16.7 通过自定义 EasyXLanguage 脚本创建用户对话框

16.7.1 功能范围

概述

使用功能“创建用户对话框”可以保证开放性，它能够让用户设计出客户专用和应用专用的 HMI 界面。

SINUMERIK 808D 提供基于 XML 的脚本语言用于创建用户对话框。

该脚本语言可以在 HMI 上的操作区“用户自定义”中显示机床专用菜单和对话框窗口。

使用

已定义的 XML 指令可以实现下列特性：

1. 显示对话框并提供：
   - 软键
   - 变量
   - 文本和帮助文本
   - 图形和帮助画面
2. 通过以下方法调用对话框：
   - 按下相应软键
3. 动态重组对话框
   - 修改、删除软键
   - 定义并设计变量栏
   - 显示、更换、删除显示文本（和语言相关或无关）
   - 显示、更换、删除图形

4. 在进行以下操作时触发动作：
   - 显示对话框
   - 输入数值（变量）
   - 按下软键
   - 关闭对话框
5. 对话框间的数据交换
6. 变量
   - 读取（NC 变量、PLC 变量、用户变量）
   - 写入（NC 变量、PLC 变量、用户变量）
   - 和数学、比较或者逻辑运算符相连
7. 执行下列功能：
   - 子程序
   - 文件功能
   - PI 服务
8. 根据用户组考虑保护等级

关于脚本语言的有效单元（标签）的描述可参见章节“XML 标签 (页 252)”。

说明

下列章节关于 XML （Extensible Markup Language 可扩展标记语言）的描述没有完整性要求。更多的信息可查阅相应的专业文献。

16.7.2 配置基础

配置文件

新操作界面的说明存储在配置文件中。 这些文件自动编译并显示屏幕上的结果。 在工具盒的“...\examples\easyXL” 文件夹中可找到配置文件（EasyXLanguage 脚本）。

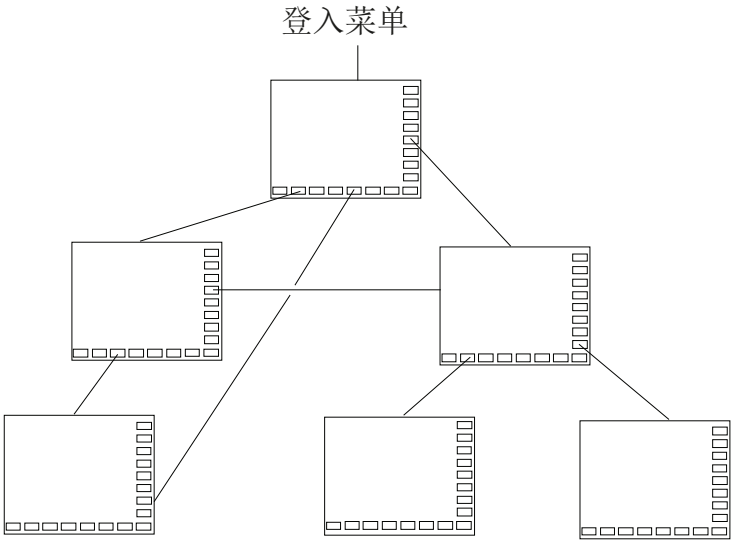
可以使用 XML 编辑器或其他的文本编辑器来创建配置文件。

说明
不区分大小写。

菜单树的原理

多个相连的对话框构成了一个菜单树。 如果能从一个对话框切换入另一个对话框，则表示这两个对话框间存在联系。 通过此对话框内重新定义的水平或者垂直软键可以返回上级对话框或者进入任意一个对话框。

可以在登入菜单后面通过配置好的登入软键生成更多的菜单树：



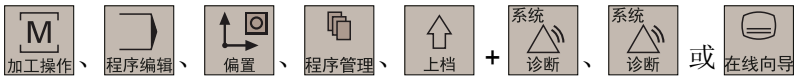
登入菜单

在文件“xmldial.xml”中使用名称“main”来定义登入菜单。 登入菜单是操作流程自身的输出点。

使用主菜单可以与加载自身对话框或其他软键条连接在一起。 而使用这些软键条又可以执行其他动作。

返回到标准应用程序

可以通过按下 PPU 上的下列键或组合键退出新建的用户界面并返回到标准应用程序。



16.7.3 配置文件（EasyXLanguage）

载入配置文件

必须将创建的配置文件从 USB 存储器复制到“系统”操作区 > “系统数据” > “808D 数据” > “HMI 数据” 菜单 > “EasyXLanguage 脚本”文件夹。 参见下屏：

用于配置的文件

为了进行用户对话框的配置需要使用数控系统中“EasyXLanguage 脚本”文件夹内的下列文件：

文件类型	文件名称	含义
脚本文件	“xmldial.xml”	该脚本文件通过 XML 标签控制已配置软键菜单的映像以及 HMI 上操作区“用户自定义”中的对话框屏幕。
文本文件	"almc.txt"	该文本文件包含有用于单个语言菜单和对话框屏幕的文本。

文件类型	文件名称	含义
位图	"*.bmp"（例如，"text.bmp"） "*.png"（例如，"text.png"）	带位图的文档。 控制器支持 BMP 格式和 PNG 格式。
XML 文件，在控制文件“xmldial.xml”中插入了 XML 标签“INCLUDE”。	例如 “machine_settings.xml	该文件同样包含用来在 HMI 上显示对话框窗口和参数的编程指令。

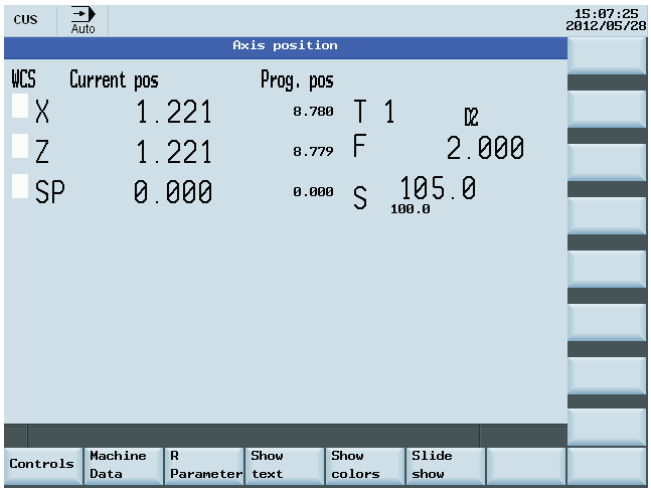
说明

一旦“xmldial.xml”脚本文件存在于“EasyXLanguage 脚本”文件夹中，便可在“用户自定义”操作区中起始该用户对话框。

初始复制过程结束后，必须通过“系统”操作区 > “调试” > “NC” > “正常引导启动”来复位数控系统。

HMI 上用户对话框示例

在调用操作区“自定义”时显示配置好的软键菜单。 用户可以通过配置好的各个对话框窗口进行操作。



说明

如果需要同时使用配置好的且已编程的对话框，则必须通过脚本语言调用已编程对话框。为此所需的功能在章节“预定义函数 (页 288)”中有所说明。

16.7.4 配置文件的结构

概述

配置文件由以下单元组成：

- 对带有登入软键的登入菜单“main”的说明
- 对话框定义
- 变量定义
- 块说明
- 定义软键条

16.7.5 语言相关性

使用和语言相关的文本：

- 软键标记
- 标题
- 辅助文本
- 其它任意文本

与语言相关的文本保存在文本文件（almc.txt）中。

16.7.6 XML 标签

16.7.6.1 通用结构

用于对话框配置的脚本文件结构与指令

所有的对话框配置都保存在 **DialogGui** 标签中。

```
<DialogGui>
…
</DialogGui>
```

示例：

```
<?xml version="1.0" encoding="utf-8"?>
<DialogGui>
…
```

```
<FORM name ="Hello_World">
<INIT>
<CAPTION>Hello World</CAPTION>
</INIT>
...
</FORM>

</DialogGui>
```

