

V1.0

SLC DI MC MTS APC

# EASYXML 应用手册

## SINUMERIK 828D & 840DSL

西门子数控界面开发应用手册

## 目录

1	声明	1
2	概述	2
3	要求	3
3.1	软硬件及选项要求	3
3.2	与 EasyScreen 兼容性	3
3.3	句法要求	3
4	设计文档的结构	4
4.1	文档及保存路径	4
4.2	界面结构	5
4.2.1	区域划分	5
4.2.2	菜单树原理	5
4.2.3	界面组成	6
4.2.4	登入菜单 (xmldial.xml)	7
4.2.5	引用/加载其他脚本文件	8
5	编程基础	9
5.1	常用句法	9
5.1.1	句法 1: 开始+结束标签	9
5.1.2	句法 2: 带 "/" 的单标签	9
5.1.3	句法 3: 特殊句法	9
5.2	替代符号 (变量)	10
5.2.1	变量取值	10
5.2.2	多语言切换	10
5.2.3	间接引用	10
5.3	格式	10
5.3.1	格式化	10

# 目录

5.3.2	Display_format	12
5.3.3	_T, _F	12
5.4	调试诊断 (debug)	12
5.4.1	激活 Easy XML 诊断	13
5.4.2	诊断方法说明	13
5.4.3	诊断示例	14
5.5	Notepad++软件快捷设置	15
5.5.1	设置语言格式	15
5.5.2	设置快捷键	17
5.5.3	自动补全结束标签	18
5.5.4	在多个文档中查找	18
5.5.5	选择替换	19
5.6	样例库说明	20
6	界面主体	21
6.1	对话框	21
6.2	MENU	21
6.2.1	SOFTKEY	22
6.2.2	预定义软键	23
6.2.3	SOFTKEY 示例	24
6.2.4	RECALL	25
6.3	FORM	25
6.3.1	窗体属性	25
6.3.2	窗体事件说明	26
6.4	FUNCTION_BODY	27
6.5	变量定义	28
6.5.1	LET	28
6.5.2	TYPEDEF	31
6.6	系统变量定址	32
6.6.1	NC 变量	32
6.6.2	PLC 变量	32

# 目录

6.6.3	驱动参数	32
6.6.4	GUD 变量	33
7	窗体事件	34
7.1	INIT	34
7.2	PAINT	34
7.3	TIMER	35
7.4	FOCUS_IN	35
7.5	INDEX_CHANGED	36
7.6	EDIT_CHANGED	37
7.7	MESSAGE	38
7.8	KEY_EVENT	38
7.9	MOUSE_EVENT	39
7.10	GESTURE_EVENT	40
7.11	RESIZE	41
7.12	CLOSE	42
8	窗体控件 (GUI)	43
8.1	CONTROL 介绍	43
8.1.1	主属性	43
8.1.2	附加属性	44
8.2	控件类型	45
8.2.1	Edit 控件	45
8.2.2	Readonly 控件	46
8.2.3	Combobox 控件	46
8.2.4	Listbox 控件	47
8.2.5	Progressbar 控件	48
8.2.6	Scrollbar 控件	49
8.2.7	Progress_bar 窗体进度条	50
8.2.8	Graphicbox 控件	50
8.2.9	Pushbutton 控件	52
8.2.10	Switch 控件	55

# 目录

8.2.11	Radiobutton 控件	57
8.2.12	Checkbox 控件	57
8.2.13	Groupbox 控件	58
8.2.14	Scrollarea 控件	59
8.3	更改控件属性	59
9	函数和方法	61
9.1	数据处理	61
9.1.1	OP	61
9.1.2	DATA	61
9.1.3	DATA_LIST	61
9.1.4	DATA_ACCESS	62
9.1.5	TYPE_CAST	63
9.1.6	UPDATE_CONTROLS	64
9.2	复位重启	64
9.2.1	CONTROL_RESET	64
9.2.2	WAITING	64
9.2.3	HMI_RESET	65
9.3	用户循环	65
9.3.1	CREATE_CYCLE	65
9.3.2	MMC	67
9.4	窗体操作	68
9.4.1	NAVIGATION	68
9.4.2	OPEN_FORM	68
9.4.3	CLOSE_FORM	69
9.4.4	POWER_OFF	69
9.4.5	MSGBOX	70
9.4.6	PRINT	71
9.4.7	PROGRESS_BAR	72
9.4.8	SEND_MESSAGE	72
9.4.9	SHOW_CONTROL	73
9.4.10	SLEEP	74
9.4.11	STOP	74
9.4.12	SWITCHTOAREA	74

# 目录

9.4.13	SWITCHTODIALOG	74
9.4.14	SWITCHTODYNAMICTARGET	75
9.5	窗体显示	75
9.5.1	CAPTION	75
9.5.2	TEXT	75
9.5.3	BOX	76
9.5.4	IMG	77
9.5.5	HELP_CONTEXT	78
9.5.6	语言文本	79
9.6	条件-循环指令	82
9.6.1	FOR 循环	82
9.6.2	While 循环	83
9.6.3	Do-while 循环	83
9.6.4	Switch 指令	84
9.6.5	Break	84
9.6.6	示例	85
9.7	信号处理	85
9.7.1	REQUEST	85
9.8	分析 XML 文件	86
9.9	系统函数	88
9.9.1	读写系统数据	88
9.9.2	选项处理	89
9.9.3	文件处理	90
9.9.4	字符串处理	91
9.9.5	数据位处理	93
9.9.6	控件处理	93
9.9.7	像素处理	94
9.9.8	图像处理	96
9.9.9	数学运算	96
9.9.10	PI 服务	97

# 目录

10	附录	99
10.1	参考手册	99
10.2	颜色代码	99
10.3	运算符	100
10.4	预定义按键	101
10.5	系统界面变量	101
10.6	语种缩写表	101
11	作者及版本	102

MTSAPC

## 1 声明

本手册及样例包均为免费提供，仅供参考。所述版本、应用及案例可能与用户实际应用不符，请用户在使用前认真阅读相关使用说明，根据自身的应用环境及机床特点进行调整，并进行严格的测试，以规避可能存在的风险。对于在使用中发生的人员、财产等损失，由用户自行承担。

以上声明内容的最终解释权归西门子（中国）有限公司所有，后续内容更新恕不做另行通知。

MTS APC

## 2 概述

“Easy XML”界面开发能够帮助客户设计出符合专有应用和客户操作需求的 Operate 用户界面，与“EasyScreen”界面开发的功能和效果大部分相同。

是西门子系统在 Operate 平台上基于 XML（可扩展标记语言，**EX**tensible **M**arkup **L**anguage）脚本语言的界面开发，语法结构上与标准的 XML 脚本语言相似，具体的功能和属性上有系统预定义方法。

Easy XML 界面开发具有如下特点：

- XML 元素均使用 XML 标签进行定义，每组标签都有打开和关闭标签
- 较 EasyScreen 而言，代码量稍大，变量可以更灵活，逻辑可以更复杂
- 门槛稍高，较趋近于前端开发编程，有一定界面开发基础的用户，上手较快
- 系统已定义应用框架和一些具体的语法规则、界面功能等
- 主要通过 XML 文件来实现界面开发
- 主界面入口固定，由固定文件名解析
- 支持运用界面底层的代码和算法实现用户工艺应用的需求
- 支持用户循环，可进行生成加工程序的界面开发
- 可自定义界面窗口的大小，支持不同分辨率下的界面切换
- 支持多语言切换，系统会从随附的语言数据库中读取要显示的文本

版本说明：

- Operate V4.7 版本起支持 Easy XML 界面开发，V4.8 版本新增了一些新功能，新增功能及其详解见 SINUMERIK 828D 或 840Dsl Easy XML 官方手册
- Operate V4.8 SP3 版本之前仅支持在<CUSTOM>区开发界面，入口为固定的脚本文件名

开发工具：文本、XML 文件、UTF8 编辑器，常用软件有

Notepad++	
Ultraedit	
Notepad2	
Notepad	
其他	...

说明：

本手册中提及的对话框即为语法结构中的 `dialoggui` 标签；窗体即为语法结构中的 `form` 标签；菜单为语法结构中的 `menu` 标签。

### 3 要求

#### 3.1 软硬件及选项要求

EasyXML 在界面开发中没有界面幅数限制，支持的软硬件详见下表。

	828D (车/铣/磨)			840Dsl		
硬件	PPU 24x/26x/28x/27x/290			NCU710	NCU720	NCU730
软件	SW24	SW26	SW28			
Operate V4.7 及以上	√/√/√	√/√/√	√/√/√	√	√	√
选项	无					
说明	√: 标配, 0: 选项, -: 不支持					

#### 3.2 与EasyScreen兼容性

- 无法与 Easy Screen 同时在<CUSTOM>区域混合使用
- 在界面开发区域分开时可与 EasyScreen 兼容使用 (例如: 可以使用 EasyScreen 在 <Machine>区开发了界面, 使用 EasyXML 在<CUSTOM>区开发界面。)
- 这里的开发区域指的是系统界面框架定义的[area]

#### 3.3 句法要求

- 自定义界面变量不允许与系统变量重名 (参见 系统变量手册 PGAsI)
- 界面文件的编码格式为 UTF8
- 独立的对话框 (如入口界面) 必须有<DialogGui> ... </DialogGui> 标签

##### 关键语法示例

- 所有的 XML 文件名都应为小写字母
  - √: main\_function.xml; 正确
  - x: Main\_function.xml; 错误
- XML 标签对大小写敏感, 必须使用相同的大小写来编写打开标签<op>和关闭标签</op>:
  - √: <op>这是正确的</op>
  - √: <OP>这是正确的</OP>
  - x: <Op>这是错误的</op>
- XML 功能和属性句法中兼容大小写, 如:
  - √: <softkey POSITION="1" >;正确
  - √: <softkey position="1" >;正确

## 4 设计文档的结构

### 4.1 文档及保存路径

.xml 文件	界面脚本文件，配置菜单及界面元素
.ts 文件	语言文本文件，用于多系统语言切换时的界面文本显示
.png 文件	图形文件，界面图形元素的引用
.html 文件	帮助文件，按下系统帮助按键时索引到的帮助说明

SINUMERIK Operate 的文件有固定的目录结构。一级目录有规定的优先级，优先级确定了当一份文件位于多个目录中时装载哪个目录中的文件。

文件的查找范围覆盖了以下目录：

1. <安装路径>/user/sinumerik/hmi
2. <安装路径>/oem/sinumerik/hmi
3. <安装路径>/addon/sinumerik/hmi
4. <安装路径>/siemens/sinumerik/hmi

目录 user 的优先级最高，然后依次是目录 oem、addon 和 siemens。首先在目录 user 中查找文件。如果在其中没有找到文件，则接着在目录 oem 中查找，依此类推。



每个上述目录都包含了子目录，子目录中保存着特定数据：

以 oem 目录为例	子目录	扩展名	保存路径
	appl	xml png,bmp..	界面脚本文件，图片
	cfg	ini	配置文件
	hlp	html	在线帮助文件
	ico	png,bmp..	图标和图片
	lng	ts,qm	语言文件
	...	...	...
	...	...	...

原则上在 siemens 目录中不允许保存界面开发文件，因为其中保存的是系统文件。

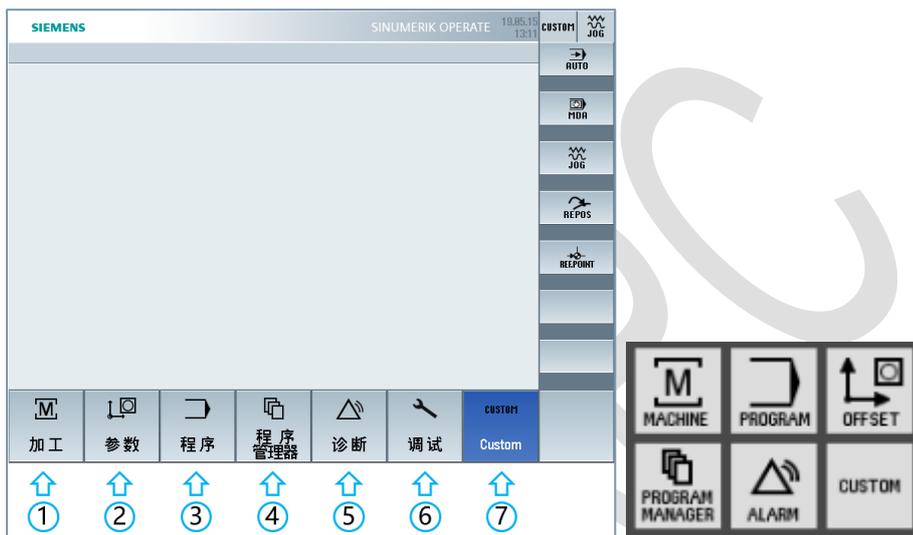
## 4.2 界面结构

### 4.2.1 区域划分

系统界面分为 7 个操作区，操作区 7【Custom】默认可用于 EasyXML 界面开发，预定义的主界面入口文件为：xmldial.xml。（见 登入菜单 章节）

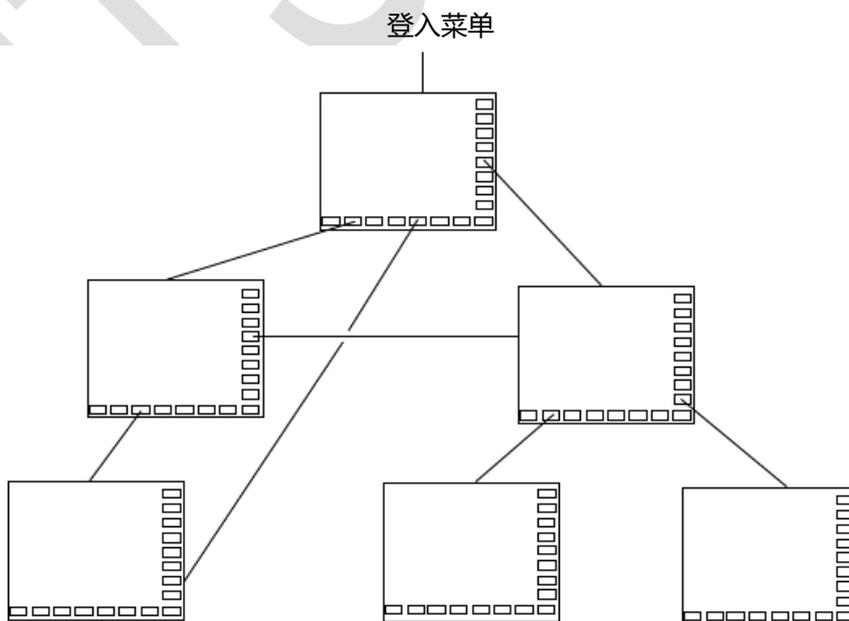
操作区 1-6 为系统界面区，在 Operate V4.8 SP4 版本以后支持用户自行配置。

（配置语法见 SINUMERIK 828D 或 840Dsl Easy XML 手册，附录章节附有下载链接）

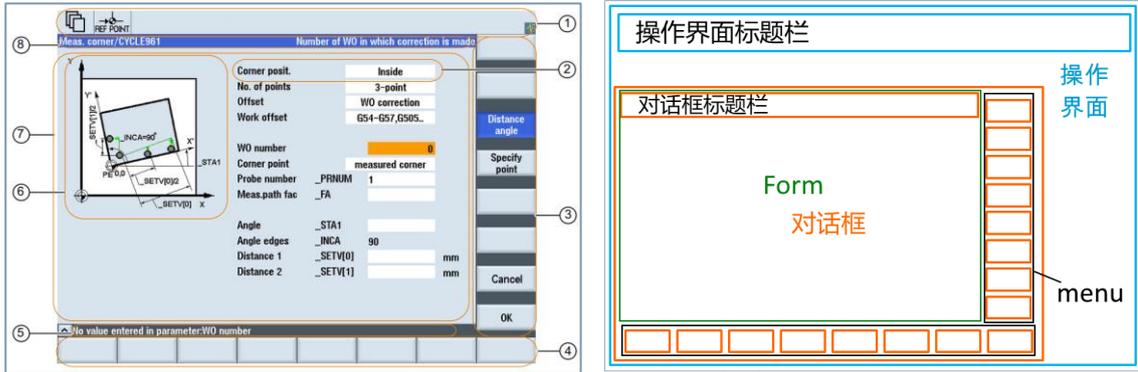


### 4.2.2 菜单树原理

- 多个相互关联的对话框构成了一个菜单树。
- 如果能从一个对话框切换到另一个对话框，或者返回到上一个对话框，则表示这两个对话框间存在关联。

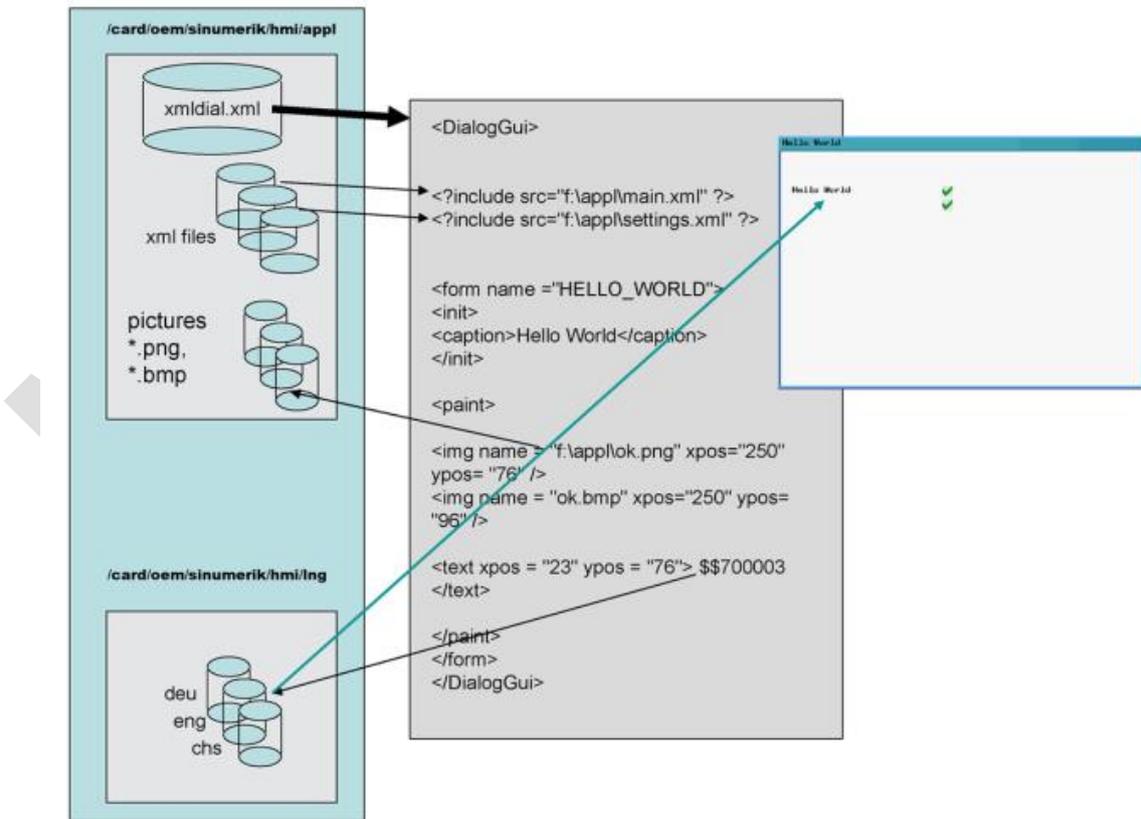


### 4.2.3 界面组成



- |                  |                   |
|------------------|-------------------|
| ① 操作界面标题栏：机床状态显示 | ⑤ <form>对话框输出状态栏  |
| ② <form>界面变量     | ⑥ <form>界面变量：图形元素 |
| ③ <menu>垂直按键：8个  | ⑦ <form>界面        |
| ④ <menu>水平按键：8个  | ⑧ <form>标题栏       |

### 文件与用户对话框配置的相关性

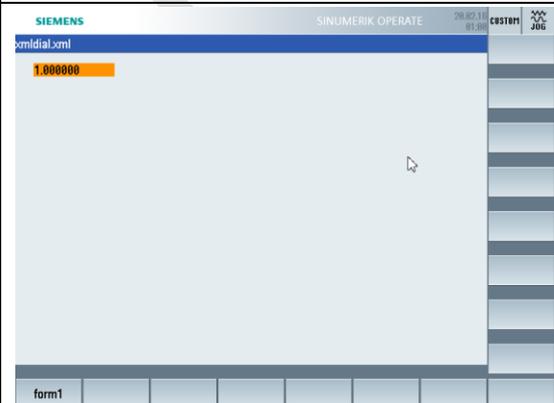
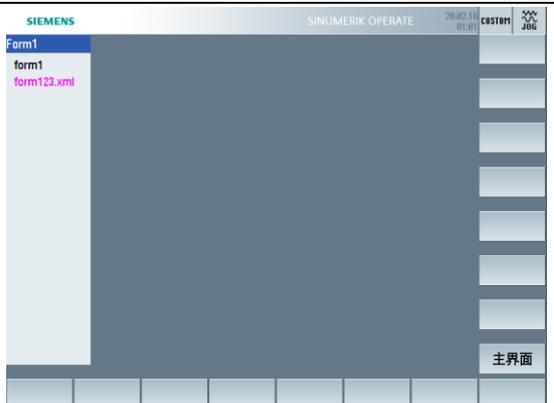


#### 4.2.4 登入菜单 (xmldial.xml)

EasyXML 默认的登入菜单为：脚本文件“xmldial.xml”中的“main”菜单。

xmldial.xml 文件中由 <DialogGui> ... </DialogGui> 标签定义了独立的对话框。菜单和窗体，还有其他脚本文件的加载都在此对话框中实现。其他 xml 脚本文件中不需要再定义 <DialogGui> ... </DialogGui> 对话框标签，否则无法由 main 菜单中跳转到其他脚本文件。

**注：**通过其他入口进入的其他 xml 脚本文件需要定义<DialogGui>对话框标签，如 MMC 指令，通过 PLC 指令调用 EasyXML 界面等。

示例	
<pre>xmldial.xml &lt;DialogGui&gt;   &lt;menu name = "main"&gt;     &lt;open form name = "form_main" /&gt;     &lt;softkey POSITION="1" &gt;       &lt;caption&gt;form1 &lt;/caption&gt;       &lt;navigation&gt;menu_form1 &lt;/navigation&gt;     &lt;/softkey&gt;   &lt;/menu&gt;    &lt;form name="form_main"&gt;     &lt;init&gt;       &lt;caption&gt;xmldial.xml &lt;/caption&gt;       &lt;control name = "edit1" xpos = "022" ypos = "34" refvar="nck/Channel/Parameter/R[1]" /&gt;     &lt;/init&gt;   &lt;/form&gt;    &lt;?include src="f:\appl\form123.xml" ?&gt; &lt;/DialogGui&gt;</pre>	
<pre>form123.xml (其他脚本文件) &lt;menu name = "menu_form1"&gt;   &lt;open form name = "form_form1" /&gt;   &lt;softkey POSITION="16" &gt;     &lt;caption&gt;主界面 &lt;/caption&gt;     &lt;navigation&gt;main &lt;/navigation&gt;   &lt;/softkey&gt; &lt;/menu&gt;  &lt;form name="form_form1" xpos="0" ypos="22" width="100"&gt;   &lt;init&gt;     &lt;caption&gt;Form1 &lt;/caption&gt;   &lt;/init&gt;   &lt;paint&gt;     &lt;text xpos="10" ypos="30" &gt;form1 &lt;/text&gt;     &lt;text xpos="10" ypos="50" color="#ff00ff"&gt;form123.xml &lt;/text&gt;   &lt;/paint&gt; &lt;/form&gt;</pre>	
<b>Main(xmldial.xml)</b>	<b>Form1(form123.xml)</b>
	

#### 4.2.5 引用/加载其他脚本文件

- 引用的绝对路径为 "f:\子目录\文件名.扩展名" ;  
如: "f:\appl\project1.xml", f:/appl/sk\_home1.png。
- 主界面 xmldial.xml 外的其他 xml 脚本文件需要引用后使用。有两种引用方式:

1. 使用 include

```
<?include src="f:\appl\main_function.xml" ?>
```

在 xmldial.xml 脚本文件中引用。在进入主界面时依次载入引用的脚本文件。

语法:

```
<?include src="path name" ?>
```

2. 使用 dynamic\_include

```
<dynamic_include src="f:\appl\project1.xml"/>
```

与标签 INCLUDE 相反, 在执行相应指令时才进行读取。在大型项目中该标签的使用能缩短用户使用或循环支持的载入时间。此外, 对系统资源的平均需求也会下降, 因为在会话过程中不需要一直调用所有的对话框。

语法:

```
<dynamic_include src="path name"/>
```

## 5 编程基础

### 5.1 常用句法

#### 5.1.1 句法 1: 开始+结束标签

开始标签+元素+结束标签: <CAPTION>Hello World</CAPTION>

示例:

```
<let name="btn_col_fg" type="string">#33495e</let>
<op>int_i=0</op>
<caption>(file:apc_mode.xml) </caption>
<function_body name="init_setting">
  <op>
    part_type="Nck/Channel/parameter/rpa[u1,57]"
    mode_type="Nck/Channel/parameter/rpa[u1,58]"
    mode_tol="Nck/Channel/parameter/rpa[u1,59]"
  </op>
</function_body>
```

#### 5.1.2 句法 2: 带 "/" 的单标签

标签的结束括号 ">" 前加 "/", "/" 与 ">" 之间无空格: <.../>

示例

```
<show_control name="lable_status_bar1" type="true"/>
<img xpos = "0" ypos = "0" width="565" height="400" name = "f:/appl/mode_die_bg4_1300x768.png" />
<control name="edit1" xpos = "50" ypos = "50" refvar="plc/mb170" />
```

#### 5.1.3 句法 3: 特殊句法

1. 在 xmldial.xml 中引用文件:

起始标签<?include ; 结束标签 ?>

```
<?include src="f:\appl\main function.xml" ?>
```

2. 单行注释:

起始标签<!-- ; 结束标签 -->

```
<!-- form入口 -->
```

3. 多行注释:

起始标签<!-- ; 结束标签 -->

```
<!-- 按键属性定义
<let name="sk9text" type="string" ></let>
<let name="sk10text" type="string" >f:/appl/sk_part1.png</let>
<let name="sk11text" type="string" ></let>
<let name="sk12text" type="string" >f:/appl/sk_set1.png</let>
<let name="sk13text" type="string" ></let>
<let name="sk14text" type="string" ></let>
<let name="sk16text" type="string" >f:/appl/sk_gcode1.png</let>
<let name="sk16status" type="int" ></let>
-->
```

## 5.2 替代符号 (变量)

### 5.2.1 变量取值

界面局部变量可使用前置的 \$ 符号 实现不同特性的转换。

\$: 将变量的值传输给标签

例:

```
<let name="my_ypos">100</let>
<control name = "edit1" xpos = "322" ypos = "$my_ypos" refvar="nck/Channel/Parameter/R[1]" />
```

### 5.2.2 多语言切换

\$\$: 用于语言文本, 源文本标签, 在关联的 xxx.ts 文本文件中有变量表示的文本。

```
<DialogGui textfile="myscreens">
...
</DialogGui>
<caption>$$MY_TEXT2</caption>
<text xpos = "120" ypos="158">$$MY_TEXT1</text>
```

```
myscreens_eng.ts
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE TS>
<TS>
<context>
  <name>EASY_XML</name>
  <message>
    <source>MY_TEXT1</source>
    <translation>text1 from textfile</translation>
  </message>
  <message>
    <source>MY_TEXT2</source>
    <translation>text2 from textfile</translation>
  </message>
</context>
</TS>
```

### 5.2.3 间接引用

\$\$\$: 如果标签要以字符串作为属性值或值, 则应在变量名称前加上符号 \$\$\$。

例:

```
<let name="field_name" type="string">abc</let>
<caption>$$$field_name</caption>
```

## 5.3 格式

### 5.3.1 格式化

利用语法"%<宽度>.<小数点后位数>类型"可以对数据进行格式化, 常用的格式化有:

控件的显示格式, 利用 format 属性实现

Print 打印的数据格式, 使用"%<宽度>.<小数点后位数>类型"来定义

### 5.3.1.1 Print

语法:

```
<PRINT name="变量名称" text="text %格式化"> Variable, ... </PRINT> <!--输出字符串到变量中-->
```

```
<PRINT text="text %格式化"> Variable, ... </PRINT> <!--输出字符串到状态栏-->
```

说明:

字符 “%” 会对作为值给定的变量进行格式化。 %[标记] [宽度] [.小数点后面的位置] 类型

- 标记:

用于确定输出格式的可选字符:

- 右对齐或左对齐 (“-” 用于左对齐)

- 前面加零 (“0”)

- 使用空格符填充

- 宽度:

确定一个非负数最小输出宽度的依据。如果待输出值占用的位置少于依据所确定的位置, 则使用空格符填充空缺的位置。

- 小数点后面的位置:

使用浮点数时优化参数用来确定小数点后面的位置数。

- 类型:

类型字符用来确定打印指令要输出哪些数据格式。该字符必须给定。

- d: 整数值

- f: 浮点数

- s: 字符串

- 值:

其值应当插入到文本中的变量数量。

变量类型必须与格式规定的相应类型标识一致并用逗号加以间隔。

示例:

在状态栏中输出文本: 信息文本

```
<PRINT text="信息文本" />
```

使用变量格式输出文本: M43

```
<LET name="trun_dir"></LET>
```

```
<PRINT text="M%d">trun_dir</PRINT>
```

在字符串变量中使用变量格式输出文本: str=M43

```
<LET name="trun_dir"></LET>
```

```
<LET name="str" type="string" ></LET>
```

```
<print name="str" text="M%d ">trun_dir</print>
```

### 5.3.1.2 Format

在 control 控件中添加 format 属性, 可定义该控件的显示格式: format="%"<宽度>.<小数点后位数>类型"

借助系统函数 nfunc.displayresolution 可以提供由控制系统确定的浮点数转换规则。

示例:

```
<let name="display_res" type="string"></let>
```

...

```
<function name="ncfunc.displayresolution" return="display_res" />
```

```
<control name = "cdistToGo" xpos = "210" ypos = "156" refvar="nck/Channel/GeometricAxis/
progDistToGo[2]" hotlink="true" height="34" fieldtype="readonly" format="$$$display_res"
time="superfast" color_bk="#ffffff"/>
```

### 5.3.2 Display\_format

该属性定义了指定变量的处理格式。在存取 PLC 浮点型变量时必须使用该属性，因为要通过双字读取来进行存取。允许使用下列数据格式：

- FLOAT
- INT
- DOUBLE
- STRING

示例:

```
<control name = "ca10" xpos = "300" ypos = "300" refvar="PLC/MD100" hotlink="true"
display_format="float" format = "%9.3F"/>
```

### 5.3.3 \_T, \_F

字符串输入格式: `_T`文本或字符串，字符串参数的输入或操作都需要使用前置标识 `_T` 说明。

浮点数输入格式: `_F%9.3f`浮点数，需要转换格式的变量使用前置标识 `_F%<格式>` 说明。

示例:

```
<LET name="buffer" type="string"></LET>
```

...

```
<OP> buffer = _T"unformatted value R0= " + "nck/Channel/Parameter/R[0]" + _T" and " + _T"$$$85051" +
_T" formatted value R1= " + _F%9.3fnck/Channel/Parameter/R[1]" </OP>
```

```
<function name="control.delete">_T"down"</function>
```

```
<op> define_sk_type = _T"PRESSED" </op>
```

```
<op>bmp_name = _T"f:\appl\red_led_on.bmp"</op>
```

结果:

Buffer 字符串为: unformatted value R0=10 and with formatted value R1=123456.789

## 5.4 调试诊断 (debug)

诊断功能可检查 XML 语法并运行属于项目的所有脚本。为此还会检查功能、菜单、格式和变量的存在/有效性。发现的错误会罗列出来。

“Easy XML 诊断”功能即可使用 DialogGui 标签中的属性，也可通过显示机床数据来激活。

### 5.4.1 激活 Easy XML 诊断

> V4.8 SP2 版本：添加 diagnose 属性="true"

<DialogGui *diagnose="true"* >

...

