上海格西信息科技有限公司

格西测控大师
用户手册

版本 2.5

目录

1. 产品简介

1.1 关于

格西测控大师是一款基于模块化技术的测控开发管理软件，可帮助用户快速开发自动化测试和控制软件系统，可帮助企业统一化、标准化测试和控制软件的开发和管理，减少企业在测试和控制软件方面的开发成本、学习成本和维护成本！

格西测控大师为自动化测试和控制的所有不同应用提供了统一环境与界面，为测控系统的开发、管理与执行提供了一个灵活而强大的框架，从而有效的解决四个关键领域的问题：

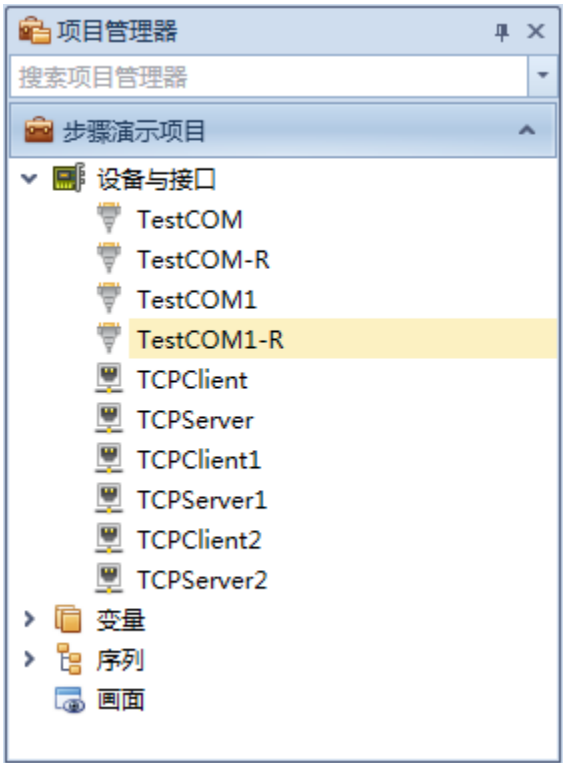
- 简化与加速复杂序列的开发
- 提高代码与测控程序的复用性和可维护性
- 提高测控系统的可扩展性
- 改进测控系统的执行性能

对于任何一个需要加速开发、代码复用、性能改进和自动化的测控项目，例如数据采集和监控系统、设计验证系统、硬件测试系统、芯片测试系统等，格西测控大师都是不可或缺的。

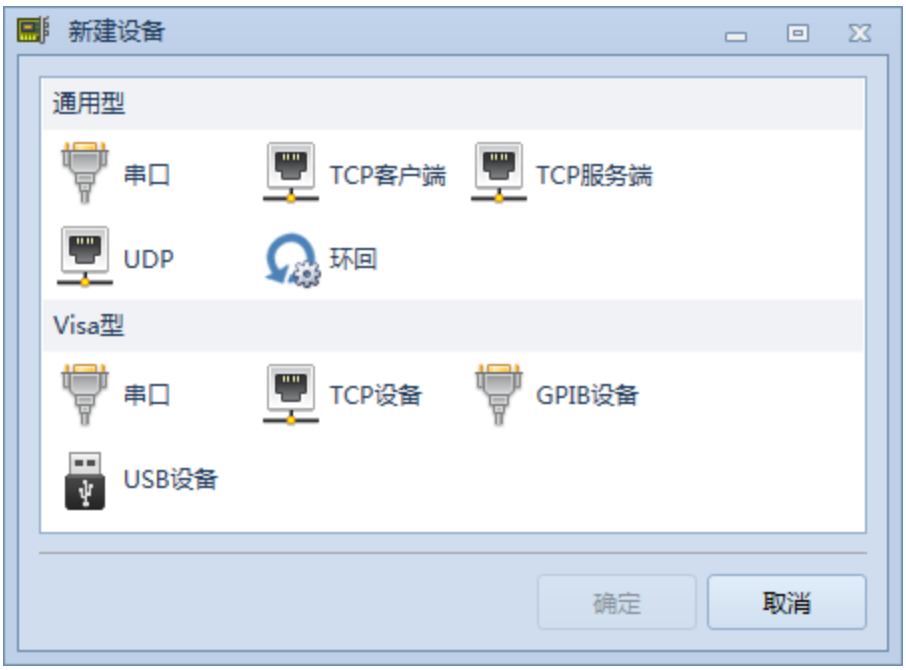
1.2 功能特性

1）自定义设备接口

借助软件内置的设备适配器，用户可以创建任意设备和接口组合，可以同时对不同的设备和接口进行通信，满足各种测控连接需求。

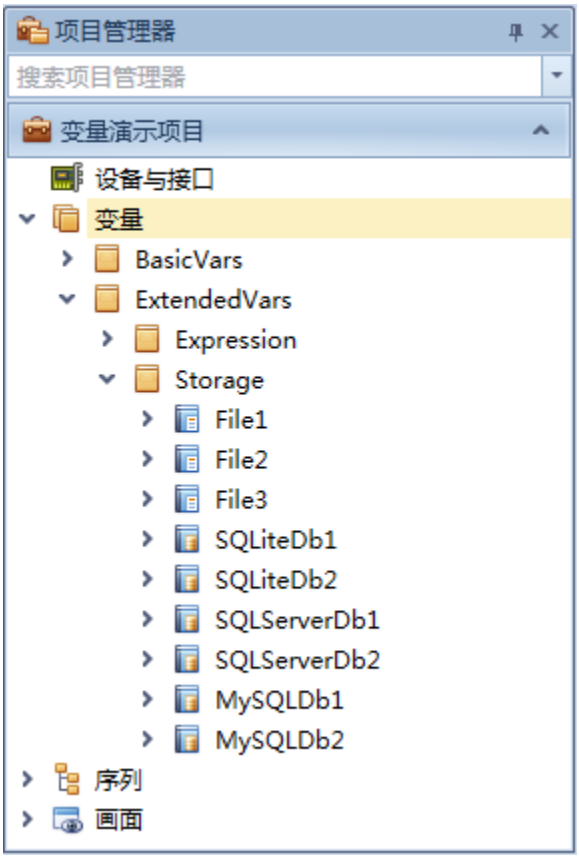


同时，设备适配器还可以通过插件的方式，让用户扩展新的设备和接口，例如监听型串口、监听型网口等，以满足特殊的连接需求。

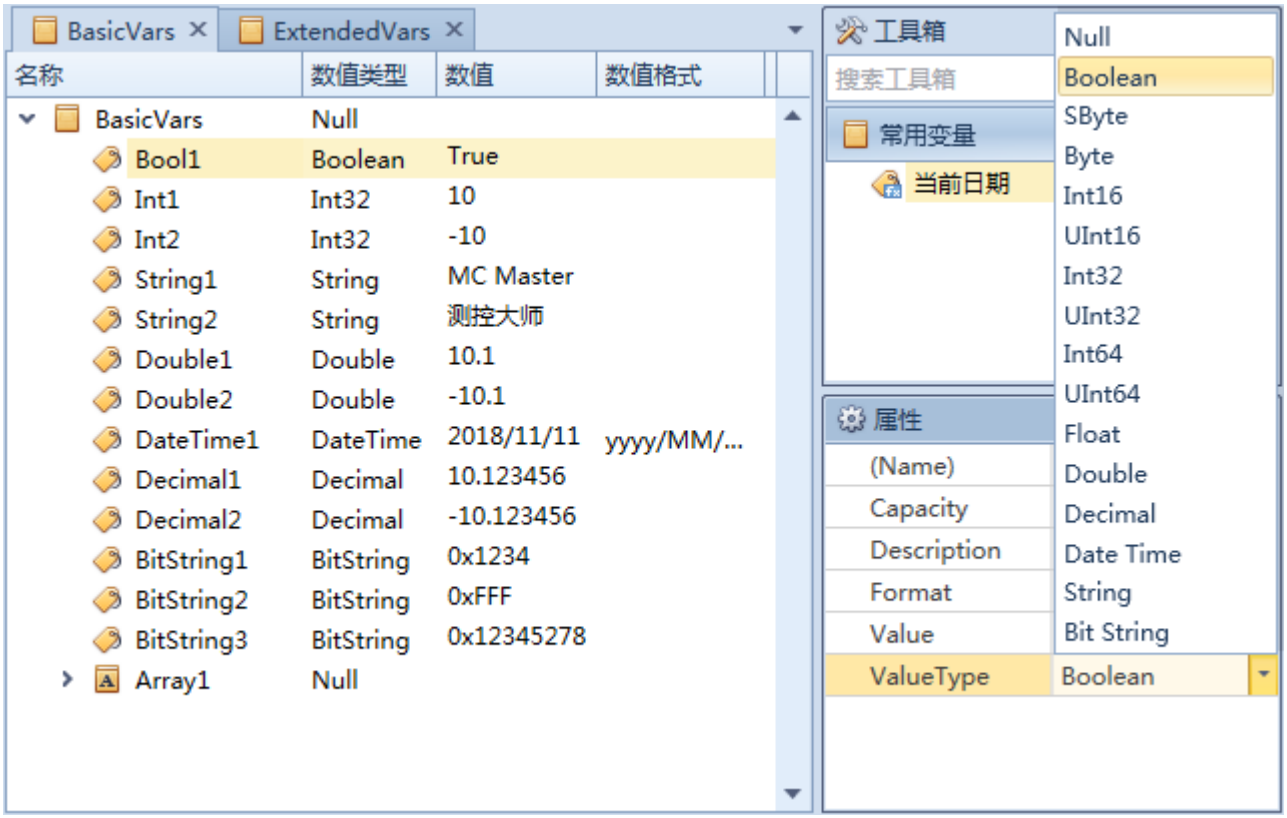


2）自定义变量数据

借助软件内置的变量适配器，用户可以创建变量、变量数组、变量容器，还可以创建扩展变量，如表达式变量、文本文件变量、数据库变量等，满足各种测控数据传递、呈现和存储需求。



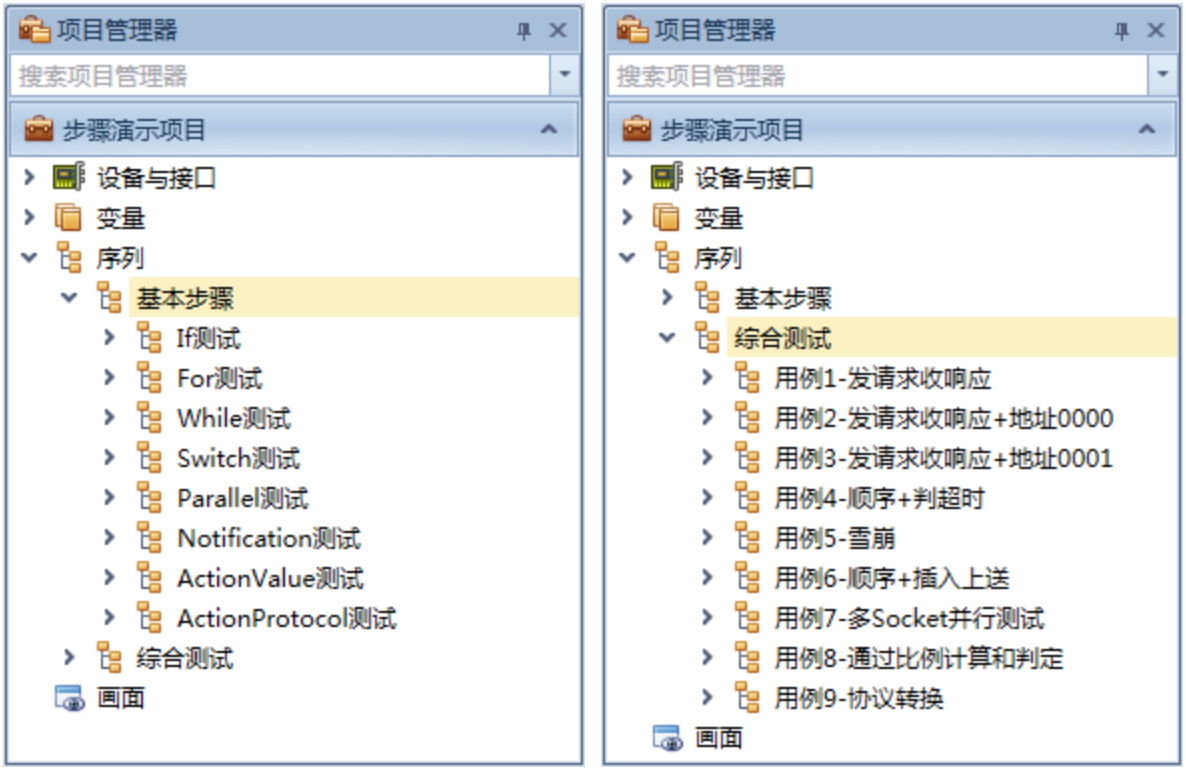
软件内置变量编辑器，用户可以很方便的创建和编辑各种类型的变量。



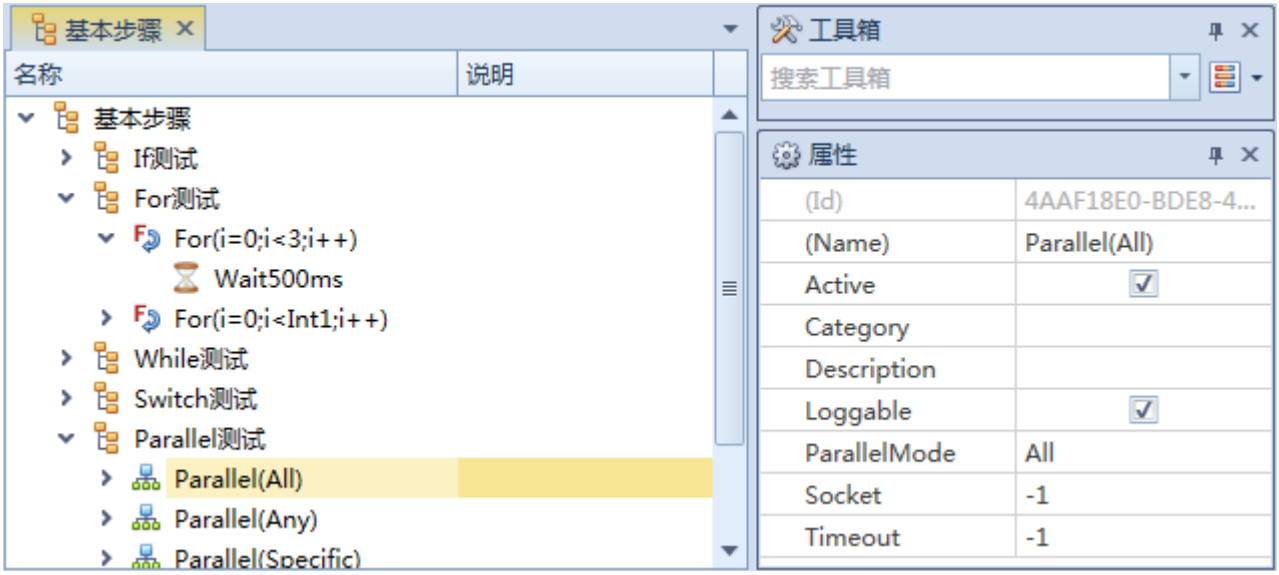
3）自定义执行序列

借助软件内置的序列适配器，用户可以创建执行序列，实现任意逻辑的执行过程，满足各种测控自动化需求。

- 支持流程控制，如分支语句 If、Switch，循环语句 For、While，并行语句 Parallel。
- 支持同步控制，如等待（Wait）、通知（Notification）。
- 支持数值类型动作步骤（Value）、协议类型动作步骤（Protocol）等。
- 支持序列嵌套，支持复杂的层次结构。
- 支持脚本，脚本可以无缝调用.Net Framework 类库，调用第三方托管库来实现执行逻辑。
- 支持协议模板和步骤模板。

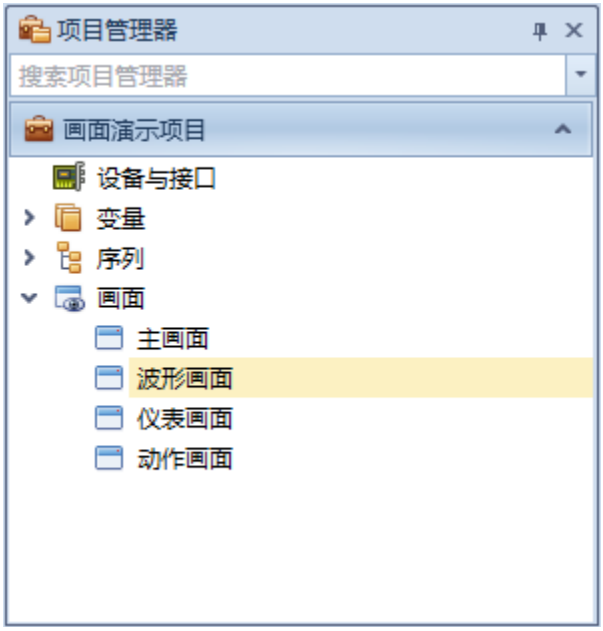


软件内置序列编辑器，用户可以很方便的创建和编辑各种类型的序列步骤。



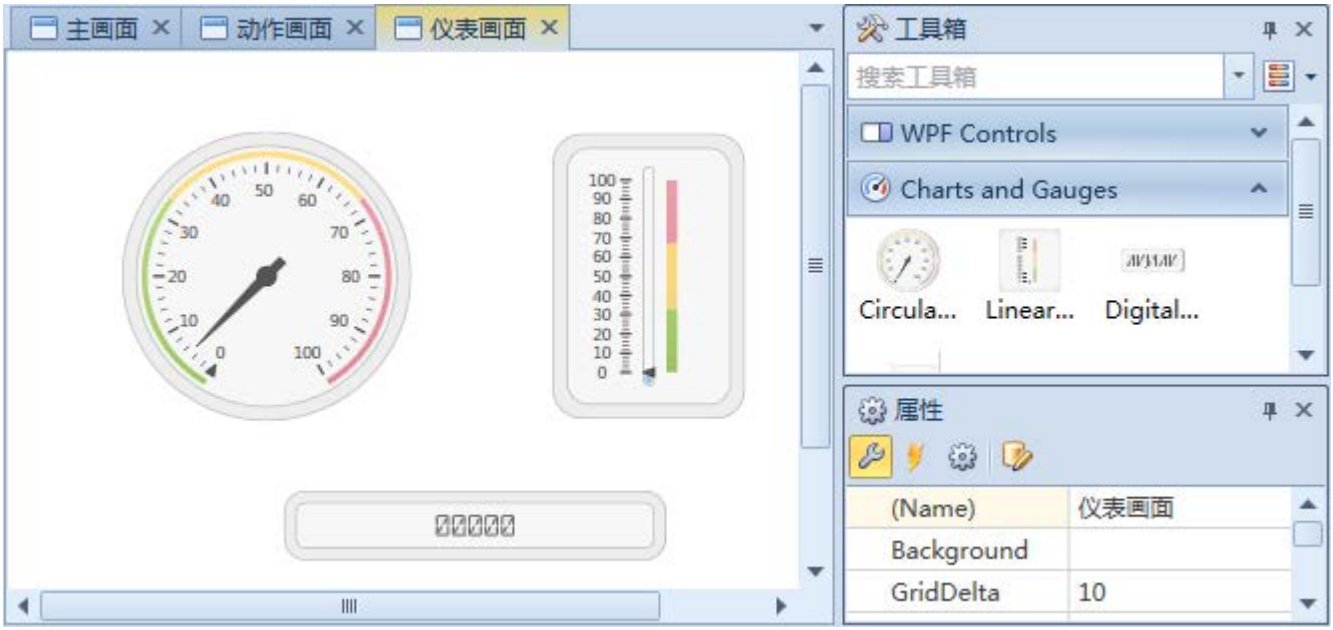
4）自定义用户界面

借助软件内置的画面适配器，用户可以创建画面，利用画面工具箱的控件、形状模版，实现任意用户界面，满足各种测控界面需求。



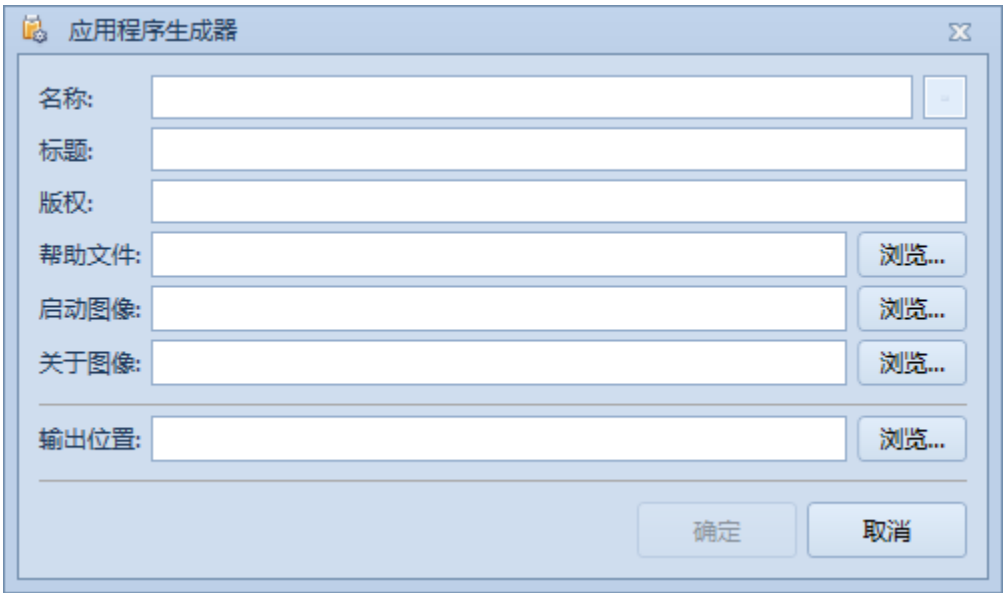
软件内置画面编辑器，用户可以很方便的创建和编辑画面，实现画面逻辑。

- 支持属性数据绑定，建立画面元素属性和变量的联系。
- 支持事件脚本，脚本可以无缝调用.Net Framework 类库，调用第三方托管库来实现画面逻辑。
- 支持动态动作，建立画面元素动作（移动、旋转、尺寸）与变量的联系。
- 支持控件模板和形状模版。



5）生成应用程序

用户在开发完成测控程序之后，可以创建运行时应用程序，自定义软件名称、标题、版权、帮助文件、启动图像、关于图像等软件信息，满足各种测控程序的部署需求。



1.3 系统要求

支持的操作系统：

- Windows Vista SP2 (x86 和 x64)
- Windows 7 SP1 (x86 和 x64)
- Windows 8 (x86 和 x64)
- Windows 8.1 (x86 和 x64)
- Windows Server 2008 SP2 (x86 和 x64)
- Windows Server 2008 R2 SP1 (x64)
- Windows Server 2012 (x64)
- Windows Server 2012 R2 (x64)
- Windows 10 (x86 和 x64)

硬件要求：

- 建议的最低要求： 1 GHz 或更快，1 GB RAM 或更大
- 最小磁盘空间： 300 MB

必备组件：

- Microsoft .NET Framework 4.8

1.4 产品许可

本产品采用硬件加密锁的方式进行许可。

许可细则：

- 一把加密锁同一时间只能授权一台电脑；

- 授权的电脑可以同时运行本产品多个实例；
- 加密锁不支持虚拟机授权。

1.5 产品支持

您在使用本软件的过程中遇到问题或者希望获得产品的支持信息，可以通过我们的网站、电子邮件等方式与我们联系。

- 公司网站：www.geshe.com
- 电子邮件：support@geshe.com
- QQ：979464

2. 快速入门

2.1 安装

欢迎安装格西测控大师!

