

RTHmi 编程手册

Version1.2.0

版权说明

Zmotion®

为了更好的服务于广大受众，深圳市正运动技术有限公司，对所发布的信息（含文字、数据、图片等）作出以下声明：

本手册版权归深圳市正运动技术有限公司所有，严禁任何媒体、网站、个人或组织以任何形式或出于任何目的在未经本公司书面授权的情况下抄袭、转载、摘编、修改本手册内容，或链接、转贴或以其他方式复制用于商业目的或发行，或稍作修改后作为他用，前述行为均将构成对本公司手册版权之侵犯，本司将依法追究其法律责任。

涉及运动控制器软件每个指令和函数的介绍和范例，请参阅正运动技术公司RTBasic/RTPlc/RTHmi 编程手册。

本手册中的信息资料仅供参考。如涉及产品升级，内容需要更改，恕不另行通知！正运动技术公司保留对本资料的最终解释权！如需获取更多详情请登陆正运动技术公司网站。



调试机器要注意安全！请务必在机器中设计有效的安全保护装置，并在软件中加入出错处理程序，否则所造成的损失，正运动技术公司没有义务或责任对此负责。

修订记录

更新日期	版本号	版本（更改）说明	更改人
2023/10/13	V1.0.0	RTHmi 编程手册初版发布	zxl
2024/1/17	V1.1.0	1. 文本库新增“代码页”功能，支持多种语言显示； 2. 补充 HMI_LASTWINDOW 指令说明； 3. HMI_CONTROLATTR、VKEY_MODE、HMI_LISTTEXTS、HMI_DEFAULTATTR、HMI_DEALINFO 指令新增参数及补充示例；	zxl
2024/2/29	V1.1.1	附录虚拟键值说明新增。	zxl
2024/4/7	V1.1.2	1. SCROLLBAR_INIT 指令最多同时滚动条数增加至 32； 2. 其他错误内容修改。	zxl
2024/6/19	V1.1.3	1. HMI_DEALINFO 指令增加参数 EDITNEXT、EDITPREV 获取下一个、上一个编辑控件； 2. HMICONTROLATTR 指令增加参数 FOCUS 设置、获取控件焦点状态； 3. 修改 1.1.6 章节内容； 4. 附录新增 hmi 版本功能介绍； 5. 附录 hmi 版本功能介绍新增：V1.02.04 版本内容。	zxl
2024/12/26	V1.2.0	1. 文本库新增“以语言方式显示”或“以状态方式显示”功能； 2. Hmi 设置：基本属性新增“不使用文本库格式文本”功能，“水平分辨率”和“垂直分辨率”可编辑功能； 3. 窗口导入新增窗口对比功能； 4. 更新图片、完善内容； 5. 控件箱元件新增报表视图、文件浏览器、菜单、树形图，新增章节 4.3.26 – 4.3.29 介绍新增元件； 6. 新增操作指令章节：6.6.22 - 6.6.28； 7. SETEX_ALPHA 指令优化； 8. HMI_DEFAULTATTR 指令新增一个参数"RGB_GRAY"，灰阶颜色； 9. HMI_CONTROLATTR 指令优化：新增参数"MSELROWS"，多选行数，正数往下多选，负数往上多选，0 表示无多选。配合"CURSORY"参数获取光标行可知道当前选中行区域； 10. 新增文件浏览器使用参考例程章节：8.7。	hj

目录

第一章 RTHmi 编程入门	1
1.1. RTSys 编程特点	1
1.1.1. Hmi 组态程序开发	1
1.1.2. Hmi 任务	7
1.1.3. 网络显示屏	8
1.1.4. Hmi 仿真运行	8
1.1.5. Xplcterm 运行	9
1.1.6. 适用的控制器	10
1.1.7. 适用的手持盒	10
1.2. 常见问题	10
第二章 HMI 菜单栏.....	11
2.1. 窗口	11
2.2. 资源	14
2.2.1. 控件箱	15
2.2.2. 文本库	15
2.2.3. 图片库	24
2.2.4. 按键转换	34
2.3. 排列	41
2.4. 编辑	42
2.4.1. 批量修改地址	42
2.4.2. Hmi 设置	45
2.5. 显示设置	46
2.5.1. 属性	46
2.5.2. 快捷图片库	47
2.5.3. 显示/隐藏图层	48
2.5.4. 栅格和元件名称	48
2.6. 语言/状态切换	49

第三章 组态窗口	51
3.1. 窗口概述	51
3.1.1. 窗口作用	51
3.2. 窗口操作	51
3.2.1. 窗口属性	51
3.2.2. 窗口建立	53
3.2.3. 窗口导入	54
3.2.4. 窗口调用	56
3.2.5. 窗口关闭	58
3.2.6. 公共窗口	58
3.3. 窗口类型	61
3.3.1. 基本窗口	61
3.3.2. 软键盘窗口	62
3.3.3. 弹出窗口	63
3.3.4. 菜单窗口	64
3.3.5. 置顶窗口	64
第四章 组态元件	66
4.1. 组态快捷工具	66
4.1.1. 元件菜单	66
4.2. 组态元件通用属性	68
4.2.1. 寄存器	68
4.2.2. 动作	69
4.2.3. 基本属性	70
4.2.4. 外观	71
4.2.5. 位置和尺寸	72
4.2.6. 格式文本	72
4.2.7. 图片库和文本库	74
4.3. 元件介绍及使用	74
4.3.1. 线段/多线段/多边形	75

4.3.2.	矩形	79
4.3.3.	贝塞尔曲线	81
4.3.4.	圆/圆弧/扇形	82
4.3.5.	刻度	86
4.3.6.	表格	88
4.3.7.	导入	90
4.3.8.	静态文本	92
4.3.9.	静态图片	93
4.3.10.	位状态显示	95
4.3.11.	多状态显示	100
4.3.12.	位状态设置	104
4.3.13.	多状态设置	109
4.3.14.	位状态切换开关	116
4.3.15.	多状态切换开关	120
4.3.16.	功能键	127
4.3.17.	物理按键	132
4.3.18.	列表	134
4.3.19.	值	139
4.3.20.	字符显示	143
4.3.21.	滑块开关	147
4.3.22.	定时器	152
4.3.23.	自定义	155
4.3.24.	CAD.....	157
4.3.25.	三次文件编辑器	159
4.3.26.	报表视图	161
4.3.27.	文件浏览器	169
4.3.28.	菜单	171
4.3.29.	树形图	182
第五章	Hmi 调用 Basic 函数	189

5.1.	Hmi 系统设置调用函数	189
5.2.	自定义元件调用函数	190
5.3.	元件调用函数	193
第六章 相关 Basic 指令		194
6.1.	基础指令	194
6.1.1.	RUN -- 启动文件任务	194
6.1.2.	SCAN_EVENT -- 数据状态变化扫描	194
6.1.3.	SET_XPLCTERM -- 显示屏启动状态设置.....	195
6.1.4.	SYSTIME -- 系统时间.....	195
6.2.	语法指令	196
6.2.1.	DMSET -- 数组区域赋值	196
6.2.2.	ZINDEX_LABEL -- 建立索引指针	197
6.2.3.	ZINDEX_ARRAY -- 访问数组.....	197
6.2.4.	ZINDEX_CALL -- SUB 函数调用.....	198
6.2.5.	ZINDEX_VAR -- 指针变量操作	198
6.2.6.	ZINDEX_MARK -- 指针标号设置	198
6.2.7.	ZINDEX_STRUCT -- 获取/访问结构变量.....	199
6.2.8.	ZINDEX_ZVOBJ -- 获取对象索引数据.....	199
6.3.	显示指令	200
6.3.1.	LCD_FEATURE -- 读取显示器特征	200
6.3.2.	LCD_LEDSTATE -- 控制 LED 灯状态.....	200
6.3.3.	LCD_WDOGTIME -- 显示器掉线处理时间.....	200
6.3.4.	DRAWNUM -- 自定义元件内显示数值.....	201
6.3.5.	DRAWNUM2 -- 指定位置显示数值.....	201
6.3.6.	DRAWTEXT -- 绘制显示字符串	202
6.3.7.	DRAWTEXT2 -- 绘制显示字符串	203
6.3.8.	DRAWLIBTEXT -- 显示文本库字符串	203
6.3.9.	DRAWLIBTEXT2 -- 显示文本库字符串	204
6.3.10.	DRAWREVERSE -- 绘制方块	204

6.3.11.	DRAWRECT -- 绘制矩形.....	205
6.3.12.	DRAWLINE -- 绘制线段.....	205
6.3.13.	DRAWCLEAR -- 清除指定区域内容.....	206
6.3.14.	DRAWPIC -- 插入图片文件.....	206
6.3.15.	DRAWARC -- 绘制圆弧.....	206
6.3.16.	DRAWLIBPIC -- 插入图片库图片.....	208
6.3.17.	DRAWBEZIER -- 绘制贝塞尔曲线.....	208
6.3.18.	DRAWBSPLINE -- 绘制 B 样条曲线.....	209
6.3.19.	DRAWDTLIST -- 绘制图形.....	210
6.3.20.	DRAWEX_LINE -- 绘制线段（带样式）.....	214
6.3.21.	DRAWEX_ARC -- 绘制圆弧（带样式）.....	214
6.3.22.	DRAWEX_BEZIER -- 绘制贝塞尔曲线（带样式）.....	216
6.3.23.	DRAWEX_BSPLINE -- 绘制 B 样条曲线（带样式）.....	216
6.3.24.	DRAWEX_RECT -- 绘制圆角矩形（带样式）.....	217
6.3.25.	DRAWEX_ROTRECT -- 绘制旋转矩形（带样式）.....	218
6.3.26.	DRAWEX_ELLIPSE -- 绘制椭圆（带样式）.....	219
6.3.27.	DRAWEX_SECTOR -- 绘制扇形（带样式）.....	220
6.3.28.	DRAWEX_POLYGON -- 绘制多边形（带样式）.....	221
6.3.29.	DRAWEX_POLYGON2 -- 绘制多边形（table 存储）.....	222
6.3.30.	SETEX_LINE -- 设置线段属性.....	223
6.3.31.	SET_FONT -- 字体设置.....	223
6.3.32.	SET_COLOR -- 设置颜色.....	224
6.3.33.	SETEX_ALPHA -- 设置绘图透明度.....	225
6.3.34.	SET_REDRAW -- 重新绘图.....	225
6.3.35.	RGB -- 颜色属性.....	226
6.3.36.	HMI_LANG -- 文本库语言切换.....	226
6.3.37.	SCROLLBAR_FREE -- 释放滚动条.....	226
6.3.38.	SCROLLBAR_INIT -- 滚动条初始化.....	226
6.3.39.	SCROLLBAR_POS -- 获取/设置滚动值.....	227

6.3.40.	SCROLLBAR_REFLASH -- 刷新滚动条.....	228
6.3.41.	SCROLLBAR_DRAW -- 绘制滚动条.....	229
6.4.	触摸屏指令	229
6.4.1.	TOUCH_ADJUST -- 触摸屏校正	229
6.4.2.	TOUCH_SCAN -- 触摸动作扫描.....	230
6.4.3.	TOUCH_STATE -- 获取触摸状态.....	231
6.4.4.	MOUSE_SCAN -- 鼠标动作扫描	231
6.4.5.	MOUSE_STATE -- 获取鼠标状态	232
6.5.	按键指令	233
6.5.1.	MOUSE_WHEEL -- 获取鼠标滚轮值	233
6.5.2.	KEY_STATE -- 获取物理按键状态	233
6.5.3.	KEY_EVENT -- 获取物理按键状态变化.....	233
6.5.4.	KEY_SCAN -- 获取物理按键编码	234
6.5.5.	VKEY_MODE -- 开启虚拟键输入模式	234
6.5.6.	VKEY_STATE -- 设置/获取虚拟键状态	235
6.5.7.	VKEY_EVENT -- 获取虚拟键状态变化.....	236
6.5.8.	VKEY_SCAN -- 获取虚拟键编码	236
6.5.9.	VKEY_INPUT -- 输入虚拟键值内容到键盘窗口	237
6.5.10.	VKEY_IME -- 设置/获取当前输入法	237
6.5.11.	ZSIMU_KEY -- 仿真物理按键	237
6.5.12.	ZSIMU_VKEY -- 仿真虚拟按键.....	238
6.6.	操作指令	238
6.6.1.	HMI_SHOWWINDOW -- 显示指定窗口	238
6.6.2.	HMI_CLOSEWINDOW -- 关闭窗口	239
6.6.3.	HMI_BASEWINDOW -- 切换基本窗口.....	239
6.6.4.	HMI_FOCUSWINDOW -- 窗口焦点模式.....	239
6.6.5.	HMI_LASTWINDOW -- 最后点击窗口	240
6.6.6.	HMI_DEFAULTATTR -- 设置/获取 HMI 内置默认属性	240
6.6.7.	HMI_DEALINFO -- 获取 HMI 处理信息	241

6.6.8.	HMI_CONTROLSIZEX -- 获取元件宽度	242
6.6.9.	HMI_CONTROLSIZEY -- 获取元件高度	242
6.6.10.	HMI_CONTROLDATA -- 设置/获取自定义元件属性	242
6.6.11.	HMI_CONTROLBACK -- 设置/获取指定元件背景颜色	243
6.6.12.	HMI_CONTROLVALID -- 设置/获取元件使能.....	243
6.6.13.	HMI_CONTROLSTRING -- 获取字符串信息.....	243
6.6.14.	HMI_CONTROLATTR -- 设置/获取元件属性	244
6.6.15.	HMI_CONTROLTEXT -- 修改元件文本.....	245
6.6.16.	HMI_LISTTEXTS -- 修改列表元件文本	246
6.6.17.	HMI_LISTITEM -- 修改指定列表项文本.....	247
6.6.18.	HMI_STRAPPEND -- 元件文本追加	248
6.6.19.	HMI_IFMONO -- 获取窗口垄断状态	249
6.6.20.	HMI_WINDOWSTATE -- 获取窗口状态	249
6.6.21.	HMI_MOVEWINDOW -- 移动指定窗口.....	250
6.6.22.	HMI_TABLEVALUE -- 设置/获取表格数值.....	250
6.6.23.	HMI_TABLETEXT -- 设置/获取表格内容	错误!未定义书签。
6.6.24.	HMI_TABLECURSOR -- 获取当前选中行列.....	252
6.6.25.	HMI_FILESE-- 获取当前选中文件	253
6.6.26.	HMI_FILEPATH -- 获取当前路径	253
6.6.27.	HMI_FILEFILTER -- 设置文件过滤器.....	254
6.6.28.	HMI_MENUITEM -- 菜单项属性获取/修改.....	255
第七章 DT 运动函数.....		错误!未定义书签。
7.1.	MOVEDTSP/MOVEDTABSSP -- DT 直线运动.....	257
7.2.	MOVECIRCDTSP/MOVECIRCDTABSSP -- DT 圆弧运动.....	257
7.3.	MOVECIRC2DTSP/MOVECIRC2DTABSSP -- DT 三点画圆弧运动.....	258
7.4.	MSPHERICALDTSP/MSPHERICALDTABSSP -- DT 空间圆弧运动	259
第八章 参考例程		261
8.1.	单轴运动	261
8.2.	物理键与虚拟键转换	266

8.3.	动态列表	269
8.4.	视图缩放	272
8.5.	滚动条使用	277
8.6.	CAD 功能.....	281
8.6.1.	CAD 导入矢量图片	281
8.6.2.	CAD 结合三次文件使用	284
8.6.3.	CAD 结合自定义元件使用	289
8.7.	文件浏览器使用	292
8.8.	例程下载	303
附录	304
RTHmi 版本功能介绍		304
虚拟键值说明		306
错误码列表		307

第一章 RTHmi 编程入门

RTHmi 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 RTHmi 功能。

编写和调试 RTHmi 程序需要使用 RTSys 软件，支持 RTHmi 功能的运动控制器或仿真器 V5.20-20230706 以上版本固件，PC 在线命令发送需要 zmotion.dll 动态库。

RTSys 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。（注：不推荐高版本创建的项目使用低版本的去打开及修改！可能导致项目异常且无法复原。）

本文主要以 RTSys 平台介绍 RTHmi 编程。

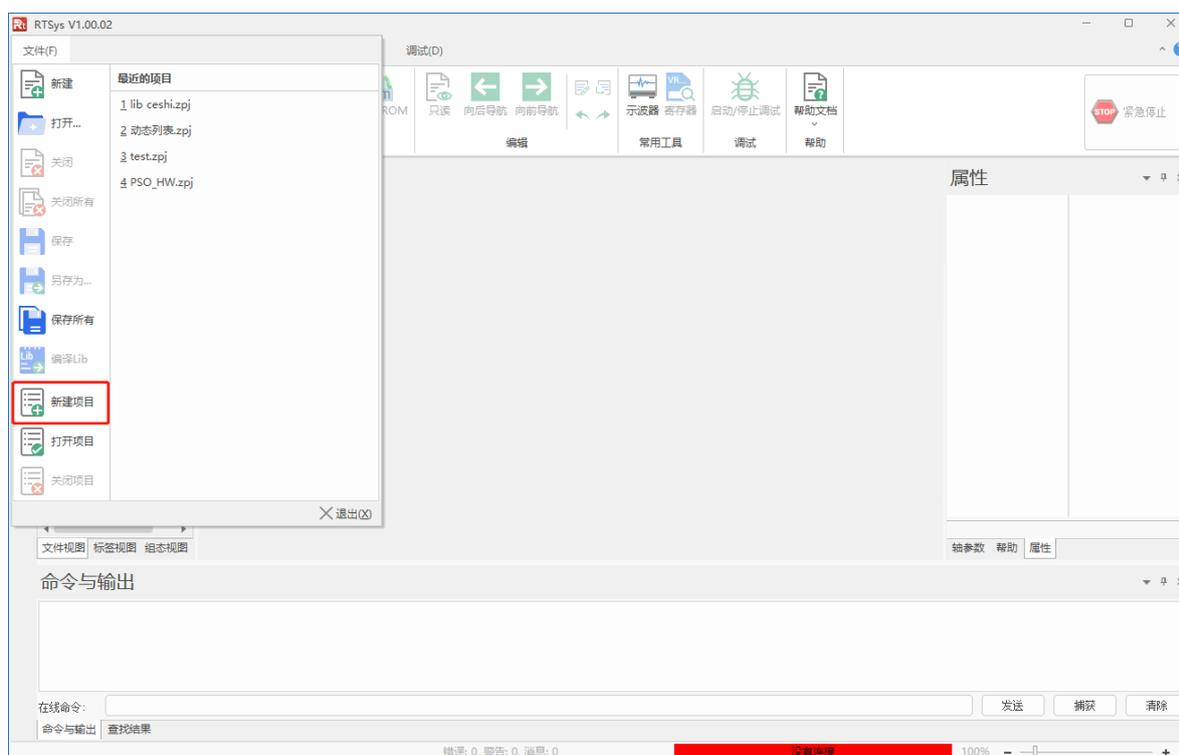
1.1. RTSys 编程特点

1.1.1. Hmi 组态程序开发

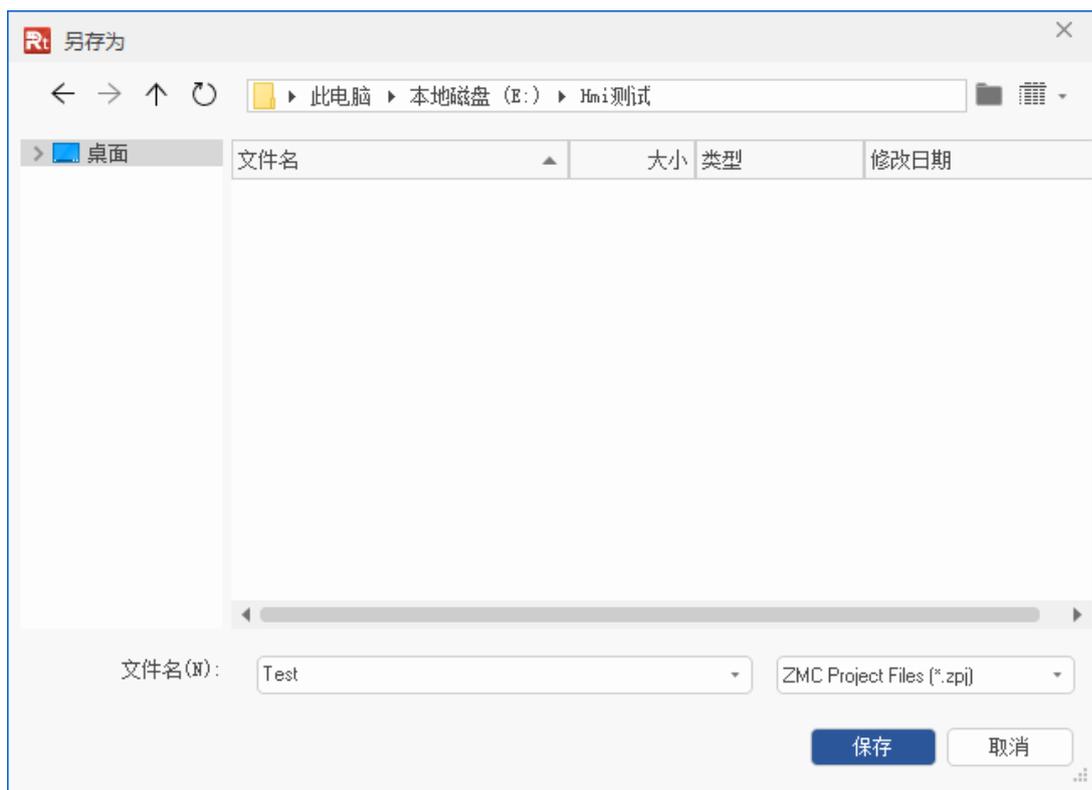
在电脑里新建一个文件夹用来保存即将要建立的工程。打开 RTSys 编程软件，当前说明例程的 RTSys 软件版本为 V1.00.02，更新软件版本请前往正运动官方网站下载，官方网站上还有触摸屏例程提供下载，网址：www.zmotion.com.cn。

RTSys 编程开发流程：

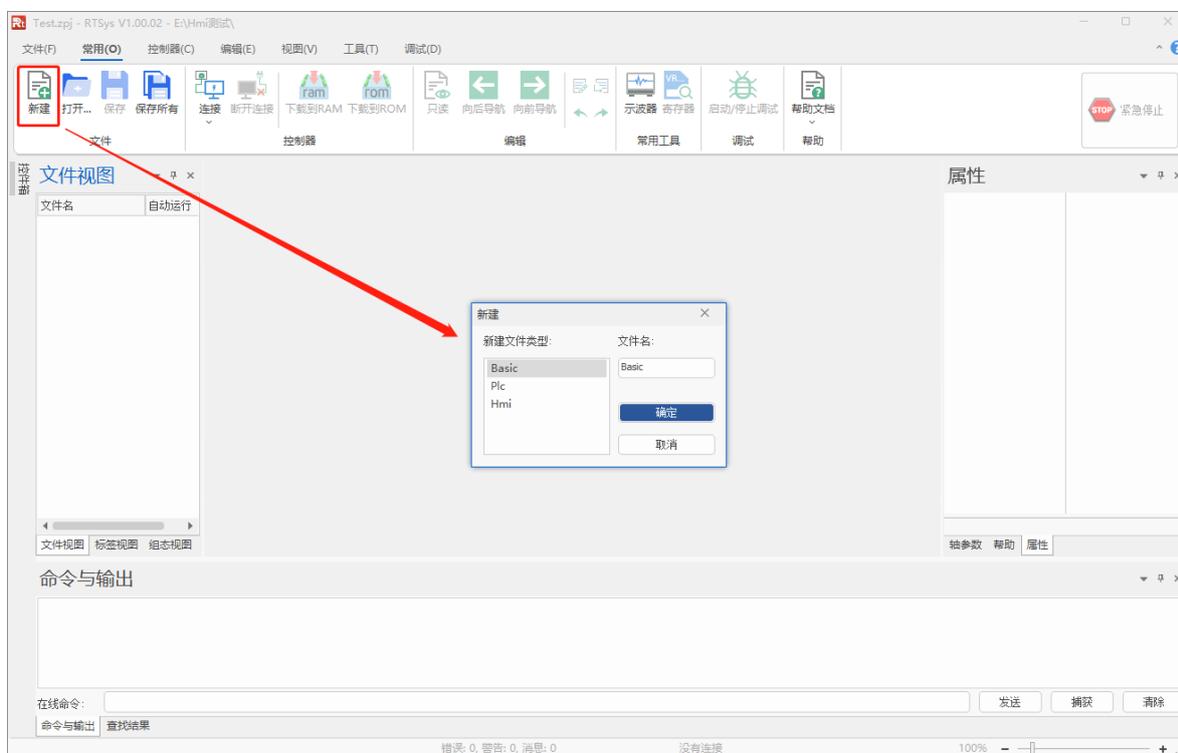
1. 新建项目：菜单栏“文件”-“新建项目”。



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”。

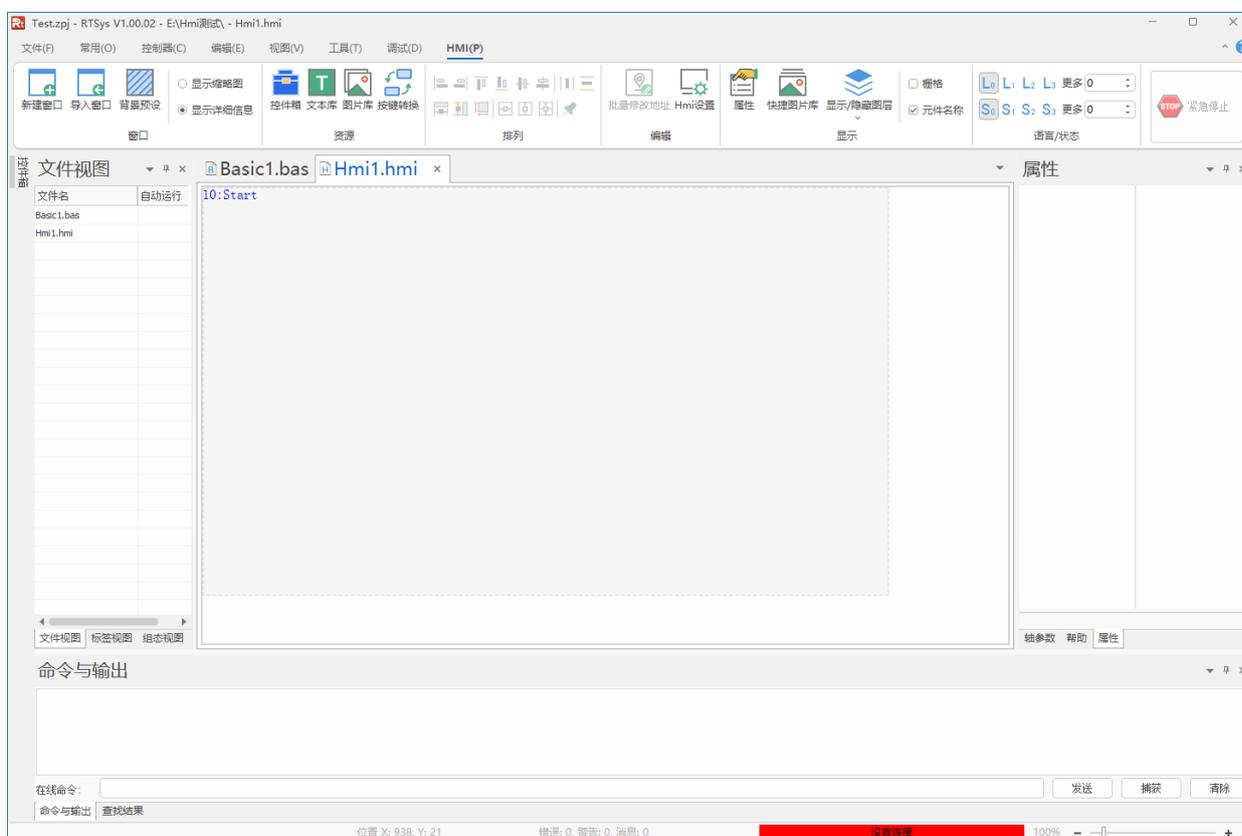


点击“新建文件”后，出现下图弹窗，Hmi 一般和 Basic 混合编程，用 Basic 函数编写 Hmi 元件要实现的功能，分别新建一个 Basic 文件和一个 Hmi 文件。

Basic/Plc/Hmi 分别针对 3 种不同类型的文件，表示 RTSys 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。

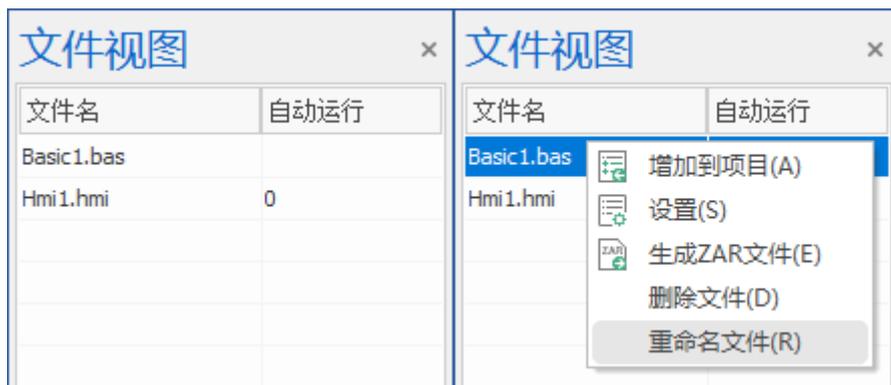


3. 设置文件自动运行：双击文件右边自动运行的位置，输入任务号“0”。

注：1. 任务号可为任意数字，但不可设置超过控制器支持的最大任务数。

2. 设置了任务号的文件会自动同时运行，任务号数值不分优先级。

文件名称可重新自定义，先在该项目内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 设置”窗口。

先切换到 Hmi 编程窗口，菜单栏“HMI”-“Hmi 设置”打开如下窗口（或者打开 Hmi 文件点击窗口外的空白地方也可以显示该属性窗口），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。

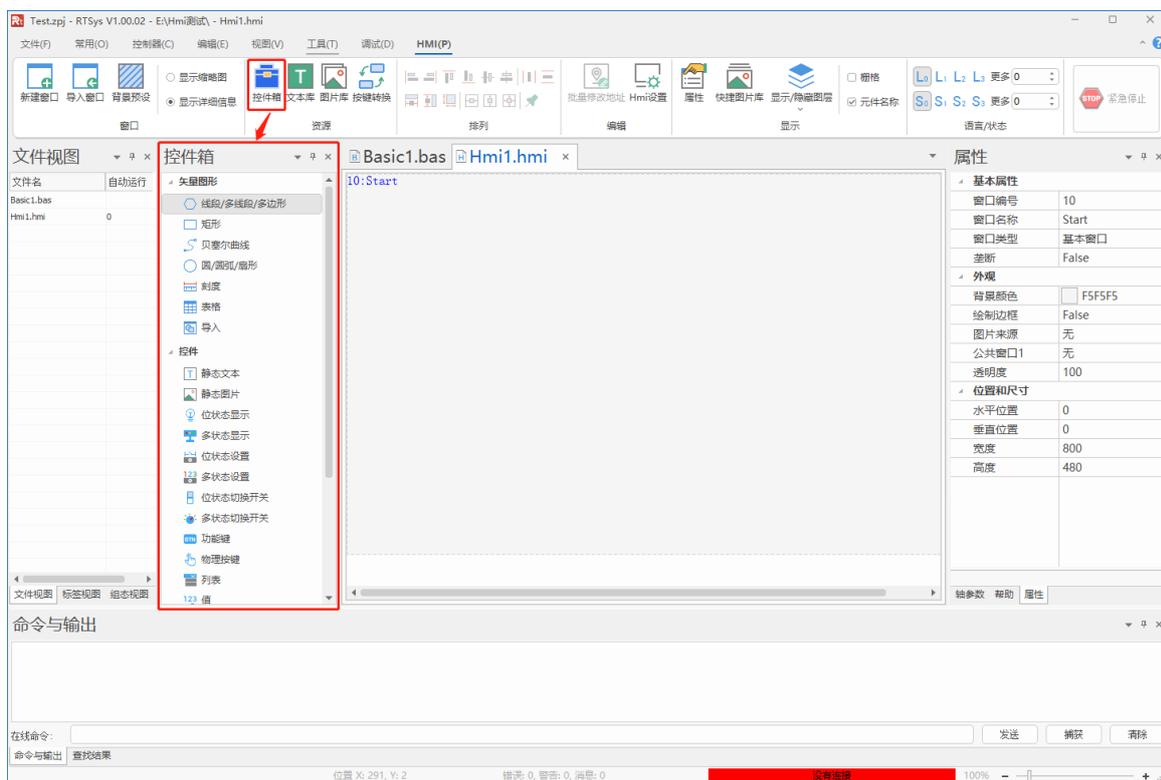
Hmi 属性窗口如下图，各属性含义请参考[第二章 Hmi 设置](#)章节



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。

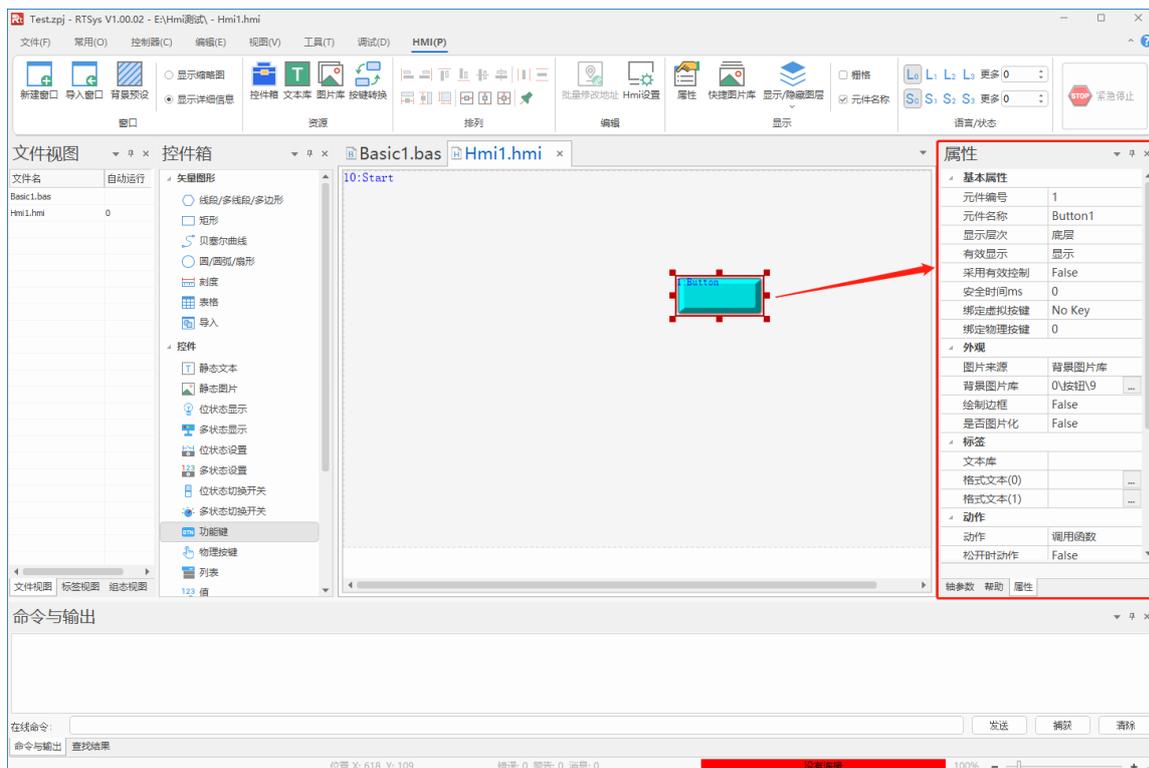
Hmi 编程所需的窗口可在“Hmi”的菜单栏中进行新建和导入，组态元件则在菜单栏“HMI”→“控件箱”里

进行选择，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三章和第四章。



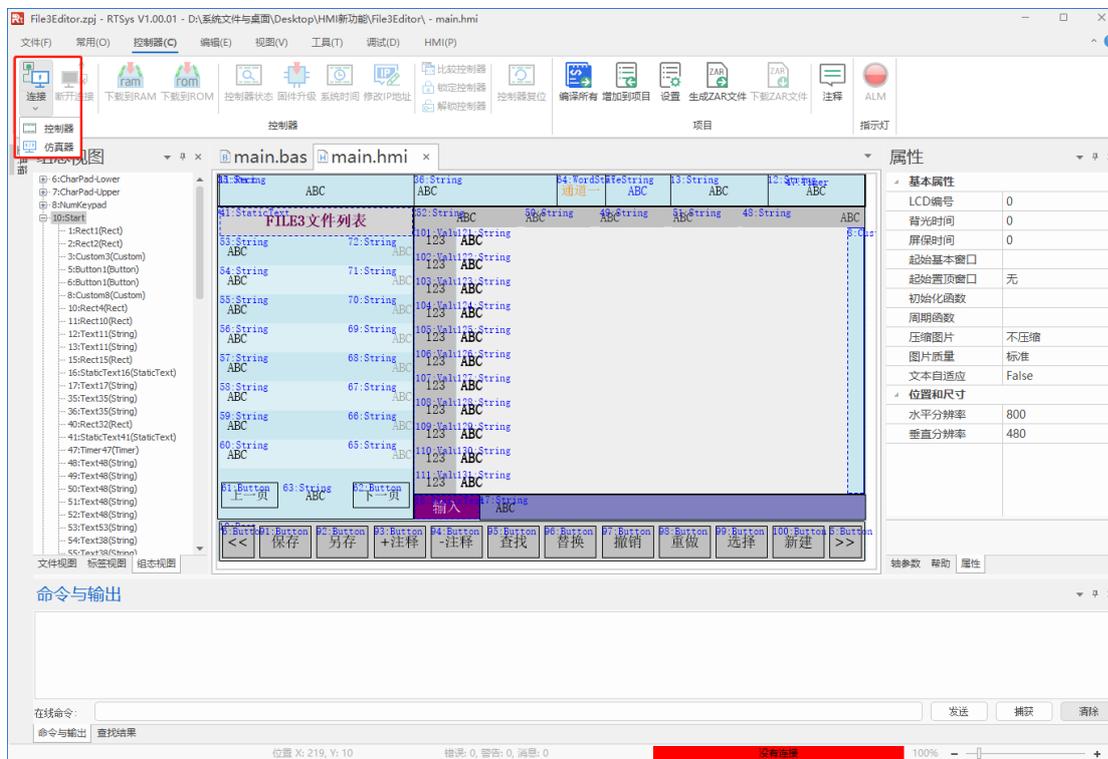
在“控件箱”中选择元件后，将元件放置于组态窗口尺寸范围内，打开元件“属性”设置元件相关参数，如下图所示。

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“常用”/“控制器”-“连接”控制器或仿真器。若选择连接控制器则将弹出“连接到控制器”的窗口，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：成功连接到控制器 ZMC406:

命令与输出

```
Connected to Controller:ZMC406 Version:4.93-20230531.
```

注意：使用 RTHmi 必须控制器固件支持！若固件不支持则同时打印如下信息：

命令与输出

```
Connected to Controller:ZMC406 Version:4.93-20230307.
Controller not support RTHmi function.
```

成功连接到仿真器:

命令与输出

```
Connected to Controller:VPLC5xx-Simu Version:5.20-20190529.
```

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：

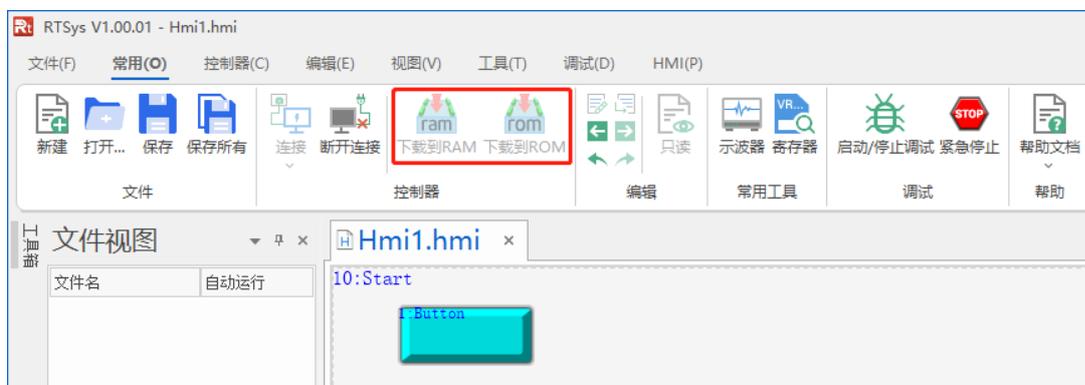
1. ZMC00x 系列控制器不支持下载到 RAM。
2. 未建立项目只打开文件则无法下载到控制器。
3. 自动运行的数字 0 表示任务编号 0，以任务 0 运行程序，任务编号不具备优先级。
4. 若整个工程项目内的文件都不设置任务编号，下载到控制器时，系统提示如下信息。

WARN: no program set autorun.

命令与输出

```
WARN: no program set autorun.
Down to Controller Ram Success, 2023-03-06 14:34:05, Elapsed time: 1094ms.
```

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 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。



文件名	自动运行
Basic1.bas	
Hmi1.hmi	0

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。

1.1.3.网络显示屏

RTHmi 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。

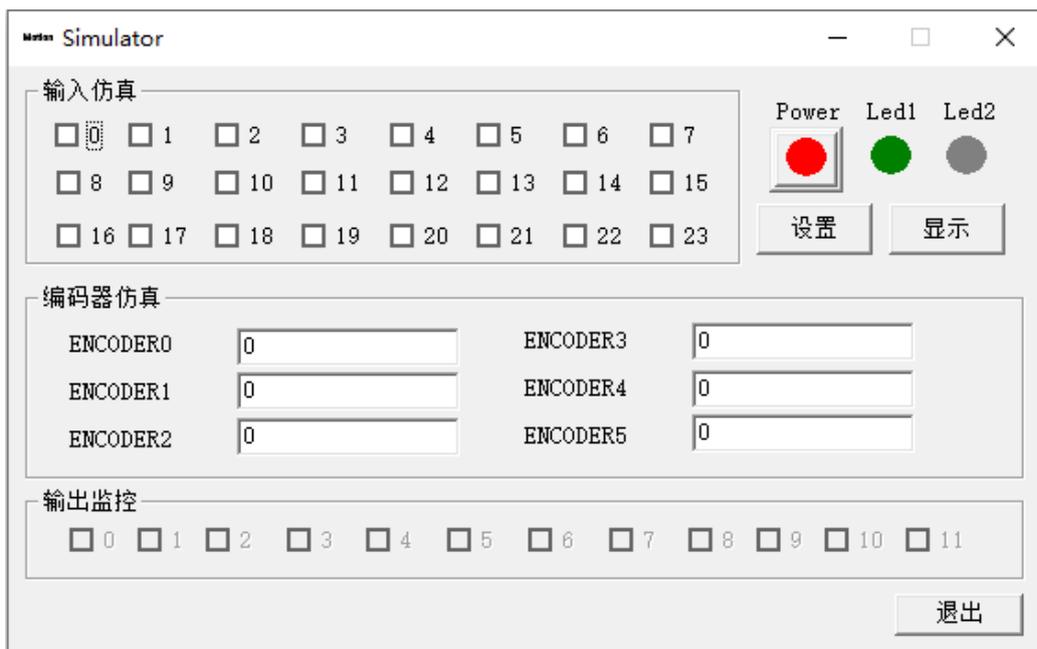
例如：某型号控制器打印 max_Hmi 结果如下

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。

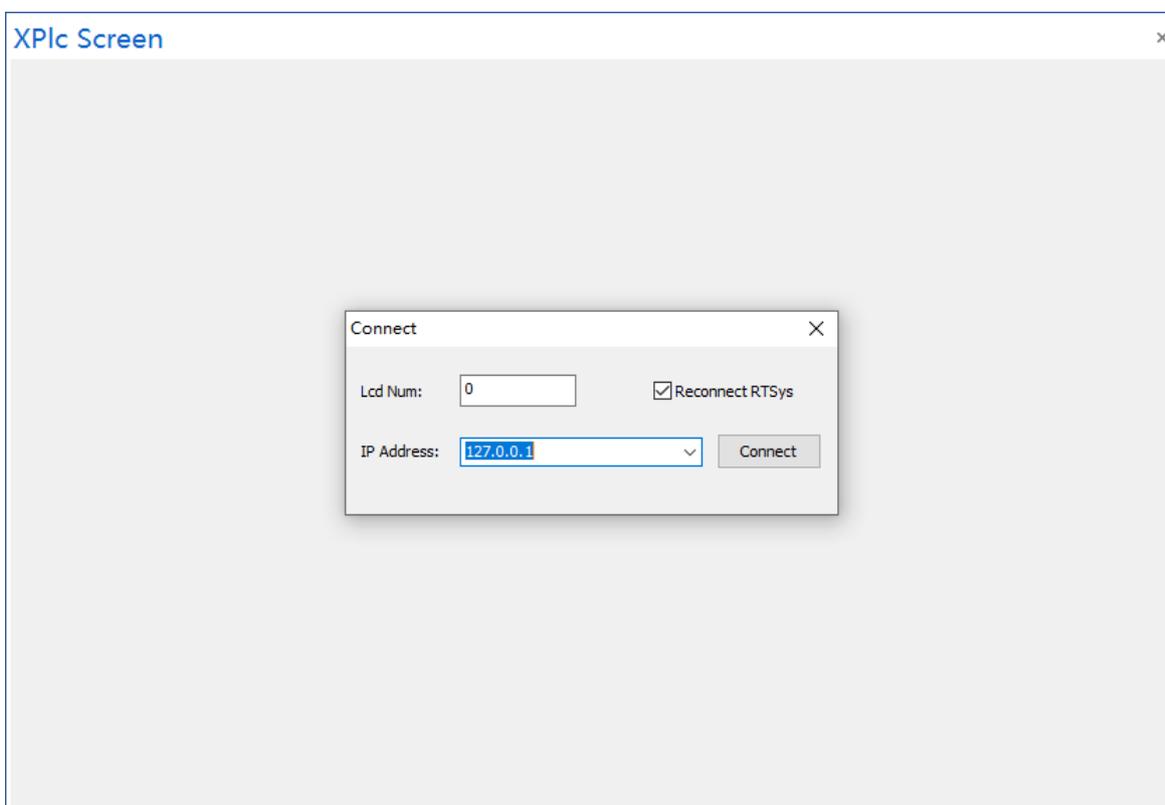
在不连接控制器和示教盒的时候，可以连接到仿真器之后调试。



1.1.5.Xplcterm 运行

在没有触摸屏的时候，使用 Xplc Screen 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 Xplc Screen，点击菜单栏“工具”-“插件”，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 RTHmi 编程方式的控制器型号有：**4 系列及 4 系列以上相应固件支持(4 系以下暂不支持 RTHmi, 型号带 H 的除外, 例如: XPLC120H 等)**，部分控制器需要升级固件才能支持，详情请联系厂家。

(其他非上述支持的低系列型号，建议使用 Zdevelop 进行开发)

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	ZHD500X
分辨率	480*272	800*480	1024*600
按键	47 个	18 个	16 个
USB 口	1 个	1 个	1 个
急停开关	1 个	1 个	1 个
触摸屏	支持	支持	支持

1.2.常见问题

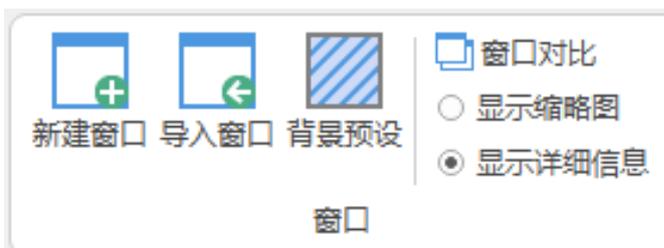
当程序运动出错后，RTSys 软件“命令与输出”窗口会显示出出错信息，如果出错信息没有看到，可以通过命令行输入?*task 或“故障诊断”窗口再次查看出错信息，双击出错信息可以自动切换到程序出错位置。

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
修改 Hmi 分辨率后页面显示不正常	修改 Hmi 分辨率后选择“否”，又将数值恢复选择了“是”，导致页面分辨率异常。此时可以采用“ctrl + z”撤回。或关闭 RTSys 时选择“不保存”即可恢复。
自定义元件无法刷新显示	在刷新函数中，没有使用 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	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，否则请使用串口的 24V 供电；
通讯不上	检查网线。

第二章 HMI 菜单栏

注：HMI 菜单栏仅在主窗口显示 Hmi 文件时才显示。

2.1. 窗口



新建窗口：在当前项目中新建一个或多个窗口。详情可查看第三章【[窗口建立](#)】章节。

导入窗口：在当前项目导入其他项目/文件中已创建的 Hmi 窗口。详情可查看第三章【[窗口导入](#)】章节。

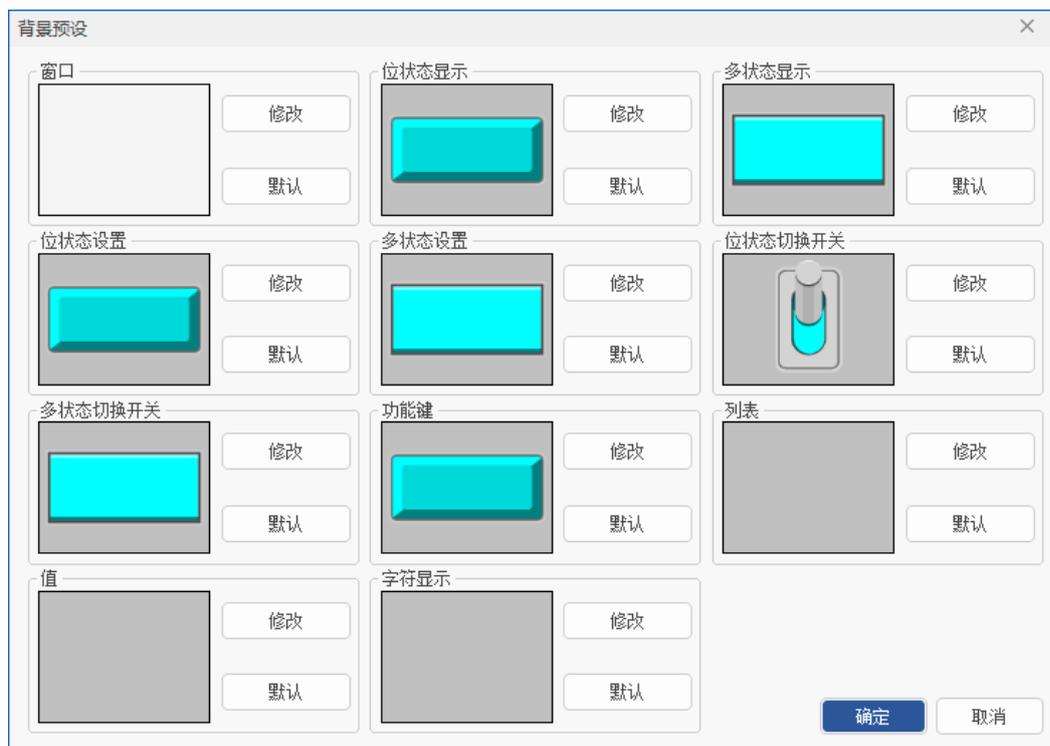
背景预设：对 Hmi 窗口背景及部分元件设置默认的样式/颜色，**保存设置后在新建窗口或元件时生效。**对已创建的窗口和元件不更改原样式。详细使用方法可参考本章【[图片库](#)】章节。

背景预设操作方法：选择需更改样式的元件/窗口，点击“修改”，有两种样式方式选择。

方法一：若使用图片库样式则选择“图片库”可将图片导入，更改图片比例可调整元件显示大小；

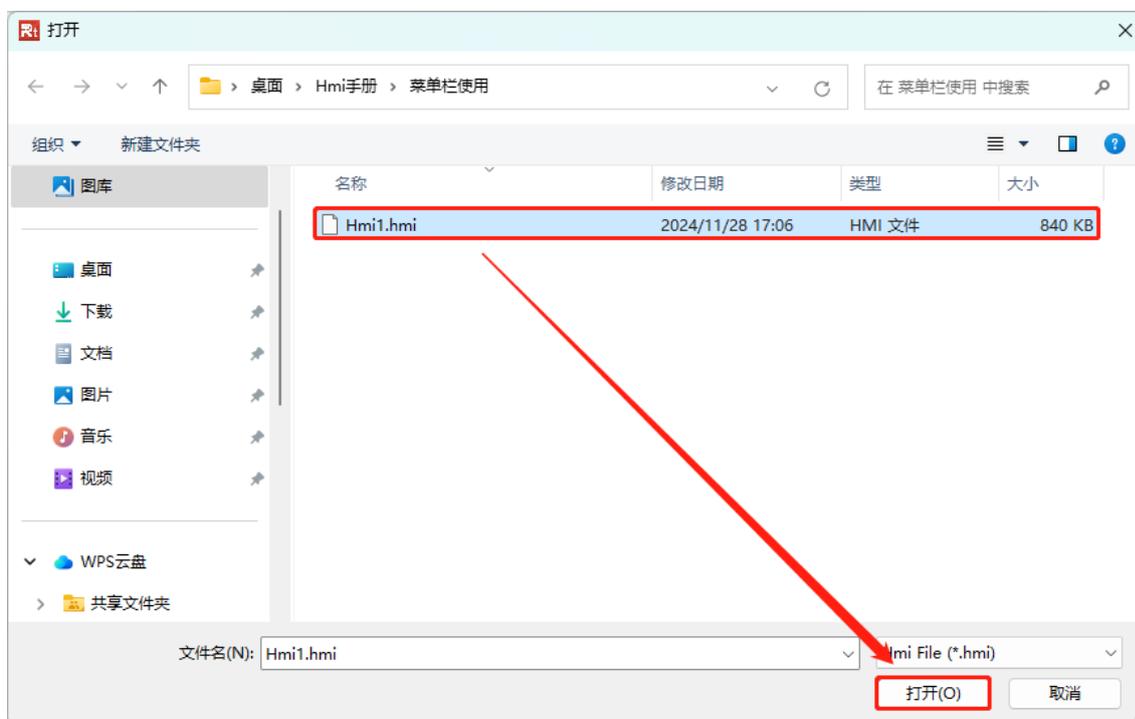
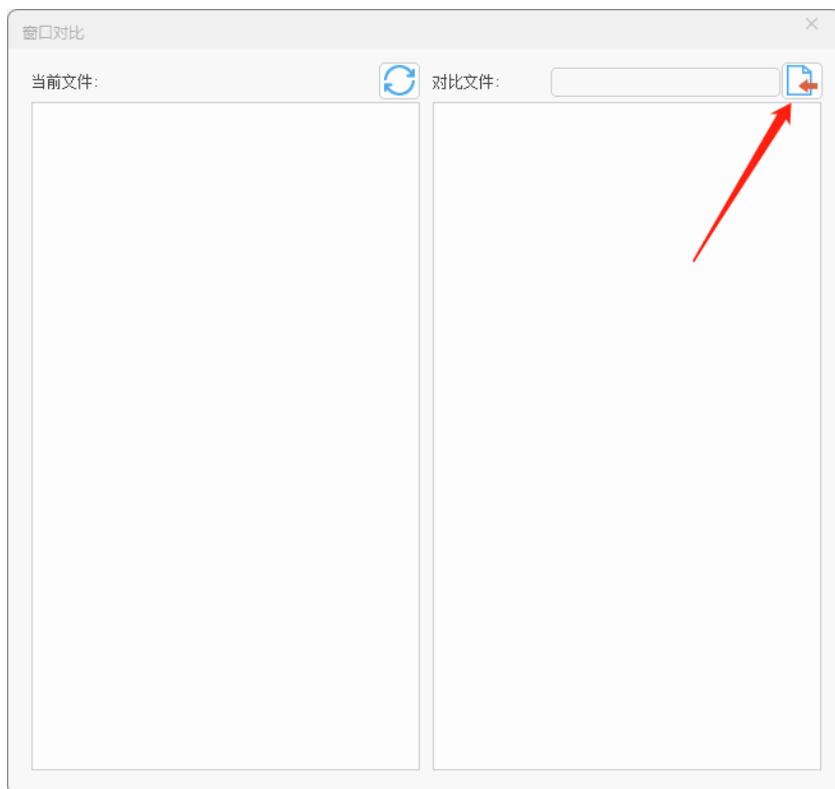
方法二：若不使用图片库样式则在“状态 0”和“状态 1”自定义颜色。（注意：两种样式方式只能二选一生效，使用了图片库样式则会覆盖自定义颜色样式）

窗口预设背景颜色则使用方法二设置“状态 0”自定义颜色，新建窗口后生效。



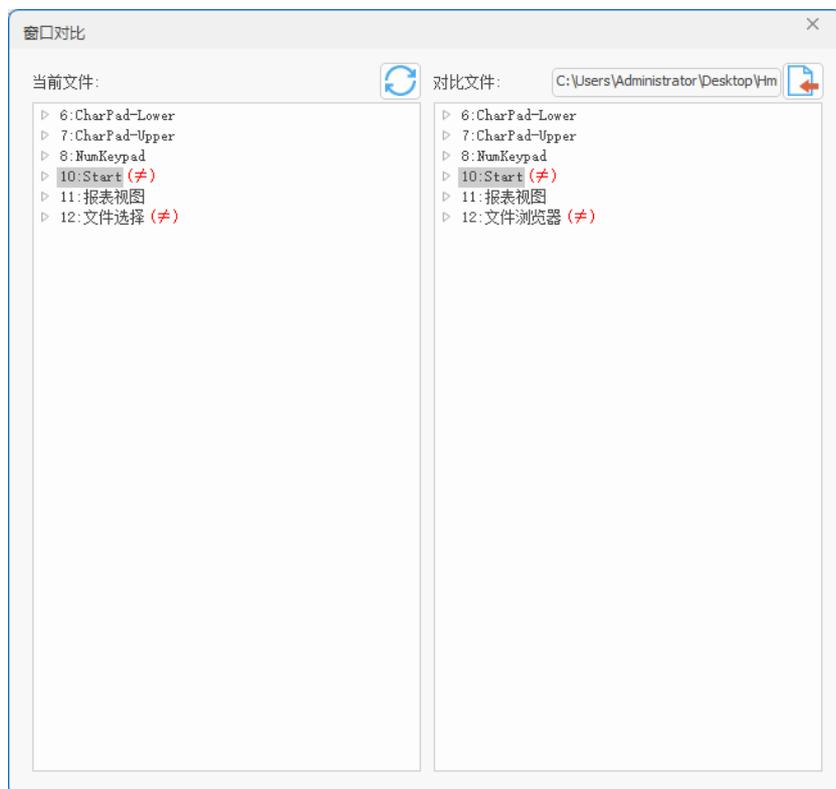
窗口对比： 选择一个 Hmi 文件与当前 Hmi 文件进行各个窗口及窗口元件的对比。具体操作如下：

(1) 点击菜单栏“HMI” – “窗口对比”，点击“”图标，弹出文件选择窗口，选择需要对比的窗口所在的项目文件路径，选择需要的.Hmi 文件。

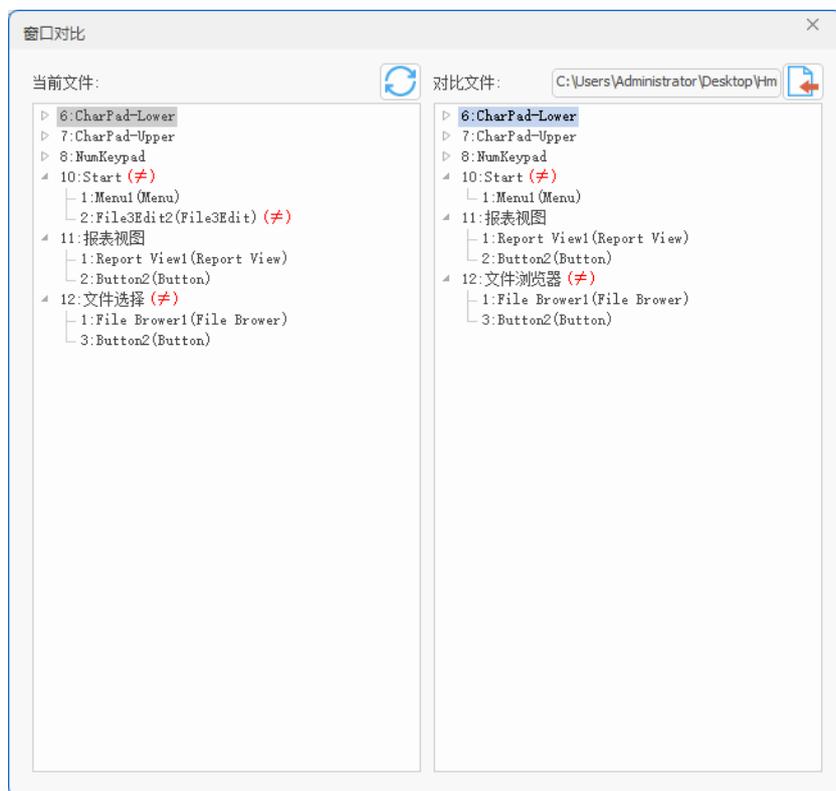


(2) 选定待对比的 HMI 文件后，“窗口对比”界面将呈现两个文件的窗口信息：左侧列出当前文件的各窗口名称，右侧则显示对比文件的相应窗口名称。系统将根据窗口号对左右两边的窗口进行逐一对

比。若窗口名称不一致，或窗口内的元件（包括类型、大小、位置等属性）存在差异，该窗口名称右侧将标记（≠）。

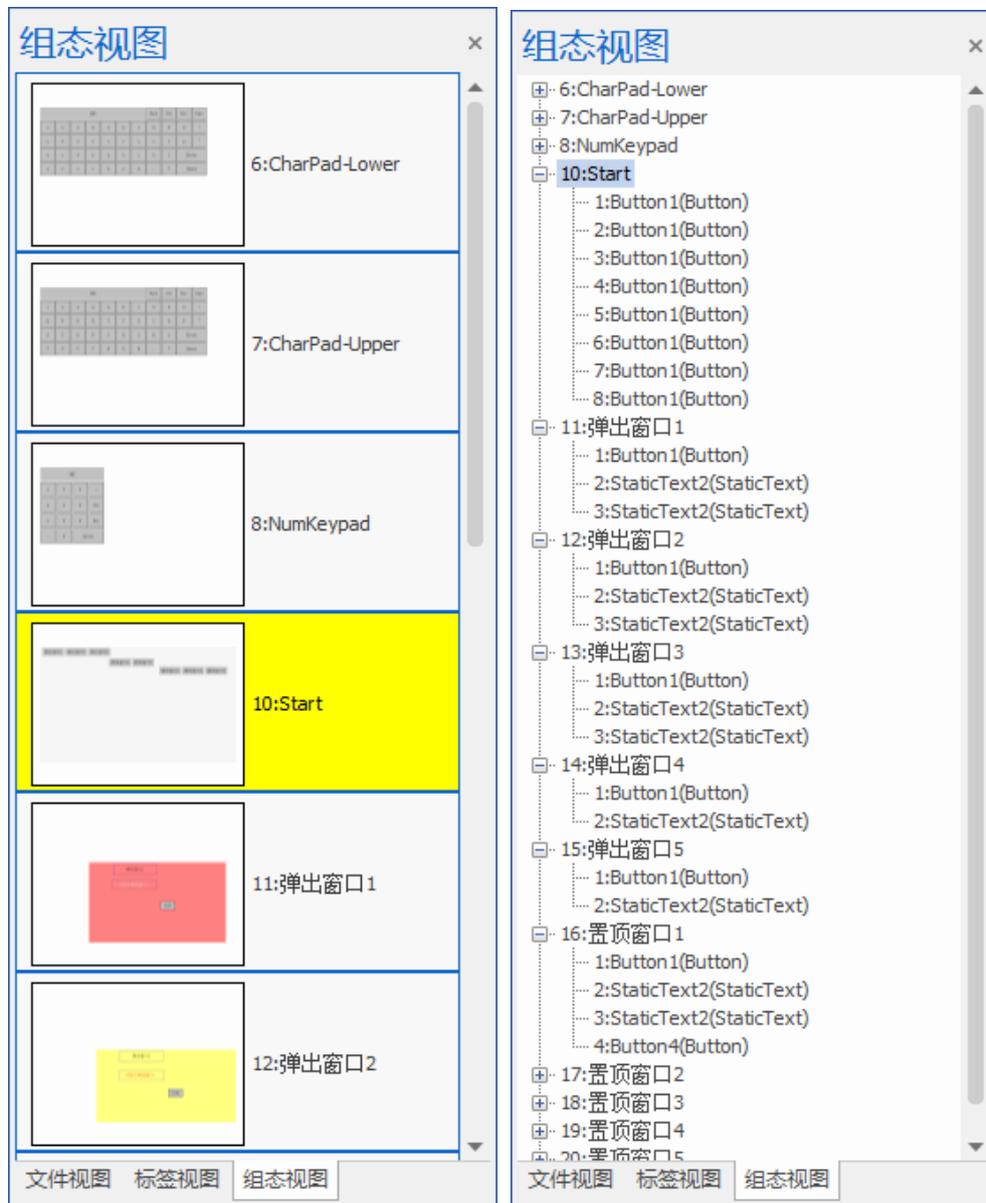


(3) 点击窗口名称前的“▶”图标，可以展开查看该窗口下的具体元件信息。对于元件不一致的情况，其名称右侧同样会标记（≠）。



显示缩略图：在组态视图显示窗口缩略图、窗口号及窗口名称。（黄色背景表示当前已打开的窗口，鼠标单击可切换）参考以下左图。

显示详细信息：在组态视图显示窗口信息（窗口号、窗口名称）和元件信息（已创建的元件编号和元件名称）。参考以下右图。



2.2. 资源



2.2.1. 控件箱

控件箱主要是存放各种组态元件。HMI 开发可在该窗口中添加组态元件。在菜单栏“HMI”→“控件箱”即可打开，一般默认将控件箱隐藏至界面最左侧。更多元件内容可参考【[第四章-组态元件](#)】介绍。

2.2.2. 文本库

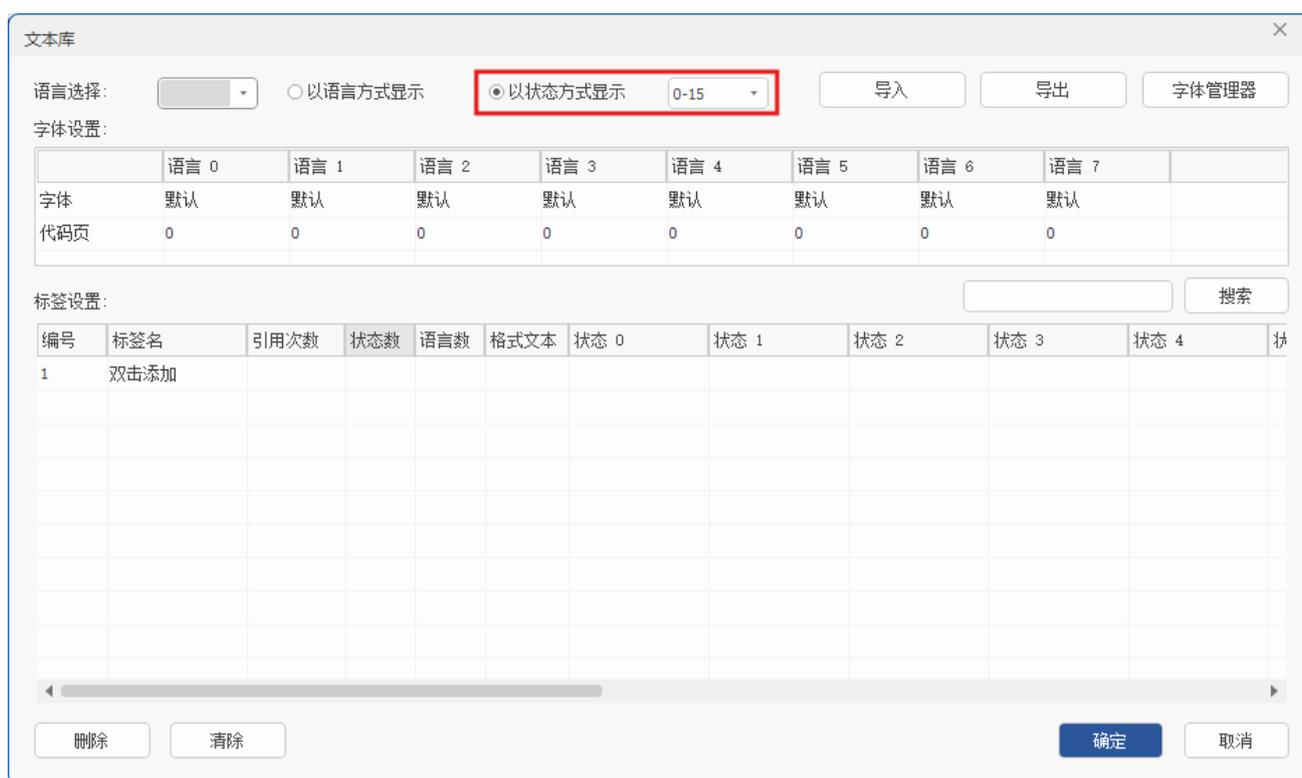
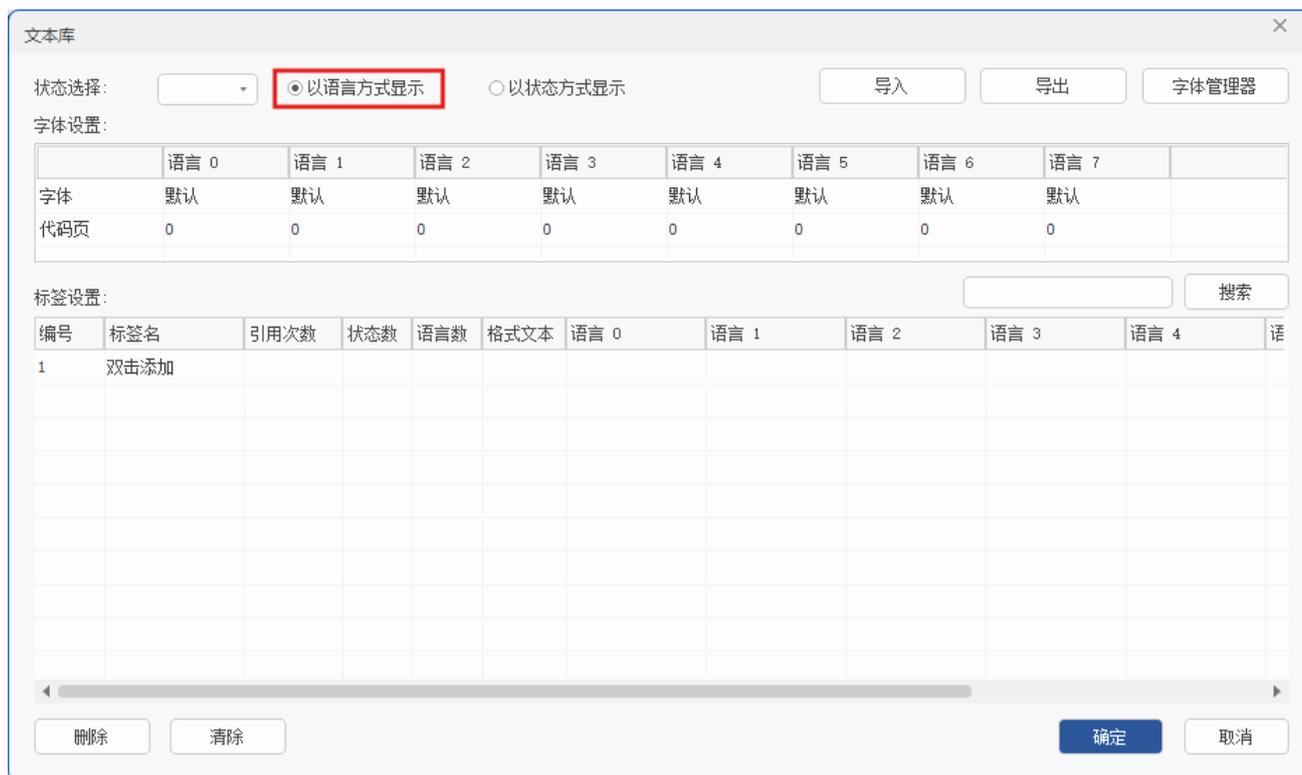
介绍：采用标签形式一次性设置不同状态下的多种语言文本以及每个文本对应的字体并在对应元件调用它。（一个标签最多支持 256 种状态，一种状态最多支持编写 8 种语言文本）

窗口各功能介绍如下表：

功能	描述	
状态选择	选择“以语言方式显示”时显示，用于切换状态，设置不同状态的文本库内容	
语言选择	选择“以状态方式显示”时显示，用于切换状态，设置不同语言的文本库内容	
以语言方式显示	标签设置栏以语言的方式显示，依次设置每种状态下的语言文本库内容，语言数可设置 0 - 7 种	
以状态方式显示	标签设置栏以状态的方式显示，依次设置每种语言下的状态文本库内容，状态数可设置 0 - 255 个	
导入/导出	导入/导出文本库文件	
字体管理器	载入/删除字体文件	
字体设置	字体	选择各语言显示的字体
	代码页	输入各国语言编码（使用其他国家语言时须填写）
标签设置	编号	/
	标签名	文本库名字
	引用次数	引用该文本库的次数（不可设置）
	状态数	设置文本库的状态数
	语言数	设置使用的语言数量
	格式文本	设置语言文本格式
	语言 0-7	选择“以语言方式显示”时显示，写入文本内容
状态 0-15	选择“以状态方式显示”时显示，写入文本内容 状态可选 0-15 / 16-31 / ... / 240-255，以 16 递增	

注意[1]：在中文操作系统下需使用其他国家语言显示时，需要在“代码页”处填入该国语言的编码，使用中国语言时可不填；若在非中文操作系统下，使用中国语言时则须填对应中文语言编码（一般为 936）。

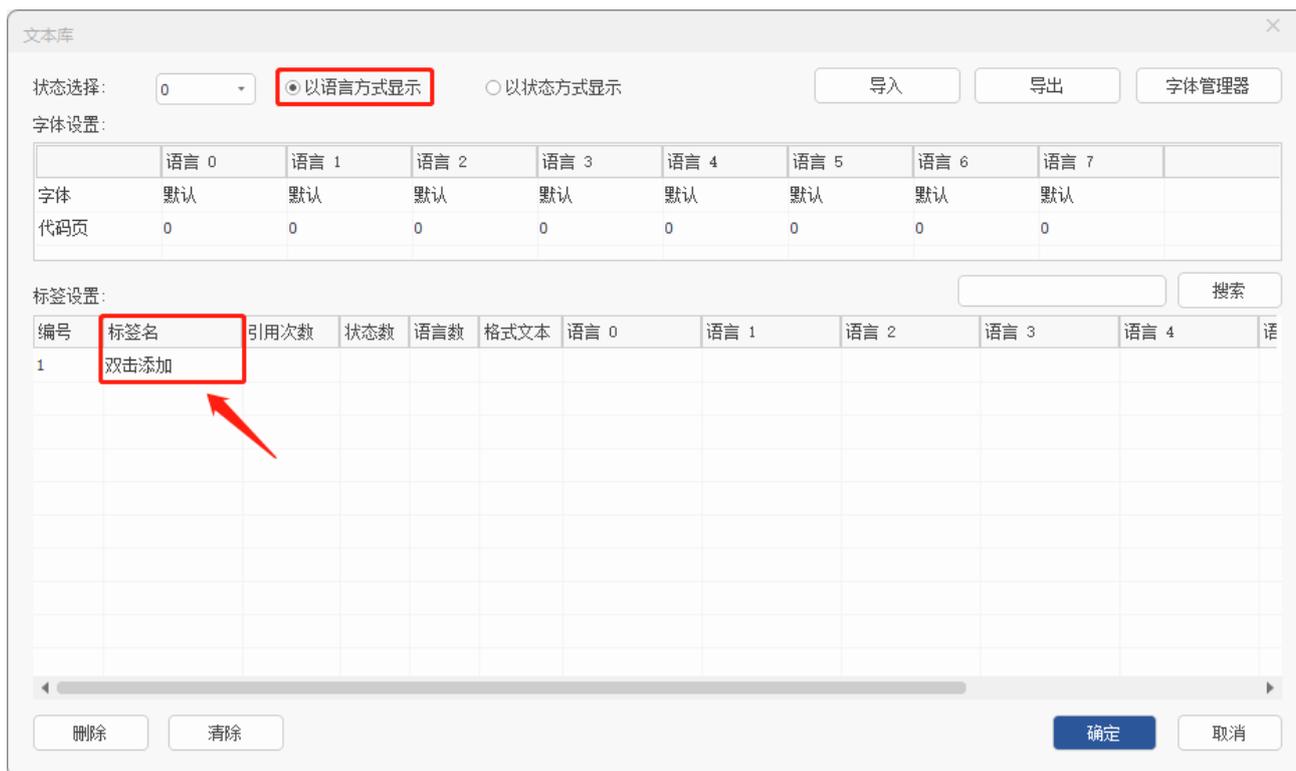
注意[2]：只要字体采用的是 ttf (TrueType Font) 格式，用户即可在“代码页”设置中填入 65001 (UTF-8 编码)，可以有效避免乱码问题的出现。



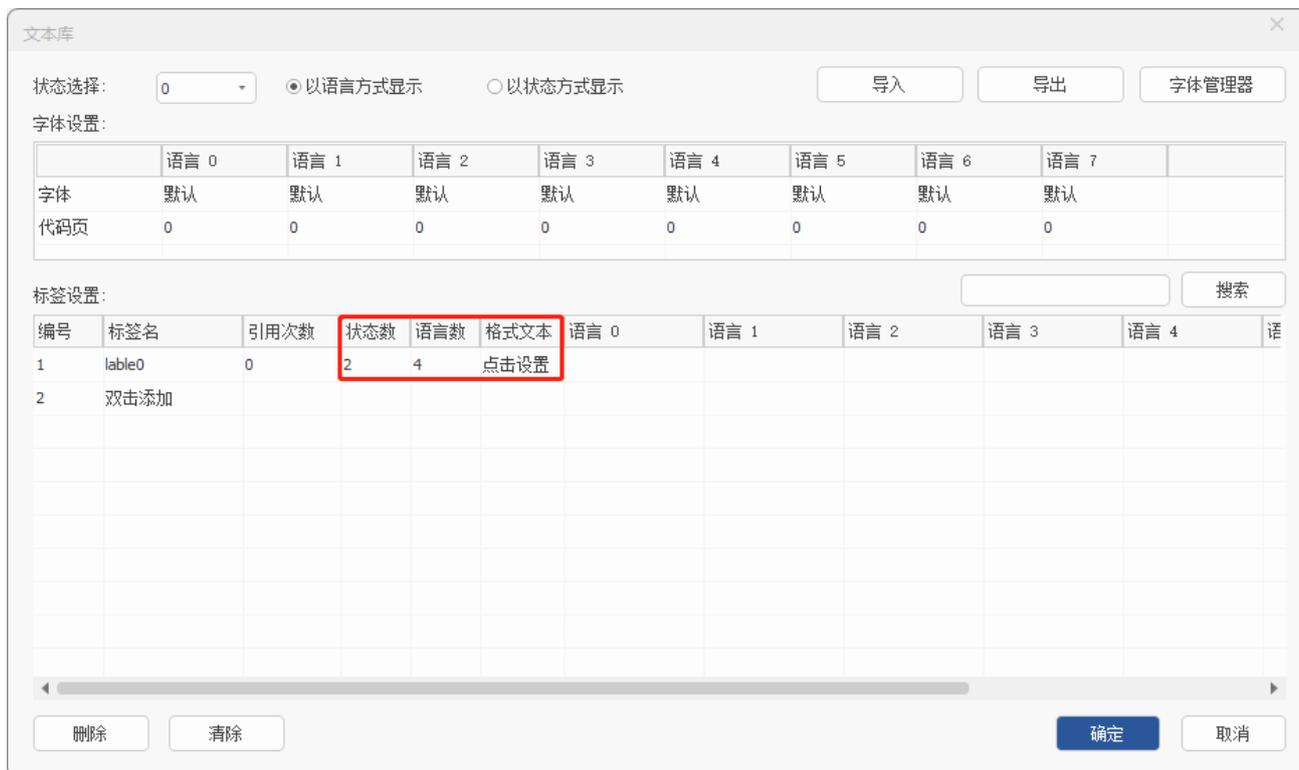
使用方法:

(一) 文本库设置 (“以语言方式显示” 为例)

1. 选择“以语言方式显示”，在[标签设置]板块双击添加“标签名”。（“标签”用于区分元件调用哪个文本库）。

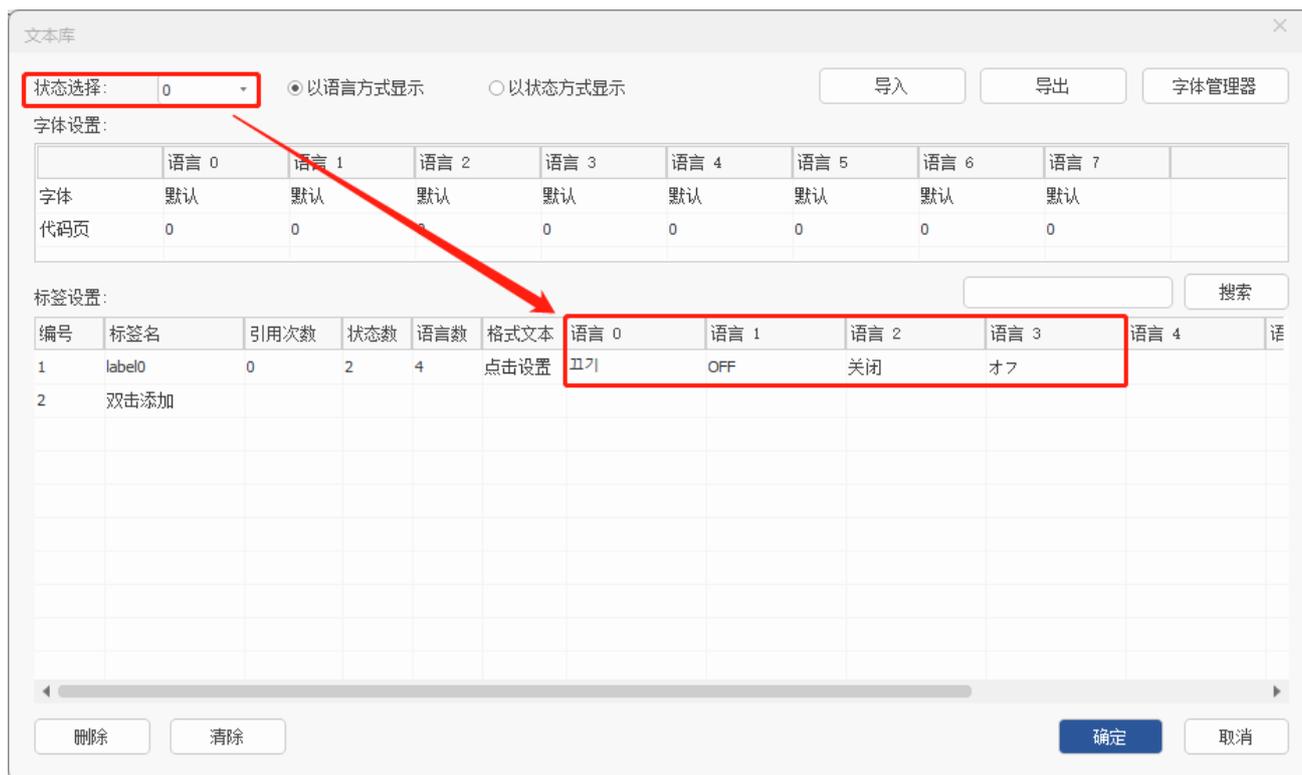


2. 设置所需的状态数和语言数、文本的格式。

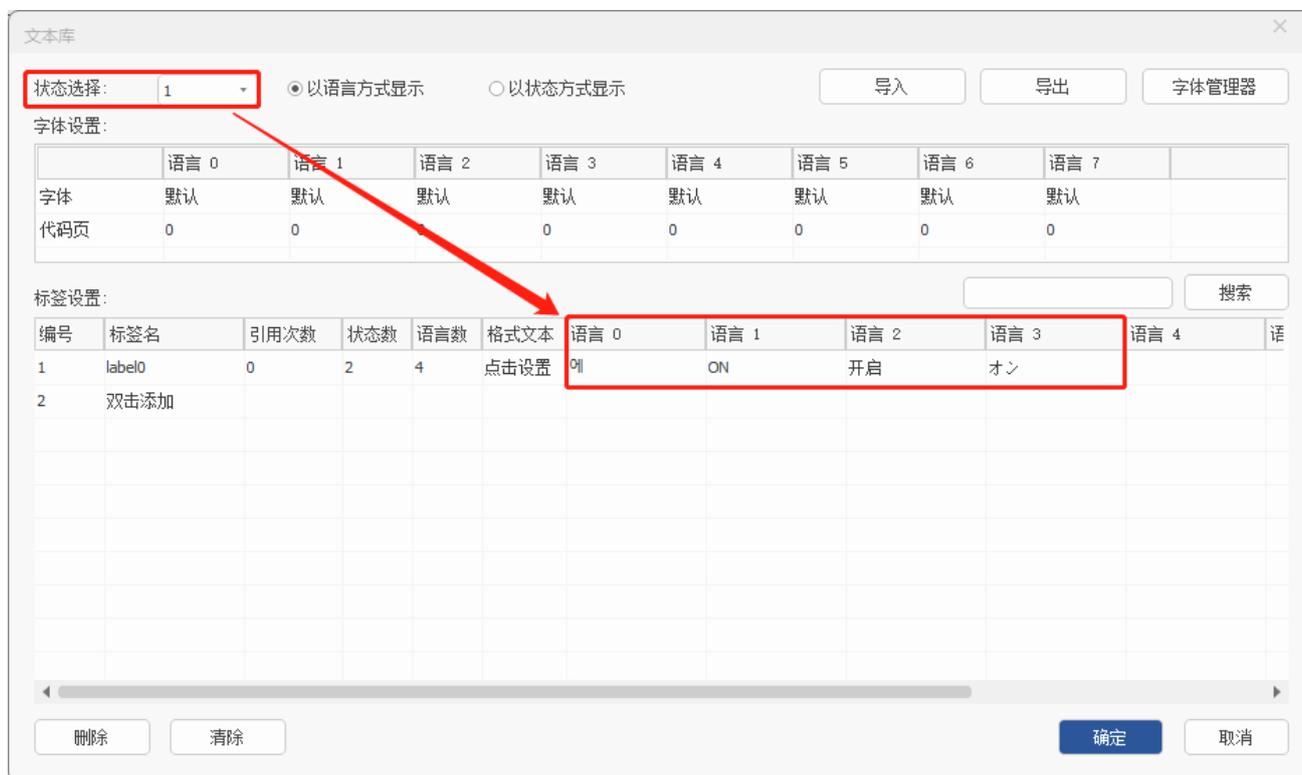


3. 对状态进行选择，在已选择的状态下给对应数量的语言添加需要显示的文本内容。

如下图所示：在状态 0 下添加了对应的语言文本。



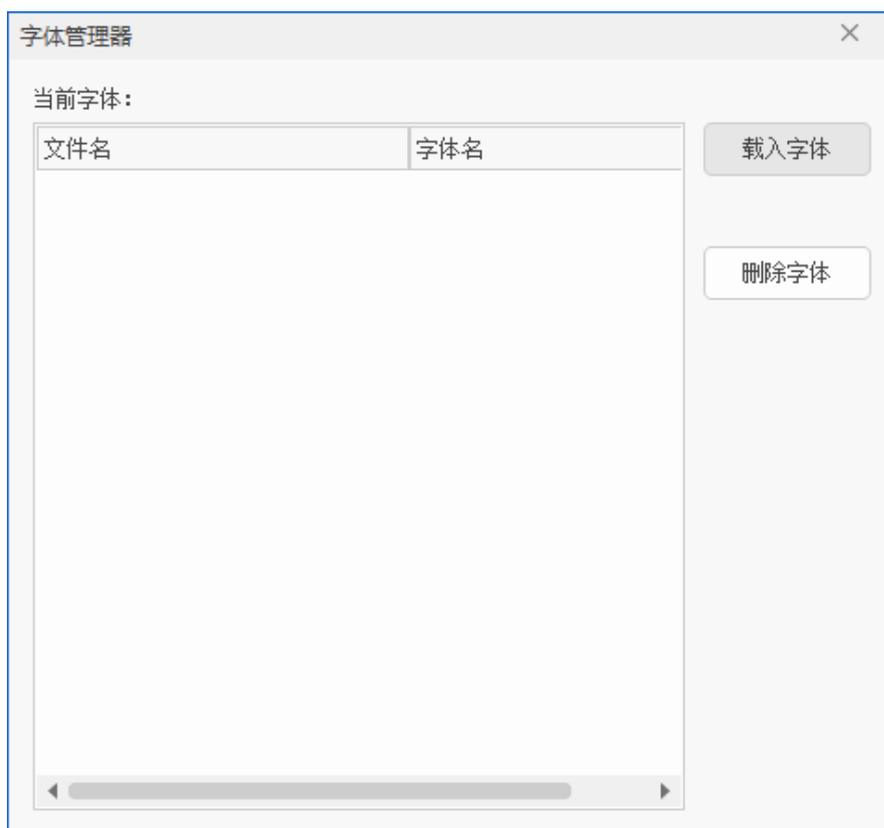
4. 在[状态选择]处切换为状态 1，可在语言处添加处于状态 1 时需显示的文本内容。（状态选择数与设置的状态数有关）