...

</DialogGui >

> V4.8 SP3 版本以后：设置显示机床数据

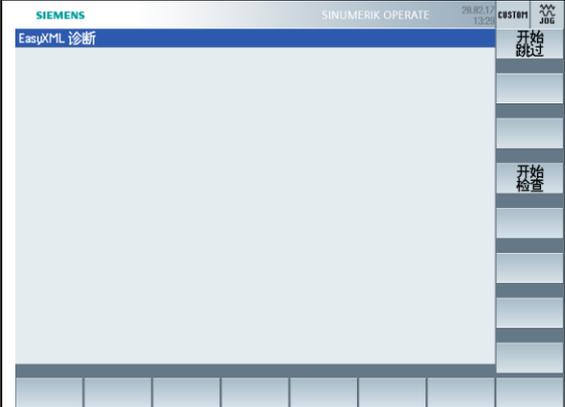
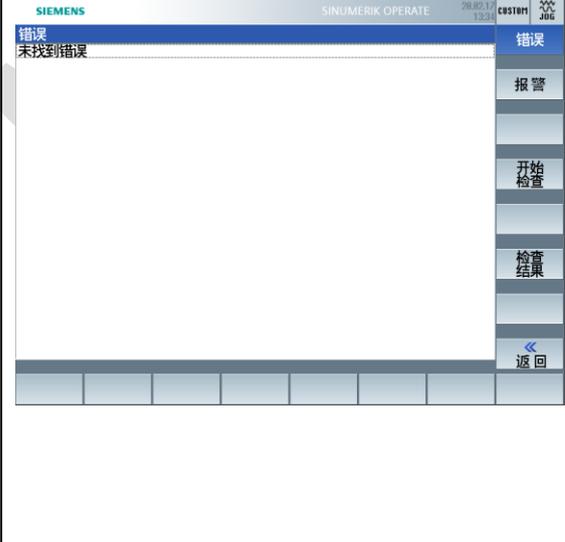
9113 \$MM\_EASY\_XML\_DIAGNOSE 设置为 1H

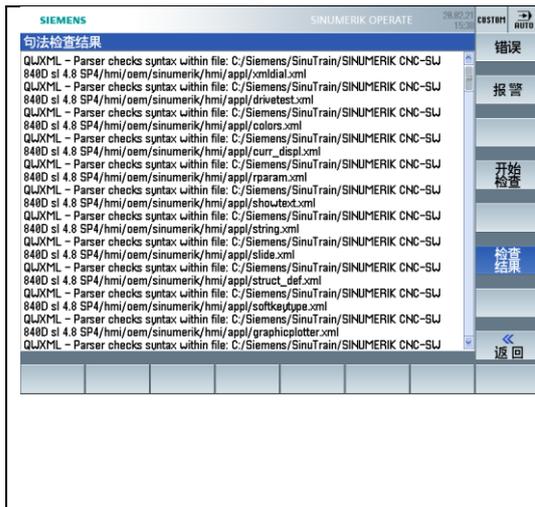
9112	\$MM_HMI_SKIN	1	po	
9113	\$MM_EASY_XML_DIAGNOSE	1H	im	
9114	\$MM_SIDESCREEN	0	po	
9115	\$MM_SAVE_CREDENTIALS	1	im	

EasyXML 诊断模式

通用 通道 轴 显示  
 设定数据 设定数据 设定数据 机床数据

### 5.4.2 诊断方法说明

	<p>激活诊断后，进入对话框时首先进入诊断界面。</p> <p><b>【开始跳过】</b></p> <ul style="list-style-type: none"> <li>- 跳过诊断检查，直接进入对话框。</li> </ul> <p><b>【开始检查】</b></p> <ul style="list-style-type: none"> <li>- 开始检查 EasyXML 语法错误，并给出检查报告</li> </ul>
	<p>检查结束后，显示检查报告：</p> <p><b>【错误】</b></p> <ul style="list-style-type: none"> <li>- 筛选出语法错误，错误会影响界面的正常显示，应按提示进行语法修改。</li> </ul> <p><b>【报警】</b></p> <ul style="list-style-type: none"> <li>- 显示界面语法的预警信息。</li> </ul> <p><b>【开始检查】</b></p> <ul style="list-style-type: none"> <li>- 界面文件更新后，可点击该按钮再次进行检查。</li> </ul> <p><b>【检查结果】</b></p> <ul style="list-style-type: none"> <li>- 检查结果日志，记录检查的文档及错误信息。</li> </ul>



**【检查结果】**

- 显示所有检查结果。

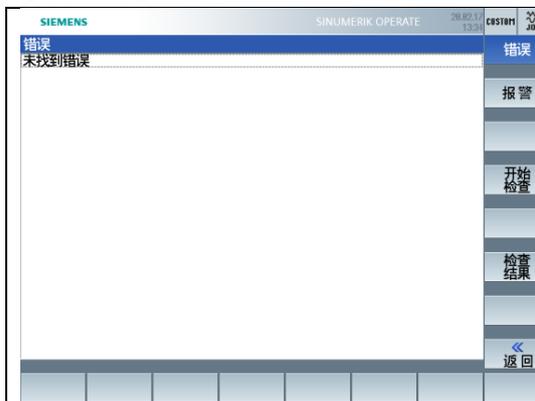
检查结果会保存在 user 目录 log 文件夹中  
 \user\sinumerik\hmi\log\easyXML\  
 easyX\_diag\_results.txt

```

address could not be checked (offline mode) : GUD/STR/
67 C:/Siemens/SinuTrain/SINUMERIK CNC-SW 840D sl 4.8 SP4/
address could not be checked (offline mode) : GUD/STR/
68 C:/Siemens/SinuTrain/SINUMERIK CNC-SW 840D sl 4.8 SP4/
address could not be checked (offline mode) : GUD/STR/
69 C:/Siemens/SinuTrain/SINUMERIK CNC-SW 840D sl 4.8 SP4/
address could not be checked (offline mode) : GUD/STR/
70 C:/Siemens/SinuTrain/SINUMERIK CNC-SW 840D sl 4.8 SP4/
address could not be checked (offline mode) : GUD/STR/
71 C:/Siemens/SinuTrain/SINUMERIK CNC-SW 840D sl 4.8 SP4/
address could not be checked (offline mode) : GUD/INT/
72 C:/Siemens/SinuTrain/SINUMERIK CNC-SW 840D sl 4.8 SP4/
address could not be checked (offline mode) : GUD/INT/
73 C:/Siemens/SinuTrain/SINUMERIK CNC-SW 840D sl 4.8 SP4/
Warning:: address could not be checked (offline mode)
74 C:/Siemens/SinuTrain/SINUMERIK CNC-SW 840D sl 4.8 SP4/
Warning:: address could not be checked (offline mode)
75 C:/Siemens/SinuTrain/SINUMERIK CNC-SW 840D sl 4.8 SP4/
Warning:: address could not be checked (offline mode)
76 C:/Siemens/SinuTrain/SINUMERIK CNC-SW 840D sl 4.8 SP4/
Warning:: address could not be checked (offline mode)
77 C:/Siemens/SinuTrain/SINUMERIK CNC-SW 840D sl 4.8 SP4/
Warning:: address could not be checked (offline mode)
  
```

### 5.4.3 诊断示例

#### 示例 1: 无错误

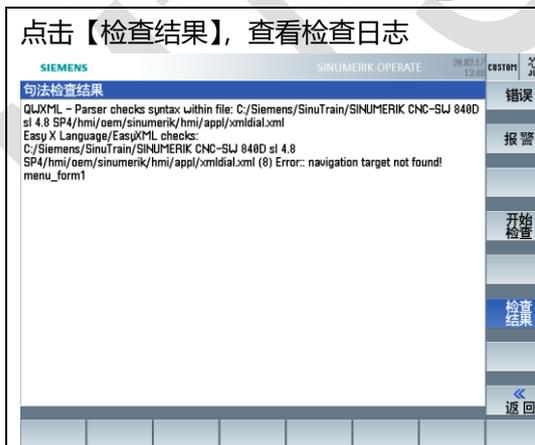


**【无错误】**

**【后续操作】**

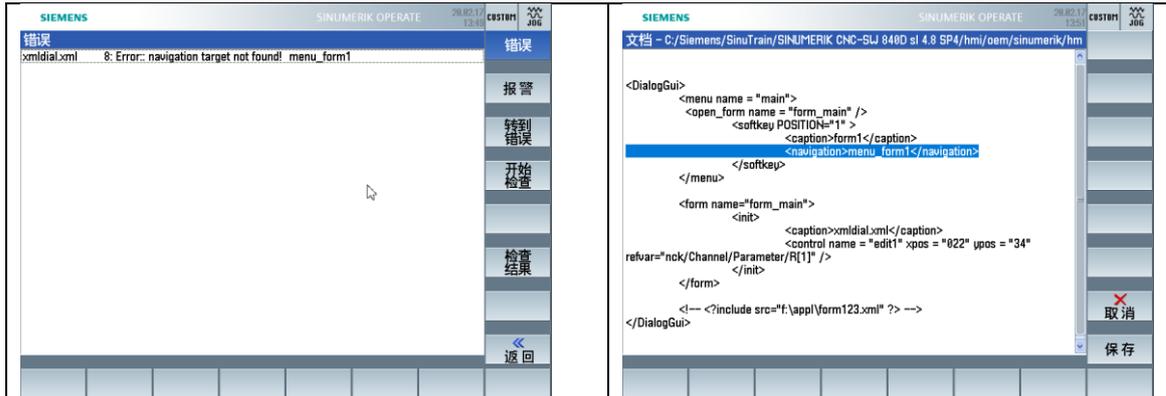
点击【返回】 - 【开始跳过】进入界面。

#### 示例 2: 显示错误: 导航菜单 menu\_form1 未找到



点击【错误】显示错误语法

点击【转到错误】可跳转到错误位置:



示例 3：显示报警：未找到变量地址



## 5.5 Notepad++ 软件快捷设置

Notepad++软件上有一些能够提高编程效率、减少语法错误的设置，辅助我们快捷高效的码代码。当然，其他的编程工具也会有类似的功能，这里就不一一列举了，如果您也使用了 Notepad++ 编程工具，建议您检查如下设置。

### 5.5.1 设置语言格式

- 设置为 XML 语言格式：

选择菜单栏中的【语言】菜单

文件(F) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L) 设置(I) 工具(O) 宏(M) 运行(R)

在其下拉菜单中选择【XML】。

T >

V >

XML

YAML

自定义语言格式...

编程界面就会以 XML 语法特点来显示：方便读写

```

<menu name = "main">
  <open_form name = "form_main" />
  <softkey POSITION="1" >
    <caption>按键名称 </caption>
  </softkey>
</menu>

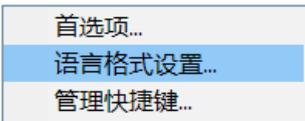
```

- 自行定义语言格式

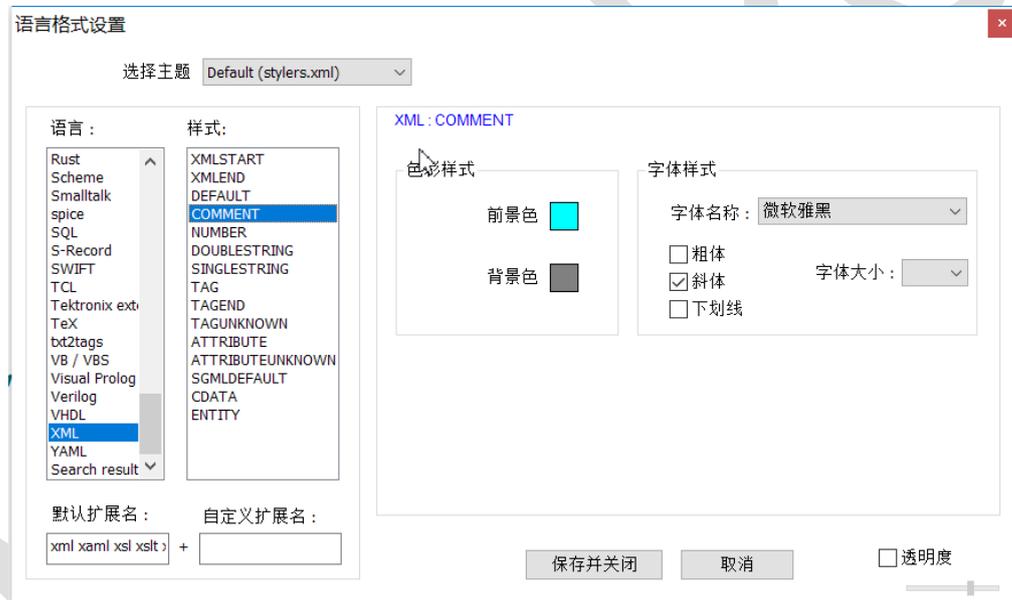
选择菜单栏中的【设置】菜单

文件(E) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L) 设置(I) 工具(O) 宏(M) 运行(R)

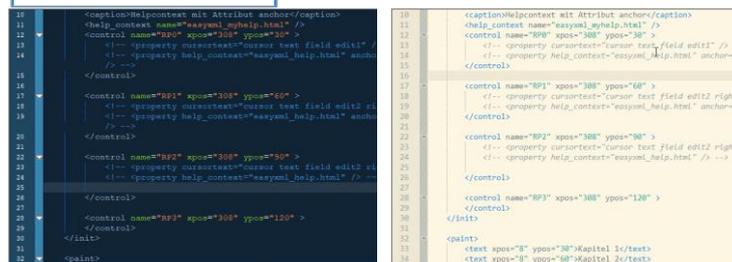
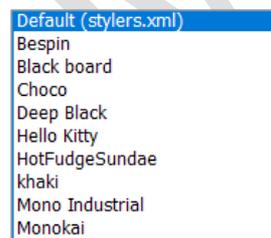
在其下拉菜单中选择【语言格式设置】。



在弹出的对话框中



可通过【选择主题】切换不同主题格式。



还可选择 XML 语言下的各样式选项，自定义字体和颜色等。（例如：COMMENT 代表注释）



### 5.5.2 设置快捷键

选择菜单栏中的【设置】菜单

文件(E) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L) 设置(I) 工具(O) 宏(M) 运行(R)

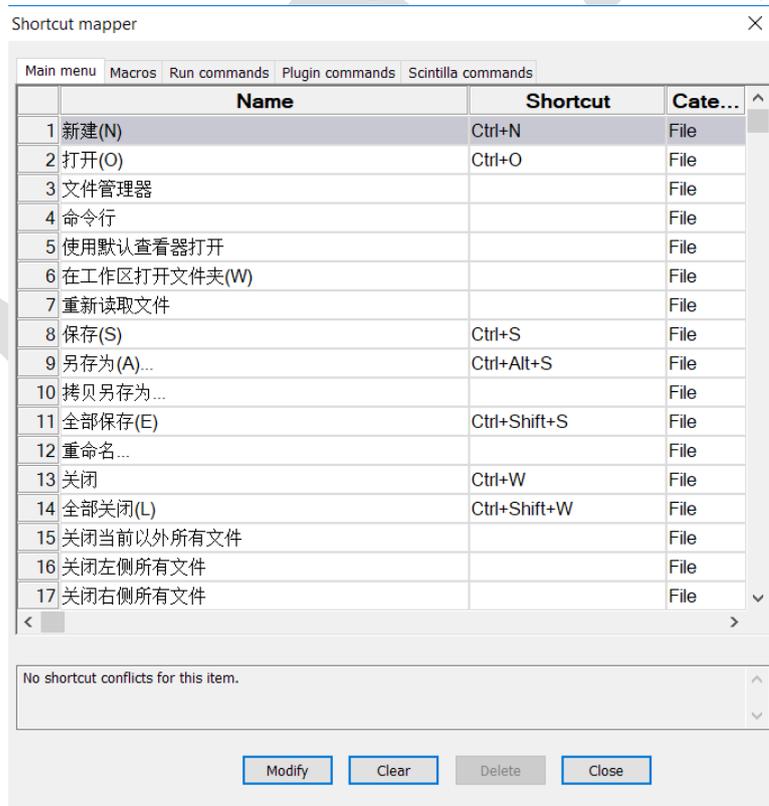
在其下拉菜单中选择【管理快捷键】。

首选项...

语言格式设置...

**管理快捷键...**

在弹出的对话框中，选择 Main\_menu 组



建议记忆常用的快捷键，如添加和删除注释（54-57 行）。

54	设置行注释	Ctrl+E	Edit
55	取消行注释	Ctrl+Shift+E	Edit
56	区块注释	Ctrl+Q	Edit
57	清除区块注释	Ctrl+Shift+Q	Edit

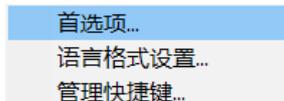
如与其他软件的快捷键冲突则需要自行设置为其他按键组合。

### 5.5.3 自动补全结束标签

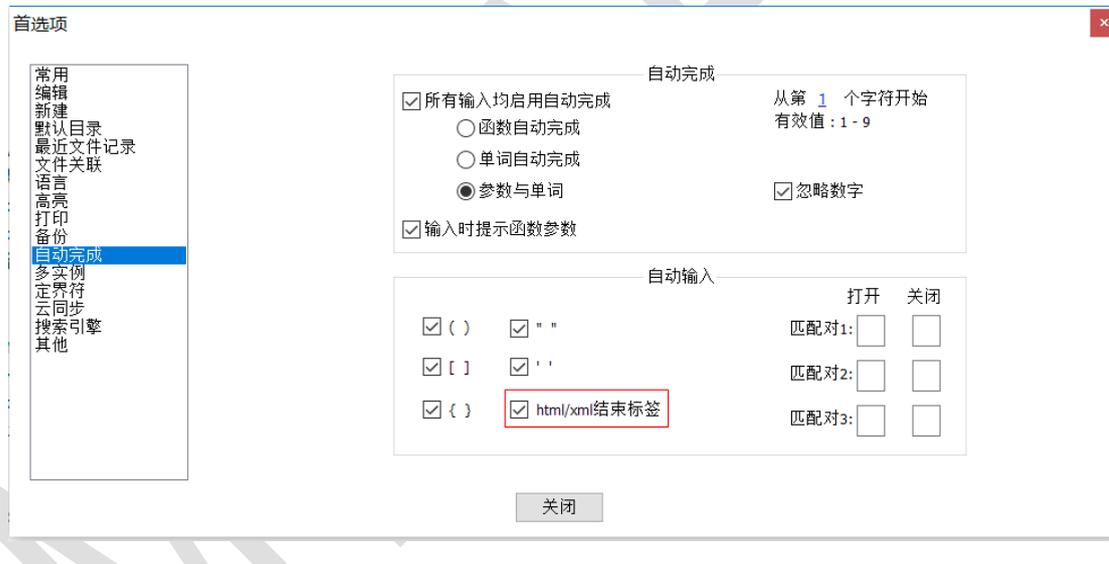
选择菜单栏中的【设置】菜单

文件(E) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L) 设置(I) 工具(O) 宏(M) 运行(R)

在其下拉菜单中选择【首选项】。



在弹出的对话框中，选择左侧列表框中的【自动完成】，在右侧的自动输入中，勾选 html/xml 结束标签。



### 5.5.4 在多个文档中查找

当需要查询某个代码时，却又忘记了其所在文档的名称，可以将样例库中的文档都打开。

点击快捷键“Ctrl+F”，在弹出的对话框中，在查找目标输入框填入要查找的代码；点击右侧【查找所有打开文件】按钮。



### 5.5.5 选择替换

当需要在某个局部区域替换多个代码时，可使用此功能。

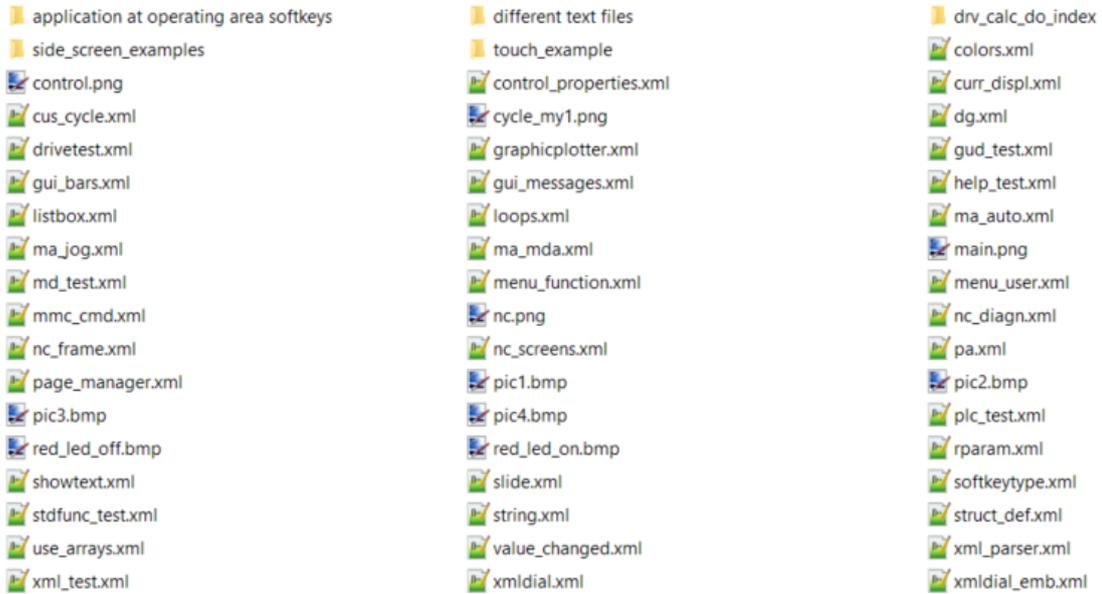
选中要替换的局部区域，点击快捷键“Ctrl+H”，在弹出的对话框中，勾选“选取范围内”，这样就可以仅在选择的区域进行代码替换了。



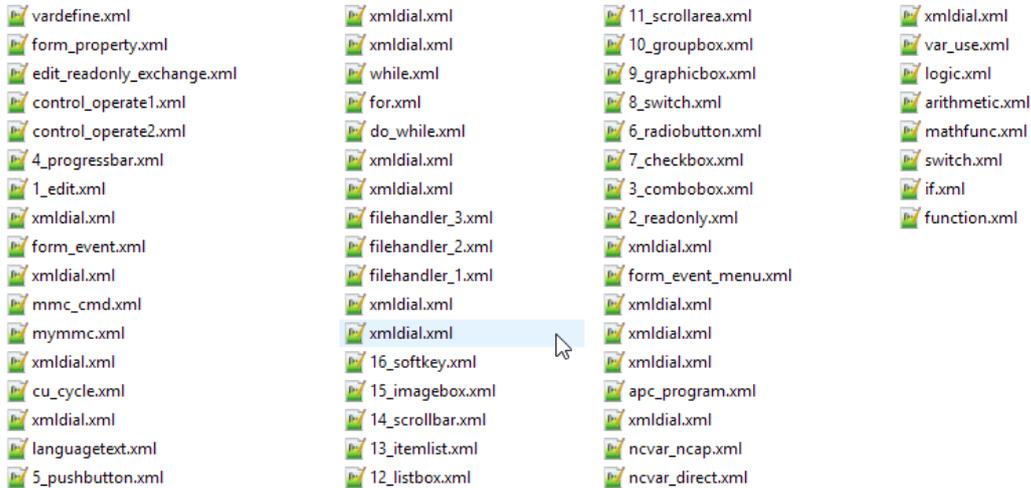
## 5.6 样例库说明

828D toolbox 中提供了 EasyXML 编程的样例库，供开发者借鉴。按系统版本分为不同的文件夹，请注意对应数控系统软件版本。以 Operate V04.08 系统版本为例，路径如下：

828D toolbox /Examples /04.08 / Custom Screen Sample



Easy XML 界面应用包中的样例库：[Easy XML Library V1.0](#)



注：本手册中所引示例均来自于 Easy XML 界面应用包中的样例库，并标注了引用路径：  
/ Easy XML Library /.../ xxx.xml。本手册和 Easy XML Library 样例库可在 CNC4YOU 网站下载。

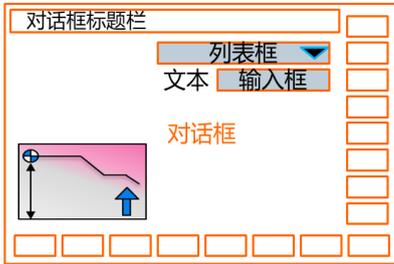
下载链接：<http://www.ad.siemens.com.cn/CNC4YOU/Home/TechCenter>



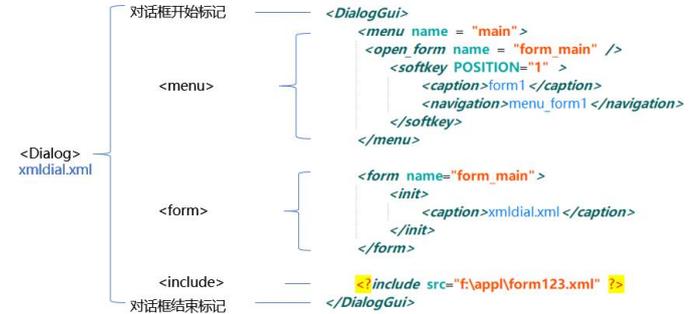
## 6 界面主体

### 6.1 对话框

对话框



代码结构



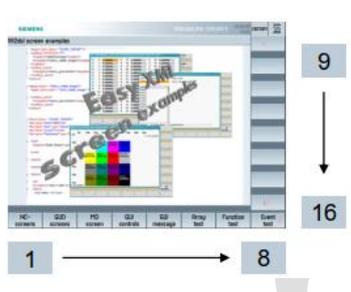
对话框属性:	对话框包含标签/支持的方法:
<ul style="list-style-type: none"> <li><i>Diagnose</i> (= "true/false") 激活诊断功能</li> <li><i>Textfile</i> (= "main_form_screens") 关联上下文的语言文本</li> </ul>	<ul style="list-style-type: none"> <li><code>&lt;menu&gt;...&lt;/menu&gt;</code> 菜单</li> <li><code>&lt;form&gt;...&lt;/form&gt;</code> 窗体</li> <li><code>&lt;?include scr="..." ?&gt;</code> 加载界面文件</li> </ul>
<p>• 示例</p> <pre>         &lt;DialogGui&gt;         &lt;menu name = "main"&gt;             &lt;open_form name = "form_main" /&gt;             &lt;softkey POSITION="1" &gt;                 &lt;caption&gt;form1 &lt;/caption&gt;             &lt;/softkey&gt;         &lt;/menu&gt;          &lt;form name="form_main"&gt;             &lt;init&gt;                 &lt;caption&gt;xmldial.xml &lt;/caption&gt;                 &lt;control name = "edit1" xpos = "022" ypos = "34" refvar="nck/Channel/Parameter/R[1]" /&gt;             &lt;/init&gt;         &lt;/form&gt;          &lt;/DialogGui&gt;     </pre>	

