ToolsForFree:

Advanced Runtime Console

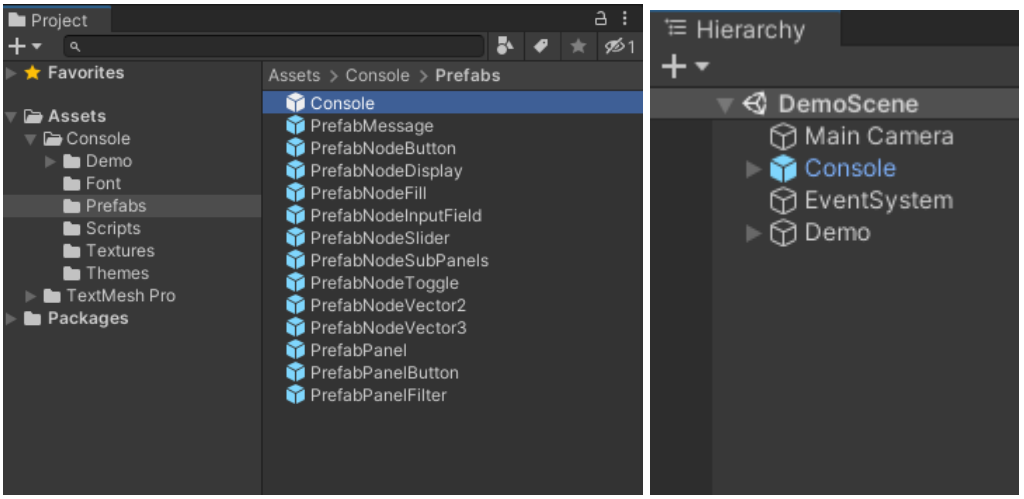
Advanced Runtime Console gives you access to a scene based Console with many features.

- Default behaviour of Unity's console mimicked.
- Pin messages and update them over time.
- Display any type of custom data with multiple types of entries:
  - Labels
  - Input fields
  - Fill bars
  - Vectors
- Set callback methods to interact with your game as you like:
  - Toggles
  - Sliders
  - Buttons
- Group entries using panels, under other entries or using subpanels.
- Create your own custom entries types with ease.
- Fully customizable aspect with just a few clicks.
- Fully documented.

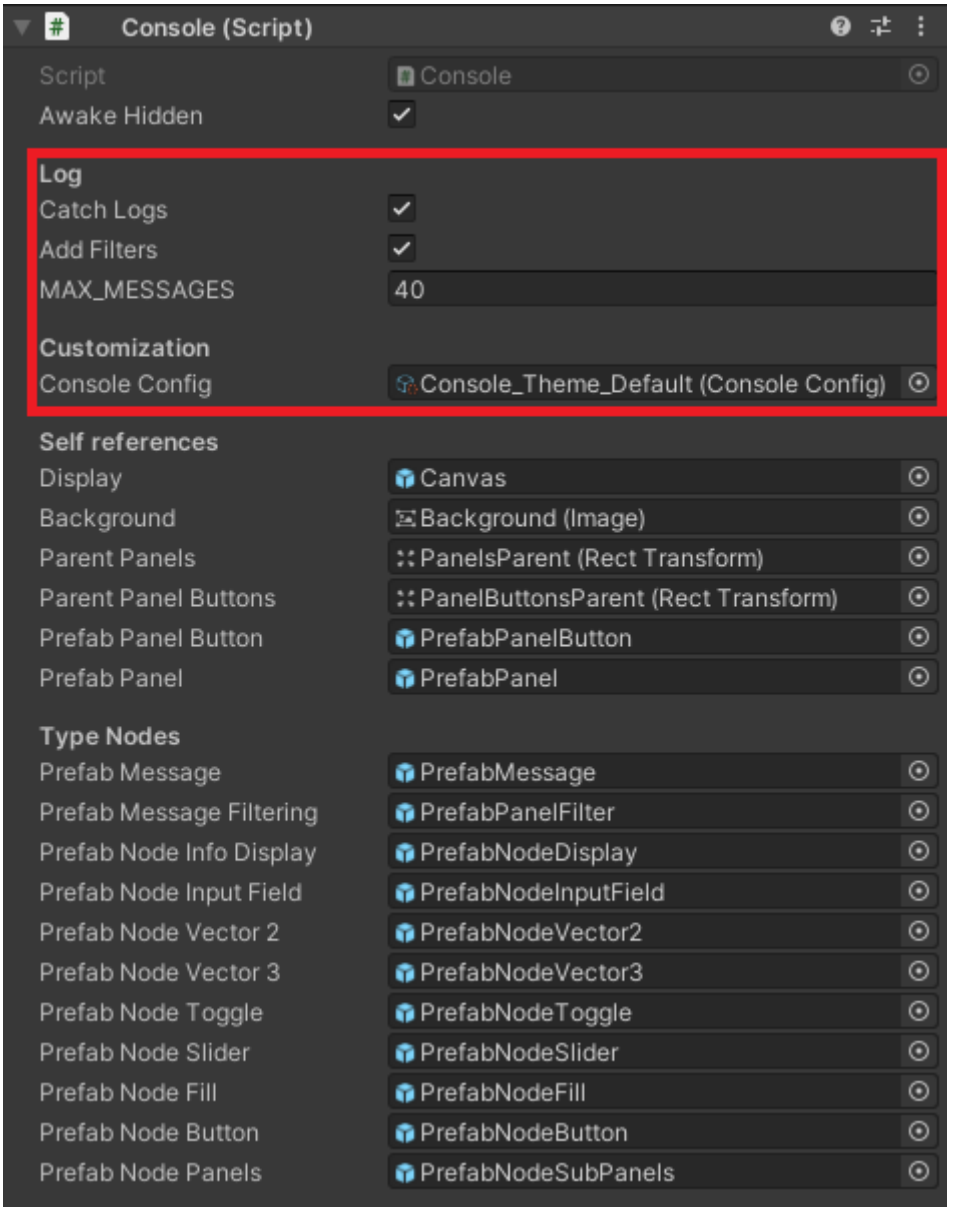
This asset is not meant for the final "release" version/build of your game, it is somewhat heavy because it's based on UI elements, consider removing it from the game, especially if the system is struggling to run the application.

