Apex Inspector

Documentation

Tamerlan Favilevich

Table of contents

1. Manual 3
1.1 Getting Started 3
1.2 Attribute Types 4
1.3 Global Settings 9
2. Attributes 11
2.1 View 11
2.2 Validator 22
2.3 Painter 27
2.4 Group 34
2.5 Conditional 38
2.6 Misc 42
2.7 Button 48
3. API References 52
3.1 Custom View 52
3.2 Custom Validator 56
3.3 Custom Painter 62
3.4 Interfaces 67

1. Manual

1.1 Getting Started

Welcome to the Apex documentation!

The information described below is recommended for all users using Apex.

1.1.1 Requirements

Apex requires Unity 2019.4 LTS or above.

> **Warning**
>
> We aim to support all upcoming Unity versions, but keep in mind that **Alpha and Beta versions of Unity are not officially supported**. This is because Unity can introduce changes under the hood that break Apex functionality, and we need some time to adjust and push fixes.

1.1.2 Download

Head over to the Unity Asset Store to download Apex!

> **Note**
>
> **Apex requires one license per seat** because it is an editor extension.
>
> This means **each person** on your team using Apex **must have their own license**. For more information, see the Unity Asset Store **EULA under Section 2.3**.

1.1.3 Setup

> **Success**
>
> After the package has been imported, no action or configuration is required on your part!

1.2 Attribute Types

1.2.1 Views

Property view attributes completely change how the property is drawing and how you interact with it.

> **Info**
>
> Each property can only have one property view.

Demo

Let's create array of materials.

```
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    public Material[] array;
}
```

Default array view

Let's add `[Array]` attribute to array of materials.

> **Note**
>
> Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.
>
> ```
> using ApexInspector;
> ```

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    [Array]
    public Material[] array;
}
```

After adding [Array] attribute

1.2.2 Validators

Property validator attributes check properties for the valid of the specified conditions.

> **Info**
>
> Each property can have multiple property validators.

Demo

Let's create float value.

```
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    public float value;
}
```

Before use validators

Let's add `[MinValue(0)]` and `[MaxValue(10)]` attributes to clamp value range in 0-10.

> **Note**
>
> Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.
>
> ```
> using ApexInspector;
> ```

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    [MinValue(0)] [MaxValue(10)]
    public float value;
}
```

Array after using Apex [MinValue] and [MaxValue] attributes

1.2.3 Painters

Property painter attributes can add additional visual elements on property.

> **Info**
>
> Each property can have multiple property painters.

Demo

Let’s create GameObject field. And add to this field default Cube primitive.

```
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    public GameObject someObject;
}
```

Default GameObject view

Let’s add `[ObjectPreview]` attribute and set attribute parameter «expandable» on true. `[ObjectPreview(true)]`

> **Note**
>
> Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.
>
> ```
> using ApexInspector;
> ```

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    [ObjectPreview(true)]
    public GameObject someObject;
}
```

GameObject field after using Apex [ObjectPreview] painter attribute

1.2.4 Groups

Property group attributes used for grouping properties.

> **Info**
>
> Each property can have multiple different group attributes.

Demo

Let's create three field.

```
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    public int health;
    public int minHealth;
    public int maxHealth;
}
```

Default fields view

Let's add `[Group]` attribute to these three fields and name it *Health Settings* `[Group("Health Settings")]`.

> **Note**
>
> Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.
>
> ```
> using ApexInspector;
> ```

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    [Group("Health Settings")]
    public int health;

    [Group("Health Settings")]
    public int minHealth;

    [Group("Health Settings")]
    public int maxHealth;
}
```

View after using [Group] attributes

You can combine differents group attributes. Let's add `[Foldout]` attribute to `minHealth` and `maxHealth` fields and name it *Limits*.

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    [Group("Health Settings")]
    public int health;

    [Group("Health Settings")]
    [Foldout("Limits")]
    public int minHealth;

    [Group("Health Settings")]
    [Foldout("Limits")]
    public int maxHealth;
}
```

View after combining [Group] and [Foldout] attributes

Let's set limits for `minHealth` and `maxHealth` fields by using `[MinValue]` and `[MaxValue]` validator attributes.

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    [Group("Health Settings")]
    [MinValue("minHealth")]
    [MaxValue("maxHealth")]
    public int health;

    [Group("Health Settings")]
    [Foldout("Limits")]
    [MinValue(0)]
    [MaxValue("maxHealth")]
    public int minHealth;

    [Group("Health Settings")]
    [Foldout("Limits")]
    [MinValue("minHealth")]
    [MaxValue(100)]
    public int maxHealth;
}
```

View after add [MinValue] and [MaxValue] validators

1.3 Global Settings

1.3.1 Settings

For edit global apex settings open **Project Settings** window:

```
Edit/Project Settings.../Apex Inspector
```

Apex Inspector settings

Base Settings

Property	Description	Type
Apex Enabled	Enable/disable apex extension in project.	`bool`
Property Space	Addition space to the properties.	`float` `Ranged: [0-∞]`
Highlight Expandable Property	Select which expandable properties needed to highlight.	`enum`
Align Expandable Property	Set true to algin all expandable properties with the others.	`bool`

Light Theme Colors

Light theme colors settings active while enabled **Ligth** editor theme.

Property	Description	Type
Group Header Color	Group attribute header color.	`color`
Property Header Color	Header color of the `Array`, `DropdownReference`	`color`
Group Border Color	Color of the group borders	`color`
Property Border Color	Border colors of the `Array`, `DropdownReference`	`color`

Dark Theme Colors

Dark theme colors settings active while enabled **Dark** editor theme.

Property	Description	Type
Group Header Color	Group attribute header color.	`color`
Property Header Color	Header color of the `Array`, `DropdownReference`	`color`
Group Border Color	Color of the group borders	`color`
Property Border Color	Border colors of the `Array`, `DropdownReference`	`color`

Reset Settings

If you want to reset all settings on default values, use *Gear* button, in the upper-right corner of the window.

Live Demo

2. Attributes

2.1 View

2.1.1 Array

`[Array]` attribute used for drawing flexible arrays.

SUPPORT TYPES

Type	Description	Example
`T[]`	Default C# array	`string[]`
`List<T>`	Collection generic List	`List<string>`

> **Info**
>
> Where T is type of array elements.