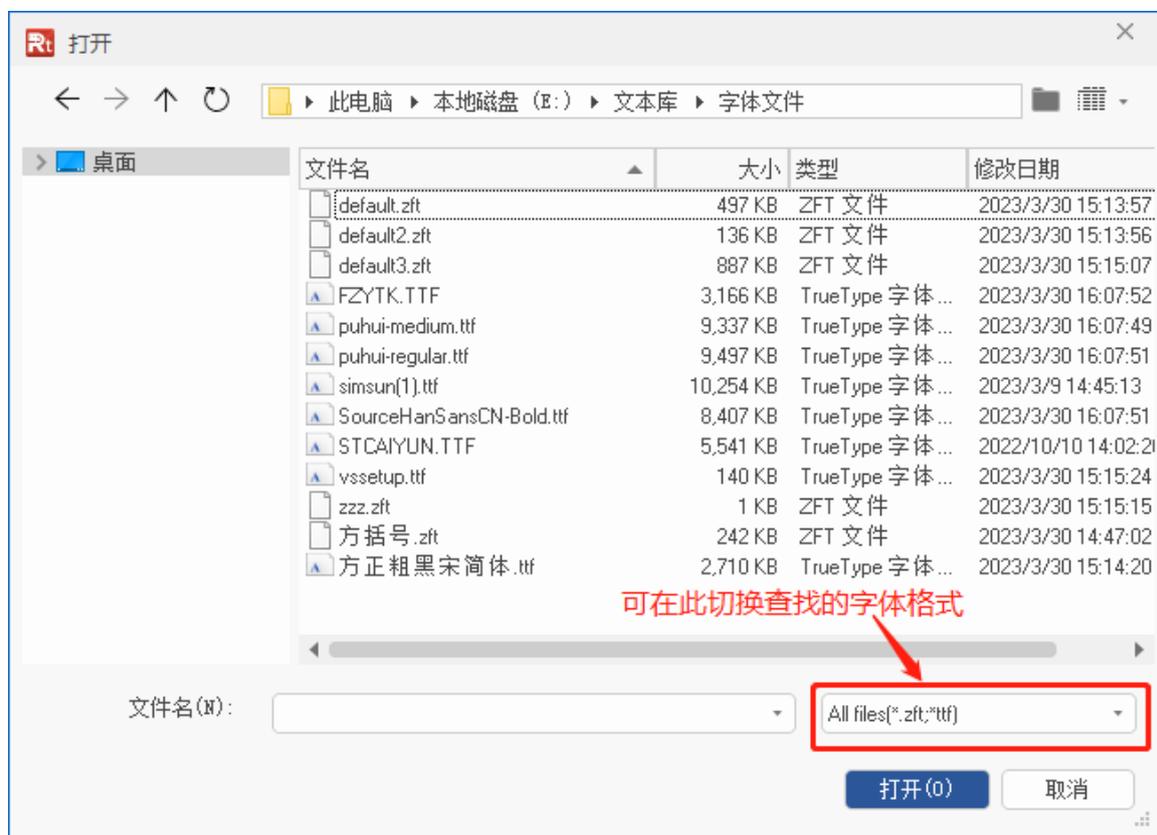
5. 设置好标签及语言文本后，若无需设置字体，点击[确定]即可保存。

若需对字体进行设置，则按以下步骤操作。

6. 首先需要在项目中添加字体文件。在文本库中右上方打开[字体管理器]。



7. 点击[载入字体], 根据存放路径找到对应的字体文件, 点击[打开]即可载入字体 (支持.ttf 和.zft 格式)

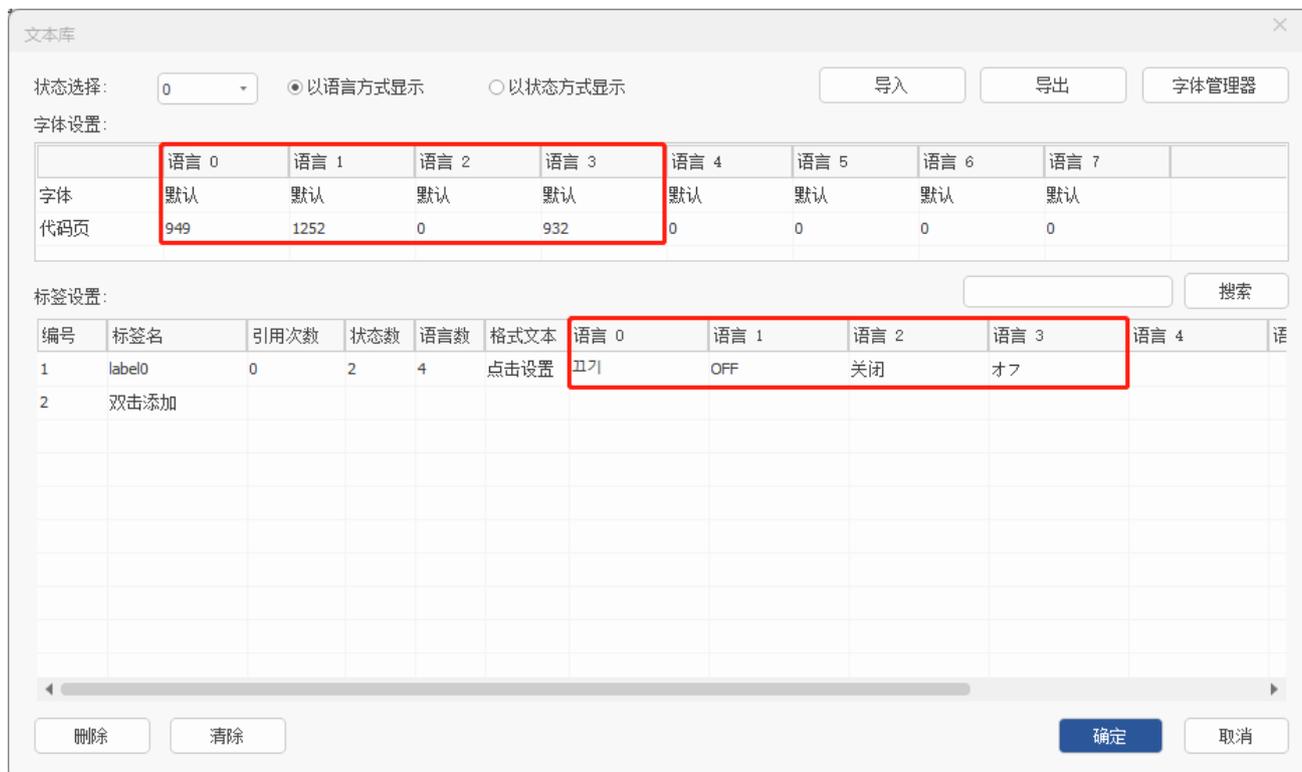


8. 成功载入字体后在列表中显示对应的文件名及字体名。（仅.ttf 格式支持显示字体名）



9. 导入字体后可在[字体设置]板块对每个语言进行字体设置，调用对应文本库后运行即可显示。

注意：由于语言 1，语言 4 分别是韩文字体和日文字体，所以需要在[代码页]处填写对应的语言编码。
(本例为中文操作系统，故中文字体可不填入代码页编码)

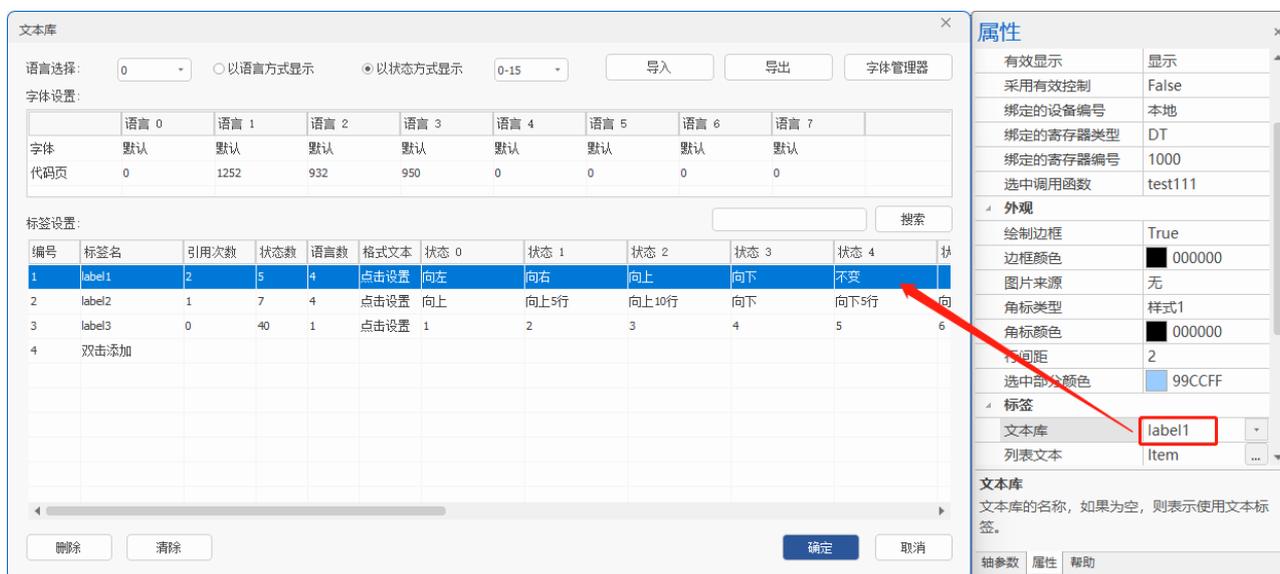


(二) 文本库调用

1. 按上述操作设置好文本库后，可在元件的“属性”中调用文本库。



双击文本库标签名，可打开文本库并跳转到对应文本库位置。



2. 下载运行后可通过指令：Hmi_LANG= ilang（语言号）进行切换。

通过修改寄存器的数值可改变语言的状态。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 1 时，显示的是语言 0 的状态 1。

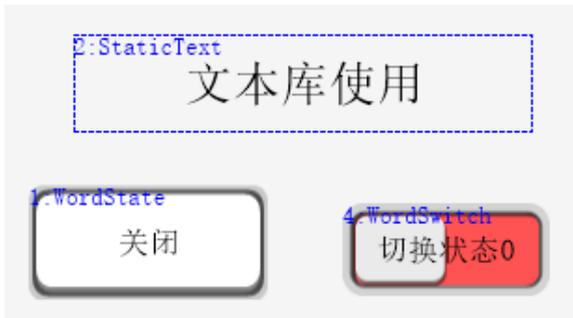
下方用例程演示。

设计思路:

- 用[多状态切换开关]元件对[多状态显示]元件实现状态切换。
- [多状态显示]元件通过调用子函数运行 Hmi_LANG 指令进行语言切换。

(1) 组态设计

在组态窗口中添加以下元件：静态文本（仅用于标题显示）、多状态显示、多状态切换开关。



(2) basic 编程

```

GLOBAL dim runsub_sign = 0  '变量标志

GLOBAL SUB sub_switch()  '函数调用切换
    IF runsub_sign = 0 THEN
        langue0
    ELSEIF runsub_sign = 1 THEN
        langue1
    ELSE
        langue2
    ENDIF

    runsub_sign=runsub_sign + 1

    IF runsub_sign > 2 THEN
        runsub_sign = 0
    ENDIF
END SUB
    
```

```

GLOBAL SUB langue0() '使用语言 0
    HMI_LANG=0
END SUB

GLOBAL SUB langue1() '使用语言 1
    HMI_LANG=1
END SUB

GLOBAL SUB langue2() '使用语言 2
    HMI_LANG=2
END SUB
    
```

(3) 调用过程

【多状态显示】元件

- 在“属性”窗口中的“文本库”选择需要使用的文本库内容（例子的文本库名称为 label0）；
- 在“属性”窗口中的“动作”调用子函数 sub_switch()；

【多状态切换开关】元件

- 在“属性”窗口中的“状态数量”设置为 2，并对应状态添加文本；
- 在“属性”窗口中的“动作”选择[循环]，“动作数据”设置为 1；

(4) 运行效果

- 当前两个元件均处于状态 0，此时[多状态显示]元件显示文本库中状态 0 时的语言 0 文本（即文本显示为：关闭）；



- 点击[关闭]元件，可实现语言切换。如下左图，切换至语言 1；右图则切换至语言 2。



- 点击[切换状态]元件，可实现状态切换。如下图两个元件切换至状态 1。[多状态显示]元件显示文本库中状态 1 时的语言 0 文本（即文本显示为：开启）；



- 点击[开启]元件，可实现语言切换。如下左图，切换至语言 1；右图则切换至语言 2。



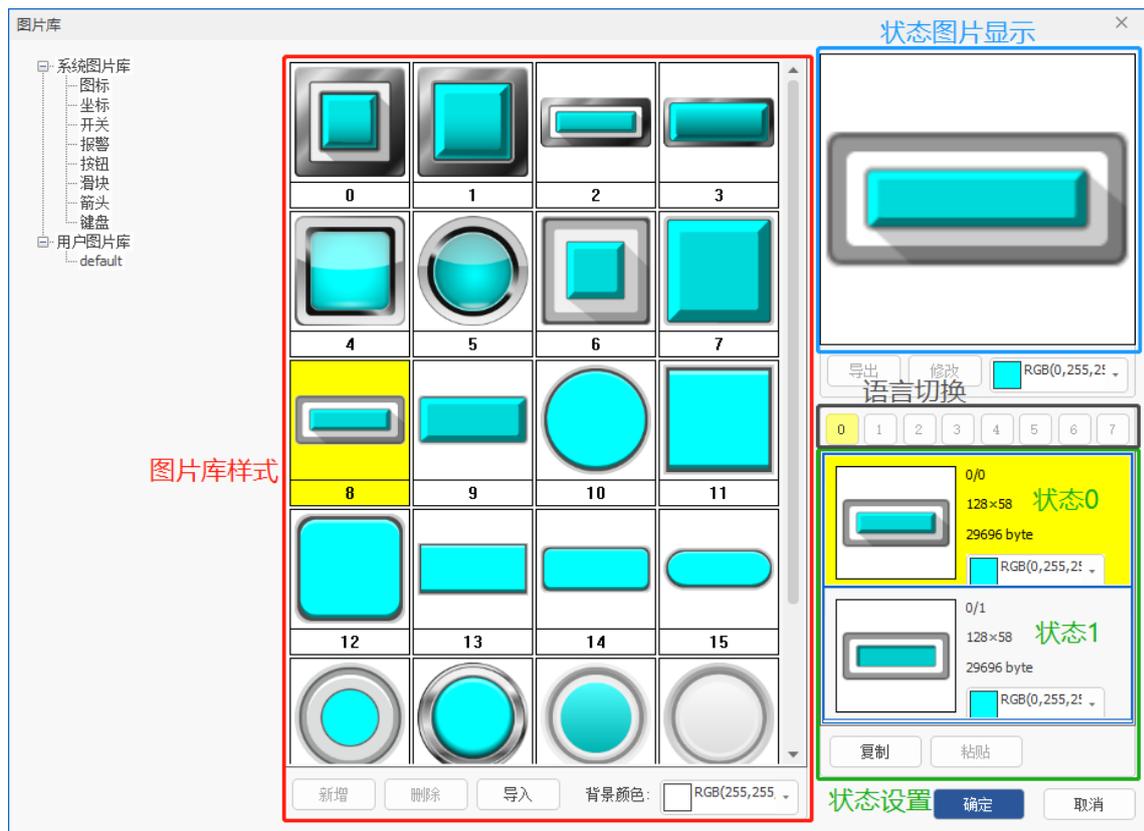
2.2.3. 图片库

介绍： 汇总存放 Hmi 元件样式图片或自定义图片的库。包括系统图片库和用户图片库。该库中支持对图片样式颜色，不同状态及不同语言对应显示内容/颜色等进行修改。

系统图片库有多个类别，均为系统默认样式，不支持删除或增加以及修改图片属性。该库为用户提供了丰富的元件样式选择，用户可直接在该处选择并应用。

用户图片库为用户创建自定义图片库，支持添加外部图片到该库中使用。

注：图片库最大数量为 512！

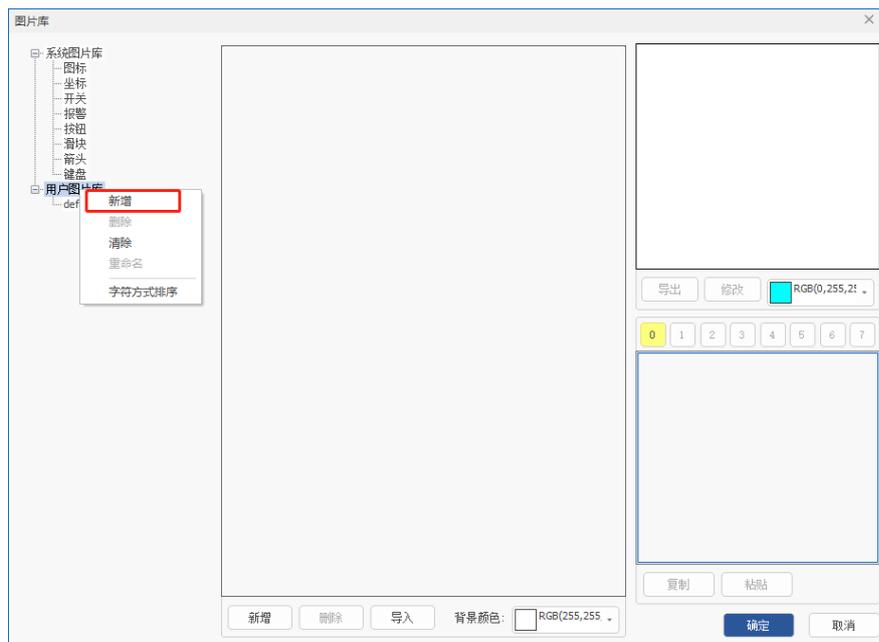


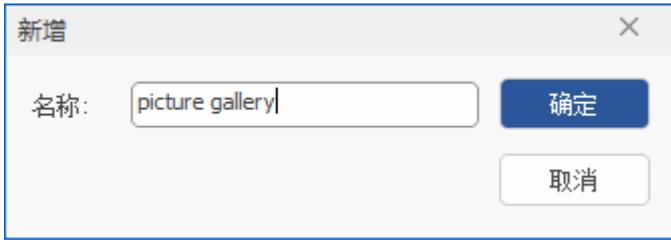
使用方法（以开关控制台灯亮灭为例）：

【新建图片库】

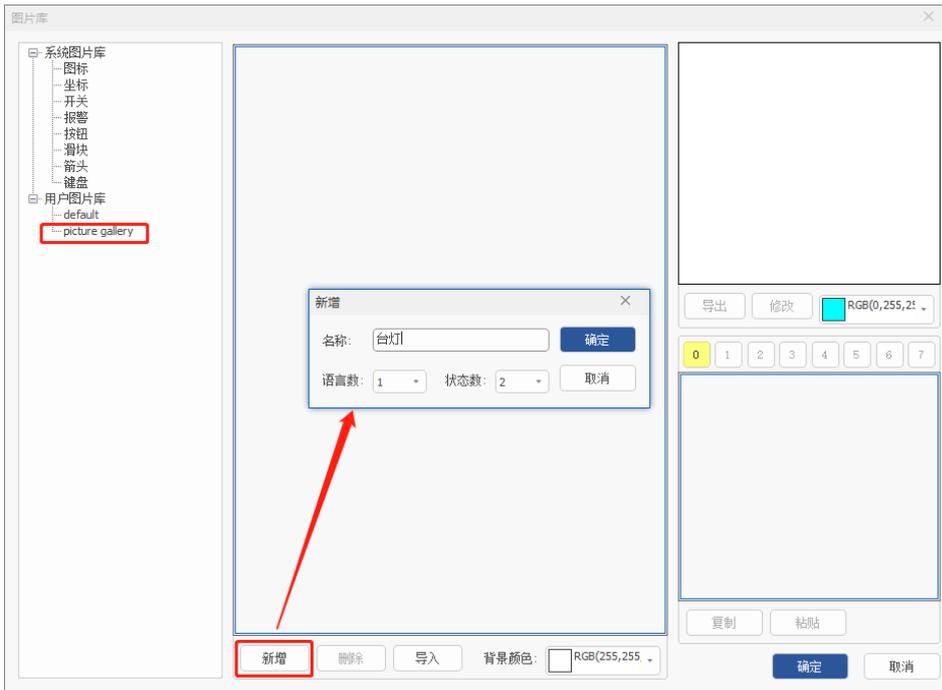
(一) 台灯图片导入

1. 鼠标右键单击[用户图片库]，选择“新增”。弹出窗口输入新建图片库名称点击确定即可。

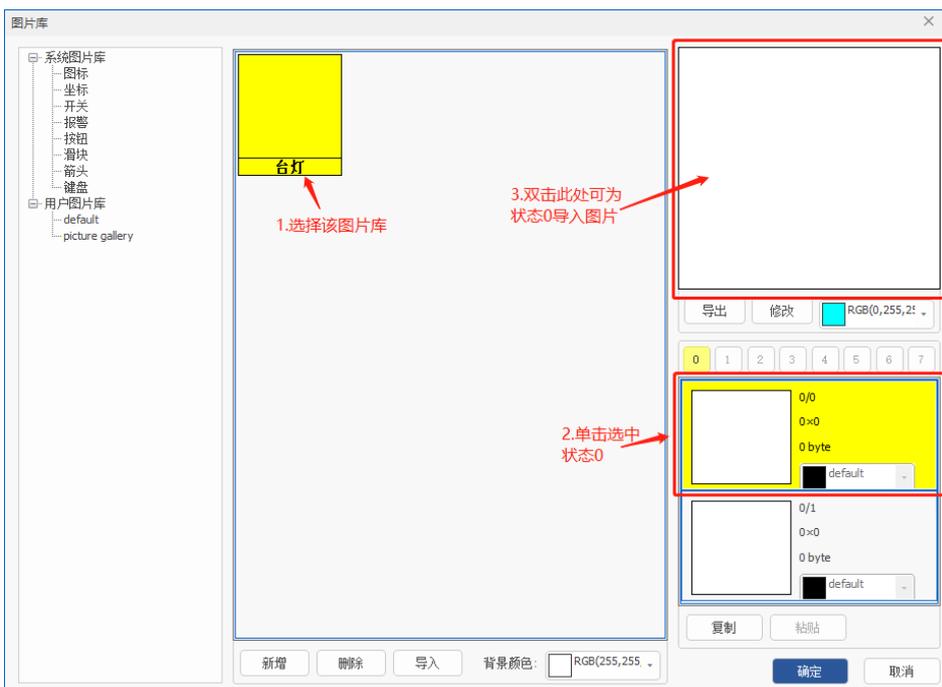




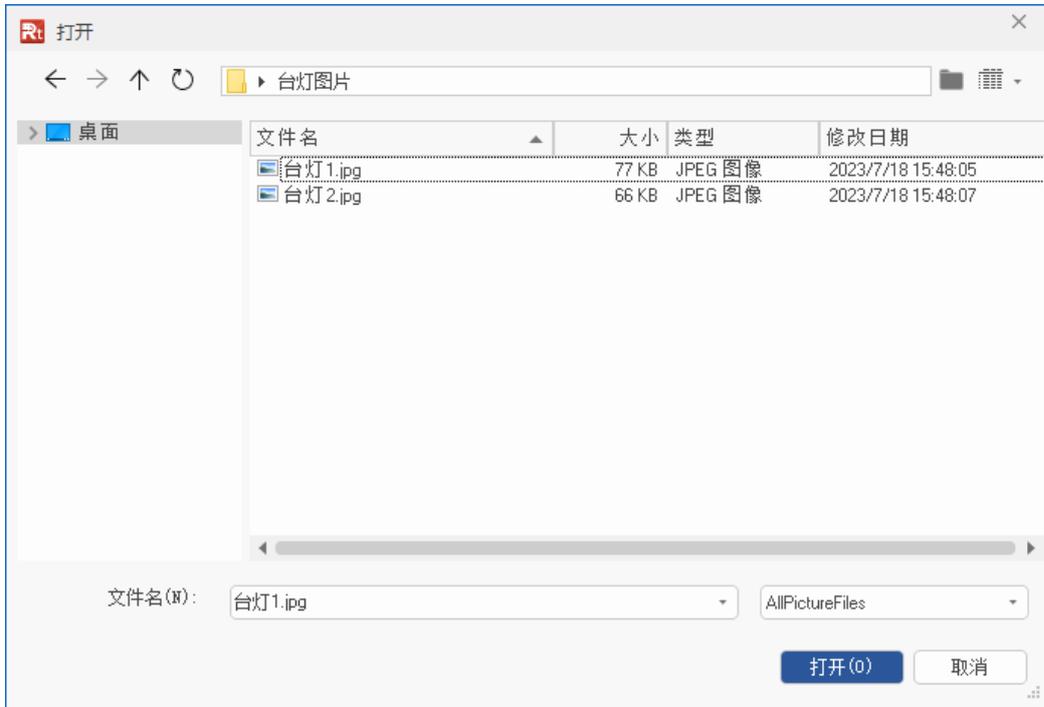
2. 双击打开已新建的[picture gallery]，点击“新增”；在弹窗中填入名称，设置好对应语言数及状态数即可。



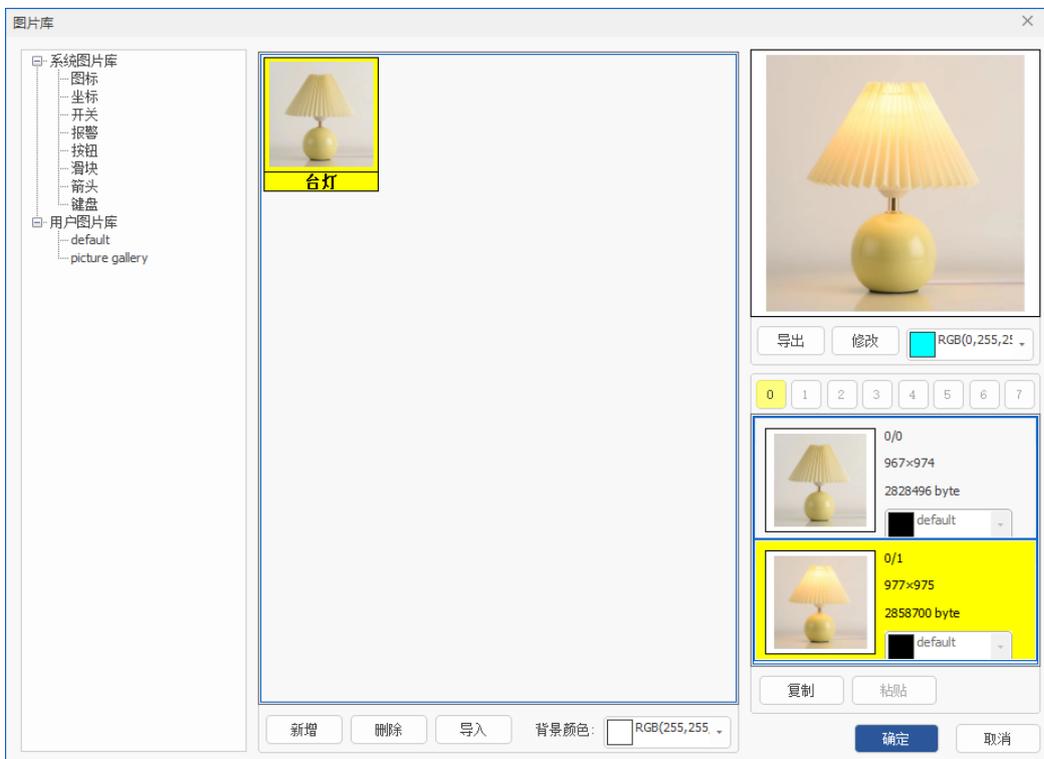
3. 新增完成后，单击选中该图片库（底色为黄色即为选中状态），在右侧选择对应的状态，给不同的状态添加对应的图片。



4. 选择状态 0，鼠标双击上方空白处（或点击“修改”），选择目标图片存放路径，单击选择图片，点击“打开”即可。

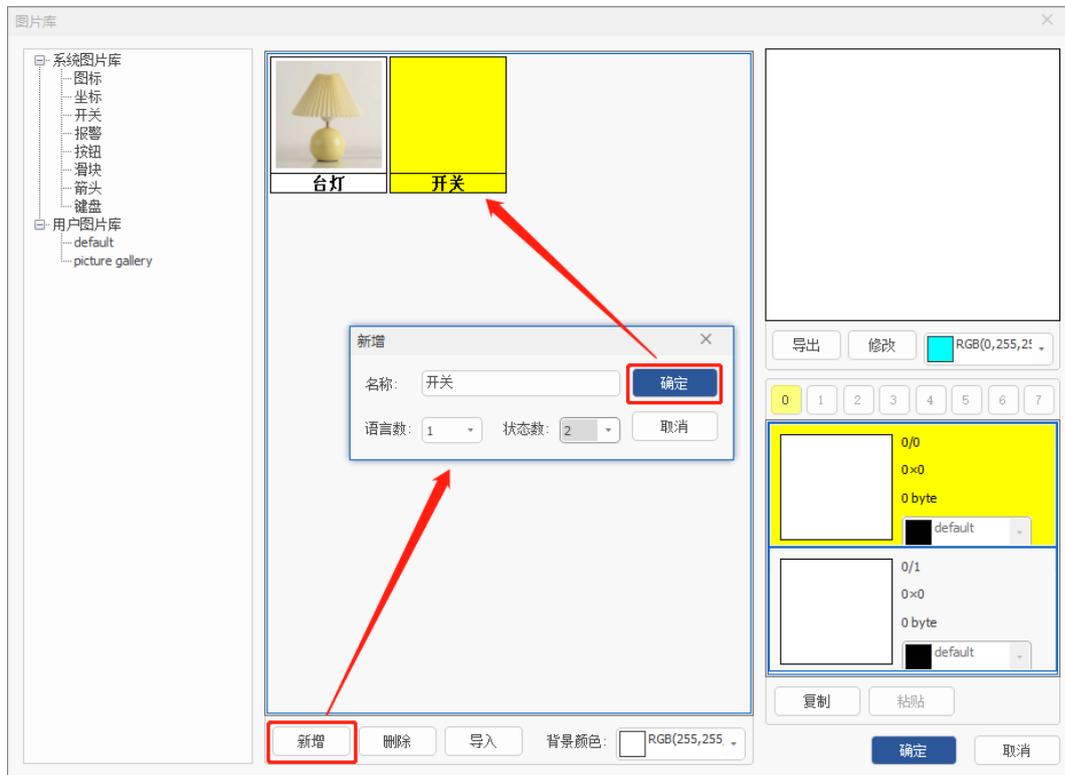


5. 状态 0 添加图片后，按上述方法给状态 1 添加对应图片，单击选择状态 1 后双击上方空白处（或点击“修改”）添加图片。添加完全部图片后点击“确定”即可保存。



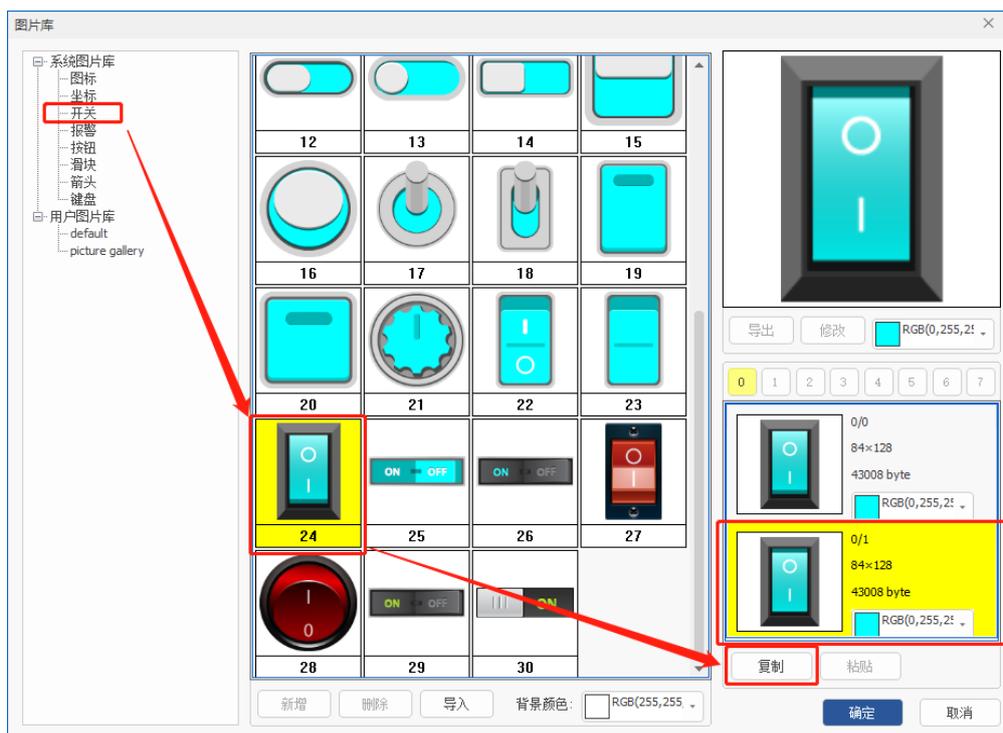
(二) 开关按钮样式复制

1. 在新建的[picture gallery]图片库中，再新增[开关]的图片库，设置好对应的语言数和状态数后点击“确定”即可完成创建。

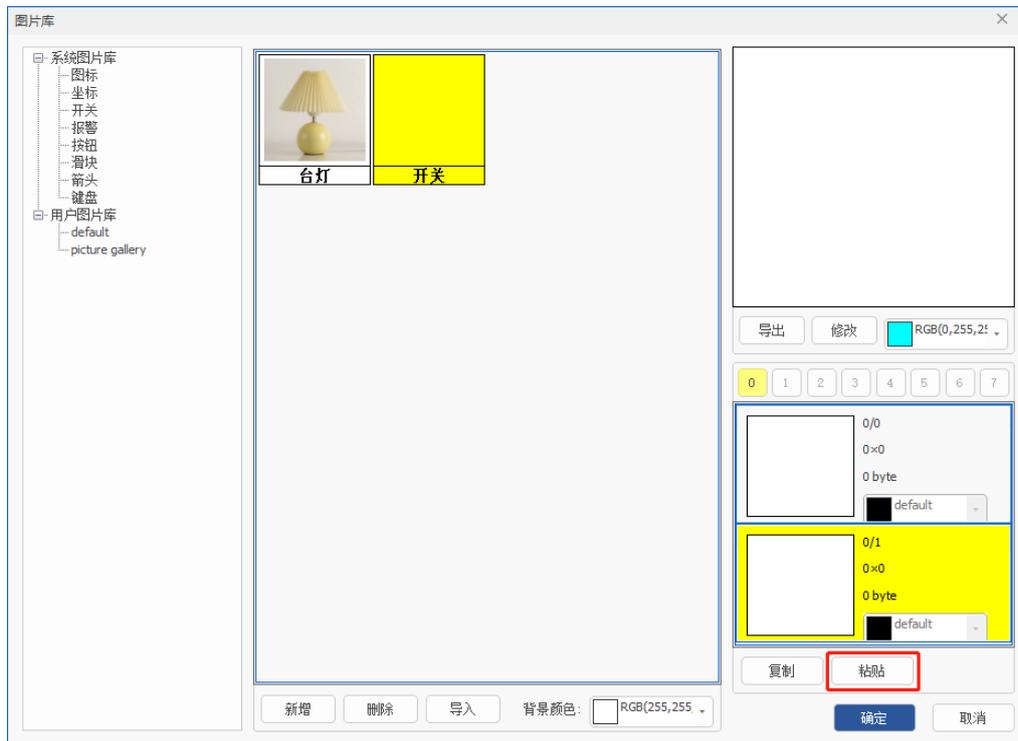


2. 在[系统图片库]的分类中选择合适的样式，选择要复制的状态，点击“复制”即可将该状态下的样式复制。

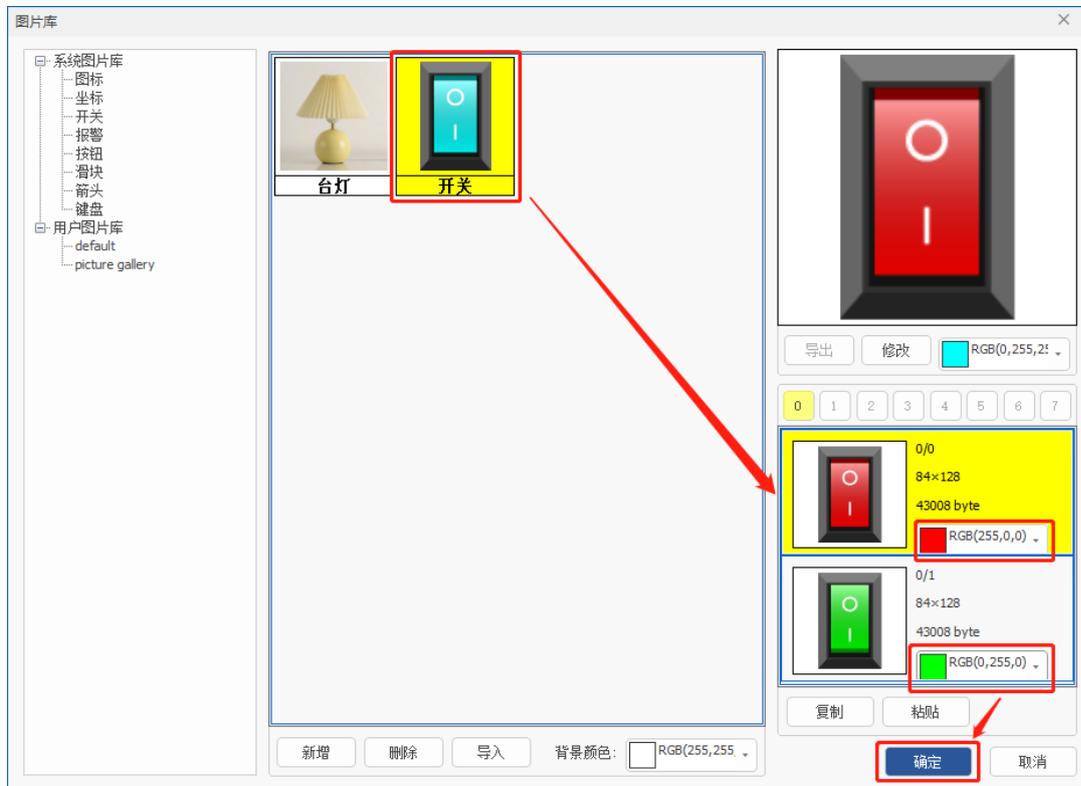
注：若不使用[系统图片库]已有样式，则可按上述新增方法自行添加图片。



3. 返回新建的[picture gallery]图片库，选择[开关]的图片库，单击选择状态 1，单击粘贴，即可将已复制的样式应用到该处。



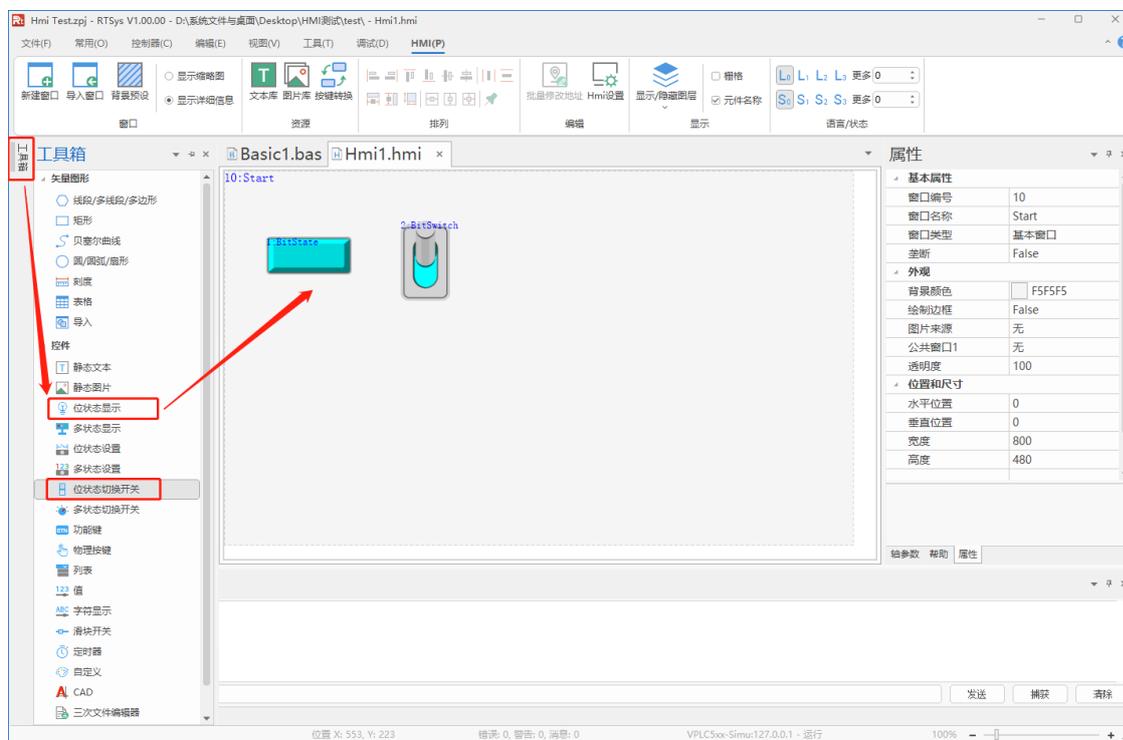
4. 添加完各状态下的图片后，可根据需求修改图片显示颜色，点击“确定”即可。至此图片库创建完成。



【图片库应用】

(一) 元件直接应用

1. 新建 HMI 文件后，给 HMI 文件添加自动运行任务号，打开 HMI 文件。设置 Hmi 系统属性后，在 HMI 窗口中添加组态元件。
2. 从控件箱中选择组态元件“位状态显示”和“位状态切换开关”添加到窗口 10 中。

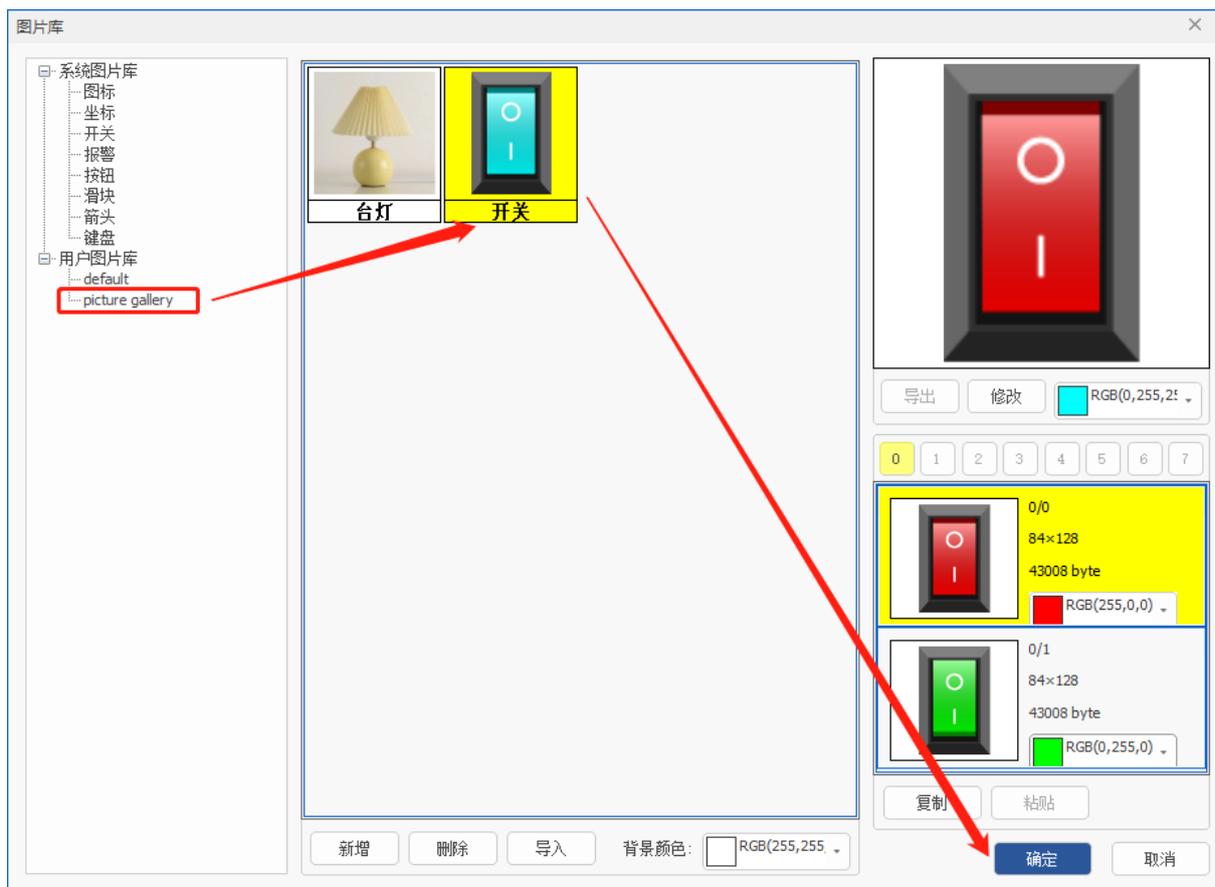


3. 单击选中元件 2“位状态切换开关”，软件界面右侧弹出该元件“属性”窗口。在“外观”→“图片来源”选择“背景图片库”，在下一项的“背景图片库”中单击“...”即可打开图片库。

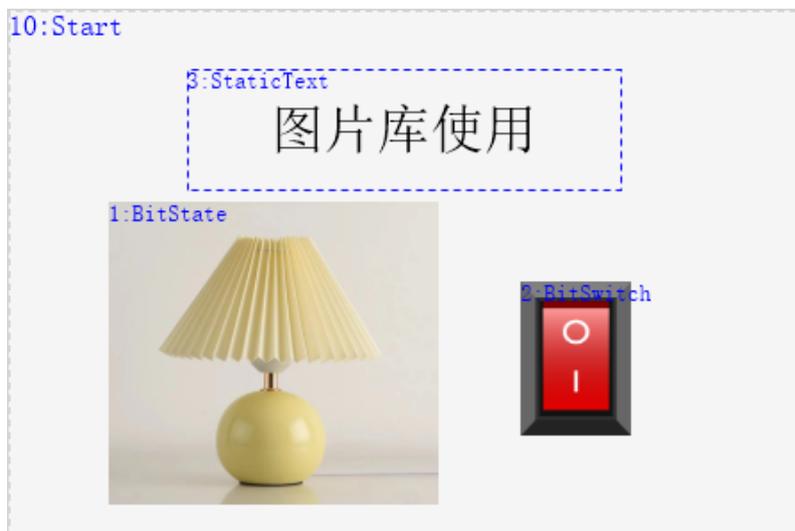


4. 打开图片库后，在图片库中找到目标样式选中后并点击“确定”即可应用到该元件，根据需求对元件

大小进行调整。（元件 1 的样式应用与上述方法一致，绑定寄存器设置为 M2）

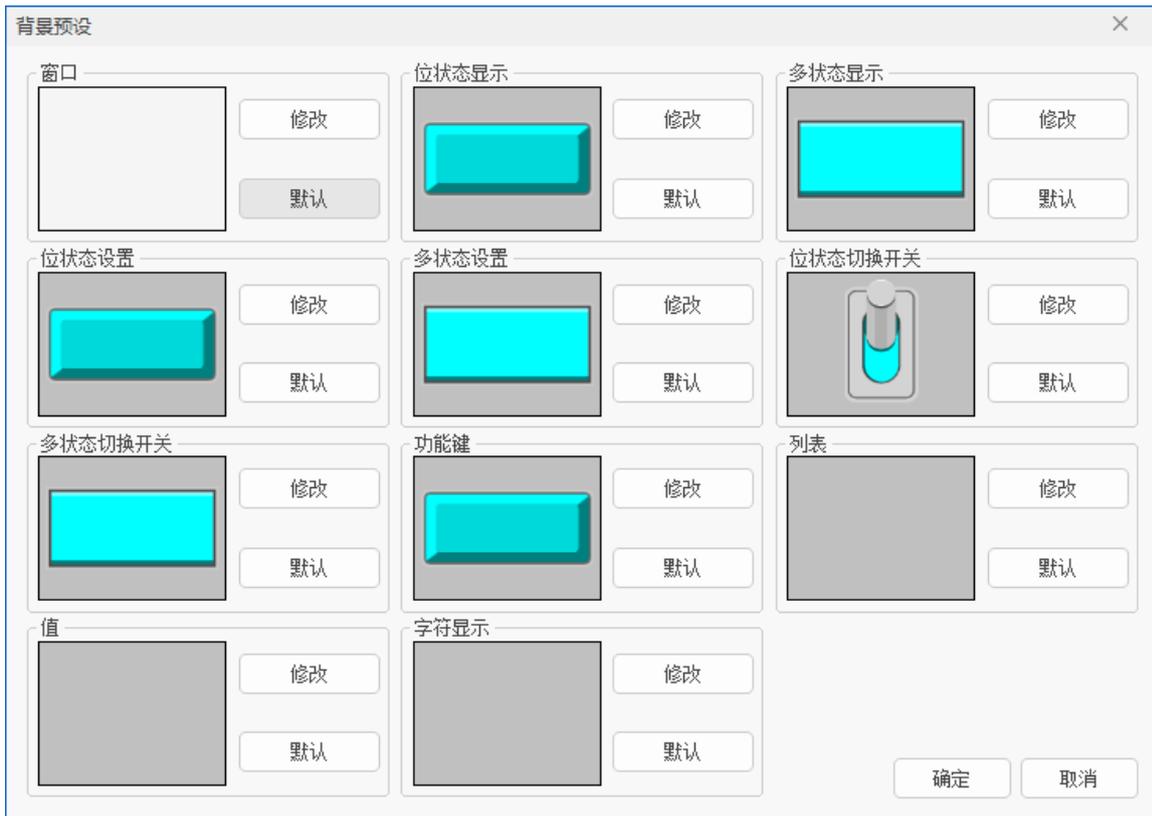


图片库样式成功应用到元件后显示如下图：

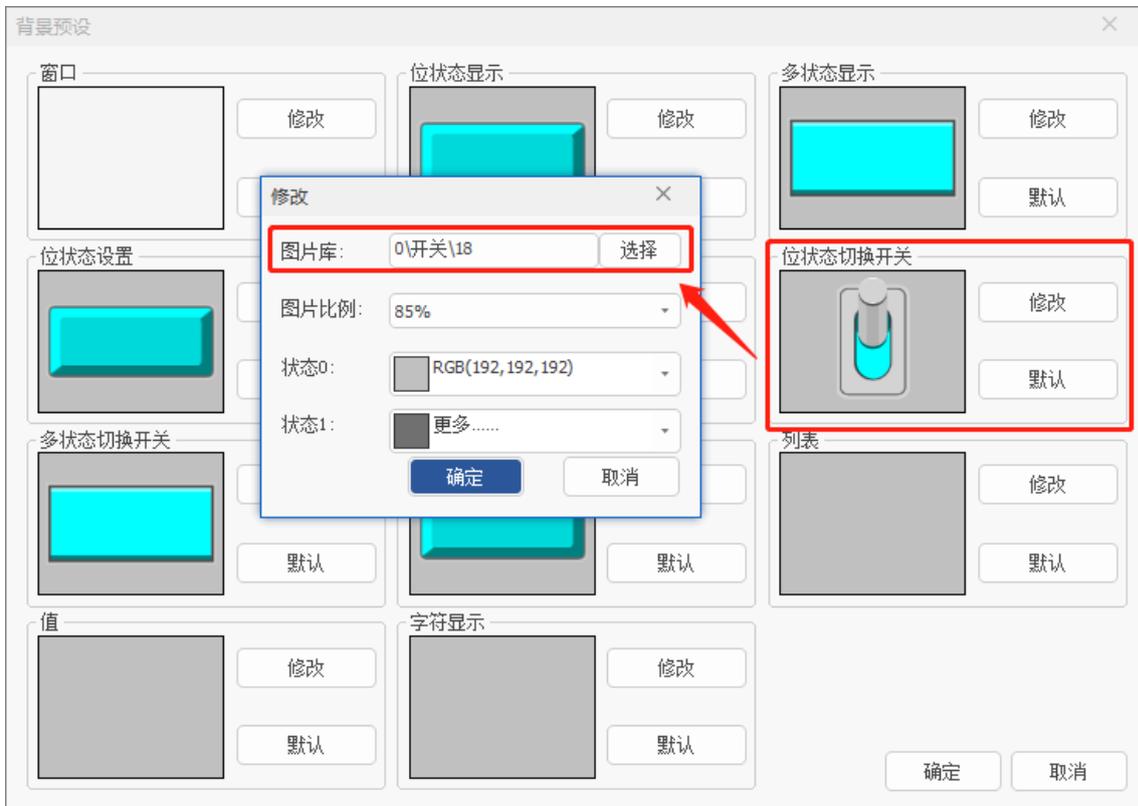


(二) 背景预设应用至元件

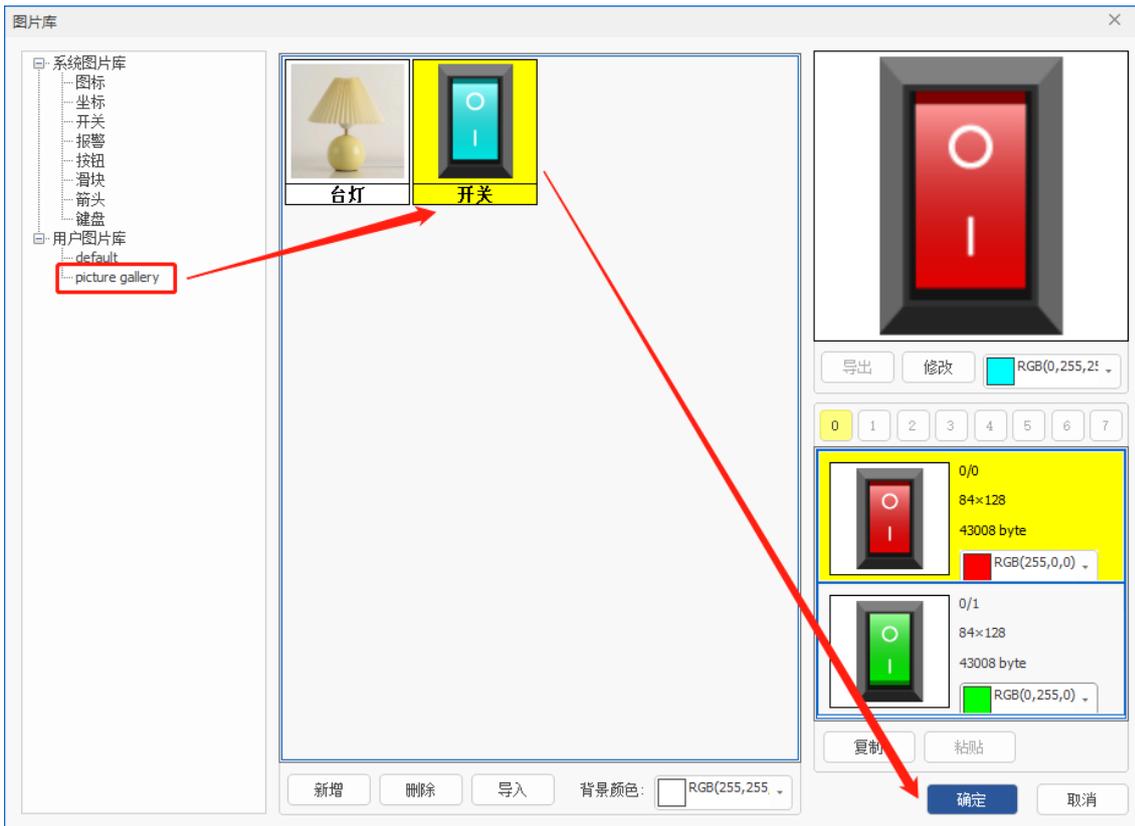
1. 打开 HMI 文件，通过菜单栏“HMI”→“背景预设”打开如下窗口，可直接在该处修改元件的样式。



2. 选择需要修改样式的元件（以位状态切换开关为例），点击“修改”弹出如下窗口。



3. 点击图片库栏后的“选择”即可打开图片库窗口，选择需要应用的样式，点击“确定”即可。



4. 完成图片库样式修改之后，返回以下窗口，可在该窗口调整图片显示比例，以及状态背景色等，完成设置后点击“确定”即可。

注：状态背景色仅供预览参考，实际应用时不生效。



【调用过程】

(一) 位状态显示元件

- 单击该元件在“属性”窗口选择绑定寄存器为 M0。

(二) 位状态切换开关

- 单击该元件在“属性”窗口选择绑定寄存器为 M0。
- 在“属性”窗口的动作选择“状态反转”。



【运行效果】

1. 将程序下载到控制器/仿真器后，打开 xplc screen 即可显示组态窗口。
2. 当两个元件处于状态 0 时，即台灯与开关均处于关闭状态，如下左图所示。当按下开关切换为状态 1 时，台灯同时切换为状态 1，即为开灯状态。如下右图所示。



2.2.4. 按键转换

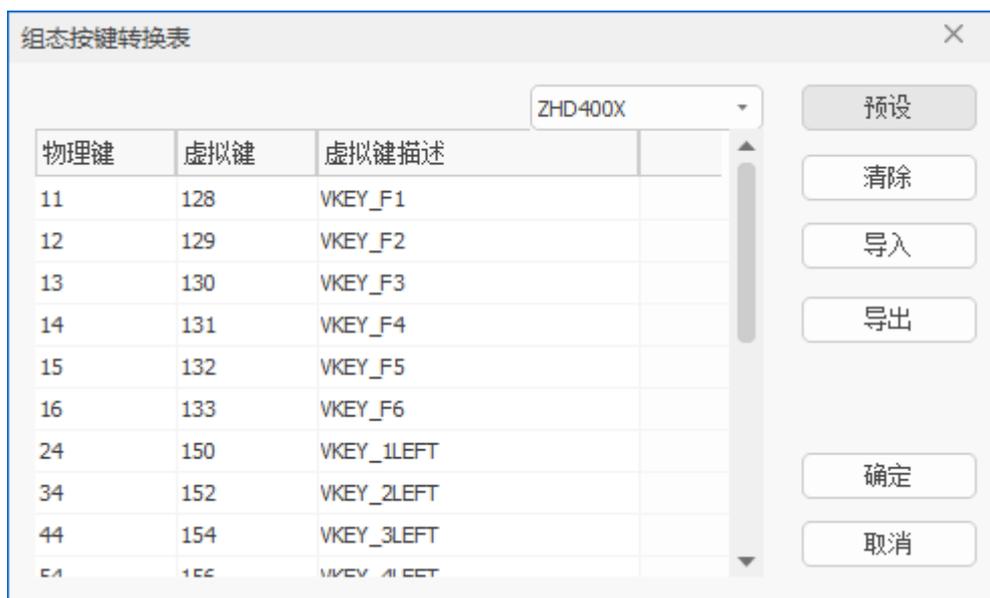
介绍: 用于将物理按键与虚拟键的功能绑定起来，实现通过操作物理按键即可使用虚拟键功能的效果。该工具已预设了 ZHD300X 和 ZHD400X 的按键功能。同时支持将已设置好的键值内容导出或导入。

使用方法:

(一) 若使用已预设的 ZHD300X 和 ZHD400X 的按键功能: 打开“按键转换”窗口→点击右上方的“下拉菜单”按钮, 即可选择对应型号示教盒→点击“预设”, 列表即可显示对应按键功能→点击“确定”设置成功。

(若要更改预设内容, 可点击“清除”即可全部清除; 若只更改部分内容可直接双击修改)

(二) 若需新建一套自定义按键功能: 打开“按键转换”窗口→在对应空白格处双击即可填入物理键值和虚拟键值→点击“确定”即可保存。



2.2.4.1. 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。

物理键的编码值由硬件决定, 程序中无法修改。使用不同的外部设备, 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 RTSys 软件里的组态按键转换表：

物理键	虚拟键	虚拟键描述
11	128	VKEY_F1
12	129	VKEY_F2
13	130	VKEY_F3
14	131	VKEY_F4
15	132	VKEY_F5
16	133	VKEY_F6
24	150	VKEY_1LEFT
34	152	VKEY_2LEFT
44	154	VKEY_3LEFT
54	156	VKEY_4LEFT

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

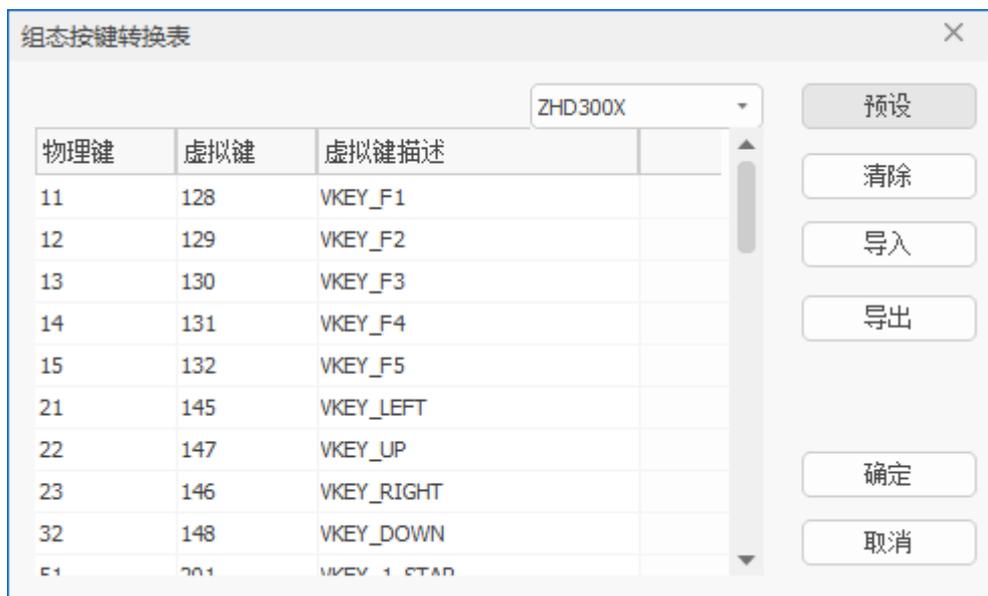
Global Const key_f3 = 13 '功能键 3

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 RTSys 软件里的组态按钮转换表：



2.2.4.2. 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 往后支持自定义功能。部分虚拟键值已定义功能，详情可参见[附录-虚拟键值表](#)。

2.2.4.3. 按键转换表的编辑

使用方法：

选择菜单栏“HMI”-“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

该工具已内置 ZHD300X 和 ZHD400X 的转换表，可在下拉菜单中选择。

2. 列表区

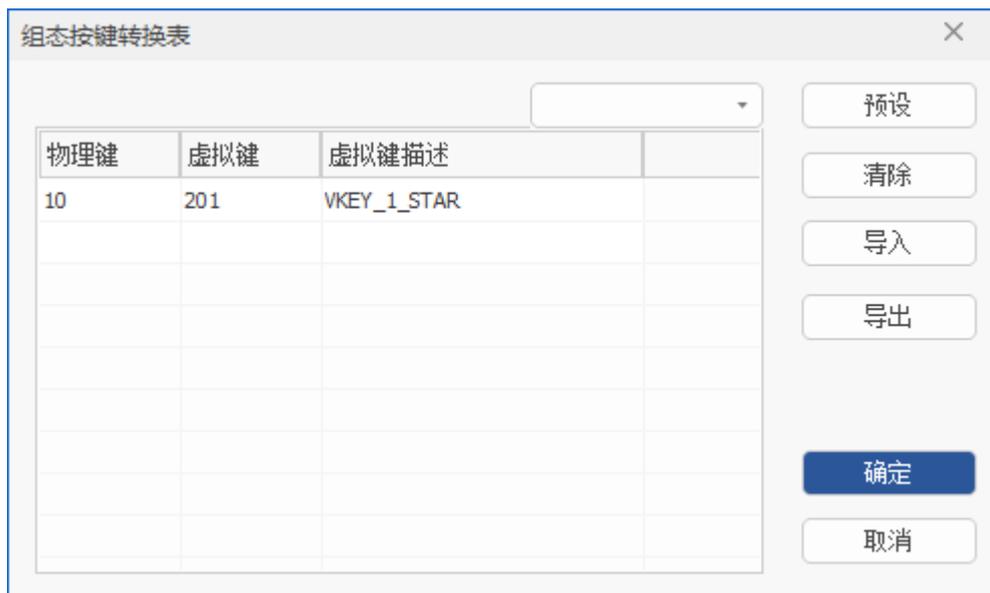
用于编辑并显示各物理键值对应的虚拟键值及功能。双击首行空白格即可手动输入。

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 往后则自定义功能。详情可参见[附录-虚拟键值表](#)。

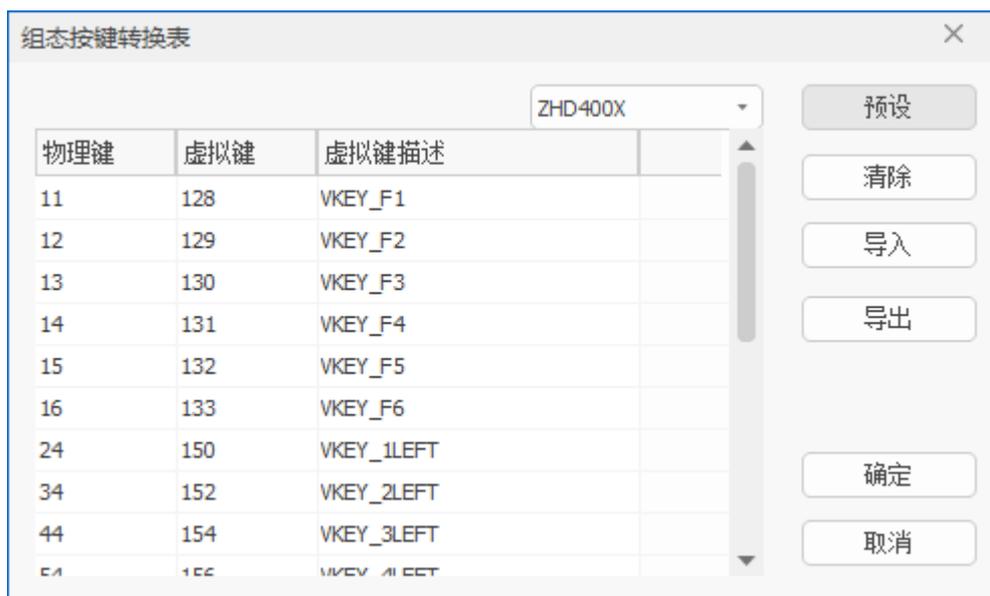
虚拟键描述：对当前虚拟键功能的说明。

示例：将物理键 10 与虚拟键 201 绑定，参考下图。



3. 功能区

预设：结合选择菜单，调出已经编辑好的转换表，目前只有 ZHD300X 和 ZHD400X。



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，否则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。

2.2.4.4. 按键转换指令

虚拟键值与虚拟键功能支持自定义绑定，可通过相应指令及程序编写实现。

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

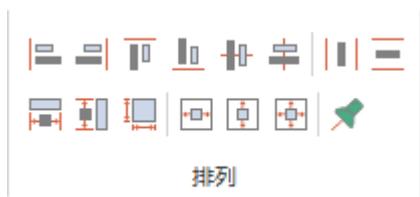
VKEY_SCAN: 读取虚拟按键编码

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。但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](#)”。

2.3. 排列

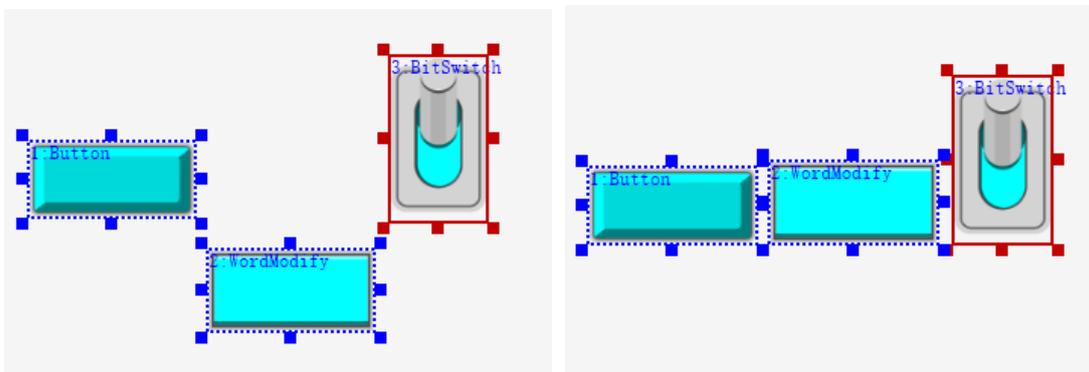


介绍：RTSys 提供多种对齐排列方式，将多个元件按一定规则整齐排列，使得整个 Hmi 界面更加美观有序。如下表：

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

使用方法：

1. 将所需排列的元素同时框选，选择适宜的排列方法即可。如下图所示，将三个元素“下对齐”。



注意：

1. HMI 使用排列功能时，需同时选中两个及以上的元件该功能才可使用。
2. 该功能的对齐目标是以框选元件中显示为红色框的为目标元件，以目标元件为标准进行排列。
3. 红色框选元件默认以最先添加的顺序为准，与元件编号顺序无关。
4. 若需自定义目标元件，则先选中目标元件，按住“ctrl”键，鼠标单个点击其他跟随元件，元件选择完毕后松开“ctrl”键，再选择排列方式。

2.4. 编辑



2.4.1. 批量修改地址

介绍：对多个 HMI 组态元件的寄存器地址进行批量修改，可修改为统一的寄存器类型，以及设置地址间距。通过菜单栏“HMI”→“批量修改地址”可打开如下窗口。

【编辑界面】

控件：显示选中多个元件中支持绑定寄存器的元件编号及名称。（注：仅显示支持绑定寄存器的组态元件。）

寄存器类型：选择对应元件需要绑定的寄存器类型，包括 AUTO（自动）、X（输入口 IN）、Y（输出口 OUT）、M(MODBUS_BIT)、S(状态寄存器)、D(MODBUS_REG/MODBUS_LONG/MODBUS_IEEE)、D.DOT（按位读取 MODBUS_REG）、DT（TABLE）、T（定时器）、C（计数器）。寄存器更多介绍可参考

个元件后按住“ctrl”键按顺序依次对元件进行选择)

4. 每次自动地址后, 若需重新修改寄存器类型并自动地址, 需重新手动对某个寄存器修改类型, 并将该寄存器之后的寄存器设置为 AUTO 类型。

2.4.2.Hmi 设置

对 HMI 系统进行初始属性设置, 可修改整体 HMI 窗口分辨率、起始基本窗口等属性, 详情可参加下表。通过菜单栏“HMI”→“Hmi 设置”即可打开 Hmi 系统设置的属性窗口。

其它打开方式: 单击 HMI 窗口画布外的空白地方也可打开 Hmi 系统设置属性窗口。

注: 使用 Hmi 组态时, 一般需先在“Hmi 设置”中设置好各参数属性。

名称	功能	说明
LCD 编号	设置 LCD 编号	连接到触摸屏时可通过该编号进行选择显示哪个 HMI 文件内容
背光时间	示教盒实际背光时间	/
屏保时间	设置屏保时间	/
起始基本窗口	设置 HMI 起始显示的基本窗口	默认显示 10 号窗口
起始置顶窗口	设置 HMI 起始置顶窗口	/
初始化函数	添加 HMI 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	添加 HMI 周期函数	上电后不断周期扫描的函数, 在 Basic 文件中定义, 函数的定义必须是全局(GLOBAL)的 SUB
压缩图片	选择是否对图片进行压缩	旧压缩: 有损压缩且不可逆(即使用 ZDevelop 中的压缩方式) 新压缩: 无损压缩, 图片画质不变 不压缩: 不压缩图片
图片质量	选择图片显示的质量 (注: 选择不压缩时, 图片质量决定是否防失真缩放)	标准: 图片显示质量较低, 但 HMI 性能较高 高: 图片显示质量较高, 但 HMI 性能降低
文本自适应	文本自适应元件大小	文本显示内容超出元件范围时自动缩小字体, 最小不低于用户设置的 50%
不使用文本库格式文本	True: 使用控件格式文本 False: 使用文本库格式文本	/

水平分辨率	窗口显示的分辨率	可编辑
垂直分辨率	窗口显示的分辨率	可编辑

用户能在“分辨率选择”界面中选择预设分辨率或自定义分辨率。在“修改内容”部分，可以选择是否让“窗口和控件”及“文字大小”（随宽度或高度变化）与新的分辨率设置同步调整。



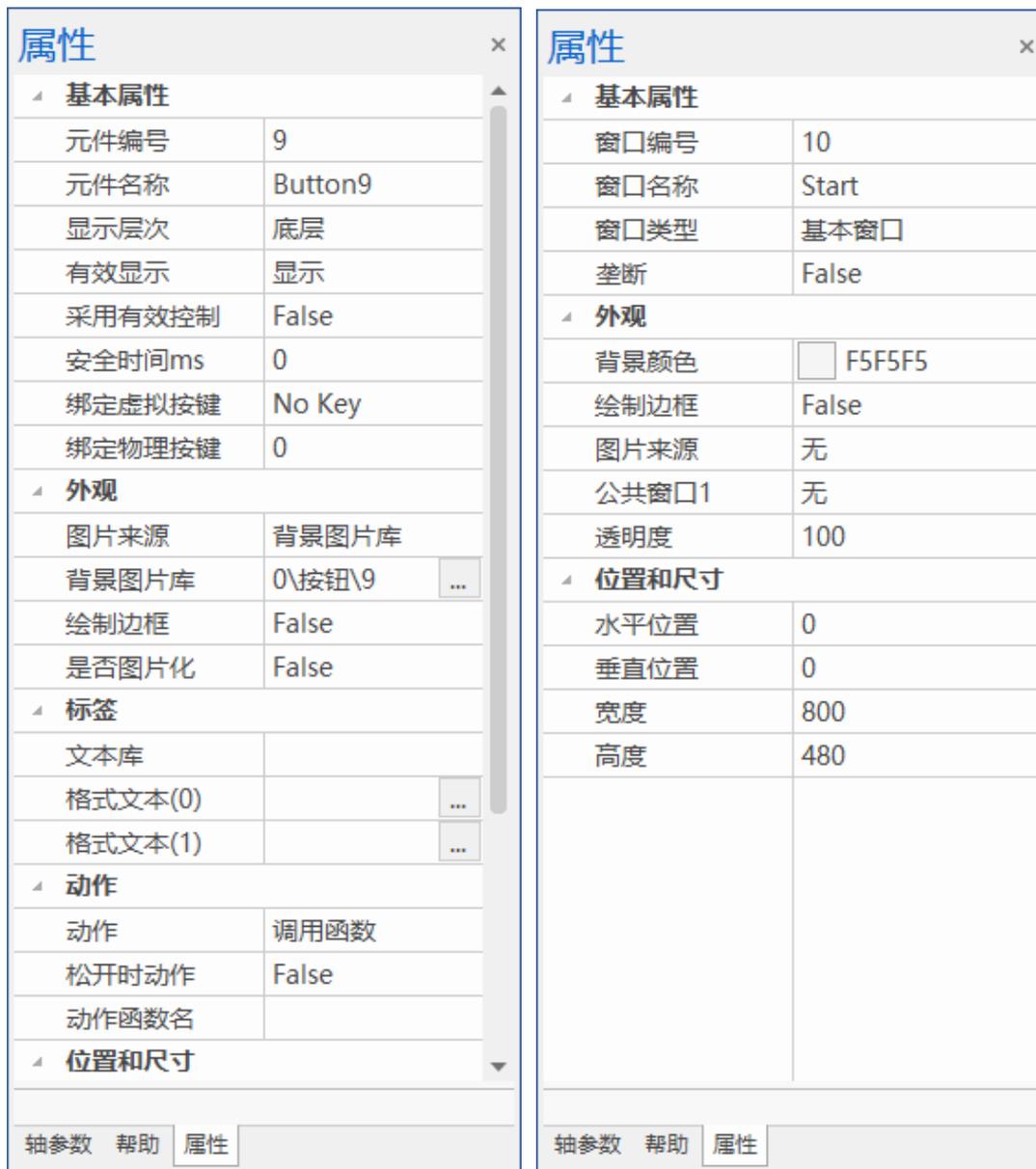
2.5. 显示设置



2.5.1. 属性

用于显示和设置 HMI 文件中的窗口/元件属性。打开该窗口需先新建/打开 HMI 文件，再在菜单栏“HMI”→“属性”即可打开，否则“属性”则呈灰色不可点击状态。

窗口和元件的属性窗口如下图所示：



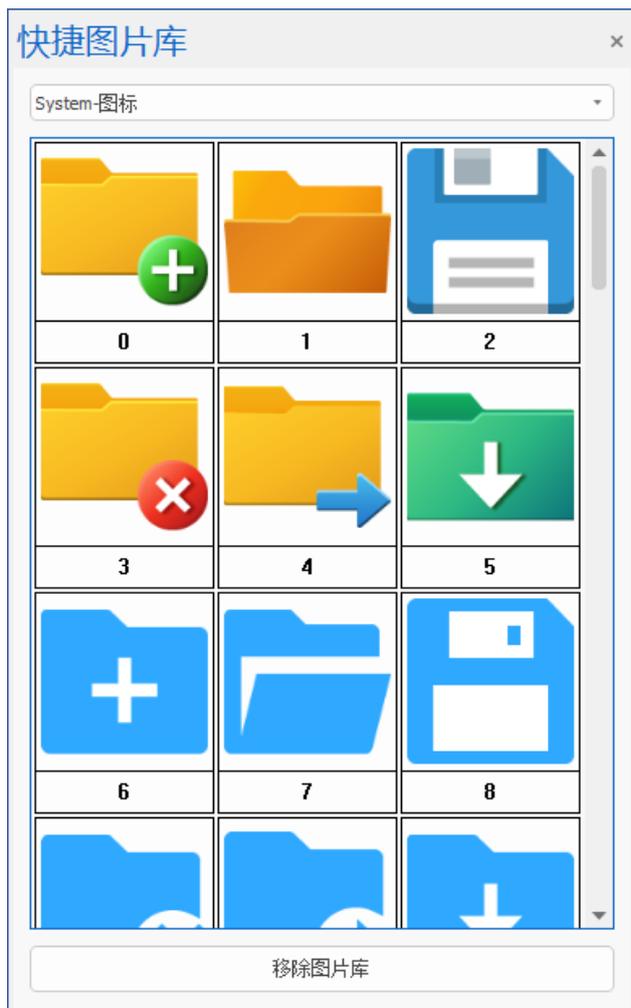
2.5.2.快捷图片库

用于给 HMI 元件快速应用或移除图片库中的样式。通过菜单栏“视图”→“快捷图片库”即可打开该窗口。

操作方式：

添加图片库：打开 HMI 文件，选中 HMI 窗口中的单个元件，打开快捷图片库，找到想要应用的样式图片，双击该图片，即可将图片快速应用到元件上。（图片库中有多个分类，可于[快捷图片库]窗口顶部点击则弹出下拉菜单进行选择）

移除图片库：单击选中元件，在[快捷图片库]窗口中点击“移除图片库”即可。



2.5.3.显示/隐藏图层

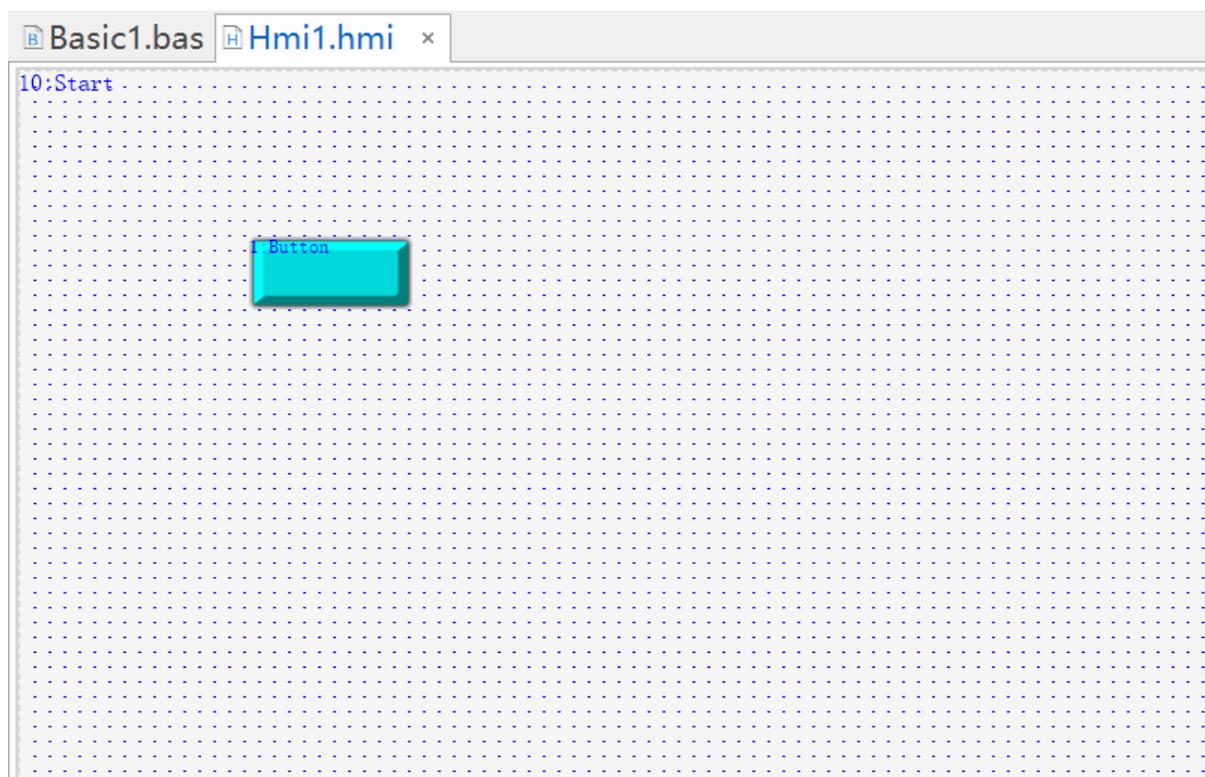
对设置在不同图层的元件进行选择显示或隐藏。（在菜单栏“HMI”→“显示/隐藏图层”进行设置）

操作方法：

1. 点击元件，打开属性窗口，在元件“显示层次”中进行选择，共有顶层、中层、底层可选。
2. 点击“显示/隐藏图层”弹出下拉菜单，选择某个图层进行显示或隐藏。

2.5.4.栅格和元件名称

选择 HMI 窗口中是否显示栅格，元件和窗口是否显示名称。栅格的作用便于用户将文件参考对齐。全部勾选则如下图显示：



2.6. 语言/状态切换

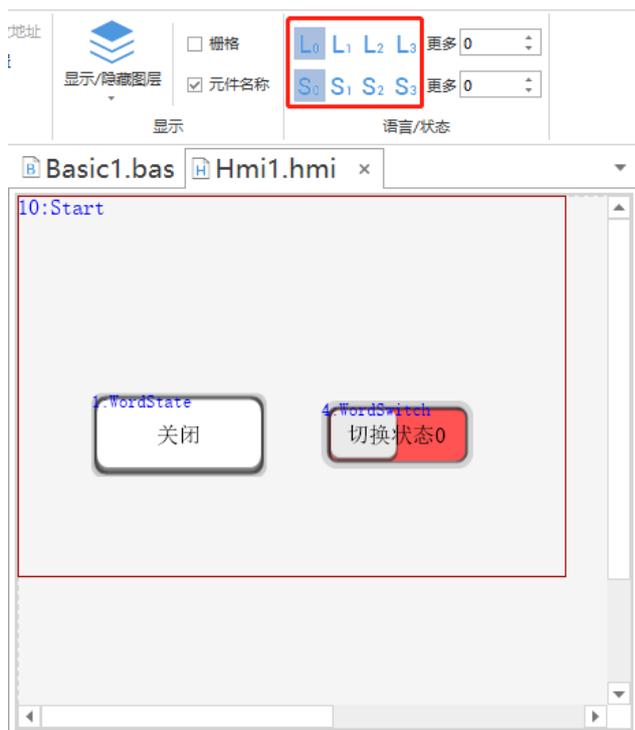


语言切换：对已调用文本库的元件进行语言切换。需先在文本库设置好当前状态下每个语言对应的内容，在该元件进行文本库调用，选择语言进行文本内容切换，L₀即对应语言 0，L₁即对应语言 1，以此类推，超出可在更多中输入语言编号。语言最多可设置 8 个，即 L₇。

注：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：对功能键或位状态/多状态元件进行状态切换，选择状态 S 即可切换到不同状态。S₀对应状态 0，S₁对应状态 1，以此类推。状态最多可设置 256 个，即 S₂₅₅。

注：当状态切换数值超过元件最大状态时，元件以最大状态值来显示。



切换语言 L₁ 后的效果如下：



切换状态 S₁ 后的效果如下：



第三章 组态窗口

3.1. 窗口概述

3.1.1. 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。（注：RTSys 新建窗口默认分辨率为 800*480。）

组态元件超出窗口尺寸时，该元件也能正常触发。

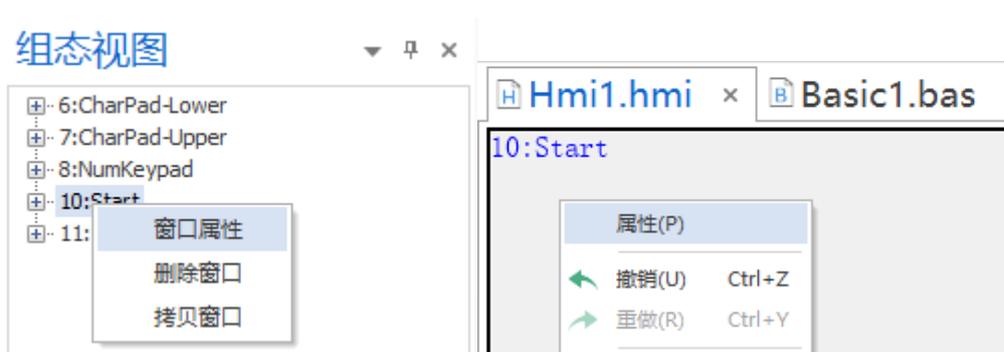
3.2. 窗口操作

3.2.1. 窗口属性

窗口属性包括窗口的类型、外观、尺寸等，可通过“属性”窗口对所选窗口进行设置修改：

打开属性窗口的方法：

1. 打开需要查看的窗口，鼠标左键单击窗口区域，右侧属性栏即可显示；
2. 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；
3. 在组态文件编辑窗口，单击右键，打开“属性”，如右图；



组态视图能显示全部窗口，窗口可设置的属性如下图所示：



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， 图片名称不超过 26 个字符
公共窗口 1	设置当前窗口的公共窗口 1	当前窗口可以显示公共窗口的控件， 最多可设置 3 个公共窗口

透明度	背景透明度	预留，暂不支持使用
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	/

3.2.2. 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。建立 Hmi 文件后自带 3 个软键盘窗口及 1 个基本窗口（窗口 10），可直接在已有的窗口 10 中进行组态编辑，也支持自行添加新建窗口。

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



拷贝窗口：拷贝已创建的窗口及其内容，并新建一个窗口。即可直接将已设计好的组态界面直接复制生成另一个新窗口。

操作方法：打开“组态视图”选中需拷贝的窗口，右键单击选择“拷贝窗口”即可弹出如下图窗口，根据需求修改新建窗口号和窗口名即可。



新建窗口后，若需对窗口进行设置相关属性或尺寸位置等信息则需要打开窗口“属性”修改。打开“属性”窗口的方法有两种：

1. 在组态视图能显示全部窗口和各窗口下的元件，选中窗口单击右键选择“窗口属性”，打开元件“属性”窗口则双击组态视图中该元件名称即可。

2. 直接单击窗口画布或单击画布中的元件便能打开对应的属性窗口。



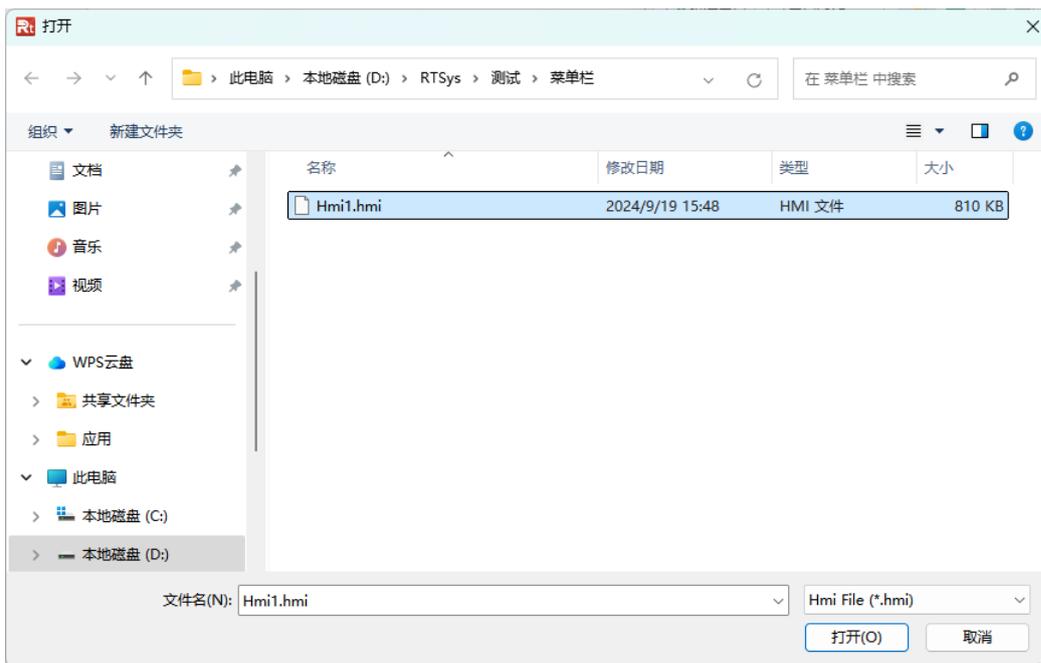
3.2.3. 窗口导入

介绍: 在当前 Hmi 项目中导入其他项目中已创建的 Hmi 窗口（可选一个或多个同时导入）。

操作步骤:

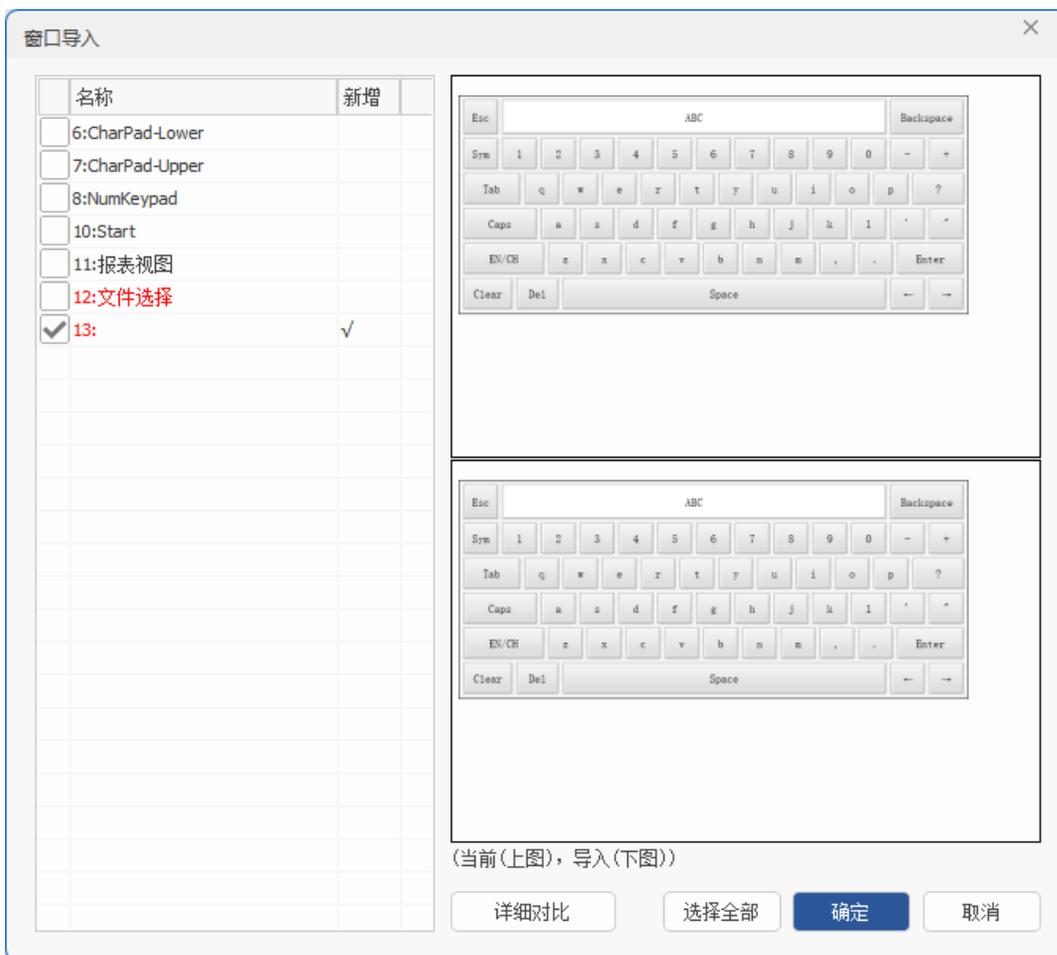
1. 在菜单栏“Hmi”中点击“导入窗口”；

2. 弹出文件选择窗口，选择需要导入的窗口所在的项目文件路径，选择需要的.Hmi 文件；

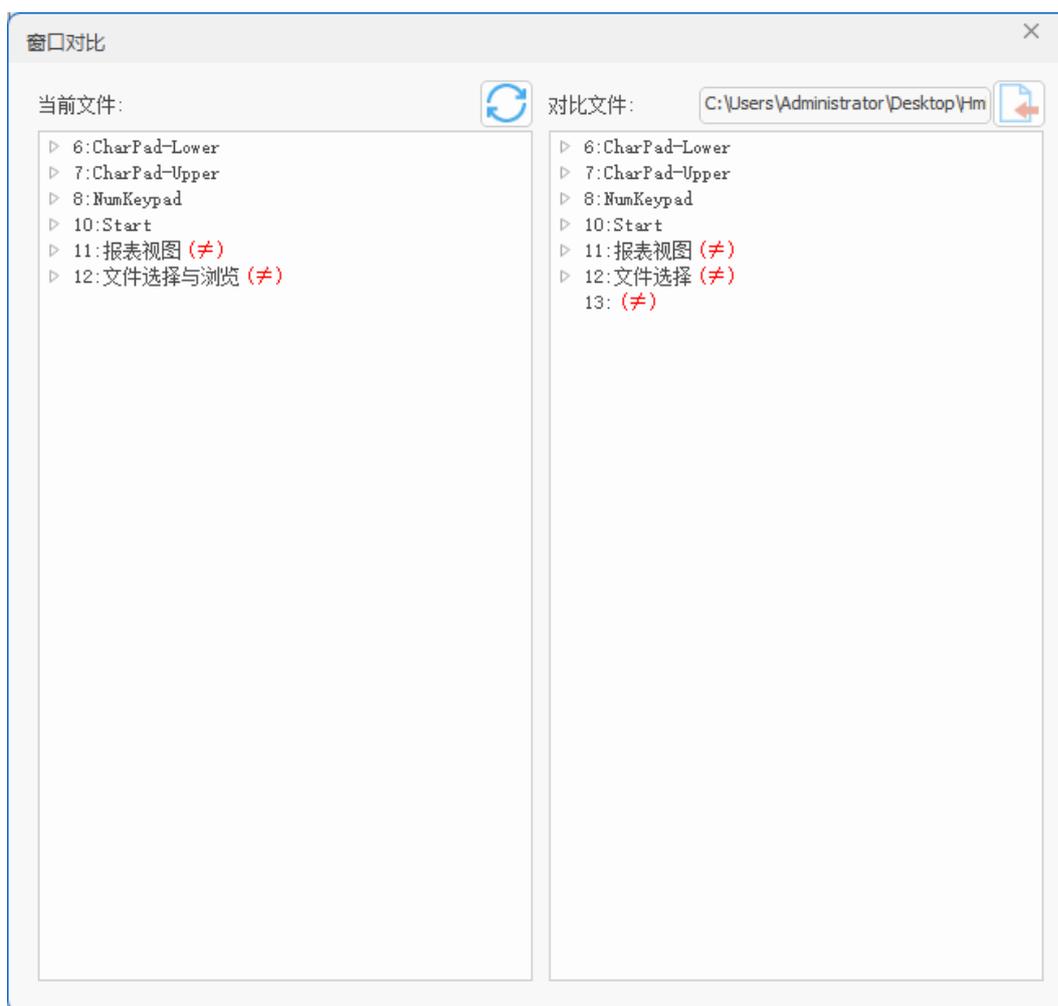


3. 打开需要的.Hmi 文件后，弹出“窗口导入”窗口。左侧显示窗口列表，右侧显示窗口对比图。

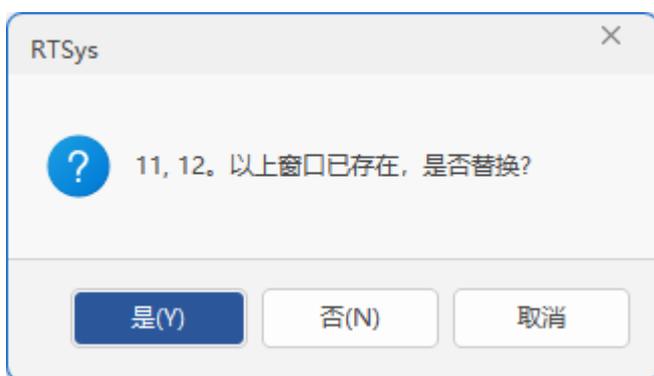
注：若导入的 Hmi 项目中有窗口 ID 与窗口名称均不一致，则窗口名称显示为红色，并默认勾选当前项目中无相同窗口 ID 的窗口，用户可选择是否导入/替换原窗口。



若点击“详细对比”，则会弹出“窗口对比”窗口。



4. 勾选需要导入的窗口，点击“确定”，若选择的窗口中有与原文件窗口冲突的窗口号，则会弹出以下窗口，选择“是”则替换，选择“否”则只导入未冲突的窗口。

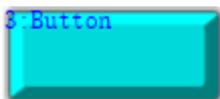


3.2.4. 窗口调用

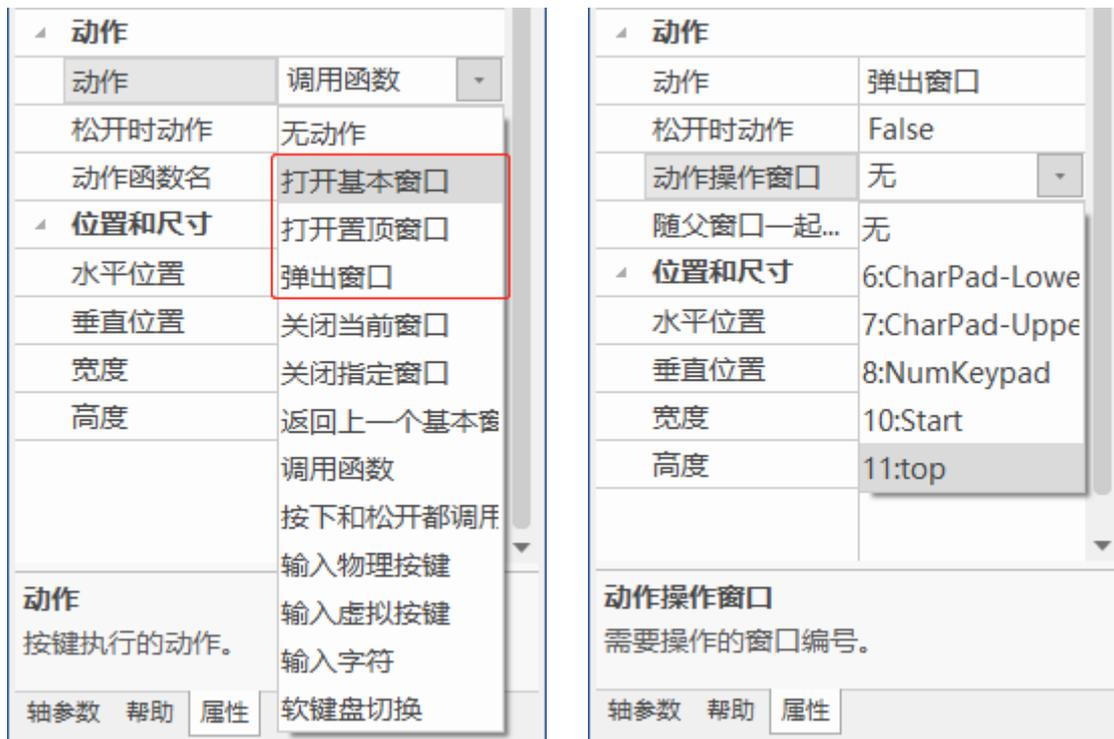
窗口调用有如下两种方式。

方式一：功能键调用

1. 选择“控件箱”-“功能键”，新建一个功能键按钮。



2. 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，其中菜单窗口属于弹出窗口类型。如下左图。



3. 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即可，如上右图。

注意：若要打开的窗口类型与选择的窗口类型不一致，则窗口类型会被强制转换。

方式二：程序指令调用

在元素属性“动作”中选择“调用函数”，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。

动作	
动作	调用函数
松开时动作	False
动作函数名	openwin

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

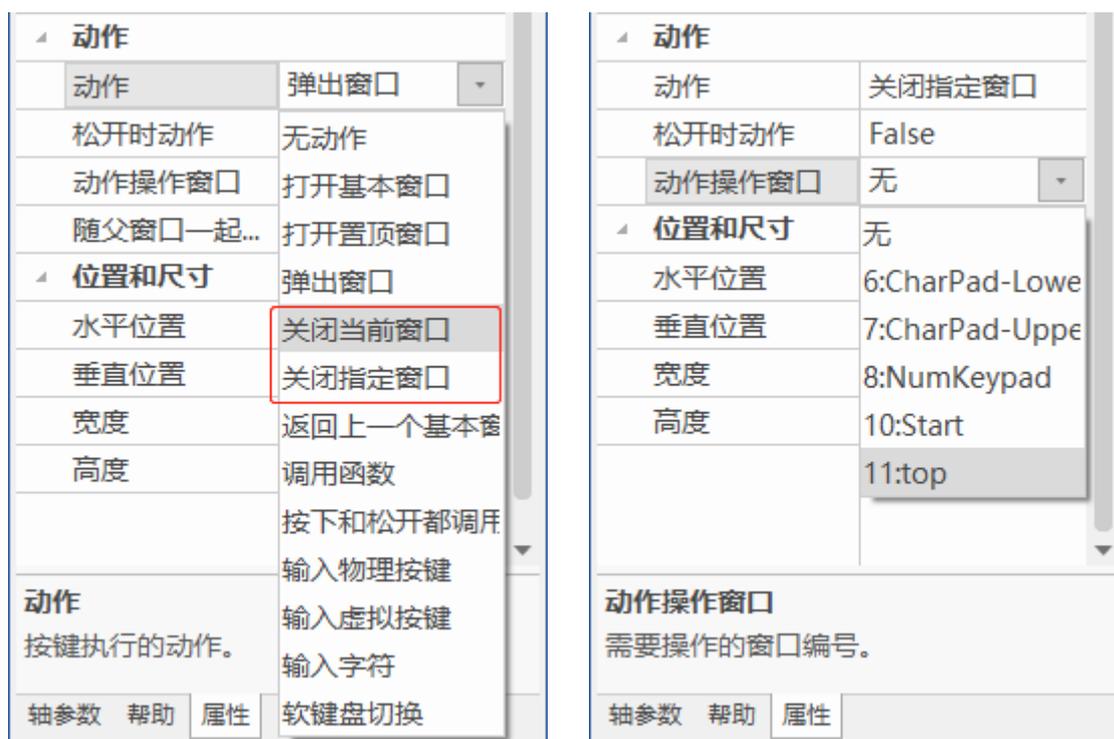
3.2.5. 窗口关闭

关闭窗口有两种方式，可参考以下内容。注意：**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，切换窗口操作一般把功能键放在基本窗口内；关闭弹出窗口和置顶窗口操作一般把功能键放在要关闭的窗口内。

