Loxodon Framework UGUI

(中文版)

Developed by Clark

Requires Unity 2021.3 or higher.

This is a text formatting plugin modified based on the official C# library. By extending the AppendFormat function of StringBuilder, it aims to avoid garbage collection (GC) when concatenating strings or converting numbers to strings. This optimization is particularly beneficial in scenarios with high-performance requirements.

Furthermore, the plugin extends Unity's Unity GUI (UGUI) by introducing two new text controls: TemplateText and FormattableText. These controls support the data binding features of MVVM, allowing the binding of ViewModel or value-type objects to the controls. This approach eliminates the need for boxing and unboxing of value-type objects, thus maximizing the optimization of garbage collection (GC).

It's worth noting that using the controls TemplateTextMeshPro or FormattableTextMeshProUGUI from Loxodon.Framework.TextMeshPro can further reduce garbage collection (GC), achieving a completely GC-free update of the game view.

Installation

Install via OpenUPM (recommended)

OpenUPM can automatically manage dependencies, it is recommended to use it to install the framework.

Requires nodejs's npm and openupm-cli, if not installed please install them first.

```
# Install openupm-cli,please ignore if it is already installed.
npm install -g openupm-cli

#Go to the root directory of your project
cd F:/workspace/New Unity Project

#Install loxodon-framework-textformatting
openupm add com.vovgou.loxodon-framework-textugui
```

Install via Packages/manifest.json

Modify the Packages/manifest.json file in your project, add the third-party repository "package.openupm.com"'s configuration and add "com.vovgou.loxodon-framework-textugui" in the "dependencies" node.

Installing the framework in this way does not require nodejs and openm-cli.

```
{
  "dependencies": {
    ...
    "com.vovgou.loxodon-framework-textugui": "2.6.5"
  },
  "scopedRegistries": [
    {
      "name": "package.openupm.com",
      "url": "https://package.openupm.com",
      "scopes": [
        "com.vovgou",
        "com.openupm"
      ]
    }
  ]
```

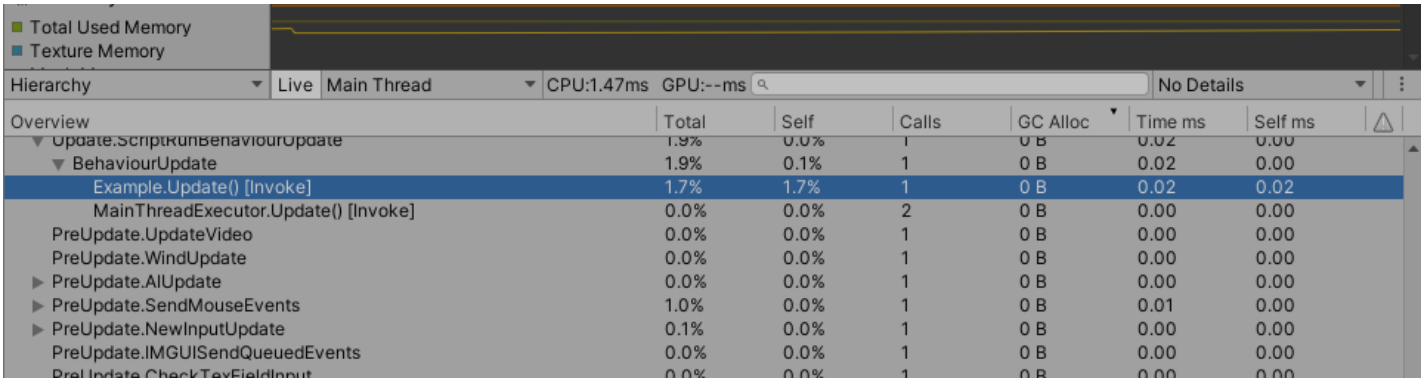
```
}
```

Quick Start

StringBuilder.AppendFormat

This plugin enhances the AppendFormat<>() function of StringBuilder. It provides support for multiple generic parameters of different types or generic array parameters. When these parameters are of numeric types, DateTime, or TimeSpan, using them to concatenate strings eliminates the need for value type boxing or unboxing. Consequently, converting numeric types to String during string concatenation does not generate garbage collection (GC). See the example below for usage details.

```
using System;
using System.Text;
using UnityEngine;
using Loxodon.Framework.TextFormatting;//make sure to first import the required namespace
public class Example : MonoBehaviour
{
    StringBuilder builder = new StringBuilder();
    void Update()
    {
        builder.Clear();
        builder.AppendFormat<DateTime,int>("Now:{0:yyyy-MM-dd HH:mm:ss} Frame:{0:D6}", DateTime.Now,Time.frameCount);
        builder.AppendFormat<float>("{0:f2}", Time.realtimeSinceStartup);
    }

}
```



