ALC平台使用手册

v2.2

hanson.yang，WILLIAM.HU

2018

# 目录

[目录 1](#_Toc23807)

[1. ALC平台简介 4](#_Toc7241)

[2. 开发环境搭建 4](#_Toc27800)

[2.1. 开发环境需求 4](#_Toc19028)

[2.2. 新建DLL项目 4](#_Toc6546)

[2.3. 配置启动目标 6](#_Toc13622)

[2.4. 配置输出路径 7](#_Toc15907)

[2.5. 添加引用 7](#_Toc26626)

[2.6. 创建插件 9](#_Toc21472)

[2.7. 小结 10](#_Toc28038)

[3. 配置文件说明 11](#_Toc31684)

[3.1. 版本配置 11](#_Toc9898)

[3.2. 用户权限 12](#_Toc12167)

[3.3. 流程控制按钮配置 12](#_Toc6383)

[3.4. TCP服务器端口配置 12](#_Toc11948)

[3.5. 插件配置 12](#_Toc7625)

[3.6. 小节 13](#_Toc3302)

[4. 插件编写说明 14](#_Toc25063)

[4.1. 成员 14](#_Toc13511)

[4.1.1. \_moduleType 14](#_Toc23468)

[4.1.2. SendMessageToNetwork 15](#_Toc2269)

[4.1.3. BroadcastMessageToPlugins 15](#_Toc23132)

[4.1.4. \_jsonMsgHandlers 15](#_Toc24221)

[4.1.5. \_binaryMsgHandlers 15](#_Toc7118)

[4.1.6. ReceiveData数据结构说明 15](#_Toc10707)

[4.2. 方法 16](#_Toc1190)

[4.2.1. MessageHandler 16](#_Toc9334)

[4.2.2. GetControl 18](#_Toc14177)

[4.2.3. GetForm 19](#_Toc24062)

[4.2.4. GetConfigView 21](#_Toc26810)

[4.2.5. UpdateModuleStatus 22](#_Toc14017)

[4.2.6. Load 22](#_Toc1067)

[4.2.7. Initialize 23](#_Toc11998)

[4.2.8. Start 23](#_Toc2464)

[4.2.9. Stop 23](#_Toc28086)

[4.2.10. Pause 24](#_Toc5523)

[4.2.11. Resume 24](#_Toc19979)

[4.2.12. Reset 24](#_Toc20263)

[4.2.13. Dispose 24](#_Toc23611)

[4.2.14. Log 24](#_Toc5116)

[4.2.15. ShowMsgBox 24](#_Toc24369)

[4.2.16. SaveCsv 24](#_Toc14295)

[4.2.17. Error 25](#_Toc2564)

[4.2.18. GetSystemStatus 25](#_Toc7761)

[4.2.19. GetUserAuthority 25](#_Toc9049)

[4.2.20. ButtonClickRequire 25](#_Toc26407)

[5. ALCSystem类 25](#_Toc6399)

[5.1. 事件 25](#_Toc27105)

[5.1.1. UserAuthorityChanged 26](#_Toc26986)

[5.1.2. FormShown 27](#_Toc25584)

[5.1.3. CanInit、CanStart、CanStop、CanPause、CanResume、CanReset 27](#_Toc23023)

[5.2. 方法 27](#_Toc5230)

[5.2.1. 插件中已介绍的方法 28](#_Toc26607)

[5.2.2. GetSoftwareVersion 28](#_Toc6946)

[6. 提供的功能 28](#_Toc5206)

[6.1. 数据库操作 28](#_Toc23680)

[6.2. SSH远程拉起 29](#_Toc25626)

[6.3. TCP客户端 29](#_Toc31414)

[6.3.1. string客户端 29](#_Toc1433)

[6.3.2. binary客户端 30](#_Toc13311)

[7. 附件 30](#_Toc18953)

[7.1. 通讯接口文档 30](#_Toc14563)

[7.2. 实例代码 30](#_Toc23391)

# ALC平台简介

ALC全称为Assembly Line Control，用于自动化线体的整线控制。如整条线的开始、停止、复位、暂停、继续等流程控制，监控线体中个设备的运行情况，并收集运行中的数据，如动作信息、告警、UPH等。

在实际自动化项目开发过程中，我们发现几乎所有的项目都会用到某些相同的功能，如整体线的启停、复位等逻辑控制，如运行日志的记录、用户管理、网络连接、模式管理、界面布局等。并且通常，这些功能不属于核心的业务逻辑，但总是会占用整个项目开发的一半以上时间，甚至一些比较简单的项目会占用到百分之八十的时间。更有甚者，由于不同的开发人员的水平不同，像日志、网络连接等一些功能，做出来的稳定性不是很高，常常为了调试这些要付出大量的时间。鉴于此，我们推出了该ALC平台，把开发人员从这些重复的、繁琐的开发中解放出来，可以集中精力在每个具体项目的核心业务逻辑上，从而可以大大减少开发和调试的时间，提供工作效率。

该ALC平台，是基于很多成熟项目的上位机软件中脱离出来的，因此平台的稳定性是可以保证的。同时为了广泛支持大多数项目，做了相应的调整。比如我们把不同项目的核心逻辑，抽象成通用的接口，不同项目的开发人员只需要开发实现这些接口的插件，并在XML文件中做简单的配置，就可以完成一个完整的项目的开发。

# 开发环境搭建

在正式开发之前，我们需要先搭建开发环境，以方便开发的时候可以方便的调试。

## 开发环境需求

操作系统：暂时只支持Windows，win7及以上

IDE：Visual Studio 2015 / 2017（推荐）

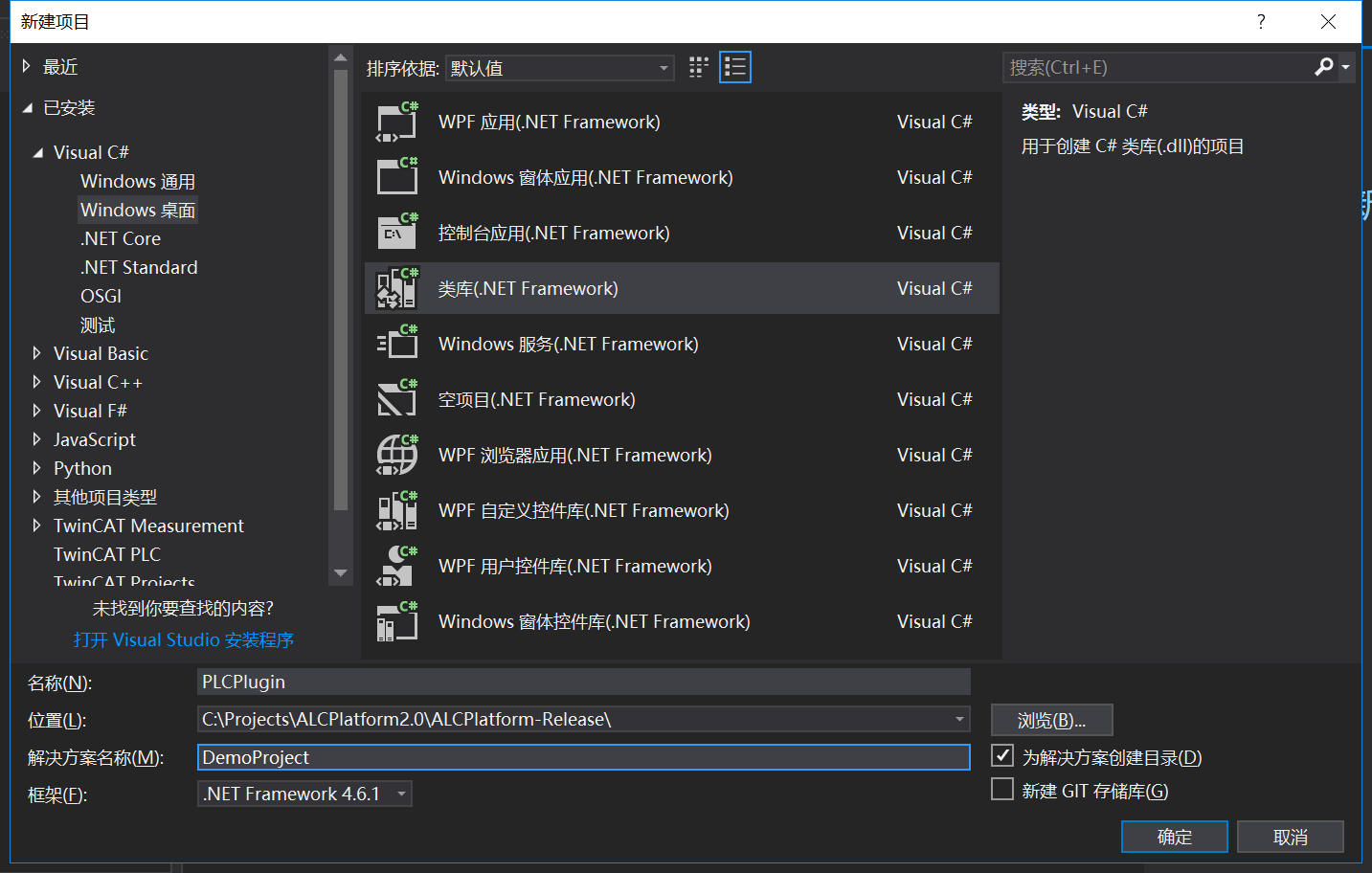
运行环境：.NET Framework 4.5以上

数据库（若需要）： MySQL 5.7

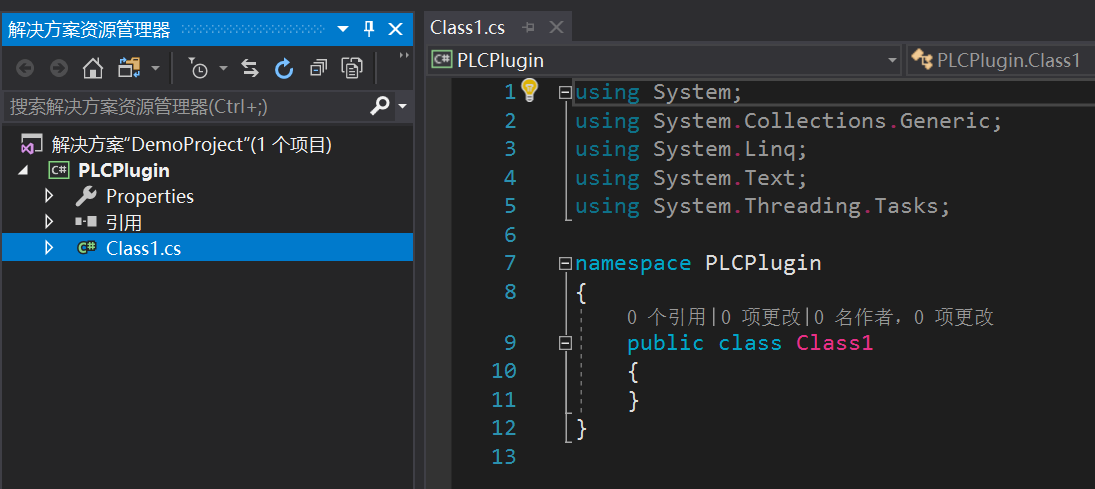
## 新建DLL项目

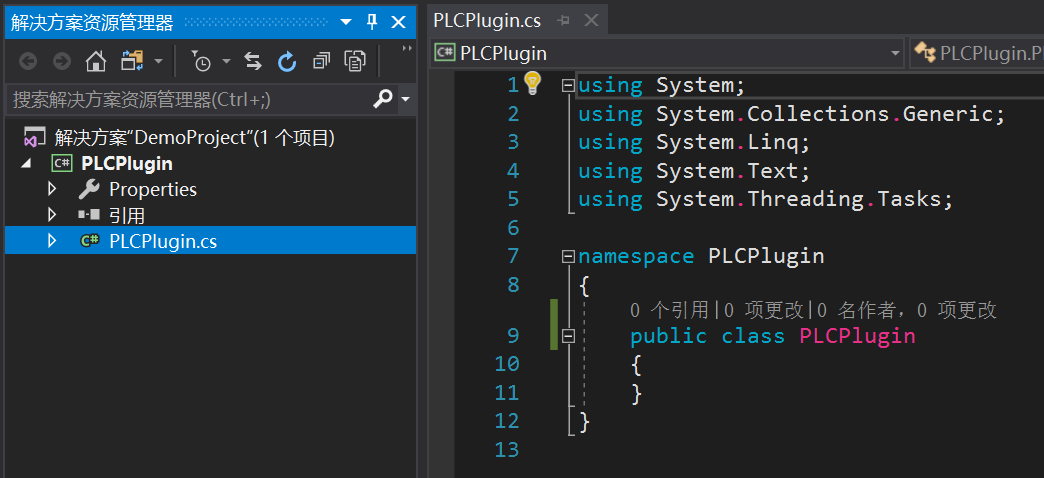
准备好以上环境后，我们可以进入到插件的开发。所谓“插件”，即是一个动态链接库文件（DLL）。该DLL可以依赖其它的DLL。

