智能化集成系统IO采集服务器

二次开发手册

南京古河软件有限公司

版本信息

|  |  |  |  |
| --- | --- | --- | --- |
| 文档名称 | 智能化集成系统IO采集服务器V1.1二次开发手册 | | |
| 文档类别 | 系统开发手册 | 文档版本 | V1.0 |
| 审批人 |  | 编制人 | 董入学 |
| 保密级别 | 保密 | 日 期 | 2017/4/8 |
| 备注： |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 前言

IO采集服务器是南京古河软件有限公司自主开发的一套多协议数据通讯网关软件，简称IOServer，IOServer可以支持将Modbus、BACnet、OPC、SNMP等多种工业通用协议或者自定义私有协议转换为本公司云平台GH-CLOUD的通讯协议，利用IOServer这套网关软件可以实现智能化设备快速接入本公司的GH-CLOUD云平台，与云平台双向通信。

IO采集服务器采用插件架构的灵活编程方式，系统可以分为应用程序框架、通用类库、通讯插件三个部分。插件结构的灵活性保证了软件方便的定制性。 同时由于插件架构的灵活性，意味着主程序和通讯插件之间可以进行很自由的组合。每个通讯插件都是可以通过注册授权的方式进行加载，便于支持灵活的部署和产品销售。

IO采集服务器是基于微软.NET 4.5开发，推荐运行了Windows平台上，可以部署在通常的Window Server2008、Window Server2012、Win7、Win10的服务器和PC机上，对于需要前端部署的推荐采用Intel NUC微型电脑。

IO采集服务器的主要功能是利用通讯插件与智能设备通讯，支持的通讯方式可以是RS232、RS485、TCP/IP，利用各设备厂商提供的协议栈获取设备的相关数据，并利过云平台的API，将相关信息上传到云端的实时数据库内，同时定时和云的MQS消息服务器通讯，收取云端下发的控制命令，转发至设备。此外IOServer还可以设置数据的报警阙值，产生报警后将相关报警上传到云平台。

# 开发环境

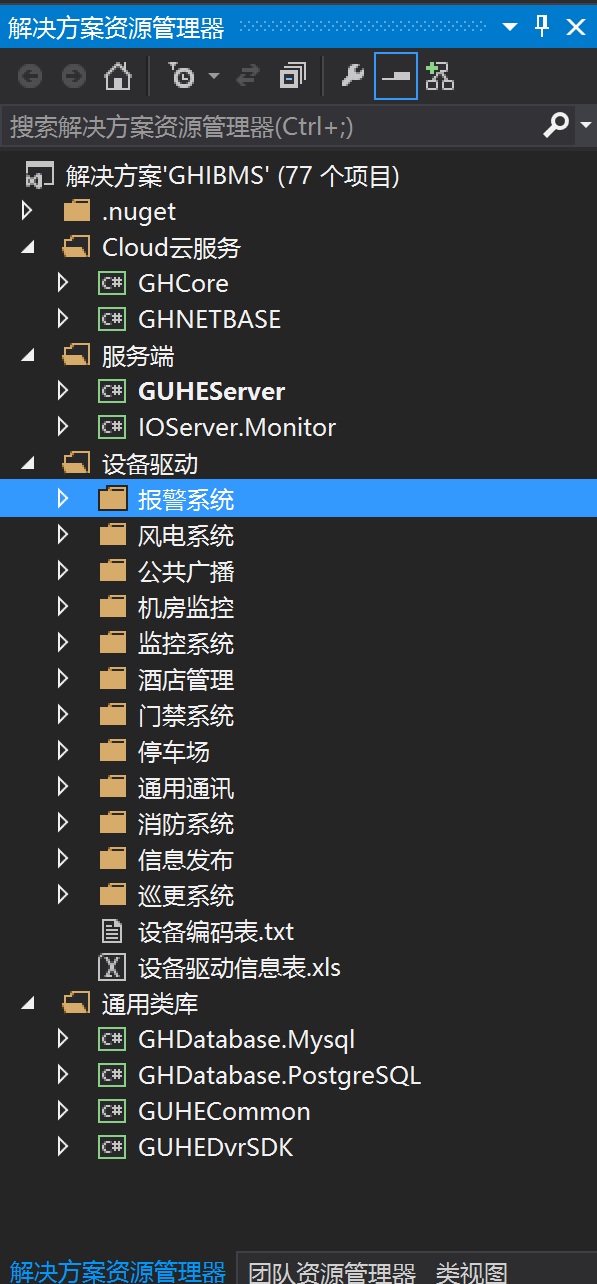
开发工具：Microsoft Visual Studio 2013

开发语言：C#

操作系统：Windows 10

开发框架：Microsoft .NET Framework 4.0

# 项目文件目录



IOServer解决方案内项目列表及简要说明

## Cloud云服务

* GHCore

与云平台通讯的基础类库，比如JSON序列化、Binary序列化，LOG记录等

* GHNETBASE

与云平台通讯的接口，包括上传数据到云端实时库，上传报警到云端实时库、自云端MQ接收消息等

## IOServer主程序

* GUHEServer

应用程序入口，包括文件管理、通道管理、控制器管理、变量管理等。

* IOServer.Monitor

主程序的守护进程。主程序非法关闭时，自动启动主程序。

## 设备驱动

|  |  |  |
| --- | --- | --- |
| 报警系统 | Communication.AL7480E.dll | AL7480报警主机通讯接口 |
| Communication.CTIP2000.dll | CT-IP2000报警接口 |
| Communication.DS7400.dll | DS7400报警接口 |
| Communication.GatoA5Alarm.dll | 广拓报警模块通讯接口 |
| Communication.HKAlarm.dll | 海康报警主机通讯接口 |
| Communication.HKMTAAlarm.dll | 海康MTA报警模块通讯接口 |
| Communication.INANTER.dll | 英安特报警通讯接口 |
| Communication.IP7400.dll | IP7400通讯接口 |
| Communication.IPM.dll | Honywell IPM报警接口 |
| Communication.LongHorn.dll | LongHorn报警接口 |
| 公共广播 | Communication.IPAppPA.dll | IP广播接口(艾普) |
| Communication.IPCast3A.dll | IP广播接口(3A、ITC) |
| Communication.TKoko.dll | TKOKO IP广播接口 |
| 机房监控 | Communication.ALP2000.dll | 艾勒普机房动环通讯接口 |
| Communication.LNSUPS.dll | 雷诺士UPS接口 |
| Communication.Xbrother.dll | 深圳共济环控通讯接口 |
| 客房管理 | Communication.HDL.dll | HDL酒店通讯接口 |
| Communication.KKAMClient.dll | AM酒店客控通讯接口 |
| 监控系统 | Communication.Camera.dll | 摄像机通讯接口 |
| Communication.DaHuaPlat.dll | 大华视频平台通讯接口 |
| Communication.DHDvr.dll | 大华DVR通讯接口 |
| Communication.FCNetDvr.dll | 南宁园博园DVR通讯接口 |
| Communication.HBDvr.dll | 汉邦DVR/NVR/DVS通讯接口 |
| Communication.HIKPlat.dll | 海康视频平台通讯接口 |
| Communication.HKDvr.dll | 海康DVR/NVR/DVS通讯接口 |
| Communication.HKVod.dll | 海康流媒体通讯接口 |
| Communication.IMOS.dll | 宇视视频平台通讯接口 |
| Communication.INFDvr.dll | 英飞拓DVR/NVR/DVS通讯接口 |
| Communication.INFIPC.dll | 英飞拓IPC通讯接口 |
| Communication.LCDScreen.dll | LCD屏接口 |
| Communication.Matrix.dll | 视频矩阵接口 |
| Communication.Monitor.dll | 监视器通讯接口 |
| Communication.TandiDvr.dll | 天地伟业DVR/NVR/DVS通讯接口 |
| Communication.VIKOR.dll | 华安泰视频平台通讯接口 |
| Communication.SMTK.dll | 智安邦智能视频通讯接口 |
| 门禁系统 | Communication.Poris.dll | PORIS门禁门禁接口 |
| Communication.WEDSAcess.dll | 威尔门禁门禁接口 |
| Communication.WGAcess.dll | WG门禁接口 |
| 停车系统 | Communication.HIKPark.dll | 海康停车场通讯接口 |
| Communication.PorisPark.dll | PorisPark通讯接口 |
| Communication.JSSTG3Park.dll | 捷顺一卡通通讯接口 |
| 通用系统 | Communication.TcpTansmit.dll | TCP转发通讯接口 |
| Communication.BACnet.dll | BacNet接口 |
| Communication.Modbus.dll | Modbus通讯接口 |
| Communication.opc.dll | OPC接口 |
| Communication.Ping.dll | 网络测试通讯接口 |
| Communication.SNMP.dll | 简单网管协议接口 |
| Communication.PSTN.dll | 电话模块通讯接口 |
| Communication.SMS.dll | 短信模块通讯接口 |
| Communication.SuperVisor.dll | 系统监视通讯接口 |
| Communication.OBIX.dll | oBIX通讯接口 |
| Communication.Virtual.dll | 虚拟通讯接口 |
| 消防系统 | Communication.BeidaJBUFA.dll | 北大青鸟消防接口 |
| Communication.LDFA.dll | 利达消防接口 |
| Communication.NotifierAM2020.dll | NotifierAM2020消防接口 |
| Communication.ShanyingFire.dll | 营口新山鹰消防接口 |
| Communication.TAFA.dll | 泰和安消防接口 |
| 信息发布 | Communication.DBClear.dll | 清鹤信息发布通讯接口 |
| Communication.SCL2008.dll | Led屏[SCL2008控制卡]通讯接口 |
| Communication.ShowInfo.dll | 视尔信息发布REST\_API接口 |
| 巡更系统 | Communication.Landwell.dll | Landwell巡更通讯接口 |

