SaintsField

`SaintsField` is a Unity Inspector extension tool focusing on script fields like NaughtyAttributes but different.

Developed by: TylerTemp , 墨瞳

Unity: 2019.1 or higher

(Yes, the project name comes from, of course, Saints Row 2)

1. Highlights

1. Works on deep nested fields!
2. Supports UI Toolkit! And it can properly handle IMGUI drawer even with UI Toolkit enabled!
3. Use and only use `PropertyDrawer` and `DecoratorDrawer` (except `SaintsEditor`, which is disabled by default), thus it will be compatible with most Unity Inspector enhancements like `NaughtyAttributes` and your custom drawer.
4. Allow stack on many cases. Only attributes that modified the label itself, and the field itself can not be stacked. All other attributes can mostly be stacked.
5. Allow dynamic arguments in many cases

2. Installation

- Using Unity Asset Store
- Using OpenUPM

```
openupm add today.comes.saintsfield
```

- Using git upm:

  add to `Packages/manifest.json` in your project

```
{
    "dependencies": {
        "today.comes.saintsfield": "https://github.com/TylerTemp/SaintsField.git",
        // your other dependencies...
    }
}
```

- Using a `unitypackage`:

  Go to the Release Page to download a desired version of `unitypackage` and import it to your project

- Using a git submodule:

```
git submodule add https://github.com/TylerTemp/SaintsField.git Assets/SaintsField
```

If you're using `unitypackage` or git submodule but you put this project under another folder rather than `Assets/SaintsField`, please also do the following:

- Create `Assets/Editor Default Resources/SaintsField`.
- Copy files from project's `Editor/Editor Default Resources/SaintsField` into your project's `Assets/Editor Default Resources/SaintsField`. If you're using a file browser instead of Unity's project tab to copy files, you may want to exclude the `.meta` file to avoid GUID conflict.

3. Change Log

2.3.8

- Add `SaintsArray`, `SaintsList` for nested array/list serialization.
- IMGUI: Change the logic of how rich text is rendered when the text is long.
- Fix `AnimatorStateChanged` not excluded Editor fields.

See the full change log .

4. Usage

4.1. Label & Text

4.1.1. `RichLabel`

- `string|null richTextXml` the content of the label, supported tag:
  - All Unity rich label tag, like `<color=#ff0000>red</color>`
  - `<label />` for current field name
  - `<icon=path/to/image.png />` for icon

  `null` means no label

  for `icon` it will search the following path:

  - `"Assets/Editor Default Resources/SaintsField/"` (You can override things here)
  - `"Assets/SaintsField/Editor/Editor Default Resources/SaintsField/"` (this is most likely to be when installed using `unitypackage`)
  - `"Packages/today.comes.saintsfield/Editor/Editor Default Resources/SaintsField/"` (this is most likely to be when installed using `upm`)
  - `Assets/Editor Default Resources/`, then fallback to built-in editor resources by name (using `EditorGUIUtility.Load`)

  for `color` it supports:

  - Standard Unity Rich Label colors:

    `aqua`, `black`, `blue`, `brown`, `cyan`, `darkblue`, `fuchsia`, `green`, `gray`, `grey`, `lightblue`, `lime`, `magenta`, `maroon`, `navy`, `olive`, `orange`, `purple`, `red`, `silver`, `teal`, `white`, `yellow`

  - Some extra colors from NaughtyAttributes:

    `clear`, `pink`, `indigo`, `violet`

  - Some extra colors from UI Toolkit:

    `charcoalGray`, `oceanicSlate`

  - html color which is supported by `ColorUtility.TryParseHtmlString`, like `#RRGGBB`, `#RRGGBBAA`, `#RGB`, `#RGBA`

- `bool isCallback=false`

  if true, the `richTextXml` will be interpreted as a property/callback function, and the string value / the returned string value (tag supported) will be used as the label content

- AllowMultiple: No. A field can only have one `RichLabel`

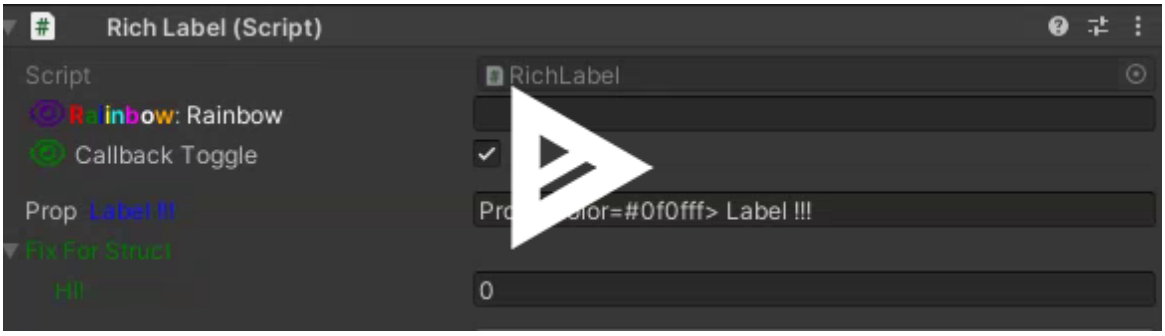
Special Note:

Use it on an array/list will apply it to all the direct child element instead of the field label itself. You can use this to modify elements of an array/list field, in this way:

1. Ensure you make it a callback: `isCallback=true`
2. Your function must receive one `int` argument
3. The `int` argument will receive a value from `0` to `length-1` of the array/list
4. Return the desired label content from the function

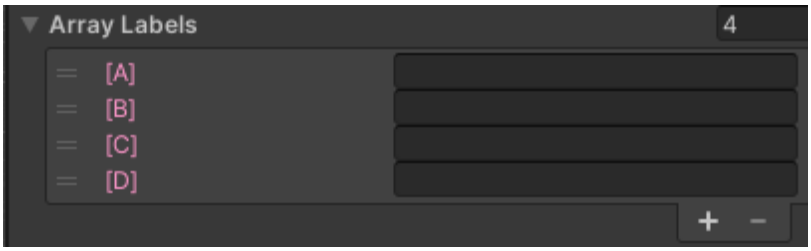
```
public class RichLabel: MonoBehaviour
{
    [RichLabel("<color=indigo><icon=eye.png /></color><b><color=red>R</color><color=green>a</co
    public string _rainbow;

    [RichLabel(nameof(LabelCallback), true)]
    public bool _callbackToggle;
    private string LabelCallback() => _callbackToggle ? "<color=green><icon=eye.png /></color>

    [Space]
    [RichLabel(nameof(_propertyLabel), true)]
    public string _propertyLabel;
    private string _rainbow;

    [Serializable]
    private struct MyStruct
    {
```

```
        [RichLabel("<color=green>HI!</color>")]
        public float LabelFloat;
    }

    [SerializeField]
    [RichLabel("<color=green>Fixed For Struct!</color>")]
    private MyStruct _myStructWorkAround;
}
```



Here is an example of using on a array:

```
[RichLabel(nameof(ArrayLabels), true)]
public string[] arrayLabels;

private string ArrayLabels(int index) => $"<color=pink>[{(char)('A' + index)}]";
```



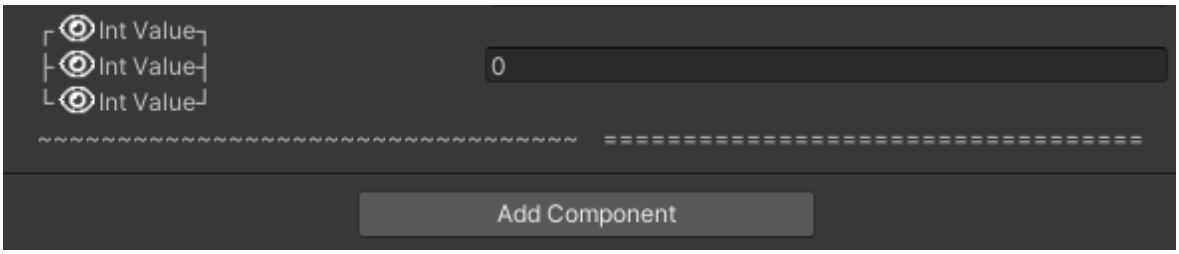
4.1.2. `AboveRichLabel` / `BelowRichLabel`

Like `RichLabel`, but it's rendered above/below the field in full width of view instead.

- `string|null richTextXml` Same as `RichLabel`
- `bool isCallback=false` Same as `RichLabel`
- `string groupBy = ""` See `GroupBy` section
- AllowMultiple: Yes

```
public class FullWidthRichLabelExample: MonoBehaviour
{
    [SerializeField]
    [AboveRichLabel("┌<icon=eye.png/><label />┐")]
    [RichLabel("├<icon=eye.png/><label />┤")]
    [BelowRichLabel(nameof(BelowLabel), true)]
```

```
    [BelowRichLabel("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~", groupBy: "example")]
    [BelowRichLabel("================================", groupBy: "example")]
    private int _intValue;

    private string BelowLabel() => "└<icon=eye.png/><label />┘";
}
```



4.1.3. `OverlayRichLabel`

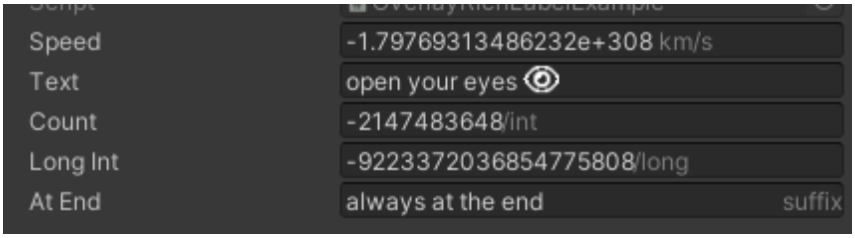
Like `RichLabel`, but it's rendered on top of the field.

Only supports string/number type of field. Does not work with any kind of `TextArea` (multiple line) and `Range`.

Parameters:

- `string richTextXml` the content of the label, or a property/callback. Supports tags like `RichLabel`
- `bool isCallback=false` if true, the `richTextXml` will be interpreted as a property/callback function, and the string value / the returned string value (tag supported) will be used as the label content
- `float padding=5f` padding between your input and the label. Not work when `end=true`
- `bool end=false` when false, the label will follow the end of your input. Otherwise, it will stay at the end of the field.
- `string GroupBy=""` this is only for the error message box.
- AllowMultiple: No

```
public class OverlayRichLabelExample: MonoBehaviour
{
    [OverlayRichLabel("<color=grey>km/s")] public double speed = double.MinValue;
    [OverlayRichLabel("<icon=eye.png/>")] public string text;
    [OverlayRichLabel("<color=grey>/int", padding: 1)] public int count = int.MinValue;
    [OverlayRichLabel("<color=grey>/long", padding: 1)] public long longInt = long.MinValue;
    [OverlayRichLabel("<color=grey>suffix", end: true)] public string atEnd;
}
```



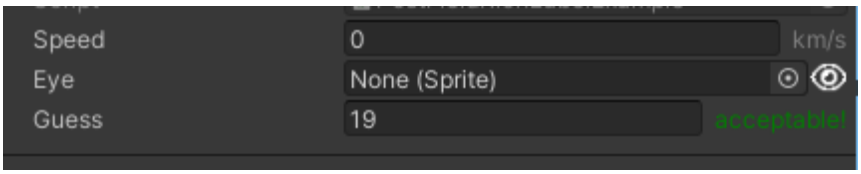
4.1.4. `PostFieldRichLabel`

Like `RichLabel`, but it's rendered at the end of the field.

Parameters:

- `string richTextXml` the content of the label, or a property/callback. Supports tags like `RichLabel`
- `bool isCallback=false` if true, the `richTextXml` will be interpreted as a property/callback function, and the string value / the returned string value (tag supported) will be used as the label content
- `float padding=5f` padding between the field and the label.
- `string GroupBy=""` this is only for the error message box.
- AllowMultiple: Yes

```
public class PostFieldRichLabelExample: MonoBehaviour
{
    [PostFieldRichLabel("<color=grey>km/s")] public float speed;
    [PostFieldRichLabel("<icon=eye.png/>", padding: 0)] public GameObject eye;
    [PostFieldRichLabel(nameof(TakeAGuess), isCallback: true)] public int guess;

    public string TakeAGuess()
    {
        if(guess > 20)
        {
            return "<color=red>too high";
        }

        if (guess < 10)
        {
            return "<color=blue>too low";
        }

        return "<color=green>acceptable!";
    }
}
```



4.1.5. `InfoBox`

Draw an info box above/below the field.

- `string content`

  The content of the info box

- `EMessageType messageType=EMessageType.Info`

  Message icon. Options are

  - `None`
  - `Info`
  - `Warning`
  - `Error`

- `string show=null`

  a callback name or property name for show or hide this info box.

- `bool isCallback=false`

  if true, the `content` will be interpreted as a property/callback function.

  If the value (or returned value) is a string, then the content will be changed

  If the value is `(EMessageType messageType, string content)` then both content and message type will be changed

- `bool above=false`

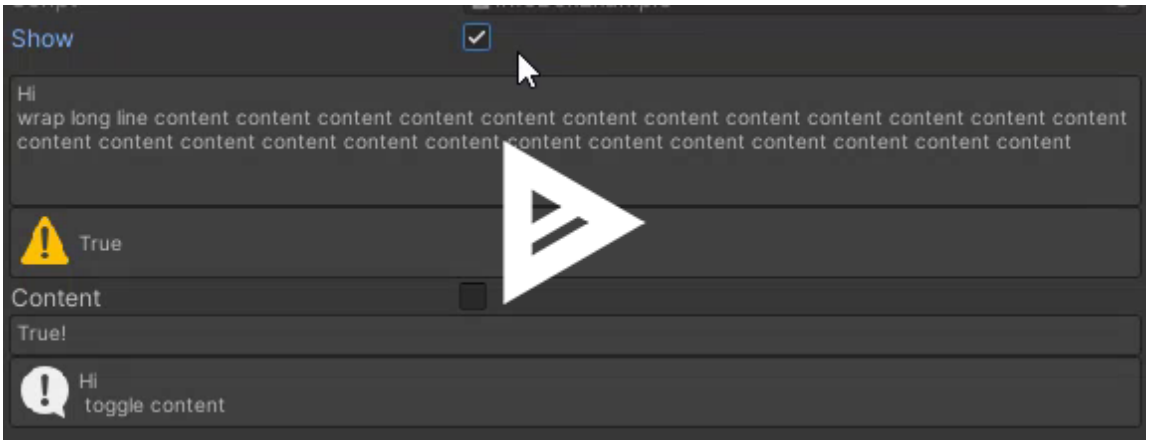
  Draw the info box above the field instead of below

- `string groupBy=""` See `GroupBy` section

- AllowMultiple: Yes

```
public class InfoBoxExample : MonoBehaviour
{
    [field: SerializeField] private bool _show;

    [Space]
    [InfoBox("Hi\nwrap long line content content content content content content content conter
    [InfoBox(nameof(DynamicMessage), EMessageType.Warning, isCallback: true, above: true)]
    [InfoBox(nameof(DynamicMessageWithIcon), isCallback: true)]
    [InfoBox("Hi\n toggle content ", EMessageType.Info, nameof(_show))]
    public bool _content;

    private (EMessageType, string) DynamicMessageWithIcon => _content ? (EMessageType.Error, "F
```

```
    private string DynamicMessage() => _content ? "False" : "True";
}
```

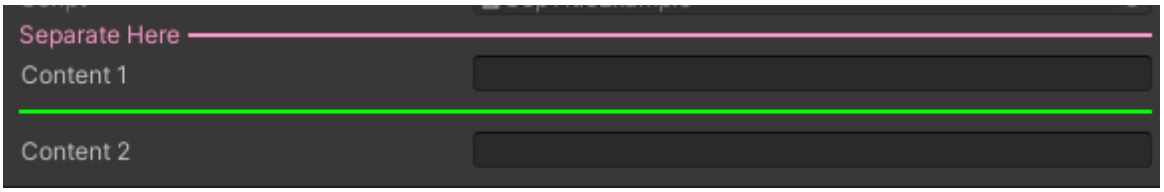


4.1.6. `SepTitle`

A separator with text

- `string title=null` title, `null` for no title at all. Does **NOT** support rich text
- `EColor color` , color for title and line separator
- `float gap = 2f` , space between title and line separator
- `float height = 2f` , height of this decorator

```
public class SepTitleExample: MonoBehaviour
{
    [SepTitle("Separate Here", EColor.Pink)]
    public string content1;

    [SepTitle(EColor.Green)]
    public string content2;
}
```



4.2. General Buttons

There are 3 general buttons:

- `AboveButton` will draw a button on above
- `BelowButton` will draw a button on below
- `PostFieldButton` will draw a button at the end of the field

All of them have the same arguments:

- `string funcName`

  called when you click the button

- `string buttonLabel=null`

  label of the button, support tags like `RichLabel` . `null` means using function name as label

- `bool isCallback = false`

  a callback or property name for button's label, same as `RichLabel`

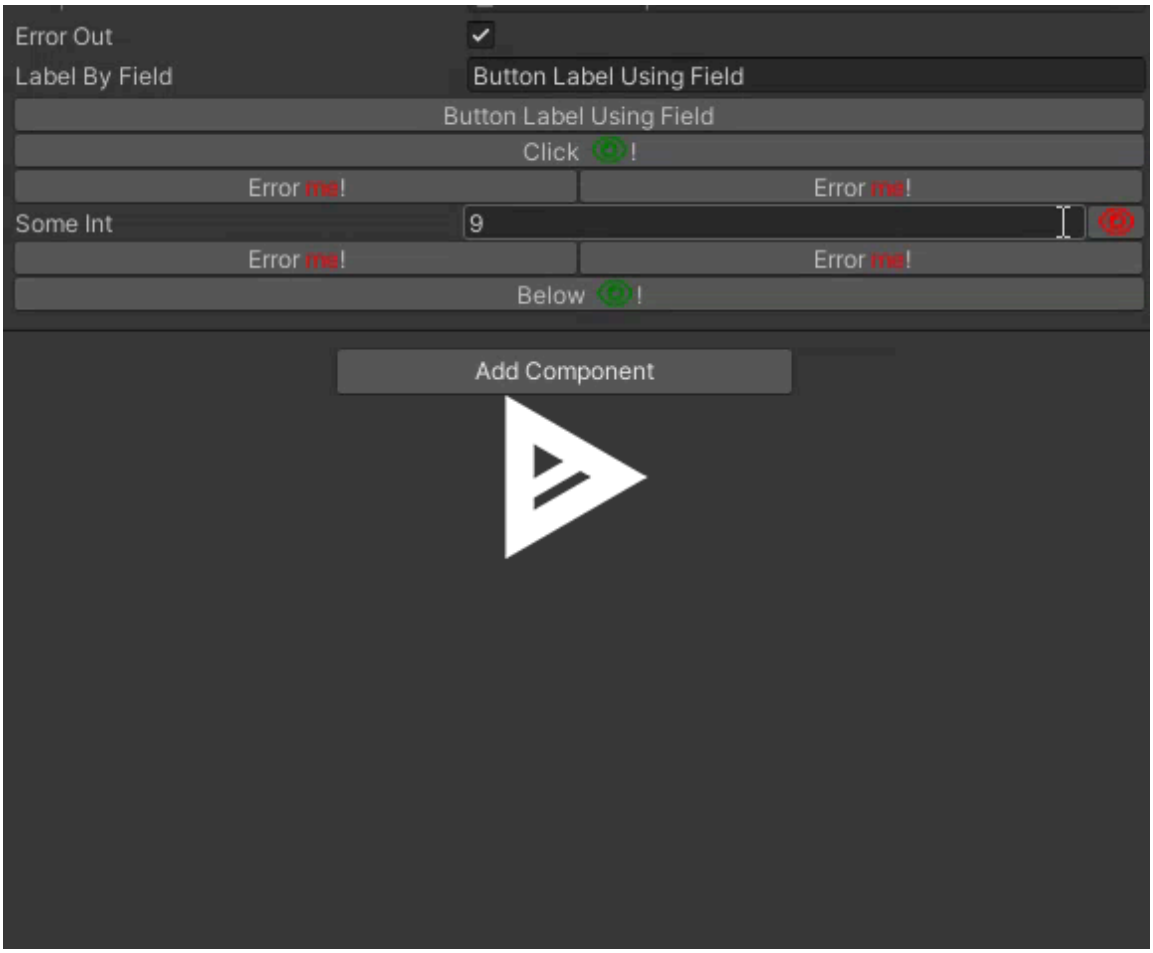
- `string groupBy = ""`

  See `GroupBy` section. Does **NOT** work on `PostFieldButton`

- AllowMultiple: Yes

```
public class ButtonsExample : MonoBehaviour
{
    [SerializeField] private bool _errorOut;

    [field: SerializeField] private string _labelByField;

    [AboveButton(nameof(ClickErrorButton), nameof(_labelByField), true)]
    [AboveButton(nameof(ClickErrorButton), "Click <color=green><icon='eye.png' /></color>!")]
    [AboveButton(nameof(ClickButton), nameof(GetButtonLabel), true, "OK")]
    [AboveButton(nameof(ClickButton), nameof(GetButtonLabel), true, "OK")]

    [PostFieldButton(nameof(ToggleAndError), nameof(GetButtonLabelIcon), true)]

    [BelowButton(nameof(ClickButton), nameof(GetButtonLabel), true, "OK")]
    [BelowButton(nameof(ClickButton), nameof(GetButtonLabel), true, "OK")]
    [BelowButton(nameof(ClickErrorButton), "Below <color=green><icon='eye.png' /></color>!")]
    public int _someInt;

    private void ClickErrorButton() => Debug.Log("CLICKED!");

    private string GetButtonLabel() =>
        _errorOut
            ? "Error <color=red>me</color>!"
            : "No <color=green>Error</color>!";

    private string GetButtonLabelIcon() => _errorOut
        ? "<color=red><icon='eye.png' /></color>"
        : "<color=green><icon='eye.png' /></color>";

    private void ClickButton()
    {
        Debug.Log("CLICKED 2!");
```

```
            if(_errorOut)
            {
                throw new Exception("Expected exception!");
            }
        }

        private void ToggleAndError()
        {
            Toggle();
            ClickButton();
        }

        private void Toggle() => _errorOut = !_errorOut;
    }
```



4.3. Field Modifier

4.3.1. `GameObjectActive`

A toggle button to toggle the `GameObject.activeSelf` of the field.

This does not require the field to be `GameObject`. It can be a component which already attached to a `GameObject`.

- AllowMultiple: No

```
public class GameObjectActiveExample : MonoBehaviour
{
    [GameObjectActive] public GameObject _go;
    [GameObjectActive] public GameObjectActiveExample _component;
}
```

4.3.2. `SpriteToggle`

A toggle button to toggle the `Sprite` of the target.

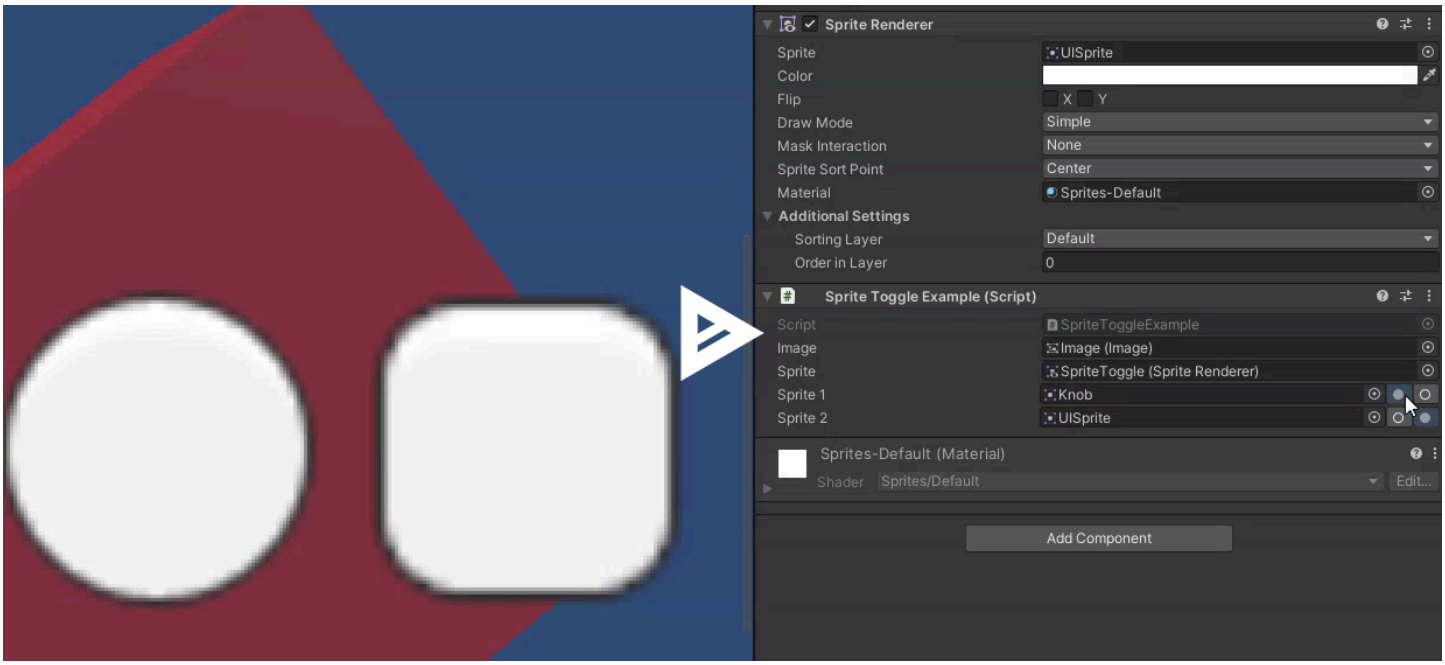
The field itself must be `Sprite`.

- `string imageOrSpriteRenderer`

  the target, must be either `UI.Image` or `SpriteRenderer`

- AllowMultiple: Yes

```
public class SpriteToggleExample : MonoBehaviour
{
    [field: SerializeField] private Image _image;
    [field: SerializeField] private SpriteRenderer _sprite;

    [SerializeField
     , SpriteToggle(nameof(_image))
     , SpriteToggle(nameof(_sprite))
    ] private Sprite _sprite1;
    [SerializeField
     , SpriteToggle(nameof(_image))
     , SpriteToggle(nameof(_sprite))
    ] private Sprite _sprite2;
}
```



4.3.3. `MaterialToggle`

A toggle button to toggle the `Material` of the target.

The field itself must be `Material`.

