SaintsField

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

Developed by: TylerTemp, 墨瞳

Unity Version Requirement: 2020.2 or higher

Highlights

1. Use and only use `PropertyDrawer` and `DecoratorDrawer`, thus it will be compatible with most Unity Inspector enhancer like `Odin` & `NaughtyAttributes`.
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 on many cases

Installation

- Using git upm:

  add this line to `manifest.json` in your project

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

- 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:

- 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.

Enhancements

Label & Text

`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/"` (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 install using `unitypackage`)
    - `"Packages/today.comes.saintsfield/Editor/Editor Default Resources/SaintsField/"` (this is most likely to be when install using `upm`)

  for `color` it supports:

    - `Clear` , `White` , `Black` , `Gray` , `Red` , `Pink` , `Orange` , `Yellow` , `Green` , `Blue` , `Indigo` , `Violet`
    - html color which 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`

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

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

     [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;
 }
```

https://github.com/TylerTemp/SaintsField/assets/6391063/25f6c7cc-ee7e-444e-b078-007dd6df499e

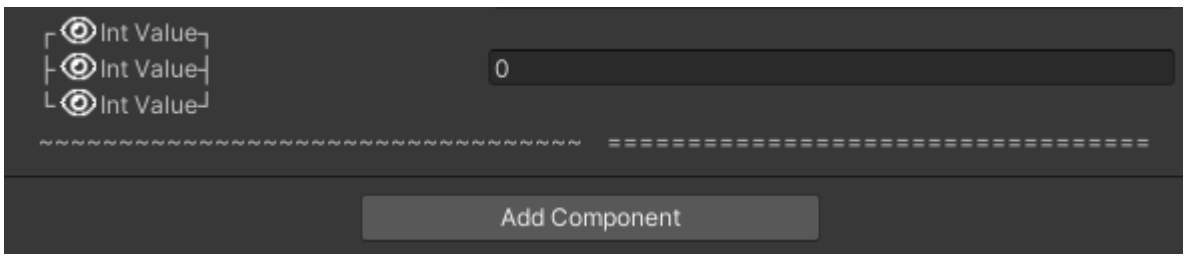
`AboveRichLabel` / `BelowRichLabel`

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

- `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 />┘";
 }
```



InfoBox

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

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

https://github.com/TylerTemp/SaintsField/assets/6391063/03ac649a-9e89-407d-a59d-3e224a7f84c8

`SepTitle`

A separator with a 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;
 }
```

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 propery name for button 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;
}
```

https://github.com/TylerTemp/SaintsField/assets/6391063/f225115b-f7de-4273-be49-d830766e82e7

Field Modifier

`GameObjectActive`

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

This does not require the field is `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;
}
```

https://github.com/TylerTemp/SaintsField/assets/6391063/ddb0bd02-8869-47b9-aac4-880ab9bfb81a

`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;
}
```

https://github.com/TylerTemp/SaintsField/assets/6391063/705498e9-4d70-482f-9ae6-b231cd9497ca