指令

语言为完成有条件指令与完成程序循环提供有下列指令：

- For loop 循环
- While loop 循环
- Do with loop 循环
- 有条件执行
- Switch 指令和 Case 指令
- 对话框窗口中的操作单元
- 软键说明
- 定义变量

指令的详细说明参见章节“指令/标签说明 (页 254)”。

16.7.6.2 指令/标签说明

为了创建对话框和菜单以及处理程序顺序需要定义下列 **XML 标签** ：

说明

应用当前使用的表达式来替换引号"<...>"中的属性值。

实例：如下对

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

进行编程：

<DATA_LIST action="read/write/append" id="my datalist">

标签名称	含义
BREAK	有条件中断一个循环。
CONTROL_RESET	该标签可以重新启动一个或多个控制器组件。 **句法：** `<CONTROL_RESET resetnc="TRUE" />` **属性：** • **RESETNC = "TRUE"** 数控系统组件重新启动

DATA	该标签启用待直接写入的 NC、PLC 和 GUD 数据。
	地址构成可以查阅章节“组件地址分配 (页 267)”。
	属性：
	• **name**
	变量地址
	标签值：
	允许使用所有包括文字和数字的表达方式作为标签值。 如要对一个局部变量中的值直接进行写操作，则要使用替代运算符 **$** ，将局部变量名跟在其后。
	句法：
	`<DATA name="<variable name>"> value </DATA>`
	示例：
	`<DATA name = "plc/mb170"> 1 </DATA>`
	…
	`<LET name = "tempVar"> 7 </LET>`
	`<!– 局部变量内容"tempVar"写入位存储字节 170 →`
	`<DATA name = "plc/mb170">$tempVar</DATA>`

标签名称	含义

DATA_LIST	该标签可以保存或者恢复所列示的机床数据。
	可以创建最多 20 个临时数据列表。
	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经存在的列表中
	write – 将保存的值复制到相应的机床数据中
	• **id**
	名称用于识别临时存储器
	句法：
	`<DATA_LIST action="<read/write/append>"  id="<list name>">`
	`NC/PLC 地址编译`
	`</DATA_LIST>`
	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>" />`

ELSE	满足条件时的指令（**IF（如果）**、**THEN（则）**、**ELSE（否则）**）

标签名称	含义
FORM	该标签包含用户对话框的说明。 相应的标签在章节创建菜单与对话框窗口中有更多说明。 **句法：** `<FORM name="<dialog name>" color="#ff0000">` **属性：** • **color** 对话框窗口的背景颜色（颜色代码参见章节 颜色代码 (页 266)） – 了解缺省值 • **name** 形状的名称 • **xpos** 对话框左上角 X 位置（可选） • **ypos** 左上角 Y 位置（可选） • **width** X 方向上的长度（以像素为单位）（可选） • **height** Y 方向上的长度（以像素为单位）（可选）
HMI_RESET	该标签触发 HMI 重启。 解释在该操作之后中断。

标签名称	含义
IF	有条件指令（**IF, THEN, ELSE**） THEN 和 ELSE 标签包含在 IF 标签中。 在 CONDITION 标签中执行的条件处于 IF 标签之后。 需要通过其他的指令处理来确定运算结果。 如果功能结果为真，则执行 THEN 分支而跳过 ELSE 分支。 如果功能结果为假，则按 ELSE 分支分步程序进行处理。 **句法：** <IF> <CONDITION> Condition != 7 </CONDITION> <THEN> 该情况下的指令： 满足条件 </THEN> <ELSE> 该情况下的指令： 不满足条件 </ELSE> </IF> **示例：** <IF> <CONDITION> "plc/mb170" != 7 </CONDITION> <THEN> <OP> "plc/mb170" = 7 </OP> … </THEN> <ELSE> <OP> "plc/mb170" = 2 </OP> … </ELSE> </IF>
INCLUDE	指令包含有 XML 说明。 （参见该表中的 DYNAMIC_INCLUDE ） **属性：** • **src** 包含路径名称。 **句法：** <?INCLUDE src="<Path name>" ?>

标签名称	含义
LET	指令以给出的名称创建一个局部变量。 **域：** 使用属性 **dim**（维度），可创建一维或二维域。 域索引用于定址单个域元素。 对于二维域，先指定行索引，而后指定列索引。 • 一维域： 索引 0 到 4 • 二维域： 索引行 0 到 3 以及索引列 0 到 5 **属性：** • **name** 变量名称 • **type** 变量类型可以为整数型（INT）、双精度型（DOUBLE）、浮点数型（FLOAT）或者字符串型（STRING）。 如果未给出类型指令，则系统会创建一个整数型变量。 `<LET name = "VAR1" type = "INT" />` • **permanent** 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。 • **dim** 必须指定下列数目的域元素。 对于二维域，先指定第一维度，再指定第二维度，两者用逗号分开。 通过域索引访问域元素，在变量名称之后于方括号中指定域索引。 name[index] 或 name[row,column] – 一维域： dim="<元素数目>" – 二维域： dim="<行数目>,<列数目>" 非初始化域元素预设为“0”。

标签名称	含义

LET 续	**示例：**
	一维域：
	`<let name="array" dim="10"></let>`
	二维域：
	`<let name="list_string" dim="10,3" type="string"></let>`
	预设值：
	可以对变量进行初始化赋值。
	`<LET name = "VAR1" type = "INT"> 10 </LET>`
	如果将 NC 或 PLC 变量的值保存在一个局部变量中，则赋值运算会自动将格式与所读入变量的格式进行匹配。
	• 字符串变量的预设值：
	只要将格式化的文本作为值进行传送，就可以为一个字符串变量赋值多行文本。 如果一行要使用换行 line feed <LF> 结尾，则可以在行的末尾添加字符 "\\n" 。
	`<LET name = "text" type = "string"> F4000 G94\\n`
	`G1 X20\\n`
	`Z50\\n`
	`M2\\n`
	`</LET>>`
	域（Arrays）：
	`<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"},`
	`  {"text 20","text 21"}`
	`</let>`
	赋值：
	可以使用赋值运算“=”将机床数据或子程序中的值赋值给一个变量。
	变量到上一级 XML 块结束时一直有效。
	全局可用的变量要直接在标签 DialogGUi 后创建。
	在对话框中要注意：
	• 信息处理时会打开相应的标签。
	• 签并在信息处理完后关闭标签。
	• 因此标签中的所有变量都会被删除。

