WispGui

Offline Documentation
Last update : August 18th 2022

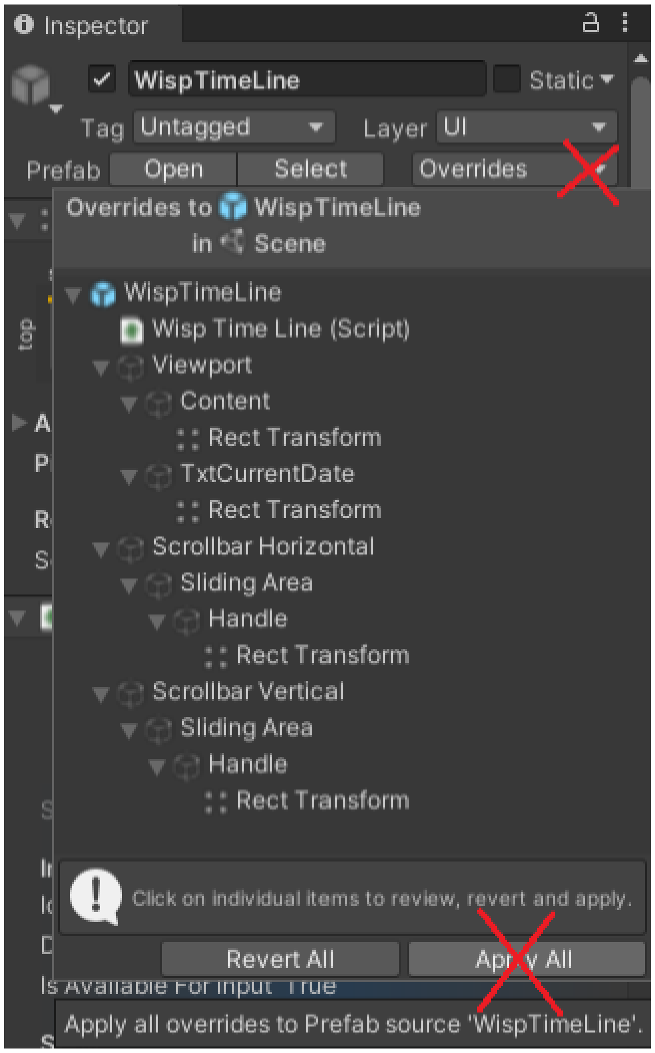
Introduction

WispGUI or The Wisp Graphical User Interface, offers a pack of graphical user interface components for the Unity game engine, the pack contains the following components :

- Editbox
- Dropdown List
- Calendar
- Button
- Checkbox
- Panel
- Scroll View
- Image
- Tab View
- Entity Editor
- Table
- Time Line
- Node Editor
- Button Panel
- Grid
- File Selector
- Input Box
- Message Box
- Popup View
- Context Menu
- Tool Tip
- Loading Panel
- Progress Bar
- Line Renderer
- Slider
- Circular Slider
- Titlebar
- Resizing Handle
- Floating Window
- Bar Chart

Important notes before using Wisp GUI

- While overriding Wisp GUI prefabs is possible, **it's highly discouraged to override prefabs or change their structure especially the hierarchy**. Example of what you should **NOT** do :



