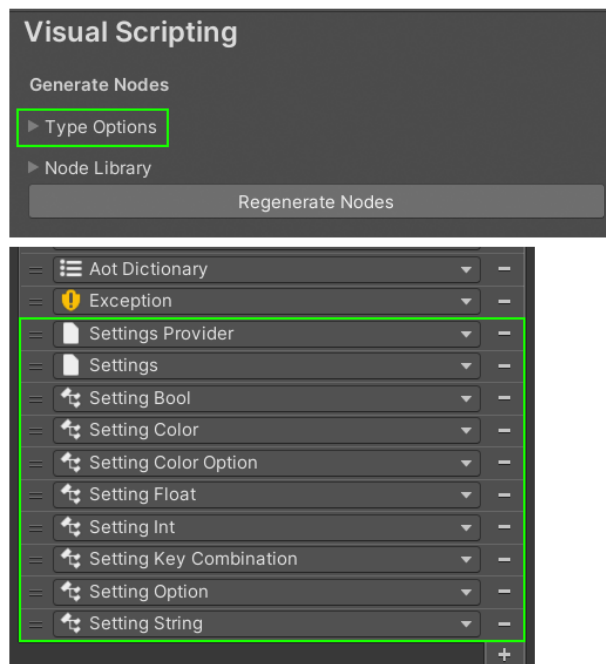
Go to **Edit > Project Settings > Visual Scripting**:

- 1) Add **Kamgam.SettingsGenerator** to the "Node Library"
- 2) Add the Settings Types (**SettingsProvider**, **Settings**, all the **Setting..**) to the "Type Options"
- 3) Hit "**Regenerate Nodes**" to apply the changes.

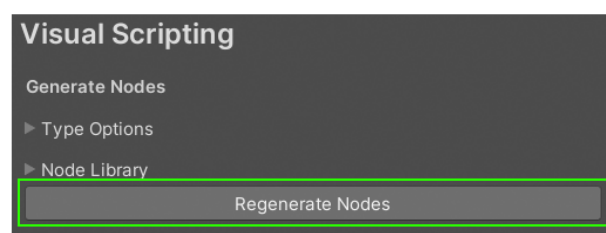
Step 1



Step 2

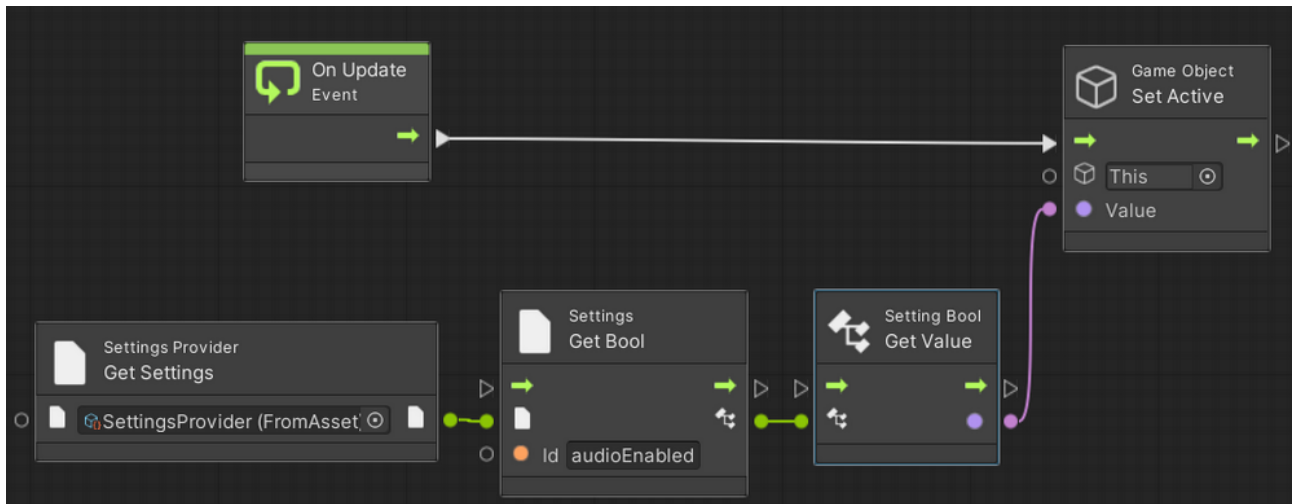


Step 3



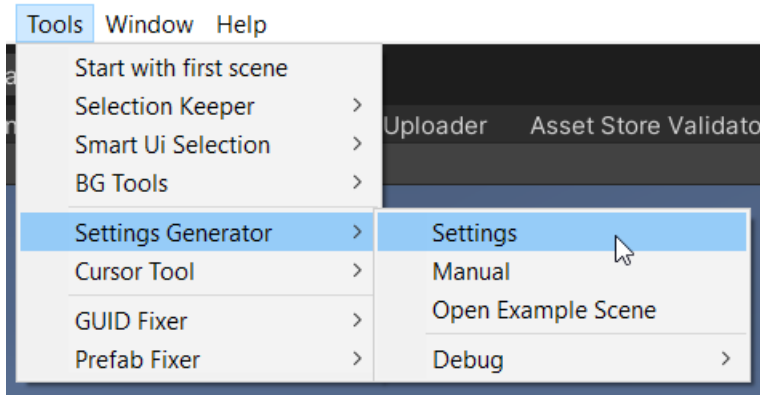
You can then retrieve any setting via its ID.

It looks like this (a simple example which disables the object if audio is disabled):

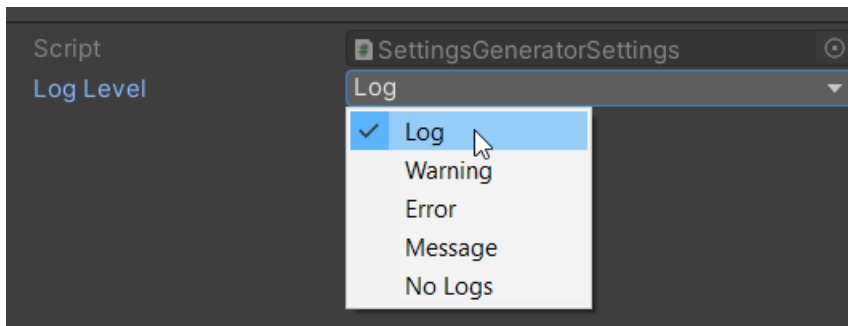


Configuration & Debugging

If a setting does not do what you want despite it being set up correctly then you can enable verbose logging to see if that gives you a clue.



Setting the LogLevel to „Log” will show you all the messages.



Input Validation

Sometimes (especially with text) you may wish to validate the input. Here is a guide on how to achieve this for a string setting. For convenience we will use the „playerName“ settings from the demo scene.

There are two options for how you could achieve this (via a custom OnChange event script or a custom connection object).

Validation via OnChange events

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Kamgam.SettingsGenerator;
using System;

namespace YourGame
{
    public class StringWithValidationTest : MonoBehaviour
    {
        public SettingsProvider SettingsProvider;
        public Settings Settings => SettingsProvider.Settings;

        public void Start()
        {
            Settings.GetString("playerName").OnStringSettingChanged += onSettingChanged;
        }

        private void onSettingChanged(SettingString setting)
        {
            // Get the value and validate/change it.
            string rawValue = setting.GetValue();
            string sanitizedValue = rawValue.Replace("aaa", "");

            // Write the changed value back to the setting.
            // propagateChange = false is important or else you would get endless loops.
            setting.SetValue(sanitizedValue, propagateChange: false);

            // Propagate the change to UI (update the text in the textfield).
            Settings.RefreshRegisteredResolvers(setting);
        }
    }
}
```

This script will replace all occurrences of „aaa“ in a text. It does it by updating the setting value in the on change event.

Validation via Connection objects

One way of hooking up some validation code to a setting is making a connection object. Below you will find the code for a very simple text validation. All it does is store the text in the „Value“ property of the connection and it automatically removes all instances of tripple-a sequences (i.e.: „testaaa“ will become „test“ because the „aaa“ will be removed).

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Kamgam.SettingsGenerator;

namespace YourGame
{
    public class StringWithValidationConnection :
        Connection<string>,
        IConnectionWithSettingsAccess
    {
        public Settings Settings;

        // This is where the value is stored at runtime.
        // In your game this probably is some external
        // service or your save-game or something.
        [System.NonSerialized]
        public string Value;

        public override string Get() => Value;
        public Settings GetSettings() => Settings;

        public void SetSettings(Settings settings)
        {
            Settings = settings;
        }

        public override void Set(string value)
        {
            if (string.IsNullOrEmpty(value))
                return;

            // The actual validation. Do whatever you need here.
            Value = value.Replace("aaa", "");

            // Refresh all string values that are using this connection.
            Settings.PullFromConnection(this, propagateChange: true);

            // Force and update on all UI elements that are using this connection.
            Settings.RefreshRegisteredResolversWithConnection(this);
        }
    }
}
```

Notice that the actual validation code is trivial: `Value = value.Replace("aaa", "");`

To hook up the connection we also want a connection asset. This is not strictly necessary since we could do this via code too but it is very convenient.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Kamgam.SettingsGenerator;

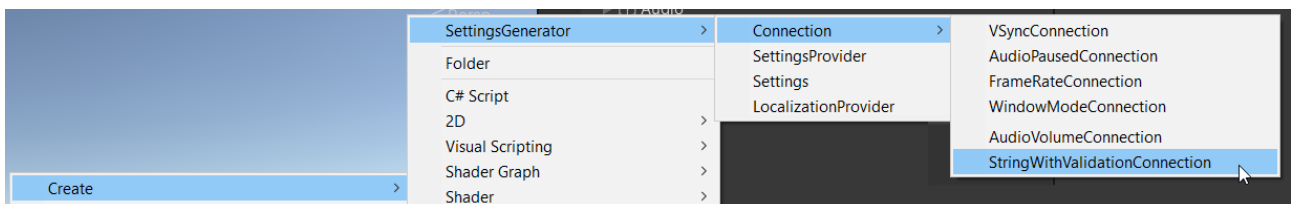
namespace YourGame
{
    [CreateAssetMenu(fileName = "StringWithValidationConnection", menuName =
"SettingsGenerator/Connection/StringWithValidationConnection", order = 4)]
    public class StringWithValidationConnectionSO : StringConnectionSO
    {
        [System.NonSerialized]
        protected StringWithValidationConnection _connection;

        public override IConnection<string> GetConnection()
        {
            _connection = new StringWithValidationConnection();
            return _connection;
        }