标签名称	含义
MSG	操作组件显示标签中所给出的信息。 如果使用报警编号，则对话框会显示为该编号所保存的文本。 **示例：** `<MSG text ="my message" />`

MSGBOX	指令会打开一个信息框，其给分支的返回值可供使用。
	句法：
	`<MSGBOX text="<Message>" caption="<caption>" retvalue="<variable name>" type="<button type>" />`
	属性：
	• **text**
	文本
	• **caption**
	标题
	• **retvalue**
	其中复制有返回值的变量名称：
	1 – OK
	0 – CANCEL
	• **type**
	应答可能：
	"BTN_OK"
	"BTN_CANCEL"
	"BTN_OKCANCEL"
	如果对属性**"text"**或**"caption"**使用报警编号，则消息框会显示为该编号所保存的文本。
	示例：
	`<MSGBOX text="测试消息" caption="信息" retvalue="result" type="BTN_OK" />`

标签名称	含义
OP	该标签执行所列出的运算。 对于 NC/PLC 上的存取和驱动数据要在 **引用字符** 中设置完整的变量名称。 PLC: "PLC/MB170" NC: "NC/Channel/…" **示例：** <LET name = "tmpVar" type="INT"> </LET> <OP> tmpVar = "plc/mb170" </OP> <OP> tmpVar = tmpVar *2 </OP> <OP> "plc/mb170" = tmpVar </OP> **字符串处理：** 运算指令可以对字符串进行处理并将结果字符串赋值给等式中所给出的字符串变量。 为了标记文本表达式需要在文本前放置标志 _T 。 此外还能对变量值进行格式化。 格式规定以标志 _F 开始，后面跟有格式指令。 接着给定变量地址。 **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>
PASSWORD	标签打开用于输入密码的对话框。 输入之后可以使用所给出参考变量中的字符串。 **句法：** <PASSWORD refVar ="<variable name>" /> **属性：** • **refVar** 参考变量的名称 **示例：** <PASSWORD refvar="plc/mw107" />
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。

标签名称	含义

PRINT	标签会在对话行中输出文本或者将文本复制到给定的变量中。
	如果文本包含有格式化标志，则会将变量值插入到相应的位置上。
	句法：
	`<PRINT name="变量名称" text="text %格式化"> Variable, … </PRINT>`
	`<PRINT text="text %格式化"> Variable, … </PRINT>`
	属性：
	• **name**
	其中应当保存文本的变量名称（可选）
	• **text**
	文本
	格式化：
	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

标签名称	含义

PRINT 续	**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符一致并使用逗号相互分隔开来。
	示例：
	在信息行中输出文本
	`<PRINT text="信息文本" />`
	使用变量格式输出文本
	`<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
	`<LET name="trun_dir"></LET>`
	`<LET name="str" type="string" ></LET>`
	`<print name="str" text="M%d ">trun_dir</print>`

STOP	解释在该位置中断。

SWITCH	**SWITCH** 指令表示一种多重选择。 表达式运用一次并与常数数目进行比较。
	如果表达式与常数一致，则处理 **CASE** 指令中的相关指令。
	如果没有常数与表达式一致，则处理 **DEFAULT** 指令。
	句法：
	`<SWITCH>`
	`<CONDITION> 值 </CONDITION>`
	`<CASE value="常数 1">`
	指令
	…
	`</CASE>`
	`<CASE value="常数 2">`
	指令
	…
	`</CASE>`
	`<DEFAULT>`
	指令
	…
	`</DEFAULT>`
	`</SWITCH>`

THEN	满足条件时的操作（**IF, THEN, ELSE**）

标签名称	含义
FOR	FOR loop 循环 for （初始化；测试；增量）指令 **句法：** <FOR> <INIT>…</INIT> <CONDITION>…</CONDITION> <INCREMENT>…</INCREMENT> 指令 … </FOR> 按下列方式执行 For 循环： 1. 运用表达式**初始化**（INIT）。 2. 运用表达式**测试**（CONDITION）作为布尔型表达式。 如果值为假（false），则结束 For 循环。 3. 执行后续的指令。 4. 运用表达式**增量**（INCREMENT）.. 5. 继续使用 2。 所有在 INIT、ONDITION 和 INCREMENT 分支中使用的变量都要在 FOR loop 循环外部创建。 **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>
WAITING	标签在数控系统复位后等待组件重新启动。 **属性：** • **WAITINGFORNC** = "TRUE" - 系统等待数控系统重启 **句法：** <WAITING WAITINGFORNC = "TRUE"/>

标签名称	含义
WHILE	While 循环 WHILE (Test) 指令 **句法：** <WHILE> <CONDITION>…</CONDITION> 指令 … </WHILE> While loop 循环用于按顺序多次执行指令，只要满足条件。 在处理指令序列前对条件进行检查。 **示例：** <WHILE> <CONDITION> "plc/ib9" == 0 </CONDITION> <DATA name = "PLC/qb11"> 15 </DATA> </WHILE>
DO_WHILE	Do-While 循环 DO 指令 WHILE (测试) **句法：** <DO_WHILE> 指令 … <CONDITION>…</CONDITION> </DO_WHILE> Do while loop 环由一个指令程序段和一个条件构成。 首先执行指令程序段中的代码，然后运用条件。 如果条件为真（true），则重新执行代码部分。 一直重复，直至条件为假（false）。 **示例：** <DO_WHILE> <DATA name = "PLC/qb11"> 15 </DATA> <CONDITIION> "plc/ib9" == 0 </CONDITION> </DO_WHILE>

16.7.6.3 颜色代码

颜色属性使用 HTML 语言的颜色代码搭配。

颜色数据通过句法由字符“#”（菱形）和六个十六进制系统数字共同组成，而每种颜色通过两个数字代表。

R – 红色

G – 绿色

B – 蓝色

#RRGGBB

示例：

color= "#ff0011"

16.7.6.4 专用 XML 句法

如果要在通用 XML 编辑器中正确显示在 XML 句法中有特殊含义的字符，则必须加以说明。

关系到下列字符：

字符	XML 中的记数法
<	&lt;
>	&gt;
&	&amp;
"	&quot;
'	&apos;

16.7.6.5 运算符

运算指令可以处理下列运算符：

运算符	含义
=	赋值
==	等于
<, <	小于
>, >	大于
<=, <=	小于等于
>=, >=	大于等于
|	位模式或（OR）连接
||	逻辑或（OR）连接
&, &	位模式与（AND）连接
&&, &&	逻辑与（AND）连接
+	加法
-	减法
*	乘法
/	除法
!	否
!=	不等

