BEAUTIFUL INTERFACE

DOCUMENTATION

Beautiful Interface is an asset that can help you create beautiful, animated UI with a few tweaks. Every single animation is created using Tweens (included) for maximum performance. Supports low end platforms like Mobile and WebGL.

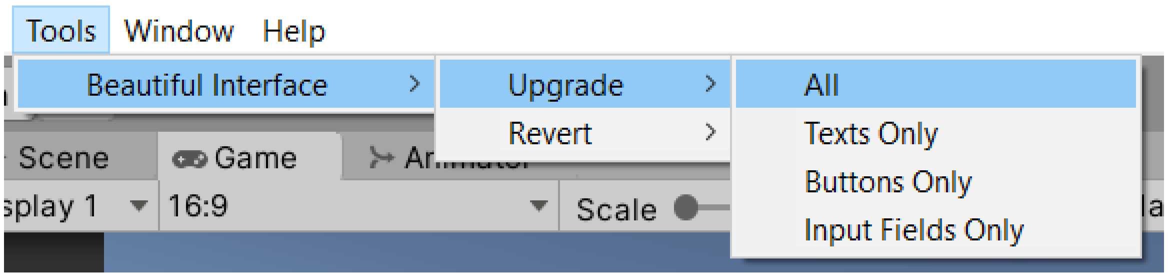
The Unity UI components are functional but limited. Beautiful Interface extends the UI components to create a more feature-rich product.

Contents

General

If you are starting out with a new scene/project, the Demo scene can help you get started with understanding how each element works.

However if you have an existing project and you are looking to improve the UI, then you can one-click upgrade your Text, Button or Input Field. Simply navigate to *Tools ➝ Beautiful Interface ➝ Upgrade*



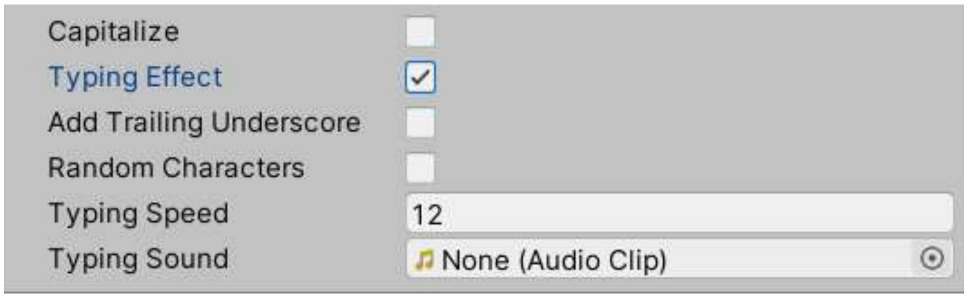
This will replace all the specific elements (Text, Button and InputField) to their upgraded component (TextUI, ButtonUI and InputUI). To undo this, simply navigate to *Tools ➝ Beautiful Interface ➝ Revert*

Note: *One-click Upgrade will not change existing values. For example; Upgrading a built-in Text component to TextUI will keep the text, font, etc. values.*

Reverting will also work the same way except, once you revert, you will lose the Beautiful Interface components' values but you will still keep the built-in values.

Text

The Text is handled by the TextUI script. This is an extension of Unity’s Text component.



Capitalize	☐
Typing Effect	✓
Add Trailing Underscore	☐
Random Characters	☐
Typing Speed	12
Typing Sound	None (Audio Clip)

