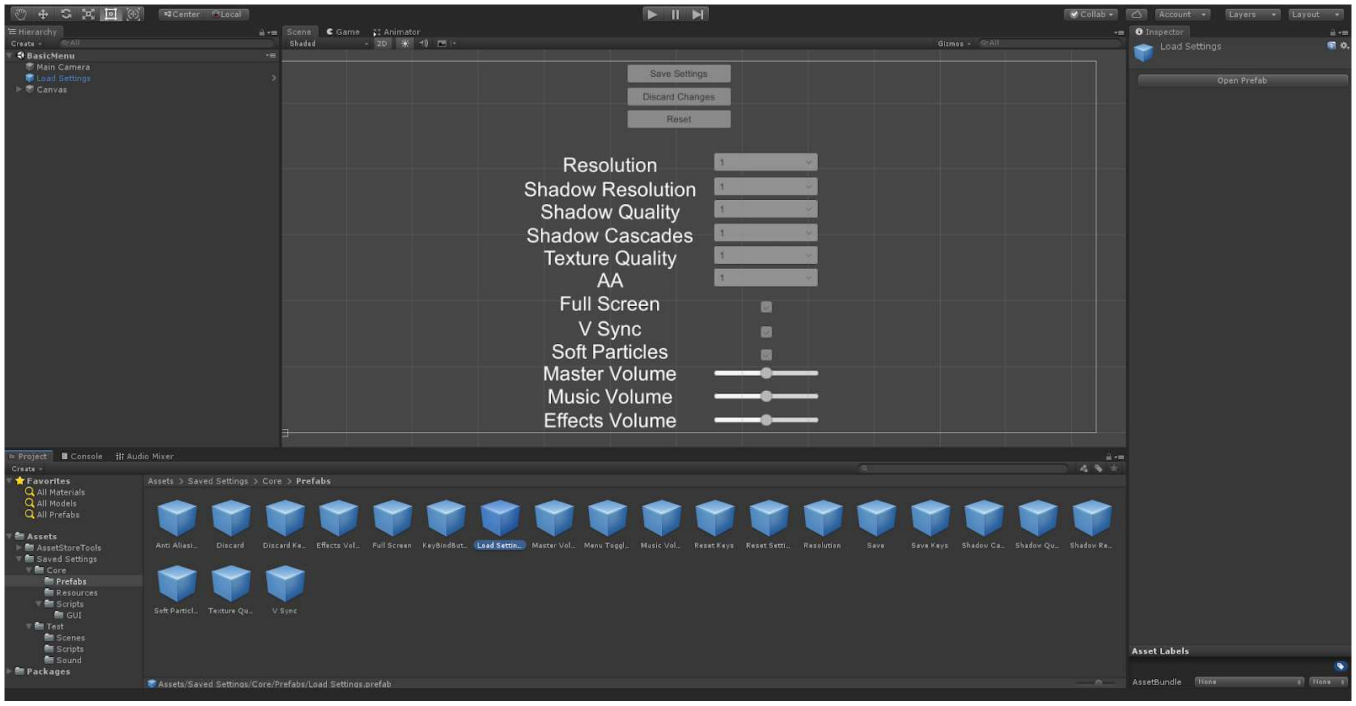
For Unity 2018.3.6f1+
Current Version 1.0
Support Email

Table of Contents

Quick Setup

Step 1 - Loading Your Settings

To load all the settings for you game, all you have to do is add the LoadSettings prefab to the first scene of your game. You can find the list of prefabs under SaveSettings/Core/Prefabs.

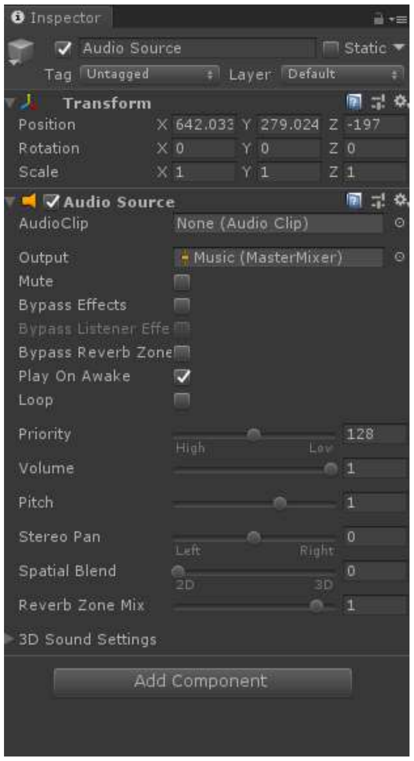


Step 2 - Basic Menu For Changing Settings

All of the settings in the above image have prefabs you can find under SaveSettings/Core/Prefabs. You can simply drag and drop them into the menu for them to start working. Then you can alter their appearance and animations as normal through unity.

