Voltage UI

Contents

1 Basics

1.1 Wild vs Stored

On Voltage there are two ways of constructing an interface. Stored construction style is performed on the Init phase, the stored elements can’t be deleted unless they are overwritten. Wild construction style is the more classic style of constructing an interface in Unity. Wild elements and areas will be added, drawn and destroyed each Repaint. This allows to create interfaces that can change over time.

1.2 Voltage Styles

Voltage provides a way to edit custom GUI Styles, the Style Editor window. Once you have set up your styles you can retrieve them using the class VoltageStyles.

1.3 Voltage Phases

Voltage Init

This step is executed when the window is opened or the editor recompiles. Here is where you would want to initialize your fields and areas.

On this section is where stored elements are added. Do not add wild elements here, they will be lost.

Voltage GUI

The method Voltage GUI is where the interface is constructed.

On this section is were wild elements are added. Do not add stored elements here, they won’t wipe out after drawn and will start to accumulate each Repaint.

Voltage Draw

This step is internal. Here is when the interface is actually displayed, and it is executed after the interface has been constructed. Here is also where all fields are updated and their values are changed.

Voltage Serialization

On the serialization step is when you should extract and use the values from your fields.

2 Areas

Areas are used to hold and draw elements. They can be nested, and can orient their elements horizontally or vertically.

Areas need to be started and ended on the VoltageGUI phase. Areas with more than one section have special actions to change between their subareas so you can add elements to each of them. All multi section areas, like SplitArea or TabArea, start with a base weight area to which you add another area on top to replace it.

2.1 Weight Area

Weight areas work by equally distributing the available space between all elements according to their weight. For example, an element with a weight of 2 will be twice as large as an element with a weight of 1.

Usage

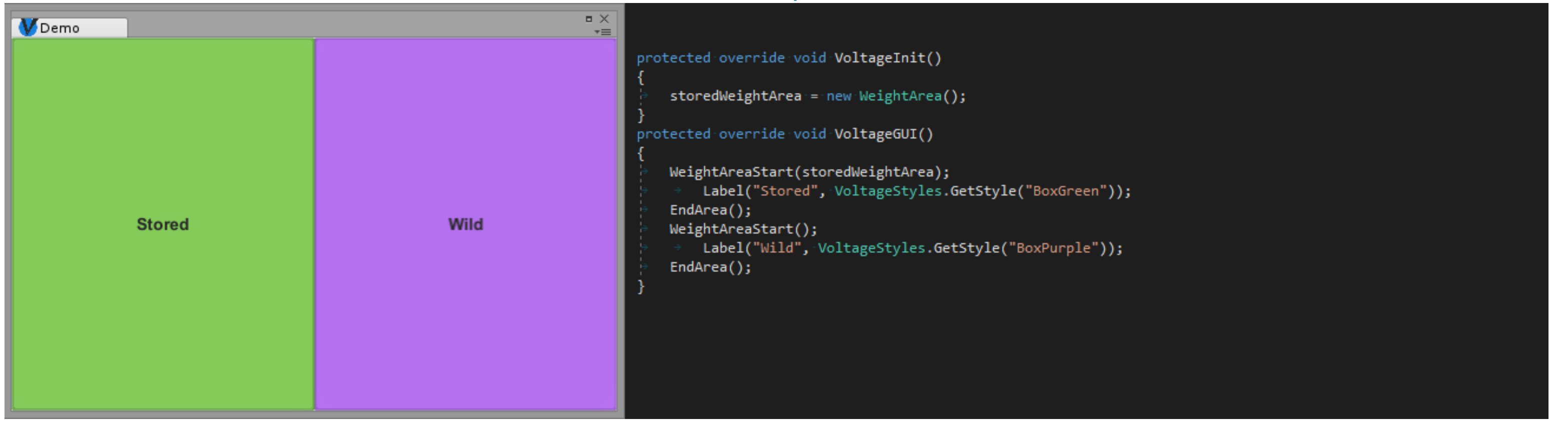
VoltageInit

Constructor	

VoltageGUI

WeightAreaStart	*Starts a WeightArea*

Example



```
protected override void VoltageInit()
{
    storedWeightArea = new WeightArea();
}
protected override void VoltageGUI()
{
    WeightAreaStart(storedWeightArea);
        Label("Stored", VoltageStyles.GetStyle("BoxGreen"));
    EndArea();
    WeightAreaStart();
        Label("Wild", VoltageStyles.GetStyle("BoxPurple"));
    EndArea();
}
```

2.2 Stream Area

Stream areas organize their elements by fitting them one next to the other with the minimum size possible. If there are any elements on a stream area set as flex elements then they will be sharing the rest of the space available according to their weights.

