SaintsField

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

Developed by: TylerTemp , 墨瞳

Unity: 2020.2 or higher

1. Highlights

1. Use and only use `PropertyDrawer` and `DecoratorDrawer`, thus it will be compatible with most Unity Inspector enhancements like `NaughtyAttributes` and your custom drawer.
2. 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.
3. Allow dynamic arguments in many cases
4. Works on deep nested fields too!

2. Installation

- 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 only image files (no `.meta` files) from project's `Editor/Editor Default Resources/SaintsField` into your project's `Assets/Editor Default Resources/SaintsField`.
- Select all the image files you copied, and enable the `Advanced - Read/Write` option for these icons.

3. Change Log

1.1.3

- `AnimatorParam` no longer offers `null` value
- `AnimatorParam` and `AnimatorState` now will try to find the `animator` on current object if the name of `animator` is not set
- Use standard field picker for `layer` and `tag`
- Add `LeftToggle`
- Fix a issue that when using `Scene` with a string without default value, it would display the first item but the actually value is null or empty string. Now it will sign the first value on it.
- Fix a issue that `Scene` will display empty when your scene name starts with an underscore.

See the full change log .

4. Document

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 (This will always fallback to built-in editor resources by name, as it uses `EditorGUIUtility.Load`):

  - `"Assets/Editor Default Resources/"` (You can override things here, or put your own icons),
  - `"Assets/Editor Default Resources/SaintsField/"` (again for override)
  - `"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`)

  for `color` it supports:

  - `clear`, `white`, `black`, `gray`, `red`, `pink`, `orange`, `yellow`, `green`, `blue`, `indigo`, `violet` (all in lower case)

  - 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

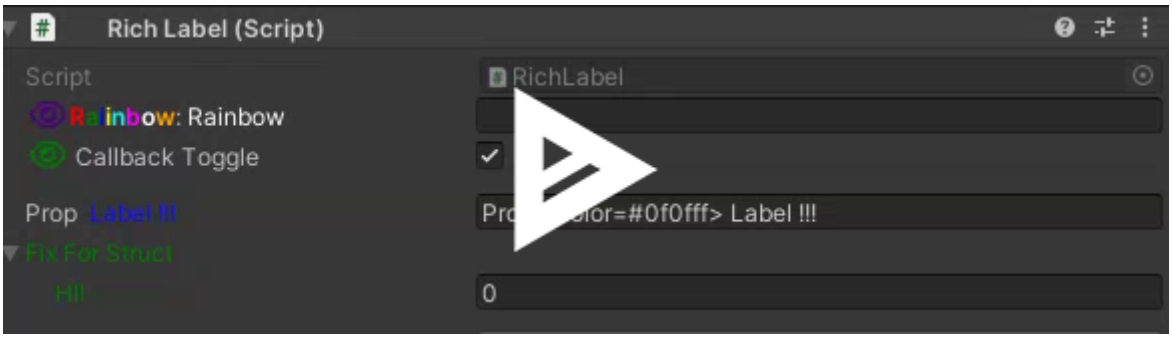
- AllowMultiple: No. A field can only have one `RichLabel`

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:

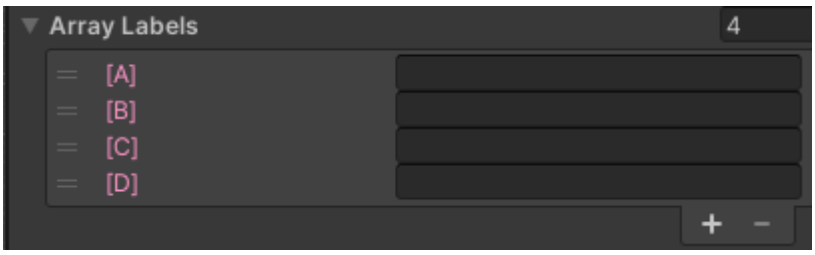
1. Ensure you make it a callback: `isCallback=true`
2. Your function must receive one `int` argument
3. The `int` argument will receive a value from `0` to `length-1` of the array/list
4. Return the desired label content from the function

```
public class RichLabel: MonoBehaviour
{
    [RichLabel("<color=indigo><icon=eye.png /></color><b><color=red>R</color><color=green>a</c
    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)}]";
```

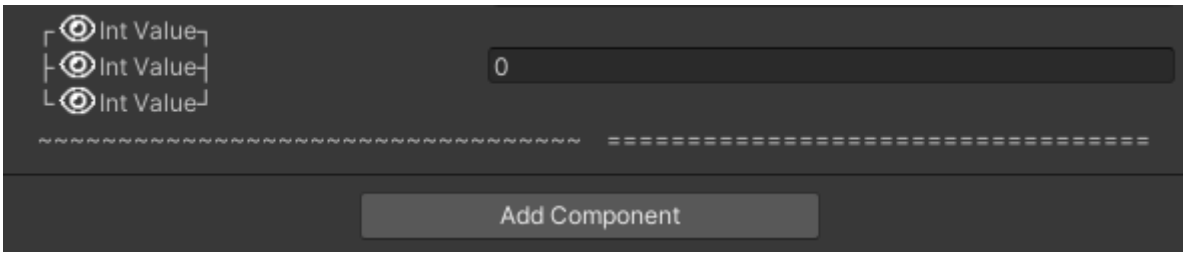


4.1.2. `AboveRichLabel` / `BelowRichLabel`

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`

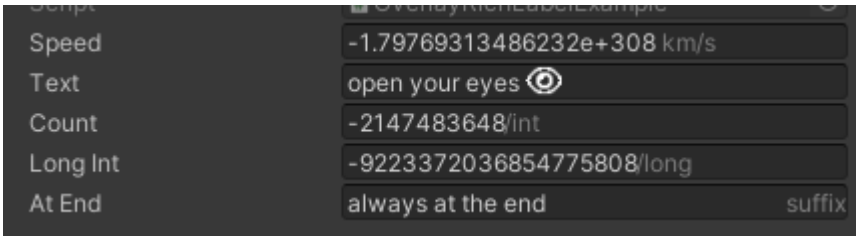
Like `RichLabel`, but it's rendered on top of the field.

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;
}
```



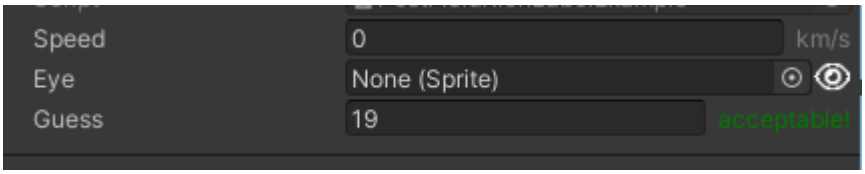
4.1.4. `PostFieldRichLabel`

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 contentIsCallback=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 `(string content, EMessageType messageType)` then both content and message type will be changed

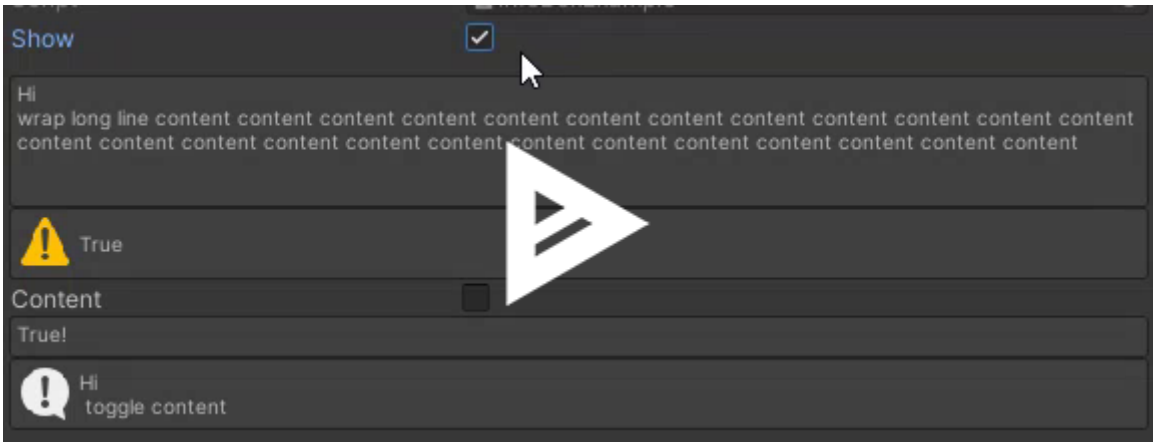
- `bool above=false`

  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 conte
    [InfoBox(nameof(DynamicMessage), EMessageType.Warning, contentIsCallback: true, above: tru
    [InfoBox(nameof(DynamicMessageWithIcon), contentIsCallback: true)]
    [InfoBox("Hi\n toggle content ", EMessageType.Info, nameof(_show))]
    public bool _content;

    private (EMessageType, string) DynamicMessageWithIcon => _content ? (EMessageType.Error, "
    private string DynamicMessage() => _content ? "False" : "True";
}
```

