

---

上海格西信息科技有限公司

---

格西调试精灵  
用户手册

版本 3.3

# 目录

1. 简介	4
1.1 关于	4
1.2 功能特性	4
1.3 系统要求	5
1.4 产品版本和许可	5
1.5 产品支持	5
2. 软件入门	6
2.1 启动软件	6
2.2 用户界面	6
2.2.1 主界面	6
2.2.2 应用程序菜单	6
2.2.3 工具栏	8
2.2.4 状态栏	10
2.2.5 控制台	10
2.2.6 数据区	12
2.3 软件设置	13
2.3.1 常规设置	13
2.3.2 显示设置	14
2.3.3 激励工程设置	15
2.3.4 插件信息	15
2.4 软件激活	16
2.4.1 加密锁授权	16
2.4.2 离线授权	16
2.4.3 在线授权	17
3. 直接激励项目	18
3.1 简介	18
3.2 基本操作	18
3.2.1 新建直接激励项目	18
3.2.2 打开直接激励项目	18
3.2.3 关闭直接激励项目	19
3.2.4 保存直接激励项目	19
3.2.5 修改直接激励项目属性	19
3.2.6 添加直接激励项	19
3.2.7 删除直接激励项	19
3.2.8 激励	20
3.2.9 停止激励	20
3.3 用途	20
4. 协议激励项目	21
4.1 简介	21
4.2 基本操作	21
4.2.1 新建协议激励项目	21
4.2.2 打开协议激励项目	21
4.2.3 关闭协议激励项目	21

---

4.2.4	保存协议激励项目	21
4.2.5	修改协议激励项目属性	21
4.2.6	添加协议集	22
4.2.7	添加协议项	23
4.2.8	激励	24
4.2.9	停止激励	24
4.3	协议帧	25
4.4	协议变量	27
4.5	协议脚本	29
4.5.1	脚本界面	29
4.5.2	脚本结构	30
4.5.3	脚本参数 BSCaseContext 类	31
4.5.4	串口参数 BSComStreamParameters 类	46
4.5.5	网口参数 BSNetStreamParameters 类	51
4.5.6	脚本中使用插件	51
5.	插件	53
5.1	托管代码与非托管代码	53
5.2	编写插件	53
5.3	使用托管代码的第三方库	53
5.4	使用非托管代码的第三方库	53
6.	工具箱	54
6.1	校验和计算器	54
6.2	CRC 计算器	54
6.3	DES 计算器	55
6.4	哈希值计算器	56
6.5	ASCII 码对照表	57
6.6	字符编码转换器	58
7.	应用技巧	59
7.1	分类组织协议激励项目的协议项	59
7.2	运行多个软件实例	59
8.	FAQ	60
8.1	进行“反馈”或者“注册软件”操作时，为什么出现 Unknown error (0x80041002) 错误？	60
8.2	进行协议激励时，从动方已经发出正确的帧，为什么主动激励方却返回失败？	60

# 用户手册

## 1. 简介

### 1.1 关于

格西调试精灵是一款强大的通信协议调试和测试软件，主要解决电子研发过程中调试和测试软件定制化过多的问题，能够快速定制任意通信协议，使得通信软件能够应付快速多变的通信测试环境，可以让企业产品开发的调试和测试工具统一化、标准化，帮助企业降低测试工具的开发成本、学习成本和维护成本！

格西调试精灵的优势：

- 快捷的测试激励定制 - 入门级的技能需求，专业级的工作成果
- 直观的测试数据呈现 - 强大的数据分类、存储、统计和显示功能
- 灵活的测试流程控制 - 支持循环测试和组合测试、支持主动设备和从动设备模拟、支持单工和双工工作模式、支持直接激励和协议激励单独测试或者混合测试

格西调试精灵主要适用于：

- 电子产品研发、测试和生产企业
- 电子产品研发、测试人员

### 1.2 功能特性

#### 1) 基本功能

- 1.1) 支持以文本或 16 进制方式接收和显示数据；
- 1.2) 支持串口，自动寻找系统支持的串口，150~256000 常见的波特率，支持自定义任意波特率；
- 1.3) 支持网络接口，支持 UDP、TCP 客户端、TCP 服务器协议类型；
- 1.4) 支持自动保存测试数据。

#### 2) 直接激励功能

- 2.1) 支持 16 进制、字符串、文件 3 种格式的数据发送；
- 2.2) 支持循环激励；
- 2.3) 支持保存为文件，方便测试项目的统一管理和重复使用。

#### 3) 协议激励功能

- 3.1) 支持主动设备（即主动发起请求、接收响应的设备）和从动设备（即被动等待请求、发送响应的设备）的激励仿真；
- 3.2) 支持任意的帧格式定制，最小解析单位为 1Bit，可直观地显示任意帧格式，不需要再为不同的协议定制软件；
- 3.3) 支持自定义协议类型，支持测试结果分类存储和显示；
- 3.4) 支持使用 C#、VB 和 JScript 脚本语言控制测试运行，调用第三方 DLL 完成复杂的计算和测试任务，具有强大的扩展性；
- 3.5) 支持循环激励和激励统计报告；
- 3.6) 支持保存为工程文件，方便测试项目的统一管理和重复使用。

#### 4) 支持常用的计算工具

- 4.1) 支持 CS、BCC、LRC 校验和计算器；

- 4.2) 支持 DES/3DES 计算器，支持 64 位、128 位和 196 位密钥；
- 4.3) 支持 CRC8/CRC16/CRC32 计算器，支持各项参数自定义，支持 16 进制、字符串和文件 3 种数据计算，附带常用 CRC 标准算法表供选择使用；
- 4.4) 支持 MD5/SHA1/SHA256/SHA384/SHA512 哈希值计算器，支持 16 进制、字符串和文件 3 种数据计算。
- 4.5) 支持 ASCII 码对照表。
- 4.6) 支持编码（ASCII、UTF8、UNICODE 等）和 Hex 互转转换器。

### 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： 300 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两种，专业版和免费版。其中专业版采用离线授权和在线授权两种方式进行许可，免费版不需要许可。

#### 专业版功能：

- 支持基本功能；
- 支持直接激励功能；
- 支持协议激励功能。

#### 免费版功能：

- 支持基本功能；
- 支持直接激励功能。

#### 专业版许可细则：

- 离线授权和电脑绑定，一经授权，不能解绑，请谨慎！
- 授权的电脑可以同时运行本产品多个实例；
- 在线授权需要联网使用，不绑定电脑，同一时间只能一台电脑使用。

### 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。

- 支持网站：[www.geshe.com](http://www.geshe.com)
- 电子邮件：[support@geshe.com](mailto:support@geshe.com)
- QQ：979464

## 2. 软件入门

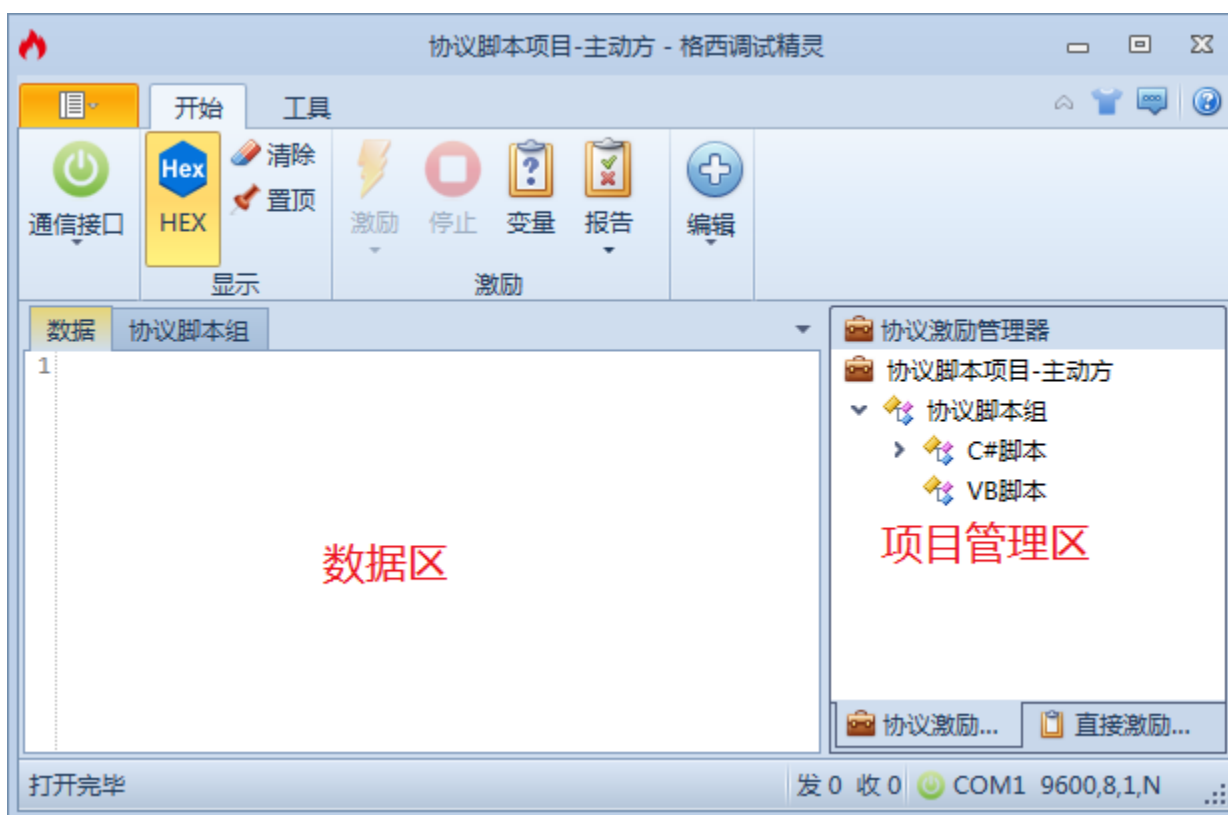
### 2.1 启动软件

本软件安装成功之后，会在 Windows 的【开始菜单】中创建菜单项，同时关联直接激励项目文件（.bsp）和协议激励项目文件（.bcp）。启动软件有两种方法：

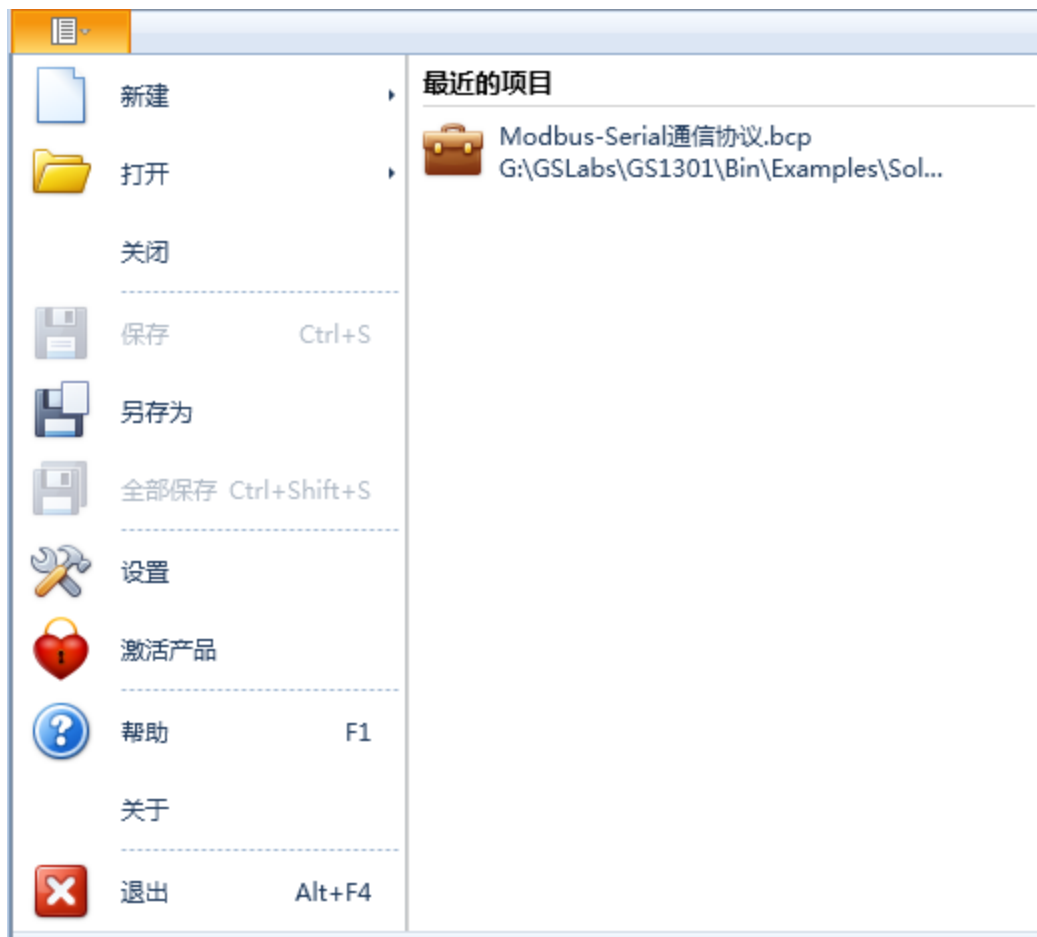
- 方法 1：Windows【开始菜单】->“程序”->“格西调试精灵”->“格西调试精灵”。
- 方法 2：鼠标双击打开直接激励项目文件（.bsp）或者协议激励项目文件（.bcp）。

### 2.2 用户界面

#### 2.2.1 主界面



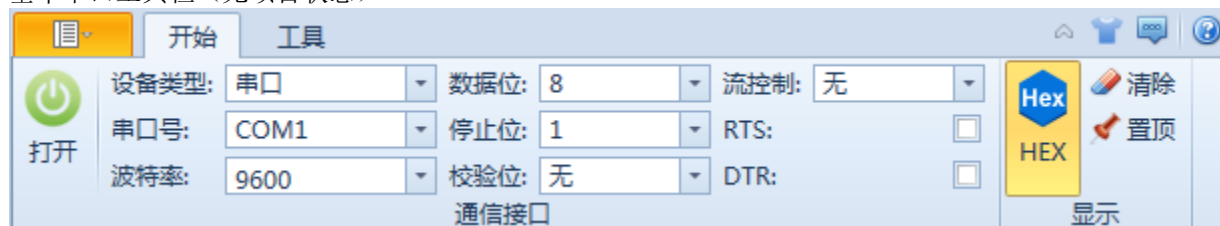
#### 2.2.2 应用程序菜单



命令	快捷键	功能
新建-->协议激励项目	Ctrl+N	新建一个协议激励项目，并建立项目环境。
新建-->直接激励项目	Ctrl+Shift+N	新建一个直接激励项目，并建立项目环境。
打开-->协议激励项目	Ctrl+O	打开一个协议激励项目，并建立项目环境。
打开-->直接激励项目	Ctrl+Shift+O	打开一个直接激励项目，并建立项目环境。
关闭		关闭当前激活的项目。
保存	Ctrl+S	保存当前激活的项目。
另存为		将当前激活的项目保存到指定路径。
全部保存	Ctrl+Shift+S	保存当前打开的所有项目。
设置		设置软件的运行参数。
激活产品		激活软件，获得专业版功能和服务。
帮助	F1	软件的使用帮助。
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。

