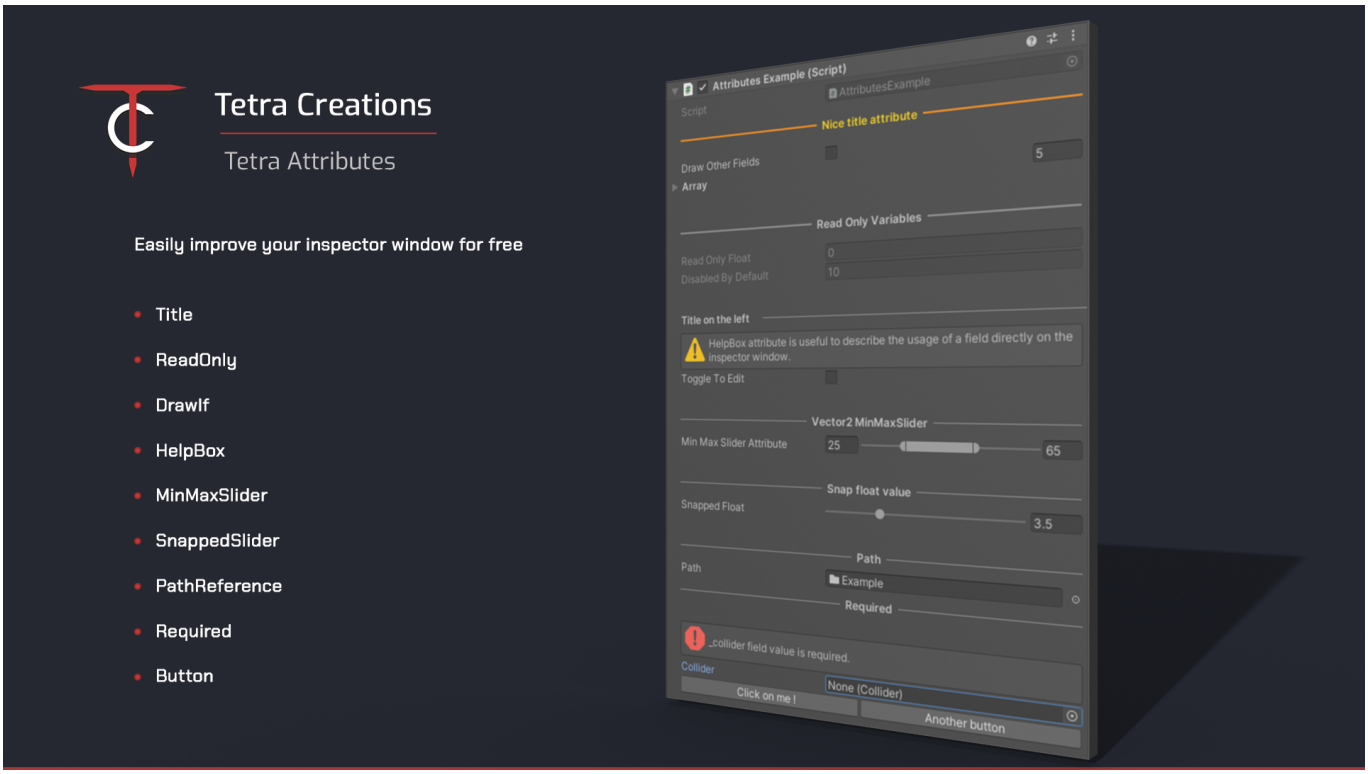
Tetra Attributes 1.0.5 : Documentation



Importing the Asset

If you not familiar on how to install an asset, once you have purchased it, click on **Window → Package Manager** to reveal a window with all your available assets.

Type in the search field **Tetra Attributes** to download and install the last version. Follow the steps and wait till Unity finishes compiling your project.

Introduction

This is a collection of C# attributes for the Unity editor that I use in most of my projects. Some are essential, like ReadOnly, which I've been using for several years. While others like Title are more for keeping the inspector window organised and clear.