运算指令由左向右处理。括号可能对于表达式也有意义，可以确定部分表达式的处理优先级。

16.7.7 组件地址分配

为了给 NC 变量、PLC 组件或驱动数据分配地址，地址名称由所需要的数据构成。一个地址由部分路径 **组件名称** 和 **变量地址**组成。使用斜线作为分隔符。

16.7.7.1 PLC 地址分配

PLC 地址分配以路径部分 **plc** 开始。

表格 16- 4　允许使用下列地址：

DBx.DB(f)	数据块
I(f)x	输入端
Q(f)x	Output
M(f)x	标志
V(f)x	变量

DBx.DBXx.b	数据块
Ix.b	输入端
Qx.b	Output
Mx.b	标志
Vx.b	变量

表格 16- 5　数据格式 **f**：

B	字节
W	字
D	双字

在位地址分配时取消数据格式标识。

地址 **x**：

有效的 S7 200 地址名称

位地址分配：

b – 位号

示例：

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

16.7.7.2 NC 变量地址分配

NC 变量地址分配以路径部分 **nck** 开始。

在该部分之后跟有数据地址，其结构可以查阅 SINUMERIK 808D 参数手册。

示例：

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

16.7.7.3 机床数据和设定数据的地址分配

可以使用 $ 符号标记设定数据，后面跟有数据名称。

机床数据：

$Mx_<名称[**索引, AX<轴编号>**]>

设定数据：

$Sx_<名称[索引, AX<轴编号>]>

x：

N – 通用的机床或设定数据

C – 通道专用的机床或设定数据

A – 轴专用的机床或设定数据

索引：

参数在该域中给出数据索引。

AX<轴编号>：

在轴专用数据时对所需要的轴（**<轴编号>**）进行详细说明。

可以选择利用“替代符号” **$<变量名称>** 从一个局部变量中读取轴索引。

例如 AX$ 局部变量

示例：

```
<DATA name ="$MN_AXCONF_MACHAX_NAME_TAB[0] ">X1</DATA>
```

轴的直接地址分配：

```
<DATA name ="$MA_CTRLOUT_MODULE_NR[0, AX1] ">1</DATA>
…
…
```

轴的间接地址分配：

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

16.7.7.4 用户数据的地址分配

用户数据地址分配以路径部分 **gud** 开始，后面跟有 GUD 名称。

为一个域分配地址，在名称后的方括号中应指定所需域索引。

示例：

```
<DATA name ="gud/syg_rm[0]"
<OP>"gud/syg_rm[0]" 0 10 </op>
```

16.7.8 创建用户菜单

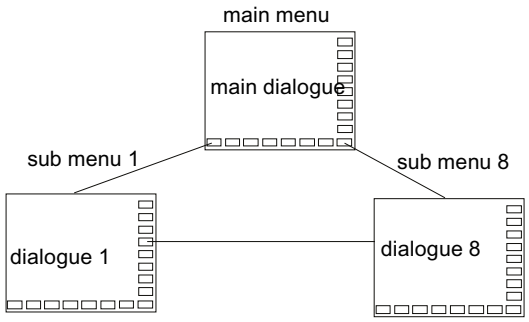
16.7.8.1 创建软键菜单和对话框窗口

要合成用户菜单必须在 XML 说明中有名为“main”的主菜单标签。 在激活操作区“用户自定义”后由系统调用该标签。在标签之中可以定义其他的菜单分支以及激活一个对话框。

```
<menu name= "MAIN">
<OPEN_FORM name= "main dialogue">
<softkey POSITION="1">
<caption>sub menu 1</caption>
<navigation>sub menu 1</navigation>
</softkey>
<softkey POSITION="8">
<caption>sub menu 8</caption>
<navigation>sub menu 8</navigation>
</softkey>
</menu>

 <menu name= "sub menu 1">
 <OPEN_FORM name= "dialogue 1">
 </menu>

<menu name= "sub menu 8">
<OPEN_FORM name= "dialogue 8“>
</menu>
```



标签名称	含义
FORM	该标签包含用户对话框的说明。 **属性：** • **color** 对话框窗口的背景颜色（颜色代码参见章节“颜色代码”） • **name** 窗体的名称 • **xpos** 对话框左上角 X 位置（可选） • **ypos** 左上角 Y 位置（可选） • **width** X 方向上的长度（以像素为单位）（可选） • **height** Y 方向上的长度（以像素为单位）（可选） **对话框信息：** • **INIT** • **PAINT** • **TIMER** • **CLOSE** • **FOCUS_IN**

标签名称	含义

FORM 续	**句法：**
	`<FORM name = "<dialog name>" color = "#ff0000">`
	示例：
	`<FORM name = "R-Parameter">`
	` <INIT>`
	` <DATA_ACCESS type = "true" />`
	` <CAPTION>R - Parameter</CAPTION>`
	` <CONTROL name = "edit1" xpos = "322" ypos = "34" refvar = "nck/Channel/Parameter/R[1]" />`
	` <CONTROL name = "edit2" xpos = "322" ypos = "54" refvar = "nck/Channel/Parameter/R[2]" />`
	` <CONTROL name = "edit3" xpos = "322" ypos = "74"`
	` </INIT>`
	
	`  <PAINT>`
	` <TEXT xpos = "23" ypos = "34">R - Parameter 1</TEXT>`
	` <TEXT xpos = "23" ypos = "54">R - Parameter 2</TEXT>`
	` <TEXT xpos = "23" ypos = "74">R - Parameter 3</TEXT>`
	
	` </PAINT>`
	
	`</FORM>`

INIT	**对话框信息** 在对话框创建之后立即处理该标签。在此创建所有的输入单元以及对话框窗口的“热连接”。

FOCUS_IN	**对话框信息**
	当系统将焦点设置到控制单元时，调用该标签。 为了识别控制单元，系统将控制单元的名称复制到变量 **$focus_name** 中并将 item_data 的属性值复制到变量 **$focus_item_data** 中。系统自动创建变量。
	例如可以使用该信息，来输出与焦点位置有关的图像。
	示例：
	`<focus_in>`
	`<PRINT text="focus on filed:%s, %d">$focus_name, $focus_item_data </PRINT>`
	`</focus_in>`

PAINT	**对话框信息** 在显示对话框时处理该标签。在此要给定需要在对话框中显示的所有文本与图像。 此外，在系统识别出部分对话框需要重新显示时处理该标签。 例如，通过关闭高层窗口来触发该操作。

标签名称	含义
TIMER	**对话框信息** 周期性处理该标签。 每一窗体均具备一个定时器，该定时器大约每隔 100 ms 触发一次定时器标签的处理。

CAPTION	该标签包含有对话框的标题。
	在 INIT 标签内使用该标签。
	句法：
	`<CAPTION>Titel</CAPTION>`
	示例：
	`<CAPTION>my_first_dialogue</CAPTION>`

