INSPECTOR BUTTON

By Sabith
mstuzombi@gmail.com

Introduction

The Inspector Button asset, part of the SABI namespace, is a versatile and feature-rich tool designed for Unity developers. By simply applying the [Button] attribute to a method, you create a clickable button directly in the Inspector that works both in Editor and Play modes. This asset not only supports methods with or without parameters but also logs return values for quick testing. Built on

the UI Toolkit, it offers extensive customization options for styling and hover effects.

Key Features

Ease of Use:

Add [Button()] to any method to generate an Inspector button with minimal effort.

Return Value Logging:

Functions returning non-void values will have their outputs logged to the Unity console, aiding debugging.

Button Grouping:

Organize buttons into groups on the same row using the optional groupTag property.

Extensive Customization:

Customize button appearance through properties like dimensions, colors, padding, borders, text styles, and animations—both in normal and hover states.

Parameter Support:

Automatically creates input fields for methods with parameters, handling common data types such as

→ Integers

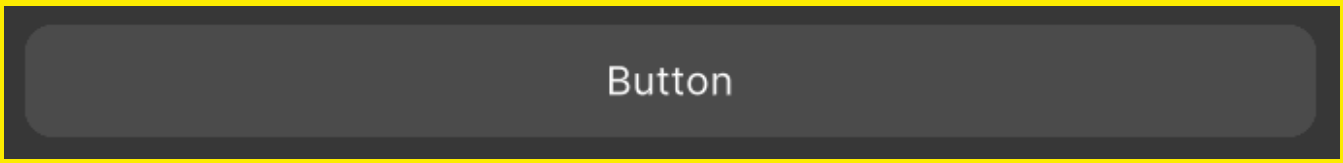
- → Floats
- → Strings
- → Booleans
- → Vector2
- → Vector3
- → Object
- → Colors
- → Enum
- → Double
- → Quaternion
- → GameObject

How to Use

Creating a Basic Button:

To generate a button, add the [Button()] attribute to any method:

```
[Button]
0 references
private string Button() ⇒ " Button return value";
```



Custom Button Text:

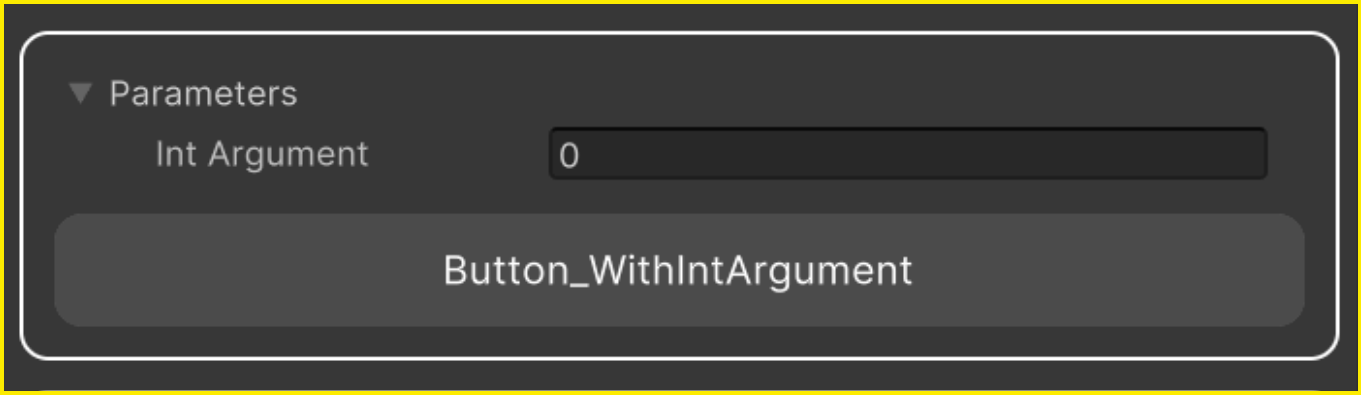
Optionally specify a custom display name:

```
[Button("Button with custom name")]
0 references
private void ButtonnWithName()
{
    Debug.Log($" Button with custon name ");
}
```

Handling Function Parameters:

When a method includes parameters, Inspector Button renders appropriate UI fields for multiple types