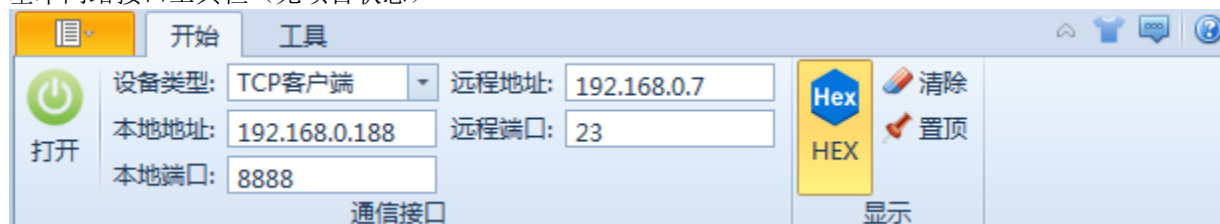
## 2.2.3 工具栏

基本串口工具栏（无项目状态）



命令	功能
打开/关闭	打开或者关闭串口
串口号	设置串口号，自动检测当前系统有效的串口设备。
波特率	设置波特率，提供标准波特率，支持自定义波特率输入。
数据位	设置数据位，支持 5、6、7、8。
停止位	设置停止位，支持 1、1.5、2。
校验位	设置校验位，支持无、奇校验、偶校验、置 1、置 0。
流控制（握手协议）	设置流控制，支持无、RequestToSend、XonXoff、RequestToSend/XonXoff。
RTS	设置在串行通信中是否启用请求发送（RTS）信号
DTR	设置在串行通信中是否启用数据终端就绪（DTR）信号。
HEX	设置数据面板数据显示格式，选中表示解析为 HEX 字符串，不选表示直接解析为字符串。
清除	清除所有数据区的显示数据，不影响已经保存的数据。
置顶	设置主窗口保持在最前面。
折叠工具栏（右上角第 1 个按钮）	显示/折叠工具栏。
反馈（右上角第 2 个按钮）	显示用户反馈对话框。
帮助（右上角第 3 个按钮）	显示用户帮助。

基本网络接口工具栏（无项目状态）



命令	功能
打开/关闭	打开或者关闭网络接口



协议类型	设置协议类型，支持 UDP、TCP 客户端和 TCP 服务器。
本地地址	设置本地地址，默认值 0.0.0.0。
本地端口	设置本地端口，作 UDP、TCP 服务器时必须设置，作 TCP 客户端时 0 表示系统默认分配。
远程地址	设置远程地址，作 UDP、TCP 客户端时必须设置，作 TCP 服务器时不能设置。
远程端口	设置远程端口，作 UDP、TCP 客户端时必须设置，作 TCP 服务器时不能设置。

协议激励工具栏（通信接口已折叠）



命令	功能
激励	单次激励选中的协议项或者协议集，下拉菜单提供循环激励功能。
停止	停止运行协议激励。
报告	下拉菜单提供显示/隐藏协议激励报告，导出协议报告。
添加	下拉菜单提供在当前选定节点的添加协议项或者协议集功能。
保存	保存协议激励项目。
剪切	剪切当前选定节点。
复制	复制当前选定节点。
粘贴	在当前选定节点粘贴。
删除	删除当前选定节点。
属性	显示当前选定节点的属性对话框。

直接激励工具栏（通信接口已折叠）



命令	功能
激励	单次激励直接激励项目所有激活的激励项，下拉菜单提供循环激励功能。

停止	停止运行直接激励。
添加	下拉菜单提供在当前选定节点的添加项功能。
保存	保存直接激励项目。
删除	删除当前选定的激励项。
属性	显示直接激励项目的属性对话框。

常用工具工具栏



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

2.2.4 状态栏

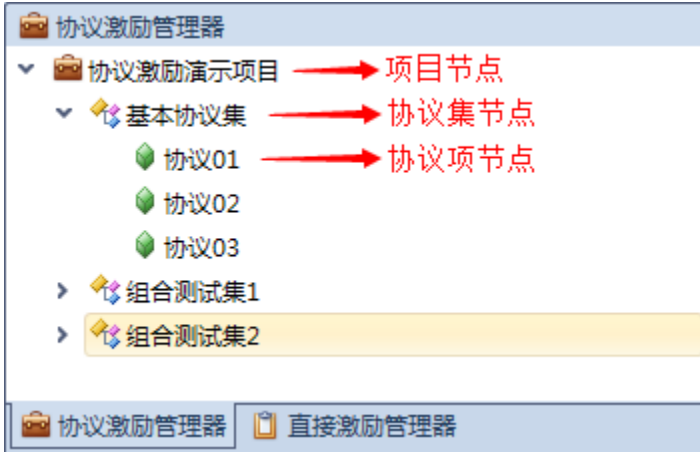


状态栏分为三部分，分别为操作状态信息、通信传输字节信息、通信接口状态信息。

状态栏	功能
操作状态信息	显示上一次操作的状态信息，有错误、警告、信息三种状态。
通信传输字节信息	显示自打开通信接口以来总共接收/发送的字节数。
通信接口状态信息	显示当前通信接口状态。

2.2.5 控制台

协议激励项目控制台，采用树形结构进行组织协议项。



协议激励项目控制台右键快捷菜单。



直接激励项目控制台，采用列表结构组织激励项。支持 16 进制、字符串和文件三种数据格式，支持激励项延时，支持激励项使能，支持立即执行激励项（左边激励按钮，忽略激活属性）。

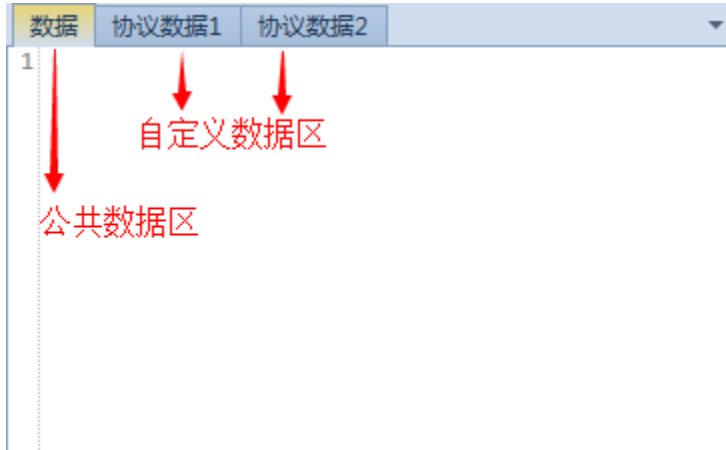
直接激励管理器						
	名称	数据格式	数据	延时(ms)	描述	
	<input checked="" type="checkbox"/> 一串HEX	十六进制	11223344AABB	50		
	<input type="checkbox"/> 一串字符	字符串	格西调试精灵	50		
	<input type="checkbox"/> 一个文件	文件	C:\Test.txt	50		
	<input type="checkbox"/> 发送\	字符串	Send \\	50		
	<input type="checkbox"/> 发送\r\n	字符串	Send \r\n	50		
	<input type="checkbox"/> 发送HEX	字符串	Send \x0d\x0a	50		

直接激励项目控制台右键快捷菜单。



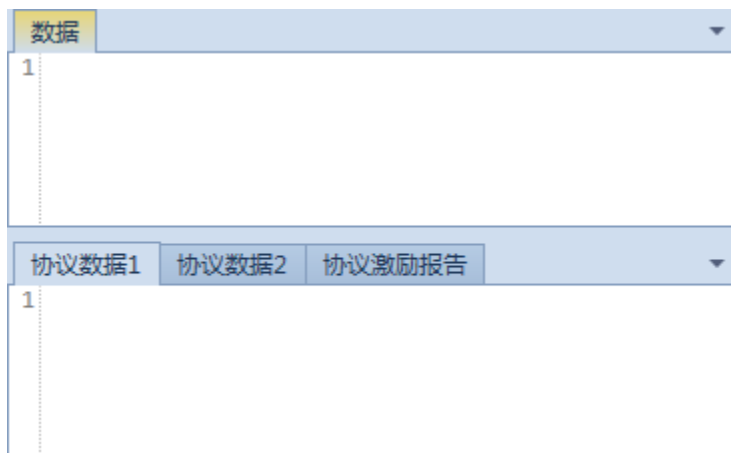
### 2.2.6 数据区

数据区是显示激励数据的区域，数据分为公共数据、自定义数据和报告数据。



数据类型	功能
公共数据	显示通信传输的实际收发数据，其中发送数据可以通过设置屏蔽，协议激励如果没有自定义数据，默认显示在公共数据区。
自定义数据	只有协议激励项目能够自定义数据类型，协议项在激励时根据所属的协议类型把结果输出到对应的自定义数据区中。
报告数据	只有协议激励项目有运行报告，协议激励报告作为独立的页面显示在数据区。

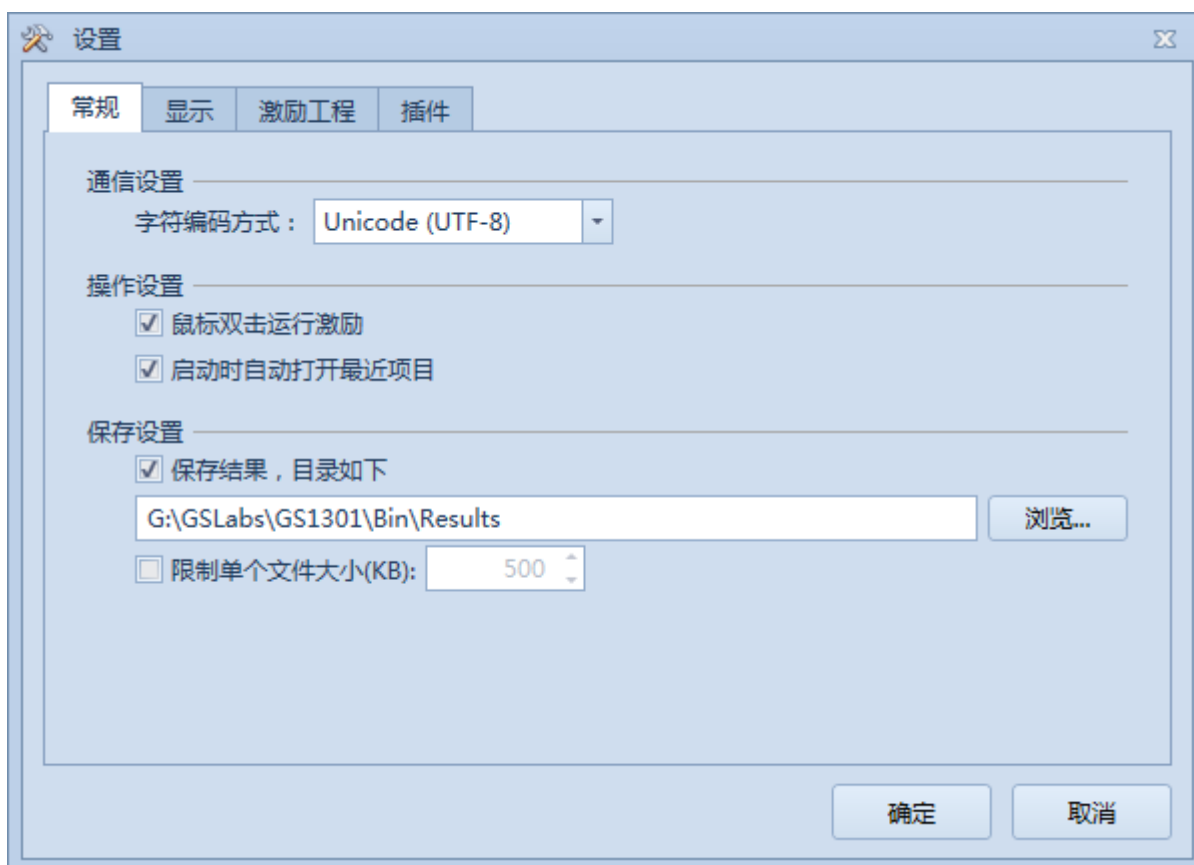
各个数据区可以按需布局，灵活排布，方便数据观测。



## 2.3 软件设置

操作：【应用程序菜单】->“设置”。

### 2.3.1 常规设置



参数	功能
字符编码方式	设置软件所有字符串所采用的编码方式，例如 UTF8、GB2312 等。
鼠标双击运行激励	设置控制台的激励项是否在打开通信接口后可以通过鼠标双击来运行激

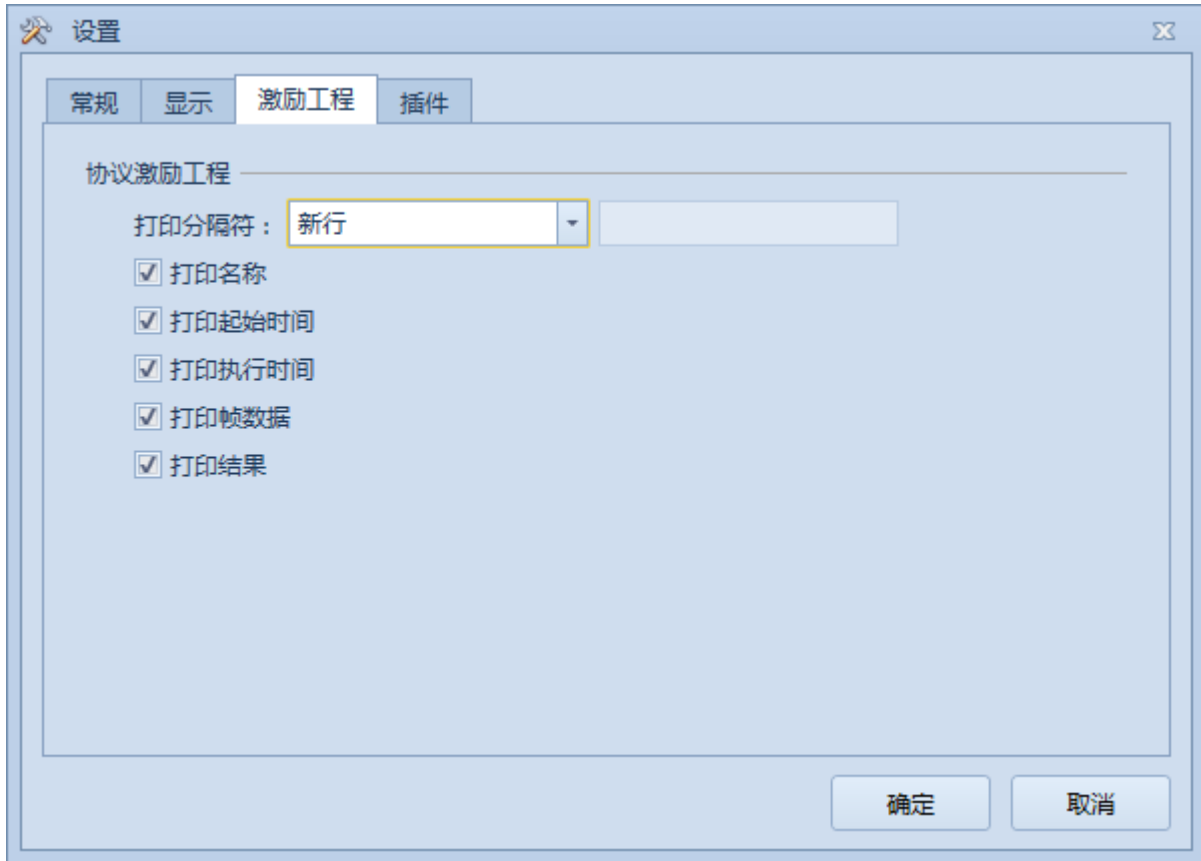
	励。
启动时自动打开最近项目	设置软件启动时是否自动打开最近一次打开的激励项目。
保存结果	使能保存功能，并设置保存结果的目录。