首先，打开VS2015/VS2017。如下图所示，选择新建项目，在弹出的“新建项目”对话框，选择模板→Visual C#→Class Library，配置好项目名称、所在路径和解决方案名称，（在本例中项目名称我们配置成PLCPlugin，解决方案为DemoProject）点击“OK”,即创建出一个DLL项目。



生成的项目如下图所示，我们可以把默认的Class1.cs文件名改为我们想要的符合逻辑的名字，在本例中我们改为“PLCPlugin”。

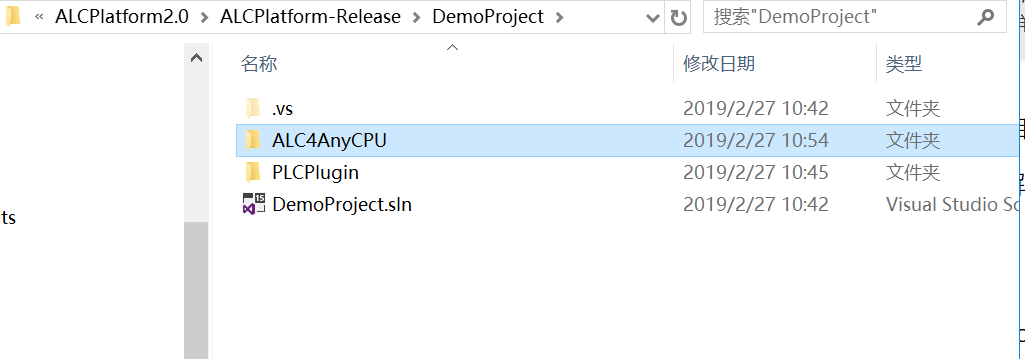


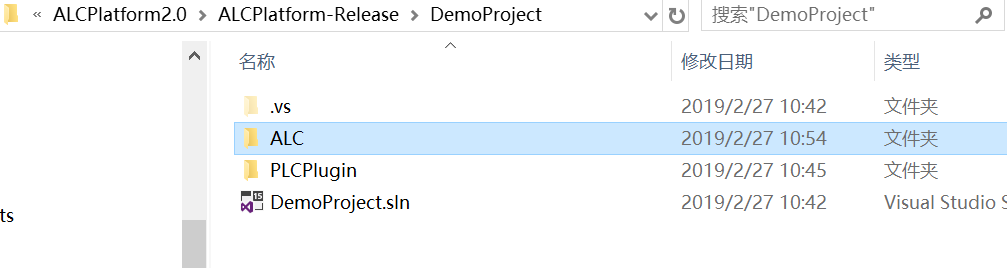


## 配置启动目标

为了方便我们写插件的时候可以直接调试代码，我们可以配置启动目标。

首先，需要把我们提供的ALC平台的程序放到任意位置（推荐放在解决方案文件夹下）。可将文件夹重命名为任意名称，这里重命名为ALC，如下：

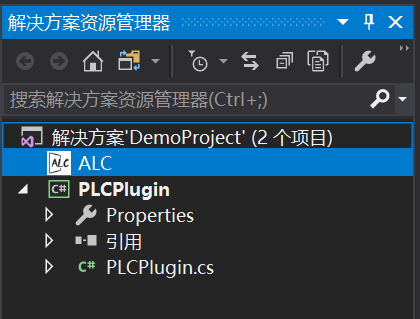




接下来，有两种方式配置启动目标。

方法1（推荐）：

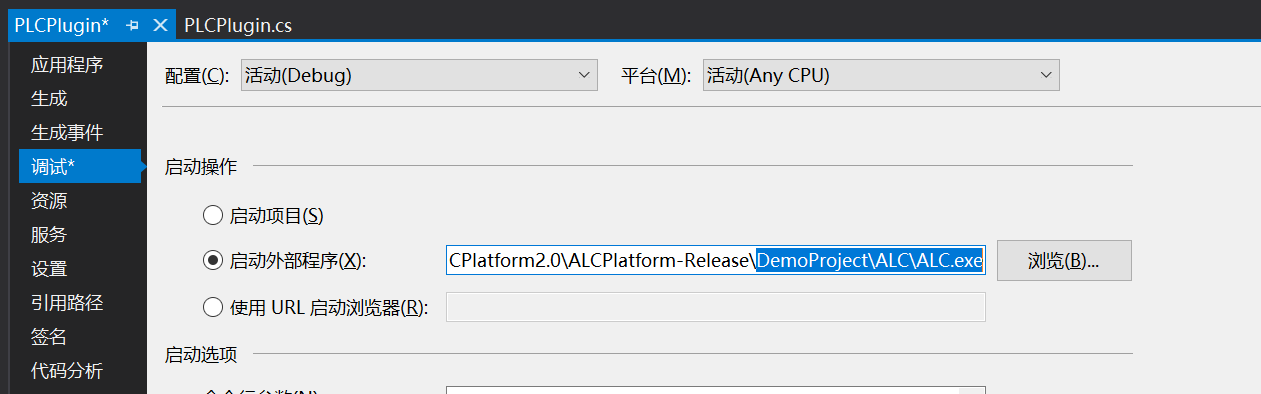
右键解决方案→添加→现有项目，选择ALC平台程序中的“ALC.exe”，添加成功如下图：



右键解决方案视图中的ALC→设为启动项。

方法2：

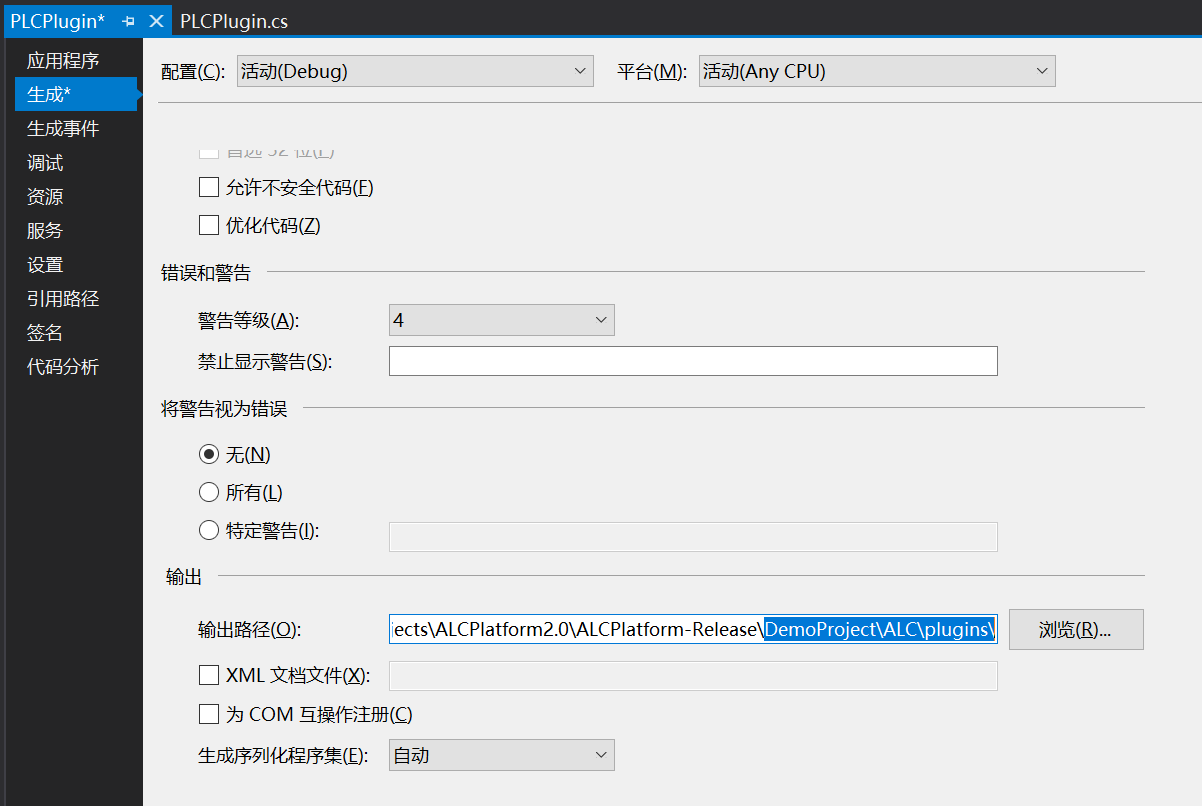
右键点击新建的DLL项目，选择属性→调试，选择“启动操作”里面的“启动外部程序”，并选择ALC平台程序的“ALC.exe”。



## 配置输出路径

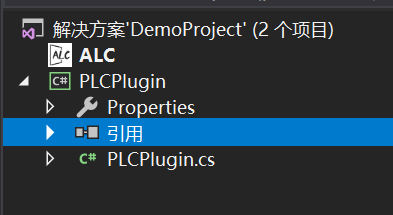
为了方便程序发布，最好配置插件DLL的输出路径，直接输出到ALC程序包里面的plugins文件夹中，若没有该文件夹新建一个即可。

