# 客户端平台接口及使用说明

## 一、设置插件组文件夹

1.将Weifenluo.WinFormsUI.Docking.dll、Platform.Core.dll、Core.xml、log4net.dll、WinForm.exe放在同一文件夹下。

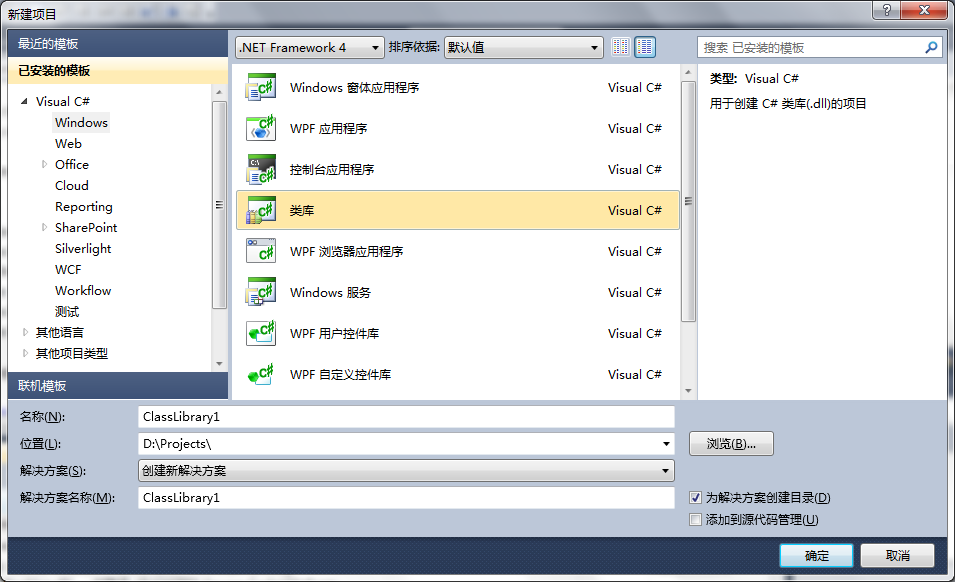
2.在此文件夹内新建名为plugin的文件夹。

3.将MainPlugin文件夹拷贝至plugin文件夹下。

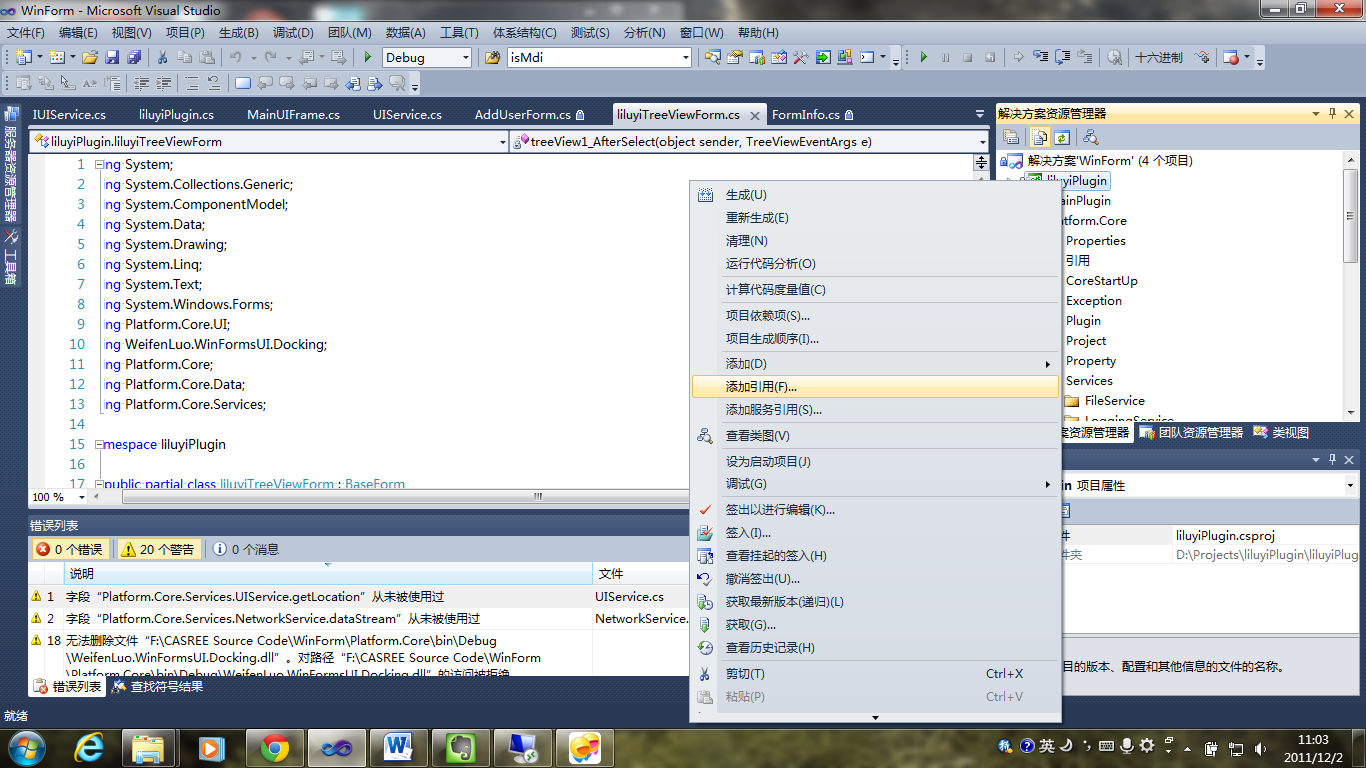
4.在plugin文件夹下新建XXXPlugin文件夹，如liluyiPlugin。

