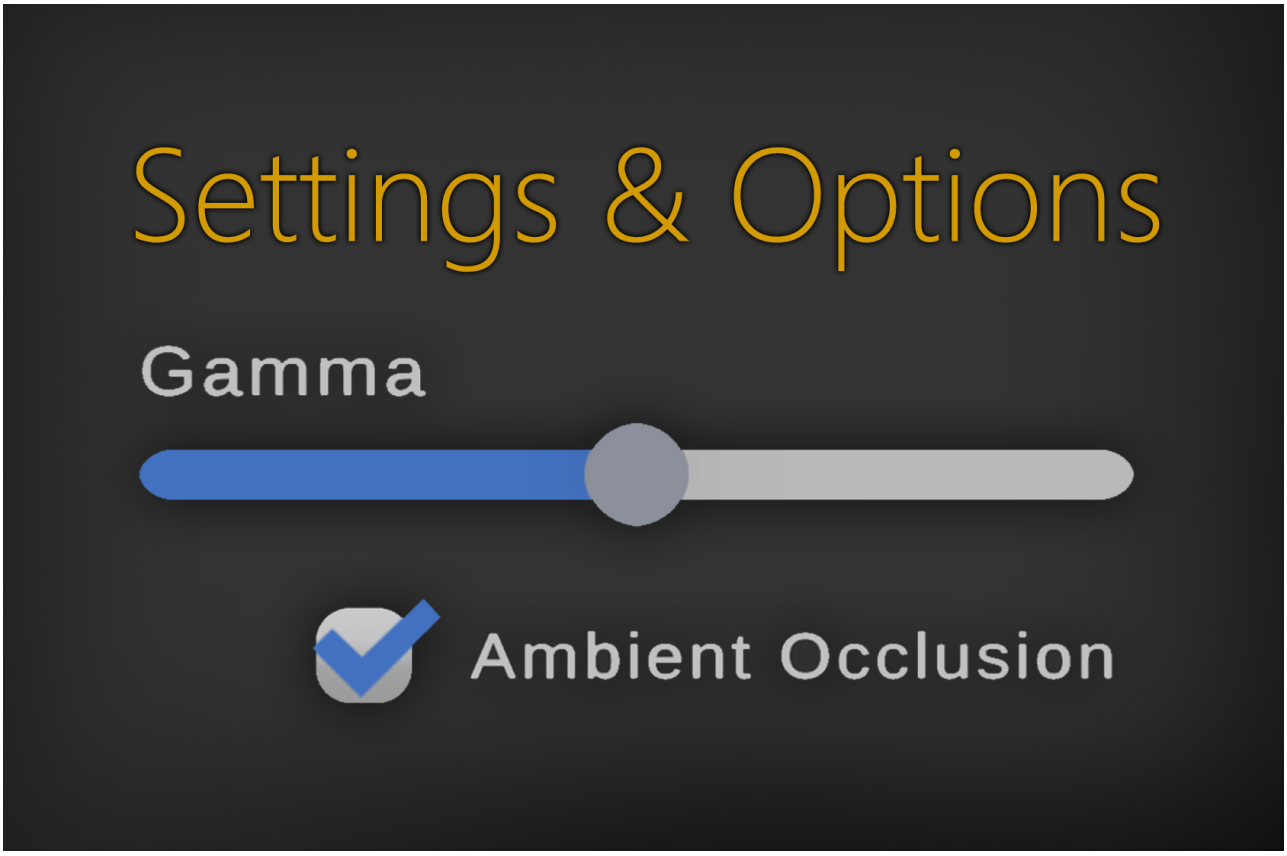
Setting Generator



Generate settings for your game within seconds.

Table of contents

Overview (READ THIS FIRST) **4**
What is it doing exactly? 4
How is it doing it? (aka „How to generate a new setting“) 5
What are those IDs? 8
Creating your own Settings and Settings Provider 9
When are the settings applied for the first time? 13
UGUI Components **14**
Implementations 15
Color Picker UGUI & Color UGUI 15
Dropdown UGUI 16
Headline UGUI 17
Input Key UGUI 17
Input Binding UGUI 17
Options Button UGUI 18
Slider UGUI 19
Stepper UGUI 20
Textfield UGUI 21
Toggle UGUI 21
Helpers 22

Auto Navigation Overrides....22
Selection Event Listener....22
Scroll Into View On Select UGUI....23
SelectionUGUI....23
Setting Resolvers....24
Setting....25
Basic Types (Bool, Int, Float, String)....25
ColorOptions....26
Options....26
Key Combination (preferred solution for the old InputSystem)....26
Input Binding (preferred solution for the new Input System)....26
Connections....27
Ambient Occlusion (SSAO)....28
Caveats (mostly URP)....28
Anti Aliasing....28
Audio Paused....29
Audio Volume....29
Audio Source Volume....29
Bloom....29
Depth Of Field....29
Frame Rate....29
Full Screen....29
GetSetConnection<T>....30
Gamma....30
InputBindingConnection....30
Motion Blur....30
Monitor....30
Quality....30
Refresh Rate....30
Resolution....30
Shadow....30
Shadow Distance....31
Shadow Resolution....31
Texture Resolution....31
Vignette....31
V-Sync....31
Window Mode....31
Input Rebinding (Using the new Input System)....32
Rebinding Overview....32
Rebinding Tutorial....33
1) Generating the InputBinding Connections....34
2) Link InputBinding Connections to Settings....35
3) Configuring the Rebinding UI....35
Scripting API (Adding custom Settings)....38
A: Custom Code Tutorial (using the GetSetConnection<T>)....38
B: Custom Code Tutorial (making a custom Connection class)....39

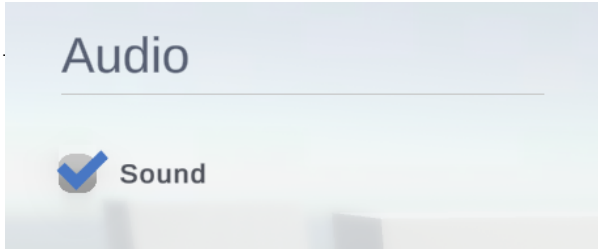
Localization **42**
Visual Scripting (formerly BOLT) **43**
Configuration & Debugging **45**
Common Issues 45
Post Processing Settings do nothing 45
Post Processing Render Context Null Pointer 45
Post Processing not visible on mobile (Built-In render pipeline) 46
Controller X is not working properly (Old Input System) 46
The example scenes are all pink after updating the asset (URP or HDRP) 46

Overview (READ THIS FIRST)

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:

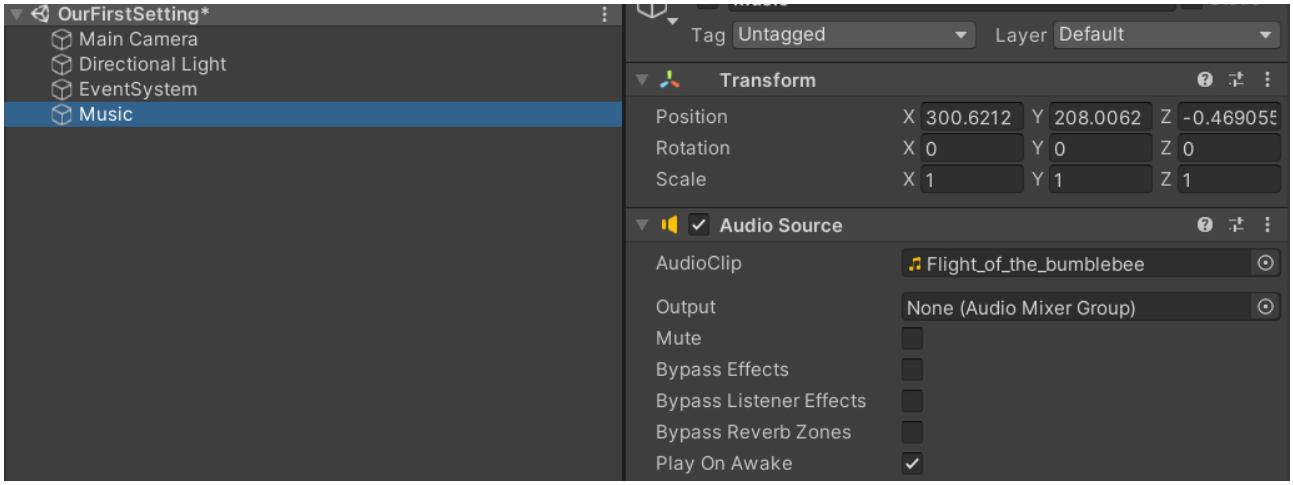
```
public override void Set(bool audioPaused)
{
    AudioListener.pause = audioPaused;
}
```

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.

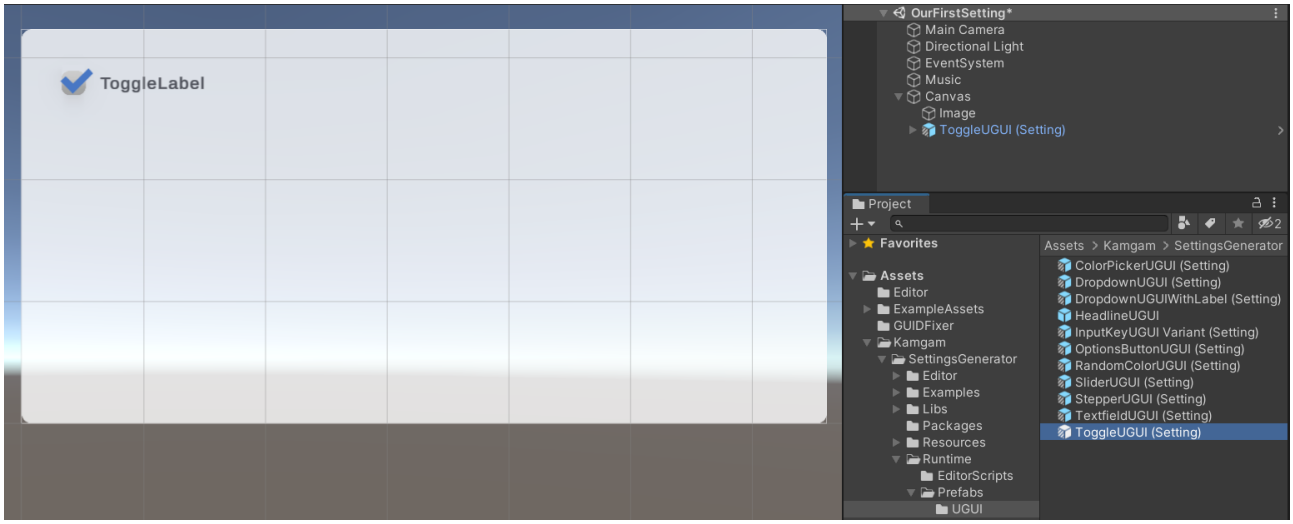
How is it doing it? (aka „How to generate a new setting“)

Let’s create that audio setting we have just seen.

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).

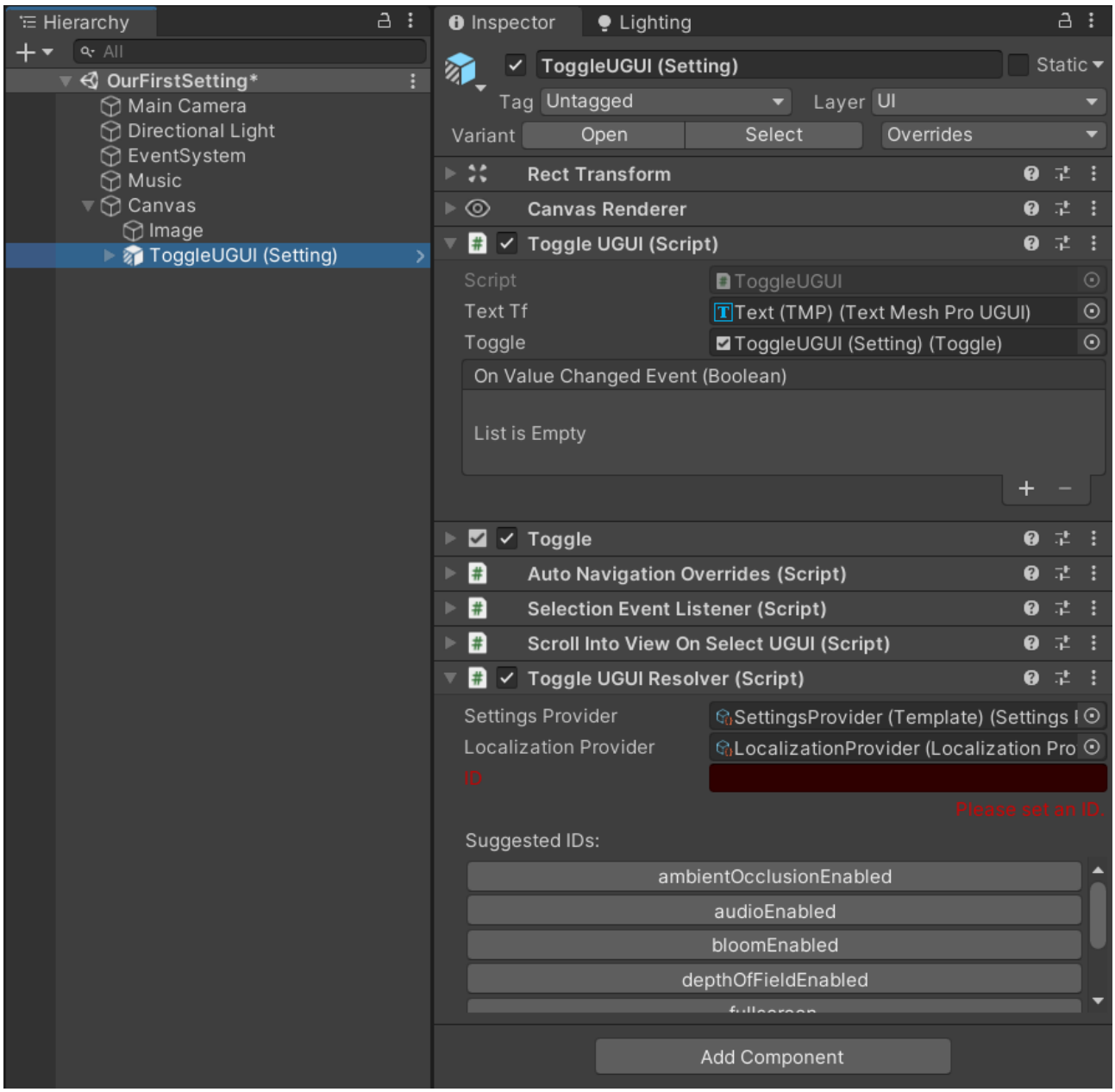


Okay, now first we need a UI. For this demo we will use the „ToggleUGUI (Setting)“ prefab. You can of course use your own UI too. The Prefabs are located under „**Runtime/Prefabs/UGUI**“ within the Settings Generator folder.