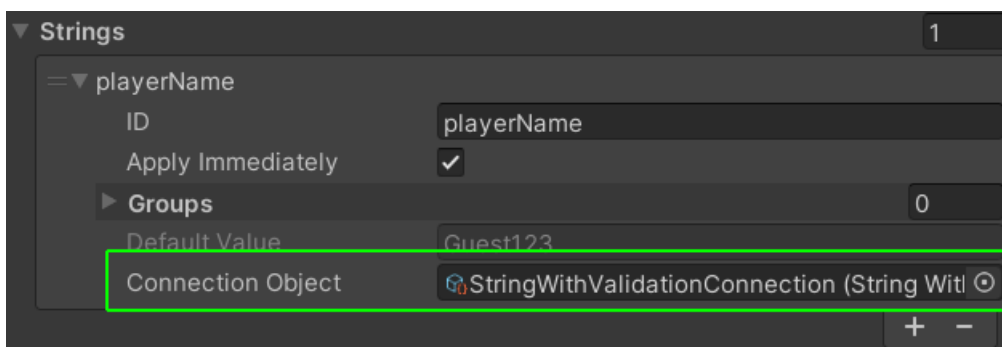
        public override void DestroyConnection()
        {
            if (_connection != null)
                _connection.Destroy();

            _connection = null;
        }
    }
}
```

Now with all the code set up we can create the connection object ..



.. and assign it to the playerName setting.



NOTICE 1: Strictly speaking this is not a validation before assign. It's more a validation after the value has changed. If you are using an OnSettingChanged event to listen for changes to this setting you will first receive the unvalidated raw value and then immediately afterwards you will get the validate value.

NOTICE 2: If you want to use this connection for multiple input fields then you will have to generate a new SO asset for each setting. Otherwise all the settings would write into the same single „Value“ field of the connection object and overwrite each other.

And that's it.

Upgrading from older versions

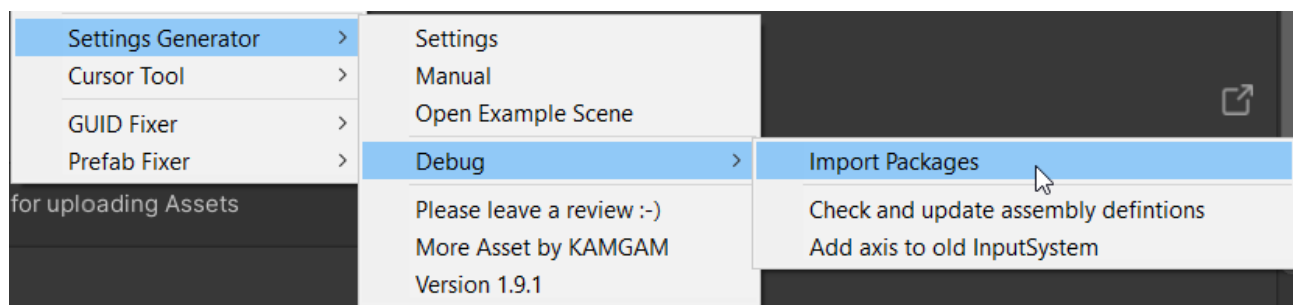
This asset follows the [Semantic Versioning](#) scheme. This means is the new version is no major version upgrade then there should be no breaking changes in the code.

Actions to perform after updating to fix example scenes

Some Unity versions require special assets (like URP and HDRP for rendering or the new input system). Upon first import these are detected and installed automatically. However after an UPDATE of the asset this is may not be done automatically.

You will have to trigger the import manually by calling:

Tools > Settings Generator > Debug > Import Packages



If there still are errors or things are not wokring (like post pro effect) then try to restart Unity.

Please contact support under office[a]kamgam.com if there still are errors afterwards.

Third Party Integrations

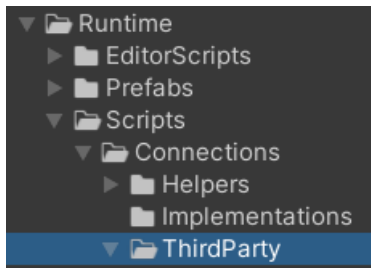
Over time some kind users (an other asset devs) have contacted me with recommendations for custom integrations. Here is a list of custom connections and integrations (with some rudimentary tutorials)

NOTICE #1: Please notice that I do not have any influence on these third party assets.

NOTICE #2: These assets may not support all render pipelines or Unity versions that Unified Settings do support. Please check the corresponding asset page for more information.

I'll try to keep my implementations up-to-date, yet realistically I only update them if someone pokes me (tells me something is broken) and if I have time to do it. Continuous testing all of these assets would be too much work for me. I also have no influence on the quality of these assets. Some may be abandonned or broken in the future. I reserve the right to drop support for any third party asset at any time. Please contact the creator of the asset (not me) for support. - Thank you for understanding.

The code of these thirparty implementations can be found here (usually in the form of a custom Connection class):



Rewired Integration

I have thought about this for a while now since it keeps getting asked. I have arrived at the conclusions that I would recommend to use the Rewired [Control Mapper](#) for your Input Rebinding UI if you are using Rewired.

For the full discussion please go to this forum thread:

<https://forum.unity.com/threads/unified-settings-game-options.1382952/#post-9028486>

Rewired is an explicit Input System it already covers all your needs in terms of input binding. It would say it is more advanced than the settings system input. It features multi user bindings, input calibration for thumb sticks,

You can mix the Settings System and Rewired.

Use Rewired for the Input UI and the settings system for the rest.