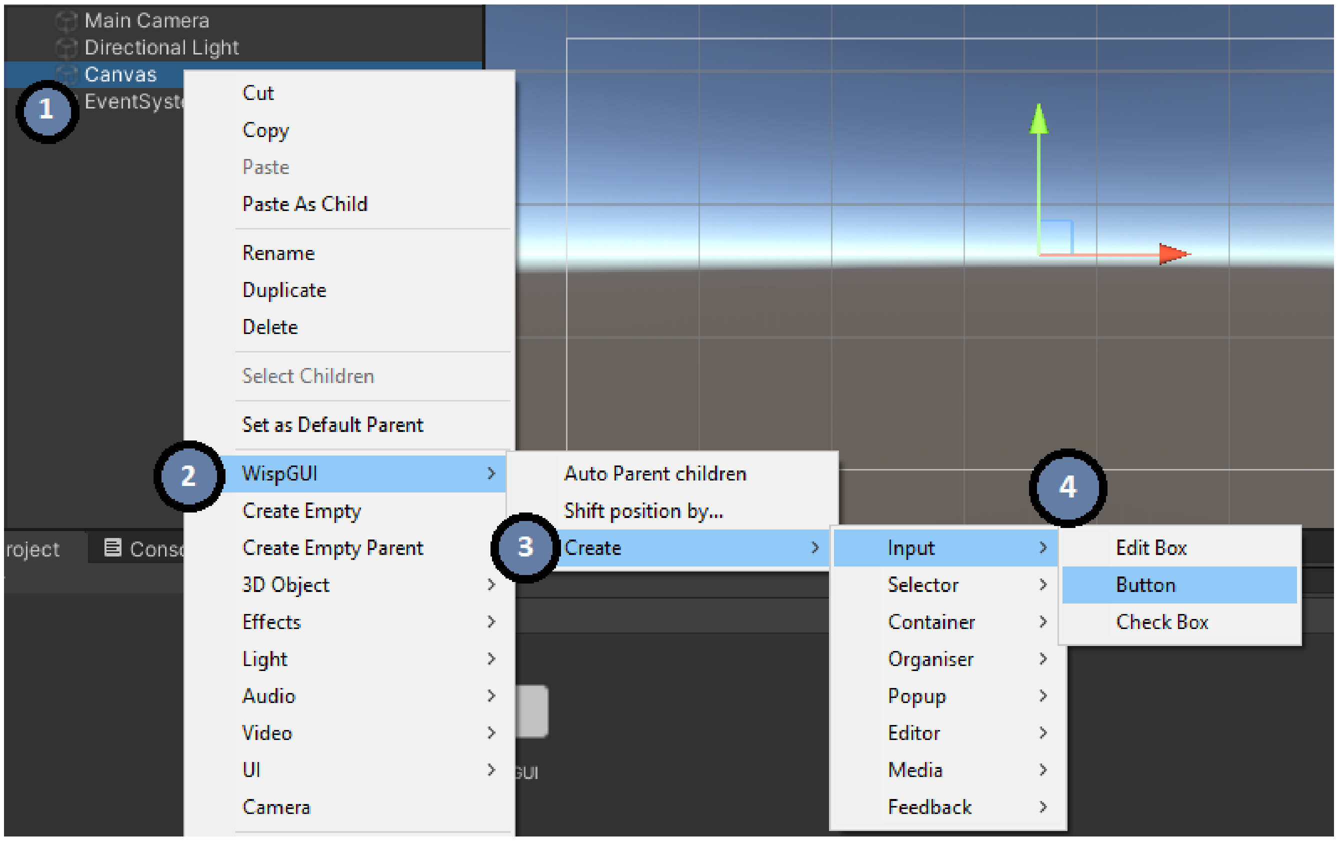
Overrides to WispTimeLine in Scene

- WispTimeLine
  - Wisp Time Line (Script)
  - Viewport
    - Content
      - Rect Transform
    - TxtCurrentDate
      - Rect Transform
  - Scrollbar Horizontal
    - Sliding Area
      - Handle
        - Rect Transform
  - Scrollbar Vertical
    - Sliding Area
      - Handle
        - Rect Transform

Click on individual items to review, revert and apply.

Apply all overrides to Prefab source 'WispTimeLine'.

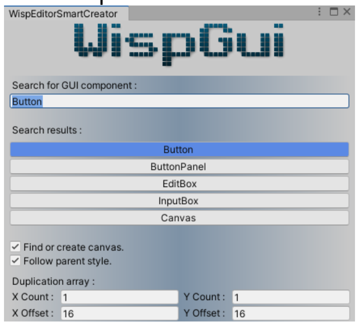
Adding components to your scene



1. The components should be added to a canvas, start by right clicking your canvas.
2. In the menu that appears navigate to the Wisp GUI sub-menu.
3. Then navigate to the Create sub-menu.
4. Choose the component you want to add to your scene.

Or you can simply use the Smart Creator

By Pressing **Control + Space** you can open the smart creator and type the component name

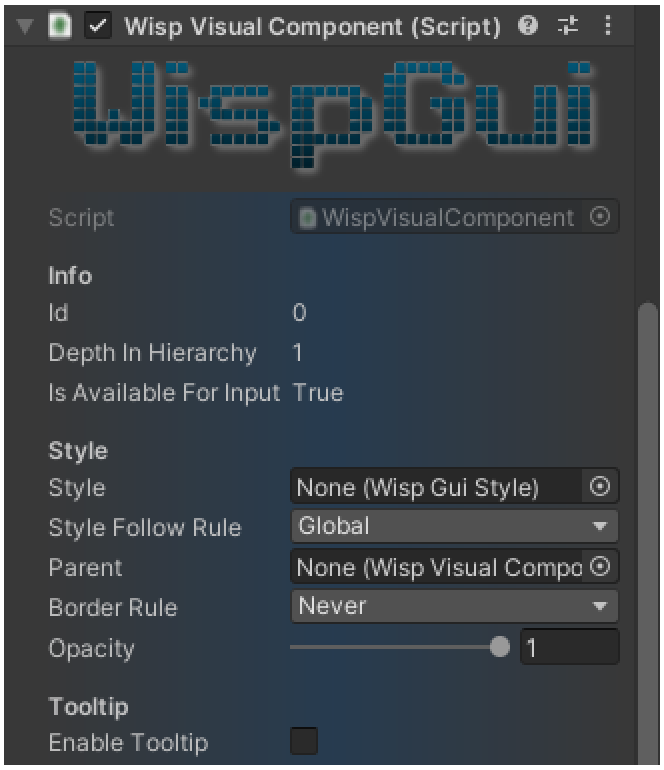


Search for GUI component : Button

✓ Find or create canvas.
✓ Follow parent style.