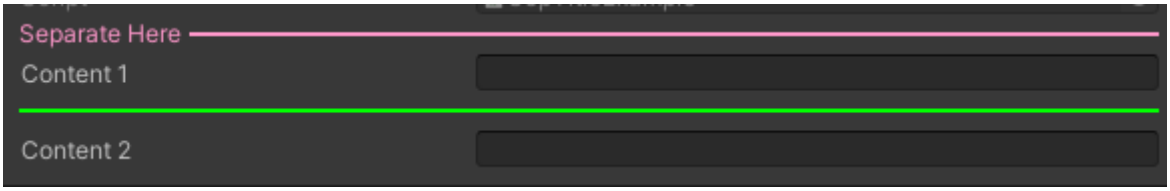


4.1.6. `SepTitle`

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`

  label of the button, support tags like `RichLabel`

- `bool buttonLabelIsCallback = false`

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

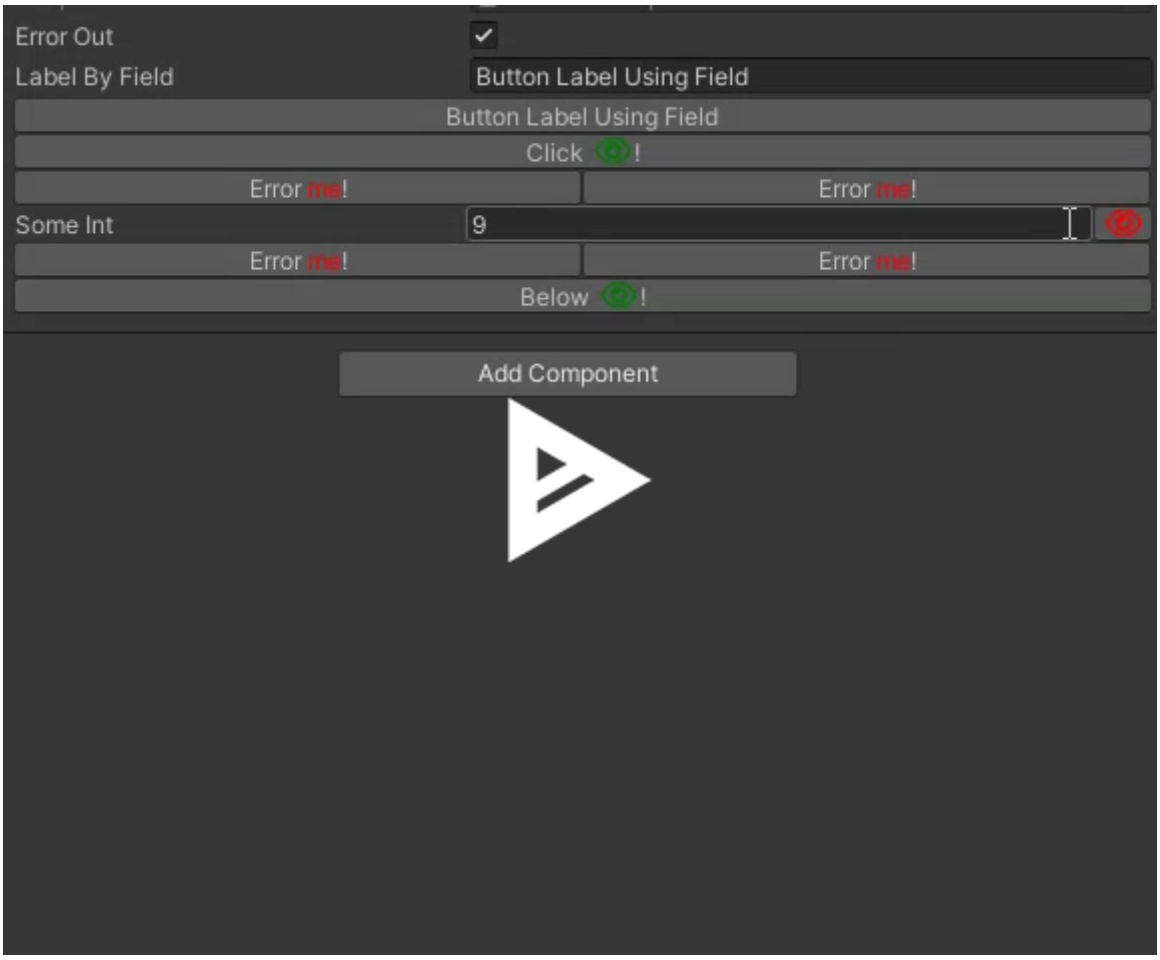
- `string groupBy = ""`

  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.

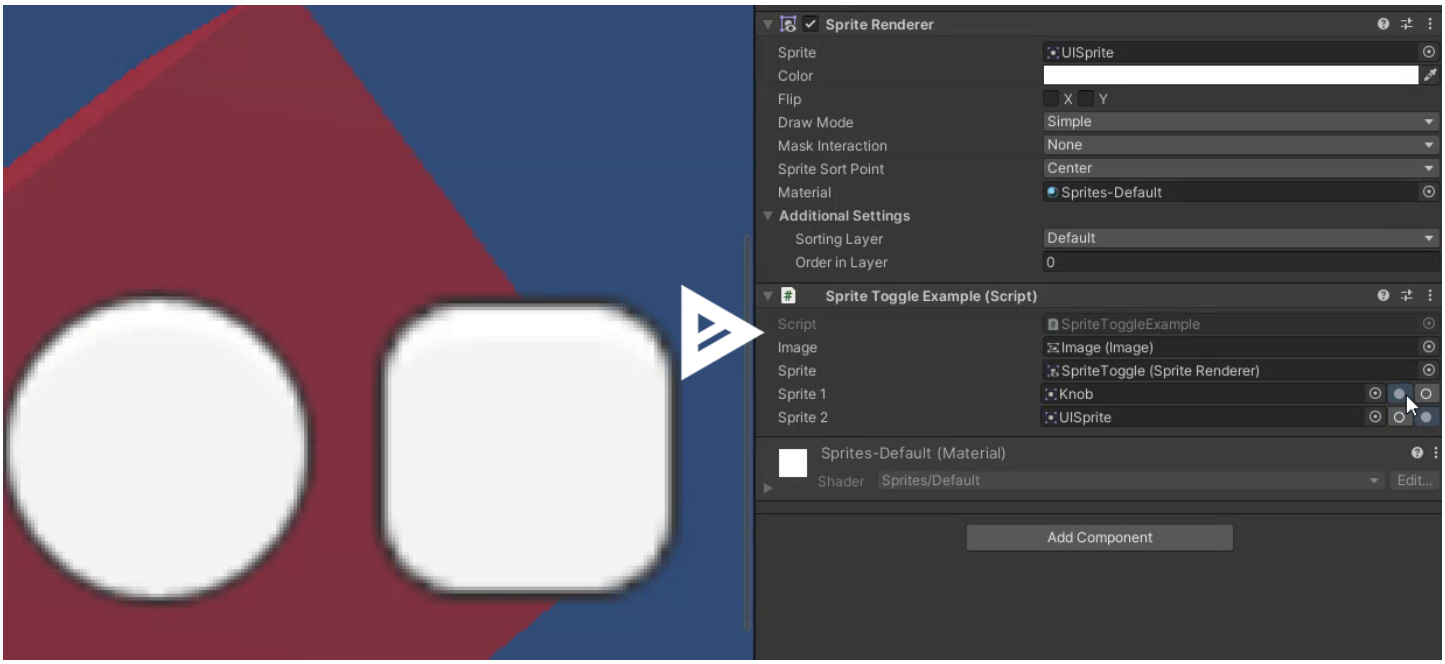
The field itself must be `Sprite`.