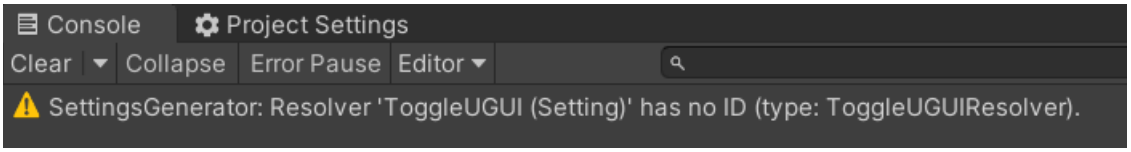


If you select the toogle you will see a LOT of components in the inspector. Most of them are just for convenience and we can ignore them for now. The most important one is:

„**Toggle UGUI Resolver**“.



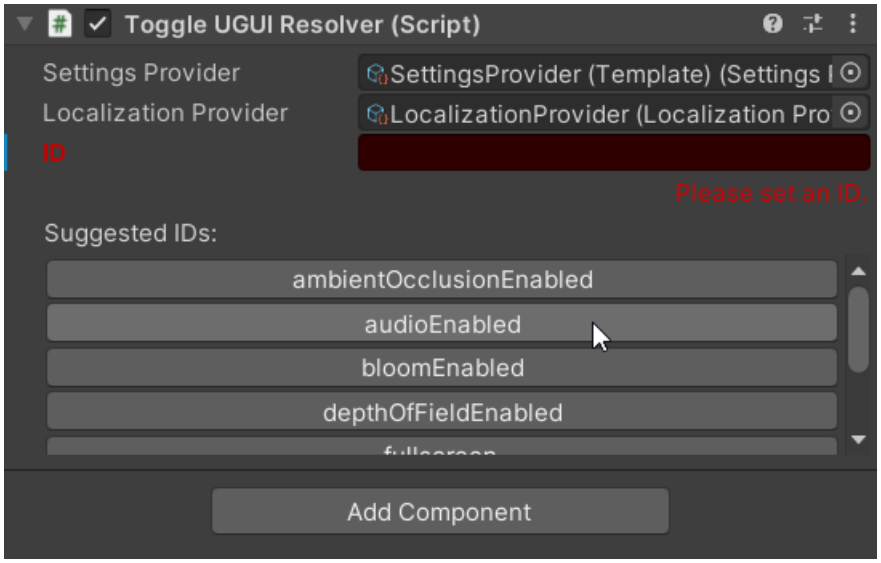
Before we make any changes let's just run our little scene and see what happens.



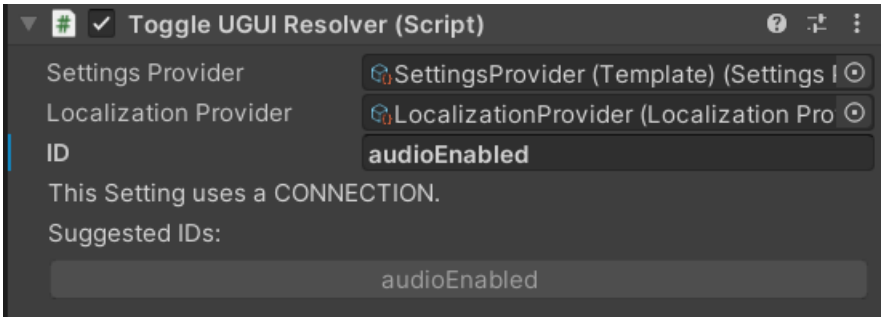
Okay, the settings tool tells us something about our newly added toggle ui is wrong. It complains that it has no ID. Now this is important. **IDs are how the UI finds the setting it belongs to.**

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.



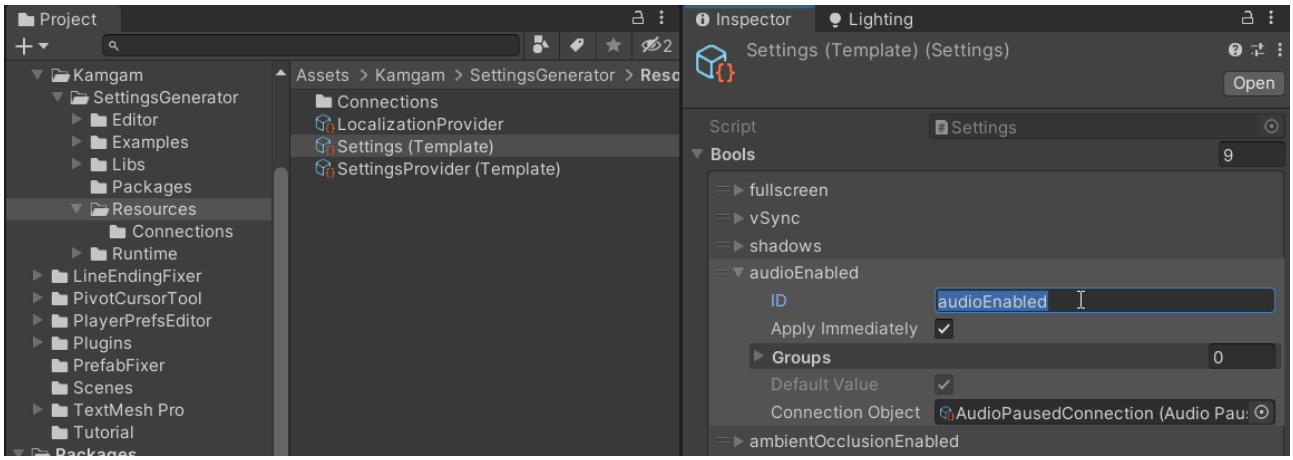
Okay great, the warning is gone and if we hit the toogle it pauses/unpauses the audio playback.

What are those IDs?

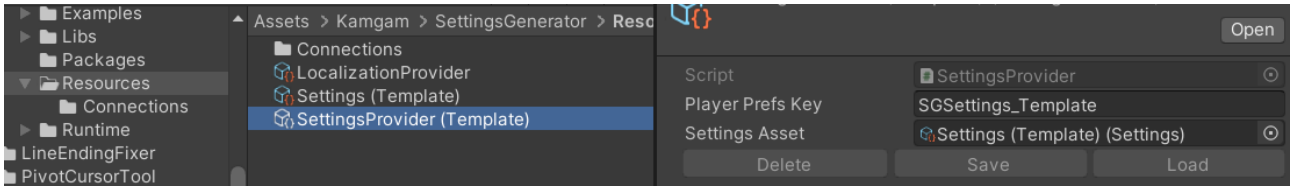
An ID is a unique name for a setting (so we can find it easily). The settings are defined within a **Settings.asset** file (surprise).

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).

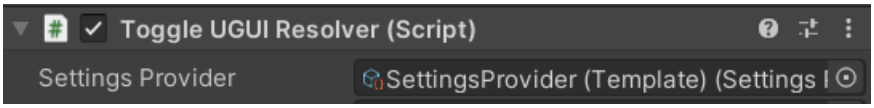


Before we dive into the details of the „audioEnabled" setting itself let's first talk about the other important asset within the Resources folder, the **SettingsProvider:**

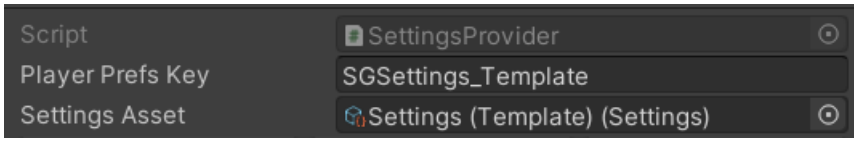


The **SettingsProvider** is the object that connects the settings with any object that is interested in it (it „provides" the settings to anyone asking for them).

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

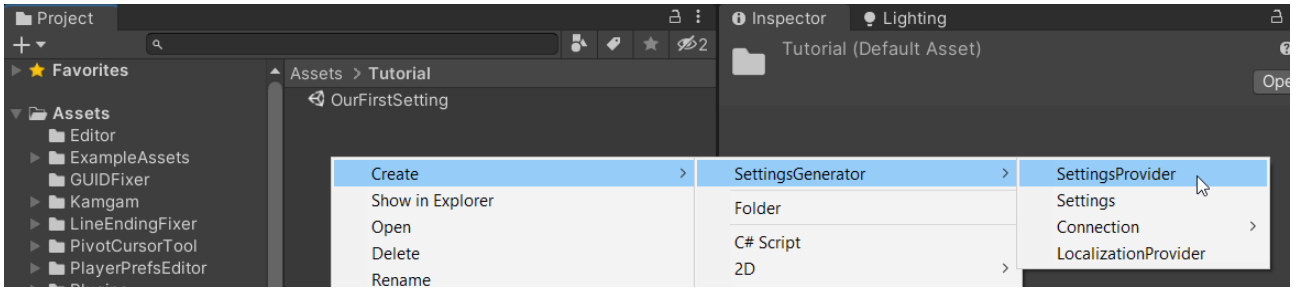


So the connection is: **Toggle UGUI Resolver** > **SettingsProvider** > **Settings** > Setting (ID)

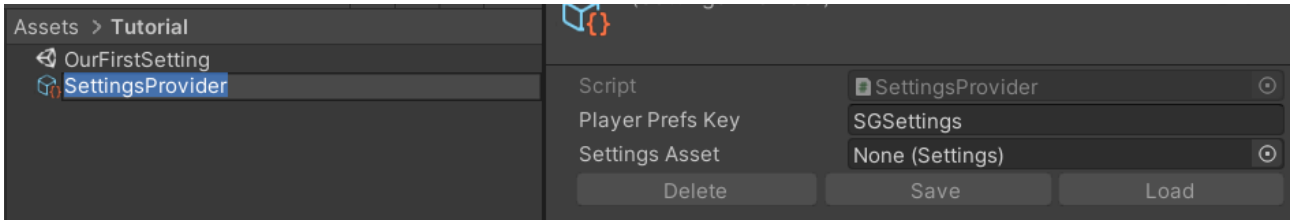
Creating your own Settings and Settings Provider

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).

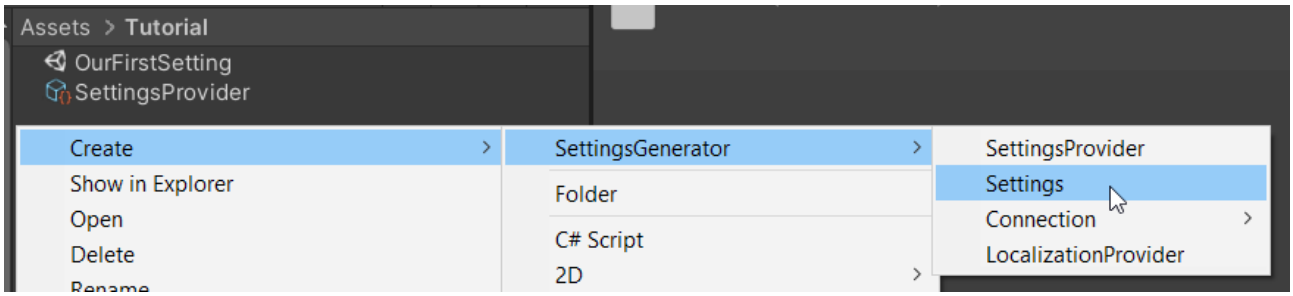
Let's go back to our audio scene and make our very own Settings and SettingsProvider. You can do that via „**Right Click** > **Create** > **SettingsGenerator** > **SettingsProvider**“.