Duplication array :
X Count : 1 Y Count : 1
X Offset : 16 Y Offset : 16

The Basic Wisp Visual Component explained



Script WispVisualComponent

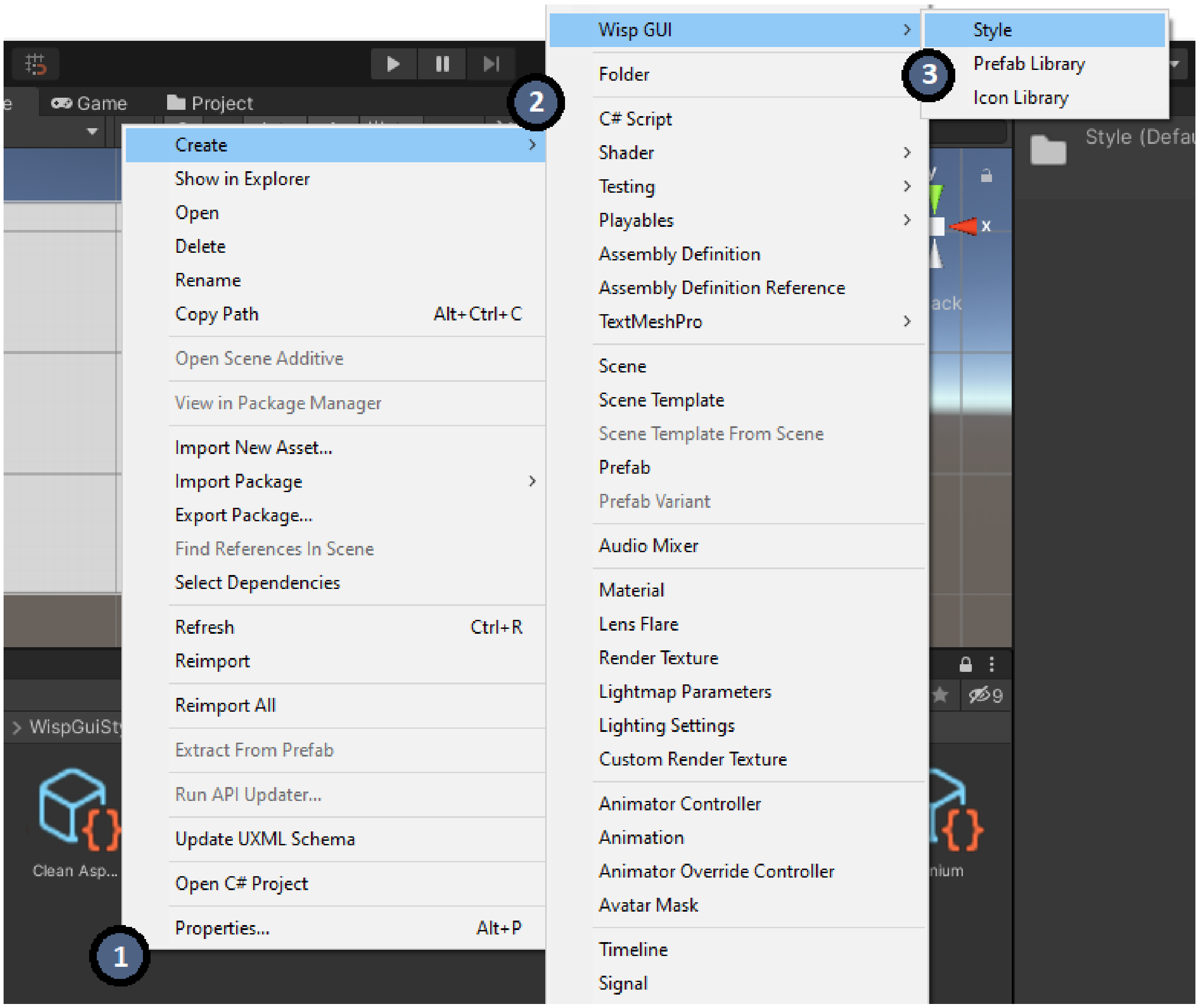
Info
Id 0
Depth In Hierarchy 1
Is Available For Input True

Style
Style None (Wisp Gui Style)
Style Follow Rule Global
Parent None (Wisp Visual Compo
Border Rule Never
Opacity 1

