上海格西信息科技有限公司

设备仿真器例子

版本 0.1

目录

1. 概述

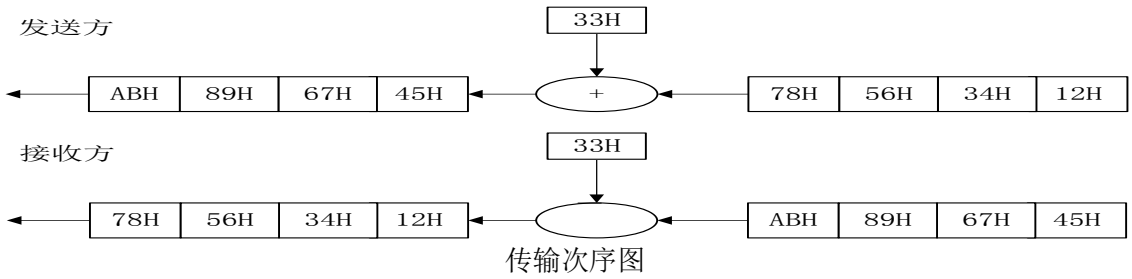
电子研发过程中，经常需要仿真一个功能或者设备，来测试另一个功能或者设备。仿真一个设备，通常是分为两部分：一部分是仿真其通信指令；另一部分是仿真其寄存器数据。本例子通过仿真智能电表的读功能来演示如何使用格西测控大师来仿真设备。

国内 DL/T645-2007 协议是智能电表通信协议，为主-从结构的半双工通信方式，智能电表为从站。每帧由帧起始符、从站地址域、控制码、数据域长度、数据域、帧信息纵向校验码及帧结束符 7 个域组成。每部分由若干字节组成。

帧是传送信息的基本单元。帧格式如下图所示。

说　　明	代　码
帧起始符	68H
地址域	A0
	A1
	A2
	A3
	A4
	A5
帧起始符	68H
控制码	C
数据域长度	L
数据域	DATA
校验码	CS
结束符	16H

传输次序：所有数据域均先传送低位字节，后传送高位字节，发送方按字节进行加33H处理，接收方按字节进行减33H处理。数据传输的举例：电能量值为123456.78kWh，其传输次序如图。