在解决方案中，右键点击新建的DLL项目，选择属性→生成，在“输出”中配置“输出路径”，路径设置为ALC程序包内的plugins目录。

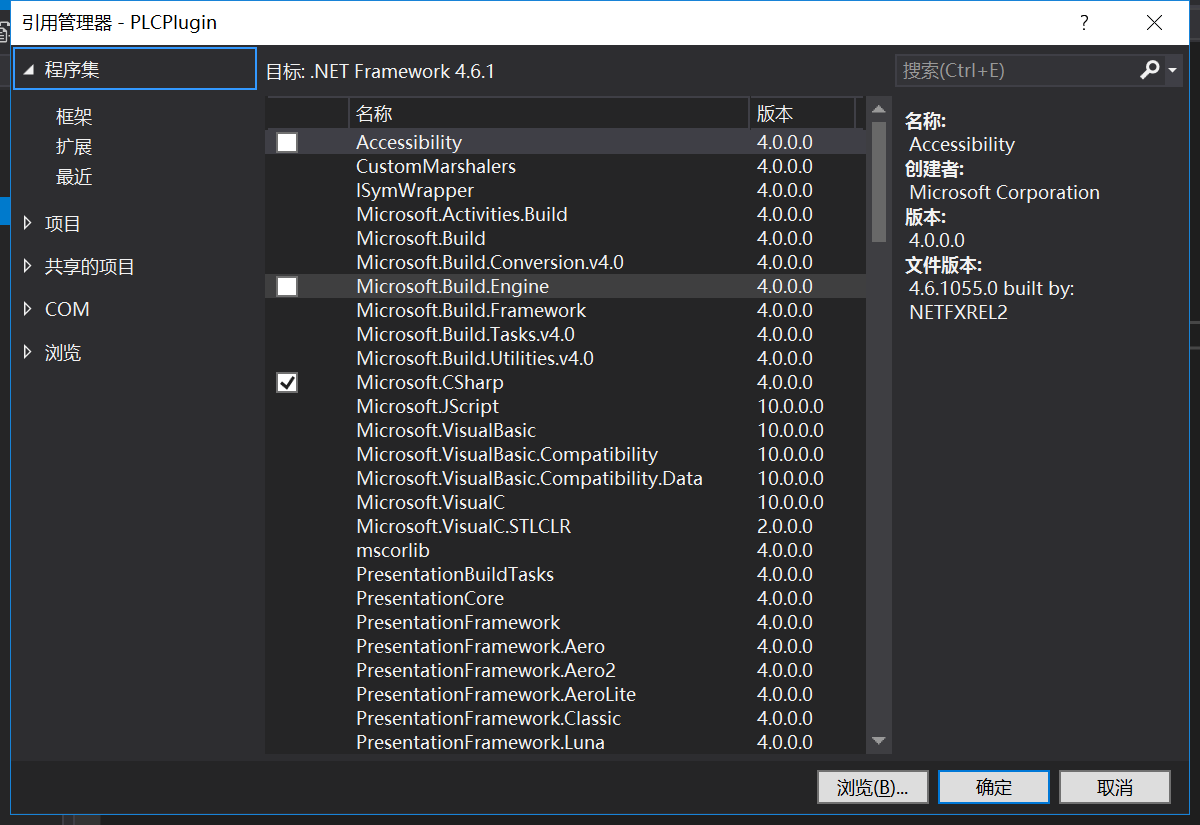


## 添加引用

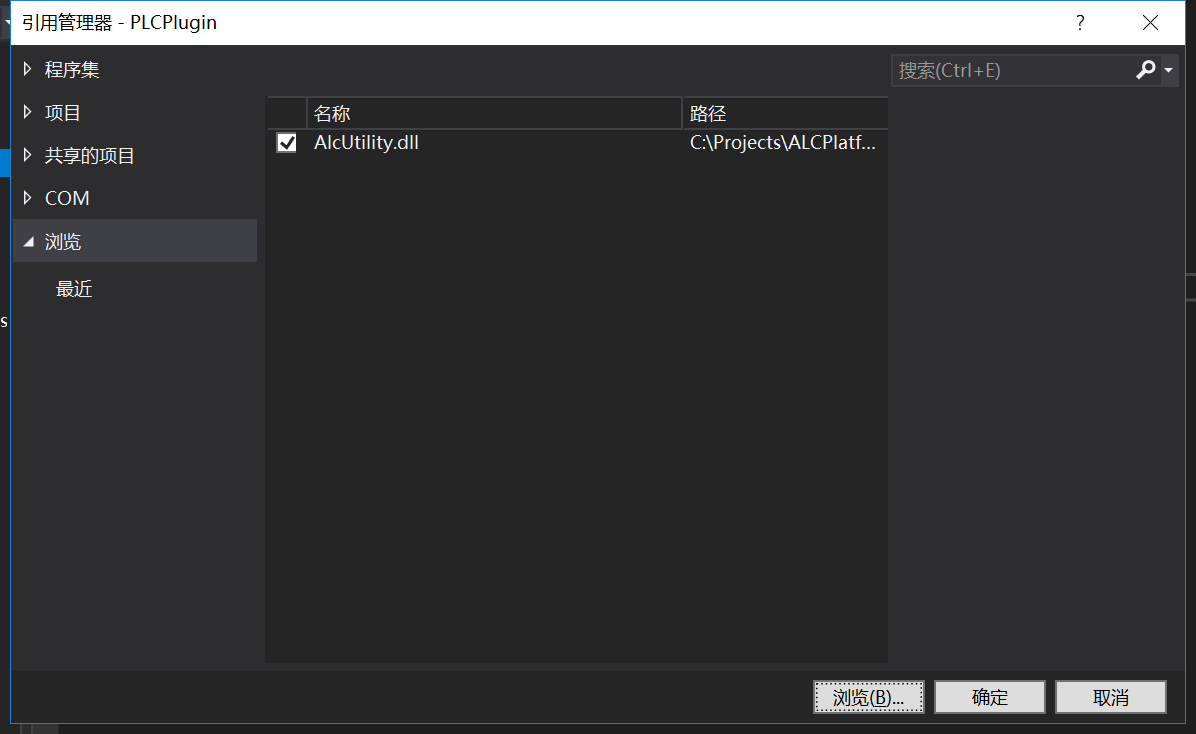
在解决方案中，右键点击新建的DLL项目下的“引用”，



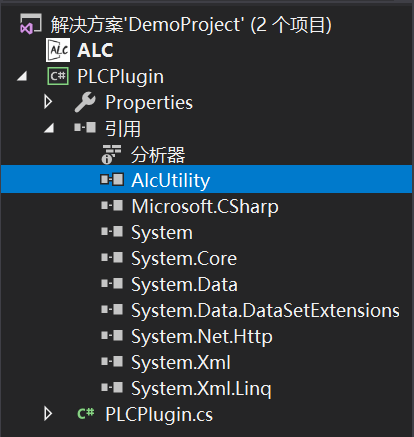
选择“添加引用”，会弹出一个对话框，如下图：



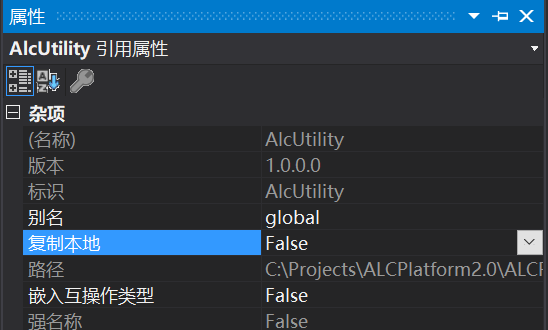
选择右下角的“浏览”,在文件管理器中选择ALC程序包中的一个名为“AlcUtility.dll”的文件。添加后如下：



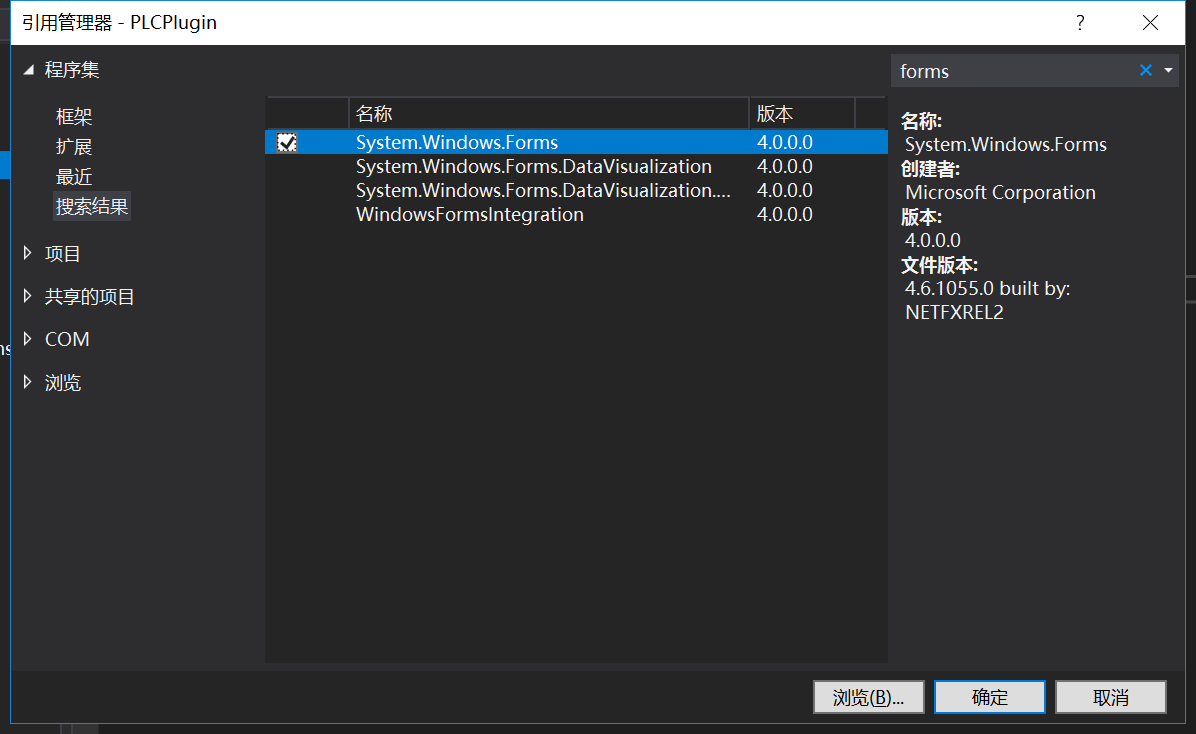
点击确定，然后在解决方案视图中如下：



右键“AlcUtility”→属性，将复制到本地改为False：

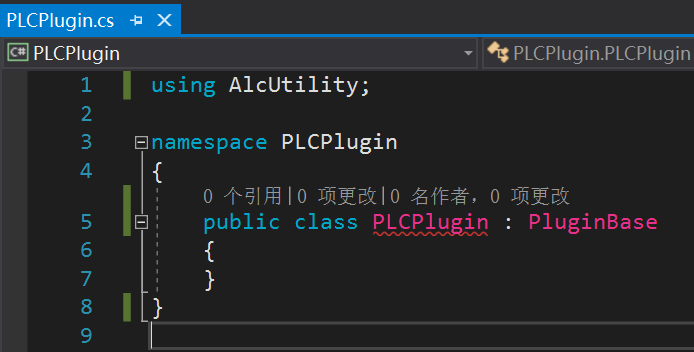


另外，由于我们的接口定义了UI的实现，还需要添加一个System.Windows.Form的引用。同样的方法添加，可以在右上角查找框中输入“forms”，勾选上该应用并点击“确定”。

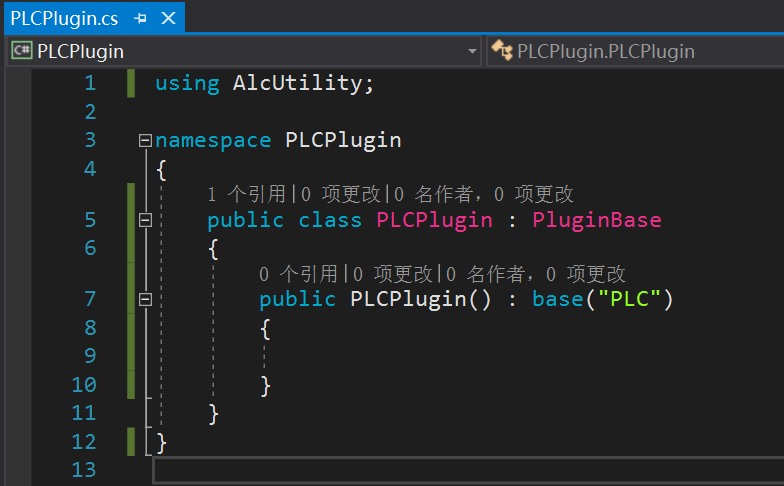


## 创建插件

添加完引用后，现在可以创建插件了。在代码文件（本例中为PLCPlugin.cs）添加“using AlcUtility;”语句，并且让该类继承“PluginBase”基类。添加完如下图：