`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 it's 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;
 }
```

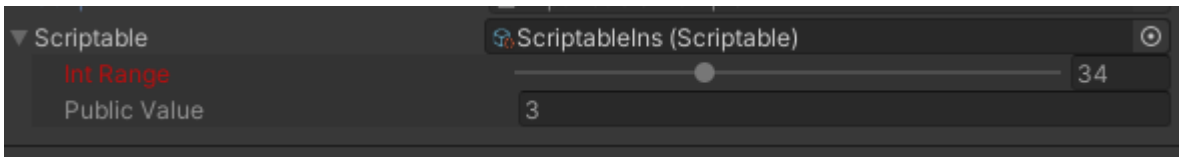
https://github.com/TylerTemp/SaintsField/assets/6391063/00c5702c-a41e-42a4-abb1-97a0713c3f66

`Expandable`

Make scriptable objects expandable.

- AllowMultiple: No

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



Field

`FieldType`

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

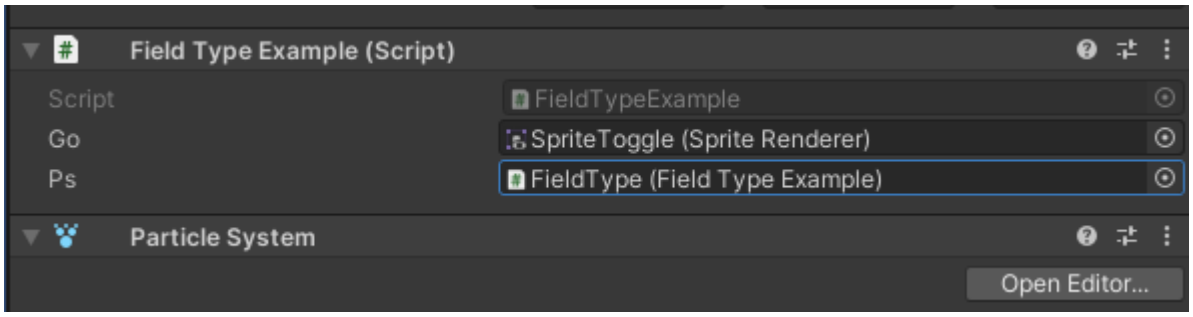
This is useful when you want to have a `GameObject` of prefab, but you want this target prefab has specific component (e.g. your own `MonoScript`, or a `ParticalSystem`). By using this you enforce the inspector to sign the required object that has your expected component, but still give you the original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

- AllowMultiple: No

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

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

```
}
```



Dropdown

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

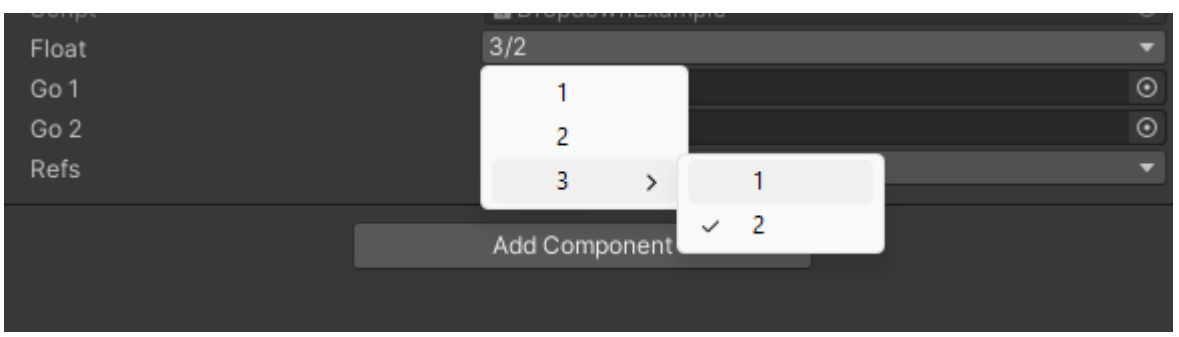
- AllowMultiple: No

```
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 separator and disabled item

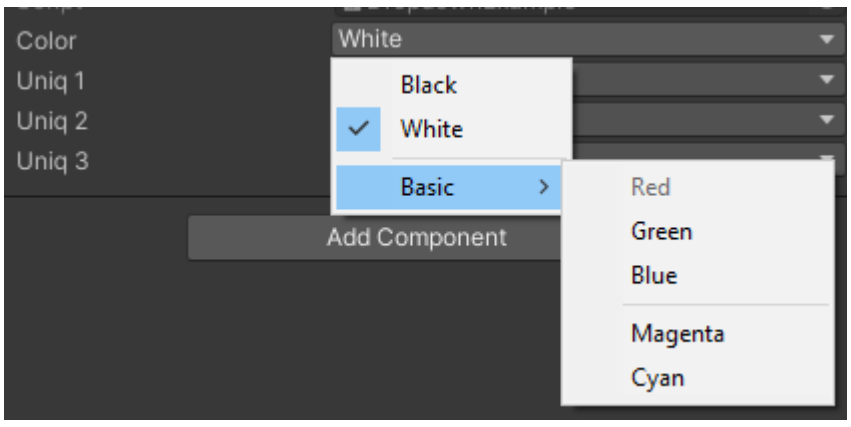
```
     [Dropdown(nameof(GetAdvancedDropdownItems))]
      public Color color;

      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
```