PARAMETERS

Parameter Name	Description	Arguments
`ElementLabel`	Custom element label display format.	`{index}` `{niceIndex}`
`CountLabel`	Custom element count label display format.	`{count}`

EXAMPLES

```
[Array]
public int[] values;

[Array(ElementLabel = "User Name: {niceIndex}")]
public List<string> names;

[Array(ElementLabel = null, CountLabel = "{count} Points")]
public Transform[] point;
```

> **Note**
>
> Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.
>
> ```
> using ApexInspector;
> ```

DEMO

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    [Array(ElementLabel = "Material: {niceIndex}", CountLabel = "{count} Items")]
    public Material[] materials;
}
```

Live demo

2.1.2 Dropdown Reference

`[DropdownReference]` attribute allows you to select one of the inheritors of the parent type.

SUPPORT TYPES

✓✓ All references.

PARAMETERS

✕ No parameters

EXAMPLES

```
[SerializeReference]
[DropdownReference]
public Bullet bullet;
```

> **Note**
>
> 1. Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.
>
>    ```
>    using ApexInspector;
>    ```
>
> 2. Make sure that you have added `[SerializeReference]` attribute to the referece field. Otherwise, it will not be displayed in the inspector.

DEMO

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    [SerializeReference]
    [DropdownReference]
    public Animal animal;
}
```

Animal class and inheritors

```
public abstract class Animal
{
    public abstract void Move(Vector3 direction);
}


public class Leon : Animal
{
    public float valueFloat;

    public override void Move(Vector3 direction)
    {
        // TODO
    }
}


public class Leopard : Animal
{
    public int valueInt;
    public string valueString;

    public override void Move(Vector3 direction)
    {
        // TODO
    }
}


[ReferenceContent("Custom Tiger Label")]
public class Tiger : Animal
{
    public float valueVector3;
    public string valueString;
    public bool valueBool;

    public override void Move(Vector3 direction)
    {
        // TODO
    }
}
```

Tip

You can change the reference display type. Since this is done in the example for `Tiger` class. For that you can use `[ReferenceContent]`.

Live demo

2.1.3 Reorderable List

`[ReorderableList]` attribute used for drawing reorderable arrays/lists.

SUPPORT TYPES

Type	Description	Example
`T[]`	Default C# array	`string[]`
`List<T>`	Collection generic List	`List<string>`

Info

Where T is type of array elements.

PARAMETERS

Parameter Name	Description	Arguments
`ElementLabel`	Custom element name display format.	`{index}` `{niceIndex}`
`Draggable`	Set false to disable element drag function.	None
`DrawClearButton`	Set true to display button to clear all list elements.	None

EXAMPLES

```
[ReorderableList]
public int[] values;

[ReorderableList(ElementLabel = "User Name: {niceIndex}", Draggable = false)]
public List<string> names;

[ReorderableList(ElementLabel = null, ClearButton = true)]
public Transform[] point;
```

Note

Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.

```
using ApexInspector;
```

DEMO

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    [ReorderableList(ElementLabel = null, ClearButton = true)]
    public List<Transform> waypoints;
}
```

Live demo

2.1.4 Asset Selecter

`[AssetSelecter]` attribute automatically showing in dropdown list all available asset type of property type.

SUPPORT TYPES

All asset types.

PARAMETERS

Parameter Name	Description	Default	Arguments
`AssetType`	Search asset type	Type of field	None
`Path`	Search asset path	Assets	None
`SearchOption`	Search asset option	All Directories	None

EXAMPLES

```
[AssetSelecter]
public PhysicsMaterial value;

[AssetSelecter(AssetType = typeof(Texture2D))]
public Object names;

[AssetSelecter(Path = "Assets/My Materials")]
public Material material;
```

Note

Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.

```
using ApexInspector;
```

DEMO

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    [AssetSelecter(AssetType = typeof(Material), Path = "Assets/My Materials")]
    public Object value;
}
```

Live demo

2.1.5 Tag Popup

`[TagPopup]` attribute allows to select tag via convenient dropdown list.

SUPPORT TYPES

- `String`

PARAMETERS

✕ No parameters

EXAMPLES

```
[TagPopup]
public string tag;
```

> **Note**
>
> 1. Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.
>
>    ```
>    using ApexInspector;
>    ```

DEMO

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    [TagPopup]
    public string value;
}
```

Live demo

2.1.6 Scene Selecter

`[SceneSelecter]` allows to select scene via convenient dropdown list.

SUPPORT TYPES

Type	Description
`Integer`	Saving selected scene id
`String`	Saving selected scene name

PARAMETERS

✕ No parameters

EXAMPLES

```
[SceneSelecter]
public int menuSceneId;

[SceneSelecter]
public string menuSceneName;
```

Warning

`SceneSelecter` attribute load scenes from build settings. If build settings don't contain scenes, field will be disabled.

This is done in order to avoid the error that developers may make if they choose a scene that was not added to the settings.

Note

Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.

```
using ApexInspector;
```

DEMO

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    [SceneSelecter]
    public int value;
}
```

Live demo

2.1.7 Toggle Left

`[ToggleLeft]` attribute draw toggle on a left side of the field.

SUPPORT TYPES

- `Boolean`

PARAMETERS

✕ No parameters

EXAMPLES

```
[ToggleLeft]
public bool toggle;
```

> **Note**
>
> 1. Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.
>
>    ```
>    using ApexInspector;
>    ```

DEMO

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    [ToggleLeft]
    public bool toggle;
}
```

Live demo

2.1.8 File Path

`[FilePath]` attribute allow to select corrent path to file via build-in in OS panel.

SUPPORT TYPES

- `String`

PARAMETERS

Parameter Name	Description
`Title`	File panel title.
`Directory`	Start panel directory.
`Extension`	File extension filter.
`RelativePath`	Convert path to project relative. *Only if selected file inside Assets folder.*

EXAMPLES

```
[FilePath]
public string filePath;

[FilePath(Title = "Select Material...")]
public string materialPath;

[FilePath(Title = "Select Texture...", "Assets/Textures", RelativePath = true)]
public string texturePath;

[FilePath(RelativePath = true)]
public string scenePath;
```

Note

1. Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.

```
using ApexInspector;
```

DEMO

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    [FilePath(RelativePath = true)]
    public string scriptPath;
}
```

Live demo

2.1.9 Folder Path

`[FolderPath]` attribute allow to select corrent path to the folder via build-in in OS panel.