FormattableText

This control extends UnityEngine.UI.Text, providing support for string formatting and data binding. The AsParameters<>() function of the FormattableText control can be used to convert to a set of generic parameters, supporting 1-4 different parameters or a generic array. It enables binding with ViewModel. With this plugin, string and array concatenation are garbage collection (GC)-free. However, due to the Text component's requirement for a string and its limited support for string assignment, there is GC allocation when calling StringBuilder.ToString(). (It is recommended to install the Loxodon.Framework.TextMeshPro plugin and use FormattableTextMeshProUGUI instead of FormattableText for a completely GC-free experience.)

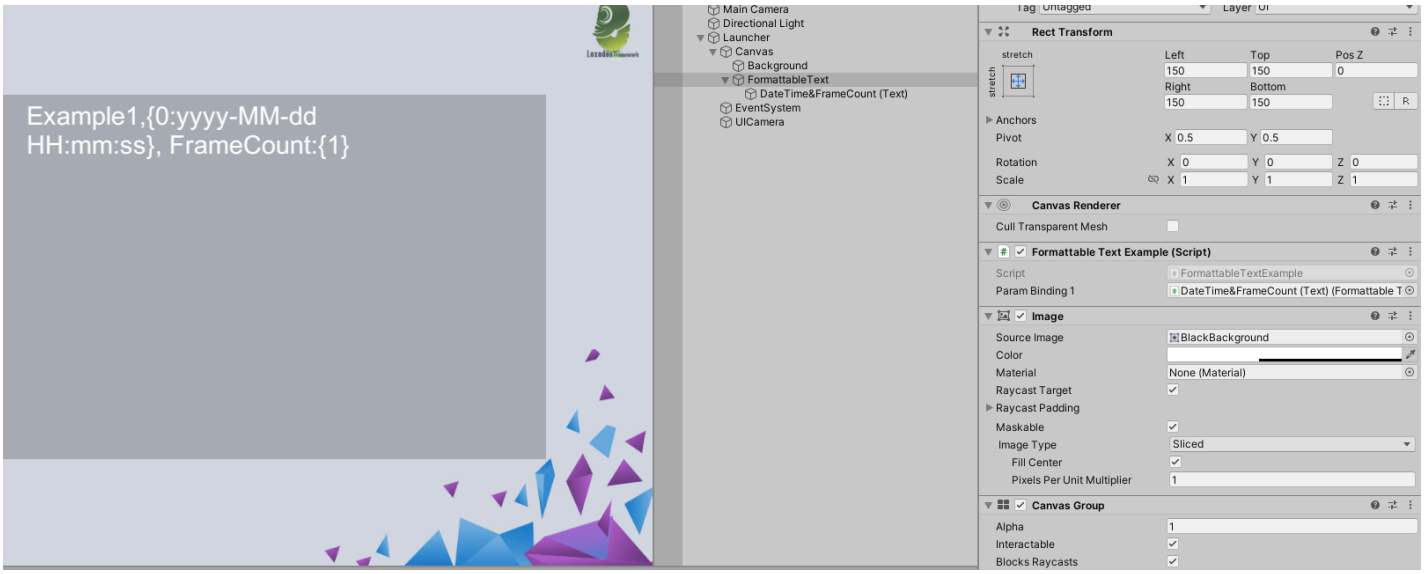
Usage 1: Utilize the FormattableText.AsParameters<DateTime, int>() method to obtain a parameter set of GenericParameters<DateTime, int>, then bind it to the view model.