- `string rendererName=null`

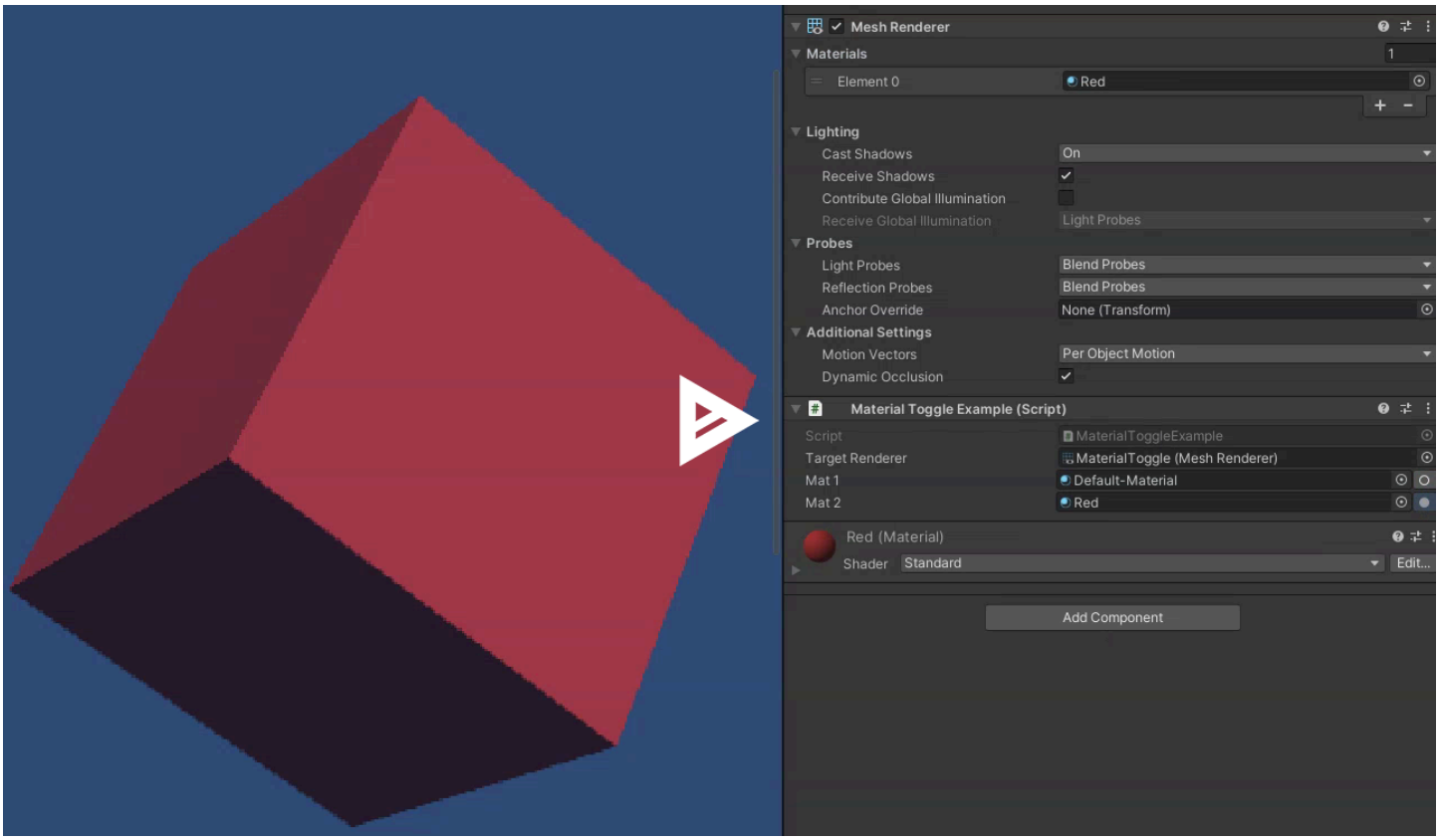
  the target, must be `Renderer` (or its subClass like `MeshRenderer`). When using null, it will try to get the `Renderer` component from the current component

- `int index=0`

  which slot index of `materials` on `Renderer` you want to swap

- AllowMultiple: Yes

```
public class MaterialToggleExample: MonoBehaviour
{
    public Renderer targetRenderer;
    [MaterialToggle(nameof(targetRenderer))] public Material _mat1;
    [MaterialToggle(nameof(targetRenderer))] public Material _mat2;
}
```



4.3.4. `ColorToggle`

A toggle button to toggle color for `Image` , `Button` , `SpriteRenderer` or `Renderer`

The field itself must be `Color` .

- `string compName=null`

  the target, must be `Image` , `Button` , `SpriteRenderer` , or `Renderer` (or its subClass like `MeshRenderer`).

  When using `null` , it will try to get the correct component from the target object of this field by order.

  When it's a `Renderer` , it will change the material's `.color` property.

  When it's a `Button` , it will change the button's `targetGraphic.color` property.

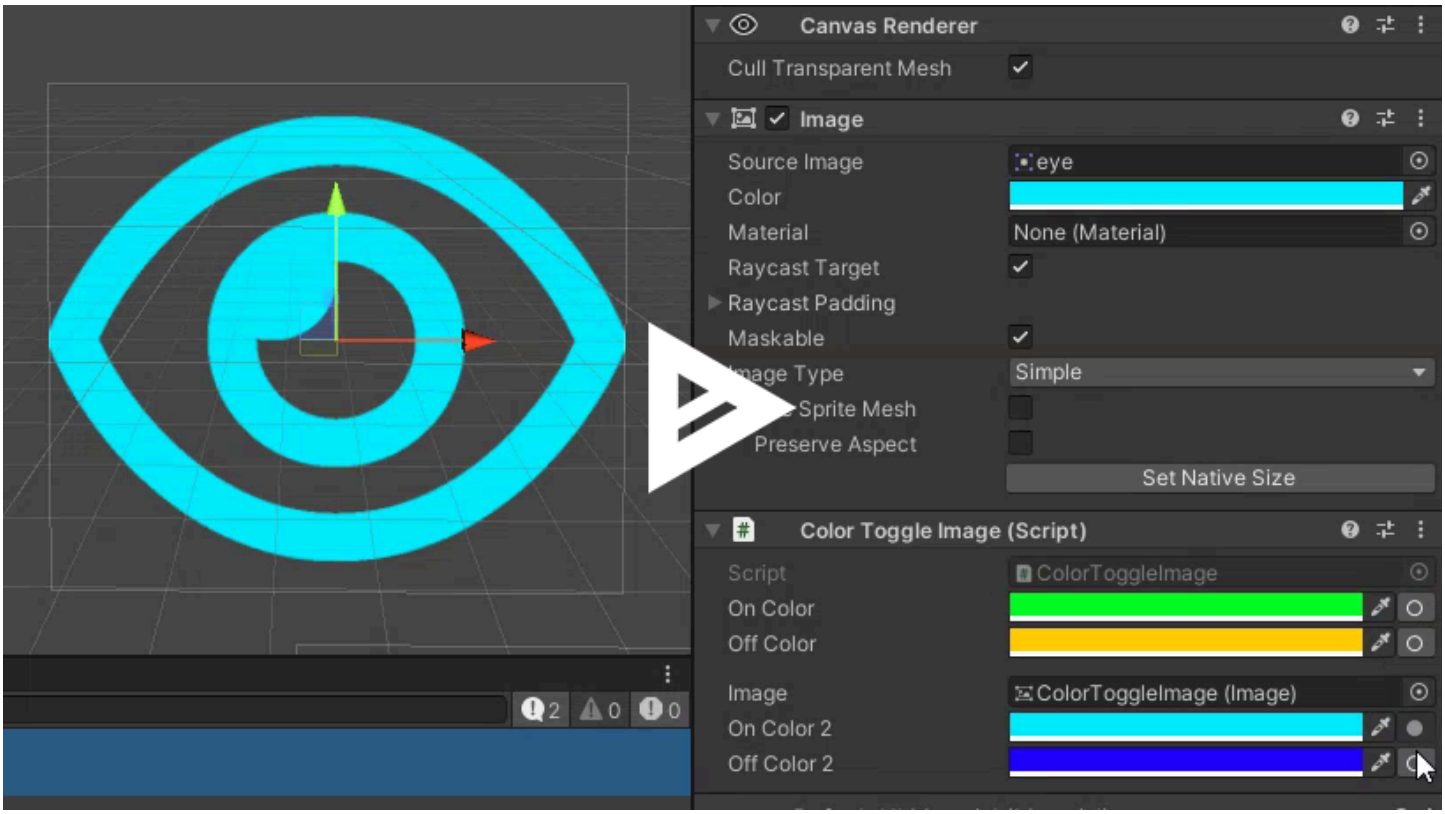
- `int index=0`

  (only works for `Renderer` type) which slot index of `materials` on `Renderer` you want to apply the color

- AllowMultiple: Yes

```
public class ColorToggleImage: MonoBehaviour
{
    // auto find on the target object
```

```
    [SerializeField, ColorToggle] private Color _onColor;
    [SerializeField, ColorToggle] private Color _offColor;

    [Space]
    // by name
    [SerializeField] private Image _image;
    [SerializeField, ColorToggle(nameof(_image))] private Color _onColor2;
    [SerializeField, ColorToggle(nameof(_image))] private Color _offColor2;
}
```



4.3.5. `Expandable`

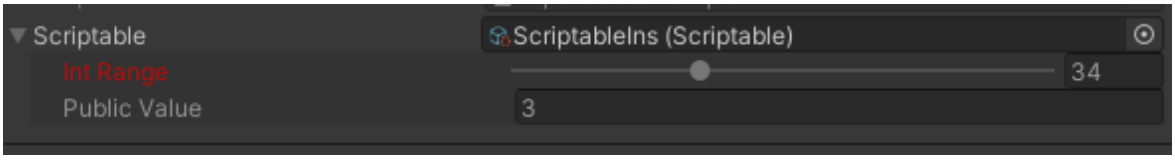
Make serializable object expandable. (E.g. `ScriptableObject`, `MonoBehavior`)

Known issue:

1. IMGUI: if the target itself has a custom drawer, the drawer will not be used, because `PropertyDrawer` is not allowed to create an `Editor` class, thus it'll just iterate and draw all fields in the object.
2. IMGUI: the `Foldout` will NOT be placed at the left space like a Unity's default foldout component, because Unity limited the `PropertyDrawer` to be drawn inside the rect Unity gives. Trying outside of the rect will make the target non-interactable. But in early Unity (like 2019.1), Unity will force `Foldout` to be out of rect on top leve, but not on array/list level... so you may see different outcomes on different Unity version.
3. UI Toolkit: `ReadOnly` (and `DisableIf`, `EnableIf`) can NOT disable the expanded fields. This is because `InspectorElement` does not work with `SetEnable(false)`, neither with `pickingMode=Ignore`. This can not be fixed unless Unity fixes it.

- AllowMultiple: No

```
public class ExpandableExample : MonoBehaviour
{
    [Expandable] public ScriptableObject _scriptable;
}
```



4.3.6. `ReferencePicker`

A dropdown to pick a referenced value for Unity's `SerializeReference`.

You can use this to pick non UnityObject object like `interface` or polymorphism `class`.

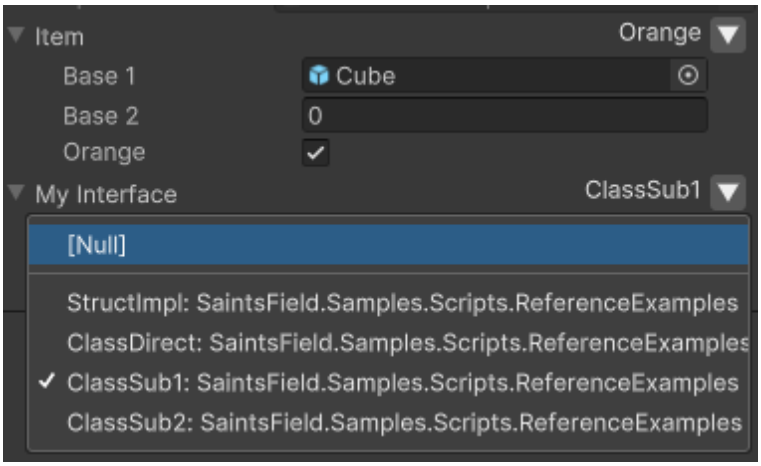
Limitation:

1. The target must have a public constructor with no required arguments.
2. It'll try to copy field values when changing types but not guaranteed. `struct` will not get copied value (it's too tricky to deal a struct)

- Allow Multiple: No

```
    public class ReferenceExample: MonoBehaviour
    {
        [Serializable]
        public class Base1Fruit
        {
            public GameObject base1;
        }

        [Serializable]
        public class Base2Fruit: Base1Fruit
        {
            public int base2;
        }

        [Serializable]
        public class Apple : Base2Fruit
        {
            public string apple;
            public GameObject applePrefab;
        }

        [Serializable]
        public class Orange : Base2Fruit
        {
```

```
        public bool orange;
    }

    [SerializeReference, ReferencePicker]
    public Base2Fruit item;

    public interface IRefInterface
    {
        public int TheInt { get; }
    }

    // works for struct
    [Serializable]
    public struct StructImpl : IRefInterface
    {
        [field: SerializeField]
        public int TheInt { get; set; }
        public string myStruct;
    }

    [Serializable]
    public class ClassDirect: IRefInterface
    {
        [field: SerializeField, Range(0, 10)]
        public int TheInt { get; set; }
    }

    // abstruct type will be skipped
    public abstract class ClassSubAbs : ClassDirect
    {
        public abstract string AbsValue { get; }
    }

    [Serializable]
    public class ClassSub1 : ClassSubAbs
    {
        public string sub1;
        public override string AbsValue => $"Sub1: {sub1}";
    }

    [Serializable]
    public class ClassSub2 : ClassSubAbs
    {
        public string sub2;
        public override string AbsValue => $"Sub2: {sub2}";
    }

    [SerializeReference, ReferencePicker]
    public IRefInterface myInterface;

}
```



4.3.7. `ParticlePlay`

A button to play a particle system of the field value, or the one on the field value.

Unity allows play ParticleSystem in the editor, but only if you selected the target GameObject. It can only play one at a time.

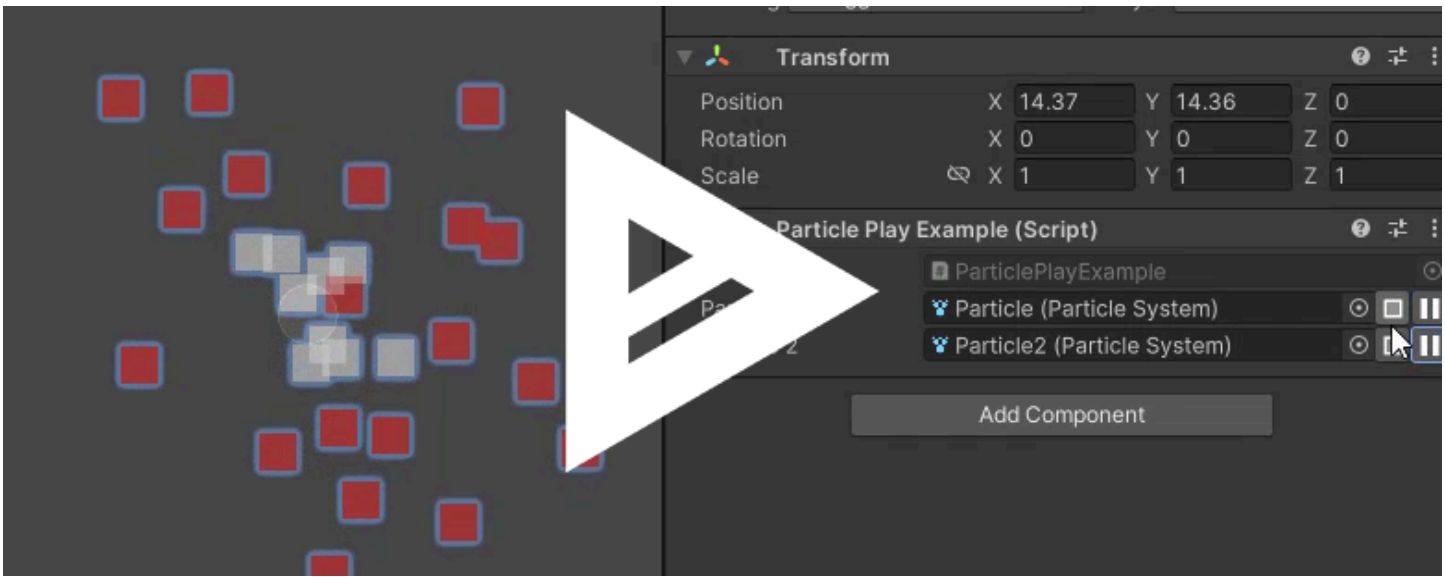
This decorator allows you to play multiple ParticleSystem as long as you have the expected fields.

Parameters:

- `string groupBy = ""` for error grouping.
- Allow Multiple: No

Note: because of the limitation from Unity, it can NOT detect if a `ParticleSystem` is finished playing

```
[ParticlePlay] public ParticleSystem particle;
// It also works if the field target has a particleSystem component
[ParticlePlay, FieldType(typeof(ParticleSystem), false)] public GameObject particle2;
```



4.4. Field Re-Draw

This will change the look & behavior of a field.

4.4.1. `Rate`

A rating stars tool for an `int` field.

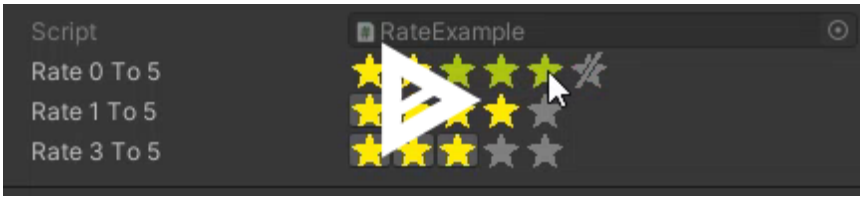
Parameters:

- `int min` minimum value of the rating. Must be equal to or greater than 0.

  When it's equal to 0, it'll draw a red slashed star to select `0`.

  When it's greater than 0, it will draw `min` number of fixed stars that you can not un-rate.
- `int max` maximum value of the rating. Must be greater than `min`.
- AllowMultiple: No

```
public class RateExample: MonoBehaviour
{
    [Rate(0, 5)] public int rate0To5;
    [Rate(1, 5)] public int rate1To5;
    [Rate(3, 5)] public int rate3To5;
}
```



4.4.2. `FieldType`

Ask the inspector to display another type of field rather than the field's original type.

This is useful when you want to have a `GameObject` prefab, but you want this target prefab to have a specific component (e.g. your own `MonoScript`, or a `ParticalSystem`). By using this you force the inspector to sign the required object that has your expected component but still gives you the original typed value to field.

Overload:

- `FieldTypeAttribute(Type compType, EPick editorPick = EPick.Assets | EPick.Scene, bool customPicker = true)`
- `FieldTypeAttribute(Type compType, bool customPicker)`

For each argument:

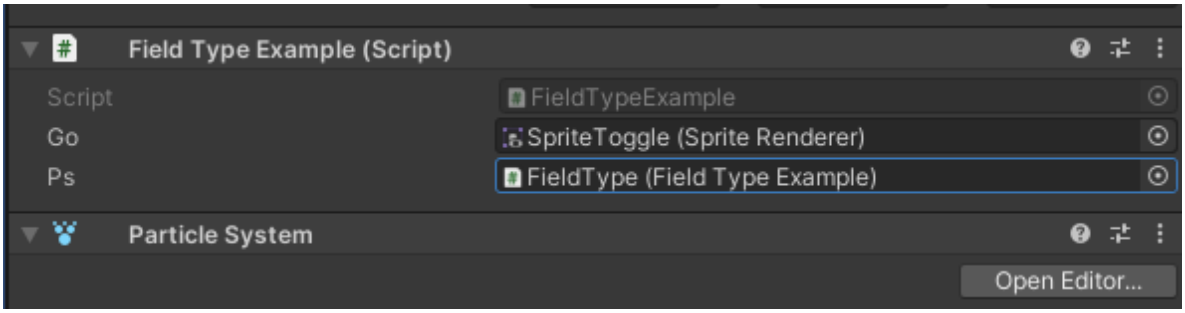
- `Type compType` the type of the component you want to pick

- `EPick editorPick` where you want to pick the component. Options are:
  - `EPick.Assets` for assets
  - `EPick.Scene` for scene objects

  For the default Unity picker: if no `EPick.Scene` is set, will not show the scene objects. However, omit `Assets` will still show the assets. This limitation is from Unity's API.

  The custom picker does **NOT** have this limitation.
- `customPicker` show an extra button to use a custom picker. Disable this if you have serious performance issue.
- AllowMultiple: No

```
public class FieldTypeExample: MonoBehaviour
{
    [SerializeField, FieldType(typeof(SpriteRenderer))]
    private GameObject _go;

    [SerializeField, FieldType(typeof(FieldTypeExample))]
    private ParticleSystem _ps;
}
```



4.4.3. `Dropdown`

A dropdown selector. Supports reference type, sub-menu, separator, and disabled select item.

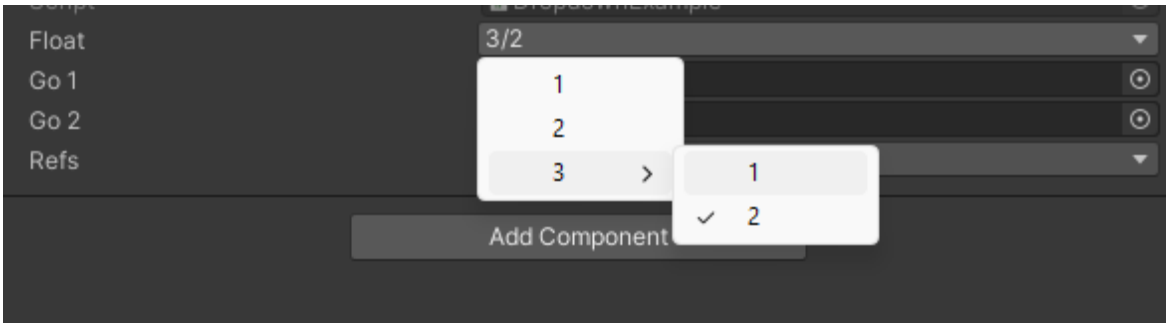
If you want a searchable dropdown, see `AdvancedDropdown`.

- `string funcName` callback function. Must return a `DropdownList<T>`.
- `bool slashAsSub=true` treat `/` as a sub item.

  Note: In `IMGUI`, this just replace `/` to unicode `\u2215` Division Slash ∕, and WILL have a little bit overlap with nearby characters.
- AllowMultiple: No

Example

```
public class DropdownExample : MonoBehaviour
{
    [Dropdown(nameof(GetDropdownItems))] public float _float;

    public GameObject _go1;
    public GameObject _go2;
    [Dropdown(nameof(GetDropdownRefs))] public GameObject _refs;

    private DropdownList<float> GetDropdownItems()
    {
        return new DropdownList<float>
        {
            { "1", 1.0f },
            { "2", 2.0f },
            { "3/1", 3.1f },
            { "3/2", 3.2f },
        };
    }

    private DropdownList<GameObject> GetDropdownRefs => new DropdownList<GameObject>
    {
        {_go1.name, _go1},
        {_go2.name, _go2},
        {"NULL", null},
    };
}
```



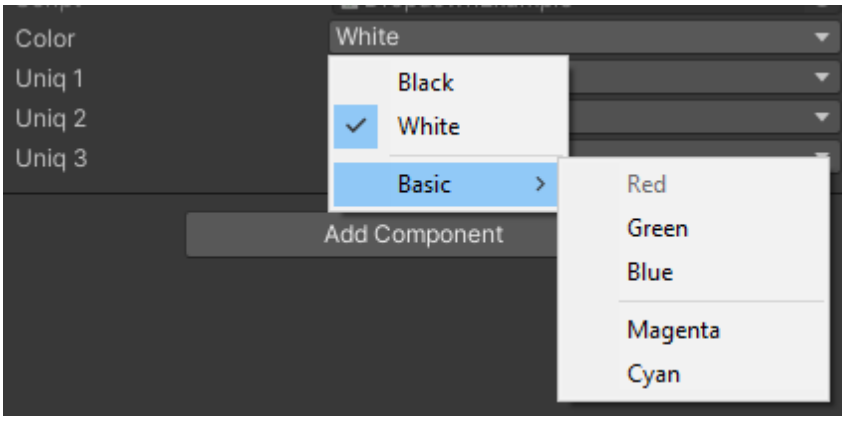
To control the separator and disabled item

```
    [Dropdown(nameof(GetDropdownItems))]
    public Color color;

    private DropdownList<Color> GetDropdownItems()
    {
        return new DropdownList<Color>
        {
            { "Black", Color.black },
            { "White", Color.white },
            DropdownList<Color>.Separator(),
            { "Basic/Red", Color.red, true },  // the third arg means it's disabled
            { "Basic/Green", Color.green },
```

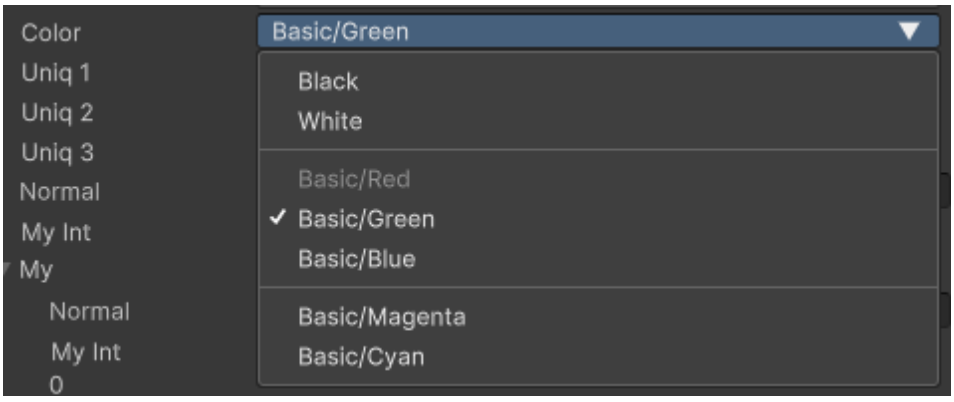
```
            { "Basic/Blue", Color.blue },
            DropdownList<Color>.Separator("Basic/"),
            { "Basic/Magenta", Color.magenta },
            { "Basic/Cyan", Color.cyan },
        };
    }
```

And you can always manually add it:

```
DropdownList<Color> dropdownList = new DropdownList<Color>();
dropdownList.Add("Black", Color.black);  // add an item
dropdownList.Add("White", Color.white, true);  // and a disabled item
dropdownList.AddSeparator();  // add a separator
```



The look in the UI Toolkit with `slashAsSub: false` :



4.4.4. `AdvancedDropdown`

A dropdown selector. Supports reference type, sub-menu, separator, search, and disabled select item, plus icon.

Known Issue :

1. IMGUI: Using Unity's `AdvancedDropdown` . Unity's `AdvancedDropdown` allows to click the disabled item and close the popup, thus you can still click the disable item. This is a BUG from Unity. I managed to "hack" it around to show again the popup when you click the disabled item, but you will see the flick of the popup.

This issue is not fixable unless Unity fixes it.

This bug only exists in IMGUI

2. UI Toolkit:

   The group indicator uses `ToolbarBreadcrumbs`. Sometimes you can see text get wrapped into lines. This is because Unity's UI Toolkit has some layout issue, that it can not has the same layout even with same elements+style+boundary size.

   This issue is not fixable unless Unity fixes it. This issue might be different on different Unity (UI Toolkit) version.

Arguments

- `string funcName` callback function. Must return a `AdvancedDropdownList<T>`.
- (IMGUI) `float itemHeight=-1f` height of each item. `< 0` means use Unity's default value. This will not change the actual height of item, but to decide the dropdown height.
- (IMGUI) `float titleHeight=Default` height of the title. This will not change the actual height of title, but to decide the dropdown height.
- (IMGUI) `float sepHeight=Default` height of separator. This will not change the actual height of title, but to decide the dropdown height.
- (IMGUI) `bool useTotalItemCount=false` if true, the dropdown height will be decided using the number of all value item, thus the search result will always fit in the position without scroll. Otherwise, it'll be decided by the max height of every item page.
- (IMGUI) `float minHeight=-1f` minimum height of the dropdown. `< 0` means no limit. Otherwise, use this as the dropdown height and ignore all the other auto height config.
- AllowMultiple: No

`AdvancedDropdownList<T>`

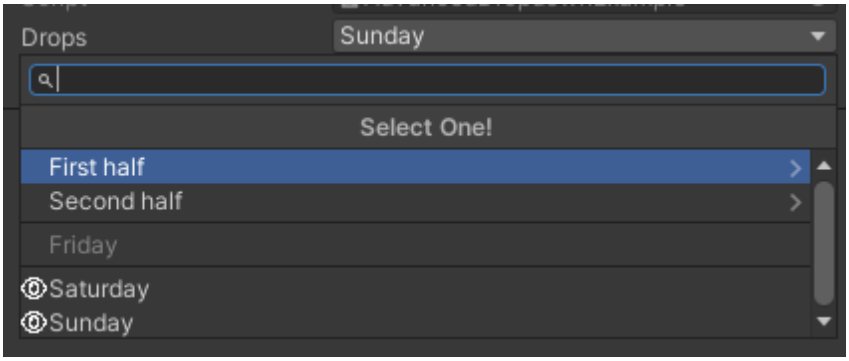
- `string displayName` item name to display
- `T value` or `IEnumerable<AdvancedDropdownList<T>> children`: value means it's a value item. Otherwise it's a group of items, which the values are specified by `children`
- `bool disabled = false` if item is disabled
- `string icon = null` the icon for the item.