CLOSE	**对话框信息** 在关闭对话框之前处理该标签。

CLOSE_FORM	该标签关闭活动对话框。
	该指令仅当其涉及用于程序编辑器区域中的循环对话框时有用。 一般而言，对话框受自动管理，无需特意关闭。
	句法：
	`<CLOSE_FORM/>`
	示例：
	`<softkey_ok>`
	`<caption>OK</caption>`
	`<CLOSE_FORM />`
	`<navigation>main_menu</navigation>`
	`</softkey_ok>`

标签名称	含义
CONTROL	该标签用来创建控制单元。 **句法：** `<CONTROL name = "<control name>" xpos = "<X position>" ypos = "<Y position>" refvar = "<NC variable>" hotlink = "true" format = "<format>" />` **属性：** • **name** 域的名称 名称会同时显示一个局部变量，且无法在窗体中多次使用。 • **xpos** 左上角的 X 位置 • **ypos** 左上角的 Y 位置 • **fieldtype** 域类型 如未给定类型，则域被设置为编辑域。 – **edit** 并可对数据进行修改 – **readonly** 无法对数据进行修改 **combobox** 域显示相应的名称来代替数值。 如果选择了域类型“**combobox**”，则要为域另外补充显示表达式。 为此要使用标签 **<ITEM>** 。 复选框将当前选定文本的索引储存在控制单元所属的变量中（参见属性 **refvar**）。 – **progressbar** 在值域 0 到 100 内显示进度条。 值域与待显示数据的最小值和最大值相匹配。

标签名称	含义
CONTROL 续	• **fieldtype** – **listbox** 该域类型创建一个空的列表框控制项。 可使用标签 **<ITEM>** 将列表框元素插入列表框中。 ITEM 属性 **value** 允许赋予该元素唯一值。 该操作，例如，可用于识别元素。 参数 **width** 和 **height** 指定列表框的宽度和高度。 创建控制项之后，可使用函数 **AddItem**、**InsertItem** 或 **LoadItem** 插入其他列表框元素。 – **graphicbox** 该域类型创建一个 2 维虚线图形控制项。 可使用标签 **<ITEM>** 将图形元素插入控制项。 参数 **width** 和 **height** 指定框的宽度和高度。 **注意：** 该控制项不关联“剪切”功能。 因而，其他元素可覆盖该控制项。 创建控制项之后，可使用函数 **AddItem** 或 **InsertItem** 插入其他元素。 对该控制项不进行参数 **itemdata** 评估。 – **itemlist** 该域类型创建一个静态控制项，该控制项显示相应的名称来代替数值。 可使用 **<ITEM>** 标签赋予该域一个名称。 • **item_data** 可以使用用户专用的整数值为该属性赋值。该值被提供给 FOCUS_IN 信息，用来识别焦点域。 • **refvar** 可连接到该域的参考变量的名称（可选）。 • **hotlink = "TRUE"**当参考变量数值改变时，该域自动更新（可选）。 • **format** 属性定义了指定变量的显示格式。 格式化数据，参见 print-Tag （可选）。

标签名称	含义
CONTROL 续	**属性：** • **time** 给定数据的刷新率（可选）。 可以使用下列数据： – **super fast** 刷新时间 < 100 ms – **fast** 刷新时间约 100 ms – **normal** 刷新时间约 200 ms – **slow** 刷新时间约 500 ms • **font** 该属性确定所使用的字体大小。 – 0: 8*8 – 1: 16*8 – 2: 24*16（仅用于数字） – 3: 8*8 两倍字符高度 – 4: 16*8 两倍字符高度 – 5: 24*16 两倍字符高度（仅用于数字） • **color_bk** 此属性可设置控制项的背景色。 • **color_fg** 此属性可设置控制项的前景色。 （“颜色代码”参见章节“颜色代码 (页 266)”） • **display_format** 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 向列表框、图形框或复选框分配表达内容（例如要显示的文本或图形元素）： **句法：** <ITEM>表达式</ITEM>

标签名称	含义
CONTROL 续	**示例：** `<CONTROL name = "button1" xpos = "10" ypos = "10" fieldtype = "combobox "> <ITEM>text1</ITEM> <ITEM>text2</ITEM> <ITEM>text3</ITEM> <ITEM>text4</ITEM> </CONTROL>` 如果要为表达式分配任意一个整数值，则要为标签添加属性 **value = "数值"**。 控制变量含有已赋值的对象值，用来代替连续的编号。 **示例：** `<CONTROL name = "button1" xpos = "10" ypos = "10" fieldtype = "combobox "> <ITEM value = "10">text1</ITEM> <ITEM value = "20">text2</ITEM> <ITEM value = "12">text3</ITEM> <ITEM value = "1">text4</ITEM>` **</CONTROL>** **渐进条形图示例** `<CONTROL name = "progress1" xpos = "10" ypos = "10" width = "100" fieldtype = "progressbar" hotlink = "true" refvar = "nck/Channel/GeometricAxis/actProgPos[1]"> <PROPERTY min = "0" /> <PROPERTY max = "1000" /> </CONTROL>` **列表框示例：** `<let name="item_string" type="string"></let> <let name="item_data" ></let>  <CONTROL name="listbox1" xpos = "360" ypos="150" width="200" height="200" fieldtype="listbox" />` • 添加元素： 可使用函数 **additem** 或 **loaditem** 添加元素。 • 清除内容： 可使用函数 **empty** 清空内容。 `<op> item_string = _T"text1\\n" </op> <function name="control.additem">_T"listbox1", item_string, item_data </function> <op> item_string = _T"text2\\n" </op> <function name="control.additem">_T"listbox1", item_string, item_data </function>`

标签名称	含义
CONTROL 续	图形框示例： `<CONTROL name= "graphic" xpos = "8" ypos="23" width="300" height="352" fieldtype="graphicbox" />` • 添加元素： 可使用函数 **additem** 或 **loaditem** 添加元素。 可以使用以下 2 维元素： – 直线 - l(inc) – 圆弧 - c(ircle) – 点 - p(oint) 元素结构： <Elementtyp>；坐标 • 行： l; xs; ys; xe, ye l - 直线符号 Xs - X 起始位置 Ys - Y 起始位置 Xe - X 结束位置 Ye - Y 结束位置 • 圆弧： C, xs, ys, xe, ye, cc_x, cc_y, r C - 圆弧符号 Xs - X 起始位置 Ys - Y 起始位置 Xe - X 结束位置 Ye - Y 结束位置 Cc_x – 圆心的 X 坐标 Cc_y – 圆心的 Y 坐标 • 半径： R • 点： P, x, y P - 点的符号 X - X 坐标 Y - Y 坐标 • 清除图形： 可使用函数 **empty** 清空内容。