### 6.2 MENU

<ol style="list-style-type: none"> <li>菜单标签 <code>&lt;menu&gt;...&lt;/menu&gt;</code>, 包含在对话框 <code>&lt;Dialoggui&gt;</code> 标签中</li> <li>属性: <code>name= ""</code> 定义菜单名称</li> <li>菜单名称用于 navigation 跳转功能</li> <li>菜单中包含软键 <code>open_form</code>、<code>softkey</code> 等方法</li> <li>一个 xml 文件中可定义多个菜单</li> </ol>	<pre>         &lt;menu name = "菜单名称"&gt;             &lt;open_form name="form name" /&gt;             &lt;softkey position="1"&gt;                 ...             &lt;/softkey&gt;             ...         &lt;/menu&gt;     </pre>
---	--

menu 属性:	Menu 内的常用方法:
<ul style="list-style-type: none"> <li><code>name</code> (= "menu name")</li> <li><code>textfile</code> (= 上下文语言文本)</li> </ul>	<ul style="list-style-type: none"> <li><code>&lt;softkey&gt;...&lt;/softkey&gt;</code></li> <li><code>&lt;open_form name="form name"/&gt;</code></li> </ul>
<ul style="list-style-type: none"> <li>示例 <pre> &lt;menu name = "main" &gt;   &lt;open_form name = "form_main" /&gt;   &lt;softkey POSITION="1" &gt;     &lt;caption&gt;form1 &lt;/caption&gt;   &lt;/softkey&gt; &lt;/menu&gt; </pre> </li> </ul>	

### 6.2.1 SOFTKEY

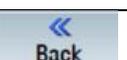
概述	
	<ol style="list-style-type: none"> <li>按键隶属于 menu</li> <li>一个 menu 有 16 个按键，位置分为水平 1-8，竖直 9-16</li> <li>按键属性: <ul style="list-style-type: none"> <li>Position</li> <li>Caption</li> <li>...</li> </ul> </li> <li>按键可以调用以下功能 <ul style="list-style-type: none"> <li>Navigation</li> <li>Function</li> <li>Message</li> <li>...</li> </ul> </li> </ol>

softkey 属性:	说明
<code>position</code>	软键的编号。1-8 水平软键，9-16 垂直软键。句法: <code>position="1"</code>
<code>type</code>	3 种软键类型。句法: <code>type="user_controlled"</code> 或 缺省 <ul style="list-style-type: none"> <li>缺省: 普通软键</li> <li><code>user_controlled</code>: 软键的显示由脚本确定，支持图片、状态切换等</li> <li><code>toggle_softkey</code>: 软键具有自锁功能</li> </ul>
<code>refvar</code>	软键关联的变量。句法: <code>refvar="var1"</code> 或 缺省 在 <code>toggle_softkey</code> 属性时返回软键状态。 返回值为 string 类型，("notpressed","pressed")。
<code>picture</code>	软键关联的图片。句法: <code>picture="f:\appl\red_led_off.bmp"</code> 或 缺省 (注: 须给出完整路径, hmi 默认盘符为 f:)
<code>picturealignment</code>	软键图片的对齐方式，句法: <code>picturealignment="center"</code> 或 缺省 <ul style="list-style-type: none"> <li>top: 上对齐</li> <li>bottom: 下对齐</li> <li>left: 左对齐 (默认)</li> </ul>

	<ul style="list-style-type: none"> <li>- right: 右对齐</li> <li>- center: 居中</li> </ul>
<i>caption</i>	<p>软键显示的文本。句法: <code>&lt;caption&gt;aabb%nsk&lt;/caption&gt;</code></p> <p>(注: 换行符为%n。显示文本支持变量格式。如显示纯图片, 则caption 标签间需编写一个空格。)</p>
<i>state</i>	<p>定义软键的显示状态。句法: <code>&lt;state type="pressed" /&gt;</code> 或 缺省</p> <ul style="list-style-type: none"> <li>- notpressed: 软键弹起时的显示状态</li> <li>- pressed: 软键按下时的显示状态</li> <li>- disable: 禁用软键的显示状态 (灰色)</li> </ul> <p>(注: 仅当 type="user_controlled" 时有效)</p>

softkey 方法	说明 (当按下软键时触发方法)
<i>open_form</i>	打开窗体。句法: <code>&lt;open_form name = "form_main" /&gt;</code>
<i>close_form</i>	关闭窗体。句法: <code>&lt;close_form /&gt;</code>
<i>function</i>	调用方法。句法: <code>&lt;function name="func_add" /&gt;</code>
<i>navigation</i>	跳转到其他菜单。句法: <code>&lt;navigation&gt;menu_form1&lt;/navigation&gt;</code>
<i>update_controls</i>	<p>更新控件。句法: <code>&lt; update_controls type="false"/&gt;</code></p> <ul style="list-style-type: none"> <li>- true – 从参考变量读取数据复制到操作单元中 (refvar -&gt; control)</li> <li>- false – 从操作单元读取数据复制到参考变量中 (control -&gt; refvar)</li> </ul>
<i>control_reset</i>	<p>脚本语言。句法: <code>&lt;control_reset resetnc="true" /&gt;</code></p> <ul style="list-style-type: none"> <li>- resetnc="true" 重新启动 nc 组件</li> <li>- resetdrive="true" 重新启动驱动组件</li> </ul>
<i>create_cycle</i>	结合 nc_instruction 指令生成用户循环。句法: <code>&lt;create_cycle refvar="cycle_block"/&gt;</code> , 详见标签用法章节。
<i>op</i>	变量运算。句法: <code>&lt;op&gt;index1 = length -index &lt;/op&gt;</code>
<i>send_message</i>	向窗体发送信息。句法: <code>&lt;send_message&gt;1,2&lt;/send_message&gt;</code>

### 6.2.2 预定义软键

预定义按键	中文	英文	默认位置	句法:
SOFTKEY_OK			<i>Position="16"</i>	<code>&lt;softkey_ok&gt;</code> ... <code>&lt;/softkey_ok&gt;</code>
SOFTKEY_CANCEL			<i>Position="15"</i>	<code>&lt;softkey_accept&gt;</code> ... <code>&lt;/softkey_accept&gt;</code>
SOFTKEY_BACK			<i>Position="16"</i>	<code>&lt;softkey_back&gt;</code> ... <code>&lt;/softkey_back&gt;</code>
SOFTKEY_ACCEPT			<i>Position="16"</i>	<code>&lt;softkey_cancel&gt;</code> ... <code>&lt;/softkey_cancel&gt;</code>

### 预定义软键说明

1. 不再有其他属性
2. 位置固定，位置冲突时，以编程顺序后者优先原则显示
3. 支持按键方法

### 6.2.3 SOFTKEY 示例

<ul style="list-style-type: none"><li>• 示例 1 <pre>&lt;softkey POSITION="1" &gt;   &lt;caption&gt;form1 &lt;/caption&gt;   &lt;navigation&gt;menu_form &lt;/navigation&gt; &lt;/softkey&gt;</pre></li></ul>	
<ul style="list-style-type: none"><li>• 示例 2: <pre>&lt;let name="define_sk_type" type="string" &gt;PRESSED &lt;/let&gt; &lt;let name="bmp_name" type="string" &gt;f:\appl\red_led_on.bmp &lt;/let&gt; &lt;softkey POSITION="2" type="user_controlled" picture="\$\$\$bmp_name" &gt;   &lt;caption&gt;Toggle%nSK &lt;/caption&gt;   &lt;if&gt;     &lt;condition&gt;sk_type == 0 &lt;/condition&gt;     &lt;then&gt;       &lt;op&gt; sk_type = 1 &lt;/op&gt;       &lt;op&gt; define_sk_type = _T"PRESSED" &lt;/op&gt;       &lt;op&gt; bmp_name = _T"f:\appl\red_led_on.bmp" &lt;/op&gt;     &lt;/then&gt;     &lt;else&gt;       &lt;op&gt; define_sk_type = _T"NOTPRESSED" &lt;/op&gt;       &lt;op&gt; bmp_name = _T"f:\appl\red_led_off.bmp" &lt;/op&gt;       &lt;op&gt; sk_type = 0 &lt;/op&gt;     &lt;/else&gt;   &lt;/if&gt;   &lt;state type="\$\$\$define_sk_type" /&gt;   &lt;print text="%s" &gt; define_sk_type &lt;/print&gt; &lt;/softkey&gt;</pre></li></ul>	<p>抬起后</p>  <p>按下时</p> 
<ul style="list-style-type: none"><li>• 示例 3: 纯图片显示，图片像素大小不匹配时无法充满整个软键;</li><li>• 充满软键的示例像素: 216x96 px, 效果见 position="10"</li></ul> <pre>&lt;menu name="menu_if_jog" &gt;   &lt;open_form name = "form_menu_if_jog" /&gt;    &lt;softkey POSITION="10" picture="f:/appl/sk_cancel1.png" picturealignment="center" &gt;     &lt;caption&gt; &lt;/caption&gt;   &lt;/softkey&gt;    &lt;softkey POSITION="11" picture="step_tog_page_up.png" picturealignment="center" &gt;     &lt;caption&gt; &lt;/caption&gt;   &lt;/softkey&gt;    &lt;softkey POSITION="12" picture="sk_sim_start.png" picturealignment="center" &gt;     &lt;caption&gt; &lt;/caption&gt;   &lt;/softkey&gt;    &lt;softkey POSITION="13" picture="sk_sim_reset.png" picturealignment="center" &gt;     &lt;caption&gt; &lt;/caption&gt;   &lt;/softkey&gt;    &lt;softkey_cancel&gt;     &lt;send_message&gt;1,2 &lt;/send_message&gt;   &lt;/softkey_cancel&gt;    &lt;softkey_back&gt;     &lt;navigation&gt;menu_sktest &lt;/navigation&gt;   &lt;/softkey_back&gt; &lt;/menu&gt;</pre>	

参见: /Easy XML Library / 3\_控件展示 / 16\_softkey.xml

## 6.2.4 RECALL

特殊标签，回调用键。

语法：

```
<recall>  
...  
</recall>
```

示例：

```
<recall>  
<navigation>main </navigation>  
</recall>
```



PPU 回调用键见下图



## 6.3 FORM

本章节示例参考 [/ Easy XML Library / 1\\_form 介绍](#) 文件夹中的项目

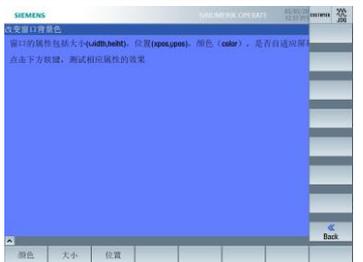
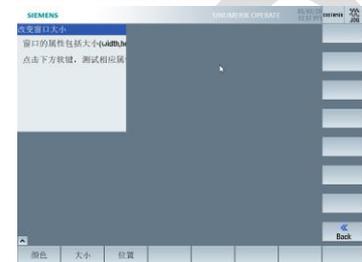
### 6.3.1 窗体属性

窗体属性包括名称 (name)，位置 (xpos, ypos)，宽高 (width, height) 等。详见下表

form 属性：	说明
<i>name</i>	窗体名称，整个项目中名称唯一。句法： <i>name</i> ="form1"
<i>type</i>	允许的值是 cycle，表示一个用户循环窗口。句法： <i>type</i> ="cycle" 或 缺省
<i>xpos</i>	窗体左上角的 X 坐标 (可选)。句法： <i>xpos</i> ="100" 或 缺省
<i>ypos</i>	窗体左上角的 Y 坐标 (可选)。句法： <i>ypos</i> ="100" 或 缺省
<i>width</i>	X 方向上的长度 (单位：像素) (可选)。句法： <i>width</i> ="100" 或 缺省
<i>height</i>	Y 方向上的长度 (单位：像素) (可选)。句法： <i>height</i> ="100" 或 缺省
<i>autoscale_content</i>	自适应屏幕分辨率设置。句法： <i>autoscale_content</i> ="off" 或 缺省 <ul style="list-style-type: none"><li>- On: 自动适应 (缺省)</li><li>- Off: 控件坐标保持原值不变 (基于 600x480)</li></ul>
<i>color</i>	窗体背景色。句法： <i>color</i> ="#RRGGBB"

示例:



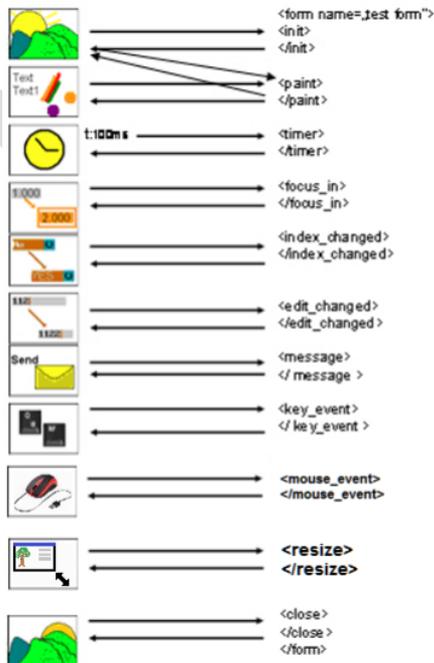
颜色	大小	位置
<code>&lt;form name="form4" color="#637DFF"&gt;</code>	<code>&lt;form name="form2" width="200" height="200"&gt;</code>	<code>&lt;form name="form3" xpos="100" ypos="100"&gt;</code>
		

参见: /Easy XML Library/ 3\_控件展示 / form\_property.xml

### 6.3.2 窗体事件说明

当进行相关操作时, 会触发窗体中的对应事件。主要事件及其标签如下:

Init, paint, focus\_in, index\_changed, message, key\_event, resize 等, 详见 [窗体事件](#) 章节。



示例:

```

<form name="value_changed_call_form" >
  <init>
    <caption>Value changed handler (file: value_changed.xml) </caption>
    <data_access type="true" />

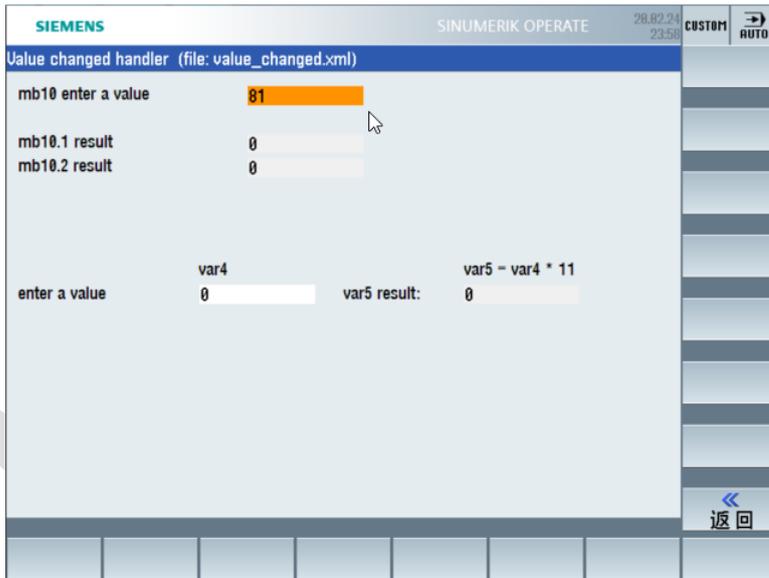
    <REQUEST name = "plc/m10.1" function="event_changed_plc_mb10_bit1" />
    <REQUEST name = "plc/m10.2" function="event_changed_plc_mb10_bit2" />
    <REQUEST name = "var4" function="event_changed_local_var" />

    <control name = "c1" xpos = "200" ypos = "34" refvar="plc/mb10" hotlink="true" />
    <control name = "c2" xpos = "200" ypos = "74" refvar="var2" hotlink="true" fieldtype="readonly" color_bk="#f0f0f0" />
    <control name = "c3" xpos = "200" ypos = "94" refvar="var3" hotlink="true" fieldtype="readonly" color_bk="#f0f0f0" />

    <control name = "c4" xpos = "160" ypos = "200" refvar="var4" hotlink="true" />
    <control name = "s5" xpos = "380" ypos = "200" refvar="var5" fieldtype="readonly" hotlink="true" color_bk="#f0f0f0" />
  </init>

  <paint>
    <text xpos="10" ypos="34">mb10 enter a value </text>
    <text xpos="10" ypos="74">mb10.1 result </text>
    <text xpos="10" ypos="94">mb10.2 result </text>
    <text xpos="10" ypos="200">enter a value </text>
    <text xpos="160" ypos="180">var4 </text>
    <text xpos="380" ypos="180">var5 = var4 * 11 </text>
    <text xpos="280" ypos="200">var5 result: </text>
  </paint>
</form>

```



#### 6.4 FUNCTION\_BODY

在<menu><form>标签外定义的函数体标签，在当前的 xml 文件被加载后有效，可以被其他 xml 文件中的事件调用。

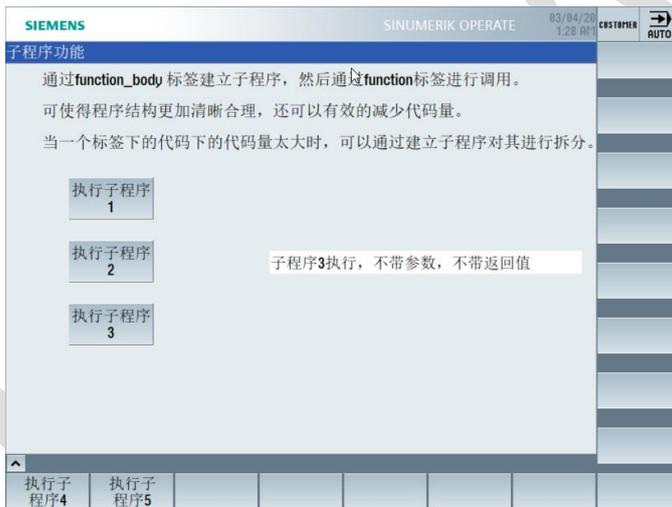
Function_body	<p>使用该标签定义一个子函数。具有如下属性</p> <ul style="list-style-type: none"> <li>- Name: 函数体的名称; 句法: name="function_body_name"</li> <li>- Parameter: 参数列表, 参数之间用逗号分开 (可选), 最多可以传输 10 个参数。句法: name="p1,p2,p3,...,p10"</li> <li>- Return: 是否有返回值 (可选); 句法: return="true" 或 "false"、</li> </ul>
---------------	---

语法:

无参数的函数体	定义	<code>&lt;function_body name = "f_function1" &gt; &lt;op&gt;pa[0]=pa[0]+1 &lt;/op&gt; &lt;/function_body&gt;</code>
	调用	<code>&lt;function name="f_function1"/&gt;</code>
带参数的函数体	定义	<code>&lt;function_body name = "f_function2" parameter="p1,p2" &gt; &lt;op&gt;pa[1]=p1+p2 &lt;/op&gt; &lt;/function_body&gt;</code>
	调用	<code>&lt;function name="f_function2" &gt;pa[0],pa[1] &lt;/function&gt;</code>
带返回值的函数体	定义	<code>&lt;function_body name = "f_function3" parameter="p1,p2" return="true" &gt; &lt;op&gt;\$return=p1+p2 &lt;/op&gt; &lt;/function_body&gt;</code>
	调用	<code>&lt;function name="f_function3" return="pa[2]" &gt;pa[0],pa[1] &lt;/function&gt;</code>

传递参数时，如果参数类型为字符串常量，则应该在其前放置标志\_T，变量无需放置，如：

`<FUNCTION name = "<function name>"> var1, var2, var3 </FUNCTION>`  
`<FUNCTION name = "<function name>"> _T"Text", 1.0, 1 </FUNCTION>`



参见: [/Easy XML Library/5\\_程序结构 / 1\\_子程序 / function.xml](#)

## 6.5 变量定义

本章节示例参考 [/Easy XML Library/2\\_定义变量](#) 文件夹中的项目

### 6.5.1 LET

使用<let>标签定义局部界面变量。

在标签内定义的变量，只在此标签内有效，直到标签关闭。

**注:** 这里只按照标签定义，与文件无关。< DialogGui >标签范围可以包含多个 XML 文件，在不同文件内定义、使用局部变量时，清晰有条理，切勿重名，混淆。

在函数体、窗体事件、逻辑循环（如<function\_body> <message>等）内定义的变量，只在标签内有效。需注意 menu 和 form 分属两个标签，如需使用同一变量，需要在对话框< DialogGui >标签定义。

句法
<p>单个变量: <code>&lt;let name="变量名称" type="变量类型" &gt;初始值&lt;/let&gt;</code> 或 <code>&lt;let name="变量名称" type="变量类型" /&gt;</code></p> <p>一维数组变量: <code>&lt;let name="变量名称" type="变量类型" dim="元素数量"&gt;{初始值 1,2,3}&lt;/let&gt;</code></p> <p>二维数组变量: <code>&lt;let name="变量名称" type="变量类型" dim="行数, 列数"&gt;{初始值 1,2,3},{初始值 1,2,3}..&lt;/let&gt;</code></p> <p>数组元素索引: <code>变量名称 [index]</code>, <code>变量名称 [row, column]</code></p> <p>结构变量访问方式: <code>结构名称.元素名称</code></p>
说明
<ul style="list-style-type: none"> <li>未赋初值的元素默认为"0"。</li> <li>未定义类型的变量, 会依据变量赋值等操作自行转换类型。</li> <li>需要全局使用的变量应直接在标签 DialogGui 后创建。</li> <li>标签中的所有变量会随着关闭而被清零。</li> </ul>

Let 属性	说明
<i>Name</i>	变量名称
<i>Type</i>	变量类型: <ul style="list-style-type: none"> <li>- 整数型 (INT)</li> <li>- 无符号整数型 (UINT)</li> <li>- 双精度型 (DOUBLE)</li> <li>- 浮点型 (FLOAT)</li> <li>- 字符串型 (STRING)</li> <li>- 结构体型 (STRUCT) ; 需要使用 typedef 指令定义结构体, 结构中的变量元素通过标签<code>&lt;element name="名称" type="变量类型" /&gt;</code> 来说明。</li> </ul>
<i>Permanent</i>	如果属性为 true, 则永久保存变量值。该属性仅对全局变量有效。
<i>Dim</i>	用来指定数组元素的数量。

示例:

```

</let name="itemindex" type="int" ></let>
</let name="boxaddfile" type="string" ></let>
<let name="menu_name" type="string" >main </let>
<let name="script_loaded" >0 </let>
</let name="prog_name" type="string" />
</let name="contents" type="string" />
</let name="ax_rev_array" dim="6" >5,4,3,2,1,0 </let>

```

## 定义数组变量

```
<let name="frame_name" dim="7" type="string" >
"G500",
"G54",
"G55",
"G56",
"G57",
"G58",
"G59",
</let>

<let name="list" dim="10,3" >
{1,2,3},
{1,20}
</let>

<let name="list_string" dim="10, 3" type="string" >
{"text 10","text 11","abc"},
{"text 20","text 21"}
</let>
```

## 定义结构体

```
<typedef name="StructRect" type="struct" >
<element name="left" type="int" >0 </element>
<element name="top" type="int" >0 </element>
<element name="right" type="int" >80 </element>
<element name="bottom" type="int" >80 </element>
</typedef>
```

## 定义结构体变量

```
<let name="rect1" type="struct" >
<element name="left" type="int" >0 </element>
<element name="top" type="int" >0 </element>
<element name="right" type="int" >180 </element>
<element name="bottom" type="int" >180 </element>
</let>

<let name="rect" type="StructRect" />

<let name="rect_array1" type="StructRect" dim="2" ></let>

<let name="rect_array2" type="StructRect" dim="2, 2" ></let>
```

## 结构体变量的使用

```
<control name = "a1" xpos = "10" ypos = "54" width="60" refvar="rect.left" hotlink="true" />
<control name = "a2" xpos = "10" ypos = "74" width="60" refvar="rect.top" hotlink="true" />
<control name = "a3" xpos = "10" ypos = "94" width="60" refvar="rect.bottom" hotlink="true" />
<control name = "a4" xpos = "10" ypos = "114" width="60" refvar="rect.right" hotlink="true" />

<control name = "b1" xpos = "200" ypos = "54" width="60" refvar="rect1.left" hotlink="true" />
<control name = "b2" xpos = "200" ypos = "74" width="60" refvar="rect1.top" hotlink="true" />
<control name = "b3" xpos = "200" ypos = "94" width="60" refvar="rect1.bottom" hotlink="true" />
<control name = "b4" xpos = "200" ypos = "114" width="60" refvar="rect1.right" hotlink="true" />

<control name = "c1" xpos = "10" ypos = "254" width="60" refvar="rect_array1[1].left" hotlink="true" />
<control name = "c2" xpos = "10" ypos = "274" width="60" refvar="rect_array1[1].top" hotlink="true" />
<control name = "c3" xpos = "10" ypos = "294" width="60" refvar="rect_array1[1].bottom" hotlink="true" />
<control name = "c4" xpos = "10" ypos = "314" width="60" refvar="rect_array1[1].right" hotlink="true" />

<control name = "d1" xpos = "200" ypos = "254" width="60" refvar="rect_array2[0,1].left" hotlink="true" />
<control name = "d2" xpos = "200" ypos = "274" width="60" refvar="rect_array2[0,1].top" hotlink="true" />
<control name = "d3" xpos = "200" ypos = "294" width="60" refvar="rect_array2[0,1].bottom" hotlink="true" />
<control name = "d4" xpos = "200" ypos = "314" width="60" refvar="rect_array2[0,1].right" hotlink="true" />
```

参见：[/Easy XML Library/2\\_定义变量/var\\_use.xml](#)

## 6.5.2 TYPEDEF

通过标签可定义结构体数据类型。可以将多个数据类型重新组合出一种新的数据类型，且支持 let 指令定义该类型变量。类型定义中的多个元素通过 < element > 标签进行声明。元素的类型属性为基本的数据类型 (int, string, double... )。

语法：

```
<typedef name="<名称>" type="struct">
  <element name="<名称>" type="<变量类型>" />
  ...
</typedef>
```

属性：

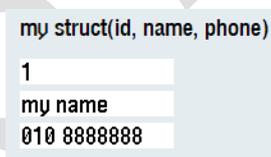
Name: 新的数据类型的名称

Type="struct": 结构体类型, 即有多个元素组成

Element: 定义元素的名称 (name) 和类型 (type)

示例:

```
<typedef name="my_struct" type="struct" >
  <element name="id" type="int" />
  <element name="name" type="string" />
  <element name="phone" type="string" />
</typedef>
<let name="info" type="my_struct" ></let>
<control name = "e1" xpos = "400" ypos = "54" width="100" refvar="info.id" hotlink="true" />
<control name = "e2" xpos = "400" ypos = "74" width="100" refvar="info.name" hotlink="true" />
<control name = "e3" xpos = "400" ypos = "94" width="100" refvar="info.phone" hotlink="true" />
<op>
  rect_array1_left = rect_array1[1].left;
  info.id = 1;
  info.name = _T"my name";
  info.phone = _T"010 8888888";
</op>
```



参见: /Easy XML Library/2\_定义变量 / var\_use.xml , vardefine.xml

## 6.6 系统变量定址

### 6.6.1 NC 变量

NC 变量的定址从路径 nck 开始。各变量名称可查阅 NC 变量和接口信号的参数手册。

通道定址：在地址中添加标识 u (Unit) 和通道号<如：u2-通道 2>；缺省则为 1 通道

示例：

```
nck/Channel/MachineAxis/actFeedRate[3]
nck/Channel/MachineAxis/actFeedRate[u1, 3]
```

示例：

```
<LET name = "tempStatus"></LET>
<OP> tempStatus = "nck/channel/state/chanstatus" </OP>
```

使用 data 标签对机床参数或 NC 变量赋值

对于多个索引的参数时，格式为[索引号， <轴名称> 或 u 和通道号]

```
<DATA name = "$MN_AXCONF_MACHAX_NAME_TAB[0]">X1</DATA>
<DATA name = "$MA_CTRLOUT_MODULE_NR[0, AX1]">1</DATA>
<DATA name = "$MC_AXCONF_GEOAX_ASSIGN_TAB[0, u1]">X1</DATA>
```

间接地址：

```
<LET name = "axisIndex"> 1 </LET>
<DATA name = "$MA_CTRLOUT_MODULE_NR[0, AX$axisIndex]">1</DATA>
```

R 参数

"nck/Channel/Parameter/R[0]" --指示的为用户数据 R0 参数

"nck/Channel/Parameter/rpa[u1, 1,5]" – 代表了一个数组：通道 1：起始 R1，结束 R5

### 6.6.2 PLC 变量

PLC 地址前加 "plc/"

示例：

```
<data name = "plc/mb170">1</data>
<data name = "plc/i0.1"> 1 </data>
<op> "plc/m19.2" = 1 </op>
```

间接地址示例：

- 变量名称写在引号内。
- 在变量名称前标记三个，\$' 符号。

```
<PRINT name="var_adr" text="DB9000.DBW%d"> 2000</PRINT>
<OP> "$$$var_adr" = 1 </OP>
```

### 6.6.3 驱动参数

驱动组件的定址从路径 drive 开始。

CU – Control Unit (控制单元)

DC – Drive Control (电机模块)

CULNK – 扩展模块 (HUBs)

TM – Terminal-Module (端子模块)

LM – Line-Module (电源模块)

示例:

```

<LET name="r0002_content"></LET>
<LET name="p107_content"></LET>
<!-- 读取 CU 上 r0002 的值 -->
<OP> r0002_content = "drive/cu/r0002" </OP>
<OP> r0002_content = "drive/cu/r0002[CU1]" </OP>
<!-- 读取 NX1 上 r0002 的值 -->
<OP> r0002_content = "drive/cu/r0002[CU2]" </OP>
<!-- 读取 CU 上 p107[0] 的值 -->
<OP> p107_content = "drive/cu/p107[0]" </OP>
<PRINT text="%d"> p107_content </PRINT>
<!-- 读取 CU 上 p107[0] 的值 -->
<OP> p107_content = "drive/cu/p107[0, CU1]" </OP>
<PRINT text="%d"> p107_content </PRINT>
<!-- 读取 NX1 上 p107[0] 的值 -->
<OP> p107_content = "drive/cu/p107[0, CU2]" </OP>
<PRINT text="%d"> p107_content </PRINT>

```

间接地址分配示例:

```

<LET name = "driveIndex" 0 </LET>
<OP> driveIndex = $ctrlout_module_nr[0, AX1] </OP>
<DATA name = "drive/dc[do$driveIndex]/p0092">1</DATA>