第 1 步 - 确保计算机支持格西测控大师
开始安装之前，请查看 1.3 节的系统要求，了解计算机系统是否支持格西测控大师。

第 2 步 - 下载格西测控大师
接下来，从官方网站 www.geshe.com 下载格西测控大师安装文件。

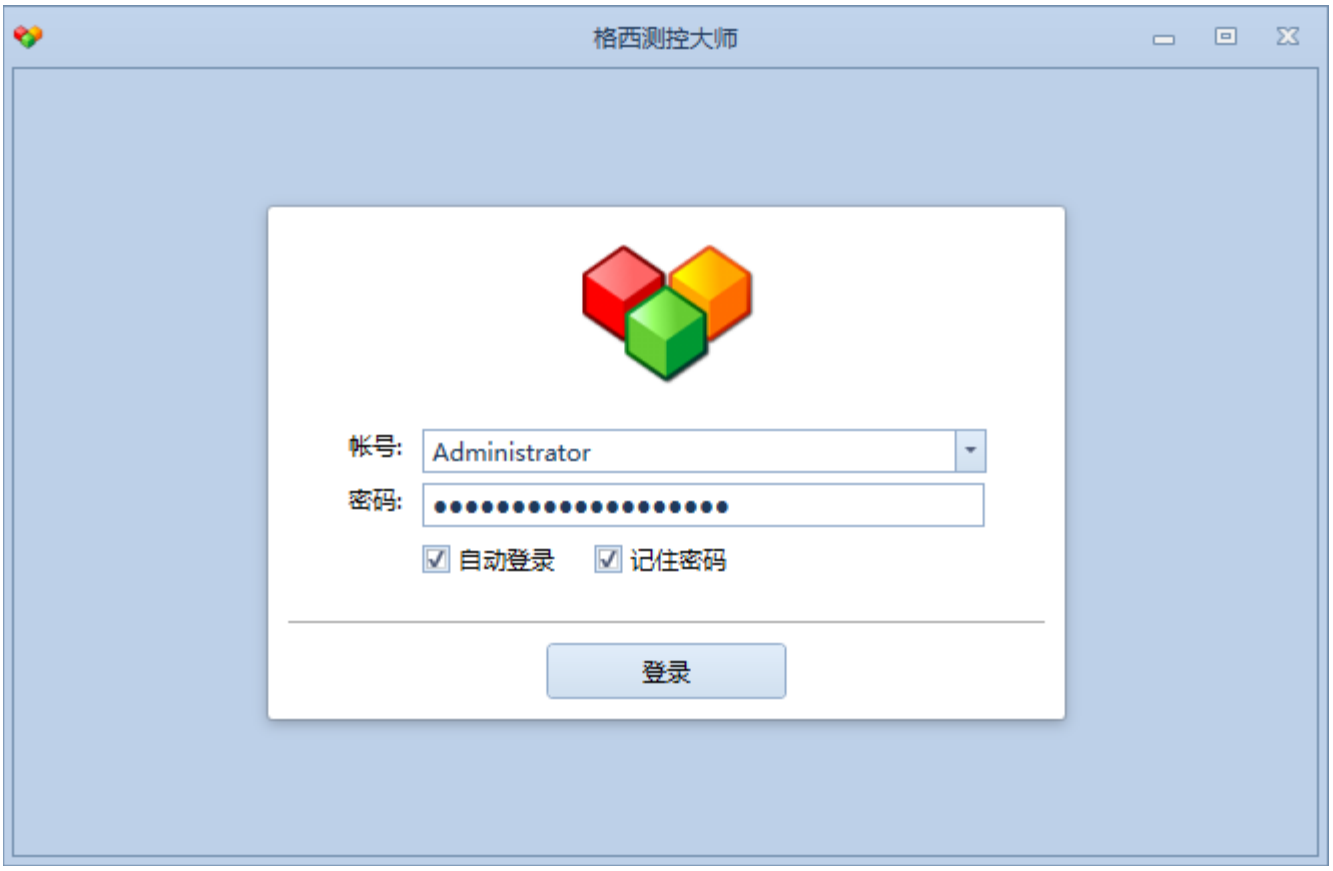
第 3 步 – 执行安装文件
接下来，运行格西测控大师安装文件进行安装。
安装程序依次安装以下程序：

- Microsoft .Net Framework 4.8（已安装则跳过）
- 格西测控大师

第 4 步 – 开始使用
在格西测控大师安装完成后，点击“开始使用”按钮，开始使用格西测控大师进行开发。

2.2 登录

使用格西测控大师之前需要登录账户，软件启动后的登录界面如下图所示。



软件内置的初始账户列表如下，不同的账户有不同的操作权限。

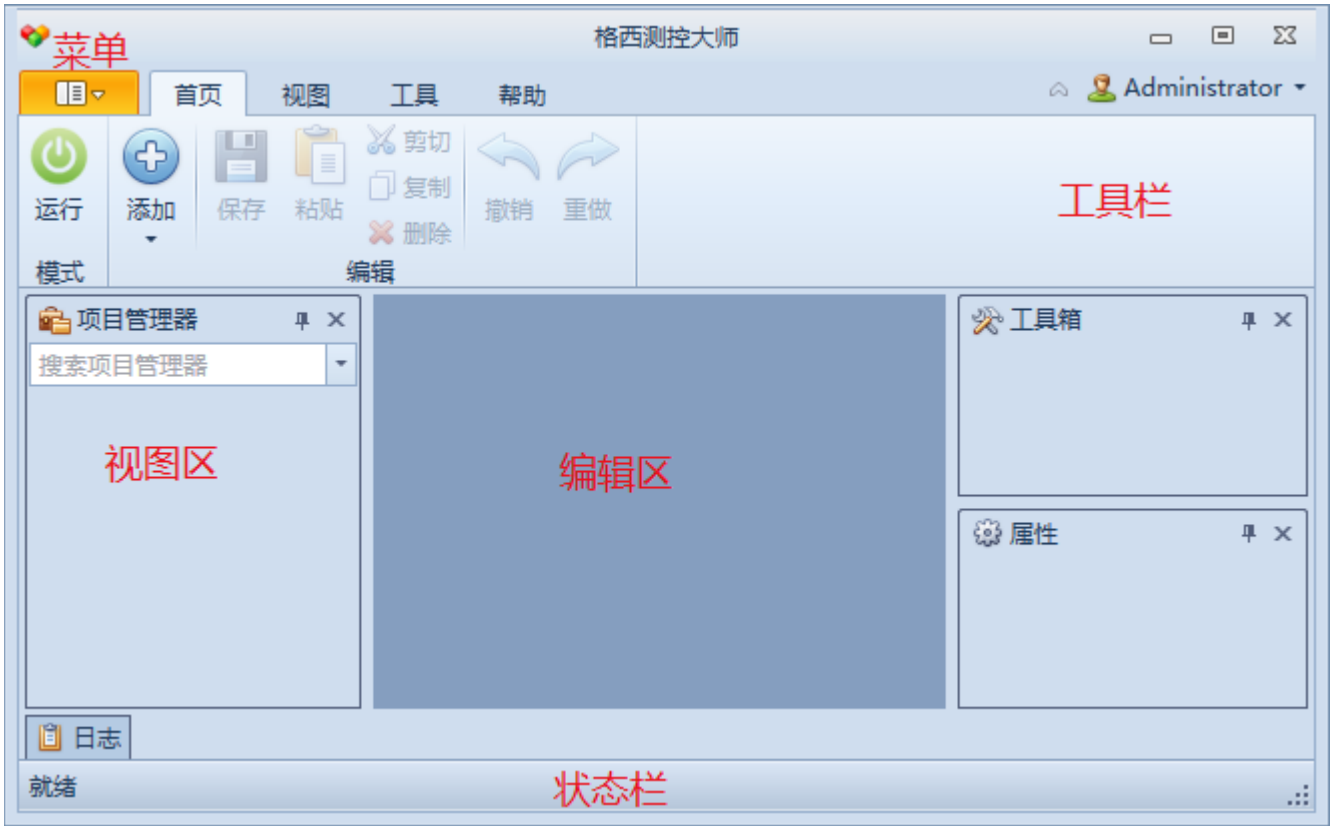
用户名	密码	描述
Administrator	（无）	管理员，拥有所有权限。
Developer	（无）	开发员，拥有除“用户管理”外的所有权限。
Operator	（无）	操作员，只有项目运行权限。

注意：用户名的字母是大小写敏感的，例如 Administrator 的首字母是大写 A。

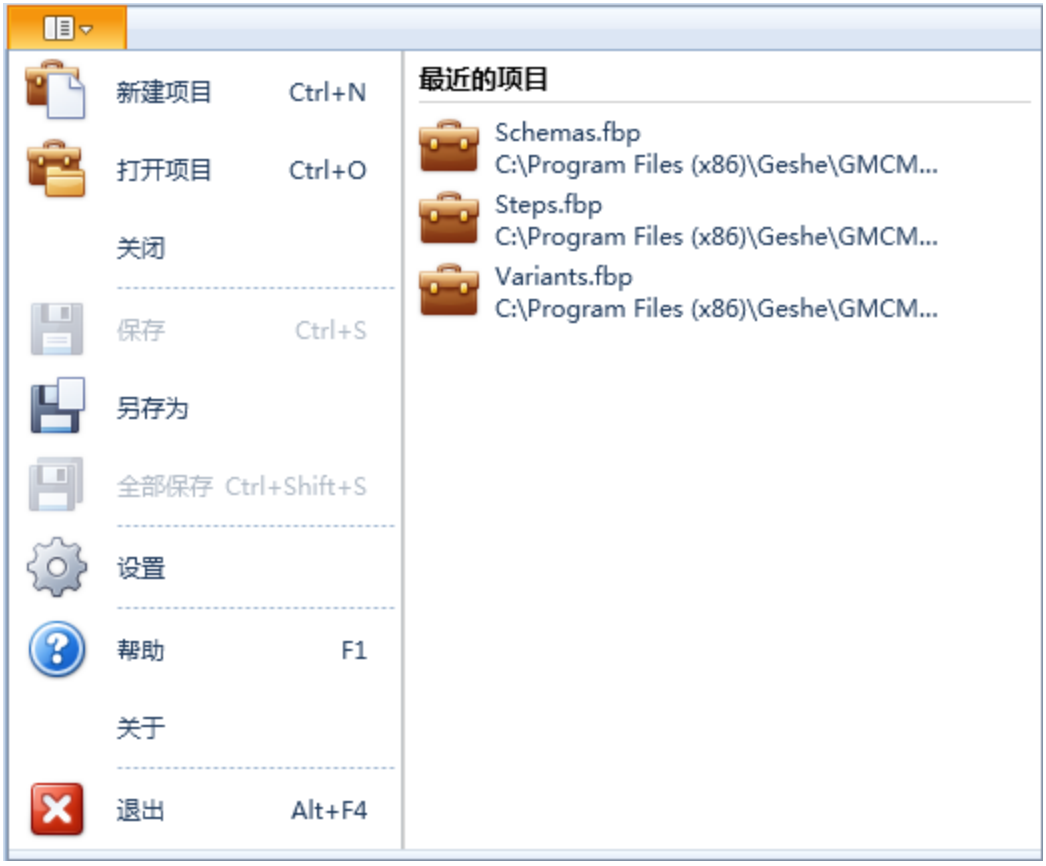
2.3 用户界面

2.3.1 主界面

格西测控大师登录后的主界面如下图所示，左上角为“菜单”按钮，可以弹出应用程序菜单，往下依次是工具栏、视图区、编辑区和状态栏。



2.3.2 应用程序菜单



命令	快捷键	功能
新建项目	Ctrl+N	新建一个项目，并建立项目环境。
打开项目	Ctrl+O	打开一个项目，并建立项目环境。
关闭		关闭当前激活的项目。
保存	Ctrl+S	保存当前激活的项目。
另存为		将当前激活的项目保存到指定路径。
全部保存	Ctrl+Shift+S	保存当前打开的所有项目。
设置		设置系统参数。
帮助	F1	用户帮助。
关于		显示软件的版权、版本以及注册信息等。
退出	Alt+F4	退出系统。

2.3.3 工具栏

首页工具栏

首页工具栏是用户操作的主要工具栏，包含基本命令按钮和当前激活编辑器的命令按钮。

命令	功能
运行/设计	切换运行态或设计态
添加	子菜单包含当前编辑状态可以添加的条目。
折叠工具栏（右上角第 1 个按钮）	显示/折叠工具栏。
登录用户（右上角第 2 个按钮）	登录用户操作菜单。

视图工具栏

命令	功能
项目管理器	打开项目管理器
用户管理器	打开用户管理器，用于进行用户权限管理。
工具箱	打开工具箱
属性	打开属性视图
日志	打开日志视图
换肤	更换皮肤
重置窗口	重置窗口排列为缺省值
置顶	主窗口置顶
全屏	全屏显示

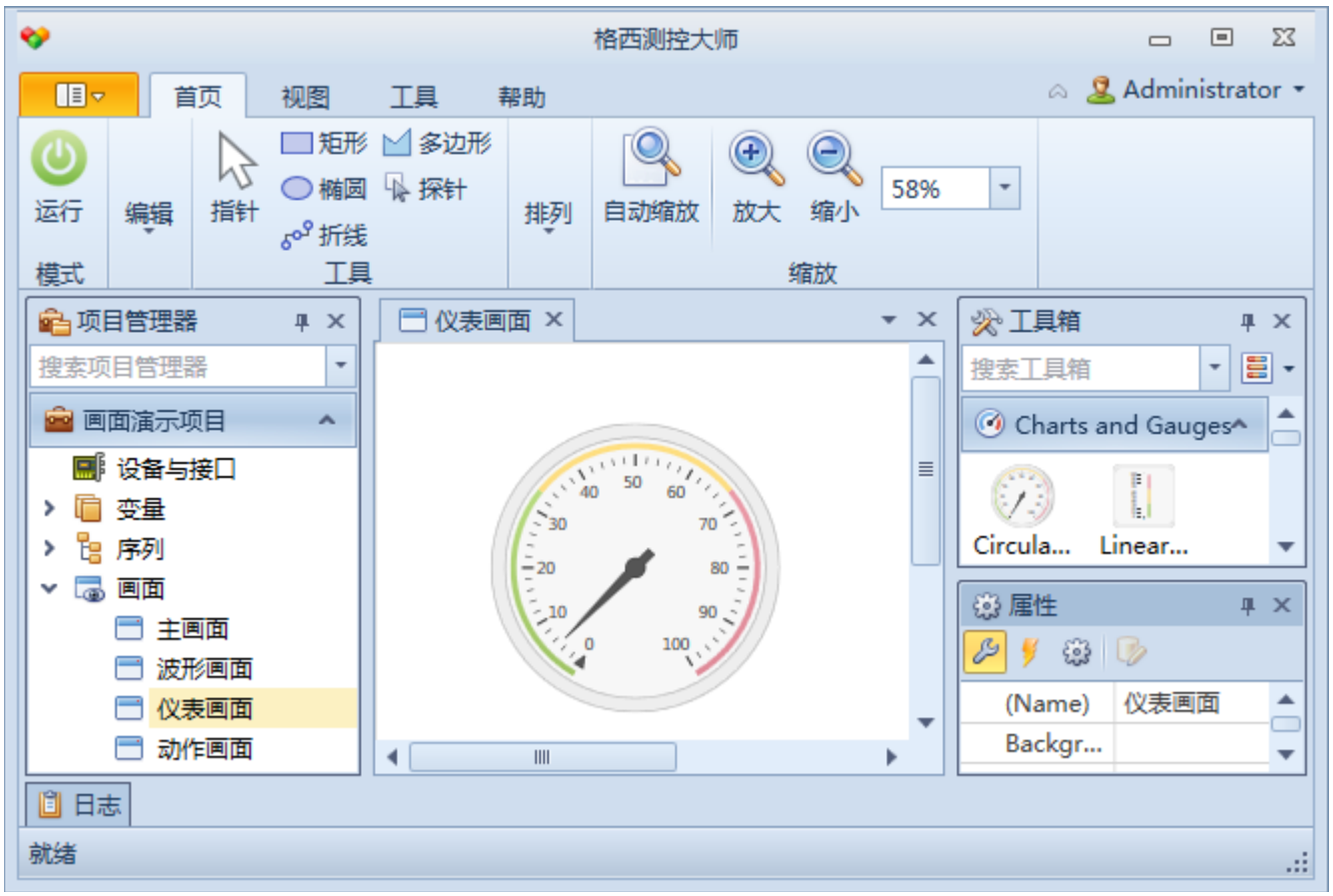
工具工具栏

命令	功能
计算器	运行系统提供的计算器程序。
校验和	运行工具箱的校验和计算器。
CRC	运行工具箱的 CRC 计算器。
DES	运行工具箱的 DES 计算器。
哈希值	运行工具箱的哈希值计算器。
Base64	运行工具箱的 Base64 字符串计算器。
ASCII 码对照表	运行工具箱的 ASCII 码对照表。
Unicode 转换器	运行工具箱的 Unicode 转换器。