Master, Music, and Effect Volume Sliders

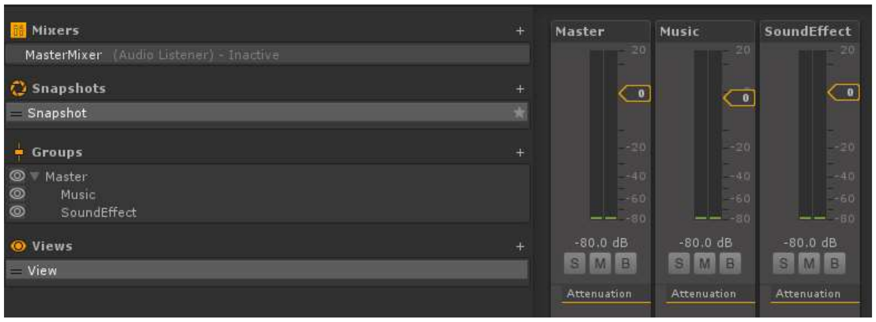
After setting up any of the volume sliders, when you create a new audio source in your scene, click on the “output” area of the audio source in the inspector and fill it with the corresponding audio group. This will cause changes to the audio group to affect that audio source. You need to do this in order for the volume sliders to work.



```
Inspector
Audio Source                         Static
Tag  Untagged        Layer  Default
Transform
Position     X 642.033  Y 279.024  Z -197
Rotation     X 0        Y 0        Z 0
Scale        X 1        Y 1        Z 1
Audio Source
AudioClip              None (Audio Clip)
Output                 Music (MasterMixer)
Mute
Bypass Effects
Bypass Listener Effe
Bypass Reverb Zone
Play On Awake          ✓
Loop
Priority               High   Low    128
Volume                               1
Pitch                                1
Stereo Pan             Left   Right  0
Spatial Blend          2D     3D     0
Reverb Zone Mix                      1
3D Sound Settings
Add Component
```

Make Your Own Audio Groups

These audio groups belong to a MasterMixer located in SaveSettings/Core/Resources. You can create your own audio groups as children of "Music" or "SoundEffect" and the volume sliders will affect them as well.

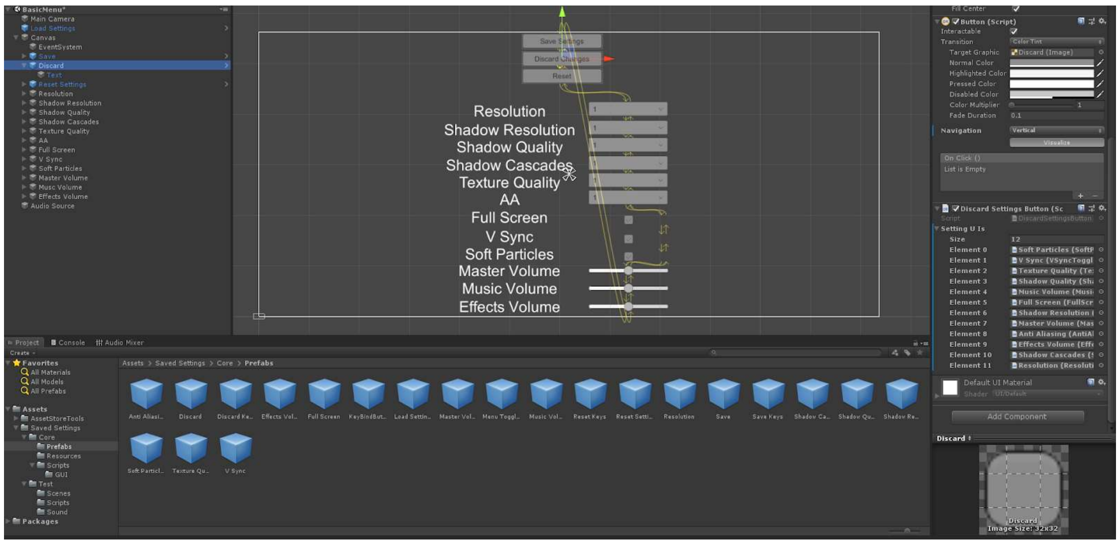


```
Mixers
  MasterMixer  (Audio Listener) - Inactive
Snapshots
  Snapshot
Groups
  Master
    Music
    SoundEffect
Views
  View

Master        Music         SoundEffect
-80.0 dB      -80.0 dB      -80.0 dB
S M B         S M B         S M B
Attenuation   Attenuation   Attenuation
```

Step 3 - Discarding, Saving, Resetting

For the graphics or volume settings, add the SaveSettingsButton, DiscardSettingsButton, and/or ResetSettingsButton to your menu. When clicked these buttons do as they specify. You can find prefabs for them under SavedSettings/Core/Prefabs (or you can just add the component to your current menu). For keybindings, instead use SaveKeysButton, DiscardKeysButton, or ResetKeysButton.

