Zmotion HMI 编程手册

Version2.0.0

第一章 ZHMI 编程入门

ZHMI 是 ZMotion 运动控制器所使用的组态设计，使用前需要确认控制器是否支持 ZHMI 功能。

编写和调试 ZHM 程序需要 ZDevelop V2.5 以上版本软件，ZMC 运动控制器或仿真器 V4.0 以上版本固件，PC 在线命令发送需要 zmotion.dll 动态库。

ZDevelopV2.5 以上版本软件支持 Basic 程序、PLC 程序、HMI 组态同时使用，可以使用程序在显示屏上动态绘图，建议下载最新版本使用。

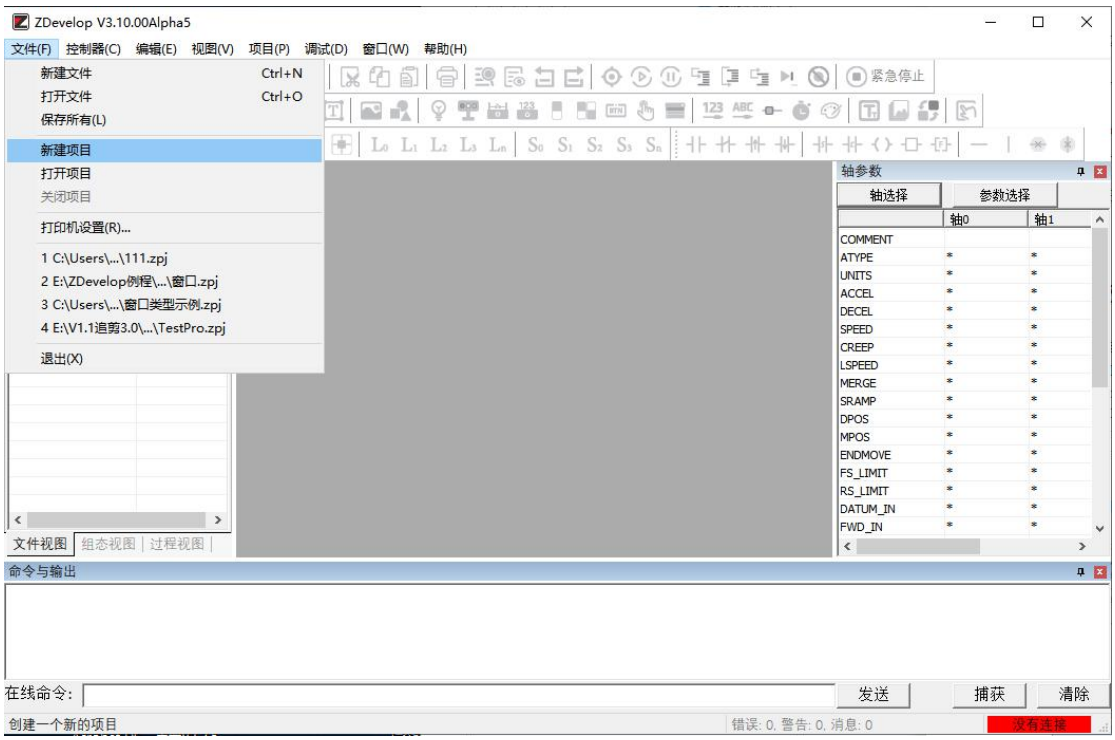
1.1.ZDevelop 编程特点

1.1.1. HMI 组态程序开发

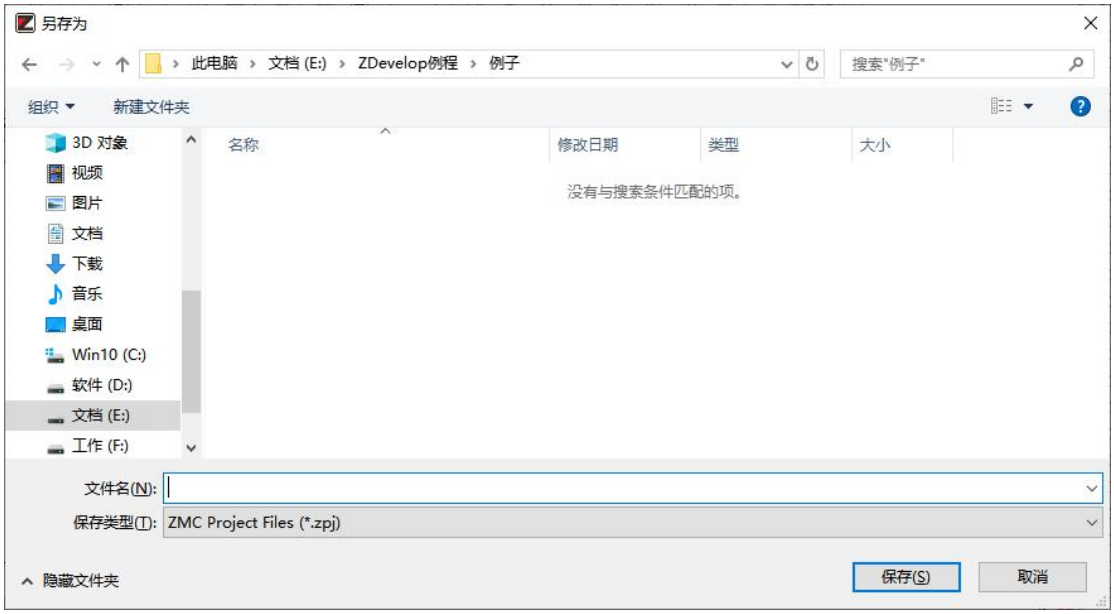
在电脑里新建一个文件夹用来保存即将要建立的工程。打开 ZDevelop 编程软件，当前说明例程的 ZDevelop 软件版本为 V3.10.00，更新软件版本请前往正运动官方网站下载，官方网站上还可以下载触摸屏例程，网址：www.zmotion.com.cn。

ZDevelop 编程开发流程：

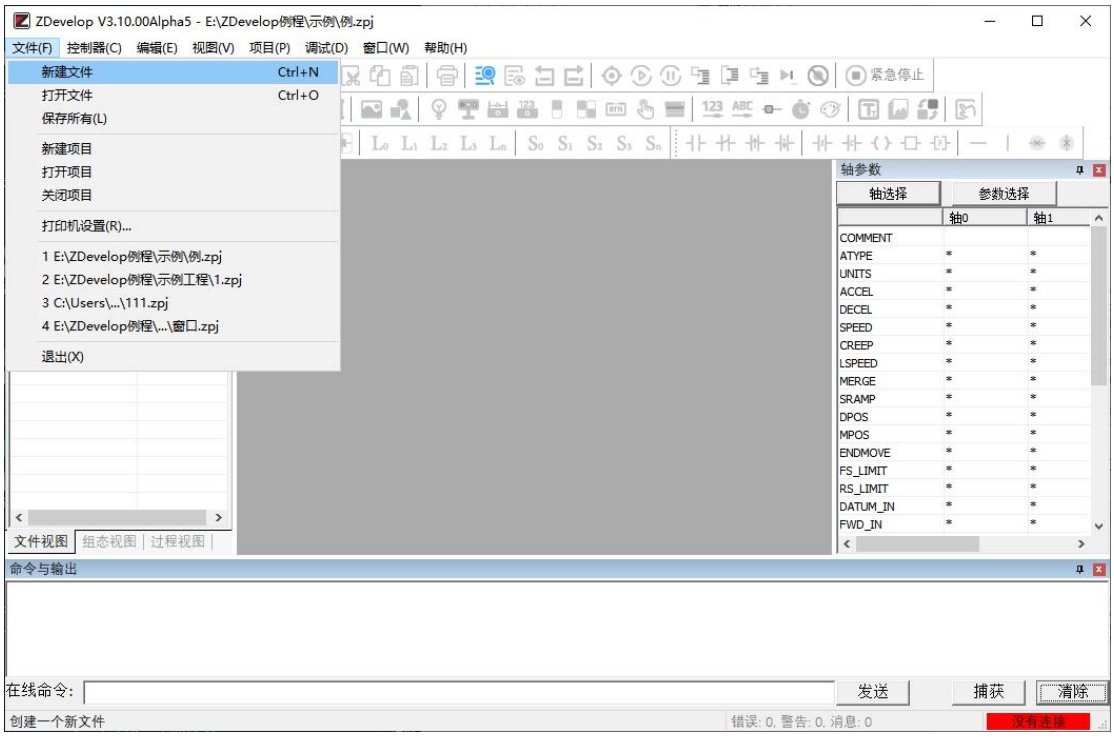
1. 新建项目：菜单栏“文件”-“新建项目”。



点击“新建项目”后弹出“另存为”界面，选择一个文件夹打开，输入文件名后保存项目，后缀为“.zpj”。

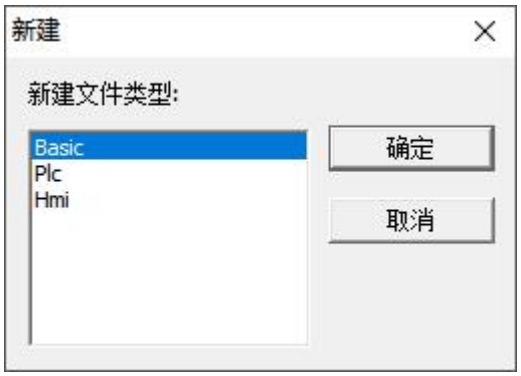


2. 新建文件：菜单栏“文件”-“新建文件”。

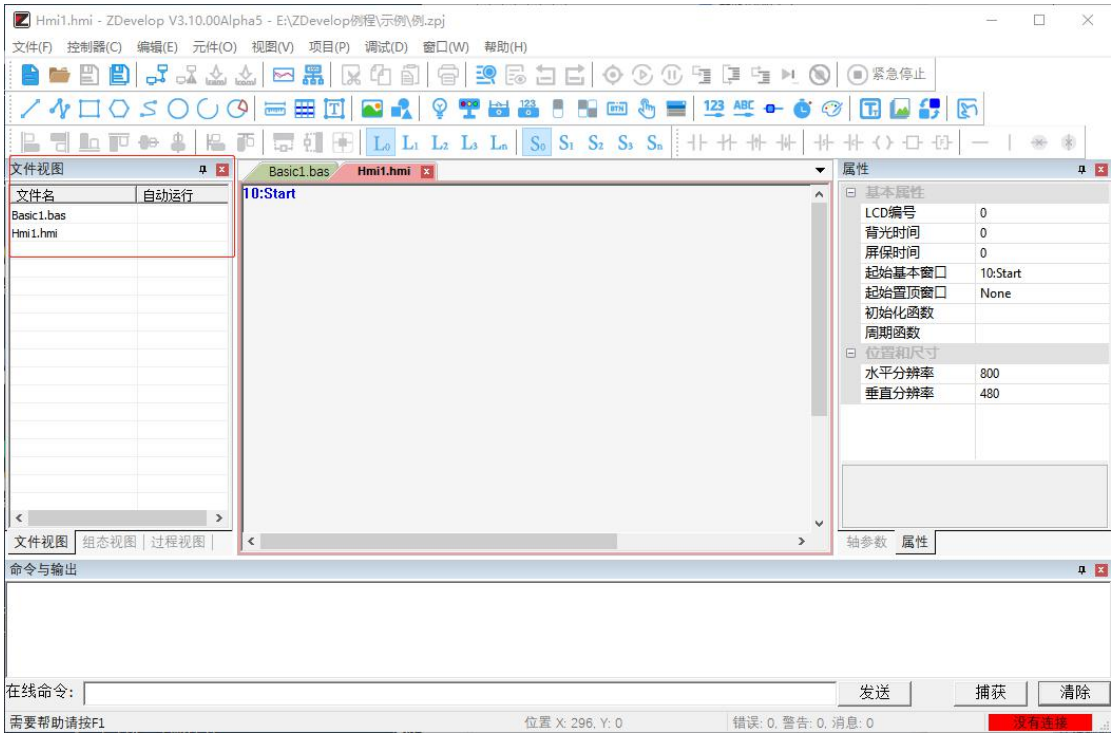


点击“新建文件”后，出现下图弹窗，Hmi 一般和 Basic 混合编程，用 Basic 函数编写 hmi 元件要实现的功能，分别新建一个 Basic 文件和一个 hmi 文件。

Basic/Plc/Hmi 分别针对 3 种不同类型的文件，表示 ZDevelop 支持的三种编程方式，基础连接使用步骤相同，支持 Basic/Plc/Hmi 混合编程。

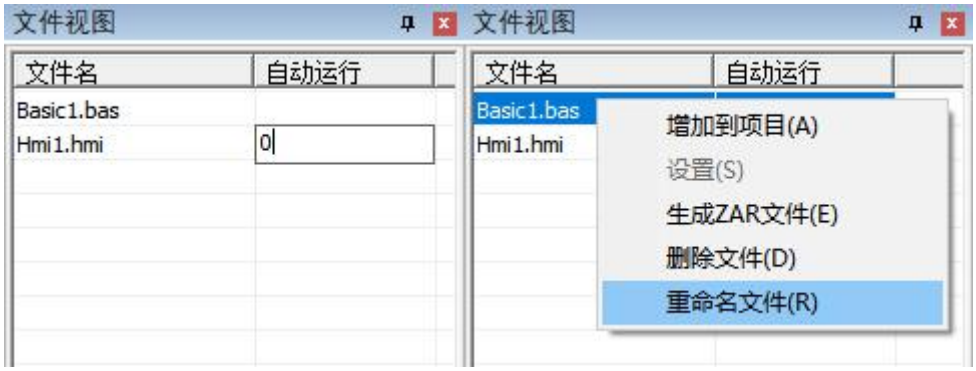


保存文件：确认后新建的文件会自动加入到项目“文件视图”中，如下图。在程序编辑窗口写好程序后，保存文件，新建的文件会自动保存到项目 zpj 所在的文件夹下。



3. 设置文件自动运行：双击文件右边自动运行的位置，输入任务号“0”。

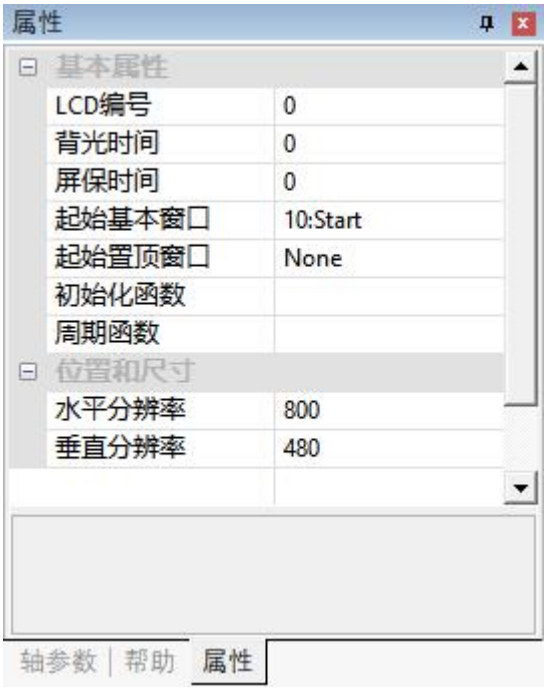
文件名称可重新自定义，先关闭要重命名的文件。然后在文件处点击鼠标“右键”-“重命名文件”修改。



4. 组态程序编辑：在编辑组态程序之前，首先要打开“hmi 系统设置”窗口。

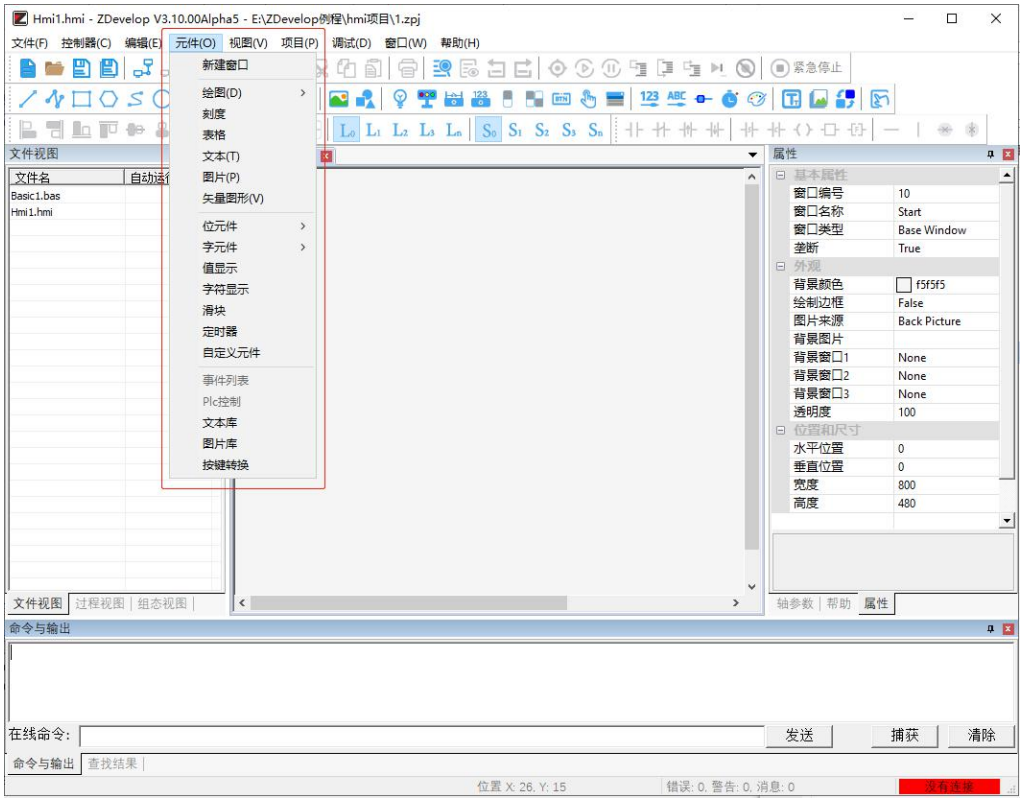
先切换到 HMI 编程窗口，菜单栏“编辑”-“hmi 系统设置”打开如下窗口，根据组态程序要应用的示教盒的尺寸，设置好水平分辨率和垂直分辨率。初始化函数和周期函数选择 Basic 里编写好的 GLOBAL 全局定义的 SUB 子函数，编辑好 Basic 函数后在此窗口添加。

此窗口的详情参见“HMI 系统设置”章节。



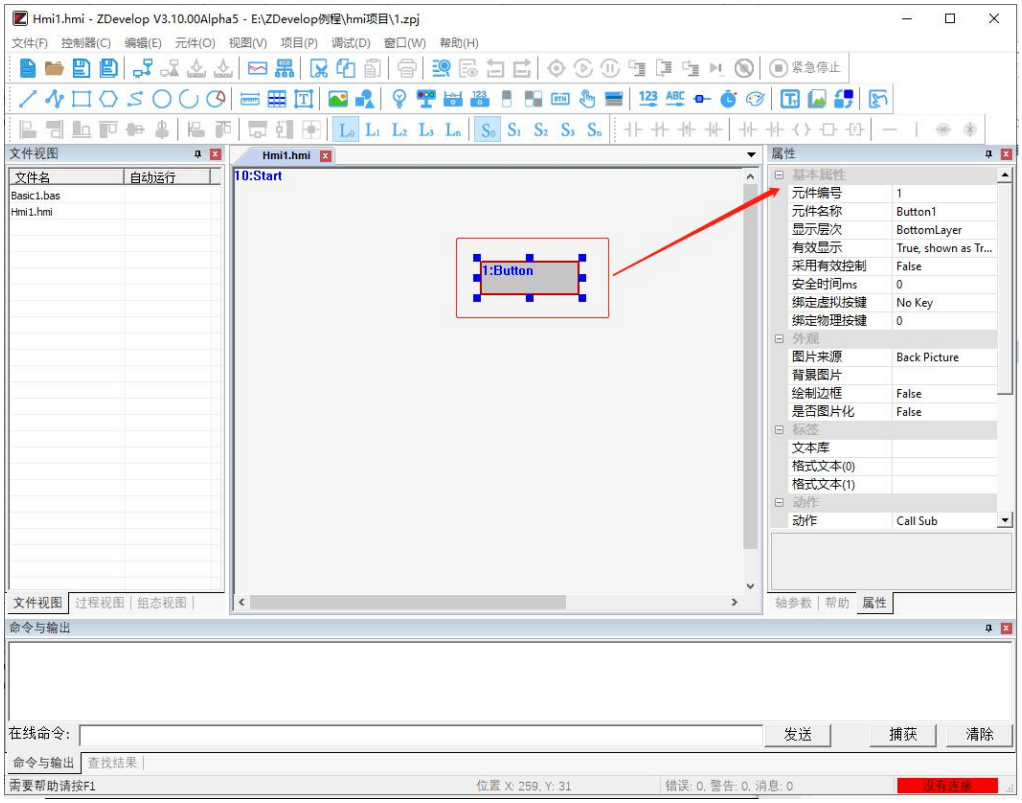
基础设置完成，新建组态窗口，添加组态元件，组态元件在组态窗口上显示。

Hmi 编程所需的窗口和元件在菜单栏“元件”里选择，建立 hmi 文件后，自动新建三个软键盘窗口和一个起始基本窗口 10:Start。窗口和元件的使用说明参见第二章和第三章。



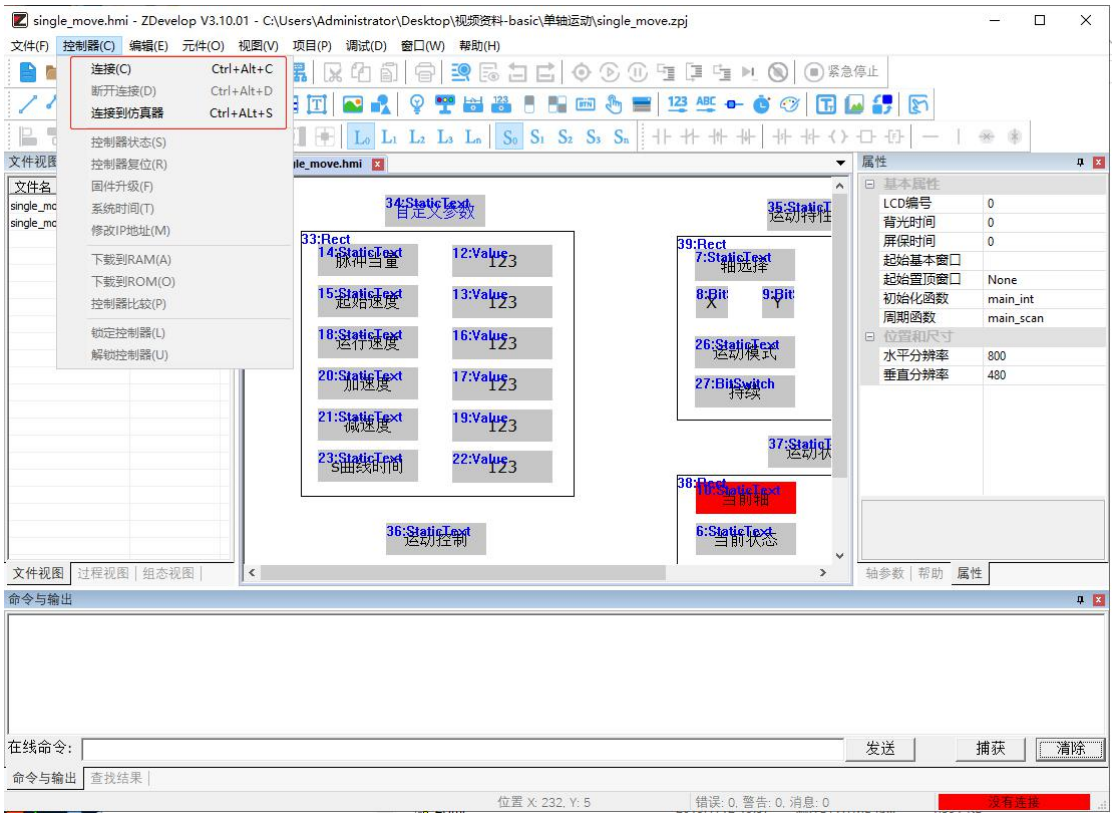
在“元件”菜单栏添加元件后，将元件放置于组态窗口尺寸范围内，打开元件“属性”设置元件相关参数，如下图。

直接拖拽元件选择放置的位置和元件大小，或在元件“属性”的“位置和尺寸”栏设置。



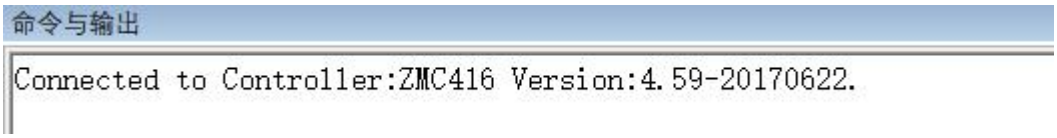
5. 连接控制器/仿真器。

编辑好 Basic 程序和 hmi 程序，点击“控制器”-“连接”或“连接到仿真器”。点击“连接”弹出“连接到控制器”窗口，连接方法参见下节。

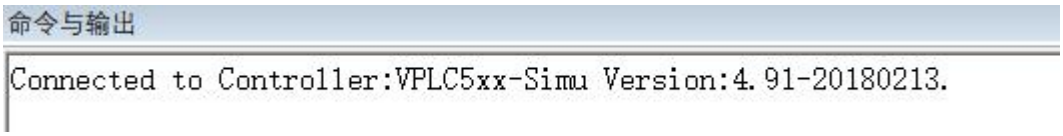


连接是否成功输出窗口会打印出信息提示。

例如：成功连接到 ZMC416：

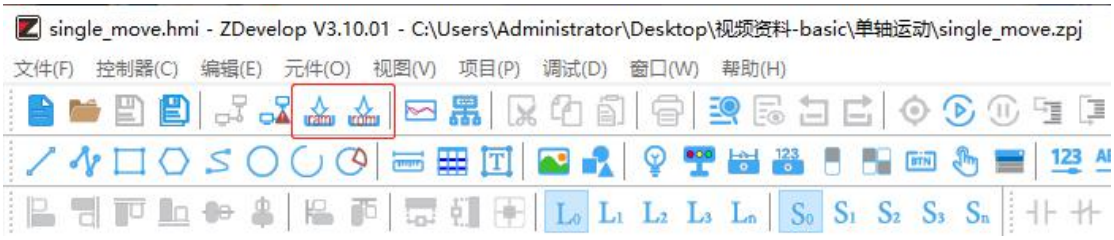


成功连接到仿真器：



若连接失败，也会有相应的信息提示。

6. 下载运行程序：下载程序到控制器/仿真器运行。



下载程序时可选择“下载到 RAM”或“下载到 ROM”，下载成功“命令与输出”窗口打印下载成功提示，同时程序下载到控制器并自动运行。

RAM 下载掉电后程序不保存，ROM 下载掉电后程序保存。下载到 ROM 的程序下次连接上控制器之后程序会自动按照任务号运行。

触摸屏程序必须下载到 ROM。

注意事项：

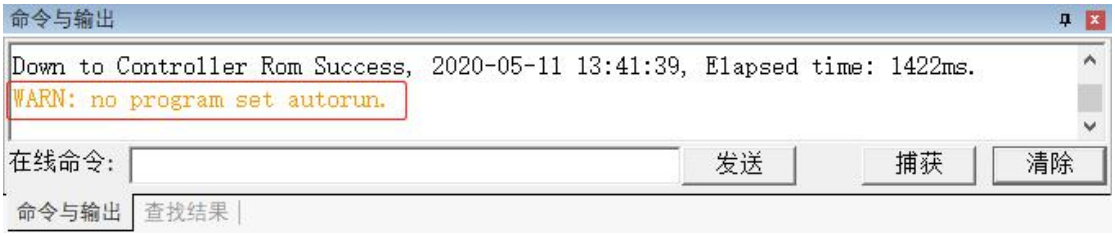
ZMC00x 系列控制器不支持下载到 RAM。

不建立项目的时候，只有文件无法下载到控制器。

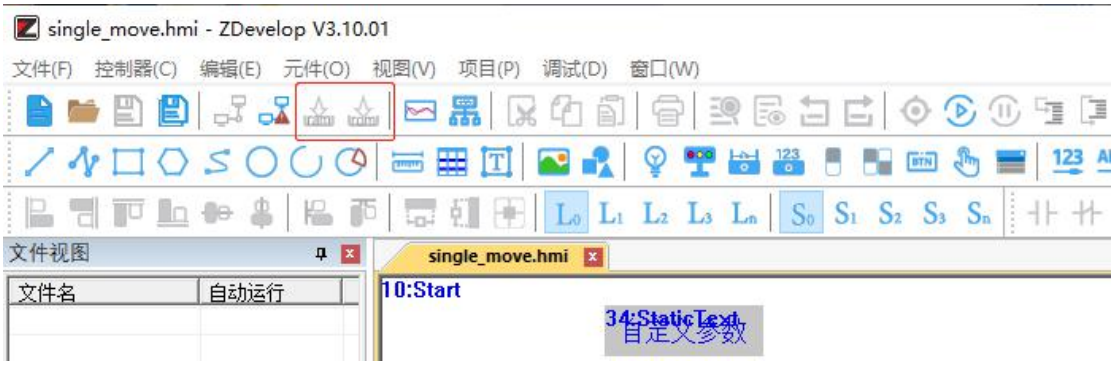
自动运行的数字 0 表示任务编号 0，以任务 0 运行程序，任务编号不具备优先级。

若整个工程项目内的文件都不设置任务编号，下载到控制器时，系统提示如下信息。

WARN: no program set autorun.



打开工程项目时，选择打开项目 zpj 文件，若只打开其中的 bas/plc/hmi 文件，程序无法下载到控制器。如下图，RAM/ROM 下载图标均为灰色。

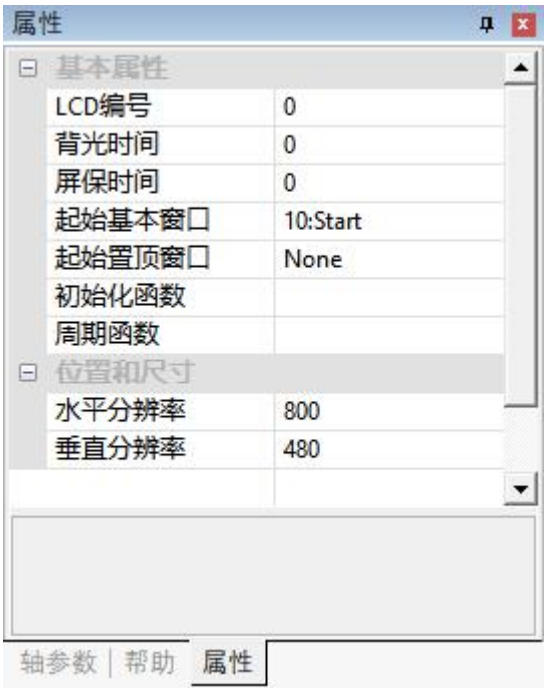


1.1.2. HMI 系统设置

在编辑 hmi 组态程序之前，首先要打开“hmi 系统设置”窗口。

先切换到 HMI 编程窗口，菜单栏“编辑”-“hmi 系统设置”打开如下窗口，根据组态程序要应用的示教盒的尺寸，设置好水平分辨率和垂直分辨率，分辨率需要提前确定，其他参数可以后续再设置。

根据需求选择是否需要设置初始化函数和周期函数，选择 Basic 里编写好的 GLOBAL 全局定义的 SUB 子函数。



LCD 编号：LCD 屏幕的编号。

背光时间：示教盒实际背光时间，ms 钟单位。背光和屏保时间单独设置时也起作用，同时设置时以屏保时间为准。不使用时设 0。

屏保时间：示教盒实际屏保时间，ms 钟单位。

起始基本窗口：选择上电初始显示的 base 类型窗口号，起始置顶窗口选择 top 类型的窗口，窗口类型查看第二章。

初始化函数：上电后只调用一次的函数，在 Basic 文件中定义，函数的定义必须是全局的(GLOBAL)。

周期函数：上电后只周期不断扫描的函数，在 Basic 文件中定义，函数的定义必须是全局的(GLOBAL)。

分辨率：根据显示屏尺寸设置，常见 7 寸屏 800*480。

1.1.3. 网络显示屏

ZHMI 支持通过以太网把电脑或其它触摸屏作为显示屏使用，也可以使用自身的显示屏（必须控制器带有显示屏）。

控制器支持多个显示屏时，通过设置组态文件属性，“HMI 系统设置”窗口选择使用的显示屏编号。

控制器支持的显示屏个数和最大分辨率在连接了控制器之后，在线命令发送?*max 打印信息，查看 max_hmi 参数。

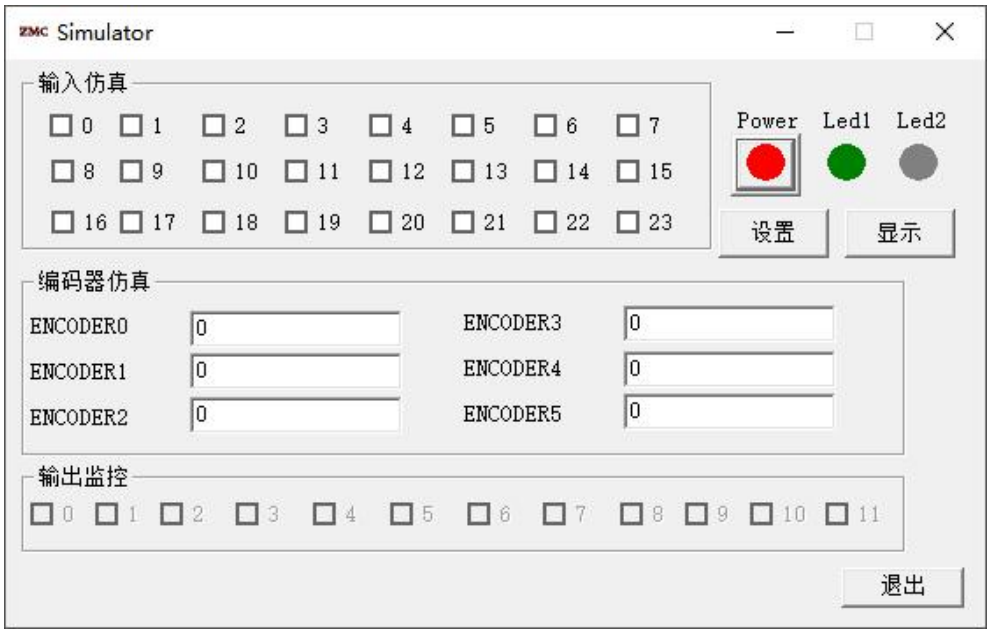
某型号控制器打印 max_hmi 结果：

max_hmi:2, x:1024 y:800　支持 2 个远端 HMI，支持的最大尺寸为 1024*800

1.1.4. HMI 仿真运行

程序下载到仿真器后，点击仿真器的“显示”，即可运行 HMI 界面，显示 xplc screen 组态界面进行仿真。

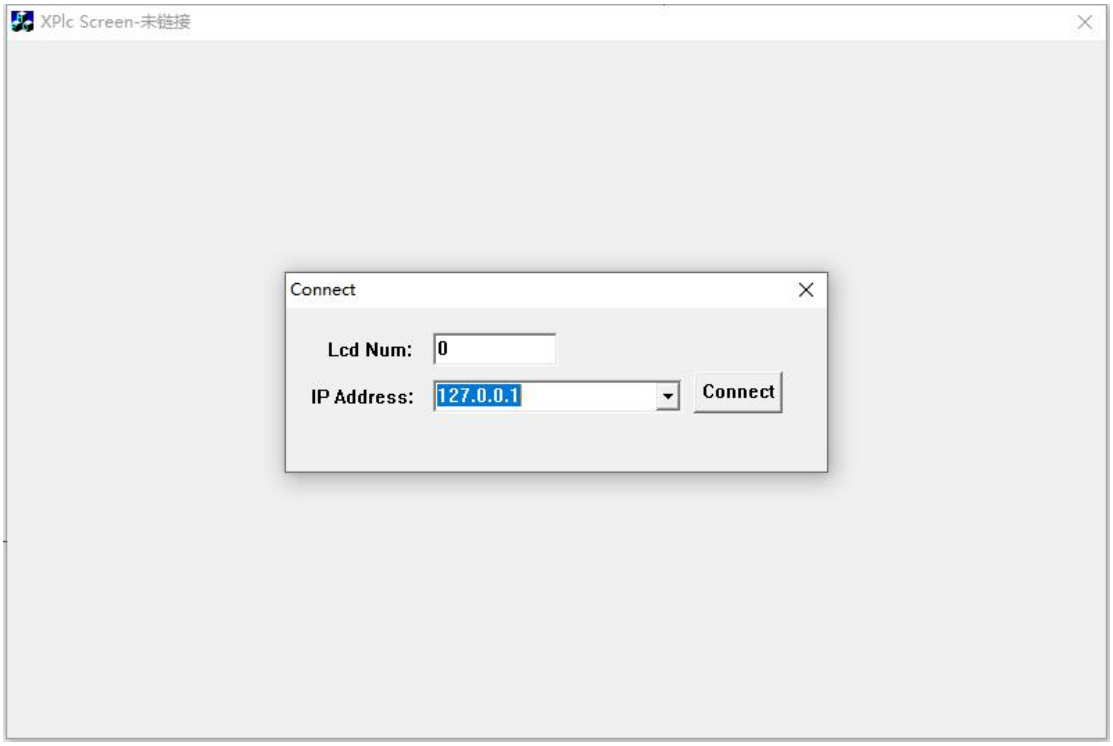
在不连接控制器和示教盒的时候，可以连接到仿真器之后调试。



1.1.5. Xplcterm 运行

在没有触摸屏的时候，使用 Xplcterm 仿真触摸屏的运行，可将 PC 作为一台人机界面，可以连接仿真器或控制器。

程序下载到仿真器或控制器后。打开 Xplcterm，点击工具栏的快捷按钮  ，设置连接的显示屏编号和 IP 地址，默认显示的是仿真器的 IP，IP 地址填入后点击 Connect 即可使用。



1.1.6. 适用的控制器

支持 HMI 编程方式的控制器型号如下，部分控制器需要升级固件才能支持，详情请联系厂家。

控制器系列	支持的控制器型号
ZMC0XX	无
ZMC1XX	无
ZMC2XX	无
ZMC3XX	ZMC306 /ZMC306X / ZMC308 / ZMC316
ZMC4XX	ZMC406 / ZMC412 / ZMC432 / ZMC464 /

	ZMC432N/ZMC460N
ECI1000 系列	无
ECI2000 系列	无
ECI3000 系列	无
XPLC	所有型号均支持

1.1.7. 适用的手持盒

ZHD 系列手持盒后缀带 X 的表示支持 HMI 组态编程方式，并且触摸屏程序可下载到控制器上（不需要单独下载到触摸屏），再将触摸屏连接到控制器即可通讯。

型号	ZHD300X	ZHD400X
分辨率	480*272	800*480
按键	47 个	18 个
USB 口	1 个	1 个
急停开关	1 个	1 个
触摸屏	支持	支持

1.1.8. HMI 任务

要运行 HMI 文件就要给 HMI 文件设置自动运行任务号，每个显示屏最多允许一个 HMI 文件运行，若两个 HMI 文件同时运行，会报错。

HMI 文件需要占用一个自动运行任务号。Basic 文件可根据需求选择是否设置自动运行任务号。

HMI 通常情况下要和 Basic 混合编程，HMI 元件调用 Basic 的函数或寄存器，HMI 也可以和 PLC 混合编程。

HMI 任务不是实时的，需要实时性高的场合请使用独立的其他任务。

1.2.常见问题

当程序运动出错后，ZDevelop 软件“命令与输出”窗口会显示出出错信息，如果出错信息没有看到，可以通过命令行输入?*task 或“故障诊断”窗口再次查看出错信息，双击出错信息可以自动切换到程序出错位置。

问题	可能原因
无法正常显示 HMI 界面	分辨率设置错误，请按照硬件要求设置分辨率
自定义元件无法刷新显示	在刷新函数中，没用使用 SET_REDRAW 指令
元件调用函数失败	函数定义必须是 GLOBAL 类型
窗口操作失败	窗口操作类型与窗口属性类型不一致。
显示屏不亮，或亮度不够 显示屏不亮，或亮度不够	检查控制器的供电：USB 供电必须用质量很好的线并保证电脑的 USB 口供电足够， 否则请使用串口头的 24V 供电；
通讯不上	检查网线。

第二章 组态窗口

2.1.窗口概述

2.1.1. 窗口作用

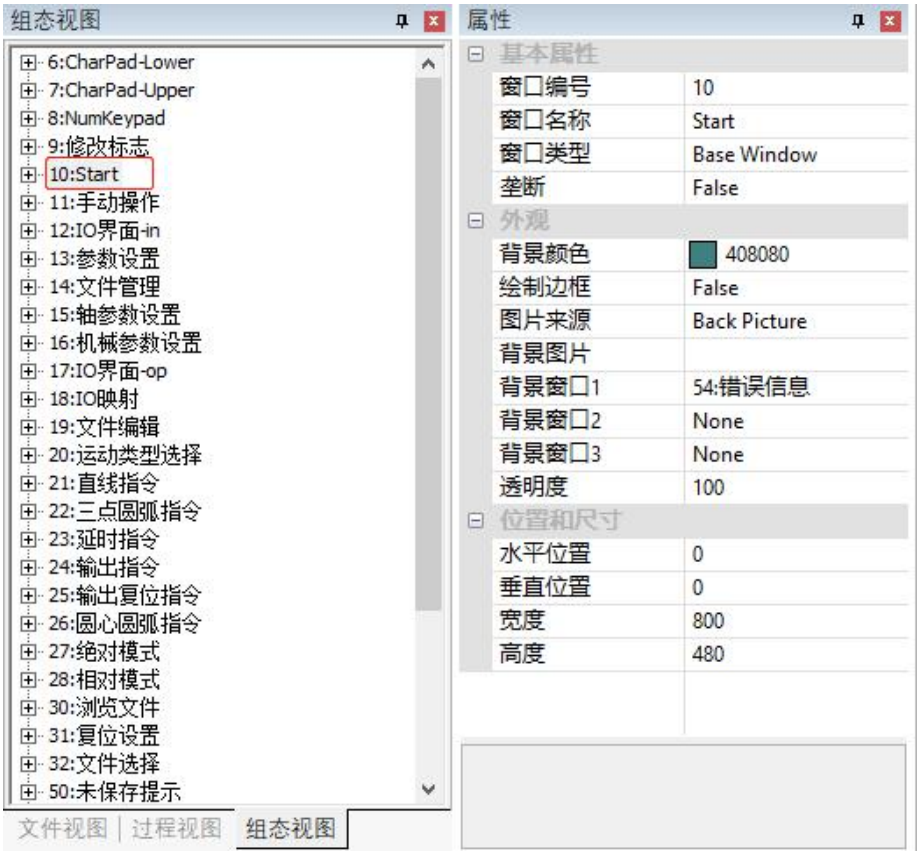
窗口是构成触摸屏画面的基本元素，也是一个重要的元素。有了窗口后，画面上的各个元件、图形、文字等信息才可以显示在触摸屏上。一般的工程文件中，包含多个窗口，所以一个功能需要建立多个窗口。

由于基本窗口的尺寸大小（分辨率）必须与触摸屏显示屏幕的尺寸相同，所以其分辨率设置也必须与所使用的触摸屏分辨率一致。

组态元件超出窗口尺寸时，该元件也能正常触发。

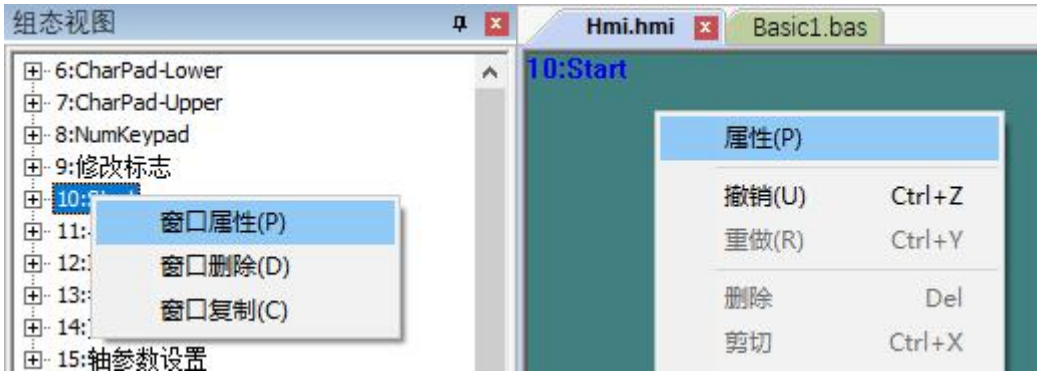
2.1.2. 窗口属性

组态视图能显示全部窗口，窗口的属性在“属性”窗口修改。



名称	功能	说明
窗口编号	当前窗口的编号	同一项目下窗口编号不能重复
窗口名称	当前窗口的名称	/
窗口类型	可选 5 种窗口类型	参见“窗口类型”说明
垄断	选择是否垄断	垄断后不能操作窗口下层的元件
背景颜色	选择窗口背景颜色	/
绘制边框	选择是否绘制边框	选择 TRUE 之后，可选择边框颜色
图片来源	从背景图片库或背景图片中选择	先添加图片才能选择
背景图片	从图片文件中选择背景图片	选择使用图片库
背景窗口	设置背景窗口	/
透明度	背景透明度	/
水平位置	窗口显示的左上角 X 坐标	不要超出水平分辨率
垂直位置	窗口显示的左上角 Y 坐标	不要超出垂直分辨率
宽度	当前窗口的显示宽度	/
高度	当前窗口的显示高度	/

打开属性窗口的方法：组态视图选中要打开的窗口，单击右键，选择“窗口属性”打开，如左图；或在组态文件编辑窗口，单击右键，打开“属性”。



2.2.窗口操作

2.2.1. 窗口建立

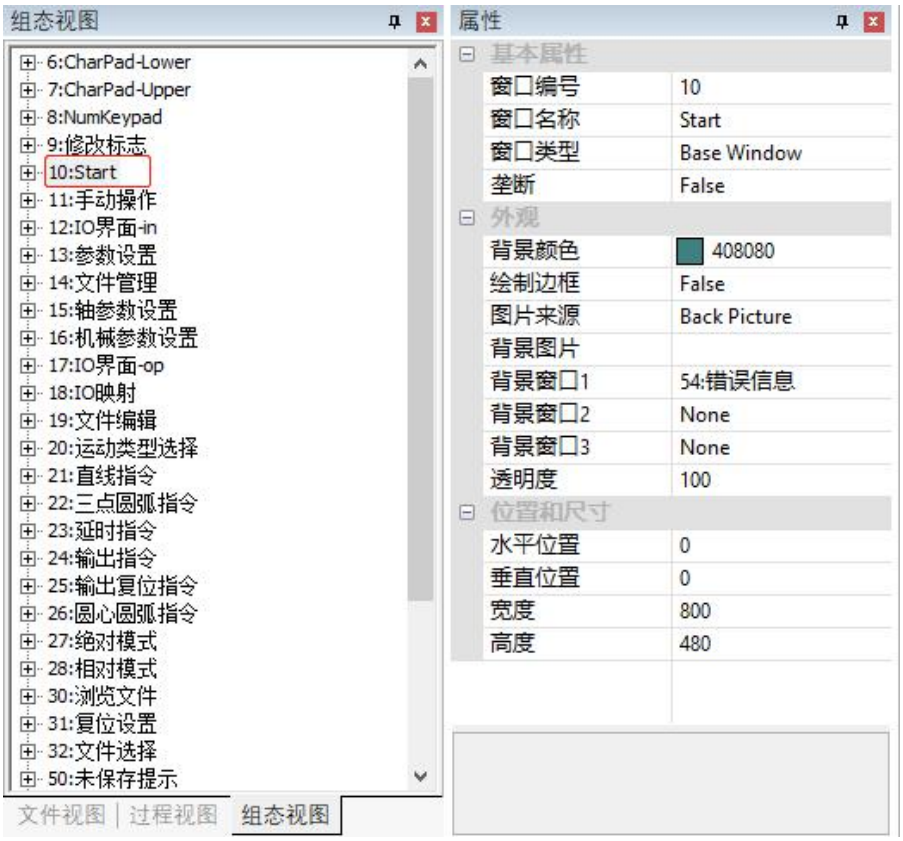
组态显示必须以一个基本窗口为底窗口，作为其他窗口的背景画面，元件需要依附窗口显示，一个组态文件下可新建多个不同类型的窗口。