### 2.3.2 显示设置



参数	功能
语言类型	设置用户界面的语言。
显示最大长度	数据显示页面容纳的最大字符数，超过则删除最早的；该选项不影响保存到硬盘的结果数据。
最小化到系统托盘	使能后主窗体在最小化时隐藏到系统托盘。
打印输出数据	设置数据区“数据”面板中是否显示发送的数据。
打印数据时间戳	设置数据区“数据”面板中是否显示数据的时间戳，使能后设置间隔为0则每次收发都打印时间戳，设置间隔>0则每隔一定时间打印一次。
打印数据自动滚动到最后	设置数据区“数据”面板中数据是否自动滚动到最后一行。

### 2.3.3 激励工程设置

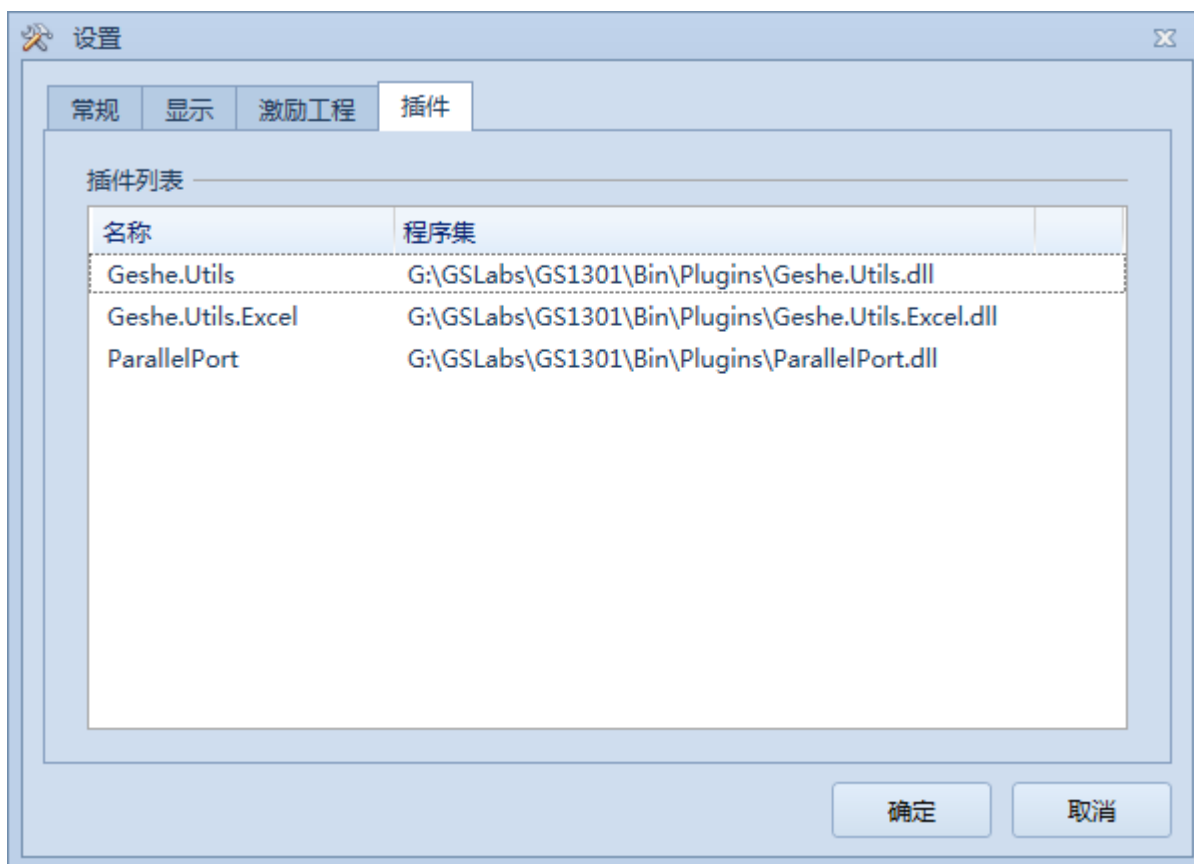


协议激励结果数据包含名称、时间（起始时间）、请求帧、响应帧、耗时（执行时间）和结果六个数据段。

协议激励工程参数	功能
打印分隔符	设置激励结果数据各个数据段之间的分隔符，支持“新行”和“空格”两种分隔方式。
打印起始时间	设置激励结果数据是否显示“时间”数据段。
打印执行时间	设置激励结果数据是否显示“耗时”数据段。
打印帧数据	设置激励结果数据是否显示“请求帧”和“响应帧”的帧数据。

### 2.3.4 插件信息

列出位于本软件安装目录的 Plugins 子目录下的可以被本软件识别的基于 Microsoft .NET Framework 的托管代码组件。



## 2.4 软件激活

软件激活是指专业版的授权，免费版无需授权。

软件专业版采用加密锁授权、离线授权和在线授权三种方式进行许可。

### 2.4.1 加密锁授权

通过以下步骤进行加密锁授权：

步骤 1：从官方网站下载【加密锁驱动】，并安装到计算机。

下载主页链接：<http://www.geshe.com/zh-cn/support/download>

步骤 2：将加密锁插入到计算机。

### 2.4.2 离线授权

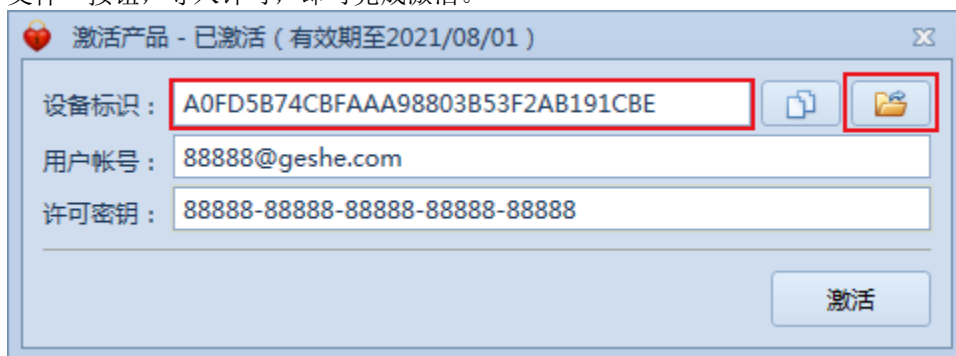
进行离线授权的计算机不需要上网，通过以下步骤进行离线授权：

步骤 1：【应用程序菜单】-> “激活产品”。





步骤 2: 在弹出的对话框中, 把设备标识通过购买渠道发给本公司, 本公司根据该设备标识生成一个许可文件, 并通过 Email 发送给购买用户。用户收到许可文件后, 通过“激活产品”对话框右上角的“导入许可文件”按钮, 导入许可, 即可完成激活。



#### 2.4.3 在线授权

进行在线授权的计算机**必须能够上网**, 通过以下步骤进行在线授权:

步骤 1: 【应用程序菜单】-> “激活产品”。

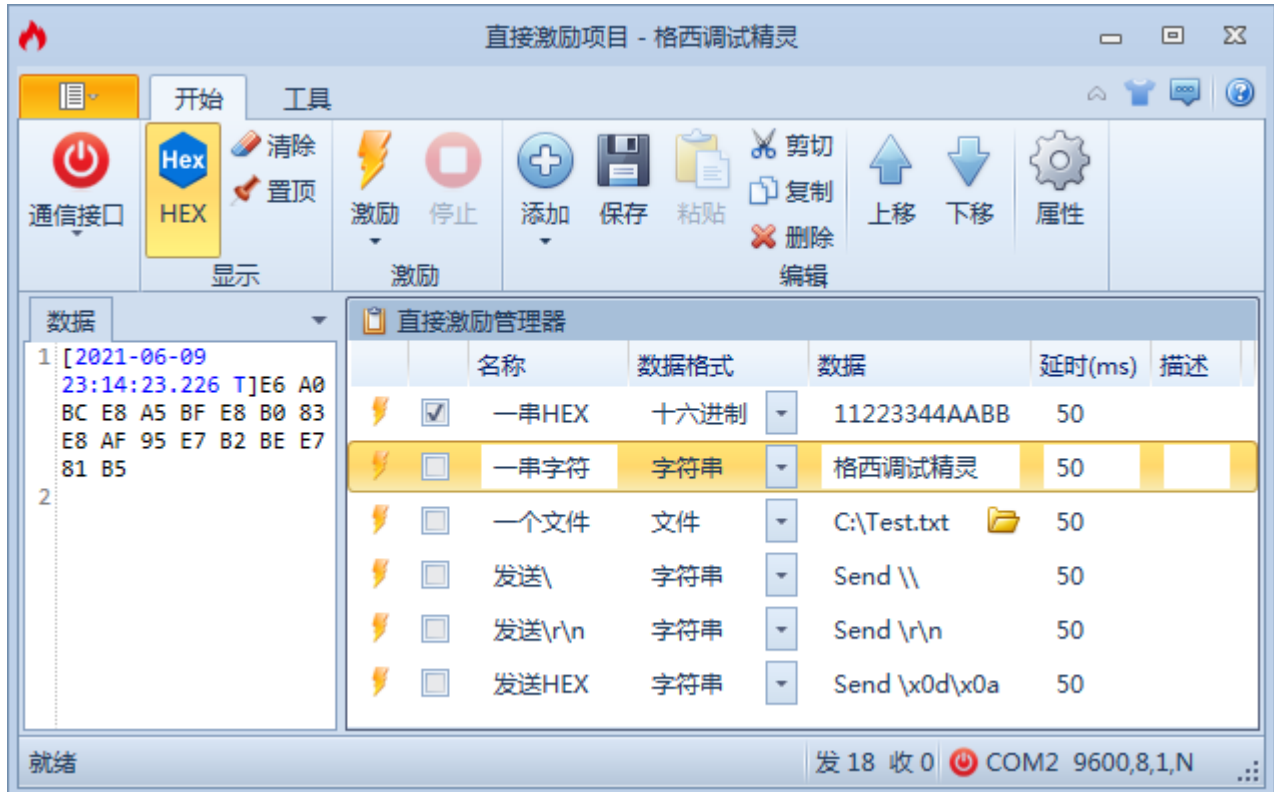
步骤 2: 在弹出的对话框中输入购买的用户帐号和许可密钥, 点击“激活”。



### 3. 直接激励项目

#### 3.1 简介

直接激励项目提供了发送数据的功能，采用列表结构组织激励项。支持 16 进制、字符串和文件三种数据格式，支持激励项延时，支持激励项使能，支持立即执行激励项（左边激励按钮，忽略激活属性）。



#### 3.2 基本操作

##### 3.2.1 新建直接激励项目

步骤 1: 【应用程序菜单】->“新建”->“直接激励项目”。

步骤 2: 选择项目路径，输入项目名称，点击“保存”。

##### 3.2.2 打开直接激励项目

方法 1:

步骤 1: 【应用程序菜单】->“打开”->“直接激励项目”。

步骤 2: 选择项目文件，点击“打开”。

方法 2:

步骤 1: 【应用程序菜单】->在“最近项目”面板中选择需要打开的项目。

### 3.2.3 关闭直接激励项目

步骤 1: 【控制台】->选择“直接激励管理器”。

步骤 2: 【应用程序菜单】->“关闭”。

### 3.2.4 保存直接激励项目

步骤 1: 【控制台】->选择“直接激励管理器”。

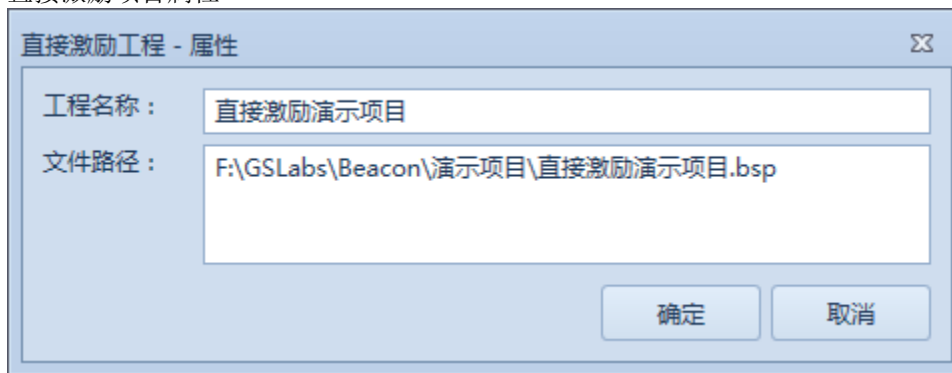
步骤 2: 【应用程序菜单】->“保存”或者“另存为”。

### 3.2.5 修改直接激励项目属性

步骤 1: 【控制台】->选择“直接激励管理器”。

步骤 2: 【工具栏】->“属性”；或者，【控制台】->鼠标右键弹出快捷菜单->“属性”。

直接激励项目属性



属性	说明
工程名称	显示工程的名称，可修改。
文件路径	显示工程文件的存储路径，不可以修改。

### 3.2.6 添加直接激励项

步骤 1: 【控制台】->选择“直接激励管理器”。

步骤 2: 在“直接激励管理器”中选中一个激励项，新添加激励项将插入选中激励项的前面。

步骤 3: 【工具栏】->“添加”->“添加直接激励”；或者，【控制台】->鼠标右键弹出快捷菜单->“添加直接激励”。

步骤 4: 编辑新添加激励项的数据格式、数据、延时、激活属性。

### 3.2.7 删除直接激励项

步骤 1: 【控制台】->选择“直接激励管理器”。

步骤 2: 在“直接激励管理器”中选中要删除的激励项。

步骤 3: 【工具栏】->“删除”；或者，【控制台】->鼠标右键弹出快捷菜单->“删除”。

### 3.2.8 激励

激励前提：通信接口已经打开。

步骤 1：【控制台】->选择“直接激励管理器”。

步骤 2：【工具栏】->“激励”；或者，【工具栏】->“激励”菜单->“激励”/“循环激励”；或者，【控制台】->鼠标右键弹出快捷菜单->“激励”/“循环激励”。