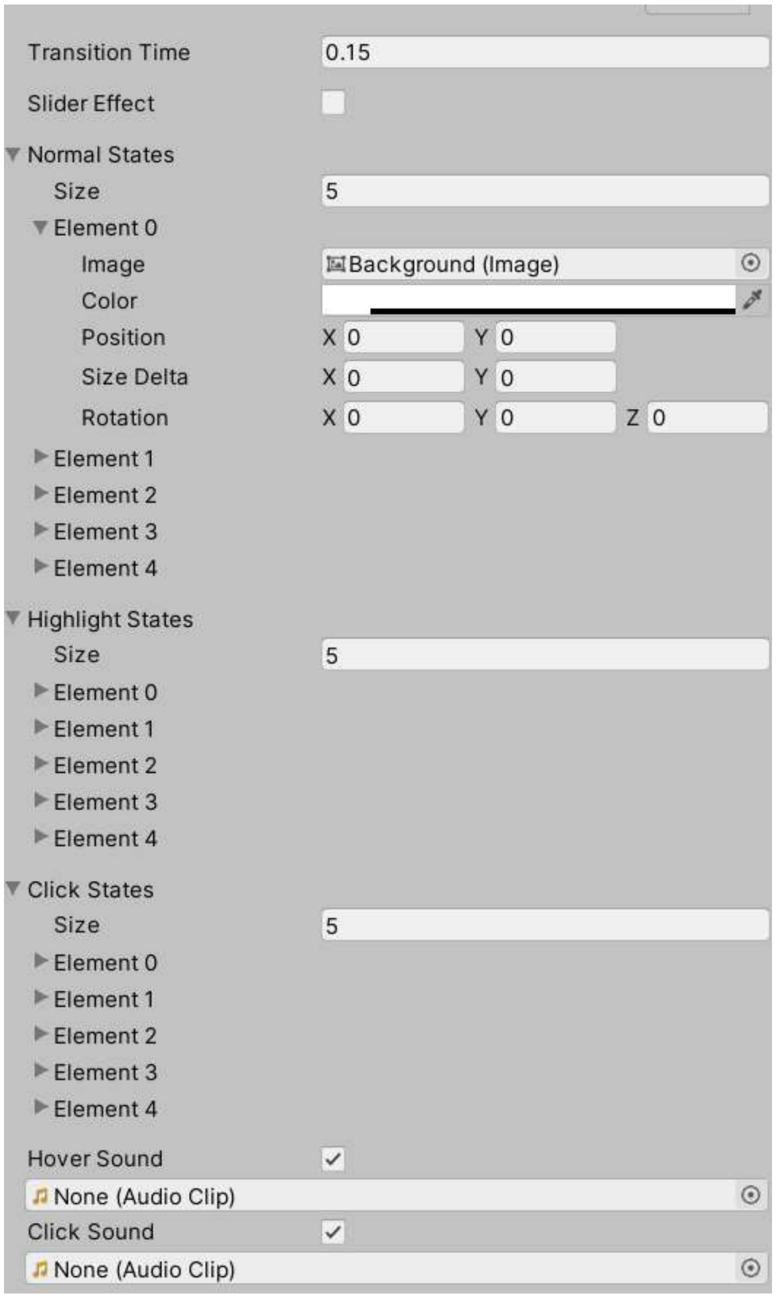
The TextUI inspector (excluding the Unity’s Text fields)

The additional inspector fields are:

- Capitalize: Toggle capitalization in the text
- Typing Effect: Enable a cool typing animation
    - Add Trailing Underscore: An underscore will be shown in the end of the text will animating
    - Random Characters: Similar to Trailing Underscore, except it will be randomized characters to signify decryption animation
    - Typing Speed: The speed of the typing animation
    - Typing Sound: The audio clip played when a character is typed (Requires AudioManager component in the scene)

Button

The Button is handled by the ButtonUI script. This is an extension of Unity's Button component. Check out example buttons in the folder *Prefabs* → *Buttons*.



```
Transition Time        0.15
Slider Effect          ☐
▼ Normal States
    Size               5
  ▼ Element 0
      Image            Background (Image)
      Color
      Position         X 0      Y 0
      Size Delta       X 0      Y 0
      Rotation         X 0      Y 0      Z 0
  ▶ Element 1
  ▶ Element 2
  ▶ Element 3
  ▶ Element 4
▼ Highlight States
    Size               5
  ▶ Element 0
  ▶ Element 1
  ▶ Element 2
  ▶ Element 3
  ▶ Element 4
▼ Click States
    Size               5
  ▶ Element 0
  ▶ Element 1
  ▶ Element 2
  ▶ Element 3
  ▶ Element 4
Hover Sound            ✓
None (Audio Clip)
Click Sound            ✓
None (Audio Clip)
```

The ButtonUI inspector (excluding the Unity's Button fields)

This inspector may seem overwhelming at first, but it is easy to grasp once you know the general idea behind the fields. Basically, ButtonUI animates the UnityEngine Button component without needing to use Animation clips, which increases performance.