新建窗口：菜单栏“元件”-“新建窗口”打开如下窗口，输入窗口号和窗口名称后确认，注意窗口号不要重复。

窗口的属性和尺寸位置等信息均需要打开窗口“属性”修改。

组态视图能显示全部窗口和各窗口下的元件，元件在窗口上添加，窗口或元件的属性在“属性”窗口修改。

点击组态窗口或元件便能打开属性窗口。



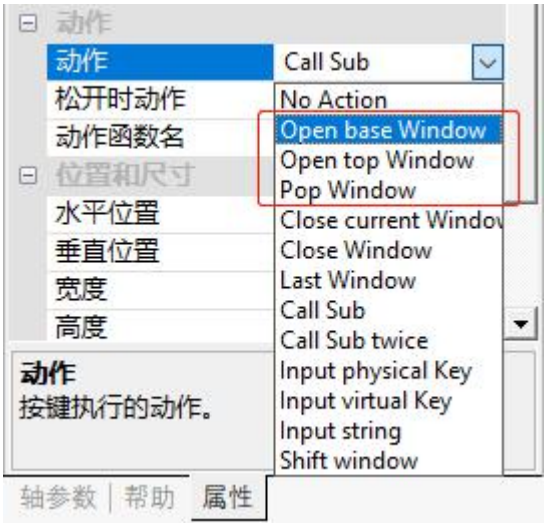
2.2.2. 窗口调用

窗口调用有如下两种方式。

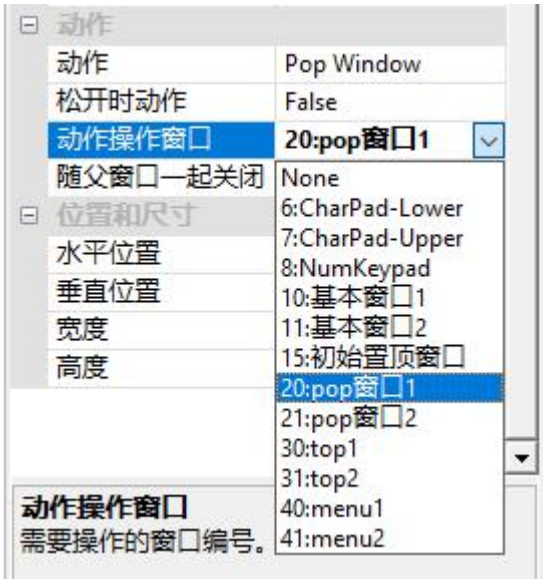
方式一：功能键调用

选择“元件”-“位元件”-“功能键”，新建一个功能键按钮。

选中功能键打开属性，找到“动作”下拉列表，可以选择打开 3 种窗口类型，其中 menu 窗口属于 Pop Window 类型。

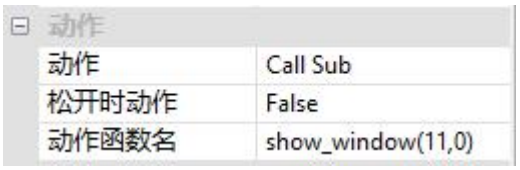


选择好要打开的窗口类型后，再找到“动作操作窗口”，选择要打开的窗口编号，注意要打开的窗口类型要与选择的窗口一致。



方式二：程序指令调用

在元件属性“动作”中选择“call sub”，调用的函数通过在 Basic 中编写程序实现，主要使用到的指令是 HMI_SHOWWINDOW 和 HMI_BASEWINDOW，指令使用方法参见指令描述。




```
GLOBAL SUB showwindow()
    HMI_SHOWWINDOW(11, 0)   '打开窗口11
END SUB
```

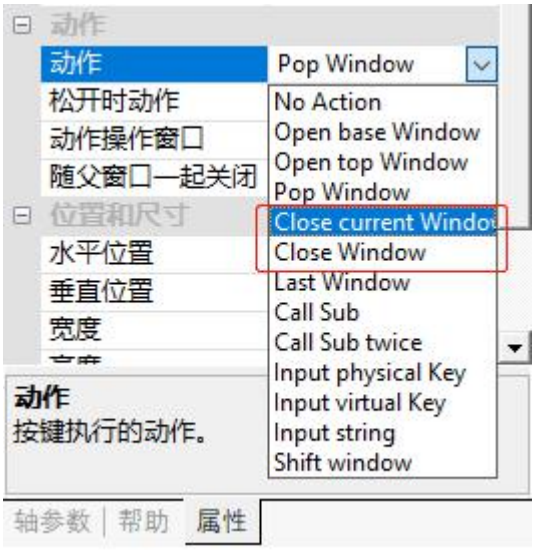
2.2.3. 窗口关闭

关闭窗口有两种方式，基本窗口不支持关闭。

方式一：功能键关闭

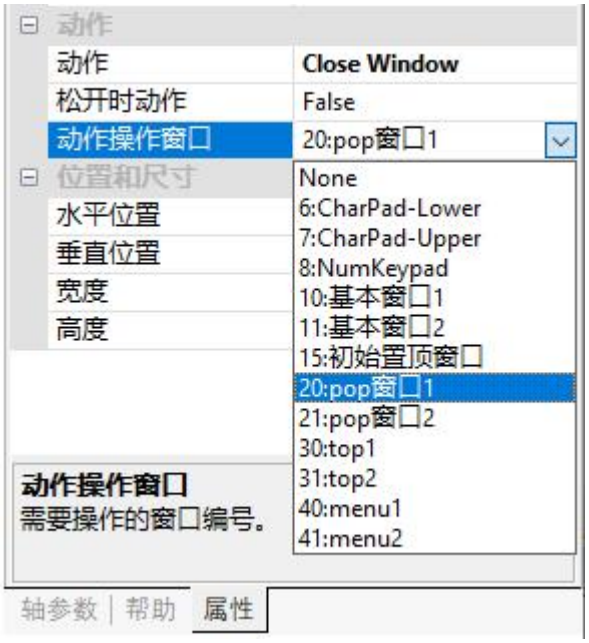
和调用窗口一样，需要先建立一个功能键，切换基本类型窗口操作一般把功能键放在基本窗口内；关闭 Pop 窗口和 Top 窗口操作一般把功能键放在要关闭的窗口内。

动作类型主要有两个。



Close current Window 就是关闭当前窗口，不需要再选择窗口编号。

Close Window 和打开窗口一样，要选择关闭的窗口编号。



方式二：程序指令关闭

程序指令关闭窗口主要通过在 Basic 中编写程序实现，主要使用到的指令是 HMI_CLOSEWINDOW。请查看本帮助文件第五章。

```
global sub closewindow()
    HMI_CLOSEWINDOW()    '参数缺省关闭当前窗口
end sub
```

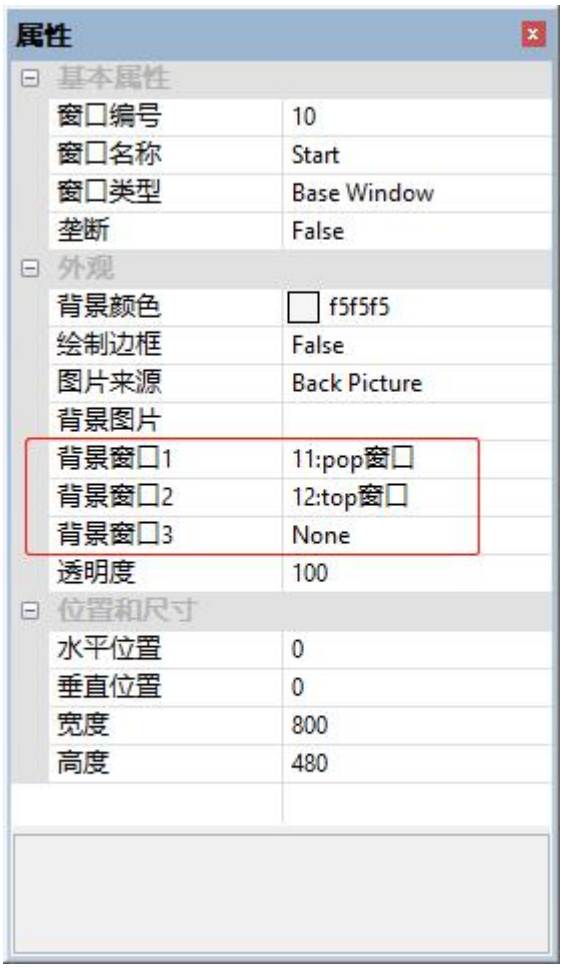
2.2.4. 添加背景窗口

在“窗口属性”中为当前窗口选择背景窗口。

可以为多个窗口指定同一个背景窗口，这样共用的内容可以放在背景窗口里面。

一个窗口最多可以设置 3 个背景窗口。

给当前窗口添加背景窗口后，将设置的背景窗口里的元件显示在当前窗口，但无法对这些元件操作，必须打开背景窗口才能操作。



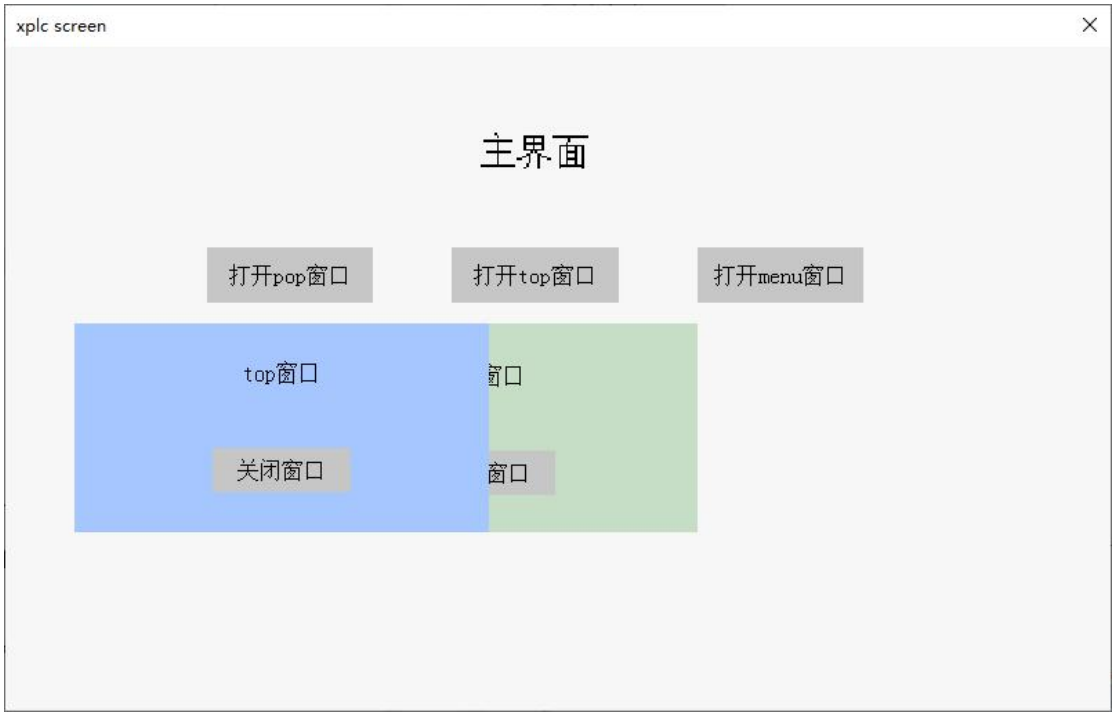
示例：

如下图，没有添加背景窗口的显示情况，每个窗口只显示当前窗口的元件。

给 10 号窗口添加两个背景窗口 11 和 12，这两个窗口的元件就会显示在 10 号窗口里，但不能通过十号窗口操作。

只显示背景窗口的元件，窗口的背景不显示。

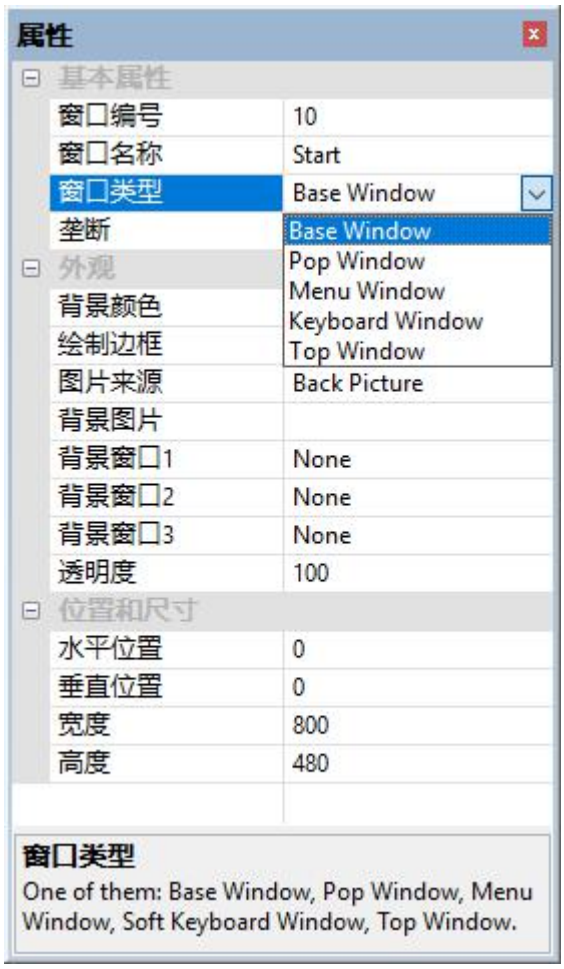
必须操作打开 11 或 12 号窗口后，才能对该窗口的元件进行操作。



2.3.窗口类型

依照功能与使用方式不同，Hmi 支持的窗口类型分为五种：基本窗口(Base Window)、软键盘窗口(Keyboard Window)、弹出窗口(Pop Window)、菜单窗口(Menu Window)、置顶窗口(Top Window)。

新建窗口默认是基本窗口类型，变更类型在窗口的“属性”窗口里修改。



2.3.1. 基本窗口(Base Window)

新建窗口默认为基本窗口类型。

组态显示必须以一个基本窗口为底窗口，作为其他窗口的背景画面。

触摸屏同一时间只能显示一个基本窗口。

基本窗口通过程序或元件操作进行切换，基本窗口不能关闭。

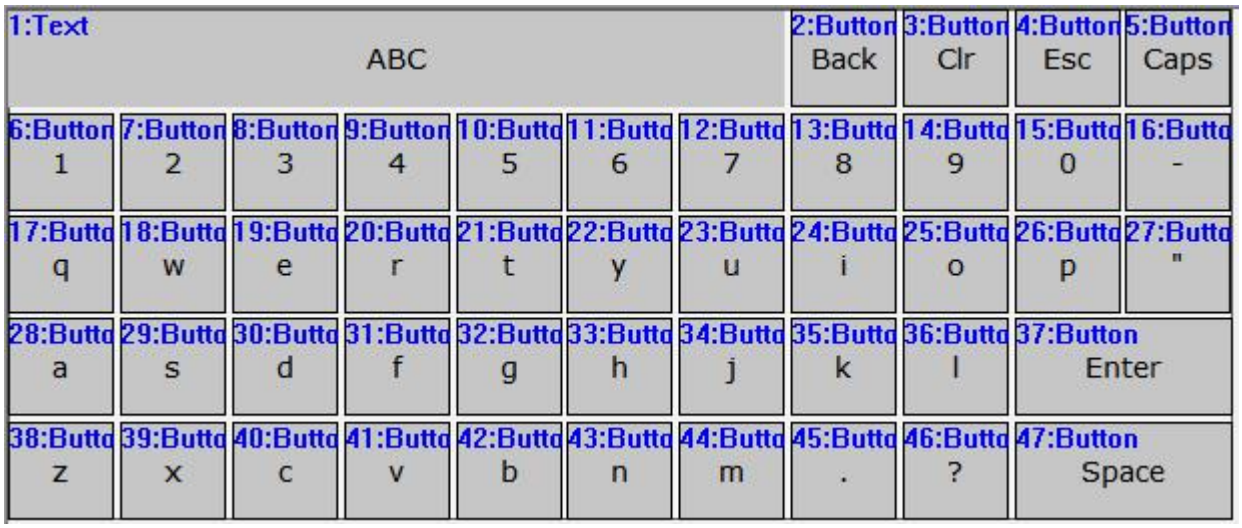
由于起始基本窗口的尺寸大小必须与触摸屏显示屏的大小相同，所以其分辨率设置也必须与所使用的触摸屏分辨率一致，其他基本窗口的大小可任意设置，只要不超出屏幕尺寸即可。

2.3.2. 软键盘窗口(Keyboard Window)

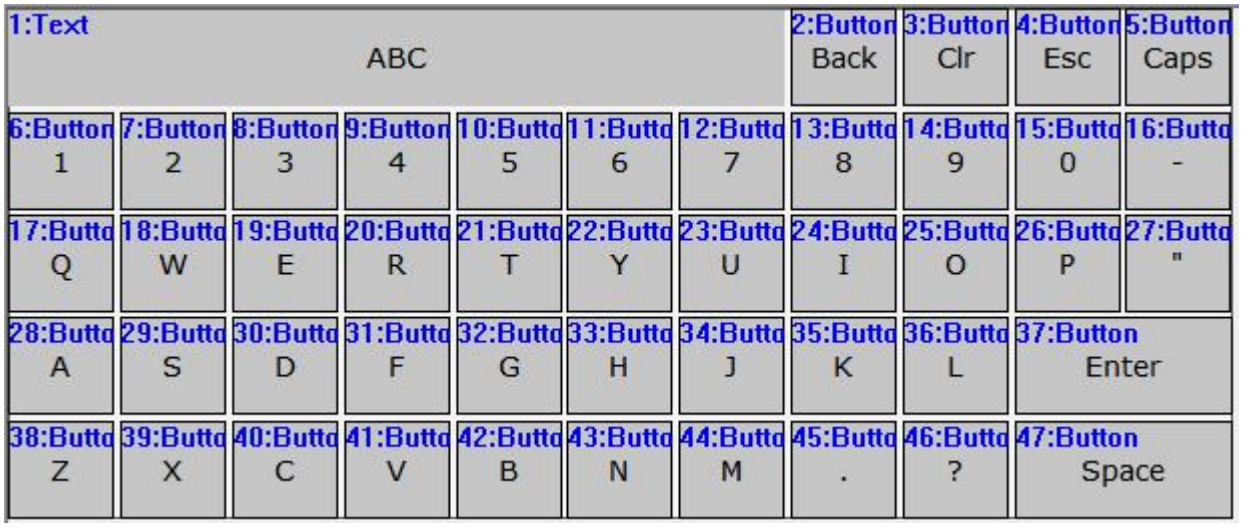
用于“值显示”元件和“字符显示”元件等需要自定义输入数据的场合。

新建的 HMI 文件内置有三种软键盘窗口可供选择，不需要用户新建，窗口号分别为 6、7、8。

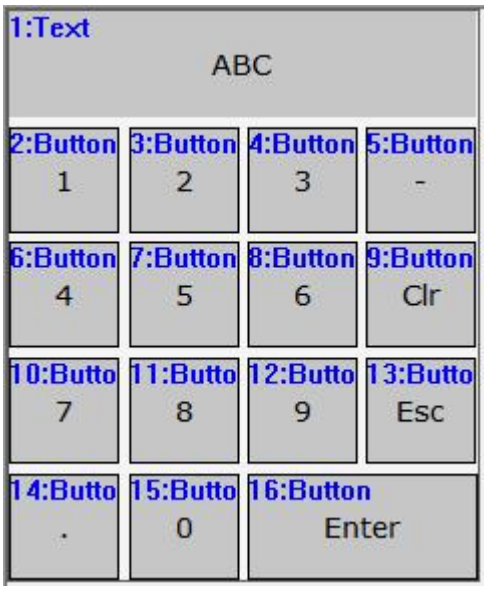
调用软键盘窗口时，只能在显示数据可以修改的元件中使用，比如“值显示”“字符显示”元件。不能用于“功能键”等元件。各类元件用法请参考第三章。



6 号软键盘窗口



7 号软键盘窗口



8 号软键盘窗口

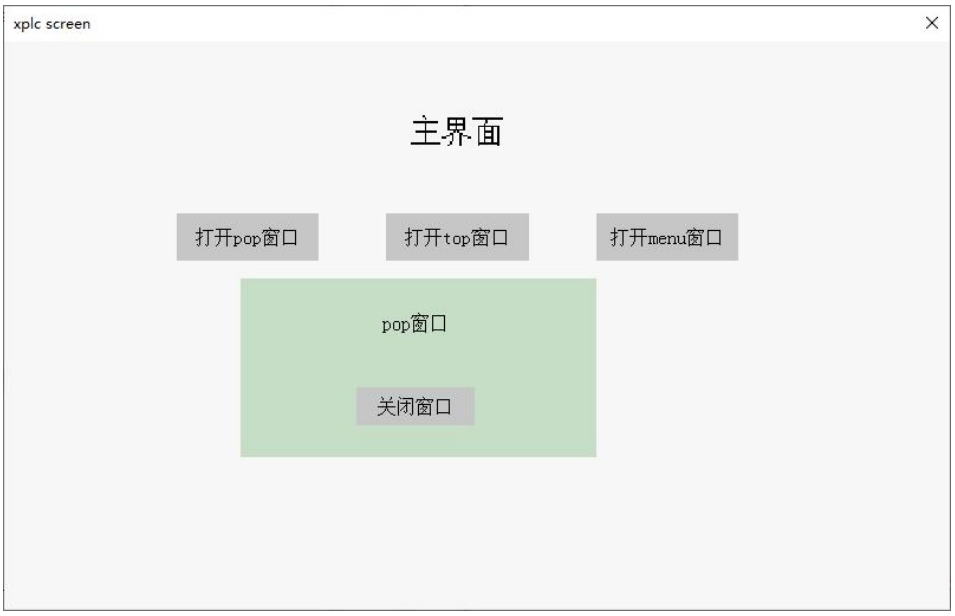
2.3.3. 弹出窗口(Pop Window)

Pop 窗口类似是对话框一样的动态弹出窗口，按照调用顺序，显示最后调用的 Pop 窗口。

当多个 Pop 窗口重叠在一起时，Pop 窗口中的功能按键都是会被正常触发的。

调用出 Pop 窗口后，可以对基本窗口等其他类型窗口进行操作。

Pop 窗口需要通过程序或元件操作关闭，切换基本窗口后也将关闭。

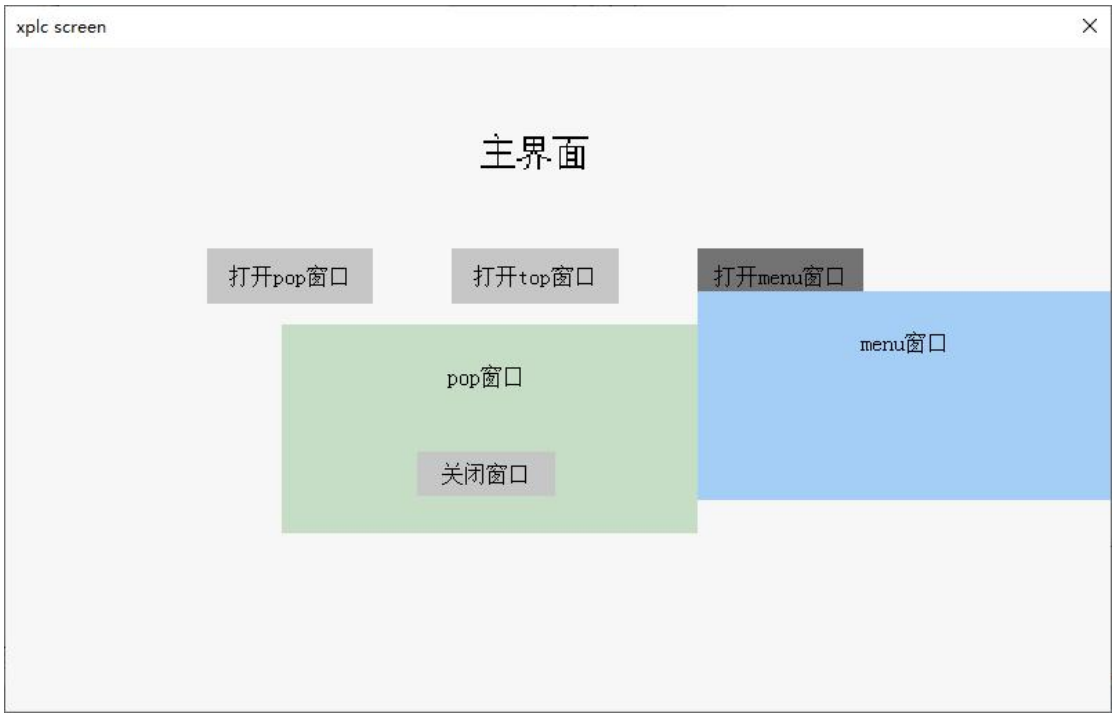


2.3.4. 菜单窗口(Menu Window)

Menu 窗口属于 Pop 窗口，都是调用后弹出。

不同之处是 Menu 窗口弹出后，获得操作最大权限，此时只能对 Menu 窗口进行操作。

当点击到非 Menu 窗口区域时，Menu 窗口自动关闭。



2.3.5. 置顶窗口(Top Window)

总是在最前端显示的窗口，一般为一个小窗口，可以用来实现工具条等。

切换基本窗口时，Top 窗口仍会显示在最前端，不会关闭。

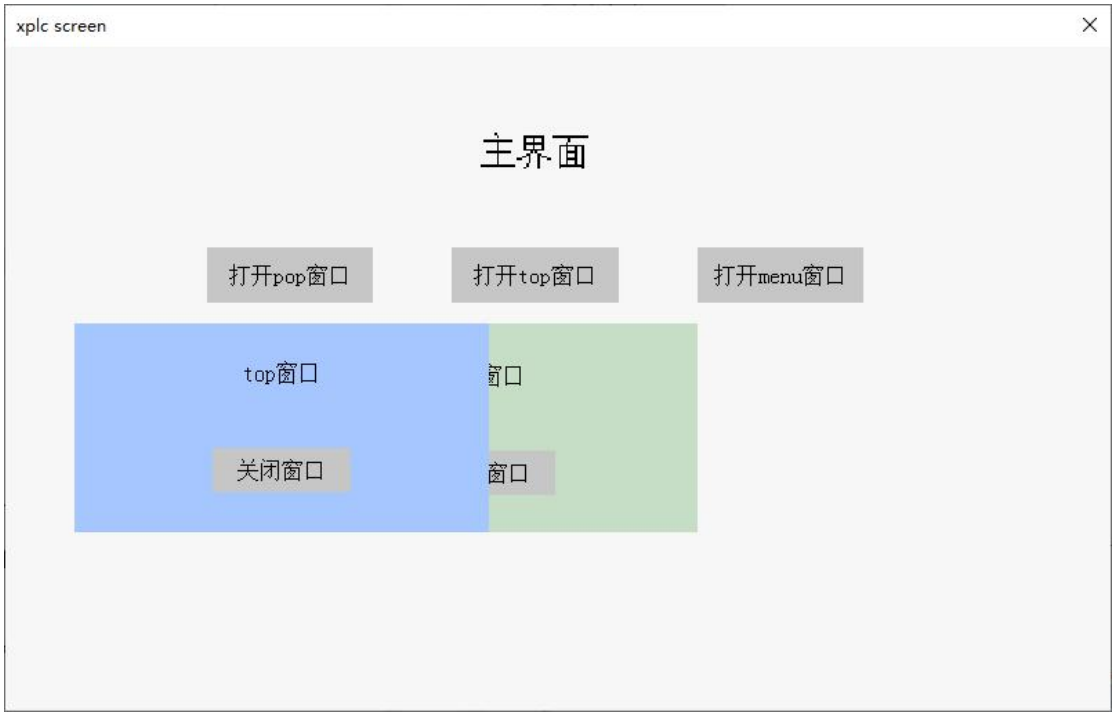
有多个 Top 窗口时，按照调用顺序显示，后调用的窗口在之前调用窗口的上层。

Top 窗口必须通过程序或元件操作关闭。

Top 窗口有在程序开始时显示，和手动调用显示两种。

程序开始时就显示通过菜单栏“编辑”-“hmi 系统设置”来选择起始 Top 窗口。

手动调用显示通过程序或元件操作来显示，如下图，通过功能键的动作打开和关闭 Top 窗口。


xplc screen
主界面
打开pop窗口
打开top窗口
打开menu窗口
top窗口
窗口
关闭窗口
窗口

第三章 组态元件

3.1.组态快捷工具

HMI 开发主要用到窗口和组态元件，先建立不同的显示窗口，在根据程序要求在各个窗口上添加不同的组态元件用于实现相应的功能。

组态元件在 HMI 编程界面下的菜单栏“元件”中选择添加到窗口，或是在元件工具栏和 HMI 工具栏中快速选择元件添加。

为触摸屏仿真快捷键。

元件工具栏：

	线段		文本		物理按键
	多线段		图片		列表
	矩形		矢量图形		值显示
	多边形		位状态显示		字符显示
	贝塞尔曲线		多状态显示		滑块
	椭圆		位状态设置		定时器
	圆弧		多状态设置		自定义元件
	扇形		位状态切换开关		文本库
	尺寸		多状态切换开关		图片库
	表格		功能键		按键转换

HMI 工具栏：

	左对齐		水平居中对齐		宽度相同
	右对齐		垂直居中对齐		高度相同
	上对齐		水平方向相同间隔		水平和垂直方向相同尺寸
	下对齐		垂直方向相同间隔	/	/

：语言切换，需要提前制作多语言的文本库或者图片库才能看到效果。

：状态切换，数值超过元件最大状态时，显示最大状态值。

Hmi 包含元件如下图所示：

一般建立一个元件的步骤如下：

1. 选择元件。

2. 将元件放在 hmi 界面的适合位置。

3. 打开元件属性设置元件的内容。

不同的元件使用时，都要打开元件的“属性”窗口去设置元件的功能。

3.2.组态元件通用属性

不同类型的元件，具有一些共通的属性功能，例如调用寄存器、调用函数、位置和尺寸等。

3.2.1. 寄存器

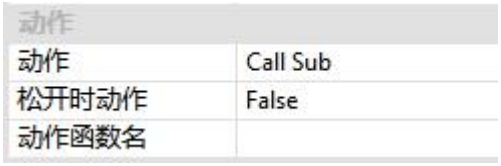
大部分的元件都包含“寄存器类型”这一属性，用来与各类寄存器建立数据联系。

可通过元件控制寄存器的值，或获取寄存器的值显示，可用寄存器类型如下表。

寄存器类型	对应控制器寄存器	说明
X	输入口 IN	此寄存器对应通用输入，编号 0 对应 MODBUS_BIT(10000)
Y	输出口 OP	此寄存器对应通用输出，编号 0 对应 MODBUS_BIT(20000)
M	MODBUS_BIT	不同型号控制器的寄存器个数有区别 掉电保持：2048-2175
S	状态寄存器 S	编号 0-999，编号 0 对应 MODBUS_BIT(30000) 掉电保持：0-127
D	MODBUS_4X 寄存器 根据数据类型 INT16：MODBUS_REG INT32：MODBUS_LONG FLOAT32：MODBUS_IEEE	不同型号控制器的寄存器个数有区别
D.DOT	按位读取 MODBUS_REG 编号=reg 号*16+dot(0-15)	请使用位状态显示元件
DT	TABLE	32 位浮点型数据
T	定时器 编号 0-127	寄存器长度 32 位，当通过 16 位指令访问时自动使用低 16 位
C	计数器	寄存器长度 32 位，当通过 16 位指令访

	编号 0-127	问时自动使用低 16 位
@	Basic 定义的变量、数组	必须是 GLOBAL 全局类型才能访问

3.2.2. 动作

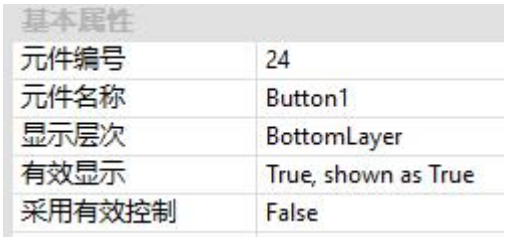


通过下拉菜单选择，不同元件的可选动作不同，动作的功能参见各元件的例程。

动作名	功能	说明
元件通用动作		
No Action	无	/
Call sub	调用 Basic 定义 SUB 函数	函数必须是 GLOBAL 全局类型。
功能键 Button		
Open base Window	以基本窗口类型打开窗口	窗口号通过“动作操作窗口”选择。要打开的窗口类型要保持一致。
Open top Window	以 top 窗口类型打开窗口	
Pop Window	以 pop 窗口类型打开窗口	
Close current Window	关闭当前窗口	关闭当前功能键所在的窗口。
Close Window	关闭选择窗口	窗口号通过“动作操作窗口”选择。
Last Window	打开最后一个基本窗口	打开最近一次操作的 Base Window
Call sub Twice	按下调用一个函数，松开调用另一个函数	需要设置“调用函数名”和“松开调用函数”
Input physical key	与物理按键绑定	需要物理按键对应表。
Input virtual key	设为虚拟按键	通过“虚拟按键码”选择编号。
Input String	输入字符串	只能在 keyboard 窗口内使用
Shift Window	切换窗口	必须在非 base 类型窗口内才可以使用。且只能在同类窗口间切换。
位状态切换开关 BitSwitch / 位状态设置 BitModify		
Set Bit	按下时，置 1	开关类型
Reset Bit	按下时，置 0	
Reverse Bit	取反，为 1 时变成 0，为 0 时变成 1	
Recovery Bit	按下时置 1，松开时置 0	
Set when Open	元件所在窗口被打开时，置 1	
Reset when Open	元件所在窗口被打开时，置 0	
Set when Close	元件所在窗口被关闭时，置 1	

Reset when Close	元件所在窗口被关闭时，置 0	
字状态切换开关 WordSwitch / 字状态设置 WordModify		
Data Write	写入数据到寄存器	数据值通过“动作数据”设置。
Data Plus	寄存器原来值加上数据	寄存器通过“寄存器类型”和编号选择。 如果原来值大于“状态数量”，会先按（“状态数量”-“动作数据”）递减为 0
Data Loop	寄存器原来值加上数据，在设置的状态之间循环切换，实现周期切换数据效果	

3.2.3. 基本属性



元件编号：在当前窗口按添加的顺序从 0 开始编号。

元件名称：名称+编号。

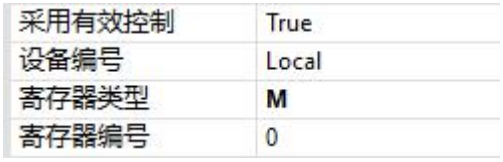
显示层次：在多个元件叠加时，可以设置元件的显示层次。

TopLayer：表层，显示在最外层，覆盖底下元件；

MidLayer：中间层；

BottomLayer：底层。

采用有效控制：默认 False，若选择 Ture，采用设备的寄存器控制该元件是否显示，选择寄存器类型和编号，如下图，将 M0，即 MODBUS_BIT(0)置 1 之后该元件才能在组态窗口显示，否则是隐藏状态。



3.2.4. 外观

外观		外观	
图片来源	Back Picture	图片来源	Back PictureLib
背景图片		背景图片库	
绘制边框	False	绘制边框	False
是否图片化	False	是否图片化	False

部分元件支持显示图片，在“图片来源”里选择使用背景图片或背景图片库，使用然后在下面一行选择目标图片添加。

Back PictureLib 背景图片库：图片先添加到图片库中，然后在背景图片库中选择。

Back Picture 背景图片：图片先添加到项目的“文件视图”中，然后在背景图片中选择。

是否图片化功能用于将元件显示内容图片化，例如：当显示的文本字体过大或包含生僻字时，导致字体不清晰，将显示内容图片化，设为 True 之后，显示效果能改善。

3.2.5. 位置和尺寸

位置和尺寸	
水平位置	320
垂直位置	180
宽度	100
高度	32

屏幕左上角的位置为(0,0)，水平位置和垂直位置元件放置于编辑窗口中的位置。

高度和宽度用于设置元件的显示大小。

元件位置和尺寸最好不要超出窗口的范围。

3.2.6. 格式文本

设置元件的显示文字内容，点击后弹出“格式文本”设置窗口。

标签	
文本库	
格式文本	

在“文本”中输入要元件显示的内容后“确定”。

操作	功能	说明
文本	输入文本内容显示	/
水平对齐	水平对齐选项	0：居中对齐（默认） >0：左边对齐，值表示距离左边的距离 <0：右边对齐，绝对值表示距离右边的距离
垂直对齐	垂直对齐选项	0：居中对齐（默认） >0：上边对齐，值表示距离上边的距离 <0：下边对齐，绝对值表示距离下边的距离
字体颜色	选择字体颜色	/
字体大小	选择字体大小	/
字体	选择字体	将字体文件添加到项目中
背景颜色	选择背景颜色	/
填充样式	填充到背景的样式	/
样式颜色	选择样式颜色	/

3.2.7. 图片库和文本库

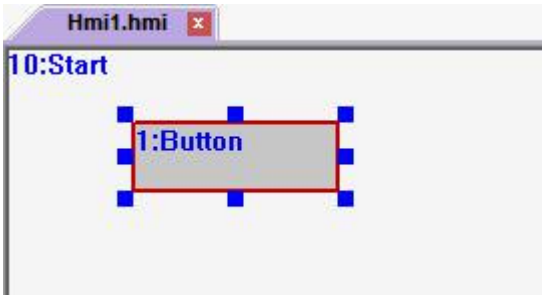
在元件“属性”中添加“图片库”、“文本库”和“背景图片”之前，先建立后选择，图片来源选择背景图片 Back Picture，需要先将图片添加到项目，然后在此窗口选择要加载的图片名称。建立图片库的方法参见“图片库”，建立文本库的方法参见“文本库”。

3.3.绘图

不同类型的的组态元件添加到组态窗口之后，均通过打开元件的“属性”窗口设置元件的功能特点。

鼠标点击元件之后，元件具有下图显示的蓝色边框，当前元件的“属性”窗口自动被打开，即可编辑元件属性，属性编辑完成后单击回车或者鼠标点到其他地方，自动保存编辑的属性信息。

的若“属性”窗口不能自动被打开，点击菜单栏“视图”-“属性”先打开属性窗口，之后再点击元件。

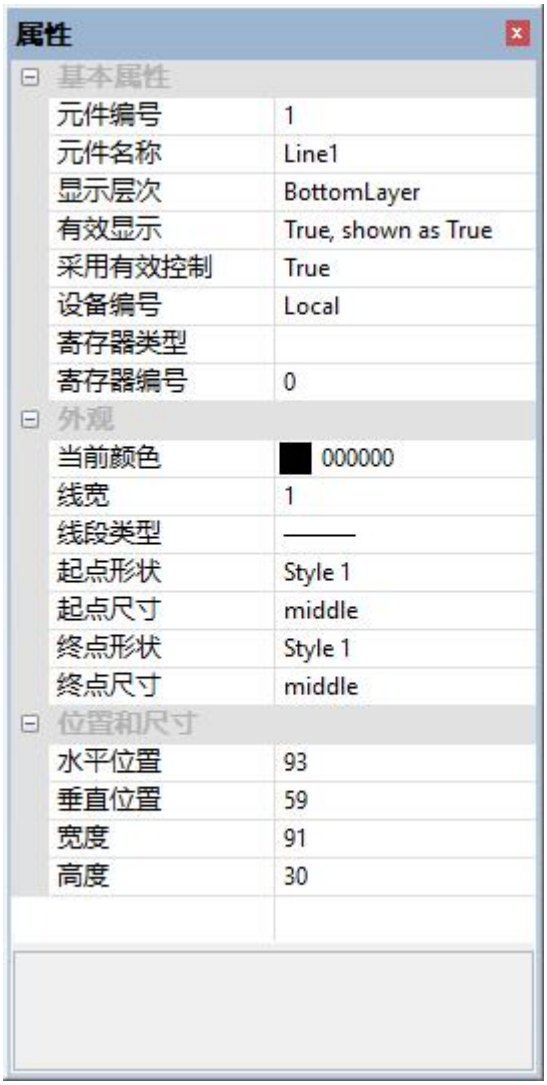


元件的大小设置：将鼠标放在上图蓝色方块上直接拖拽改变，或由“属性”窗口的宽度高度设置。

3.3.1. 线段

“线段”为直线，直线的长度和斜率率由包围它的元件的高度和宽度定义，可直接拖拽元件定义高宽，也可在元件属性里定义。

1. 属性窗口：



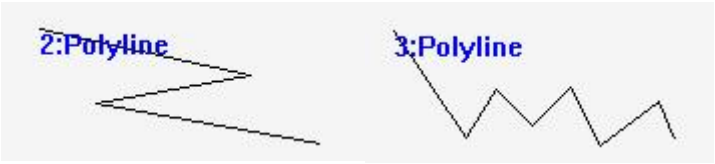
2. 属性说明：

属性	功能	说明
元件编号	/	/
元件名称	/	/
显示层次	选择元件显示层次	TopLayer：表层，显示在最外层，覆盖底下元件

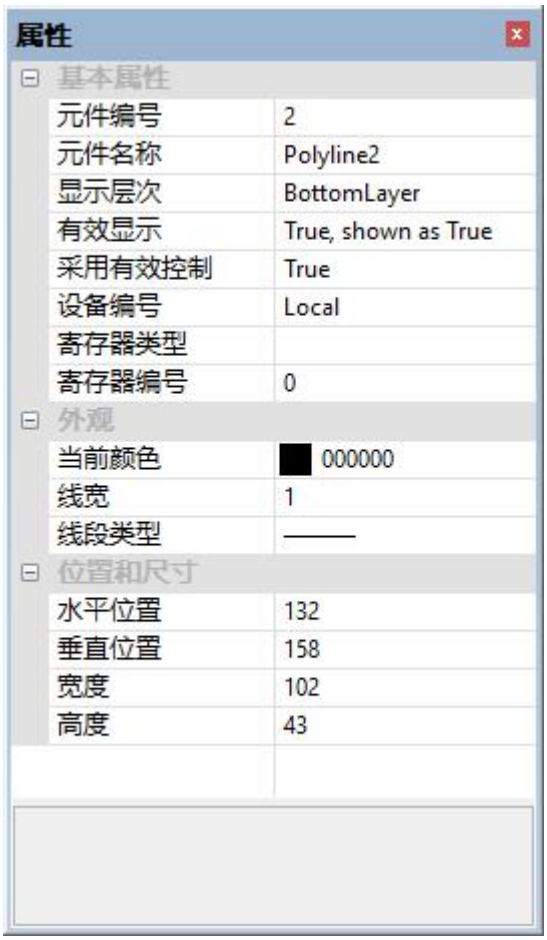
		MidLayer：中间层 BottomLayer：底层(默认)
有效显示	选择元件是否显示	默认 True，选择 False 时，元件不显示且无功能作用
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 Ture 才会出现下方的三个参数
设备编号	有效控制的设备编号	默认 local
寄存器类型	选择寄存器类型	多种寄存器下拉列表选择
寄存器编号	选择寄存器编号	寄存器值为 0 时不显示，非 0 时使用
当前颜色	选择线段颜色	/
线宽	线段的宽度	默认宽度为 1
线段类型	线段的样式	下拉列表选择实线或虚线等类型
起点形状	多种形状下拉列表选择	/
起点尺寸	起点形状的尺寸大小	选了特殊起点形状才有效
终点形状	多种形状下拉列表选择	/
终点尺寸	终点形状的尺寸大小	选了特殊起点形状才有效
水平位置	元件的水平起始位置	不要超出水平分辨率
垂直位置	元件的垂直起始位置	不要超出垂直分辨率
宽度	元件的宽度	/
高度	元件的高度	/

3.3.2. 多线段

选择“多线段”元件，用鼠标左键画出多个连续的线段，点击鼠标右键表示画线完成，画出的线段可以拥有任意数目转角，可以画成任意图形，线段画完后不能更改，只能调整大小。虽然始点和结束点可能重合在同一个坐标，但是不能填充由线定义的区域。



1. 属性窗口：



2. 属性说明：

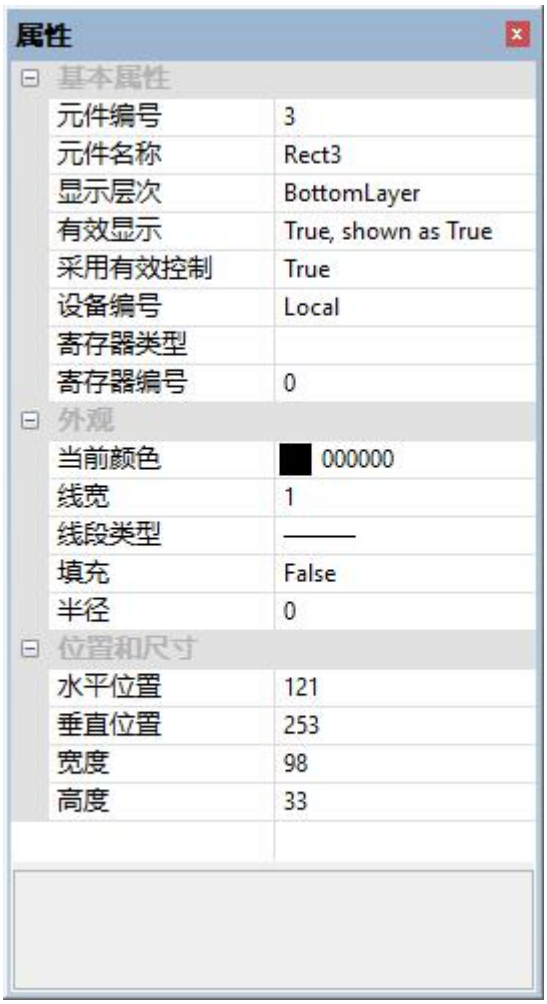
属性	功能	说明
元件编号	/	/
元件名称	/	/
显示层次	选择元件显示层次	TopLayer：表层，显示在最外层，覆盖底下元件 MidLayer：中间层 BottomLayer：底层(默认)
有效显示	选择元件是否显示	默认 True，选择 False 时，元件不显示且无功能作用
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 Ture 才会出现下方的三个参数
设备编号	有效控制的设备编号	默认 local
寄存器类型	选择寄存器类型	多种寄存器下拉列表选择
寄存器编号	选择寄存器编号	寄存器值为 0 时不显示，非 0 时使用
当前颜色	选择线段颜色	/
线宽	线段的宽度	/
线段类型	线段的样式	/

水平位置	元件的水平起始位置	不要超出水平分辨率
垂直位置	元件的垂直起始位置	不要超出垂直分辨率
宽度	元件的宽度	/
高度	元件的高度	/

3.3.3. 矩形

创建一个矩形，“矩形” 是一个可以填充背景颜色的闭合对象。

1. 属性窗口：



2. 属性说明：

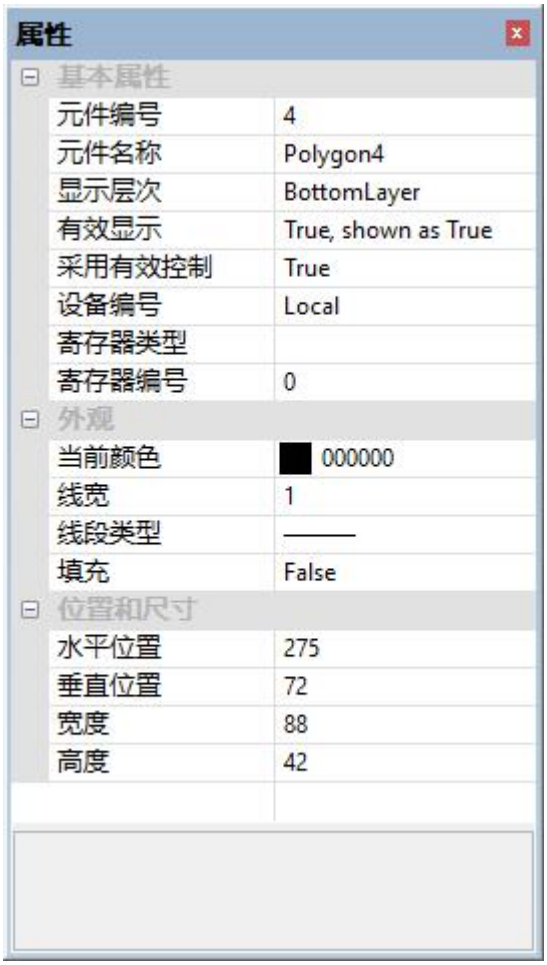
属性	功能	说明
元件编号	/	/

元件名称	/	/
显示层次	选择元件显示层次	TopLayer：表层，显示在最外层，覆盖底下元件 MidLayer：中间层 BottomLayer：底层(默认)
有效显示	选择元件是否显示	默认 True，选择 False 时，元件不显示且无功能作用
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 Ture 才会出现下方的三个参数
设备编号	有效控制的设备编号	默认 local
寄存器类型	选择寄存器类型	多种寄存器下拉列表选择
寄存器编号	选择寄存器编号	寄存器值为 0 时不显示，非 0 时使用
当前颜色	选择线段颜色	/
线宽	线段的宽度	/
线段类型	线段的样式	/
填充	选择是否填充颜色	填充整个元件
半径	倒角半径	设置四个角是否要倒圆角
水平位置	元件的水平起始位置	不要超出水平分辨率
垂直位置	元件的垂直起始位置	不要超出垂直分辨率
宽度	元件的宽度	/
高度	元件的高度	/