传输次序图

本例子只演示读数据命令，命令的定义如下：

主站请求帧

a) 功能：请求读电能表数据

b) 控制码：C=11H

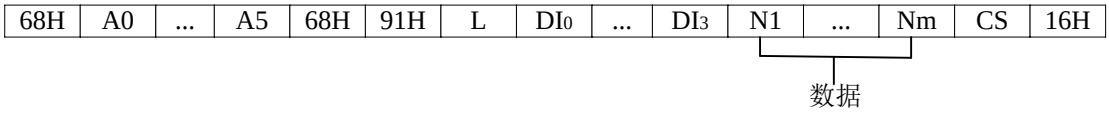
c) 数据域长度：L=04H（数据长度）

d) 帧格式：

从站正常应答

a) 控制码：C=91H 无后续数据帧。

b) 数据域长度：L=04H+m（数据长度）。

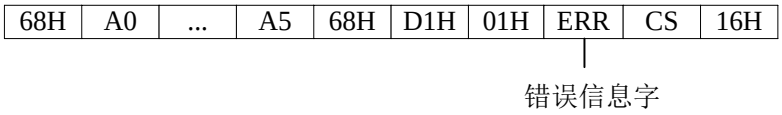


从站异常应答帧

a) 控制码：C=D1H

b) 数据域长度:L=01H

c) 帧格式：



本例子文件位于：<软件安装目录>\Examples\Solutions\DeviceSimulation\DeviceSimulator。

文件说明：

✓ DeviceSimulator.gpj - 设备仿真演示项目 - 中文 – 串口版

例子自带仿真器，可以脱离设备仿真运行。

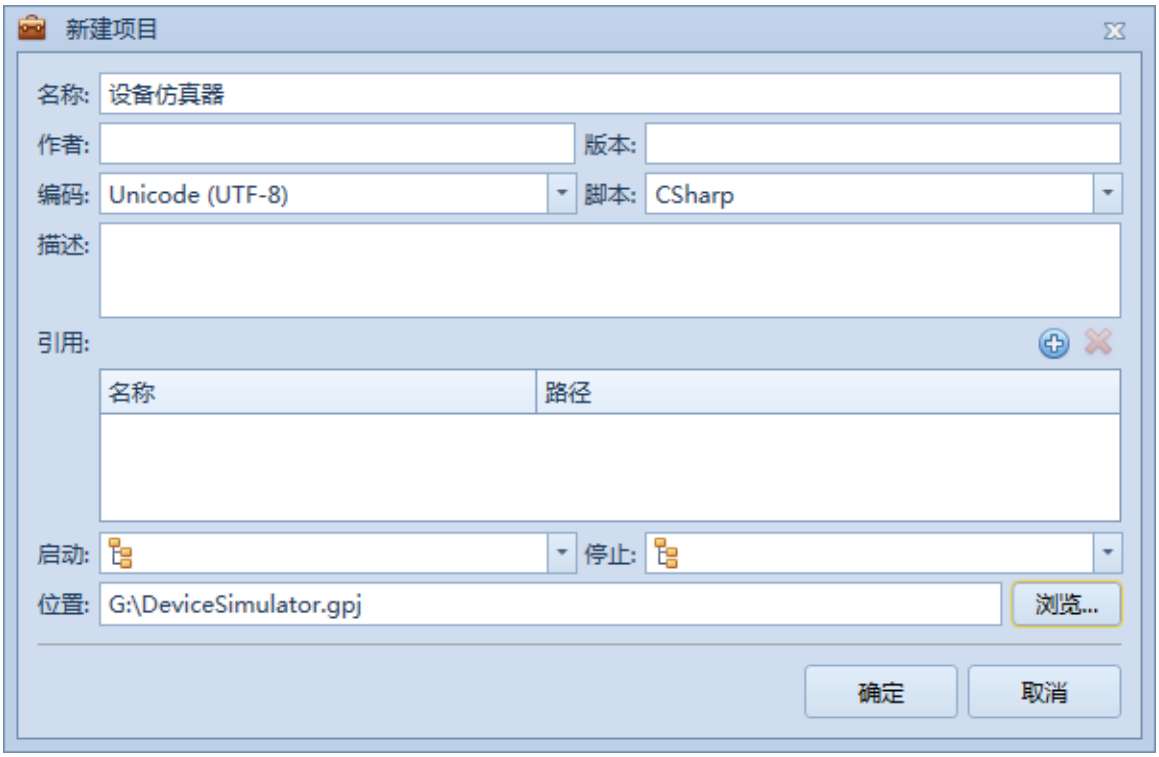
串口版：需要使用串口虚拟软件，如 VSPD 等，虚拟出一对串口（COM2 和 COM3）进行仿真运行。如果虚拟的串口号和例子预定义的串口号不同，可以修改例子串口号，也可以修改虚拟串口号。

2. 创建项目

2.1 第 1 步 新建项目

启动格西测控大师，在左上角菜单中选择“新建项目”，然后在弹出的“新建项目”对话框中，填写项目名称“设备仿真器”，然后点击“浏览...”按钮，选择保存路径和填写项目文件名“设备仿真器”，最后点击“确定”按钮。

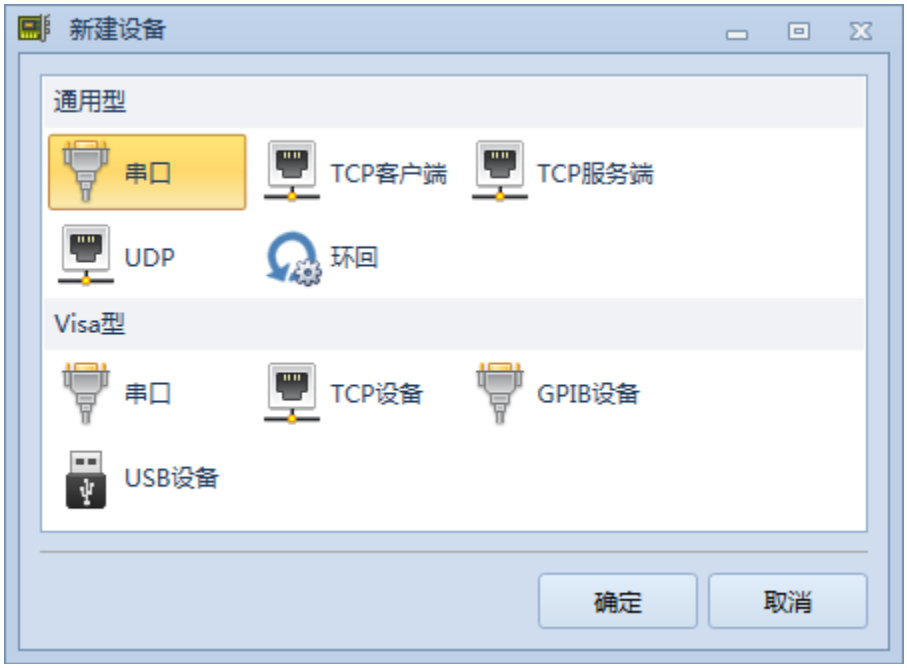
其中，“启动”属性用于设置项目开始运行时自动执行的序列，待后续创建了启动序列再设置。



2.2 第 2 步 添加串口设备

在项目管理器中选择“设备与接口”节点，然后点击鼠标右键，在弹出菜单中选择“新建设备...”。

弹出新建设备对话框中，选择“串口”，点击“确定”。



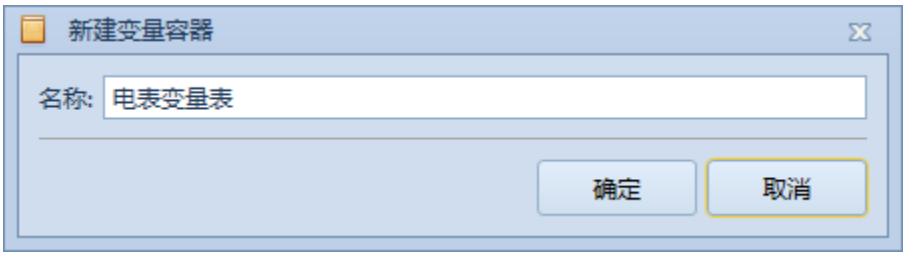
弹出设备属性对话框，填写“设备名”和其他设备参数，最后点击“确定”按钮。其中，“设备