Adding a wrapper around Rewired would only add some new complexity and it would always be behind in features compared to the original.

If Rewired did not have it's own save and UI system then it would make sense to add one in the settings system. But since Rewired already has these I don't think it is a worthwhile endeavour to try to replace it.

You can install the Rewired Control Mapper via:

Window > Rewired > Extras > Control Mapper > Install

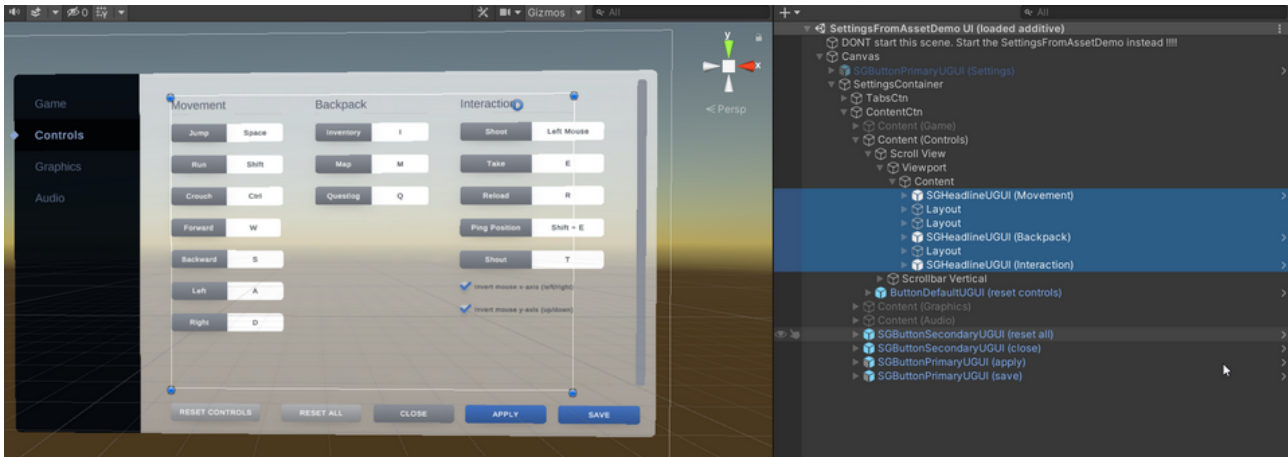
If you just rip out the input controls from the Settings System UI and add in the Rewired Control Mapper (ControlMapper Prefab) you should be ready to go (in case you have Rewired setup up already).

I know this may not be the answer you wanted but I really think adding yet another layer of abstraction around Rewired is not the way to go. If you are using Rewired, best use it all the way.

If you are not sure how to add Rewired to the existing settings UI then there is a tutorial in the next section.

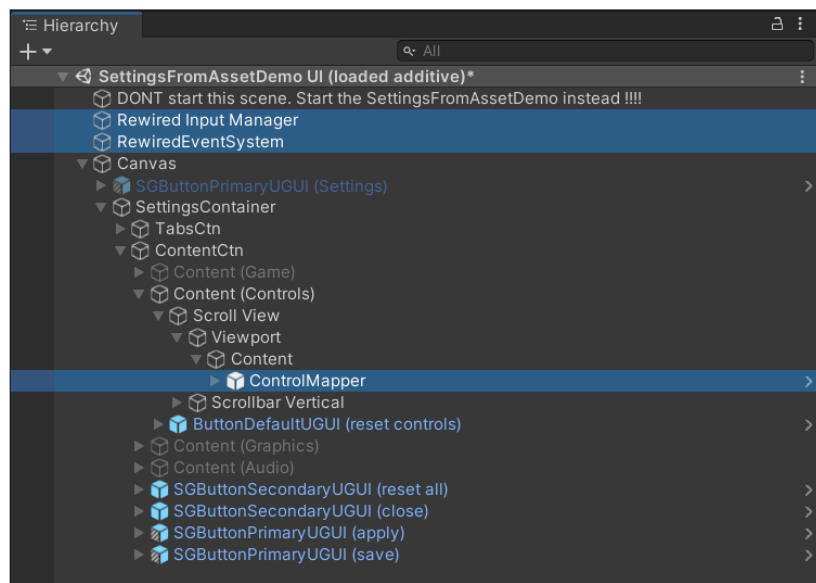
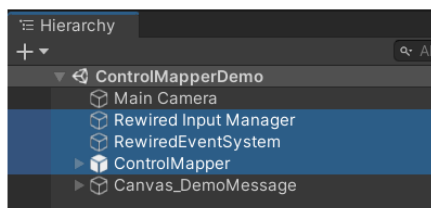
Rewired Integration Tutorial

- 1) Install the Mapper via Window > Rewired > Extras > Control Mapper > Install
- 2) Remove the existing inputs. Select the in the UI and delete them:



You may also want to delete the reset controls button (or your change it to access the rewired ControlMapper).

- 3) Copy the rewired parts from the ControlMapper demo. You probably already have a "Rewired Input Manager" and a "RewiredEventSystem". If you do then use them instead. The important part of the UI is the ControlMapper Prefab.



4) Now, sadly, we have to UNPACK the prefab. We have to do that because the Prefab root uses a Transform instead of a RectTransform, yet we need it to have a rect transform to integrate it into our canvas.

4.1) Right-Click the prefabs and choose Prefab > Unpack

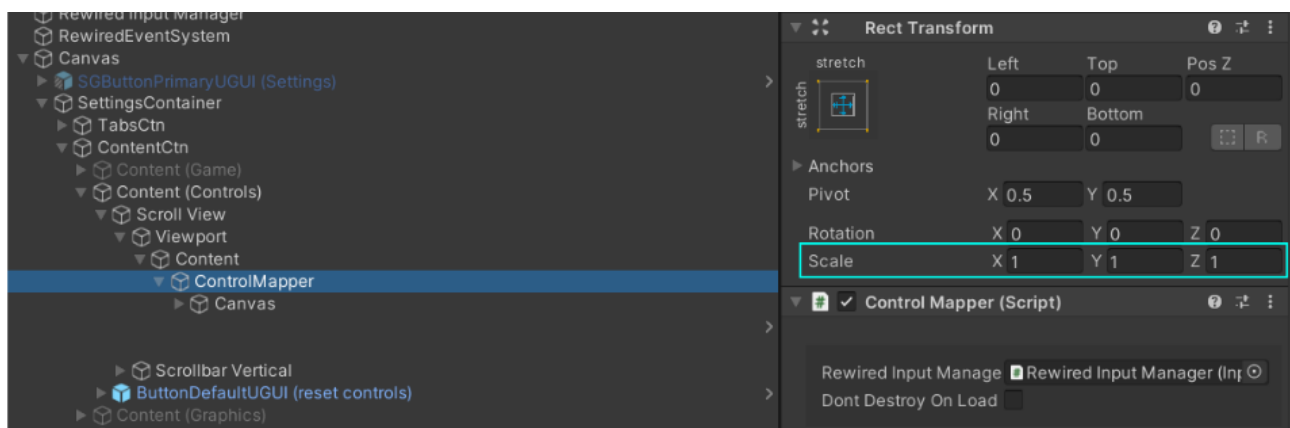
4.2) Right on top the the Control Mapper create a new empty object and name it "ControlMapper".

4.3) Copy the control mapper component from the unpacked ControlMapper onto the new object.

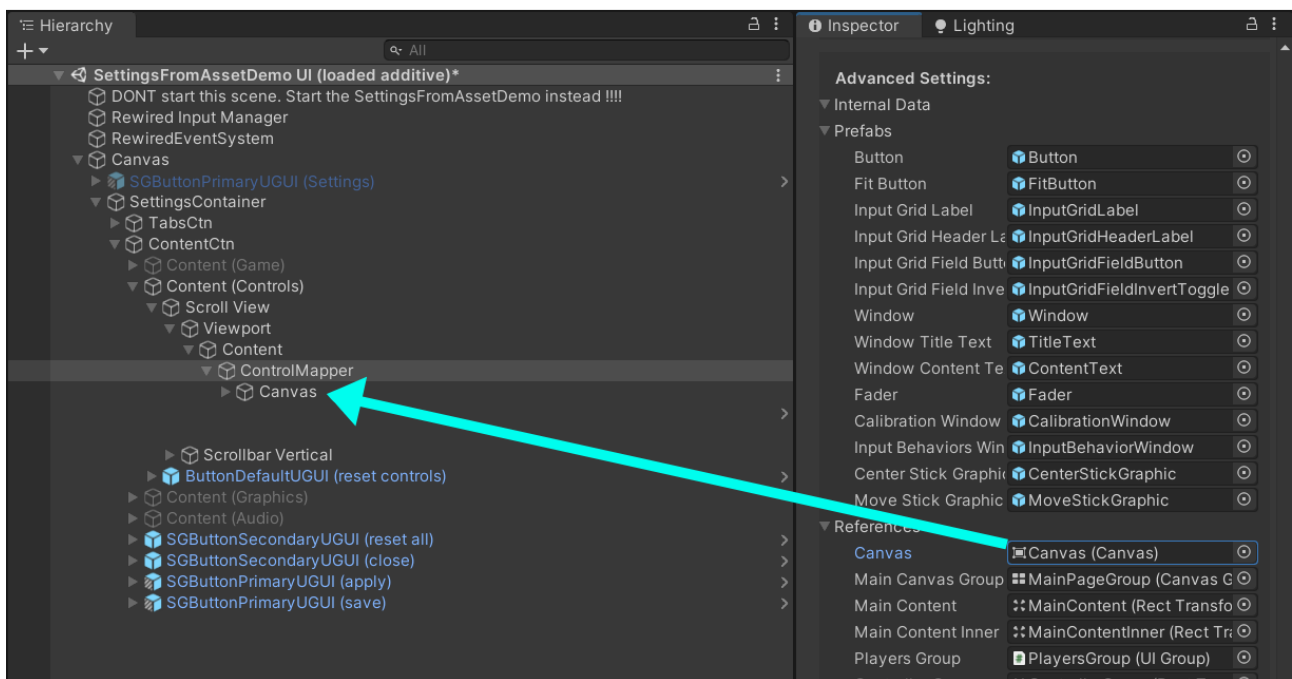
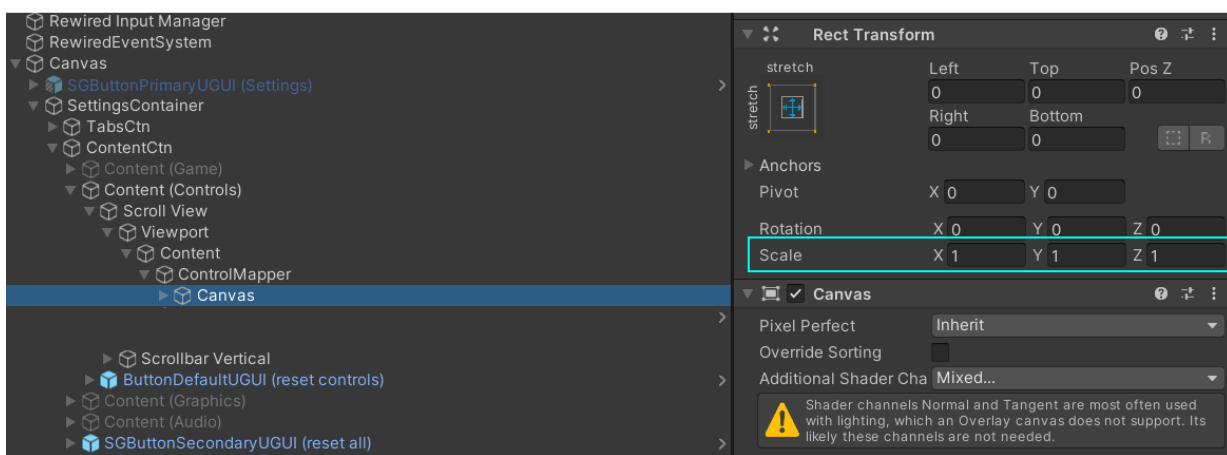
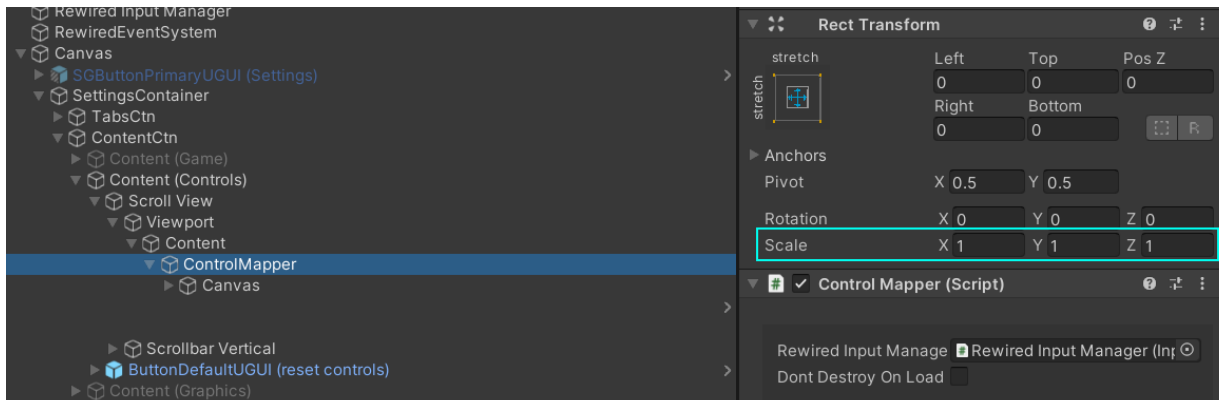
4.4) Move (don't copy!) the whole Canvas from the unpacked ControlMapper into the new object.

4.5) Delete the old ControlMapper Prefab.

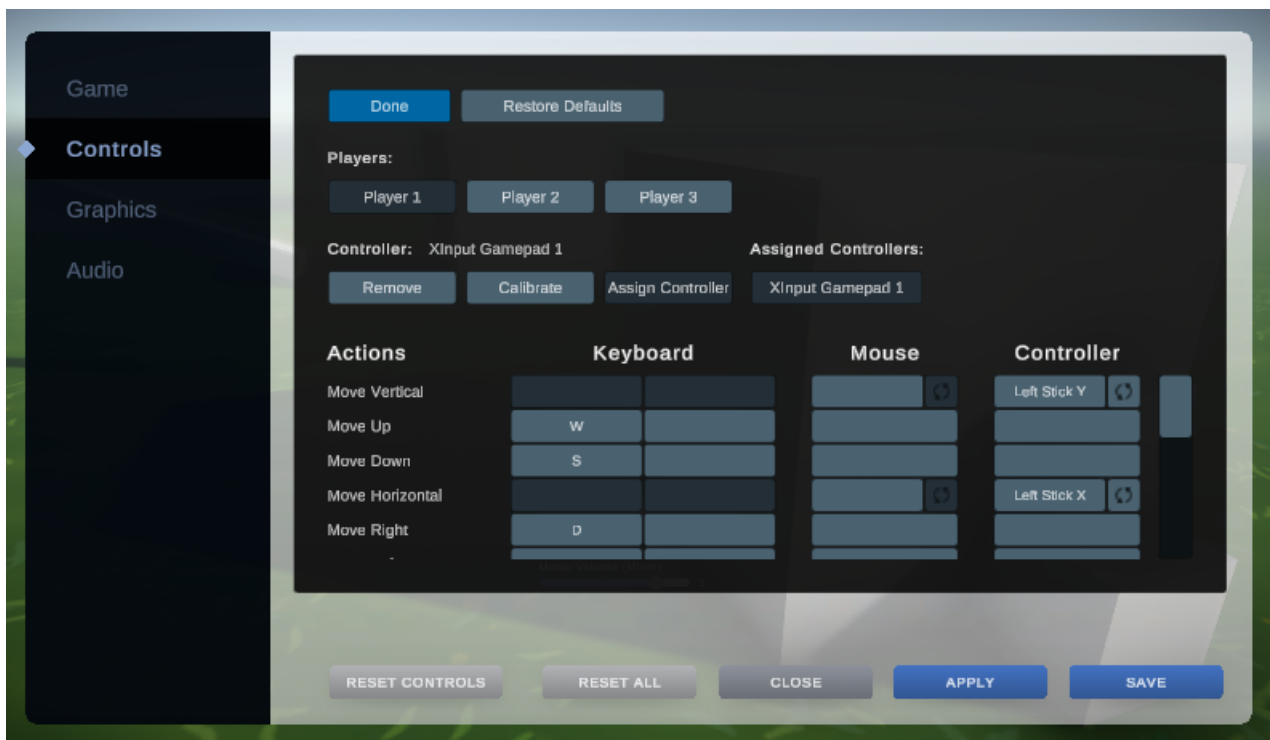
Now you should have something looking like this:.



5) Make sure the canvases all have proper rect transform sizes and scales (they tend to reset to 0/0/0 when copied or moved). Also make sure the internal data of the new ControlMapper (scroll way down) are linked to the right canvas elements.



In the end, if you hit play then your UI should look like this:



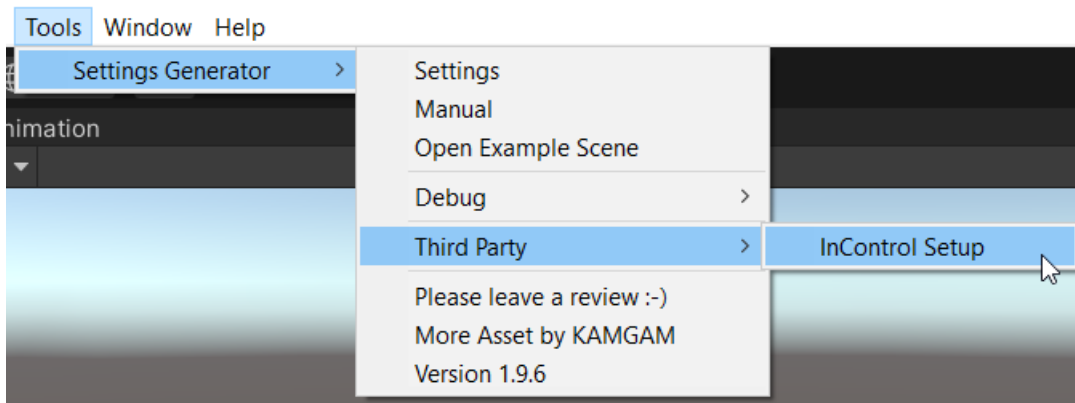
Here are the docs on how to [design \(theme\) the Rewired UI](#). The creator of Rewired opted to use the Unity UI theming system instead of a prefabs based approach like the settings system uses.

And that's it.

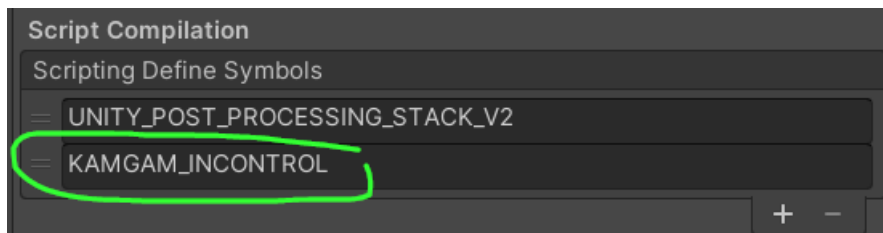
InControl Integration

If you are using the new input system then I recommend skipping [InControl](#). This section here is for those who are using the old Input System and want to use the KeyCombination settings (UniversalKeyCode) with InControl.

First step would be to let the settings system know InControl is used:



This will add a define to the symbols:



Once that define is set you will be able to use the **InControlUtils** class to convert between InControl controls and UniversalKeyCodes.

I recommend using „InControlUtils.UniversalKeyCodeToControlFunc(UniversalKeyCode keyCode)” since that will return a function that always returns the correct InControl control.