3.3.4. 多边形

“多边形”的形状需要用户手绘，绘制完成点击鼠标右键图形自动封闭，形状确定后不能修改，只支持调整大小，“多边形”是一个可以填充背景颜色的闭合对象。

1. 属性窗口：



2. 属性说明：

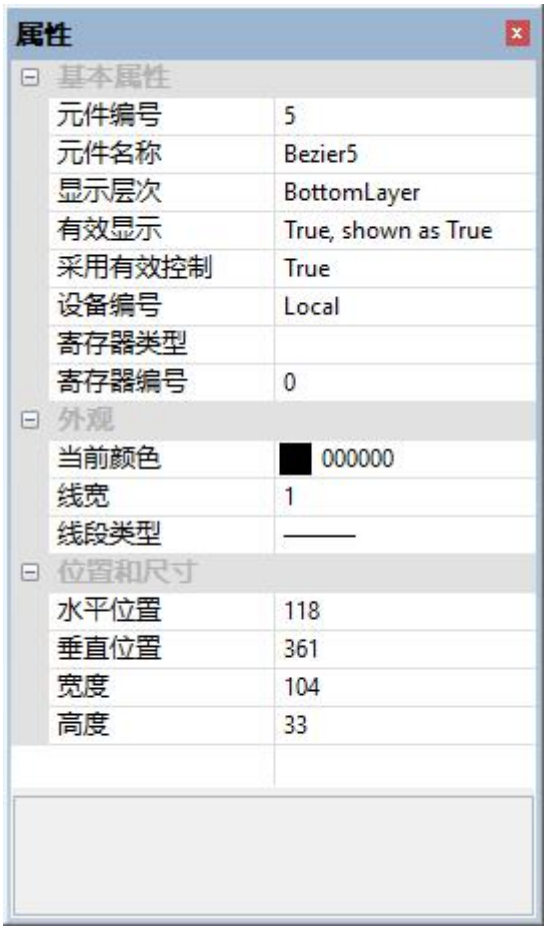
属性	功能	说明
元件编号	/	/
元件名称	/	/
显示层次	选择元件显示层次	TopLayer：表层，显示在最外层，覆盖底下元件 MidLayer：中间层 BottomLayer：底层(默认)
有效显示	选择元件是否显示	默认 True，选择 False 时，元件不显示且无功能作用
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 Ture 才会出现下方的三个参数
设备编号	有效控制的设备编号	默认 local
寄存器类型	选择寄存器类型	多种寄存器下拉列表选择
寄存器编号	选择寄存器编号	寄存器值为 0 时不显示，非 0 时使用
当前颜色	选择线段颜色	/
线宽	线段的宽度	/
线段类型	线段的样式	/

填充	选择是否填充颜色	填充整个元件
水平位置	元件的水平起始位置	不要超出水平分辨率
垂直位置	元件的垂直起始位置	不要超出垂直分辨率
宽度	元件的宽度	/
高度	元件的高度	/

3.3.5. 贝塞尔曲线

“贝塞尔曲线”显示贝塞尔曲线，绘制时由四个点自动创建贝塞尔曲线，形状确定后不能修改，只支持调整大小。

1. 属性窗口：



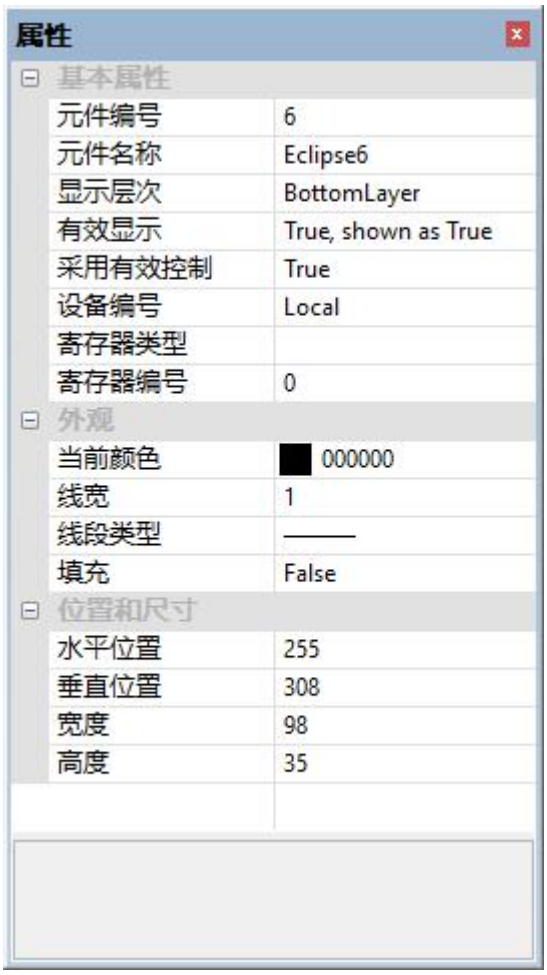
2. 属性说明：

属性	功能	说明
元件编号	/	/
元件名称	/	/
显示层次	选择元件显示层次	TopLayer：表层，显示在最外层，覆盖底下元件 MidLayer：中间层 BottomLayer：底层(默认)
有效显示	选择元件是否显示	默认 True，选择 False 时，元件不显示且无功能作用
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 Ture 才会出现下方的三个参数
设备编号	有效控制的设备编号	默认 local
寄存器类型	选择寄存器类型	多种寄存器下拉列表选择
寄存器编号	选择寄存器编号	寄存器值为 0 时不显示，非 0 时使用
当前颜色	选择线段颜色	/
线宽	线段的宽度	/
线段类型	线段的样式	/
水平位置	元件的水平起始位置	不要超出水平分辨率
垂直位置	元件的垂直起始位置	不要超出垂直分辨率
宽度	元件的宽度	/
高度	元件的高度	/

3.3.6. 椭圆

“椭圆”"是一个可以填充背景颜色的闭合对象。

1. 属性窗口：



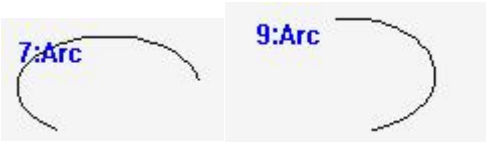
2. 属性说明：

属性	功能	说明
元件编号	/	/
元件名称	/	/
显示层次	选择元件显示层次	TopLayer：表层，显示在最外层，覆盖底下元件 MidLayer：中间层 BottomLayer：底层(默认)
有效显示	选择元件是否显示	默认 True，选择 False 时，元件不显示且无功能作用
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 Ture 才会出现下方的三个参数
设备编号	有效控制的设备编号	默认 local
寄存器类型	选择寄存器类型	多种寄存器下拉列表选择
寄存器编号	选择寄存器编号	寄存器值为 0 时不显示，非 0 时使用
当前颜色	选择线段颜色	/
线宽	线段的宽度	/

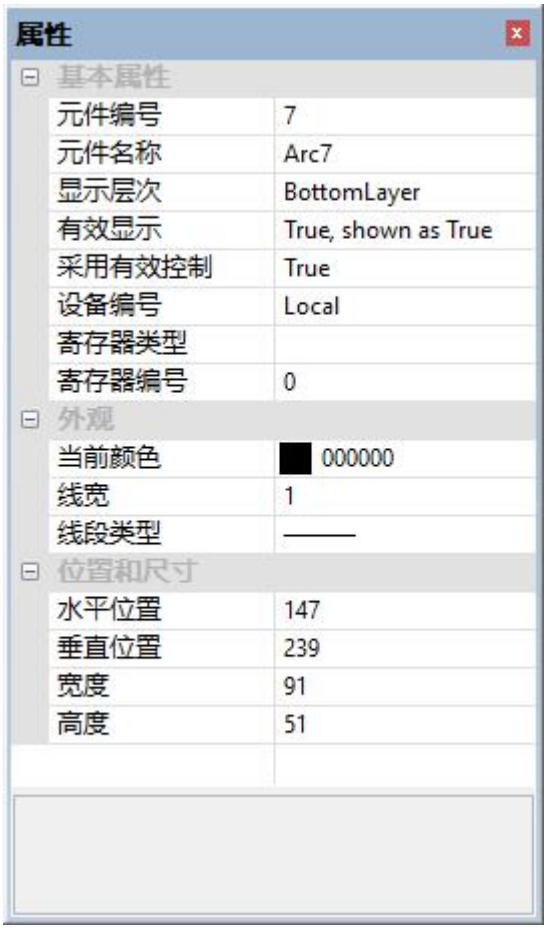
线段类型	线段的样式	/
填充	选择是否填充颜色	填充整个元件
水平位置	元件的水平起始位置	不要超出水平分辨率
垂直位置	元件的垂直起始位置	不要超出垂直分辨率
宽度	元件的宽度	/
高度	元件的高度	/

3.3.7. 圆弧

绘制“圆弧”时，先拉一个矩形框，接下来要确定圆弧的起点和终点，按起点到终点绘制圆弧，不支持画整圆。形状确定后不能修改，只支持调整大小。



1. 属性窗口：



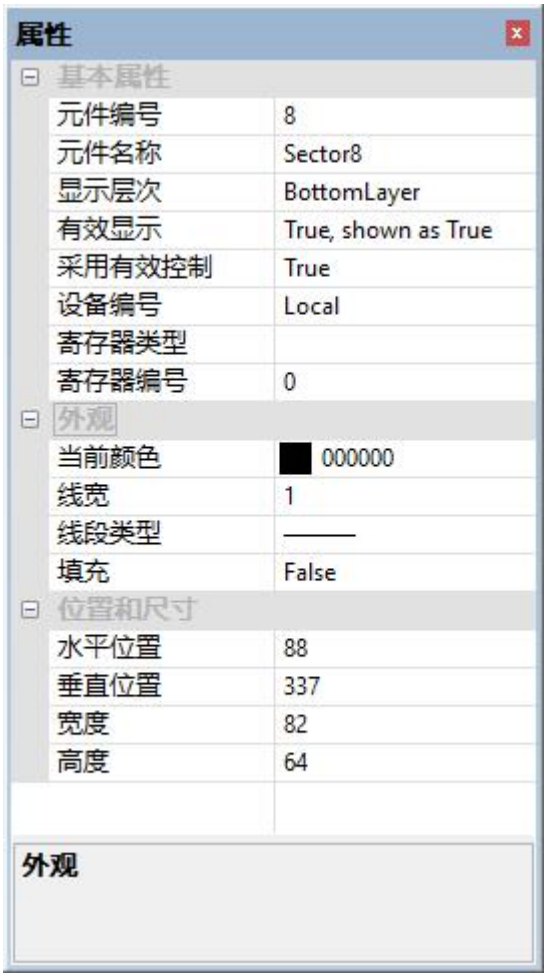
2. 属性说明：

属性	功能	说明
元件编号	/	/
元件名称	/	/
显示层次	选择元件显示层次	TopLayer：表层，显示在最外层，覆盖底下元件 MidLayer：中间层 BottomLayer：底层(默认)
有效显示	选择元件是否显示	默认 True，选择 False 时，元件不显示且无功能作用
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 Ture 才会出现下方的三个参数
设备编号	有效控制的设备编号	默认 local
寄存器类型	选择寄存器类型	多种寄存器下拉列表选择
寄存器编号	选择寄存器编号	寄存器值为 0 时不显示，非 0 时使用
当前颜色	选择线段颜色	/
线宽	线段的宽度	/
线段类型	线段的样式	/
水平位置	元件的水平起始位置	不要超出水平分辨率
垂直位置	元件的垂直起始位置	不要超出垂直分辨率
宽度	元件的宽度	/
高度	元件的高度	/

3.3.8. 扇形

绘制“扇形”时，先拉一个矩形框，接下来要确定圆弧的起点和终点，从起点到终点绘制圆弧，再把圆弧的起点和终点与圆心连接起来，构成一个扇形。不支持画整圆。形状确定后不能修改，只支持调整大小。

1. 属性窗口：



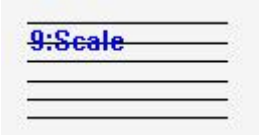
2. 属性说明：

属性	功能	说明
元件编号	/	/
元件名称	/	/
显示层次	选择元件显示层次	TopLayer：表层，显示在最外层，覆盖底下元件 MidLayer：中间层 BottomLayer：底层(默认)
有效显示	选择元件是否显示	默认 True，选择 False 时，元件不显示且无功能作用
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 Ture 才会出现下方的三个参数
设备编号	有效控制的设备编号	默认 local
寄存器类型	选择寄存器类型	多种寄存器下拉列表选择
寄存器编号	选择寄存器编号	寄存器值为 0 时不显示，非 0 时使用
当前颜色	选择线段颜色	/
线宽	线段的宽度	/

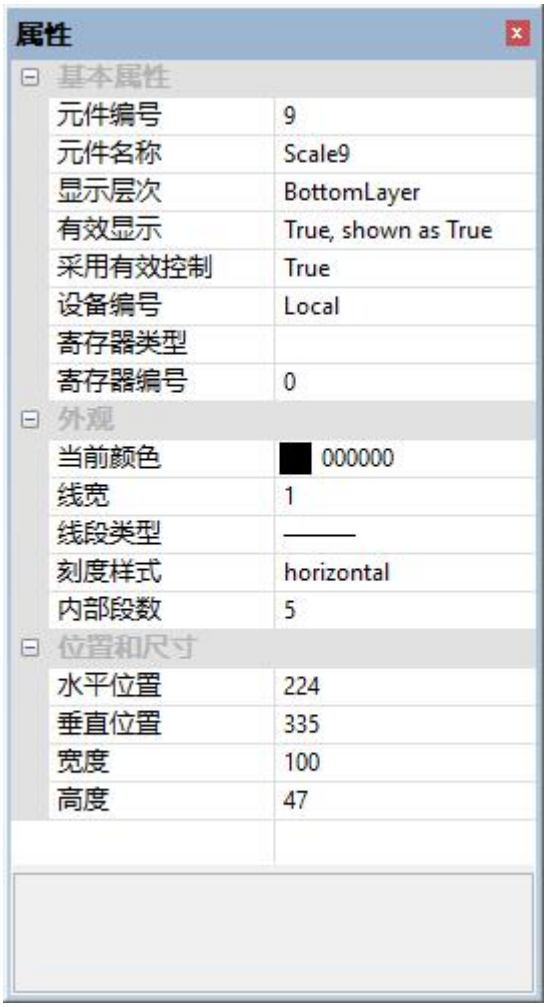
线段类型	线段的样式	/
填充	选择是否填充颜色	填充整个元件
水平位置	元件的水平起始位置	不要超出水平分辨率
垂直位置	元件的垂直起始位置	不要超出垂直分辨率
宽度	元件的宽度	/
高度	元件的高度	/

3.4.刻度

“刻度”功能为预留，暂不支持。



1. 属性窗口：

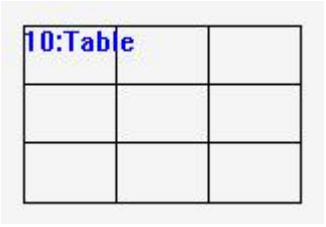


2. 属性说明：

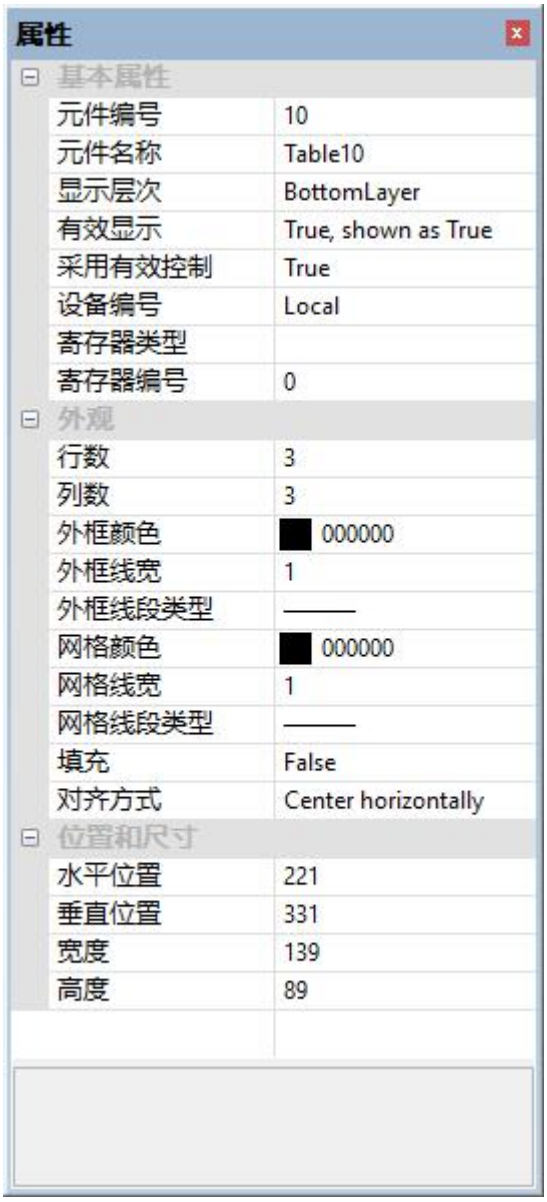
属性	功能	说明
元件编号	/	/
元件名称	/	/
显示层次	选择元件显示层次	TopLayer：表层，显示在最外层，覆盖底下元件 MidLayer：中间层 BottomLayer：底层(默认)
有效显示	选择元件是否显示	默认 True，选择 False 时，元件不显示且无功能作用
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 Ture 才会出现下方的三个参数
设备编号	有效控制的设备编号	默认 local
寄存器类型	选择寄存器类型	多种寄存器下拉列表选择
寄存器编号	选择寄存器编号	寄存器值为 0 时不显示，非 0 时使用
当前颜色	选择线段颜色	/
线宽	线段的宽度	/
线段类型	线段的样式	/
刻度样式	选择刻度纵向或横向显示	/
内部段数	设置内部的刻度段数	默认为 5
水平位置	元件的水平起始位置	不要超出水平分辨率
垂直位置	元件的垂直起始位置	不要超出垂直分辨率
宽度	元件的宽度	/
高度	元件的高度	/

3.5.表格

“表格”功能为预留，暂不支持。



1. 属性窗口：



2. 属性说明：

属性	功能	说明
元件编号	/	/
元件名称	/	/
显示层次	选择元件显示层次	TopLayer：表层，显示在最外层，覆盖底下元件 MidLayer：中间层 BottomLayer：底层(默认)
有效显示	选择元件是否显示	默认 True，选择 False 时，元件不显示且无功能作用
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 Ture 才会出现下方的三个参数
设备编号	有效控制的设备编号	默认 local

寄存器类型	选择寄存器类型	多种寄存器下拉列表选择
寄存器编号	选择寄存器编号	寄存器值为 0 时不显示，非 0 时使用
行数	设置表格行数	/
列数	设置表格列数	/
外框颜色	设置表格外框颜色	/
外框线宽	设置表格外框线宽	/
外框线段类型	设置表格外框线段类型	/
网格颜色	设置表格网格颜色	/
网格线宽	设置表格网格线宽	/
网格线段类型	设置表格网格线段类型	/
填充	选择是否填充颜色	/
对齐方式	设置表格内容的对齐方式	/
水平位置	元件的水平起始位置	不要超出水平分辨率
垂直位置	元件的垂直起始位置	不要超出垂直分辨率
宽度	元件的宽度	/
高度	元件的高度	/

3.6.文本

“文本”支持显示单行文字，在“属性”的“格式文本”中输入要显示的文本内容，文本较多的时候，注意条件文件尺寸，元件太小会显示不全。

1. 属性窗口：

2. 属性说明：

属性	功能	说明
元件编号	/	/
元件名称	/	/
显示层次	选择元件显示层次	TopLayer：表层，显示在最外层，覆盖底下元件 MidLayer：中间层 BottomLayer：底层(默认)
有效显示	选择元件是否显示	默认 True，选择 False 时，元件不显示且无功能作用
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 Ture 才会出现下方的三个参数
设备编号	有效控制的设备编号	默认 local
寄存器类型	选择寄存器类型	多种寄存器下拉列表选择
寄存器编号	选择寄存器编号	寄存器值为 0 时不显示，非 0 时使用
闪烁	交替显示间隔时间	可选不闪烁、500ms 和 1000ms
是否图片化	元件变为图片的形式	默认 False

文本库	文本库的名称	不设置文本库显示格式文本
格式文本	元件文本显示	打开格式文本设置窗口设置元件要显示的文本
水平位置	元件的水平起始位置	不要超出水平分辨率
垂直位置	元件的垂直起始位置	不要超出垂直分辨率
宽度	元件的宽度	/
高度	元件的高度	/

3.7.图片

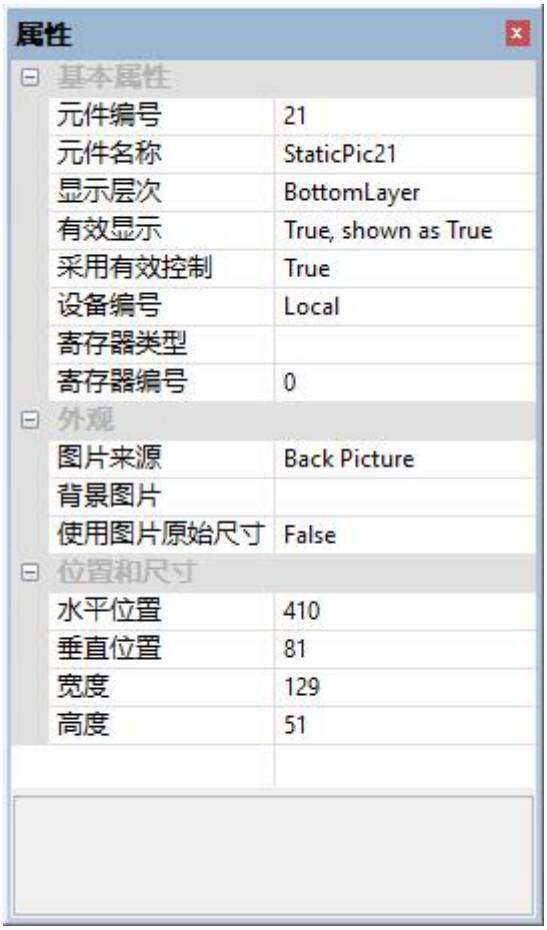
先添加“图片”元件，在元件属性里添加要显示的图片，可选择图片库已有的图片，或选择增加到文件视图的背景图片。

Back PictureLib 背景图片库：图片先添加到图片库中，然后在背景图片库中选择。

Back Picture 背景图片：图片先添加到项目的“文件视图”中，然后在背景图片中选择图片。

在使用视觉的时候，图片元件支持使用视觉拍摄的图片，在“背景图片”中选择视觉通道，支持三个视觉通道：@ZV0 、@ZV1、@ZV2，使用前先将图片存入视觉锁存通道才能获取显示。

1. 属性窗口：



2. 属性说明：

属性	功能	说明
元件编号	/	/
元件名称	/	/
显示层次	选择元件显示层次	TopLayer：表层，显示在最外层，覆盖底下元件 MidLayer：中间层 BottomLayer：底层(默认)
有效显示	选择元件是否显示	默认 True，选择 False 时，元件不显示且无功能作用
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 Ture 才会出现下方的三个参数
设备编号	有效控制的设备编号	默认 local
寄存器类型	选择寄存器类型	多种寄存器下拉列表选择
寄存器编号	选择寄存器编号	寄存器值为 0 时不显示，非 0 时使用
图片来源	Back PictureLib 或 Back Picture	图片库或背景图片中选择
背景图片	背景图片选择	在图片来源先选择背景图片后添加

使用图片原始尺寸	是否使用图片原始尺寸	False 图片自适应元件大小，True 元件适应图片原始尺寸
水平位置	元件的水平起始位置	不要超出水平分辨率
垂直位置	元件的垂直起始位置	不要超出垂直分辨率
宽度	元件的宽度	/
高度	元件的高度	/

例一：读取视觉通道图片

1．Basic 文件读取图片并保存到视觉通道 0

```
GLOBAL ZVOBJECT Image

ZV_READIMAGE(Image,"test.bmp",0) '从默认路径读取图片

ZV_LATCH(Image,0)                    '图片存入视觉锁存通道 0
```

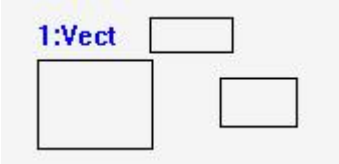
2．新建图片元件，属性的图片来源选择@ZV0

显示效果：

图片元件显示图片，图片大小默认由元件大小决定，可选择使用图片原始尺寸。

3.8.矢量图形

选择“矢量图形”元件后，在建立时先画出一个方框，弹出文件选择窗口，打开系统盘的矢量文件填充到元件内显示。



1. 属性窗口：

2. 属性说明：

属性	功能	说明
元件编号	/	/
元件名称	/	/
显示层次	选择元件显示层次	TopLayer：表层，显示在最外层，覆盖底下元件 MidLayer：中间层 BottomLayer：底层(默认)
有效显示	选择元件是否显示	默认 True，选择 False 时，元件不显示且无功能作用
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 Ture 才会出现下方的三个参数
设备编号	有效控制的设备编号	默认 local
寄存器类型	选择寄存器类型	多种寄存器下拉列表选择

寄存器编号	选择寄存器编号	寄存器值为 0 时不显示，非 0 时使用
当前颜色	选择线段颜色	/
线宽	线段的宽度	默认宽度为 1
线段类型	线段的样式	下拉列表选择实线或虚线等类型
填充	是否填充颜色	默认 False 不填充
水平位置	元件的水平起始位置	不要超出水平分辨率
垂直位置	元件的垂直起始位置	不要超出垂直分辨率
宽度	元件的宽度	/
高度	元件的高度	/

3.9.位元件

3.9.1. 位状态显示

元件主要功能为根据寄存器的位状态 0 或 1 来显示格式文本 0 或 1，不能通过直接按下元件来切换显示状态，可通过“动作”设置后，按下元件调用“动作”切换显示状态，。

双击元件打开属性，可以设置两种状态要显示的文本，只能显示寄存器位状态，还可以调用函数，元件按下时调用。

选择位寄存器时，寄存器值为 0 显示格式文本 0，寄存器值为 1 显示格式文本 1，如果选择的寄存器是不是位寄存器，那么寄存器值为 0 显示格式文本 0，寄存器值不为 1 显示格式文本 1，多个位的显示使用字元件。

1. 属性窗口：

2. 属性说明：

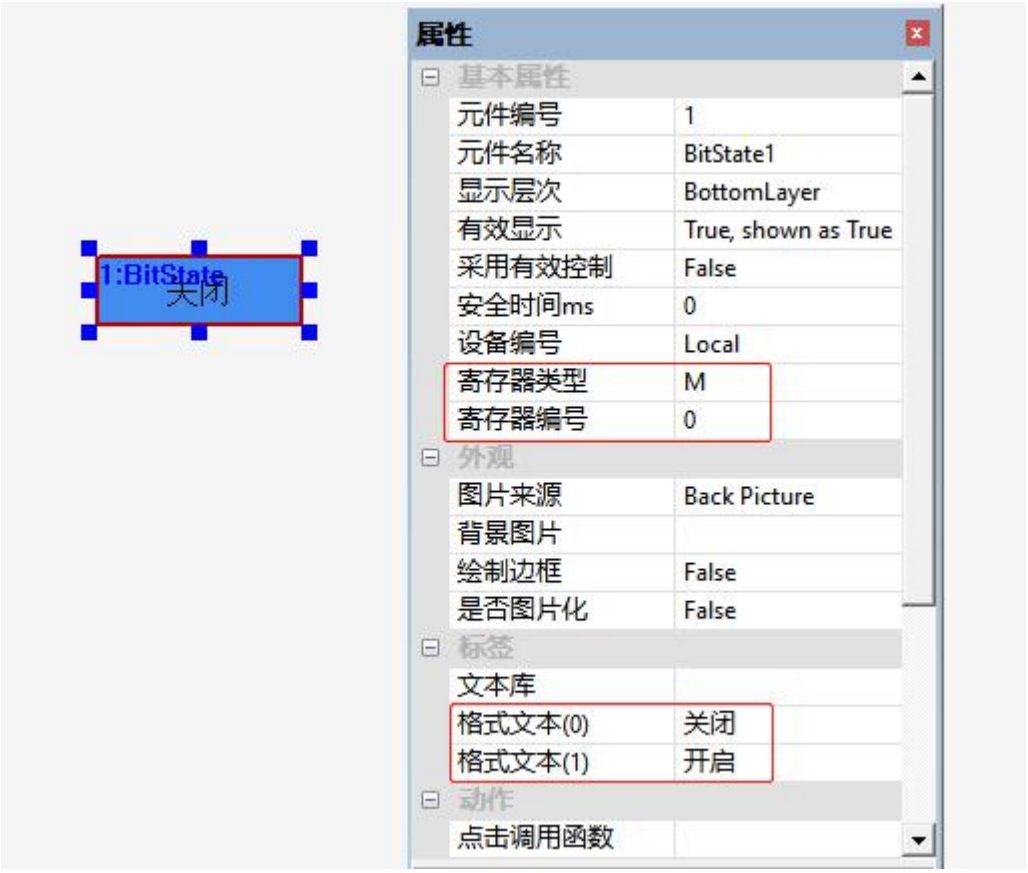
属性	功能	说明
元件编号	/	/
元件名称	/	/
显示层次	选择元件显示层次	TopLayer：表层，显示在最外层，覆盖底下元件 MidLayer：中间层 BottomLayer：底层(默认)
有效显示	选择元件是否显示	默认 True，选择 False 时，元件不显示且无功能作用
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 Ture 通过寄存器控制元件是否显示

设备编号	设备编号	默认 local
寄存器类型	选择寄存器类型	多种寄存器下拉列表选择
寄存器编号	选择寄存器编号	寄存器值为 0 时不显示，非 0 时使用
图片来源	Back PictureLib 或 Back Picture	图片库或背景图片中选择
背景图片	背景图片选择	在图片来源先选择背景图片后添加
绘制边框	选择是否绘制边框	/
是否图片化	元件变为图片的形式	默认 False
文本库	文本库的名称	不设置文本库显示格式文本
格式文本 0/1	打开格式文本设置窗口设置元件要显示的文本	寄存器值为 0 显示文本 0，寄存器值不为 0 时显示文本 1
点击调用函数	按键按下时调用函数	下拉框选择可以调用的函数名
水平位置	元件的水平起始位置	不要超出水平分辨率
垂直位置	元件的垂直起始位置	不要超出垂直分辨率
宽度	元件的宽度	/
高度	元件的高度	/

例一：根据寄存器的不同状态来显示不同的文本

1．选择寄存器类型和编号，M0 对应 MODBUS_BIT(0)；

2．在格式文本中填入不同状态对应的文本。



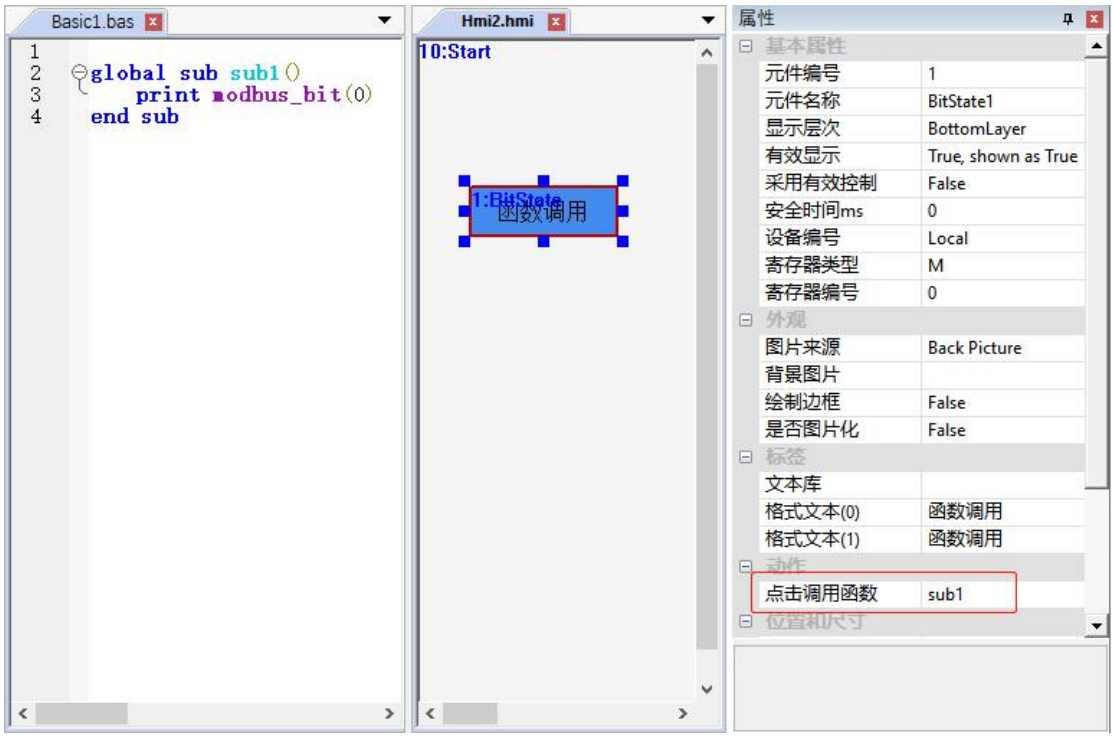
实现效果：

当 MODBUS_BIT(0)=0 时，元件显示“关闭”，当 MODBUS_BIT(0)=1 时，元件显示“开启”。

例二：调用 SUB 函数

1．在 Basic 里编辑好 HMI 要调用的全局 SUB 子函数。

2．在元件属性的“点击调用函数”处选择上一步编辑好的 SUB 子函数名称。



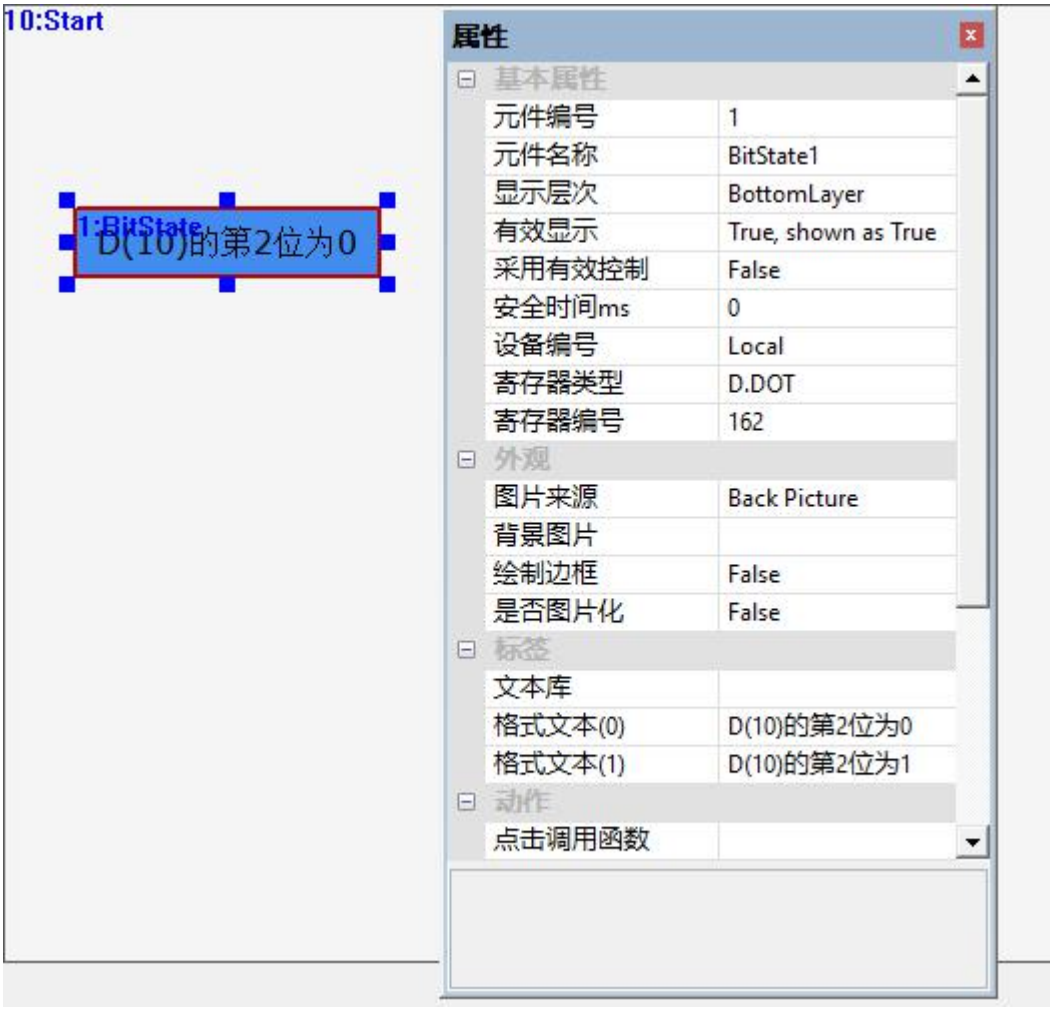
实现效果：

当元件被按下时，调用 Basic 的 SUB 子函数执行，每按下一次调用一次函数。

例三：位切换功能

1．寄存器选择 D.DOT，寄存器编号=reg 编号*16+dot(0-15)，0 到 15 即是 reg 的 0 到 15 位选择；

2．在格式文本中填入不同状态对应的文本。



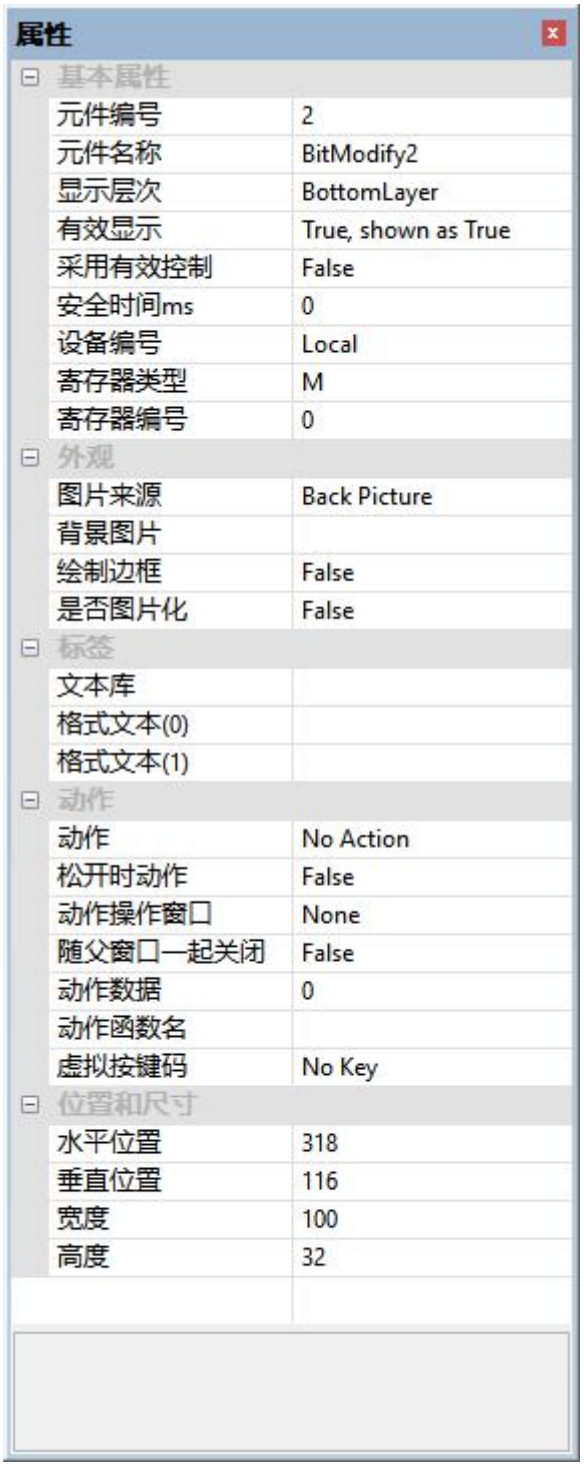
实现效果：

162=10*16+2，即当 D(10)第二位赋予 1 时，元件显示“D(10)的第 2 位为 1”。

3.9.2. 位状态设置

元件显示状态根据元件按下状态来确定，位状态元件只能显示两种状态，初始默认显示格式文本 0，按下时显示格式文本 1，不能通过寄存器的值来切换显示状态，支持调用函数，可选择元件按下或松开时调用。

1. 属性窗口：



2. 属性说明：

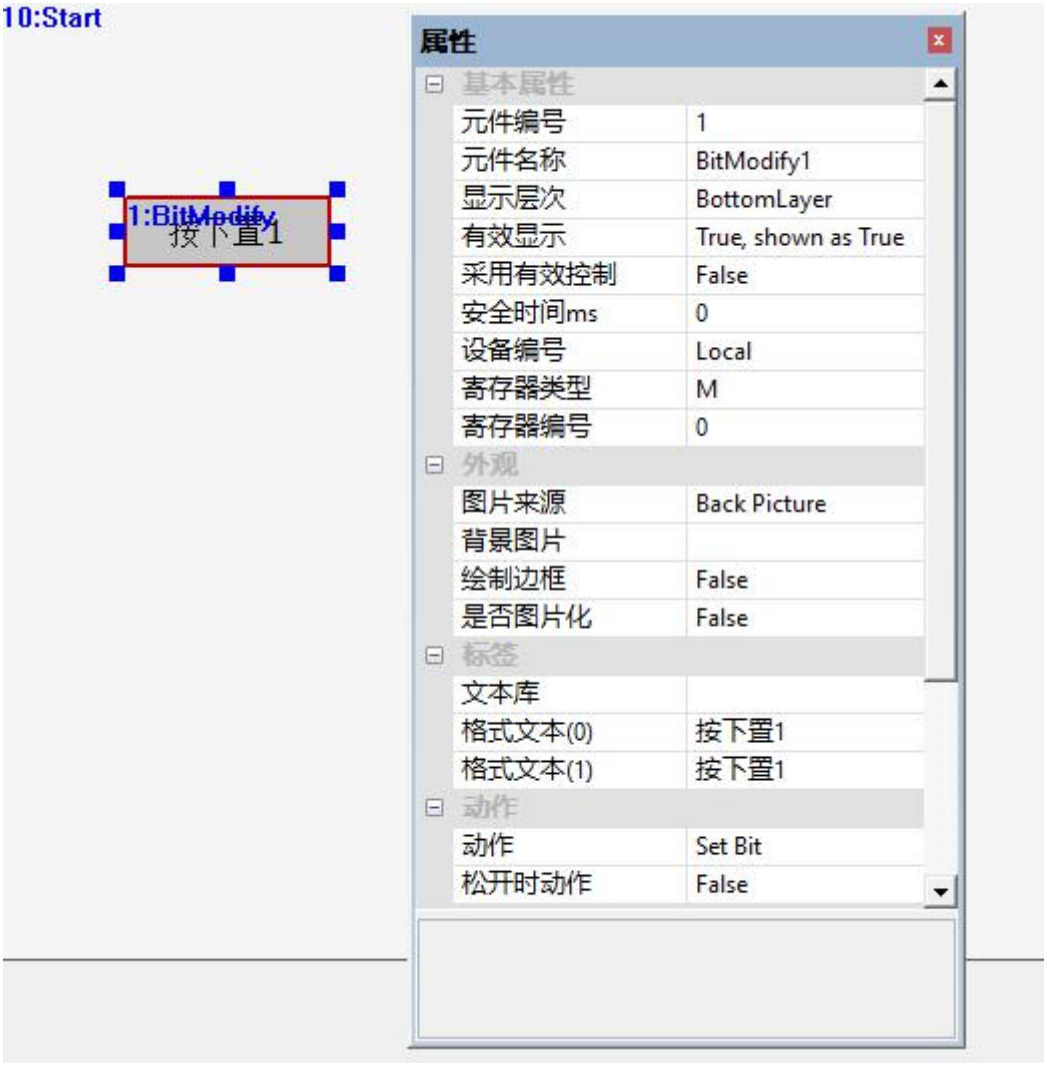
属性	功能	说明
元件编号	/	/
元件名称	/	/
显示层次	选择元件显示层次	TopLayer：表层，显示在最外层，覆盖底下元件

		MidLayer：中间层 BottomLayer：底层(默认)
有效显示	选择元件是否显示	默认 True，选择 False 时，元件不显示且无功能作用
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 Ture 通过寄存器控制元件是否显示
安全时间 ms	最少按键时间	单位 ms
设备编号	设备编号	默认 local
寄存器类型	选择寄存器类型	多种寄存器下拉列表选择
寄存器编号	选择寄存器编号	寄存器值为 0 时不显示，非 0 时使用
图片来源	Back PictureLib 或 Back Picture	图片库或背景图片中选择
背景图片	背景图片选择	在图片来源先选择背景图片后添加
绘制边框	选择是否绘制边框	/
是否图片化	元件变为图片的形式	默认 False
文本库	文本库的名称	不设置文本库显示格式文本
格式文本 0/1	打开格式文本设置窗口设置元件要显示的文本	默认显示文本 0，按下时显示文本 1
动作	按键执行时的动作	参见“动作”章节描述
松开时动作	选择按下时或松开时执行动作	默认 False 为按下执行动作，Ture 为松开时动作
动作操作窗口	选择需要操作的窗口编号	下拉列表选择已有窗口
随父窗口一起关闭	子窗口随父窗口一起关闭	默认 False
动作数据	按键动作后给寄存器写入指定值	/
动作函数名	按键动作后要调用的 SUB 函数	下拉列表选择 Basic 已有全局 SUB 函数
虚拟按键码	选择虚拟按键码	默认不选择
水平位置	元件的水平起始位置	不要超出水平分辨率
垂直位置	元件的垂直起始位置	不要超出垂直分辨率
宽度	元件的宽度	/
高度	元件的高度	/

例一：给寄存器赋值 1

1．选择寄存器类型和编号；

2．“动作”选择“Set Bit”。



实现效果：

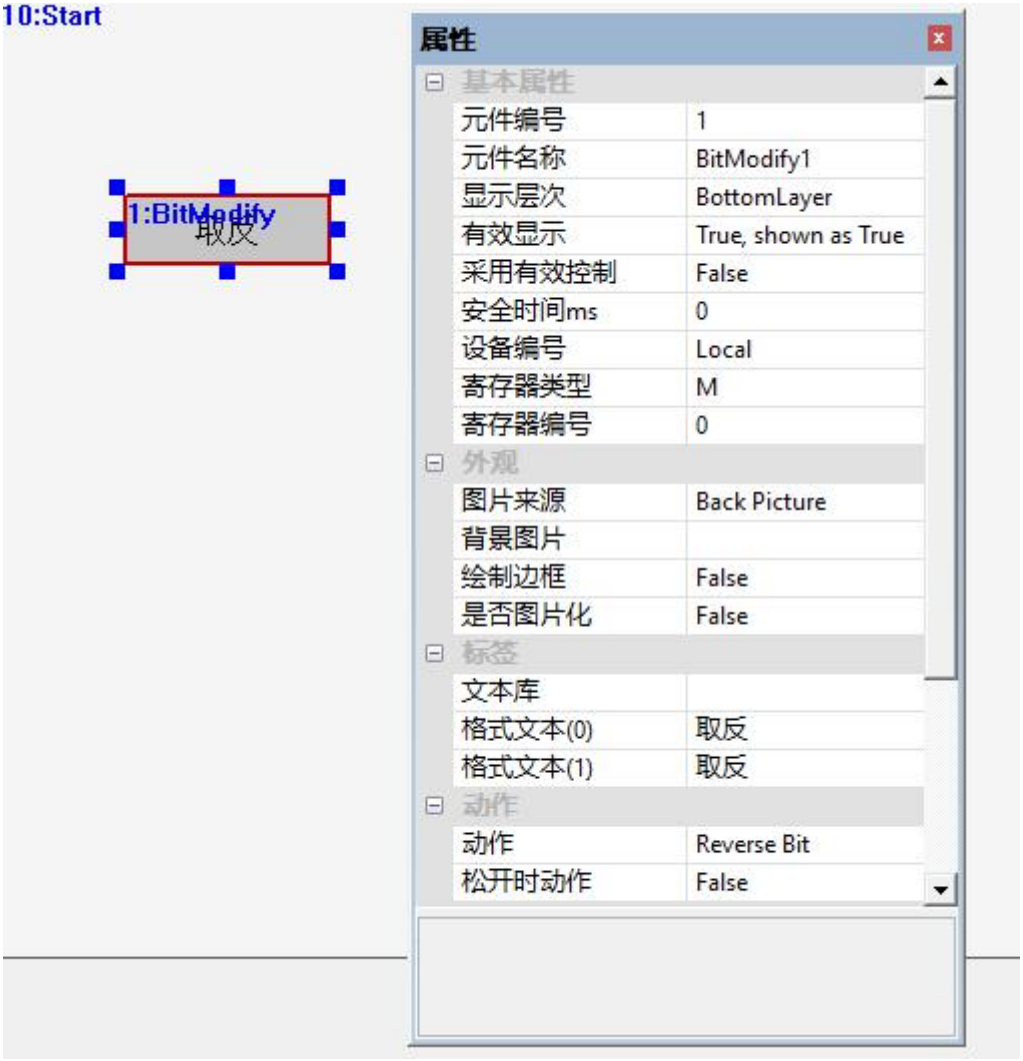
按下元件时，MODBUS_BIT(0)=1，如果选择了松开时动作，即按下元件再松开后，MODBUS_BIT(0)=1，MODBUS_BIT(0)的值为保持为 1。