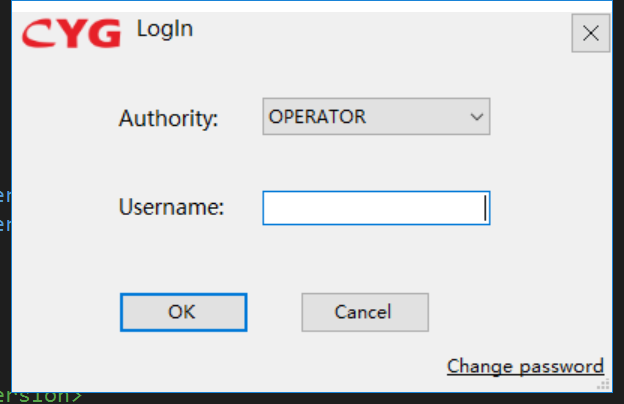
从上图中可以看出，“PLCPlugin”下面有红色波浪线，为语法错误提示。这是因为我们需要为基类的构造函数提供参数，以指明该插件的moduleType，本例中moduleType为“PLC”，具体做法如下：



至此，插件开发的准备工作完成。接下来就可以参照后面的介绍，完善插件的功能。

## 小结

至此，我们可以启动调试一下，来检查我们的环境配置是否正常。点机debug或者快捷键F5，会弹出ALC程序的登录框。点击登录框的OK按钮，会弹出ALC平台的主界面。登录框如下图所示：



主界面如下图所示：



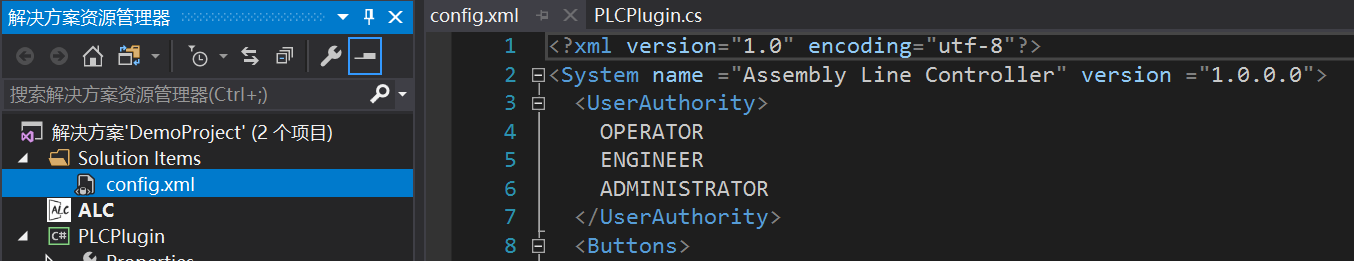
等按照下一节的介绍在配置文件中配置好插件相关信息后，我们的插件就会自动显示到ALC主界面上。

# 配置文件说明

插件添加好后，需要根据具体项目的业务定义及插件的开发情况，来进行相关配置，才能正确加载插件并显示所期望的流程控制按钮。

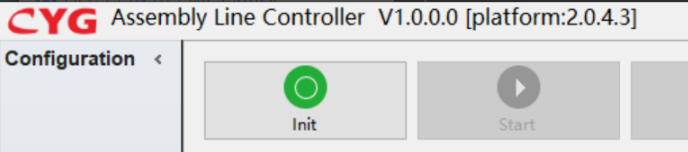
配置文件名为config.xml，在ALC的运行程序（ALC.EXE）目录下。

我们可以把它添加到我们的解决方案中来，右键解决方案→添加→现有项，选择config.xml文件，添加好后如下图：

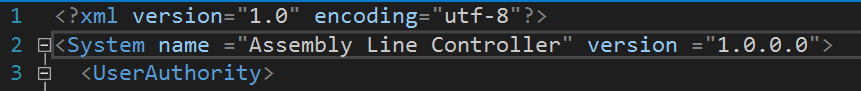


## 版本配置

ALC主界面左上角会显示软件的名称和版本号，和平台的版本号，如图：

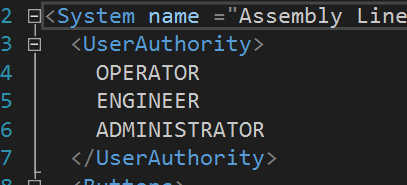


软件名称和版本就在配置文件中System元素中的name属性和version属性中配置：



## 用户权限

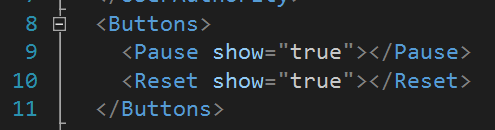
软件开启时，需要选择权限，输入相应的密码才能正常运行。其中“Operator”权限不需要密码，其他权限默认密码和权限名称相同。配置文件中已默认配置了三个权限：



需要不同的权限直接在这里修改即可。

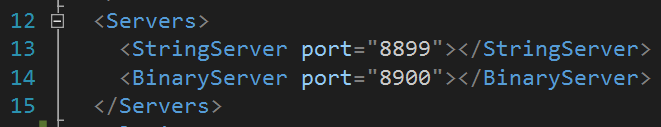
## 流程控制按钮配置

ALC主界面上默认提供四个按钮，分别是Init初始化按钮， start/stop开始/停止按钮，reset复位按钮，pause/resume暂停/继续按钮。其中，Init初始化按钮和start/stop开始/停止按钮是必须要的，因为每个项目必定有初始化流程和开始结束流程，所以这个按钮会一直显示。其它的按钮，如果业务流程中没有定义该流程，则可以不用显示。可以在config.xml中通过修改show属性来实现是否该按钮需要显示。



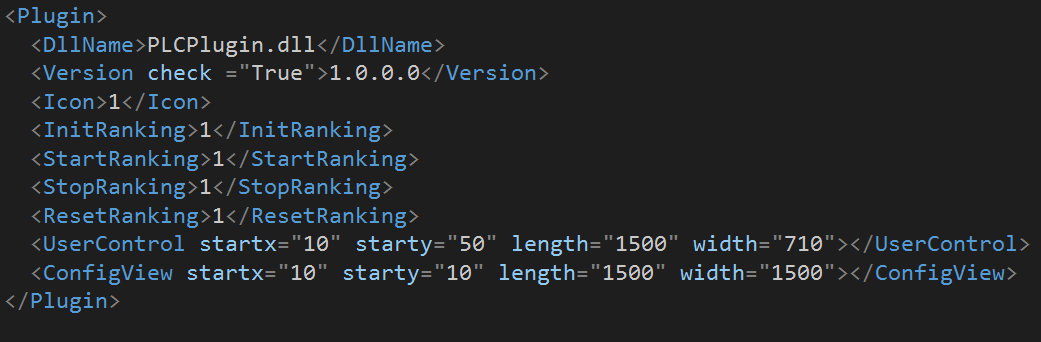
## TCP服务器端口配置

ALC默认作为服务器，提供两种接口的TCP服务器。一种Json格式的字符串服务器，一种二进制格式的服务器。详细参加附件7.1节通讯接口文档。端口配置如下：



## 插件配置

插件需要配置一些信息，如下：

 其中**DllName**是该项目release后生成的DLL文件名。

**Version**是该插件的版本号，check属性是标识是否需要ALC平台来验证这个版本号是否正确。

**Icon**是ALC平台内置了几个图标，用来显示在导航栏上。

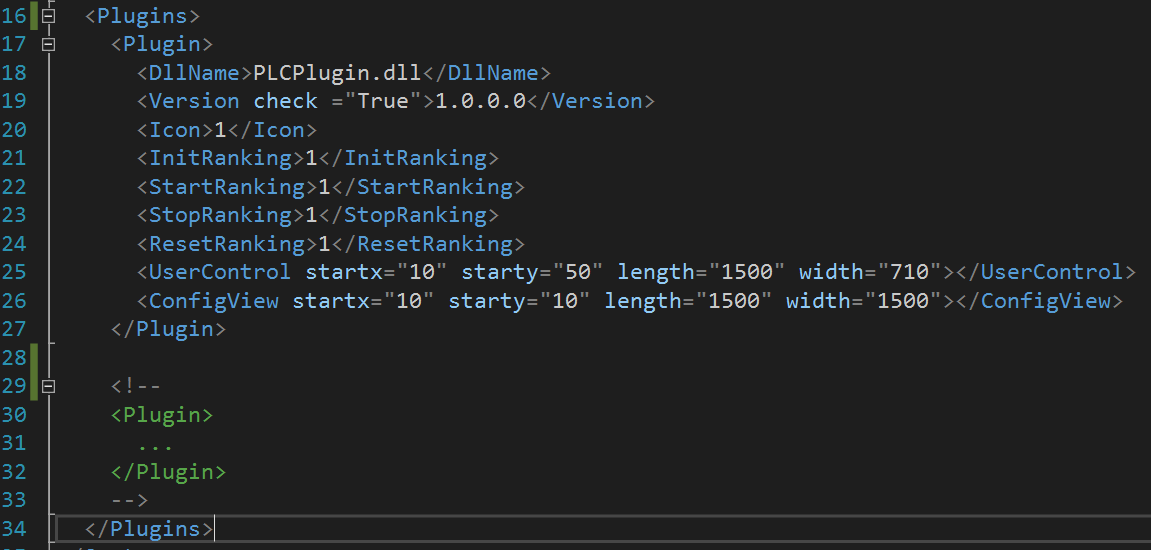
**StartRanking**、**StopRanking**、**ResetRanking**是用来分别设置start、stop、reset的时候，插件的执行顺序，为大于0的整数。顺序相同的时候同时执行，不同的时候，要等比自己顺序小插件执行完毕后才执行。