  Note: setting an icon for a parent group will result an weird issue on it's sub page's title and block the items. This is not fixable unless Unity decide to fix it.

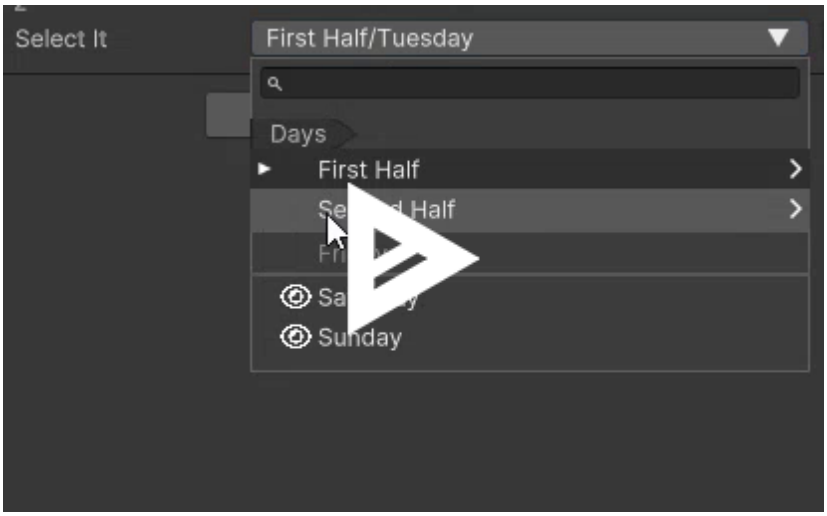
- `bool isSeparator = false` if item is a separator. You should not use this, but `AdvancedDropdownList<T>.Separator()` instead

```
public class AdvancedDropdownExample: MonoBehaviour
{
    [AdvancedDropdown(nameof(AdvDropdown)), BelowRichLabel(nameof(drops), true)] public int dro

    public AdvancedDropdownList<int> AdvDropdown()
    {
        return new AdvancedDropdownList<int>("Days")
        {
            // a grouped value
            new AdvancedDropdownList<int>("First Half")
            {
                // with icon
                new AdvancedDropdownList<int>("Monday", 1, icon: "eye.png"),
                // no icon
                new AdvancedDropdownList<int>("Tuesday", 2),
            },
            new AdvancedDropdownList<int>("Second Half")
            {
                new AdvancedDropdownList<int>("Wednesday")
                {
                    new AdvancedDropdownList<int>("Morning", 3, icon: "eye.png"),
                    new AdvancedDropdownList<int>("Afternoon", 8),
                },
                new AdvancedDropdownList<int>("Thursday", 4, true, icon: "eye.png"),
            },
            // direct value
            new AdvancedDropdownList<int>("Friday", 5, true),
            AdvancedDropdownList<int>.Separator(),
            new AdvancedDropdownList<int>("Saturday", 6, icon: "eye.png"),
            new AdvancedDropdownList<int>("Sunday", 7, icon: "eye.png"),
        };
    }
}
```

IMGUI

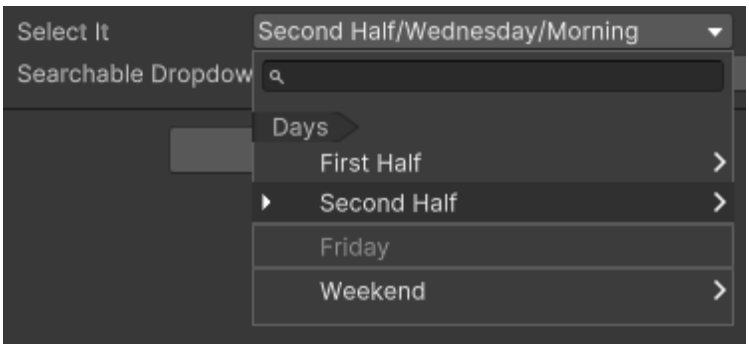


UI Toolkit



There is also a parser to automatically separate items as sub items using `/`:

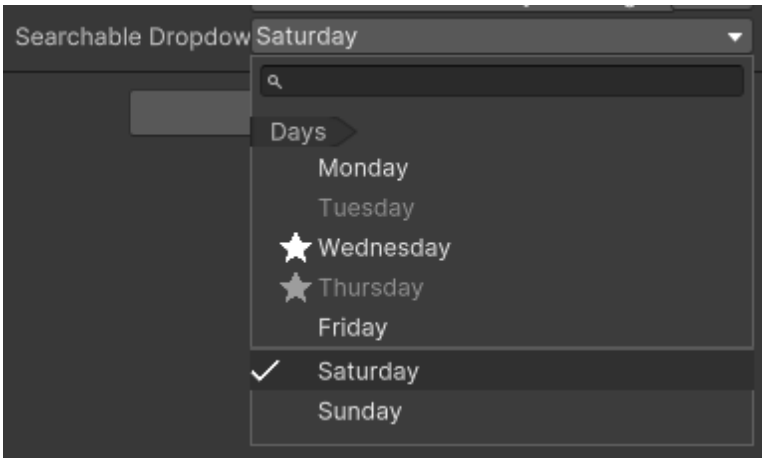
```
[AdvancedDropdown(nameof(AdvDropdown))] public int selectIt;

public AdvancedDropdownList<int> AdvDropdown()
{
    return new AdvancedDropdownList<int>("Days")
    {
        {"First Half/Monday", 1, false, "star.png"},  // enabled, with icon
        {"First Half/Tuesday", 2},

        {"Second Half/Wednesday/Morning", 3, false, "star.png"},
        {"Second Half/Wednesday/Afternoon", 4},
        {"Second Half/Thursday", 5, true, "star.png"},  // disabled, with icon
        "",  // root separator
        {"Friday", 6, true},  // disabled
        "",
        {"Weekend/Saturday", 7, false, "star.png"},
        "Weekend/",  // separator under `Weekend` group
        {"Weekend/Sunday", 8, false, "star.png"},
    };
}
```



You can use this to make a searchable dropdown:

```
[AdvancedDropdown(nameof(AdvDropdownNoNest))] public int searchableDropdown;
```

```
public AdvancedDropdownList<int> AdvDropdownNoNest()
{
    return new AdvancedDropdownList<int>("Days")
    {
        {"Monday", 1},
        {"Tuesday", 2, true},  // disabled
        {"Wednesday", 3, false, "star.png"},  // enabled with icon
        {"Thursday", 4, true, "star.png"},  // disabled with icon
        {"Friday", 5},
        "",  // separator
        {"Saturday", 6},
        {"Sunday", 7},
    };
}
```



4.4.5. `PropRange`

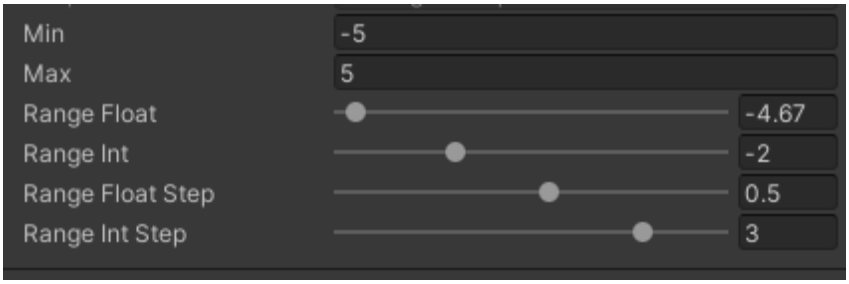
Very like Unity's `Range` but allow you to dynamically change the range, plus allow to set range step.

For each argument:

- `string minCallback` or `float min` : the minimum value of the slider, or a property/callback name.
- `string maxCallback` or `float max` : the maximum value of the slider, or a property/callback name.
- `float step=-1f` : the step for the range. `<= 0` means no limit.

```
public class RangeExample: MonoBehaviour
{
    public int min;
    public int max;

    [PropRange(nameof(min), nameof(max))] public float rangeFloat;
    [PropRange(nameof(min), nameof(max))] public int rangeInt;

    [PropRange(nameof(min), nameof(max), step: 0.5f)] public float rangeFloatStep;
```

```
    [PropRange(nameof(min), nameof(max), step: 2)] public int rangeIntStep;
}
```



4.4.6. `MinMaxSlider`

A range slider for `Vector2` or `Vector2Int`

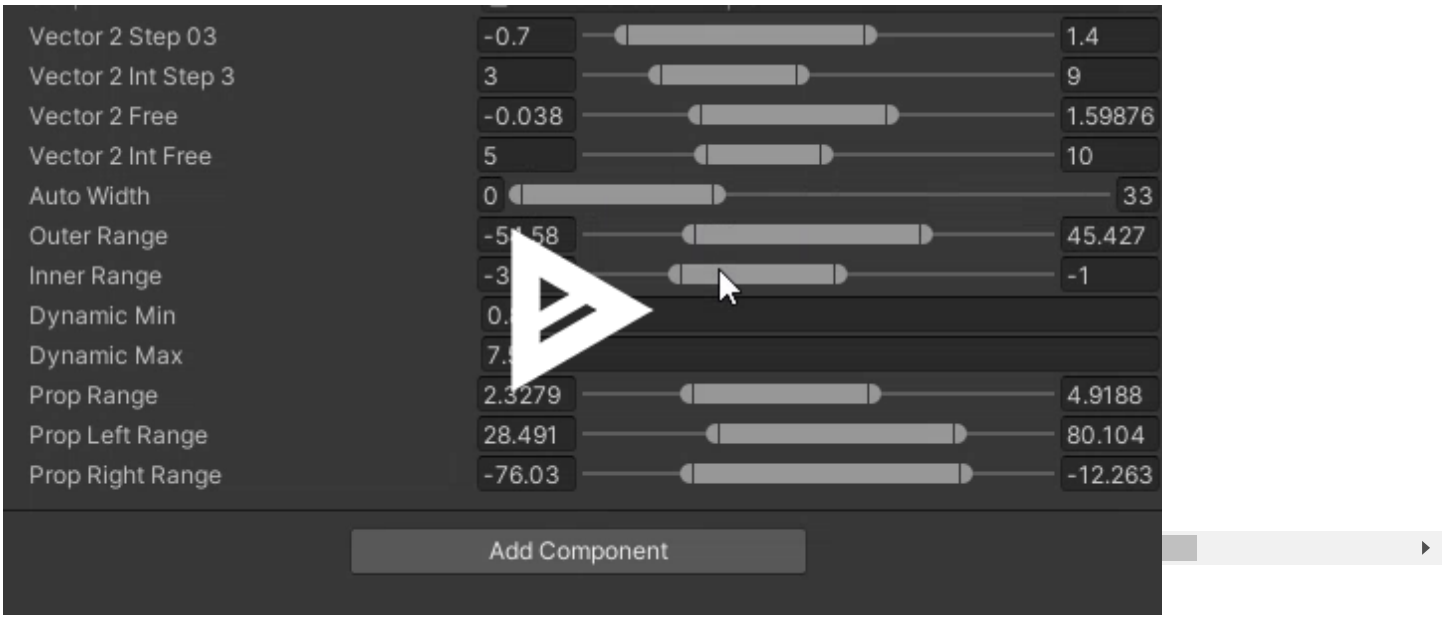
For each argument:

- `int|float min` or `string minCallback` : the minimum value of the slider, or a property/callback name.
- `int|float max` or `string maxCallback` : the maximum value of the slider, or a property/callback name.
- `int|float step=1|-1f` : the step of the slider, `<= 0` means no limit. By default, int type use `1` and float type use `-1f`
- `float minWidth=50f` : (IMGUI Only) the minimum width of the value label. `< 0` for auto size (not recommended)
- `float maxWidth=50f` : (IMGUI Only) the maximum width of the value label. `< 0` for auto size (not recommended)
- AllowMultiple: No

a full-featured example:

```
public class MinMaxSliderExample: MonoBehaviour
{
    [MinMaxSlider(-1f, 3f, 0.3f)]
    public Vector2 vector2Step03;

    [MinMaxSlider(0, 20, 3)]
    public Vector2Int vector2IntStep3;

    [MinMaxSlider(-1f, 3f)]
    public Vector2 vector2Free;

    [MinMaxSlider(0, 20)]
    public Vector2Int vector2IntFree;
```

```
    // not recommended
    [SerializeField]
    [MinMaxSlider(0, 100, minWidth:-1, maxWidth:-1)]
    private Vector2Int _autoWidth;

    [field: SerializeField, MinMaxSlider(-100f, 100f)]
    public Vector2 OuterRange { get; private set; }

    [SerializeField, MinMaxSlider(nameof(GetOuterMin), nameof(GetOuterMax), 1)] public Vector2I

    private float GetOuterMin() => OuterRange.x;
    private float GetOuterMax() => OuterRange.y;

    [field: SerializeField]
    public float DynamicMin { get; private set; }
    [field: SerializeField]
    public float DynamicMax { get; private set; }

    [SerializeField, MinMaxSlider(nameof(DynamicMin), nameof(DynamicMax))] private Vector2 _prc
    [SerializeField, MinMaxSlider(nameof(DynamicMin), 100f)] private Vector2 _propLeftRange;
    [SerializeField, MinMaxSlider(-100f, nameof(DynamicMax))] private Vector2 _propRightRange;
}
```



4.4.7. `EnumFlags`

A toggle buttons group for enum flags (bit mask). It provides a button to toggle all bits on/off.

This field has compact mode and expanded mode.

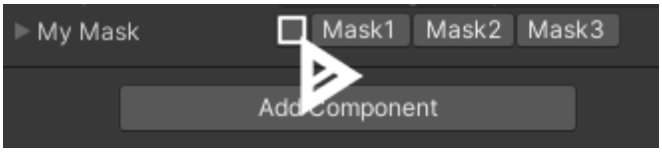
For each argument:

- `bool autoExpand=true` : if the view is not enough to show all buttons in a row, automatically expand to a vertical group.

- `bool defaultExpanded=false` : if true, the buttons group will be expanded as a vertical group by default.
- AllowMultiple: No

Note: If you have a lot of flags and you turn **OFF** `autoExpand` , The buttons **WILL** go off-view.

```
public class EnumFlagsExample: MonoBehaviour
{
    [Serializable, Flags]
    public enum BitMask
    {
        None = 0,  // this will be hide as we will have an all/none button
        Mask1 = 1,
        Mask2 = 1 << 1,
        Mask3 = 1 << 2,
    }

    [EnumFlags] public BitMask myMask;
}
```



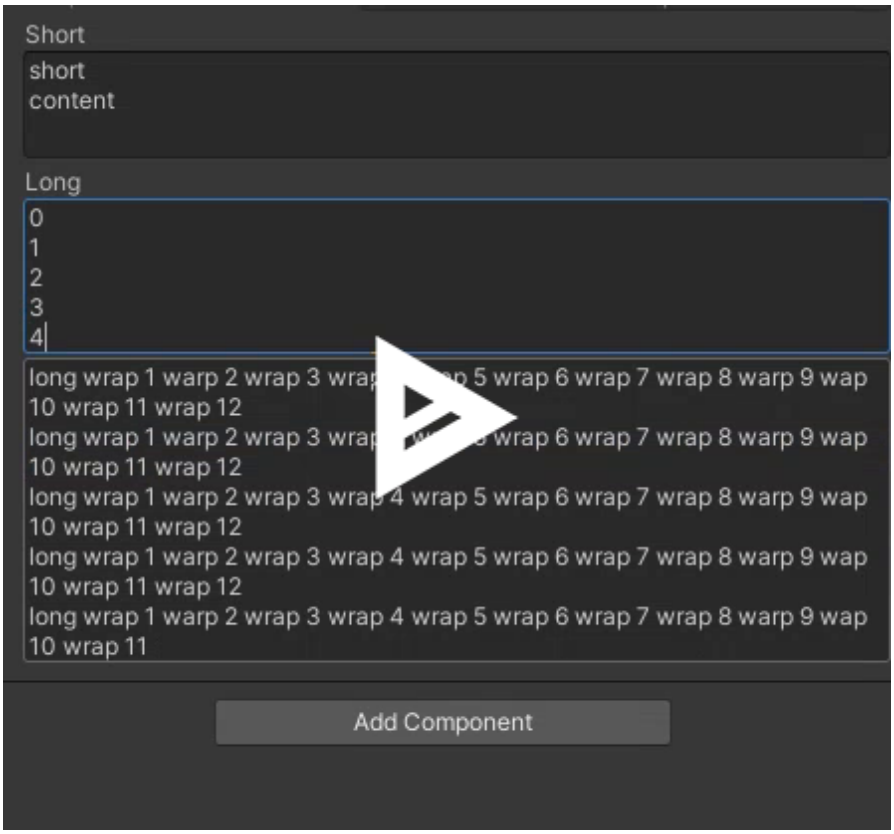
4.4.8. `ResizableTextArea`

This `TextArea` will always grow its height to fit the content. (minimal height is 3 rows).

Note: Unlike NaughtyAttributes, this does not have a text-wrap issue.

- AllowMultiple: No

```
public class ResizableTextAreaExample : MonoBehaviour
{
    [SerializeField, ResizableTextArea] private string _short;
    [SerializeField, ResizableTextArea] private string _long;
    [SerializeField, RichLabel(null), ResizableTextArea] private string _noLabel;
}
```



4.4.9. `AnimatorParam`

A dropdown selector for an animator parameter.

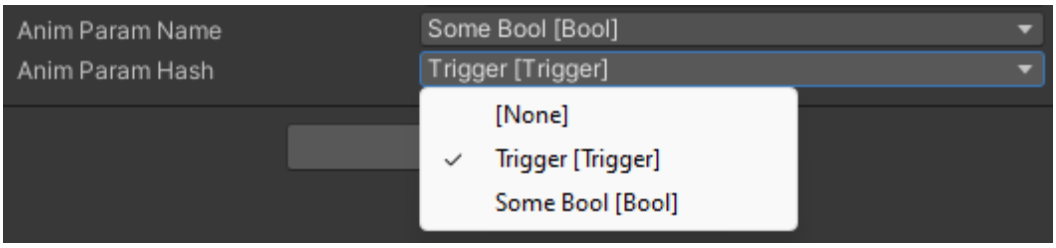
- `string animatorName=null`

  name of the animator. When omitted, it will try to get the animator from the current component

- (Optional) `AnimatorControllerParameterType animatorParamType`

  type of the parameter to filter

```
public class Anim : MonoBehaviour
{
    [field: SerializeField]
    public Animator Animator { get; private set;}

    [AnimatorParam(nameof(Animator))]
    private string animParamName;

    [AnimatorParam(nameof(Animator))]
    private int animParamHash;
}
```



4.4.10. `AnimatorState`

A dropdown selector for animator state.

- `string animatorName=null`

  name of the animator. When omitted, it will try to get the animator from the current component

to get more useful info from the state, you can use `AnimatorStateBase` / `AnimatorState` type instead of `string` type.

`AnimatorStateBase` has the following properties:

- `int layerIndex` index of layer
- `int stateNameHash` hash value of state
- `string stateName` actual state name
- `float stateSpeed` the `Speed` parameter of the state
- `string stateTag` the `Tag` of the state
- `string[] subStateMachineNameChain` the sub-state machine hierarchy name list of the state

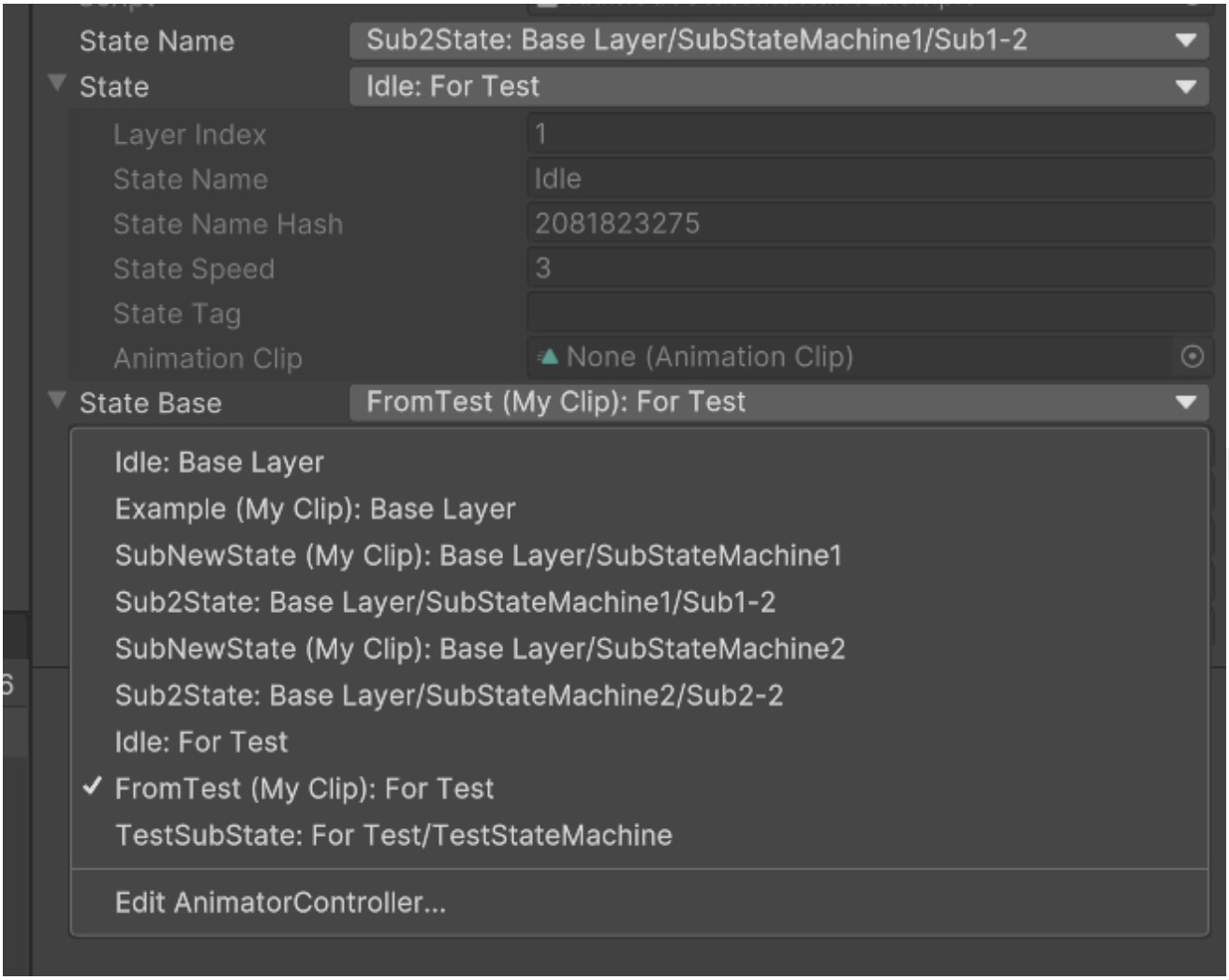
`AnimatorState` added the following attribute(s):

- `AnimationClip animationClip` is the actual animation clip of the state (can be null). It has a `length` value for the length of the clip. For more detail see Unity Doc of AnimationClip

Special Note: using `AniamtorState` / `AnimatorStateBase` with `OnValueChanged`, you can get a `AnimatorStateChanged` on the callback (rather than the value of the field). This is because `AnimatorState` expected any class/struct with satisfied fields.

```
public class Anim : MonoBehaviour
{
    [AnimatorState, OnValueChanged(nameof(OnChanged))]
    public string stateName;

    [AnimatorState, OnValueChanged(nameof(OnChangedState))]
    public AnimatorState state;

    // This does not have a `animationClip`, thus it won't include a resource when serialized:
    [AnimatorState, OnValueChanged(nameof(OnChangedState))]
    public AnimatorStateBase stateBase;
```

```
    private void OnChanged(string changedValue) => Debug.Log(changedValue);
    private void OnChangedState(AnimatorStateChanged changedValue) => Debug.Log($"layerIndex={c
}
```



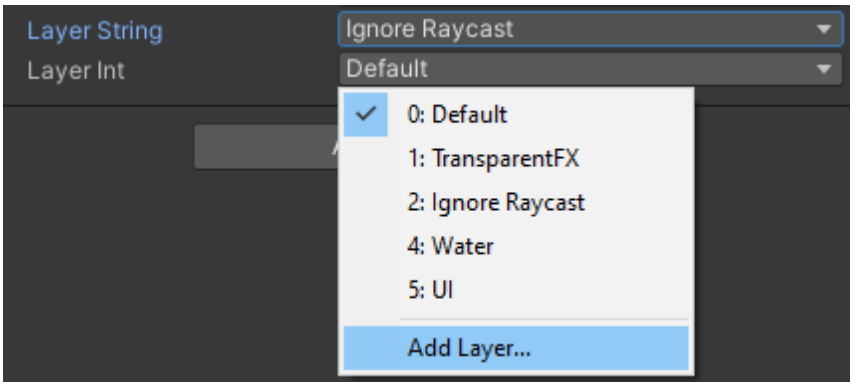
4.4.11. `Layer`

A dropdown selector for layer.

- AllowMultiple: No

Note: want a bitmask layer selector? Unity already has it. Just use `public LayerMask myLayerMask;`

```
public class LayerAttributeExample: MonoBehaviour
{
    [Layer] public string layerString;
    [Layer] public int layerInt;

    // Unity supports multiple layer selector
    public LayerMask myLayerMask;
}
```

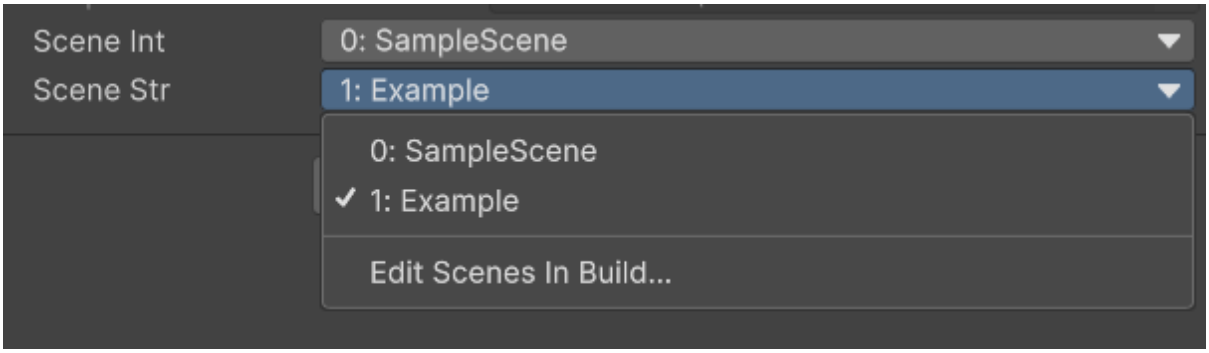


4.4.12. `Scene`

A dropdown selector for a scene in the build list, plus a "Edit Scenes In Build..." option to directly open the "Build Settings" window where you can change building scenes.