动作“Reset Bit”为将寄存器置 0，与“Set Bit”相反。

例二：寄存器数值取反

1．选择寄存器类型和编号；

2．“动作”选择“Reverse Bit”。



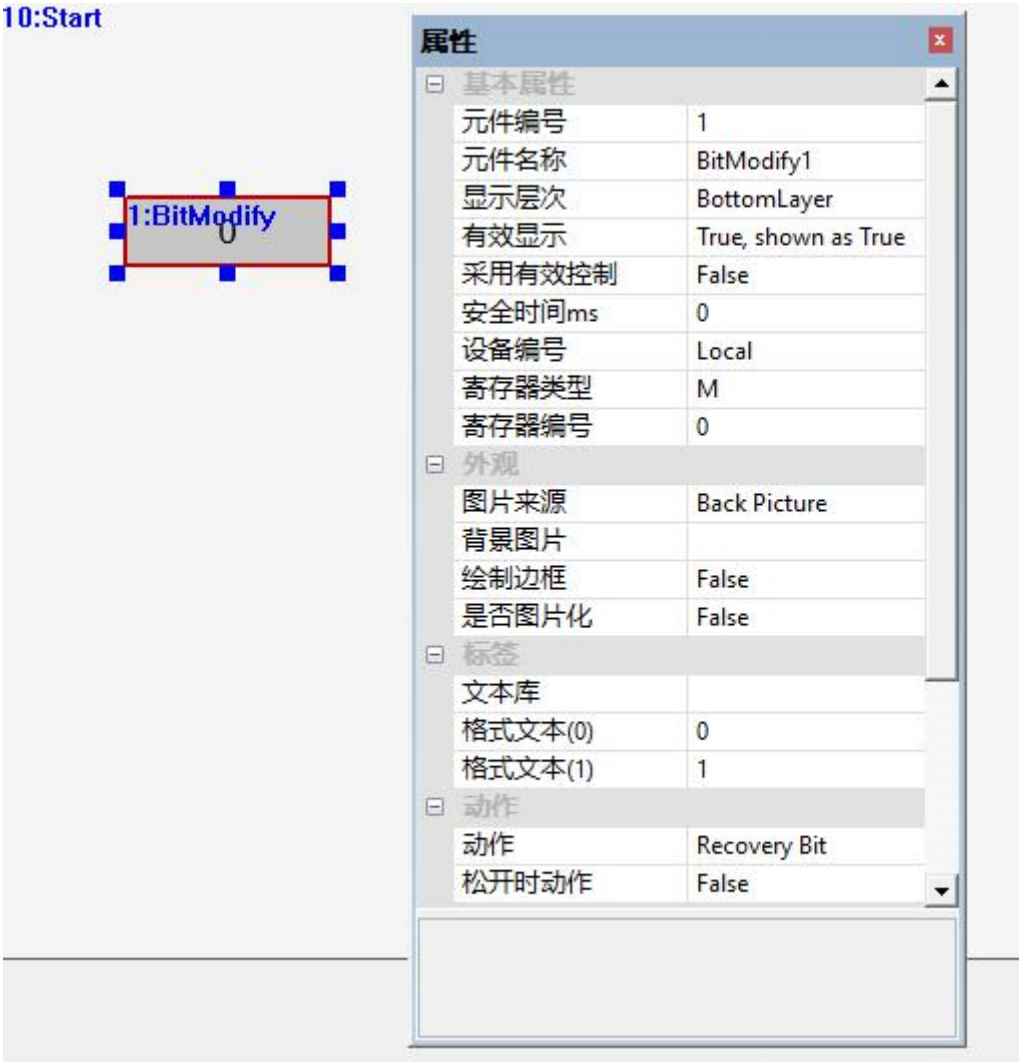
实现效果：

如果 MODBUS_BIT(0)的初始值为 0，按下元件后取反，MODBUS_BIT(0)=1，再次按下元件，MODBUS_BIT(0)=0。

例三：按下时寄存器置 1，松开时寄存器置 0

1．选择寄存器类型和编号；

2．“动作”选择“Recovery Bit”。



实现效果：

按下元件后，MODBUS_BIT(0)=1，松开元件后，MODBUS_BIT(0)=0。

例四：调用 SUB 函数

例程参见功能键例一。

3.9.3. 位状态切换开关

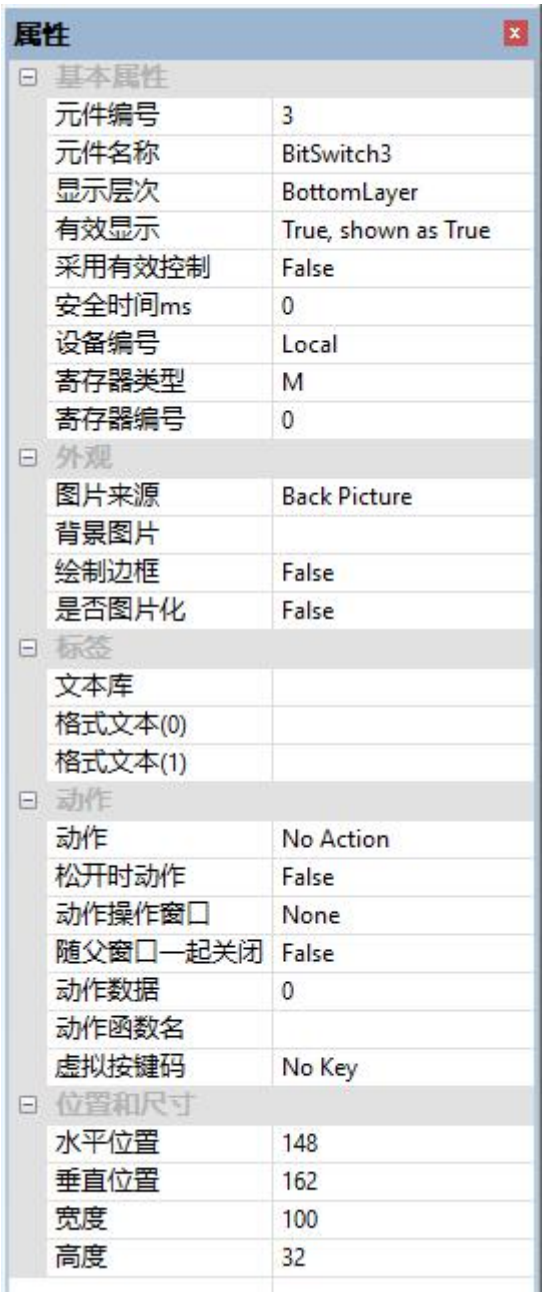
元件主要功能与“位状态显示”相似，仅是“动作”功能有区别，根据寄存器的位状态 0 或 1 来显示格式文本 0 或 1，不能通过直接按下元件来切换显示状态，可通过“动作”设置后，按下元件调用动作切换显示状态。

双击元件打开属性，设置两种状态要显示的文本，只能显示寄存器位状态。

“动作”功能不仅可以调用函数，可选择元件按下或松开时调用；还可以设置寄存器的值。

选择位寄存器时，寄存器值为 0 显示格式文本 0，寄存器值为 1 显示格式文本 1，如果选择的寄存器是不是位寄存器，那么寄存器值为 0 显示格式文本 0，寄存器值不为 1 显示格式文本 1，多个位的显示使用字元件。

1. 属性窗口：



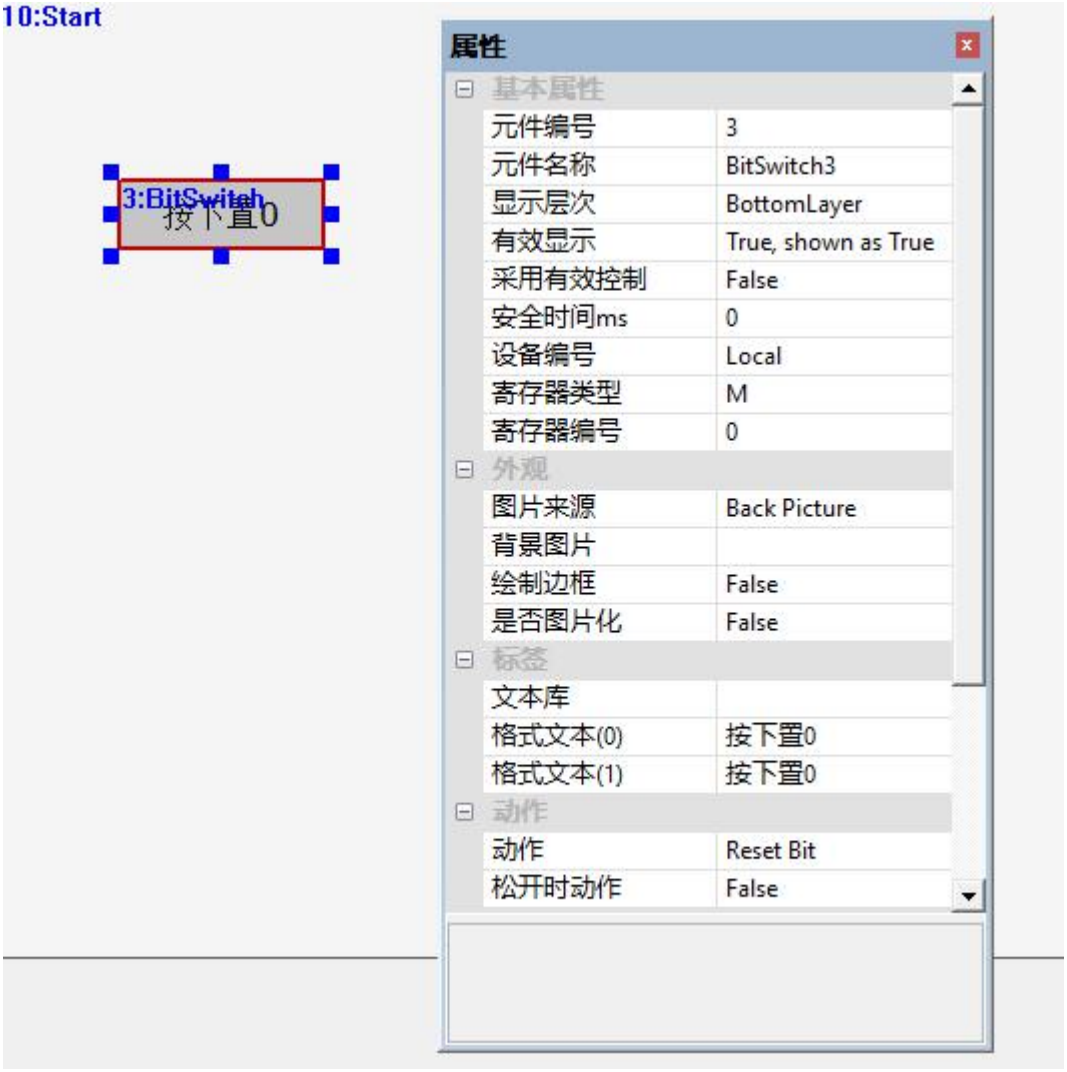
2. 属性说明：

属性	功能	说明
元件编号	/	/
元件名称	/	/
显示层次	选择元件显示层次	TopLayer：表层，显示在最外层，覆盖底下元件 MidLayer：中间层 BottomLayer：底层(默认)
有效显示	选择元件是否显示	默认 True，选择 False 时，元件不显示且无功能作用
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 Ture 通过寄存器控制元件是否显示
安全时间 ms	最少按键时间	单位 ms
设备编号	设备编号	默认 local
寄存器类型	选择寄存器类型	多种寄存器下拉列表选择
寄存器编号	选择寄存器编号	寄存器值为 0 时不显示，非 0 时使用
图片来源	Back PictureLib 或 Back Picture	图片库或背景图片中选择
背景图片	背景图片选择	在图片来源先选择背景图片后添加
绘制边框	选择是否绘制边框	默认 False
是否图片化	元件变为图片的形式	默认 False
文本库	文本库的名称	不设置文本库显示格式文本
格式文本 0/1	打开格式文本设置窗口设置元件要显示的文本	默认显示文本 0，按下时显示文本 1
动作	按键执行时的动作	参见“动作”章节描述
松开时动作	选择按下时或松开时执行动作	默认 False 为按下执行动作，Ture 为松开时动作
动作操作窗口	选择需要操作的窗口编号	下拉列表选择已有窗口
随父窗口一起关闭	子窗口随父窗口一起关闭	默认 False
动作数据	按键动作后给寄存器写入指定值	/
动作函数名	按键动作后要调用的 SUB 函数	下拉列表选择 Basic 已有全局 SUB 函数
虚拟按键码	选择虚拟按键码	默认不选择
水平位置	元件的水平起始位置	不要超出水平分辨率
垂直位置	元件的垂直起始位置	不要超出垂直分辨率
宽度	元件的宽度	/
高度	元件的高度	/

例一：给寄存器赋值 1

1．选择寄存器类型和编号；

2. “动作”选择“Reset Bit”。



实现效果：

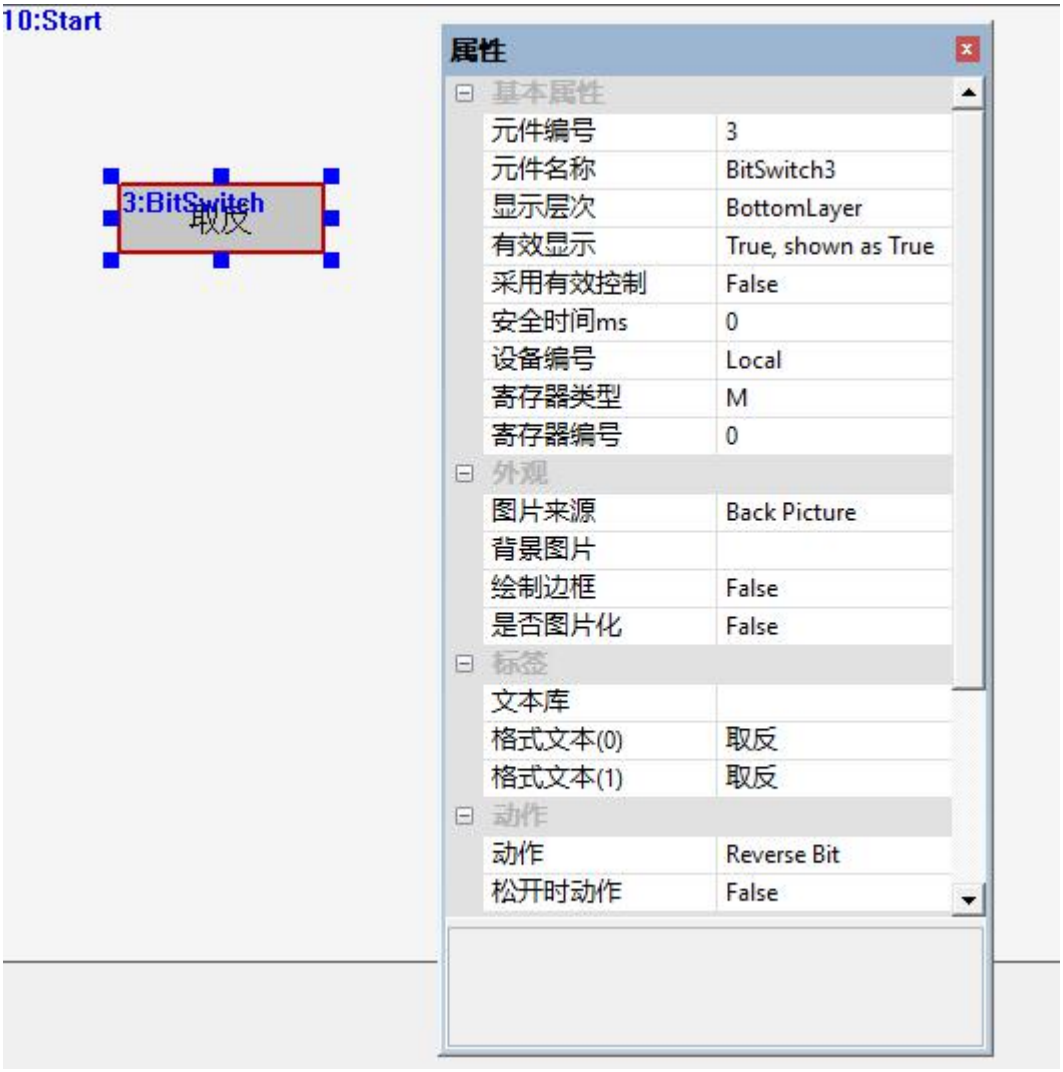
按下元件时，MODBUS_BIT(0)=0，如果选择了松开时动作，即按下元件再松开后，MODBUS_BIT(0)=0，MODBUS_BIT(0)的值为保持为 0。

动作“Set Bit”为将寄存器置 1，与“Reset Bit”相反。

例程二：寄存器数值取反

1. 选择寄存器类型和编号；

2. “动作”选择“Reverse Bit”。



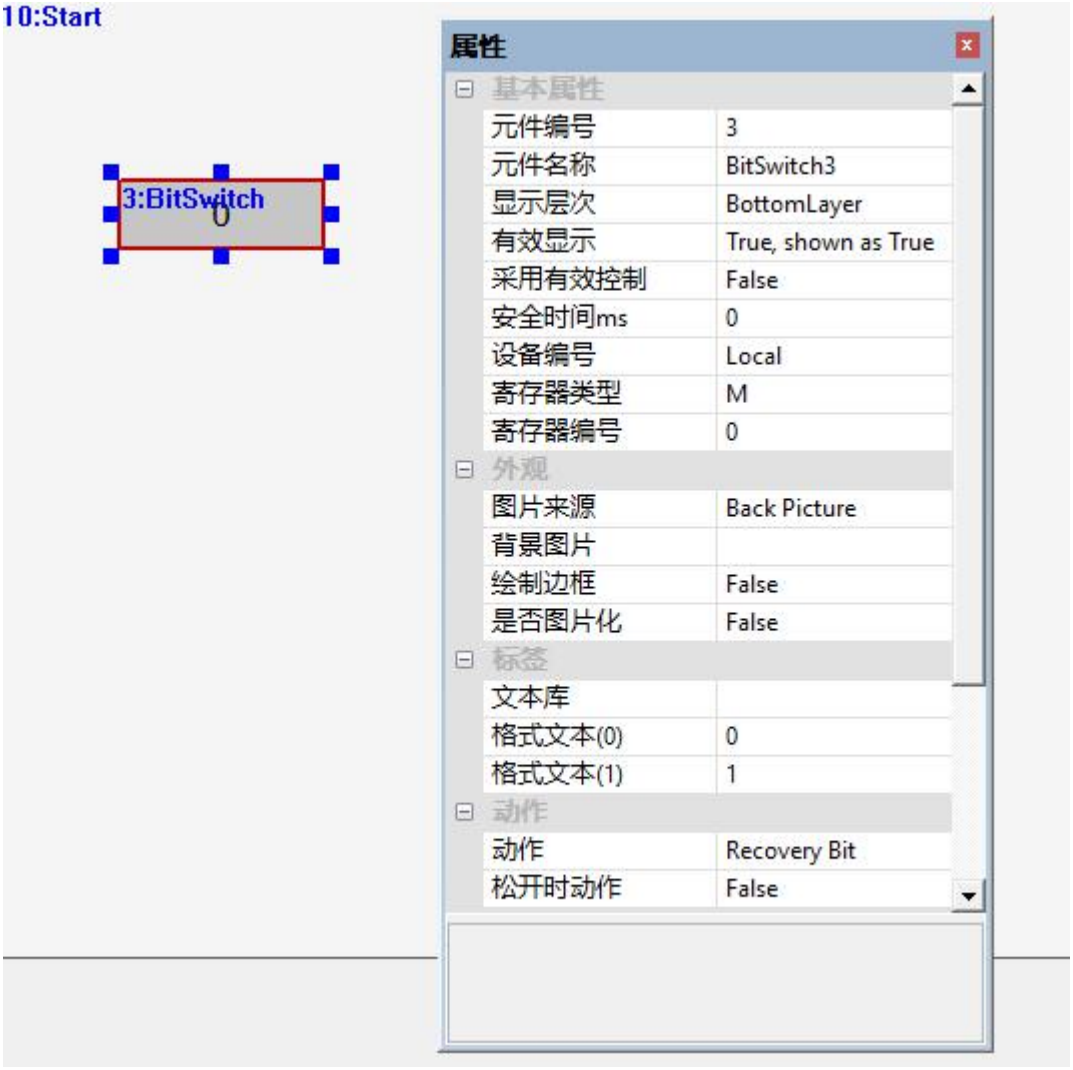
实现效果：

如果 MODBUS_BIT(0)的初始值为 0，按下元件后取反，MODBUS_BIT(0)=1，再次按下元件，MODBUS_BIT(0)=0。

例程三：按下时寄存器置 1，松开时寄存器置 0

1．选择寄存器类型和编号；

2．“动作”选择“Recovery Bit”。



实现效果：

按下元件后，MODBUS_BIT(0)=1，松开元件后，MODBUS_BIT(0)=0。

例四：调用 SUB 函数

例程参见功能键例一。

3.9.4. 功能键

功能键无法绑定寄存器，位状态元件只能显示两种状态，初始默认显示格式文本 0，按下时显示格式文本 1。

功能键的主要功能：

1. 调用 Basic 函数。

2. 打开窗口。

3. 关闭窗口。

通过属性中“动作”来选择功能。

1. 属性窗口：

2. 属性说明：

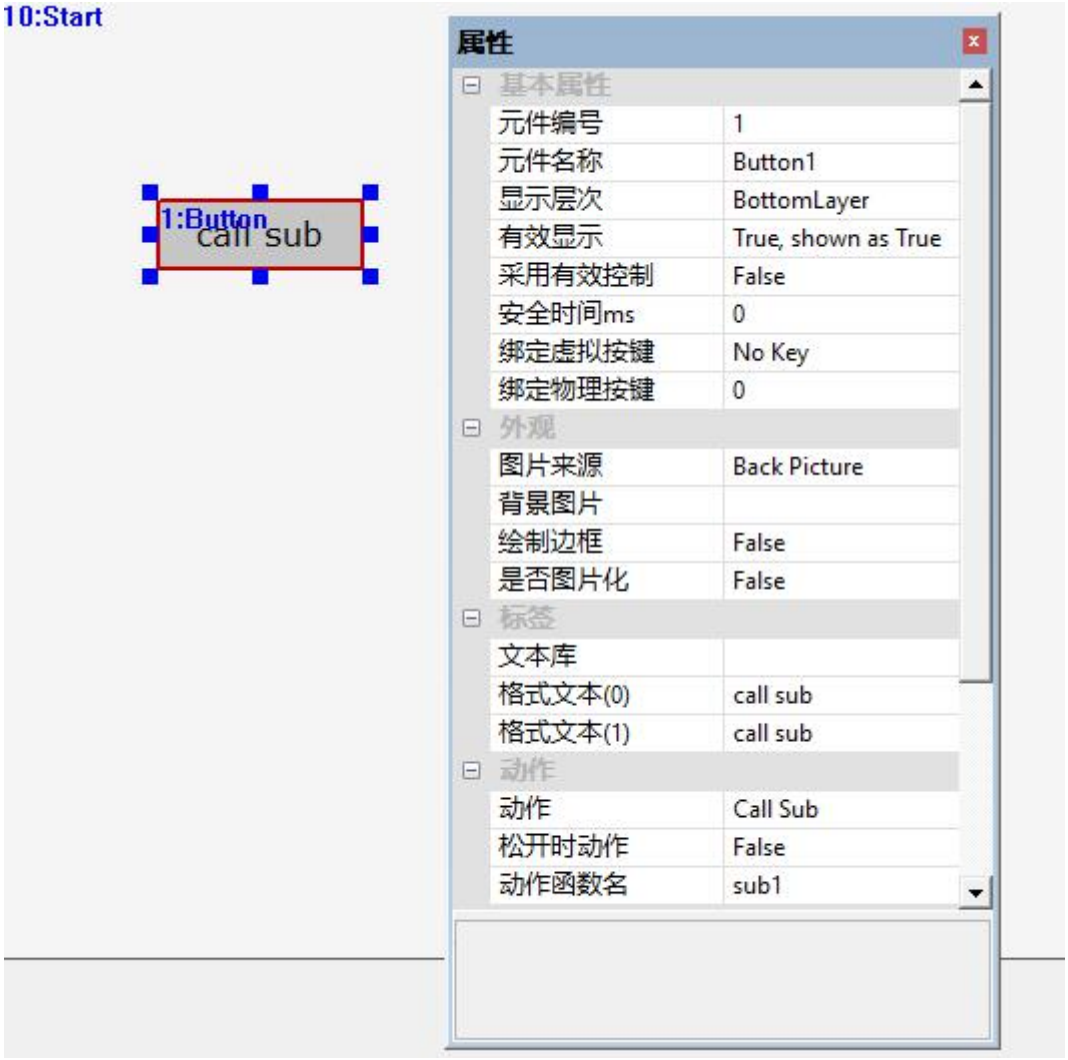
属性	功能	说明
元件编号	/	/
元件名称	/	/
显示层次	选择元件显示层次	TopLayer：表层，显示在最外层，覆盖底下元件 MidLayer：中间层 BottomLayer：底层(默认)
有效显示	选择元件是否显示	默认 True，选择 False 时，元件不显示且无功能作用
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 Ture 通过寄存器控制元件是否显示
安全时间 ms	最少按键时间	单位 ms
绑定虚拟按键	选择要绑定的虚拟按键码	默认不选择
绑定物理按键	绑定示教盒上面的物理按键	按键码值查看“虚拟键”章节
图片来源	Back PictureLib 或 Back Picture	图片库或背景图片中选择
背景图片	背景图片选择	在图片来源先选择背景图片后添加
绘制边框	选择是否绘制边框	/
是否图片化	元件变为图片的形式	默认 False
文本库	文本库的名称	不设置文本库显示格式文本
格式文本 0/1	打开格式文本设置窗口设置元件要显示的文本	默认显示文本 0，按下时显示文本 1
动作	按键执行时的动作	参见“动作”章节描述
松开时动作	选择按下时或松开时执行动作	默认 False 为按下执行动作，Ture 为松开时动作
动作函数名	按键动作后要调用的 SUB 函数	下拉列表选择 Basic 已有全局 SUB 函数
水平位置	元件的水平起始位置	不要超出水平分辨率
垂直位置	元件的垂直起始位置	不要超出垂直分辨率
宽度	元件的宽度	/
高度	元件的高度	/

例一：调用 SUB 函数

1．设置功能键的“动作”为“call sub”；

2．选择要调用的“动作函数名”；

3．选择按下时调用或松开时调用。



被调用的 Basic 函数：

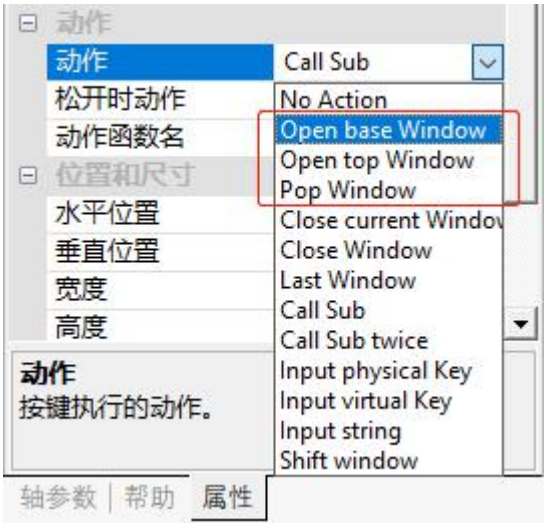
```
global sub sub1()
    print "调用函数"
end sub
```

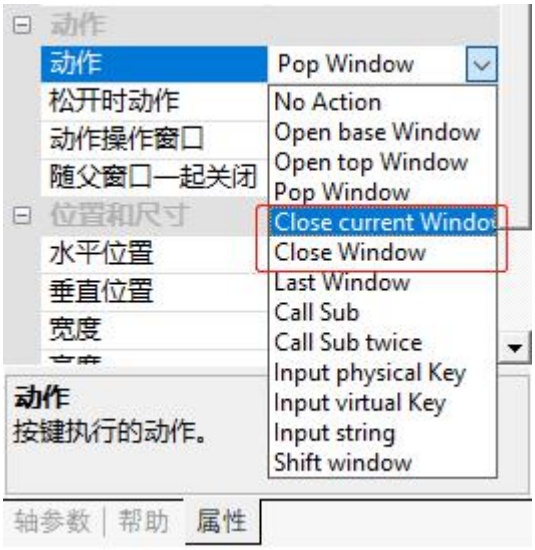
实现效果：

功能键按下时，调用 Basic 函数 sub1 执行打印命令。

例二：打开/关闭窗口

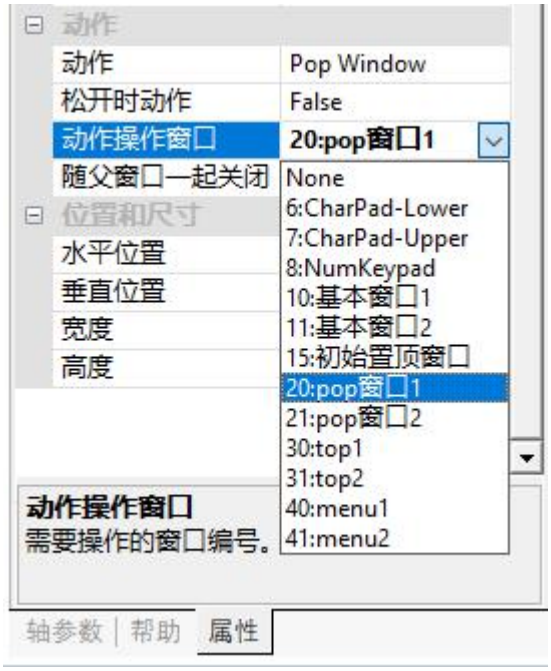
1．在功能键的“动作”选择下图动作操作窗口；





打开窗口　　关闭窗口

2．以打开窗口为例，在功能键的“动作”选择好要打开的窗口类型后，再找到“动作操作窗口”，选择要打开的窗口编号，注意要打开的窗口类型要与选择的窗口一致。



实现效果：

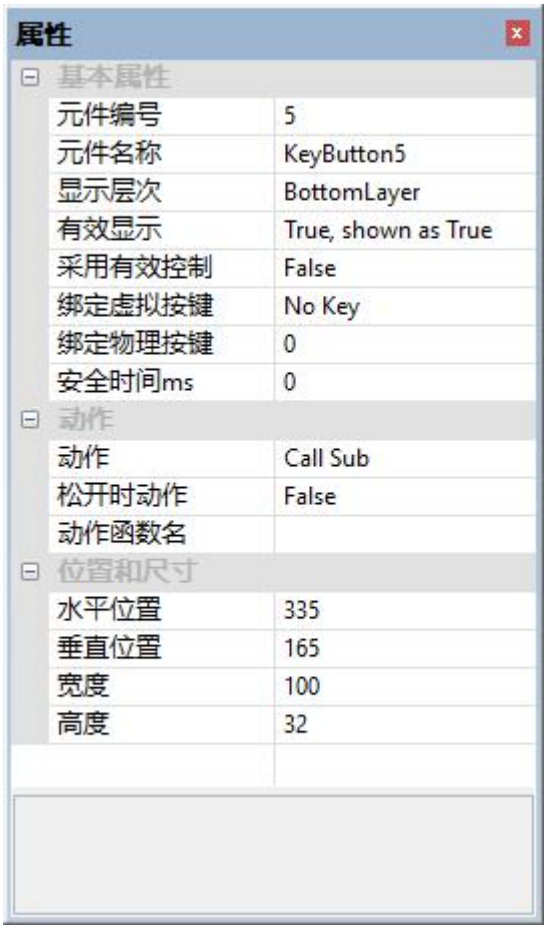
按下功能键，立即打开 20 号窗口。

3.9.5. 物理按键

物理按键只在 hmi 界面编辑时显示，实际运行界面不会显示出来，主要用来绑定虚拟按键和物理按键，其中虚拟按键编号已经设定好，只需要选择物理按键编号即可，物理按键则与实际硬件按钮绑定，程序中具体编号要查看硬件手册。

虚拟按键和物理按键绑定后，设置物理按键要调用的函数，这样在按下触摸屏自带的实际按键时 就能调用该函数。

1. 属性窗口：

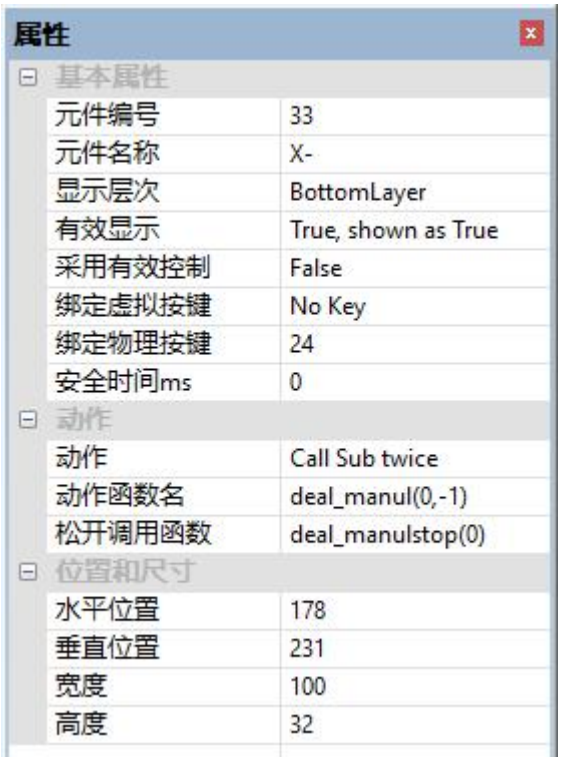


2. 属性说明：

属性	功能	说明
元件编号	/	/
元件名称	/	/
显示层次	选择元件显示层次	TopLayer：表层，显示在最外层，覆

		盖底下元件 MidLayer：中间层 BottomLayer：底层(默认)
有效显示	选择元件是否显示	默认 True，选择 False 时，元件不显示且无功能作用
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 Ture 通过寄存器控制元件是否显示
绑定虚拟按键	选择要绑定的虚拟按键码	默认不选择
绑定物理按键	绑定示教盒上面的物理按键	按键码值查看“虚拟键”章节
安全时间 ms	最少按键时间	单位 ms
动作	按键执行时的动作	参见“动作”章节描述
松开时动作	选择按下时或松开时执行动作	默认 False 为按下执行动作，Ture 为松开时动作
动作函数名	按键动作后要调用的 SUB 函数	下拉列表选择 Basic 已有全局 SUB 函数
水平位置	元件的水平起始位置	不要超出水平分辨率
垂直位置	元件的垂直起始位置	不要超出垂直分辨率
宽度	元件的宽度	/
高度	元件的高度	/

例程：将外部物理按键 X-（对应的物理按键码值 24）绑定到虚拟按键，并在属性窗口设置外部物理按键按下后要调用的函数，运行时，按下外部物理按键，立即执行调用的函数。



3.10. 字元件

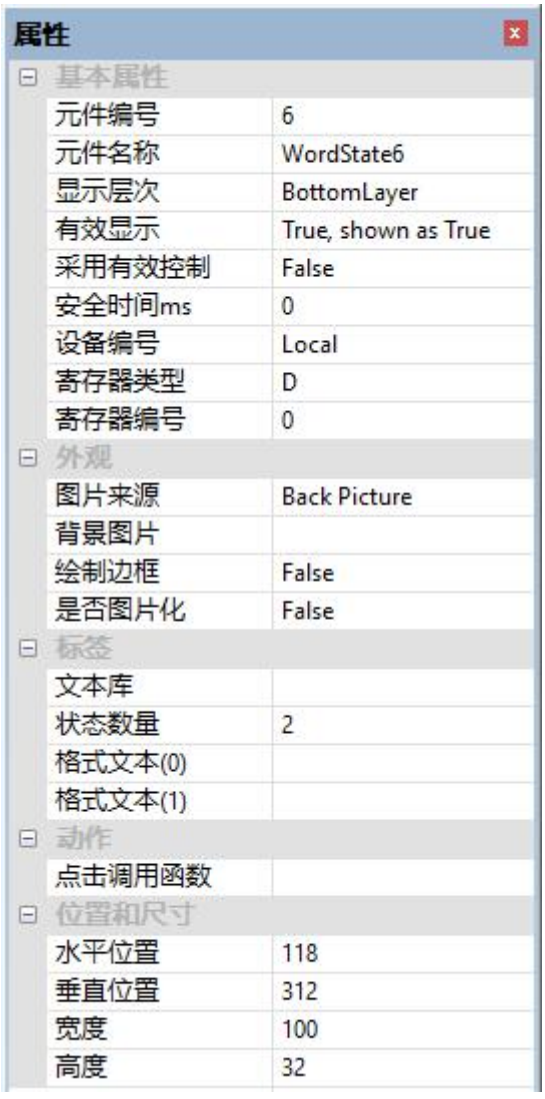
3.10.1. 多状态显示

主要作用是根据寄存器的不同状态来显示不同的文本或是调用 SUB 函数。

可显示的文本数量根据“状态数量”自定义设置，数量范围 0-255。

建议选择字寄存器控制，默认 D0，对应寄存器值为 0 时，显示格式文本 0；值为 1 时，显示格式文本 1；依此类推，一一对应显示。

1. 属性窗口：



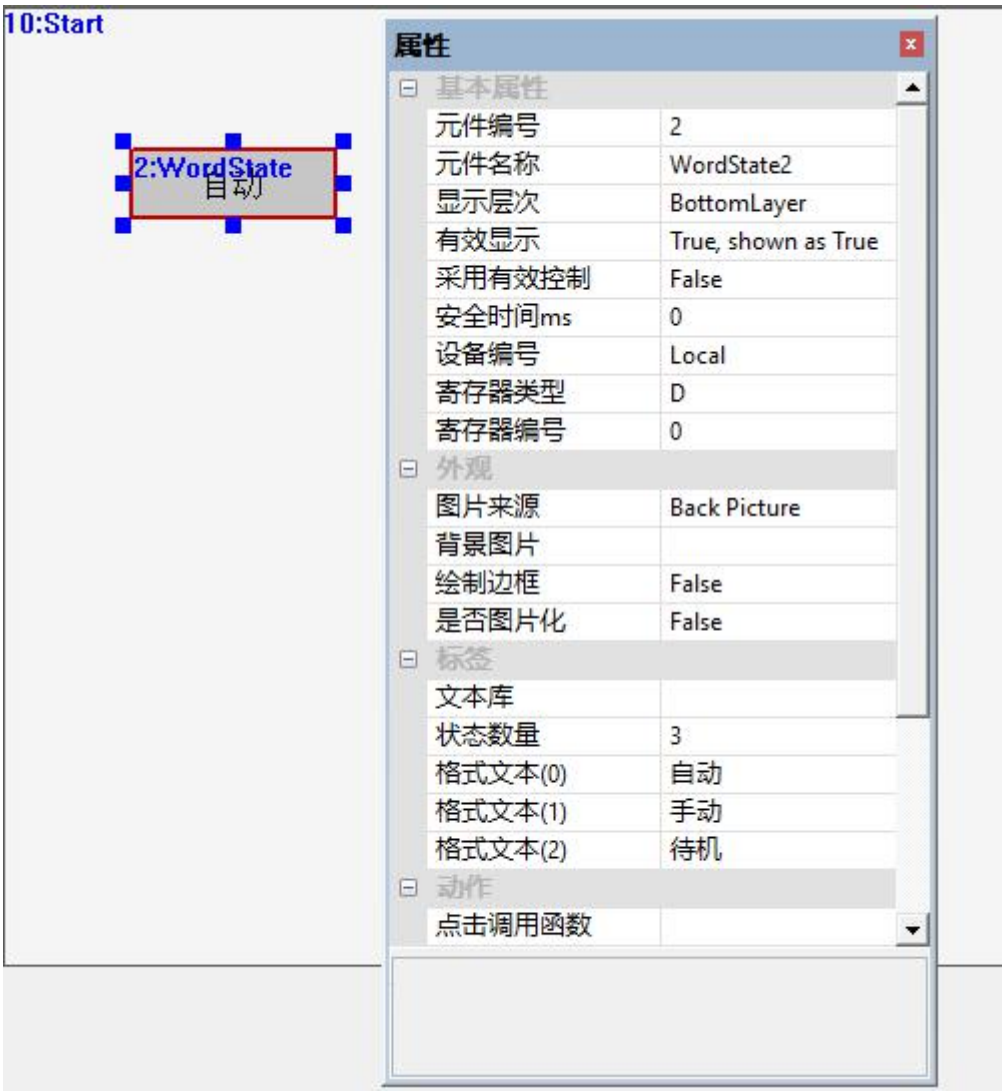
2. 属性说明：

属性	功能	说明
元件编号	/	/
元件名称	/	/
显示层次	选择元件显示层次	TopLayer：表层，显示在最外层，覆盖底下元件 MidLayer：中间层 BottomLayer：底层(默认)
有效显示	选择元件是否显示	默认 True，选择 False 时，元件不显示且无功能作用
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 Ture 通过寄存器控制元件是否显示
安全时间 ms	最少按键时间	单位 ms
设备编号	设备编号	默认 local
寄存器类型	选择寄存器类型	多种寄存器下拉列表选择
寄存器编号	选择寄存器编号	寄存器值为 0 时不显示，非 0 时使用
图片来源	Back PictureLib 或 Back Picture	图片库或背景图片中选择
背景图片	背景图片选择	在图片来源先选择背景图片后添加
绘制边框	选择是否绘制边框	/
是否图片化	元件变为图片的形式	默认 False
文本库	文本库的名称	不设置文本库显示格式文本
状态数量	元件的状态数量(0-255)	可显示多个状态
格式文本	打开格式文本设置窗口设置元件要显示的文本	格式文本的数量由状态数量决定
点击调用函数	按键按下时调用函数	下拉框选择可以调用的函数名
水平位置	元件的水平起始位置	不要超出水平分辨率
垂直位置	元件的垂直起始位置	不要超出垂直分辨率
宽度	元件的宽度	/
高度	元件的高度	/

例一：根据寄存器的不同状态来显示不同的文本

1．选择寄存器类型和编号，默认字寄存器 D0，即 MODBUS_REG(0)；

2．选择状态数量，在“格式文本”中填入不同状态对应的文本。



实现效果：

当 MODBUS_REG(0)=0 时，元件显示“自动”；

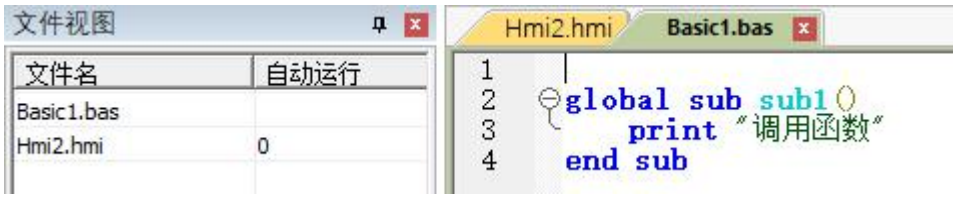
当 MODBUS_REG(0)=1 时，元件显示“手动”；

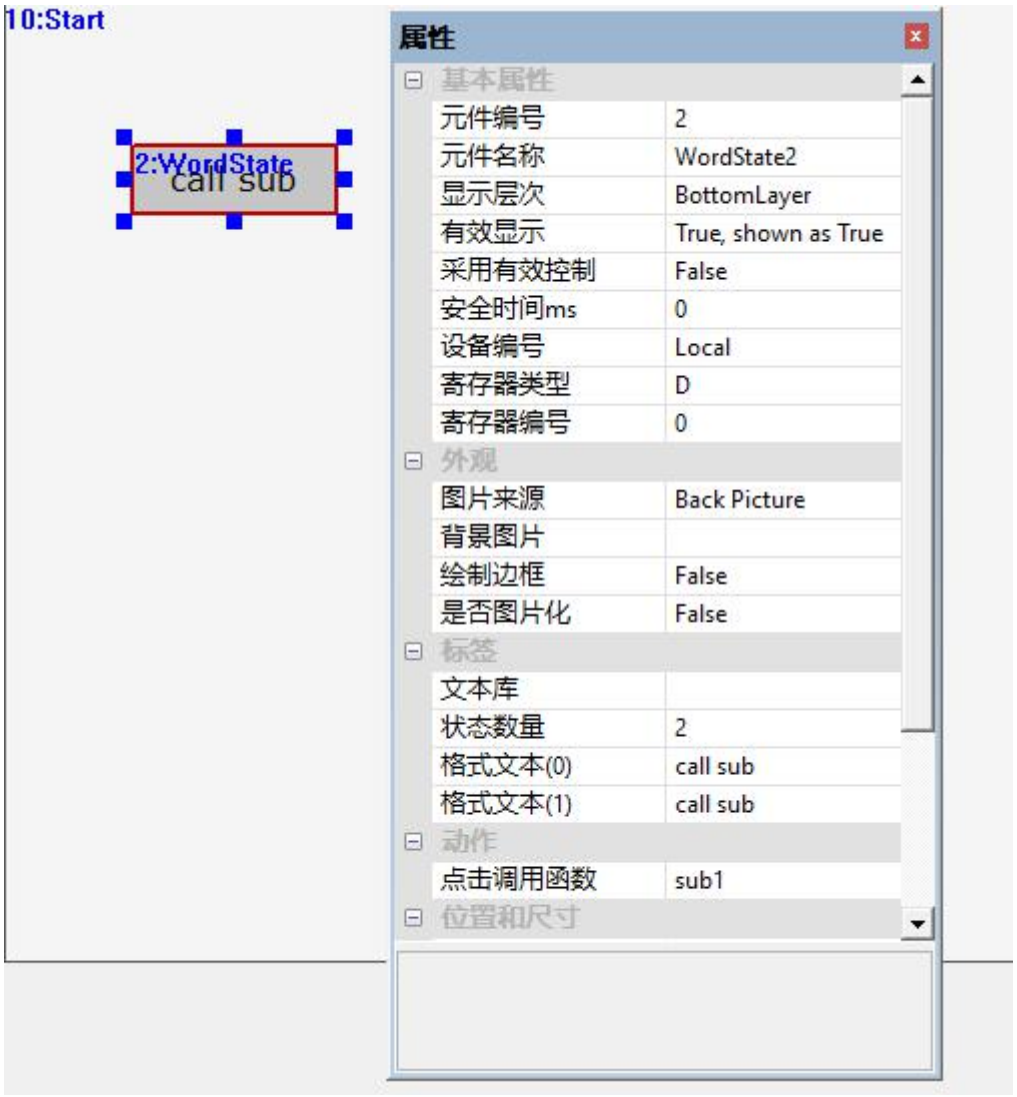
当 MODBUS_REG(0)=2 时，元件显示“待机”。

例程二：调用 SUB 函数

1．在 Basic 文件中，编写一个全局的 SUB 函数；

2．在元件属性“动作”选择调用的 sub 函数。





实现效果：

按下元件时，执行 SUB 函数内打印代码。

3.10.2. 多状态设置

用于对寄存器进行操作，也可以调用函数。详细使用方法参见例程。

1. 属性窗口：

2. 属性说明：

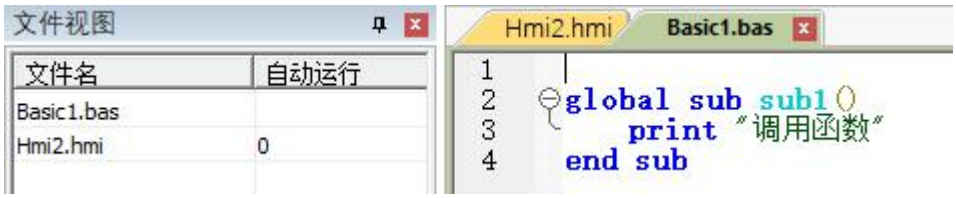
属性	功能	说明
元件编号	/	/
元件名称	/	/
显示层次	选择元件显示层次	TopLayer：表层，显示在最外层，覆