窗口关闭的动作类型主要有两个：关闭当前窗口；关闭指定窗口（须在“动作操作窗口”设置窗口编号，如右图）



方式二：程序指令关闭

程序指令关闭窗口主要通过 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
```

3.2.6. 公共窗口

介绍：用于对多个窗口指定同一个公共窗口，有利于共用的内容可以显示在当前窗口里面。

使用方法：在“窗口属性”中为当前窗口选择公共窗口。一个窗口最多可以设置 3 个公共窗口。

注意：添加公共窗口后，公共窗口里的元件在当前窗口仅显示（窗口背景不显示），不可操作修改元件属性。修改元件属性须打开元件所在窗口才可。对已设置好的元件动作下载运行后动作可执行生效。



示例:

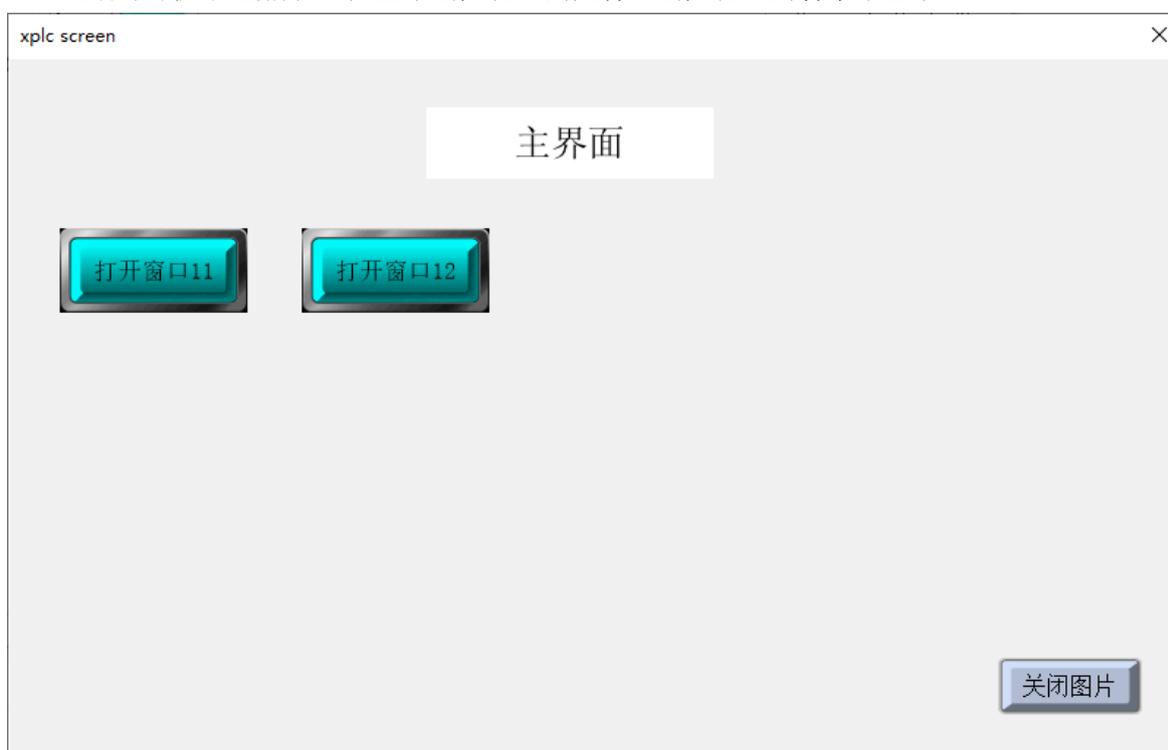
1. 如下图，没有添加公共窗口的显示情况，每个窗口只显示当前窗口的元件。



2. 给 10 号窗口添加一个公共窗口 13（最多可添加 3 个），这个窗口的元件就会显示在 10 号窗口里，但不能通过 10 号窗口对元件进行修改。



3. 运行下载后在当前窗口只显示公共窗口的元件，公共窗口的背景不显示。



通过操作主窗口（窗口 10）的元件可打开对应的窗口，在窗口 10 中的公共窗口元件动作也可生效。

以下面图片为例：点击打开窗口 11 和 12 之后，主窗口上显示了窗口 11 和窗口 12，公共窗口 13 的元件可对窗口 11 和窗口 12 进行关闭操作，但无法对窗口 10 和窗口 13 进行操作。

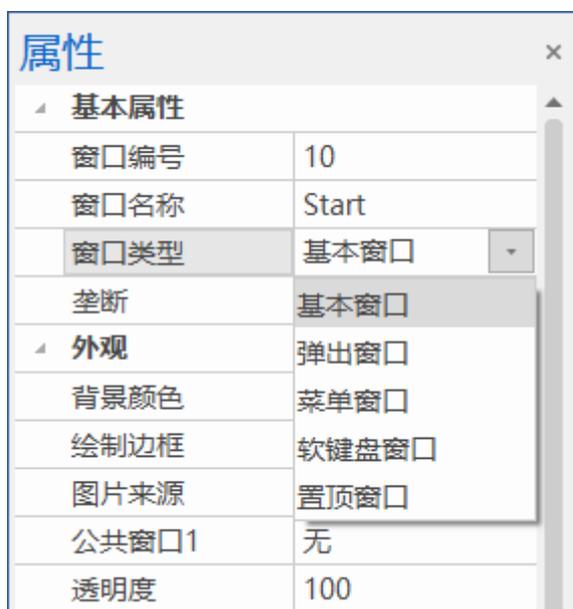
（注：本例对窗口 11 和 12 的大小及位置做了调整，因此打开不会覆盖主窗口）



3.3. 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。



3.3.1. 基本窗口

介绍：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。新建窗口默认为基本窗口类型，且基本窗口可通过程序或元件操作进行切换。

注意:

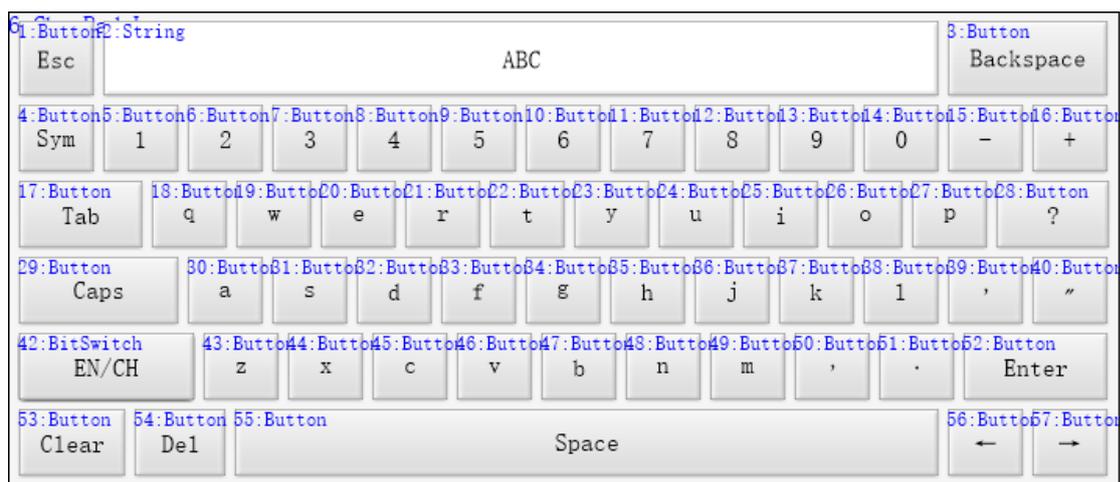
1. 基本窗口不支持关闭。
2. 触摸屏同一时间只能显示一个基本窗口。
3. 与触摸屏连接使用时，仅须保证 HMI 分辨率与触摸屏分辨率一致即可。各个窗口大小可任意设置，不超出屏幕尺寸即可。

3.3.2.软键盘窗口

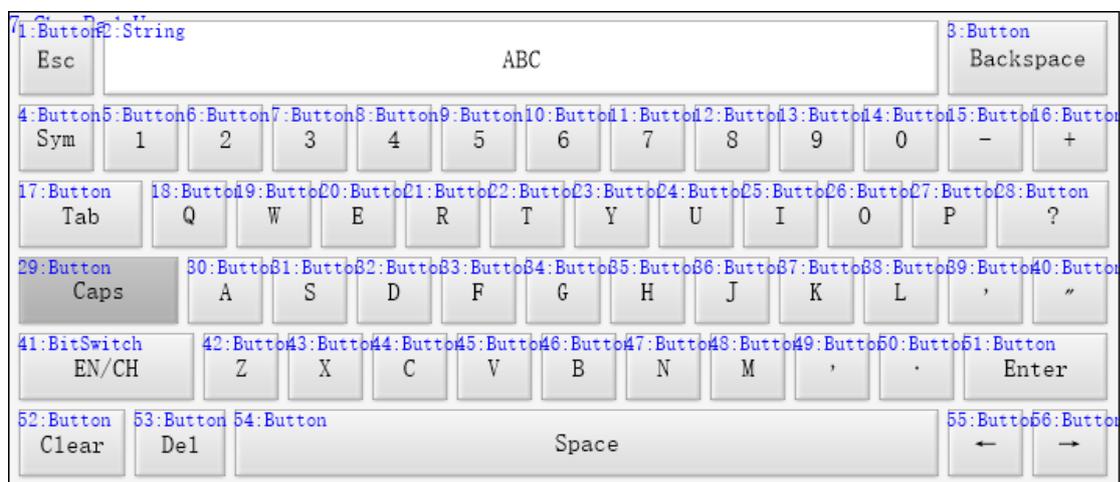
介绍: 用于“值”元件和“字符显示”元件等需要自定义输入数据的场合。

使用方法: 新建的 Hmi 文件内置有三种软键盘窗口可供选择，无需新建。可在允许软键盘编辑的元件中直接调用。窗口号分别为 6、7、8。（即：打开“值”元件的属性窗口→“可编辑”选择 True→选择“软键盘窗口号”）

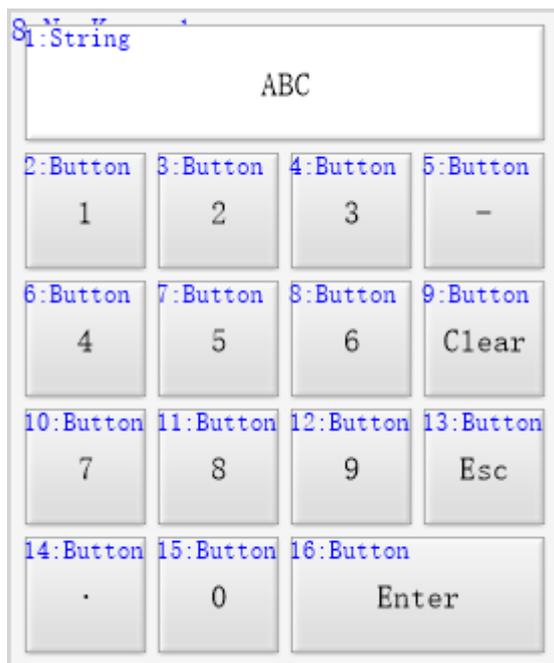
注意：调用软键盘窗口时，**只能在显示数据支持修改的元件中使用**，比如“值”、“字符显示”元件。不能用于“功能键”等元件。各类元件用法请参考[第四章](#)。



6 号软键盘窗口



7 号软键盘窗口



8号软键盘窗口

3.3.3.弹出窗口

介绍：弹出窗口类似是对话框一样的动态弹出窗口。

使用方法：弹出窗口需要通过程序或元件操作打开，同时也需要通过程序指令或元件（位元件/字元件等）操作关闭/切换窗口。当建立并打开多个弹出窗口时，按照调用顺序，依次显示调用的弹出窗口。

提示：

1. 当多个弹出窗口重叠在一起时，弹出窗口中的功能按键仍然能正常触发。
2. 调用出弹出窗口后，仍支持对基本窗口等其他类型窗口进行操作。



3.3.4. 菜单窗口

介绍：菜单窗口属于弹出型窗口，通过程序或元件操作调用后弹出。调用与关闭方法与弹出窗口一致。

注意：

1. 菜单窗口弹出后，**获得最大操作权限**，此时只能对菜单窗口进行操作。当点击到非菜单窗口区域时，菜单窗口自动关闭。
2. 菜单窗口打开后的**位置是跟随对应操作的元件位置进行显示**，与菜单窗口属性中设置的位置无关，因此设置弹出菜单窗口时注意调整好操作元件位置。



3.3.5. 置顶窗口

介绍：总是在最前端显示的窗口。

使用方法：调用/关闭置顶窗口须通过程序或元件操作。（一般建议设置为一个小窗口，可以用来实现工具条等。）有多个置顶窗口时，按照调用顺序显示，后调用的窗口在之前调用窗口的上层。

置顶窗口有两种显示方式：初始化显示和手动调用显示。

显示方式	说明
初始化显示	通过菜单栏“Hmi”-“Hmi 设置”-“属性”-“起始置顶窗口”来选择某个窗口是否上电即显示置顶窗口（设置该属性后，置顶窗口将覆盖起始基本窗口）。

<p>手动调用显示</p>	<p>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置顶窗口。 注意：同一窗口不可由两个元件设置“打开不同类型窗口”的动作。 (即一个窗口只能选择一种窗口类型打开)</p>
---------------	---

注意：将窗口设置为置顶窗口类型后，打开该窗口将一直处于置顶状态，但对其他窗口中的元件也可进行操作，置顶窗口不会关闭，关闭置顶窗口须通过程序或元件操作关闭。



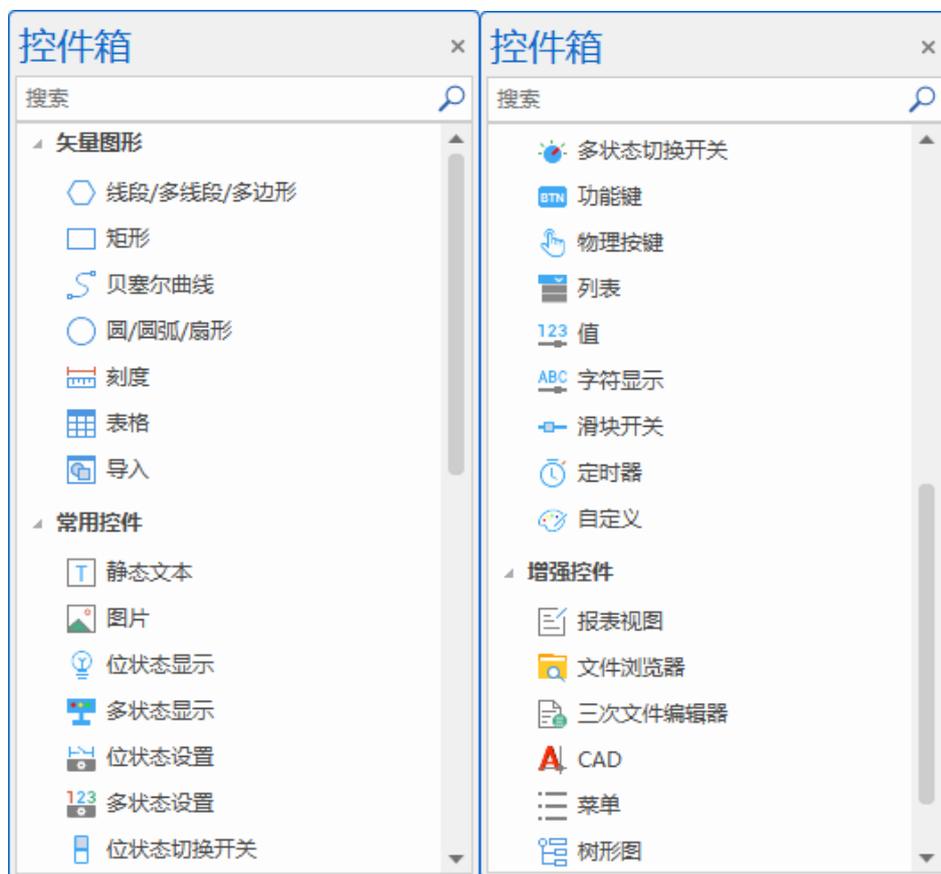
第四章 组态元件

4.1. 组态快捷工具

4.1.1. 元件菜单

Hmi 开发主要用到窗口和组态元件，先建立不同的显示窗口，再根据需求在各个窗口上添加不同的组态元件，并结合程序开发实现相应的功能。

组态元件在平台界面最左侧“控件箱”中选择并添加到窗口（支持搜索功能），或是从菜单栏中“HMI”-“控件箱”中打开。



控件箱元件菜单：

名称	图示	说明
矢量图形		
线段/多线段/多边形		根据绘制点数画出相应的线段或多边形
矩形		绘制矩形

贝塞尔曲线		四点绘制三阶贝塞尔曲线
圆/圆弧/扇形		拖动绘制整圆/椭圆/圆弧/扇形
刻度		绘制等距的间隔刻度
表格		绘制 3×3 表格，可自定义修改表格样式
导入		导入矢量图形
控件		
静态文本		在窗口添加静态文字，可自定义设置相关属性
静态图片		从系统/背景图片库插入图片
位状态显示		根据绑定的位寄存器地址的值显示对应的状态
多状态显示		根据绑定的字寄存器地址的值显示对应的状态
位状态设置		根据元件动作的状态设置位寄存器地址的值
多状态设置		根据元件动作的状态设置字寄存器地址的值
位状态切换开关		根据元件动作设置位寄存器地址的值并显示对应状态 (位状态显示和位状态设置的功能结合)
多状态切换开关		根据元件动作设置字寄存器地址的值并显示对应状态 (字状态显示和字状态设置的功能结合)
功能键		根据元件动作实现状态切换/窗口切换/软键盘切换等，只有两种显示状态并且无法绑定寄存器
物理按键		用于与虚拟按键/实际按键绑定后通过实际按键动作
列表		可显示多个列表项，通过绑定的寄存器的值切换对应选项
值		编辑并显示数值以及修改对应绑定寄存器的值
字符显示		编辑或显示字符串并更改字寄存器的值
滑块开关		通过拖拽滑块更改字寄存器数值
定时器		定时刷新进行重复动作

自定义		在元件区域内通过调用 basic 函数实现动态绘图
报表视图		以表格的形式呈视多组数据，用于显示和管理报表数据
文件浏览器		显示当前目录、以表格形式显示文件内容
三次文件编辑器		支持 Hmi 中开发三次程序的编辑元件
CAD		显示矢量图形
菜单		设置菜单项，通过点击菜单项触发调用 SUB 动作
树形图		以树状图形式显示所有表项，单击树节点坐标的角图标（或双击树节点内容）可进行子树的展开/缩起，单击树节点内容进行触发动作

4.2. 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。

4.2.1. 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。

绑定的寄存器类型	M
绑定的寄存器编号	0

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	不同型号控制器的寄存器个数有区别

	INT32: MODBUS_LONG FLOAT32: MODBUS_IEEE	
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 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
@	Basic 定义的变量、数组	必须是 GLOBAL 全局类型才能访问

4.2.2.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	
动作	调用函数
松开时动作	False
动作函数名	

动作名	功能	说明
元件通用动作		
无动作	无	/
调用函数	调用 Basic 定义的 SUB 函数	函数必须是 GLOBAL 全局类型。
功能键 Button		
打开基本窗口	以基本窗口类型打开窗口	窗口号通过“动作操作窗口”选择。
打开置顶窗口	以置顶窗口类型打开窗口	
弹出窗口	以弹出窗口类型打开窗口	
关闭当前窗口	/	关闭当前功能键所在的窗口。
关闭指定窗口	关闭所选择的窗口	窗口号通过“动作操作窗口”选择。
返回上一个基本窗口	打开最后一个基本窗口	打开最近一次操作的基本窗口
按下和松开都调用函数	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”
输入物理按键	与物理按键绑定	需要物理按键对应表。
输入虚拟按键	设为虚拟按键	通过“虚拟按键码”选择编号。
输入字符	输入字符串	只能在软键盘窗口内使用
软键盘切换	切换窗口	必须在非基本类型窗口内才可以使用。且只能同类窗口间切换。

位状态切换开关 BitSwitch / 位状态设置 BitModify		
状态设置为 1	按下时, 置 1	开关类型
状态设置为 0	按下时, 置 0	
状态反转	取反, 为 1 时变成 0, 为 0 时变成 1	
状态恢复	按下时置 1, 松开时置 0	
多状态切换开关 WordSwitch / 多状态设置 WordModify		
修改数据	写入数据到寄存器	数据值通过“动作数据”设置。
数据增加/减少	寄存器原来值增加/减少数据	寄存器通过“寄存器类型”和编号选择。
循环	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	寄存器通过“寄存器类型”和编号选择。 如果寄存器原来值大于“状态数量”, 会先按“原来值”-“动作数据”递减至设定的状态数量范围内, 即开始循环。

4.2.3.基本属性

基本属性	
元件编号	1
元件名称	Button1
显示层次	底层
有效显示	显示
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

基本属性	描述
元件编号	在当前窗口按添加的顺序从 1 开始编号
元件名称	名称+编号。支持自定义修改
显示层次	在多个元件叠加时, 可以设置元件的显示层次 顶层: 显示在最外层, 覆盖底下元件; 中层: 被顶层元件覆盖, 覆盖底层元件; 底层: 被顶层和中间层元件覆盖
有效显示	决定该元件是否显示于界面之中 显示: 该元件显示于界面之中; 不显示: 该元件不显示于界面之中;

	仅显示, 不可用: 该元件显示于界面之中但是点击该元件时不会产生任何改变。
采用有效控制	采用设备的寄存器控制该元件是否显示, 默认 False 若选择 True, 须选择寄存器类型和编号。当寄存器置 0 时该元件则隐藏, 非 0 时则显示。
安全时间 ms	最少按键时间(ms)
绑定虚拟按键	选择要绑定的虚拟按键码
绑定物理按键	绑定示教盒上面的物理按键

4.2.4.外观

外观		外观	
图片来源	背景图片	图片来源	背景图片库
背景图片		背景图片库	0\按钮\13 ...
绘制边框	False	绘制边框	False
是否图片化	False	是否图片化	False

外观	描述
图片来源	部分元件支持显示图片, 可在此选择图片来源, 选择图片来源后可在下一行选择目标图片添加。 无: 不添加图片 背景图片库: 从背景图片库选用图片 背景图片: 从项目中选用图片
背景图片库	从背景图片库中选择图片。图片先添加到图片库中, 再在该处进行选择
背景图片	从项目中选择图片。图片先添加到项目的“文件视图”中, 再在此处进行选择 图片命名不可超过 26 个字符
绘制边框	是否绘制元件的边框
边框颜色	选择元件边框的颜色
是否图片化	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

4.2.5.位置和尺寸

位置和尺寸	
水平位置	478
垂直位置	340
宽度	109
高度	53

位置和尺寸	描述
水平位置	元件在 Hmi 窗口中放置位置的水平距离。以 Hmi 显示窗口左上角的位置为(0,0)
垂直位置	元件在 Hmi 窗口中放置位置的垂直距离。以 Hmi 显示窗口左上角的位置为(0,0)
宽度	设置元件的显示宽度
高度	设置元件的显示高度
注意：元件位置和尺寸最好不要超出窗口的范围	

4.2.6.格式文本

用于编辑元件各状态的文字显示内容。同时也支持在元件上直接编辑文字。

对于需要设置“状态数量”的元件，“状态数量”支持自定义修改（最多支持 256 种状态）。“格式文本”的个数随状态数量修改而增减。

标签	
文本库	
状态数量	4
格式文本(0)	...
格式文本(1)	...
格式文本(2)	...
格式文本(3)	...

点击格式文本后的三个点弹出“格式文本”设置窗口。如下图所示。该窗口支持内容文本输入以及提供多种文本样式进行设置。具体使用如下：

- **【内容】**：自定义写入文本内容，支持换行。输入内容后点击“确定”即可保存。同时支持对该文本样式进行设置，包括对齐方式、文本颜色、对齐颜色等，样式具体设置可参考下表。注意：若使用修改元件背景颜色及样式，须先取消该元件已应用的图片库/背景图片样式才显示。

- **【拷贝到其他状态】**：将该格式文本除了“内容”项的其他设置项拷贝到其他状态的格式文本中（即：将当前状态的文本样式应用到其余状态下），单击其他状态的格式文本可以查看到参数与该文本参数一致。**注意：仅应用于当前元件的不同状态，无法拷贝至其他元件！**
- **【将该格式文本保存为默认格式文本】**：将当前元件设置的文本样式保存为全局默认格式，即将当前文本样式应用至该项目中的元件样式。勾选即生效。**注意：只能应用于新创建的元件，已创建的元件样式不改变！**
- **【创建时使用默认格式文本】**：新创建的元件应用已设置的默认格式文本。勾选即生效。

操作	功能	说明
内容	输入要显示的文本内容	/
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
颜色	选择字体颜色	/
尺寸	选择字体大小	/
字体	选择字体	将字体文件（支持后缀为.ttf/.zft）添加到项目中
背景颜色	选择背景颜色	元件图片来源选择“无”时背景颜色才应用至元件中
填充样式	填充到背景的样式	包括六种样式，元件图片来源选择“无”时填充样式才

		应用至元件中
样式颜色	选择样式颜色	元件图片来源选择“无”时样式颜色才应用至元件中
闪烁时间	选择闪烁时间	格式文本内容闪烁显示，闪烁间隔时间为用户设置的时间，0ms 为不闪烁
方向	选择跑马灯的方向	格式文本内容按照用户选择的方向移动
移动速度	选择跑马灯的移动速度	格式文本内容按照用户选择的速度移动，若方向为不移动则该项不会作用

4.2.7. 图片库和文本库



组态元件支持添加图片及文本，图片可从“背景图片库”、“背景图片”中添加；文本则支持从“文本库”直接调用。

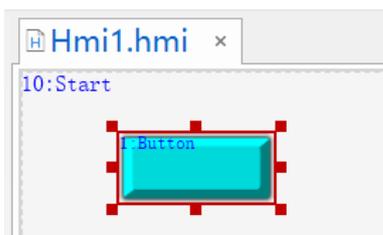
使用以上途径添加前均需先创建/添加图片或文本内容。插入背景图片需先将图片添加到项目中，再在此处选择要加载的图片名称。

图片库和文本库的建立方法参见【[图片库](#)】、【[文本库](#)】章节。

4.3. 元件介绍及使用

组态元件的基本使用介绍：

1. 添加组态元件：打开 Hmi 文件，在窗口 10 或新建窗口中，从控件箱（菜单栏“HMI”→“控件箱”）单击选择某个元件，移动至窗口中单击放置即可添加成功。（若想取消已选择但未放置的元件，将鼠标置于窗口中单击右键即可）。
2. 修改元件大小/位置：单击选中元件后，将鼠标放在下图红色方块上直接拖拽即可改变大小，或由“属性”窗口的宽度、高度设置。修改位置则直接选中元件按住鼠标左键不放拖拽即可，或由“属性”窗口的水平位置和垂直位置调整。



3. 设置元件功能属性：鼠标单击选中元件，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。（若“属性”窗口不能被自动打开，点击菜单栏“HMI”-“属性”先打开属性窗口后，再点击元件。）

注意：“属性”中部分功能需下载运行仿真后才可看到效果！

4.3.1. 线段/多线段/多边形

4.3.1.1. 线段

介绍：“线段”为直线，直线的长度和斜率由用户自定义。可直接拖拽元件定义宽高，也可在元件属性里定义。

使用方法： 点击菜单栏“HMI”→“控件箱”→“矢量图形”→“线段/多线段/多边形”，确定两点即可画出一个线段，在即将开启第二段画线操作时点击右键，在弹出的小菜单中点击“完成”即可画出线段。具体操作方式如下图所示。**提示：** 绘制水平直线或垂直直线时，只需按住“Ctrl”键移动鼠标确定方向和长度即可。



1. 属性窗口：

属性		起点形状	样式1
基本属性		起点尺寸	中
元件编号	1	终点形状	样式1
元件名称	Line1	终点尺寸	中
显示层次	底层	位置和尺寸	
有效显示	显示	水平位置	205
采用有效控制	False	垂直位置	149
外观		宽度	128
当前颜色	000000	高度	31
线宽	1		
线段类型	默认		

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	顶层：显示在最外层，覆盖底下元件； 中层：被顶层元件覆盖，覆盖底层元件； 底层：被顶层和中间层元件覆盖（默认）。
有效显示	选择元件是否显示	显示：该元件显示于界面之中； 不显示：该元件不显示于界面之中； 仅显示，不可用：该元件可正常显示，但点击该元件时动作不生效
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 True 才会出现下方的三个参数
有效的设备编号 (有效控制为 True)	设备编号	默认 local
有效的寄存器类型 (有效控制为 True)	选择寄存器类型	多种寄存器下拉列表选择
有效的寄存器编号 (有效控制为 True)	选择寄存器编号	寄存器值为 0 时不显示，非 0 时显示
当前颜色	选择线段颜色	/
线宽	线段的宽度	默认宽度为 1，最大为 20
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	选了终点形状才有效
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	/

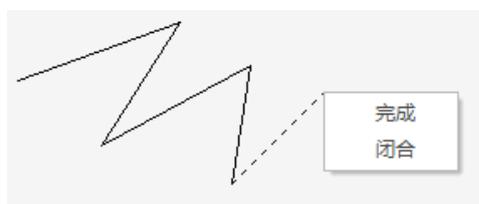
4.3.1.2. 多线段

介绍：由多个连续的线段组合而成的非封闭图形，即为多线段。

使用方法：点击菜单栏“HMI”→“控件箱”→“矢量图形”→“线段/多线段/多边形”，连续在不同位置点击多个点画出多个连续的线段，点击鼠标右键会弹出一个小窗口，点击“完成”即可画出多线段。画出的线段可以拥有任意数目转角，可以画成任意图形。**提示：**绘制水平直线或垂直直线时，只需按住“Ctrl”键移动鼠标确定方向和长度即可。

注意：

1. 线段确定后各点的相对位置不能修改，只支持对整体形状进行缩放；
2. 若起始点和结束点重合在同一个坐标，但不能认为其是封闭图形，无法填充由线定义的区域。



1. 属性窗口：

属性		线宽	1
基本属性		线段类型	默认
元件编号	1	位置和尺寸	
元件名称	Polyline1	水平位置	112
显示层次	底层	垂直位置	141
有效显示	显示	宽度	159
采用有效控制	False	高度	51
外观			
当前颜色	000000		

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	顶层：显示在最外层，覆盖底下元件； 中层：被顶层元件覆盖，覆盖底层元件； 底层：被顶层和中间层元件覆盖（默认）。
有效显示	选择元件是否显示	显示：该元件显示于界面之中； 不显示：该元件不显示于界面之中； 仅显示，不可用：该元件可正常显示，但点击该元件时动作不生效
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 True 才会出现下方的三个参数
有效的设备编号 (有效控制为 True)	设备编号	默认 local
有效的寄存器类型 (有效控制为 True)	选择寄存器类型	多种寄存器下拉列表选择
有效的寄存器编号 (有效控制为 True)	选择寄存器编号	寄存器值为 0 时不显示，非 0 时显示
当前颜色	选择线段颜色	/

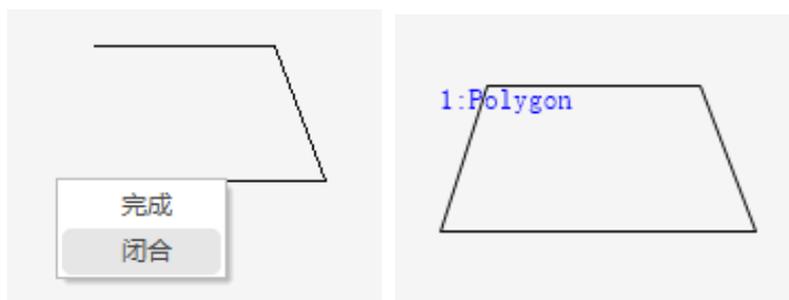
线宽	线段的宽度	/
线段类型	线段的样式	/
水平位置	元件的水平起始位置	不要超出水平分辨率
垂直位置	元件的垂直起始位置	不要超出垂直分辨率
宽度	元件的宽度	/
高度	元件的高度	/

4.3.1.3. 多边形

介绍：由多条连续线段组合成的封闭图形，支持填充背景颜色。“多边形”的形状需要用户手绘。

使用方法：点击菜单栏“HMI”→“控件箱”→“矢量图形”→“线段/多线段/多边形”，单击左键确定第一个点位置开始绘制，当绘制到 n 个点的时候，点击鼠标右键选择“闭合”图形自动封闭。（即：若画一个四边形，则确定 4 个点后，最后一段线段无需手动绘制，选择“闭合”即可）。**提示：**绘制水平直线或垂直直线时，只需按住“Ctrl”键移动鼠标确定方向和长度即可。

注意：形状确定后各点的相对位置不能修改，只支持对整体形状进行缩放。



1. 属性窗口：

属性		线宽	1
基本属性		线段类型	默认
元件编号	1	填充	False
元件名称	Polygon1	位置和尺寸	
显示层次	底层	水平位置	87
有效显示	显示	垂直位置	151
采用有效控制	False	宽度	118
外观		高度	77
当前颜色	000000		

2. 属性说明：

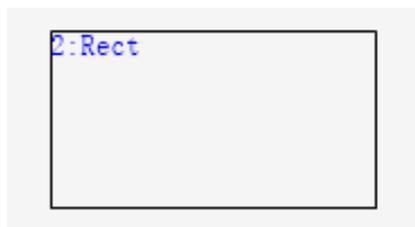
属性	功能	说明
元件编号	/	/

元件名称	/	/
显示层次	选择元件显示层次	顶层：显示在最外层，覆盖底下元件； 中层：被顶层元件覆盖，覆盖底层元件； 底层：被顶层和中间层元件覆盖（默认）。
有效显示	选择元件是否显示	显示：该元件显示于界面之中； 不显示：该元件不显示于界面之中； 仅显示，不可用：该元件可正常显示，但点击该元件时动作不生效
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 True 才会出现下方的三个参数
有效的设备编号 (有效控制为 True)	设备编号	默认 local
有效的寄存器类型 (有效控制为 True)	选择寄存器类型	多种寄存器下拉列表选择
有效的寄存器编号 (有效控制为 True)	选择寄存器编号	寄存器值为 0 时不显示，非 0 时显示
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	/

4.3.2. 矩形

介绍： 创建一个矩形，“矩形”是一个可以填充背景颜色的闭合对象。

使用方法： 点击菜单栏“HMI”→“控件箱”→“矢量图形”→“矩形”，确定单点位置后按住鼠标左键不放，拖拽至合适长度、宽度松开即可。**注意：** 形状确定后各点的相对位置不能修改，只支持对整体形状进行缩放。



1. 属性窗口：

属性		线段类型	默认
基本属性		半径	0
元件编号	4	填充	False
元件名称	Rect4	位置和尺寸	
显示层次	底层	水平位置	590
有效显示	显示	垂直位置	327
采用有效控制	False	宽度	158
外观		高度	117
当前颜色	000000		
线宽	1		

2. 属性说明:

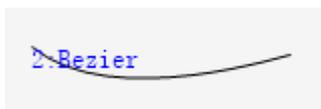
属性	功能	说明
元件编号	/	/
元件名称	/	/
显示层次	选择元件显示层次	顶层: 显示在最外层, 覆盖下层元件; 中层: 被顶层元件覆盖, 覆盖底层元件; 底层: 被顶层和中间层元件覆盖 (默认)。
有效显示	选择元件是否显示	显示: 该元件显示于界面之中; 不显示: 该元件不显示于界面之中; 仅显示, 不可用: 该元件可正常显示, 但点击该元件时动作不生效
采用有效控制	通过寄存器控制元件是否显示	默认 False, 选择 True 才会出现下方的三个参数
有效的设备编号 (有效控制为 True)	设备编号	默认 local
有效的寄存器类型 (有效控制为 True)	选择寄存器类型	多种寄存器下拉列表选择
有效的寄存器编号 (有效控制为 True)	选择寄存器编号	寄存器值为 0 时不显示, 非 0 时显示
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	/
半径	倒角半径	设置四个角是否要倒圆角
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	/

4.3.3. 贝塞尔曲线

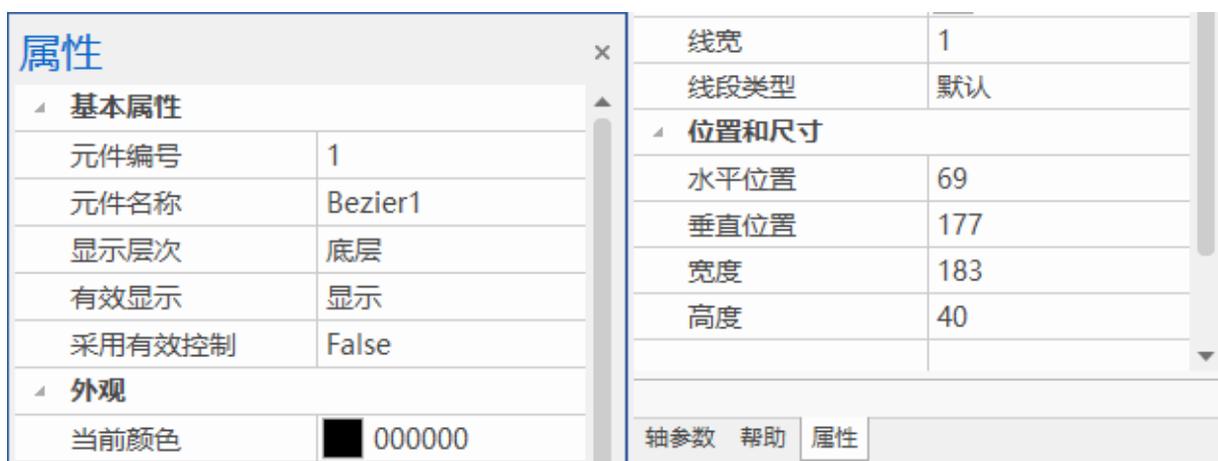
介绍：基于二维平面的矢量数学曲线。

使用方法：点击菜单栏“HMI”→“控件箱”→“矢量图形”→“贝塞尔曲线”，绘制时确定四个点位置即可自动创建贝塞尔曲线。

注意：形状确定后各点的相对位置不能修改，只支持对整体形状进行缩放。



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	顶层：显示在最外层，覆盖底下元件； 中层：被顶层元件覆盖，覆盖底层元件； 底层：被顶层和中间层元件覆盖（默认）。
有效显示	选择元件是否显示	显示：该元件显示于界面之中； 不显示：该元件不显示于界面之中； 仅显示，不可用：该元件可正常显示，但点击该元件时动作不生效
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 True 才会出现下方的三个参数
有效的设备编号 (有效控制为 True)	设备编号	默认 local
有效的寄存器类型 (有效控制为 True)	选择寄存器类型	多种寄存器下拉列表选择
有效的寄存器编号 (有	选择寄存器编号	寄存器值为 0 时不显示，非 0 时显示

效控制为 True)		
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	/

4.3.4.圆/圆弧/扇形

4.3.4.1. 圆

介绍：“圆”是一个可以填充背景颜色的闭合对象。支持对圆/椭圆设置线条类型、颜色及填充颜色和样式等。

使用方法： 点击菜单栏“HMI”→“控件箱”→“矢量图形”→“圆/圆弧/扇形”。鼠标移动至窗口中单击确定图形所在窗口中位置，再按如下方式进行绘制。

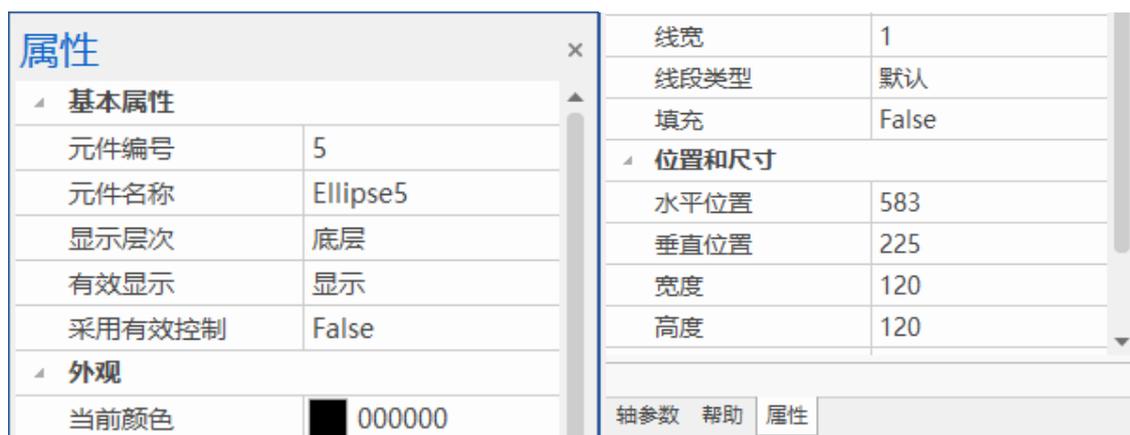
绘制椭圆只需拖动鼠标确认大小后单击鼠标右键点击“完成”（无需另外确定位置点）。

绘制整圆则需在鼠标拖动确认大小时持续按下 Ctrl 键，确认大小后松开鼠标和按键，接着单击鼠标右键点击“完成”（无需另外确定位置点）。

注意：形状确定后各点的相对位置不能修改，只支持对整体形状进行缩放。由于圆/椭圆的弧线分布着无数点，此时调整宽高可改变形状（如：绘制整圆后调整大小即可变换成椭圆）



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	顶层：显示在最外层，覆盖底下元件； 中层：被顶层元件覆盖，覆盖底层元件； 底层：被顶层和中间层元件覆盖（默认）。
有效显示	选择元件是否显示	显示：该元件显示于界面之中； 不显示：该元件不显示于界面之中； 仅显示，不可用：该元件可正常显示，但点击该元件时动作不生效
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 True 才会出现下方的三个参数
有效的设备编号 (有效控制为 True)	设备编号	默认 local
有效的寄存器类型 (有效控制为 True)	选择寄存器类型	多种寄存器下拉列表选择
有效的寄存器编号 (有效控制为 True)	选择寄存器编号	寄存器值为 0 时不显示，非 0 时显示
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	/

4.3.4.2. 圆弧

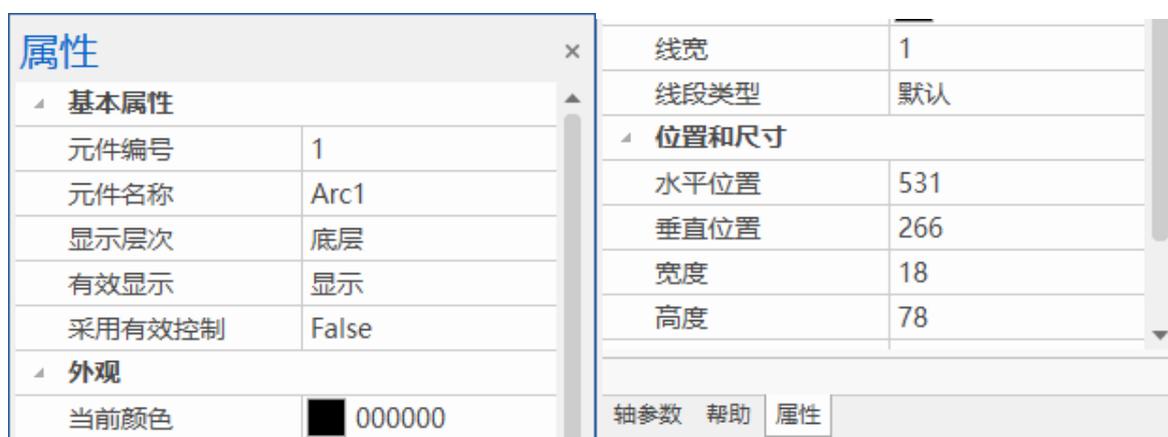
介绍：圆上任意两点间的曲线即圆弧。

使用方法：点击菜单栏“HMI”→“控件箱”→“矢量图形”→“圆/圆弧/扇形”。先拉一个矩形框确定大小，再确定圆弧的起点和终点，按起点到终点顺时针方向绘制圆弧，绘制完成后点击“完成”即可绘制成功。

注意：形状确定后各点的相对位置不能修改，只支持对整体形状进行缩放。



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	顶层：显示在最外层，覆盖底下元件； 中层：被顶层元件覆盖，覆盖底层元件； 底层：被顶层和中间层元件覆盖（默认）。
有效显示	选择元件是否显示	显示：该元件显示于界面之中； 不显示：该元件不显示于界面之中； 仅显示，不可用：该元件显示于界面之中但是点击该元件时不会产生任何改变。
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 True 才会出现下方的三个参数
有效的设备编号 (有效控制为 True)	设备编号	默认 local
有效的寄存器类型 (有效控制为 True)	选择寄存器类型	多种寄存器下拉列表选择

有效的寄存器编号（有效控制为 True）	选择寄存器编号	寄存器值为 0 时不显示，非 0 时显示
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	/

4.3.4.3. 扇形

介绍：圆弧与两条半径组成的闭合对象。

使用方法：点击菜单栏“HMI”→“控件箱”→“矢量图形”→“圆/圆弧/扇形”。先拉一个矩形框确定大小，再确定扇形的起点和终点，确定两点位置后右键点击“扇形”即可按顺时针方向生成扇形区域。

注意：形状确定后各点的相对位置不能修改，只支持对整体形状进行缩放。



1. 属性窗口：

属性		线段类型	默认
基本属性		填充	False
元件编号	1	位置和尺寸	
元件名称	Sector1	水平位置	171
显示层次	底层	垂直位置	160
有效显示	显示	宽度	57
采用有效控制	False	高度	79
外观			
当前颜色	000000		
线宽	1		

2. 属性说明：

属性	功能	说明
元件编号	/	/

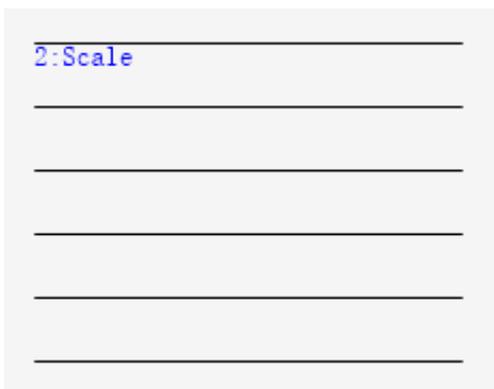
元件名称	/	/
显示层次	选择元件显示层次	顶层：显示在最外层，覆盖底下元件； 中层：被顶层元件覆盖，覆盖底层元件； 底层：被顶层和中间层元件覆盖（默认）。
有效显示	选择元件是否显示	显示：该元件显示于界面之中； 不显示：该元件不显示于界面之中； 仅显示，不可用：该元件显示于界面之中但是点击该元件时不会产生任何改变。
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 True 才会出现下方的三个参数
有效的设备编号 (有效控制为 True)	设备编号	默认 local
有效的寄存器类型 (有效控制为 True)	选择寄存器类型	多种寄存器下拉列表选择
有效的寄存器编号 (有效控制为 True)	选择寄存器编号	寄存器值为 0 时不显示，非 0 时显示
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	/

4.3.5. 刻度

介绍：绘出多行的等高线条，用作元件对齐等辅助标记。

使用方法：点击菜单栏“HMI”→“控件箱”→“矢量图形”→“刻度”。鼠标移动至窗口合适位置单击放置即可。支持调整整体大小及段落数，同时支持切换线条方向（可在“属性”窗口修改）。

注意：行间距不支持手动修改，可通过对该元件进行缩放或改变元件宽度和高度实现改变行间距的效果。



1. 属性窗口:

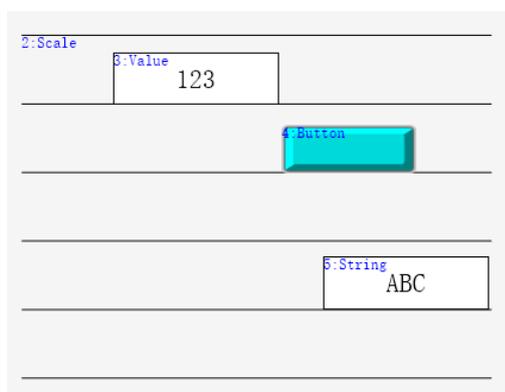
属性		线段类型	默认
基本属性		刻度样式	水平方向
元件编号	1	内部段数	5
元件名称	Scale1	位置和尺寸	
显示层次	底层	水平位置	26
有效显示	显示	垂直位置	70
采用有效控制	False	宽度	201
外观		高度	151
当前颜色	000000		
线宽	1		

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	顶层：显示在最外层，覆盖底下元件； 中层：被顶层元件覆盖，覆盖底层元件； 底层：被顶层和中间层元件覆盖（默认）。
有效显示	选择元件是否显示	显示：该元件显示于界面之中； 不显示：该元件不显示于界面之中； 仅显示，不可用：该元件可正常显示，但点击该元件时动作不生效
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 True 才会出现下方的三个参数
有效的设备编号 (有效控制为 True)	设备编号	默认 local
有效的寄存器类型 (有效控制为 True)	选择寄存器类型	多种寄存器下拉列表选择
有效的寄存器编号 (有	选择寄存器编号	寄存器值为 0 时不显示，非 0 时显示

效控制为 True)		
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	/

例：将元件放置于刻度线条之间，起到对齐布局的作用。



4.3.6.表格

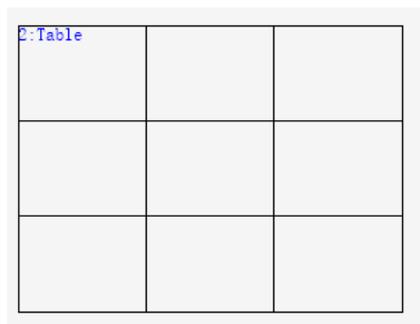
介绍：绘制一个网格型表项，可将元件放置于表格中，主要用作元件对齐布局。

使用方法：点击菜单栏“HMI”→“控件箱”→“矢量图形”→“表格”。鼠标移动至窗口合适位置单击放置即可。

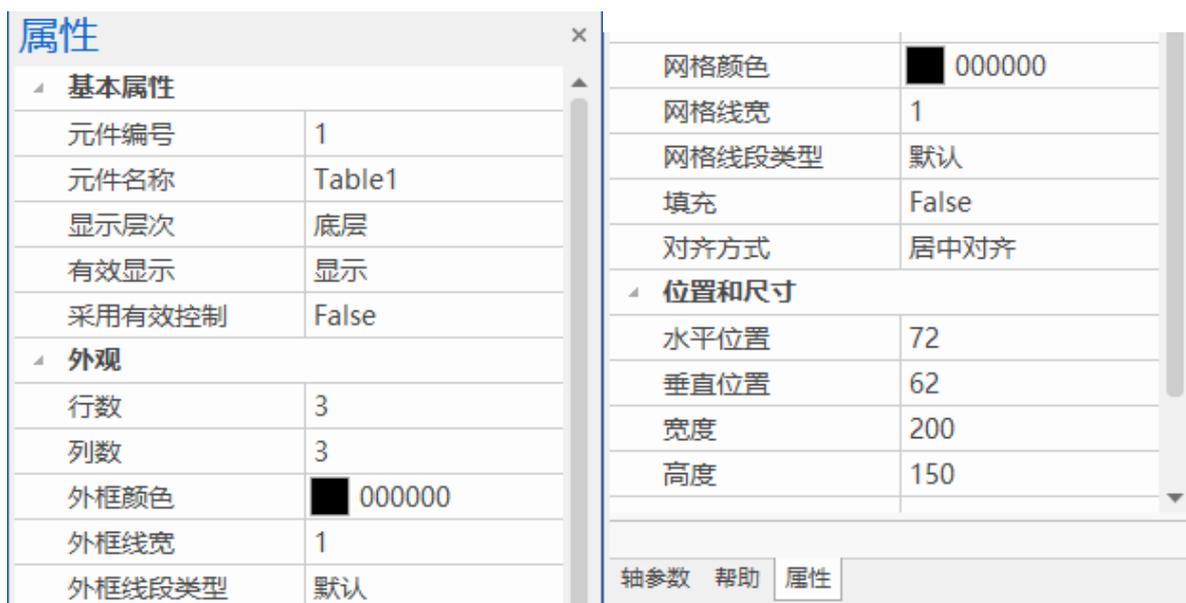
[元件放置方法]：将元件拖到表格处自动吸附对齐，对齐方式可在属性中修改。

注意：

1. 一个表格只能设置一种对齐方式。
2. 需布局放置的元件大小必须小于表格内部单格的大小，才有自动对齐吸附效果。
3. 使用时需要注意元件的显示层次的设置。



1. 属性窗口：

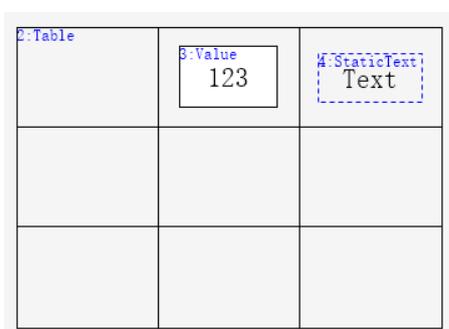


2. 属性说明：

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	顶层：显示在最外层，覆盖底下元件； 中层：被顶层元件覆盖，覆盖底层元件； 底层：被顶层和中间层元件覆盖（默认）。
有效显示	选择元件是否显示	显示：该元件显示于界面之中； 不显示：该元件不显示于界面之中； 仅显示，不可用：该元件可正常显示，但点击该元件时动作不生效
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 True 才会出现下方的三个参数
有效的设备编号 (有效控制为 True)	设备编号	默认 local
有效的寄存器类型 (有效控制为 True)	选择寄存器类型	多种寄存器下拉列表选择
有效的寄存器编号 (有效控制为 True)	选择寄存器编号	寄存器值为 0 时不显示，非 0 时显示
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	默认黑色
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	设置表格内框线颜色	默认黑色
网格线宽	设置表格内框线线宽	/

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	9种对齐方式：左上对齐/中上对齐/右上对齐/左中对齐/居中对齐/右中对齐/左下对齐/中下对齐/右下对齐
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	/

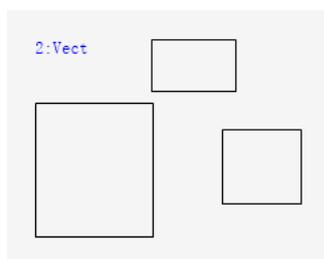
例：将元件调整至合适大小后，直接拖拽到表格内部中，会有自动吸附对齐的效果，通过设置对齐方式使得元件在表格内部按对应方式摆放。



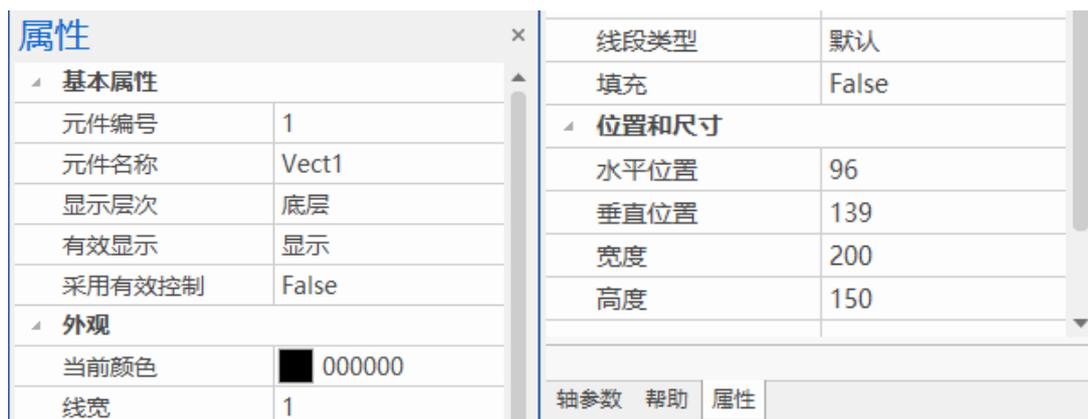
4.3.7. 导入

介绍：导入矢量图形。

使用方法：点击菜单栏“HMI”→“控件箱”→“矢量图形”→“导入”。在建立时先确定方框的位置，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。（目前支持导入的图形格式为：.dxf/.ai/.plt/.dst/）。支持修改矢量图形的颜色、线宽、背景颜色等。



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	顶层：显示在最外层，覆盖底下元件； 中层：被顶层元件覆盖，覆盖底层元件； 底层：被顶层和中间层元件覆盖（默认）。
有效显示	选择元件是否显示	显示：该元件显示于界面之中； 不显示：该元件不显示于界面之中； 仅显示，不可用：该元件可正常显示，但点击该元件时动作不生效
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 True 才会出现下方的三个参数
有效的设备编号 (有效控制为 True)	设备编号	默认 local
有效的寄存器类型 (有效控制为 True)	选择寄存器类型	多种寄存器下拉列表选择
有效的寄存器编号 (有效控制为 True)	选择寄存器编号	寄存器值为 0 时不显示，非 0 时显示
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	/

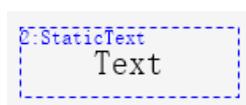
4.3.8. 静态文本

介绍：输入并显示文本内容。支持显示单行或多行文字

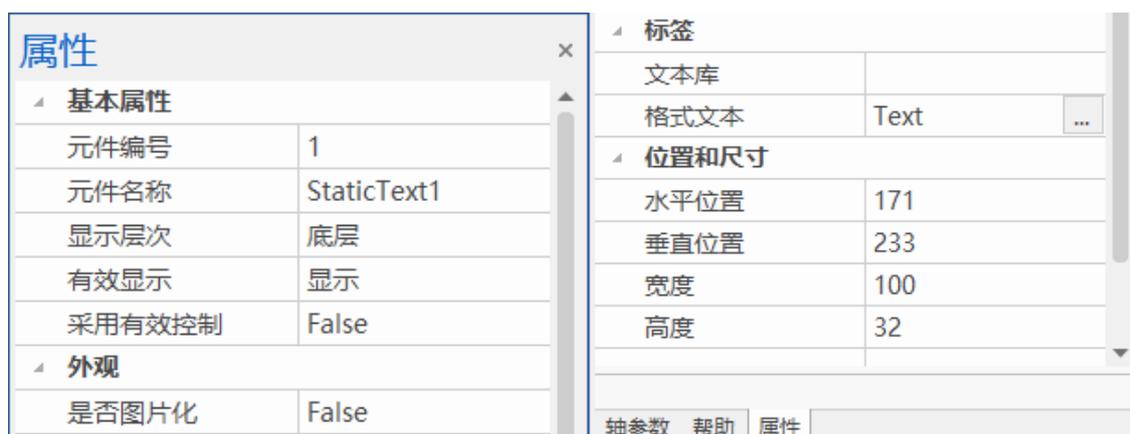
使用方法：点击菜单栏“HMI”→“控件箱”→“静态文本”。选中后将矩形框放至合适位置，直接输入文本即可。

注意：

1. **元件直接输入文本不支持直接换行。**文本换行须在“属性”→“格式文本”的“内容”编辑处进行换行。格式文本使用具体可参见上述“[格式文本](#)”章节。
2. **文本较多的时候，注意调整元件尺寸，元件太小可能导致显示不全。**



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	顶层：显示在最外层，覆盖底下元件； 中层：被顶层元件覆盖，覆盖底层元件； 底层：被顶层和中间层元件覆盖（默认）。
有效显示	选择元件是否显示	显示：该元件显示于界面之中； 不显示：该元件不显示于界面之中； 仅显示，不可用：该元件可正常显示，但点击该元件时动作不生效
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 True 才会出现下方的三个参数

有效的设备编号 (有效控制为 True)	设备编号	默认 local
有效的寄存器类型 (有效控制为 True)	选择寄存器类型	多种寄存器下拉列表选择
有效的寄存器编号 (有效控制为 True)	选择寄存器编号	寄存器值为 0 时不显示, 非 0 时显示
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	/

4.3.9.静态图片

介绍: 导入并显示静态图片。

使用方法: 点击菜单栏“HMI”→“控件箱”→“静态图片”。在窗口中添加“静态图片”元件, 在元件“属性”中添加要显示的图片, 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。

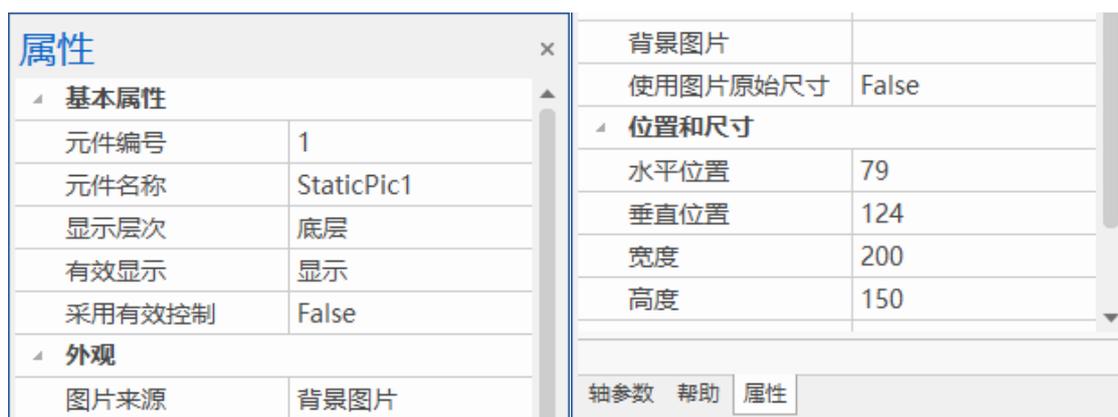
背景图片库: 图片先添加到图片库中, 然后在背景图片库中选择。

背景图片: 图片先添加到项目的“文件视图”中, 再在背景图片中选择图片。



在使用视觉的时候, 该元件支持显示视觉拍摄的图片, 在“背景图片”中选择视觉通道, 支持四个视觉通道: @ZV0、@ZV1、@ZV2、@ZV3, 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	顶层: 显示在最外层, 覆盖底下元件; 中层: 被顶层元件覆盖, 覆盖底层元件; 底层: 被顶层和中间层元件覆盖 (默认)。
有效显示	选择元件是否显示	显示: 该元件显示于界面之中; 不显示: 该元件不显示于界面之中; 仅显示, 不可用: 该元件可正常显示, 但点击该元件时动作不生效
采用有效控制	通过寄存器控制元件是否显示	默认 False, 选择 True 才会出现下方的三个参数
有效的设备编号 (有效控制为 True)	设备编号	默认 local
有效的寄存器类型 (有效控制为 True)	选择寄存器类型	多种寄存器下拉列表选择
有效的寄存器编号 (有效控制为 True)	选择寄存器编号	寄存器值为 0 时不显示, 非 0 时显示
图片来源	无、背景图片库或背景图片	无图片、背景图片库或背景图片中选择
背景图片	背景图片选择	图片来源设置为选择背景图片显示
背景图片库	背景图片库选择	图片来源设置为选择背景图片库显示
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. 先将图片存放至控制器或仿真器的 flash 文件夹中。

2. 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

3. 新建“静态图片”元件，属性的图片来源选择背景图片，再手动输入@ZV0 即可读取该图片。

外观	
图片来源	背景图片
背景图片	@ZV0
使用图片原始尺寸	False

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。



4.3.10. 位状态显示

介绍: 根据寄存器的位状态 0 或 1 来显示格式文本 0 或 1。

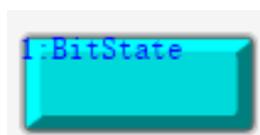
使用方法: 点击菜单栏“HMI”→“控件箱”→“位状态显示”。将元件放至合适位置，在该元件“属性”窗口选择需绑定的寄存器类型及地址。并在不同状态的格式文本输入显示内容。切换显示状态需通过切换寄存器状态实现。

切换寄存器状态的方式有三种:

- 点击菜单栏“HMI”→“语言/状态”→“S0/S1/S2...”即可实现状态切换;
- 在[命令与输出]窗口的“在线命令”行,输入绑定的寄存器地址及状态值,点击“发送”即可(如输入: MODBUS_BIT(0)=1);
- 通过“动作”设置调用子函数, Basic 子函数写入相关寄存器状态切换的程序,按下元件调用子函数程序可实现切换。

注意: 若无设置相关动作,则无法直接按下元件来切换显示状态。

提示：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。若选择的寄存器不是位寄存器，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，多个位的显示使用多状态。



1. 属性窗口：

属性		是否图片化	False
基本属性		标签	
元件编号	3	文本库	
元件名称	BitState3	格式文本(0)	...
显示层次	底层	格式文本(1)	...
有效显示	显示	动作	
采用有效控制	False	点击调用函数	
安全时间ms	0	位置和尺寸	
绑定的设备编号	本地	水平位置	252
绑定的寄存器类型	M	垂直位置	186
绑定的寄存器编号	0	宽度	109
外观		高度	49
图片来源	背景图片库		
背景图片库	0\按钮\9 ...		
绘制边框	False		

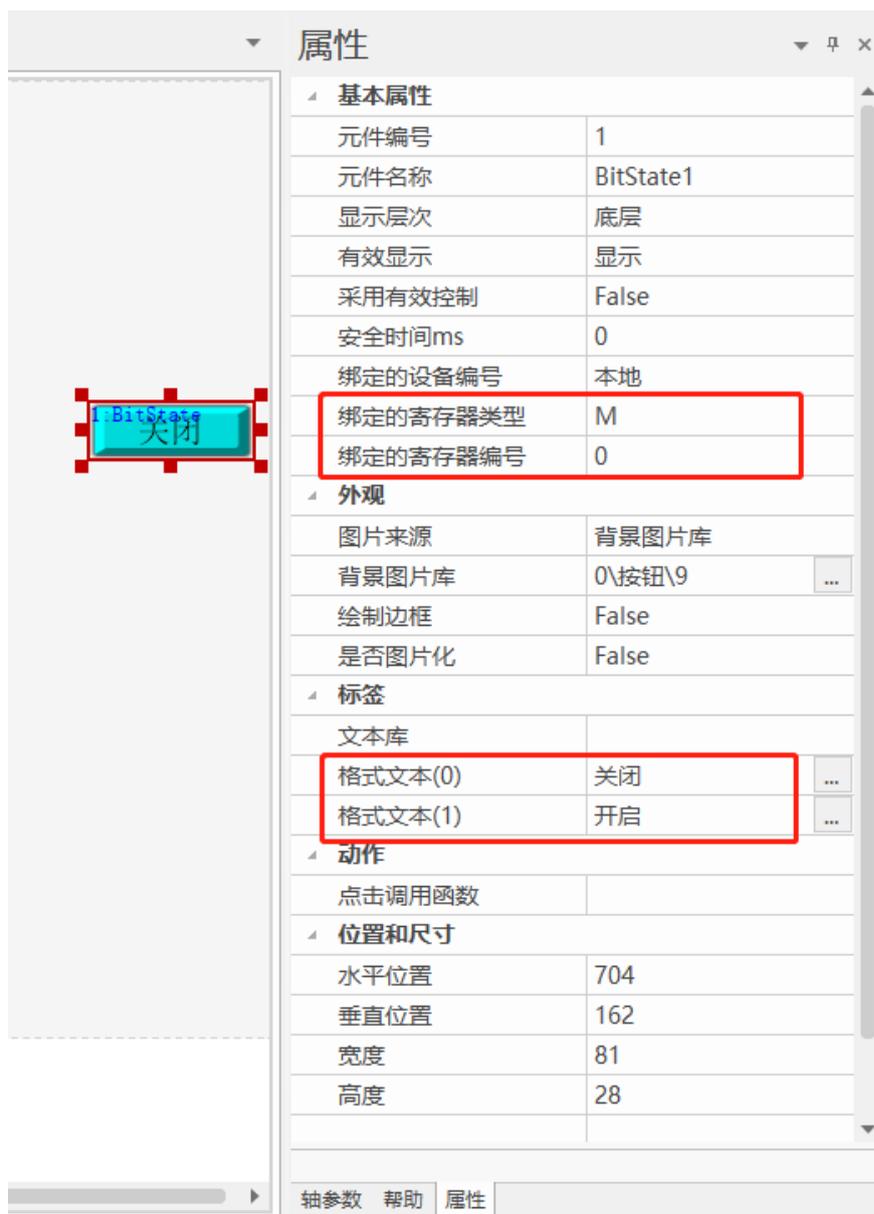
2. 属性说明：

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	顶层：显示在最外层，覆盖底下元件； 中层：被顶层元件覆盖，覆盖底层元件； 底层：被顶层和中间层元件覆盖（默认）。
有效显示	选择元件是否显示	显示：该元件显示于界面之中； 不显示：该元件不显示于界面之中； 仅显示，不可用：该元件可正常显示，但点击该元件时动作不生效
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 True 才会出现下方的三个参数，通过寄存器控制元件是否显示
有效的设备编号 (有效控制为 True)	设备编号	默认 local

有效的寄存器类型（有效控制为 True）	选择寄存器类型	多种寄存器下拉列表选择
有效的寄存器编号（有效控制为 True）	选择寄存器编号	寄存器值为 0 时不显示，非 0 时显示
安全时间 ms	最少按键时间	单位 ms
绑定的设备编号	指定设备	默认 local
绑定的寄存器类型	选择寄存器类型	多种寄存器下拉列表选择
绑定的寄存器编号	设置寄存器的编号	通过获取寄存器不同数值，控制元件不同状态
图片来源	无、背景图片库或背景图片	无图片、背景图片库或背景图片中选择
背景图片	背景图片选择	图片来源设置为选择背景图片显示
背景图片库	背景图片库选择	图片来源设置为选择背景图片库显示
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. 选择绑定的寄存器类型和编号，M0 对应 MODBUS_BIT(0)；
2. 在格式文本中填入不同状态对应的文本。



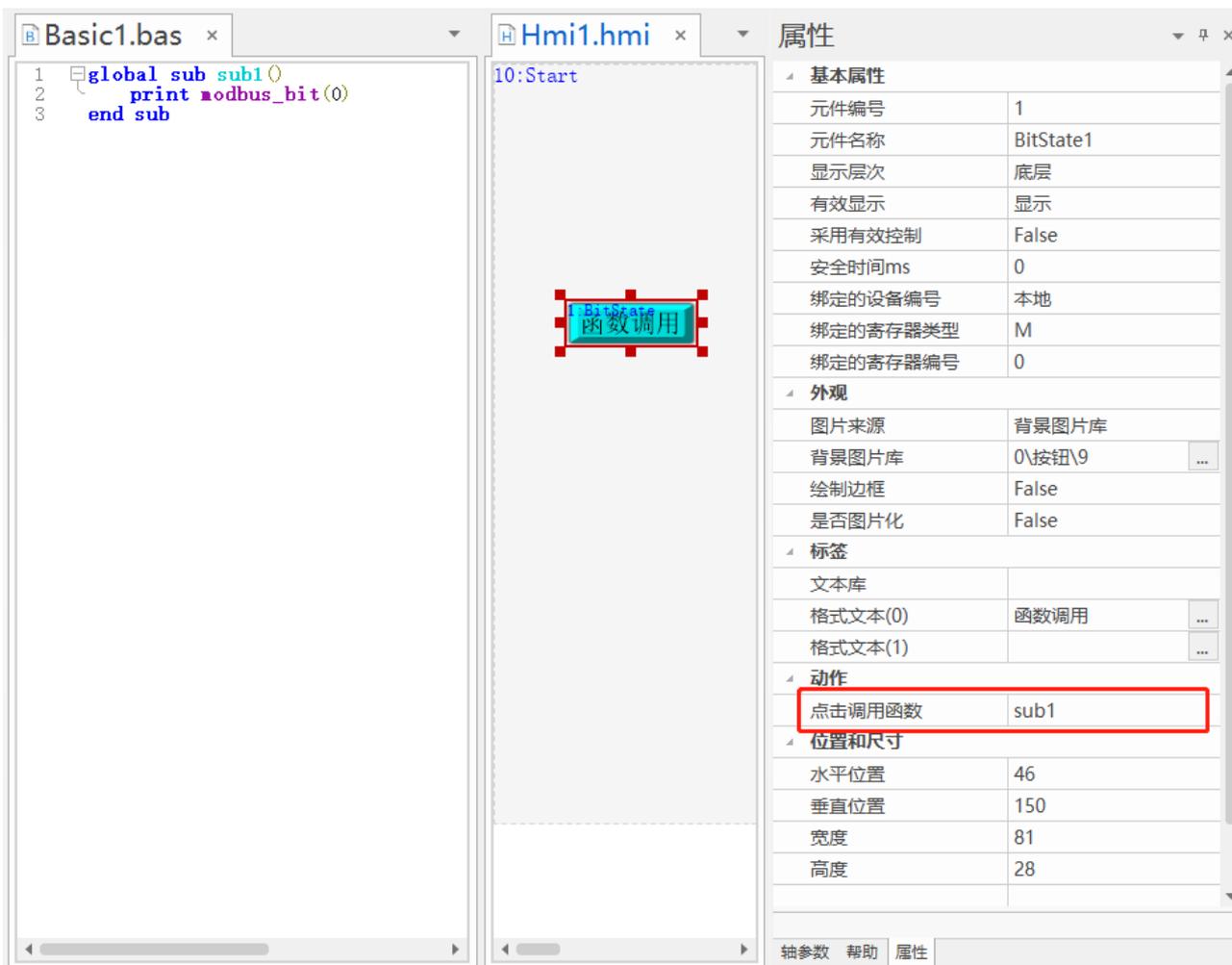
实现效果:

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

提示: 切换寄存器状态方法可参考本节上方内容。

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。

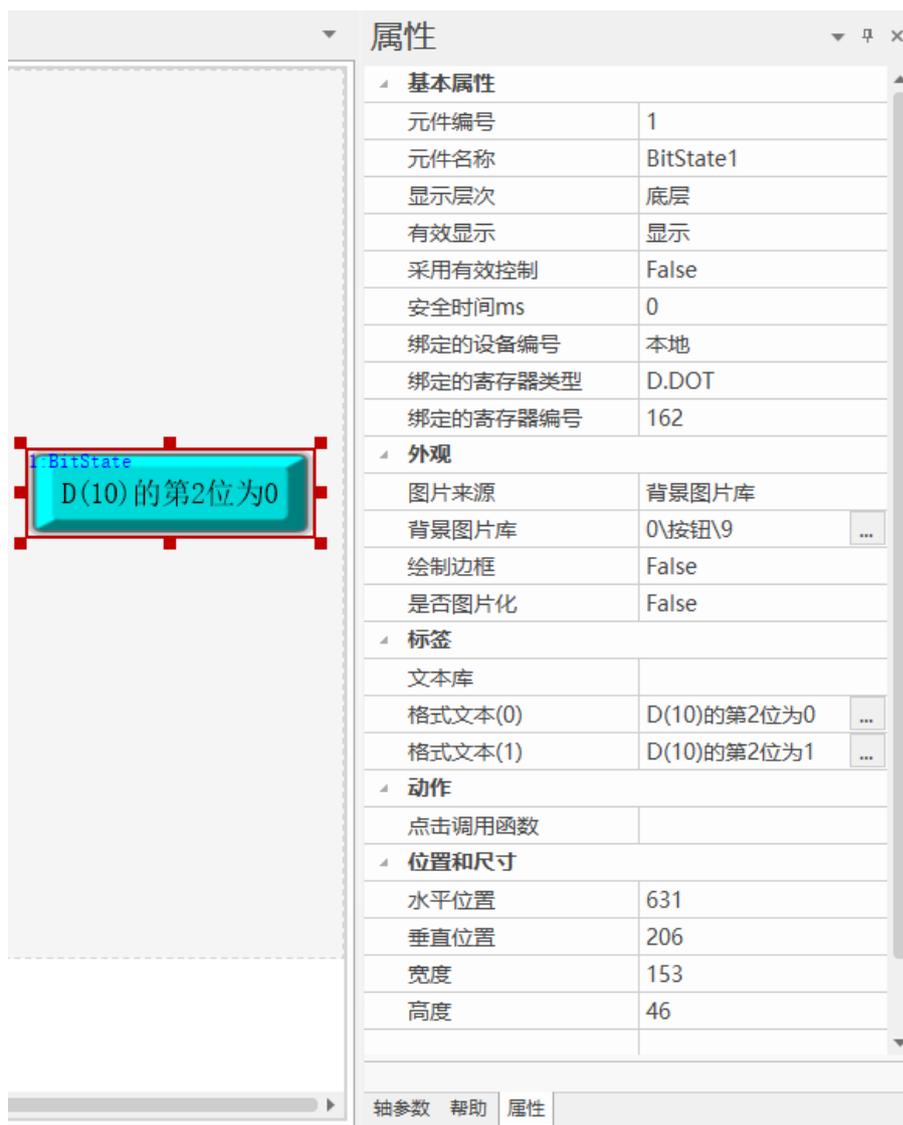


实现效果:

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（即可设置 MODBUS_REG 寄存器具体某 bit 位的状态值）；
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”。

4.3.11. 多状态显示

介绍：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。

使用方法：点击菜单栏“HMI”→“控件箱”→“多状态显示”。将元件放至合适位置，在该元件“属性”窗口选择需绑定的寄存器类型及地址。并在不同状态的格式文本输入显示内容。切换显示状态需通过切换寄存器状态实现。

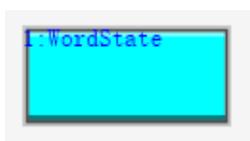
切换寄存器状态的方式有三种：

- 点击菜单栏“HMI”→“语言/状态”→“S0/S1/S2...”即可实现状态切换；

- 在[命令与输出]窗口的“在线命令”行，输入绑定的寄存器地址及状态值，点击“发送”即可（如输入：MODBUS_REG(0)=1）；
- 通过“动作”设置调用子函数，Basic 子函数写入相关寄存器状态切换的程序，按下元件调用子函数程序可实现切换。

注意：

1. 若无设置相关动作，则无法直接按下元件来切换显示状态。
2. 可显示的文本数量根据“状态数量”自定义设置，数量范围 1-256。
3. 建议选择寄存器控制，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：

属性			
基本属性		是否图片化	False
元件编号	4	标签	
元件名称	WordState4	文本库	
显示层次	底层	状态数量	2
有效显示	显示	格式文本(0)	...
采用有效控制	False	格式文本(1)	...
安全时间ms	0	动作	
绑定的设备编号	本地	点击调用函数	
绑定的寄存器类型	D	位置和尺寸	
绑定的寄存器编号	0	水平位置	589
外观		垂直位置	300
图片来源	背景图片库	宽度	109
背景图片库	0\按钮\13 ...	高度	53
绘制边框	False		

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	顶层：显示在最外层，覆盖底下元件； 中层：被顶层元件覆盖，覆盖底层元件；

		底层：被顶层和中间层元件覆盖（默认）。
有效显示	选择元件是否显示	显示：该元件显示于界面之中； 不显示：该元件不显示于界面之中； 仅显示，不可用：该元件可正常显示，但点击该元件时动作不生效
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 True 才会显示以下三个参数，通过寄存器控制元件是否显示
有效的设备编号 (有效控制为 True)	设备编号	默认 local
有效的寄存器类型 (有效控制为 True)	选择寄存器类型	多种寄存器下拉列表选择
有效的寄存器编号 (有效控制为 True)	选择寄存器编号	寄存器值为 0 时不显示，非 0 时显示
安全时间 ms	最少按键时间	单位 ms
绑定的设备编号	指定设备	默认 local
绑定的寄存器类型	选择寄存器类型	多种寄存器下拉列表选择
绑定的寄存器编号	设置寄存器的编号	通过获取寄存器不同数值，控制元件不同状态
图片来源	无、背景图片库或背景图片	无图片、背景图片库或背景图片中选择
背景图片	背景图片选择	图片来源设置为选择背景图片显示
背景图片库	背景图片库选择	图片来源设置为选择背景图片库显示
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(1-256)	可显示多个状态
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	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”；

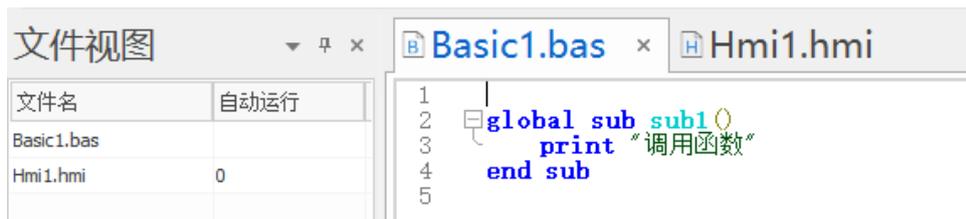
当 MODBUS_REG(0)=1 时，元件显示“手动”；

当 MODBUS_REG(0)=2 时，元件显示“待机”。

提示： 切换寄存器状态方法可参考本节上方内容。

例二：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:

下载到控制器/仿真器运行后，在触摸屏/仿真界面按下元件时，执行 SUB 函数内打印代码。

4.3.12. 位状态设置

介绍: 根据元件动作的状态设置位寄存器地址的值，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。

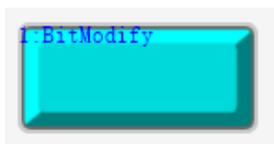
使用方法： 点击菜单栏“HMI”→“控件箱”→“位状态设置”。将元件放至合适位置，在该元件“属性”窗口选择需绑定的寄存器类型及地址，在不同状态的格式文本输入显示内容。并设置元件的“动作”类型。根据元件的动作类型对元件进行操作，实现对位寄存器值的设置。（支持调用函数，可选择元件按下或松开时调用。）

查看寄存器值的方法：

- 点击菜单栏“工具”→“寄存器”，选择绑定的寄存器类型及地址，点击“读取”即可查看。
- 在[命令与输出]窗口的“在线命令”行，输入：print+绑定的寄存器地址及状态值，点击“发送”即可（如输入：print MODBUS_BIT(0) / ?MODBUS_BIT(0)）；

注意：

1. 不能通过寄存器的值来切换显示状态。
2. 元件显示状态不一定与寄存器值状态对应。



1. 属性窗口：

属性		是否图片化		False
基本属性		标签		
元件编号	5	文本库		
元件名称	BitModify5	格式文本(0)		...
显示层次	底层	格式文本(1)		...
有效显示	显示	动作		
采用有效控制	False	动作		无动作
安全时间ms	0	松开时动作		False
绑定的设备编号	本地	位置和尺寸		
绑定的寄存器类型	M	水平位置		619
绑定的寄存器编号	0	垂直位置		212
外观		宽度		109
图片来源	背景图片库	高度		49
背景图片库	0\按钮\9 ...			
绘制边框	False			

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	顶层：显示在最外层，覆盖底下元件； 中层：被顶层元件覆盖，覆盖底层元件； 底层：被顶层和中间层元件覆盖（默认）。
有效显示	选择元件是否显示	显示：该元件显示于界面之中； 不显示：该元件不显示于界面之中； 仅显示，不可用：该元件可正常显示，但点击该元件时动作不生效
采用有效控制	通过寄存器控制元件是否显示	默认 False ，选择 True 通过寄存器控制元件是否显示
有效的设备编号 (有效控制为 True)	设备编号	默认 local
有效的寄存器类型 (有效控制为 True)	选择寄存器类型	多种寄存器下拉列表选择
有效的寄存器编号 (有效控制为 True)	设置寄存器的编号	寄存器值为 0 时不显示，非 0 时显示
安全时间 ms	最少按键时间	单位 ms
绑定的设备编号	指定设备	默认 local
绑定的寄存器类型	选择寄存器类型	多种寄存器下拉列表选择
绑定的寄存器编号	设置寄存器的编号	通过获取寄存器不同数值，控制元件不同状态
图片来源	无、背景图片库或背景图片	无图片、背景图片库或背景图片中选择
背景图片	背景图片选择	图片来源设置为选择背景图片显示
背景图片库	背景图片库选择	图片来源设置为选择背景图片库显示
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， True 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：给寄存器赋值 1

1. 选择寄存器类型和编号；
2. “动作”选择“状态设置为 1”。



实现效果：

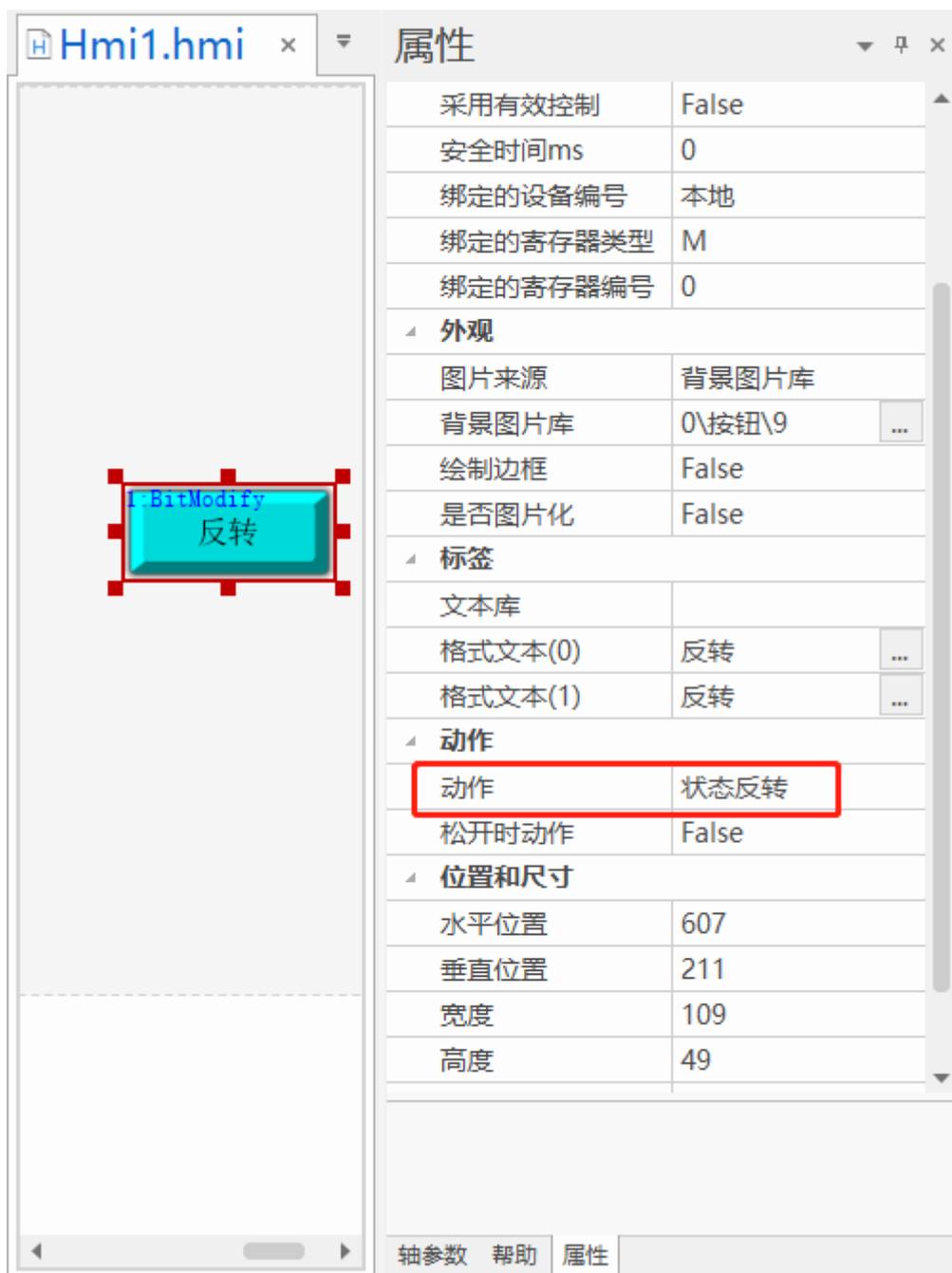
按下元件后，将寄存器的值置为 1（即 MODBUS_BIT(0)=1），此时元件显示状态为状态 1。松开后元件显示状态为 0，但寄存器值仍为 1。

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“状态设置为 0”则将寄存器置 0，与“状态设置为 1”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“状态反转”。

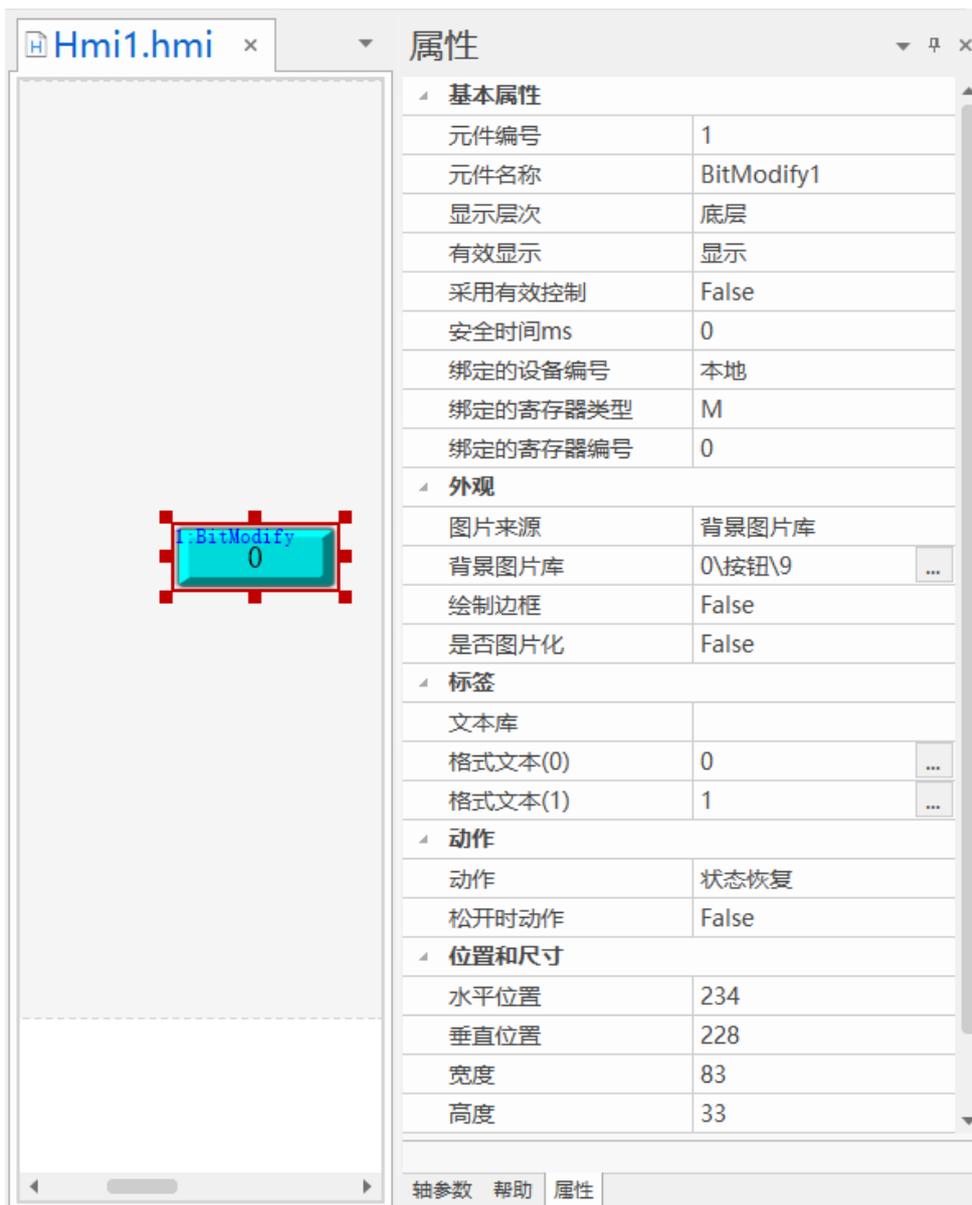


实现效果:

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“状态恢复”。



实现效果:

按下元件时，位寄存器 MODBUS_BIT(0)=1；松开元件后，位寄存器 MODBUS_BIT(0)=0。

例四：调用 SUB 函数

例程参见[功能键例一](#)。

4.3.13. 多状态设置

介绍: 根据元件动作的状态设置字寄存器地址的值，同时支持调用函数。详细使用方法参见下方例子。

使用方法: 点击菜单栏“HMI”→“控件箱”→“多状态设置”。将元件放至合适位置，在该元件“属性”窗口

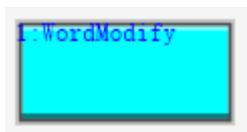
选择需绑定的寄存器类型及地址，在不同状态的格式文本输入显示内容。并设置元件的“动作”类型。根据元件的动作类型对元件进行操作，实现对寄存器值的设置。（支持调用函数，可选择元件按下或松开时调用。）

查看寄存器值的方法：

- 点击菜单栏“工具”→“寄存器”，选择绑定的寄存器类型及地址，点击“读取”即可查看。
- 在[命令与输出]窗口的“在线命令”行，输入：print+绑定的寄存器地址及状态值，点击“发送”即可（如输入：print MODBUS_BIT(0) / ?MODBUS_BIT(0)）；

注意：

1. 不能通过寄存器的值来切换显示状态。
2. 元件显示状态不一定与寄存器值状态对应。



1. 属性窗口：

属性	
基本属性	
元件编号	6
元件名称	WordModify6
显示层次	底层
有效显示	显示
采用有效控制	False
安全时间ms	0
绑定的设备编号	本地
绑定的寄存器类型	D
绑定的寄存器编号	0
外观	
图片来源	背景图片库
背景图片库	0\按钮\13 ...
绘制边框	False
是否图片化	False

标签	
文本库	
状态数量	2
格式文本(0)	...
格式文本(1)	...
动作	
动作	无动作
松开时动作	False
位置和尺寸	
水平位置	663
垂直位置	118
宽度	109
高度	53

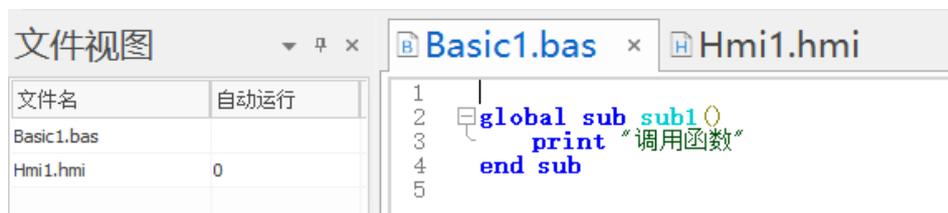
2. 属性说明：

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	顶层：显示在最外层，覆盖底下元件； 中层：被顶层元件覆盖，覆盖底层元件； 底层：被顶层和中间层元件覆盖（默认）。
有效显示	选择元件是否显示	显示：该元件显示于界面之中； 不显示：该元件不显示于界面之中； 仅显示，不可用：该元件可正常显示，但点击该元件时动作不生效
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 True 才会显示以下三个参数，通过寄存器控制元件是否显示
有效的设备编号 (有效控制为 True)	设备编号	默认 local
有效的寄存器类型 (有效控制为 True)	选择寄存器类型	多种寄存器下拉列表选择
有效的寄存器编号 (有效控制为 True)	选择寄存器编号	寄存器值为 0 时不显示，非 0 时显示
安全时间 ms	最少按键时间	单位 ms
绑定的设备编号	指定设备	默认 local
绑定的寄存器类型	选择寄存器类型	多种寄存器下拉列表选择
绑定的寄存器编号	设置寄存器的编号	通过获取寄存器不同数值，控制元件不同状态
图片来源	无、背景图片库或背景图片	无图片、背景图片库或背景图片中选择
背景图片	背景图片选择	图片来源设置为选择背景图片显示
背景图片库	背景图片库选择	图片来源设置为选择背景图片库显示
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(1-256)	可显示多个状态
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	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，True 为松开时动作
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
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“调用函数”，“动作函数名”选择对应的 SUB 函数名。

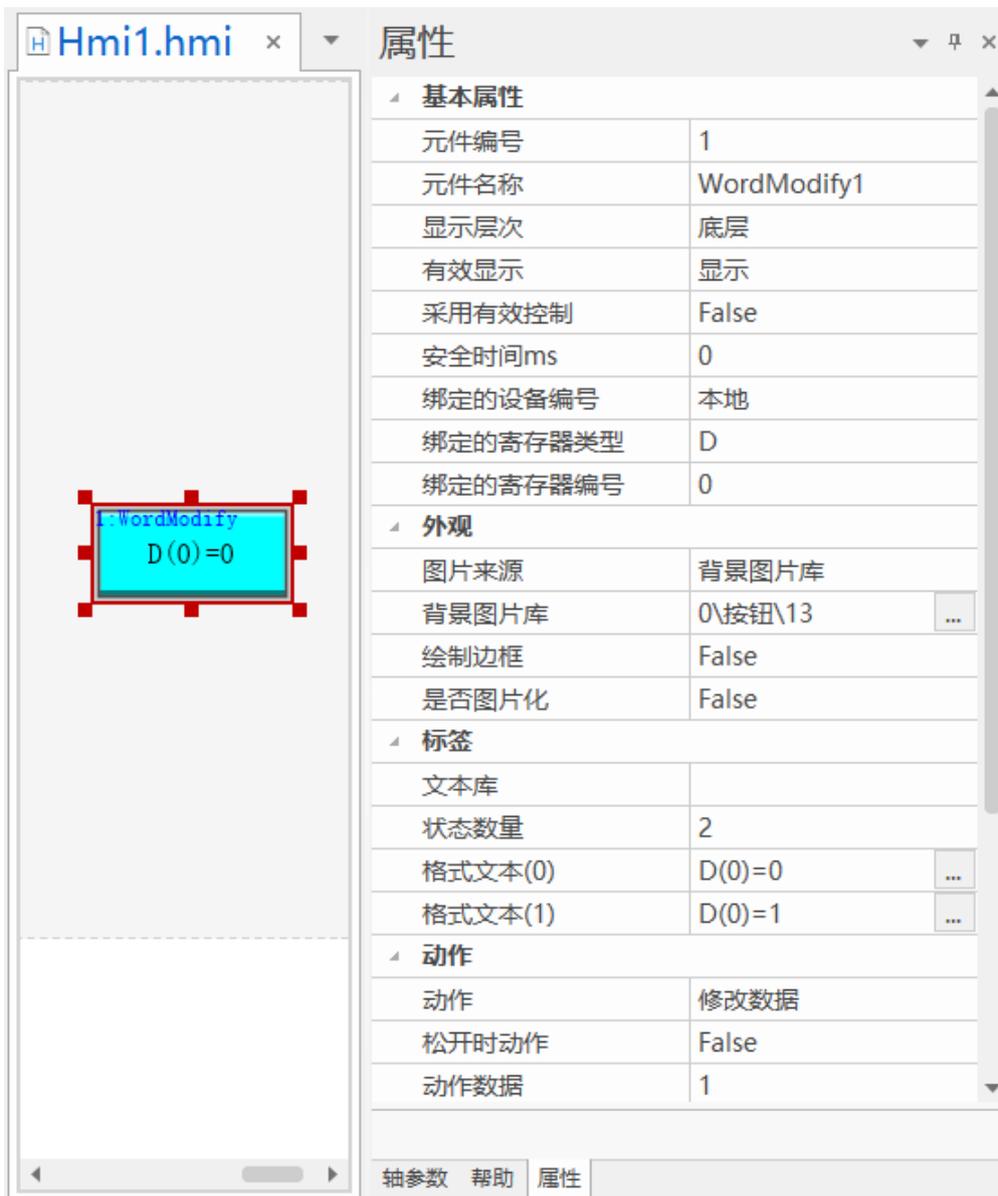


实现效果：按下元件时，执行 SUB 函数内打印代码。

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“修改数据”，设置“动作数据”写入寄存器的值，如例子设置“动作数据”为 1；
状态支持显示两种，按下时显示“格式文本 1”，松开显示“格式文本 0”。

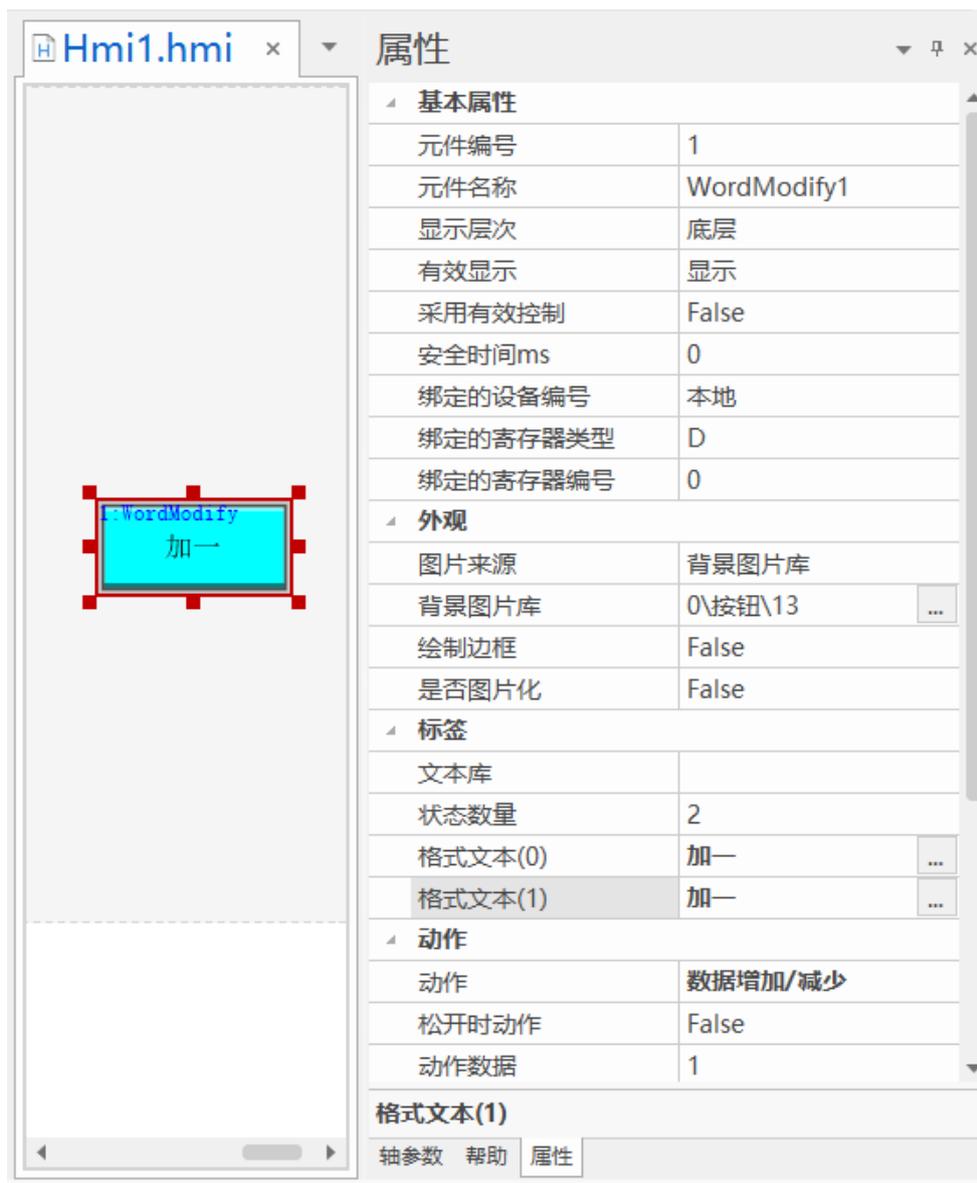


实现效果:

按下元件后，寄存器 MODBUS_REG(0)写入数据 1。若“动作数据”设置为其他数值，则寄存器写入对应数值。

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“数据增加/减少”，“动作数据”填入寄存器每次要增加/减少的数据。

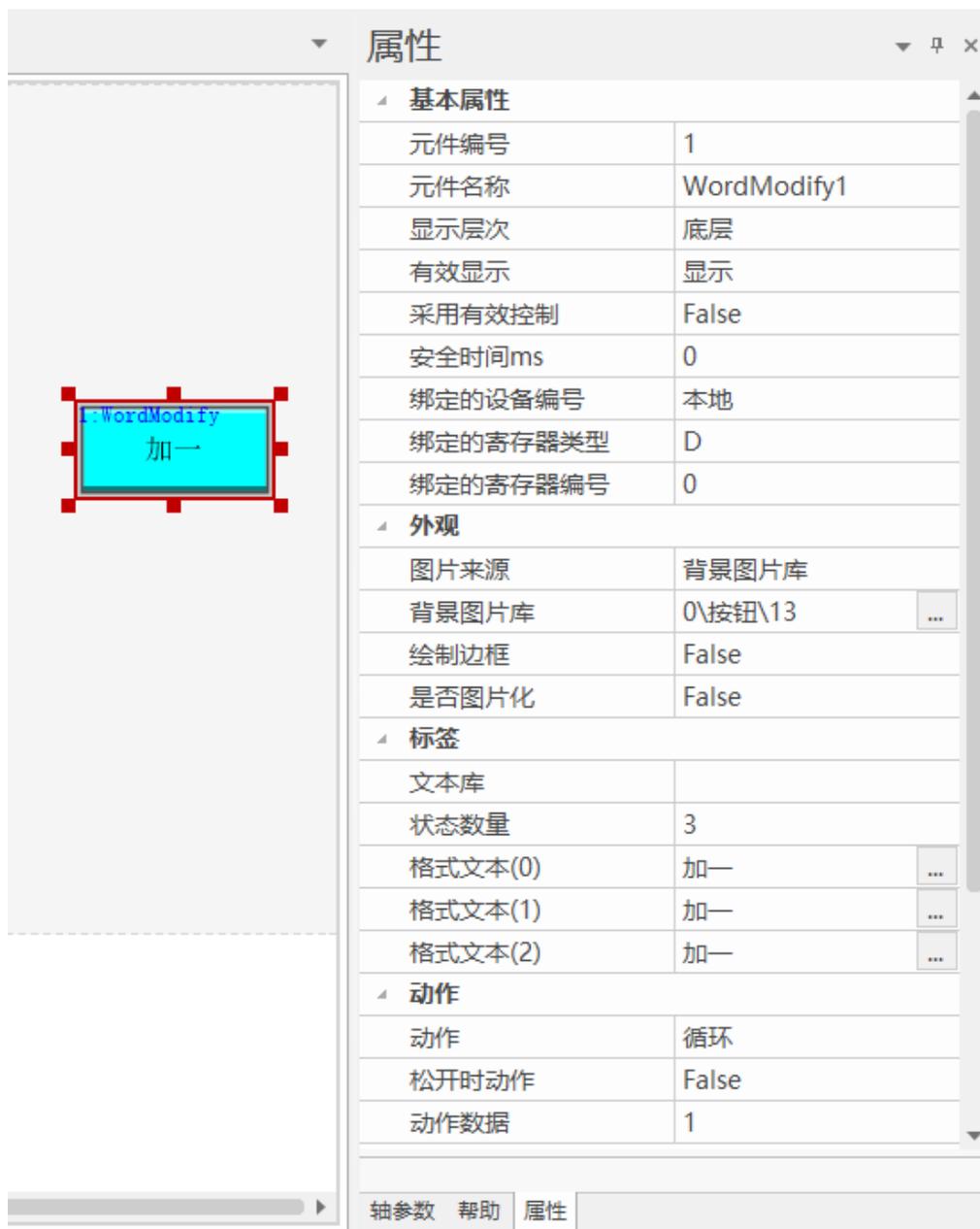


实现效果：

每按下元件一次后，MODBUS_REG(0)等于原来的值加 1。

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“循环”，“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, 按一下即可自动计算并递减至设定的状态数量范围内, 再开始在 0 到 2 之间切换。

寄存器的值根据状态的数量循环, 例如:

- 状态数量为 3, 动作数据为 1 时, 对应的寄存器值在 0、1、2 之间切换;
- 状态数量为 5, 动作数据为 2 时, 对应的寄存器值在 0、2、4、1、3 之间切换。

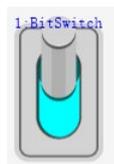
4.3.14. 位状态切换开关

介绍：根据元件动作设置位寄存器地址的值并显示对应状态。根据动作设置寄存器的值使得显示状态改变。寄存器值为 0 或 1 时则对应显示格式文本 0 或 1。

使用方法：点击菜单栏“HMI”→“控件箱”→“位状态切换开关”。将元件放至合适位置，在该元件“属性”窗口选择需绑定的寄存器类型及地址，在不同状态的格式文本输入显示内容。并设置元件的“动作”类型。根据元件的动作类型对元件进行操作，实现对位寄存器值的设置及显示状态切换。（支持调用函数，可选择元件按下或松开时调用。）

注意：

1. 当无设置动作时，无法直接通过按下元件来切换显示状态，需手动修改寄存器值进行切换。
2. 选择位寄存器时，寄存器值为 0 时显示格式文本 0；寄存器值为 1 时显示格式文本 1。如果选择的寄存器不是位寄存器，那么寄存器值为 0 时显示格式文本 0；寄存器值不为 1 时显示格式文本 1，多个位的显示使用多状态元件。



1. 属性窗口：

属性		是否图片化	False
基本属性		标签	
元件编号	7	文本库	
元件名称	BitSwitch7	格式文本(0)	...
显示层次	底层	格式文本(1)	...
有效显示	显示	动作	
采用有效控制	False	动作	无动作
安全时间ms	0	松开时动作	False
绑定的设备编号	本地	位置和尺寸	
绑定的寄存器类型	M	水平位置	702
绑定的寄存器编号	0	垂直位置	351
外观		宽度	63
图片来源	背景图片库	高度	109
背景图片库	0\开关\18 ...		
绘制边框	False		

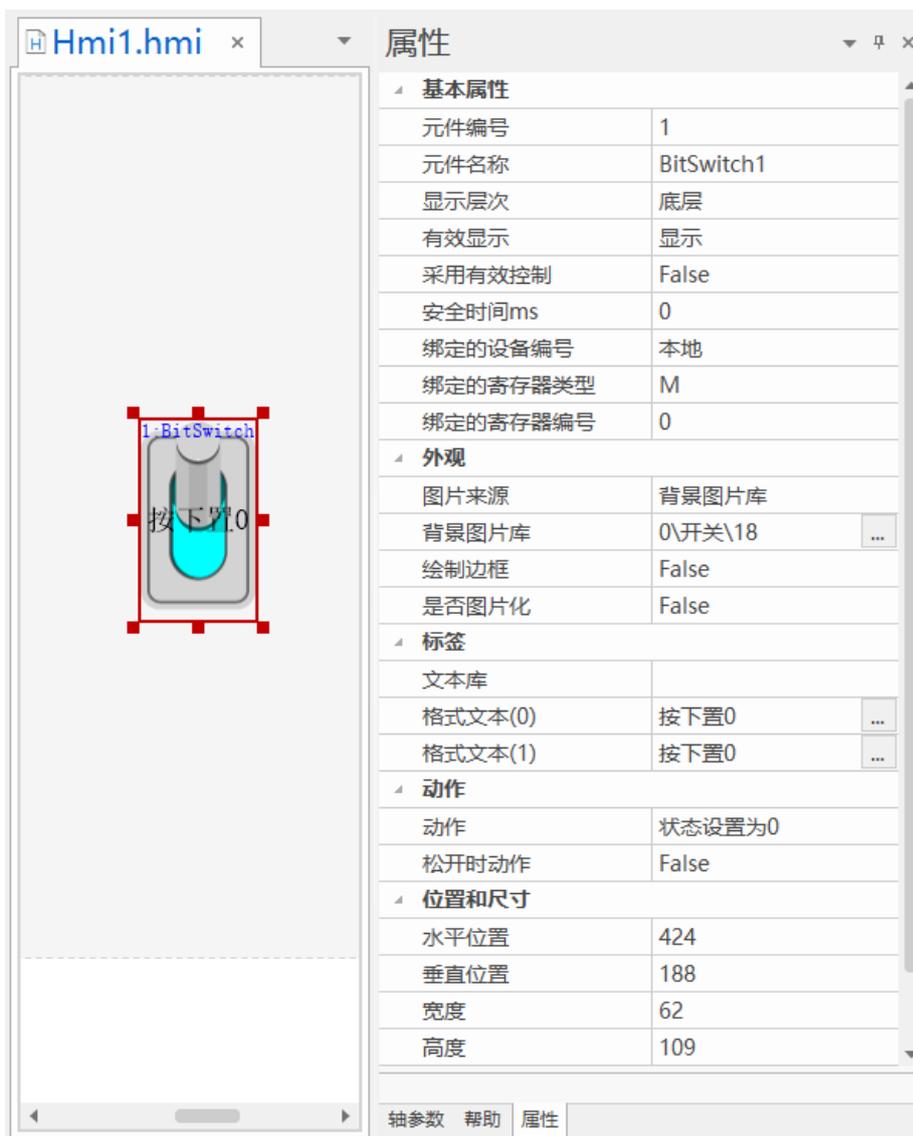
2. 属性说明：

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	顶层：显示在最外层，覆盖底下元件； 中层：被顶层元件覆盖，覆盖底层元件； 底层：被顶层和中间层元件覆盖（默认）。
有效显示	选择元件是否显示	显示：该元件显示于界面之中； 不显示：该元件不显示于界面之中； 仅显示，不可用：该元件可正常显示，但点击该元件时动作不生效
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 True 才会显示以下三个参数，通过寄存器控制元件是否显示
有效的设备编号 (有效控制为 True)	设备编号	默认 local
有效的寄存器类型 (有效控制为 True)	选择寄存器类型	多种寄存器下拉列表选择
有效的寄存器编号 (有效控制为 True)	选择寄存器编号	寄存器值为 0 时不显示，非 0 时显示
安全时间 ms	最少按键时间	单位 ms
绑定的设备编号	指定设备	默认 local
绑定的寄存器类型	选择寄存器类型	多种寄存器下拉列表选择
绑定的寄存器编号	设置寄存器的编号	通过获取寄存器不同数值，控制元件不同状态
图片来源	无、背景图片库或背景图片	无图片、背景图片库或背景图片中选择
背景图片	背景图片选择	图片来源设置为选择背景图片显示
背景图片库	背景图片库选择	图片来源设置为选择背景图片库显示
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，True 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：给寄存器赋值 0

1. 选择寄存器类型和编号；
2. “动作”选择“状态设置为 0”。



实现效果:

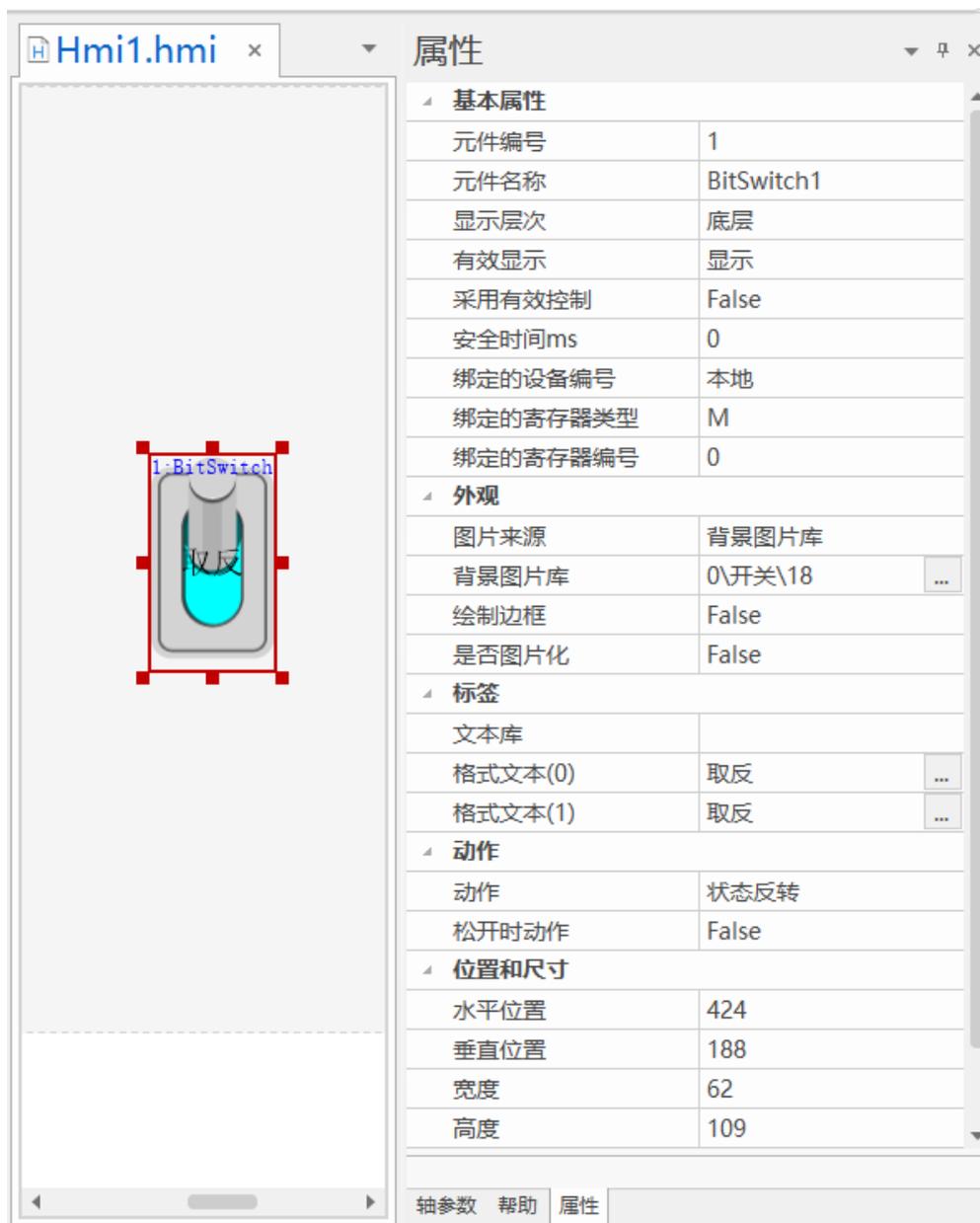
按下元件时, 位寄存器 MODBUS_BIT(0)=0, 元件显示状态为 1 状态; 松开后 MODBUS_BIT(0)仍为 0, 同时元件显示状态切换为 0。

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“状态设置为 1”为将寄存器置 1, 与“状态设置为 0”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“状态反转”。



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。即可实现同时切换元件和寄存器的状态值。

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“状态恢复”。



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](#)。

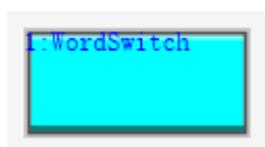
4.3.15. 多状态切换开关

介绍：根据元件动作设置字寄存器地址的值并显示对应状态。根据动作设置寄存器的值使得显示状态改变。寄存器值为 1 或 2 时则对应显示格式文本 1 或 2。

使用方法： 点击菜单栏“HMI”→“控件箱”→“多状态切换开关”。将元件放至合适位置，在该元件“属性”窗口选择需绑定的寄存器类型及地址，在不同状态的格式文本输入显示内容。并设置元件的“动作”类型。根据元件的动作类型对元件进行操作，实现对寄存器值的设置及显示状态切换。（支持调用函数，可选择元件按下或松开时调用。）

注意：

1. 当无设置动作时，无法直接通过按下元件来切换显示状态，需手动修改寄存器值进行切换。
2. 可显示的文本数量根据“状态数量”自定义设置，数量范围 1-256。
3. 建议选择寄存器控制，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：

属性	
基本属性	
元件编号	8
元件名称	WordSwitch8
显示层次	底层
有效显示	显示
采用有效控制	False
安全时间ms	0
绑定的设备编号	本地
绑定的寄存器类型	D
绑定的寄存器编号	0
外观	
图片来源	背景图片库
背景图片库	0\按钮\13 ...
绘制边框	False
是否图片化	False
标签	
文本库	
状态数量	2
格式文本(0)	...
格式文本(1)	...
动作	
动作	无动作
松开时动作	False
位置和尺寸	
水平位置	475
垂直位置	411
宽度	109
高度	53

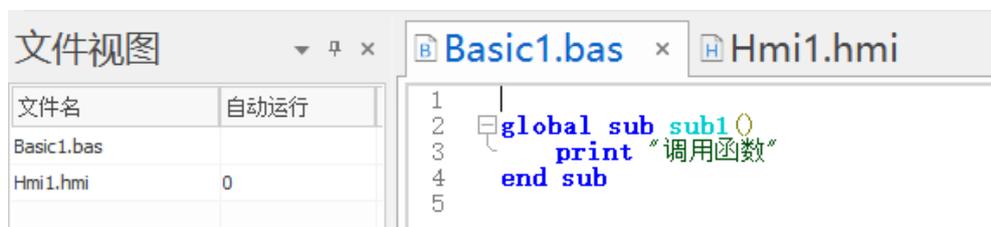
2. 属性说明：

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	顶层：显示在最外层，覆盖底下元件；

		中层：被顶层元件覆盖，覆盖底层元件； 底层：被顶层和中间层元件覆盖（默认）。
有效显示	选择元件是否显示	显示：该元件显示于界面之中； 不显示：该元件不显示于界面之中； 仅显示，不可用：该元件可正常显示，但点击该元件时动作不生效
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 True 才会显示以下三个参数，通过寄存器控制元件是否显示
有效的设备编号 (有效控制为 True)	设备编号	默认 local
有效的寄存器类型 (有效控制为 True)	选择寄存器类型	多种寄存器下拉列表选择
有效的寄存器编号 (有效控制为 True)	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
绑定的设备编号	指定设备	默认 local
绑定的寄存器类型	选择寄存器类型	多种寄存器下拉列表选择
绑定的寄存器编号	设置寄存器的编号	通过获取寄存器不同数值，控制元件不同状态
图片来源	无、背景图片库或背景图片	无图片、背景图片库或背景图片中选择
背景图片	背景图片选择	图片来源设置为选择背景图片显示
背景图片库	背景图片库选择	图片来源设置为选择背景图片库显示
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(1-256)	可显示多个状态
格式文本 1/0	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，True 为松开时动作
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
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“调用函数”，“动作函数名”选择对应的 SUB 函数名。

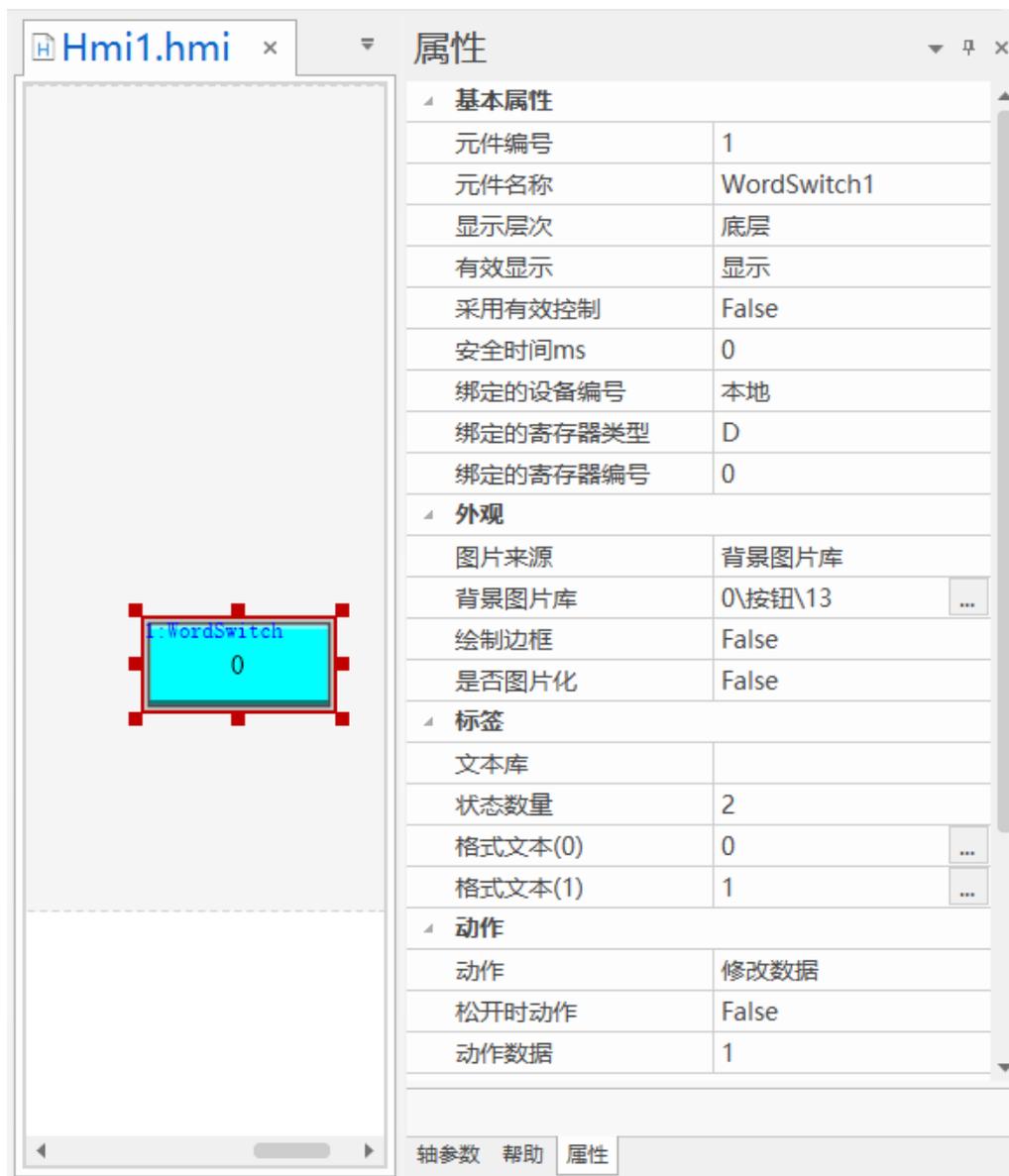


实现效果：

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“修改数据”，设置“动作数据”写入寄存器的值。

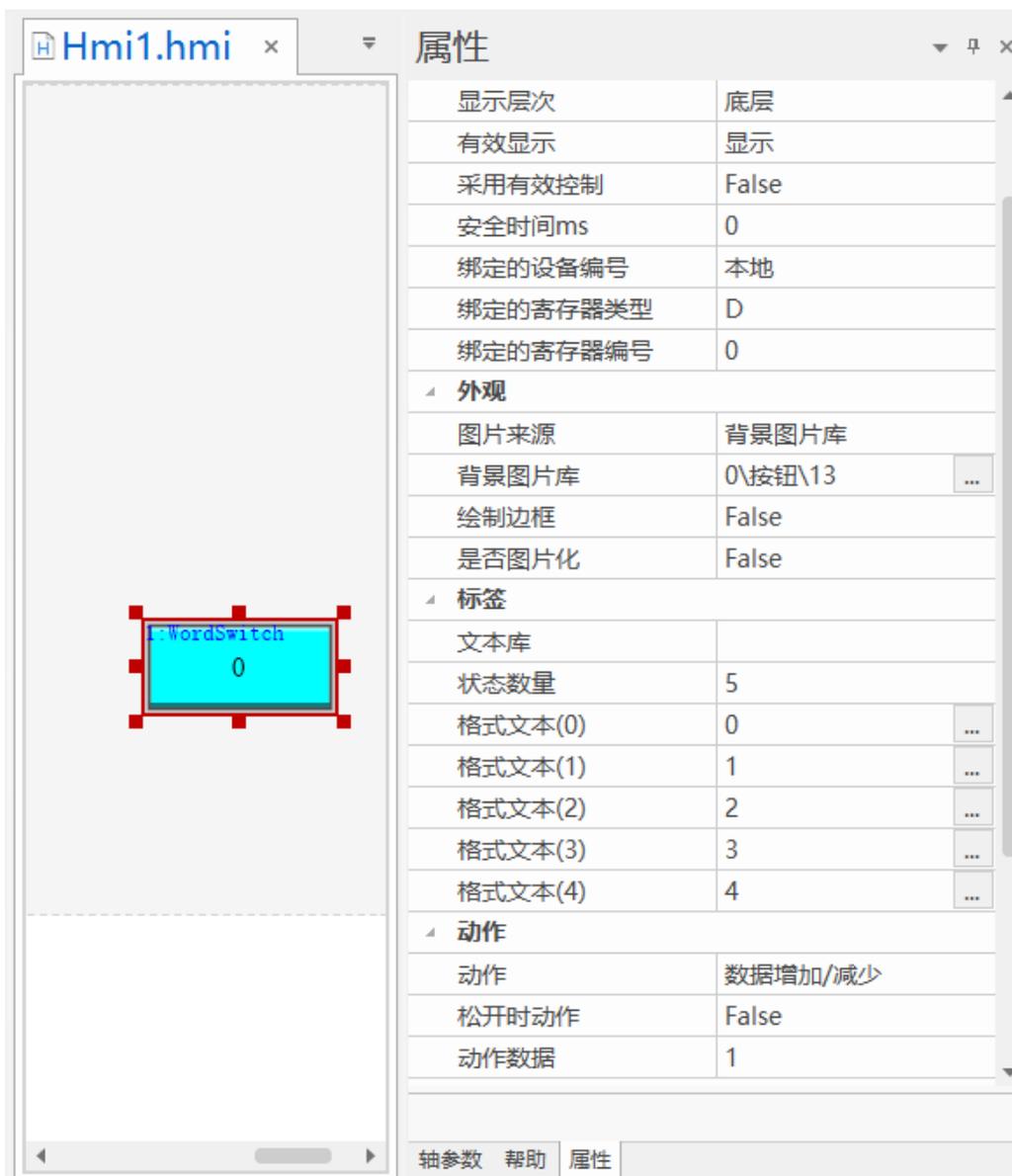


实现效果：

按下元件后，将动作数据值写入绑定寄存器，即 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“数据增加/减少”，并在“动作数据”输入数值。



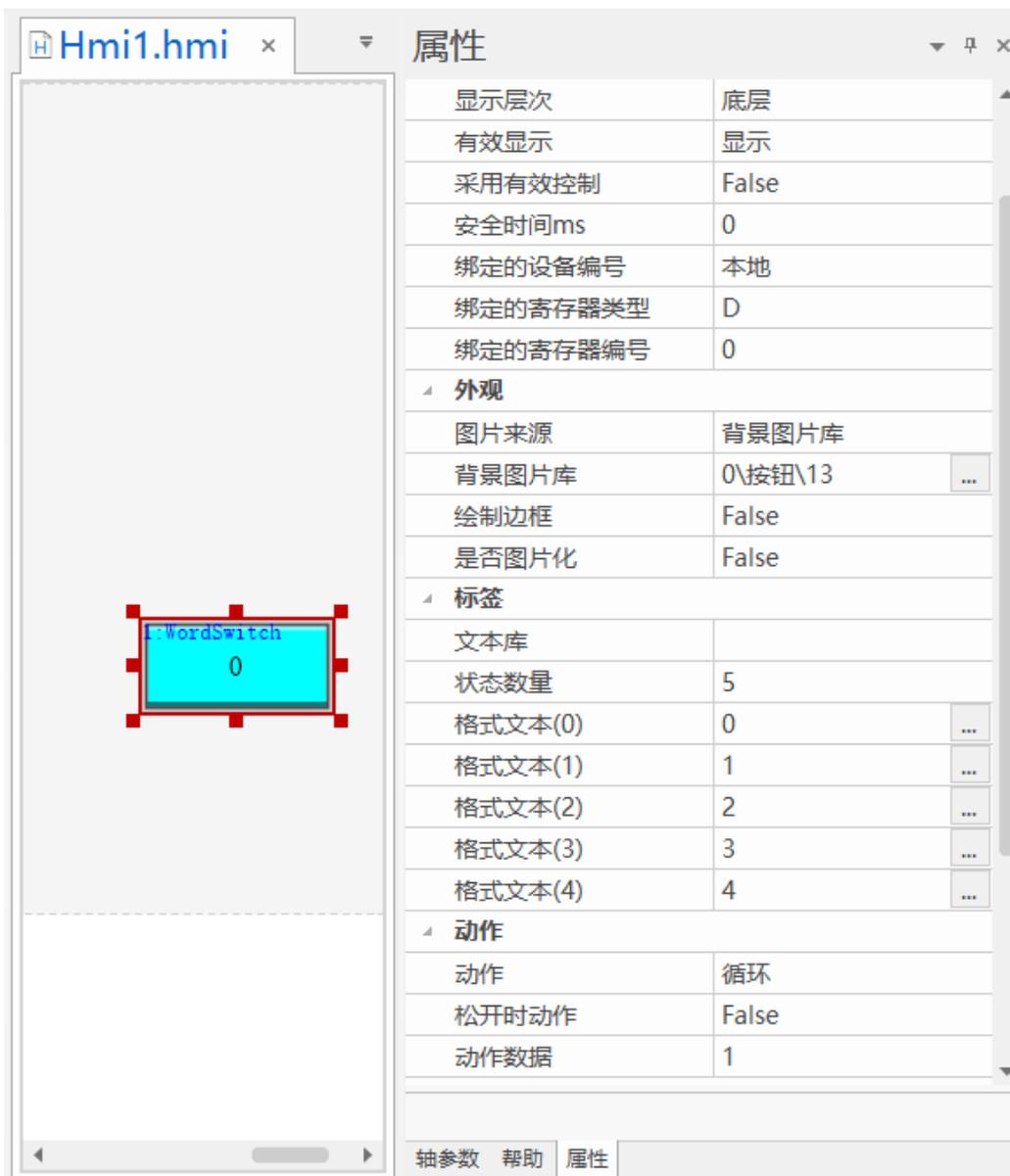
实现效果:

每按下元件一次后, MODBUS_REG(0)等于原来的值加 1。

注意: 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“循环”, “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 到 4 之间切换。

如果 MODBUS_REG(0)的初始值大于 2 时, 按一下即可自动计算并递减至设定的状态数量范围内, 再开始在 0 到 2 之间切换。

寄存器的值根据状态的数量循环, 例如:

- 状态数量 3, 动作数据 1, 对应的寄存器值在 0、1、2 之间切换;
- 状态数量 5, 动作数据 2, 对应的寄存器值在 0、2、4、1、3 之间切换。

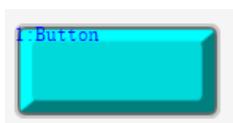
4.3.16. 功能键

介绍：根据元件动作实现状态切换/窗口切换/软键盘切换等。只有两种显示状态，初始默认显示格式文本 0，按下时显示格式文本 1，相当于一个“开关按钮”。

使用方法：点击菜单栏“HMI”→“控件箱”→“功能键”。将元件放至合适位置，在该元件“属性”窗口的格式文本输入显示内容。并选择元件的“动作”类型。根据元件的动作类型对元件进行操作，实现相应的效果。

功能键的主要功能包括：调用 Basic 函数、打开/关闭窗口、按键/字符输入、切换软键盘。通过“属性”窗口中的“动作”来选择功能。

注意：功能键不支持绑定寄存器。



1. 属性窗口：

属性	
基本属性	
元件编号	9
元件名称	Button9
显示层次	底层
有效显示	显示
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	背景图片库
背景图片库	0\按钮\9 ...
绘制边框	False
是否图片化	False

属性	
标签	
文本库	
格式文本(0)	...
格式文本(1)	...
动作	
动作	调用函数
松开时动作	False
动作函数名	
位置和尺寸	
水平位置	246
垂直位置	371
宽度	109
高度	49

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	顶层：显示在最外层，覆盖底下元件； 中层：被顶层元件覆盖，覆盖底层元件； 底层：被顶层和中间层元件覆盖（默认）。
有效显示	选择元件是否显示	显示：该元件显示于界面之中；

		不显示：该元件不显示于界面之中； 仅显示，不可用：该元件可正常显示，但点击该元件时动作不生效
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 True 才会显示以下三个参数，通过寄存器控制元件是否显示
有效的设备编号 (有效控制为 True)	设备编号	默认 local
有效的寄存器类型 (有效控制为 True)	选择寄存器类型	多种寄存器下拉列表选择
有效的寄存器编号 (有效控制为 True)	设置寄存器的编号	寄存器值为 0 时不显示，非 0 时显示
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	无、背景图片库或背景图片	无图片、背景图片库或背景图片中选择
背景图片	背景图片选择	图片来源设置为选择背景图片显示
背景图片库	背景图片库选择	图片来源设置为选择背景图片库显示
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，True 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
动作操作窗口	指定动作的作用窗口	/
虚拟按键码	选择虚拟按键码	默认不选择
字符串	输入字符串	只能在软键盘窗口内使用
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	/

例一：调用 SUB 函数

1. 设置功能键的“动作”为“调用函数”；
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：

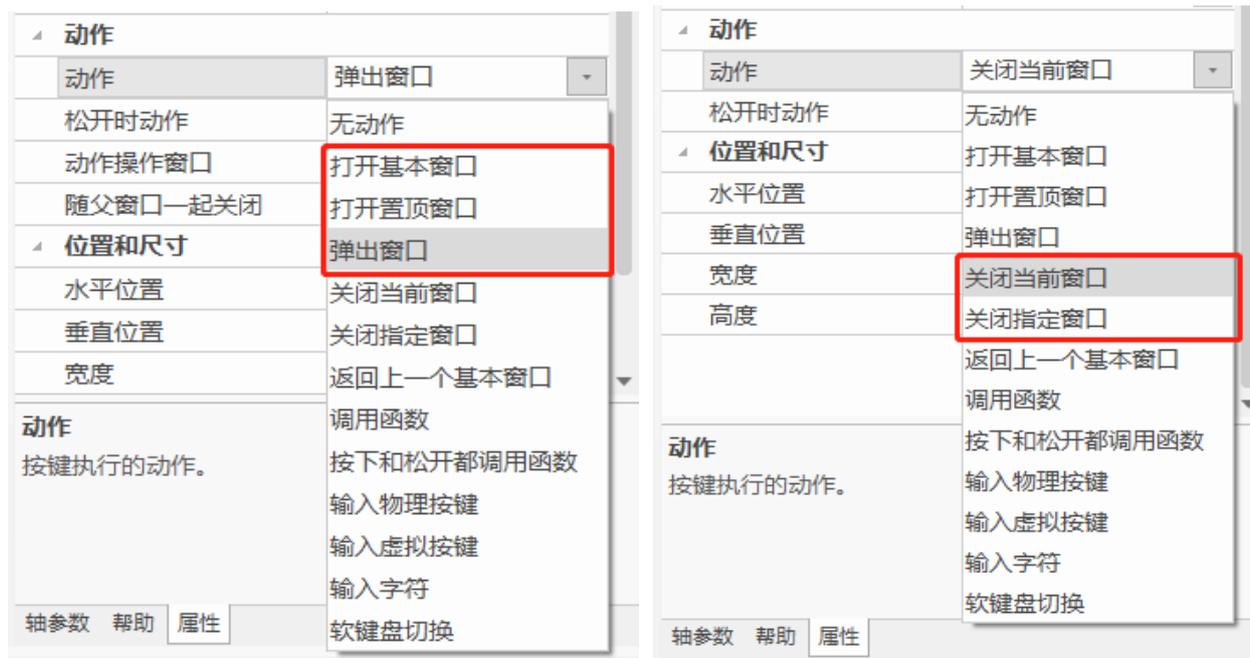
```
global sub sub1()
print "调用函数"
end sub
```

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。

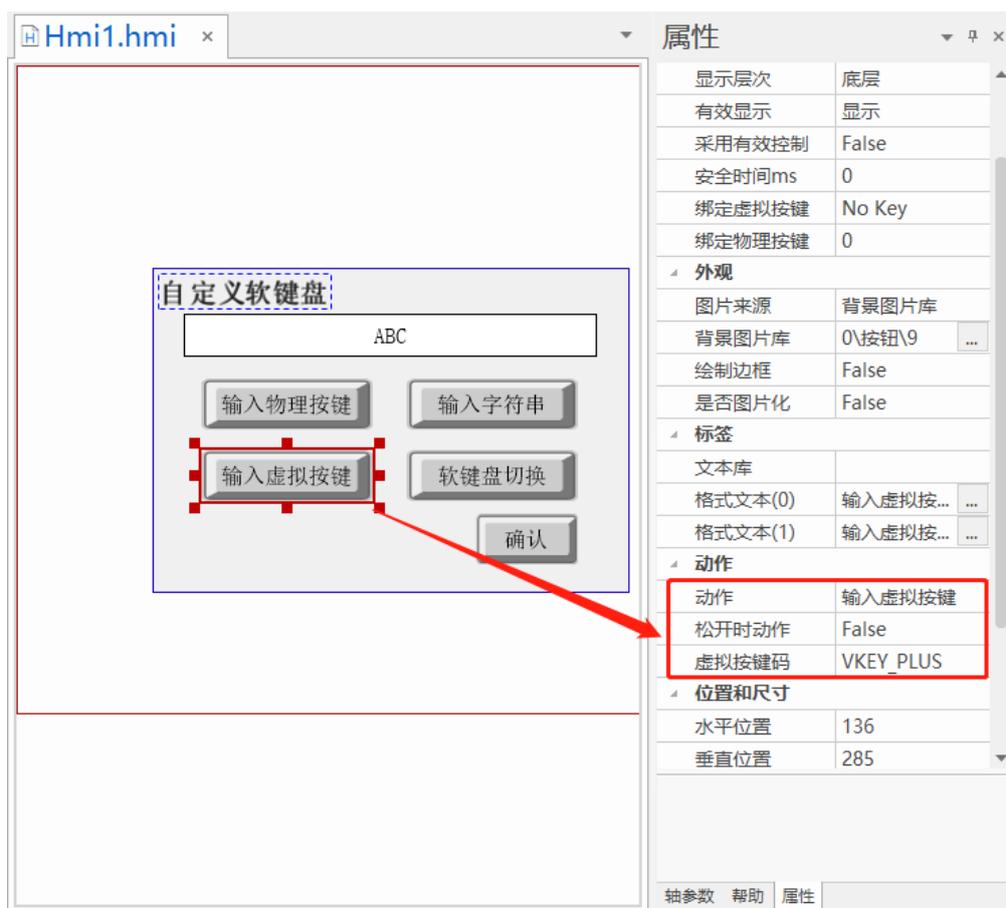


实现效果：

如若“动作”选择“弹出窗口”，“动作操作窗口”选择 11 号窗口，则按下功能键，立即弹出 11 号窗口。

例三：自定义软键盘功能

1. 新建一个基本窗口和一个软键盘窗口（新建窗口在窗口的“属性”将窗口类型修改为软键盘窗口即可），调整软键盘窗口大小（建议调整至比基本窗口小）；
2. 在基本窗口中添加“字符显示”元件，设置该元件“属性”→“可编辑”为 True→“软键盘窗口号”选择上述新建的软键盘窗口号；
3. 在软键盘窗口中添加一个“字符显示”和多个“功能键”。设置“字符显示”的属性→“软键盘显示专用”为 True；设置“功能键”的属性→“动作”选择“输入字符”/“输入虚拟按键”/“输入物理按键”/“软键盘切换”，并对应绑定按键值/软键盘窗口或字符串内容。



实现效果:

下载运行后，点击“功能键”按钮可触发对应的效果，如按下“输入字符串”则在“字符显示”元件上输入“Zmotion”；按下“软键盘切换”按钮则可切换其他软键盘窗口等。



4.3.17. 物理按键

介绍：通过该元件与虚拟按键/示教盒物理按键绑定，实现自定义物理按键动作。

使用方法：点击菜单栏“HMI”→“控件箱”→“物理按键”。将该元件放至合适位置，在该元件“属性”窗口绑定“虚拟按键”/“物理按键”，只能二选一。在“动作”选择需要实现的动作效果，即可实现通过虚拟键/实际硬件按钮进行对应动作。（如：该元件绑定了示教盒物理按键某个键，动作选择“调用函数”，那么当按下示教盒该按钮时，即可调用对应函数）

注意：

1. 该元件只在 Hmi 界面编辑时显示，**实际运行界面不显示**。
2. 绑定物理按键需获取按键键值。**键值需从对应型号的示教盒手册查看**。

1:KeyButton

1. 属性窗口：

属性	
基本属性	
元件编号	1
元件名称	KeyButton1
显示层次	底层
有效显示	显示
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
动作	
动作	调用函数
松开时动作	False
动作函数名	
位置和尺寸	
水平位置	74
垂直位置	109
宽度	100
高度	32

轴参数 帮助 属性

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	顶层：显示在最外层，覆盖底下元件； 中层：被顶层元件覆盖，覆盖底层元件； 底层：被顶层和中间层元件覆盖（默认）。
有效显示	选择元件是否显示	显示：该元件显示于界面之中； 不显示：该元件不显示于界面之中； 仅显示，不可用：该元件可正常显示，但点击该元件时动作不生效
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 True 通过寄存器控制元件是否显示
有效的设备编号 (有效控制为 True)	设备编号	默认 local
有效的寄存器类型 (有效控制为 True)	选择寄存器类型	多种寄存器下拉列表选择
有效的寄存器编号 (有效控制为 True)	设置寄存器的编号	寄存器值为 0 时不显示，非 0 时显示
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
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，True 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
动作操作窗口	指定动作的作用窗口	/
虚拟按键码	选择虚拟按键码	默认不选择
字符串	输入字符串	只能在软键盘窗口内使用
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	/

例：将 ZHD400X 示教盒外部物理按键 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。



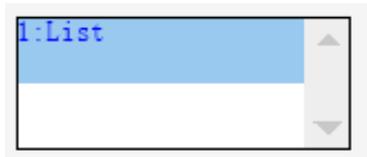
4.3.18. 列表

介绍：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 1-256。

使用方法：点击菜单栏“HMI”→“控件箱”→“列表”。将该元件放至合适位置，在“属性”窗口绑定寄存器及对应编号，设置需显示的“状态数量”，填入对应列表文本即可。该元件也可结合指令 HMI_LISTTEXTS 进行设置。

提示：

1. 当调整元件高度不足以显示全部列表项时，运行后会自动显示滚动条。
2. 选择某行文本时，绑定寄存器的值与文本对应的状态值对应。



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	顶层：显示在最外层，覆盖底下元件； 中层：被顶层元件覆盖，覆盖底层元件； 底层：被顶层和中间层元件覆盖（默认）。
有效显示	选择元件是否显示	显示：该元件显示于界面之中； 不显示：该元件不显示于界面之中； 仅显示，不可用：该元件可正常显示，但点击该元件时动作不生效
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 True 才会显示以下三个参数，通过寄存器控制元件是否显示
有效的设备编号 (有效控制为 True)	设备编号	默认 local
有效的寄存器类型 (有效控制为 True)	选择寄存器类型	多种寄存器下拉列表选择
有效的寄存器编号 (有效控制为 True)	选择寄存器编号	寄存器值为 0 时不显示，非 0 时显示
绑定的设备编号	指定设备	默认 local
绑定的寄存器类型	选择寄存器类型	多种寄存器下拉列表选择
绑定的寄存器编号	设置寄存器的编号	通过获取寄存器不同数值，控制元件不同状态

选中调用函数	选中操作调用的函数	/
动态项	动态读取并修改列表项	使用 HMI_LISTTEXTS 指令设置动态读取列表时必须将动态项开启，
列表方式	设置列表元件为采样列表或下拉式	默认 True，选择 False 时会弹出“下拉调用函数”
弹出方式	可选向下弹出或向上弹出	默认向下弹出
图片来源	无、背景图片库或背景图片	无图片、背景图片库或背景图片中选择
背景图片	背景图片选择	图片来源设置为选择背景图片显示
背景图片库	背景图片库选择	图片来源设置为选择背景图片库显示
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	/
行间距	列表行间距，上下间距相等	默认为 2
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	/
列表文本 0/1	设置列表中的文本	/
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	/

示例：

1. 选择寄存器类型和编号；
2. 选择“状态数量”，设置每个状态下列表对应显示的文本内容。
3. 切换列表选中行可使绑定寄存器的状态值变化。修改寄存器状态值也可切换选中行。
4. 列表文本内容也可通过 HMI_LISTTEXTS 列表指令设置，**但需将属性中“动态项”设置为 True**。指令设置优先级大于元件属性设置。
5. 选择的状态显示颜色、行间距、弹出方式等可通过“属性”窗口进行设置。



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)=1 (为第三个状态 2 的值), 选择的行显示颜色为所设置的颜色。反之, 若使 MODBUS_REG(0)=0 时, 文本 0 行即被选中; MODBUS_REG(0)=3 时, 文本 1 行即被选中。

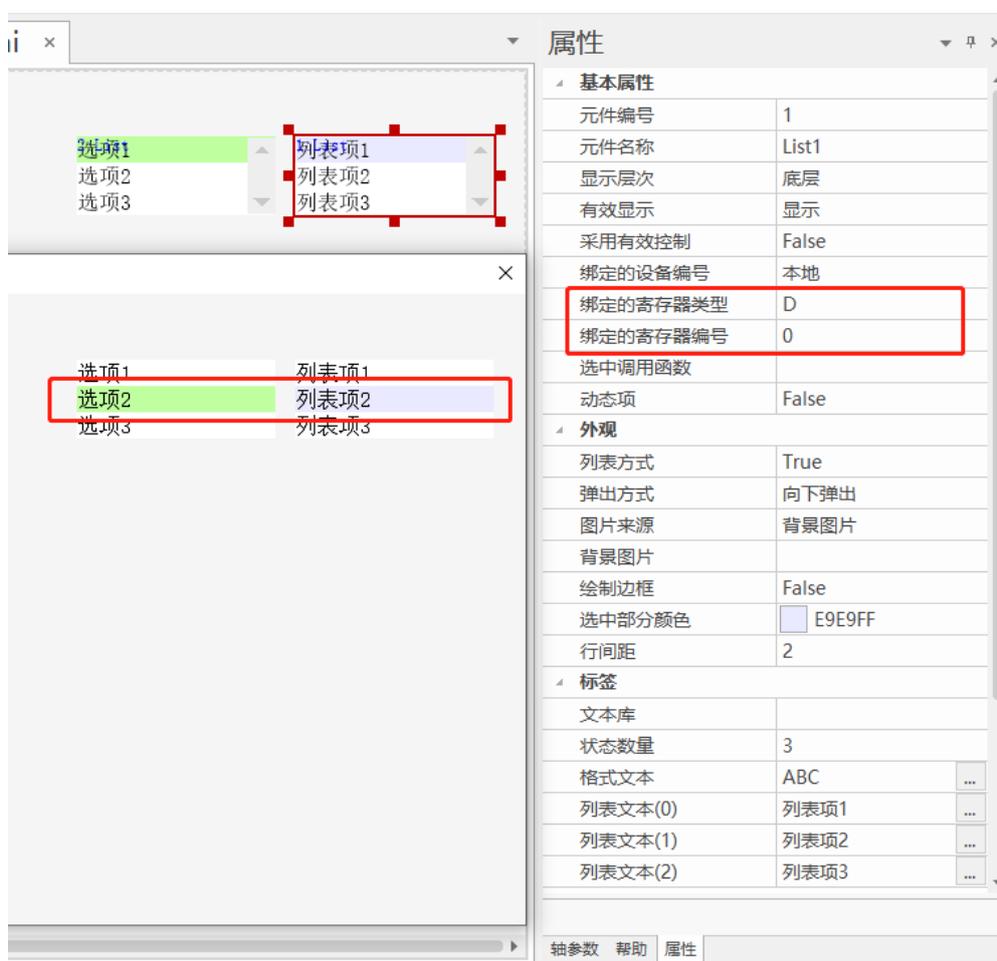
状态值若相同, 相同状态之间无法切换。

详细使用可参见例程: [动态列表](#)

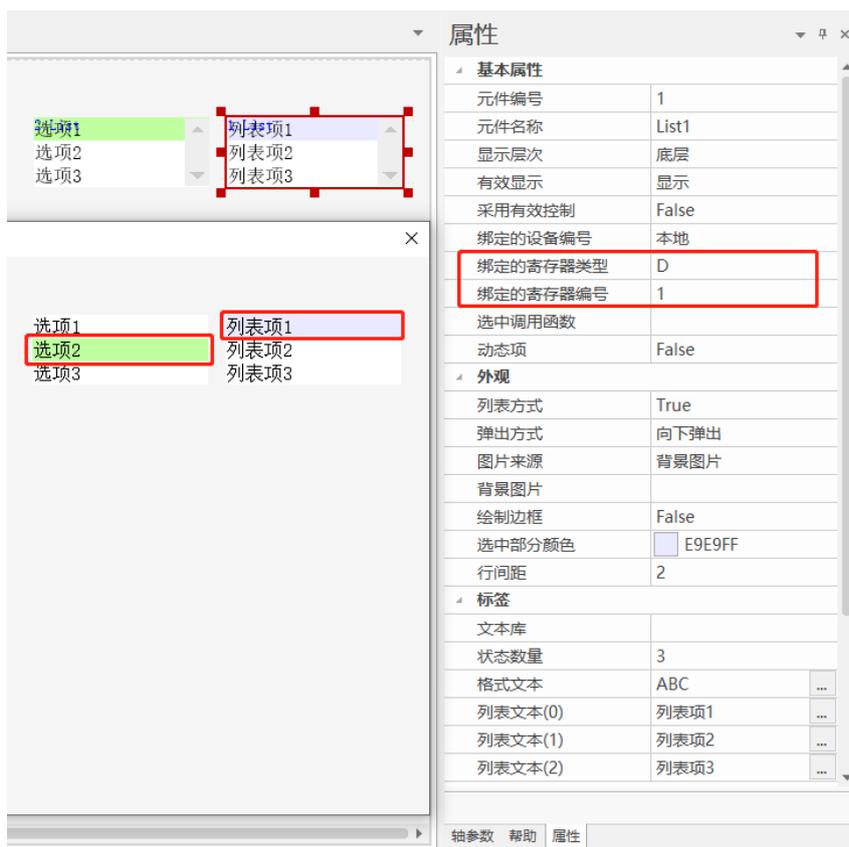
特别注意:

设置多个列表时, 注意每个列表绑定的寄存器变量是否相同。在选择列表选项时, 若设置的寄存器变量相同, 各列表之间互相影响, 会同步选定相同的选项; 若设置的寄存器变量不同, 则每个列表可独立选择选项, 互不影响。如下图。

当两个列表绑定的寄存器都为 D0 时, 运行后选择“列表项 2”的同时, “选项 2”也会被同步选中。



当一个列表绑定的寄存器为 D0, 另一个列表绑定的寄存器为 D1 时, 运行后选择“列表项 1”时, 另一列表可自由选择其他项, 两者互不影响。



4.3.19. 值

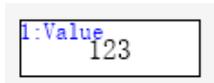
介绍: 支持编辑并显示数值，并将值存入对应绑定寄存器中。

使用方法: 点击菜单栏“HMI”→“控件箱”→“值”。将该元件放至合适位置，在“属性”窗口中选择绑定的寄存器及编号，以及选择是否使用“可编辑”功能调用软键盘。并对值的其他项进行设置，如：数据类型、字符数、小数位数等。

值元件常用功能：

- 显示寄存器的内容，只能显示非字符类型数据
- 调用软键盘窗口直接设置绑定的寄存器的值
- 调用 SUB 函数

注意: 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	顶层：显示在最外层，覆盖底下元件； 中层：被顶层元件覆盖，覆盖底层元件； 底层：被顶层和中间层元件覆盖（默认）。
有效显示	选择元件是否显示	显示：该元件显示于界面之中； 不显示：该元件不显示于界面之中； 仅显示，不可用：该元件可正常显示，但点击该元件时动作不生效
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 True 才会显示以下三个参数，通过寄存器控制元件是否显示
有效的设备编号（有效控制为 True）	设备编号	默认 local
有效的寄存器类型（有效控制为 True）	选择寄存器类型	多种寄存器下拉列表选择
有效的寄存器编号（有效控制为 True）	选择寄存器编号	寄存器值为 0 时不显示，非 0 时显示
可编辑	数据元件是否启用输入	默认 False，选择 True 调用软键盘窗口输入
软键盘窗口号	数据元件编辑时使用的软键盘窗口	可编辑选择 True 时存在的属性
数据类型	数值元件的数据类型设置	默认 INT32

绑定的设备编号	设备编号	默认 local
绑定的寄存器类型	选择寄存器类型	多种寄存器下拉列表选择
绑定的寄存器编号	选择寄存器编号	通过获取寄存器不同数值，控制元件不同状态
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 True 时选择通知的寄存器
通知的设备编号（修改时通知为 True）	设备编号	默认 local
通知的寄存器类型（修改时通知为 True）	选择寄存器类型	多种寄存器下拉列表选择
通知的寄存器编号（修改时通知为 True）	选择寄存器编号	通过获取寄存器不同数值，控制元件不同状态
图片来源	无、背景图片库或背景图片	无图片、背景图片库或背景图片中选择
背景图片	背景图片选择	图片来源设置为选择背景图片显示
背景图片库	背景图片库选择	图片来源设置为选择背景图片库显示
绘制边框	选择是否绘制边框	/
边框颜色	设置边框颜色	设置边框为 True 时显示该属性
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	设置为 True 时元件显示内容为 *号	默认为 False
最小值	输入下限	/
最大值	输入上限	/
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	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

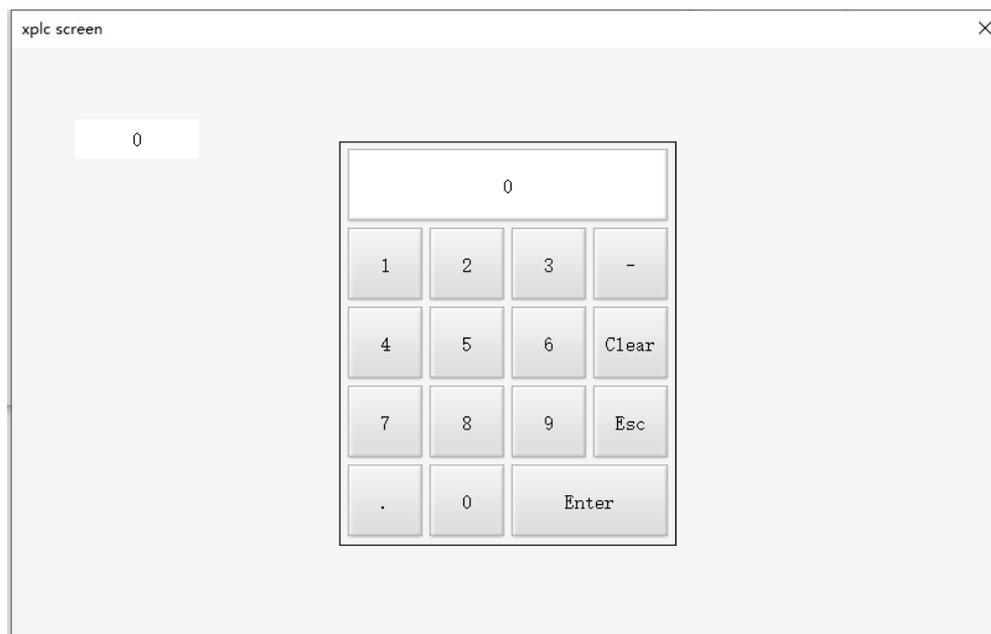
1. “可编辑”选择要调用的软件盘窗口
2. 设置寄存器的类型和编号
3. 根据需求设置数据类型，最大值，最小值等



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。



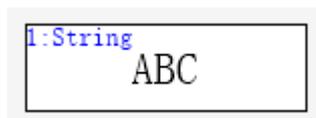
4.3.20. 字符显示

介绍：用于编辑并显示所有字符串类型数据。

使用方法：点击菜单栏“HMI”→“控件箱”→“字符显示”。将该元件放至合适位置，在“属性”窗口中选择绑定的寄存器及编号，以及选择是否使用“可编辑”功能调用软键盘。并对其他项进行设置，如字符数、多行显示、调用函数等。

注意：

1. 寄存器类型中的数据必须为字符串类型；
2. 调用软键盘窗口可自定义输入数据并显示。当寄存器类型为自定义@且所设置的变量不为数组时，软键盘不可调用编辑；
3. “可编辑”设置为 True，选择调用软键盘窗口后，便不能调用函数。
4. 开启“多行”和“可编辑”属性时，使用中不弹出软键盘窗口，可用物理键盘直接输入。（输入中文时需用“Ctrl+Shift”键进行切换到内置中文输入法）



1. 属性窗口：

属性	
基本属性	
元件编号	1
元件名称	String1
显示层次	底层
有效显示	显示
采用有效控制	False
软键盘显示专用	False
可编辑	False
绑定的设备编号	本地
绑定的寄存器类型	D
绑定的寄存器编号	0
安全时间ms	0
修改时通知	False
外观	
多行	False

字符数	16
图片来源	无
绘制边框	True
边框颜色	000000
密码显示	False
标签	
格式文本	ABC
动作	
点击调用函数	
位置和尺寸	
水平位置	106
垂直位置	154
宽度	100
高度	32

轴参数 帮助 属性

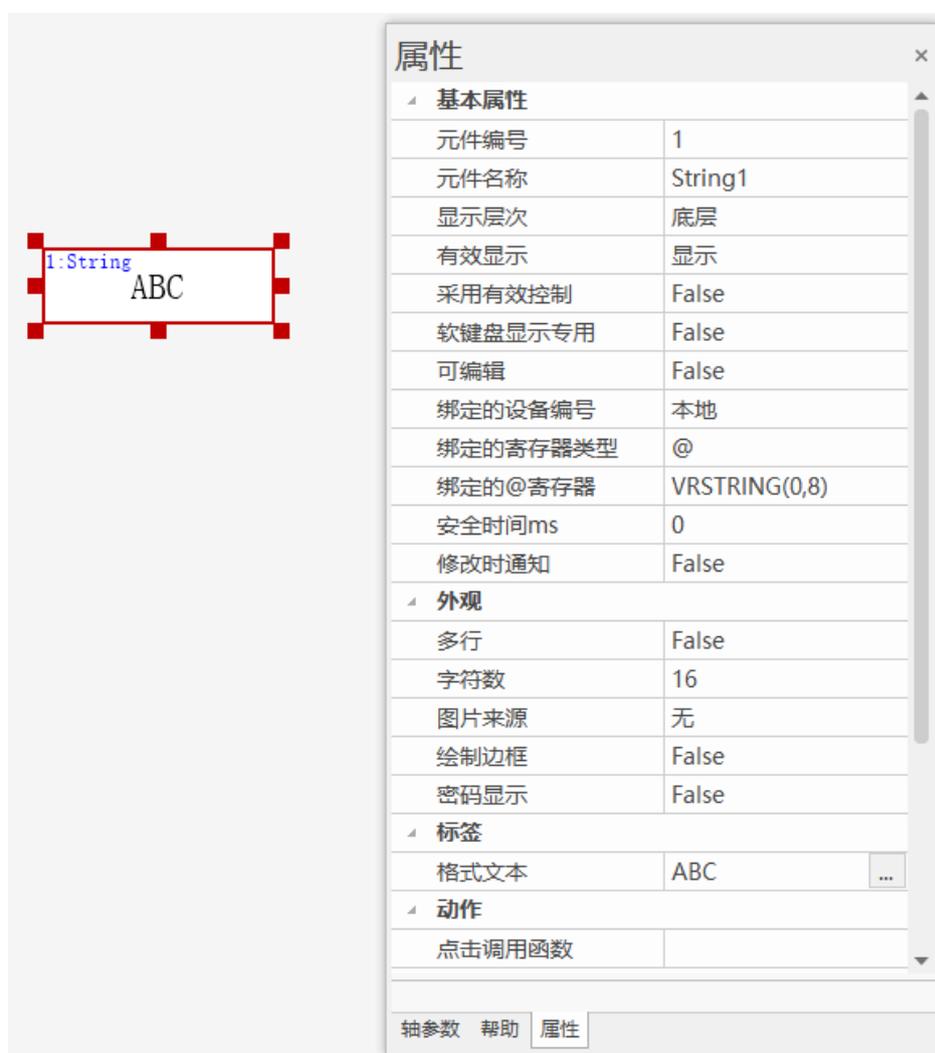
2. 属性说明：

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	顶层：显示在最外层，覆盖底下元件； 中层：被顶层元件覆盖，覆盖底层元件； 底层：被顶层和中间层元件覆盖（默认）。
有效显示	选择元件是否显示	显示：该元件显示于界面之中； 不显示：该元件不显示于界面之中； 仅显示，不可用：该元件可正常显示，但点击该元件时动作不生效
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 True 通过寄存器控制元件是否显示
有效的设备编号（有效控制为 True）	设备编号	默认 local
有效的寄存器类型（有效控制为 True）	选择寄存器类型	多种寄存器下拉列表选择
有效的寄存器编号（有效控制为 True）	选择寄存器编号	寄存器值为 0 时不显示，非 0 时显示
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	字符显示元件是否启用输入	默认 False，选择 True 调用软键盘窗口输入
软键盘窗口号	字符显示元件编辑时使用的软键盘窗口	可编辑选择 True 时存在的属性
绑定的设备编号	设备编号	默认 local
绑定的寄存器类型	选择寄存器类型	多种寄存器下拉列表选择
绑定的寄存器编号	选择寄存器编号	通过获取寄存器不同数值，控制元件不同状态
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 True 时选择通知的寄存器
通知的设备编号（修改时通知为 True）	设备编号	默认 local
通知的寄存器类型（修改时通知为 True）	选择寄存器类型	多种寄存器下拉列表选择
通知的寄存器编号（修改时通知为 True）	选择寄存器编号	通过获取寄存器不同数值，控制元件不同状态
多行	字符为多行字符时设置为 True	/
字符数	设置字符元件操作字符的长度	默认 16
图片来源	无、背景图片库或背景图片	无图片、背景图片库或背景图片中选择
背景图片	背景图片选择	图片来源设置为选择背景图片显示
背景图片库	背景图片库选择	图片来源设置为选择背景图片库显示
绘制边框	选择是否绘制边框	/

边框颜色	设置边框颜色	设置边框为 True 时显示该属性
密码显示	设置为 True 时元件显示内容为 *号	默认为 False
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
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。

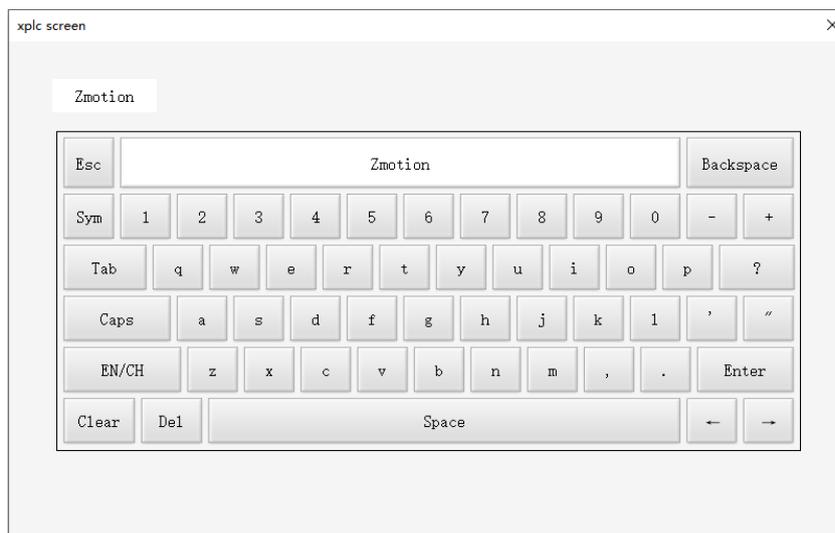
例二：自定义元件显示文本

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。



4.3.21. 滑块开关

介绍: 在一定取值范围内, 能进行拖动实现改变数值大小的元件。如: 可通过滑块实现视图缩放、倍率调节、充当滑块开关等, 具体应用需根据实际需求。

使用方法: 点击菜单栏“HMI”→“控件箱”→“滑块开关”。将该元件放至合适位置, 在“属性”窗口中选择绑定的寄存器及编号。通过寄存器变量控制数据变化从而实现效果。

注意: 目前控制器仅 4 系列及以上控制器支持, 示教盒 ZHD500X 已支持, ZHD300X 和 ZHD400X 升级固件可支持。



1. 属性窗口

属性		属性	
基本属性		刻度颜色	000000
元件编号	1	主刻度数	3
元件名称	Slip1	副刻度数	4
显示层次	底层	是否显示符号	True
有效显示	显示	滑块选择	0\滑块\0
采用有效控制	False	滑块颜色	00FFFF
绑定的设备编号	本地	滑块水平位置	0
绑定的寄存器类型	D	滑块垂直位置	0
绑定的寄存器编号	0	滑块宽度	16
修改时通知	False	滑块高度	24
外观		滑轨1颜色	404040
图片来源	无	滑轨2颜色	C0C0C0
绘制边框	False	标签	
填充	False	格式文本	符号
边界	1	位置和尺寸	
方向	从左到右	水平位置	32
是否显示刻度	True	垂直位置	136
最小刻度	1	宽度	200
最小值	0.000000	高度	48
最大值	100.000000	轴参数 帮助 属性	

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	顶层：显示在最外层，覆盖底下元件； 中层：被顶层元件覆盖，覆盖底层元件； 底层：被顶层和中间层元件覆盖（默认）。
有效显示	选择元件是否显示	显示：该元件显示于界面之中； 不显示：该元件不显示于界面之中； 仅显示，不可用：该元件可正常显示，但点击该元件时动作不生效
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 True 才会显示以下三个参数，通过寄存器控制元件是否显示
有效的设备编号（有效控制为 True）	设备编号	默认 local
有效的寄存器类型（有效控制为 True）	选择寄存器类型	多种寄存器下拉列表选择
有效的寄存器编号（有效控制为 True）	选择寄存器编号	寄存器值为 0 时不显示，非 0 时显示
绑定的设备编号	指定设备	默认 local
绑定的寄存器类型	选择寄存器类型	多种寄存器下拉列表选择
绑定的寄存器编号	设置寄存器的编号	通过获取寄存器不同数值，控制元件不同状态
修改时通知	修改后发出 bit 位通知(设 ON 或 OFF)	默认 False，选择 True 时选择通知的寄存器
图片来源	无、背景图片库或背景图片	无图片、背景图片库或背景图片中选择
背景图片	背景图片选择	图片来源设置为选择背景图片显示
背景图片库	背景图片库选择	图片来源设置为选择背景图片库显示
绘制边框	选择是否绘制边框	/
填充	选择是否填充颜色	填充整个元件
边界	调节元件与该元件边框的距离	边界值越大，距离越大
方向	选择滑块的方向	有 4 个方向选择：从左到右、从右到左、从上到下、从下到上
是否显示刻度	显示刻度时设置为 True	设置为 True 时显示最小刻度属性
最小刻度	设置滑块每一次拖动的刻度	最小为 1
最小值	刻度的最小起始数值	正负数均可
最大值	刻度的最大结束数值	正负数均可
刻度颜色	选择刻度数的显示颜色	默认为黑色
主刻度数	设置滑块长刻度的段数	/
副刻度数	设置滑块短刻度的段数	/
是否显示符号	是否显示刻度	/
滑块选择	选择滑块的样式	/
滑块颜色	设置滑块的颜色	/
滑块水平位置	设置滑块水平位置	为 0 时滑块水平居中

滑块垂直位置	设置滑块垂直位置	为 0 时滑块垂直居中
滑块宽度	设置滑块宽度	/
滑块高度	设置滑块高度	/
滑轨 1 颜色	滑块右边滑轨的颜色	/
滑轨 2 颜色	滑块左边滑轨的颜色	/
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：滑块刻度的基础设置

选择寄存器类型，用于存放滑块变化的数值参数，拖动滑块可根据实际调节大小。滑块显示的刻度数及分度数可通过主刻度数和副刻度数设置。先确定最小值和最大值后根据实际情况设置主刻度数插入的线数为 5，再设置副刻度数插入单格中的线数为 2，从而将单格分成 3 小格。根据喜好对进度条颜色，滑块颜色和样式进行选择。

例程详情参见[视图缩放](#)。

属性

绑定的寄存器编号	0
修改时通知	False
外观	
图片来源	无
绘制边框	False
填充	False
边界	10
方向	从左到右
是否显示刻度	True
最小刻度	1
最小值	0.000000
最大值	200.000000
刻度颜色	000000
主刻度数	5
副刻度数	2
是否显示符号	True
滑块选择	0\滑块\0
滑块颜色	FFE9FF
滑块水平位置	0
滑块垂直位置	0
滑块宽度	15
滑块高度	24
滑轨1颜色	C0A0FF
滑轨2颜色	000000

轴参数 帮助 属性

例二：“滑块开关”绑定寄存器的方法

1. 添加“滑块开关”元件；

2. 在元件“属性”中的“寄存器类型”选择合适的寄存器类型，并填写寄存器编号；（例如：下图选择寄存器为DT0）

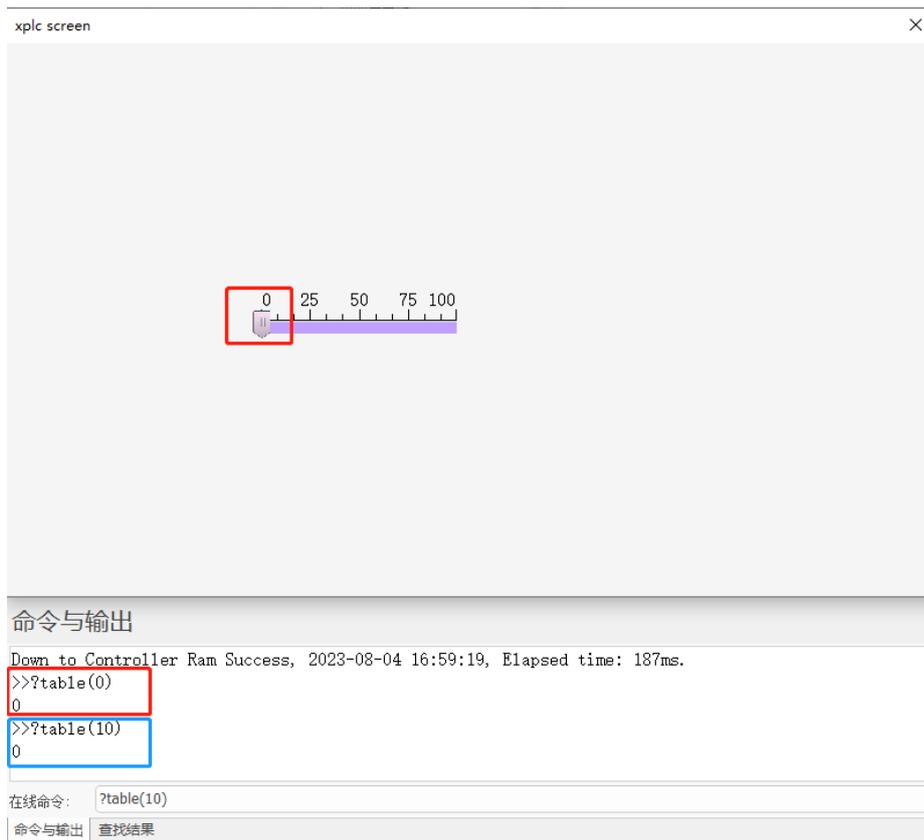
注意：若选择“修改时通知”为 True 时，“通知的设备编号”寄存器编号不能与设备编号的寄存器重复。

3. 可在 Basic 函数中设置该寄存器的初始值，未设置默认初始值为 0。

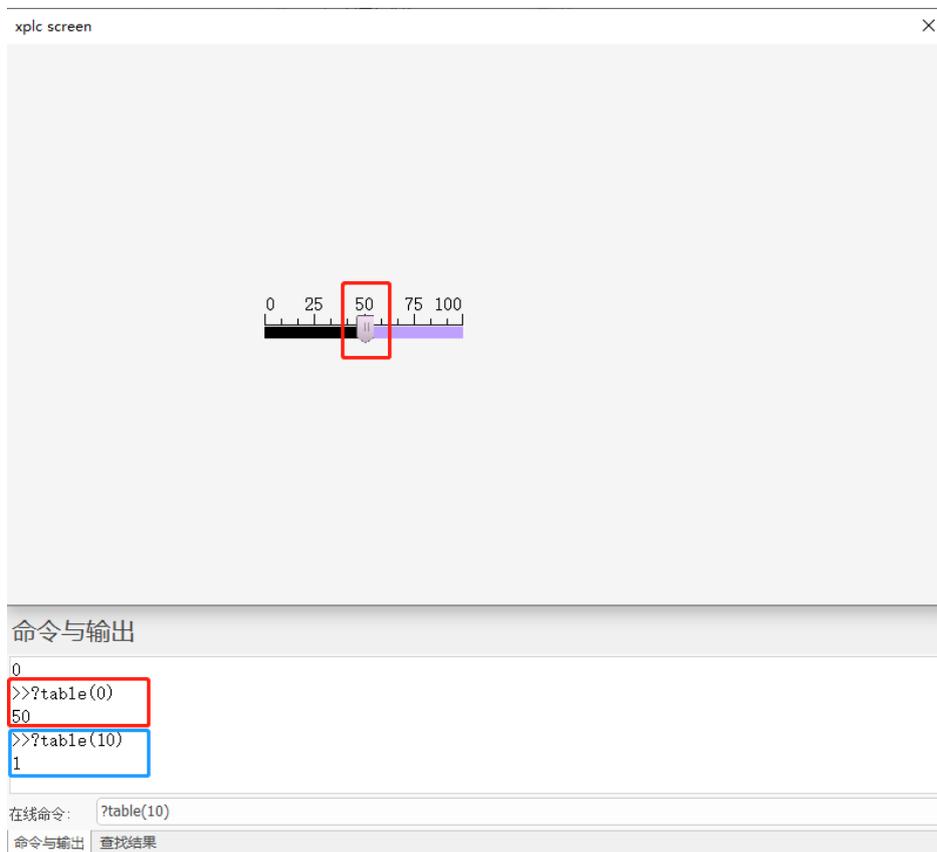


实现效果：

初始值默认为 0，table(0)的打印值为 0，同时由于未操作拖动滑块，此时“修改时通知”的寄存器 table(10)打印值也为 0。



当滑块滑到 50 处，table(0)的打印值同步刷新为 50。而由于已拖动滑块做出修改，此时 table(10)的值变为 1，表示该元件修改已生效。

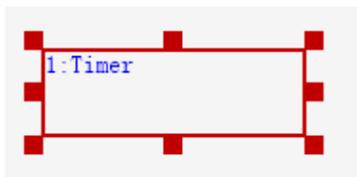


4.3.22. 定时器

介绍：利用定时器功能定时刷新进行重复动作。

使用方法：点击菜单栏“HMI”→“控件箱”→“定时器”。将该元件放至合适位置，在“属性”窗口中设置好定时器启动间隔时间，并设置好“动作”内容，即可实现定时重复动作。“动作”支持调用 SUB 函数或对寄存器进行赋值。

注意：Hmi 界面编辑时显示，实际运行中不显示。



1. 属性窗口：

属性	
基本属性	
元件编号	1
元件名称	Timer1
显示层次	底层
有效显示	显示
采用有效控制	False
时间ms	1000
循环类型	False
写入的设备编号	本地
写入的寄存器类型	
写入的寄存器编号	0
动作	
动作	无动作
松开时动作	False
位置和尺寸	
水平位置	130
垂直位置	117
宽度	100
高度	32

轴参数 帮助 属性

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	顶层：显示在最外层，覆盖底下元件； 中层：被顶层元件覆盖，覆盖底层元件； 底层：被顶层和中间层元件覆盖（默认）。
有效显示	选择元件是否显示	显示：该元件显示于界面之中； 不显示：该元件不显示于界面之中； 仅显示，不可用：该元件可正常显示，但点击该元件时动作不生效
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 True 通过寄存器控制元件是否显示
有效的设备编号（有效控制为 True）	设备编号	默认 local
有效的寄存器类型（有效控制为 True）	选择寄存器类型	多种寄存器下拉列表选择
有效的寄存器编号（有效控制为 True）	选择寄存器编号	寄存器值为 0 时不显示，非 0 时使用
时间 ms	最少按键时间	单位 ms（仅支持整数，不支持浮点数）
循环类型	选择定时器是否循环	默认 False
写入的设备编号	设备编号	默认 local
写入的寄存器类型	选择寄存器类型	多种寄存器下拉列表选择
写入的寄存器编号	选择寄存器编号	通过获取寄存器不同数值，控制元件不同状态
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，True 为松开时动作
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
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“调用函数”，“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。

属性	
基本属性	
元件编号	1
元件名称	Timer1
显示层次	底层
有效显示	显示
采用有效控制	False
时间ms	1000
循环类型	False
写入的设备编号	本地
写入的寄存器类型	
写入的寄存器编号	0
动作	
动作	调用函数
松开时动作	False
动作函数名	sub1
位置和尺寸	
水平位置	205
垂直位置	183
宽度	100
高度	32

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。

例二：给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“数据增加/减少”，“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。

查看寄存器值的方法:

- 点击菜单栏“工具”→“寄存器”，选择绑定的寄存器类型及地址，点击“读取”即可查看。
- 在[命令与输出]窗口的“在线命令”行，输入：`print+绑定的寄存器地址及状态值`，点击“发送”即可（如输入：`print MODBUS_BIT(0) / ?MODBUS_BIT(0)`）；

4.3.23. 自定义

介绍: 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。

使用方法：点击菜单栏“HMI”→“控件箱”→“自定义”。将该元件放至合适位置，调整元件大小确定绘图区域。在 Basic 文件中先定义全局的刷新函数和绘图函数，后在该元件“属性”窗口中“刷新函数”、“绘图函数”分别对应调用 Basic 的 SUB 函数。

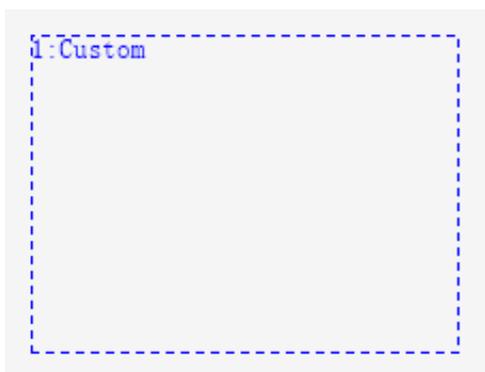
注意：绘图函数的坐标都是相对于元件的左上方为零点。

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。

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：

属性		刷新函数	
基本属性		绘图函数	
元件编号	1	位置和尺寸	
元件名称	Custom1	水平位置	93
显示层次	底层	垂直位置	136
有效显示	显示	宽度	200
采用有效控制	False	高度	150
数据索引	0		

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	顶层：显示在最外层，覆盖底下元件； 中层：被顶层元件覆盖，覆盖底层元件； 底层：被顶层和中间层元件覆盖（默认）。
有效显示	选择元件是否显示	显示：该元件显示于界面之中； 不显示：该元件不显示于界面之中； 仅显示，不可用：该元件可正常显示，但点

		击该元件时动作不生效
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 True 通过寄存器控制元件是否显示
有效的设备编号（有效控制为 True）	设备编号	默认 local
有效的寄存器类型（有效控制为 True）	选择寄存器类型	多种寄存器下拉列表选择
有效的寄存器编号（有效控制为 True）	选择寄存器编号	寄存器值为 0 时不显示，非 0 时显示
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	/

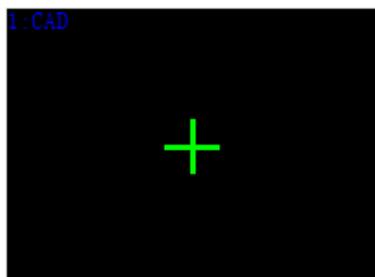
4.3.24. CAD

介绍：用于显示矢量图形的元件。

使用方法：点击菜单栏“HMI”→“控件箱”→“CAD”。将该元件放至合适位置，选择三次文件导入的通道号（仅 0、1 两个通道可设置），根据需求设置图层/轨迹/空移颜色等。导入/关闭图形等操作需配合 CAD 指令进行使用。

注意：

1. 该矢量图片文件必须存放至 flash 文件中。
2. 目前支持导入的图形格式为：.dxf/.ai/.plt/.dst/.nc/.cnc。



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	顶层：显示在最外层，覆盖底下元件； 中层：被顶层元件覆盖，覆盖底层元件； 底层：被顶层和中间层元件覆盖（默认）。
有效显示	选择元件是否显示	显示：该元件显示于界面之中； 不显示：该元件不显示于界面之中； 仅显示，不可用：该元件可正常显示，但点击该元件时动作不生效
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 True 通过寄存器控制元件是否显示
有效的设备编号（有	设备编号	默认 local

效控制为 True)		
有效的寄存器类型 (有效控制为 True)	选择寄存器类型	多种寄存器下拉列表选择
有效的寄存器编号 (有效控制为 True)	选择寄存器编号	寄存器值为 0 时不显示, 非 0 时显示
通道号	设置通道号	最多只支持两个通道: 0/1, 最多能同时运行的文件个数
绘制边框	选择是否绘制边框	默认 False, 选择 True 弹出[边框颜色]选择
背景颜色	选择背景颜色	默认黑色
图层颜色	是否使用图层颜色	设置为 True 时, 不同图层图形颜色不一致
轨迹颜色	选择轨迹颜色	/
空移颜色	选择空移颜色	不连贯的区域之间的位移为空移
选中颜色	选择选中颜色	/
辅助颜色	选择辅助颜色	/
格式文本	设置元件文本的样式等参数	/
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	/

初级的 CAD 使用可以参照第八章【[CAD 导入矢量图片](#)】

4.3.25. 三次文件编辑器

介绍: 在 Hmi 界面上进行三次程序开发的元件, 支持编辑和显示程序等。需配合三次文件指令使用。

使用方法: 点击菜单栏“HMI”→“控件箱”→“三次文件编辑器”。将该元件放至合适位置, 设置通道号后, 配合其他元件和 Basic 程序, 实现程序的导入和实时编辑(编辑需在“属性”窗口中将“可编辑”切换为 True)。三次文件编辑控件“选中多行”功能, 可通过鼠标按下滑动或者通过指令进行操作。支持对选中部分进行删除、编辑。

File Edit	Row 1	Col 1	Ln 11111
1G01 X0.000 Y0.000 Z0.000			
2G01 X0.000 Y0.000 Z0.000			
3G01 X0.000 Y0.000 Z0.000			
4G01 X0.000 Y0.000 Z0.000			
5G01 X0.000 Y0.000 Z0.000			
6G01 X0.000 Y0.000 Z0.000			
7G01 X0.000 Y0.000 Z0.000			
8G01 X0.000 Y0.000 Z0.000			
9G01 X0.000 Y0.000 Z0.000			

1. 属性窗口:

属性	
基本属性	
元件编号	1
元件名称	File3Edit1
显示层次	底层
有效显示	显示
采用有效控制	False
通道号	0
可编辑	False
安全时间ms	0
修改时通知	False
外观	
绘制边框	False
图片来源	无
行间距	2
行号	True

行号宽度	30
文件信息	True
垂直滚动条	False
水平滚动条	False
标签	
格式文本(0)	Line number ...
格式文本(1)	Code ...
格式文本(2)	Select ...
格式文本(3)	File info ...
位置和尺寸	
水平位置	33
垂直位置	53
宽度	400
高度	200

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	顶层：显示在最外层，覆盖底下元件； 中层：被顶层元件覆盖，覆盖底层元件； 底层：被顶层和中间层元件覆盖(默认)。
有效显示	选择元件是否显示	显示：该元件显示于界面之中； 不显示：该元件不显示于界面之中； 仅显示，不可用：该元件可正常显示，但点击该元件时动作不生效
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 True 通过寄存器控制元件是否显示
有效的设备编号 (有效控制为 True)	设备编号	默认 local
有效的寄存器类型 (有效控制为 True)	选择寄存器类型	多种寄存器下拉列表选择
有效的寄存器编号 (有效控制为 True)	选择寄存器编号	寄存器值为 0 时不显示，非 0 时显示
通道号	设置三次文件通道号	最多只支持三个通道：0/1/2
可编辑	是否启用元件的编辑功能	默认 False
软键盘窗口号	数据元件编辑时使用的软键盘窗口	可编辑选择 True 时存在的属性
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	默认 False，选择 True 时选择通知的寄存器

	(设 ON 或 OFF)	
通知的设备编号 (修改时通知为 True)	设备编号	默认 local
通知的寄存器类型 (修改时通知为 True)	选择寄存器类型	多种寄存器下拉列表选择
通知的寄存器编号 (修改时通知为 True)	选择寄存器编号	通过获取寄存器不同数值，控制元件不同状态
绘制边框	选择是否绘制边框	/
图片来源	无、背景图片库或背景图片	无图片、背景图片库或背景图片中选择
背景图片	背景图片选择	图片来源设置为选择背景图片显示
背景图片库	背景图片库选择	图片来源设置为选择背景图片库显示
行间距	设置行与行之间的距离	默认为 2
行号	选择是否显示行序号	/
行号宽度	设置行序号位置宽度	行号设置为 True 时显示该属性
文件信息	设置是否显示标题行	/
垂直滚动条	设置是否使用垂直方向滚动条	/
水平滚动条	设置是否使用水平方向滚动条	/
格式文本 0	设置行号(Line number)样式	文本名称不可更改
格式文本 1	设置编程区域(Code)样式	文本名称不可更改
格式文本 2	设置编程区域选中状态(Select)样式	文本名称不可更改
格式文本 3	设置文件信息行(File info)样式	文本名称不可更改
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	/

初级的三次文件编辑器使用可以参照第八章【[CAD 结合三次文件使用](#)】。

4.3.26. 报表视图

介绍： [报表]元件以表格的形式呈现多组数据，用于显示和管理报表数据。在[报表]元件中，用户可以查看和编辑报表数据，支持字符和中文输入。

注意： 使用该控件需同时确保 RTHmi 版本为 V1.3.0 版本及以上和 RTSys 版本为 V1.2.02 及以上版本！

使用方法： 点击菜单栏“HMI”→“控件箱”→“报表视图”。将该元件放至合适位置，设置行列数、是否可编辑、是否显示表头行/列等属性，然后点击“单元格数据”，对报表内容进行编辑，设置完成、保存下载后，可以查看（和编辑）报表数据。

1:Report View	A	B	C
1			
2			
3			

1. 属性窗口：

属性

安全时间ms	0
外观	
行数	3
行高	20
列数	3
显示表头行	True
表头行高	20
显示表头列	True
表头列宽	20
选中部分颜色	 99CCFF
外框颜色	 000000
外框线宽	1
外框线段类型	默认
网格颜色	 000000
网格线宽	1
网格线段类型	默认
固定列宽	False
固定行高	False

垂直滚动条	False
水平滚动条	False
标签	
文本库	
格式文本(0)	Cell ...
格式文本(1)	Header row ...
格式文本(2)	Header colu... ...
单元格数据	Modify ...
动作	
点击调用函数	
修改调用函数	
位置和尺寸	
水平位置	0
垂直位置	205
宽度	400
高度	200

轴参数 属性 帮助

2. 属性说明：

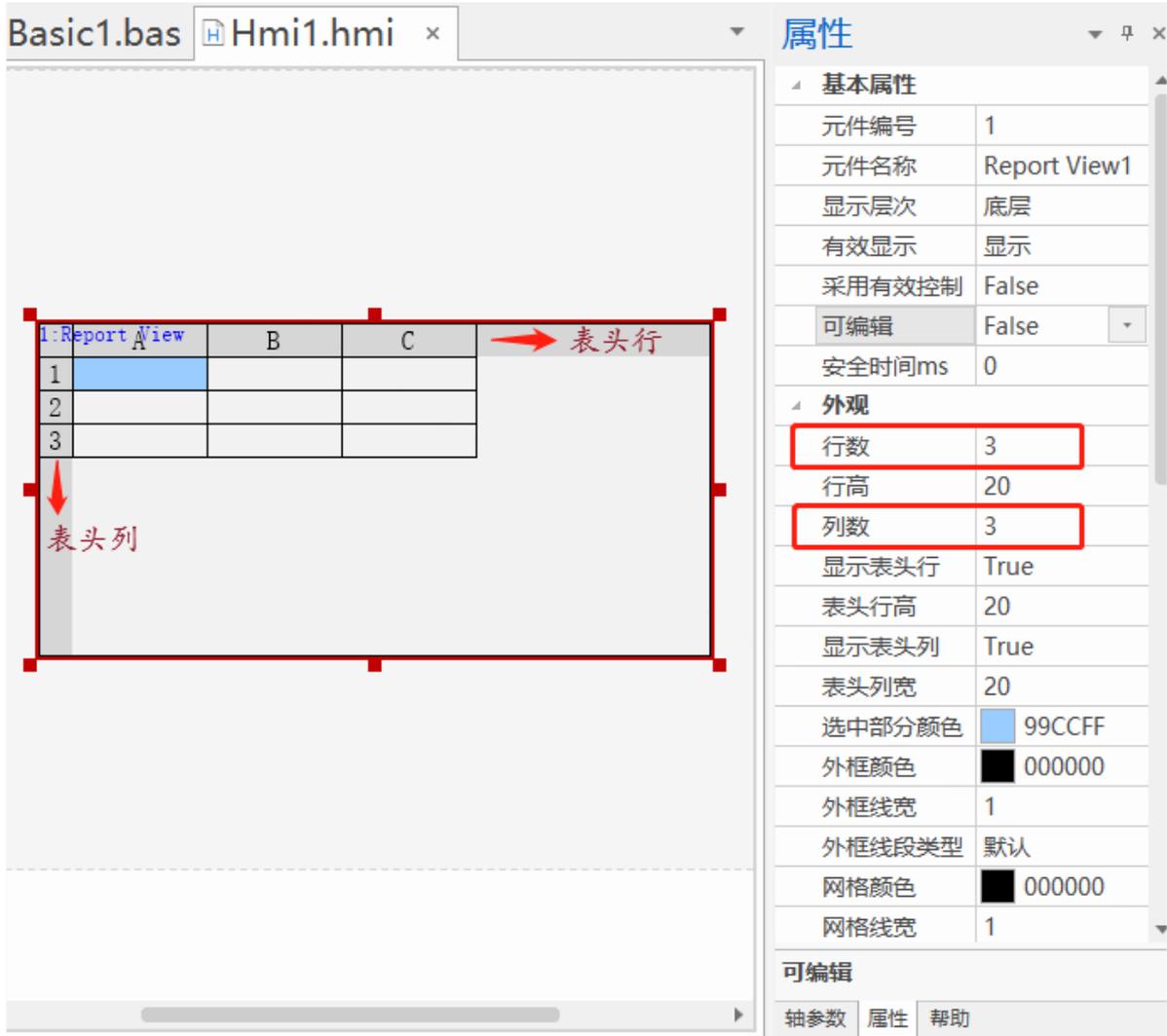
属性	功能	说明
元件编号	/	/
元件名称	/	/
显示层次	选择元件显示层次	顶层：显示在最外层，覆盖底下元件 中层：被顶层元件覆盖，覆盖底层元件 底层：被顶层和中间层元件覆盖（默认）
有效显示	选择元件是否显示	显示：该元件显示于界面之中

		不显示：该元件不显示于界面之中 仅显示，不可用：该元件可正常显示，但点击该元件时动作不生效
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 True 通过寄存器控制元件是否显示
可编辑	报表视图元件是否启用输入	默认 False，选择 True 调用软键盘窗口输入
软键盘窗口号	数据元件编辑时使用的软键盘窗口	可编辑选择 True 时存在的属性
安全时间 ms	最少按键时间	单位 ms
行数	设置表格行数	不包含表头行，最多可设置 512 行
行高	每一行的显示行高度（1~n 行）	每一行的行高可以在下载后的 Hmi 界面拖动修改 每一列的列宽可以在“单元格数据”界面 → “文件头设置”框 → “宽度”修改，或者直接拖动修改，又或者下载后的 Hmi 界面拖动修改
列数	设置表格列数（1~n 行）	不包含表头列，最多可设置 16 列
显示表头行	True：显示表头列 False：不显示表头列	默认为 True
表示行高	设置表头行的高度	/
显示表头列	True：显示表头列 False：不显示表头列	默认为 True
表头列宽	设置表头列的宽度	/
选中部分颜色	设置列表元件选中部分的颜色	默认
外框颜色	设置表格外框颜色	默认黑色
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	默认实线
网格颜色	设置表格内框线颜色	默认黑色
网格线宽	设置表格网格的线段宽度	默认宽度为 1，最大为 20
网格线段类型	选择绘制网格框的线段的样式	默认实线
固定列宽	True：列宽固定 False：可以拉伸列的宽度	默认为 False
固定行高	True：行高固定 False：可以拉伸行的高度	默认为 False
垂直滚动条	设置是否使用垂直方向滚动条	/
水平滚动条	设置是否使用水平方向滚动条	/
格式文本 0（Cell）	设置单元格显示文本格式	文本名称不可更改
格式文本 1（Header row）	设置报表首行的样式	文本名称不可更改
格式文本 2（Header column）	设置报表首列的样式	文本名称不可更改

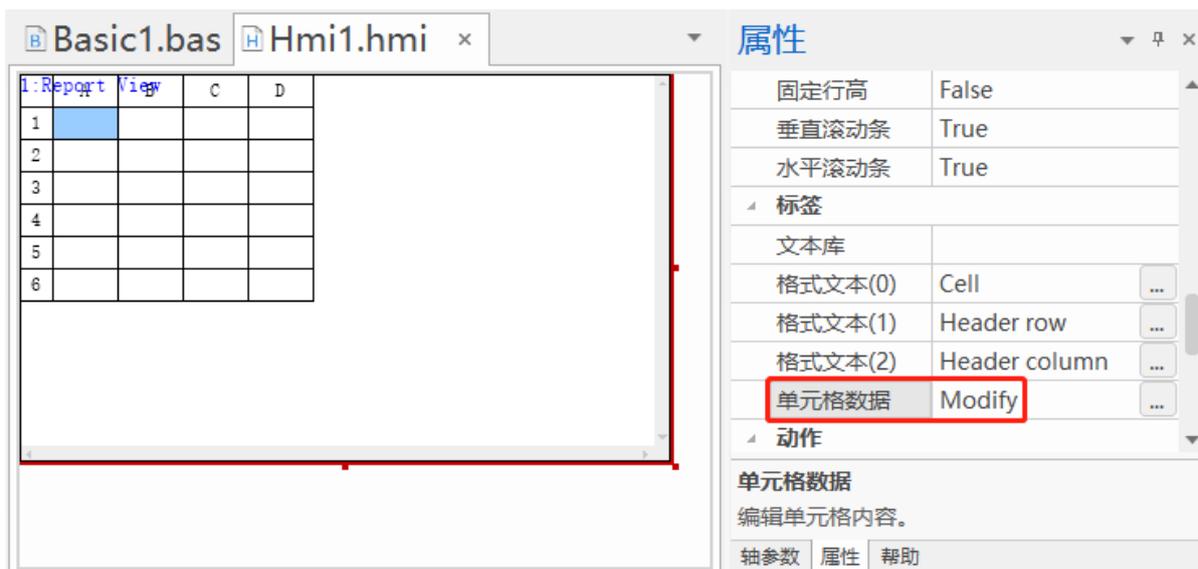
单元格数据	<p>编辑单元格内容</p> <p>字符数: 单元格最大显示字符数, 最大限制 255 (使用字符或数值显示时都会受字符数大小的限制)</p> <p>数据类型: 限制输入类型, 选择“字符显示”或者“数值”,</p> <p>= 字符显示时为空;</p> <p>= 数值时以下属性有效:</p> <p>小数位数: 限制输入的有效小数位数, 0~13</p> <p>是否限制范围: 勾选限制, 勾选时下列属性有效: (最小值<最大值有效)</p> <p>最小值: 限制输入最小值</p> <p>最大值: 限制输入最大值</p>	/
点击调用函数	按键按下时调用 Basic 定义的 SUB 函数	函数必须是 GLOBAL 全局类型
修改调用函数	修改完成后触发调用的 Basic 定义的 SUB 函数	函数必须是 GLOBAL 全局类型
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	/

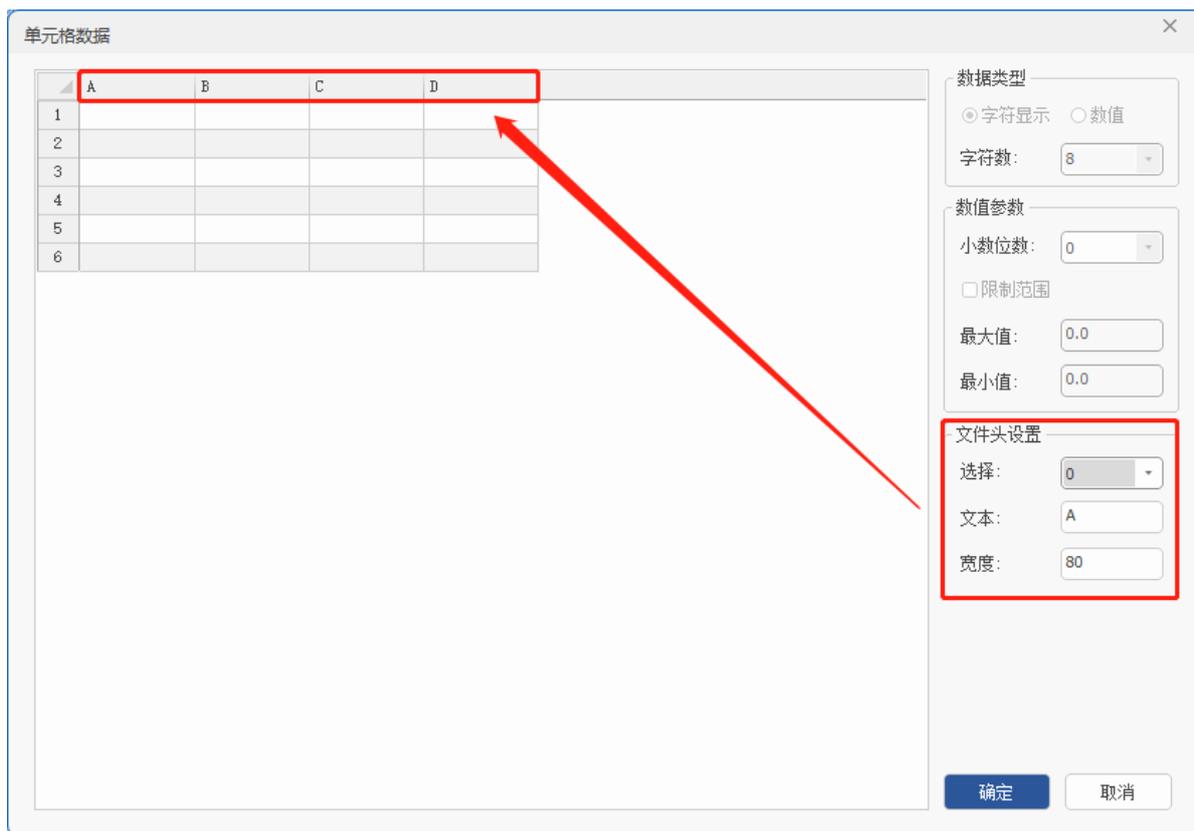
示例:

1. 在报表视图属性-外观栏设置行数和列数, 以及行高和列高, 不包括表头行和表头列。
2. 设置报表是否可编辑, 是否显示表头行/列, 行(列)的高(宽), 格式文本以及选择外框颜色、线宽等属性(在此例中设置为“可编辑”, 显示表头行/列)。

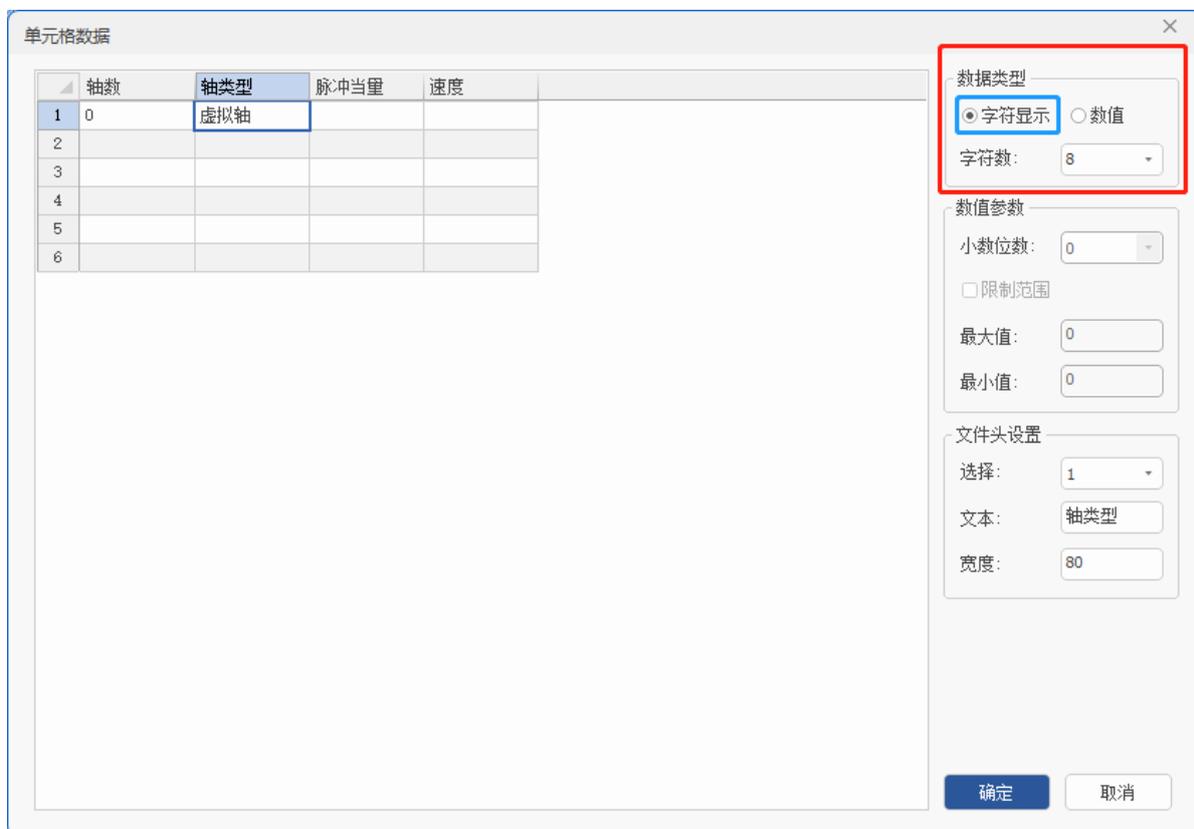


3. 点击“单元格数据”，设置文件头（表头行）文本。





4. 编写表格内容。数据类型选择“字符显示”时，只需设置字符数；数据类型选择“数值”时，还可设置该表格数值的小数位数及限制范围等数值参数。如果设置了小数位数，但小数位没写或者写的是0，则该参数的小数位只会在下载后的仿真界面显示。



5. Hmi 窗口报表视图效果图如下：



6. 连接控制器，将 Hmi 文件下载到控制器，可看到 xplc screen 窗口显示效果图。当元件属性的“固定列宽（行高）”为 false 时，在下载后的仿真界面中，用户可以通过将鼠标悬停在表头行上来调整每列的宽度，或者将鼠标悬停在表头列上来调整每行的高度。



如下图所示，双击单元格，可对表格内容进行编辑。表头行和表头列的内容只能在属性“单元格数据”中编辑，不支持在线编辑！

xplc screen

轴数	轴类型	脉冲当量	速度
1	0 (虚拟轴)	10	100.00

2	Esc 200 Backspace		
3			
4	Sym 1 2 3 4 5 6 7 8 9 0 - +		
5	Tab q w e r t y u i o p ?		
6	Caps a s d f g h j k l ' "		
	EN/CH z x c v b n m , . Enter		
	Clear Del Space ← →		

xplc screen

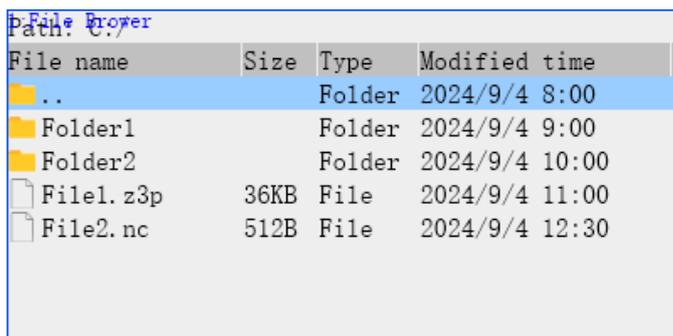
轴数	轴类型	脉冲当量	速度
1	0 (虚拟轴)	10	100.00
2	1 (脉冲方向方式的步进或伺服)	1000	500.00
3	3 (正交编码器)	1	10.00
4	6 (脉冲方向方式的编码器)	200	
5			
6			

4.3.27. 文件浏览器

介绍：显示当前目录、以表格形式显示文件内容，需要指定显示格式（必选文件名称、可选文件大小、修改时间、文件类型）。鼠标双击文件夹进入，鼠标双击文件打开文件（调用指定 SUB）。

注意：使用该控件需同时确保 RTHmi 版本为 V1.3.0 版本及以上和 RTSys 版本为 V1.2.02 及以上版本！

使用方法：点击菜单栏“HMI”→“控件箱”→“文件浏览器”。将该元件放至合适位置，调用指定 Basic 程序的全局 SUB 函数，实现文件的选择与打开。



1. 属性窗口：

属性		显示文件类型	
基本属性		显示文件类型	True
元件编号	1	文件类型列宽度	60
元件名称	File Brower1	显示文件修改时间	True
显示层次	底层	文件修改时间列宽度	150
有效显示	显示	行间距	2
采用有效控制	False	垂直滚动条	False
双击调用函数		水平滚动条	False
选中调用函数		标签	
外观		文本库	
绘制边框	True	格式文本(0)	File list ...
边框颜色	000000	格式文本(1)	Title ...
图片来源	无	格式文本(2)	Select ...
显示路径	True	位置和尺寸	
显示标题	True	水平位置	0
标题高度	20	垂直位置	131
文件名列宽度	140	宽度	399
显示文件大小	True	高度	200
文件大小列宽度	45		

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	顶层：显示在最外层，覆盖底下元件； 中层：被顶层元件覆盖，覆盖底层元件； 底层：被顶层和中间层元件覆盖（默认）。
有效显示	选择元件是否显示	显示：该元件显示于界面之中； 不显示：该元件不显示于界面之中； 仅显示，不可用：该元件可正常显示，但点击该元件时动作不生效
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 True 通过寄存器控制元件是否显示
双击调用函数	双击时触发调用 Basic 定义的 SUB 函数	函数必须是 GLOBAL 全局类型
选中调用函数	选中操作时触发调用的 Basic 定义的 SUB 函数	函数必须是 GLOBAL 全局类型
绘制边框	选择是否绘制边框	/
图片来源	无、背景图片库或背景图片	无图片、背景图片库或背景图片中选择
背景图片	背景图片选择	图片来源设置为选择背景图片显示
背景图片库	背景图片库选择	图片来源设置为选择背景图片库显示
显示路径	是否显示文件路径	默认为 True
显示标题	是否显示标题	默认为 True
标题高度	标题的高度	/
文件名列宽度	文件名列的宽度	/
显示文件大小	是否显示文件大小	默认为 True
文件大小列宽度	文件大小列的宽度	/
显示文件类型	是否显示文件类型	默认为 True
文件类型列宽度	文件类型列的宽度	/
显示文件修改时间	是否显示文件修改时间	默认为 True
文件修改时间列宽度	文件修改时间列的宽度	/
行间距	显示每一行的上下间距，包括文件列表及当前路径行 如字体大小=16，行间距=4，则每一行高度=16+2*4=24	默认为 2，可设置为 0~100
垂直滚动条	设置是否使用垂直方向滚动条	/
水平滚动条	设置是否使用水平方向滚动条	/
格式文本 0 (File list)	设置文件列表行的字体格式，包括对当前路径行的设置	文本名称不可更改
格式文本 1 (Title)	设置当前文件列表标题行的字体格式	文本名称不可更改
格式文本 2 (Select)	设置当前文件列表选中行的字体格式	文本名称不可更改
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	/

初级的三次文件编辑器使用可以参照第八章【[文件浏览器使用](#)】。

4.3.28. 菜单

介绍：类似 window 风格的菜单栏，**最多只能嵌套 5 层菜单**。点击弹出菜单项，每一个菜单项可以设置是否禁用（不使能则灰阶显示），点击菜单项可触发调用 SUB 动作，每一个菜单项还可以设置是否选中状态。鼠标悬停处的菜单项将处于高亮状态。可在任意菜单项上面增加分隔符。

注意：使用该控件需同时确保 RTHmi 版本为 V1.3.0 版本及以上和 RTSys 版本为 V1.2.02 及以上版本！

使用方法：点击菜单栏“HMI”→“控件箱”→“菜单”。将该元件放至合适位置，打开菜单属性，设置列表文本内容；然后根据各项的菜单编号在 Basic 文件中编写与之对应的全局 SUB 函数，以实现各菜单项的具体操作；最后，在菜单属性中调用已经编写好的 SUB 函数，以完成菜单功能的绑定与实现。



1. 属性窗口：

2. 属性说明：

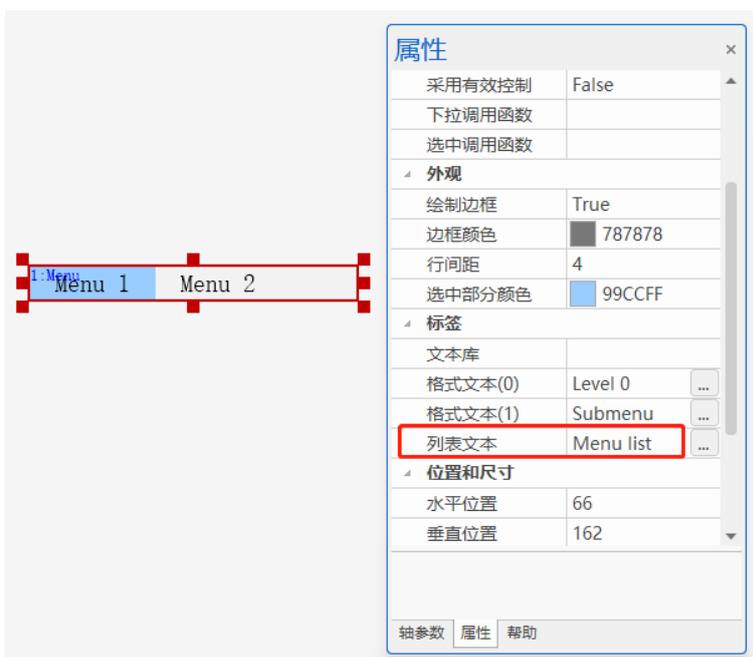
属性	功能	说明
元件编号	/	/
元件名称	/	/
显示层次	选择元件显示层次	顶层：显示在最外层，覆盖底下元件； 中层：被顶层元件覆盖，覆盖底层元件； 底层：被顶层和中间层元件覆盖（默认）。

有效显示	选择元件是否显示	显示：该元件显示于界面之中； 不显示：该元件不显示于界面之中； 仅显示，不可用：该元件可正常显示，但点击该元件时动作不生效
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 True 通过寄存器控制元件是否显示
下拉调用函数	进行下拉操作时触发调用 Basic 定义的 SUB 函数 注：可在该动作函数内对子菜单项的灰阶状态、选中状态进行管理	函数必须是 GLOBAL 全局类型
选中调用函数	弹出菜单项区域时，点击选中任意一个有效的菜单项时触发调用的函数 注：可在该动作函数判断最后点击选中项并进行相应的动作绑定动作 SUB_XXXX，但定义动作时需要传入一个 ID 参数，如 SUB_XXXX(ID)，ID 表示最后点击选中菜单项的编号。	函数必须是 GLOBAL 全局类型
绘制边框	选择是否绘制子菜单的边框	/
边框颜色	选择子菜单边框的颜色	默认灰色
行间距	设置子菜单行与行之间的距离	每一行菜单项上下的行距，例如字体大小=16，行距=4，那么每一行菜单项的行高等于 $16+4*2=24$ 默认为 4，可设为 0~100
选中部分颜色	主菜单按钮和所有子菜单项的选中高亮背景颜色 鼠标悬停在菜单按钮或菜单项时，背景颜色进行高亮显示	默认蓝色
文本库	文本库的名称，如果为空，则表示使用文本标签	/
格式文本 0 (Level 0)	主菜单按钮的显示格式 在 RTSys 传入的控件大小即为菜单按钮的大小，菜单项区域大小是根据菜单项数量，即相关属性自动计算并显示的	文本名称不可更改
格式文本 1 (Submenu)	子菜单项的显示格式，包括所有级别的子菜单项 该格式文本的背景颜色作为整个弹出菜单区域的背景颜色	文本名称不可更改

	限制只能左对齐，不支持右对齐、居中对齐	
列表文本 (Menu list)	设置各项菜单项的文本	/
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	/

示例:

1. 在菜单属性 - 标签栏点击 “列表文本” 打开。

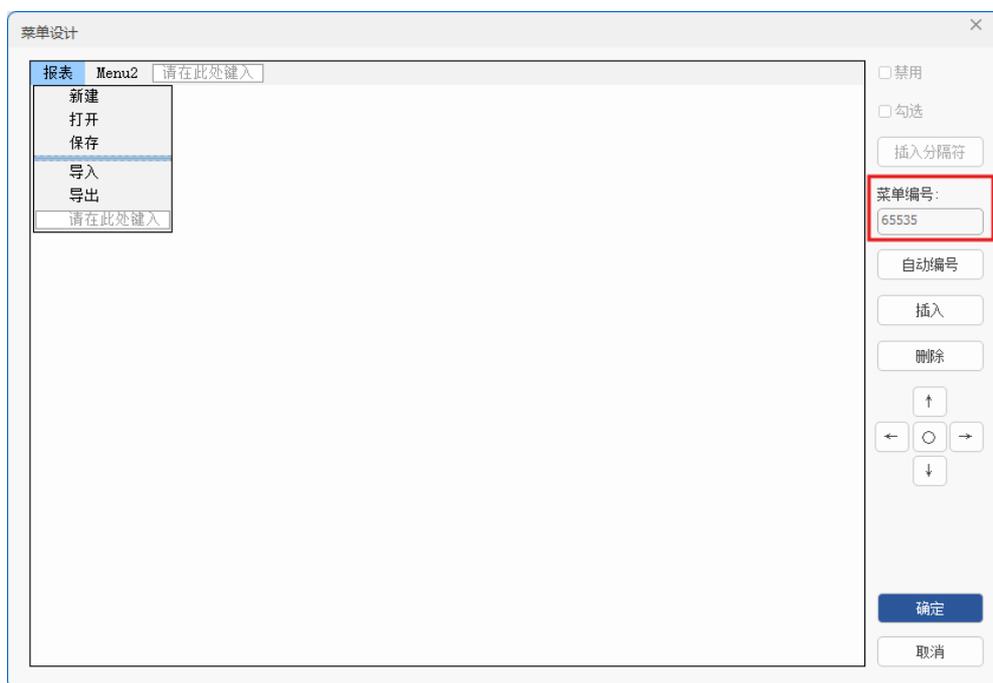
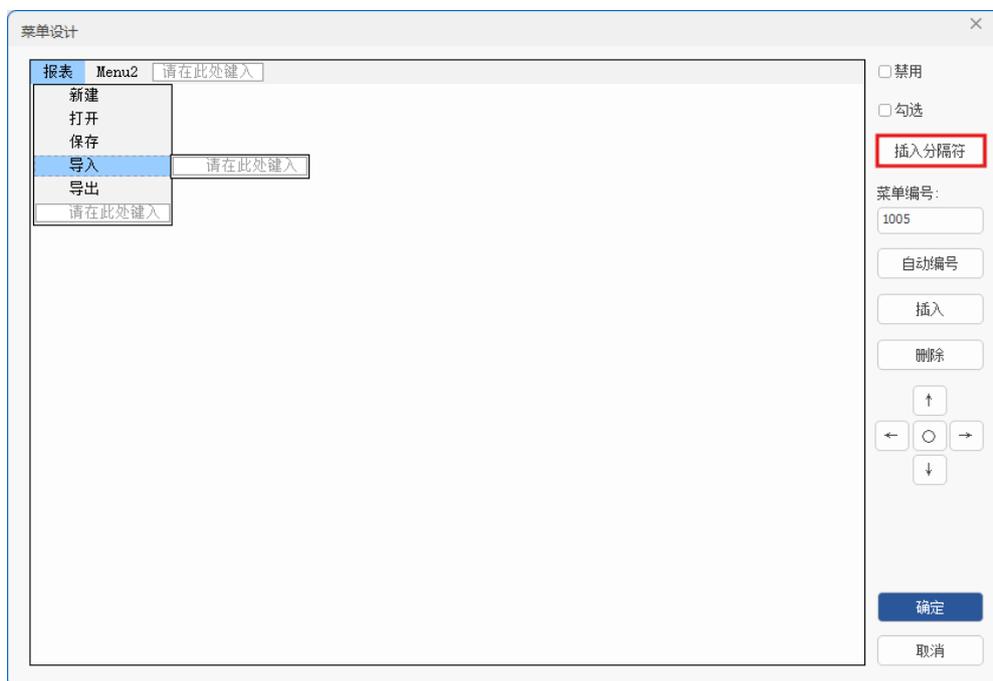


2. 编辑菜单项内容，每项都对应唯一的菜单编号，菜单编号按其编写顺序编号。

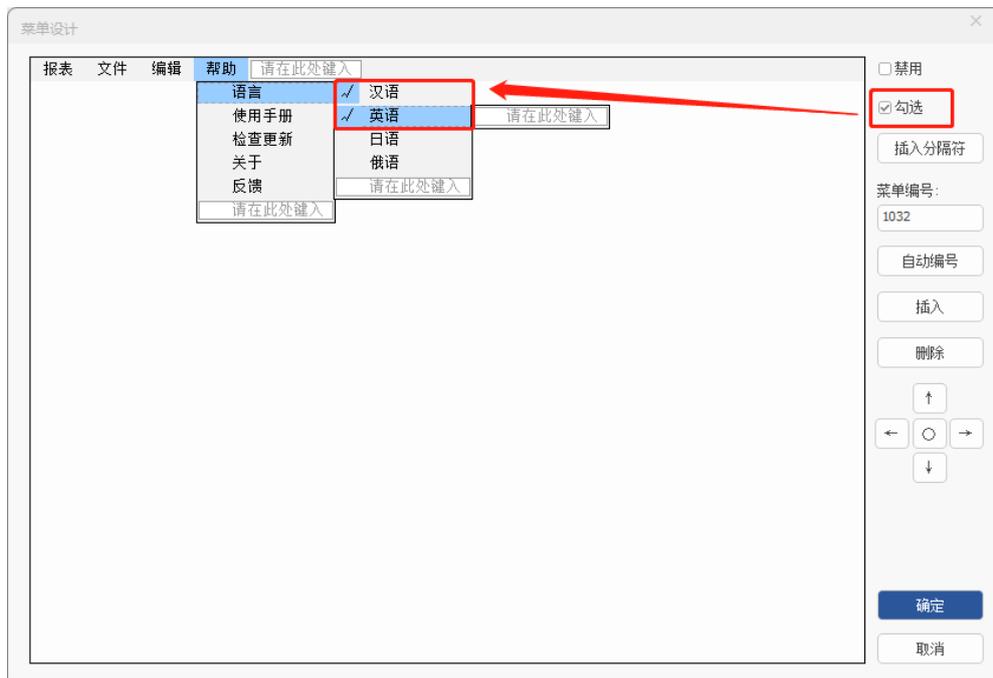


注意：菜单编号支持手动设置和自动设置，但所用的编号不可重复！菜单编号可设置的最大值为：65535。“自动编号”功能会自动寻找未使用的编号，可保证编号不重复，但手动设置则无法检测是否有冲突。

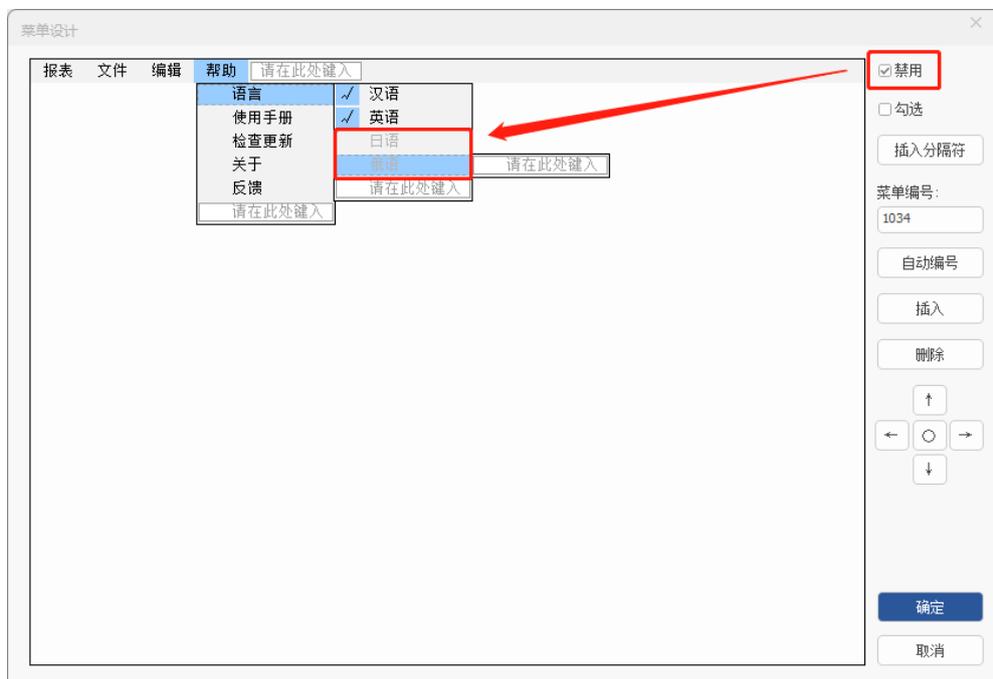
选中一项，点击“插入分隔符”，分隔符将插入选中项的上方，其菜单编号为固定值：65535。



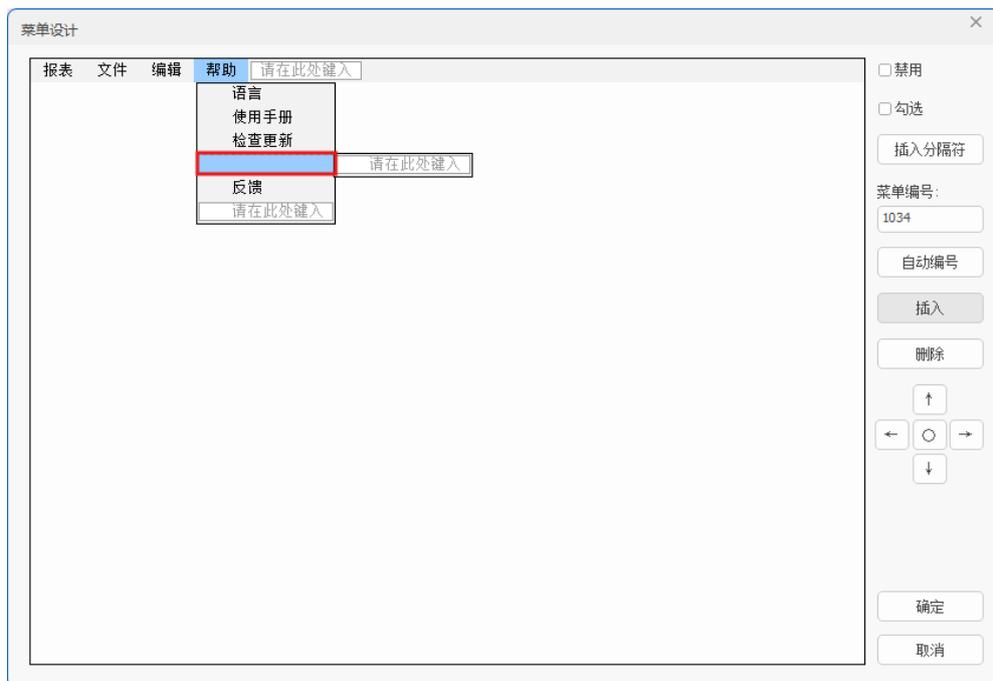
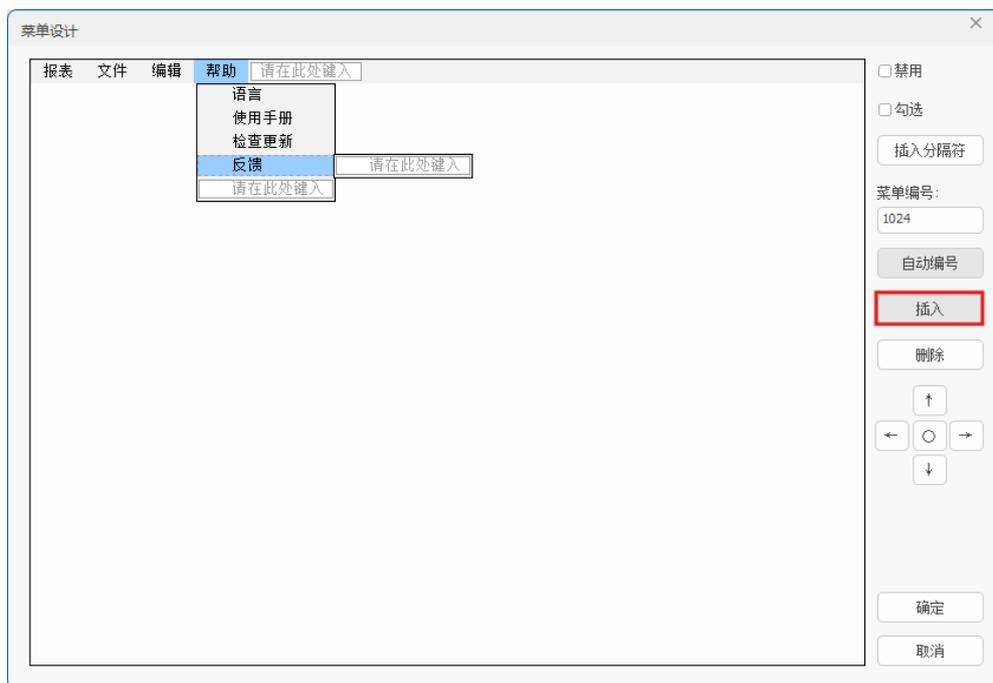
选中一项，点击“勾选”，选中项前面会出现一个“√”。也可直接通过控件操作指令 [HMI_MENUITEM](#) 中的参数对其进行设置，“设置选中状态”，下载实际操作时可对此项进行勾选。



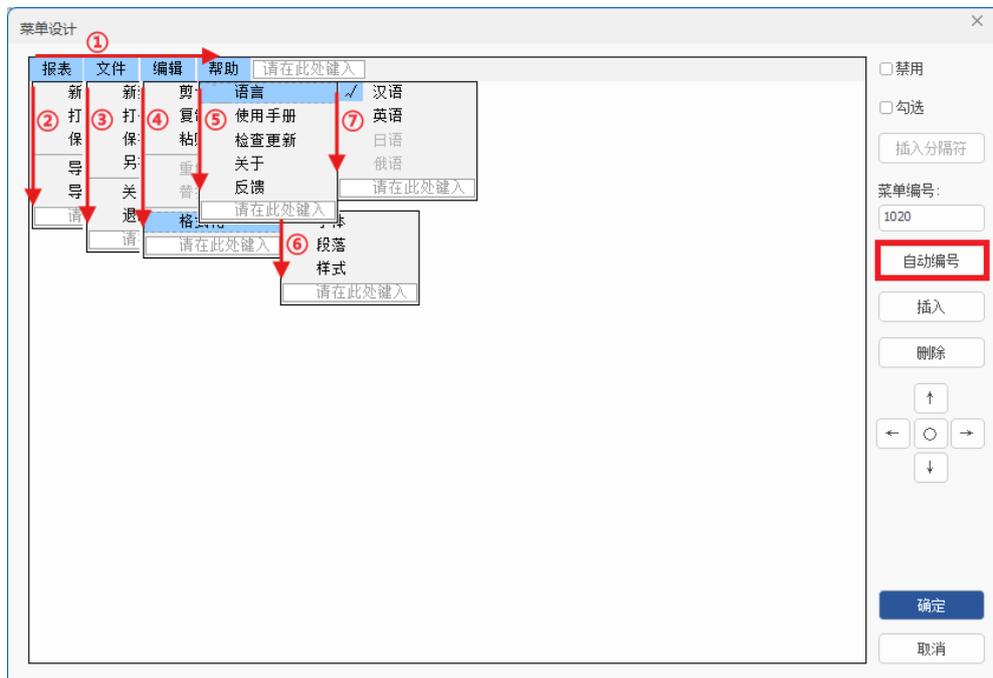
选中一项，点击“禁用”，选中项将变为灰色，实际操作时无法选中。也可直接通过控件操作指令 [HMI_MENUITEM](#) 中的参数对其进行设置，“设置灰阶状态”。



选中一项，点击“插入”，插入项将插入选中项的上方。



编辑完成后，若点击“自动编号”，系统首先将会按照从左至右的顺序对主菜单进行编号，随后再依次从左至右、从上至下对下一级子菜单进行编号，如此逐级类推，直至全部编号完成，图示如下：



3. 菜单各项内容编辑完成后，点击“确认”，Hmi 界面只会显示主菜单项。根据需求，在菜单基本属性中选择对应 Basic 全局 SUB 子函数调用。相关指令可查看“[HMI_MENUITEM](#)”。



“选中调用函数” Basic 全局 SUB 函数举例：