Changelog

1.0.5

- DrawIf : Fixed a bug where enum values were compared as flags by default. The Type of the enum need to be passed as the fourth parameter of the DrawIf constructor to know if the enum has the [Flags] attribute, to compare values using bit shifting only if that the case.

1.0.4

- PathReference : Removed default parameters from the constructor so it's properly serialized with its default values. Before that it was display as a read only field. Added public properties instead to change 'Editable', 'AutoUpdate' and 'EnableDebug'.

1.0.3

- Added Tags attribute to display a dropdown list of all available tags.

1.0.2

- PathReference : Path is now cached. Added methods to update the GUID or/and the Path using AssetDatabase.AssetPathToGUID and AssetDatabase.GUIDToAssetPath. Retrieveing the path using the property getter will always check if the path is still valid.
- Added PathUtility static class to verify if a selected path is valid. It also provides methods to automatically import folder that exist on disk in the AssetDatabase, from the 'Assets' folder.

1.0.1

- Added SpritePreview attribute to display the texture below a Sprite field.
- PathReference : Fixed console error "InvalidOperationException: Stack empty". After closing the folder selection dialog window without selecting anything.
- PathReference : Added the possibility to delete the folder reference when pressing the delete key while hovering the field.
- Renamed TitleColor enum to CustomColor because it's now used in SpritePreview.

Important : You will have to replace all Title references using **TitleColor** in you project with **CustomColor**.

Usage

Once imported simply add this line to your file header to use any attributes :

```
using TetraCreations.Attributes;
```

All Property, Decorator drawers and Editor scripts are inside the namespace :

```
TetraCreations.Attributes.Editor
```

An example scene with the AttributesExample script using every attributes is available in :

```
Assets/Tetra Creations/Attributes/Example/Example.unity
```

Table of Contents

Normal Attributes

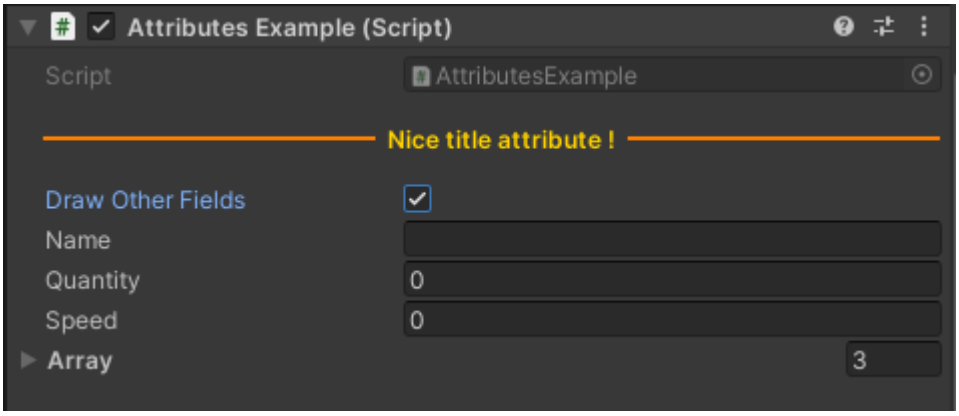
[Tile]

Alternative to **Header** attribute but with a line to separate the title from other fields.