- AllowMultiple: No

```
public class SceneExample: MonoBehaviour
{
    [Scene] public int _sceneInt;
    [Scene] public string _sceneString;
}
```

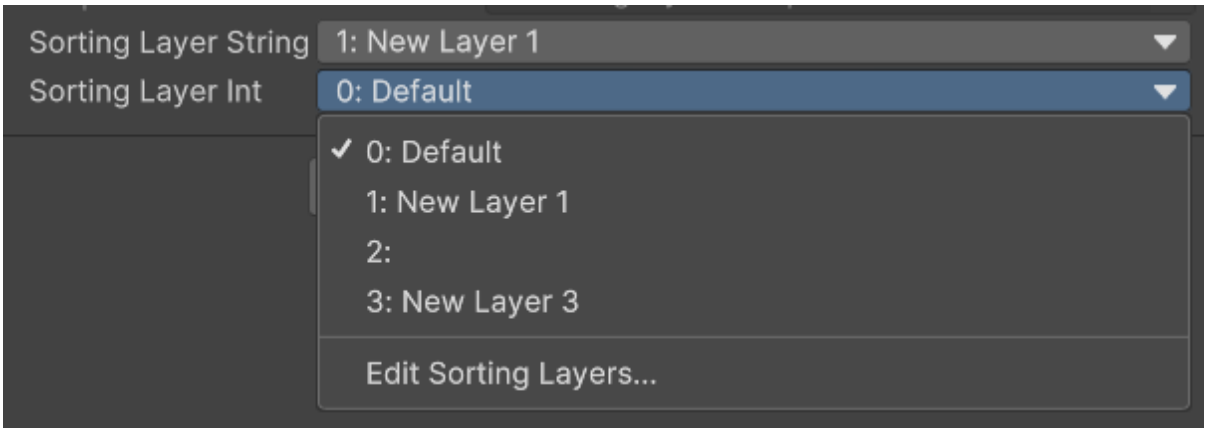


4.4.13. `SortingLayer`

A dropdown selector for sorting layer, plus a "Edit Sorting Layers..." option to directly open "Sorting Layers" tab from "Tags & Layers" inspector where you can change sorting layers.

- AllowMultiple: No

```
public class SortingLayerExample: MonoBehaviour
{
    [SortingLayer] public string _sortingLayerString;
    [SortingLayer] public int _sortingLayerInt;
}
```

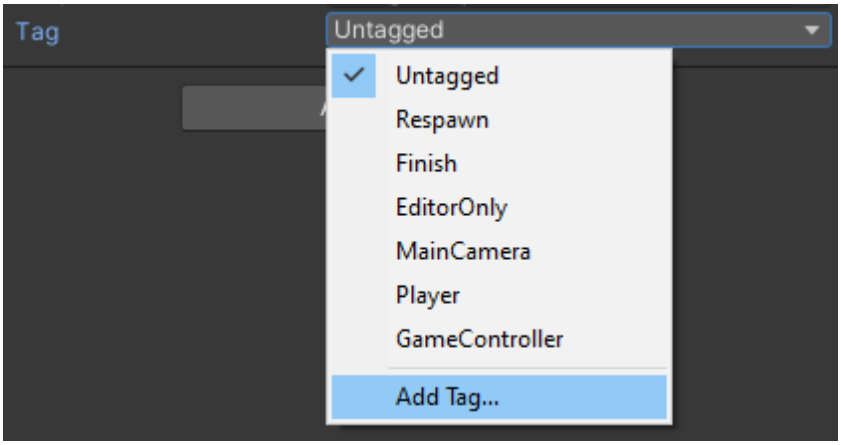


4.4.14. `Tag`

A dropdown selector for a tag.

- AllowMultiple: No

```
public class TagExample: MonoBehaviour
{
    [Tag] public string tag;
}
```

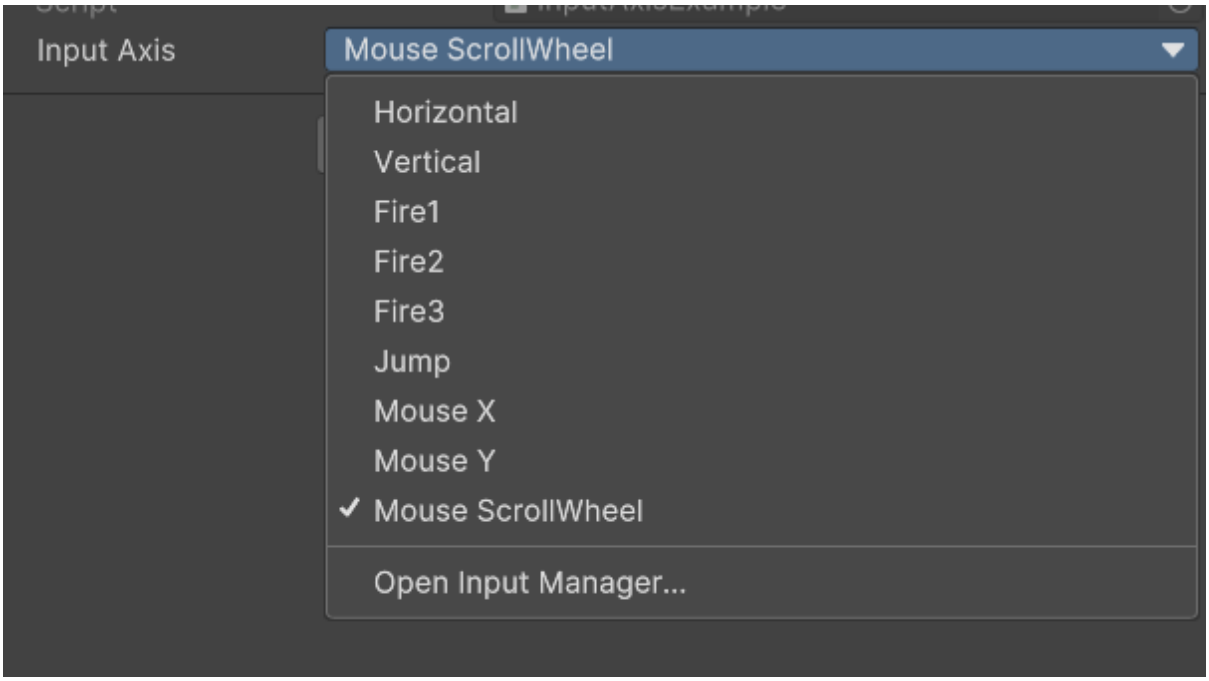


4.4.15. `InputAxis`

A string dropdown selector for an input axis, plus a "Open Input Manager..." option to directly open "Input Manager" tab from "Project Settings" window where you can change input axes.

- AllowMultiple: No

```
public class InputAxisExample: MonoBehaviour
{
    [InputAxis] public string inputAxis;
}
```

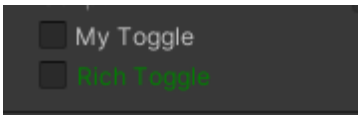


4.4.16. `LeftToggle`

A toggle button on the left of the bool field. Only works on boolean field.

IMGUI: To use with `RichLabel`, you need to add 6 spaces ahead as a hack

```
public class LeftToggleExample: MonoBehaviour
{
    [LeftToggle] public bool myToggle;
    [LeftToggle, RichLabel("      <color=green><label />")] public bool richToggle;
}
```



4.4.17. `CurveRange`

A curve drawer for `AnimationCurve` which allow to set bounds and color

Override 1:

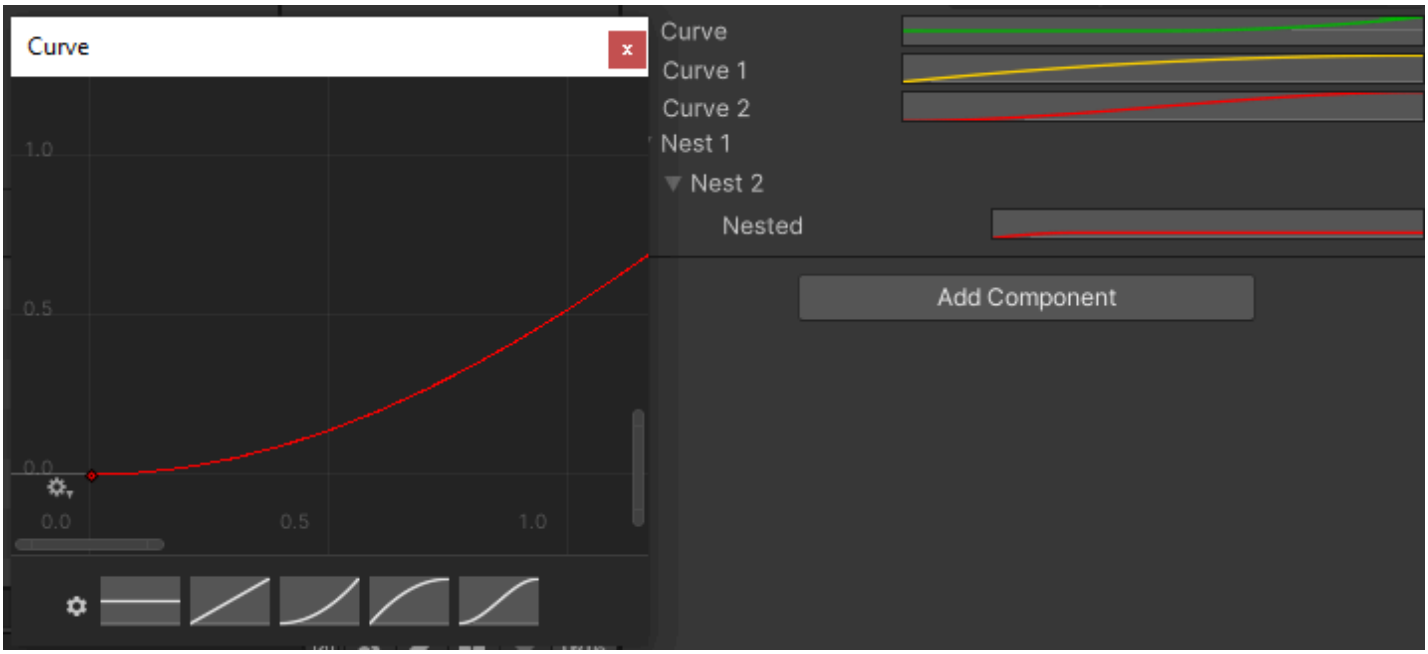
- `Vector2 min = Vector2.zero` bottom left for bounds
- `Vector2 max = Vector2.one` top right for bounds
- `EColor color = EColor.Green` curve line color

Override 2:

- `float minX = 0f` bottom left x for bounds
- `float minY = 0f` bottom left y for bounds
- `float maxX = 1f` top right x for bounds

- `float maxY = 1f` top right y for bounds
- `EColor color = EColor.Green` curve line color

```
public class CurveRangeExample: MonoBehaviour
{
    [CurveRange(-1, -1, 1, 1)]
    public AnimationCurve curve;

    [CurveRange(EColor.Orange)]
    public AnimationCurve curve1;

    [CurveRange(0, 0, 5, 5, EColor.Red)]
    public AnimationCurve curve2;
}
```



4.4.18. `ProgressBar`

A progress bar for `float` or `int` field. This behaves like a slider but more fancy.

Note: Unlike NaughtyAttributes (which is read-only), this **is interactable** .

Parameters:

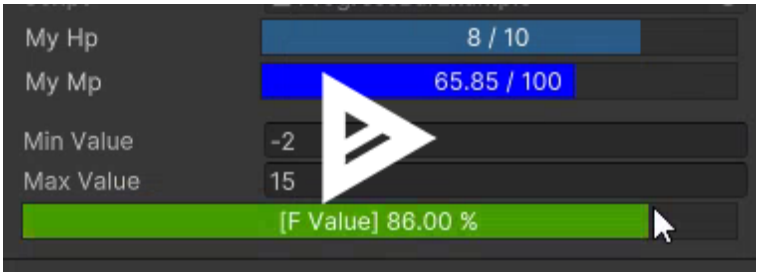
- (Optional) `float minValue=0` | `string minCallback=null` : minimum value of the slider
- `float maxValue=100` | `string maxCallback=null` : maximum value of the slider
- `float step=-1` : the growth step of the slider, `<= 0` means no limit.
- `EColor color=EColor.OceanicSlate` : filler color
- `EColor backgroundColor=EColor.CharcoalGray` : background color

- `string colorCallback=null` : a callback or property name for the filler color. The function must return a `EColor` , `Color` , a name of `EColor` / `Color` , or a hex color string (starts with `#`). This will override `color` parameter.
- `string backgroundColorCallback=null` : a callback or property name for the background color.
- `string titleCallback=null` : a callback for displaying the title. The function signature is:

  ```
  string TitleCallback(float curValue, float min, float max, string label);
  ```

  rich text is not supported here

```
public class ProgressBarExample: MonoBehaviour
{
    [ProgressBar(10)] public int myHp;
    // control step for float rather than free value
    [ProgressBar(0, 100f, step: 0.05f, color: EColor.Blue)] public float myMp;

    [Space]
    public int minValue;
    public int maxValue;

    [ProgressBar(nameof(minValue)
            , nameof(maxValue)  // dynamic min/max
            , step: 0.05f
            , backgroundColorCallback: nameof(BackgroundColor)  // dynamic background color
            , colorCallback: nameof(FillColor)  // dynamic fill color
            , titleCallback: nameof(Title)  // dynamic title, does not support rich label
        ),
    ]
    [RichLabel(null)]  // make this full width
    public float fValue;

    private EColor BackgroundColor() => fValue <= 0? EColor.Brown: EColor.CharcoalGray;

    private Color FillColor() => Color.Lerp(Color.yellow, EColor.Green.GetColor(), Mathf.Pow(Ma

    private string Title(float curValue, float min, float max, string label) => curValue < 0 ?
}
```



4.4.19. `ResourcePath`

A tool to pick an resource path (a string) with:

1. required types or interfaces
2. display a type instead of showing a string
3. pick a suitable object using a custom picker

Parameters:

- `EStr eStr = EStr.Resource` : which kind of string value you expected:
  - `Resource` : a resource path
  - `AssetDatabase` : an asset path. You should NOT use this unless you know what you are doing.
  - `Guid` : the GUID of the target object. You should NOT use this unless you know what you are doing.
- `bool freeSign=false` :

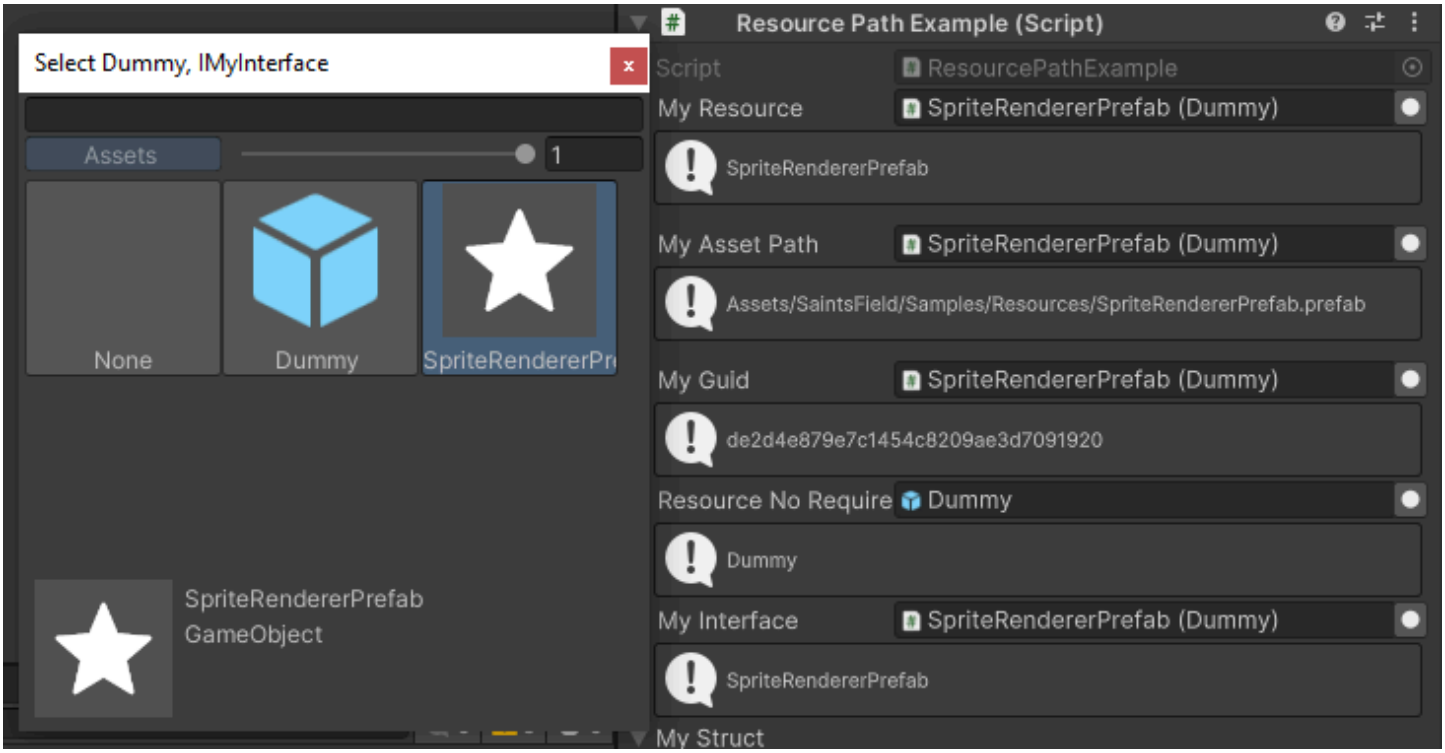
  `true` to allow to sign any object, and gives a message if the signed value does not match.

  `false` to only allow to sign matched object, and trying to prevent the change if it's illegal.
- `bool customPicker=true` : use a custom object pick that only display objects which meet the requirements
- `Type compType` : the type of the component. It can be a component, or an object like `GameObject` , `Sprite` . The field will be this type. It can NOT be an interface
- `params Type[] requiredTypes` : a list of required components or interfaces you want. Only objects with all of the types can be signed.
- AllowMultiple: No

Known Issue : IMGUI, manually sign a null object by using Unity's default pick will sign an empty string instead of null. Use custom pick to avoid this inconsistency.

```
// resource: display as a MonoScript, requires a BoxCollider
[ResourcePath(typeof(Dummy), typeof(BoxCollider))]
[InfoBox(nameof(myResource), true)]
public string myResource;

// AssetDatabase path
[Space]
[ResourcePath(EStr.AssetDatabase, typeof(Dummy), typeof(BoxCollider))]
[InfoBox(nameof(myAssetPath), true)]
public string myAssetPath;
```

```
// GUID
[Space]
[ResourcePath(EStr.Guid, typeof(Dummy), typeof(BoxCollider))]
[InfoBox(nameof(myGuid), true)]
public string myGuid;

// prefab resource
[ResourcePath(typeof(GameObject))]
[InfoBox(nameof(resourceNoRequire), true)]
public string resourceNoRequire;

// requires to have a Dummy script attached, and has interface IMyInterface
[ResourcePath(typeof(Dummy), typeof(IMyInterface))]
[InfoBox(nameof(myInterface), true)]
public string myInterface;
```



4.5. Field Utilities

4.5.1. `AssetPreview`

Show an image preview for prefabs, Sprite, Texture2D, etc. (Internally use `AssetPreview.GetAssetPreview`)

Note: Sometimes `AssetPreview.GetAssetPreview` simply does not return a correct preview image or returns an empty image. When no image returns, nothing is shown. If an empty image returns, an empty rect is shown. This can not be fixed unless Unity decides to fix it.

Note: Recommended to use `AboveImage` / `BelowImage` for image/sprite/texture2D.

- `int width=-1`

preview width, -1 for original image size that returned by Unity. If it's greater than current view width, it'll be scaled down to fit the view. Use `int.MaxValue` to always fit the view width.

- `int height=-1`

  preview height, -1 for auto resize (with the same aspect) using the width

- `EAlign align=EAlign.End`

  Align of the preview image. Options are `Start`, `End`, `Center`, `FieldStart`

- `bool above=false`

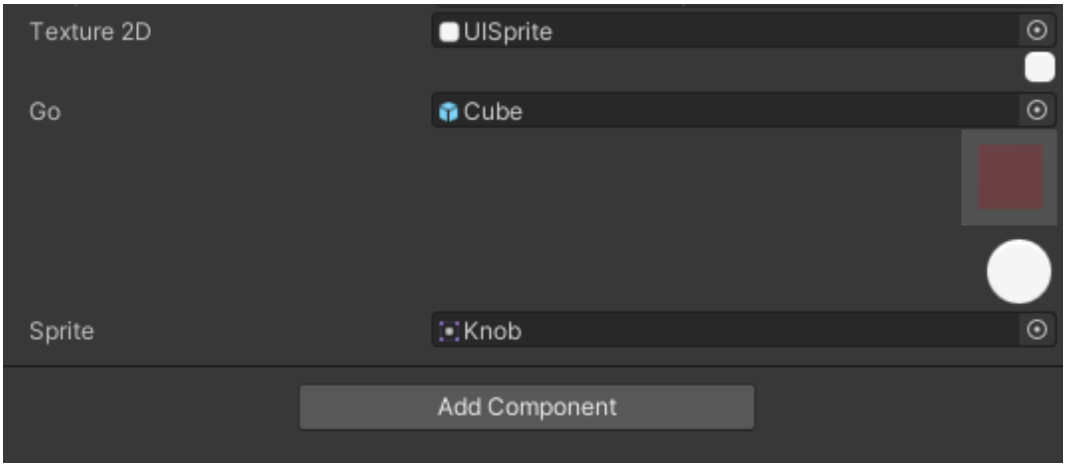
  if true, render above the field instead of below

- `string groupBy=""`

  See the `GroupBy` section

- AllowMultiple: No

```
public class AssetPreviewExample: MonoBehaviour
{
    [AssetPreview(20, 100)] public Texture2D _texture2D;
    [AssetPreview(50)] public GameObject _go;
    [AssetPreview(above: true)] public Sprite _sprite;
}
```



4.5.2. `AboveImage` / `BelowImage`

Show an image above/below the field.

- `string image = null`

  An image to display. This can be a property or a callback, which returns a `Sprite`, `Texture2D`, `SpriteRenderer`, `UI.Image`, `UI.RawImage` or `UI.Button`.

If it's null, it'll try to get the image from the field itself.

- `string maxWidth=-1`

  preview max width, -1 for original image size. If it's greater than current view width, it'll be scaled down to fit the view. . Use `int.MaxValue` to always fit the view width.

- `int maxHeight=-1`

  preview max height, -1 for auto resize (with the same aspect) using the width

- `EAlign align=EAlign.Start`

  Align of the preview image. Options are `Start` , `End` , `Center` , `FieldStart`

- `string groupBy=""`

  See the `GroupBy` section

- AllowMultiple: No

```
public class ShowImageExample: MonoBehaviour
{
    [AboveImage(nameof(spriteField))]
    // size and group
    [BelowImage(nameof(spriteField), maxWidth: 25, groupBy: "Below1")]
    [BelowImage(nameof(spriteField), maxHeight: 20, align: EAlign.End, groupBy: "Below1")]
    public Sprite spriteField;

    // align
    [BelowImage(nameof(spriteField), maxWidth: 20, align: EAlign.FieldStart)]
    [BelowImage(nameof(spriteField), maxWidth: 20, align: EAlign.Start)]
    [BelowImage(nameof(spriteField), maxWidth: 20, align: EAlign.Center)]
    [BelowImage(nameof(spriteField), maxWidth: 20, align: EAlign.End)]
    public string alignField;
}
```

4.5.3. `OnValueChanged`

Call a function every time the field value is changed

- `string callback` the callback function name

  It'll try to pass the new value and the index (only if it's in an array/list). You can set the corresponding parameter in your callback if you want to receive them.

- AllowMultiple: Yes

Special Note: `AnimatorState` will have a different `OnValueChanged` parameter passed in. See `AnimatorState` for more detail.

```
public class OnChangedExample : MonoBehaviour
{
    // no params
    [OnValueChanged(nameof(Changed))]
    public int value;
    private void Changed()
    {
        Debug.Log($"changed={value}");
    }

    // with params to get the new value
    [OnValueChanged(nameof(ChangedAnyType))]
    public GameObject go;

    // it will pass the index too if it's inside an array/list
    [OnValueChanged(nameof(ChangedAnyType))]
```

```
    public SpriteRenderer[] srs;

    // it's ok to set it as the super class
    private void ChangedAnyType(object anyObj, int index=-1)
    {
        Debug.Log($"changed={anyObj}@{index}");
    }
}
```

4.5.4. `ReadOnly` / `DisableIf` / `EnableIf`

`ReadOnly` equals `DisableIf` , `EnableIf` is the opposite of `DisableIf`

Arguments:

- (Optional) `EMode editorMode=EMode.Edit | EMode.Play`

  Condition: if it should be in edit mode or play mode for Editor. By default (omiting this parameter) it does not check the mode at all.

- `string by...`

  callbacks or attributes for the condition.

- AllowMultiple: Yes

For `ReadOnly` / `DisableIf` : The field will be disabled if **ALL** condition is true (`and` operation)

For `EnableIf` : The field will be enabled if **ANY** condition is true (`or` operation)

This is the same logic as the `ShowIf` / `HideIf` pair, which `DisableIf` is the major attribute:

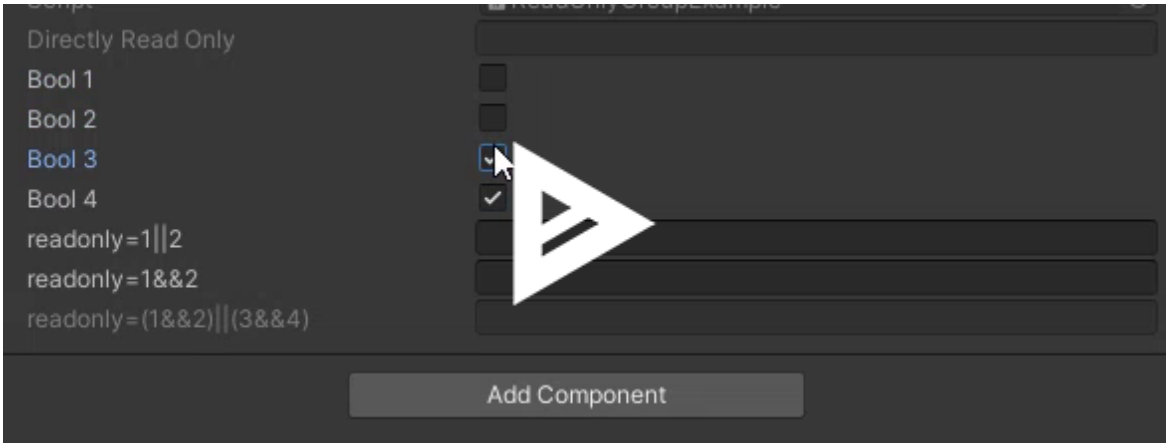
- `EnableIf(A)` == `DisableIf(!A)`
- `EnableIf(A, B)` == `EnableIf(A || B)` == `DisableIf(!(A || B))` == `DisableIf(!A && !B)`
- `[EnableIf(A), EnableIf(B)]` == `[DisableIf(!A), DisableIf(!B)]` == `DisableIf(!A || !B)` == `DisableIf(!(A && B))`

A simple example:

```
[ReadOnly(nameof(ShouldBeDisabled))] public string disableMe;

private bool ShouldBeDisabled  // change the logic here
{
    return true;
}
```

A more complex example:

```
public class ReadOnlyGroupExample: MonoBehaviour
{
    [ReadOnly] public string directlyReadOnly;

    [SerializeField] private bool _bool1;
    [SerializeField] private bool _bool2;
    [SerializeField] private bool _bool3;
    [SerializeField] private bool _bool4;

    [SerializeField]
    [ReadOnly(nameof(_bool1))]
    [ReadOnly(nameof(_bool2))]
    [RichLabel("readonly=1||2")]
    private string _ro1and2;


    [SerializeField]
    [ReadOnly(nameof(_bool1), nameof(_bool2))]
    [RichLabel("readonly=1&&2")]
    private string _ro1or2;


    [SerializeField]
    [ReadOnly(nameof(_bool1), nameof(_bool2))]
    [ReadOnly(nameof(_bool3), nameof(_bool4))]
    [RichLabel("readonly=(1&&2)||(3&&4)")]
    private string _ro1234;
}
```



EMode example:

```
public bool boolVal;

[DisableIf(EMode.Edit)] public string disEditMode;
[DisableIf(EMode.Play)] public string disPlayMode;

[DisableIf(EMode.Edit, nameof(boolVal))] public string disEditAndBool;
[DisableIf(EMode.Edit), DisableIf(nameof(boolVal))] public string disEditOrBool;
```

```
[EnableIf(EMode.Edit)] public string enEditMode;
[EnableIf(EMode.Play)] public string enPlayMode;

[EnableIf(EMode.Edit, nameof(boolVal))] public string enEditOrBool;
// dis=!editor || dis=!bool => en=editor&&bool
[EnableIf(EMode.Edit), EnableIf(nameof(boolVal))] public string enEditAndBool;
```

4.5.5. `Required`

Reminding a given reference type field to be required.