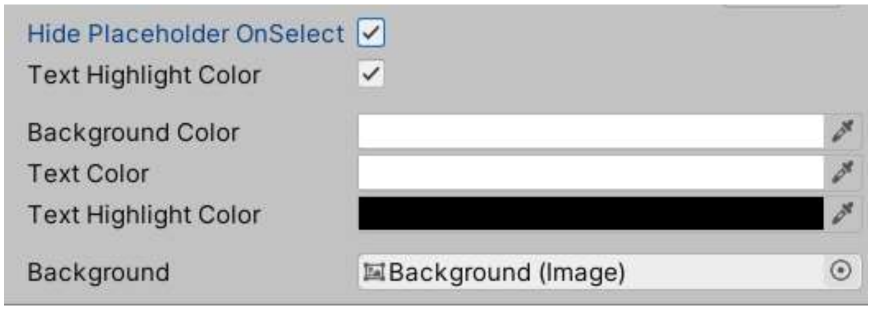
There are three states for the Button: **Normal**, **Highlight** and **Click**. You can set the color, position, size and rotation of each Graphic element in the Button corresponding to a state.

For Lite version, you can only set the color transition states.

Enable/Disable the Hover and Click sounds (**must** have AudioManager component in the scene to play the audio or else disable the fields)

Input Field

The Input Field is handled by the InputUI script. This is an extension of Unity's InputField component. Check out example buttons in the folder *Prefabs→Input Fields*.



Hide Placeholder OnSelect	✓
Text Highlight Color	✓
Background Color	
Text Color	
Text Highlight Color	■
Background	Background (Image)

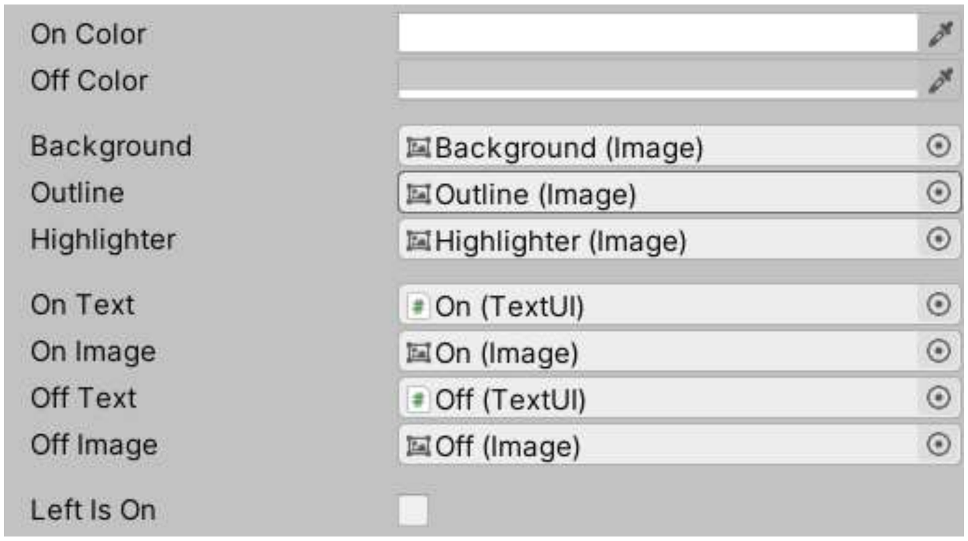
The InputUI inspector (excluding the Unity's InputField fields)

The inspector fields are:

- Hide Placeholder OnSelect: If enabled, the placeholder will fade out when selected. If disabled, the placeholder will zoom out and move to the top of the Input to give a modern style (see InputModern prefab in the examples).
- Text Highlight Color: Show a different color for the text when the input has finished editing and contains some text.
- Background Color: The color of the background image
- Text Color: The color of the text while editing. The placeholder will have a light shade of this color
- Text Highlight Color: The color of the text when finished editing.
- Background: The background image

Toggle

The Toggle is handled by the ToggleUI script. This is an extension of Unity's Toggle component. Check out example buttons in the folder *Prefabs* → *Toggle*.