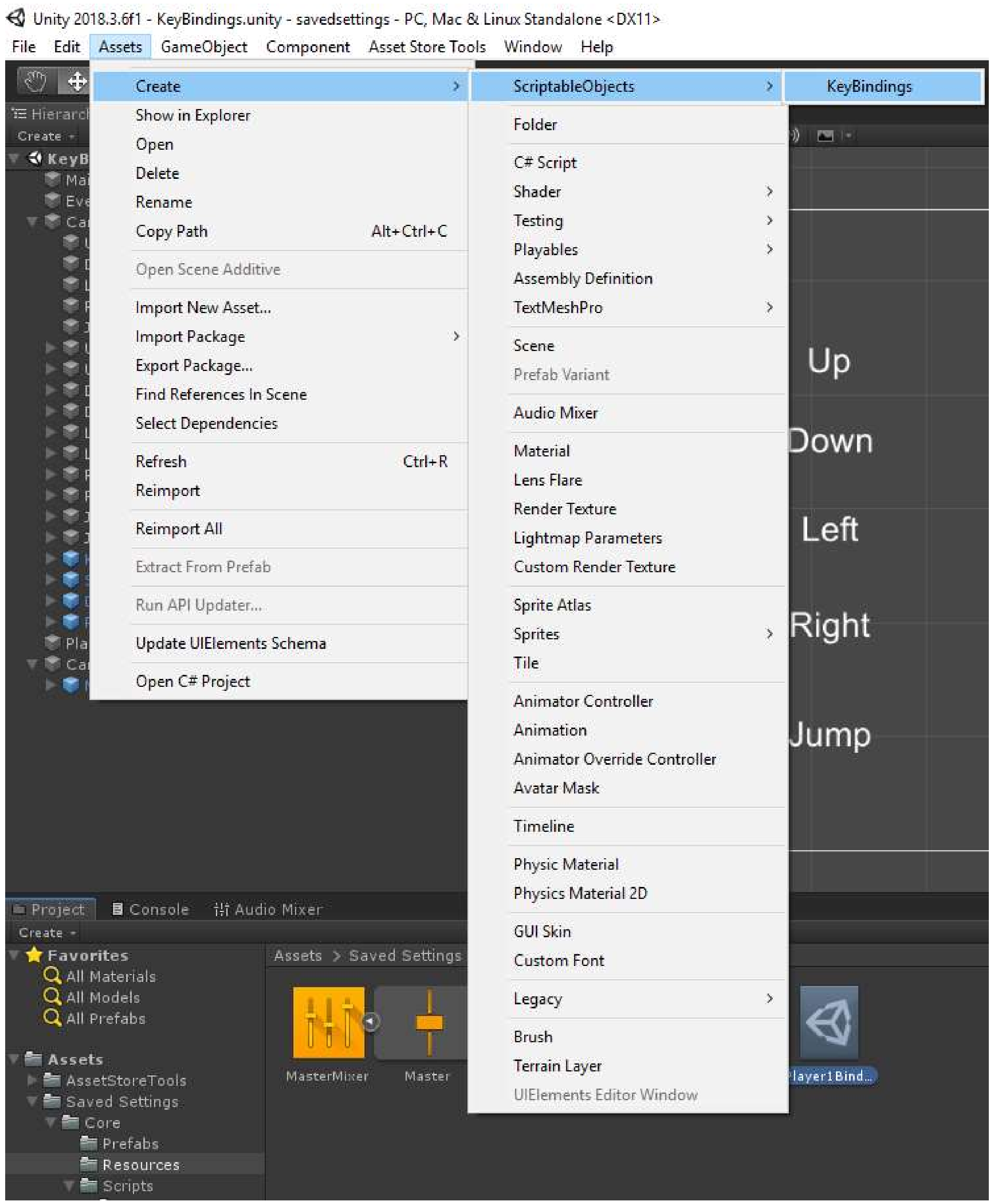
For the discard and reset buttons you must populate their list of UI elements in the inspector with the other settings buttons in the scene. This is so that when the settings are discarded/reset, the UI element will reload its visual to match.