名”是设备的标识，可以是任意字符串，引用设备必须使用设备名。

2.3 第 3 步 添加变量

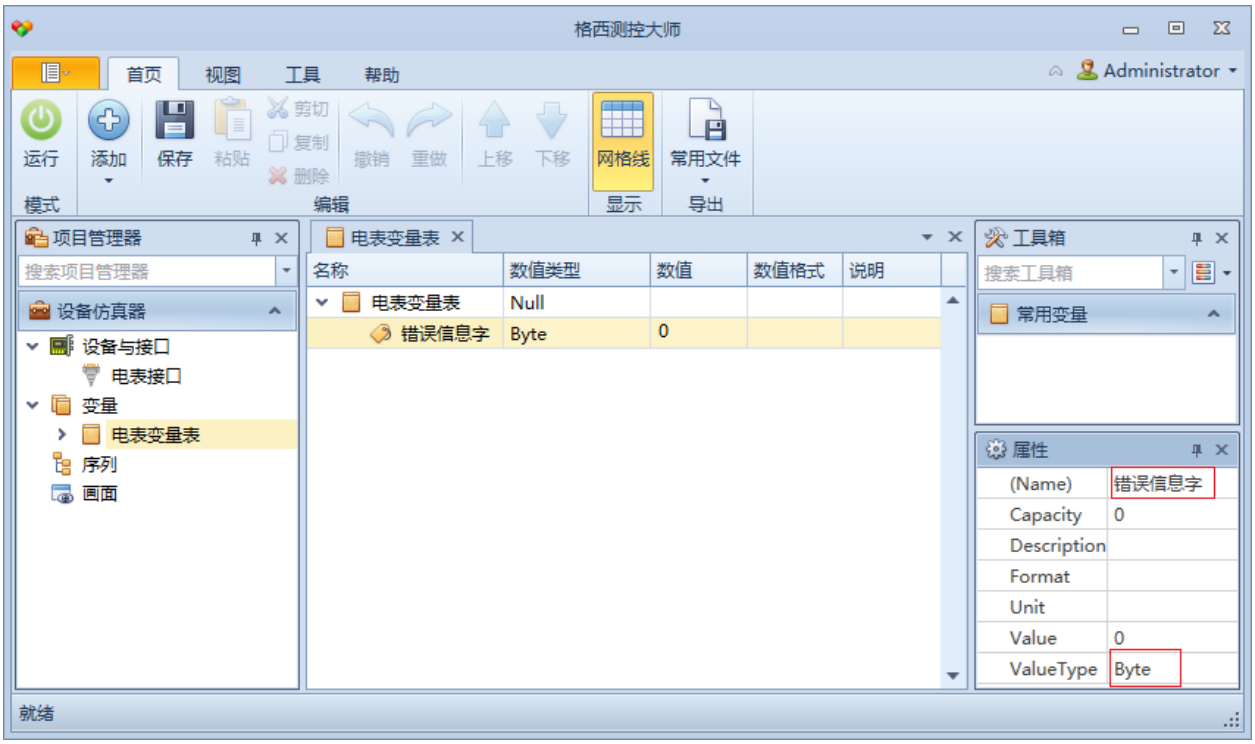
在项目管理器中选择“变量”节点，然后点击鼠标右键，在弹出菜单中选择“新建变量容器...”。

