SIEMENS

SINUMERIK

SINUMERIK 840D sl / 828D
SINUMERIK Integrate
Run MyScreens（BE2）

编程手册

适用于：

数控软件 V4.7 SP2

10/2015
6FC5397-1DP40-5RA3

法律资讯

警告提示系统

为了您的人身安全以及避免财产损失，必须注意本手册中的提示。人身安全的提示用一个警告三角表示，仅与财产损失有关的提示不带警告三角。警告提示根据危险等级由高到低如下表示。

⚠ 危险
表示如果不采取相应的小心措施，**将会**导致死亡或者严重的人身伤害。

⚠ 警告
表示如果不采取相应的小心措施，**可能**导致死亡或者严重的人身伤害。

⚠ 小心
表示如果不采取相应的小心措施，可能导致轻微的人身伤害。

注意
表示如果不采取相应的小心措施，可能导致财产损失。

当出现多个危险等级的情况下，每次总是使用最高等级的警告提示。如果在某个警告提示中带有警告可能导致人身伤害的警告三角，则可能在该警告提示中另外还附带有可能导致财产损失的警告。

合格的专业人员

本文件所属的产品/系统只允许由符合各项工作要求的**合格人员**进行操作。其操作必须遵照各自附带的文件说明，特别是其中的安全及警告提示。 由于具备相关培训及经验，合格人员可以察觉本产品/系统的风险，并避免可能的危险。

按规定使用 Siemens 产品

请注意下列说明：

⚠ 警告
Siemens 产品只允许用于目录和相关技术文件中规定的使用情况。如果要使用其他公司的产品和组件，必须得到 Siemens 推荐和允许。正确的运输、储存、组装、装配、安装、调试、操作和维护是产品安全、正常运行的前提。必须保证允许的环境条件。必须注意相关文件中的提示。

商标

所有带有标记符号 ® 的都是西门子股份有限公司的注册商标。本印刷品中的其他符号可能是一些其他商标。若第三方出于自身目的使用这些商标，将侵害其所有者的权利。

责任免除

我们已对印刷品中所述内容与硬件和软件的一致性作过检查。然而不排除存在偏差的可能性，因此我们不保证印刷品中所述内容与硬件和软件完全一致。印刷品中的数据都按规定经过检测，必要的修正值包含在下一版本中。

Siemens AG
Division Digital Factory
Postfach 48 48
90026 NÜRNBERG
德国

文件订购号: 6FC5397-1DP40-5RA3
℗ 11/2015 本公司保留更改的权利

目录

前言 1

概述

通过“Run MyScreens”机床制造商或最终用户可以自行设计扩展功能的操作界面，或者设计操作界面的布局。

还可以在其他循环调用下生成自定义的操作界面。或直接在控制系统上创建对话框。

“Run MyScreens”通过编译器和包含操作界面说明的配置文件实现。

通过 ASCII 文件配置“Run MyScreens”：该配置文件包含了关于操作界面的说明。创建文件所需的句法参见下列章节。

基本功能

通过“Run MyScreens”机床制造商可自定义对话框。基本功能支持在操作菜单树中设计 5 个对话框，或者用于设计用户自定义的循环对话框。

软件选件

扩展对话框数量，需要下列软件选件：

- SINUMERIK 828D/840D sl, SINUMERIK Integrate Run MyScreens (6FC5800-0AP64-0YB0)
- SINUMERIK 840D sl, SINUMERIK Integrate Run MyScreens + Run MyHMI (6FC5800-0AP65-0YB0)
- SINUMERIK 840D sl, SINUMERIK Integrate Run MyHMI / 3GL (6FC5800-0AP60-0YB0)
- SINUMERIK 840D sl, SINUMERIK Integrate Run MyHMI / WinCC (6FC5800-0AP61-0YB0)

边界条件

必须符合以下边界条件：

- 只允许在一个操作区内切换对话框。
- 用户变量不允许与系统变量或者 PLC 变量有相同的名称（另请参见参数手册系统变量/PGAsl/）
- 由 PLC 激活的对话框构成一个单独的操作区（和测量循环屏幕类似）。

工具

- UTF8 编辑器（例如：操作界面的集成编辑器或记事本）
- 创建图形/屏幕所需的图形程序。

文件名和编码

说明

使用 NCU 中的 HMI Operate 时要注意 CF 卡上的所有文件名都是以小写字母保存的（如：com、png、txt）。这是 Linux 操作系统的要求。

在 PCU 上您可以任意以大小写字母保存文件名。为便于之后传送到 Linux，此处仍建议只使用小写字母。

说明

保存配置文件和语言文件时请注意，应将您所使用的编辑器中的编码设为 UTF 8。

保存路径

保存配置文件、帮助文件等时应遵循以下规定：

[系统西门子目录]	
Linux：	/card/siemens/sinumerik/hmi
Windows 7：	C:\ProgramData\Siemens\MotionControl\siemens
[系统 oem 目录]	
Linux：	/card/oem/sinumerik/hmi
Windows 7：	C:\ProgramData\Siemens\MotionControl\oem
[系统用户目录]	
Linux：	/card/user/sinumerik/hmi
Windows 7：	C:\ProgramData\Siemens\MotionControl\user
[系统插件目录]	
Linux：	/card/addon/sinumerik/hmi

	Windows 7：	C:\ProgramData\Siemens\MotionControl\addon
保存位置		
	“Run MyScreens” 配置：	*[系统 oem 目录]*/proj
	配置文件：	*[系统 oem 目录]*/cfg
	语言文件：	*[系统 oem 目录]*/lng
	图形文件：	*[系统 oem 目录]*/lco/ico*[分辨率]* 示例： *[系统 oem 目录]*/lco/ico640
	在线帮助：	*[系统 oem 目录]*/hlp/*[语言]* 示例： *[系统 oem 目录]*/hlp/eng

使用

可以实现以下功能：

显示对话框并提供：	● 软键 ● 变量 ● 文本和帮助文本 ● 图形和帮助图形
通过以下方法调用对话框：	● 按下（登入）软键 ● 选择 PLC/NC
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
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 ● 系统或用户变量的值变化
对话框间的数据交换	

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
执行下列功能：	• 子程序 • 文件功能 • PI 服务
根据用户组考虑保护等级	

基本操作

2

2.1 简介

根据以下示例可了解使用 Run MyScreen 在 SINUMERIK Operate 操作界面中添加自定义对话框的必要操作步骤。 此外还可了解如何创建自定义对话框、如何添加上下文有关的帮助图形和帮助调用框、如何定义软键以及如何在对话框之间进行导航。

2.2 示例

2.2.1 任务说明

本例中将创建以下对话框。

对话框 1

第一个对话框中显示可写 R 参数（0 和 1）和几何轴名称（输入栏）。两个 R 参数均链接有相应的帮助图形。几何轴名称未链接上下文有关的帮助信息。此外对话框中还包含诸如分割线（水平和垂直）、转换栏、集成了单位选择框的输入/输出栏和进度条（变色或不变色）等元素。