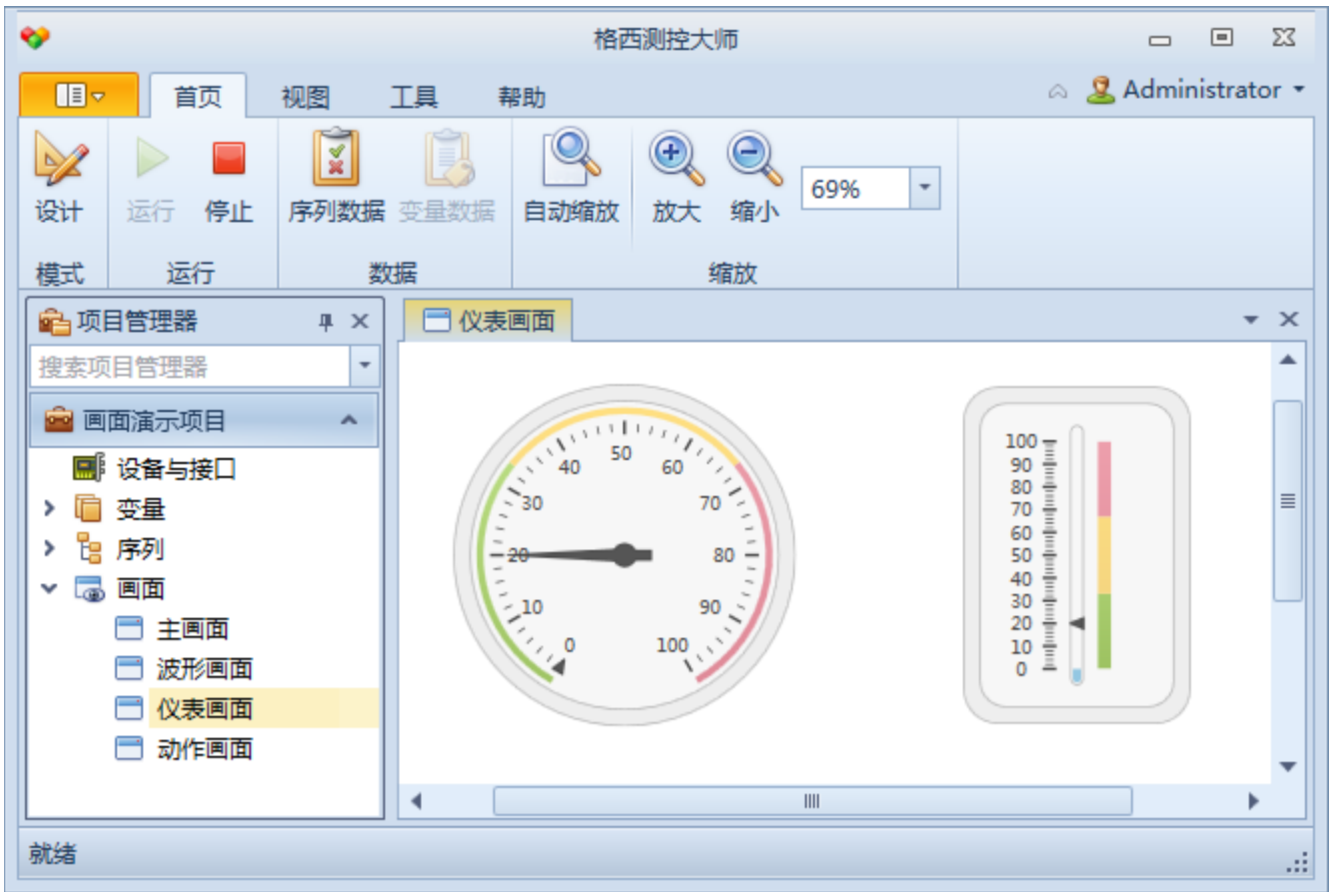
2.3.4 编辑区

编辑区在设计状态下可以用来编辑项目条目，如序列、变量、画面等，在运行状态则可以用来显示项目运行数据、运行状态和进行用户交互。

在设计状态下，当打开需要编辑的项目条目时，对应的编辑器显示在编辑区，对应的工具命令按钮显示在首页工具栏，对应的模版显示在工具箱，对应的属性集显示在属性视图。如下图所示，打开一个画面进行编辑，首页工具栏增加了许多画面编辑命令按钮，工具箱加载了最近编辑的画面元素模版，属性视图加载了当前选中的画面元素的属性集。



在运行状态下，当打开需要显示的项目条目时，对应的页面显示在编辑区，对应的工具命令按钮显示在首页工具栏。如下图所示，运行一个画面。



2.4 创建一个数据采集与监控项目

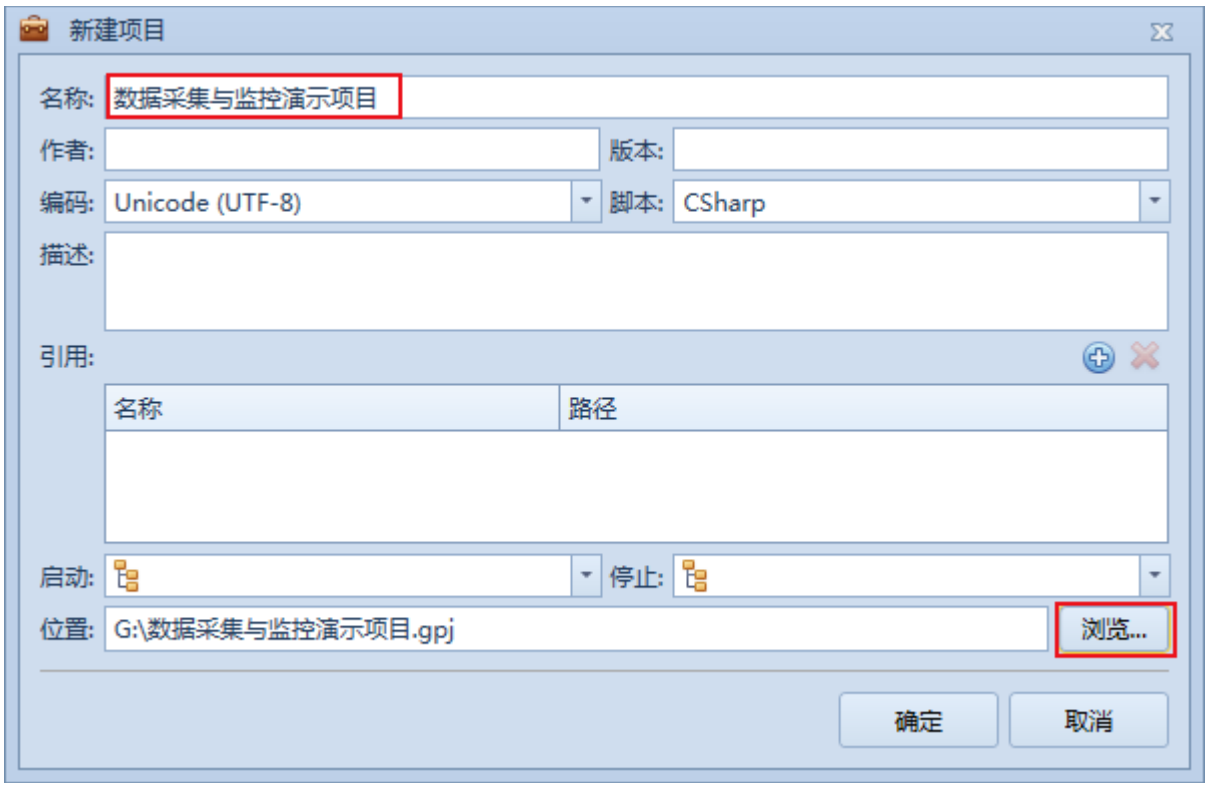
某热电偶采集模块，采用 Modbus RTU 通信协议，其中读温度命令为 03 命令，地址为 0000，温度值为 16 位有符号整数。

本项目演示读温度，然后把采集时间和温度数据保存到文本文件型变量“温度数据”中，最后用曲线图显示温度随时间变化过程。

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

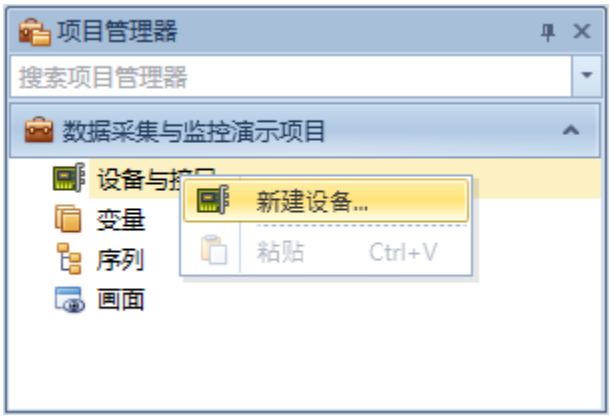
2.4.1 第 1 步 新建项目

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“数据采集与监控演示项目”，然后点击“浏览...”按钮，选择保存路径和填写项目文件名“数据采集与监控演示项目”，最后点击“确定”按钮。

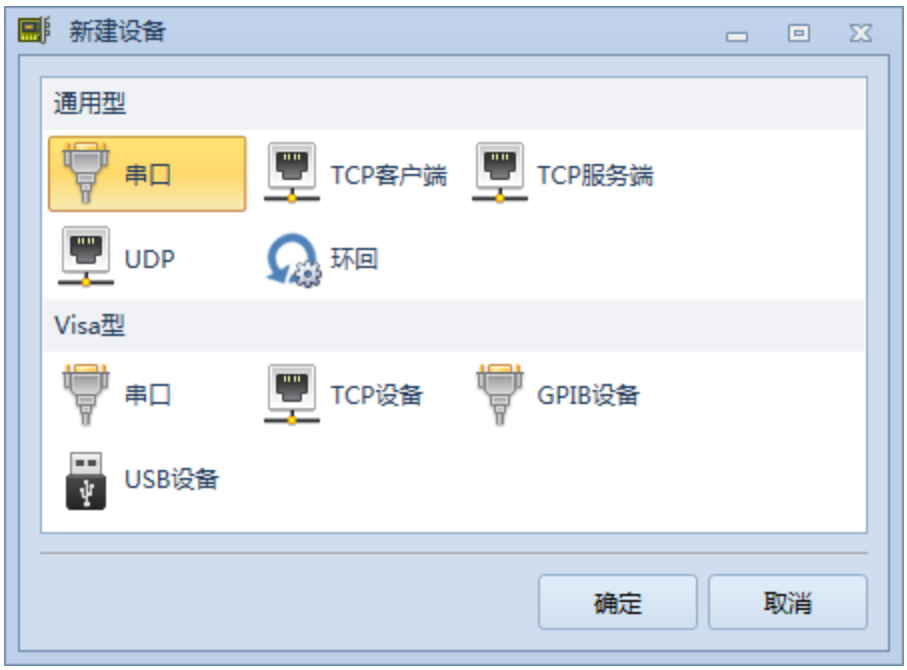


2.4.2 第 2 步 添加串口设备

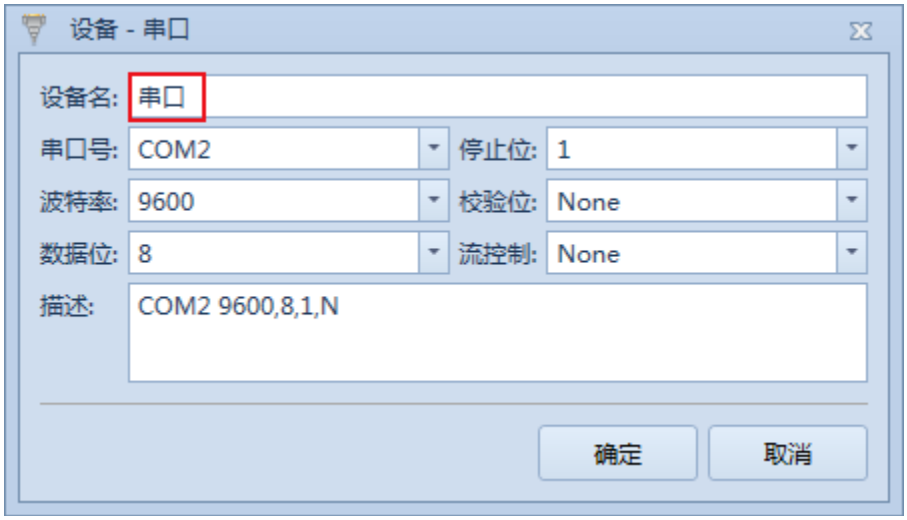
在项目管理器中选择“设备与接口”节点，然后点击鼠标右键，在弹出菜单中选择“新建设备...”。



弹出新建设备对话框中，选择“串口”，点击“确定”。

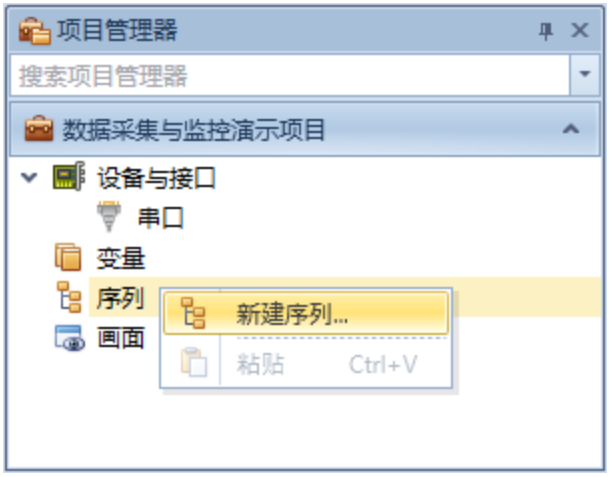


弹出设备属性对话框，填写“设备名”和其他设备参数，最后点击“确定”按钮。其中，“设备名”是设备的标识，可以是任意字符串，引用设备必须使用设备名。

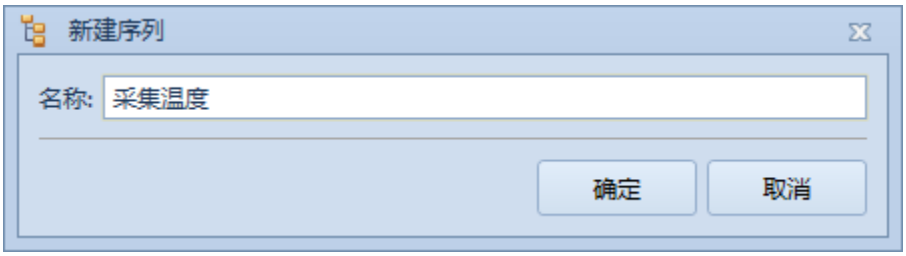


2.4.3 第 3 步 添加序列

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

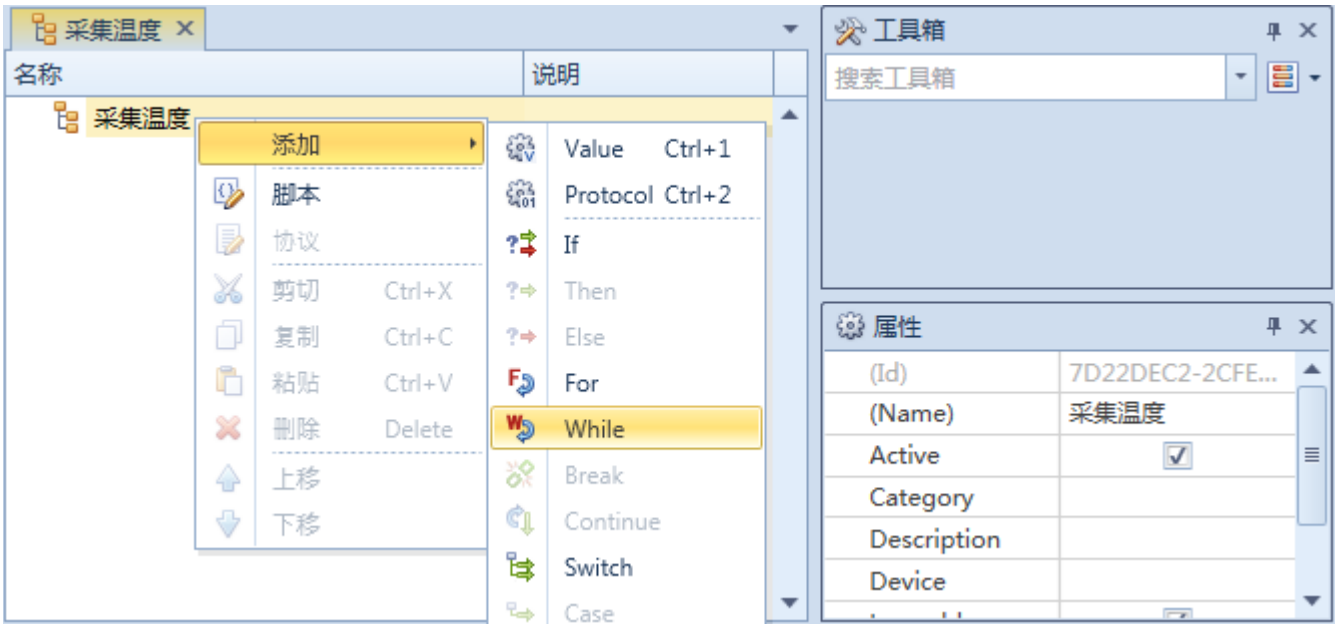


弹出新建序列对话框中，填写“名称”，点击“确定”。

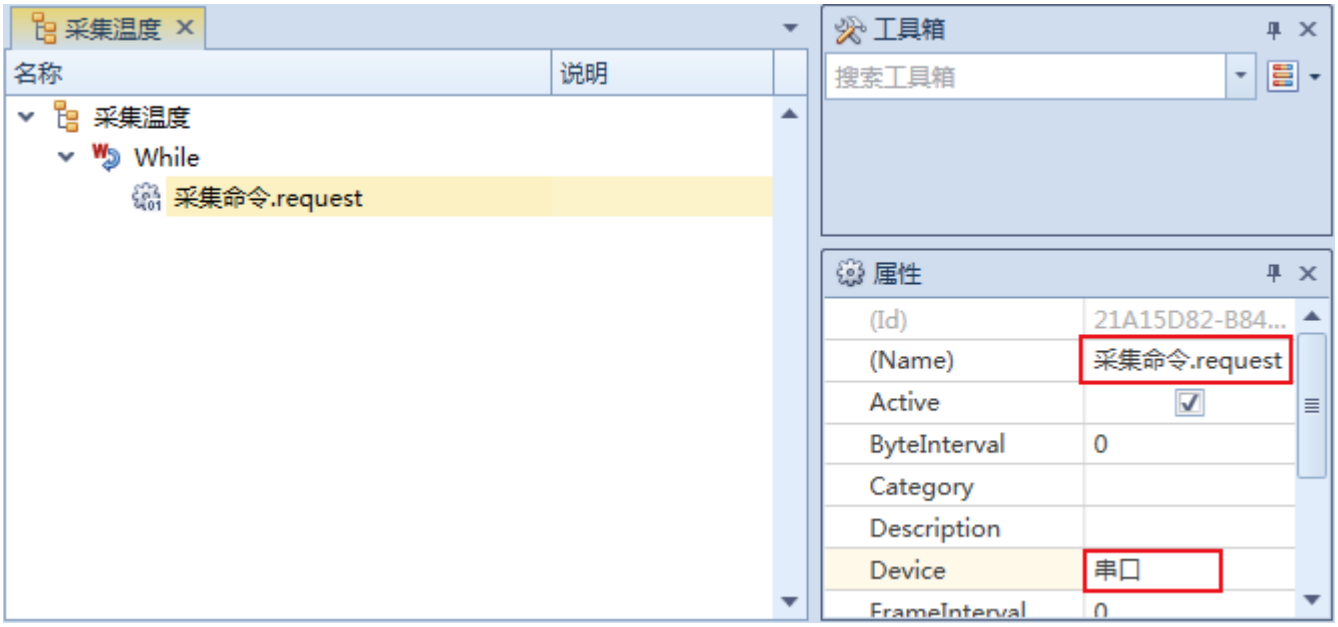


在项目管理器中选择新创建的“采集温度”节点，然后双击鼠标左键，或者点击鼠标右键，在弹出菜单中选择“编辑...”，打开序列编辑页面。

在“采集温度”编辑页面，选中“采集温度”节点，单击鼠标右键，在弹出菜单中选择“添加 -> While”，添加 While 类型步骤，条件参数 ConditionExpression 设置为 True，无限循环执行采集任务。



接下来，添加发送采集命令，选中“While”节点，单击鼠标右键，在弹出菜单中选择“添加 -> Protocol”，添加协议类型步骤，然后在属性面板，修改步骤名称“(Name)”为“采集命令.request”，TransceiveMode 设置为 Send，表示主动发送，Device 属性选择名称为“串口”的设备。

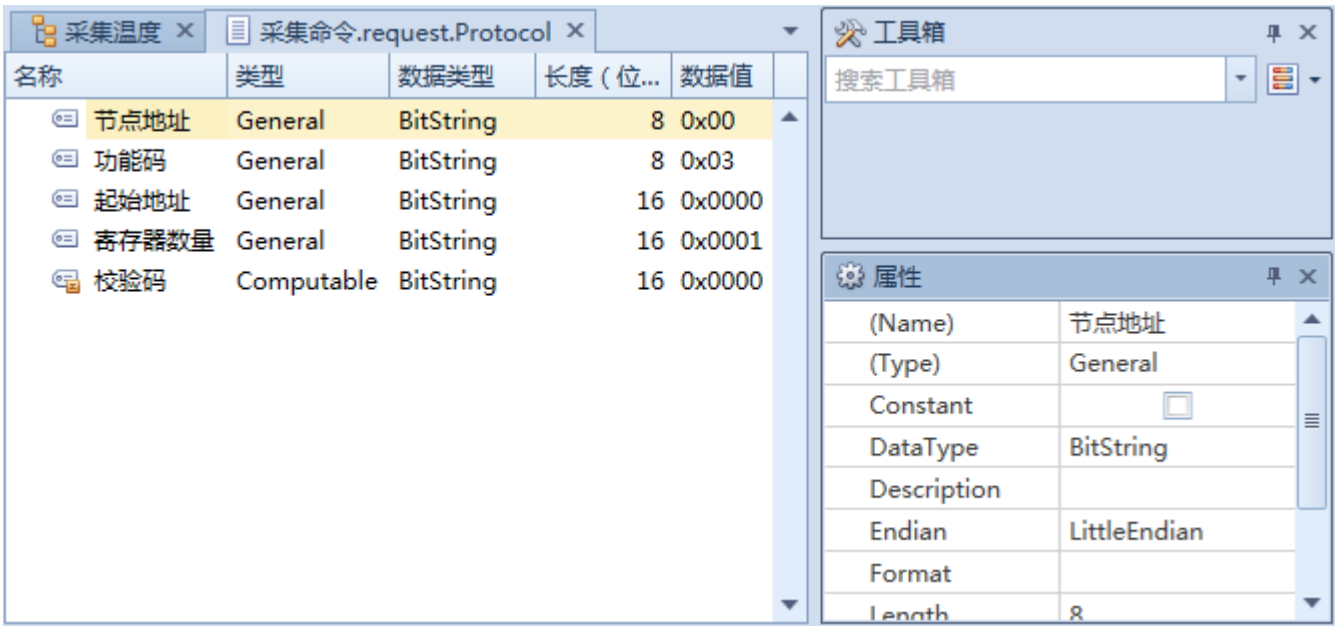


接下来，编辑发送采集命令的协议数据，选中“采集命令.request”节点，单击工具栏的“协议”按钮，打开协议编辑界面，通过单击鼠标右键，在弹出菜单中选择“添加 -> 协议字段”，依次按以下属性表添加协议字段。