Usage

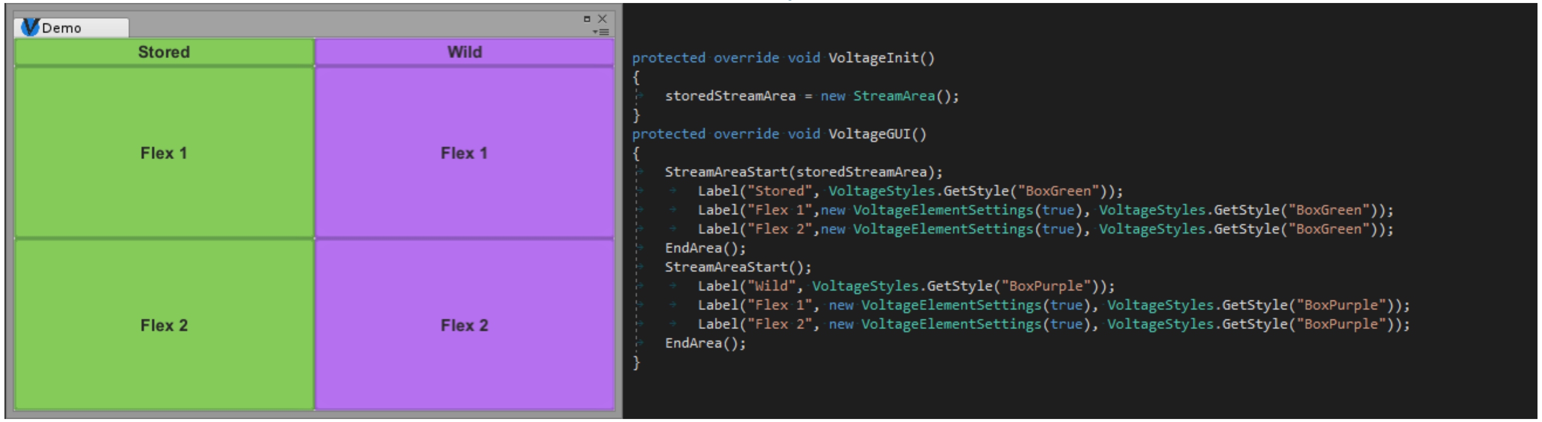
VoltageInit

Constructor	

VoltageGUI

StreamAreaStart	*Starts a StreamArea*

Example



```
protected override void VoltageInit()
{
    storedStreamArea = new StreamArea();
}
protected override void VoltageGUI()
{
    StreamAreaStart(storedStreamArea);
        Label("Stored", VoltageStyles.GetStyle("BoxGreen"));
        Label("Flex 1",new VoltageElementSettings(true), VoltageStyles.GetStyle("BoxGreen"));
        Label("Flex 2",new VoltageElementSettings(true), VoltageStyles.GetStyle("BoxGreen"));
    EndArea();
    StreamAreaStart();
        Label("Wild", VoltageStyles.GetStyle("BoxPurple"));
        Label("Flex 1", new VoltageElementSettings(true), VoltageStyles.GetStyle("BoxPurple"));
        Label("Flex 2", new VoltageElementSettings(true), VoltageStyles.GetStyle("BoxPurple"));
    EndArea();
}
```

2.3 Split Area

Split area lets you define an area divided in two, which division can be resized. Split areas can only be stored so the division value can persist, their fields can be wild.