弹出新建变量容器对话框中，填写“名称”，点击“确定”。



在项目管理器中选择新创建的“电表变量表”节点，然后双击鼠标左键，或者点击鼠标右键，在弹出菜单中选择“编辑...”，打开变量编辑页面。

在“电表变量表”编辑页面，选中“电表变量表”节点，单击鼠标右键，在弹出菜单中选择“添加 ->变量”，并设置变量的名称（Name）和值类型（ValueType）。



2.4 第 4 步 添加序列

本例子建立两个序列，分别是“电表仿真器”和“启动”序列，其中“启动”序列用于放置全局仿真脚本和启动运行。

2.4.1 “启动”序列

双击“启动”序列节点，打开变量编辑页面，然后在工具栏中点击“脚本”按钮，进入脚本编辑页面。在这里实现电表寄存器数据仿真和电表仿真器全局类。

```
// 电表仿真器数据
public class SimulatorData
{
  public string DI {get;set;}
  public string Value {get;set;}
  public int Length {get;set;}
}

// 电表仿真器全局类，用于模拟电表的通信状态和寄存器状态
public static class Simulator
{
  private static Dictionary<string, SimulatorData> s_data; // 保存以 DI 为 Key 的所有需要仿真的设备数据

  //
  public static string LastReadAddress { get; private set; } // 最近一次读命令的设备地
```

```
址
  public static string LastReadDI { get; private set; }   // 最近一次读命令的 DI
  public static string LastReadData { get; private set; }  // 最近一次读命令的数据
  public static int LastReadDataLength { get; private set; } // 最近一次读命令的数据长
度
  public static int LastReadErrorCode { get; private set; } // 最近一次读命令的错误信
息码
  //
  public static string LastWriteAddress { get; private set; } // 最近一次写命令的设备
地址
  public static string LastWriteDI { get; private set; }   // 最近一次写命令的 DI
  public static int LastWriteErrorCode { get; private set; } // 最近一次写命令的错误信
息码

  //
  public static void Initialize()
  {
    s_data = new Dictionary<string,SimulatorData>();

    s_data.Add("0x02010100",new SimulatorData() {DI="0x02010100", Value="0x0001",
Length=2}); // 当前 A 相电压值
    s_data.Add("0x02010200",new SimulatorData() {DI="0x02010200", Value="0x0002",
Length=2}); // 当前 B 相电压值
    s_data.Add("0x02010300",new SimulatorData() {DI="0x02010300", Value="0x0003",
Length=2}); // 当前 C 相电压值

    s_data.Add("0x02020100",new SimulatorData() {DI="0x02020100", Value="0x000001",
Length=3}); // 当前 A 相电流值
    s_data.Add("0x02020200",new SimulatorData() {DI="0x02020200", Value="0x000002",
Length=3}); // 当前 B 相电流值
    s_data.Add("0x02020300",new SimulatorData() {DI="0x02020300", Value="0x000003",
Length=3}); // 当前 C 相电流值
  }

  public static string GetDataValue(string di)
  {
    if(s_data.ContainsKey(di))
    {
      return s_data[di].Value;
    }
    return null;
  }

  public static int GetDataValueLength(string di)
  {
    if(s_data.ContainsKey(di))
    {
      return s_data[di].Length;
```

```
    }
    return 0;
  }

  public static void SetDataValue(string di, string value)
  {
    s_data[di].Value = value;
  }

  public static bool ContainsData(string di)
  {
    return s_data.ContainsKey(di);
  }

  // 读命令，在 读数据.request 协议步骤中使用，记录本次读命令的各个参数，以便在后续响应帧中使用
  public static int Read(string address, string di)
  {
    int errorCode = 1;
    LastReadAddress = address;
    LastReadDI = di;
    if(ContainsData(di))
    {
      LastReadData = GetDataValue(di);
      LastReadDataLength = GetDataValueLength(di);
      errorCode = 0;
    }
    LastReadErrorCode = errorCode;
    return errorCode;
  }

  // 写命令
  public static int Write(string address, string di, string data)
  {
    int errorCode = 1;

    //
    LastWriteAddress = address;
    LastWriteDI = di;
    if(ContainsData(di))
    {
      errorCode = 0;
    }
    LastWriteErrorCode = errorCode;

    return errorCode;
  }
```

```
  // 将传入的位串按字节加 0x33，返回新的位串
  public static BitString Add33(BitString origin)
  {
    byte[] data = origin.Bytes;
    for(int i = 0; i< data.Length; i++)
    {
      data[i] = (byte)(data[i] + 0x33);
    }
    return new BitString(data);
  }
  // 将传入的位串按字节减 0x33，返回新的位串
  public static BitString Subtract33(BitString origin)
  {
    byte[] data = origin.Bytes;
    for(int i = 0; i< data.Length; i++)
    {
      data[i] = (byte)(data[i] - 0x33);
    }
    return new BitString(data);
  }
}
```

“启动”序列步骤的脚本，用于启动和配置仿真器运行。

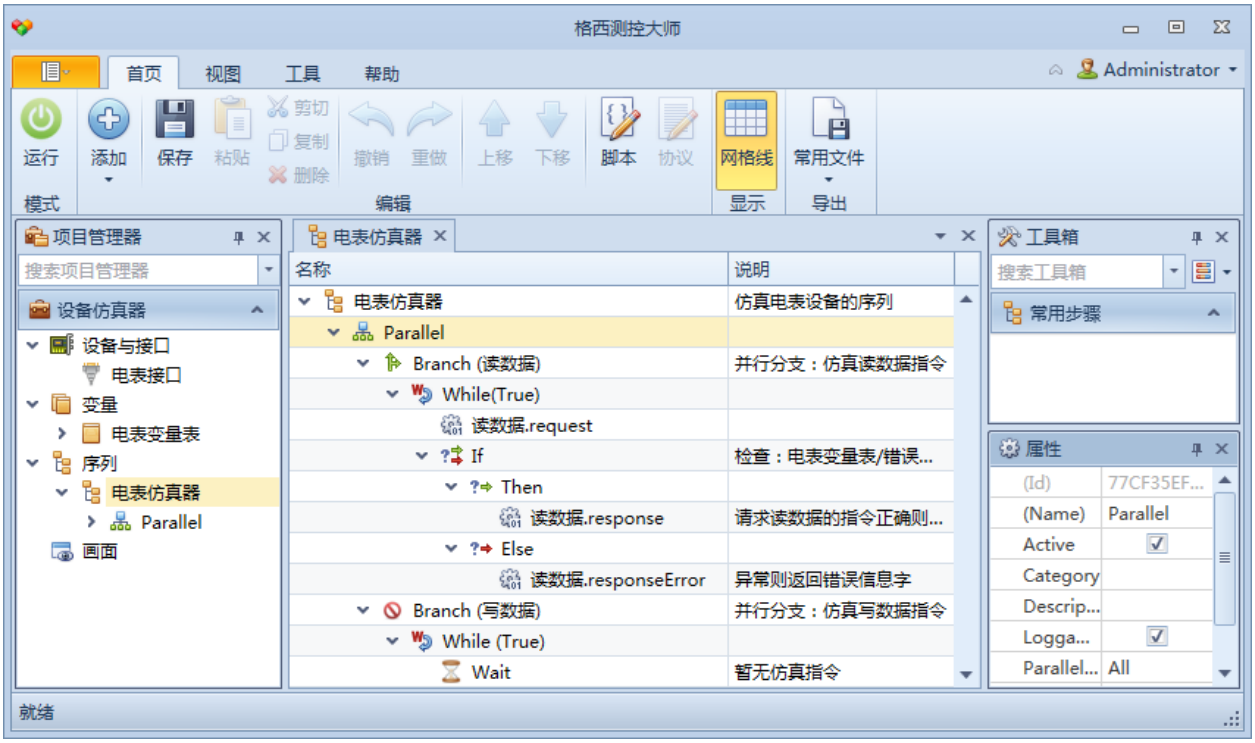
```
public class Step_C977A1247AFE480F94445222D33FEFA3
{
  public ScriptContext Context { get; set; }
  //
  public Int32 BeginExecute(IStepContext context, IStep step)
  {
    Simulator.Initialize();
    // 打开电表仿真器串口通信接口
    IDeviceSession dev = this.Context.GetDeviceSession("电表接口");
    dev.Open();
    // 启动 电表仿真器 序列
    this.Context.StartStep("电表仿真器");
    // 打开 电表仿真器 页面
    this.Context.CloseAllEditors();
    this.Context.OpenSchema("电表仿真器画面");
    return 0;
  }
  //
  public Int32 EndExecute(IStepContext context, IStep step)
  {
    return 0;
  }
}
```