(Name)	(Type)	Constant	DataType	Endian	Length	Value
节点地址	General		BitString	LittleEndian	8	0x00
功能码	General	√	BitString	LittleEndian	8	0x03
起始地址	General	√	BitString	BigEndian	16	0x0000
寄存器数量	General		BitString	BigEndian	16	0x0001
校验码	Computable	√	BitString	LittleEndian	16	0x0000

其中，校验码的参数配置如下表。

属性	值	描述
Algorithm	CRC16MODBUS	计算的算法
Priority	1	计算优先级，在有多个计算型字段时有用。
Location	Back	表示计算型字段在需要计算的数据后面。
StartPosition	0	起始计算字节序号
EndPosition	-1	结束计算字节序号，-1 表示计算到该字段前面。



接下来，添加接收采集数据命令，选中“While”节点，单击鼠标右键，在弹出菜单中选择“添加 -> Protocol”，然后在属性面板，修改步骤名称“(Name)”为“采集命令.response”，TransceiveMode 设置为 Receive，表示接收，Device 属性选择名称为“串口”的设备。

接下来，编辑接收采集数据命令的协议数据，选中“采集命令.response”节点，单击工具栏的“协议”按钮，打开协议编辑界面，通过单击鼠标右键，在弹出菜单中选择“添加 -> 协议字段”，依次按以下属性表添加协议字段。

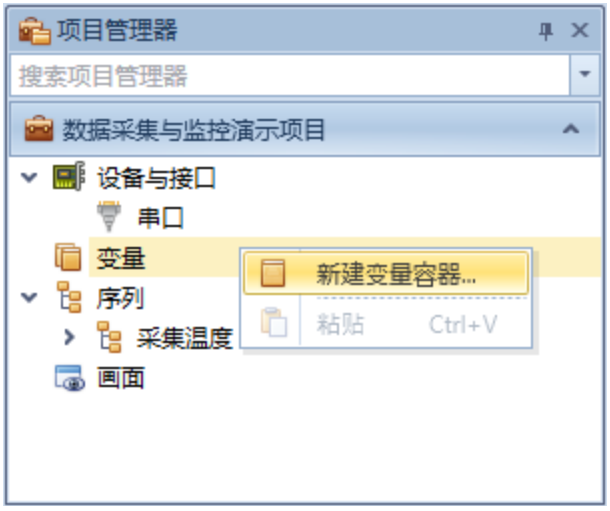
(Name)	(Type)	Constant	DataType	Endian	Length	Value
节点地址	General		BitString	LittleEndian	8	0x00
功能码	General	√	BitString	LittleEndian	8	0x03
字节数	General	√	BitString	LittleEndian	8	0x02
温度	General		Int16	BigEndian	16	0
校验码	Computable	√	BitString	LittleEndian	16	0x0000

其中，校验码的参数配置如下表。

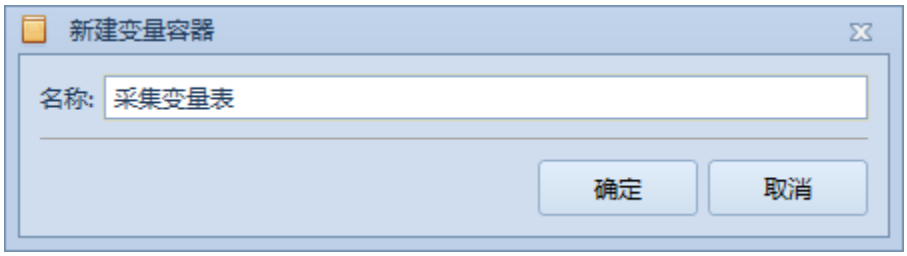
Algorithm	Priority	Location	StartPosition	EndPosition
CRC16MODBUS	1	Back	0	-1

2.4.4 第 4 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...”。

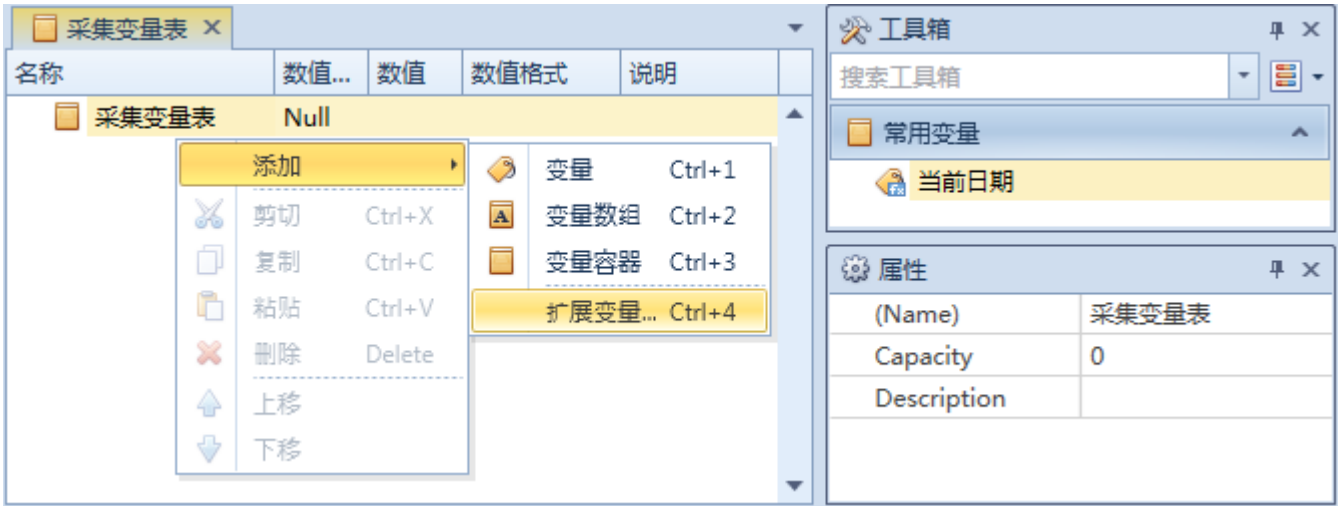


弹出新建变量容器对话框中，填写“名称”，点击“确定”。

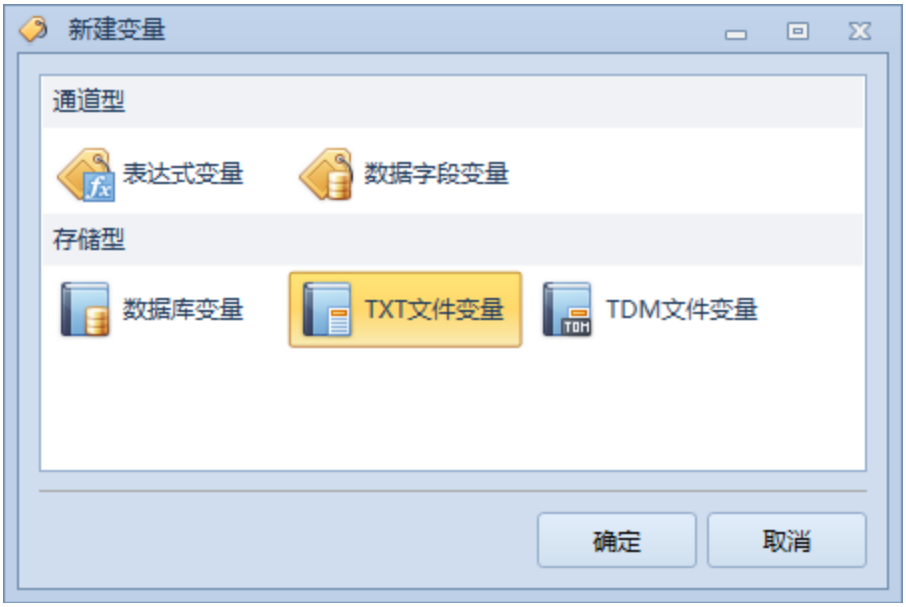


在项目管理器中选择新创建的“采集变量表”节点，然后双击鼠标左键，或者点击鼠标右键，在弹出菜单中选择“编辑...”，打开变量编辑页面。

在“采集变量表”编辑页面，选中“采集变量表”节点，单击鼠标右键，在弹出菜单中选择“添加 -> 扩展变量...”。



弹出新建变量对话框中，选择“TXT 文件变量”，点击“确定”。



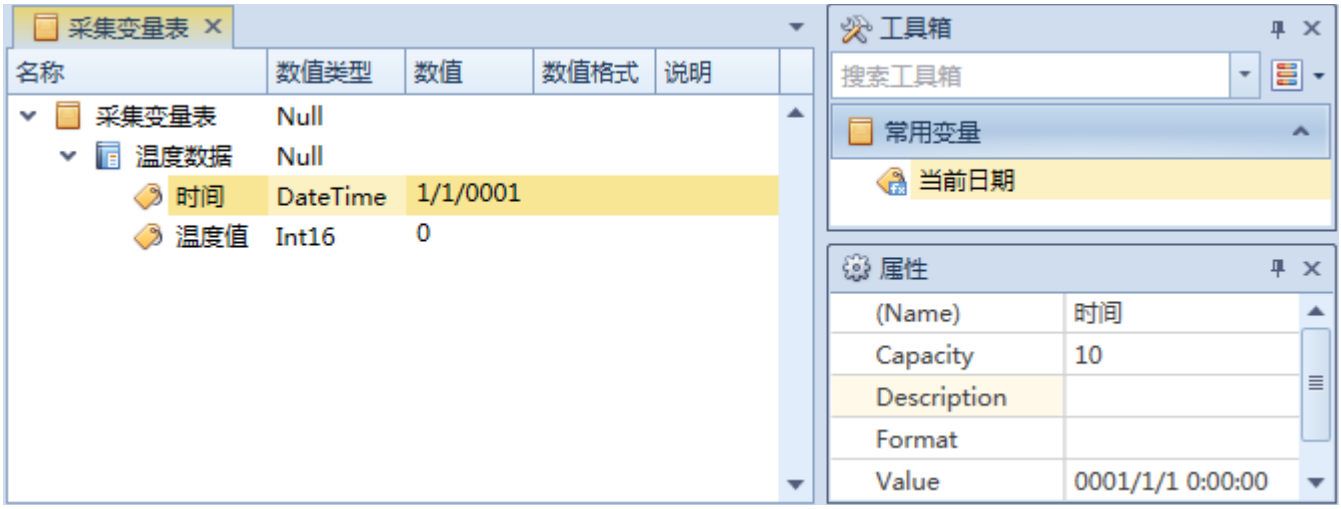
接下来，选中新创建的 TXT 文件变量，在属性面板，依次按以下属性表修改属性值。

属性	值	描述
(Name)	温度数据	变量名称
Capacity	10	变量缓存容量，采集速度越高，缓存要越大。
Directory	D:\Temp	文件存储的目录
FileName	温度数据.txt	文件名

接下来，选中“温度数据”节点，单击鼠标右键，在弹出菜单中选择“添加 -> 变量”，连续添加两个变量，分别命名为“时间”和“温度值”，在属性面板，依次按以下属性表修改属性值。

属性	值	描述
(Name)	时间	
Capacity	0	
ValueType	DateTime	

属性	值	描述
(Name)	温度值	
Capacity	0	
ValueType	Int16	

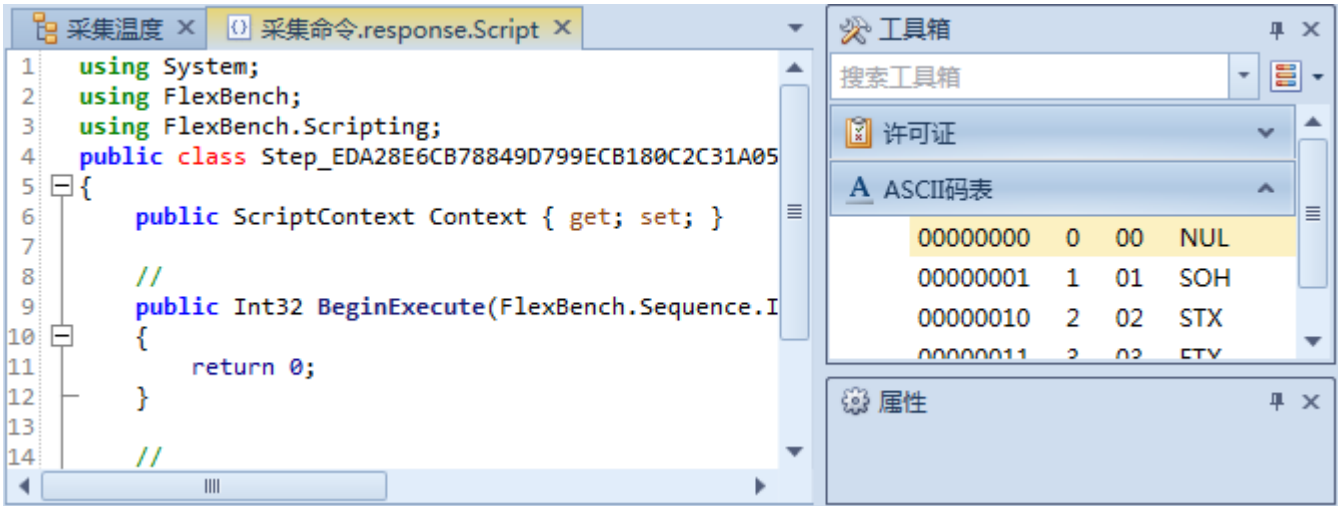


文本文件类型变量，当其所有子变量都改变过后，形成一条记录，保存到文件中。本例子中的温度数据，每次采集完毕，通过脚本把采集时间和温度值分别赋给“时间”和“温度值”变量，即可生成一条记录。

2.4.5 第 5 步 使用脚本关联序列数据和变量

序列中采集到的温度数据，可以通过脚本给变量表中的变量赋值，进而关联采集数据和变量值。软件系统支持的脚本类型有 C#、Visual Basic 和 Python，可以在项目属性中设置，本例子使用 C#脚本进行演示。

在“采集温度”序列编辑页面，选中“采集命令.response”节点，单击工具栏的“脚本”按钮，即可打开脚本编辑页面。



在打开的脚本编辑页面中，显示“采集命令.response”步骤的脚本代码。下面是 C#版本的步骤脚本模版。

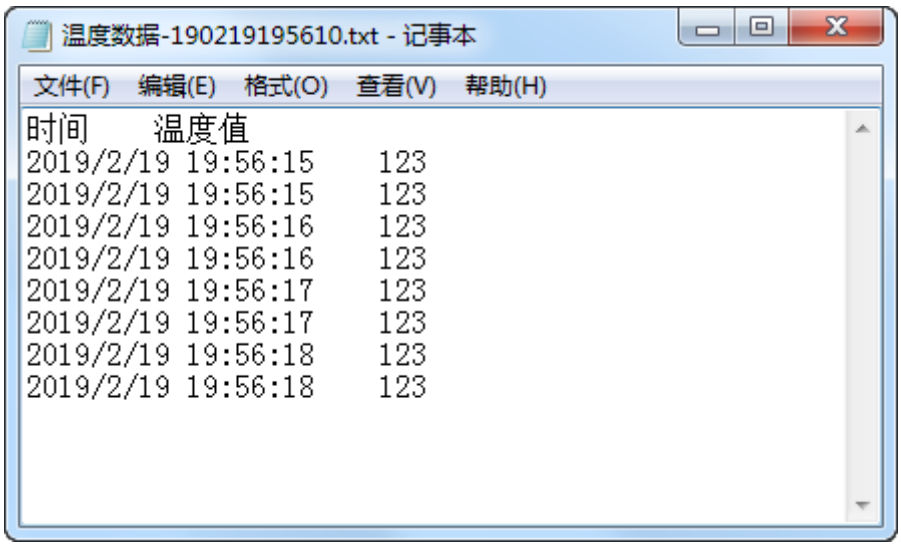
```
// 命名空间
using System;
using Genesis;
using Genesis.Scripting;
```

```
///
/// 脚本类，类名
///
public class Step_EDA28E6CB78849D799ECB180C2C31A05
{
  // 脚本上下文
  public ScriptContext Context { get; set; }

  // 步骤开始执行之前执行。
  // 参数：context – 步骤运行时上下文
  //       step – 当前执行的步骤
  // 返回：暂不定义
  public Int32 BeginExecute(Genesis.Sequence.IStepContext context,
      Genesis.Sequence.IStep step)
  {
    return 0;
  }

  // 步骤执行完毕之后执行。
  // 参数：context – 步骤运行时上下文
  //       step –当前执行的步骤
  // 返回：暂不定义
  public Int32 EndExecute(Genesis.Sequence.IStepContext context,
      Genesis.Sequence.IStep step)
  {
    return 0;
  }
}
```

接下来，在 EndExecute 函数中实现提取采集的数据并赋值给指定的变量。

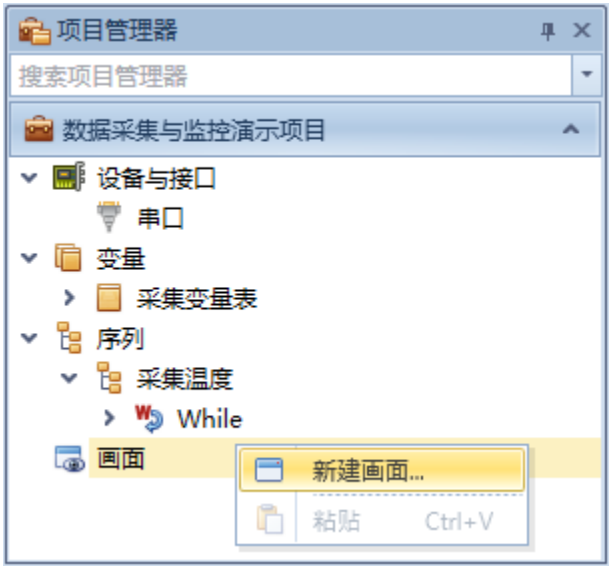
```
  public Int32 EndExecute(Genesis.Sequence.IStepContext context,
      Genesis.Sequence.IStep step)
  {
    if (step.Result.Status == (int) Genesis.Sequence.ResultStatus.Passed)
    {
      // 提取采集的温度值，索引号是 3。
      Int16 temperature = (Int16)step.Result.DataFields[3].Value;
      DateTime time = DateTime.Now;
      // 设置变量表的变量值，变量表为 Variants 容器，通过路径的方式访问。
      context.Variants["采集变量表/温度数据/时间"] = time;
      context.Variants["采集变量表/温度数据/温度值"] = temperature;
    }
    return 0;
  }
```

至此，已经完成温度数据的采集和保存功能，保存的数据格式如下图所示。

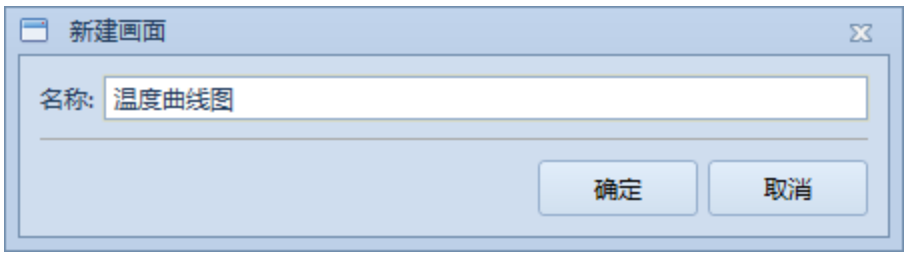


2.4.6 第 6 步 添加界面

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

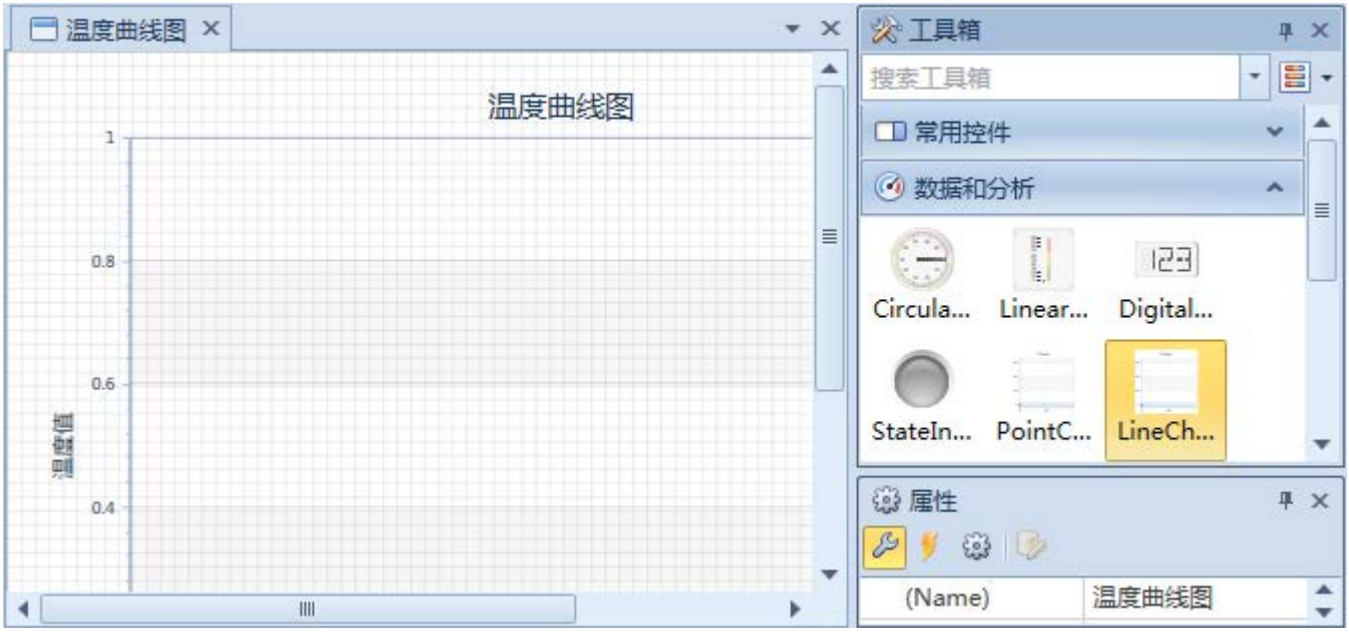


弹出新建画面对话框中，填写“名称”，点击“确定”。



在项目管理器中选择新创建的“温度曲线图”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...”，打开画面编辑页面。

