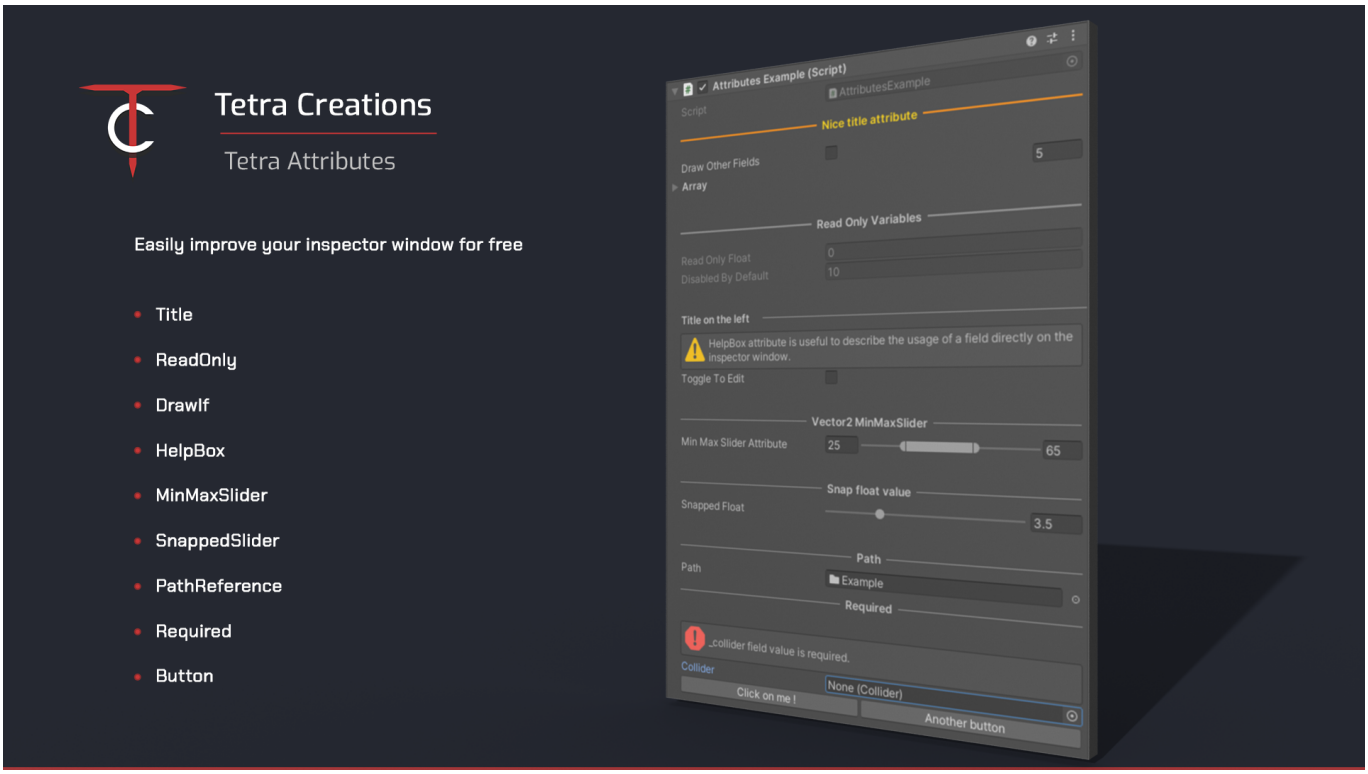
Tetra Attributes : Documentation



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.

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

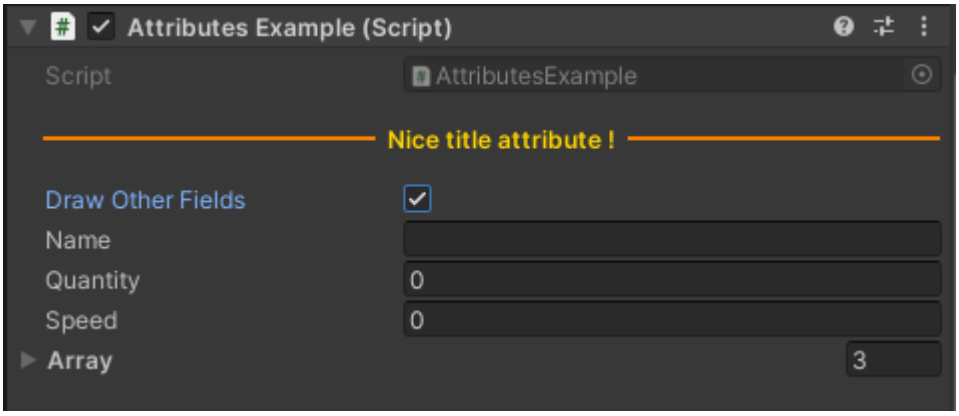
[Tile]

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

Usage

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

Result



Constructor

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

Constants

```
public const float DefaultLineHeight = 1f;
public const TitleColor DefaultLineColor = TitleColor.LightGray;
public const TitleColor DefaultTitleColor = TitleColor.Bright;
```

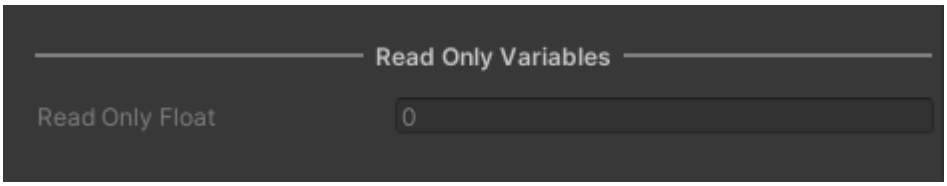
[ReadOnly]