On Color	
Off Color	
Background	Background (Image)
Outline	Outline (Image)
Highlighter	Highlighter (Image)
On Text	On (TextUI)
On Image	On (Image)
Off Text	Off (TextUI)
Off Image	Off (Image)
Left Is On	☐

The ToggleUI inspector (excluding the Unity's Toggle fields)

ToggleUI is used to create a sliding toggle (see Switch example prefab). If you are looking for a Checkbox, then you are better off using Unity's built-in Toggle (see Simple example prefab).

The inspector fields are:

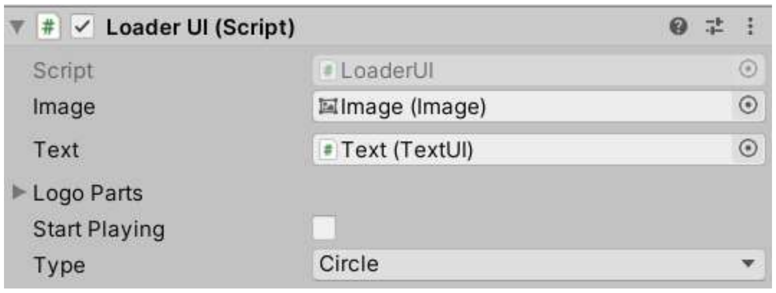
- On Color: The color of the Background image when ON
- Off Color: The color of the Background image when OFF
- Background: The background image component
- Outline: The outline image
- Highlighter: The highlighter image
- On Text: The Text shown when ON
- On Image: The Image to enable when ON
- Off Text: The Text shown when OFF
- Left Is On: If enabled, sliding the toggle to the left equates ON

Loading Icon

You can create custom loading icons using the LoaderUI component. You can find prebuilt icons in the folder *Prefabs → Loader.* You can create 3 types of loading icon (additional customization will be added in due time): Circle loader, Line loader and Logo loader.

See the animations showcased in Demo Full scene to understand the following inspector fields.

Circle Loader:

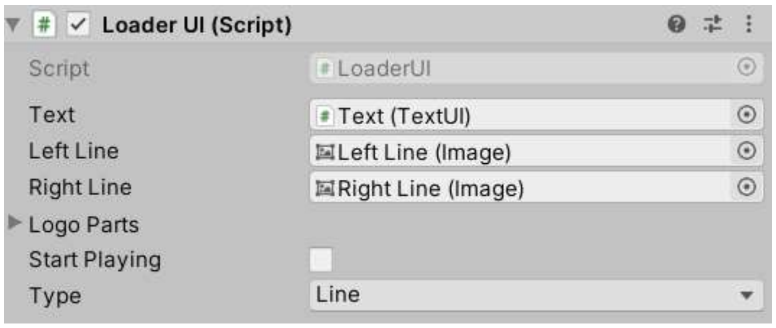


Loader UI (Script)

Script	LoaderUI
Image	Image (Image)
Text	Text (TextUI)
▸ Logo Parts	
Start Playing	☐
Type	Circle

Select *Type* as *Circle*. *Image* is the circle image that will rotate to indicate loading. *Text* is the loading text (optional). *Start Playing* will play the loader animation when scene loads.

Line Loader:

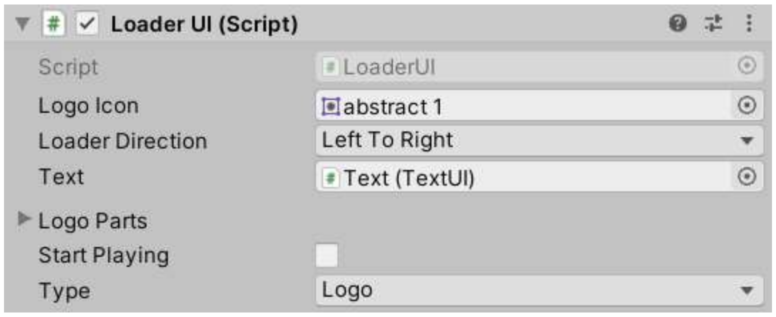


Loader UI (Script)

Script	LoaderUI
Text	Text (TextUI)
Left Line	Left Line (Image)
Right Line	Right Line (Image)
▸ Logo Parts	
Start Playing	☐
Type	Line

Select *Type* as *Line*. *Text* is the loading text (optional). *Left Line* is the line beginning from the left that will extend halfway through the animation. *Right Line* is the line beginning from the right. *Start Playing* will play the loader animation when scene loads.