Usage

```
public class AttributesExample : MonoBehaviour
{
      [Title("Nice title attribute !",
      CustomColor.Yellow, CustomColor.Orange, 2f, 20f)]
      public bool DrawOtherFields
}
```

Result



Constructor

```
public TitleAttribute(string title = "",
    CustomColor CustomColor = DefaultCustomColor,
    CustomColor lineColor = DefaultLineColor,
    float lineHeight = DefaultLineHeight,
    float spacing = 14f,
    bool alignTitleLeft = false)
{
    Title = title;
    CustomColor = CustomColor;
    LineColor = lineColor;
    CustomColorString = ColorUtility.ToHtmlStringRGB(CustomColor.GetColor());
    LineColorString = ColorUtility.ToHtmlStringRGB(LineColor.GetColor());
    LineHeight = Mathf.Max(1f, lineHeight);
    Spacing = spacing;
    AlignTitleLeft = alignTitleLeft;
}
```

Constants

```
public const float DefaultLineHeight = 1f;
public const CustomColor DefaultLineColor = CustomColor.LightGray;
public const CustomColor DefaultCustomColor = CustomColor.Bright;
```

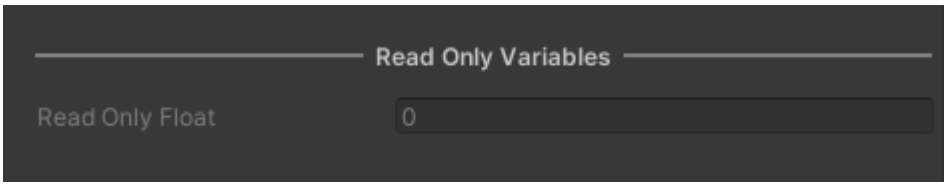
[ReadOnly]

Used to disable modifications to a serialized field.

Usage

```
public class AttributesExample : MonoBehaviour
{
    [ReadOnly]
    public float ReadOnlyFloat;
}
```

Result



Constructor

There are no parameters for this attribute.

[DrawIf]

Draw a property field if the condition is true. (Only for Boolean and Enum)
In the example below, we are hiding the field **Name** until the **DrawOtherFields** field value is set to true.

Usage

```
public class AttributesExample : MonoBehaviour
{
    public bool DrawOtherFields = false;

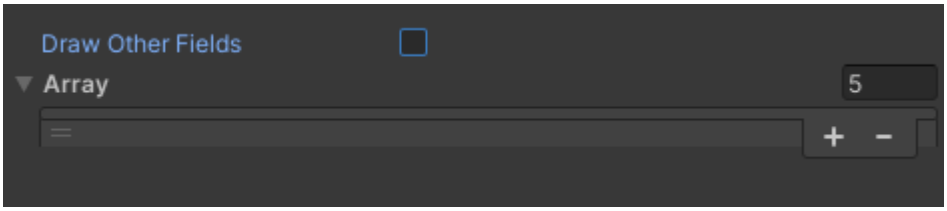
    [DrawIf(nameof(DrawOtherFields), true)]
    public string Name;

    // Draw if using an Enum flags, NorthWest will be only visible if
DirectionsExample.North and DirectionsExample.West are selected
    public DirectionsExample SelectionDirections;
    [DrawIf(nameof(SelectionDirections), DirectionsExample.North |
DirectionsExample.West, type: typeof(DirectionsExample))]
    public string NorthWest;

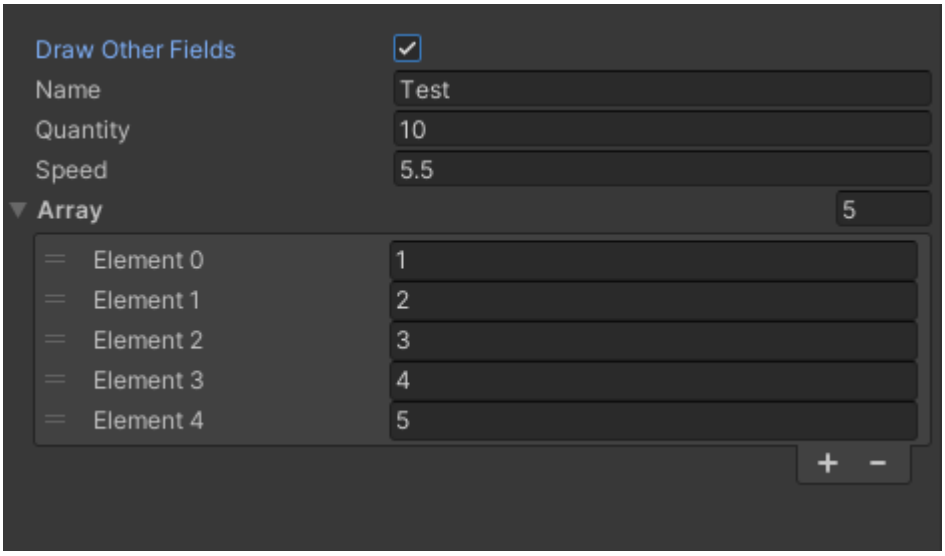
}
```