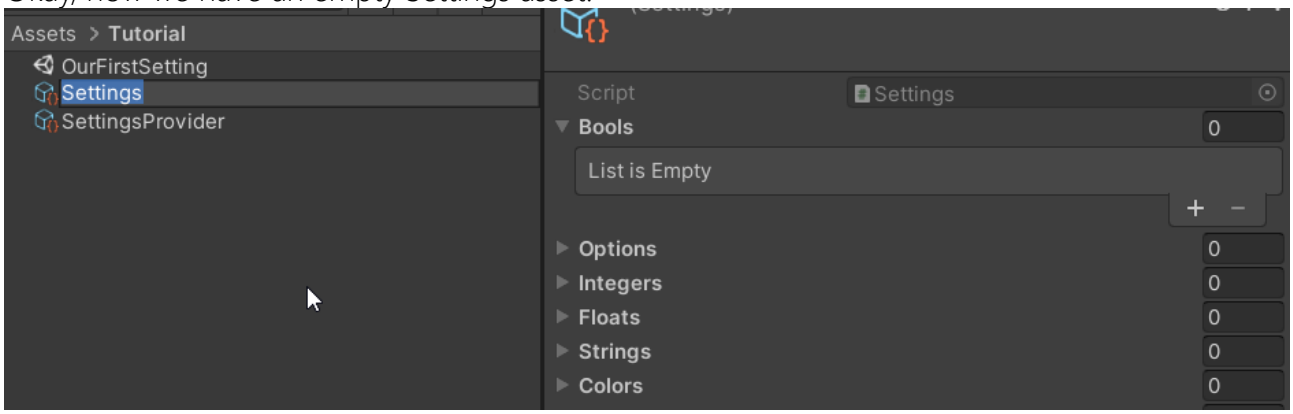
The result will be a new SettingsProvider without a Settings asset:



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

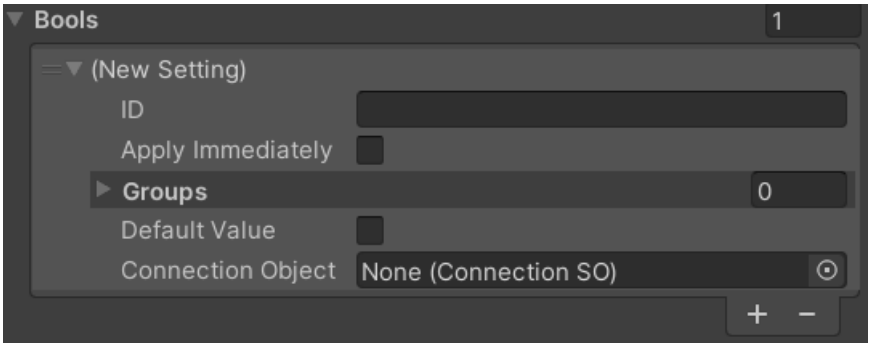


Okay, now we have an empty Settings asset:

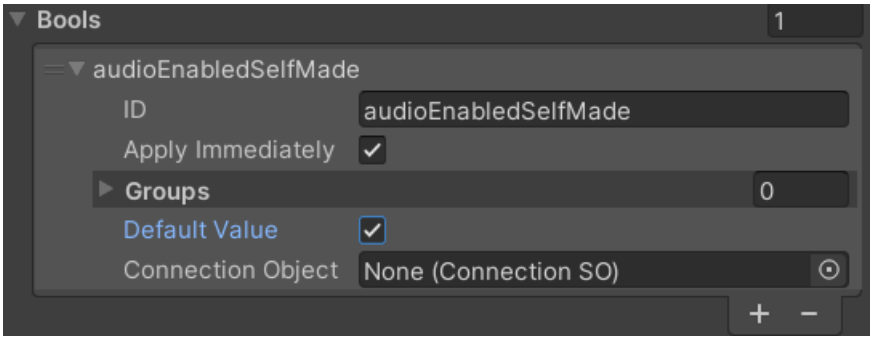


Let's add our own „audioEnabled" setting to the Settings asset. To do that click on the „+" at the bottom of the Bools list. It will look like this:

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



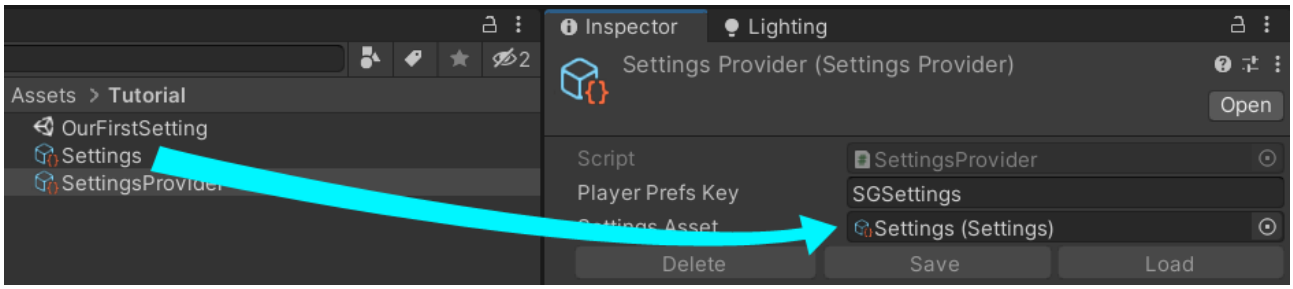
Now we enter our ID „audioEnabledSelfMade" and check the „Apply Immediately" and „Default Value" checkboxes.



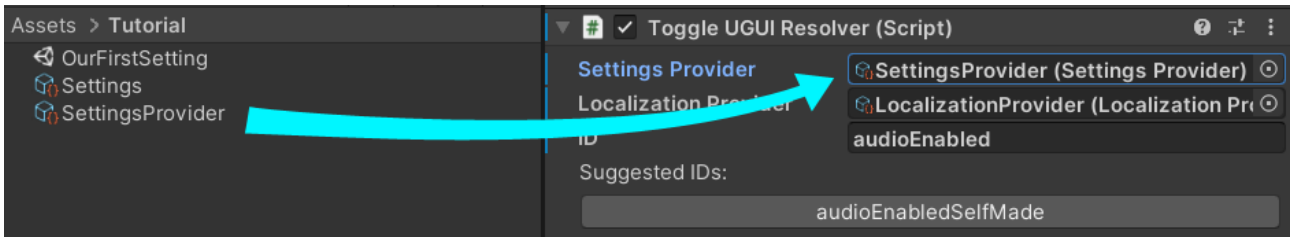
Okay, we have made our first setting (yay).

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

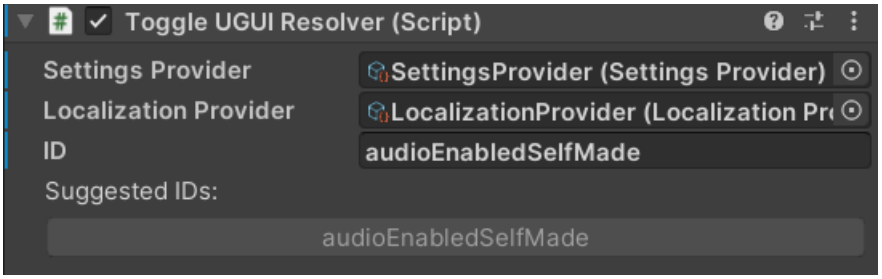
The first thing to do is to hook the Settings up with our SettingsProvider. Simply drag the settings into the „Settings Asset" field of the provider.



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



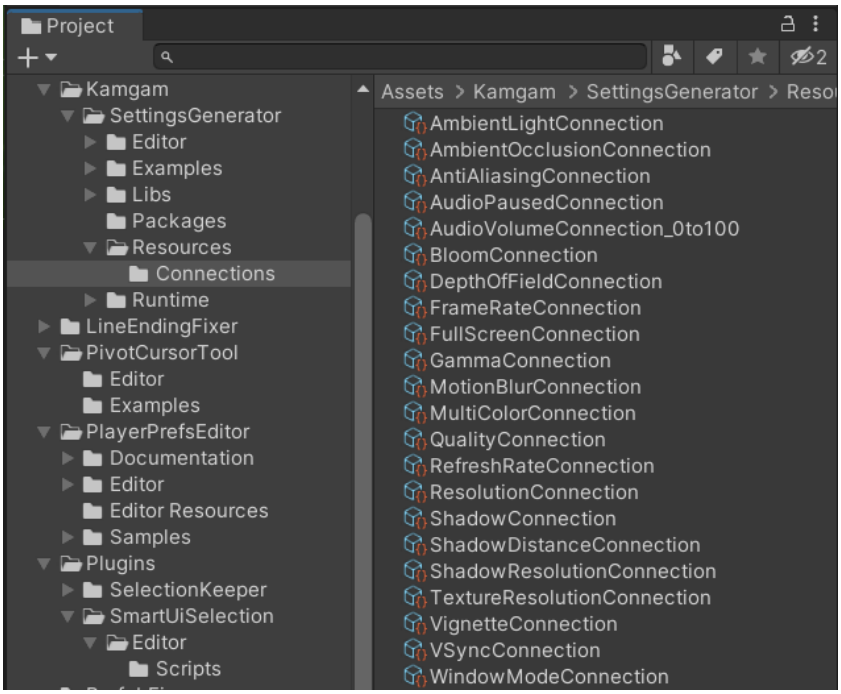
Notice how the „Suggested IDs“ change to the ID we have added to our own Settings asset. Update the ID to use our new setting ID.



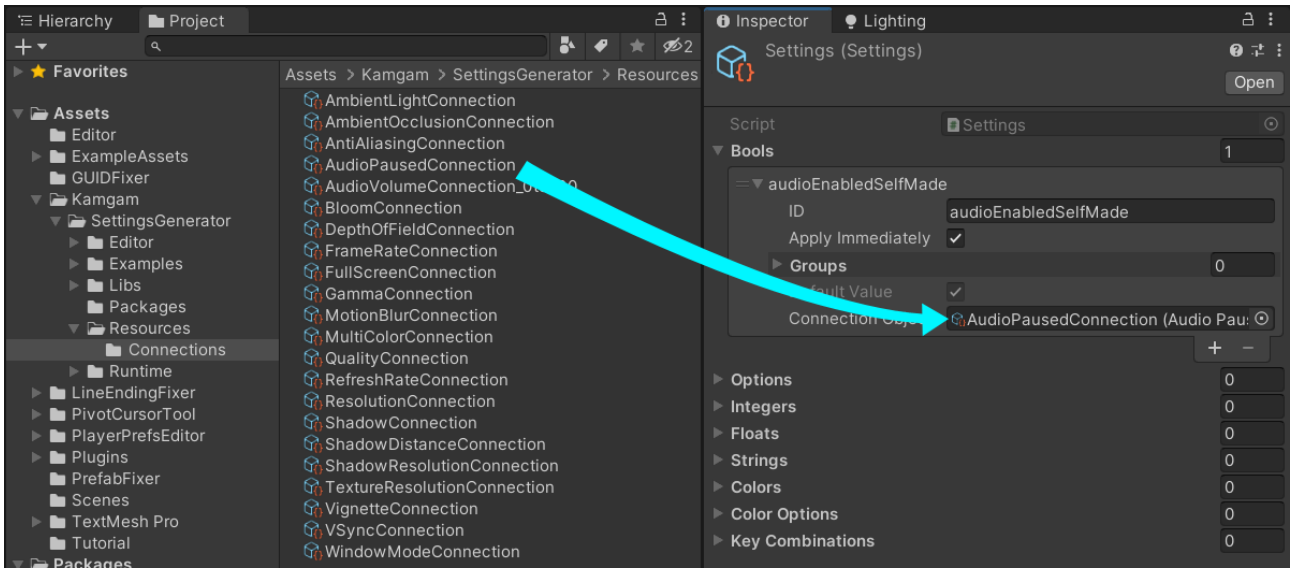
Okay, ready? Let’s start our demo scene again.
Aaaand it does nothing ?!? And there is no warning shown. How disappointing :-/

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).

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).

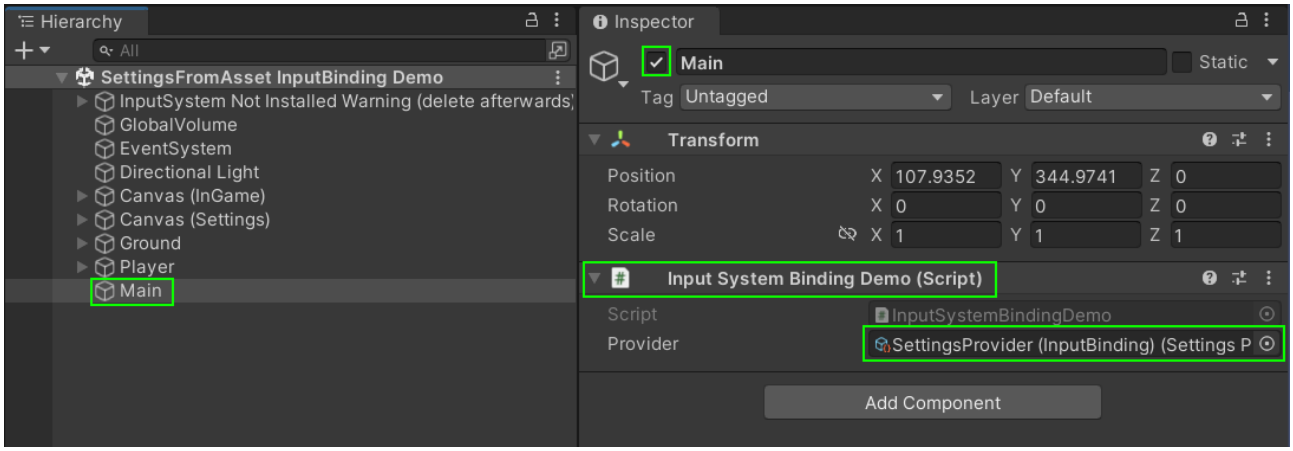
But most games don't immediately show the settings at the start so how do we intialize the settings without showing the UI? Well, all you need is just one line of code on a game object in your very first scene.

