上海格西信息科技有限公司

格西测控大师
用户手册

版本 **5.8**

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：

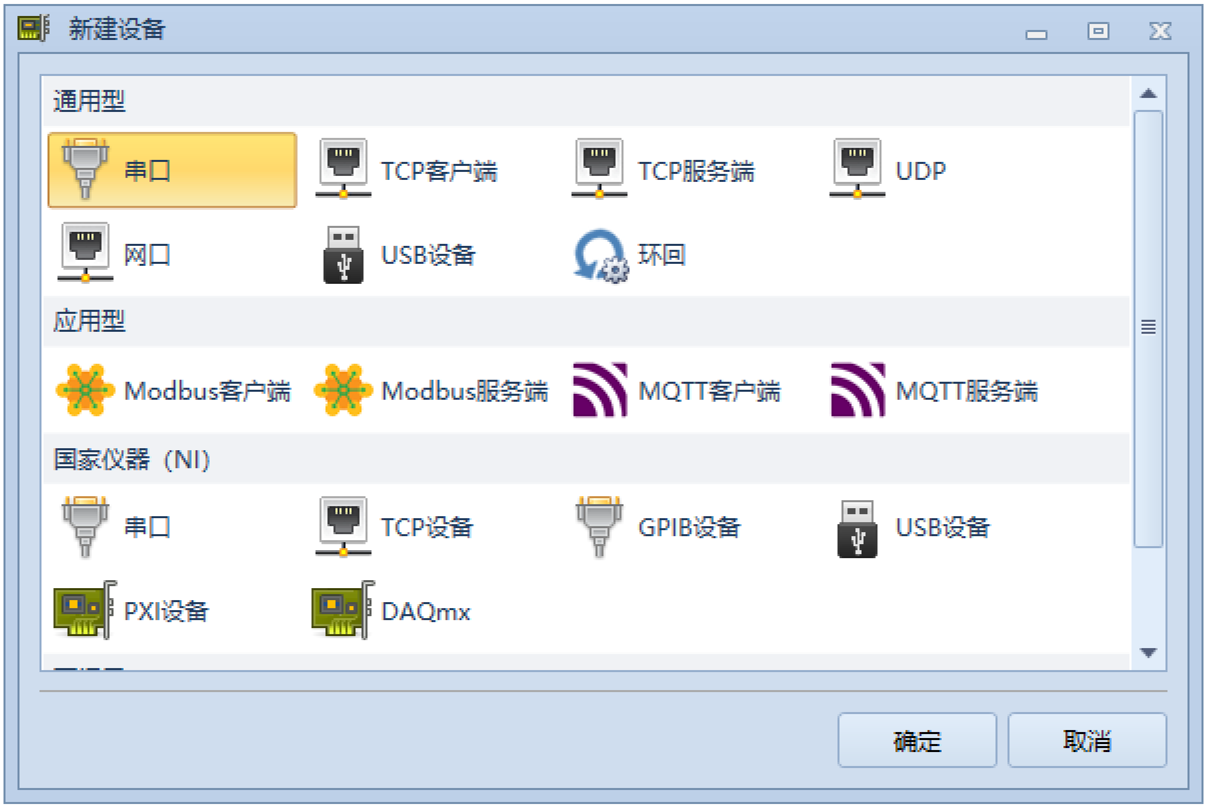
- 简化与加速复杂序列的开发
- 提高代码与测控程序的复用性和可维护性
- 提高测控系统的可扩展性
- 改进测控系统的执行性能

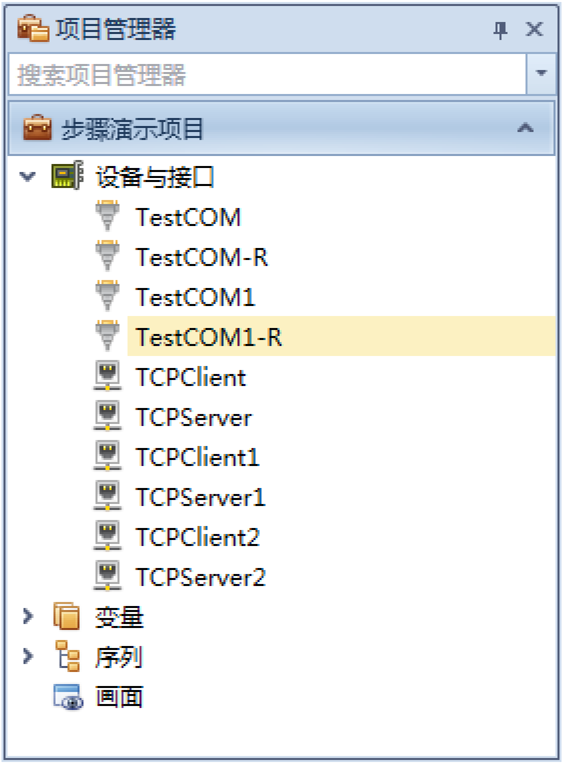
对于任何一个需要加速开发、代码复用、性能改进和自动化的测控项目，例如数据采集和监控系统、设计验证系统、硬件测试系统、芯片测试系统等，格西测控大师都是不可或缺的。

1.2 功能特性

1）自定义设备接口

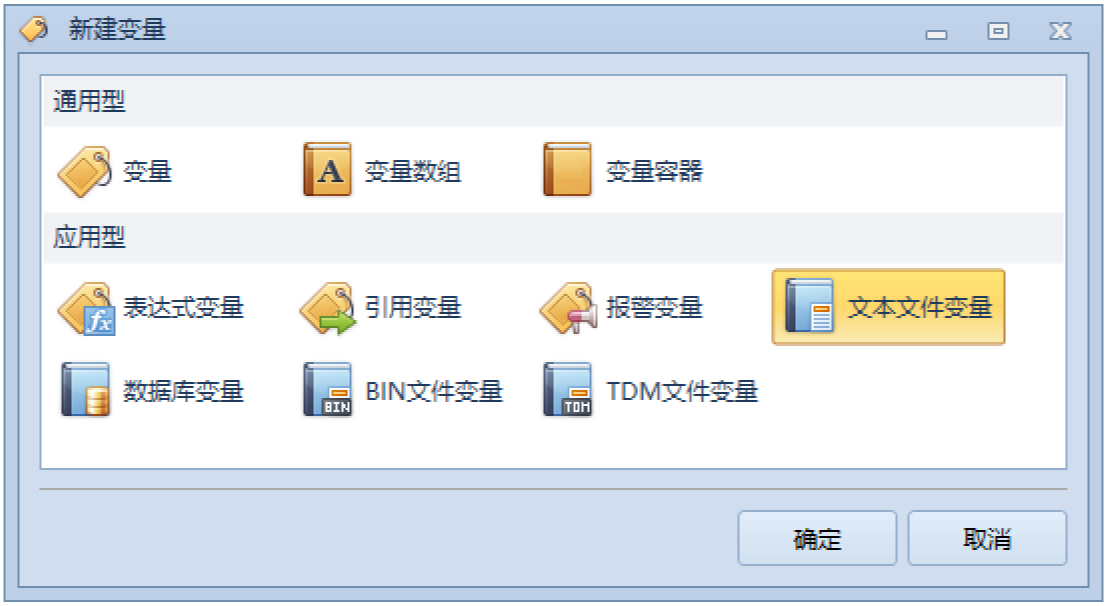
借助软件内置的设备适配器，用户可以创建任意设备和接口组合，可以同时对不同的设备和接口进行通信，满足各种测控连接需求。设备适配器还可以通过插件的方式，让用户扩展新的设备和接口，以满足特殊的连接需求。



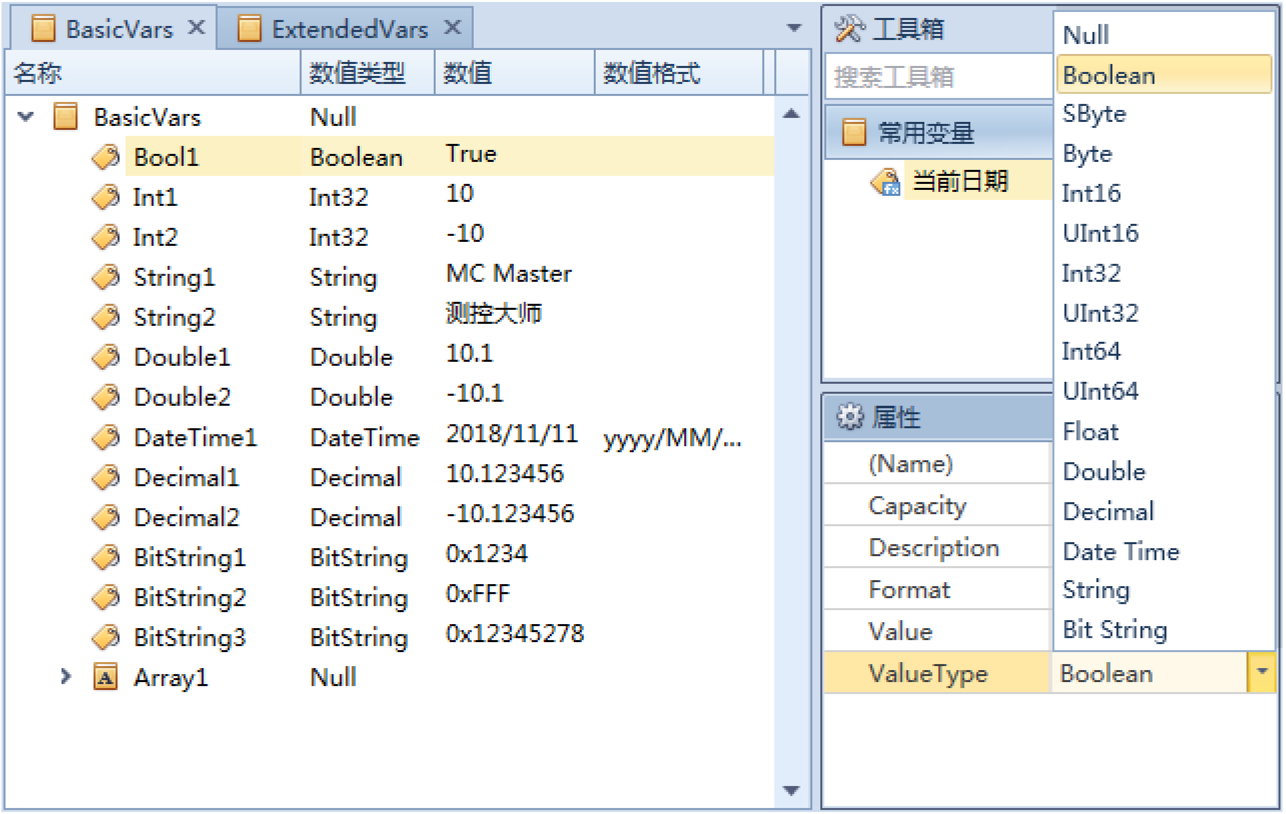


2）自定义变量数据

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、引用变量、报警变量、文本文件变量、数据库变量、BIN 文件变量、TDM 文件变量等应用型变量，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。



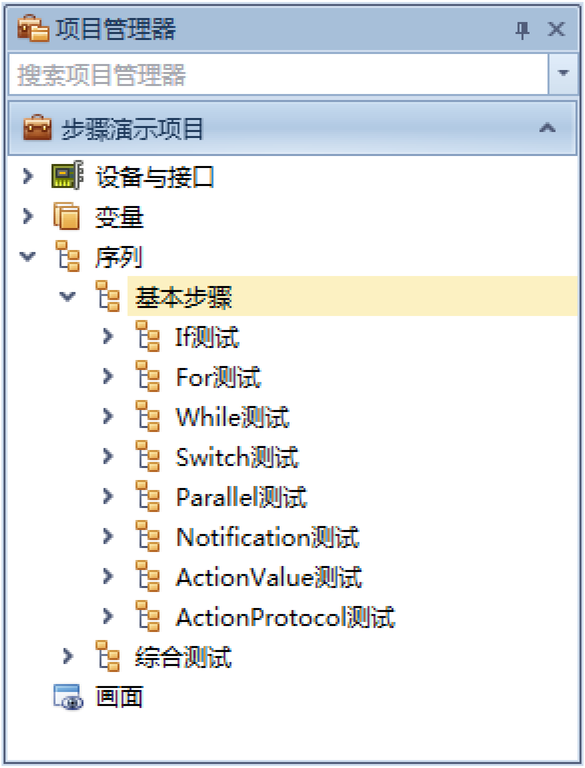
名称	数值类型	数值	数值格式
BasicVars	Null		
Bool1	Boolean	True	
Int1	Int32	10	
Int2	Int32	-10	
String1	String	MC Master	
String2	String	测控大师	
Double1	Double	10.1	
Double2	Double	-10.1	
DateTime1	DateTime	2018/11/11	yyyy/MM/...
Decimal1	Decimal	10.123456	
Decimal2	Decimal	-10.123456	
BitString1	BitString	0x1234	
BitString2	BitString	0xFFF	
BitString3	BitString	0x12345278	
Array1	Null		

(Name)	
Capacity	
Description	
Format	
Value	
ValueType	Boolean

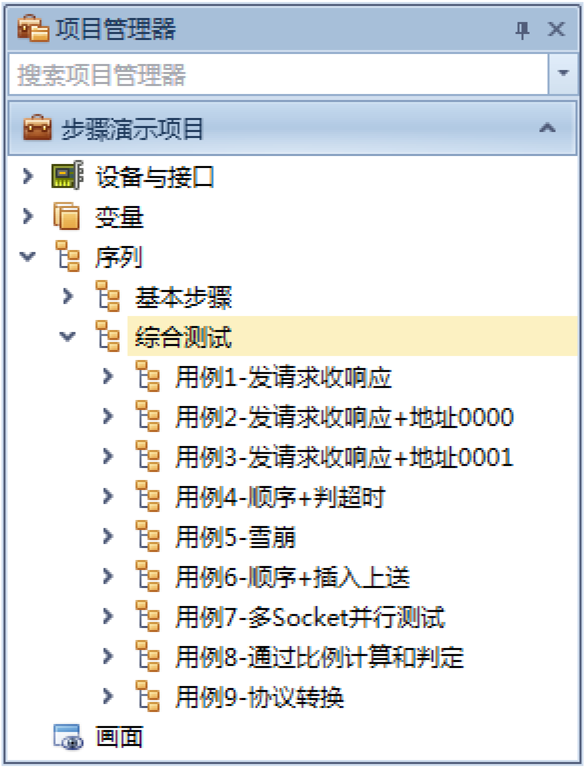
3）自定义执行序列

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）、指示（Indication）。
- 支持数值型动作步骤（Value）、消息型动作步骤（Message）、寄存器型动作步骤（Register）、进程型动作步骤（Process）。
- 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。

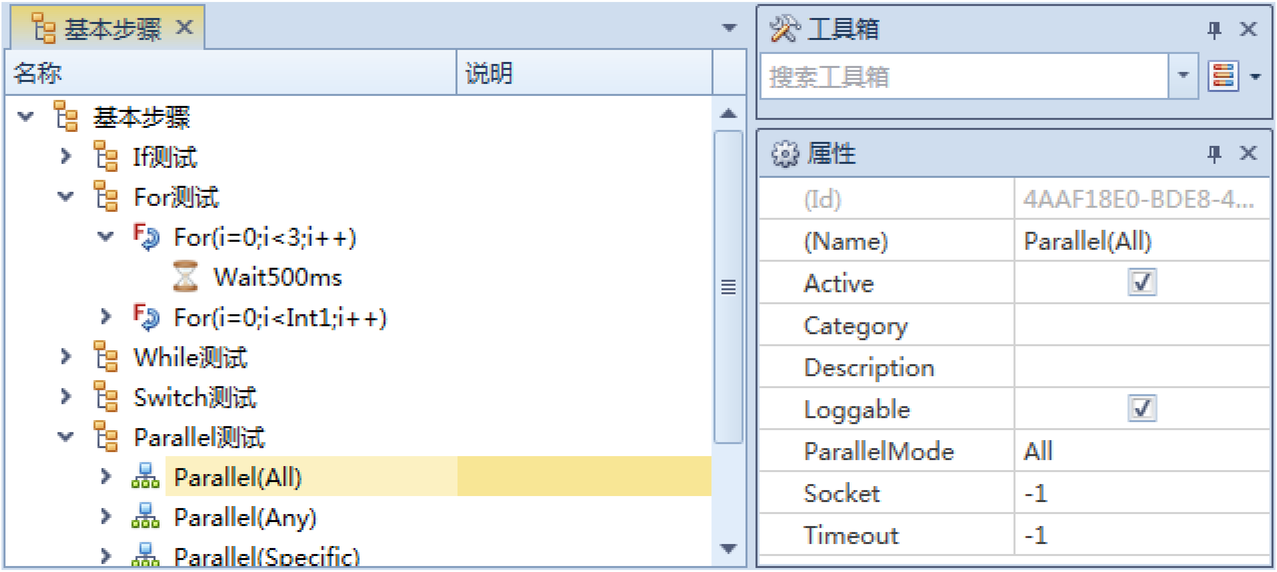


- 设备与接口
- 变量
- 序列
  - 基本步骤
    - If测试
    - For测试
    - While测试
    - Switch测试
    - Parallel测试
    - Notification测试
    - ActionValue测试
    - ActionProtocol测试
  - 综合测试
- 画面



- 设备与接口
- 变量
- 序列
  - 基本步骤
  - 综合测试
    - 用例1-发请求收响应
    - 用例2-发请求收响应+地址0000
    - 用例3-发请求收响应+地址0001
    - 用例4-顺序+判超时
    - 用例5-雪崩
    - 用例6-顺序+插入上送
    - 用例7-多Socket并行测试
    - 用例8-通过比例计算和判定
    - 用例9-协议转换
- 画面

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。

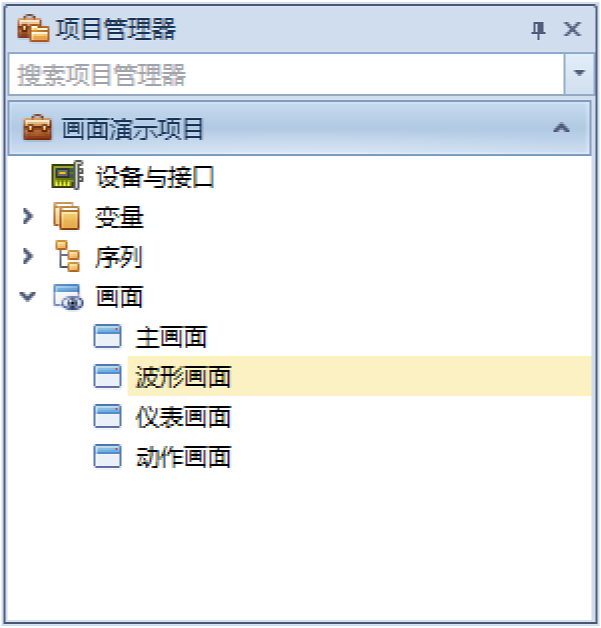


- 基本步骤
  - If测试
  - For测试
    - For(i=0;i<3;i++)
      - Wait500ms
    - For(i=0;i<Int1;i++)
  - While测试
  - Switch测试
  - Parallel测试
    - Parallel(All)
    - Parallel(Any)
    - Parallel(Specific)

(Id)	4AAF18E0-BDE8-4...
(Name)	Parallel(All)
Active	☑
Category	
Description	
Loggable	☑
ParallelMode	All
Socket	-1
Timeout	-1

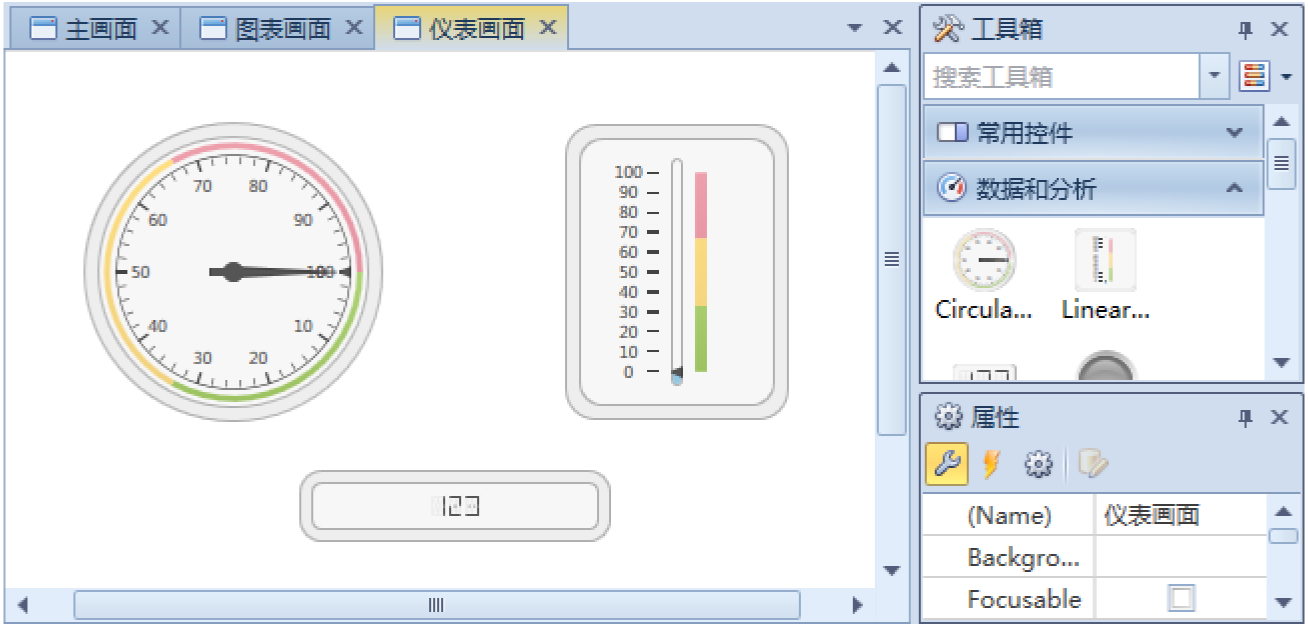
4）自定义用户界面

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。



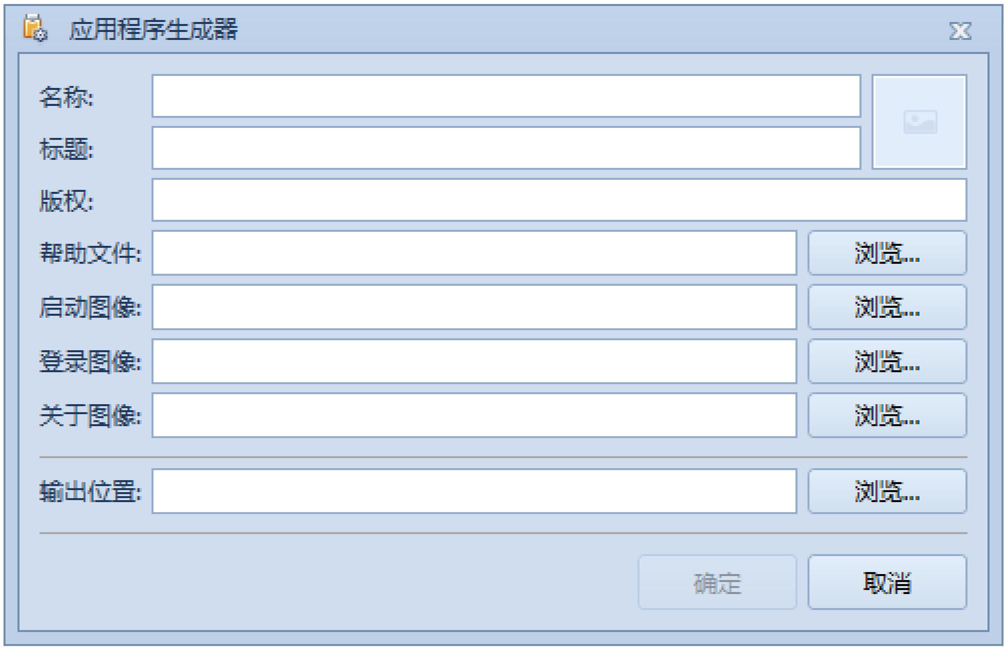
软件内置画面编辑器，用户可以很方便的创建和编辑画面，实现画面逻辑。

- 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、倾斜、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、软件的图标、标题、版权、帮助文件、启动图像、登录图像、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 7 SP1（x86 和 x64）及以上版本

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： 500 MB

必备组件：

- Microsoft .NET Framework 4.8

1.4 产品版本和许可

本产品的授权方式有四种，免费版、基础版、标准版和专业版。其中基础版、标准版和专业版采用硬件加密锁的方式进行许可，免费版不需要硬件加密锁。

基础版：包含运行环境，只能加载和运行由标准版或专业版开发出来的程序，不能开发和发布程序。
标准版：包含开发环境和运行环境，支持开发、调试、运行和发布程序。
专业版：标准版的所有内容以及：数据分析、应用型设备驱动。

免费版：专业版的所有内容，限制资源数（项目 1 个，设备 2 个，变量 16 个，步骤 16 个，画面 2 个），限制运行时间（累计时长 1 小时）。

基础版、标准版和专业版许可细则：

- 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。

免费版许可细则：

- 仅可用于个人、非商业和非工业目的；
- 禁止用于营利性目的。

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。

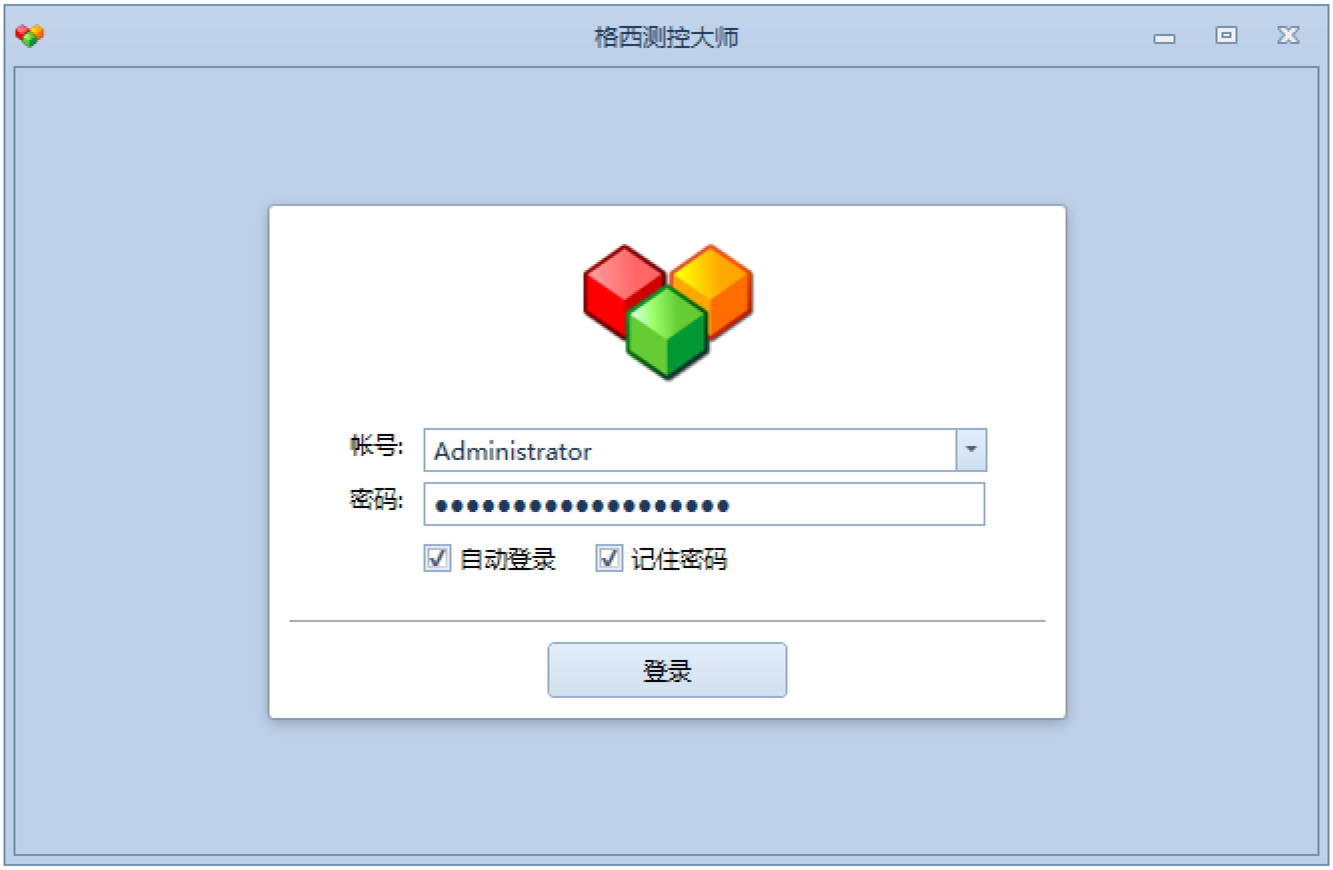
第 3 步 – 执行安装文件
接下来，运行格西测控大师安装文件进行安装。
安装程序依次安装以下程序：

- 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。



帐号: Administrator

密码: ●●●●●●●●●●●●●●●●●●●

☑ 自动登录 ☑ 记住密码

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。

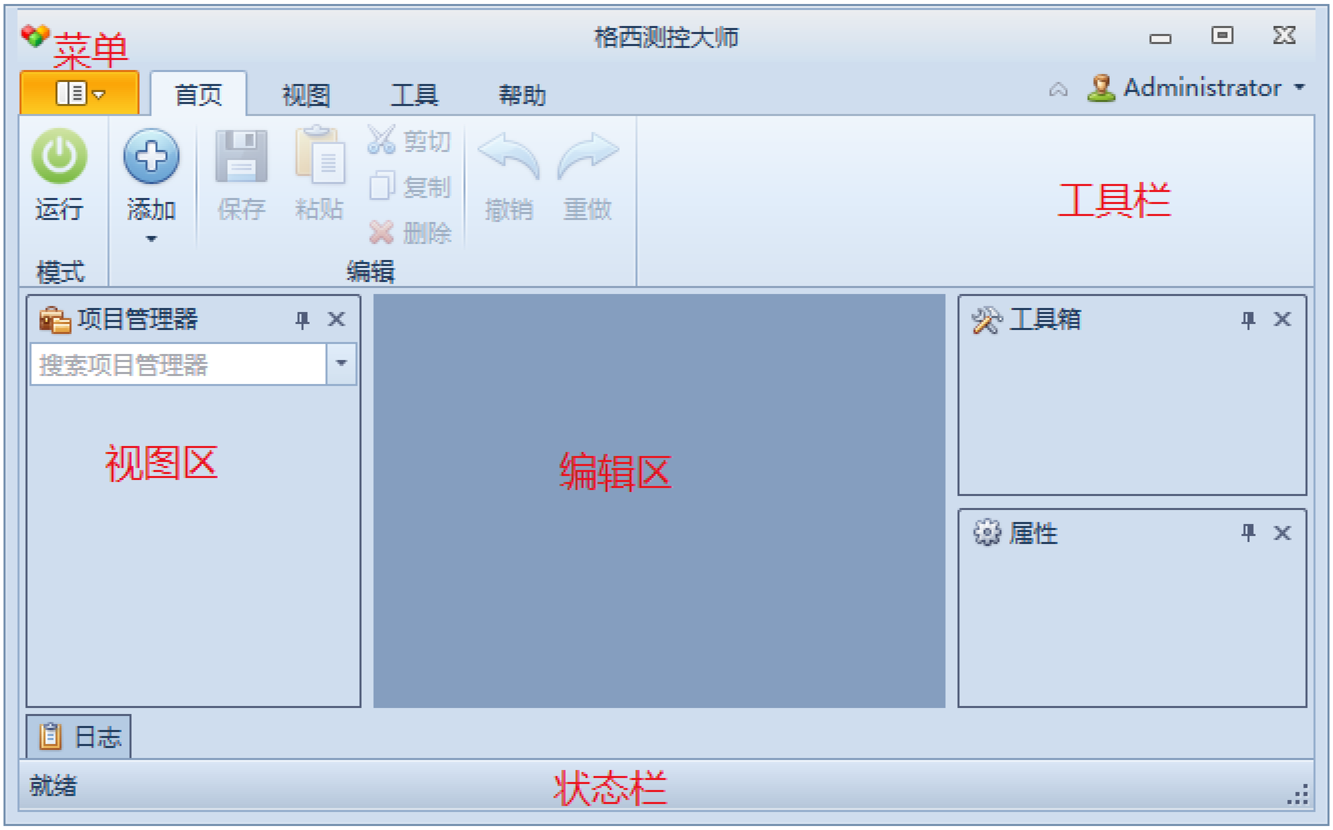
用户名	密码	描述
Administrator	（无）	管理员，拥有所有权限。
Developer	（无）	开发员，拥有除“用户管理”外的所有权限。
Operator	（无）	操作员，只有项目运行权限。