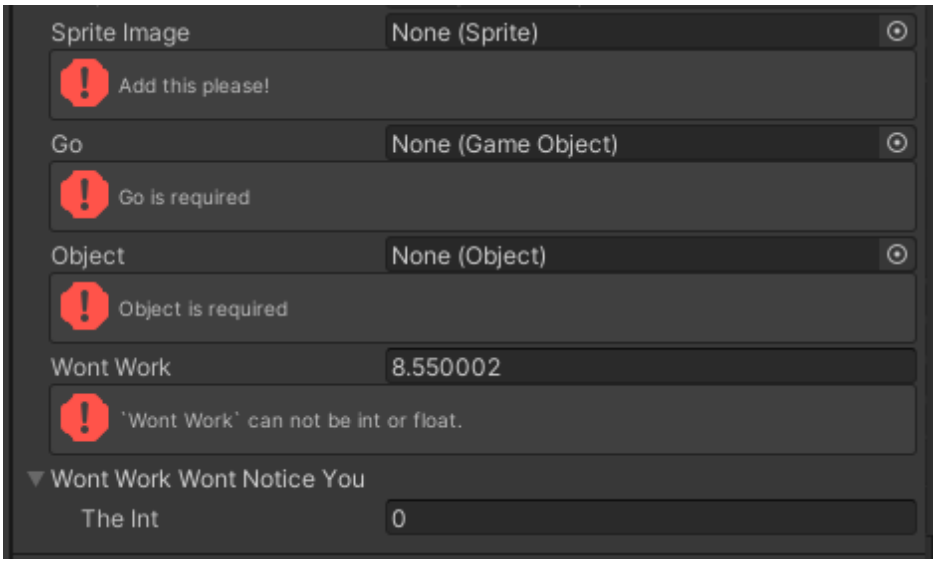
This will check if the field value is a `truly` value, which means:

1. `ValuedType` like `struct` will always be `truly` because `struct` is not nullable and Unity will fill a default value for it no matter what
2. It works on reference type and will NOT skip Unity's life-circle null check
3. You may not want to use it on `int`, `float` (because only `0` is not `truly`) or `bool`, but it's still allowed if you insist

Parameters:

- `string errorMessage = null` Error message. Default is `{label} is required`
- AllowMultiple: No

```
public class RequiredExample: MonoBehaviour
{
    [Required("Add this please!")] public Sprite _spriteImage;
    // works for the property field
    [field: SerializeField, Required] public GameObject Go { get; private set; }
    [Required] public UnityEngine.Object _object;
    [SerializeField, Required] private float _wontWork;

    [Serializable]
    public struct MyStruct
    {
        public int theInt;
    }

    [Required]
    public MyStruct warnsYouNow;
}
```



4.5.6. `ValidateInput`

Validate the input of the field when the value changes.

- `string callback` is the callback function to validate the data.

  Parameters :

  i. If the function accepts no arguments, then no argument will be passed
  ii. If the function accepts required arguments, the first required argument will receive the field's value. If there is another required argument and the field is inside a list/array, the index will be passed.
  iii. If the function only has optional arguments, it will try to pass the field's value and index if possible. Otherwise the default value of the parameter will be passed.

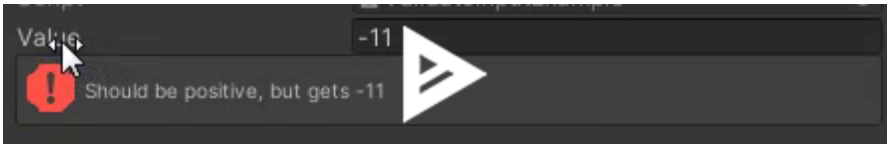
  Return :

  i. If return type is `string` , then `null` or empty string for valid, otherwise, the string will be used as the error message
  ii. If return type is `bool` , then `true` for valid, `false` for invalid with message "`{label}` is invalid`"

- AllowMultiple: Yes

```
public class ValidateInputExample : MonoBehaviour
{
    // string callback
    [ValidateInput(nameof(OnValidateInput))]
    public int _value;
    private string OnValidateInput() => _value < 0 ? $"Should be positive, but gets {_value}" :

    // property validate
    [ValidateInput(nameof(boolValidate))]
    public bool boolValidate;
```

```
    // bool callback
    [ValidateInput(nameof(BoolCallbackValidate))]
    public string boolCallbackValidate;
    private bool BoolCallbackValidate() => boolValidate;

    // with callback params
    [ValidateInput(nameof(ValidateWithReqParams))]
    public int withReqParams;
    private string ValidateWithReqParams(int v) => $"ValidateWithReqParams: {v}";

    // with optional callback params
    [ValidateInput(nameof(ValidateWithOptParams))]
    public int withOptionalParams;

    private string ValidateWithOptParams(string sth="a", int v=0) => $"ValidateWithOptionalPara

    // with array index callback
    [ValidateInput(nameof(ValidateValArr))]
    public int[] valArr;

    private string ValidateValArr(int v, int index) => $"ValidateValArr[{index}]: {v}";
}
```



4.5.7. `ShowIf` / `HideIf`

Show or hide the field based on a condition.

Arguments:

- (Optional) `EMode editorMode=EMode.Edit | EMode.Play`

  Condition: if it should be in edit mode or play mode for Editor. By default (omiting this parameter) it does not check the mode at all.

- `string by...`

  callbacks or attributes for the condition.

- AllowMultiple: Yes

You can use multiple `ShowIf` , `HideIf` , and even a mix of the two: the field will be shown if **ANY** of them is shown.(`or` operation)

For example, `[ShowIf(A...), ShowIf(B...)]` will be shown if `ShowIf(A...) || ShowIf(B...)` is true.

`HideIf` is the opposite of `ShowIf`. Please note "the opposite" is like the logic operation, like `!(A && B)` is `!A || !B`, `!(A || B)` is `!A && !B`.

- `HideIf(A)` == `ShowIf(!A)`
- `HideIf(A, B)` == `HideIf(A || B)` == `ShowIf(!(A || B))` == `ShowIf(!A && !B)`
- `[Hideif(A), HideIf(B)]` == `[ShowIf(!A), ShowIf(!B)]` == `ShowIf(!A || !B)` == `ShowIf(!(A && B))`

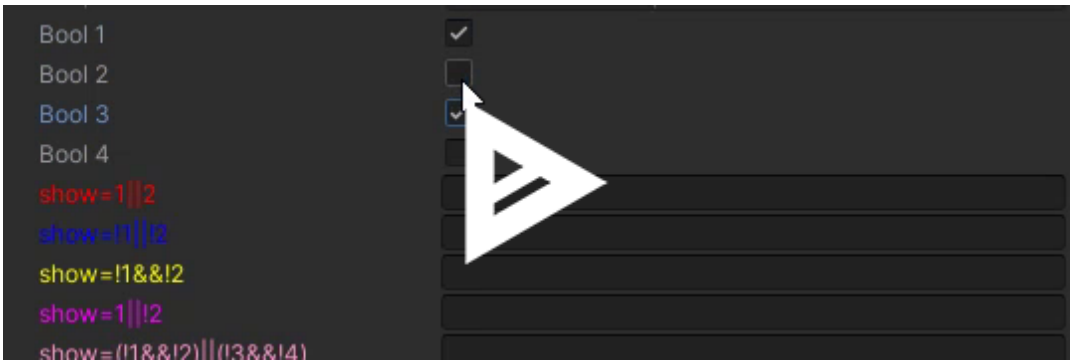
A simple example:

```
[ShowIf(nameof(ShouldShow))]
public int showMe;

public bool ShouldShow()  // change the logic here
{
    return true;
}
```

A full featured example:

```
public class ShowHideExample: MonoBehaviour
{
    public bool _bool1;
    public bool _bool2;
    public bool _bool3;
    public bool _bool4;

    [ShowIf(nameof(_bool1))]
    [ShowIf(nameof(_bool2))]
    [RichLabel("<color=red>show=1||2")]
    public string _showIf1Or2;


    [ShowIf(nameof(_bool1), nameof(_bool2))]
    [RichLabel("<color=green>show=1&&2")]
    public string _showIf1And2;

    [HideIf(nameof(_bool1))]
    [HideIf(nameof(_bool2))]
    [RichLabel("<color=blue>show=!1||!2")]
    public string _hideIf1Or2;


    [HideIf(nameof(_bool1), nameof(_bool2))]
    [RichLabel("<color=yellow>show=!(1||2)=!1&&!2")]
    public string _hideIf1And2;

    [ShowIf(nameof(_bool1))]
    [HideIf(nameof(_bool2))]
```

```
    [RichLabel("<color=magenta>show=1||!2")]
    public string _showIf1OrNot2;

    [ShowIf(nameof(_bool1), nameof(_bool2))]
    [ShowIf(nameof(_bool3), nameof(_bool4))]
    [RichLabel("<color=orange>show=(1&&2)||(3&&4)")]
    public string _showIf1234;

    [HideIf(nameof(_bool1), nameof(_bool2))]
    [HideIf(nameof(_bool3), nameof(_bool4))]
    [RichLabel("<color=pink>show=!(1||2)||!(3||4)=(!1&&!2)||(!3&&!4)")]
    public string _hideIf1234;
}
```



Example about EMode:

```
public bool boolValue;

[ShowIf(EMode.Edit)] public string showEdit;
[ShowIf(EMode.Play)] public string showPlay;

[ShowIf(EMode.Edit, nameof(boolValue))] public string showEditAndBool;
[ShowIf(EMode.Edit), ShowIf(nameof(boolValue))] public string showEditOrBool;

[HideIf(EMode.Edit)] public string hideEdit;
[HideIf(EMode.Play)] public string hidePlay;

[HideIf(EMode.Edit, nameof(boolValue))] public string hideEditOrBool;
[HideIf(EMode.Edit), HideIf(nameof(boolValue))] public string hideEditAndBool;
```

4.5.8. `MinValue` / `MaxValue`

Limit for int/float field

They have the same overrides:

- `float value`: directly limit to a number value
- `string valueCallback`: a callback or property for limit

- AllowMultiple: Yes

```
public class MinMaxExample: MonoBehaviour
{
    public int upLimit;

    [MinValue(0), MaxValue(nameof(upLimit))] public int min0Max;
    [MinValue(nameof(upLimit)), MaxValue(10)] public float fMinMax10;
}
```

4.5.9. `GetComponent`

Automatically sign a component to a field, if the field value is null and the component is already attached to current target. (First one found will be used)

- `Type compType = null`

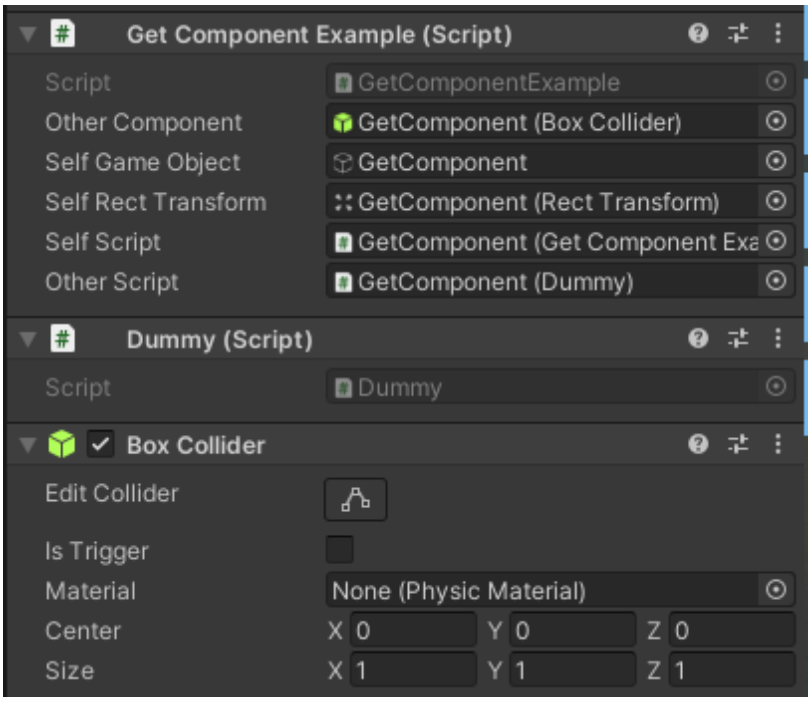
  The component type to sign. If null, it'll use the field type.

- `string groupBy = ""`

  For error message grouping.

- AllowMultiple: No

```
public class GetComponentExample: MonoBehaviour
{
    [GetComponent] public BoxCollider otherComponent;
    [GetComponent] public GameObject selfGameObject;  // get the GameObject itself
    [GetComponent] public RectTransform selfRectTransform;  // useful for UI

    [GetComponent] public GetComponentExample selfScript;  // yeah you can get your script itse
    [GetComponent] public Dummy otherScript;  // other script
}
```



4.5.10. `GetComponentInChildren`

Automatically sign a component to a field, if the field value is null and the component is already attached to its child GameObjects. (First one found will be used)

NOTE: Unlike `GetComponentInChildren` by Unity, this will **NOT** check the target object itself.

- `bool includeInactive = false`

  Should inactive children be included? `true` to include inactive children.

- `Type compType = null`

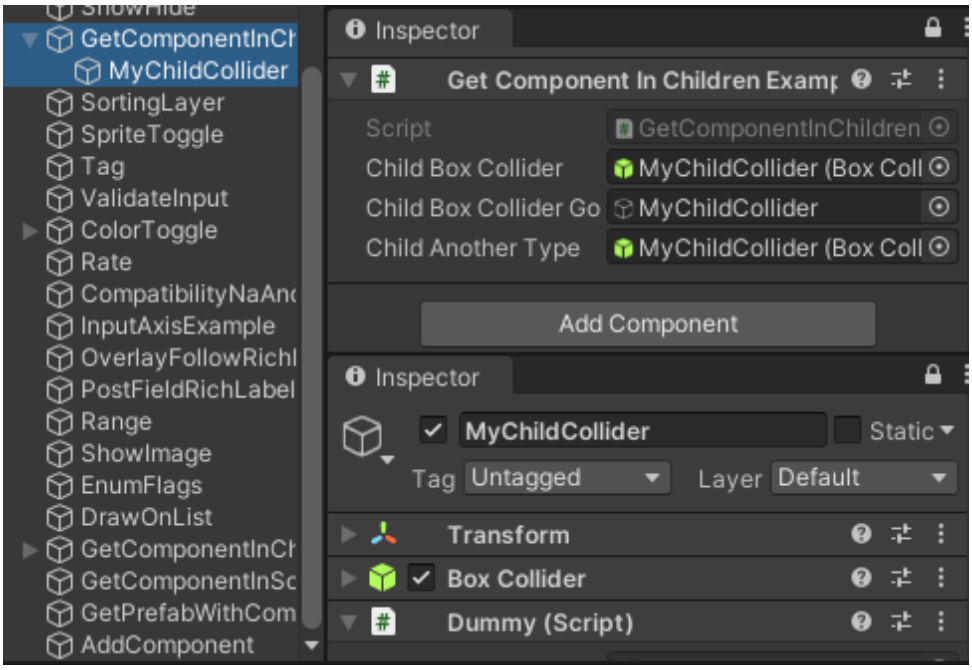
  The component type to sign. If null, it'll use the field type.

- `string groupBy = ""`

  For error message grouping.

- AllowMultiple: No

```
public class GetComponentInChildrenExample: MonoBehaviour
{
    [GetComponentInChildren] public BoxCollider childBoxCollider;
    // by setting compType, you can sign it as a different type
    [GetComponentInChildren(compType: typeof(Dummy))] public BoxCollider childAnotherType;
    // and GameObject field works too
    [GetComponentInChildren(compType: typeof(BoxCollider))] public GameObject childBoxColliderG
}
```

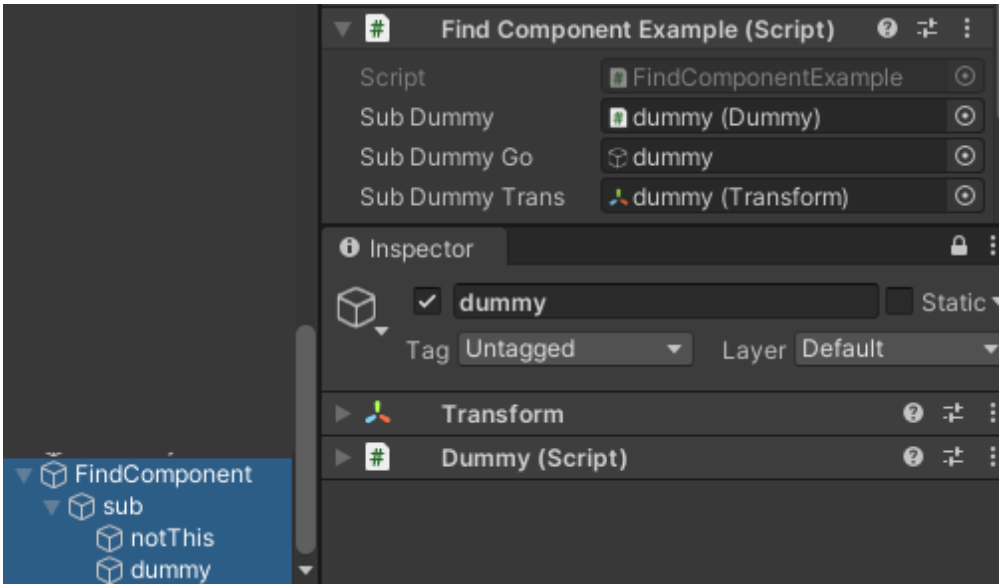


4.5.11. `FindComponent`

Automatically find a component under the current target. This is very similar to Unity's `transform.Find`, except it accepts many paths, and it's returning value is not limited to `transform`

- `string path` a path to search
- `params string[] paths` more paths to search
- AllowMultiple: Yes but not necessary

```
public class FindComponentExample: MonoBehaviour
{
    [FindComponent("sub/dummy")] public Dummy subDummy;
    [FindComponent("sub/dummy")] public GameObject subDummyGo;
    [FindComponent("sub/noSuch", "sub/dummy")] public Transform subDummyTrans;
}
```



4.5.12. `GetComponentInParent` / `GetComponentInParents`

Automatically sign a component to a field, if the field value is null and the component is already attached to its parent GameObject(s). (First one found will be used)

- `Type compType = null`

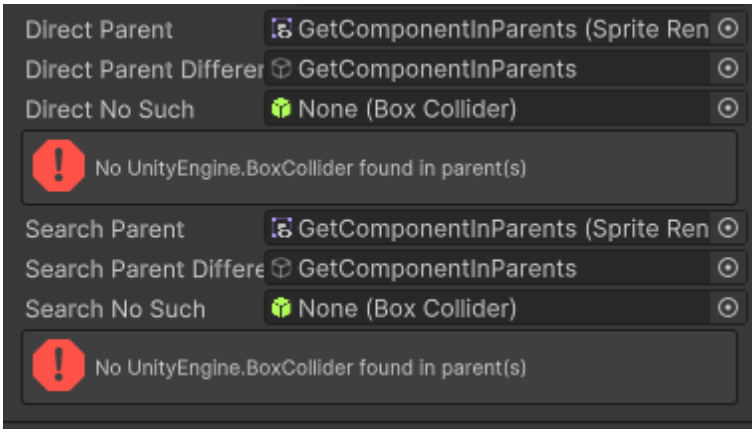
  The component type to sign. If null, it'll use the field type.

- `string groupBy = ""`

  For error message grouping.

- AllowMultiple: No

```
public class GetComponentInParentsExample: MonoBehaviour
{
    [GetComponentInParent] public SpriteRenderer directParent;
    [GetComponentInParent(typeof(SpriteRenderer))] public GameObject directParentDifferentType;
    [GetComponentInParent] public BoxCollider directNoSuch;

    [GetComponentInParents] public SpriteRenderer searchParent;
    [GetComponentInParents(typeof(SpriteRenderer))] public GameObject searchParentDifferentType
    [GetComponentInParents] public BoxCollider searchNoSuch;
}
```



4.5.13. `GetComponentInScene`

Automatically sign a component to a field, if the field value is null and the component is in the currently opened scene. (First one found will be used)

- `bool includeInactive = false`

  Should inactive GameObject be included? `true` to include inactive GameObject.

- `Type compType = null`

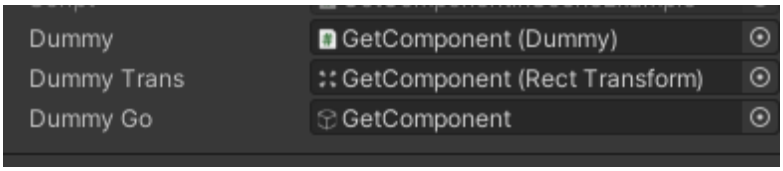
  The component type to sign. If null, it'll use the field type.

- `string groupBy = ""`

  For error message grouping.

- AllowMultiple: No

```
public class GetComponentInSceneExample: MonoBehaviour
{
    [GetComponentInScene] public Dummy dummy;
    // by setting compType, you can sign it as a different type
    [GetComponentInScene(compType: typeof(Dummy))] public RectTransform dummyTrans;
    // and GameObject field works too
    [GetComponentInScene(compType: typeof(Dummy))] public GameObject dummyGo;
}
```



4.5.14. `GetComponentByPath`

Automatically sign a component to a field by a given path.

- (Optional) `EGetComp config`

  Options are:

  - `EGetComp.ForceResign`: when the target changed (e.g. you delete/create one), automatically resign the new correct component.
  - `EGetComp.NoResignButton`: do not display a resign button when the target mismatches.

- `string paths...`

  Paths to search.

- AllowMultiple: Yes. But not necessary.

The `path` is a bit like html's `XPath` but with less functions:

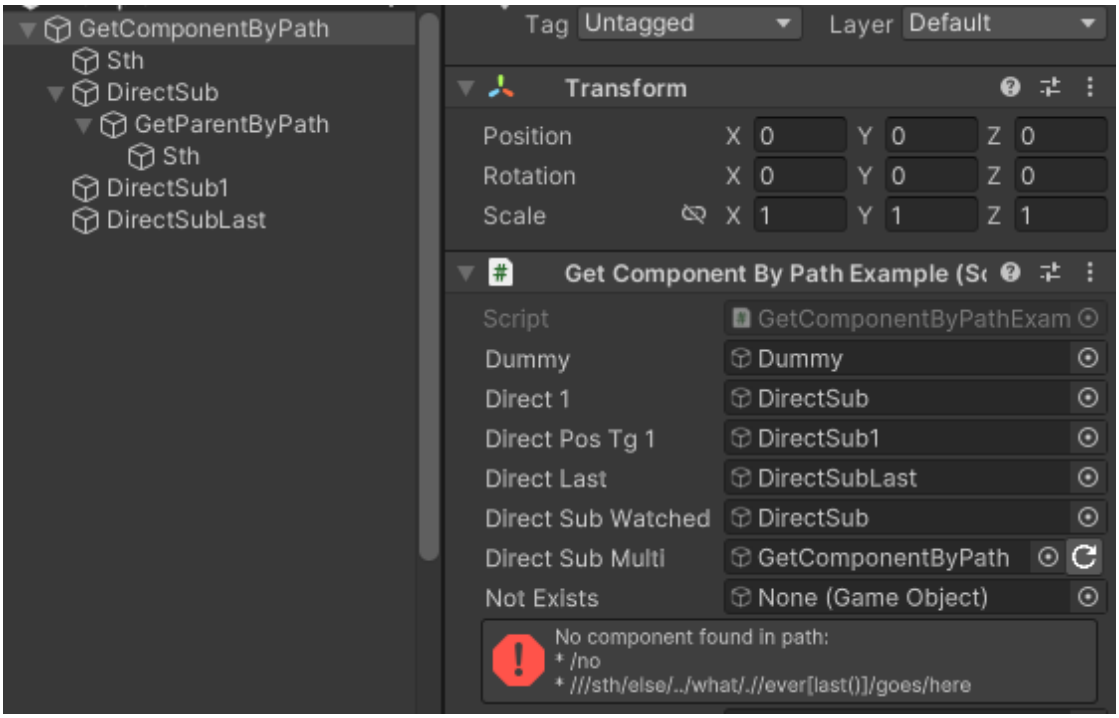
Path	Meaning
`/`	Separator. Using at start means the root of the current scene.
`//`	Separator. Any descendant children
`.`	Node. Current node
`..`	Node. Parent node

Path	Meaning
`*`	All nodes
name	Node. Any nodes with this name
`[last()]`	Index Filter. Last of results
`[index() > 1]`	Index Filter. Node index that is greater than 1
`[0]`	Index Filter. First node in the results

For example:

- `./sth` or `sth` : direct child object of current object named `sth`
- `.//sth` : any descendant child under current. (descendant::sth)
- `..//sth` : first go to parent, then find the direct child named `sth`
- `/sth` : top level node in current scene named `sth`
- `//sth` : first go to top level, then find the direct child named `sth`
- `///sth` : first go to top level, then find any node named `sth`
- `./get/sth[1]` : the child named `get` of current node, then the second node named `sth` in the direct children list of `get`

```
public class GetComponentByPathExample: MonoBehaviour
{
    // starting from root, search any object with name "Dummy"
    [GetComponentByPath("///Dummy")] public GameObject dummy;
    // first child of current object
    [GetComponentByPath("./*[1]")] public GameObject direct1;
    // child of current object which has index greater than 1
    [GetComponentByPath("./*[index() > 1]")] public GameObject directPosTg1;
    // last child of current object
    [GetComponentByPath("./*[last()]")] public GameObject directLast;
    // re-sign the target if mis-match
    [GetComponentByPath(EGetComp.NoResignButton | EGetComp.ForceResign, "./DirectSub")] public
    // without "ForceResign", it'll display a reload button if mis-match
    // with multiple paths, it'll search from left to right
    [GetComponentByPath("/no", "./DirectSub1")] public GameObject directSubMulti;
    // if no match, it'll show an error message
    [GetComponentByPath("/no", "///sth/else/../what/.//ever[last()]/goes/here")] public GameObj
}
```



4.5.15. `GetPrefabWithComponent`

Automatically sign a prefab to a field, if the field value is null and the prefab has the component. (First one found will be used)

Recommended to use it with `FieldType`!