**UserControl**是用来设置插件主界面的UserControl在Home页面的绝对位置。

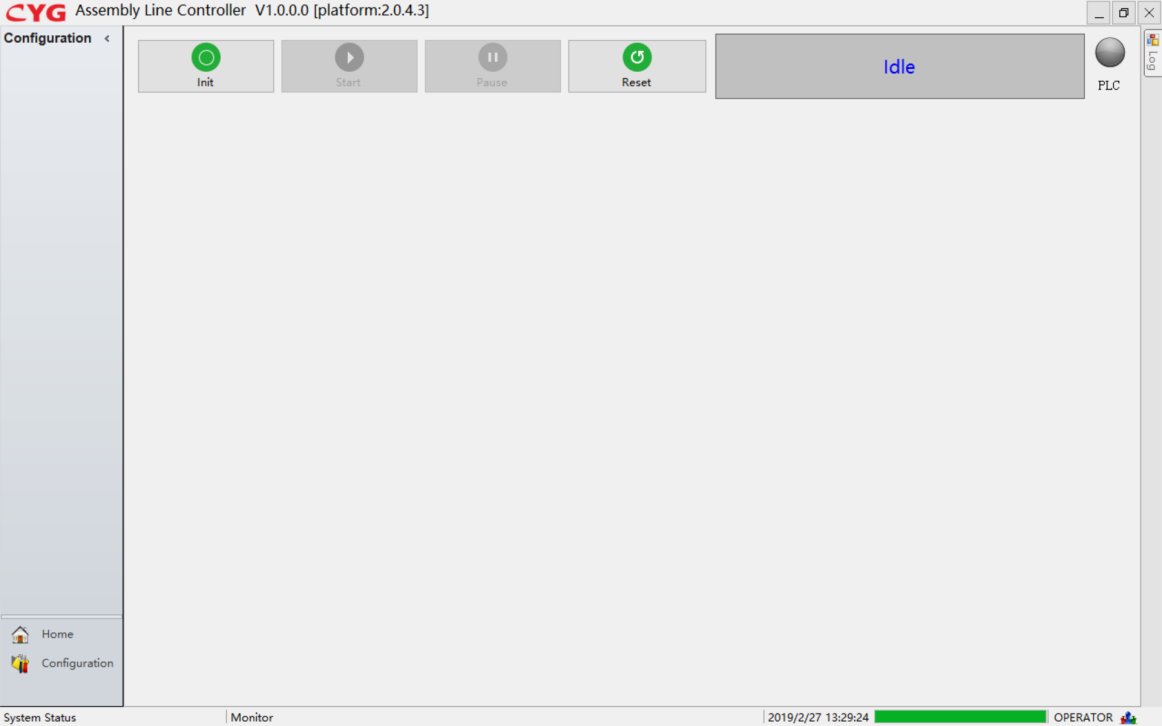
**ConfigView**是用来设置插件配置页面在ALC配置页面的绝对位置。

## 小节

现在我们将DemoProject配置的Plugins修改如下，其他保持不变：



生成解决方案后调试程序，进入主界面如下：



我们看到右上角出现了名为PLC的信号灯，说明插件已经成功加载。

# 插件编写说明

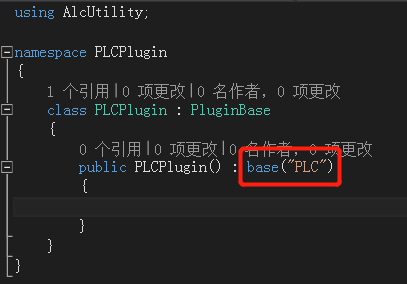
对于整个插件的开发，主要是通过插件类重写本节介绍的所有的方法来实现的。其中有些方法可能不是每个项目中都会用到，用不到的什么也不用做就好了。下面具体介绍每一个成员，和方法。

需要注意的一点：每一个DLL只能有一个类继承PluginBase作为插件类。多个插件需要创建多个DLL项目。

## 成员

### \_moduleType

插件所对应的module type，就是在构造函数传给基类的参数，如下图的“PLC”：



### SendMessageToNetwork

Action类型，带一个参数为ReceiveData类型。该action用来给网口发送消息。发送消息的内容为ReceiveData里面的。

SendMessageToNetwork(new ReceivedData() { 。。。});//其中“。。。”表示根据具体的消息内容来实现。

### BroadcastMessageToPlugins

Action类型，带一个String类型的参数和一个ReceiveData类型的参数。该Action用来给ALC平台的各插件之间发送消息，不会经过网口。其中，string类型是指接收方的ModuleType。ReceiveData为发送的消息。

**注：**ModuleType是标识一个网路客户端的类型。如PLC连接到ALC，那么PLC作为一个网络客户端，在和ALC建立链接后，需要给ALC发送注册消息，注册消息中带有该网络客户端的ModuleType。详细请参看附件7.1节通讯接口文档。

需要给插件发送消息的时候，直接调用：

BroadcastMessageToPlugins("CCD", new ReceivedData() {。。。 });//其中“。。。”表示根据具体的消息内容来实现。

### \_jsonMsgHandlers

Dictionary<string, ICmdHandler>类型，其中key为message name，value为实现ICmdHandler接口的对象。在收到消息时，如果字符串类型的数据不为空，会默认调用该字典中message name对应的对象来处理。

### \_binaryMsgHandlers

Dictionary<int, ICmdHandler>类型，其中key为command id，value为实现ICmdHandler接口的对象。在收到消息时，如果二进制类型的数据不为空，会默认调用该字典中command id对应的对象来处理。

### ReceiveData数据结构说明

该数据结构为插件直接交互的消息内容。具体定义为：

public class ReceivedData

{

public string ip { get; set; }//消息发送方IP

public int port { get; set; }//消息发送方的端口号

public string moduleId { get; set; }//消息发送方的moduleId,可能有相同的moduleType, 建立不同的连接，每个连接的moduleId不一样。

public string moduleType { get; set; }//消息发送方的moduleType，用来标识一类连接客户端

public BinaryData binaryData { get; set; }//二进制数据，为网络上传输的消息的实际内容

public JsonData strData { get; set; }//Json串数据，为网络上传输的消息的实际内容

public string toString();//提供了友好的把整个消息转换成字符串的方法，可以方便的进行日志打印

}

其中，BinaryData 和JsonData具体参见附件的7.1的《通讯接口文档》。

## 方法

### MessageHandler

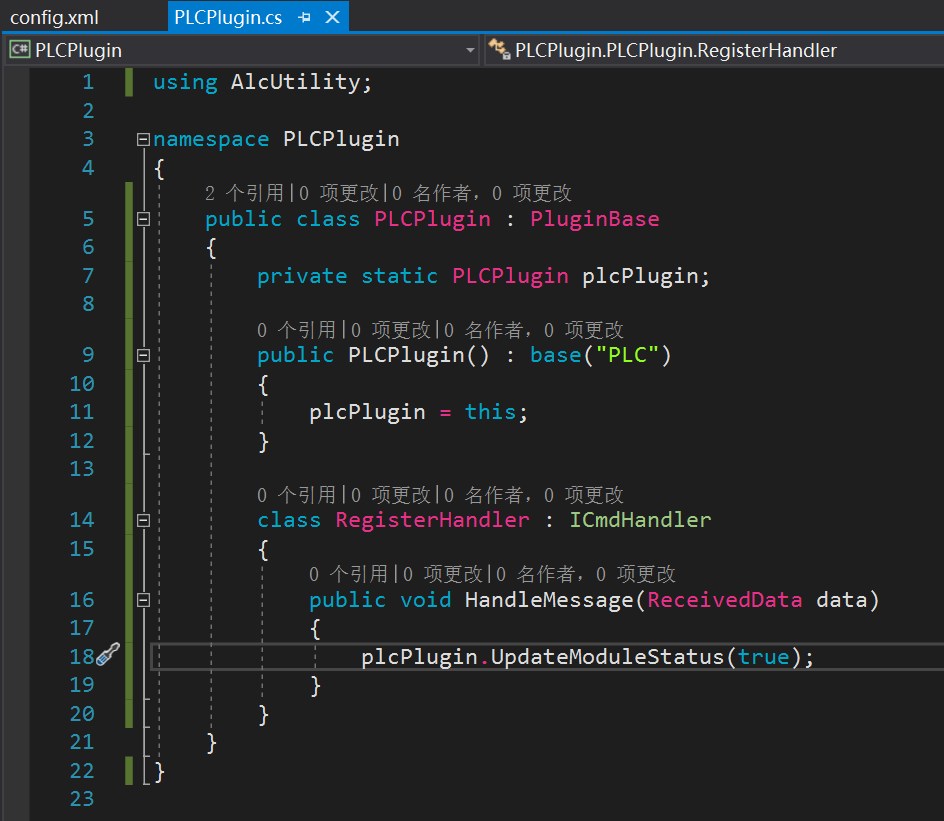
该方法用来处理从网口收到的消息和从其它插件收到的消息。具体方法为：

void MessageHandler(ReceivedData data)

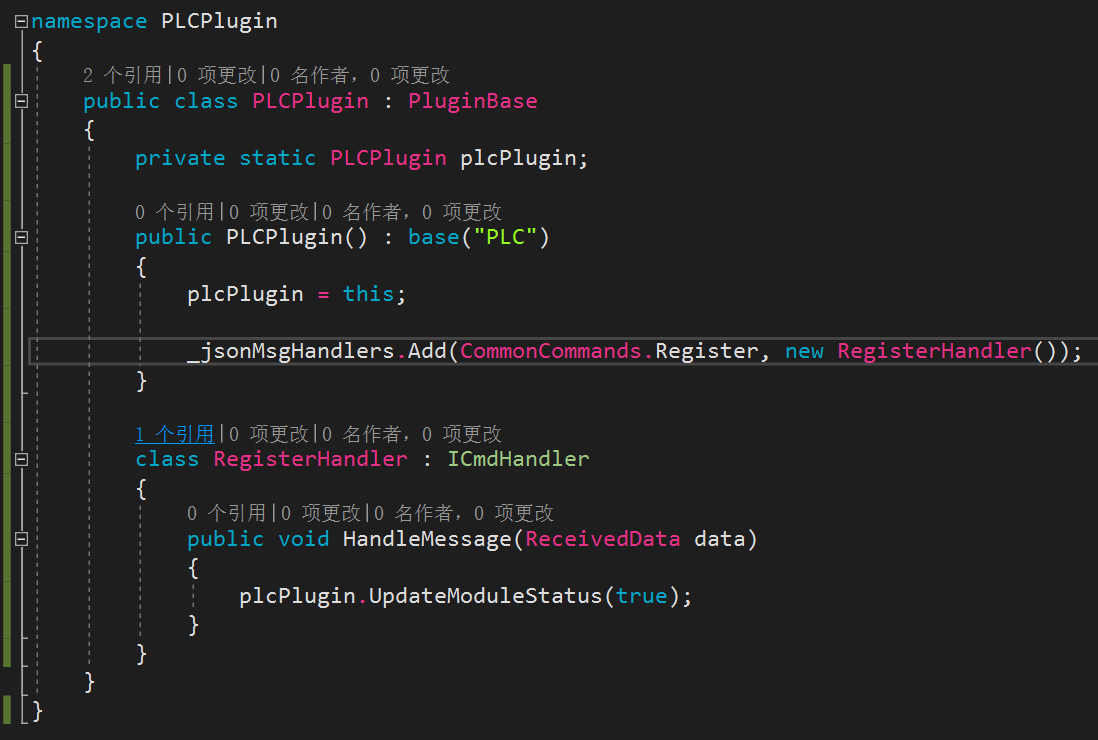
其中，参数data为收到的具体消息。在该方法中，实现对各种消息的处理。需要注意的是，所有的消息都是以异步的方式出现的，即可以理解为所有的消息都是一个独立的线程。如果一个消息的处理需要等收到另一个消息，可以通过加ResetEvent来实现同步。

如4.1.4和4.1.5所说，该方法默认会调用两个字典中的对象来处理消息，所有只需要在字典中添加消息处理对象就行了，下面为以register举个例子：

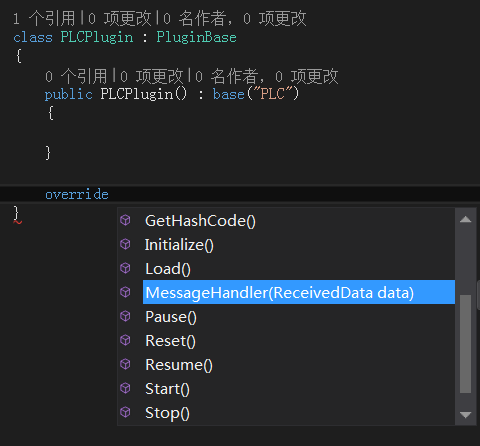
1. 创建一个消息处理类registerHandler，并实现接口ICmdHandler。为了在该类中调用插件的方法，我们在PLCPlugin中添加了静态的plcPlugin，并在构造函数中对其赋值。此例用到的UpdateModuleStatus会在后面具体说明。



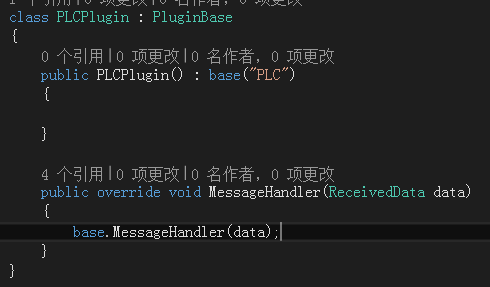
1. 在插件的构造函数中，将该类的实例添加到字典中，这样就完成了消息处理的添加。register指令我们已经在CommonCommands中定义好了，所有直接调用。如果是其他指令，则需要自行定义。