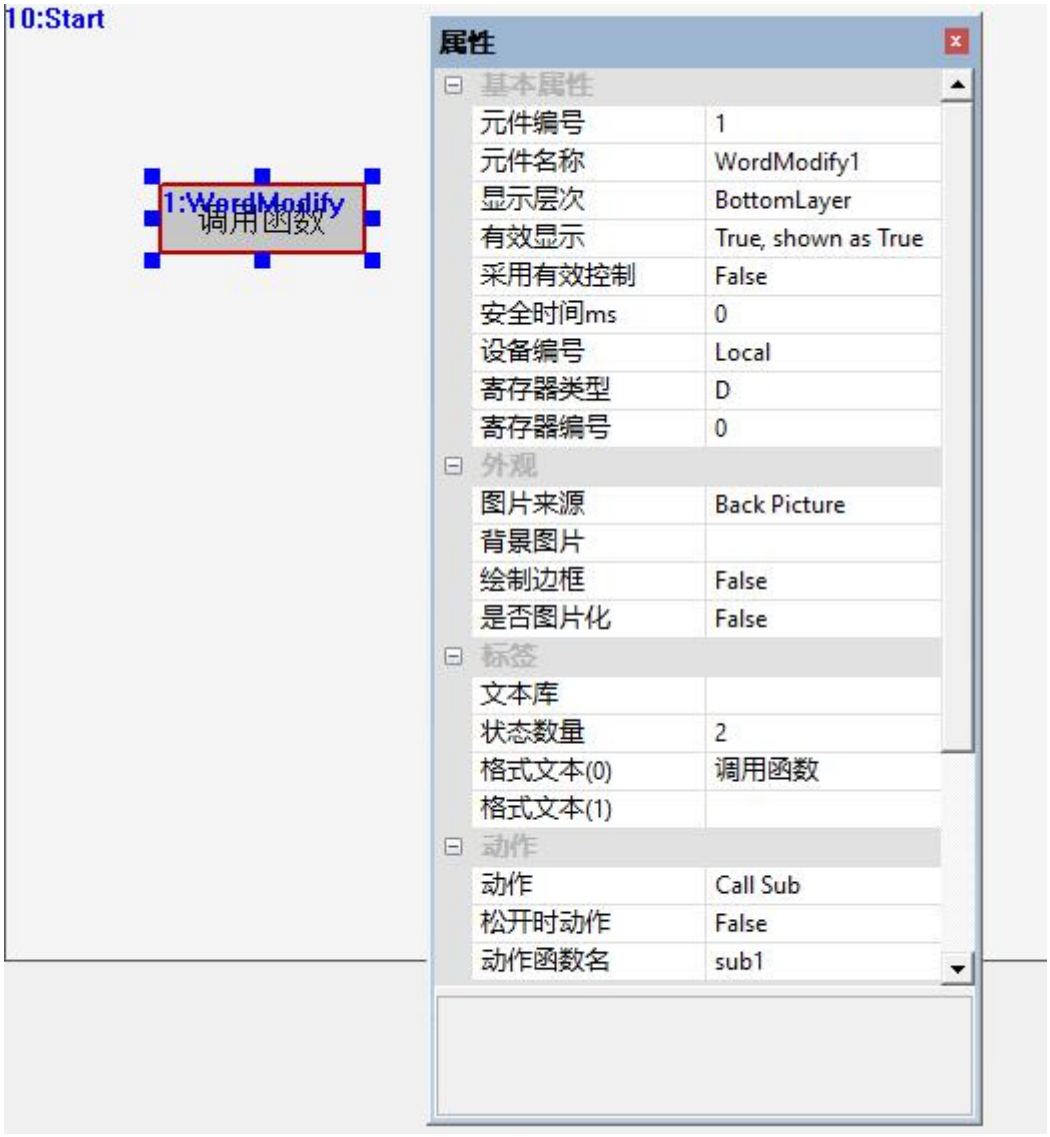
		盖底下元件 MidLayer：中间层 BottomLayer：底层(默认)
有效显示	选择元件是否显示	默认 True，选择 False 时，元件不显示且无功能作用
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 Ture 通过寄存器控制元件是否显示
安全时间 ms	最少按键时间	单位 ms
设备编号	设备编号	默认 local
寄存器类型	选择寄存器类型	多种寄存器下拉列表选择
寄存器编号	选择寄存器编号	寄存器值为 0 时不显示，非 0 时使用
图片来源	Back PictureLib 或 Back Picture	图片库或背景图片中选择
背景图片	背景图片选择	在图片来源先选择背景图片后添加
绘制边框	选择是否绘制边框	默认 False
是否图片化	元件变为图片的形式	默认 False
文本库	文本库的名称	不设置文本库显示格式文本
状态数量	元件的状态数量(0-255)	可显示多个状态
格式文本	打开格式文本设置窗口设置元件要显示的文本	格式文本的数量由状态数量决定
动作	按键执行时的动作	参见“动作”章节描述
松开时动作	选择按下时或松开时执行动作	默认 False 为按下执行动作，Ture 为松开时动作
动作操作窗口	选择需要操作的窗口编号	下拉列表选择已有窗口
随父窗口一起关闭	子窗口随父窗口一起关闭	默认 False
动作数据	按键动作后给寄存器写入指定值	/
动作函数名	按键动作后要调用的 SUB 函数	下拉列表选择 Basic 已有全局 SUB 函数
虚拟按键码	选择虚拟按键码	默认不选择
水平位置	元件的水平起始位置	不要超出水平分辨率
垂直位置	元件的垂直起始位置	不要超出垂直分辨率
宽度	元件的宽度	/
高度	元件的高度	/

例一：调用 SUB 函数

1．在 Basic 文件中，编写一个全局的 SUB 函数；

2．在元件属性，动作选择“Call Sub”，“动作函数名”选择对应的 SUB 函数名。





实现效果：

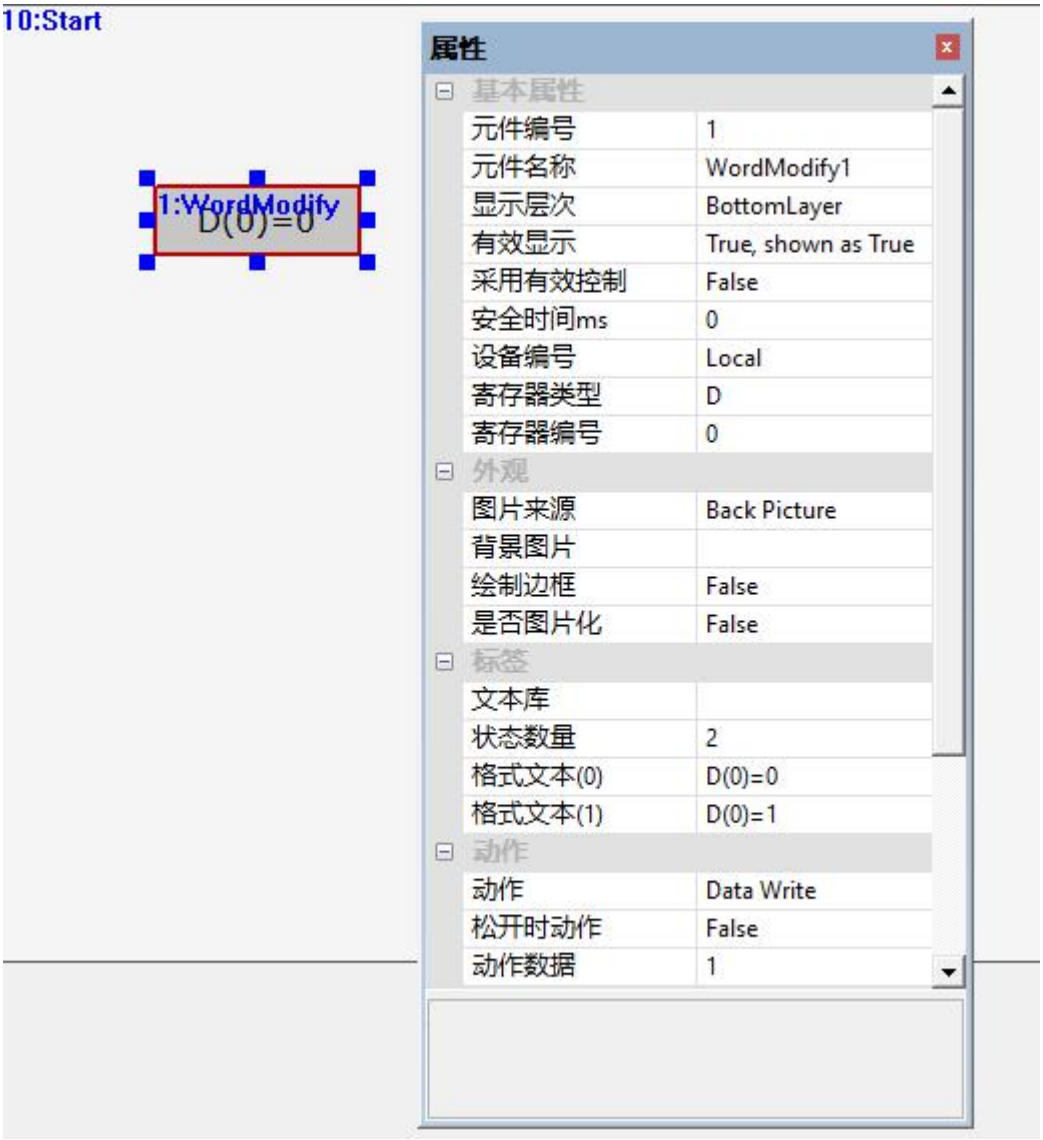
按下元件时，执行 SUB 函数内打印代码。

例二：写入数据到寄存器

1．选择寄存器类型和编号

2．选择动作“Data Write”，设置“动作数据”写入寄存器的值。

状态支持显示两种，按下时显示“格式文本 0”，松开显示“格式文本 1”。



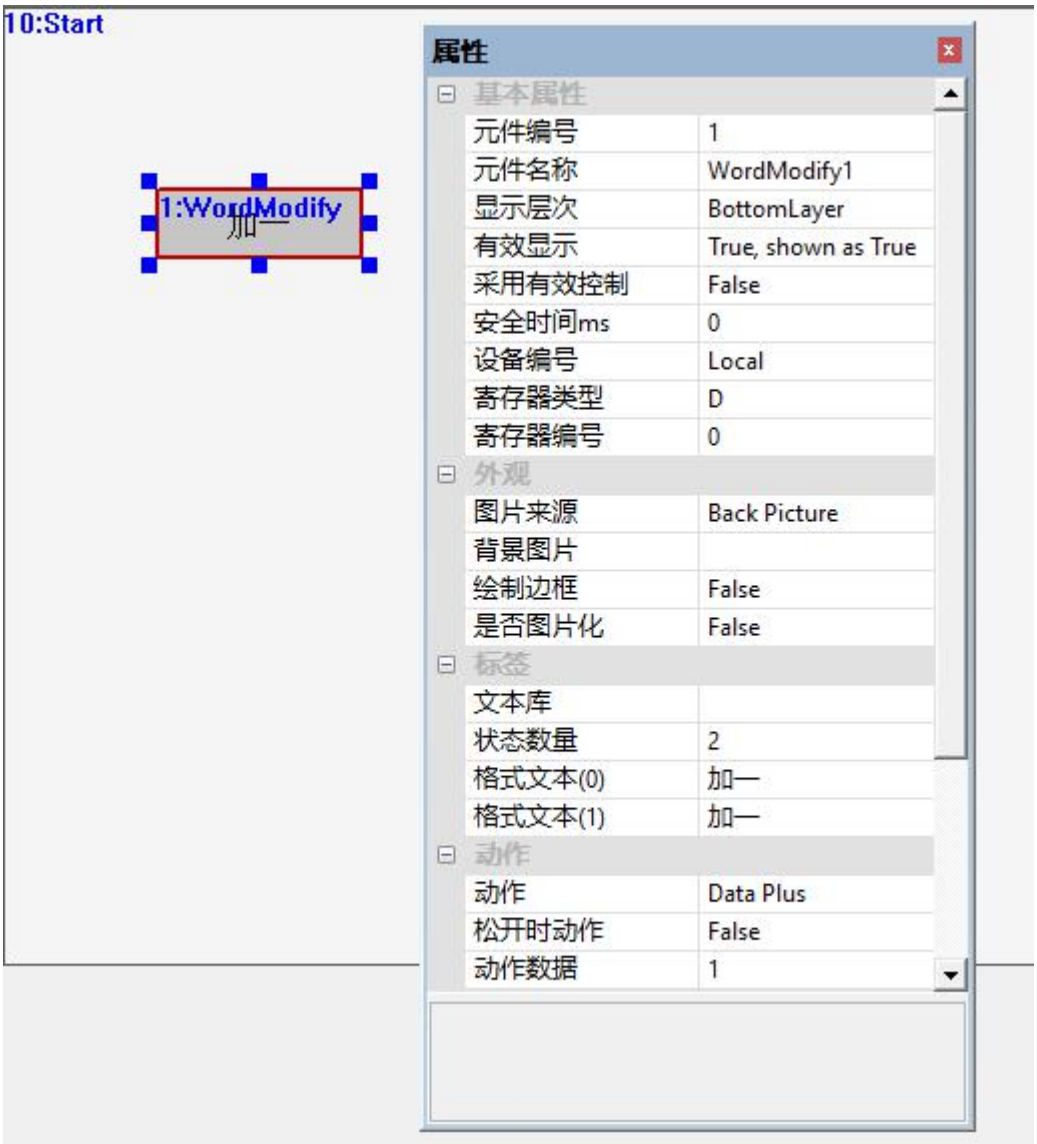
实现效果：

按下元件后，MODBUS_REG(0)写入数据 1。

例三：寄存器在原来的值上加上动作数据的值

1．选择寄存器类型和编号；

2．选择动作“Data Plus”，“动作数据”填入每次寄存器要增加的数据。



实现效果：

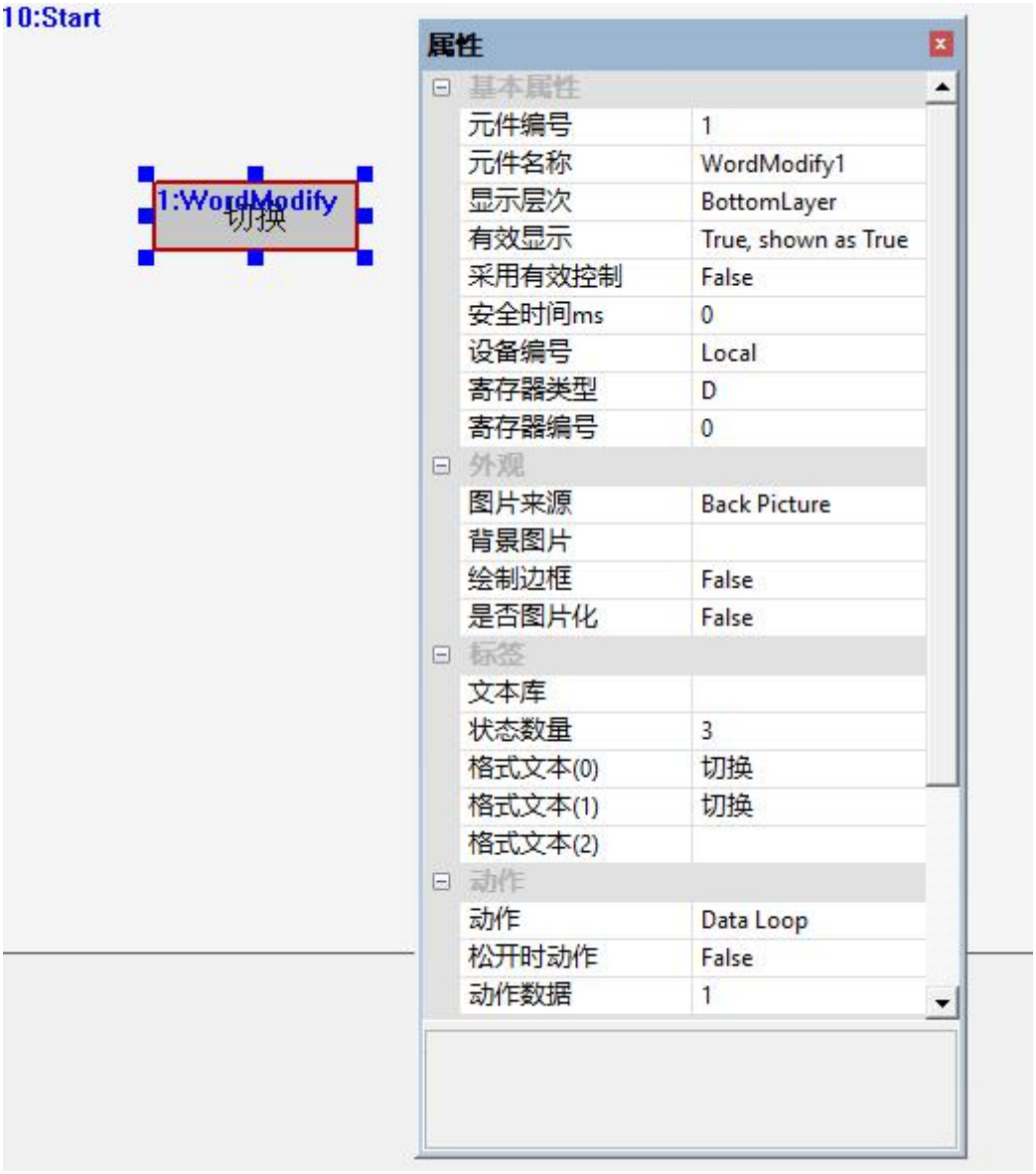
每按下元件一次后，MODBUS_REG(0)等于原来的值加 1。

当寄存器的值超过状态数量设置时，元件不显示，但触摸仍有作用效果。

例四：寄存器原来值加上数据，在设置的状态之间循环切换

1．选择寄存器类型和编号；

2．选择动作“Data Loop”，“动作数据”填入要增加的数据。



实现效果：

按下元件后，MODBUS_REG(0)的值在 0、1、2 之间切换。

如果 MODBUS_REG(0)的初始值大于 2 时，每按一下递减 2，直到寄存器的值为 0 后，开始在 0 到 2 之间切换。

寄存器的值根据状态的数量循环，比如：状态数量 3，动作数据 1，对应的寄存器值在 0、1、2 之前切换；状态数量 5，动作数据 2，对应的寄存器值在 0、2、4、1、3 之前切换。

3.10.3. 多状态切换开关

根据寄存器的不同状态显示对应的文本，状态数量可以自己设置。

8:WordSwitch

1. 属性窗口：

2. 属性说明：

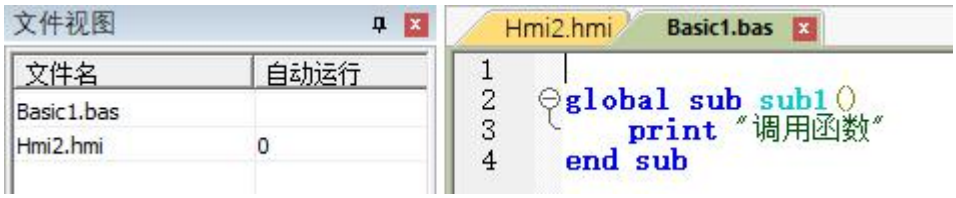
属性	功能	说明
元件编号	/	/

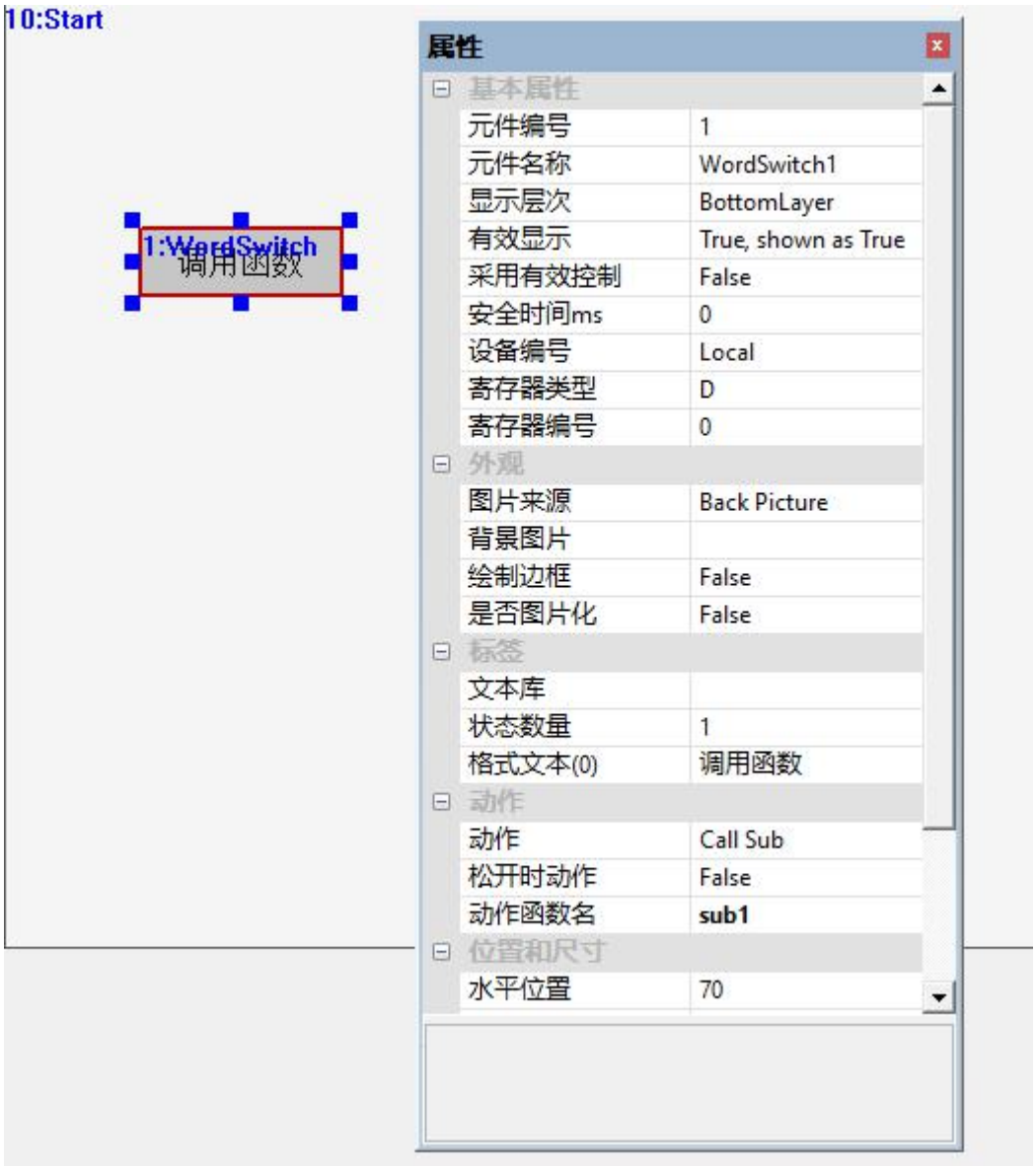
元件名称	/	/
显示层次	选择元件显示层次	TopLayer：表层，显示在最外层，覆盖底下元件 MidLayer：中间层 BottomLayer：底层(默认)
有效显示	选择元件是否显示	默认 True，选择 False 时，元件不显示且无功能作用
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 Ture 通过寄存器控制元件是否显示
安全时间 ms	最少按键时间	单位 ms
设备编号	设备编号	默认 local
寄存器类型	选择寄存器类型	多种寄存器下拉列表选择
寄存器编号	选择寄存器编号	寄存器值为 0 时不显示，非 0 时使用
图片来源	Back PictureLib 或 Back Picture	图片库或背景图片中选择
背景图片	背景图片选择	在图片来源先选择背景图片后添加
绘制边框	选择是否绘制边框	/
是否图片化	元件变为图片的形式	默认 False
文本库	文本库的名称	不设置文本库显示格式文本
状态数量	元件的状态数量(0-255)	可显示多个状态
格式文本	打开格式文本设置窗口设置元件要显示的文本	格式文本的数量由状态数量决定
动作	按键执行时的动作	参见“动作”章节描述
松开时动作	选择按下时或松开时执行动作	默认 False 为按下执行动作，Ture 为松开时动作
动作操作窗口	选择需要操作的窗口编号	下拉列表选择已有窗口
随父窗口一起关闭	子窗口随父窗口一起关闭	默认 False
动作数据	按键动作后给寄存器写入指定值	/
动作函数名	按键动作后要调用的 SUB 函数	下拉列表选择 Basic 已有全局 SUB 函数
虚拟按键码	选择虚拟按键码	默认不选择
水平位置	元件的水平起始位置	不要超出水平分辨率
垂直位置	元件的垂直起始位置	不要超出垂直分辨率
宽度	元件的宽度	/
高度	元件的高度	/

例一：调用 SUB 函数

1．在 Basic 文件中，编写一个全局的 SUB 函数；

2．在元件属性，动作选择“Call Sub”，“动作函数名”选择对应的 SUB 函数名。





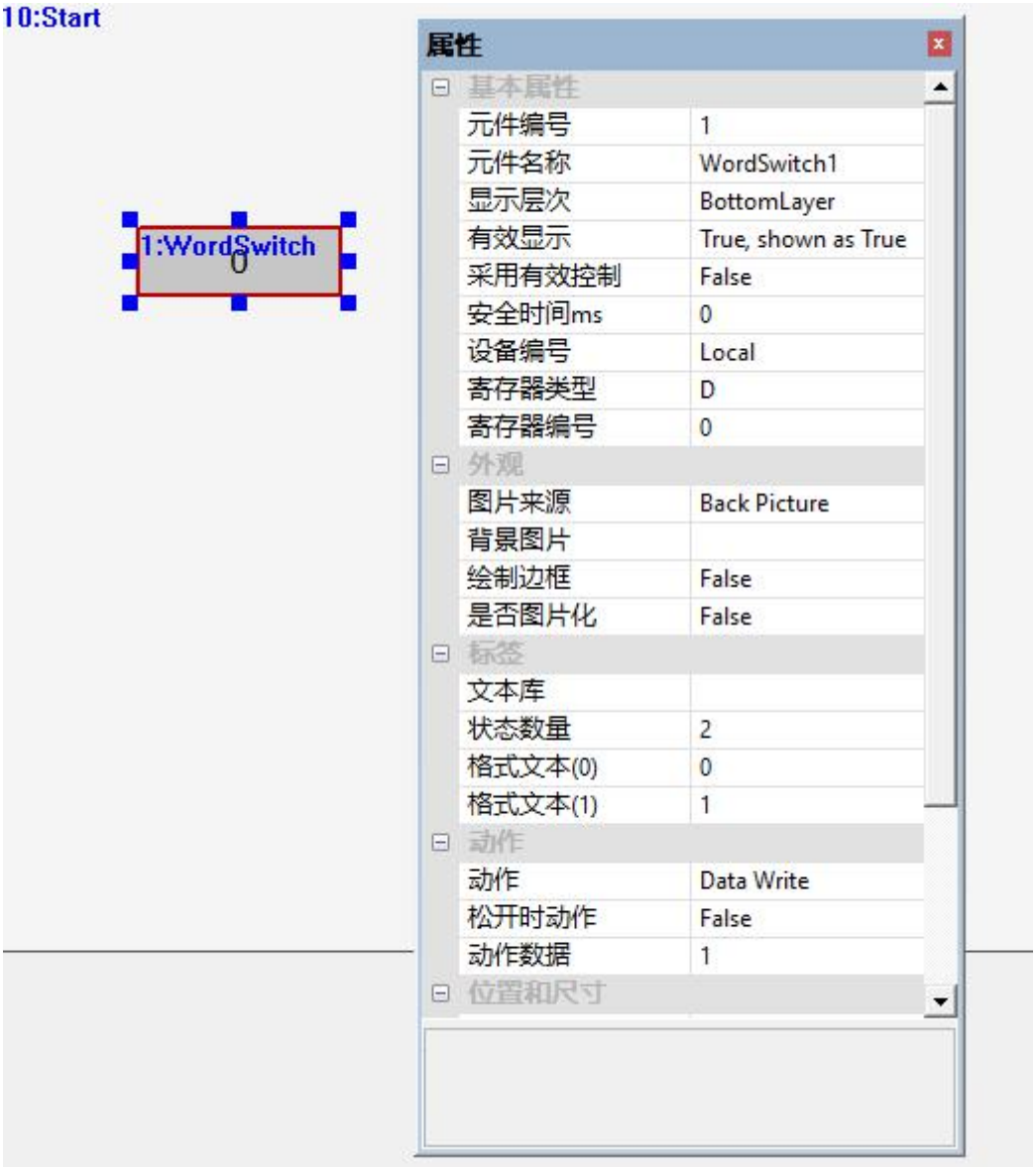
实现效果：

按下元件时，执行 SUB 函数内打印代码。

例二：写入数据到寄存器

1．选择寄存器类型和编号

2．选择动作“Data Write”，设置“动作数据”写入寄存器的值。



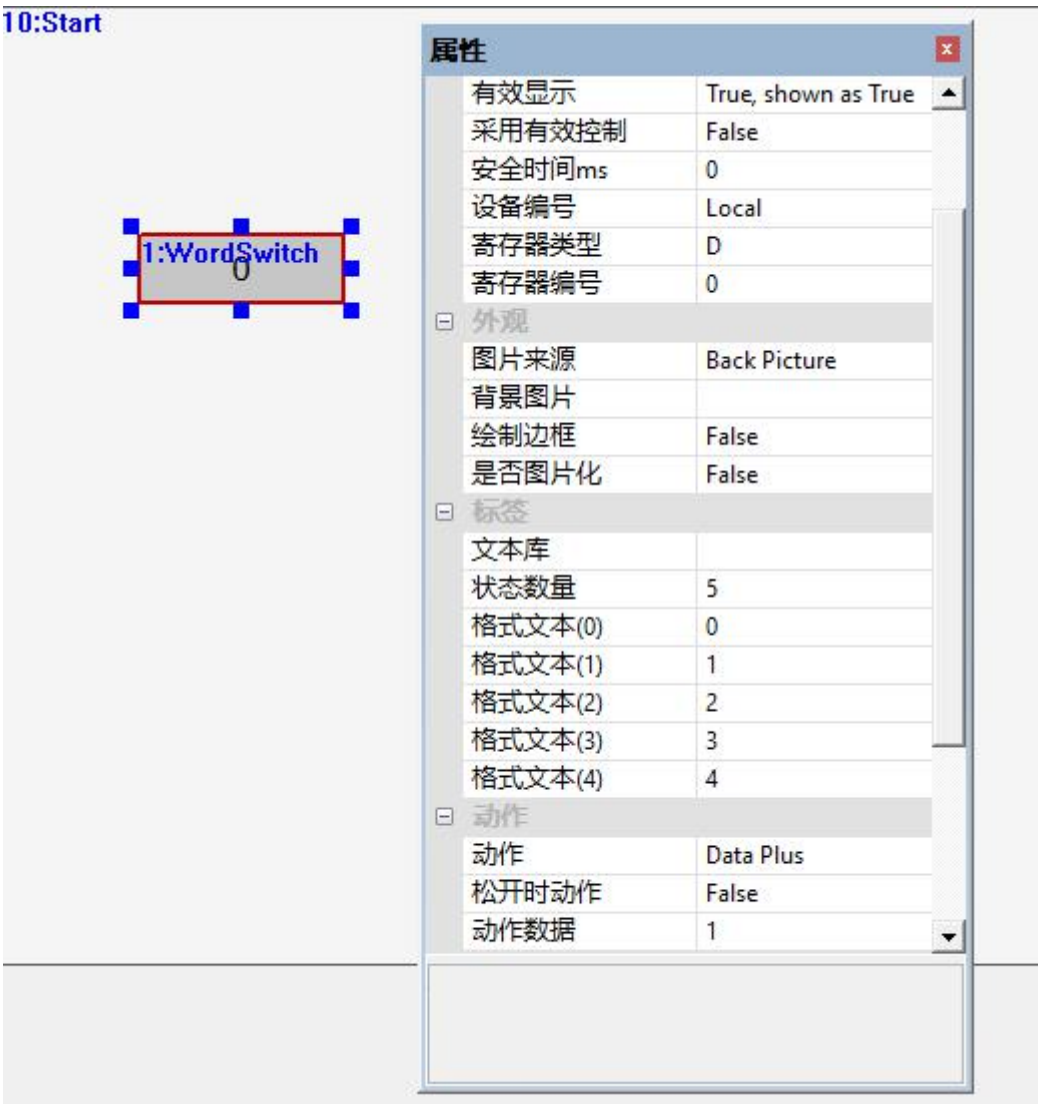
实现效果：

按下元件后，MODBUS_REG(0)=1，同时显示格式文本 1。

例三：寄存器在原来的值上加上动作数据的值

1．选择寄存器类型和编号

2．选择动作“Data Plus”和“动作数据”



实现效果：

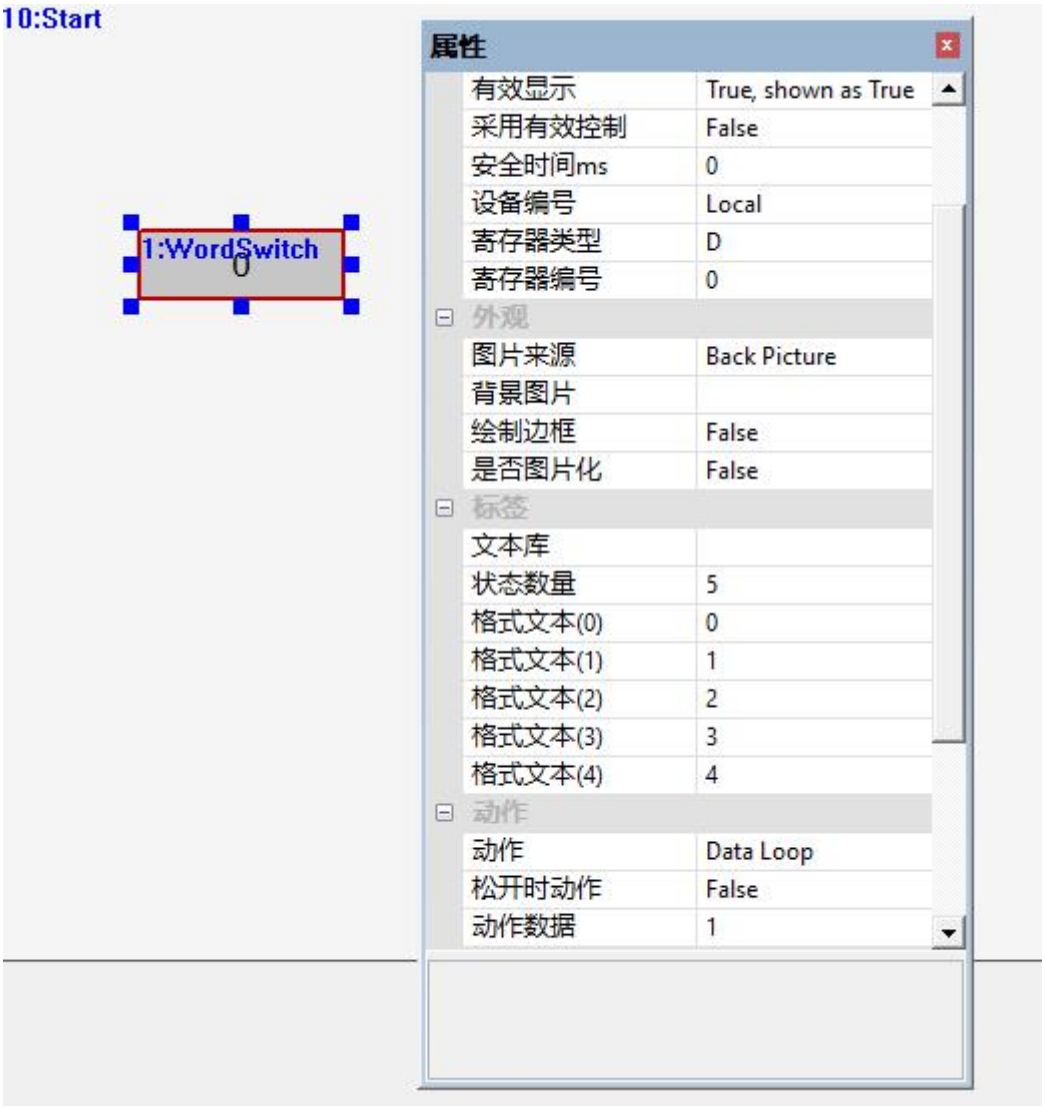
每按下元件一次后，MODBUS_REG(0)等于原来的值加 1。

当寄存器的值超过状态数量设置时，元件不显示，但触摸仍有作用效果，如上图，寄存器的值大于 4 时，元件不显示。

例四：寄存器原来值加上数据，在设置的状态之间循环切换

1．选择寄存器类型和编号

2．选择动作 Data Loop 和动作数据



实现效果：

按下元件后，MODBUS_REG(0)的值在 0 到 4 之间切换

如果 MODBUS_REG(0)的初始值大于 4 时，每按一下递减 4，直到寄存器的值为 0 后，开始在 0 到 4 之间切换。

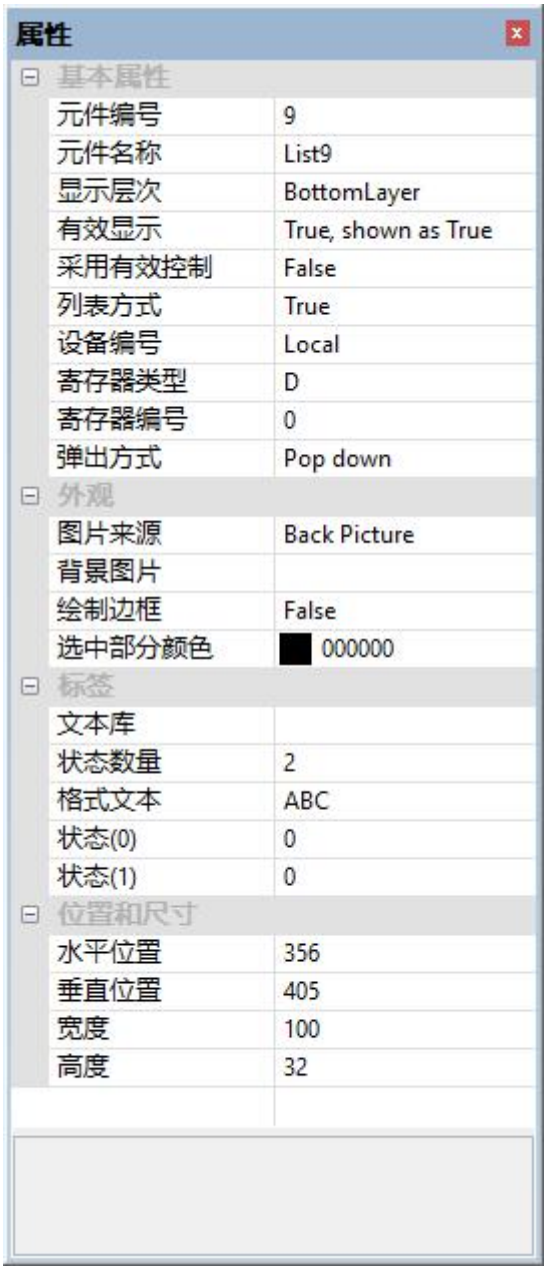
3.10.4. 列表

元件通过列表的方式显示多个状态，列表行数通过“状态数量”设置，范围 0-255。

选择某行文本时，能设置寄存器的值为该文本对应的状态的值。

列表显示不全时，通过修改元件大小解决。

1. 属性窗口：



2. 属性说明：

属性	功能	说明
元件编号	/	/
元件名称	/	/
显示层次	选择元件显示层次	TopLayer：表层，显示在最外层，覆盖底下元件 MidLayer：中间层

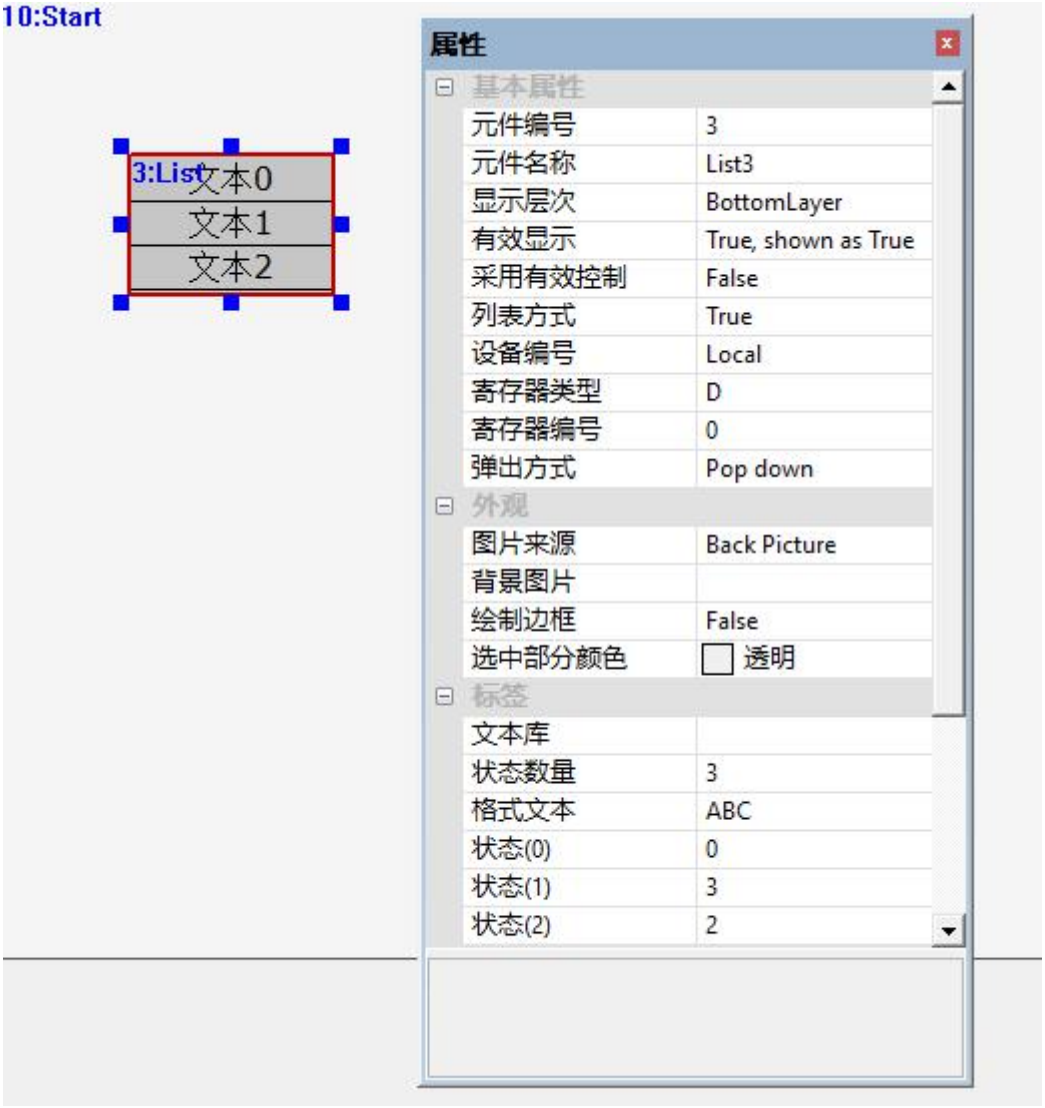
		BottomLayer：底层(默认)
有效显示	选择元件是否显示	默认 True，选择 False 时，元件不显示且无功能作用
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 Ture 通过寄存器控制元件是否显示
列表方式	设置列表元件采样列表或下拉式	默认 Ture
设备编号	设备编号	默认 local
寄存器类型	选择寄存器类型	多种寄存器下拉列表选择
寄存器编号	选择寄存器编号	寄存器值为 0 时不显示，非 0 时使用
弹出方式	可选向下弹出或向上弹出	默认向下弹出(Pop down)
图片来源	Back PictureLib 或 Back Picture	图片库或背景图片中选择
背景图片	背景图片选择	在图片来源先选择背景图片后添加
绘制边框	选择是否绘制边框	/
选中部分颜色	设置列表元件选中部分的颜色	/
文本库	文本库的名称	不设置文本库显示格式文本
状态数量	元件的状态数量(0-255)	可显示多个状态
格式文本	设置空间标签的样式	/
状态	设置列表的状态	由状态数量的值决定状态的个数
水平位置	元件的水平起始位置	不要超出水平分辨率
垂直位置	元件的垂直起始位置	不要超出垂直分辨率
宽度	元件的宽度	/
高度	元件的高度	/

使用例程：

1．选择寄存器类型和编号；

2．选择“状态数量”，打开状态文本窗口，设置状态值和状态显示的文本。

选择的状态显示颜色通过“选中部分颜色设置”。



实现效果：

按下文本 0 行时，MODBUS_REG(0)=0（为第一个状态 0 的值），按下文本 1 行时，MODBUS_REG(0)=3（为第二个状态 1 的值），按下文本 2 行时，MODBUS_REG(0)=2（为第三个状态 2 的值），选择的行显示颜色为“透明”。

状态值若相同，相同状态之间无法切换。

3.11. 值显示

值显示元件常用功能：

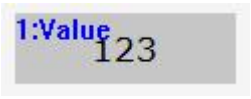
1．显示寄存器的内容，只能显示非字符类型数据

2．调用软键盘窗口直接设置绑定的寄存器的值。

3．调用 SUB 函数

可以选择要显示寄存器类型，设置显示数据类型，设置要显示数据的字符数和小数位数，设置要显示数据的上下限。

可编辑设置为 True，选择调用软键盘窗口后，便不能调用函数。



1. 属性窗口：

2. 属性说明：

属性	功能	说明
元件编号	/	/
元件名称	/	/
显示层次	选择元件显示层次	TopLayer：表层，显示在最外层，覆盖底下元件 MidLayer：中间层

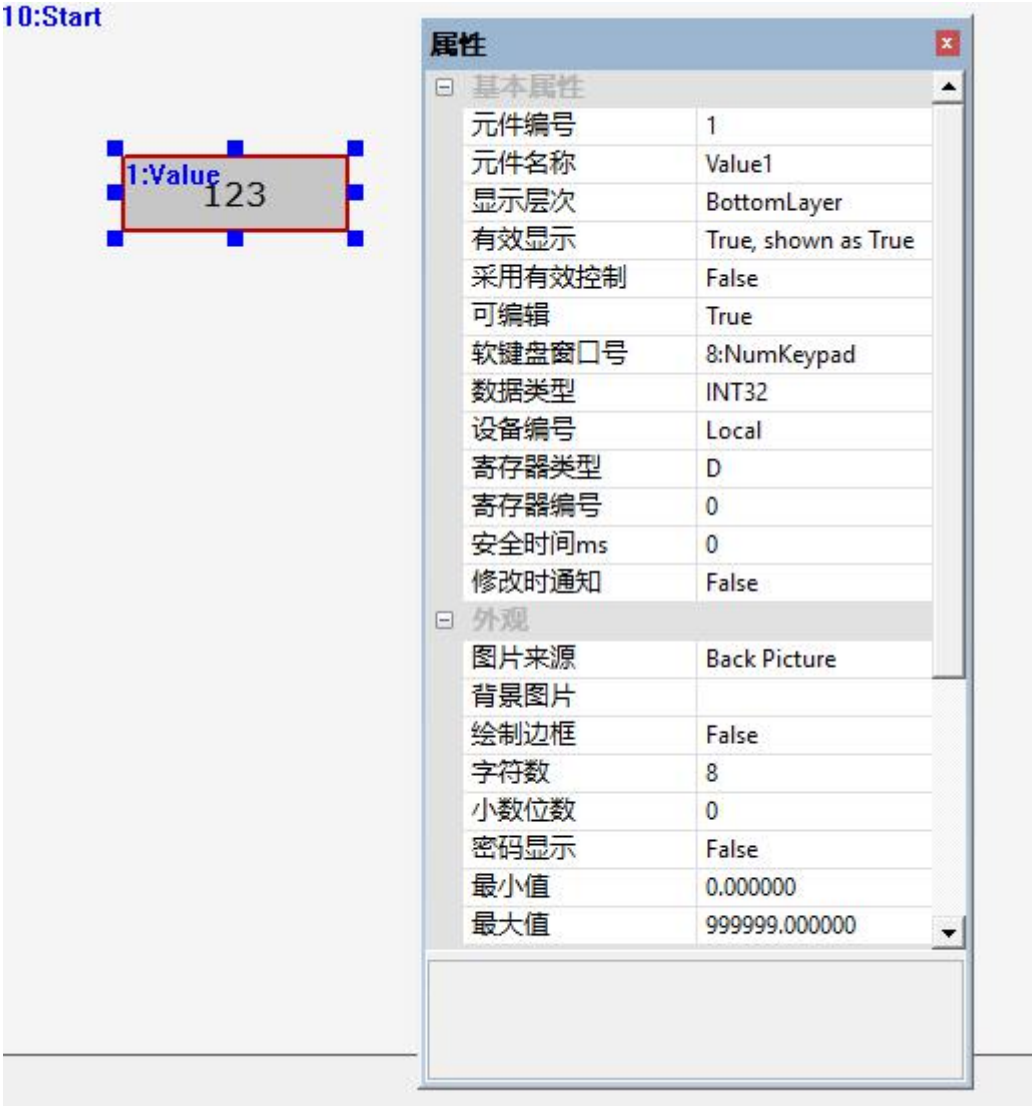
		BottomLayer：底层(默认)
有效显示	选择元件是否显示	默认 True，选择 False 时，元件不显示且无功能作用
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 Ture 通过寄存器控制元件是否显示
可编辑	数据元件是否启用输入	默认 False，选择 Ture 调用软键盘窗口输入
数据类型	数值元件的数据类型设置	默认 INT32
设备编号	设备编号	默认 local
寄存器类型	选择寄存器类型	多种寄存器下拉列表选择
寄存器编号	选择寄存器编号	寄存器值为 0 时不显示，非 0 时使用
安全时间 ms	最少按键时间	单位 ms
修改时通知	修改后发出 bit 位通知(设 ON 或 OFF)	默认 False，选择 Ture 时选择通知的寄存器
图片来源	Back PictureLib 或 Back Picture	图片库或背景图片中选择
背景图片	背景图片选择	在图片来源先选择背景图片后添加
绘制边框	选择是否绘制边框	/
字符数	设置字符元件操作的字符的长度	默认 8
小数位数	小数点位数	默认 0
密码显示	设置元件显示内容为*号	/
最小值	输入下限	/
最大值	输入上限	/
格式文本	设置空间标签的样式	/
点击调用函数	调用 Basic 函数	/
水平位置	元件的水平起始位置	不要超出水平分辨率
垂直位置	元件的垂直起始位置	不要超出垂直分辨率
宽度	元件的宽度	/
高度	元件的高度	/