2.4.2 “电表仿真器”序列

电表仿真器序列采用 Parallel 型步骤，并行仿真不同类型的指令，本例子用一个分支仿真读数据指令。在独立运行的分支中，循环执行：等待读指令->判断读指令合法性->根据判断判断回应正确响应帧和异常响应帧。

其中，“读数据.request”、“读数据.response”和“读数据.responseError”这三个 ProtocolAction 型步骤中，均配置了协议和脚本，可以通过工具栏的“协议”和“脚本”按钮打开查看和编辑。

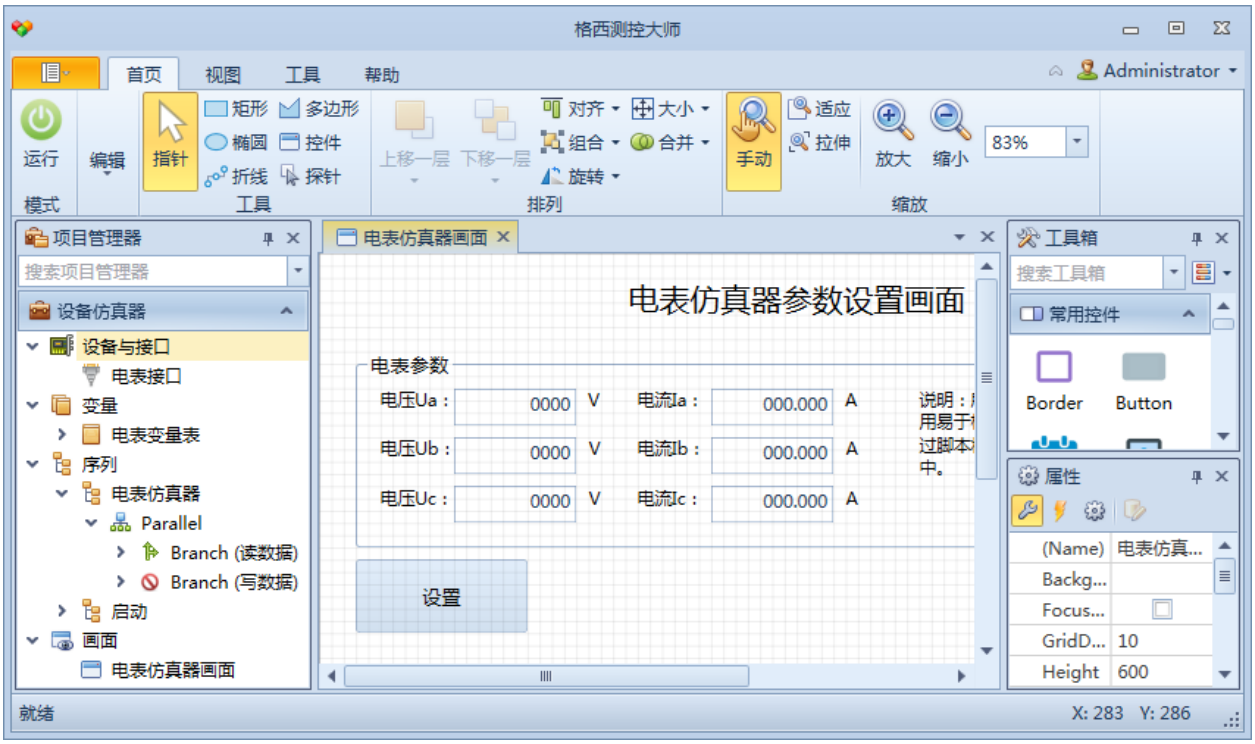
各个步骤的详细配置请参考本例子项目文件。