### 3.2.9 停止激励

停止激励前提：正在激励。

步骤 1：【控制台】->选择“直接激励管理器”。

步骤 2：【工具栏】->“停止”；或者，【控制台】->鼠标右键弹出快捷菜单->“停止”。

## 3.3 用途

直接激励项目有如下几种使用环境。

- 与被激励方进行命令交互，作为监控被激励方的命令控制台
- 给被激励方增加干扰信号，测试被激励方

## 4. 协议激励项目

### 4.1 简介

协议是通信系统必不可少的部分，如何组织和开展有效的协议测试，低成本地应付快速多变的通信环境，越来越受到企业和研发人员的关注。过去，需要为每一份通信协议定制一个测试工具，众多定制的测试工具，极大的增加测试开发、维护和学习成本，与竞争日益激烈的现代电子研发行业背道而驰。

格西调试精灵的协议激励功能彻底地把多变的通信规约脱离通信软件，使得通信软件能够应付快速多变的通信环境。

格西调试精灵协议激励功能带来的好处：

- 单一的测试工具取代众多定制的测试工具，极大的降低测试开发、维护和学习成本
- 快捷的激励源管理方式，即改即测，极大的降低研发成本
- 以文件的方式组织项目，有利于管理众多的测试项目，有利于测试标准化
- 支持循环激励与激励报告，提高测试自动化水平

### 4.2 基本操作

#### 4.2.1 新建协议激励项目

步骤 1：【应用程序菜单】->“新建”->“协议激励项目”。

步骤 2：选择项目路径，输入项目名称，点击“保存”。

#### 4.2.2 打开协议激励项目

方法 1：

步骤 1：【应用程序菜单】->“打开”->“协议激励项目”。

步骤 2：选择项目文件，点击“打开”。

方法 2：

步骤 1：【应用程序菜单】->在“最近项目”面板中选择需要打开的项目。

#### 4.2.3 关闭协议激励项目

步骤 1：【控制台】->选择“协议激励管理器”。

步骤 2：【应用程序菜单】->“关闭”。

#### 4.2.4 保存协议激励项目

步骤 1：【控制台】->选择“协议激励管理器”。

步骤 2：【应用程序菜单】->“保存”或者“另存为”。

#### 4.2.5 修改协议激励项目属性

步骤 1：【控制台】->选择“协议激励管理器”。

步骤 2：在“协议激励管理器”中选择项目节点。

步骤 3：【工具栏】->“属性”；或者，【控制台】->鼠标右键弹出快捷菜单->“属性”。

协议激励项目属性

属性	说明
工程名称	显示工程的名称，可修改。
工作模式	协议激励工程支持“单工”和“双工”两种工作模式。“单工”模式下不管有没有“被动模式”的协议项，一律按照顺序执行选中的协议项；“双工”模式下“主动模式”协议项和“被动模式”协议项是分别同时执行的，“主动模式”协议项按顺序执行，“被动模式”的协议项是不按照顺序的，匹配正确即执行。 例如：模拟主动型设备，按需求可用“单工”或者“双工”；模拟被动型设备，一般使用“双工”，等待需要执行的命令。
文件路径	显示工程文件的存储路径，不可以修改。
协议变量（通过工具栏的“变量”按钮打开）	支持设置工程的全局变量，一条记录由“变量名称”、“变量值”和“赋值掩码”组成，协议执行过程中，自动匹配“变量名称”，如果协议项的对应字段的名称和变量名称一致，并且默认值等于“赋值掩码”，则使用“变量值”替换对应的字段值。一般用在类似设备地址常出现而且变化的变量字段中，方便在更换设备时统一操作。

#### 4.2.6 添加协议集

步骤 1: 【控制台】->选择“协议激励管理器”。

步骤 2: 在“协议激励管理器”中选中一个协议项/协议集，作为新添加协议集的位置。

步骤 3: 【工具栏】->“添加”->“添加协议集”；或者，【控制台】->鼠标右键弹出快捷菜单->“添加协议集”。

步骤 4: 在协议集属性对话框中输入协议集名称，点击“确定”。

步骤 5: 选择新添加协议集相对当前选中项的位置, 点击“确定”。



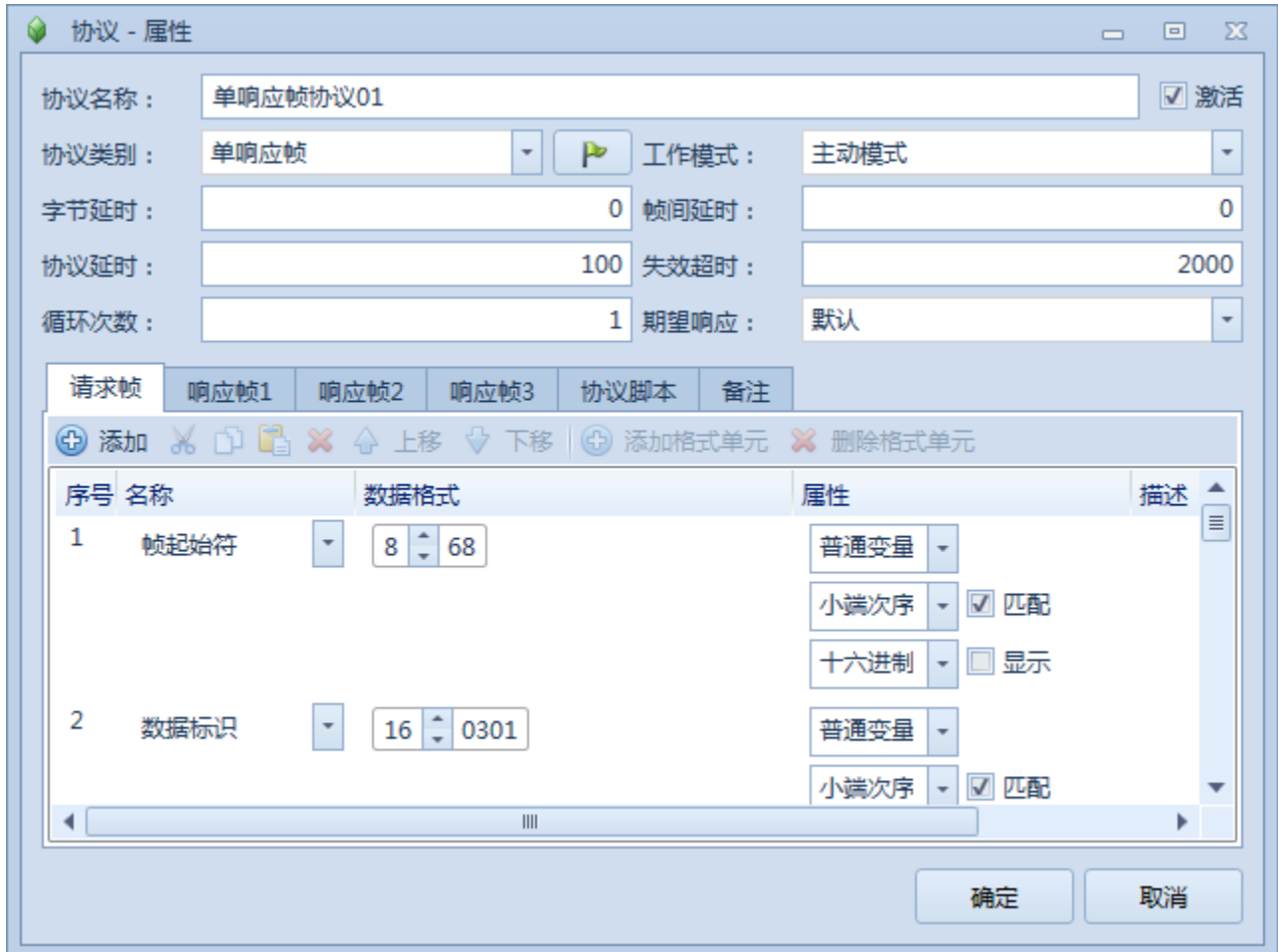
#### 4.2.7 添加协议项

步骤 1: 【控制台】->选择“协议激励管理器”。

步骤 2: 在“协议激励管理器”中选中一个协议项/协议集, 作为新添加协议的位置。

步骤 3: 【工具栏】->“添加”->“添加协议”; 或者, 【控制台】->鼠标右键弹出快捷菜单->“添加协议”。

步骤 4: 在协议属性对话框中, 输入相关参数, 点击“确定”。



属性	说明
协议名称	描述协议名称。
激活	使能协议节点。
协议类型	为了实现结果数据的分类显示，协议激励项目能够自定义数据类型，协议项在激励时根据所属的协议类型把结果输出到对应的自定义数据区中。可以通过“管理协议类型”按钮打开协议类型管理器。
工作模式	协议项的工作模式有“主动模式”和“被动模式”两种。“主动模式”表示该协议项是主动命令，是发送请求帧，校验响应帧；“被动模式”表示该协议项是被动命令，是等待请求帧，然后根据请求帧情况发送响应帧。
协议延时	表示协议执行完毕后延时多长时间才进入下一个协议激励。
失效超时	表示协议在匹配到正确数据之前最长等待时间，超过该时间长度即判失败。
循环次数	表示循环激励该协议的次数。
期望响应	“主动模式”表示发送请求帧后期望收到的响应帧，默认值表示自动匹配；“被动模式”表示收到请求帧后发送出去的响应帧，默认值表示发送响应帧 1。
请求帧	显示和编辑请求帧的格式和内容。参见 4.3 节。
响应帧 1/响应帧 2/响应帧 3	显示和编辑响应帧 1/响应帧 2/响应帧 3 的格式和内容。参见 4.3 节。
协议脚本	显示和编辑协议脚本。参见 4.4 节。

步骤 5：选择新添加协议相对当前选中项的位置，点击“确定”。

#### 4.2.8 激励

激励前提：通信接口已经打开。

步骤 1：【控制台】->选择“协议激励管理器”。

步骤 2：【工具栏】->“激励”；或者，【工具栏】->“激励”菜单->“激励”/“循环激励”；或者，【控制台】->鼠标右键弹出快捷菜单->“激励”/“循环激励”。

#### 4.2.9 停止激励

停止激励前提：正在激励。

步骤 1：【控制台】->选择“协议激励管理器”。

步骤 2：【工具栏】->“停止”；或者，【控制台】->鼠标右键弹出快捷菜单->“停止”。

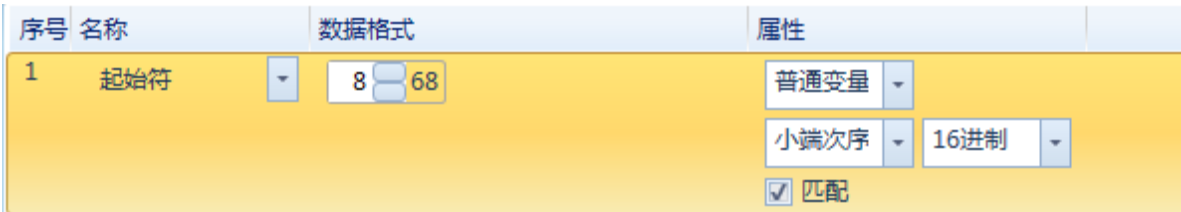


### 4.3 协议帧

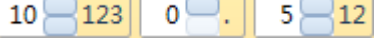
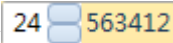

协议帧由一个或者多个帧格式单元组成，帧格式单元可以非字节对齐，但协议帧必须字节对齐。



帧格式单元



属性	说明
序号	表示帧格式单元的排列顺序。
名称	表示帧格式单元的名称。
数据格式	表示帧格式单元的数据构成，可以有多段数据格式构成。数据格式左边表示位数，右边表示数值。数据格式的位数可以是任意位数。例如： 单段字节对齐 24 123456， 多段字节对齐 10 123 0 . 6 12， 单段非字节对齐 15 4FFF，

	多段非字节对齐  。
属性（变量类型）	帧格式单元的变量类型支持普通变量、计算变量和重复变量三种。普通变量是常量；计算变量是通过计算前面数据得到的结果作为变量值，如校验和；重复变量是指该变量在该数据帧中可能重复多次，0 表示自动，可表示 $0 \sim n$ 次，>0 表示精确次数。
属性（存储模式）	帧格式单元的数据支持小端模式和大端模式两种存储模式，例如，字节顺序从低到高（即发送/接收的字节顺序）为 0x12 0x34 0x56，小端模式下设置为  ，大端模式下设置为  。
属性（显示方式）	帧格式单元的数据支持 16 进制和字符串两种显示方式，其中字符串的字符编码方式由设置中的字符编码方式决定。
属性（匹配）	匹配选择用来判断协议帧完整性。

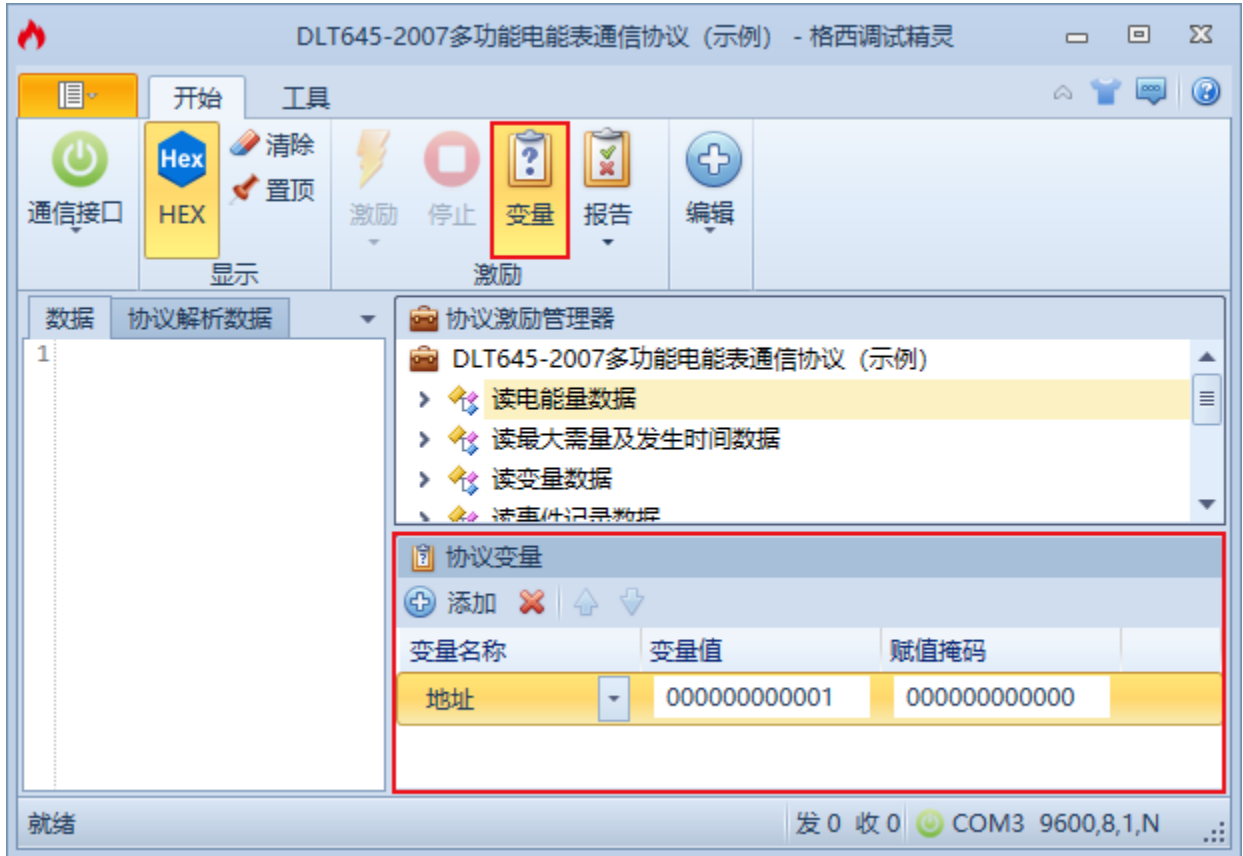
## 工具条



命令	功能
添加	在当前选择的帧格式单元中插入新的帧格式单元。
删除	删除当前选择的帧格式单元。
上移	将当前选择的帧格式单元向上移动一格。
下移	将当前选择的帧格式单元向下移动一格。
添加格式单元	在当前选择的帧格式单元中添加一个数据格式单元。
删除格式单元	删除当前选择的帧格式单元中的数据格式单元。