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]

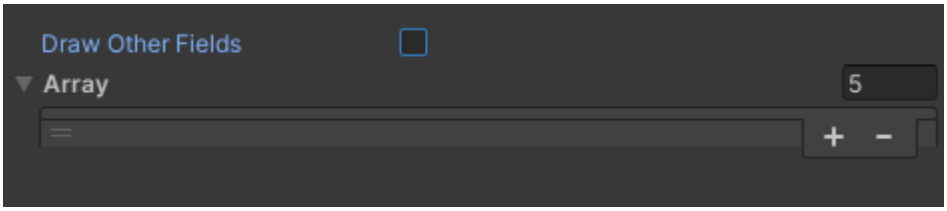
Draw a property field if the condition is true. (Only for Boolean and Enum)
In the example below, we are hiding the field **Name** until the **DrawOtherFields** field value is set to true.

Usage

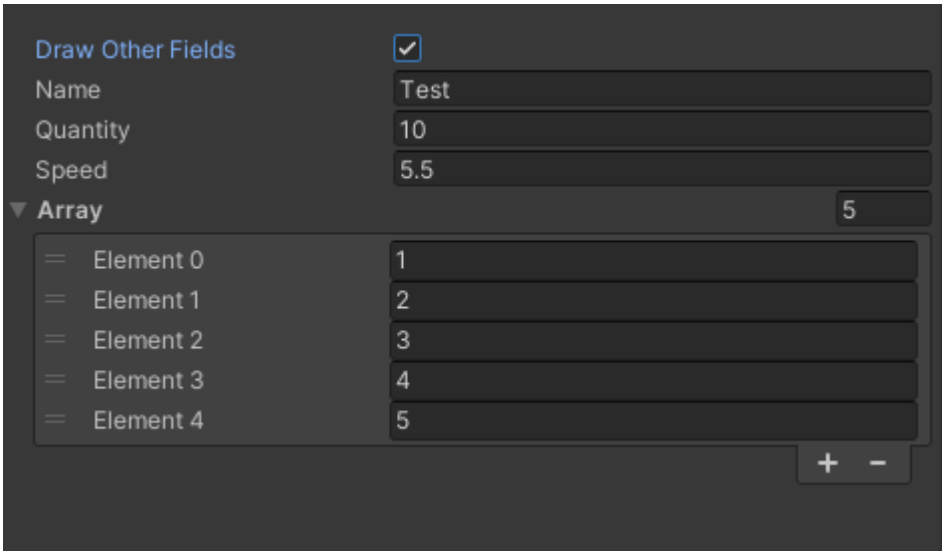
```
public class AttributesExample : MonoBehaviour
{
    public bool DrawOtherFields = false;

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

Result

False



True



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.
/// Defaulted to DisablingType.DontDraw.</param>

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

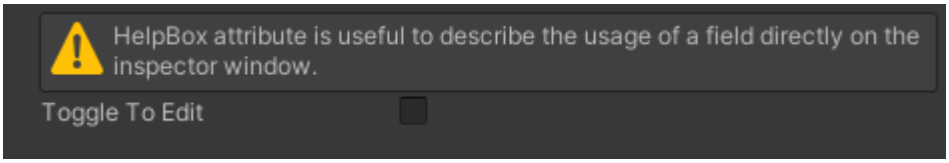
[HelpBox]

Display an 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;
}
```

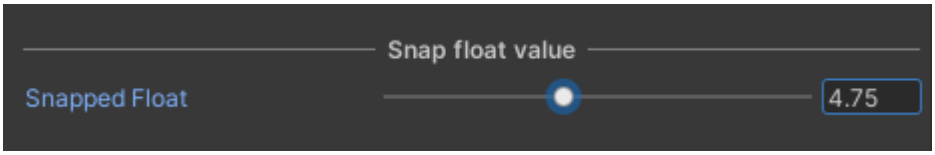
[SnappedSlider]

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.

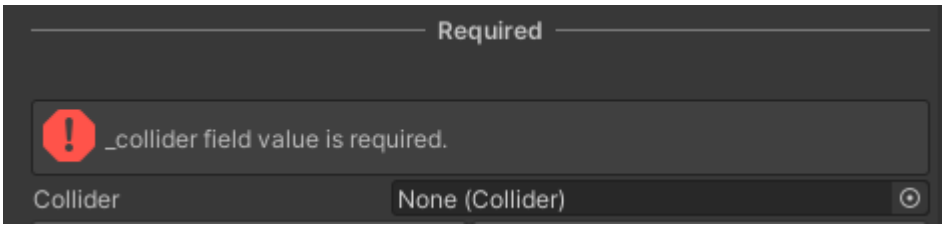
Supported SerializedPropertyType

- String
- ObjectReference
- ExposedReference
- ManagedReference

Usage

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

Result



Constructor

There are no parameters for this attribute.

Special Attributes

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

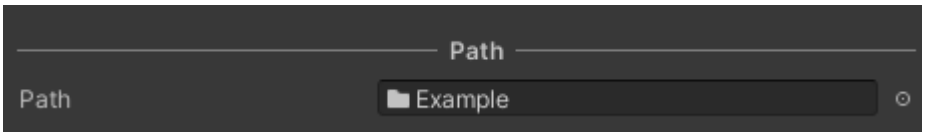
[PathReference]

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

There are no parameters for this attribute.

Limitations

You cannot call PathReference.Path at Runtime, because it's using AssetDatabase class.
You can only use the GUI Property.

[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.