How to use

1. Drag and drop the prefab TABBB/Console/Prefabs/**Console** into your scene.



2. Change its configuration as you like, variables to you might be interested in:
   - **catchLogs**: Do we create the generic log panel?
   - **addFilters**: Add filter options? (log type & message content)
   - **MAX_MESSAGES**: Size of the message pool
   - **consoleConfig**: Configuration of the colors and font



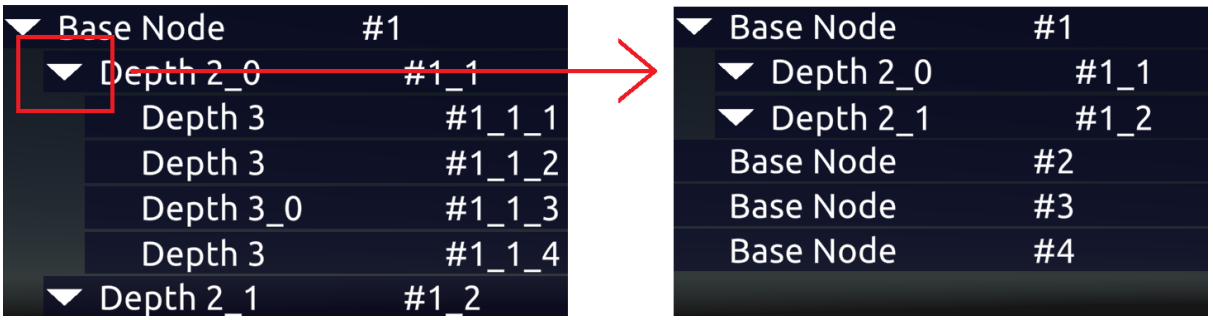
3. Create a **Panel** from your script

```
Panel panel = Console.I.AddPanel("MyPanel");
```

4. Add **Nodes** to the **Panel** and handle them

```
panel.AddInfo("Fixed Value", "Hello World");
panel.AddInputField("Input Field", new InputValue("Initial"), InputFieldCallback);
panel.AddToggle("Toggle", false, ToggleCallback);
panel.AddSlider("Slide", -2f, 4f, 0f, SliderCallback);
panel.AddFillBar("Fill Bar", 0f, 100f, 50);
panel.AddButton("Button", "Press", ButtonCallback);
panel.AddVector2("Vector2", new Vector2(Random.value, Random.value), Vector2Callback);
panel.AddVector3("Vector3", new Vector3(Random.value, Random.value), Vector3Callback);
```

To create nested **Nodes**, instead of calling any of the **Node** creation methods from the **Panel** object, call them from a **Node** object, when doing so, you can expand and collapse the child nodes with the dropdown icon that will appear at the beginning of the **Node** entry.



Check the scripts **ConsoleDemo** and **ConsoleTests** in the Demo folder to see many examples on how to handle **Nodes** of multiple types.

How to create your own custom Nodes

1. Create your information holder class

```
public class MyInfoHolderClass {
    public string label;
    public Color color;
    public int number;
    public MyInfoHolderClass(string label, Color color, int number) {
        this.label = label;
        this.color = color;
        this.number = number;
    }
}
```

2. Create a new **Node** script that inherits **ConsoleEntryNode**

```
using TABBB.Tools.Console;
using UnityEngine;

public class ConsoleCustomNodeExample : ConsoleEntryNode {

}
```

3. Override the methods **SetNode** and **UpdateValue** (SetUp too if you want to change the visuals in the same way it's done in all the **Nodes**)
    a. If you want to assign an action to a Node, you need to pass a callback to the AddNode method, you also need to call the method SetValueFromConsole

```
    // Copy this changing your actual class name to hold a reference to
    // the node of its specific type
    public new ValueNode<MyInfoHolderClass> node;
    protected override void SetNode(Node node) {
        base.SetNode(node);
        this.node = node as ValueNode<MyInfoHolderClass>;
    }

    public TMP_Text myTextComponent;
    protected override void SetUp() {
        base.SetUp();
        myTextComponent.color = Console.Foreground;
        myTextComponent.font = Console.Font;
    }
    public override void UpdateValue() {
        myTextComponent.color = node.value.color;
        myTextComponent.text = node.value.label + "[" + node.value.number + "]";
    }
```

4. (Optional) Create an extension method to facilitate its creation process

```
public static class MyCustomNodeExtension {
    public static ValueNode<MyInfoHolderClass> AddCustomNode(this NodeHolder nodeHolder,
        string name, string label, Color color, int number) {
        return Console.I.AddNode(nodeHolder, name, new MyInfoHolderClass(label, color, number));
    }
}
```

5. Create the Prefab object for your Node script and add it to the Console through **AddPrefabNode**

Check any **ConsoleEntryNode** script and their associated **Prefabs** to familiarize yourself with their structure if you want to implement your own, there's plenty of examples.