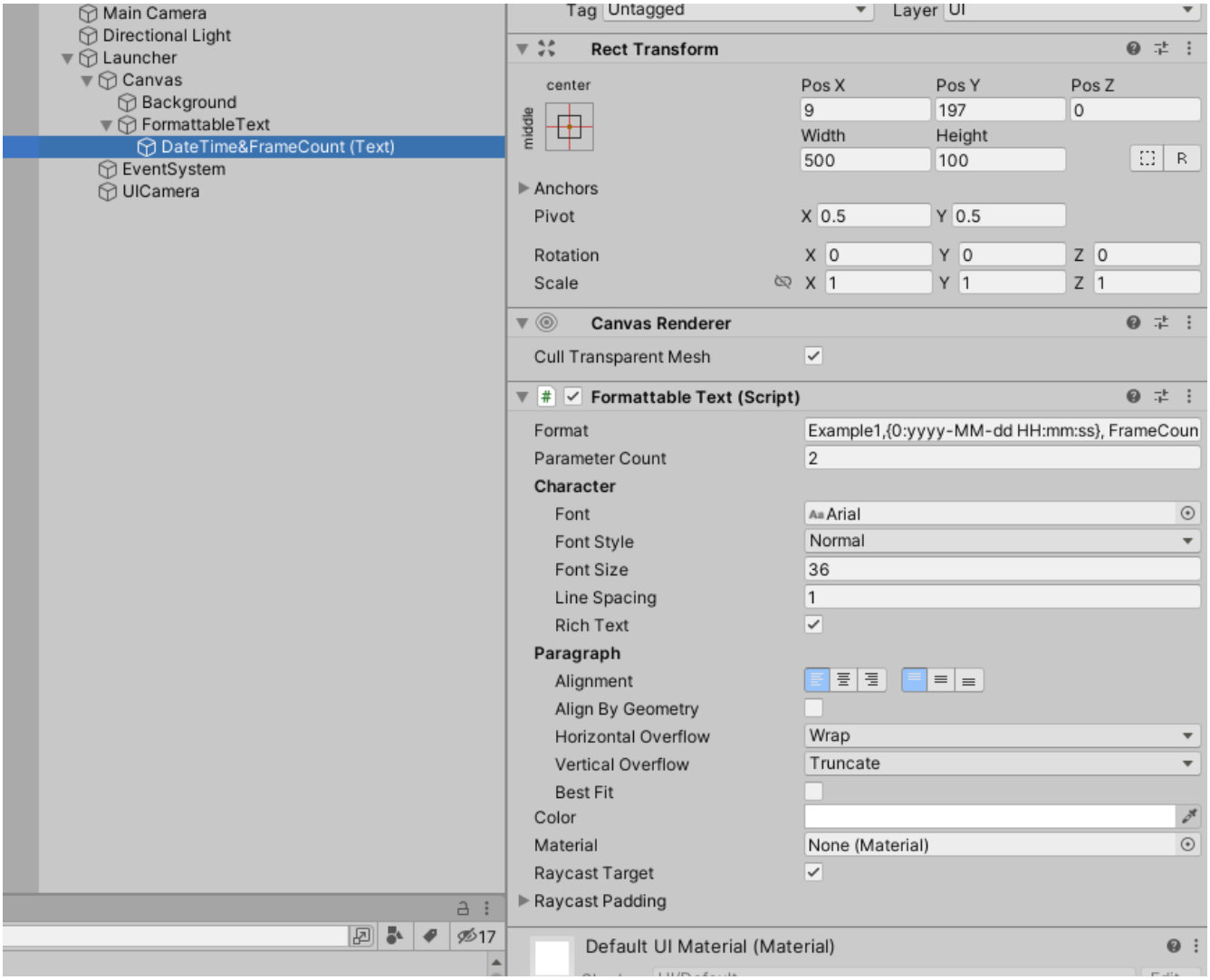
```
public class TemplateTextAndFormattableTextExample : MonoBehaviour
{
    public FormattableText paramBinding1;

    private ExampleViewModel viewModel;

    private void Start()
    {
        ApplicationContext context = Context.GetApplicationContext();
        IServiceContainer container = context.GetContainer();
```

```
        BindingServiceBundle bundle = new BindingServiceBundle(context.GetContainer());
        bundle.Start();

        BindingSet<TemplateTextAndFormattableTextExample, ExampleViewModel> bindingSet = this.CreateBindingSet<TemplateTextAndForm

        //Create a parameter collection using the AsParameters<P1, P2, ...>()
        //format:The format is identical to the formatting parameters of string.Format(). For example: DateTime - Example1, {0:yyy
        bindingSet.Bind(paramBinding1.AsParameters<DateTime, int>()).For(v => v.Parameter1).To(vm => vm.Time);
        bindingSet.Bind(paramBinding1.AsParameters<DateTime, int>()).For(v => v.Parameter2).To(vm => vm.FrameCount);

        bindingSet.Build();

        this.viewModel = new ExampleViewModel();
        this.viewModel.Time = DateTime.Now;
        this.viewModel.FrameCount = 1;
        this.SetDataContext(this.viewModel);
    }
}
```