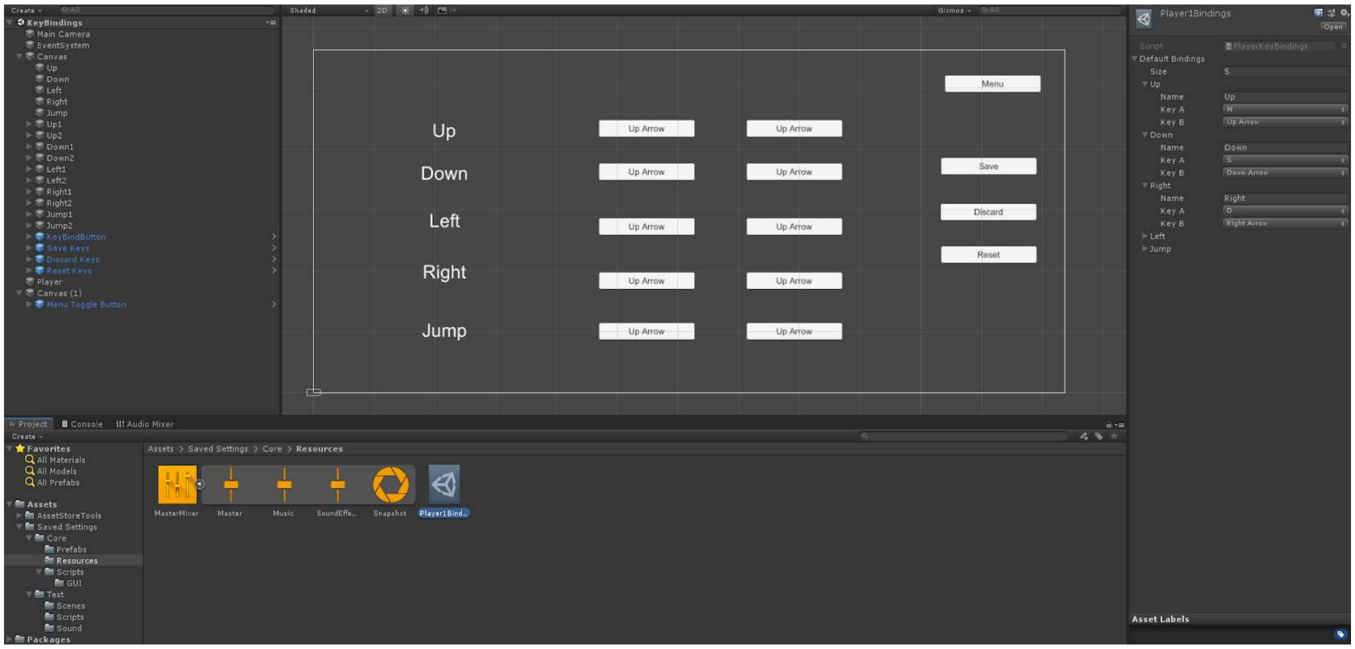
```
BasicMenu*
  Main Camera
  Load Settings
  Canvas
    EventSystem
    Save
    Discard
      Text
    Reset Settings
    Resolution
    Shadow Resolution
    Shadow Quality
    Shadow Cascades
    Texture Quality
    AA
    Full Screen
    V Sync
    Soft Particles
    Master Volume
    Musc Volume
    Effects Volume
  Audio Source

Button (Script)
Interactable         ✓
Transition           Color Tint
  Target Graphic     Discard (Image)
  Fade Duration      0.1
Navigation           Vertical
On Click ()
List is Empty

Discard Settings Button (Sc
Script               DiscardSettingsButton
Setting U Is
  Size               12
  Element 0          Soft Particles (SoftP
  Element 1          V Sync (VSyncToggl
  Element 2          Texture Quality (Te
  Element 3          Shadow Quality (Sh
  Element 4          Music Volume (Musi
  Element 5          Full Screen (FullScr
  Element 6          Shadow Resolution (
  Element 7          Master Volume (Mas
  Element 8          Anti Aliasing (AntiAl
  Element 9          Effects Volume (Effe
  Element 10         Shadow Cascades (S
  Element 11         Resolution (Resoluti
Default UI Material
Shader  UI/Default
Add Component

Assets > Saved Settings > Core > Prefabs
```

Key Bindings

You can create a PlayerKeyBinding scriptable object from Assets/Create/ScriptableObjects/KeyBindings. Please see scriptable objects unity documentation for a better idea as to how this works.

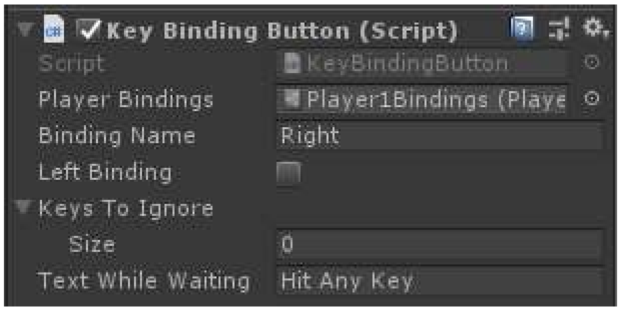


Once you have a keybinding created, the default keybindings are changeable through the inspector.



```
Player1Bindings
Script              PlayerKeyBindings
▼ Default Bindings
    Size            5
  ▼ Up
      Name          Up
      Key A         W
      Key B         Up Arrow
  ▼ Down
      Name          Down
      Key A         S
      Key B         Down Arrow
  ▼ Right
      Name          Right
      Key A         D
      Key B         Right Arrow
  ► Left
  ► Jump
```

For a player to change these bindings, they will need to use a KeyBindingButton in your menu, which can be found as a prefab under SavedSettings/Core/Prefabs Make sure the KeyBindingButton is setup with a reference to your player key bindings scriptable object, and has the correct binding name. Because each individual keybinding includes 2 keys, here you can toggle “Left Binding” to determine which key of the binding is being changed. You can also setup keys that the binding button will ignore. By default the button is setup to ignore Escape, Return, Enter, Backspace, and Menu.



```
Key Binding Button (Script)
Script              KeyBindingButton
Player Bindings     Player1Bindings (Playe
Binding Name        Right
Left Binding        ☐
▼ Keys To Ignore
    Size            0
Text While Waiting  Hit Any Key
```

You can find the corresponding DiscardKeysButton, SaveKeysButton, and ResetKeysButton for your menu under SavedSettings/Core/Prefabs

For your movement code, here is an example of how to use the keybindings to get input. You can find this code in SavedSettings/Test/Scripts/TestKeysMove.cs and see it in action in SavedSettings/Test/Scenes/KeyBindings

You will need a reference to the Keybindings scriptable object you’re getting input from, then get the bindings from its Keys dictionary. The bindings come with small helper functions such as Down, Up, Held, and AxisInput.

```
void Update()
{
    transform.position += new Vector3(  _Bindings.Keys["Right"].AxisInput(_Bindings.Keys["Left"]),
                                        _Bindings.Keys["Up"].AxisInput(_Bindings.Keys["Down"]),
                                        0f)
                                        * Time.deltaTime;

        if (_Bindings.Keys["Jump"].Down)
        {
                Debug.Log("Jump Pressed");
        }
}
```

Making Your Own Settings

For custom settings, such as turning on and off specular light, ambient occlusion, lightbloom, adding joystick sensitivity, etc...Please see the last section at the end of this documentation.