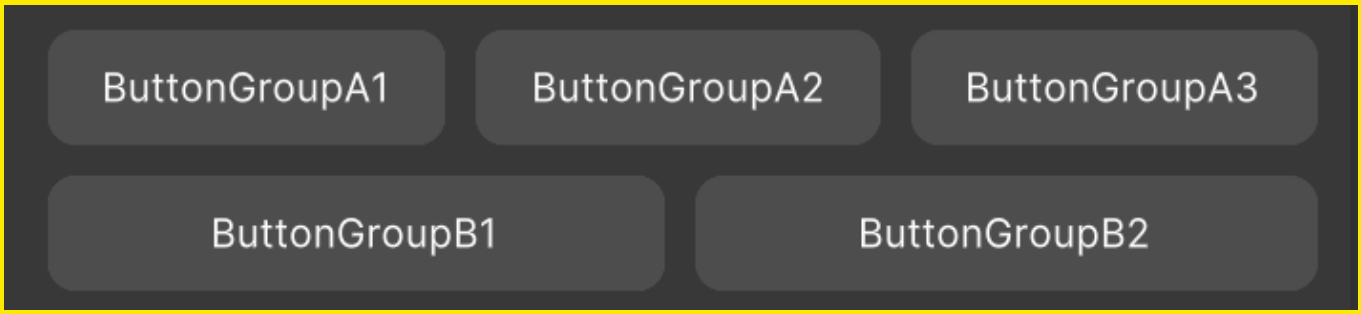
```
[Button]
0 references
private void Button_WithIntArgument(int intArgument)
{
    Debug.Log($" Button intArgument:{intArgument}  ");
}
```



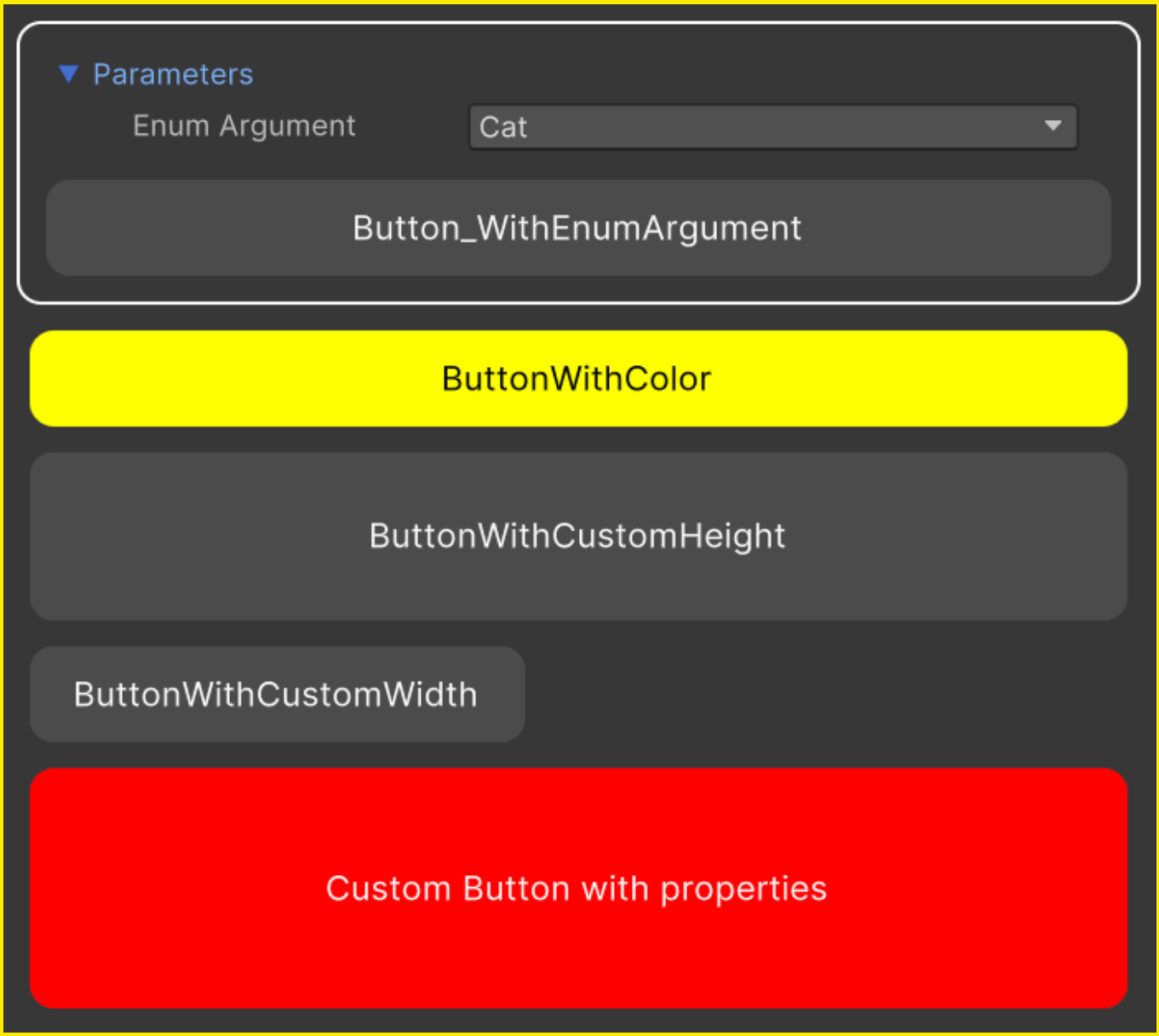
Grouping Buttons:

To display multiple buttons on the same row, assign them the same groupTag value:

```
[Button(groupTag: "GroupA")]
0 references
private string ButtonGroupA1() => " 1st Button og group A";

[Button(groupTag: "GroupA")]
0 references
private string ButtonGroupA2() => " 2nd Button og group A";

[Button(groupTag: "GroupA")]
0 references
private string ButtonGroupA3() => " 3rd Button og group A";

[Button(groupTag: "GroupB")]
0 references
private string ButtonGroupB1() => " 1st Button og group B";

[Button(groupTag: "GroupB")]
0 references
private string ButtonGroupB2() => " 2nd Button og group B";
```



More Examples:



API Reference

`AttributeStyle` **Class**

The `AttributeStyle` class defines the styling options for the Inspector buttons. It supports both normal and hover states.

Constructor Behavior:

- Default values (e.g., `-1` for numeric values) determine whether a property is set or remains null.
- String color values are parsed to Unity's `Color` objects using `ColorUtility.TryParseHtmlString`.

`ButtonAttribute` Class

The `ButtonAttribute` class, defined in the SABI namespace, is the core attribute for creating Inspector buttons. It aggregates two `AttributeStyle` instances: one for normal styling and another for hover styling.

Constructor Parameters:

- **General Button Options:**
  - `groupTag` — (Optional) Groups buttons together.
  - `customName` — (Optional) Custom display text for the button.
- **Normal State Styling:**
  Accepts parameters such as `width`, `height`, `bgColor`, `margin`, `padding`, `borderRadius`, `borderWidth`, `borderColor`, `opacity`, `rotation`, `tooltip`, and various text styling options.
- **Hover State Styling:**
  Offers parallel parameters prefixed with `hover_` (e.g., `hover_width`, `hover_bgColor`, `hover_textColor`, etc.) to define how the button appears on mouse-over. If not provided, normal state values are used as fallbacks.

Example of Using Hover Properties:

Hover styling ensures that when the user hovers over a button, properties such as background color, border, and text style animate smoothly:

```
[Button(
    height: 100,
    bgColor: "#FF0000",
    textColor: "#FFFFFF",
    hover_height: 150,
    hover_bgColor: "#FFFFFF",
    hover_textColor: "#FF0000"
)]
0 references
private void ButtonWithHoverAnimation()
{
    Debug.Log($" Button With Hover Animation ");
}
```

Hover Effects

Overview

The Inspector Button asset includes comprehensive hover effect customization through the `hover_attributeStyle` property in the `ButtonAttribute` class. This allows for seamless and smooth animations on mouse-over, enhancing the visual feedback of buttons.

Customizable Hover Properties

The same styling properties available for the normal state (dimensions, colors, padding, borders, text styles, etc.) are available for hover effects. This enables developers to create visually distinct hover states without additional code.

How Hover Effects Work

- **Property Fallbacks:**
  When hover properties are not explicitly provided, the system falls back to using the corresponding normal state values.
- **Smooth Animations:**
  Built on the UI Toolkit, hover effects transition smoothly. This means properties like background color, border style, and text styling animate gracefully, providing a polished user experience.
- **Implementation:**
  The hover behavior is automatically applied by the custom editor when the Inspector renders the button. Developers simply specify hover parameters in the `[Button]` attribute.

Key Properties for normal and hover states :

Custom Text

customName

Custom display text to show instead of the Function name.

Group

groupTag

To display multiple buttons on the same row, assign them the same groupTag value.

Dimensions

width	Define the button's width. By default, it will take all the available width. If used with groupTag the width will be divided with all buttons in the same group
height	Define the button's height
hover_width	Width on hover
hover_height	height on hover

Background Colors

bgColor	Set the primary background colors
bgColor2	Create a gradient using bgColor and bgColor2 for the background.
hover_bgColor	Set the primary background colors on hover.
hover_bgColor2	Create a gradient using bgColor and bgColor2 for the background on hover.