## 通用类库

* GHDatabase.Mysql

Mysql数据库操作类库

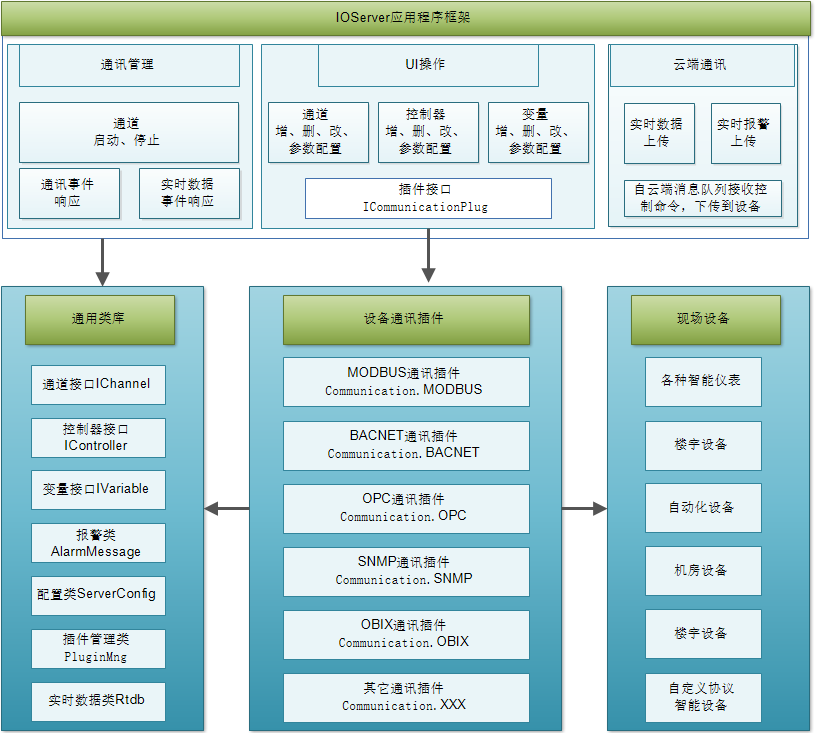
* GHDatabase.PostgreSQL

PostgreSQL数据库操作类库

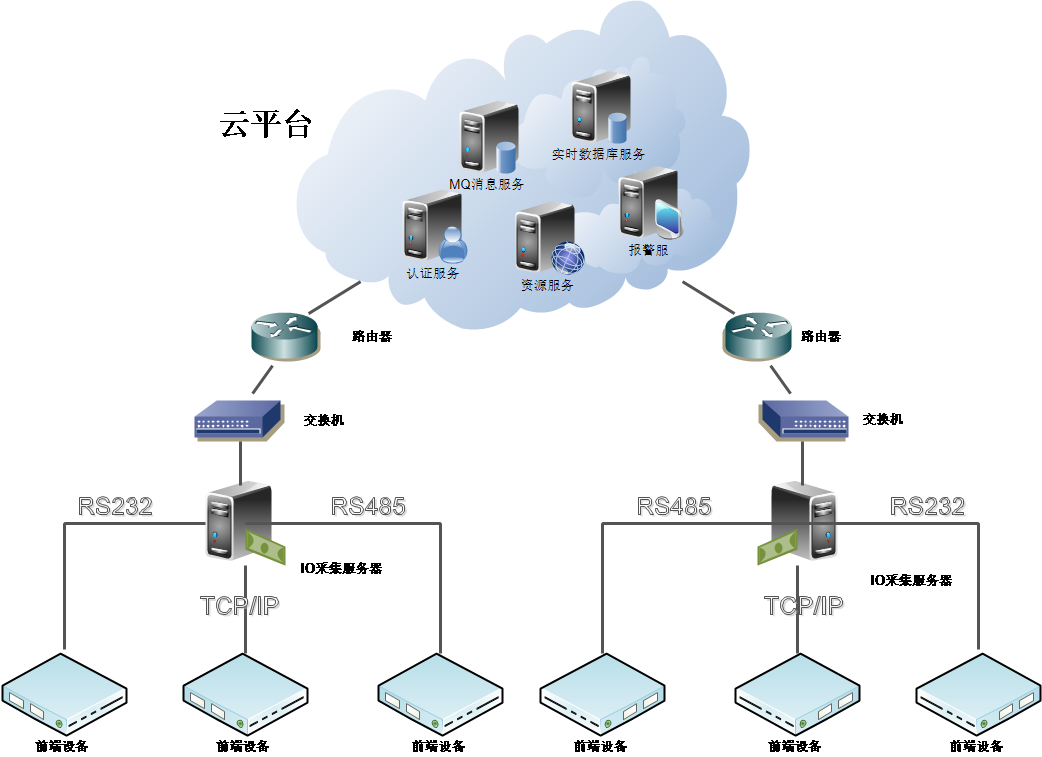
* GUHECommon

项目自定义的通用类库

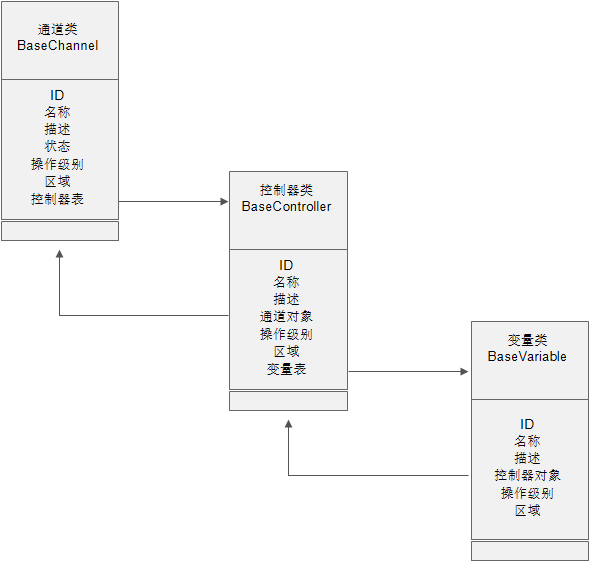
# 软件架构图



# 网络结构图



# 设备模型图



通道类、控制器类、变量类这三个基本类用于将智能化系统中常用的设备功能软件模型化，通道类用于描述通讯相关，控制器类用于描述设备控制器相关，变量用于对应设备内的功能点。

通道类、控制器类、变量类通过属性关联可以实现双向寻址。

# 关键接口简介

## ICommunicationPlug

文件的位置：通用类库\GUHECommon\Interface\ICommunicationPlug.cs

每一个通讯驱动插件都必须实现这个接口，主程序通过这个接口判断一个程序集是不是一个自定义的通讯驱动插件。因此开发一个插件，首先是实现这个借口。

这个接口定义了一个通讯驱动的类的基本属性，包括：

* + 插件名称

String Name

* + 插件ID

String PlugID

每个插件都必须有一个唯一的ID

* + 当前选中的协议

string SelectedProtocol

多协议插件，在创建一个通道时，当前选中的协议名称

* + 支持的协议

List<IProtocol> ProtocolList

插件支持的协议列表，比如MODBUS TCP,MODBUS RTU

* + 系统分类

string SystemLabel

用于插件管理时分类

* + 获取插件的协议码

IProtocol GetProtocol(string code);

多协议支持时，通过协议码，获取协议对象

* + 获取插件支持的通讯接口名称

string[] GetCommInterface();

比如网络、串口、API

* + 获取插件中设备命名前缀

string GetUniqueName(UniqueNameType type);

实现不同的插件通道、控制器、变量采用个性化的前缀

* + 判断插件内是否支持该协议

bool IsSupportProtocol(string code);

* + 新建一个通道

IChannel CreateChannel(string code,string type);

用于新建一个通道

* + 从XML中反序列化一个通道

IChannel CreateChannel(XmlNode node)

用于通道的载入

* + 新建一个控制器

IController CreateController(IChannel channel)

* + 新建一个变量

IVariable CreateVariable(IController controller)

* + 保存通道为XML

XmlElement SaveChannel(XmlDocument doc,IChannel cha)

* + 导入控制器

List<IController> ImportController(IChannel channel)

用于服务端通道的快捷菜单，批量导入控制器

* + 导入变量

bool ImportVariable(IController controller)

用于服务端控制器的快捷菜单，批量导入变量

## IChannel

文件的位置：通用类库\GUHECommon\Interface\IChannel.cs

每一个通讯驱动插件都必须实现这个接口，这个接口是描述设备驱动通讯插件采用何种方式与设备进行通信以及通信过程需要配置哪些基本参数，应用程序框架就是通过这个接口，启动和停止通讯的。

接口的基本内容如下：

* + 通讯事件

event CommMsgDelegate OnCommMsg

定义了一个事件，用于上报通信过程中的事件，比如网络连接中断，网络恢复，控制器离线，通信过程中的数据收发记录，总之通过这个事件，插件中可以上报信息，通知应用程序框架，应用启动后会遍历所有通道，注册这个事件，在框架中定义了常用事件的处理方式，主要有改变通道、控制器图标状态，向通过记录Listview中写入记录等。

* + ID,唯一

string ID

通道ID，是通道的唯一ID，用于后面的对通道的操作，一但生成后不可以修改，目前依据名称自动生成。

* + 通道名称

string Name

用于标识通道，也是唯一的，但是可以修改

* + 通道描述

string Description

对通道的说明性描述

* + 通道区域

string Area

可以设置一个分区，用于对应用户权限中的区域

* + 通道类型

string SystemLabel

用于说明通道对应的子系统分类，目前没使用

* + 启用与禁用使能

bool Enable

用于设置通道是否使用，True:启用 False:禁用

* + 通道的当前状态

bool Active

True:激活状态 False:非激活状态

* + 通讯接口

string CommType

用于说明通讯接口方式网络、串口或其它

* + 通讯时间戳

DateTime DateStamp

最后一次通讯成功的时间

* + 通讯间隔

int Interval

采用轮询方式进行和设备通讯，两个周期之前的间隔

* + 持续运行时间

uint RunTime

通道启动后的持续运行时间，用于通道注册，未注册通道只能运行4个小时

* + 控制器组

List<IController> ConList

这是一个关键的属性，用于保存通道内的所有控制器，可通过遍历这个属性，获取控制器。

* + 启动通道通讯

void Start()

用于应用程序启动通道进行通讯。

* + 停止通道通讯

void Stop();

用于应用程序停止通道进行通讯。

* + 向设备写值

bool WriteValue(IVariable var, object value)

外部程序通过这个方法向设备写值，每个设备驱动通过实现这个接口，接收外部指令，并发送到设备。

* + 向UI界面发送通讯过程中的事件

void SendCommEvent(Severity severity ,CommunicationEvent commMsgType, string wParamm, string lParamm)

OnCommMsg事件的发送方法

* + 通道克隆

IChannel Clone()

用于通道复制

* + 执行外部命令

bool ExecCommand(IController con, int cmdCode, params object[] CmdParams)

用于响应自定义的外部部分，目前没用

* + 向设备发送字符串

void Send(string cmd)

实际的通讯实现，利用API其它方式不在此实现，可以插件中定义私有方法。

* + 向设备发送字节组

void Send(byte[] cmd)