The „InputSystemBindingDemo.cs" is a nice example:

```
using UnityEngine;

namespace Kamgam.SettingsGenerator.Examples
{
    public class InputSystemBindingDemo : MonoBehaviour
    {
        public SettingsProvider Provider;

        public void Awake()
        {
            // 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:

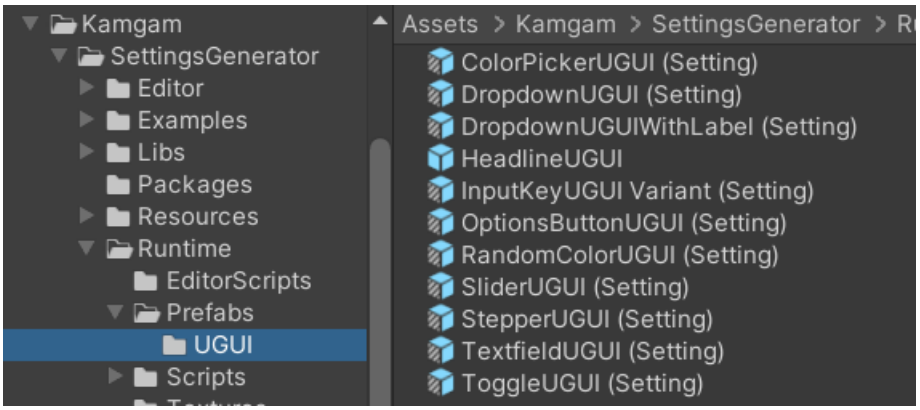


Hint: Be sure to link the right SettingsProvider if you have multiple.

That's it.

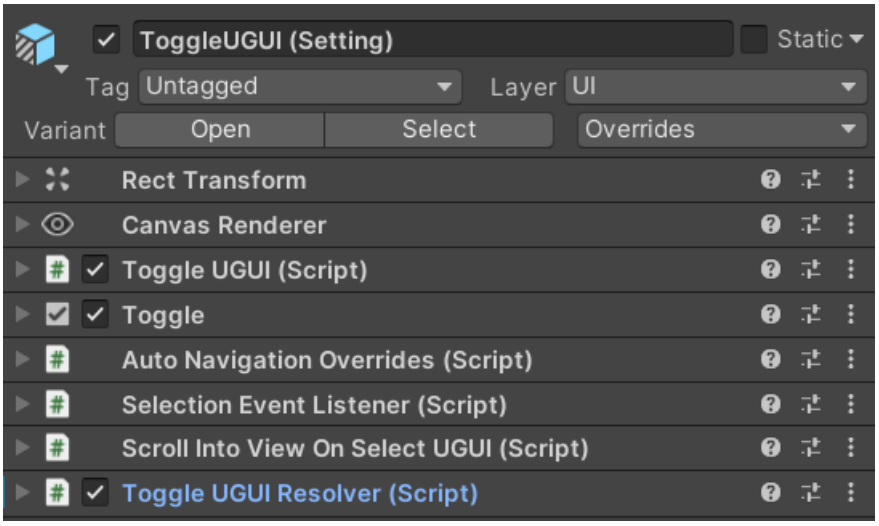
UGUI Components

The settings generator contains a set of UGUI prefabs which 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).



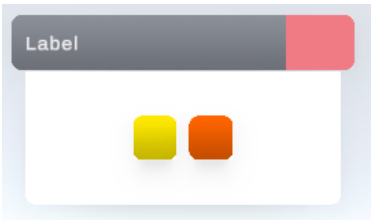
Implementations

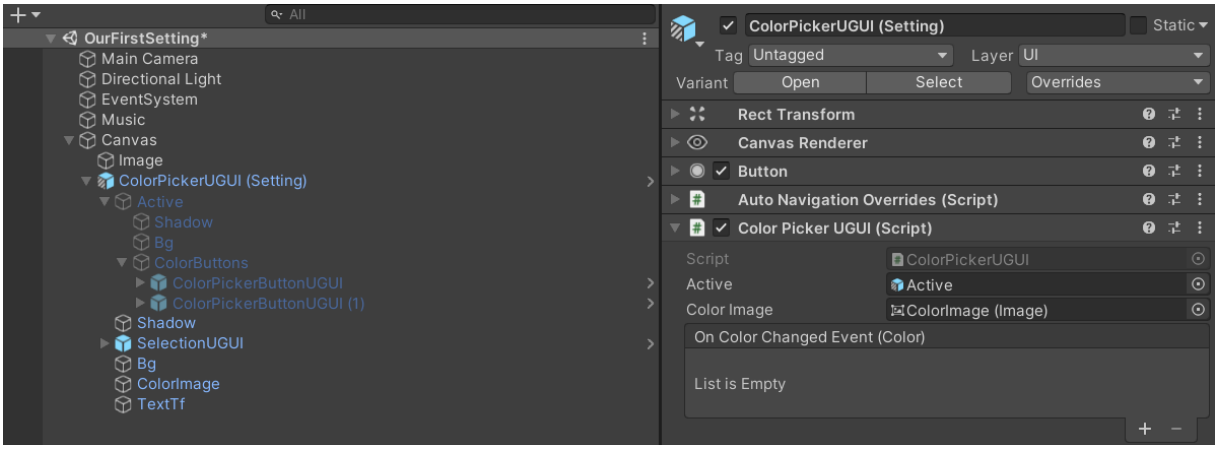
When searching for component prefabs make sure you are not accidentally using the components from the „Libs/UGUIComponentsForSettings“ library. These are the base prefabs.

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.

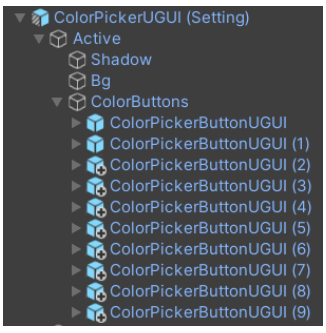
Color Picker UGUI & Color UGUI

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).





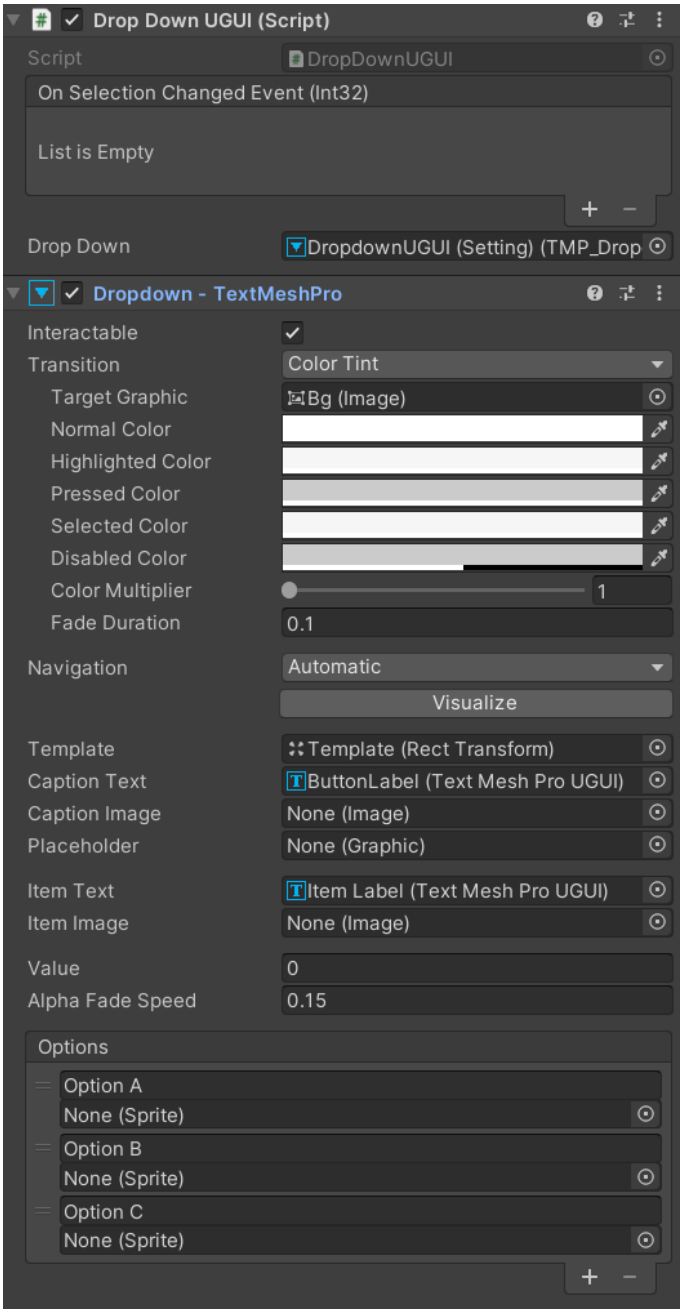
If you need more colors then duplicate the buttons and specify the color within each button.



The ColoerPickerUGUI 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.

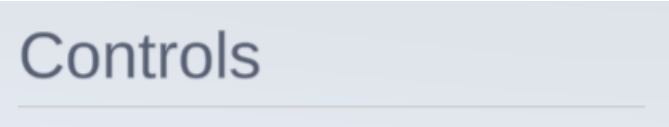
If C is present then it will override A and B.

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

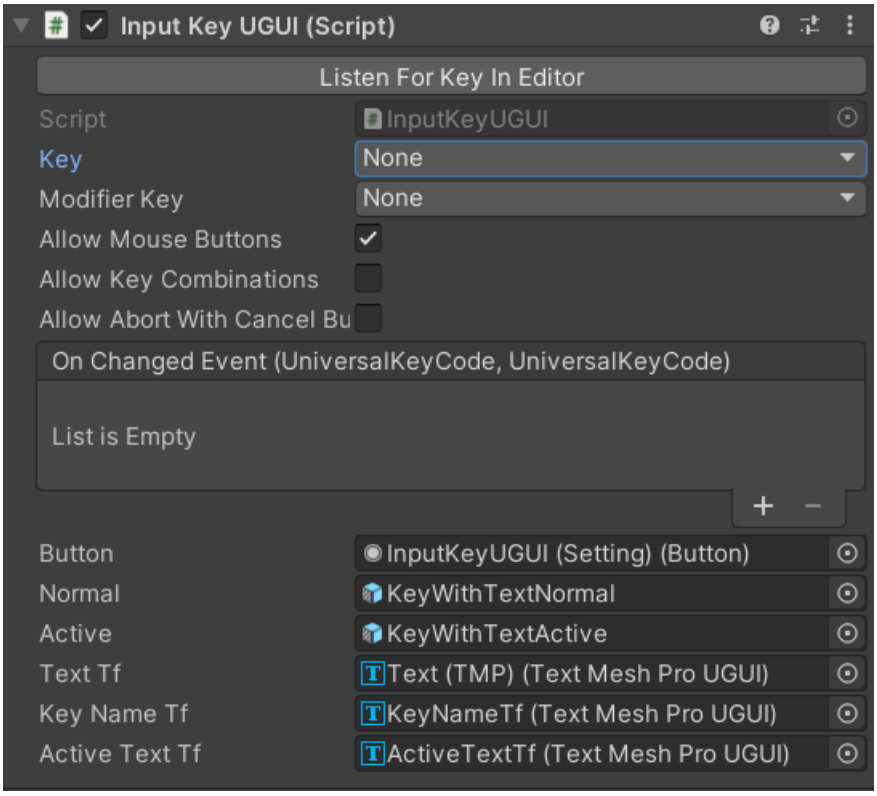
Just a label with a line beneath it. No settings related stuff here.



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

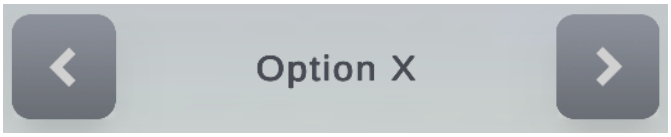
Similar to Options but it simply iterates through the options and shows the current one in the center. It is best suited for mobile devices or very short lists where a dropdown would be to cumbersome.

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).

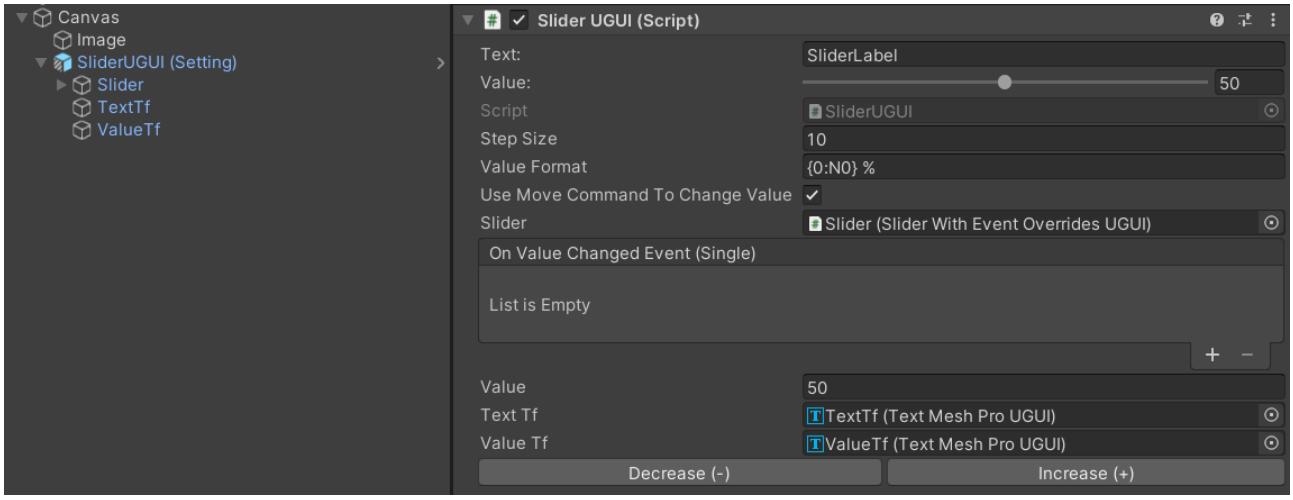
Slider UGUI

The slider is built based on the default Unity Slider.

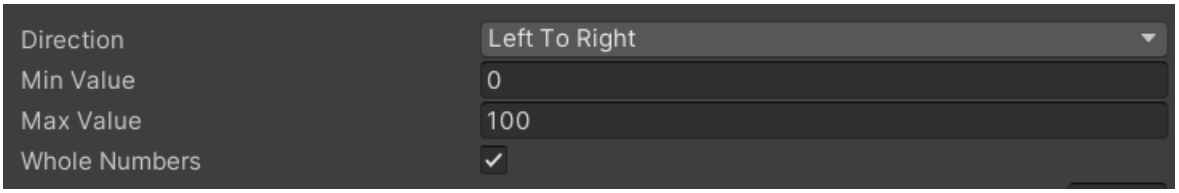
You can specify a value format which is based on the c# Standard Numeric Format.

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.

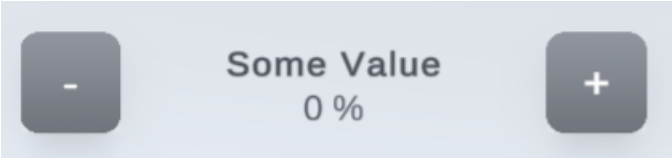


Some values (min, max, int/float) are configured directly on the Unity Slider.

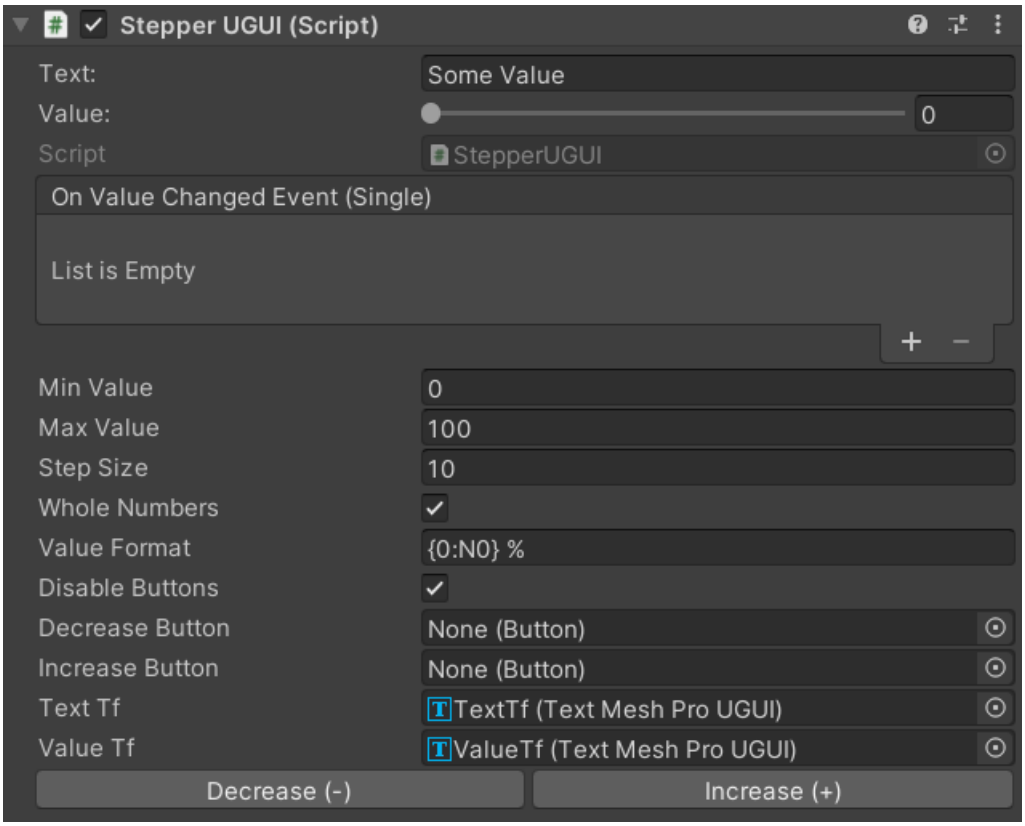


Stepper UGUI

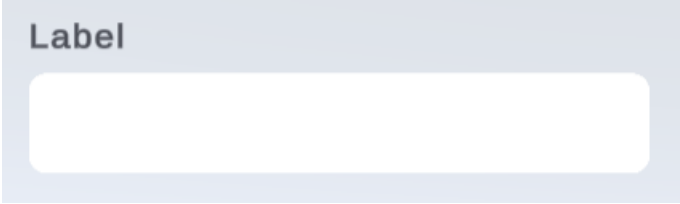
The stepper is similar to a slider but it makes thes steps more explicit. This one is useful for touch interfaces or for options with few values.



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.



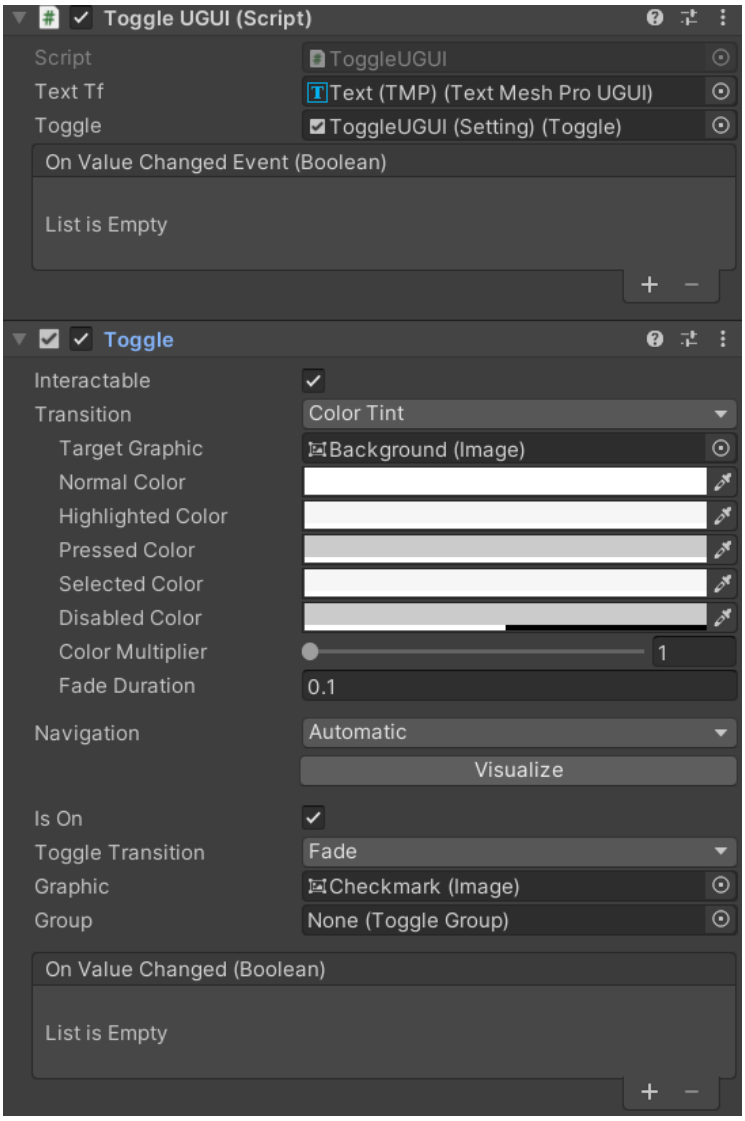
Textfield UGUI



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.



Helpers

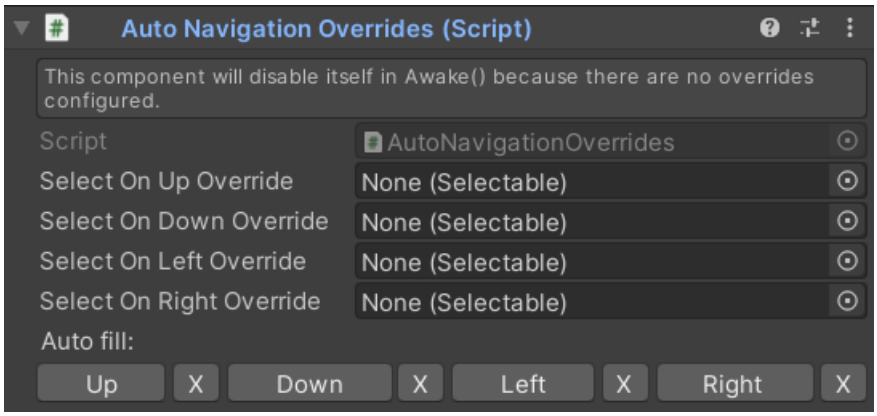
Various ccomponents are attached to the UGUI components to help out with common tasks.

Auto Navigation Overrides