SUPPORT TYPES

- `String`

PARAMETERS

Parameter Name	Description
`Title`	Folder panel title.
`Folder`	Start panel folder.
`DefaultName`	Default folder name.
`RelativePath`	Convert path to project relative. *Only if selected folder inside Assets folder.*

EXAMPLES

```
[FolderPath]
public string folderPath;

[FolderPath(Title = "Select Material Folder...")]
public string materialFolder;

[FolderPath(Title = "Select Folder With Texture...", RelativePath = true)]
public string textureFolder;

[FolderPath(RelativePath = true)]
public string sceneFolder;
```

Note

1. Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.

```
using ApexInspector;
```

DEMO

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    [FolderPath(RelativePath = true)]
    public string folderPath;
}
```

Live demo

2.1.10 Selectable Enum

`[SelectableEnum]` attribute display enums in user-friendly menu with search feature.

SUPPORT TYPES

✓✓ All enum types.

PARAMETERS

Parameter Name	Description
`Sort`	Automatically sort enum values.
`Height`	Set max menu height.
`DisableValues`	Array of enum values, which should be disabled.
`HideValues`	Array of enum values, which should be hided.

EXAMPLES

```
[SelectableEnum]
public KeyCode key;

[SelectableEnum(Sort = true)]
public KeyCode key;

[SelectableEnum(DisableValues = new string[1] { "None" })]
public KeyCode key;

[SelectableEnum(Sort = true, HideValues = new string[1] { "None" })]
public KeyCode key;
```

> **Note**
>
> Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.
>
> ```
> using ApexInspector;
> ```

DEMO

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    [SelectableEnum]
    public KeyCode value;
}
```

Live demo

2.2 Validator

2.2.1 Asset Only

`[AssetOnly]` The attribute checks that the asset you want to add to the field is an asset from the project window.

SUPPORT TYPES

✓✓ All asset types.

PARAMETERS

✕ No parameters

EXAMPLES

```
[AssetOnly]
public Object value;
```

> **Note**
>
> Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.
>
> ```
> using ApexInspector;
> ```

DEMO

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    [AssetOnly]
    public GameObject someObject;
}
```

Live demo

2.2.2 Max Value

`[MaxValue]` The attribute checks that the number field is less than/equal to specific value.

SUPPORT TYPES

Type
`Integer`
`Float`
`Double`

PARAMETERS

Parameter Name	Description
`Value`	Max possible value for number field.
`PropertyName`	Max possible value represented in property for number field.
`Tolerance`	TODO!

EXAMPLES

```
[MaxValue("maxHealth")]
public int health;

[MaxValue(100)]
public int maxHealth;

[MaxValue(10)]
public float runSpeed;

[MaxValue("runSpeed", 1)]
public float walkSpeed;

[MaxValue(10)]
public double someValue;

[MaxValue("someValue", 0.9f)]
public double someValue2;
```

> **Note**
>
> Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.
>
> ```
> using ApexInspector;
> ```

DEMO

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    [MaxValue(100)]
    public int speed;
}
```

Live demo

2.2.3 Min Value

`[MinValue]` The attribute checks that the number field is greather than/equal to specific value.

SUPPORT TYPES

Type
`Integer`
`Float`
`Double`

PARAMETERS

Parameter Name	Description
`Value`	Min possible value for number field.
`PropertyName`	Min possible value represented in property for number field.
`Tolerance`	TODO!

EXAMPLES

```
[MinValue("minHealth")]
public int health;

[MinValue(0)]
public int minHealth;

[MinValue("walkSpeed", 1)]
public float runSpeed;

[MinValue(1)]
public float walkSpeed;

[MinValue(10)]
public double someValue;

[MinValue("someValue", 0.1f)]
public double someValue2;
```

> **Note**
>
> Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.
>
> ```
> using ApexInspector;
> ```

DEMO

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    [MinValue(1)]
    public int speed;
}
```

Live demo

2.2.4 Not Null

`[NotNull]` The attribute checks that the Object field is not null.

SUPPORT TYPES

Type
`Object`

PARAMETERS

Parameter Name	Description	Arguments
`Format`	Custom message format	`{name}`
`Size`	Box message size.	None

EXAMPLES

```
[NotNull]
public GameObject player;

[NotNull(Format = "{name} is required!")]
public GameObject player;

[NotNull(Size = MessageBoxSize.Inline)]
public GameObject player;
```

Note

Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.

```
using ApexInspector;
```

DEMO

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    [NotNull(Format = "{name} is cannot be empty!", Size = MessageBoxSize.Small)]
    public GameObject player;
}
```

Live demo

2.2.5 Scene Object Only

`[SceneObjectOnly]` The attribute checks that the Object you want to add to the field is in scene.

SUPPORT TYPES

✓✓ All object types.

PARAMETERS

✕ No parameters

EXAMPLES

```
[SceneObjectOnly]
public GameObject value;
```

> **Note**
>
> Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.
>
> ```
> using ApexInspector;
> ```

DEMO

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    [SceneObjectOnly]
    public GameObject someObject;
}
```

Live demo

2.3 Painter

2.3.1 Object Preview

`[ObjectPreview]` attribute used for drawing Object preview window.

SUPPORT TYPES

All Object types.

PARAMETERS

Name	Description
`Height`	Height of the preview window.
`Expandable`	Set true to hided in expandable foldout.

EXAMPLES

```
[ObjectPreview]
public Object values;

[ObjectPreview(Height = 120)]
public GameObject weapon;

[ObjectPreview(Height = 200, Expandable = true)]
public GameObject sword;
```

Note

Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.

```
using ApexInspector;
```

DEMO

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    [ObjectPreview(Expandable = true)]
    public GameObject someObject;
}
```

Live demo

2.3.2 Message

`[Message]` attribute used for drawing some text.

SUPPORT TYPES

Type
`String`

PARAMETERS

Name	Description
`Text`	Text of the message.
`Message Type`	Type of the message.
The color changes depending on the type.	

EXAMPLES

```
[Message("Some text here...")]
public string value1;

[Message("Some text here...", MessageType.Warning)]
public GameObject value2;
```

Note

Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.

```
using ApexInspector;
```

DEMO

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    [Message("Hello!\nThis is Apex Message attribute!", MessageType.Warning)]
    public GameObject value;
}
```

Static demo

2.3.3 Property Space

`[PropertySpace]` attribute add space after property.

SUPPORT TYPES

✓✓ Any types.

PARAMETERS

Name	Description
`Space`	Space after field.

EXAMPLES

```
    [PropertySpace(10)]
    public float floatValue;

    [PropertySpace(5)]
    public int intValue;
```

> **Note**
>
> Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.
>
> ```
> using ApexInspector;
> ```

DEMO

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    [PropertySpace(10)]
    public float floatValue;
    public int intValue;
}
```

Static demo

2.3.4 Label Width

`[LabelWidth]` attribute allow to change width between label and property.

