



Loxodon Framework

MVVM Framework for Unity3D (C# & XLua)

Version 2.0.0

Developed by Clark

Chinese Manual by Clark; and translated by Tien Nguyen and Xu Rongfeng

This is a preliminary translated manual according Chinese version, which may includes translation or express errors, if any divergences Chinese shall prevail. You are welcome to join us to provide any supports.

Email: yangpc.china@gmail.com

Table of Contents

- [Table of Contents](#)
- [Overview](#)
- [Installation](#)
 - [Install via OpenUPM \(recommended\)](#)
 - [Install via modified Packages/manifest.json](#)
 - [Install via git URL](#)
 - [Install via *.unitypackage file](#)
 - [Import example](#)
- [Official Plugin\(optional\)](#)
- [Lua plugin installation \(optional\)](#)
 - [XLua Installation](#)
 - [Import Lua plugin](#)
 - [Import the examples](#)
- [Quick start](#)
 - [C# example](#)
 - [Lua example](#)
- [Features Introduction](#)
 - [Context](#)
 - [Application Context](#)
 - [Player Context](#)
 - [Other Contexts](#)
 - [Service Container](#)
 - [Service Registry\(IServiceRegistry\)](#)
 - [Service Locator\(IServiceLocator\)](#)
 - [Service Bundle\(IServiceBundle\)](#)
 - [Application configuration\(Preference\)](#)
 - [Configuration\(Properties File\)](#)
 - [Supported Data Types](#)
 - [Array Delimiter](#)
 - [Configuration File Combine](#)
 - [A subset of Configuration File](#)
 - [Example of Configuration File](#)
 - [Internationalization and localization](#)
 - [Directory Structure](#)
 - [Format of the configuration file](#)
 - [XML special characters](#)
 - [Numeric types supported by XML](#)
 - [Generate C# Code](#)
 - [Localized view components](#)
 - [Localization Data Binder\(LocalizedDataBinder\)](#)
 - [Data provider\(IDataProvider\)](#)
 - [Get the device's current language](#)
 - [Usage example](#)
 - [Localization plugin supporting CSV format](#)
 - [Logging system](#)
 - [StreamingAssets dirctory file reading \(Android\)](#)
 - [Thread/Coroutine asynchronous results and asynchronous tasks](#)
 - [AsyncResult](#)
 - [ProgressResult](#)
 - [AsyncTask](#)
 - [ProgressTask](#)
 - [CoroutineTask](#)
 - [Async & Await](#)
 - [async & await in C#](#)
 - [Coroutine from Task to Unity](#)
 - [async & await in Lua](#)

- C# call async function of Lua
 - try/catch/finally of Lua
- Thread/Coroutine Executor
 - Executors
 - Scheduled Task Executor(IScheduledExecutor)
 - Interceptable Enumerator(InterceptableEnumerator)
- Message System(Messenger)
- Observables
- Databinding
 - Data binding example
 - Binding mode
 - Type converter(IConverter)
 - Binding type
 - Command Parameter
 - Scope Key
 - Binding life cycle
 - Accessors for properties and fields
- UI framework
 - Variables
 - UI view locator (UIViewLocator)
 - Animations
 - UI controls
 - Views, windows, and window managers
 - Interaction Request
 - Interaction Action
 - Collection and list view binding
 - Data binding and asynchronous loading sprites
- Lua
 - Modules and inheritance
 - Lua's ObserableObject
 - Coroutines in Lua using Unity
 - Using the logging system in Lua
 - Lua precompiled tools
 - Lua loader
 - Example
 - Expand other encryption way
- Layered architecture
 - View
 - Application layer (Service)
 - Domain Model
 - Infrastructure
- Contact information

Overview

Requires Unity 2018.4 or later

LoxodonFramework is a light Model-View-ViewModel framework (MVVM) . It is specially designed for Unity3D game development and refers to the MVVM design of WPF and Android. It provides several basic components, such as data binding of View and ViewModel, localization, a light service container, configuration component, thread component, application context and player context, asynchronous thread and coroutine task components etc. It also provides a framework for UI views. All code is designed based on the concept of object-oriented and interface-oriented, and almost all functions can be customized. In addition, performance optimization has been performed in the data binding part. It provides binding method on the JIT supported platform and reflecting method as default on the JIT unsupported platform, but it can be optimized by injecting the delegate function.

LoxodonFramework is developed through C# language, and also supports XLua development. Using Lua develop game project provides hot update after install the XLua plugin.

This plugin is compatible with MacOSX, Windows, Linux, UWP, WebGL, IOS and Android etc. It's a completely open source plugin.

Tested platforms:

- **PC/Mac/Linux** (.Net4.x; .Net Standard 2.0; IL2CPP)
- **IOS** (.Net4.x; .Net Standard 2.0; IL2CPP)
- **Android** (.Net4.x; .Net Standard 2.0; IL2CPP)
- **UWP(window10)** (.Net4.x; .Net Standard 2.0; IL2CPP)
- **WebGL**(.Net4.x;.Net Standard 2.0; IL2CPP)

Key features

- Supports multiple platforms, high scalability, and interface-oriented development;
- Supports UGUI and FaiyGUI;
- Supports XLua, develops using Lua scripts completely(optional);
- Supports async & await by both C# and Lua;
- Supports try & catch & finally by Lua;
- Supports asynchronous results and asynchronous tasks of threads and coroutines, adopting Future/Promise design model;
- Provides multi-threaded components, thread switching components and timer executors;
- Provides a messaging system that supports subscription and publishing;
- Provides encrypted configuration files, supports object access, custom type converters, and extended functions;
- Provides localization support, similar to Android's localization, supporting basic data types, arrays, and some value types of U3D;
- Supports global context and player context;
- Provides a service container to support registration and de-registration services;
- Provides universal UI controls such as AlertDialog, Loading, Toast, and supports custom appearance;
- Provides UI view control and management functions;
- Provides data binding function:
 - Field binding, only supports OneTime mode, as it cannot support change notification;
 - Property binding, support Two-Way binding, automatic notification at value modification;
 - Common dictionary, list binding, does not support change notification;
 - Supports C# event binding;
 - Supports Unity3D EventBase binding;
 - Supports binding of static class properties and Field;
 - Supports method binding (including static methods);
 - Supports command binding, which can conveniently control the effective and invalid state of the button through command binding;
 - Supports binding of observable attributes, dictionaries, and lists; supports change notifications; and UI display can change automatically after view model change;
 - Supports expression binding;
 - Supports binding of interactive requests and behaviors;
 - Supports type converter, you can convert the picture name to Sprite in the atlas;
 - You can customize and extend more binding types;

Installation

Loxodon.Framework keeps the orginal .unitypackage installation method since version 2.0, and add UPM installation method which requires Unity 2018.4.2 or later. New version also do some adjustments about Framework catalog structure and API, at the same time, import some new feature

such as async/await/task etc. Please refer to below notes before updating.

Installation notes: The third part warehouse is blocked by Unity which downloads from China zone, that shall get UPM installation fault. Unity China responses that it will be opened ASAP. Please adapt .unitypackage installation method while UPM installation fault, or use Unity which downloads from other zone.

Notes for updating from 1.x.x to 2.0

Please remove all existing files before updating from 1.x.x to 2.0, and follow the below sequence to install the new version. Tutouial and example can't be imported automatically, you can manually import them while it's necessary to you.

Loxodon.Framework.XLua and **Loxodon.Framework.Bundle** are still released through the traditional method.

Incompatible modification:

- Modified IUIViewLocator interface and implementation, please adjust it while inherite it's custom implementation.
- Modified IDataProvider interface and implementation of the localization module. If there is no custom class, it will not be affected.
- Multi-threading is provided in IAsyncTask and IProgressTask, but it's not supported in WebGL platform. It is not recommended to use them after version 2.0. They are changed to IAsyncResult and IProgressResult in new framework.
- Async/Await and Task are provided in the new API, and no longer [supports.NET](#) 2.0.
- Modified the function of Window, WindowManager and other classes, changed IAsyncTask to IAsyncResult.

Install via OpenUPM (recommended)

[OpenUPM](#) is a open-source UPM package warehouse, which supports the third-party UPM package and manage dependencies automatically. It is recommended to adapt it as installation method.

Requires [nodejs](#) and openupm-cli client through openupm method, please install them firstly.

```
#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
openupm add com.vovgou.loxodon-framework
```

Install via modified Packages/manifest.json

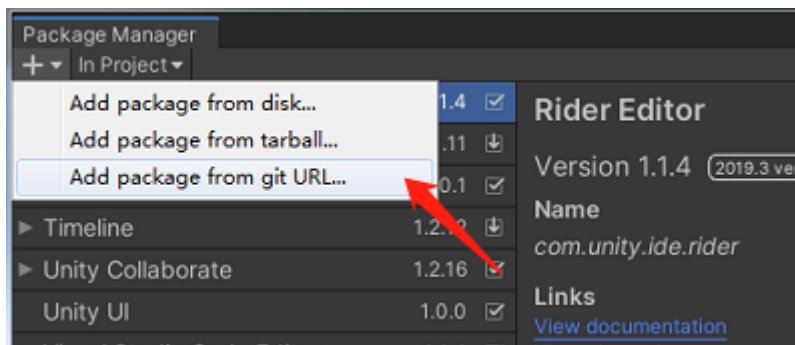
Modify the Packages/manifest.json file in your project, nodejs and openm-cli clients are not required. Find Packages/manifest.json file in your project root folder, add the third-party repository "[package.openupm.com](#)" configuration in "scopeRegistries" node and add "com.vovgou.loxodon-framework" in "dependencies" node. Save and switch to Unity window, then complete installation.

```
{
  "dependencies": {
    ...
    "com.unity.modules.xr": "1.0.0",
    "com.vovgou.loxodon-framework": "2.0.0-preview"
  },
  "scopedRegistries": [
    {
      "name": "package.openupm.com",
      "url": "https://package.openupm.com",
      "scopes": [
        "com.vovgou.loxodon-framework",
        "com.openupm"
      ]
    }
  ]
}
```

Install via git URL

Unity provides git URL installation method for version 2019.3.4f1 or later.

Add <https://github.com/vovgou/loxodon-framework.git?path=Loxodon.Framework/Assets/LoxodonFramework> to UPM manager as below picture, installation will be completed after waiting a moment.



Install via *.unitypackage file

Download [Loxodon.Framework2.x.x.unitypackage](#) and import it into your project.

- [AssetStore](#)
- [Releases](#)

Import example

- Import example by Package Manager for Untiy 2019 or later. Click "Import into Project" button and import example in Package Manager.
- For Unity 2018, you can find Examples.unitypackage and Tutorials.unitypackage in folder Packages/Loxodon Framework/Package Resources/, then double-click to import example.

Official Plugin(optional)

- [Loxodon Framework XLua](#)

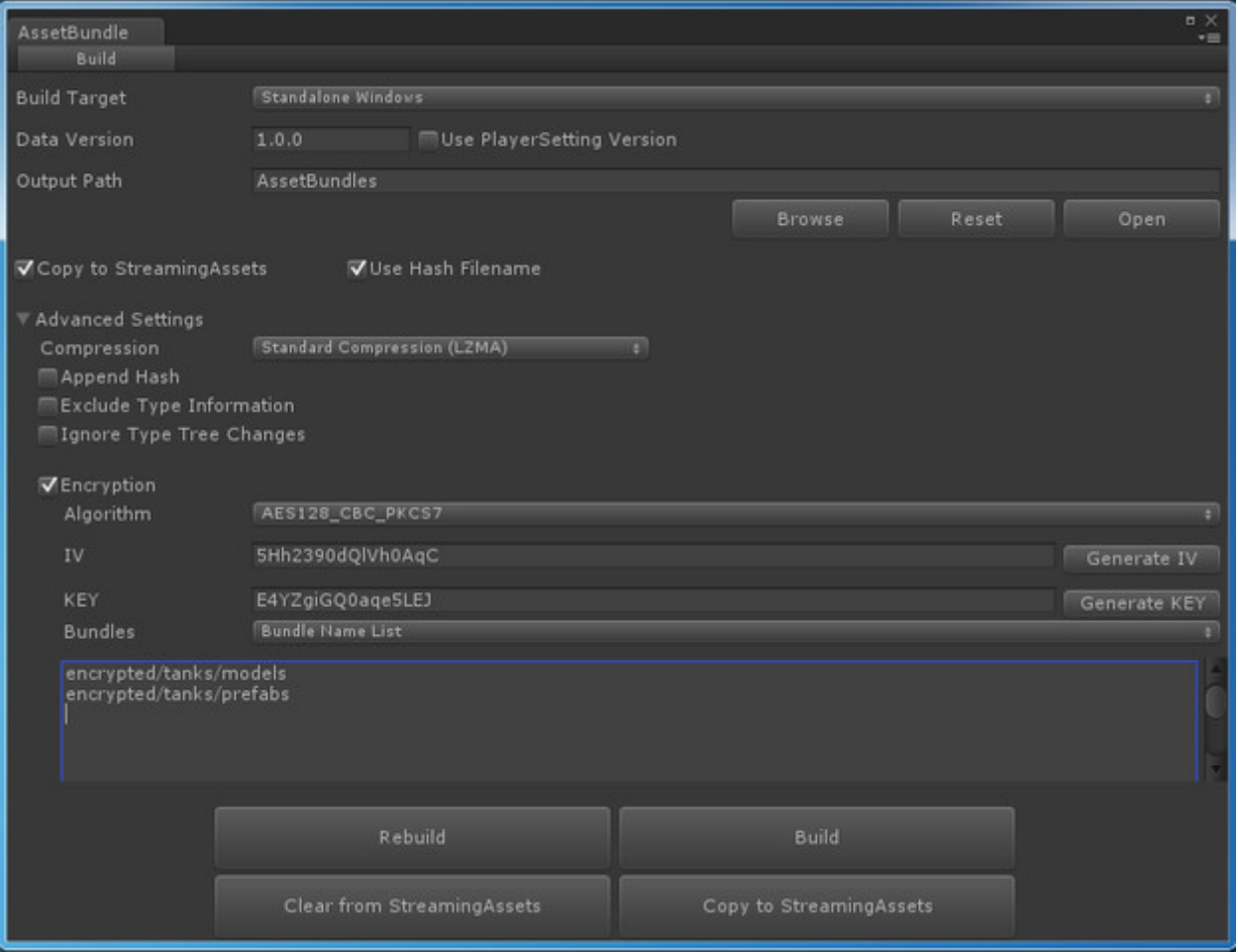
XLua plugin in Loxodon.Framework is a Lua MVVM framework, which supports mixed programming by Lua and C# in entire game. See the next chapter for installation steps or refer the documents in [Loxodon.Framework.XLua](#).

- [Loxodon Framework Localization For CSV](#)

Support localization file as csv file, requires Unity2018.4 or later.

- [Loxodon Framework Bundle](#)

Loxodon.Framework.Bundle is a tool for loading and managing AssetBundle, as well as an asset redundancy analysis tool for AssetBundle. It can automatically manage complex dependencies between AssetBundles, and it maintains dependencies between AssetBundles through reference counting. You can either pre-load an AssetBundle and manage its release yourself, or you can directly load the resource directly through the asynchronous resource loading function. The resource loading function will automatically find the AB package where the resource is located, automatically load AB, and automatically use it after use. Release AB. It also supports weak caching. If the object template is already in the cache, there is no need to reopen AB. It supports multiple loading methods, WWW loading, UnityWebRequest loading, File loading, etc. (on Unity 5.6 and above, please do not use WWW loader, it will produce memory spikes). It provides a package interface for AssetBundle and supports encrypted AB packages (only sensitive resources are recommended to be encrypted because it will affect performance). At the same time, it also bypasses some bugs in earlier versions of Unity3D, such as multiple coroutines loading the same resource concurrently, which will cause errors in the android system. Its redundancy analysis is performed by unpacking the AssetBundle, which is more accurate than analyzing the redundancy in editor mode.



- [Loxodon Framework FairyGUI](#)

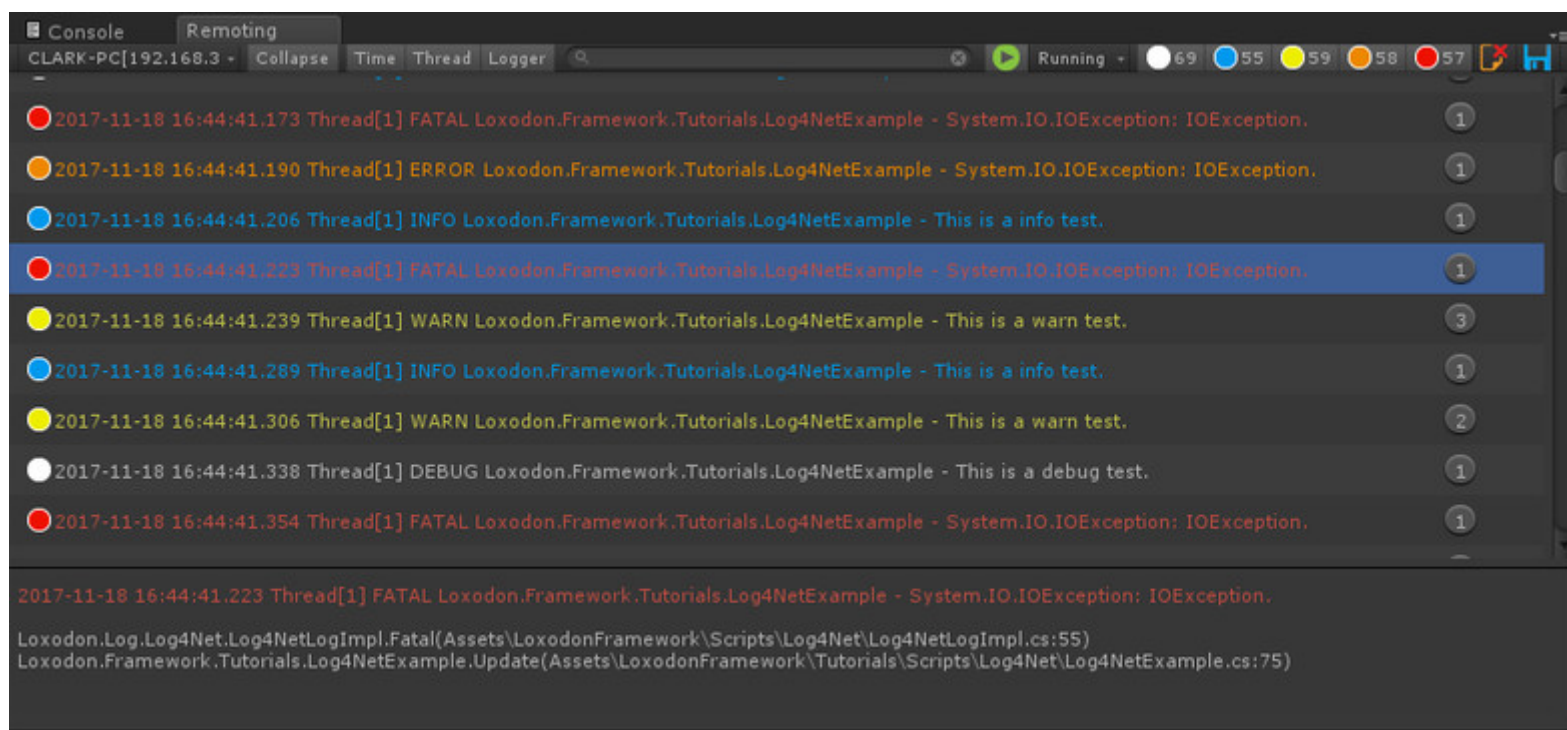
The framework already supports data binding for FairyGUI controls. Please download FairyGUI-unity and Loxodon Framework FairyGUI, then import them into your project.

Download them from below address.

- [FairyGUI-unity](#)
- [Loxodon Framework FairyGUI](#)

- [Loxodon Framework Log4Net](#)

Plug-in for printing logs in Unity using Log4Net, and support remote debugging in LAN.



- [Loxodon Framework Obfuscation](#)

It's a data type and memory confused plug-in; supports ObfuscatedByte, ObfuscatedShort, ObfuscatedInt, ObfuscatedLong, ObfuscatedFloat, ObfuscatedDouble; prevents memory modifier to modify game data; supports all calculation operators; byte, short, int, long, float, double can automatic conversion between types during using.

Float and Double are confused and converted to Int and Long to do and/or calculation, make sure not lose precision. Using unsafe code for type converting, concerns over performance.

Note: Please enable "Allow unsafe code" for Unity 2018 or later. example as below.

```
ObfuscatedInt length = 200;
ObfuscatedFloat scale = 20.5f;
int offset = 30;

float value = (length * scale) + offset;
```

- [Loxodon Framework Addressable](#)

It's function extend and support for Addressable Asset System.

Lua plugin installation (optional)

In this framework, Lua language is supported by plug-in extensions. It relies on Tencent's XLua project and the Loxodon.Framework plugin. You can download [Loxodon.Framework.XLua.unitypackage](#) from Github, and import into your project. It is optional and only needs to be installed for projects that need hot update and are developed in Lua language. The specific installation steps are as follows.


XLua Installation

Download the latest version of XLua from the Xlua Github repository, you can use the source code "Source code.zip" or xlua_v2.x.xx.zip version (xlua_v2.x.xx.zip version is recommended to avoid conflicts with XLua example class names). Please unzip the downloaded xlua and copy it into the current project.

Note: XLua has compatibility issues in Unity2018. In editor mode, please use .Net3.5 or .Net4.x. Do not use .Net Standard2.0, otherwise errors will occur. If you want to use .Net Standard2.0, please refer to xlua The FAQ addresses compatibility issues.

[XLua FAQ](#)

[XLua Download](#)

 xlua_v2.1.14.zip 	5.01 MB
 xlua_v2.1.14_general.zip	4.48 MB
 xlua_v2.1.14_luajit.zip	5.15 MB
 Source code (zip)	
 Source code (tar.gz)	

Import Lua plugin

Download [Loxodon.Framework.XLua.unitypackage](#) from github and import it into your Unity project.

If there is a compilation error, please check whether the XLua Examples directory is imported. The InvokeLua.cs file in this directory defines the PropertyChangedEventArgs class. Because namespace is NOT used, class names will conflict. Please delete the Examples folder in the XLua directory or Add a namespace to the PropertyChangedEventArgs class in the InvokeLua.cs file.

Import the examples

Find Examples.unitypackage file in the "LoxodonFramework/XLua/Package Resources/" folder and import it into the project.

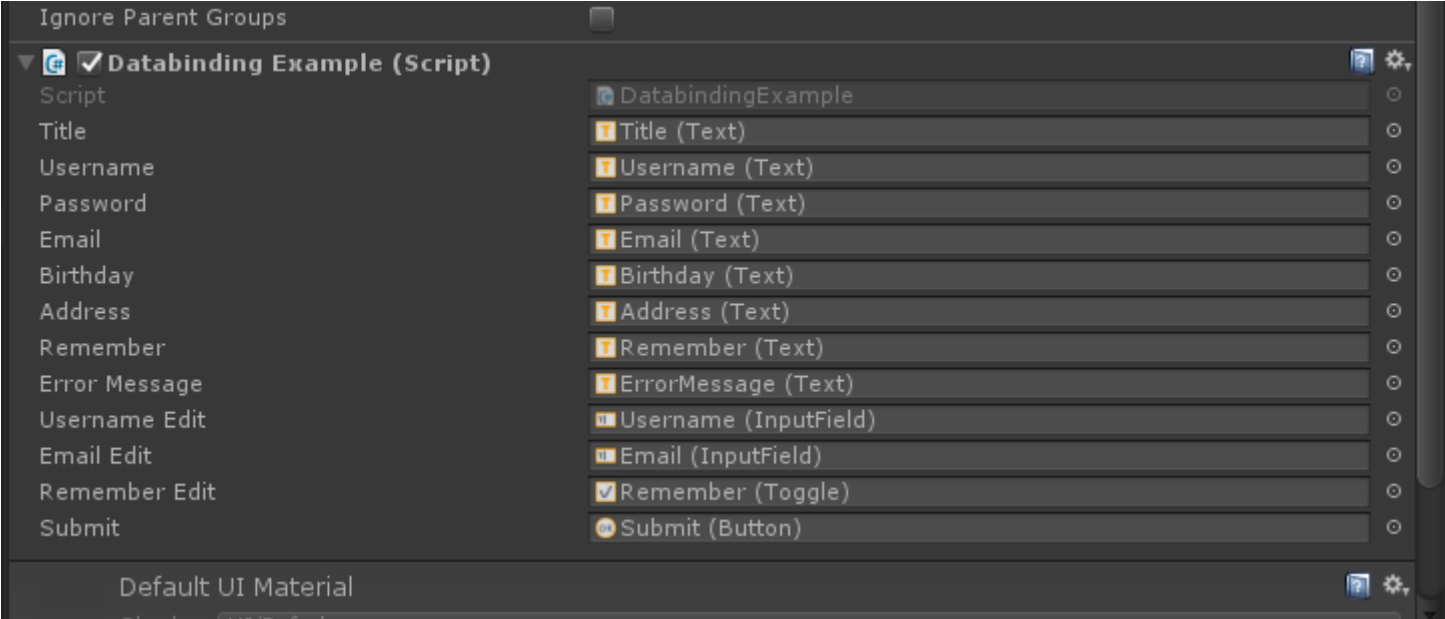
Quick start

Create a view which displays account information on the left and a form on the right. You can modify the account information on the left through the submit button. Now we will demonstrate how to do it through the frame view and data binding functions. The view is as follows:



C# example

Add the viewscript component DatabindingExample to the root object of a UI view, assign the UI control to the corresponding property. In this example, the properties are defined by C # hard coding. Of course, you can also use the dynamic property table VariableArray to Dynamically define attributes. For details, refer to Lua example. The configrued attributes are displayed as below screen.