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

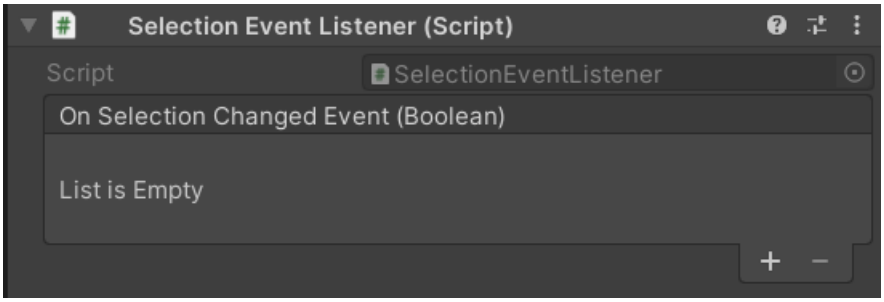
Usually auto navigatoin does a fine job, but sometimes it is better to set the selection by hand. The simple approach would be to use "explicit" mode instead. Sadly in explicit mode the selection will just do nothing if the target is disabled.

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 the as explicit values which fall back on auto select if the specified target can not be selected.



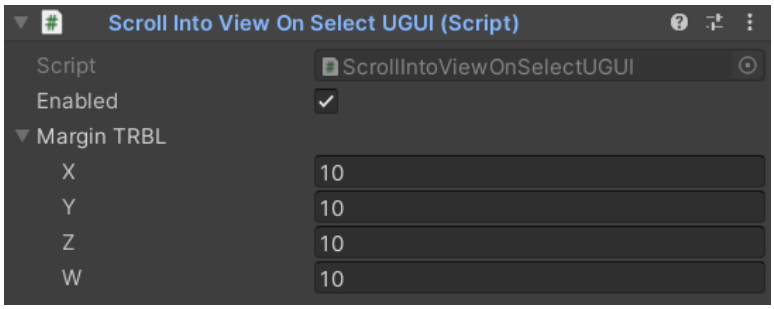
Selection Event Listener

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.

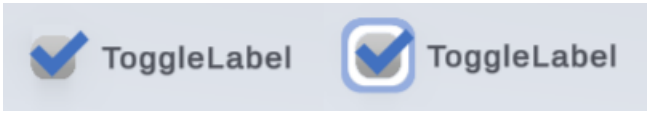


SelectionUGUI

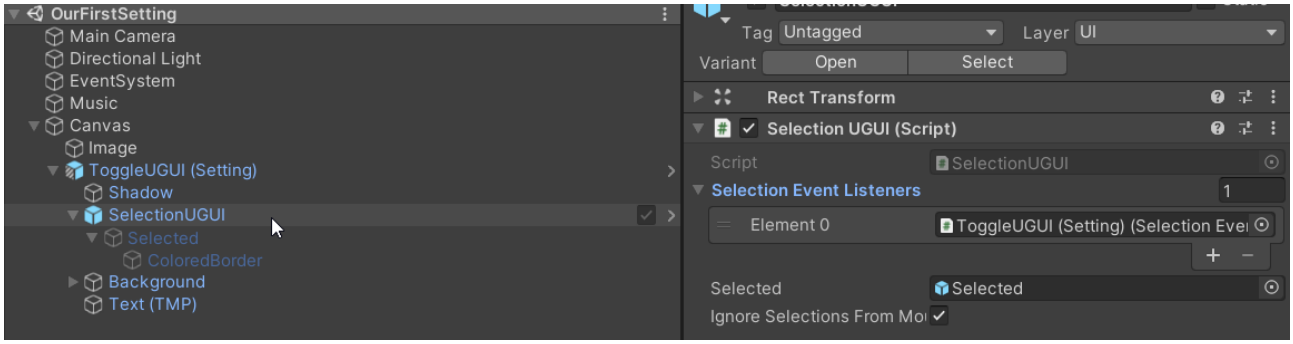
You will find a SelectionUGUI object within all UGUI component prefabs. These are the selection highlights for controller & gamepd input.

They look like this:

not selected selected (notice the border)



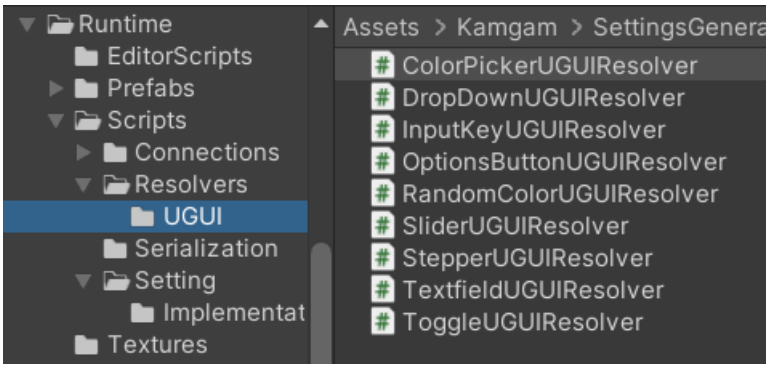
You can customize them for all elements by editing the „SelectionUGUI“ prefab.



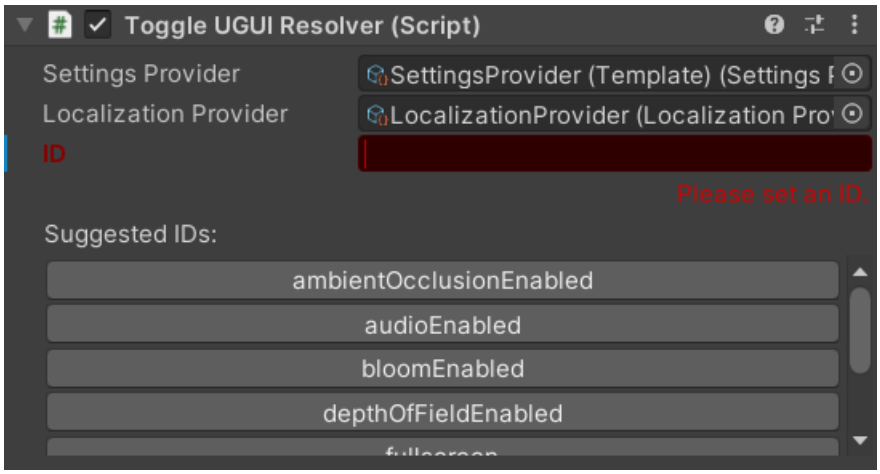
Setting Resolvers

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/UGUI"). 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:

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

Setting

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

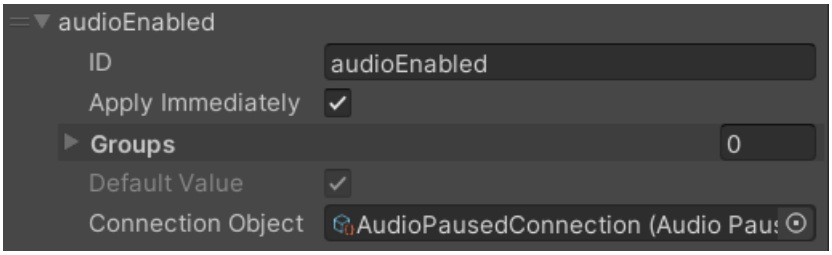
ID:
Each settings has a unique ID which identifies it globally.

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.

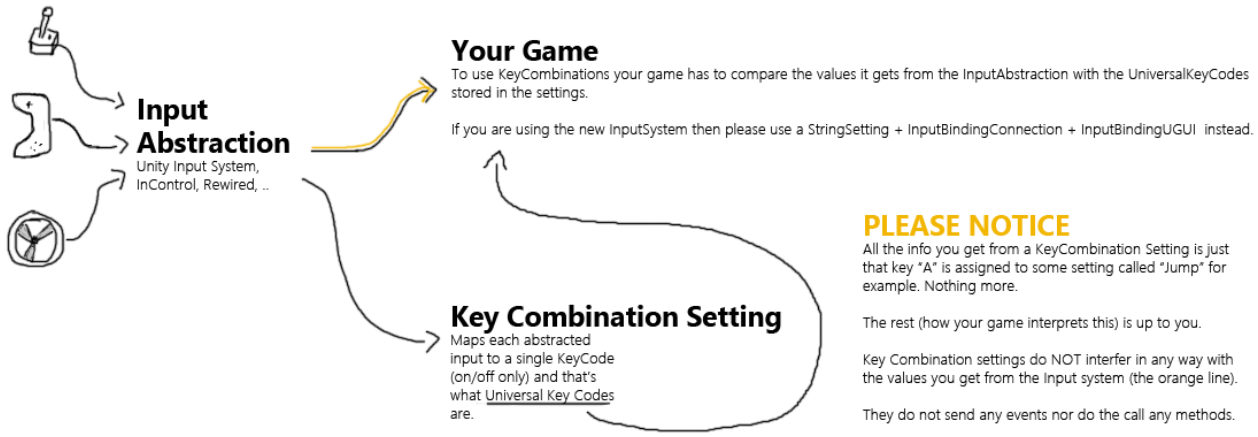
Options

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.



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.

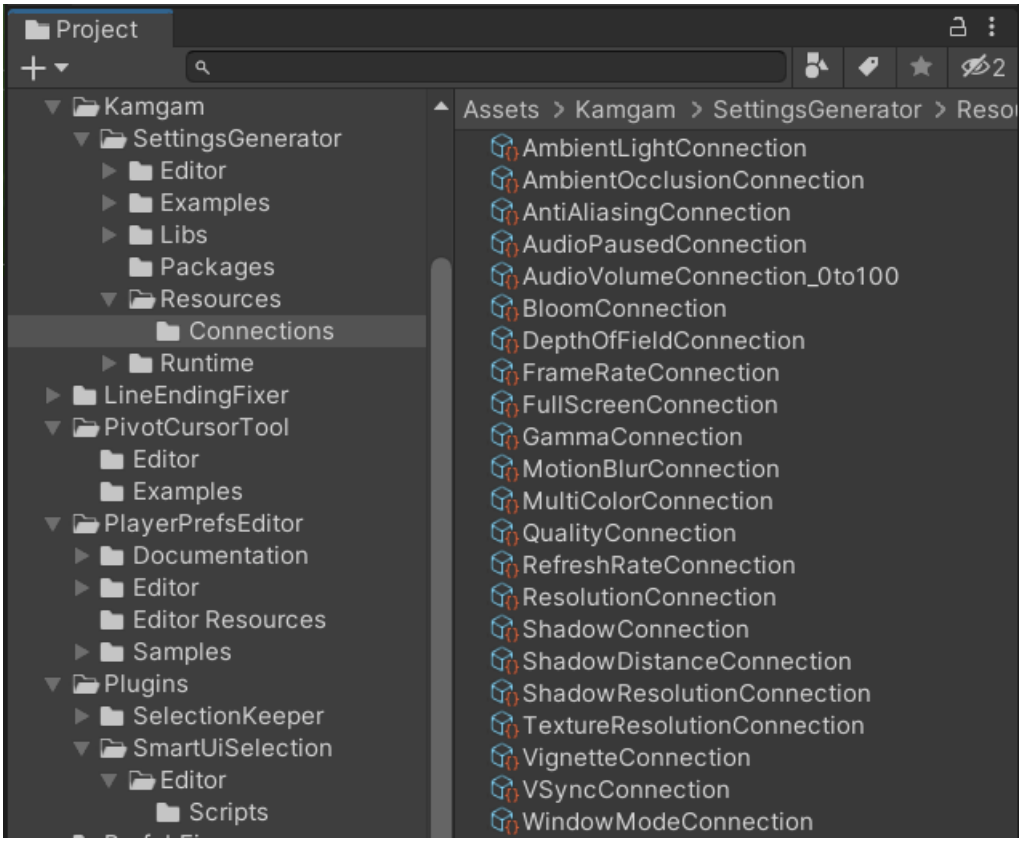
The **Input Rebinding** section contains a detailed explanation on how to do that.

You can find a demo of this under Asset/Kamgam/SettingsGenerator/InputSystemBinding.

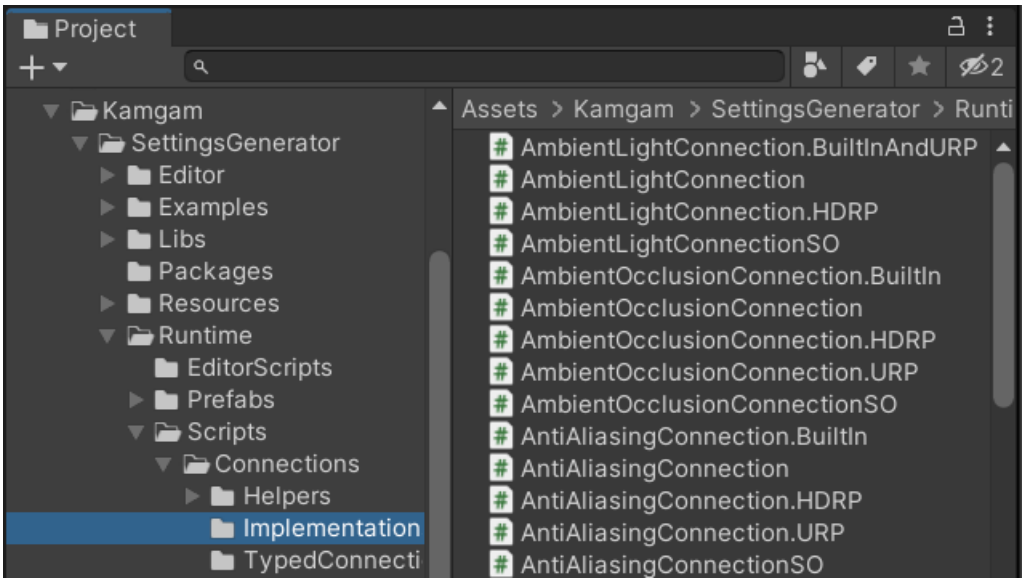
Connections

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.

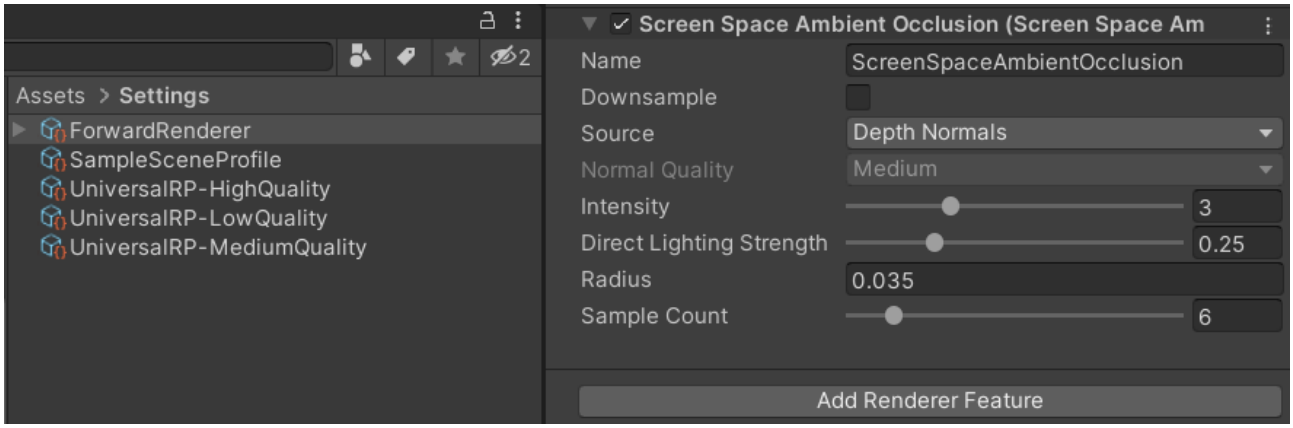
Ambient Occlusion (SSAO)

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 Paused

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

Audio Volume

Controls the Audio Listener volume (useful for global audio volume).

Audio Source Volume

Controls the volume of one or more specific AudioSource Components.

Bloom

On/Off for the bloom post processing effect.

Depth Of Field

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

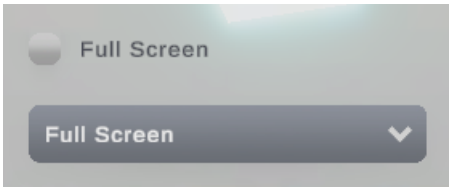
Frame Rate

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.

Resolution

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

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.

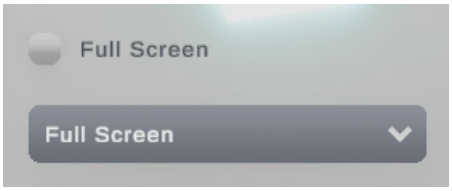
V-Sync

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:



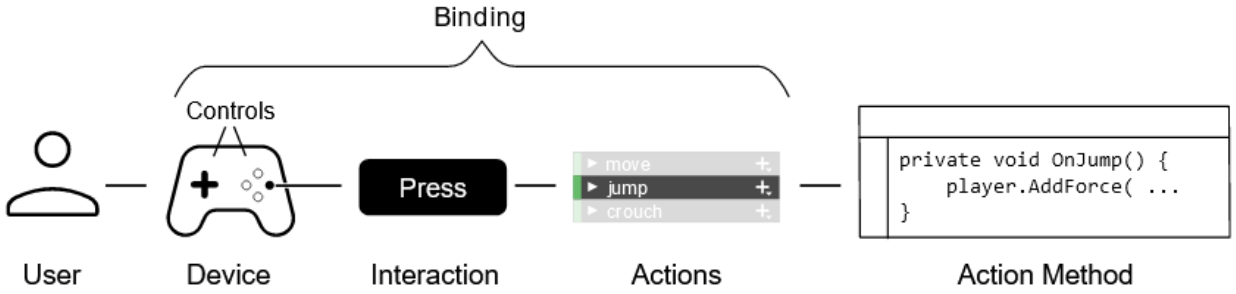
Is fullscreen now ON or OFF?

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.