SUPPORT TYPES

Any types.

PARAMETERS

Name	Description
`Widith`	Width between label and property.

EXAMPLES

```
    [LabelWidth(150)]
    public float floatValue;

    [Indent(250)]
    public int intValue;
```

> **Note**
>
> Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.
>
> ```
> using ApexInspector;
> ```

DEMO

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    [LabelWidth(300)]
    public float customWidth;

    public float defaultWidth;
}
```

Static demo

2.3.5 Prefix

`[Prefix]` attribute allow to add prefix text to property.

SUPPORT TYPES

Any types.

PARAMETERS

Name	Description
`Text`	Prefix text.
`Before Property`	Set true to draw prefix before property field.

EXAMPLES

```
    [Prefix("Some text")]
    public float floatValue;

    [Prefix("Some text", true)]
    public int intValue;
```

> **Note**
>
> Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.
>
> ```
> using ApexInspector;
> ```

DEMO

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    [Prefix("Some text")]
    public float floatValue;

    [Prefix("[Some text]", true)]
    public int intValue;
}
```

Static demo

2.3.6 Suffix

`[Suffix]` attribute allow to add suffix text to property.

SUPPORT TYPES

✓✓ Any types.

PARAMETERS

Name	Description
`Text`	Suffix text.
`Muted`	Muted text.

EXAMPLES

```
    [Suffix("Some text")]
    public float floatValue;

    [Suffix("Some text", true)]
    public int intValue;
```

> **Note**
>
> Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.
>
> ```
> using ApexInspector;
> ```

DEMO

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    [Suffix("m/s")]
    public float runSpeed;

    [Suffix("m/s", true)]
    public int walkSpeed;
}
```

Static demo

2.3.7 Indent

`[Indent]` attribute allow to change indent level of property.

SUPPORT TYPES

✓✓ Any types.

PARAMETERS

Name	Description
`Level`	Indent level of the field.
`Following`	Add current indent level for all following properties. *If some following property has `[Indent]` attribute, Following will be ignored for it.*

EXAMPLES

```
    [Indent(1)]
    public float floatValue;

    [Indent(2)]
    public int intValue;
```

> **Note**
>
> Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.
>
> ```
> using ApexInspector;
> ```

DEMO

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    public float value1;

    [Indent(1)]
    public int value2;

    [Indent(2)]
    public int value3;
}
```

Static demo

2.4 Group

2.4.1 Group

`[Group]` attribute used for layout properties in group.

SUPPORT TYPES

Any types.

PARAMETERS

Parameter Name	Description
`Title`	Title of group

EXAMPLES

```
[Group("Some Title")]
public float value1;

[Group("Some Title")]
public float value2;
```

> **Note**
>
> Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.
>
> ```
> using ApexInspector;
> ```

DEMO

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    [Group("Values")]
    public int value1;

    [Group("Values")]
    public float value2;

    [Group("Values")]
    public string value3;
}
```

Static demo

2.4.2 Foldout

`[Foldout]` attribute used for layout properties in expandable foldout.

SUPPORT TYPES

✓✓ Any types.

PARAMETERS

Parameter Name	Description
`Title`	Title of foldout

EXAMPLES

```
public float value1;

[Foldout("Some Title")]
public float value2;

[Foldout("Some Title")]
public float value3;
```

> **Note**
>
> Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.
>
> ```
> using ApexInspector;
> ```

DEMO

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    public float value1;

    [Foldout("Some Title")]
    public float value2;

    [Foldout("Some Title")]
    public float value3;
}
```

Live demo

2.4.3 Tab

`[Tab]` attribute used for layout properties in specific tabs.

SUPPORT TYPES

✓✓ Any types.

PARAMETERS

Parameter Name	Description
`Name`	Name target tab group.
`Title`	Title of tab

EXAMPLES

```
[Tab("Tab Group 1", "Tab 1")]
public float value1;

[Tab("Tab Group 1", "Tab 1")]
public float value2;

[Tab("Tab Group 1", "Tab 2")]
public float value3;

[Tab("Tab Group 1", "Tab 2")]
public float value4;
```

> **Note**
>
> Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.
>
> ```
> using ApexInspector;
> ```

DEMO

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    [Tab("Tab Group 1", "Tab 1")]
    public float value1;

    [Tab("Tab Group 1", "Tab 1")]
    public float value2;

    [Tab("Tab Group 1", "Tab 2")]
    public float value3;

    [Tab("Tab Group 1", "Tab 2")]
    public float value4;
}
```

Live demo

2.4.4 Button Horiaontal Group

`[ButtonHorizontalGroup]` attribute used for layout button in horizontal group.

SUPPORT TYPES

✓✓ Any types.

PARAMETERS

Parameter Name	Description
`Name`	Name of group

EXAMPLES

```
[Button]
[ButtonHorizontalGroup("Functions")]
public void FirstFunction()
{
    // FirstFunction code...
}

[Button]
[ButtonHorizontalGroup("Functions")]
public void SecondFunction()
{
    // SecondFunction code...
}
```

Note

Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.

```
using ApexInspector;
```

DEMO

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    public float value;

    [Button]
    [ButtonHorizontalGroup("Functions")]
    public void FirstFunction()
    {
        Debug.Log("Called First Function!");
    }

    [Button]
    [ButtonHorizontalGroup("Functions")]
    public void SecondFunction()
    {
        Debug.Log("Called Second Function!");
    }
}
```

Live demo

2.5 Conditional

2.5.1 Active If

`[ActiveIf]` attribute allow disable/enable property by specific condition.

SUPPORT TYPES

✓✓ Any types.

PARAMETERS

Option 1

Name	Description
`Property Name`	Boolean property name.

Option 2

Name	Description
`Property Name`	Boolean property name.
`Condition`	Set true to make this property active, while `Property Name` is true. Set false to make this property active, while `Property Name` is false.

Option 3

Name	Description
`First Property`	Numeric property name.
`Condition`	Set specific condition: `>` `<` `<=` `>=`
`Second Property`	Numeric property name.