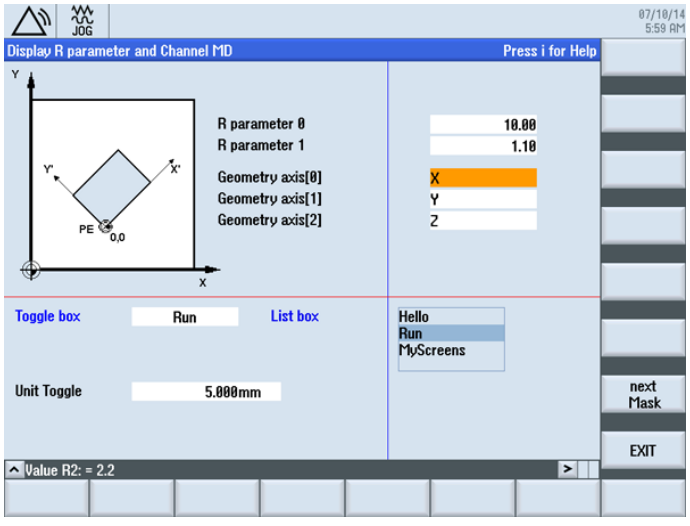


图 2-1 带辅助图的 R 参数

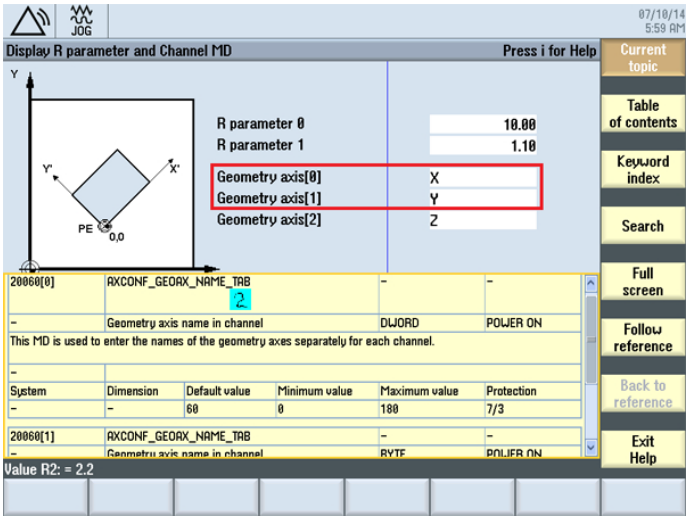


图 2-2　　带上下文相关帮助信息的几何轴

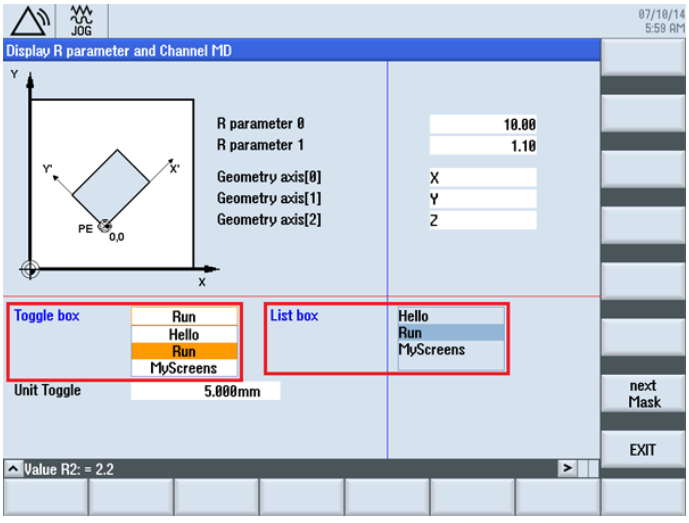


图 2-3　　转换栏：列表和选项框

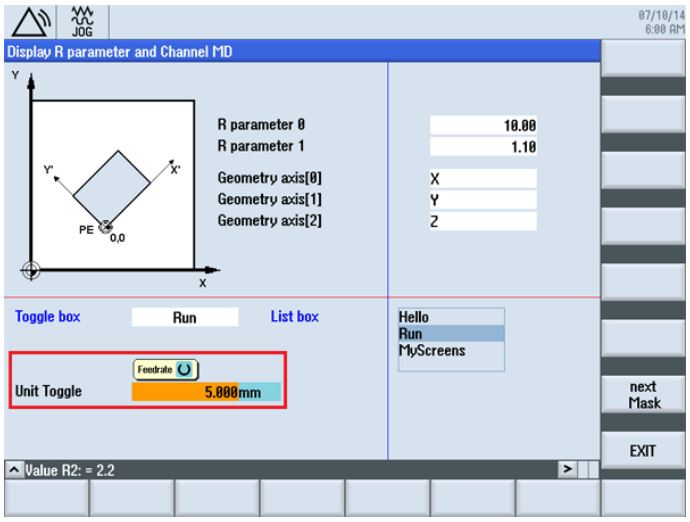


图 2-4　　集成了单位选择框的输入/输出栏

对话框 2

第二个对话框中显示 MCS 值和 WCS 值。此外对话框中还含有变色或不变色的进度显示。

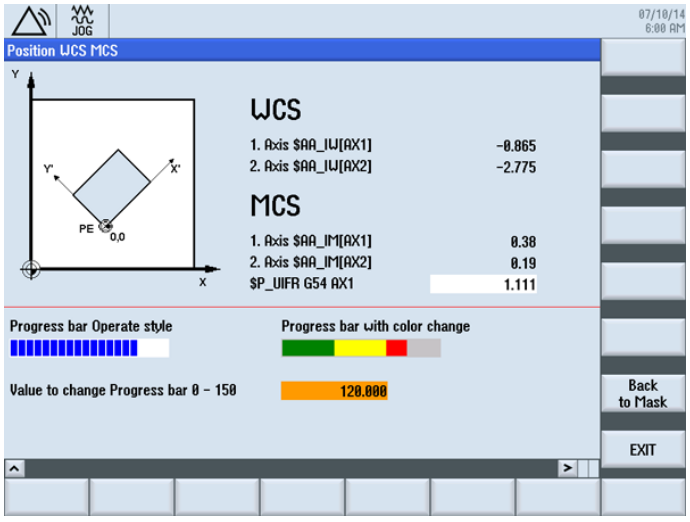


图 2-5　　变色（右）和不变色（左）的进度条

浏览

通过“诊断”操作区中的 “START” 登入软键调用第一个对话框。使用水平软键 SK7。

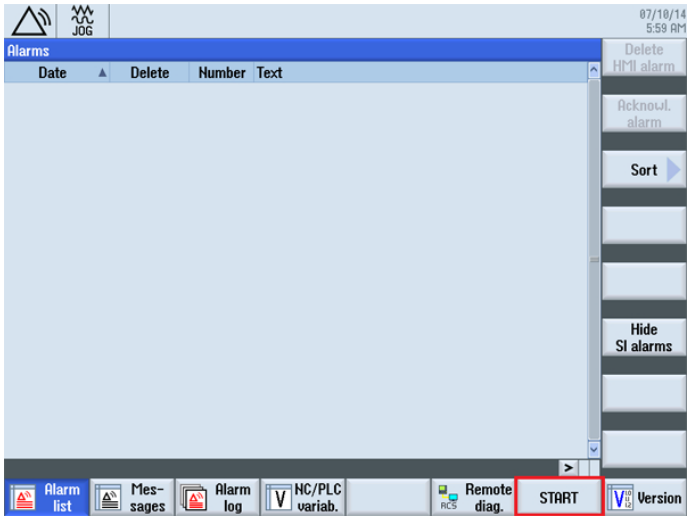


图 2-6　　AUTO 运行模式下“加工”操作区的登入软键„START“

在第一个对话框中使用软键 “next Mask” 可调用第二个对话框，使用软键 “EXIT” 返回操作区的基本画面（见上图）。

在第二个对话框中使用软键 “EXIT” 也同样能返回操作区的基本画面（见上图）。使用软键 “Back to Mask” 可以返回第一个对话框。

步骤

以下章节中说明了必要的操作步骤：

1. 创建配置文件（COM 文件）
2. 将配置文件保存到 OEM 目录下
3. 创建在线帮助
4. 将在线帮助保存到 OEM 目录下
5. 将 easyscreen.ini 复制到 OEM 目录中
6. 申明 easyscreen.ini 中的 COM 文件
7. 测试项目

2.2.2 创建配置文件

配置文件的内容

在 UTF8 编辑器中为两个对话框创建配置文件 diag.com 。

```
；登入软键开始标记
//S(START)
   ；纯文本登入软键
   HS7=("START")
   ；登入软键，带和语言相关的文本和 png
   ;HS7=([$80792,"\\sk_ok.png"])

   ；PRESS 方法
   PRESS(HS7)
      ；功能 LM 或者 LS
      LM("MASK1")
      ；带 com 文件说明的 LM
      LM("MASK1","TEST.COM")
   END_PRESS
；登入软键结束标记
//END

；带标题和图像的对话框 1 的定义
//M(MASK1/"Display R parameter and Channel MD"/"mz961_01.png")
   ；变量定义
   DEF VAR1 = (R2///,"R parameter 0"///"$R[0]"/200,50,150/400,50,100,)
   ；带帮助图形
   DEF VAR2 = (R2///,"R parameter 1"//"mz961_02.png"/"$R[1]"/200,70,150/400,70,100)
   ；带在线帮助
   DEF ACHS_NAM1 = (S///"Press i for Help","Geometry axis[0]"////
200,100,150/400,100,100//"sinumerik_md_1.html","20060[0]")
   ；带在线帮助
   DEF ACHS_NAM2 = (S///"Press i for Help","Geometry axis[1]"////
200,120,150/400,120,100//"sinumerik_md_1.html","20060[1]")
   DEF ACHS_NAM3 = (S///,"Geometry axis[2]"////200,140,150/400,140,100)
   ；转换栏和单位转换栏的定义
   DEF VAR_TGL = (S/* "Hello", "Run", "MyScreens"/"Run"/,"Toggle box"////10,230,100/120,230,100/,,6)

   DEF VAR_TGB = (S/* "Hello", "Run", "MyScreens"/"Run"/,"List box"/
dt4///250,230,100/370,230,100,60/,,"#0602ee")
   ; BC, FC, BC_ST, FC_ST, BC_GT, FC_GT, BC_UT, FC_UT, SC1, SC2
  DEF VarEdit = (R///,"Unit Toggle",,,"Feedrate"///"$R[11]"/5,300,100/120,300,100///"VarTgl"),
VarTgl = (S/*0="mm",1="inch"/0//WR2////220,300,40)
```

```
; 对话框中的软键定义
HS1=("")
HS2=("")
HS3=("")
HS4=("")
HS5=("")
HS6=("")
HS7=("")
HS8=("")
VS1=("")
VS2=("")
VS3=("")
VS4=("")
VS5=("")
VS6=("")
VS7=("next Mask")
VS8=("EXIT")

; LOAD 块定义
LOAD
   ; 通过 RNP 读取值
   ACHS_NAM1 = RNP("$MC_AXCONF_GEOAX_NAME_TAB[0]")
   ACHS_NAM2 = RNP("$MC_AXCONF_GEOAX_NAME_TAB[1]")
   ACHS_NAM3 = RNP("$MC_AXCONF_GEOAX_NAME_TAB[2]")
   ; 对话框行输出
   DLGL("Value R2: = " << RNP("$R[2]"))
   ; 通过 WNP 写入值
   WNP("$R[3]",VAR0)

  ; 垂直分割线
  V_SEPARATOR(360,1,6,1)
  ; 水平分割线
  H_SEPARATOR(220,1,7,1)
END_LOAD

; PRESS 方法
PRESS(VS7)
   ; 载入其他对话框
   LM("MASK2")
; PRESS 方法的结束标记
END_PRESS

; PRESS 方法
PRESS(VS8)
```

```
      ；退出对话框
      EXIT
   ；PRESS 方法的结束标记
   END_PRESS
；对话框 1 的结束标记
//END

；带标题和图像的对话框 2 的定义
//M(MASK2/"Position WCS MCS"/"mz961_01.png")
   ；变量定义
   DEF TEXT1 = (I///,"WCS"/WR0,fs2///230,30,120/,,1)
   DEF VAR1 = (R3///,"1. Axis $AA_IW[AX1]"/WR1//"$AA_IW[AX1]"/230,70,150/400,70,100)
   DEF VAR2 = (R3///,"2. Axis $AA_IW[AX2]"/WR1//"$AA_IW[AX2]"/230,90,150/400,90,100)

   DEF TEXT2 = (I///,"MCS"/WR0,fs2///230,120,120/,,1)
   DEF VAR3 = (R2///,"1. Axis $AA_IM[AX1]"/WR1//"$AA_IM[AX1]"/230,160,150/400,160,100)
   DEF VAR4 = (R2///,"2. Axis $AA_IM[AX2]"/WR1//"$AA_IM[AX2]"/230,180,150/400,180,100)
   DEF VAR5 = (R3///,"$P_UIFR G54 AX1"///"$P_UIFR[1,AX1,TR]"/230,200,150/400,200,100)

   ；进度条的定义
   DEF PROGGY0 = (R/0,150//,"Progress bar Operate style"/DT2,DO0//"$R[10]"/5,240,190/5,260,150/6,10)
   DEF PROGGY4 = (R/0,150,50,100/120/,"Progress bar with color change"/DT1,DO0//"$R[10]"/
260,240,190/260,260,150/3,4,,,9,7)

   ；变量定义
   DEF PROGVAL = (R/0,150//,"Value to change Progress bar 0 - 150"///"$R[10]"/5,300,230/260,300,100)

  ；对话框中的软键定义
  HS1=("")
  HS2=("")
  HS3=("")
  HS4=("")
  HS5=("")
  HS6=("")
  HS7=("")
  HS8=("")
  VS1=("")
  VS2=("")
  VS3=("")
  VS4=("")
  VS5=("")
  VS6=("")
  VS7=("Back to Mask")
  VS8=("EXIT")
```

```
   ; Load 方法
   LOAD
      H_SEPARATOR(230,1,7,1)
   END_LOAD

   ; 更改方法
   CHANGE(VAR5)
      ; PI_START 功能
      PI_START("/NC,201,_N_SETUFR")
   ; 更改方法的结束标记
   END_CHANGE

   ; PRESS 方法
   PRESS(RECALL)
      ; 返回主调对话框
      LM("MASK1")
   ; PRESS 方法的结束标记
   END_PRESS

   ; PRESS 方法
   PRESS(VS7)
      ; 返回主调对话框
      LM("MASK1")
   ; PRESS 方法的结束标记
   END_PRESS

   ; PRESS 方法
   PRESS(VS8)
      ; 退出对话框，返回到标准应用程序
      EXIT
   ; PRESS 方法的结束标记
   END_PRESS
; 对话框的结束标记
//END
```