```

#### 6.6.4 GUD 变量

类别	说明	示例
sgud	西门子 GUD	refvar="gud/channel/gd1/_ZSD[0]"
mgud	机床制造商 GUD	refvar="gud/channel/mgud/_M_VAR1" (通道 1) refvar="gud/channel/mgud/_M_VAR1[u2]" (通道 2)
gud4 至 gud9	用户 GUD 扩展	refvar="gud/channel/gd4/_A_VAR2" (通道 1) refvar="gud/nck/gd4/_A_VAR1" (NCK 全局变量)

对于通道专用 GUD 而言:

- 必须在行/列指示前或后给定通道编号, 形式为跟随有编号的标识 u (Unit)。
- 必须用逗号将这些序列相互分隔 (数组变量)。
- 在未给定通道编号的情况下, 将会访问第一个通道。

示例:

```

<data name = "gud/channel/mgud/syg_rm[0]">1</data>
<op>"gud/channel/mgud/syg_rm[0]" = 5*2 </op>

```

```
"gud/Channel/sgud/_WP[u2, 2.0]"
```

或

```
"gud/Channel/sgud/_WP[2.0, u2]"
```

## 7 窗体事件

本章节示例参考 / Easy XML Library / 1\_form 介绍 文件夹中的项目

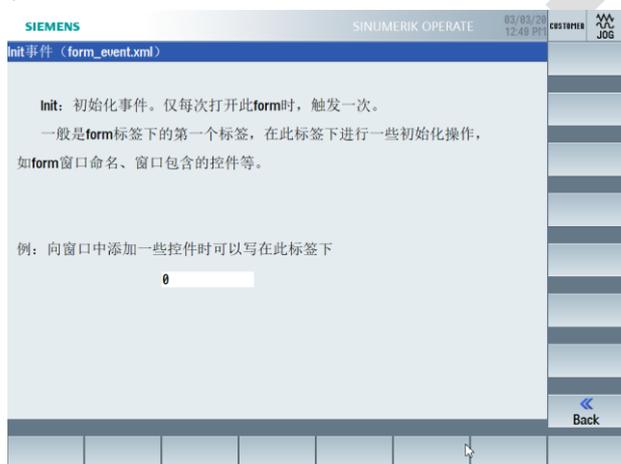
### 7.1 INIT

在窗体创建后, 调用该事件。

创建界面的操作控件 (输入框等)。根据代码书写的先后顺序创建控件。

示例:

```
<form name="init_form">
  <init>                                <!--Init 事件-->
    <caption>Init 事件</caption>         <!--form 命名-->
    <control name="con_1" xpos="60" ypos="200" fieldtype="edit" />
    <control name="con_2" xpos="60" ypos="230" fieldtype="readonly" /><!--添加一些控件-->
  </init>
</form>
```



参见: / Easy XML Library / 1\_form 介绍 / form\_event.xml

注: 在时序上需要先进行 init 初始化, 再对控件进行 timer 标签或属性更改等处理。

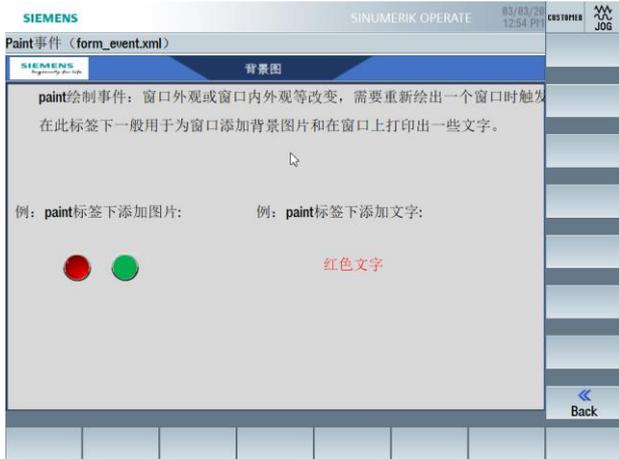
### 7.2 PAINT

显示或重新显示窗体时, 调用该事件。

可绘制文本, 图片。

示例:

```
<paint>                                <!--Paint 事件-->
  <img xpos = "0" ypos = "0" AspectRatioMode="Ignore" width="560" height="395" name =
    "f:/appl/paint_bkg.png" />         <!--添加背景图-->
  <text xpos = "35" ypos = "60">paint 绘制事件: 即窗口外观或窗口内外观等改变, 需要重新绘出一个窗
    口时触发</text>                     <!--添加文字-->
  <text xpos = "35" ypos = "90">在此标签下一般用于为窗口添加背景图片和在窗口上打印出一些文字。
</text>
</paint>
```



参见: /Easy XML Library/ 1\_form 介绍 / form\_event.xml

注: 显示内容与 timer 标签冲突时, 无法正常显示。

### 7.3 TIMER

当前显示的窗体, 系统会每 100ms 调用一次该事件。

每个 form 都分配了一个 timer, 每 100ms 会执行一次 timer 标签。

示例:

```
<timer>                                <!--Timer 事件-->
  <op>var_1=var_1+1</op>
</timer>
```



参见: /Easy XML Library/ 1\_form 介绍 / form\_event.xml

### 7.4 FOCUS\_IN

在界面切换输入焦点时, 系统调用该事件。

系统将控件的名称复制到界面变量\$focus\_name中, 将控件属性 item\_data 的值复制到变量 \$focus\_item\_data 中。

示例:

```
<init>
  <caption>FocusIn 事件</caption>
  <control name="edit_1" xpos="20" ypos="190" height="25" width="100" fieldtype="edit"
  item_data="1"/>
```

```

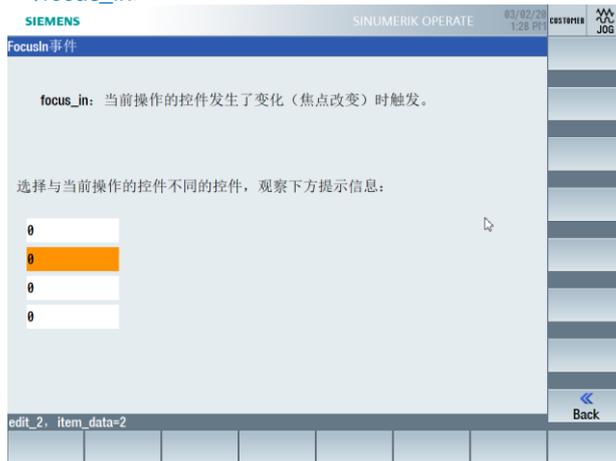
<control name="edit_2" xpos="20" ypos="220" height="25" width="100" fieldtype="edit"
item_data="2"/>
<control name="edit_3" xpos="20" ypos="250" height="25" width="100" fieldtype="edit"
item_data="3"/>
<control name="edit_4" xpos="20" ypos="280" height="25" width="100" fieldtype="edit"
item_data="4"/>
</init>

```

```
<focus_in> <!--focus_in 事件-->
```

```
<print text="%s, item_data=%d">$focus_name, $focus_item_data</print>
```

```
</focus_in>
```



参见: [/Easy XML Library/ 1\\_form 介绍 / form\\_event.xml](#)

注:

1. 如未编辑控件的 item\_data 属性，item\_data 默认为“-1”。item\_data 为 INT 类型，数值可由用户自由定义。
2. Switch 逻辑中的 Case 比较仅支持数值，不支持字符串比较，如要使用 \$focus\_name 作为 focus\_in 的判断条件，则无法使用 switch 方法。

## 7.5 INDEX\_CHANGED

当列表框控件 (fieldtype="combobox") 内容发生变化时，调用该事件。

同时系统将 control 的名称复制到界面变量 \$focus\_name 中，将属性 item\_data 的值复制到变量 \$focus\_item\_data 中。

可利用此条件锁定当前更改的 combobox 控件。

示例:

```
<init>
```

```
<caption>Index_Change 事件</caption>
```

```
<control name="combobox_1" xpos="20" ypos="200" height="25" width="100" fieldtype="combobox"
color_bk="#ffffff" refvar="var_1" hotlink="true">
```

```
<ITEM value="1">text1</ITEM>
```

```
<ITEM value="2">text2</ITEM>
```

```
<ITEM value="3">text3</ITEM>
```

```
<ITEM value="4">text4</ITEM>
```

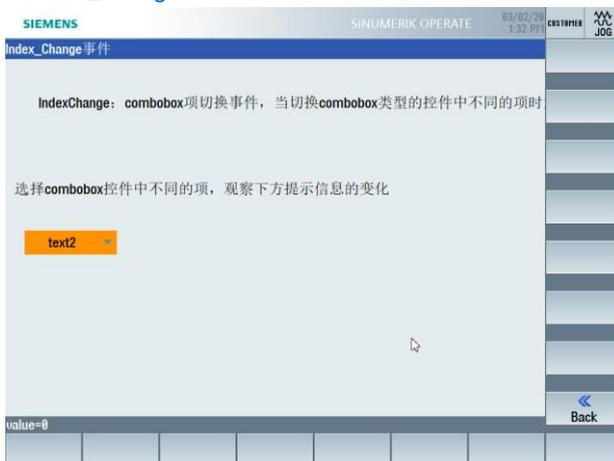
```
</control>
```

```
</init>
```

```

<index_changed>      <!--IndexChange 事件-->
  <print text="value=%d">var_1</print>
</index_changed>

```



参见: /Easy XML Library/ 1\_form 介绍 / form\_event.xml

## 7.6 EDIT\_CHANGED

当 edit 控件 (fieldtype="edit") 内容发生变化时, 调用该事件。

同时系统将 control 的名称复制到界面变量 \$focus\_name 中, 将属性 item\_data 的值复制到变量 \$focus\_item\_data 中。

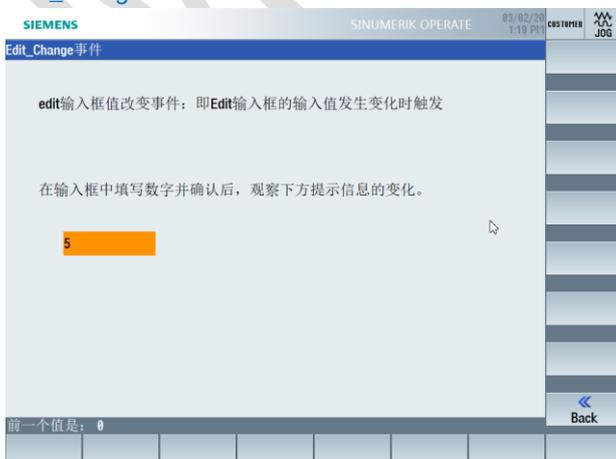
可利用此条件锁定当前更改的 edit 控件。

示例:

```

<let name="var_1" type="int"></let>
<init>
  <caption>Edit_Change 事件</caption>
  <control name="edit_1" xpos="60" ypos="200" height="25" width="100" fieldtype="edit"
    refvar="var_1" hotlink="true" />
</init>
<edit_changed>      <!--edit_changed 事件-->
  <print text="前一个值是: %d">var_1</print>
</edit_changed>

```



参见: /Easy XML Library/ 1\_form 介绍 / form\_event.xml

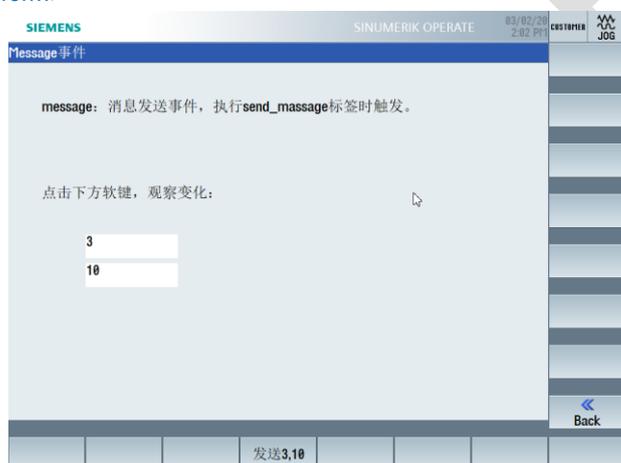
## 7.7 MESSAGE

当在脚本中执行了 `send_message` 指令时, 调用该事件。

同时 `send_message` 传递的两个参数 `p1,p2` 会复制到变量 `$message_par1` 和 `$message_par2`。

示例:

```
<softkey position="4">
  <caption>发送 3, 10</caption>
  <send_message>3, 10</send_message>
</softkey>
...
<form name="message_form">
  <MESSAGE>
    <op>var_1=$message_par1</op>
    <op>var_2=$message_par2</op>
  </MESSAGE>
</form>
```



参见: [/Easy XML Library/ 1\\_form 介绍 / form\\_event.xml](#)

## 7.8 KEY\_EVENT

在窗体中有键盘动作时, 调用该事件。

标签 `KEY_EVENT` 可将对键盘事件的评估链接到标签 `FORM` 中。如果在 `FORM` 中存在该标签, 则系统会将 MF2 键盘代码发送给有效的 `FORM`。

键盘代码会作为整型值在变量 `$keycode` 中提供。

如果变量 `$actionresult` 未置零, 系统会接着处理键盘事件。

示例:

```
<key_event>          <!--KeyEvent 事件-->
  <print text="键盘码为: %d">$keycode</print>
</key_event>
```



参见: /Easy XML Library/ 1\_form 介绍 / form\_event.xml

## 7.9 MOUSE\_EVENT

具有如下鼠标操作时, 调用该事件。

- 鼠标某个按键被按下
- 鼠标按键被松开
- 移动鼠标

解析器在结构中提供信息并创建带有以下元素的结构变量\$mouse\_event:

*\$mouse\_event.button*

- 0: 无按键
- 1: 左键
- 2: 右键
- 3: 中键

*\$mouse\_event.type*

- 2: 按下按键
- 3: 松开按键
- 5: 移动鼠标

*\$mouse\_event.id*

- -1, 该位置没有分配控件时
- != -1, 鼠标光标位于控件内时

*\$mouse\_event.x*

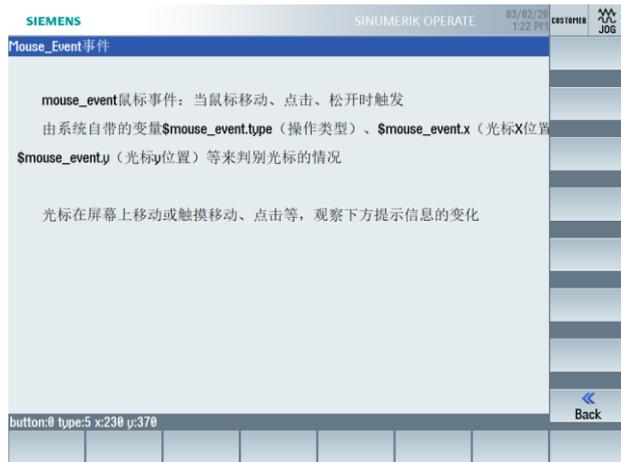
- 鼠标所在的 X 坐标位置, 单位: 像素

*\$mouse\_event.y*

- 鼠标所在的 Y 坐标位置, 单位: 像素

示例:

```
<!--mouse_event 事件-->
<print text="button:%d type:%d x:%d y:%d ">$mouse_event.button, $mouse_event.type,
  $mouse_event.x, $mouse_event.y</print>
</mouse_event>
```



参见: /Easy XML Library/ 1\_form 介绍 / form\_event.xml

## 7.10 GESTURE\_EVENT

*多点触控操作时，调用该事件。执行多点触控操作中的手势。*

轻击 (Tap) 或滑动 (Flick) 手势，会在系统变量 \$gestureinfo 中提供手势信息，该变量为结构变量，有如下子元素: type, flag, state, item\_data, point, delta

[\\$gestureinfo.type](#)

- 3: 移动手势
- 4: 缩小手势
- 5: 轻击手势

[\\$gestureinfo.flag](#)

- 1: 缩放系数已改变
- 2: 旋转角已改变

[\\$gestureinfo.state](#)

- 1: 已启动
- 2: 已更新
- 3: 已结束

[\\$gestureinfo.item\\_data](#)

- N: 生效控制项的 item\_data 值
- -1: 手势没有对应的控制项
- != -1: 手势已在控制项中执行，在创建时已使这个值与该控制项对应

[\\$gestureinfo.point.x](#)

- 手势的 X 坐标位置

[\\$gestureinfo.point.y](#)

- 手势的 Y 坐标位置

[\\$gestureinfo.delta](#)

- 手势的开始与当前事件之间的缩放差或旋转角度差

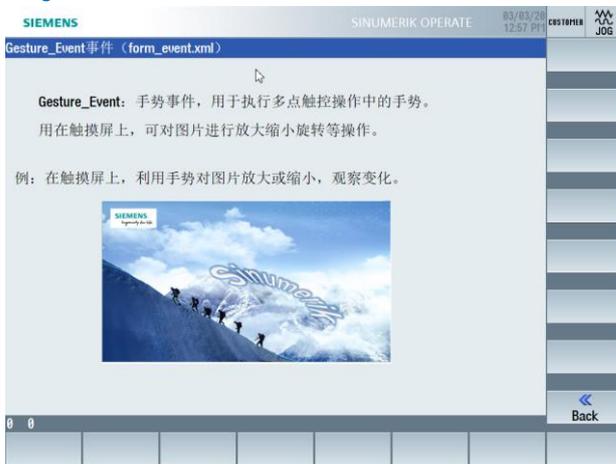
示例

```
<gesture_event>
  <print text="%d %d">$message_par1,$message_par2</print>
  <switch>
    <condition>$message_par2</condition>
```

```

        <case value="-1">
<print text="%s">向左滚动</print>
        </case>
        <case value="1">
<print text="%s">向右滚动</print>
        </case>
        <case value="-2">
<print text="%s">向上滚动</print>
        </case>
        <case value="2">
<print text="%s">向下滚动</print>
        </case>
</switch>
</gesture_event>

```



参见: [/Easy XML Library/ 1\\_form 介绍 / form\\_event.xml](#)

## 7.11 RESIZE

*界面分辨率变化时, 调用该事件。*

适用环境: sidescreen 屏幕展开/合并时; 多屏幕 (分辨率不同) 切换时。

示例:

```

<typedef name="hmi_size" type="struct" >
  <element name="width" type="int">0</element>
  <element name="height" type="int">0</element>
</typedef>
<let name="screen_size" type="hmi_size" />
...
<resize>
  <!--resize 事件-->
  <let name="result" />
  <MSGBOX text="窗口大小改变, resize 事件触发" caption="提示信息" retvalue="result" type="BTN_OK" />
</resize>
...
<timer>
  <!--利用 timer 监控一下分辨率的变化-->
  <function name="hmi.get_hmi_resolution">screen_size</function>

```

</timer>

...



参见: /Easy XML Library/ 1\_form 介绍 / form\_event.xml

## 7.12 CLOSE

对话框关闭前, 调用该事件。句法: <close>.. </close>

示例:

```
<softkey position="16">  
  <caption>关闭并返回</caption>  
  <navigation>main</navigation>  
</softkey>  
<close> <!--close 事件-->  
  <MSGBOX text="这个窗口即将关闭" caption="信息" retvalue="result" type="BTN_OK" />  
</close>
```



参见: /Easy XML Library/ 1\_form 介绍 / form\_event.xml

## 8 窗体控件 (GUI)

本章节示例参考 / Easy XML Library / 3\_控件展示 文件夹中的项目

### 8.1 CONTROL 介绍

窗体中所显示的文本，列表框，输入框，图片等元素都是由定义好的界面变量实现的。Control 控件有多种类型，类型由 fieldtype 定义，不同类型的属性略有差异。

语法：

```
<CONTROL name = "<控件名称>" xpos = "<X 坐标>" ypos = "<Y 坐标>" refvar = "<关联变量>"  
fieldtype= "<控件类型>" hotlink = "true" format = "<格式>" />
```

或：

或：<带有附加属性>

```
<CONTROL name = "<控件名称>" xpos = "<X 坐标>" ypos = "<Y 坐标>" refvar = "<关联变量>"  
fieldtype= "<控件类型>" hotlink = "true" format = "<格式>" >  
  <property 属性1 = "<值>" />  
  <property 属性2 = "<值>" />  
  ...  
</CONTROL>
```

#### 8.1.1 主属性

control 属性：	说明
<i>name</i>	<i>控件名称</i> 。句法： <i>name="form1"</i> 在 form 中名称唯一，该名称同时代表一个局部变量，不需要使用 let 指令定义该变量。
<i>xpos</i>	控件左上角的 X 坐标。句法： <i>xpos="100"</i>
<i>ypos</i>	控件左上角的 Y 坐标。句法： <i>ypos="100"</i>
<i>width</i>	X 方向上的长度（单位：像素）（可选）。句法： <i>width="100" 或缺省</i>
<i>height</i>	Y 方向上的长度（单位：像素）（可选）。句法： <i>height="100" 或缺省</i>
<i>fieldtype</i>	控件类型。句法： <i>fieldtype="edit"</i> <ul style="list-style-type: none"><li>- Edit：编辑输入框</li><li>- Readonly：只读状态框（支持显示多行文本）</li><li>- Combobox：下拉框</li><li>- Listbox：列表框</li><li>- Progressbar：进度条</li><li>- Graphicbox：二维图形控件</li><li>- Pushbutton：触摸屏按钮</li><li>- Radiobutton：单选框</li><li>- Checkbox：复选框</li><li>- Groupbox：组合框</li><li>- Scrollarea：滚动区域</li></ul>
<i>item_data</i>	用户可使用整数值为该属性赋值，缺省值：-1。句法： <i>item_data="10"</i> 该值被提供给 focus_in 等用于识别焦点字段。
<i>refvar</i>	控件关联的参考变量（可选）。句法： <i>refvar="var1" 或缺省</i>

<i>Hotlink</i>	如果参考变量的值发生了变化, 该字段会自动进行更新 (可选)。句法: <i>hotlink = "true" 或缺省 (false)</i>
<i>Format</i>	定义了指定变量的显示格式 (可选)。
<i>Display_format</i>	定义了指定变量的处理格式 (可选)。 <ul style="list-style-type: none"> <li>- Float: 浮点型</li> <li>- Int: 整型</li> <li>- Double: 双整型</li> <li>- String: 字符串</li> <li>- <i>注</i>: 在存取 PLC 浮点型变量时必须使用该属性, 因为要通过双字读取来进行存取。</li> </ul>
<i>Item</i>	向列表框、图形框或组合框分配显示项内容。句法: <code>&lt;item&gt;text1&lt;/item&gt;</code> <code>&lt;item value = "10"&gt;item1:abcde&lt;/item&gt;</code>
<i>font</i>	指定字体。句法: <i>font = "2" 或缺省(1)</i>

### 8.1.2 附加属性

使用 `property` 标签可以为控件添加附加属性。该标签为单标签, 嵌入在控件标签中。

附加属性	说明
<i>Min</i>	极小值
<i>Max</i>	极大值
<i>Default</i>	预设值。与 <code>refvar</code> 属性不兼容。
<i>factor</i>	换算系数。可被整除的最小单位值。 <i>例如: 输入框的 factor="3"; 当输入 2.002 时, 系统自动换算为 2.001。</i>
<i>disable</i>	该属性禁止/允许在 EDIT 控件中输入。
<i>tooltip</i>	当鼠标光标移动到控件上时, 浮动显示提示文本。
<i>access level</i>	访问等级。
<i>password</i>	密码输入框。
<i>color_bk</i>	控件背景色 (可选)。句法: <i>color_bk="#RRGGBB"</i>
<i>color_fg</i>	控件前景色 (可选)。句法: <i>color_fg="#RRGGBB"</i>
<i>Multiline</i>	多行输入。
<i>transparent</i>	图片透明色。
<i>abscissa</i>	第一个坐标轴的名称 (仅用于 <code>graphicbox</code> )
<i>ordinate</i>	第二个坐标轴的名称 (仅用于 <code>graphicbox</code> )
<i>cursortext</i>	此属性确定要显示的光标文本。

示例:

```

<control name = "progress1" xpos = "10" ypos = "10" width = "100" fieldtype = "progressbar"
hotlink = "true" refvar = "nck/channel/geometricaxis/actpropos[1]" >
  <property min = "0" />
  <property max = "1000" />
</control>

...

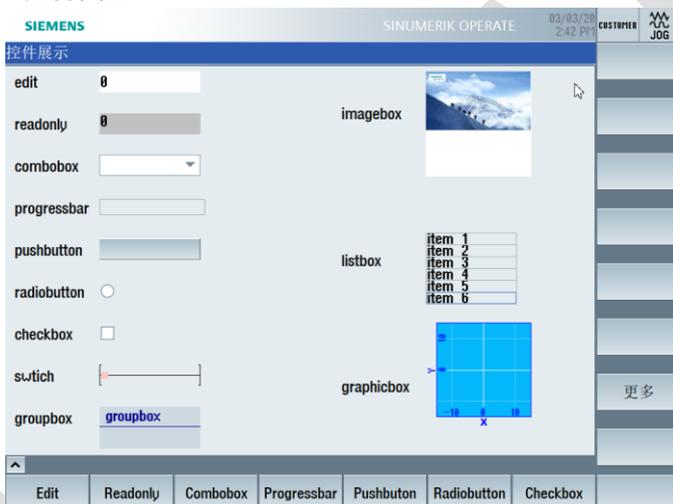
<control name = "edit1" xpos = "10" ypos = "10" >
  <property min = "20" />
  <property max = "40" />
  <property default = "25" />
</control>

<control name="edit1" xpos="208" ypos="32" >
  <property cursortext="cursor text field edit1" />
</control>

```



## 8.2 控件类型



参考 / Easy XML Library / 3\_控件展示 文件夹中的项目

### 8.2.1 Edit 控件

可编辑输入框，支持数值，字符串等数据格式的读写。

更改 edit 控件输入值时，会自动触发 form 的 edit\_changed 事件（详见 form-事件说明章节）。

**注：**输入框默认为左对齐。通过 type 属性定义显示类型（如 int，string 等），通过 format 属性定义显示格式（如：小数点位数）。

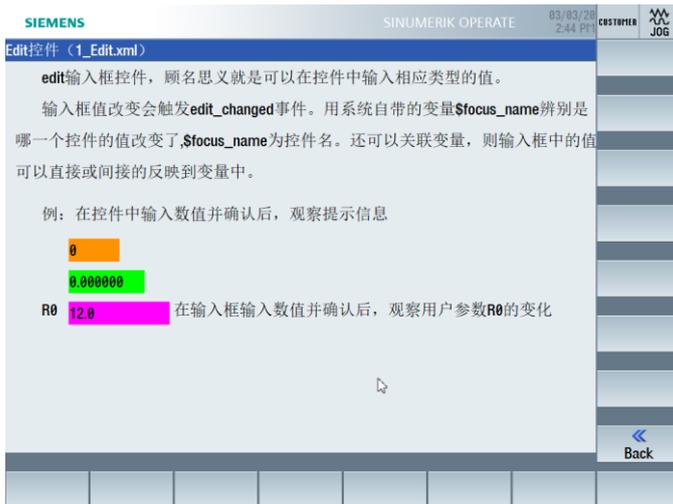
示例

```

<control name = "edit1" xpos = "110" ypos = "050" fieldtype = "edit" item_data="1"
hotlink="true" refvar="var1" type="double" format="%0.3f" color_bk="#33EECC"/>

```





参见: /Easy XML Library/ 3\_控件展示/1\_edit.xml

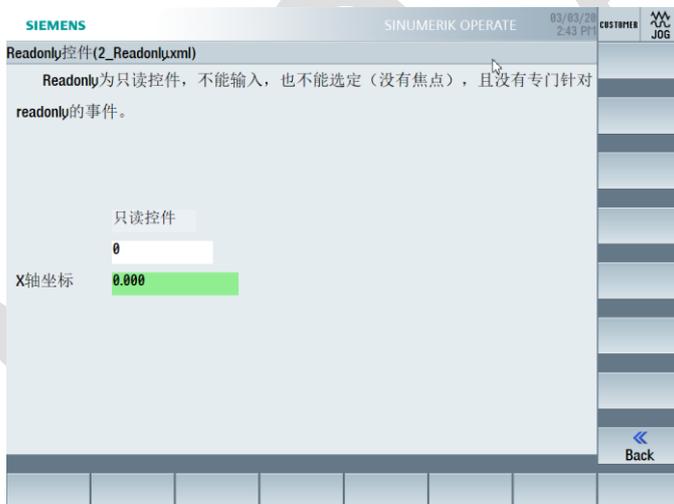
## 8.2.2 Readonly 控件

与 edit 控件相似, readonly 为只读控件, 无焦点输入。

```

<init>
<caption>Readonly控件展示</caption>
<control name="readonly_1" xpos="60" ypos="160" fieldtype="readonly" width="100" height="22" type="string"
refvar="var_1" color_bk="#ffffff" />
<control name="readonly_2" xpos="60" ypos="190" fieldtype="readonly" width="100" height="22" type="string"
refvar="nck/Channel/MachineAxis/actToolBasePos[u1,1]" hotlink="true" color_bk="#ffffff" />
</init>

```



参见: /Easy XML Library/ 3\_控件展示/2\_readonly.xml

## 8.2.3 Combobox 控件

通过 control 标签, 属性 fieldtype="combobox" 生成一个空的下拉框控件。通过标签 <ITEM>可以向控件增加下拉框选项, 通过属性 ITEM value 为该选项赋唯一的一个值。

在创建了控件后, 可以使用系统预定义函数 control.loaditem, control.addItem 等来增删选项、清空下拉框 (详见系统函数-选项处理章节)。