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。

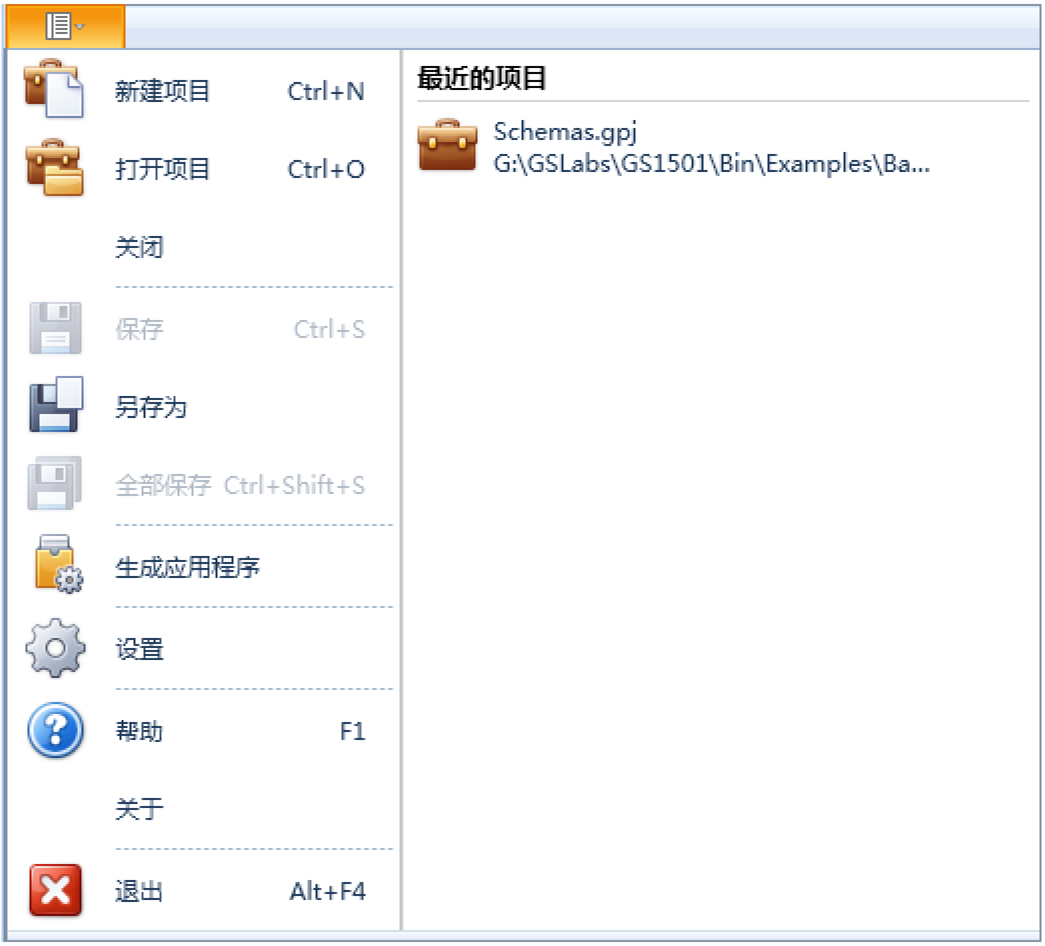
2.3 用户界面

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

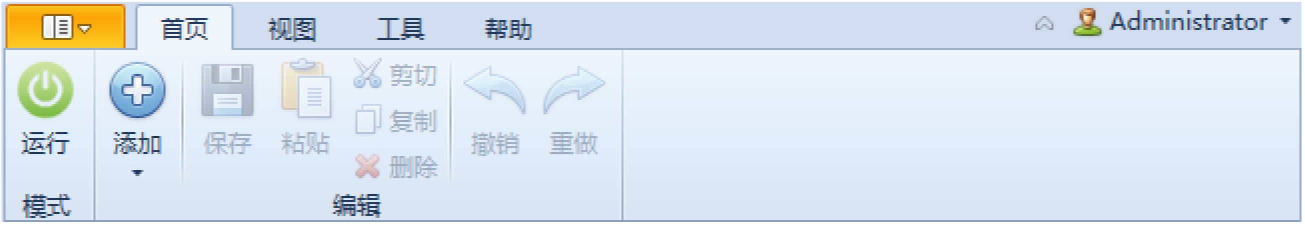


命令	快捷键	功能
新建项目	Ctrl+N	新建一个项目，并建立项目环境。
打开项目	Ctrl+O	打开一个项目，并建立项目环境。
关闭		关闭当前激活的项目。
保存	Ctrl+S	保存当前激活的项目。
另存为		将当前激活的项目保存到指定路径。
全部保存	Ctrl+Shift+S	保存当前打开的所有项目。
生成应用程序		把当前打开的项目打包为一个运行时软件包，供开发完毕的项目进行部署使用。
设置		设置系统参数。
帮助	F1	用户帮助。
关于		显示软件的版权、版本以及注册信息等。
退出	Alt+F4	退出系统。

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
数据管理器	打开数据管理器，用于管理历史序列数据、变量数据和设备数据。
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

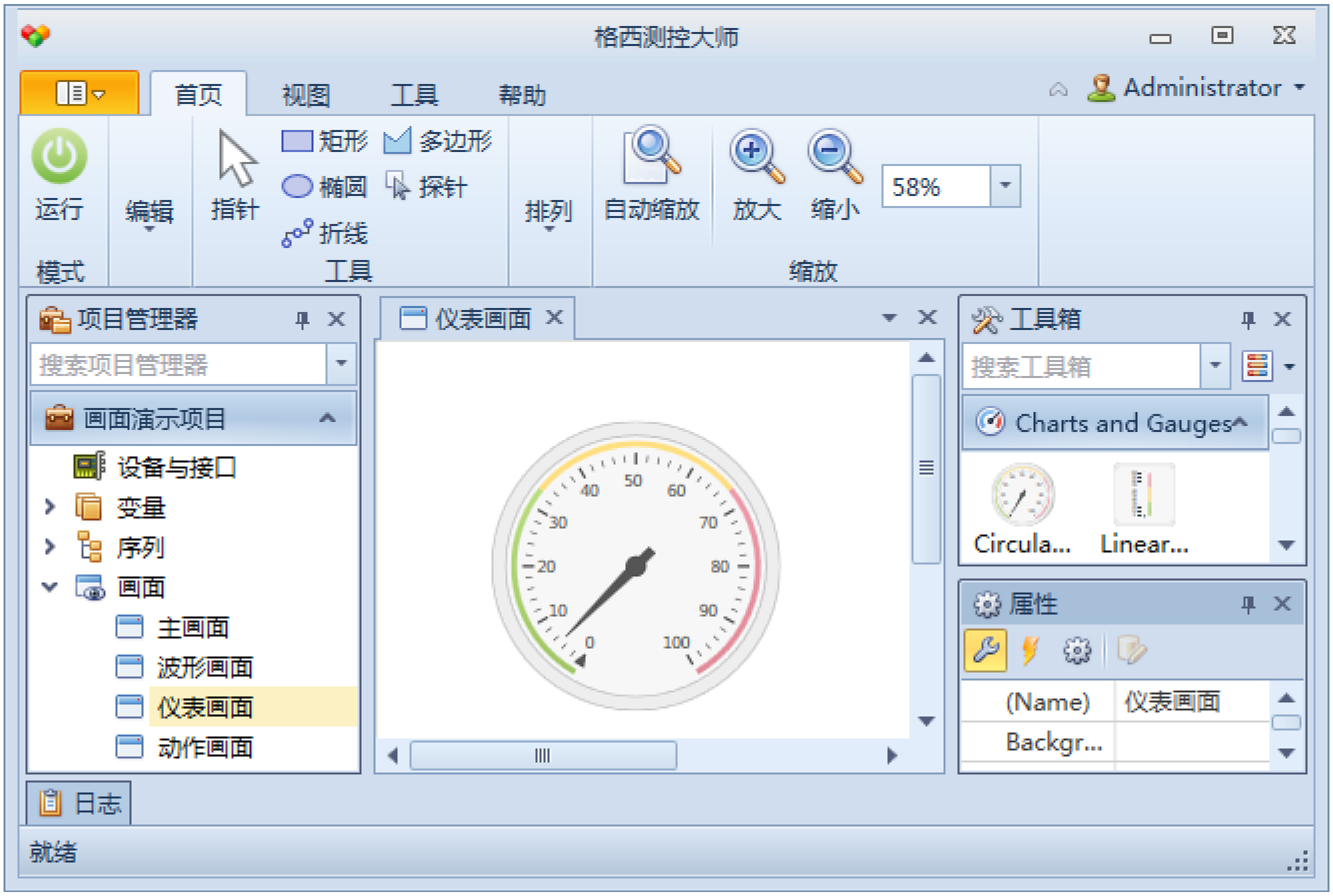
工具工具栏

命令	功能
计算器	运行计算器程序。
校验和	运行校验和计算器。
CRC	运行 CRC 计算器。
DES	运行 DES 计算器。
哈希值	运行哈希值计算器。
Base64	运行 Base64 字符串计算器。
Unicode 转换器	运行 Unicode 转换器。
ASCII 码对照表	运行 ASCII 码对照表。
报表设计器	运行报表设计器。

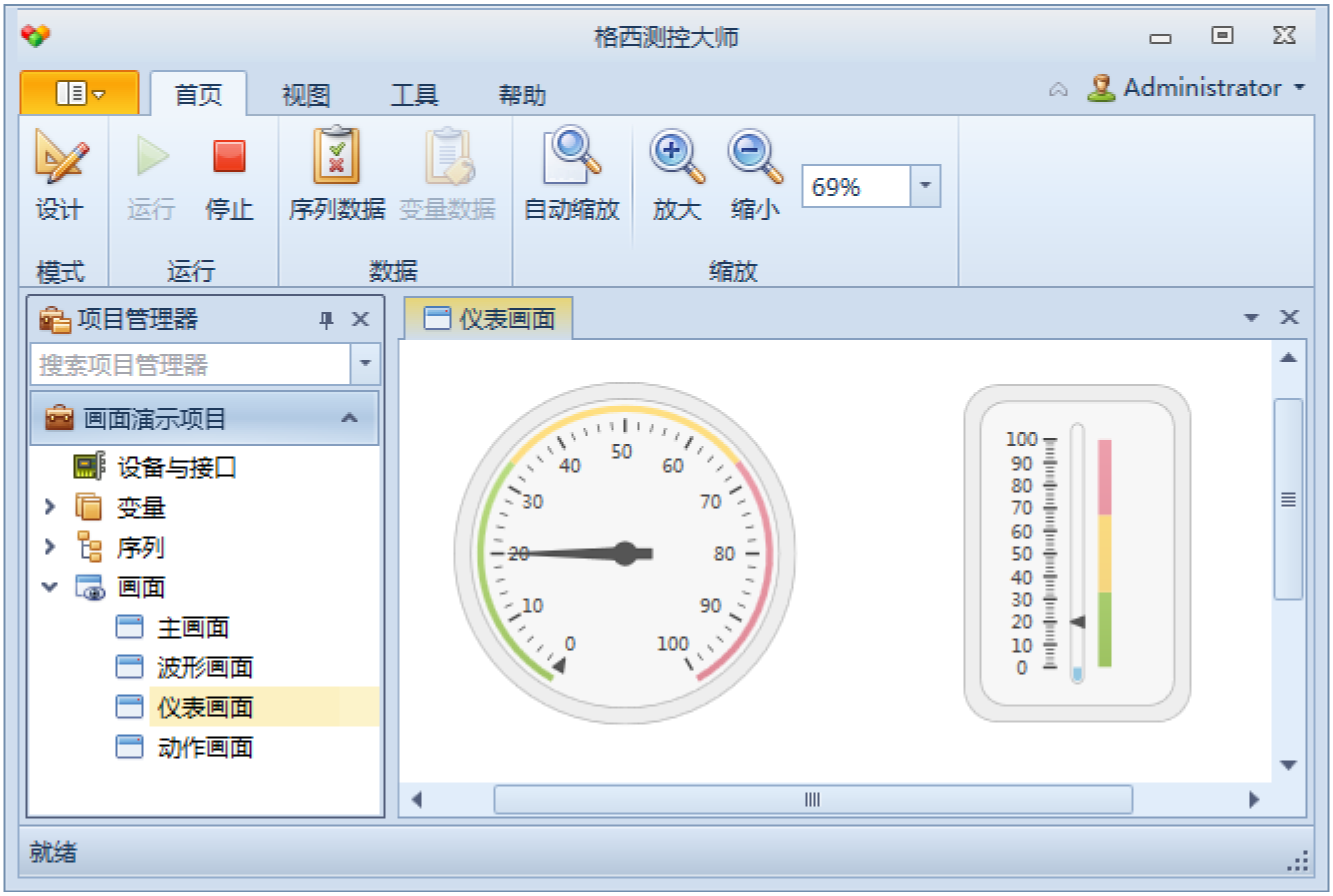
2.3.4 编辑区

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

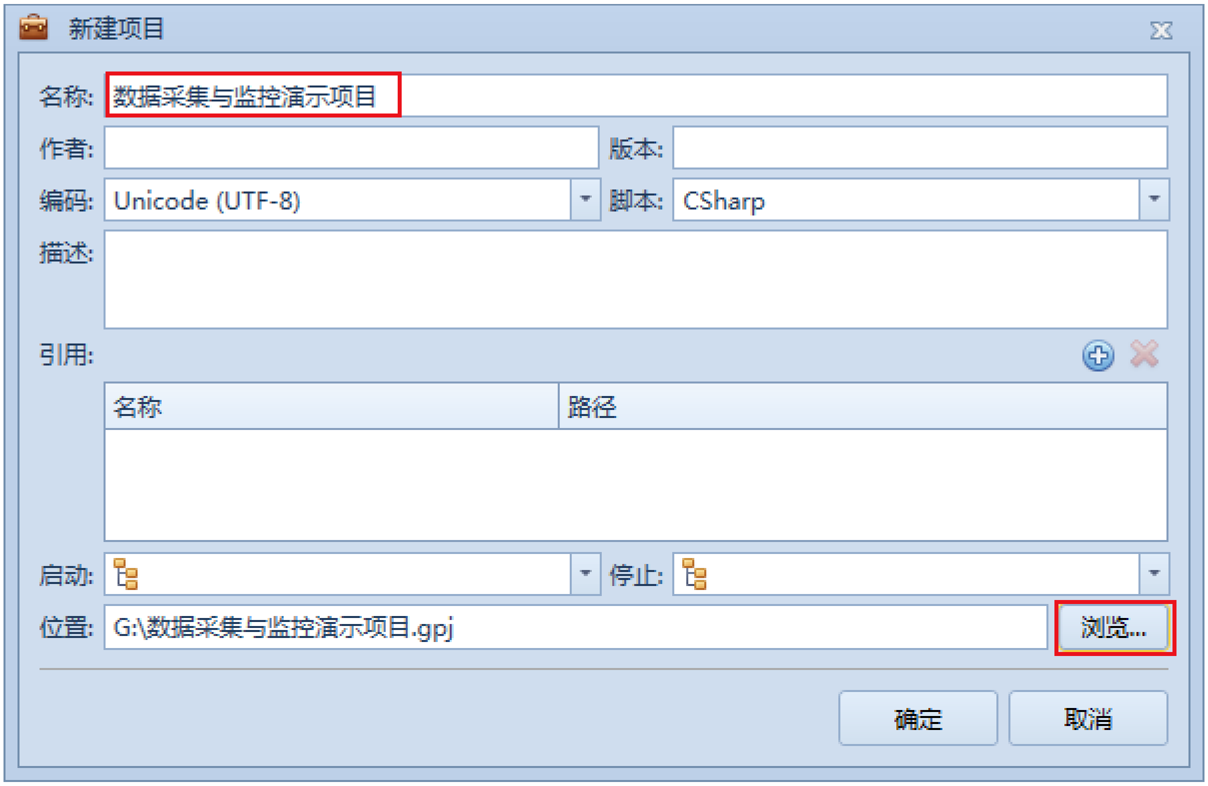
某热电偶采集模块，采用 Modbus RTU 通信协议，其中读温度命令为 03 命令，地址为 0000，温度值为 16 位有符号整数。

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\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。



新建项目

名称: 数据采集与监控演示项目

作者:

版本:

编码: Unicode (UTF-8)

脚本: CSharp

描述:

引用:

名称	路径

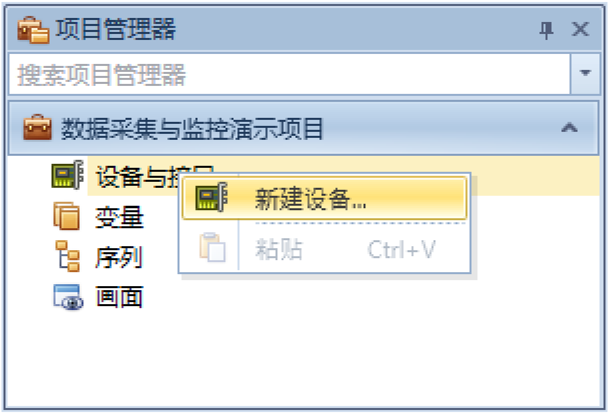
启动:

停止:

位置: G:\数据采集与监控演示项目.gpj

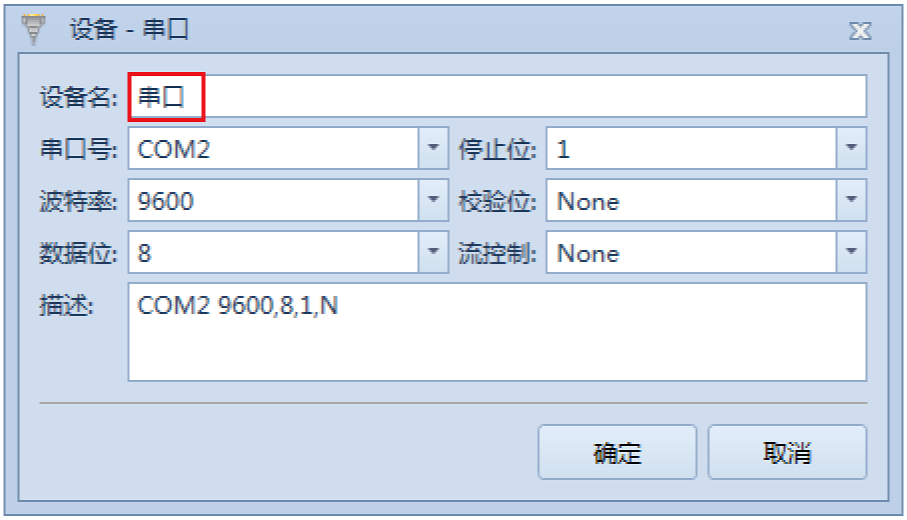
2.4.2 第 2 步 添加串口设备

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。


设备 - 串口

设备名: 串口

串口号: COM2

停止位: 1

波特率: 9600

校验位: None

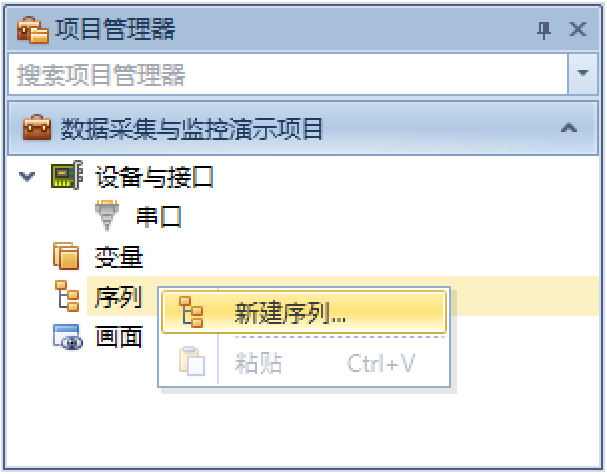
数据位: 8

流控制: None

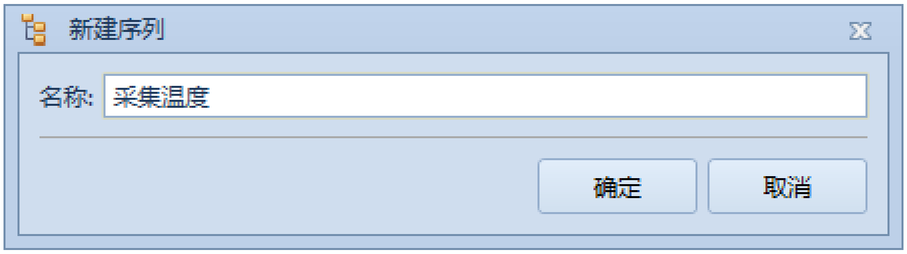
描述: COM2 9600,8,1,N

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...”。



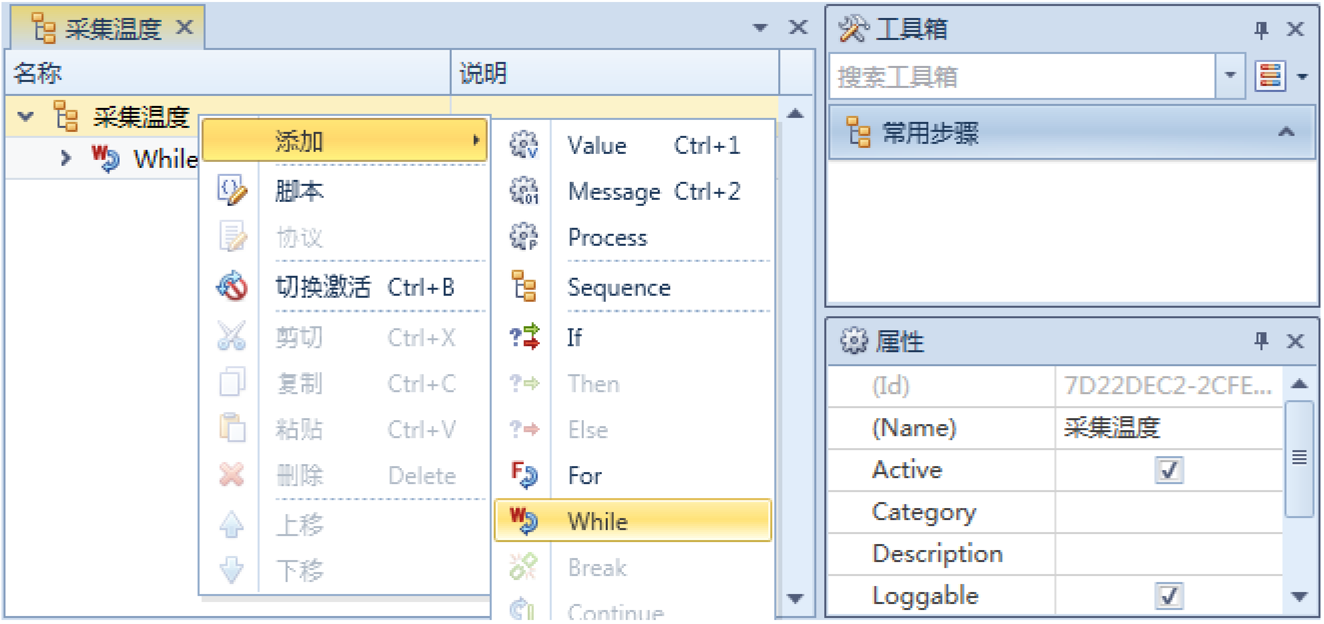
弹出新建序列对话框中，填写“名称”，点击“确定”。



名称: 采集温度

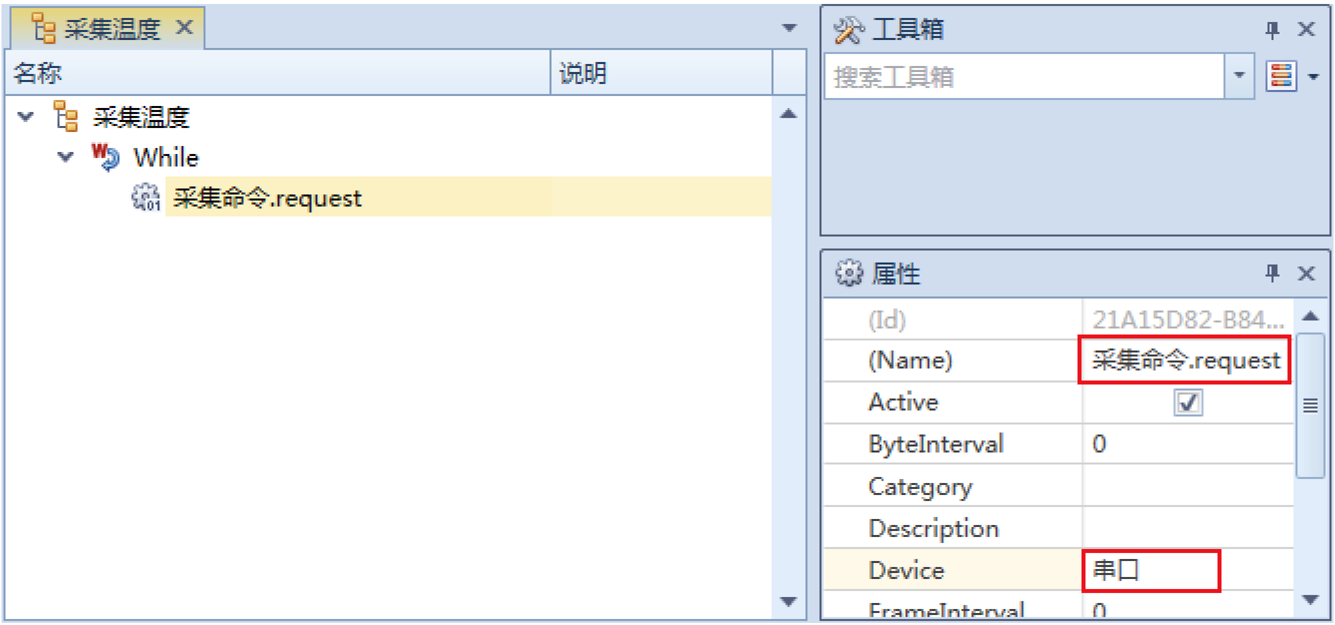
在项目管理器中选择新创建的“采集温度”节点，然后双击鼠标左键，或者点击鼠标右键，在弹出菜单中选择“编辑...”，打开序列编辑页面。

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。



(Id)	7D22DEC2-2CFE...
(Name)	采集温度
Active	☑
Category	
Description	
Loggable	☑

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 -> Message”，添加消息类型步骤，然后在属性面板，修改步骤名称“(Name)”为“采集命令.request”，OperationMode 设置为 Send，表示主动发送，Device 属性选择名称为“串口”的设备。



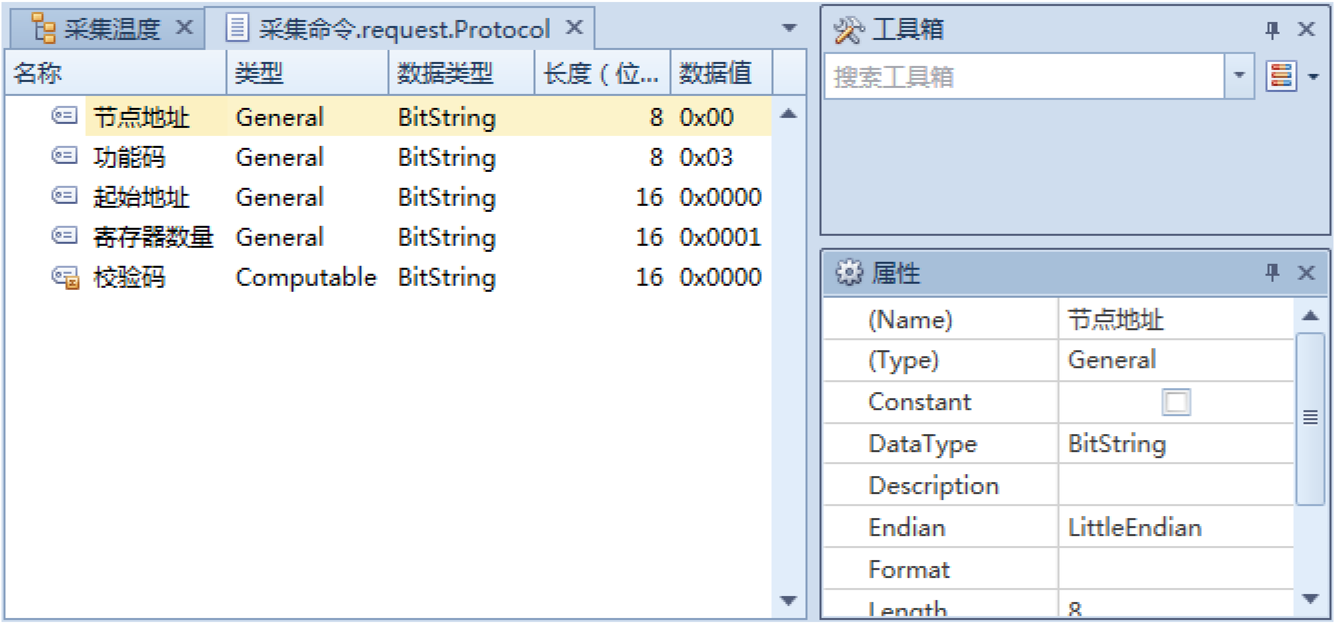
(Id)	21A15D82-B84...
(Name)	采集命令.request
Active	☑
ByteInterval	0
Category	
Description	
Device	串口
FrameInterval	0

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。



名称	类型	数据类型	长度（位...	数据值
节点地址	General	BitString	8	0x00
功能码	General	BitString	8	0x03
起始地址	General	BitString	16	0x0000
寄存器数量	General	BitString	16	0x0001
校验码	Computable	BitString	16	0x0000

(Name)	节点地址
(Type)	General
Constant	□
DataType	BitString
Description	
Endian	LittleEndian
Format	
Length	8

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 -> Message”，然后在属性面板，修改步骤名称“(Name)”为“采集命令.response”，OperationMode 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。

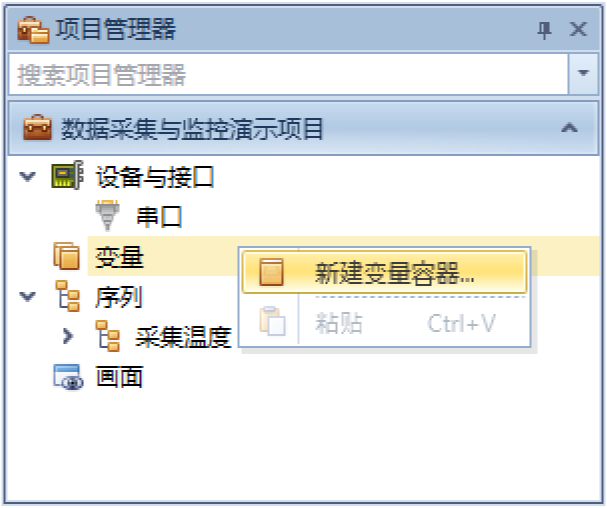
(Name)	(Type)	Constant	DataType	Endian	Length	Value
节点地址	General		BitString	LittleEndian	8	0x00
功能码	General	√	BitString	LittleEndian	8	0x03
字节数	General	√	BitString	LittleEndian	8	0x02
温度	General		Int16	BigEndian	16	0
校验码	Computable	√	BitString	LittleEndian	16	0x0000

其中，校验码的参数配置如下表。

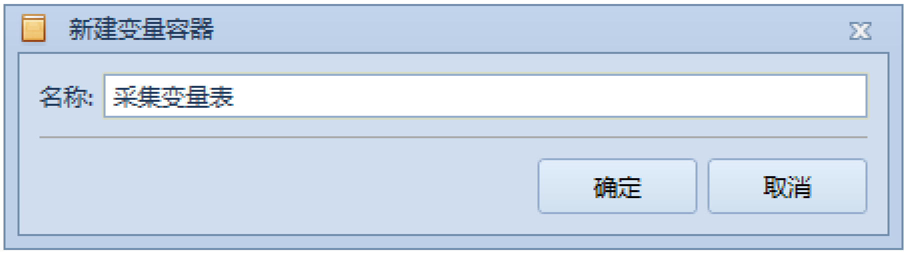
Algorithm	Priority	Location	StartPosition	EndPosition
CRC16MODBUS	1	Back	0	-1