## 二、自主编写工具

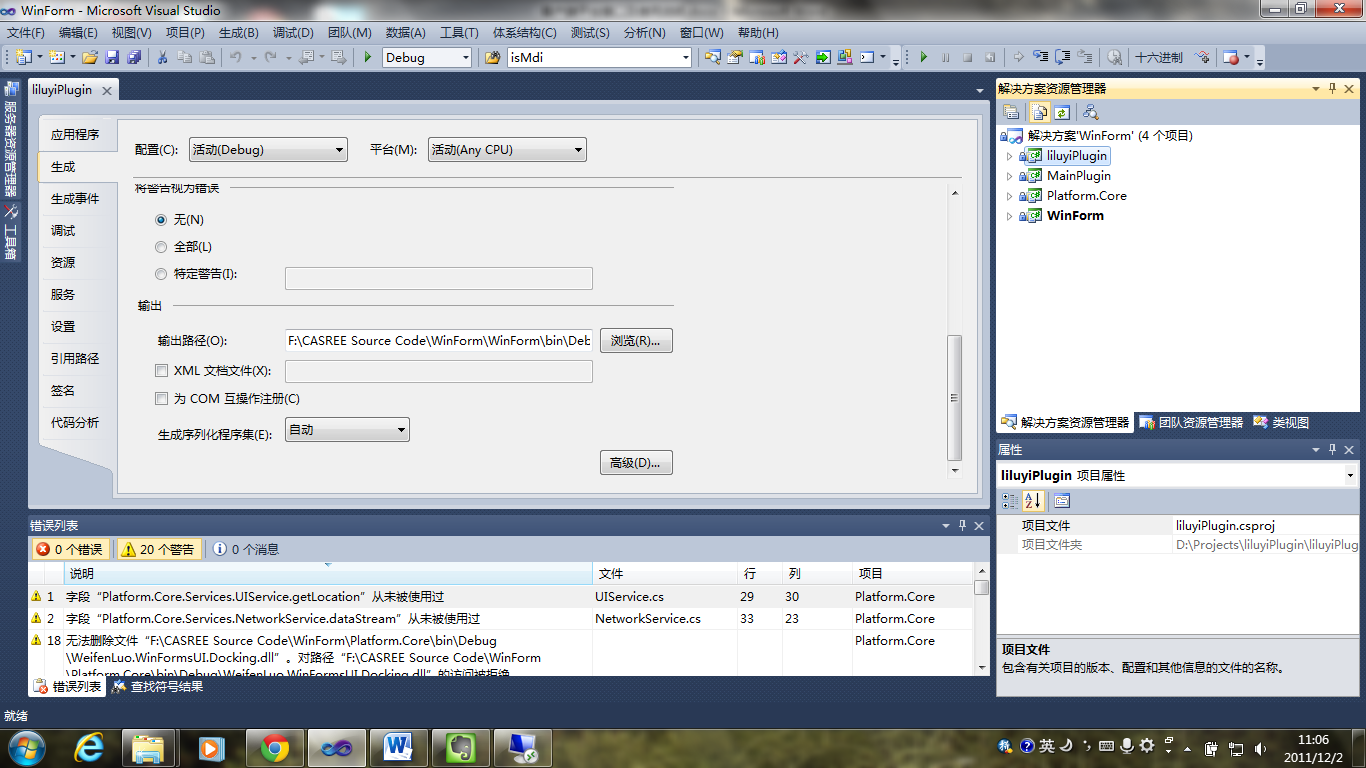
5.打开Visual Studio 2010，新建C#的“类库”类型的工程，如liluyiPlugin。如下图所示：