EXAMPLES

```
/* --- Option 1 --- */
public bool toggle;

[ActiveIf("toggle")]
public int intValue;
/* ---------------- */


/* --- Option 2 --- */
public bool toggle2;

[ActiveIf("toggle2", true)]
public float someValue;
/* ---------------- */


/* --- Option 3 --- */
public float value1;
public float value2;

[ActiveIf("value1", ">", "value2")]
public string someText;
/* ---------------- */
```

Note

Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.

```
using ApexInspector;
```

DEMO

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    public bool toggle;

    [VisibleIf("toggle")]
    public float value;
}
```

Live demo

2.5.2 Visible If

`[VisibleIf]` attribute allow show/hide property by specific condition.

SUPPORT TYPES

✓✓ Any types.

PARAMETERS

Option 1

Name	Description
`Property Name`	Boolean property name.

Option 2

Name	Description
`Property Name`	Boolean property name.
`Condition`	Set true to make this property active, while `Property Name` is true. Set false to make this property active, while `Property Name` is false.

Option 3

Name	Description
`First Property`	Numeric property name.
`Condition`	Set specific condition: `>` `<` `<=` `>=`
`Second Property`	Numeric property name.

EXAMPLES

```
/* --- Option 1 --- */
public bool toggle;

[VisibleIf("toggle")]
public int intValue;
/* ---------------- */


/* --- Option 2 --- */
public bool toggle2;

[VisibleIf("toggle2", true)]
public float someValue;
/* ---------------- */


/* --- Option 3 --- */
public float value1;
public float value2;

[VisibleIf("value1", ">", "value2")]
public string someText;
/* ---------------- */
```

Note

Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.

```
using ApexInspector;
```

DEMO

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    public bool toggle;

    [VisibleIf("toggle")]
    public float value;
}
```

Live demo

2.6 Misc

2.6.1 Label

`[Label]` attribute used changing deafult label of property.

SUPPORT TYPES

✓✓ Any fields

PARAMETERS

Parameter Name	Description
`Name`	New name of property

EXAMPLES

```
[Label("Custom Label")]
public int value;
```

> **Note**
>
> Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.
>
> ```
> using ApexInspector;
> ```

DEMO

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    [Label("Custom Label")]
    public int value;
}
```

Static demo

2.6.2 Hide Label

`[HideLabel]` attribute used hiding label of property.

SUPPORT TYPES

✓✓ Any fields

PARAMETERS

✕ No parameters

EXAMPLES

```
[HideLabel]
public int value;
```

> **Note**
>
> Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.
>
> ```
> using ApexInspector;
> ```

DEMO

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    [HideLabel]
    public int value;
}
```

Static demo

2.6.3 ReadOnly

`[ReadOnly]` attribute used mark property as readonly. Property cannot be edited.

SUPPORT TYPES

✓✓ Any fields

PARAMETERS

✕ No parameters

EXAMPLES

```
[ReadOnly]
public int value;
```

> **Note**
>
> Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.
>
> ```
> using ApexInspector;
> ```

DEMO

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    [ReadOnly]
    public int value;
}
```

Live demo

2.6.4 Hide Script Field

`[HideScriptField]` attribute used for hiding default script reference field of component.

SUPPORT TYPES

✓✓ Any components

PARAMETERS

✕ No parameters

EXAMPLES

```
[HideScriptField]
public class ExampleComponent : MonoBehaviour
{
    // Script content...
}
```

> **Note**
>
> Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.
>
> ```
> using ApexInspector;
> ```

DEMO

```
using ApexInspector;
using UnityEngine;

[HideScriptField]
public class ExampleComponent : MonoBehaviour
{
    public int value;
}
```

Static demo

2.6.5 Reference Content

`[ReferenceContent]` attribute used for changing content label of reference which used in `[DropdownReference]`.

SUPPORT TYPES

✓✓ Any references

PARAMETERS

Parameter Name	Description
`Name`	Name of reference.
`Tooltip`	Tooltip for reference.

EXAMPLES

```
public abstract class Animal
{
    public abstract void Move(Vector3 direction);
}


public class Leon : Animal
{
    public float valueFloat;

    public override void Move(Vector3 direction)
    {
        // TODO
    }
}


public class Leopard : Animal
{
    public int valueInt;
    public string valueString;

    public override void Move(Vector3 direction)
    {
        // TODO
    }
}


[ReferenceContent("Custom Tiger Label")]
public class Tiger : Animal
{
    public float valueVector3;
    public string valueString;
    public bool valueBool;

    public override void Move(Vector3 direction)
    {
        // TODO
    }
}
```

Note

Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.

```
using ApexInspector;
```

DEMO

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    [SerializeReference]
    [DropdownReference]
    public Animal animal;
}

public abstract class Animal
{
    public abstract void Move(Vector3 direction);
}


public class Leon : Animal
{
    public float valueFloat;

    public override void Move(Vector3 direction)
    {
        // TODO
    }
}


public class Leopard : Animal
{
    public int valueInt;
    public string valueString;

    public override void Move(Vector3 direction)
    {
        // TODO
    }
}


[ReferenceContent("Custom Tiger Label")]
public class Tiger : Animal
{
    public float valueVector3;
    public string valueString;
    public bool valueBool;

    public override void Move(Vector3 direction)
    {
        // TODO
    }
}
```

Live demo

2.7 Button

2.7.1 Button

`[Button]` attribute used adding button to inspector.

SUPPORT TYPES

✓✓ Any methods

PARAMETERS

Parameter Name	Description
`Label`	Custom name for button. *Use the `@` prefix to indicate, that a texture will be used instead of the name.* *Arguments:* `@{Default Unity Icon Name}`, `@{Path to texture}` *Example:* `@_Popup`, `@Assets/...`
`Height`	Custom button height.
`Style`	Custom style for button.

EXAMPLES

```
[Button]
public void Function()
{
    //...
}
```

Note

Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.

```
using ApexInspector;
```

DEMO

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    public float value;

    [Button]
    public void FirstFunction()
    {
        Debug.Log("Called First Function!");
    }
}
```

Live demo

2.7.2 Inline Button

`[Button]` attribute used adding button to field.

SUPPORT TYPES

✓✓ Any fields

PARAMETERS

Parameter Name	Description
`Name`	Name of method.
`Label`	Custom name for button. *Use the `@` prefix to indicate, that a texture will be used instead of the name.* *Arguments:* `@{Default Unity Icon Name}`, `@{Path to texture}` *Example:* `@_Popup`, `@Assets/...`
`Width`	Custom button width.
`Style`	Custom style for button.

EXAMPLES

```
[InlineButton("Function")]
public float value;

public void Function()
{
    //...
}
```

Note

Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.