- `Type compType = null`

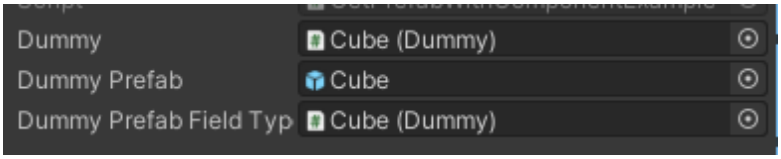
  The component type to sign. If null, it'll use the field type.

- `string groupBy = ""`

  For error message grouping.

- AllowMultiple: No

```
public class GetPrefabWithComponentExample: MonoBehaviour
{
    [GetPrefabWithComponent] public Dummy dummy;
    // get the prefab itself
    [GetPrefabWithComponent(compType: typeof(Dummy))] public GameObject dummyPrefab;
    // works so good with `FieldType`
    [GetPrefabWithComponent(compType: typeof(Dummy)), FieldType(typeof(Dummy))] public GameObje
}
```



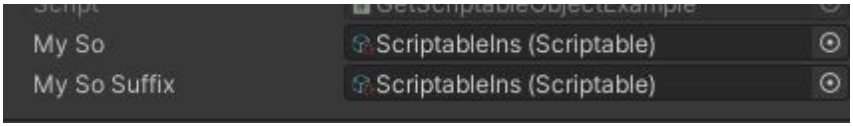
4.5.16. `GetScriptableObject`

Automatically sign a `ScriptableObject` file to this field. (First one found will be used)

Recommended to use it with `Expandable` !

- `string pathSuffix=null` the path suffix for this `ScriptableObject` . `null` for no limit. for example: if it's `/Resources/mySo` , it will only sign the file whose path is ends with `/Resources/mySo.asset` , like `Assets/proj/Resources/mySo.asset`
- AllowMultiple: No

```
public class GetScriptableObjectExample: MonoBehaviour
{
    [GetScriptableObject] public Scriptable mySo;
    [GetScriptableObject("RawResources/ScriptableIns")] public Scriptable mySoSuffix;
}
```



4.5.17. `AddComponent`

Automatically add a component to the current target if the target does not have this component. (This will not sign the component added)

Recommended to use it with `GetComponent` !

- `Type compType = null`

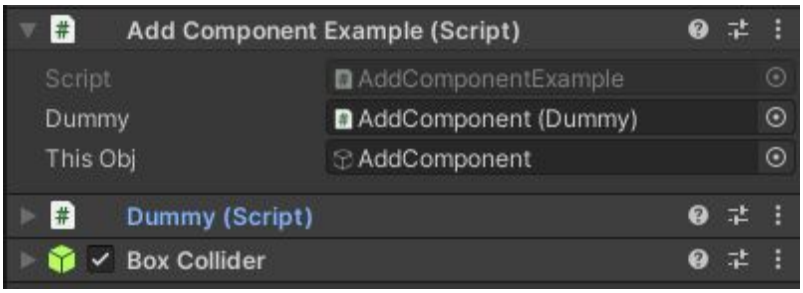
  The component type to add. If null, it'll use the field type.

- `string groupBy = ""`

  For error message grouping.

- AllowMultiple: Yes

```
public class AddComponentExample: MonoBehaviour
{
    [AddComponent, GetComponent] public Dummy dummy;
    [AddComponent(typeof(BoxCollider)), GetComponent] public GameObject thisObj;
}
```



4.5.18. `ButtonAddOnClick`

Add a callback to a button's `onClick` event. Note this at this point does only supports callback with no arguments.

- `string funcName` the callback function name
- `string buttonComp=null` the button component name.

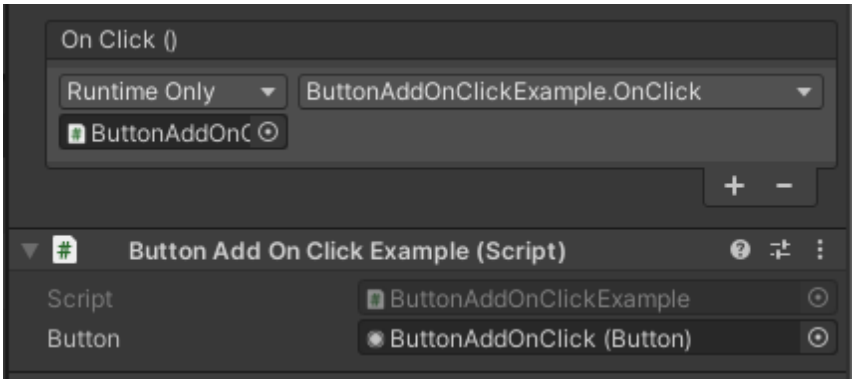
  If null, it'll try to get the button component by this order:

  i. the field itself
  ii. get the `Button` component from the field itself
  iii. get the `Button` component from the current target

  If it's not null, the search order will be:

  i. get the field of this name from current target
  ii. call a function of this name from current target

```
public class ButtonAddOnClickExample: MonoBehaviour
{
    [GetComponent, ButtonAddOnClick(nameof(OnClick))] public Button button;

    private void OnClick()
    {
        Debug.Log("Button clicked!");
    }
}
```



4.5.19. `RequireType`

Allow you to specify the required component(s) or **interface** (s) for a field.

If the signed field does not meet the requirement, it'll:

- show an error message, if `freeSign=false`
- prevent the change, if `freeSign=true`

`customPicker` will allow you to pick an object which are already meet the requirement(s).

Overload:

- `RequireTypeAttribute(bool freeSign = false, bool customPicker = true, params Type[] requiredTypes)`
- `RequireTypeAttribute(bool freeSign, params Type[] requiredTypes)`
- `RequireTypeAttribute(EPick editorPick, params Type[] requiredTypes)`
- `RequireTypeAttribute(params Type[] requiredTypes)`

For each argument:

- `bool freeSign=false`

  If true, it'll allow you to sign any object to this field, and display an error message if it does not meet the requirement(s).

  Otherwise, it will try to prevent the change.

- `bool customPicker=true`

  Show a custom picker to pick an object. The showing objects are already meet the requirement(s).

- `EPick editorPick=EPick.Assets | EPick.Scene`

  The picker type for the custom picker. `EPick.Assets` for assets, `EPick.Scene` for scene objects.

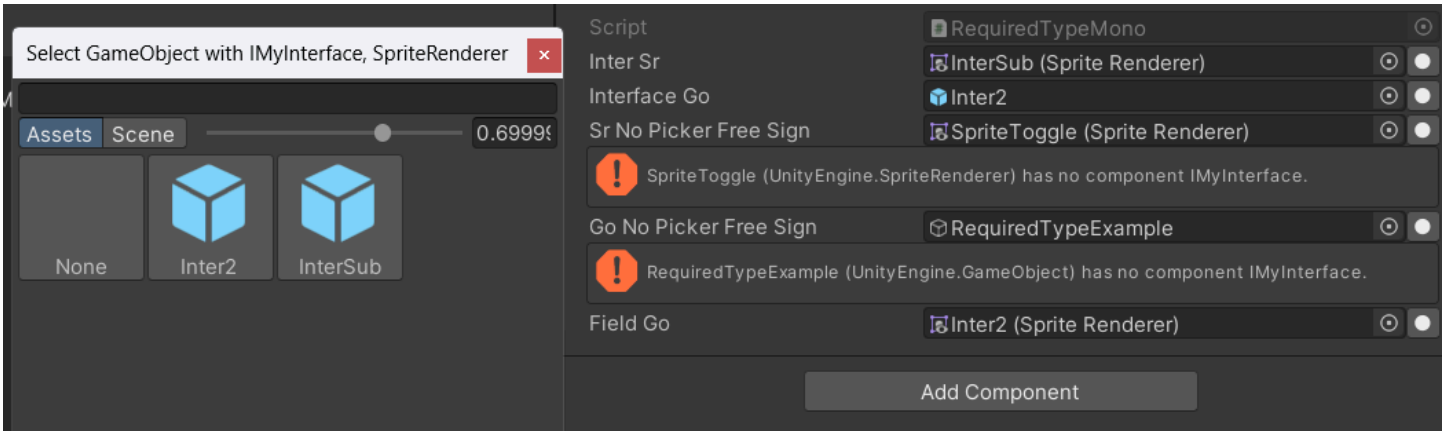
- `params Type[] requiredTypes`

  The required component(s) or interface(s) for this field.

- AllowMultiple: No

```
public interface IMyInterface {}

public class MyInter1: MonoBehaviour, IMyInterface {}
public class MySubInter: MyInter1 {}

public class MyInter2: MonoBehaviour, IMyInterface {}

[RequireType(typeof(IMyInterface))] public SpriteRenderer interSr;
```

```
[RequireType(typeof(IMyInterface), typeof(SpriteRenderer))] public GameObject interfaceGo;

[RequireType(true, typeof(IMyInterface))] public SpriteRenderer srNoPickerFreeSign;
[RequireType(true, typeof(IMyInterface))] public GameObject goNoPickerFreeSign;
```



4.5.20. `ArraySize`

A decorator that limit the size of the array or list.

Note: Because of the limitation of `PropertyDrawer`:

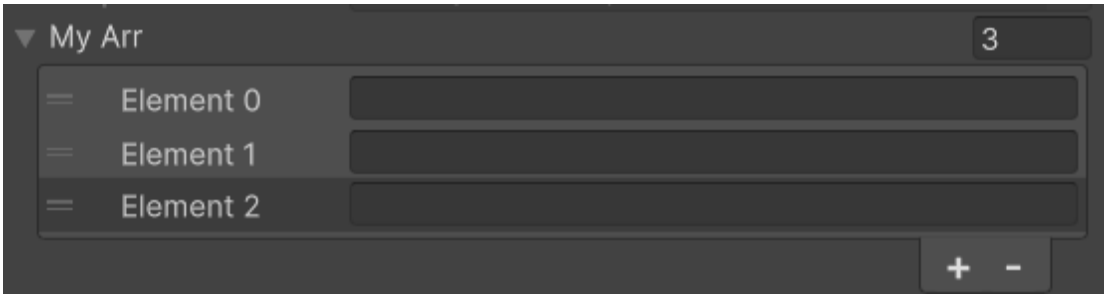
1. When the field is 0 length, it'll not be filled to target size.
2. You can always change it to 0 size.
3. Delete an element will first be deleted, then the array will duplicated the last element.
4. UI Toolkit: you might see the UI flicked when you remove an element.

(If you enable `SaintsEditor`, there is a `PlayaArraySize` that does NOT have issue 1 & 2)

Parameters:

- `int size` the size of the array or list
- `string groupBy = ""` for error message grouping
- AllowMultiple: No

```
[ArraySize(3)]
public string[] myArr;
```



4.5.21. `SaintsRow`

`SaintsRow` attribute allows you to draw `Button`, `Layout`, `ShowInInspector`, `DOTweenPlay` etc (all `SaintsEditor` attributes) in a `Serializable` object (usually a class or a struct).

This attribute does NOT need `SaintsEditor` enabled. It's an out-of-box tool.

Parameters:

- `bool inline=false`

  If true, it'll draw the `Serializable` inline like it's directly in the `MonoBehavior`

- `bool tryFixUIToolkit=defaultValue`

  Should it try to fix the UI Toolkit label width issue? (See the UI Toolkit section). By default it will try, unless you toggled the `Disable UI Toolkit Label Fix` marco, or you passed this parameter.

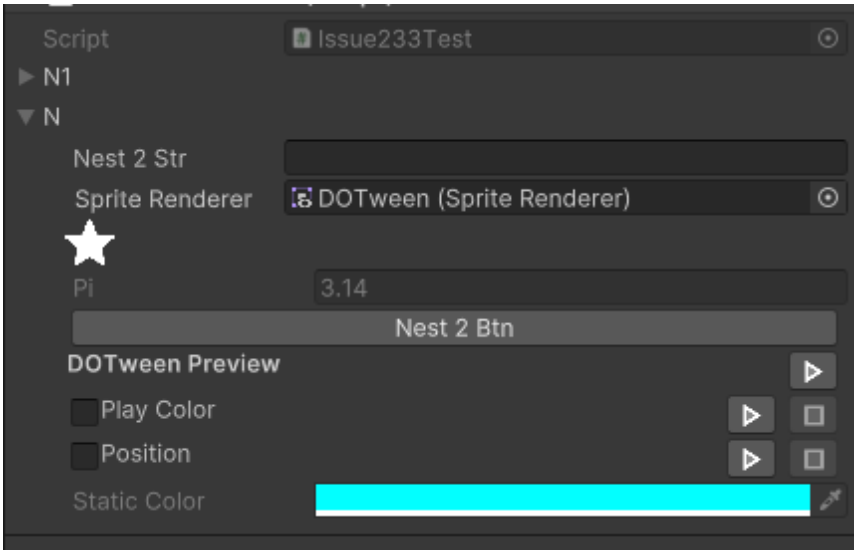
- AllowMultiple: No

Special Note:

1. After applying this attribute, only pure `PropertyDrawer`, and decorators from `SaintsEditor` works on this target. Which means, using third party's `PropertyDrawer` is fine, but decorator of Editor level (e.g. Odin's `Button`, NaughtyAttributes' `Button`) will not work.
2. IMGUI: `ELayout.Horizontal` does not work here
3. IMGUI: `DOTweenPlay` might be a bit buggy displaying the playing/pause/stop status for each function.

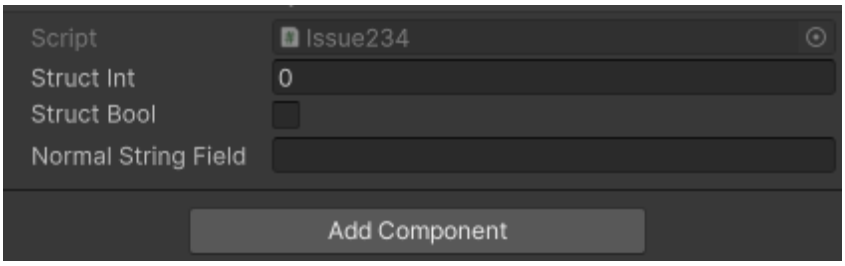
```
[Serializable]
public struct Nest
{
    public string nest2Str;  // normal field
    [Button]  // function button
    private void Nest2Btn() => Debug.Log("Call Nest2Btn");
    // static field (non serializable)
    [ShowInInspector] public static Color StaticColor => Color.cyan;
    // const field (non serializable)
    [ShowInInspector] public const float Pi = 3.14f;
    // normal attribute drawer works as expected
    [BelowImage(maxWidth: 25)] public SpriteRenderer spriteRenderer;

    [DOTweenPlay]  // DOTween helper
    private Sequence PlayColor()
    {
        return DOTween.Sequence()
            .Append(spriteRenderer.DOColor(Color.red, 1f))
            .Append(spriteRenderer.DOColor(Color.green, 1f))
            .Append(spriteRenderer.DOColor(Color.blue, 1f))
            .SetLoops(-1);
```

```
    }
    [DOTweenPlay("Position")]
    private Sequence PlayTween2()
    {
        return DOTween.Sequence()
                .Append(spriteRenderer.transform.DOMove(Vector3.up, 1f))
                .Append(spriteRenderer.transform.DOMove(Vector3.right, 1f))
                .Append(spriteRenderer.transform.DOMove(Vector3.down, 1f))
                .Append(spriteRenderer.transform.DOMove(Vector3.left, 1f))
                .Append(spriteRenderer.transform.DOMove(Vector3.zero, 1f))
            ;
    }
}

[SaintsRow]
public Nest n1;
```



To show a `Serializable` inline like it's directly in the `MonoBehavior`:

```
[Serializable]
public struct MyStruct
{
    public int structInt;
    public bool structBool;
}

[SaintsRow(inline: true)]
public MyStruct myStructInline;

public string normalStringField;
```



4.5.22. `UIToolkit`

Add this only to field that has not `SaintsField` attribute to make this field's label behave like UI Toolkit. This does not work for pure `IMGUI` drawer. This is a fix for Unity's bugged `PropertyField` label.

This is only available if you have `UI Toolkit` enabled (Unity 2022.2+ without disable UI Toolkit in `SaintsField`)

4.6. Other Tools

4.6.1. Addressable

These tools are for Unity Addressable . It's there only if you have `Addressable` installed.

Namespace: `SaintsField.Addressable`

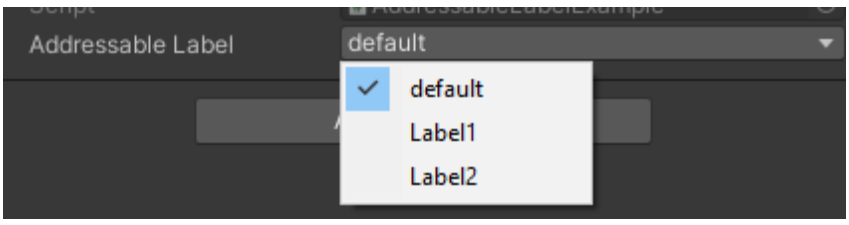
If you encounter issue because of version incompatible with your installation, you can add a macro `SAINTSFIELD_ADDRESSABLE_DISABLE` to disable this component (See "Add a Macro" section for more information)

4.6.1.1 `AddressableLabel`

A picker to select an addressable label.

- Allow Multiple: No

```
public class AddressableLabelExample : MonoBehaviour
{
    [AddressableLabel]
    public string addressableLabel;
}
```



4.6.1.2 `AddressableAddress`

A picker to select an addressable address (key).

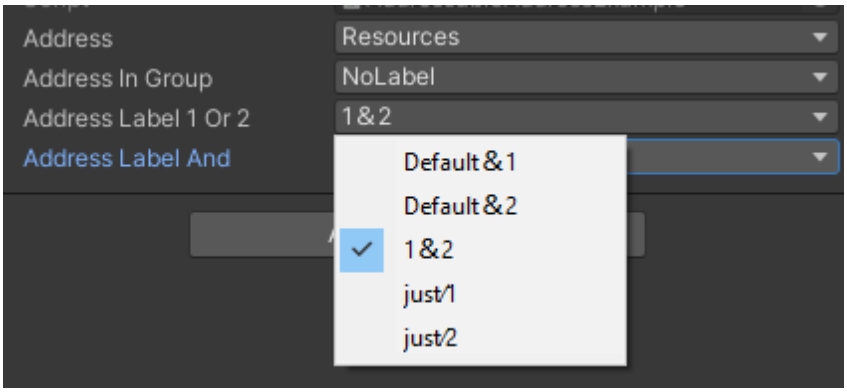
- `string group = null` the Addressable group name. `null` for all groups
- `params string[] orLabels` the addressable label names to filter. Only `entries` with this label will be shown. `null` for no filter.

  If it requires multiple labels, use `A && B`, then only entries with both labels will be shown.

  If it requires any of the labels, just pass them separately, then entries with either label will be shown. For example, pass `"A && B", "C"` will show entries with both `A` and `B` label, or with `C` label.

- Allow Multiple: No

```
public class AddressableAddressExample: MonoBehaviour
{
    [AddressableAddress]  // from all group
    public string address;

    [AddressableAddress("Packed Assets")]  // from this group
    public string addressInGroup;

    [AddressableAddress(null, "Label1", "Label2")]  // either has label `Label1` or `Label2`
    public string addressLabel1Or2;

    // must have both label `default` and `Label1`
    // or have both label `default` and `Label2`
    [AddressableAddress(null, "default && Label1", "default && Label2")]
    public string addressLabelAnd;
}
```



4.6.2. AI Navigation

These tools are for Unity AI Navigation (`NavMesh`). It's there only if you have `AI Navigation` installed.

Namespace: `SaintsField.AiNavigation`

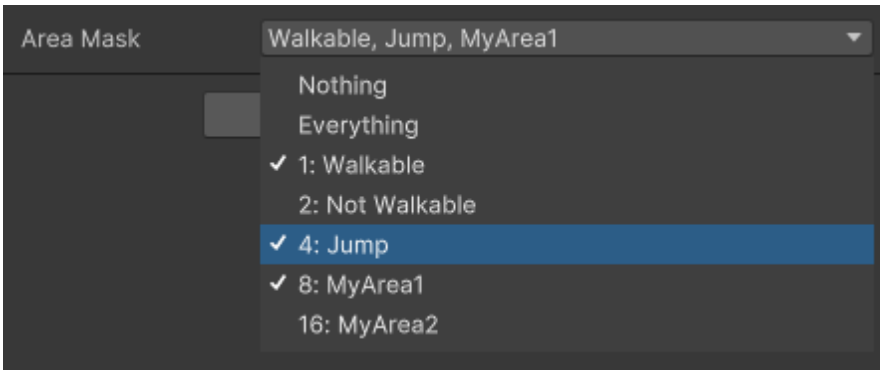
Adding marco `SAINTSFIELD_AI_NAVIGATION_DISABLED` to disable this component. (See "Add a Macro" section for more information)

4.6.2.1 `NavMeshAreaMask`

Select `NavMesh` area bit mask for an integer field. (So the integer value can be used in `SamplePathPosition`)

- Allow Multiple: No

```
[NavMeshAreaMask]
public int areaMask;
```

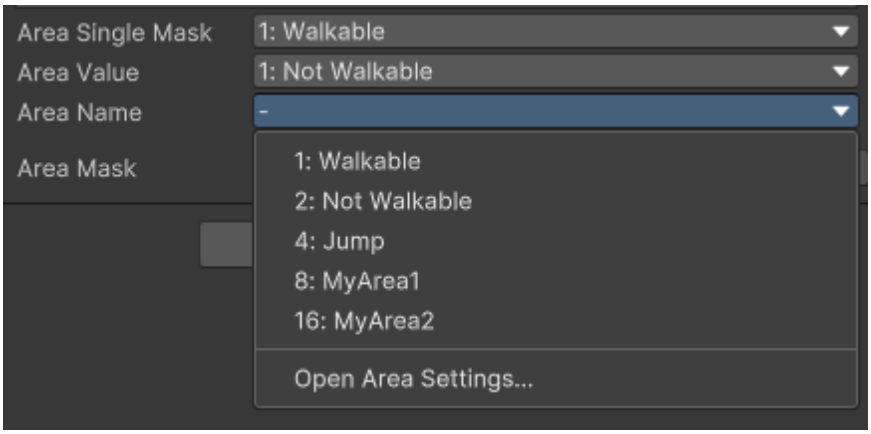


4.6.2.2 `NavMeshArea`

Select a `NavMesh` area for a string or an interger field.

- `bool isMask=true` if true, it'll use the bit mask, otherwise, it'll use the area value. Has no effect if the field is a string.
- `string groupBy = ""` for error message grouping
- Allow Multiple: No

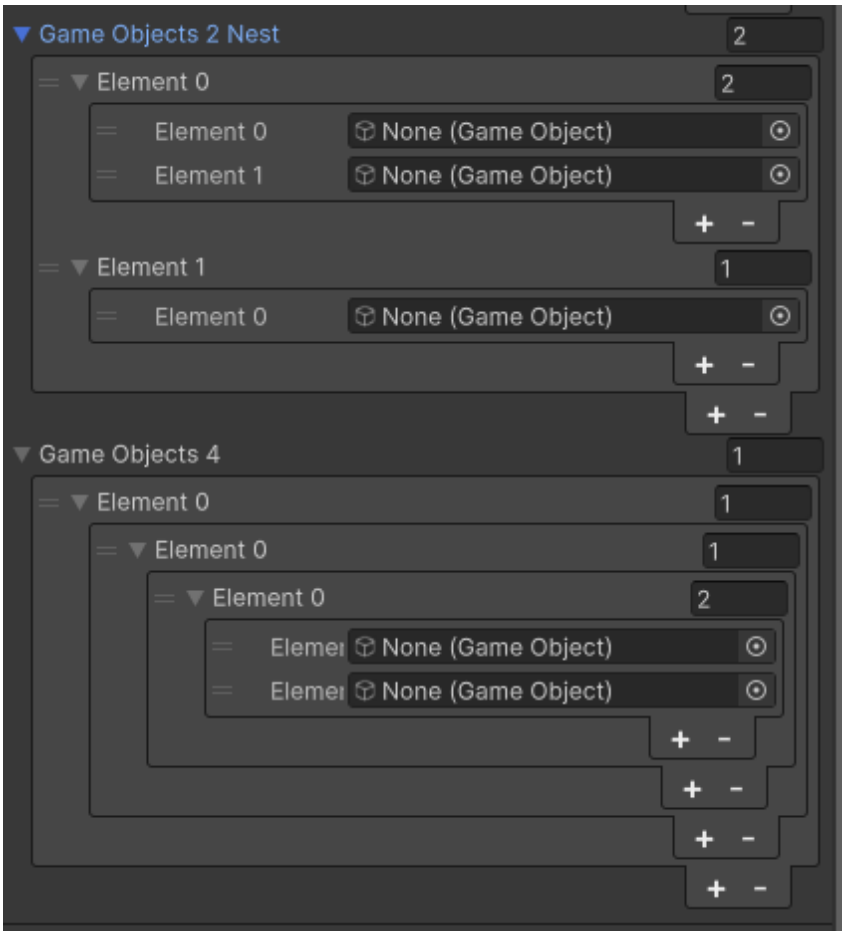
```
[NavMeshArea]  // then you can use `areaSingleMask1 | areaSingleMask2` to get multiple masks
public int areaSingleMask;

[NavMeshArea(false)]  // then you can use `1 << areaValue` to get areaSingleMask
public int areaValue;

[NavMeshArea]  // then you can use `NavMesh.GetAreaFromName(areaName)` to get areaValue
public int areaName;
```



4.6.3. `SaintsArray` / `SaintsList`

Unity does not allow to serialize two dimensional array or list. `SaintsArray` and `SaintsList` are there to help.

```
// two dimensional array
public SaintsArray<GameObject>[] gameObjects2;
public SaintsArray<SaintsArray<GameObject>> gameObjects2Nest;
// four dimensional array, if you like.
// it can be used with array, but ensure the `[]` is always at the end.
public SaintsArray<SaintsArray<SaintsArray<GameObject>>>[] gameObjects4;
```



`SaintsArray` implements `IReadOnlyList`, `SaintsList` implements `IList`:

```
// SaintsArray
GameObject firstGameObject = saintsArrayGo[0];
Debug.Log(saintsArrayGo.value); // the actual array value

// SaintsList
saintsListGo.Add(new GameObject());
saintsListGo.RemoveAt(0);
Debug.Log(saintsListGo.value);  // the actual list value
```

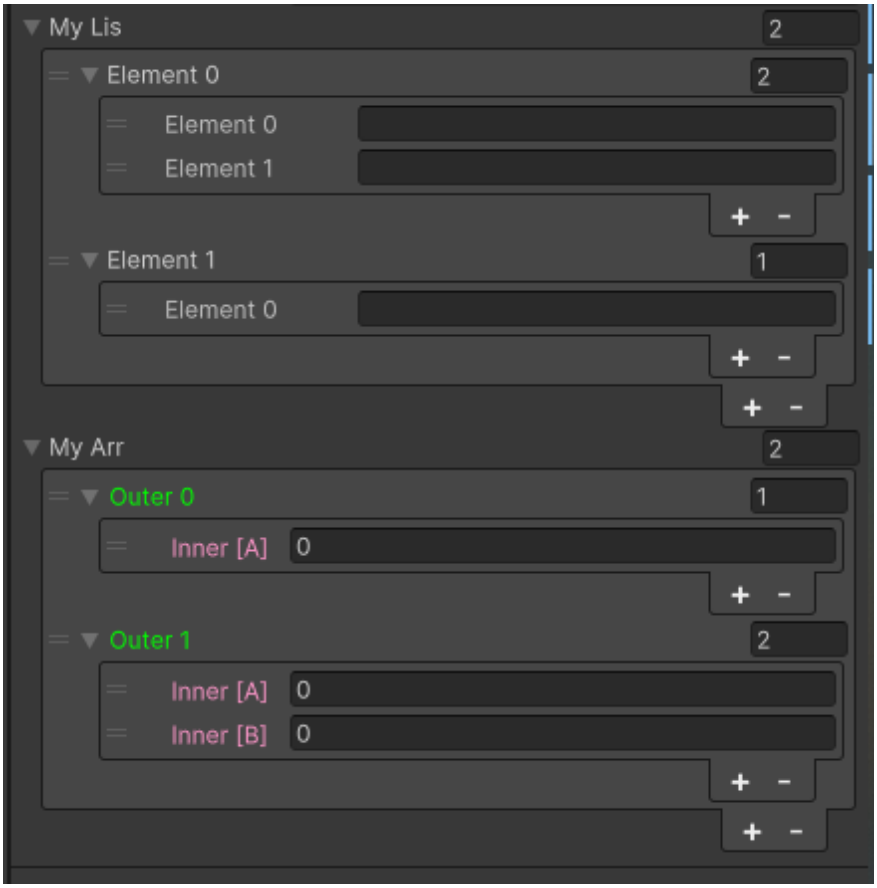
These two can be easily converted to array/list:

```
// SaintsArray to Array
GameObject[] arrayGo = saintsArrayGo;
// Array to SaintsArray
SaintsArray<GameObject> expSaintsArrayGo = (SaintsArray<GameObject>)arrayGo;

// SaintsList to List
List<GameObject> ListGo = saintsListGo;
// List to SaintsList
SaintsList<GameObject> expSaintsListGo = (SaintsList<GameObject>)ListGo;
```

Because it's actually a struct, you can also implement your own Array/List, using `[SaintsArray]`. Here is an example of customize your own struct:

```
// example: using ISaintsArray so you don't need to specify the type name everytime
[Serializable]
public class MyList : ISaintsArray
{
    [SerializeField] public List<string> myStrings;

#if UNITY_EDITOR
    public string EditorArrayPropertyName => nameof(myStrings);
#endif
}

[SaintsArray]
public MyList[] myLis;


// example: any Serializable which hold a serialized array/list is fine
[Serializable]
public struct MyArr
{
    [RichLabel(nameof(MyInnerRichLabel), true)]
    public int[] myArray;

    private string MyInnerRichLabel(object _, int index) => $"<color=pink> Inner [{(char)('A' +
}
```

```
[RichLabel(nameof(MyOuterLabel), true), SaintsArray("myArray")]
public MyArr[] myArr;

private string MyOuterLabel(object _, int index) => $"<color=Lime> Outer {index}";
```



5. SaintsEditor

`SaintsField` is a `UnityEditor.Editor` level component.