Summary

Saved Settings is component-based, and the functionality is separated into 3 sections.

Helpers

CrytoHelper

AES 256-bit crytopgraphy helper class. SaveHelper uses these helper functions to scramble the text written in savefiles. If you are in the editor SaveHelper will not use CrytoHelper so you can still read the save file in the appData folder.

PlayerKeyBindings

It contains a dictionary of keybindings that are loaded on startup.
The default keybindings are changeable through the inspector.
You can create a PlayerKeyBinding from Assets/Create/ScriptableObjects/KeyBindings.
You can access these keybindings using (referenceToThePlayerKeyBindings).Keys["Right"]
When the player finishes setting the keys through the menu call Save(), or call Load() to discard any changes.
Please see scriptable objects unity documentation for a better idea as to how this works.
If you add/remove keybindings after your full game releases you have to update the game's version number through unity and handle the different versions of keybindings when Load() is called. This is due to the nature of loading and converting data from older versions. When data is saved, it’s saved with the game version number attached to the data.

ReSelect

Attach to a gameobject with an event system. If the currently selected gameobject for this event system is null, reselects the lasts selected game object. This serves as a simple fix for a control setup which both allows for a mouse and controller, in which a mouse could deselect everything on the menu.

SaveHelper

Acts as a helper class with public static functions for saving files to the game's appData folder. If attached to a gameobject in the scene, can be used to save and load scene gameobjects.

SceneHelper

Contains public static helper functions for changing scenes.

SettingsHelper

Public static helper class for changing common settings in games. Attach to an object in the first scene of your game to load the settings when the game loads. Once you have changed a setting, you can save it so it's loaded next session by calling SettingsHelper.SaveSettings(). Note that not all settings are available on all platforms. You can't change fullscreen/resolution/vSync on console or IOS.

UI Elements

BaseUILoadKey

The base class of any UI element that handles key bindings.

BaseUILoadSetting

The base class of any UI element that handles graphics or sound settings.

The following are settings that are synced by the button, dropdown, slider, or toggle button listed in its name.

AntiAliasingDropdown
AutoScrollDropdown
CanvasButton
EffectsVolumeSlider
FullScreenToggle
KeyBindingButton
LoadFinalSceneButton
LoadSceneButton
MasterVolumeSlider
MusicVolumeSlider
QuitButton

ReloadSceneButton
ResolutionDropdown
ShadowCascadesDropdown
ShadowQualityDropdown
ShadowResolutionDropdown
SoftParticlesToggle
TextureQualityDropdown
VSyncToggle

Discarding, Resetting, and Saving

The following buttons are setup to discard, reset, or save the settings. For the discard and reset buttons you must populate their list of UI elements in the inspector with the other settings buttons in the scene. This is so that when the settings are discarded/reset, the UI element will reload its visual to match.

DiscardKeysButton
DiscardSettingsButton
ResetKeysButton
ResetSettingsButton
SaveKeysButton
SaveSettingsButton

Misc

SimpleSavedTransform

A simple example class for using the SaveHelper in a scene to save/load a gameobject's position and rotation.

JsonArray

Helper class that allows you to save arrays of objects to a json format.

JsonDictionary

Helper class that allows you to save dictionaries to a json format.

Scripting References

Helper Classes Scripting Reference

CryptoHelper

Public static class Cryptography.CryptoHelper

Summary

AES 256-bit crytopgraphy helper class. SaveHelper uses these helper functions to scramble the text written in savefiles. If you are in the editor SaveHelper will not use CrytoHelper so you can still read the save file in the appData folder.

Public Properties

_salt	byte[]	Salt used for the encoding.
_initVector	byte[]	Initial vector used for encoding.

Public Methods

Encrypt(string toEncrypt, string password = "DefaultPassword")	string	Encrypts a string using AES.
Decrypt(string encryptedText, string password = "DefaultPassword")	string	Decrypts a string using AES.

PlayerKeyBindings

Public class PlayerKeyBindings : Monobehavior

Summary

It contains a dictionary of keybindings that are loaded on startup.
The default keybindings are changeable through the inspector.
You can create a PlayerKeyBinding from Assets/Create/ScriptableObjects/KeyBindings.
You can access these keybindings using (referenceToThePlayerKeyBindings).Keys["Right"]
When the player finishes setting the keys through the menu call Save(), or call Load() to discard any changes.
Please see scriptable objects unity documentation for a better idea as to how this works.

If you add/remove keybindings after your full game releases you have to update the player key bindings version number through the inspector and handle the different versions of keybindings when Load() is called. This is due to the nature of loading and converting data from older versions. When data is saved, it's saved with the game version number attached to the data.