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...”。



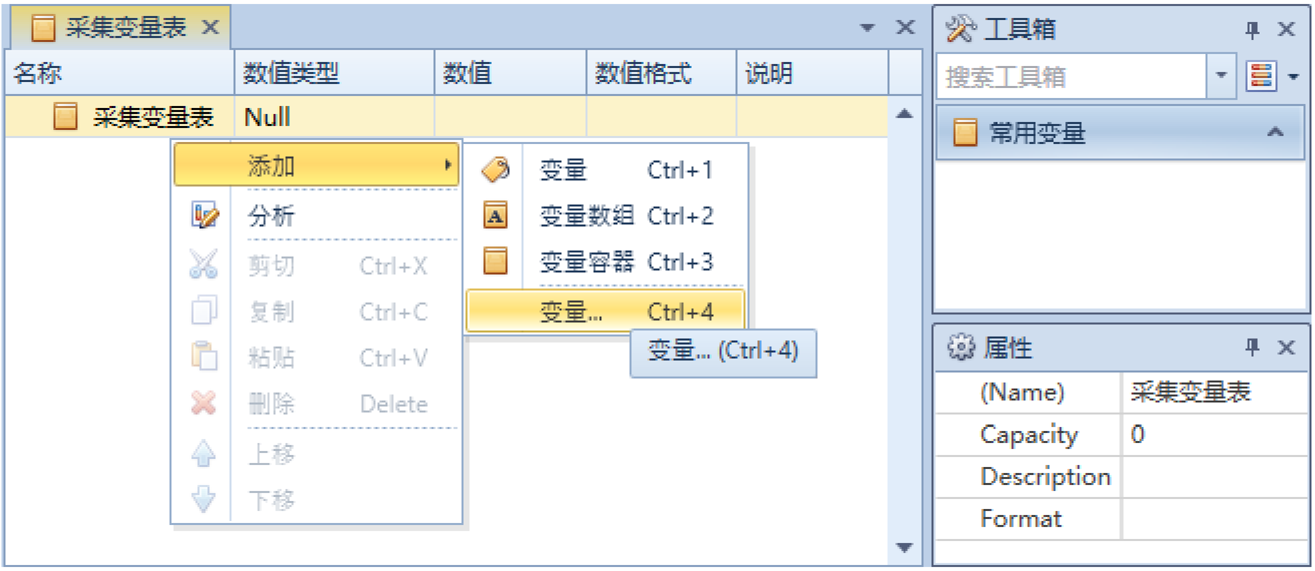
弹出新建变量容器对话框中，填写“名称”，点击“确定”。



名称: 采集变量表

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。

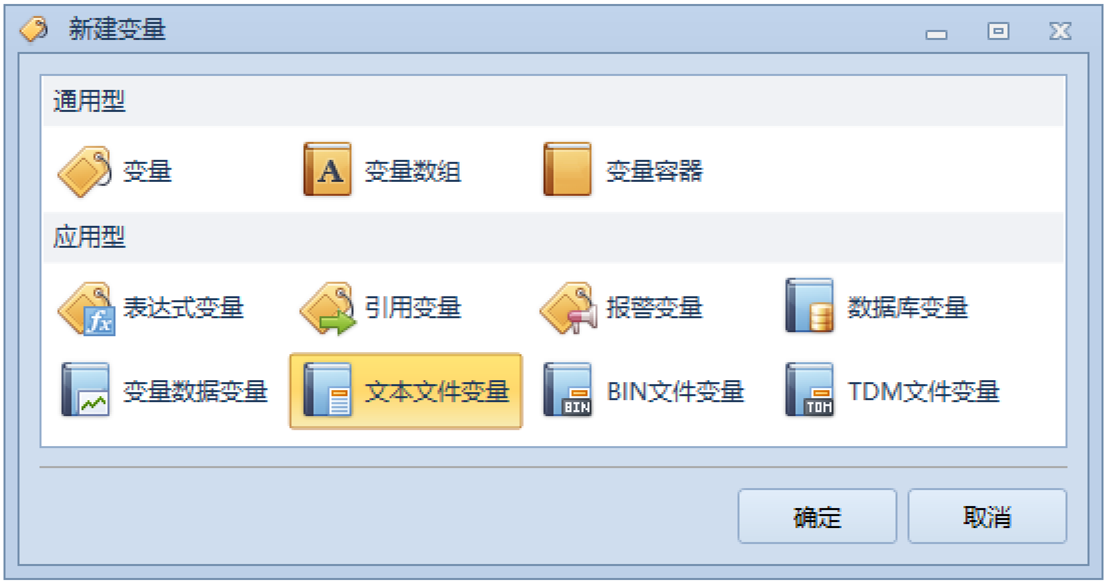
在“采集变量表”编辑页面，选中“采集变量表”节点，单击鼠标右键，在弹出菜单中选择“添加 ->变量...”。



名称	数值类型	数值	数值格式	说明
采集变量表	Null			

(Name)	采集变量表
Capacity	0
Description	
Format	

弹出新建变量对话框中，选择“文本文件变量”，点击“确定”。



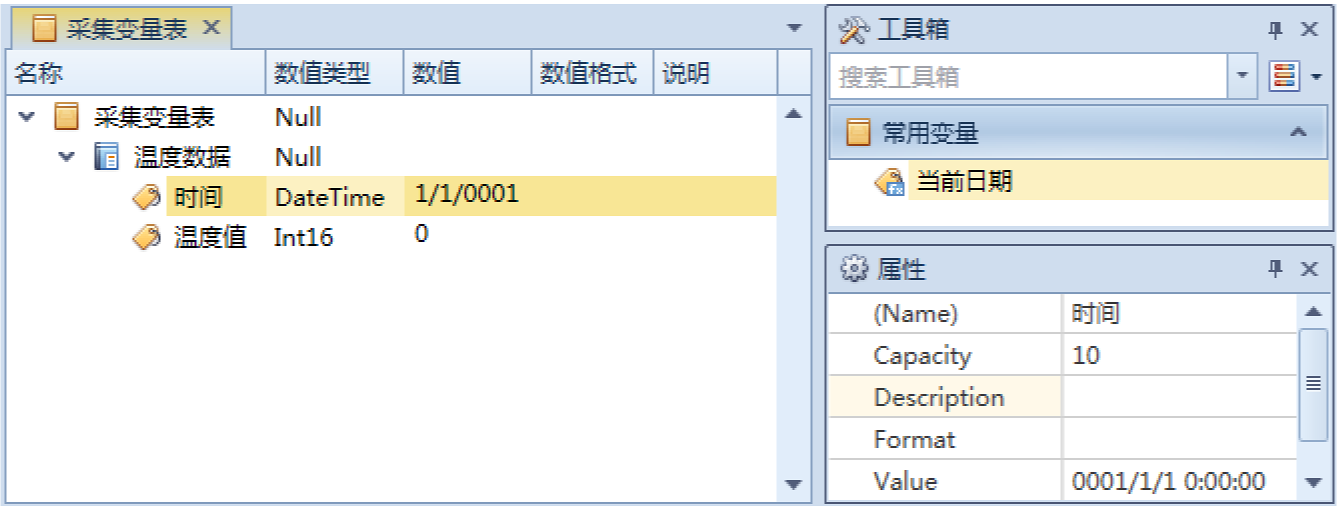
接下来，选中新创建的 TXT 文件变量，在属性面板，依次按以下属性表修改属性值。

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	



名称	数值类型	数值	数值格式	说明
采集变量表	Null			
温度数据	Null			
时间	DateTime	1/1/0001		
温度值	Int16	0		

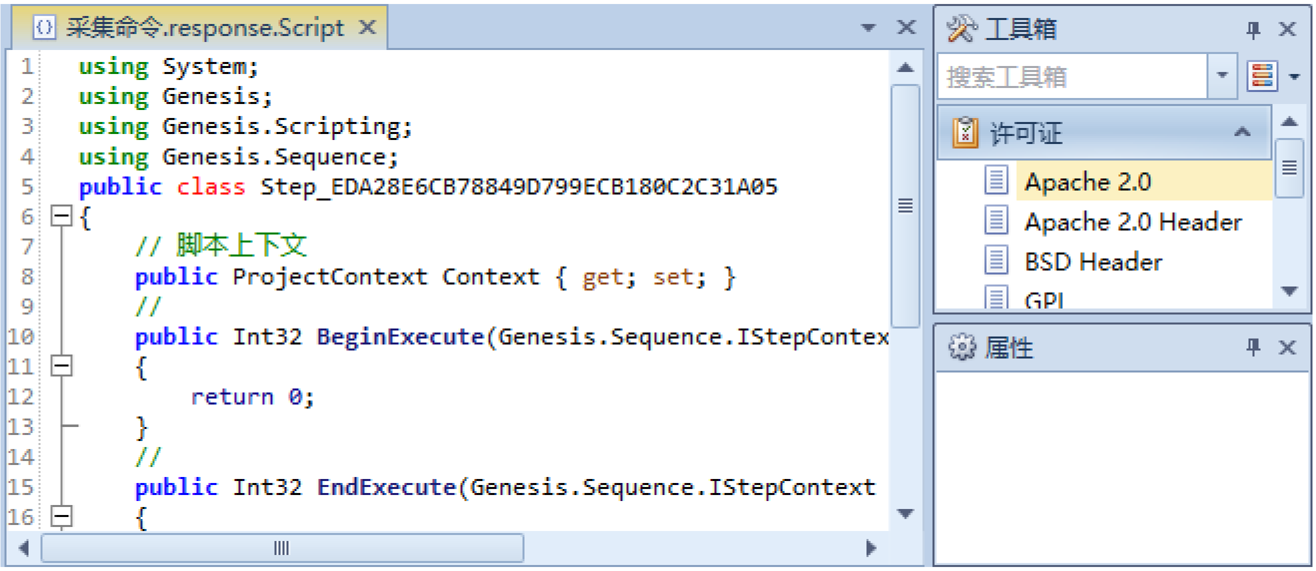
(Name)	时间
Capacity	10
Description	
Format	
Value	0001/1/1 0:00:00

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，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。



```
using System;
using Genesis;
using Genesis.Scripting;
using Genesis.Sequence;
public class Step_EDA28E6CB78849D799ECB180C2C31A05
{
    // 脚本上下文
    public ProjectContext Context { get; set; }
    //
    public Int32 BeginExecute(Genesis.Sequence.IStepContex
    {
        return 0;
    }
    //
    public Int32 EndExecute(Genesis.Sequence.IStepContext
    {
```

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。

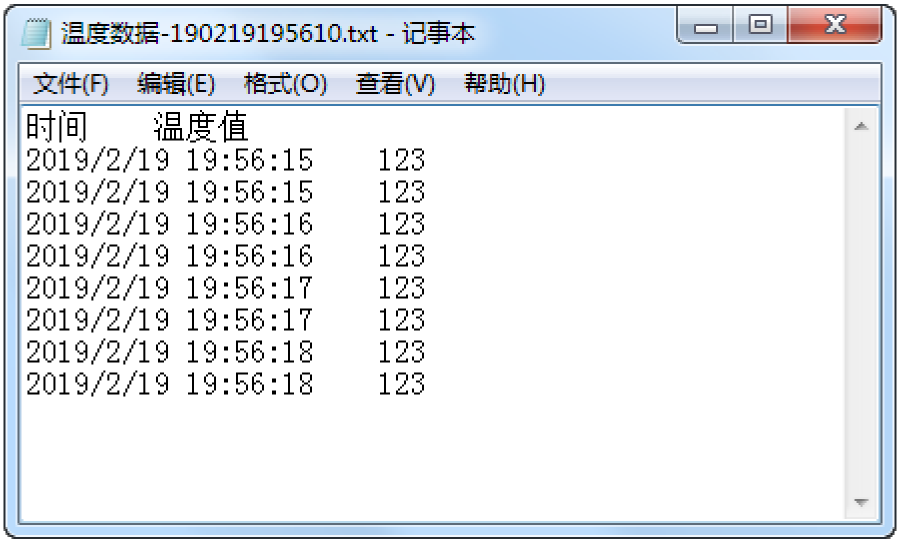
```
using System;
using Genesis;
using Genesis.Scripting;
```

```
/// 脚本类。
/// 注意：本软件的步骤脚本类和画面脚本类，用于在需要执行脚本函数时临时创建一个对象作为
/// 容器媒介来执行脚本函数，不是持久存在的一个对象，故不要使用私有变量保存状态。
/// 需要状态保存，可以使用项目上下文变量 Context 的数据操作函数；或者创建静态类保存。
public class Step_EDA28E6CB78849D799ECB180C2C31A05
{
  // 项目上下文
  public ProjectContext 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;
  }
```

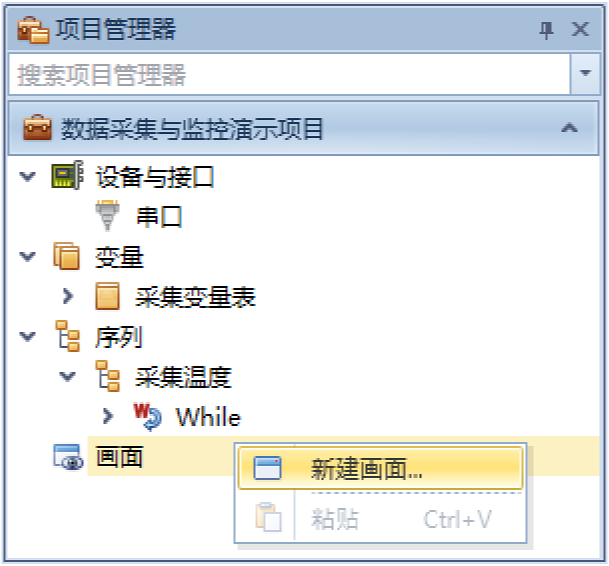
至此，已经完成温度数据的采集和保存功能，保存的数据格式如下图所示。



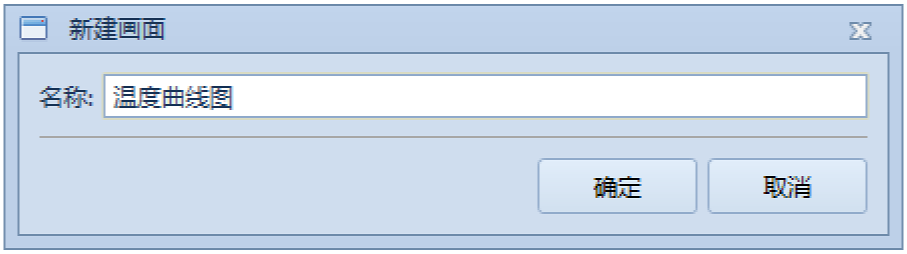
```
时间	温度值
2019/2/19 19:56:15	123
2019/2/19 19:56:15	123
2019/2/19 19:56:16	123
2019/2/19 19:56:16	123
2019/2/19 19:56:17	123
2019/2/19 19:56:17	123
2019/2/19 19:56:18	123
2019/2/19 19:56:18	123
```

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…”。



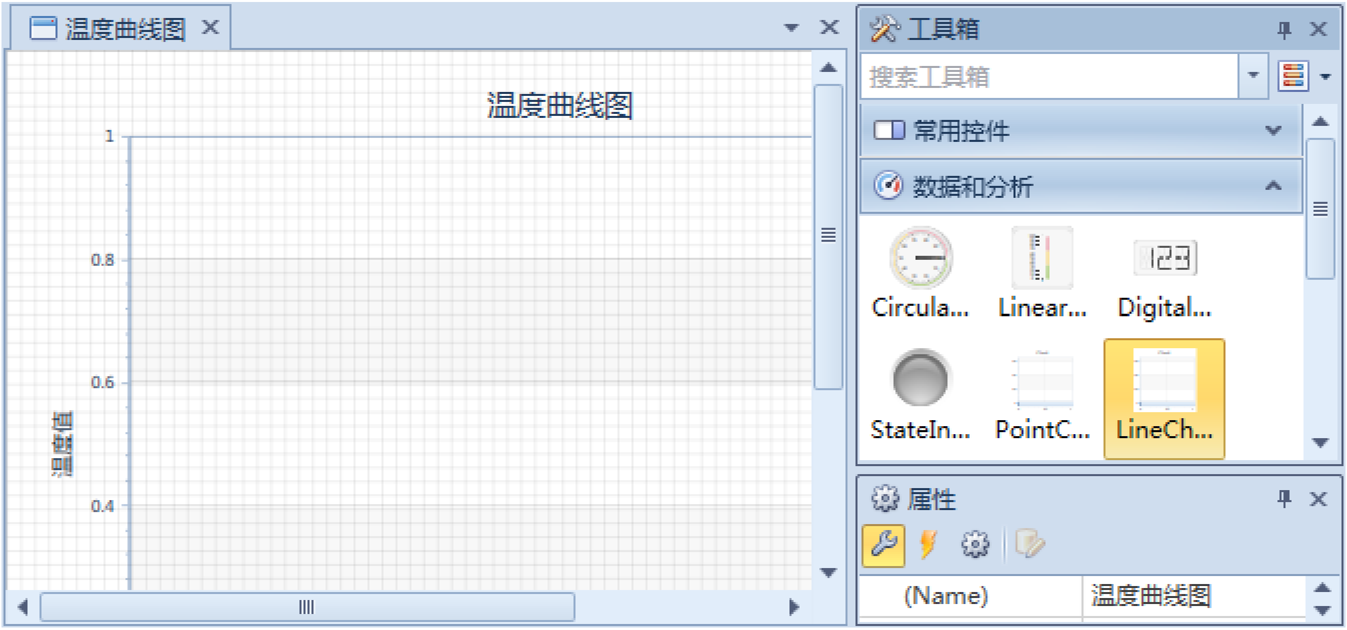
弹出新建画面对话框中，填写“名称”，点击“确定”。



名称: 温度曲线图

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。

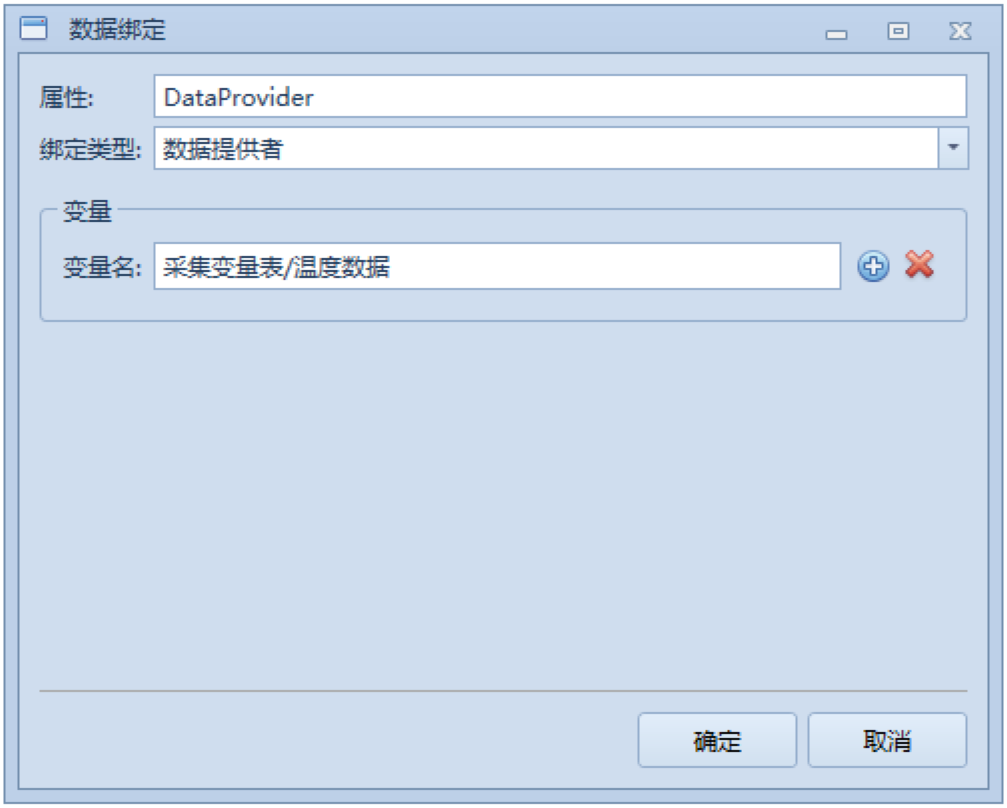


(Name) 温度曲线图

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”。



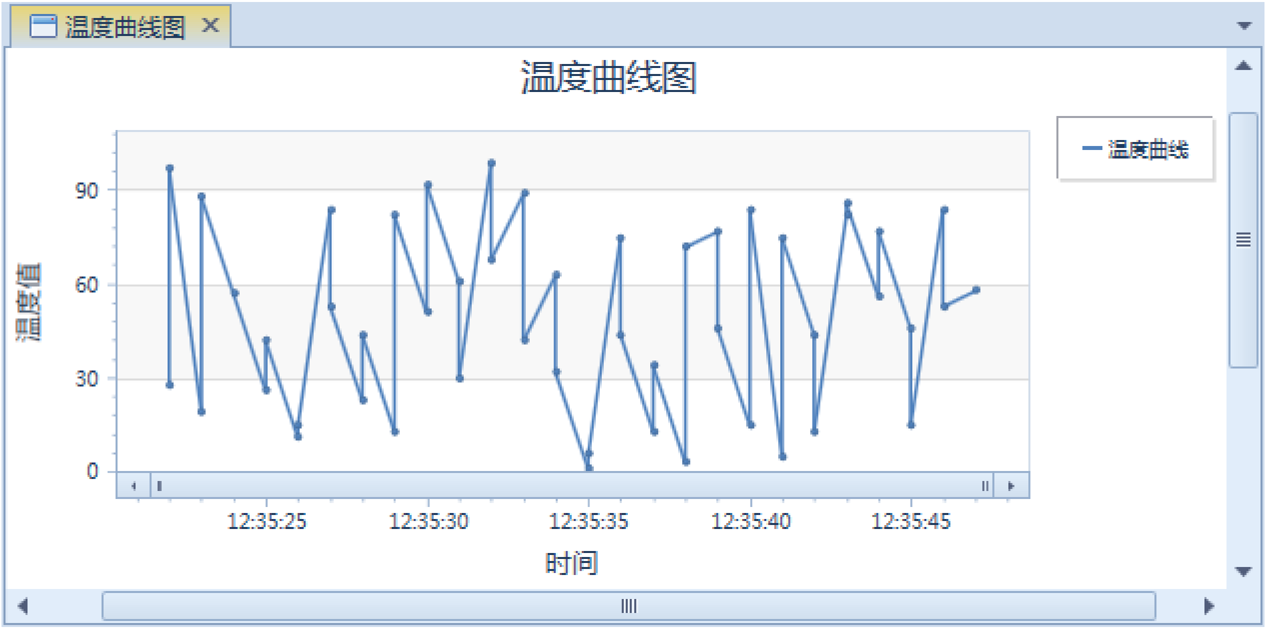
属性: DataProvider

绑定类型: 数据提供者

变量

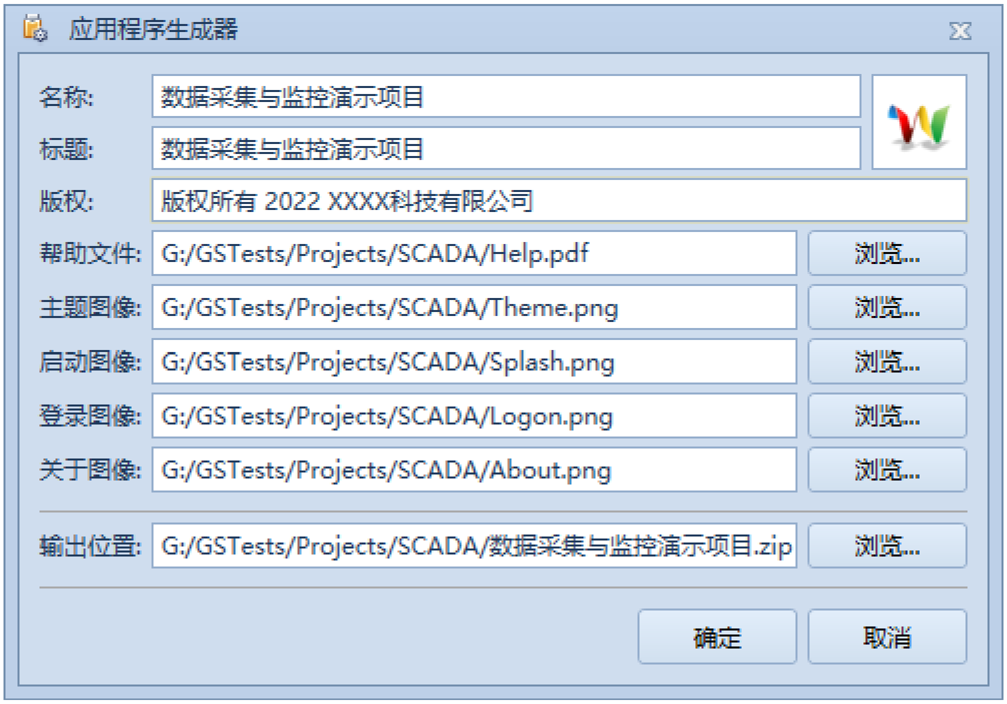
变量名: 采集变量表/温度数据

至此，温度曲线图界面功能完成。



2.4.7 第 7 步 生成应用程序

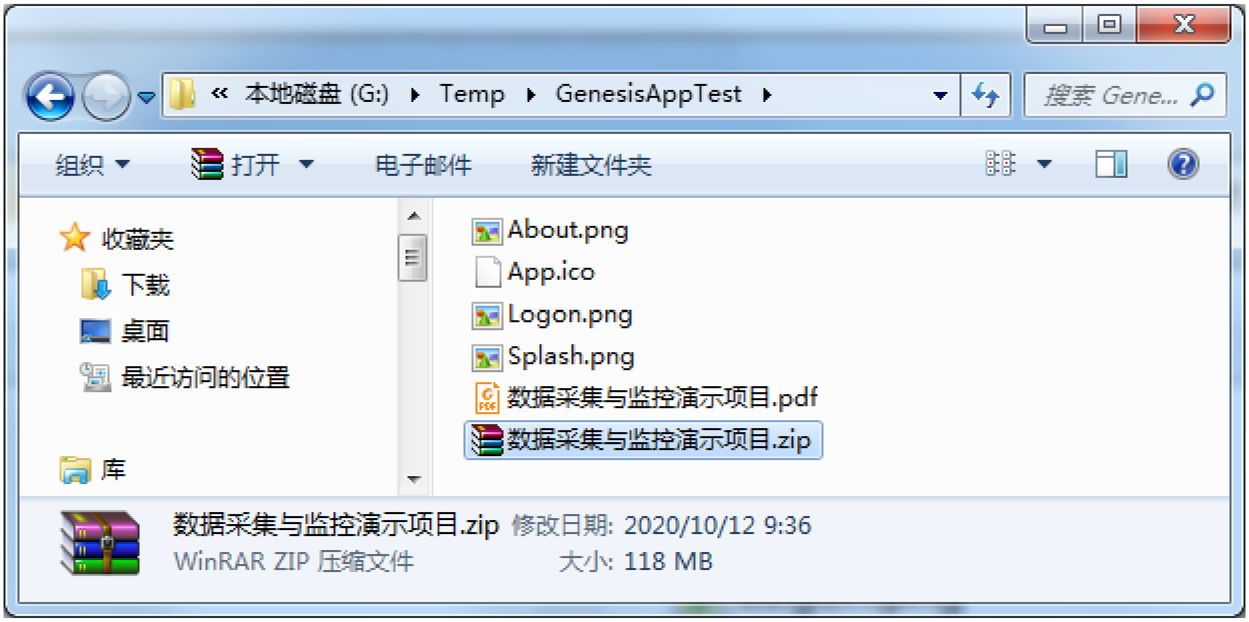
如果测控程序有部署的需求，可以在开发完成测控程序之后，通过左上角菜单中的“生成应用程序”功能创建运行时应用程序，用户可以自定义软件名称、软件的图标、标题、版权、帮助文件、主题图像、启动图像、登录图像、关于图像等软件信息。



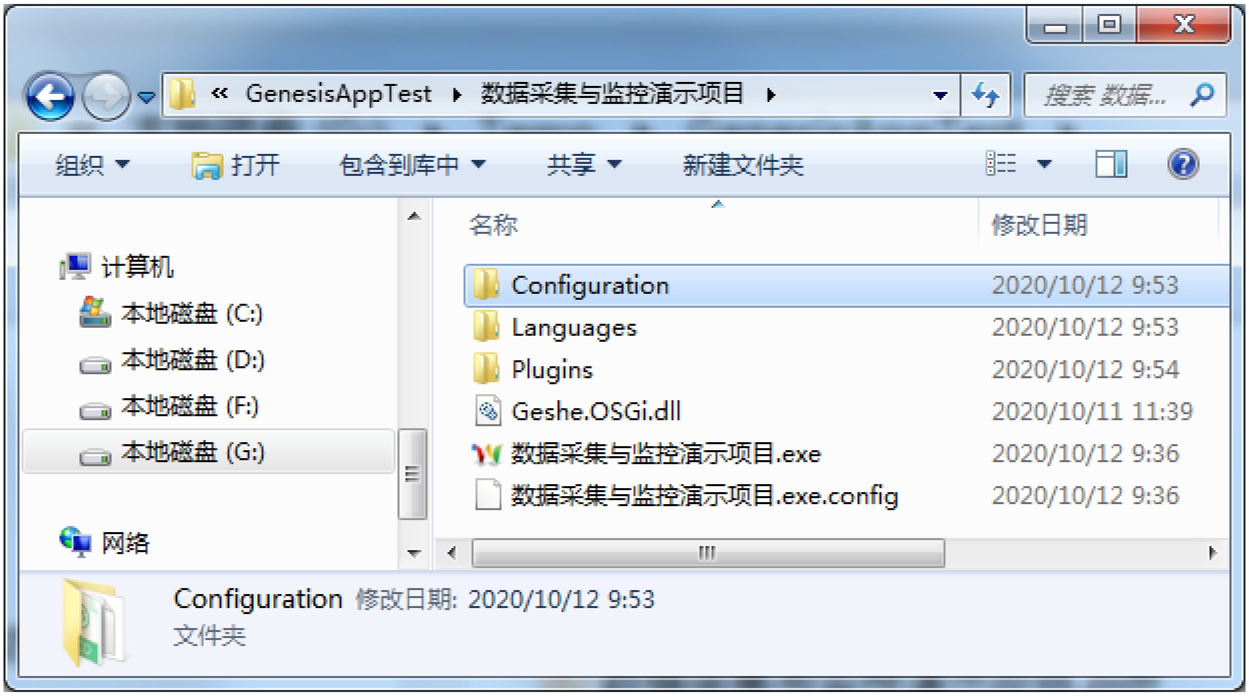
名称: 数据采集与监控演示项目
标题: 数据采集与监控演示项目
版权: 版权所有 2022 XXXX科技有限公司
帮助文件: G:/GSTests/Projects/SCADA/Help.pdf
主题图像: G:/GSTests/Projects/SCADA/Theme.png
启动图像: G:/GSTests/Projects/SCADA/Splash.png
登录图像: G:/GSTests/Projects/SCADA/Logon.png
关于图像: G:/GSTests/Projects/SCADA/About.png
输出位置: G:/GSTests/Projects/SCADA/数据采集与监控演示项目.zip

信息项	功能
名称	应用程序 exe 文件名称。
标题	应用程序标题，显示在主窗体标题栏。
应用程序图标	应用程序 exe 文件的图标。
版权	版权信息显示在启动画面和关于对话框中。
帮助文件	点击应用程序的帮助菜单，启动帮助文件，文件格式可以是任意被 Windows 认识的文件格式，如 PDF、Word 等。
主题图像	应用程序登录页面或者锁定用户页面的背景图像。
启动图像	应用程序启动时的图像。
登录图像	应用程序登录或锁定用户时对话框上方的图像。
关于图像	应用程序关于对话框上方的图像。
输出位置	生成的应用程序将被打包为一个.zip 文件，输出位置指定.zip 文件的输出路径和文件名。

打包完毕的应用程序“数据采集与监控演示项目.zip”，可以拷贝到任何安装了 Microsoft .Net Framework 4.8 的计算机中，解压后运行。

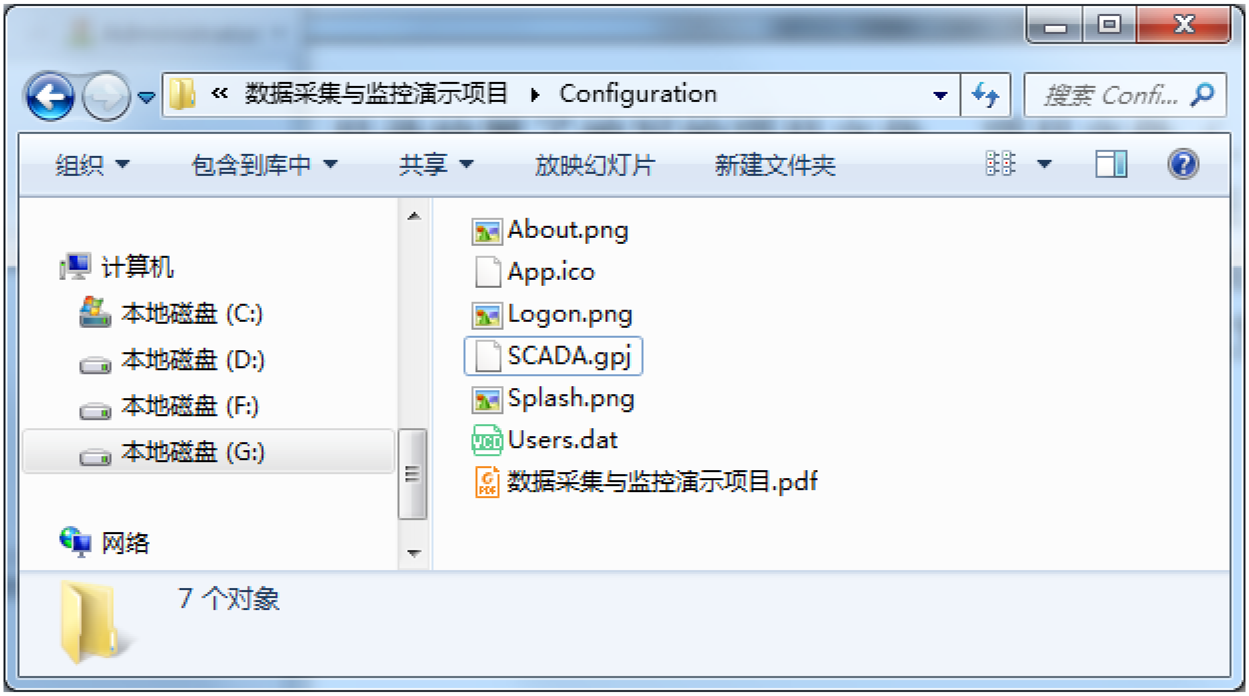


解压“数据采集与监控演示项目.zip”，目录结构如下：



名称	修改日期
Configuration	2020/10/12 9:53
Languages	2020/10/12 9:53
Plugins	2020/10/12 9:54
Geshe.OSGi.dll	2020/10/11 11:39
数据采集与监控演示项目.exe	2020/10/12 9:36
数据采集与监控演示项目.exe.config	2020/10/12 9:36