Look at the code below, which introduces how to define the view model and the view script, how to bind the view to the view model.

```

/// <summary>
/// AccountViewModel
/// </summary>
public class AccountViewModel : ObservableObject
{
    private int id;
    private string username;
    private string password;
    private string email;
    private DateTime birthday;
    private readonly ObservableProperty<string> address = new ObservableProperty<string>();

    public int ID
    {
        get { return this.id; }
        set { this.Set<int>(ref this.id, value, "ID"); }
    }

    public string Username
    {
        get { return this.username; }
        set { this.Set<string>(ref this.username, value, "Username"); }
    }

    public string Password
    {
        get { return this.password; }
        set { this.Set<string>(ref this.password, value, "Password"); }
    }

    public string Email
    {
        get { return this.email; }
        set { this.Set<string>(ref this.email, value, "Email"); }
    }

    public DateTime Birthday
    {
        get { return this.birthday; }
        set { this.Set<DateTime>(ref this.birthday, value, "Birthday"); }
    }

    public ObservableProperty<string> Address
    {
        get { return this.address; }
    }
}

/// <summary>
/// DatabindingViewModel
/// </summary>
public class DatabindingViewModel : ViewModelBase
{
    private AccountViewModel account;
    private bool remember;
    private string username;
    private string email;
    private ObservableDictionary<string, string> errors = new ObservableDictionary<string, string>();

    public AccountViewModel Account
    {
        get { return this.account; }
        set { this.Set<AccountViewModel>(ref account, value, "Account"); }
    }

    public string Username
    {
        get { return this.username; }
        set { this.Set<string>(ref this.username, value, "Username"); }
    }

    public string Email
    {
        get { return this.email; }
        set { this.Set<string>(ref this.email, value, "Email"); }
    }
}

```

```

public bool Remember
{
    get { return this.remember; }
    set { this.Set<bool>(ref this.remember, value, "Remember"); }
}

public ObservableDictionary<string, string> Errors
{
    get { return this.errors; }
    set { this.Set<ObservableDictionary<string, string>>(ref this.errors, value, "Errors"); }
}

public void OnUsernameValueChanged(string value)
{
    Debug.LogFormat("Username ValueChanged:{0}", value);
}

public void OnEmailValueChanged(string value)
{
    Debug.LogFormat("Email ValueChanged:{0}", value);
}

public void OnSubmit()
{
    if (string.IsNullOrEmpty(this.Username) || !Regex.IsMatch(this.Username, "[a-zA-Z0-9_-]{4,12}$"))
    {
        this.errors["errorMessage"] = "Please enter a valid username.";
        return;
    }

    if (string.IsNullOrEmpty(this.Email) || !Regex.IsMatch(this.Email, @"^\w+([-+.] \w+)*@\w+([-.] \w+)*\.\w+([-.] \w+)*$"))
    {
        this.errors["errorMessage"] = "Please enter a valid email.";
        return;
    }

    this.errors.Clear();
    this.Account.Username = this.Username;
    this.Account.Email = this.Email;
}
}

```

```

/// <summary>
/// DatabindingExample
/// </summary>
public class DatabindingExample : UIView
{
    public Text title;
    public Text username;
    public Text password;
    public Text email;
    public Text birthday;
    public Text address;
    public Text remember;

    public Text errorMessage;

    public InputField usernameEdit;
    public InputField emailEdit;
    public Toggle rememberEdit;
    public Button submit;

    protected override void Awake()
    {
        //get application context
        ApplicationContext context = Context.GetApplicationContext();

        //launch data binding service
        BindingServiceBundle bindingService = new BindingServiceBundle(context.GetContainer());
        bindingService.Start();

        //initial localiztion service
        CultureInfo cultureInfo = Locale.GetCultureInfo();
        var provider = new DefaultDataProvider("LocalizationTutorials", new XmlDocumentParser())
        Localization.Current = Localization.Create(provider, cultureInfo);
    }
}

```

```

protected override void Start()
{
    //create account sub-view
    AccountViewModel account = new AccountViewModel()
    {
        ID = 1,
        Username = "test",
        Password = "test",
        Email = "yangpc.china@gmail.com",
        Birthday = new DateTime(2000, 3, 3)
    };
    account.Address.Value = "beijing";

    //create data binding view
    DatabindingViewModel databindingViewModel = new DatabindingViewModel()
    {
        Account = account
    };

    //get data binding context
    IBindingContext bindingContext = this.BindingContext();

    //assign view model to DataContext
    bindingContext.DataContext = databindingViewModel;

    //binding UI controller with view modle
    BindingSet<DatabindingExample, DatabindingViewModel> bindingSet;
    bindingSet = this.CreateBindingSet<DatabindingExample, DatabindingViewModel>();

    //binding left-side view with account sub-view model
    bindingSet.Bind(this.username).For(v => v.text).To(vm => vm.Account.Username).OneWay();
    bindingSet.Bind(this.password).For(v => v.text).To(vm => vm.Account.Password).OneWay();
    bindingSet.Bind(this.email).For(v => v.text).To(vm => vm.Account.Email).OneWay();
    bindingSet.Bind(this.remember).For(v => v.text).To(vm => vm.Remember).OneWay();
    bindingSet.Bind(this.birthday).For(v => v.text).ToExpression(vm => string.Format("{0} ({1})",
        vm.Account.Birthday.ToString("yyyy-MM-dd"), (DateTime.Now.Year - vm.Account.Birthday.Year))).OneWay();
    bindingSet.Bind(this.address).For(v => v.text).To(vm => vm.Account.Address).OneWay();

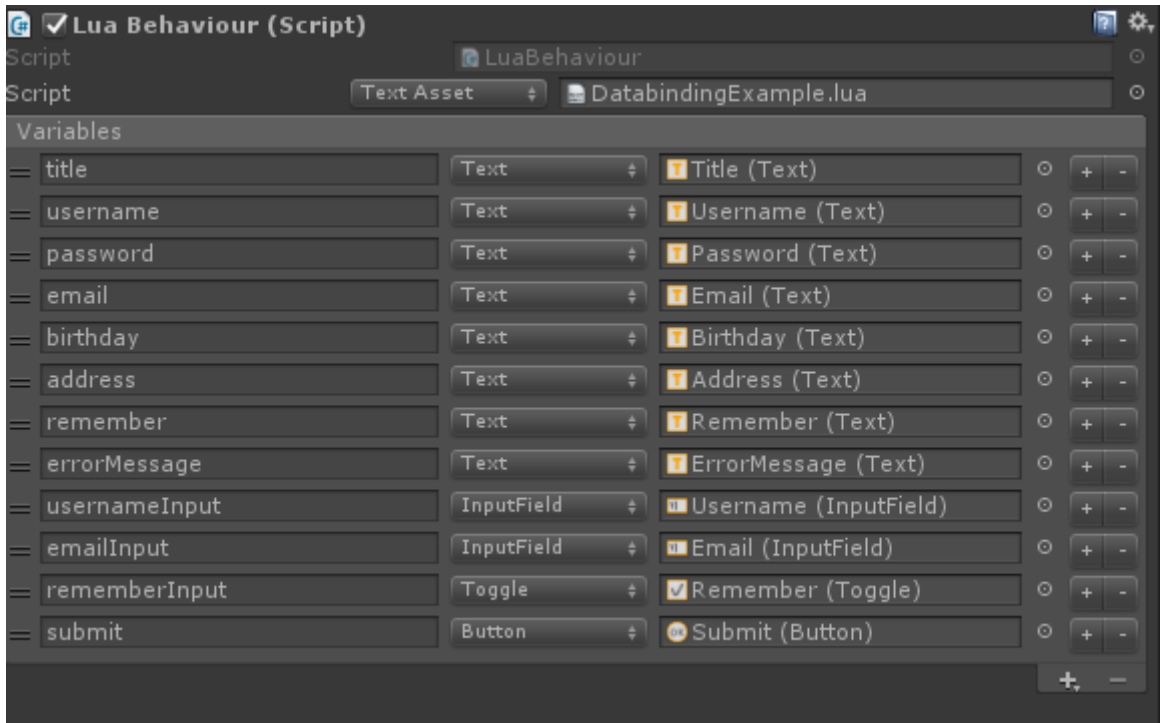
    //binding right-side form with view model
    bindingSet.Bind(this.errorMessage).For(v => v.text).To(vm => vm.Errors["errorMessage"]).OneWay();
    bindingSet.Bind(this.usernameEdit).For(v => v.text, v => v.onEndEdit).To(vm => vm.Username).TwoWay();
    bindingSet.Bind(this.usernameEdit).For(v => v.onValueChanged).To<string>(vm => vm.OnUsernameValueChanged);
    bindingSet.Bind(this.emailEdit).For(v => v.text, v => v.onEndEdit).To(vm => vm.Email).TwoWay();
    bindingSet.Bind(this.emailEdit).For(v => v.onValueChanged).To<string>(vm => vm.OnEmailValueChanged);
    bindingSet.Bind(this.rememberEdit).For(v => v.isOn, v => v.onValueChanged).To(vm => vm.Remember).TwoWay();
    bindingSet.Bind(this.submit).For(v => v.onClick).To(vm => vm.OnSubmit);
    bindingSet.Build();

    //binding title, and title is configed through localized file
    BindingSet<DatabindingExample> staticBindingSet = this.CreateBindingSet<DatabindingExample>();
    staticBindingSet.Bind(this.title).For(v => v.text).To(() => Res.databinding_tutorials_title).OneTime();
    staticBindingSet.Build();
}
}

```

Lua example

In the Lua example, the LuaBehaviour script is a general-purpose script provided by the framework. We only need to write a Lua script which bound to LuaBehaviour script, as shown DatabindingExample.lua in the following figure. In LuaBehaviour, to ensure commonality, all member attributes are also dynamically defined through the VariableArray attribute table, as shown in the following figure.



In the Lua script DatabindingExample.lua, all the dynamic properties in the above figure are registered into the Lua environment. We can access all the properties through the self object. Please check the following code.

```

require("framework.System")

local Context = CS.Loxodon.Framework.Contexts.Context
local LuaBindingServiceBundle = CS.Loxodon.Framework.Binding.LuaBindingServiceBundle
local ObservableObject = require("framework.ObservableObject")
local ObservableDictionary = require("framework.ObservableDictionary")

---
--creat an account sub-view model
--@module AccountViewModel
local AccountViewModel = class("AccountViewModel",ObservableObject)

function AccountViewModel:ctor(t)
    --applied father class ObservableObject's constructor. It's necessary, otherwise it can't listen to data change
    AccountViewModel.base(self).ctor(self,t)

    if not (t and type(t)=="table") then
        self.id = 0
        self.username = ""
        self.Password = ""
        self.email = ""
        self.birthday = os.time({year =1970, month = 00, day =00, hour =00, min =00, sec = 00})
        self.address = ""
    end
end

---
--create a view modle of data binding example
--@module DatabindingViewModel
local DatabindingViewModel = class("DatabindingViewModel",ObservableObject)

function DatabindingViewModel:ctor(t)
    --applied father class ObservableObject's constructor. It's necessary, otherwise it can't listen to data change

    DatabindingViewModel.base(self).ctor(self,t)

    if not (t and type(t)=="table") then
        self.account = Account()
        self.remember = false
        self.username = ""
        self.email = ""
        self.errors = ObservableDictionary()
    end
end

end

function DatabindingViewModel:submit()
    if #self.username < 1 then
        --pay attention to C# dictionary operating method, accessed via set_Item or get_Item
        self.errors:set_Item("errorMessage","Please enter a valid username.")
        return
    end

    if #self.email < 1 then
        --pay attention to C# dictionary operating method, accessed via set_Item or get_Item
        self.errors:set_Item("errorMessage","Please enter a valid email.")
        return
    end

    self.errors:Clear()

    self.account.username = self.username
    self.account.email = self.email
    self.account.remember = self.remember
end

---
--creat a data binding view, expand class of DatabindingExample.cs, and target is transfered from C# scripts.
--@module DatabindingExample
local M = class("DatabindingExample",target)

function M:awake()
    local context = Context.GetApplicationContext()
    local container = context:GetContainer()

    --initial Lua data binding service, recommand to create it in C# start scripts of game.
    local bundle = LuaBindingServiceBundle(container)
    bundle:Start();

```

```

end

function M:start()
    --initial Account sub-view model
    local account = AccountViewModel({
        id = 1,
        username = "test",
        password = "test",
        email = "yangpc.china@gmail.com",
        birthday = os.time({year =2000, month = 03, day =03, hour =00, min =00, sec = 00}),
        address = "beijing",
        remember = true
    })

    --initial view model
    self.viewModel = DatabindingViewModel({
        account = account,
        username = "",
        email = "",
        remember = true,
        errors = ObservableDictionary()
    })

    self.BindingContext().DataContext = self.viewModel

    --binding UI controller with view model
    local bindingSet = self:CreateBindingSet();

    bindingSet:Bind(self.username):For("text"):To("account.username"):OneWay()
    bindingSet:Bind(self.password):For("text"):To("account.password"):OneWay()
    bindingSet:Bind(self.email):For("text"):To("account.email"):OneWay()
    bindingSet:Bind(self.remember):For("text"):To("account.remember"):OneWay()
    bindingSet:Bind(self.birthday):For("text"):ToExpression(function(vm)
        return os.date("%Y-%m-%d",vm.account.birthday)
    end ,"account.birthday"):OneWay()
    bindingSet:Bind(self.address):For("text"):To("account.address"):OneWay()
    bindingSet:Bind(self.errorMessage):For("text"):To("errors[ 'errorMessage' ]"):OneWay()
    bindingSet:Bind(self.usernameInput):For("text", "onEndEdit"):To("username"):TwoWay()
    bindingSet:Bind(self.emailInput):For("text", "onEndEdit"):To("email"):TwoWay()
    bindingSet:Bind(self.rememberInput):For("isOn", "onValueChanged"):To("remember"):TwoWay()
    bindingSet:Bind(self.submit):For("onClick"):To("submit"):OneWay()

    bindingSet:Build()
end

return M

```


Features Introduction

Context

In many frameworks, we shall face the concept of context. It is an environment related to the current running code. You can provide the environment data or service in the context which is required by the current operation. Loxodon-framework provides ApplicationContext and PlayerContext according to the characteristics of game development, and also support developers to create other contexts according to requirement.

In the context, Loxodon-framework created a service container (service container be refer to the next chapter) to store services related to the current context, and also created a dictionary to store data. Through Dispose() in the context, you can release all services registered in the context container. **However, it should be noted that the service must be inherited from the System.IDisposable interface, otherwise it cannot be released automatically.**

Application Context

The application context is a global and gingleton context, which mainly stores data and service need be shared globally. All basic services, such as view positioning service, resource loading service, network connection service, localization service, configuration file service, Json/XML parsing service, data binding service etc. These are the basics services in the integrated game and should be registered in the service container of the application context, which can be obtained through the application context.

```
//get application context
ApplicationContext context = Context.GetApplicationContext();

//get service container of context
IServiceContainer container = context.GetContainer();

//initial data binding service, which is a set of services that are initialized and registred into service container through ServiceBundle
BindingServiceBundle bundle = new BindingServiceBundle(context.GetContainer());
bundle.Start();

//initial IUIViewLocator, and registred into container
container.Register<IUIViewLocator>(new ResourcesViewLocator ());

//initial localization service, and registred into container
CultureInfo cultureInfo = Locale.GetCultureInfo();
var dataProvider = new ResourcesDataProvider("LocalizationExamples", new XmlDocumentParser());
Localization.Current = Localization.Create(dataProvider, cultureInfo);
container.Register<Localization>(Localization.Current);

//get IUIViewLocator service from context
IUIViewLocator locator = context.GetService<IUIViewLocator>();

//get localiztion service from context
Localization localization = context.GetService<Localization>();
```

Player Context

The player context is only relevant to the currently logged-in game player. For example, after a game player Clark logs in to the game, his basic information and related services in the game should be stored in the player context. Knapsack service is responsible for pulling and synchronizing the player's knapsack data, which caches the weapons, equipment, and props in the player's knapsack. It is only relevant to the current player. When the player logs out or switches accounts, these data should be released and cleared. When use the player context to store services and values, designer only need to call the PlayerContext.Dispose() function to release all data and services related to the current player.

The player context inherits all the services and attributes of the global context by default, all global services and data can be obtained through the player context. When the player context registers the same service or attribute as the key value in the global context, It will be stored in the player context and will not overwrite the data stored in the global context. When accessed through the key, the data in the player context will be returned first. The global context will be searched only if it is not found in the player context.

```

//create a player context for Player clark
PlayerContext playerContext = new PlayerContext("clark");

//get service container from player context
IServiceContainer container = playerContext.GetContainer();

//save roler information into player context
playerContext.Set("roleInfo", roleInfo);

//initial knapsack service, and registred it into service container of player context
container.Register<IKnapsackService>(new KnapsackService());

//get ViewLocator service registred in overall context through player context
IUIViewLocator locator = playerContext.GetService<IUIViewLocator>();

//get Localization service registred in overall context through player context
Localization localization = playerContext.GetService<Localization>();

//When player clark sign-out, dispose player context, automatic dispose all service register in current player context
playerContext.Dispose();

```

Other Contexts

In general, global context and player context are enough form most game developments. But in some case, we still need a context to store environmental data, such as in MMO games, a specific game is loaded, then an exclusive context is created for this copy. After the copy is end, then that exclusive context is released resources.

```

//create a context, and context shall set parameter of container as null automaticly
//set parameter contextBase as playerContext, which inherit the service and property of playerContext automaticly
Context context = new Context(null,playerContext);

//get service container of context
IServiceContainer container = context.GetContainer();

//register a battle service into service container
container.Register<IBattleService>(new BattleService());

```

Service Container

At the beginning of this project, I learned a lot of open-source projects in C# control inversion and dependency injection(IoC/DI). At first, I plan to use Zenject as service container. Considered the memory and CPU resource are quite valuable in mobile projects, I refused to adapt a heavy library to consume memory and waste performance caused by reflection. It is also inappropriate to force users to accept IoC/DI, After all, not everyone likes it. According above ideas, I designed a simple service container by myself to meet the most basic functions of service registration, deregistration, and reading.

Note: All registered services can be automatically released at IServiceContainer.Dispose() only if they inherit the System.IDisposable interface and implement the Dispose function.

Service Registry(IServiceRegistry)

Service registry is responsible for registering and unregistering services. It can register a service instance into the container according to the service type or name, or register a service factory into the container. Users can decide to register a service factory according requirement, create a singleton service or a new service instance every time.

```

IServiceContainer container = ...
IBinder binder = ...
IPathParser pathParser = ...

// Register a service of type IBinder into the container, by using container.Resolve <IBinder>() or
// container.Resolve("IBinder") to access this service.
// By default, the container uses typeof(IBinder).name as the Key.
container.Register<IBinder>(binder);

// If you need to register more than one IPathParser into the container,
// please use the name parameters to distinguish them and get value through name parameters,
// such as container.Resolve("parser")
container.Register<IPathParser>("parser",pathParser);
container.Register<IPathParser>("parser2",pathParser2);

```

Service Locator(IServiceLocator)

Services can be obtained through the service locator. The service locator can query the service according to the service name or type. While service is registered by type, query can be by type or type name; while service is registered by a specific name as the Key, query can only be by service name.

```
IServiceContainer container = ...

// The IBinder service was registered as a type in the previous section of code,
// so you can query the service by type or name
IBinder binder = container.Resolve<IBinder>(); //or container.Resolve("IBinder")

//IPathParser is registered with the specific name "parser" in the previous section of code,
//so the service can only be queried by the name "parser"
IPathParser pathParser = container.Resolve("parser");
```

Service Bundle(IServiceBundle)

The ServiceBundle is responsible to register and unregister a group of related services. For example, data binding service is to register all data binding related services at one time by the ServiceBundle.Start(). When the service is not useful, ServiceBundle.Stop() can unregister all services of the entire module (refer to below code). That is wonderful some times, such as starting or stopping all services of a module.

```
// Initialize the data binding module, start the data binding service, register the service
BindingServiceBundle bundle = new BindingServiceBundle(context.GetContainer());
bundle.Start();

// Stop the data binding module and logout of all data binding related services
bundle.Stop();
```

Application configuration(Preference)

Perference can be recognized as PlayerPrefs in Unity3d; but I do extend, supplement and standard for the PlayerPrefs functions. Except store basic data type, such as boolean/int/float/string etc., Perference can also store DateTime/Vector2/Vector3/Vector4/Color/Version and any object type which can be serialized by JsonUtility. You can even customize type encoder and decoder(ILogger) to extend any type which you wish to store. Perference supports encrypted data storage, and already implemented two persistence methods. The first is to convert the data to a string and store it in PlayerPrefs of Unity3D. The second is to store in file with binary. Generally, I favor the file persistence method at project testing stage, because I can directly delete the files in the folder of Application.persistentDataPath to easily erase the configuration.

In addition to extending the above functions, Perference also extends the scope of configuration. Like Context in the previous intro, it provides global configuration and player configuration, and also supports the configuration for a partial module. The global configuration can be used to store the current resource update version, the last login accounts and other application-related information. The player configurations can be multiple(if multiple accounts are logged in at one machine), it can store a specific player's configuration information at this machine, such as the player's background music, sound effects, picture quality, visibility settings etc. in the game.

Let me introduce how it works as below code.

```
// register a Preference factory, the default is PlayerPrefsPreferencesFactory factory,
// only register the BinaryFilePreferencesFactory factory while need File persistent
Preferences.Register(new BinaryFilePreferencesFactory());

// Get the global configuration, or create it automatically if it does not exist
Preferences globalPreferences = Preferences.GetGlobalPreferences();

// Stores the updated data version for the current resource
globalPreferences.SetObject<Version>("DATA_VERSION",dataVersion);

// Save the user name of the last successful login in the game,
// and fill it in the account input box automatically when starting the game at next time
globalPreferences.SetString("username","clark");

// Call the Save function to save the data after it has been modified
globalPreferences.Save();

// Get the configuration based on the key value "clark@zone5". Or create it automatically if it does not exist,.
// in the function Preferences.GetPreferences (), the name is just an access Key,
// you can use it completely according to its own meaning.
Preferences userPreferences Preferences.GetPreferences("clark@zone5");

// Set the game music, sound effect and save it
userPreferences.SetBool("Music_Enable",true);
userPreferences.SetBool("Sound_Enable",true);
userPreferences.Save();
```

Although Preferences already supports most data types, but you can keep extend yourself type through ITypeEncoder; if you have requirements for the security of configuration data, you can also use your own password to encrypt the data.

```
/// <summary>
/// Customize a type encoder
/// </summary>
public class ColorTypeEncoder : ITypeEncoder
{
    private int priority = 900;          //When a type is supported by more than one type encoder,
        //the highest priority is valid (between -999 and 999).

    public int Priority
    {
        get { return this.priority; }
        set { this.priority = value; }
    }

    public bool IsSupport(Type type)
    {
        if (type.Equals(typeof(Color)))
            return true;
        return false;
    }

    // exchange the string type to the object type
    public object Decode(Type type, string value)
    {
        if (string.IsNullOrEmpty(value))
            return null;

        Color color;
        if(ColorUtility.TryParseHtmlString(value,out color))
            return color;

        return null;
    }

    // Convert the object to a String to save it, because PlayerPrefs only supports String data
    public string Encode(object value)
    {
        return ColorUtility.ToHtmlStringRGBA((Color)value);
    }
}

// The default encryption is AES128_CBC_PKCS7, but you can also implement your own IEncryptor interface
// and define your own encryption algorithm.
byte[] iv = Encoding.ASCII.GetBytes("5CyM5tcL3yDFiWlN");
byte[] key = Encoding.ASCII.GetBytes("W8fnmqMynlTJXPM1");

IEncryptor encryptor = new DefaultEncryptor(key, iv);

// Serialize and deserialize classes
ISerializer serializer = new DefaultSerializer();

// Add a custom type encoder
serializer.AddTypeEncoder(new ColorTypeEncoder());

// Register for the Preferences factory
BinaryFilePreferencesFactory factory = new BinaryFilePreferencesFactory(serializer, encryptor);
Preferences.Register(factory);
```

More examples can be found at [Basic Tutorials.unity](#)

Configuration(Properties File)

The configuration file is an necessary part in game or application developmetn. It's used to manage the configuration parameters of the game or application, especially to support different platform, with several SDK configuration requirement. There are different access requirements, updating policy in different platform, which need to config different parameters. Although this configuration can inherit the class of ScriptableObject in Unity3D, but parameters are not uniform due to adapt much platform purpose. As result, the developers has to modify these parameters frequently. To avoid above situation, I use traditional properties file to config paramters, and one properties file can cover all configuration requirement.

Supported Data Types

All below types are supported by default, and new data types can be supported through a custom type converter ITypeConverter.

Type	Default Value	Description
sbyte	0	signed byte, -127-128
byte	0	unsigned byte, 0-255
short	0	short type
ushort	0	Unsigned short
int	0	Integer type
uint	0	Unsigned integer
long	0	Long type
ulong	0	Unsigned long
char	“	Character type
float	0	Float type
double	0	Double type
datetime	1970-01-01T00:00:00	Time type
vector2	(0,0)	Vector2 type,eg: (0,0)
vector3	(0,0,0)	Vector3 type,eg: (0,0,0)
vector4	(0,0,0)	Vector4 type,eg: (0,0,0,0)
color	#000000	Color type, eg: #FF0000
rect	(0,0,0,0)	Rect type, eg:(x,y,width,height)
version	1.0.0	Version type, eg: 1.0.0

Array Delimiter

As with the localized configuration in CSV format, the array is separated by commas, and commas between double quotes, single quotes, parentheses (), square brackets [], braces {}, and angle brackets <> are ignored. If there are commas in the string of the array, please use double or single quotes to enclose the string.

Configuration File Combine

Configuration file provides combine function which can combine multiple profiles into a single profile. While your developes a game which supports multiple platforms, and configuartion parameters are not uniformed. You can firstly creat an "application.properties.txt" as default, then creat an "application.android.properties.txt" for Android extend; and creat an "application.ios.properties.txt" for IOS extend. Only the different parts need be reconfiged in these extend files, the common parts can be left unconfigured and used directly from the default configuration file.

Example is as follows. First load default configuration into combined configuration, then load special configuration related to current platform into combined configuration. The platform configuration shall be queried firstly, the default configuration shall be queried in the next.

Note: the later added configuration file has higher priority in combined configuration file.

```

/* Create a composite configuration */
CompositeConfiguration configuration = new CompositeConfiguration();

/* Load the default configuration file */
string defaultText = FileUtil.ReadAllText(Application.streamingAssetsPath + "/application.properties.txt");
configuration.AddConfiguration(new PropertiesConfiguration(defaultText));

#if UNITY_EDITOR
    string text = FileUtil.ReadAllText(Application.streamingAssetsPath + "/application.editor.properties.txt");
#elif UNITY_ANDROID
    string text = FileUtil.ReadAllText(Application.streamingAssetsPath + "/application.android.properties.txt");
#elif UNITY_IOS
    string text = FileUtil.ReadAllText(Application.streamingAssetsPath + "/application.ios.properties.txt");
#endif

/* Loads the configuration file of the current platform */
configuration.AddConfiguration(new PropertiesConfiguration(text));

/* Register the configuration file into the container */
container.Register<IConfiguration>(configuration);

```

A subset of Configuration File

Except combin function, Configuration File also supports subset function. When using this function, set Key according function module and separate by dot. One Key refer to one subset of configuration file. Below example shows how to use it.

Example of Configuration File

The properties file format is as follows, configure everything with key = value, the comment text that starts with #, blank lines are ignored. In below code, I config four groups in updating module, which are local/develop/pre-release/release group. Application can locate the effective group relation to the value of "application.config-group", such as "application.config-group=local" in example. Using " application.local" can get local group subset, localConfig=config.Subset("application.local").

```

#application config
application.app.version = 1.0.0
application.data.version = 1.0.0

#gateway
application.config-group = local

#local
application.local.upgrade.url = http://test.your domain name.com/loxodon/framework/upgrade/check
application.local.username = loxodon.framework
application.local.password = loxodon.framework
application.local.gateway = 127.0.0.1:8000 , 192.168.0.30:8000

#develop
application.develop.upgrade.url = http://test.your domain name.com/loxodon/framework/upgrade/check
application.develop.username = loxodon.framework
application.develop.password = loxodon.framework
application.develop.gateway = 192.168.0.1:8000

#pre-release
application.pre-release.upgrade.url = http://pre.release.your domain name.com/loxodon/framework/upgrade/check
application.pre-release.username = loxodon.framework
application.pre-release.password = loxodon.framework
application.pre-release.gateway = 172.217.160.78:8000 , 172.217.160.79:8000 , 172.217.160.80:8000

#release
application.release.upgrade.url = http://release.your domain name.com/loxodon/framework/upgrade/check
application.release.username = loxodon.framework
application.release.password = loxodon.framework
application.release.gateway = 172.217.161.78:8000 , 172.217.161.79:8000 , 172.217.161.80:8000

```

Configuration file load example

```
//Initialize the configuration file
TextAsset text = Resources.Load<TextAsset>("application.properties");
IConfiguration conf = new PropertiesConfiguration(text.text);

//Application version number
Version appVersion = conf.GetVersion("application.app.version");
//data version number
Version dataVersion = conf.GetVersion("application.data.version");

//The currently configured group name
string groupName = conf.GetString("application.config-group");

//Gets a subset of the configuration through prefix
IConfiguration currentGroupConf = conf.Subset("application." + groupName);

//Obtain configuration information through subsets
string upgradeUrl = currentGroupConf.GetString("upgrade.url");
string username = currentGroupConf.GetString("username");
string password = currentGroupConf.GetString("password");
string[] gatewayArray = currentGroupConf.GetArray<string>("gateway");
```

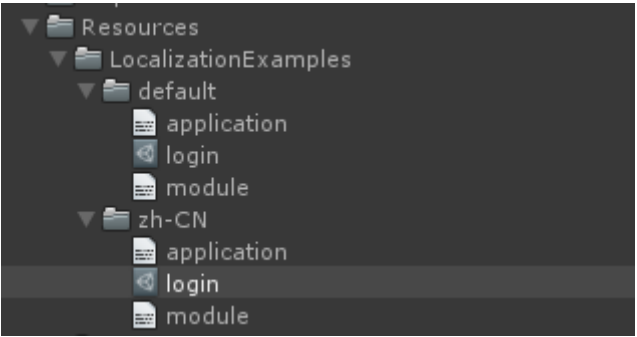
Internationalization and localization

For software, application and game etc., Internationalization and localization are necessary functions in order to meet different market requirements during game development. I designed the localization module in LoxodonFramework refer to the localization concept in Android development. The localization module can be customized and extended same as the other modules. Let me introduce how to use the localization module.

Directory Structure

Localized files can be saved under the Resources directory, accessed through Unity3D's Resources; or placed into AssetBundle, loaded through AssetBundle; even you can leave it anywhere and read it through customed IDataProvider. All these methods can exist at the same time, and later method shall cover the early method. In this framework, I provide two data providers, DefaultDataProvider and AssetBundleDataProvider, to load localized data files in Resources and AssetBundle respectively. Whatever in Resources or AssetBundle, the directory structure and loading policy are the same. First, there must be a root directory of the localization configuration file, as shown the LocalizationExamples directory in the following figure. After that, create sub-directories under the root directory for each language, such as default, zh, zh-CN, zh-TW, zh-HK, en, en-US, En-CA, en-AU, and so on (For details, refer to the Name of the System.Globalization.CultureInfo class and TwoLetterISOLanguageName. Such as zh-CN is the Name and zh is the TwoLetterISOLanguageName). The default directory must be the whole and default language configuration, and it is required, while other directories are optional. The zh directory is a Chinese directory, zh-CN is a configuration directory in mainland China, zh-TW is a configuration directory in Taiwan, and zh-HK is a configuration directory in Hong Kong, China. The priority of the configuration file is (zh-CN | zh-TW | zh-HK)> zh > default, and higher priority will overwrite the lower one.

In each configuration file directory, the configuration file is recommended to be divided into multiple files according to the business module. Do not write all the configurations in a text file. As shown in the following figure, the global configuration is in application.xml file, and other configurations are named according the module name.