标签名称	含义
CONTROL 续	示例： <let name="item_string" type="string"></let> <let name="s_z" type="double">100</let> <let name="s_x" type="double">50</let> <let name="itemdata"></let> … … <control name= "gbox" xpos = "6" ypos="24" width="328" height="356" fieldtype="graphicbox" /> … … <print name ="item_string" text="p; %f; %f">s_z, s_x</print> <function name="control.additem">_T"gbox", item_string, itemdata</function> … 示例 itemlist： <CONTROL name = "itemlist1" xpos = "10" ypos = "10" fieldtype = " itemlist"> <ITEM value = "10">text1</ITEM> <ITEM value = "20">text2</ITEM> <ITEM value = "12">text3</ITEM> <ITEM value = "1">text4</ITEM> </CONTROL>
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。 在属性中给定的名称会加上前缀 XmlUserDlg_ 并转发到帮助系统中。 这样可以从在线帮助主题中提取出帮助文件的关联结构。 激活帮助系统的过程： 1. 按下“信息”键。 2. 对话画面会显示""my_dlg_help""。 3. 分析程序会将表达式转换为""XmlUserDlg_my_dlg_help" "。 4. 帮助系统激活。 5. 传递查找关键字""XmlUserDlg_my_dlg_help""。 句法： <HELP_CONTEXT name="<context name>" /> 示例： … <INIT> … <CAPTION>my dialogue</CAPTION> <HELP_CONTEXT name="my_dlg_help" /> … </INIT>

标签名称	含义
DATA_ACCESS	该标签在储存用户输入数据时用来控制对话框窗口的特性。 在 INIT 初始化标签中确定其特性。 如未使用标签，则总是使用输入数据中间存储器。 例外： 属性 **hotlink** 设置为 **true** 。 **属性：** • **type** = "TRUE" - 不对输入值进行缓存。对话框窗口直接将输入值复制到参考变量中。 • **type** = "FALSE" – 值只通过标签 **UPDATA_DATA** type = "FALSE" 复制到参考变量中。 **示例：** `<DATA_ACCESS type = "true" />`

MENU	该标签定义一个菜单，其中包含有软键说明与待打开的对话框。
	属性：
	• **name**
	菜单名称
	句法：
	`<MENU name = "<menu name>">`
	…
	`<open_form …>`
	…
	`<SOFTKEY …>`
	`</SOFTKEY>`
	`</MENU>`

NAVIGATION	该标签用来确定所要调用的菜单。只能在一个软键程序块内使用该标签。
	句法：
	`<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>

标签名称	含义

OPEN_FORM	该标签可以打开名称下指令的对话框窗口。
	属性：
	• **name**
	对话框窗体的名称
	句法：
	`<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>`
	
	`<form name="main_form">`
	`<init>`
	`</init>`
	
	`<paint>`
	`</paint>`
	
	`</form>`

标签名称	含义

PROPERTY	使用该标签可以确定操作单元的附加特性。
	属性：
	• **max** = “<最大值>”
	• **min** = “<最小值>”
	• **default** = “<预设值>”：
	• **factor** = “换算系数”
	• **color_bk** = “<背景色代码>”
	• **color_fg** = “<前景色代码>”
	• **font** = “<字体编号>”
	• **password** = “<true>” - 所输入的字符显示为“*”
	• **multiline** = “<true>” - 在 EDIT 控制项中允许多行输入
	• **disable** = “"<true/false>” - 在 EDIT 控制项禁止/允许输入
	示例：
	`<CONTROL name = "progress1" xpos = "10" ypos = "10" width = "100" fieldtype = "progressbar" hotlink = "true" refvar = "nck/Channel/GeometricAxis/actProgPos[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>`

SOFTKEY	该标签定义了软键的特性与反应。
	属性：
	• **position**
	软键的编号。1-8 水平软键， 9-16 垂直软键
	可以在软键程序块中确定下列其他动作。
	• **caption**
	• **navigation**
	• **update_controls**
	• **function**
	句法：
	`<softkey position = "<1>">`
	…
	…
	`</softkey>`

标签名称	含义
TEXT	该标签用来在指定的位置上显示文本。 如果使用报警编号，则对话框会显示为该编号所保存的文本。 **句法：** `<TEXT xpos = "<X 坐标>" ypos = "<Y 坐标>"> Text </TEXT>` **属性：** • **xpos** 左上角的 X 位置 • **ypos** 左上角的 Y 位置 • **color** 文本颜色（颜色代码） **值：** 待显示的文本

IMG	该标签用来在指定的位置上显示图像。支持 BMP 和 PNG 图像格式。
	句法：
	`<IMG xpos = "<X 坐标>" ypos = "<Y 坐标>" name = "<名称>" />`
	属性：
	• **xpos**
	左上角的 X 位置
	• **ypos**
	左上角的 Y 位置
	• **name**
	完整路径名称
	• **transparent**
	位图的透明色（参见章节“颜色代码”）
	可选：
	希望修改图片的原始大小时，可以使用属性 **width** 和 **height** 来确定尺寸。
	• **width**
	宽度（像素）
	• **height**
	高度（像素）
	示例：
	`<IMG xpos = "20" ypos = "40" name = "f:/appl/test.bmp" />`
	`<IMG xpos = "5" ypos = "23" name = "f:/appl/test.bmp" height = "355" width = "550"/>`

标签名称	含义
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。 **句法：** `<BOX xpos = "<X 坐标>" ypos = "<Y 坐标>" width = "<X 长度>" height = "<Y 长度>" color = "<颜色代码>" />` **属性：** • **xpos** 左上角的 X 位置 • **ypos** 左上角的 Y 位置 • **width** X 方向上的长度（以像素为单位） • **height** Y 方向上的长度（以像素为单位） • **color** 颜色代码（颜色代码参见章节“颜色代码”）

FUNCTION	**函数调用**
	该标签可以执行属性“**name**”中给出的。
	属性：
	• **name =** “函数体的名称”
	• **return =** “用于储存函数结果的变量名称”
	值：
	需要将其传输到函数体的变量列表。变量相互之间以逗号分隔。最多可以传输 10 个参数。
	此外，还可以将常数或文本表达式指定为调用参数。为了标记文本表达式需要在文本前放置标志 _T 。
	句法：
	`<FUNCTION name = "<function name>" />`
	待调用的函数等待一个返回值
	`<FUNCTION name = "<function name>" return = "<Variablenname>" />`
	参数传递
	`<FUNCTION name = "<function name>"> var1, var2, var3 </FUNCTION>`
	`<FUNCTION name = "<function name>"> _T"Text", 1.0, 1 </FUNCTION>`
	示例：
	参见“FUNCTION_BODY”

标签名称	含义
FUNCTION_BODY	函数体 该标签包含有一个子函数的函数体。 D 要在对话框 DialogGui 标签中对函数体进行编程。 属性： • name = “函数体的名称” • parameter = “参数列表”（可选） 属性列出了所需传输参数。参数相互之间以逗号分隔。 调用函数体时将函数调用中给定参数的值复制到所列出的传输参数中。 • return = "true" 如果属性为 true，则创建局部变量 $return。将函数的返回值复制到其中，在放弃该函数时可以将其继续传输给待调用的函数。 句法： 无参数的函数体 <FUNCTION_BODY name = "<function name>"> … … … </ FUNCTION_BODY> 带参数的函数体 <FUNCTION_BODY name = "<function_name>" parameter = "<p1, p2, p3>"> … <LET name = "tmp"></LET> <OP> tmp = p1 </OP> … </FUNCTION_BODY> 带返回值的函数体 <FUNCTION_BODY name = "<function_name>" parameter = "<p1, p2, p3>" return = "true"> … <LET name = "tmp"></LET> <OP> tmp = p1 </OP> … <OP> $return = tmp </OP> </FUNCTION_BODY>