Result

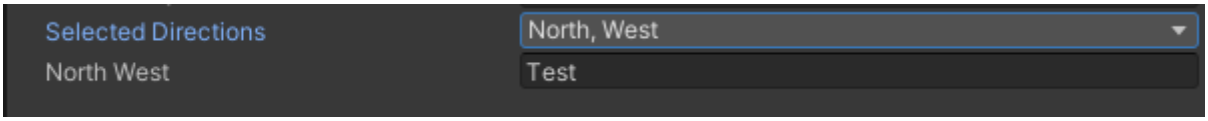
False



True



With Enum Flags



Constructor

```
/// <summary>
/// Only draws the field if the condition is true.<br></br>
/// Supports Boolean and Enum.
/// </summary>
/// <param name="comparedPropertyName">The name of the property that is being
compared (case sensitive).</param>
/// <param name="comparedValue">The value the property is being compared to.
</param>
/// <param name="disablingType">Determine if it will hide the field or make it
read only if the condition is NOT met.
/// <param name="type">The type of the ComparedValue object to determine if it's
an enum with the [Flags] attribute or not.
/// Defaulted to DisablingType.DontDraw.</param>

public DrawIfAttribute(string comparedPropertyName,
    object comparedValue,
    DisablingType disablingType = DisablingType.DontDraw)
{
    ComparedPropertyName = comparedPropertyName;
    ComparedValue = comparedValue;
    DisablingType = disablingType;
    IsEnumWithFlags = HasFlagsAttribute(type);
}
```

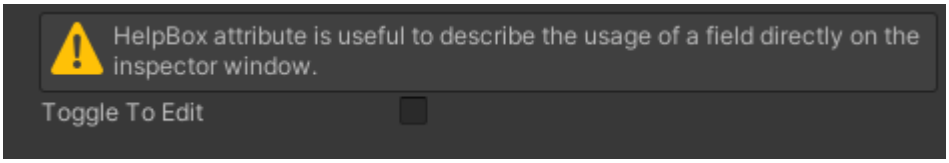
[HelpBox]

Display a help box in the inspector with a message and a type (None, Info, Warning, Error)

Usage

```
public class AttributesExample : MonoBehaviour
{
    [HelpBox("HelpBox attribute is useful to describe the usage of a field
directly on the inspector window.", HelpBoxMessageType.Warning)]
    public bool ToggleToEdit = false;
}
```

Result



Constructor

```
public HelpBoxAttribute(string text,
    HelpBoxMessageType messageType = HelpBoxMessageType.None,
    float minimumHeight = 20,
    int fontSize = 12)
{
    Text = text;
    MessageType = messageType;
    MinimumHeight = minimumHeight;
    FontSize = fontSize;
}
```

[MinMaxSlider]

Show a slider with minimum and maximum values for a Vector2.