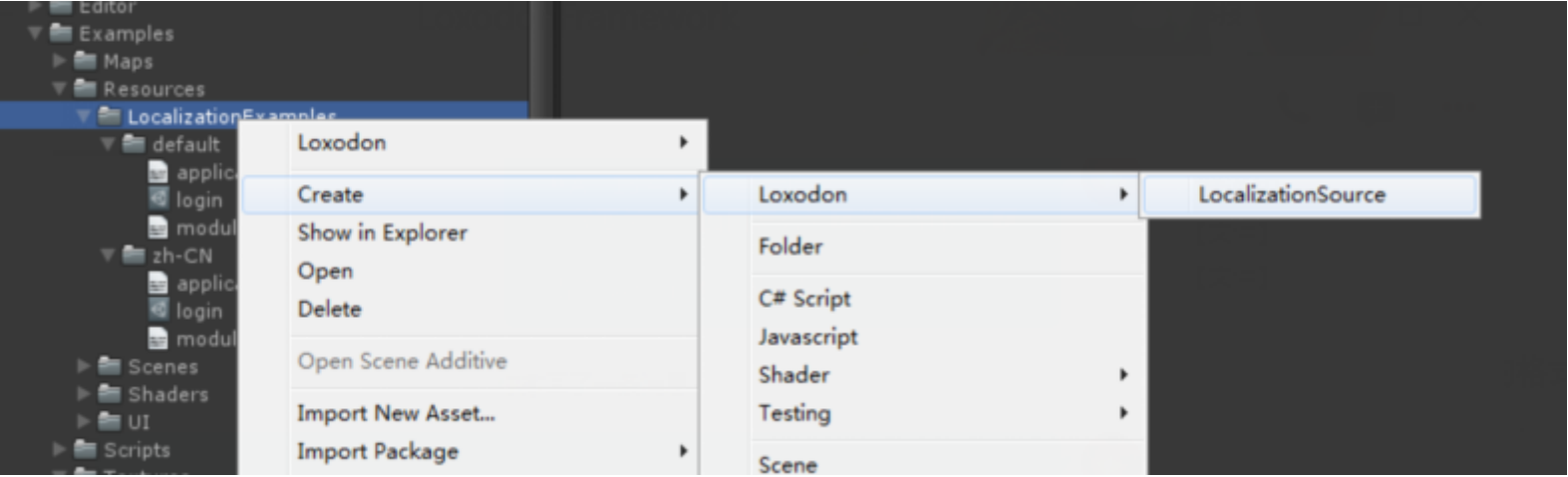
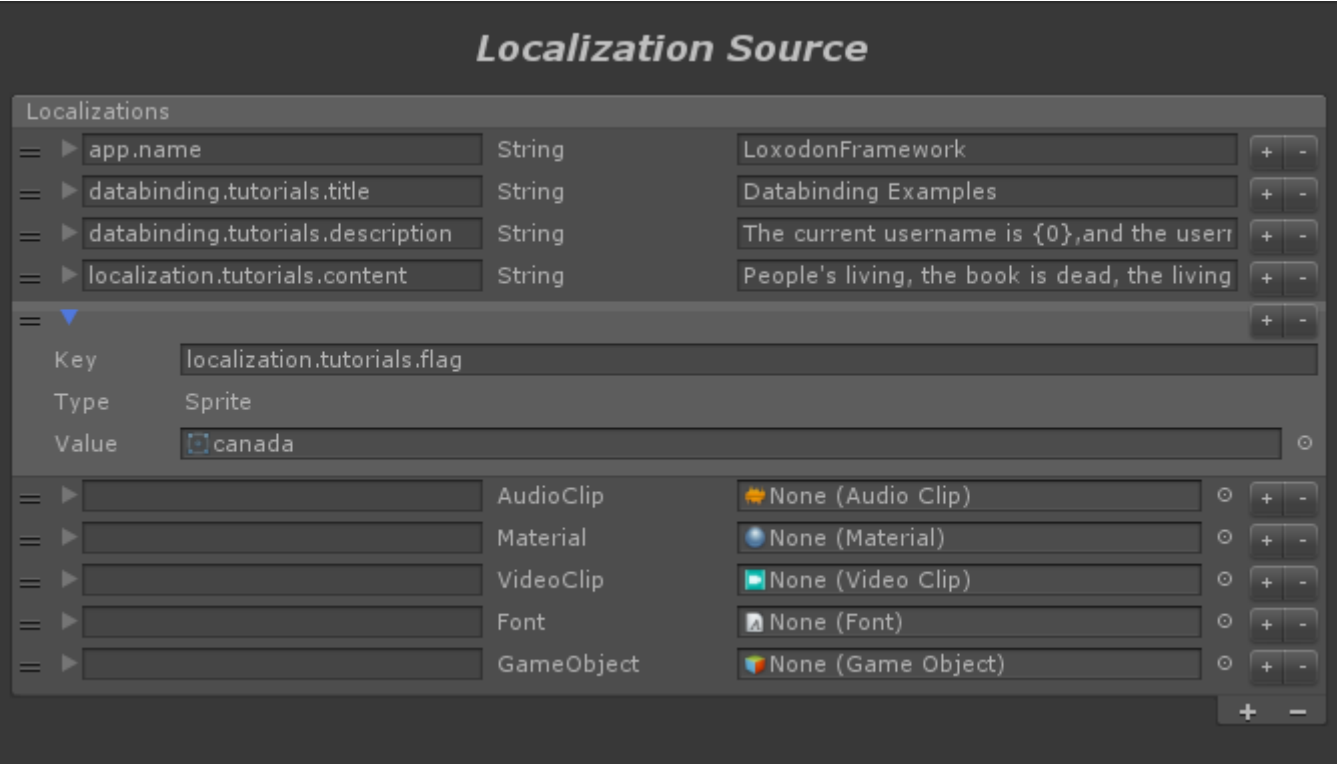
Format of the configuration file

The localization configuration file supports XML, Asset file (LocalizationSourceAsset) format and localized data source script by default. If necessary, you can also customize IDocumentParser to support other formats, such as Json format, csv format, binary format, or SQLite.

Sprites, textures (Texture2D / Texture3D), fonts (Font), audio effects (AudioClip), video (VideoClip), etc. which belongs to UnityEngine.Object resources can only be stored using the Asset file format or localized data source script. Other textual resources are recommended to be stored in XML or other text formats.

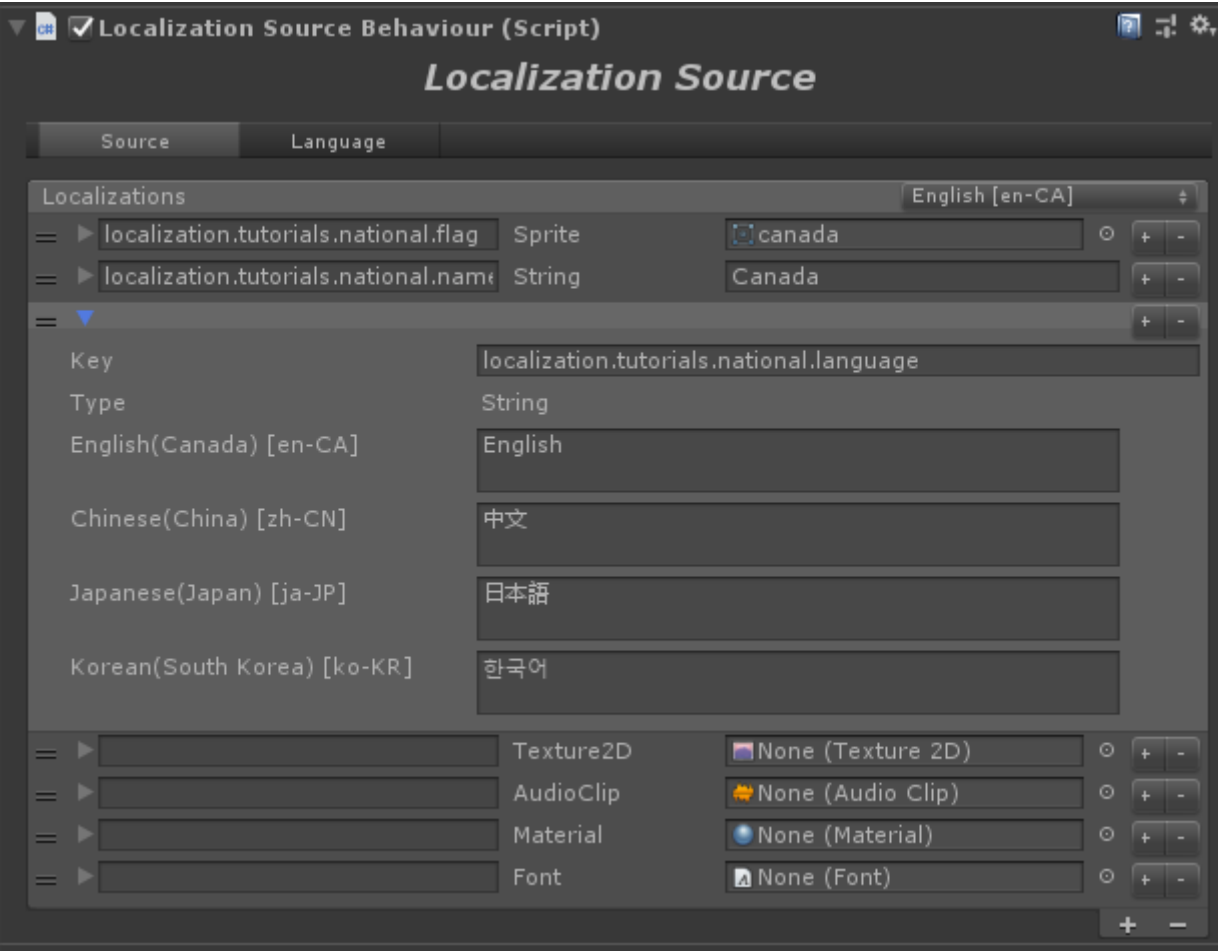
- Localization DataSource Asset File Format (LocalizationSourceAsset)
The localization sourceAsset file format is as shown below, which can be configured multiple types of resources. Each file corresponds to a language resource. Its directory structure is completely the same as the XML method, except the file format.

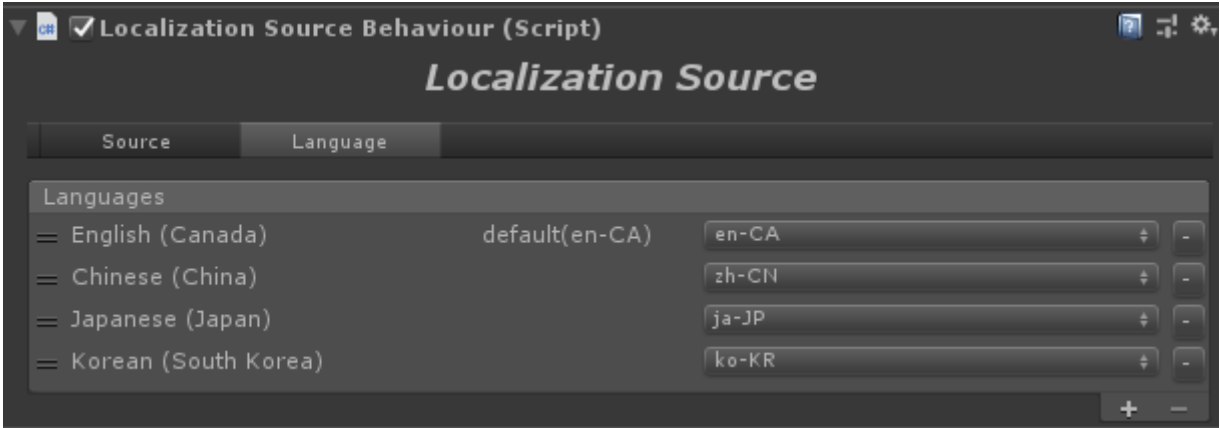
Files such as pictures and sounds take up more memory, please split resources by business module. The configuration of the same module is in the same Asset file. Load it into memory before using it, and release resources from memory after using it.



Localization DataSource Script method (LocalizationSourceBehaviour)

The localization datasource script is attached to the GameObject object and can be stored directly in Prefab or in the scene. It cannot be stored separately by language, and all localized resources that support languages should be configured in the same script file. The LocalizationSourceBehaviour script comes with a DataProvider, the data is automatically loaded when the script is run, and the data is automatically unloaded when the object is destroyed. This method is particularly suitable for use with UIView. Localized data is automatically loaded when UIView is created, and localized data is released when UIView is closed. Compared with the Asset file format, it has the advantage that it can be used like an Unity object and can be dragged into the scene or prefab, and there is no need to write any script to manage it. Its disadvantage is that data for multiple language versions will be completely loaded into memory, and that will take up more memory.





- XML file format

The XML file format can easily configure text-type data, but it cannot directly configure the object resources of the UnityEngine.Object. If you wish to use XML to configure resources such as sounds, pictures, and fonts etc., you can only configure the file paths of resources such as sounds, pictures, and fonts in XML, and dynamically load these resources by changing the file path when you use them.

The localization of text type does not spend too much memory, it is recommended to load all into memory when the game starts, and do not release them.

The XML format is configured as follows:

```
<!-- application.xml -->
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app.name">Loxodon Framework Examples</string>
  <string name="framework.name">LoxodonFramework</string>
  <vector3 name="user.position">(20 , 20.2 , 30)</vector3>
  <color name="color.black">#000000</color>
  <color-array name="button.transition.colors">
    <item>#FFFFFF</item>
    <item>#F5F5F5</item>
    <item>#C8C8C8</item>
    <item>#C8C8C8</item>
  </color-array>
  <datetime name="created">2016-10-27T00:00:00.000</datetime>
</resources>

<!-- module.xml -->
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="startup.progressbar.tip.loading">Loading...</string>
  <string name="startup.progressbar.tip.unzipping">Unzipping...</string>
  <string name="login.failure.tip">Login failure.</string>
  <string name="login.exception.tip">Login exception.</string>
  <string name="login.validation.username.error">Please enter a valid username.</string>
  <string name="login.validation.password.error">Please enter a valid password.</string>
  <string name="login.label.title.text">Sign in</string>
  <string name="login.button.confirm.text">Confirm</string>
  <string name="login.button.cancel.text">Cancel</string>
  <string name="login.label.username.text">Username:</string>
  <string name="login.label.password.text">Password:</string>
  <string name="login.input.username.prompt">Enter username...</string>
  <string name="login.input.password.prompt">Enter password...</string>
</resources>
```

XML special characters

In the name, attributes, and text content of XML, the characters of "<", ">", "&" etc. cannot be used directly. If these characters appear in an XML tag, XML parsing will report an error. While the text content must include these characters, there are two solutions. The first is to use escape characters. For example, the three characters in the preceding text can be replaced with "<", ">", and "&". The second way is to wrap the text content with tags <![CDATA[]]>, for example,<![CDATA[<color=#FF0000>This is a test.</color>]]>, the text content it represents is "<color=#FF0000>This is a test </color>". <![CDATA[]]> tags are generally recommended.

Escape character table

<	<	小于号
>	>	大于号
&	&	和
'	'	单引号
"	"	双引号

Example of escape character or

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="mainpage.title"><![CDATA[This text is <color=#FF0000>red</color>]]></string>
    <string name="mainpage.text">This text is &lt;color=#FF0000&gt;red&lt;/color&gt;</string>
</resources>
```

Numeric types supported by XML

All the following types and their array types are supported by default, and new data types can be supported through a custom type converter `ITypeConverter`.

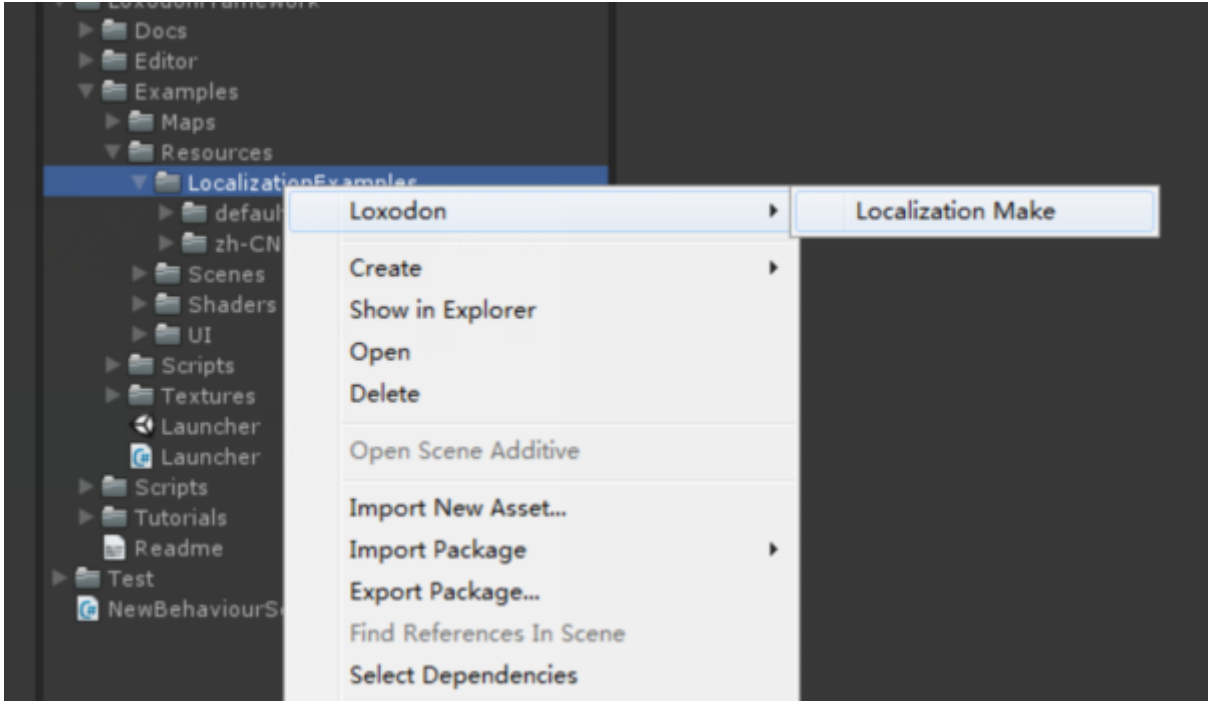
The array type is expressed by adding a "-array" suffix after the basic type. Such as string array type: string-array, using `<item><item>` between `<string-array></string-array>` to add array elements.

Basic Type	Default Value	Description
string	""	String type
boolean	false	Boolean type, flase or true
sbyte	0	Signed byte, -127-128
byte	0	Unsigned byte, 0-255
short	0	short type
ushort	0	Unsigned short
int	0	Integer type
uint	0	Unsigned integer
long	0	Long type
ulong	0	Unsigned long
char	“	Character type
float	0	Float type
double	0	Double type
decimal	0	number type
datetime	1970-01-01T00:00:00	Time type
vector2	(0,0)	Vector2 type,eg: (0,0)
vector3	(0,0,0)	Vector3 type,eg: (0,0,0)
vector4	(0,0,0)	Vector4 type,eg: (0,0,0,0)
color	#000000	Color type, eg: #FF0000
rect	(0,0,0,0)	Rect type, eg:(x,y,width,height)

Generate C# Code

The properties of the localized configuration, similar to the Android configuration, can be used to generate a static class. It's recommended while you use the C# version of MVVM, this adds a language compilation check mechanism to avoid errors. It's not recommended while you are programing in Lua, you can use the Localization class directly.

Right-click on the root directory of the localization configuration, and pop up the code generation menu as shown below. Click Localization Make, select the code directory and file name, and generate a C# static class.



```
public static partial class R
{
    public readonly static V<string> startup_progressbar_tip_loading = new V<string>("startup.progressbar.tip.loading");

    public readonly static V<string> startup_progressbar_tip_unzipping = new V<string>("startup.progressbar.tip.unzipping");

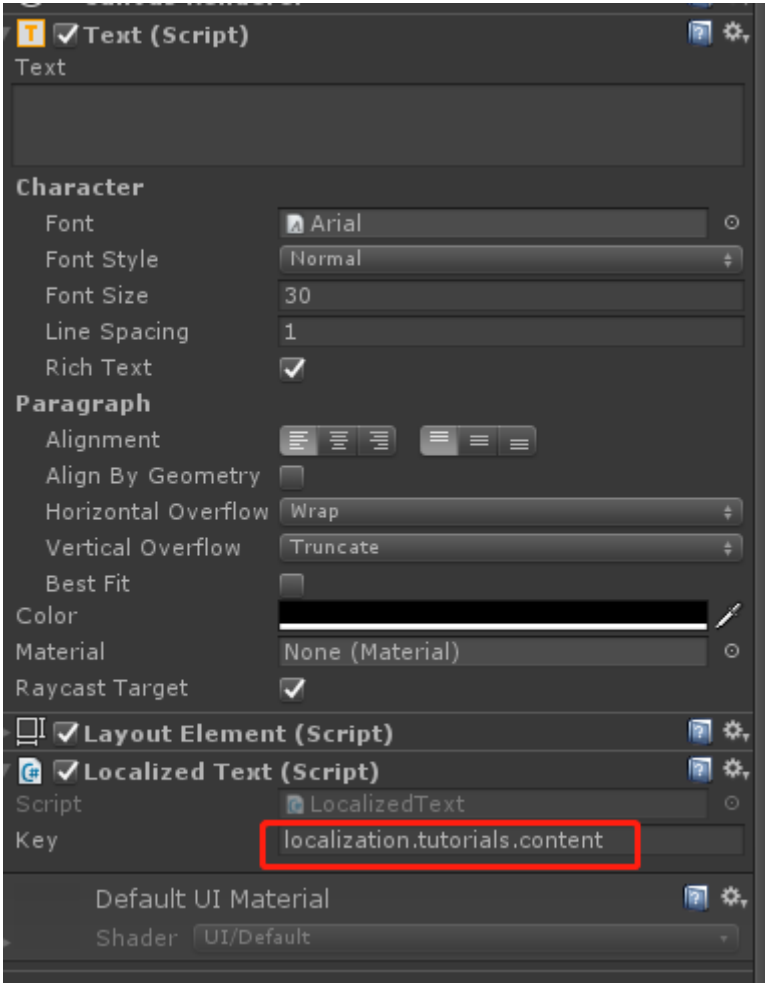
    public readonly static V<string> login_failure_tip = new V<string>("login.failure.tip");

    public readonly static V<string> login_exception_tip = new V<string>("login.exception.tip");
}
```

Localized view components

- **Text Localization**

Supports hanging a script on UnityEngine.UI.Text or UnityEngine.TextMesh object, and configuring the key of the localized string, it can automatically support multi-language switching. If it is just displaying text, it will not change with business logic. Support the use of this method to configure. Of course, you can also update the text of Text or TextMesh through data binding. If you modify the text in the ViewModel, the text in the view changes accordingly.



- **Localization of images, sounds, videos, fonts, materials**

The localization of image, sound, video, font, and material resources is recommended to use Asset file configuration (LocalizationSourceAsset). The resource configuration of different language versions is classified by business module and configured in different Asset files. For example, when you need to access the UI of a business module, load the localized resources of the current language version of this module before displaying the UI.

Of course, except to using the Asset file configuration method, you can also use XML and other text methods to configure the resource loading path in the XML file. When the language changes, the path of the picture or sound will also change, and the resource is loaded asynchronously through the view script, then the resource is replaced. This method is very flexible, but you need to write your own code to implement the loading logic. Localized resources such as pictures, sounds, fonts can be placed in Resources or AssetBundle, or sprite atlas

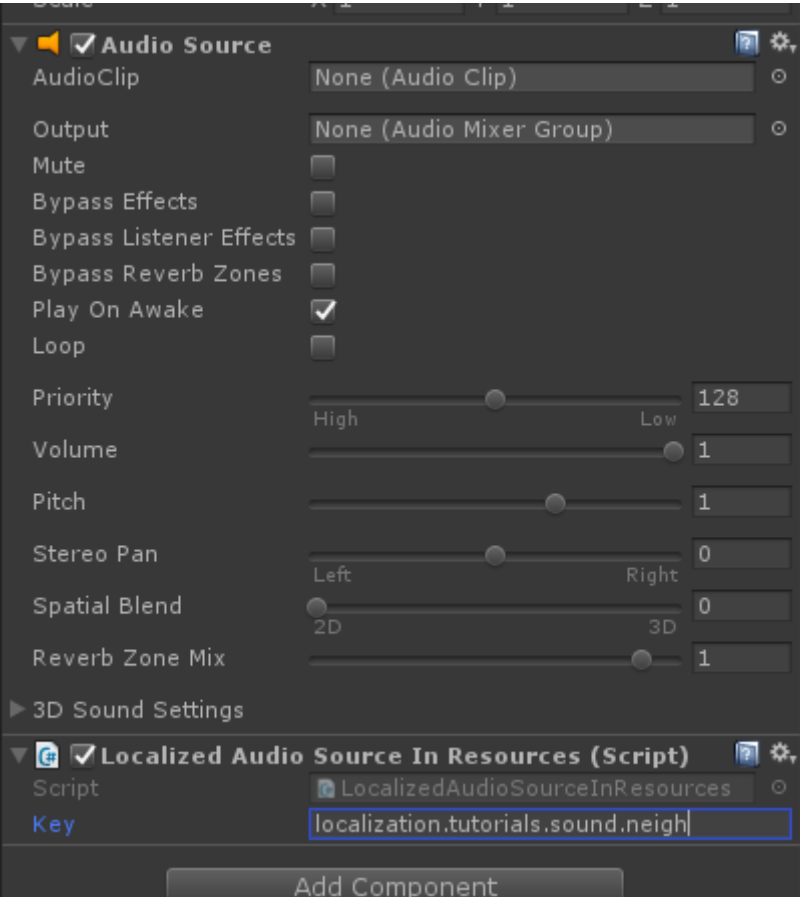
etc. I can't code a script to meet all functions, but I provides a component that load sounds or pictures from Resources (such as: LocalizedAudioSourceInResources.cs) as reference. You can refer to my component for more ways. The following example is the code for loading sound effects using my Loxodon.Framework.Bundle plugin.

```
[RequireComponent(typeof(AudioSource))]
public class LocalizedAudioSource : AbstractLocalized<AudioSource>
{
    private IResources resources;
    private void Awake()
    {
        Context context = Context.GetApplicationContext();
        this.resources = context.GetService<IResources>();
    }

    protected override void OnValueChanged(object sender, EventArgs e)
    {
        string path = (string)this.value.Value;
        var result = resources.LoadAssetAsync<AudioClip>(path);
        result.Callbackable().OnCallback(r =>
        {
            if (r.Exception != null)
                return;

            this.target.clip = r.Result;
        });
    }
}
```

The following figure is an example of loading sound effects from Resources using LocalizedAudioSourceInResources.



• UI size and color localization

The localization component supports types such as Rect, Color, and Vector2-4. In addition to the localization of pictures/sounds/text, the size/position/color of the UI view can also be localized. In particular, the localization of UI size can better adapt to the inconsistent requirements of different language text lengths. The setting of RectTransform is related to the anchor point position. The following code is just an example. Please modify it according to your actual situation.

```
public class LocalizedRectTransform : AbstractLocalized<RectTransform>
{
    protected override void OnValueChanged(object sender, EventArgs e)
    {
        RectTransform rectTransform = this.target;
        if (this.value.Value is Vector2[])
        {
            Vector2[] vectors = (Vector2[])this.value.Value;
            rectTransform.offsetMin = vectors[0];
            rectTransform.offsetMax = vectors[1];
            return;
        }

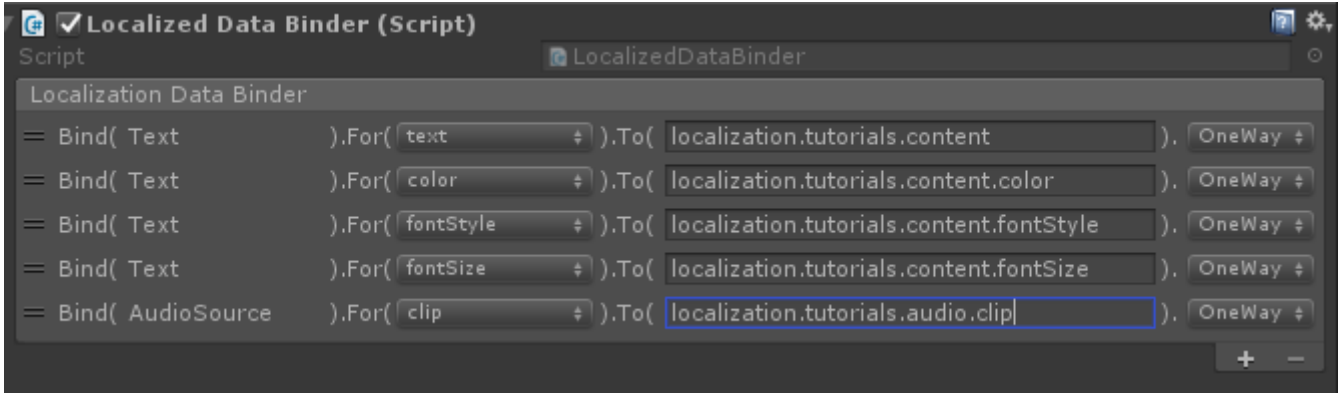
        if (this.value.Value is Rect)
        {
            Rect rect = (Rect)this.value.Value;
            rectTransform.anchoredPosition = new Vector2(rect.x, rect.y);
            rectTransform.sizeDelta = new Vector2(rect.width, rect.height);
            return;
        }
    }
}
```

The localization configuration is as follows:

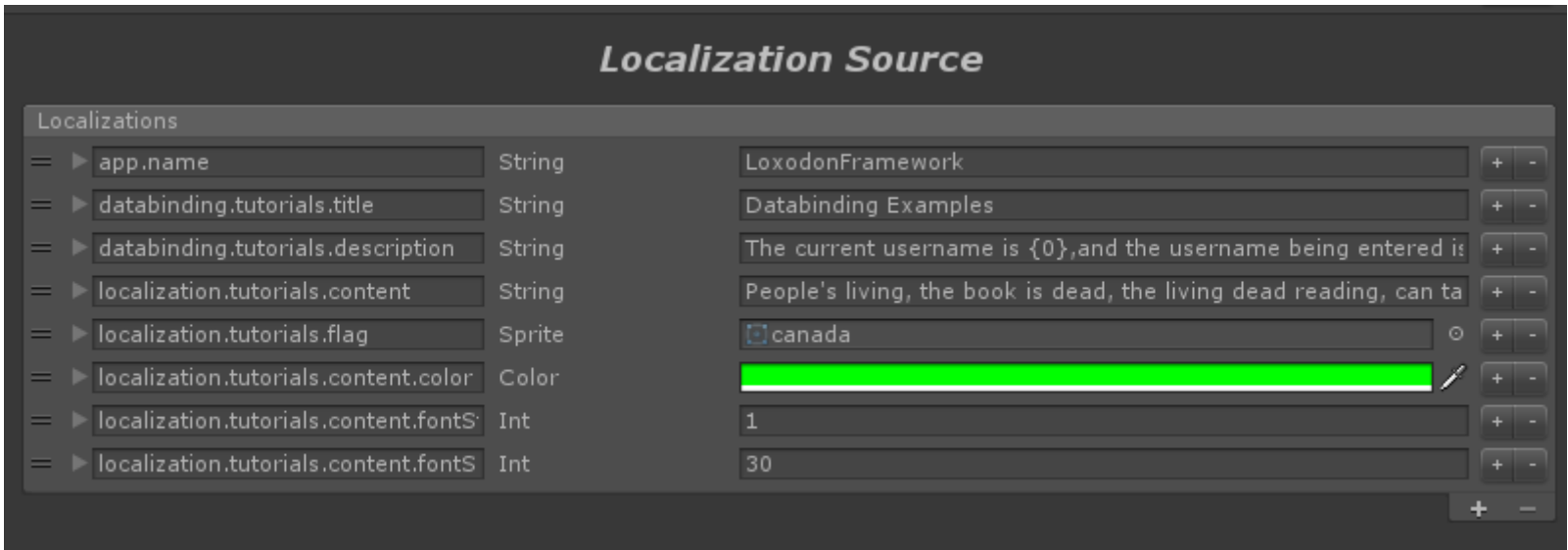
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <vector2-array name="button.position">
        <item>(100,50)</item>
        <item>(-100,-50)</item>
    </vector2-array>
    <rect name="button.position2">(100,100,200,60)</rect>
</resources>
```

Localization Data Binder(LocalizedDataBinder)

Localized data binder can carry on the batch quantity of binding for the properites and localized data of Text, TextMesh, Image, RawImage, SpriteRenderer, AudioSource, VideoPlayer etc. It uses the data binding service to work, and it must be initialized before using data binding services. In a game with the above UI components, the components properties can be binding after add LocalizedDataBinder script in node. The specific binding method is shown in the below figure.