- `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`

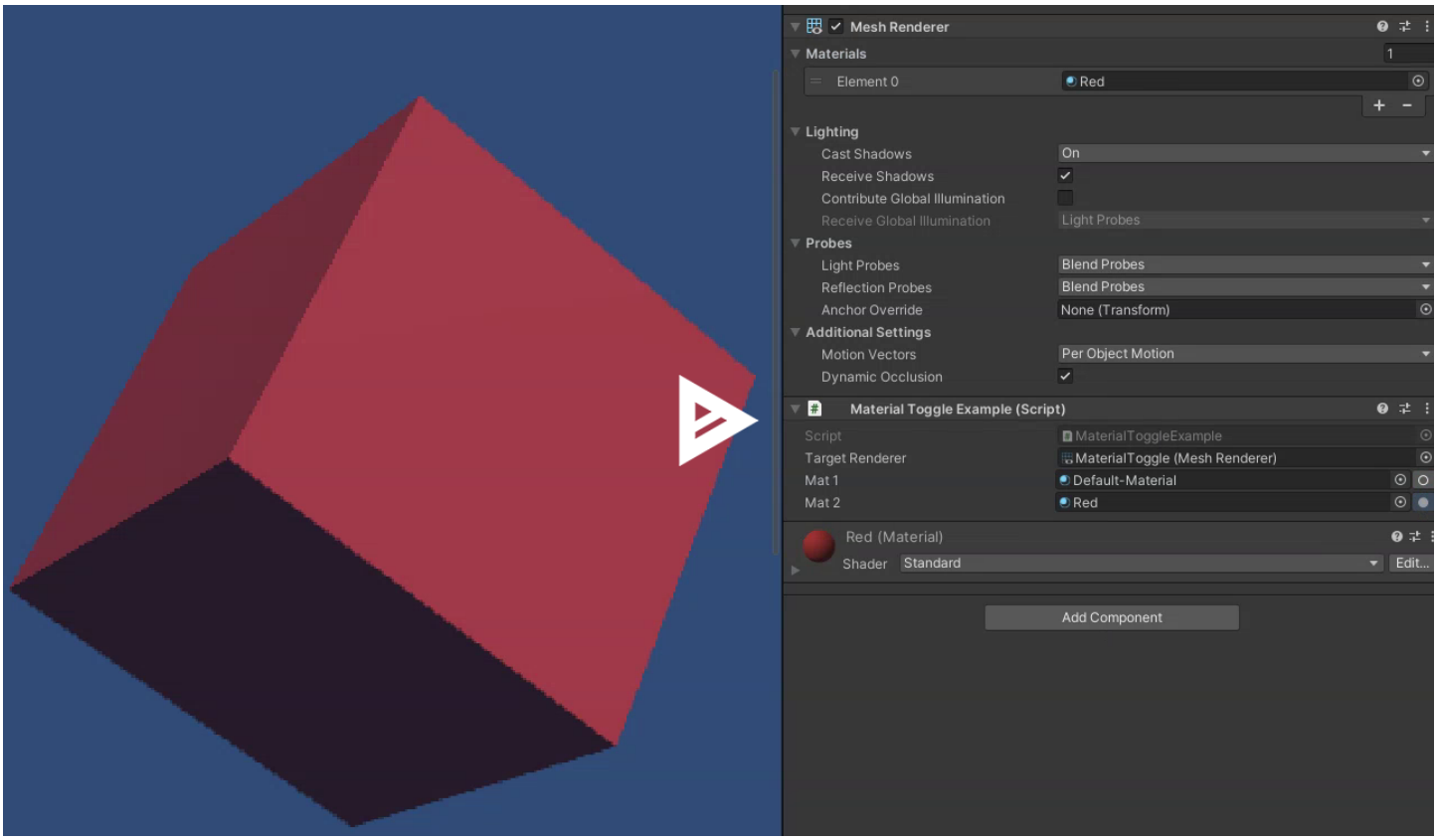
  the target, must be `Renderer` (or its subClass like `MeshRenderer`). When using null, it will try to get the `Renderer` component from the current component

- `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 `Material`.

- `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.