```
1      end ''结束主程序
2
3      ''菜单项选中函数（部分功能演示）
4      ''传入一个参数（菜单编号）
5      global sub Sub_MenuSelect(menuid)
6
7          ?"点击了菜单项",menuid
8
9          ''处理菜单项点击事件
10         if menuid = 1005 then
11             HMI_SHOWWINDOW(11) ''打开报表
12
13         elseif menuid = 1027 then
14             HMI_SHOWWINDOW(12) ''打开文件浏览器
15
16         elseif menuid >= 1031 AND menuid <= 1034 then ''选择语言
17
18             ''语言只能选择一种，取消其他的，再选中当前的
19             HMI_MENUITEM(1031, 0, 0)
20             HMI_MENUITEM(1032, 0, 0)
21             HMI_MENUITEM(1033, 0, 0)
22             HMI_MENUITEM(1034, 0, 0)
23             HMI_MENUITEM(menuid, 0, 1)
24
25         endif
26     end sub
27
28
29
```

源代码：

```
end "结束主程序

"菜单项选中函数（部分功能演示）
"传入一个参数（菜单编号）
global sub Sub_MenuSelect(menuid)
    ?"点击了菜单项",menuid
    "处理菜单项点击事件
    if menuid = 1005 then
        HMI_SHOWWINDOW(11) "打开报表
    elseif menuid = 1027 then
        HMI_SHOWWINDOW(12) "打开文件浏览器
    elseif menuid >= 1031 AND menuid <= 1034 then "选择语言
        "语言只能选择一种，取消其他的，再选中当前的
        HMI_MENUITEM(1031, 0, 0)
        HMI_MENUITEM(1032, 0, 0)
        HMI_MENUITEM(1033, 0, 0)
        HMI_MENUITEM(1034, 0, 0)
        HMI_MENUITEM(menuid, 0, 1)
    endif
end sub
```

4. 连接控制器，将 Hmi 文件下载到控制器，可看到 xplc screen 窗口显示效果图。

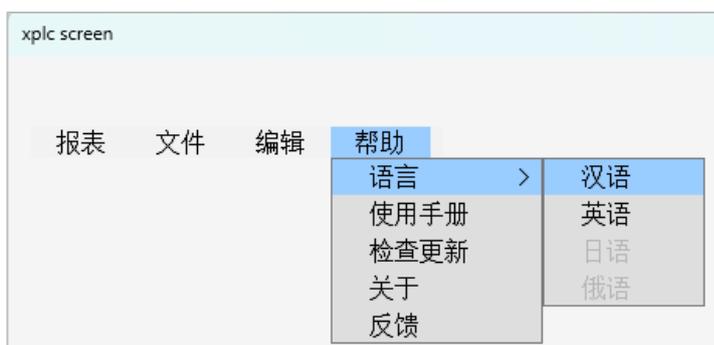


打开报表：

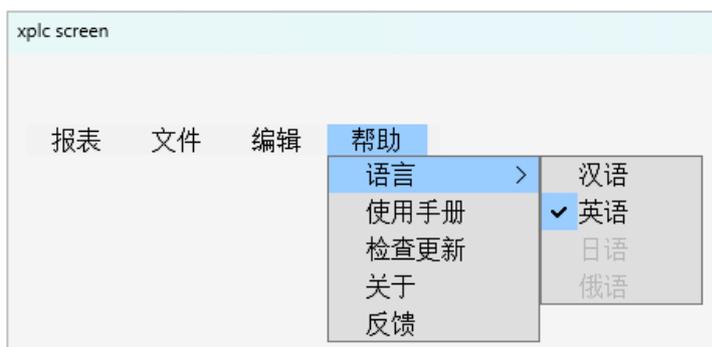
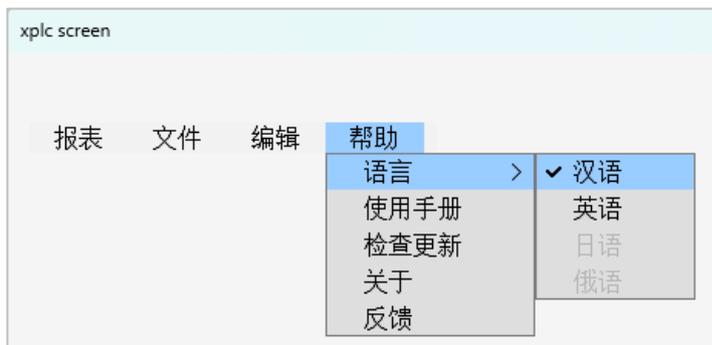


	配方名	面粉/g	水/ml	鸡蛋/个	盐/g	蜂蜜/g
1	甜面包	100.0	50.0	3	5.0	35.0
2	无糖面包	100.0	65.0	2	3.5	0.0
3	全麦面包	100.0	55.0	0	5.0	0.0
4						
5						
6						
7						
8						

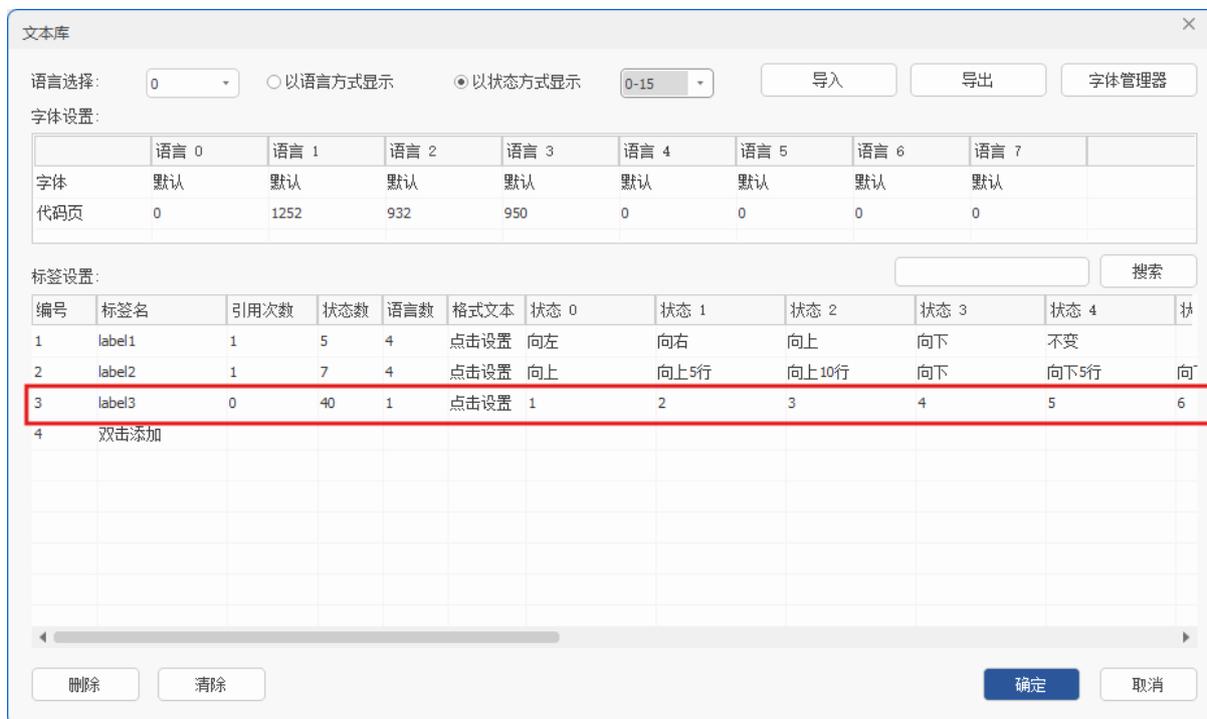
选择语言：

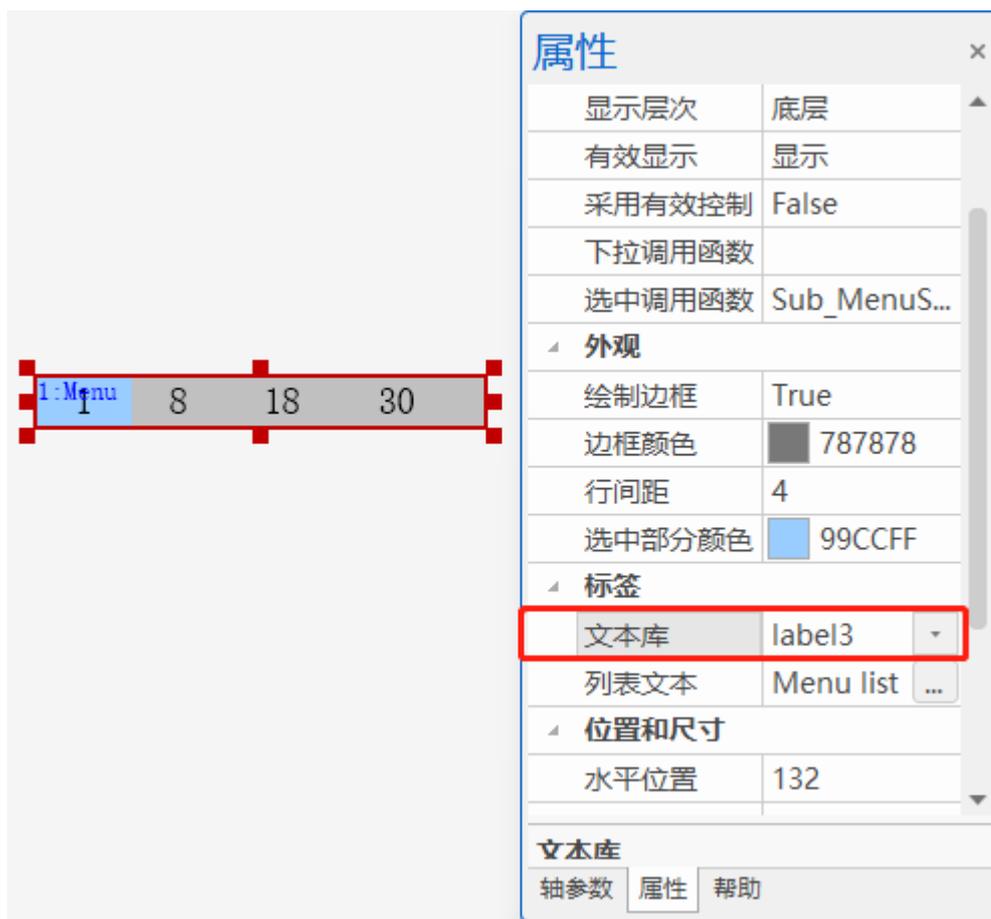


程序中对语言选择进行了限制，即一次只能选择一种，选中了一种语言后，换成另一种语言时，会将当前选中项置为“未选中”状态。被禁用的菜单项无法选中。

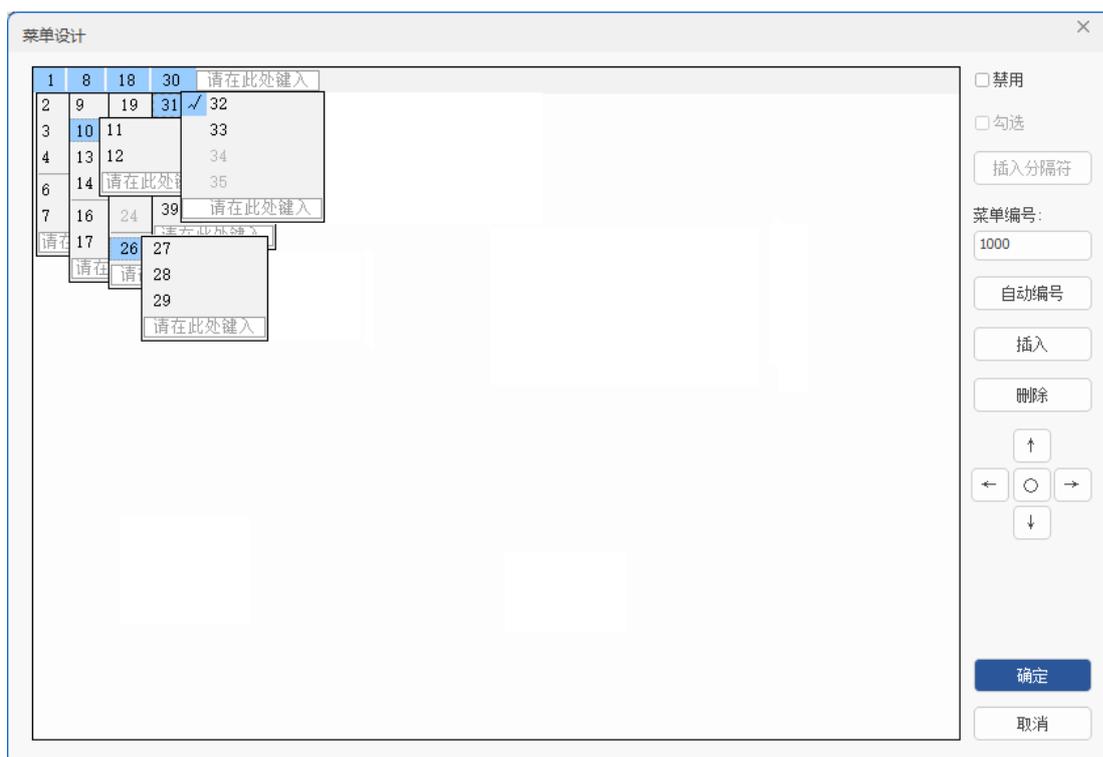


5. 菜单控件的文本设置若使用文本库，则其内容将按照层次结构依次填充至各菜单项中。





如下图所示，当使用文本库时，其状态内容将从主菜单的第一项开始，依次填充至该项下的所有子菜单中（分隔符也占一个文本库状态），逐层深入直到填满后，再填入主菜单的第二项，以此类推。



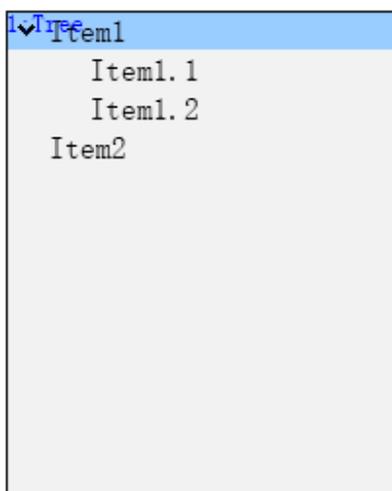
4.3.29. 树形图

介绍：树形图是一种以树状结构展示所有表项（树形图第一层的树节点）的控件，支持单击角图标展开/缩起子树，单击树节点内容触发动作。当树节点数量超出显示范围时，自动启用垂直滚动条，便于用户浏览和管理。

注意：使用该控件需同时确保 RTHmi 版本为 V1.3.0 版本及以上和 RTSys 版本为 V1.2.02 及以上版本！

当前版本的树形图控件不支持在下载后的仿真界面动态修改、添加、清空树节点，只能在 Hmi 文件内设计好所有的树节点。

使用方法：点击菜单栏“HMI”→“控件箱”→“树形图”。将该元件放至合适位置，在列表文本里进行树形图的结构设计，每一树节点都绑定一个唯一的 ID。选中树节点时，其关联的寄存器（状态）值将自动设为该项 ID；修改寄存器值时，对应 ID 的树节点会被选中。通过树形图属性框“选中调用函数”，调用相应动作的全局 SUB 函数，实现单击树节点内容进行触发动作的功能。



3. 属性窗口：

属性		角标类型	样式1
基本属性		角标颜色	000000
元件编号	1	行间距	2
元件名称	Tree1	选中部分颜色	99CCFF
显示层次	底层	标签	
有效显示	显示	文本库	
采用有效控制	False	格式文本	Item
绑定的设备编号	本地	列表文本	Item
绑定的寄存器类型	D	位置和尺寸	
绑定的寄存器编号	0	水平位置	180
选中调用函数		垂直位置	70
外观		宽度	200
绘制边框	True	高度	250
边框颜色	000000		
图片来源	无		

轴参数 属性 帮助

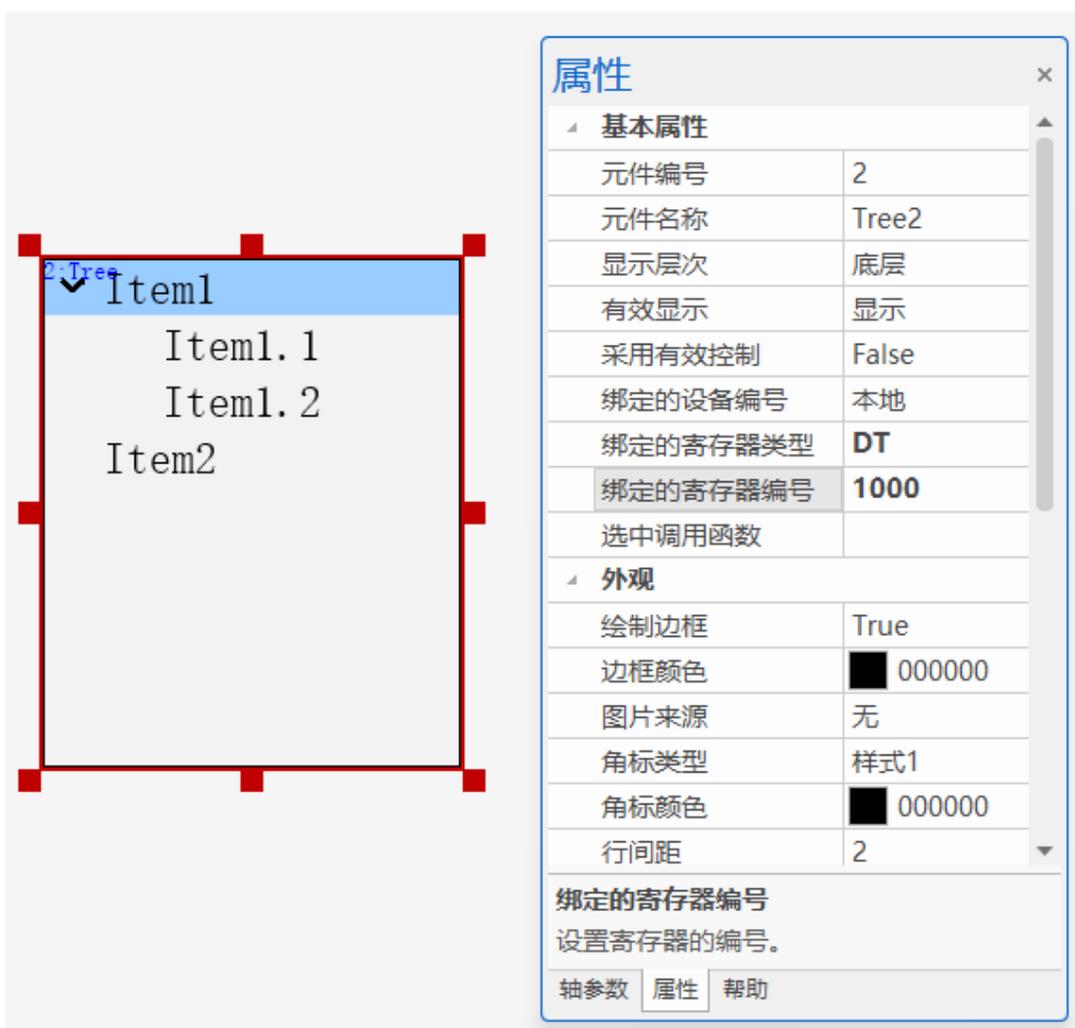
4. 属性说明:

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	顶层：显示在最外层，覆盖底下元件； 中层：被顶层元件覆盖，覆盖底层元件； 底层：被顶层和中间层元件覆盖（默认）。
有效显示	选择元件是否显示	显示：该元件显示于界面之中； 不显示：该元件不显示于界面之中； 仅显示，不可用：该元件可正常显示，但点击该元件时动作不生效
采用有效控制	通过寄存器控制元件是否显示	默认 False，选择 True 通过寄存器控制元件是否显示
绑定的设备编号	指定设备	默认 local
绑定的寄存器类型	选择寄存器类型	多种寄存器下拉列表选择
绑定的寄存器编号	设置寄存器的编号	通过获取寄存器不同数值，控制元件不同状态
选中调用函数	指定点击选中选项时触发调用的 SUB 函数	函数必须是 GLOBAL 全局类型
绘制边框	选择是否绘制边框	为 TRUE 时有属性：边框颜色
边框颜色	选择边框的颜色	默认灰色
图片来源	背景图片或背景图片库	/
背景图片	背景图片选择	图片来源设置为选择背景图片显示
背景图片库	背景图片库选择	图片来源设置为选择背景图片库显示
角标类型	父树角标的显示类型，支持样式 1~样式 3	缩起时：> ▶ ▷ 展开时：▼ ▼ ▣
角标颜色	显示角标的颜色	
行间距	设置行与行之间的距离	每一行菜单项上下的行距，例如字体大小=16，行距=4，那么每一行菜单项的行高等于 16+4*2=24 默认为 2，可设为 0~100
选中部分颜色	鼠标悬浮在树表项上或树表项被选中时，该项高亮显示的颜色	默认蓝色
文本库	文本库的名称，如果为空，则表示使用文本标签	/
格式文本 (Item)	设置树形图标签的样式	文本名称不可更改
列表文本 (Item)	所有树（父树和子树）节点的显示文本及 ID，当前选中项会关联树节点的 ID 编号	/
水平位置	元件的水平起始位置	不要超出水平分辨率

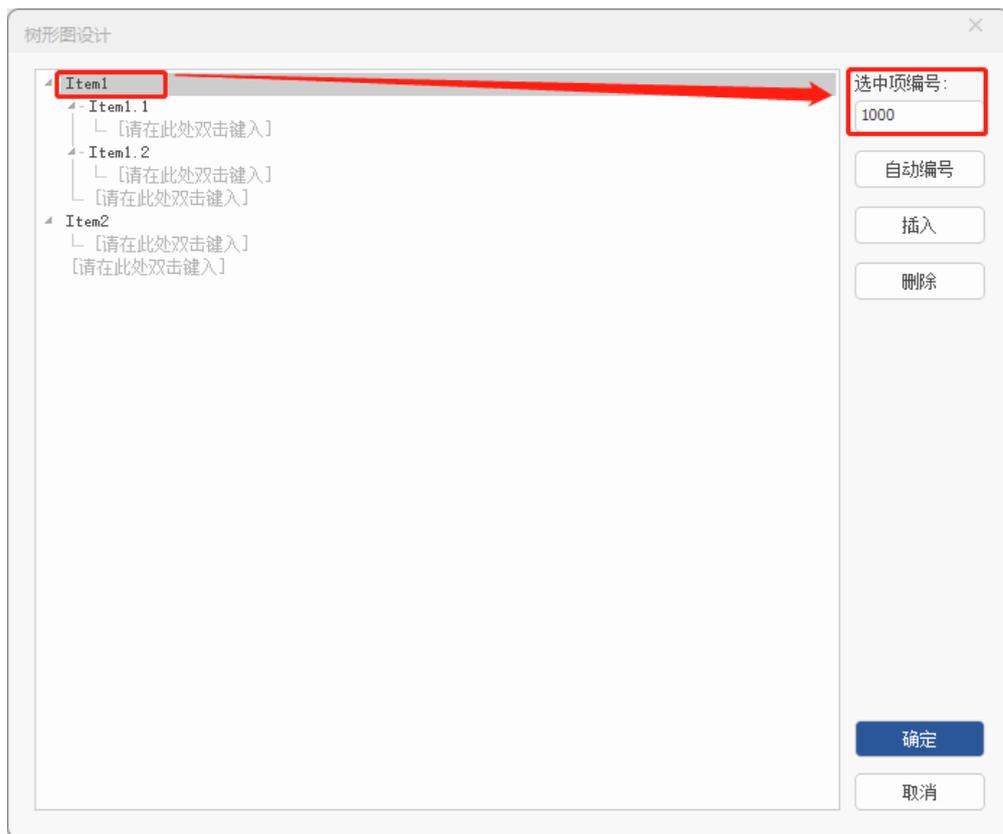
垂直位置	元件的垂直起始位置	不要超出垂直分辨率
宽度	元件的宽度	/
高度	元件的高度	/

示例：

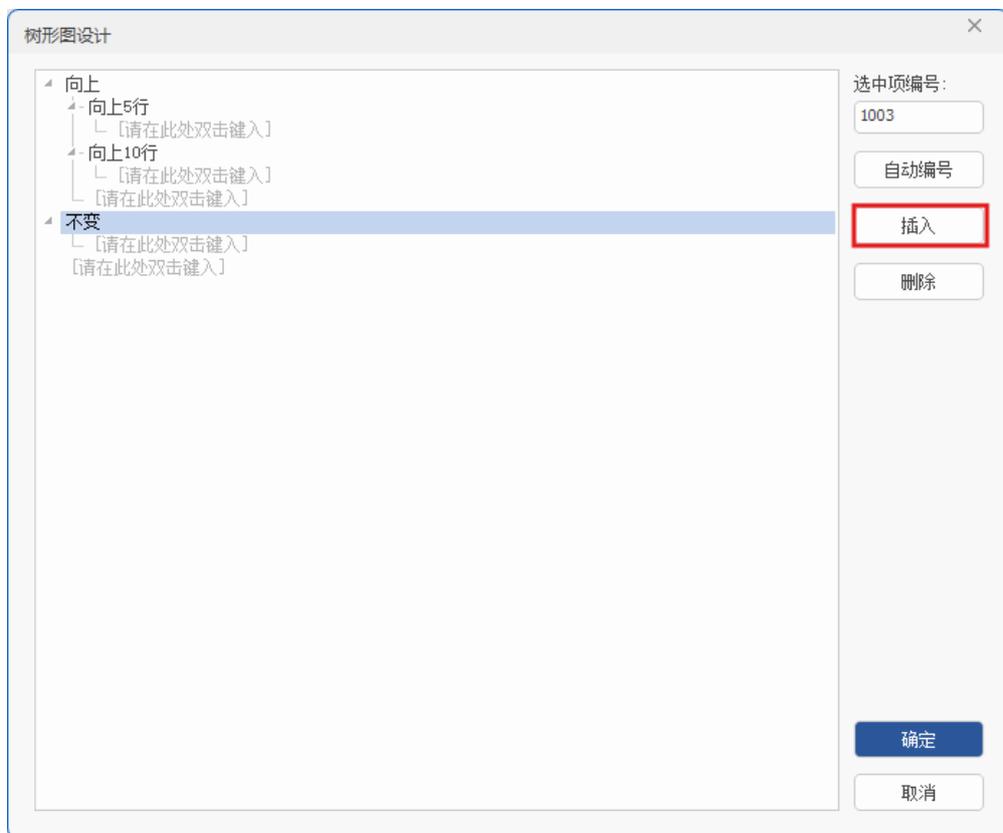
1. 在树形图属性-基本属性栏选择寄存器类型和编号（在此例中将“绑定的寄存器类型”设为 DT，“绑定的寄存器编号”设为 1000，即绑定的寄存器地址为 **TABLE (1000)**）。



2. 点击树形图属性框下的“列表文本”，设计树节点结构并编写其文本内容。每一树节点都绑定一个唯一的 ID，树节点的 ID 按其编写顺序编号。编辑完成后，若点击“自动编号”，则表项 ID 将从上往下依次进行编号。

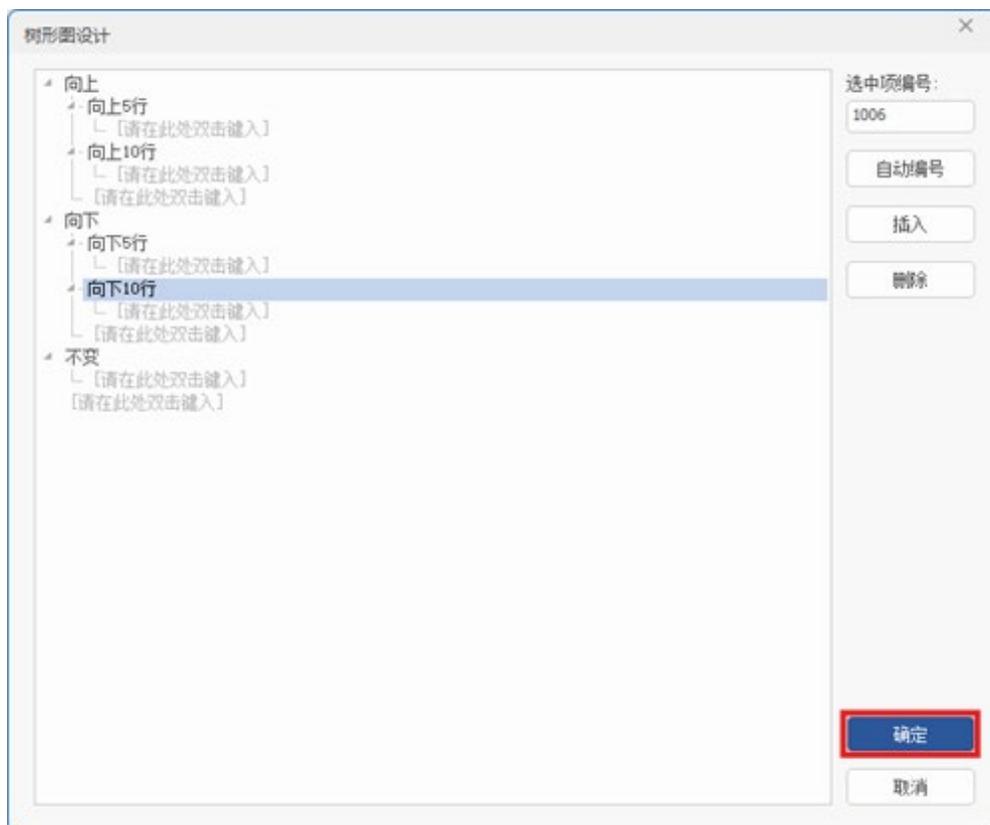


“插入”操作是指在选中项的上方添加与其同级别的一个新项。





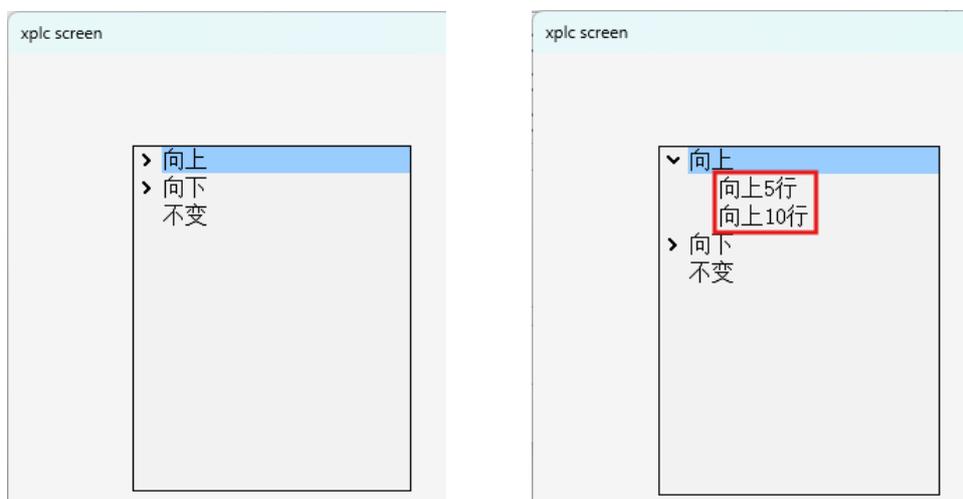
3. 编辑完成后，点击“确定”。



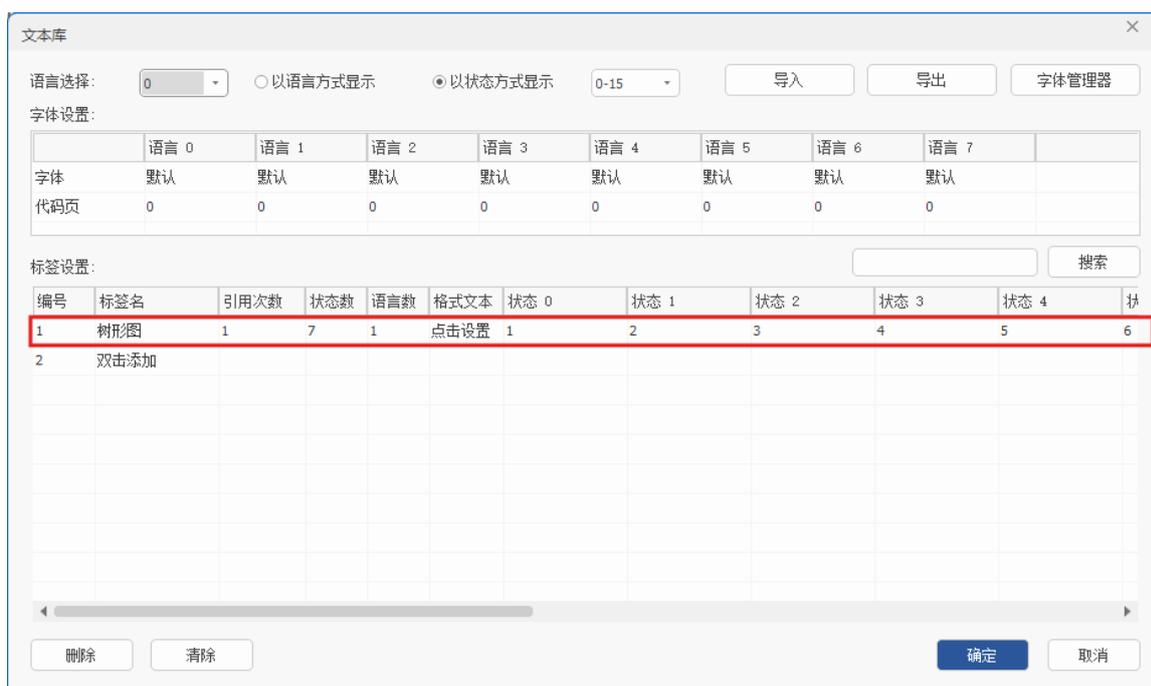
4. Hmi 窗口树形图效果图如下：

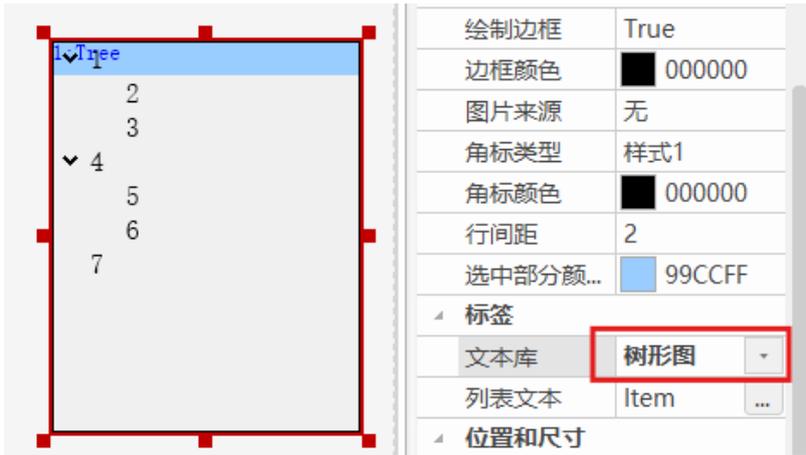


5. 连接控制器，将 Hmi 文件下载到控制器，可看到 xplc screen 窗口显示效果图，如左下图所示，只显示第一层的树节点；点击“>”将显示该父树对应的子树项，如右下图所示。



6. 若使用文本库，则其内容将按照层次结构依次填充至树形图中，从顶层的第一个标题开始，逐级向下展开至第二个标题，乃至更深层次的子标题。





7. 选择的状态显示颜色、行间距、弹出方式等可通过“属性”窗口进行设置。
8. 树形图调用 Basic 文件全局 SUB 函数的具体使用请参照第八章【[文件浏览器使用](#)】。

第五章 Hmi 调用 Basic 函数

本章主要介绍 Hmi 调用 Basic 函数的方法以及多种不同函数调用的介绍。

函数调用基本通用方法：

1. 在 Basic 文件中定义全局（GLOBAL）SUB 子函数，定义语法：