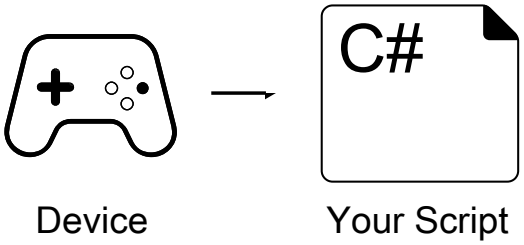
Source: https://docs.unity3d.com/Packages/com.unity.inputsystem@1.5/manual/Concepts.html

There are four major ways of how your game can interact with the InputSystem.
From the Unity Manual:

Directly Reading Device States

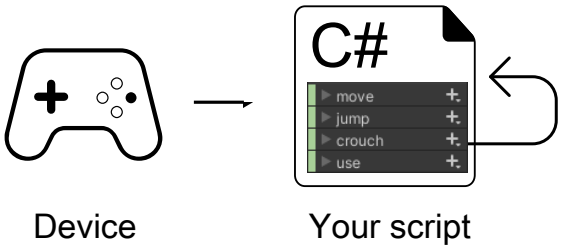
Your script explicitly refers to device controls and reads the values directly.

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



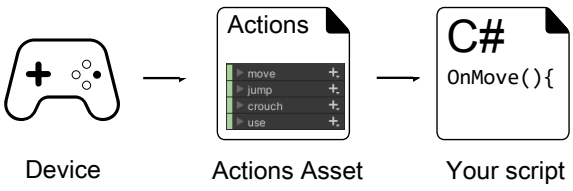
Using Embedded Actions

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



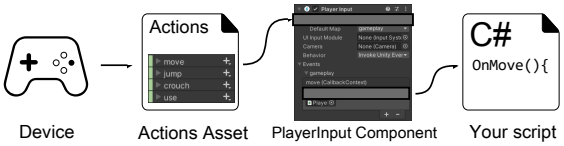
Using an Actions Asset

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 progammer then you can forward the binding path via the AddChangeListener() API of the connection.

Rebinding Tutorial

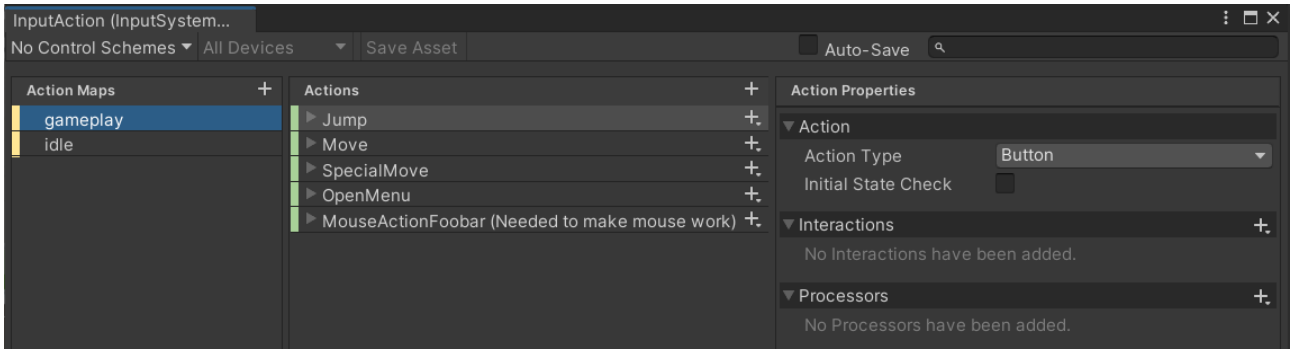
The Settings System needs to know which Input Actions (Bindings) do exist so it can connect the Settings and UI with it.

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

If you want to skip to the end then you can find a complete sample under: **Assets/Kamgam/SettingsGenerator/Examples/InputSystemBinding**.

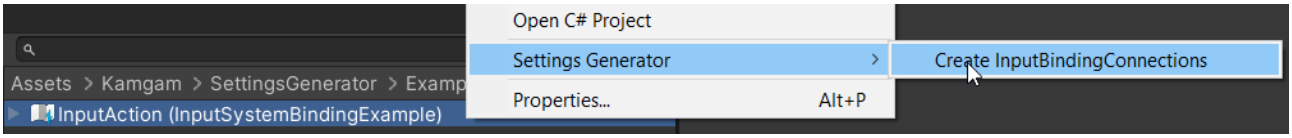
1) Generating the InputBinding Connections

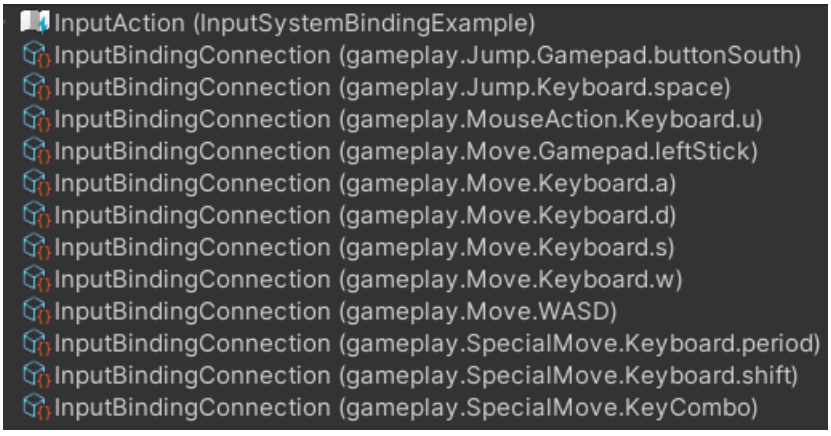
Let's assume you have an InputActions Asset like this:



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).**

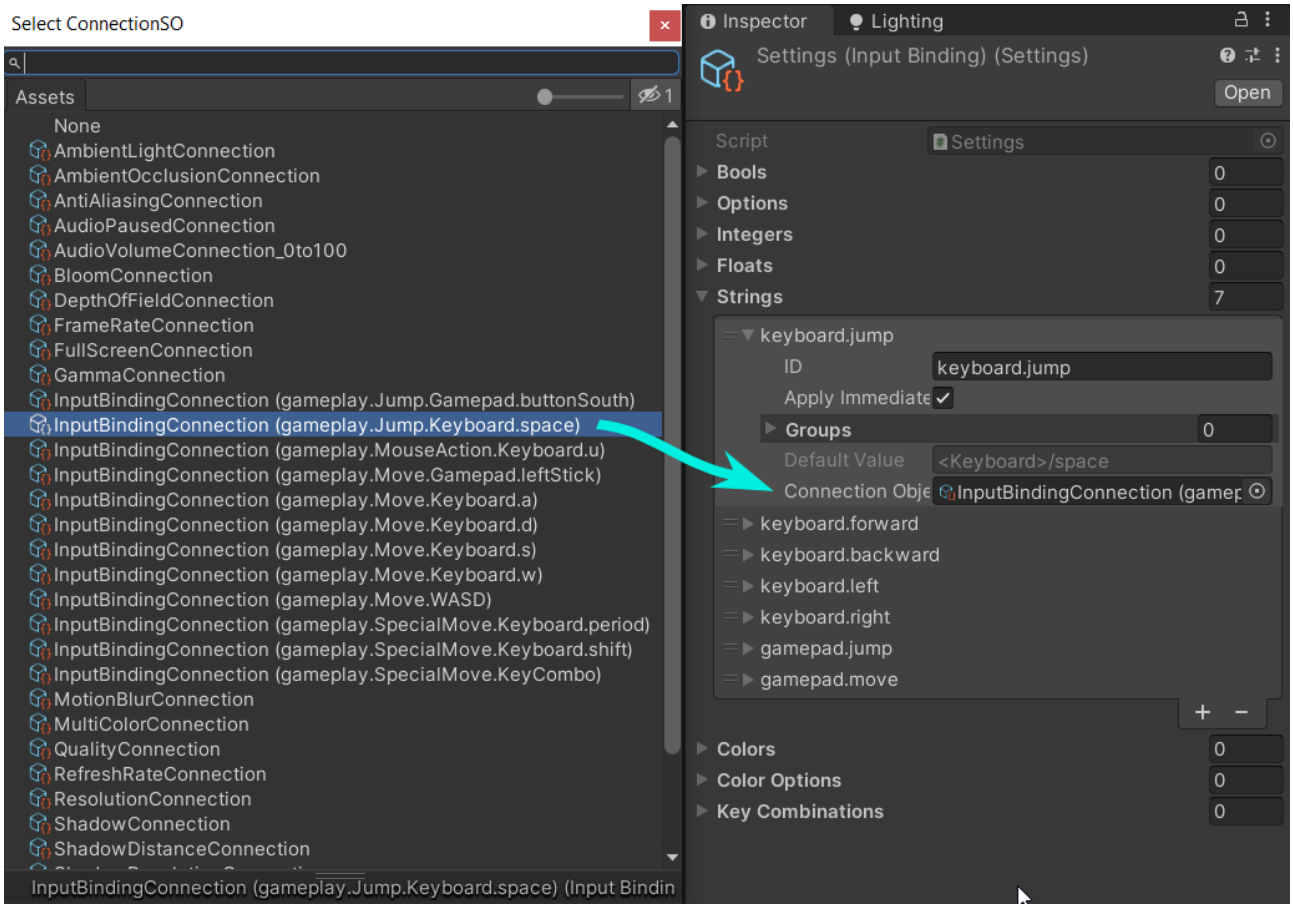




Info: Each Connection is linked to a Binding via two things. A) the Actions Asset and B) the Binding GUID (see image on the right). Usually you can just leave the connections objects alone after creation. However if needed you can edit the linked GUID manually or via the „Available Bindings" buttons. 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.

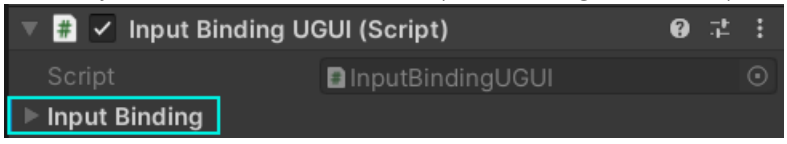


It's a bit cumbersome but you will only have to do this once (as long as you don't delete your connection objects).

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:

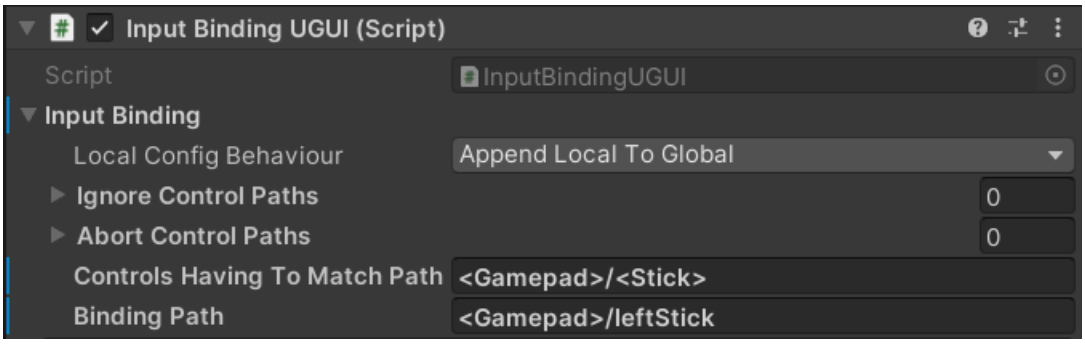
3) Configuring the Rebinding UI

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

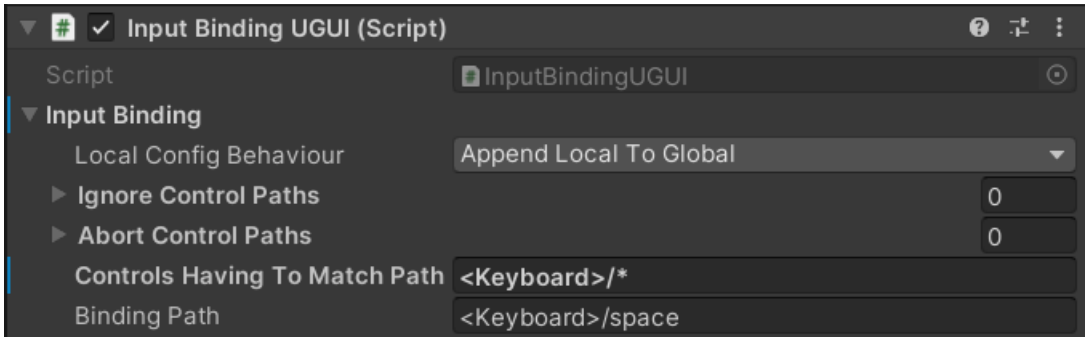


If you open it you will see some configuration options. These are used to LIMIT the inputs this UI will react to. This is useful for example to limit a MOVE action for a controller to only take Thumb Sticks and ignore buttons („<Gamepad>/<Stick>").

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 envents

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.

Scripting API (Adding custom Settings)

There are two ways to connect a setting to your custom code. Let's call them A and B.

A: Custom Code Tutorial (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 interal
// 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 to 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);
}
```