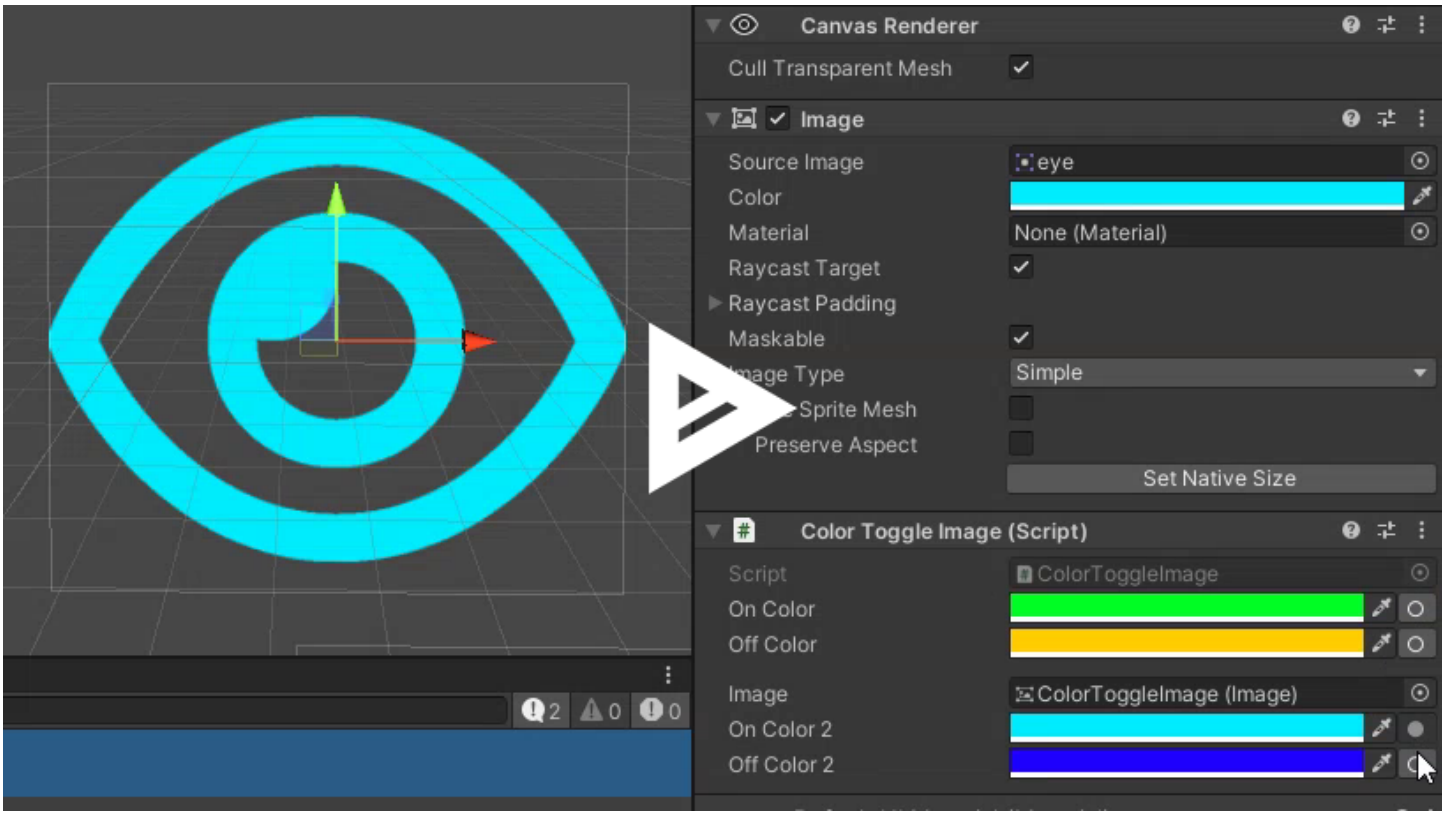
- `int index=0`

  (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;
}
```

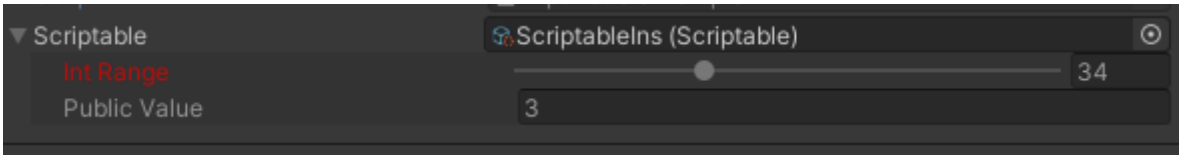


4.3.5. `Expandable`

Make scriptable objects expandable.

- AllowMultiple: No

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



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`

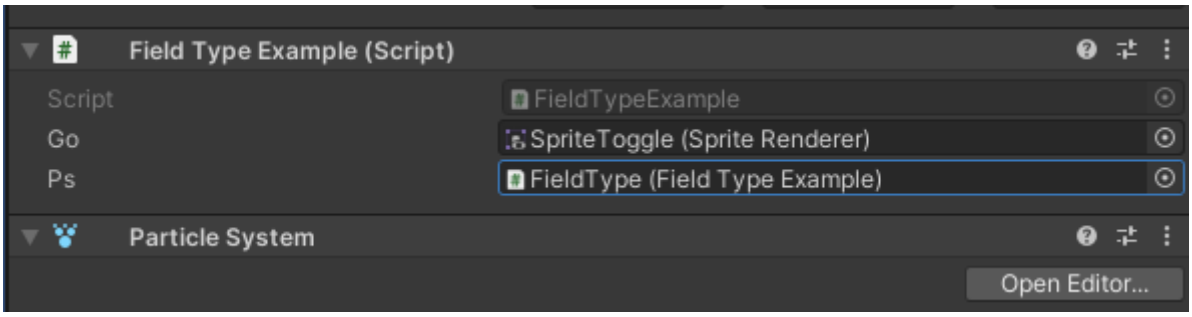
Ask the inspector to display another type of field rather than the field's original type.

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.

- AllowMultiple: No

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

    [SerializeField, FieldType(typeof(FieldTypeExample))]
```

```
    private ParticleSystem _ps;
}
```

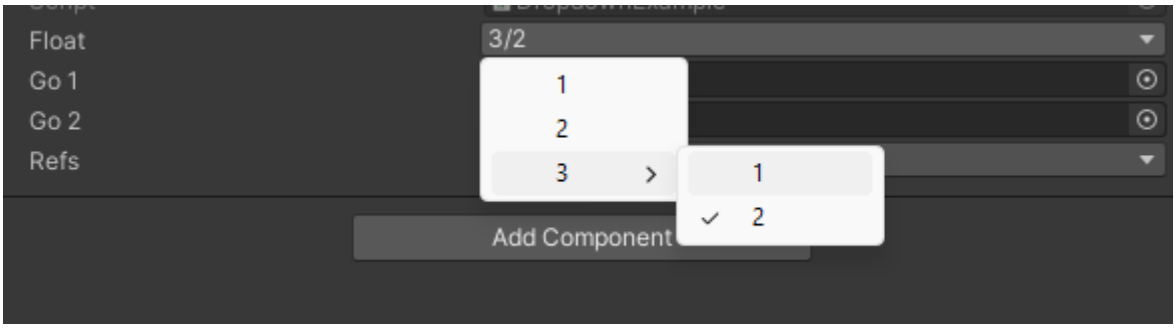


4.4.3. `Dropdown`

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

- AllowMultiple: No

```
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},
    };
}
```

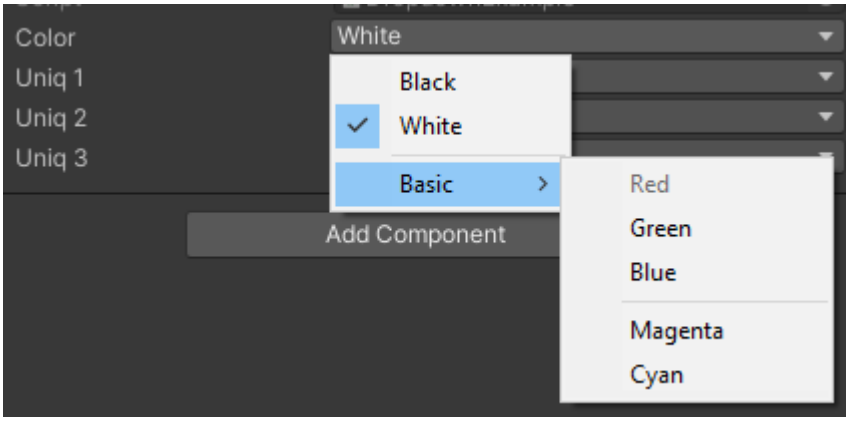


To control the separator and disabled item

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

private DropdownList<Color> GetAdvancedDropdownItems()
{
    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
```



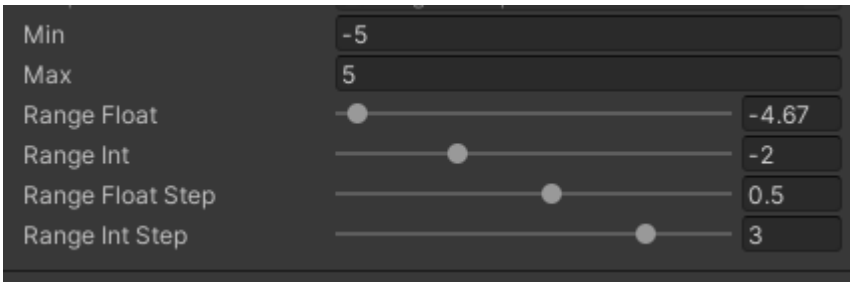
4.4.4. `PropRange`

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;
}
```