```
global sub 子函数名()
end sub
```

函数名和括号必须为英文字符。如：GLOBAL SUB redraw() END SUB

2. 在已定义好的 SUB 子函数中，编写该函数体的程序。

3. 打开/新建 Hmi 文件，添加对应元件，打开对应属性窗口，“动作”选择“调用函数”，在动作函数名处选择需对应调用的函数。

4. 若是 Hmi 设置或自定义元件，在属性窗口中“xx 函数”处直接调用即可。

5.1. Hmi 系统设置调用函数

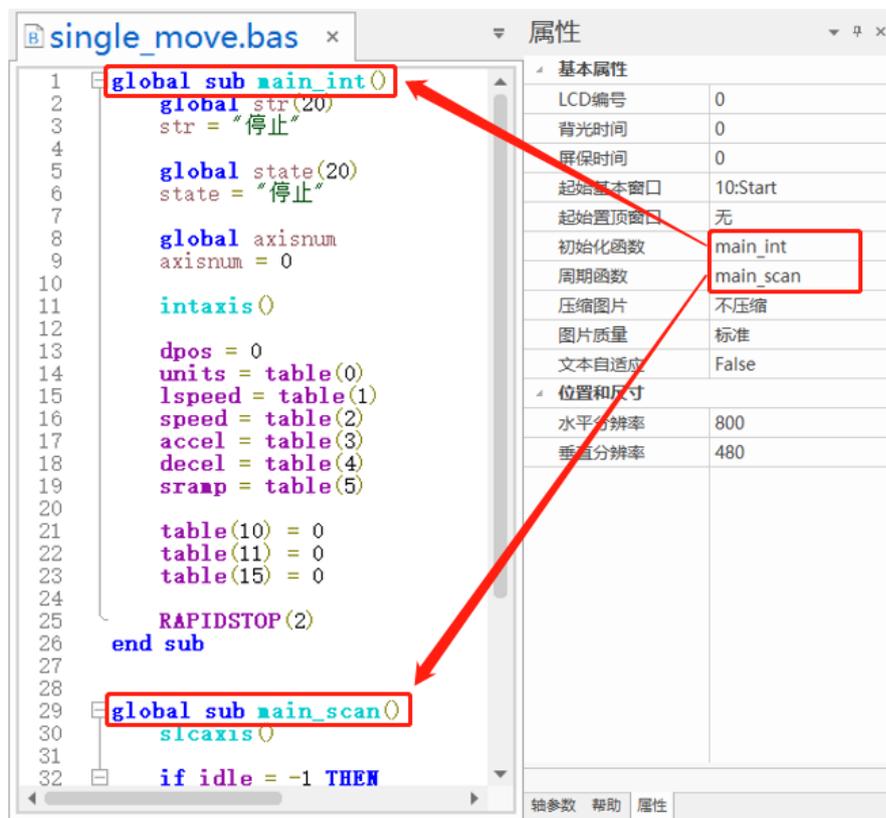
菜单栏“Hmi 系统设置”打开如下窗口，初始化函数和周期函数的调用可根据需求选择是否设置。



初始化函数： 上电后只调用一次的函数，一般将初始化的相关参数定义等内容写入定义好的初始化全局 SUB 子函数，并在 Hmi 设置中调用。

周期函数：上电后周期不断扫描的函数。

注意：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。

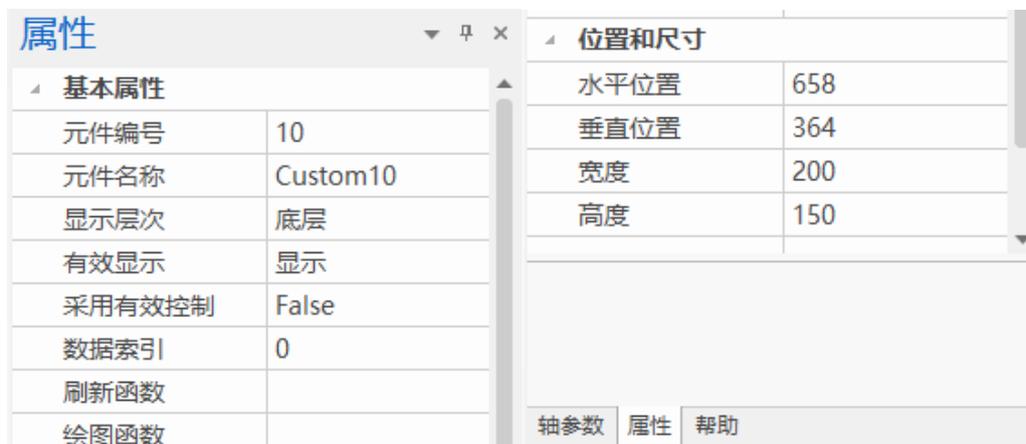


5.2. 自定义元件调用函数

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。

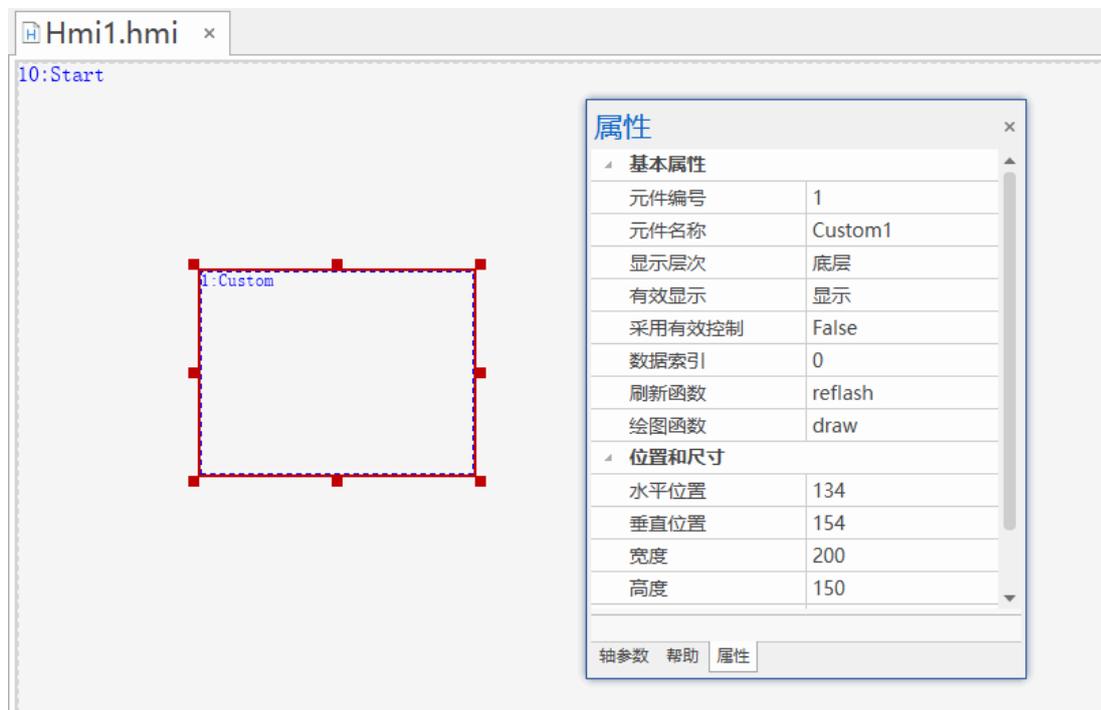
绘图函数：需要绘图时自动被调用，此函数里面通过调用 DRAW/DRAWEX 相关函数来自己绘图，绘图函数的零点是自定义元件的左上角。

刷新函数：周期调用来判断是否要重新绘图（系统会自动周期性调用刷新函数），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 程序:

```

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

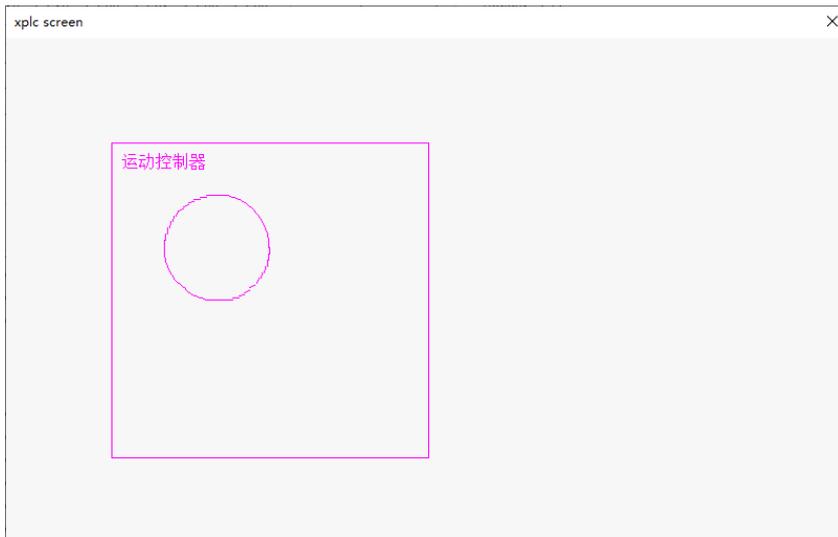
    DRAWRECT(0,0,300,300)        '自定义元件内绘制边框

    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

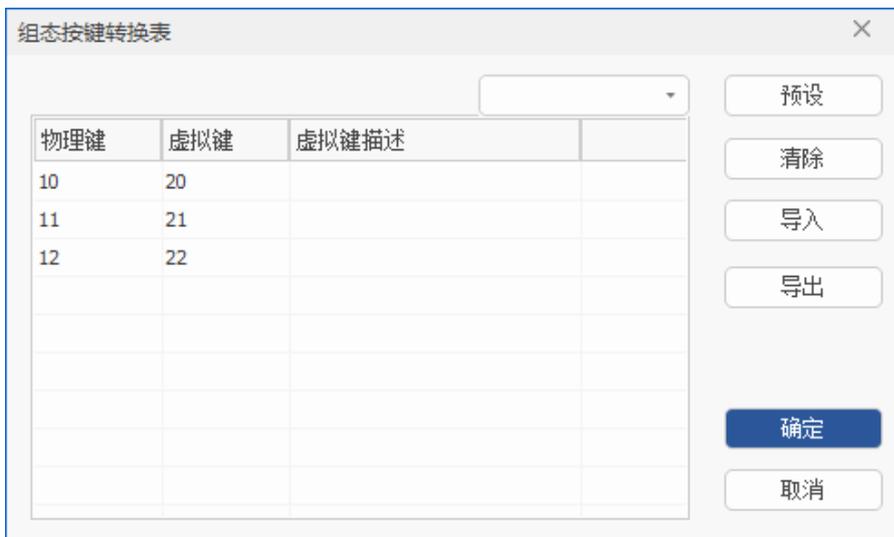
END SUB
    
```

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()
    if num=20 then
        print 1
    elseif num=21 then
        table(10)=100
    elseif num=22 then
        function1()
    endif
end sub

global sub slt()
    if VKKEY_SCAN<>0 then
        num=VKKEY_SCAN
    endif
    SET_REDRAW
end sub
    
```

'绘图函数
 '物理按键10按下
 '命令行打印 1
 '物理按键11按下
 'table10赋值为100
 '物理按键12按下
 '调用自定义函数
 '刷新函数
 '扫描虚拟按键，返回值给变量num

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。

5.3. 元件调用函数

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“调用函数”，“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。



第六章 相关 Basic 指令

以下是 Hmi 增加的 Basic 指令。

6.1. 基础指令

6.1.1. RUN -- 启动文件任务

类型	任务指令
描述	<p>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。</p> <p>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</p>
语法	<p>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</p> <p>RUN "file.Hmi", TASKID, [LCDNUM] filename: 程序文件名, BAS 文件可不加扩展名 TASKID: 任务号</p>
适用控制器	通用
例子	RUN "aaa", 1 '启动任务 1 运行 aaa.bas 文件

6.1.2. SCAN_EVENT -- 数据状态变化扫描

类型	输入输出函数								
描述	<p>数据变化扫描。</p> <table border="1"> <thead> <tr> <th>返回值</th> <th>描述</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>数据 0 变为非 0</td> </tr> <tr> <td>-1</td> <td>数据非 0 变为 0</td> </tr> <tr> <td>0</td> <td>BOOL 数据没有变化</td> </tr> </tbody> </table>	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
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	<p>event = SCAN_EVENT (condition) condition: 数据条件表达式</p>								

适用控制器	通用
例子	<pre> 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 </pre>

6.1.3.SET_XPLCTERM -- 显示屏启动状态设置

类型	显示操作指令
描述	读取显示屏状态。
语法	<p>SET_XPLCTERM=value</p> <p>value: 0-缺省, 不启动 1-自动启动全屏的 XPLCTERM HDMI 版本</p>
适用控制器	仅支持 5 系列控制器
例子	SET_XPLCTERM=1 ‘自动启动全屏 XPLCTERM

6.1.4.SYSTIME -- 系统时间

类型	系统时间
描述	获取系统当前时间
语法	<p>string= SYSTIME(style)</p> <p>style: 时间格式, ”%Y/%m/%d %H:%M:%S” => 2021/08/14/ 11:03</p> <p>返回时间字符串</p> <p>时间格式:</p> <ul style="list-style-type: none"> %a 星期几的简写 %A 星期几的全称 %b 月份的简写 %B 月份的全称 %c 标准的日期的时间串 %C 年份的前两位数字 %d 十进制表示的每月的第几天 %D 月/天/年 %e 在两字符域中, 十进制表示的每月的第几天 %F 年-月-日 %g 年份的后两位数字, 使用基于周的年 %G 年份, 使用基于周的年 %h 简写的月份名

	<p>%H 24 小时制的小时</p> <p>%I 12 小时制的小时</p> <p>%j 十进制表示的每年的第几天</p> <p>%m 十进制表示的月份</p> <p>%M 十进制表示的分钟数</p> <p>%n 换行符</p> <p>%p 本地的 AM 或 PM 的等价显示</p> <p>%r 12 小时的时间</p> <p>%R 显示 24 小时制的小时和分钟: hh:mm</p> <p>%S 十进制的秒数</p> <p>%t 水平制表符</p> <p>%T 显示时分秒: hh:mm:ss (24 小时制)</p> <p>%u 每周的第几天, 星期一为第一天 (值从 1 到 7, 星期一为 1)</p> <p>%V 每年的第几周, 使用基于周的年</p> <p>%w 十进制表示的星期几 (值从 0 到 6, 星期天为 0)</p> <p>%U 每年的第几周, 把星期日做为第一天 (值从 0 到 53)</p> <p>%W 每年的第几周, 把星期一做为第一天 (值从 0 到 53)</p> <p>%x 标准的日期串</p> <p>%X 标准的时间串</p> <p>%y 不带世纪的十进制年份 (值从 0 到 99)</p> <p>%Y 带世纪部分的十进制年份</p> <p>%z, %Z 时区名称, 如果不能得到时区名称则返回空字符。</p> <p>%% 百分号</p>
适用控制器	支持 4 系列和 5 系列以上控制器
例子	<p>例一: ?SYSTIME(“%Y/%m/%d %H:%M:%S”)</p> <p>2021/11/12 11:30</p> <p>例二: ?”日期: ”, SYSTIME(“%x”)</p> <p>?”每周第几天: ”, SYSTIME(“%u”)</p> <p>打印结果如下:</p> <p>日期: 07/13/22</p> <p>每周第几天: 3</p>

6.2. 语法指令

6.2.1.DMSET -- 数组区域赋值

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

6.2.2.ZINDEX_LABEL -- 建立索引指针

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

6.2.3.ZINDEX_ARRAY -- 访问数组

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

6.2.4.ZINDEX_CALL -- SUB 函数调用

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

6.2.5.ZINDEX_VAR -- 指针变量操作

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

6.2.6.ZINDEX_MARK -- 指针标号设置

类型	语法指令
描述	给 label 设置标号，从而可以把 label 的数组索引记录下来。
语法	ZINDEX_MARK(labelname) = mark varmark = ZINDEX_MARK(labelname) labelname: 如果不同文件都有定义，注意不同文件里面调用的是不同的结果 mark: 标号
适用控制器	支持 ZV 功能或者 5 系列以上的控制器
例子	dim var1 dim arr1(10),MarkArr(10) dim ArrIndex,VarIndex

	<pre>arr1(0,1,2,3,4,5,6,7,8,9,10) VarIndex = zindex_label(var1) '获取变量指针、数组指针 ArrIndex = zindex_label(arr1) zindex_mark("VarIndex") = 0 '设置变量指针、数组指针标号 zindex_mark("ArrIndex") = 1 markarr(zindex_mark("VarIndex")) = VarIndex '通过获取的标号将指针存储至指针数组指定数组下标处 markarr(zindex_mark("ArrIndex")) = ArrIndex zindex_var(MarkArr(zindex_mark("VarIndex"))) = 777 '通过标号访问指针数组对应位置变量、数组 ?zindex_var(MarkArr(0)) ?var1 ?zindex_array(MarkArr(zindex_mark("ArrIndex")))(9) ?zindex_array(MarkArr(1))(9) end</pre>
相关指令	ZINDEX_ARRAY , ZINDEX_LABEL

6.2.7.ZINDEX_STRUCT -- 获取/访问结构变量

类型	语法指令
描述	获取结构变量的指针后，通过指针来访问结构变量。
语法	<pre>index = ZINDEX_LABEL(structvarname) ZINDEX_STRUCT(structname,index).item = var var = ZINDEX_STRUCT(structname,index).item</pre>
适用控制器	支持 ZV 功能或者 5 系列以上的控制器
例子	/
相关指令	ZINDEX_ARRAY , ZINDEX_LABEL

6.2.8.ZINDEX_ZVOBJ -- 获取对象索引数据

类型	语法指令
描述	获取对象索引的数据。
语法	ZINDEX_ZVOBJ(index)
适用控制器	支持 ZV 功能或者 5 系列以上的控制器
例子	<pre>ZVOBJECT img ZV_READIMAGE(img,"logo.png",0) '读取一张图像</pre>

	index = ZINDEX_LABEL(img) '获取图像对象的索引 ZV_LATCH(ZINDEX_ZV OBJ(index), 0) '获取图像索引的数据，并将图像显示到锁存通道 0 上
相关指令	ZINDEX_ARRAY , ZINDEX_LABEL

6.3. 显示指令

6.3.1.LCD_FEATURE -- 读取显示器特征

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 RTHmi
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

6.3.2.LCD_LEDSTATE -- 控制 LED 灯状态

类型	系统屏参数
描述	设置示教盒上 LED 灯的状态。 必须控制器和显示器同时支持此功能。
语法	LCD_LEDSTATE(lcdnum) = VAR lcdnum: 显示器编号 按 Bit 位依次设置，缺省=1，亮运行灯。
适用控制器	支持 RTHmi，固件版本 230801 以上
适用示教盒	ZHD500X
例子	LCD_LEDSTATE(0) = 3 '亮运行灯和报警灯

6.3.3.LCD_WDOGTIME -- 显示器掉线处理时间

类型	系统屏参数
----	-------

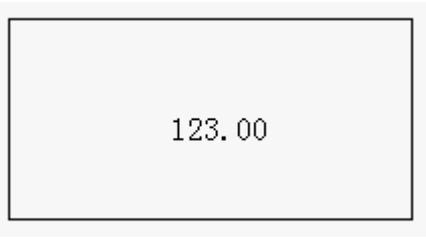
描述	设置屏掉线处理时间，ms 单位，当屏操作这个时间没有通讯时，急停开关(物理按键编号=5)自动按下，等于 0 时此功能不启用。 需要控制器 5 系列 20180404 以上固件版本支持，4 系列标准固件 20170721 以上版本支持。
语法	LCD_WDOGTIME(lcdnum) = time lcdnum: 0-显示器编号 time: ms 时间
适用控制器	支持 RTHmi
例子	LCD_WDOGTIME(0) = 100

6.3.4.DRAWNUM -- 自定义元件内显示数值

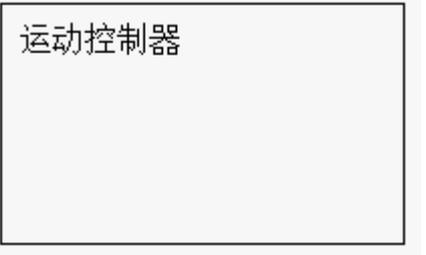
类型	显示指令
描述	显示一个数值。 此指令只能在自定义元件的绘图函数内使用，请查看 自定义元件调用函数 参考例程。
语法	DRAWNUM (x,y,value,n,dot) x, y: 显示区域左上角的坐标位置 value: 缺省显示值 n: 总长度位数，包括小数点和符号位。当 N 设置负数时，表示右对齐 dot: 小数点后取几位
适用控制器	支持 RTHmi
例子	DRAWRECT(0,0,200,100) '自定义元件内绘制边框 DRAWNUM(10,10,0,4,2) '在自定义元件显示区域左上角（10,10）显示 0.00 
相关指令	DRAWNUM2

6.3.5.DRAWNUM2 -- 指定位置显示数值

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

	<p>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</p> <p>value: 缺省显示值</p> <p>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</p> <p>dot: 小数点后取几位</p> <p>drawrect: 可选, 0-不画边框 (缺省), 1-画边框</p>
适用控制器	支持 RTHmi
例子	<p>DRAWRECT(0,0,200,100) '自定义元件内绘制边框'</p> <p>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'</p> 
相关指令	DRAWNUM

6.3.6.DRAWTEXT -- 绘制显示字符串

类型	显示指令
描述	<p>显示一个字符串, 支持中文, STRING 可以为字符串表达式。</p> <p>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。</p>
语法	<p>DRAWTEXT (x,y,STRING)</p> <p>X, Y: 显示区域左上角的坐标位置</p> <p>STRING: 显示的字符串</p>
适用控制器	支持 RTHmi
例子	<p>DRAWRECT(0,0,200,120) '自定义元件内绘制边框'</p> <p>DRAWTEXT(10,10, "运动控制器") '在自定义元件上显示文本“运动控制器”'</p> 
相关指令	DRAWTEXT2

6.3.7.DRAWTEXT2 -- 绘制显示字符串

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。请查看 自定义元件调用函数 参考例程。
语法	DRAWTEXT (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 RTHmi
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

6.3.8.DRAWLIBTEXT -- 显示文本库字符串

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 RTHmi
例子	DRAWLIBTEXT (10,10, 0, "text1")
相关指令	DRAWLIBTEXT2 , DRAWTEXT , DRAWTEXT2

6.3.9.DRAWLIBTEXT2 -- 显示文本库字符串

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	<p>DRAWLIBTEXT2(x1,y1, x2,y2, align, state, "textname"[,drawrect])</p> <p>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 state: 文本库的状态 textname: 文本库的名称 drawrect: 1-画边框, 0-不画边框.</p>
适用控制器	支持 RTHmi
例子	DRAWLIBTEXT2(10,10,100,50,0,0,"label")
相关指令	DRAWLIBTEXT , DRAWTEXT , DRAWTEXT2

6.3.10. DRAWREVERSE -- 绘制方块

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	<p>DRAWREVERSE(x1,y1,x2,y2)</p> <p>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</p>
适用控制器	支持 RTHmi
例子	<p>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</p> 

6.3.11. DRAWRECT -- 绘制矩形

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 RTHmi
例子	DRAWRECT(0,0,200,100) '自定义元件内绘制边框' 
相关指令	DRAWEX_RECT , DRAWEX_ELLIPSE

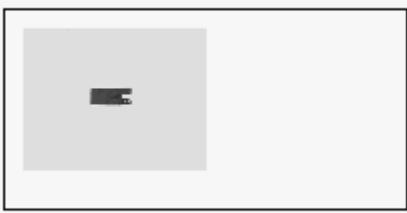
6.3.12. DRAWLINE -- 绘制线段

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 RTHmi
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	DRAWEX_LINE , DRAWARC

6.3.13. DRAWCLEAR -- 清除指定区域内容

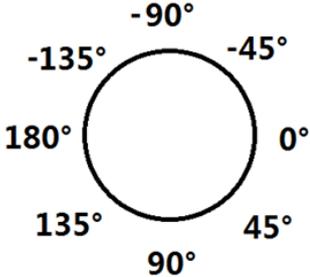
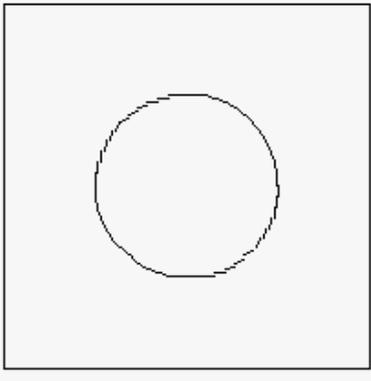
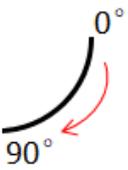
类型	显示指令
描述	清除指定读取区域。 此指令只能在自定义元件的绘图函数内使用，请查看 自定义元件调用函数 参考例程。
语法	DRAWCLEAR ([x1,y1,x2,y2]) x1,y1: 清除区域左上角的坐标位置 x2,y2: 清除区域右下角的坐标位置 无参数时全部清除
适用控制器	支持 RTHmi
例子	DRAWCLEAR (10,10,50,50)

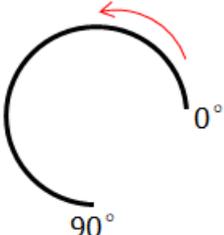
6.3.14. DRAWPIC -- 插入图片文件

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 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 RTHmi
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

6.3.15. DRAWARC -- 绘制圆弧

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。

<p>语法</p>	<p>DRAWARC(<i>centx</i>, <i>centy</i>, <i>radius</i>, <i>startangle</i>, <i>endangle</i>)</p> <p><i>centx</i>, <i>centy</i>: 圆心的位置 <i>radius</i>: 半径 <i>startangle</i>: 起始角度, 弧度单位 (公式: 弧度=角度*PI/180) <i>endangle</i>: 结束角度, 弧度单位</p> <p>绘制的角度说明:</p>  <p>从起始角度向结束角度绘制圆弧: 若起始角度<结束角度: 顺时针绘制圆弧 若起始角度>结束角度: 逆时针绘制圆弧</p>
<p>适用控制器</p>	<p>支持 RTHmi</p>
<p>例子</p>	<p>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</p>  <p>例二: 画圆弧</p>  <p>DRAWARC(<i>centerx</i>,<i>centery</i>,<i>radius</i>,$0*PI/180$,$90*PI/180$)</p> <p>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 $2*pi$。</p>

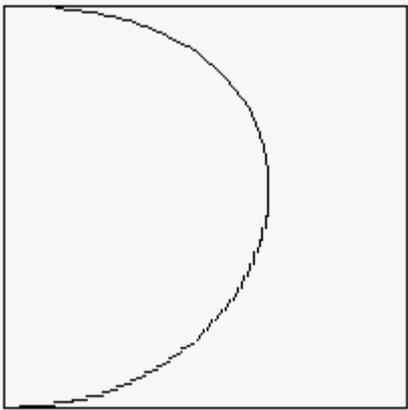
	 <p>$\text{DRAWARC}(\text{centerx}, \text{centery}, \text{radius}, 0 * \text{PI} / 180 + 2 * \text{PI}, 90 * \text{PI} / 180)$</p>
相关指令	DRAWLINE , DRAWEX_ARC

6.3.16. DRAWLIBPIC -- 插入图片库图片

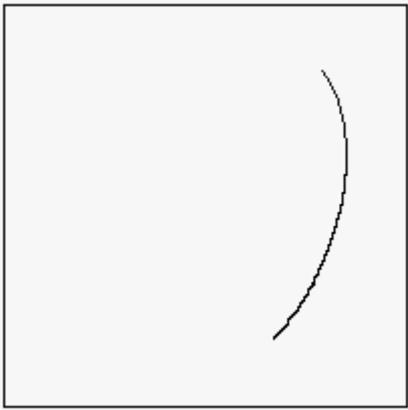
类型	显示指令	
描述	<p>绘制图片，图片文件要加入图片库，注意图片比较占空间，不需要的图片不要加入。</p> <p>此指令只能在自定义元件的绘图函数内使用，请查看自定义元件调用函数参考例程。</p>	
语法	<p>$\text{DRAWLIBPIC}(\text{piclibnamePath}, \text{state}, \text{x1}, \text{y1}[\text{x2}, \text{y2}])$</p> <p>piclibnamePath: 要加载的图片库路径</p> <ol style="list-style-type: none"> 1、系统图片库下：System\二级目录名称\加载图片的名称 2、用户图片库：User\二级目录名称\图片库名称 <p>state: 图片的状态编号</p> <p>x1,y1: 显示区域左上角的坐标位置</p> <p>x2,y2: 显示区域右下角的坐标位置</p>	
适用控制器	支持 RTHmi	
例子	<p>$\text{DRAWLIBPIC}(\text{"User\default\LOGO"}, 0, 100, 100, 300, 300)$ ‘在 (100,100) 到 (300,300) 区域显示图片库中名称为 LOGO 的 0 状态</p>	
相关指令	DRAWPIC	

6.3.17. DRAWBEZIER -- 绘制贝塞尔曲线

类型	显示指令	
描述	<p>画贝塞尔曲线。</p> <p>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。</p>	
语法	<p>$\text{DRAWBEZIER}(\text{x1}, \text{y1}, \text{x2}, \text{y2}, \text{x3}, \text{y3}, \text{x4}, \text{y4})$</p> <p>x1,y1: 第 1 个控制点</p> <p>x2,y2: 第 2 个控制点</p> <p>x3,y3: 第 3 个控制点</p> <p>x4,y4: 第 4 个控制点</p>	
适用控制器	支持 RTHmi	

例子	<p>DRAWRECT(0,0,200,200) '自定义元件内绘制边框'</p> <p>DRAWBEZIER(0,0,200,0,150,200,0,200) '画贝塞尔曲线'</p> 
相关指令	<p>DRAWEX_BEZIER, DRAWBSPLINE, DRAWEX_BSPLINE</p>

6.3.18. DRAWBSPLINE -- 绘制 B 样条曲线

类型	显示指令
描述	<p>画 B 样条曲线。</p> <p>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。</p>
语法	<p>DRAWBSPLINE (x1,y1,x2,y2,x3,y3,x4,y4)</p> <p>x1,y1: 第 1 个控制点</p> <p>x2,y2: 第 2 个控制点</p> <p>x3,y3: 第 3 个控制点</p> <p>x4,y4: 第 4 个控制点</p>
适用控制器	支持 RTHmi
例子	<p>DRAWRECT(0,0,200,200) '自定义元件内绘制边框'</p> <p>DRAWBSPLINE (0,0,200,0,150,200,0,200) '画 B 样条曲线'</p> 

6.3.19. DRAWDTLIST -- 绘制图形

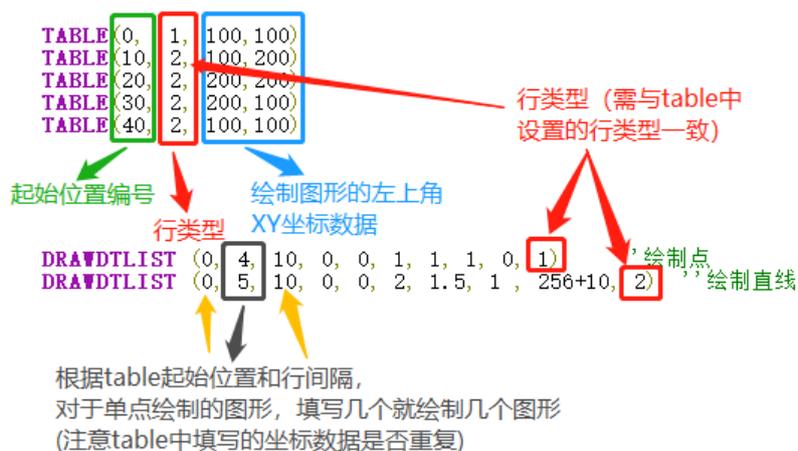
类型	显示指令																																			
描述	绘图指令，用于加快 TABLE 数据的绘制。 此指令只能在自定义元件的绘图函数内使用，请查看 自定义元件调用函数 参考例程。																																			
语法	<p>DRAWDTLIST (dtstart, imax, ispace, fxstart, fystart, fxlevel, fylevel, imode , [drawtype], [TYPE1, TYPE2, TYPE3, TYPE4])</p> <p>dtstart: table 起始位置，指向第一行的行类型 行类型：由用户自行定义，即可设置 0 为点或直线，1 也可为点或直线。</p> <p>imax: 行数（设置的 table 的组数） 行数不一定和设置的 table 组数相等，可根据实际情况填写。</p> <p>ispace: 行间隔（即每个 table 起始位置之差，若 imode 参数为 1 时，则行间隔一般不能<2；若 imode 偏移了 N 个数据，则最小行间隔需相应增加 N）</p> <p>fxstart: 左上角 X 坐标的偏移量，相对于 table 中设置的 X 坐标进行偏移(>0—向左偏移，<0—向右偏移，=0 不偏移)</p> <p>fystart: 左上角 Y 坐标的偏移量，相对于 table 中设置的 Y 坐标进行偏移(>0—向上偏移，<0—向下偏移，=0 不偏移)</p> <p>fxlevel : table 中 X 坐标的数值比例</p> <p>fylevel: table 中 Y 坐标的数值比例</p> <p>imode: 存储格式</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 20%;">值</th> <th>描述</th> </tr> </thead> <tbody> <tr> <td>10</td> <td>DSB 读取存储的格式，此时 X 的位置与行类型偏移 3 个数据</td> </tr> <tr> <td>1-9</td> <td>此时代表 X 在 table 中的位置与行类型偏移了 N 个数据，XY 的数据紧挨着</td> </tr> <tr> <td>其它值</td> <td>无效</td> </tr> </tbody> </table> <p>drawtype: 显示方式</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 20%;">值</th> <th colspan="2">描述</th> </tr> </thead> <tbody> <tr> <td>0 (缺省)</td> <td colspan="2">直接绘制一个点</td> </tr> <tr> <td rowspan="7">N*256+ 子 显示模式 N= 半径或 线宽 (N 的限 制范围： 1 ≤ N ≤ 16)</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 colspan="2">前后两点之间画直线，必须都是同一种类型</td> </tr> <tr> <td>N*256+ 11</td> <td colspan="2">设置圆弧起始点、中间点和结束点，自动计算圆弧并显示(该方式下不能通过 N 调节半径大小)</td> </tr> </tbody> </table>	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 个数据	1-9	此时代表 X 在 table 中的位置与行类型偏移了 N 个数据，XY 的数据紧挨着	其它值	无效	值	描述		0 (缺省)	直接绘制一个点		N*256+ 子 显示模式 N= 半径或 线宽 (N 的限 制范围： 1 ≤ N ≤ 16)	N*256+ 1	单点实心圆弧	N*256+ 2	单点空心圆弧	N*256+ 3	单点实心正方形	N*256+ 4	单点虚正方形	N*256+ 5	单点十字架	N*256+ 6	单点叉	N*256+ 9	单点显示，和上一点之间画一个虚线，N 只能为 1	N*256+ 10	前后两点之间画直线，必须都是同一种类型		N*256+ 11	设置圆弧起始点、中间点和结束点，自动计算圆弧并显示(该方式下不能通过 N 调节半径大小)	
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 个数据																																			
1-9	此时代表 X 在 table 中的位置与行类型偏移了 N 个数据，XY 的数据紧挨着																																			
其它值	无效																																			
值	描述																																			
0 (缺省)	直接绘制一个点																																			
N*256+ 子 显示模式 N= 半径或 线宽 (N 的限 制范围： 1 ≤ N ≤ 16)	N*256+ 1	单点实心圆弧																																		
	N*256+ 2	单点空心圆弧																																		
	N*256+ 3	单点实心正方形																																		
	N*256+ 4	单点虚正方形																																		
	N*256+ 5	单点十字架																																		
	N*256+ 6	单点叉																																		
	N*256+ 9	单点显示，和上一点之间画一个虚线，N 只能为 1																																		
N*256+ 10	前后两点之间画直线，必须都是同一种类型																																			
N*256+ 11	设置圆弧起始点、中间点和结束点，自动计算圆弧并显示(该方式下不能通过 N 调节半径大小)																																			

	N*256+ 12	设置圆弧结束点、中间点和起始点，自动计算圆弧并显示(该方式下不能通过 N 调节半径大小)
	N*256+ 13	整圆，和前面 2 个点计算圆并显示(该方式下不能通过 N 调节半径大小)
	N*256+ 19	依次为孤立点，直线终点，圆弧结束点，整圆终点，示教程快速显示
	N*256+ 90	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

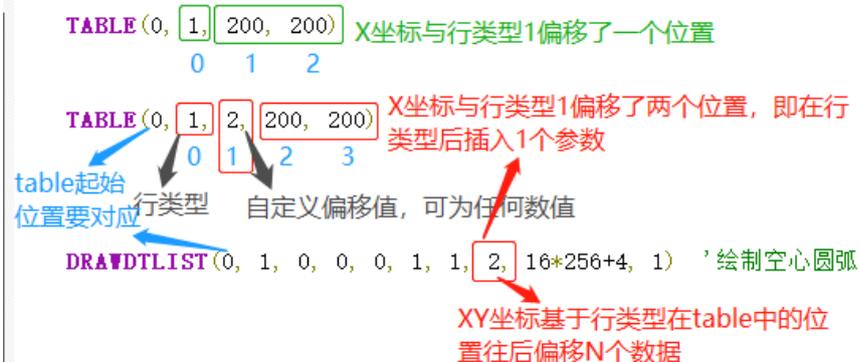
TYPE1~TYPE4: 需要绘制的行类型，一次最多可填 4 种行类型做同样的绘制(即该参数在一条 DRAWDTLIST 指令最多填 4 个行类型)，**必须和 table 中设置的行类型对应才能绘制图形。**

适用控制器 支持 RTHmi (建议用最新固件)

例子 行类型数据存储在 table 中第一个位置，在该例子中，1 表示单点，2 表示直线，X 坐标存储在第二个位置，Y 数据存储在第三个位置，XY 坐标不填默认为 0，此处的 XY 坐标是相对于自定义元件而定，自定义元件坐标以左上角为原点(0,0)。



imode(1-9)存储格式参数的使用方法:

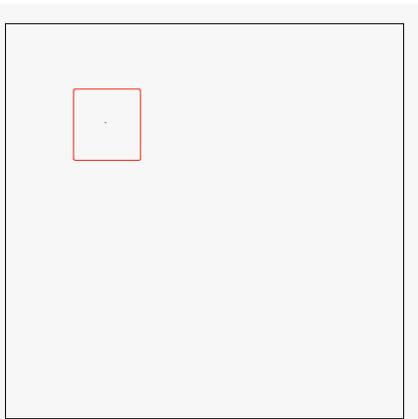


例一: 绘制一个点

TABLE(0, 1, 100,100)

DRAWDTLIST (0, 1, 10, 0, 0, 1, 1, 1, 0, 1) '绘制点 (点的位置在矩形框中，图中

矩形并非指令绘制，仅作提示)

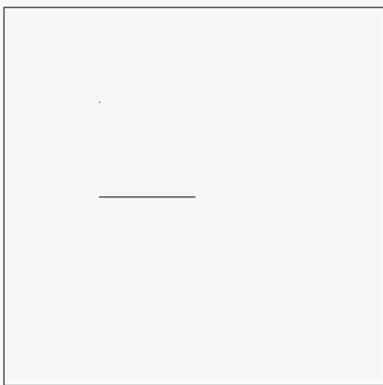


例二：绘制一段直线

TABLE(10,2,100,200)

TABLE(20,2,200,200)

DRAWDTLIST(10,2,10,0,0,1,1,1,256+10,2) ‘绘制直线（实线）’



例三：

TABLE(0,2,100,100)

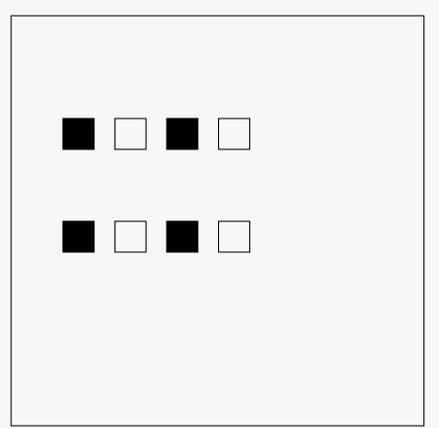
TABLE(10,2,100,200)

TABLE(20,2,200,200)

TABLE(30,2,200,100)

DRAWDTLIST(0,4,10,0,0,1,1,1,16*256+4,2) ‘绘制空心矩形’

DRAWDTLIST(0,4,10,40,0,1,1,1,16*256+3,2) ‘绘制实心矩形’



例四：

TABLE(0,1,100,100)

TABLE(3,1,100,150)

TABLE(6,1,200,150)

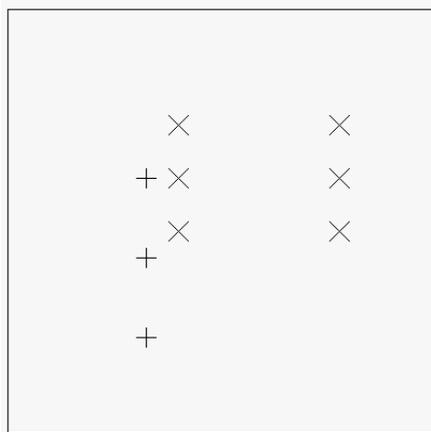
TABLE(9,1,200,100)

TABLE(12,1,100,200)

TABLE(15,1,200,200)

DRAWDTLIST(0,6,3,0,0,1.5,1,1,10*256+6,1) ‘绘制 6 个叉 (X 坐标扩大 1.5 倍)’

DRAWDTLIST(9,3,3,80,0,1,1.5,1,10*256+5,1) ‘绘制 3 个十字架 (Y 坐标扩大 1.5 倍, 并且 X 坐标往左偏移 80)’



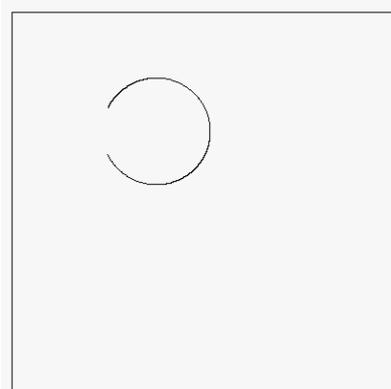
例五：

TABLE(0,1,100,100) ‘圆弧的起点’

TABLE(10,1,200,150) ‘圆弧的中间点’

TABLE(20,1,100,150) ‘圆弧的结束点’

DRAWDTLIST(0,3,10,0,0,1,1,1,10*256+11,1) ‘绘制圆弧 (三点画弧的方式)’



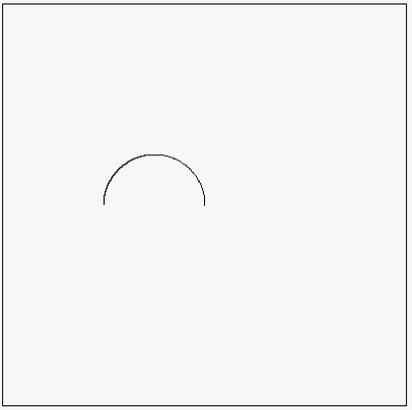
例六：画圆弧

TABLE(0,1,200,200)

TABLE(10,1,150,150)

TABLE(20,1,100,200)

DRAWDTLIST(0,4,10,0,0,1,1,1,16*256+12,1) ‘画半圆弧’

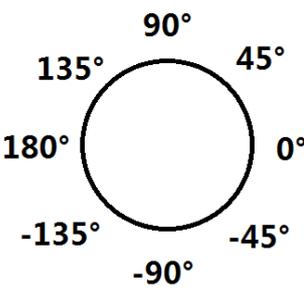
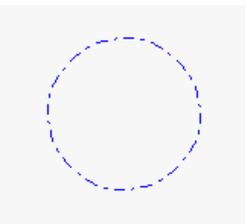
	
相关指令	DRAWLINE

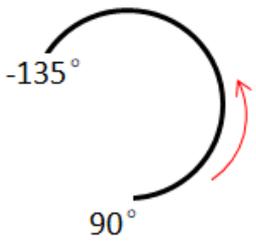
6.3.20. DRAWEX_LINE -- 绘制线段（带样式）

类型	绘图指令
描述	<p>画一条直线。与 DRAWLINE 的区别在于 DRAWEX_LINE 可以设置线条宽度和线条类型，而 DRAWLINE 不支持。</p> <p>DRAWEX 绘图指令也将对绘制区域进行限制，原则上是不会允许画出自定义控件之外的，DRAW 相关指令则无限制。</p> <p>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。</p>
语法	<p>DRAWEX_LINE(x1,y1,x2,y2)</p> <p style="padding-left: 20px;">x1,y1: 直线起始点的坐标位置</p> <p style="padding-left: 20px;">x2,y2: 直线结束点的坐标位置</p>
适用控制器	支持 RTHmi（仅支持 4 系列及以上控制器）
例子	<p>SETEX_LINE(3,0) ‘设置线条宽度</p> <p>SET_COLOR(RGB(0,0,255)) ‘设置线条颜色</p> <p>DRAWEX_LINE(100,100,200,50) ‘绘制直线</p> <div style="text-align: center;">  </div>
相关指令	DRAWLINE , SETEX_LINE , SET_COLOR

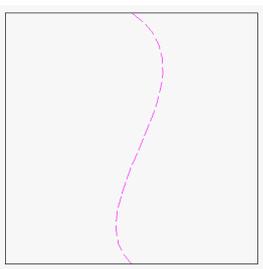
6.3.21. DRAWEX_ARC -- 绘制圆弧（带样式）

类型	绘图指令
描述	<p>画圆弧/椭圆弧。与 DRAWARC 的区别在于 DRAWEX_ARC 可以设置线条宽度和线条类型，而 DRAWARC 不支持。</p> <p>DRAWEX 绘图指令也将对绘制区域进行限制，原则上是不会允许画出自定义控件</p>

	<p>之外的，DRAW 相关指令则无限制。 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。</p>
<p>语法</p>	<p>DRAWARC(centx, centy, radiusx, radiusy, startangle, endangle)</p> <p>centx, centy: 圆心的位置 radiusx, radiusy: X 和 Y 方向的半径 startangle: 起始角度，弧度单位（公式：弧度=角度*PI/180） endangle: 结束角度，弧度单位</p> <p>绘制的角度说明：</p>  <p>从起始角度向结束角度绘制圆弧： 若起始角度<结束角度：顺时针绘制圆弧 若起始角度>结束角度：逆时针绘制圆弧</p>
<p>适用控制器</p>	<p>支持 RTHmi（仅支持 4 系列及以上控制器）</p>
<p>例子</p>	<p>例一：画整圆 SETEX_LINE(1,3) ‘设置线条样式 DRAWEX_ARC(200,200,50,50,0,2*PI) ‘绘制一个整圆</p>  <p>例二： SETEX_LINE(3,0) ‘设置线条宽度 DRAWEX_ARC(200,200,50,50,0*PI/180,PI) ‘顺时针画半圆弧</p>  <p>例三：</p>

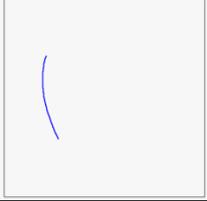
	<p>SETEX_LINE(2,0) ‘设置线条宽度 DRAWEX_ARC(200,200,50,50,90*PI/180,-135*PI/180) ‘逆时针画圆弧</p> 
相关指令	DRAWARC , DRAWEX_LINE , SETEX_LINE , SET_COLOR

6.3.22. DRAWEX_BEZIER -- 绘制贝塞尔曲线（带样式）

类型	绘图指令
描述	<p>画贝塞尔曲线。与 DRAWBEZIER 的区别在于 DRAWEX_BEZIER 可以设置线条宽度和线条类型，而 DRAWBEZIER 不支持。 DRAWEX 绘图指令也将对绘制区域进行限制，原则上是不会允许画出自定义控件之外的，DRAW 相关指令则无限制。 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。</p>
语法	<p>DRAWEX_BEZIER(x1, y1, x2, y2, x3, y3, x4, y4)</p> <p>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</p>
适用控制器	支持 RTHmi（仅支持 4 系列及以上控制器）
例子	<p>SETEX_LINE(1,1) ‘设置线条类型为虚线 DRAWEX_BEZIER(200,0,350,100,100,300,200,400) ‘绘制贝塞尔曲线</p> 
相关指令	DRAWBEZIER , DRAWBSPLINE , SETEX_LINE , SET_COLOR

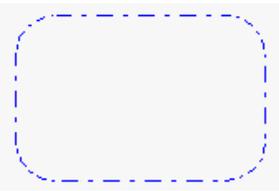
6.3.23. DRAWEX_BSPLINE -- 绘制 B 样条曲线（带样式）

类型	绘图指令
----	------

描述	<p>画 B 样条曲线。与 DRAWBSPLINE 的区别在于 DRAWEX_BSPLINE 可以设置线条宽度和线条类型，而 DRAWBSPLINE 不支持。</p> <p>DRAWEX 绘图指令也将对绘制区域进行限制，原则上是不会允许画出自定义控件之外的，DRAW 相关指令则无限制。</p> <p>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。</p>
语法	<p>DRAWEX_BSPLINE(x1, y1, x2, y2, x3, y3, x4, y4)</p> <p>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</p>
适用控制器	支持 RTHmi（仅支持 4 系列及以上控制器）
例子	<p>SETEX_LINE(2,0) ‘设置线条宽度 SET_COLOR(RGB(0,0,255)) ‘设置线条颜色 DRAWEX_BSPLINE(200,0,50,100,100,300,200,400) ‘绘制 B 样条曲线</p> 
相关指令	DRAWBSPLINE , DRAWBEZIER , DRAWEX_BEZIER , SETEX_LINE , SET_COLOR

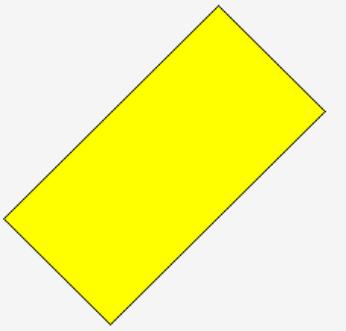
6.3.24. DRAWEX_RECT -- 绘制圆角矩形（带样式）

类型	绘图指令
描述	<p>在指定区域画一个圆角矩形。与 DRAWRECT 的区别在于 DRAWEX_RECT 支持设置线条宽度、线条类型和填充绘图，而 DRAWRECT 不支持。</p> <p>DRAWEX 绘图指令也将对绘制区域进行限制，原则上是不会允许画出自定义控件之外的，DRAW 相关指令则无限制。</p> <p>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。</p>
语法	<p>DRAWEX_RECT(x1,y1,x2,y2,round [, iffill])</p> <p>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 round: 圆角半径 iffill: 是否填充，缺省为 0（不填充） 0 - 不填充，非 0 - 填充</p> <p>（注意：后绘制的可能会遮挡前绘制的图，因此选择填充时建议先画填充，后画边框）</p>
适用控制器	支持 RTHmi（仅支持 4 系列及以上控制器）
例子	例一：

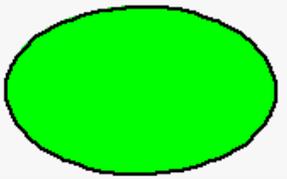
	<p>SETEX_LINE(1,3) ‘设置线条类型 SET_COLOR(0,0,255) ‘设置线条颜色 DRAWEX_RECT(50,50,200,150,25) ‘画半径为 25 的圆角矩形边框</p>  <p>例二： SET_COLOR(255,255,255),RGB(0,0,255) ‘设置填充颜色 DRAWEX_RECT(50,50,200,150,20,1) ‘绘制可填充的圆角矩形</p> 
相关指令	DRAWRECT , DRAWEX_ELLIPSE , DRAWEX_POLYGON , SETEX_LINE , SET_COLOR , DRAWEX_POLYGON2

6.3.25. DRAWEX_ROTRECT -- 绘制旋转矩形（带样式）

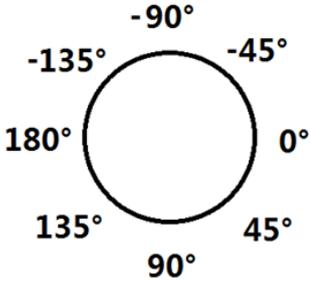
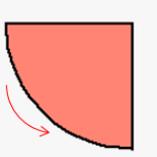
类型	绘图指令
描述	<p>在指定区域画一个旋转矩形。 此指令支持设置线条宽度、线条类型和填充绘图，但不允许画出自定义控件之外。 此指令只能在自定义元件的绘图函数内使用。请查看自定义元件调用函数参考例程。</p>
语法	<p>DRAWEX_ROTRECT(cx,cy,w,h,angle[, iffill])</p> <p>cx, cy: 旋转矩形中心坐标位置 w, h: 旋转矩形宽高 angle: 矩形旋转角度，弧度单位，逆时针方向 iffill: 是否填充，缺省为 0（不填充） 0 - 不填充，非 0 - 填充</p> <p>（注意：后绘制的可能会遮挡前绘制的图，因此选择填充时建议先画填充，后画边框）</p>
适用控制器	支持 RTHmi（仅支持 4 系列及以上控制器）
例子	<p>例：SET_COLOR(0,0,0),RGB(0,255,0) DRAWEX_ROTRECT(150,150,200,100,45*PI/180, 1) ‘绘制逆时针旋转 45 弧度的矩形 DRAWEX_ROTRECT(150,150,200,100, 45*PI/180, 0) ‘绘制逆时针旋转 45 弧度的矩形边框</p>

	
相关指令	DRAWEX_RECT , SETEX_LINE , SET_COLOR

6.3.26. DRAWEX_ELLIPSE -- 绘制椭圆（带样式）

类型	绘图指令
描述	在指定区域画一个椭圆。 此指令支持设置线条宽度、线条类型和填充绘图，但不允许画出自定义控件之外。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	<p><code>DRAWEX_ELLIPSE(centx, centy, radiusx, radiusy [, iffill])</code></p> <p>centx, centy: 圆心的位置 radiusx: X 方向的半径 radiusy: Y 方向的半径 ifill: 是否填充，缺省为 0（不填充） 0 - 不填充，非 0 - 填充</p> <p>（注意：后绘制的可能会遮挡前绘制的图，因此选择填充时建议先画填充，后画边框）</p>
适用控制器	支持 RTHmi（仅支持 4 系列及以上控制器）
例子	<p><code>SETEX_LINE(2,0)</code> ‘设置线条宽度为 2 的实线’</p> <p><code>SET_COLOR(0,0,0,0,0,255,0)</code> ‘设置椭圆边框颜色为黑色，填充颜色为绿色’</p> <p><code>DRAWEX_ELLIPSE(100,150,80,50,1)</code> ‘绘制可填充的椭圆’</p> <p><code>DRAWEX_ELLIPSE(100,150,80,50,0)</code> ‘绘制椭圆边框’</p> <div style="text-align: center;">  </div>
相关指令	DRAWEX_RECT , DRAWEX_SECTOR , SETEX_LINE , SET_COLOR

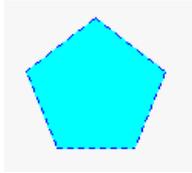
6.3.27. DRAWEX_SECTOR -- 绘制扇形（带样式）

类型	绘图指令
描述	<p>在指定区域画一个扇形。</p> <p>此指令支持设置线条宽度、线条类型和填充绘图，但不允许画出自定义控件之外。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。</p>
语法	<p>DRAWEX_SECTOR(centx, centy, radiusx, radiusy, startangle, endangle [, iffill])</p> <p>centx, centy: 圆心的位置 radiusx, radiusy: 半径大小 startangle: 起始角度，弧度单位 endangle: 结束角度 iffill: 是否填充，缺省为0（不填充） 0- 不填充，非0- 填充</p> <p>（注意：后绘制的内容可能会遮挡前绘制的内容，因此选择填充时建议先画填充，后画边框）</p> <p>绘制的角度说明：</p> <div style="text-align: center;">  </div> <p>从起始角度向结束角度绘制扇形： 若起始角度<结束角度：顺时针绘制扇形 若起始角度>结束角度：逆时针绘制扇形</p>
适用控制器	支持 RTHmi（仅支持 4 系列及以上控制器）
例子	<p>例一：</p> <pre>SETEX_LINE(2,0) ‘设置线条宽度 SET_COLOR(RGB(0,0,0),RGB(250,128,114)) ‘设置边框颜色和填充颜色 DRAWEX_SECTOR(200,150,100,100,PI,90*PI/180,1) ‘绘制扇形边框 DRAWEX_SECTOR(200,150,100,100,PI,90*PI/180,0) ‘逆时针绘制可填充扇形</pre> <div style="text-align: center;">  </div> <p>例二：画优质扇形，起始角度+PI*2 即可</p> <pre>SETEX_LINE(2,0) ‘设置线条宽度</pre>

	<p>SET_COLOR(0,0,0,RGB(250,128,114)) ‘设置边框颜色和填充颜色</p> <p>DRAWEX_SECTOR(200,150,50,50,PI/2+2*PI,4*PI,1) ‘绘制扇形边框</p> <p>DRAWEX_SECTOR(200,150,50,50,PI/2+2*PI,4*PI,0) ‘顺时针绘制可填充扇形</p> 
相关指令	DRAWEX_ELLIPSE , SETEX_LINE , SET_COLOR

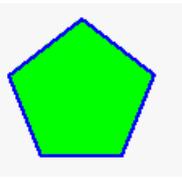
6.3.28. DRAWEX_POLYGON -- 绘制多边形（带样式）

类型	绘图指令
描述	<p>在指定区域画一个多边形。</p> <p>此指令支持设置线条宽度、线条类型和填充绘图，但不允许画出自定义控件之外。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。</p>
语法	<p>DRAWEX_POLYGON(points [, x1,y1,...,xn,yn] [, iffill])</p> <p>points: 多边形点个数，决定后面有多少组 xy 数据（封闭图形起点和终点坐标需传两次，则总需传入点的坐标个数=多边形点个数+1）</p> <p>xn,yn: 第 n 个点的 x 和 y 的坐标值</p> <p>iffill: 是否填充，缺省为 0（不填充）</p> <p>0 - 不填充，非 0 - 填充</p> <p>（注意：后绘制的可能会遮挡前绘制的图，因此选择填充时建议先画填充，后画边框）</p> <p>特别说明：</p> <p>若绘制封闭的多边形，如绘四边形需要传 5 个点，且起点=终点，反之则是普通的多线段（不能进行填充）。</p> <p>若绘制的填充多边形不显示，请先检查是否是封闭图形。</p> <p>多边形的边数是有限制的，当前版本最多支持 32 边形。（不同控制器支持参数个数不同，多边形的边数由控制器支持参数个数决定）</p>
适用控制器	支持 RTHmi（仅支持 4 系列及以上控制器）
例子	<p>SETEX_LINE(1,2) ‘设置线条类型为虚线</p> <p>SET_COLOR(0,0,255,RGB(0,255,255)) ‘设置边框颜色和填充颜色</p> <p>DRAWEX_POLYGON(6,100,90,145,125,125,175,75,175,55,125,100,90,1) ‘绘制可填充的五边形</p> <p>DRAWEX_POLYGON(6,100,90,145,125,125,175,75,175,55,125,100,90) ‘绘制五边形边框</p>

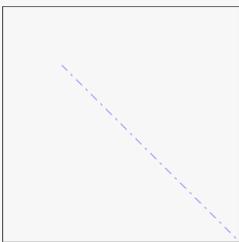
	
相关指令	DRAWRECT , DRAWEX_RECT , SETEX_LINE , SET_COLOR

6.3.29. DRAWEX_POLYGON2 -- 绘制多边形 (table 存储)

类型	绘图指令
描述	<p>在指定区域画一个多边形</p> <p>此指令支持设置线条宽度、线条类型和填充绘图，但不允许画出自定义控件之外。</p> <p>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。</p>
语法	<p>DRAWEX_POLYGON2(points, tableindex [, iffill])</p> <p>points: 多边形点个数，决定后面有多少组 xy 数据（封闭图形起点和终点坐标需传两次，则总需传入点的坐标个数=多边形点个数+1）</p> <p>tableindex: 存储起始点数据的 table 下标 (x1,y1,..xn,yn)</p> <p>iffill: 是否填充，缺省为 0（不填充）</p> <p style="padding-left: 20px;">0 - 不填充，非 0 - 填充</p> <p style="color: red;">（注意：后绘制的可能会遮挡前绘制的图，因此选择填充时建议先画填充，后画边框）</p> <p>特别说明：</p> <p style="padding-left: 20px;">若绘制封闭的多边形，如绘四边形需要传 5 个点，且起点=终点，反之则是普通的多线段（不能进行填充）。</p> <p style="color: red;">若绘制的填充多边形不显示，请先检查是否是封闭图形。</p> <p style="color: red;">多边形的边数是有限制的，当前版本最多支持 32 边形。</p>
适用控制器	支持 RTHmi（仅支持 4 系列及以上控制器）
例子	<pre> SETEX_LINE(2,0) ‘设置线条宽度为 2，实线类型 SET_COLOR(RGB(0,0,255),RGB(0,255,0)) ‘设置线条颜色和填充颜色 TABLE(0)=100 'x1 TABLE(1)=90 'y1 TABLE(2)=55 'x2 TABLE(3)=125 'y2 TABLE(4)=75 'x3 TABLE(5)=175 'y3 TABLE(6)=125 'x4 TABLE(7)=175 'y4 TABLE(8)=145 'x5 TABLE(9)=125 'y5 TABLE(10)=100 'x6 </pre>

	<p>TABLE(11)=90 'y6</p> <p>DRAWEX_POLYGON2(6,0,1) ‘绘制可填充的五边形</p> <p>DRAWEX_POLYGON2(6,0) ‘绘制五边形边框</p> 
相关指令	DRAWRECT , DRAWEX_RECT , SETEX_LINE , SET_COLOR

6.3.30. SETEX_LINE -- 设置线段属性

类型	显示指令
描述	<p>绘图线宽度设置（该指令与 DRAWEX 类指令配套使用）</p> <p>所有的 DRAWEX 指令均支持线段属性</p> <p>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。</p>
语法	<p>SETEX_LINE (width [, type])</p> <p>width: 线段宽度（范围：1-20）</p> <p>type: 线段类型（线宽>1 时只能实线）</p> <p>0-实线（缺省）</p> <p>1-长虚线 DASH: —————</p> <p>2-虚线 DOT: ·····</p> <p>3- DASHDOT: -·-·-·-·-·</p> <p>4- DASHDOTDOT: ·-·-·-·-·-·</p>
适用控制器	支持 RTHmi（仅支持 4 系列及以上控制器）
例子	<p>SETEX_LINE(1,3) ‘设置线条宽度为 1，线条类型为 DASHDOT</p> <p>SET_COLOR(RGB(0,0,255))</p> <p>DRAWEX_LINE(100,100,400,400)</p> 
相关指令	SET_COLOR

6.3.31. SET_FONT -- 字体设置

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

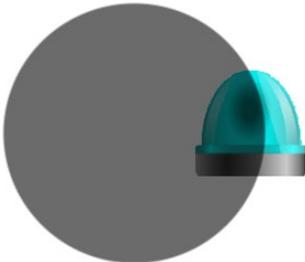
	不一致时，会出现缩放；如果需要自己的字体，请使用 zfontmaker 制作专门的字体文件。
语法	SET_FONT (width, height, [fontname]) width: 字体宽度，英文自动减半 height: 字体高度 fontname: 使用的字体文件名，不设置时不修改当前字体
适用控制器	支持 RTHmi
例子	SET_FONT(16,16, "16X16 字体文件小五.zft")
相关指令	SET_COLOR

6.3.32. SET_COLOR -- 设置颜色

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: 线段颜色 backcor: DRAWTEXT 时的背景色，不填的时候为透明(-1)
适用控制器	支持 RTHmi
例子	<p>例一： SET_COLOR(RGB(255,255,255)) '设置颜色为白色</p> <p>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</p> <div data-bbox="411 1444 837 1668" data-label="Image"> </div> <p>例三： SET_COLOR(RGB(0,150,255),RGB(255,255,0)) '蓝色线条，黄色填充背景 DRAWEX_RECT(50,50,200,150,0,1) '自定义元件内绘制填充矩形 DRAWEX_RECT(50,50,200,150,0) '自定义元件内绘制矩形边框</p>

	
相关指令	RGB

6.3.33. SETEX_ALPHA -- 设置绘图透明度

类型	显示指令
描述	绘图透明度设置。 从 RTHMI_V1.3.0 版本开始，该指令对所有 DRAWEX 绘图指令、滚动条指令和绘图片指令（DRAWPIC、DRAWLIBPIC）生效。
语法	SETEX_ALPHA(alpha) alpha: 透明度值, 0-不透明, 100-完成透明
适用控制器	支持 RTHmi
例子	SETEX_ALPHA(50) '设置透明度 50 DRAWEX_ELLIPSE(150, 150, 100, 100, 1) 

6.3.34. SET_REDRAW -- 重新绘图

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 RTHmi
例子	SET_REDRAW '重新绘制全部区域

6.3.35. RGB -- 颜色属性

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 RTHmi
例子	RGB(255,255,0) '纯黄色
相关指令	SET_COLOR

6.3.36. HMI_LANG -- 文本库语言切换

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 RTHmi
例子	Hmi_LANG=0 '选择文本库语言编号 0 的内容
相关指令	/

6.3.37. SCROLLBAR_FREE -- 释放滚动条

类型	显示指令
描述	释放滚动条。 删除释放已创建的滚动条，释放后的滚动条 ID 将在自定义控件中的刷新函数、绘图函数的调用失效。
语法	SCROLLBAR_FREE (id) id: 滚动条唯一编号
适用控制器	支持 RTHmi (仅支持 4 系列及以上控制器)
例子	/

6.3.38. SCROLLBAR_INIT -- 滚动条初始化

类型	显示指令
描述	指定 ID 初始化一个滚动条。具体使用可参见 滚动条使用 例程。 可用滚动条 ID(0~31)
语法	SCROLLBAR_INIT(id,x1,y1,width,height,maxlines,showlines[forecor,backcor,presscor])

	<p>id = SCORLLBAR_INIT (x1, y1, width, height, maxlines, showlines, [forecor,backcor,presscor])</p> <p>id: 滚动条唯一编号, 不传 id 参数会自动获取可用 ID 并返回作为返回值, 返回-1 表示无空闲的 ID, 自动初始化失败</p> <p>x1,y1: 滚动条左上角位置 (以自定义元件框的左上角为原点)</p> <p>width,height: 滚动条宽高</p> <p>maxlines: 最大显示长度</p> <p>showlines: 一页显示长度</p> <p>当 maxlines=20, showlines=10 时, 滚动条可滚动 (20-10=10) 个单位, 滚动值(SCROLLBAR_POS)范围为 0~10</p> <p>forecor: 滚动条颜色, 可缺省</p> <p>backcor: 滚动条背景颜色, 可缺省</p> <p>presscor: 滚动条按压颜色, 可缺省</p> <p>特别说明: 若 width>height, 则为水平滚动条, 反之为垂直滚动条。</p>
适用控制器	支持 RTHmi (仅支持 4 系列及以上控制器)
例子	<p>SCROLLBAR_INIT(0, 0, 0, 21, 330, 50, 10, RGB(255,0,0), RGB(0,255,0), RGB(0,0,255))</p> <p>滚动条显示效果:</p>  <p>按住滚动条时显示效果:</p> 
相关指令	SCROLLBAR_POS , SCROLLBAR_REFLASH , SCROLLBAR_DRAW

6.3.39. SCROLLBAR_POS -- 获取/设置滚动值

类型	刷新指令
描述	<p>设置/获取指定滚动条的当前滚动位置。</p> <p>处理滚动条刷新事件, 控制滚动条上下拖动、滚动</p> <p>此指令只能在自定义元件的刷新函数内使用。具体使用可参见滚动条使用例程。</p>
语法	<p>自动刷新滚动条事件</p> <p>value = SCROLLBAR_POS(id [,winid,controlid])</p> <p>id: 滚动条唯一编号</p> <p>value: 返回当前滚动值, 返回-1 表示无刷新</p> <p>winid: 窗口 ID</p> <p>controlid: 控件 ID</p> <p>设置滚动值</p> <p>SCROLLBAR_POS (id, value [, winid, controlid])</p> <p>id: 滚动条唯一编号</p> <p>value: 强制设定当前滚动值</p>

	<p>winid: 窗口 ID controlid: 控件 ID</p> <p>注：非自定义控件上使用时，id=0 表示操作垂直滚动条，id=1 表示水平滚动条。若控件只有一个滚动条，则 id 参数无效。</p>
适用控制器	支持 RTHmi（仅支持 4 系列及以上控制器）
例子	<p>刷新函数</p> <pre>value = SCROLLBAR_POS(0) ?"当前滚动值: "value</pre>
相关指令	SCROLLBAR_INIT , SCROLLBAR_REFLASH , SCROLLBAR_DRAW

6.3.40. SCROLLBAR_REFLASH -- 刷新滚动条

类型	刷新指令
描述	<p>刷新滚动条。</p> <p>处理滚动条刷新事件，控制滚动条上下拖动、滚动。</p> <p>此指令只能在自定义元件的刷新函数内使用。具体使用可参见滚动条使用例程。</p>
语法	<p>’自动刷新滚动条事件，自动触发重绘</p> <pre>SCROLLBAR_REFLASH (id) id: 滚动条唯一编号</pre> <p>’滚动条事件触发时，用户可根据返回值来判断是否重绘</p> <pre>bifupdate = SCROLLBAR_REFLASH (id) id: 滚动条唯一编号 bifupdate: 是否触发更新, 0-未触发 or 1-触发</pre>
适用控制器	支持 RTHmi（仅支持 4 系列及以上控制器）
例子	<p>例子 1:</p> <p>’刷新函数</p> <pre>SCROLLBAR_REFLASH (0) ’鼠标事件将自动触发 SET_REDRAW</pre> <p>’绘图函数</p> <pre>SCROLLBAR_DRAW (0)</pre>  <p>例子 2:</p> <p>’刷新函数（需通过自定义元件的刷新函数进行调用）</p> <pre>global sub sub_reflash() bifredraw = SCROLLBAR_REFLASH (0) ’根据返回值手动判断是否重绘 if bifredraw > 0 then ?"当前滚动值: " SCROLLBAR_POS (0) SET_REDRAW endif end sub</pre>

	<pre> '绘图函数（需通过自定义元件的绘图函数进行调用） global sub sub_redraw() SCROLLBAR_DRAW (0) end sub </pre> 
相关指令	SCROLLBAR_INIT , SCROLLBAR_DRAW , SCROLLBAR_POS

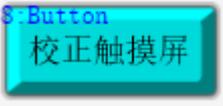
6.3.41. SCROLLBAR_DRAW -- 绘制滚动条

类型	绘图指令
描述	滚动条绘图指令。此指令只能在自定义元件的绘图函数内使用。具体使用可参见 滚动条使用 例程。 滚动条显示并不限制在自定义控件内，请用户使用前计算好空间位置，避免使用异常。
语法	SCROLLBAR_DRAW (id) id: 滚动条唯一编号
适用控制器	支持 RTHmi（仅支持 4 系列及以上控制器）
例子	SCROLLBAR_DRAW (0) 
相关指令	SCROLLBAR_INIT , SCROLLBAR_POS , SCROLLBAR_REFLASH

6.4. 触摸屏指令

6.4.1. TOUCH_ADJUST -- 触摸屏校正

类型	触摸屏指令
描述	<p>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。</p> <p>注意：该指令连接 ZHD400X 示教盒、500X 示教盒或支持该指令的手持盒设备触摸屏上才起效！</p> <p>【校正方式】</p> <p>方式 1：通过点击设置界面的 Touch Adjust 功能进入触摸校正界面（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。）</p> <p>方式 2：连接控制器后，通过控制器的 TOUCH_ADJUST 指令来触发校正。</p> <p>方式 3：不连接控制器，按下 12(F2)按键不松开，再按下 11(F1)按键。</p> <p>不连接控制器，按下 16 (F6)按键，不松开时继续按下 11 (F1)按键</p>
语法	TOUCH_ADJUST ()
适用控制器	支持 RTHmi
例子	用例 1:

	<p>(1) 使用一个 Hmi 按钮控件 </p> <p>(2) 在该控件的“属性”中调用子函数</p> <table border="1" data-bbox="446 369 981 761"> <tr> <td colspan="2">标签</td> </tr> <tr> <td>文本库</td> <td></td> </tr> <tr> <td>格式文本(0)</td> <td>校正触摸屏 ...</td> </tr> <tr> <td>格式文本(1)</td> <td>...</td> </tr> <tr> <td colspan="2">动作</td> </tr> <tr> <td>动作</td> <td>调用函数</td> </tr> <tr> <td>松开时动作</td> <td>False</td> </tr> <tr> <td>动作函数名</td> <td>correcting</td> </tr> </table> <p>(3) Basic 子函数 <pre>global sub correcting() TOUCH_ADJUST() end sub</pre> </p> <p>(4) 运行含有该指令的程序项目，点击该按钮</p> <p>(5) 直接进入触摸校正界面，后根据“+”的位置长按进行校正即可。</p> <p>用例 2：直接放在程序初始化函数里 TOUCH_ADJUST () ‘进行触摸校正</p>	标签		文本库		格式文本(0)	校正触摸屏 ...	格式文本(1)	...	动作		动作	调用函数	松开时动作	False	动作函数名	correcting
标签																	
文本库																	
格式文本(0)	校正触摸屏 ...																
格式文本(1)	...																
动作																	
动作	调用函数																
松开时动作	False																
动作函数名	correcting																
相关指令	/																

6.4.2.TOUCH_SCAN -- 触摸动作扫描

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。 预留触摸屏模拟鼠标右键、中键功能。
语法	<pre>action = TOUCH_SCAN (num [, optkey])</pre> <p>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)</p> <p>optkey: 操作鼠标按键，可选，默认为 0 鼠标左键</p> <ul style="list-style-type: none"> 0 – 鼠标左键 1 – 鼠标右键 2 – 鼠标中键 <p>action: 返回鼠标动作，1-按下，-1 松开，0 无变化</p>
适用控制器	支持 RTHmi
例子	<p>例一：</p> <pre>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</pre>