The following diagram shows a localized resource. It is configured in the method of Localization Source. Of course, you can also configure it in XML or CSV file format.

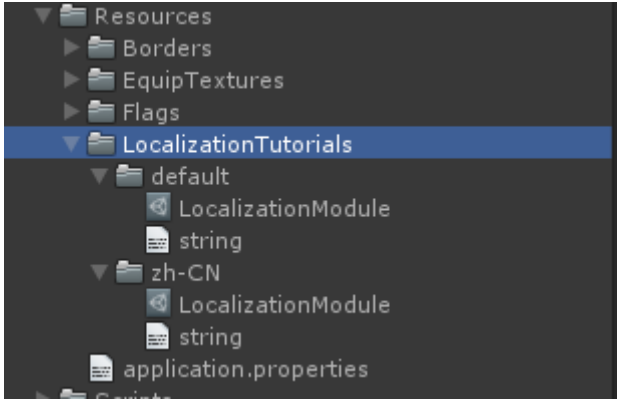


Data provider(IDataProvider)

The localization component of the framework supports the use of multiple data formats to configure localized resources. They have different file formats, different directory structures, and even different file search rules. No matter how complicated the situation is, you can use the data

provider (IDataProvider) and document parser (IDocumentParser) to unify them, load data through the data provider, and parse resource files through the document parser. In the framework, I provide some default data loaders, which can load localized data from Resources directory or AssetBundle based on the directory rules mentioned in the article. If you need to support more data formats, or to customize file search rules and loading methods, please refer to my code to implement a custom data provider.

The following code uses the default data provider to load the files in xml and asset format from the Resources/LocalizationTutorials/(the root directory of the tutorial localization resources, the directory structure is as shown in the figure below). All xml files of the current language in directory shanll be loaded by DefaultDataProvide, and do not need to release them due to less memory taken. The asset format file is loaded using the DefaultLocalizationSourceDataProvider. It is configured function need provide a specific asset file name list. And it only loads files in the name list, and configures multimedia resources such as pictures and sounds in the asset file. Please rember to release local resources through delete DefaultLocalizationSourceDataProvider from Localization.



```
var localization = Localization.Current;
localization.CultureInfo = new CultureInfo("en-US"); // set language

// Add a default XML data loader, load the localized Resources
// from the Resources/LocalizationTutorials directory

// Load file according to the ruler of (zh-CN|zh-TW|zh-HK) > zh > default
// if it doesn't meet requirement, you can customize the data loader

// Text resources don't take up much memory,
// and all XML files for the current language are loaded by default
localization.AddDataProvider(new DefaultDataProvider("LocalizationTutorials", new XmlDocumentParser()));

// To add an Asset data loader,
// and load resource login.asset from the Resources/LocalizationExamples directory

// Asset type resources should be loaded before use and released when they are not needed
var provider = new DefaultLocalizationSourceDataProvider("LocalizationTutorials","LocalizationModule.asset");
localization.AddDataProvider(provider);

// When the data is no longer in use, the data loader is removed and memory is freed
localization.RemoveDataProvider(provider);
```

Get the device's current language

In the older version of Unity3D, CultureInfo.CurrentCulture is invalid. Only the English language information can be obtained no matter on the PC or mobile device. So I provide the Locale tool to transfer from SystemLanguage to CultureInfo in Unity. The current language information can be obtained through Locale.GetCultureInfo (). And Chinese CultureInfor can be obtained through Locale.CultureInfo GetCultureInfoByLanguage(SystemLanguage.Chinese).

In the Unity 2018 with .net standard 2.0, CultureInfo.CurrentCulture is valid while I tested it on an Android phone. You can use CultureInfo.CurrentCulture to get the current system language information with Unity 2018.

Usage example

The previous article introduced some functions of localized components. Here we use an examples to understand the use of localized components.

The following example is how to use the localization function in C # code, get the localized string through the generated C# static class R or through the Localization class.

```

var localization = Localization.Current;
localization.CultureInfo = CultureInfo.CurrentCulture;
// set language, old version can use Locale.GetCultureInfo()
localization.AddDataProvider(new DefaultDataProvider("LocalizationTutorials", new XmlDocumentParser()));

// A member method call via Localization
string errorMessage = localization.GetText("login.validation.username.error", "Please enter a valid username.");

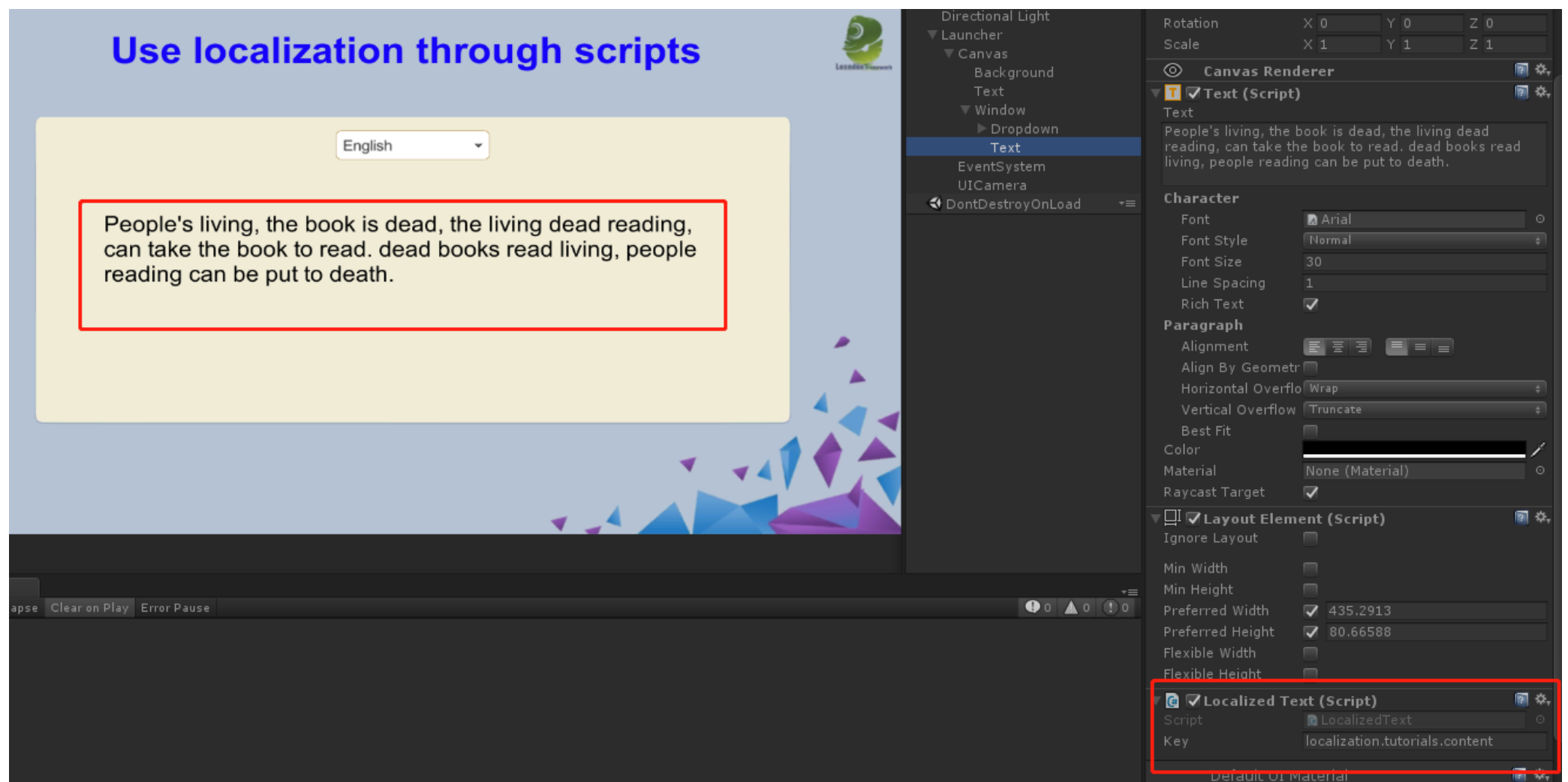
// Call via generated static code (Loxodon/Localization Make generates C# code,
// see Generating C# code section above)
string loadingMessage = R.startup_progressbar_tip_loading;

// Get a subset of the localized configuration, accessed through the subset
ILocalization localizationSubset = localization.Subset("login");
errorMessage = localizationSubset.GetText("validation.username.error", "Please enter a valid username.");

// For data binding use, use localize.getValue () to get an ObservableProperty
// that supports notification of value changes
bindingSet.Bind(target).For(v=>v.text)
    .ToValue(localization.GetValue("login.validation.username.error")).OneWay();

```

Using localized configuration with UI components, let's simulate a scenario of language switching in a game to understand the use of localized modules. In the below figure, the English in the red wireframe is loaded and modified through the localization service. It uses the LocalizedText component hanging on the Text object to switch between Chinese and English.




```

public class LocalizationExample : MonoBehaviour
{
    public Dropdown dropdown;

    private Localization localization;

    void Awake ()
    {
        CultureInfo cultureInfo = Locale.GetCultureInfoByLanguage (SystemLanguage.English);

        // Create a data provider and load the localization file
        // from the LocalizationTutorials directory
        var dataProvider = new DefaultDataProvider ("LocalizationTutorials", new XmlDocumentParser ());

        // Create a localized service
        Localization.Current = Localization.Create (dataProvider, cultureInfo);
        this.localization = Localization.Current;

        // Listen for changes in the drop-down list and switch between English and Chinese
        this.dropdown.onValueChanged.AddListener (OnValueChanged);
    }

    void OnValueChanged (int value)
    {
        switch (value) {
            case 0:
                // Set the current language of the localization service to English
                this.localization.CultureInfo = Locale.GetCultureInfoByLanguage (SystemLanguage.English);
                break;
            case 1:
                // Set the current language of the localization service to Chinese
                this.localization.CultureInfo = Locale.GetCultureInfoByLanguage (SystemLanguage.ChineseSimplified);
                break;
            default:
                // Set the current language of the localization service to English
                this.localization.CultureInfo = Locale.GetCultureInfoByLanguage (SystemLanguage.English);
                break;
        }
    }

    void OnDestroy ()
    {
        this.dropdown.onValueChanged.RemoveListener (OnValueChanged);
    }
}

```

The localization file configuration is as follows:

```

<!-- English Ver -->
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app.name">LoxodonFramework</string>
    <string name="databinding.tutorials.title">Databinding Examples</string>
    <string name="localization.tutorials.content">People's living, the book is dead,
        the living dead reading, can take the book to read. dead books read living,
        people reading can be put to death.</string>
</resources>

<!-- Chinese Ver -->
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app.name">LoxodonFramework</string>
    <string name="databinding.tutorials.title">数据绑定示例</string>
    <string name="localization.tutorials.content">人是活的，书是死的，活人读死书，可以把书读活。
        死书读活人，可以把人读死。</string>
</resources>

```

For more examples, refer to the Localization course in Tutorials.unity

Localization plugin supporting CSV format

If you are accustomed to using Excel, welcome to download my CSV plug-in, which supports CSV filereading with localized configuration. This plugin requires that the Unity version is above 2018 and supports .net 4.x or .net standard 2.0.

Download address: [Loxodon Framework Localization For CSV](#)

The configuration file format is as follows

- key: The key of the configuration file. It cannot be empty. This column must exist.
- type: The type of profile value. This column must exist. For example: string type string, integer array int-array
- description:description, can be empty, and this column can be omitted
- default: the default value, it is best not to be empty, if this column does not exist, the first column of the value will be used as the default column
- zh:Chinese configuration, the value of zh is taken from CultureInfo.TwoLetterISOLanguageName, if the field is empty, the default configuration is used
- zh-CN: China, Simplified Chinese configuration, zh-CN takes value from [CultureInfo.Name](#), if the field is empty, the configuration of zh is used

Only the key column and the type column are necessary, others can be added or omitted according to the actual situation.

******The localization query rules for values are queried based on the TwoLetterISOLanguageName and Name fields of the System.Globalization.CultureInfo class. Query by [CultureInfo.Name](#) first. If it does not exist, use CultureInfo.TwoLetterISOLanguageName to query, then the default value will be used.

For example, suppose the current language is zh-CN, the configuration of zh-CN will be used first. If the -CN column or zh-CN configuration is empty, the configuration of the zh column is used. If the zh column does not exist or the field is empty, the configuration of the default column is used.******

A	B	C	D	E	F
key	type	description	default	zh	zh-CN
app.name	string		LoxodonFramework	LoxodonFramework框架	LoxodonFramework框架
databinding.tutorials.title	string		Databinding Examples	数据绑定示例	数据绑定示例
databinding.tutorials.description	string		The current username is {0}, a	当前的用户名是:{0}, 正在输入的是:{1}	当前的用户名是:{0}, 正在输入的是:{1}
localization.tutorials.content	string		People's living, the book is	人是活的, 书是死的, 活人读死书, 可以把书读活。	人是活的, 书是死的, 活人读死书, 可以把书读活。

File encoding

If file contents Chinese, please make sure use UTF-8 encoding for CSV file. otherwise

If the file contains Chinese, please make sure that the CSV file using the utf-8 encoding, otherwise when the file conversion may be garbled words. While use the WPS export Xls file to CSV file, please review whether coding format for utf-8 encoding (can use notepad or EditPlus viewer).

Supported type and array representation

The CSV configuration also supports all the basic data types supported by the XML configuration in the previous section. The only difference is that the CSV file uses a comma separator to support the array type, as shown in the following table.

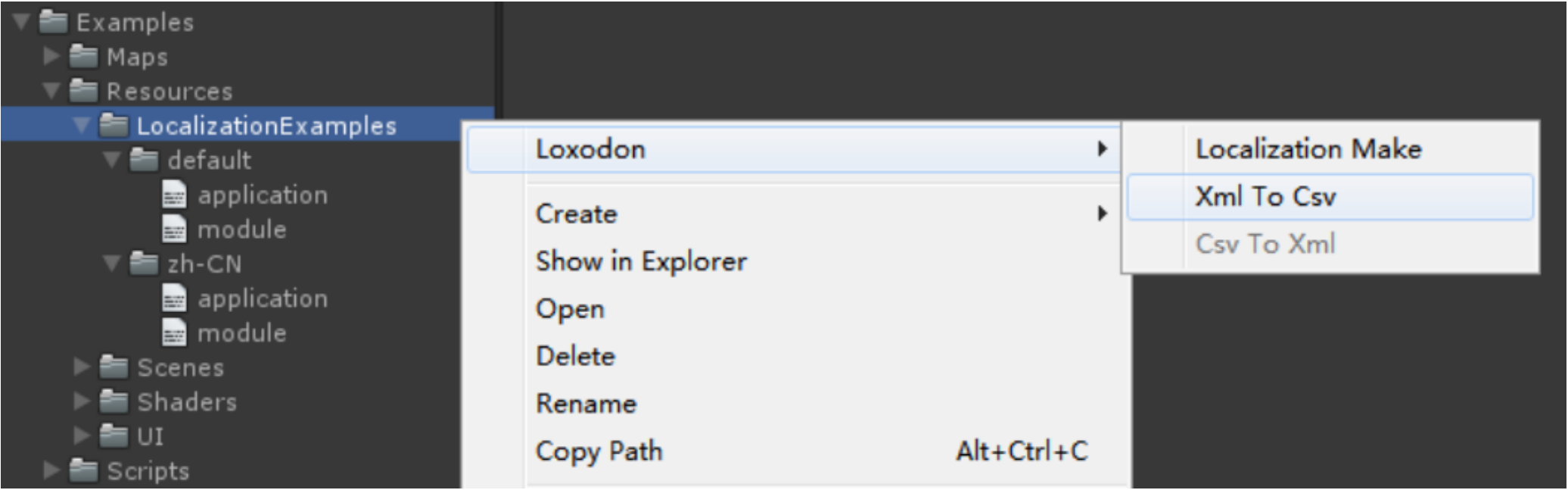
Note: Arrays are separated by commas, and commas between double quotes, single quotes, brackets (), brackets [], braces {}, and angle brackets <> are ignored, as in the array string There are commas, please use double or single quotes to enclose the string, otherwise an error will occur when the array is separated

key	type	us-EN
usernames	string-array	tom,clark,kate
chars	string-array	"a,b,c","d,e,f","g,h,i"
positions	vector3-array	(0,1,1.2),(2,2,0),(1.3,0.5,5)
colors	color-array	#FF0000,#00FF00

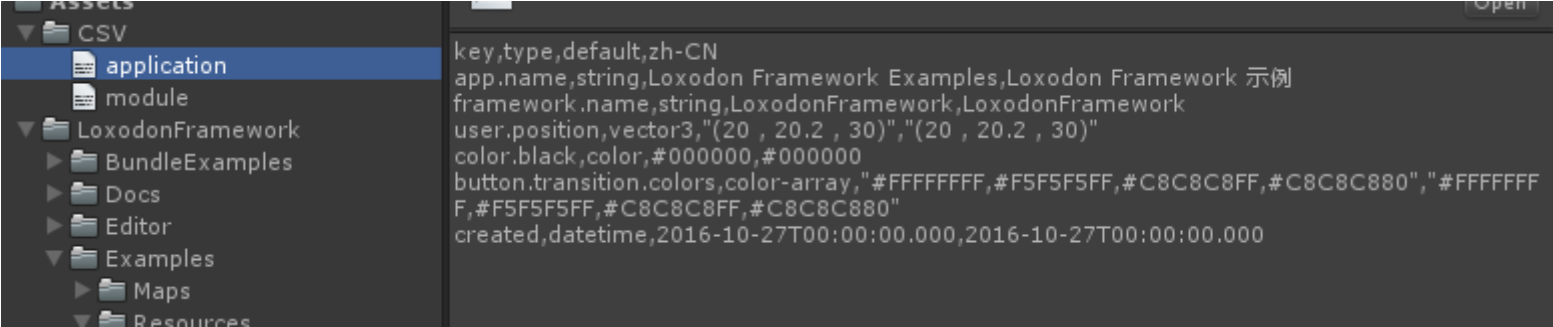
Conversion between XML and CSV

The XML configuration file and the CSV configuration file can be converted to each other, but you need to pay attention to the configuration of the array type. In the CSV, "," is used for division, and it is used to identify the division in XML. There may be an error while converting from "," to csv file format.

Select the root directory of the XML configuration file, right-click and select the Loxodon/Xml To Csv command, all xml files in the directory will be automatically converted into csv format files, and different language versions in XML will be combined into the same csv file. Similarly, the CSV file can also be converted into an XML file. If the CSV file contains configuration versions in multiple languages, it will be split into multiple XML files.



Generate csv file as follows



Sample of file

CSV source file

	A	B	C	D	E
1	key	type	zh-CN	default	en-US
2	application.data.version	version	1.0.1	1.0.0	1.0.0
3	application.colors	color-array	#000000,#0000FFFF	#000000,#FF000000	
4	application.game.name	string	WarCraft StarCraft Age of Empires	魔兽世界 星际争霸 帝国时代	WarCraft StarCraft Age of Empires
5	application.framework.author	string-array		clark,vovgou	
6	application.game.test.text	string	<a>This is a test.	<a>这是一个测试	<a>This is a test.
7	application.framework.descriptions	string-array	Author:Clark,""LoxodonFramework", 是一个轻量级MVVM框架,它使你的游戏开发更快速更容易.</item>	Author:Clark,"The "LoxodonFramework", is a lightweight MVVM framework,It makes your game development fa	
8					
9					
10					

Localized Chinese file while is converted to XML file format (the newline character after World of Warcraft, Starcraft, Age of Empires still exists, but is not visible).



Logging system

The framework provides a scalable logging system, which supports ALL, DEBUG, INFO, WARN, ERROR, FATAL and other levels. Different levels of log can be used to print during the development phase and release of the project.

I provide a debug version of Logging system in Unity3D, which basically meets general development and debugging needs, but if more powerful logging functions are needed, such as printing logs to a file system, mobile terminals printing logs to a computer via a LAN, etc. You can download my log plugin [Loxodon.Framework.Log4Net](#), which is a plugin implemented under Log4Net, which is very powerful.

Example of using the default logging system

```
// Set the logging level of the default logging system.
// Default log factories are automatically initialized
LogManager.Default.Level = Level.DEBUG

// If it is a custom logging implementation, it can be initialized as follows
DefaultLogFactory factory = new DefaultLogFactory();
factory.Level = Level.ALL
LogManager.Registry(factory)

// Define an ILOG for class AsyncResult
private static readonly ILog log = LogManager.GetLogger(typeof(AsyncResult));

// Print log
log.DebugFormat("My name is {0}",name)
```

StreamingAssets dirctory file reading (Android)

On the Android platform, the StreamingAssets directory is in the apk compressed package, so it cannot be accessed directly through the API of the C # file system. Loxodon.Framework.Utilities.FileUtil can be used to replace the C # File class to read these files. I also provide JNI to call the java interface to access it. The specific implementation is in FileUtil.Android.cs. File, but it can only read resources in apk file and can not read resources in obb file. If the obb package is split, use the implementation in FileUtil.Compression.cs or FileUtil.IonicZip.cs. FileUtil.Compression.cs uses the native decompression function in .net standard 2.0 to implement, which needs Unity2018 or above. FileUtil.IonicZip.cs is implemented using IonicZip's compression library, and please choose this version under the .net 3.5 library. To use this version, you need to find and copy IonicZip.dll in your project and configure the macro definition IONIC_ZIP in the Unity project.

Thread/Coroutine asynchronous results and asynchronous tasks

In order to facilitate the asynchronous calling of coroutines and threads, I designed a set of asynchronous results and asynchronous tasks according to the design pattern of Future / Promise. When using them, we can obtain the task execution results in a synchronous manner or through a callback. To get the results of the task, following the example below, let's understand the use of asynchronous results.

AsyncResult

Using AsyncResult, let's create a coroutine task that can be canceled, and obtain the execution result by synchronous blocking and callback respectively.

```

public class AsyncResultExample : MonoBehaviour
{

    protected IEnumerator Start ()
    {
        //*****Start the task and invoke the example synchronously*****//
        IAsyncResult<bool> result = StartTask();

        // Waiting for the task to complete,
        // the result.waitForDone () function returns an iterator, IEnumerator
        yield return result.WaitForDone ();

        if(r.Exception !=null)
        {
            Debug.LogFormat("Task fault: {0}",r.Exception);
        }
        else
        {
            Debug.LogFormat("Task success result = {0}",r.Result);
        }

        //*****Start the task and invoke the example with a callback*****//
        result = StartTask();
        result.Callbackable().OnCallback((r) =>
        {
            if(r.Exception !=null)
            {
                Debug.LogFormat("Task fault: {0}",r.Exception);
            }
            else
            {
                Debug.LogFormat("Task success result = {0}",r.Result);
            }
        }
        ));

    }

    // Creat a task
    public IAsyncResult<bool> StartTask()
    {
        // Create an asynchronous result with cancelable = true that supports cancellation
        AsyncResult<bool> result = new AsyncResult<bool> (true);

        // Task start
        this.StartCoroutine (DoTask (result));

        return result;
    }

    /// <summary>
    /// Simulate a task
    /// </summary>
    /// <returns>The task.</returns>
    /// <param name="promise">Promise.</param>
    protected IEnumerator DoTask (IPromise<bool> promise)
    {
        for (int i = 0; i < 20; i++) {

            // If the asyncResult.cancel () function is called externally,
            // then IsCancellationRequested = true here,
            // requesting the cancellation of the task
            if (promise.IsCancellationRequested) {
                promise.SetCancelled ();
                yield break;
            }
            yield return new WaitForSeconds (0.5f);
        }

        // The result must be set when the execution is complete
        promise.SetResult (true);
    }
}

```

ProgressResult

The ProgressResult is similar to the AsyncResult function, except that the task progress is increased. Let me take an example.

```

/// <summary>
/// Task schedule
/// </summary>
public class Progress
{
    public int bytes;
    public int TotalBytes;

    public int Percentage { get { return (bytes * 100) / TotalBytes; } }
}

public class ProgressResultExample : MonoBehaviour
{
    protected void Start()
    {
        // start a task
        IProgressResult<Progress, string> result = StartTask();

        // print task schedule
        result.Callbackable().OnProgressCallback(progress =>
        {
            Debug.LogFormat("Percentage: {0}% ", progress.Percentage);
        });

        // listen task result
        result.Callbackable().OnCallback(r =>
        {
            Debug.LogFormat("IsDone:{0} Result:{1}", r.IsDone, r.Result);
        });
    }

    public IProgressResult<Progress, string> StartTask()
    {
        ProgressResult<Progress, string> result = new ProgressResult<Progress, string>(true);

        this.StartCoroutine(DoTask(result));

        return result;
    }

    /// <summary>
    /// Simulate a progressive task
    /// </summary>
    /// <returns>The task.</returns>
    /// <param name="promise">Promise.</param>
    protected IEnumerator DoTask(IProgressPromise<Progress, string> promise)
    {
        int n = 50;
        Progress progress = new Progress();
        progress.TotalBytes = n;
        progress.bytes = 0;
        StringBuilder buf = new StringBuilder();
        for (int i = 0; i < n; i++)
        {
            /* If the task is cancelled, then stop the task */
            if (promise.IsCancellationRequested)
            {
                promise.SetCancelled();
                yield break;
            }

            progress.bytes += 1;
            buf.Append(" ").Append(i);
            promise.UpdateProgress(progress);/* update the progress of task. */
            yield return new WaitForSeconds(0.01f);
        }

        // The result must be set when the execution is complete
        promise.SetResult(buf.ToString());
    }
}

```

AsyncTask

An asynchronous task is an encapsulation of a thread task or a coroutine task. Passing an iterator `IEnumerator` to `AsyncTask` can create a coroutine task, or passing a delegate function to create a task that a background thread executes. According to the task execution process, a task

is divided into several stages before execution, after execution is successful / after execution fails, and execution ends. In each stage, you can register your own code block through a delegate callback. In the following example, let's see how to create a coroutine task.

```
public class AsyncTaskExample : MonoBehaviour
{
    protected IEnumerator Start()
    {
        AsyncTask task = new AsyncTask(DoTask(), true);

        /* Start task */
        task.OnPreExecute(() =>
        {
            // Invoked before the task executes
            Debug.Log("The task has started.");
        }).OnPostExecute(() =>
        {
            // Invoked after the task executes
            Debug.Log("The task has completed.");/* only execute successfully */
        }).OnError((e) =>
        {
            // The task failed to execute the call
            Debug.LogFormat("An error occurred:{0}", e);
        }).OnFinish(() =>
        {
            // When the task completes, it will be called whether it succeeds or fails
            Debug.Log("The task has been finished.");/* completed or error or canceled*/
        }).Start();

        // wait task finish
        yield return task.WaitForDone();

        Debug.LogFormat("IsDone:{0} IsCanceled:{1} Exception:{2}", task.IsDone, task.IsCancelled, task.Exception);
    }

    /// <summary>
    /// Simulate the execution of a task
    /// </summary>
    /// <returns>The task.</returns>
    /// <param name="promise">Promise.</param>
    protected IEnumerator DoTask()
    {
        int n = 10;
        for (int i = 0; i < n; i++)
        {
            yield return new WaitForSeconds(0.5f);
        }
    }
}
```

ProgressTask

ProgressTask is similar to AsyncTask in that it only increases the task progress. Similarly, ProgressTask can create a coroutine task or a background thread task.

```

public class ProgressTaskExample : MonoBehaviour
{
    protected IEnumerator Start()
    {
        // Create a task that will be executed in a background thread
        ProgressTask<float, string> task = new ProgressTask<float, string>(
            new Action<IPromise<float, string>>(DoTask), false, true);

        /* Start a task */
        task.OnPreExecute(() =>
        {
            // Called before the task executes
            Debug.Log("The task has started.");
        }).OnPostExecute((result) =>
        {
            // Called after successful execution of the task
            Debug.LogFormat("The task has completed. result:{0}", result);/* only execute successfully */
        }).OnProgressUpdate((progress) =>
        {
            // The progress of task execution
            Debug.LogFormat("The current progress:{0}%", (int)(progress * 100));
        }).OnError((e) =>
        {
            // Called after the task failed to execute
            Debug.LogFormat("An error occurred:{0}", e);
        }).OnFinish(() =>
        {
            // When the task completes, it will be called whether it succeeds or fails
            Debug.Log("The task has been finished.");/* completed or error or canceled*/
        }).Start();

        yield return task.WaitForDone();

        Debug.LogFormat("IsDone:{0} IsCanceled:{1} Exception:{2}", task.IsDone, task.IsCancelled, task.Exception);
    }

    /// <summary>
    /// To simulate a task, this is not an iterator,
    /// this will be executed in a background thread
    /// </summary>
    /// <returns>The task.</returns>
    /// <param name="promise">Promise.</param>
    protected void DoTask(IPromise<float, string> promise)
    {
        try
        {
            int n = 50;
            float progress = 0f;
            StringBuilder buf = new StringBuilder();
            for (int i = 0; i < n; i++)
            {
                /* If the task is cancelled, then stop the task */
                if (promise.IsCancellationRequested)
                {
                    promise.SetCancelled();
                    break;
                }

                progress = i / (float)n;
                buf.Append(" ").Append(i);
                promise.UpdateProgress(progress);/* update the progress of task. */
                Thread.Sleep(200);
            }
            promise.UpdateProgress(1f);
            promise.SetResult(buf.ToString()); /* update the result. */
        }
        catch (System.Exception e)
        {
            promise.SetException(e);
        }
    }
}

```

CoroutineTask

Before C # 4.0, you needed to perform a complex asynchronous operation, generally using thread pool technology to perform a task. Introduced the Task (System.Threading.Tasks.Task) mechanism in C # 4.0, which provides a more convenient and concise API. In order to keep Unity3D

coroutine asynchronous tasks and thread asynchronous tasks Task consistent, I implemented the CoroutineTask class Its API is basically the same as Task. The only difference is that it executes coroutine asynchronous tasks, and all its tasks are executed in the main thread.

```
public class CoroutineTaskExample : MonoBehaviour
{
    IEnumerator Start()
    {
        CoroutineTask task = new CoroutineTask(DoTask())
            .ContinueWith(
                DoContinueTask(),
                CoroutineTaskContinuationOptions.OnCompleted
                | CoroutineTaskContinuationOptions.OnFaulted
            ).ContinueWith(
                () => { Debug.Log("The task is completed"); }
            );

        yield return task.WaitForDone();

        Debug.LogFormat("IsDone:{0} IsCompleted:{1} IsFaulted:{2} IsCancelled:{3}",
            task.IsDone, task.IsCompleted, task.IsFaulted, task.IsCancelled);
    }

    /// <summary>
    /// Simulate a task.
    /// </summary>
    /// <returns>The task.</returns>
    /// <param name="promise">Promise.</param>
    protected IEnumerator DoTask()
    {
        int n = 10;
        for (int i = 0; i < n; i++)
        {
            Debug.LogFormat("Task:i = {0}", i);
            yield return new WaitForSeconds(0.5f);
        }
    }

    protected IEnumerator DoContinueTask()
    {
        int n = 10;
        for (int i = 0; i < n; i++)
        {
            Debug.LogFormat("ContinueTask:i = {0}", i);
            yield return new WaitForSeconds(0.5f);
        }
    }
}
```

For more examples, refer to the Basic Tutorials.unity

Async & Await

async & await in C#

Since the release of Unity2017, it is already possible to use the new C# features of async and await [using.NET 4.x](#) or [.NET Standard 2.0](#) libraries. The framework extends the GetAwaiter() function for IEnumerator, YieldInFormaldeference, CustomYieldInFormaldeference, AsyncOperation, IAsyncResult, CoroutInetAsk and so on to support the async-await feature. Also add WaitForMainThread and WaitForBackgroundThread classes to switch worker threads for code fragments.