其中“Configuration”目录放置了编写的项目文件、项目文件（.gpj）引用的 DLL 文件、帮助文件以及相关的图像文件。需要注意的是，如果项目文件所引用的文件在生成的时候没有被格西测控大师检测到并自动包含到 Configuration 目录，用户需在生成.zip 文件后手动解压，自行拷贝缺失的文件到“Configuration”目录，手动打包后作为最终的应用程序部署包发布。



About.png
App.ico
Logon.png
SCADA.gpj
Splash.png
Users.dat
数据采集与监控演示项目.pdf

7 个对象

3. 开发指南

3.1 设备与接口

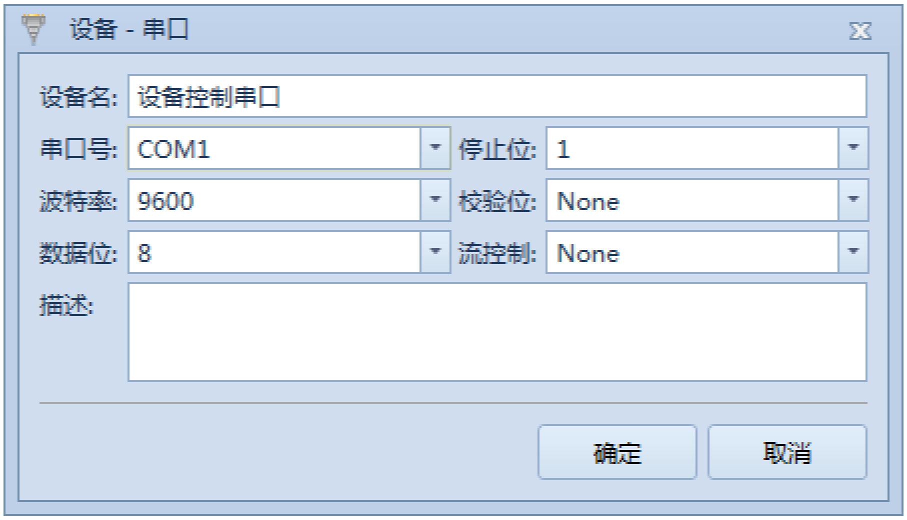
3.1.1 简介

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。设备和接口分为消息型和寄存器型，软件内置了串口、USB 接口和网口等设备和接口，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 通用设备类型

3.1.2.1串口

串口设备是消息型设备，用于连接串口型设备，支持 RS232、RS485，USB 转串口和虚拟串口等。

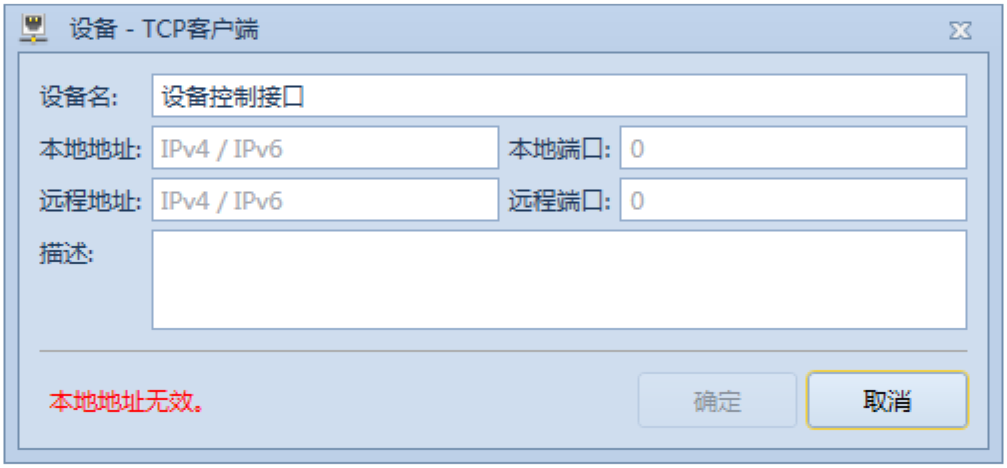


设备名: 设备控制串口
串口号: COM1
停止位: 1
波特率: 9600
校验位: None
数据位: 8
流控制: None
描述:

属性	描述
设备名	设备的唯一标识，任意字符串，同一个项目下的设备不允许同名，引用设备必须使用设备名。（定义适用于所有设备）
串口号	串口设备的端口，下拉列表列出系统支持的所有串口设备。
波特率	串口的通信波特率，支持手动输入波特率。
描述	设备描述字符串，任意字符串，描述信息可以作为 ToolTip 在项目管理器中显示。（定义适用于所有设备）

3.1.2.2TCP 客户端

TCP 客户端设备是消息型设备，用于连接 TCP 服务器型设备，收发基于 TCP 协议的数据包，可以控制支持网络接口的测试测量仪器和设备。



设备名: 设备控制接口
本地地址: IPv4 / IPv6
本地端口: 0
远程地址: IPv4 / IPv6
远程端口: 0
描述:

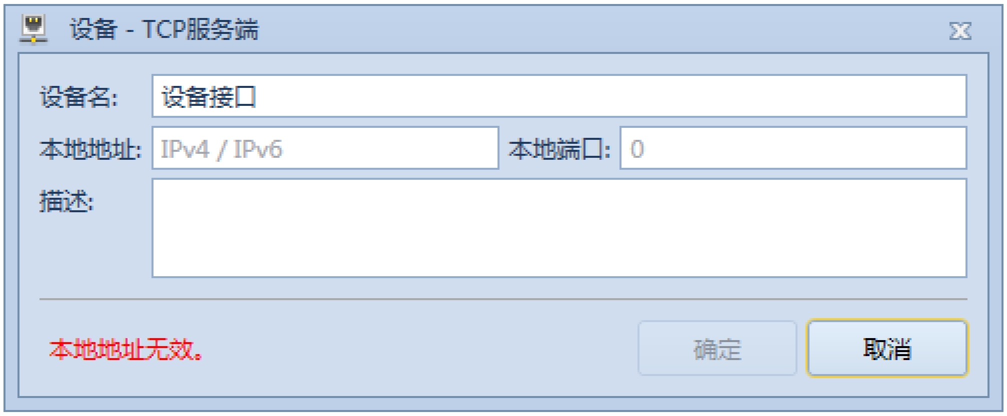
本地地址无效。

属性	描述
设备名	同上。
本地地址	本机用于和远程设备通信的 IP 地址，支持 Ipv4 和 Ipv6 类型地址。
本地端口	本机用于和远程设备通信的端口号，已经被占用的端口号不能使用。

远程地址	远程设备的 IP 地址，支持 Ipv4 和 Ipv6 类型地址。
远程端口	远程设备的端口号。
描述	同上。

3.1.2.3TCP 服务端

TCP 服务端设备是消息型设备，用于建立 TCP 服务器，供客户端连接，只支持一个客户端连接，多于一个客户端连接，后面的将踢掉前面的客户端连接，一般用于仿真基于 TCP 的设备。



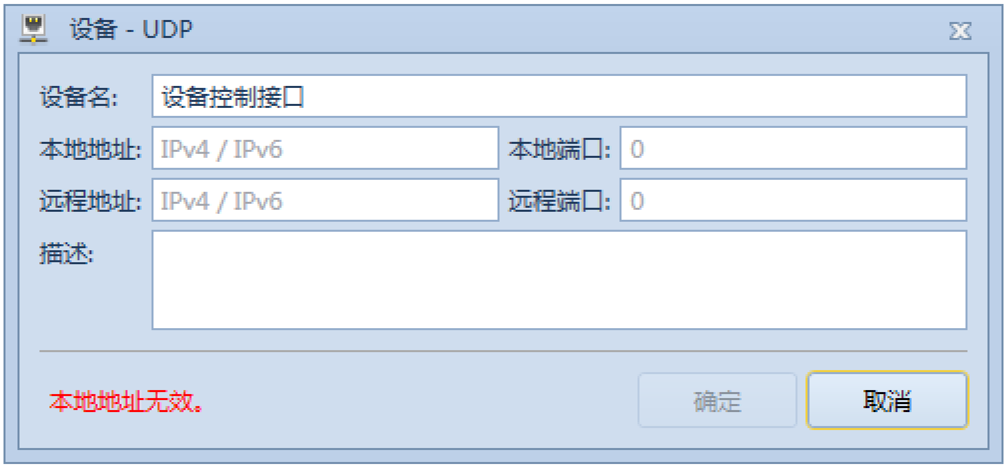
设备名: 设备接口
本地地址: IPv4 / IPv6
本地端口: 0
描述:

本地地址无效。

属性	描述
设备名	同上。
本地地址	本机用于建立服务器的 IP 地址，支持 Ipv4 和 Ipv6 类型地址。
本地端口	本机用于建立服务器的端口号，已经被占用的端口号不能使用。
描述	同上。

3.1.2.4UDP

UDP 设备是消息型设备，用于和 UDP 协议的设备通信，收发基于 UDP 协议的数据包。



设备名: 设备控制接口
本地地址: IPv4 / IPv6
本地端口: 0
远程地址: IPv4 / IPv6
远程端口: 0
描述:

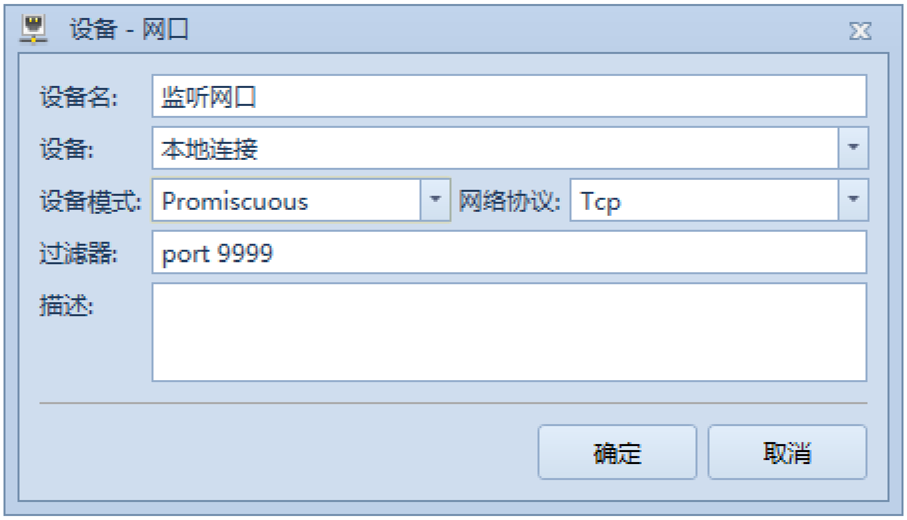
本地地址无效。

属性	描述
设备名	同上。

本地地址	本机用于和远程设备通信的 IP 地址，支持 Ipv4 和 Ipv6 类型地址。
本地端口	本机用于和远程设备通信的端口号，已经被占用的端口号不能使用。
远程地址	远程设备的 IP 地址，支持 Ipv4 和 Ipv6 类型地址。
远程端口	远程设备的端口号。
描述	同上。

3.1.2.5网口

网口设备是消息型设备，是基于网络设备的通信接口，能够操纵底层网络设备进行网络数据的抓取和分析，可以用于网络侦听、通信协议数据抓取和分析等。使用本设备需要安装 Npcap 软件驱动包。



设备名: 监听网口
设备: 本地连接
设备模式: Promiscuous
网络协议: Tcp
过滤器: port 9999
描述:

属性	描述
设备名	同上。
设备	选择用于通信的本机网络适配器，下拉框列出本机支持的所有适配器。
设备模式	支持 Normal（常规模式）和 Promiscuous（混杂模式）。Normal 模式网卡只接受来自网络端口的目的地址（MAC 地址）指向自己的数据；Promiscuous 模式网卡可以接收来自接口的所有数据。
网络协议	支持 Tcp 协议和 Udp 协议，指接收 Tcp 数据还是接收 Udp 数据。非基于 Tcp 或 Udp 协议的数据将被过滤。
过滤器	数据过滤表达式，表达式由一个或多个原语组成，原语通常由一个 id（名称或数字）加上一个或多个限定符组成。有两种不同的限定符： 类型限定符：表示 id 所指的类型。可能的类型有 host、net 和 port。例如，“host foo”，“net 128.3”，“port 20”。如果没有类型限定符，则默认为 host。 方向限定符：指定 id 之间的特定传输方向。可能的选择有 src、dst、src or dst 以及 src and dst。例如，“src foo”，“dst net 128.3”，“src or dst port 21”。 如果没有方向限定符，则默认为 src or dst。
描述	同上。

3.1.2.6USB 设备

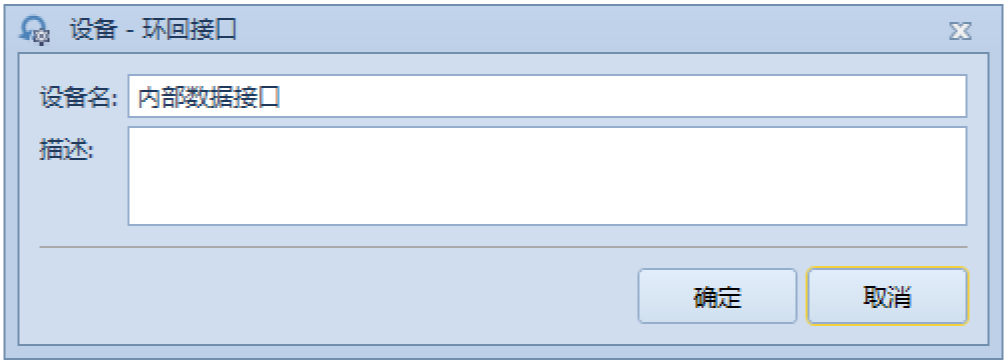
USB 设备是消息型设备，用于访问基于 USB 接口的设备。

设备名: USB设备
设备: SONiX USB DEVICE VID0C45,PID6741
读端点: Ep01
写端点: Ep01
描述: SONiX USB DEVICE VID0C45,PID6741

属性	描述
设备名	同上。
设备	选择用于通信的 USB 设备，下拉框列出本机支持的所有 USB 设备。
读端点	用于读的端点。
写端点	用于写的端点。
描述	同上。

3.1.2.7环回

环回设备是消息型设备，是一个模拟的设备，用于内部的数据流通，例如，可以搭配消息型步骤做协议数据解析器。



设备名: 内部数据接口

属性	描述
设备名	同上。
描述	同上。

3.1.3 应用设备类型

应用型设备多种多样，一般是基于链路层以上的应用层协议构建而来的设备类型，一般是寄存器

型设备，例如 Mqtt 服务器、Mqtt 客户端、Modbus 服务器、Modbus 客户端、Alarm 服务器、Alarm 客户端、Http 服务器、Http 客户端、OPCUA 服务器、OPCUA 客户端、IEC101 服务器、IEC101 客户端、IEC104 服务器、IEC104 客户端等设备类型。

3.1.4 扩展设备类型-NI 设备

VISA（Virtual Instrument Software Architecture），即虚拟仪器软件结构，是 VXI plug & play 联盟制定的 I/O 接口软件标准及其规范的总称。VISA 提供用于仪器编程的标准 I/O 函数库，是计算机与仪器的标准软件通信接口，计算机通过它来控制仪器。

软件内置的 NI-VISA 设备适配器，支持串口、TCP 设备、GPIB 设备、USB 设备、PXI 设备、DAQmx 设备，可以方便和支持这些接口的仪器和设备进行通信。VISA 型设备和接口需要美国 NI 公司的 NI-VISA 运行时库支持，DAQmx 设备需要 NI-DAQmx 库支持，用户可以到 NI 官网进行下载安装。

3.1.5 扩展设备类型-西门子设备

软件内置了西门子公司的 S7TCP 服务端和 S7TCP 客户端。

3.1.6 扩展设备类型-自定义设备

软件将设备和接口分为两种类型：消息型和寄存器型。对应的接口分别为消息型 IMessageDeviceSession 和寄存器型 IRegisterDeviceSession，两者都继承自 IDeviceSession，软件的设备适配器支持通过插件的方式，让用户扩展新的设备和接口，以满足特殊的连接需求。

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 类或者 RegisterDeviceSession 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=”某公司设备与接口”>
      <Description>某公司的设备</Description>
    </DeviceCategory>
    <Device Id=”CompanySerialX” Category=”Company” Name=”X型串口” Class=”
Genesis.Device.Company.SerialSessionX” EditorClass=”
Genesis.Device.Company.SerialSessionXDialog”>
      <Image>Base64编码的小图标</Image>
      <LargeImage> Base64编码的大图标</LargeImage>
      <Description>X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。

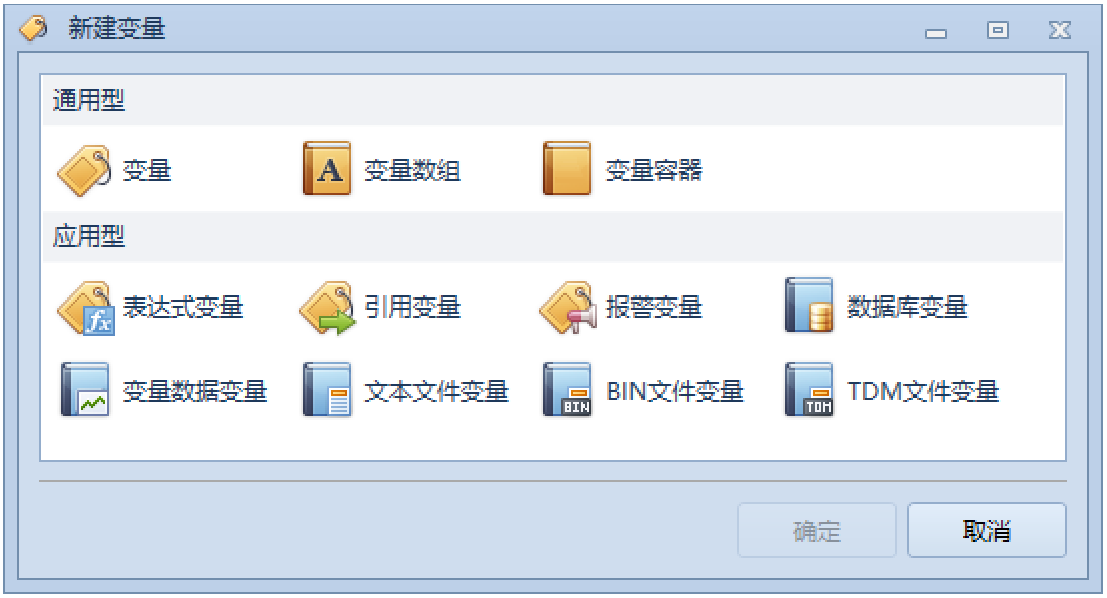
关于扩展设备和接口的用法例子，请参考：<软件安装目录>\Examples\Basics\Devices。

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，还可以创建扩展的应用型变量，如表达式变量、引用变量、报警变量、数据库变量、变量数据变量、文本文件变量、BIN 文件变量等应用型变量，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（基类）

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。
Unit	变量单位
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（派生自变量）

数组变量中的元素都是同一个类型的变量。

属性	描述
(Name)	同基类
Capacity	变量缓存容量，指能够缓存多少个数据记录。数据记录产生的规则是数组

	中的所有子变量的值都改变后，生成一行记录。
Description	同基类
Format	数组变量的格式，字符串格式支持 Json 和 Xml，设置格式影响 Value 属性和 ToString 函数。
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（派生自变量）

容器变量可以容纳任何类型的变量。

属性	描述
(Name)	同基类
Capacity	变量缓存容量，指能够缓存多少个数据记录。数据记录产生的规则是容器中的所有子变量的值都改变后，生成一行记录。
Description	同基类
Format	容器变量的格式，字符串格式支持 Json 和 Xml，设置格式影响 Value 属性和 ToString 函数。

应用型变量 – 通道变量（派生自变量）

通道型变量本质是设备通道的代理，软件提供了通道型变量的抽象基类 ChannelVariant，用户可以通过基类派生出各种设备的数据访问变量。

属性	描述
Active	是否激活，激活的通道变量才能执行周期更新。
ChannelType	通道类型，支持 Input 和 Output 两种，分别表示输入通道和输出通道。
Category	通道类别，类别相同的通道在相同的线程中执行。
Frequency	通道采样频率，单位 Hz。
Timeout	数据采样失败的超时时间，单位是毫秒。
TimingType	通道采样定时类型，支持 SampleClock 和 Demand 两种。SampleClock 类型指通道管理器依据 Frequency 属性的值自动定时采样；Demand 类型指按需采样，调用 Variant 的 Read 或 Write 函数对变量的读或写操作产生采样，采样数量由读或写的数量决定，通常配合变量传输步骤（Transmission 步骤）实现按需采样，产生采样数据。

应用型变量 – 表达式变量（派生自通道型变量）

属性	描述
Expression	表达式，运行时动态计算和产生变量的值，可供使用的表达式变量有 N、T 和 Value，N（整数类型）表示自项目运行或上次变量复位以来经过的采样次数，T（浮点类型）根据 Frequency 和采样次数 N 计算得到的秒数，只有 Frequency 大于 0 才有值，Value 表示当前变量值，Variants 表示当前项目的变量集，变量引用可以使用相对路径符..和.。 例如：50*Sin(T)、(Value-100)/100、50*Sin(3.1416*N/180)等 详情请参考 3.5 节表达式。
TimingType	SampleClock 类型时依据 Frequency 属性的值自动定时计算表达式的值更新本变量值；Demand 类型时调用 Variant 的 Read 函数或者使用变量传输步骤（Transmission 步骤）更新本变量值。

应用型变量 – 引用变量（派生自通道型变量）

引用变量可以跟随所引用的变量的值，可以配合 Frequency 属性用作重采样等场景。

属性	描述
Expression	表达式，用于对引用的变量值进行计算，并把计算结果作为最终变量的值，可供使用的表达式变量 Value 表示当前变量值，Variants 表示当前项目的变量集，变量引用可以使用相对路径符..和.。 表达式可以为空，为空时直接使用引用的变量值作为最终变量的值。 请参考 3.5 节表达式。
Reference	引用的变量全名或相对路径名，格式如： 变量集 1/变量 1、../变量 1。
TimingType	SampleClock 类型时依据 Frequency 属性的值自动定时采样引用的变量的值更新本变量值；Demand 类型时引用的变量值变化更新本变量值。

应用型变量 – 报警变量（派生自通道型变量）

报警变量可以提供报警功能，提供报警触发和报警恢复事件，只能通过脚本的方式注册和响应事件；可以跟随所引用的变量的值，可以配合 Frequency 属性用作重采样等场景。

属性	描述
Reference	引用的变量全名或相对路径名，格式如： 变量集 1/变量 1、../变量 1。
TimingType	SampleClock 类型时依据 Frequency 属性的值自动定时采样引用的变量的值更新本变量值；Demand 类型时引用的变量值变化更新本变量值。
AlarmConfiguration	报警配置 限值报警：当前变量值超过定义的限制值，发生报警。 偏差报警：当前变量减去目标值的差值，差值和定义的限制值比较，超过阀值则发生报警。 速率报警：(当前变量值-上一时间点变量值)/(当前时间-上一时间)，比率与定义的限制值比较，超过阀值则发生报警。 状态报警：当前变量值等于状态设置值，则发生报警。 死区：死区是为了防止变量值在报警限上下频繁波动时，产生不真实的报警，在原报警限上下增加一个报警限的阀值，使原报警界限由一条线变为一条报警界限带，当变量的值在报警界限带范围内变化时，不会发生和恢复报警，而只有超出该报警界限带范围时，才发生报警信息，对消除波动信号的无效报警有积极作用。
AlarmDelay	报警延时，单位 ms。当变量值达到报警条件时，延时一段时间再触发报警，如果在延时时间内变量值恢复到正常值，则不触发报警。 通常设置一个大于 0 的值，可以消除一些瞬时剧烈波动或者传感器有时返回错误的数据，但通常很快恢复正常的设备。
AlarmRelations	报警关联表，可以是变量全名或者一个字符串。 当报警发生时可以跟踪额外的变量值，或者跟踪固定的字符串，帮助诊断问题。

AlarmTriggered	报警触发事件
	利用脚本注册事件，在事件函数中获取报警信息。
	涉及到的类：AlarmChannelVariant、Alarm、AlarmEventArgs
	实现方面一般通过一个全局脚本类来承载事件函数，在画面或者步骤的脚本中注册事件。例如：
	`public static class Global`
	`{`

```
  private static string AlarmFormat = "触发时间:{0:HH:mm:ss} 恢复
时间:{1:HH:mm:ss} 消息:{2} 类型:{3} 优先级:{4} 名称:{5} 变
量:{6}";

  public static void Alarm_AlarmTriggered(object sender,
AlarmEventArgs e)
  {
    SystemContext.LogMessage("触发: "+string.Format(AlarmFormat,
e.Alarm.TriggerTime,e.Alarm.RecoverTime, e.Alarm.Text,
e.Alarm.AlarmType.ToString(), e.Alarm.Priority, e.Alarm.Name,
e.Alarm.Variant));
  }
  public static void Alarm_AlarmRecovered(object sender,
AlarmEventArgs e)
  {
  }
}
public class Schema_CE25DA4A835B4B0D9E6F50E5685F7270
{
  public ProjectContext Context { get; set; }
  //
  public void BtnRegAlarm_Click(Object sender,
System.Windows.RoutedEventArgs e)
  {
    AlarmChannelVariant v = Context.Variants["Vars/ValueAlarm"]
as AlarmChannelVariant;
    v.AlarmTriggered += Global.Alarm_AlarmTriggered;
    v.AlarmRecovered += Global.Alarm_AlarmRecovered;
  }
}
```

AlarmRecovered	报警恢复事件 使用方式同上。

应用型变量 – 通道容器变量（派生自容器变量）

通道容器变量本质是设备通道组的代理，软件提供了通道容器变量的抽象基类 ChannelVariantContainer，用户可以通过基类派生出各种设备的数据组访问变量。

属性	描述
Active	是否激活，激活的通道变量才能执行周期更新。
ChannelType	通道类型，支持 Input 和 Output 两种，分别表示输入通道和输出通道。
Frequency	通道频率，单位 Hz。
TimingType	通道采样定时类型，支持 SampleClock 和 Demand 两种。SampleClock 类型指通道管理器依据 Frequency 属性的值自动定时采样；Demand 类型指按需采样，调用 Variant 的 Read 或 Write 函数对变量的读或写操作产生采样，采样数量由读或写的数量决定，通常配合变量传输步骤（Transmission 步骤）实现按需采样，产生采样数据。
Timeout	数据采样失败的超时时间，单位是毫秒。

应用型变量 – 数据库变量（派生自通道容器变量）

数据库变量是一个容器类变量，可以容纳任意类型子变量，可以通过数据库变量进行数据库读写。

属性	描述
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 SqlServer 数据库连接串：Data Source=.\SQLEXPRESS;Initial Catalog=VariantsSQLServer;User ID=sa;Password=12345678;Connection Timeout=2 注意：当 ChannelType 为 Output 时，SQLite 数据库类型在运行启动时自动检查和创建数据库，但是 MySQL 和 SqlServer 数据库类型必须手工创建数据库。
ChannelType	通道类型。Input 表示从数据库读数据，Output 表示对数据库写数据。 提示：如果对同一个数据库即有读操作又有写操作，可以通过分别建立 ChannelType 为 Input 和 Output 的数据库变量来实现。
DatabaseType	数据库类型，软件支持 MySQL，SqlServer 和 SQLite。
DataTable	数据表名称。ChannelType 为 Output 时，数据表按需自动生成，表字段名称采用容器中变量名；ChannelType 为 Input 时，指需要读取数据的表名，读回来的数据赋值给子变量名称和表列名一致的子变量。
DataTableIndices	数据表索引。索引由列名称组成，用分号分隔不同索引，用逗号分隔组合索引，例如：列 1;列 2,列 3。
StorageDuration	存储时长，0 表示不分数据库存储，大于 0 表示按时长分隔数据库。注：只支持 SQLite 数据库。
StorageDurationUnit	存储时长单位，支持 Second、Minute、Hour、Day，其中 Hour 和 Day 在生成数据库时除第一个数据库外，其他数据库对齐整点单位，例如单位是 Day 则按天对齐，在 0 点整切分数据库。
StorageQuantity	存储数量，0 表示无限个，大于 0 表示数量超过设定值后，循环删除最早的数据库。该属性在 StorageDuration 大于 0 时有效。
TimingType	SampleClock 类型：ChannelType 为 Input 时自动定时每次读取 1 条记录，ChannelType 为 Output 时自动定时将最新的数据记录写入数据库； Demand 类型：ChannelType 为 Input 时调用 Variant 的 Read 函数或者使用变量传输步骤（Transmission 步骤）读取数据库数据。ChannelType 为 Output 时调用 Variant 的 Write 函数或者使用变量传输步骤写入数据库。

应用型变量 – 变量数据变量（派生自通道容器变量）

变量数据变量是一个容器类变量，可以容纳任意类型子变量，可以对变量数据 VData 文件进行读写，VData 文件是一个 SQLite 数据库文件，可以用数据管理器直接打开浏览和进行数据分析。

属性	描述
DataTableIndices	变量数据表的索引。索引由列名称组成，用分号分隔不同索引，用逗号分隔组合索引，例如：列 1;列 2,列 3。
Directory	变量数据文件保存的目录，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	变量数据文件名，后缀名强制为“.vdata”。
StorageDuration	存储时长，0 表示不分文件存储，大于 0 表示按时长分隔文件。
StorageDurationUnit	存储时长单位，支持 Second、Minute、Hour、Day，其中 Hour 和 Day 在生成文件时除第一个文件外，其他文件对齐整点单位，例如单位是 Day 则按天对齐，在 0 点整切分文件。
StorageQuantity	存储数量，0 表示无限个，大于 0 表示数量超过设定值后，循环删除最早的文件。该属性在 StorageDuration 大于 0 时有效。
TimingType	SampleClock 类型：ChannelType 为 Input 时自动定时每次读取 1 条记录，ChannelType 为 Output 时自动定时将最新的数据记录写入数据文件； Demand 类型：ChannelType 为 Input 时调用 Variant 的 Read 函数或者使用变量传输步骤（Transmission 步骤）读取数据库数据。ChannelType 为 Output 时调用 Variant 的 Write 函数或者使用变量传输步骤写入数据文件。

应用型变量 – 文本文件变量（派生自通道容器变量）

文本文件变量是一个容器类变量，可以容纳任意类型子变量，可以对文本型文件进行读写。读写以行为单位，以 Delimiter 属性指定的列分隔符分隔数据。

属性	描述
Delimiter	数据分隔符。
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。
FileHeaderEnabled	文件列头使能，表示文件列头是否有效。当为 True 时，表示第一行为列表头，表头的列名称和子变量名称对应；当为 False 时，表示无列表头，第一行即为数据行，数据列按顺序和子变量对应。
StorageDuration	存储时长，0 表示不分文件存储，大于 0 表示按时长分隔文件。
StorageDurationUnit	存储时长单位，支持 Second、Minute、Hour、Day，其中 Hour 和 Day 在生成文件时除第一个文件外，其他文件对齐整点单位，例如单位是 Day 则按天对齐，在 0 点整切分文件。
StorageQuantity	存储数量，0 表示无限个，大于 0 表示数量超过设定值后，循环删除最早的文件。该属性在 StorageDuration 大于 0 时有效。
TimingType	SampleClock 类型：ChannelType 为 Input 时自动定时每次读取 1 行记录，ChannelType 为 Output 时自动定时将最新的数据记录写入数据库； Demand 类型：ChannelType 为 Input 时调用 Variant 的 Read 函数或者使用变量传输步骤（Transmission 步骤）读取行数据。ChannelType 为 Output 时调用 Variant 的 Write 函数或者使用变量传输步骤写入文件。