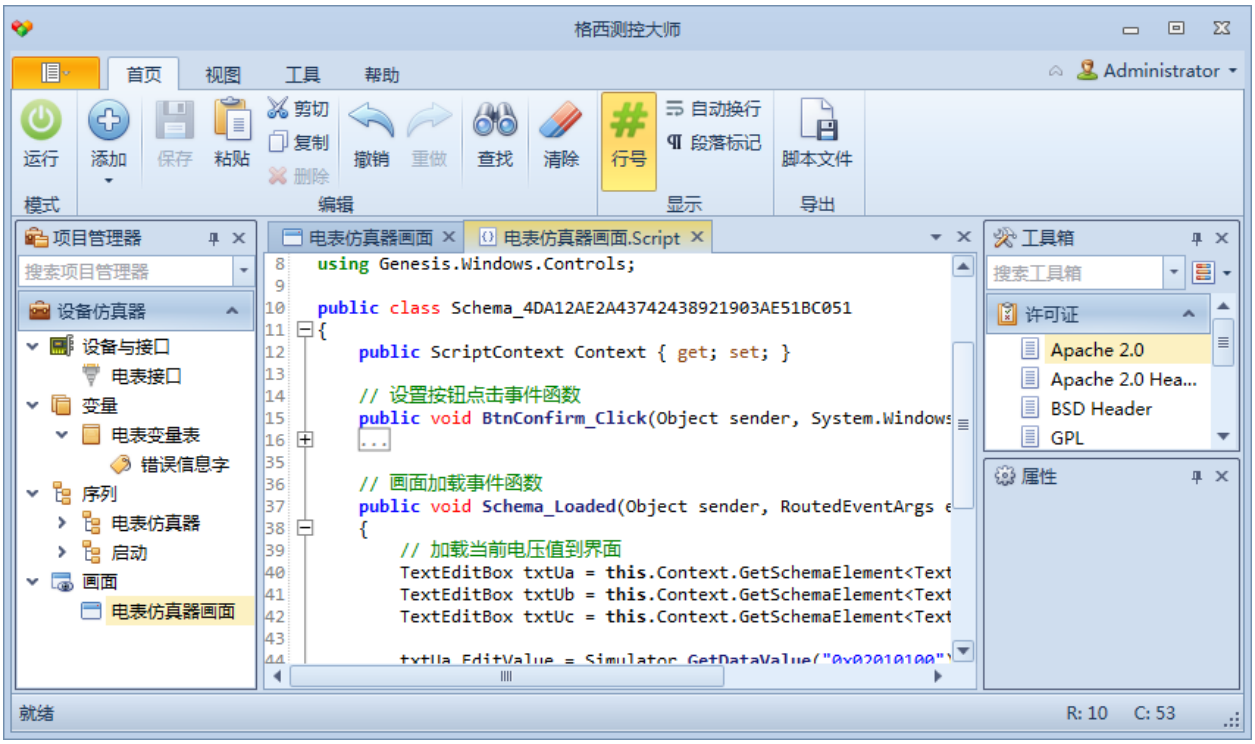
2.5 第 5 步 添加电表仿真器设置画面

本例子建立一个画面——“电表仿真器画面”，用于设置电表仿真器的参数。

画面的输入控件采用易于格式化输入的 TextEditBox 控件，格式设置使用了 Mask 和 MaskType 两个属性。



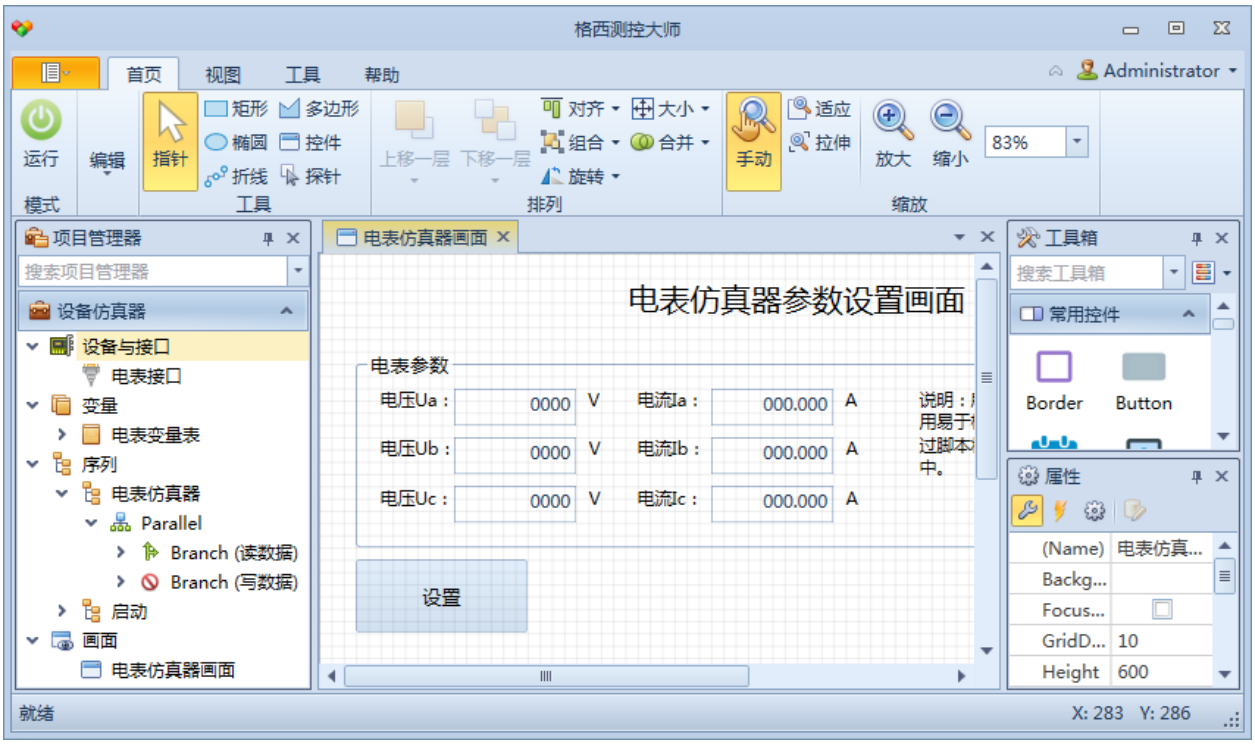
画面的脚本处理“设置”按钮的点击事件和画面的加载事件。



3. 运行项目

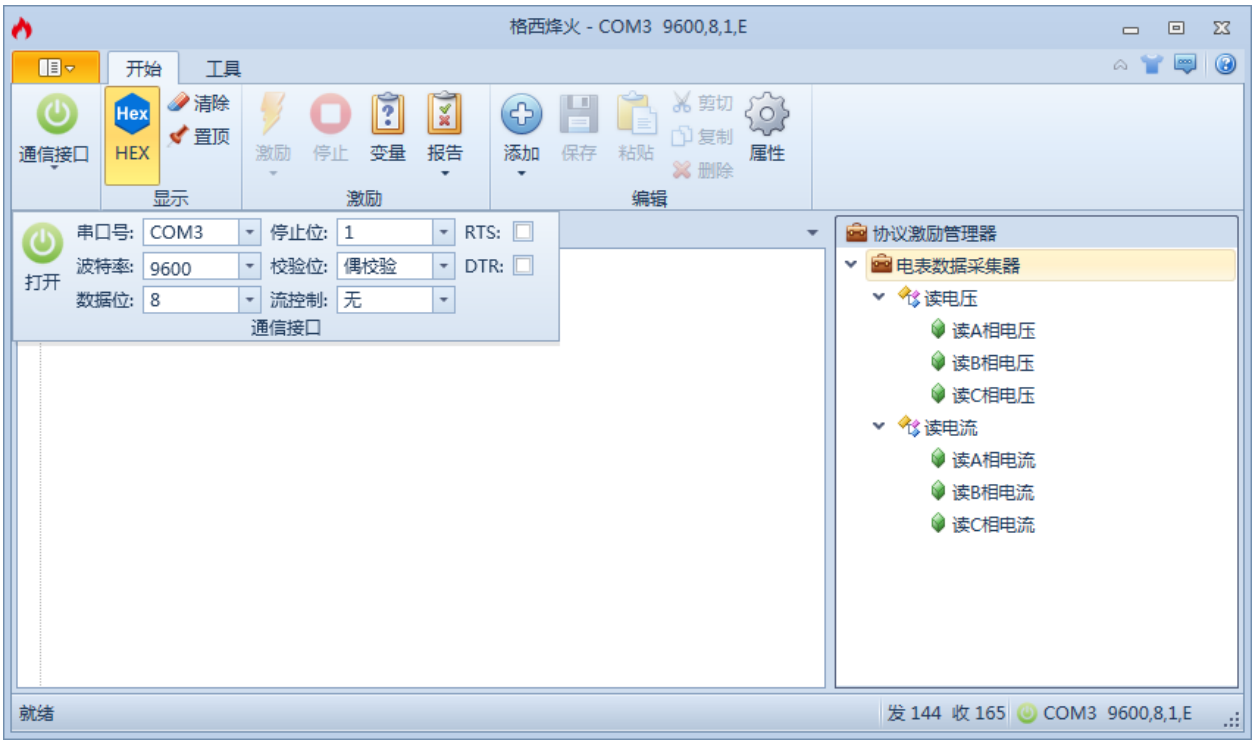
3.1 打开电表仿真器项目