	<pre>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</pre>
相关指令	MOUSE_SCAN

6.4.3.TOUCH_STATE -- 获取触摸状态

类型	触摸屏指令
描述	<p>读取触摸状态，返回>0 表示按下，返回 0 表示松开，将触摸对应的位置的 X,Y 坐标保存到 table 表中。</p> <p>预留触摸屏模拟鼠标右键、中键功能。</p>
语法	<pre>state = TOUCH_STATE (num [, optkey])</pre> <p>num: 触摸对应的位置 X,Y 坐标分别存储在 table(num), table(num+1)</p> <p>optkey: 操作鼠标按键，可选，默认为 0 鼠标左键</p> <ul style="list-style-type: none"> 0 – 鼠标左键 1 – 鼠标右键 2 – 鼠标中键 <p>state: 返回鼠标状态，1-按下，0 松开</p>
适用控制器	支持 RTHmi
例子	<pre>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</pre>
相关指令	MOUSE_STATE

6.4.4.MOUSE_SCAN -- 鼠标动作扫描

类型	触摸屏指令
描述	扫描鼠标按下动作，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。 注：使用该指令需通过函数调用方式实现。
语法	<pre>action = MOUSE_SCAN (num [,optkey])</pre> <p>num: 触摸对应的位置的 X,Y 坐标分别存储在 table(num), table(num+1)</p> <p>optkey: 操作鼠标按键, 可选, 默认为 0 鼠标左键</p> <p>0 – 鼠标左键</p> <p>1 – 鼠标右键</p> <p>2 – 鼠标中键</p> <p>action: 返回鼠标动作, 1-按下, -1 松开, 0 无变化</p>
适用控制器	支持 RTHmi
例子	<pre>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</pre>
相关指令	TOUCH_SCAN

6.4.5.MOUSE_STATE -- 获取鼠标状态

类型	触摸屏指令
描述	读取鼠标状态, 返回>0 表示按下, 返回 0 表示松开, 将触摸对应的位置的 X,Y 坐标保存到 table 表中。
语法	<pre>state = MOUSE_STATE (num [, optkey])</pre> <p>num: 触摸对应的位置的 X,Y 坐标分别存储在 table(num), table(num+1)</p> <p>optkey: 操作鼠标按键, 可选, 默认为 0 鼠标左键</p> <p>0 – 鼠标左键</p> <p>1 – 鼠标右键</p> <p>2 – 鼠标中键</p> <p>state: 返回鼠标状态, 1-按下, 0 松开</p>
适用控制器	支持 RTHmi
例子	<pre>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</pre>
相关指令	TOUCH_STATE

6.5. 按键指令

6.5.1.MOUSE_WHEEL -- 获取鼠标滚轮值

类型	虚拟按键指令
描述	获取鼠标滚轮值，此指令只能在自定义元件的刷新函数内使用。
语法	wheel = MOUSE_WHEEL (tabid) tabid: 鼠标位置的 X,Y 坐标分别存储在 table(tabid), table(tabid +1) wheel: 返回鼠标滚轮值，-120 表示向下滚动一格，120 表示向上滚动一格
适用控制器	支持 RTHmi (5 系列及以上，4 系仅 ZMC432 最新固件支持)
例子	wheelval = MOUSE_WHEEL(0) if wheelval <> 0 then wheelval = wheelval/120 ' 表示滚动量，>0 向上滚动，<0 向下滚动 endif

6.5.2.KEY_STATE -- 获取物理按键状态

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 RTHmi
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

6.5.3.KEY_EVENT -- 获取物理按键状态变化

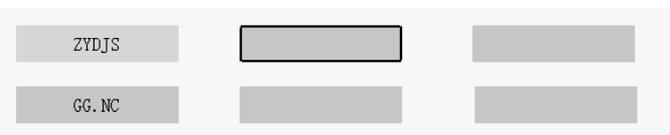
类型	按键指令
描述	读取物理按键状态变化，1-按下，-1-松开，0-不变。 此指令只能在自定义元件的刷新函数内使用，请查看 自定义元件调用函数 参考例程。
语法	KEY_EVENT (keynum) Keynum 按键编号
适用控制器	支持 RTHmi
例子	num =KEY_SCAN()

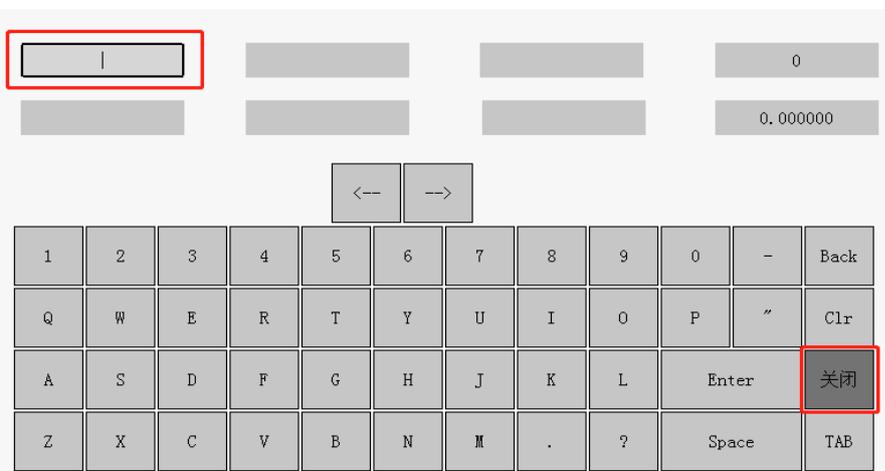
	<pre>IF KEY_EVENT (num)=1 THEN '扫描按下操作, 打印按键编号 ?"按下" num ENDIF</pre>
相关指令	KEY_SCAN , KEY_STATE

6.5.4.KEY_SCAN -- 获取物理按键编码

类型	按键指令
描述	<p>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。</p> <p>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。</p> <p>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。</p>
语法	value=KEY_SCAN()
适用控制器	支持 RTHmi
例子	<pre>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</pre>
相关指令	KEY_EVENT , KEY_STATE

6.5.5.VKEY_MODE -- 开启虚拟键输入模式

类型	虚拟按键指令
描述	<p>开启/关闭虚拟键输入模式, 开启虚拟键输入模式后, 值显示、字符显示控件将会获得焦点 (同一时刻有且只有一个控件获取焦点), 获得焦点的控件可以直接接收虚拟键值, 且获得高亮提示 (如下第一行第二个控件)</p> 
语法	<pre>VKEY_MODE (mode[, winid]) mode = VKEY_MODE() mode: 0 - 关闭虚拟键输入模式 1- 开启虚拟键输入模式 1, 可编辑控件有效; 点击切换焦点, 若当前处于焦点状态, 再点击弹出键盘 2- 开启虚拟键输入模式 2, 可编辑控件有效, 屏蔽弹出键盘 3- 开启虚拟键输入模式 3, 可编辑控件有效 通过 TAB 键切换焦点, 首次按下 ENTER 键后才能进入编辑状态 (未进入编辑状态不可输入, 进入编辑状态后显示输入光标), 再次</pre>

	<p>按 ENTER 则确认编辑并切换下一个控件</p> <p>4- 开启虚拟键输入模式 4，可编辑控件有效 跟模式 3 一样，不同的是模式 4 进入编辑状态会默认清空原来的数据</p> <p>winid: 指定窗口号，缺省为当前（顶层）窗口</p> <p>注：窗口号缺省时，会自动扫描最顶层窗口并切换当前焦点，只能在当前最顶层窗口上使用输入焦点功能，无法在其他窗口上使用。</p>
适用控制器	支持 RTHmi
例子	<p>VKEY_MODE(1) '开启虚拟键输入模式'</p> <p>VKEY_MODE(1, 10) '指定窗口 10 开启虚拟键输入模式'</p> 
相关指令	VKEY_SCAN , VKEY_STATE

6.5.6.VKEY_STATE -- 设置/获取虚拟键状态

类型	虚拟按键指令
描述	<p>读取虚拟按键状态，1-按下、0-松开。</p> <p>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。</p>
语法	<p>VKEY_STATE (keynum,state)</p> <p>state=VKEY_STATE(keynum)</p> <p>keynum: 虚拟按键编号</p> <p>state: 虚拟按键状态</p>
适用控制器	支持 RTHmi
例子	<p>例一：</p> <p>'设置虚拟按键状态，通过功能键选择动作“Call Sub Twice”，分别指定按下时和松开时调用函数</p> <pre>global sub VKeyPress_A() VKEY_STATE (65, 1) '虚拟键'A'按下</pre>

	<pre>end sub global sub VKeyRelease_A() VKEY_STATE (65, 0) '虚拟键'A'松开 end sub 例二: '自定义控件刷新函数内获取虚拟按键状态 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</pre>
相关指令	VKEY_SCAN , VKEY_EVENT

6.5.7.VKEY_EVENT -- 获取虚拟键状态变化

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 RTHmi
例子	<pre>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</pre>
相关指令	VKEY_SCAN , VKEY_STATE

6.5.8.VKEY_SCAN -- 获取虚拟键编码

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 RTHmi

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

6.5.9.VKEY_INPUT -- 输入虚拟键值内容到键盘窗口

类型	按键指令
描述	键盘窗口上控件模拟虚拟按键输入。
语法	VKEY_INPUT (vkeynum) vkeynum: 虚拟按键编号
适用控制器	支持 RTHmi
例子	/
相关指令	VKEY_STATE , VKEY_EVENT

6.5.10.VKEY_IME -- 设置/获取当前输入法

类型	虚拟按键指令
描述	设置/获取当前输入法类型 中文输入仅对虚拟键 VKEY_a~VKEY_z 有效，对输入的大写字符无效。
语法	ime = VKEY_IME () VKEY_IME (ime) ime: 当前输入法类型 0: 默认值，英文输入法 1: 中文拼音输入法
适用控制器	支持 RTHmi
例子	VKEY_IME (1) ? VKEY_IME () ; 输出 1

6.5.11.ZSIMU_KEY -- 仿真物理按键

类型	按键指令
描述	仿真物理按键。
语法	ZSIMU_KEY(keycode, state) keycode: 物理按键编码 state: 按下状态, 1-按下
适用控制器	支持 RTHmi
例子	ZSIMU_KEY(3, 1) '仿真物理键 3 被按下
相关指令	ZSIMU_VKEY

6.5.12. ZSIMU_VKEY -- 仿真虚拟按键

类型	按键指令
描述	仿真虚拟按键。
语法	ZSIMU_VKEY(vkeycode, state) vkeycode: 虚拟按键编码 state: 按下状态, 1-按下
适用控制器	支持 RTHmi
例子	ZSIMU_VKEY(4, 1) '仿真虚拟键 4 被按下
相关指令	ZSIMU_KEY

6.6. 操作指令

6.6.1.HMI_SHOWWINDOW -- 显示指定窗口

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] [,ifclosewithfather] [,fatherwindow]) 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_KEYBOARD 弹出软键盘窗口类型时，对应要编辑的本窗口元件 ID ifclosewithfather: 是否随父窗口一起关闭，缺省 1=true fatherwindow: 指定父窗口，缺省当前窗口 注意: 1.Modifycontrol 参数不可缺省，且元件 ID 必须是支持软键盘输入的元件。 2.软键盘默认垄断，无法通过元件关闭，但可使用指令操作关闭，同时随所在窗口的关闭而关闭。 3.从哪个窗口打开另一个窗口，即为父子窗口关系。
适用控制器	支持 RTHmi

例子	<p>例一：在窗口 10 设置按键弹出 13 窗口，窗口 10 即为窗口 13 的父窗口。 HMI_SHOWWINDOW(12,6) ‘弹出窗口 12 HMI_SHOWWINDOW(13,6,0,1) ‘弹出窗口 13 并随父窗口关闭 HMI_SHOWWINDOW(13,6,0,1,11) ‘弹出窗口 13 并指定随父窗口 11 关闭</p> <p>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</p>
相关指令	HMI_CLOSEWINDOW

6.6.2.HMI_CLOSEWINDOW -- 关闭窗口

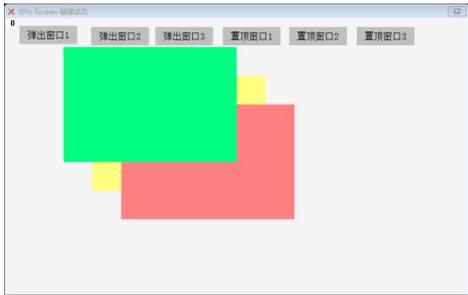
类型	窗口操作指令
描述	关闭指定窗口。
语法	HMI_CLOSEWINDOW([winid]) winid: 窗口号，缺省 0-当前函数调用的元件所在的窗口，其它编号- Hmi 组态里面的窗口号
适用控制器	支持 RTHmi
例子	HMI_CLOSEWINDOW() ‘关闭当前窗口
相关指令	HMI_SHOWWINDOW

6.6.3.HMI_BASEWINDOW -- 切换基本窗口

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 RTHmi
例子	HMI_BASEWINDOW = 11 ‘切换到 11 号基本窗口
相关指令	HMI_SHOWWINDOW

6.6.4.HMI_FOCUSWINDOW -- 窗口焦点模式

类型	窗口操作指令
描述	设置/获取 HMI 窗口是否随鼠标点击焦点自动进行切换（被点中的窗口将自动在最上层显示） 注：弹出窗口即使切换到最顶层，也始终在置顶窗口的下方。
语法	HMI_FOCUSWINDOW(mode) mode = HMI_FOCUSWINDOW() mode: 焦点窗口切换模式

	<p>0: 默认不切换</p> <p>1: bit0=1, 弹出窗口 (POP) 随鼠标点击焦点切换</p> <p>2: bit1=1, 置顶窗口 (TOP) 随鼠标点击焦点切换</p> <p>注: 弹出窗口使用该指令焦点切换不能覆盖置顶窗口。</p>
适用控制器	支持 RTHmi
例子	<p>HMI_FOCUSWINDOW(1) '设置弹出窗口随鼠标焦点自动切换</p> <p>HMI_FOCUSWINDOW(2) '设置置顶窗口随鼠标焦点自动切换</p> <p>HMI_FOCUSWINDOW(1+2) '设置置顶和弹出窗口都随鼠标焦点自动切换</p> <p>以弹出窗口为例 (其余模式等同), 运行效果图如下:</p> 

6.6.5.HMI_LASTWINDOW -- 最后点击窗口

类型	窗口操作指令
描述	获取最后点击的窗口
语法	winid = HMI_LASTWINDOW()
适用控制器	支持 RTHmi
例子	? HMI_LASTWINDOW() '打印最后点击的窗口 ID

6.6.6.HMI_DEFAULTATTR -- 设置/获取 HMI 内置默认属性

类型	HMI 指令
描述	<p>设置/获取 HMI 内置的默认属性, 对整个 HMI 生效。</p> <p>注: 需要在 HMI 初始化前进行设置才会生效。</p>
语法	<p>value = HMI_DEFAULTATTR(strAttr)</p> <p>HMI_DEFAULTATTR(strAttr, value)</p> <p>strAttr: 指定操作的控件属性</p> <ul style="list-style-type: none"> "SBR_W": 滚动条宽度 "SBR_F": 滚动条前景颜色 "SBR_B": 滚动条背景颜色 "SBR_P": 滚动条按压颜色 "BTN_RLM": 按钮松开模式 <p>0-默认</p> <p>1-鼠标移出控件取消触发</p>

	<p>“DRAW_M”: 绘图模式 0-默认点阵绘图 1-抗锯齿绘图</p> <p>“MONO_M”: 控件垄断模式（有弹窗的控件，如下拉列表、菜单栏等） 0-不垄断 1-垄断按键 2-垄断鼠标 3-同时垄断按键鼠标（默认）</p> <p>“RGB_GRAY”: 灰阶颜色</p> <p>注：在 HMI 程序启动之前，必须预先配置好 SBR_W、SBR_F、SBR_B、SBR_P 以及 RGB_GRAY 等参数。为此，可以在 Basic 程序的末尾 添加一个专门用于启动 HMI 的函数。这样做可以可以使必要参数都已正确设置后再启动 HMI，以确保其按照预期正常工作。</p> <p>value: 属性值</p>
适用控制器	支持 RTHmi
例子	HMI_DEFAULTATTR("SBR_W", 25) '修改滚动条默认宽度为 25 HMI_DEFAULTATTR("SBR_P", RGB(255,0,0)) '修改滚动条默认按压颜色

6.6.7.HMI_DEALINFO -- 获取 HMI 处理信息

类型	HMI 指令
描述	获取当前 HMI 处理的信息
语法	<p>value = HMI_DEALINFO(strAttr)</p> <p>strAttr: 指定获取的 HMI 处理信息 "CURWIN": 当前窗口号 "CURCTRL": 当前控件号 "TOPWIN": 最顶层显示的窗口号 "EDITWIN": 当前编辑输入窗口 "EDITCTRL": 当前编辑输入控件 "EDITNEXT": 当前焦点的下一个编辑控件 "EDITPREV": 当前焦点的上一个编辑控件</p> <p>value: 数值</p> <p>注：“EDITWIN”、“EDITCTRL”、“EDITNEXT”、“EDITPREV”参数需要在虚拟键模式下才生效</p>
适用控制器	支持 RTHmi
例子	<p>例一： global dim aaa aaa=HMI_DEALINFO("CURWIN") '打印当前窗口号 ?aaa</p> <p>例二：在 HMI 窗口中添加“可编辑”的控件 VKEY_MODE(1) '开启虚拟键输入模式</p>

	? HMI_DEALINFO("EDITCTRL") '打印当前编辑的控件编号 打印结果为：当前编辑的控件编号
--	--

6.6.8.HMI_CONTROLSIZEX -- 获取元件宽度

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 RTHmi
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

6.6.9.HMI_CONTROLSIZEY -- 获取元件高度

类型	窗口操作指令
描述	获取元件高度
语法	value= HMI_CONTROLSIZEY ([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 RTHmi
例子	PRINT HMI_CONTROLSIZEY (10,11) '打印 10 号窗口 11 号元件高度
相关指令	HMI_CONTROLSIZEX

6.6.10.HMI_CONTROLDATA -- 设置/获取自定义元件属性

类型	窗口操作指令
描述	获取或设置自定义元素的特殊属性, 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 RTHmi
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

6.6.11. HMI_CONTROLBACK -- 设置/获取指定元件背景颜色

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 RTHmi
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

6.6.12. HMI_CONTROLVALID -- 设置/获取元件使能

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 RTHmi
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	/

6.6.13. HMI_CONTROLSTRING -- 获取字符串信息

类型	窗口操作指令
描述	获取字符串控件或输入显示控件的字符串信息。 5 系列 20180405 以上固件支持。
语法	string= HMI_CONTROLSTRING ([winid, controlid]) HMI_CONTROLSTRING (winid, controlid) = string 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 String: 字符串
适用控制器	支持 RTHmi
例子	VRSTRING(0,10)= HMI_CONTROLSTRING (10, 2) '10 号窗口第 2 个元件的数据读取到 VR
相关指令	/

6.6.14. HMI_CONTROLATTR -- 设置/获取元件属性

类型	控件操作指令
描述	获取/设置控件的指定属性值 注：操作前需确保操作的控件具有该属性。
语法	<p>value=HMI_CONTROLATTR(strProperty[, winid, controlid]) HMI_CONTROLATTR(strProperty, value[, winid, controlid])</p> <p>strProperty:指定操作的控件属性</p> <ul style="list-style-type: none"> "POSX": 控件 X 位置 "POSY": 控件 Y 位置 "SIZEX": 控件宽度 "SIZEY": 控件高度 "MIN": 最小值（包含该属性的单一状态控件才有效，下同） "MAX": 最大值 "FORE": 前景颜色 "BACK": 背景颜色 "PWD": 密码显示 "DECS": 小数位数 "STATE": 控件状态 "POPOP": 弹出状态（下拉列表） "CURSORX": 控件输入光标列 "CURSORY": 控件输入光标行 "FOCUS": 控件焦点状态 "MSELROWS": 多选行数，正数往下多选，负数往上多选，0 表示无多选。配合"CURSORY"参数获取光标行可知道当前选中行区域。 <p>value: 属性值 winid: HMI 文件里面窗口编号，缺省为当前窗口 controlid: 元件编号，缺省为当前自定义元件</p>
适用控制器	支持 RTHmi
例子	<p>例一：</p> <p>HMI_CONTROLATTR ("FORE",RGB(255,0,0),10,1) '设置窗口 10 控件 1 的前景颜色为红色</p> <p>HMI_CONTROLATTR ("BACK",RGB(255,255,255),10,1) '设置窗口 10 控件 1 的背景颜色为白色</p> <p>➤ 设置前的样式：</p> <div style="background-color: #e0e0e0; padding: 10px; text-align: center; margin-top: 10px;"> <p>深圳市正运动技术</p> </div>

➤ 设置后的样式:



深圳市正运动技术

例二:

VRSTRING(0,8) = "123456"

HMI_CONTROLATTR("PWD",123456,10,5) '设置窗口 10 控件 5 的字符为密码显示

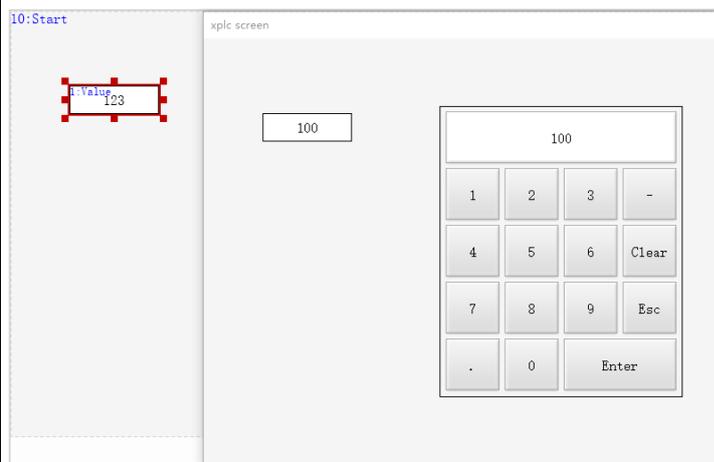


例三:

HMI_CONTROLATTR("MIN",10,10,6) '设置值显示控件的最小值

HMI_CONTROLATTR("MAX",100,10,6) '设置值显示控件的最大值

注: 最小值数值必须比最大值小才有效, 大于等于均无效。同时应考虑“属性”栏中的最小值和最大值的传入值。



6.6.15. HMI_CONTROLTEXT -- 修改元件文本

类型	控件操作指令
描述	修改控件显示文本, 支持所有带格式文本(可输入)的控件。 注: 修改文本长度不能超过当前控件文本, 否则会被截断。
语法	HMI_CONTROLTEXT (winid, controlid, state, strText) winid: HMI 文件里面窗口编号 controlid: 元件编号 state: 指定修改哪个状态的控件文本 strText: 修改控件显示文本
适用控制器	支持 RTHmi

例子	HMI_CONTROLTEXT (10, 1, 0, "松开") '修改按钮松开状态显示文本
	HMI_CONTROLTEXT (10, 1, 1, "按下") '修改按钮按下状态显示文本

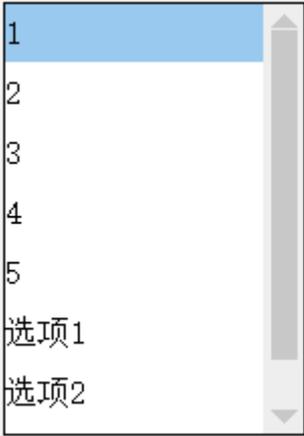


修改后元件文本显示情况：
 左图：0 状态时显示“松开”文本；
 右图：1 状态时显示“按下”文本；




6.6.16. HMI_LISTTEXTS -- 修改列表元件文本

类型	控件操作指令
描述	强制修改列表控件的文本列表，具体应用例程参见 动态列表 。 注意：使用该指令须先开启“列表”控件的“动态项”属性。
语法	<p>HMI_LISTTEXTS (winid, controlid, strList,[,mode] [,lineheight])</p> <p>winid: Hmi 文件里面窗口编号 controlid: 元件编号 strList: 文本列表，以换行符隔开 mode: 可选，修改模式，缺省 0 0 - 覆盖方式，清空原来的列表项重新写入新的列表项 1 - 追加方式，在原来的列表项基础上末尾增加新的列表项，且滚动条自动跟随到追加位置 lineheight: 可选，强制指定行高度，缺省-1 不指定 -1 - 自动计算行高度，行高度将随列表项目数量变化 =0 - 使用当前行高度 >0 - 强制指定行高度，小于最小行高度时按最小行高度设置 最小行高度 = 字高 + 2*行间距</p> <p>注意： 1. 字符串最大字符数限制 2049。 2. 使用该指令修改的列表项，其列表编号是逐行递增的，不能指定编号，追加操作的列表项也是。 3. 非追加操作方式，列表控件的滚动条位置将会自动复位，追加操作方式可以选择保持滚动条在追加位置。 4. 列表控件对于动态列表项缓冲区有大小限制，最多 256 项，当加满 256 项时，每一项列表项文本不得超过 12 个字符。</p>
适用控制器	支持 RTHmi

例子	<pre>dim strList(100) = "选项 1\n 选项 2\n 选项 3"</pre> <p>HMI_LISTTEXTS (10, 3, strList,1,30) '在原有的列表项基础上追加新的列表项, 并指定行高度为 30</p> <p>显示结果如下:</p> 
----	---

6.6.17. HMI_LISTITEM -- 修改指定列表项文本

类型	控件操作指令
描述	<p>获取/强制修改列表控件的指定列表项文本</p> <p>注意：使用该指令须先开启“列表”控件的“动态项”属性。</p>
语法	<pre>HMI_LISTITEM (winid, controlid, id, strText)</pre> <pre>strText = HMI_LISTITEM (winid, controlid, id)</pre> <p>winid: Hmi 文件里面窗口编号</p> <p>controlid: 元件编号</p> <p>id: 要修改的列表项 ID</p> <p>strText: 修改的列表项文本</p>
适用控制器	支持 RTHmi
例子	<pre>dim strList(100) = "选项 1\n 选项 2\n 选项 3"</pre> <p>Hmi_LISTTEXTS (10, 3, strList) '修改列表控件文本, 自动更新状态数</p> <p>Hmi_LISTITEM (10, 3, 1, "TWO") '修改元件 3 中第二个列表项为“TWO”</p> <p>显示结果如下:</p> 

6.6.18. HMI_STRAPPEND -- 元件文本追加

类型	控件操作指令
描述	<p>在“字符显示”元件中追加字符串</p> <p>注：使用该指令需在元件“属性”中设置为多行文本状态才生效。</p>
语法	<p>HMI_STRAPPEND (strAppend, winid, controlid, mode [,ifNewLine])</p> <p>strAppend: 追加的字符串</p> <p>winid: HMI 文件里面窗口编号</p> <p>controlid: 元件编号</p> <p>mode: 追加模式 (bit)，按 bit0 和 bit1 的组合值确定模式。（当字符串超过最大字符数限制时，会对添加后的总字符串按模式清除溢出部分）</p> <p>bit0: 溢出方向</p> <p>0 - 超限从末尾处清除溢出部分，即超限部分不追加</p> <p>1 - 从字符串开头处清除溢出部分</p> <p>bit1: 追加方向</p> <p>0 - 从末尾处追加，滚动条自动跟随到文本末尾处</p> <p>1 - 从起始处追加，滚动条自动跟随到文本起始处</p> <p>ifNewLine: 是否追加新行，缺省 0（否）</p> <p>注：追加新行默认占用一个字符空间</p>
适用控制器	支持 RTHmi
例子	<p>例一：HMI_STRAPPEND ("zmotion",10, 2, 0, 0) '向元件文本末尾处追加一行字符串</p> <div style="display: flex; justify-content: space-around; align-items: center;"> <div style="border: 1px solid black; padding: 5px; width: 150px; height: 100px; display: flex; align-items: center; justify-content: center;"> <p>正运动技术</p> </div> <div style="border: 1px solid black; padding: 5px; width: 150px; height: 100px; display: flex; align-items: center; justify-content: center;"> <p>正运动技术zmotion</p> </div> </div> <p>例二：对元件文本限制字符数为 16，对该元件追加文本 HMI_STRAPPEND ("zmotion",10, 2, 2, 1) '向元件文本起始处追加字符串，且追加新行</p> <p>运行效果如下：仅显示不超过 16 个字符数的内容，且从末尾处清除溢出部分</p> <div style="border: 1px solid black; padding: 5px; width: 150px; height: 100px; display: flex; flex-direction: column; align-items: center; justify-content: center;"> <p>zmotion</p> <p>正运动技</p> </div>

6.6.19. HMI_IFMONO -- 获取窗口垄断状态

类型	窗口操作指令
描述	自定义元素 reflash 刷新函数里使用，判断当前元素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 RTHmi
例子	<pre>GLOBAL SUB reflash() IF SCAN_EVENT(Hmi_IFMONO)<0 THEN ?"返回自定义元素窗口" ENDIF IF SCAN_EVENT(Hmi_IFMONO)>0 THEN ?"离开自定义元素窗口" ENDIF END SUB</pre>

6.6.20. HMI_WINDOWSTATE -- 获取窗口状态

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	<p>value= HMI_WINDOWSTATE (winid [, tablenum])</p> <p>winid: Hmi 文件里面窗口编号</p> <p>tablenum: 存储窗口的位置和大小，顺序存储 posx, posy, sizex, sizey</p> <p>返回值对应的窗口类型：</p> <p>ZPLC_WIN_TYPE_AUTO = 0, 没有显示</p> <p>ZPLC_WIN_TYPE_TOP = 1, 顶层窗口</p> <p>ZPLC_WIN_TYPE_BOTTOM = 2, 底层窗口</p> <p>ZPLC_WIN_TYPE_BASE = 4, 基本窗口</p> <p>ZPLC_WIN_TYPE_KEYBOARD = 5, 软键盘.</p> <p>ZPLC_WIN_TYPE_POP = 6, 弹出窗口</p> <p>ZPLC_WIN_TYPE_MENU = 7, 菜单窗口，自动关闭</p>
适用控制器	支持 RTHmi
例子	 <pre>>>?HMI_WINDOWSTATE(10) 4</pre> <p>读取结果：4，表示 10 号窗口为基本窗口</p>

6.6.21. HMI_MOVEWINDOW -- 移动指定窗口

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 RTHmi
例子	HMI_MOVEWINDOW (11,100,100) 将 11 号窗口的显示位置改为 (100,100)

6.6.22. HMI_TABLEVALUE -- 设置/获取表格数值

类型	报表视图控件操作指令
描述	设置/获取表格指定项数值 表格项指定为数值类型时调用该指令设置获取数据。 获取数值时，多选只会返回选中的第一个单元格数值。
语法	设置/获取表格数值，只写入单个数据 HMI_TABLEVALUE (winid, controlid, row, col, value) value = HMI_TABLEVALUE (winid, controlid, row, col) winid: HMI 文件里面窗口编号 controlid: 元件编号 row: 表格指定行，-1 表示选中所有行 col: 表格指定列，-1 表示选中所有列 value: 表格指定项数值 多选时，选中的项目都写入相同 value 值。 获取时，不支持多选。 批量设置/获取表格数值，从 table 写入或读取 HMI_TABLEVALUE (winid, controlid, row, col, tabid, ifset) tabid: 表格数值保存得 table 起始位置 Ifset: 是否写入，0-读取，1-写入 注：选中光标行和列同时等于-1 时表示为全选，表示不选中，该指令只能选中一个单元格、一行或者一列。行号、列号从 0 开始编码（表头行/列不计入行/列号）！
适用控制器	支持 RTHmi
例子	HMI_TABLEVALUE(16, 1 ,2, 1, 500) 将 16 号窗口第 1 个控件的第 3 行 2 列单元格的数值设置为 500

运行效果如下：
 ➤ 在“在线命令”中输入该指令，表格中第 3 行 2 列单元格的数值被设为 500

	轴数	速度	加速度	减速度
1	0	100	10	10
2	1			
3	2			
4	3			

	轴数	速度	加速度	减速度
1	0	100	10	10
2	1			
3	2	500		
4	3			

6.6.23. HMI_TABLETEXT -- 设置/获取表格内容

类型	报表视图控件操作指令												
描述	设置/获取表格指定内容（字符串） 表格项指定为字符串类型时调用该指令设置获取数据。												
语法	<p>HMI_TABLETEXT (winid, controlid, row, col, "string") string = HMI_TABLETEXT (winid, controlid, row, col)</p> <p>winid: HMI 文件里面窗口编号 controlid: 元件编号 row: 表格指定行, -1 表示选中所有行 col: 表格指定列, -1 表示选中所有列 string: 表格指定项内容（字符串）</p> <p>多选时，使用表格分隔符","分割多个字符串，使用换行符"CHR(10)"给字符串换行。</p> <p>如：HMI_TABLETEXT(10,1,-1,-1,"aaa, bbb, ccc + CHR(10) + ddd, eee, fff, ggg, hhh, iii")表示选中 10 号窗口第一个控件的所有行和列，为其设置数据如下：两行 9 个单元格数据，第一行 3 个数据，第二行 6 个数据，如下表格所示。</p> <table border="1"> <tr> <td>aaa</td> <td>bbb</td> <td>ccc</td> <td></td> <td></td> <td></td> </tr> <tr> <td>ddd</td> <td>eee</td> <td>fff</td> <td>ggg</td> <td>hhh</td> <td>iii</td> </tr> </table> <p>注：指令只会操作选中部分。行号、列号从 0 开始编码（表头行/列不计入行/列号）！</p>	aaa	bbb	ccc				ddd	eee	fff	ggg	hhh	iii
aaa	bbb	ccc											
ddd	eee	fff	ggg	hhh	iii								
适用控制器	支持 RTHmi												
例子	<p>HMI_TABLETEXT(16, 1, 1, 1, "x/t") 将 16 号窗口第 1 个控件的第 2 行 2 列单元格的内容设置为 x/t</p> <p>运行效果如下： ➤ 在“在线命令”中输入该指令，表格中第 2 行 2 列单元格的内容被设为 x/t</p>												

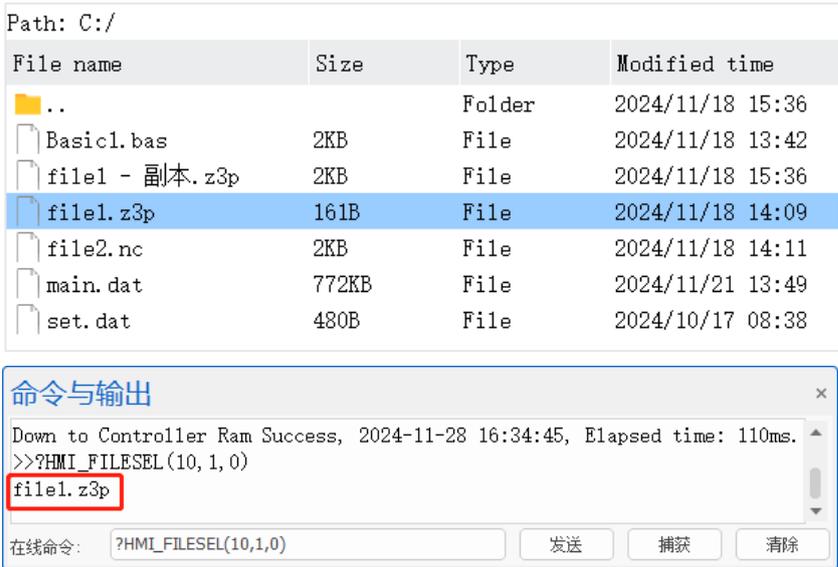
	轴数	速度	加速度	减速度
1	0	100	10	10
2	1			
3	2			
4	3			

	轴数	速度	加速度	减速度
1	0	100	10	10
2	1	x/t		
3	2			
4	3			

6.6.24. HMI_TABLECURSOR -- 获取当前选中行列

类型	报表视图控件操作指令																									
描述	获取指定报表控件当前选中行列编号，将其保存到 table 表中。																									
语法	<p>HMI_TABLECURSOR (winid, controlid, num)</p> <p>winid: HMI 文件里面窗口编号</p> <p>controlid: 元件编号</p> <p>num: 当前选中行号、列号分别存储在 table(num), table(num+1)</p> <p>行 = -1 表示选中所有行，列 = -1 表示选中所有列。</p> <p>行列同时 = -1 表示未选中。</p> <p>注：行号、列号从 0 开始编码（表头行/列不计入行/列号）！</p>																									
适用控制器	支持 RTHmi																									
例子	<p>HMI_TABLECURSOR (16,1,2) '获取 16 号窗口第 1 个控件当前选中的行列编号，将其分别保存到 table(2)和 table(3)中</p> <p>运行效果如下：</p> <p>➤ 选中单元格后，在“在线命令”中输入“?table(2)”以及“?table(3)”，“命令与输出栏”打印当前选中单元格的行列号</p> <table border="1"> <thead> <tr> <th></th> <th>轴数</th> <th>速度</th> <th>加速度</th> <th>减速度</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>0</td> <td>100</td> <td>10</td> <td>10</td> </tr> <tr> <td>2</td> <td>1</td> <td></td> <td></td> <td></td> </tr> <tr> <td>3</td> <td>2</td> <td></td> <td></td> <td></td> </tr> <tr> <td>4</td> <td>3</td> <td></td> <td></td> <td></td> </tr> </tbody> </table> <div style="border: 1px solid gray; padding: 5px;"> <p>命令与输出</p> <p>Down to Controller Ram Success, 2024-11-28 16:39:57, Elapsed time: 437ms.</p> <pre>>>?table(2) 0 >>?table(3) 1</pre> <p>在线命令: <input type="text" value="?table(3)"/> <input type="button" value="发送"/> <input type="button" value="捕获"/> <input type="button" value="清除"/></p> </div>		轴数	速度	加速度	减速度	1	0	100	10	10	2	1				3	2				4	3			
	轴数	速度	加速度	减速度																						
1	0	100	10	10																						
2	1																									
3	2																									
4	3																									

6.6.25. HMI_FILESEL-- 获取当前选中文件/文件夹名称

类型	文件浏览器控件操作指令
描述	获取文件浏览器控件当前选中文件/文件夹名称（不带路径）
语法	filename = HMI_FILESEL(winid, controlid [,tabid]) winid: HMI 文件里面窗口编号 controlid: 元件编号 tabid: table 编号, 可选, 输出是否选中文件夹 table(tabid)=0 为文件, =1 为文件夹, =-1 为未选中 filename: 返回当前选中的文件/文件夹名称, 未选中状态返回空
适用控制器	支持 RTHmi
例子	<p>?HMI_FILESEL(10,1,0) 获取并打印 10 号窗口第 1 个控件当前选中文件的名称 运行效果如下:</p> <p>➤ 选中文件后, 在“在线命令”中输入该指令, “命令与输出栏”打印选中文件的名称</p>  <p>The example shows a file browser window with the path 'C:/' and a list of files. The file 'file1.z3p' is selected. Below it, a terminal window titled '命令与输出' shows the command '?HMI_FILESEL(10, 1, 0)' being executed, resulting in the output 'file1.z3p'.</p>

6.6.26. HMI_FILEPATH – 设置/获取当前路径

类型	文件浏览器控件操作指令
描述	设置/获取当前路径（以 '/' 结尾）
语法	HMI_FILEPATH(winid, controlid, filepath) filepath = HMI_FILEPATH(winid, controlid) winid: HMI 文件里面窗口编号 controlid: 元件编号 filepath: 返回/设置当前路径 接受盘符“C:/”和“A:/”, 不输入盘符时默认 C 盘 C 盘为 flash 目录, A 盘为 U 盘目录
适用控制器	支持 RTHmi

例子	<p>HMI_FILEPATH(10, 1, "A:") 设置 10 号窗口第 1 个控件的当前路径为 A 盘 运行效果如下：</p> <p>➤ 当前路径默认 C 盘</p> <table border="1"> <tr><td colspan="4">Path: C:/</td></tr> <tr> <th>File name</th> <th>Size</th> <th>Type</th> <th>Modified time</th> </tr> <tr> <td>..</td> <td></td> <td>Folder</td> <td>2024/11/18 15:36</td> </tr> <tr> <td>Basic1.bas</td> <td>2KB</td> <td>File</td> <td>2024/11/18 13:42</td> </tr> <tr> <td>file1 - 副本.z3p</td> <td>2KB</td> <td>File</td> <td>2024/11/18 15:36</td> </tr> <tr> <td>file1.z3p</td> <td>161B</td> <td>File</td> <td>2024/11/18 14:09</td> </tr> <tr> <td>file2.nc</td> <td>2KB</td> <td>File</td> <td>2024/11/18 14:11</td> </tr> <tr> <td>main.dat</td> <td>772KB</td> <td>File</td> <td>2024/11/21 13:49</td> </tr> <tr> <td>set.dat</td> <td>480B</td> <td>File</td> <td>2024/10/17 08:38</td> </tr> </table> <p>➤ 在“在线命令”中输入该指令后，当前路径被设为 A 盘</p> <table border="1"> <tr><td colspan="4">Path: A:/</td></tr> <tr> <th>File name</th> <th>Size</th> <th>Type</th> <th>Modified time</th> </tr> <tr> <td>..</td> <td></td> <td>Folder</td> <td>2024/10/11 09:04</td> </tr> </table>	Path: C:/				File name	Size	Type	Modified time	..		Folder	2024/11/18 15:36	Basic1.bas	2KB	File	2024/11/18 13:42	file1 - 副本.z3p	2KB	File	2024/11/18 15:36	file1.z3p	161B	File	2024/11/18 14:09	file2.nc	2KB	File	2024/11/18 14:11	main.dat	772KB	File	2024/11/21 13:49	set.dat	480B	File	2024/10/17 08:38	Path: A:/				File name	Size	Type	Modified time	..		Folder	2024/10/11 09:04
	Path: C:/																																																
File name	Size	Type	Modified time																																														
..		Folder	2024/11/18 15:36																																														
Basic1.bas	2KB	File	2024/11/18 13:42																																														
file1 - 副本.z3p	2KB	File	2024/11/18 15:36																																														
file1.z3p	161B	File	2024/11/18 14:09																																														
file2.nc	2KB	File	2024/11/18 14:11																																														
main.dat	772KB	File	2024/11/21 13:49																																														
set.dat	480B	File	2024/10/17 08:38																																														
Path: A:/																																																	
File name	Size	Type	Modified time																																														
..		Folder	2024/10/11 09:04																																														

6.6.27. HMI_FILEFILTER -- 设置文件过滤器

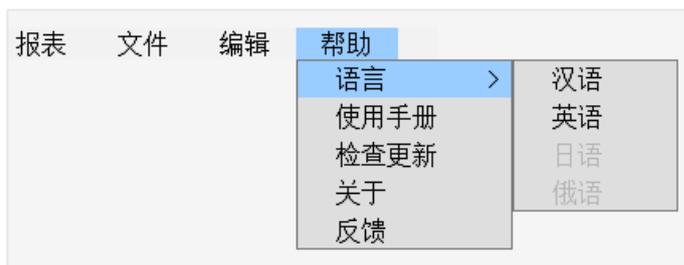
类型	文件浏览器控件操作指令
描述	设置文件过滤器，过滤显示当前路径所有文件 注：文件过滤器只过滤文件，不过滤文件夹
语法	<p>HMI_FILEFILTER(winid, controlid, strfilter)</p> <p>strfilter = HMI_FILEFILTER (winid, controlid)</p> <p>winid: HMI 文件里面窗口编号</p> <p>controlid: 元件编号</p> <p>strfilter: 文件过滤器选项，支持通配符 '*'，以 ' ' 隔开多个过滤项</p> <p>输入 "", 表示取消文件过滤器，显示所有文件</p> <p>输入 "*.*", 表示显示所有带后缀的文件</p> <p>输入 "*.nc .cnc", 表示显示所有 nc、cnc 文件</p> <p>注：过滤器选项不区分大小写</p>
适用控制器	支持 RTHmi
例子	<p>HMI_FILEFILTER(10, 1, "*.bas") 显示 10 号窗口第 1 个控件的所有 bas 文件 运行效果如下：</p> <p>➤ 文件浏览器默认显示 C 盘所有文件</p>

<div style="font-size: 2em; font-weight: bold; margin-bottom: 10px;">▶</div>	Path: C:/ <table border="1"> <thead> <tr> <th>File name</th> <th>Size</th> <th>Type</th> <th>Modified time</th> </tr> </thead> <tbody> <tr> <td>..</td> <td></td> <td>Folder</td> <td>2024/11/18 15:36</td> </tr> <tr> <td>Basic1.bas</td> <td>2KB</td> <td>File</td> <td>2024/11/18 13:42</td> </tr> <tr> <td>file1 - 副本.z3p</td> <td>2KB</td> <td>File</td> <td>2024/11/18 15:36</td> </tr> <tr> <td>file1.z3p</td> <td>161B</td> <td>File</td> <td>2024/11/18 14:09</td> </tr> <tr> <td>file2.nc</td> <td>2KB</td> <td>File</td> <td>2024/11/18 14:11</td> </tr> <tr> <td>main.dat</td> <td>772KB</td> <td>File</td> <td>2024/11/21 13:49</td> </tr> <tr> <td>set.dat</td> <td>480B</td> <td>File</td> <td>2024/10/17 08:38</td> </tr> </tbody> </table>	File name	Size	Type	Modified time	..		Folder	2024/11/18 15:36	Basic1.bas	2KB	File	2024/11/18 13:42	file1 - 副本.z3p	2KB	File	2024/11/18 15:36	file1.z3p	161B	File	2024/11/18 14:09	file2.nc	2KB	File	2024/11/18 14:11	main.dat	772KB	File	2024/11/21 13:49	set.dat	480B	File	2024/10/17 08:38
	File name	Size	Type	Modified time																													
..		Folder	2024/11/18 15:36																														
Basic1.bas	2KB	File	2024/11/18 13:42																														
file1 - 副本.z3p	2KB	File	2024/11/18 15:36																														
file1.z3p	161B	File	2024/11/18 14:09																														
file2.nc	2KB	File	2024/11/18 14:11																														
main.dat	772KB	File	2024/11/21 13:49																														
set.dat	480B	File	2024/10/17 08:38																														
在“在线命令”中输入该指令后，文件浏览器仅显示 C 盘所有 bas 文件 Path: C:/ <table border="1"> <thead> <tr> <th>File name</th> <th>Size</th> <th>Type</th> <th>Modified time</th> </tr> </thead> <tbody> <tr> <td>..</td> <td></td> <td>Folder</td> <td>2024/11/18 15:36</td> </tr> <tr> <td>Basic1.bas</td> <td>2KB</td> <td>File</td> <td>2024/11/18 13:42</td> </tr> </tbody> </table>	File name	Size	Type	Modified time	..		Folder	2024/11/18 15:36	Basic1.bas	2KB	File	2024/11/18 13:42																					
File name	Size	Type	Modified time																														
..		Folder	2024/11/18 15:36																														
Basic1.bas	2KB	File	2024/11/18 13:42																														

6.6.28. HMI_MENUITEM -- 菜单项状态获取/修改

类型	菜单控件操作指令
描述	获取/修改菜单项状态
语法	HMI_MENUITEM(id, option, value [,winid, controlid]) value = HMI_MENUITEM(id, option [,winid, controlid]) id: 菜单项编号 (ID) option: 数据选项 0 – 设置菜单项选中 (勾选) 状态 1 – 设置菜单项灰阶 (禁用) 状态 value: 选项值 option = 0 时, value = 0: 将此菜单项设为非选中状态 value ≠ 0: 将此菜单项设为选中状态 option = 1 时, value = 0: 将此菜单项设为正常状态 value ≠ 0: 将此菜单项设为灰阶状态 winid: HMI 文件里面窗口编号, 缺省当前窗口 controlid: 元件编号, 缺省当前元件
适用控制器	支持 RTHmi
例子	HMI_MENUITEM(1031, 0, 1, 10, 1) 将 10 号窗口第 1 个控件中菜单编号为 1031 的项设为选中状态 HMI_MENUITEM(1032, 1, 0, 10, 1) 将 10 号窗口第 1 个控件中菜单编号为 1032 的项设为灰阶状态 运行效果如下:

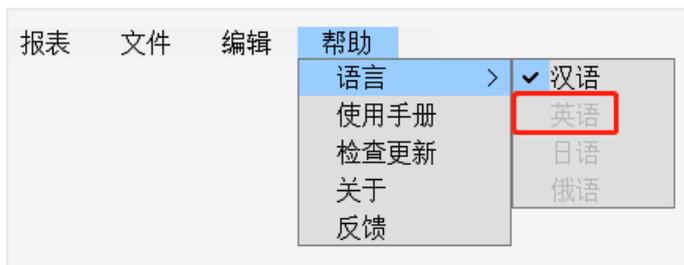
➤ 在本菜单中，“汉语”的菜单编号为 1031，“英语”的菜单编号为 1032



➤ 在“在线命令”中输入第一行指令后，菜单栏中“汉语”被勾选



➤ 在“在线命令”中输入第二行指令后，菜单栏中“英语”被禁用



第七章 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。

7.1. MOVEDTSP/MOVEDTABSSP -- DT 直线运动

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	<pre>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</pre>

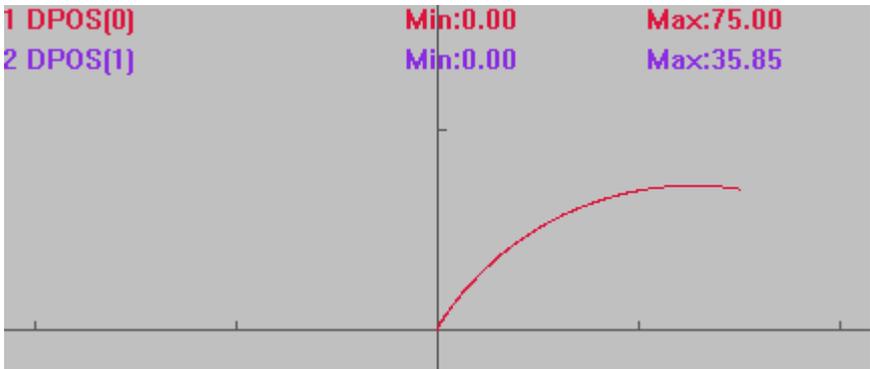
7.2. MOVECIRCDTSP/MOVECIRCDTABSSP -- DT 圆弧运动

类型	DT 运动函数
描述	<p>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。</p> <p>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。</p>
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	<pre>BASE(0,1,2) ATYPE=1,1,1 DPOS=0,0,0 TABLE(10,0,1) 'TABLE(10)存放轴 01</pre>

	<pre>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 'MOVECIRC2TABSSP(6,3,10,20,50,1) MOVECIRC2TSP(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</pre>						
	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="color: red;">1 DPOS[0]</td> <td style="color: red;">Min:0.00</td> <td style="color: red;">Max:6.70</td> </tr> <tr> <td style="color: purple;">2 DPOS[1]</td> <td style="color: purple;">Min:0.00</td> <td style="color: purple;">Max:50.00</td> </tr> </table> 	1 DPOS[0]	Min:0.00	Max:6.70	2 DPOS[1]	Min:0.00	Max:50.00
1 DPOS[0]	Min:0.00	Max:6.70					
2 DPOS[1]	Min:0.00	Max:50.00					

7.3. MOVECIRC2DTSP/MOVECIRC2DTABSSP -- DT 三点画圆弧运动

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	<pre> 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 </pre> 
----	---

7.4. MSPHERICALDTSP/MSPHERICALDTABSSP -- DT 空间圆弧运动

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 <table border="1" data-bbox="478 1653 1220 1825"> <thead> <tr> <th>值</th> <th>描述</th> </tr> </thead> <tbody> <tr> <td>-1</td> <td>在当前点的前面，只是参考，不运行到参考点</td> </tr> <tr> <td>0</td> <td>中间，运行到参考点</td> </tr> <tr> <td>1</td> <td>在当前点的和结束点的后面</td> </tr> </tbody> </table>	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
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	<pre> BASE(0,1,2) ATYPE=1,1,1 DPOS=0,0,0 </pre>								

	<pre> 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 </pre>
--	---

第八章 参考例程

8.1. 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。该例程是各元件的综合使用。

【实现目标】：通过 Hmi 元件实现单轴运动的控制及状态读取。

【设计思路】：

- 通过“功能键”调用 Basic 函数控制轴的运动和停止。
- 通过多个“值”元件进行数据读取/写入：读取运动状态、写入轴参数。
- 通过“位状态切换开关”实现运动模式、运动方向等的切换。

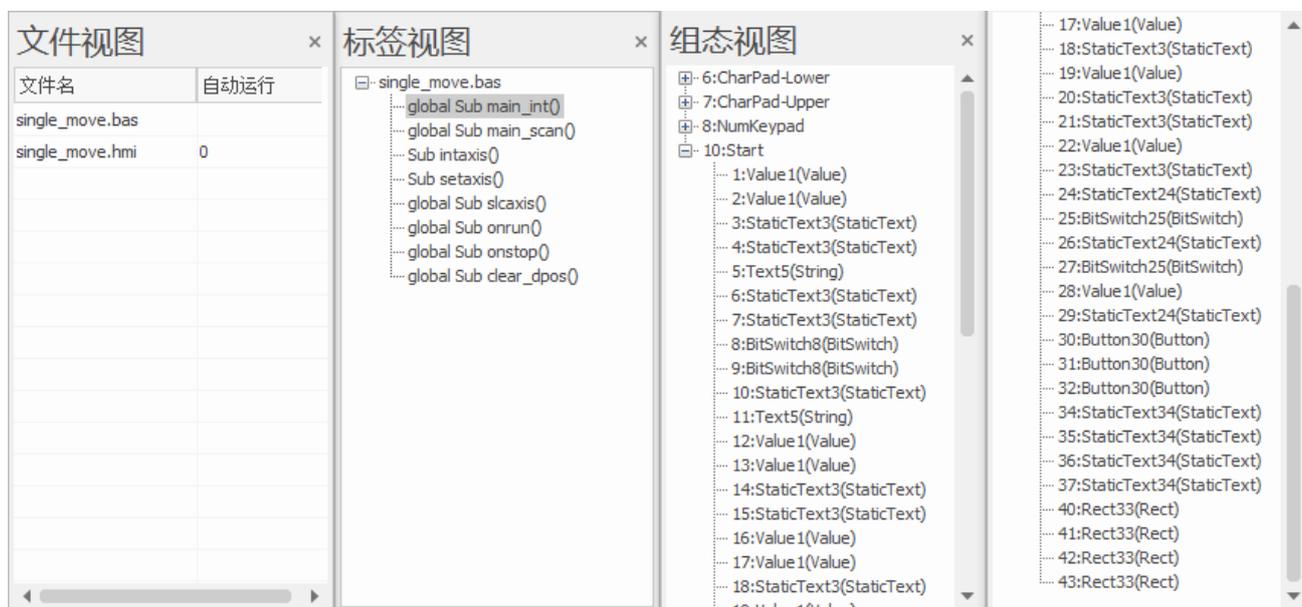
【界面创建】：

创建 Basic 文件和 Hmi 文件，定义 Hmi 需调用的全局 SUB 子函数。根据下图中内容添加 Hmi 组态元件。

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）



【例程使用】：

(一) Basic 程序界面

```

B single_move.bas × H single_move.hmi
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

```

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
```

```

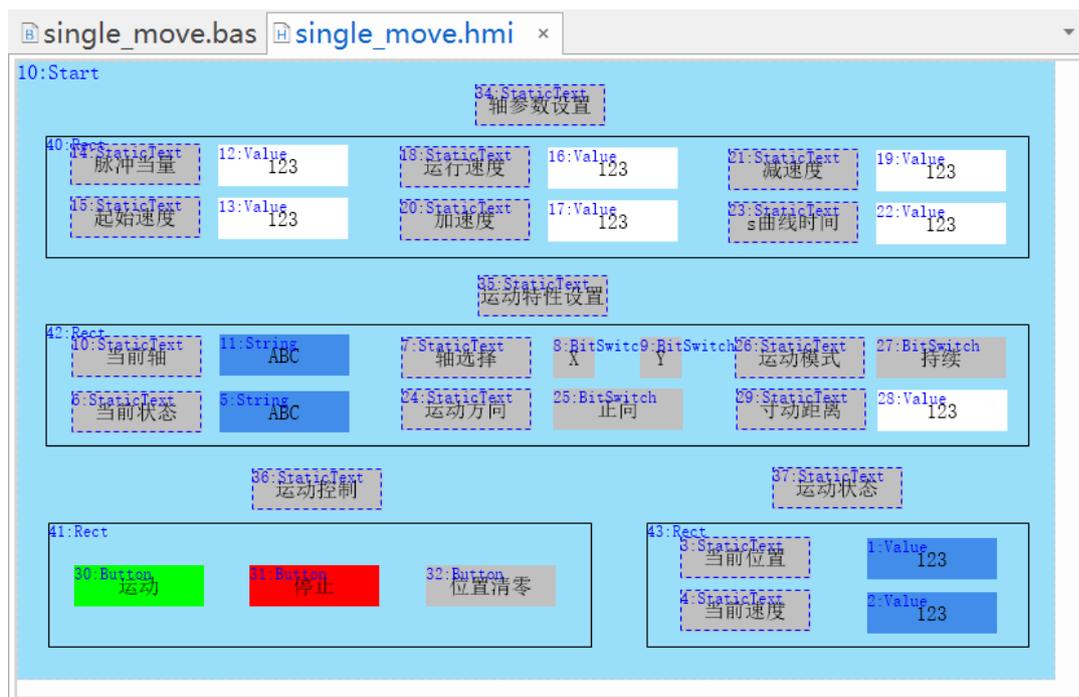
if idle=0 then
    state="运动"
endif
end sub

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
    
```

(二) Hmi 组态界面

1. 使用该例程时，先选择要运动的轴号，X 轴或 Y 轴，否则无法运动。再根据需求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。
2. 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，可点击运动让轴动起来。运行的速度 SPEED 和轴位置 DPOS 分别被读取到“值”元件 1 和 2 动态显示。
3. 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。



8.2. 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，参考例程如下。

【实现目标】：将示教盒 ZHD400X 的物理按键绑定自定义虚拟键功能。实现按下物理按键完成对应功能。

【设计思路】：

- 通过编写 Basic 程序定义虚拟键功能（以打印值功能为例）。
- 通过“按键转换”功能将物理按键键值与虚拟键键值绑定。
- HMI 使用“自定义”元件调用 Basic 程序中的刷新函数和绘图函数。

【操作流程】：

(一) Basic 程序定义变量及构建 SUB 子函数

新建项目.zpj 后，新建 Basic 文件，设置自动运行任务号，在程序编辑界面添加如下程序：

```
1
2  global dim num
3  end
4
5  global sub redraw() '绘图函数，自定义虚拟键功能
6  if num=20 THEN
7  print 1
8  elseif num=21 THEN
9  table(10)=100
10 elseif num=22 THEN
11 function1()
12 endif
13 end sub
14
15 global sub function1()
16 print num
17 end sub
18
19 global sub refresh() '刷新函数
20 if VKEY_SCAN<>0 THEN
21 num=VKEY_SCAN
22 SET_REDRAW
23 endif
24 end sub
--
```

Basic 程序：

```
global dim num
end

global sub redraw() '绘图函数，自定义虚拟键功能
  if num=20 THEN
    print 1
  elseif num=21 THEN
    table(10)=100
  elseif num=22 THEN
    function1()
  
```

```

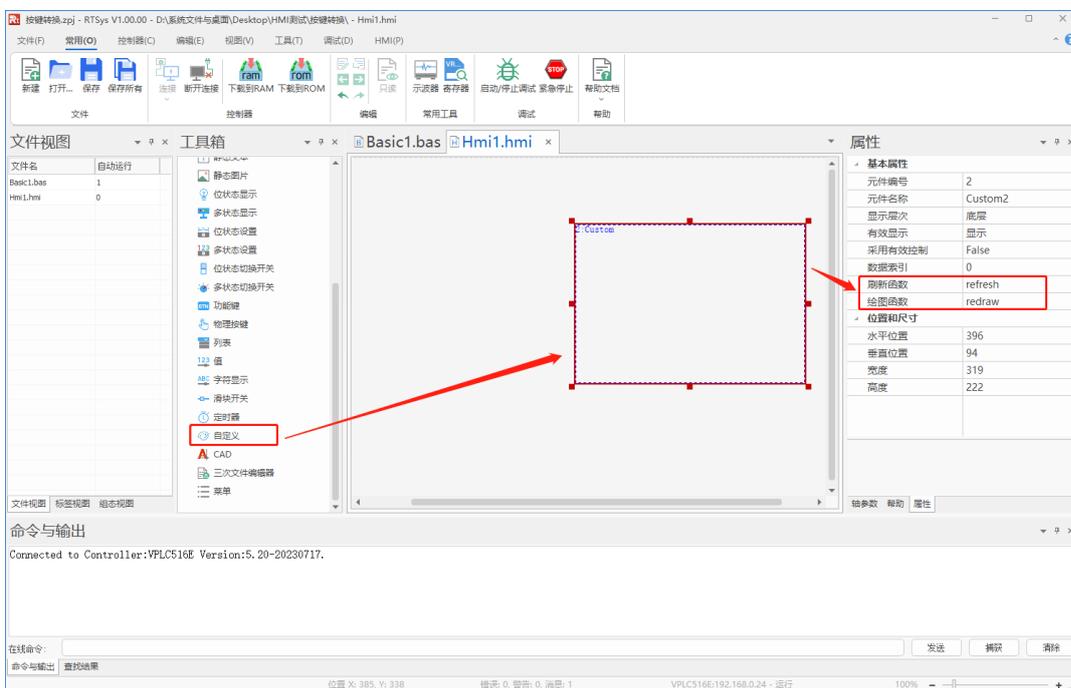
endif
end sub

global sub function1()
    print num
end sub

global sub refresh() '刷新函数
    if VKEY_SCAN<>0 THEN
        num=VKEY_SCAN
        SET_REDRAW
    endif
end sub
    
```

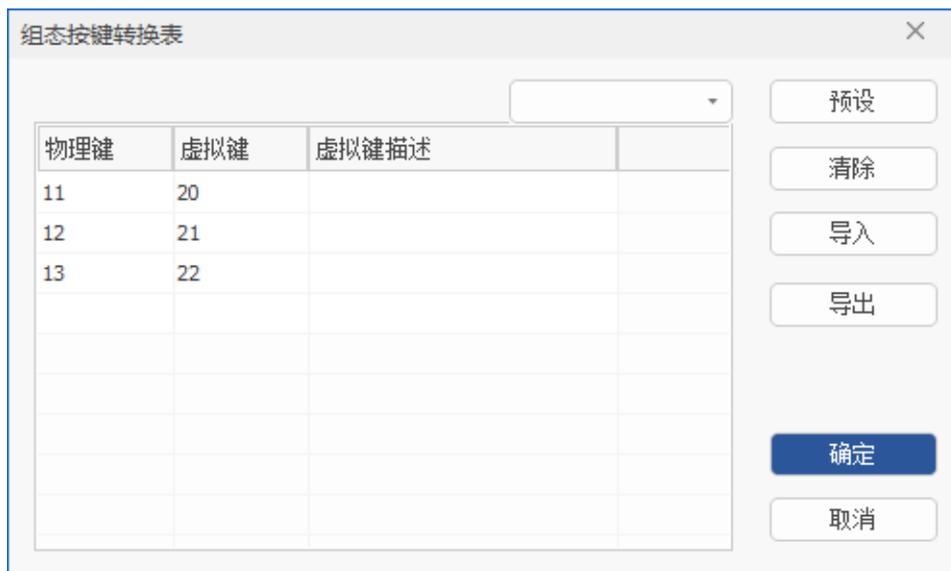
(二) Hmi 组态界面

1. 新建 Hmi 文件，设置自动运行任务号，打开 Hmi 窗口。
2. 添加一个“自定义元件”，调整其大小置于合适位置，在“属性”中的刷新函数(refresh)和绘图函数(redraw)分别调用对应的 Basic 子函数。



3. 打开菜单栏“按键转换”，按下图将虚拟键 20、21、22 分别绑定物理键 11、12、13，点击“确定”。

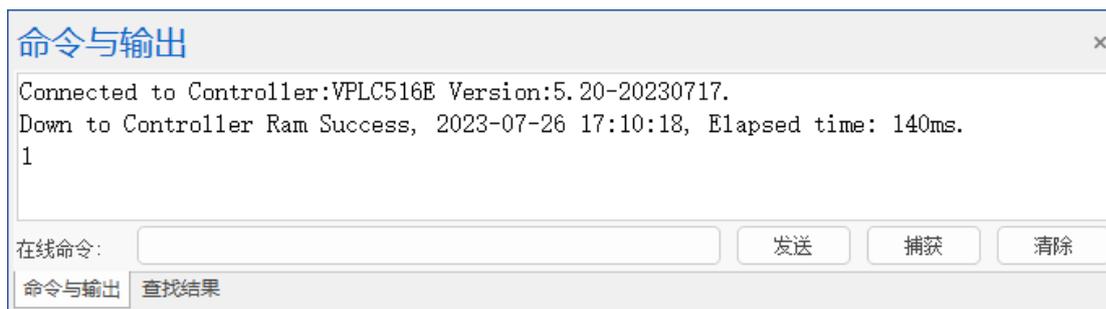
注：物理键键值需从《ZHD400X 示教盒用户手册》获取。若使用第三方外部设备，则物理按键键值需从对应设备说明书获取。



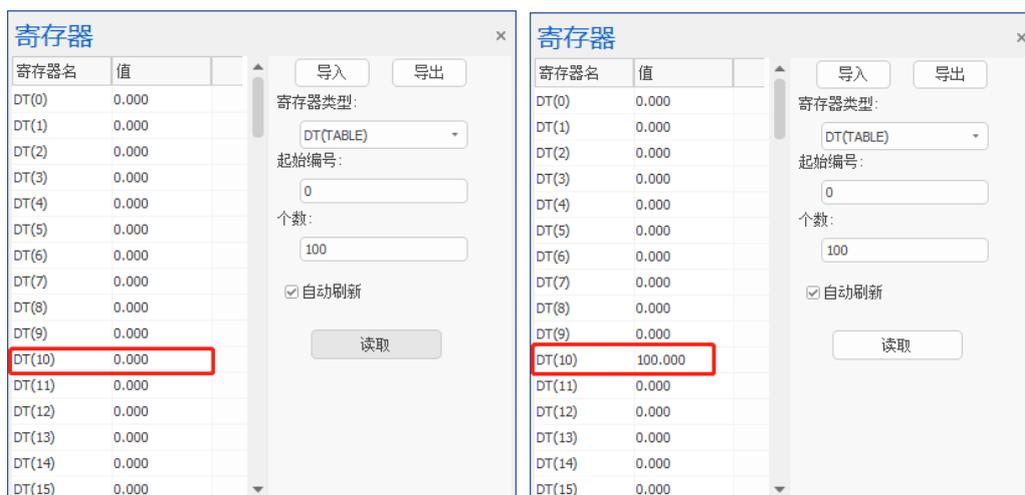
【运行效果】：

连接好控制器和示教盒，将程序下载到控制器和示教盒中。

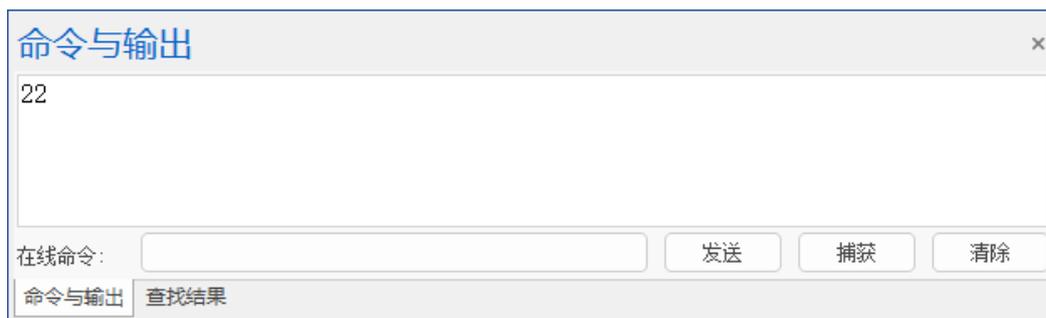
- 物理键 11 按下时，命令行打印 1。



- 物理键 12 按下时，TABLE(10)赋值为 100。



- 物理键 13 按下时，调用函数 function1，函数功能可以自定义（此处以打印 num 值为例）。



8.3. 动态列表

此例程为列表的使用示例，参考例程如下：

【实现目标】：使用 HMI_LISTTEXTS 指令配合“列表”元件实现动态列表控制效果。即通过一个列表的选择控制另一个列表的表项显示，进而可省略由于选项众多而建立多个列表。

【设计思路】：

- 在 Basic 文件中使用 HMI_LISTTEXTS 指令编写程序并定义全局 SUB 子函数。
- 添加多个“列表”元件，并将其绑定各个寄存器。（建议不同列表绑定不同的寄存器，以免产生影响冲突）
- 将第一个“列表”元件调用 Basic 子函数，实现动态控制。

【操作流程】：

(一) Basic 程序定义变量及构建 SUB 子函数

新建项目.zpj 后，新建 Basic 文件，设置自动运行任务号，在程序编辑界面添加如下程序：

Basic 程序界面：

```

1 global dim g_CurSelItem '定义列表选项类型，0-列表项、1-数字、2-选项、3-英文
2 global dim g_ShowStr1(128) '定义字符串数组
3
4
5 '''列表切换
6 全局子函数 sub_SwitchList0
7
8 HMI_LISTTEXTS(10,1,"列表项\n数字\n选项\n英文") '对控件1设置4个选项的列表文本
9
10
11 if g_CurSelItem = 0 then '选择类型0-列表项
12 HMI_LISTTEXTS(10,2,"列表项1\n列表项2\n列表项3\n列表项4\n列表项5\n列表项6\n列表项7\n列表项8\n列表项9\n列表项10\n列表项11\n列表项12") '对控件2设置列表文本
13
14 HMI_LISTTEXTS(10,3,"列表项1\n列表项2\n列表项3\n列表项4\n列表项5\n列表项6\n列表项7\n列表项8\n列表项9\n列表项10\n列表项11\n列表项12") '对控件10设置列表文本
15
16 HMI_LISTTEXTS(10,4,"列表项1\n列表项2\n列表项3\n列表项4\n列表项5\n列表项6\n列表项7\n列表项8\n列表项9\n列表项10\n列表项11\n列表项12") '对控件12设置列表文本
17
18 elseif g_CurSelItem = 1 then '选择类型1-数字
19
20 HMI_LISTTEXTS(10,2,"111111\n222222\n333333\n444444\n555555\n666666\n777777\n888888\n999999\n000000")
21
22 HMI_LISTTEXTS(10,3,"111111\n222222\n333333\n444444\n555555\n666666\n777777\n888888\n999999\n000000")
23
24 HMI_LISTTEXTS(10,4,"111111\n222222\n333333\n444444\n555555\n666666\n777777\n888888\n999999\n000000")
25
26
27 elseif g_CurSelItem = 2 then '选择类型2-选项
28
29 HMI_LISTTEXTS(10,2,"选项1\n选项2\n选项3\n选项4\n选项5\n选项6\n选项7\n选项8\n选项9")
30
31 HMI_LISTTEXTS(10,3,"选项1\n选项2\n选项3\n选项4\n选项5\n选项6\n选项7\n选项8\n选项9")
32
33 HMI_LISTTEXTS(10,4,"选项1\n选项2\n选项3\n选项4\n选项5\n选项6\n选项7\n选项8\n选项9")
34
35 else '选择类型3-英文
36
37 HMI_LISTTEXTS(10,2,"one\n two\n three\n four\n five\n six\n seven\n eight\n nine")
38
39 HMI_LISTTEXTS(10,3,"one\ntwo\nthree\nfour\nfive\nsix\nseven\neight\nnine")
40
41 HMI_LISTTEXTS(10,4,"one\ntwo\nthree\nfour\nfive\nsix\nseven\neight\nnine")
42
43 endif
44
45 g_ShowStr1 = "触发选中调用函数，选中表项" & g_CurSelItem+1 '控件3显示寄存器存储的字符串
46
47
48 endsub
49

```

Basic 程序:

```

global dim g_CurSelItem    '定义列表选项类型, 0-列表项、1-数字、2-选项、3-英文
global dim g_SHowStr1(128) '定义字符数组

"列表切换
global sub sub_SwitchList()

    HMI_LISTTEXTS(10,1,"列表项\n 数字\n 选项\n 英文")    '对控件 1 设置 4 个选项的列表文本

    if g_CurSelItem = 0 then    '选择类型 0-列表项

        HMI_LISTTEXTS(10,2, "列表项 1\n 列表项 2\n 列表项 3\n 列表项 4\n 列表项 5\n 列表项 6\n 列表项 7\n 列表项 8\n 列表项 9\n 列表项 10\n 列表项 11\n 列表项 12") '对控件 2 设置列表文本

        HMI_LISTTEXTS(10,3, "列表项 1\n 列表项 2\n 列表项 3\n 列表项 4\n 列表项 5\n 列表项 6\n 列表项 7\n 列表项 8\n 列表项 9\n 列表项 10\n 列表项 11\n 列表项 12") '对控件 3 设置列表文本

        HMI_LISTTEXTS(10,4, "列表项 1\n 列表项 2\n 列表项 3\n 列表项 4\n 列表项 5\n 列表项 6\n 列表项 7\n 列表项 8\n 列表项 9\n 列表项 10\n 列表项 11\n 列表项 12") '对控件 4 设置列表文本

    elseif g_CurSelItem = 1 then    '选择类型 1-数字

        HMI_LISTTEXTS(10,2,
"111111\n222222\n333333\n444444\n555555\n666666\n777777\n888888\n999999\n000000")

        HMI_LISTTEXTS(10,3,
"111111\n222222\n333333\n444444\n555555\n666666\n777777\n888888\n999999\n000000")

        HMI_LISTTEXTS(10,4,
"111111\n222222\n333333\n444444\n555555\n666666\n777777\n888888\n999999\n000000")

    elseif g_CurSelItem = 2 then    '选择类型 2-选项

        HMI_LISTTEXTS(10,2, "选项 1\n 选项 2\n 选项 3\n 选项 4\n 选项 5\n 选项 6\n 选项 7\n 选项 8\n 选项 9")

        HMI_LISTTEXTS(10,3, "选项 1\n 选项 2\n 选项 3\n 选项 4\n 选项 5\n 选项 6\n 选项 7\n 选项 8\n

```

```

选项 9")

    HMI_LISTTEXTS(10,4, "选项 1\n 选项 2\n 选项 3\n 选项 4\n 选项 5\n 选项 6\n 选项 7\n 选项 8\n
选项 9")

else    '选择类型 3-英文

    HMI_LISTTEXTS(10,2, "one\ntwo\nthree\nfour\nfive\nsix\nseven\neight\nnine")

    HMI_LISTTEXTS(10,3, "one\ntwo\nthree\nfour\nfive\nsix\nseven\neight\nnine")

    HMI_LISTTEXTS(10,4, "one\ntwo\nthree\nfour\nfive\nsix\nseven\neight\nnine")
endif

g_SHowStr1 = "触发选中调用函数，选中表项" + g_CurSelItem + 1    '控件 5 显示寄存器存储的字符
串

endsub
    
```

(二) Hmi 组态页面

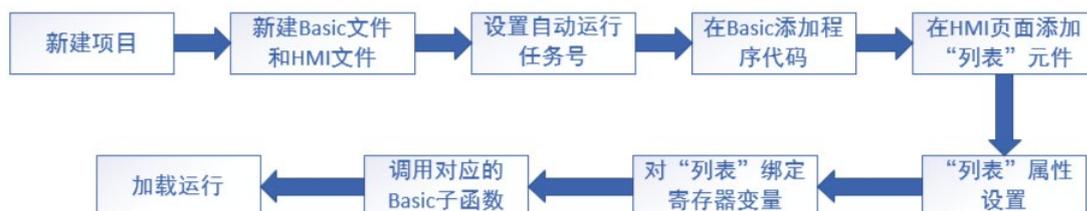
1. 新建一个 Hmi 文件，设置自动运行任务号。

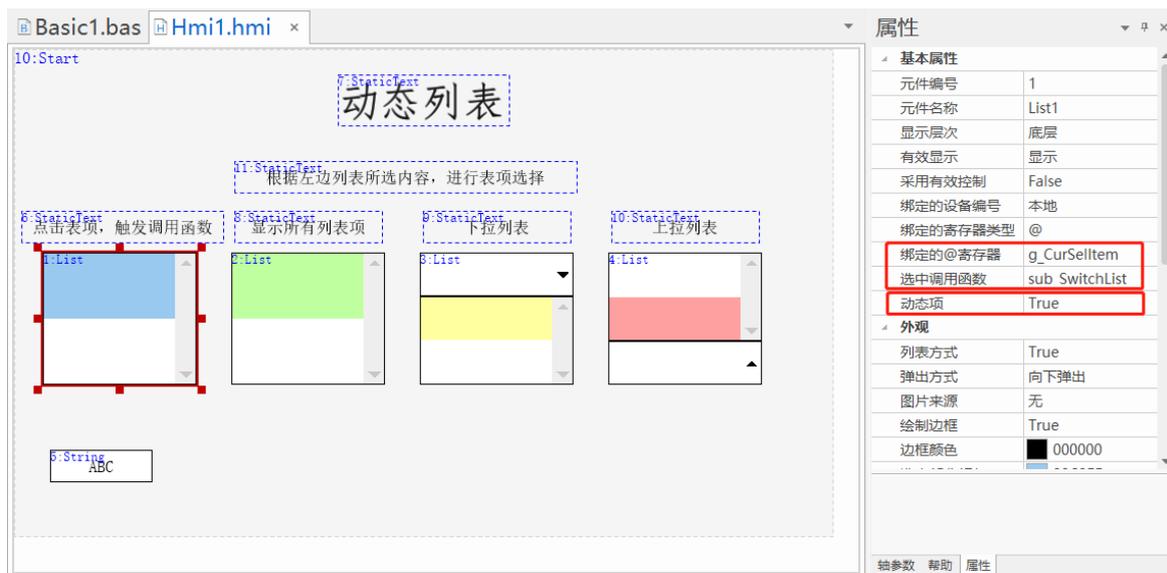
2. 打开 Hmi 窗口，选择“列表”元件，放置 4 个列表（列表数量视具体需求而定），对每个列表的属性进行设置，例如调整“列表方式”：True——显示全部（注意调整元件高度），Flase——下拉/上拉列表。可通过“弹出方式”调整向上弹出或向下弹出。

3. 对第一个列表选择绑定的寄存器变量，并调用 Basic 子函数实现列表的动态控制，另外三个列表绑定的寄存器变量编号保证不相同即可。

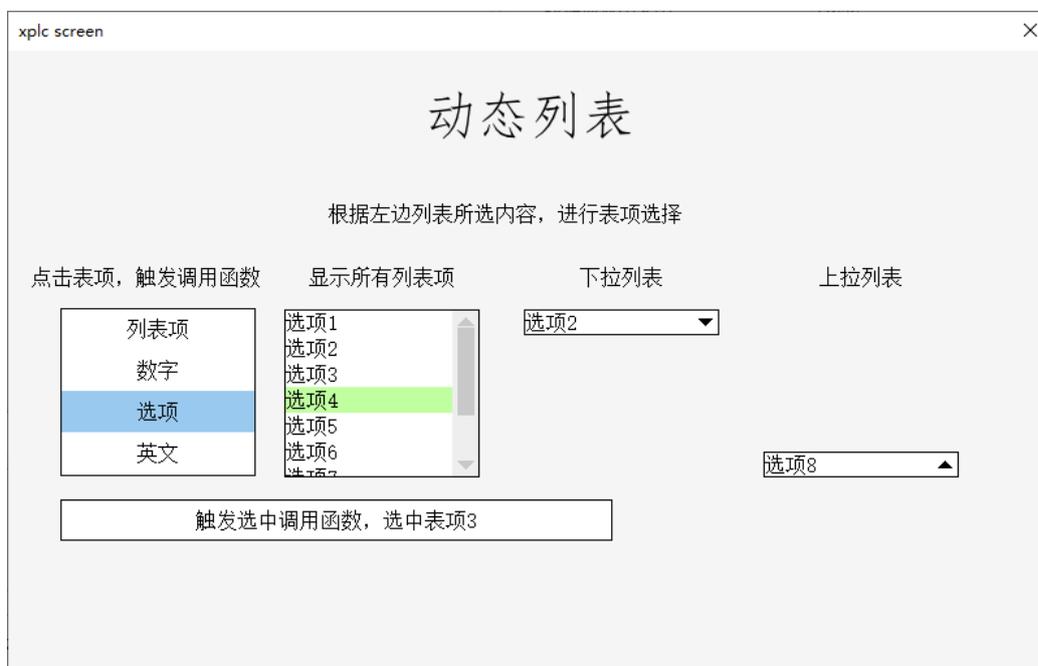
4. 最后可根据需求建立“文本”元件添加说明。

注意：使用 HMI_LISTTEXTS 指令，“列表”元件需将“属性”中的“动态项”选择为 True。





【运行效果】：



8.4. 视图缩放

此例程为滑块的使用示例，参考例程如下：

【实现目标】：通过“滑块开关”元件实现“自定义”元件中图形的大小缩放效果。（以矩形为例）

【设计思路】：

➤ 通过编写 Basic 程序定义矩形变量，以及刷新函数和绘图函数。

（一）定义参数变量，设置初始值。

1. 定义滑块的全局缩放变量（g_zoom），设置初始值。
2. 定义绘制图形的坐标数组，对各组 XY 坐标赋初始值。

(二) 定义全局刷新函数

给“自定义元件”调用刷新函数（refresh）。

(三) 定义全局绘图函数

该例程主要以倍数关系对图形实现缩小和放大。

1. 设置局部变量参数(zoom)和 g_zoom 得到其倍数关系。
2. 设置绘制图形的局部变量 XY 坐标，根据缩放量与 XY 坐标数组的关系，得到最终的 XY 坐标。
3. 写入需要绘制的图形指令。

➤ 添加“自定义”元件，并调用 Basic 定义的刷新函数和绘图函数

➤ 添加“滑块”元件并绑定寄存器（g_zoom）

【操作流程】：

(一) Basic 程序定义变量及构建 SUB 子函数

新建项目.zpj 后，新建 Basic 文件，设置自动运行任务号，在程序编辑界面添加如下程序：

Basic 程序界面：

```

1
2 GLOBAL DIM g_zoom '缩放
3 GLOBAL DIM g_pointx(2) 'x坐标集合
4 GLOBAL DIM g_pointy(2) 'y坐标集合
5 GLOBAL DIM g_round '倒角半径
6
7 '参数初始化
8 g_zoom=100
9 g_pointx(0)=20
10 g_pointy(0)=20
11 g_pointx(1)=100
12 g_pointy(1)=60
13 g_round=10
14
15 end
16
17 GLOBAL sub refresh() '刷新函数
18 SET_REDRAW
19 end sub
20
21 GLOBAL sub redraw() '绘图函数
22 LOCAL zoom,pointx,pointy
23 zoom=g_zoom/100.0
24 SETEX_LINE(2,0) '设置线条类型和线条宽度
25 SET_COLOR(255,0,0),RGB(255,255,0)
26 pointx=g_pointx(0)+g_pointx(1)*zoom
27 pointy=g_pointy(0)+g_pointy(1)*zoom
28
29 DRAWEX_RECT(g_pointx(0),g_pointy(0),pointx,pointy,g_round*zoom,1)
30 DRAWEX_RECT(g_pointx(0),g_pointy(0),pointx,pointy,g_round*zoom,0)
31 end sub
32
  
```

Basic 程序：

```
GLOBAL DIM g_zoom '缩放
```

```
GLOBAL DIM g_pointx(2)    'x 坐标集合
GLOBAL DIM g_pointy(2)    'y 坐标集合
GLOBAL DIM g_round        '倒角半径

'参数初始化
g_zoom=100
g_pointx(0)=20
g_pointy(0)=20
g_pointx(1)=100
g_pointy(1)=60
g_round=10

end

GLOBAL sub refresh()      '刷新函数
    SET_REDRAW
end sub

GLOBAL sub redraw()      '绘图函数
    LOCAL zoom,pointx,pointy
    zoom=g_zoom/100.0
    SETEX_LINE(2,0)      '设置线条类型和线条宽度
    SET_COLOR(RGB(255,0,0),RGB(255,255,0))
    pointx=g_pointx(0)+g_pointx(1)*zoom
    pointy=g_pointy(0)+g_pointy(1)*zoom