`MinMaxSlider`

A range slider for `Vector2` or `Vector2Int`

This Attribute has overrides:

- `MinMaxSliderAttribute(float min, float max, float step=-1f, float minWidth=DefaultWidth, float maxWidth=DefaultWidth)`
- `MinMaxSliderAttribute(int min, int max, int step=1, float minWidth=DefaultWidth, float maxWidth=DefaultWidth)`
- `MinMaxSliderAttribute(string minCallback, string maxCallback, int step=-1, float minWidth=DefaultWidth, float maxWidth=DefaultWidth)`
- `MinMaxSliderAttribute(float min, string maxCallback, float step=-1f, float minWidth=DefaultWidth, float maxWidth=DefaultWidth)`
- `MinMaxSliderAttribute(string minCallback, float max, float step=-1f, float minWidth=DefaultWidth, float maxWidth=DefaultWidth)`

For each arguments:

- `min` : the minimum value of the slider
- `max` : the maximum value of the slider

- `minCallback` : use a function or property as the minimum value of the slider
- `maxCallback` : use a function or property as the maximum value of the slider
- `step` : the step of the slider, `<= 0` means no limit and float type will not be limited
- `minWidth` : the minimum width of the value label. -1 for default (not recommended)
- `maxWidth` : the maximum width of the value label. -1 for default (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 Vector2Int _innerRange;

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

https://github.com/TylerTemp/SaintsField/assets/6391063/2ffb659f-a5ed-4861-b1ba-65771db5ab47

`ResizableTextArea`

This `TextArea` will always grow it's 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;
 }
```

https://github.com/TylerTemp/SaintsField/assets/6391063/64ad9c16-19e2-482d-9186-60d42fb34922

`AnimatorParam`

A dropdown selector for animator parameter.

- `string animatorName`

  name of the animator

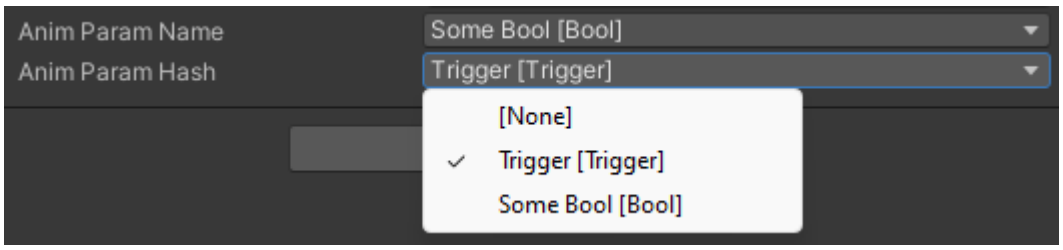
- (Optional) `AnimatorControllerParameterType animatorParamType`

  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;
 }
```



`AnimatorState`

A dropdown selector for animator state.

- `string animatorName`

  name of the animator

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

`AnimatorState` has the following properties:

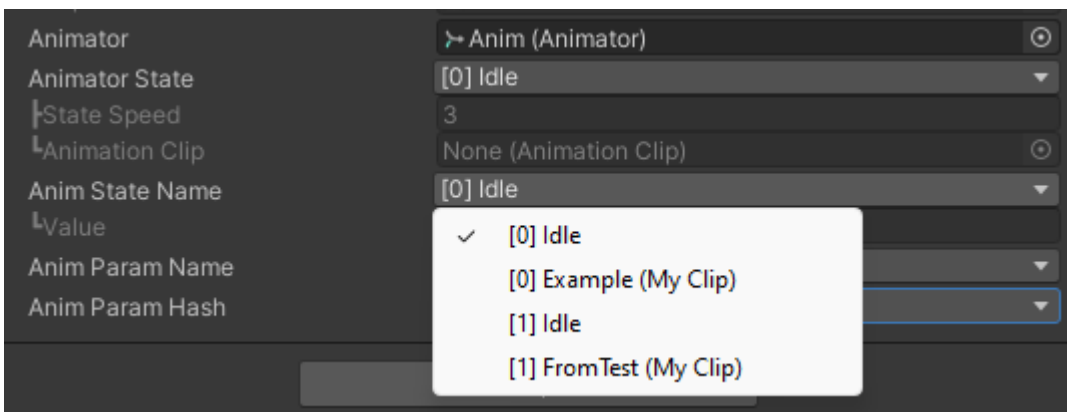
- `int layerIndex` index of layer
- `int stateNameHash` hash value of state
- `string stateName` actual state name
- `float stateSpeed` the `Speed` paramater of the state