从<软件安装目录>\Examples\Solutions\DeviceSimulation\DeviceSimulator 目录中，打开 DeviceSimulator.gpj 串口版项目文件。



3.2 打开电表数据采集器项目

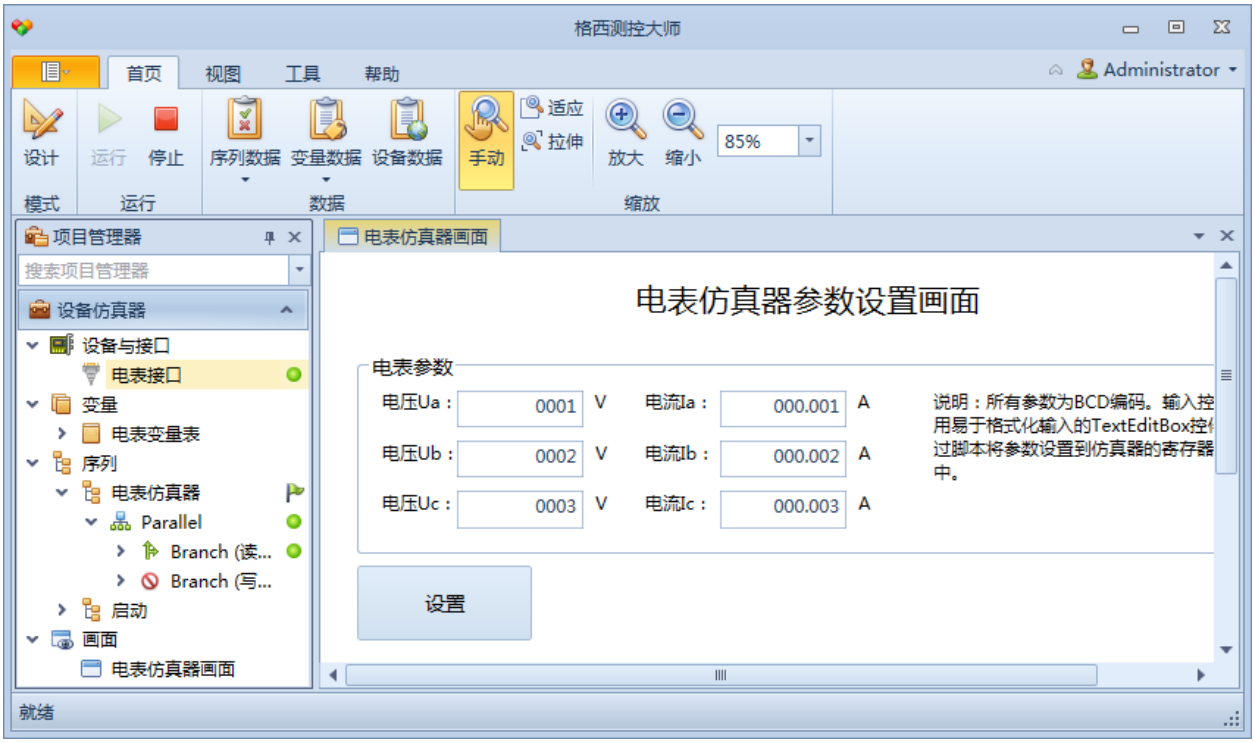
从<软件安装目录>\Examples\Solutions\DeviceSimulation\DeviceSimulator 目录中，用格西烽火软件打开 DeviceSimulatorMaster.bcp 项目文件。配置串口参数为 COM3，波特率 9600，数据位 8，停止位 1，偶校验。