Usage

```
public class AttributesExample : MonoBehaviour
{
    [MinMaxSlider(0, 100)]
    public Vector2 MinMaxSliderAttribute;
}
```

Result

Constructor

```
public MinMaxSliderAttribute(float min, float max)
{
    Min = min;
    Max = max;
}
```

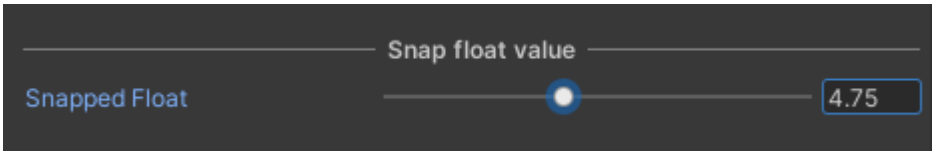
[SnappedSlider]

Draw a slider to increase an integer or a float value by a certain amount (step) and clamped by a minimum and a maximum value. (Only for Integer and Float)

Usage

```
public class AttributesExample : MonoBehaviour
{
    [SnappedSlider(0.25f, 1f, 10f)]
    Public float SnappedFloat;
}
```

Result



Constructors

```
/// <summary>
/// Increase a float value in step<br></br>
/// Value is clamped by min and max parameters
/// </summary>
/// <param name="step">Value to add</param>
/// <param name="min"></param>
/// <param name="max"></param>
public SnappedSliderAttribute(float step, float min, float max)
{
    Step = step;
    Min = min;
    Max = max;
    Precision = MathExtensions.CountFloatDigits(step);
}

/// <summary>
/// Increase an int value in step<br></br>
/// Value is clamped by min and max parameters
/// </summary>
/// <param name="step">Value to add</param>
/// <param name="min"></param>
/// <param name="max"></param>
/// <param name="allowNonStepReach"></param>
public SnappedSliderAttribute(int step, int min, int max, bool allowNonStepReach =
true)
{
    Min = min;
    Max = max;
    Step = step;
    AllowNonStepReach = allowNonStepReach;
    IsInt = true;
}
```

[Required]

Draw an Help Box (Error Type) if a field value is empty or null.

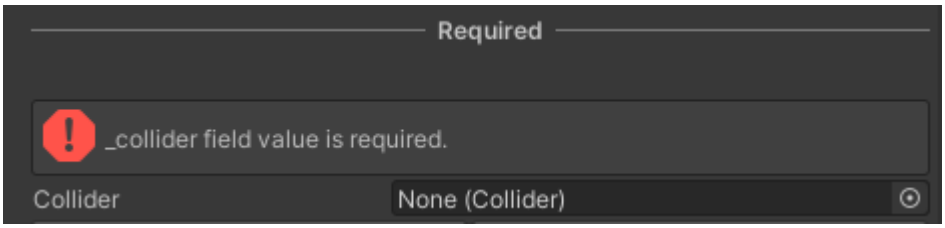
Supported SerializedPropertyType

- String
- ObjectReference
- ExposedReference
- ManagedReference

Usage

```
public class AttributesExample : MonoBehaviour
{
    [Required]
    public Collider Collider;
}
```

Result



Constructor

There are no parameters for this attribute.