切换 combobox 控件的选项值时, 会自动触发 form 的 index\_changed 事件 (详见 form-事件说明章节)。

```

<control name="conbobox_1" xpos="60" ypos="235" width="100" height="22" fieldtype="combobox"
refvar="var" hotlink="true">
  <item value="11">第一项</item>
  <item value="22">第二项</item>
  <item value="33">第三项</item>
</control>

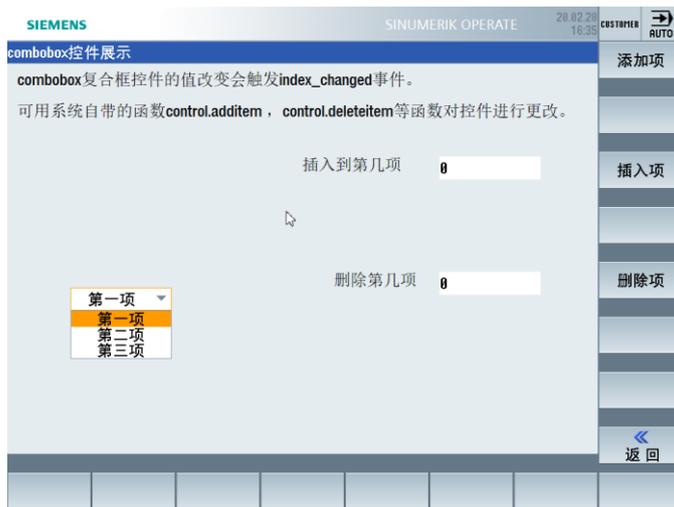
```

添加选项

```

<function name="control.additem">_T"conbobox_1", var_4,var_5</function>

```



参见: /Easy XML Library / 3\_控件展示 / 3\_combobox.xml

#### 8.2.4 Listbox 控件

通过 control 标签, 属性 fieldtype="listbox" 生成一个空的列表框控件。通过标签 <ITEM> 会向列表框中添加一个列表框元素。通过属性 ITEM value 为该元素赋唯一的一个值。

在创建了控件后, 可以使用系统预定义函数 control.loaditem, control.additem 等来增删列表框元素、清空列表框 (详见系统函数-选项处理章节)。

相关函数:

增加元素	<function name="control.additem"> control name, item, itemdata </function>
插入元素	<function name="control.insertitem"> control name, index, item, itemdata </function>
删除元素	<function name="control.deleteitem"> control name, index </function>
添加列表	<function name="control.loaditem"> control name, list </function>
清空列表框	<function name="control.empty"> control name </function>
获取控件名称	<function name="control.getfocus" return="focus_name" />
获取列表框光标所在下标	<function name="control.getcurssel" return="var">control name </function>
在将光标放置在相应行中	<function name="control.setcurssel" > control name, index</function>
获取列表框中所选行的内容	<function name="control.getitem" return="var">control name, index </function>
获取所选行的自定义元素值	<function name="control.getitemdata" return="var">control name, index </function>

示例:

```

<let name="itemlist" type="string">text11\\ntext12\\ntext13\\ntext14\\n</let>

```

控件:

```

<control name="lb1" xpos = "10" ypos = "94" width="200" height="250" fieldtype="listbox" refvar="tmp" hotlink="true" >
<item value="2">text1 </item>
<item value="3">text2 </item>
<item value="4">text3 </item>
<item value="5">text4 </item>
<item value="6">text5 </item>
<item value="7">text6 </item>
<item>text7 </item>
<item>text8 </item>
</control>

```

添加元素:

```

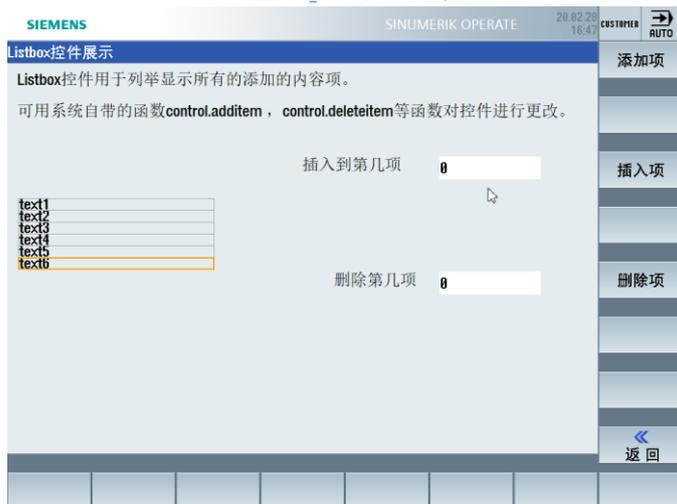
<function name="control.additem" >_T"lb1",_T"text9",10 </function>
<function name="control.additem" >_T"lb1",_T"text10",11 </function>

```

```

<function name="control.loaditem" >_T"lb1",itemlist </function>

```



参见: /Easy XML Library / 3\_控件展示 / 12\_listbox.xml

## 8.2.5 Progressbar 控件

通过 control 标签, 属性 fieldtype="progressbar"可创建一个进度条控件。在 0 到 100 内显示进度条。值域与控件的最小值 min 和最大值 max 属性相匹配。可通过 alignment 属性定义水平 (默认, hr) 或垂直 (vr) 进度条。通过 width, height 属性定义进度条的宽和高。

示例:

```

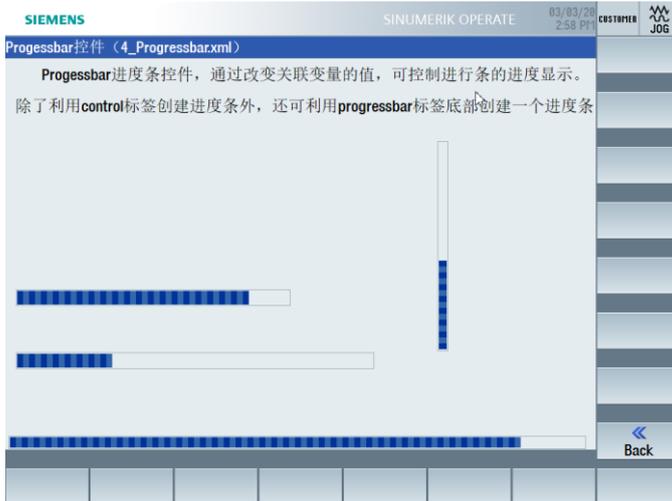
<CONTROL name = "progress1" xpos = "10" ypos = "240" width = "260" fieldtype = "progressbar"
hotlink = "true" refvar = "var" >
<PROPERTY min = "0" />
<PROPERTY max = "100" />
</CONTROL>

```

```

<CONTROL name = "progress2" xpos = "410" ypos = "100" width = "10" height = "200"
fieldtype = "progressbar" hotlink = "true" refvar = "var">
<PROPERTY min = "0" />
<PROPERTY max = "200" />
<property alignment="vr"/>
</CONTROL>

```



参见: /Easy XML Library / 3\_控件展示 / 4\_progressbar.xml

## 8.2.6 Scrollbar 控件

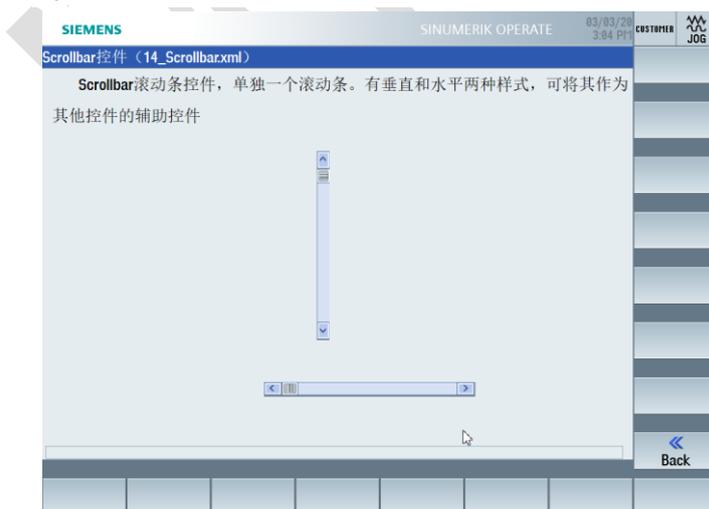
通过 control 标签, 属性 fieldtype="scrollbar"可创建一个滚动条控件。值域与滚动条的最小值和最大值属性相匹配。可通过 alignment 属性定义水平 (默认,hr) 或垂直 (vr) 滚动条。

通过 width, height 属性定义滚动条的宽和高。

示例:

```
<control name = "scrollbar1" xpos = "260" ypos = "100" width = "12" height="180" fieldtype = "scrollbar"
hotlink = "true" refvar = "bar_sensor_bit" >
<PROPERTY min = "0" />
<PROPERTY max = "100" />
<property alignment="vr"/>
</control>
```

```
<control name = "scrollbar2" xpos = "210" ypos = "320" width = "200" height="14" fieldtype = "scrollbar"
hotlink = "true" refvar = "bar_sensor_bit" >
<PROPERTY min = "0" />
<PROPERTY max = "100" />
<property color_bk="#fff0ff"/>
</control>
```



参见: /Easy XML Library/ 3\_控件展示 / 14\_scrollbar.xml

## 8.2.7 Progress\_bar 窗体进度条

通过 progress\_bar 标签可创建一个窗体进度条。进度条显示在窗体的下方。每个窗体有一个。

语法: `<PROGRESS_BAR type="true/ or false" min="最小值" max="最大值" > value </PROGRESS_BAR>`

属性:

- type = "TRUE" - 打开进度条
- type = "FALSE" - 关闭进度条
- min (可选) - 最小值
- max (可选) - 最大值
- Value - 进度条的百分比位置

示例:

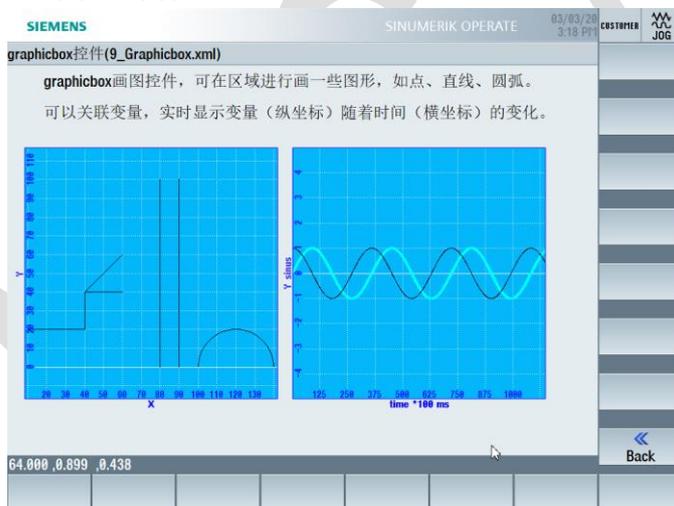
```
<timer>
  <op> var=var+1 </op>
  <progress_bar type="true" min="0" max="100">var </progress_bar>
</timer>
```



参见: [/Easy XML Library/ 3\\_控件展示 / progress\\_bar.xml](#)

## 8.2.8 Graphicbox 控件

### 1. 绘图 2. 采样



参见: [/Easy XML Library/ 3\\_控件展示 / 9\\_Graphicbox.xml](#)

- 通过 control 标签, 属性 fieldtype="graphicbox"可创建一个 2 维图形控件
- 通过<item>标签定义控件中的图形元素, 通过 control.additem, control.loaditem 系统函数添加图形元素
- 可分配参考变量, 每 100ms 周期采样一次数值, 最多记录 10000 个值。属性 max 有数值时, 到达最大记录时间, 则会删除最早记录的数值。

- 使用 channel 属性 (channel="0/1/2/3") 最多可分配 4 个参考变量, channel2,3,4 需要使用 request 指令声明 (详见[系统函数-信号处理](#)章节), 否则无法采集。
- 属性 abscissa = "第一个坐标轴的名称", ordinate = "第二个坐标轴的名称"

### 8.2.8.1 绘制图形

使用函数 control.additem, control.loaditem 添加以下 2 维元素 :

- 支持直线、圆弧、点 3 中基本元素
- 语法格式固定, 由关键字+图形坐标值组成, 整体为字符串类型, 需加前置符\_T
- 如要使用变量作为起点或终点时, 需要使用 print 指令将其生成为符合语法格式的字符串。
- 关键字: 直线为 l, 圆弧为 c, 点为 p;
- 关键字, 各坐标值之间由 ";" 隔开

元素	语法格式 (string)	示例	参数数量
直线 l	l; xs; ys; xe; ye	l;10;20;40;20	4 (起点, 终点)
圆弧 c	c;xs;ys;xe;ye;c_x;c_y;r	c;100;0;140;0;120;0;20	7 (起点, 终点, 圆心, 半径)
点 p	p; x; y		2 (点的坐标)

xs/ys: x/y 起点坐标; xe/ye: x/y 终点坐标; c\_x/c\_y: 圆心坐标; r: 圆半径; x/y: x/y 坐标

示例:

建立控件

```
<control name="groupbox_1" xpos="6" ypos="100" width="250" height="250" fieldtype="graphicbox" />
```

添加元素: 使用 loaditem 方法可一次添加多个元素, 元素中间需要使用换行转义符\n

```
<function name="control.loaditem">_T"l;10;20;40;20\nl;40;40;60;60"</function>
<function name="control.additem">_T"l;40;20;40;40",0</function>
<function name="control.additem">_T"l;40;40;60;40",0</function>

<function name="control.additem">_T"l;80;0;80;100",0</function>
<function name="control.additem">_T"l;90;0;90;100",0</function>

<function name="control.additem">_T"c;100;0;140;0;120;0;20",0</function>
```

添加变量元素:

```
<print name="item_string" text="l; %f; %f; %f; %f\n">s_z, s_x, e_z, e_x </print>
<function name="control.additem">_T"c_gbox", item_string, 0</function>
```

### 8.2.8.2 采样跟踪

- 在 control 控件中增加 refvar 属性, 默认为 channel0, 关联采样变量 (局部变量, PLC 变量, NC 变量, 驱动变量等), 注意 hotlink 属性应为 true 才能实时更新
- 如要监控多个信号, 需要使用 request 先对采样信号处理, 然后再 channel1 属性中增加关联变量
- 可使用 color 属性定义信号绘制的颜色, penwidth 属性定义线条粗细 (单位: 像素)

示例:

```
<init>
  <REQUEST name="s_var3" /> <!-- 激活信号采样 -->
  ...
```

<init>

### 建立控件

```
<control name="a_gbox" xpos="260" ypos="100" width="250" height="250" fieldtype="graphicbox"
  refvar="s_var2" hotlink="true">
  <property max="60000" /> <!-- steps of ms -->
  <property min="0" />
  <property channel="0" color="#00ffff" penwidth="3"/>
  <property ORDINATE="Y sinus" />
  <property ABSCISSA="time *100 ms" />
  <property channel="1" refvar="s_var3" color="#000000" />
</control>
```

### 8.2.9 Pushbutton 控件

- 通过 control 标签，属性 fieldtype="pushbutton"可创建一个触摸屏下压按钮
- 可通过触控或鼠标来操作
- 通过属性 color\_fg、color\_bk、color\_fg\_pressed 和 color\_bk\_pressed 修改前景色和背景色。
- 通过 checkable 属性定义按键是否可以自锁，通过 disabled 属性控制按钮的可操作性
- 通过属性 function 指定处理程序功能。
- 在触控操作中，当手指手势“松开”结束后，才提交手指手势“按下”和“松开”。

语法:

```
<control name="name" xpos="x position" ypos="y position" fieldtype="pushbutton" >
  <caption></caption>
</control>
```

或者 带处理程序功能

```
<control name="name" xpos="x position" ypos="y position" fieldtype="pushbutton"
  function="button_handler" hotlink="true" >
  <caption></caption>
</control>
```

...

```
<function_body name="button_handler">
```

...

```
</function_body>
```

#### 8.2.9.1 按钮状态

		
按钮未按下	按钮按下	禁用按钮

#### 8.2.9.2 按钮类别

- 普通按钮（默认）:
- 带自锁功能的按钮：通过属性 <property checkable="true" /> 定义
- 禁用按钮：通过属性<property disabled="true " /> 定义

示例:

```
<control name="name" xpos="x position" ypos="y position" fieldtype="pushbutton" >
<caption alignment="left">Button text</caption>
<property checkable="true/false" />
</control>
```

```
<control name="name" xpos="x position" ypos="y position" fieldtype="pushbutton" >
<caption alignment="left">Button text</caption>
<property disabled="true " />
</control>
```

### 8.2.9.3 按钮属性

按钮未按下

<Caption>.. <caption&gt;< td=""> <td>按钮文本</td> </caption&gt;<>	按钮文本
<Caption alignment="xxx">.. <caption&gt;< td=""> <td>按钮文本对齐方式</td> </caption&gt;<>	按钮文本对齐方式
picture	前景图
picture_alignment	前景图对齐方式
picture_scaled	根据按钮的尺寸调整图标 (bool)
picture_keepaspectratio	前景图长宽比 (bool)
picture_textalignedtopicture	文本与图片对齐方式 (bool)
backgroundpicture	背景图
backgroundpicture_keepaspectratio	背景图长宽比 (bool)
backgroundpicture_scaled	根据按钮的尺寸调整背景图标 (bool)
backpicture_alignment	背景图对齐方式

按钮按下

<Caption pressed="true">.. <caption&gt;< td=""> <td>按钮文本</td> </caption&gt;<>	按钮文本
<Caption pressed="true" alignment="xxx">.. <caption&gt;< td=""> <td>按钮文本对齐方式</td> </caption&gt;<>	按钮文本对齐方式
picturepressed	前景图
picturepressed_alignment	前景图对齐方式
backgroundpicturepressed	背景图
picturepressed_textalignedtopicture	文本与图片对齐方式 (bool)
picturepressed_keepaspectratio	前景图长宽比 (bool)
backgroundpicturepressed_keepaspectratio	背景图长宽比 (bool)
picturedisabled_scaled	根据按钮的尺寸调整图标 (bool)
backgroundpicturepressed_scaled	根据按钮的尺寸调整背景图标 (bool)
backgroundpicturepressed_alignment	背景图对齐方式

### 禁用按钮

captiondisabled_alignment	按钮文本对齐方式
picturedisabled	前景图
picturedisabled_alignment	前景图对齐方式
backgroundpicturedisabled	背景图
picturedisabled_textalignedtopicture	文本与图片对齐方式 (bool)
picturedisabled_keepaspectratio	前景图长宽比 (bool)
backgroundpicturedisabled_keepaspectratio	背景图长宽比 (bool)
picturepressed_scaled	根据按钮的尺寸调整图标 (bool)
backgroundpicturedisabled_scaled	根据按钮的尺寸调整背景图标 (bool)
backgroundpicturedisabled_alignment	背景图对齐方式

### 按钮属性示例:

左对齐显示	<code>&lt;caption alignment="left"&gt;Button text&lt;/caption&gt;</code>
按下后显示	<code>&lt;caption pressed="true" alignment="left"&gt;Button text pressed&lt;/caption&gt;</code>
文本左对齐, 背景图透明, 前景图缩放	<code>&lt;caption alignment="left" &gt;This is my first touch button!&lt;/caption&gt; &lt;property picture="sk_circ_grind_cg.png" alignment="left" TextAlignedToPicture="true" scaled="true"/&gt; &lt;property backgroundpicture="pbutton.png" alignment="stretch" transparent="#ffffff"/&gt;</code>

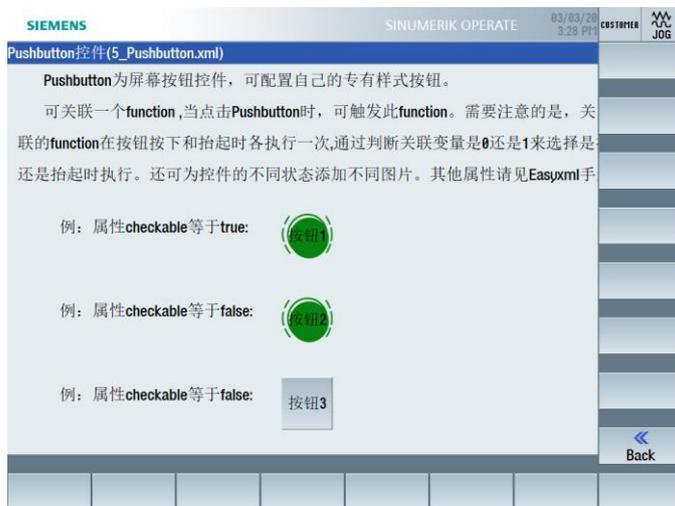
### 按钮展示:

Picture 未按下按钮	PicturePressed 按下按钮	PictureDisabled 禁用按钮	BackgroundPicture + Picture 背景图+前景图
			
TextAlignedToPicture="true" 基于图标将文本对齐	scaled="true" 前景图缩放, 文字自动对齐	alignment="stretch" transparent="#ffffff"	alignment="stretch" transparent="#ffffff"
			

### 更多:

- 按钮文本 Caption 具有附属属性 alignment, 可赋值: left, right, top, bottom, center
- 按钮图标 (图片) 具有附属属性 alignment, 可赋值: left, right, top, bottom, center
- 在按钮图标 (图片) 对齐属性 alignment 中, 背景图标可以赋值="stretch", 程序会将图标缩放至按钮的矩形范围。借助通过属性 transparent 指定透明色, 能够产生按钮的任意轮廓。

### 8.2.9.4 示例



参见: [/Easy XML Library/ ../ 5\\_pushbutton.xml](#)

### 8.2.10 Switch 控件

- 通过 control 标签, 属性 fieldtype="switch"可创建一个触摸屏开关, 通过图标指示两种状态
- 可通过触控或鼠标来操作
- 属性 alignment 实现开关的垂直 (vr) 或水平 (hr) 显示: alignment="hr/vr"

#### 句法

```
<control name="name" xpos = "x position" ypos="y position" width="width" height="height"
  fieldtype="switch" alignment="hr/vr">
  <property left_position="value" caption="on" picture="name" .../>
  <property right_position="value" caption="off" picture="name" .../>
</control>
```

#### 主属性

left_position	向左运动的情况下, 为控制项变量分配这个属性的值。
right_position	向右运动的情况下, 为控制项变量分配这个属性的值。
upper_position	向上运动的情况下, 为控制项变量分配这个属性的值。
lower_position	向下运动的情况下, 为控制项变量分配这个属性的值。
disabled	定义控制开关的可操作性。为 true, 则不对操作进行分析。

位置属性: 归属于主属性, 如 left\_position 等

caption	归属于开关位置的文本, 受开关尺寸限制
transparent	消除图片背景色, 用于与开关位置对应的图标
picture	前景色图片
pictureDisabled	禁用状态图片

示例:

水平显示

```
<control name="name" xpos="x position" ypos="y position" width="width" height="height" fieldtype="switch"
  alignment="hr">
  <property left_position="value" picture="name" />
  <property right_position="value" picture="name" />
</control>
```

垂直显示

```
<control name="name" xpos="x position" ypos="y position" width="width" height="height" fieldtype="switch"
  alignment="vr">
  <property upper_position="value" picture="name" />
  <property lower_position="value" picture="name" />
</control>
```

或者分配处理程序功能

```
<control name="name" xpos="x position" ypos="y position" width="width" height="height" fieldtype="switch"
  function="switch_handler" hotlink="true" alignment="vr">
</control>
```

```
<let name="switch_state" />
<control name="main_switch" xpos="100" ypos="53" width="40" height="30" fieldtype="switch"
  refvar="switch_state" hotlink="true" alignment="hr">
  <property left_position="10" picture="icon_left.bmp" />
  <property right_position="11" picture="icon_right.bmp" />
</control>
```

icon\_left.bmp



icon\_right.bmp



### 8.2.10.1 示例



参见: [/Easy XML Library/ 3\\_控件展示 / 8\\_switch.xml](#)

### 8.2.11 Radiobutton 控件

- 通过 control 标签，属性 fieldtype="radiobutton"可创建一个单选框
- 借助单选框能够从多个选项选择一个。按钮状态被传输至控制项变量。
- 可通过触控或鼠标来操作

#### 句法

```
<control name="name" xpos="x position" ypos="y position" width="width" height="height"
  fieldtype="radiobutton" >
  <caption>button text</caption>
</control>
```

#### 示例:



参见: [/Easy XML Library/ 3\\_控件展示 / 6\\_radiobutton.xml](#)

### 8.2.12 Checkbox 控件

- 通过 control 标签，属性 fieldtype="checkbox"可创建一个复选框。
- 借助复选框能够选择多个选项。复选框的状态被传输至控制项变量。
- 可通过触控或鼠标来操作

#### 句法

```
<control name="name" xpos="x position" ypos="y position" width="width" height="height"
  fieldtype="checkbox" >
  <caption>button text</caption>
</control>
```

#### 示例



参见: [/Easy XML Library/ 3\\_控件展示 / 7\\_checkbox.xml](#)

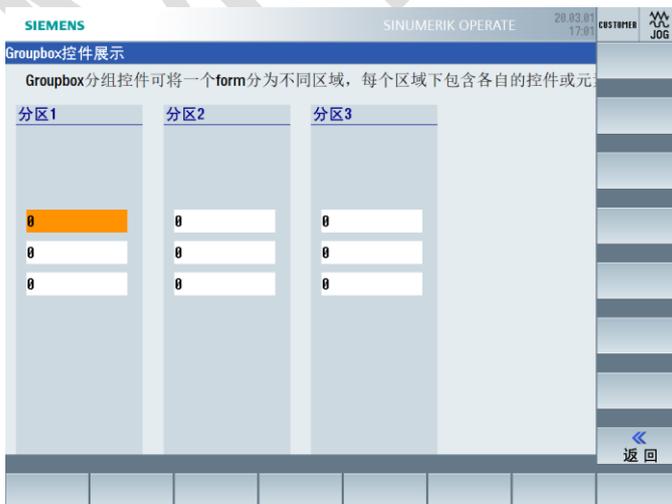
### 8.2.13 Groupbox 控件

- 通过 control 标签，属性 fieldtype="groupbox"可创建一个分组框
- 分组框将一组控制项包围，具有框架和标题。
- 通过标签 caption 定义分组框的标题。
- 所有归属于组的 control 控件被作为子控制项嵌入 Control 标签。

句法

```
<control name="name" xpos="x position" ypos="y position" width="width" height="height"
  fieldtype="groupbox">
  <caption>caption text</caption>
  <control>...</control>
  ...
</control>
```

示例



参见: [/Easy XML Library/ 3\\_控件展示 / 10\\_groupbox.xml](#)

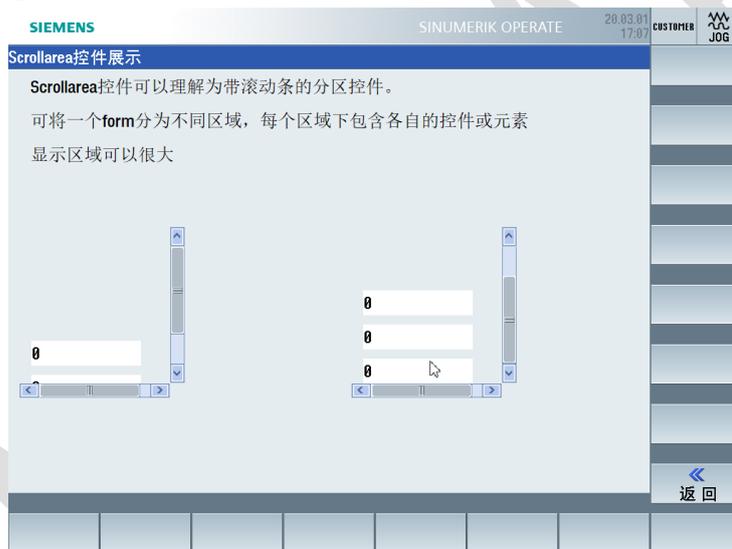
### 8.2.14 Scrollarea 控件

- 通过 control 标签，属性 fieldtype="scrollarea"可创建一个滚动区域
- 滚动区域用于显示定义的区域内的 control 控件。
- 支持触控或鼠标操作
- 嵌入的控制项的坐标以滚动区域的左上角为基准，控件长度超出滚动区域则自动激活滚动条。
- 所有归属于滚动区域的 control 控件被作为子控制项嵌入 Control 标签。

#### 语法

```
<control name="name" xpos="x position" ypos="y position"
width="width" height="height" fieldtype="scrollarea">
  <control>...</control>
  <control>...</control>
  ...
</control>
```

#### 示例



参见: /Easy XML Library/ 3\_控件展示 / 11\_scrollarea.xml

### 8.3 更改控件属性

- 在 form 中更改控件属性，可借助 function 或在事件中更改
- 在 menu 中操作控件，需要借助 send\_message 方法，使用系统定义的 send\_message 方法，将参数传递给窗体，同时触发 message 事件，在事件中定义要修改控件的新的属性值。

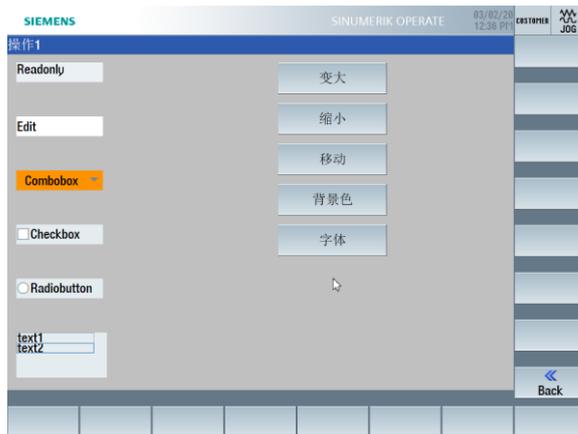
```
<menu>
  ...
  <send_message>1,2</send_message>
  ...
</menu>
<form>
  ...
```

```

<message>
<control name="edit_1" fieldtype="readonly" color_bk="#00ff00"/>
<control name="edit_1" xpos="140"/>
...
</message>
...
</form>