4.4.5. `MinMaxSlider`

A range slider for `Vector2` or `Vector2Int`

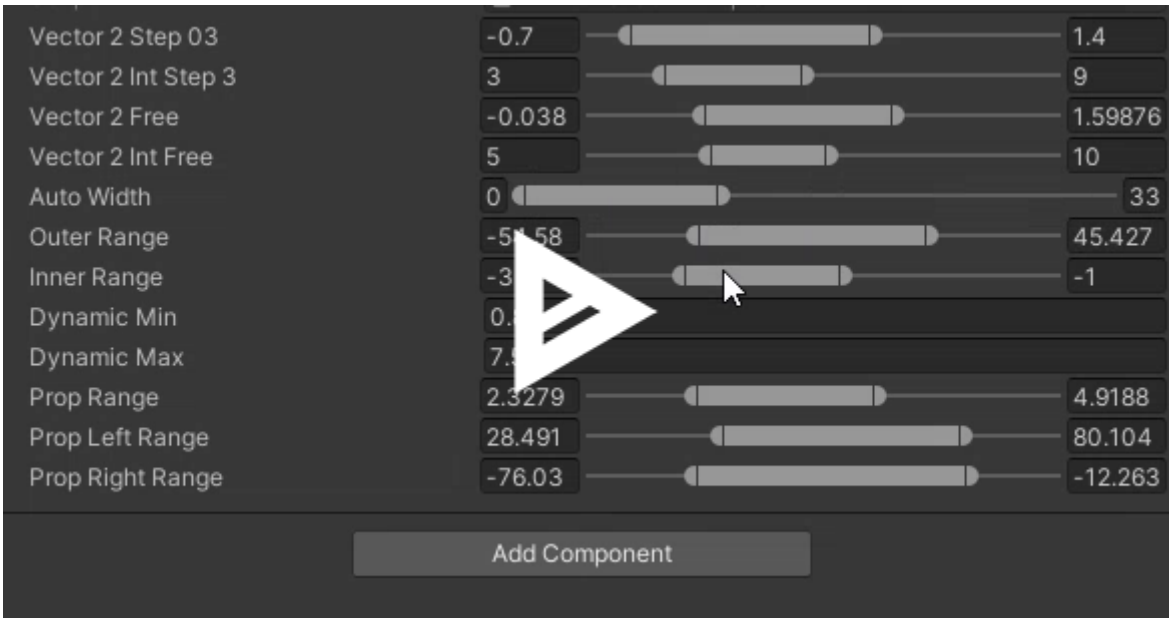
For each argument:

- `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` : the minimum width of the value label. `< 0` for auto size (not recommended)

- `float maxWidth=50f` : 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 Vector2

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



4.4.6. `EnumFlags`

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

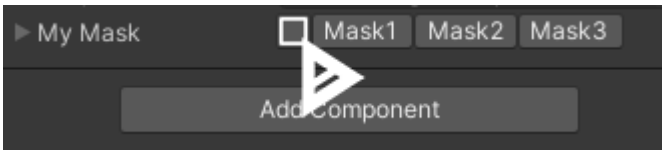
This field has compact mode and expanded mode.

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;
}
```



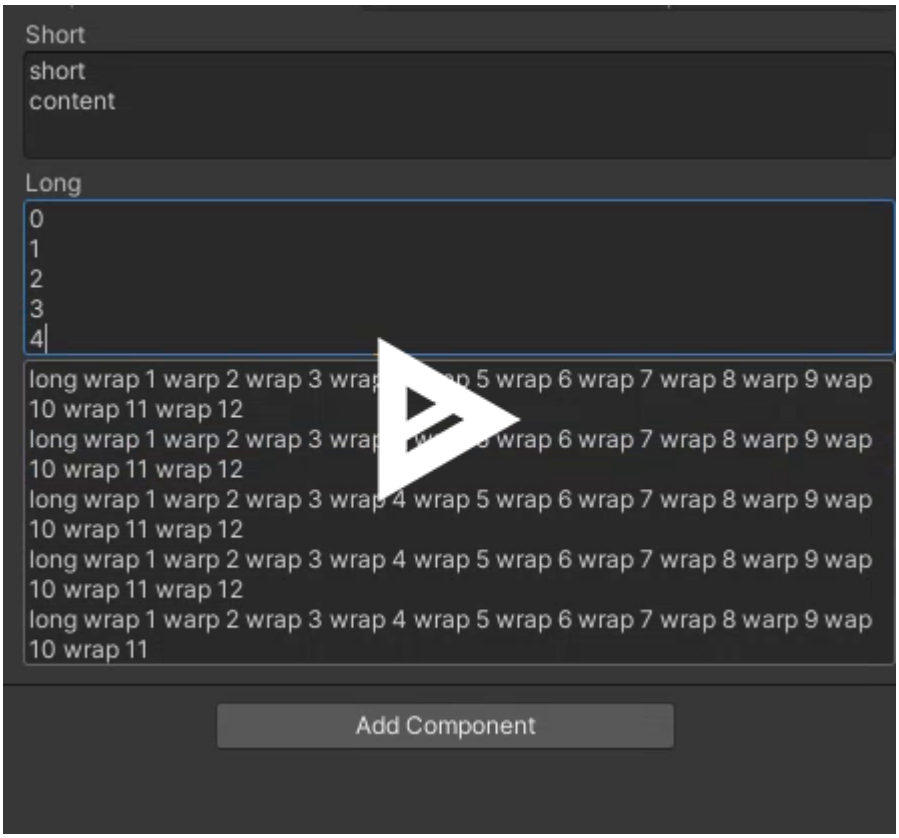
4.4.7. `ResizableTextArea`

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;
}
```



4.4.8. `AnimatorParam`

A dropdown selector for an animator parameter.

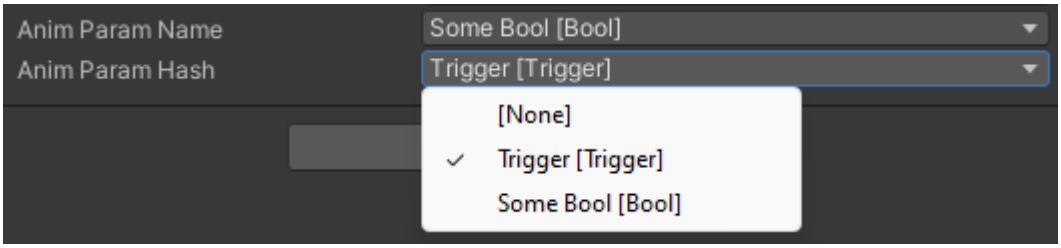
- `string animatorName=null`

  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;
}
```



4.4.9. `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

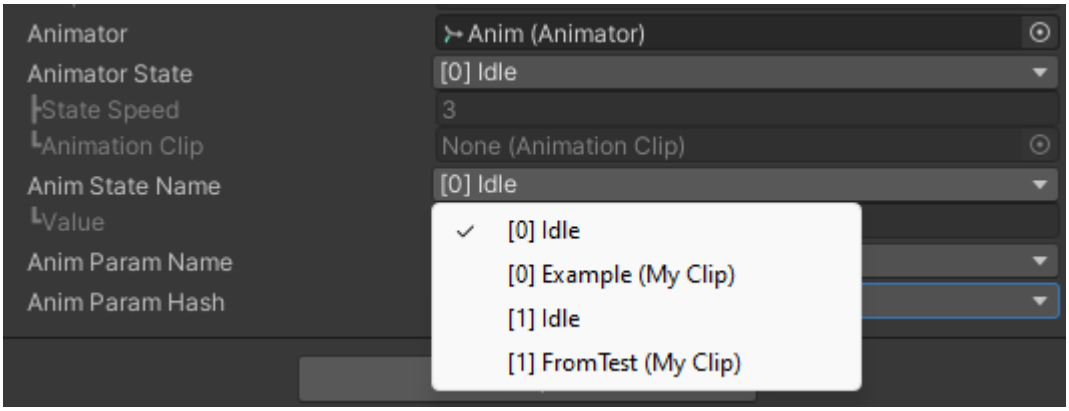
to get more useful info from the state, you can use `AnimatorState` type instead of `string` type.

`AnimatorState` 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
- `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

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

    [AnimatorState(nameof(Animator))]
```

```
    public AnimatorState animatorState;

    [AnimatorState(nameof(Animator))]
    public string animStateName;
}
```