6.在解决方案资源管理器的工程节点上点击右键，添加对于Weifenluo.WinFormsUI.Docking.dll、Platform.Core.dll、log4net.dll的引用。如下图所示：



7.同样在工程节点上点击右键，选择“属性”，在打开的属性页中，将“生成”标签中的“输出路径”更改至刚才新建的XXXPlugin文件夹下。



这样，编译工程时生成的dll文件将自动生成至该文件夹下，以便主程序启动时动态加载。

8.将项目下.cs文件重命名为任意名称。

## 三、编写工具Plugin类

9.打开该.cs文件，更改namespace为XXXPlugin

10.在该文件中添加引用：

using Platform.Core;

using Platform.Core.UI;

using Platform.Core.Data;

using Platform.Core.Services;

11.将该文件中的类名更改为XXXPlugin，如liluyiPlugin，并实现IPlugin接口，例如：

public class liluyiPlugin:IPlugin

{

private string token;

private string projectsuffix = string.Empty;

private string projectclassfullname = string.Empty;

private string mutableresourcefullname = string.Empty;

#region IPlugin Members

public AbstractProject GetDefaultProject()

{

return new liluyiProject();

}

public AbstractProject GetProjectFromPath(string path)

{

liluyiProject project = new liluyiProject();

return project;

}

public AbstractProjectData GetDefaultProjectData()

{

return new liluyiProjectData();

}

public AbstractProjectData GetProjectDataFromPath(string path)

{

liluyiProjectData projectdata = new liluyiProjectData();

return projectdata;

}

public string MutableResourceClassFullName

{

get

{

return mutableresourcefullname;

}

set

{

mutableresourcefullname = value;

}

}

public string ProjectClassFullName

{

get

{

return projectclassfullname;

}

set

{

projectclassfullname = value;

}

}

public string ProjectSuffix

{

get

{

return projectsuffix;

}

set

{

projectsuffix = value;

}

}

public string Token

{

get

{

return token;

}

set

{

token = value;

}

}

#endregion

public liluyiPlugin()

{

}

}

此类将用于平台启动时提供工具的工具信息类、数据类、UI类，平台将调用加载。

12.务必更改各个属性下的默认get与set方法，更改为上面代码所示。

## 四、编写工具数据类

13.在该工程上点击右键，新建类，名为XXXProject，如liluyiProject，将namespace改为liluyiPlugin，与上一个类的命名空间相同。

14.添加引用：

using Platform.Core;

using Platform.Core.UI;

using Platform.Core.Data;

using Platform.Core.Services;

using System.Xml.Serialization;

using System.Runtime.Serialization.Formatters.Binary;

using System.Runtime.Serialization;

using System.IO;

using System.Reflection;

15.使该类继承于平台提供的工程文件基类——AbstractProject类，如

[System.Serializable]

public class liluyiProject : AbstractProject

{

public liluyiProject()

{

}

public override string Suffix

{

get { return "liluyi"; }///务必返回工程文件后缀

}

}

16.修改Suffix属性的get方法，返回值即为将来工程文件的后缀名。且务必与将要添加的Addin文件中的Suffix属性一致。

17.本类为工具的工程数据类，平台提供公共菜单，实现将本类的实例保存至磁盘，并提供打开等功能。

18.因此，可以在上面的类中继续添加自己需要的字段、属性等。

19.上述代码中黄色底色部分[System.Serializable]为序列化的标签，只有打了该标签，工程文件才能被平台所序列化（即保存至本地磁盘）和反序列化（即从本地磁盘打开）。

20.用13-19步类似的步骤添加liluyiProjectData类，此类需继承与AbstractProjectData，如：

[System.Serializable]

public class liluyiProjectData : AbstractProjectData

{

public int nodenum=0;

public string username = "jdslfjdsla;gh";

public string password = "pwpwpwpwpw ";

public liluyiProjectData() { }

public override string Suffix

{

get { return "liluyi"; }

}

public int NodeNum

{

get { return nodenum; }

set { nodenum = value; }

}

public string UserName

{

get { return username; }

set { username = value; }

}

public string Password

{

get { return password; }

set { password = value; }

}

}

21.暂时可以不自主添加字段，本类将来将用于支撑工具间数据流的传输。

## 五、编写工具UI类

22. 在该工程上点击右键，新建类，名为XXXMutableResource，如liluyiMutableResource，将namespace改为liluyiPlugin，与上一个类的命名空间相同。