```

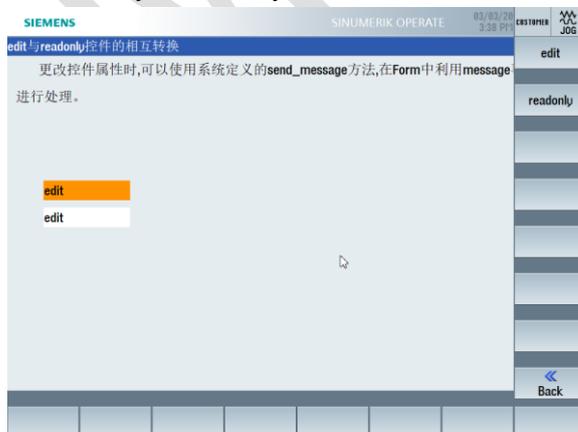
### 示例



参见: [/Easy XML Library/3\\_控件展示/2\\_控件的处理/control\\_operate1.xml](#)



参见: [/Easy XML Library/3\\_控件展示/2\\_控件的处理/control\\_operate2.xml](#)



参见: [/Easy XML Library/3\\_控件展示/2\\_控件的处理/edit\\_readonly\\_exchange.xml](#)

## 9 函数和方法

### 9.1 数据处理

标签：OP, DATA, DATA\_LIST, DATA\_ACCESS, TYPE\_CAST, UPDATE\_CONTROLS

#### 9.1.1 OP

该标签会执行给定的操作。

- 可以执行“运算符”章节中所列出的各种运算。
- 一个运算标签内可以使用多个等式。用分号标记指令结束。

示例：

```
<let name = "tmpvar" type="int" > </let>

<op> tmpvar = "plc/mb170" </op>
<op> tmpvar = tmpvar *2 </op>
<op> "plc/mb170" = tmpvar </op>
```

字符串：

<OP>标签可以对字符串进行处理，将结果赋值给等式中指定的字符串变量。

- 字符串需要在其前放置\_T（如\_T"abc"）。
- 还可对变量值进行格式化，在其前放置\_F 格式指令（如\_F%9.3f "变量"）

示例：

```
<LET name="buffer" type="string" ></LET>

<OP> buffer = _T"unformatted value R0= " + "nck/Channel/Parameter/
R[0]" + _T" and " + _T"$85051" + _T" formatted value R1 " + _F
%9.3f"nck/Channel/Parameter/R[1]" </OP>
```

#### 9.1.2 DATA

该标签可结合定址写入 NC、PLC、GUD 和驱动数据（定址语法说明参见本手册“系统变量定址”章节）。

语法：

```
<DATA name=<variable name>"> value </DATA>
```

属性：

- name：变量地址
- value：写入值，支持变量，变量名称前需加“\$”如 \$tempVar

示例：

```
<DATA name = "plc/mb170" > 1 </DATA>
...
<LET name = "tempVar" > 7 </LET>
<DATA name = "plc/mb170" >$tempVar </DATA>
```

#### 9.1.3 DATA\_LIST

该标签可以保存或者恢复所列出的驱动数据和机床数据。

按行方式列出地址。可以创建最多 20 个临时数据列表。

（定址语法说明参见本手册“系统变量定址”章节）

语法:

```
<DATA_LIST action="<read/write/append>" id="<list name>">
```

```
NC/PLC 地址 1
```

```
NC/PLC 地址 2
```

```
...
```

```
</DATA_LIST>
```

属性:

action

- read: 将列出变量的值存储到一个临时存储器中
- append - 将列出变量的值添加到已有列表中
- write - 将保存的值复制到相应的机床数据中

id: 用于识别临时存储器

示例:

```
<DATA_LIST action = "read" id = "<name>" >
nck/channel/parameter/r[2]
nck/channel/parameter/r[3]
nck/channel/parameter/r[4]
$MN_USER_DATA_INT[0]
...
</ DATA_LIST >
<DATA_LIST action = "write" id = "<name>" />
```

#### 9.1.4 DATA\_ACCESS

该标签在储存用户输入数据时用来控制窗体的数据传输特性。

- 在 form 中<init>标签定义。
- 未使用该标签时, 则总是使用输入数据中间存储器。
- 例外: 在 hotlink="true"的控件中直接读写, 不经过中间存储器。

句法:

```
<DATA_ACCESS type = "true" />
```

属性:

- Type="true": 不对输入值进行缓存。窗体直接将输入值复制到参考变量中。
- Type="false": 值只通过标签 update\_controls type = "false" 复制到参考变量中。

示例:

用按键调用函数

```
<softkey position="9" >
<caption>refresh </caption>
<function name="data_refresh" />
</softkey>
```

```
<softkey_ok>
<update_controls type="false" />
</softkey_ok>
```

```
<softkey_cancel>
<update_controls type="true" />
</softkey_cancel>
```

## Form 窗体编程

```
<init>
  <caption>(data_access.xml) </caption>
  <data_access type="false" />
  <control name = "c1" xpos = "322" ypos = "34" refvar="nck/Channel/Parameter/R[0]" />
  <control name = "c2" xpos = "322" ypos = "54" refvar="nck/Channel/Parameter/R[1]" />
  <control name = "c3" xpos = "322" ypos = "74" refvar="nck/Channel/Parameter/R[2]" hotlink="true" />
</init>
```

窗体输入框数值改变时调用如下事件，并在状态栏显示变量值

```
<edit_changed>
  <op>
    rpa_v[0]="nck/Channel/Parameter/R[0]"
    rpa_v[1]="nck/Channel/Parameter/R[1]"
    rpa_v[2]="nck/Channel/Parameter/R[2]"
    rpa_v[3]=c1
    rpa_v[4]=c2
    rpa_v[5]=c3
  </op>
  <print text="data: R0=%d, R1=%d, R2=%d; c1=%d, c2=%d, c3=%d, " >
    rpa_v[0],rpa_v[1],rpa_v[2],rpa_v[3],rpa_v[4],rpa_v[5] </print>
</edit_changed>
```

定义的函数，用于在状态栏监控变量数值

```
<function_body name="data_refresh" >
  <print text="data: R0=%d, R1=%d, R2=%d; c1=%d, c2=%d, c3=%d, " >
    rpa_v[0],rpa_v[1],rpa_v[2],rpa_v[3],rpa_v[4],rpa_v[5] </print>
</function_body>
```

### 9.1.5 TYPE\_CAST

该标签用于局部变量的数据类型转换。

语法：

```
<type_cast name="variable name" type="new type" />
```

属性：

- name：变量名称
- type：给变量赋值新的数据类型。
- convert：给变量赋值新的数据类型。此外，变量值转换为新的数据类型。

示例 1：

<将字符串转换为整数>

```
<LET name="excl_keycode" type="string"/>
<OP>excl_keycode = exclude_key </OP>
<type_cast name="excl_keycode" type="int" />
<PRINT text="keycode=%d, excl_keycode=%d" >$keycode, excl_keycode </PRINT>
```

示例 2：

<用户循环中的类型转换>

```

<form name = "cycle_my1_form" type= "cycle" >
<nc_instruction refvar="cycle_block">CYCLE_MY1($p1, $p2, $p3, $p4, $p5) </nc_instruction>
<init>
<caption>cycle (file:cus_cycle.xml) </caption>

<type_cast name="$p1" type="double"/>
<type_cast name="$p2" type="double"/>
<type_cast name="$p3" type="double"/>
<type_cast name="$p4" type="int"/>
<type_cast name="$p5" type="int"/>

<op>p5_part1 = $p5 & #f </op>
<op>p5_part2 = $p5/16 & #f </op>

```

通过运算符转换类型:

```

<op>p5_part1 = $p5 & #f </op>
<op>p5_part2 = $p5/16 & #f </op>
<op> $p5 = p5_part1 | (p5_part2 * 16) </op>

```

### 9.1.6 UPDATE\_CONTROLS

该标签用于控件变量和参考变量之间的传输。

语法:

```
<UPDATE_CONTROLS type = "true/false"/>
```

type="true": 从参考变量读取数据并复制到控件变量中。

type="false": 从控件变量读取数据并复制到参考变量中。

示例:

```

<SOFTKEY_OK>
< UPDATE_CONTROLS type="false"/>
</SOFTKEY_OK>

```

可参考< DATA\_ACCESS>标签。

## 9.2 复位重启

标签: CONTROL\_RESET, WAITING, HMI\_RESET

### 9.2.1 CONTROL\_RESET

该标签可以重新启动一个或多个控制系统组件。

语法:

```
<CONTROL_RESET 关键字="TRUE" />
```

关键字:

- RESETNC = "TRUE" : 重启 NC 组件
- RESETDRIVE = "TRUE" : 重启驱动组件

示例:

```

<CONTROL_RESET resetnc="TRUE" />
<CONTROL_RESET resetnc = "true" resetdrive = "true"/>

```

### 9.2.2 WAITING

标签在 NC 或者驱动复位后等待组件重新启动。

语法:

```
<WAITING WAITINGFORNC = "TRUE"/>
```

属性:

- WAITINGFORNC = "TRUE" - 等待 NC 重启
- WAITINGFORDRIVE = "TRUE" - 等待驱动重启

示例:

```
<CONTROL_RESET resetnc = "true" resetdrive = "true"/>  
<WAITING waitingfornc = "true" waitingfordrive = "true" />
```

### 9.2.3 HMI\_RESET

该标签会触发 HMI 重新启动。

示例:

```
<HMI_RESET />
```

注: 编译在该指令后会被中断。

## 9.3 用户循环

标签: CREATE\_CYCLE, CREATE\_CYCLE\_EVENT, NC\_INSTRUCTION, MMC

### 9.3.1 CREATE\_CYCLE

循环支持功能可以通过格式对话框自动创建和反编译循环调用。

为了标记循环窗口, 应在标签 FORM 中将属性 type 的值定义为 cycle。该标记可实现指令 NC\_INSTRUCTION 的执行。

主要逻辑:

1. 触发方法: 执行标签 <CREATE\_CYCLE />
2. 自动响应事件, 执行标签 <CREATE\_CYCLE\_EVENT>
3. 事件结束后, 输出 NC 指令, 执行标签 <NC\_INSTRUCTION>

标签	说明
Form	建立循环的标签要求 form 的 type 属性必须为 cycle <pre>&lt;form name = "cycle_my1_form" type= "cycle" &gt;</pre>
CREATE_CYCLE	建立循环。保存各参数变量的值并生成 NC 指令。 语法: <pre>&lt;CREATE_CYCLE refvar="name" /&gt;</pre> Refvar: 关联变量, 用于接收<CREATE_CYCLE_EVENT>返回值。
CREATE_CYCLE_EVENT	处理循环参数, 完成底层参数运算。 语法: <pre>&lt;CREATE_CYCLE_EVENT&gt; ... &lt;/CREATE_CYCLE_EVENT&gt;</pre>
NC_INSTRUCTION	输出 NC 指令。含有要执行的循环调用, 参数通过占位符预留。 示例: <pre>&lt;NC_INSTRUCTION&gt;MY_CYCLE(\$P1, \$P2)&lt;/ NC_INSTRUCTION &gt;</pre>
写入 NC 程序	通过系统预定义方法 doc.writetofile 写入加工程序 <pre>&lt;function name="doc.writetofile" &gt;nc_program, nc_file&lt;/function&gt;</pre>

NC 程序选择	<p>可以选择将要处理的程序。程序必须保存在 NC 文件系统中。</p> <pre>&lt;function name="ncfunc.select"&gt; progname &lt;/function&gt;</pre> <p>示例:</p> <pre>&lt;function name="ncfunc.select"&gt; _T"n:\mpftest.mpf" &lt;/function&gt;</pre>
---------	---

示例:

定义要写入的 NC 程序路径

```
</let name="nc_program" type="string" >n:\mpf\cycle_wrapper.mpf </let>
```

定义 form 属性为 cycle，声明输出的 NC 指令格式，多行指令使用换行符\ln。

```
<form name = "cycle_my1_form" type= "cycle" >
  <nc_instruction refvar="cycle_block" >CYCLE_MY1($p1, $p2, $p3, $p4, $p5) </nc_instruction>
  <init>
```

...

```
</init>
```

在 form 内建立<create\_cycle\_event>标签，当执行<CREATE\_CYCLE />方法时自动跳转到该标签中处理参数。

```
<create_cycle_event>
  <op> $p5 = p5_part1 | (p5_part2 * 16) </op>
</create_cycle_event>
```

事件结束，将 NC 指令返回到关联的变量 cycle\_block 中。通过 doc.writetofile 方法写入指定的 NC 程序中。

```
<softkey_ok>
  <create_cycle refvar="cycle_block" />
  <op>nc_file = cycle_block + _T"M17" </op>
  <function name="doc.writetofile" >nc_program, nc_file </function>
  <close_form />
  <navigation>menu_simple_cycle </navigation>
</softkey_ok>
```

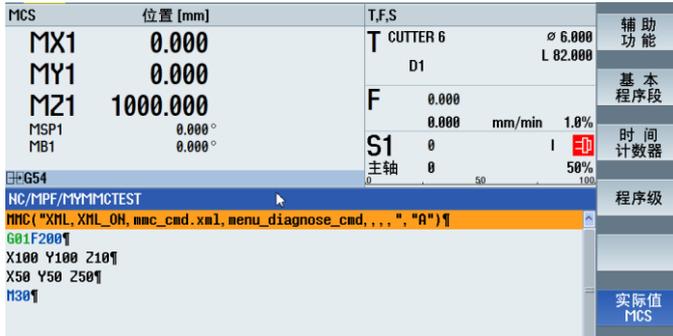
示例:



参见: /Easy XML Library/ 1\_NC 功能\_读写变量等 / 3\_创建用户循环程序 / cu\_cycle.xml

### 9.3.2 MMC

可以在零件程序中通过 MMC 指令显示用户自定义对话框 (Dialog)。



语法:

MMC ( "操作区, 指令, XML 脚本文件, 菜单名称, , , 时间, " , "应答模式" )

示例:

MMC("CYCLES,PICTURE\_ON,mmc\_cmd.xml,menu\_diagnose\_cmd,,,","S")

MMC("CYCLES,PICTURE\_ON,mmc\_cmd.xml,menu\_diagnose\_cmd,,,","A")

MMC("CYCLES,PICTURE\_ON,mmc\_cmd.xml,menu\_diagnose\_cmd,,,","N")

脚本文件: mmc\_cmd.xml <menu name = " menu\_diagnose\_cmd ">

MMC("XML,XML\_ON,mmc\_cmd.xml,menu\_diagnose\_cmd,,,","S")

MMC("XML,XML\_ON,mmc\_cmd.xml,menu\_diagnose\_cmd,,,","A")

MMC("XML,XML\_ON,mmc\_cmd.xml,menu\_diagnose\_cmd,,,","N")

脚本文件: mmc\_cmd.xml <menu name = " menu\_diagnose\_cmd ">

参见: /Easy XML Library/ 1\_NC 功能\_读写变量等 / 3\_创建用户循环程序 / mmc\_cmd.xml

操作区	CYCLES	Run MyScreens 或 Easy XML 对话框
	POPUPDLG	Run MyScreens 或 Easy XML 弹窗
	XML	用户 XML
指令	PICTURE_ON, PICTURE_OFF	Run MyScreens 或 Easy XML 弹窗 或 对话框
	XML_ON, XML_OFF	用户 XML
脚本文件	<name>.com	Run MyScreens 脚本文件
	<name>.xml	XML 脚本文件
	xmlodial_emb.xml	Easy XML 弹窗
菜单名称		要显示的脚本文件中的菜单名称
应答类型	N: 无应答	指令发送后, 程序将继续执行。若无法成功执行指令, 则不进行反馈。
	A: 异步应答	指令发送后, 程序将继续执行。返回值会保存在对话框设置中定义的用户专用应答变量中, 如果为 GUD 变量, 则可在数控程序中读取。
	S: 同步应答	需对话框反馈应答后, 程序才可继续执行, 带返回值

## 9.4 窗体操作

**标签:** NAVIGATION, OPEN\_FORM, CLOSE\_FORM, POWER\_OFF, MSGBOX, PRINT, PROGRESS\_BAR, SEND\_MESSAGE, SHOW\_CONTROL, SLEEP, STOP, SWITCHTOAREA, SWITCHTODYNAMICTARGET

### 9.4.1 NAVIGATION

该标签用来确定所要调用的菜单。可以在软键程序块、菜单程序块和 Form 中设置。如果给标签分配了一个变量名称作为值，则解析器会激活变量中的菜单。菜单程序块中，导航至说明位置。后续说明将不再执行。

提示:

在需要通过变量的内容定义导航目标的情况下，必须将此变量声明为全局变量。

语法:

```
<NAVIGATION>menu name</NAVIGATION>
```

示例:

```
<menu name = "main">
  <softkey POSITION="1">
    <caption>sec. form</caption>
    <navigation>sec_menu</navigation>
  </softkey>
</menu>

<menu name = "sec_menu">
  <open_form name = "sec_form" />
  <softkey_back>
    <navigation>main</navigation>
  </softkey_back>
</menu>
```

### 9.4.2 OPEN\_FORM

该标签可以打开指定名称的对话框窗体。

属性:

- name

对话框窗口的名称

语法:

```
<OPEN_FORM name = "<form name>" />
```

示例:

```
<menu name = "main">
  <open_form name = "main_form" />
  <softkey POSITION="1">
    <caption>main form</caption>
    <navigation>main</navigation>
  </softkey>
</menu>
```

```

</softkey>
</menu>

<form name="main_form">
  <init>
  </init>

</form>

```

### 9.4.3 CLOSE\_FORM

关闭窗口体。只有通过 MMC 指令打开对话框且为操作者提供了关闭对话框的软键功能时，该指令才是必须的。一般而言，会自动管理对话框，而无需另外加以关闭。

语法: `<CLOSE_FORM />`

示例:

```

<softkey_ok>
<caption>OK</caption>
<CLOSE_FORM />
<navigation>main_menu</navigation>
</softkey_ok>

```

### 9.4.4 POWER\_OFF

显示信息要求操作者关闭机床。显示文本固定保存在系统中。

语法:

```
<power_off />
```

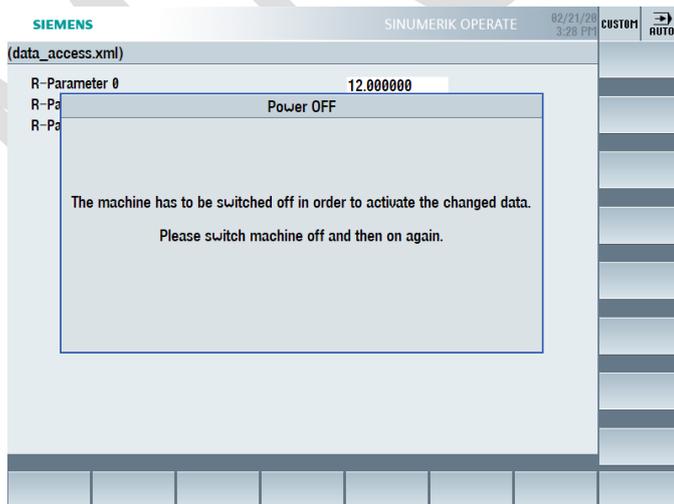
示例:

```

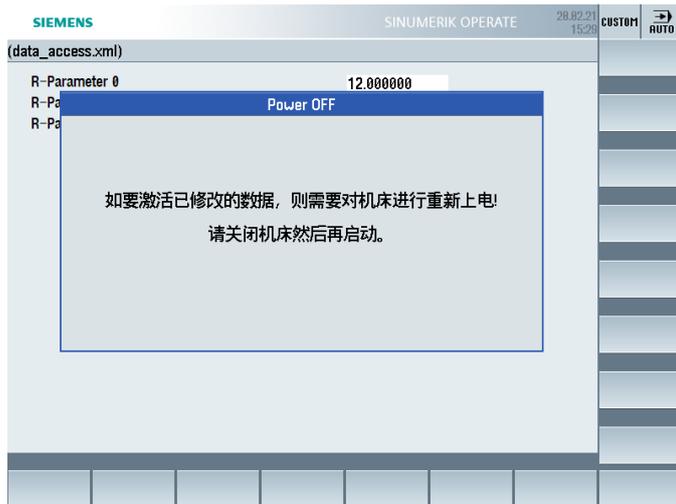
<softkey_ok>
<power_off />
</softkey_ok>

```

英文:



中文:



#### 9.4.5 MSGBOX

该指令会打开一个信息盒，其给分支的返回值可供使用。

语法: `<MSGBOX text="<Message>" caption="<caption>" retvalue="<variable name>" type="<button type>" />`

属性:

- Text: 文本
- Caption: 标题
- Retvalue: 返回应答信息 (0 – CANCEL, 1 – OK)
- Type: 应答类型  
("BTN\_OK", "BTN\_CANCEL", "BTN\_OKCANCEL")

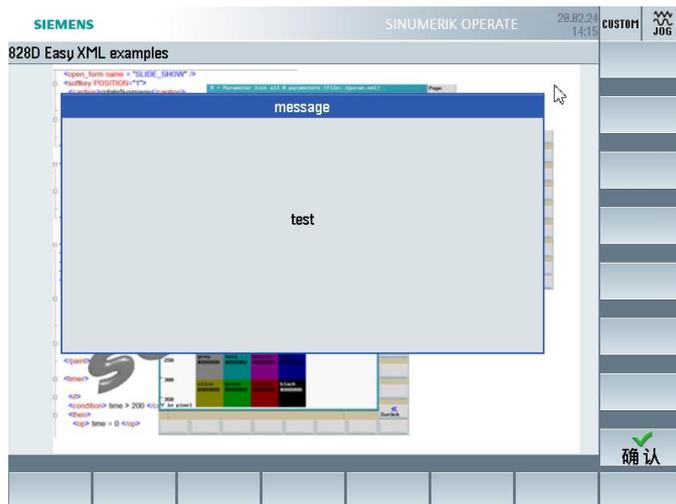
如果对属性 "text" 或 "caption" 使用报警编号 (8xxxx)，则信息盒会显示为该编号所保存的文本。

示例:

```

<softkey POSITION="11" >
  <caption>Message%Box </caption>
  <MSGBOX text="test" caption="message" retvalue="msg_ret" type="BTN_OK" />
  <if>
    <condition>msg_ret == 1 </condition>
    <then>
      <print text="OK was pressed" />
    </then>
    <else>
      <print text="Cancel was pressed" />
    </else>
  </if>
</softkey>

```



按下【确认】键，在状态栏监控变量值



#### 9.4.6 PRINT

标签功能:

1. 在窗体-对话框状态栏中输出文本,
2. 或者将文本复制到给定的变量中。

语法:

```
<PRINT name="变量名称" text="text %格式化"> Variable, ... </PRINT>
```

```
<PRINT text="text %格式化"> Variable, ... </PRINT>
```

属性:

- Name: 保存文本的变量名称 (可选)
- Text: 输出或要复制给变量的文本  
格式化: 以%标记开始, 如: %0.3f, 格式说明如下  
%【标记】【宽度】.【小数点位数】类型
- 标记: 确定输出格式的可选字符: "-", "0", "空格" (可选)
- 宽度: 确定输出宽度, 包含小数点后宽度, 不足时用空格符填充 (可选)
- 小数点位数: 使用浮点数时优化参数显示格式 (可选)
- 类型: d: 整数型, f: 浮点数, s: 字符串

<print>标签中的元素:

元素之间以", "隔开, 其数值会按顺序复制到文本的格式化变量中(%s)

示例:

<在窗体-对话框状态栏中输出文本>

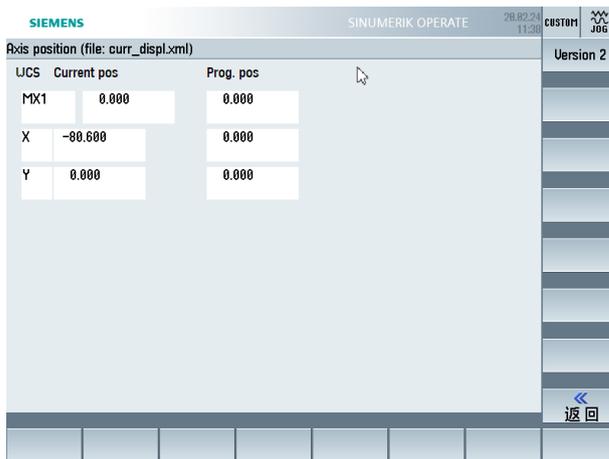
```
<let name="nc_type" />
<let name="nc_type_string" type="string" />
<!-- print version to info line -->
<print text="version: %s %d" >nc_type_string, nc_type </print>
```



<将文本复制到给定的变量中>

```
<let name="control_name" type="string"></let>
<let name="var_name_s" type="string"></let>
<let name="var_name_d" type="string"></let>
<let name="count" />
<print name="control_name" text="curr_pos%d">count</print>
<print name="var_name_d" text="nck/Channel/GeometricAxis/actProgPos[%d]">count</print>

<control name="$$$control_name" xpos="50" ypos="$ypos" refvar="$$$var_name_d" hotlink="true"
height="34" fieldtype="readonly" format="%9.3f" font="$font" time="superfast" color_bk="#ffffff"/>
```



#### 9.4.7 PROGRESS\_BAR

标签打开或关闭一根进度条。进度条显示在窗体的下方。每个窗体有一个。

语法: `<PROGRESS_BAR type="true/ or false" min="最小值" max="最大值" >当前值</PROGRESS_BAR>`

属性:

- type = "TRUE" - 打开进度条
- type = "FALSE" - 关闭进度条
- min (可选) - 最小值
- max (可选) - 最大值
- Value - 进度条的百分比位置



#### 9.4.8 SEND\_MESSAGE

该标签会向有效的 FORM 标签传递两个参数，触发 message 事件，在 form 中的 <Message> 标签中进行处理。

传递参数会自动复制到系统预定义变量 \$message\_par1 和 \$message\_par2 中。

语法:

`<SEND_MESSAGE>p1, p2</SEND_MESSAGE>`

结合 switch 方法示例

```

<message>
  <switch>
    <condition>$message_par1 </condition>
    <case value="1" >
      <op>message_par1 = $message_par1 </op>
      <op>message_par2 = $message_par2 </op>
    </case>
    <case value="2" >
      <op>message_t = $message_par2 </op>
    </case>
  </switch>
</message>

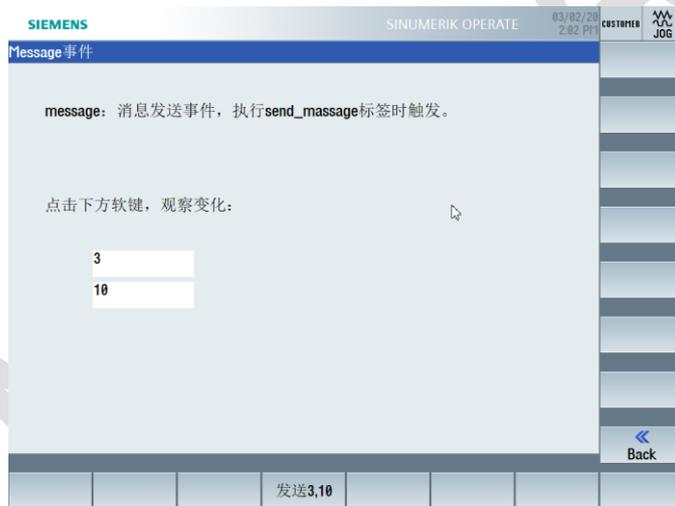
```

示例:

```

<menu>
  <softkey position="2" >
    <caption>发送3,10 </caption>
    <send_message>3, 10 </send_message>
  </softkey>
</menu>
<form>
  <MESSAGE>
    <op>var_1=$message_par1 </op>
    <op>var_2=$message_par2 </op>
  </MESSAGE>
</form>

```



参见: [/Easy XML Library/ 1\\_form 介绍 / form\\_event.xml](#)

#### 9.4.9 SHOW\_CONTROL

借助该标签可以指定是否显示控件。

语法:

```
<SHOW_CONTROL name="<name>" type="<type>" />
```

属性:

- Name: 控件的名称
- type = "TRUE" - 控件可见
- type= "FALSE" - 控件不可见 (隐藏)

示例:

#### 9.4.10 SLEEP

该标签会在指定的时间段内中断脚本的执行（单位：50ms）。

语法：

```
<SLEEP value="中断时间" />
```

示例：

```
<SLEEP value="30" />      等待时间 30x50ms=1.5 s。
```

参见：[/Easy XML Library/5\\_程序结构/2\\_循环结构/do\\_while.xml](#), [for.xml](#), [while.xml](#)

#### 9.4.11 STOP

脚本文件解释在该位置中断

#### 9.4.12 SWITCHTOAREA

该标签从用户操作区切换至指定的操作区。参数指定为属性值。

语法：

```
<switchToArea name="area" args="argument" />
```

属性：

name：要跳转到的操作区名称，与操作区一致：

- AreaMachine - 机床
- AreaParameter - 参数
- AreaProgramEdit - 编辑器
- AreaProgramManager - 程序管理器
- AreaDiagnosis - 诊断
- AreaStartup - 调试

args（预留）。

示例：

```
<switchToArea name="AreaMachine" />
```

#### 9.4.13 SWITCHTODIALOG

目录 `siemens\sinumerik\hmi\template\cfg\sl<操作区>_oem.xml` 下提供相应的模板。

将属于该操作区的 XML 文件复制到目录 `oem\sinumerik\hmi\cfg` 下并进行如下编辑：

使用函数 `switchToDialog` 作为软键响应。使用关键词 `area` 中输入 `CustomXML` 并在 `dialog` 中输入 `SIEECustomDialog` 作为自变量。

语法：

```
<FUNCTION args="-area CustomXML -dialog SIEECustomDialog -mainModule <name of the main module>
-entry <menu name>" name="switchToDialog"/>
```