当然，如果开发人员不喜欢这样方式。想要自己的处理方式，也很简单，重写MessageHandler方法就行了。打出override，再按个空格，然后就可以选择MessageHandler：



回车后就自动补全了代码如下：



然后删掉base.MessageHandler(data);填写自己的代码就好了。

### GetControl

该方法为返回显示在ALC的HOME界面的UserControl。具体为：

UserControl GetControl()

该方法默认返回空，即没有界面。若该插件需要在主界面显示，则可以在插件项目里面，添加一个UserControl，画好界面，然后重写此方法返回即可。如：

public override UserControl GetControl()

{

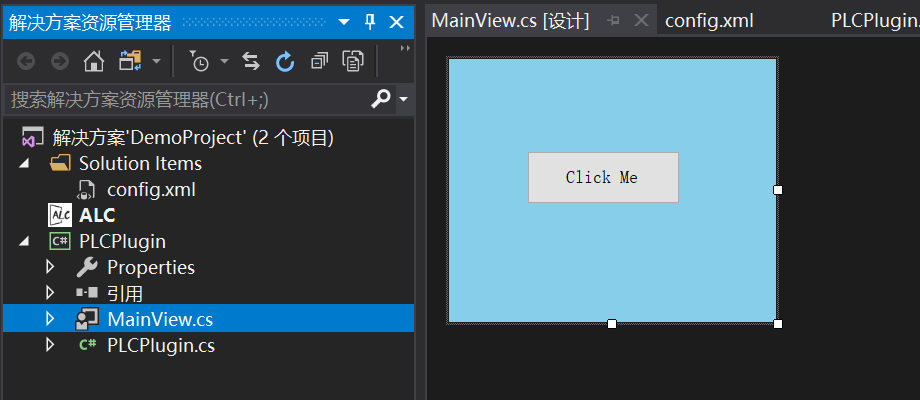
return new PLCMainDlg();

}

其中，PLCMainDlg为自定义的UserControl类。

示例：

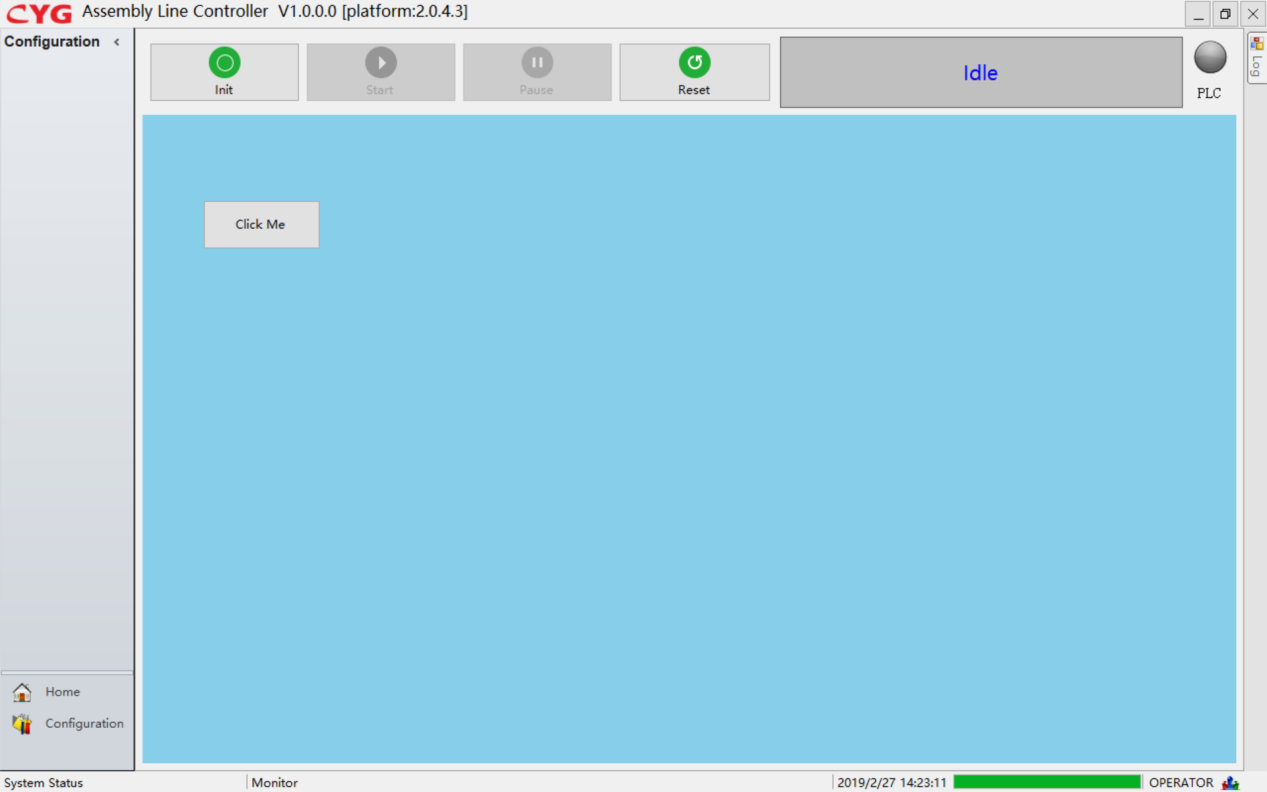
我们在DemoProject的PLCPlugin中创建一个叫做MainView的UserControl：



在GetControl中返回MainView，我们把它的Dock属性设为Fill，让它铺满整个界面：



生成解决方案后调试程序，进入主界面如下图：



### GetForm

该方法为返回插件专属的Form。会在ALC的导航栏添加一项名为该插件ModuleType的菜单，在ALC平台上点击该菜单会切换到该接口返回的Form上。完整定义如下：

Form GetForm()

该方法默认返回空，即没有专属页面。若需要有插件自己的专属页面，可以在项目里添加一个WinForm，画好界面后，然后重写此方法返回即可。如：

public override Form GetForm()

{

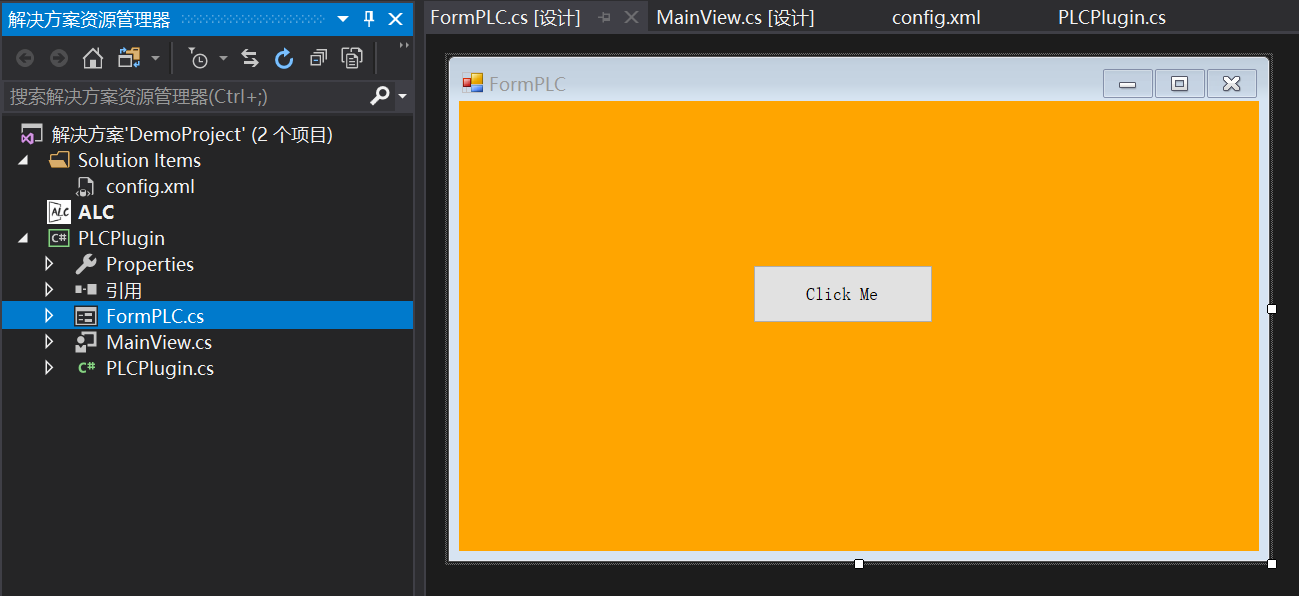
return new PLCForm();

}

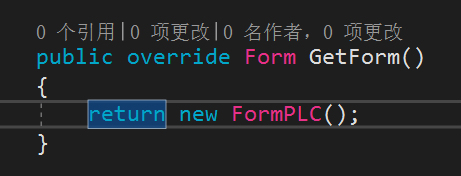
其中，PLCForm为自定义的WinForm。

示例：

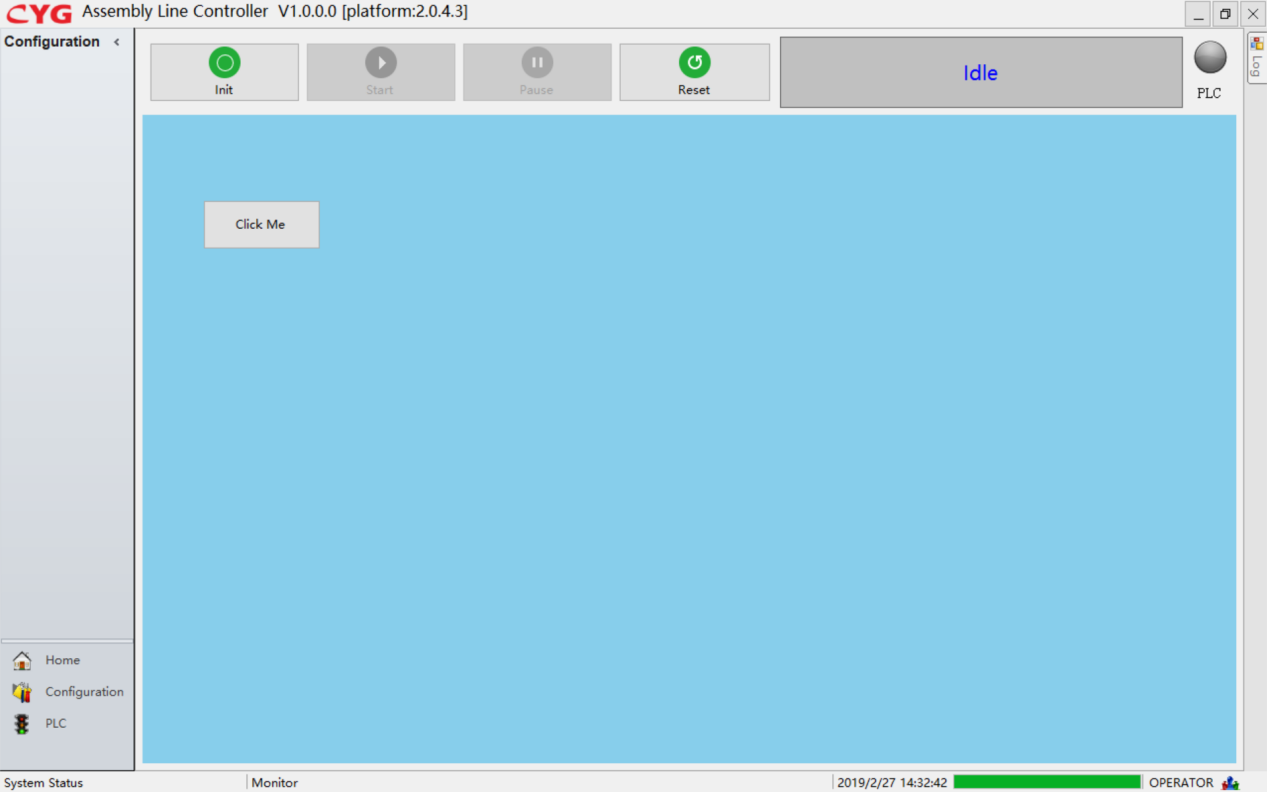
我们在DemoProject的PLCPlugin中创建一个叫做FormPLC的Form：