2.2.3 将配置文件保存到 OEM 目录下

保存路径

将配置文件 diag.com 保存到以下路径中：

[系统 oem 目录]\proj

2.2.4 创建在线帮助

在线帮助的内容

创建 HTML 文件 sinumerik_md_1.html。 下段中通过 **name="20060[0]"** 和 **name="20060[1]"** 显示上下文有关的帮助调用框。 帮助文件没有加注释，因为如何创建在线帮助不属于本手册内容。 如何加入在线帮助的说明参见章节设计在线帮助 (页 65)。

```
<html><head><meta http-equiv="Content-Type" content="text/html; charset="UTF-8"/><title></
title></head>
   <body>
      <table border="1" cellspacing="0" cellpadding="2" width="100%">
         <tr>
            <td><a name="20060[0]"><b>20060[0]</b></a></td>
            <td colspan="3">AXCONF_GEOAX_NAME_TAB</td>
            <center>
               <img src="../picture/bitmap3.png">
            </center>
            <td>-</td>
            <td>-</td>
         </tr>
         <tr>
            <td>-</td>
            <td colspan="3">Geometry axis name in channel</td>
            <td>DWORD</td>
            <td>POWER ON</td>
         </tr>
         <tr>
            <td colspan="6">This MD is used to enter the names of the geometry axes
separately for each channel.<br />
         <tr>
            <td>-</td>
            <td colspan="5"> </td>
         </tr>
         <tr>
          <td width="16*">System</td><td width="16*">Dimension</td><td width="16*">Default
value</td>
            <td width="16*">Minimum value</td><td width="16*">Maximum value</td>
            <td width="16*">Protection</td>
         </tr>
         <tr>
            <td>-</td>
            <td>-</td>
            <td>60</td>
            <td>0</td>
            <td>180</td>
            <td>7/3</td>
         </tr>
      </table>
      <p></p>
      <table border="1" cellspacing="0" cellpadding="2" width="100%">
         <tr>
            <td><a name="20060[1]"><b>20060[1]</b></a></td>
            <td colspan="3">AXCONF_GEOAX_NAME_TAB</td>
            <td>-</td>
            <td>-</td>
         </tr>
         <tr>
```

```
            <td>-</td>
            <td colspan="3">Geometry axis name in channel</td><td>BYTE</td>
            <td>POWER ON</td>
         </tr>
         <tr>
            <td colspan="6">This MD is used to enter the names of the geometry axes
separately for each channel.<br />
         <tr>
            <td>-</td>
            <td colspan="5"> </td>
         </tr>
         <tr>
            <td width="16*">System</td>
            <td width="16*">Dimension</td>
            <td width="16*">Default value</td>
            <td width="16*">Minimum value</td>
            <td width="16*">Maximum value</td>
            <td width="16*">Protection</td>
         </tr>
         <tr>
            <td>-</td>
            <td>-</td>
            <td>0</td>
            <td>0</td>
            <td>2</td>
            <td>7/3</td>
         </tr>
      </table>
      <p></p>
   </body>
</html>
```

2.2.5 将在线帮助保存到 OEM 目录下

保存路径

将 HTML 文件 sinumerik_md_1.html 的德语帮助信息保存到以下路径下：

*[系统 oem 目录]*hlp/deu

必要时，需要保存其他语言文件夹（例如：chs、eng、esp、fra、ita ...）。

语种缩写列表参见附录。

2.2.6 将 easyscreen.ini 复制到 OEM 目录中

保存路径

从以下目录中复制文件 easyscreen.ini：

[系统西门子目录]\cfg

复制到

[系统 oem 目录]\cfg.

2.2.7 申明 easyscreen.ini 中的 COM 文件

easyscreen.ini 中的调整

在 OEM 目录下的 easyscreen.ini 中进行如下修改，以便申明配置文件 diag.com。

```
;#########################
;# AREA Diagnosis       #
;#########################
;<=============================================================>
;< OEM Softkey on first horizontal Main Menu                    >
;< SOFTKEY position="7"                                         >
;<=============================================================>
StartFile28 = area := AreaDiagnosis, dialog:= SlDgDialog, menu := DgGlobalHu, startfile := diag.com
```

2.2.8 测试项目

测试对话框调用

转换至“诊断”操作区。 按下水平软键 “START”。

如果 “Run MyScreens” 编译配置文件时发现错误，则该错误会保存在文本文件 easyscreen_log.txt 中（参见章节 错误处理（日志） (页 33)）。

测试上下文相关的帮助图形和在线帮助

测试上下文相关的帮助图形和在线帮助，如果出现错误请检查保存路径。

基本知识 3

3.1 设计文件的结构

简介

新操作界面的描述保存在配置文件中。这些文件自动编译并显示屏幕上的结果。在供货时并不提供配置文件，因此必须首先创建此文件。

创建配置文件和语言文件时使用 UTF 8 编辑器（例如：操作界面的集成编辑器或记事本）。描述可以通过注释来进一步说明。注释字符前会添加“；”。

说明

保存配置文件和语言文件时请注意，应将您所使用的编辑器中的编码设为 UTF 8。

然而，关键字只允许由 ASCII 字符集中的字符组成（UTF 8 编译的 COM 文件中也一样）。否则就无法保证编译和屏幕/对话框的显示。

配置

每个 HMI 操作区都包含固定的登入软键，以进入新创建的对话框。

调用配置文件中的“载入屏幕” (LM) 或者“载入软键栏” (LS)可以重新命名被调对象所在的文件。 采用这种方式可以对配置文件进行分类，例如：同一操作级的所有功能在一个单独的配置文件中。

配置文件的结构

配置文件由以下单元组成：

1. 登入软键说明
2. 对话框定义
3. 变量定义
4. 方法说明
5. 软键栏定义

说明

顺序

必须遵循上文列出的各单元在配置文件中的顺序。

示例：

```
//S (START)                    ; 登入软键定义（可选）
....
//END
//M (.....)                    ; 对话框定义
DEF .....                      ; 变量定义
LOAD                           ; 块描述
...
END_LOAD
UNLOAD
...
END_UNLOAD
...
//END
//S (...)                      ; 软键条定义
//END
```

配置文件的保存

配置文件保存在目录 *[系统用户目录]*/proj 下，也相应的保存在目录 *[系统插件目录]* 和 *[系统 oem 目录]* 下。

转换其他 HMI 应用程序中的文本

从 Codepage 编码的文本文件向 UTF-8 编码文本转换的步骤：

1. 在 PG/PC 上的文本编辑器中打开文本文件。
2. 保存时选择 UTF 8 编码。

使用 Codepage 编码的读取机制还继续被支持。

3.2 操作树的结构

操作树的原理

多个相互关联的对话框构成了一个操作树。 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关联。 点击在这种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。