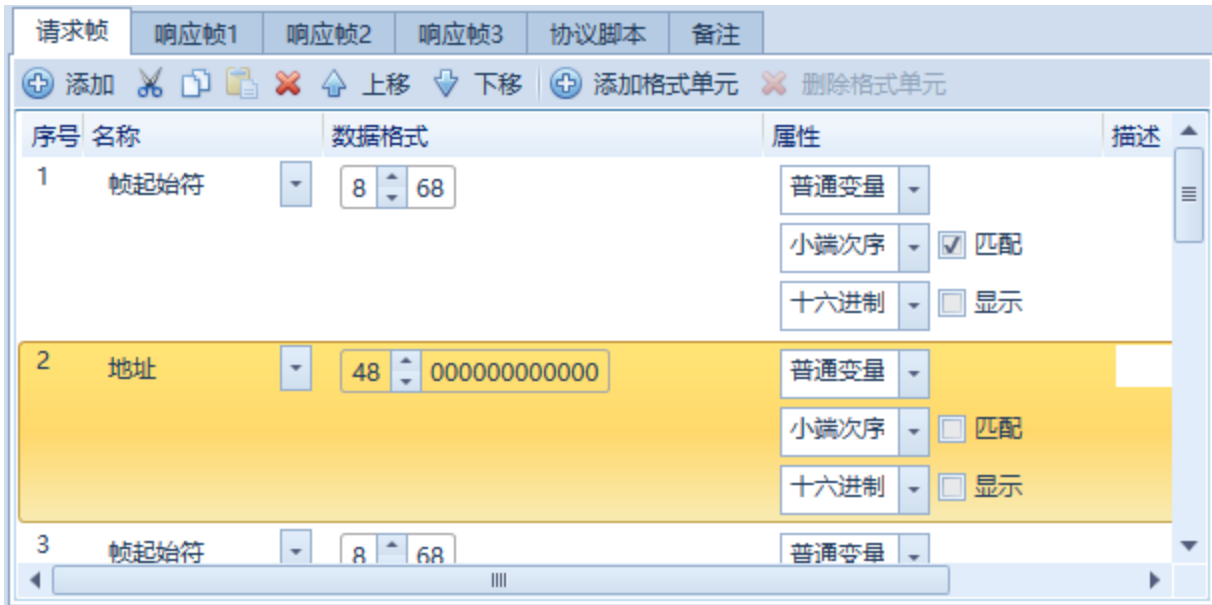
#### 4.4 协议变量

协议变量的作用是给一些可变值的字段提供便利，例如协议中的“地址”字段，用协议变量可以在运行时统一对所有协议条目中的“地址”字段替换为新的地址值，不必每条协议条目重复修改。

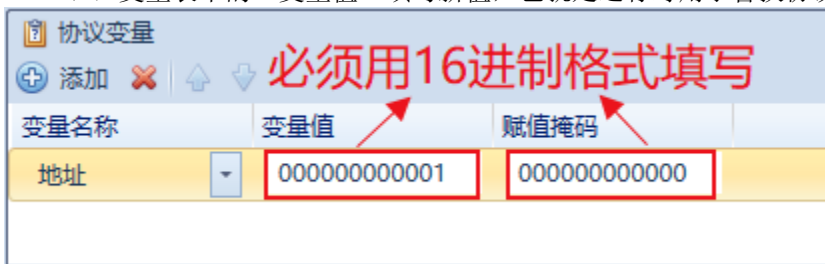


协议变量必须满足如下三条准则，才能在运行时替换协议条目对应字段的值。

- (1) 变量表中的“变量名称”和协议条目对应的字段名称一样；
- (2) 变量表中的“赋值掩码”和协议条目对应的字段的“数据格式”中的值一样，比如协议条目中“地址”字段是 48 位 000000000000，则变量表的赋值掩码也要填写 000000000000。



(3) 变量表中的“变量值”填写新值，也就是运行时用于替换协议条目中的旧值。



**注意：**变量值和赋值掩码必须转换为 16 进制格式填写，即使协议条目对应字段是 10 进制或字符串的表示。

4.5 协议脚本

4.5.1 脚本界面

```

12 /**
13  * 脚本类
14  */
15 public class Script
16 {
17     /*****
18     函数名称：OnRequest
19     功能说明：发送请求帧之前执行。
20     输入参数：context - 运行时上下文，存储运行时的参数
21     输出参数：无
22     返回参数：成功返回1，失败返回0
23     *****/
24     public int OnRequest(BSCaseContext context)
25     {
26         int val = ParallelPort.Read(0x378);
27         context.Msg = "ParallelPort.Read=" + val.ToString();
28         return 1;
29     }
30

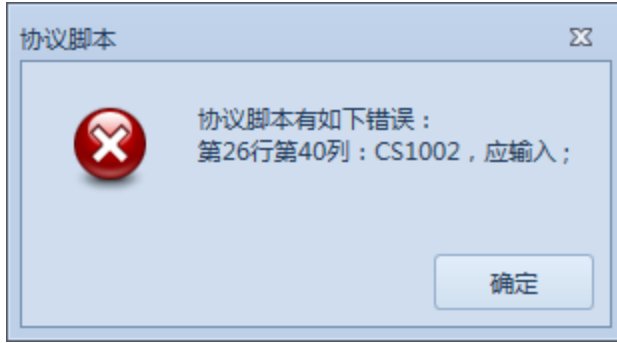
```

工具条



命令	功能
脚本语言	设置本协议项的脚本语言，支持 C#、VB、Jscript 三种语言。
编译检查	通过编译检查当前脚本的是否有错误。
导入	导入外部存储的脚本。
导出	将当前的脚本导出为文件。
剪切	剪切选中的脚本文本。
复制	复制选中的脚本文本。
粘贴	在当前光标位置粘贴剪切板的文本。
删除	删除选中的脚本文本。
清除	清除所有脚本。

脚本有错误，编译检查会提示错误的位置、错误码以及错误信息。



#### 4.5.2 脚本结构

协议脚本支持 C#、VB、Jscript 三种语言，软件安装目录下的 Scripts 子目录保存了三种语言的脚本模版，可以通过“导入”命令导入脚本编辑器。

下面是 C#版本的脚本模版，由一个.NET 类 Script 构成，该类包含三个方法，分别是 OnRequest 方法，OnProcess 方法，OnRespond 方法。

```

/*****
Copyright (c) 2014, 上海格西信息科技有限公司
文件名称: Script.cs
文件描述: C#脚本模版
*****/

/**
 * 命名空间定义
 */
//using System;

/**
 * 脚本类
 */
public class Script
{
    /**
    函数名称: OnRequest
    功能说明: 主动模式在发送请求帧之前执行，被动模式在接收请求帧之后执行。
    输入参数: context - 运行时上下文，存储运行时的参数
    输出参数: 无
    返回参数: 成功返回 1，失败返回 0
    *****/
    public int OnRequest(BSCaseContext context)
    {
        return 1;
    }

    /**
    函数名称: OnProcess
    功能说明: 等待接收/发送响应帧时每一个处理周期（约 5ms）执行 1 次。

```

```

输入参数: context - 运行时上下文, 存储运行时的参数
输出参数: 无
返回参数: 成功返回 1, 失败返回 0
*****/
public int OnProcess(BSCaseContext context)
{
    return 1;
}

/*****
函数名称: OnRespond
功能说明: 主动模式接收响应帧之后执行, 被动模式在发送响应帧之后执行。
输入参数: context - 运行时上下文, 存储运行时的参数
输出参数: 无
返回参数: 成功返回 1, 失败返回 0
*****/
public int OnRespond(BSCaseContext context)
{
    return 1;
}
}
    
```

### 4.5.3 脚本参数 BSCaseContext 类

脚本函数的唯一输入参数是 BSCaseContext 类型, 该类型实例承载了整个激励过程的脚本执行上下文, BSCaseContext 类提供了属性和方法让脚本调用, 完成对激励、协议帧、通信接口参数等执行参数的控制和管理。

#### 4.5.3.1 Power 属性

获取或者设置激励的运行状态, 该值表明激励是否在运行。

#### 语法

C#	<code>public bool Power { get; set; }</code>
	属性值 类型: <code>System.Boolean</code>
VB	<code>Public Property Power As Boolean</code> Get Set
	属性值 类型: <code>System.Boolean</code>

#### 备注

当需要停止激励运行时, 可以在脚本中设置 Power 值为 false, 脚本运行完毕后停止激励。

#### 示例

下面的示例在脚本的 OnRequest 方法中设置 Power 为 false。

C#	<pre>public int OnRequest(BSCaseContext context) {     context.Power = false;     context.Msg = "Power=false.\r\n";     return 1; }</pre>
VB	<pre>Public Function OnRequest(ByRef context As BSCaseContext) As Integer     context.Power = false     context.Msg = "Power=false.\r\n"     OnRequest = 1 End Function</pre>

#### 4.5.3.2 CommParameters 属性

获取用于描述通信接口的对象。在串口通信中，该对象的类型为 BSComStreamParameters，参见 4.4.4，在网络接口通信中，该对象的类型为 BSNetStreamParameters，参见 4.4.5。

#### 语法

C#	<pre>public object CommParameters { get; }</pre> <p>属性值 类型: <a href="#">System.Object</a></p>
VB	<pre>Public ReadOnly Property CommParameters As Object     Get</pre> <p>属性值 类型: <a href="#">System.Object</a></p>

#### 备注

当需要运行中获取或者修改通信接口参数时，可以在脚本中先把 CommParameters 转换通信接口参数类型，然后再进行操作。

#### 示例

下面的示例在脚本的 OnRespond 方法中设置串口通信接口参数。

C#	<pre>public int OnRespond(BSCaseContext context) {     BSComStreamParameters comParams = context.CommParameters         as BSComStreamParameters;     comParams.BaudRate = 115200; // 波特率设置为 115200     comParams.Parity = Parity.Odd; // 校验位设置为奇校验      StringBuilder sb = new StringBuilder();     sb.Append(string.Format("串口通信参数设置为: BaudRate={0}, Parity={1}\r\n",         comParams.BaudRate, comParams.Parity));     // 输出信息     context.Msg = sb.ToString();      return 1; }</pre>
----	---



VB	<pre> Public Function OnRespond (ByRef context As BSCaseContext) As Integer     Dim comParams As BSComStreamParameters     comParams = DirectCast(context.CommParameters, BSComStreamParameters)     comParams.BaudRate = 115200 ' 波特率设置为 115200     comParams.Parity = Parity.Odd ' 校验位设置为奇校验      Dim sb As StringBuilder     sb = new StringBuilder()     sb.Append(String.Format("串口通信参数设置为: BaudRate={0}, Parity={1}\r\n",         comParams.BaudRate, comParams.Parity))     ' 输出信息     context.Msg = sb.ToString()      OnRequest = 1 End Function </pre>
----	--

#### 4.5.3.3 Msg 属性

获取或者设置一个信息字符串，该字符串在脚本函数结束后显示到数据区。

#### 语法

C#	<pre>public string Msg { get; set; }</pre> <p>属性值 类型: <a href="#">System.String</a></p>
VB	<pre>Public Property Msg As String     Get     Set</pre> <p>属性值 类型: <a href="#">System.String</a></p>

#### 备注

当需要在脚本函数结束后显示信息到数据区时，可以在脚本函数中设置 Msg 值，Msg 的值是读后清除的，如果需要设置多个信息，可以先通过 StringBuilder 拼接，再统一赋值给 Msg，或者使用 AppendMsg 方法。

#### 示例

下面的示例在脚本的 OnRequest 方法中设置 Msg。

C#	<pre> public int OnRequest(BSCaseContext context) {     context.Msg = "Hello World.\r\n";     return 1; } </pre>
VB	<pre> Public Function OnRequest(ByRef context As BSCaseContext) As Integer     context.Msg = "Hello World.\r\n"     OnRequest = 1 End Function </pre>

## 4.5.3.4 MatchResult 属性

获取协议匹配的索引号。从 0 开始，匹配请求帧有效值为 0，匹配响应帧有效值为 0, 1 和 2。

## 语法

C#	<code>public int MatchResult { get; }</code>
	属性值 类型: <code>System.Integer</code>
VB	Public Property MatchResult As Integer Get
	属性值 类型: <code>System.Integer</code>

## 备注

## 示例

下面的示例在脚本的 OnRespond 方法中获取。

C#	<pre>public int OnRespond (BSCaseContext context) {     if (context.MatchResult == 0)     {         // 期望收到响应帧 1         string msg = "响应帧 1 的帧格式单元 3:" + context.GetResponseValue(0, 2) + "\r\n";         context.Msg = msg;         return 1; // 返回成功     }     return 0; // 返回失败 }</pre>
VB	<pre>Public Function OnRespond (ByRef context As BSCaseContext) As Integer     Dim msg As String     If context.MatchResult = 0 Then         '期望收到响应帧 1         msg = "响应帧 1 的帧格式单元 3:" + context.GetResponseValue(0, 2) + "\r\n"         context.Msg = msg         OnRespond = 1 '返回成功     End If      OnRespond = 0 '返回失败 End Function</pre>

## 4.5.3.5 ExpectedRespond 属性

获取或设置 Respond 属性的索引，有效值从 0 开始。主动模式表示期望收到的 Respond 的 index，-1 表示自动检测，从动模式表示收到 Request 后要响应的 Respond 的索引。

## 语法

C#	<code>public int ExpectedRespond{ get; set; }</code>
	属性值 类型: <code>System. Integer</code>
VB	Public Property ExpectedRespond As Integer Get Set
	属性值 类型: <code>System. Integer</code>