Public Properties

Keys	Dictionary<string, KeyBinding>	The keys you save are kept in a dictionary.

Public Methods

Load ()	void	Loads the player key bindings from a saved json file. This also acts as a Discard() function.
Save()	void	Loads the player key bindings from a saved json file.
ResetKeyBindings ()	void	Resets the key bindings to the defaults and saves.

Public Struct KeyBinding Methods

KeyBinding(string aName, KeyCode a, KeyCode b)	KeyBinding	Constructor.
Held ()	bool	If either key is held.
Down ()	bool	If either key has just been pressed down.
Up ()	bool	If either key has just stopped being held.
AxisInput(KeyBinding negative)	float	Gets input on a scale between -1 to 1 using another keybinding as the negative.

ReSelect

Public class ReSelect : Monobehavior

Summary

Attach to a gameobject with an event system. If the currently selected gameobject for this event system is null, reselects the lasts selected game object. This serves as a simple fix for a control setup which both allows for a mouse and controller, in which a mouse could deselect everything on the menu.

No Public Properties or Functions

SaveHelper

Public class SaveHelper : MonoBehaviour

Summary

Acts as a helper class with public static functions for saving files to the game's appData folder.
If attached to a gameobject in the scene, can be used to save and load scene gameobjects.

Public Methods

SaveScene (string subFolderName, float version)	bool	Saves all saveable gameobjects this component has reference to.
LoadScene (string subFolderName)	bool	Loads the scene's savedata and sends it to all local saveable objects this component has reference to.

Public Static Methods

Save(string path, string data, float version)	bool	Saves json data to a file in the game's file folder. The data provided must first be parsed using JsonUtility.ToJson() before being sent. To reset a save file, you can overwrite the file's save data with an empty string. As long as new saves are handled correctly, the save file will be reset when that file is next loaded.
SaveIntoDirectory(string directory, string name, string data, float version)	bool	Checks if a directory exists and creates it if it does not. Then calls Save() - See this function for more information.

Load(string path)	SaveData	Loads a saved file in the game's folder. You will have to use JsonUtility.FromJson() to convert the string to your saved data. Can be used to load files within subfolders if a valid path is given. If the loading fails, an empty string and version 0 are returned.
DeleteSceneFile(string subFolderName)	bool	Delete's a scene save in a given subfolder.
DeleteFile(string path)	bool	Delete's the given file.
DeleteFolder(string path)	bool	Delete's the given folder and all files within it.
CopyFile (string fromPath, string toPath)	bool	Creates a copy of a file at a given path.
CopyFolder(string fromPath, string toPath)	bool	Creates a copy of a folder at a given path and copies all the files within.

Public Struct SaveData

string data	Save data to be converted from json.
float version	A version set by you that will remain with the saved data when loaded.

Public Abstract Class Saved : MonoBehaviour

Abstract class for objects that want to saved when the scene saves.

Save()	string	Saved the game object's data. You must use JsonUtility.ToJson() to convert your data to a string and return it.
Load(string data, float version)	void	Loads the game object's data. You must use JsonUtility.FromJson() to convert the string to your loadable data.

SceneHelper

Public static class SceneHelper

Summary

Contains public static helper functions for changing scenes.

Public Methods

Reload()	void	Reloads the current scene.
LoadScene(int level)	void	Loads the given level. Checks if the level exists first.
FinalScene()	void	Loads the final scene in build settings.

SettingsHelper

Public class SettingsHelper : MonoBehaviour

Summary

Helper class for changing common settings in games.
Attach to an object in the first scene of your game to load the settings when the game loads.
Once you have changed a setting, you can save it so it's loaded next session by calling SaveSettings()
Note that not all settings are available on all platforms. You can't change fullscreen/resolution/vSync on console or IOS.

Public Static Properties

MasterVolume	float	The volume for all audio players using the Master Audio Mixer. 0 - 100
MusicVolume	float	The volume for audio players using the Master Audio Mixer's "MusicVolume". 0 - 100
SoundEffectVolume	float	The volume for audio players that use the Master Audio Mixer's "SoundEffectVolume". 0 - 100
TextureQuality	int	Mipmaps

		0 (default) = full size 1 = 1/2 size 2 = 1/4 size 3 = 1/8 size
AntiAliasing	int	Multi-stampling level. Options are: 0, 2, 4, 8
FullScreen	bool	Sets the screen to full screen or windowed.
ResolutionOfWindow	Resolution	Sets the current game window resolution.
VSync	bool	Enables or disables VSync.

ShadowQuality	ShadowQuality	Sets the quality of shadows in the game.
		ShadowQuality.All = soft and hard shadows.
		ShadowQuality.HardOnly = hard shadows only.
		ShadowQuality.Disable = shadows disabled.

ShadowsResolution	ShadowResolution	Sets the resolution of shadows being used. ShadowResolution.Low ShadowResolution.Medium ShadowResolution.Hight ShadowResolution.VeryHigh
ShadowCascades	int	Sets the number of shadow cascades used for shadow mapping. Valid values are 0, 2, and 4 Please see https://docs.unity3d.com/Manual/DirLightShadows.html
SoftParticles	bool	Sets if soft particles are enabled.