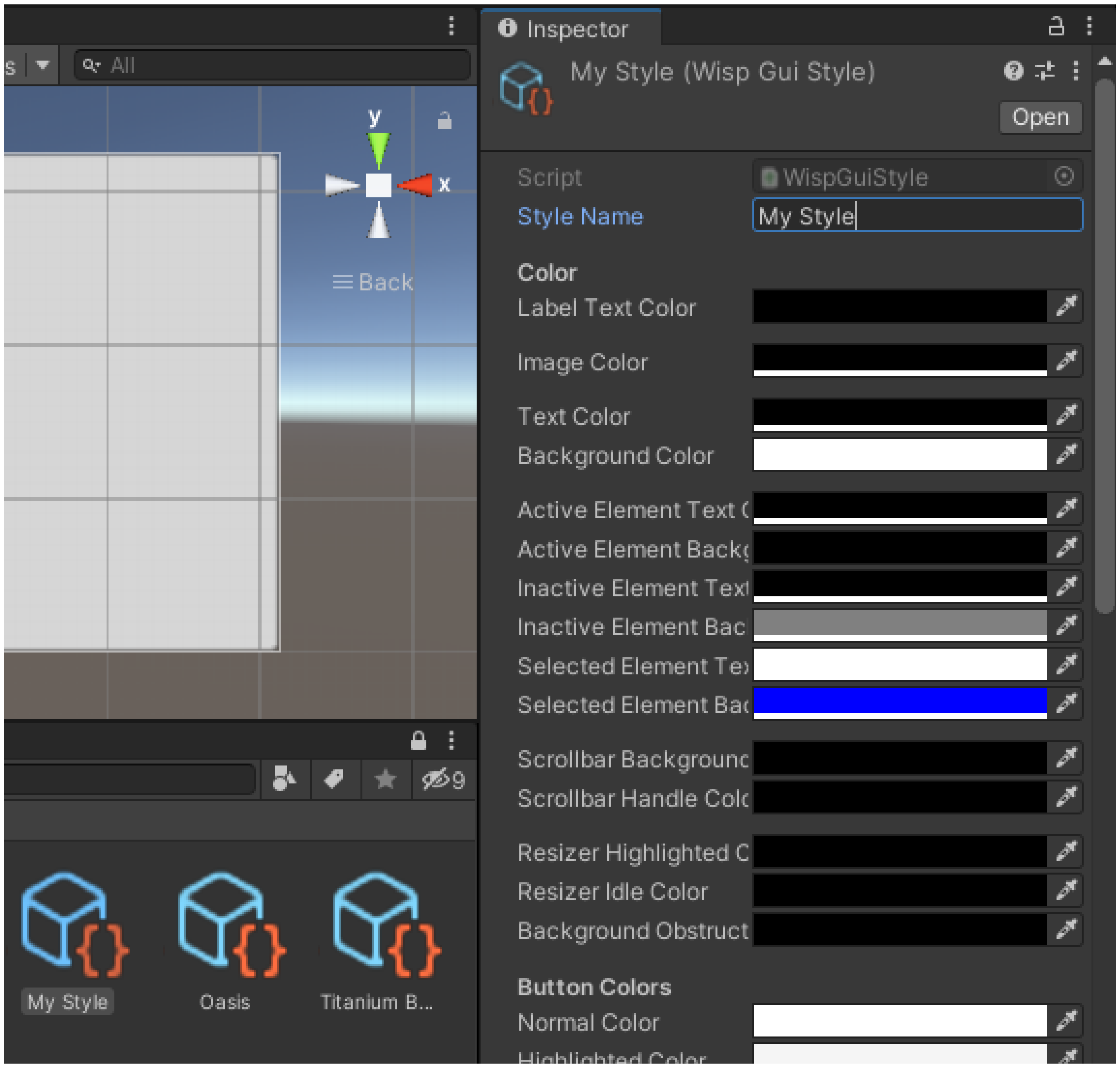
Tooltip
Enable Tooltip

- **ID** : Each WispVisualComponent is assigned a unique ID at initialization, the ID can be used to keep track of the component or search for it.
- **Depth In Hierarchy** : By assigning a parent to the component it's depth hierarchy changes to the depth of the parent plus one.
- **Is Available For Input** : Indicated wheter or not the component should respond to kayboard input. For example components obstructed by a dialog window or components in innactive tabs are not available for input.
- **Style** : Assign a style sheet that defines the visual appearance of the component, the object type of the style sheet must be WispGuiStyle, styles can be found in WispGui/Assets/WispGuiStyle/Style folder.
- **Parent** : One way to assign a parent to a component is through this field in it's inspector.
- **Border Rule** : Choose when to display the borders of the component, the border color is designated in the style sheet used by the component.
- **Opacity** : Defines the opacity of the component by changing the alpha value of it's colors.
- **Enable Tooltip** : Choose whether the component shows a tooltip when the mouse cursor is over it.

Creating custom styles



1. Start by left clicking in your asset folder.
2. Navigate to the Create sub-menu.
3. Navigate to the Wisp GUI sub-menu and select Style.
4. Give a name to your newly created style and customize it using the inspector :



Inspector

My Style (Wisp Gui Style)

Open

Script	WispGuiStyle
Style Name	My Style

Color

Label Text Color	⬛
Image Color	⬛
Text Color	⬛
Background Color	⬜
Active Element Text C	⬛
Active Element Backg	⬛
Inactive Element Text	⬛
Inactive Element Bac	▨
Selected Element Tex	⬜
Selected Element Bac	🟦
Scrollbar Backgrounc	⬛
Scrollbar Handle Colc	⬛
Resizer Highlighted C	⬛
Resizer Idle Color	⬛
Background Obstruct	⬛

Button Colors

Normal Color	⬜
Highlighted Color	⬜

Component usage examples

01 - Edit Box : An extension of Unity's Input Field, allows the input of a string.

Script example :
```
    // Set value from script.
    GetComponent<WispEditBox>().SetValue("Some random text...");
```

02 - Dropdown List : Can be directly used from an Edit Box, allows the selection of a string from a list.

Script example :
```
    // Add an item to the list.
    GetComponent<WispDropDownList>().AddItem("1","First Item");
```

03 - Calendar : Can be directly used from an Edit Box, allows the selection of a date string.

Script example :
```
    // Print the currently selected date to the console.
    string date = GetComponent<WispCalendar>().GetValue();
    print(date);
```

04 - Button : An extension of Unity's Button, triggers one or multiple actions when pressed.

Script example :
```
    // Set button text from script.
    GetComponent<WispButton>().SetValue("Click Me");
```

05 - Checkbox : An extension of Unity's Toggle, returns true or false.

Script example :
```
    // Check the box from script.
    GetComponent<WispCheckBox>().SetValue("true");
```

06 - Panel : Simply a Game object with a Rect transform and an image attached to it, that serves as a background for other components.