在每个登入软键下都可以生成一个操作树：

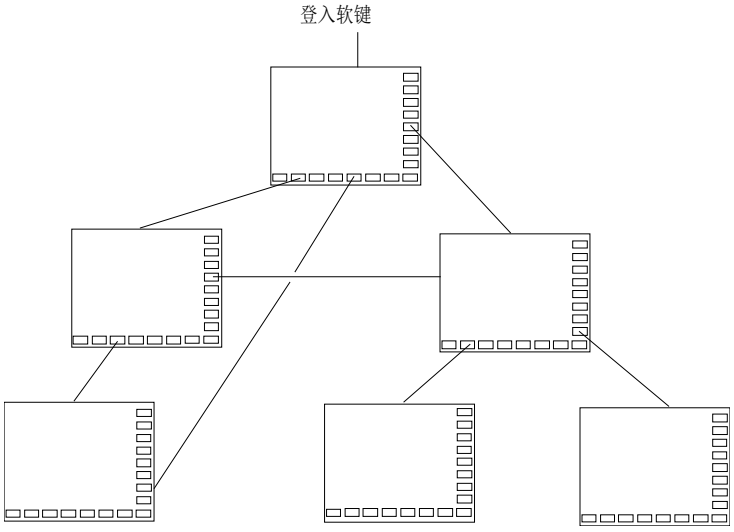


图 3-1 操作树

登入软键

在一个 easyscreen.ini 中指明的配置文件中，定义了一个或者多个软键（登入软键），这些软键可以作为操作的出发点。

软键的定义决定执行下一步动作的自定义对话框或者其他软键栏。

按下登入软键则载入所属的对话框。 同时，属于对话框的相应软键激活。 如果没有设计特定位置，则在标准位置上给出变量。

返回到标准应用程序

可以退出新定义的对话框并返回到标准操作区。

通过<RECALL>（回调）键可以退出新定义的操作界面，如果这个按键还未被设计用于其它用途。

说明

在 PLC 用户程序中调用对话框

除了软键，也可以由 PLC 选择对话框。 DB19.DBB10 中的接口信号用于 PLC → HMI 的信号交换。

3.3 登入软键的定义及功能

3.3.1 定义登入软键

和对话框无关的软键

登入软键是和对话框无关的软键，它不由对话框调用，而是在第一个新对话框**之前**设计。为访问一个登入对话框或者一个登入软键栏，必须对登入软键进行定义。

编程

登入软键的描述块如下构建：

```
//S(Start)                    ; 登入软键开始标记
HS1=(...)                     ; 定义登入软键： 水平软键 SK 1
PRESS(HS1)                    ; 方法
LM...                         ; 功能 LM 或者 LS
END_PRESS                     ; 方法结束
//END                         ; 登入软键结束标记
```

登入软键的允许位置

在操作区中“Run MyScreens”登入软键的允许位置有：

操作区	位置
加工	HSK6
参数	HSK7

操作区	位置
程序	HSK6 和 HSK15 测量循环： HSK13 和 HSK14
程序管理器	HSK2-8 和 HSK12-16，如未被驱动器占用。
诊断	HSK7
调试	HSK7

登入软键在专门的文件中设计。 文件的名称在文件 easyscreen.ini 中声明。 通常为一个操作区的名称（例如：调试区为 startup.com）。 加工区有所例外，针对不同运行方式有不同文件 (ma_jog.com, ma_auto.com)。

带有登入软键的软键栏叫“Start”。 现有的登入软键设计仍可使用。 登入软键与登入软键菜单中各个 HMI 应用（操作区）中软键的汇总功能（Merge）还不被支持。

直到首次对话框调用时，即从全部功能（例如，执行 PRESS 块）都可用的时刻起，才能完全替换菜单或软键栏。

登入软键的配置模板

有关登入软键的全部允许位置的详细描述及其设计见以下目录中的文件 easyscreen.ini：

[系统西门子目录]/cfg

该文件用作登入软键的自定义配置模板。

另见

登入软键表 (页 259)

3.3.2 登入软键的功能

和对话框无关的软键的功能

通过登入软键只可以触发特定的功能。

允许下列功能：

- 通过**功能 LM** 可以装载另一个对话框。 **LM***("名称"[,"文件"])*
- 通过**功能 LS** 可以显示另一个软键栏。 **LS***("名称"[, "文件"][, 合并])*

- 通过**功能 "EXIT"**可以离开新配置的操作界面并返回标准应用程序。
- 通过**功能 "EXITLS"**可以离开当前的操作界面并装载一个定义的软键栏。

PRESS 方法

在描述块内定义软键并在方法 PRESS 内分配功能 "LM" 或者 "LS"。

如果登入软键定义标记为注释（在行开始处用分号 ;）或者已删除配置文件，则登入软键无效。

```
//S(Start)                          ; 开始标记
HS6=("第 1 屏幕")                   ; 水平软键 6，文字为“第 1 屏幕”
PRESS(HS6)                          ; 水平软键 6 的 PRESS 方法
  LM("屏幕 1")                      ; 装载功能屏幕 1，此时必须在同一文件内定义了屏幕 1。
END_PRESS                           ; PRESS 方法结束标记
HS7=("第 2 屏幕")                   ; 水平软键 7，文字为“第 2 屏幕”
PRESS(HS7)                          ; 水平软键 7 的 PRESS 方法
LM("屏幕 2")                        ; 装载功能屏幕 2，此时必须在同一文件内定义屏幕 2。
END_PRESS                           ; PRESS 方法结束标记
//END                               ; 登入块的结束标记
```

示例

```
HS1 = ("新的软键栏")
HS2 = ("没有功能")
PRESS(HS1)
  LS("软键栏 1")                    ; 载入新的软键栏
END_PRESS
PRESS (HS2)                         ; 空的 PRESS 方法
END_PRESS
```

各种登入软键设计

各个登入软键的设计会汇总在一起。 其中，首先从 easyscreen.ini 中读取需要编译的文件的名称。 然后在以下目录中查找后缀为 .com 的文件：

- *[系统用户目录]*\proj
- *[系统 oem 目录]*\proj

- *[系统插件目录]*/proj
- *[系统西门子目录]*/proj

此时所包含的登入软键设计会被汇总到一个设计中，即会对单个软键进行比较。 如果对于一个软键存在两个或多个设计，则总是将较高的值接收到汇总版本。

可能包含的软键栏或对话框被忽略。 如果一个软键包含一个无文件设定的指令，例如 LM(“test”)，因为所需的软键栏或者对话框包含在相同的文件中，则在内部汇总版本中会添加相应的文件名，这样就不需再进行调整。 接下来显示得到的汇总设计。

3.4 错误处理（日志）

概述

如果 “Run MyScreens” 编译配置文件时出现错误，则该错误会保存在文本文件 easyscreen_log.txt 中。 系统只记录当前选中的对话框的错误。之前选中的对话框中的错误记录则被删除。

文件包含以下信息：

- 在执行何种动作时出现错误
- 第一个错误字符的行号和列号
- 设计文件中所有的错误行
- DEBUG 功能生成的记录

说明

在每个记录前会附加一个带有方括号的当前时间戳。 这可为（例如）分析对时间要求苛刻的配置提供帮助。

easyscreen-log.txt 的保存

文件 easyscreen_log.txt 保存在以下目录中：

[系统用户目录]/log

句法

只有在定义了登入软键并且设计了带开始和结束标记的对话框和定义行后，才开始编译句法。

```
//S(Start)
HS6=("第 1 屏幕")
PRESS(HS6)
  LM("Maske1")
END_PRESS
//END

//M(Maske1)
  DEF Var1=(R)
  DEF VAR2 = (R)
  LOAD
  VAR1 = VAR2 + 1              ；日志中的错误报告，因为 VAR2 没有数值
...
//END

；正确，比如：
//M(Maske1)
   DEF Var1=(R)
   DEF VAR2 = (R)

   LOAD
      VAR2 = 7
      VAR1 = VAR2 + 1 ;
...
```

3.5 easyscreen.ini 的说明

自 SINUMERIK Operate V4.7 起，easycreen.ini 还扩展了本章节中描述的记录。easyscreen.ini 位于目录 *[系统西门子目录]*cfg 下。

帮助画面起始位置

easyscreen.ini 中的记录：

```
[GENERAL]
HlpPicFixPos=true
```

说明：

帮助画面的起始位置会定位在所设置的像素位置，与分辨率无关（默认=true）。

拉伸特性，默认行高和行间距

easyscreen.ini 中的记录：

```
[GENERAL]
SymmetricalAspectRatio=false
DefaultLineHeight=18
DefaultLineSpacing=3
```

说明：

- “SymmetricalAspectRatio”来确定，根据 X 和 Y 方向上确定的屏幕分辨率来调整配置时是否要使用相同的拉伸系数。
  - “false”（默认）：使用宽屏分辨率时，各栏和图片会在 Y 方向上被压缩（基于 640x480 的非对称拉伸）。例如，一个以 640x480 设置的正方形在宽屏面板上会被垂直压缩，而成为一个长方形。
  - “true”：在 X 和 Y 方向上以相同的拉伸系数进行拉伸，此时各栏和图片会以 640x480 为基础并保持其原先设置的比例。例如，一个以 640x480 设置的正方形在宽屏面板上仍然会显示为正方形。
- “DefaultLineHeight” 和 “DefaultLineSpacing” 可在 640x480 的基础上确定默认行高（默认值：18 像素）和行间距（默认值：3 像素）。只有在短文本或输入/输出栏的位置设置中未给定 Y 位置或高度，此项设置才会一直有效。

与分辨率相关的屏幕位置：窗体面板

easyscreen.ini 中的记录：

```
[640x480]
MyPanel = x:=0, y:=220, width:=340, height:=174
[800x480]
MyPanel = x:=0, y:=220, width:=420, height:=174
...
```

说明：

屏幕位置可通过以下方式定义：

- 屏幕位置可以“基于 640x480 的像素”给定。
  示例：
  `//M(MyMask/"MyCaption"/"\\myhelp.png"/`**0,219,335,174**）
- 屏幕位置可从 Operate 默认屏幕布局（Operate Standard-Screen-Layout）中，例如来自“机床”区域的屏幕布局“slmastandardscreenlayout.SlMaStandardScreenLayout”中的“FormPanel4”（“辅助功能”）
  `//M(MyMask/"MyCaption"/"\`
  `\myhelp.png"/`**"slmastandardscreenlayout.SlMaStandardScreenLayout.FormPanel4"**）
  连接到与分辨率相关的位置。
  这样可更为方便地使 Operate 的默认窗体交叉淡入淡出，或者可将 Easyscreen 屏幕精确地放置到位。
  示例：
  `//`
  `M(Mask/"Mask"//`**"slstandardscreenlayout.SlStandardScreenLayout.LowerForm"**）
  但也可以使用其他任意的屏幕布局。
- 屏幕位置可以连接到 easyscreen.ini 文件中自行定义的、与分辨率相关的窗体面板的定义。
  示例：
  `//M(MyMask/"MyCaption"/"\\myhelp.png"/`**"MyPanel"**）