Public Methods

DiscardChanges ()	void	Discard's the current settings for the last settings saved.
ResetData ()	void	Resets the current settings to the initial default settings.
Save()	void	Saves the current settings so they are loaded the next time the game is loaded.

UI Elements Scripting Reference

AntiAliasingDropdown

Public class AntiAliasingDropdown : BaseUILoadSetting

Summary

Syncs the drop down with the anti-aliasing setting.

Public Methods

LoadValue ()	void	Called by ResetSettings and DiscardSettings after the settings are changed.

AutoScrollDropDown

Public class AutoScrollDropDown : MonoBehaviour, IPointerEnterHandler, IPointerExitHandler

Summary

Causes a dropdown list to auto-scroll to match keycode/joystick input.

No Public Properties or Methods

BaseUILoadKey

Public abstract class BaseUILoadKey : MonoBehaviour

Summary

Base class used for all UI elements that display key bindings.

Public Methods

LoadValue()	void	Called by ResetKeys and DiscardKeys after the settings are changed.

BaseUILoadSetting

Public abstract class BaseUILoadSetting : MonoBehaviour

Summary

Base class used for all UI elements that display settings.

Public Methods

LoadValue()	void	Called by ResetSettings and DiscardSettings after the settings are changed.

CanvasButton

Public class CanvasButton : MonoBehaviour

Summary

Toggles for a canvas to be enabled or disabled using a button.

No Public Properties or Methods

DiscardKeysButton

Public class DiscardKeysButton : MonoBehaviour

Summary

Syncs the button to discarding the changed key bindings.

No Public Properties or Methods

DiscardSettingsButton

Public class DiscardSettingsButton : MonoBehaviour

Summary

Syncs the button to discarding the changed settings.

No Public Properties or Methods

EffectsVolumeSlider

Public class EffectsVolumeSlider : BaseUILoadSetting

Summary

Syncs the slider value to the sound effects volume.

Public Methods

LoadValue()	void	Called by ResetSettings and DiscardSettings after the settings are changed.

FullScreenToggle

Public class FullScreenToggle : BaseUILoadSetting

Summary

Syncs the toggle value to the full screen setting.

Public Methods

LoadValue()	void	Called by ResetSettings and DiscardSettings after the settings are changed.

KeyBindingButton

Public class KeyBindingButton : BaseUILoadKey

Summary

Syncs the button to changing a player's key bindings.

LoadValue()	void	Called by ResetSettings and DiscardSettings after the settings are

		changed.

LoadFinalSceneButton

Public class LoadFinaSceneButton : MonoBehaviour

Summary

Syncs the button to changing a player's key bindings.

No Public Properties or Methods

LoadSceneButton

Public class LoadSceneButton : MonoBehaviour

Summary

Syncs the button to changing a player's key bindings.

No Public Properties or Methods

MasterVolumeSlider

Public class MasterVolumeSlider : BaseUILoadSetting

Summary

Syncs the slider value to the master volume.

Public Methods

LoadValue()	void	Called by ResetSettings and DiscardSettings after the settings are changed.

MusicVolumeSlider

Public class MusicVolumeSlider : BaseUILoadSetting

Summary

Syncs the slider value to the music volume.

Public Methods

LoadValue()	void	Called by ResetSettings and DiscardSettings after the settings are changed.

QuitButton

Public class QuitButton : MonoBehaviour

Summary

Quits the application when the button is pressed.

No Public Properties or Methods

ReloadSceneButton

Public class ReloadSceneButton : MonoBehaviour

Summary

Syncs the button to changing a player's key bindings.

No Public Properties or Methods

ResetKeysButton

Public class KeyBindingButton : BaseUILoadKey

Summary

When pressed, the button will reset the given key bindings.

No Public Properties or Methods

ResetSettingsButton

Public class ResetSettingsButton : MonoBehaviour

Summary

Syncs the button to resetting the changed settings.

No Public Properties or Methods

ResolutionDropdown

Public class ResolutionDropdown : BaseUILoadSetting

Summary

Syncs the scroll rect with the resolution setting.

Public Methods

LoadValue()	void	Called by ResetSettings and DiscardSettings after the settings are changed.

SaveKeysButton

Public class SaveKeysButton : MonoBehaviour

Summary

Syncs the button to saving the current key bindings.

No Public Properties or Methods

SaveSettingsButton

Public class SaveSettingsButton : MonoBehaviour

Summary

Syncs the button to saving the settings.

No Public Properties or Methods

ShadowCascadesDropdown

Public class ShadowCascadesDropdown : BaseUILoadSetting

Summary

Syncs the drop down with the shadow cascade setting.

Public Methods

LoadValue()	void	Called by ResetSettings and DiscardSettings after the settings are changed.

ShadowQualityDropdown

Public class ShadowQualityDropdown : BaseUILoadSetting

Summary

Syncs the toggle with the soft particles setting.

Public Methods

LoadValue()	void	Called by ResetSettings and DiscardSettings after the settings are changed.