**07 - Scroll View : **An extension of Unity's Scroll view.

Script example :

```
    // Scroll to target postion over 1 second.
    Vector3 targetPosition = new Vector3(512,512,0);
    GetComponent<WispScrollView>().ScrollToPosition_Async(targetPosition, 1f);
```

**08 - Image : **An extension of Unity's Image, Allows loading images from sprites or URLs.

Script example :

```
    // Change current image, the source is a URL in this case.
    GetComponent<WispImage>().SetValue("https://picsum.photos/200/300?grayscale");
```

**09 - Tab View : **Allows the organisation of scroll views into tabs as pages.

Script example :

```
    // Add a page and it's tab to the tab view.
    GetComponent<WispTabView>().AddPage("0","Test Page");
```

**10 - Entity Editor : **Generates an input form of a WispEntity according to it's properties.

Script example :

```
    // Create a new entity.
    WispEntityInstance person = new WispEntityInstance("person","Person");

    // Add properties to the entity.
    person.AddProperty(new WispEntityPropertyText("name", "Full Name"));
    person.AddProperty(new WispEntityPropertyDate("date_of_birth", "Date of Birth"));
    person.AddProperty(new WispEntityPropertyBool("is_online", "Is Online ?"));

    // Generate a form to record a person, in this case two edit boxes and a check box will be rendered.
    GetComponent<WispEntityEditor>().RenderInstance(person);
```

11 - Table : Allows the organisation of data in columns and rows.

Script example :

```
    WispTable table = GetComponent<WispTable>();

    // Add columns to the table.
    table.AddColumn("1", "First column");
    table.AddColumn("2", "Second column");
    table.AddColumn("3", "Third column");

    // Add rows to the table and fill each cell with a letter.
    table.AddRowWithValues("A","B","C");
    table.AddRowWithValues("D","E","F");
    table.AddRowWithValues("G","H","I");
```

12 - Time Line : Display events or groups of events on a time line between two dates.

Script example :

```
    // Create a new event.
    WispTimeLineEvent myEvent = new WispTimeLineEvent("big_bang","Big Bang", new System.DateTime(0,0,0));

    // Add the event to the Time Line.
    GetComponent<WispTimeLine>().AddEvent(myEvent);

    // Update event marks positions on the Time Line.
    GetComponent<WispTimeLine>().UpdatePositions();
```

13 - Node Editor : Provides a canvas for building node trees.

Script example :

```
    // Add a new node at the center of the node editor, it will be hovering until a mouse click is detected.
    GetComponent<WispNodeEditor>().AddNewNode(Vector2.zero, true);
```

14 - Button Panel : Render buttons from a list onto a panel.

Script example :

```
    // Add two buttons and assign actions to both.
    GetComponent<WispButtonPanel>().AddButton("ok_button", "Ok", okOnPress);
    GetComponent<WispButtonPanel>().AddButton("cancel_button", "Cancel", cancelOnPress);
```

15 - Grid : Allows the rendering of a grid and the organisation of components into a grid layout.

Script example :

```
    // Define a standard button size for later use;
    const float buttonSize = 64f;

    // Assign grid dimensions, 2 columns and 2 rows in this case.
    // The result is 4 cells in total.
    grid = GetComponent<WispGrid>();
    grid.SetDimensions(2,2);

    // Create a button at cell 0.
    WispButton btnAdd = WispButton.Create(grid.GetCell(0).MyRectTransform);
    // Assign a width and a height to the button.
    btnAdd.Width = buttonSize;
    btnAdd.Height = buttonSize;
    // Assign an icon to the button.
    btnAdd.IconPlacement = WispButton.WispButtonIconPlacement.Full;
    btnAdd.SetIcon(WispIconLibrary.Default.Add);

    // Do the same with a button at cell 1.
    WispButton btnEdit = WispButton.Create(grid.GetCell(1).MyRectTransform);
    btnEdit.Width = buttonSize;
    btnEdit.Height = buttonSize;
    btnEdit.IconPlacement = WispButton.WispButtonIconPlacement.Full;
    btnEdit.SetIcon(WispIconLibrary.Default.Edit);

    // Do the same with a button at cell 2.
    WispButton btnDelete = WispButton.Create(grid.GetCell(2).MyRectTransform);
    btnDelete.Width = buttonSize;
    btnDelete.Height = buttonSize;
    btnDelete.IconPlacement = WispButton.WispButtonIconPlacement.Full;
    btnDelete.SetIcon(WispIconLibrary.Default.Delete);

    // Do the same with a button at cell 3.
    WispButton btnLoad = WispButton.Create(grid.GetCell(3).MyRectTransform);
    btnLoad.Width = buttonSize;
    btnLoad.Height = buttonSize;
    btnLoad.IconPlacement = WispButton.WispButtonIconPlacement.Full;
    btnLoad.SetIcon(WispIconLibrary.Default.Directory);

    // Make the grid cells fit the total size of the grid game object.
    grid.AutoFit();
```

16 - File Selector : A dialog that allows the selection of a file for saving and loading purposes.

Script example :

```
private void OpenFileSelector()
{
    // Open a selector to load a file, run printFile() when the OK button is pressed.
    WispFileSelector.OpenAuto("", "", printFile, false, false);
}

private void printFile()
{
    // Get the path to the selected file.
    string path = WispFileSelector.GetLastSelectedFilePath();

    // Load the content of the file.
    string fileContent = File.ReadAllText(path);

    // Print the content of the file to the console.
    print(fileContent);
}
```

17 - Input Box : A dialog that allows the input of a string.

Script example :

```
private void OpenInputBox()
{
    // Prepare a container for the resulting input.
    WispInputResult result = null;

    // Open the input dialog box.
    result = WispInputBox.OpenInputDialog("Enter your name : ", delegate { printName(result); });
}

private void printName(WispInputResult ParamResult)
{
    // Print input to the console.
    print(ParamResult.Result);
}
```

18 - Message Box : A dialog that displays a message and buttons to respond with.