3.6 针对改用 “Run MyScreens” 人员的说明

说明

使用 NCU 中的 HMI Operate 时要注意 CF 卡上的所有文件名都是以小写字母保存的（如：com、png、txt）。

说明

保存配置文件和语言文件时请注意，应将您所使用的编辑器中的编码设为 UTF 8。

图形文件

图形文件必须保存为 PNG 格式（xxx.png）。

数据（例如：用于用于 OEM 自定义的数据）必须保存在

[系统 oem 目录]/ico/[分辨率]

目录中。

更多信息另见 使用图/图形 (页 56) 一章。

调整配置文件

根据以下步骤检查配置文件：

- 比较登入软键和当前允许的软键（参见附录中的登入软键列表），必要时进行调整。
- 根据上述“图形文件”重命名相连的图形文件。

数据（例如：用于 OEM 自定义的数据）保存在以下目录中：

[系统 oem 目录]/proj

[系统用户目录]/proj

[系统插件目录]/proj

调整帮助文件

所有帮助文件必须以 UTF 8 格式保存。检查已有文件并使用合适的编辑器重新保存。

HTML 文件保存在以下目录中（例如：德语）：

[系统 oem 目录]/hlp/deu

[系统用户目录]/hlp/deu

[系统插件目录]/hlp/deu

其他语言的目录必须根据相应的语种缩写（参见附录）保存。

检查 “Run MyScreens” 授权

检查所添加的对话框数目是否超出了最大 5 个对话框的基本范围。

扩展对话框数量，需要下列软件选件：

- SINUMERIK 828D/840D sl, SINUMERIK Integrate Run MyScreens (6FC5800-0AP64-0YB0)
- SINUMERIK 840D sl, SINUMERIK Integrate Run MyScreens + Run MyHMI (6FC5800-0AP65-0YB0)
- SINUMERIK 840D sl, SINUMERIK Integrate Run MyHMI / 3GL (6FC5800-0AP60-0YB0)
- SINUMERIK 840D sl, SINUMERIK Integrate Run MyHMI / WinCC (6FC5800-0AP61-0YB0)

3.7 扩展配置句法

自 SINUMERIK Operate V4.7 起，提供了用于屏幕定义、变量定义、软键定义以及表格列定义的简化句法。该句法改善了可读性并缩短了等待时间。特性和属性可以任意顺序给定，空白记录被取消。相比于以前的句法，特性和属性列表不再使用圆括号 “(” 和 “)”，而是使用圆弧括号 “{” 和 “}” 。

特性和属性按以下方式给定：
{<名称> = <值>, <名称> = <值>, …}

以前的句法仍然兼容。

屏幕定义的扩展句法

//M {<屏幕名称> [,HD=<标题>] [,HLP=<图片>] [,X=<X 位置>] [,Y=<Y 位置>] [,W=<宽度>] [,H=<高度>] [,VAR=<>系统变量或用户变量] [,HLP_X=<辅助画面 X 位置>] [,HLP_Y=<辅助画面 Y 位置>] [,CM=<列对齐>] [,CB=对话框打开方式] [,XG=<将辅助画面编译为 X3d 图片>] [,PANEL=<已链接 FormPanels 的名称>][,MC=<屏幕背景色>][,HD_AL=<屏幕标题对齐>][,LANGFILELIST=<屏幕专用语言文件列表>]}

示例：
```
//M{VariantTest, HD="My Mask"}
```

变量定义的扩展句法

DEF <变量名称> = {[TYP=<类型>] [,MIN=<最小值>] [,MAX=<最大值>] [,TGL=<转换值>] [,VAL=<预设值>] [,LT=<长文本>] [,ST=<短文本>] [,GT=<图片文本>] [,UT=<单位文本>] [,TT=<提示框文本>] [,TG=<转换选项>] [,WR=<输入模式>] [,AC=<访问级>] [,AL=<文本对齐>] [,FS=<字体大小>] [,LI=<极限值处理>] [,UR=<刷新率>] [,CB=<对话框打开方式>] [,HLP=<辅助画面>] [,VAR=<系统变量和用户变量>] >] [,TXT_X=<短文本 X 位置>] [,TXT_Y=<短文本 Y 位置>] [,TXT_W=<短文本宽度>] [,TXT_H=<短文本高度>] [,X=<输入/输出栏 X 位置>] [,Y=<输入/输出栏 Y 位置>] [,W=<输入/输出栏宽度>] [,H=<输入/输出栏高度>] [,UT_DX=<输入/输出栏之间的间距和单位文本>] [,UT_W=<单位文本宽度>] [,BC=<输入/输出栏背景色>] [,FC=<输入/输出栏前景色>] [,BC_ST=<短文本背景色>] [,FC_ST=<短文本前景色>] [,BC_GT=<图片文本背景色>] [,FC_GT=<图片文本前景色>] [,BC_UT=<单位文本背景色>] [,FC_UT=<单位文本前景色>] [,SC1=<进度条信号色 1>] [,SC2=<进度条信号色 2>] [,SVAL1=<进度条阈值 1>] [,SVAL2=<进度条阈值 2>] [,DT=<显示类型>] [,DO=<显示对齐>] [,OHLP=<在线帮助>][,LINK_TGL=<已链接转换变量的名称>]}

示例：

```
DEF MyVar5={TYP="R2", ST="MyVar5", VAL=123.4567, OHLP="myhelp.html", MIN=100.1,
MAX=200.9}
DEF MyVar2={TYP="I", TGL="*1,2,3", VAL=1}
DEF MyVar3={TYP="R2", TGL="*0=""Aus"", 1=$80000", VAL=1}
DEF MyVar4={TYP="R2", TGL="*MyArray",VAL=1}
DEF MyVar1={TYP="R2", TGL="%grid99", X = 0, W=300, H=200}
DEF MyVar6={TYP="R2", TGL="+$80000", VAR="$R[10]", ST="Textoffset"}
```

软键定义的扩展句法

SK = {[ST=<名称>] [,AC=<访问级>] [,SE=<状态>]}

示例：

```
HS1={ST="""MySk""", AC=6, SE=1}
HS3={ST="SOFTKEY_CANCEL"}
HS5={ST="[$81251,""\\sk_ok.png""]"}
HS8={ST="[""Test"",""\\sk_ok.png""]"}
```

表格列定义的扩展句法

{[TYP=<类型>] [,MIN=<最小值>] [,MAX=<最大值>] [,LT=<长文本>] [,ST=<短文本>] [,WR=<输入模式>] [,AC=<访问级>] [,AL=<文本对齐>] [,FS=<字体大小>] [,LI=<极限值处理>] [,UR=<刷新率>] [,HLP=<辅助画面>] [,VAR=<系统变量或用户变量>] >] [,W=<列宽度>] [,OF1=<偏移 1>] [,OF2=<偏移 2>] [,OF3=<偏移 3>]}

示例：

```
DEF MyGridVar={TYP="R", TGL="%MyGrid1", X=10, W=550, H=100}
//G(MyGrid1/0/5)
   {TYP="I", ST="Index", WR=1, VAR="1", W=80, OF1=1}
   {TYP="S", LT="LongText2", ST="Text", WR=1, VAR="$80000", AL=2, W=330, OF1=1}
   {TYP="R3", LT="LongText1", ST="R9,R11,R13,R15", WR=2, VAR="$R[1]", W=110, OF1=2}
//END
```

3.8 SmartOperation 及 MutliTouch 操作

以下设置可专门用于调整 SmartOperation 及 MutliTouch 操作：

- 自动将栏高及栏宽调节至最小可操作的栏尺寸及行间距（屏幕属性 MA）。
- 提高分辨率时精确按像素优化拉伸各栏（屏幕属性 PA）。
- 与字体成比例设置栏高及行间距（屏幕属性 FA）。
- 实现各栏自由滚动，以免虚拟键盘覆盖输入栏（屏幕属性 KM）。