实际的通讯实现，利用API其它方式不在此实现，可以插件中定义私有方法。

## IController

文件的位置：通用类库\GUHECommon\Interface\IController.cs

每一个通讯驱动插件都必须实现这个接口，控制器对应于通道上的设备，对于OPC协议控制器对应于协议中的一个组，对于虚拟通道，对应于一个虚拟设备。

* + ID,唯一

string ID

通道ID，是控制器的唯一ID，用于后面的对控制器的操作，一但生成后不可以修改，目前依据名称自动生成。

* + 名称

string Name

用于标识控制器，也是唯一的，但是可以修改

* + 描述

string Description

对控制器的说明性描述

* + 区域

string Area

可以设置一个分区，用于对应用户权限中的区域

* + 类型

string SystemLabel

用于说明控制器对应的子系统分类，目前没使用

* + 启用与禁用使能

bool Enable

用于设置控制器是否使用，True:启用 False:禁用

* + 当前状态

bool Active

True:激活状态 False:非激活状态

* + 控制器通讯地址

string Address

* + 控制级别

int OperLevel

预留权限控制用，用户操作级别不低于该级别才可操作

* + 通讯时间戳

DateTime DateStamp

通常是最后一次通讯成功的时间

* + 控制器所属通道对象

IChannel ChannelObject

用于关联通道

* + 变量链表

List<IVariable> VarList

控制器上所有的变量

* + 对象克隆

IController Clone()

* + 执行外部控制命令

bool ExecCommand(int cmdCode, params object[] CmdParams)

控制器执行自定义的命令

## IVariable

文件的位置：通用类库\GUHECommon\Interface\IVariable.cs

每一个通讯驱动插件都必须实现这个接口，Variable变量对应于设备中的功能点，比如可以一个报警防区，一个门禁的门状态，一个空调机组的送风温度等等，对于OPC协议，变量对应于标签，对于Bacnet协议，变量对应于对象的属性。

* + 数值变量事件

event OnValueChangeDelegate OnValueChange;

来自设备更新的值和当前值不同时，触发该事件

* + 数值更新事件

event OnCounterChangeDelegate OnCounterChange;

来自设备的第一次数值更新，触发该事件

* + ID,唯一

string ID

通道ID，是控制器的唯一ID，用于后面的对控制器的操作，一但生成后不可以修改，目前依据名称自动生成。

* + 名称

string Name

用于标识控制器，也是唯一的，但是可以修改

* + 描述

string Description

对控制器的说明性描述

* + 区域

string Area

可以设置一个分区，用于对应用户权限中的区域

* + 类型

string SystemLabel

用于说明控制器对应的子系统分类，目前没使用

* + 启用与禁用使能

bool Enable

用于设置控制器是否使用，True:启用 False:禁用

* + 当前状态

bool Active

True:激活状态 False:非激活状态

* + 设备分类

string DeviceLabel

通常将变量做如下分类：

* + - 通用变量
    - 通讯报警
    - 数字输入
    - 数字输出
    - 模拟输入
    - 模拟输出
    - 报警输入
    - 报警输出

报警一键复位时，会判断是否是报警输入或通讯报警，这两种变量会被复位为缺省值

* + 只读

bool ReadOnly

设置为只读不支持外部控制

* + 缺省数值

string DefaultValue

用于自动复位和报警复位

* + 自动复位使能

bool AutoReset

* + 自动复位延时

uint AutoResetDelay

激活后过多时间复动复位（毫秒）

* + 数值格式

string Format

对应C#中的Format参数

* + 数值中小数位数

byte FloatNumb

数值类型为浮点数时生效

* + 当前数值

object Value

变量的当前值

* + 数值描述

string ValueDesc

数据的描述，比如0表示：停止，1表示启动，目前停用

* + 数值单位

string Unit

* + 数值更新计数器

UInt32 Counter

指示来自设备的数值更新次数

* + 数值类型

TypeCode ValueType

* + 时间标签

DateTime DateStamp

通常是最后一次更新数值的时间

* + 变量地址

string Address

* + 操作级别

int OperLevel

用于操作授权，用户的级别不低于该值时，才可以读写操作

* + 通讯质量

short Quality

兼容OPC,以后各个协议都应参照OPC实现

* + 关联的视图名称

string AssociateForm

老版软件报警联动用，目前停用

* + 关联的视图名称（辅助）

string AssociateForm2

老版软件报警联动用，目前停用

* + 关联的摄像机名称

string AssociateVideo

老版软件报警联动用，目前停用

* + 关联的摄像机名称（辅助）

string AssociateVideo2

老版软件报警联动用，目前停用

* + 数值写操作

bool WriteValue(object value);

通常是来自UI或联动写操作

* + 控制器

IController ControllerObject

变量所属的控制器对象

* + 优先级

Priority

操作的优先级

* + 变量在控制器列表中的序号

int VarListIndex

便于查找，更新变量

* + 历史记录器的类型

HistoryTimerRecordEnum HistoryRecorder

老版软件后台记录数据用，目前停用

* + 报警记录使能

bool AlarmRecorderEnable

老版软件后台记录数据用，目前停用

* + 变化记录使能

bool DataChangedRecorderEnable

老版软件后台记录数据用，目前停用

* + 报警信息

string AlarmMsg

用于发生报警时存储报警信息代码或说明，目前停用

* + 变量的描述链表

List<VariableDesc> DescList

设置变量数值与数值描述关系

* + 报警条件链表

List<VariableTrigger> WayList

设置报警触发条件

* + 变量的动作联动链表

List<VariableAction> ActionList

设置变量连接动作

* + 变量的视频联动链表

List<VariableVideo> VideoList

设置视频联动，目前停用

* + 变量的短信发送表

List<VariableSMS> SmsList

设置事件发送短信或电话拨号

* + 获得当前值的对应文字描述

string GetValueStateDesc();

方法返回当前状态值的文字描述

* + 将Value属性设置为缺省值

void SetDefaultValue();

* + 更新当前Value

void UpdateValue(object newValue)

通常是来自设备的更新

* + 克隆

IVariable Clone();

复制产生一个新的实例

## ITcpChannel

文件的位置：通用类库\GUHECommon\Interface\ITcpChannel.cs

派生自IChannel,便于TCP通信，增加了以下属性和方法：

* + 网络通讯协议

NetProcotol NetProtocol

TCP/UDP

* + 本地端口号

int Port

* + IP地址

string IpAddress

本地通信绑定的IP

* + 远端IP

string RemoteIp

* + 远端端口号

int RemotePort

* + 作为客户端还是服务端

bool IsClient

* + 发送字节流

void Send(byte[] cmd, string RemoteIp, int Port)

* + 发送文本

void Send(string cmd, string RemoteIp, int Port);

* + 接收

void OnReceived(string remoteIp, byte[] msg)

接收到的字节流

## ISerialChannel

文件的位置：通用类库\GUHECommon\Interface\ISerialChannel.cs

派生自IChannel,便于串口通信，增加了以下属性和方法：

* + 串口端口号

int Port

* + 串口波特率

int Baud

* + 串口通讯数据位

int DataBit

* + 串口通讯停止位

StopBits StopBit

* + 串口通讯奇偶校验方式

Parity Parity

* + 串口通讯握手方式

Handshake Handshake

* + 接收到的字节

void OnReceived(byte[] msg);

# 关键类的简介

## BaseChannel

文件的位置：通用类库\GUHECommon\ BaseClass\BaseChannel.cs

这是道讯通道的基类，这个基类实现了IChannel接口。通常开发一个新的通讯通道需要继承这个基类。

## BaseController

文件的位置：通用类库\GUHECommon\ BaseClass\BaseController.cs

这是控制器的基类，这个基类实现了IController接口。通常开发一个通讯插件需要定义一个控制器类，要继承这个基类。

## BaseVariable

文件的位置：通用类库\GUHECommon\ BaseClass\BaseVariable.cs

这是变量的基类，这个基类实现了IVariable接口。通常开发一个通讯插件需要定义一个变量类，要继承这个基类。

## Rtdb

文件的位置：通用类库\GUHECommon\ DataBase\Rtdb.cs

一个静态类，用于应用程序框架和插件操作通道、控制器、变量

* + 通道链表

public static List<IChannel> ChanList = new List<IChannel>();

* + 运行时动态生成的以变量名为索引的变量字典

public static Dictionary<string, IVariable> VarDict = new Dictionary<string, IVariable>();

* + 运行时动态生成的以变量ID为索引的变量字典

public static Dictionary<string, IVariable> VarDictID = new Dictionary<string, IVariable>();

## Almdb

文件的位置：通用类库\GUHECommon\ DataBase\Almdb.cs

一个静态类，用于缓存报警记录

* + 报警记录表

private static readonly List<AlarmMessage> AlarmList = new List<AlarmMessage>();

* + 插入一条新报警

public static void InsertAlarm(AlarmMessage newAlm)

* + 读取报警且清空报警列表

public static AlarmMessage[] GetAlarm()

* + 清空报警列表

public static void RemoveAllAlm()

* + 复位标识为报警变量

public static void ResetVarAlarm(User user)

## AlarmMessage

文件的位置：通用类库\GUHECommon\ BaseClass \AlarmMessage.cs

报警消息类，用于传递报警消息

* + 报警消息唯一性ID

public string AlarmGuid

* + 报警触发条件

public string AlarmWay

* + 报警描述

public string AlarmDesc

* + 报警时变量的值

public string AlarmValue

* + 报警变量的地址

public string AlarmAddr

* + 报警变量的IOSERVER名称

public string AlarmIOServer

* + 报警的通讯通道

public string AlarmChannel

* + 报警的控制器

public string AlarmController

* + 报警变量名称

public string AlarmVariable

* + 报警变量ID

public string AlarmVariableID

* + 报警变量描述

public string AlarmVariableDesc

* + 设备分类

public string DeviceLabel

* + 报警的区域

public string AlarmArea

* + 报警关联的视图

public string AlarmForm

* + 报警关联的视图2（辅助）

public string AlarmForm2

* + 报警关联的摄像机名称

public string AlarmVideo

* + 报警关联的摄像机名称2（辅助）

public string AlarmVideo2

* + 报警发生的时间

public DateTime DateStamp

* + 报警优先级

public uint Priority

* + 报警事件的类型

public int EventType

0:普通事件

1、报警事件

2、设备故障

* + 用于客户端标识是不是首次启动从服务端一次同步的报警列表

public bool ServerCacheAlm

(停用)