示例：显示/更改寄存器的值

1. “可编辑”选择要调用的软件盘窗口

2. 设置寄存器的类型和编号

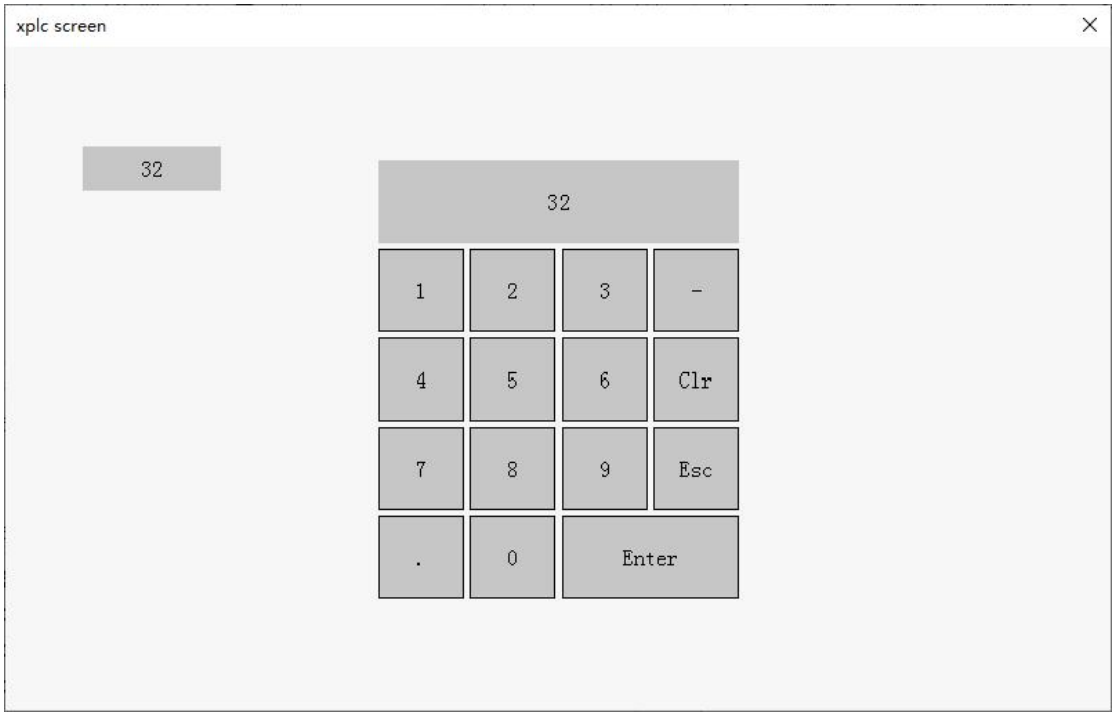
3. 根据需求设置数据类型，最大值，最小值等



实现效果：

元件初始的显示内容为绑定的寄存器的值，MODBUS_REG(0)=32，实时获取寄存器的值并刷新元件内容。

点击元件后打开软键盘窗口，此时可输入数据，确认后更改寄存器的值。

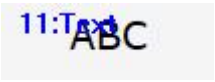


3.12. 字符显示

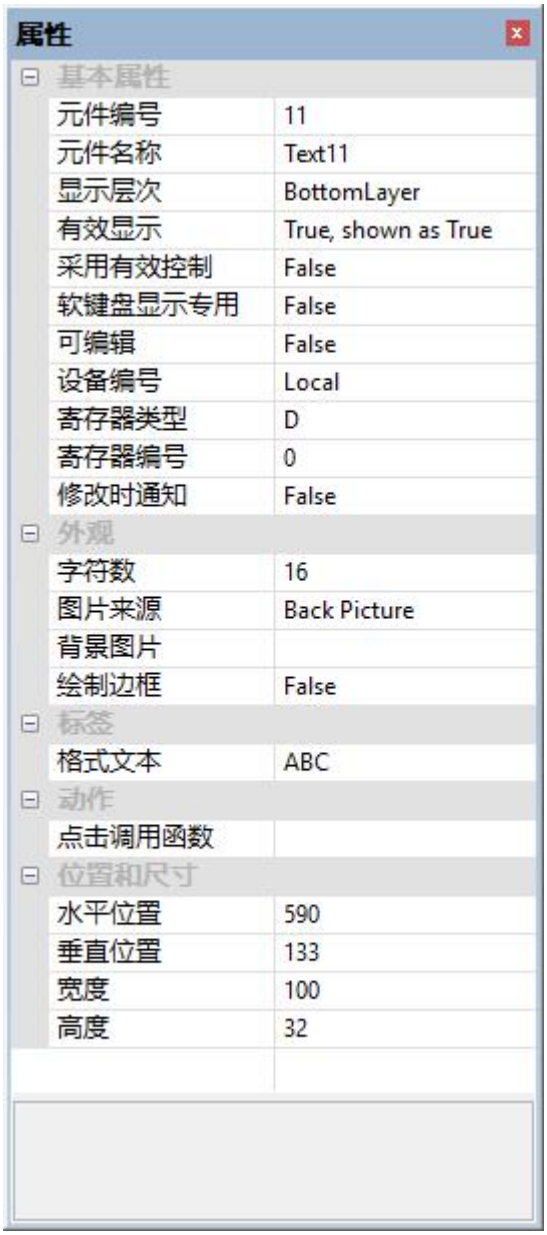
字符显示元件用来显示所有字符串类型数据，寄存器类型中的数据一定要是字符串类型。

调用软键盘窗口，显示自定义输入的数据。

还支持调用 SUB 函数。



1. 属性窗口：



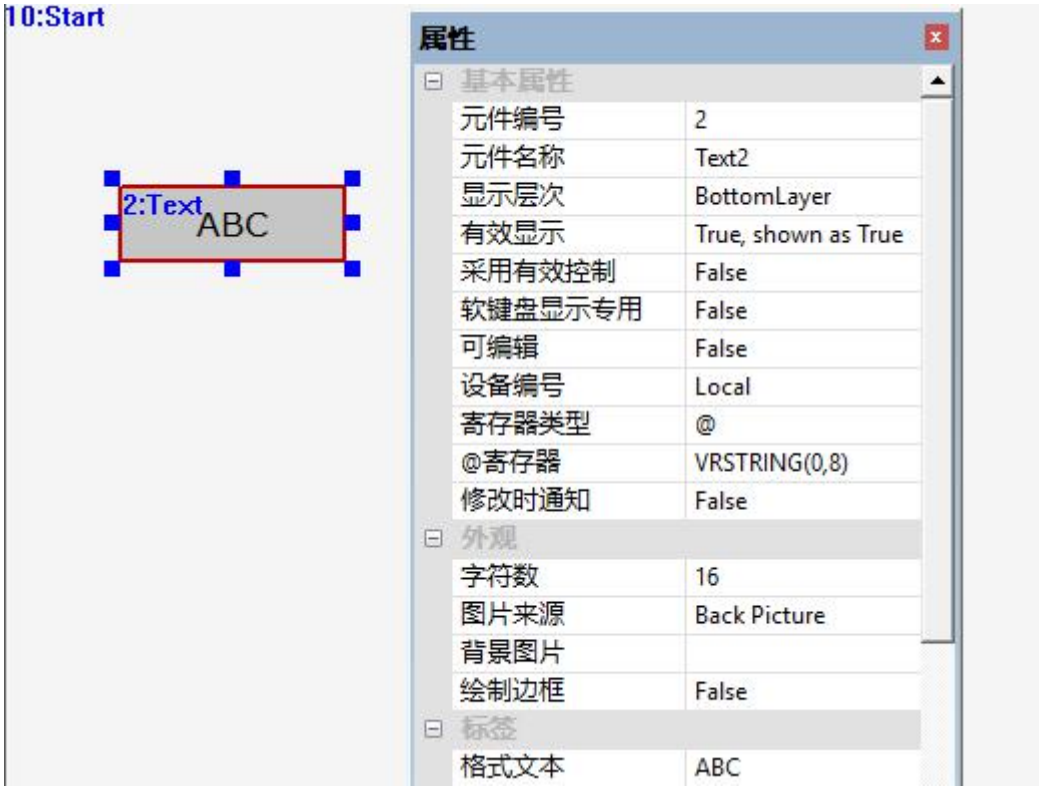
2. 属性说明：

属性	功能	说明
元件编号	/	/
元件名称	/	/
显示层次	选择元件显示层次	TopLayer：表层，显示在最外层，覆盖底下元件 MidLayer：中间层 BottomLayer：底层(默认)
有效显示	选择元件是否显示	默认 True，选择 False 时，元件不显示且无功能作用
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 Ture 通过寄存器控制元件是否显示

软键盘显示专用	软键盘的输入显示	一般用于键盘窗口
可编辑	数据元件是否启用输入	默认 False，选择 Ture 调用软键盘窗口输入
设备编号	设备编号	默认 local
寄存器类型	选择寄存器类型	多种寄存器下拉列表选择
寄存器编号	选择寄存器编号	寄存器值为 0 时不显示，非 0 时使用
修改时通知	修改后发出 bit 位通知(设 ON 或 OFF)	默认 False，选择 Ture 时选择通知的寄存器
字符数	设置字符元件操作字符的长度	默认 16
图片来源	Back PictureLib 或 Back Picture	图片库或背景图片中选择
背景图片	背景图片选择	在图片来源先选择背景图片后添加
绘制边框	选择是否绘制边框	/
格式文本	设置空间标签的样式	/
点击调用函数	按键按下时调用函数	下拉框选择可以调用的函数名
水平位置	元件的水平起始位置	不要超出水平分辨率
垂直位置	元件的垂直起始位置	不要超出垂直分辨率
宽度	元件的宽度	/
高度	元件的高度	/

例一：显示寄存器存储的字符串

1．设置寄存器的类型和编号，显示字符串类型数据。



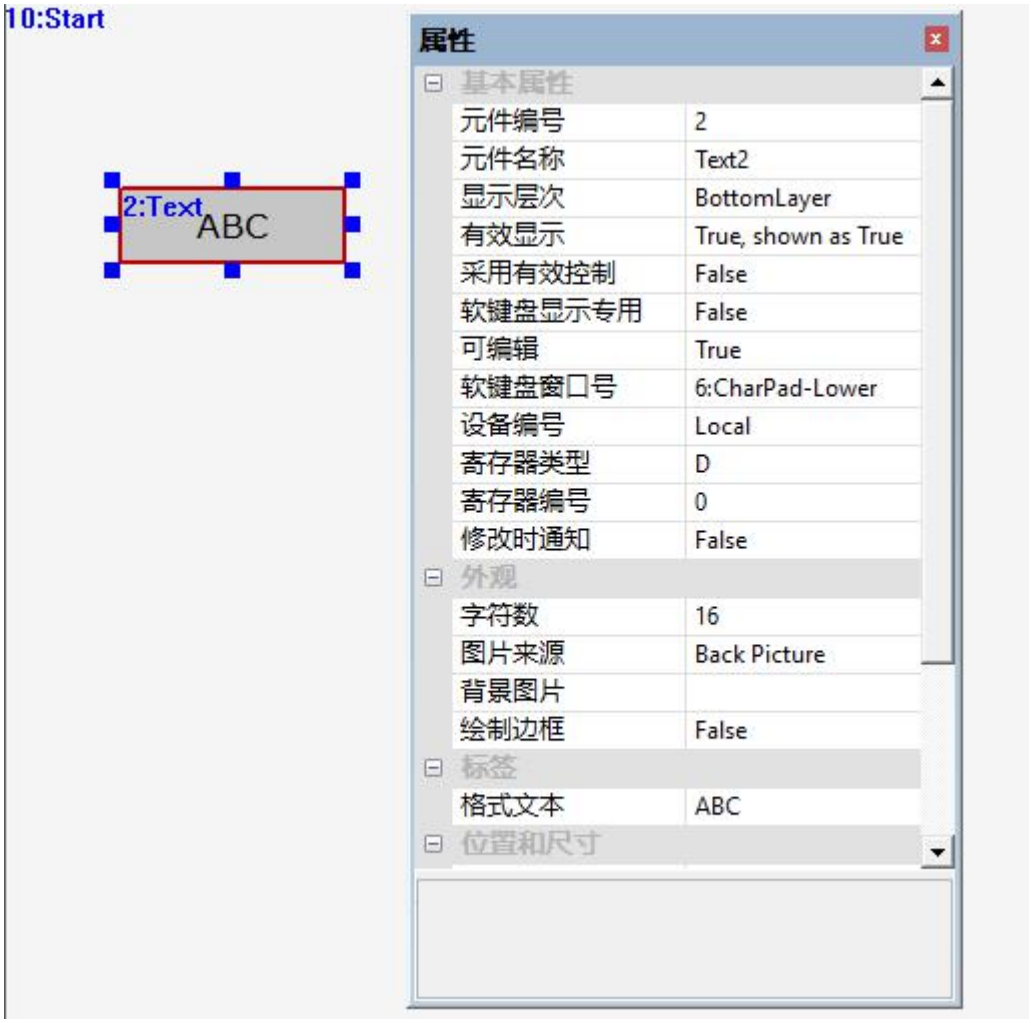
实现效果：

当 VRSTRING(0,8) = "abc" ，元件显示寄存器保存字符串 abc。

例二：自定义元件显示文本

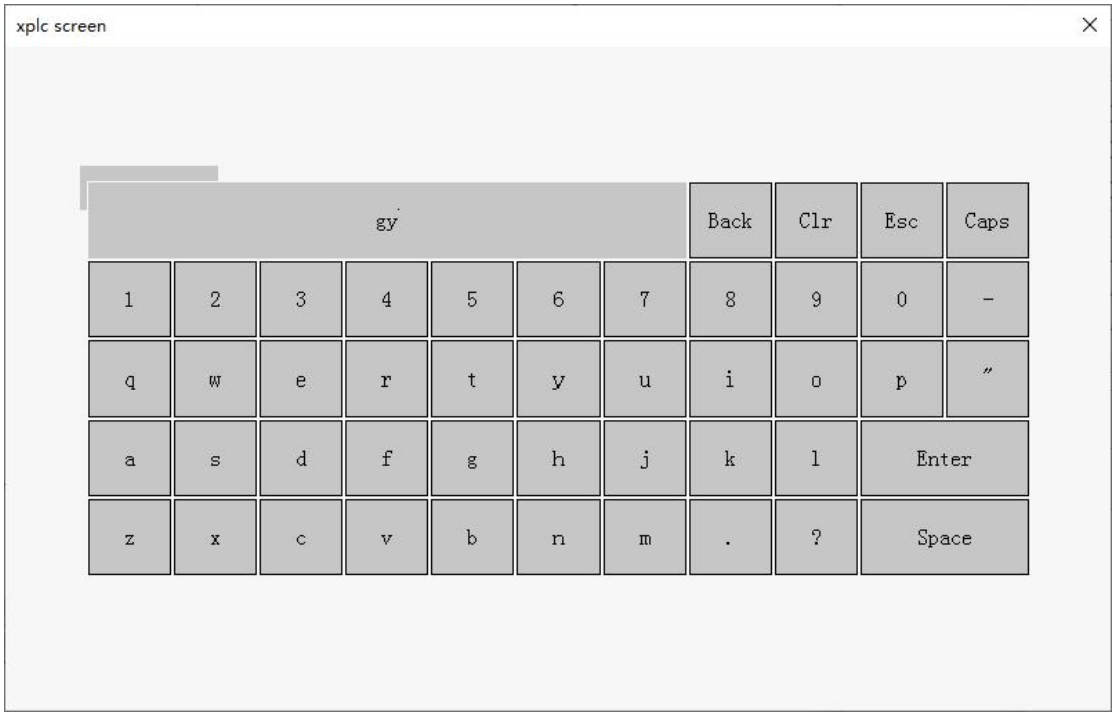
1. “可编辑”选择要调用的软件盘窗口；

2. 设置寄存器的类型和编号。



实现效果：

点击字符显示元件，弹出软键盘窗口，设置元件要显示的内容后 Enter 确定。



3.13. 滑块

“滑块”功能为预留，暂不支持。

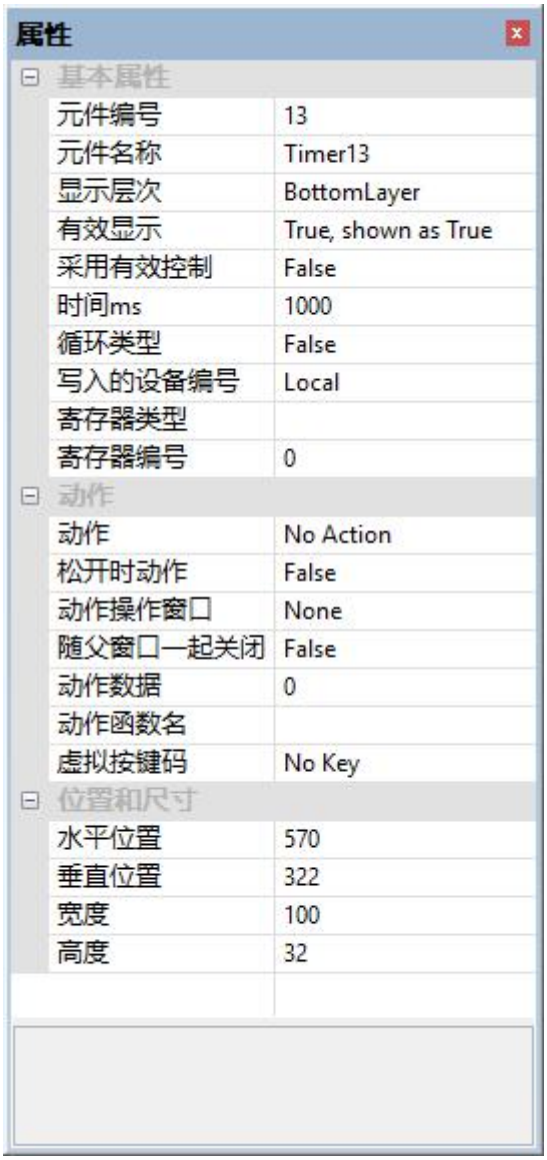
3.14. 定时器

Hmi 界面编辑时显示，实际运行中不显示。

设置好定时器启动间隔时间，重复进行动作。

可以调用函数和对寄存器进行赋值。

1. 属性窗口：



2. 属性说明：

属性	功能	说明
元件编号	/	/
元件名称	/	/
显示层次	选择元件显示层次	TopLayer：表层，显示在最外层，覆盖底下元件 MidLayer：中间层 BottomLayer：底层(默认)
有效显示	选择元件是否显示	默认 True，选择 False 时，元件不显示且无功能作用
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 Ture 通过寄存器控制元件是否显示
安全时间 ms	最少按键时间	单位 ms
循环类型	选择定时器是否循环	默认 False

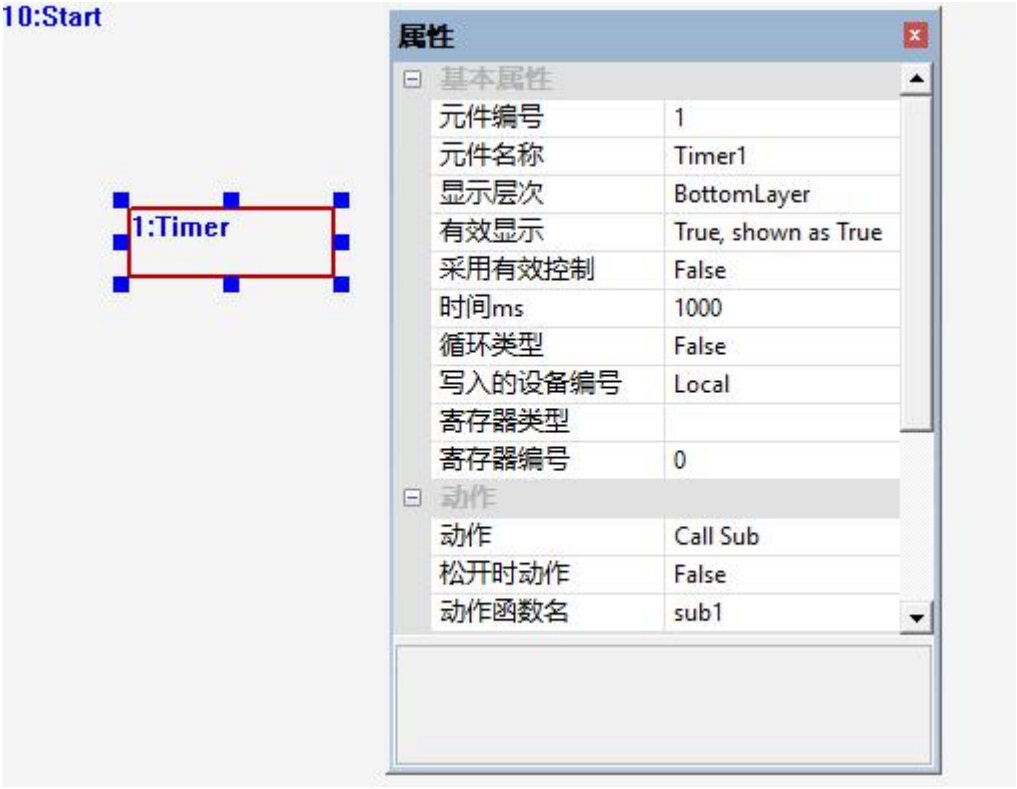
设备编号	设备编号	默认 local
寄存器类型	选择寄存器类型	多种寄存器下拉列表选择
寄存器编号	选择寄存器编号	寄存器值为 0 时不显示，非 0 时使用
图片来源	Back PictureLib 或 Back Picture	图片库或背景图片中选择
背景图片	背景图片选择	在图片来源先选择背景图片后添加
绘制边框	选择是否绘制边框	/
是否图片化	元件变为图片的形式	默认 False
文本库	文本库的名称	不设置文本库显示格式文本
格式文本 0/1	打开格式文本设置窗口设置元件要显示的文本	默认显示文本 0，按下时显示文本 1
动作	按键执行时的动作	参见“动作”章节描述
松开时动作	选择按下时或松开时执行动作	默认 False 为按下执行动作，Ture 为松开时动作
动作操作窗口	选择需要操作的窗口编号	下拉列表选择已有窗口
随父窗口一起关闭	子窗口随父窗口一起关闭	默认 False
动作数据	按键动作后给寄存器写入指定值	/
动作函数名	按键动作后要调用的 SUB 函数	下拉列表选择 Basic 已有全局 SUB 函数
虚拟按键码	选择虚拟按键码	默认不选择
水平位置	元件的水平起始位置	不要超出水平分辨率
垂直位置	元件的垂直起始位置	不要超出垂直分辨率
宽度	元件的宽度	/
高度	元件的高度	/

例一：调用 SUB 函数

1．在 Basic 文件中，编写一个全局的 SUB 函数；

2．在元件属性，动作选择“Call Sub”，“动作函数名”选择对应的 SUB 函数名；

3．在“时间 ms”中填入调用函数的时间间隔。



实现效果：

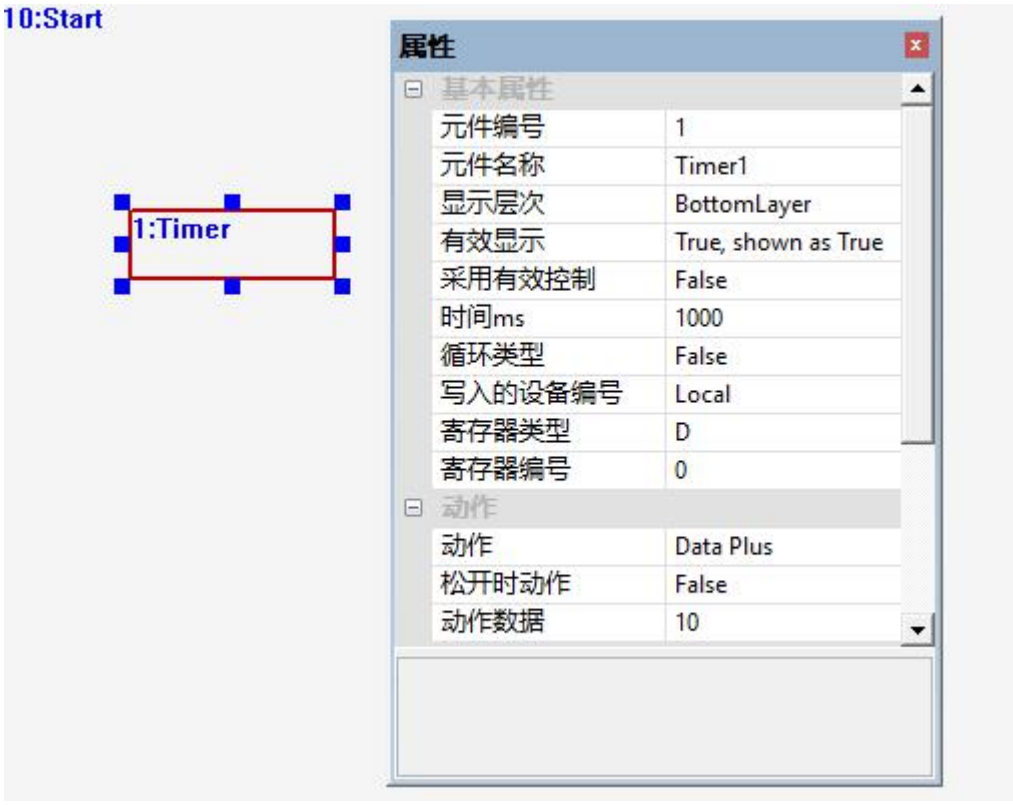
HMI 运行后，元件不显示，每隔 1000ms 调用一次函数 SUB 执行。

例三：寄存器在原来的值上加上动作数据的值

1．选择寄存器类型和编号；

2．选择动作“Data Plus”，“动作数据”填入每次寄存器要增加的数据。

3．在“时间 ms”中填入寄存器累加的时间间隔。



实现效果：

每间隔 1000ms，MODBUS_REG(0)的值等于原来的值加 10。

3.15. 自定义元件

自定义元件是通过调用 Basic 程序来定义其行为的元件，与程序配合在显示屏上动态绘图，通过元件大小确定绘图区域。

绘图函数的坐标都是相对于元件的左上方为零点。

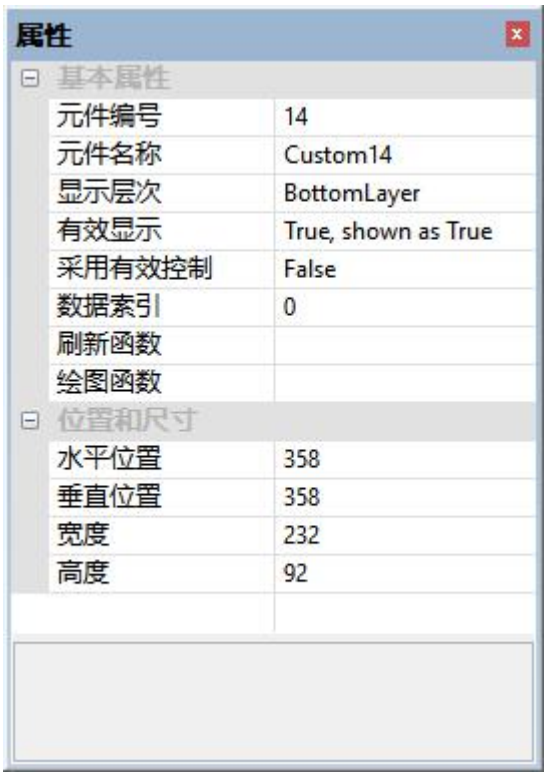
自定义元件的 3 个特别属性：

数据索引：一般用于 TABLE 编号，可以指明当前元件的数据位于的 TABLE 位置。

刷新函数，绘图函数：指明元件行为的 SUB 函数，参见第四章说明。

自定义元件的使用例程参见正运动官方网站 www.zmotion.com.cn 触摸屏例程。

1. 属性窗口：



2. 属性说明：

属性	功能	说明
元件编号	/	/
元件名称	/	/
显示层次	选择元件显示层次	TopLayer：表层，显示在最外层，覆盖底下元件 MidLayer：中间层 BottomLayer：底层(默认)
有效显示	选择元件是否显示	默认 True，选择 False 时，元件不显示且无功能作用
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 Ture 通过寄存器控制元件是否显示
数据索引	指明当前元件的数据位于的 TABLE 位置	/
刷新函数	指明元件行为的 SUB 函数	周期调用来判断是否要重新绘图
绘图函数	指明元件行为的 SUB 函数	需要绘图时自动被调用
水平位置	元件的水平起始位置	不要超出水平分辨率
垂直位置	元件的垂直起始位置	不要超出垂直分辨率
宽度	元件的宽度	/
高度	元件的高度	/

3.16. 事件列表

预留

3.17. PLC 控制

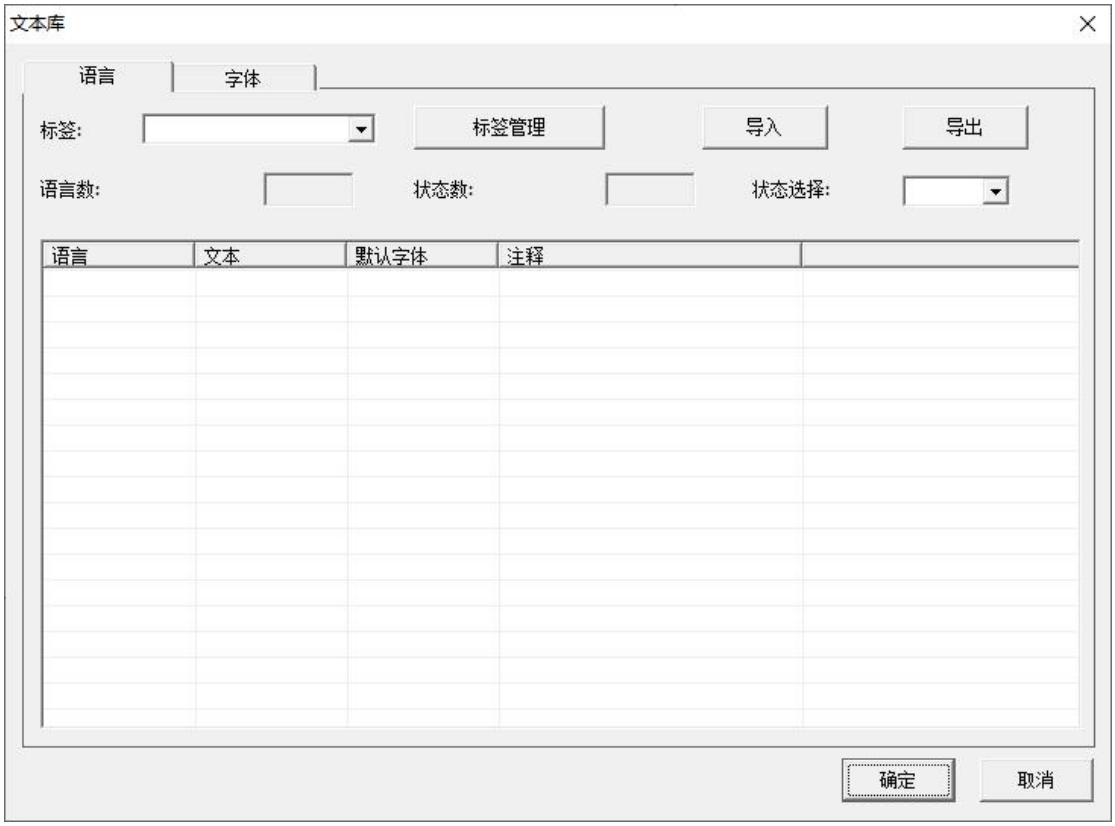
预留

3.18. 文本库

1. 文本库建立

点击菜单栏“元件”-“文本库”打开如下窗口，在此窗口的“标签管理”里创建文本库后才能调用文本库。

文本库的字体由正运动 Zfontmake 字库工具制作，生成的字库文件后缀.zft，或后缀.ttf 的文件，用然后在下方窗口的“字体”窗口里载入新字库或删除字库。

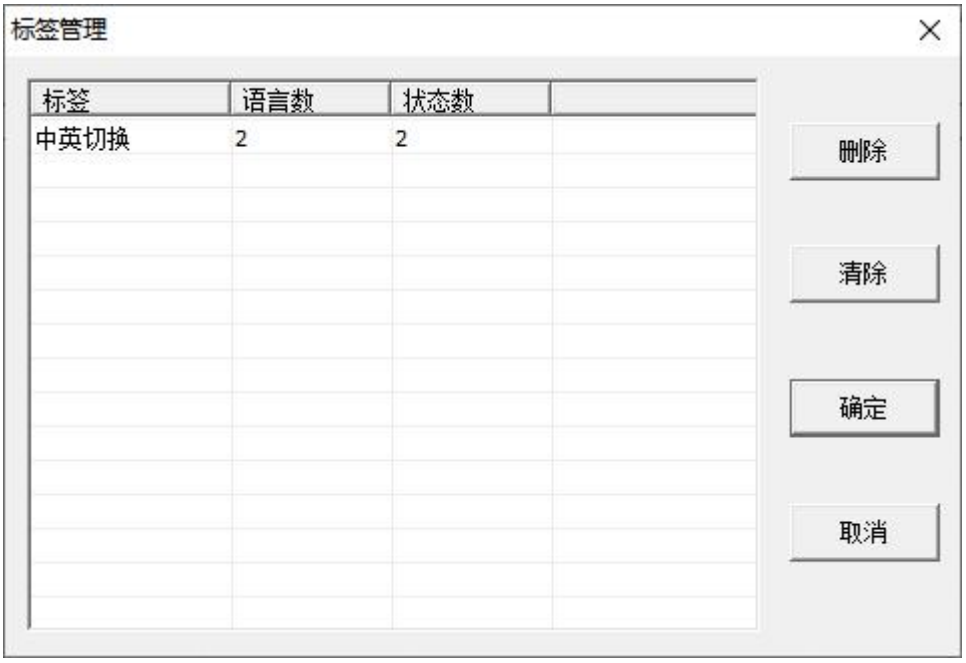


点击“标签管理”打开如下窗口，空白处双击输入内容，依次输入标签名称、标签语言数和状态数后的，点击“确认”新建标签。

语言数：指文本在一个状态下有多少个数据。

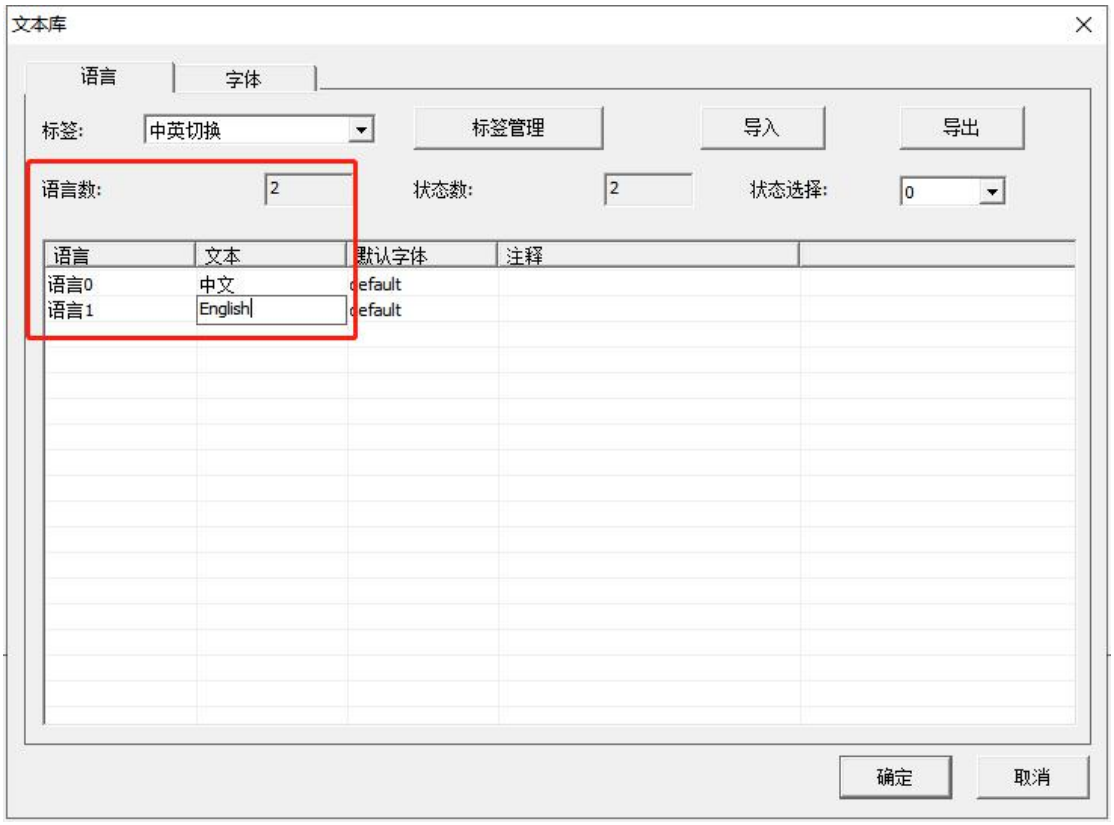
状态数：指文本共有多少个状态，通过状态选择下拉菜单选取。

文本库可以为每个文本显示添加 8 种语言模式。每种语言模式下又可设置 128 种状态。双击对应状态栏下方进行修改。

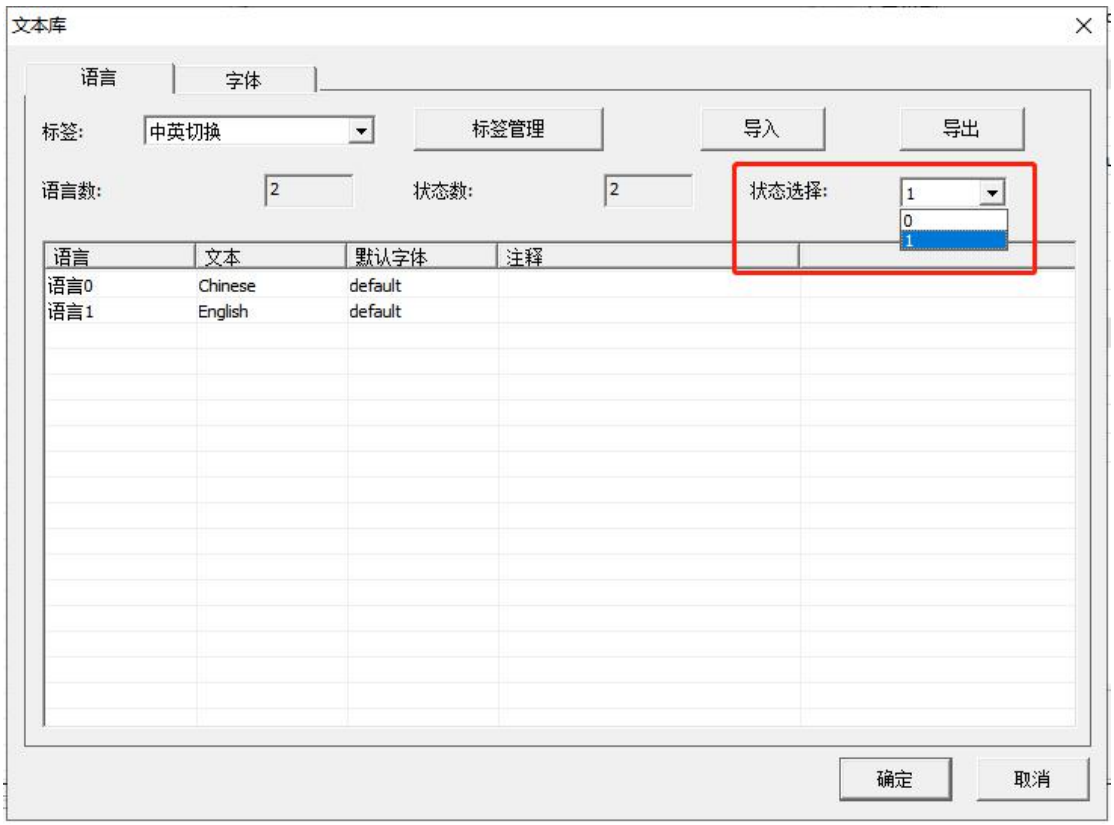


标签建立成功在下方窗口中显示标签的语言数，输入显示文本。

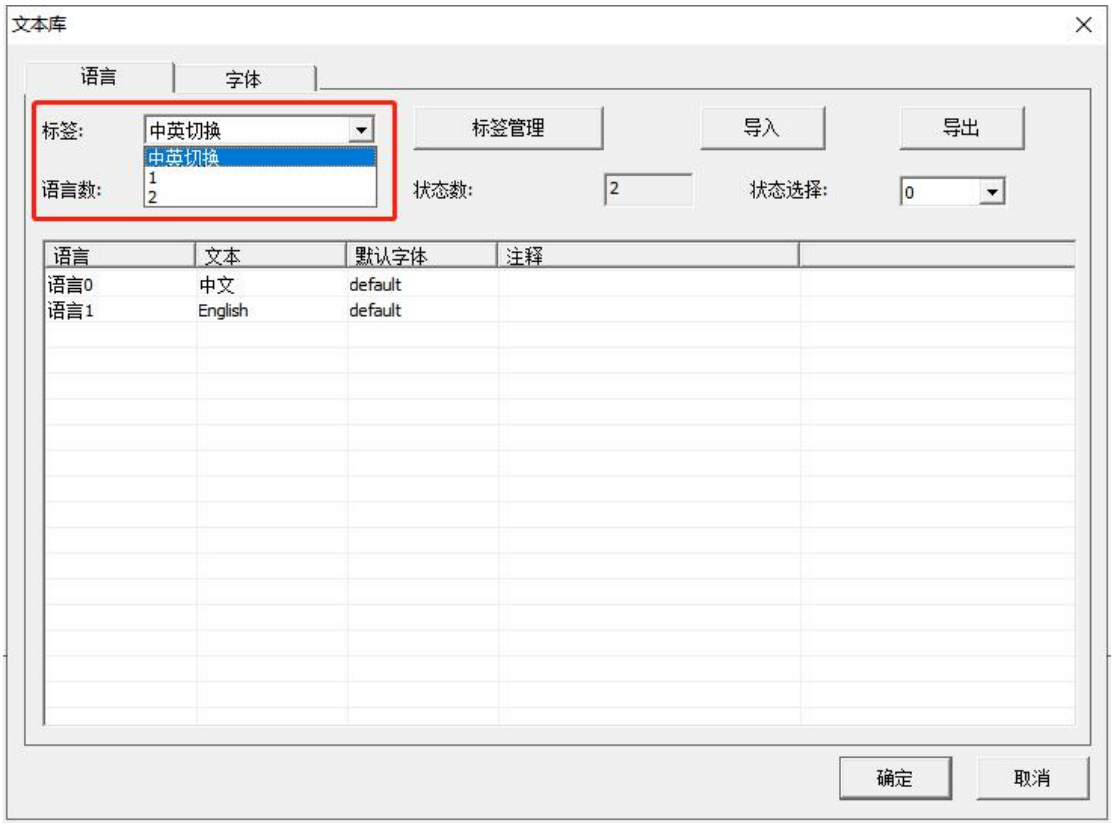
语言数 2：如下图语言 0 和语言 1



状态数为 2.，包含状态 0 和状态 1，在状态选择栏切换状态，上图为状态 0 的显示内容，下图为状态 1 的显示内容。



文本库可以新建多个，在“标签管理”里新建，通过标签栏切换文本库。



2．导入导出

预留，暂不支持。

3．文本库使用示例

可以通过 HMI_LANG=ilang（语言标号）函数，设置各种语言的切换。

可以通过改变寄存器的数值改变语言的状态。例如：寄存器值为 0 显示的是语言 0 的状态 0，当寄存器的值变为为 1 时，显示的是语言 0 的状态 1。

下方例程调用 HMI_LANG 函数改变语言状态，寄存器改变语言的状态的例程参见“图片库”。

(1) HMI 组态程序

(2) Basic 程序

```
GLOBAL SUB langue_chinese()
    HMI_LANG=0      '选择文本库语言编号 0，中文
END SUB
GLOBAL SUB langue_english()
    HMI_LANG=1      '选择文本库语言编号 1，English
END SUB
```

(3) 调用过程

HMI 的文本元件 3 调用文本库“中英切换”语言显示，如下图。

功能键 7 调用 Basic 函数 langue_chinese，显示文本库的语言 0；功能键 8 调用 Basic 函数 langue_english，显示文本库的语言 1。

(4) 运行效果

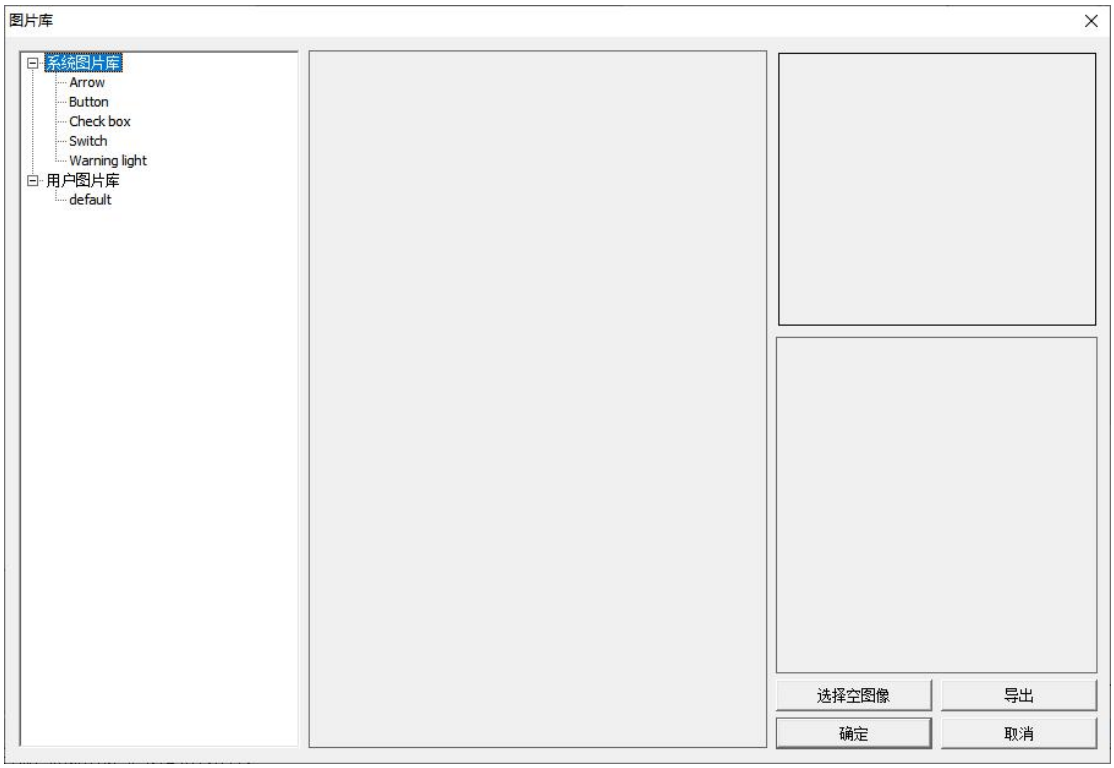
按下“中文”功能键的文本元件显示效果：

按下“英文”功能键的文本元件显示效果：

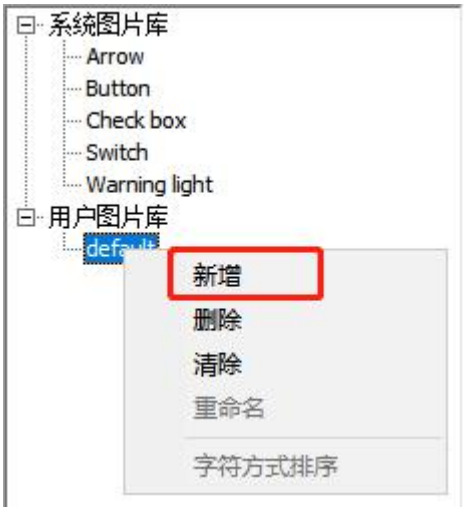
3.19. 图片库

1. 图片库建立

点击菜单栏“元件”-“图片库”打开如下窗口，在此窗口包含“系统图片库”系统自带的图片，和“用户图片库”，由用户自定义上传。



选择“用户图片库”下方的 default 点击右键，选择“新增” 新建图片库。



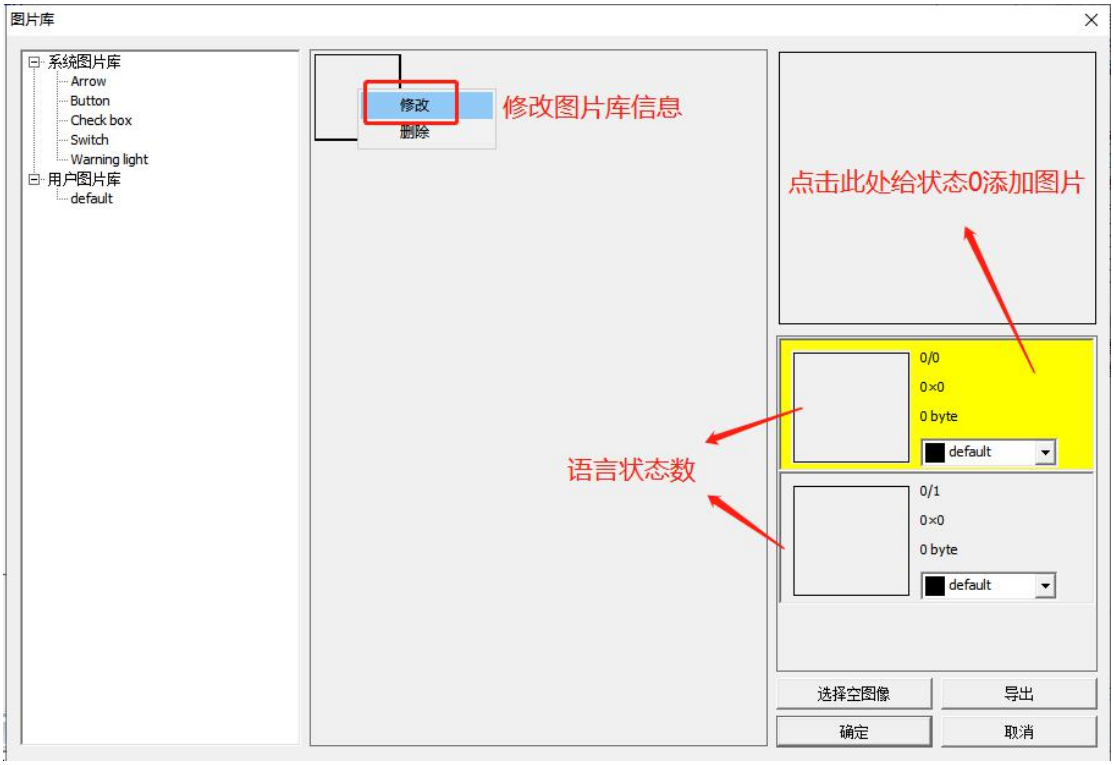
填入新增图片库的名称、语言数和状态数后确定。

每个图片库可设置 8 种语言模式。

每种语言模式下可设置 128 种状态。

新建图片库成功后也可点击右键修改上方的图片库信息，这里语言个数为 1，名称为语言 0，语言 0 的状态数设置为 2，0/0 表示语言 0 的状态 0，0/1 表示语言 0 的状态 1。