应用型变量 – BIN 文件变量（派生自通道容器变量）

BIN 文件变量是一个容器类变量，可以容纳任意类型子变量，可以以二进制的方式对文件进行读写，以容器中的子变量的字节长度为单位进行读写。

属性	描述
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。
Endian	文件存储的字节序，BigEndian 为大端次序，LittleEndian 为小端次序；例如，选择 BigEndian 表示文件存储一个数据是以大端方式存储的，即高

	字节在前，低字节在后。
FileName	文件名。
StorageDuration	存储时长，0 表示不分文件存储，大于 0 表示按时长分隔文件。
StorageDurationUnit	存储时长单位，支持 Second、Minute、Hour、Day，其中 Hour 和 Day 在生成文件时除第一个文件外，其他文件对齐整点单位，例如单位是 Day 则按天对齐，在 0 点整切分文件。
StorageQuantity	存储数量，0 表示无限个，大于 0 表示数量超过设定值后，循环删除最早的文件。该属性在 StorageDuration 大于 0 时有效。
TimingType	SampleClock 类型：ChannelType 为 Input 时自动定时每次读取的字节数是所有子变量的数据长度，ChannelType 为 Output 时自动定时将最新的数据记录写入数据库； Demand 类型：ChannelType 为 Input 时调用 Variant 的 Read 函数或者使用变量传输步骤（Transmission 步骤）读取行数据。ChannelType 为 Output 时调用 Variant 的 Write 函数或者使用变量传输步骤写入文件。

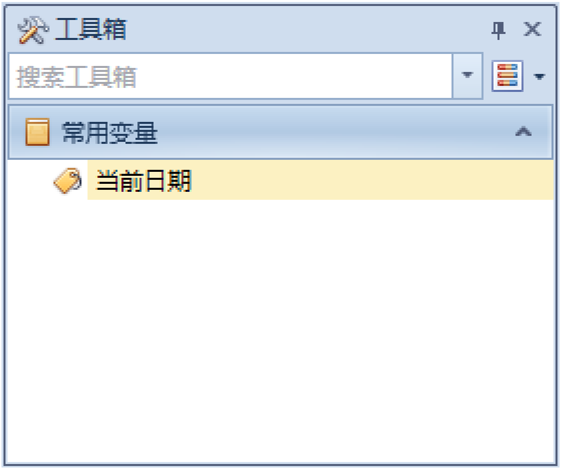
应用型变量 – TDM 文件变量（派生自通道容器变量）

TDM 文件变量是一个容器类变量，可以容纳任意类型子变量，可以对 NI 公司的 TDM/TDMS 格式文件进行读写。

属性	描述
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。
FileType	文件类型，支持 TDM 和 TDMS 两种格式。
StorageDuration	存储时长，0 表示不分文件存储，大于 0 表示按时长分隔文件。
StorageDurationUnit	存储时长单位，支持 Second、Minute、Hour、Day，其中 Hour 和 Day 在生成文件时除第一个文件外，其他文件对齐整点单位，例如单位是 Day 则按天对齐，在 0 点整切分文件。
StorageQuantity	存储数量，0 表示无限个，大于 0 表示数量超过设定值后，循环删除最早的文件。该属性在 StorageDuration 大于 0 时有效。
TimingType	SampleClock 类型：ChannelType 为 Input 时自动定时每次读取 1 行记录，ChannelType 为 Output 时自动定时将最新的数据记录写入数据库； Demand 类型：ChannelType 为 Input 时调用 Variant 的 Read 函数或者使用变量传输步骤（Transmission 步骤）读取行数据。ChannelType 为 Output 时调用 Variant 的 Write 函数或者使用变量传输步骤写入文件。

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。

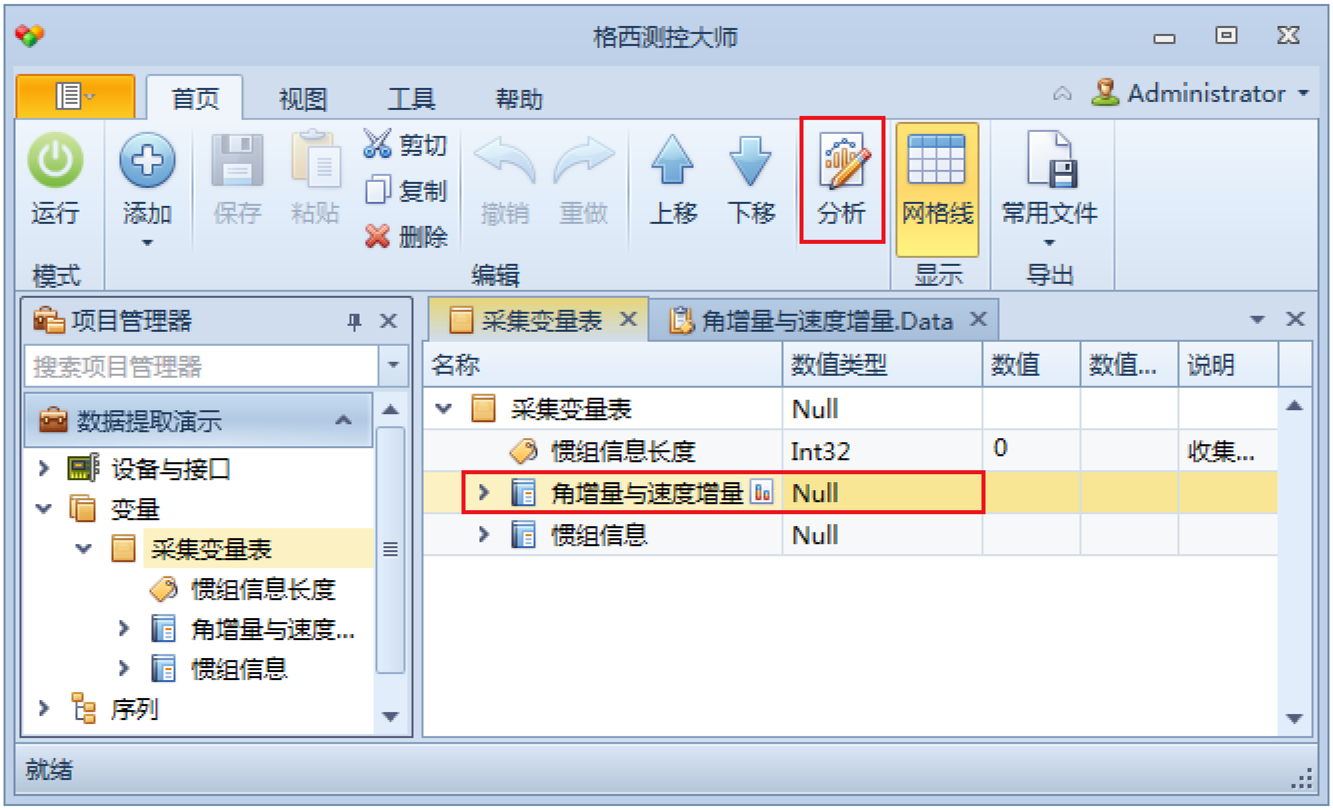


3.2.4 变量数据的采集与分析

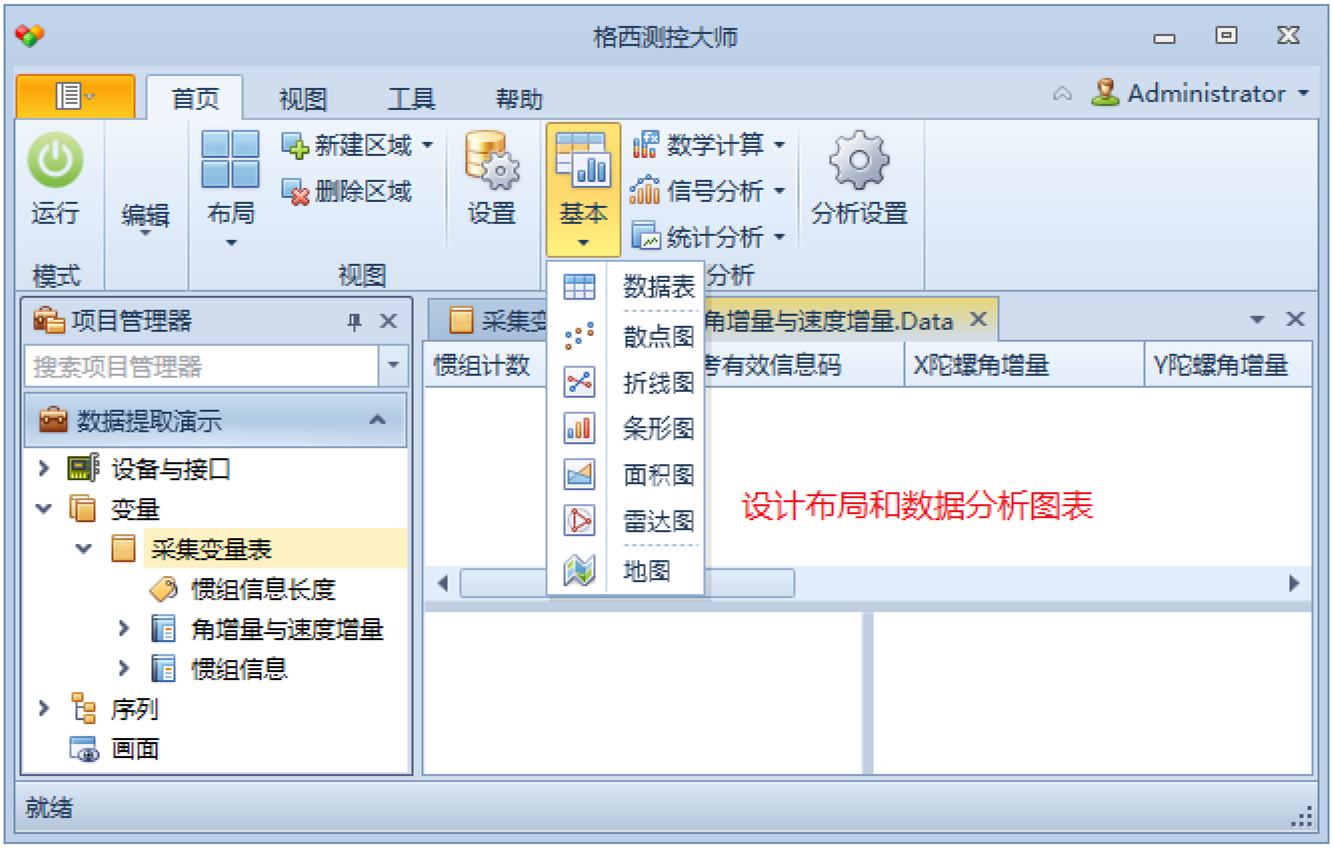
软件支持在运行时对变量数据进行显示、采集和分析。

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。

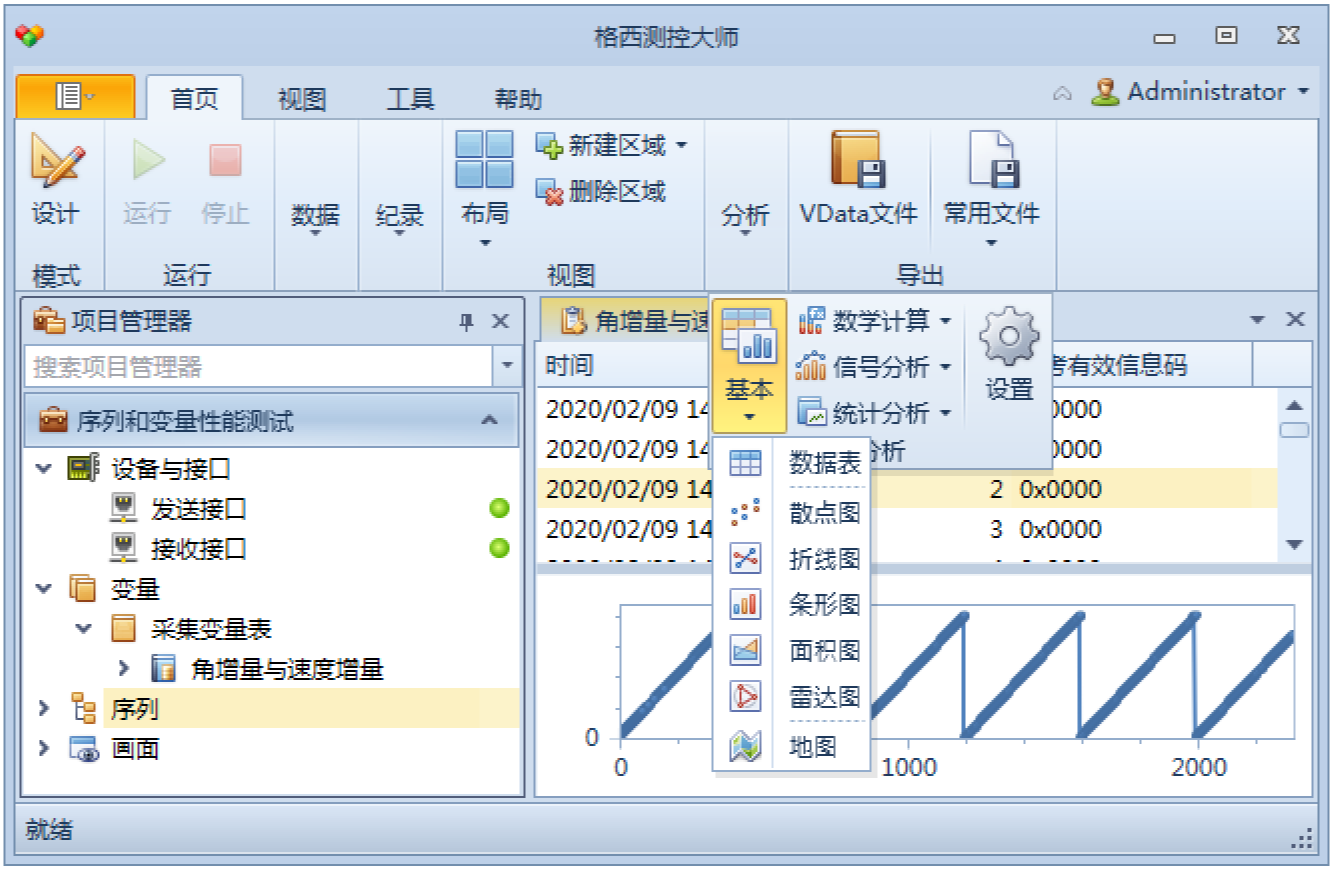
设计时，从项目管理器中打开变量编辑器，然后选择需要采集和分析的变量，点击工具栏的“分析”按钮，打开变量数据分析设计页面，如图所示。



名称	数值类型	数值	数值...	说明
采集变量表	Null			
惯组信息长度	Int32	0		收集...
角增量与速度增量	Null			
惯组信息	Null			



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。



时间		
2020/02/09 14		0000
2020/02/09 14		0000
2020/02/09 14	2	0x0000
2020/02/09 14	3	0x0000

关于变量数据的用法例子，请参考：<软件安装目录>\Examples\Solutions\DataCommunication\DataExtractor。

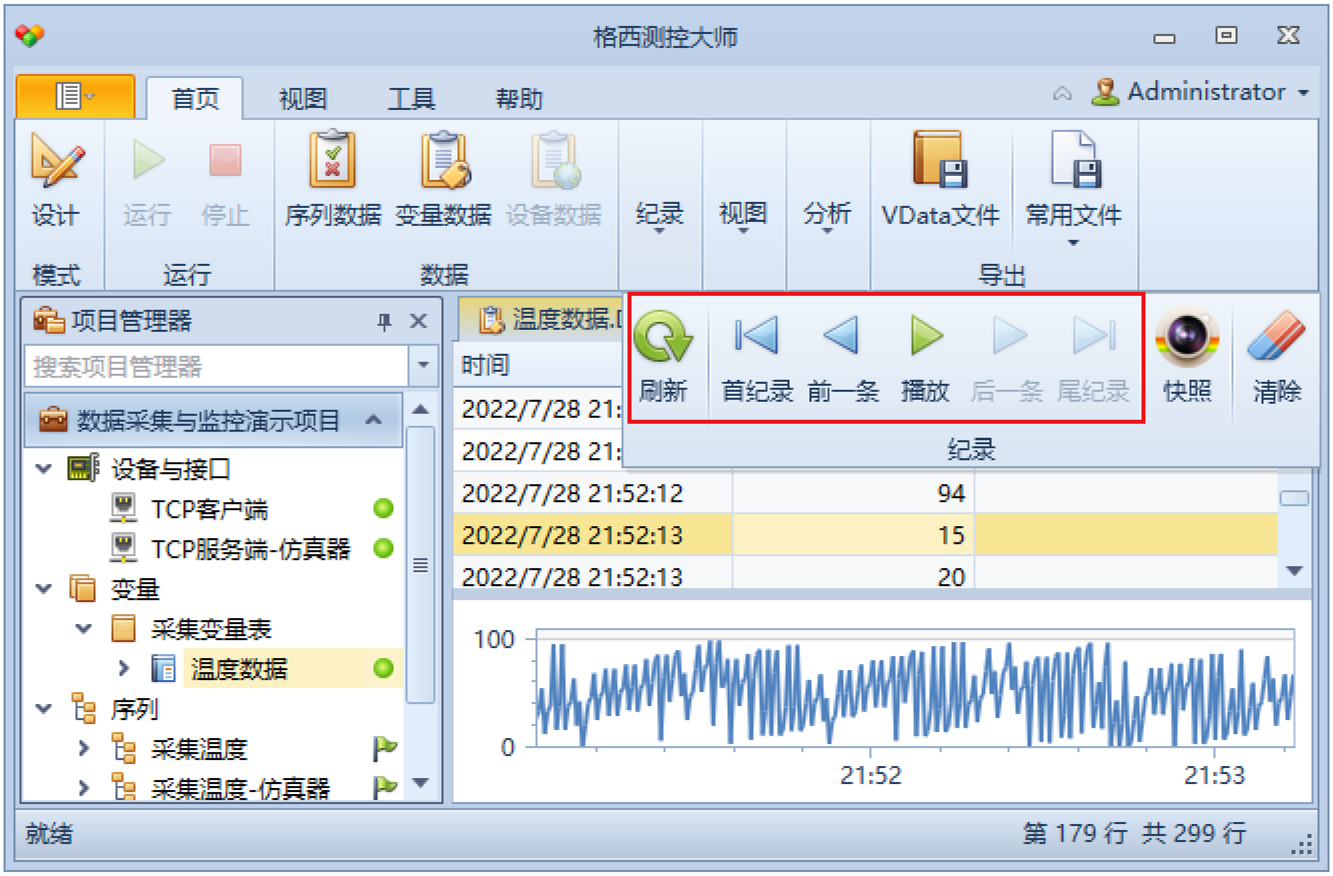
3.2.5 变量数据的快照

软件支持对变量数据进行快照，存储当前变量的缓存数据。可以通过三种方式抓取变量的快照数据：第一种方式是通过 VariantStorage 步骤存储；第二种方式是通过变量数据面板的“快照”按钮；第三种方式是通过脚本类 ProjectContext 的 SaveVariantData 函数。保存的快照文件后缀名为.vdata，是一个 SQLite 数据库，可以通过数据管理器打开。

3.2.6 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。



时间	
2022/7/28 21:	
2022/7/28 21:	
2022/7/28 21:52:12	94
2022/7/28 21:52:13	15
2022/7/28 21:52:13	20

3.2.7 *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。

- 支持动作型步骤，动作型步骤支持判错，支持数值型动作步骤（Value）、消息型动作步骤（Message）、寄存器型动作步骤（Register）、进程型动作步骤（Process）。
- 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）、指示（Indication）。
- 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。
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，单位是毫秒。>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。
Result	步骤执行结果，详情请参考 3.7.14。

动作类 – 值（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。 详情请参考 3.5 节表达式。
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。

值（Value）步骤是值类型动作步骤，该步骤根据 LimitExpression 来判断是否通过，表达式支持判断项目变量表的变量值，支持判断执行结果变量值；可以使用脚本执行测试和控制逻辑，得到执行数值后，建立一个 Variant 变量并赋值给 Value 属性。

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 – 消息（Message）步骤

属性	描述
ByteInterval	协议字节间隔时间，单位是毫秒。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。设备类型只能是消息型设备（IMessageDeviceSession）类型。
EscapeParameters	转义参数，格式：属性=属性值,...。 支持 StartPosition 和 EndPosition。StartPosition>=0 和 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。 例如：StartPosition=1,EndPosition=-2（默认值）
EscapeTable	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
LimitExpression	限定值表达式。 详情请参考 3.5 节表达式。
MaxLength	协议帧最大字节数，大于零时有效。
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。
OperationMode	操作模式，有发送（Send）和接收（Receive）两种模式。

消息（Message）步骤是消息类型动作步骤，该步骤可以自定义任意格式的协议报文（详见 3.3.3 消息型步骤的内容结构），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 – 寄存器（Register）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。设备类型只能是寄存器型设备（IRegisterDeviceSession）类型。
LimitExpression	限定值表达式。 详情请参考 3.5 节表达式。
OperationMode	操作模式，有读（Read）和写（Write）两种模式。

寄存器（Register）步骤是寄存器类型动作步骤，该步骤可以访问寄存器类型设备的寄存器数值，可以一次访问任意数量的设备寄存器（详见 3.3.4 寄存器型步骤的内容结构）。

动作类 – 进程（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。

	详情请参考 3.5 节表达式。
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。
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。

进程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。

进程步骤对外部可执行文件的编写要求：

- 进程终止时返回的 int 类型值作为进程步骤的结果状态值，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，都被保存到进程步骤的结果变量 Result.Data 中，用户可以对该值进行处理。

流程控制类 – 如果（If）步骤、那么（Then）步骤、否则（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"] 详情请参考 3.5 节表达式。

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 – 循环（For）步骤

属性	描述
StartExpression	初始值表达式，如 1。 详情请参考 3.5 节表达式。
IncrementExpression	增量值表达式，可以是正数或负数。 详情请参考 3.5 节表达式。
EndExpression	结束值表达式，如 3，Variants["Vars/Int1"]等。 详情请参考 3.5 节表达式。

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 – 循环（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 详情请参考 3.5 节表达式。

流程控制类 – 中断（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 – 继续（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 – 开关（Switch）步骤

属性	描述
DiscriminantExpression	判别式，一般引用变量的值，如 Variants["Vars/Int1"]。 详情请参考 3.5 节表达式。

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 – 开关分支（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。 详情请参考 3.5 节表达式。

Case 步骤只能配合 Switch 步骤使用。

流程控制类 – 并行（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 – 并行分支（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。

变量类 – 变量赋值步骤

属性	描述
Variant	变量名，如：变量集 1/变量 1。
Expression	变量赋值表达式。当表达式为空时，复位变量，清空变量历史数据。

变量赋值（Variant Assignment）步骤是给变量名为 Variant 的变量赋值的操作，使用 Expression 属性表示的表达式进行赋值。

变量类 – 变量传输步骤

属性	描述
Count	传输数量，如果 Source 的数据量不足，则按实际数量传输。
Destination	目的变量名，如：变量集 1/变量 1。
Source	源变量名。

变量传输（Variant Transmission）步骤是从变量名为 Source 的变量中读出数量为 Count 的最新数据，赋值给变量名为 Destination 的变量。当 Destination 为空时，不执行赋值操作。该步骤可以和通道类变量一起使用，达到按需控制通道变量的数据更新。

变量类 – 变量存储步骤

属性	描述
Variant	变量名，如：变量集 1/变量 1。
Location	存储文件位置表达式。例如： 绝对路径："D:\\Temp\\Test.vdata" 相对项目文件所在目录的路径："Test.vdata" 取当前日期时间格式化路径：Format("Test{0}.vdata", Now.ToString("yyyyMMddHHmmss"))

变量存储（Variant Storage）步骤是存储变量缓存数据的操作，可以对指定变量的缓存数据做快照存储。

同步类 – 等待步骤

属性	描述
Time	等待时间，单位是毫秒，表示等待一定时间再执行后续的步骤。支持表达式计算，例如： Variants[“变量集 1/变量 1”]+100
HighPrecisionLimit	高精度延时阈值，单位是毫秒；当该值>0 时，剩余时间低于该值后的使用高精度延时；触发高精度延时会消耗更多 CPU 资源。

同步类 – 通知步骤

属性	描述
ContentTopic	通知的内容主题，可以是一个字符串，也可以引用一个变量，引用格式用@()符号括住，如@(变量集 1/变量 1)，如果是变量引用，执行步骤相当于把 Content 属性的值赋给该变量。
Content	通知的内容，是一个表达式，可以为空。

通知（Notification）步骤是用于并行执行中不同分支的同步，同步的方式可以用消息，也可以用一个变量，通常和 Indication 步骤联合一起使用。

同步类 – 指示步骤

属性	描述
ContentTopic	指示的内容主题，可以是一个字符串，也可以引用一个变量，引用格式用@()符号括住，如@(变量集 1/变量 1)。
Content	指示的内容，是一个表达式，可以为空。当指示的内容和收到的内容一致时，步骤完成执行。如果为空，则收到指示的内容主题即可完成执行。

指示（Indication）步骤用于等待同步信号，等待的信号可以是消息，也可以是一个变量发生变化后的变量值，通常和 Notification 步骤联合一起使用。

容器类 – 序列步骤

序列（Sequence）步骤是容器类型步骤，可以容纳的任意类型步骤，使用层次化方式组织执行序列。

属性	描述
SequenceType	序列类型。 ✧ Container – 容器类型，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。 ✧ Sequence- 序列类型，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。

容器类 – 序列容器步骤

序列容器（SequenceContainer）步骤是容器类型步骤，只能容纳 Sequence 步骤和 SequenceContainer 步骤，使用层次化方式组织 Sequence 步骤。

属性	描述
SequenceType	序列类型，参考 Sequence 步骤的定义。

注：使用 SequenceContainer 组织用例集， Sequence 组织用例，可以满足粗粒度和细粒度的数据管理和报表管理要求。

工具类 – 声音步骤

声音（Audio）步骤用于播放.wav 文件声音。

属性	描述
FileName	声音文件的文件名，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。

工具类 – 说话步骤

说话（Speech）步骤用于文字转语音播放。

属性	描述
Text	要朗读的文本
Gender	合成语音的性别
Age	合成语音的年龄
Volumn	合成语音的声响，0 到 100 之间
Rate	合成语音的语速，-10 到 10 之间

3.3.3 消息型步骤的内容结构

消息型步骤的内容描述的是通信交互过程的通信协议数据结构，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。 参考链接：https://docs.microsoft.com/zh-cn/dotnet/standard/base-types/formatting-types
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。 剪切规则： ✧ Boolean、Sbyte、Byte、Int16、UInt16、Int32、Uint32、Int64、Uint64、Float、Double、Decimal – 保留低位，剪切多余的高位。例如，Boolean 为 True，二进制编码 1 字节为 0b0000001，如果 Length 为 1，则剪切后为 0b1。 ✧ String – 数据根据系统设定的字符编码方式编码为字节数组，然后从左到右依次取位，剪切多余的低位。例如，ABC，UTF8 编码 16 进制表示为 0x414243，如果 Length 为 12，则剪切后为 0x41,0b0100。 ✧ BitString - 从左到右依次取位，剪切多余的低位。注意：BitString 在创建的时候会根据初始化字符串自动生成位串长度。例如，Length 为 1 时，BitString(0x1)剪切后为 0b0（0x1 相当于 0b0001,一个 4 位的 BitString），BitString(0b1)剪切后为 0b1，BitString(0b11)剪切后为 0b1。
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，支持通过绑定变量的方式自动双向设置。 ✧ 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 – 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36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中的ProtocolAlgorithmExt1.cs。 扩展算法的代码除了使用外部DLL的方式编写外，也可以直接在脚本中编写。
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。
Display	计算单元的结果显示方式，Original表示采用计算前数据显示，Computed表示采用计算后的数据显示。
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.gpj 的“序列\基本步骤\ActionMessage 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 寄存器型步骤的内容结构

寄存器型步骤的内容描述是一张寄存器表，定义了需要读或写的一组寄存器。

属性	描述
(Name)	寄存器单元的名称，可以是任意字符串。
Address	寄存器单元的地址，地址的格式由设备驱动定义。
DataType	寄存器单元的数据类型，软件支持的类型有 Boolean、Sbyte、Byte、Int16、UInt16、Int32、Uint32、Int64、Uint64、Float、Double、Decimal、DateTime、String、BitString。
Description	寄存器单元的描述字符串。
Endian	寄存器单元的字节序。 ✧ 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	寄存器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。 参考链接：https://docs.microsoft.com/zh-cn/dotnet/standard/base-types/formatting-types
Length	寄存器单元的位长，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。对于 String 或者 BitString 类型的数据，可以设置为负数，表示不定长。
Value	寄存器单元的数值，根据设置的 DataType 和 Length 来设置，支持通过绑定变量的方式自动双向设置。 ✧ 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 – 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	寄存器单元是否解析和显示。

3.3.5 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
{
    /**
     项目上下文
     */
    public ProjectContext 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;
    }
}
```

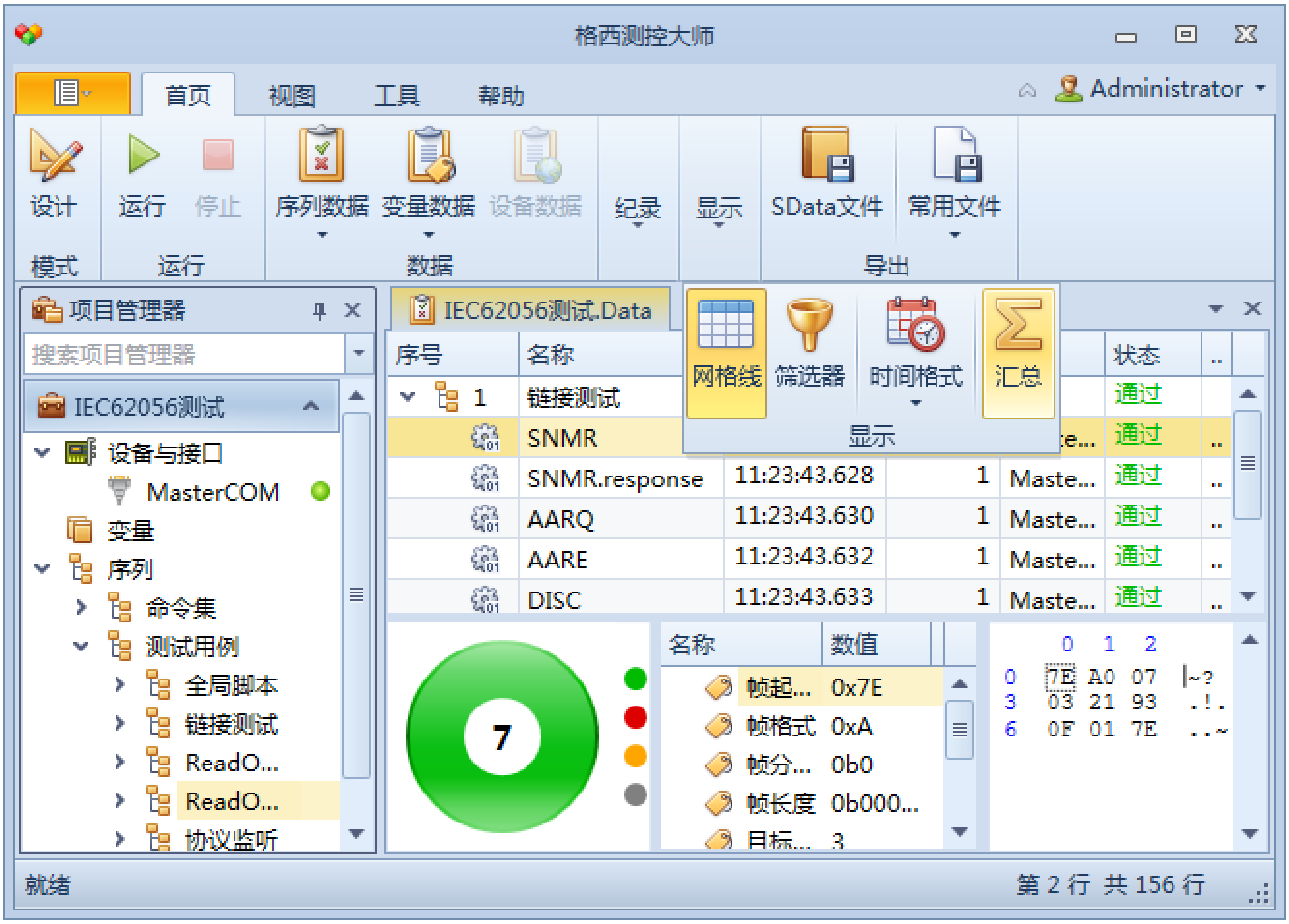
注意：本软件的步骤脚本类和画面脚本类，用于在需要执行脚本函数时临时创建一个对象作为容器媒介来执行脚本函数，不是持久存在的一个对象，故不能使用私有变量保存状态。需要状态保存，可以使用项目上下文变量 Context 的数据操作函数或者创建静态类保存。

3.3.6 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。

3.3.7 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。



序号	名称				状态	..
1	链接测试				通过	
	SNMR			...e...	通过	..
	SNMR.response	11:23:43.628	1	Maste...	通过	..
	AARQ	11:23:43.630	1	Maste...	通过	..
	AARE	11:23:43.632	1	Maste...	通过	..
	DISC	11:23:43.633	1	Maste...	通过	..

名称	数值
帧起...	0x7E
帧格式	0xA
帧分...	0b0
帧长度	0b000...
目标...	3