    DRAWEX_RECT(g_pointx(0),g_pointy(0),pointx,pointy,g_round*zoom,1)
    DRAWEX_RECT(g_pointx(0),g_pointy(0),pointx,pointy,g_round*zoom,0)
end sub
```

(二) Hmi 组态页面：

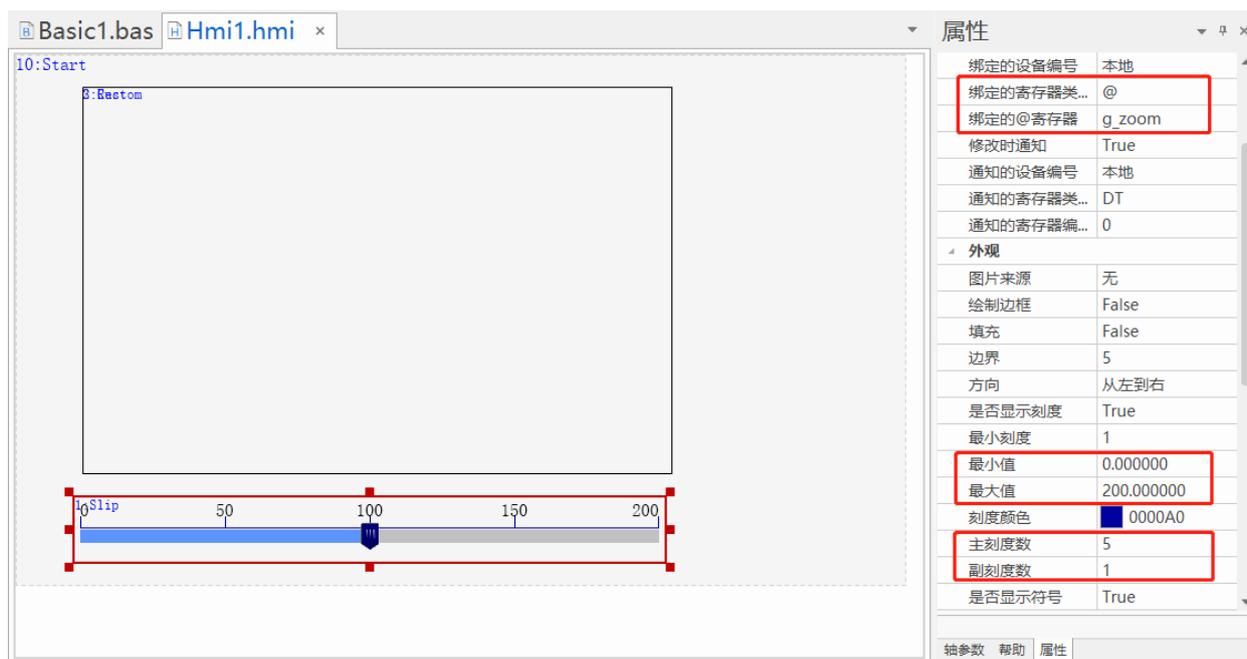
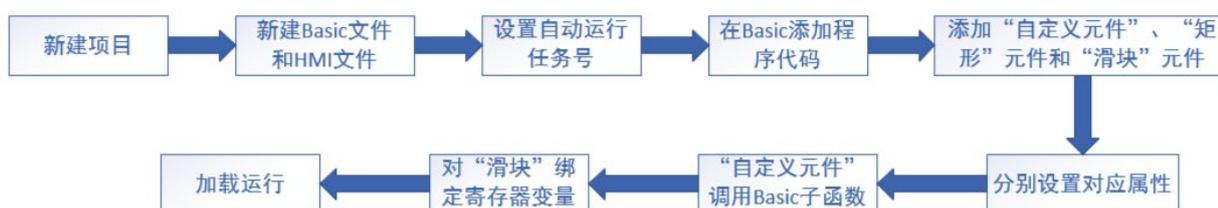
在自定义元件框中，采用指令进行矩形绘制，通过滑块可调节矩形放大或缩小。

1. 新建 Hmi 文件，设置自动运行任务号，打开 Hmi 窗口。

2. 添加一个“自定义元件”，调整其大小置于合适位置，在“属性”中的刷新函数和绘图函数分别调用对应的 Basic 子函数（refresh 和 redraw）。

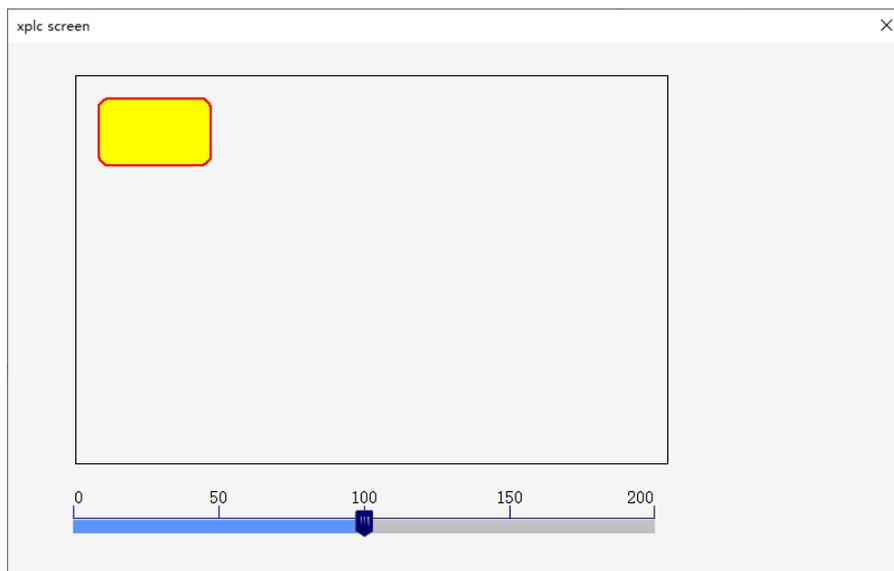
3. 绘制一个“矩形”元件，大小与“自定义元件”一致，移动到和“自定义元件”位置重合。

4. 绘制“滑块”元件，在“属性”中调整样式、大小，显示数值范围等，刻度数可通过“属性”的主刻度数和副刻度数进行设置（下图主刻度数为 5，副刻度数为 1），并放置于合适位置（建议放置在“自定义元件”外），同时绑定寄存器变量。如下图所示。

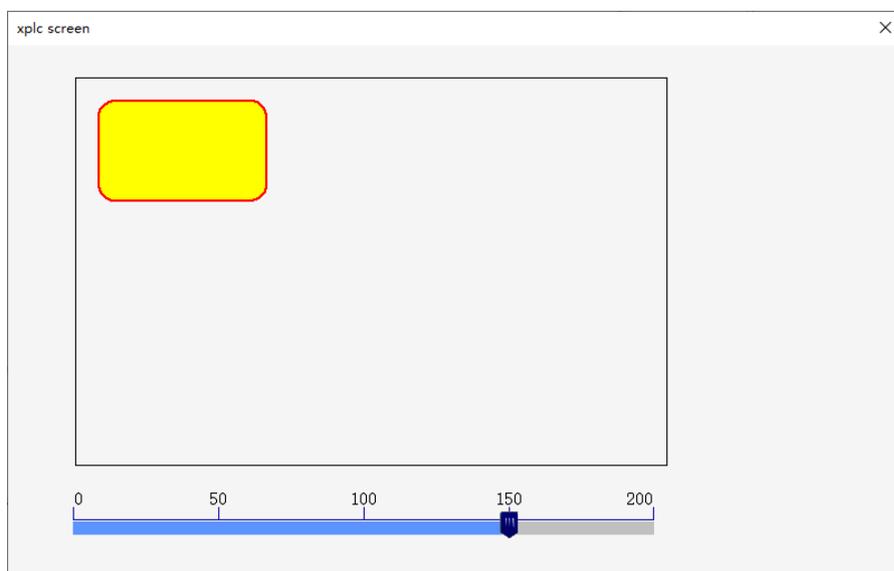


【运行效果】：

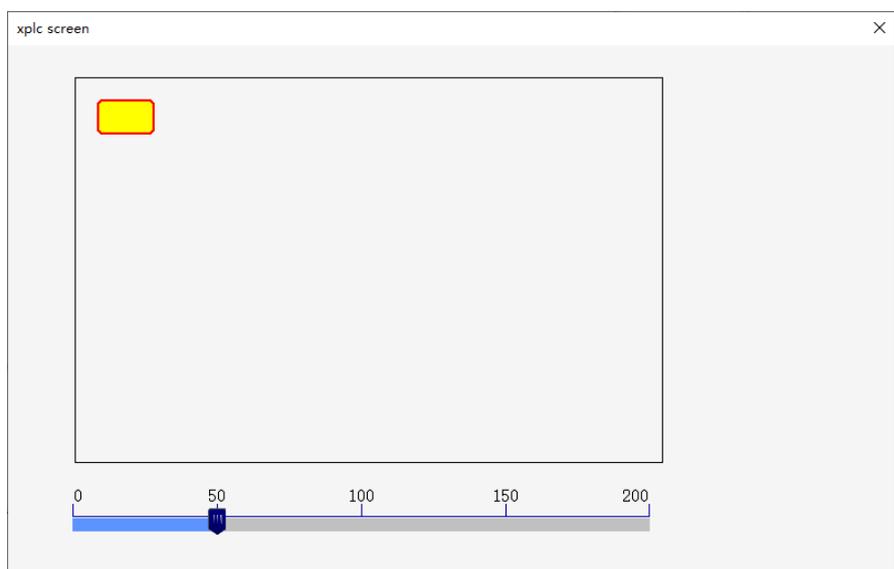
运行后，由于设置的初始值为 100，滑块默认处于中间位置，即 100 的刻度处，图形初始大小如图所示。



➤ 滑块拖动到 150 后，实现的放大效果如图所示。



➤ 滑块拖动到 50 后，实现的缩小效果如图所示。



8.5. 滚动条使用

此例程为滚动条的使用示例，参考例程如下：

【实现目标】：使用滚动条指令结合“自定义”元件和“值”元件实现自定义滚动条及行号刷新。

【设计思路】：

- 在 Basic 文件中使用滚动条指令编写程序并定义全局 SUB 子函数。

滚动条的设置

- 1、滚动条初始化：利用 [SCROLLBAR_INIT](#) 指令确定滚动条的编号和放置的位置，设置滚动条的宽度和高度、颜色，以及总行数和显示行数。
- 2、滚动条的刷新：利用 [SCROLLBAR_RELASH](#) 指令实现对对应编号滚动条的刷新，并对滚动条变换位置的时候实现刷新重绘。
- 3、滚动条的绘制：利用 [SCROLLBAR_DRAW](#) 指令绘制滚动条。

- 添加“自定义”元件，并分别调用刷新函数和绘图函数。
- 添加多个“值”元件，绑定寄存器变量，每行绑定的寄存器变量数值需加一。
- 添加“定时器”元件，实现定时刷新效果。

【应用场景关联】

在文件管理、写多行文本或自定义绘图视图缩放等场景时，都可添加滚动条使得有更多的管理空间。（本例子主要以多行文本的场景进行滚动条演示。可通过行号进行观察变化。）

【操作流程】：

(一) Basic 程序定义变量及构建 SUB 子函数

新建项目.zpj 后，新建 Basic 文件，设置自动运行任务号，在程序编辑界面添加如下程序：

```

Basic1.bas x Hmi1.hmi
1  global DIM Linenum          '首行行号
2  global DIM ScrollValue     '滚动条滚动值
3  global DIM maxlines       '滚动条总行数
4  global DIM showlines      '滚动条显示行数
5
6  maxlines=50 '总行数为50
7  showlines=10 '显示行数为10
8  sub ScrollBar_Init() '滚动条初始化
9
10 end
11
12 '定时器刷新数据，每秒刷新，处理不需要频繁刷新的变量以及状态
13 global sub Timer_RefreshData()
14
15     dim timercnt
16
17     Linenum = ScrollValue+ 1 '行号随滚动值变化
18
19 end sub
20
21 global sub correct()
22     TOUCH_ADJUST()
23 end sub
24
25 '=====滚动条刷新=====
26 global sub sub_ScrollBar_reflash() '在自定义元件2中的刷新函数进行调用
27
28     local value, bIfRedraw '定义变量用于判断是否重绘
29
30     bIfRedraw = SCROLLBAR_REFLASH() '根据返回值手动判断是否重绘
31     if bIfRedraw > 0 then
32         value = SCROLLBAR_POS() '获取滚动条0的当前位置滚动值

```

Basic 程序:

```
global DIM Linenum          '首行行号
global DIM ScrollValue      '滚动条滚动值
global DIM maxlines        '滚动条总行数
global DIM showlines       '滚动条显示行数

maxlines=50  '总行数为 50
showlines=10 '显示行数为 10
sub_ScrollBar_Init()  '滚动条初始化

end

'定时器刷新数据，每秒刷新，处理不需要频繁刷新的变量以及状态
global sub Timer_RefreshData()

    dim timercnt

    Linenum = ScrollValue+ 1  '行号随滚动值变化

end sub

global sub correct()
    TOUCH_ADJUST()
end sub

'=====滚动条刷新=====
global sub sub_ScrollBar_reflash()  '在自定义元件 2 中的刷新函数进行调用

    local value, bIfRedraw  '定义变量用于判断是否重绘

    bIfRedraw = SCROLLBAR_REFLASH(0)  '根据返回值手动判断是否重绘
    if bIfRedraw > 0 then
        value = SCROLLBAR_POS(0)      '获取滚动条 0 的当前位置滚动值
        if ScrollValue <> value then  '判断当前的滚动值是否一致
            ScrollValue = value
        endif
    end if
end sub
```

```
        SET_REDRAW      '重新绘制
    endif

end sub

'=====滚动条绘制=====
global sub sub_ScrollBar_redraw()      '在自定义元件 2 中的绘图函数进行调用

    SCROLLBAR_DRAW(0)      '绘制滚动条

end sub

'=====滚动条初始化=====
global sub sub_ScrollBar_Init()

    SCROLLBAR_INIT(0, 378, 0, 22, 346, maxlines, showlines)      '设置滚动条 0 的宽度为 22，高度为
    350，总行数为 50，显示行数为 10。注意：滚动条坐标位置不可超出自定义元件大小范围。

end sub
```

(二) Hmi 组态页面：

在自定义元件中绘制滚动条，并采用“值”元件来显示行号。使用“定时器”对行号进行刷新。

1. 新建 Hmi 文件和 Basic 文件，分别设置自动运行任务号，打开 Hmi 窗口；

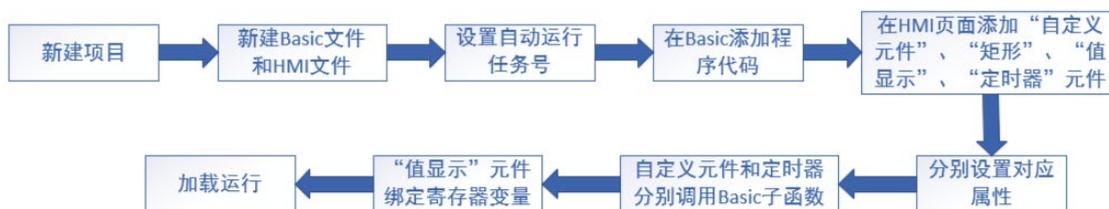
2. 添加一个“自定义”，调整其大小置于合适位置（本例程设置元件尺寸为：400*350），在“属性”中分别调用对应的刷新函数（例程中的：sub_ScrollBar_reflash()）和绘图函数（例程中的：sub_ScrollBar_redraw()）。

3. 添加一个“矩形”元件用作边框，调整其大小与“自定义元件”相同，放置与“自定义元件”重合的位置。（目的是便于确定无边框自定义元件的位置）

4. 添加多个“值”元件，数量根据实际情况而定（本例程添加 10 个），调整其大小并放置于合适位置。在“属性”中将各个“值”元件绑定到对应寄存器变量，各“值显示”元件绑定的寄存器变量逐个往下加 1（如第一个元件的寄存器变量+0，第二个元件的寄存器变量+1，第三个寄存器变量+2，以此类推），再根据需求调整其他属性内容。如下图。

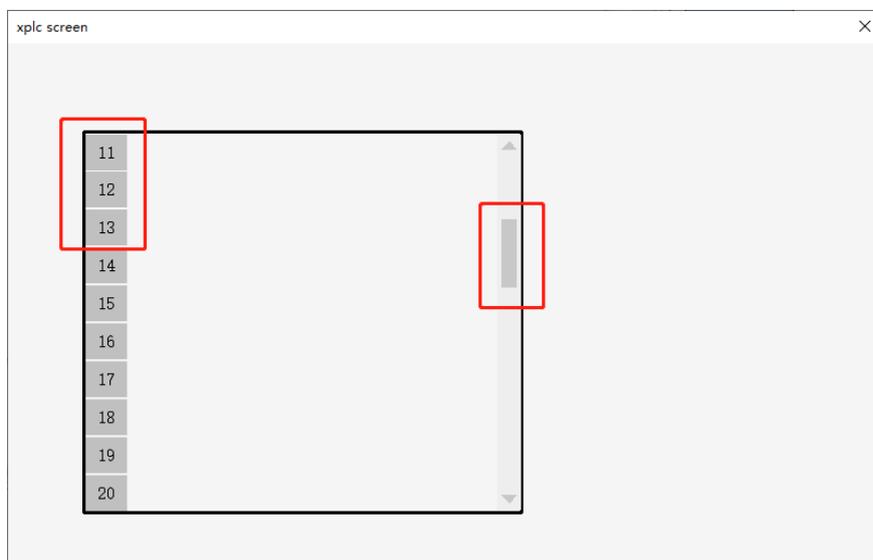
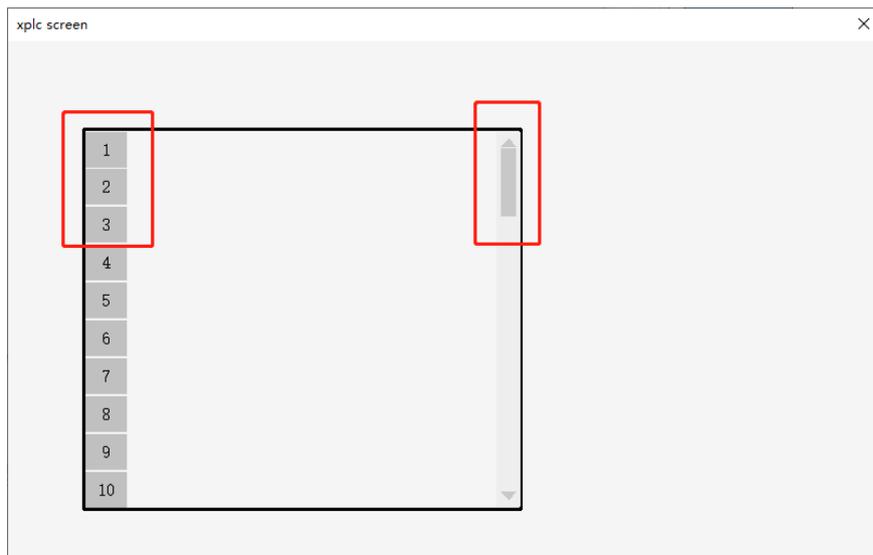


5. 添加一个“定时器”元件，放置于合适位置，适当调整“属性”中的“时间 ms”，值越小定时器的触发周期越短。在“属性”的“动作”中选择 call sub，在“动作函数名”处调用 Basic 定时器刷新子函数。



【运行效果】：

随着滚动条的滚动，行号会发生变化。



8.6. CAD 功能

本章节仅介绍元件的简单使用例程，CAD、File3 等完整功能 Demo 可联系正运动技术工程师获取。

8.6.1. CAD 导入矢量图片

此例程为使用 CAD 元件操控矢量图片的使用示例。

【实现目标】：使用 CAD 指令结合“CAD”元件和“功能键”元件实现矢量图片插入和关闭的效果。

【设计思路】：

- 在 Basic 文件中使用 CAD_LOADFILE 和 CAD_CLOSE 指令等编写程序并定义全局 SUB 子函数。

(一) 定义导入 CAD 图像函数

“功能键”元件一调用动作函数，函数体中使用 CAD_LOADFILE 指令导入 CAD 图像。

(二) 定义关闭 CAD 图像函数

“功能键”元件二调用动作函数，函数体中使用 CAD_CLOSE 指令关闭 CAD 图像。

- 添加“CAD”元件，设置通道号、图层颜色等。
- 添加“功能键”元件并在属性“动作”中调用对应函数。

【操作流程】：

(一) Basic 程序定义变量及构建 SUB 子函数

新建项目.zpj 后，新建 Basic 文件，设置自动运行任务号，在程序编辑界面添加如下程序：



语法	<p>CAD_LOADFILE(dxfname [, refdis[, layers]])</p> <p>dxfname = CAD_LOADFILE ()</p> <p>dxfname: dxf文件名，带路径</p> <p>refdis: 导入复杂曲线分割精度，范围[0.01-0.5]</p> <p style="color: red;">该分割精度决定导入复杂曲线时是否自动分割为线段，refdis缺省或=0时，不自动分割曲线，反之则根据refdis精度进行曲线分割</p> <p>layers: 选择加载图层，bit0-bit31位有效，支持最大32个图层</p> <p style="color: red;">缺省加载所有图层（加载G代码文件时无效）</p>
----	--

Basic 程序:

```

‘打开矢量图形
global sub Btn_LoadCadFile()

dim loadfile(128)
loadfile = “CADTest/1.dxf”          ‘保存矢量图片的路径
?”加载图形”,loadfile
CAD_LOADFILE(loadfile,0)

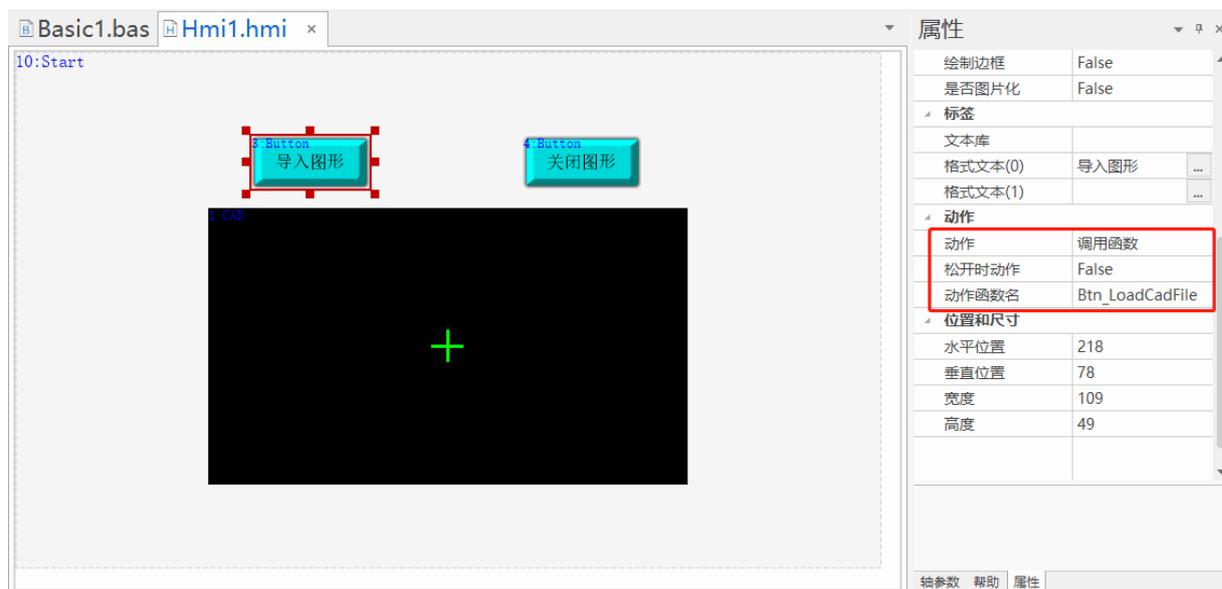
end sub
    
```

```

‘关闭矢量图形
global sub CloseCad()
    CAD_CLOSE()
end sub
    
```

(二) Hmi 组态界面

1. 新建 Hmi 文件，设置 Hmi 文件自动运行任务号，打开 Hmi 窗口。
2. 在控制器或者仿真器的 flash 目录下，插入一个带有矢量图片的文件夹。
3. 添加一个“CAD”元件，调整其大小并置于合适位置。
4. 在 Basic 文件中使用 CAD_LOADFILE 编写插入矢量图片功能的代码，设置矢量图片带路径的文件名、导入复杂曲线分割精度以及选择加载图层。
5. 在 Hmi 文件中添加一个“功能键”元件，调整其大小置于合适位置。格式文本设置为“导入图形”，动作设置为“调用函数”，动作函数名选择第四步中设计的函数名。
6. 在 Basic 文件中使用 CAD_CLOSE 编写关闭矢量图片功能的代码。
7. 在 Hmi 文件中添加一个“功能键”元件，调整其大小置于合适位置。格式文本设置为“关闭图形”，动作设置为“调用函数”，动作函数名选择第六步中设计的函数名。



【运行效果】：



- 点击“导入图形”功能键后，CAD 元件区域显示导入的 CAD 矢量图片。此时将鼠标停至 CAD 元件区域使用滚轮该 CAD 图形会相应的缩放，点击 CAD 区域画面拖动时 CAD 图形位置会改变，此时画面偏移。



8.6.2.CAD 结合三次文件使用

此例程为 CAD 元件结合三次文件元件的使用示例。

【实现目标】：在上节例程的基础上将 CAD 图形文件的 Basic 代码和 G 代码导入三次文件编辑器元件中，并在界面上展示。

【设计思路】：

- 在 Basic 文件中使用 CAD_TOCODE 和 FILE3_BUFLOAD 指令等编写程序并定义全局 SUB 子函数。

(一) 定义导出 CAD 图像 Basic 代码的 SUB 子函数

1. 定义数组，用于存储图片路径。
2. 函数体中使用 CAD_TOCODE 指令，CAD_TOCODE 中 type=0 为导出 Basic 类型代码。
3. 使用 FILE3_BUFLOAD 指令将三次文件加载出来。

(二) 定义导出 CAD 图像 G 代码的 SUB 子函数

1. 定义数组，用于存储图片路径。
2. 函数体中使用 CAD_TOCODE 指令，CAD_TOCODE 中 type=1 为导出 G 代码。
3. 使用 FILE3_BUFLOAD 指令将三次文件加载出来。

- 添加“三次文件编辑器”元件，设置通道号、可编辑等属性。

- 添加“功能键”元件并在属性“动作”中调用对应函数。

【操作流程】：

(一) Basic 程序定义变量及构建 SUB 子函数

新建项目.zpj 后，新建 Basic 文件，在程序编辑界面添加如下程序：

```

1  global sub Btn_LoadCadFile()
2
3
4      dim loadfile(128)
5      loadfile="CADTest/1.nc"
6      ?"加载图片",loadfile
7      CAD_LOADFILE(loadfile,0,1)
8
9  end sub
10
11 GLOBAL SUB closecad()
12     CAD_CLOSE()
13 end sub
14
15 '导出G代码
16 global sub Btn_ExportGCode()
17     dim FileName(128)
18
19     '' 文件名称
20     FileName = "CADTest/1.nc"
21
22     CAD_TOCODE(FileName, 1, -1)          '导出代码文件 (NC/Z3P)
23     ?"导出G代码"
24
25     FILE3_BUFLOAD(0, FileName)
26
27 endsub
28
29 '导出Basic
30 global sub Btn_ExportBasic()
31     dim FileName(128)
32
33     '' 文件名称
34     FileName = "CADTest/1.Z3P"          '导出代码文件 (NC/Z3P)

```

语法	<p>FILE3_BUFLOAD(mode, filename)</p> <p>mode: 加载模式, 0-仅加载已存在的文件, 1-新建空白文件再加载 2-清空文件再加载 (若文件不存在, 等同模式1)</p> <p>filename: 3 次文件名</p>
----	--

语法	<p>CAD_TOCODE(filename, type, layers)</p> <p>filename: 导出代码文件名 xxx.Z3P输出basic的Z3P文件 xxx.nc输出G代码的NC文件 当filename=""时, 直接输出到当前3次文件加载区 通过FILE3_CHANNEL指令指定输出到哪个三次文件通道</p> <p>type: 导出代码类型, 0-Z3P (Basic), 1-NC (G代码)</p> <p>layers: 导出图层, bit0~bit31位有效, -1表示导出所有图层 如layers=1+2+4+8, 表示导出图层0、1、2、3</p>
----	--

Basic 程序:

```
'加载图形文件
global sub Btn_LoadCadFile()

    dim loadfile(128)
    loadfile = "CADTest/1.dxf"
    ?"加载图形",loadfile

    CAD_LOADFILE(loadfile, 0,1) '从文件中载入图片

end sub

'关闭图形
global sub CloseCad()

    CAD_CLOSE()          '关闭 CAD 加载图形

end sub

global sub Btn_ExportGCode()

    dim FileName(128)
```

```

"文件名称
FileName = "CADTest/1.nc"

CAD_TOCODE(FileName, 1, -1)      '导出代码文件（NC）
?"导出 G 代码 "

FILE3_BUFLOAD(0, FileName)      '加载指定 3 次文件到内存中,0 表示仅加载已存在的文件

endsub

'导出 Basic
global sub Btn_ExportBasic()
  dim FileName(128)
  "文件名称
  FileName = "CADTest/1.Z3P"

  CAD_TOCODE(FileName, 0, -1)    '导出代码文件（Z3P）
  ?"导出 Basic"

  FILE3_BUFLOAD(0, FileName)    '加载指定 3 次文件到内存中,0 表示仅加载已存在的文件
endsub

```

(二) Hmi 组态页面

1. 在上一例程——【CAD 导入矢量图片】程序的基础上添加一个“三次文件编辑器”元件，调整其大小并置于合适位置。“三次文件编辑器”元件属性中“垂直滚动条”和“水平滚动条”都需要设置为 True 才可滑动显示完整的 Basic 代码和 G 代码。

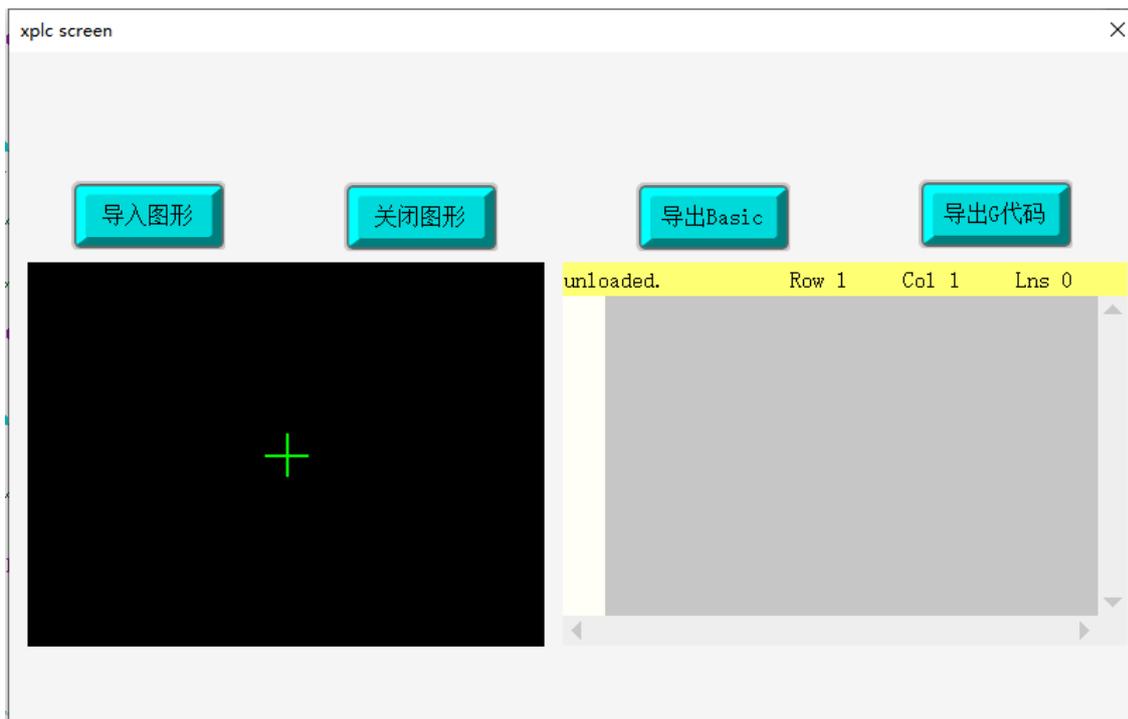
2. 在 Basic 文件中使用 CAD_TOCODE 指令编写导出 Basic 类型代码功能的程序，设置导出代码文件名、导出类型且 type=0 以及导出图层。

3. 在 Hmi 文件中添加一个“功能键”元件，调整其大小置于合适位置。格式文本设置为“导出 Basic”，动作设置为“调用函数”，动作函数名选择第二步中设计的函数名，此例程调用的是 Btn_ExportBasic 函数。

4. 在 Basic 文件中使用 CAD_TOCODE 指令编写导出 G 代码功能的程序，设置导出代码文件名、导出类型且 type=1 以及导出图层。

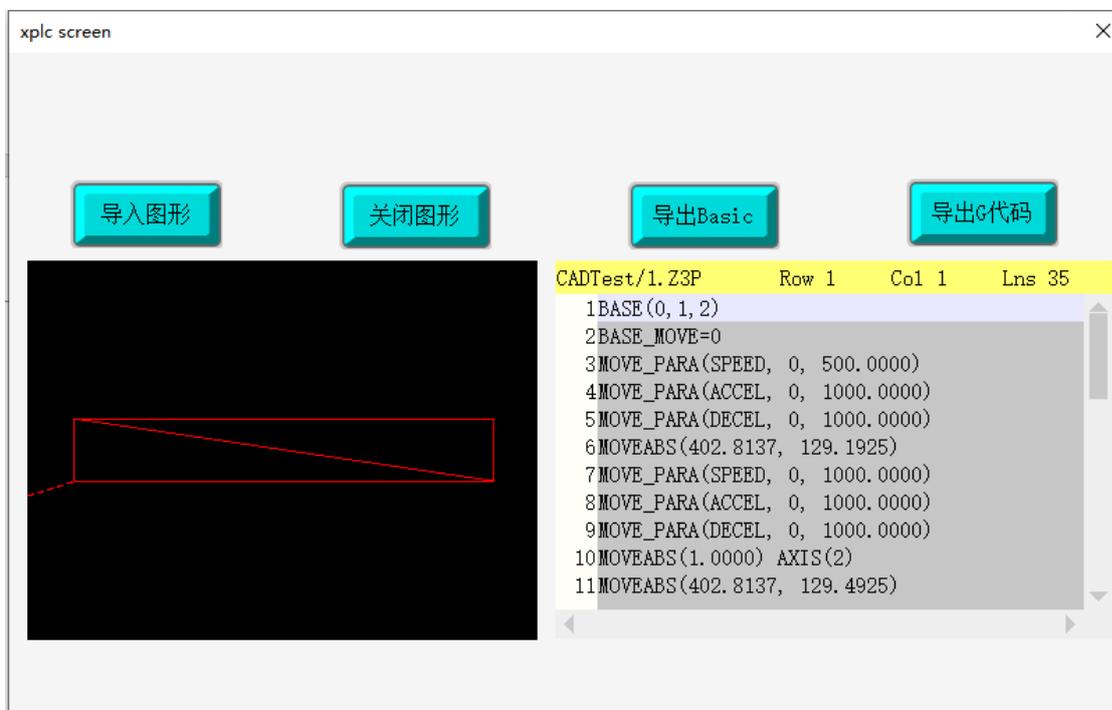
5. 在 Hmi 文件中添加一个“功能键”元件，调整其大小置于合适位置。格式文本设置为“导出 G 代码”，动作设置为“调用函数”，动作函数名选择第四步中设计的函数名，此例程调用的是 Btn_ExportGCode 函数。

【运行效果】：

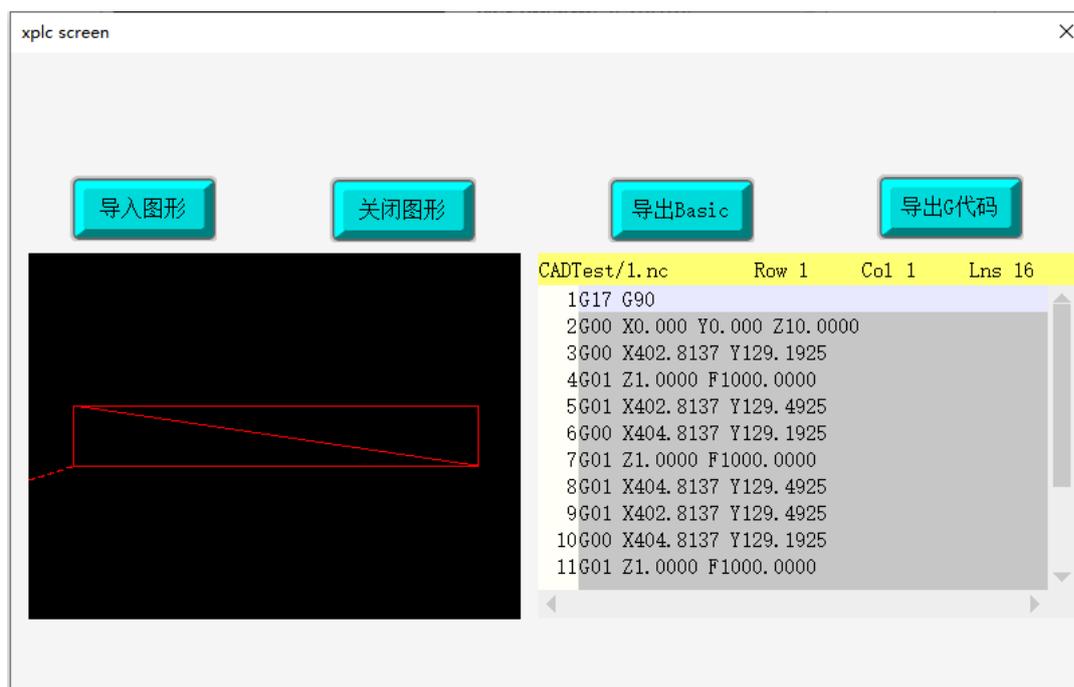


- 点击“导入图形”后，“CAD”元件显示对应矢量图片。再点击“导出 Basic”或“导出 G 代码”元件时三次文件编辑器元件中会显示相应的 Basic 代码和 G 代码。

导出 Basic 效果图：



导出 G 代码效果图:



8.6.3.CAD 结合自定义元件使用

此例程为 CAD 元件结合自定义元件使用的示例。

【实现目标】：使用 CAD_TOTABLE、CAD_LOADTABLE 和 CAD_DRAW 等指令结合“CAD 元件”实现在“自定义”元件中绘制出 CAD 图形。

【设计思路】：

- 在 Basic 文件中使用 CAD_LOADTABLE 和 CAD_DRAW 指令等编写程序并定义全局 SUB 子函数。

使用 CAD_LOADFILE 指令导入 CAD 图形，使用 CAD_TOTABLE 指令导出到 TABLE 后关闭 CAD，再使用 CAD_LOADTABLE 指令从 TABLE 中加载图形。

(一) 定义绘图函数

给“自定义”元件中的绘图函数调用。使用 CAD_DRAW 指令绘制 CAD 图形。

(二) 定义刷新函数

给“自定义”元件中的刷新函数调用。函数体中使用 CAD_SETCOLOR 等指令在自定义元件内刷新 CAD 图形颜色、显示方式等。

- 添加“自定义”元件，属性中“刷新函数”、“绘图函数”调用 Basic 对应的 SUB 函数。
- 添加“CAD”元件，并设置通道号、图层颜色等。

【操作流程】：

(一) Basic 程序定义变量及构建 SUB 子函数

新建项目.zpj 后，新建 Basic 文件，在程序编辑界面添加如下程序：

```

Basic1.bas x Hmi1.hmi
1
2 CAD_LOADFILE("CADTest/3D-玫瑰花.dxf", 0) '从文件中载入图片
3 CAD_TOTABLE(10000, -1) '导出TABLE数据，导出到table(10000)之后
4 CAD_CLOSE() '关闭CAD加载图形
5 CAD_LOADTABLE(10000) '从TABLE中载入图形，从TABLE(10000)开始加载图形
6
7 end
8
9 '' 绘图函数
10 GLOBAL SUB drawing1()
11     CAD_DRAW(0, 0, 1) '绘制CAD图形
12
13 END SUB
14
15 '' 刷新函数
16 GLOBAL SUB refresh1()
17
18     CAD_MOUSE(2) '设置鼠标模式
19     CAD_SETCOLOR(0, RGB(255, 0, 255)) '设置CAD绘制颜色
20     CAD_DISPLAY(3) '设置CAD显示方式
21
22 END SUB
23
    
```

语法	<p>CAD_TOTABLE(datatabid, layers)</p> <p>datatabid: table地址，输出table数据</p> <p>table存储数据格式如下： 详看TABLE数据格式设计小节。</p> <p>layers: 导出图层，bit0~bit31位有效，-1表示导出所有图层</p> <p>如layers=1+2+4+8，表示导出图层0、1、2、3</p>
语法	<p>CAD_LOADTABLE(tabid)</p> <p>tabid = CAD_LOADTABLE()</p> <p>tabid: table下标</p>
语法	<p>CAD_DRAW(offsetX, offsetY, fZoom [,plane, RotX, RotY, RotZ])</p> <p>offset1, offset2: 显示图形偏移，=(0,0)时，图形中心处于自定义控件中心位置</p> <p>fZoom: 显示缩放比例，1.0=自定义控件范围刚好完整显示加载图形</p> <p>以下属性为预留三维CAD图形：</p> <p>plane: 显示平面，缺省0-XY, 1-YZ, 2-ZX</p> <p>RotX, RotY, RotZ: 绕XYZ轴旋转角度，0~360，缺省0。</p>

语法	<p>CAD_SELECT (ix1, iy1, ix2, iy2, offsetX, offsetY, fZoom[,plane, RotX, RotY, RotZ])</p> <p>ix1,iy1,ix2,iy2: 坐标的, 选择框两个对顶点坐标 (两点成矩形)</p> <p>offsetX, offsetY: 显示图形偏移, =(0,0)时, 图形中心处于自定义控件中心位置</p> <p>fZoom: 显示缩放比例, 1.0=自定义控件范围刚好完整显示加载图形</p> <p>以下属性为预留三维CAD图形:</p> <p>plane: 显示平面, 缺省0-XY, 1-YZ, 2-ZX</p> <p>RotX, RotY, RotZ: 绕XYZ轴旋转角度, 0~360, 缺省0。</p>
----	--

Basic 程序:

```

CAD_LOADFILE("CADTest/3D-玫瑰花.dxf",0)    '从文件中载入图片
CAD_TOTABLE(10000, -1)                    '导出 TABLE 数据, 导出到 table(10000)之后
CAD_CLOSE()                               '关闭 CAD 加载图形
CAD_LOADTABLE(10000)                      '从 TABLE 中载入图形, 从 TABLE(10000)开始加载图形
end

"绘图函数
GLOBAL SUB drawing1()
    CAD_DRAW(0, 0, 1)                    '绘制 CAD 图形
END SUB

"刷新函数
GLOBAL SUB refresh1()
    CAD_MOUSE(2)                        '设置鼠标模式
    CAD_SETCOLOR(0,RGB(255,0,255))      '设置 CAD 绘制颜色
    CAD_DISPLAY(3)                      '设置 CAD 显示方式
END SUB

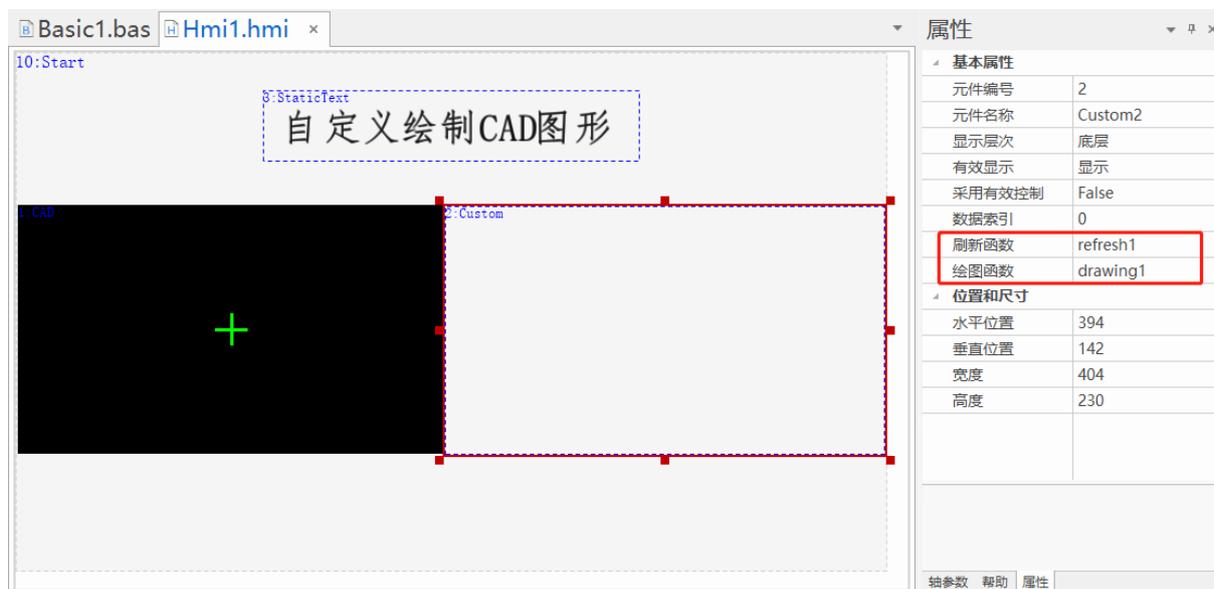
```

(二) Hmi 组态页面

1. 新建 Hmi 和 Basic 文件, 设置 Hmi 文件自动运行任务号, 打开 Hmi 窗口。
2. 在控制器或者仿真器的 flash 目录下, 插入一个带有矢量图片的文件夹。
3. 添加一个“CAD”元件, 调整其大小并置于合适的位置。添加一个“自定义”元件, 调整其大小并置于合适的位置。
4. 在 Basic 文件中编写绘图函数和刷新函数。本例程使用的绘图函数为 drawing1, 使用的刷新函数为

refresh1。

5. 在自定义元件中调用第四步中编写的绘图函数和刷新函数。



【运行效果】：



8.7. 文件浏览器使用

此例程为文件浏览器的使用示例，包含两个文件，Hmi 文件通过 Basic 文件打开，Basic 文件的其他函数由 Hmi 调用执行。该例程是多个元件的综合使用。

【实现目标】：使用 HMI_FILEFILTER、HMI_FILESEL、HMI_FILEPATH、HMI_CONTROLATTR 等指令结合“文件浏览器”元件、“列表”元件、“字符显示”元件、“功能键”元件、“三次文件编辑器”元件以及“树形图”等元件实现各种类型文件的显示、打开、选中与修改。

【设计思路】：

- 在 Basic 文件中使用 HMI_FILEFILTER、HMI_FILESEL、HMI_FILEPATH 指令等编写程序并定义全局 SUB 子函数。
- 使用“文件浏览器”元件调用 Basic 全局 SUB 函数实现文件类型的过滤与文件的选择。
- 使用“列表”元件绑定与文件类型选择相关的寄存器，设置“选中调用函数”以及需显示的“状态数量”，填入对应列表文本。
- 使用“字符显示”元件绑定与文件名显示相关的寄存器。
- 使用“功能键”元件调用 Basic 全局 SUB 函数打开对应文件。
- 使用 FILE3_BUFLOAD 指令将选择的三次文件加载出来。
- 添加“三次文件编辑器”元件，设置通道号、可编辑等属性。
- 添加“树形图”元件，设置绑定的寄存器类型及编号，接着在“列表文本”内设计树节点结构并编写其文本内容。根据树节点对应的 ID 编号在 Basic 文件中结合“HMI_CONTROLATTR”指令编写全局 SUB 函数，然后在树形图元件属性框中调用，实现三次文件编辑器“选中多行”的功能。

【操作流程】：

(一) Basic 程序启动 Hmi 程序、定义变量及构建 SUB 子函数

新建项目.zpj 后，新建 Basic 文件，设置自动运行任务号，在程序编辑界面添加如下程序：

```

1  '' 全局变量
2  global dim g_iCurSelFileFilter  '' 文件过滤器选择
3  global dim g_iCurSelFileName(128) '' 当前选择文件名
4
5  '' 启动HMI任务
6  RUN "Hmi1.hmi", 1
7  end
8
9
10 '' 文件浏览器 过滤器选择
11 global sub Sub_FileFiterSel()
12     if g_iCurSelFileFilter = 0 then  '' 显示NC文件
13         HMI_FILEFILTER(10, 1, "*.nc|*.cnc")
14     elseif g_iCurSelFileFilter = 1 then  '' 显示Bas文件
15         HMI_FILEFILTER(10, 1, "*.bas")
16     elseif g_iCurSelFileFilter = 2 then  '' 显示Z3P文件
17         HMI_FILEFILTER(10, 1, "*.z3p")
18     elseif g_iCurSelFileFilter = 3 then  '' 显示所有文件
19         HMI_FILEFILTER(10, 1, " *.*")
20     endif
21 end sub
22
23 '' 文件浏览器 单击选择文件
24 global sub Sub_FileSelect()
25     local strCurFileName(128)

```

语法	<pre>FILE3_BUFLOAD(mode, filename) mode: 加载模式, 0-仅加载已存在的文件, 1-新建空白文件再加载 2-清空文件再加载 (若文件不存在, 等同模式 1) filename: 3 次文件名</pre>
----	---

Basic 程序:

```
"全局变量
global dim g_iCurSelFileFilter    "文件过滤器选择
global dim g_iCurSelFileName(128) "当前选择文件名

"启动 HMI 任务
RUN "Hmi1.hmi", 1
end

"文件浏览器 过滤器选择
global sub Sub_FileFiterSel()
  if g_iCurSelFileFilter = 0 then  "显示 NC 文件
    HMI_FILEFILTER(10, 1, "*.nc*.cnc")
  elseif g_iCurSelFileFilter = 1 then  "显示 Bas 文件
    HMI_FILEFILTER(10, 1, "*.bas")
  elseif g_iCurSelFileFilter = 2 then  "显示 Z3P 文件
    HMI_FILEFILTER(10, 1, "*.z3p")
  elseif g_iCurSelFileFilter = 3 then  "显示所有文件
    HMI_FILEFILTER(10, 1, ".*.*")
  endif
end sub

"文件浏览器 单击选择文件
global sub Sub_FileSelect()
  local strCurFileName(128)
  strCurFileName = HMI_FILESEL(10, 1, 0)
  if table(0) = 0 then  "选中了文件
    g_iCurSelFileName = strCurFileName
  endif
end sub

"文件浏览器 双击文件
global sub Sub_FileDlbClick()
```

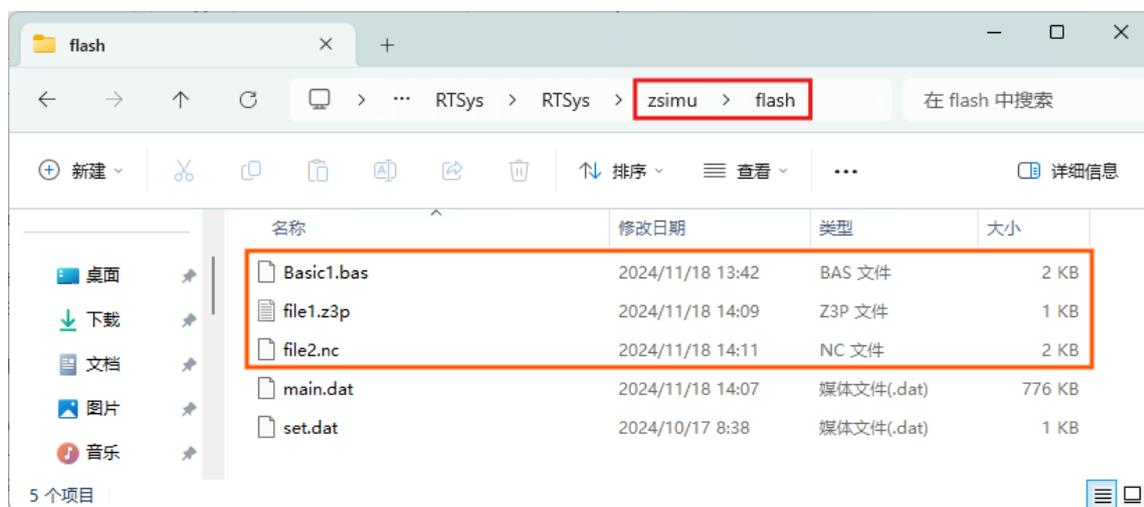
```
local strCurFileName(128)
strCurFileName = HMI_FILESEL(10, 1, 0)
if table(0) = 0 then    "选中了文件，直接打开
    g_iCurSelFileName = strCurFileName
    Sub_FileOpen()
endif
end sub

"文件浏览器 打开文件
global sub Sub_FileOpen()
    HMI_SHOWWINDOW(11,6)
    local strFilePathName(512)
    "根目录为“ C:/ ”时，不补“/ ”
    if strFilePathName(2) = 47 then
        strFilePathName = HMI_FILEPATH(10, 1) + g_iCurSelFileName
    else
        strFilePathName = HMI_FILEPATH(10, 1) + "/" + g_iCurSelFileName
    endif
    "打开三次文件
    FILE3_BUFLOAD(0, strFilePathName)
    ?"打开三次文件", strFilePathName
end sub

"树形图 结合“HMI_CONTROLATTR”指令实现三次文件编辑器“选中多行”功能
global sub Sub_Tree()
    if TABLE(1000) = 1001 THEN
        HMI_CONTROLATTR("MSELROWS",-5,11,1) "往上选中 5 行
    elseif TABLE(1000) = 1002 THEN
        HMI_CONTROLATTR("MSELROWS",-10,11,1) "往上选中 10 行
    elseif TABLE(1000) = 1004 THEN
        HMI_CONTROLATTR("MSELROWS",5,11,1) "往下选中 5 行
    elseif TABLE(1000) = 1005 THEN
        HMI_CONTROLATTR("MSELROWS",10,11,1) "往下选中 10 行
    elseif TABLE(1000) = 1006 THEN
        HMI_CONTROLATTR("MSELROWS",0,11,1) "选中当前行不变
    endif
end sub
```

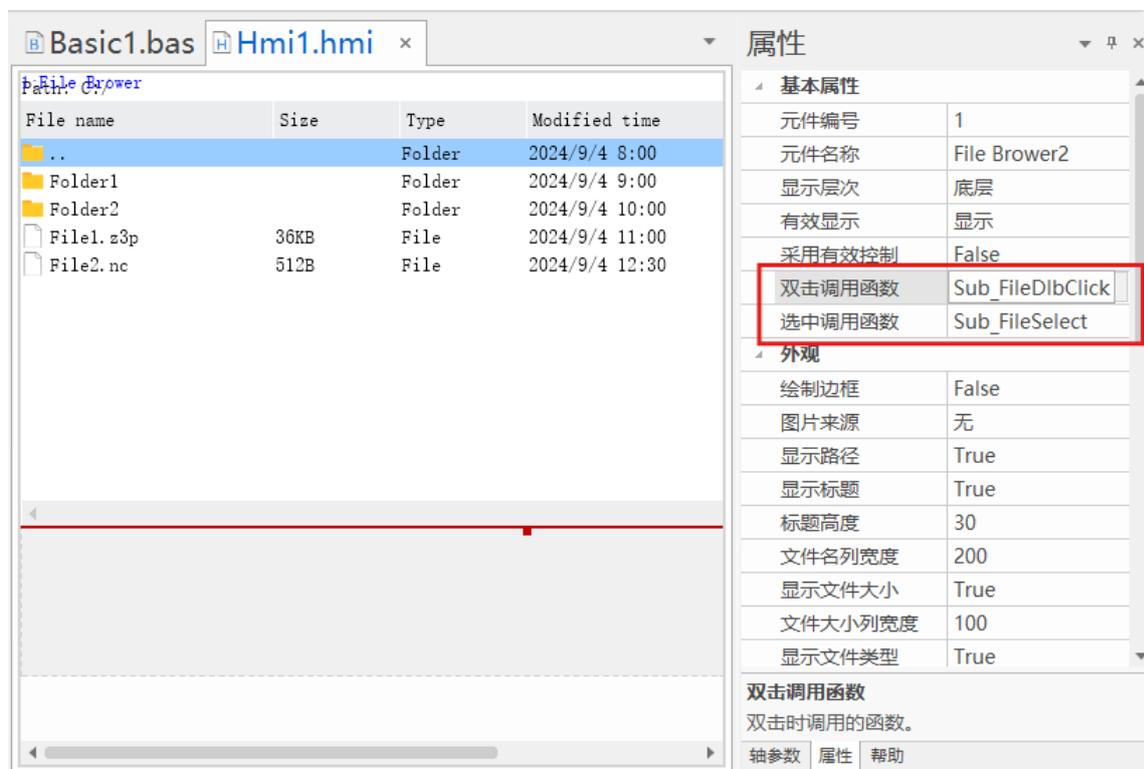
(二) Hmi 组态页面

1. 在控制器或者仿真器的 flash 目录下，插入一些 basic、z3p、nc 等类型的文件夹。



2. 新建 Hmi 文件，打开 Hmi 基本窗口 10。

3. 在基本窗口 10 添加一个“文件浏览器”元件，选择“双击调用函数”和“选中调用函数”，合理按需设置其他属性。



4. 在“文件浏览器”元件下方添加一个“静态文本”元件、一个“字符显示”元件、一个“列表”元件以及两个“功能键”元件，调整其大小并置于合适位置。

(1) 将“静态文本”元件的格式文本设为“文本名：”。

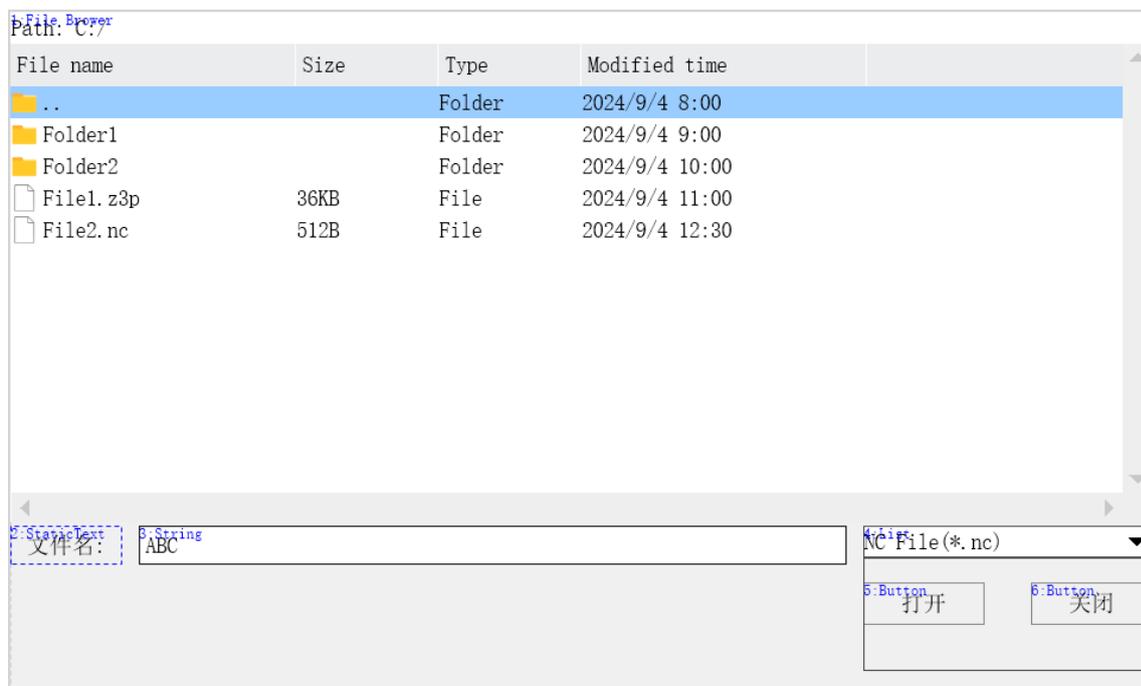
(2) “字符显示” 元件绑定 “g_iCurSelFileName” 寄存器，用于显示选中文件的名称。

(3) “列表” 元件选中调用函数 “Sub_FileFiterSel”，弹出方式设置为 “向下弹出”，绑定 “g_iCurSelFileFilter” 寄存器，状态数量设为 4，每个状态对应的列表文本设置如下：



(4) 一个 “功能键” 元件调用 “Sub_FileOpen” 函数，实现打开三次文件的功能；另一个 “功能键” 元件的动作选择 “关闭指定窗口”，实现关闭三次文件的功能。

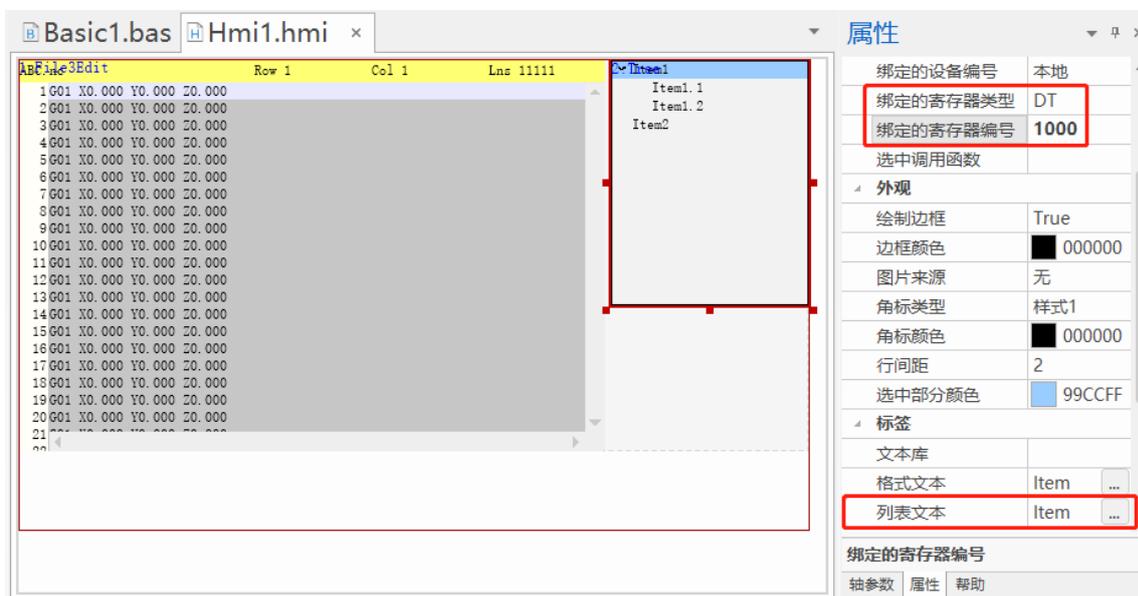
5. 基本窗口 10 整体界面如下图：



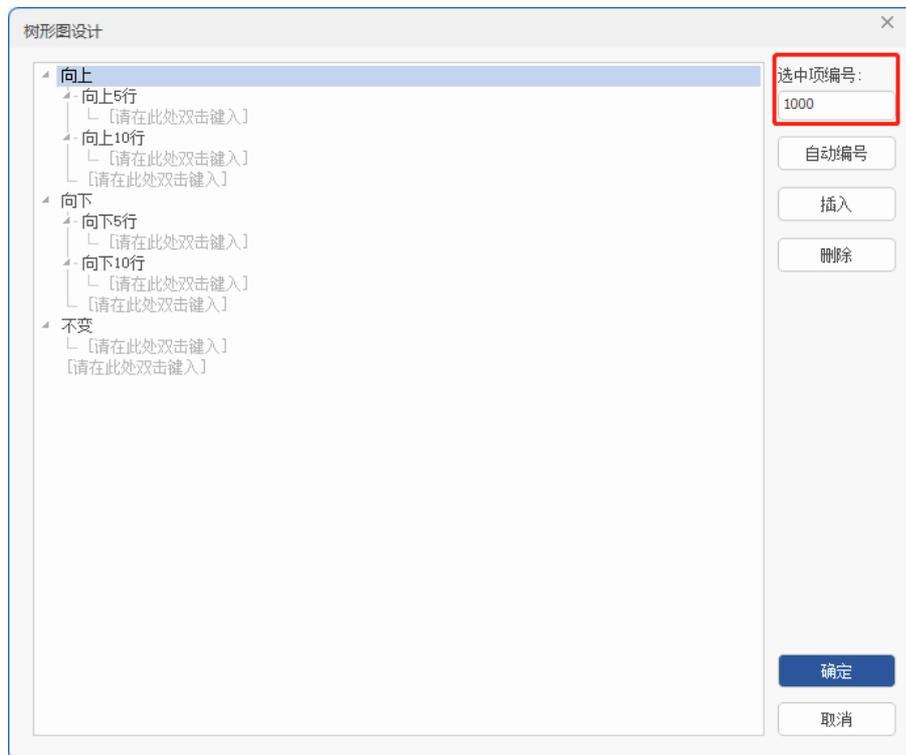
6. 新建弹出窗口 11，添加一个“三次文件编辑器”元件，设置其通道号、是否可编辑、外观等相关属性。使用 FILE3_BUFLOAD 指令加载指定三次文件到内存中显示。



7. 在 11 窗口添加一个“树形图”元件，设置绑定的寄存器类型及编号，接着在“列表文本”内设计树节点结构并编写其文本内容。



根据树节点对应的 ID 编号在 Basic 文件中结合 “HMI_CONTROLATTR” 指令编写 Sub_Tree 函数，然后在树形图元件属性框 “选中调用函数” 中调用该函数，实现三次文件编辑器 “选中多行” 的功能。

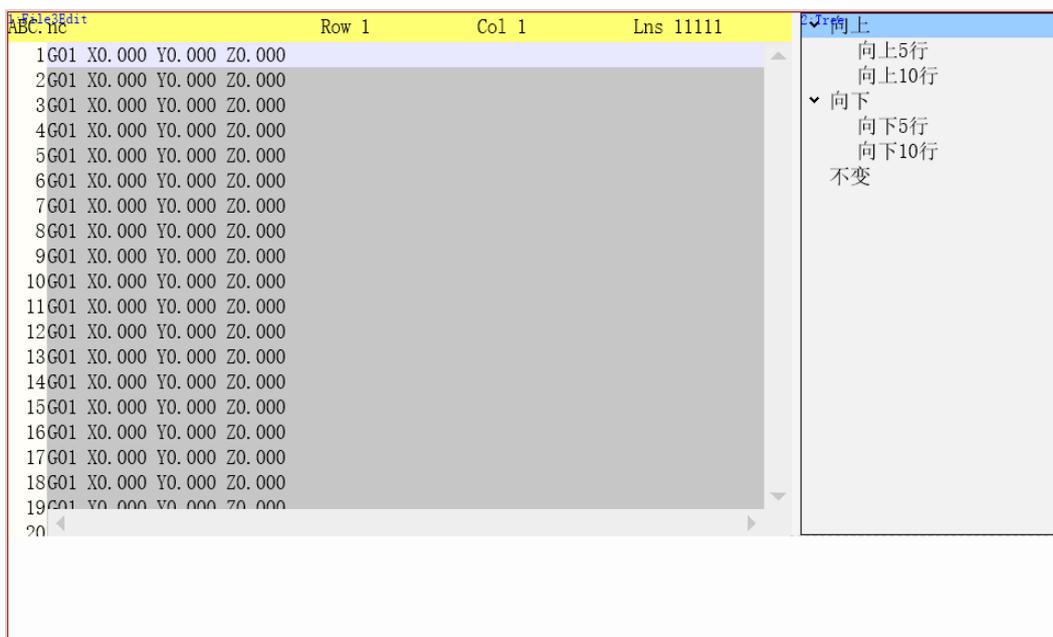


代码段如下图所示：

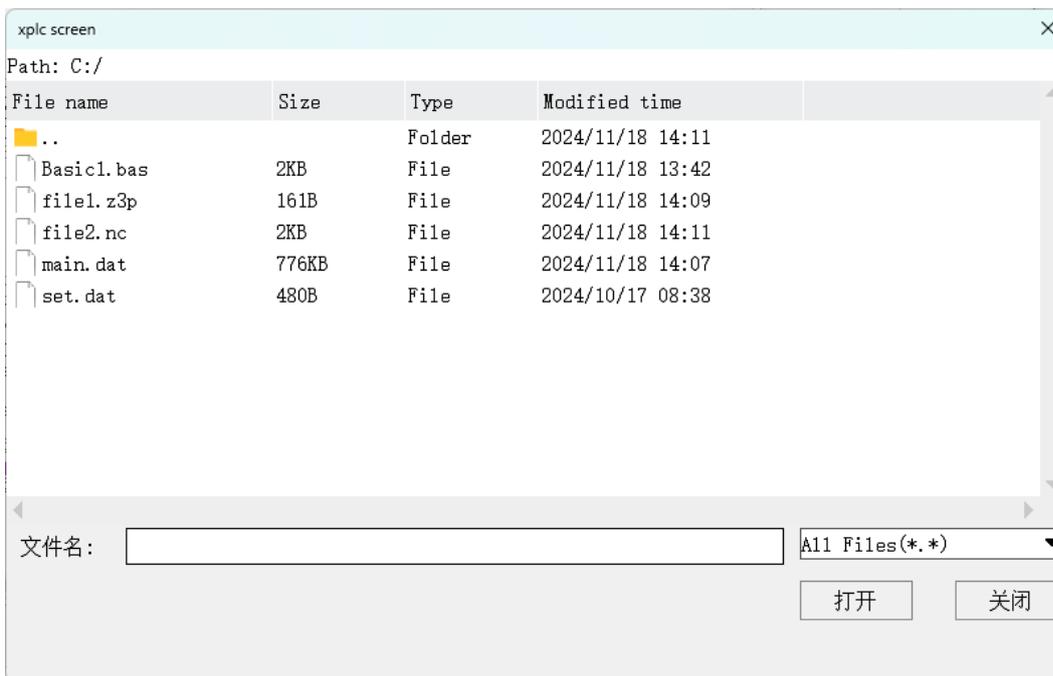
```

''' 树形图 结合 “HMI_CONTROLATTR” 指令实现三次文件编辑器 “选中多行” 功能
global sub Sub_Tree()
if TABLE(1000) = 1001 THEN
    HMI_CONTROLATTR("MSELROWS", -5, 11, 1) ''' 往上选中5行
elseif TABLE(1000) = 1002 THEN
    HMI_CONTROLATTR("MSELROWS", -10, 11, 1) ''' 往上选中10行
elseif TABLE(1000) = 1004 THEN
    HMI_CONTROLATTR("MSELROWS", 5, 11, 1) ''' 往下选中5行
elseif TABLE(1000) = 1005 THEN
    HMI_CONTROLATTR("MSELROWS", 10, 11, 1) ''' 往下选中10行
elseif TABLE(1000) = 1006 THEN
    HMI_CONTROLATTR("MSELROWS", 0, 11, 1) ''' 选中当前行不变
endif
end sub
    
```

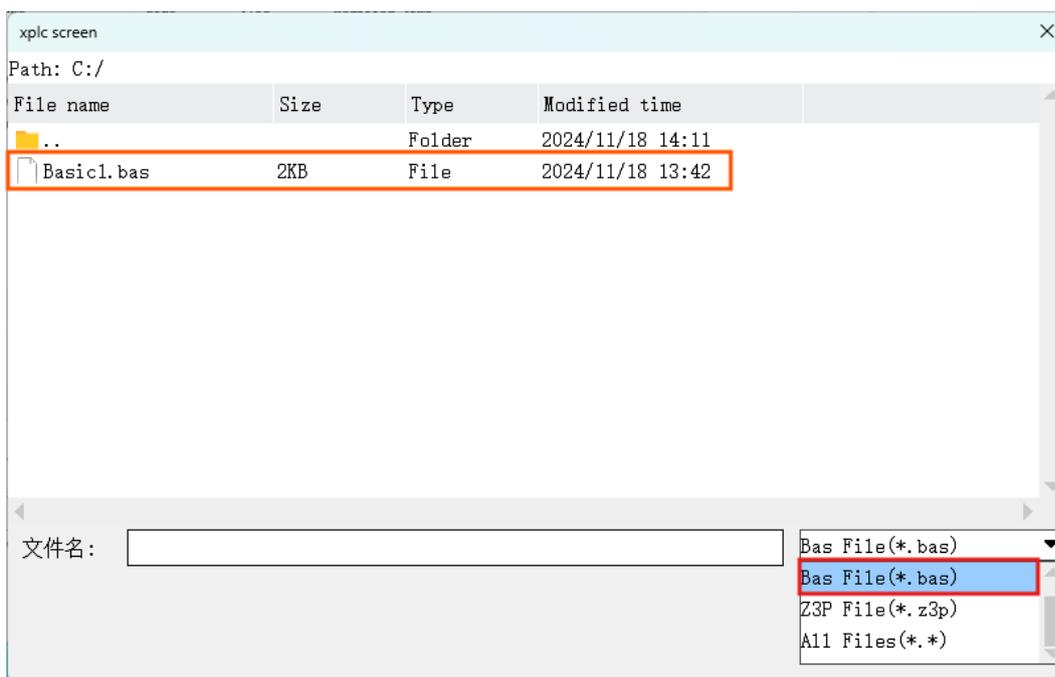
8. 基本窗口 11 整体界面如下图:



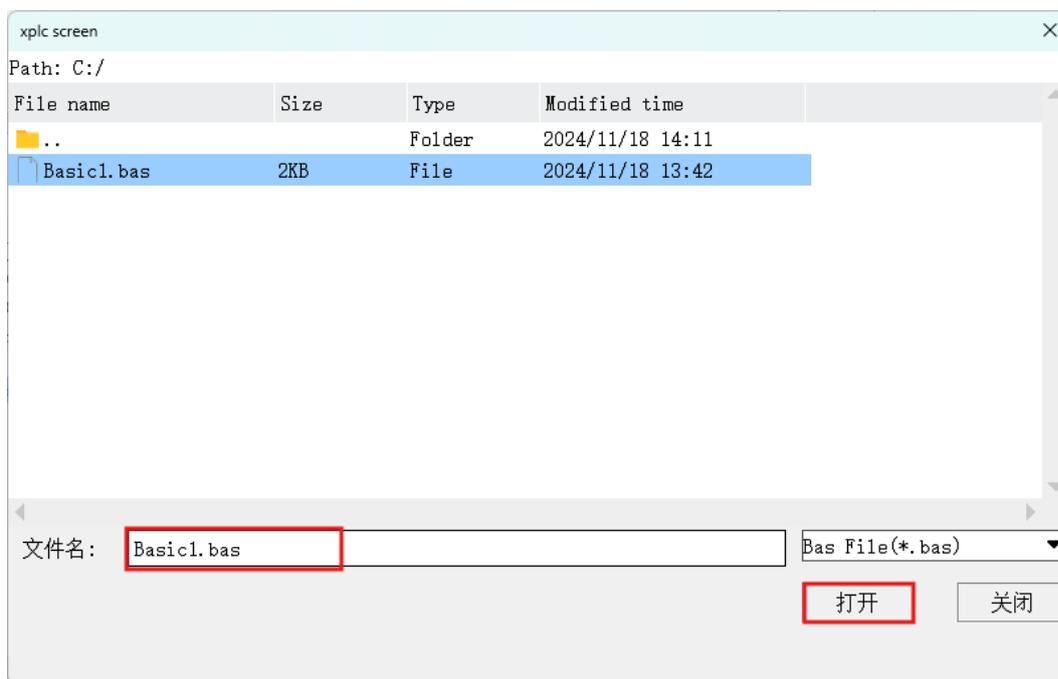
【运行效果】:



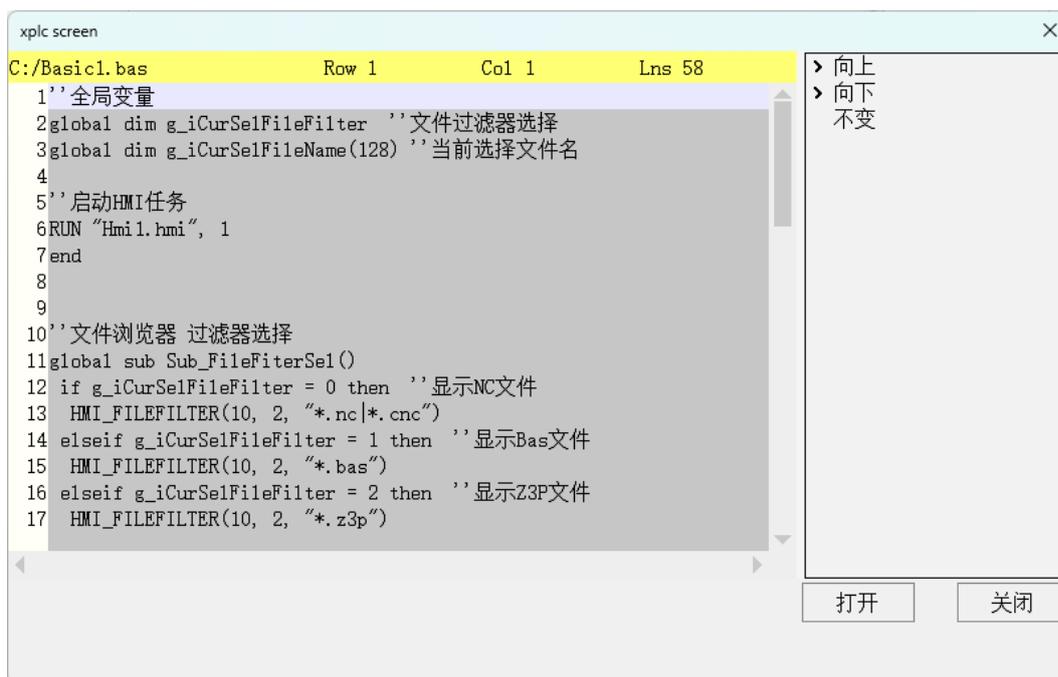
- 点击“列表”元件，选择需要显示的文件类型后，“文件浏览器”元件内显示相应文件类型的文件。



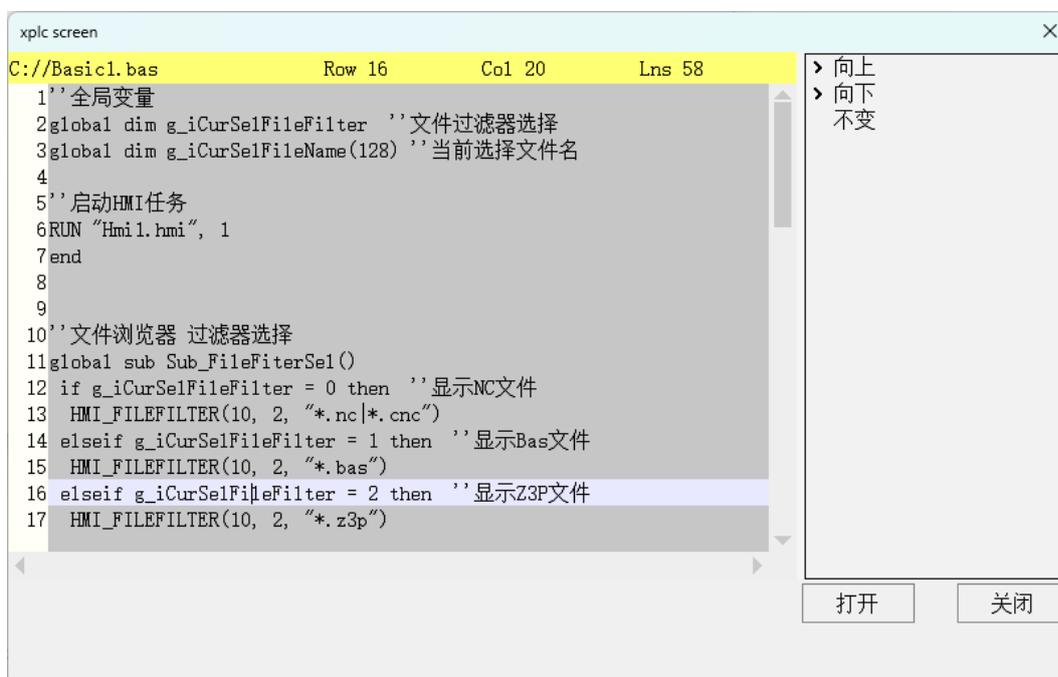
- 选中文件后，“字符显示”元件将显示该文件的文件名，点击“打开”或双击文件打开。

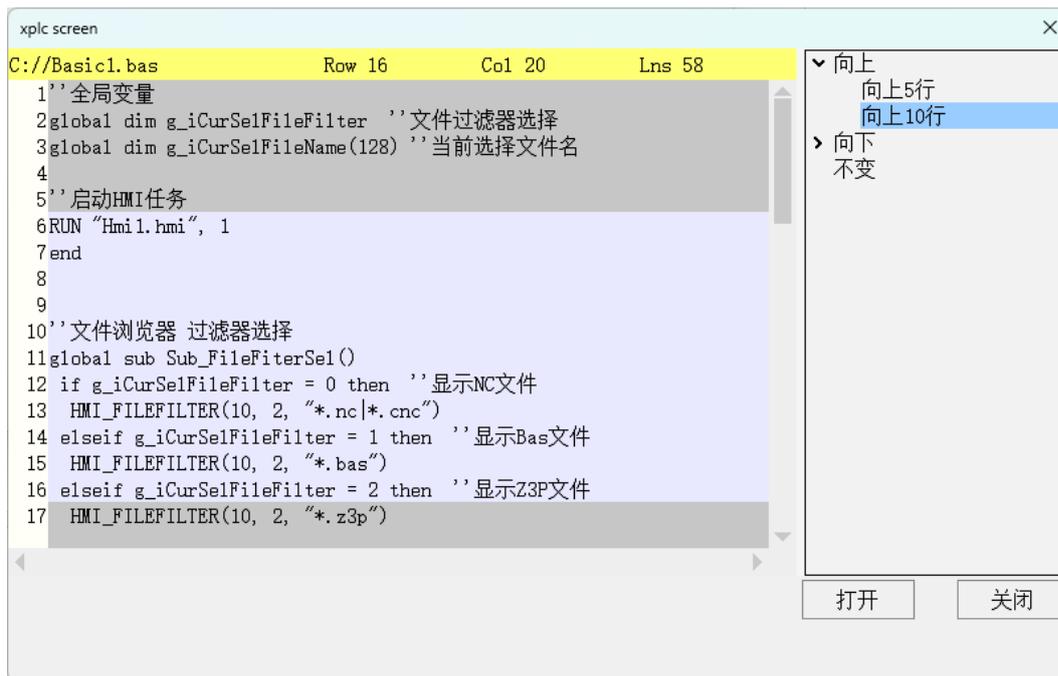


- 弹出“三次文件编辑器”界面，显示文件内容，可以对文本内容进行编辑。

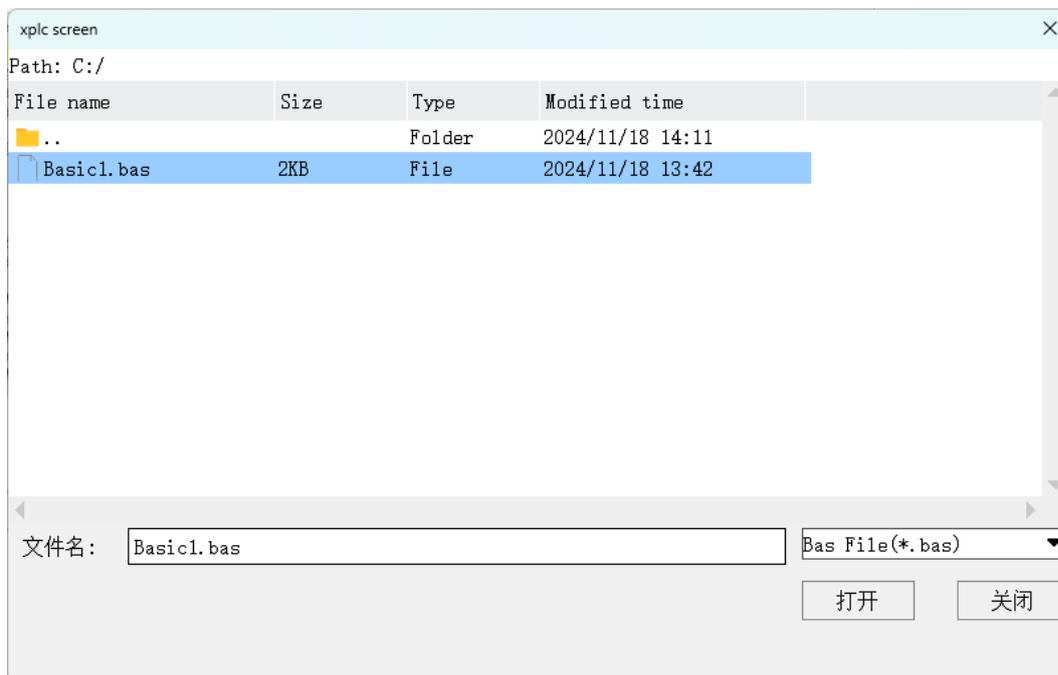


- 选中任意一行后，展开树形图的所有树节点。接着，点击“向上 10 行”按钮，如下二图所示，系统将自动从选中的那一行开始，向上额外选中 10 行数据，共选中 11 行。





➤ 点击“关闭”按钮，文件内容被关闭，界面回退到文件选择界面。



8.8. 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。

附录

RTHmi 版本功能介绍

不同的 RTHmi 版本可支持的 Hmi 功能有所不同，详见下表。

查询 RTHmi 版本的方法：RTSys 软件连接控制器→打开“控制器状态”→“RTHmiVersion: 10300(20241008)”。数值表示 V1.3.0 版本，括号中数值表示日期。

RTHmi 版本号	功能更新描述
V1.0.00 (ZDevelop 支持)	支持基础控件及功能、指令。
V1.1.00	<ol style="list-style-type: none"> 1. 新增滑块开关、表格、多行文本、三次文件编辑、中文输入法、CAD 控件； 2. 新增 DRAWEX 绘图指令集、滚动条绘图指令、CAD 基础指令集等； 3. 升级所有绘图控件； 4. 升级列表控件，支持滚动条显示，支持动态列表，支持选中函数调用、下拉函数调用，美化外观； 5. 背景图片、图片库支持 32 位图片带透明度显示；
V1.2.00 (RTSys 支持，ZDevelop 不支持该版本及后续版本的新增功能)	<ol style="list-style-type: none"> 1. 新增支持无损压缩图片、防失真缩放显示等； 2. 新增 SDF 抗锯齿绘图，通过 HMI_DEFAULTATTR 指令开关； 3. DRAWEX 指令新增透明度显示，通过 SETEX_ALPHA 设置； 4. 优化控件、zft 字体等内容；
V1.2.01	<ol style="list-style-type: none"> 1. 新增支持扫码枪等快速输入设备批量输入字符不丢失字符； 2. 新增了 CAD 自定义扩展指令集和高级指令集； 3. 支持多个刷新 dirty，刷新 Dirty 新增至 4 个；
V1.2.02	<ol style="list-style-type: none"> 1. 文本库新增支持语言代码页，优化 HMI 对外国语言的支持； 2. 新增控件垄断功能，支持下拉列表仅垄断鼠标，不垄断按键；字符显示控件增加直接输入中文方式，无需弹出键盘； 3. 指令 VKEY_MODE 、 HMI_LISTITEM 、 HMI_CONTROLATTR、HMI_DEALINFO 新增参数；
V1.2.03	<ol style="list-style-type: none"> 1. HMI_DEALINFO 指令增加参数 EDITNEXT、EDITPREV 获取下一个、上一个编辑控件； 2. HMI_CONTROLATTR 指令增加参数 FOCUS 设置、获取控件焦点状态；

	<ul style="list-style-type: none"> 3. 滑块控件新增支持小数滑动(最小刻度设置为 0 时生效); 4. 下拉列表控件中心支持选中背景高亮;
V1.2.04	<ul style="list-style-type: none"> 1. 支持文本库各国语言都可以直接使用 65001 的代码页。 2. HMICONTROLATTR 指令的"FOCUS"参数支持设置 0 关闭指定控件编辑焦点状态。
V1.3.0	<ul style="list-style-type: none"> 1. 文本库新增“以语言方式显示”或“以状态方式显示”功能; 2. Hmi 设置: 基本属性新增“不使用文本库格式文本”功能,“水平分辨率”和“垂直分辨率”可编辑功能; 3. 窗口导入新增窗口对比功能; 4. 控件箱元件新增报表视图、文件浏览器、菜单、树形图; 5. 新增操作指令: HMI_TABLEVALUE、HMI_TABLETEXT、HMI_TABLECURSOR、HMI_FILESEL、HMI_FILEPATH、HMI_FILEFILTER、HMI_MENUITEM; 6. SETEX_ALPHA 指令优化; 7. HMI_DEFAULTATTR、HMI_CONTROLATTR 指令新增一个参数; 8. 新增文件浏览器使用参考例程章节。

虚拟键值说明

虚拟键	键值	功能描述
VKEY_0~VKEY9	48~57	输入数字或字符“0~9”
VKEY_PLUS	43	输入字符“+”
VKEY_POINT	46	输入字符“.”或小数点
VKEY_MINUS	45	输入字符“-”或当前数值符号取反
VKEY_ENTER	10	输入确定，并切换编辑焦点
VKEY_CLR	13	清空当前输入
VKEY_SPACE	32	输入空格字符“ ”
VKEY_TAB	9	切换当前编辑焦点
VKEY_BackSpace	8	退格，从光标处删除一个字符
VKEY_DEL	127	标准删除
VKEY_ESC	27	标准退出，并关闭虚拟键模式
VKEY_SHIFT	171	shift 按键
VKEY_MENU	172	暂不可用
VKEY_CONTROL	173	暂不可用
VKEY_PAGE	174	翻页，暂不可用
VKEY_SWITCH	175	暂不可用
VKEY_INS	176	插入
VKEY_CAPS	177	切换大小写，输入复合按键时有效
VKEY_EN_CH	178	切换中英文输入法
VKEY_SYMBOL	179	符号（预留）
ZKEY_F1	128	快捷按键 F1
ZKEY_F2	129	快捷按键 F2
ZKEY_F3	130	快捷按键 F3
ZKEY_F4	131	快捷按键 F4
ZKEY_F5	132	快捷按键 F5
ZKEY_F6	133	快捷按键 F6
ZKEY_F7	134	快捷按键 F7
ZKEY_F8	135	快捷按键 F8
ZKEY_START	140	启动按键
ZKEY_STOP	141	停止按键
VKEY_LEFT	145	方向按键，向左移动

VKEY_RIGHT	146	方向按键，向右移动
VKEY_UP	147	方向按键，向上移动
VKEY_DOWN	148	方向按键，向下移动
ZKEY_1LEFT	150	JOG 按键
ZKEY_1RIGHT	151	JOG 按键
ZKEY_2LEFT	152	JOG 按键
ZKEY_2RIGHT	153	JOG 按键
ZKEY_3LEFT	154	JOG 按键
ZKEY_3RIGHT	155	JOG 按键
ZKEY_4LEFT	156	JOG 按键
ZKEY_4RIGHT	157	JOG 按键
ZKEY_5LEFT	158	JOG 按键
ZKEY_5RIGHT	159	JOG 按键
ZKEY_6LEFT	160	JOG 按键
ZKEY_6RIGHT	161	JOG 按键
VKEY_1_STAR	201	复合按键，1*
VKEY_2_ABC	202	复合按键，2abc 或 2ABC，通过 VKEY_CAPS 切换
VKEY_3_DEF	203	复合按键，3def 或 3DEF
VKEY_4_GHI	204	复合按键，4ghi 或 4GHI
VKEY_5_JKL	205	复合按键，5jkl 或 5JKL
VKEY_6_MNO	206	复合按键，6mno 或 6MNO
VKEY_7_PQRS	207	复合按键，7pqrs 或 7PQRS
VKEY_8_TUV	208	复合按键，8tuv 或 8TUV
VKEY_9_WXYZ	209	复合按键，9wxyz 或 9WXYZ
VKEY_POINT_HASH	210	复合按键，.#
VKEY_A~VKEY_Z	65~90	输入大写字母“A~Z”
VKEY_a~VKEY_z	97~122	输入小写字母“a~z”
其他	0~127	作为 ASCII 码输入

错误码列表

错误码	意义	可能原因
5000	LCD 号错误	
5001	Hmi 文件错误	窗口号相同

5002	LCD 号冲突	
5003	不支持对象	
5004	内存不够	
5005	控件层次错误	
5006	窗口号超过	
5007	无效窗口号	
5008	HMI 文件内容错误	
5009	窗口号重复	
5010	对象属性丢失	
5011	输入窗口有多个显示元件	
5012	ACTION 类型错误	
5013	事件过多	
5014	返回上个窗口失败	
5015	不能关闭基本窗口	
5016	字体中找不到对应字符	
5017	必须在 HMI 任务中使用	
5018	HMI 中对应的控件类型不支持这个指令	
5020	控件 ID 冲突	
5021	LCD 号错误	
5022	找不到可用 LCD	
5023	LCD 没有打开	
5024	LCD 无数据	
5025	程序复位	
5026	LCD 已经打开了	
5027	不是网络 LCD	
5028	不支持的压缩方式	
5029	颜色深度不支持	
5030	不支持的数据类型	
5031	设备号错误	
5032	LCD_SEL 不能使用	
5033	设置 REDRAW 不能再 DRAW 阶段	
5034	DRAW 函数只能在 DRAW 阶段	
5035	操作不能再 DRAW 阶段调用	

5036	内部 LCD 分辨率固定	
5037	LCD 分辨率超过	
5038	库文件名错误	
5039	字符过多	
5040	元件属性丢失	
5041	没有输入显示元件,无法输入	
5042	状态数过多	
5043	不支持的绘图属性	
5044	远程通讯设备名称错误	
5045	远程通讯数据没有更新	
5101	无效日期格式	
5102	控件不存在	
5103	多边形点数太少或太多	
5104	无空闲可用的滚动条	
5105	无效的滚动条 ID	
5106	尚不支持功能	
5107	未加载图形	
5108	文件已损坏	
5109	菜单参数错误	
5110	导出 table 空间不足, 溢出	
5111	不支持的数据类型	
5112	指令不支持的控件类型	
5113	数组溢出	传入数组下标超过最大值
5114	内部错误	
5115	通道溢出	传入通道号超过最大值