示例：

将 Easy XML 应用链接至诊断区：使用 “`dg.xml`” 作为脚本文件名称。在该文件中调用名称为 `main` 的菜单。

```
<MENU name="DgGlobalHu">
<ETCLEVEL id="0">
```

```

<SOFTKEYGROUP name="GroupEtc">
<SOFTKEY position="7">
<PROPERTY name="textID" type="QString">DG_SK</PROPERTY>
<PROPERTY name="translationContext" type="QString">EASY_XML</PROPERTY>
<FUNCTION args="-area CustomXML -dialog SIEECustomDialog -mainModuledg.xml -entry main"
name="switchToDialog"/>
</SOFTKEY>
</SOFTKEYGROUP>
</ETCLEVEL>
</MENU>

```

注: Operate V04.08 SP4 及以上版本支持该功能。

#### 9.4.14 SWITCHTODYNAMICTARGET

如果将之前的对话框定义为动态跳转目标, 则 SWITCHTODYNAMICTARGET 标签会激活该对话框并结束脚本编辑。

语法:

```
<switchToDynamicTarget />
```

## 9.5 窗体显示

### 9.5.1 CAPTION

该标签为标题标签, 用于窗体标题, 按钮标题等。

在 INIT (初始化) 标签内使用该标签可定义窗体标题。标题行可划分为多列。

为了划分标题行, 在输出文本前应使用属性 "define\_section" 对标签 "caption" 进行编程。

属性 "define\_section" 确定列数。通过 "property" 属性定义每一列的长度、起始位置和文本方向。位置和长度说明是基于表格宽度的百分数值。

属性值 "value" 表示列编号 (从零开始)

```

<caption define_sections="3">
  <property value="0" length="20" position="2" alignment="left" />
  <property value="1" length="20" position="79"
alignment="right" />
</caption>

```

然后可以借助列索引指定属性 index 来指定列文本。

```

<caption index="0">Spalte1</caption>
<caption index="1">Spalte2</caption>

```

语法:

```
<CAPTION>Titel</CAPTION>
```

示例:

```
<CAPTION>my first dialogue</CAPTION>
```

### 9.5.2 TEXT

该标签用来在窗体上指定的位置上显示文本。

如果使用报警编号 (8xxxx), 则对话框会显示为该编号所保存的文本。

语法:

```
<TEXT xpos = "<X 坐标>" ypos = "<Y 坐标>" color="#RRGGBB" > Text </TEXT>
```

Color: 文本颜色 (详见附录: 颜色代码)

Text: 要显示的文本

示例:

<常量形式>

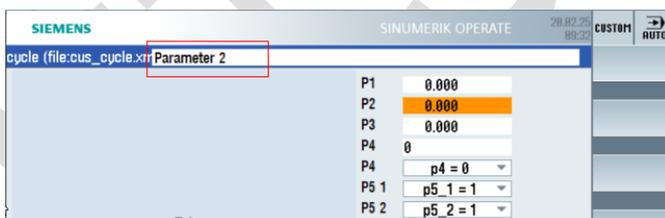
```
<text xpos = "50" ypos = "38" >50 </text>
<text xpos = "170" ypos = "310" >green </text>
<text xpos = "310" ypos = "310" color = "#FFFFFF" >black </text>
<text xpos = "310" ypos = "325" color = "#FFFFFF" >#000000 </text>
```

<变量形式>

```
</let name="cursor_texts" dim="7" type="string" >
{"Parameter 1"},
{"Parameter 2"},
{"Parameter 3"},
{"Parameter 4"},
{"Parameter 4 combobox"},
{"Parameter 5 digits 0-3"},
{"Parameter 5 digits 4-7"},
</let>
<op> index = $focus_item_data - 1000 </op>
<text xpos="142" ypos="6" color="#000000" >$$cursor_texts[$index] </text>
```

\$: 变量实际值。如上: 当变量 index=1 时, 表达式为 cursor\_texts[1]

\$\$\$: 取表达式的内容。如上: \$\$\$cursor\_texts[1], 文本则显示 Parameter 2



### 9.5.3 BOX

该标签可以在指定的位置以指定的颜色绘制一个填满的矩形。

语法:

```
<BOX xpos = "<X 坐标>" ypos = "<Y 坐标>" width = "<X 长度>" height = "<Y 长度>" color = "<颜色代码>" />
```

示例:

```

<box xpos="80" ypos="40" width="350" height="20" color="#00FF00" />
<text xpos = "210" ypos = "42" >#00FF00 </text>
<box xpos="80" ypos="70" width="350" height="20" color="#FFFF00" />
<text xpos = "210" ypos = "72" >#FFFF00 </text>
<box xpos="80" ypos="100" width="350" height="20" color="#FF0000" />
<text xpos = "210" ypos = "102" >#FF0000 </text>

...

<box xpos="80" ypos="130" width="350" height="20" color="#0000FF" />
<text xpos = "210" ypos = "132" >#0000FF </text>
<box xpos="80" ypos="160" width="350" height="20" color="#FFFFFF" />
<text xpos = "210" ypos = "162" >#FFFFFF </text>
<box xpos="80" ypos="190" width="350" height="20" color="#000000" />
<text xpos = "210" ypos = "192" color="#FFFFFF" >#000000 </text>

```



参见: /Easy XML Library/ 7\_颜色展示 /xmldial.xml

#### 9.5.4 IMG

该标签用来在窗体指定的位置上显示图像。支持 BMP、PNG、JPG 图像格式。

语法:

```

<IMG xpos = "<X 坐标>" ypos = "<Y 坐标>" height="高度" width="宽度" name = "<名称>" xrot= "X 轴角度"
  yrot= "Y 轴角度" zrot= "Z 轴角度" transparent= "#RRGGBB" />

```

属性:

- Name: 图像完整的路径名称。路径名称只能用小写字母。
- Xrot/yrot/zrot: 沿坐标轴的空间旋转角度
- Transparent: 利用该属性可以将图片背景色 (单色) 去掉, 变成透明的。
- Height/Width: 使用属性 width 和 height 来确定图片显示的大小 (单位: 像素)。
- AspectRatioMode: 长宽比。四种模式: Ignore, keep, keepByExpanding, ExpandingForRotation

"Ignore": 忽略长宽比, 按给定的高度和宽度显示。

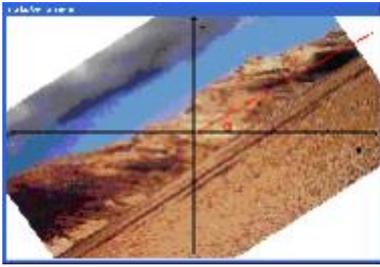
"Keep": 保持长宽比, 以高度/宽度中, 最小的比例显示。

"KeepbyExpanding": 保持长宽比, 以高度/宽度中, 最大的比例显示。

"ExpandingForRotation": 保持长宽比, 并显示图片旋转后溢出矩形框的部分。

示例：图片绕 Z 轴旋转 34°。

```
<img xpos = "5" ypos = "23" name = "f:/appl/pic3.bmp" height="355" width="550" zrot="34" > </img>
```



示例：

```
<paint>
<img xpos = "008" ypos = "030" name = "test.bmp" height="100" width="80" AspectRatioMode="KeepbyExpanding" />
<img xpos = "008" ypos = "120" name = "test.bmp" height="100" width="80" AspectRatioMode="Ignore" />
<img xpos = "200" ypos = "120" name = "test.bmp" height="100" width="80" AspectRatioMode="Keep" />
<img xpos = "200" ypos = "150" name = "test.bmp" height="100" width="80" AspectRatioMode="Keep" />

<img xpos = "138" ypos = "220" name = "pic1.bmp" height="155" width="150" />
<op> zrot = 45.0 </op>
<img xpos = "355" ypos = "023" name = "pic1.bmp" height="155" width="150" zrot="$zrot" AspectRatioMode="ExpandingForRotation" />
<img xpos = "355" ypos = "220" name = "pic1.bmp" height="155" width="150" zrot="$zrot" transparent="#000000" />
<img xpos = "008" ypos = "220" name = "pic2.png" height="155" width="150" transparent="#4472C4" />
</paint>
```

示例：

Transparent="" 数值需与图片背景色一致，否则无法消除背景色。

消除前	消除后	
transparent="#000000"	transparent="#4472C4"	transparent="#ED7D31"
		

### 9.5.5 HELP\_CONTEXT

该标签用来确定所要调用的帮助主题。在 INIT 初始化数据块中进行编程。

语法

```
<help_context name="xxx.html" />
```

属性：

name: 在线帮助关联的 html 文件名称。

示例

```
<init>
<help_context name="easyxml_myhelp.html" />
...
</init>
```

更多知识请参考《SINUMERIK Operate》oem 专用在线帮助 章节。下载连接如下

<https://support.industry.siemens.com/cs/document/109761932/sinumerik-828d-sinumerik-operate?dti=0&lc=en-WW>

## 9.5.6 语言文本

### 9.5.6.1 定义

通过在窗体中使用文本标签（加前缀\$\$，如：\$\$TEXT1），所有在窗体中的标签文本都会被替代为当前语种的文本显示。此文本会和已提供的每个语种文本标签一同保存在一个 UTF8 格式的文本文件 (\*.ts) 中。

EASY\_XML 默认的语言文本为：oem\_xml\_screens\_<语种缩写>.ts

（语种缩写：chs-简体中文，eng-英语，deu-德语，更多缩写见附录：语种缩写表）

如果默认语言文本 oem\_xml\_screens\_xxx.ts 无效，则可按照如下两种方法重新声明：

方法 1：在 xmldial.xml 中的 <DialogGui> 标签中使用 textfile 属性声明（推荐）

```
<DialogGui textfile="oem_xml_screens">
```

...

```
</DialogGui>
```

方法 2：在 slamconfig.ini 文件中声明：

```
<安装路径>/oem/sinumerik/hmi/cfg 或 /user/sinumerik/hmi/cfg
```

原理：

可在配置文件 “slamconfig.ini” 中为已在文件 “systemconfiguration.ini” 中配置过的操作区域创建一个段落。此段落名称必须包含有已配置的操作区域的名称，比如 AreaOEM。

systemconfiguration.ini 中对应 EasyXML 的区域为：[Custom]

在 slamconfig.ini 中增加如下声明：

```
[Custom]
Picture=hd_custom.png
SoftkeyPosition=12
Visible=false
  TextFile=oem_xml_screens
```

*注：*slamconfig.ini 也可从 siemens/Sinumerik/hmi/template/cfg 中获得。

上述两种声明方法同样适用于自定义语言文本。只需要将文本名称 oem\_xml\_screens 更改为自定义语言文本名称。

### 9.5.6.2 自定义语言文本

自定义语言文本名称：main\_form\_screens

中文语言文本：main\_form\_screens\_chs.ts

英语语言文本：main\_form\_screens\_eng.ts

语法：

（定义全局语言文本）

```
<DialogGui textfile="main_form_screens">
```

...

```
</DialogGui>
```

或 (定义局部语言文本)

```
<form name="main_form" textfile="main_form_screens">
```

...

```
</form>
```

或

```
<menu name="main" textfile="main_form_screens">
```

...

```
</menu>
```

### 9.5.6.3 语言文本格式 (.ts)

示例: (蓝色斜体为用户自定义内容)

*main\_form\_screens\_eng.ts*

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE TS>
<TS>
<context>
  <name>EASY_XML</name>
  <message>
    <source>MAIN_FORM_TITLE</source>
    <translation>user text file title</translation>
  </message>
  <message>
    <source>test1</source>
    <translation>text1 from text file</translation>
  </message>
</context>
</TS>
```

说明:

encoding="UTF-8": 源码文件以 UTF-8 (万国码的一种实现方式) 编码方式处理;  
(UTF-8 的更多知识请自行查补)

标签:

*<!DOCTYPE TS>* 标签: 文档格式描述标签;

*<TS>* 标签: ts 主体

*<context>* 标签: 文本上下文, 属性: name, message, EASY\_XML 的语言文本, context.name 为 EASY\_XML;

*<message>* 标签: 属性: source: 文本源代码, translation: 替代文本。其中文本源代码与界面脚本 xml 文件中的文本变量必须一致, 不含前缀 \$\$。

注: <TS>标签中可包含多个<context>标签, <context>标签中可包含多个<message>标签。

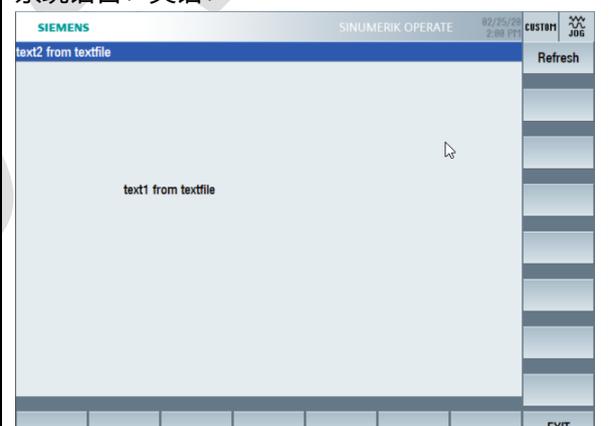
### 9.5.6.4 示例

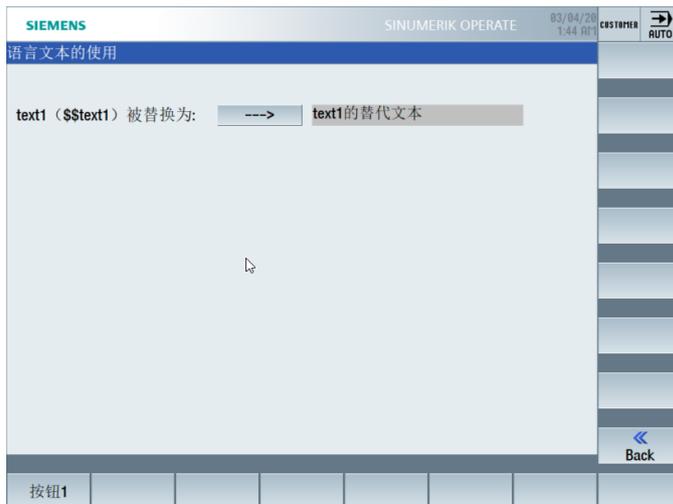
关联语言文本: *textfile="myscreens"*

语言文本:

<i>myscreens_eng.ts</i>	<i>myscreens_chs.ts</i>
<i>&lt;?xml version="1.0" encoding="UTF-8" ?&gt;</i>	<i>&lt;?xml version="1.0" encoding="UTF-8" ?&gt;</i>
<i>&lt;!DOCTYPE TS&gt;</i>	<i>&lt;!DOCTYPE TS&gt;</i>
<i>&lt;TS&gt;</i>	<i>&lt;TS&gt;</i>
<i>&lt;context&gt;</i>	<i>&lt;context&gt;</i>

<pre> &lt;name&gt;EASY_XML&lt;/name&gt; &lt;message&gt; &lt;source&gt;SK8&lt;/source&gt; &lt;translation&gt;EXIT&lt;/translation&gt; &lt;/message&gt; &lt;message&gt; &lt;source&gt;SK9&lt;/source&gt; &lt;translation&gt;Refresh&lt;/translation&gt; &lt;/message&gt; &lt;message&gt; &lt;source&gt;MY_TEXT1&lt;/source&gt; &lt;translation&gt;text1 from textfile&lt;/translation&gt; &lt;/message&gt; &lt;message&gt; &lt;source&gt;MY_TEXT2&lt;/source&gt; &lt;translation&gt;text2 from textfile&lt;/translation&gt; &lt;/message&gt; &lt;message&gt; &lt;source&gt;85066&lt;/source&gt; &lt;translation&gt;this is the text for 85066&lt;/translation&gt; &lt;/message&gt; &lt;message&gt; &lt;source&gt;85067&lt;/source&gt; &lt;translation&gt;this is the text for 85067&lt;/translation&gt; &lt;/message&gt; &lt;/context&gt; &lt;/TS&gt; </pre>	<pre> &lt;name&gt;EASY_XML&lt;/name&gt; &lt;message&gt; &lt;source&gt;SK8&lt;/source&gt; &lt;translation&gt;退出&lt;/translation&gt; &lt;/message&gt; &lt;message&gt; &lt;source&gt;SK9&lt;/source&gt; &lt;translation&gt;刷新&lt;/translation&gt; &lt;/message&gt; &lt;message&gt; &lt;source&gt;MY_TEXT1&lt;/source&gt; &lt;translation&gt;来自语言文本的文本 1&lt;/translation&gt; &lt;/message&gt; &lt;message&gt; &lt;source&gt;MY_TEXT2&lt;/source&gt; &lt;translation&gt;来自语言文本的文本 2&lt;/translation&gt; &lt;/message&gt; &lt;message&gt; &lt;source&gt;85066&lt;/source&gt; &lt;translation&gt;这是 85066 显示的文本&lt;/translation&gt; &lt;/message&gt; &lt;message&gt; &lt;source&gt;85067&lt;/source&gt; &lt;translation&gt;这是 85067 显示的文本&lt;/translation&gt; &lt;/message&gt; &lt;/context&gt; &lt;/TS&gt; </pre>
--	---

<p>界面脚本文件: gui_caption.xml</p> <pre> &lt;menu name = "menu_gui_caption" &gt; &lt;open_form name = "gui_caption" /&gt;  &lt;softkey position="8" &gt; &lt;caption&gt;\$\$\$K8&lt;/caption&gt; &lt;navigation&gt;main&lt;/navigation&gt; &lt;/softkey&gt;  &lt;softkey position="9" &gt; &lt;caption&gt;\$\$\$K9&lt;/caption&gt; &lt;/softkey&gt;  &lt;/menu&gt;  &lt;form name="gui_caption" textfile="myscreens" &gt; &lt;init&gt; &lt;caption&gt;\$\$MY_TEXT2&lt;/caption&gt; &lt;/init&gt;  &lt;paint&gt; &lt;text xpos = "120" ypos="158"&gt;\$\$MY_TEXT1&lt;/text&gt; &lt;/paint&gt;  &lt;/form&gt; </pre>	<p>系统语言: 英语:</p>  <p>系统语言: 中文:</p> 
--	---



参见: [/Easy XML Library/6\\_语言文本 / languagetext.xml](#)

## 9.6 条件-循环指令

### 9.6.1 FOR 循环

1. 运用标签< INIT >初始化。
2. 运用标签< CONDITION > 作为布尔型条件表达式。如果为 false, 则结束 For 循环。
3. 执行后续的指令。
4. 运用标签< INCREMENT > 继续。
5. 跳转 2。

所有在 INIT、CONDITION 和 INCREMENT 分支中使用的变量都要在 For 循环的外部创建。

语法:

```
<FOR>
<INIT>...</INIT>
<CONDITION>...</CONDITION>
<INCREMENT>...</INCREMENT>
```

指令

...

```
</FOR>
```

示例:

```
<LET name = "count">0</LET>
<FOR>
  <INIT>
    <OP> count = 0</OP>
  </INIT>
  <CONDITION> count <= 7 </CONDITION>
  <INCREMENT>
    <OP> count = count + 1 </OP>
  </INCREMENT>
  <OP> "plc/qb10" = 1+ count </OP>
</FOR>
```

### 9.6.2 While 循环

While 循环用于按顺序多次执行指令。在处理指令序列前对条件进行检查，只要满足条件，就会循环执行。

语法：

```
<WHILE>  
<CONDITION>...</CONDITION>
```

指令

...

```
</WHILE>
```

示例：

```
<while>  
  <condition>count < num_mach_axis</condition>  
  <op>ax_name[$count] = "nck/Channel/GeometricAxis/name[$count]" </op>  
  <op>count = count + 1</op>  
</while>
```

```
<paint>  
  <text xpos = "10" ypos = "90">User frame line shift rotation</text>  
  <text xpos = "10" ypos = "114">G56</text>  
  <text xpos = "40" ypos = "114">$$$ax_name[0]</text>  
  <text xpos = "10" ypos = "134">G57</text>  
  <text xpos = "40" ypos = "134">$$$ax_name[1]</text>  
  <text xpos = "10" ypos = "154">G55</text>  
  <text xpos = "40" ypos = "154">$$$ax_name[0]</text>  
</paint>
```

### 9.6.3 Do-while 循环

语法：

```
<DO_WHILE>
```

指令

...

```
<CONDITION>...</CONDITION>
```

```
</DO_WHILE>
```

示例：

```
<do_while>  
  <condition>loop_start_double < loop_end_double</condition>  
  
  <!-- instructions -->  
  <PRINT text="%f"> loop_start_double </PRINT>  
  
  <OP>loop_start_double = loop_start_double + loop_increment_double </OP>  
  <!-- increment -->  
</do_while>
```

#### 9.6.4 Switch 指令

指令 SWITCH 表示多项选择。表达式与 CASE 标签的常数（*不支持字符串比较*）进行比较。与常数一致，则执行 CASE 标签中的指令。与常数均不一致，则执行 DEFAULT 指令。

语法：

```
<SWITCH>
<CONDITION> 表达式 </CONDITION>
<CASE value="<常数 1>">
...
</CASE>
<CASE value="<常数 2>">
...
</CASE>
...
<DEFAULT>
...
</DEFAULT>
</SWITCH>
```

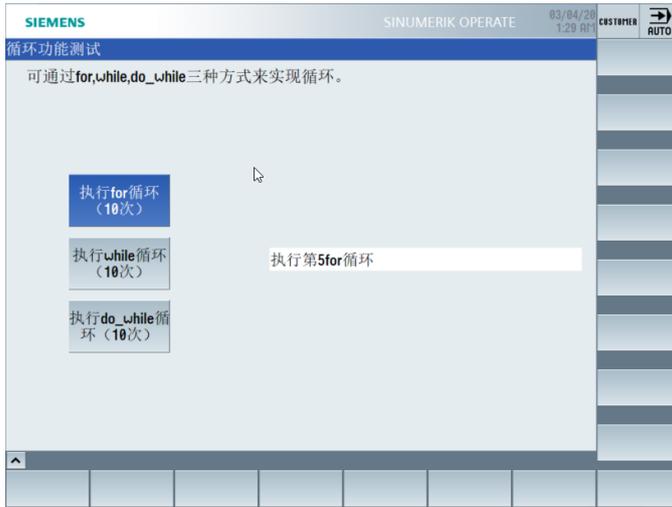
示例：

```
<switch>
<condition>$message_par1 </condition>
<case value="1" >
  <if>
    <condition>$message_par2 == 1 </condition>
    <then>
      <control name = "c1" fieldtype="edit"/>
    </then>
    <else>
      <control name = "c1" fieldtype="readonly"/>
    </else>
  </if>
</case>
</switch>
```

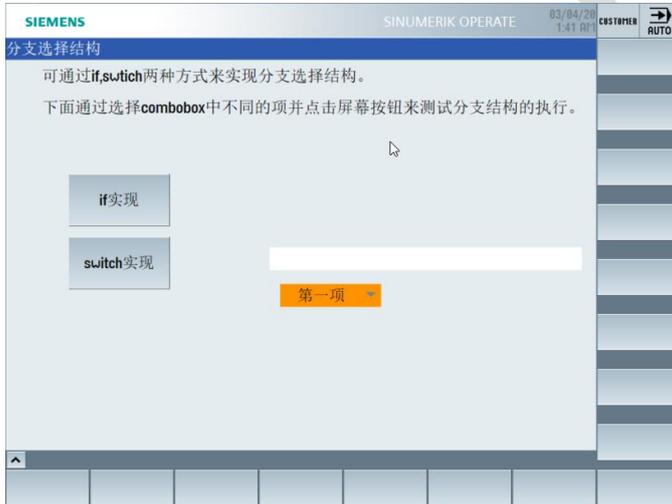
#### 9.6.5 Break

有条件中断一个循环，执行至该指令时自动跳出。

## 9.6.6 示例



参见: [/Easy XML Library/5\\_程序结构/2\\_循环结构/do\\_while.xml](#), [for.xml](#), [while.xml](#)



参见: [/Easy XML Library/5\\_程序结构/3\\_选择结构/if.xml](#), [switch.xml](#)

## 9.7 信号处理

### 9.7.1 REQUEST

该标签仅在窗体的<INIT>标签中进行处理。

标签可以为 hotlink="true"的变量建立一个周期性读取服务。这样可以缩短对未与控件相连的变量的存取时间。

如果要在值变化时自动调用一个函数，则应将函数的名称作为其它属性加以定义。

语法：

```
<REQUEST name = "<NC-Variable>" />
```

或

```
<REQUEST name = "<NC-Variable>" function="<function name>" />
```

示例：

```
<request name = "plc/mb10" />
```

或

```
<function_body name="my_function" >
<print text="value changed" />
</function_body>

...
<request name = "plc/mb10" function="my_function"/>
```

可参考 graphicbox 控件章节及示例。

## 9.8 分析 XML 文件

标签 “XML\_PARSER” 可用于分析 XML 文件。

### <工作原理>

1. 解析器编译 XML 文件并调用定义的回调函数。每个回调函数都属于一个预定义事件。
2. 编程人员可在该函数内部处理 XML 文件。
3. 出现一个事件时，解析器会调用回调函数并检查函数的返回值。函数返回值 “true” 时，解析器会继续进程。

### <预定义事件>

- start document：解析器打开文档并开始分析。
- end document：解析器关闭文档。
- start element：解析器找到了一个元素并创建了一个含所有属性和属性值的列表。这些列表被转送给回调函数。
- end element：找到元素末尾。
- characters：解析器转送元素的所有字符。
- error：解析器发现了一个语法错误。

### <属性>

- Name：要分析的 XML 文件的路径+名称
- StartElementHandler：
- EndElementHandler：
- CharactersHandler：
- ErrorHandler：（可选）
- DocumentHandler：（可选）

示例：

```
<XML_PARSER name="f:\app\xml_test.xml">
  <!-- standard handler -->
  <property startElementHandler="startElementHandler" />
  <property endElementHandler="endElementHandler" />
  <property charactersHandler="charactersHandler" />
  <!-- optional handler -->
  <property errorHandler="errorHandler" />
  <property documentHandler="documentHandler" />
</XML_PARSER>
```

### 9.8.1.1 startElementHandler

函数参数：

*tag\_name*: 标签名称

*num*: 找到的属性数。

系统变量

*\$xmlAttribute* 具有一定数目的通过 *num* 给定的单元的字符串数组。

*\$xmlValue* 属性值范围为 0-*num* 的字符串数组。

示例:

```
<function_body name="startElementHandler" return="true" parameter="tag_name, num">
  <print text="attribute name %s"> $xmlAttribute[0]</print>
  <print text="attribute value %s"> $xmlValue[0]</print>
</function_body>
```

### 9.8.1.2 endElementHandler

函数参数

*tag\_name*: 标签名称

示例:

```
<function_body name="endElementHandler" parameter="tag_name">
  <print text="name %s"> tag_name </print>
</function_body>
```

### 9.8.1.3 charactersHandler

函数参数: 无

系统变量:

*\$xmlCharacters* 含数据的字符串

*\$xmlCharactersStart* 始终为 0

*\$xmlCharactersLength* 字节数

示例:

```
<function_body name="charactersHandler" return="true" >
  <print text="chars %s"> $xmlCharacters </print>
</function_body>
```

### 9.8.1.4 documentHandler

函数参数

*state* 1 start document, 2 end document

### 9.8.1.5 errorHandler

函数参数 无

## 系统变量

`$xmlErrorString` 包含无效行 (系统变量)

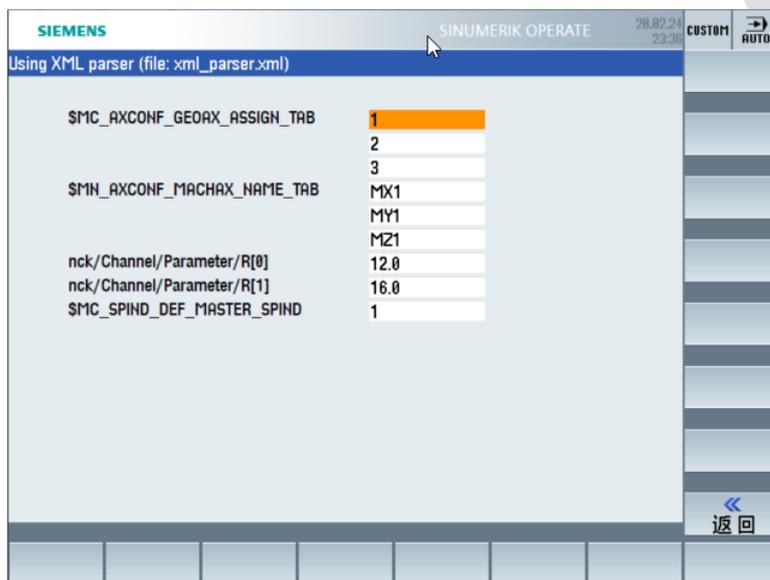
### 9.8.1.6 回调结果

`$return` 如果为 1 (true), 分析程序会继续分析文件

示例:

```
<function_body name="errorHandler" >
  <print text="error %s">$xmlErrorString</print>
</function_body>
```

### 9.8.1.7 示例



参见: 828D toolbox / ./ Custom Screen Sample / xml\_parser.xml

## 9.9 系统函数

### 9.9.1 读写系统数据

函数	说明	语法
Ncfunc cap read	读系统数据 - return - 执行状态 = 0 - 正常 = 1 - 变量的读取无法进行 - rows - 数组中额外读取的行数 (可选)	<function name="nfunc.cap.read" return="error" rows="23"> var1, "address"</function>  示例: <function name="nfunc.cap.read" return="error"> 3, "drive/cu/p0009"</function>
Ncfunc cap write	写系统数据 - return - 执行状态 = 0 - 正常 = 1 - 变量的读取无法进行 - rows - 数组中额外读取的行数	<function name="nfunc.cap.read" return="error"> rows, "address"</function>  示例: <let name="cu_p978" dim="25" ></let> ...

	(可选)	<function name="ncfunc.cap.write" return="error" rows="23">cu_p978, "drive/cu/p0978[0, 1]"</function>
Ncfunc Get drive by axis name	通过轴名称来给出电机模块的寻址下标 - str - 轴名称	<function name="ncfunc.getdrvbyax_name" return="<index>">str</function>
Ncfunc Get drive by axis index	通过轴下标来给出电机模块的寻址下标 - index - 轴下标	<function name="ncfunc.getdrvbyax_idx" return="<index>">index</function>
Ncfunc Get drive by drive name	通过给定电机模块名称来给出电机模块的寻址下标。 - string - 电机模块名称	<function name="NCFUNC.GETDRVBYDRV_NAME" return="<index>">string</function>
Ncfunc Get drive by bud address	通过给定电机模块总线地址来给出电机模块的寻址下标 - bus - 总线编号 - slave - 从站编号 - component - 组件编号	<function name="NCFUNC.GETDRVBYBUS_ADDR" return="<index>">bus,slave,component</function>
Ncfunc bico to int	将指定的 BICO 格式的字符串转换为整型值	<function name="ncfunc.bicotoint" return="integer variable">bico-string</function>
Ncfunc int to bico	将整型值转换为 BICO 格式的字符串	<function name="ncfunc.inttobico" return="string variable">integer variable</function>
Ncfunc is bico str valid	当结果是以 BICO 格式给定的字符串时，该函数会提供零值。	<function name="ncfunc.isbicostrvalid" return="integer variable">string variable</function>
Ncfunc password	该函数用来设置或取消 NC 口令等级。 • 设置口令： 所需口令等级的口令将作为参数给定。 • 删除口令： 空字符串会取消口令等级。	<function name="ncfunc.password">pw_string</function>  示例： 设定口令 <function name="ncfunc.password" >_T"CUSTOMER" </function> 删除口令 <function name="ncfunc.password" >_T""</function>

参见: /Easy XML Library/ 4\_系统预定义功能 / 1\_NC 功能\_读写变量等 / 1\_读写 NC 变量 / ncvvar\_ncap.xml

### 9.9.2 选项处理

增加元素	在列表末尾添加一个新元素 - control name - 控件名称 - item - 要添加的表达式 - itemdata - 整数值, 由用户确定	语法: <function name="control.additem"> control name, item, itemdata </function>  <i>只能用于控件类型 "listbox" 和 "graphicbox"</i>
插入元素	可在指定位置添加一个新元素 - control name - 控件名称 - index - 位置从零开始 - item - 要添加的表达式	语法: <function name="control.insertitem"> control name, index, item, itemdata </function>  <i>只能用于控件类型 "listbox"</i>

	<ul style="list-style-type: none"> <li>- itemdata - 整数值; 由用户确定</li> </ul>	
删除元素	可删除指定位置上的元素 <ul style="list-style-type: none"> <li>- control name - 控件名称</li> <li>- index- 下标从 0 开始</li> </ul>	语法: <code>&lt;function name="control.deleteitem"&gt; control name, index &lt;/function&gt;</code>  <i>只能用于控件类型 "listbox"</i>
添加列表	可向控件中添加一个元素列表 <ul style="list-style-type: none"> <li>- control name - 控件名称</li> <li>- list - 字符串变量</li> </ul> 多个元素之间用 \n 隔开	语法: <code>&lt;function name="control.loaditem"&gt; control name, list &lt;/function&gt;</code>  <i>只能用于控件类型 "listbox" 和 "graphicbox"</i>
清空列表框	可清除整个列表框元素 <ul style="list-style-type: none"> <li>- control name - 控件名称</li> </ul>	语法: <code>&lt;function name="control.empty"&gt; control name, &lt;/function&gt;</code>

参见: [/Easy XML Library/3\\_控件展示/1\\_控件展示/12\\_listbox.xml.xml](#)

### 9.9.3 文件处理

读文件 Read from file	可以将指定文件的内容读到一个字符串变量中。或者可以将待读取的字符数量指定为第二参数。	语法: <code>&lt;function name="doc.readfromfile" return="&lt;string var"&gt; progname, number of characters &lt;/function&gt;</code>  返回值: Return: 返回读取的字符串 参数: Progname: 文件名 Number of characters: 待读取的字符数量: 字节 (可选)
写文件 Write to file	可以将一个字符串变量的内容写入指定文件中。文件名只能用小写字母。	语法: <code>&lt;function name="doc.writetofile" &gt; progname, str1 &lt;/function&gt;</code>  参数: progname - 文件名 str1 - 字符串
删除文件 Remove	可以从目录中删除指定的文件。文件名只能用小写字母。	语法: <code>&lt;function name="doc.remove" &gt; progname &lt;/function&gt;</code>  参数: progname - 文件名
提取脚本文件 Load script	将零件程序中包含的对话框说明复制到指定的局部变量中。	语法: <code>&lt;function name="doc.loadscript" return="&lt;name of script variable"&gt; progname, _T"dialog part name", main menu &lt;/function&gt;</code>  返回值: return - 保存有所提取的脚本的变量 参数: progname - 文件名

		main menu - 找到的菜单名称复制到该变量中 dialog part name - 包含对话框说明的标签名称
文件存在 exist	如存在, 返回值为 1。 文件名需为小写字母。	语法: <function name="doc.exist" return="<int_var>" > progname </function>
程序选择 Nc select	可以选择将要处理的程序。程序必须保存在 NC 文件系统中。	语法: <function name="ncfunc.select"> progname </function>
相对路径	脚本 xml 文件所在路径, 文件名前加"."代表本级目录文件夹	_"\test.mpf"

参见: [/Easy XML Library/ 4\\_系统预定义功能 / 2\\_文件处理 \(字符串处理等\) / filehandler\\_1.xml](#)

示例:

<let name = "my_var" type="string" ></let>
<function name="doc.readfromfile" return="my_var"> _T"\mpf\test.mpf" </function>
<function name="doc.readfromfile" return="my_var"> _T"f:\appl\test.mpf" </function>
<function name="doc.readfromfile" return="my_var"> _T".\test.mpf" </function>
<function name="doc.loadscript" return="contents">prog_name, _T"main_dialog", entry</function>
<function name="ncfunc.select"> _T"\mpf\test.mpf" </function>

#### 9.9.4 字符串处理

函数	说明	语法
String find	正向查找字符串 - string - 字符串变量 - find string - 要查找的字符串 - startindex - 起始索引 (可选)	<function name="string.find" return="<int val>"> str1, find string </function>
String reverse find	反向查找字符串 - string - 字符串变量 - find string - 要查找的字符串 - startindex - 起始索引 (可选)	<function name="string.reversefind" return="<int val>"> str1, find string </function>
String GetAt	读取特定位置的一个字符 - str - 字符串 - index - 待读取字符的下标, 从零开始	<function name="string.getat" return="<result string"><string>, <index></function>
String SetAt	在特定位置写一个字符 - str - 字符串 - char - 应在特定位置写入的字符 - index - 目标字符串的下标, 从零开始	<function name="string.setat" > <destination string>, <character string>, <index> </function>
String Split	将一个字符串分隔为多个子字符串并将其复制到给定的字符串数组中。 分隔符不会存储在子字符串中。 分割数量超出数组时自动扩展数组。 - str - 字符串 - char - 包含分隔符的变量名称 - number - 包含此功能执行完毕后产生的	<function name=" string.split" return="<result string array"> <string>, <char>, <number> </function>  示例: <function name="string.split" return="strlist"> _T"brown:green;blue:red", _T";", str_num </function>

	子字符串数量的变量名称。	
String to compare	对两个字符串从书写方式上进行比较。相同时返回值零。不同返回差值 - str1 - 字符串 - str2 - 比较字符串	<function name="string.cmp" return ="<int var>" > str1, str2 </function>
string.icmp	不区分大/小写的字符串比较 - str1 - 字符串 - str2 - 比较字符串	<function name="string.icmp" return ="<int var>" > str1, str2 </function>
String left	提取字符串 1 中第一个的 nCount 字符并将其复制到返回变量中。 - str1 - 字符串 - nCount - 字符数量	<function name="string.left" return="<result string>"> str1, nCount </function>
String right	提取字符串 1 中最后一个 nCount 字符并将其复制到返回变量中。 - str1 - 字符串 - nCount - 字符数量	<function name="string.right" return="<result string>"> str1, nCount </function>
String middle	从下标 i 开始提取字符串 1 中指定数量的字符并将其复制到返回变量中。 - str1 - 字符串 - i - 起始下标 - nCount - 字符数量	<function name="string.middle" return="<result string>"> str1, i, nCount </function>
String length	获取字符串的字符数量 - str1 - 字符串	<function name="string.length" return="<int var>"> str1 </function>
Strings to replace	使用新的字符串替代所有查找到的字符串部分。 - string - 字符串变量 - find string - 要替代的字符串 - new string - 新字符串	<function name="string.replace"> string, find string, new string </function>
Strings to remove	可以删除所有查找到的字符串部分。 - string - 字符串变量 - remove string - 要删除的字符串部分	<function name="string.remove"> string, remove string </function>
Strings to insert	可以在指定下标中插入一个字符串。 - string - 字符串变量 - index - 下标 (从零开始) - insert string - 要插入的字符串	<function name="string.insert"> string, index, insert string </function>
String delete	可以从指定的起始位置开始删除确定数量的字符。 - string - 字符串变量 - start index - 起始下标 (从零开始) - nCount - 要删除的字符数量	<function name="string.delete"> string, start index , nCount </function>
String trim left	可以从字符串中删除打头的空格。 - str1 - 字符串变量	<function name="string.trimleft" > str1 </function>
String trim right	可以从字符串中删除后续的空格 - str1 - 字符串变量	<function name="string.trimright" > str1 </function>

参见: [/Easy XML Library/ 4\\_系统预定义功能 / 2\\_文件处理 \(字符串处理等\) / filehandler\\_2.xml](#)

### 9.9.5 数据位处理

<p>设置给定变量的各个位。 可以置位或复位。</p>	<p>语法: &lt;function name="ncfunc.bitset" refvar="address" value="0/1" &gt; bit0, bit1, ... bit9 &lt;/function&gt;</p> <p>属性: refvar - 指定要写入的位组合所在变量的名称 value - 位的取值为 0 或 1 (复位或置位)</p> <p>参数 位号从 0 开始给定 (bit0) 。每次调用最多可以修改 10 个位 (bit0-bit9) 。</p>
---------------------------------	--

示例:

置位	<function name="ncfunc.bitset" refvar="nck/Channel/Parameter/R[1]" value="1" > 0, 2, 3, 7 </function>
复位	<function name="ncfunc.bitset" refvar="nck/Channel/Parameter/R[1]" value="0" > 1, 4 </function>

### 9.9.6 控件处理

函数	说明	语法
Delete control	删除指定的控件	<function name="control.delete"> control name </function>
Get focus	获取当前正处在输入状态的控件的名称	<function name="control.getfocus" return="focus_name" />
Set focus	可将指定的控件设置到输入状态。	<function name="control.setfocus"> control name </function>
Get curssel	返回列表框中光标所在的下标。	<function name="control.getcursel" return="var">control name </function>
Set curssel	可在列表框中将光标放置在相应的行中	<function name="control.setcursel" > control name, index</function>
Get item	将列表框中所选择行的内容 (item) 复制到指定的变量中。	<function name="control.getitem" return="var"> control name, index </function>
Get item data	将列表框中用户自定义的元素值 (item_data) 复制到指定的变量中。	<function name="control.getitemdata" return="var"> control name, index </function>
Display format	提供由系统确定的浮点数格式规则。	<function name="ncfunc.displayresolution" return="string" />
Form color	以字符串形式提供对话框的文本颜色或背景颜色。Var1: - BACKGROUND - 请求背景颜色值 - TEXT - 请求文本 (前景) 颜色值	<function name="control.formcolor" return="variable">_T"Var1"</function>

local time	<p>将本地时间复制到包含 7 个元素的数组中。</p> <p>数组元素中保存以下内容：</p> <ul style="list-style-type: none"> <li>- 下标 0 - 年份</li> <li>- 下标 1 - 月份</li> <li>- 下标 2 - 星期</li> <li>- 下标 3 - 日</li> <li>- 下标 4 - 小时</li> <li>- 下标 5 - 分钟</li> <li>- 下标 6 - 秒</li> </ul>	<pre>&lt;let name="time_array" dim="7" /&gt; &lt;function name ="control.localtime"&gt;_T"time_array" &lt;/function&gt;</pre>
------------	--	---

示例：

<pre>&lt;function name="control.delete"&gt;_T"my_edit" &lt;/function&gt;</pre>
<pre>&lt;function name="control.setfocus" "&gt;_T"listbox1" &lt;/function&gt;</pre>
<pre>&lt;function name="control.getcurssel" return="index"&gt;_T"listbox1"&lt;/function&gt;</pre>
<pre>&lt;function name="control.setcurssel" "&gt;_T"listbox1",2 &lt;/function&gt;</pre>
<pre>&lt;function name="control.getitem" return="item" "&gt;_T"listbox1",2&lt;/function&gt;</pre>

示例：ncfunc.displayresolution

```
<function name="ncfunc.displayresolution" return="dislay_res" />
<control name="edit9" xpos="148" ypos="290" refvar=
"nck/Channel/GeometricAxis/progDistToGo[1]" hotlink="true" fieldtype=
"readonly" format="$$dislay_res" color_bk="#00FFFF"/>
dislay_res=%9.3f
```

## 9.9.7 像素处理

### 9.9.7.1 BITMAP.DIM

获取图片 (bmp/png/gif/jpg) 文件长和宽的像素信息，需要用到结构体定义。

语法：

```
<function name="bitmap.dim" >name, variable type </function>
```

示例：

```
<typedef name="size" type="struct" >
  <element name="width" type="int">0 </element>
  <element name="height" type="int">0 </element>
</typedef>

<let name="bmp_size" type="size" />
<softkey position="9" >
  <caption>bitmap </caption>
  <function name="bitmap.dim" >_T"test.bmp", bmp_size </function>
</softkey>
<init>
<caption>file: func_stdfunc1.xml </caption>
  <control name="edit1" xpos="8" ypos="50" refvar="bmp_size.width" hotlink="true"/>
  <control name="edit2" xpos="8" ypos="70" refvar="bmp_size.height" hotlink="true"/>
</init>
```

bitmap.width	658
bitmap.height	494

### 9.9.7.2 Screen Resolution

将系统的绝对屏幕分辨率重新复制到 SIZE (width, height) 结构类型的变量中。

语法:

```
<function name="hmi.screen_resolution" >variable type </function>
```

示例:

```
<let name="scn_resolution" type="size" />
<softkey position="10" >
  <caption>screen%resolution </caption>
  <function name="hmi.screen_resolution" >scn_resolution </function>
</softkey>
<control name="edit3" xpos="8" ypos="110" refvar="scn_resolution.width" hotlink="true" />
<control name="edit4" xpos="8" ypos="130" refvar="scn_resolution.height" hotlink="true" />
```

scn_resolution.width	1366
scn_resolution.height	768

### 9.9.7.3 Get HMI Resolution

该函数将 SINUMERIK Operate (不包含边屏等区域) 所用的屏幕分辨率重新复制到 SIZE 结构类型的变量中。

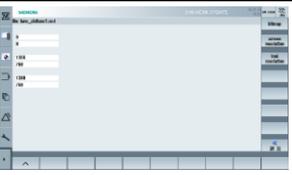
语法:

```
<function name="hmi.get_hmi_resolution" >variable type </function>
```

示例:

```
<let name="hmi_resolution" type="size" />
<control name="edit5" xpos="8" ypos="170" refvar="hmi_resolution.width" hotlink="true" />
<control name="edit6" xpos="8" ypos="190" refvar="hmi_resolution.height" hotlink="true" />
<softkey position="11" >
  <caption>hmi%resolution </caption>
  <function name="hmi.get_hmi_resolution" >hmi_resolution </function>
</softkey>
```

分辨率 1366x768 的屏幕上测试功能, 结果如下 (开启边屏功能):

	展开	关闭
		
Screen Resolution	1366 768	1366 768
Get HMI Resolution	1300 768	1024 768

### 9.9.7.4 Get Caption Height

该函数提供标题栏高度, 单位: 像素。

语法:

```
<function name="hmi.get_caption_height" return="<return var>" />
```

### 9.9.8 图像处理

- Image Box Set

用于 Imagebox 控件可见区域的控制。

语法:

`<function name="control.imageboxset">control name, command, command parameter</function>`

Command	说明	command parameter <类型>
x_offs	将图像区移至指定的 X 位置	P1<像素>
y_offs	将图像区移至指定的 Y 位置。	P1<像素>
xy_offs	将图像区移至指定的 X/Y 位置	P1,P2<像素>
followCursor	图像区遵循设置的光标位置。	
zoomplus	图像以 0.12 的系数放大。	
zoomminus	图像以 0.12 的系数缩小。	
autozoom	图像根据显示区域自动缩放。	
Update	控件重新标记	
SetBkColor	设置指定的背景颜色	P1<"#RRGGBB">
SetCursorRect	指定为矩形的图像区移至可见区域。	
SetAnimationState	(只针对 Gif 动画) 启动/停止动画	P1<"start" 或 "stop">

- Image Box Get

该函数获取 Imagebox 控件属性。

语法:

`<function name="control.imageboxget">control name, command, command parameter</function>`

Command	说明	command parameter <类型>
GetViewSize	提供显示区尺寸	P1<像素>

示例:

```
<function name="control.imageboxset"> _T"topview", _T"zoomplus"</function>
<let name="view_size" type="size" />
<function name="control.imageboxget"> _T"topview", _T"GetViewSize", view_size</function>
```

(size 类型用 typedef 标签定义)

### 9.9.9 数学运算

函数	说明	语法
ABS	取绝对值	<code>&lt;function name="abs" return="var"&gt; value &lt;/function&gt;</code>
SDEG	换算为度	<code>&lt;function name="sdeg" return="var"&gt; value &lt;/function&gt;</code>
SRAD	换算为弧度	<code>&lt;function name="srad" return="var"&gt; value &lt;/function&gt;</code>
SQRT	平方根	<code>&lt;function name="sqrt" return="var"&gt; value &lt;/function&gt;</code>
ROUND	设定小数点位数	<code>&lt;function name="round" return="var"&gt; value, nDecimalPlaces &lt;/function&gt;</code>

FLOOR	向下取整	<function name="floor" return="var"> value </function>
CEIL	向上取整	<function name="ceil" return="var"> value </function>
LOG	对数计算	<function name="log" return="var"> value </function>
LOG10	10 进制对数	<function name="log10" return="var"> value </function>
POW	幂运算 (a 的 b 次幂)	<function name="pow" return="var"> a, b </function>
MIN	取最小数	<function name="min" return="var"> value1, value2 </function>
MAX	取最大数	<function name="max" return="var"> value1, value2 </function>
RANDOM	取随机数	<function name="random" return="var" </function>

参见: /Easy XML Library/ 4\_系统预定义功能 / 3\_数学运算 / arithmetic.xml

### 9.9.10PI 服务

PI 服务	Ncfunc PI-Service
<function name="ncfunc.pi_service" return="error"> pi name, var1, var2, var3, var4, var5 </function>	
通道 PI 服务	ncfunc chan PI-Service
<let name="chan" >1</let> ;通道号, 从 1 开始 <function name="ncfunc.chan_pi_service" return="error"> _T"pi name", chan, var1,var2...</function>	

函数	说明	语法
刀具表的操作		
_N_CREATO	创建刀具 - var1 - 刀具号	<function name="ncfunc.pi_service" return="error ">_T" _N_CREATO", var1</function>  <function name="ncfunc.chan_pi_service" return="error"> _T" _N_CREATO", chan, var1</function>
s_N_DELETEO	删除刀具 - var1 - 刀具号	<function name="ncfunc.pi_service" return="error ">_T" _N_DELETEO", var1</function>  <function name="ncfunc.chan_pi_service" return="error"> _T" _N_DELETEO", chan, var1</function>
_N_CREACE	创建刀沿 - var1 - 刀具号 - var2 - 刀沿号	<function name="ncfunc.pi_service" return="error ">_T" _N_CREACE", var1, var2</function>  <function name="ncfunc.chan_pi_service" return="error"> _T" _N_CREACE", chan, var1, var2</function>
_N_DELECE	删除刀沿 - var1 - 刀具号 - var2 - 刀沿号	<function name="ncfunc.pi_service" return="error ">_T" _N_DELECE", var1, var2</function>  <function name="ncfunc.chan_pi_service" return="error"> _T" _N_DELECE", chan, var1, var2</function>
零点偏移的激活		
_N_SETUFR	将当前的用户框架激活	<function name="ncfunc.pi_service" return="error"> _T" _N_SETUFR"</function>  <function name="ncfunc.chan_pi_service" return="error"> _T" _N_SETUFR", chan</function>

_N_SETUDT	将当前的用户数据激活 - var1 - 待激活的用户数据区域 = 1 - 刀具补偿数据 = 2 - 有效基本框架 = 3 - 有效可设置框架	<pre>&lt;function name="ncfunc.pi_service" return="error" "&gt;_T"_N_SETUDT", var1&lt;/function&gt;</pre> <pre>&lt;function name="ncfunc.chan_pi_service" return="error"&gt; _T"_N_SETUDT ", chan, var1&lt;/function&gt;</pre>
<b>程序段查找</b>		
_N_FINDBL	激活查找 - var1 - 查找模式 =2 - 带轮廓计算的查找 =4 - 查找到程序段末尾 =1 - 不带计算的查找	<pre>&lt;function name="ncfunc.pi_service" return="error" "&gt;_T"_N_FINDBL", var1&lt;/function&gt;</pre> <pre>&lt;function name="ncfunc.chan_pi_service" return="error"&gt; _T"_N_FINDBL ", chan, var1&lt;/function&gt;</pre>
_N_FINDAB	取消程序段查找	<pre>&lt;function name="ncfunc.pi_service" return="error" "&gt;_T"_N_FINDAB"&lt;/function&gt;</pre> <pre>&lt;function name="ncfunc.chan_pi_service" return="error"&gt; _T"_N_FINDAB", chan&lt;/function&gt;</pre>

参见: [/Easy XML Library/ 4\\_系统预定义功能 / 1\\_NC 功能\\_读写变量等 / 2\\_PI\\_Service 功能 / nc\\_pi\\_service.xml](#)

## 10 附录

### 10.1 参考手册

如下手册可在西门子官方网站下载，链接如下：

SINUMERIK 840D sl Easy XML

<https://support.industry.siemens.com/cs/document/109769181/sinumerik-840d-sl-easy-xml?dti=0&lc=en-VWV>

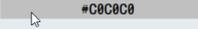
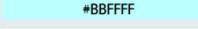
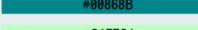
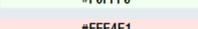
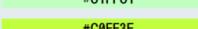
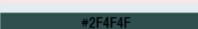
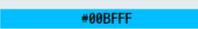
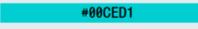
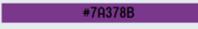
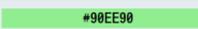
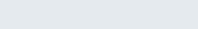
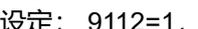
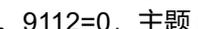
SINUMERIK 828D Easy XML

<https://support.industry.siemens.com/cs/document/109761933/sinumerik-828d-easy-xml?dti=0&lc=en-VWV>

### 10.2 颜色代码

EasyXML 界面开发中使用的颜色属性使用 RGB 颜色代码搭配 (R – 红色, G – 绿色, B – 蓝色)。颜色数据通过字符 “#” 和六个 16 进制系统数字共同组成 (#RRGGBB), 而每种颜色通过两位 16 进制数表示。总共可支持  $255 \times 255 \times 255 = 16,581,375$  种颜色搭配。

常用颜色：

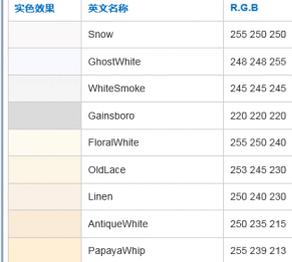
实色效果	颜色	#RRGGBB		
	黑色	#000000		
	白色	#FFFFFF		
	红色	#FF0000		
	青蓝色	#00FFFF		
	蓝色	#0000FF		
	绿色	#00FF00		
	黄色	#FFFF00		
				
				
				
				
				

界面背景色：主题 0 (#D7E1EB)，主题 1 (#E5ECF0)

显示机床数据 9112 操作界面设计 (主题) 设定；9112=1，主题 1。9112=0，主题 0。

参见：[/Easy XML Library/7\\_颜色展示/ xmlldial.xml](#)

开发者可以结合很多选色板工具，实现颜色的快速查找。

实色效果	英文名称	R.G.B	16色	实色效果	英文名称	R.G.B	16色
	Snow	255 250 250	#FFFAFA		PaleTurquoise1	187 255 255	#BBFFFF
	GhostWhite	248 248 255	#F8F8FF		PaleTurquoise2	174 238 238	#AEEEEE
	WhiteSmoke	245 245 245	#F5F5F5		PaleTurquoise3	150 205 205	#96CDCD
	Gainsboro	220 220 220	#DCDCDC		PaleTurquoise4	102 139 139	#668BB8
	FloralWhite	255 250 240	#FFFAF0		CadetBlue1	152 245 255	#98F5FF
	OldLace	253 245 230	#FDF5E6		CadetBlue2	142 229 238	#8EE5EE
	Linen	250 240 230	#FAF0E6		CadetBlue3	122 197 205	#7AC5CD
	AntiqueWhite	250 235 215	#FAEBD7		CadetBlue4	83 134 139	#S3868B
	PapayaWhip	255 239 213	#FFEFD5		Turquoise1	0 245 255	#00F5FF

<http://tool.oschina.net/commons?type=3>

### 10.3 运算符

数学运算符	说明	数学运算符	说明
+	加法	*	乘法
-	减法	/	除法

逻辑运算符	说明	逻辑运算符	说明
	位方式或 (OR)	==	相等
	逻辑或 (OR)	!=	不等
&, &amp;	位方式与 (AND)	>, &gt;	大于
&&, &amp;&amp;	逻辑与 (AND)	<, &lt;	小于
!	逻辑非 (NOT)	>=, &gt;=	大于等于
BNOT	位方式 NOT	<=, &lt;=	小于等于
<<	向左移位		
>>	向右移位		

说明	系统函数
正弦 sin	<function name="sin" return="<double val>"> double </function>
余弦 cos	<function name="cos" return="<double val>"> double </function>
正切 tan	<function name="tan" return="<double val>"> double </function>
反正弦 arcsin	<function name="arcsin" return="<double val>"> double </function>
反余弦 arcos	<function name="arccos" return="<double val>"> double </function>
反正切 arctan	<function name="arctan" return="<double val>"> double </function>
取绝对值	<function name="abs" return="var"> value </function>
换算为度	<function name="sdeg" return="var"> value </function>
换算为弧度	<function name="srad" return="var"> value </function>
平方根	<function name="sqrt" return="var"> value </function>
设定小数点位数	<function name="round" return="var"> value, nDecimal </function>
向下取整	<function name="floor" return="var"> value </function>
向上取整	<function name="ceil" return="var"> value </function>
对数计算	<function name="log" return="var"> value </function>
10 进制对数	<function name="log10" return="var"> value </function>
幂运算 (a 的 b 次幂)	<function name="pow" return="var"> a, b </function>
取最小数	<function name="min" return="var"> value1, value2 </function>
取最大数	<function name="max" return="var"> value1, value2 </function>
取随机数	<function name="random" return="var" </function>

## 10.4 预定义按键

预定义按键	中文	英文	默认位置	句法:
SOFTKEY_OK			Position="16"	<softkey_ok> ... </softkey_ok>
SOFTKEY_CANCEL			Position="15"	<softkey_accept> ... </softkey_accept>
SOFTKEY_BACK			Position="16"	<softkey_back> ... </softkey_back>
SOFTKEY_ACCEPT			Position="16"	<softkey_cancel> ... </softkey_cancel>

位置固定，位置冲突时，以编程顺序后者优先原则显示

## 10.5 系统界面变量

名称	说明	适用标签
\$actionresult	解析器报告结果，是否要通过解析器处理事件	<KEY_EVENT>
\$focus_name	当前光标所在控件的名称	<FOCUS_IN>
\$focus_item_data	当前光标所在控件（元素）的属性 item_data 数值	<INDEX_CHANGED> <EDIT_CHANGED>
\$gestureinfo	手势操作变量，并执行标签 <gesture_event>	<GESTURE_EVENT>
\$return	Function_body 函数体返回值	<FUNCTION_BODY>
\$message_par1	<SEND_MESSAGE>标签传递的参数 1	<MESSAGE>
\$message_par2	<SEND_MESSAGE>标签传递的参数 2	
\$xmlAttribute	包含已找到的标签属性的列表	XML_PARSER startElementHandler
\$xmlValue	包含已找到的标签值的列表	
\$xmlCharacters	包含数据流	XML_PARSER charactersHandler
\$xmlCharactersStart	包含起始下标	
\$xmlCharactersLength	包含数据流中保存的字符的数量	
\$mouse_event.type	鼠标事件参数的传输结构 (详见 窗体事件-mouse_event 章节.)	<MOUSE_EVENT>
\$mouse_event.x		
\$mouse_event.y		
\$mouse_event.id		
\$mouse_event.buttons		

## 10.6 语种缩写表

简体中文	chs	繁体中文	cht	马来西亚语	msl	瑞典语	sve
德语	deu	丹麦语	dan	荷兰语	nld	斯洛伐克语	sky
英语	eng	芬兰语	fin	波兰语	plk	斯洛文尼亚语	slv
法语	fra	印度尼西亚语	ind	葡萄牙语 (巴西)	ptb	泰语	tha
意大利语	ita	日语	jpn	罗马尼亚语	rom	捷克语	csy
西班牙语	esp	韩语	kor	俄语	rus	土耳其语	trk
越南语	vit	匈牙利语	hun	越南语			

## 11 作者及版本

DI MC MTS APC Chengfei

如需给与好评或建议，可扫描反馈，感谢使用。



版本信息

版本	日期	修改内容
V1.0	2020.03.04	第一版