Compared with `NaughtyAttributes` and `MarkupAttributes`:

1. `NaughtyAttributes` has `Button`, and has a way to show a non-field property(`ShowNonSerializedField`, `ShowNativeProperty`), but it does not retain the order of these fields, but only draw them at the end. It has layout functions (`Foldout`, `BoxGroup`) but it has not `Tab` layout, and much less powerful compared to `MarkupAttributes`. It's IMGUI only.
2. `MarkupAttributes` is super powerful in layout, but it does not have a way to show a non-field property. It's IMGUI only.
3. `SaintsEditor`
    - `Layout` like markup attributes. Compared to `MarkupAttributes`, it allows a non-field property (e.g. a button or a `ShowInInspector` inside a group) (like `OdinInspector`). However, it does not have a `Scope` for convenience coding.

- It provides `Button` (with less functions) and a way to show a non-field property (`ShowInInspector`).
- It tries to retain the order, and allows you to use `[Ordered]` when it can not get the order (c# does not allow to obtain all the orders).
- Supports both `UI Toolkit` and `IMGUI` .
- When using `UI Toolkit` , it'll try to fix the old style field, change the label behavior like UI Toolkit. (This fix does not work if the fallback drawer is a pure `IMGUI` drawer)

Please note, any `Editor` level component can not work together with each other (it will not cause trouble, but only one will actually work). Which means, `OdinInspector` , `NaughtyAttributes` , `MarkupAttributes` , `SaintsEditor` can not work together.

If you are interested, here is how to use it.

5.1. Set Up SaintsEditor

`Window` - `Saints` - `Apply SaintsEditor` . After the project finish re-compile, go `Window` - `Saints` - `SaintsEditor` to tweak configs.

If you want to do it manually, check ApplySaintsEditor.cs for more information

5.2. `DOTweenPlay`

A method decorator to play a `DOTween` animation returned by the method.

The method should not have required parameters, and need to return a `Tween` or a `Sequence` (`Sequence` is actually also a tween).

Parameters:

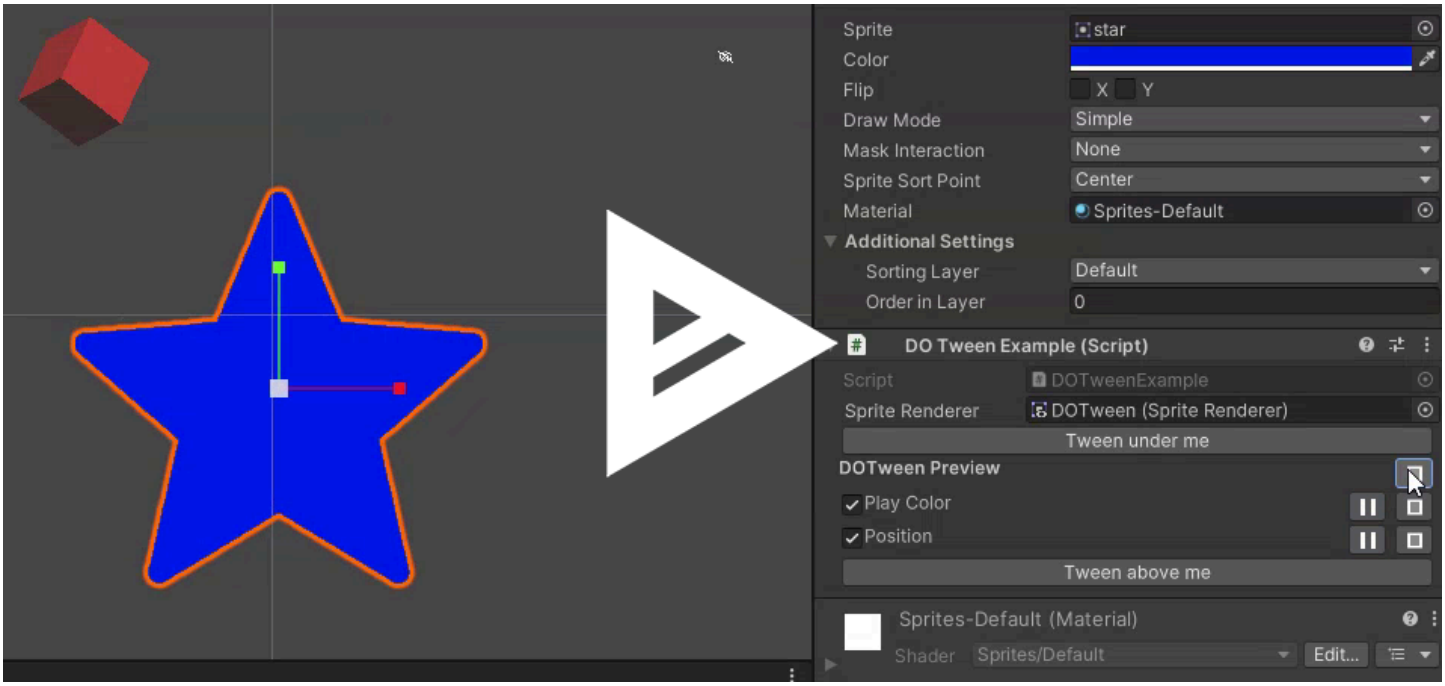
- `[Optional] string label = null` the label of the button. Use method name if null. Rich label not supported.
- `ETweenStop stopAction = ETweenStop.Rewind` the action after the tween is finished or killed. Options are:
  - `None` : do nothing
  - `Complete` : complete the tween. This only works if the tween get killed
  - `Rewind` : rewind to the start state

```
public class DOTweenExample : MonoBehaviour
{
    [GetComponent]
    public SpriteRenderer spriteRenderer;

    [DOTweenPlay]
    private Sequence PlayColor()
    {
```

```
        return DOTween.Sequence()
            .Append(spriteRenderer.DOColor(Color.red, 1f))
            .Append(spriteRenderer.DOColor(Color.green, 1f))
            .Append(spriteRenderer.DOColor(Color.blue, 1f))
            .SetLoops(-1);  // Yes you can make it a loop
    }

    [DOTweenPlay("Position")]
    private Sequence PlayTween2()
    {
        return DOTween.Sequence()
            .Append(spriteRenderer.transform.DOMove(Vector3.up, 1f))
            .Append(spriteRenderer.transform.DOMove(Vector3.right, 1f))
            .Append(spriteRenderer.transform.DOMove(Vector3.down, 1f))
            .Append(spriteRenderer.transform.DOMove(Vector3.left, 1f))
            .Append(spriteRenderer.transform.DOMove(Vector3.zero, 1f))
        ;
    }
}
```

The first row is global control. Stop it there will stop all preview.

The check of each row means auto play when you click the start in the global control.



Set Up

`DOTween` is not a standard Unity package , so `SaintsField` can **NOT** detect if it's installed.

To use `DOTweenPlay` :

1. `Tools` - `Demigaint` - `DOTween Utility Panel` , click `Create ASMDEF`
2. `Window` - `Saints` - `Enable DOTween Support` (See "Add a Macro" section for more information)

5.3. `Button`

Draw a button for a function.

Compared to `NaughtyAttributes`, this does not allow to specific for editing and playing mode. It also does not handle an `IEnumerator` function, it just `Invoke` the target function.

- `string buttonLabel = null` the button label. If null, it'll use the function name.

```
[Button]
private void EditorButton()
{
    Debug.Log("EditorButton");
}

[Button("Label")]
private void EditorLabeledButton()
{
    Debug.Log("EditorLabeledButton");
}
```

5.4. `ShowInInspector`

Show a non-field property.

```
// const
[ShowInInspector, Ordered] public const float MyConstFloat = 3.14f;
// static
[ShowInInspector, Ordered] public static readonly Color MyColor = Color.green;

// auto-property
[ShowInInspector, Ordered]
public Color AutoColor
{
    get => Color.green;
    set {}
}
```

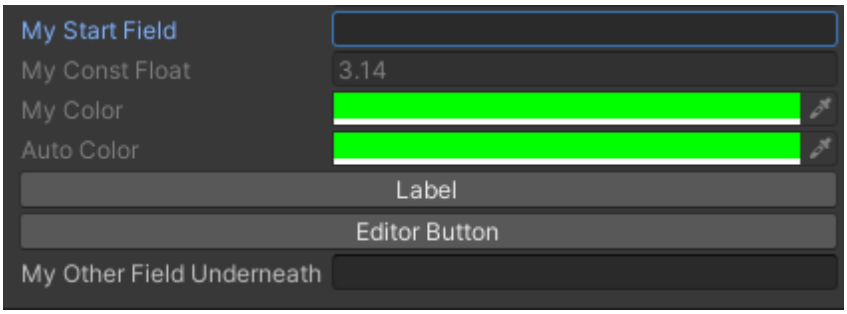
5.5. `Ordered`

`SaintsEditor` uses reflection to get each field. However, c# reflection does not give all the orders: `PropertyInfo`, `MethodInfo` and `FieldInfo` does not order with each other.

Thus, if the order is incorrect, you can use `[Ordered]` to specify the order. But also note: `Ordered` ones are always after the ones without an `Ordered`. So if you want to add it, add it to every field.

- `[CallerLineNumber] int order = 0` the order of this field. By default it uses line number of the file. You may not want to override this.

```
[Ordered] public string myStartField;

[ShowInInspector, Ordered] public const float MyConstFloat = 3.14f;
[ShowInInspector, Ordered] public static readonly Color MyColor = Color.green;

[ShowInInspector, Ordered]
public Color AutoColor
{
    get => Color.green;
    set {}
}

[Button, Ordered]
private void EditorButton()
{
    Debug.Log("EditorButton");
}

[Ordered] public string myOtherFieldUnderneath;
```



5.6. Layout

A layout decorator to group fields.

- `string groupBy` the grouping key. Use `/` to separate different groups and create sub groups.
- `ELayout layout=ELayout.Vertical` the layout of the current group. Note this is a `EnumFlag`, means you can mix with options.

Options are:

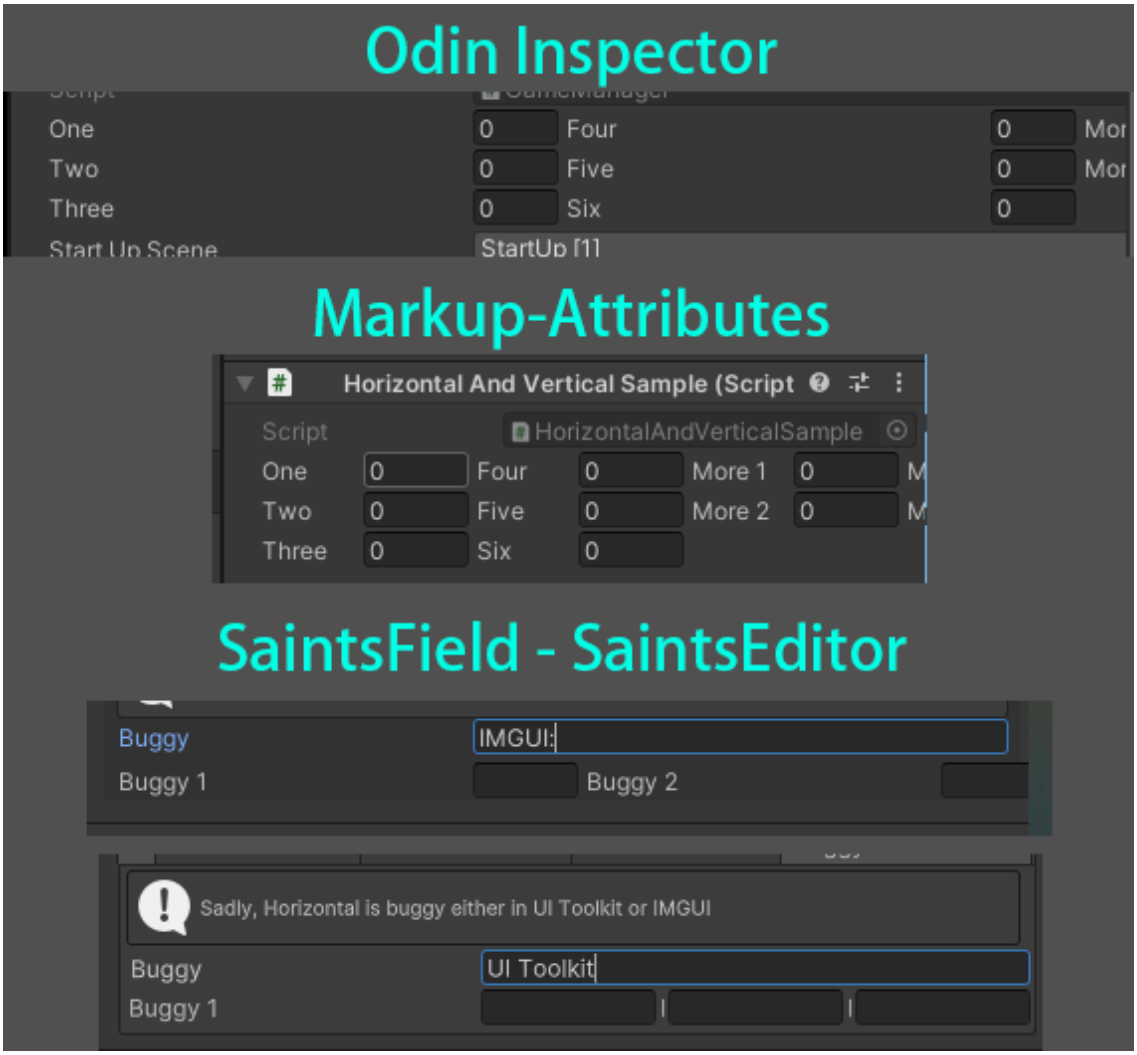
- `Vertical`

- `Horizontal`
- `Background` draw a background color for the whole group
- `TitleOut` make `title` more visible if you have `Title` enabled. On `IMGUI` it will draw an separator between title and the rest of the content. On `UI Toolkit` it will draw a background color for the title.
- `Foldout` allow to fold/unfold this group. If you have no `Tab` on, then this will automatically add `Title`
- `Tab` make this group a tab page separated rather than grouping it
- `Title` show the title

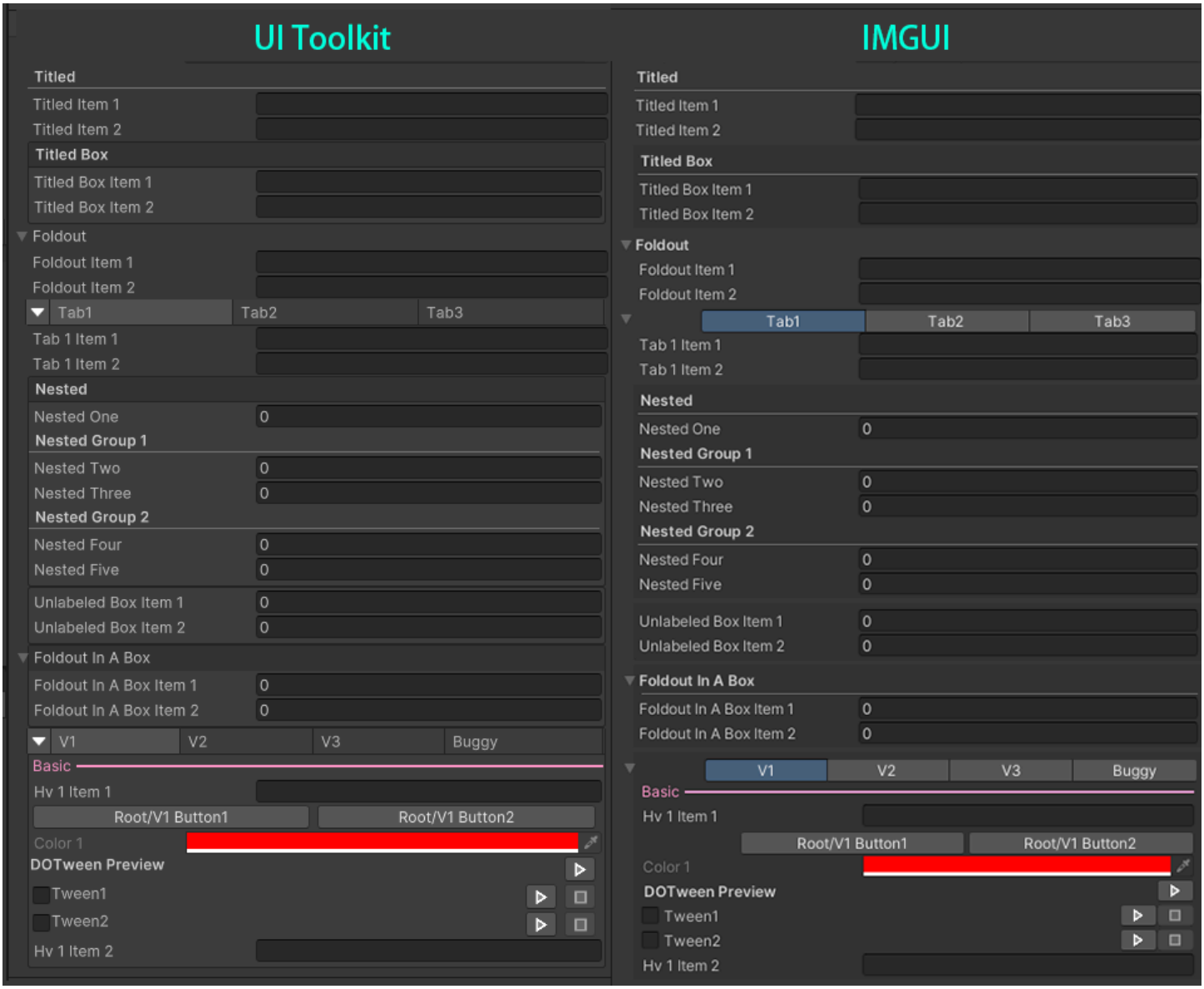
Known Issue

`Horizental` style is buggy, for the following reasons:

1. On IMGUI, `HorizontalScope` does **NOT** shrink when there are many items, and will go off-view without a scrollbar. Both `Odin` and `Markup-Attributes` have the same issue. However, `Markup-Attribute` uses `labelWidth` to make the situation a bit better, which `SaintsEditor` does not provide (at this point at least).
2. On UI Toolkit we have the well-behaved layout system, but because of its buggy "Label Width" (see "Common Pitfalls & Compatibility" - "UI Toolkit" section for more information), all the field except the first one will get the super-shrank label width which makes it unreadable.



Appearance

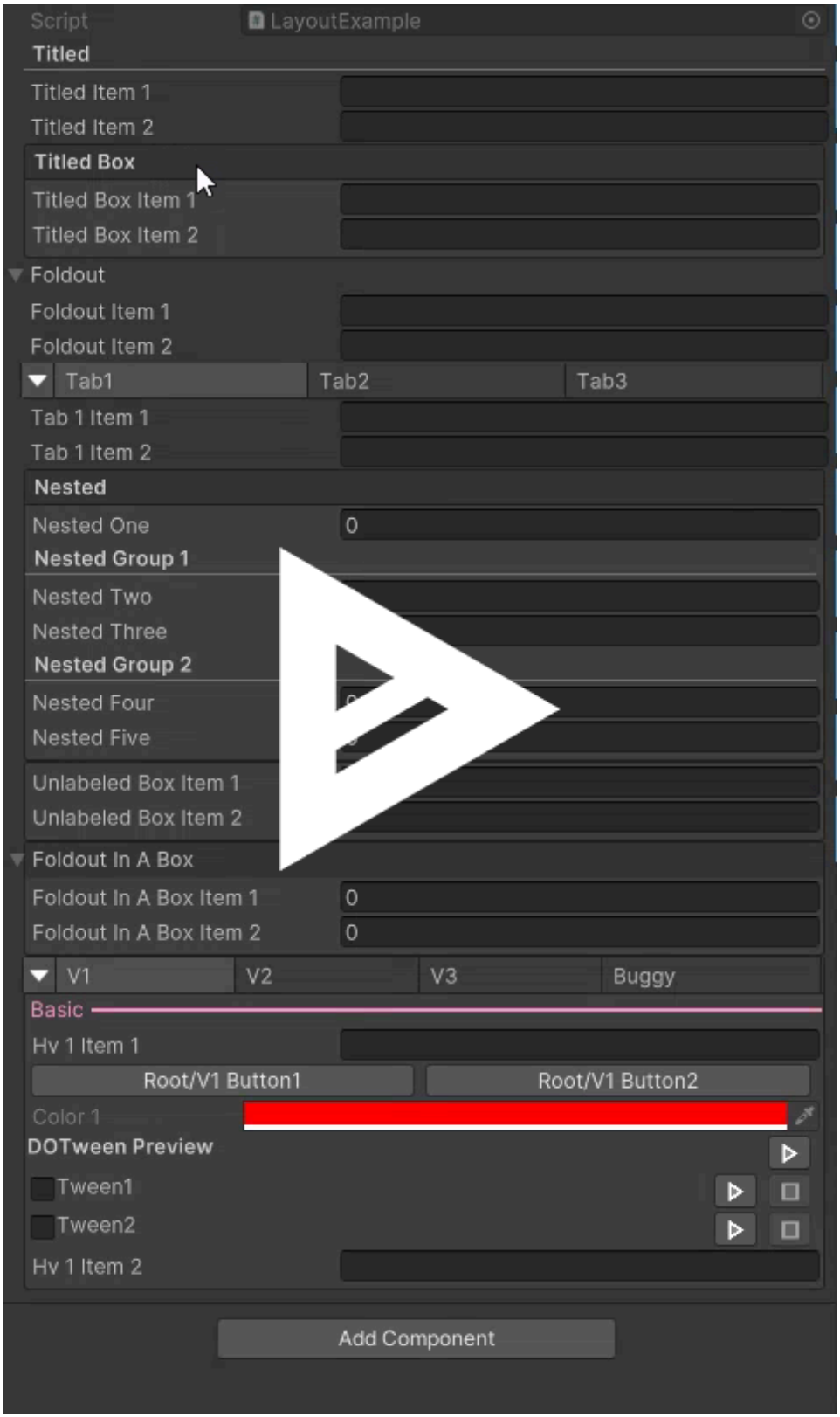


Example

```
public class LayoutExample: MonoBehaviour
{
    [Layout("Titled", ELayout.Title | ELayout.TitleOut)]
    public string titledItem1, titledItem2;

    // title
    [Layout("Titled Box", ELayout.Title | ELayout.Background | ELayout.TitleOut)]
    public string titledBoxItem1;
    [Layout("Titled Box")]  // you can omit config when you already declared one somewhere (no
    public string titledBoxItem2;

    // foldout
    [Layout("Foldout", ELayout.Foldout)]
    public string foldoutItem1, foldoutItem2;

    // tabs
    [Layout("Tabs", ELayout.Tab | ELayout.Foldout)]
```

```
    [Layout("Tabs/Tab1")]
    public string tab1Item1, tab1Item2;

    [Layout("Tabs/Tab2")]
    public string tab2Item1, tab2Item2;

    [Layout("Tabs/Tab3")]
    public string tab3Item1, tab3Item2;

    // nested groups
    [Layout("Nested", ELayout.Title | ELayout.Background | ELayout.TitleOut)]
    public int nestedOne;
    [Layout("Nested/Nested Group 1", ELayout.Title | ELayout.TitleOut)]
    public int nestedTwo, nestedThree;
    [Layout("Nested/Nested Group 2", ELayout.Title | ELayout.TitleOut)]
    public int nestedFour, nestedFive;

    // Unlabeled Box
    [Layout("Unlabeled Box", ELayout.Background)]
    public int unlabeledBoxItem1, unlabeledBoxItem2;

    // Foldout In A Box
    [Layout("Foldout In A Box", ELayout.Foldout | ELayout.Background | ELayout.TitleOut)]
    public int foldoutInABoxItem1, foldoutInABoxItem2;

    // Complex example. Button and ShowInInspector works too
    [Ordered]
    [Layout("Root", ELayout.Tab | ELayout.TitleOut | ELayout.Foldout | ELayout.Background)]
    [Layout("Root/V1")]
    [SepTitle("Basic", EColor.Pink)]
    public string hv1Item1;

    [Ordered]
    [Layout("Root/V1/buttons", ELayout.Horizontal)]
    [Button("Root/V1 Button1")]
    public void RootV1Button()
    {
        Debug.Log("Root/V1 Button");
    }
    [Ordered]
    [Layout("Root/V1/buttons", ELayout.Horizontal)]
    [Button("Root/V1 Button2")]
    public void RootV1Button2()
    {
        Debug.Log("Root/V1 Button");
    }

    [Ordered]
    [Layout("Root/V1")]
    [ShowInInspector]
    public static Color color1 = Color.red;

    [Ordered]
```

```
    [DOTweenPlay("Tween1", "Root/V1")]
    public Tween RootV1Tween1()
    {
        return DOTween.Sequence();
    }

    [Ordered]
    [DOTweenPlay("Tween2", "Root/V1")]
    public Tween RootV1Tween2()
    {
        return DOTween.Sequence();
    }

    [Ordered]
    [Layout("Root/V1")]
    public string hv1Item2;

    // public string below;

    [Ordered]
    [Layout("Root/V2")]
    public string hv2Item1;

    [Ordered]
    [Layout("Root/V2/H", ELayout.Horizontal), RichLabel(null)]
    public string hv2Item2, hv2Item3;

    [Ordered]
    [Layout("Root/V2")]
    public string hv2Item4;

    [Ordered]
    [Layout("Root/V3", ELayout.Horizontal)]
    [ResizableTextArea, RichLabel(null)]
    public string hv3Item1, hv3Item2;

    [Ordered]
    [Layout("Root/Buggy")]
    [InfoBox("Sadly, Horizontal is buggy either in UI Toolkit or IMGUI", above: true)]
    public string buggy = "See below:";

    [Ordered]
    [Layout("Root/Buggy/H", ELayout.Horizontal)]
    public string buggy1;
    [Ordered]
    [Layout("Root/Buggy/H", ELayout.Horizontal)]
    public string buggy2;
}
```