ShadowResolutionDropdown

Public class ShadowResolutionDropdown : BaseUILoadSetting

Summary

Syncs the toggle with the soft particles setting.

Public Methods

LoadValue()	void	Called by ResetSettings and DiscardSettings after the settings are changed.

SoftParticlesToggle

Public class SoftParticlesToggle : BaseUILoadSetting

Summary

Syncs the toggle with the soft particles setting.

Public Methods

LoadValue()	void	Called by ResetSettings and DiscardSettings after the settings are changed.

TextureQualityDropdown

Public class TextureQualityDropDown : BaseUILoadSetting

Summary

Syncs the drop down with the master texture limit setting.

Public Methods

LoadValue()	void	Called by ResetSettings and DiscardSettings after the settings are changed.

VSyncToggle

Public class VSyncToggle : BaseUILoadSetting

Summary

Syncs the toggle value to the vsync setting.

Public Methods

LoadValue()	void	Called by ResetSettings and DiscardSettings after the settings are changed.

Misc Scripting Reference

SimpleSavedTransform

Public class SimpleSavedTransform : Saved

Summary

A simple example class for using the SaveHelper in a scene to save/load a game object' position and rotation.

Public Methods

Save ()	string	Saved the game object's data. You must use JsonUtility.ToJson() to convert your data to a string and return it.
Load(string data, float version)	void	Loads the game object's data. You must use JsonUtility.FromJson() to convert the string to your loadable data.

JsonArray

Public static class JsonArray

Summary

Helper class that allows you to save arrays of objects to a json format.

Public Methods

arrayToJson<T>(T[] array)	string	Converts an array to a json string. If the conversion fails, null is returned.
jsonToArray<T>(string json)	T[]	Converts a json string to an array. If the conversion fails, null is returned.

JsonDictionary

Public static class JsonDictionary

Summary

Helper class that allows you to save dictionaries to a json format.

Public Methods

dictionaryToJson<A, B>(Dictionary<A, B> dictionary)	string	Converts a dictionary to json string. The type's used by the dictionary must be serializable.
jsonToDictionary<A, B>(string json)	Dictionary <A, B>	Converts a json string to a dictionary. The type's used by the dictionary must be serializable.

Make Your Own Settings

Saved Settings is component-based, so you can add and replace parts with ease. It is built in C#.

Let's start with a simple toggle button, I'll using setting the game to full screen as an example.

If you create a UI button in the scene with a new script attached to it, you can add a function from any gameobject in the scene to be called when the toggle value changes either through the inspector, or set it through code when the gameobject loads using
`GetComponent<Toggle>().onValueChanged.AddListener ( someFunction() )`

Then in that function, you can set the value of the setting to the value of the toggle like such.

```
Screen.fullScreen = toggle.isOn;
```

In order for discarding or resetting the settings through the ResetSettingsButton, DiscardSettingsButton, ResetKeysButton, or DiscardKeysButton, you must inherit from the abstract class BaseUILoadSetting and implement the LoadValue() function. You must also drop your new component into the list of BaseUILoadSettings or BaseUILoadKeys of the reset/discard button through the inspector.

The LoadValue function is called by those UI elements after the settings are discarded or reset. This is used to update the UI element to match the new settings value. For example:

```
GetComponent<Toggle>().isOn = Screen.fullScreen;
```

This is what the real FullScreenToggle class included in SavedSettings looks like.

```
using UnityEngine;
using UnityEngine.UI;

public class FullScreenToggle : BaseUILoadSetting
   {
       void Start()
       {
           Toggle toggle = GetComponent<Toggle>();
           LoadValue();
           toggle.onValueChanged.AddListener(delegate { Screen.fullScreen = toggle.isOn; });
       }

       public override void LoadValue()
       {
           GetComponent<Toggle>().isOn = Screen.fullScreen;
       }
   }
```

I’ve tried my best to cover most graphics settings covered by Unity that can be easily applied. There are other settings you likely might want to cover, such as mouse sensitivity for a PC shooter. For the corresponding UI element, the implementation is similar to the above button, but you will use a slider instead (see MusicVolumeSlider’s code for an example). However, the setting itself is new and therefore isn’t saved or loaded for you by adding SaveHelper to the first scene, so you must save and load it yourself. There is a helper class to simplify this called SaveHelper()

In whichever class you need to access/load the Mouse Sensitivity from, you can save the value as such. This assumes that mouseSensitivity is a float so it’s serializable. You can also save a collection of data using a struct if you want to load/save multiple new settings at once. Some things cannot be converted to json, such as whole gameobjects. The final value of the Save() function represents the version of the data you are saving. For now we will set this to 1.

```
SaveHelper.Save("MouseSensitivity", JsonUtility.FromJson<float>(mouseSensitivty), 1f);
```

Now you can load the value like such.

```
SaveData loaded = SaveHelper.Load("MouseSensitivity");
if (string.IsNullOrEmpty(loaded.data))
{
	float mouseSensitivty = JsonUtility.FromJson<float>(loaded.data);

	//Apply the settings to your code

	// SaveData is a struct that includes the version number you provided to Save() (loaded.version)
	// You can use the version number to tell old versions of saved data from the current version.
}
```

That should be everything you need to make a new setting for things in your game.