Script example :

```
    // Open a message box with three buttons.
    WispMessageBox.OpenThreeButtonsDialog("What is your choice ?", "A", chooseA, "B", chooseB, "C", chooseC);
```

19 - Popup View : A dialog that displays a scroll view and button panel, which serves as a multi purpose dialog that can be customized depending on your needs.

Script example :
```
    // Create and open a popup view.
    WispPopupView view = WispPopupView.CreateAndOpen();

    // Add a close button to the popup view.
    view.ButtonPanel.AddButton("close","Close",WispModalWindow.ClosePopupInParent);

    // Create a button with some text and put it inside the scroll view of the popup view.
    WispButton btn = WispButton.Create(view.ScrollView.ContentRect);
    btn.Width = 128;
    btn.Height = 128;
    btn.SetValue("Put what you want in this window");
```

20 - Context Menu : Renders a menu from a list of items with the posibility of alligning it with the position of the mouse cursor.

Script example :
```
    // Create and open a context menu at mouse position as a child of the main canvas.
    RectTransform mainCanvasRT = WispVisualComponent.GetMainCanvas().GetComponent<RectTransform>();
    WispContextMenu menu = WispContextMenu.CreateAndOpenAtMousePosition(mainCanvasRT);

    // Add items to the menu.
    menu.AddItem("1","Option 1", option_1);
    menu.AddItem("2","Option 2", option_2);
    menu.AddItem("3","Option 3", option_3);
```

21 - Tooltip : A Text box with a header and a text, If Tooltip is enabled for the target component it will automaticly show up when hovering over the component.

Script example :
```
    // Enable tooltip from script.
    GetComponent<WispVisualComponent>().EnableTooltip = true;

    // Set tooltip title and content from script.
    GetComponent<WispVisualComponent>().SetTooltipText("Tooltip Title","Some text here...");
```

22 - Loading Panel : A panel with an animation that can be used to indicate operations like loading or processing.

Script example :
```
private void LoadingPanelExample()
{
   // Prepare RectTransform of the main canvas.
   RectTransform mainCanvasRt = WispVisualComponent.GetMainCanvas().GetComponent<RectTransform>();

   // Create a loading panel as a child of the main canvas.
   WispLoadingPanel loadingPanel = WispLoadingPanel.Create(mainCanvasRt);

   // Disable the loading panel for now.
   loadingPanel.gameObject.SetActive(false);

   // Start the loading process.
   StartCoroutine(LoadSomething(loadingPanel));
}

IEnumerator LoadSomething(WispLoadingPanel ParamPanel)
{
   // Enable the loading panel.
   ParamPanel.gameObject.SetActive(true);

   // Wait 5 seconds.
   yield return new WaitForSeconds(5f);

   // Then disable the loading panel to indicate that the loading is done.
   ParamPanel.gameObject.SetActive(false);

   yield return null;
}
```

23 – Progress Bar : Display a horizontal or vertical value indicator that accepts float values from 0 to 100.

Script example :
```
// Using the progress bar as a health bar.
public static void UpdateHealthBar(float ParamHealth)
{
     WispProgressBar bar = main.healthBar;
     bar.SetValue(ParamHealth);
}
```

24 - Line Renderer : A component that render lines on the canvas.

Script example :

```
[SerializeField] private RectTransform a;
[SerializeField] private RectTransform b;
[SerializeField] private RectTransform canvas;

private WispLineRenderer line;

void Start()
{
  line = GetComponent<WispLineRenderer>();

  // Always set the width of the line, default is 0.
  line.Width = 1f;

  // Draw line from object a to object b.
  line.SetStartAndEndPoint(a,b);
}

void Update()
{
  b.anchoredPosition = canvas.GetMousePositionInMe();

  // Update line when objects move.
  line.SetStartAndEndPoint(a,b);
}
```

25 - Slider : An extension of Unity's Slider, outputs a percentage or a value between 0 and 1 depending on the position of it's Handle or a value between Min and Max, The handle can be dragged across the slider bar using mouse or touch.

Script Example :

```
public WispVisualComponent[] targets;

private WispSlider slider;

// Start is called before the first frame update
void Start()
{
  slider = GetComponent<WispSlider>();
  slider.Base.onValueChanged.AddListener(OnValueChanged);
}

// Method to call whenever the slider value has changed
private void OnValueChanged(float ParamValue)
{
  // Slider value, between Min and Max.
  float valueFromEvent = ParamValue;

  // Slider value, between Min and Max, returned as string and conveted to float.
  float sliderValue = slider.GetValue().ToFloat();

  // Slider value, between 0 and 1, depending on the handle position.
  float value01 = slider.GetValue01();

  foreach(WispVisualComponent vc in targets)
  {
    vc.Opacity = 0.5f + (value01 / 2);
  }
}
```

26 – Circular Slider : An circular version of the slider, outputs a percentage or a value between 0 and 1 depending on the position of it's Handle, The handle can be dragged across the circular bar using mouse or touch.