如右下角，黄底表示选择该状态，然后在右上角的空白处双击，从电脑里给语言 0 的状态 0 选择图片添加。



支持新建多个图片库。“导出”用于把当前选中的图片（黄色底的图）导出到电脑上。

图片库的新建或更改后需要点击右下角的“确定”按钮保存设置，直接关闭图片库窗口或点击“取消”按钮均无法保存更改。

2. 图片库使用示例

可以通过 HMI_LANG=ilang（语言标号）函数，设置各种语言的切换。

可以通过改变寄存器的数值改变语言的状态。例如：寄存器值为 0 显示的是语言 0 的状态 0，当寄存器的值变为为 1 时，显示的是语言 0 的状态 1。

下方例程采用寄存器改变语言状态，HMI_LANG 函数改变语言状态参见文本库例程。

(1) HMI 程序：

(2) 调用过程：

位状态元件 4、5 由位状态切换开关 1 控制，位状态切换开关 1 按下寄存器 M0 的值取反（M0 的值在 0 和 1 之间来回切换），位状态元件 4、5 均采用寄存器 M0 控制，从而切

换语言的状态，显示不同状态下的图片。

位状态元件 4 调用图片库“deng”。

(3) 运行效果

位状态切换开关 1 中 M0 值为 0 的效果：

位状态切换开关 1 中 M0 值为 1 的效果：

3.20. 按键转换

3.20.1. 物理键

物理键是指外部设备上的实际按键，每个按键都有独有的编码值，按下时会发送一条信息，这条信息就是按键的编码值。

物理键的编码值有硬件决定，程序中无法修改。外设不同，对应按键的编码值也不同。

ZHD400X 标准物理按键编码：

```
Global Const key_f1 = 11 '功能键 1
Global Const key_f2 = 12 '功能键 2
Global Const key_f3 = 13 '功能键 3
Global Const key_f4 = 14 '功能键 4
Global Const key_f5 = 15 '功能键 5
Global Const key_f6 = 16 '功能键 6
Global Const key_X-- = 24 '轴移动按键
Global Const key_X+ = 25
Global Const key_Y-- = 34
Global Const key_Y+ = 35
```

Global Const key_Z-- = 44

Global Const key_Z+ = 45

Global Const key_U-- =54

Global Const key_U+ = 55

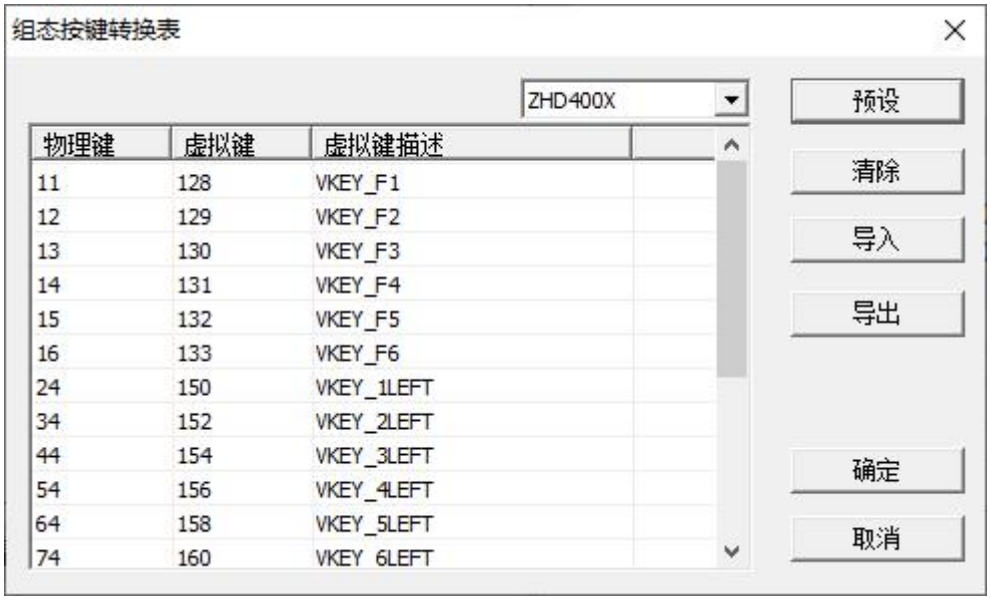
Global Const key_A-- =64

Global Const key_A+ = 65

Global Const key_B-- =74

Global Const key_B+ = 75

或直接查看 ZDevelop 软件里的组态按键转换表：



ZHD300X 物理按键的编码按行列组合而成，键值=行号(1-10)×10+列号(1-5)。

ZHD300X 标准物理按键编码：

Global Const key_f1 = 11 '功能键 1

Global Const key_f2 = 12 '功能键 2

Global Const key_f3 = 13 '功能键 3

Global Const key_f4 = 14 '功能键 4

Global Const key_f5 = 15 '功能键 5

```
Global Const key_1 = 51 '数字键 1，同时字母按键切换.
Global Const key_2 = 52
Global Const key_3 = 53
Global Const key_4 = 61
Global Const key_5 = 62
Global Const key_6 = 63
Global Const key_7 = 71
Global Const key_8 = 72
Global Const key_9 = 73
Global Const key_0 = 81 '数字键 0
Global Const key_Add= 83 '加号
Global Const key_Point=82 '小数点
Global Const key_xUp=25 'JOG 第一轴
Global Const key_yUp=35 '第 2 轴
Global Const key_zUp=45 '第 3 轴
Global Const key_rUp=55 '第 4 轴
Global Const key_xDown =24 'JOG 第一轴
Global Const key_yDown =34
Global Const key_zDown =44
Global Const key_rDown =54
Global Const key_Jog5L=64
Global Const key_Jog5R=65
Global Const key_Jog6L=74
Global Const key_Jog7R=75
Global Const key_Left=21 '左移
Global Const key_Up=22
```

Global Const key_Right=23

Global Const key_Down=32

Global Const key_SpeedUp=41

Global Const key_SpeedDown=43

Global Const key_Step=84

Global Const key_Manual=85

Global Const key_Reset =91 '复位

Global Const key_Del =92 '删除

Global Const key_Inset =93 '插入

Global Const key_Switch=94 'SHIFT 切换

Global Const key_Save =95 '保存

Global Const key_Esc =101 '取消

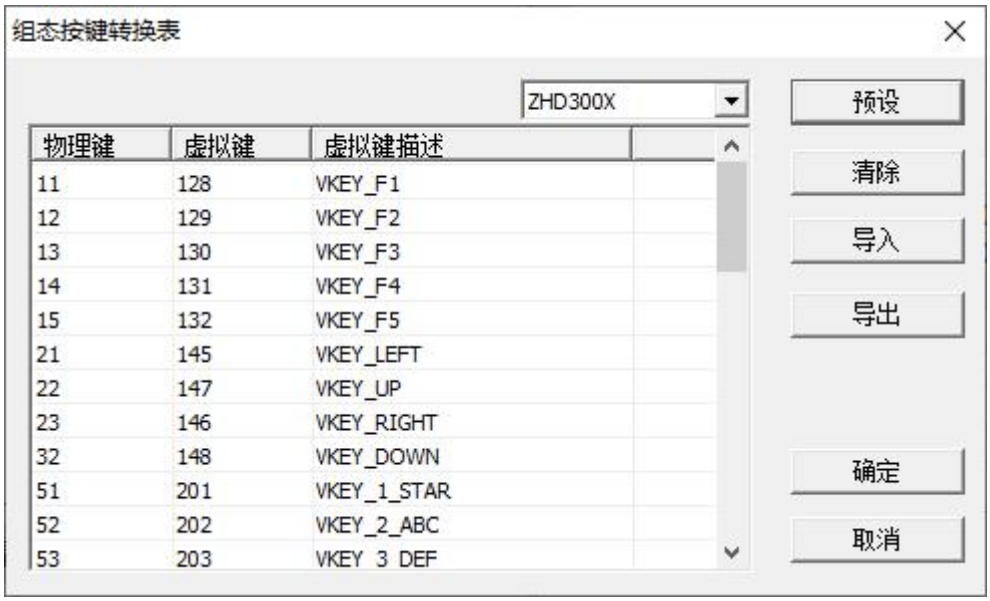
Global Const key_Edit =102 '编辑监控

Global Const key_File =103 '文件管理

Global Const key_Set =104 '参数设置

Global Const key_Ent =105 '输入确定

或直接查看 ZDevelop 软件里的组态按键转换表：



3.20.2. 虚拟键

在实际编程中，如果使用物理键编码编写程序，那么程序的可移植性很低，所以程序编写时希望有一个编码可以用在所有外设上，所以虚拟编码就出现了，只要将外设的物理键编码与虚拟编码一一对应，程序就可以用在不同的外设上。

由于虚拟编码的操作方式和物理键编码相似，所以就叫做虚拟键。

Hmi 中，虚拟键编码由底层封装而成，程序中无法修改。

虚拟键编码值 0-127 都对应 ASCII 码表，128 往后则自定义了功能。

3.20.3. 按键转换表的编辑

选择“元件”-“按键转换”打开下方页面，组态按键转换表主要由列表区、功能区和选择菜单 3 部分组成。

1．选择菜单

在下拉菜当中选择已经编辑好的转换表，目前有 ZHD300X 和 ZHD400X。

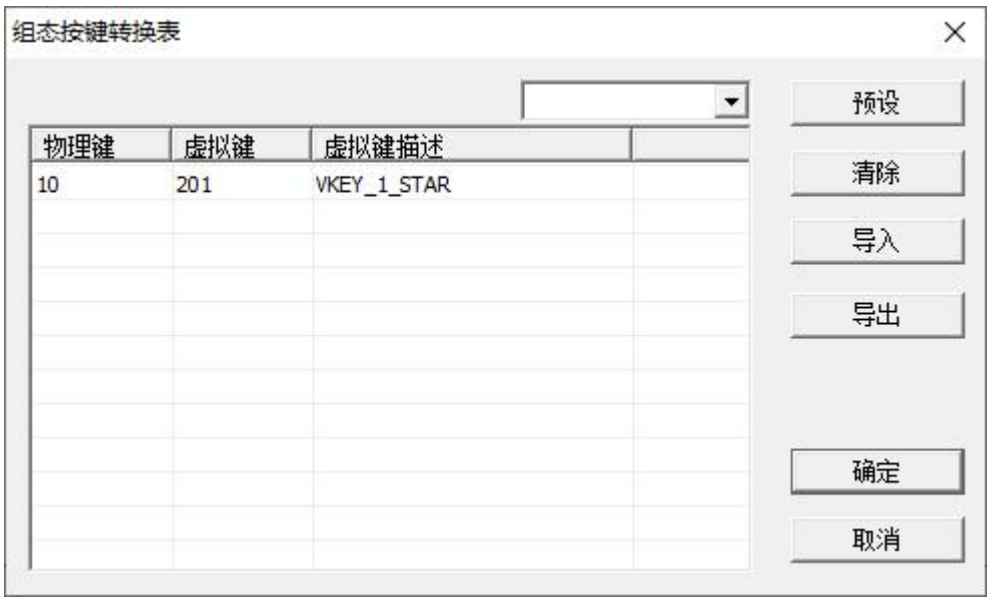
2．列表区

物理键：设置值为外部设备按键的编码值。

虚拟键：设置值为希望与外部物理键绑定的虚拟编码值，0-127 都对应 ASCII 码表，128 往后则自定义了功能，参考 3.10.5。

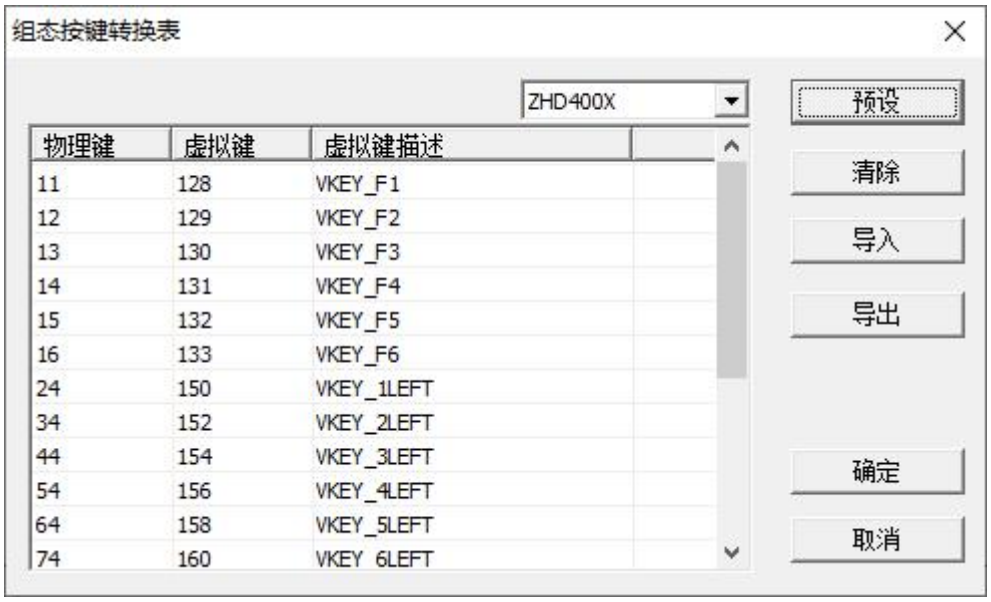
虚拟键描述：对当前虚拟键功能的说明。

示例：将物理键 10 与虚拟键 201 绑定，参考下图。



3．功能区

预设：结合下拉菜单，调出已经编辑好的转换表，目前只有 ZHD300X。



清除：删除上图所有的转换设置，清空列表。

导入：从外部文件调用编辑好的转换表，选择由本页面导出的保存转换表信息的文件，文件格式 ini。

导出：把当前编辑的转换表导出为 ini 格式文件，用来保存当前转换表。

确定：编辑好后，要点击确定才可以应用转换表，否侧下次打开按键转换表时为空。

取消：取消对转换表的操作并退出。

3.20.4. 按键转换指令

与按键转换相关的 basic 指令主要为以下 6 条。

KEY_STATE：物理按键状态

KEY_EVENT：物理按键状态扫描

KEY_SCAN：读取物理按键编码

VKEY_STATE：虚拟按键状态

VKEY_EVENT：虚拟按键状态扫描

VKEY_SCAN：读取虚拟按键编码

程序中可以使用 VKEY_SCAN 来捕捉是哪个虚拟键按下，根据按键转换表就可以知道对应的是哪个物理键；也可以直接使用 KEY_SCAN 捕捉是哪个物理键按下。

一般情况下不建议使用 KEY_SCAN 及物理键相关指令，因为不同外设的物理键编码都不同，这么用程序的可移植性较低，建议使用 VKEY_SCAN 及其他虚拟键相关指令。

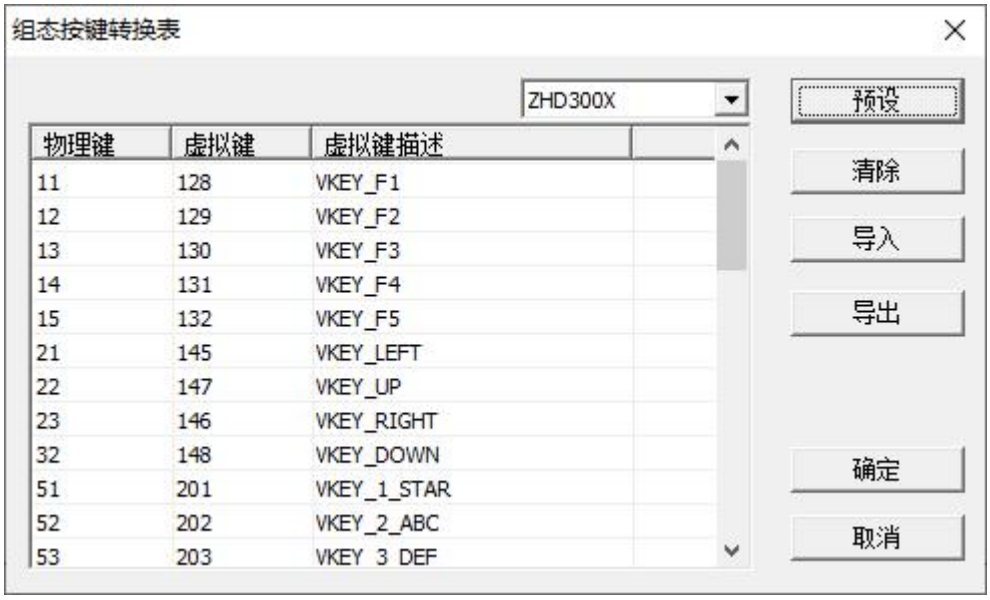
由于这些指令只能在自定义元件的刷新函数中使用（HMI 的初始化函数也行，但是不建议这么做），所以至少要有一个自定义元件存在。

扫描到按键按下后，把返回值赋值到一个自定义变量，在自定义元件的绘图函数中根据返回值的不同，来调用不同的函数，实现不同的功能。

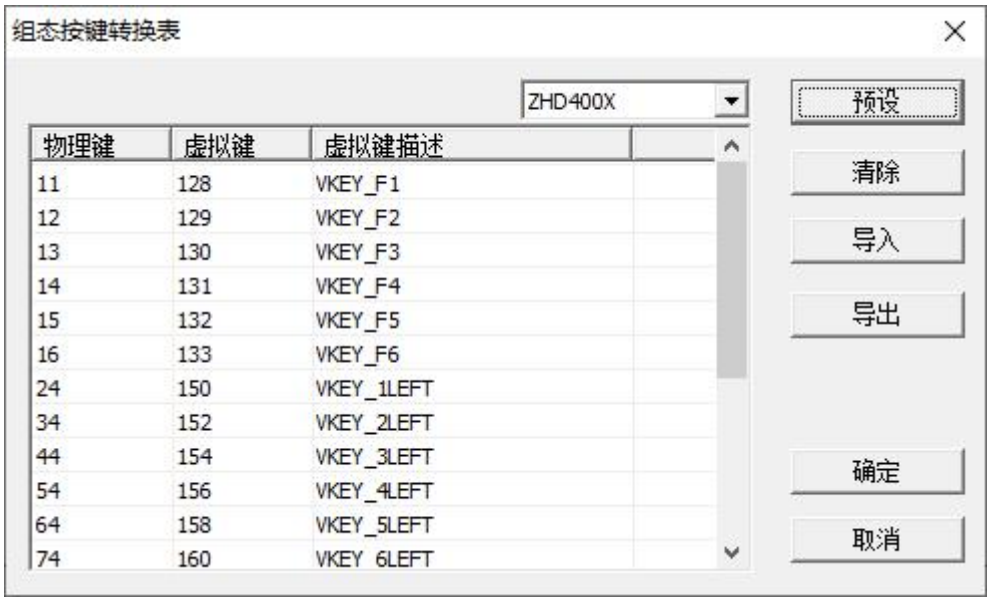
参考例程参见“物理键与虚拟键转换”。

3.20.5. 预设的按键转换表

组态按键转换表内的选择菜单，目前预设有 ZHD300X 和 ZHD400X 的按键转换表，在下拉菜单中先选择 ZHD300X，再点击右侧“预设”，列表区即可显示下图信息，点击“确认”即可应用。



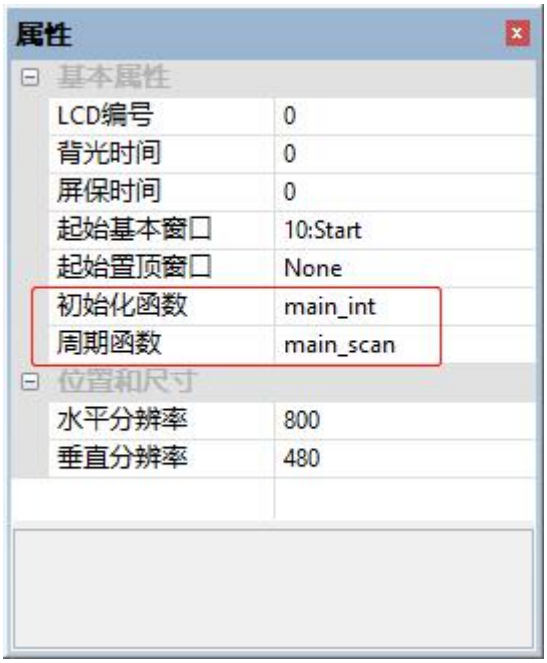
在下拉菜单中选择 ZHD400X，如下图。



第四章 HMI 调用 Basic 函数

4.1.HMI 系统设置

菜单栏“HMI 系统设置”打开如下窗口，初始化函数和周期函数的调用可根据需求选择是否设置。

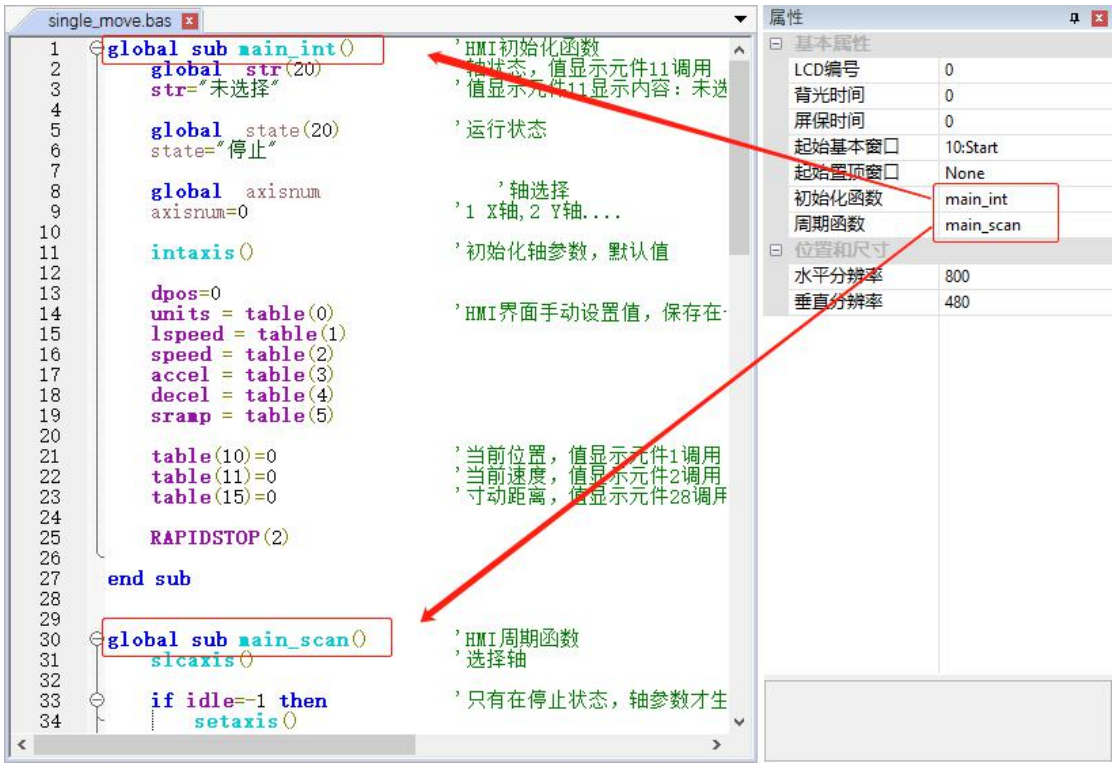


初始化函数：上电后只调用一次的函数，在 Basic 文件中定义，函数的定义必须是全局(GLOBAL)的 SUB。

周期函数：上电后周期不断扫描的函数，在 Basic 文件中定义，函数的定义必须是全局(GLOBAL)的 SUB。

根据组态程序要应用的示教盒的尺寸，设置好水平分辨率和垂直分辨率。选择起始基本窗口（即触摸屏显示的初始界面），初始化函数和周期函数选择 Basic 里编写好的 GLOBAL 全局定义的 SUB 子函数。

初始化程序 HMI 上电后运行一次，周期函数周期循环扫描。



4.2.自定义元件调用函数

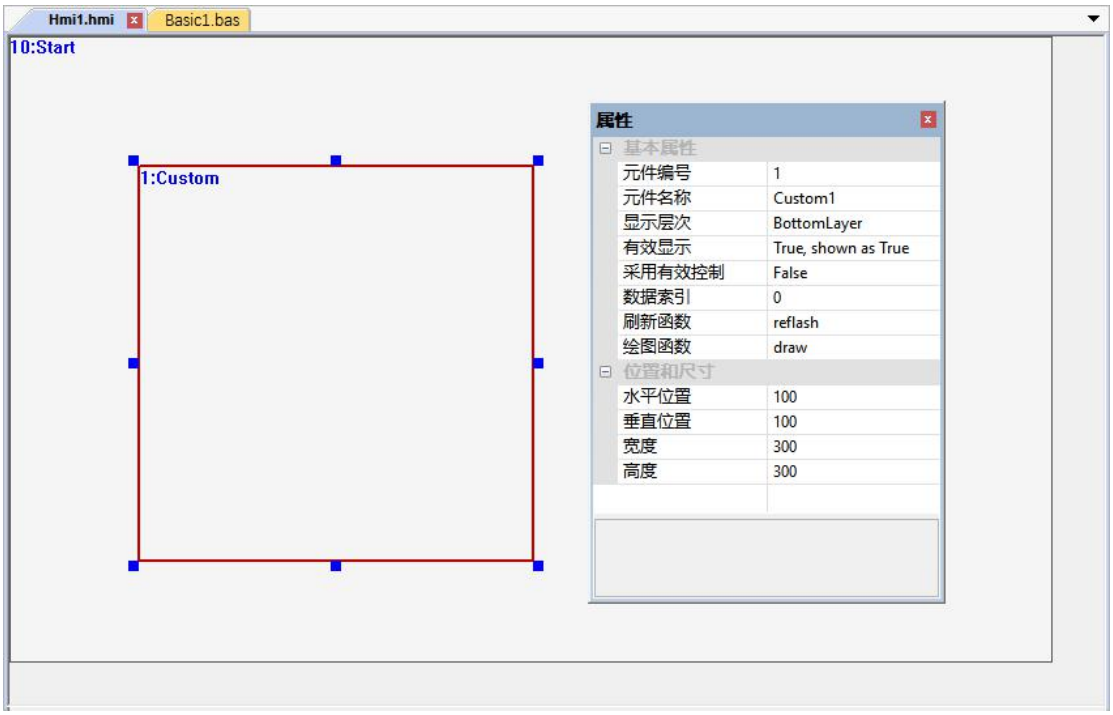
自定义元件 CUSTOM 里可添加绘图函数和刷新函数是由 Basic 编写的全局 SUB 子函数。

绘图函数：周期调用来判断是否要重新绘图，此函数里面通过调用 Draw 相关函数来自己绘图，绘图函数的零点是自定义元件的左上角。

刷新函数：需要绘图时自动被调用，刷新绘图区域，通过调用 SET_REDRAW 指令来指明哪部分区域要刷新。

1．绘图函数参考例程：

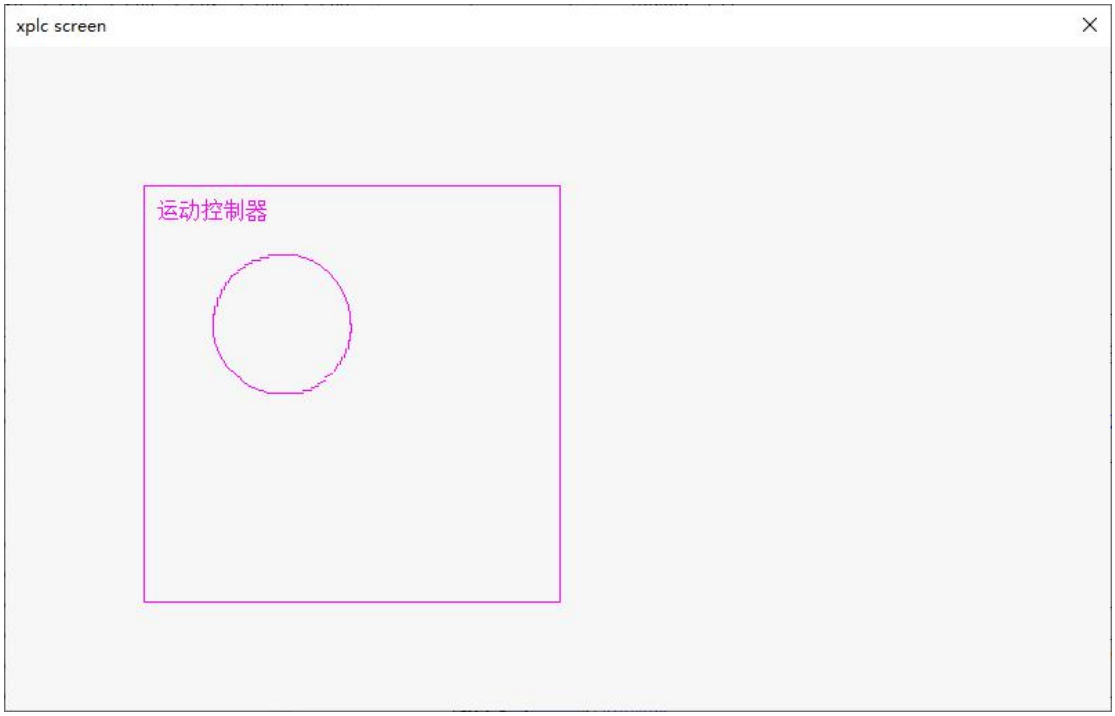
在 hmi 窗口创建一个自定义元件 CUSTOM，打开“属性”窗口，设置元件区域、绘图函数、刷新函数。



Basic 函数：

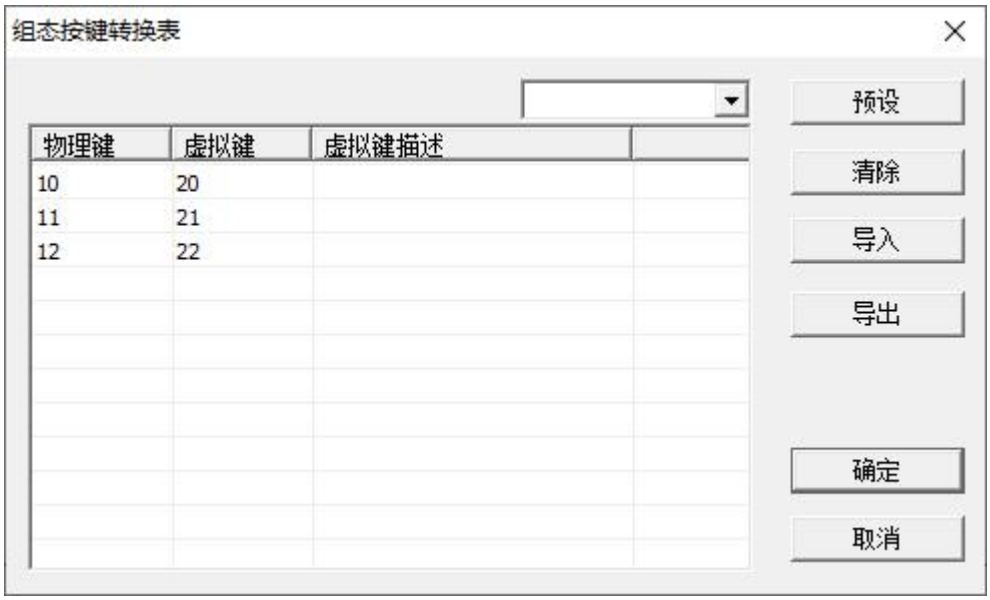
```
GLOBAL SUB reflash()        '刷新函数
    SET_REDRAW
END SUB
GLOBAL SUB draw()           '绘图函数
    SET_COLOR(RGB(255,0,255))          '设置颜色
    DRAWRECT(0,0,300,300)              '自定义元件内绘制边框
    DRAWTEXT(10,10, "运动控制器")      '自定义元件内显示字符串
    DRAWARC (100,100,50, 0, PI*2)      '画一个整圆
END SUB
```

实现效果：



2．刷新函数参考例程：

虚拟键 20、21、22 分别绑定物理键 10、11、12。



Basic 函数：

```
global sub runv()          '绘图函数
    if num=20 then         '物理按键10按下
        print 1            '命令行打印 1

    elseif num=21 then     '物理按键11按下
        table(10)=100      'table10赋值为100

    elseif num=22 then     '物理按键12按下
        function1()        '调用自定义函数
    endif
end sub

global sub slt()           '刷新函数

  if vKEY_SCAN<>0 then
    num=vKEY_SCAN          '扫描虚拟按键，返回值给变量num
  endif
    SET_REDRAW

end sub
```

运行效果：

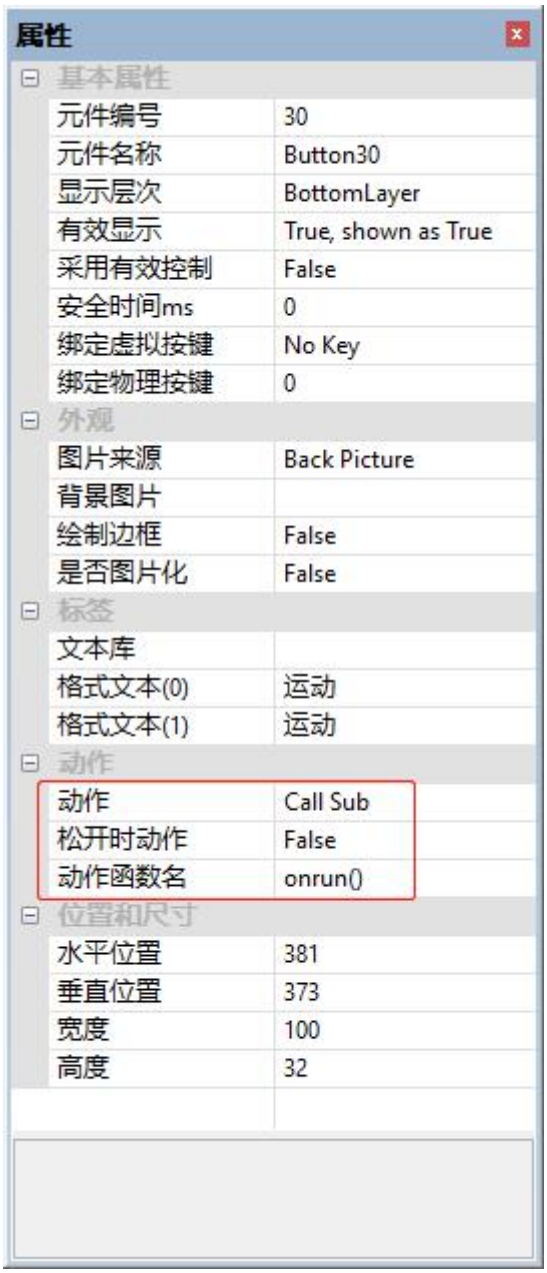
物理键 10 按下时，命令行打印 1。

物理键 11 按下时，TABLE(10)赋值为 100。

物理键 12 按下时，调用函数 function1，函数功能可以自定义。

4.3.元件通用属性：动作

HMI 组态元件动作调用 Basic 自定义全局 SUB 子函数，很多元件都具有调用函数的功能，例如功能键 BUTTON，在“动作”选项中选择 call sub，“动作函数名”选择全局 SUB 子函数，在按下或松开功能键后就能调用 SUB 子函数执行。


属性
基本属性
元件编号 30
元件名称 Button30
显示层次 BottomLayer
有效显示 True, shown as True
采用有效控制 False
安全时间ms 0
绑定虚拟按键 No Key
绑定物理按键 0
外观
图片来源 Back Picture
背景图片
绘制边框 False
是否图片化 False
标签
文本库
格式文本(0) 运动
格式文本(1) 运动
动作
动作 Call Sub
松开时动作 False
动作函数名 onrun()
位置和尺寸
水平位置 381
垂直位置 373
宽度 100
高度 32

第五章 相关 Basic 指令

以下是 HMI 增加的 Basic 指令。

5.1.DMSET

类型	语法指令
描述	数组区域赋值。
语法	dmset arrayname(pos, size, data) pos：起始索引 size：长度 data：设置的数值
适用控制器	通用
例子	DMSET table(0,10,2) '数组区域赋值 FOR i=0 TO 9 PRINT "table",i, table(i) '打印数组 NEXT DMSET table(0,10,3) '数组区域赋值 FOR i=0 TO 9 PRINT "table",i, table(i) '打印数组 NEXT
相关指令	ZINDEX_LABEL,ZINDEX_ARRAY, ZINDEX_CALL

5.2.RUN

类型	任务指令
描述	**新建一个任务来执行控制器上的一个文件。** 重复启动同一任务会报错。 多任务操作指令有： END：当前任务正常结束 STOP：停止指定文件 STOPTASK：停止指定任务 HALT：停止所有任务 RUN：启动文件执行 RUNTASK：启动任务在一个 SUB 或标记上执行
语法	RUN "filename"[, tasknum] filename：程序文件名，BAS 文件可不加扩展名

	tasknum：任务号，缺省查找第一个有效的 RUN "file.HMI", TASKID, [LCDNUM] filename：程序文件名，BAS 文件可不加扩展名 TASKID：任务号
适用控制器	通用
例子	RUN "aaa", 1　　'启动任务 1 运行 aaa.bas 文件

5.3.SCAN_EVENT

类型	输入输出函数
描述	数据变化扫描。 返回值 | 描述 1 | 数据 0 变为非 0 -1 | 数据非 0 变为 0 0 | BOOL 数据没有变化
语法	event = SCAN_EVENT (condition) condition：数据条件表达式
适用控制器	通用
例子	WHILE 1 IF SCAN_EVENT(IN(0)) > 0 THEN　　'IN0 上升沿触发 PRINT "IN0 ON" ELSEIF SCAN_EVENT(IN(0))<0 THEN　　'IN0 下降沿触发 PRINT "IN0 Off" ENDIF WEND

5.4.ZINDEX_LABEL

类型	语法指令
描述	可以为 SUB 函数或数组建立索引指针，后面通过索引指针来调用。
语法	Pointer = zindex_label(subname) subname：数组或 SUB 名称
适用控制器	通用
例子	DIM arr1(100) ‘定义数组 Arr1(0,1) ‘对数组赋值 Pointer = ZINDEX_LABEL(arr1) ‘建立索引指针 PRINT ZINDEX_ARRAY(Pointer) (0) ‘访问数组，打印数组第一位

相关指令	ZINDEX_ARRAY, ZINDEX_CALL

5.5.ZINDEX_ARRAY

类型	语法指令
描述	通过索引指针来访问数组。
语法	var = ZINDEX_ARRAY (pointer)(index) subname：数组或 SUB 名称
适用控制器	通用
例子	DIM arr1(100) ‘定义数组 Arr1(0,1) ‘对数组赋值 Pointer = ZINDEX_LABEL(arr1) ‘建立索引指针 PRINT ZINDEX_ARRAY(Pointer) (0) ‘访问数组，打印数组第一位
相关指令	ZINDEX_CALL,ZINDEX_LABEL

5.6.ZINDEX_CALL

类型	语法指令
描述	通过索引指针来调用 SUB 函数。
语法	ZINDEX_CALL(zidnex) (subpara, …) zidnex：索引指针 subpara：sub 的参数调用
适用控制器	通用
例子	Pointer = ZINDEX_LABEL(sub1) ‘建立索引指针 ZINDEX_CALL(Pointer) (2) ‘调用函数 End SUB sub1(a) PRINT a END SUB
相关指令	ZINDEX_ARRAY,ZINDEX_LABEL

5.7.ZINDEX_VAR

类型	语法指令
描述	通过索引指针来操作变量。
语法	ZINDEX_VAR(zindex) zidnex：通过 ZINDEX_LABEL 生成的 z 指针

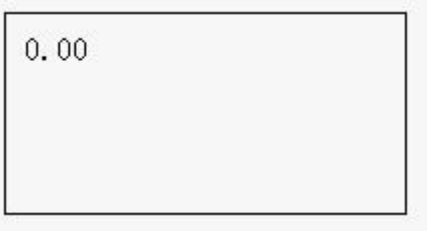
适用控制器	通用
例子	zindex= ZINDEX_LABEL(varname) ZINDEX_VAR(zindex)=value VAR2 = ZINDEX_VAR(zindex)
相关指令	ZINDEX_ARRAY,ZINDEX_LABEL

5.8.LCD_FEATURE

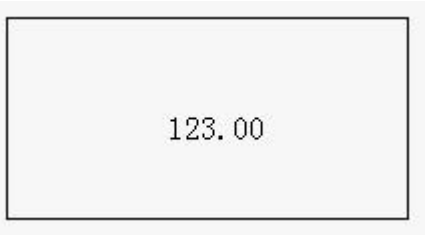
类型	显示指令
描述	显示器的特征读取。必须控制器和显示器同时支持此功能才可以。
语法	var = LCD_FEATURE(lcdnum, featurenum) lcdnum：0-显示器编号 featurenum：特征号 0- type 0-控制器内置的显示器；1-电脑 term；-1-未连接；-2-不存在 300-ZHD300x 400-ZHD400X, 701-7 寸触摸屏. 1- width 显示器物理宽度，0-可变的宽度范围 2- height 显示器物理高度，0-可变的高度范围
适用控制器	支持 ZHMI
例子	PRINT LCD_FEATURE(0,0) ‘打印显示器类型 PRINT LCD_FEATURE(0,1) ‘打印显示器物理宽度 PRINT LCD_FEATURE(0,2) ‘打印显示器物理高度

5.9.DRAWNUM

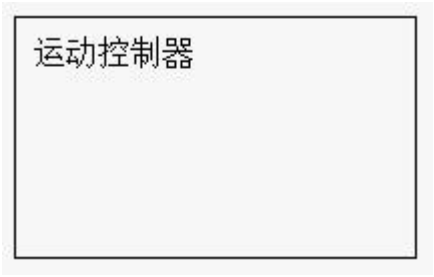
类型	显示指令
描述	显示一个数值。 此指令只能在自定义元件的绘图函数内使用，请查看自定义元件调用函数参考例程。
语法	DrawNum(x,y,value,n,dot) x, y：显示区域左上角的坐标位置 value：缺省显示值 n：总长度位数，包括小数点和符号位。当 N 设置负数时，表示右对齐 dot：小数点后取几位
适用控制器	支持 ZHMI
例子	DRAWRECT(0,0,200,100) '自定义元件内绘制边框 DRAWNUM(10,10,0,4,2) '在自定义元件显示区域左上角（10,10）显示 0.00

	
相关指令	DRAWNUM2

5.10. DRAWNUM2

类型	显示指令
描述	显示一个数值，指定一个方框内显示。 此指令只能在自定义元件的绘图函数内使用，请查看自定义元件调用函数参考例程。
语法	DrawNum2(x1,y1, x2,y2, align,value,n,dot[,drawrect]) x1, y1：显示区域左上角的坐标位置 x2, y2：显示区域右下角的坐标位置 align：对齐选项 0-居中 >0，左边对齐，值表示距离左边的距离 <0，右边对齐，绝对值表示距离右边的距离 value：缺省显示值 n：总长度位数，包括小数点和符号位。当 n 设置负数时，表示右对齐 dot：小数点后取几位 drawrect：可选，0-不画边框（缺省），1-画边框
适用控制器	支持 ZHMI
例子	DRAWRECT(0,0,200,100)　　'自定义元件内绘制边框 DRAWNUM2(10,10,200,100, 0,123,6,2)　'显示在两个坐标位置的中间，保留两位小数，显示结果：123.00 
相关指令	DRAWNUM

5.11. DRAWTEXT

类型	显示指令
描述	显示一个字符串，支持中文，STRING 可以为字符串表达式。 此指令只能在自定义元件的绘图函数内使用，请查看自定义元件调用函数参考例程。
语法	DrawText(x,y,STRING) X, Y：显示区域左上角的坐标位置 STRING：显示的字符串
适用控制器	支持 ZHMI
例子	DRAWRECT(0,0,200,120) '自定义元件内绘制边框 DRAWTEXT(10,10, "运动控制器") '在自定义元件上显示文本“运动控制器” 
相关指令	DRAWTEXT2

5.12. DRAWTEXT2

类型	显示指令
描述	显示一个字符串，指定一个方框内显示。 此指令只能在自定义元件的绘图函数内使用。
语法	DrawText2(x1,y1, x2,y2, align, STRING[,drawrect]) x1, y1：显示区域左上角的坐标位置 x2, y2：显示区域右下角的坐标位置 align：对齐选项 0-居中 >0，左边对齐，值表示距离左边的距离 <0，右边对齐，绝对值表示距离右边的距离 STRING：显示的字符串 drawrect：0-不画边框（缺省），1-画边框
适用控制器	支持 ZHMI
例子	DRAWTEXT2(10,10,100,100,0,"运动控制器",1) '在自定义元件上显示文本，给指定的区域绘制边框

	
相关指令	DRAWTEXT

5.13. DRAWLIBTEXT

类型	显示指令
描述	显示文本库的字符串内容。 此指令只能在自定义元件的绘图函数内使用，请查看自定义元件调用函数参考例程。
语法	DRAWLIBTEXT(x,y, state, “textname”) x, y：显示区域左上角的坐标位置 state：文本库的状态 textname：文本库的名称
适用控制器	支持 ZHMI
例子	DRAWLIBTEXT (10,10, 0, “text1”)
相关指令	DRAWLIBTEXT2,DRAWTEXT,DRAWTEXT2

5.14. DRAWLIBTEXT2

类型	显示指令
描述	格式显示文本库的字符串内容。 此指令只能在自定义元件的绘图函数内使用，请查看自定义元件调用函数参考例程。
语法	DRAWLIBTEXT2(x1,y1, x2,y2, align, state, “textname”[,drawrect]) x1, y1：显示区域左上角的坐标位置 x2, y2：显示区域右下角的坐标位置 align：对齐选项 0-居中 >0，左边对齐，值表示距离左边的距离 <0，右边对齐，绝对值表示距离右边的距离 textname：文本库的名称 drawrect：1 画边框，0-不画边框.
适用控制器	支持 ZHMI
例子	DRAWLIBTEXT2(10,10,100,50,0,)
相关指令	DRAWLIBTEXT,DRAWTEXT,DRAWTEXT2

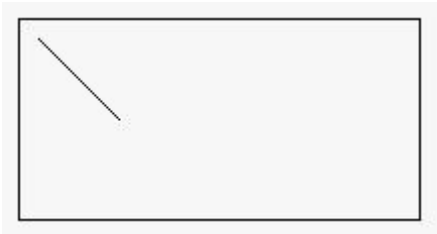
5.15. DRAWREVERSE

类型	显示指令
描述	画一个填充（黑色）的方框。 此指令只能在自定义元件的绘图函数内使用，请查看自定义元件调用函数参考例程。
语法	DRAWREVERSE(x1,y1,x2,y2) x1,y1：显示区域左上角的坐标位置 x2,y2：显示区域右下角的坐标位置
适用控制器	支持 ZHMI
例子	DRAWRECT(0,0,200,100) '自定义元件内绘制边框 DRAWREVERSE(10,10,50,50) '黑色方框填充 

5.16. DRAWRECT

类型	显示指令
描述	画一个方框。 此指令只能在自定义元件的绘图函数内使用，请查看自定义元件调用函数参考例程。
语法	DRAWRECT(x1,y1,x2,y2) x1,y1：显示区域左上角的坐标位置 x2,y2：显示区域右下角的坐标位置
适用控制器	支持 ZHMI
例子	DRAWRECT(0,0,200,100) '自定义元件内绘制边框 