B: Custom Code Tutorial (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)
```

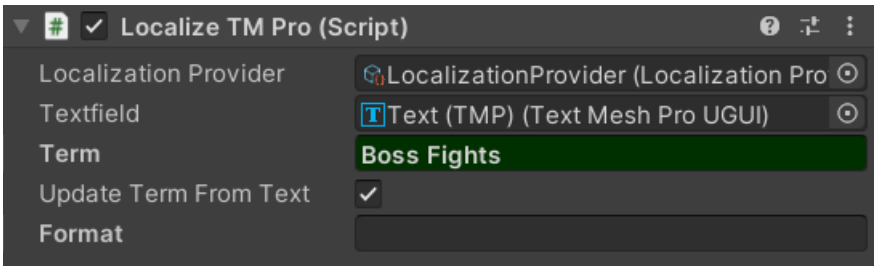
```
                Create();

            return _connection;
        }

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

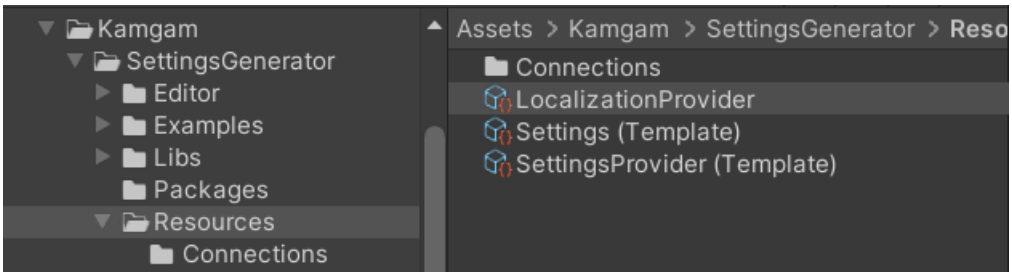
Now you can create an SO Asset and use it for any setting matching the type (bool in this case).