Usage

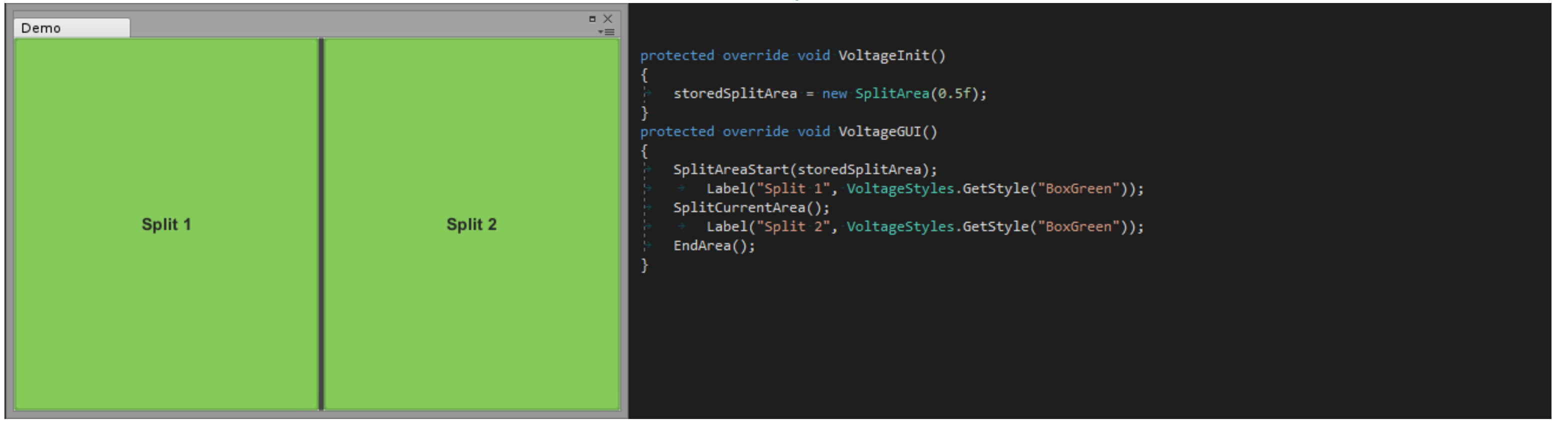
VoltageInit

Constructor	
SplitArea.Split	*Makes the second section active to add stored elements*

VoltageGUI

SplitAreaStart	*Starts a SplitArea*
SplitCurrent	*Makes the second section of the current SplitArea active to add wild elements*

Example



```
protected override void VoltageInit()
{
    storedSplitArea = new SplitArea(0.5f);
}
protected override void VoltageGUI()
{
    SplitAreaStart(storedSplitArea);
        Label("Split 1", VoltageStyles.GetStyle("BoxGreen"));
    SplitCurrentArea();
        Label("Split 2", VoltageStyles.GetStyle("BoxGreen"));
    EndArea();
}
```

2.4 Foldout Area

A foldout area can be compacted to hide the elements inside of itself. It's useful for separating sections that are relatively unrelated. Foldout areas can only be stored so the foldout value can persist, their fields can be wild.