```
var speedSetting = Settings.GetKeyCombination("controller.speed");  
var speed = InControlUtils.UniversalKeyCodeToControlFunc(speedSetting .GetValue().Key);  
bool isSpeedPressed = speed().IsPressed;
```

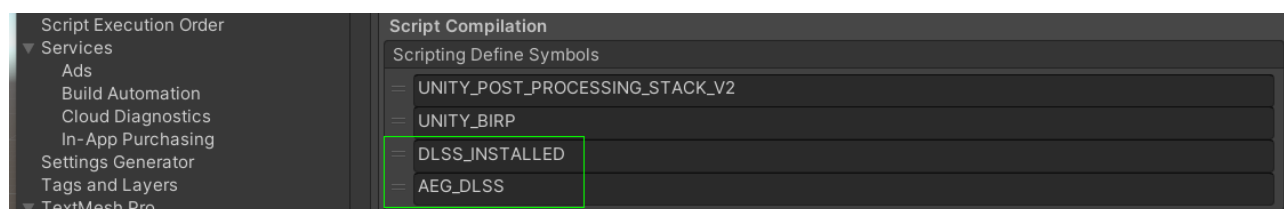
DLSS Connection (by Altergo Games)

The „AltergoDLSSConnection“ add support for the [DLSS Anti Aliasing Asset](#) by Altergo Games.

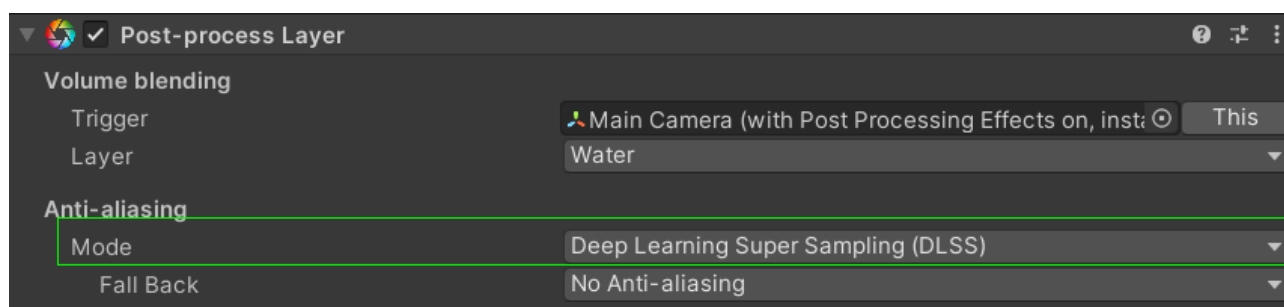
If you encounter any DLSS related issues, you can contact Altergo Games by emailing to info@thenakeddev.com, joining their [discord](#) in the “Unity Tools” channel or on the [Unity Forum](#).

To use it you will have to install the Altergo DLSS package and the NVIDIA package (instructions are in the DLSS manual that ship with the package). Please notice that the NVIDIA package will not install automatically. You will see the install link once you add DLSS to your post-processing layer on your camera.

To verify you have DLSS properly installed and set up please check the script defines. If DLSS is installed you should see two defines (DLSS_INSTALLED and AEG_DLSS).

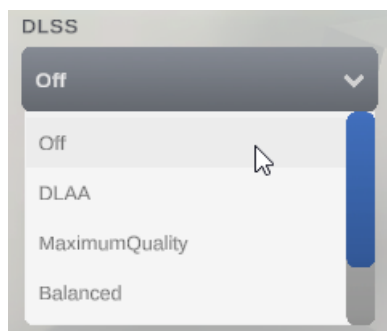


Here is how you enable DLSS on your post processing layer:



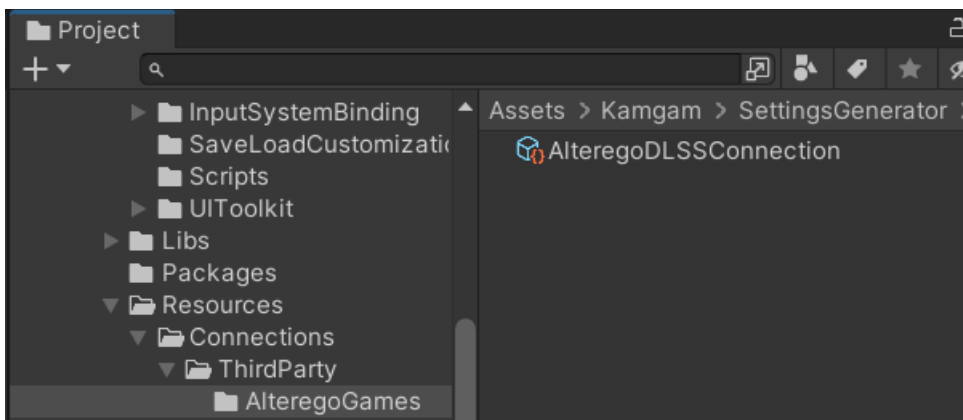
NOTICE: At the moment you can use either the normal AntiAliasing or DLSS. If DLSS is enabled on the camera the regular AntiAliasing options will do nothing (DLSS has taken over).

Here is how the DLSS options look at the moment (notice the „off“ option for turning DLSS off):

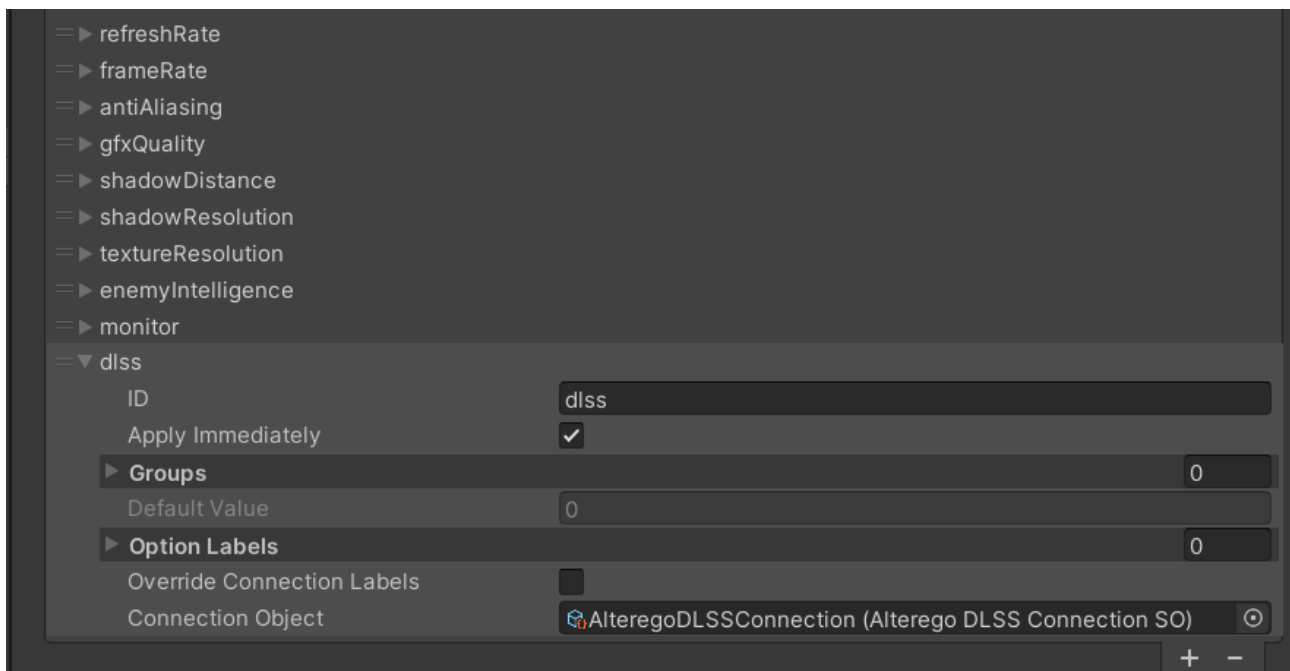


Notice the option labels are not localized. I'll leave localization to you. To localize them simply add the terms to the LocalizationProvider like any other setting localization.

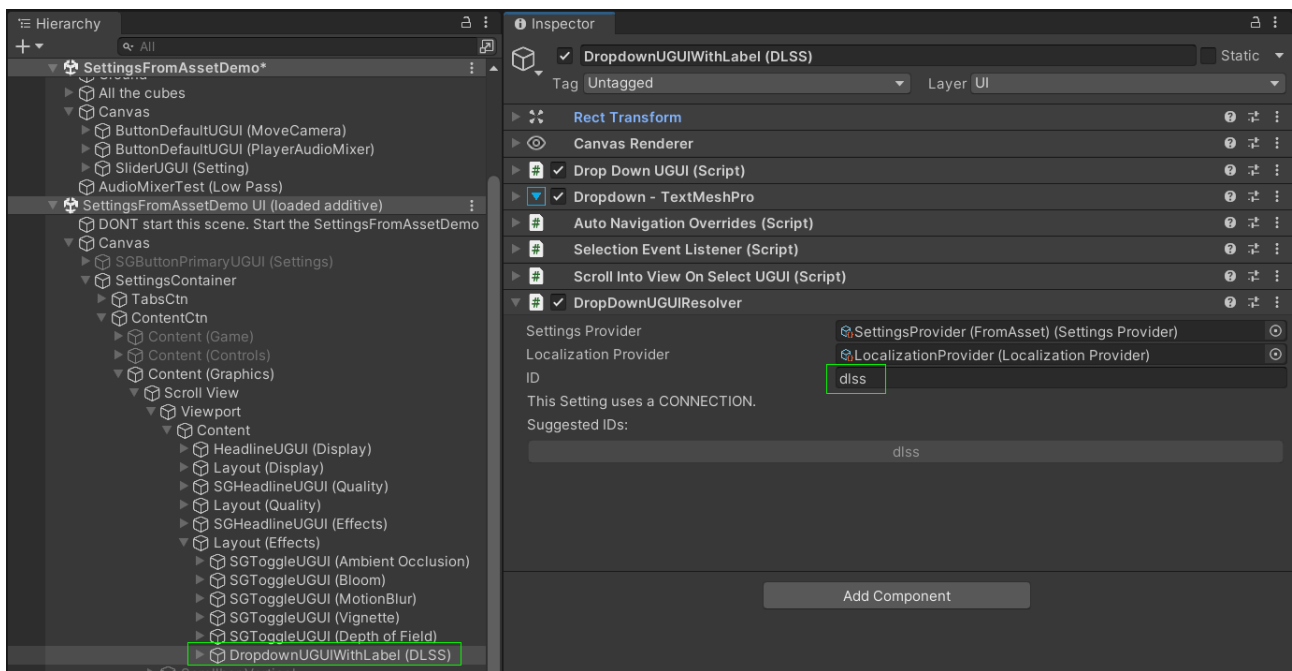
There is a premade „AlteregeDLSSConnection“ object which you can use in the settings.



Simply drop it in as the Connection Object:



In the UI the DLSS setting works just like any other options setting. Notice that the DLSS dropdown is not part of the default demo. You will have to add a new options dropdown on your own.