Usage 2: In the script FormattableTextExample, define a variable of type GenericParameters<DateTime, int> as a parameter set. In the UnityEditor, drag and drop the FormattableText onto the property paramBinding1 in the script shown below. Subsequently, bind it to the view model.

```
public class FormattableTextExample : MonoBehaviour
{
    public GenericParameters<DateTime,int> paramBinding1;

    private ExampleViewModel viewModel;

    private void Start()
    {
        ApplicationContext context = Context.GetApplicationContext();
        IServiceContainer container = context.GetContainer();
        BindingServiceBundle bundle = new BindingServiceBundle(context.GetContainer());
        bundle.Start();

        BindingSet<FormattableTextExample, ExampleViewModel> bindingSet = this.CreateBindingSet<FormattableTextExample, ExampleVie

        //format:The format is identical to the formatting parameters of string.Format(). For example: DateTime - Example1, {0:yyy
        bindingSet.Bind(paramBinding1).For(v => v.Parameter1).To(vm => vm.Time);
        bindingSet.Bind(paramBinding1).For(v => v.Parameter2).To(vm => vm.FrameCount);

        bindingSet.Build();

        this.viewModel = new ExampleViewModel();
        this.viewModel.Time = DateTime.Now;
        this.viewModel.FrameCount = 1;
        this.SetDataContext(this.viewModel);
    }
}
```





TemplateText

This control surpasses the capabilities of a formatted text control, providing enhanced functionality and ease of use. It supports binding a ViewModel object or sub-object to the TemplateText.Data property. The template control comes equipped with built-in path resolution and data binding features, allowing automatic binding of properties from the Data object using the text template.

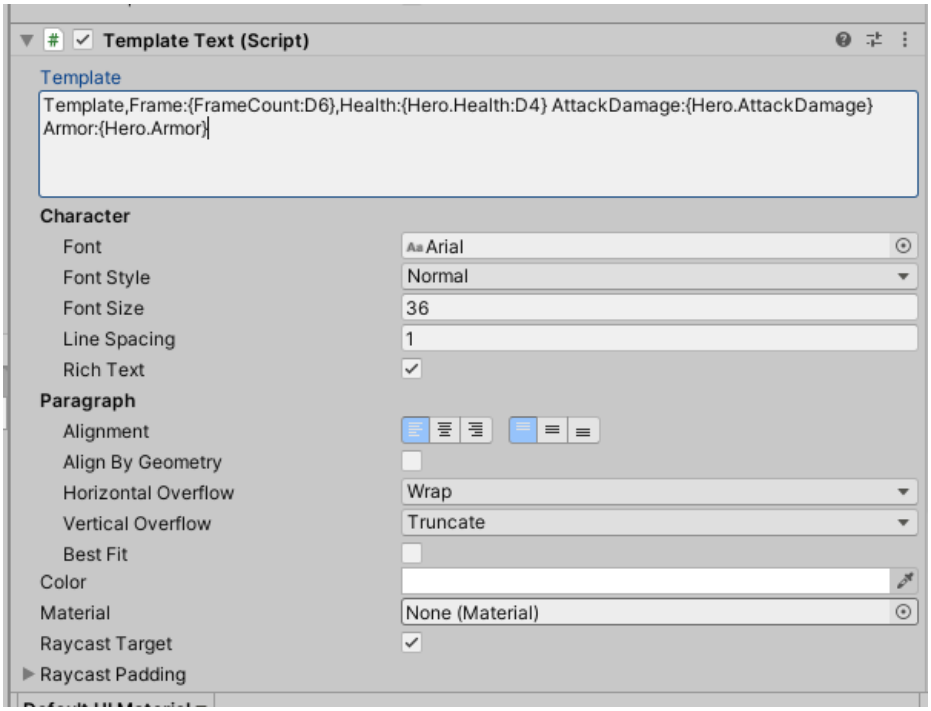
Template Format: Template, Frame: {FrameCount:D6}, Health: {Hero.Health:D4}, AttackDamage: {Hero.AttackDamage}, Armor: {Hero.Armor}