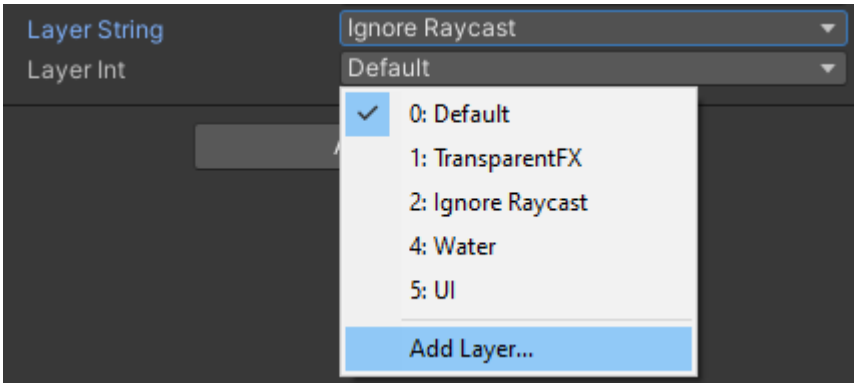
4.4.10. `Layer`

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;
}
```

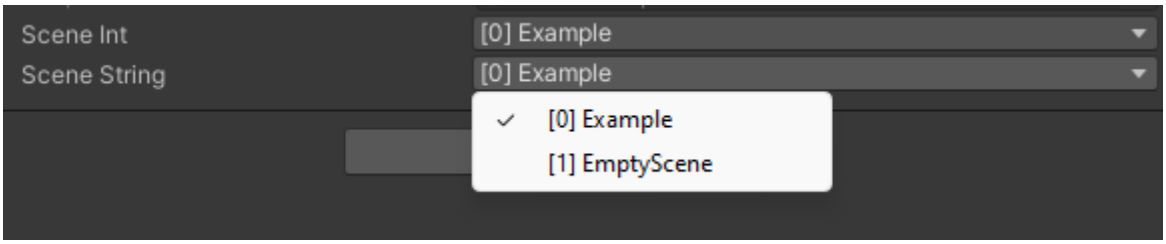


4.4.11. `Scene`

A dropdown selector for a scene in the build list.

- AllowMultiple: No

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

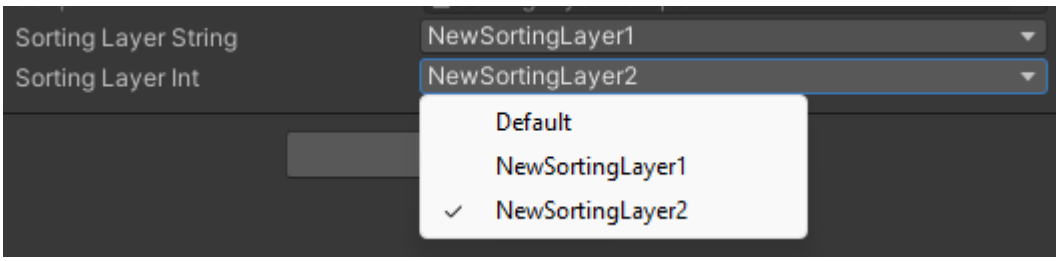


4.4.12. `SortingLayer`

A dropdown selector for sorting layer.

- AllowMultiple: No

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

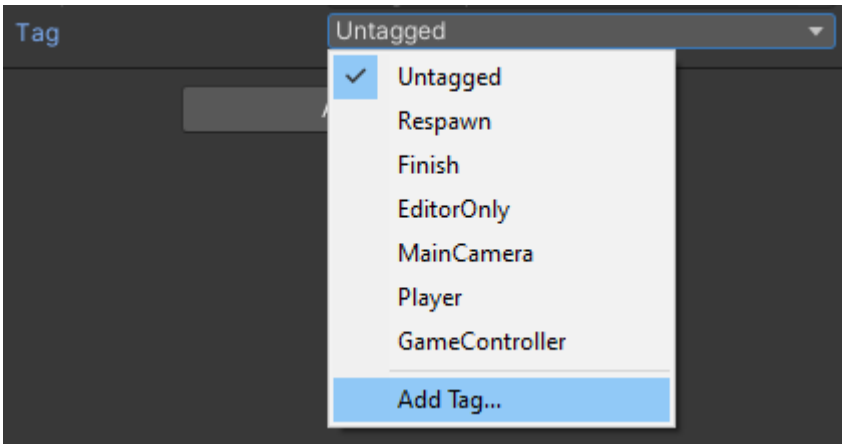


4.4.13. `Tag`

A dropdown selector for a tag.

- AllowMultiple: No

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

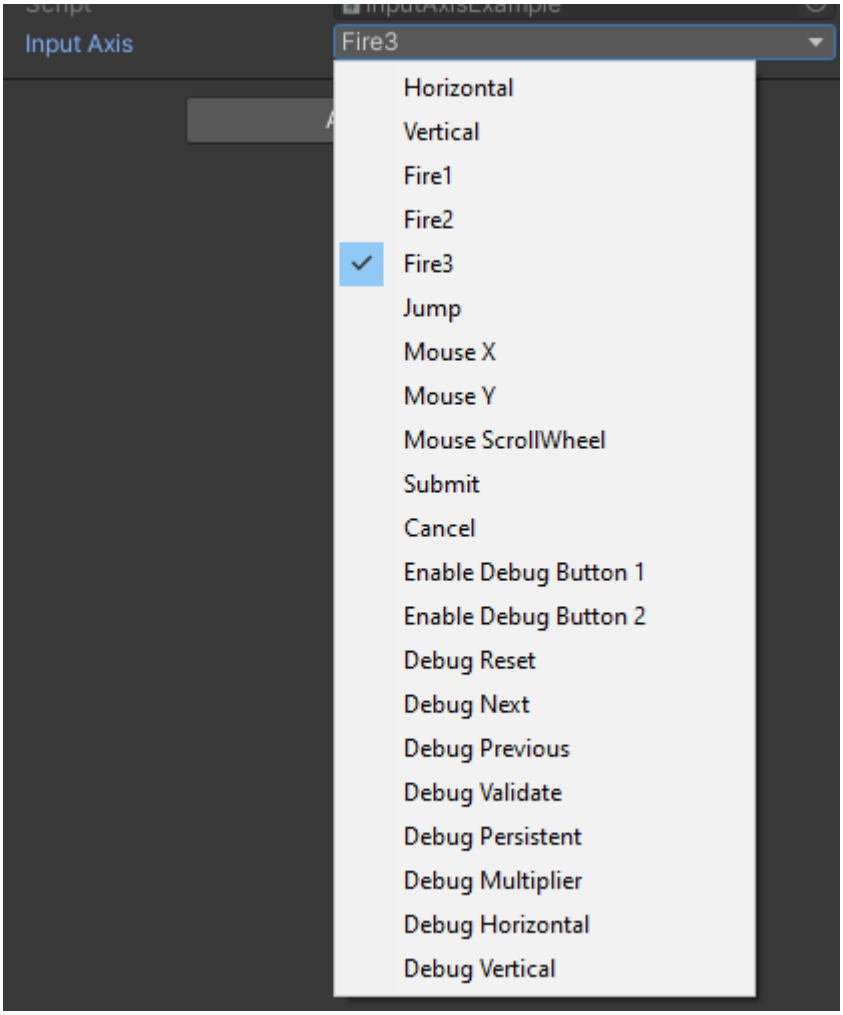


4.4.14. `InputAxis`

A string dropdown selector for an input axis.

- AllowMultiple: No

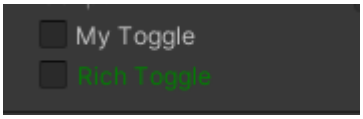
```
public class InputAxisExample: MonoBehaviour
{
    [InputAxis] public string inputAxis;
}
```



4.4.15. `LeftToggle`

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