Example one: async and await using method


```

public class AsyncAndAwaitExample : MonoBehaviour
{
    async void Start()
    {
        await new WaitForSeconds(2f);
        Debug.Log("WaitForSeconds End");

        await Task.Delay(1000);
        Debug.Log("Delay End");

        UnityWebRequest www = UnityWebRequest.Get("http://www.baidu.com");
        await www.SendWebRequest();
        Debug.Log(www.downloadHandler.text);

        int result = await Calculate();
        Debug.LogFormat("Calculate Result = {0} Calculate Task End", result);

        await new WaitForSecondsRealtime(1f);
        Debug.Log("WaitForSecondsRealtime End");

        await DoTask(5);
        Debug.Log("DoTask End");
    }

    IAsyncResult<int> Calculate()
    {
        return Executors.RunAsync<int>(() =>
        {
            Debug.LogFormat("Calculate Task ThreadId:{0}", Thread.CurrentThread.ManagedThreadId);
            int total = 0;
            for (int i = 0; i < 20; i++)
            {
                total += i;
                try
                {
                    Thread.Sleep(100);
                }
                catch (Exception) { }
            }
            return total;
        });
    }

    IEnumerator DoTask(int n)
    {
        yield return new WaitForSeconds(1f);

        for (int i = 0; i < n; i++)
        {
            yield return null;
        }
    }
}

```

Example two: Within the function, the main thread and background thread can be switched between `WaitForBackgroundThread` and `WaitForMainThread`, and different code fragments can be executed in different threads.

```

using Loxodon.Framework.Asynchronous;
//Extends the namespace in which the GetAwaiter() function resides

using System.Threading;
using System.Threading.Tasks;
public class AsyncAndAwaitSwitchThreadsExample : MonoBehaviour
{
    async void Start()
    {
        //Unity Thread
        Debug.LogFormat("1. ThreadID:{0}",Thread.CurrentThread.ManagedThreadId);

        await new WaitForBackgroundThread();

        //Background Thread
        Debug.LogFormat("2.After the WaitForBackgroundThread.ThreadID:{0}", Thread.CurrentThread.ManagedThreadId);

        await new WaitForMainThread();

        //Unity Thread
        Debug.LogFormat("3.After the WaitForMainThread.ThreadID:{0}", Thread.CurrentThread.ManagedThreadId);

        await Task.Delay(3000).ConfigureAwait(false);

        //Background Thread
        Debug.LogFormat("4.After the Task.Delay.ThreadID:{0}", Thread.CurrentThread.ManagedThreadId);

        await new WaitForSeconds(1f);

        //Unity Thread
        Debug.LogFormat("5.After the WaitForSeconds.ThreadID:{0}", Thread.CurrentThread.ManagedThreadId);
    }
}

```

See Async & Await Tutorials. Unity for more examples

Coroutine from Task to Unity

The framework extends the AsCoroutine() function for tasks to support the coroutine from Task to Unity. See the below example.

```

using Loxodon.Framework.Asynchronous;
//Extends the namespace in which the AsCoroutine() function resides
public class TaskToCoroutineExample : MonoBehaviour
{
    IEnumerator Start()
    {
        Task task = Task.Run(() =>
        {
            for (int i = 0; i < 5; i++)
            {
                try
                {
                    Thread.Sleep(200);
                }
                catch (Exception) { }

                Debug.LogFormat("Task ThreadId:{0}", Thread.CurrentThread.ManagedThreadId);
            }
        });

        yield return task.AsCoroutine();
        Debug.LogFormat("Task End,Current Thread ID:{0}", Thread.CurrentThread.ManagedThreadId);

        yield return Task.Delay(1000).AsCoroutine();
        Debug.LogFormat("Delay End");
    }
}

```

async & await in Lua

In order to keep Lua development in sync with C# development, I have also added support for async & await in Lua, and made sure that C# and Lua can call each other.

Async is a function in Lua, which can only take one input parameter and must be a function. Async wraps the input function as a Lua coroutine and returns a wrapped function. An async input function can be either an argument or a no-argument function. The function can have one or more

return values or no return values.

Await is a function in Lua. The input parameter of await must be an AsyncTask object, or any asynchronous result that implements getAwaiter() function. Both C# objects and Lua objects are supported, so the input parameters of await can be from not only Task and UniTask in C# but also asynchronous results in Unity. The await function shall listens for the callback of the asynchronous result and suspends the current coroutine. When the asynchronous task completes, the callback will trigger the coroutine to continue execution. Await also supports asynchronous results with no return value, single return value or multiple return values.

The Async & await function is defined in the AsyncTask module, and which can be used after the AsyncTask module is imported in the Lua file via require.

For an example, the following Lua class is hanged under LuabeHaviour and automatically call the start function through LuabeHaviour.

```
require("framework.System")
local AsyncTask = require("framework.AsyncTask")
-- Import the AsyncTask module and import the async, await, and try functions

local M=class("Example",target)

-- Define the position function with input parameter xyz
-- and return the AsyncTask asynchronous object
-- Async supports functions with multiple return values
M.position = async(function(x,y,z)
    return x/1000.0,y/1000.0,z/1000.0
end)

M.start = async(function(self)
    await(AsyncTask.Delay(1000)) --Delay 1000 milliseconds

    local x,y,z = await(M.position(1200,500,240))
    -- Call position function asynchronously, returning x,y, and z

    printf("x=%s y=%s z=%s",x,y,z)

    -- call Resources.LoadAsync asynchronously
    local goTemplate = await(CS.UnityEngine.Resources.LoadAsync("Prefabs/Cube",typeof(CS.UnityEngine.GameObject)))

    local go = GameObject.Instantiate(goTemplate)

    go.transform.localPosition = CS.UnityEngine.Vector3.zero;
end)
```

C# call async function of Lua

I have implemented the Iluataask interface in C#, which can automatically convert from the AsyncTask object to Iluataask object.

Start() returns an AsyncTask Lua object. refer the below C# code.

```

public class LuaBehaviour : MonoBehaviour, ILuaExtendable
{
    protected LuaTable metatable;
    protected Func<MonoBehaviour, ILuaTask> onStart;

    protected virtual void Awake()
    {
        ...

        metatable = (LuaTable)result[0];
        // Call start function of Lua.
        // This function can be either an asynchronous function wrapped by async or a normal function
        onStart = metatable.Get<Func<MonoBehaviour, ILuaTask>>("start");
    }

    protected virtual async void Start()
    {
        if (onStart != null)
        {
            // if start is an asynchronous function wrapped by async,
            // it will return the ILuataask object, otherwise it will return null
            ILuaTask task = onStart(this);
            if (task != null)
                await task;
        }
    }
}

```

try/catch/finally of Lua

In conjunction with async and await, we wrap Lua's xpcall functions with a try function to catch exceptions in Lua's functions.

The input parameters of the try function are a Lua table with the following structure: t[0] is the main function, t.Catch is the catch function, and t.fupward is the finally function.

```

{
    function()
        --this is main function.
    end,
    catch=function(e)
        --this is catch function
    end,
    finally =function()
        --this is finally function
    end
}

```

example of try/catch

```

local position = async(function(x,y,z)

    --try is a function. If the return value is required, add return before try;
    --Otherwise, the return is not required to be added.

    return try{
        function()
            --this is main function.
            error("This a test,throw an exception")
            return x/1000.0,y/1000.0,z/1000.0
        end,
        catch=function(e)
            --this is catch function
            printf("Catch exception:%s",e)
            return 0,0,0 --The default value is returned when an exception occurs
        end,
        finally =function()
            --this is finally function
            print("Execute the finally block")
        end
    }
end)

```

Thread/Coroutine Executor

In the development of Unity3d logic scripts, multi-threading is not supported. All UnityEngine.Object objects can only be accessed and modified in the main thread. However, during game development, it is difficult to avoid using multi-threaded programming. For example, it accepts data from the network through a Socket connection, downloads resources through multiple threads, and some pure CPU calculation logic is switched to a background thread to perform operations. There will be a problem of thread switching. So in the Loxodon.Framework framework, I designed a thread and coroutine executor to be used with the task results in the previous article. It can easily switch the task to the main thread for execution, and it can also easily start a background thread task.

Executors

```
public class ExecutorExample : MonoBehaviour
{
    IEnumerator Start()
    {
        //Run a task asynchronously in a background thread
        Executors.RunAsync(() =>
        {
            Debug.LogFormat("RunAsync ");
        });

        //Run a task asynchronously in a background thread
        Executors.RunAsync(() =>
        {
            //sleep 1000 micro second
            Thread.Sleep(1000);

            //Switch from the background thread to the main thread,
            //waitForExecution = true, the current function can be return
            //until main thread execute complete
            Executors.RunOnMainThread(() =>
            {
                Debug.LogFormat("RunOnMainThread Time:{0} frame:{1}", Time.time, Time.frameCount);
            }, true);
        });

        //Run a coroutine task
        IAsyncResult result = Executors.RunOnCoroutine(DoRun());

        //waiting for task complete
        yield return result.WaitForDone();
    }

    IEnumerator DoRun()
    {
        for (int i = 0; i < 10; i++)
        {
            Debug.LogFormat("i = {0}", i);
            yield return null;
        }
    }
}
```

Scheduled Task Executor(IScheduledExecutor)

In this framework, a threaded timed task executor (ThreadScheduledExecutor) and a Unity3D coroutine timed task executor (CoroutineScheduledExecutor) are provided. Below we take the threaded timed task executor as an example to introduce its usage.

```
//Creates and starts a thread's timer task executor
var scheduled = new ThreadScheduledExecutor();
scheduled.Start();

//After a delay of 1000 micro seconds,
//print "This is a test." every 2000 micro seconds at a fixed frequency.
IAsyncResult result = scheduled.ScheduleAtFixedRate(() =>
{
    Debug.Log("This is a test.");
}, 1000, 2000);
```

Interceptable Enumerator(InterceptableEnumerator)

In Unity3D coroutines, if an exception occurs, it is impossible to catch the exception. Try catch is not allowed to be used across yield, finally cannot ensure that the code block can still be executed when the coroutine ends abnormally, so it is often impossible to know a coroutine. It is not convenient to find the cause if the program execution ends normally. According to the principle that the Unity3D coroutine is an iterator, I designed

a interceptable iterator that can inject code blocks during the execution of the coroutine and catch exceptions. Using `InterceptableEnumerator` to wrap the original iterator, you can capture the execution exception of the coroutine code, and no matter whether the coroutine ends normally, you can insert a code block before the coroutine exits, ensuring that this code block will definitely be at the end of the coroutine. carried out. In my `Executors`, I use the `InterceptableEnumerator` to ensure that the task ends normally. No matter whether the coroutine is executed successfully or abnormally, I can set the result of the `AsyncResult` by registering the `Finally` block. Ensure that `AsyncResult.IsDone` is equal to `true`, which will not cause the task `Stuck`.

`InterceptableEnumerator` supports conditional statement blocks. You can insert a conditional statement block outside to control the coroutine logic or abort the coroutine. Exception statement block, you can catch coroutine exceptions, `Finally` statement block, to ensure that the end of the coroutine will call this statement block. Let's take a look at an example.

```
/// <summary>
/// This is the wrapper function for an iterator
/// </summary>
protected static InterceptableEnumerator WrapEnumerator(IEnumerator routine, IPromise promise)
{
    InterceptableEnumerator enumerator;
    if(routine is InterceptableEnumerator)
        enumerator = (InterceptableEnumerator)routine;
    else
        enumerator = new InterceptableEnumerator(routine);

    //Registers a conditional statement block, and if the task is canceled,
    //isCancellationRequested = true, then terminates the task
    enumerator.RegisterConditionBlock(() => !(promise.IsCancellationRequested));

    //Registrates an exception-catching statement block
    //that assigns the exception to the task result and prints the error
    //if the coroutine executes an error
    enumerator.RegisterCatchBlock(e =>
    {
        if (promise != null)
            promise.SetException(e);

        if (log.IsErrorEnabled)
            log.Error(e);
    });

    //Register a Finally block to ensure that the task can exit properly
    enumerator.RegisterFinallyBlock(() =>
    {
        if (promise != null && !promise.IsDone)
        {
            if (promise.GetType().IsSubclassOfGenericTypeDefintion(typeof(IPromise<>)))
                promise.SetException(new Exception("No value given the Result"));
            else
                promise.SetResult();
        }
    });
    return enumerator;
}
```

For more examples, see the `Basic Tutorials.unity`

Message System(Messenger)

`Messenger` is used for communication between application modules. It provides the function of message subscription and publishing. `Messenger` supports subscribing and publishing messages by message type, and subscribing and publishing messages by channel.

```

public class MessengerExample : MonoBehaviour
{
    private IDisposable subscription;
    private IDisposable chatroomSubscription;
    private void Start()
    {
        //get default Messenger
        Messenger messenger = Messenger.Default;

        //Subscribe to a message, ensuring that Subscription is a member variable,
        //otherwise it will automatically unsubscribe when it is collected by GC
        subscription = messenger.Subscribe((PropertyChangedMessage<string> message) =>
        {
            Debug.LogFormat("Received Message:{0}", message);
        });

        //Publish a message about a property name change
        messenger.Publish(new PropertyChangedMessage<string>("clark", "tom", "Name"));

        //Subscribe to the chat channel "ChatRoom1"
        chatroomSubscription = messenger.Subscribe("chatroom1", (string message) =>
        {
            Debug.LogFormat("Received Message:{0}", message);
        });

        //send a message to chat channel "chatroom1"
        messenger.Publish("chatroom1", "hello!");
    }

    private void OnDestroy()
    {
        if (this.subscription != null)
        {
            //Unsubscribe message
            this.subscription.Dispose();
            this.subscription = null;
        }

        if (this.chatroomSubscription != null)
        {
            //Unsubscribe message
            this.chatroomSubscription.Dispose();
            this.chatroomSubscription = null;
        }
    }
}

```

For more examples, see the Basic Tutorials.unity

Observables

ObservableObject, ObservableList, ObservableDictionary are indispensable in the data binding of the MVVM framework. They respectively implement the INotifyPropertyChanged and INotifyCollectionChanged interfaces. When the property of an object changes or the item in the collection changes, we can monitor the PropertyChanged and CollectionChanged events. Received notifications of property changes and collection changes. In the data binding function, only objects that implement these two interfaces will automatically notify the UI view of changes when the properties or collections change. Otherwise, you can only give the UI the initial binding. The control is assigned once, and the value of the view model is changed after binding, and the UI control cannot be notified of the change.

Let's take a look at the usage example of ObservableDictionary. When we need to create a custom ListView control, we need to understand its principle.

```

public class ObservableDictionaryExample : MonoBehaviour
{
    private ObservableDictionary<int, Item> dict;

    protected void Start()
    {
#if UNITY_IOS
        //In IOS, the generic type dictionary needs to provide the IEqualityComparer <TKey>,
        //otherwise it may be a JIT exception
        this.dict = new ObservableDictionary<int, Item>(new IntEqualityComparer());
#else
        this.dict = new ObservableDictionary<int, Item>();
#endif

        dict.CollectionChanged += OnCollectionChanged;

        //add Item
        dict.Add(1, new Item() { Title = "title1", IconPath = "xxx/xxx/icon1.png", Content = "this is a test." });
        dict.Add(2, new Item() { Title = "title2", IconPath = "xxx/xxx/icon2.png", Content = "this is a test." });

        //delete Item
        dict.Remove(1);

        //remove dictionary
        dict.Clear();
    }

    protected void OnDestroy()
    {
        if (this.dict != null)
        {
            this.dict.CollectionChanged -= OnCollectionChanged;
            this.dict = null;
        }
    }

    //Set changing event
    protected void OnCollectionChanged(object sender, NotifyCollectionChangedEventArgs eventArgs)
    {
        switch (eventArgs.Action)
        {
            case NotifyCollectionChangedAction.Add:
                foreach (KeyValuePair<int, Item> kv in eventArgs.NewItems)
                {
                    Debug.LogFormat("ADD key:{0} item:{1}", kv.Key, kv.Value);
                }
                break;
            case NotifyCollectionChangedAction.Remove:
                foreach (KeyValuePair<int, Item> kv in eventArgs.OldItems)
                {
                    Debug.LogFormat("REMOVE key:{0} item:{1}", kv.Key, kv.Value);
                }
                break;
            case NotifyCollectionChangedAction.Replace:
                foreach (KeyValuePair<int, Item> kv in eventArgs.OldItems)
                {
                    Debug.LogFormat("REPLACE before key:{0} item:{1}", kv.Key, kv.Value);
                }
                foreach (KeyValuePair<int, Item> kv in eventArgs.NewItems)
                {
                    Debug.LogFormat("REPLACE after key:{0} item:{1}", kv.Key, kv.Value);
                }
                break;
            case NotifyCollectionChangedAction.Reset:
                Debug.LogFormat("RESET");
                break;
            case NotifyCollectionChangedAction.Move:
                break;
        }
    }
}

```

For more examples, see the [Basic Tutorials](#).unity

Databinding

Data binding is the key technology of MVVM. It is used to bind and connect the view with the view model. The connection between the view and the view model can be two-way or one-way. The data of the view model can be changed through data binding. The function automatically notifies the view change, and the same view change can also notify the view model value to change. In addition to the connection of values, data binding can also support the binding of events, methods, and commands. Data binding exists in the framework as a service module. It consists of many functional components, such as data binding context, type converters, expression parsers, path parsers, object and method agents, properties, and fields. Visitor, etc. The data binding service is optional. It is necessary only when the framework's view module is used and the UI is developed using MVVM. Of course, you can not use the view module of this framework, but only use the data binding service.

The data binding service is a basic component. We can start the data binding service in the game initialization script and register all the components in the service container of the global context. If a friend wants to use third-party IoC components, such as Autofac, Zenject, etc., then they need to refer to the code of the BindingServiceBundle and create all the classes initialized in the OnStart function with other containers.

```
//Obtain the global context
ApplicationContext context = Context.GetApplicationContext();

//Initialize the data binding service
BindingServiceBundle bindingService = new BindingServiceBundle(context.GetContainer());
bindingService.Start();
```

If the Lua plugin is installed, when writing a game in Lua, the data binding service is initialized as follows. LuaBindingServiceBundle adds components that support Lua objects.

```
//Obtain the global context
ApplicationContext context = Context.GetApplicationContext();

//Initialize the data binding service
LuaBindingServiceBundle bundle = new LuaBindingServiceBundle(context.GetContainer());
bundle.Start();
```

Data binding example

```
//Create a data binding collection with the generic parameter
//DataBindingExample for the view and AccountViewModel for the view model
BindingSet<DataBindingExample, AccountViewModel> bindingSet;
bindingSet = this.CreateBindingSet<DataBindingExample, AccountViewModel>();

//Bind the property of Text.text to Account.Username. OneWay is one-way.
//Assign the value of Account.Username to the UI control
bindingSet.Bind(this.username).For(v => v.text).To(vm => vm.Account.Username).OneWay();

// Bind the property of Username by InputField.text, bind in both ways,
// modify Username, automatically update InputField control,
// modify InputField automatically update Username property
bindingSet.Bind(this.usernameEdit).For(v => v.text, v => v.onEndEdit).To(vm => vm.Username).TwoWay();

//Bind Button to the OnSubmit method of view model. The direction property is invalid
bindingSet.Bind(this.submit).For(v => v.onClick).To(vm => vm.OnSubmit);

bindingSet.Build();
```

Binding mode

- **OneWay**(View <-- ViewModel)
One-way binding, only the view model can modify the value of the UI control in the view. The ViewModel must inherit the INotifyPropertyChanged interface, and the PropertyChanged event will be triggered when the property value changes. Otherwise, the effect is consistent with OneTime, and only the initialization binding is assigned once. For example, Field is only valid for the first time.
- **TwoWay**(View <--> ViewModel)
Two-way binding, view control modification will automatically modify the view model, and view model modification will automatically modify the view control. The ViewModel must support the PropertyChanged event, and the UI control must support the onEndEdit event and be bound to the onEndEdit event.
- **OneTime**(View <-- ViewModel)
It is only assigned once, and the value of the ViewModel is assigned to the view control only when the binding relationship is initialized.
- **OneWayToSource**(View --> ViewModel)
One-way binding, the opposite direction to OneWay, can only assign view UI controls to properties of the view model.

Type converter(IConverter)

In general, basic data types are automatically converted when the field type of the view control is inconsistent with the field type of the view model, unless a custom type converter is required to support it. However, when you modify the picture or atlas on the view control through the picture path, picture name, or atlas name saved in the view model, you must use a type converter to convert it.

```
//Load a Sprite atlas
Dictionary<string, Sprite> sprites = new Dictionary<string, Sprite>();
foreach (var sprite in Resources.LoadAll<Sprite>("EquipTextures"))
{
    if (sprite != null)
        sprites.Add(sprite.name, sprite);
}

//Create a converter that supports the Sprite name to the Sprite
var spriteConverter = new SpriteConverter(sprites);

//Obtain the converter registration service, which is automatically created and injected
//into the context container when the data binding service starts
IConverterRegistry converterRegistry = context.GetContainer().Resolve<IConverterRegistry>();

//Register Sprite Converter
converterRegistry.Register("spriteConverter", spriteConverter);

//Using the view model Icon, change the Sprite name,
//convert it to the corresponding Sprite through the SpriteConverter,
//and assign the value to the Sprite property of the image.

bindingSet.Bind(this.image).For(v => v.sprite).To(vm => vm.Icon).WithConversion("spriteConverter").OneWay();
```

For more examples, see the [ListView And Sprite Databinding Tutorials](#).unity

Binding type

- **Property and Field Binding**

Property and Field binding is very simple, see the example directly

```
//C#, one-way bindiung
bindingSet.Bind(this.username).For(v => v.text).To(vm => vm.Account.Username).OneWay();

//C# Bidirectional binding. When bidirectional binding occurs,
// the view object must support events that change the view,
// such as "onEndEdit", which must be configured in the For function
bindingSet.Bind(this.usernameEdit).For(v => v.text, v => v.onEndEdit).To(vm => vm.Username).TwoWay();

//C#, 非拉姆达表达式的方式
bindingSet.Bind (this.username).For ("text").To ("Account.Username").OneWay ();

--Lua, 非拉姆达表达式参数的版本
bindingSet:Bind(self.username):For("text"):To("account.username"):OneWay()
bindingSet:Bind(self.errorMessage):For("text"):To("errors['errorMessage']"):OneWay()
```

- **Expression binding**

Expression binding only supports one or more properties of the view model. Values converted to a certain type by expressions are assigned to view UI controls, which can only be of the type OneTime or OneWay. An expression binding function supports two configuration modes of lambda expression parameters and string parameters. C # code only supports the method of lambda expression parameters. The code will automatically analyze one or more attributes of the view model that the expression is concerned with. Automatically listen for changes in these attributes; Lua code only supports the method using the string parameter version. It cannot automatically analyze which attributes of the view model are used. You need to configure the attributes used by the expression in the parameters.

```
//C#代码, 使用拉姆达表达式为参数的ToExpression方法, 自动分析监听视图模型的Price属性
bindingSet.Bind(this.price).For(v => v.text).ToExpression(vm => string.Format("${0:0.00}", vm.Price)).OneWay();

--Lua代码, 使用string参数版本的ToExpression方法, 需要手动配置price属性, 如果表达式使用了vm的多个属性,
--则在"price"后继续配置其他属性
bindingSet:Bind(self.price):For("text"):ToExpression(function(vm)
    return string.format(tostring("%0.2f"), vm.price)
end , "price"):OneWay()
```

- **Method binding**

Method binding is similar to property binding. It also supports two versions of lambda expressions and string parameters. Method binding must ensure that the event parameter type of the control is the same as the parameter type of the bound method of the view model.

```
//C#, 拉姆达表达式方式的绑定, Button.onClick 与视图模型的成员OnSubmit方法绑定
bindingSet.Bind(this.submit).For(v => v.onClick).To(vm => vm.OnSubmit);
```

```
//C#, 拉姆达表达式方式的绑定, 如果方法带参数, 请在To后面加上泛型约束
bindingSet.Bind(this.emailEdit).For(v => v.onValueChanged).To<string>(vm => vm.OnEmailValueChanged);
```

```
--Lua, 通过字符串参数绑定, Button.onClick 与视图模型的成员submit方法绑定
bindingSet.Bind(self.submit):For("onClick"):To("submit"):OneWay()
```

- **Command and interactive request binding**

A command is a wrapper for a view model method. The binding of a general UI button onClick can be either a method of the view model or a command of the view model. However, it is recommended to bind to the command. The command can not only respond to the click event of the button, but also control the clickable state of the button. The button can be grayed out immediately after the button is pressed, and the button state can be restored after the button event response is completed.

Interaction Request (InteractionRequest) Interaction requests are often paired with commands. Commands respond to UI click events, handle click logic, and interact with requests to the control layer to control the creation, modification, and destruction of UI messages.

```
//C#, 绑定控制层的OnOpenAlert函数到交互请求AlertDialogRequest上
bindingSet.Bind().For(v => this.OnOpenAlert).To(vm => vm.AlertDialogRequest);
```

```
//绑定Button的onClick事件到OpenAlertDialog命令上
bindingSet.Bind(this.openAlert).For(v => v.onClick).To(vm => vm.OpenAlertDialog);
```

- **Collection binding**

The dictionary and list binding is basically the same as the property/field binding, see the following code

```
//C#, 绑定一个Text.text属性到一个字典ObservableDictionary中key ="errorMessage" 对应的对象
bindingSet.Bind(this.errorMessage).For(v => v.text).To(vm => vm.Errors["errorMessage"]).OneWay();
```

- **Static class binding**

The only difference between static class binding and view model binding is that static class binding creates a static binding set. The static binding set does not require a view model object.

```
//C#, 创建一个静态类的绑定集
BindingSet<DataBindingExample> staticBindingSet = this.CreateBindingSet<DataBindingExample>();

//绑定标题到类Res的一个静态变量databinding_tutorials_title
staticBindingSet.Bind(this.title).For(v => v.text).To(() => Res.databinding_tutorials_title).OneWay();
```

- **Localized data binding**

For localized data binding, please use the static binding set ToValue () function binding. First, obtain the IObservableProperty object through Localization.GetValue (). This is an observable property. When the language is switched, you will receive a notification of the value change. ToValue function binding, see the example below.

```
//C#, 创建一个静态类型的绑定集
BindingSet<DataBindingExample> staticBindingSet = this.CreateBindingSet<DataBindingExample>();

var localization = Localization.Current;

//通过本地化key获得一个IObservableProperty属性,
//必须是IObservableProperty类型, 否则切换语言不会更新
var value = localization.GetValue("databinding.tutorials.title"); //OK
//var value = localization.Get<string>("databinding.tutorials.title"); //NO
staticBindingSet.Bind(this.title).For(v => v.text).ToValue(value).OneWay();
```

Command Parameter

Binding from event to command (ICommand) or method supports custom parameters. Use Command Parameter to add a custom parameter to a UI event without parameters (such as the Click event of a Button). If the UI event itself has parameters, it will be commanded Parameter override. Use Command Parameter to easily bind the Click events of multiple Buttons to the same function OnClick (int buttonNo) of the view model. Please ensure that the parameter type of the function matches the command parameter, otherwise it will cause an error. Please refer to the example below for details.