3.3 运行项目

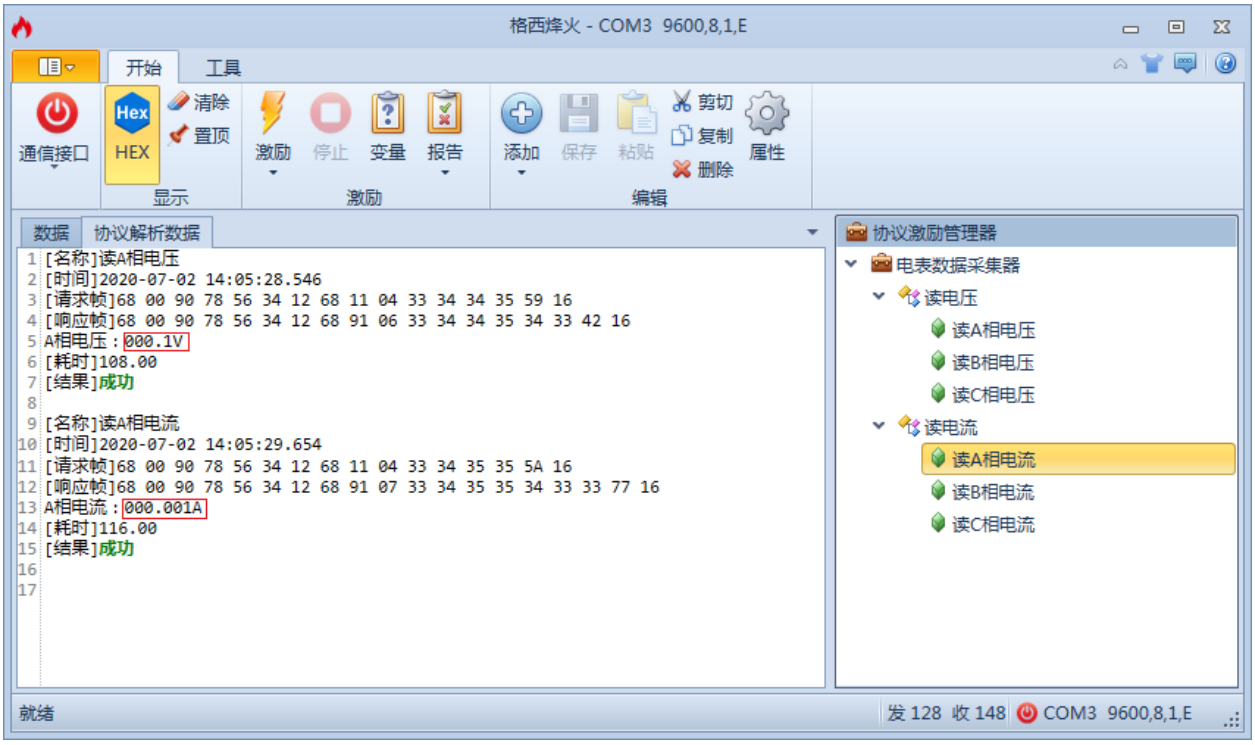
3.3.1 第 1 步 运行设备仿真项目

点击工具栏的“运行”按钮，进入运行模式。仿真项目自动运行“启动”序列，通过脚本打开电表串口，运行“电表仿真器”序列，打开“电表仿真器画面”，仿真环境准备就绪。



3.3.2 第 2 步 使用电表数据采集器采集数据

切换到格西烽火软件，在工具栏中点击“打开”按钮，打开串口，然后用鼠标双击右边面板中的协议命令条目，即可对电表仿真器进行数据抄读。



3.3.3 第 3 步 修改电表仿真器的参数并再次抄读

切换到格西测控大师软件，在“电表仿真器画面”，可以修改仿真器的参数值，修改后，可以切换到格西烽火软件，再次抄读测试。