Script Example :

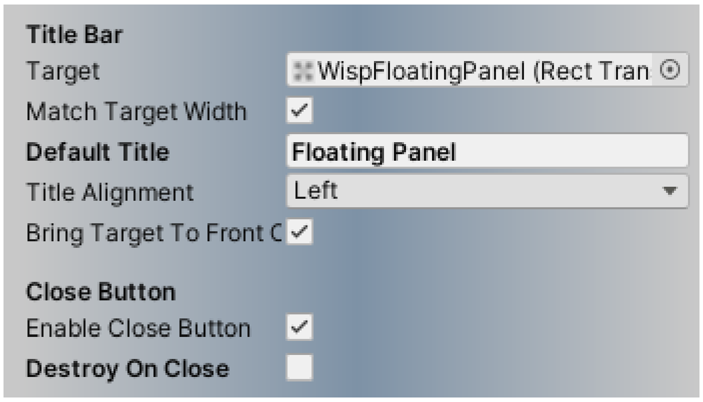
```
public float minTemperature = 12f;
public float maxTemperature = 36f;

private WispCircularSlider slider;

// Start is called before the first frame update
void Start()
{
  slider = GetComponent<WispCircularSlider>();
  slider.OnValueChanged.AddListener(UpdateText);
}

void UpdateText()
{
  float t = slider.GetValue01();
  temperatureText.text = Mathf.Lerp(minTemperature, maxTemperature, t).ToString("N0") + "° C";
}
```

27 – Title Bar : A bar that can be attached to any RectTransform in order to control it's position as well as providing an icon, a label and a button to close it or perform another action like hide or minimize.

Script Example :

```
private WispTitleBar bar;

void Start()
{
    bar = transform.Find("WispTitleBar").GetComponent<WispTitleBar>();

    // Assigning a method to call when the close button is pressed
    bar.ExitButton.AddOnClickAction(OnCloseButtonClick);
}

// Method to call when the Close button is pressed
private void OnCloseButtonClick()
{
    if (bar.Parent.Opacity > 0.5f)
    {
        bar.Parent.Opacity = 0.5f;
    }
    else
    {
        Destroy(bar.Target.gameObject);
    }
}
```

In order for this to work correctly make sure EnableCloseButton is set to TRUE and DestroyOnClose is set to FALSE :



Title Bar

Target	WispFloatingPanel (Rect Tran
Match Target Width	✓
Default Title	Floating Panel
Title Alignment	Left
Bring Target To Front C	✓

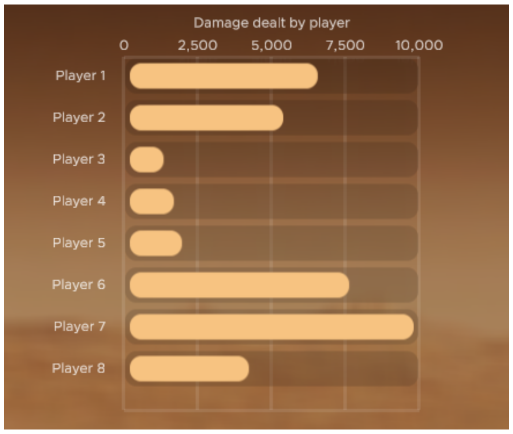
Close Button

Enable Close Button	✓
Destroy On Close	

28 – Resizing Handle : A handle that appears as a little triagle in the bottom right of the component it's targeting, The handle serves to resize the the target RectTransform when dragged with mouse or touch.

29 – Floating Window : A panel with a **Title Bar** and a **Resizing Handle.**

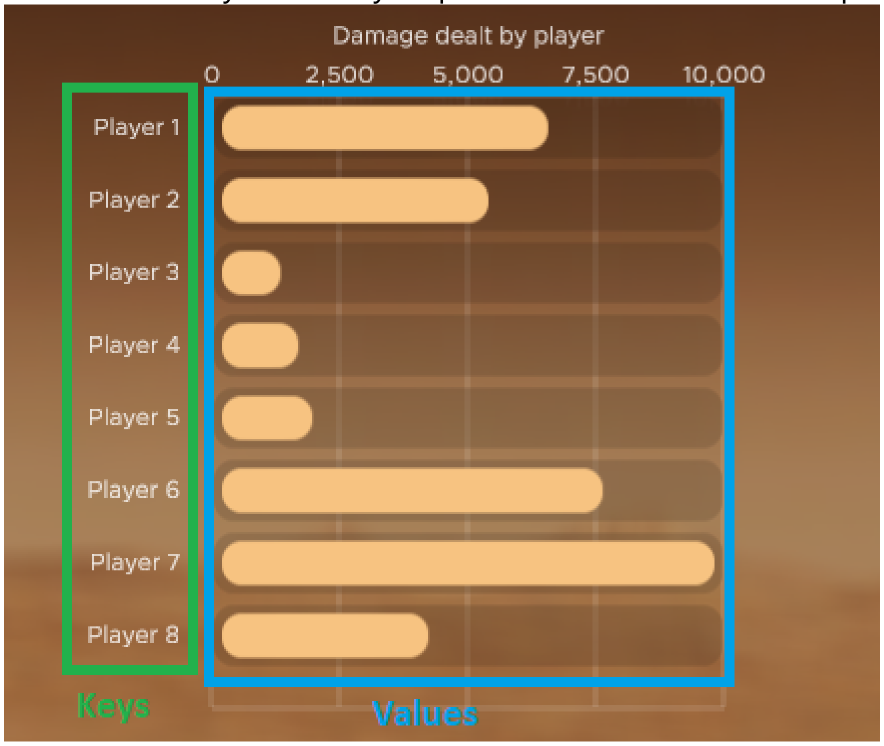
30 – Bar Chart : A chart that presents data with rectangular bars with lengths proportional to the values that they represent.