23.添加引用：

using System.Windows.Forms;

using System.Drawing;

using Platform.Core;

using Platform.Core.UI;

using Platform.Core.Data;

using Platform.Core.Services;

using WeifenLuo.WinFormsUI.Docking;

24.使该类继承与平台提供的界面资源基类MutableResource，并实现基类的属性，务必添加各属性的get与set方法，如：

public class liluyiMutableResource : MutableResource

{

private BaseForm viewform = new liluyiTreeViewForm();

private BaseForm tabform = new liluyiTabViewForm();

private BaseForm infoform = new liluyiInfoViewForm();

private BaseForm serverform = new liluyiServerViewForm();

private ToolStripMenuItem[] menus;

private ToolStrip[] tools;

private Dictionary<string, Platform.Core.UI.BaseForm> toolformdictionary=new Dictionary<string,BaseForm>();

private Dictionary<string, FormLoc> formlocationdictionary = new Dictionary<string, FormLoc>();

#region MutableResource

public override Dictionary<string, Platform.Core.UI.BaseForm> ToolFormDictionary

{

get { return toolformdictionary; }

}

public override Dictionary<string, FormLoc> FormLocationDictionary

{

get { return formlocationdictionary; }

}

public override BaseForm ViewForm

{

get

{

return viewform;

}

set

{

}

}

public override BaseForm TabForm

{

get

{

return tabform;

}

set

{

}

}

public override BaseForm InfoForm

{

get

{

return infoform;

}

set

{

}

}

public override BaseForm ServerForm

{

get

{

return serverform;

}

set

{

}

}

public override ToolStripMenuItem[] PluginMenus

{

get { return menus; }

}

public override ToolStrip[] PluginTools

{

get { return tools; }

}

#endregion

}

public liluyiMutableResource()

{

//构造方法，添加工具启动时窗体

}}

25.可以在该类中添加新的字段，如其他自定义窗体，如

private AddUserForm adduserform1=new AddUserForm();

private AddUserForm adduserform2=new AddUserForm();

26. 在构造方法中添加代码，以便将工具启动时需要显示的界面元素注册至平台，如

public liluyiMutableResource()

{

//可以添加菜单、工具栏等

menus = new ToolStripMenuItem[2];

for (int i = 0; i < 2; i++)

{

menus[i] = new ToolStripMenuItem("自Á?定¡§义°?菜?单Ì£¤" + i.ToString());

for (int j = 0; j < 3; j++)

{

menus[i].DropDownItems.Add("自Á?定¡§义°?子Á¨®菜?单Ì£¤" + i.ToString() + "\_" + j.ToString());

}

}

///**添加工具启动时窗体：以下三步必不可少，例如添加viewform：**

//首先设置窗体ID

viewform.Name = "TreeViewForm";

//将窗体ID加入窗体词典中

toolformdictionary.Add(viewform.Name, viewform);

//将窗体位置注册至窗体位置词典

formlocationdictionary.Add(viewform.Name, new FormLoc(DockState.DockLeft));

tabform.Name = "TabViewFrom";

toolformdictionary.Add(tabform.Name, tabform);

formlocationdictionary.Add(tabform.Name, new FormLoc(DockState.Document));

infoform.Name = "InfoViewForm";

toolformdictionary.Add(infoform.Name, infoform);

formlocationdictionary.Add(infoform.Name, new FormLoc("TabViewForm", DockAlignment.Bottom, 0.3));

serverform.Name = "ServerViewForm";

toolformdictionary.Add(serverform.Name, serverform);

formlocationdictionary.Add(serverform.Name, new FormLoc(DockState.DockLeftAutoHide));

//添¬¨ª加¨®工¡è具?自Á?定¡§义°?启?动¡¥时º¡À窗ä¡ã体¬?及¡ã其?位?置?

adduserform1.Name = "AddUserFormIni";

toolformdictionary.Add(adduserform1.Name, adduserform1);

formlocationdictionary.Add(adduserform1.Name, new FormLoc(DockState.DockRight));

//不?设¦¨¨置?自Á?定¡§义°?窗ä¡ã体¬?位?置?时º¡À默?认¨?显?示º?于®¨²文?档Ì¦Ì区?

adduserform2.Name = "AddUserFormIni2";

toolformdictionary.Add(adduserform2.Name, adduserform2);

formlocationdictionary.Add(adduserform2.Name, new FormLoc(adduserform1.Name,DockAlignment.Bottom,0.5));

#region MutableResource

27. 注意：上述代码中添加工具启动时窗体时：以下三步必不可少：

（1）设置窗体ID，并设为窗体的Name属性

（2）在本工具窗体词典中注册（此词典在运行时由平台维护）。词典以窗体ID为键，以窗体实例为值。

（3）在本工具窗体位置词典中注册窗体位置（此词典在运行时由平台维护）。词典以窗体ID为键，以窗体位置为值。窗体位置设置方式见29.

