Settings

Writing New Settings

To write a new setting you need to derive it from the Setting class or a class derived from it. The most commonly used base class is GenericUnitySetting.

GenericUnitySetting

You can specify the setting value type and only need to override the 'ApplySettingChangeWithValue' method as shown in the code example below.

```
using CitrioN.Common;
using CitrioN.SettingsMenuCreator;
using System.ComponentModel;

[MenuPath("My Settings/")]
[DisplayName("Print Me To Console")]
0 references
public class MyCustomSetting : GenericUnitySetting<string>
{
    12 references
    public override string EditorName => "Print To Console";

    3 references
    public override string EditorNamePrefix => "[Custom]";

    5 references
    public override string RuntimeName => "Print To Console Runtime";

    2 references
    public override bool StoreValueInternally => true;

    2 references
    protected override object ApplySettingChangeWithValue(SettingsCollection settings, string value)
    {
        ConsoleLogger.Log(value);
        return value;
    }
}
```

Typically you would apply the value provided as a parameter to wherever you need it and then return back the value so it can be saved internally if enabled for this setting. Here would also be the place to modify the value if needed before returning back the modified value. In the code example are also a few other optional overrides and attributes shown that you can use.

Optional Overrides

MenuPath Attribute
This allows you to specify a path for the advanced dropdown selection when selecting a setting. You can use forward slashes to make subdirectories for better organization.

DisplayName Attribute
Can be used to specify a custom name to be shown in the setting selection dropdown. Without this attribute it will show the class name with ‘Setting’ being removed and spaces added between capitalized letters (similar to how fields are formatted in the inspector based on their field names). In the code example below it would show as ‘My Custom’ without the attribute.TODO

EditorName
Can be used to specify the name shown in the settings list (not the dropdown).

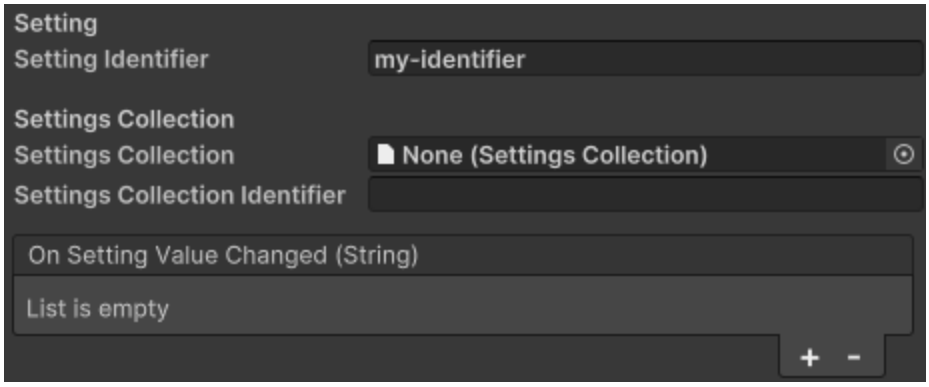
EditorNamePrefix
Can be used to add a prefix to the editor name. Useful for grouping settings to improve readability.

RuntimeName
The display name in your actual runtime menu. If not overridden it will show the class name with ‘Setting’ being removed and spaces added between capitalized letters (similar to how fields are formatted in the inspector based on their field names). If a custom display name in the settings option on the SettingsCollection is specified it will be used instead of this RuntimeName.

StoreValueInternally
Allows you to specify if the system should track the value for this setting. Should only be set to false in rare use cases. Enabled by default.

SettingChangeListener scripts

The setting change listener scripts allow you to add functionality via UnityEvents to any setting (including your own). You can find a listener script for the most common data types such as boolean, integer, float & string.



Setting Identifier

The identifier of the setting to react to.

Settings Collection

A settings collection reference. Only settings with the correct identifier and collection will be reacted to. If you leave this empty and no string is specified in the settings collection identifier field any setting matching the setting identifier will be reacted to. If a collection is referenced the collection identifier field will be ignored.

Settings Collection Identifier

An identifier for a settings collection to react to. If no collection reference is assigned only settings part of a settings collection matching this identifier will be reacted to.