详细信息，请参见定义对话框属性 (页 43)一章的“编程”一节。

对话框

4

4.1 对话框的结构和组成单元

4.1.1 定义对话框

定义

对话框是操作界面上的一个组成部分，操作界面包含标题行、对话框单元和/或图形、显示消息的输出行以及 8 个水平软键和 8 个垂直软键。

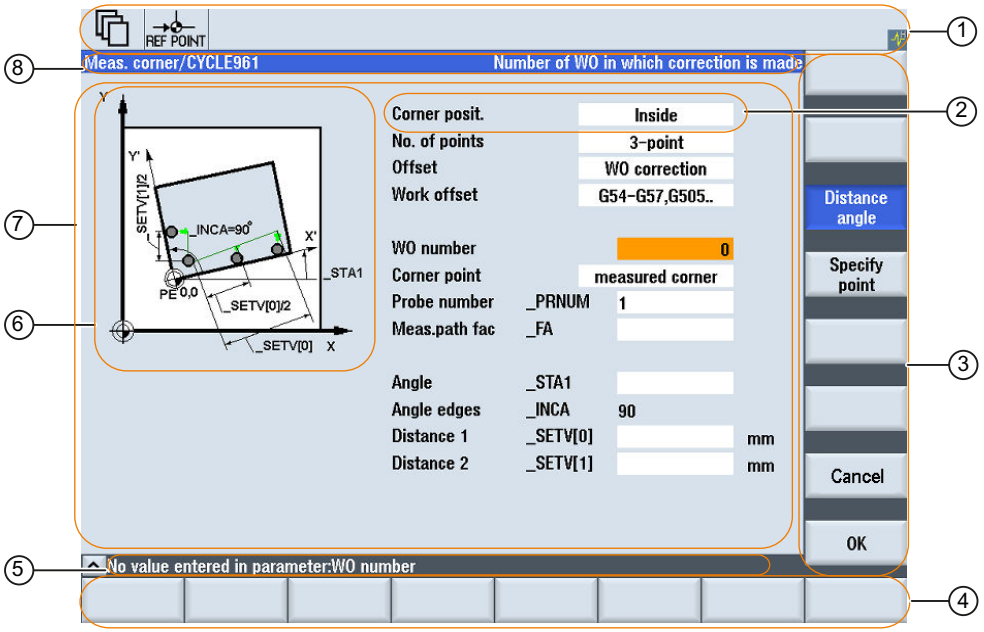
对话框单元包含：

- 变量
  - 极限值/转换栏
  - 变量预设值
- 帮助图形
- 文本
- 属性
- 系统或者用户变量
- 短文本的位置
- 输入/输出栏的位置
- 颜色

对话框的属性：

- 标题
- 图形
- 尺寸
- 系统或者用户变量

- 图形的位置
- 属性



① 机床状态显示（“标题”）
② 对话框单元
③ 8 个垂直软键
④ 8 个水平软键
⑤ 诊断信息的输出
⑥ 图形
⑦ 对话框
⑧ 对话框的标题栏，包含标题和长文本

图 4-1 对话框结构

概述

基本上对话框说明（说明块）的结构如下：

说明块	注释	参考章节
//M...	; 对话框的开始标记	
DEF Var1=... ...	; 变量	参见章节“变量”

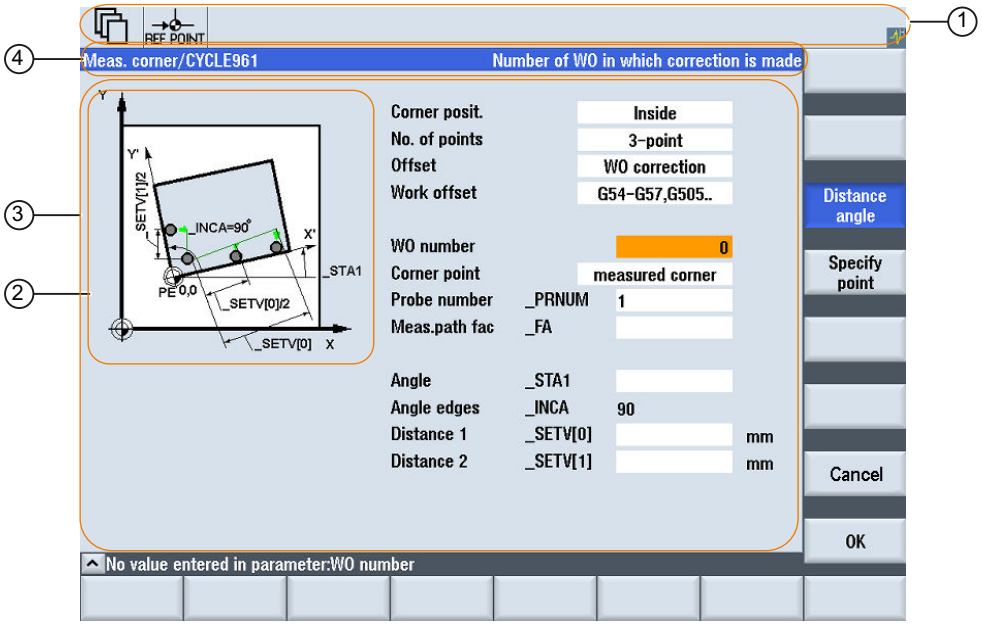
说明块	注释	参考章节
HS1=(...) ...	; 软键	参见章节“软键栏”
PRESS(HS1) LM... END_PRESS	; 方法的开始标记 ; 动作 ; 方法的结束标记	参见章节“方法”
//END	; 对话框的结束标记	

在对话框的说明块中，首先定义在对话框的对话框单元中显示的不同变量，然后定义水平和垂直软键。 然后在方法中设计不同的动作。

4.1.2 定义对话框属性

说明

使用对话框的开始标记可以同时定义对话框的属性。



① 机床状态显示（“标题”）

② 图形

③ 对话框

④ 对话框的标题栏，包含标题和长文本

图 4-2 对话框属性

编程

句法：	**//M**(名称/[标题]/[图形]/[尺寸]/[系统或用户变量]/[图形位置]/[属性]/屏幕专用语言文件列表) 另见章节 扩展配置句法 (页 38)。
说明：	定义对话框

参数：	名称			对话框的名称
	标题			对话框的文本标题或者从和语言相关的文本文件中调用文本（例如：$85011）
	图形			图形文件，路径在双引号内
	尺寸			对话框的位置和大小，单位像素（和左/右边缘的间距、宽度、高度），以屏幕的左上角为基准。数据值用逗号相隔。
	系统或者用户变量			指定当前光标位置的系统和用户变量。可以通过系统或者用户变量将光标位置传送给 NC 或 PLC。 第一个变量为索引 1。顺序按照变量的设计顺序。
	图形的位置			图形的位置，单位像素（和左/上边缘的间距），以对话框的左上角为基准。数据值用逗号相隔。
	属性			给定的属性用逗号相隔。 允许的属性有：
		CM		Column Mode:列对齐
			CM0	预设置：每行单独分列。
			CM1	以包含最多列的行为标准分列。
		CB		CHANGE 块:打开对话框时的属性：定义变量时规定的 cb 属性，优先于定义对话框时的总定义 另见 对话框打开方式（属性 CB） (页 265)。
			CB0	预设置：在打开对话框时处理所有 CHANGE 部分。
			CB1	只有在附属的值改变后才处理 CHANGE 部分。
		XG		连接 X3D 动画作为辅助画面（只用在工步链编程中）。 示例： `//M(Meas/ $85605/"myx3dhelpfile.hmi,,Z_Animation,,G17"/// 30,10/XG1)`
			XG0	缺省值 ＝ 0
			XG1	屏幕属性 XG 必须已在屏幕定义中设为 1，运行期间无法进行更改。 说明： 屏幕属性 HLP 中的数据也应进行相应的设置。
		AL		使用屏幕属性 AL 可以控制屏幕标题的对齐。 示例： 窗口标题“Set password”居中

			//M(MY_PWD_SET/"Set password"//"EasyPwdModalLayout"//AL2)
		AL0	左对齐，缺省设置
		AL1	右对齐
		AL2	居中
	KM		当 MultiTouch 上的虚拟键盘显示时，屏幕上的自由滚动条（SIGfwScrollArea）可能会受以下因素影响： ● 屏幕属性为"KM"的屏幕定义行中（KeyboardMode） 示例： //M(MyMTMask/"MultiTouch Mask"////KM1) ● 在 easyscreen.ini 中作为所有 Run MyScreens 屏幕的全局设置 示例： [GENERAL] DefaultVirtualKeyboardMode=1 提示： 如果在屏幕设置中精确设置了屏幕属性 KM，则它的优先级高于 easyscreen.ini 中的全局设置。 （另见章节 SmartOperation 及 MutliTouch 操作 (页 40)）
		KM1	自由滚动条激活（缺省）
		KM0	自由滚动条禁用
	PG		对齐并预览带 PNG 图片的窗口标题（只在编辑器中有效！） 示例： //M(MyPg1SampleMask/"My PG1 Sample Mask"////PG1)
		PG0	(缺省)
		PG1	对齐并预览带 PNG 图片的窗口标题 路径说明（最左），屏幕名称（右） 在此模式下无法显示长文本。您可另行选择使用提示框。
	MA		使用 MultiTouch 操作(MutliTouch Adjustment)时自动匹配栏高 MultiTouch 操作面板的匹配设置包括将各栏（进度条、GIF 动图、CustomWidget 除外）扩大至最小可操作尺寸（宽、高）以及行间距。 MultiTouch 匹配可

		● 在 easyscreen.ini 中作为所有 Run MyScreens 屏幕的全局设置，例如 [GENERAL] DefaultMultiTouchAdjustmentLevel=1 ● 或者可针对具体屏幕在屏幕定义行中使用屏幕属性“MA”将 easyscreen.ini 中的全局设置覆盖掉。例如 //M(MyMTMask/"MultiTouch Mask"/////MA1) 如果在屏幕设置中明确设置了屏幕属性 MA，则它的优先级高于 easyscreen.ini 中的全局设置。 （另见章节 SmartOperation 及 MutliTouch 操作 (页 40)）
	MA0	未进行任何匹配
	MA1	只匹配未设置上间距和/或栏高/栏宽的栏 – 也包括使用缺省位置并且由 Run MyScreens 来计算上间距和/或栏高/栏宽的栏。(缺省)
	MA2	匹配所有栏，不考虑所设置的栏大小。
PA		提高分辨率时精确按像素优化扩展各栏（Pixel Fine Adjustment）。 此时会首先基于 640x480 计算所有的栏位置，然后根据相应的水平或垂直拉伸系数进行缩放。但是该属性也有缺点，如若“拉伸系数不合适”则会导致取整误差。比如栏高或行与行之间的间距可能会“跳过”一个像素。为避免出现这种情况，而提供了“Pixel Fine Adjustment”模式，在此模式下栏位置会直接根据当前分辨率来确定。 //M(MyMask/"My Mask"/////PA1) 该设置也可在 easyscreen.ini 中作为所有 Run MyScreens 屏幕的全局设置，例如 [GENERAL] DefaultPixelFineAdjustment=1 如果在屏幕设置中明确设置了屏幕属性 PA，那么在 easyscreen.ini 中则具有比全局设置更高的优先级、 如果屏幕通过 Font Adjustment = FA1 来显示，那么对于该屏幕 Pixel Fine Adjustment 模式也可自动生效。 （另见章节 SmartOperation 及 MutliTouch 操作 (页 40)）
	PA0	拉伸属性保持当前设置，即先基于 640x480 计算位置，然后拉伸。 （由于兼容性的原因：缺省）
	PA1	提高分辨率时精确按像素优化拉伸各栏
FA		与字体成比例设置栏高及行间距（Font Adjustment）