* + 报警产生的变量对象

public IVariable Variable

* + 报警优先级

public uint AlarmLevel

1. 普通 2-重要 3-紧急
   * 报警来源

public uint AlarmSource

普通事件=0,

防盗报警=1,

门禁报警=2,

视频报警=3,

火灾报警=4,

求助报警=5,

设备报警=6,

故障报警=7,

普通报警=8

* + 克隆

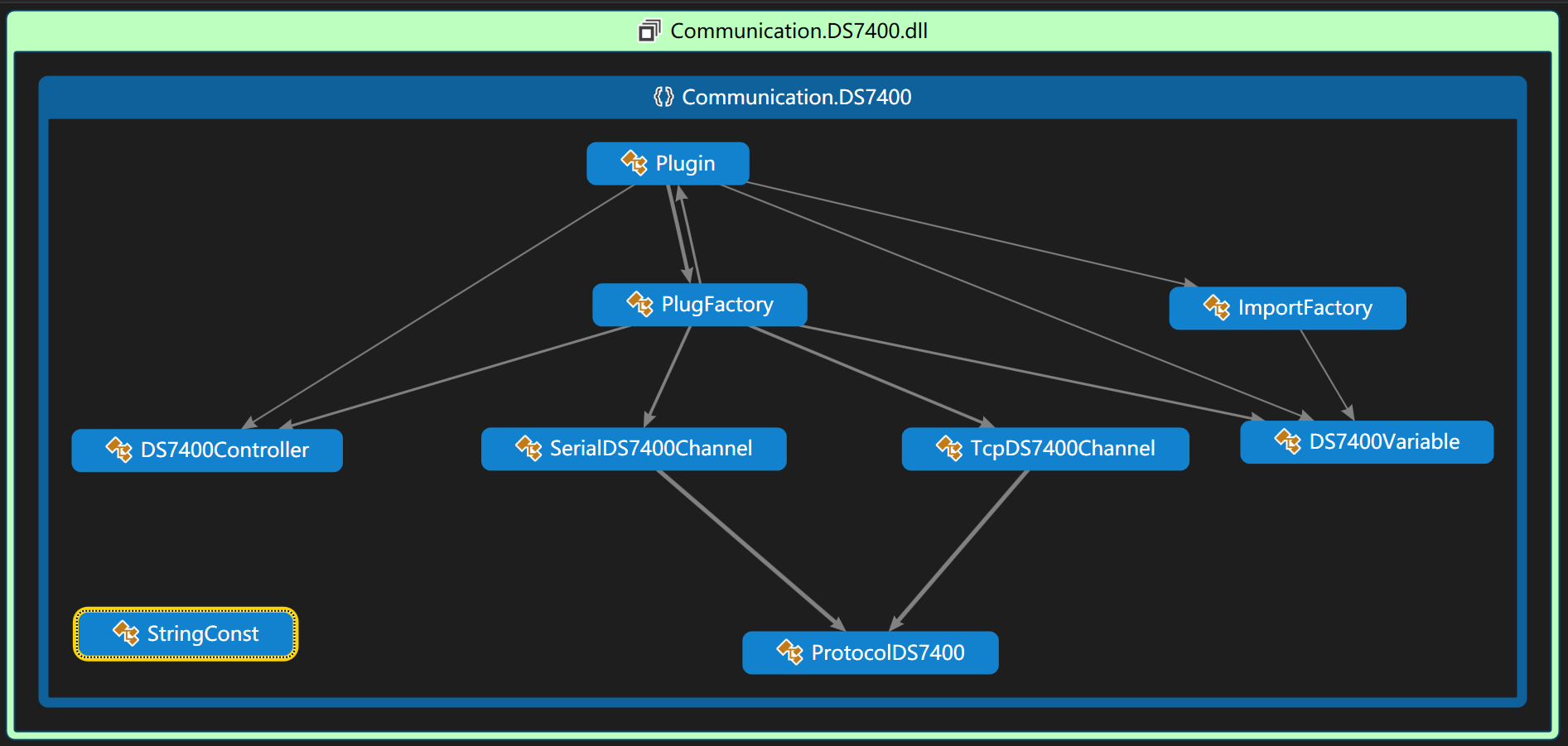
public AlarmMessage Clone()

* + 按时间排序

public static int SortCompare(AlarmMessage AF1, AlarmMessage AF2)

# 二次开发流程

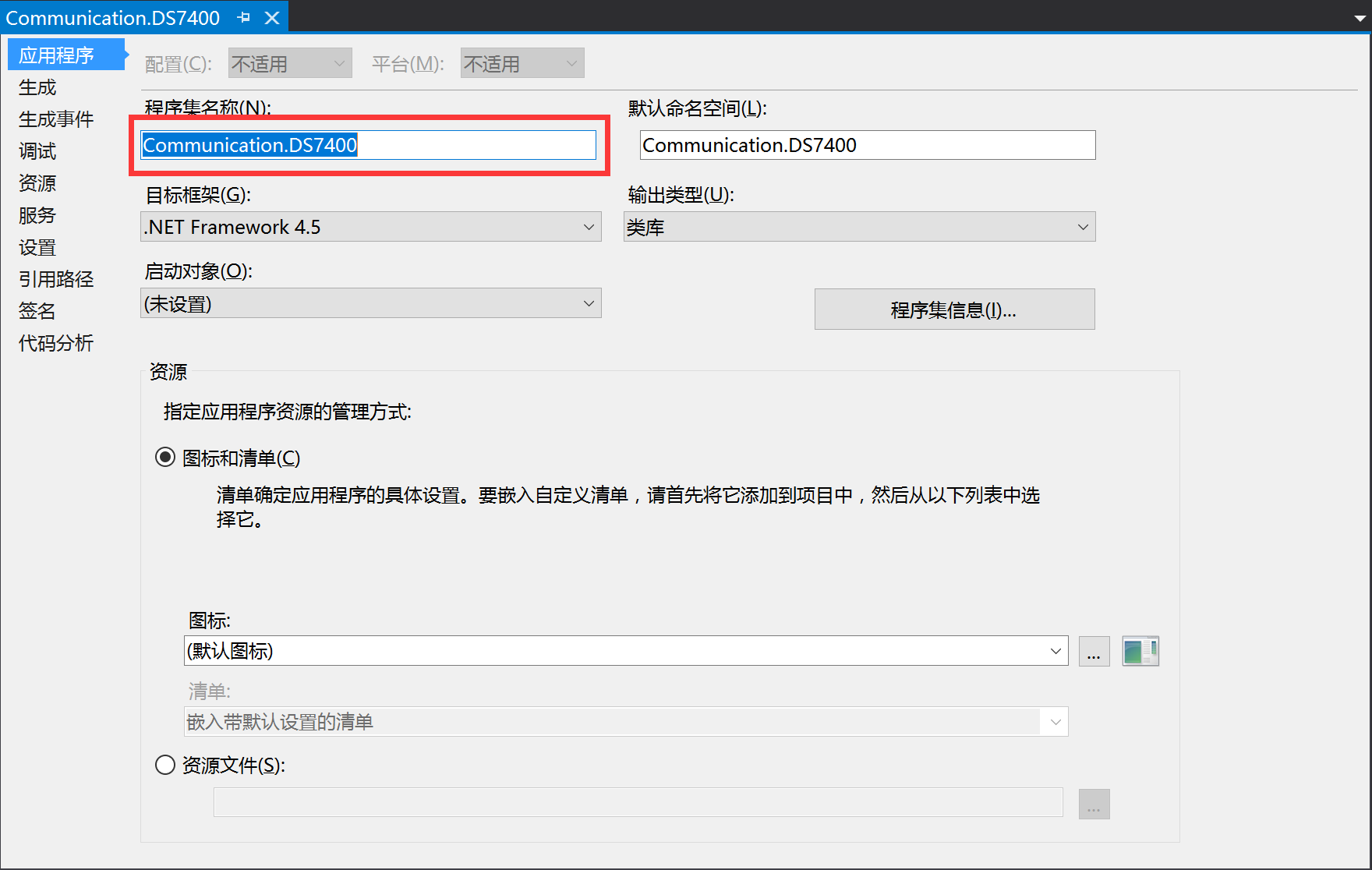
## 驱动插件代码图示例



## 新建一个类库项目

新建一个Windows类库项目，程序集的名称为：Communication.xxxxx，要注意的是Communication.是固定的前缀，是用于识别设备驱动插件的固定格式。

目标框架通常选.NET 4.5。



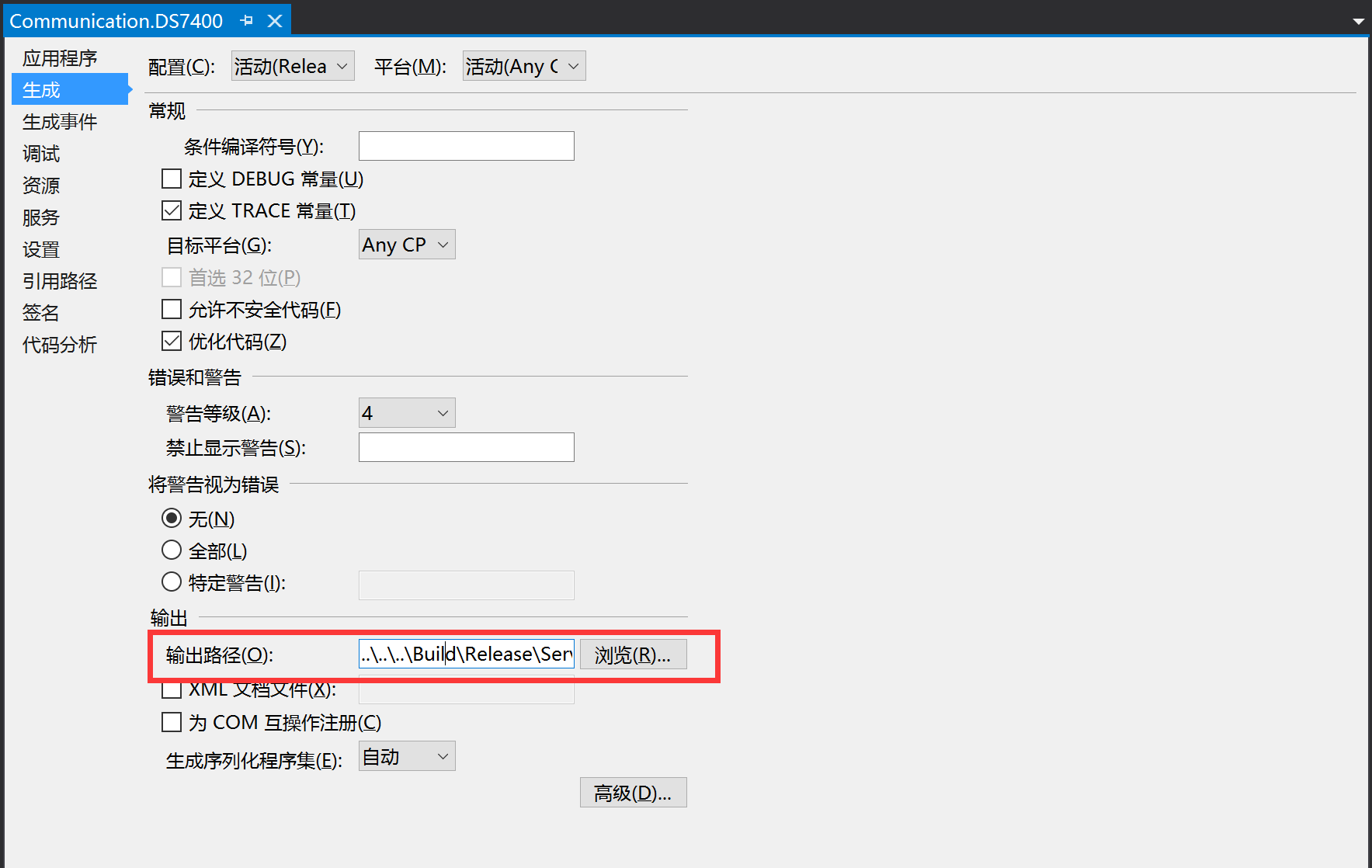
设置输出路径,输出到应用程序框架目录下。

Release:

..\..\..\Build\Release\Server\

Debug:

..\..\..\Build\Debug\Server\



## 新建一个Plugin类

public class Plugin:ICommunicationPlug

这个类实现了ICommunicationPlug接口，框架通过这个接口操作插件。

同时新建一个StringConst用于集中定义Plugin中用到的常量。

新建一个PlugFactory，用于在Plugin中调用，新建通道、控制器、变量等操作。

类与应用框架的对应如下：

* + 保存项目

界面：



类的方法：

public XmlElement SaveChannel(XmlDocument doc, IChannel cha)

将通道序列化为XmlElement ,由应用程序框架保存为XML文件。

* + 载入项目

界面：



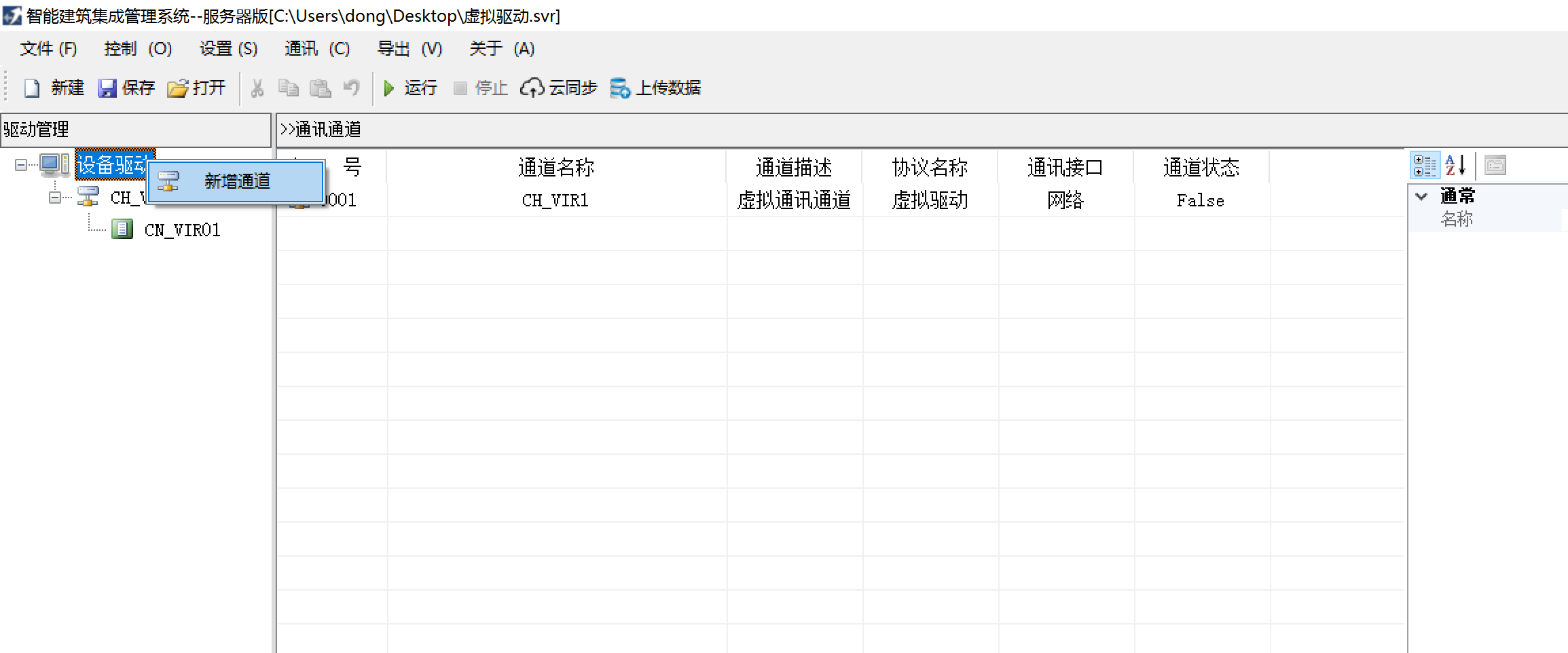
类的方法：

public IChannel CreateChannel(XmlNode node)

应用程序框架自XML文件读取XmlNode，由Plugin类利用该方法反序列化通道。

* + 新建通道

界面：



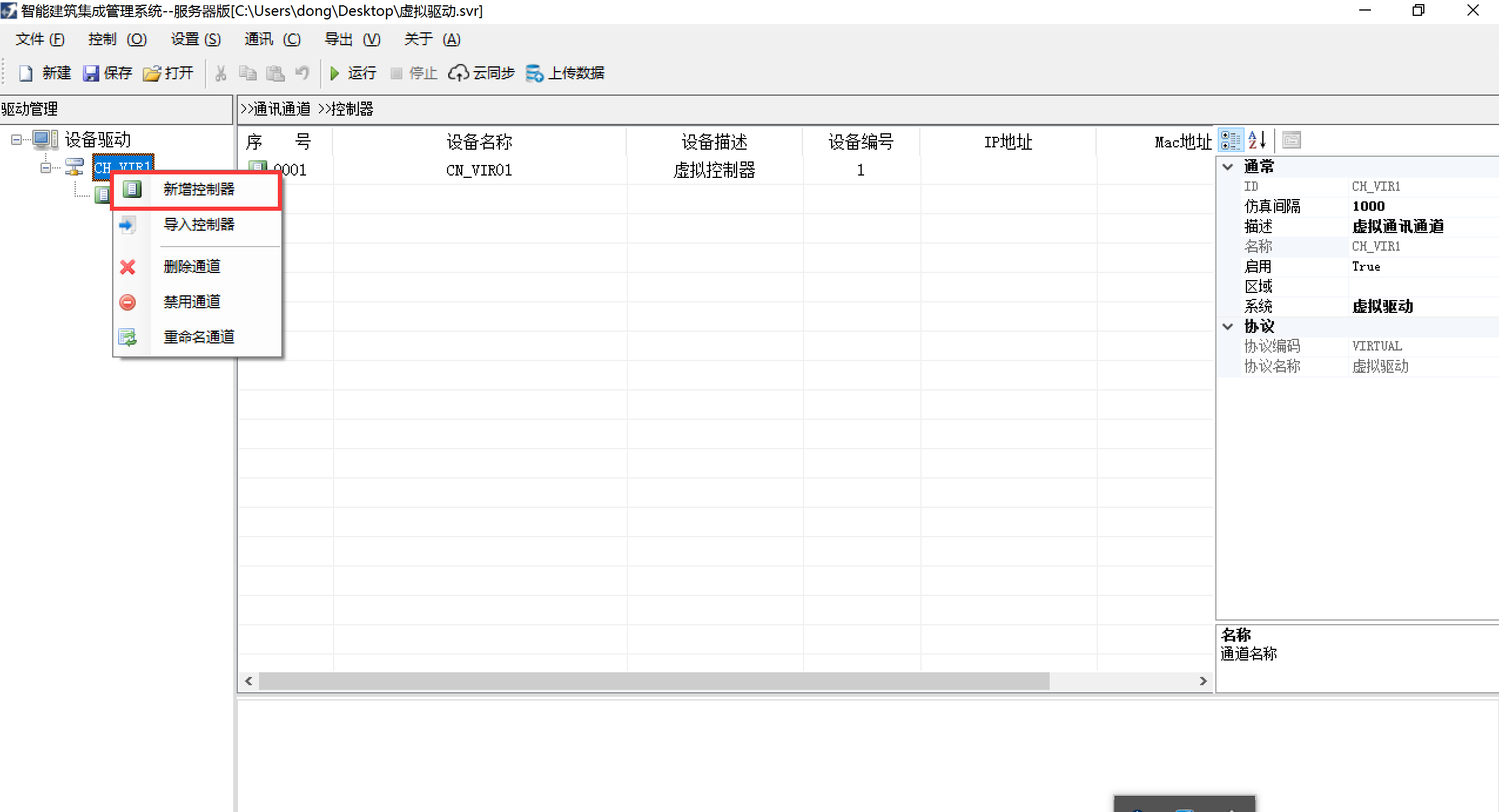
类的方法：

public IChannel CreateChannel(string code, string type)