- `AnimationClip animationClip` the actual animation clip of the state (can be null). It has `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;
 }
```

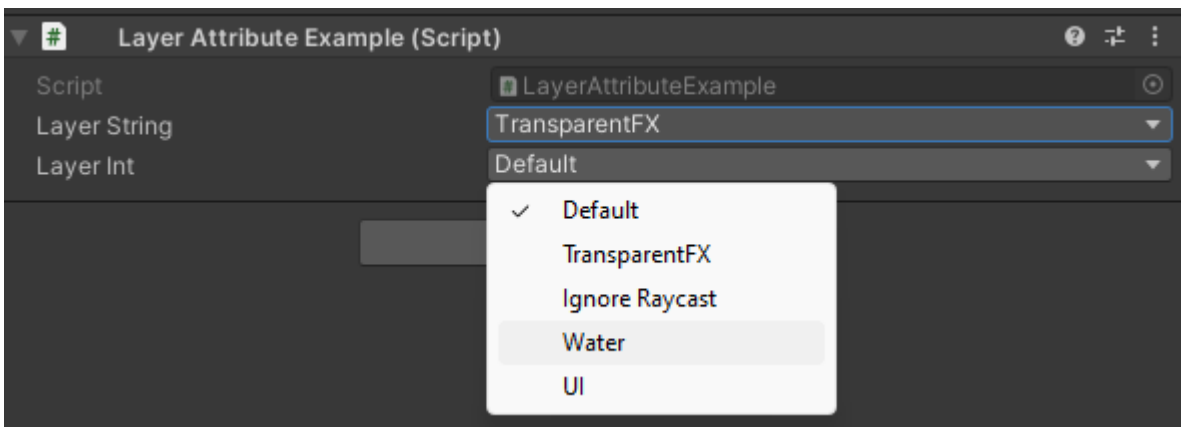


`Layer`

A dropdown selector for layer.

- AllowMultiple: No

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



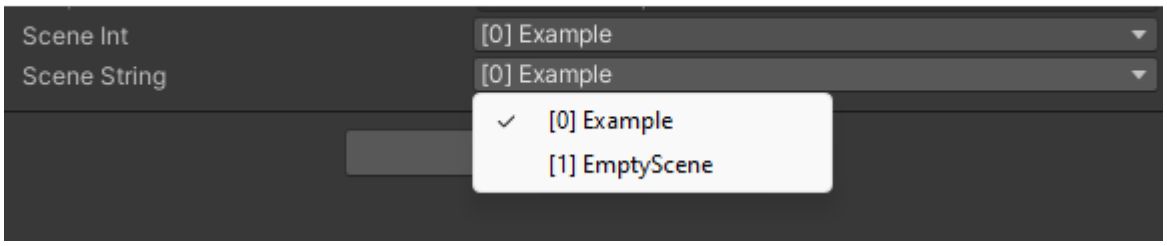
`Scene`

A dropdown selector for scene in build list.

- AllowMultiple: No