```
   0  1  2
0  7E A0 07  |~?
3  03 21 93  .!.
6  0F 01 7E  ..~
```

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。

● 17 (94.44%) 通过
● 0 (0.00%) 失败
● 1 (5.55%) 终止
● 0 (0.00%) 完成

94.44%
通过

名称	执行次数	执.. (ms)	通过	通过率	失败	失败率	终止	终止率	完..	完..
链接测试	1	8	1	100.00%	0	0.00%	0	0.00%	0	0...
ReadObject（弹出对话框）	1	1644	1	100.00%	0	0.00%	0	0.00%	0	0...
ReadObject（Associatio...	16	15580	15	93.75%	0	0.00%	1	6.25%	0	0...

就绪 第 3 行 共 3 行

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、倾斜、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。

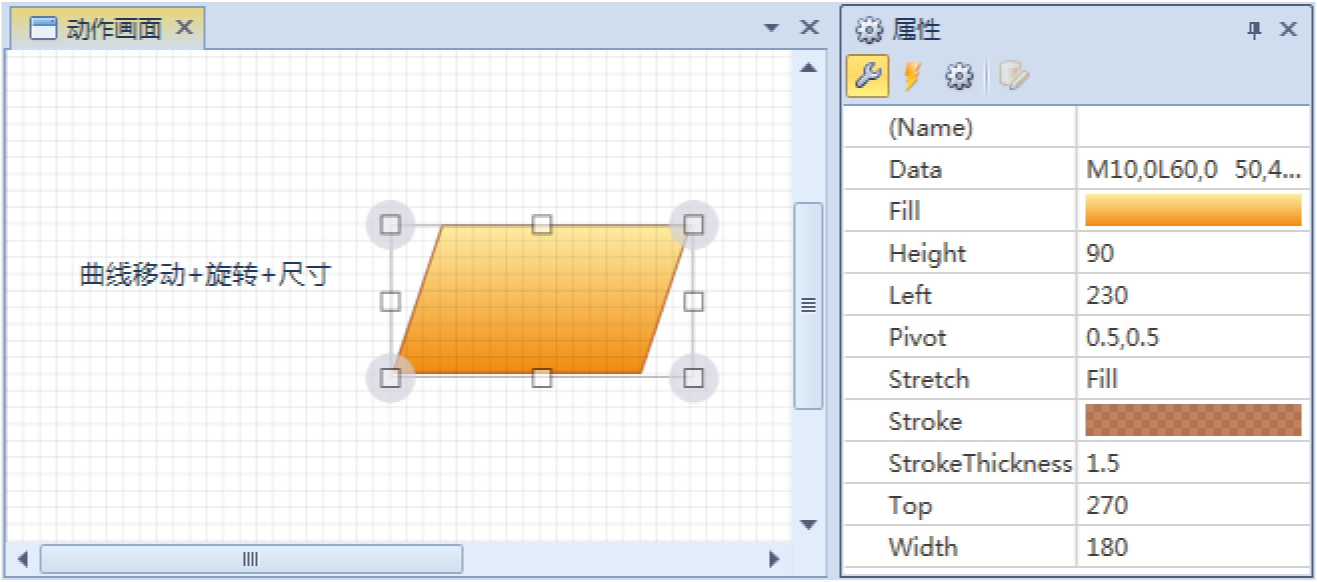
3.4.2 画面元素

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。

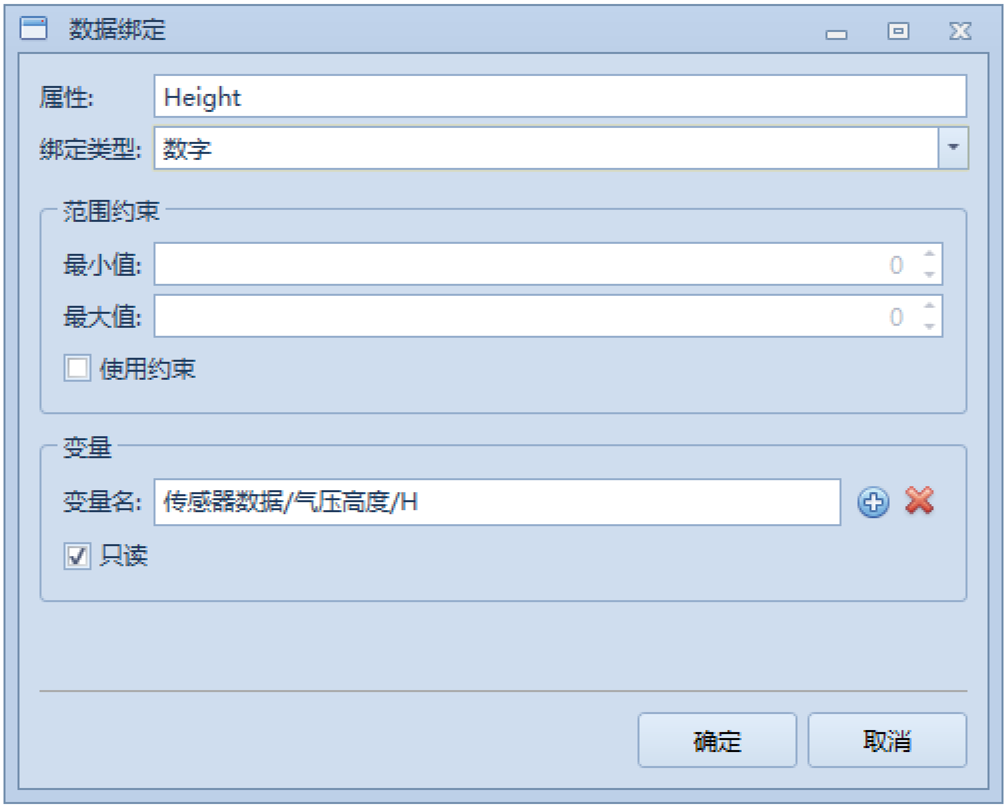


(Name)	
Data	M10,0L60,0 50,4...
Fill	🟧
Height	90
Left	230
Pivot	0.5,0.5
Stretch	Fill
Stroke	▨
StrokeThickness	1.5
Top	270
Width	180

画面元素的属性还可以进行数据绑定，通过数据绑定功能，可以让元素的属性跟随变量的变化而变化，从而实现丰富的数据展现功能。软件支持“数字绑定”、“表达式绑定”、“字符串绑定”、“符合字符串绑定”、“画刷绑定”、“离散画刷绑定”以及“数据提供者绑定”。

数字型绑定

数字型绑定支持数字型属性、布尔型属性，即属性的数据类型是数字型或布尔型的，如 bool、int、double 等。



属性: Height
绑定类型: 数字

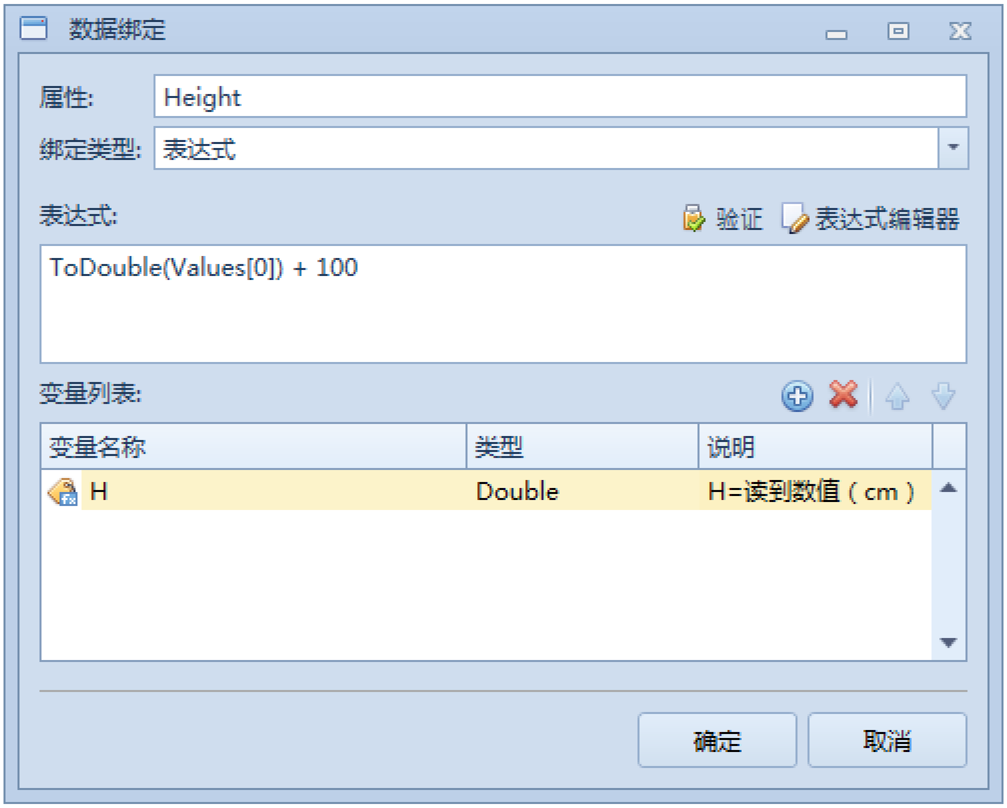
最小值: 0
最大值: 0
☐ 使用约束

变量名: 传感器数据/气压高度/H
☑ 只读

定义	描述
变量名	绑定的变量完整路径名称。
变量只读	勾选只读，绑定方向是从变量到元素属性；不勾选，则是双向的，变量值变化，可以反映到属性，属性值更改也能导致变量值变化。
范围最大值	元素属性接受的最大值，绑定的变量超过最大值，取设定的范围最大值。
范围最小值	元素属性接受的最小值，绑定的变量超过最小值，取设定的范围最小值。
使用约束	是否使用范围约束。

表达式型绑定

表达式型绑定可以支持数字型属性、布尔型属性、字符串型属性，如 bool、int、float、double、string 等。



属性: Height

绑定类型: 表达式

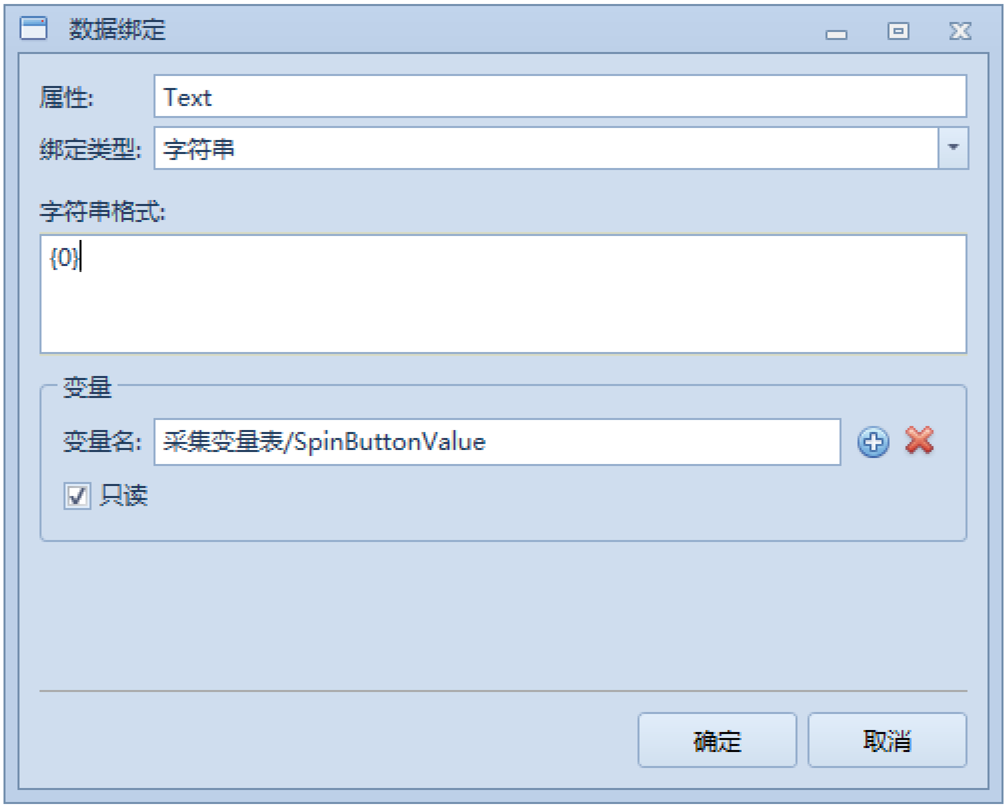
表达式: ToDouble(Values[0]) + 100

变量名称	类型	说明
H	Double	H=读到数值（cm）

定义	描述
变量列表	进行数据绑定的一个变量列表，支持添加、删除和上下移动变量位置，列表中的变量按顺序依次表示为 Values[0]、Values[1]等。
表达式	用于计算最终赋给属性的值，表达式中使用到的变量值必须用对应的转换函数转换，使得表达式中的数据类型一致。例如，属性是 double 类型的，绑定的变量加 100 后赋值给属性，则表达式为 ToDouble(Values[0])+100。

字符串型绑定

字符串型绑定可以支持字符串型属性、对象型属性，如 string、object 等。



属性: Text

绑定类型: 字符串

字符串格式:

{0}

变量

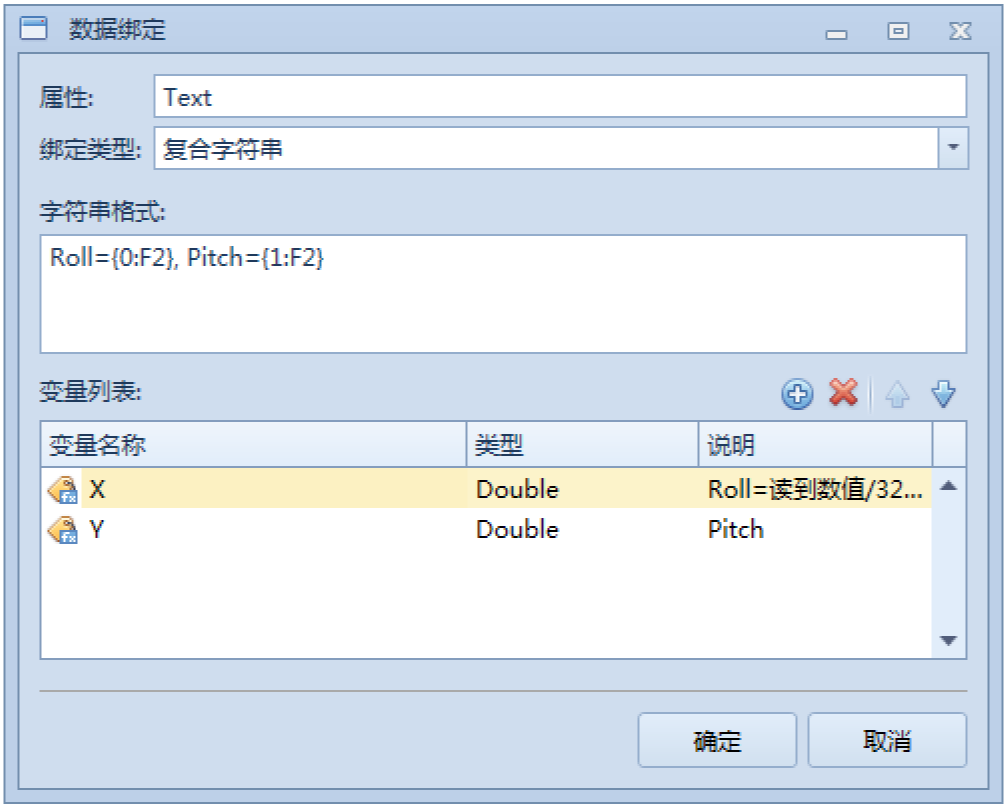
变量名: 采集变量表/SpinButtonValue

☑ 只读

定义	描述
变量名	绑定的变量完整路径名称。
变量只读	勾选只读，绑定方向是从变量到元素属性；不勾选，则是双向的，变量值变化，可以反映到属性，属性值更改也能导致变量值变化。
字符串格式	字符串的格式完全和微软.Net 类库的 String.Format 函数的格式字符串参数定义一致的，详情可以参考：https://docs.microsoft.com/zh-cn/dotnet/api/system.string.format?view=netframework-4.8#Starting

复合字符串型绑定

复合字符串型绑定可以支持字符串型属性、对象型属性，如 string、object 等。



属性: Text

绑定类型: 复合字符串

字符串格式: Roll={0:F2}, Pitch={1:F2}

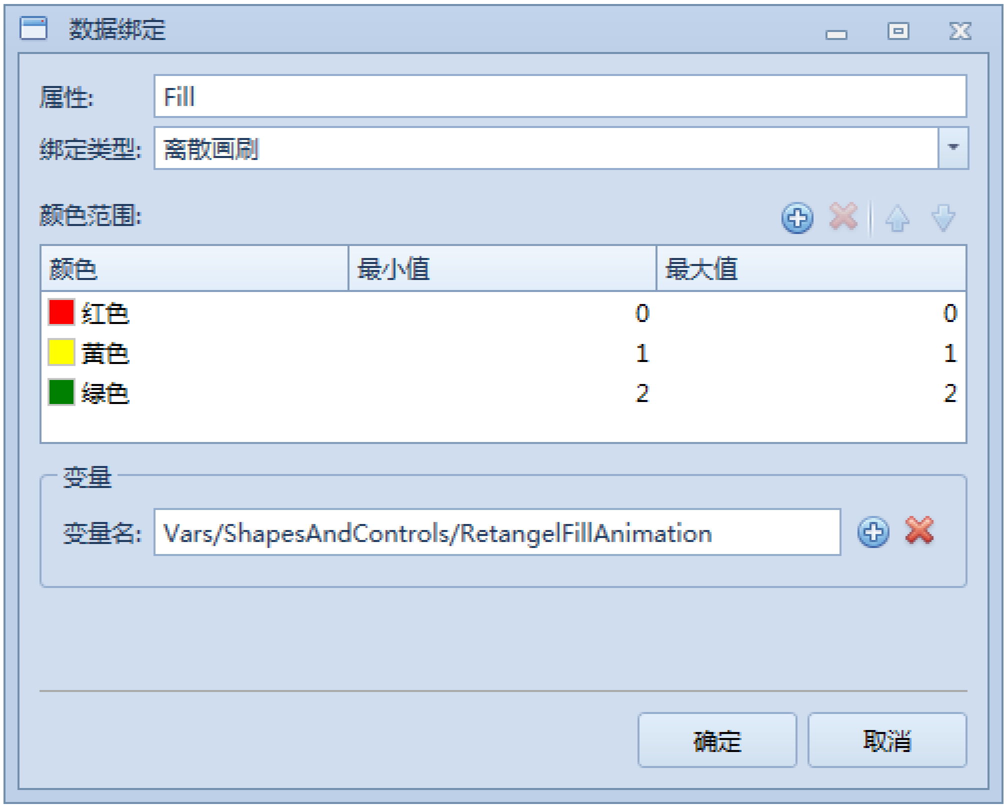
变量列表:

变量名称	类型	说明
X	Double	Roll=读到数值/32...
Y	Double	Pitch

定义	描述
变量列表	进行数据绑定的一个变量列表，支持添加、删除和上下移动变量位置，列表中的变量按顺序依次表示为{0}、{1}等格式项。
字符串格式	字符串的格式完全和微软.Net 类库的 String.Format 函数的格式字符串参数定义一致的，详情可以参考：https://docs.microsoft.com/zh-cn/dotnet/api/system.string.format?view=netframework-4.8#Starting 例如，绑定两个变量，保留两位小数，输出属性的值的格式如 Roll=0.00,Pitch=1.11，则字符串格式为 Roll={0:F2},Pitch={1:F2}

离散画刷型绑定

离散画刷型绑定可以支持画刷型属性，即数据类型为 Brush 的属性。用于变量值在一定范围内，保持一种颜色，而在另一个范围内，颜色变为另一种的场合。



属性: Fill

绑定类型: 离散画刷

颜色范围:

颜色	最小值	最大值
🟥 红色	0	0
🟨 黄色	1	1
🟩 绿色	2	2

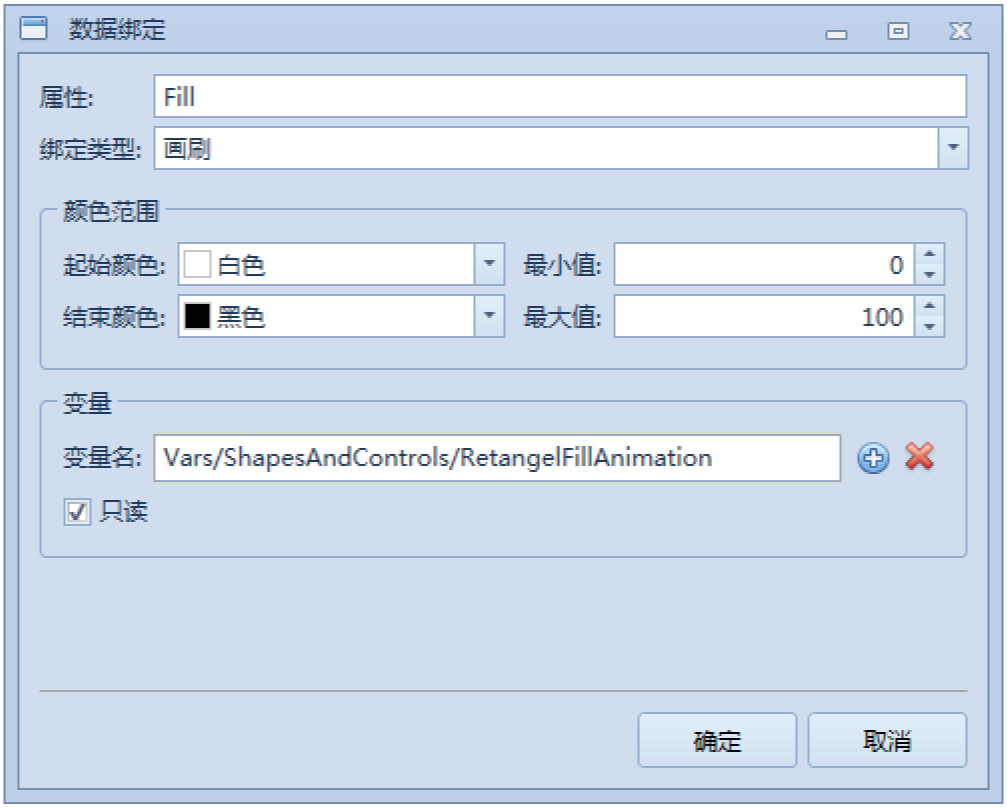
变量

变量名: Vars/ShapesAndControls/RetangelFillAnimation

定义	描述
变量名	绑定的变量完整路径名称
最小值和最大值	表示绑定的变量的值在这个范围内所取的颜色值。 例如，变量值为 0 时取红色，变量值为 1 是取黄色，变量值为 2 时取绿色，则可以按上图所示进行配置。

画刷型绑定

画刷型绑定可以支持画刷型属性，即数据类型为 Brush 的属性。用于颜色随变量连续变化的场合，颜色的变化算法是从变量最小值到最大值，颜色从起始颜色到结束颜色的 RGB 值按比例变化。



属性: Fill
绑定类型: 画刷

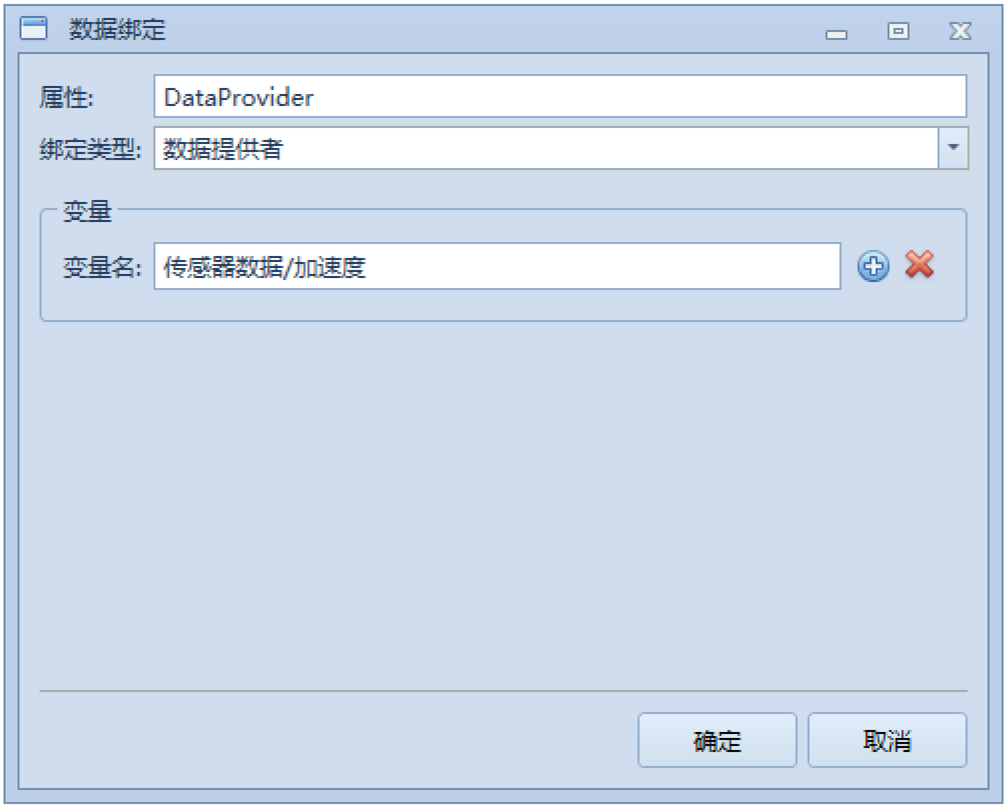
颜色范围
起始颜色: 白色
最小值: 0
结束颜色: 黑色
最大值: 100

变量
变量名: Vars/ShapesAndControls/RetangelFillAnimation
☑ 只读

定义	描述
变量名	绑定的变量完整路径名称
变量只读	勾选只读，绑定方向是从变量到元素属性；不勾选，则是双向的，变量值变化，可以反映到属性，属性值更改也能导致变量值变化。
最大值	元素属性接受的最大值，绑定的变量超过最大值，取设定的最大值。
最小值	元素属性接受的最小值，绑定的变量超过最小值，取设定的最小值。
起始颜色	当变量值为最小值时所取的颜色。
结束颜色	当变量值为最大值时所取的颜色。

数据提供者型绑定

数据提供者型绑定支持数据提供者型属性，即数据类型为 IDataProvider 的属性。软件内置的各种 Chart 控件、TableGrid 控件，均有对应的属性供数据绑定，持续批量获取变量数据。



数据绑定

属性: DataProvider

绑定类型: 数据提供者

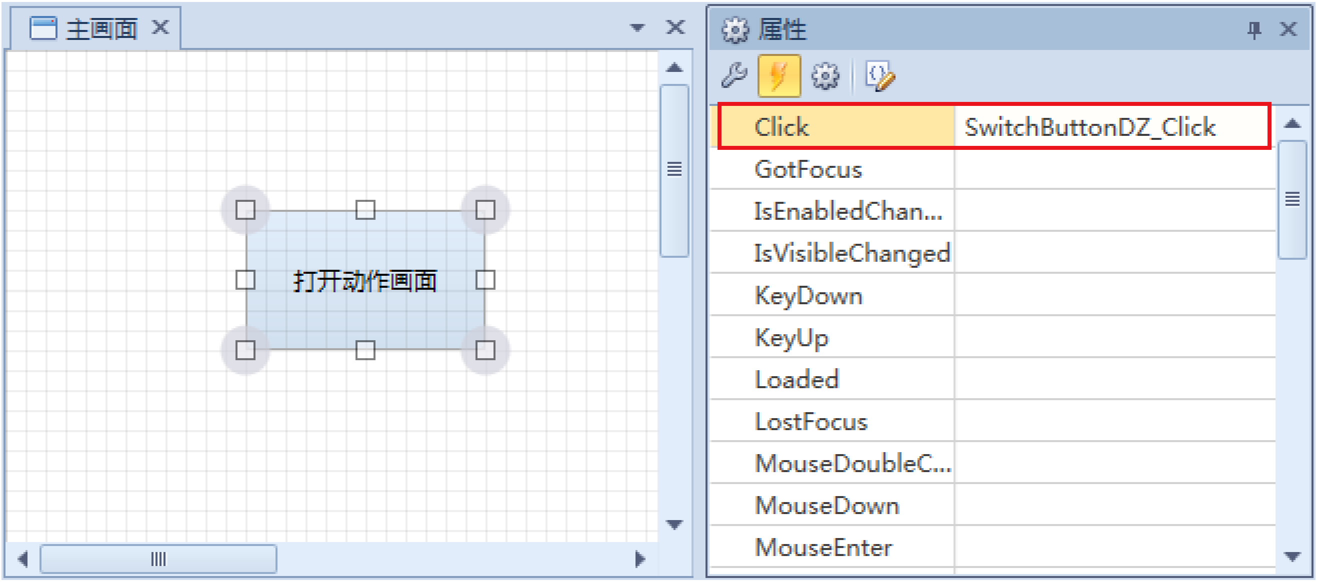
变量

变量名: 传感器数据/加速度

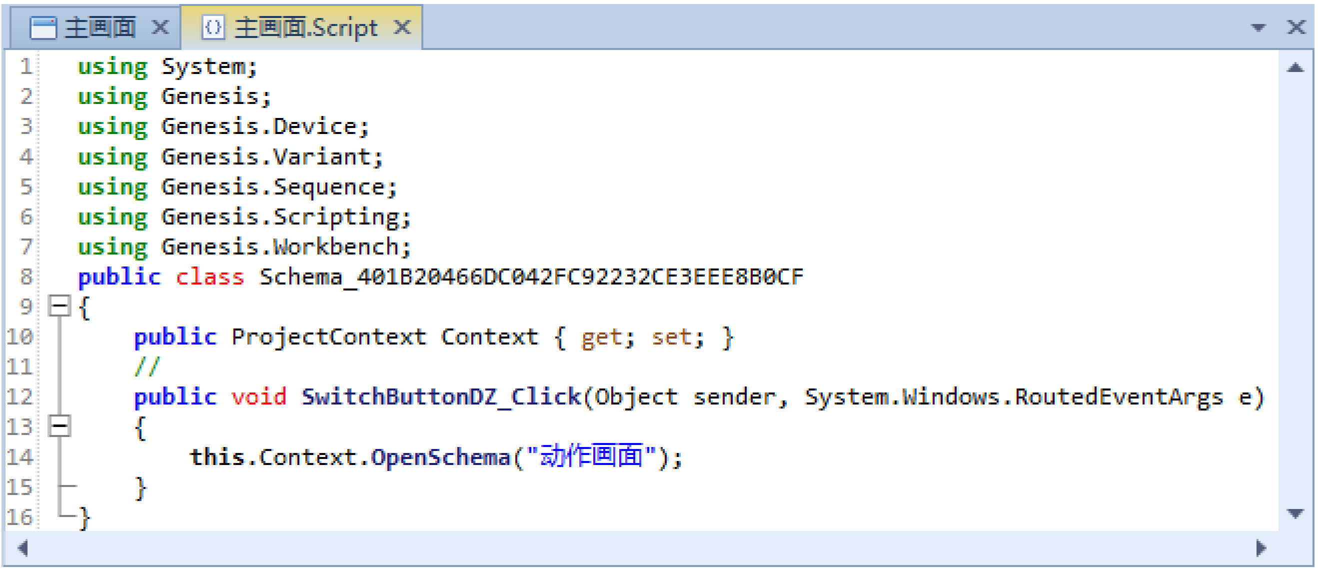
定义	描述
变量名	绑定的变量完整路径名称

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。



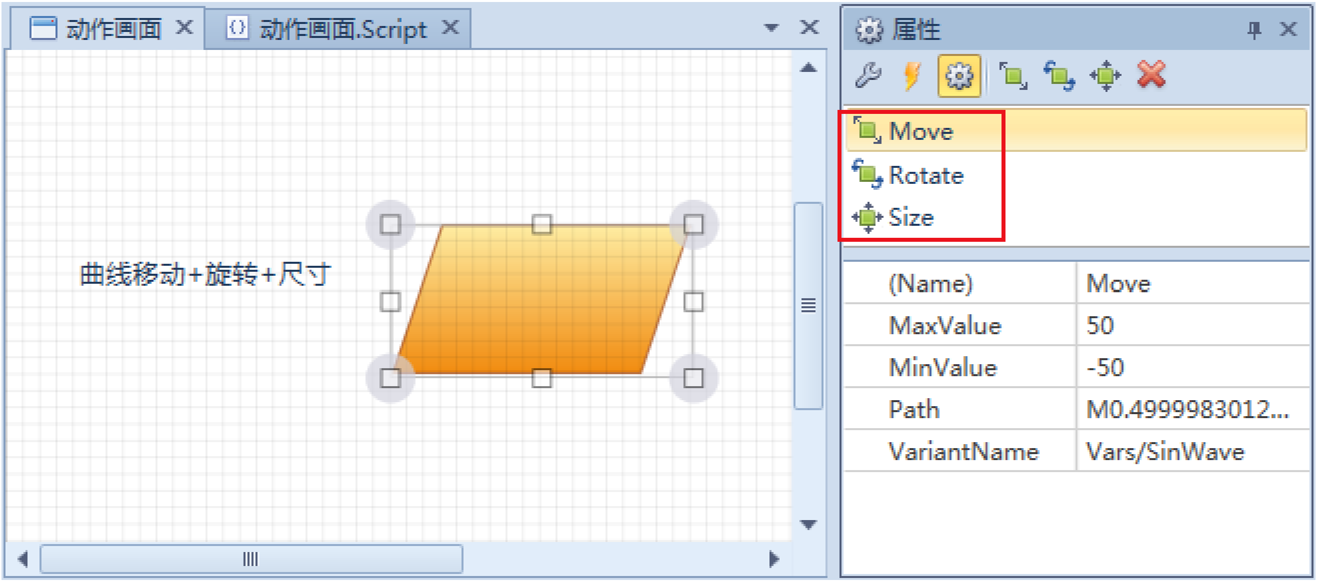
Click	SwitchButtonDZ_Click
GotFocus	
IsEnabledChan...	
IsVisibleChanged	
KeyDown	
KeyUp	
Loaded	
LostFocus	
MouseDoubleC...	
MouseDown	
MouseEnter	