Logo Loader:



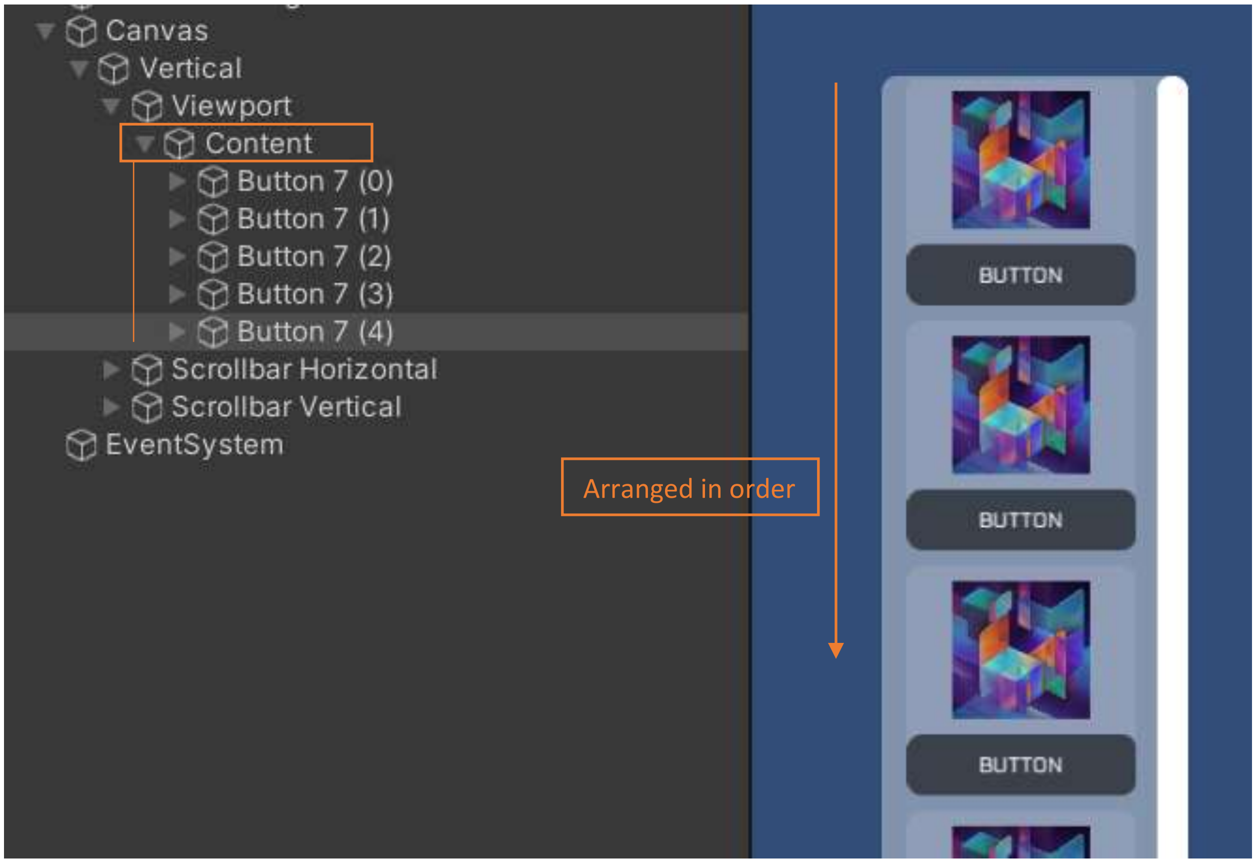
Loader UI (Script)

Script	LoaderUI
Logo Icon	abstract 1
Loader Direction	Left To Right
Text	Text (TextUI)
Logo Parts	
Start Playing	☐
Type	Logo

Select *Type* as *Logo*. *Logo Icon* is the logo that will be animated. This field accepts a Sprite component. *Loader Direction* is the starting and ending direction for the logo animation. *Text* is the loading text (optional). *Logo Parts* are the dissected parts of the Logo that will be animated individually. *Start Playing* will play the loader animation when scene loads.