```
public class SceneExample: MonoBehaviour
 {
```

```
    [Scene] public int _sceneInt;
    [Scene] public string _sceneString;
}
```

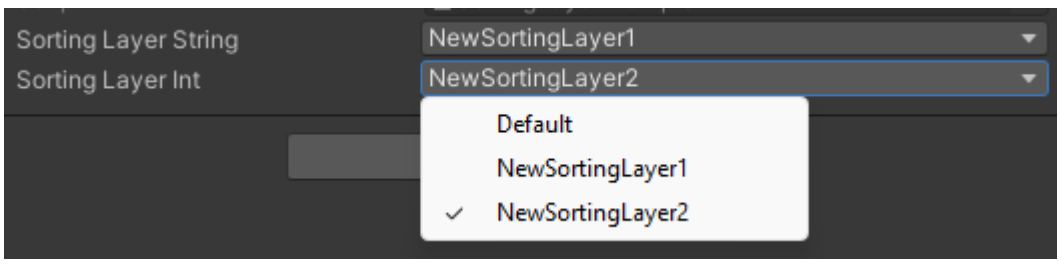


`SortingLayer`

A dropdown selector for sorting layer.

- AllowMultiple: No

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

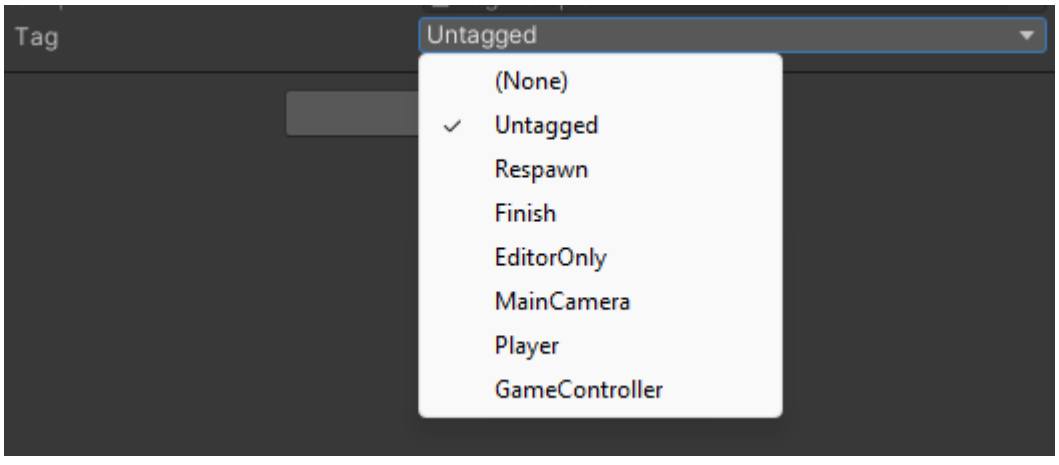


`Tag`

A dropdown selector for tag.

- AllowMultiple: No

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



Field Utilities

`AssetPreview`

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

- `int maxWidth=-1`

  preview max width, -1 for current view width

- `int maxHeight=-1`

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

- `bool above=false`

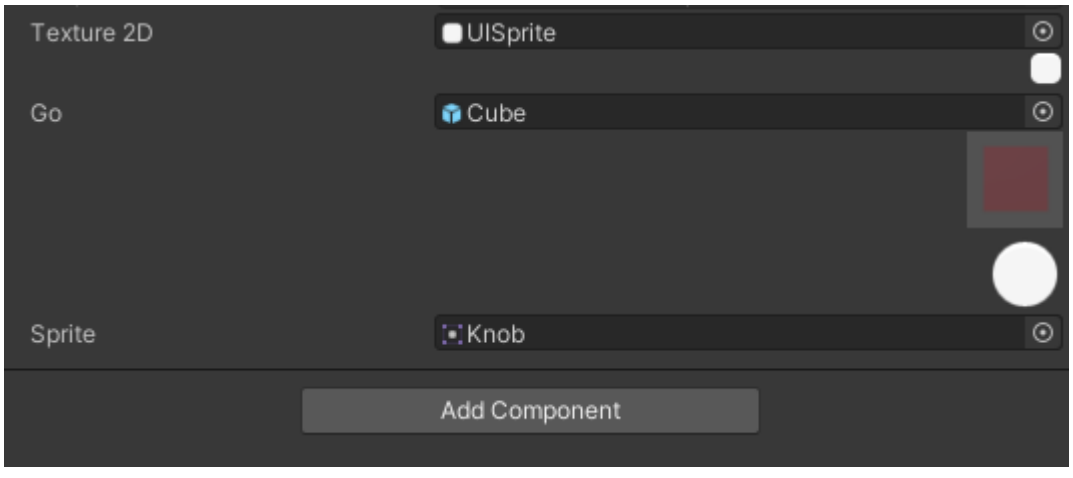
  if true, render above the field instead of below

- `string groupBy=""`

  See `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;
 }
```