在GetForm中返回此Form：



生成解决方案后调试程序，进入主界面如下图：



左下角导航栏中多了一项“PLC”，点击“PLC”如下图：



### GetConfigView

该方法可以让插件有自己的配置页面，并把这个配置页面显示到平台的配置页面内。完整定义为：

System.Windows.Forms.UserControl GetConfigView()

该方法默认返回空，即没有配置页面。若需要配置，可以重写该方法返回自定义的UserControl就可以了。如：

public override UserControl GetConfigView()

{

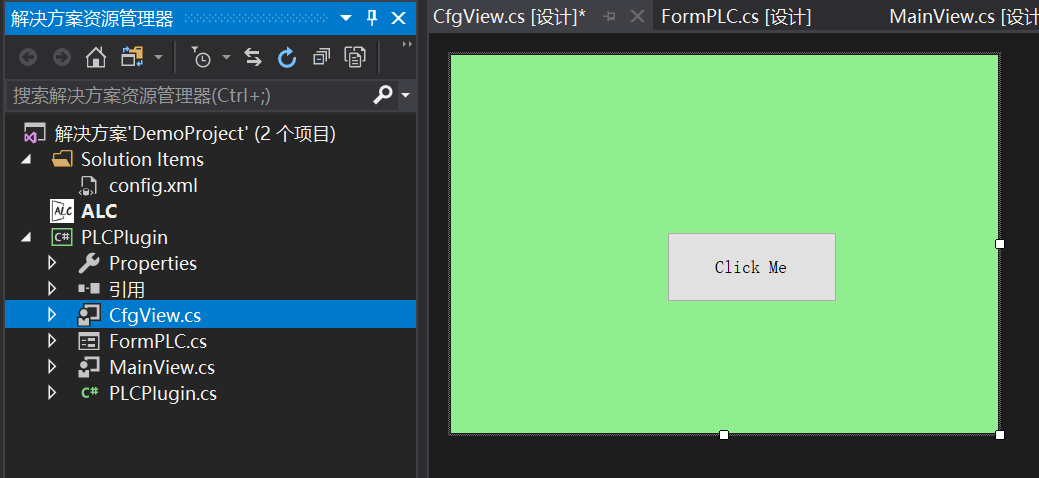
return new PLCCfgDlg();

}

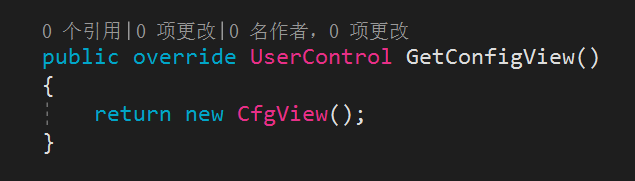
其中，PLCCfgDlg为自定义的UserControl类。

示例：

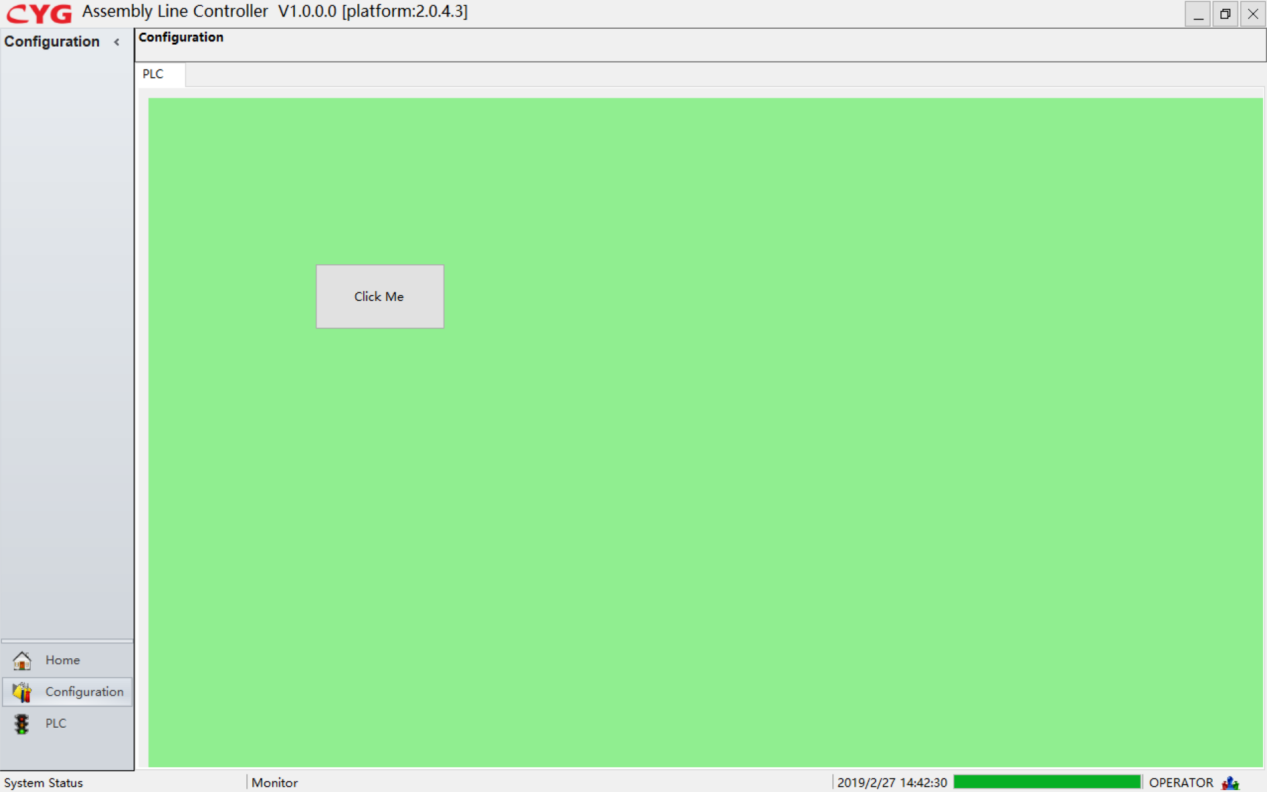
我们在DemoProject的PLCPlugin中创建一个叫做CfgView的UserControl：



在GetConfigView中返回此UserControl：



生成解决方案后调试程序，进入主界面并选择左下角导航栏中的Configuration：



### UpdateModuleStatus

该方法用来更新模块的状态，当插件模块对应的其他程序注册到了ALC就设为True，断开了就设为False。在ALC主界面右上角有每个模块状态信号灯，默认为灰色，用该方法将状态更新为true，这会显示为绿色。更新为false，则显示为红色。主界面的的Init、Start等几个按钮必须在所有信号灯都被点亮的情况下才能起作用，具体如下：

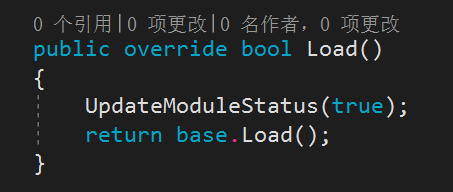
public void UpdateModuleStatus(bool alive);

### Load

该方法在插件加载的时候，会被ALC平台调用。可以粗暴地理解为当程序启动的时候会执行该接口。因此可以重写该方法用来做一些需要在整线跑起来之前做的工作，比如说一些资源的申请、初始化等操作。特别是用C#封装的C++接口，在启动的时候一般都需要申请资源。

示例：

在PLCPlugin的Load中更新模块的状态：



生成解决方案后调试程序，进入主界面，PLC的状态等变为了绿色，如下图：



### Initialize

该方法用来实现整条自动化线的初始化动作逻辑。例如在PLC中一般会复位一些IO，检查一些变量状态，机械回原位等。即在ALC界面点击“Init”按钮的时候，会调用所有插件的该方法，使整条线初始化到可以正式运行前需要的状态。初始化成功返回true，失败返回false。

### Start

该方法所有的项目都会有，是整线开始运行。即在ALC界面点击“Start”按钮的时候，会调用所有插件的start方法，从而实现整条线体的启动。启动成功返回true，失败返回false。

### Stop

该方法所有的项目都会有，是整线停止运行。即在ALC界面点击“Stop”按钮的时候，会调用所有插件的stop方法，从而实现整条线体的停止运行。停止成功返回true，失败返回false。

### Pause

该方法为整线暂停运行，并且随时可恢复运行。即在ALC界面点击“Pause”按钮的时候，会调用所有插件的pause方法，从而实现整条线体的暂停运行。暂停成功返回true，失败返回false。

### Resume

该方法为在线体暂停以后，恢复运行。即在ALC界面点击“Resume”按钮的时候，会调用所有插件的resume方法，从而从暂停中恢复整条线体的运行。恢复成功返回true，失败返回false。

### Reset

该方法为控制整条线体的复位。即在ALC界面点击“Reset”按钮的时候，会调用所有插件的reset方法，从而实现整条线体的复位。复位成功返回true，失败返回false。

### Dispose

该方法是在插件卸载或者整个ALC程序关闭的时候调用，主要用来做一些资源的清理释放。特别是当插件中调用有C++代码的时候，需要手工释放资源，可以放在这个接口中完成。

### Log

前面的方法都是需要重写的方法，从这个方法开始都是不用重写，提供给插件使用的方法。

该方法用来打印日志到对应模块的日志文件和日志窗口，具体为：

public void Log(string text, AlcErrorLevel errLevel = AlcErrorLevel.TRACE);

text：日志内容。必填。

errLevel：错误等级，总共有5个level，分别为DEBUG,TRACE,WARN,ERROR,FATAL。每个level会将日志存到对应的日志文件中。默认为TRACE，可以不用填写。

### ShowMsgBox

该方法用来显示弹窗，具体如下：

public AlcMsgBoxResult ShowMsgBox(string text, string caption, AlcMsgBoxButtons buttons = AlcMsgBoxButtons.OK, AlcMsgBoxIcon icon = AlcMsgBoxIcon.Information, AlcMsgBoxDefaultButton defaultButton = AlcMsgBoxDefaultButton.Button1);

### SaveCsv

该方法用来保存CSV文件，具体如下：

public void SaveCsv(string csvName, string head, string text);

### Error

该方法用来向平台上报Error，平台有统一的Error handle。具体如下：

public AlcMsgBoxResult Error(string errorMsg, int errorCode, AlcErrorLevel errorLevel);

其中errorLevel分为5个等级：

1. DEBUG：将错误存入DEBUG日志中，无其他处理。
2. TRACE：将错误存入TRACE日志中，无其他处理。
3. WARN：将错误存入WARN日志中，并弹框提醒，不影响流程。
4. ERROR：将错误存入ERROR日志中，并弹框报错，提供abort，retry，ignor三个按钮，选择abort则触发abort流程。操作员选择的结果会通过方法返回。
5. FATAL：将错误存入FATAL日志中，并直接触发abort流程，弹框报错。

### GetSystemStatus

该方法用来获取系统的状态，即主界面显示的状态，有idle，ready等，具体如下：

public SYSTEM\_STATUS GetSystemStatus();