List

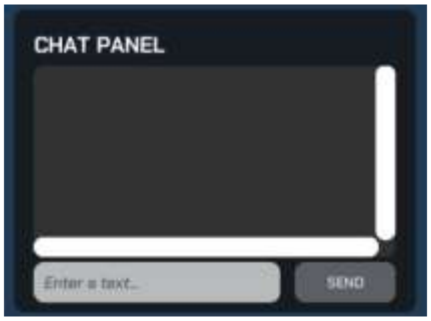
Drag and drop a list layout from the folder *Prefabs → List*. Navigate to the hierarchy *ViewPort → Content* under the List. **The Content is where all the list items need to be placed under.** Place or Instantiate any gameobject as a child of Content to display the item in the list.

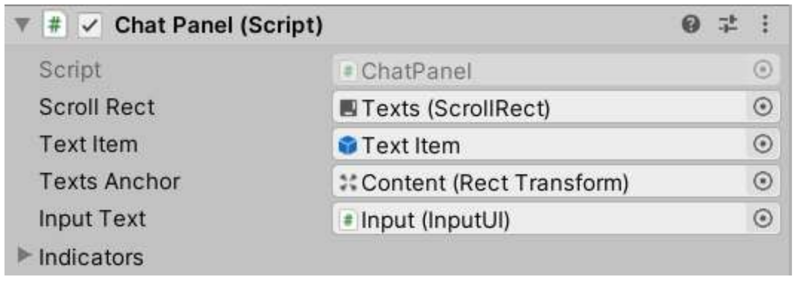


Check out the Friends list in the Demo scene or folder *Prefabs → Panels* to get an idea of how to setup a beautiful list of items.

Chat Panel

Drag and drop a chat panel from the folder *Prefabs*→*Panels*. A chat panel is basically a Vertical list (for the texts) combined with an Input Field (for message) and a Button (to send the message).





Chat Panel (Script)

Script	ChatPanel
Scroll Rect	Texts (ScrollRect)
Text Item	Text Item
Texts Anchor	Content (Rect Transform)
Input Text	Input (InputUI)
▶ Indicators	

The inspector fields are:

- Scroll Rect: The ScrollRect component of the texts list. This will be used to scroll to the bottom when a message is received.
- Text Item: The GameObject item to spawn in the list to display a text.
- Texts Anchor: The Content anchor for the texts list
- Input Text: The Input Field to enter the text
- Indicators: The list of GameObjects that enables when a message is received and the Chat Panel is not active. This can be used as a notification light to notify the user a new message has received

Notification

The Notification script is used to display a notification by spawning a NotificationUI GameObject. The Notification script **must** be attached to another Canvas. To ensure this Canvas stays on top of others, set the SortOrder of the canvas to a high value. **This Canvas will persist throughout the game and will not be destroyed when a new scene is loaded. It is recommended to keep this in the first scene of your game/app.**

You can show a notification through script by calling the *Show* method. This method has two overloads.

```
Show(string title, string description)
```

This will show a notification with the provided title and description. The rest of the default values will be taken from the Notification component inspector fields.

The second overloaded Show(…) is as follows:

```
 <summary>
       Show notification
 </summary>
 Parameters
 <"title">Title
 <"description">Description
 <"icon">Icon
 <"delay">The duration to show the notification
 <"position">The position to show the notification
 <"style">The notification style
 <"isRoundIcon">The notification icon is round
 <"outlineColor">The color of the icon outline
 <"showButtons">Show extra buttons
 <"positiveText">The text on the positive button
 <"onPositive">The action to execute when the positive button is clicked
 <"negativeText">The text on the negative button
 <"onNegative">The action to execute when the negative button is clicked
 <returns>
       The notification ID (used to destroy or hide the notification)
</returns>
```

To access the NotificationUI, store the Notification ID (returned by the Show() method) and use the ID to retrieve the UI component by accessing,

```
public static NotificationUI Get(int id)
```

which returns a NotificationUI component you can edit.