## 备注

## 示例

下面的示例在脚本的 OnRequest 方法中获取。

C#	<pre>public int OnRequest(BSCaseContext context) {     // 期望的响应帧     string msg = "协议期望的响应帧:" + context.ExpectedRespond + "\r\n";     context.Msg = msg;     return 1; // 返回成功 }</pre>
VB	<pre>Public Function OnRequest(ByRef context As BSCaseContext) As Integer     Dim msg As String     msg = "协议期望的响应帧:" + context.ExpectedRespond + "\r\n"     context.Msg = msg     OnRequest = 1 ' 返回成功 End Function</pre>

## 4.5.3.6 CaseProjectName 属性

获取协议激励项目名称。

## 语法

C#	<code>public string CaseProjectName{ get; }</code>
	属性值 类型: <code>System. String</code>

## 4.5.3.7 CaseName 属性

获取协议激励项目正在执行的协议条目名称。

## 语法

C#	<code>public string CaseName{ get; }</code>
	属性值 类型: <code>System. String</code>

## 4.5.3.8 WorkingDirectory 属性

获取当前工作目录。

## 语法

C#	<code>public string WorkingDirectory { get; }</code>
	属性值 类型: <code>System.String</code>

## 4.5.3.9 Loops 属性

获取或设置当前执行协议条目的循环次数。

## 语法

C#	<code>public int Loops { get; set; }</code>
	属性值 类型: <code>System.Integer</code>

## 4.5.3.10 AppendMsg 方法

添加一个信息字符串，该字符串将附加在原来 Msg 后面，在脚本函数结束后显示到数据区。

## 语法

C#	<code>public void AppendMsg(string msg)</code>
	参数 msg 类型: <code>System.String</code> 要附加到 Msg 后面的新字符串
VB	<code>Public Sub AppendMsg(msg As String)</code>
	参数 msg 类型: <code>System.String</code> 要附加到 Msg 后面的新字符串

## 备注

当需要设置多个信息时，使用 AppendMsg 方法。

## 示例

下面的示例在脚本的 OnRequest 方法中使用 AppendMsg 添加多个信息字符串。

C#	<pre>public int OnRequest(BSCaseContext context) {     context.AppendMsg("Hello A. \r\n");     context.AppendMsg("Hello B. \r\n");     return 1; }</pre>
----	--

VB	<pre>Public Function OnRequest(ByRef context As BSCaseContext) As Integer     context.AppendMsg("Hello A. \r\n")     context.AppendMsg("Hello B. \r\n")     OnRequest = 1 End Function</pre>
----	--

#### 4.5.3.11 GetVariant 方法

获取用户自定义变量的值。

#### 语法

	<pre>public object GetVariant(object key)</pre>
C#	<p>参数 key 类型: <code>System.Object</code> 用户变量的名称, 是一个关键字, 不允许有重复的 key</p> <p>返回值 类型: <code>System.Object</code> 用户变量的值, 用户通过强制类型转换获取实际类型的值</p>
VB	<pre>Public Function GetVariant (key As Object) As Object</pre> <p>参数 key 类型: <code>System.Object</code> 用户变量的名称, 是一个关键字, 不允许有重复的 key</p> <p>返回值 类型: <code>System.Object</code> 用户变量的值, 用户通过强制类型转换获取实际类型的值</p>

#### 备注

当需要在激励运行过程中设置用户自定义变量时, 使用 `SetVariant` 方法, 在需要获取时通过 `GetVariant` 方法获取, 自定义的变量在整个激励运行过程中有效。

#### 示例

下面的示例在脚本的 `OnRequest` 方法中使用 `GetVariant`。

C#	<pre>public int OnRequest(BSCaseContext context) {     context.SetVariant ("Str1", "Hello A. \r\n");     context.Msg = context.GetVariant ("Str1"). ToString();     return 1; }</pre>
VB	<pre>Public Function OnRequest(ByRef context As BSCaseContext) As Integer     context.SetVariant ("Str1", "Hello A. \r\n")     context.Msg = context.GetVariant ("Str1"). ToString()     OnRequest = 1 End Function</pre>

## 4.5.3.12 SetVariant 方法

设置用户自定义变量的值。

## 语法

C#	<code>public void SetVariant(object key, object val)</code>
	参数 key 类型: <code>System.Object</code> 用户变量的名称, 是一个关键字, 如果变量存在, 修改原来变量的值, 不存在则创建 val 类型: <code>System.Object</code> 用户变量的值
VB	<code>Public Sub SetVariant (key As Object, val As Object)</code>
	参数 key 类型: <code>System.Object</code> 用户变量的名称, 是一个关键字, 如果变量存在, 修改原来变量的值, 不存在则创建 val 类型: <code>System.Object</code> 用户变量的值

## 备注

当需要在激励运行过程中设置用户自定义变量时, 使用 SetVariant 方法。

## 示例

下面的示例在脚本的 OnRequest 方法中使用 SetVariant。

C#	<pre>public int OnRequest(BSCaseContext context) {     context.SetVariant ("Str1", "Hello A.\r\n");     context.Msg = context.GetVariant("Str1").ToString();     return 1; }</pre>
VB	<pre>Public Function OnRequest(ByRef context As BSCaseContext) As Integer     context.SetVariant ("Str1", "Hello A.\r\n")     context.Msg = context.GetVariant("Str1").ToString()     OnRequest = 1 End Function</pre>

## 4.5.3.13 GetRequestValue 方法

获取请求帧的帧格式单元的值。

## 语法

C#	<code>public string GetRequestValue(int unitIndex)</code>
	参数

	unitIndex 类型: <code>System.Integer</code> 帧格式单元索引, 索引号从 0 开始。  返回值 类型: <code>System.String</code> 帧格式单元的值
VB	<pre>Public Function GetRequestValue (unitIndex As Integer) As String</pre> 参数 unitIndex 类型: <code>System.Integer</code> 帧格式单元索引  返回值 类型: <code>System.String</code> 帧格式单元的值

**备注**

获取请求帧的帧格式单元的值。

**示例**

下面的示例在脚本的 OnRespond 方法中使用 GetRequestValue。

C#	<pre>public int OnRespond (BSCaseContext context) {     string val = context.GetRequestValue (0);     context.Msg = val;     return 1; }</pre>
VB	<pre>Public Function OnRespond (ByRef context As BSCaseContext) As Integer     Dim val As String     val = context.GetRequestValue (0)     context.Msg = val     OnRequest = 1 End Function</pre>

## 4.5.3.14 SetRequestValue 方法

设置请求帧的帧格式单元的值, **通常用于主动模式**, 在发送请求帧之前修改帧数据。

**语法**

	<pre>public int SetRequestValue(int unitIndex, string unitValue)</pre>
C#	参数 unitIndex 类型: <code>System.Integer</code> 帧格式单元索引, 索引号从 0 开始。 unitValue 类型: <code>System.String</code>

	新帧格式单元的值  返回值 类型: <code>System.Integer</code> 成功返回 0, 失败返回-1
VB	<pre>Public Function SetRequestValue (unitIndex As Integer, unitValue As String) As Integer</pre> 参数 unitIndex 类型: <code>System.Integer</code> 帧格式单元索引, 索引号从 0 开始。 unitValue 类型: <code>System.String</code> 新帧格式单元的值  返回值 类型: <code>System.Integer</code> 成功返回 0, 失败返回-1

**备注**

当需要在激励运行过程中需要更改协议帧的帧格式单元的值时, 使用 SetRequestValue 方法。

**示例**

下面的示例在脚本的 OnRequest 方法中使用 SetRequestValue。

C#	<pre>public int OnRequest(BSCaseContext context) {     context.SetRequestValue (1, "02");     return 1; }</pre>
VB	<pre>Public Function OnRequest(ByRef context As BSCaseContext) As Integer     context.SetRequestValue (1, "02")     OnRequest = 1 End Function</pre>

## 4.5.3.15 GetRespondValue 方法

获取响应帧的帧格式单元的值。

**语法**

	<pre>public string GetRespondValue (int respondIndex, int unitIndex)</pre>
C#	参数 respondIndex 类型: <code>System.Integer</code> 响应帧索引, 目前合法值 0、1 和 2。 unitIndex 类型: <code>System.Integer</code> 帧格式单元索引, 索引号从 0 开始。



	返回值 类型: <code>System.String</code> 帧格式单元的值
VB	<pre>Public Function GetRespondValue (respondIndex As Integer, unitIndex As Integer) As String</pre> 参数 respondIndex 类型: <code>System.Integer</code> 响应帧索引, 目前合法值 0、1 和 2。 unitIndex 类型: <code>System.Integer</code> 帧格式单元索引  返回值 类型: <code>System.String</code> 帧格式单元的值

**备注**

获取响应帧的帧格式单元的值, 判断收到的激励响应是否正确。

**示例**

下面的示例在脚本的 `OnRespond` 方法中使用 `GetRespondValue`。

C#	<pre>public int OnRespond (BSCaseContext context) {     string val = context.GetRespondValue(0, 1);     context.Msg = val;     return 1; }</pre>
VB	<pre>Public Function OnRespond (ByRef context As BSCaseContext) As Integer     Dim val As String     val = context.GetRespondValue(0, 1)     context.Msg = val     OnRequest = 1 End Function</pre>

4.5.3.16 `SetRespondValue` 方法

设置响应帧的帧格式单元的值, **通常用于被动模式**, 在发送响应帧之前修改帧数据。

**语法**

	<pre>public int SetRespondValue(int respondIndex, int unitIndex, string unitValue)</pre>
C#	参数 respondIndex 类型: <code>System.Integer</code> 响应帧索引, 目前合法值 0、1 和 2。 unitIndex 类型: <code>System.Integer</code>

	<p>帧格式单元索引，索引号从 0 开始。 unitValue 类型: <code>System.String</code> 新帧格式单元的值</p> <p>返回值 类型: <code>System.Integer</code> 成功返回 0，失败返回-1</p>
VB	<p><code>Public Function SetRespondValue (respondIndex As Integer, unitIndex As Integer, unitValue As String) As Integer</code></p> <p>参数 respondIndex 类型: <code>System.Integer</code> 响应帧索引，目前合法值 0、1 和 2。 unitIndex 类型: <code>System.Integer</code> 帧格式单元索引，索引号从 0 开始。 unitValue 类型: <code>System.String</code> 新帧格式单元的值</p> <p>返回值 类型: <code>System.Integer</code> 成功返回 0，失败返回-1</p>

**备注**

一般在被动模式协议项中使用该方法，收到请求帧后，做出响应。

**示例**

下面的示例在脚本的 OnRequest 方法中使用 SetRepondValue。

C#	<pre>public int OnRequest(BSCaseContext context) {     context.SetRespondValue (0, 1, "02");     return 1; }</pre>
VB	<pre>Public Function OnRequest(ByRef context As BSCaseContext) As Integer     context.SetRespondValue (0, 1, "02")     OnRequest = 1 End Function</pre>

4.5.3.17 GetRequestData 方法

获取请求帧的字节数组。

**语法**

C#	<pre>public byte[] GetRequestData ()</pre>
	<p>参数 无。</p>

	返回值 类型: <code>System.Byte[]</code> 请求帧的字节数组
VB	<code>Public Function GetRequestData () As Byte()</code>
	参数 无  返回值 类型: <code>System.Byte()</code> 请求帧的字节数组

**备注**

无。

## 4.5.3.18 GetRespondData 方法

获取响应帧的字节数组。

**语法**

C#	<code>public byte[] GetRespondData(int respondIndex)</code>
	参数 respondIndex 类型: <code>System.Integer</code> 响应帧索引, 目前合法值 0、1 和 2。  返回值 类型: <code>System.Byte[]</code> 响应帧的字节数组
VB	<code>Public Function GetRespondData (respondIndex As Integer) As Byte()</code>
	参数 respondIndex 类型: <code>System.Integer</code> 响应帧索引, 目前合法值 0、1 和 2。  返回值 类型: <code>System.Byte()</code> 响应帧的字节数组

**备注**

无。

## 4.5.3.19 GetMatchData 方法

获取匹配的结果字节数组, 不管主动模式还是被动模式, 均采用该方法获取。

**语法**

C#	<code>public byte[] GetMatchData()</code>
----	---

	参数 无。  返回值 类型: <code>System.Byte[]</code> 匹配的结果字节数组
VB	<code>Public Function GetMatchData() As Byte()</code>  参数 无  返回值 类型: <code>System.Byte()</code> 匹配的结果字节数组

**备注**

## 4.5.3.20 GetMatchData 方法

根据单元索引获取匹配的结果字节数组。

**语法**

	<code>public byte[] GetMatchData(int unitIndex)</code>
C#	参数 unitIndex 类型: <code>System.Integer</code> 帧格式单元索引, 索引号从 0 开始。  返回值 类型: <code>System.Byte[]</code> 匹配的单元结果字节数组

**备注**

不管主动模式还是被动模式, 均采用该组方法获取收到的数据。

## 4.5.3.21 GetMatchValue 方法

根据单元索引获取匹配的结果字符串值。

**语法**

	<code>public string GetMatchValue(int unitIndex)</code>
C#	参数 unitIndex 类型: <code>System.Integer</code> 帧格式单元索引, 索引号从 0 开始。  返回值 类型: <code>System.String</code>

	匹配的单元结果字符串值
--	-------------

**备注**

不管主动模式还是被动模式，均采用该组方法获取收到的数据。

## 4.5.3.22 GetCaseVariable 方法

根据变量名称获取协议变量表的值。

**语法**

	<code>public string GetCaseVariable(string name)</code>
C#	参数 name 类型: <code>System.String</code> 协议变量表的变量名称。
	返回值 类型: <code>System.String</code> 协议变量值

**备注**

## 4.5.3.23 SetCaseVariable 方法

设置协议变量表的变量值。

**语法**

	<code>public bool SetCaseVariable(string name, string value)</code>
C#	参数 name 类型: <code>System.String</code> 协议变量表的变量名称。 value 类型: <code>System.String</code> 新的变量值（必须为等价的 16 进制表示字符串）。
	返回值 类型: <code>System.Boolean</code> 成功返回 true，失败返回 false

**备注**

## 4.5.3.24 SetCaseVariable 方法

设置协议变量表的变量值和掩码。

## 语法

	<code>public bool SetCaseVariable(string name, string value , string valueMask)</code>
C#	<p>参数</p> <p>name 类型: <code>System.String</code> 协议变量表的变量名称。</p> <p>value 类型: <code>System.String</code> 新的变量值（必须为等价的 16 进制表示字符串）。</p> <p>valueMask 类型: <code>System.String</code> 新的变量掩码（必须为等价的 16 进制表示字符串）。</p> <p>返回值 类型: <code>System.Boolean</code> 成功返回 true, 失败返回 false</p>

## 备注

## 4.5.3.25 GotoCase 方法

跳到指定偏移的 Case 执行。

## 语法

	<code>public void GotoCase(int offset)</code>
C#	<p>参数</p> <p>offset 类型: <code>System.Integer</code> 偏移值, 向前跳用负数, 向后跳用正数, 0 表示再执行本 Case。</p> <p>返回值 无</p>