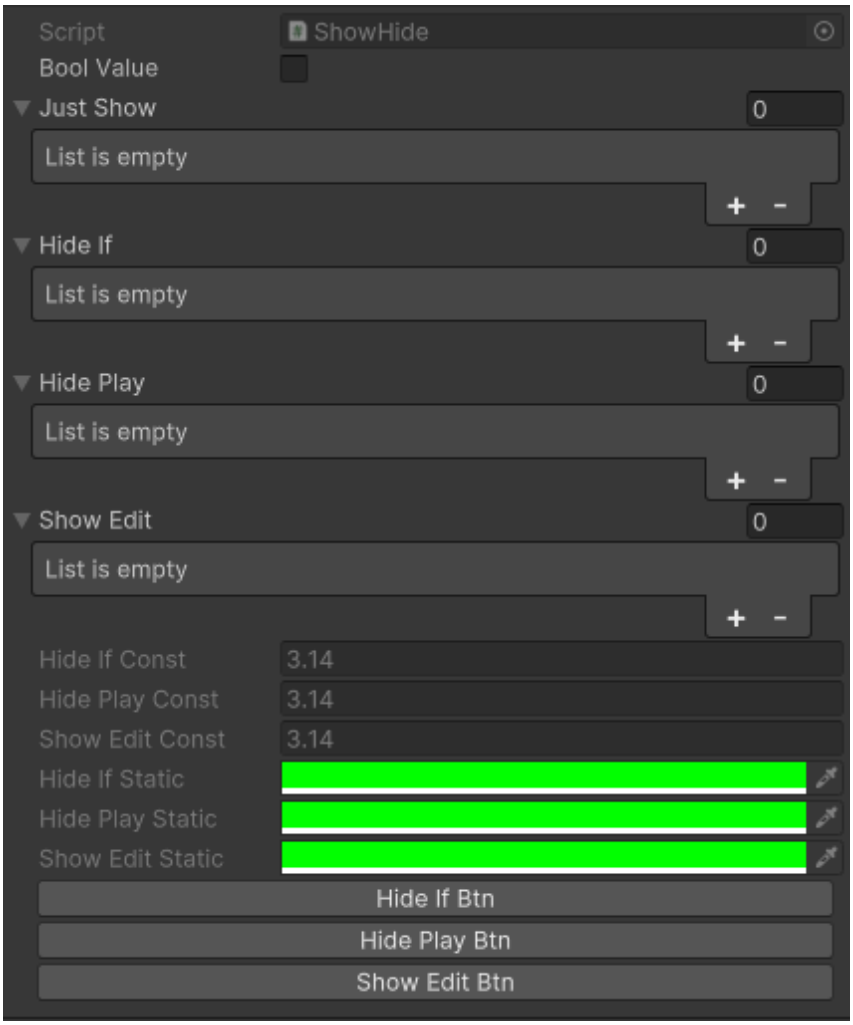


5.7. `PlayaShowIf` / `PlayaHideIf`

This is the same as `ShowIf`, `HideIf`, plus it's allowed to be applied to array, `Button`, `ShowInInspector`

Different from `ShowIf` / `HideIf`: apply on an array will directly show or hide the array itself, rather than each element.

```
public bool boolValue;

[PlayaHideIf] public int[] justHide;
[PlayaShowIf] public int[] justShow;

[PlayaHideIf(nameof(boolValue))] public int[] hideIf;
[PlayaShowIf(nameof(boolValue))] public int[] showIf;

[PlayaHideIf(EMode.Edit)] public int[] hideEdit;
[PlayaHideIf(EMode.Play)] public int[] hidePlay;
[PlayaShowIf(EMode.Edit)] public int[] showEdit;
[PlayaShowIf(EMode.Play)] public int[] showPlay;

[ShowInInspector, PlayaHideIf(nameof(boolValue))] public const float HideIfConst = 3.14f;
[ShowInInspector, PlayaShowIf(nameof(boolValue))] public const float ShowIfConst = 3.14f;
[ShowInInspector, PlayaHideIf(EMode.Edit)] public const float HideEditConst = 3.14f;
[ShowInInspector, PlayaHideIf(EMode.Play)] public const float HidePlayConst = 3.14f;
[ShowInInspector, PlayaShowIf(EMode.Edit)] public const float ShowEditConst = 3.14f;
[ShowInInspector, PlayaShowIf(EMode.Play)] public const float ShowPlayConst = 3.14f;

[ShowInInspector, PlayaHideIf(nameof(boolValue))] public static readonly Color HideIfStatic = C
[ShowInInspector, PlayaShowIf(nameof(boolValue))] public static readonly Color ShowIfStatic = C
[ShowInInspector, PlayaHideIf(EMode.Edit)] public static readonly Color HideEditStatic = Color.
[ShowInInspector, PlayaHideIf(EMode.Play)] public static readonly Color HidePlayStatic = Color.
[ShowInInspector, PlayaShowIf(EMode.Edit)] public static readonly Color ShowEditStatic = Color.
[ShowInInspector, PlayaShowIf(EMode.Play)] public static readonly Color ShowPlayStatic = Color.

[Button, PlayaHideIf(nameof(boolValue))] private void HideIfBtn() => Debug.Log("HideIfBtn");
[Button, PlayaShowIf(nameof(boolValue))] private void ShowIfBtn() => Debug.Log("ShowIfBtn");
[Button, PlayaHideIf(EMode.Edit)] private void HideEditBtn() => Debug.Log("HideEditBtn");
[Button, PlayaHideIf(EMode.Play)] private void HidePlayBtn() => Debug.Log("HidePlayBtn");
[Button, PlayaShowIf(EMode.Edit)] private void ShowEditBtn() => Debug.Log("ShowEditBtn");
[Button, PlayaShowIf(EMode.Play)] private void ShowPlayBtn() => Debug.Log("ShowPlayBtn");
```



5.8. `PlayaEnableIf` / `PlayaDisableIf`

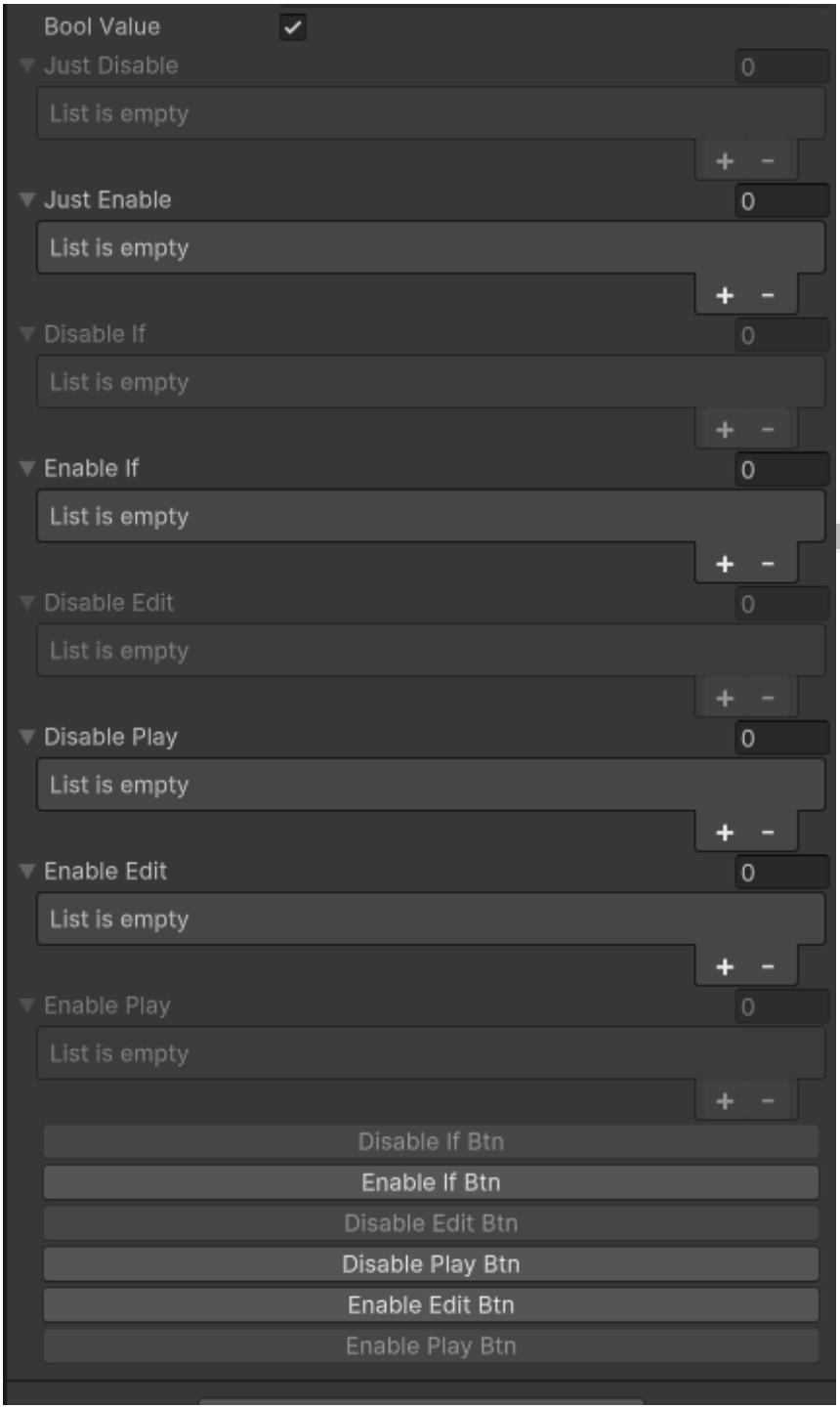
This is the same as `EnableIf`, `DisableIf`, plus it can be applied to array, `Button`

Different from `EnableIf` / `DisableIf` in the following:

1. apply on an array will directly enable or disable the array itself, rather than each element.
2. this method can not detect foldout, which means using it on `Expandable`, `EnumFlags`, the foldout button will also be disabled. For this case, use `DisableIf` / `EnableIf` instead.

```
[PlayaDisableIf] public int[] justDisable;
[PlayaEnableIf] public int[] justEnable;

[PlayaDisableIf(nameof(boolValue))] public int[] disableIf;
[PlayaEnableIf(nameof(boolValue))] public int[] enableIf;

[PlayaDisableIf(EMode.Edit)] public int[] disableEdit;
[PlayaDisableIf(EMode.Play)] public int[] disablePlay;
[PlayaEnableIf(EMode.Edit)] public int[] enableEdit;
[PlayaEnableIf(EMode.Play)] public int[] enablePlay;

[Button, PlayaDisableIf(nameof(boolValue))] private void DisableIfBtn() => Debug.Log("DisableIf
[Button, PlayaEnableIf(nameof(boolValue))] private void EnableIfBtn() => Debug.Log("EnableIfBtn
[Button, PlayaDisableIf(EMode.Edit)] private void DisableEditBtn() => Debug.Log("DisableEditBtn
```

```
[Button, PlayaDisableIf(EMode.Play)] private void DisablePlayBtn() => Debug.Log("DisablePlayBtn");
[Button, PlayaEnableIf(EMode.Edit)] private void EnableEditBtn() => Debug.Log("EnableEditBtn");
[Button, PlayaEnableIf(EMode.Play)] private void EnablePlayBtn() => Debug.Log("EnablePlayBtn");
```



5.9. `PlayaArraySize`

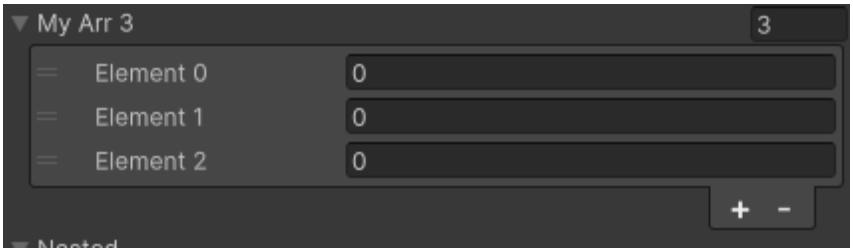
Like `ArraySize`, but:

1. it will set array size to expected size when the array is empty
2. it does not have a `groupBy`, and will not give any error if the target is not an array/list

Parameters:

- `int size` the size of the array or list
- AllowMultiple: No

```
[PlayaArraySize(3)] public int[] myArr3;
```



6. About GroupBy

group with any decorator that has the same `groupBy` for this field. The same group will share even the width of the view width between them.

This only works for decorator draws above or below the field. The above drawer will not grouped with the below drawer, and vice versa.

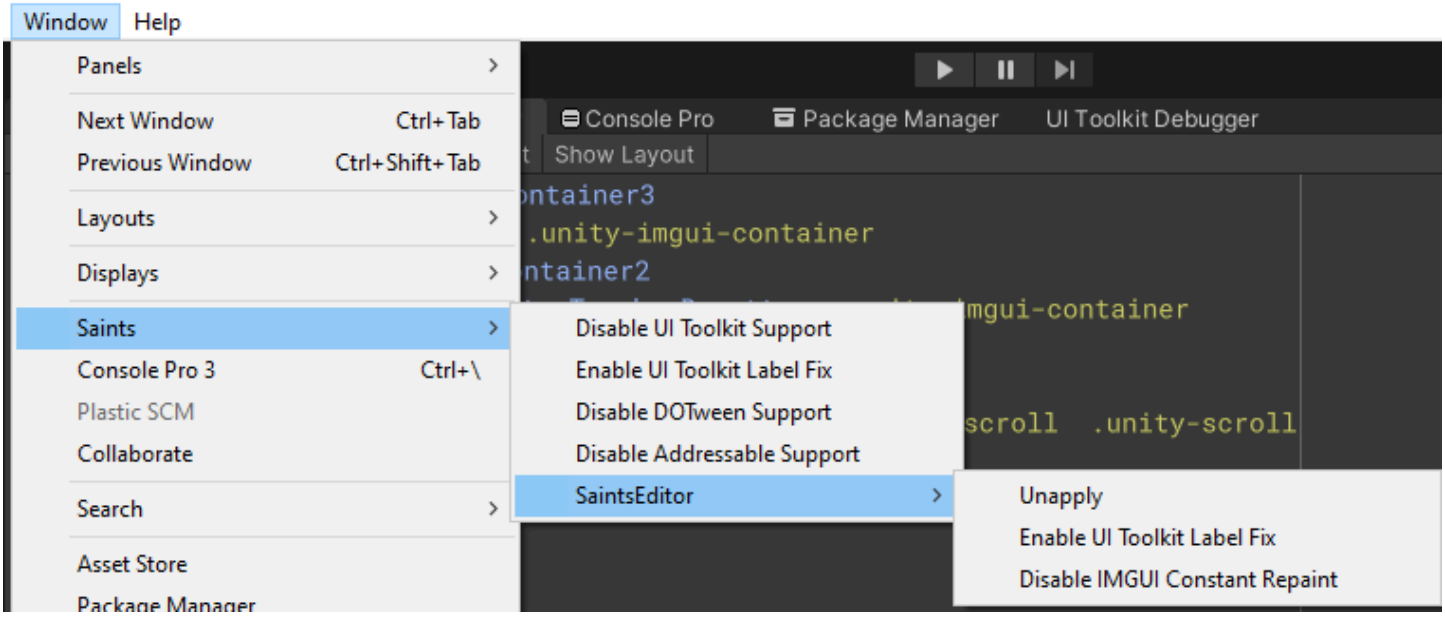
`""` means no group.

7. Add a Macro

Pick a way that is most convenient for you:

Using Saints Menu

Go to `Window` - `Saints` to enable/disable functions you want



Using csc.rsp

1. Create file `Assets/csc.rsp`
2. Write marcos like this:

```
#"Enable DOTween"
-define:SAINTSFIELD_DOTWEEN

#"Disable Addressable"
-define:SAINTSFIELD_ADDRESSABLE_DISABLE

#"Disable AI Navigation"
-define:SAINTSFIELD_AI_NAVIGATION_DISABLED

#"Disable UI Toolkit"
-define:SAINTSFIELD_UI_TOOLKIT_DISABLE

#"Disable UI Toolkit label fix for PropertyDrawer"
-define:SAINTSFIELD_UI_TOOLKIT_LABEL_FIX_DISABLE

#"Apply SaintsEditor project wide"
-define:SAINTSFIELD_SAINTS_EDITOR_APPLY

#"Disable SaintsEditor IMGUI constant repaint"
-define:SAINTSFIELD_SAINTS_EDITOR_IMGUI_CONSTANT_REPAINT_DISABLE

#"Disable SaintsEditor UI Toolkit label fix"
-define:SAINTSFIELD_SAINTS_EDITOR_UI_TOOLKIT_LABEL_FIX_DISABLE
```

Note: `csc.rsp` can override settings by Saints Menu.

Using project settings

`Edit` - `Project Settings` - `Player`, find your platform, then go `Other Settings` - `Script Compliation` - `Scripting Define Symbols` to add your marcos. Don't forget to click `Apply` before closing the window.

8. Common Pitfalls & Compatibility

8.1. List/Array & Nesting

Directly using on list/array will apply to every direct element of the list, this is a limit from Unity.

Unlike NaughtyAttributes, `SaintsField` does not need a `AllowNesting` attribute to work on nested element.

```
public class ArrayLabelExample : MonoBehaviour
{
    // works for list/array
```

```
    [Scene] public int[] myScenes;

    [System.Serializable]
    public struct MyStruct
    {
        public bool enableFlag;

        [AboveRichLabel("No need for `[AllowNesting]`, it just works")]
        public int integer;
    }

    public MyStruct myStruct;
}
```

8.2. Order Matters

`SaintsField` only uses `PropertyDrawer` to draw the field, and will properly fall back to the rest drawers if there is one. This works for both 3rd party drawer, your custom drawer, and Unity's default drawer.

However, Unity only allows decorators to be loaded from top to bottom, left to right. Any drawer that does not properly handle the fallback will override `PropertyDrawer` follows by. Thus, ensure `SaintsField` is always the first decorator.

An example of working with NaughtyAttributes:

```
public class CompatibilityNaAndDefault : MonoBehaviour
{
    [RichLabel("<color=green>+NA</color>"),
     NaughtyAttributes.CurveRange(0, 0, 1, 1, NaughtyAttributes.EColor.Green),
     NaughtyAttributes.Label(" ")  // a little hack for label issue
    ]
    public AnimationCurve naCurve;

    [RichLabel("<color=green>+Native</color>"), Range(0, 5)]
    public float nativeRange;

    // this wont work. Please put `SaintsField` before other drawers
    [Range(0, 5), RichLabel("<color=green>+Native</color>")]
    public float nativeRangeHandled;

    // this wont work too. Please put `SaintsField` before other drawers
    [NaughtyAttributes.CurveRange(0, 0, 1, 1, NaughtyAttributes.EColor.Green)]
    [RichLabel("<color=green>+NA</color>")]
    public AnimationCurve naCurveHandled;
}
```

8.3. Fallback To Other Drawers

`SaintsField` is designed to be compatible with other drawers if

1. the drawer itself respects the `GUIContent` argument in `OnGUI` for IMGUI, or add `unity-label` class to Label for UI Toolkit

   NOTE: `NaughtyAttributes` (IMGUI) uses `property.displayName` instead of `GUIContent`. You need to set `Label(" ")` if you want to use `RichLabel`. Also, `NaughtyAttributes` tread some attributes as first-class citizen, so the compatibility is not guaranteed.

2. if the drawer hijacks the `CustomEditor`, it must fall to the rest drawers

   NOTE: In many cases `Odin` does not fallback to the rest drawers, but only to `Odin` and Unity's default drawers. So sometimes things will not work with `Odin`

Special Note:

NaughtyAttributes uses only IMGUI. If you're using Unity 2022.2+, `NaughtyAttributes`'s editor will try fallback default drawers and Unity will decide to use UI Toolkit rendering `SaintsField` and cause troubles. Please disable `SaintsField`'s UI Toolkit ability by `Window` - `Saints` - `Disable UI Toolkit Support` (See "Add a Macro" section for more information)

My (not full) test about compatibility:

- Markup-Attributes : Works very well.
- NaughtyAttributes : Works well, need that `Label` hack.
- OdinInspector : Works mostly well for MonoBehavior/ScriptableObject. Not so good for Odin's `EditorWindow`.

8.4. UI Toolkit

If you encounter any issue, please report it to the issue page. However, there are many issues that is just not fixable:

1. Label width. UI Toolkit uses a fixed label width 120px. However, this value is different when the field is nested and indented.

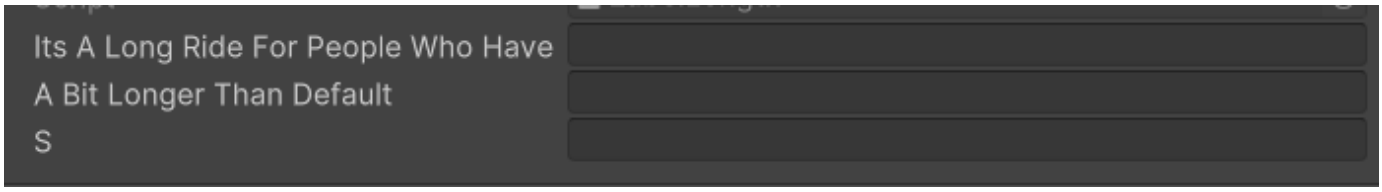
   In IMGUI, we have `EditorGUIUtility.labelWidth`, `EditorGUI.indentLevel`, which is absolutely not available in UI Toolkit, and there is no way to get the label width.

   Since SaintsField 2.3.0, this issue is much better handled!

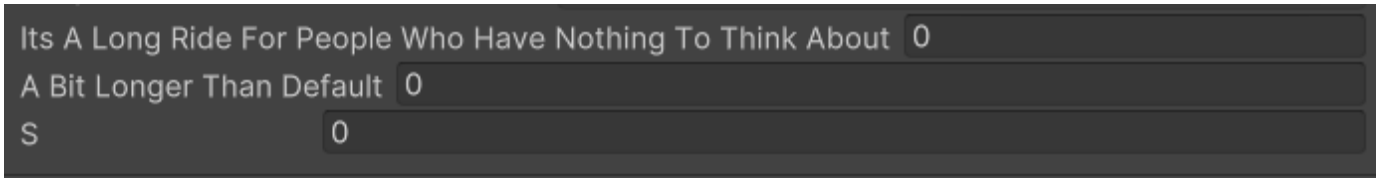
   Considering the following labels with UI Toolkit enabled:

   ```
   public string itsALongRideForPeopleWhoHaveNothingToThinkAbout;
   public string aBitLongerThanDefault;
   public string s;  // short
   ```

By default (or with `PropertyField` from UI Toolkit), Unity display it as this (it's a IMGUI style even with UI Toolkit on):



Now, let's apply any UI Toolkit component (except `PropertyField`), it will (surprisingly!) use the modern UI Toolkit flavor label layout:



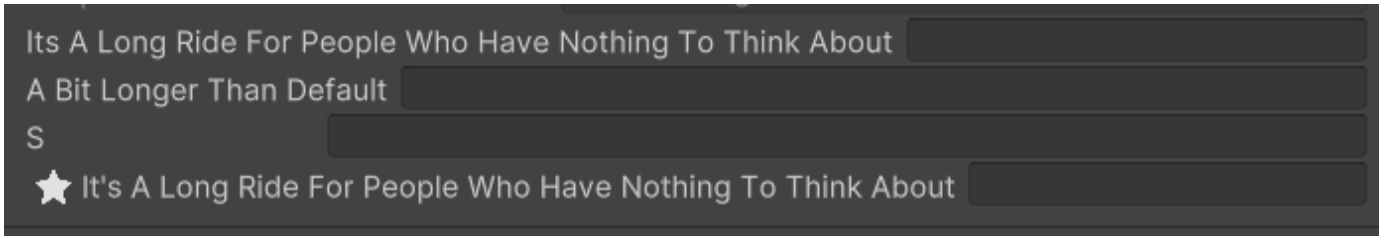
This inconsistency can make your inspector looks soooooooo weird and very funny because of un-aligned fields. This problem is reported to Unity but never got fixed. Considering:

```
// default field with UI Toolkit, Unity will truncate it to IMGUI width, instead of UI Too
public string thereIsSomeGoodNewsForPeopleWhoLoveBadNews;
// UI Toolkit component! Unity will grow the space like UI Toolkit
[UIToolkit] public string weWereDeadBeforeTheShipEvenSank;
// another default, Unity will use the IMGUI style width (some mix of a percent, min-width
public string myString;
// another UI Toolkit component! Unity will use the UI Toolkit style width (120px) like UI
[UIToolkit] public string myUiToolkit;
```

The field indent is a mess, even you're sticking to the UI Toolkit inspector:

This issue is so difficult to solve, that even OdinInspector does not try to fix it. (Or maybe they just don't care...?)

SaintsField now use some trick to make `PropertyField` label behaves much more like the vanilla UI Toolkit component:



That means:

i. Label will by default get 120px width, which is UI Toolkit's default
ii. The space gets grow to fit the label when the label gets longer, which is also UI Toolkit's default
iii. If you have a very looooong label, the value field will be shrank out of view. This is also how UI Toolkit works.

2. ~~`PropertyField`~~ ~~of an~~ ~~`Object`~~ ~~will give an error when click:~~ ~~`NullReferenceException: ... UnityEditor.ProjectBrower.FrameObject...`~~ ~~. Clicking will still lead you to active the target object, but I have no idea where this came from. Even official's example will have this error if you just add a~~ ~~`PropertyField`~~ ~~to it. Clicking on the error message will lead to the~~ ~~`Console`~~ ~~tab's layout got completely messed up.~~

   This is now fixed after dealing with the `PropertyField` binding.

3. `DropdownField` will tread a label's change as a value change... I have no idea why this happens and why only `DropdownField` . Luckily this change event will give a `newValue=null` so I can work around with it.

4. When leaving an PropertyDrawer to switch to a new target, the old one's `CreatePropertyGUI` will also get called once. This... makes the nested fallback difficult. Currently I use some silly way to work around with it, and you will see the inspector flick one frame at the beginning.

If you're in Unity 2022.2+ (from which Unity use UI Toolkit as default inspector), `SaintsField` will switch to UI Toolkit by default. In this case, if you want to use the IMGUI version, you can go `Window` - `Saints` - `Disable UI Toolkit Support` (See "Add a Macro" section for more information) to disable UI Toolkit.