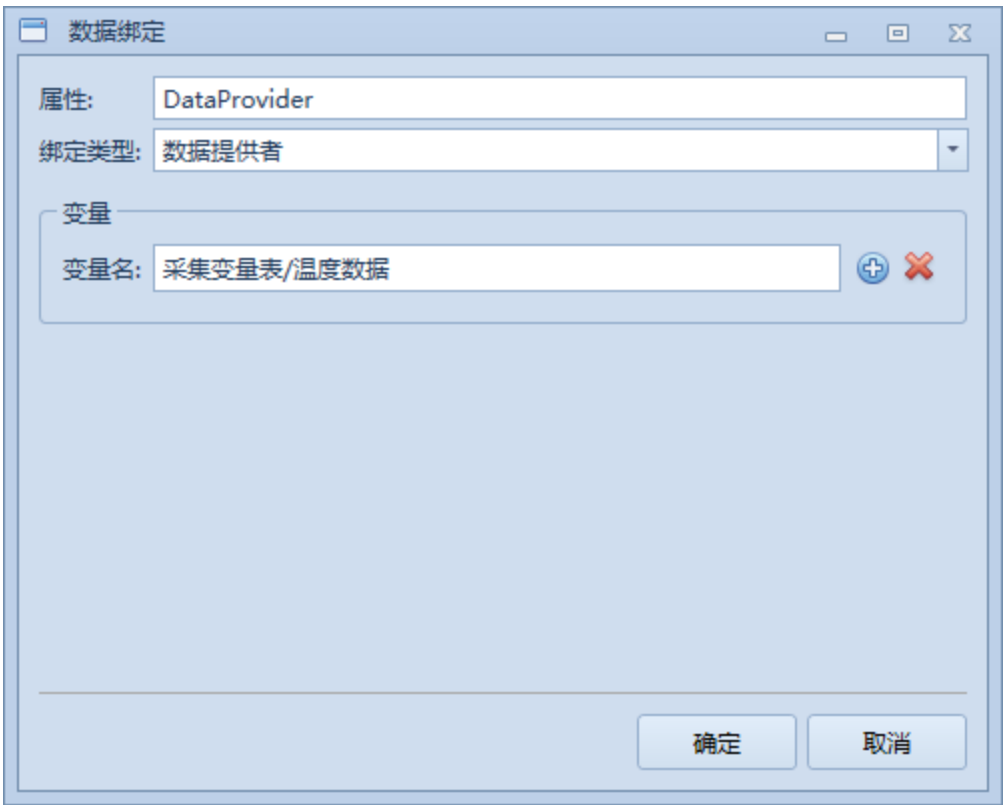
点击工具箱右上角菜单，在弹出菜单中选择“打开模具...”，在 Controls 目录下选择模版文件“DataAndAnalytics.schema”，点击“打开”；然后用鼠标在工具箱中选中“LineChart”条目，在画面中创建一个曲线显示控件。



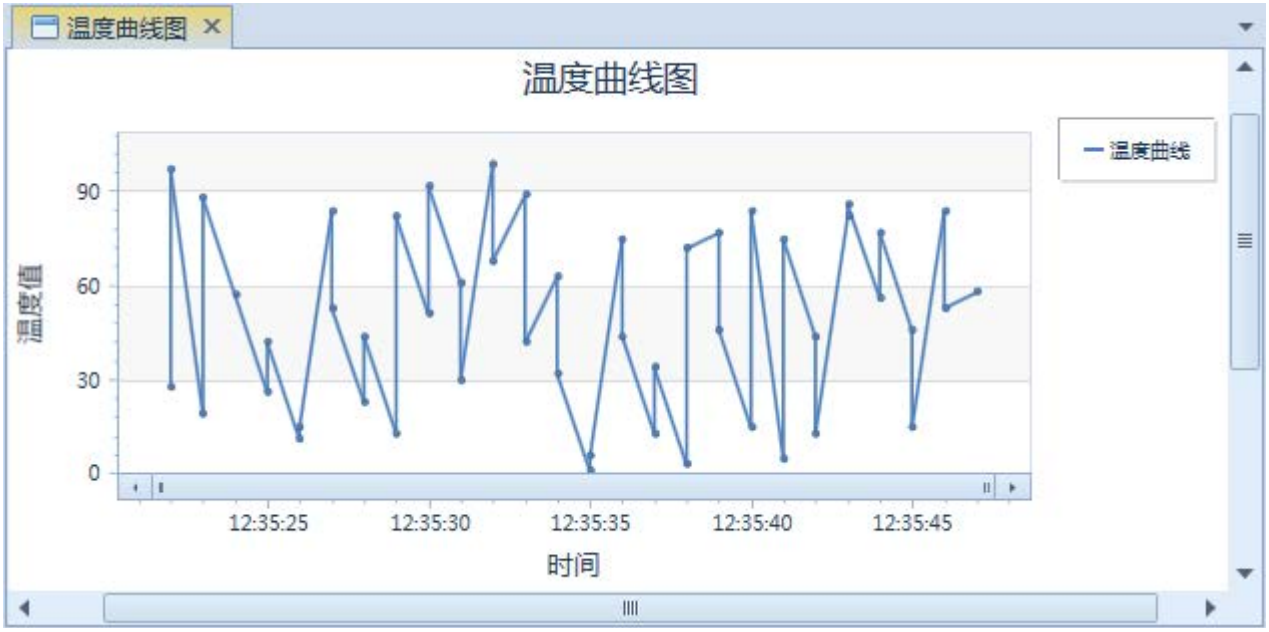
接下来，在属性面板，依次按以下属性表修改控件的属性值。

属性	值	描述
AxisXScaleType	DateTime	X 轴的数据类型
AxisXTitle	时间	X 轴标题
AxisYTitle	温度值	Y 轴标题
Title	温度曲线图	总标题
ChartSeries	温度曲线,时间,温度值	图表曲线系列，格式为“<系列名称 1>,<X 变量名>,<Y 变量名>;<系列名称 2>,<X 变量名>,<Y 变量名>;...”
DataProvider		数据提供者，通过绑定数据的方法和变量关联。

接下来，处理数据绑定问题，选择 DataProvider 属性，点击“创建数据绑定”按钮，弹出数据绑定对话框，绑定类型选择“数据提供者”，然后点击“添加变量”按钮，在弹出的变量选择对话框中选择“温度数据”变量，点击“确定”。



至此，温度曲线图界面功能完成。

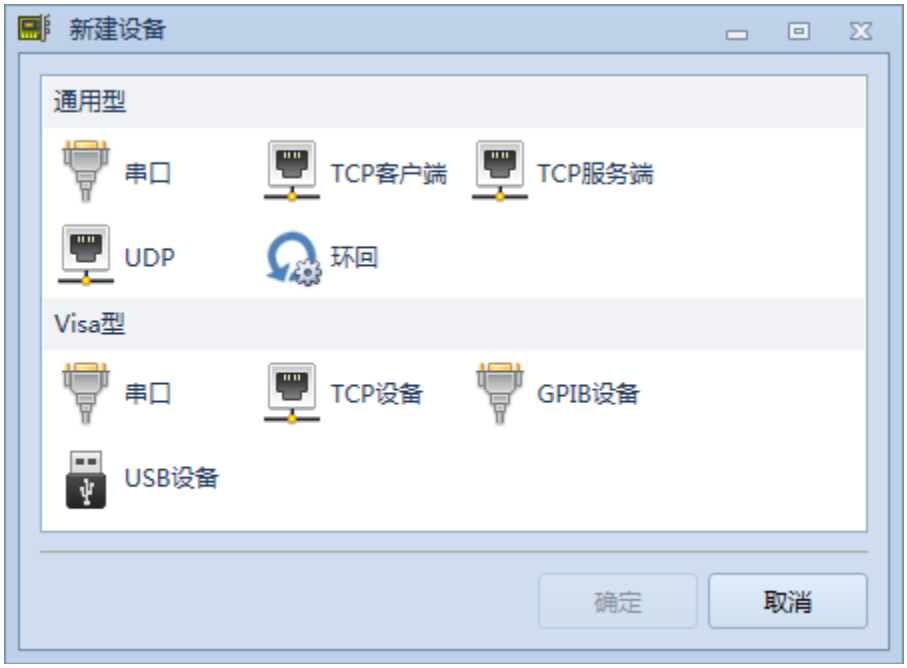


3. 开发指南

3.1 设备与接口

3.1.1 简介

用户可以创建任意设备和接口组合，可以同时对不同的设备和接口进行通信，满足各种测控连接需求。软件内置了通用型串口和网口，Visa 型串口、网口、GPIB 以及 USB 等设备和接口，满足各种标准设备和接口的连接和通信；同时，设备适配器还可以通过插件的方式，让用户扩展自定义的，非标准的设备和接口，以满足特殊的连接需求。



3.1.2 VISA 接口类型

VISA 型设备和接口，需要美国 NI 公司的 NI-VISA 运行时库支持，用户可以到 NI 官网进行下载安装。

3.1.3 自定义接口类型

软件将设备和接口分为两种类型：消息型和寄存器型。对应的接口分别为 IMessageDeviceSession 和 IRegisterDeviceSession，两者都继承自 IDeviceSession，目前软件仅支持消息型设备和接口。

扩展一个新的消息型接口类型，分三步走：

第一步：编写接口通信驱动和接口参数设置界面。第一部分是通信驱动，通信类必须从 MessageDeviceSession 类继承，实现通信参数的传递和数据的收发功能；第二部分是接口参数设置界面，界面类必须实现 IDeviceSessionEditor 接口。

第二步：配置 Manifest.xml 文件。在 Manifest.xml 文件中配置设备扩展点，形式如下所示。

```
<Extension Point="Genesis.Device.Devices">
    <DeviceCategory Id="Company" Name="XX公司设备与接口">
      <Description>XX公司的设备</Description>
```

```
    </DeviceCategory>
    <Device Id="CompanySerialXX" Category="Company" Name="XX型串口"
Class="Company.Device.SerialSessionXX" EditorClass="Company.Device.UI.SerialSessionXXDialog">
      <Image>Base64编码的小图标</Image>
      <LargeImage> Base64编码的大图标</LargeImage>
      <Description>XX型串口-具有通用串口功能和增强串口功能</Description>
    </Device>
  </Extension>
```

第三步：按照系统支持的插件打包方式打包，拷贝到系统的 Plugins 目录下。

关于扩展设备和接口的用法例子，请参考：<软件安装目录>\Examples\Basics\Devices。

3.2 变量

3.2.1 简介

用户可以创建变量、变量数组、变量容器，还可以创建扩展变量，如表达式变量、文本文件变量、数据库变量等，满足各种测控数据传递、呈现和存储需求。

关于变量的用法例子，请参考：<软件安装目录>\Examples\Basics\Variants。

3.2.2 变量描述

变量（普通变量）

属性	描述
(Name)	变量名，同一父节点下的子变量名称不能重复。
Capacity	变量缓存容量，指能够缓存多少次变化值，一般用于采集转存或显示应用，采集速度越高，缓存要越大。
Description	描述
Format	变量的格式，字符串格式的定义和微软.Net 系统的字符串格式定义一样，例如日期类型变量，格式可以定义为 yyyy/MM/dd HH:mm:ss。
Value	变量值
ValueType	变量值的类型，软件支持的类型有 Boolean、Sbyte、Byte、Int16、UInt16、Int32、Uint32、Int64、Uint64、Float、Double、Decimal、DateTime、String、BitString。

数组变量

数组变量中的元素都是同一个类型的变量。

属性	描述
(Name)	同上
Capacity	同上
Description	同上
ItemType	数组元素的类型，软件支持的类型同上 ValueType。
Rank	数组维数和大小，格式：m,n,...，多维数组将展开为一维数组。

容器变量

容器变量可以容纳任何类型的变量。

属性	描述
(Name)	同上
Capacity	同上
Description	同上

扩展变量 – 表达式变量

属性	描述
(Name)	同上
Capacity	同上
Description	同上
Format	同上
Value	同上
ValueType	同上
Device	绑定的设备或接口名称，暂不使用。
Expression	表达式，运行时动态计算和产生变量的值。
Frequency	通道频率，单位 Hz，表示变量数据每秒钟的采样次数。
Timeout	变量数据采样失败的超时时间，单位 ms。

扩展变量 – 数据字段变量

数据字段变量可以采集数据库指定表中指定字段的最新值。

属性	描述
(Name)	同上
Capacity	同上
Description	同上
Format	同上
Value	同上
ValueType	同上
Device	绑定的设备或接口名称，暂不使用。
Frequency	通道频率，单位 Hz，表示变量数据每秒钟的采样次数。
Timeout	变量数据采样失败的超时时间，单位 ms。
ConnectionString	数据库连接串 例如： SQLite 数据库连接串（Data Source 可以是绝对路径或者相对路径，相对路径相对于项目文件所在目录）：Data Source=D:\VariantsSQLite.db MySQL 数据库连接串：Data Source=localhost;port=3306;Initial Catalog=VariantsMySQL;User ID=root;Password=12345678
DatabaseType	数据库类型，软件支持 MySQL，SqlServer 和 SQLite。
FieldName	字段名称，需要获取数值的字段。
OrderFieldName	排序字段名称，使用该字段排序后，能够得到数据表的最新一行记录。
TableName	数据表名称，该表包含 FieldName 和 OrderFieldName 字段。

扩展变量 – 文本文件变量

文本文件变量是一个容器类变量，可以容纳任意类型变量。容器中的所有变量的值都改变后，生成一

行记录。在这个过程中，多次改变值的变量，以最后一次的值为准。

属性	描述
(Name)	同上
Capacity	同上
Description	同上
Delimiter	列分隔符
Directory	文件保存的目录，可以是绝对路径或者相对路径，相对路径相对于项目文件所在目录，如果为空，则默认为项目文件所在目录。
FileName	文件名
FileSize	单个文件大小限制，单位为字节，0 表示不限制大小，>0 表示实际限制值，超过限制值，自动新建一个文件，在文件后用自动增加序号。
PrintHeader	是否打印列表头。
Timeout	超时时间，单位 ms。

扩展变量 – 数据库变量

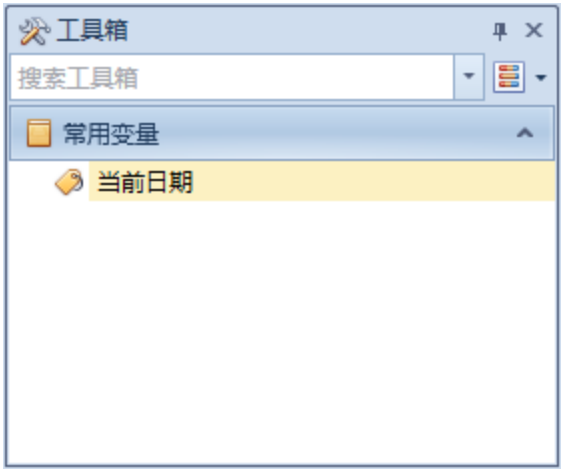
数据库变量是一个容器类变量，可以容纳任意类型变量。容器中的所有变量的值都改变后，生成一行记录。在这个过程中，多次改变值的变量，以最后一次的值为准。

属性	描述
(Name)	同上
Capacity	同上
Description	同上
ConnectionString	数据库连接串 例如： SQLite 数据库连接串（Data Source 可以是绝对路径或者相对路径，相对路径相对于项目文件所在目录）：Data Source=D:\VariantsSQLite.db MySQL 数据库连接串：Data Source=localhost;port=3306;Initial Catalog=VariantsMySQL;User ID=root;Password=12345678
DatabaseType	数据库类型，软件支持 MySQL，SqlServer 和 SQLite。
TableName	数据表名称，该表自动生成，表字段名称采用容器中变量名。
Timeout	超时时间，单位 ms。

注：SQLite 数据库变量在运行启动时自动检查和创建数据库和数据表； MySQL 和 SqlServer 数据库变量必须手工创建数据库，在运行启动时自动检查和创建数据表。

3.2.3 变量模版

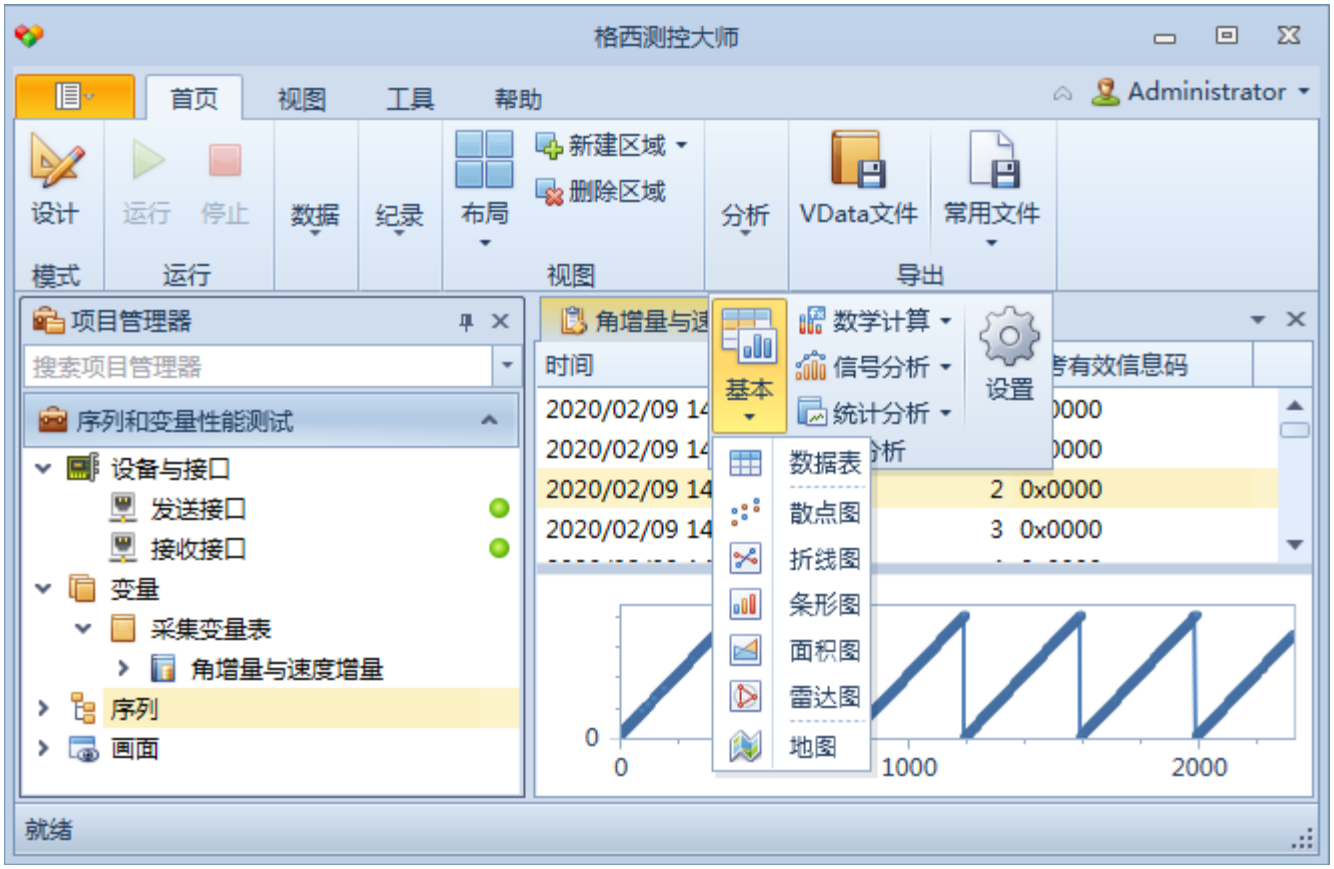
软件支持变量模版功能，用户可以把常用的变量保存为模版，使用的时候直接从模板库拖放到变量编辑器即可创建。



3.2.4 变量数据的采集与分析

软件支持在运行时对变量数据进行显示、采集和分析。运行时，从项目管理器中选择需要观察的变量，然后点击工具栏的“变量数据”->“变量数据”，打开变量数据页面，如图所示。