标签名称	含义
FUNCTION_BODY 续	示例： <function_body name = "test" parameter = "c1,c2,c3" return = "true"> <LET name = "tmp">0</LET> <OP> tmp = c1+c2+c3 </OP> <OP> $return = tmp </OP> </function_body> … … … <LET name = "my_var"> 4 </LET> <function name = "test" return = " my_var "> 2, 3,4</function> <print text = "result = %d"> my_var </print> … …
REQUEST	借助该标签可以在周期性读取服务（hotlink 热连接）中使用一个变量。 这样可以缩短对未与控制项相连的变量的存取时间。 如果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。 该标签仅在 INIT 初始化指令中进行处理。 属性： • name 地址识别符 句法： <REQUEST name = "<NC-Variable>" />
UPDATE_CONTROLS	该标签在操作单元和参考变量之间进行补偿。 属性： • type 该属性可以确定数据比较的方向。 = TRUE – 从参考变量读取数据并复制到操作单元中。 = FALSE – 从操作单元读取数据并复制到参考变量中。 句法： <UPDATE_CONTROLS type = "<方向>"/> 示例： <SOFTKEY_OK> < UPDATE_CONTROLS type="false"/> </SOFTKEY_OK>

16.7.8.2 替代符号

系统可以确定有关运行时间的控制特性（属性值）。为了能够使用该函数，需要在一个局部变量中提供所需的特性并使用前置的 **$ 符号** 将变量名称作为属性值传输给标签。

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 **$$$**。

示例：

```
<let name="my_ypos">100</let>
<let name="field_name" type="string"></let>

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

<op>my_ypos = my_ypos +20 </op>

<control name = "edit2" xpos = "322" ypos = "$my_ypos"
refvar="nck/Channel/Parameter/R[2]" />

<print name =" field_name" text="edit%d">3</print>
<op>my_ypos = my_ypos +20 </op>

<control name = "$field_name" xpos = "322" ypos = "$my_ypos"
refvar="nck/Channel/Parameter/R3]" />

<caption>$$$field_name</caption>
```

16.7.9 预定义函数

脚本语言提供有不同的字符串处理和数学标准函数。

后面列出的函数名称是预留的并且不能重载。

函数名称	含义

String.cmp	从字典编辑角度对两个字符串进行相互比较。
	当字符串相同时该函数提供返回值零，当第一个字符串小于第二个时返回值小于零或者当第二个字符串小于第一个时返回值大于零。
	参数：
	str1 - 字符串
	str2 - 比较字符串
	句法：
	`<function name="string.cmp" return ="<int var>" > str1, str2 </function>`
	示例：
	`<let name="rval">0</let>`
	`<let name="str1" type="string">A brown bear hunts a brown dog.</let>`
	`<let name="str2" type="string">A brown bear hunts a brown dog.</let>`
	
	`<function name="string.cmp" return="rval"> str1, str2 </function>`
	结果：
	rval= 0

函数名称	含义

String.icmp	在不区分大/小写的情况下从字典编辑角度对两个字符串进行相互比较。
	当字符串相同时该函数提供返回值零，当第一个字符串小于第二个时返回值小于零或者当第二个字符串小于第一个时返回值大于零。
	参数：
	str1 - 字符串
	str2 - 比较字符串
	句法：
	`<function name="string.icmp" return ="<int var>" > str1, str2 </function>`
	示例：
	`<let name="rval">0</let>`
	
	`<let name="str1" type="string">A brown bear hunts a brown dog.</let>`
	`<let name="str2" type="string">A brown Bear hunts a brown Dog.</let>`
	`<function name="string. icmp" return="rval"> str1, str2 </function>`
	结果：
	rval= 0

String left	该函数提取字符串 1 中最初的 n 数量字符并将其复制到返回变量中。
	参数：
	str1 - 字符串
	nCount - 字符数量
	句法：
	`<function name="string.left" return="<result string>"> str1,  nCount </function>`
	示例：
	`<let name="str1" type="string">A brown bear hunts a brown dog.</let>`
	`<let name="str2" type="string"></let>`
	
	`<function name="string. left" return="str2"> str1, 12 </function>`
	结果：
	str2="A brown bear"

函数名称	含义
String.right	该函数提取字符串 1 中最后的 n 数量字符并将其复制到返回变量中。 **参数：** **str1** - 字符串 **nCount** - 字符数量 **句法：** <function name="string.right" return="<result string>"> str1, nCount </function> **示例：** <let name="str1" type="string">A brown bear hunts a brown dog.</let> <let name="str2" type="string"></let <function name="string. right " return="str2"> str1, 10 </function> **结果：** str2="brown dog."
String middle	该函数从索引 iFirst 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。 **参数：** **str1** - 字符串 **iFirst** - 开始索引 **nCount** - 字符数量 **句法：** <function name="string.middle" return="<result string>"> str1, iFirst, nCount </function> **示例：** <let name="str1" type="string">A brown bear hunts a brown dog.</let> <let name="str2" type="string"></let <function name="string. middle " return="str2"> str1, 2, 5 </function> **结果：** str2="brown"

函数名称	含义

String.length	该函数可以提供字符串的字符数量。
	参数：
	str1 - 字符串
	句法：
	`<function name="string.length" return="<int var>"> str1 </function>`
	示例：
	`<let name="length">0</let>`
	
	`<let name="str1" type="string">A brown bear hunts a brown dog.</let>`
	`<function name="string.length" return="length"> str1 </function>`
	结果：
	length = 31

Strings.replace	该函数可以使用新的字符串替代所有搜索到的部分字符串。
	参数：
	string - 字符串变量
	find string - 要替代的字符串
	new string - 新的字符串
	句法：
	`<function name="<string.replace>"> string, find string, new string </function>`
	示例：
	`<let name="str1" type="string">A brown bear hunts a brown dog. </let>`
	
	`<function name="string.replace" > str1, _T"a brown dog" , _T"a big salmon"</function>`
	结果：
	str1 = "A brown bear hunts a big salmon!"