```
using System;
using Genesis;
using Genesis.Device;
using Genesis.Variant;
using Genesis.Sequence;
using Genesis.Scripting;
using Genesis.Workbench;
public class Schema_401B20466DC042FC92232CE3EEE8B0CF
{
    public ProjectContext Context { get; set; }
    //
    public void SwitchButtonDZ_Click(Object sender, System.Windows.RoutedEventArgs e)
    {
        this.Context.OpenSchema("动作画面");
    }
}
```

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”、“倾斜动作”和“尺寸动作”四种动作。



(Name)	Move
MaxValue	50
MinValue	-50
Path	M0.4999983012...
VariantName	Vars/SinWave

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。

Variant	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。
Variant	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。

倾斜动作

定义	描述
(Name)	名称
MaxAngleX	转动的最大角度 X。
MaxAngleY	转动的最大角度 Y。
MaxValue	绑定变量允许变化的最大值。
MinAngleX	转动的最小角度 X。
MinAngleY	转动的最小角度 Y。
MinValue	绑定变量允许变化的最小值。
SkewMode	倾斜模式。 软件支持 Up、Down、Left、Right。
Variant	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[MinAngleX,MaxAngleX]角度和[MinAngleY,MaxAngleY]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。
Operation	尺寸操作方式。 Resize：缩放方式改变元素大小 Crop：剪裁方式改变元素大小
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。
Variant	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。

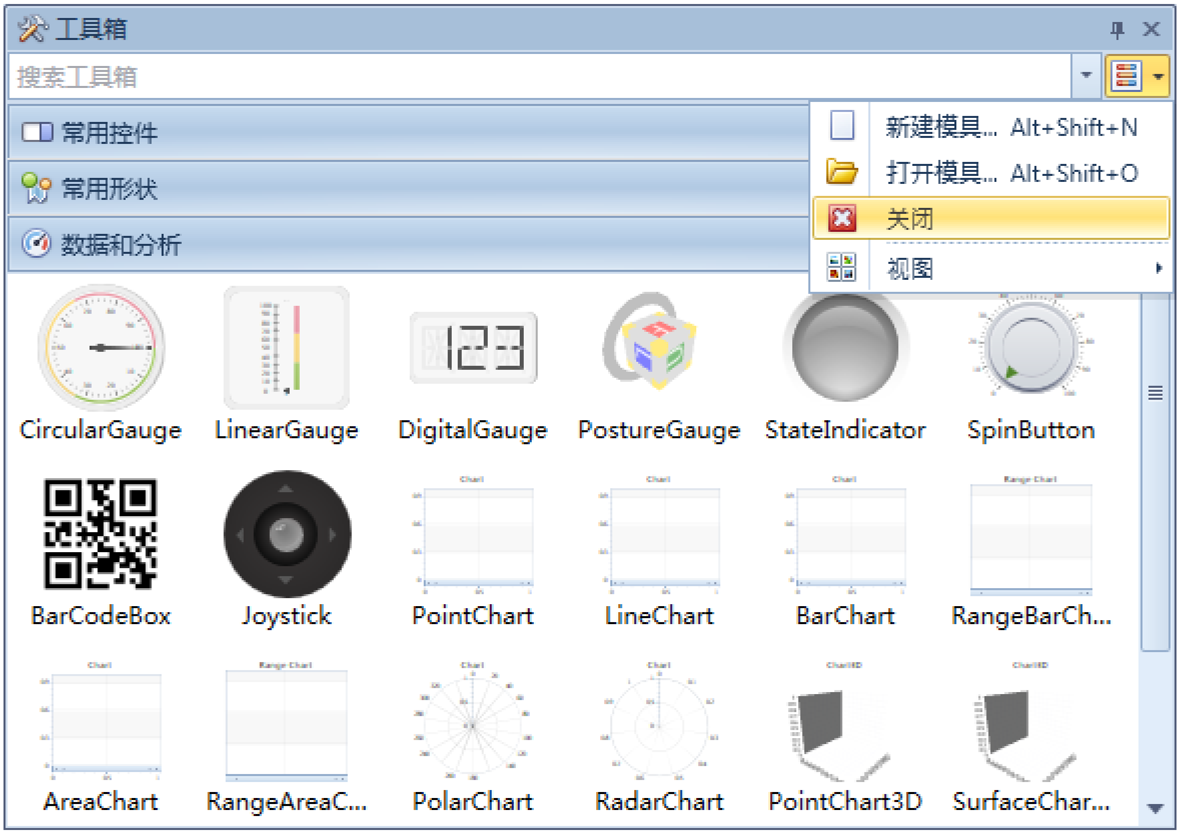
下面是 C#版本的步骤脚本模版，由一个.NET 类 Schema_<画面 Id>构成，该类包含一个类型为 ProjectContext 的 Context 属性。

```
/**
 * 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
{
  /**
   项目上下文
   */
  public ProjectContext 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");
  }
}
```

注意：本软件的步骤脚本类和画面脚本类，用于在需要执行脚本函数时临时创建一个对象作为容器媒介来执行脚本函数，不是持久存在的一个对象，故不能使用私有变量保存状态。需要状态保存，可以使用项目上下文变量 Context 的数据操作函数或者创建静态类保存。

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 表达式

软件支持表达式计算功能，主要应用于动作型步骤、流程控制类型步骤，以及表达式变量等方面。表达式是大小写字母敏感的。

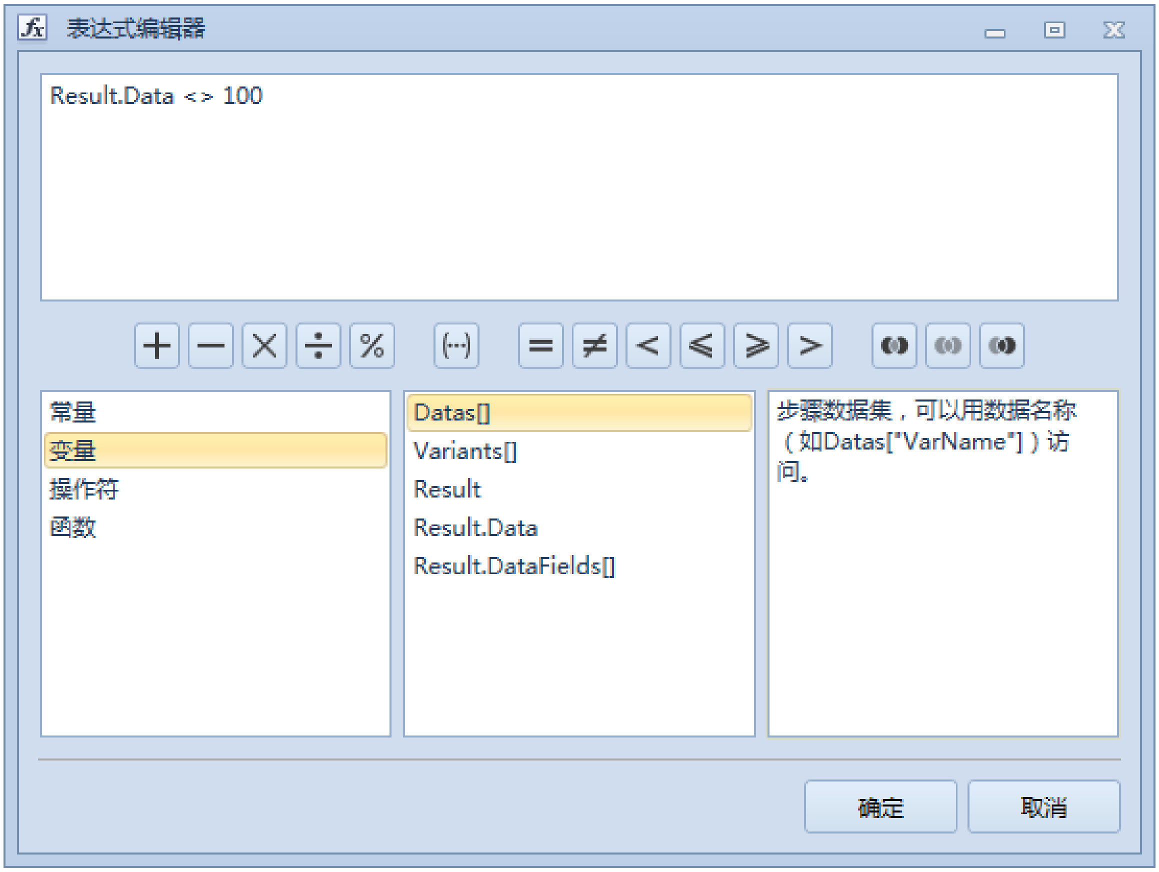
3.5.1 常量

在表达式中支持以下常量：

- Char -单引号中的字符，例如：’a’
- Boolean - true 或者 false
- Real - 任何带小数点的数字。可以使用 d、f 或 m 后缀来指定数字应分别存储为双精度、单精度还是十进制，默认为 double。
- Integral - 任何没有小数点的数字。附加 L 强制数字为 64 位整数，附加 U 强制数字为无符号。软件默认将尝试转为第一个可以包含该值的整数类型。
- Hex –整型常数也可以用十六进制表示法指定，例如：0xFF12
- String -字符串文字用双引号括起来，转义字符遵循与 C#相同的规则，例如：″string\u0021\r\n a \″new\″ line″
- Null -使用关键字 null 会将 null 引用加载到表达式中。
- DateTime – 支持由#包围的有效.NET 日期时间模式，例如：#08/06/2008#.ToLongDateString()
- TimeSpan – 时间范围表示格式为##[d.]hh:mm[:ss[.ff]]#，例如：#08/06/2008# + ##1.23:45#

3.5.2 变量

表达式中支持的变量类型和表达式的应用有关，可以从表达式编辑器的“变量”页面获取。



Result.Data <> 100

步骤数据集，可以用数据名称（如Datas["VarName"]）访问。

3.5.3 *算术运算符*

支持所有标准算术运算符以及模（%）和幂（^）运算符。
例子： a*2 + b^2 - 100%5 （其中 a 和 b 代表变量）

3.5.4 *比较运算符*

支持所有比较运算符，其中不相等运算符为<>，相等运算符为=。
例子： a <> 100

3.5.5 *And/Or/Xor/Not 运算符*

支持 And/Or/Xor/Not 运算符用于逻辑和位运算。如果两个操作数都是布尔运算，则该运算是逻辑运算。如果两者都是整数，则运算是按位进行的。任何其他组合都会导致编译错误。
例子（逻辑运算）： a > 100 And Not b = 100
例子（位运算）： (100 or 2) and 1

3.5.6 *移位运算符*

左（<<）和右（>>）移位运算符执行移位，并且仅对整数类型有效。
例子：100 >> 2

3.5.7 字符串连接

+运算符还用作字符串连接运算符。如果它的任何一个操作数是字符串，它将执行串联而不是加法。只有一个操作数是字符串是有效的，在这种情况下，两个操作数都转换为 Object 并相应地格式化。

例子："abc" + "def"

例子："the number is: " + 100

3.5.8 索引

索引操作符的形式为：member[indexExpression]，任何表达式都可以出现在括号内。索引不是数组且没有默认索引器的类型会生成编译异常。

例子：arr[i + 1] + 100

3.5.9 类型转换

使用 cast 函数执行类型转换，函数形式：cast(value, type)

例子：100 + cast(obj, int)

3.5.10 条件运算符

支持条件运算符 IF，允许您根据布尔条件选择结果，格式：if（condition，whenTrue，whenFalse）

例子：If(a > 100 and b > 10, "both greater", "less")

3.5.11 包含运算符

In 运算符是布尔二进制运算符，如果第一个操作数包含在第二个操作数中，则返回 true，否则返回 false，它有两种形式：

- List: 在列表中搜索给定值，格式： value IN (value1, value2, value3,...)。
- Collection: 在单个集合中搜索给定值，格式：value IN collection。其中 collection 变量必须实现 ICollection<T>、IDictionary<K、V>、IList 或 Idictionary，否则编译出错。

例子（List）：If(100 in (100, 200, 300, -1), "in", "not in")

例子（Collection）：If(100 in collection, "in", "not in")

3.5.12 类型的重载运算符

当计算操作数不是原始的算术、比较或转换操作时，系统将查找并使用在操作数上定义的重载运算符。例如可以在自定义类型中创建诸如 a+b（其中 a 和 b 是自定义类型）之类的表达式。

3.5.13 函数库

3.5.13.1 数学函数

定义	描述
double PI	圆周率，3.1415926535897931
double E	自然对数的底，2.7182818284590451
decimal Abs(decimal value)	返回绝对值
double Abs(double value)	
float Abs(float value)	

int Abs(int value)	
short Abs(short value)	
sbyte Abs(sbyte value)	
long Abs(long value)	
double Acos(double d)	返回余弦值为指定数字的角度（弧度）
double Asin(double d)	返回正弦值为指定数字的角度（弧度）
double Atan(double d)	返回正切值为指定数字的角度（弧度）
double Atan2(double y, double x)	返回正切值为两个指定数字的商的角度（弧度）
long BigMul(int a, int b)	生成两个 32 位数字的完整乘积
decimal Ceiling(decimal d)	返回大于或等于指定的十进制数的最小整数值
double Ceiling(double a)	返回大于或等于指定的双精度浮点数的最小整数值
double Cos(double d)	返回指定角度（弧度）的余弦值
double Cosh(double value)	返回指定角度（弧度）的双曲余弦值
double Exp(double d)	返回 e 的指定次幂
decimal Floor(decimal d)	返回小于或等于指定小数的最大整数值
double Floor(double d)	返回小于或等于指定双精度浮点数的最大整数值
double IEEERemainder(double x, double y)	返回一指定数字被另一指定数字相除的余数 x：被除数 y：除数
double Log(double a, double newBase)	返回指定数字在使用指定底时的对数
double Log(double d)	回指定数字的自然对数（底为 e）
double Log10(double d)	返回指定数字以 10 为底的对数
short Max(short val1, short val2)	回两个 16 位有符号的整数中较大的一个
ushort Max(ushort val1, ushort val2)	
int Max(int val1, int val2)	
long Max(long val1, long val2)	
ulong Max(ulong val1, ulong val2)	
float Max(float val1, float val2)	
double Max(double val1, double val2)	
decimal Max(decimal val1, decimal val2)	
uint Max(uint val1, uint val2)	
sbyte Max(sbyte val1, sbyte val2)	
byte Max(byte val1, byte val2)	
int Min(int val1, int val2)	返回两个 32 位有符号整数中较小的一个
decimal Min(decimal val1, decimal val2)	
double Min(double val1, double val2)	
float Min(float val1, float val2)	
ulong Min(ulong val1, ulong val2)	
long Min(long val1, long val2)	
uint Min(uint val1, uint val2)	
ushort Min(ushort val1, ushort val2)	
short Min(short val1, short val2)	
sbyte Min(sbyte val1, sbyte val2)	
byte Min(byte val1, byte val2)	
double Pow(double x, double y)	返回指定数字的指定次幂

	x：要乘幂的双精度浮点数。 y：指定幂的双精度浮点数。
decimal Round(decimal d, int decimals)	将小数值舍入到最接近的整数，并为中点值使用指定的舍入规则
decimal Round(decimal d)	将小数值舍入到最接近的整数值，并将中点值舍入到最接近的偶数
double Round(double value, int digits, int mode)	将双精度浮点值舍入到最接近的整数，并为中点值使用指定的舍入约定 mode：在两个数字之间时如何舍入 d 的规范，0 表示当一个数字是其他两个数字的中间值时，会将其舍入为最接近的偶数，1 表示当一个数字是其他两个数字的中间值时，会将其舍入为两个值中绝对值较小的值
double Round(double value, int digits)	将双精度浮点值舍入到指定数量的小数位，并将中点值舍入到最接近的偶数
double Round(double a)	将双精度浮点值舍入到最接近的整数值，并将中点值舍入到最接近的偶数
decimal Round(decimal d, int decimals, int mode)	将小数值舍入到指定数量的小数位，并为中点值使用指定的舍入规则 d：要舍入的小数 decimals：返回值中的小数位数 mode：同上
int Sign(sbyte value)	返回一个整数，该整数指示 8 位带符号整数的符号
int Sign(decimal value)	返回表示十进制数符号的整数
int Sign(double value)	返回一个整数，该整数表示双精度浮点数字的符号
int Sign(float value)	返回一个整数，该整数表示单精度浮点数字的符号
int Sign(long value)	返回一个整数，该整数指示 64 位带符号整数的符号
int Sign(short value)	返回表示 16 位带符号整数的整数
int Sign(int value)	返回表示 32 位带符号整数的整数
double Sin(double a)	返回指定角度（弧度）的正弦值
double Sinh(double value)	返回指定角度（弧度）的双曲正弦值
double Sqrt(double d)	返回指定数字的平方根
double Tan(double a)	返回指定角度（弧度）的正切值
double Tanh(double value)	返回指定角度（弧度）的双曲正切值
double Truncate(double d)	计算指定双精度浮点数的整数部分
decimal Truncate(decimal d)	计算一个数字的整数部分
int Rnd()	返回一个非负随机整数
int Rnd(int maxValue)	返回一个小于所指定最大值的非负随机整数
int Rnd(int minValue, int maxValue)	返回在指定范围内的随机整数
double RndDouble()	返回一个大于或等于 0.0 且小于 1.0 的随机浮点数
double RndDouble(double minValue, double	返回在指定范围内的随机浮点数

maxValue)	

3.5.13.2 字符串函数

定义	描述
int Compare(String strA, String strB, bool ignoreCase)	比较两个指定的 System.String 对象（其中忽略或考虑其大小写），并返回一个整数，指示二者在排序顺序中的相对位置。 返回值<0：strA 在排序顺序中位于 strB 之前。 返回值=0：strA 与 strB 在排序顺序中出现的位置相同。 返回值>0：大于零 strA 在排序顺序中位于 strB 之后。
int Compare(String strA, int indexA, String strB, int indexB, int length, bool ignoreCase)	
int Compare(String strA, int indexA, String strB, int indexB, int length)	
int Compare(String strA, String strB)	
String Concat(String str0, String str1)	连接 System.String 的两个指定实例。
String Concat(String str0, String str1, String str2)	连接 System.String 的三个指定实例。
String Concat(String str0, String str1, String str2, String str3)	连接 System.String 的四个指定实例。
String Concat(params String[] args)	连接 System.String 的 n 个指定实例。
String Concat(object arg0)	
String Concat(object arg0, object arg1)	
String Concat(object arg0, object arg1, object arg2, object arg3)	
String Concat(object arg0, object arg1, object arg2)	
String Concat(params object[] args)	
String Format(String format, object arg0)	将字符串中的一个或多个格式项替换为指定对象的字符串表示形式。 format：复合格式字符串，与.Net 的字符串格式用法一样。
String Format(String format, object arg0, object arg1, object arg2)	
String Format(String format, object arg0, object arg1)	
String Format(String format, params object[] args)	
bool IsNullOrEmpty(String value)	判断指定字符串是否为空。
bool IsNullOrWhiteSpace(String value)	
String Substring(String value, int startIndex)	获取指定字符串的子串。
String Substring(String value, int startIndex, int length)	

3.5.13.3 日期和时间函数

定义	描述
DateTime Now	获取当前的本地日期时间
DateTime UtcNow	获取当前的 UTC 日期时间
DateTime Today	获取当前的日期，时间为 00:00:00
DateTime DateTime(long ticks)	返回根据 ticks 创建的 DateTime
DateTime DateTime(long ticks, bool isUtc)	
DateTime DateTime(int year, int month, int day)	
DateTime DateTime(int year, int month, int day, int hour, int minute, int second)	
DateTime DateTime(int year, int month, int day, int hour, int minute, int second, bool isUtc)	
DateTime DateTime(int year, int month, int day, int hour, int minute, int second, int millisecond)	
DateTime DateTime(string s)	
int DaysInMonth(int year, int month)	
bool IsLeapYear(int year)	
//	
TimeSpan TimeSpan(long ticks)	返回根据 ticks 创建的 TimeSpan
TimeSpan TimeSpan(int hours, int minutes, int seconds)	
TimeSpan TimeSpan(int days, int hours, int minutes, int seconds)	
TimeSpan TimeSpan(int days, int hours, int minutes, int seconds, int milliseconds)	
TimeSpan TimeSpan(string s)	
TimeSpan TimeSpan(TimeSpan timeSpan)	
TimeSpan FromDays(double value)	
TimeSpan FromHours(double value)	
TimeSpan FromMilliseconds(double value)	
TimeSpan FromMinutes(double value)	
TimeSpan FromSeconds(double value)	
TimeSpan FromTicks(long value)	

3.5.13.4 转换函数

定义	描述
byte[] FromBase64String(string s)	Base64 字符串转为字节数组
string ToBase64String(byte[] inArray, int offset, int length)	
string ToBase64String(byte[] inArray)	字节数组转为 Base64 字符串
// ToBitString	
BitString ToBitString(string s)	参数 s 是带格式的字符串，可以是多个分段，段

	之间用逗号分隔，0x 开头的段为 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，忽略符号串中的空格。例如："0xF1,0b10,0o77",表示两个字节，对应二进制为 0b11110001 10111111。
BitString ToBitString(string s, string format)	指定参数 s 的格式 format 来转换 BitString。format 参数：B 表示 s 是二进制，O 表示 s 是八进制，H 表示 s 是十六进制
// ToBoolean	
bool ToBoolean(decimal value)	转换为 bool
bool ToBoolean(double value)	
bool ToBoolean(float value)	
bool ToBoolean(ulong value)	
bool ToBoolean(object value)	
bool ToBoolean(bool value)	
bool ToBoolean(sbyte value)	
bool ToBoolean(string value)	
bool ToBoolean(byte value)	
bool ToBoolean(ushort value)	
bool ToBoolean(int value)	
bool ToBoolean(uint value)	
bool ToBoolean(long value)	
bool ToBoolean(short value)	
// ToByte	
byte ToByte(sbyte value)	转换为 byte
byte ToByte(bool value)	
byte ToByte(char value)	
byte ToByte(short value)	
byte ToByte(int value)	
byte ToByte(uint value)	
byte ToByte(long value)	
byte ToByte(ulong value)	
byte ToByte(float value)	
byte ToByte(double value)	
byte ToByte(decimal value)	
byte ToByte(string value)	
byte ToByte(object value)	
byte ToByte(ushort value)	
byte ToByte(string value, int fromBase)	
byte ToByte(byte value)	
// ToChar	
char ToChar(byte value)	转换为 char
char ToChar(short value)	
char ToChar(ushort value)	
char ToChar(int value)	
char ToChar(uint value)	

char ToChar(long value)	
char ToChar(ulong value)	
char ToChar(string value)	
char ToChar(object value)	
char ToChar(sbyte value)	
char ToChar(char value)	
// ToDateTime	
DateTime ToDateTime(DateTime value)	转换为 DateTime
DateTime ToDateTime(string value)	
DateTime ToDateTime(object value)	
// ToDecimal	
decimal ToDecimal(ushort value)	转换为 decimal
decimal ToDecimal(object value)	
decimal ToDecimal(bool value)	
decimal ToDecimal(decimal value)	
decimal ToDecimal(string value)	
decimal ToDecimal(double value)	
decimal ToDecimal(float value)	
decimal ToDecimal(ulong value)	
decimal ToDecimal(sbyte value)	
decimal ToDecimal(byte value)	
decimal ToDecimal(int value)	
decimal ToDecimal(short value)	
decimal ToDecimal(uint value)	
decimal ToDecimal(long value)	
// ToDouble	
double ToDouble(long value)	转换为 double
double ToDouble(bool value)	
double ToDouble(object value)	
double ToDouble(byte value)	
double ToDouble(short value)	
double ToDouble(ushort value)	
double ToDouble(int value)	
double ToDouble(uint value)	
double ToDouble(ulong value)	
double ToDouble(float value)	
double ToDouble(double value)	
double ToDouble(decimal value)	
double ToDouble(string value)	
double ToDouble(sbyte value)	
// ToInt16	
short ToInt16(ushort value)	转换为 short
short ToInt16(decimal value)	
short ToInt16(double value)	
short ToInt16(float value)	

short ToInt16(ulong value)	
short ToInt16(long value)	
short ToInt16(short value)	
short ToInt16(uint value)	
short ToInt16(int value)	
short ToInt16(sbyte value)	
short ToInt16(char value)	
short ToInt16(bool value)	
short ToInt16(object value)	
short ToInt16(string value, int fromBase)	
short ToInt16(byte value)	
short ToInt16(string value)	
// ToInt32	
int ToInt32(int value)	
int ToInt32(string value, int fromBase)	
int ToInt32(long value)	
int ToInt32(ulong value)	
int ToInt32(float value)	
int ToInt32(double value)	
int ToInt32(decimal value)	
int ToInt32(string value)	
int ToInt32(object value)	
int ToInt32(ushort value)	
int ToInt32(short value)	
int ToInt32(byte value)	
int ToInt32(sbyte value)	
int ToInt32(char value)	
int ToInt32(bool value)	
int ToInt32(uint value)	
// ToInt64	
long ToInt64(string value, int fromBase)	
long ToInt64(string value)	
long ToInt64(ushort value)	
long ToInt64(double value)	
long ToInt64(decimal value)	
long ToInt64(object value)	
long ToInt64(bool value)	
long ToInt64(char value)	
long ToInt64(byte value)	
long ToInt64(short value)	
long ToInt64(int value)	
long ToInt64(uint value)	
long ToInt64(ulong value)	
long ToInt64(long value)	
long ToInt64(sbyte value)	

long ToInt64(float value)	
// ToSByte	
sbyte ToSByte(double value)	
sbyte ToSByte(decimal value)	
sbyte ToSByte(long value)	
sbyte ToSByte(uint value)	
sbyte ToSByte(string value, int fromBase)	
sbyte ToSByte(int value)	
sbyte ToSByte(ulong value)	
sbyte ToSByte(ushort value)	
sbyte ToSByte(short value)	
sbyte ToSByte(float value)	
sbyte ToSByte(string value)	
sbyte ToSByte(byte value)	
sbyte ToSByte(object value)	
sbyte ToSByte(bool value)	
sbyte ToSByte(sbyte value)	
sbyte ToSByte(char value)	
// ToSingle	
float ToSingle(ushort value)	
float ToSingle(object value)	
float ToSingle(sbyte value)	
float ToSingle(byte value)	
float ToSingle(int value)	
float ToSingle(ulong value)	
float ToSingle(bool value)	
float ToSingle(string value)	
float ToSingle(decimal value)	
float ToSingle(double value)	
float ToSingle(float value)	
float ToSingle(uint value)	
float ToSingle(long value)	
float ToSingle(short value)	
// ToString	
string ToString(ushort value)	
string ToString(byte value)	
string ToString(short value)	
string ToString(bool value)	
string ToString(char value)	
string ToString(object value)	
string ToString(sbyte value)	
string ToString(long value, int toBase)	
string ToString(int value, int toBase)	
string ToString(short value, int toBase)	
string ToString(byte value, int toBase)	

string ToString(string value)	
string ToString(DateTime value)	
string ToString(int value)	
string ToString(decimal value)	
string ToString(double value)	
string ToString(float value)	
string ToString(ulong value)	
string ToString(long value)	
string ToString(uint value)	
string ToString(BitString value)	
string ToString(BitString value, string format)	按照格式转换 BitString 为字符串。 format 参数：B 表示二进制，O 表示八进制，H 表示十六进制
// ToUInt16	
ushort ToUInt16(string value, int fromBase)	
ushort ToUInt16(byte value)	
ushort ToUInt16(float value)	
ushort ToUInt16(string value)	
ushort ToUInt16(object value)	
ushort ToUInt16(double value)	
ushort ToUInt16(ulong value)	
ushort ToUInt16(long value)	
ushort ToUInt16(decimal value)	
ushort ToUInt16(ushort value)	
ushort ToUInt16(int value)	
ushort ToUInt16(short value)	
ushort ToUInt16(sbyte value)	
ushort ToUInt16(char value)	
ushort ToUInt16(bool value)	
ushort ToUInt16(uint value)	
// ToUInt32	
uint ToUInt32(string value, int fromBase)	
uint ToUInt32(string value)	
uint ToUInt32(decimal value)	
uint ToUInt32(double value)	
uint ToUInt32(float value)	
uint ToUInt32(ulong value)	
uint ToUInt32(uint value)	
uint ToUInt32(long value)	
uint ToUInt32(ushort value)	
uint ToUInt32(short value)	
uint ToUInt32(byte value)	
uint ToUInt32(sbyte value)	
uint ToUInt32(char value)	
uint ToUInt32(bool value)	

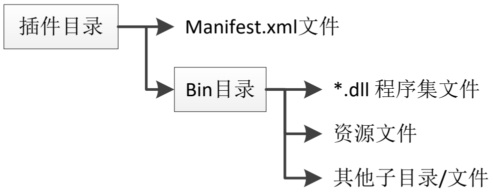
`uint ToUInt32(object value)`	
`uint ToUInt32(int value)`	
`// ToUInt64`	
`ulong ToUInt64(float value)`	
`ulong ToUInt64(decimal value)`	
`ulong ToUInt64(double value)`	
`ulong ToUInt64(uint value)`	
`ulong ToUInt64(string value)`	
`ulong ToUInt64(int value)`	
`ulong ToUInt64(long value)`	
`ulong ToUInt64(short value)`	
`ulong ToUInt64(string value, int fromBase)`	
`ulong ToUInt64(byte value)`	
`ulong ToUInt64(sbyte value)`	
`ulong ToUInt64(char value)`	
`ulong ToUInt64(bool value)`	
`ulong ToUInt64(object value)`	
`ulong ToUInt64(ushort value)`	
`ulong ToUInt64(ulong value)`	

3.6 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。

3.6.1 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：



3.6.2 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>
```

3.6.3 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
- ✧ 扩展新的变量和变量数据分析方法
- ✧ 扩展新的画面控件类型
- ✧ 领域专用模块
- ✧ 企业专用模块

3.7 脚本类库

脚本类库主要描述 SystemContext（系统上下文）、ProjectContext（项目上下文）类以及和它相关的类和接口。脚本可以通过 using 命名空间的方式，直接使用.Net Framework 的类库、WPF 的类库以及通过引用方式引用的自定义 DLL 类库。

脚本类库中接口函数中的参数类型，除了少数软件自定义的类型之外，其他大部分均来自.Net Framework 的类库和 WPF 的类库，如 int、string、ICommand、DataTable、Size、Window 等，使用的时候可以参考微软的 MSDN 帮助或直接咨询我们。