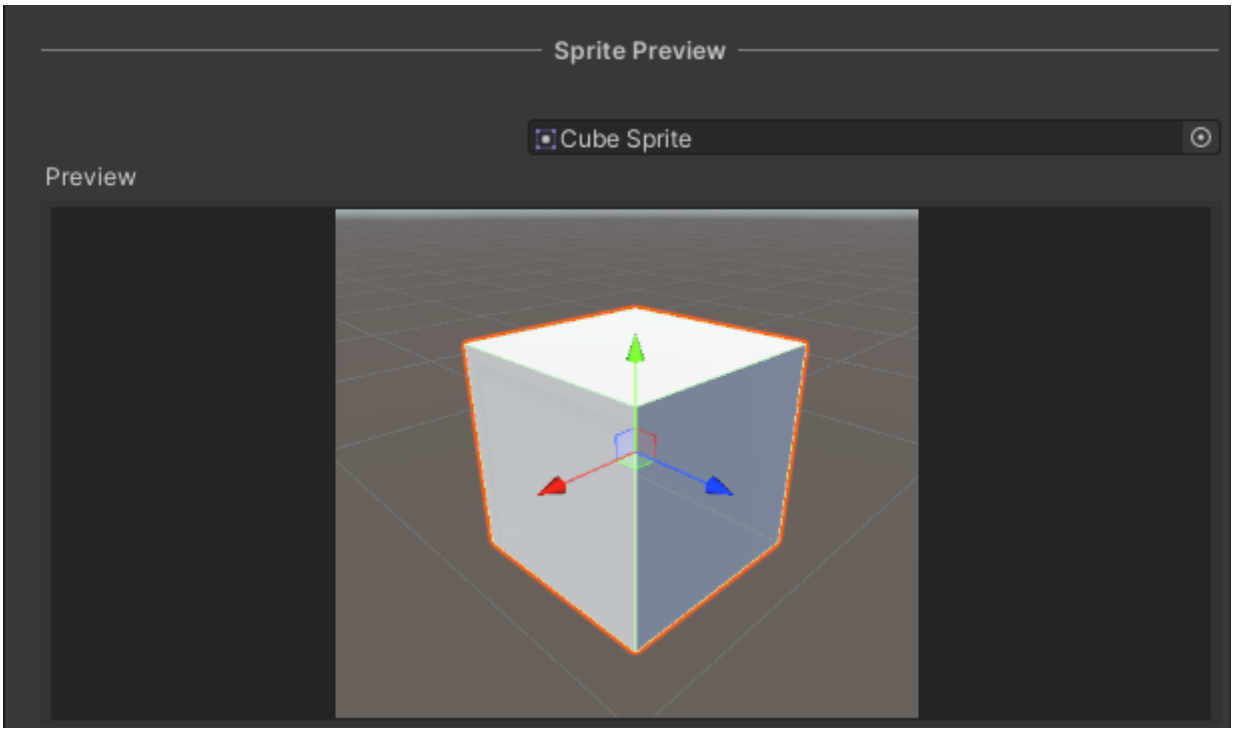
[SpritePreview]

Draw the texture below a sprite field.

Usage

```
public class AttributesExample : MonoBehaviour
{
    [SpritePreview]
    public Sprite Sprite;
}
```

Result



Constructor

```
/// <summary>
/// Dispaly the texture below a sprite field.
/// </summary>
/// <param name="maximumHeight">Maximum height of the preview (With
useAssetPreview set to false)</param>
/// <param name="backgroundColor">The color behind the texture</param>
/// <param name="useAssetPreview">If true it will use AssetPreview.GetAssetPreview
to draw the texture, the maximumHeight doesn't change anyting</param>
public SpritePreviewAttribute(float maximumHeight = 256f, CustomColor
backgroundColor = DefaultBackgroundColor, bool useAssetPreview = false)
{
    UseAssetPreview = useAssetPreview;
    MaximumHeight = maximumHeight;
    BackgroundColor = backgroundColor;
    BackgroundColorString =
ColorUtility.ToHtmlStringRGB(BackgroundColor.ToColor());
}
```