Usage

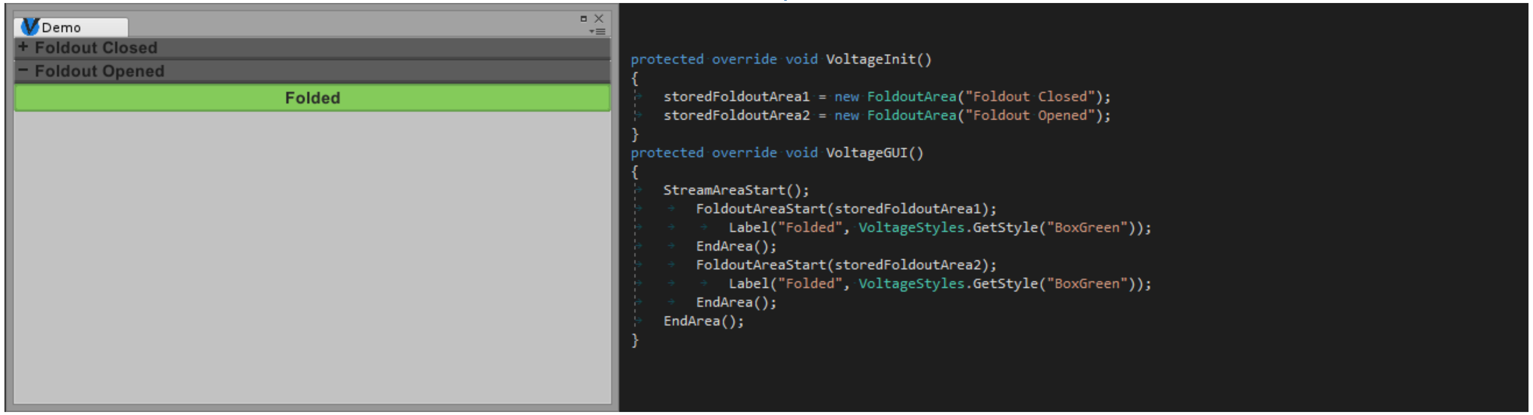
VoltageInit

Constructor	

VoltageGUI

FoldoutAreaStart	*Starts a FoldoutArea*

Example



```
protected override void VoltageInit()
{
    storedFoldoutArea1 = new FoldoutArea("Foldout Closed");
    storedFoldoutArea2 = new FoldoutArea("Foldout Opened");
}
protected override void VoltageGUI()
{
    StreamAreaStart();
        FoldoutAreaStart(storedFoldoutArea1);
            Label("Folded", VoltageStyles.GetStyle("BoxGreen"));
        EndArea();
        FoldoutAreaStart(storedFoldoutArea2);
            Label("Folded", VoltageStyles.GetStyle("BoxGreen"));
        EndArea();
    EndArea();
}
```

2.5 Tab Area

Tab areas have multiple areas inside and will let you choose which area to show. Tab areas can only be stored so the tab selected can persist, their field can be wild.

Usage

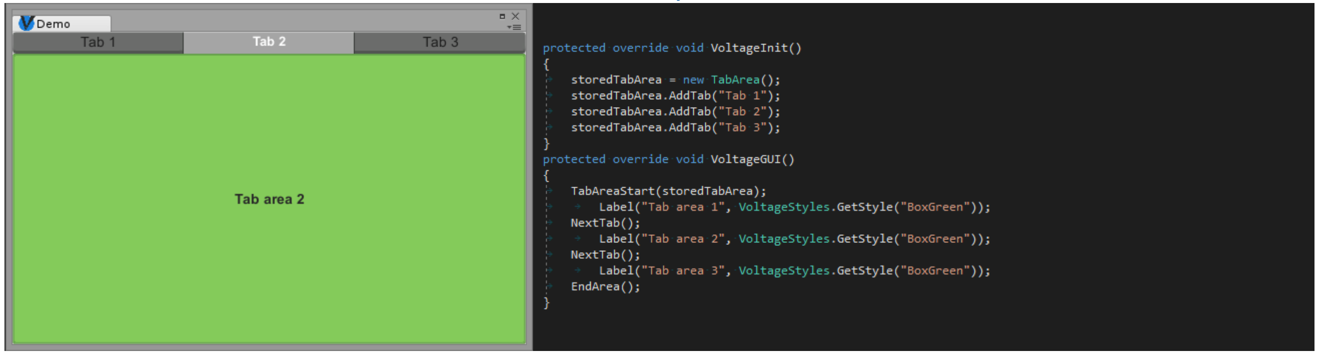
VoltageInit

Constructor	
TabArea.AddTab	*Adds a new tab*
TabArea.NextTab	*Makes the next tab active to add stored elements*

VoltageGUI

TabAreaStart	*Starts a TabArea*
NextTab	*Makes the next tab of the current TabArea active to add wild elements*

Example



```
protected override void VoltageInit()
{
    storedTabArea = new TabArea();
    storedTabArea.AddTab("Tab 1");
    storedTabArea.AddTab("Tab 2");
    storedTabArea.AddTab("Tab 3");
}
protected override void VoltageGUI()
{
    TabAreaStart(storedTabArea);
        Label("Tab area 1", VoltageStyles.GetStyle("BoxGreen"));
    NextTab();
        Label("Tab area 2", VoltageStyles.GetStyle("BoxGreen"));
    NextTab();
        Label("Tab area 3", VoltageStyles.GetStyle("BoxGreen"));
    EndArea();
}
```