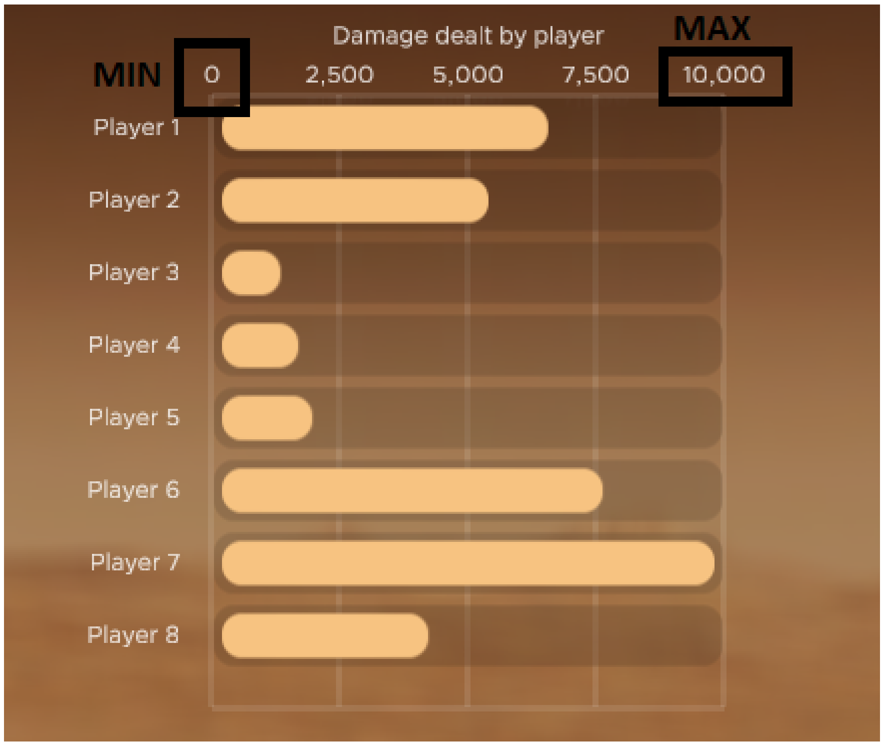
Script Example :

```
private void Regenerate()
{
  #region Generate some random data
  const float minDamage = 1000;
  const float maxDamage = 10000;

  Dictionary<string, float> playerDamage = new Dictionary<string, float>();

  float maxDamagePerPlayer = UnityEngine.Random.Range(minDamage, maxDamage);

  for (int i = 1; i <= 8; i++)
  {
    playerDamage.Add("Player " + i.ToString(), UnityEngine.Random.Range(minDamage, maxDamagePerPlayer));
  }
  #endregion

  // Use the random data to draw a chart
  chart.DrawChart(playerDamage, 0, maxDamage, 4, "Damage dealt by player");
}
```

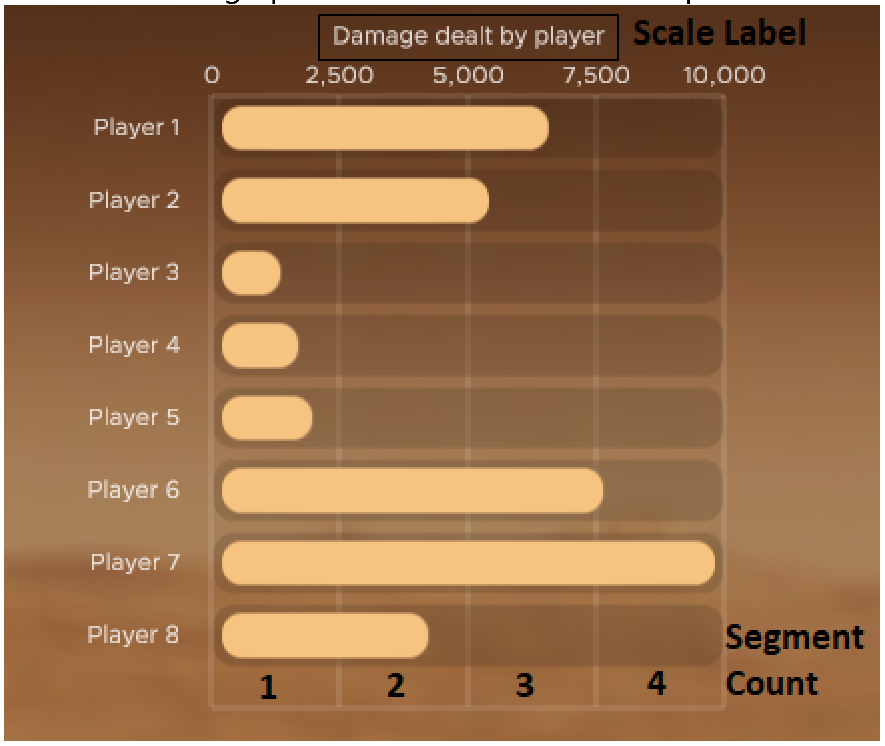
The DrawChart() method takes 5 parameters :
Labels and Values : A dictionary where keys represents labels and values represent values.



Minimum Label Value : The minimum value on the scale.
Maximum Label Value : The maximum value on the scale.



Segment Count : The number of segments that devides the scale.
Scale Label : A label above the graph that describes the data or provides a unit of measure.


Damage dealt by player
Scale Label
Player 1
Player 2
Player 3
Player 4
Player 5
Player 6
Player 7
Player 8
Segment Count