Margins and Padding

margin	Control spacing around the button.
padding	Control spacing within the button
hover_margin	Control spacing around the button on hover.
hover_padding	Control spacing within the button on hover.

Borders

borderRadius	Rounds button corners
borderColor	Set the primary border colors
hover_borderColor2	Create a gradient using borderColor and borderColor2 for the border on hover.
hover_borderWidth	Determines the thickness of the border on hover.

Opacity and Rotation

opacity	Adjusts transparency
rotation	Rotates the button
hover_opacity	Adjusts transparency on hover.
hover_rotation	Rotates the button on hover.

Text Styling

textSize	Specifies font size.
textColor	Set text color.
textOutlineColor	Define outline color.
textOutlineWidth	Define outline width.
boldText	Toggle text style bold.
italicText	Toggle text style italic
textAlign	Aligns text
textLetterSpacing	Adjust letter spacing.
textWordSpacing	Adjust word spacing.
textOverflow	Handles text-overflow
tooltip	Displays tooltip text on hover

hover_textSize	Specifies font size on hover.
hover_textColor	Set the text color on hover.
hover_textOutlineColor	Define outline color on hover.
hover_textOutlineWidth	Define outline width on hover.
hover_boldText	Toggle text style bold on hover.
hover_italicText	Toggle text style italic on hover.
hover_textAlign	Aligns text on hover.
hover_textLetterSpacing	Adjust letter spacing on hover.
hover_textWordSpacing	Adjust word spacing on hover.
hover_textOverflow	Handles text overflow on hover.
hover_tooltip	Displays tooltip text on hover on hover.

Example Usage

```
0 references | You, 2 seconds ago | 1 author (You) | 1 scene usage | 0 prefab usages
public class ExampleOfInspectorButton : MonoBehaviour
{
    1 reference | 0 references | 0 references | 0 references | 0 references
    public enum exampleEnum { Cat, Dog, Rat, Ball }

    [Button]
    0 references
    private string Button() => " Button return value";

    [Button("Button with custom name")]
    0 references
    private void ButtonnWithName()
    {
        Debug.Log($" Button with custon name ");
    }

    [Button]
    0 references
    private void Button_WithIntArgument(int intArgument)
    {
        Debug.Log($" Button intArgument:{intArgument}  ");
    }

    [Button]
    0 references
    private void Button_WithIntAndStringArgument(int intArgument, string stringArgument)
    {
        Debug.Log($" Button intArgument:{intArgument} stringArgument:{stringArgument} ");
    }

    [Button]
    0 references
    private void Button_WithEnumArgument(exampleEnum enumArgument) =>
        Debug.Log($" Button intArgument:{enumArgument}  ");

    [Button(bgColor: "#FFFF00", textColor: "#000000")]
    0 references
    private void ButtonWithColor() => Debug.Log($" Button ");

    [Button(height: 70)]
    0 references
    private void ButtonWithCustomHeight() => Debug.Log($" ButtonWithCustomHeight ");
```

```
    [Button(width: 200)]
    0 references
    private void ButtonWithCustomWidth() => Debug.Log($" ButtonWithCustomWidth ");

    [Button(
        customName: "Custom Button with properties",
        height: 100,
        bgColor: "#FF0000",
        textColor: "#FFFFFF"
    )]
    0 references
    private void ButtonWithProperties() => Debug.Log($" Button With properties ");

    [Button(groupTag: "GroupA")]
    0 references
    private string ButtonGroupA1() => " 1st Button og group A";

    [Button(groupTag: "GroupA")]
    0 references
    private string ButtonGroupA2() => " 2nd Button og group A";

    [Button(groupTag: "GroupA")]
    0 references
    private string ButtonGroupA3() => " 3rd Button og group A";

    [Button(groupTag: "GroupB")]
    0 references
    private string ButtonGroupB1() => " 1st Button og group B";

    [Button(groupTag: "GroupB")]
    0 references
    private string ButtonGroupB2() => " 2nd Button og group B";

    [Button(
        height: 100, bgColor: "#FF0000",
        textColor: "#FFFFFF", hover_height: 150,
        hover_bgColor: "#FFFFFF",
        hover_textColor: "#FF0000"
    )]
    0 references
    private void ButtonWithHoverAnimation() => Debug.Log($" Button With Hover Animation ");
}
```