其中依据type的不同，可以创建不同的通道，比如串口或TCP。

* + 新建控制器

界面：

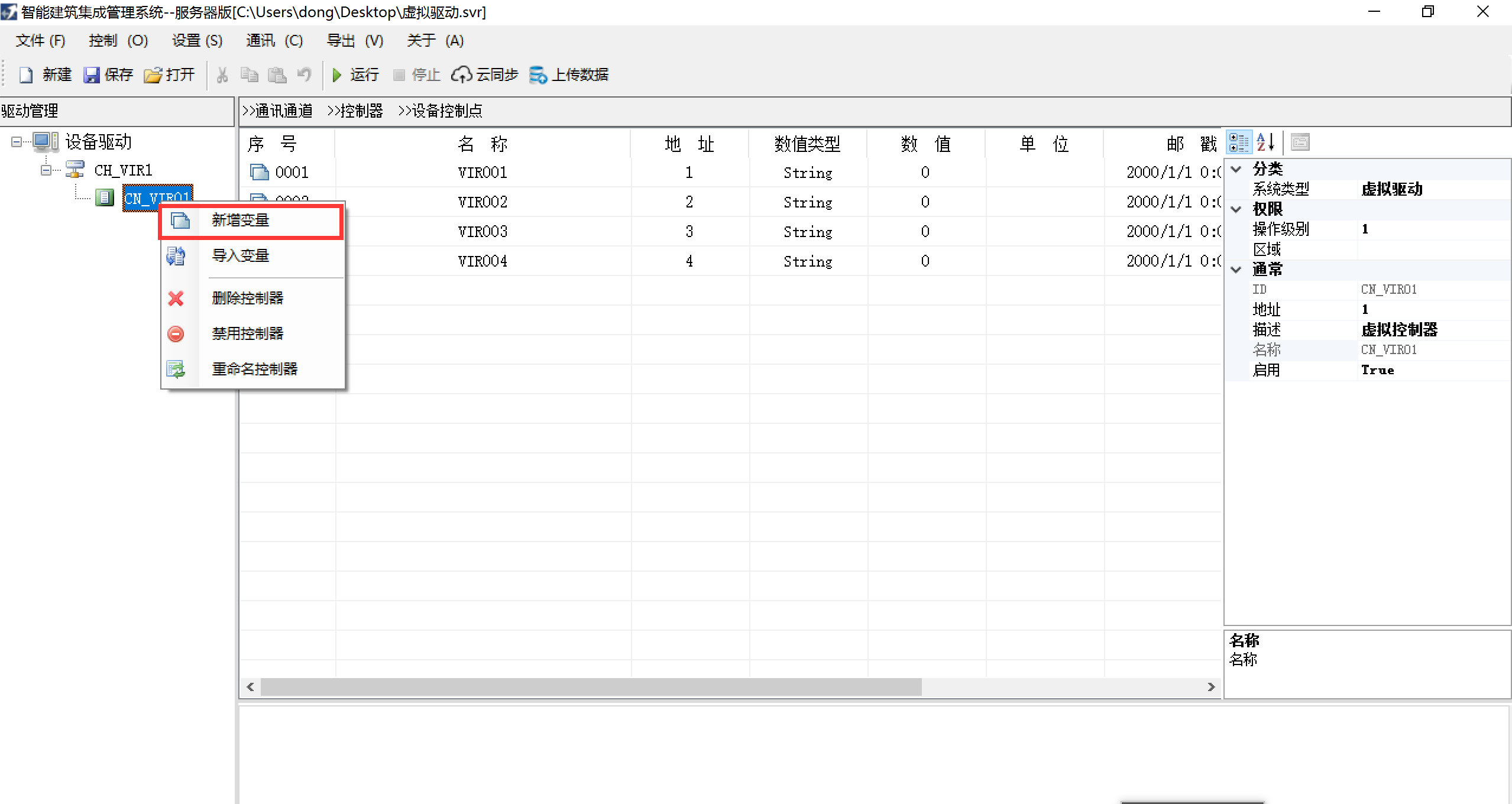


类的方法：

public IController CreateController(IChannel channel)

* + 新建变量

界面：

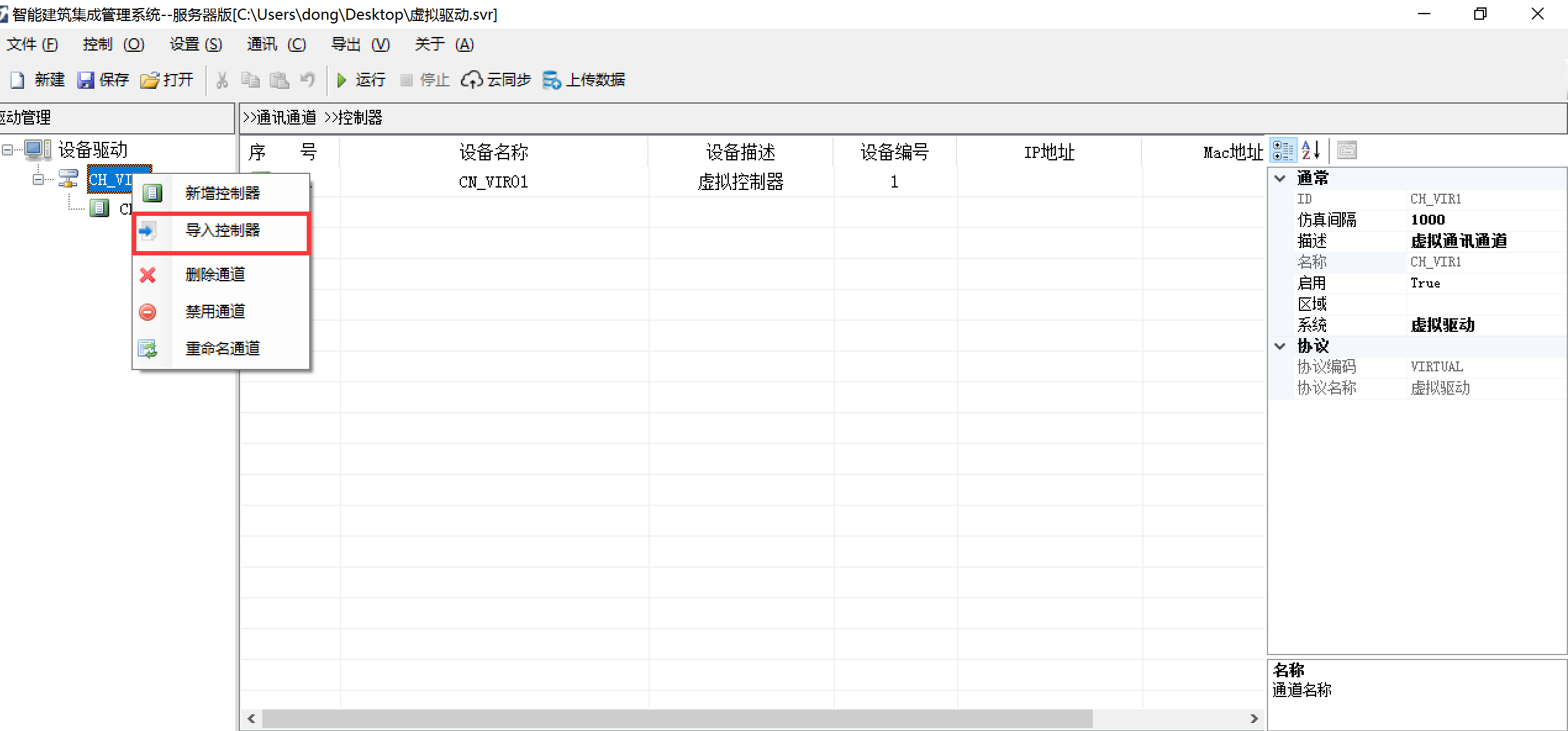


类的方法：

public IVariable CreateVariable(IController controller)

* + 导入控制器

界面：



类的方法：

public List<IController> ImportController(IChannel channel)

{

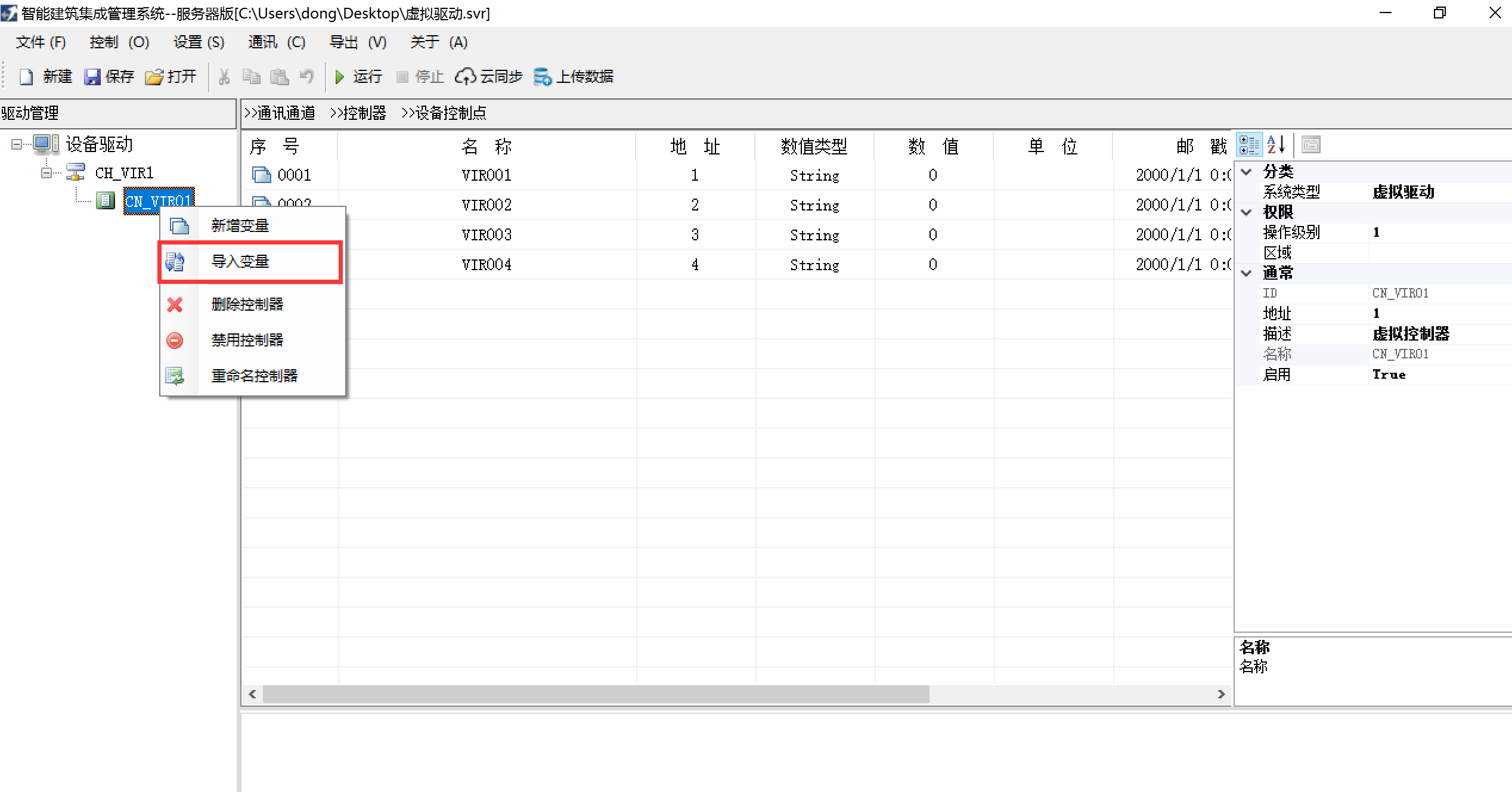
return new List<IController>();

}

批量导入预置的控制器，如果没有，返回一个空的List。

* + 导入变量

界面：



类的方法：

public bool ImportVariable(IController controller)