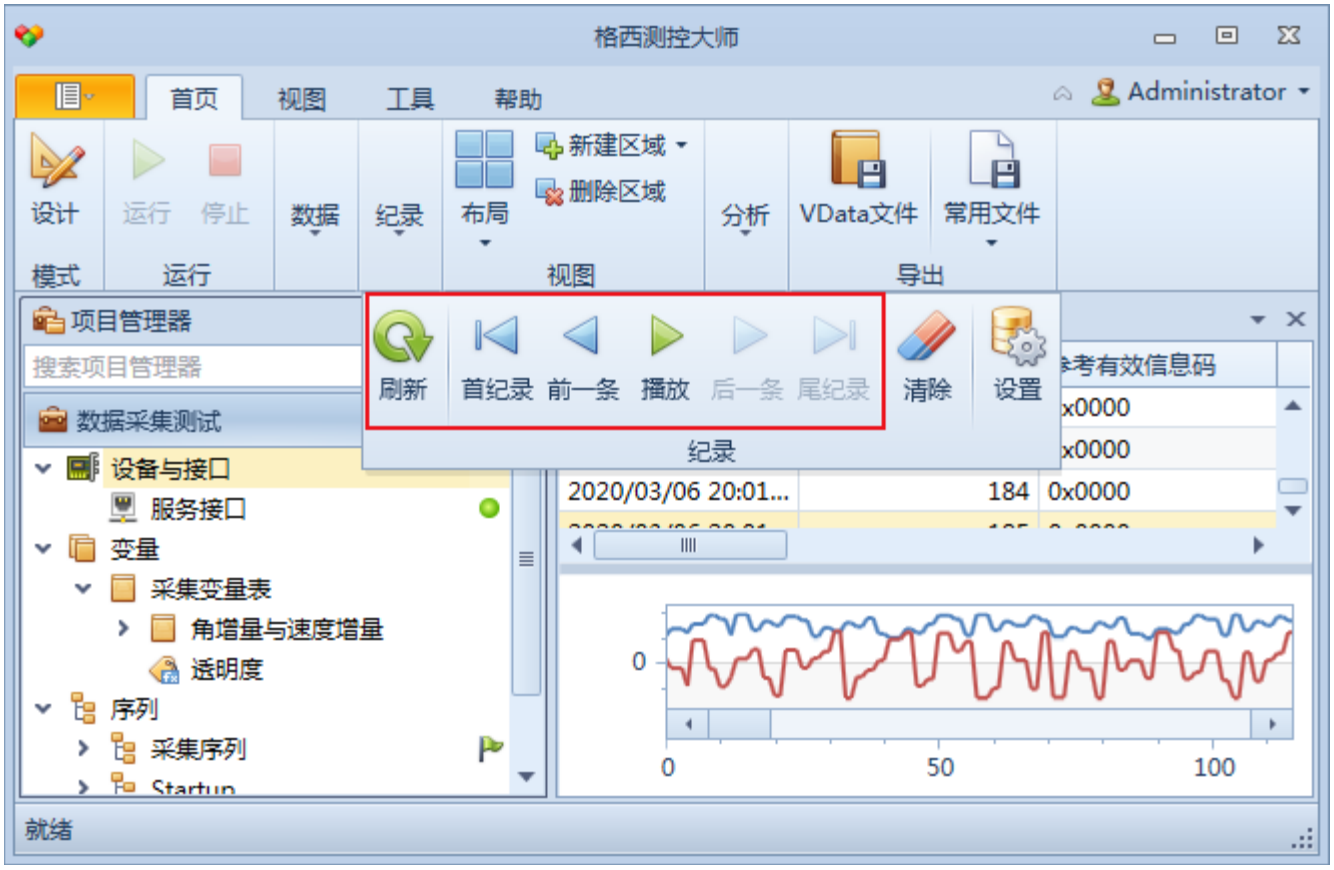
用户可以通过布局工具，建立不同的区域，分别对变量数据进行观察和分析。软件提供的观察和分析的方式和方法有数据表、图表、数学分析、信号分析以及统计分析等，同时，也允许用户支持通过插件的方式扩展数据观察和分析功能，以满足特定的需求。



3.2.5 变量数据的回放

软件支持在运行时对变量数据进行回放。进行回放操作时，需要先停止“刷新”，即工具栏的“刷新”按钮处于不选择状态，如图所示。

播放速度可以通过“设置”对话框进行设置。



3.2.6 自定义变量数据分析类型

软件支持通过插件的方式扩展数据观察和分析功能，以满足特定的需求。

扩展一个新的变量数据分析类型，分三步走：

第一步：编写分析对象（继承自 VariantAnalysisObject），分析对象的设置信息（实现接口 IVariantAnalysisObjectSettings）；编写分析视图（实现接口 IVariantAnalysisView），分析视图设置信息（实现接口 IVariantAnalysisViewSettings）；编写分析设置面板（实现接口 IVariantAnalysisEditor）。

第二步：配置 Manifest.xml 文件。在 Manifest.xml 文件中配置设备扩展点，形式如下所示。

```
<Extension Point="Genesis.Variant.Analytics">
  <VariantAnalysis Id="ExampleVariantAnalysis" Category="Basics" Group="Example" Name="表格分析例子" ObjectClass="Company.Variant.Analytics.ExampleVariantAnalysisObject" ViewClass="Company.Variant.Analytics.ExampleVariantAnalysisView" EditorClass="Company.Variant.Analytics.ExampleVariantAnalysisEditor">
    <Image> </Image>
    <LargeImage> </LargeImage>
    <Description>表格分析例子</Description>
  </VariantAnalysis>
</Extension>
```

第三步：按照系统支持的插件打包方式打包，拷贝到系统的 Plugins 目录下。

关于扩展变量数据分析类型的用法例子，请参考：
<软件安装目录>\Examples\Basics\Variants\Extensions。

3.3 序列

3.3.1 简介

用户可以创建执行序列，实现任意逻辑的执行过程，并且序列可以并行执行，满足各种测控自动化需求。

- 支持流程控制，如分支语句 If、Switch，循环语句 For、While，并行语句 Parallel。
- 支持同步控制，如等待（Wait）、通知（Notification）。
- 支持数值类型动作步骤（Value）、协议类型动作步骤（Protocol）。
- 支持序列嵌套，支持复杂的层次结构。
- 支持脚本，脚本可以无缝调用.Net Framework 类库，调用第三方托管库来实现执行逻辑。
- 支持协议模板和步骤模板。

关于序列和步骤的用法例子，请参考：<软件安装目录>\Examples\Basics\Steps。

3.3.2 步骤描述

步骤是序列的基本组成部分，所有步骤类型都有一组公共的属性。

属性	描述
(Id)	唯一标识步骤的字符串。
(Name)	步骤名称，名称不是步骤的标识，在同一等级上允许步骤名称相同。
Active	是否激活，不激活的步骤不执行。
Category	步骤分类，暂不使用。
Description	步骤描述。
Loggable	是否记录结果，不记录的步骤能够执行，结果不记录到结果表，也不参与判错。
Socket	执行槽，用于多个器件并行执行分开记录每个器件的结果。>=0 表示槽号，步骤执行记录归到该槽号；<0 表示步骤执行记录所有槽都有记录。单槽执行，默认为-1 即可。
Timeout	步骤执行超时时间，单位 ms。>0 表示实际超时时间，超时后终止步骤执行；<=0 表示无限等待步骤执行完毕。

动作类 – Value 步骤

属性	描述
Device	序列绑定的设备和接口的名称，可以在脚本中通过名称访问设备。
LimitExpression	限定值表达式。
Value	步骤的值，是一个变量类型，在执行过程通过脚本进行设置。

Value 步骤是值类型动作步骤，该步骤需要使用脚本，根据执行要求，得到执行数值后，赋值给 Value 属性，步骤的执行结果根据 LimitExpression 和 Value 属性来判断是否通过。

该步骤的主要应用场合：

- 用于综合判断汇总执行结果。
- 用于调用第三方库，得到执行的结果，相当于对第三方库的一个包装。

动作类 – Protocol 步骤

属性	描述
ByteInterval	协议字节间隔时间，单位 ms。该属性在发送时有效，指每发送一个字节，延时多少毫秒。
Device	序列绑定的设备和接口的名称，可以在脚本中通过名称访问设备。
EscapeTable	协议转义表，格式：转义符=转义数据,...，数据是 16 进制。对应发送，数据发送之前除头尾之外，把和转义符一样的字节变为转义数据；对于接收，数据收完之后除头尾之外，把和转义数据一样的数据变为转义符字节。 例如：02=1BE7,03=1BE8,1B=1B00
FrameInterval	协议帧间隔时间，单位 ms。指发送或接收一帧完毕后，延时多少毫秒。
LimitExpression	限定值表达式。
MatchMode	匹配模式，Static 表示静态匹配，从收到的第一个字节开始匹配，匹配不上认为失败；Scanning 扫描匹配，从收到的第一个字节开始匹配，匹配不上后移一个字节继续匹配，直到匹配成功或者字节用完。
TransceiveMode	传输模式，有发送和接收两种模式。

Protocol 步骤是协议类型动作步骤，步骤执行时：

- 发送模式下，自动组帧发送。
- 接收模式下，根据帧结构自动匹配数据，当匹配正确后，自动解析帧数据，如果匹配超时，执行结果设置为失败。

动作类 – Process 步骤

属性	描述
Device	序列绑定的设备和接口的名称，可以在脚本中通过名称访问设备。
LimitExpression	限定值表达式，可以用于判定步骤的执行结果通过与否。表达式不为空时，表达式的值覆盖可执行文件进程终止时返回的值。
FileName	可执行文件的文件名，可以是绝对路径或者相对路径，相对路径相对于项目文件所在目录。
Arguments	可执行文件的命令行参数，支持表达式。
DataName	为进程的标准输出数据指定名称，即 Result.Data 的名称。
CreateNoWindow	执行时不创建窗口。
WindowStyle	指定在启动进程时新窗口应如何显示。
WaitMode	等待模式，No 表示触发执行后立即完成步骤，WaitForExit 表示步骤等待，直到执行的进程结束或者步骤超时。

Process 步骤是执行外部可执行文件的动作步骤，主要应用场合：

- 调用外部可执行文件，获取执行的结果。

Process 步骤对外部可执行文件的编写要求：

- 进程终止时返回的 int 类型值作为 Process 步骤的结果状态值，大于 0 表示执行通过，小于 0 表示执行失败。

- 进程执行过程中，所有输出到标准输出流的字符串，都被保存到 Process 步骤的结果变量 Result.Data 中，用户可以对该值进行处理。

流程控制类 – If 步骤、Then 步骤、Else 步骤

属性	描述
ConditionExpression	条件表达式，可以通过表达式编辑器进行编辑。 例如：Variants["变量表/发送位置"] < Variants["变量表/总长度"]

If 步骤是分支类型步骤，配合 Then 步骤和 Else 步骤一起使用，当 ConditionExpression 的值为 True 时，执行 Then 分支，否则执行 Else 分支。

流程控制类 – For 步骤

属性	描述
StartExpression	初始值表达式，如 1。
IncrementExpression	增量值表达式，可以是正数或负数。
EndExpression	结束值表达式，如 3，Variants["Vars/Int1"]等。

For 步骤是循环类型步骤，循环范围[Start,End]，包括初始值和结束值。

流程控制类 – While 步骤

属性	描述
ConditionExpression	条件表达式，可以通过表达式编辑器进行编辑。 例如：Variants["变量表/结束标志"] = False

流程控制类 – Break 步骤

Break 步骤用于循环类型步骤中，终止整个循环的执行，用法类似 C/C++等高级语言 break 语句。

流程控制类 – Continue 步骤

Continue 步骤用于循环类型步骤中，结束本次循环，而不终止整个循环的执行。用法类似 C/C++等高级语言 continue 语句。

流程控制类 – Switch 步骤

属性	描述
DiscriminantExpression	判别式，一般引用变量的值，如 Variants["Vars/Int1"]。

Switch 步骤是分支类型步骤，用法类似 C/C++等高级语言 switch 语句。

流程控制类 – Case 步骤

属性	描述
ConditionExpression	条件表达式，一般为常量，可以是软件支持的任意数据类型。

Case 步骤只能配合 Switch 步骤使用。

流程控制类 – Parallel 步骤

属性	描述
ParallelMode	并行模式。 ✧ All –无 Specific 分支，则完成所有分支后退出；有 Specific 分支，则完成所有 Specific 分支后马上退出。 ✧ Any –无 Specific 分支，完成任意一个分支后退出；有 Specific 分支，完成所有 Specific 分支，并完成除 Specific 分支之外的任意一个 Default 分支后马上退出。

Parallel 步骤为并行执行步骤，可以同时执行其下的所有 Branch 步骤。

流程控制类 – Branch 步骤

属性	描述
BranchMode	分支模式。 ✧ Default – 默认 ✧ Specific - 分支运行完毕是退出判断的一个必要条件，具体用法需要结合其父 Parallel 步骤的并行模式设置。

Branch 步骤为并行执行步骤的分支步骤，必须配合 Parallel 步骤使用。

同步类 – Wait 步骤

属性	描述
WaitTime	等待时间，单位 ms。表示等待一定时间再执行后续的步骤，不影响其他并行执行的步骤。

同步类 – Notification 步骤

属性	描述
NotificationName	通知的名称，项目内唯一，标识一个通知。
NotificationOperation	通知的操作类型 Wait – 表示等待通知 Notify – 表示发出通知，
NotificationParameter	通知的参数，字符串类型，携带自定义字符串或变量，只有 Notify 操作有效。

Notification 步骤用于并行执行中不同分支的同步，同步原理和广播方式类似，所有调用 Wait 操作类型的分支阻塞，直到收到相同名称的通知，或超时失败，才执行后续步骤。

容器类 – Sequence 步骤

Sequence 步骤是容器类型步骤，可以容纳任意类型步骤，可以分类组织步骤。

属性	描述
SequenceMode	序列模式。 ✧ Random – 随机模式，允许单个执行子步骤；运行时子步骤失败不影响继续运行其他子步骤。 ✧ Sequential- 顺序模式，不允许单个执行子步骤；运行时任何一个子步骤失败，则整个序列失败并终止执行。

3.3.3 协议描述

通信协议是指双方实体完成通信或服务所必须遵循的规则和约定。协议定义了数据单元使用的格式，信息单元应该包含的信息与含义，连接方式，信息发送和接收的时序，从而确保在通信中数据顺利地传送到确定的地方。

软件可以根据协议数据单元格式的定义，把协议拆分为多个协议单元，逐个进行构建，完全和协议标准的描述一一对应。

软件定义了协议单元的基本结构，每个协议单元都有一组公共的属性。

属性	描述
(Name)	协议单元的名称，可以是任意字符串。
(Type)	协议单元的类型，有以下 5 种类型。 ✧ General – 普通型，一般固定长度的单元。 ✧ Variable – 变长型，数据是可变长度的，两个变长字段之间需要至少有一个 Constant 属性为 True 的普通型单元分隔，单元的 Length 恒为 -1。 ✧ Repeatable - 重复型，可以指定重复 n 次，或者匹配重复 n 次。 ✧ Choosable - 选择型，根据条件选择字段是否有效，该类型单元的 Constant 属性恒为 True。 ✧ Computable - 计算型，由自身或者其他数据通过一定的算法计算得到。
Constant	是否为常量，一般情况协议标准定义的数据单元为常数，则设置为 True。接收时根据该属性决定是否匹配数据，当一条协议的所有 Constant 为 True 的协议单元均匹配正确，软件才认为正确收到该协议帧。
DataType	协议单元的数据类型，软件支持的类型有 Boolean、Sbyte、Byte、Int16、UInt16、Int32、Uint32、Int64、Uint64、Float、Double、Decimal、DateTime、String、BitString。
Description	协议单元的描述字符串。
Endian	协议单元的字节序。 ✧ BigEndian – 大端，即高字节在前面，低字节在后面。 ✧ LittleEndian – 小端，即低字节在前面，高字节在后面。 例如，一个 4 字节的 Int32 类型数据 1，大端发送顺序是 00000001，小端发送顺序是 01000000。
Format	协议单元的解析和显示格式，字符串格式的定义和微软.Net 系统的字符串格式定义一样，例如日期类型变量，格式可以定义为 yyyy/MM/dd HH:mm:ss。
Length	协议单元的位长，即比特位的个数，如果设置的 Length 比 DataType 的默认位长小，则多余的位会被剪切。
Parameters	协议单元的参数，不同的协议单元的类型，有不同的参数组合，详见下面描述。
Value	协议单元的数值，根据设置的 DataType 和 Length 来设置。 ✧ Boolean – True 或 False ✧ Sbyte、Byte、Int16、UInt16、Int32、Uint32、Int64、Uint64、Float、Double、Decimal – 10 进制数值 ✧ DateTime - 微软.Net 系统的 DateTime 格式支持的字符串 ✧ String – 任意字符串 ✧ BitString – 本软件支持的位串格式，用逗号分隔的一个符号字符

	串，0x 开头为 16 进制，0o 开头为八进制，0b 开头为二进制，按从左到右的顺序大端方式依次排列。例如："0xF1,0b10,0o77",则对应"0b11110001 10 111 111"。
Visible	协议单元是否解析和显示。

Computable 型协议单元的参数

属性	描述
Algorithm	算法类型。 软件支持常用的 32 种计算算法，如果内置的算法不能满足要求，用户可以通过 IprotocolAlgorithm 和 IprotocolAlgorithmParameter 接口扩展自定义的算法。 扩展方法请参考：<软件安装目录>\Examples\Basics\Steps\Extensions
Priority	计算的优先级，是大于等于 0 的整数，0 优先级最高，依次降低，优先级越高计算越优先，同级则按出现顺序计算。
Location	计算单元在待计算数据的位置。 ✧ Front – 前面，StartPosition 或 EndPosition >=0 表示从计算单元末尾起始开始计算，0 表示紧挨着计算单元末尾的第 1 个字节，依次类推；StartPosition 或 EndPosition <0 表示从整帧数据末尾开始计算，-1 表示整帧数据末尾第 1 字节，-2 表示整帧数据末尾第 2 字节，依次类推。 ✧ Middle – 中间，StartPosition 在计算单元前面，StartPosition>=0 表示从整帧数据开头开始计算，0 表示整帧数据的第 1 个字节，依次类推，StartPosition <0 表示从计算字段的前面开始计算，-1 表示紧跟计算字段的前 1 个字节，-2 表示前 2 个字节起始，依次类推。EndPosition 在计算单元后面，EndPosition >=0 表示从计算单元末尾起始开始计算，0 表示紧挨着计算单元末尾的第 1 个字节，依次类推，EndPosition <0 表示从整帧数据末尾开始计算，-1 表示整帧数据末尾第 1 字节，-2 表示整帧数据末尾第 2 字节，依次类推。 ✧ Back – 后面，StartPosition 在计算单元前面， StartPosition>=0 表示从整帧数据开头开始计算，0 表示整帧数据的第 1 个字节，依次类推，StartPosition <0 表示从计算字段的前面开始计算，-1 表示紧跟计算字段的前 1 个字节，-2 表示前 2 个字节起始，依次类推。EndPosition 在计算单元前面，EndPosition>=0 表示从整帧数据开头开始计算，0 表示整帧数据的第 1 个字节，依次类推，EndPosition <0 表示从计算字段的前面开始计算，-1 表示紧跟计算字段的前 1 个字节，-2 表示前 2 个字节起始，依次类推。 用法请参考：<软件安装目录>\Examples\Basics\Steps\Steps.fbp 的“序列\基本步骤\ActionProtocol 测试\计算型测试”
StartPosition	待计算数据的起始字节序号，具体定义参考 Location 属性的描述。
EndPosition	待计算数据的结束字节序号，具体定义参考 Location 属性的描述。

Repeatable 型协议单元的参数

属性	描述
Repetitions	重复次数。范围-1~n。

	接收模式时,-1 表示 0~任意次重复，>=0 按实际次数接收; 发送模式时，>0 按实际次数发送，<=0 不发送。

3.3.4 步骤脚本

步骤支持脚本功能，脚本可以调用公共的接口，可以无缝调用.Net Framework 类库，可以调用第三方托管库来实现执行逻辑和处理执行的结果。

下面是 C#版本的步骤脚本模版，由一个.NET 类 Step_<步骤 Id>构成，该类包含一个 Context 属性，两个方法，分别是 BeginExecute 方法，EndExecute 方法。

```
/**
 * 命名空间定义
 */
using System;
using Genesis;
using Genesis.Scripting;
using Genesis.Sequence;
using Genesis.Workbench;

/**
 * 步骤脚本类
 */
public class Step_21A15D82B8474E86B99D20F11663BD07
{
  /**
   脚本上下文
   */
  public ScriptContext Context { get; set; }
  /*****************************************************************************
   函数名称：BeginExecute
   功能说明：步骤完成初始化，执行之前执行。
   输入参数：context – 运行时步骤上下文，存储运行时的参数
             step – 当前步骤
   返回参数：暂无定义
   *****************************************************************************/
  public Int32 BeginExecute(IStepContext context, IStep step)
  {
    return 1;
  }
  /*****************************************************************************
   函数名称：EndExecute
   功能说明：步骤执行完毕之后执行。
   输入参数：context – 运行时步骤上下文，存储运行时的参数
             step – 当前步骤
   返回参数：暂无定义
   *****************************************************************************/
```

```
  public Int32 EndExecute(IStepContext context, IStep step)
  {
    return 1;
  }
}
```