函数名称	含义
String.remove	该函数可以删除所有搜索到的部分字符串。 **参数：** **string** - 字符串变量 **remove string** - 要删除的部分字符串 **句法：** <function name="string.remove"> string, remove string </function> **示例：** <let name="index">0</let> <let name="str1" type="string">A brown bear hunts a brown dog. </let> <function name="string.remove" > str1, _T"a brown dog" </function **结果：** str1 = "A brown bear hunts"
Strings.insert	该函数可以在指定的索引位置插入一个字符串。 **参数：** **string** - 字符串变量 **index** - 索引（以零为基准） **insert string** - 要插入的字符串 **句法：** <function name="string.insert"> string, index, insert string </function> **示例：** <let name="str1" type="string">A brown bear hunts. </let> <let name="str2" type="string">a brown dog</let> <function name="string.insert" > str1, 19, str2</function> **结果：** str1 = "A brown bear hunts a brown dog"

函数名称	含义

String delete	该函数可以从指定的启始位置开始删除确定数量的字符。
	参数：
	string - 字符串变量
	start index - 启始索引（以零为基准）
	nCount - 要删除字符的数量
	句法：
	`<function name="string.delete"> string, start index , nCount </function>`
	示例：
	`<let name="str1" type="string">A brown bear hunts. </let>`
	`<function name="string.delete" > str1, 2, 5</function>`
	结果：
	str1 = "A bear hunts"

String.find	该函数可以在所传输的字符串中搜索与部分字符串一致的第一处地方。
	如果找到了部分字符串，则该函数会提供到第一个字符的索引（从零开始），否则为 -1。
	参数：
	string - 字符串变量
	findstring- 要搜索的字符串
	句法：
	`<function name="string.find" return="<int val>"> str1, find string </function>`
	示例：
	`<let name="index">0</let>`
	`<let name="str1" type="string">A brown bear hunts a brown dog. </let>`
	`<function name="string.find" return="index"> str1, _T"brown" </function>`
	结果：
	Index = 2

函数名称	含义

String.reversefind	该函数可以在所传输的字符串中搜索与部分字符串一致的最后一处地方。
	如果找到了部分字符串，则该函数会提供到第一个字符的索引（从零开始），否则为 -1。
	参数：
	string - 字符串变量
	find string- 要搜索的字符串
	句法：
	`<function name="string.reversefind" return="<int val>"> str1, find string </function>`
	示例：
	`<let name="index">0</let>`
	`<let name="str1" type="string">A brown bear hunts a brown dog. </let>`
	`<function name="string.reversefind" return="index"> str1, _T"brown" </function>`
	结果：
	Index = 21

String.trimleft	该函数可以从字符串中删除打头的空格。
	参数：
	str1 - 字符串变量
	句法：
	`<function name="string.trimleft" > str1 </function>`
	示例：
	`<let name="str1" type="string">  test trim left</let>`
	`<function name="string.trimleft" > str1 </function>`
	结果：
	str1 = "test trim left"

函数名称	含义
String.trimright	该函数可以从字符串中删除后续的空格。 **参数：** **str1** - 字符串变量 **句法：** <function name="string.trimright" > str1 </function> **示例：** <let name="str1" type="string"> test trim right </let> <function name="string.trimright" > str1 </function> **结果：** str1 = "test trim right"
sin	该函数可以计算所传输值（以度为单位）的正弦值。 **参数：** **double** - 角度 **句法：** <function name="sin" return="<double val>"> double </function> **示例：** <let name= "sin_val" type="double"></let> <function name="sin" return="sin_val"> 20.0 </function>
cos	该函数可以计算所传输值（以度为单位）的余弦值。 **参数：** **double** - 角度 **句法：** <function name="cos" return="<double val>"> double </function> **示例：** <let name= "cos_val" type="double"></let> <function name="cos" return="cos_val"> 20.0 </function>

函数名称	含义
tan	该函数可以计算所传输值（以度为单位）的正切值。 **参数：** **double** - 角度 **句法：** <function name="tan" return="<double val>"> double </function> **示例：** <let name= "tan_val" type="double"></let> <function name="tan" return="tan_val"> 20.0 </function>
arcsin	该函数可以计算所传输值（以度为单位）的反正弦值。 **参数：** **double** - x 取值范围 -PI/2 至 +PI/2 **句法：** <function name="arcsin" return="<double val>"> double </function> **示例：** <let name= "arcsin_val" type="double"></let> <function name="arcsin" return=" arcsin_val"> 20.0 </function>
arccos	该函数可以计算所传输值（以度为单位）的反余弦值。 **参数：** **double** - x 取值范围 -PI/2 至 +PI/2 **句法：** <function name="arccos" return="<double val>"> double </function> **示例：** <let name= "arccos_val" type="double"></let> <function name="arccos" return=" arccos_val"> 20.0 </function>

函数名称	含义
arctan	该函数可以计算所传输值（以度为单位）的反正切值。 **参数：** **double** - 反正切 y/x **句法：** <function name="arctan" return="<double val>"> double </function> **示例：** <let name= "arctan_val" type="double"></let> <function name="arctan" return="arctan_val"> 20.0 </function>
dll.load	该函数可以在存储器中装载其他的用户 DLL。 **参数：** **dll_name** - DLL name **class_name** - “函数等级的名称” **句法：** <function name="dll.load"> dll_name, class_name </function> **示例：** <function name="dll.load"> _T"customer.dll", _T"customer" </function>
dll.function	该函数可以调用用户 DLL 中的函数。将参数 ID 后面列出的所有参数传输给所调用的函数。 **参数：** **class_name** - 函数等级的名称 **id** - 函数标识 **parameter** - 最多七个函数参数（字符串变量） **句法：** <function name="dll.function"> class_name, id, parameter1, parameter2</function> **示例** <function name="dll.function"> _T"customer", 290, _T"par1", _T"par2"</function>

函数名称	含义
文件处理	

doc.readfromfile	该函数可以将指定文件的内容装载到一个字符串变量中。
	属性：
	Return - 局部变量的名称
	参数：
	Progname - 文件名称
	句法：
	`<function name="doc.readfromfile" return="<string var>"> progname </function>`
	示例：
	`<let name = "my_var" type="string" ></let>`
	
	`<function name=" doc.readfromfile " return="my_var">_T"\spf\test.mpf" </function>`

doc.writetofile	该函数可以将一个字符串变量的内容写入指定文件中。
	参数：
	progname - 文件名称
	str1 - 字符串
	句法：
	`<function name="doc.writetofile" > progname, str1 </function>`
	示例：
	`<let name = "my_var" type="string" > file content </let>`
	
	`<function name="doc.writetofile">_T"\spf\test.mpf", my_var </function>`

doc.remove	该函数可以从目录中删除指定的文件。
	参数：
	progname - 文件名称
	句法：
	`<function name="doc.remove" > progname </function>`
	示例：
	`<function name="doc.remove">_T"\mpf\test.mpf" </function>`

函数名称	含义

doc.exist	如文件存在，则函数提供值 1。
	参数：
	progname - 文件名称
	句法：
	`<function name="doc.exist" return="<int_var>" > progname </function>`
	示例：
	`<let name ="exist">0</let>`
	
	`<function name="doc.exist" return="exist">_T"\mpf\test.mpf" </function>`

ncfunc.select	该函数可以选择将要处理的程序。程序必须保存在数控系统文件系统中。
	参数：
	progname - 文件名称
	句法：
	`<function name="ncfunc.select"> progname </function>`
	示例：
	`<function name="ncfunc.select"> _T"\mpf\test.mpf" </function>`