## 备注

4.5.4 串口参数 *BSComStreamParameters* 类

## 4.5.4.1 Port 属性

获取或设置串口通信端口。

## 语法

	<code>public string Port { get; set; }</code>
C#	<p>属性值 类型: <code>System.String</code></p>

VB	Public Property Port As String Get Set
	属性值 类型: <a href="#">System.String</a>

**备注****示例**

下面的示例在脚本的 OnRequest 方法中获取串口 Port。

C#	<pre>public int OnRequest(BSCaseContext context) {     BSComStreamParameters comParams = context.CommParameters         as BSComStreamParameters;     StringBuilder sb = new StringBuilder();     sb.Append(string.Format("串口号为: Port={0}\r\n",         comParams.Port));     // 输出信息     context.Msg = sb.ToString();      return 1; }</pre>
VB	<pre>Public Function OnRequest(ByRef context As BSCaseContext) As Integer     Dim comParams As BSComStreamParameters     comParams = DirectCast(context.CommParameters, BSComStreamParameters)      Dim sb As StringBuilder     sb = new StringBuilder()     sb.Append(String.Format("串口号为: Port={0}\r\n",         comParams.Port))     ' 输出信息     context.Msg = sb.ToString()     OnRequest = 1 End Function</pre>

## 4.5.4.2 BaudRate 属性

获取或设置串口通信端口的波特率。

**语法**

C#	<a href="#">public int BaudRate { get; set; }</a>
	属性值 类型: <a href="#">System.Integer</a>
VB	Public Property BaudRate As Integer Get Set
	属性值

	类型: <code>System.Integer</code>
--	---------------------------------

**备注**

用户的串行驱动程序必须支持波特率。

**示例**

下面的示例在脚本的 `OnRequest` 方法中设置。

C#	<pre>public int OnRequest(BSCaseContext context) {     BSComStreamParameters comParams = context.CommParameters         as BSComStreamParameters;     comParams.BaudRate = 115200; // 波特率设置为 115200      return 1; }</pre>
VB	<pre>Public Function OnRequest(ByRef context As BSCaseContext) As Integer     Dim comParams As BSComStreamParameters     comParams = DirectCast(context.CommParameters, BSComStreamParameters)      comParams.BaudRate = 115200 ' 波特率设置为 115200     OnRequest = 1 End Function</pre>

## 4.5.4.3 DataBits 属性

获取或设置每个字节的标准数据位长度。

**语法**

C#	<pre>public int DataBits{ get; set; }</pre>
	属性值 类型: <code>System.Integer</code>
VB	<pre>Public Property DataBits As Integer     Get     Set</pre>
	属性值 类型: <code>System.Integer</code>

**备注**

此属性的值范围为 5 到 8。

**示例**

无

## 4.5.4.4 Parity 属性

获取或设置奇偶校验检查协议。



## 语法

C#	<code>public Parity Parity{ get; set; }</code>
	属性值 类型: System.IO.Ports.Parity Parity 值之一, 表示奇偶校验检查协议。默认值为 None。
VB	Public Property Parity As Parity Get Set
	属性值 类型: System.IO.Ports.Parity Parity 值之一, 表示奇偶校验检查协议。默认值为 None。

## 备注

如果在流的尾字节上出现奇偶校验错误, 将向输入缓冲区添加一个值为 126 的额外字节。

## 示例

无

## 4.5.4.5 StopBits 属性

获取或设置每个字节的标准停止位数。

## 语法

C#	<code>public StopBits StopBits{ get; set; }</code>
	属性值 类型: System.IO.Ports.StopBits StopBits 值之一。
VB	Public Property StopBits As StopBits Get Set
	属性值 类型: System.IO.Ports.StopBits StopBits 值之一。

## 备注

StopBits 的默认值为 One。

不支持 StopBits.None 选项。将 StopBits 属性设置为 None 将会引发 ArgumentOutOfRangeException。

## 示例

无

## 4.5.4.6 Handshake 属性

获取或设置串行端口数据传输的握手协议。

## 语法

C#	<code>public Handshake Handshake{ get; set; }</code>
----	--

	属性值 类型: System.IO.Ports.Handshake Handshake 值之一。默认值为 None。
VB	Public Property Handshake As Handshake Get Set
	属性值 类型: System.IO.Ports.Handshake Handshake 值之一。默认值为 None。

**备注****示例**

无

## 4.5.4.7 DtrEnable 属性

获取或设置一个值，该值在串行通信过程中启用数据终端就绪（DTR）信号。

**语法**

C#	<code>public bool DtrEnable{ get; set; }</code>
	属性值 类型: System.Boolean 如果为 true，则启用数据终端就绪（DTR）；否则为 false。默认值为 false。
VB	Public Property DtrEnable As Boolean Get Set
	属性值 类型: System.Boolean 如果为 true，则启用数据终端就绪（DTR）；否则为 false。默认值为 false。

**备注**

在 XON/XOFF 软件握手、请求发送/可以发送（RTS/CTS）硬件握手和调制解调器通信的过程中通常启用数据终端就绪（DTR）。

**示例**

无

## 4.5.4.8 RtsEnable 属性

获取或设置一个值，该值指示在串行通信中是否启用请求发送（RTS）信号。

**语法**

C#	<code>public bool RtsEnable{ get; set; }</code>
	属性值 类型: System.Boolean

	如果为 true, 则启用请求发送 (RTS); 否则为 false。默认值为 false。
VB	Public Property RtsEnable As Boolean Get Set
	属性值 类型: System.Boolean 如果为 true, 则启用请求发送 (RTS); 否则为 false。默认值为 false。

**备注**

请求发送 (RTS) 信号通常用在请求发送/可以发送 (RTS/CTS) 硬件握手中。

**示例**

无

4.5.5 网口参数 *BSNetStreamParameters* 类

## 4.5.5.1 Socket 属性

获取网口通信端口。

**语法**

C#	<code>public Socket Socket { get; }</code>
	属性值 类型: <code>System.Net.Sockets.Socket</code>
VB	Public Property Port As Socket Get Set
	属性值 类型: <code>System.Net.Sockets.Socket</code>

**备注**

## 4.5.6 脚本中使用插件

本软件支持基于 Microsoft .NET Framework 的托管代码组件作为插件扩展脚本的功能。关于插件及其编写和部署方式, 参见 5。

在脚本中使用插件, 非常方便, 直接在脚本的函数中调用插件所提供的公共服务即可。

**示例**

下面的示例在脚本的 OnRequest 方法中调用 ParallelPort.dll 插件 (源代码位于软件安装目录下 Samples\Plugins\ParallelPort 中)。

C#	<pre>public int OnRequest(BSCaseContext context) {     int val = Geshe.Utils.ParallelPort.Read(0x378);     context.Msg = "Read ParallelPort 0x378. Value=" + val.ToString();     return 1; }</pre>
----	--

VB	<pre>Public Function OnRequest(ByRef context As BSCaseContext) As Integer     Dim val As Integer     val = Geshe.Utils.ParallelPort.Read(&amp;H378)     context.Msg = "Read ParallelPort 0x378. Value=" + val.ToString()     OnRequest = 1 End Function</pre>
----	---

注：ParallelPort.dll 插件第一次访问需要往系统 System32 目录安装并口驱动，这时候需要管理员权限才能执行。

## 5. 插件

本软件在运行时动态识别所有位于 Plugins 目录下的基于 Microsoft .NET Framework 的托管代码组件，并把它们作为插件在脚本中按需调用它们的公共服务。

### 5.1 托管代码与非托管代码

Microsoft .NET Framework 是支持生成和运行下一代应用程序和网络服务的 Windows 组件。.NET Framework 具有两个主要组件：公共语言运行库（Common Language Runtime）和 .NET Framework 类库（Framework Class Library）。公共语言运行库是 .NET Framework 的基础，可以将运行库看作一个在运行时管理代码的代理，它提供内存管理、线程管理和远程处理等核心服务，并且还强制实施严格的类型安全以及可提高安全性和可靠性的其他形式的代码准确性。事实上，代码管理的概念是运行库的基本原则，以运行库为目标的代码称为托管代码（Managed code），而不以运行库为目标的代码称为非托管代码（Unmanaged code）。.NET Framework 的另一个主要组件是类库，它是一个综合性的面向对象的 reusable 类型集合，可以使用它开发多种应用程序和网络服务程序。

托管代码与非托管代码的区别：

- 托管代码是一种中间语言，以运行库为目标，运行在 CLR 上；非托管代码不以运行库为目标，被编译为机器码，运行在机器上。
- 托管代码独立于平台和语言，能更好的实现不同语言平台之间的兼容；非托管代码依赖于平台和语言。
- 托管代码可享受 CLR 提供的服务，如安全检测、垃圾回收等，不需要自己完成这些操作；非托管代码需要自己提供安全检测、垃圾回收等操作。

托管代码可以使用 20 多种支持 Microsoft .NET Framework 的高级语言编写，它们包括 C#、J#、Microsoft Visual Basic .NET、Microsoft JScript .NET，以及 Visual C++ .NET 等。

### 5.2 编写插件

所有基于 Microsoft .NET Framework 的托管代码组件都可以被本软件识别，并作为插件在脚本中使用它们向外提供的功能，对插件的接口协议无任何要求。

编写插件的过程和编写普通基于 Microsoft .NET Framework 的托管代码类库和应用程序一样，只要确保所要提供的功能是公开的（public）即可。

### 5.3 使用托管代码的第三方库

直接把基于 Microsoft .NET Framework 托管代码的第三方库，以及其依赖的库拷贝到本软件的 Plugins 目录中即可。

### 5.4 使用非托管代码的第三方库

非托管代码，如 C/C++ 编写的驱动程序库，不能被本软件识别和直接使用，需要通过一个托管代码类库包装，以间接的方式使用。

首先，编写一个基于 Microsoft .NET Framework 的托管代码类库，在该类库中使用通过 .NET Framework 类库的 System.Runtime.InteropServices 命名空间中提供的各种各样支持 COM Interop 及平台调用服务的成员，与非托管代码进行交互操作。

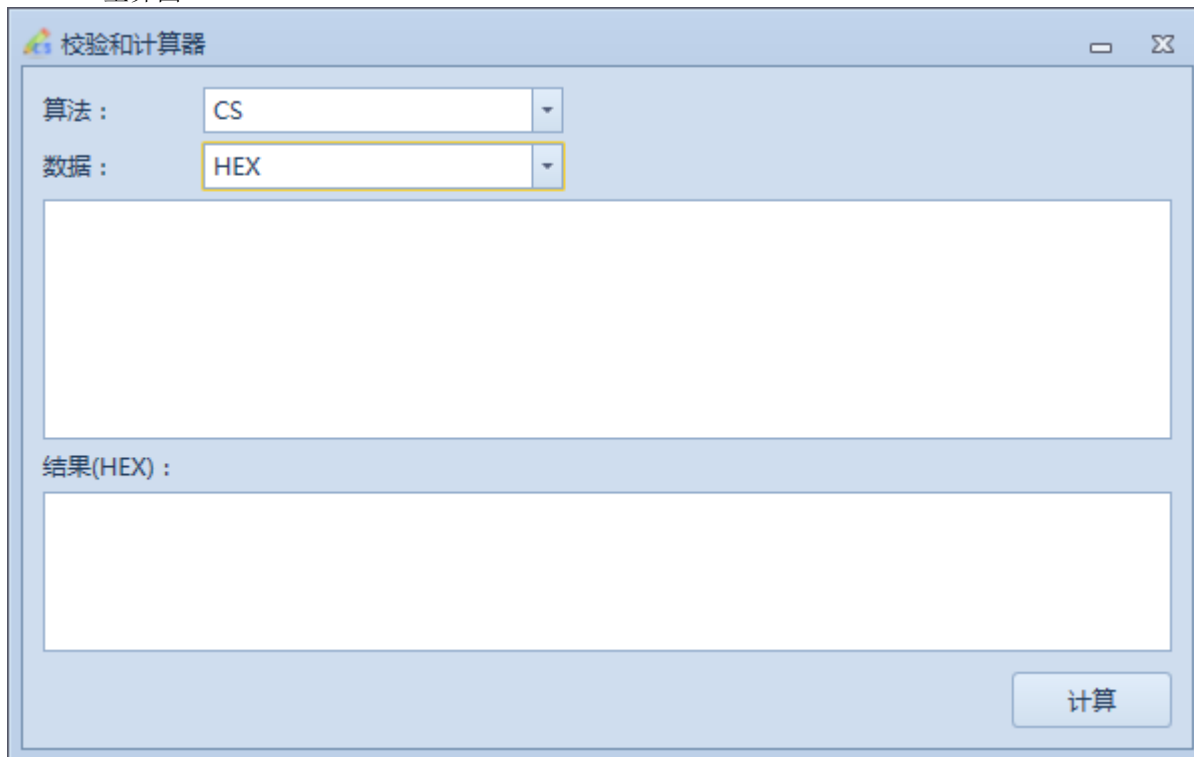
最后，把包装好的托管类库以及引用的非托管代码库拷贝到本软件的 Plugins 目录中，脚本便可以通过访问包装类库间接执行非托管代码库。

## 6. 工具箱

### 6.1 校验和计算器

校验和计算器是一款支持 CS、BCC、LRC 校验和计算器计算，数据源可以是 HEX 数据字符串、字符串（字符编码方式有设置中的字符编码方式决定）以及文件。

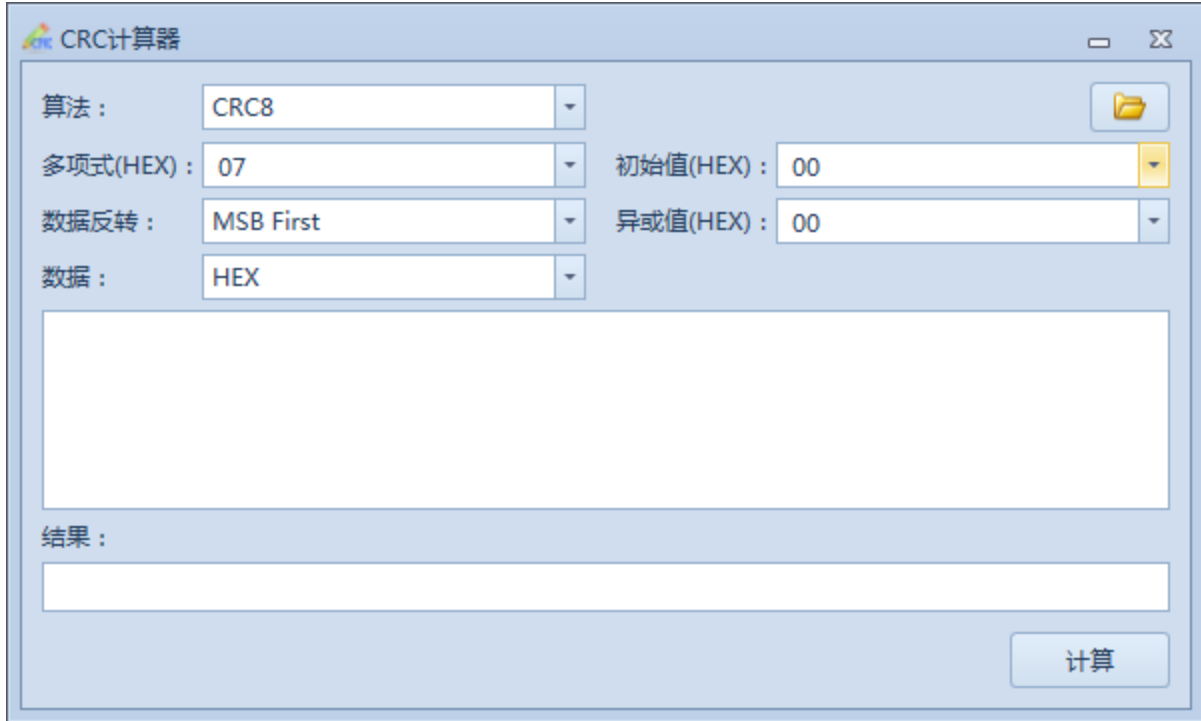
主界面



### 6.2 CRC 计算器

CRC 计算器是一款通用的循环冗余校验码（Cyclic Redundancy Check）计算工具。支持 CRC8、CRC16 和 CRC32 算法，可以自定义多项式、初始值、数据反转以及结果异或值，计算的数据源可以是 HEX 数据字符串、字符串（字符编码方式有设置中的字符编码方式决定）以及文件。支持常用的 CRC 标准算法。