```
public class LeftToggleExample: MonoBehaviour
{
    [LeftToggle] public bool myToggle;
    // To use with `RichLabel`, you need to add 6 spaces ahead as a hack
    [LeftToggle, RichLabel("      <color=green><label />")] public bool richToggle;
}
```



4.5. Field Utilities

4.5.1. `AssetPreview`

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

- `int maxWidth=-1`

  preview max-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 maxHeight=-1`

  preview max 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

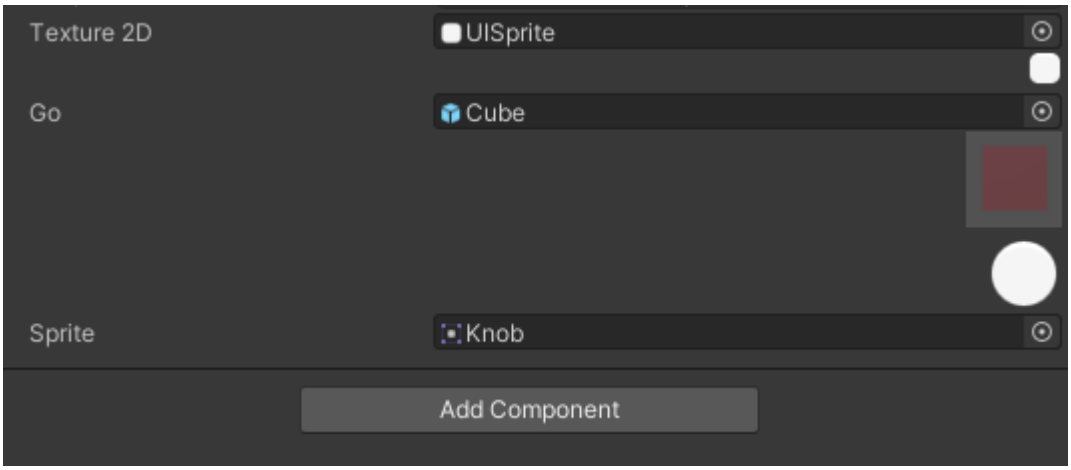
- `string groupBy=""`

  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`

  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`

- `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
- AllowMultiple: Yes

```
public class OnChangedExample : MonoBehaviour
{
    [OnValueChanged(nameof(Changed))]
    public int _value;

    private void Changed()
    {
        Debug.Log($"changed={_value}");
    }
}
```

4.5.4. `ReadOnly`

This has two overrides:

- `ReadOnlyAttribute(bool directValue)`
- `ReadOnlyAttribute(params string[] by)`

Each arguments:

- `bool directValue=false`

  if true, the field will be read-only

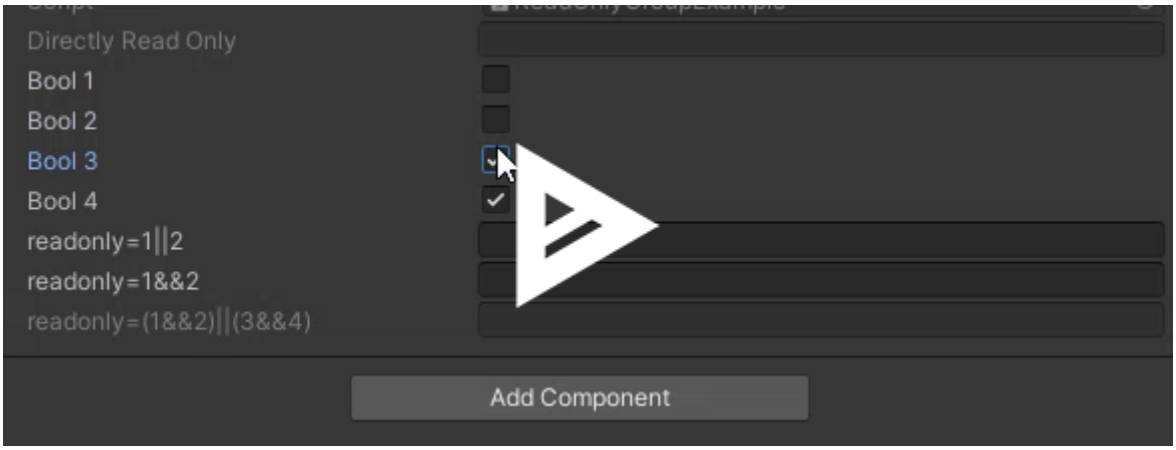
- `string[] by`

  a callback or property name, if **ALL** the value is truly, the field will be read-only

- AllowMultiple: Yes

  When using multiple `ReadOnly` on a field, the field will be read only if **ANY** of them is read-only

```
public class ReadOnlyGroupExample: MonoBehaviour
{
    [ReadOnly(true)] 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;
}
```



4.5.5. `Required`

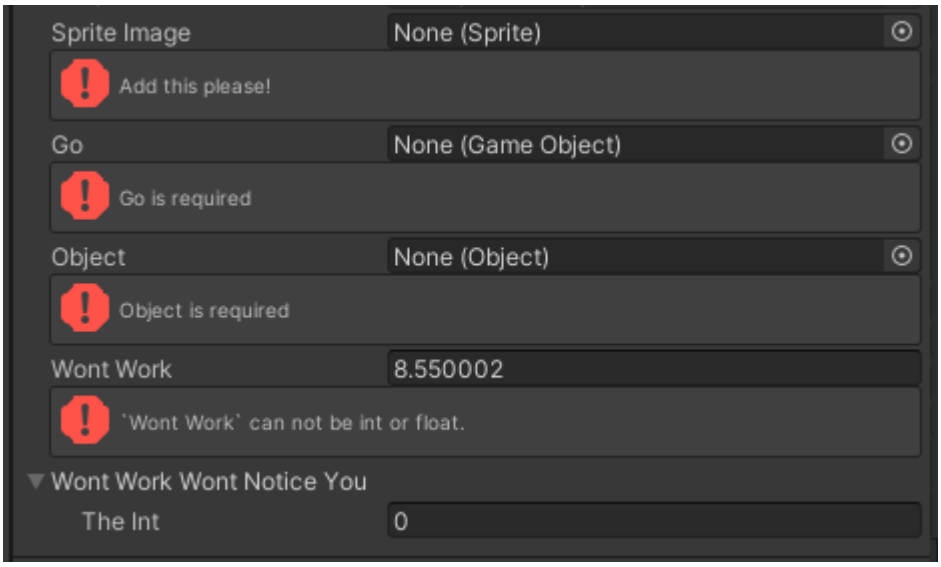
Remide a given reference type field to be required.

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

1. Won't work for int and float (It'll give an error, asking you to not use on int/float)
2. The `struct` value will always be `truly` because `struct` is not nullable and Unity will fill a default value for it no matter what
3. It works on reference type and will NOT skip Unity's life-circle null check

- `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 wontWorkWontNoticeYou;
}
```

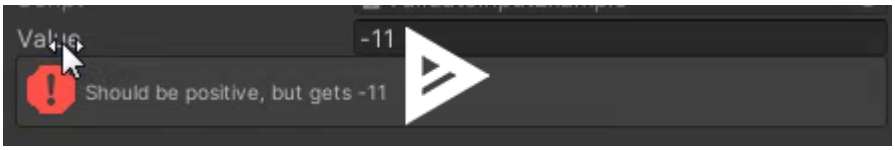


4.5.6. `ValidateInput`

Validate the input of the field when the value changes.

- `string callback` is the callback function to validate the data. note: return type is `string` not bool! return a null or empty string for valid, otherwise, the string will be used as the error message
- AllowMultiple: Yes

```
public class ValidateInputExample : MonoBehaviour
{
    [ValidateInput(nameof(OnValidateInput))]
    public int _value;

    private string OnValidateInput() => _value < 0 ? $"Should be positive, but gets {_value}"
}
```



4.5.7. `ShowIf` / `HideIf`

Show or hide the field based on a condition.