			该设置也可在 easyscreen.ini 中作为所有 Run MyScreens 屏幕的全局设置，例如
			`[GENERAL]`
			`DefaultFontAdjustment=1`
			如果在屏幕设置中明确设置了屏幕属性 FA，则它的优先级高于 easyscreen.ini 中的全局设置。
			DefaultLineHeight 和 DefaultLineSpacing 的设置在 Font Adjustment 模式生效时无用。
			（另见章节 SmartOperation 及 MutliTouch 操作 (页 40)）

		FA0	栏高和行间距按当前方式拉伸（由于兼容性的原因：缺省）
		FA1	与字体成比例设置栏高及行间距（自动设置屏幕属性 PA=1）
		FA2	与 FA1 相同，但会额外与字体成比例设置 X 坐标和栏宽。 （自动设置屏幕属性 PA=1） （只在同时使用属性 XG=1 时可能！）

	MC		屏幕背景色(Mask Color)
			示例：
			设置屏幕背景色为蓝色(= 6)
			`//M(MyMask/"MyMask"//////6)`
			或者
			`PRESS(HS1)`
			`  MC=6`
			`END_PRESS`

	屏幕专用语言文件列表		用逗号隔开

存取对话框属性

在方法（例如：PRESS 块）的范围内可以读取和写对话框的以下属性：

- HD = 标题 (Header)
- HLP = 辅助画面
- VAR = 系统或用户变量
- MC = 屏幕背景色
- CM = 列对齐（只读）
- CB = 打开方式（只读）

- XG = 连接 X3d（只读）
- AL = 屏幕标题对齐（只读）

示例

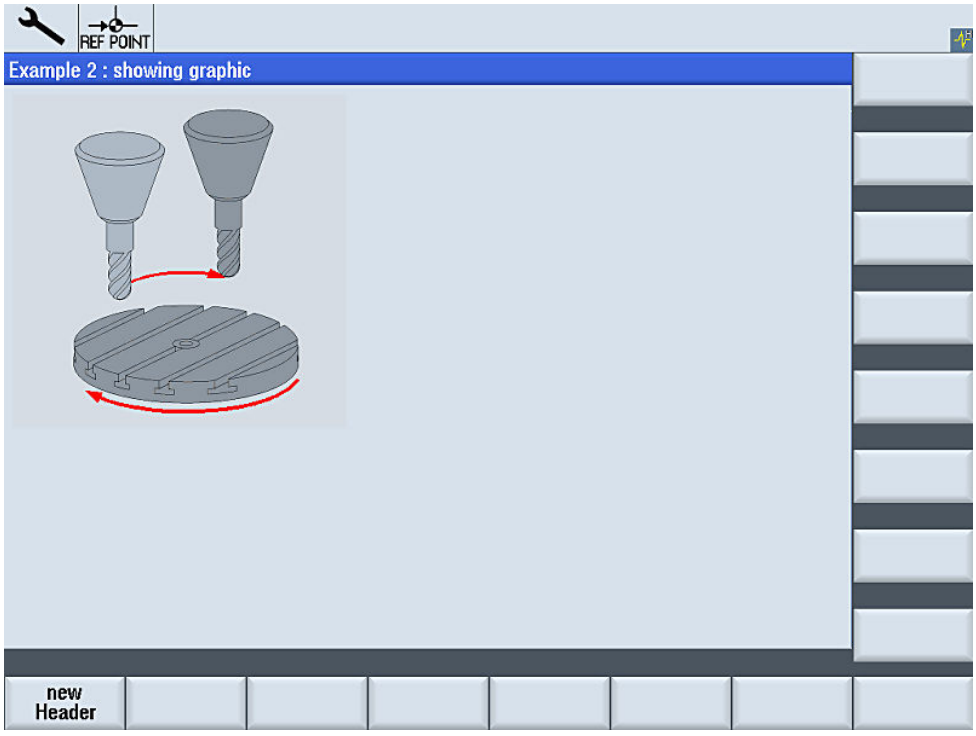


图 4-3 "Example 2: showing graphic"

```
//S(Start)
HS7=("Example", se1, ac7)

PRESS(HS7)
  LM("Mask2")
END_PRESS

//END
//M(Mask2/"Example 2 : showing graphic"/"example.png")
HS1=("new%nHeader")
HS2=("")
HS3=("")
HS4=("")
HS5=("")
```

```
HS6=("")
HS7=("")
HS8=("")
VS1=("")
VS2=("")
VS3=("")
VS4=("")
VS5=("")
VS6=("")
VS7=("")
VS8=("")

PRESS(HS1)
  Hd= "new Header"
END_PRESS
...
//END
```

参见

操作区“自定义”的编程示例 (页 249)

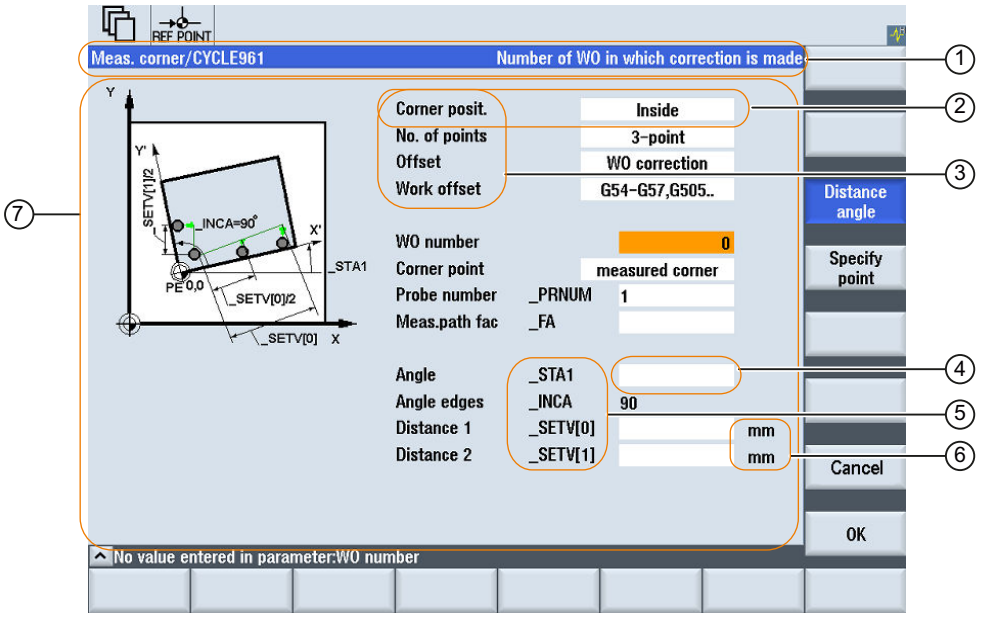
4.1.3 定义对话框单元

对话框单元

对话框单元是变量的可见部分，即短文本、图形文本、输入/输出栏、单位文本和提示框。 对话框单元位于对话框主体的行中。 每行可以定义一个或者多个对话框单元。

变量属性

所有变量仅在激活的对话框中有效。 通过定义变量指定其属性。 在方法（例如：PRESS 方法）的范围内可以存取对话框属性值。



① 对话框的标题栏，包含标题和长文本

② 对话框单元

③ 短文本

④ 输入/输出栏

⑤ 图形文本

⑥ 单位文本

⑦ 对话框主体

图 4-4 对话框单元

编程一览

在圆括号中通过逗号隔开各个参数。

DEF *Bezeichner* =	**Bezeichner = 变量名称**
	变量类型
	/[极限值或转换栏]
	/[预设值]
	/[文本（长文本，短文本|图，图形文本，单位文本）]

	/[属性]
	/[帮助画面]
	/[系统变量或者用户变量]
	/[短文本位置]
	/[输入栏/输出栏位置（左、上、宽度、高度）]
	/[颜色]
	/[在线帮助] (页 65)

另见

变量参数 (页 79)

4.1.4 定义多列对话框

概述

在对话框中，一行可以显示多个变量。 在这种情况下，设计文件中的所有变量都定义在一个定义行内。

```
DEF VAR11 = (S///"Var11"), VAR12 = (I///"Var12")
```

为了可以更清除地显示设计文件中的各个变量，在每个变量定义和其紧随的逗号后可以换行。

关键字“DEF”总是表示新的一行的开始：

```
DEF Tnr1=(I//1/"","T ",""/wr1///,,10/20,,50),
   TOP1=(I///,"Typ="/WR2//"$TC_DP1[1,1]"/80,,30/120,,50),
   TOP2=(R3///,"L1="/WR2//"$TC_DP3[1,1]"/170,,30/210,,70),
   TOP3=(R3///,"L2="/WR2//"$TC_DP4[1,1]"/280,,30/320,,70),
   TOP4=(R3///,"L3="/WR2//"$TC_DP5[1,1]"/390,,30/420,,70)
DEF Tnr2=(I//2/"","T ",""/wr1///,,10/20,,50),
   TOP21=(I///,"Typ="/WR2//"$TC_DP1[2,1]"/80,,30/120,,50),
```

```
TOP22=(R3///,"L1="/WR2//"$TC_DP3[2,1]"/170,,30/210,,70),
TOP23=(R3///,"L2="/WR2//"$TC_DP4[2,1]"/280,,30/320,,70),
TOP24=(R3///,"L3="/WR2//"$TC_DP5[2,1]"/390,,30/420,,70)
```