5.17. DRAWLINE

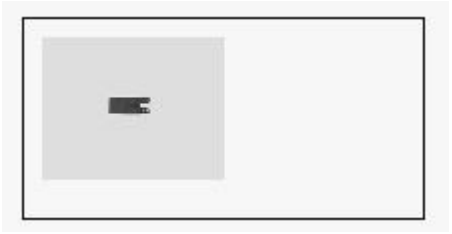
类型	显示指令
描述	画一条直线。 此指令只能在自定义元件的绘图函数内使用，请查看自定义元件调用函数参考例程。
语法	DRAWLINE(x1,y1,x2,y2) x1,y1：直线起始点的坐标位置 x2,y2：直线结束点的坐标位置
适用控制器	支持 ZHMI
例子	DRAWRECT(0,0,200,100) '自定义元件内绘制边框 DRAWLINE(10,10,50,50) '画直线 
相关指令	DRAWARC

5.18. DRAWCLEAR

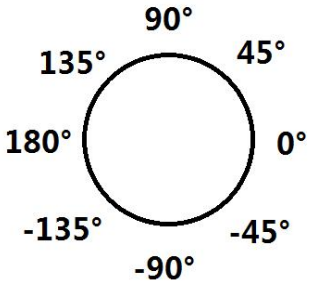
类型	显示指令
描述	清除指定读取区域。 此指令只能在自定义元件的绘图函数内使用，请查看自定义元件调用函数参考例程。
语法	DRAWCLEAR ([x1,y1,x2,y2]) x1,y1：清除区域左上角的坐标位置 x2,y2：清除区域右下角的坐标位置 无参数时全部清除
适用控制器	支持 ZHMI
例子	DRAWCLEAR (10,10,50,50)

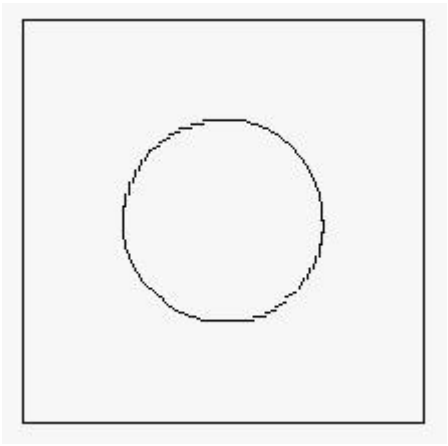
5.19. DRAWPIC

类型	显示指令
描述	绘制图片，图片文件要先加入工程，在文件视图添加图片，注意图片比较占空间，不需要的图片不要加入工程。

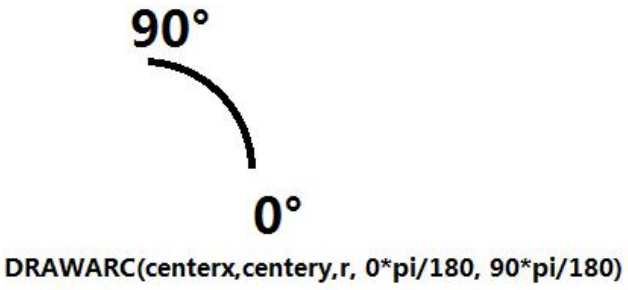
	此指令只能在自定义元件的绘图函数内使用，请查看自定义元件调用函数参考例程。
语法	DRAWPIC (Picname, X1,Y1[,X2,Y2]) picname：图片文件名
适用控制器	支持 ZHMI
例子	DRAWRECT(0,0,200,100) '自定义元件内绘制边框 DRAWPIC ("1.bmp",10,10,100,80) '加入图片 
相关指令	DRAWLIBPIC

5.20. DRAWARC

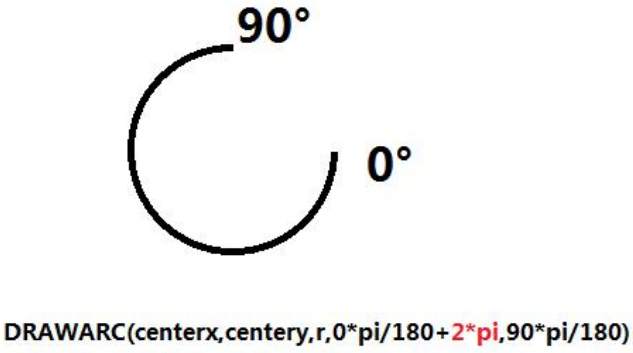
类型	显示指令
描述	画圆弧。 此指令只能在自定义元件的绘图函数内使用，请查看自定义元件调用函数参考例程。
语法	DRAWARC(centx, centy, radius, startangle, endangle) centx, centy：圆心的位置 radius：半径 startangle：起始角度，弧度单位 endangle：结束角度 绘制的角度说明： 
适用控制器	支持 ZHMI
例子	例一：画整圆 DRAWRECT(0,0,200,200) '自定义元件内绘制边框 DRAWARC (100,100,50,0,PI*2) '画一个整圆



例二：画圆弧



例三：需要绘制优弧段时，在起始角度和结束角度二者中，角度小的那个加上 2*pi。



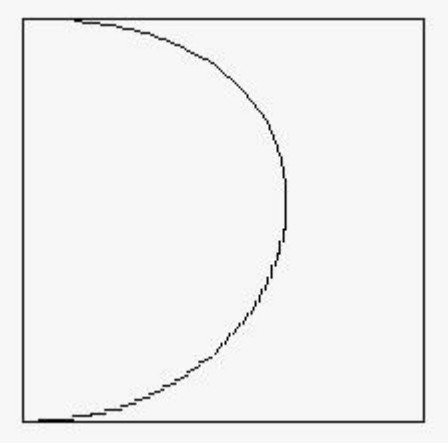
相关指令	DRAWLINE

5.21. DRAWLIBPIC

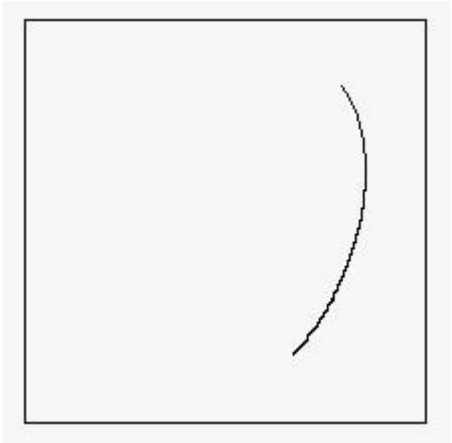
类型	显示指令
描述	绘制图片，图片文件要加入图片库，注意图片比较占空间，不需要的图片

	不要加入。 此指令只能在自定义元件的绘图函数内使用，请查看自定义元件调用函数参考例程。
语法	DRAWLIBPIC (piclibname, state, x1,y1[,x2,y2]) piclibname：图片文件名 state：图片的状态编号 x1,y1：显示区域左上角的坐标位置 x2,y2：显示区域右下角的坐标位置
适用控制器	支持 ZHMI
例子	DRAWLIBPIC (LOGO,0,100,100,300,300)　　'在(100,100)到(300,300)区域显示图片库中名称为 LOGO 的 0 状态
相关指令	DRAWPIC

5.22. DRAWBEZIER

类型	显示指令
描述	画贝塞尔曲线。 此指令只能在自定义元件的绘图函数内使用，请查看自定义元件调用函数参考例程。
语法	DRAWBEZIER(x1,y1,x2,y2,x3,y3,x4,y4) x1,y1：第 1 个控制点 x2,y2：第 2 个控制点 x3,y3：第 3 个控制点 x4,y4：第 4 个控制点
适用控制器	支持 ZHMI
例子	DRAWRECT(0,0,200,200)　　'自定义元件内绘制边框 DRAWBEZIER(0,0,200,0,150,200,0,200)　　'画贝塞尔曲线 

5.23. DRAWBSPLINE

类型	显示指令
描述	画 B 样条曲线。 此指令只能在自定义元件的绘图函数内使用，请查看自定义元件调用函数参考例程。
语法	DRAWBSPLINE (x1,y1,x2,y2,x3,y3,x4,y4) x1,y1：第 1 个控制点 x2,y2：第 2 个控制点 x3,y3：第 3 个控制点 x4,y4：第 4 个控制点
适用控制器	支持 ZHMI
例子	DRAWRECT(0,0,200,200) '自定义元件内绘制边框 DRAWBSPLINE (0,0,200,0,150,200,0,200) '画 B 样条曲线 

5.24. DRAWDTLIST

类型	显示指令
描述	绘图指令，用于加快 TABLE 数据的绘制。 此指令只能在自定义元件的绘图函数内使用，请查看自定义元件调用函数参考例程。
语法	DRAWDTLIST (dtstart, imax, ispace, fxstart, fystart, fxlevel, fylevel, imode , [drawtype], [TYPE1，TYPE2, TYPE3, TYPE4]) dtstart：table 起始位置，指向第一行的行类型 imax：行数 ispace：行间隔 fxstart：左上角的 X 坐标 fystart：左上角的 Y 坐标 fxlevel ：X 方向比例 fylevel：Y 方向比例 imode：存储格式

值	描述
10	DSB 读取存储的格式，此时 X 的位置与行类型偏移 3 个

<table>
<tr><td></td><td>
<table>
<tr><td></td><td>数据</td></tr>
<tr><td>1-9</td><td>此时代表 X 的位置与行类型偏移 N 个数据，XY 的数据紧挨着</td></tr>
<tr><td>其它值</td><td>无效</td></tr>
</table>
drawtype：显示方式
<table>
<tr><td colspan="2">值</td><td>描述</td></tr>
<tr><td colspan="2">0（缺省）</td><td>直接绘制一个点</td></tr>
<tr><td rowspan="14">N*256+ 子显示模式
N= 半径或线宽，>= 1</td><td>N*256+ 1</td><td>单点实心圆弧</td></tr>
<tr><td>N*256+ 2</td><td>单点空心圆弧</td></tr>
<tr><td>N*256+ 3</td><td>单点实心正方形</td></tr>
<tr><td>N*256+ 4</td><td>单点虚正方形</td></tr>
<tr><td>N*256+ 5</td><td>单点十字架</td></tr>
<tr><td>N*256+ 6</td><td>单点叉</td></tr>
<tr><td>N*256+ 9</td><td>单点显示，和上一点之间画一个虚线，N 只能为 1</td></tr>
<tr><td>N*256+ 10</td><td>前后两点之间画直线，N 只能为 1 必须都是同一种类型，当点数过多时相近的点跳过不显示</td></tr>
<tr><td>N*256+ 11</td><td>圆弧中间点，和前后的计算圆弧并显示</td></tr>
<tr><td>N*256+ 12</td><td>圆弧结束点，和前面 2 个点计算圆弧并显示</td></tr>
<tr><td>N*256+ 13</td><td>整圆，和前面 2 个点计算圆并显示</td></tr>
<tr><td>N*256+ 19</td><td>依次为孤立点，直线终点，圆弧结束点，整圆终点，示教例程快速显示</td></tr>
<tr><td>N*256+ 90</td><td>预留数控准系统，标准定义好的类型 G00 G01 G02 G03 等，行类型根据数控准系统设计来</td></tr>
</table>
TYPE1~TYPE4：需要绘制的行类型，最多一次 4 种行类型做同样的绘制</td></tr>
<tr><td>适用控制器</td><td>支持 ZHMI，4 系列 20170515 以上固件</td></tr>
<tr><td>例子</td><td>/</td></tr>
<tr><td>相关指令</td><td>DRAWLINE</td></tr>
</table>

5.25. SET_FONT

类型	显示指令
描述	字体设置，缺省使用内置 16*16 的中文字体，英文为 16*8。但尺寸与字库的尺寸不一致时，会出现缩放；如果需要自己的字体，请使用 zfontmaker 制作专门的字体文件。
语法	SET_FONT (width, height, [fontname]) width：字体宽度，英文自动减半

	height：字体高度 fontname：使用的字体文件名，不设置时不修改当前字体
适用控制器	支持 ZHMI
例子	SET_FONT(16,16, "16X16 字体文件小五.zft")
相关指令	SET_COLOR

5.26. SET_LINE

类型	显示指令
描述	绘图线宽度设置，暂时不支持此指令。
语法	SET_LINE (width) width：字体宽度
适用控制器	支持 ZHMI
例子	/
相关指令	SET_COLOR

5.27. SET_COLOR

类型	显示指令
描述	指定之后 draw 指令使用的颜色，不设置颜色默认黑色。 此指令只能在自定义元件的绘图函数内使用，请查看自定义元件调用函数参考例程。
语法	SET_COLOR (cor[,backcor]) cor：颜色，缺省为黑色。 backcor：DRAWTEXT 时的背景色，不填的时候为透明
适用控制器	支持 ZHMI
例子	例一： SET_COLOR(RGB(255,255,255)) '设置颜色为白色 例二： SET_COLOR(RGB(0,0,0),RGB(255,0,0)) '颜色黑色，DRAWTEXT 显示的字符串背景为红色。 DRAWRECT(0,0,200,100) '自定义元件内绘制边框 DRAWTEXT(10,10, "运动控制器") '自定义元件内显示字符串

	
相关指令	RGB

5.28. SET_REDRAW

类型	显示指令
描述	设置指定区域要重新绘制，自定义元件的刷新函数中使用。 此指令只能在自定义元件的刷新函数内使用，请查看自定义元件调用函数参考例程。
语法	SET_REDRAW ([x, y, width, height]) X, Y：显示区域左上角的坐标位置 width, height：区域宽和高 无参数时绘制全部区域
适用控制器	支持 ZHMI
例子	SET_REDRAW　　‘重新绘制全部区域

5.29. RGB

类型	显示指令
描述	生成一个颜色。
语法	COR = RGB(R,G,B) RGB：代表红、绿、蓝三个通道的颜色 R,G,B：每个分量的颜色 0-255
适用控制器	支持 ZHMI
例子	RGB(255,255,0)　　‘纯黄色
相关指令	SET_COLOR

5.30. HMI_LANG

类型	显示指令
描述	选择语言版本，文本库可以自动切换语言，参见文本库例程
语法	HMI_LANG = ilang

	语言编号 0-7
适用控制器	支持 ZHMI
例子	HMI_LANG=0　　'选择文本库语言编号 0 的内容
相关指令	/

5.31. TOUCH_ADJUST

类型	触摸屏指令
描述	进行触摸屏校正，此时不要刷新屏幕，校正后参数会自动保存。 通过左上，右上，左下，右下，左上，右上，左下，右下的方式连续点击，可以弹出设置窗口，可以进行触摸校正。 不连接控制器，按下 16 (F6)按键，不松开时继续按下 11 (F1)按键
语法	TOUCH_ADJUST ()
适用控制器	支持 ZHMI
例子	TOUCH_ADJUST ()　　'进行触摸校正
相关指令	/

5.32. TOUCH_SCAN

类型	触摸屏指令
描述	扫描触摸按下动作，返回 1 表示扫描到按下，1 表示有松开，0 没有变化，将触摸对应的位置的 X,Y 坐标保存到 table 表中。
语法	TOUCH_SCAN (num) num：存储触摸 XY 坐标的 table 编号，X,Y 坐标分别存储在 table(num), table(num+1)
适用控制器	支持 ZHMI
例子	例一： IF　TOUCH_SCAN(0) = 1　THEN　　'扫描按下操作，显示按下位置 ?"按下的位置为:"　TABLE(0),TABLE(1) ENDIF IF　TOUCH_SCAN(0) = -1　THEN　　'扫描松开操作 ?"松开" ENDIF 例二： IF　TOUCH_SCAN(0)= 1 THEN　'扫描按下操作 TIMES = TIME ENDIF IF　TOUCH_SCAN(0) = -1 THEN　　'扫描松开操作

	?"按下的时间为:" TIME-TIMES ENDIF
相关指令	MOUSE_SCAN

5.33. TOUCH_STATE

类型	触摸屏指令
描述	读取触摸状态，返回 TRUE 表示按下，将触摸对应的位置的 X,Y 坐标保存到 table 表中。
语法	TOUCH_STATE (num) num：触摸对应的位置 X,Y 坐标分别存储在 table(num), table(num+1)
适用控制器	支持 ZHMI

例子	WHILE 1
	IF SCAN_EVENT(TOUCH_STATE(0))>0 THEN
	?"按下的位置为："TABLE(0),TABLE(1)
	ENDIF
	IF SCAN_EVENT(TOUCH_STATE(0))<0 THEN
	?"松开"
	ENDIF
	WEND

相关指令	MOUSE_STATE

5.34. MOUSE_SCAN

类型	触摸屏指令
描述	扫描触摸按下动作，返回 1 表示扫描到按下，-1 表示有松开，0 没有变化，将触摸对应的位置的 X,Y 坐标保存到 table 表中。
语法	MOUSE_SCAN (num) num：触摸对应的位置的 X,Y 坐标分别存储在 table(num), table(num+1)
适用控制器	支持 ZHMI
例子	IF MOUSE_SCAN(0) = 1 THEN '扫描按下操作，显示按下位置 ?"按下的位置为:" TABLE(0),TABLE(1) ENDIF IF MOUSE_SCAN(0) = -1 THEN '扫描松开操作 ?"松开" ENDIF
相关指令	TOUCH_SCAN

5.35. MOUSE_STATE

类型	触摸屏指令
描述	读取触摸状态，返回 TRUE 表示按下，将触摸对应的位置的 X,Y 坐标保存到 table 表中。
语法	MOUSE_STATE (num) 触摸对应的位置的 X,Y 坐标分别存储在 table(num), table(num+1)
适用控制器	支持 ZHMI

例子	WHILE 1
	IF SCAN_EVENT(MOUSE_STATE(0))>0 THEN
	?"按下的位置为："TABLE(0),TABLE(1)
	ENDIF
	IF SCAN_EVENT(MOUSE_STATE(0))<0 THEN
	?"松开"
	ENDIF
	WEND

相关指令	TOUCH_STATE

5.36. KEY_STATE

类型	按键指令
描述	读取物理按键状态，1-按下。 此指令只能在自定义元件的刷新函数内使用，请查看自定义元件调用函数参考例程。
语法	KEY_STATE (keynum) keynum：按键编号
适用控制器	支持 ZHMI
例子	num =KEY_SCAN() IF KEY_STATE (num)=1 THEN '扫描按下操作，打印按键编号 ?"按下" num ENDIF
相关指令	KEY_EVENT,KEY_SCAN

5.37. KEY_EVENT

类型	按键指令

描述	读取物理按键状态变化，1-按下，-1-松开，0-不变。 此指令只能在自定义元件的刷新函数内使用，请查看自定义元件调用函数参考例程。
语法	KEY_EVENT (keynum) Keynum 按键编号.
适用控制器	支持 ZHMI
例子	num =KEY_SCAN() IF KEY_EVENT (num)=1 THEN '扫描按下操作，打印按键编号 ?"按下" num ENDIF
相关指令	KEY_SCAN，KEY_STATE

5.38. KEY_SCAN

类型	按键指令
描述	读取当前按下的物理按键编码，按下返回按键编码，当松开的时候按键编码的负数, 返回 0 表示没有按键状态变化。 物理按键编码值由硬件决定，程序中无法修改，但可以修改与物理键绑定的虚拟键。 此指令只能在自定义元件的刷新函数内使用，请查看自定义元件调用函数参考例程。
语法	value=KEY_SCAN()
适用控制器	支持 ZHMI
例子	num =KEY_SCAN() IF KEY_EVENT (num)=1 THEN '扫描按下操作，打印按键编号 ?"按下" num ENDIF
相关指令	KEY_EVENT,KEY_STATE

5.39. VKEY_STATE

类型	虚拟按键指令
描述	读取虚拟按键状态，1-按下。 此指令只能在自定义元件的刷新函数内使用，请查看自定义元件调用函数参考例程。
语法	VKEY_STATE (keynum) Keynum：按键编号
适用控制器	支持 ZHMI
例子	num =VKEY_SCAN() IF VKEY_STATE (num)=1 THEN '扫描按下操作，打印按键编号

	?"按下" num ENDIF
相关指令	VKEY_SCAN,VKEY_EVENT

5.40. VKEY_EVENT

类型	虚拟按键指令
描述	读取虚拟按键状态变化，1-按下，-1-松开，0-不变。 此指令只能在自定义元件的刷新函数内使用，请查看自定义元件调用函数参考例程。
语法	VKEY_EVENT (keynum) keynum 按键编号
适用控制器	支持 ZHMI
例子	num =VKEY_SCAN() IF VKEY_ EVENT (num)=1 THEN '扫描按下操作，打印按键编号 ?"按下" num ENDIF
相关指令	VKEY_SCAN,VKEY_STATE

5.41. VKEY_SCAN

类型	按键指令
描述	读取当前按下的虚拟按键编码，按下返回按键编码，当松开的时候按键编码的负数，返回 0 表示没有按键操作。 虚拟按键编码内部已经定义，无法修改，但可以修改与虚拟键绑定的物理键。 此指令只能在自定义元件的刷新函数内使用，请查看自定义元件调用函数参考例程。
语法	value=VKEY_SCAN()
适用控制器	支持 ZHMI
例子	Dim Curkey Curkey = VKEY_SCAN() '读取当前按键值(消息码)
相关指令	VKEY_STATE,VKEY_EVENT

5.42. HMI_SHOWWINDOW

类型	窗口操作指令

描述	显示指定窗口。 软键盘窗口要在编辑窗口的相关函数里面调用，否则无法确定是哪个 窗口的元件要编辑。 不确定时默认选择最顶层窗口。
语法	HMI_SHOWWINDOW(winid, [showmode],[modifycontrol]) winid：窗口号 showmode：显示方式 ZPLC_WIN_TYPE_AUTO = 0，HMI 文件里面指定的窗口模式 ZPLC_WIN_TYPE_TOP = 1 ZPLC_WIN_TYPE_BOTTOM = 2 ZPLC_WIN_TYPE_BASE = 4，基本窗口 只能有一个的，一旦 切换所有的子窗口都关闭. ZPLC_WIN_TYPE_KEYBOARD = 5，软键盘必然弹出窗口 ZPLC_WIN_TYPE_POP = 6，弹出窗口 ZPLC_WIN_TYPE_MENU = 7，菜单窗口，自动关闭 Modifycontrol：ZPLY_WIN_TYPE_KEBOARD 弹出软键盘窗口类型时，对应要编辑的本窗口元件 ID
适用控制器	支持 ZHMI
例子	例一 HMI_SHOWWINDOW(13,6)　　‘弹出窗口 13 例二 HMI_SHOWWINDOW(8,5,1)　’弹出软键盘窗口 8，关联当前窗口 1 号元件
相关指令	HMI_CLOSEWINDOW

5.43. HMI_CLOSEWINDOW

类型	窗口操作指令
描述	关闭指定窗口。
语法	HMI_CLOSEWINDOW([winid]) winid：窗口号，缺省 0-当前函数调用的元件所在的窗口，其它编号-HMI 组态里面的窗口号
适用控制器	支持 ZHMI
例子	HMI_CLOSEWINDOW()　‘关闭当前窗口
相关指令	HMI_SHOWWINDOW

5.44. HMI_BASEWINDOW

类型	窗口操作指令

描述	切换基本窗口。
语法	HMI_BASEWINDOW = winid winid：HMI 文件里面窗口编号
适用控制器	支持 ZHMI
例子	HMI_BASEWINDOW = 11 ‘切换到 11 号基本窗口
相关指令	HMI_SHOWWINDOW

5.45. HMI_CONTROLSIZEX

类型	窗口操作指令
描述	获取元件宽度
语法	value= HMI_CONTROLSIZEX ([winid, controlid]) winid：HMI 文件里面窗口编号 controlid：元件编号，缺省为当前自定义元件宽度
适用控制器	支持 ZHMI
例子	PRINT HMI_CONTROLSIZEX(10,11) ‘打印 10 号窗口 11 号元件宽度
相关指令	HMI_CONTROLSIZEY

5.46. HMI_CONTROLSIZEY

类型	窗口操作指令
描述	获取元件高度
语法	value= HMI_CONTROLSIZEX ([winid, controlid]) winid：HMI 文件里面窗口编号 controlid：元件编号，缺省为当前自定义元件高度
适用控制器	支持 ZHMI
例子	PRINT HMI_CONTROLSIZEX (10,11) ‘打印 10 号窗口 11 号元件高度
相关指令	HMI_CONTROLSIZEX

5.47. HMI_CONTROLDATA

类型	窗口操作指令
描述	获取或设置自定义元件的特殊属性，在 HMI 里面指定，通过这个可以区分多个类似的元件。
语法	value= HMI_CONTROLDATA ([winid, controlid]) HMI_CONTROLDATA (winid, controlid) = value

	winid：HMI 文件里面窗口编号 controlid：元件编号，缺省为当前自定义元件
适用控制器	支持 ZHMI
例子	HMI_CONTROLDATA(10,1)=5　‘设置多个自定义元件为相同属性 HMI_CONTROLDATA(10,2)=5
相关指令	HMI_CONTROLSIZEX，HMI_CONTROLSIZEY

5.48. HMI_CONTROLBACK

类型	窗口操作指令
描述	获取或设置值显示及字符显示元件的背景颜色
语法	value= HMI_CONTROLBACK ([winid, controlid]) HMI_CONTROLBACK (winid, controlid) = value winid：HMI 文件里面窗口编号 controlid：元件编号，缺省为当前自定义元件
适用控制器	支持 ZHMI
例子	HMI_CONTROLBACK(10,1)=RGB(255,255,0)　'黄色 HMI_CONTROLBACK(10,1)=RGB(255,0,0)　'红色
相关指令	RGB

5.49. HMI_CONTROLVALID

类型	窗口操作指令
描述	获取或设置元件的使能，在 HMI 里面可以指定。
语法	value= Hmi_ControlValid ([winid, controlid]) Hmi_ControlValid (winid, controlid) = value winid：HMI 文件里面窗口编号 controlid：元件编号，缺省为当前自定义元件 value：为 1 时，元件触摸有作用为 0 时，元件触摸无作用效果
适用控制器	支持 ZHMI
例子	Hmi_ControlValid(10,5)=0　‘10 号窗口第 5 个元件无作用效果
相关指令	/

5.50. HMI_IFMONO

类型	窗口操作指令

描述	自定义元件 reflash 刷新函数里使用，判断当前元件是否被其他窗口垄断，垄断时不要响应鼠标和按键消息，-1-被垄断，0-没有被垄断
语法	Value=HMI_IFMONO
适用控制器	支持 ZHMI

例子	GLOBAL SUB reflash()
	IF SCAN_EVENT(HMI_IFMONO)<0 THEN
	?"返回自定义元件窗口"
	ENDIF
	IF SCAN_EVENT(HMI_IFMONO)>0 THEN
	?"离开自定义元件窗口"
	ENDIF
	END SUB

5.51. HMI_WINDOWSTATE

类型	窗口操作指令
描述	获取窗口状态。 20161112 以后固件版本支持。
语法	value= HMI_WINDOWSTATE (winid [, tablenum]) winid：HMI 文件里面窗口编号 tablenum：存储窗口的位置和大小，顺序存储 posx, posy, sizex, sizey 返回值对应的窗口类型： ZPLC_WIN_TYPE_AUTO = 0，没有显示 ZPLC_WIN_TYPE_TOP = 1，顶层窗口 ZPLC_WIN_TYPE_BOTTOM = 2，底层窗口 ZPLC_WIN_TYPE_BASE = 4，基本窗口 ZPLC_WIN_TYPE_KEYBOARD = 5，软键盘. ZPLC_WIN_TYPE_POP = 6，弹出窗口 ZPLC_WIN_TYPE_MENU = 7，菜单窗口，自动关闭
适用控制器	支持 ZHMI
例子	命令与输出 >> ?HMI_WINDOWSTATE (10) 4 读取结果：4，表示 10 号窗口为基本窗口

5.52. HMI_MOVEWINDOW

类型	窗口操作指令

描述	移动指定窗口。 20161112 以后固件版本支持。
语法	HMI_MOVEWINDOW (winid, posx, posy [, sizex, sizey]) winid：窗口号 posx：水平坐标 posy：垂直坐标 sizex：水平尺寸 sizey：垂直尺寸
适用控制器	支持 ZHMI
例子	HMI_MOVEWINDOW (11,100,100) '将 11 号窗口的显示位置改为（100,100）

第六章 DT 运动函数

为了支持 G 代码的变参数个数，增加 DT 运动函数，指令调用 TABLE 表的参数运动。
没有 ABS 后缀的为相对运动指令，带 ABS 后缀的为绝对运动指令。

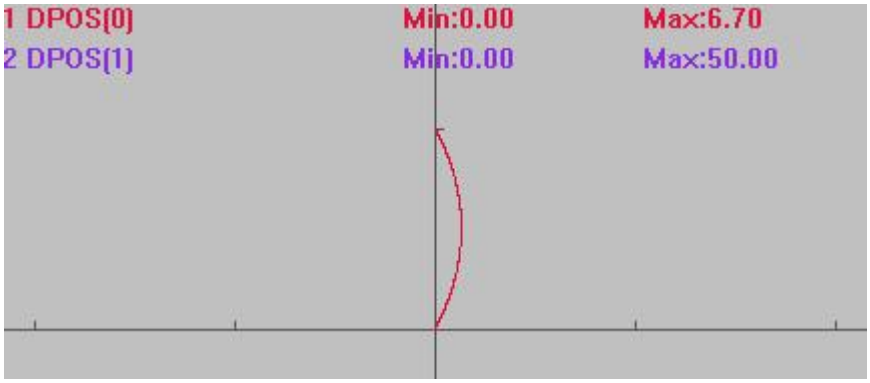
6.1.MOVEDTSP/MOVEDTABSSP

类型	DT 运动函数
描述	将轴号和运动距离分别存放到 TABLE 表，通过 TABLE 列表进行直线运动。使用位存在选择 TABLE 表里的轴号。
语法	MOVEDTSP（最大轴数，位存在，轴 dt 列表起始编号，距离 dt 列表起始编号）
适用控制器	通用

例子	TABLE(10,4,5,6) 'TABLE 表 10 存放 456 三个轴
	TABLE(20,100,50,-10) 'TABLE 表 20 存放三个轴的运距离
	WHILE 1
	IF SCAN_EVENT(IN(0))>0 THEN
	MOVEDTSP(3,5,10,20) '每次运动 TABLE 表中的距离
	'MOVEDTSPABS(3,5,10,20) '运动到 TABLE 表中的位置
	'位存在为 5，转换为二进制 0101，只选择了轴 4 和轴 6 运动
	ENDIF
	WEND

6.2.MOVECIRCDTSP/MOVECIRCDTABSSP

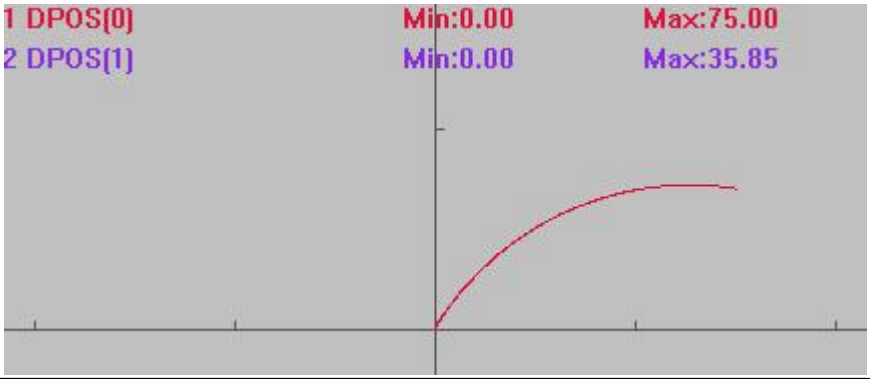
类型	DT 运动函数
描述	将轴号和运动距离分别存放到 TABLE 表，通过 TABLE 列表进行圆弧运动。使用位存在选择 TABLE 表里的轴号。 根据当前点(起始点)和中点的位置，以及半径画圆，圆心自动计算。当终点与起始点的直线距离大于半径时，以这两点画半圆，半径为连线距离一半，圆心为连线中点。
语法	MOVECIRCDTSP（最大轴数，位存在，轴 dt 列表，终点 dt 列表，半径，顺时针 0/逆时针 1）
适用控制器	通用

例子	BASE(0,1,2)
	ATYPE=1,1,1
	DPOS=0,0,0
	TABLE(10,0,1)　'TABLE(10)存放轴 01
	TABLE(20,0,50)　'只需要放一个终点坐标（X，Y）
	TRIGGER
	WHILE 1
	　IF SCAN_EVENT(IN(0))>0 THEN
	　　'MOVECIRCDTABSSP(6,3,10,20,50,1)
	　　MOVECIRCDTSP(6,3,10,20,50,1)
	　　'从起始点经过（0,50），半径为 50，逆时针画圆弧
	　ENDIF
	WEND
	 

6.3.MOVECIRC2DTSP/MOVECIRC2DTABSSP

类型	DT 运动函数
描述	将轴号和运动距离分别存放到 TABLE 表，通过 TABLE 列表进行圆弧运动。使用位存在选择 TABLE 表里的轴号。 根据起始点、参考点、终点三点画圆弧。
语法	MOVECIRC2DTSP（最大轴数，位存在，轴 dt 列表，终点 dt 列表，参考点 dt 列表，mode）

	mode：<0 参考点在当前点的前面 =0 参考点在中间 >0 参考点在当前结束点的后面
适用控制器	通用

例子	BASE(0,1)
	ATYPE=1,1
	DPOS=0,0
	TABLE(10,0,1)　'TABLE(10)存储轴 0,1
	TABLE(20,50,10)　'TABLE(20)存储终点坐标
	TABLE(30,25,25)　'TABLE(30)存储参考点坐标
	TRIGGER
	WHILE 1
	IF SCAN_EVENT(IN(0))>0 THEN
	MOVECIRC2DTSP(3,3,10,20,30,0)
	'MODE=0 时，轨迹经过参考点，运行到参考点+终点的坐标位置
	ENDIF
	WEND
	 

6.4.MSPHERICALDTSP/MSPHERICALDTABSSP

类型	DT 运动函数
描述	将轴号和运动距离分别存放到 TABLE 表，通过 TABLE 列表进行空间圆弧运动。使用位存在选择 TABLE 表里的轴号。根据起始点、参考点、终点三点画圆弧。
语法	MSPHERICALDTSP (轴数，位存在，轴 dt 列表，终点 dt 列表，参考点 dt 列表，mode) mode：指定第二个点的位置 值 | 描述 -1 | 在当前点的前面，只是参考，不运行到参考点 0 | 中间，运行到参考点 1 | 在当前点的和结束点的后面
适用控制器	通用

例子	BASE(0,1,2)
	ATYPE=1,1,1
	DPOS=0,0,0
	TABLE(10,0,1,2)　　'TABLE(10)轴号 0，1,2
	
	TABLE(20,0,0,100)　　'TABLE(20)终点位置
	TABLE(30,30,40,50)　　'TABLE(30)参考点位置
	WHILE 1
	　　IF SCAN_EVENT(IN(0))>0 THEN
	　　　　MSPHERICALDTSP(3,7,10,20,30,0)
	　　　　'MODE=0 时，轨迹从当前点开始，经过参考点，运行参考点+终点坐标位置
	　　ENDIF
	WEND

第七章 参考例程

7.1.单轴运动

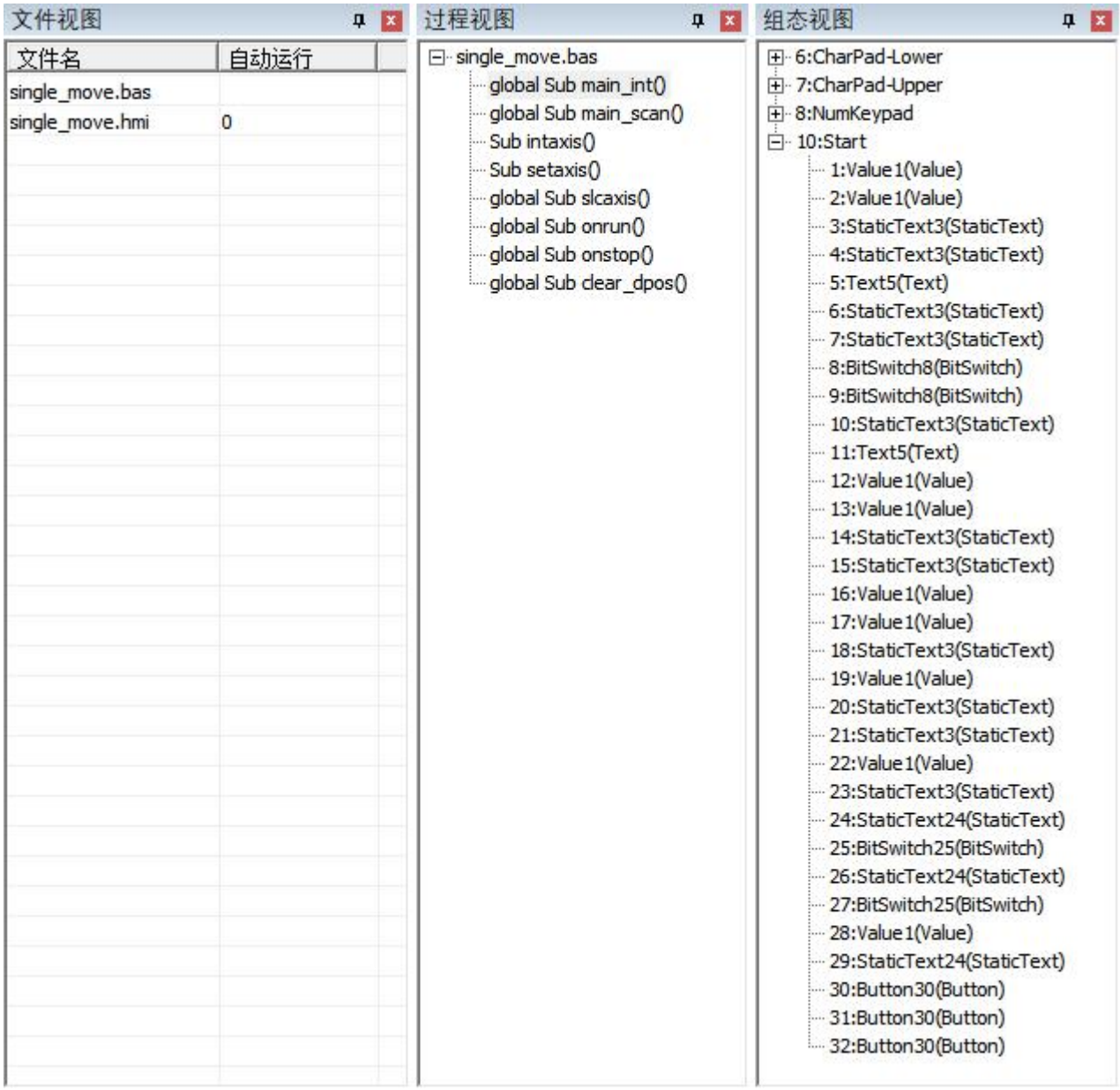
此例程为单轴运动例程，包含两个文件，Basic 的程序由 HMI 调用执行。

文件视图：工程下包含的文件；

过程视图：各文件内包含的 SUB 子过程；

组态视图：组态窗口和组态元件。

在 HMI 编程模式下，经常要使用组态视图切换组态编程窗口。

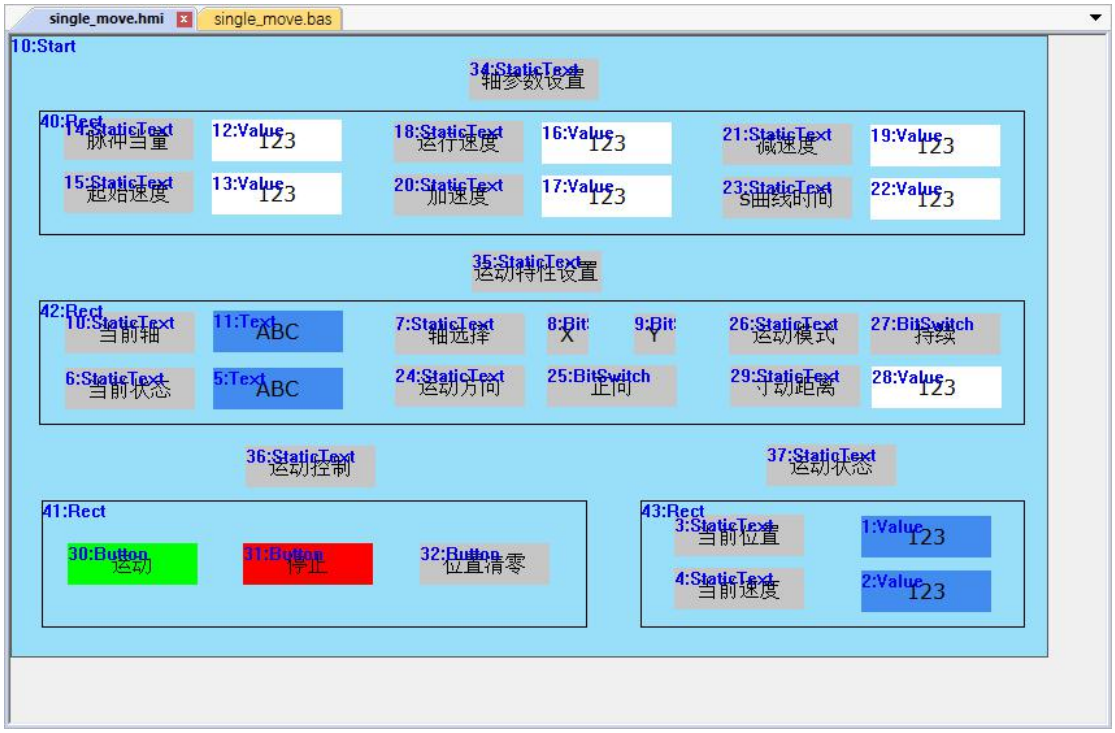


HMI 组态界面：

先选择要运动的轴号，X 轴或 Y 轴，不选择轴号无法运动，再选择运动方向和运动模式，若选择运动模式为寸动，还需要设置寸动距离。

左侧的基本轴参数可自定义设置或采用默认值，调用软键盘窗口自定义输入值，以上设置完成后，可点击运动让轴运动起来，运行的速度 SPEED 和轴位置 DPOS 分被获取到值显示元件 1 和 2 动态显示。

按下停止按钮立即停止当前运动，按下位置清零按钮清零 DPOS。



Basic 程序界面：

single_move.hmi | single_move.bas

```
1  global sub main_int()          'HMI初始化函数
2      global  str(20)            '轴状态，值显示元件11调用
3      str="未选择"               '值显示元件11显示内容：未选择
4
5      global  state(20)          '运行状态
6      state="停止"
7
8      global  axisnum                '轴选择
9      axisnum=0                  '1 X轴,2 Y轴....
10
11     intaxis()                  '初始化轴参数，默认值
12
13     dpos=0
14     units = table(0)           'HMI界面手动设置值，保存在table
15     lspeed = table(1)
16     speed = table(2)
17     accel = table(3)
18     decel = table(4)
19     sramp = table(5)
20
21     table(10)=0                '当前位置，值显示元件1调用
22     table(11)=0                '当前速度，值显示元件2调用
23     table(15)=0                '寸动距离，值显示元件28调用
24
25     RAPIDSTOP(2)
26
27 end sub
28
29
30 global sub main_scan()          'HMI周期函数
31     slcaxis()                  '选择轴
32
33     if idle=-1 then            '只有在停止状态，轴参数才生效
34         setaxis()
```

Basic 程序：

```
global sub main_int()              'HMI 初始化函数
    global   str(20)               '轴状态，值显示元件 11 调用
    str="未选择"                       '值显示元件 11 显示内容：未选择

    global   state(20)             '运行状态
    state="停止"

    global   axisnum                   '轴选择
    axisnum=0                          '1 X 轴,2 Y 轴....

    intaxis()                      '初始化轴参数，默认值

    dpos=0
    units = table(0)               'HMI 界面手动设置值，保存在 table
    lspeed = table(1)
    speed = table(2)
    accel = table(3)
    decel = table(4)
    sramp = table(5)
```

```
    table(10)=0                    '当前位置，值显示元件 1 调用
    table(11)=0                    '当前速度，值显示元件 2 调用
    table(15)=0                    '寸动距离，值显示元件 28 调用

    RAPIDSTOP(2)
end sub

global sub main_scan()      'HMI 周期函数
    slcaxis()                  '选择轴

    if idle=-1 then            '只有在停止状态，轴参数才生效
        setaxis()
    endif

    table(10)=DPOS                 '动态获取显示
    table(11)=MSPEED

    if idle=-1 then
        state="停止"
    endif
end sub

sub intaxis()                  '轴参数初始化
    table(0)=10                    'units   脉冲当量
    table(1)=10                    'lspeed  起始速度
    table(2)=100               'speed   运行速度
    table(3)=1000                  'accel   加速度
    table(4)=1000                  'decel   减速度
    table(5)=10                    'sramp   s 曲线时间
end sub

sub setaxis()                  '轴参数设置
    units = table(0)
    lspeed = table(1)
    speed = table(2)
    accel = table(3)
    decel = table(4)
```

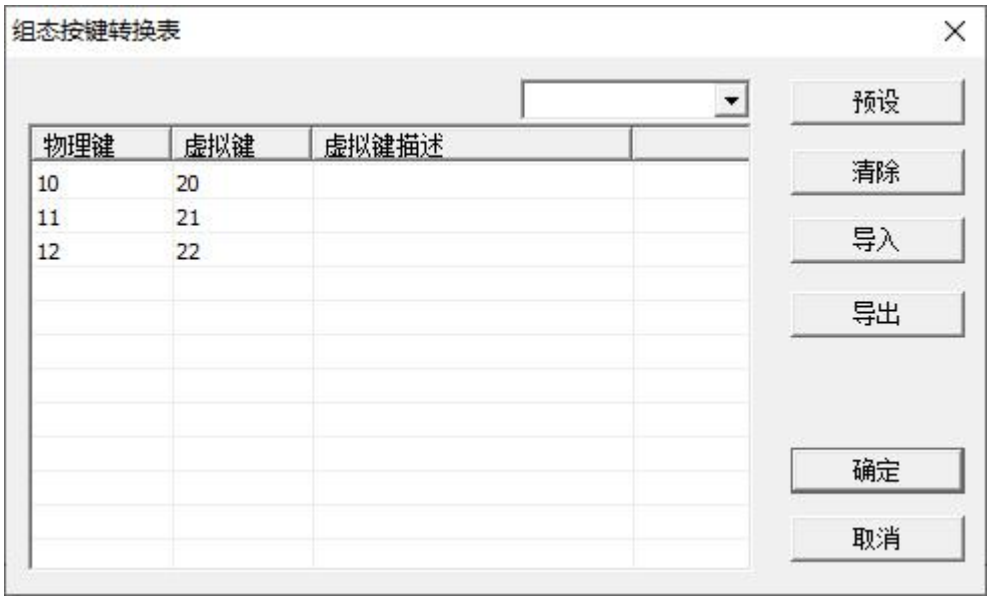
```
    sramp = table(5)
end sub

global sub slcaxis()                    '轴选择函数
    if MODBUS_BIT(0)=1 then             'modbus_bit(0)对应 hmi 界面的 X 轴选择按钮
        cancel(2) axis(1)               '更换选择的轴时，停止 Y 轴 axis1 的运动

        str="X 轴"                      '显示内容为：X 轴
        axisnum=1
        base(0)                         '选定 X 轴
    elseif MODBUS_BIT(1)=1 then         'modbus_bit(1)对应 hmi 界面的 Y 轴选择按钮
        cancel(2) axis(0)               '更换选择的轴时，停止 X 轴 Axis0 的运动

        str="Y 轴"                      '显示内容为：Y 轴
        axisnum=2
        base(1)                         '选定 Y 轴
    endif
end sub

global sub onrun()                      '运动功能键调用
    if axisnum=0 then
        return                              'axisnum=0 未选择轴号
    elseif MODBUS_BIT(20)=0 then        'modbus_bit(20)对应 hmi 界面的运动模式按钮，等于 0 为持续

        if MODBUS_BIT(10)=0 then 'modbus_bit(10)对应 hmi 界面的方向选择按钮
            vmove(1)
        elseif MODBUS_BIT(10)=1 then
            VMOVE(-1)
        endif
    elseif MODBUS_BIT(20)=1 then        '运动模式，等于 1 为寸动
        move(table(15))                 '寸动距离指定，值显示元件 28
    endif

    if idle=0 then
        state="运动"
    endif
```

```
end sub

global sub onstop()          '停止功能键调用
    state="停止"
    RAPIDSTOP(2)
end sub

global sub clear_dpos()      '位置清零功能键调用
    dpos=0
end sub
```

7.2.物理键与虚拟键转换

物理键与虚拟键相关指令只能在自定义元件的刷新函数中使用，参考例程：

虚拟键 20、21、22 分别绑定物理键 10、11、12。



自定义控件“属性”添加如下 Basic 函数：

```
global sub runv()          '绘图函数
    if num=20 then         '物理按键10按下
        print 1            '命令行打印 1

    elseif num=21 then     '物理按键11按下
        table(10)=100      'table10赋值为100

    elseif num=22 then     '物理按键12按下
        function1()        '调用自定义函数
    endif
end sub


global sub slt()           '刷新函数

  if vKEY_SCAN<>0 then
    num=vKEY_SCAN          '扫描虚拟按键，返回值给变量num
  endif
    SET_REDRAW

end sub
```

运行效果：

物理键 10 按下时，命令行打印 1。

物理键 11 按下时，TABLE(10)赋值为 100。

物理键 12 按下时，调用函数 function1，函数功能可以自定义。

7.3.例程下载

更多触摸屏应用例程请前往正运动官方网站下载，网址：www.zmotion.com.cn。

下载路径：官网首页→支持与服务→下载中心→例程资料→触摸屏程序。