```
using ApexInspector;
```

DEMO

```
using ApexInspector;
using UnityEngine;

#if UNITY_EDITOR
using UnityEditor;
#endif

public class ExampleComponent : MonoBehaviour
{
    [InlineButton("Function", Label = "@_Popup", Style = "IconButton")]
    public string value;

#if UNITY_EDITOR
    public void Function()
    {
        GenericMenu menu = new GenericMenu();
        menu.AddItem(new GUIContent("Text 1"), false, () => value = "Text 1");
        menu.AddItem(new GUIContent("Text 2"), false, () => value = "Text 2");
        menu.AddItem(new GUIContent("Text 3"), false, () => value = "Text 3");
        menu.ShowAsContext();
    }
#endif
}
```

Live demo

2.7.3 Bottom Button

`[BottomButton]` attribute used adding button to field placed in bottom.

SUPPORT TYPES

✓✓ Any fields

PARAMETERS

Parameter Name	Description
`Name`	Name of method.
`Label`	Custom name for button. *Use the `@` prefix to indicate, that a texture will be used instead of the name.* *Arguments:* `@{Default Unity Icon Name}`, `@{Path to texture}` *Example:* `@_Popup`, `@Assets/...`
`Group`	Group name of buttons.
`Height`	Custom button height.
`Style`	Custom style for button.

EXAMPLES

```
[ButtomButton("Function")]
public float value;

public void Function()
{
    //...
}
```

> **Note**
>
> Make sure that you have added ApexInspector namespace in your script, to get access to all attributes.
>
> ```
> using ApexInspector;
> ```

DEMO

```
using ApexInspector;
using UnityEngine;

public class ExampleComponent : MonoBehaviour
{
    [BottomButton("FirstFunction", Label = "First Function", Group = "Functions Group", Style = "ButtonLeft")]
    [BottomButton("SecondFunction", Label = "Second Function", Group = "Functions Group", Style = "ButtonRight")]
    public float value;

    public void FirstFunction()
    {
        value = 10;
    }

    public void SecondFunction()
    {
        value = 30;
    }
}
```

Live demo

3. API References

3.1 Custom View

3.1.1 View Attirbute

1. Create new class, for example `ExampleAttribute.cs`

```
public class ExampleAttribute
{

}
```

2. Inherit from `ViewAttribute` class.

```
public class ExampleAttribute : ViewAttribute
{

}
```

3. Optionaly add *Required*/*Optional* parameters.

```
public class ExampleAttribute : ViewAttribute
{
    // Required parameter.
    public readonly string text;

    // Constructor with required parameter.
    public ExampleAttribute(string text)
    {
        this.text = text;
    }

    // Optional parameter.
    public float SomeValue { get; set; }
}
```

Example

```
public class ExampleComponent : MonoBehaviour
{
    // Required parameter.
    [Example("Some Text")]
    public float value;

    // Required and optional parameter.
    [Example("Some Text", Value = 10.0f)]
    public float value;
}
```

3.1.2 Property View

1. Create new folder and name it `Editor`.
2. Inside `Editor` folder create new class, `ExampleView.cs`

```
public class ExampleView
{

}
```

3. Inherit from `PropertyView` class.

```
public class ExampleView : PropertyView
{

}
```

4. Implement abstract method `OnGUI`.

```
public class ExampleView : PropertyView
{
    public override void OnGUI(Rect position, SerializedProperty property, GUIContent label)
    {
        // Property GUI here...
    }
}
```

5. Add `ViewTarget` attribute and set `ExampleAttribute` as target to this view.

```
[ViewTarget(typeof(ExampleAttribute))]
public class ExampleView : PropertyView
{
    public override void OnGUI(Rect position, SerializedProperty property, GUIContent label)
    {
        // Property GUI here...
    }
}
```

3.1.3 Virtual Methods

OnInitialize

DESCRIPTION

Called once when initializing PropertyView.

Very useful for make some initializations of properties and get view attribute.

```
void OnInitialize(SerializedProperty property, ViewAttribute viewAttribute, GUIContent label)
{
    // Some initialization...
}
```

PARAMETERS

Parameter	Description
property	Serialized property with ViewAttribute.
viewAttribute	ViewAttribute of serialized property.
label	Label of serialized property.

EXAMPLE

```
[ViewTarget(typeof(ExampleAttribute))]
public class ExampleView : PropertyView
{
    private ExampleAttribute exampleAttribute;

    // Called once when initializing PropertyView.
    public override void OnInitialize(SerializedProperty property, ViewAttribute viewAttribute, GUIContent label)
    {
        // Getting ExampleAttribute attribute from field.
        exampleAttribute = viewAttribute as ExampleAttribute;

        // Some other initializations here...
    }
}
```

OnGUI

DESCRIPTION

Called for rendering and handling GUI events.

```
void OnGUI(Rect position, SerializedProperty property, GUIContent label)
{
    // Property GUI here...
}
```

PARAMETERS

Parameter	Description
position	Position of the serialized property.
property	Serialized property with ViewAttribute.
label	Label of serialized property.

EXAMPLE

```
[ViewTarget(typeof(ExampleAttribute))]
public class ExampleView : PropertyView
{
    // Called for rendering and handling GUI events.
    public override void OnGUI(Rect position, SerializedProperty property, GUIContent label)
    {
```

```
        GUI.Label(position, "Custom property view");
    }
}
```

GetPropertyHeight

DESCRIPTION

Return height which needed to draw property.

```
float GetPropertyHeight(SerializedProperty property, GUIContent label)
{
    // Return height of property.
}
```

PARAMETERS

Parameter	Description
property	Serialized property with ViewAttribute.
label	Label of serialized property.

EXAMPLE

```
[ViewTarget(typeof(ExampleAttribute))]
public class ExampleView : PropertyView
{
    // Return height which needed to draw property.
    public virtual float GetPropertyHeight(SerializedProperty property, GUIContent label)
    {
        return 20;
    }
}
```

3.2 Custom Validator

3.2.1 Validator Attirbute

1. Create new class, for example `ExampleAttribute.cs`

```
public class ExampleAttribute
{

}
```

2. Inherit from `ValidatorAttribute` class.

```
public class ExampleAttribute : ValidatorAttribute
{

}
```

3. Optionaly add *Required*/*Optional* parameters.