命名空间	类和接口
Genesis	SystemContext、ProjectContext、SystemMode、MessageSeverity、DataType、DataValue、BitString、Endian、IMemento
Genesis.Device	IDeviceSesstion、IMessageDeviceSession、IRegisterDeviceSession
Genesis.Variant	VariantType 、Variant、VariantContainer
Genesis.Sequence	StepType 、IStep、IStepContext、StepResult、StepState、IActionValue、IActionMessage、IActionRegister、IProtocolAlgorithm、IProtocolAlgorithmParameter
Genesis.Schema	Schema
Genesis.Data	IDataManager、IResultDataManager
Genesis.Expressions	ExpressionContext、IGenericExpression、IDynamicExpression
Genesis.Workbench	PerspectiveStyle
Genesis.Windows.Controls 系统内置控件的命名空间	TableGrid、LineChart、PointChart、ImageButton、ImageBox、ImageCheckBox、SpinEditBox、TextEditBox 等
.Net 类库，参考微软 MSDN 帮助系统	Encoding、ICommand、Key、ModifierKeys、MouseAction、DataTable、Size、Window、IEnumerable、IDictionary、Guid、DateTime、Dispatcher、MessageBoxResult、MessageBoxButton、MessageBoxImage、ResizeMode、Window、WindowStyle、Point、FrameworkElement、Canvas

3.7.1 系统上下文 SystemContext

定义	描述
public static SystemMode Mode { get; }	获取软件系统工作模式，即设计时或者运行时。
public static string SystemDirectory { get; }	获取软件系统根目录，即软件安装的根目录。
public static string WorkDirectory{ get; }	获取软件系统工作目录。
项目相关操作	
public static ProjectContext GetProjectContext()	获取当前项目上下文。
public static ProjectContext GetProjectContext(string projectName)	根据项目名称获取对应项目名称的项目上下文。
public static ProjectContext GetProjectContext(IDeviceSession deviceSession)	根据设备会话对象获取对应项目名称的项目上下文。
public static ProjectContext GetProjectContext(Variant variant)	根据变量对象获取对应项目名称的项目上下文。

public static ProjectContext GetProjectContext(IStep step)	根据步骤对象获取对应项目名称的项目上下文。
public static ProjectContext GetProjectContext(Schema schema)	根据画面对象获取对应项目名称的项目上下文。
public static ProjectContext OpenProject(string projectLocation)	根据项目文件路径名打开项目。
public static bool CloseProject(string projectLocation)	根据项目文件路径名关闭项目。
日志记录相关操作	
public static 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 static 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 static 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 static void LogMessage(string location, string format, object arg0, object arg1)	
public static void LogMessage(string location, string format, params object[] args)	
public static void LogMessage(MessageSeverity severity, string location, object value)	MessageSeverity 是 Enum 类型，定义三个值{Info, Warning, Error}。
public static void LogMessage(MessageSeverity severity, string location, string format, object arg0)	
public static void LogMessage(MessageSeverity severity, string location, string format, object arg0, object arg1)	
public static void LogMessage(MessageSeverity severity, string location, string format, params object[] args)	
转换相关操作	
public static string ConvertToBinString(byte number)	将字节转为二进制字符串。
public static string ConvertToBinString(byte number, int bitCount)	
public static string ConvertToHexString(byte b)	将字节转为十六进制字符串。
public static string ConvertToHexString(byte[] data)	
public static string ConvertToHexString(byte[] data, int offset, int len)	
public static string ConvertToHexString(byte[] data, int offset, int len, bool space)	
public static string ConvertToHexString(byte[] bytes, int offset, int len, string prefix, string suffix)	
public static byte[] ConvertToBytes(string hexString)	将十六进制字符串转为字节数组。
public static 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。
public static VariantContainer ConvertToVariantContainer(DataTable dataTable)	将传入 DataTable 变量转换为 VariantContainer 变量，数据内容存在返回变量的历史数据中。 由于 VariantContainer 是 IDataProvider 类型，可作为 IDataProvider 类型数据源使用。
public static VariantContainer ConvertToVariantContainer(DataTable dataTable, VariantContainer container)	将传入 DataTable 变量转换到传入的 container 变量中，并返回 container 变量。
JSON 相关操作	
public static string ConvertToJsonString(object obj)	将对象转换为一个 JSON 字符串。
public static object ConvertToJsonObject(string json, Type type)	将 JSON 字符串转换为指定类型的对象。
public static T ConvertToJsonObject<T>(string json)	将 JSON 字符串转换为指定类型的对象。
其他操作	
public static IDataManager CreateDataManager(DatabaseType dbType, string connectionString)	创建数据管理器。
public static IResultDataManager CreateResultDataManager(string connectionString)	创建执行步骤的结果数据管理器。 参数 connectionString 为 SQLite 数据库连接字符串，例如：Data Source=D:\Result.db。
public static IEnumerable<IPAddress> GetNetworkAddresses()	获取本机的网络地址列表。 返回已运行、支持多播、支持 IPv4、非环回的地址列表。
public static int GetNetworkPort()	获取本机中在[5001,65535]范围内第一个可用的端口号。
public static int GetNetworkPort(int begin, int end = 65535)	获取本机中在[begin,end]范围内第一个可用的端口号。

SystemContext 界面操作

定义	描述
public static Window MainWindow { get; }	获取应用程序主窗口。
public static Dispatcher Dispatcher { get; }	获取应用程序窗口调度器 一般应用调度器的 Invoke 方法在线程中访问窗口控件。
窗体和对话框相关操作	
public static MessageBoxResult ShowMessageBox(string title, string messageBoxText, MessageBoxButton button, MessageBoxImage icon)	弹出消息框。 参数 title 为标题 参数 messageBoxText 为消息内容 参数 button 参考 WPF

	参数 icon 参考 WPF
public static MessageBoxResult ShowMessageBox(string messageBoxText)	弹出消息框。
public static 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 static void ShowWindow(string title, Type contentType, ResizeMode resizeMode)	
public static void ShowWindow(tring title, Type contentType, object contentArgs)	
public static void ShowWindow(string title, Type contentType, object contentArgs, ResizeMode resizeMode)	
public static void ShowWindow(tring title, Type contentType, object[] contentArgs)	
public static void ShowWindow(string title, Type contentType, object[] contentArgs, ResizeMode resizeMode)	
public static void ShowWindow(string title, object content)	显示一个窗体，窗体内容来自 content，可以是用户定义的内容。
public static void ShowWindow(string title, object content, ResizeMode resizeMode)	
public static 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 static bool ShowDialog(string title, Type contentType, ResizeMode resizeMode)	
public static bool ShowDialog(string title, Type contentType, object contentArgs)	
public static bool ShowDialog(string title, Type contentType, object contentArgs, ResizeMode resizeMode)	
public static bool ShowDialog(string title, Type contentType, object[] contentArgs)	
public static bool ShowDialog(string title, Type contentType, object[] contentArgs, ResizeMode resizeMode)	
public static bool ShowDialog(string title, object content)	显示一个对话框窗体，窗体内容来自 content，可以是用户定义的内容。
public static bool ShowDialog(string title, object content, ResizeMode resizeMode)	
public static 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 static string ShowOpenFileDialog(string title, string filter, string fileName)	显示打开文件对话框。 参数 title 同上

	参数 filter 同上 参数 fileName 是默认文件名
public static string ShowSaveFileDialog(string title, string filter)	显示保存文件对话框。 参数定义同上。
public static string ShowSaveFileDialog(string title, string filter, string fileName)	显示保存文件对话框。 参数定义同上。
public static string ShowOpenFolderDialog(string title)	显示打开文件夹对话框。 参数 title 为对话框标题
public static string ShowOpenFolderDialog(string title, string folderName)	显示打开文件夹对话框。 参数 title 同上 参数 folderName 是默认文件夹名
public static 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 static string[] ShowParameterDialog(string title, string[] paramNames, string[] paramDefaults)	显示参数设置对话框,参数个数由参数名称数组 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 static 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。
public static string[] ShowParameterDialog(string title, string[] paramNames, string[] paramDefaults, string[][] parameterOptions, string[] paramTooltips)	paramTooltips 参数决定对应参数的提示信息。
public static void ShowAboutDialog()	显示关于对话框。
视图和编辑器相关操作	
public static 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 static void CloseView(String viewId)	关闭指定 Id 的视图。
public static void CloseAllViews()	关闭所有视图。
public static void CloseEditor(string uri)	关闭 uri 所指示的编辑器。
public static void CloseActiveEditor()	关闭当前激活的编辑器。
public static void CloseAllEditors(string projectLocation)	关闭项目文件路径对应的项目的所有编辑器。
public static void CloseAllEditors()	关闭所有编辑器。
public static void ShowEditorHeaders()	显示编辑器页面的标签头。
public static void HideEditorHeaders()	隐藏编辑器页面的标签头。

public static void ShowFullScreen(bool fullScreen)	显示/关闭全屏状态。
public static void ShowPerspectiveStyle(PerspectiveStyle style)	设置视景窗口的样式。 参数 style 的值定义：Normal 表示停靠面板之间有边框；Compact 表示停靠面板之间无边框。
public static void ResetPerspective()	重置视景窗口到缺省状态。
public static void ShowMainWindow()	显示主窗口。
public static void ShowMainWindow(WindowStyle windowStyle, ResizeMode resizeMode)	显示主窗口。 参数 windowStyle：异形窗口/不规则窗口设置为 WindowStyle.None
public static void ShowMainWindow(Point position, Size size, WindowStyle windowStyle, ResizeMode resizeMode)	显示主窗口。 指定窗口位置、窗口大小、窗口样式和窗口调整大小方式。
public static void HideMainWindow()	隐藏主窗口。
public static void MinimizeMainWindow()	最小化主窗口。
public static void MaximizeMainWindow()	最大化主窗口。
工具栏和状态栏相关操作	
public static void ShowToolBar()	显示工具栏。
public static void HideToolBar()	隐藏工具栏。
public static void MinimizeToolBar()	最小化工具栏。
public static bool IsToolBarMinimized()	获取工具栏当前最小化状态。
public static bool IsToolBarVisible()	获取工具栏当前可视状态。
public static void ShowStatusBar()	显示状态栏。
public static void HideStatusBar()	隐藏状态栏。
public static bool IsStatusBarVisible()	获取状态栏当前可视状态。
键盘和鼠标快捷键相关操作	
public static bool AttachKeyBinding(ICommand command, Key key, ModifierKeys modifiers)	添加全局的键盘快捷键。 参数 command 是实现 ICommand 接口的命令。
public static bool DetachKeyBinding(ICommand command)	删除全局的键盘快捷键。
public static bool AttachMouseBinding(ICommand command, MouseAction mouseAction, ModifierKeys modifiers)	添加全局的鼠标快捷键。
public static bool DetachMouseBinding(ICommand command)	删除全局的鼠标快捷键。
用户相关操作	
public static void Login()	执行登录命令。
public static void Logout()	执行注销命令。
public static string GetUserName()	获取登录用户名。
public static string[] GetUserRoleNames()	获取登录用户拥有的角色名列表。
public static string HasUserRole(string roleName)	获取登录用户是否拥有角色。
public static void LockUser()	锁定登录用户。
public static void ShowUserExplorerDialog(string title)	打开用户管理器对话框。 参数 title 是对话框标题。
系统相关操作	
public static void Shutdown()	执行退出应用程序命令。
public static void Restart()	执行重启应用程序命令。
public static void ShutdownComputer()	执行关闭计算机命令。

public static void RestartComputer()	执行重启计算机命令。

3.7.2 项目上下文 ProjectContext

定义	描述
public string ProjectId { get; }	获取项目 Id 号。
public string ProjectLocation { get; }	获取项目文件完整路径。
public string ProjectDirectory { get; }	获取项目文件所在目录。
public string ProjectFilename{ get; }	获取项目文件名。
public string ProjectName{ get; }	获取项目名称。
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。
public void SetDeviceSessionPrintParameters(string name, bool printHex, long printTimestamp)	设置设备结果数据的打印参数。 name：设备路径名 printHex：true 表示以 16 进制方式打印结果数据，false 表示以字符串方式打印 printTimestamp：打印时间戳参数，小于 0 表示不打印，大于等于 0 表示打印间隔（ms）
public void SetDeviceSessionCompensationTime(int compensationTime)	设置设备会话补偿时间，单位毫秒，默认 100 毫秒，补偿并行分支订阅时各分支的启动时间差。
public void SetDeviceSessionSurvivalTime(int survivalTime)	设置设备会话数据帧生存时间，单位毫秒，默认 10000 毫秒，表示数据帧收到后最长存活时间。
变量相关操作	
public VariantContainer Variants { get; }	获取项目的变量集。
public Variant GetVariant(string name)	根据变量名获取变量。 参数 name：变量的路径名
public string GetVariantPath(Variant variant)	根据变量获取变量路径名。
public bool SaveVariantData(string name, string location)	保存变量的缓存数据，可以用来做数据快照。 参数 name：变量的路径名 参数 location：变量数据文件路径，扩展名为.vdata
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 IStep GetStep(string step)	获取步骤实例。 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 HasStep(string step)	判断步骤是否存在。 参数 step 的定义同上
public bool StartStep(string step)	执行一个步骤。 参数 step 的定义同上
public bool StartStep(string step, int times)	多次执行一个步骤。 参数 step 的定义同上 参数 times：执行次数
public bool StartStep(string step, int times, IDictionary<string, object> datas)	多次执行一个步骤。 参数 step 的定义同上。 参数 times：执行次数 参数 datas：IStepContext 的 Datas 自定义数据集
public bool StartStep(string step, int times, IDictionary<string, object> datas, StepRuntimeMode runtimeMode, StepRuntimeFailurePolicy runtimeFailurePolicy)	多次执行一个步骤。 参数runtimeMode：运行模式 Continuous或Step。 参数 runtimeFailurePolicy：失效策略 Continuous、Pause 或 Stop。
public bool StartSteps(string[] steps)	多个步骤依次执行一次。 参数 steps：步骤 Id 或路径数组，定义同上。
public bool StartSteps(string[] steps, int times)	多个步骤依次执行多次。 参数 steps 的定义同上 参数 times：执行次数 参数 datas：IStepContext 的 Datas 自定义数据集
public bool StartSteps(string[] steps, int times, IDictionary<string, object> datas)	多个步骤依次执行多次。 参数 datas：IStepContext 的 Datas 自定义数据集
public bool StartSteps(string[] steps, int times, IDictionary<string, object> datas, string[] excludedSteps)	多个步骤依次执行多次。 参数 excludedSteps：排除的步骤集，用于排除 steps 顶级执行步骤的某些子步骤。
public bool StartSteps(string[] steps, int times, IDictionary<string, object> datas, string[] excludedSteps, StepRuntimeMode runtimeMode, StepRuntimeFailurePolicy runtimeFailurePolicy)	多个步骤依次执行多次。 参数runtimeMode：同上 参数runtimeFailurePolicy：同上
public bool PauseStep(string step)	暂停步骤执行。
public bool ResumeStep(string step)	恢复步骤执行。
public bool StopStep(string step)	停止步骤执行。
public IStep StartStepScheme(string schemeFileName)	执行步骤方案

	参数 schemeFileName：方案文件名 返回值：成功运行返回父步骤，否则返回 null
`public IStep StartStepScheme(string schemeFileName, StepRuntimeMode runtimeMode, StepRuntimeFailurePolicy runtimeFailurePolicy)`	执行步骤方案
`public IStep StartStepScheme(StepExecutionScheme scheme)`	执行步骤方案 参数 scheme：步骤执行方案对象 返回值：成功运行返回父步骤，否则返回 null
`public IStep StartStepScheme(StepExecutionScheme scheme, StepRuntimeMode runtimeMode, StepRuntimeFailurePolicy runtimeFailurePolicy)`	执行步骤方案
`public bool PauseStepScheme(IStep schemeStep)`	暂停步骤方案
`public bool ResumeStepScheme(IStep schemeStep)`	继续步骤方案
`public bool StopStepScheme(IStep schemeStep)`	停止步骤方案执行。 参数 schemeStep：步骤方案的父步骤
`public bool ExecuteStep(string step)`	执行步骤，阻塞执行线程等待步骤执行完毕。 参数定义同上
`public bool ExecuteStep(string step, int times)`	多次执行步骤，阻塞执行线程等待步骤执行完毕。 参数定义同上
`public bool ExecuteStep(string step, int times, IDictionary<string, object> datas)`	多次执行步骤，阻塞执行线程等待步骤执行完毕。 参数定义同上
`public bool ExecuteSteps(string[] steps, int times, IDictionary<string, object> datas, string[] excludedSteps)`	多个步骤依次执行多次，阻塞执行线程等待步骤执行完毕。 参数定义同上
`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。 参数定义同上
`public async Task<bool> ExecuteStepASync(string step, int times, IDictionary<string, object> datas)`	异步多次执行步骤，并等待步骤执行完毕。 参数定义同上
`public async Task<bool> ExecuteStepsASync(string[] steps, int times, IDictionary<string, object> datas, string[] excludedSteps)`	异步多个步骤依次执行多次，并等待步骤执行完毕。 参数定义同上
`public IStepContext GetStepContext(string step)`	获取步骤上下文。 参数 step 的定义同上。
`public string GetStepPath(IStep step)`	获取以步骤的名称组成的路径。
`public string GetStepResultFileLocation()`	获取步骤记录文件路径。
`public bool NewStepResultFile()`	新建记录文件存储后续执行结果

	返回值：成功返回 true，有步骤在运行时不能新建，返回 false。
画面相关	
public bool HasSchema(string name)	判断画面是否存在于项目中。 参数 name 是画面的完整的路径名称。 例如：画面组 1/画面 A。
public Schema GetSchema(string name)	获取画面对象。 参数 name 是画面的完整路径名。
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。
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。

ProjectContext 界面扩展

定义	描述
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 是画面中的元素，一般为控件事件的 sender 参数； 参数 name 是获取的元素的名称。
public T GetSchemaElement<T> (object element, string name)	获取画面中指定名称的元素。
public void OpenSchema(string name) public void OpenSchemaEditor(string name)	打开指定名称的画面。 参数 name 是画面的完整路径名。
public void CloseSchema(string name) public void CloseSchemaEditor(string name)	关闭指定名称的画面。 参数 name 是画面的完整路径名。
public bool ShowSchemaDialog(string name, string windowTitle, Size windowSize, WindowStyle windowStyle)	打开指定名称的画面对话框。 参数 name 是画面的完整路径名。 参数 windowStyle 参考 WPF 的 WindowStyle 枚举类型。
public bool ShowSchemaDialog(string name, string windowTitle, Size windowSize)	打开指定名称的画面对话框。 参数 name 是画面的完整路径名。
public void ShowSchemaWindow(string name, string windowTitle, Size windowSize, WindowStyle windowStyle)	打开指定名称的画面窗口。 参数 name 是画面的完整路径名。 参数 windowStyle 参考 WPF 的 WindowStyle 枚举类型。
public void ShowSchemaWindow(string name, string windowTitle, Size windowSize)	打开指定名称的画面窗口。 参数 name 是画面的完整路径名。
视图和编辑器相关操作	
public void CloseEditors()	关闭当前项目所有编辑器。
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 OpenVariantEditor(string name)	打开当前项目指定路径的变量编辑器。 参数 name 可以是变量完整路径名 路径格式：name1[/name2]...，以/分隔的字符串。
public void CloseVariantEditor(string name)	关闭当前项目指定路径的变量编辑器。 参数 name 定义同上。
public void OpenVariantDataEditor(string name)	打开当前项目指定路径的变量数据编辑器。 参数 name 可以是变量完整路径名 路径格式：name1[/name2]...，以/分隔的字符串；也可以是变量数据文件的路径（扩展名为.vdata）。

`public void CloseVariantDataEditor(string name)`	关闭当前项目指定路径的变量数据。 参数 name 定义同上。
`public IEnumerable<string> ShowSelectVariantDialog()`	弹出变量选择对话框。
`public IEnumerable<string> ShowSelectVariantDialog(bool allowMultipleSelection)`	弹出变量选择对话框。 参数 allowMultipleSelection：是否允许多选。
`public void OpenDeviceDataEditor(string name)`	打开当前项目指定设备名称的设备数据编辑器。 参数 name 是设备名称 格式：Name1[/Name2]...
`public void CloseDeviceDataEditor(string name)`	关闭当前项目指定设备名称的设备数据编辑器。 参数 name 定义同上。
`public void ShowDeviceDialog(string name)`	弹出设备属性设置对话框。 参数 name 定义同上。
键盘和鼠标快捷键相关操作	
`public bool AttachKeyBinding(object element, ICommand command, Key key, ModifierKeys modifiers)`	添加画面的键盘快捷键。 参数 element 是画面中的元素，一般为控件事件的 sender 参数。 参数 command 是实现 ICommand 接口的命令。
`public bool DetachKeyBinding(object element, ICommand command)`	删除画面的键盘快捷键。
`public bool AttachMouseBinding(object element, ICommand command, MouseAction mouseAction, ModifierKeys modifiers)`	添加画面的鼠标快捷键。
`public bool DetachMouseBinding(object element, ICommand command)`	删除画面的鼠标快捷键。
`public void ClearInputBindings(object element)`	清除画面所有绑定的快捷键。

3.7.3 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][,...]
`Encoding Encoding { get; set; }`	字符编码方式，用于字符型数值的读写。
`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。

3.7.4 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。
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。

3.7.5 寄存器型设备会话 IRegisterDeviceSession

定义	描述
IEnumerable<T> Read<T>(string register)	读单个寄存器的值，返回的值数量由设备驱动配置。 参数 register：寄存器描述符（注 1）
IEnumerable<object>[] Read(string[] registers)	读 n 个寄存器，返回的数列索引和地址一一对应。
Task<IEnumerable<T>> ReadAsync<T>(string register)	异步读单个寄存器的值。
Task<IEnumerable<object>[]> ReadAsync(string[] registers)	异步读 n 个寄存器。
bool Write<T>(string register, T value)	写单个寄存器单个值。 参数 register：同上 参数 value：写入的数值
bool Write<T>(string register, IEnumerable<T> values)	写单个寄存器多个值。 参数 register：同上

	参数 values：写入的数值列表
bool Write(string[] registers, IEnumerable<object>[] values)	写 n 个寄存器，每个寄存器写入一列数，写入的数列索引和地址一一对应。
Task WriteAsync<T>(string register, T value)	异步写单个寄存器单个值。
Task WriteAsync<T>(string register, IEnumerable<T> values)	异步写单个寄存器多个值。
Task WriteAsync(string[] registers, IEnumerable<object>[] values)	异步写 n 个寄存器，每个寄存器写入一列数，写入的数列索引和地址一一对应。

注 1：寄存器描述符的格式可以由设备驱动自主定义，分隔符有分号（;）、冒号（:）和逗号（,）三种。分号为第一层次分隔符，用于分隔描述符段，例如用于分隔地址描述符和数据类型描述符；冒号为第二层次分隔符，用于分隔段的类型和参数，例如数据描述段的数据类型和类型参数；逗号用于分隔参数。一般格式为<地址类型>:<地址>,<位地址>;<数据类型>:<数据位长度>,<数据字节序>
其中：<地址类型>:<地址>,<位地址>由设备驱动定义；<数据类型>为 DataType 支持的类型，可以是 Boolean、Sbyte、Byte、Int16、UInt16、Int32、Uint32、Int64、Uint64、Float、Double、Decimal、DateTime、String、BitString；<数据字节序>为 B/BigEndian（大端）或 L/LittleEndian（小端）。

3.7.6 变量 Variant

定义	描述
string Name { get; set; }	变量名称。
VariantType Type { get; }	变量类型，可以是 Value（值类型）、Array（数组类型）或者 Container（容器类型）。
DataType ValueType { get; set; }	变量值的 DataType 类型，可以是 Boolean、Sbyte、Byte、Int16、UInt16、Int32、Uint32、Int64、Uint64、Float、Double、Decimal、DateTime、String、BitString。
object Value { get; set; }	变量值。
string Unit { get; set; }	变量单位。
string Format { get; set; }	变量值的格式字符串。
string Description { get; set; }	变量描述。
IDataProvider 相关操作（用于处理 Variant 的缓存数据）	
DataValue DataSchema { get; }	获取数据的架构。
DataValue[] Data { get; }	获取缓存数据。
int Quantity { get; }	缓存数据总数量。
int Capacity { get; set; }	缓存数据存储的容量，超过后把最旧的删除，大于 0 才存储数据。
IEnumerable<DataValue> Take(DataValue start, int count)	从 start 开始（不包括 start）获取 count 个记录，如果 start 为 null，则从第一个记录开始获取。
IEnumerable<DataValue> Take(int startIndex, int count)	从 startIndex 开始（不包括 startIndex）获取 count 个记录，如果 startIndex 为-1，则从第一

	个记录开始获取。
int Remove(int startIndex, int count)	删除指定范围的缓存数据。
void Reset()	复位变量并清除缓存数据。
运算符重载相关操作	
static bool operator ==(Variant a, <Type> b)	重载==和!= ，其中<Type>可以是 bool、sbyte、byte、short、ushort、int、uint、long、ulong、float、double、decimal、string、BitString
static bool operator !=(Variant a, <Type> b)	
static bool operator ==(<Type> a, Variant b)	
static bool operator !=(<Type> a, Variant b)	
static bool operator <(Variant a, Variant b)	重载小于<
static bool operator >(Variant a, Variant b)	重载大于>
static bool operator <=(Variant a, Variant b)	重载小于等于<=
static bool operator >=(Variant a, Variant b)	重载大于等于>=
static Variant operator +(Variant a, Variant b)	重载+
static Variant operator -(Variant a, Variant b)	重载-
static Variant operator *(Variant a, Variant b)	重载*
static Variant operator /(Variant a, Variant b)	重载/
静态操作	
static bool IsNull(Variant variant)	判断变量是否为 null 操作。 注意：由于重载了操作符，不能使用 variant==null 来判断，要使用 Variant.IsNull(variant)判断。
static void SetNull(ref Variant variant)	

3.7.7 变量容器 VariantContainer

变量容器 VariantContainer 继承自 Variant。

定义	描述
Variant this[int index] { get; set; }	Get 方法返回容器中 index 索引的变量；Set 方法设置容器中 index 索引的变量的 Value 属性值。
Variant this[string name] { get; set; }	Get 方法返回容器中名称路径 name 的变量；Set 方法设置容器中名称路径 name 的变量的 Value 属性值。注：name 可以是变量名称路径。
IEnumerable<Variant> Items { get; }	容器包含的变量集。
Variant Find(string path)	使用变量名称访问路径获取变量，规则：用/分隔变量名，如"A/B/C"。
Variant FindByNamePath(string path)	同上。
Variant FindByIndexPath(string path)	使用变量索引访问路径获取变量，规则：用/分隔索引号，如"2/0/1"。
bool VerifyName(string name)	验证一个变量名是否合法，一个容器内，变量名不为空不重复。
bool Add(Variant item)	增加一个变量。
int Insert(int index, Variant item)	插入一个变量。
bool Remove(Variant item)	删除一个变量。
void Clear()	清除所有子变量。

3.7.8 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。
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。

3.7.9 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 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

3.7.10 步骤 IActionValue

值类型动作步骤的接口。

定义	描述
Genesis.Variant.Variant Value { get; set; }	见 3.3.2 动作类 - Value 步骤

3.7.11 步骤 IActionMessage

消息型动作步骤的接口。

定义	描述
MessageMatchMode MatchMode { get; }	见 3.3.2 动作类 - Message 步骤
MessageOperationMode OperationMode { get; }	见 3.3.2 动作类 - Message 步骤
int ByteInterval { get; set; }	见 3.3.2 动作类 - Message 步骤
int MaxLength { get; set; }	见 3.3.2 动作类 - Message 步骤
string EscapeTable { get; set; }	见 3.3.2 动作类 - Message 步骤
string EscapeParameters { get; set; }	见 3.3.2 动作类 - Message 步骤
string GetMessageFieldValue(int index)	获取指定索引号的协议字段值。 参数 index 从 0 开始计算。
string GetMessageFieldValue(string path)	根据索引路径获取协议字段值，用于协议字段有嵌套的情况。 层次间用/分隔索引号，索引号从 0 开始计算，如"2/0/1"。
void SetMessageFieldValue(int index, object value)	设置指定索引号的协议字段值。用于在运行时，用脚本动态改变协议数据的值。
void SetMessageFieldValue(string path, object value)	根据索引路径设置协议字段值，用于协议字段有嵌套的情况。用于在运行时，用脚本动态改变协议数据的值。

3.7.12 步骤 IActionRegister

寄存器型动作步骤的接口。

定义	描述
RegisterOperationMode OperationMode { get; }	见 3.3.2 动作类 - Register 步骤
string GetRegisterFieldValue(int index)	获取指定索引号的寄存器字段值。 参数 index 从 0 开始计算。
void SetRegisterFieldValue(int index, object value)	设置指定索引号的寄存器字段值。用于在运行时，用脚本动态改变协议数据的值。

3.7.13 步骤状态 StepState

定义	描述
Init = 0	步骤初始化完成
Running = 1	步骤执行中
Done = 2	步骤执行完成
Terminated = 3	步骤已经终止

3.7.14 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 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。

3.7.15 步骤结果状态 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

3.7.16 协议计算型字段算法接口 IProtocolAlgorithm

定义	描述
string Id { get; }	算法标识，在整个软件系统中必须唯一。
string Name { get; }	算法名称，在算法列表中显示。
string Description { get; }	算法描述，暂不用。
IEnumerable<IProtocolAlgorithmParameter> Parameters { get; }	算法的参数，不同算法参数可能不同。
BitString Compute(BitString data, int fieldStartBit, int fieldEndBit, DataType fieldType, Endian fieldEndian, out int dataStartBit, out int dataEndBit);	计算协议字段数值 参数 data：原始数据位串 参数 fieldStartBit：协议字段在数据位串的起始位置 参数 fieldEndBit：协议字段在数据位串的结束位置 参数 fieldType：协议字段的数据类型 参数 fieldEndian：协议字段的大小端 参数 dataStartBit：返回待计算数据在位串的起始位置 参数 dataEndBit：返回待计算数据在位串的结束位置 返回：结果位串，大小端由 fieldEndian 决定，长度为 fieldEndBit-fieldStartBit+1

3.7.17 协议计算型字段算法参数接口 IProtocolAlgorithmParameter

定义	描述
string Name { get; }	算法参数名称，系统通过名称对参数进行赋值。
Type ValueType { get; }	算法参数的类型。
string Value { get; set; }	算法参数值。
string Description { get; }	算法参数描述，一般用作 Tooltip 之类的帮助文本。

3.7.18 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

3.7.19 数据库管理器 IDataManager

定义	描述
通用泛型增删改查	
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>> deleteWhere) 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>> selectWhere) 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[] orderFields) where T : class	获取所有实体集合(多个排序)
IQueryable<T> Sort<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> Sort<T>(IQueryable<T> data, params DataOrder[] orderFields) where T : class	多个排序通用方法
List<T> GetList<T>(Expression<Func<T, bool>> selectWhere) where T : class	根据条件查询实体集合，查询条件 selectWhere 是 lambel 表达式
List<T> GetList<T, TValue>(Expression<Func<T, TValue>> selectWhere, IEnumerable<TValue> conditions) where T : class	根据条件查询实体集合，查询条件 selectWhere 是 lambel 表达式
List<T> GetList<T, Tkey>(Expression<Func<T, bool>> selectWhere, Expression<Func<T, Tkey>> orderWhere, bool isDesc) where T : class	根据条件查询实体集合，查询条件 selectWhere 是 lambel 表达式
List<T> GetList<T>(Expression<Func<T, bool>> selectWhere, params DataOrder[] orderFields) where T : class	根据条件查询实体集合(多个字段排序)，查询条件 selectWhere 是 lambel 表达式
List<T> GetListPaged<T, Tkey>(int pageIndex, int	获取分页集合(无条件默认 ID 升序排序)

pageSize, out int totalcount) where T : class	
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, DataOrder[] orderFields, out int totalcount) 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>> selectWhere, out int totalcount) where T : class	获取分页集合(有条件 ID 升序排序)
List<T> GetListPaged<T, Tkey>(int pageIndex, int pageSize, Expression<Func<T, bool>> selectWhere, 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>> selectWhere, DataOrder[] orderFields, out int totalcount) 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, List<Object[]> lsParas)	通过事务执行批量操作。
通用非泛型操作	
String[] GetTableNames()	获取数据库所有表的表名。
DataColumn[] GetTableColumns(string tableName)	获取数据库指定表的所有列信息。
int CreateTable(string tableName, string[] fieldNames, Type[] fieldTypes)	创建数据表。
int CreateTable(string tableName, string[] fieldNames, Type[] fieldTypes, string[] indices)	创建数据表和数据表索引。 其中 indices 数组每一项创建一个索引，多列索引用逗号分隔列名，所创建的索引名称为：<列名>Index。
int ClearTable(string tableName)	清空指定的数据表。
int GetListCount(string tableName)	获取指定的数据表的总行数。
DataTable GetList(string tableName, int index, int count, params DataOrder[] dataOrders)	获取指定的数据表指定行范围的数据。
int AddList(string tableName, string[] fieldNames, List<Object[]> dataRows)	添加数据行到指定的数据表。