{

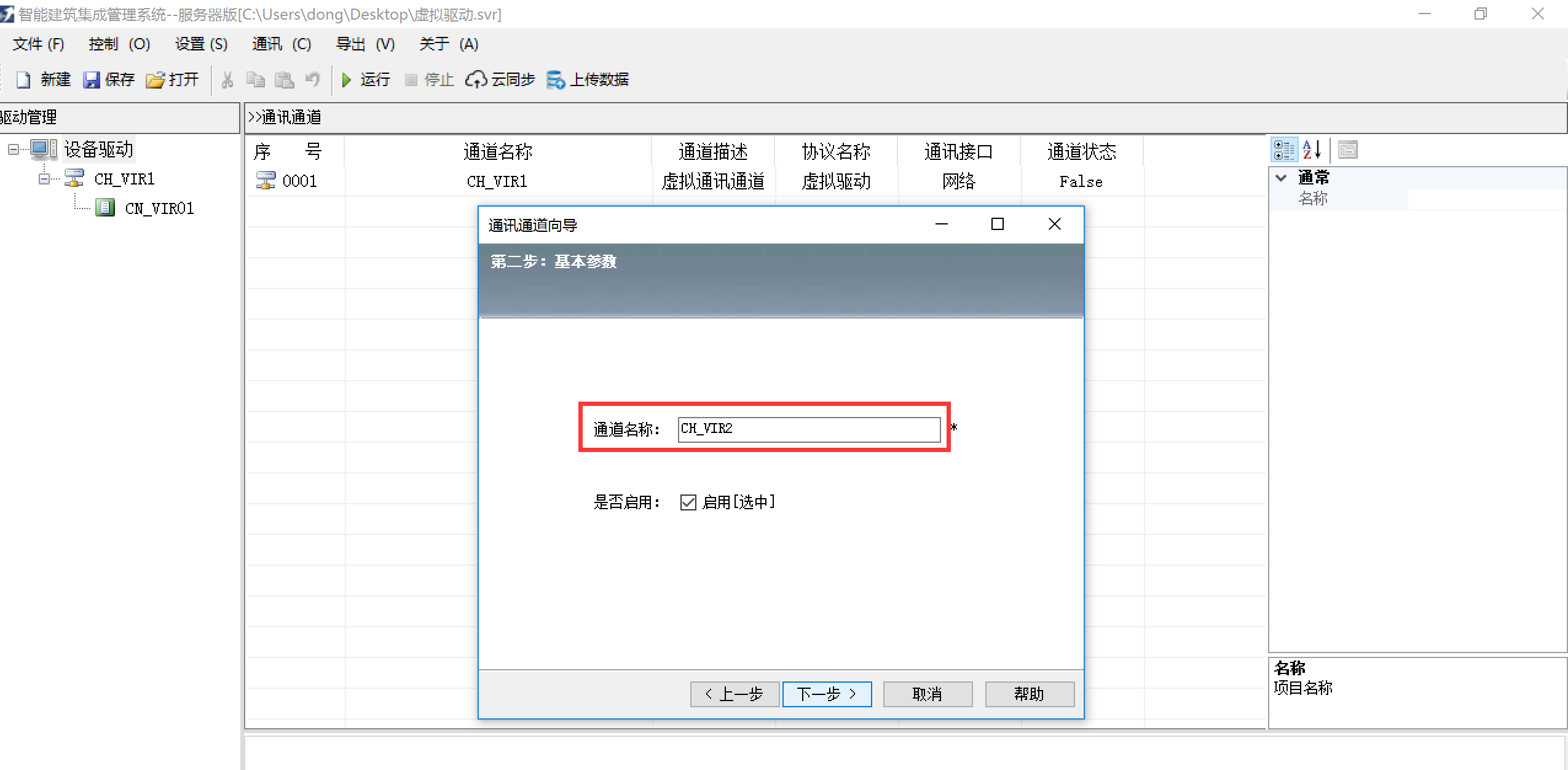
return True;

}

批量导入预置的变量，如果没有，返回True。

* + 新增通道时获取默认名称

界面：

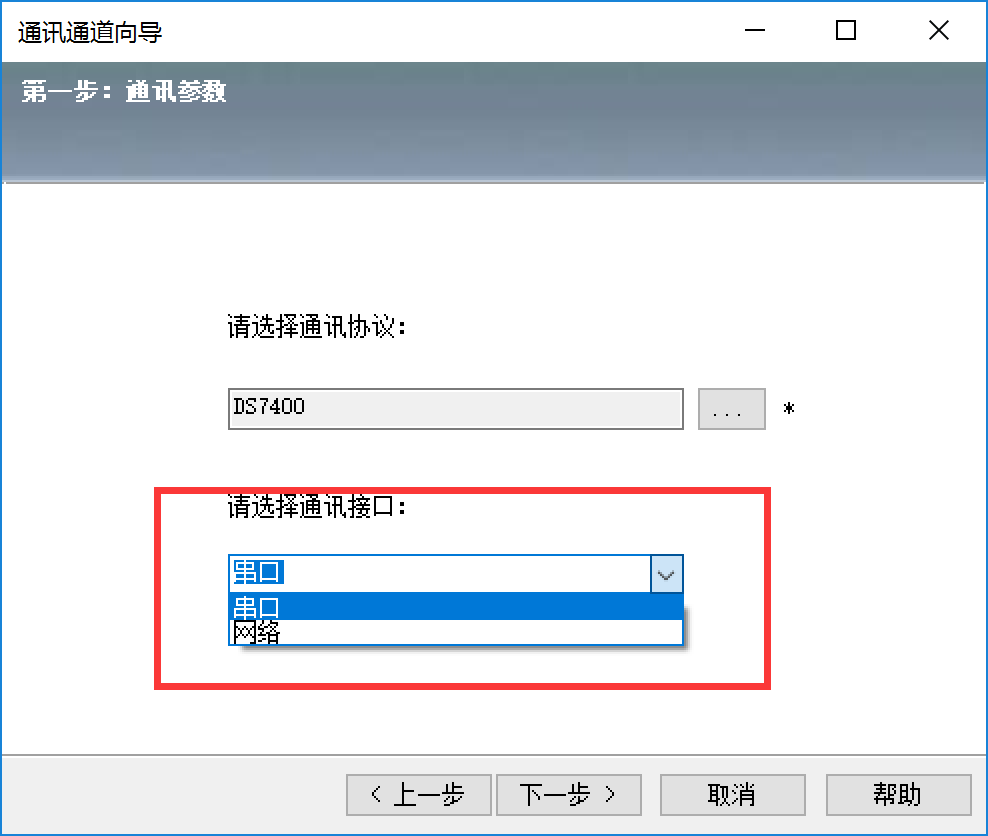


类的方法：

public string GetUniqueName(UniqueNameType type)

* + 选择通讯接口

界面：



类的方法：

public string[] GetCommInterface()

{

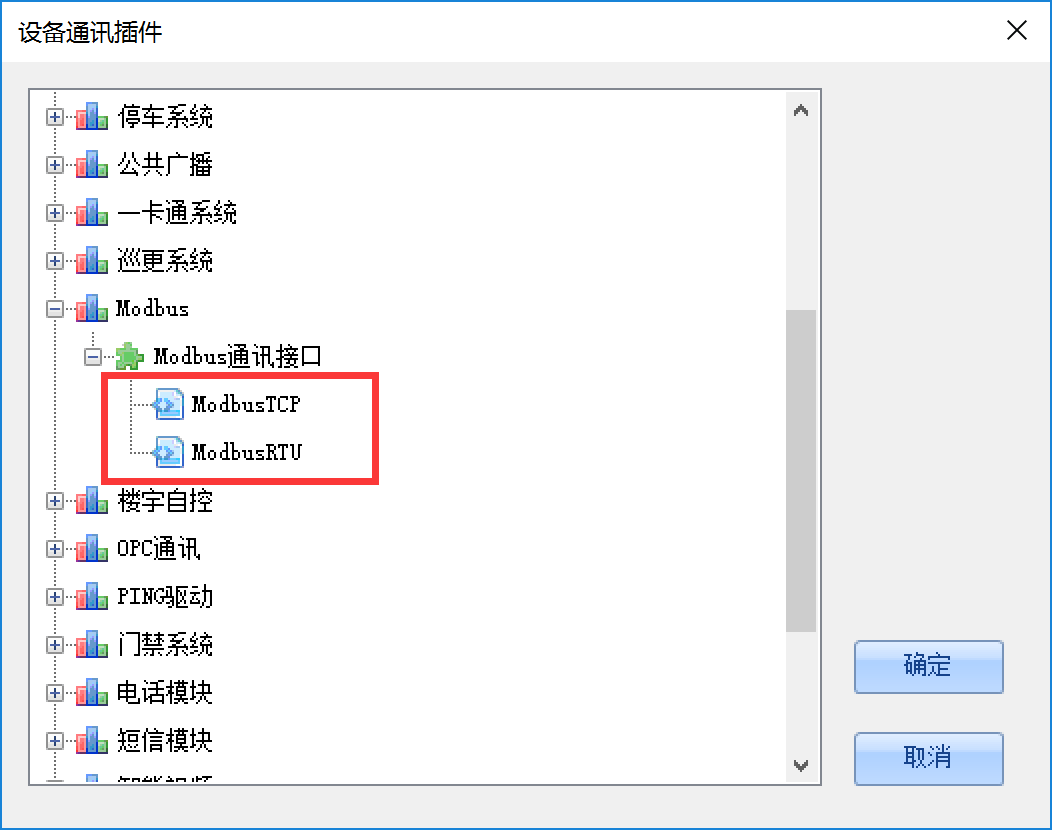
return new string[] { CommInterface.串口.ToString(), CommInterface.网络.ToString() };

}

类中自定义接口，界面下接列表中会显示。

* + 新增通道->选择通讯协议

界面：



类的方法：

public List<IProtocol> ProtocolList

一个通道内可以支持多个协议，在这个方法中获取。

## 新建一个Channel类

public class XXXXChannel : BaseChannel

比如：

public class TcpDS7400Channel : TcpChannel,IAlarmChannel

public class SerialDS7400Channel : SerialChannel, IAlarmChannel

TcpChannel、SerialChannel都是派生自BaseChannel，实现了IChannel。

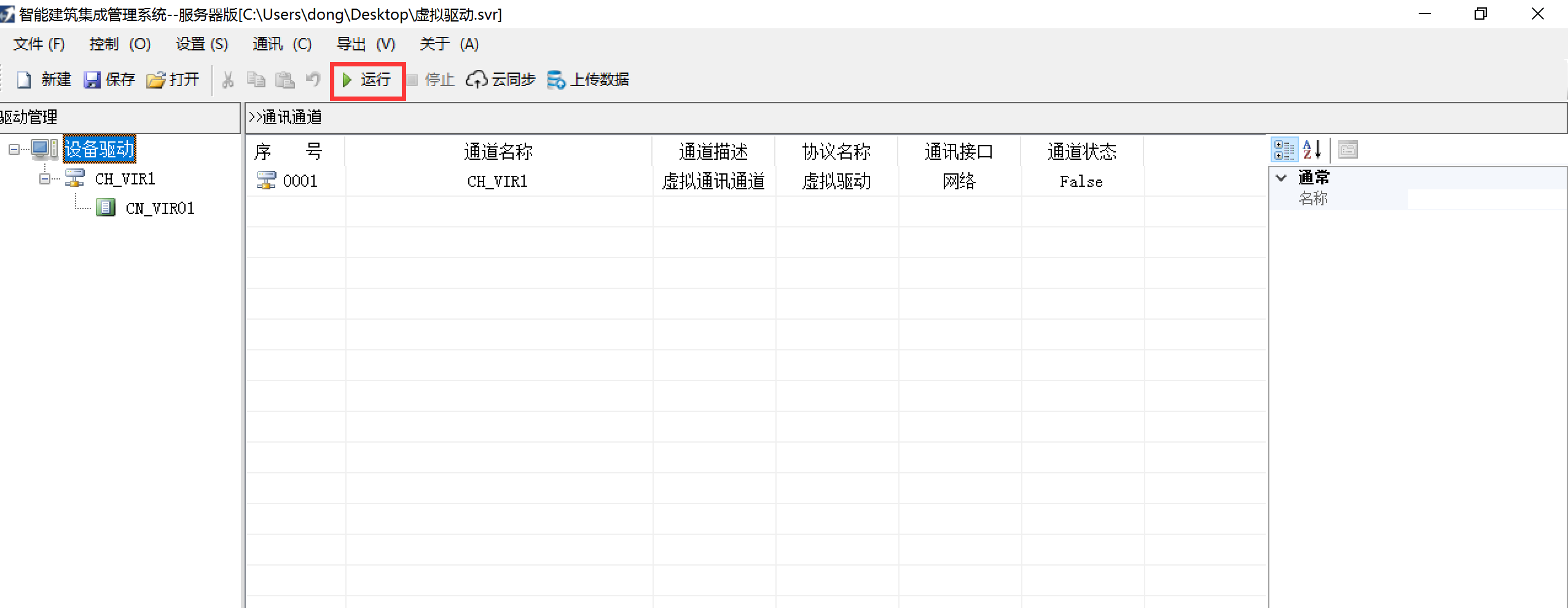
这个新建Channel类，继承BaseChannel，自可以依据需要在基类的基础上增加个性的属性和方法。