In the example, the Click event of a group of Button buttons is bound to the OnClick function of the view model. You can know which button is currently pressed by the parameter buttonNo.

```

public class ButtonGroupViewModel : ViewModelBase
{
    private string text;
    private readonly SimpleCommand<int> click;
    public ButtonGroupViewModel()
    {
        this.click = new SimpleCommand<int>(OnClick);
    }

    public string Text
    {
        get { return this.text; }
        set { this.Set<string>(ref text, value, "Text"); }
    }

    public ICommand Click
    {
        get { return this.click; }
    }

    public void OnClick(int buttonNo)
    {
        Executors.RunOnCoroutineNoReturn(DoClick(buttonNo));
    }

    private IEnumerator DoClick(int buttonNo)
    {
        this.click.Enabled = false;
        this.Text = string.Format("Click Button:{0}.Restore button status after one second", buttonNo);
        Debug.LogFormat("Click Button:{0}", buttonNo);

        //Restore button status after one second
        yield return new WaitForSeconds(1f);
        this.click.Enabled = true;
    }
}

```

```

protected override void Start()
{
    ButtonGroupViewModel viewModel = new ButtonGroupViewModel();

    IBindingContext bindingContext = this.BindingContext();
    bindingContext.DataContext = viewModel;

    /* databinding */
    BindingSet<DatabindingForButtonGroupExample, ButtonGroupViewModel> bindingSet;
    bindingSet = this.CreateBindingSet<DatabindingForButtonGroupExample, ButtonGroupViewModel>();
    bindingSet.Bind(this.button1).For(v => v.onClick).To(vm => vm.Click).CommandParameter(1);
    bindingSet.Bind(this.button2).For(v => v.onClick).To(vm => vm.Click).CommandParameter(2);
    bindingSet.Bind(this.button3).For(v => v.onClick).To(vm => vm.Click).CommandParameter(3);
    bindingSet.Bind(this.button4).For(v => v.onClick).To(vm => vm.Click).CommandParameter(4);
    bindingSet.Bind(this.button5).For(v => v.onClick).To(vm => vm.Click).CommandParameter(5);

    bindingSet.Bind(this.text).For(v => v.text).To(vm => vm.Text).OneWay();

    bindingSet.Build();
}

```

Scope Key

In some views, you may need to dynamically create binding relationships and dynamically remove binding relationships. Here we provide a way to remove binding relationships in batches, which is Scope Key.

```

//C#,
string scopeKey = "editKey";
bindingSet.Bind(this.username).For(v => v.text).To(vm => vm.Account.Username).WithScopeKey(scopeKey).OneWay();
bindingSet.Bind(this.submit).For(v => v.onClick).To(vm => vm.OnSubmit()).WithScopeKey(scopeKey);

//Remove bindings by Scope Key
this.ClearBindings(scopeKey); //or this.BindingContext().Clear(scopeKey)

```

Binding life cycle

Generally, the data binding is initialized in the view creation function. The binding relationship between the view control and the view model is configured through the BindingSet. When the Build function of the BindingSet is called, Binder creates all the binding relationship pairs in the BindingSet. The created binding pair will be saved in the BindingContext of the current view. The BindingContext is automatically created when it is first called, and a BindingContextLifecycle script is automatically generated. It is hung on the current view object, which controls the life cycle of the BindingContext. When the view is destroyed, the BindingContext is destroyed along with it. The bindings stored in the BindingContext Relationships will also be destroyed.

Accessors for properties and fields

In the IOS platform, JIT compilation is not allowed, and dynamic code generation is not allowed. When data binding functions access object properties, fields, and methods, they cannot be accessed through dynamic generation delegation like other platforms. They can only be accessed through reflection. The efficiency of reflection is well known. It's bad, so I provide the ability to statically inject accessors to bypass reflection. By default, I have created property accessors for some classes of the UGUI and Unity engines. With reference to my code, you can also register the accessors of common properties of the view model class to the type proxy.

```

public class UnityProxyRegister
{
    [RuntimeInitializeOnLoadMethod(RuntimeInitializeLoadType.BeforeSceneLoad)]
    static void Initialize()
    {
        Register<Transform, Vector3>("localPosition", t => t.localPosition, (t, v) => t.localPosition = v);
        Register<Transform, Vector3>("eulerAngles", t => t.eulerAngles, (t, v) => t.eulerAngles = v);
        Register<Transform, Vector3>("localEulerAngles", t => t.localEulerAngles, (t, v) => t.localEulerAngles = v);
        Register<Transform, Vector3>("right", t => t.right, (t, v) => t.right = v);
        Register<Transform, Vector3>("up", t => t.up, (t, v) => t.up = v);
        Register<Transform, Vector3>("forward", t => t.forward, (t, v) => t.forward = v);
        Register<Transform, Vector3>("position", t => t.position, (t, v) => t.position = v);
        Register<Transform, Vector3>("localScale", t => t.localScale, (t, v) => t.localScale = v);
        Register<Transform, Vector3>("lossyScale", t => t.lossyScale, null);
        Register<Transform, Quaternion>("rotation", t => t.rotation, (t, v) => t.rotation = v);
        Register<Transform, Quaternion>("localRotation", t => t.localRotation, (t, v) => t.localRotation = v);
        Register<Transform, Matrix4x4>("worldToLocalMatrix", t => t.worldToLocalMatrix, null);
        Register<Transform, Matrix4x4>("localToWorldMatrix", t => t.localToWorldMatrix, null);
        Register<Transform, int>("childCount", t => t.childCount, null);

        Register<RectTransform, Vector2>("offsetMax", t => t.offsetMax, (t, v) => t.offsetMax = v);
        Register<RectTransform, Vector2>("offsetMin", t => t.offsetMin, (t, v) => t.offsetMin = v);
        Register<RectTransform, Vector2>("pivot", t => t.pivot, (t, v) => t.pivot = v);
        Register<RectTransform, Vector2>("sizeDelta", t => t.sizeDelta, (t, v) => t.sizeDelta = v);
        Register<RectTransform, Vector2>("anchoredPosition", t => t.anchoredPosition, (t, v) => t.anchoredPosition = v);
        Register<RectTransform, Vector2>("anchorMax", t => t.anchorMax, (t, v) => t.anchorMax = v);
        Register<RectTransform, Vector3>("anchoredPosition3D", t => t.anchoredPosition3D, (t, v) => t.anchoredPosition3D = v);
        Register<RectTransform, Vector2>("anchorMin", t => t.anchorMin, (t, v) => t.anchorMin = v);
        Register<RectTransform, Rect>("rect", t => t.rect, null);

        Register<GameObject, bool>("activeSelf", t => t.activeSelf, (t, v) => t.SetActive(v));
        Register<GameObject, int>("layer", t => t.layer, (t, v) => t.layer = v);
        Register<GameObject, string>("tag", t => t.tag, (t, v) => t.tag = v);

        Register<Behaviour, bool>("enabled", t => t.enabled, (t, v) => t.enabled = v);
        Register<Behaviour, bool>("isActiveAndEnabled", t => t.isActiveAndEnabled, null);

        Register<Component, string>("tag", t => t.tag, (t, v) => t.tag = v);

        Register<Canvas, float>("planeDistance", t => t.planeDistance, (t, v) => t.planeDistance = v);
        Register<Canvas, string>("sortingLayerName", t => t.sortingLayerName, (t, v) => t.sortingLayerName = v);
        Register<Canvas, int>("sortingLayerID", t => t.sortingLayerID, (t, v) => t.sortingLayerID = v);
        Register<Canvas, int>("renderOrder", t => t.renderOrder, null);

        Register<CanvasGroup, float>("alpha", t => t.alpha, (t, v) => t.alpha = v);
        Register<CanvasGroup, bool>("interactable", t => t.interactable, (t, v) => t.interactable = v);
        Register<CanvasGroup, bool>("blocksRaycasts", t => t.blocksRaycasts, (t, v) => t.blocksRaycasts = v);
        Register<CanvasGroup, bool>("ignoreParentGroups", t => t.ignoreParentGroups, (t, v) => t.ignoreParentGroups = v);

        Register<GraphicRaycaster, bool>("ignoreReversedGraphics", t => t.ignoreReversedGraphics, (t, v) => t.ignoreReversedGraphics = v);

        Register<Mask, bool>("showMaskGraphic", t => t.showMaskGraphic, (t, v) => t.showMaskGraphic = v);

        Register<Selectable, SpriteState>("spriteState", t => t.spriteState, (t, v) => t.spriteState = v);
        Register<Selectable, ColorBlock>("colors", t => t.colors, (t, v) => t.colors = v);
        Register<Selectable, bool>("interactable", t => t.interactable, (t, v) => t.interactable = v);

        Register<Button, Button.ButtonClickedEvent>("onClick", t => t.onClick, null);

        Register<InputField, InputField.OnChangeEvent>("onValueChanged", t => t.onValueChanged, null);
        Register<InputField, InputField.SubmitEvent>("onEndEdit", t => t.onEndEdit, null);
        Register<InputField, string>("text", t => t.text, (t, v) => t.text = v);

        Register<Scrollbar, Scrollbar.ScrollEvent>("onValueChanged", t => t.onValueChanged, null);
        Register<Scrollbar, float>("size", t => t.size, (t, v) => t.size = v);
        Register<Scrollbar, float>("value", t => t.value, (t, v) => t.value = v);

        Register<Slider, Slider.SliderEvent>("onValueChanged", t => t.onValueChanged, null);
        Register<Slider, float>("value", t => t.value, (t, v) => t.value = v);
        Register<Slider, float>("maxValue", t => t.maxValue, (t, v) => t.maxValue = v);
        Register<Slider, float>("minValue", t => t.minValue, (t, v) => t.minValue = v);

        Register<Dropdown, int>("value", t => t.value, (t, v) => t.value = v);
        Register<Dropdown, Dropdown.DropdownEvent>("onValueChanged", t => t.onValueChanged, null);

        Register<Text, string>("text", t => t.text, (t, v) => t.text = v);
        Register<Text, int>("fontSize", t => t.fontSize, (t, v) => t.fontSize = v);
    }
}

```

```
Register<Toggle, bool>("isOn", t => t.isOn, (t, v) => t.isOn = v);
Register<Toggle, Toggle.ToggleEvent>("onValueChanged", t => t.onValueChanged, (t, v) => t.onValueChanged = v);

Register<ToggleGroup, bool>("allowSwitchOff", t => t.allowSwitchOff, (t, v) => t.allowSwitchOff = v);
}

static void Register<T, TValue>(string name, Func<T, TValue> getter, Action<T, TValue> setter)
{
    var propertyInfo = typeof(T).GetProperty(name);
    if (propertyInfo is PropertyInfo)
    {
        ProxyFactory.Default.Register(new ProxyPropertyInfo<T, TValue>(name, getter, setter));
        return;
    }

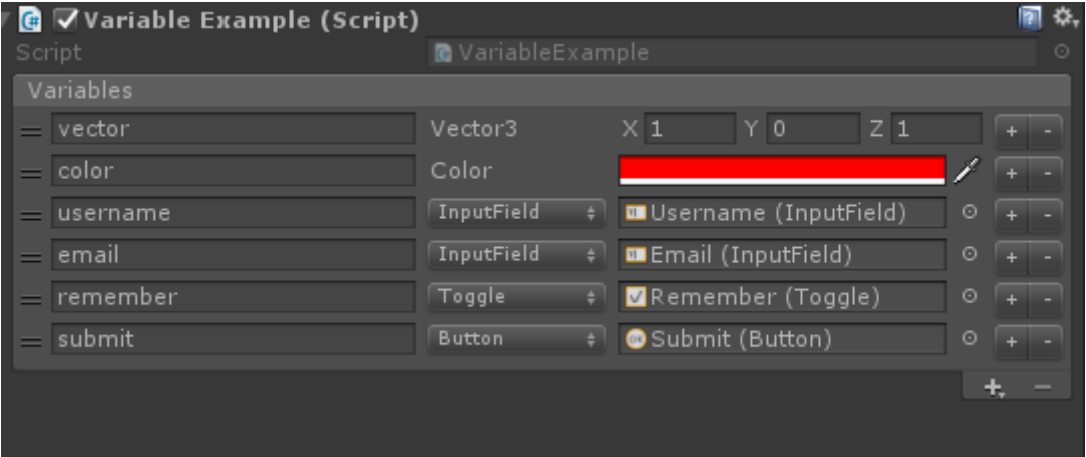
    var fieldInfo = typeof(T).GetField(name);
    if (fieldInfo is FieldInfo)
    {
        ProxyFactory.Default.Register(new ProxyFieldInfo<T, TValue>(name, getter, setter));
        return;
    }

    throw new Exception(string.Format("Not found the property or field named '{0}' in {1} type", name, typeof(T).Name));
}
}
```

UI framework

Variables

During the development of the UI, the view script often needs to access and control the UI controls on the UI interface. Generally speaking, we must either use Transform.Find to find it, or define a property in the View script, and control the control when editing the UI interface. Drag and drop onto this property. The first method is not efficient, and the second method adds and deletes the script properties again, which is not so flexible. Here, I provide the third method, VariableArray, which is a dynamic variable set that can be easily added and deleted, and can be used like a member property. And it not only supports all basic data types, but also supports Unity component types and value types.



```
//C#, Access to the variable
Color color = this.variables.Get<Color>("color");
InputField usernameInput = this.variables.Get<InputField>("username");
InputField emailInput = this.variables.Get<InputField>("email");

--Lua, You can access variables directly through self,
--just as you can access member properties in the current Lua table
printf("vector:%s",self.vector.ToString())
printf("color:%s",self.color.ToString())
printf("username:%s",self.username.text)
printf("email:%s",self.email.text)
```

UI view locator (UIViewLocator)

The UI view locator is a service for querying and loading UI views. It provides services for loading UI views synchronously and asynchronously. Depending on the project, you can customize its functionality. You can load views from Resources, you can also load views from an AssetBundle, or both.

```
//C#, Create a default view locator that supports loading views from Resources.  
//If you want to load views from AssetBundle, you need to implement it by yourself  
UIViewLocator locator = new DefaultUIViewLocator()  
  
//Loads a Loading window view with a UI pathname through the UI view locator  
var window = locator.LoadWindow<LoadingWindow>("UI/Loading");  
window.Show();
```

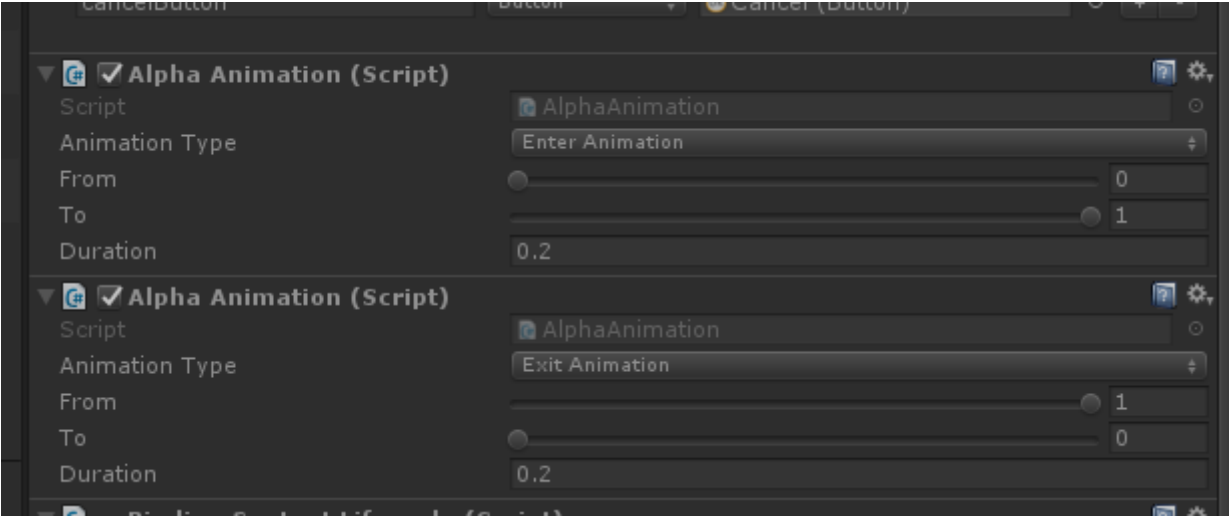
Animations

According to the process of opening, closing, gaining focus, and losing focus of a UI view, the view animation can be divided into admission animation, exit animation, activation animation, and passivation animation. Inherit UINavigation or IAnimation, use DoTween, iTween, etc., you can create your own UI animation.

UIView supports entry and exit animations in the framework. The animation can be played when a view is opened or a view is hidden. In addition to supporting entry and exit animations, Window also supports activation and passivation animations, and automatically controls playback. When a window gains focus, it plays an activation animation, and when it loses focus, it plays a passivation animation.

As shown below, in Examples, I created a fade-out animation, hung them on a Window view, and set it as an entrance animation and an exit animation, which gradually appear when the window opens, and slowly when the window closes. Slowly disappear.

Customize a C# fade-in animation




```

public class AlphaAnimation : UIAnimation
{
    [Range (0f, 1f)]
    public float from = 1f;
    [Range (0f, 1f)]
    public float to = 1f;

    public float duration = 2f;

    private IUIView view;

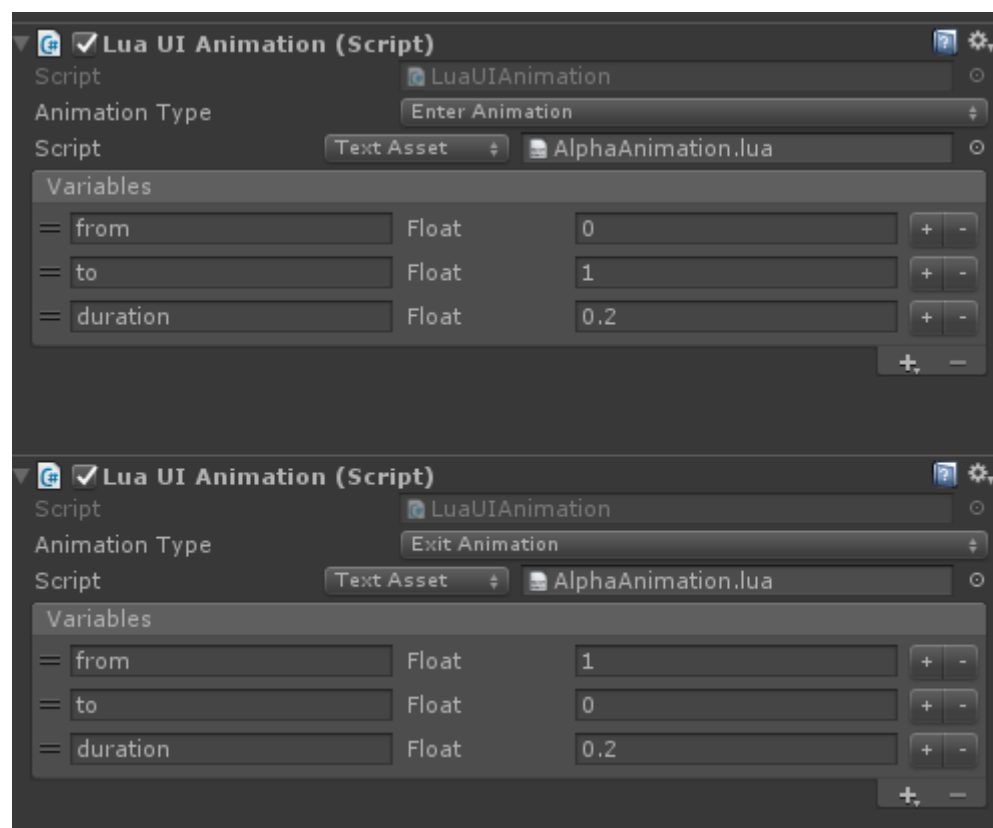
    void OnEnable ()
    {
        this.view = this.GetComponent<IUIView> ();
        switch (this.AnimationType) {
            case AnimationType.EnterAnimation:
                this.view.EnterAnimation = this;
                break;
            case AnimationType.ExitAnimation:
                this.view.ExitAnimation = this;
                break;
            case AnimationType.ActivationAnimation:
                if (this.view is IWindowView)
                    (this.view as IWindowView).ActivationAnimation = this;
                break;
            case AnimationType.PassivationAnimation:
                if (this.view is IWindowView)
                    (this.view as IWindowView).PassivationAnimation = this;
                break;
        }

        if (this.AnimationType == AnimationType.ActivationAnimation
            || this.AnimationType == AnimationType.EnterAnimation)
        {
            this.view.CanvasGroup.alpha = from;
        }
    }

    public override IAnimation Play ()
    {
        this.view.CanvasGroup.DOFade (this.to, this.duration)
            .OnStart (this.OnStart)
            .OnComplete (this.OnEnd)
            .Play ();
        return this;
    }
}

```

Using DoTween to customize a Lua animation



```

require("framework.System")

---
--module
--@module AlphaAnimation
local M=class("AlphaAnimation",target)

function M:play(view,startCallback,endCallback)
    view.CanvasGroup:DOFade(self.to, self.duration)
        :OnStart(function() startCallback() end)
        :OnComplete(function() endCallback() end)
        :Play()
end

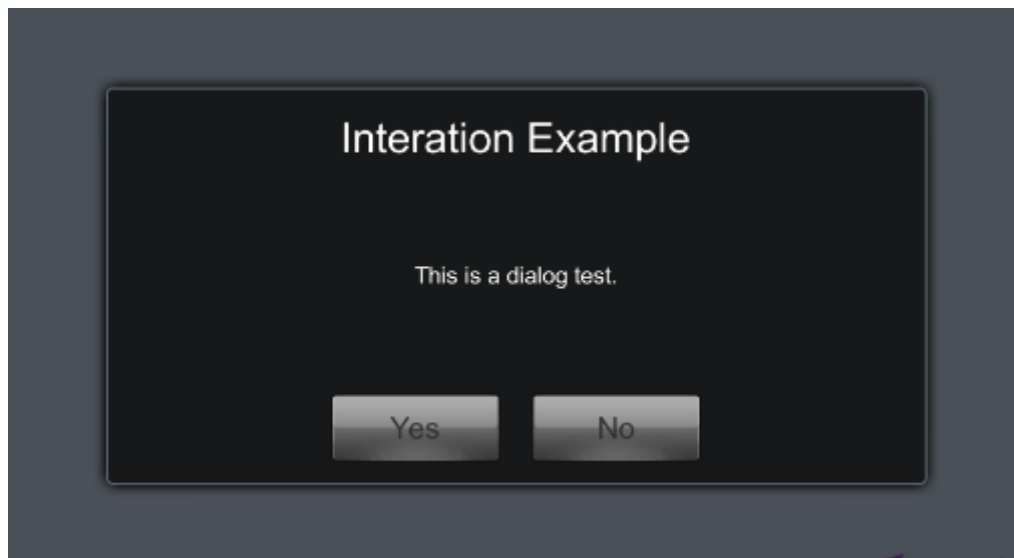
return M

```

UI controls

Although UGUI provides us with a rich UI control library, at some time, it still cannot meet our requirements. For example, we need a ListView with superior performance. At this time, we need to customize our own UI controls. In this framework, I provide some common UI controls, such as AlertDialog, Loading, Toast, etc. In the Examples / Resources / UI directory, you can find the default view interface. Refer to these interfaces to redefine the interface appearance and modify the static The ViewName property of the class can be used to reformulate the loading path of the view.

The following uses AlertDialog as an example to introduce their usage.



```

//The default directory path for a dialog view is UI/AlertDialog.
//You can modify the view path as follows
AlertDialog.ViewName = "Your view directory/AlertDialog";

//C#, Opens a dialog window
AlertDialog.ShowMessage("This is a dialog test.", "Interation Example", "Yes", null, "No", true,
result =>
{
    Debug.LogFormat("Result:{0}",result);
});

```

Views, windows, and window managers

- **IView/IUIView**

Views are popularly presented to the UI interface, images, animations, etc. that the user sees. In this framework, according to the characteristics of the game view layer, it is divided into two categories, scene views and UI views. The UI view corresponds to the IUIView interface, and the scene view corresponds to the IView interface.

- **IViewGroup/IUIViewGroup**

A view group is a collection of views. It can also be said as a view container. It consists of multiple views. You can add and delete subviews in a view group. At the same time, the view group itself is a view, and it can also be used as a subview of other view groups.

In UI development, we often find that a UI interface can be divided into many areas, such as the Top bar, the left bar, the right bar, the Bottom bar, the content area, etc., and some parts can be shared between multiple UI interfaces. of. Based on these characteristics, I can make different areas into different views. When the final interface is displayed, the view group is assembled into a complete view. This not only helps to improve the reuse of the code, but also greatly reduces the code. Coupling and complexity. **The important point is that we can use this design idea to design the novice guidance system for the game. Only when the interface needs to display guidance, the guidance interface is dynamically inserted into the current interface. The novice guidance logic is completely separated from the normal game logic to avoid a high degree of coupling between the guidance logic and the game logic.**

Similarly, in the game scene view, we can also split complex views into large and small view groups and subviews, and dynamically add and delete subviews during the game. For example, a game character is a subview in the scene. When the character enters the field of view, the view is added, and when it disappears from the field of view, the view is deleted.

Taking the King Glory daily activity interface as an example, it can be split into a top menu bar, a left menu bar, and a content area. The menu bar view can be reused. You only need to change the view of the content area each time.



• IWindow

Window is the root container of a UI interface view (IUIViewGroup, IUIView). It is also a controller. It is responsible for creating, destroying, displaying, and hiding window views. It is responsible for managing the life cycle of views and view models. Window interaction, etc.

```
//C#, creat a window
public class ExampleWindow : Window
{
    public Text progressBarText;
    public Slider progressBarSlider;
    public Text tipText;
    public Button button;

    protected override void OnCreate(Bundle bundle)
    {
        BindingSet<ExampleWindow, ExampleViewModel> bindingSet;
        bindingSet = this.CreateBindingSet(new ExampleViewModel());

        bindingSet.Bind(this.progressBarSlider).For("value", "onValueChanged").To("ProgressBar.Progress").TwoWay();
        bindingSet.Bind(this.progressBarSlider.gameObject).For(v => v.activeSelf)
            .To(vm => vm.ProgressBar.Enable).OneWay();
        bindingSet.Bind(this.progressBarText).For(v => v.text)
            .ToExpression(
                vm => string.Format("{0}%", Mathf.FloorToInt(vm.ProgressBar.Progress * 100f)))
            .OneWay();
        bindingSet.Bind(this.tipText).For(v => v.text).To(vm => vm.ProgressBar.Tip).OneWay();
        bindingSet.Bind(this.button).For(v => v.onClick).To(vm => vm.Click).OneWay();
        binding, bound to the onClick event and interactable property.
        bindingSet.Build();
    }

    protected override void OnDismiss()
    {
    }
}

--Lua, create a window
require("framework.System")

local ExampleViewModel = require("LuaUI.Startup.ExampleViewModel")

---
--module
--@module ExampleWindow
local M=class("ExampleWindow",target)

function M: onCreate(bundle)
    self.viewModel = ExampleViewModel()

    self:BindingContext().DataContext = self.viewModel

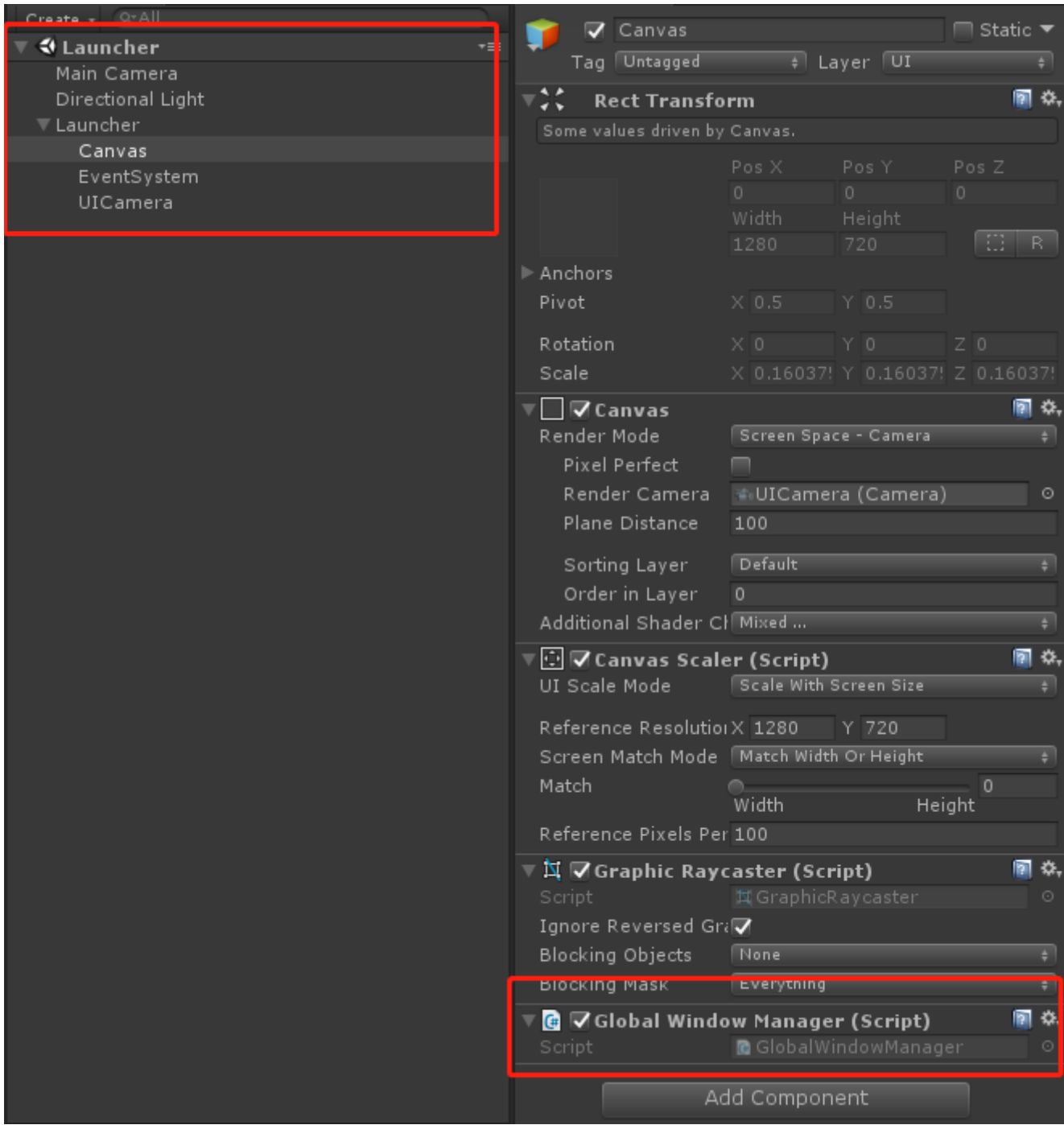
    local bindingSet = self:CreateBindingSet()

    bindingSet:Bind(self.progressBarSlider):For("value", "onValueChanged"):To("progressBar.progress"):TwoWay()
    bindingSet:Bind(self.progressBarSlider.gameObject):For("activeSelf"):To("progressBar.enable"):OneWay()
    bindingSet:Bind(self.progressBarText):For("text"):ToExpression(
        function(vm) return string.format("%0.2f%%",vm.progressBar.progress * 100) end,
        "progressBar.progress"):OneWay()
    bindingSet:Bind(self.tipText):For("text"):To("progressBar.tip"):OneWay()
    bindingSet:Bind(self.button):For("onClick"):To("command"):OneWay()
    bindingSet:Build()
end

return M
```

- **Window container and window manager**

The window manager is a container for managing windows. When the game starts, you need to create a global window manager, GlobalWindowManager, and hang it on the outer root Canvas (see the figure below). Create and edit other root Canvas under this root Canvas Window view.