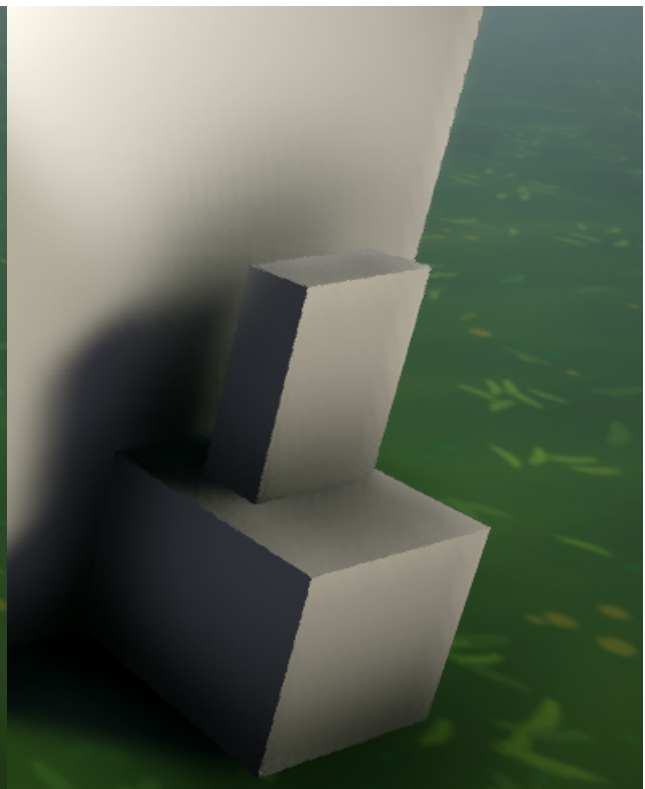
And that's it. Here is the result after I added DLSS to the demo scene (that's some nice anti aliasing):

Max Quality



vs

Max Performance

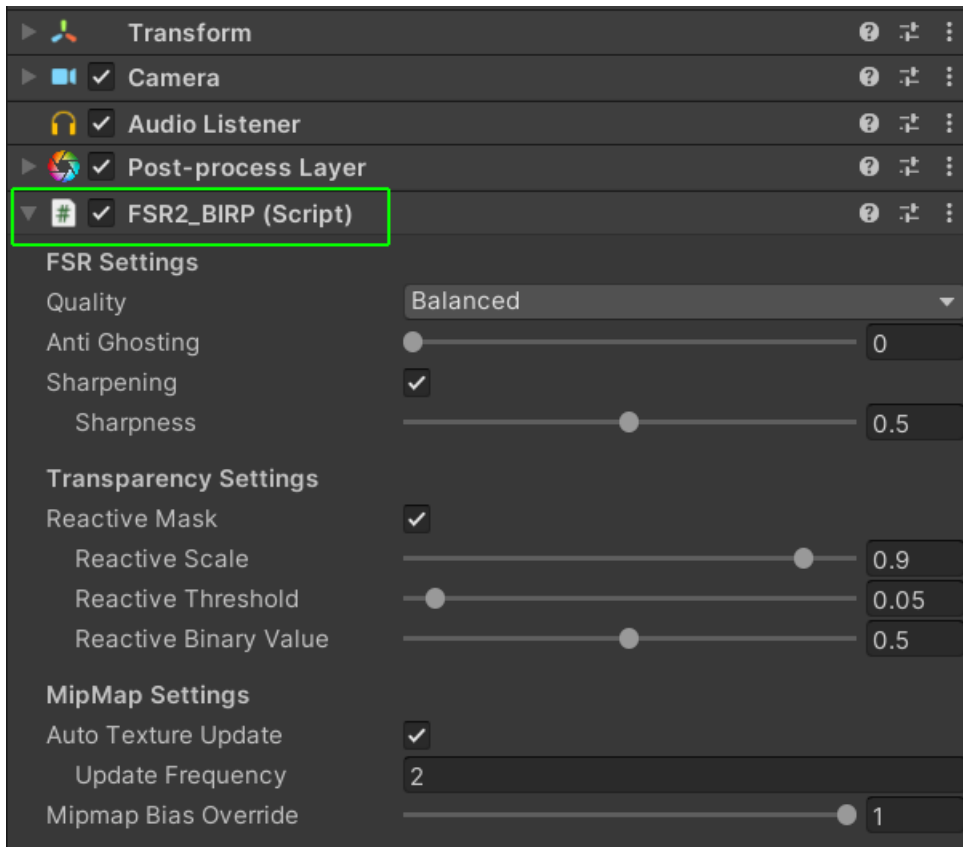


FSR 2 Connection (by Altergo Games)

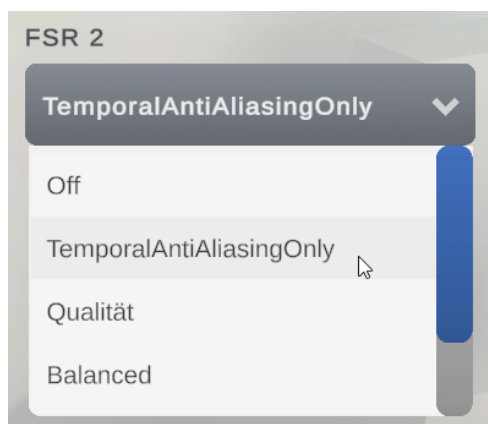
The „AltergoFSR2Connection“ add support for the [FSR2 Asset](#) by Altergo Games.

If you encounter any DLSS related issues, you can contact Altergo Games by emailing to info@thenakeddev.com, joining their [discord](#) in the “Unity Tools” channel or on the [Unity Forum](#).

After installation you will have to add the FSR2 component to your camera (more details on that are in the FSR2 manual):

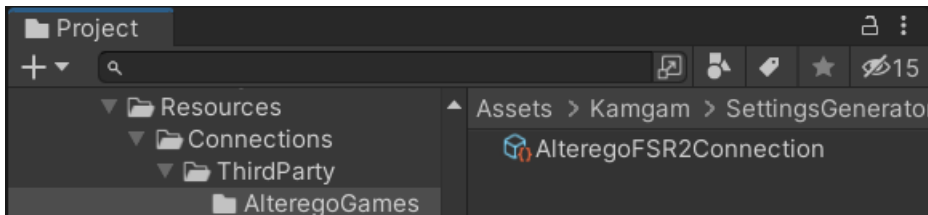


Here is how the FSR 2 options look:

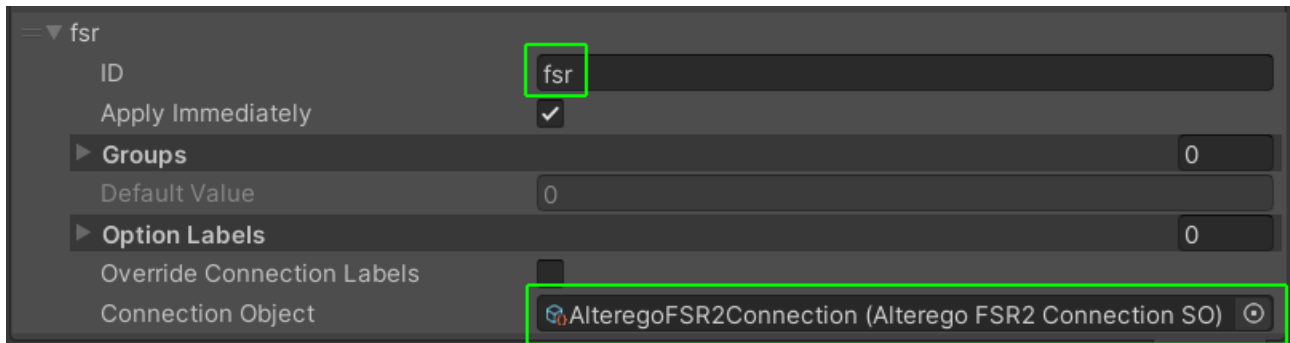


Notice the option labels are not localized. I'll leave localization to you. To localize them simply add the terms to the LocalizationProvider like any other setting localization.

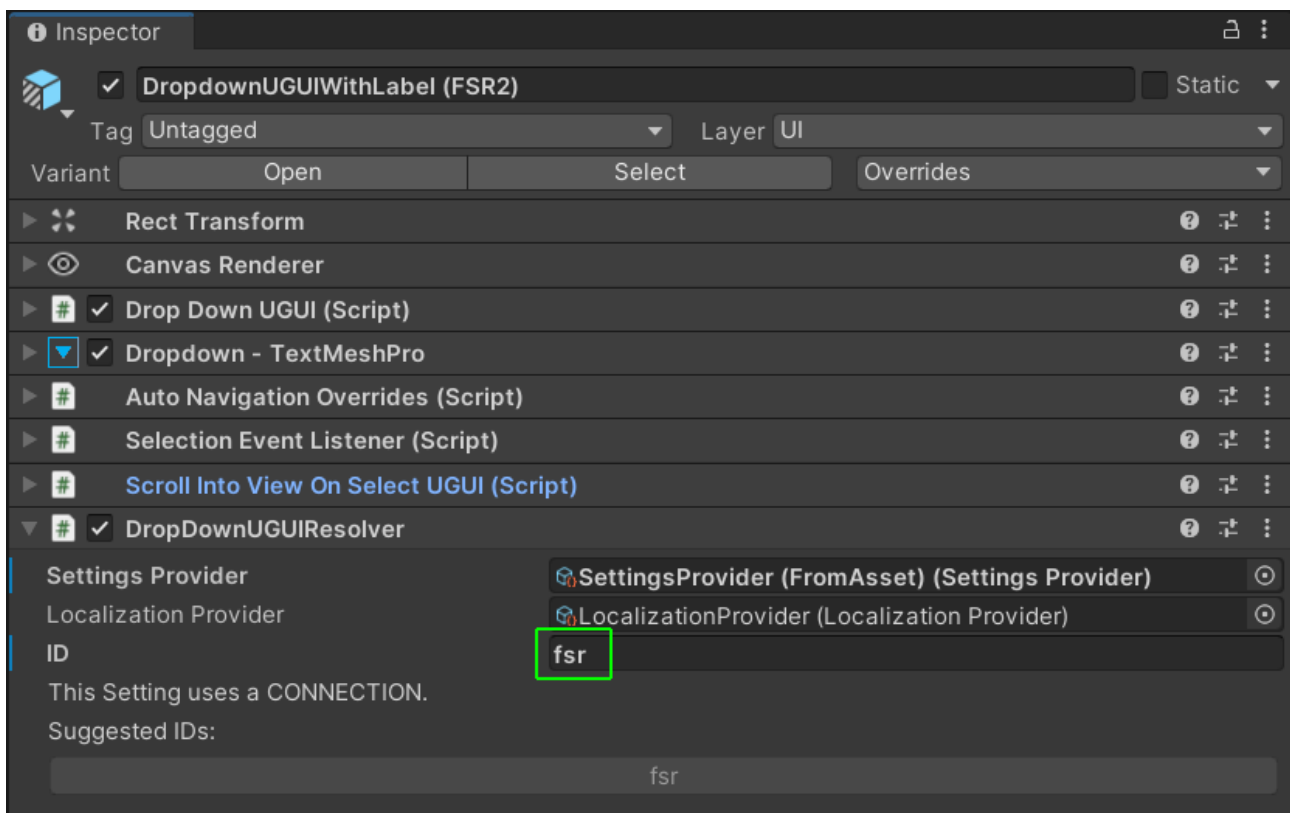
There is a premade „AltereGoFSR2Connection“ object which you can use in the settings.



Simply drop it in as the Connection Object:



In the UI the FSR2 setting works just like any other options setting. Notice that the FSR2 dropdown is not part of the default demo. You will have to add a new options dropdown on your own.



Common Issues / FAQ

Post Processing Settings do nothing

Solution: For Built-In renderer projects you need to install the post processing stack package from the package manager AND you will have to add a PostProcessing Layer to your camera and a PostProcessing Volume to your scene. Check out the examples under „Examples/FromAsset“. They already include a proper PostProcessing setup.

URP: Make sure you have „PostProcessing“ enabled on your main camera.

URP: Make sure that you are NOT using the Built-In postprocessing stack v2 package (URP has its own PostPro stack included).

Post Processing Render Context Null Pointer

You are getting an error like this:

```
NullReferenceException: Object reference not set to an instance of an object
UnityEngine.Rendering.PostProcessing.AmbientOcclusion.IsEnabledAndSupported
(UnityEngine.Rendering.PostProcessing.PostProcessRenderContext context) (at
Library/PackageCache/com.unity.postprocessing@2.1.3/PostProcessing/Runtime/
Effects/AmbientOcclusion.cs:179)
```

Solution: Recreate the PostProcessing Layer component on your camera or switch the Inspector to debug mode and set the „PostProcess Resource“ to a valid resource (see [forum](#)).

Post Processing not visible on mobile (Built-In render pipeline)

Post processing support for the built-in render pipeline is very hit and miss on mobile. Don't use Built-In PostPro on mobile. Use the URP renderer instead. It comes with its own PostProcessing stack and that one works on mobile too.

Controller X is not working properly (Old Input System)

Controller support for the OLD input system is based on the standard xbox controller layout. I encourage you to use the NEW InputSystem which has a much better controller abstraction included. If you are fixed to the old system then I'd recommend using [InControl](#) for controller abstraction.

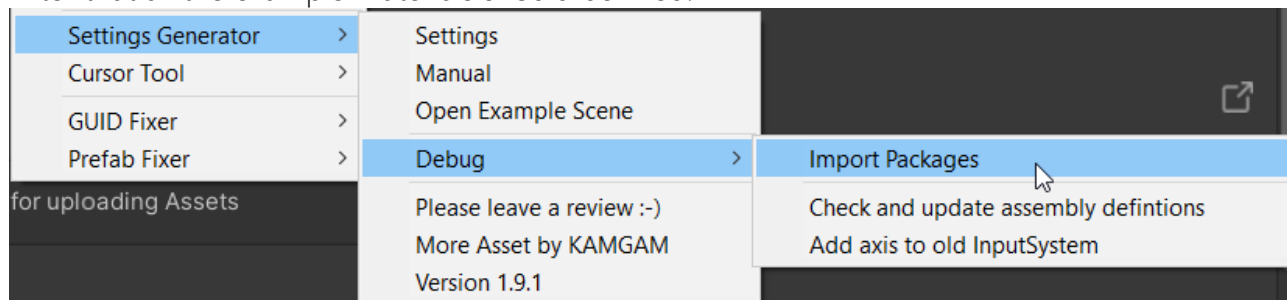
The example scenes are all pink after updating the asset (URP or HDRP)

URP and HDRP require different materials (shaders). After first install the tool imports these automatically but, depending on what version you are using, it may not do it after an update.

You will have to trigger the import manually by calling:

Tools > Settings Generator > Debug > Import Packages

After that all the example materials should be fixed.



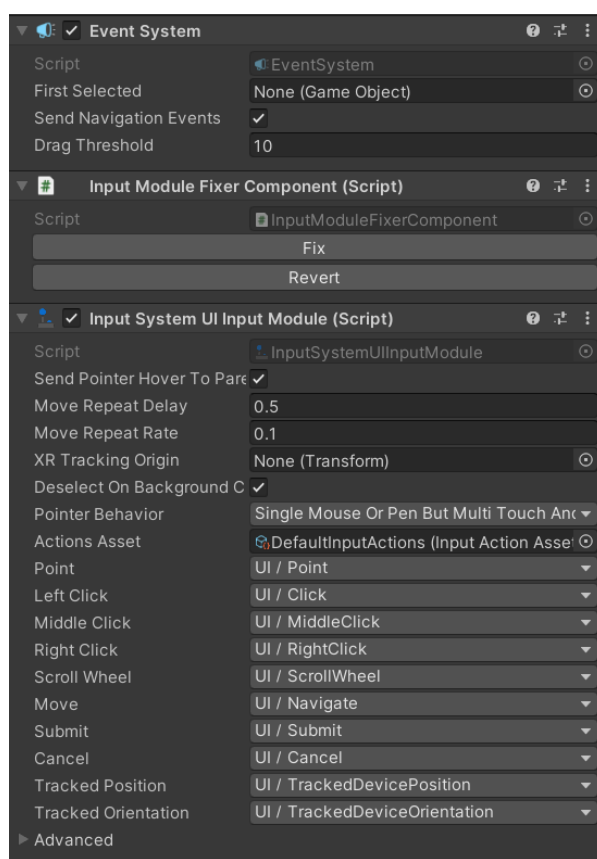
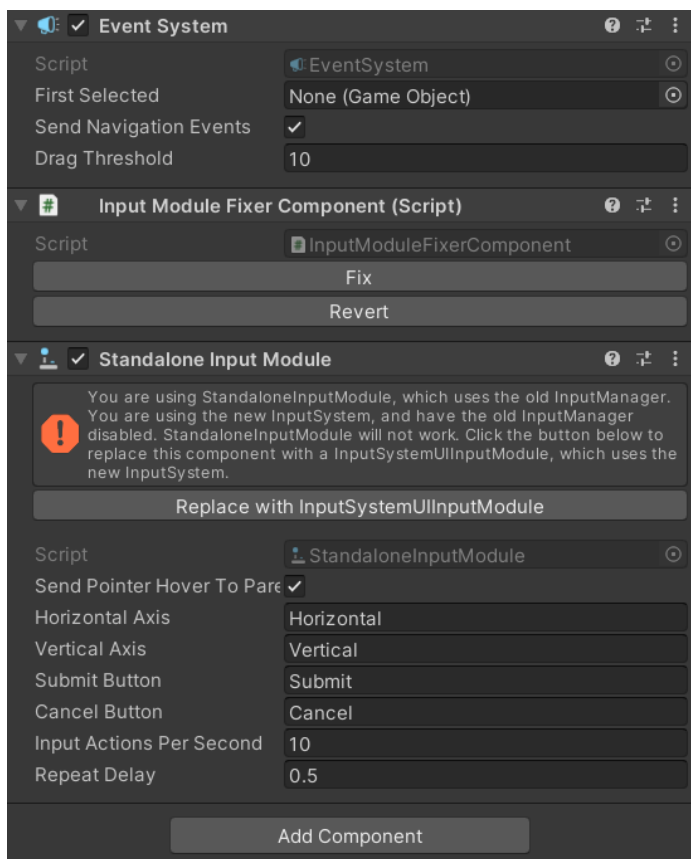
If you are not using the example scenes anymore then this step is not necessary. The settings system will work just fine if you don't do it. It's just the materials and example scenes that are changed by this not the system itself.

InvalidOperationException: You are trying to read Input using ..

InvalidOperationException: You are trying to read Input using the UnityEngine.Input class, but you have switched active Input handling to Input System package in Player Settings.

This may happen after upgrading the asset if you run the game without opening the scene first. Please open the demo (not the ui) scene once (it will fix itself automatically).

You can also fix it manually by using the Input Module Fixer > „Fix“ button

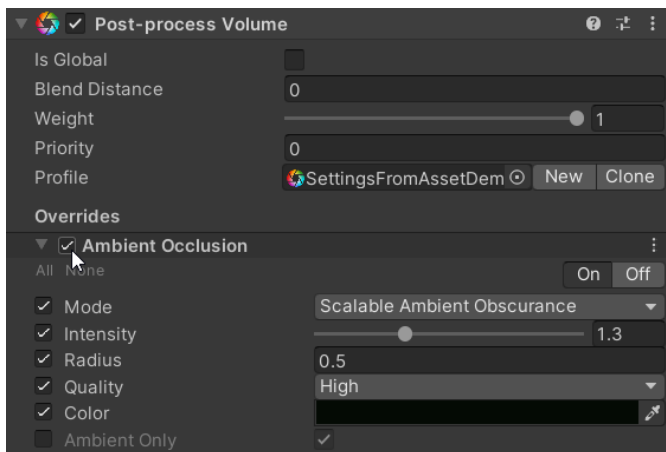


The Post Processing effect is not found

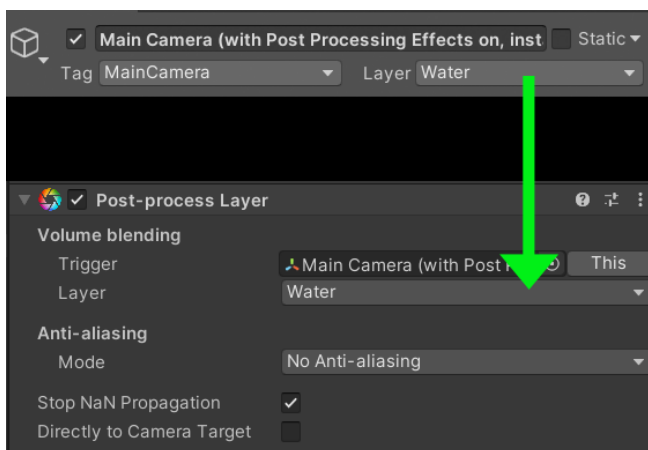
This will result in a warning like „There was no ambient occlusion setting found ..“ or „There was no active 'AmbientOcclusion' PostPro effect (BuiltIn) found.“

Basicall what you need is three prerequisites for post pro effects to work:

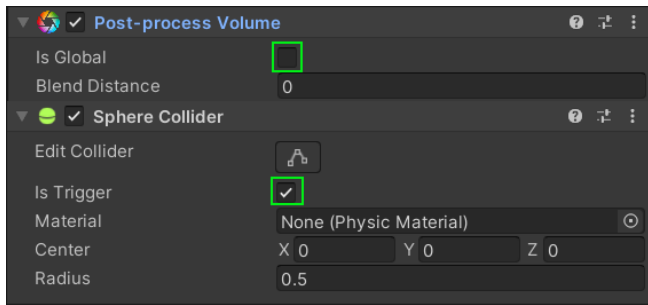
1) You have to add a PostPro Volume with a profile containing 'Ambient Occlusion' to your post pro Volume. Please also make sure it is enabled.



2) Make sure the layers of your camera and the PostProcessing-Layer component do match and that all the components are active.



3) If your Volume is NOT a global volume (global checkbox off) then you will have to add a Trigger Collider (a collider with "trigger" enabled). Place the collider directly on your Post-process Volume.



The demo scenes

(Assets/Kamgam/SettingsGenerator/Examples/FromAssets/SettingsFromAssetDemo) contains a working setup of this for you to look at. The demo uses the default "water" layer for post effects. You may want to change this.

A good test whether or not your setup is correct is if you see the effect in the "Game" view (is there some ambient occlusion). If you don't see it then the settings system will also not find it.

You may see these logs even if everything is set up correctly. Here is why:

At the start the settings system searches for volumes with post pro effects in all currently loaded scenes. However, if you are loading the scenes that contain the post-pro effects later (via code) then the settings system will log these messages because it can not find them at the start.

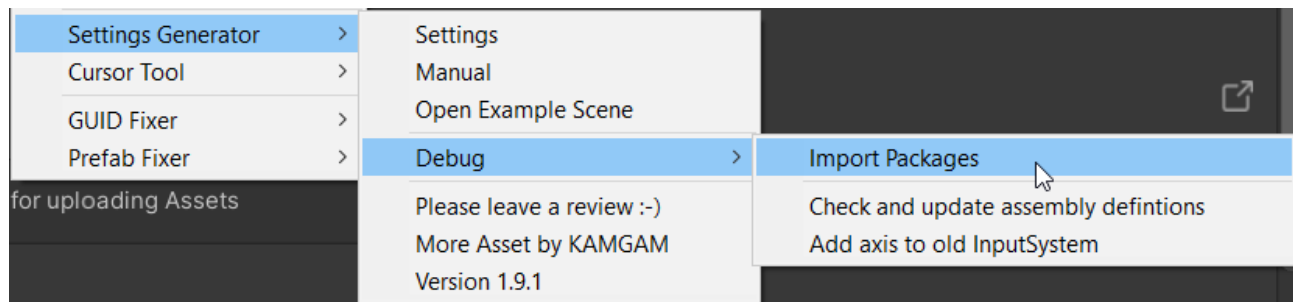
This has far reaching consequences because at the very first load the settings system will search for the post-pro effects to GET the default values. It does this to answer questions like: „Should bloom be ON or OFF when the game is loaded for the first time?“. If it can not find any bloom effect then that's when you see the warning message.

If you want to dive into how it searches for the post-pro effects then you can find the code in the "SettingsVolume.cs" class in "FindDefaultVolumeComponent<T>()" - It returns the active component of the first found global volume or a volume that is child of the main camera. This means that if your volume is a local volume somewhere in the scene then it will not be found (except if it's a child of the main camera).

The logs may also show if you initialize the settings system in Awake(). That's because the volumes initialize in Awake() too and thus they may be available only after the settings system was already initialized. To avoid this simply initialize the settings in Start() instead of Awake().

I have upgraded from an older version but don't see the new examples?

They are part of the fixes. Please import them manually by calling this:



Rebinding input actions has no effect. Why?

Either you have an error in your setup or (if the rebinding works in the demo) you are using the generated c# input actions class. If you do, then you will have to reassign the actions asset at runtime like this:

```
public class Test : MonoBehaviour
{
    YouInputActionsClass inputControls;
    public SettingsProvider SettingsProvider;

    void Start()
    {
        if (inputControls == null)
        {
            inputControls = new YouInputActionsClass();
        }

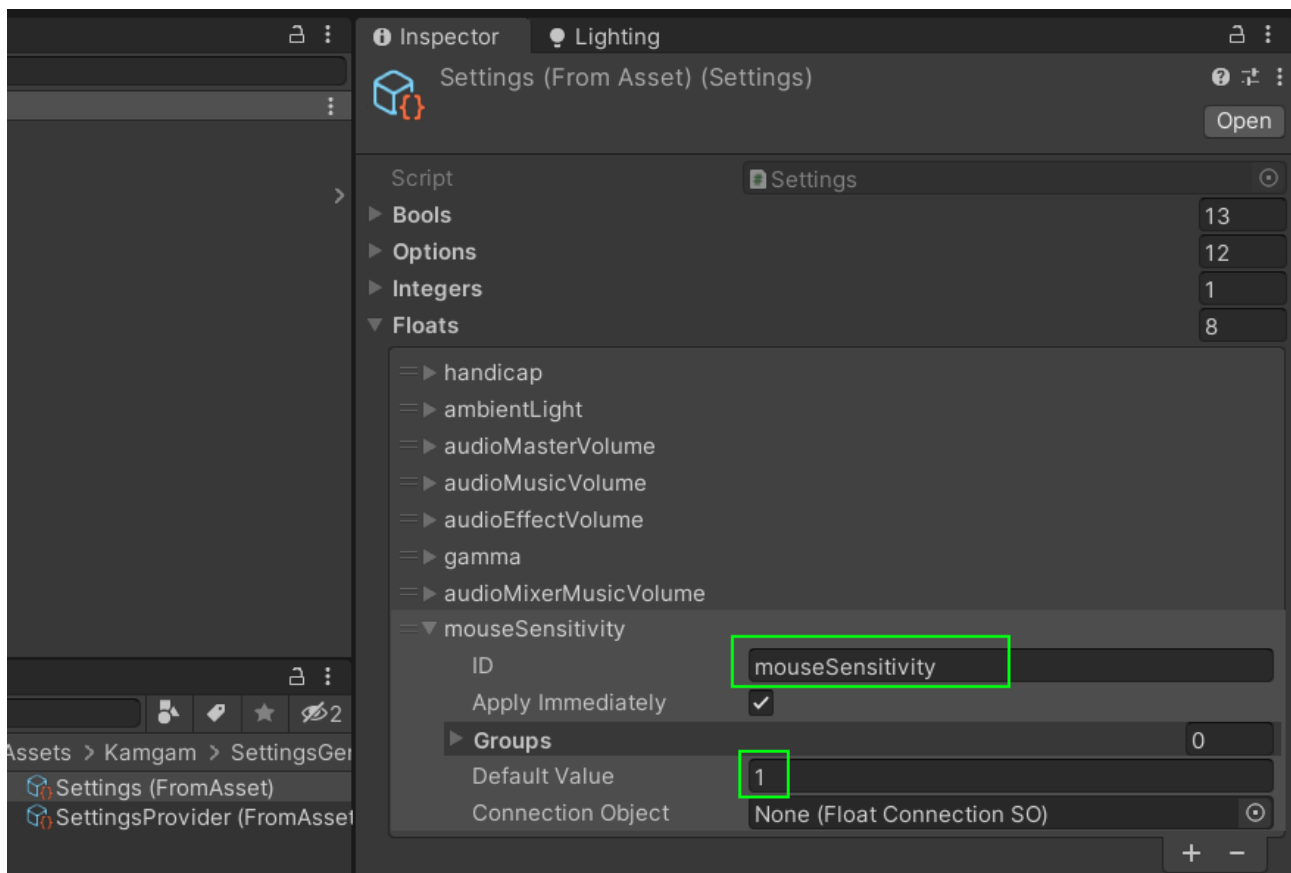
        var settings = SettingsProvider.Settings;
        settings.SetInputActionAsset(inputControls.asset);
    }
}
```

For more details please refer to the Input Binding section.

How to add a „Mouse Sensitivity“ setting?

You may have noticed that there is no MouseSensitivityConnection despite that being one of the most common settings found in many games. It is on the list of features to investigate, though it is not an easy one to add. The sensitivity value itself would need to be fed into whatever Character Controller is used (and sadly there are many different ones). Also the settings system can not know what changes have been made to the controller (i.e. if your code overrides the value too, which would generate a conflict).

At the moment the recommend way is to simply add a float setting and use that as a multiplier for your mouse sensitivity. Like this:



In your code you can then fetch that value like this:

```
// "SettingsProvider" is the SettingsProvider asset.
```

```
Settings Provider SettingsProvider (FromAsset) (Settings P ⓘ)

var multiplier = SettingsProvider.Settings.GetOrCreateFloat("mouseSensitivity");
```

The correct place and time to use the „multiplier“ variable depends on your setup. This is the tricky part. You may apply it within the character controller or in your input code.

Please ask the developer of your character controller on how to best apply a mouse sensitivity value.

There are some errors in URP or HDRP Connections but I am not using URP or HDRP?!

UPDATE: As of version 1.9.1 this should no longer happen.

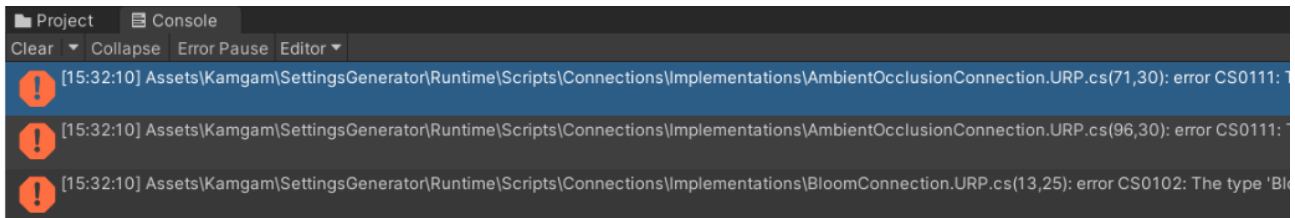
This happens if you have two render pipelines installed at the same time. There are three render pipelines: Built-In (the old standard), URP (Universal) and HDRP (High Definition).

The Settings Generator uses the installed packages information to guess which render pipeline you are using. If URP or HDRP are installed it will assume that you are actually using those.

The system tries to find out what pipeline you are using automatically. Though, if you have multiple installed it will enable the code for all of them (via `#ifdef`). This leads to duplicate code (one for each pipeline).

Please make sure you only have one render pipeline installed at once!

Possible Errors: Null References, Duplicate Connection, .. already defines a member ...



If you are NOT using URP or HDRP then please double-check in the [PackageManager](#) that these packages are NOT installed. They may have unexpected names like URP, which is named „Universal RP“ in the package manager.

If you are using only the Built-In renderer then please uninstall the URP (Universal Render Pipeline) package and also uninstall the HDRP (High Definition Render Pipeline) package.

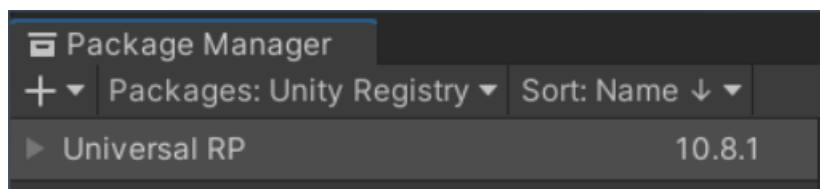
If you are using the URP (Universal Render Pipeline) then make sure you do uninstall the HDRP (High Definition Render Pipeline) package and the other way around.

Maybe also restart Unity once after uninstalling the packages . It sounds ridiculous but it sometimes helps.

EXAMPLE:

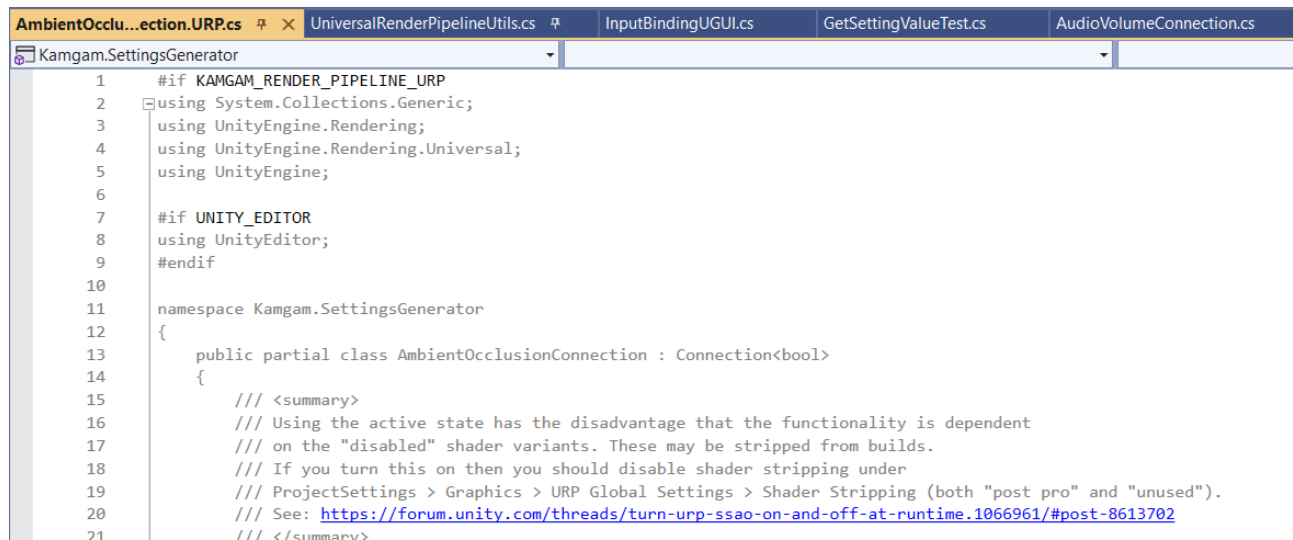
Let's assume you are not using URP but the URP classes show some errors after import.

Make sure to uninstall URP if you are using Built-In or HDRP.



If you open the file with the error in your IDE (VisualStudio) then the code should be greyed out since you are not using URP.

It should look like this:



```
1  #if KAMGAM_RENDER_PIPELINE_URP
2  using System.Collections.Generic;
3  using UnityEngine.Rendering;
4  using UnityEngine.Rendering.Universal;
5  using UnityEngine;
6
7  #if UNITY_EDITOR
8  using UnityEditor;
9  #endif
10
11 namespace Kamgam.SettingsGenerator
12 {
13     public partial class AmbientOcclusionConnection : Connection<bool>
14     {
15         /// <summary>
16         /// Using the active state has the disadvantage that the functionality is dependent
17         /// on the "disabled" shader variants. These may be stripped from builds.
18         /// If you turn this on then you should disable shader stripping under
19         /// ProjectSettings > Graphics > URP Global Settings > Shader Stripping (both "post pro" and "unused").
20         /// See: https://forum.unity.com/threads/turn-urp-ssao-on-and-off-at-runtime.1066961/#post-8613702
21         /// </summary>
```

If the code is not greyed out then you still have references to URP assemblies in your project. Find those and remove them (usually removing from the package manager is enough)

My settings work in the menu but not in the level?

There are two common reasons for this:

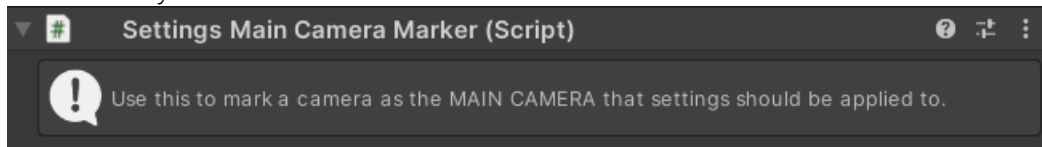
- A) Some third party asset is resetting the settings (maybe even every frame).
- B) You have unloaded your UI scene and you do not have set „DoNotDestroy“ on the Settings_INITIALIZER and you do not use a SettingsApplier in your scene.

Please refer to the „[Combining the Settings System with other Assets](#)“ section at the beginning of the manual for more details on how to fix these.

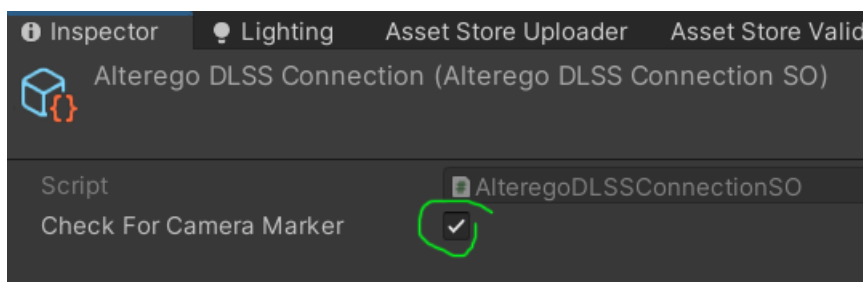
DLSS: I have a multi camera setup and the DLSS setting does only apply to the camera tagged with „MainCamera“.

The settings system assumes that usually the camera with the MainCamera tag is the one you want to configure with the settings. If you want to specify a camera without tags then you can add the „SettingsMainCameraMarker“ component to a camera and enable the „

Add this to your camera:



Enable Marker Search in the connection:



I can't get mouse interaction to work with the settings prefabs?

Solution: It may be that you are missing an **EventSystem** in your scene (usually by adding a new canvas one is created automatically).

The thing is that in the demo (where the UI is loaded additively) the EventSystem is in the main scene (not the UI scene). Therefore if you copy only the UI scene into your project you will have to make sure that your scene already has an EventSystem in it.

Usually people already have one but this can be easily overlooked.

Remember: For interaction with canvases you need a RayCaster on the canvas AND and active EventSystem in the scene.