3.3.4.1脚本上下文 ScriptContext

定义	描述
public VariantContainer Variants { get; }	获取项目的变量集。
public string SystemDirectory { get; }	获取软件系统根目录。
public string ProjectDirectory { get; }	获取项目文件所在目录。
public Stopwatch WatchTimer { get; }	获取监视计数器，可用于计时应用。
设备会话相关操作	
public IEnumerable<IDeviceSession> DeviceSessions { get; }	获取项目的设备会话集。
public IDeviceSession GetDeviceSession(string name)	根据设备名获取项目的设备会话。
public T GetDeviceSession<T>(string name)	根据设备名获取项目的设备会话。
public void FlushDeviceSession(string name)	清除设备缓存。
步骤相关操作	
public IEnumerable<IStep> RunningSteps { get; }	获取项目正在运行的步骤列表。
public bool StartStep(string step)	执行步骤。 参数 step 可以是步骤的 Id，也可以是步骤的路径（路径格式：Id|Name [/Id|Name]，是以/分隔的 Id 或 Name 的混合字符串，由于步骤允许同名，路径匹配以第一个为准。建议：如果以名称作为路径，则保证名称在同一级别的步骤中唯一。）
public bool StartStep(string step, int times)	多次执行步骤。 参数 step 的定义同上。
public bool StopStep(string step)	停止步骤执行。 参数 step 的定义同上。
public bool ExecuteStep(string step)	执行步骤，并等待步骤执行完毕。 参数 step 的定义同上。
public bool ExecuteStep(string step, int times)	多次执行步骤，并等待步骤执行完毕。 参数 step 的定义同上。
public async Task<bool> ExecuteStepASync(string step)	异步执行步骤，并等待步骤执行完毕。 参数 step 的定义同上。
public async Task<bool> ExecuteStepASync(string step, int times)	异步多次执行步骤，并等待步骤执行完毕。 参数 step 的定义同上。

public IStepContext GetStepContext(string step)	获取步骤上下文。 参数 step 的定义同上。
public string GetStepDisplayPath(IStep step)	获取步骤的显示路径。
public void BreakIterationStep(IStep step)	终止包含 step 的循环步骤。
public void ContinueIterationStep(IStep step)	终止包含 step 的当次循环。
表达式相关操作	
public ExpressionContext CreateExpressionContext()	创建表达式计算上下文。
public IGenericExpression<T> CreateGenericExpression<T>(string expression)	创建泛型表达式。
public IDynamicExpression CreateDynamicExpression(string expression)	创建动态表达式。
public T EvaluateExpression<T>(ExpressionContext context, string expression)	通过传入的上下文计算泛型表达式。
public T EvaluateExpression<T>(string expression)	计算表达式。
public object EvaluateExpression(ExpressionContext context, string expression)	通过传入的上下文计算表达式。
public object EvaluateExpression(string expression)	计算表达式。
数据操作（该数据表为 key-value 值对，用于用户自定义数据，作用域是整个项目）	
IDictionary<string, object> Datas { get; }	数据集。数据集为 key-value 值对，通过名称键值来访问。
public object GetData(string name)	根据名称获取数据值。
public T GetData<T>(string name)	根据名称获取指定类型数据值。
public void SetData(string name, object value)	设置或添加数据到数据表。
public bool ContainData(string name)	检查是否有指定名称的数据存在。
public bool RemoveData(string name)	根据名称删除数据。
public bool TryGetData(string name, out object value)	尝试获取数据值，获取成功则返回 true，否则返回 false。
public bool TryGetData<T>(string name, out T value)	尝试获取指定类型数据值。
public void ClearDatas()	清除所有数据。
日志记录相关操作	
public void LogMessage(object value)	记录消息，显示到 Log 窗口中。 参数 value 为信息值，取 value 对象字符串显示在 Log 窗口的信息列中。
public void LogMessage(string location, object value)	记录消息，显示到 Log 窗口中。 参数 location 显示在 Log 窗口的位置列中。 参数 value 同上。
public void LogMessage(string location, string format, object arg0)	记录消息，显示到 Log 窗口中。 参数 location 同上。 参数 format 是信息值 arg0 的格式化字符串，详情参考.Net 库的 String 类的 Format 方法。
public void LogMessage(string location, string format, object arg0, object arg1)	
public void LogMessage(string location, string format, params object[] args)	

public void LogMessage(MessageSeverity severity, string location, object value)	MessageSeverity 是 Enum 类型，定义三个值 {Info, Warning, Error} 。
public void LogMessage(MessageSeverity severity, string location, string format, object arg0)	
public void LogMessage(MessageSeverity severity, string location, string format, object arg0, object arg1)	
public void LogMessage(MessageSeverity severity, string location, string format, params object[] args)	
转换相关操作	
public string ConvertToBinString(byte number)	将字节转为二进制字符串。
public string ConvertToBinString(byte number, int bitCount)	
public string ConvertToHexString(byte b)	将字节转为十六进制字符串。
public string ConvertToHexString(byte[] data)	
public string ConvertToHexString(byte[] data, int offset, int len)	
public string ConvertToHexString(byte[] data, int offset, int len, bool space)	
public string ConvertToHexString(byte[] bytes, int offset, int len, string prefix, string suffix)	
public byte[] ConvertToBytes(string hexString)	将十六进制字符串转为字节数组。
public string ConvertToString(Variant variant, int nameIndent, int nameAlignment, int valueAlignment, int descriptionAlignment)	将变量 Variant 转为字符串格式，输出名称、值和描述信息。 nameIndent 表示名称缩进字符数，xxxAlignment 是一个有符号的整数，指示插入参数以及它为右对齐 （正整数） 还是左对齐 （负整数） 到该字段的总长度。
数据库相关操作	
public IDataManager CreateDataManager(DatabaseType dbType, string connectionString)	创建数据库管理器。
JSON 相关操作	
public string ConvertToJsonString(object obj)	将对象转换为一个 JSON 字符串。
public object ConvertToJsonObject(string json, Type type)	将 JSON 字符串转换为指定类型的对象。
public T ConvertToJsonObject<T>(string json)	将 JSON 字符串转换为指定类型的对象。
项目配置文件相关操作	
public IMemento CreateProjectConfiguration()	创建新的项目配置文件，文件路径与项目文件在同一个目录，文件名为<项目文件名>.config。 返回 IMemento 类型。
public IMemento OpenProjectConfiguration()	打开项目配置文件，文件路径与项目文件在同一个目录，文件名为<项目文件名>.config。 返回 IMemento 类型，可以操作返回对象读取保存的项目参数。
public bool SaveProjectConfiguration(IMemento config)	将传入的配置内容 config 保存为项目配置文件，文件路径与项目文

	件在同一个目录，文件名为<项目文件名>.config。

ScriptContext 界面扩展

定义	描述
Window 窗体相关操作	
public MessageBoxResult ShowMessageBox(string title, string messageBoxText, MessageBoxButton button, MessageBoxImage icon)	弹出消息框。
public void ShowWindow(string title, Type contentType)	显示一个窗体，窗体内容来自 contentType，可以是用户定义的内容。
public void ShowWindow(string title, Type contentType, ResizeMode resizeMode)	显示一个窗体。
public void ShowWindow(tring title, Type contentType, object contentArgs)	
public void ShowWindow(string title, Type contentType, object contentArgs, ResizeMode resizeMode)	
public void ShowWindow(tring title, Type contentType, object[] contentArgs)	
public void ShowWindow(string title, Type contentType, object[] contentArgs, ResizeMode resizeMode)	
public bool ShowDialog(string title, Type contentType)	显示一个对话框窗体，窗体内容来自 contentType，可以是用户定义的内容。
public bool ShowDialog(string title, Type contentType, ResizeMode resizeMode)	
public bool ShowDialog(string title, Type contentType, object contentArgs)	
public bool ShowDialog(string title, Type contentType, object contentArgs, ResizeMode resizeMode)	
public bool ShowDialog(string title, Type contentType, object[] contentArgs)	
public bool ShowDialog(string title, Type contentType, object[] contentArgs, ResizeMode resizeMode)	
public string ShowOpenFileDialog(string title, string filter)	显示打开文件对话框。 参数 title 为对话框标题 参数 filter 是文件过滤器，用|符合分隔文件说明和文件类型，多个文件类型用;符号分隔，例如：文本文件|*.txt|脚本文件|*.cs;*.vb
public string ShowSaveFileDialog(string title, string filter)	显示保存文件对话框。 参数定义同上。
public string[] ShowParameterDialog(string title, string[] paramNames)	显示参数设置对话框,参数个数由参数名称数组 paramNames 决定，返回是参数字符串数组。
public string[] ShowParameterDialog(string title, string[]	显示参数设置对话框,参数个数由

paramNames, string[] paramDefaults)	参数名称数组 paramNames 决定，参数的缺省值由数组 paramDefaults 传入，返回是参数字符串数组。
public string[] ShowParameterDialog(string title, string[] paramNames, string[] paramDefaults, string[][] parameterOptions)	parameterOptions 参数决定对应参数的可选项，如果有可选项，对话框就以下拉列表框控件显示该参数。
Schema 画面相关操作	
public Canvas GetSchemaCanvas(object element)	获取画面的画布 参数 element 一般为控件事件的 sender 参数。
public FrameworkElement GetSchemaElement (object element, string name)	获取画面中指定名称的元素 参数 element 一般为控件事件的 sender 参数; 参数 name 是画面中元素的名称。
public T GetSchemaElement<T> (object element, string name)	同上。
public void OpenSchema(string name)	打开指定名称的画面。 参数 name 是画面的名称。
public void CloseSchema(string name)	关闭指定名称的画面。
public void CloseSchema()	关闭当前激活状态的画面。
设备相关操作	
public void ShowDeviceDialog(string name)	弹出设备属性设置对话框。 参数 name 是设备名称。
视图和编辑器相关操作	
public void OpenView(String viewId)	打开指定 Id 的视图。 参数 viewId 的值定义如下： 项目管理器： Workbench.View.ProjectExplorer 用户管理器： Workbench.View.UserExplorer 日志视图：Workbench.View.Log 工具箱视图： Workbench.View.ToolBox 属性视图： Workbench.View.Properties
public void CloseView(String viewId)	关闭指定 Id 的视图。
public void CloseAllViews()	关闭所有视图。
public void CloseEditor(string uri)	关闭 uri 所指示的编辑器。
public void CloseEditors()	关闭当前项目所有编辑器。
public void CloseAllEditors()	关闭所有编辑器。
public void ShowEditorHeaders()	显示编辑器页面的标签头。
public void HideEditorHeaders()	隐藏编辑器页面的标签头。
public void OpenStepDataEditor()	打开当前项目序列数据。
public void CloseStepDataEditor()	关闭当前项目序列数据。
public void OpenVariantDataEditor(string path)	打开当前项目指定路径的变量数据编辑器。

	参数 path 是变量的路径（路径格式： Name [/Name]，是以/分隔的 Name 字符串，由于变量同级不允许同名，故能保证路径唯一。）。
public void CloseVariantDataEditor(string path)	关闭当前项目指定路径的变量数据。 参数 path 定义同上。
public void ShowFullScreen(bool fullScreen)	显示/关闭全屏状态。
工具栏和状态栏相关操作	
public void ShowToolBar()	显示工具栏。
public void HideToolBar()	隐藏工具栏。
public void MinimizeToolBar()	最小化工具栏。
public bool IsToolBarMinimized()	获取工具栏当前最小化状态。
public bool IsToolBarVisible()	获取工具栏当前可视状态。
public void ShowStatusBar()	显示状态栏。
public void HideStatusBar()	隐藏状态栏。
public bool IsStatusBarVisible()	获取状态栏当前可视状态。
用户相关操作	
public void Login()	执行登录命令。
public void Logout()	执行注销命令。
public string GetUserName()	获取登录用户名。
public void LockUser()	锁定登录用户。
public void Shutdown()	执行退出应用程序命令。

3.3.4.2 设备会话 IDeviceSession

定义	描述
string Type { get; }	设备类型，对应 Device 的 Id。
string Name { get; set; }	设备会话的逻辑名称。
string Address { get; set; }	设备会话的地址，如常规串口的 COM1，VISA 串口的 ASRL1::INSTR。
string Parameters { get; set; }	参数格式： [parameter=string][,...]
string Description { get; set; }	设备会话描述。
int Timeout { get; set; }	操作的最小超时时间， TMO_INFINITE=-1 表示无限超时， TMO_IMMEDIATE=0 表示操作立即返回，不等设备返回数据。
bool State { get; }	当前会话是否打开，true 为打开，false 为关闭。
object GetParameter(string name)	获取会话参数。
T GetParameter<T>(string name)	获取会话参数。
bool SetParameter(string name, object val)	设置会话参数。

bool Open()	打开设备会话。
void Close()	关闭设备会话。

3.3.4.3消息型设备会话 IMessageDeviceSession

定义	描述
int BufferSize { get; set; }	设备会话的缓存大小。
Encoding Encoding { get; set; }	消息编码方式，用户对字符串读写的编码。
int Read(byte[] data, int offset, int count)	从 offset 位置开始，最大读取会话的 count 个数据到 data 中。
int Read(char[] data, int offset, int count)	从 offset 位置开始，最大读取会话的 count 个字符到 data 中。
int Write(byte[] data, int offset, int count)	从 offset 位置开始，最大写入 count 个字节到会话中。
int Write(char[] data, int offset, int count)	从 offset 位置开始，最大写入 count 个字符到会话中。
int Write(string data)	将 data 字符串写入会话。
int Write(string format, object arg0)	按格式字符串要求写入会话。
int Write(string format, object arg0, object arg1)	按格式字符串要求写入会话。
int Write(string format, object arg0, object arg1, object arg2)	按格式字符串要求写入会话。
int Write(string format, params object[] args)	按格式字符串要求写入会话。
void Flush()	清空缓存。

3.3.4.4步骤上下文 IStepContext

定义	描述
IStep Step { get; }	本次运行的入口步骤。
VariantContainer Variants { get; }	执行步骤所在的项目的变量集。
long Times { get; }	运行总次数。
long CurrentTimes { get; }	当前运行的次数
Result[] Results { get; }	当前运行的结果集，多 socket 时有多个，单 socket 时只有一个。
bool Power { get; set; }	启动运行或停止。
表达式相关操作	
ExpressionContext CreateExpressionContext()	创建表达式计算上下文。
IGenericExpression<T> CreateGenericExpression<T>(string expression)	创建泛型表达式。
IDynamicExpression CreateDynamicExpression(string expression)	创建动态表达式。
T EvaluateExpression<T>(ExpressionContext context, string expression)	通过传入的上下文计算泛型表达式。
T EvaluateExpression<T>(string expression)	计算表达式。

object EvaluateExpression(ExpressionContext context, string expression)	通过传入的上下文计算表达式。
object EvaluateExpression(string expression)	计算表达式。
设备会话相关操作	
IEnumerable<IDeviceSession> DeviceSessions { get; }	获取项目的设备会话集。
T GetDeviceSession<T>(string name)	根据设备名获取项目的设备会话。
T GetDeviceSession<T>(IStep step)	根据设备名获取项目的设备会话。
void FlushDeviceSession(string name)	清除设备缓存。
数据操作（该数据表为 key-value 值对，用于用户自定义数据，作用域是执行步骤）	
IDictionary<string, object> Datas { get; }	数据集。
object GetData(string name)	获取数据值。
T GetData<T>(string name)	获取数据值。
void SetData(string name, object value)	设置或添加数据到数据表。
bool ContainData(string name)	检查是否有数据存在。
bool RemoveData(string name)	删除数据。
bool TryGetData(string name, out object value)	尝试获取数据值。
bool TryGetData<T>(string name, out T value)	尝试获取数据值。
void ClearDatas()	清除所有数据。

3.3.4.5步骤 IStep

定义	描述
string Id { get; }	步骤 Id
string Name { get; set; }	步骤名称
StepType Type { get; }	步骤类型
string Category { get; set; }	步骤类别
string Description { get; set; }	步骤描述
bool Active { get; set; }	是否激活状态，不激活状态的步骤不会执行。
bool Loggable { get; set; }	是否打开记录开关，不记录则运行过程和结果不记录。
int Socket { get; set; }	执行槽，用于多槽执行分开记录。>=0 表示需要分槽记录，<0 表示所有槽都有记录。
string Script { get; set; }	步骤的脚本
int Timeout { get; set; }	超时时间，单位 ms
IStep Parent { get; }	父步骤
StepResult Result { get; }	步骤的结果
StepState State { get; }	步骤的运行状态

3.3.4.6步骤结果 StepResult

定义	描述
public Guid Id { get; }	结果 Id
public string SN { get; set; }	结果编号
public string Name { get; }	结果名称

public string StepId	产生结果的步骤的 Id
public StepType StepType	产生结果的步骤的类型
string Category { get; set; }	产生结果的步骤的类别
string Description { get; set; }	结果描述
public DateTime StartTime{ get; set; }	产生结果的步骤的开始执行时间
public DateTime EndTime{ get; set; }	产生结果的步骤的结束执行时间
public int Socket { get; set; }	结果所在的槽号，动作类型的步骤有效。
public string Device { get; set; }	产生结果的步骤的设备名，动作类型的步骤有效。
public Variant Data { get; set; }	数据值，步骤运行结束时在执行引擎中设置，值类型步骤存结果数值，协议类型步骤存原始数据帧。
public VariantContainer DataFields { get; }	数据字段集，即已经经过解析的数据，步骤运行结束时在执行引擎中设置，协议类型步骤存解析完的协议单元数据。
public string DataLimits { get; set; }	结果值的限制，从产生结果的步骤中获取，也可以在脚本中设置。
public int Status { get; set; }	结果状态值，结果状态在不违反 ResultStatus 值的基础上，可以自定义，用于分 Bin 用途，通常大于 0 用于通过分 Bin，小于 0 用于失效分 Bin。如果有特殊需求，结果状态值可以在脚本的 EndExecute 方法中更改执行引擎的设置。

3.3.4.7结果状态 ResultStatus

定义	描述
Done = Int32.MaxValue	完成态，执行步骤执行完毕，无限制值
Passed = 1	结果通过限制值检查
None = 0	无状态，初始态或者运行态
Failed = -1	结果不能通过限制值检查
Terminated = Int32.MinValue	终止态，执行步骤未执行完毕被终止，无限制值

3.3.4.8数据库管理器 IDataManager

定义	描述
通用增删改查	
bool Add<T>(T entity) where T : class	新增
bool Update<T>(T entity) where T : class	更新
bool Delete<T>(T entity) where T : class	删除
bool DeleteByConditon<T>(Expression<Func<T, bool>> deleWhere) where T : class	根据条件删除
T GetSingleById<T>(int id) where T : class	查找单个
T GetSingle<T>(Expression<Func<T, bool>> seleWhere) where T : class	查找单个

List<T> GetAll<T>() where T : class	获取所有实体集合
List<T> GetAll<T, Tkey>(Expression<Func<T, Tkey>> orderWhere, bool isDesc) where T : class	获取所有实体集合(单个排序)
List<T> GetAll<T>(params DataOrder[] orderByExpression) where T : class	获取所有实体集合(多个排序)
IQueryable<T> CommonSort<T, Tkey>(IQueryable<T> data, Expression<Func<T, Tkey>> orderWhere, bool isDesc) where T : class	单个排序通用方法
IQueryable<T> CommonSort<T>(IQueryable<T> data, params DataOrder[] orderByExpression) where T : class	多个排序通用方法
List<T> GetList<T>(Expression<Func<T, bool>> seleWhere) where T : class	根据条件查询实体集合，查询条件 seleWhere 是 lambel 表达式
List<T> GetList<T, TValue>(Expression<Func<T, TValue>> seleWhere, IEnumerable<TValue> conditions) where T : class	根据条件查询实体集合，查询条件 seleWhere 是 lambel 表达式
List<T> GetList<T, Tkey>(Expression<Func<T, bool>> seleWhere, Expression<Func<T, Tkey>> orderWhere, bool isDesc) where T : class	根据条件查询实体集合，查询条件 seleWhere 是 lambel 表达式
List<T> GetList<T>(Expression<Func<T, bool>> seleWhere, params DataOrder[] orderByExpression) where T : class	根据条件查询实体集合(多个字段排序)，查询条件 seleWhere 是 lambel 表达式
List<T> GetListPaged<T, Tkey>(int pageIndex, int pageSize, out int totalcount) where T : class	获取分页集合(无条件无排序)
List<T> GetListPaged<T, Tkey>(int pageIndex, int pageSize, Expression<Func<T, Tkey>> orderWhere, bool isDesc, out int totalcount) where T : class	获取分页集合(无条件单个排序)
List<T> GetListPaged<T>(int pageIndex, int pageSize, out int totalcount, params DataOrder[] orderByExpression) where T : class	获取分页集合(无条件多字段排序)
List<T> GetListPaged<T, Tkey>(int pageIndex, int pageSize, Expression<Func<T, bool>> seleWhere, out int totalcount) where T : class	获取分页集合(有条件无排序)
List<T> GetListPaged<T, Tkey>(int pageIndex, int pageSize, Expression<Func<T, bool>> seleWhere, Expression<Func<T, Tkey>> orderWhere, bool isDesc, out int totalcount) where T : class	获取分页集合(有条件单个排序)
List<T> GetListPaged<T>(int pageIndex, int pageSize, Expression<Func<T, bool>> seleWhere, out int totalcount, params DataOrder[] orderModelFiled) where T : class	获取分页集合(有条件多字段排序)
原始 sql 操作	
int ExecuteSql(string sql, params object[] paras)	执行操作 Sql 语句
List<T> QueryList<T>(string sql, params object[] paras)	查询列表数据
T QuerySingle<T>(string sql, params object[] paras)	查询单个数据
object QuerySingle(Type elementType, string sql, params object[] paras)	查询单个数据
void ExecuteTransaction(List<String> lsSql,	执行事务