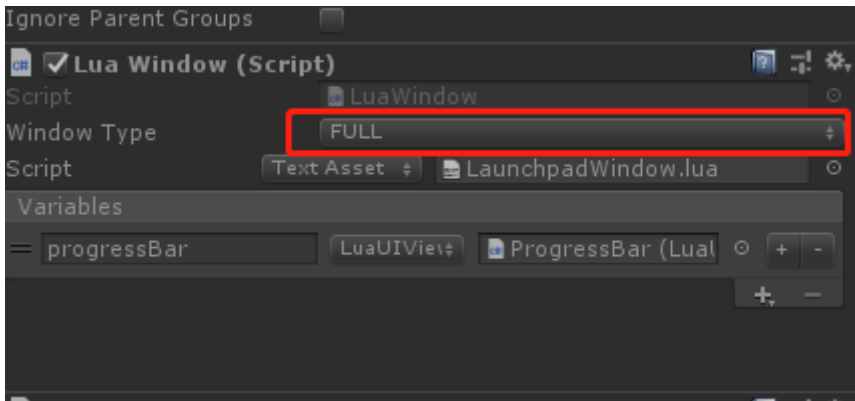
The window container is both a window manager and a window. In the window container, you can add and delete child windows, manage child windows, and also display and hide like a normal window. Take our MMO game. Generally, a main window container named "Main" and a "Battle" window container are created. All window views opened in the main interface will be placed in the Main container, but when you enter a certain When there are two battle copies, the Main container will be hidden and the "Battle" container will be displayed. All UI windows in the battle copy will be managed by the Battle container. When exiting the copy, you only need to close the Battle container and set the Main container to be visible. Restores the hierarchical relationship of windows in the Main container.

```
//C#, Create a MAIN container, which by default is created in the global window manager
WindowContainer winContainer = WindowContainer.Create("MAIN");
IUIViewLocator locator = context.GetService<IUIViewLocator>();

//Open a window in the MAIN container
StartupWindow window = locator.LoadWindow<StartupWindow>(winContainer, "UI/Startup/Startup");
ITransition transition = window.Show()
```

• Window type

The window types are divided into four types, full-screen windows (FULL), pop-up windows (POPUP), dialog windows (DIALOG), and progress bar windows (PROGRESS). Different window types will behave differently when the window is opened and blocked. Pop-up windows will automatically close when they are covered by other windows. The dialog window and progress window have the highest priority. It will be displayed at the top level and only one can be opened. When a dialog window or progress window is displayed, if other windows are opened, other windows will not be displayed. Or the progress window will be displayed when it is closed. If multiple dialog windows are opened at the same time, the dialog windows will be queued for processing, and the next one will be displayed only when the previous one is closed.



Interaction Request

InteractionRequest (InteractionRequest) is the most difficult to understand, the most complicated and the most winding place in the use of the MVVM framework, and it is not mentioned in many MVVM examples on the Internet. Why do we need interactive requests? What problem does the interactive request solve? The main purpose of introducing interactive requests is to decouple the view model (ViewModel) and the view (View). In the view model, we should not create, reference and directly control the view, because that is the work of the control layer, and it should not be the view model layer. Work, the view layer can depend on the view model layer, but the opposite is not allowed, remember . In the click event of a button, the creation or destruction of the view is often triggered. In MVVM, the button click event is usually bound to a command (ICommand) in the view model layer, that is, bound to the view model. In a member method, in addition to view-independent logic, this method also contains logic that controls the creation, opening, and destruction of views. As mentioned earlier, these logics will cause references and dependencies on the view layer. This is It is not allowed, so we have introduced the concept of InteractionRequest. Through the interaction request, the view control logic is sent back to the control layer for processing (in this framework, View and Window scripts, which are both the view layer and the Control layer, see MVVM architecture diagram in previous chapter).

Take a look at the following code example, using an interactive request to open a warning dialog window, and receive the result selected by the user when the dialog window is closed.

```

public class InteractionExampleViewModel : ViewModelBase
{
    private InteractionRequest<DialogNotification> alertDialogRequest;

    private SimpleCommand openAlertDialog;

    public InteractionExampleViewModel()
    {
        // Create an interaction request that sends an open dialog notification
        // to the control layer (InteractionExample)
        this.alertDialogRequest = new InteractionRequest<DialogNotification>(this);

        // Creates a command to press the response button event
        this.openAlertDialog = new SimpleCommand(Click);
    }

    public IInteractionRequest AlertDialogRequest { get { return this.alertDialogRequest; } }

    public ICommand OpenAlertDialog { get { return this.openAlertDialog; } }

    public void Click()
    {
        // Set command Enable to false, decouple by data binding,
        // and indirectly make view-layer buttons unclickable
        this.openAlertDialog.Enabled = false;

        // Create a dialog notification
        DialogNotification notification = new DialogNotification("Interaction Example",
            "This is a dialog test.", "Yes", "No", true);

        // Create a callback function
        // that will be called when the AlertDialog dialog is closed
        Action<DialogNotification> callback = n =>
        {
            // Set the Enable of the command to true,
            // and the button will be automatically restored by binding
            this.openAlertDialog.Enabled = true;

            if (n.DialogResult == AlertDialog.BUTTON_POSITIVE)
            {
                // The YES button of the dialog box is pressed
                Debug.LogFormat("Click: Yes");
            }
            else if (n.DialogResult == AlertDialog.BUTTON_NEGATIVE)
            {
                // The NO button of the dialog box is pressed
                Debug.LogFormat("Click: No");
            }
        };

        // The interactive request sends a notification
        // to the View layer onOpenAlert function
        this.alertDialogRequest.Raise(notification, callback);
    }
}

public class InteractionExample : WindowView
{
    public Button openAlert;
    protected override void Start()
    {
        InteractionExampleViewModel viewModel = new InteractionExampleViewModel();
        this.SetDataContext(viewModel);

        // create a bindingSet
        BindingSet<InteractionExample, InteractionExampleViewModel> bindingSet;
        bindingSet = this.CreateBindingSet<InteractionExample, InteractionExampleViewModel>();

        // Bind the onOpenAlert function of this view to the interactive
        // request AlertDialogRequest of the viewmodel.
        // When the interactive request is triggered,
        // the onOpenAlert function will be called automatically

        bindingSet.Bind().For(v => this.OnOpenAlert(null, null)).To(vm => vm.AlertDialogRequest);

        // Bind the onClick event of the button
        // to the OpenAlertDialog command in the viewmodel
        bindingSet.Bind(this.openAlert).For(v => v.OnClick).To(vm => vm.OpenAlertDialog);
    }
}

```

```

        bindingSet.Build();
    }

    // Functions that create and open dialogs, triggered by interactive requests
    private void OnOpenAlert(object sender, InteractionEventArgs args)
    {
        // Receive notification from AlertDialogRequest for viewmodel layer interaction

        // Get notification data
        DialogNotification notification = args.Context as DialogNotification;

        // Gets the callback function when the AlertDialog window is closed
        var callback = args.Callback;

        if (notification == null)
            return;

        // Create a dialog window
        AlertDialog.ShowMessage(notification.Message, notification.Title, notification.ConfirmButtonText,
            null,
            notification.CancelButtonText,
            notification.CanceledOnTouchOutside,
            (result) =>
            {
                // Assign the result of the dialog button event response to
                // the Notification and pass it to the viewmodel layer for use
                notification.DialogResult = result;

                // When the dialog window closes, the callback function
                // set in the interaction request is invoked to inform
                // the viewmodel layer to process the subsequent logic
                if (callback != null)
                    callback();
            });
    }
}

```

For more examples, see the [Interaction Tutorials.unity](#)

Interaction Action

InteractionAction is used in conjunction with InteractionRequest. An interaction request is initiated by an interaction request. The interaction task is used to complete the interaction task. It is an extension of the view method binding to the interaction request in the previous section. Generally speaking, using method binding to an interaction request Yes, but for some common functions, such as requesting to open or close a Loading window, you can use InteractionAction to facilitate code reuse. In different views, you only need to create a LoadingInteractionAction instance to complete the opening function of the Loading window. See the example below to enable Loading

```

// Create an interaction request in the ViewModel
this.loadingRequest = new InteractionRequest<VisibilityNotification>();

// Create a command to display the Loading window in the ViewModel,
// and open a Loading interface through the command call interaction request
this.ShowLoading = new SimpleCommand(() =>
{
    VisibilityNotification notification = new VisibilityNotification(true);
    this.loadingRequest.Raise(notification);
});

// Create an interactive request LoadingInterActionAction in the View
this.loadingInteractionAction = new LoadingInteractionAction();

// Bind InteractionAction to InteractionRequest
bindingSet.Bind().For(v => v.loadingInteractionAction).To(vm => vm.loadingRequest);

```

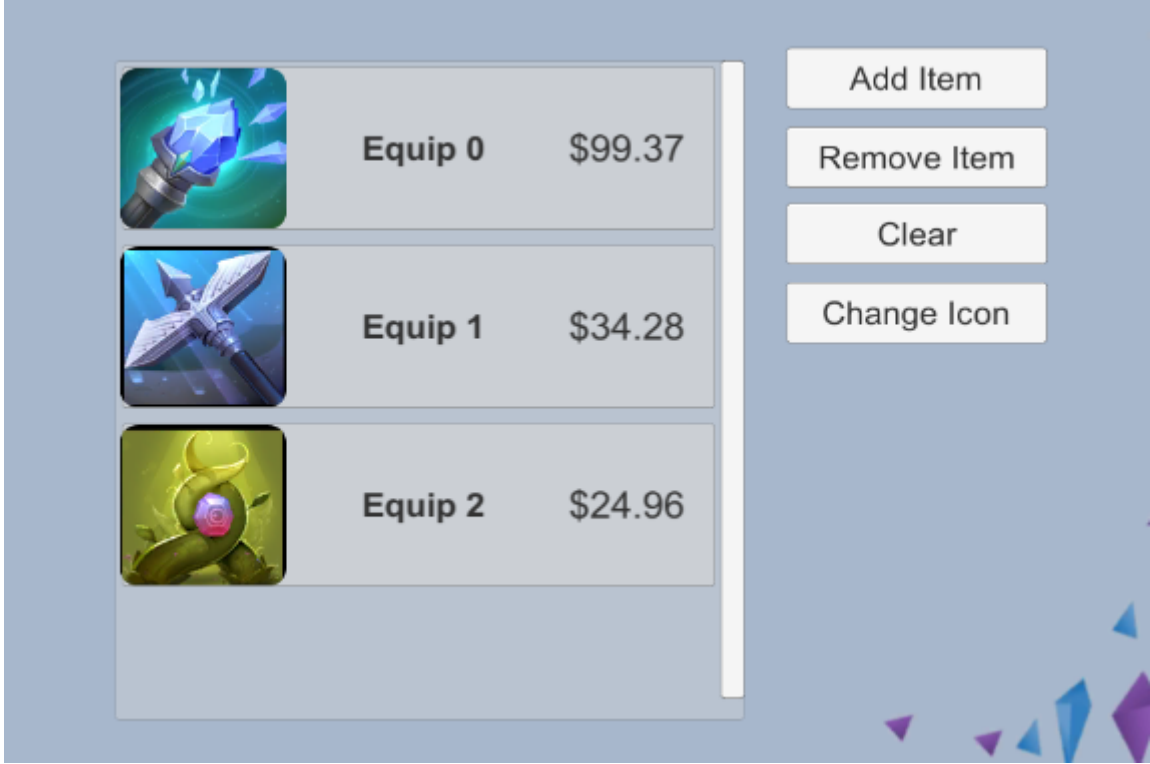
For more examples, see the [Interaction Tutorials.unity](#)

Collection and list view binding

In Unity3D game development, we often need to use UGUI's ScrollRect control. For example, we want to display a list of equipment, or all the items in a backpack. So can we use the data binding function to automatically update the content in the list, such as adding, deleting, and modifying data in a gear set, will the gear list view automatically update the interface content? The answer is yes. Use the ObservableList or ObservableDictionary collection to store equipment information. Through the data binding collection to a view script, you can automatically update

the content of the equipment list, but the view script here needs to be implemented by ourselves because each item List views are not standardized, and I cannot provide a universal script to provide bindings to collections.

In the following example, I created a ListView view script and used it to dynamically update the view of an equipment list.



First we create a ListView control, and use this control to listen for changes in the ObservableDictionary of the equipment collection. When the content in the collection changes, the UGUI view is automatically updated, and equipment is added to and removed from the equipment list.

```

public class ListView : UIView
{
    public class ItemClickedEvent : UnityEvent<int>
    {
        public ItemClickedEvent()
        {
        }
    }

    private ObservableList<ListItemViewModel> items;

    public Transform content;

    public GameObject itemTemplate;

    public ItemClickedEvent OnSelectChanged = new ItemClickedEvent();

    // Equipment collection, assigned via data binding
    public ObservableList<ListItemViewModel> Items
    {
        get { return this.items; }
        set
        {
            if (this.items == value)
                return;

            if (this.items != null)
                this.items.CollectionChanged -= OnCollectionChanged;

            this.items = value;

            this.OnItemsChanged();

            if (this.items != null)
                this.items.CollectionChanged += OnCollectionChanged;
        }
    }

    /// <summary>
    /// Monitor the change of equipment set
    /// and update the equipment list interface automatically
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="eventArgs"></param>
    protected void OnCollectionChanged(object sender, NotifyCollectionChangedEventArgs eventArgs)
    {
        switch (eventArgs.Action)
        {
            case NotifyCollectionChangedAction.Add:
                this.AddItem(eventArgs.NewStartingIndex, eventArgs.NewItems[0]);
                break;
            case NotifyCollectionChangedAction.Remove:
                this.RemoveItem(eventArgs.OldStartingIndex, eventArgs.OldItems[0]);
                break;
            case NotifyCollectionChangedAction.Replace:
                this.ReplaceItem(eventArgs.OldStartingIndex, eventArgs.OldItems[0], eventArgs.NewItems[0]);
                break;
            case NotifyCollectionChangedAction.Reset:
                this.ResetItem();
                break;
            case NotifyCollectionChangedAction.Move:
                this.MoveItem(eventArgs.OldStartingIndex, eventArgs.NewStartingIndex, eventArgs.NewItems[0]);
                break;
        }
    }

    protected virtual void OnItemsChanged()
    {
        for (int i = 0; i < this.items.Count; i++)
        {
            this.AddItem(i, items[i]);
        }
    }

    protected virtual void OnSelectChange(GameObject itemViewGo)
    {
        if (this.OnSelectChanged == null || itemViewGo == null)
            return;
    }
}

```

```

        for (int i = 0; i < this.content.childCount; i++)
        {
            var child = this.content.GetChild(i);
            if (itemViewGo.transform == child)
            {
                this.OnSelectChanged.Invoke(i);
                break;
            }
        }
    }

protected virtual void AddItem(int index, object item)
{
    var itemViewGo = Instantiate(this.itemTemplate);
    itemViewGo.transform.SetParent(this.content, false);
    itemViewGo.transform.SetSiblingIndex(index);

    Button button = itemViewGo.GetComponent<Button>();
    button.onClick.AddListener(() => OnSelectChange(itemViewGo));
    itemViewGo.SetActive(true);

    UIView itemView = itemViewGo.GetComponent<UIView>();
    itemView.SetDataContext(item);
}

protected virtual void RemoveItem(int index, object item)
{
    Transform transform = this.content.GetChild(index);
    UIView itemView = transform.GetComponent<UIView>();
    if (itemView.GetDataContext() == item)
    {
        itemView.gameObject.SetActive(false);
        Destroy(itemView.gameObject);
    }
}

protected virtual void ReplaceItem(int index, object oldItem, object item)
{
    Transform transform = this.content.GetChild(index);
    UIView itemView = transform.GetComponent<UIView>();
    if (itemView.GetDataContext() == oldItem)
    {
        itemView.SetDataContext(item);
    }
}

protected virtual void MoveItem(int oldIndex, int index, object item)
{
    Transform transform = this.content.GetChild(oldIndex);
    UIView itemView = transform.GetComponent<UIView>();
    itemView.transform.SetSiblingIndex(index);
}

protected virtual void ResetItem()
{
    for (int i = this.content.childCount - 1; i >= 0; i--)
    {
        Transform transform = this.content.GetChild(i);
        Destroy(transform.gameObject);
    }
}
}

```

Then create an Item view `ListItemView` of the equipment list, which is responsible for binding the UGUI control on the Item view to the equipment's view model, and automatically update the content of the Item view when the equipment's view model changes.

```

public class ListViewItem : UIView
{
    public Text title;
    public Text price;
    public Image image;
    public GameObject border;

    protected override void Start()
    {
        // Bind the view element on the Item
        BindingSet<ListViewItem, ListViewItemModel> bindingSet = this.CreateBindingSet<ListViewItem, ListViewItemModel>();
        bindingSet.Bind(this.title).For(v => v.text).To(vm => vm.Title).OneWay();
        bindingSet.Bind(this.image).For(v => v.sprite).To(vm => vm.Icon).WithConversion("spriteConverter").OneWay();
        bindingSet.Bind(this.price).For(v => v.text).ToExpression(vm => string.Format("${0:0.00}", vm.Price)).OneWay();
        bindingSet.Bind(this.border).For(v => v.activeSelf).To(vm => vm.IsSelected).OneWay();
        bindingSet.Build();
    }
}

```

Finally, the view model code for the ListView control and ListViewItem is as follows.

```

public class ListViewViewModel : ViewModelBase
{
    private readonly ObservableList<ListItemViewModel> items = new ObservableList<ListItemViewModel>();

    public ObservableList<ListItemViewModel> Items
    {
        get { return this.items; }
    }

    public ListItemViewModel SelectedItem
    {
        get
        {
            foreach (var item in items)
            {
                if (item.IsSelected)
                    return item;
            }
            return null;
        }
    }

    public void AddItem()
    {
        int i = this.items.Count;
        int iconIndex = Random.Range(1, 30);
        this.items.Add(new ListItemViewModel() {
            Title = "Equip " + i,
            Icon = string.Format("EquipImages_{0}", iconIndex),
            Price = Random.Range(10f, 100f)
        });
    }

    public void RemoveItem()
    {
        if (this.items.Count <= 0)
            return;

        int index = Random.Range(0, this.items.Count - 1);
        this.items.RemoveAt(index);
    }

    public void ClearItem()
    {
        if (this.items.Count <= 0)
            return;

        this.items.Clear();
    }

    public void ChangeItemIcon()
    {
        if (this.items.Count <= 0)
            return;

        foreach (var item in this.items)
        {
            int iconIndex = Random.Range(1, 30);
            item.Icon = string.Format("EquipImages_{0}", iconIndex);
        }
    }

    public void Select(int index)
    {
        if (index <= -1 || index > this.items.Count - 1)
            return;

        for (int i = 0; i < this.items.Count; i++)
        {
            if (i == index)
            {
                items[i].IsSelected = !items[i].IsSelected;
                if (items[i].IsSelected)
                    Debug.LogFormat("Select, Current Index:{0}", index);
                else
                    Debug.LogFormat("Cancel");
            }
            else

```

```

        {
            items[i].IsSelected = false;
        }
    }
}

```

```

public class ListViewItemViewModel : ViewModelBase
{
    private string title;
    private string icon;
    private float price;
    private bool selected;

    public string Title
    {
        get { return this.title; }
        set { this.Set<string>(ref title, value, "Title"); }
    }
    public string Icon
    {
        get { return this.icon; }
        set { this.Set<string>(ref icon, value, "Icon"); }
    }

    public float Price
    {
        get { return this.price; }
        set { this.Set<float>(ref price, value, "Price"); }
    }

    public bool IsSelected
    {
        get { return this.selected; }
        set { this.Set<bool>(ref selected, value, "IsSelected"); }
    }
}

```

```

public class ListViewDatabindingExample : MonoBehaviour
{
    private int itemCount;
    private ListViewViewModel viewModel;

    public Button addButton;

    public Button removeButton;

    public Button clearButton;

    public Button changeIconButton;

    public ListView listView;

    void Awake()
    {
        ApplicationContext context = Context.GetApplicationContext();
        BindingServiceBundle bindingService = new BindingServiceBundle(context.GetContainer());
        bindingService.Start();

        Dictionary<string, Sprite> sprites = new Dictionary<string, Sprite>();
        foreach (var sprite in Resources.LoadAll<Sprite>("EquipTextures"))
        {
            if (sprite != null)
                sprites.Add(sprite.name, sprite);
        }
        IConverterRegistry converterRegistry = context.GetContainer().Resolve<IConverterRegistry>();
        converterRegistry.Register("spriteConverter", new SpriteConverter(sprites));
    }

    void Start()
    {
        viewModel = new ListViewViewModel();
        for (int i = 0; i < 3; i++)
        {
            viewModel.AddItem();
        }

        IBindingContext bindingContext = this.BindingContext();
        bindingContext.DataContext = viewModel;
    }
}

```

```

BindingSet<ListViewDataBindingExample, ListViewViewModel> bindingSet;
bindingSet = this.CreateBindingSet<ListViewDataBindingExample, ListViewViewModel>();
bindingSet.Bind(this.listView).For(v => v.Items).To(vm => vm.Items).OneWay();
bindingSet.Bind(this.listView).For(v => v.OnSelectChanged).To(vm => vm.Select(0)).OneWay();

bindingSet.Bind(this.addButton).For(v => v.onClick).To(vm => vm.AddItem());
bindingSet.Bind(this.removeButton).For(v => v.onClick).To(vm => vm.RemoveItem());
bindingSet.Bind(this.clearButton).For(v => v.onClick).To(vm => vm.ClearItem());
bindingSet.Bind(this.changeIconButton).For(v => v.onClick).To(vm => vm.ChangeItemIcon());

bindingSet.Build();
}
}

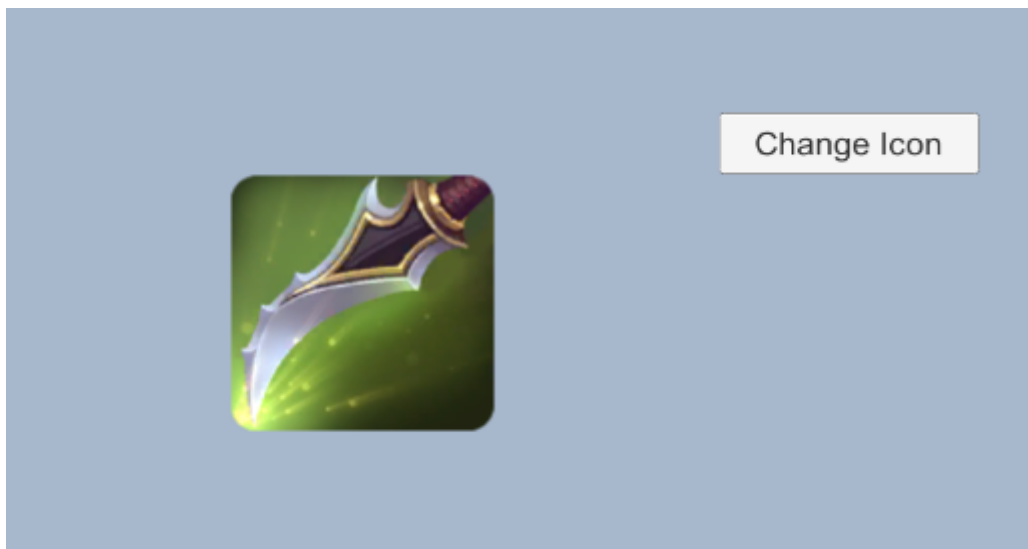
```

For more examples, see the [ListView And Sprite Databinding Tutorials.unity](#)

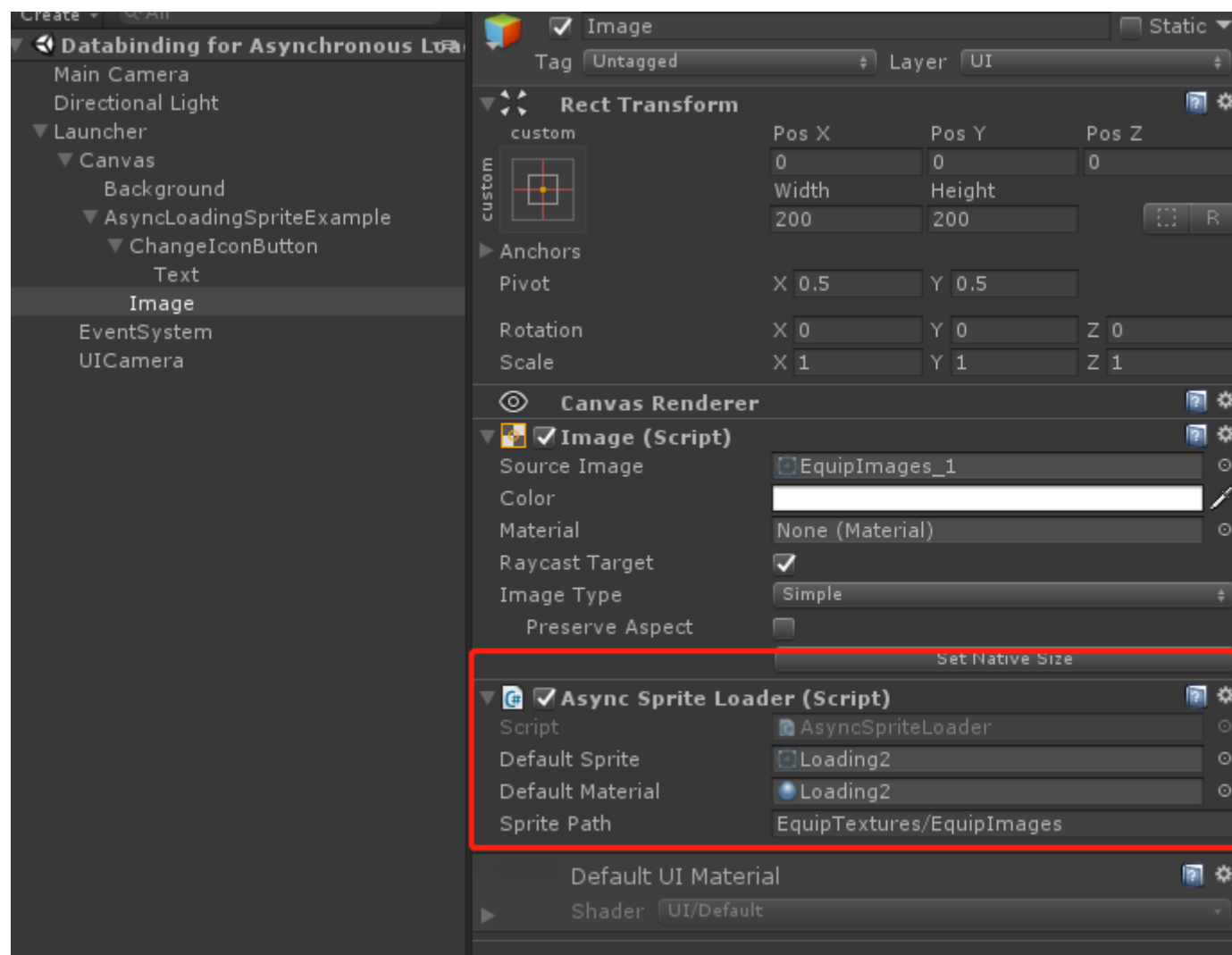
Data binding and asynchronous loading sprites

In the previous example, I had a binding to a sprite, but it was loaded into memory ahead of time. Here I will talk about how to load a sprite asynchronously through data binding. Similar to collection binding in the previous section, asynchronous loading of sprites can be easily achieved with a view script. Let's look at an example.