主界面



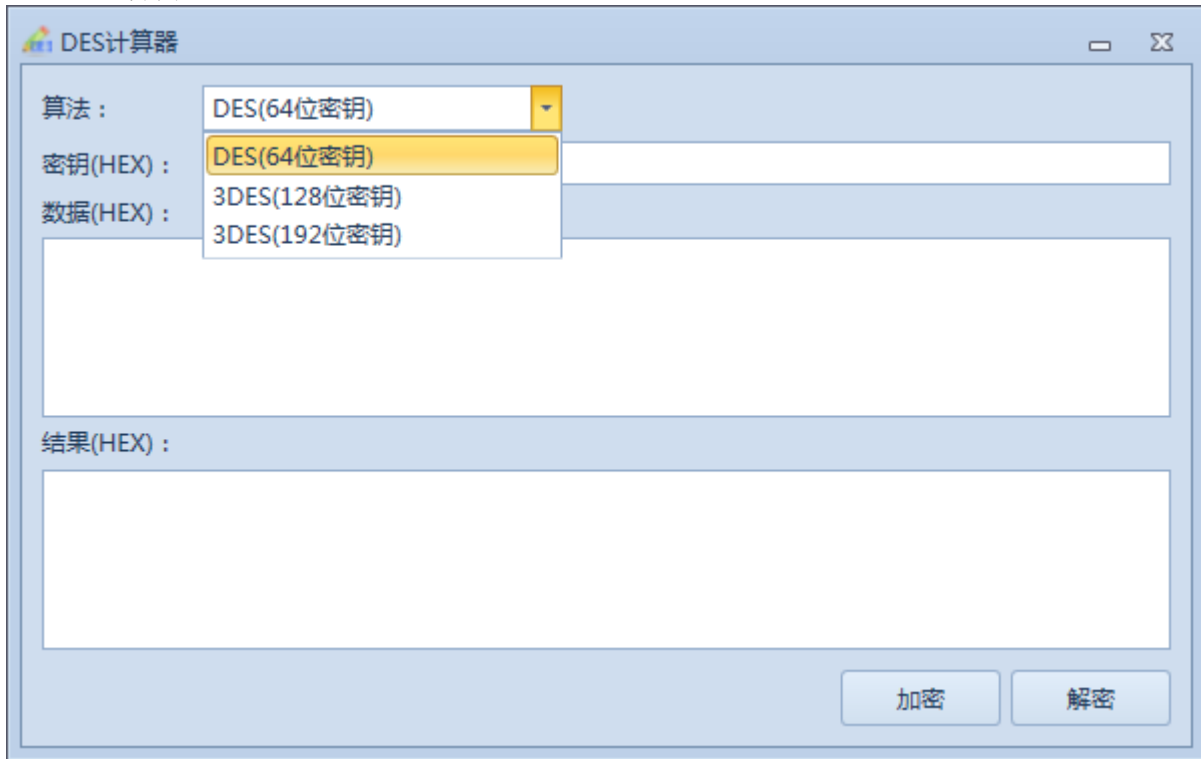
常用的 CRC 标准算法选择界面



### 6.3 DES 计算器

DES 计算器是一款支持 DES (64 位密钥)、3DES (128 位密钥)、3DES (192 位密钥) 的 DES 计算工具。计算的数据源是 HEX 数据字符串，数据不足用 0 填充。

主界面

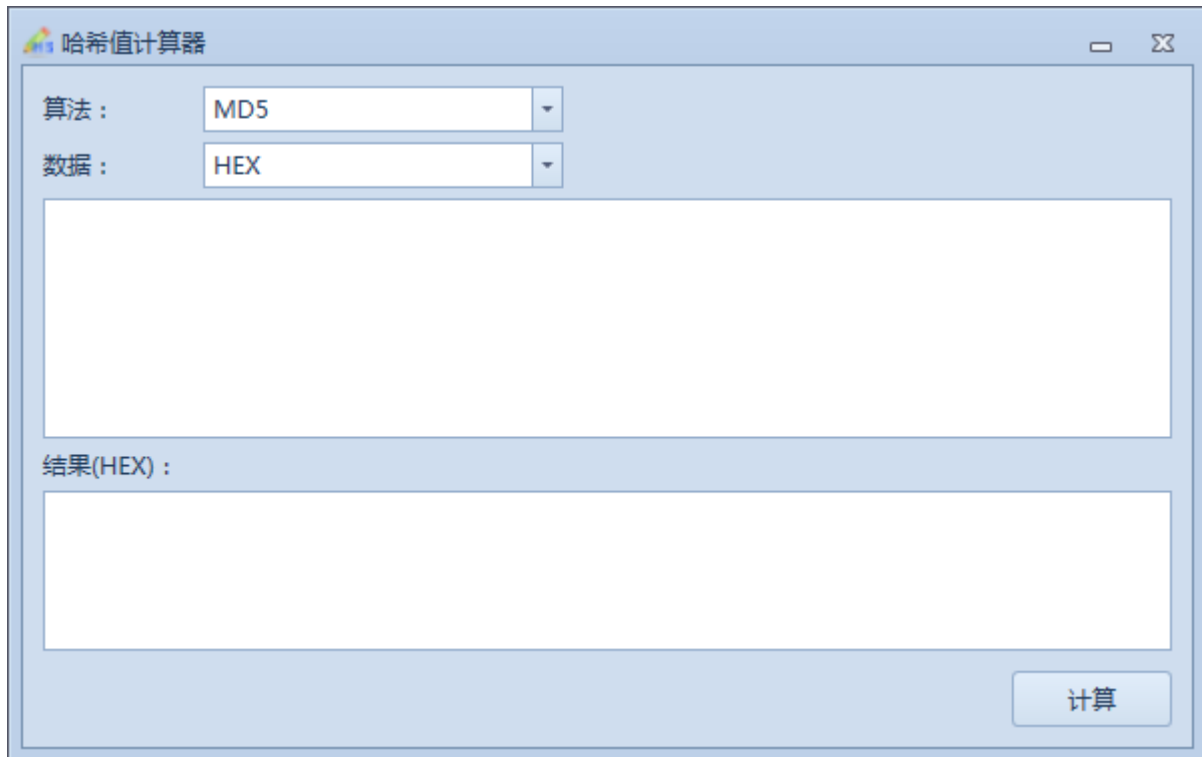


#### 6.4 哈希值计算器

哈希值计算器是一款支持 MD5、SHA1、SHA256、SHA384、SHA512 哈希算法的计算工具。计算的数据源可以是 HEX 数据字符串、字符串（字符编码方式有设置中的字符编码方式决定）以及文件。

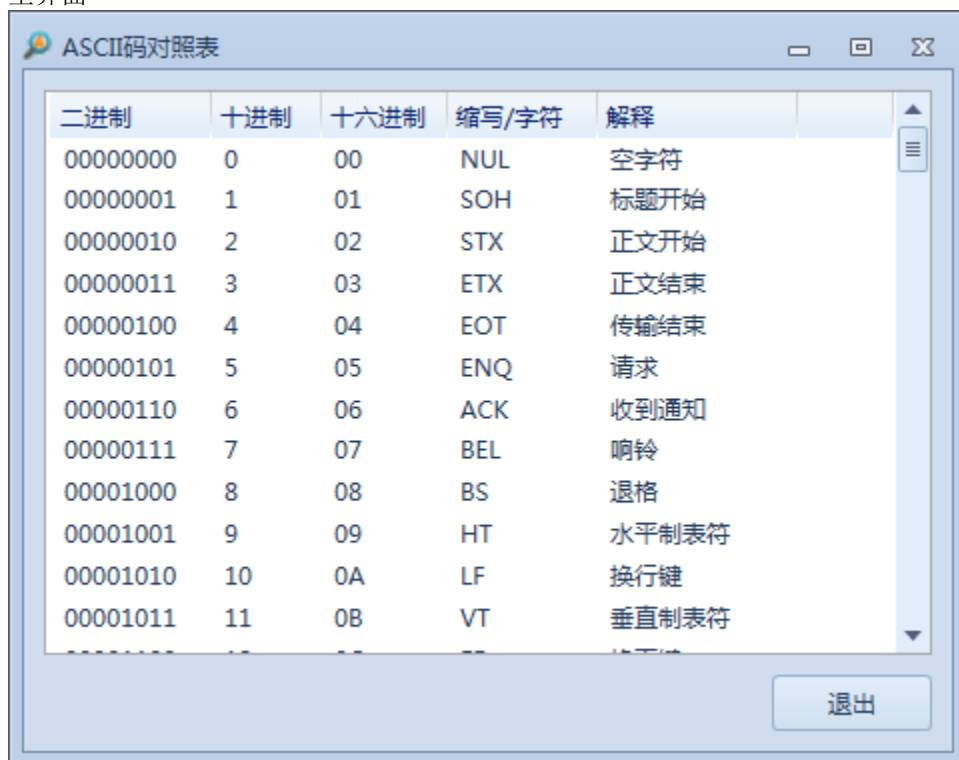
主界面



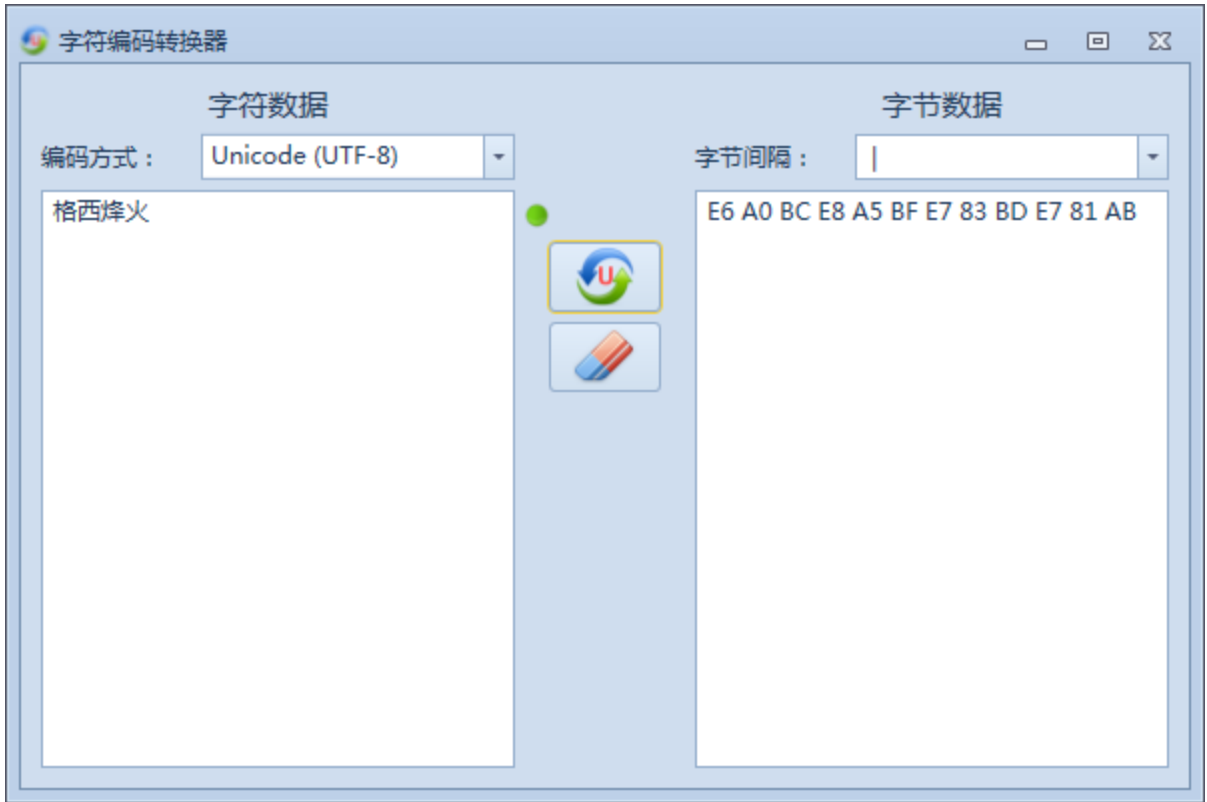


## 6.5 ASCII 码对照表

主界面



6.6 字符编码转换器  
主界面



## 7. 应用技巧

### 7.1 分类组织协议激励项目的协议项

对于一个包含较多协议项的协议激励项目来说，可以按照应用的需求有组织地分类管理，组织成一个树型结构，管理上更加有条理，查找也更加方便。

例如，在一个通信协议测试中，通常把通信系统支持的协议组织成一个基本协议集，作为一个公共的协议库使用，然后再针对不同的测试用例创建特定的组合测试集。



### 7.2 运行多个软件实例

本软件支持多个软件实例同时运行，能够满足不同的使用环境和用户需求。

主要的应用场景：

- 并行测试。只要计算机配置和性能足够，可以同时运行多个软件实例，每一个软件实例检测一个通信接口，同时检测多个外部设备，最大限度降低测试成本。
- 复制和粘贴激励项。本软件不仅支持软件实例内的复制和粘贴功能，还支持从一个软件实例到另一个软件实例的复制和粘贴功能，可以方便进行激励项的筛选和重构。

## 8. FAQ

### 8.1 进行“反馈”或者“注册软件”操作时，为什么出现 Unknown error (0x80041002) 错误？

答：在无法和外部网络服务器联系情况下进行“反馈”和“注册软件”操作，本软件会尝试启动 Email 程序进行 Email 发送，如果本机没有 Email 客户端程序，则出现此错误。

### 8.2 进行协议激励时，从动方已经发出正确的帧，为什么主动激励方却返回失败？

答：一种原因是主动激励协议的失效超时设计不合理，可以适当延长该协议项的失效超时。