### GetUserAuthority

该方法用来获取当前的用户权限，默认有ADMINISTRATOR，ENGINEER，OPERATOR三种权限，具体如下：

public String GetUserAuthority();

### ButtonClickRequire

该方法用来通过代码触发init，start等流程。具体如下：

public void ButtonClickRequire(ButtonsName button);

# ALCSystem类

上一节，我们介绍了PluginBase中的功能，那些功能只能在继承了PluginBase的类中才能调用，而且每一个DLL只能有一个类继承PluginBase作为插件类。

这节介绍的ALCSystem类这是独立于PluginBase存在的，直接获取其实例即可使用。

## 事件

ALCSystem中提供了一些系统的事件，在对应的事件发生时便会触发。事件的触发是与主程序同步的，即意味着会阻塞主程序，所以，如果希望事件响应程序是异步的需要在响应程序中进行异步处理。

### UserAuthorityChanged

用户权限发生改变时被触发。具体定义如下：

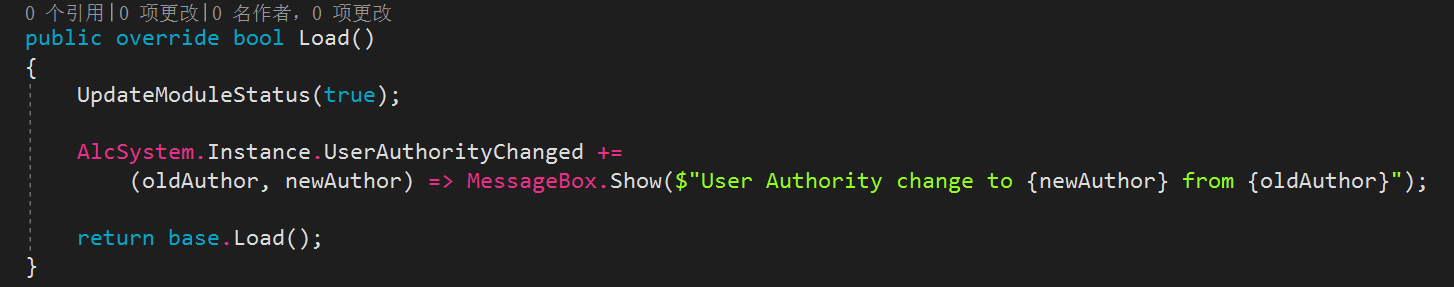


第一个参数为 改变前的用户权限

第二个参数为 改变后的用户权限

示例：

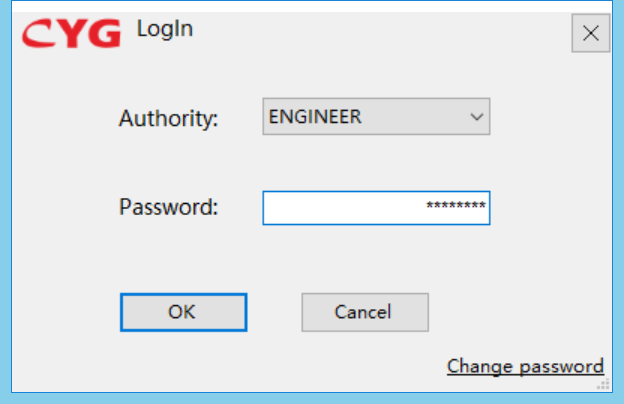
在PLCPlugin的Load中添加响应函数：



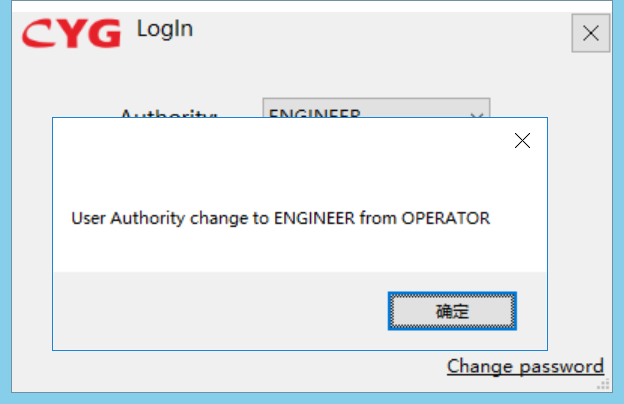
生成解决方案，调试程序。点击主界面右下角的用户：



弹出登录窗口切换到其他权限，默认密码与权限名相同：



点击OK后，弹出了MessageBox：



### FormShown

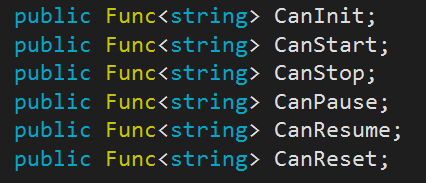
主界面左下角的导航栏被点击时触发。具体定义如下：



参数为导航栏选项的名字，其中Home和Configure的名字和显示的不同，分别为band\_Home和band\_Config。

### CanInit、CanStart、CanStop、CanPause、CanResume、CanReset

在执行Init、Start、Stop、Pause、Resume、Reset流程之前触发，并根据返回的字符串决定是否可以执行响应的流程。具体定义如下：



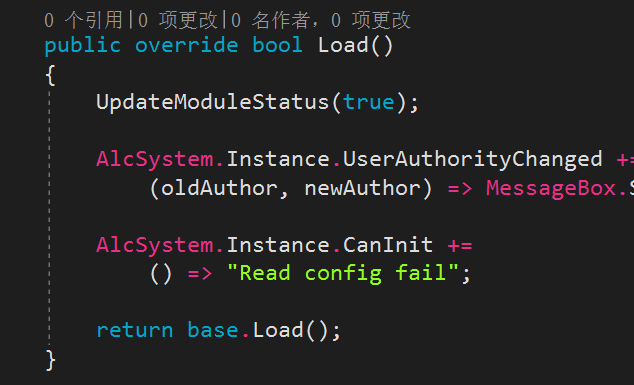
没有参数，只有一个string类型的返回值

返回空字符串或Null，则说明可以执行该流程；

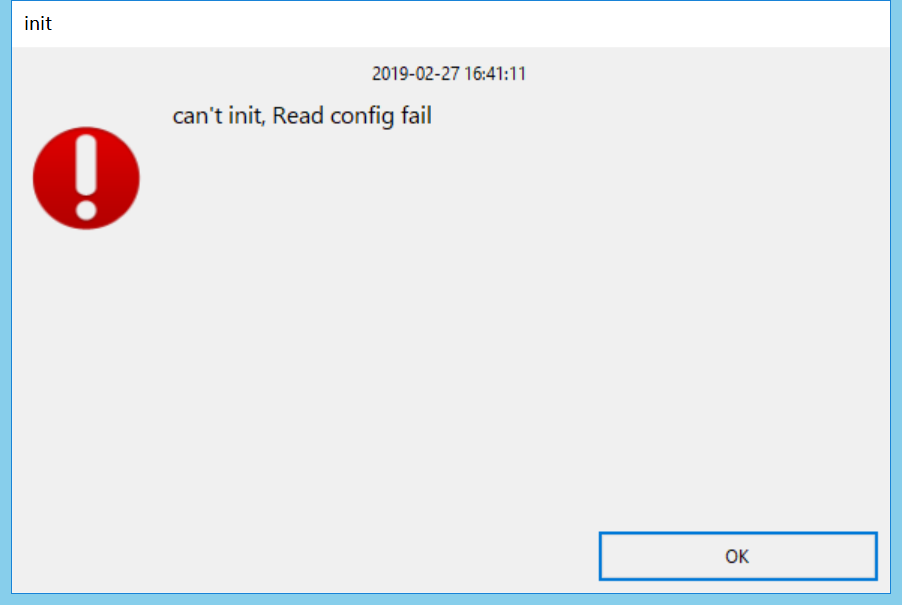
若返回不为空字符串和Null，这说明不可执行该流程，并在返回的字符串中说明原因。

示例：

在PLCPlugin的Load中添加响应函数：



生成解决方案，调试程序。点击主界面的Init按钮，弹框提示无法Init：



## 方法

### 插件中已介绍的方法

下面为在插件中已经介绍过的方法，在ALCSystem中也可以调用。

* ButtonClickRequire
* Error
* GetSystemStatus
* GetUserAuthority
* Log
* SaveCsv
* ShowMsgBox
* UpdateModuleStatus

### GetSoftwareVersion

获取软件的版本号，即在第三节中讲的配置文件中配置的版本。具体定义如下：



# 提供的功能

## 数据库操作

我们提供了一个NativeDatabaseHelper类来进行关于MySQL数据库的相关操作。

提供了一个构造函数：

public NativeDatabaseHelper(string conn\_str)；

参数是数据库的链接字符串。当new一个NativeDatabaseHelper的对象的时候，会根据这个连接字符串自动连上MySQL数据库。

然后提供了3个执行sql语句的方法，分别为：

public DataTable ExecuteDataTable(string sql, Dictionary<string, object> arges)

输入参数sql是sql语句，args是sql语句执行的参数，返回值是sql语句执行的结果以DataTable的结构返回。

public void ExecuteAction(string sql, Dictionary<string, object> arges,Action<SqlDataReader> Ac)

输入参数sql是sql语句，args是sql语句执行的参数，第三个参数是Action类型，是sql语句执行的结果，用DataReader的方式返回，每一行结果调用一次Ac

public int executeNoResult(string sql, Dictionary<string, object> arges)

这个方法用来执行非select的sql语句，如insert、delete等。返回值表示执行完该sql语句影响的行数。输入参数sql是sql语句，args是sql语句执行的参数。

## SSH远程拉起

我们提供了一个可以从ALC通过SSH远程执行shell脚本的类SshRun。构造函数为：

public SshRun(string ip, string username, string password)

ip为远程电脑的IP， username为远程电脑的登录用户名，password为远程电脑的登录密码。

创建完SshRun对象了以后，可以调用以下方法在远程电脑上执行Shell脚本语句。

public int RunSsh(string shellScript)

输入参数shellScript是在远程电脑上执行的shell语句。只要用户权限高，基本上可以做任何事情。

返回值为-9999的话，说明远程执行脚本语句出错。其它为执行语句返回的结果。

## TCP客户端

我们同样提供了两种TCP客户端：通过回车换行作为结束符的字符串客户端和定长的二进制客户端。其中二进制客户端的定长需要在config文件中配置。

### string客户端

我们提供了一个StringClients类来提供字符串客户端的功能。构造函数如下：

public StringClients(int localPort, Action<string> msgHandle)

输入参数localPort是本地端口，msgHandle是处理收到string消息的事件。当收到消息的时候，会触发这个事件。

注意：收到的消息必须已回车换行作为结束符。

当创建完该类的对象后，可以调用connect函数去连接远程服务器。方法如下：

public bool Connect(string remoteIP, int remoteport)

输入参数为所要连接的远程服务器的IP和端口号。

当需要给服务器发送消息的时候，可以调用SendMessage方法：

public bool sendMessage(string msg)

输入参数为要发送的消息内容。返回true表示发送成功，false表示发送失败。

当需要断开当前链接的时候，可以调用close方法：

public void close()

### binary客户端

我们提供了一个BinaryClients的类来提供二进制客户端功能。构造函数为：

public BinaryClients(int localPort, Action<byte[]> msgHandle, int FixedMsgLen)

前两个参数和StringClients一样，最后一个参数为二进制消息定长。即接收到的每一条消息都必须是这么长。

和StringClients一样，同样提供了Connect、sendMessage、close方法。用法也是一样的，这里就不再介绍了。

# 附件

## 通讯接口文档

<ALC平台接口定义.docx>

## 实例代码