For `ShowIf` :

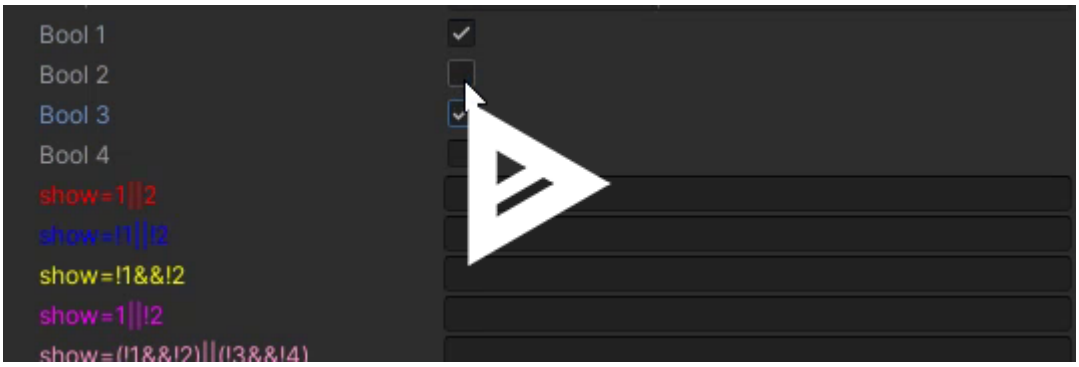
- `string andCallbacks...` a list of callback or property names, if **ALL** the value is truly, the field will be shown/hidden
- AllowMultiple: Yes

When using multiple `ShowIf` on a field, the field will be shown if **ANY** of them is shown

`HideIf` is the opposite of `ShowIf`. You can use multiple `ShowIf`, `HideIf`, and even a mix of the two

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")]
    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)")]
    public string _hideIf1234;
}
```



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`

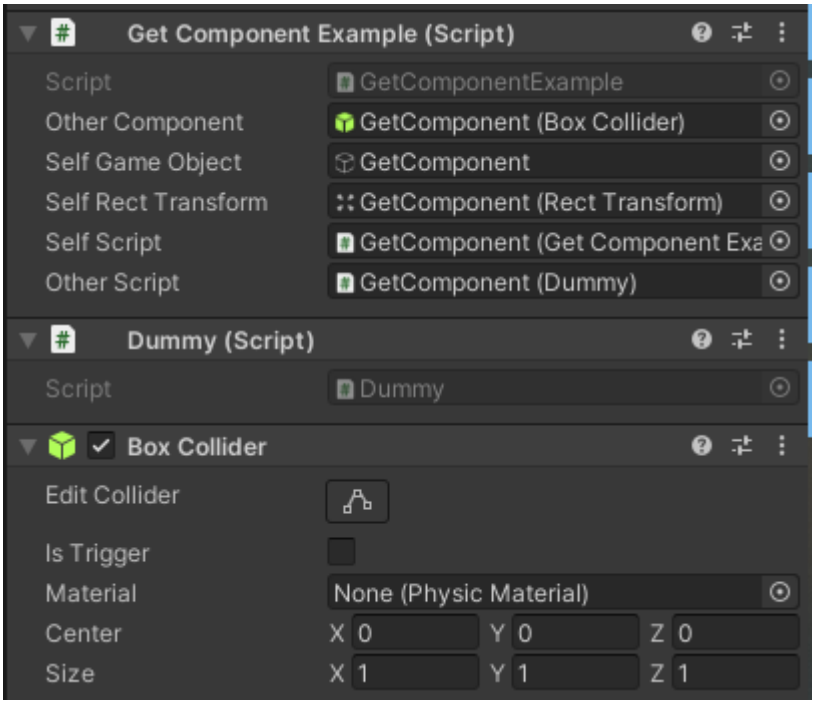
  The component type to sign. If null, it'll use the field type.

- `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 its
    [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`

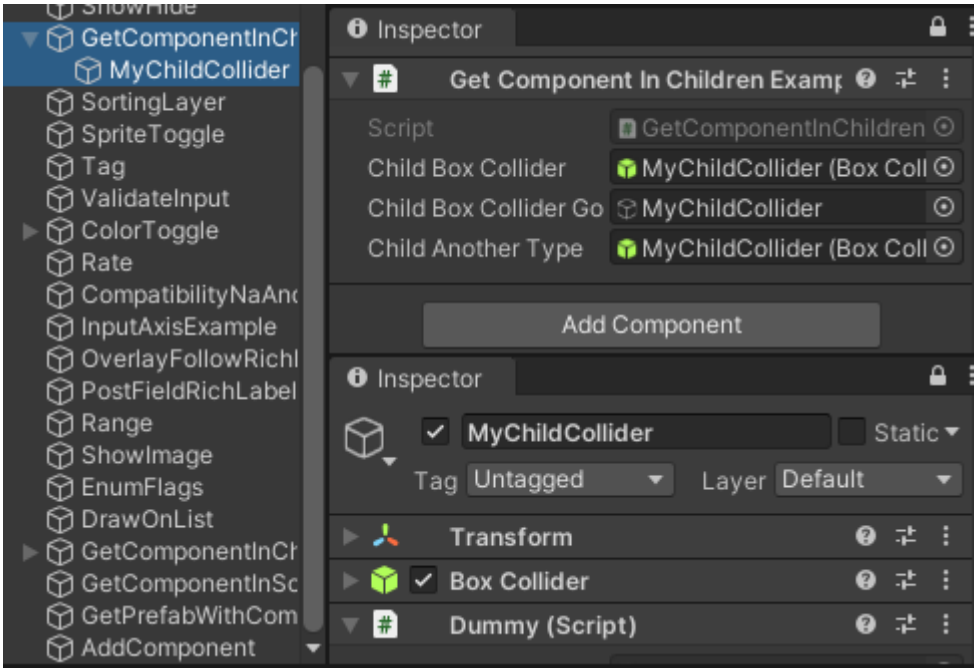
  The component type to sign. If null, it'll use the field type.

- `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 childBoxCollider
}
```



4.5.11. `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.

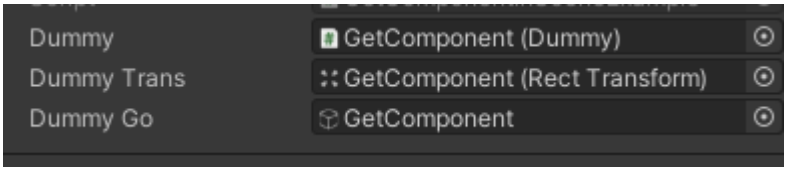
- `string groupBy = ""`

  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;
}
```



4.5.12. `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`

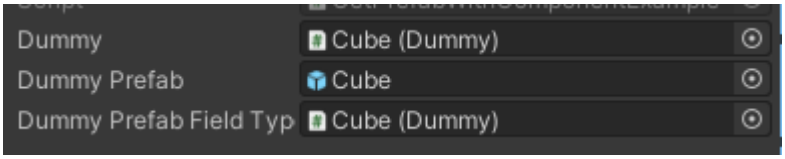
  The component type to sign. If null, it'll use the field type.

- `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 GameObj
}
```



4.5.13. `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`

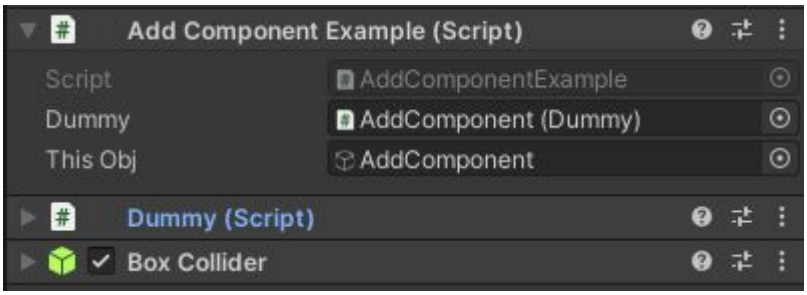
  The component type to add. If null, it'll use the field type.

- `string groupBy = ""`

  For error message grouping.

- AllowMultiple: Yes

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



5. 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.

6. Common Pitfalls & Compatibility

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;
}
```

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;
}
```

Multiple Fields Handling

Unlike `NaghthyAttributes` / `Odin`, `SaintsField` does not have a decorator like `Tag`, or `GroupBox` that puts several fields into one place because it does not inject a global `CustomEditor`.

For the same reason, it can not handle `NonSerializedField` and `AutoPropertyField` (unless you give a `[field: SerializedFile]` decorator to make it as a normal property). Because they are all not `PropertyAttribute`.

Other Drawers

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

1. the drawer itself respects the `GUIContent` argument in `OnGUI`

   NOTE: `NaughtyAttributes` uses `property.displayName` instead of `GUIContent`. You need to set `Label(" ")` if you want to use `RichLabel`. Also, `NaughtyAttributes` tread many Attribute 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`

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`.