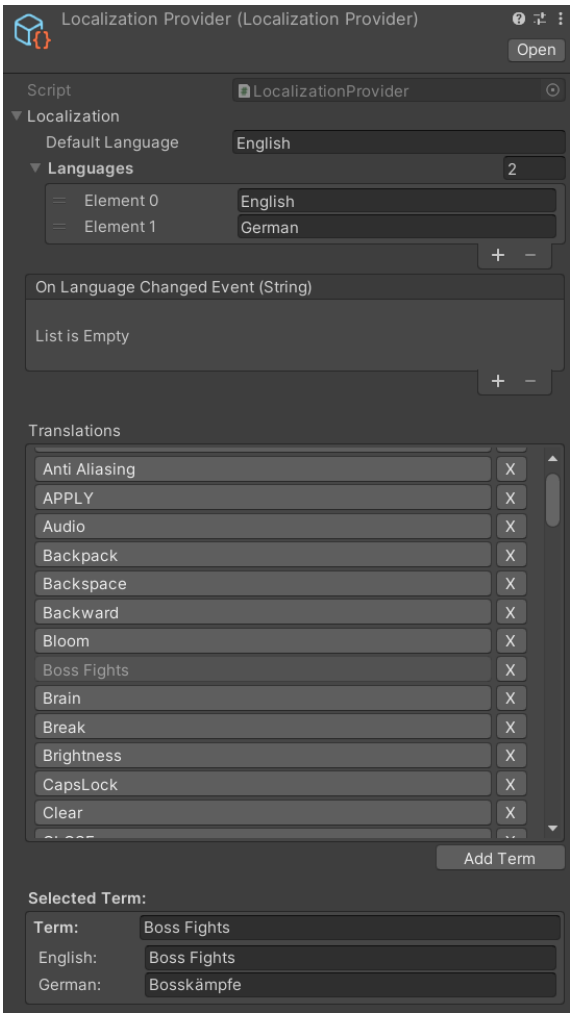
If you are not a coder then maybe you are able to do it with the VisualScripting package and the GetSetConnection, though it's cumbersome compared to the code solution.

Localization

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







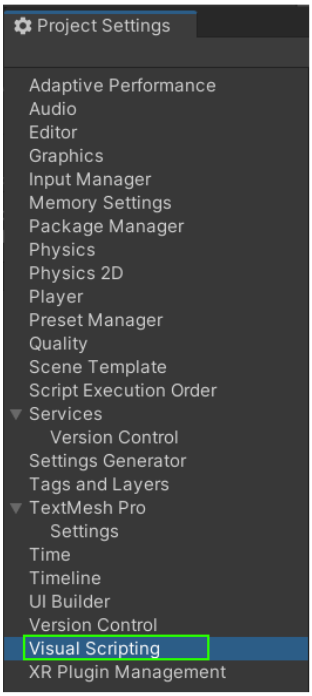
It is a fairly rudimentary localization system but for the settings it is sufficient. You can easily replace it with some dedicated localization system like I2 Localization (see „ILocalization.SetDynamicLocalizationCallback“)

Visual Scripting (formerly BOLT)

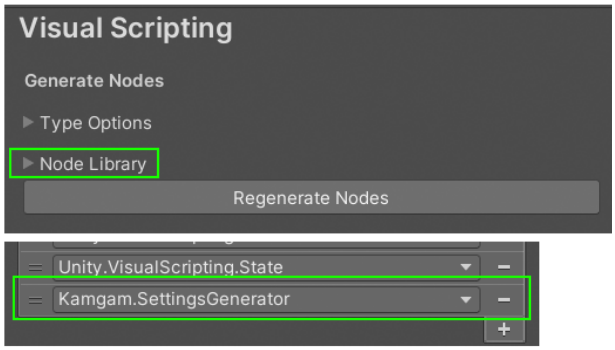
Visual Scripting requires a manual setup (let BOLT know the settings exist). Don't worry this will take just one minute.

Here are the steps:

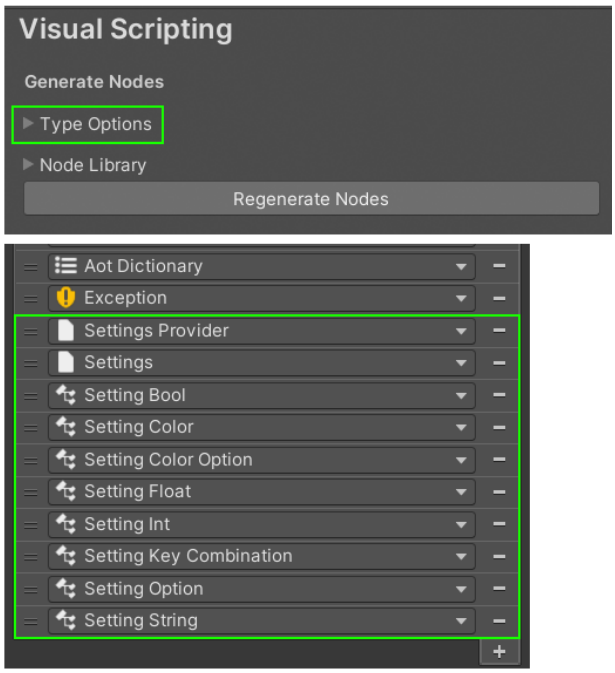
Go to **Edit > Project Settings > Visual Scripting**:
1) Add **Kamgam.SettingsGenerator** to the "**Node Library**"
2) Add the Settings Types (**SettingsProvider**, **Settings**, all the **Setting..**) to the "**Type Options**"
3) Hit "**Regenerate Nodes**" to apply the changes.



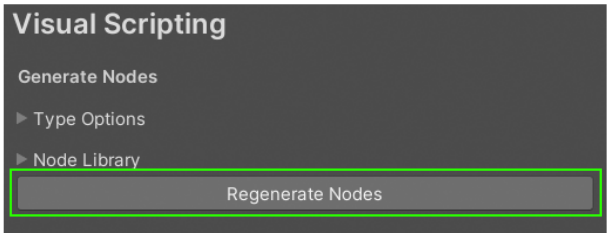
Step 1



Step 2

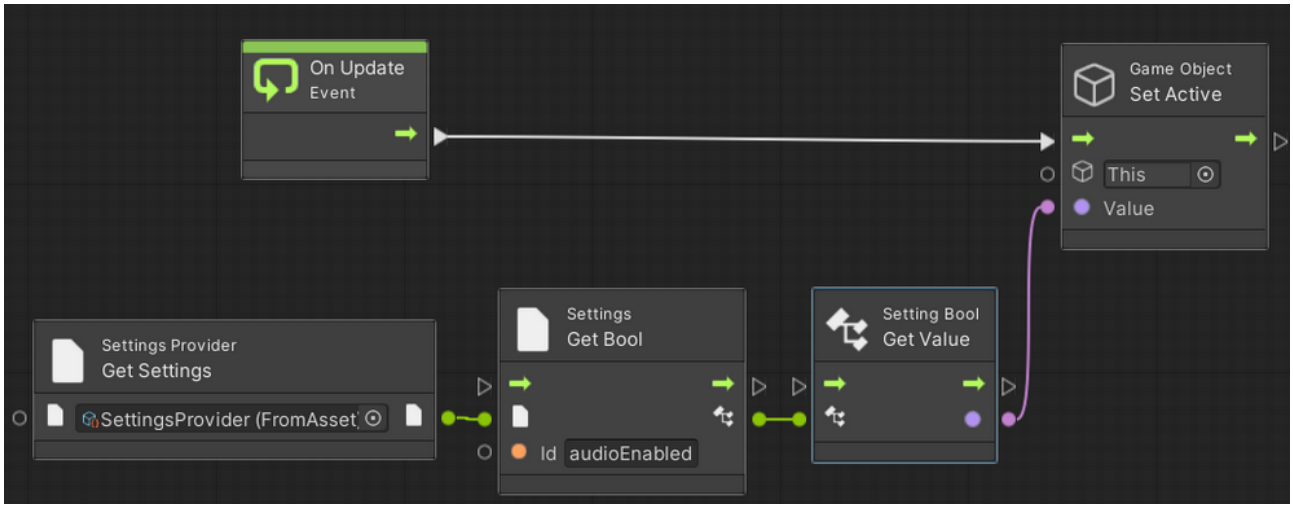


Step 3



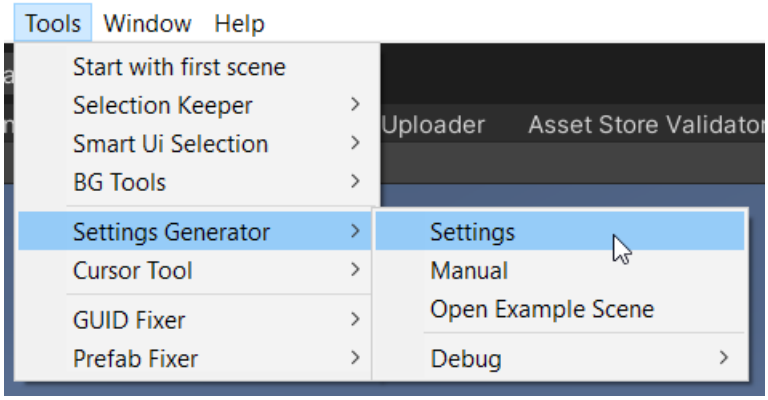
You can then retrieve any setting via its ID.

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

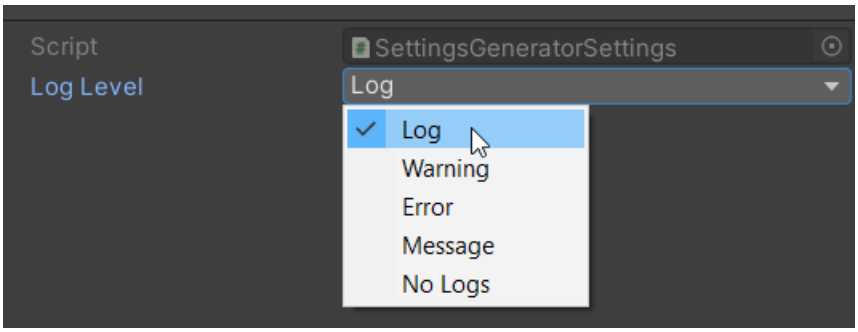


Configuration & Debugging

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.



Common Issues

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)

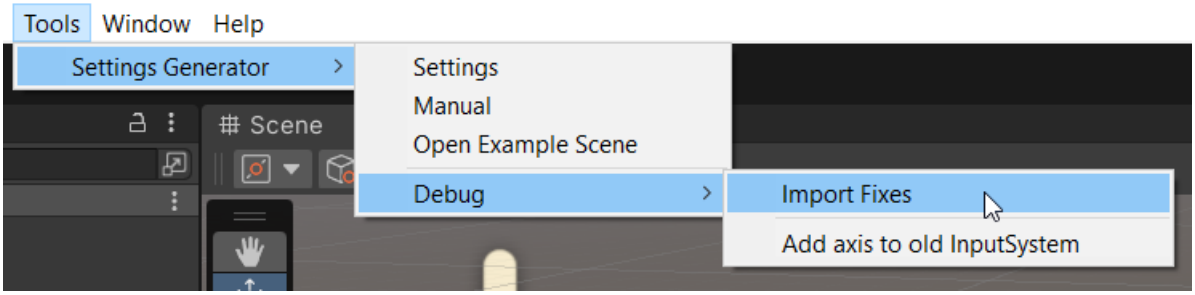
Controller support for the OLD input system is based on the standard xbox controller layout. I encourage you to use the NEW InputSystem which has a much better controller abstraction included. However, if you are fixed to the old system then I'd recommend using InControl for controller abstraction.

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 it does not do it after an update.

You will have to trigger the import manually by calling:
Tools > Settings Generator > Debug > Import Fixes

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.