同时新建通道类中需要重写以下方法：

* + 启动通道通讯方法

public override void Start()

对应的UI界面操作：



这时应用程序框架操控通道的入口，通常要依据通讯协议实现这个方法，以下是一些经验：

* + - 串口设备

打开串口端口，注册串口数据接收事件，响应数据接收事件，并处理这个事件，最终依据协议分析，将接到的信息，变为变量的数值变化。

* + - 网络设备

创建一个TcpClient，连接远端的Tcp设备，开启一个接收线程，定时接收设备发送的数据，最终依据协议分析，将接到的信息，变为变量的数值变化。

* + - SDK开发包设备

如果提供的是C、C++开发包，首先要用C#进行封装，然后依据开发包的接口，发送和接收数据，最终依据协议分析，将接到的信息，变为变量的数值变化。

* + 停止通道通讯方法

public override void Stop()

这时应用程序框架操控通道的入口，通常要依据通讯协议实现这个方法，以下是一些经验：

* + - 串口设备

关闭串口，清理资源。

* + - 网络设备

断开连接，清理资源。

* + - SDK开发包设备

调用SDK的退出清理接口。

对应的UI界面操作：

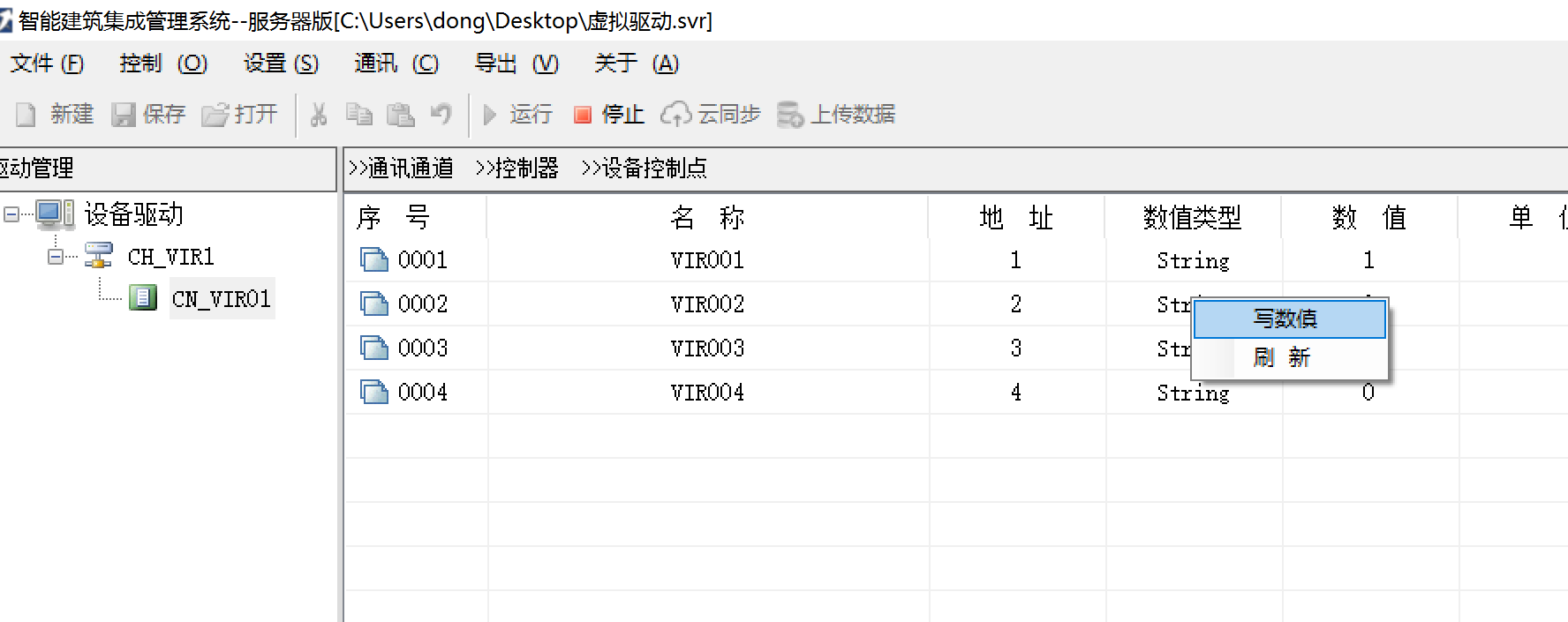


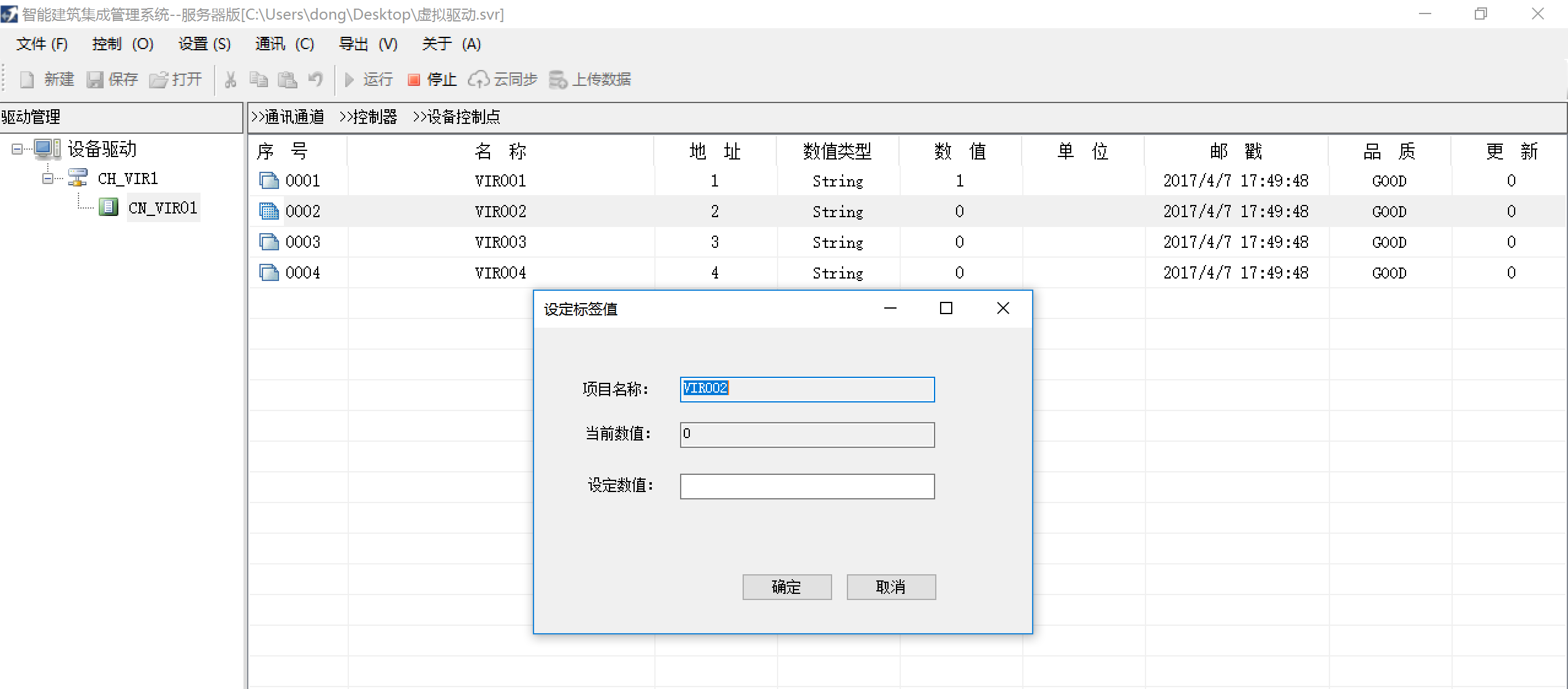
* + 向设备写操作

public override bool WriteValue(IVariable var, object value)

重写这个方法，在这个方法内实现向设备写操作，通常用户界面上对设备进行控制的操作，最终会调用这个方法，在这个方法内依据设备协议的实际情况，编写对应的指令，并通过设备通道接口发送到设备。

对应的UI界面操作：





## 新建一个Controller类

public class XXXXController : BaseController

这个新建Controller类，继承自BaseController，可以依据需要在基类的基础上增加个性的属性和方法。

## 新建一个Variable

public class XXXXVariable : BaseVariable

这个新建变量类，继承自BaseVariable，可以依据需要在基类的基础上增加个性的属性和方法。

# 常见问题

## 程序启动后找不到你开发的驱动

首先检查驱动的文件名是不是Communication.xxxxx，并且和应用程序在同一级目录下。

其次检查：

internal class StringConst

{

public const string PluginName = "DS7400报警接口";

public const string PluginID = "DS7400Channel";

public const string ProtocolName = "DS7400";

public const string ProtocolCode = "BJ\_BOSCH\_DS7400";

…

}

检查PluginID、ProtocolName、ProtocolCode这些定义是否和其它插件相同。

## 如何实现和设备之间的通讯

在驱动中Channel类的Start()方法中，按照设备提供的协议或开发包的方法，启动接口与设备通讯。

## 如何让设备数据可视化

设备的数据主要通过变量来可视化，来自Channel类中接收到设备发来的数据后，通过协议解析，最终通过Variable类的

public void UpdateValue(object newValue)方法更变量值。