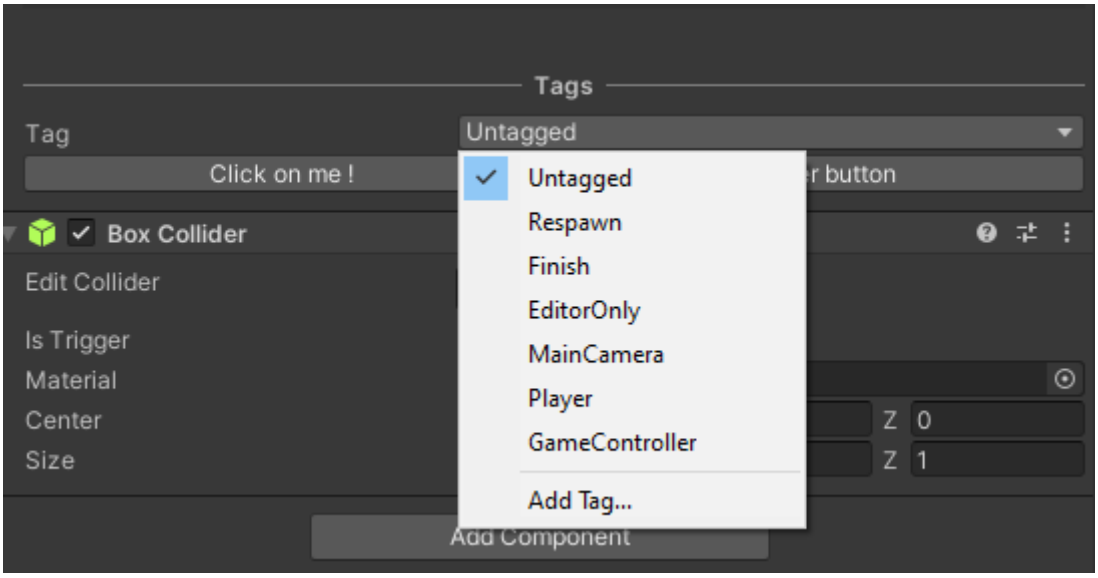
[Tags]

Display a dropdown list of all available tags.

Usage

```
public class AttributesExample : MonoBehaviour
{
    [Tags]
    public string Tag;
}
```

Result



Constructor

There are no parameters for this attribute.

Special Attributes

Theses are not working like usual attributes, PathReference is not even an attribute it's a serializable class.

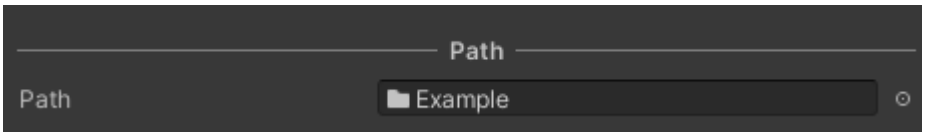
[PathReference]

Allow to store the GUI and the Path of an asset folder.
You can either drag and drop a folder or select it by clicking on the icon on the right.

Usage

```
public class AttributesExample : MonoBehaviour
{
    Public PathReference Path;
}
```

Result



Constructor

```
```cs
/// <summary>
/// Default constructor to display a folder reference by it's GUID or/and path.
/// </summary>
public PathReference()
{
    _editable = true;
    _autoUpdate = true;
    _enableDebug = true;
}
```

Limitations

- You cannot call PathReference.Path at Runtime, because it's using AssetDatabase class.
- You can only use the GUID Property.
- Folder must be inside 'Assets'

```

[Button]

Draw button in the inspector. This works using several classes :

- ButtonAttribute
- Button
- ButtonDrawers
- EditorButtons

Usage

```
public class AttributesExample : MonoBehaviour
{
    [Button(nameof(ButtonCallback), "Click on me !", 100f, row: "first")]
    public void ButtonCallback()
    {
        Debug.Log("You clicked on a button, congrats.");
    }

    [Button(nameof(Test), "Another button", 100f, row:"first")]
    public void Test()
    {
        Debug.Log("This method is incredibly useful.");
    }
}
```

Result

Constructors

```
public ButtonAttribute(string methodName,
    string label = "",
    float width = default,
    int space = default,
    string row = default)
{
    MethodName = methodName;
    Label = label;
    Space = space;
    Row = row;
    HasRow = !string.IsNullOrEmpty(Row);
}

public Button(MethodInfo method, ButtonAttribute buttonAttribute)
{
    ButtonAttribute = buttonAttribute;
    Label = string.IsNullOrEmpty(buttonAttribute.Label) ?
ObjectNames.NicifyVariableName(method.Name) : buttonAttribute.Label;
    Method = method;
}
```

Limitations

By default this wont work inside your custom editor because you need them to inherit from EditorButtons.