List<Object[]> lsParas)	
通用属性	
DateTime ServerTime { get; }	获取数据库服务器当前时间。
String DatabaseVersion { get; }	获取数据库版本。

3.3.4.9对象存储 IMemento

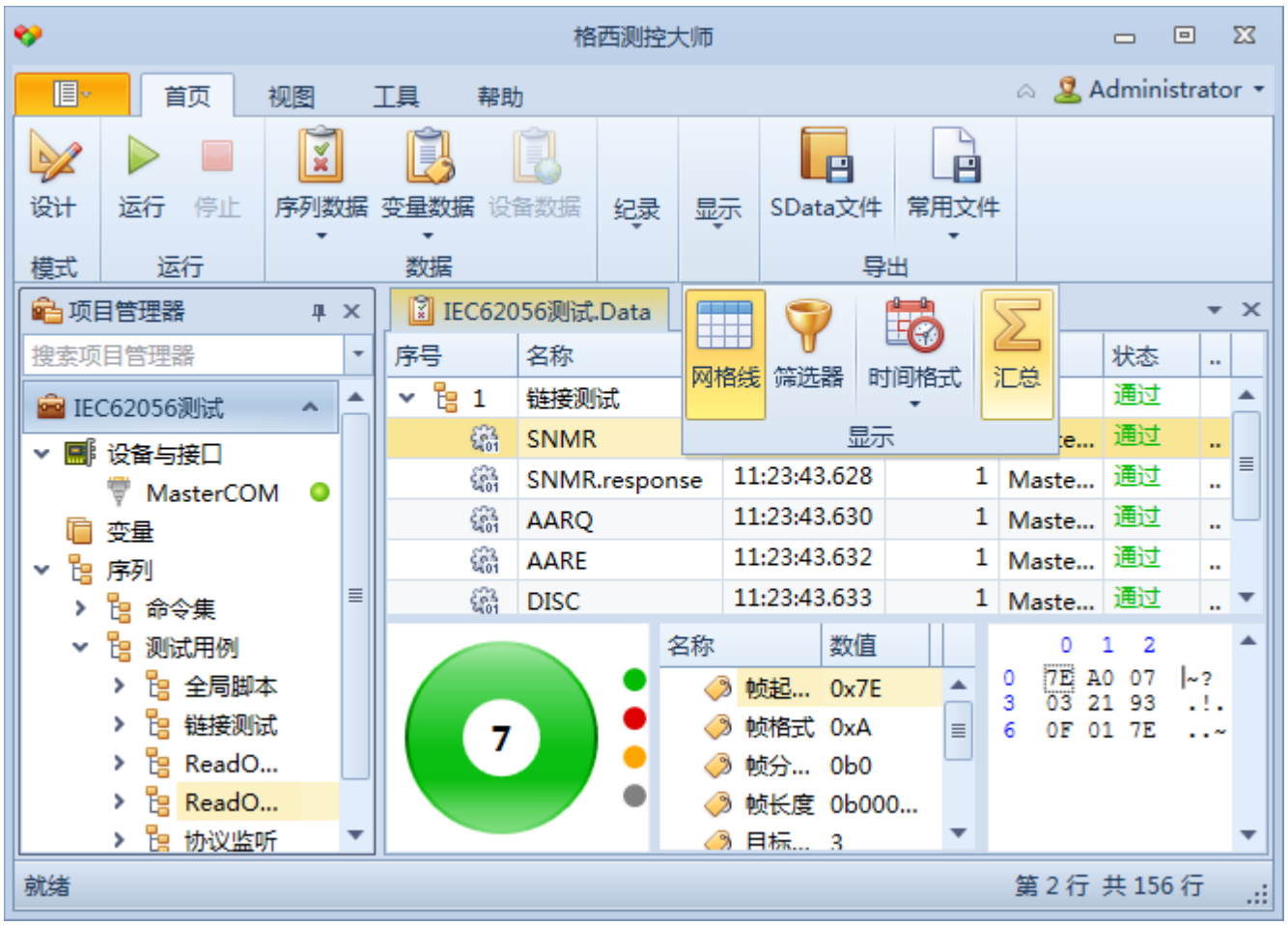
定义	描述
IMemento CreateChild(String name)	创建指定名称的子节点
IMemento GetChild(String name)	获取指定名称的第一个子节点
IMemento[] GetChildren()	获取所有子节点
IMemento[] GetChildren(String name)	根据名称获取子节点集
int GetChildrenCount()	获取子节点的数量
float GetFloat(String key)	获取 key 名称的属性的 float 值
double GetDouble(String key)	获取 key 名称的属性的 double 值
int GetInteger(String key)	获取 key 名称的属性的 int 值
Int64 GetInteger64(String key)	获取 key 名称的属性的 Int64 值
String GetString(String key)	获取 key 名称的属性的字符串值
Boolean GetBoolean(String key)	获取 key 名称的属性的布尔值
String GetTextData()	获取 Text 节点值
String GetCData()	获取 Cdata 节点值
void PutFloat(String key, float value)	设置 key 名称的属性为 float 值
void PutDouble(String key, double value)	设置 key 名称的属性为 double 值
void PutInteger(String key, int value)	设置 key 名称的属性为 int 值
void PutInteger64(String key, Int64 value)	设置 key 名称的属性为 Int64 值
void PutString(String key, String value)	设置 key 名称的属性为字符串值
void PutBoolean(String key, bool value)	设置 key 名称的属性为布尔值
void PutTextData(String data)	设置 Text 节点的值
void PutCData(String data)	设置 Cdata 节点的值
bool Contain(String key)	检查是否有 key 名称对应的属性
bool ContainTextData()	检查是否有 Text 节点
bool ContainCData()	检查是否有 CData 节点

3.3.5 步骤模版

软件支持步骤模版功能，用户可以把常用的步骤保存为模版，使用的时候直接从模板库拖放到步骤编辑器即可创建。

3.3.6 步骤数据的采集与回放

软件支持在运行时对步骤数据进行显示、采集和回放。运行时，点击工具栏的“序列数据”，打开当前项目的序列数据页面，如图所示。



点击工具栏的“显示”-> “汇总”，可以切换到汇总页面，分类统计步骤的执行信息，如图所示。

3.4 画面

3.4.1 简介

用户可以创建画面，利用画面工具箱的控件、形状模版，实现任意用户界面，满足各种测控界面需求。

软件内置画面编辑器，用户可以很方便的创建和编辑画面，实现画面逻辑。

- 支持属性数据绑定，建立画面元素属性和变量的联系。
- 支持事件脚本，脚本可以无缝调用.Net Framework 类库，调用第三方托管库来实现画面逻辑。
- 支持动态动作，建立画面元素动作（移动、旋转、尺寸）与变量的联系。
- 支持控件模板和形状模版。

关于画面的用法例子，请参考：<软件安装目录>\Examples\Basics\Schemas。

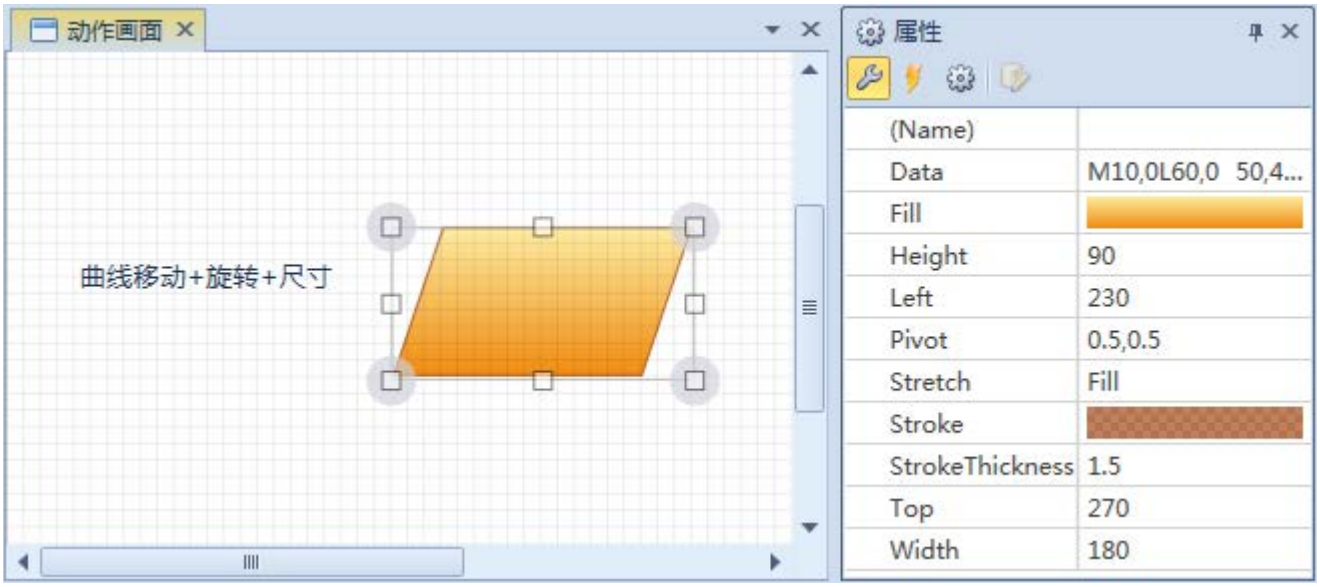
3.4.2 画面元素

画面元素分为两类，一类是通过类型创建的控件类，另一类是通过 Xaml 创建的形状类。两者都可以通过画面工具箱的模版进行创建。

使用画面编辑器工具栏的“指针”工具，可以选择画面元素，对元素属性、事件和动作进行编辑。

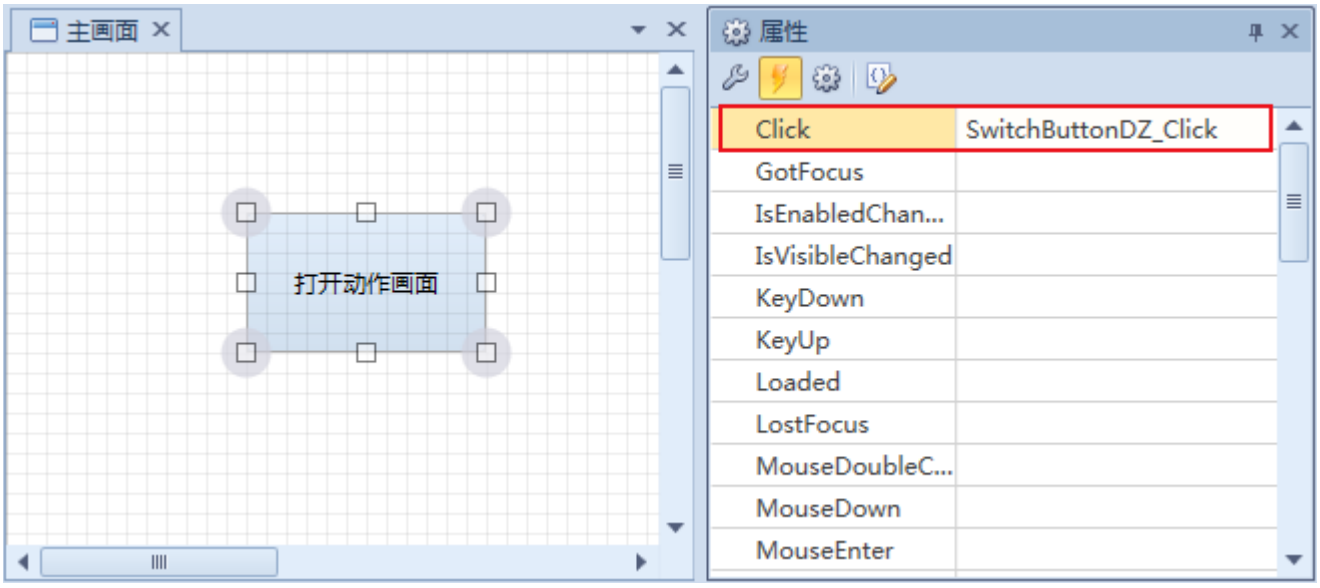
3.4.2.1 属性

选中画面元素之后，点击属性面板上的“属性”按钮，即可显示所有与该元素相关的属性，用户可以在面板上编辑元素的属性。



3.4.2.2 事件

选中画面元素之后，点击属性面板上的“事件”按钮，即可显示所有与该元素相关的事件，用户可以在面板上双击事件条目添加事件脚本。

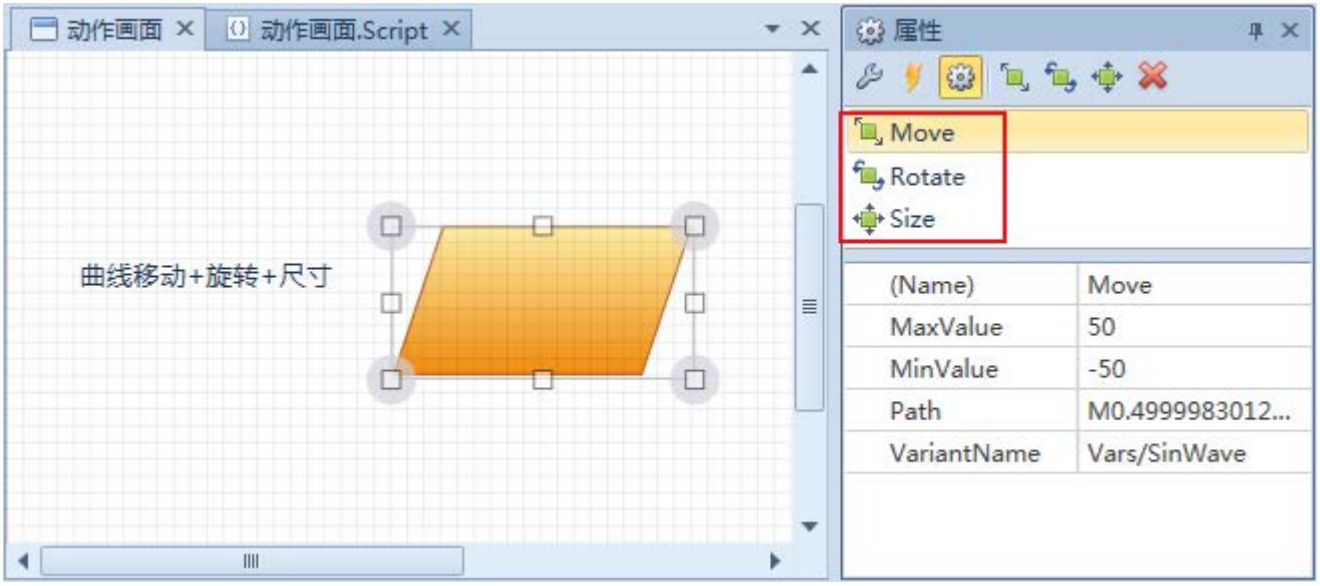


主画面 × 主画面.Script ×

```
1   using System;
2   using FlexBench;
3   using FlexBench.Scripting;
4   using FlexBench.Workbench;
5   public class Schema_75AAE89EC31748338A78FB356E624B67
6   {
7       public ScriptContext Context { get; set; }
8
9       //
10      public void SwitchButtonDZ_Click(Object sender, System.Windows.RoutedEventArgs e)
11      {
12          this.Context.OpenSchema("动作画面");
13      }
14  }
```

3.4.2.3 动作

选中画面元素之后，点击属性面板上的“动作”按钮，即可显示所有与该元素相关的动作，软件支持添加“移动动作”、“旋转动作”和“尺寸动作”三种动作。



移动动作

定义	描述
(Name)	名称
MaxValue	绑定变量允许变化的最大值。
MinValue	绑定变量允许变化的最小值。
Path	移动的路径，可以预先使用“折线”等工具画好路径，然后再选择即可。画面元素在绑定变量的值变化时，按照变量值占[MinValue,MaxValue]范围的百分比，移动到路径对应百分比的地方。

VariantName	绑定的变量名

旋转动作

定义	描述
(Name)	名称
MaxAngle	转动的最大角度。
MaxValue	绑定变量允许变化的最大值。
MinAngle	转动的最小角度。
MinValue	绑定变量允许变化的最小值。
VariantName	绑定的变量名

画面元素在绑定变量的值变化时，按照变量值占[MinValue,MaxValue]范围的百分比，旋转到[MinAngle,MaxAngle]角度范围对应百分比的地方。

尺寸动作

定义	描述
(Name)	名称
MaxSizePercent	最大尺寸时占元素原来大小百分比。
MaxValue	绑定变量允许变化的最大值。
MinSizePercent	最小尺寸时占元素原来大小百分比。
MinValue	绑定变量允许变化的最小值。
SizeMode	尺寸变化的模式。 软件支持 Up、Down、Left、Right、UpDown、LeftRight、UpDownLeftRight、UpLeft、UpRight、DownLeft、DownRight、LeftRightUp、LeftRightDown、UpDownLeft、UpDownRight。
VariantName	绑定的变量名

画面元素在绑定变量的值变化时，按照变量值占[MinValue,MaxValue]范围的百分比，尺寸变化到[MinSizePercent,MzxSizePercent]范围对应百分比的地方。

3.4.3 画面脚本

画面支持脚本功能，通过事件脚本来处理画面元素的事件，脚本可以调用公共的接口，可以无缝调用.Net Framework 类库，可以调用第三方托管库来实现执行逻辑。

下面是 C#版本的步骤脚本模版，由一个.NET 类 Schema_<画面 Id>构成，该类包含一个类型为 ScriptContext 的 Context 属性。

```
/**
 * 命名空间定义
 */
using System;
using Genesis;
using Genesis.Scripting;
using Genesis.Sequence;
using Genesis.Workbench;
```

```
/**
 * 步骤脚本类
 */
public class Schema_75AAE89EC31748338A78FB356E624B67
{
  /**
   脚本上下文
   */
  public ScriptContext Context { get; set; }

  // SwitchButtonDZ 按钮的点击事件
  public void SwitchButtonDZ_Click(Object sender, System.Windows.RoutedEventArgs e)
  {
    this.Context.OpenSchema("动作画面");
  }
}
```

3.4.4 画面模版

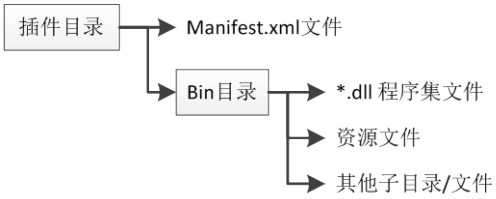
软件支持画面模版功能，提供了常用的控件类模版和形状类模版，模版存放在<软件安装目录>\Templates\Schemas 目录下面。

3.5 插件

插件是一个具备物理隔离性、完全重用的功能模块。本软件的插件由 Manifest.xml 文件、类和资源组成，Manifest.xml 是插件的描述文件，位于插件目录的根目录下，类即插件的类型空间，由插件自身的程序集和依赖的插件/程序集组成，资源即由插件本身的资源和依赖的资源组成。

3.5.1 插件目录结构

软件所有的插件，都存放在<软件安装目录>\Plugins 目录中，插件的标准目录结构如下：



3.5.2 插件清单文件 Manifest.xml

插件清单文件（Manifest.xml）位于模块标准目录结构的根目录之下，定义模块的基本信息、模块激活信息、模块类加载相关的运行时信息、服务定义信息、模块扩展定义信息以及模块详细信息。

本例子为软件 Genesis.Data.Repositories 插件的清单文件。

```
<?xml version="1.0" encoding="utf-8"?>
<Bundle xmlns="urn:schemas-geshe-com:osgi:bundle" Name="Genesis.Data.Repositories"
SymbolicName="Genesis.Data.Repositories" Version="1.0.0.0" InitialState="Active">
  <!— 插件激活器，用于在插件启动或停止时执行相关操作，无此需求可以不配置。 -->
  <Activator Type="Genesis.Data.Repositories.Activator" Policy="Immediate" />
```

```
  <Runtime>
    <!— 本插件.Net程序集，必须配置。 -->
    <Assembly Path="bin\Genesis.Data.Repositories.dll" Share="true" Multiversion="false" />
    <!-- 添加本插件依赖的本地程序集，必须配置。-->
    <Assembly Path="bin\EntityFramework.dll" Share="true" />
    <Assembly Path="bin\EntityFramework.SqlServer.dll" Share="true" />
    <Assembly Path="bin\System.Data.SQLite.dll" Share="true" />
    <Assembly Path="bin\System.Data.SQLite.EF6.dll" Share="true" />
    <Assembly Path="bin\System.Data.SQLite.Linq.dll" Share="true" />
    <Assembly Path="bin\MySql.Data.dll" Share="true" />
    <Assembly Path="bin\MySql.Data.EntityFramework.dll" Share="true" />
    <!-- 添加本插件依赖的其他插件模块，必须配置。 -->
    <Dependency BundleSymbolicName="Genesis.Data" />
  </Runtime>
</Bundle>
```

3.5.3 插件的应用场景

插件可以很方便扩充软件的功能，常见的应用场景：

- ✧ 扩展新的设备和接口类型
- ✧ 扩展新的协议计算算法
- ✧ 扩展新的变量数据分析方法
- ✧ 扩展新的画面控件模版
- ✧ 领域专用模块
- ✧ 企业专用模块

4. 常见问题