Click the "Change Icon" button in the figure to change the icon. The loading of the icon is asynchronous, and there is a loading animation.



First, we implement a sprite asynchronous loader and hang it on the Image control that needs to load sprite images asynchronously.



```

[RequireComponent(typeof(Image))]
public class AsyncSpriteLoader : MonoBehaviour
{
    private Image target;
    private string spriteName;
    public Sprite defaultSprite;
    public Material defaultMaterial;
    public string spritePath;

    public string SpriteName
    {
        get { return this.spriteName; }
        set
        {
            if (this.spriteName == value)
                return;

            this.spriteName = value;
            if (this.target != null)
                this.OnSpriteChanged();
        }
    }

    protected virtual void OnEnable()
    {
        this.target = this.GetComponent<Image>();
    }

    protected virtual void OnSpriteChanged()
    {
        if (string.IsNullOrEmpty(this.spriteName))
        {
            this.target.sprite = null;
            this.target.material = null;
            return;
        }

        this.target.sprite = defaultSprite;
        this.target.material = defaultMaterial;

        StartCoroutine(LoadSprite());
    }

    /// <summary>
    /// Load the Sprite asynchronously,
    /// wait a second in the loader for the effect to be obvious
    /// </summary>
    /// <returns></returns>
    IEnumerator LoadSprite()
    {
        yield return new WaitForSeconds(1f);

        Sprite[] sprites = Resources.LoadAll<Sprite>(this.spritePath);
        foreach(var sprite in sprites)
        {
            if(sprite.name.Equals(this.spriteName))
            {
                this.target.sprite = sprite;
                this.target.material = null;
            }
        }
    }
}

```

Then create the view and view model code of the sample interface as follows.


```

public class SpriteViewModel : ViewModelBase
{
    private string spriteName = "EquipImages_1";

    public string SpriteName
    {
        get { return this.spriteName; }
        set { this.Set<string>(ref spriteName, value, "SpriteName"); }
    }

    public void ChangeSpriteName()
    {
        this.SpriteName = string.Format("EquipImages_{0}", Random.Range(1, 30));
    }
}

public class DatabindingForAsyncLoadingSpriteExample : MonoBehaviour
{
    public Button changeSpriteButton;

    public AsyncSpriteLoader spriteLoader;

    void Awake()
    {
        ApplicationContext context = Context.GetApplicationContext();
        BindingServiceBundle bindingService = new BindingServiceBundle(context.GetContainer());
        bindingService.Start();
    }

    void Start()
    {
        var viewModel = new SpriteViewModel();

        IBindingContext bindingContext = this.BindingContext();
        bindingContext.DataContext = viewModel;

        BindingSet<DatabindingForAsyncLoadingSpriteExample, SpriteViewModel> bindingSet;
        bindingSet = this.CreateBindingSet<DatabindingForAsyncLoadingSpriteExample, SpriteViewModel>();
        bindingSet.Bind(this.spriteLoader).For(v => v.SpriteName).To(vm => vm.SpriteName).OneWay();

        bindingSet.Bind(this.changeSpriteButton).For(v => v.onClick).To(vm => vm.ChangeSpriteName());

        bindingSet.Build();
    }
}

```

For more examples, see the [Databinding for Asynchronous Loading Sprites Tutorials.unity](#)

Lua

Modules and inheritance

Using Lua's original table inheritance, the concepts of classes (modules) and inheritance are simulated in Lua development. Through the class function of the System module, you can define modules, inherit modules, inherit C # classes, extend C # instances, and write with object-oriented thinking. lua code.

With the following code example, let's see how to define modules and inherit modules.

```
-- Define a base class called Animal
local Animal = class("Animal")

-- Creates an instance of Animal from Animal(), and calls this constructor
function Animal:ctor(...)
end

-- Define a walk() method for Animal
function Animal:walk()
    print("animal walk")
end

-- Define a run() method for Animal
function Animal:run()
    print("animal run")
end

-- Define a class named Cat that inherits from the Animal class
local Cat = class("Cat",Animal)

-- The constructor of the Cat class
function Cat:ctor()
    -- When the constructor is overloaded,
    -- the superclass constructor is overridden and called as shown below
    Cat.super.ctor(self)
    self.age = 5
end
```

In addition to Lua, modules can also inherit C # classes, and of course static classes. To inherit a non-static C # class in Lua, this class must be able to be instantiated by the new keyword or provide other instantiation functions. For example, the MonoBehaviour script class cannot be instantiated by the new keyword and cannot be inherited in Lua. In the class function, the first parameter is the class name, and the second parameter must be a C # class or an instantiated function of the C # class. We can rewrite the function of the parent class in Lua script, or call the function of the parent class in Lua, see the following code.

Note: The function of the parent class must be called using the module name, not self

```
M.super.Get(self,name,cascade) --correct

self.super.Get(self,name,cascade) --wrong

M.super:Get(name,cascade) --wrong
```

Lua inherits C # class Loxodon.Framework.Contexts.Context, adds GetName () function, and overrides Context.Get (string name, bool cascade) function.

```

-- Define a module that extends from C# class Context.
-- It is recommended that the variable names of modules be M by default
local M = class("LuaContext",CS.Loxodon.Framework.Contexts.Context)

-- Add a new function
function M:GetName()

    -- The code is omitted

end

-- Overrides the functions of the parent class and calls the functions of the parent class

function M:Get(name,cascade)
    -- Call the function of the parent class
    local ret = M.super.Get(self,name,cascade)
    if ret then return ret end

    -- The code is omitted

end

return M

```

A MonoBehaviour script cannot be inherited, but its instance can be extended by Lua. Using the class function, we can add new properties and methods to it. Unlike C # class inheritance, the second parameter of class is an instance of a C # class. Look at the Lua example, the C # script LuaLauncher extension code.

The "target" object is in the C # script LuaLauncher. When initializing the lua script environment, it injects its own instance into the lua environment. In all extension scripts of this framework, the "target" variable name is used uniformly. Follow this rule.

Attention

After Lua inherit a C# class or Lua expands the MonoBehaviour instance, C# examples and Lua module corresponding examples are the same object in Lua runtime. You got the MonoBehaviour instance objects, not only have access to all functions and attributes of MonoBehaviour, at the same time can also visit the Lua script extension functions and attributes

C# code, part of LuaLauncher script to initialize lua execution environment.

```

var luaEnv = LuaEnvironment.LuaEnv;
scriptEnv = luaEnv.NewTable();

LuaTable meta = luaEnv.NewTable();
meta.Set("__index", luaEnv.Global);
scriptEnv.SetMetaTable(meta);
meta.Dispose();

// Inject this into the Lua environment table, using the unify target variable name
scriptEnv.Set("target", this);

string scriptText = "";
if(script.Type == ScriptReferenceType.TextAsset)
    scriptText = script.Text.text;
else
    string.Format("require(\"framework.System\");local cls = require(\"{0}\");return extends(target,cls);",
        script.Filename)

object[] result = luaEnv.DoString(scriptText, string.Format("{0}({1})", "Launcher", this.name), scriptEnv);

if (result.Length != 1 || !(result[0] is LuaTable))
    throw new Exception();

metatable = (LuaTable)result[0];

onAwake = metatable.Get<Action<MonoBehaviour>>("awake");
onEnable = metatable.Get<Action<MonoBehaviour>>("enable");
onDisable = metatable.Get<Action<MonoBehaviour>>("disable");
onStart = metatable.Get<Action<MonoBehaviour>>("start");
onDestroy = metatable.Get<Action<MonoBehaviour>>("destroy");

```

Extend LuaLauncher script function through lua, awake, enable, disable, start, destroy functions can all be implemented in lua and called in C #.

```

require("framework.System")

local WindowContainer = CS.Loxodon.Framework.Views.WindowContainer
local Context = CS.Loxodon.Framework.Contexts.Context
---
--Launcher module, the target parameter is agreed, please do not change.
--@module Launcher
local M=class("Launcher",target)

function M:start()
    -- To get an application context,
    -- one game suggests creating an application context and a player context.
    -- Global services are placed in the application context,
    -- such as account services, network components, configuration services,
    -- and other basic components and services
    -- Items that are only relevant to a player, such as knapsack services,
    -- equipment services, and character services, are placed in the player's context
    -- and can be released when logged out of the game
    local context = Context.GetApplicationContext()

    -- Get a view locator from the application context
    local locator = context:GetService("IUIViewLocator")

    -- Create a window container named MAIN
    local winContainer = WindowContainer.Create("MAIN")

    -- Load a startup window view through the view locator
    local window = locator:LoadWindow(winContainer, "LuaUI/Startup/Startup")

    -- Create a window
    window:Create()

    -- display window returns a Transition object.
    -- Window displays are usually animated Windows, so they are an ongoing process
    local transition = window:Show()

    -- Listens the window state of the display window procedure
    transition:OnStateChanged(function(w,state) print("Window:"..w.Name.." State:"..state.ToString()) end)

    -- The listening window displays completion events
    transition:OnFinish(function() print("OnFinished") end)
end

return M

```

Lua's ObserableObject

To meet the requirements of MVVM data binding, Lua's Table can trigger the notification of property modification when the property changes, then it must inherit ObserableObject. It is similar to the ObserableObject function of C #, but it is a version reimplemented in Lua language to adapt to Lua development. The view model and subview model defined in Lua must inherit this class. See example below.

```

require("framework.System")

local ObservableObject = require("framework.ObservableObject")

---
--Create an Account view model
--@module AccountViewModel
local M = class("AccountViewModel",ObservableObject)

function M:ctor(t)
    --Execute the constructor of the parent observableObject.
    --This is important, otherwise you cannot listen for data changes
    M.base(self).ctor(self)

    self.id = 0
    self.username = ""
    self.Password = ""
    self.email = ""
    self.birthday = os.time({year =1970, month = 00, day =00, hour =00, min =00, sec = 00})
    self.address = ""

    if t and type(t)=="table" then
        for k,v in pairs(t) do self[k] = v end
    end
end

return M

```

Coroutines in Lua using Unity

XLua provides us with a function `util.cs_generator ()` that creates an iterator (IEnumerator) in lua. Through this function, a lua method can be wrapped into a C # IEnumerator, and then a coroutine is executed in C#.

The following `doLoad` function simulates a loading task, executes a loop from 1 to 50, uses the lua coroutine yield method, and sleeps for 0.1 seconds each time.

```

---
--Simulate a load task
function M:doLoad(promise)
    print("task start")

    for i = 1, 50 do
        --If there is a Cancel request, that is, the function Cancel() of progressResult is called, then the task is terminated
        if promise.IsCancellationRequested then
            break
        end

        promise.UpdateProgress(i/50) --Update task progress

        --You can input no parameter in the coroutine.yield,
        --which means that it is executed once per frame.
        --You can also input all inherited YieldInstruction parameters,
        --such as: UnityEngine. WaitForSeconds (0.1);
        --You can also input an IEnumerator object, such as AsyncResult.WaitForDone()
        coroutine.yield(CS.UnityEngine.WaitForSeconds(0.1))--wait 0.1 second
    end
    promise.UpdateProgress(1)
    promise.SetResult() --Set the task to be completed
    print("task end")
end

```

Use XLua's function `util.cs_generator` to wrap `doLoad` into an IEnumerator and execute it in `Executors.RunOnCoroutineNoReturn`.

```

local Executors = require("framework.Executors")

local result = ProgressResult(true)
Executors.RunOnCoroutineNoReturn(util.cs_generator(function() self:doLoad(result) end))

```

In Lua, the C # Executors class is extended, and two functions `RunLuaOnCoroutine` and `RunLuaOnCoroutineNoReturn` are extended. Through them, Lua functions can be automatically wrapped into an IEnumerator and executed in the coroutine of Unity3D.

```

local Executors = require("framework.Executors")

--In the previous example, we could also execute the DOLOAD function as follows
local result = ProgressResult(true)
Executors.RunLuaOnCoroutineNoReturn(function(r) self:doLoad(r) end,result)

--or use the below method, it's equal to the above method.
--Self.doLoad is the function that needs to be executed.
--Self and result are the arguments to the doLoad function
Executors.RunLuaOnCoroutineNoReturn(self.doLoad,self,result)
return result

```

Define and execute a coroutine function as a closure.

```

--Executes a coroutine and returns an IAsyncResult.
--An expiration time, duration (in seconds) is passed, and the coroutine exits after duration is executed

return Executors.RunLuaOnCoroutine(function(duration)
    local wait = CS.UnityEngine.WaitForSeconds(0.05)
    local startTime = Time.realtimeSinceStartup
    while Time.realtimeSinceStartup - startTime < duration do
        coroutine.yield(wait)
    end
end,duration)

```

For more information about Lua coroutines, see `framework.Executors` and examples `LoxodonFramework/Lua/Examples/Coroutine Tutorials`.

Using the logging system in Lua

The framework provides a Lua version of the logging system. The underlying layer still uses `Loxodon.Log.ILog` to provide services, but functions are repackaged in Lua. It supports multiple levels of DEBUG, INFO, WARN, ERROR, and FATAL. The level of log printing can be set in the code or configuration file (if log4net is used). At the same time, it also supports displaying the file path and line number of the log, which is convenient for code debugging.

```

--If you use the default log factory, you can set the printing log level as follows
--If you use Log4Net, you can set the printing log level in the Log4Net configuration file
CS.Loxodon.Log.LogManager.Default.Level = CS.Loxodon.Log.Level.INFO

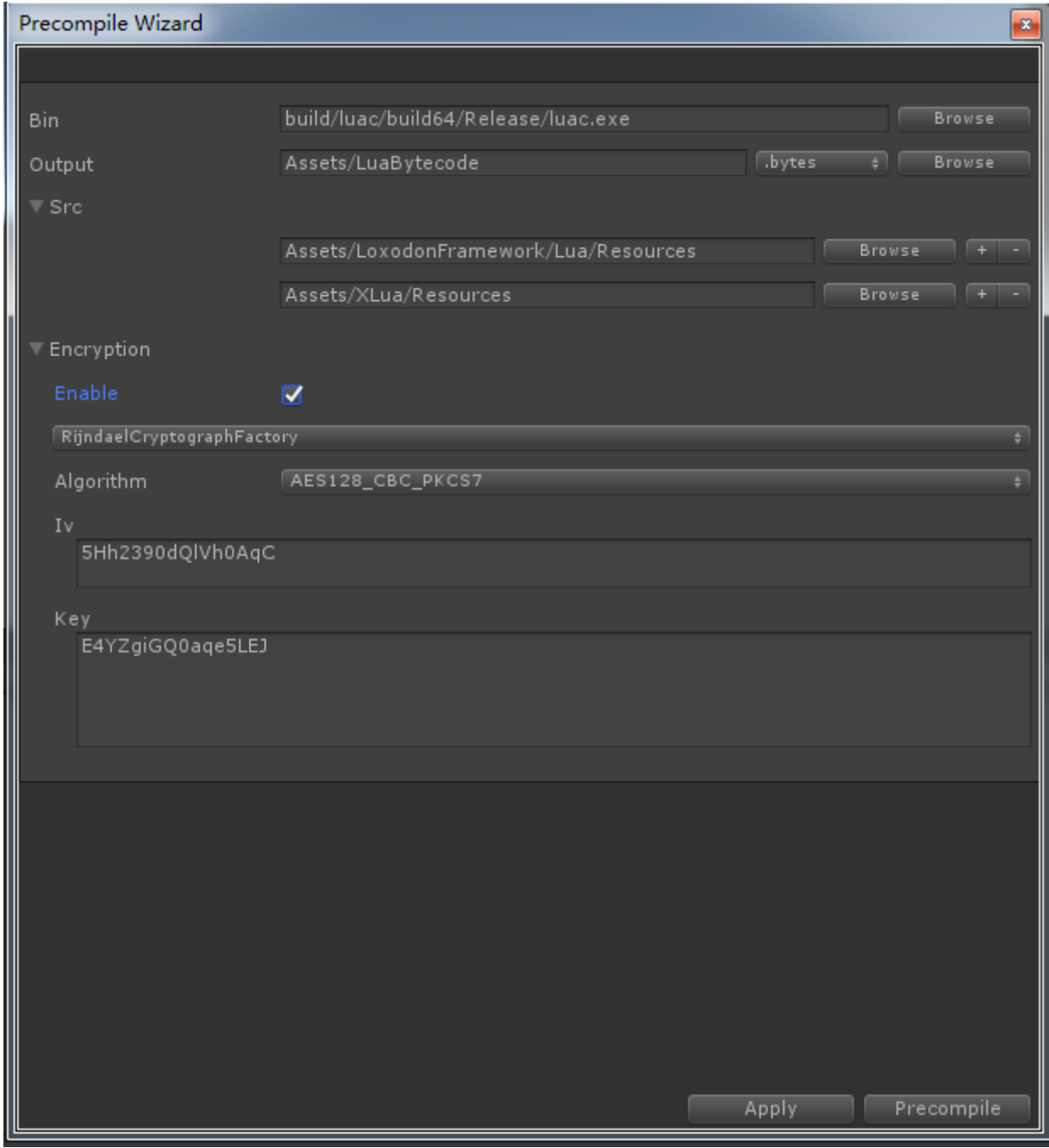
--Initialize the logging system
local logger = require("framework.Logger").GetLogger()

--printing log
logger:debug("This is a test.")
logger:info("This is a test.")

```

Lua precompiled tools

Lua precompilation tools can be used to precompile Lua scripts into bytecode files, and you can choose whether to encrypt the file. Lua official `luac` command compiled byte code points 64 and 32, if you want to compile 64-bit and 32-bit compatible bytecode, please refer XLua official documents, relating to common bytecode compiler part ["universal bytes Code"](#).



- Bin:luac command path. If you want to compile 64-bit, 32-bit, or compatible bytecode, select the corresponding luac command.
- Output: The directory where the compiled bytecode files are stored. You can select one of the directories under Assets and package them into an AssetBundle, or you can select a directory under StreamingAssets and load it directly from the StreamingAssets folder. The extension of pre-compiled bytecode files can be ".luac" or ".bytes", or other extensions can be customized.
- Src: the root directory of lua source code, supports multiple source code directories. The source code file extension must be ".lua" or ".lua.txt".
- Encryption: Encryption function, which supports AES encryption by default, or you can extend the new encryption method yourself. The panel of the encryption part is dynamic, and the new encryption method is extended, and it will be automatically displayed in the interface of the tool.
- Apply: ☐Save settings
- Precompile: ☐Precompile Lua scripts

Lua loader

- FileLoader
File loader, supports loading Lua scripts or Lua bytecode files from local folders, and also supports loading files from Android apk or obb, so if your lua source code or bytecode files are stored in the StreamingAssets folder , Can also load correctly on the Android platform.
- AssetBundleLoader
Support loading lua scripts or bytecodes from AssetBundle. Generally speaking, it is recommended to put all lua bytecodes in the same AssetBundle, load them into memory when the game starts, configure AssetBundleLoader loader, and load from this AssetBundle first lua code.
- DecodableLoader
Decodeable loader, which works with file loader or AssetBundle loader to decrypt binary data.

Example

In the following example, in Editor mode, a Lua file with the extension ".lua.txt" or ".lua" is loaded from the "Assets/LuaScripts/" directory via FileLoader. In the real machine mode, load the lua bytecode file from the Application.persistentDataPath + "/LuaScripts/" directory through FileLoader. If it is not found, search the Application.streamingAssetsPath + "/LuaScripts/" directory and use the DecodableLoader loader to decrypt it.

Multiple loaders can be added in LuaEnv. Loaders added later take precedence over those added first.

```

var luaEnv = LuaEnvironment.LuaEnv;

#if UNITY_EDITOR
    //Development mode to load Lua source code from a local directory
    luaEnv.AddLoader(new FileLoader(Application.dataPath + "/LuaScripts/", ".lua"));
    luaEnv.AddLoader(new FileLoader(Application.dataPath + "/LuaScripts/", ".lua.txt"));
#else
    //True machine mode to load Lua's bytecode from the PersistentDataPath
    //or StreamingAssetsPath directory.
    var key = Encoding.ASCII.GetBytes("E4YZgiGQ0aqe5LEJ");
    var iv = Encoding.ASCII.GetBytes("5Hh2390dQlVh0AqC");
    var decryptor = new RijndaelCryptography(128, key, iv);

    var loader1 = new FileLoader(Application.streamingAssetsPath + "/LuaScripts/", ".bytes");
    luaEnv.AddLoader(new DecodableLoader(loader1, decryptor));

    var loader2 = new FileLoader(Application.persistentDataPath + "/LuaScripts/", ".bytes");
    luaEnv.AddLoader(new DecodableLoader(loader2, decryptor));
#endif

```

Expand other encryption way

Inherit the Loxodon.Framework.Security.Cryptography.IDecryptor and Loxodon.Framework.Security.Cryptography.IEncryptor interfaces to create a cryptographic decryptor. For specific implementation, refer to the source code of the RijndaelCryptography class.

Extending the Loxodon.Framework.XLua.Editors.EncryptorFactory class can create a factory class for the new encryptor, define the parameters required for encryption as class member variables in the class, and identify the field as serializable, that is, add "SerializeField" Attributes. So that the pre-compilation tool can automatically search for this class and create an editing interface for it. For specific implementation, please refer to the RijndaelCryptographyFactory class.

For example, if you add the following code, you can see the interface as shown in the editing interface.

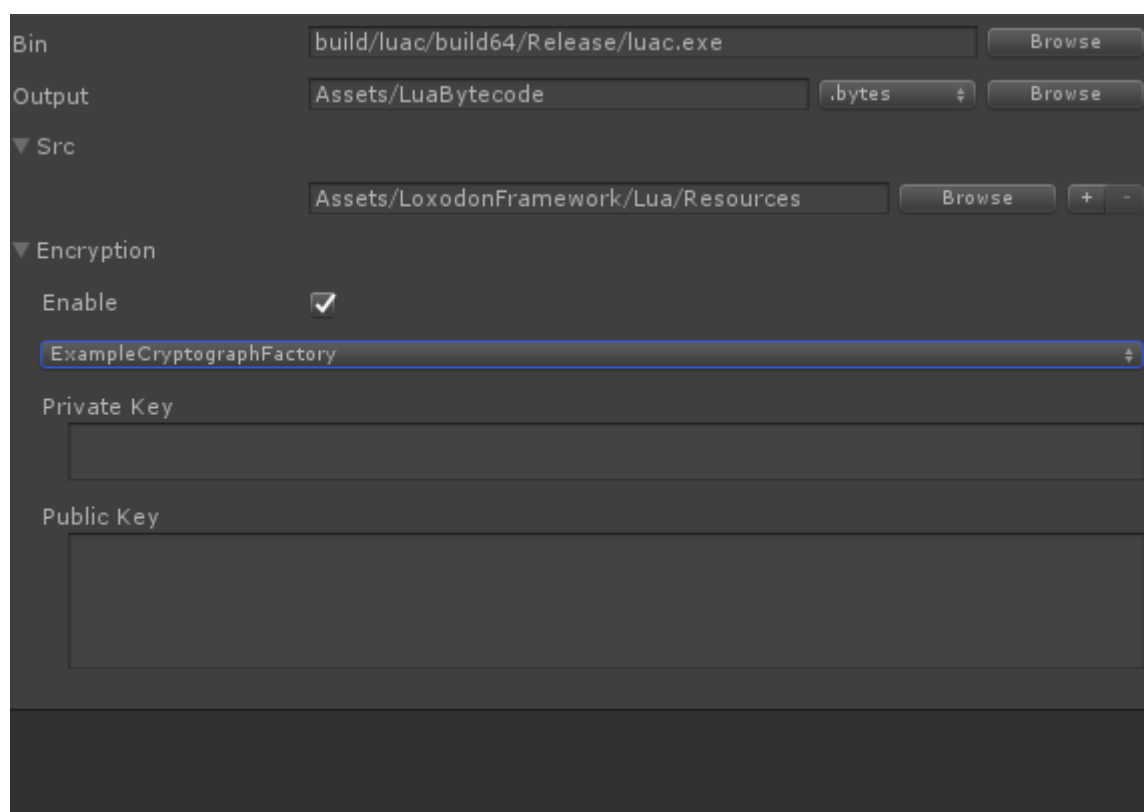
```

public class ExampleCryptographyFactory : EncryptorFactory
{
    [Multiline(2)]
    [SerializeField]
    private string privateKey;

    [Multiline(5)]
    [SerializeField]
    private string publicKey;

    public override IEncryptor Create()
    {
        throw new NotImplementedException();
    }
}

```



Layered architecture

Generally speaking, in order to reduce the complexity of project development, the complex business is decomposed, divided and conquered, and a classification and layered solution will be adopted. Divided from the vertical direction, a complex project can be composed of multiple subsystems, and a subsystem can be composed of multiple business modules. From the horizontal direction, the three-tier architecture can be divided into the presentation layer, the domain layer, and the base layer, and the four-tier architecture can be divided into the presentation layer, the application layer, the domain layer, and the base layer. Generally speaking, you should choose according to the specific situation and complexity of your project. You can find related articles or books on the Internet. If you are interested in DDD programming, you can learn about the knowledge of "Domain Driven Design (DDD Programming)". I won't go into too much detail here, but I just combine it MVVM framework, briefly explain how a game client project should be layered.

Please refer to the directory structure of my project examples

The presentation layer is the layer responsible for presenting information to the user and receiving user input. Combined with the MVVM framework, it can be divided into view layer / control layer (similar to Android's Activity class, view and control are combined in one class), and view model layer.

View

The view layer generally includes windows, view scripts, UI controls, animation scripts, view resources, and other view layer auxiliary tools, such as view locators. Specifically, you can abstract and plan according to your own project situation.

- Window/UIView
Window and view scripts control the life cycle of all views, such as the creation and destruction of subviews and subwindows should be written in this layer of code. If the logic of opening and closing the interface is triggered by functions in the ViewModel layer, then use IDialogService or exchange requests to send events to the view script for execution.
- UI controls (UGUI controls or custom controls)
UI control layer, custom UI controls should be written in this layer, and it is strongly recommended that UI functions be controlled, such as lists, dialog boxes, progress bars, Grid, Menu, etc. should be written as universal UI controls.
- Animation
UI animation layer, for example, you can use DoTween to write various window animations or interface animations, and directly hang them on the UI GameObject. You can refer to my example to write. If it is a window animation, please inherit my UIAnimation or use GenericUIAnimation to implement.
- View locator(IUIViewLocator)
View locator, which uses the view locator to load view templates from Resources or from AssetBundle. Of course, you can refer to my UI view locator to write your own 3D view locator.
- Iteration Action
Interaction behavior. This is an abstraction for window and view creation code reuse. It encapsulates some frequently used interface creation code as interaction behavior.
- ViewModel
The view model layer contains all the view models and subview models. Generally, the view models of Window and View are paired one by one. A window must have a view model, and the subviews under a window should generally have corresponding subviews. model. However, pure view object-encapsulated subview models, such as UserInfoVM, can be shared by multiple views, and when the UserInfoVM property changes, multiple interfaces bound to it will change at the same time.
The view model is not allowed to depend on the objects of the view layer, but the view layer can depend on the view model, so the dependence of the view on the view model is one-way. The view model creates or destroys the view through an interactive request or IDialogService.
The view model directly calls the application layer Service to process the business. The view model can register events to the application layer Service to listen to changes in model object data. For example, if the role information changes, then the role service should trigger an event where the role information changes. When the view model layer receives the event, it updates the values in the role information view model object to trigger all UI interface changes.
 - View model locator (IViewModelLocator)
View model locator, which is used to manage the shared sub-view model, or to save the window view model (such as the window is closed but the view model is not destroyed, the next time you open the window, you can use it to restore the window state). This layer is not necessary, it can be omitted or replaced with another solution.

Application layer (Service)

The application layer is mainly used to express user use cases and coordinate the behavior between objects in different domains. If the design concept of the DDD congestion model is adopted, it is only a very thin layer. It exists as a bridge between the presentation layer and the domain

layer, and provides services for the presentation layer through application services. If the design idea of the traditional anemia model is adopted, it should include all the business logic processing. I don't know much about DDD programming among the students who use the framework, so here I recommend the design idea of the traditional anemia model to develop the game. In my project example, it corresponds to the Services layer.

For example, a game project may include character services, backpack services, equipment services, skill services, chat services, item services, and so on. These services are used to manage character information, items in the backpack, user equipment, user learned skills, chat information, Chat room information, and more. The service caches this information and ensures that they are synchronized with the server through Load or server push. When there is a message update, an event is triggered to notify the view model layer to update. For example, various red dots on the main interface (the state that prompts a new message) can be designed through the events of each service and the red dot status on the view model.

Domain Model

The domain layer is responsible for the expression and processing of business logic and is the core of the entire business. If you program according to DDD, the domain layer generally includes concepts such as entities, value objects, domain services, aggregation, aggregation roots, storage, and factories, because the concepts involved are numerous and persistence needs to be compatible with the CQRS + ES model, and there are considerable thresholds to master. So if you are not very familiar with DDD, I don't recommend designing your code completely according to the DDD programming idea, but adopt the idea of the anemia model. Below, I will only make a simple idea of some concepts to be used in the anemia model. Introduction.

- Entity

Entities must have unique identifiers, such as objects such as account numbers, characters, skills, equipment, and props in the game, which are all entity objects.

- Value Object

The value object is used to describe an object in a certain aspect of the domain that does not have a conceptual identifier. The value object is different from the entity. It has no unique identifier and its attributes are immutable, such as some game table information.

- Repository

The warehousing layer is responsible for functions such as adding, deleting, modifying, and checking the entity objects. Through the warehousing layer, you can read data or persist data. The data can be saved in local Json, xml, SQLite, or on the server through the network.

Infrastructure

The base layer contains the framework, database access components, network components, Log components, Protobuf components, public helper classes and methods.

Contact information

Email: yangpc.china@gmail.com

Website: <https://vovgou.github.io/loxodon-framework/>

QQ group: 622321589