Here, FrameCount and Hero are properties bound to the Data object, while Health, AttackDamage, and Armor are properties of the Hero object. The D6 after FrameCount represents a numeric format parameter for frame count.

Similar to the previous case, garbage collection (GC) occurs only when calling StringBuilder.ToString(). (It is recommended to install Loxodon.Framework.TextMeshPro and use TemplateTextMeshProUGUI instead of TemplateText for a completely GC-free experience.)

```
public class TemplateTextAndFormattableTextExample : MonoBehaviour
{
    public TemplateText template;

    private ExampleViewModel viewModel;

    private void Start()
    {
        ApplicationContext context = Context.GetApplicationContext();
        IServiceContainer container = context.GetContainer();
        BindingServiceBundle bundle = new BindingServiceBundle(context.GetContainer());
        bundle.Start();

        BindingSet<TemplateTextAndFormattableTextExample, ExampleViewModel> bindingSet = this.CreateBindingSet<TemplateTextAndForm

        bindingSet.Bind(template).For(v => v.Template).To(vm => vm.Template);//The Template property of the template control can b
        bindingSet.Bind(template).For(v => v.Data).To(vm => vm);
        bindingSet.Build();

        this.viewModel = new ExampleViewModel();
        this.viewModel.Template = "Template,Frame:{FrameCount:D6},Health:{Hero.Health:D4} AttackDamage:{Hero.AttackDamage} Armor:{
        this.viewModel.Time = DateTime.Now;
        this.viewModel.TimeSpan = TimeSpan.FromSeconds(0);
        this.viewModel.Hero = new Hero();
        this.SetDataContext(this.viewModel);
    }
}

public class ExampleViewModel : ObservableObject
{
    private DateTime time;
    private TimeSpan timeSpan;
    private string template;
    private int frameCount;
    private Hero hero;
    public DateTime Time
    {
        get { return this.time; }
        set { this.Set(ref time, value); }
    }

    public TimeSpan TimeSpan
    {
        get { return this.timeSpan; }
        set { this.Set(ref timeSpan, value); }
    }

    public int FrameCount
    {
        get { return this.frameCount; }
        set { this.Set(ref frameCount, value); }
    }

    public string Template
    {
        get { return this.template; }
        set { this.Set(ref template, value); }
    }

    public Hero Hero
```

```
    {
        get { return this.hero; }
        set { this.Set(ref hero, value); }
    }
}

public class Hero : ObservableObject
{
    private float attackSpeed = 95.5f;
    private float moveSpeed = 2.4f;
    private int health = 100;
    private int attackDamage = 20;
    private int armor = 30;

    public float AttackSpeed
    {
        get { return this.attackSpeed; }
        set { this.Set(ref attackSpeed, value); }
    }

    public float MoveSpeed
    {
        get { return this.moveSpeed; }
        set { this.Set(ref moveSpeed, value); }
    }

    public int Health
    {
        get { return this.health; }
        set { this.Set(ref health, value); }
    }

    public int AttackDamage
    {
        get { return this.attackDamage; }
        set { this.Set(ref attackDamage, value); }
    }

    public int Armor
    {
        get { return this.armor; }
        set { this.Set(ref armor, value); }
    }

}
```



Contact Us

Email: yangpc.china@gmail.com
Website: https://vovgou.github.io/loxodon-framework/
QQ Group: 622321589