例如添加viewform：

//首先设置窗体ID

viewform.Name = "TreeViewForm";

//将窗体ID加入窗体词典中

toolformdictionary.Add(viewform.Name, viewform);

//将窗体位置注册至窗体位置词典

formlocationdictionary.Add(viewform.Name, new FormLoc(DockState.DockLeft));

28.注意：平台提供了四个窗体，**默认启动时全部不显示**！

各工具可以自主选择是否启动时显示，建议显示。而且这四个窗体的位置可以由工具自己设定。显示及位置见27所示。

29.窗体停靠位置采用的是平台提供的FormLoc类，这个类有两个构造方法，一个构造方法接受Weifenluo的Dockstate值，另一个构造方法由（本窗体要停靠的窗体ID，停靠位置DockAlignment，位置比例）三个参数组成，见上面代码中的实例）。

30.可以在启动时就显示一些工具自主定义的窗体，如PropertyGrid等，只需按照上述方法设置即可。例如：在类中声明两个启动时显示的窗体：

private AddUserForm adduserform1=new AddUserForm();

private AddUserForm adduserform2=new AddUserForm();

在构造方法中对应添加：

//工具自设启动时窗体1

adduserform1.Name = "AddUserFormIni";

toolformdictionary.Add(adduserform1.Name, adduserform1);

formlocationdictionary.Add(adduserform1.Name, new FormLoc(DockState.DockRight));

//工具自设启动时窗体2

adduserform2.Name = "AddUserFormIni2";

toolformdictionary.Add(adduserform2.Name, adduserform2);

formlocationdictionary.Add(adduserform2.Name, new FormLoc(adduserform1.Name,DockAlignment.Bottom,0.5));

## 六、添加窗体类

31.上面代码中全部的窗体类均由各工具自己写，但是务必使窗体类继承与平台提供的BaseForm基类，例如：

public partial class liluyiTabViewForm : BaseForm

{

public liluyiTabViewForm()

{

InitializeComponent();

}

private void liluyiTabViewForm\_Load(object sender, EventArgs e)

{

}

}

32.各工具务必添加TabViewForm、TreeViewForm、InfoViewForm、ServerViewForm四个类。其他自定义类可以自主添加。全部窗体的HideOnClose属性务必设置为TRUE！

## 七、窗体间通信方法

33.可以随时获取工具当前的工程数据类liluyiProject的实例，平台提供的ProjectService接口即可，如：

liluyiProject project = new liluyiProject();

AbstractProject currentproject = ProjectManager.ProjectManagerSington.GetCurrentProject();

project = currentproject as liluyiProject;

这样，就可以对于当前工程数据中的对应字段进行赋值、更改等操作了，如：

project.s1 = this.usernametextBox.Text;

project.s2 = this.passwordBox.Text;

34.可以随时根据窗体ID获取平台当前正在维护着的窗体实例，如

AddUserForm form = (AddUserForm)ServicesManager.ServicesManagerSingleton.UIService.GetUserForm(null, new UserUIEventArgs(“//窗体名称字符串”,null));

这样就可以修改该窗体中对应属性并刷新，刷新方式如：

ServicesManager.ServicesManagerSingleton.UIService.ShowForm(this, new UserUIEventArgs(“//窗体名称”));

35.可以随时动态窗台窗体并注册至平台，由平台负责加载、维护与显示，加载方式：

ServicesManager.ServicesManagerSingleton.UIService.AddMutableResourceSelf(adduser2, new UserUIEventArgs(adduser2.Text, new FormLoc(DockState.DockBottom)));

36.注意ServicesManager.ServicesManagerSingleton.UIService.AddMutableResourceSelf的参数为两个，一是BaseForm类型的窗体实例，一是UserUIEventArgs。UserUIEventArgs构造函数接收两个参数，一是窗体ID，二是窗体位置类FormLoc的实例。

## 八、公共菜单项

37.“文件”菜单：允许工程的创建、打开、保存、另存为、关闭、转为XML。点击对应菜单项即可实现上述功能。

38.“连接”菜单：运行程序连接服务器、获取服务器端项目列表从而配置当前工程所属项目、向服务器端发送本地任意文件、向服务器端传输工程XML流、从服务器端下载任意文件、从服务器端下载工程XML流等功能。点击对应菜单项即可实现上述功能。