```
public class ExampleAttribute : ValidatorAttribute
{
    // Required parameter.
    public readonly string text;

    // Constructor with required parameter.
    public ExampleAttribute(string text)
    {
        this.text = text;
    }

    // Optional parameter.
    public float SomeValue { get; set; }
}
```

Example

```
public class ExampleComponent : MonoBehaviour
{
    // Required parameter.
    [Example("Some Text")]
    public float value;

    // Required and optional parameter.
    [Example("Some Text", Value = 10.0f)]
    public float value;
}
```

3.2.2 Property Validator

1. Create new folder and name it `Editor`.
2. Inside `Editor` folder create new class, `ExampleValidator.cs`

```
public class ExampleValidator
{

}
```

3. Inherit from `PropertyValidator` class.

```
public class ExampleValidator : PropertyValidator
{

}
```

4. Implement abstract method `Validate`.

```
public class ExampleValidator : PropertyValidator
{
    public override void Validate(SerializedProperty property)
    {
        // Property validation here...
    }
}
```

5. Add `ValidatorTarget` attribute and set `ExampleAttribute` as target to this view.

```
[ValidatorTarget(typeof(ExampleAttribute))]
public class ExampleValidator : PropertyValidator
{
    public override void Validate(SerializedProperty property)
    {
        // Property validation here...
    }
}
```

3.2.3 Virtual Methods

OnInitialize

DESCRIPTION

Called once when initializing PropertyValidator.

Very useful for make some initializations of properties and get view attribute.

```
void OnInitialize(SerializedProperty property, ValidatorAttribute validatorAttribute, GUIContent label)
{
    // Some initialization...
}
```

PARAMETERS

Parameter	Description
property	Serialized property with ValidatorAttribute.
validatorAttribute	ValidatorAttribute of serialized property.
label	Label of serialized property.

EXAMPLE

```
[ValidatorTarget(typeof(ExampleAttribute))]
public class ExampleValidator : PropertyValidator
{
    private ExampleAttribute exampleAttribute;

    // Called once when initializing PropertyValidator.
    public override void OnInitialize(SerializedProperty property, ValidatorAttribute validatorAttribute, GUIContent label)
    {
        // Getting ExampleAttribute attribute from field.
        exampleAttribute = validatorAttribute as ExampleAttribute;

        // Some other initializations here...
    }
}
```

Validate

DESCRIPTION

Called before drawing property.

```
void Validate(SerializedProperty property)
{
    // Property validation here...
}
```

PARAMETERS

Parameter	Description
property	Serialized property with ValidatorAttribute.

EXAMPLE

```
[ValidatorTarget(typeof(ExampleAttribute))]
public class ExampleValidator : PropertyValidator
{
    // Called before drawing property.
    public override void Validate(SerializedProperty property)
    {
        if(property.floatValue < 0)
        {
            property.floatValue = 0;
        }
```

```
    }
}
```

ModifyPropertyPosition

DESCRIPTION

Called before OnValidatorGUI() for modify property position.

```
void ModifyPropertyPosition(Rect originalPosition, ref Rect modifiedPosition)
{
    // Modify property position here.
}
```

PARAMETERS

Parameter	Description
originalPosition	Stored original position of the property.
modifiedPosition	Current position which has been modified by other validators, if this property contains other validator attributes.

EXAMPLE

```
[ValidatorTarget(typeof(ExampleAttribute))]
public class ExampleValidator : PropertyValidator
{
    // Called before OnValidatorGUI() for modify property position.
    public override void ModifyPropertyPosition(Rect originalPosition, ref Rect modifiedPosition)
    {
        const float offset = 5.0f;
        modifiedPosition.x += offset;
        modifiedPosition.width -= offset;
    }
}
```

BeforePropertyGUI

DESCRIPTION

Called before drawing property and before OnValidatorGUI().

```
void BeforePropertyGUI(Rect position, SerializedProperty property, GUIContent label)
{
    // Before property GUI calls here...
}
```

PARAMETERS

Parameter	Description
position	Position of the serialized property.
property	Serialized property with ValidatorAttribute.
label	Label of serialized property.

EXAMPLE

```
[ValidatorTarget(typeof(ExampleAttribute))]
public class ExampleValidator : PropertyValidator
{
    // Called before drawing property and before OnValidatorGUI().
    public override void BeforePropertyGUI(Rect position, SerializedProperty property, GUIContent label)
    {
        EditorGUI.BeginDisabledGroup(true);
    }
}
```

OnValidatorGUI

DESCRIPTION

Called for rendering and handling GUI events.

```
void OnValidatorGUI(Rect originalPosition, Rect validatorPosition, SerializedProperty property, GUIContent label)
{
    // Property GUI here...
}
```

PARAMETERS

Parameter	Description
originalPosition	Stored original position of the property.
validatorPosition	Rectangle on the screen to use for the validator GUI.
property	Serialized property with ValidatorAttribute.
label	Label of serialized property.

EXAMPLE

```
[ValidatorTarget(typeof(ExampleAttribute))]
public class ExampleValidator : PropertyValidator
{
    // Called for rendering and handling GUI events.
    public override void OnValidatorGUI(Rect originalPosition, Rect validatorPosition, SerializedProperty property, GUIContent label)
    {
        GUI.Label(validatorPosition, "Custom validator message");
    }
}
```

AfterPropertyGUI

DESCRIPTION

Called after drawing property and after OnValidatorGUI().

```
void AfterPropertyGUI(Rect position, SerializedProperty property, GUIContent label)
{
    // After property GUI calls here...
}
```

PARAMETERS

Parameter	Description
position	Position of the serialized property.
property	Serialized property with ValidatorAttribute.
label	Label of serialized property.

EXAMPLE

```
[ValidatorTarget(typeof(ExampleAttribute))]
public class ExampleValidator : PropertyValidator
{
    // Called after drawing property and after OnValidatorGUI().
    public override void AfterPropertyGUI(Rect position, SerializedProperty property, GUIContent label)
    {
        EditorGUI.EndDisabledGroup();
    }
}
```

GetValidatorHeight

DESCRIPTION

Get the height of the validator, which required to display it.

Calculate only the size of the current validator, not the entire property.

The validator height will be added to the total size of the property.