`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}");
     }
 }
```

`ReadOnly`

This has two overrides:

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

Each arguemnts:

- `bool directValue=false`

  if true, the field will be readonly

- `string[] by`

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

- AllowMultiple: Yes

  When use mutiple `ReadOnly` on a field, the field will be readonly if **ANY** of them is readonly

```
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;
```

```
}
```

https://github.com/TylerTemp/SaintsField/assets/6391063/6761a0f2-07c2-4252-9dd0-c1a60091a891

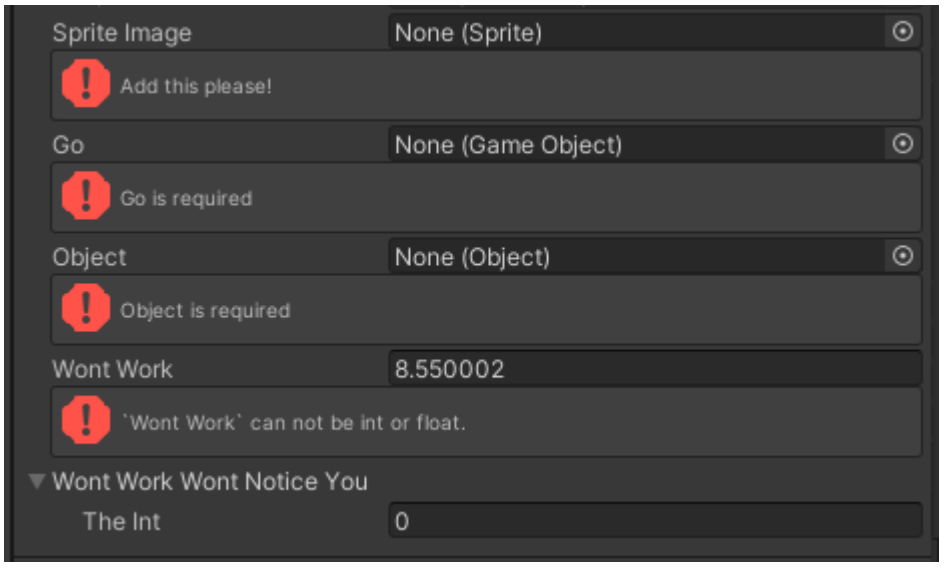
`Required`

Remide a given reference type field to be required.

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

1. Won't work for int and float (It'll give an error, asking you to not use on int/float)
2. `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 refenece 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 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;
 }
```

`ValidateInput`

Validate the input of the field when value changed.

- `string callback` the callback function to validate the data. note: return type is **`string`** not bool! return null or empty string for valid, otherwise the string will be used as 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}" : null;
 }
```

https://github.com/TylerTemp/SaintsField/assets/6391063/9d52e663-c9f8-430a-814c-011b17b67a86

`ShowIf` / `HideIf`

Show or hide the field based on a condition.

For `ShowIf`:

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

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

`HideIf` is the oppsite of `ShowIf`. You can use mutiple `ShowIf`, `HideIf` and even the mix of the two

A full futured 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;
}
```

https://github.com/TylerTemp/SaintsField/assets/6391063/dc7f8b78-de4c-4b12-a383-005be04c10c0

`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;
}
```

https://github.com/TylerTemp/SaintsField/assets/6391063/ea2efa8d-86e6-46ba-bd7d-23e7577f7604

GroupBy

group with any decorator that has the same `groupBy` for this field. Same group will share even 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 below drawer, and vice versa.

`""` means no group.

Known Issues

~~Because it handle the label drawing process seperated from the actual field drawing, the label is not using Unity's default label. Which lead to the problem:~~

1. ~~Dragging label to change value for int/float field is not very smooth~~ (FIXED)
2. ~~Click a label will not focus the field~~ (FIXED)

~~Note: These issue only affect the fields that uses this project's feature. It won't affect other fields, even it's on the same script. It should be safe enough to use it together with your other inspector enhancement~~