```
// Return height which needed to draw validator.
float GetPropertyHeight(SerializedProperty property, GUIContent label)
{
    // Return height of validator.
}
```

PARAMETERS

Parameter	Description
property	Serialized property with ValidatorAttribute.
label	Label of serialized property.

EXAMPLE

```
[ValidatorTarget(typeof(ExampleAttribute))]
public class ExampleValidator : PropertyValidator
{
    // Return height which needed to draw validator.
    public virtual float GetPropertyHeight(SerializedProperty property, GUIContent label)
    {
        return 20;
    }
}
```

3.3 Custom Painter

3.3.1 Painter Attirbute

1. Create new class, for example `ExampleAttribute.cs`

```
public class ExampleAttribute
{

}
```

2. Inherit from `PainterAttribute` class.

```
public class ExampleAttribute : PainterAttribute
{

}
```

3. Optionaly add *Required*/*Optional* parameters.

```
public class ExampleAttribute : PainterAttribute
{
    // Required parameter.
    public readonly string text;

    // Constructor with required parameter.
    public ExampleAttribute(string text)
    {
        this.text = text;
    }

    // Optional parameter.
    public float SomeValue { get; set; }
}
```

Example

```
public class ExampleComponent : MonoBehaviour
{
    // Required parameter.
    [Example("Some Text")]
    public float value;

    // Required and optional parameter.
    [Example("Some Text", Value = 10.0f)]
    public float value;
}
```

3.3.2 Property Painter

1. Create new folder and name it `Editor`.
2. Inside `Editor` folder create new class, `ExamplePainter.cs`

```
public class ExamplePainter
{

}
```

3. Inherit from `PropertyPainter` class.

```
public class ExamplePainter : PropertyPainter
{

}
```

4. Implement virtual method `OnPainterGUI`.

```
public class ExamplePainter : PropertyPainter
{
    public override void OnPainterGUI(Rect originalPosition, Rect painterPosition, SerializedProperty property, GUIContent label)
    {
        // Painter GUI here...
    }
}
```

5. Add `PainterTarget` attribute and set `ExampleAttribute` as target to this view.

```
[PainterTarget(typeof(ExampleAttribute))]
public class ExamplePainter : PropertyPainter
{
    public override void OnPainterGUI(Rect originalPosition, Rect painterPosition, SerializedProperty property, GUIContent label)
    {
        // Property GUI here...
    }
}
```

3.3.3 Virtual Methods

OnInitialize

DESCRIPTION

Called once when initializing PropertyPainter.

Very useful for make some initializations of properties and get view attribute.

```
void OnInitialize(SerializedProperty property, PainterAttribute painterAttribute, GUIContent label)
{
    // Some initialization...
}
```

PARAMETERS

Parameter	Description
property	Serialized property with ViewAttribute.
painterAttribute	PainterAttribute of serialized property.
label	Label of serialized property.

EXAMPLE

```
[PainterTarget(typeof(ExampleAttribute))]
public class ExampleView : PropertyPainter
{
    private ExampleAttribute exampleAttribute;

    // Called once when initializing PropertyPainter.
    public override void OnInitialize(SerializedProperty property, PainterAttribute painterAttribute, GUIContent label)
    {
        // Getting ExampleAttribute attribute from field.
        exampleAttribute = painterAttribute as ExampleAttribute;

        // Some other initializations here...
    }
}
```

ModifyPropertyPosition

DESCRIPTION

Called before OnPainterGUI() for modify property position.

```
void ModifyPropertyPosition(Rect originalPosition, ref Rect modifiedPosition)
{
    // Modify property position here.
}
```

PARAMETERS

Parameter	Description
originalPosition	Stored original position of the property.
modifiedPosition	Current position which has been modified by other painters, if this property contains other painter attributes.

EXAMPLE

```
[PainterTarget(typeof(ExampleAttribute))]
public class ExampleView : PropertyPainter
{
    // Called before OnPainterGUI() for modify property position.
    public override void ModifyPropertyPosition(Rect originalPosition, ref Rect modifiedPosition)
    {
        const float offset = 5.0f;
        modifiedPosition.x += offset;
        modifiedPosition.width -= offset;
```

```
    }
}
```

OnPainterGUI

DESCRIPTION

Called for rendering and handling GUI events.

```
void OnPainterGUI(Rect originalPosition, Rect painterPosition, SerializedProperty property, GUIContent label)
{
    // Painter GUI here...
}
```

PARAMETERS

Parameter	Description
originalPosition	Stored original position of the property.
painterPosition	Rectangle on the screen to use for the painter GUI.
property	Serialized property with PainterAttribute.
label	Label of serialized property.

EXAMPLE

```
[PainterTarget(typeof(ExampleAttribute))]
public class ExampleView : PropertyPainter
{
    // Called for rendering and handling GUI events.
    public override void OnPainterGUI(Rect originalPosition, Rect painterPosition, SerializedProperty property, GUIContent label)
    {
        GUI.Label(painterPosition, "Custom property painter");
    }
}
```

GetPainterHeight

DESCRIPTION

Get the height of the painter, which required to display it.

Calculate only the size of the current painter, not the entire property.

The painter height will be added to the total size of the property with other painters.

```
// Return height which needed to draw painter.
float GetPainterHeight(SerializedProperty property, GUIContent label)
{
    // Return height of painter.
}
```

PARAMETERS

Parameter	Description
property	Serialized property with ViewAttribute.
label	Label of serialized property.

EXAMPLE

```
[PainterTarget(typeof(ExampleAttribute))]
public class ExampleView : PropertyPainter
{
    // Return height which needed to draw painter.
    public virtual float GetPainterHeight(SerializedProperty property, GUIContent label)
    {
```

```
        return 20;
    }
}
```

3.4 Interfaces

3.4.1 IPropertyValidatorReceiver

Interface to receive callbacks when initializing Apex attributes.

The callback interface only works with Apex attributes: **View**, **Painter**, **Validator**.

Description

Use this interface to restrict how your attribute works with specific properties.

Implement IPropertyValidatorReceiver interface and implement `IsValidProperty` method.

```
bool IsValidProperty(SerializedProperty property, GUIContent label)
{
    return true/false;
}
```

Parameter	Description
property	Serialized property of current attribute.
label	Label of serialized property.

✓ Return true if this property valid the using with this attribute.

✗ If return false, this attribute will be ignored.

Example

For example you created an attribute for working with arrays and you want to be sure that this attribute will only work with arrays, and for other types it will be ignored.

```
[ViewTarget(typeof(ArrayAttribute))]
public class ArrayView : PropertyView, IPropertyValidatorReceiver
{
    // Array view code..

    public bool IsValidProperty(SerializedProperty property, GUIContent label)
    {
        return property.isArray;
    }
}
```