说明

在设计多列对话框时注意，列过多可能会导致系统变慢！

4.1.5 口令对话框

使用标准口令对话框

您可在屏幕配置中连接以下预定义的口令对话框：

- 设置口令

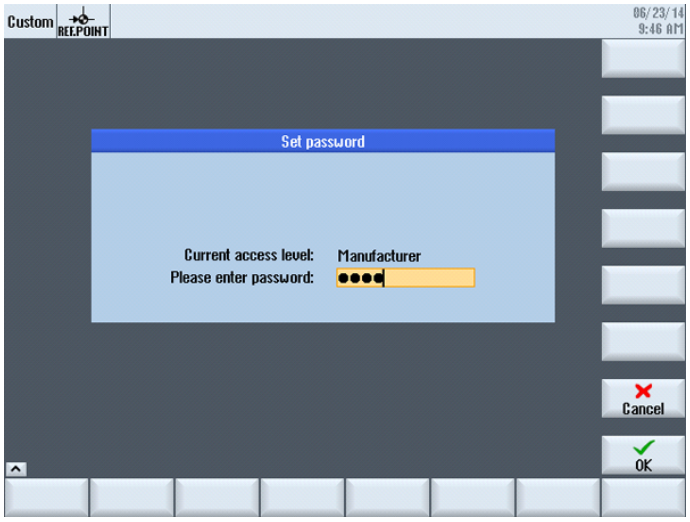


图 4-5 设置口令对话框

- 更改口令

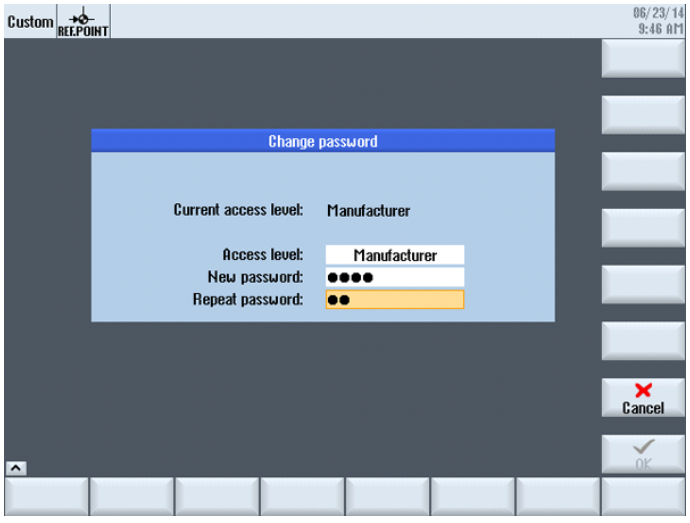


图 4-6 更改口令对话框

- 删除口令

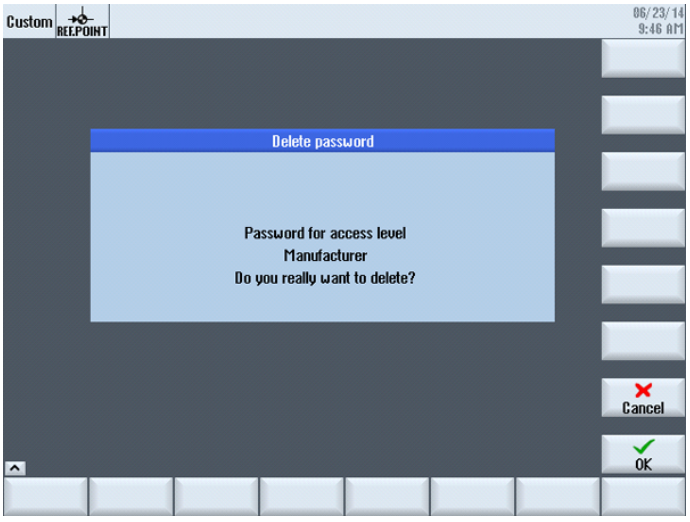


图 4-7　　删除口令对话框

说明

使用这些对话框可为您提供有关口令的各项功能。这些对话框与 SINUMERIK Operate 的对话框并不一样。

调用对话框既可以通过功能 Load Softkey (LS)进行，也可以通过 Load Mask (LM)进行：

1. 通过功能 Load Softkey (LS)调用：
   `LS("Passwd","slespasswd.com",1)`
   垂直软键扩展：
   - 软键 1：设置口令
   - 软键 2：更改口令
   - 软键 3：删除口令
2. 或者也可以直接通过调用功能 Load Mask (LM)跳转到这三个屏幕：
   - 设置口令：`LM("PWD_SET", "slespasswd.com",1)`
   - 更改口令：`LM("PWD_CHG", "slespasswd.com",1)`
   - 删除口令：`LM("PWD_CLEAR", "slespasswd.com",1)`

4.1.6 使用图/图形

使用图形

分为：

- 使用图/图形
- 帮助图形，例如：图示说明各个变量并突出显示在图形区中。
- 可以设计其它帮助图形替代可任意定位的短文本或者输入/输出栏。

保存位置

与相连监视器的分辨率相匹配的图片，请先从相应的分辨率目录下按顺序查找：

[系统用户目录]/ico/ico<分辨率>
[系统 oem 目录]/ico/ico<分辨率>
[系统插件目录]/ico/ico<分辨率>

如果未显示或未找到图片，将其复制到分辨率为 640 x 480 像素的目录中：

[系统用户目录]/ico/ico640
[系统 oem 目录]/ico/ico640
[系统插件目录]/ico/ico640

说明
在面板分辨率不同时，图片会按比例定位。

4.2 定义软键栏

定义

所有水平软键和所有垂直软键分别布置在一起称作软键栏。另外针对已有的软键栏，还可以定义其它软键栏，可以部分或者完全覆盖已有的软键栏。

确定软键名称。不必占用所有软键。

HSx x 1 - 8, 水平软键 1 至 8

VSy y 1 - 8, 垂直软键 1 至 8

原则上软键栏描述（描述块）如下构建：

说明块	注释	参考章节
//S...	; 软键栏开始标记	
HSx=...	; 定义软键	
PRESS(HSx) LM... END_PRESS	; 方法的开始标记 ; 动作 ; 方法的结束标记	参见章节“方法”
//END	; 软键栏结束标记	

说明

定义软键栏也同时分配软键属性。

编程

句法：	//S*(名称)*
	...
	//END
		;软键栏开始标记
		
		;软键栏结束标记

	另见章节 扩展配置句法 (页 38)。	
说明：	定义软键栏	
参数：	名称	软键栏名称
	文本或图形文件名称	
句法：	SK = *(文本[，访问等级][，状态][，软键图对齐][，以软键图为基准的文本对齐])*	
说明：	定义软键	
参数：	SK	软键，例如： HS1 到 HS8, VS1 到 VS8

	文本	给定文本
	图文件名称	"\\my_pic.png" 或者通过单独的文本文件 $85199，例如：（和语言相关的）文本文件中的下列文本：85100 0 0 "\\my_pic.png"。 所能显示的软键图片的大小取决于所使用的 OP:
		OP 08: 640 x 480 mm → 25 x 25 像素 OP 010: 640 x 480 mm → 25 x 25 像素 OP 012: 800 x 600 mm → 30 x 30 像素 OP 015: 1024 x 768 mm → 40 x 40 像素 OP 019: 1280 x 1024 mm → 72 x 72 像素
	访问等级	ac0 到 ac7（ac7：缺省设置）
	状态	se1:可见（预设置） se2:不可操作（灰色标签） se3:高亮显示（最后操作的软键）
	软键图对齐	PA (PictureAlignment)
		有效值： 0:左 1:右 2:居中 3:上（默认） 4:下
	以软键图为基准的文本对齐	TP (TextAlignedToPicture)
		有效值： 0:文本和图形未对齐 1:文本和图形已对齐（默认）

说明

对于软键标签，用 %n 进行换行。
最多 2 行，每行各 9 个字符。

分配访问等级

操作员只能访问符合其访问等级和各个低于其访问等级的信息（另见章节 访问等级列表 (页 262)）。

示例

```
//S(软键栏 1)                                  ;软键栏开始标记
HS1=("新建", ac6, se2)                         ; 定义软键 HS1：软键标签为 "新建"，保护等级为 6，状态为"不可操作"。
HS2=("\\bild1.png")                            ; 指定软键图片
HS3=("退出")
HS4=(["Confirm","\\sk_ok.png"],PA0,TP1)        ; 带文本和图形的软键，文本="Confirm"，图形="sk_ok.png"，软键图对齐：左，文本 和图形对齐

VS1=("子窗口")
VS2=($85011, ac7, se2)                         ; 定义软键 VS2：软键标签来自语言文件，保护等级为 1，状态为"不可操作"
VS3=("取消", ac1, se3)                         ; 定义软键 VS3：软键标签为"取消"，保护等级为 1，状态为"高亮"
VS4=("确认", ac6, se1)                         ; 定义软键 VS4：软键标签为"确认"，保护等级为 6，状态为"可见"
VS5=(SOFTKEY_CANCEL,,se1)                      ; 定义标准软键 VS5：软键标签为"取消"，状态为"可见"
VS6=(SOFTKEY_OK,,se1)                          ; 定义标准软键 VS6：软键标签为"确认"，状态为"可见
VS7=(["\\bild1.png","OEM 文本"],,se1)          ; 定义软键 VS7：指定图片，软键标签为"OEM 文本"，状态为"可见"
VS8=(["\\bild1.png",$83533],,se1)              ; 定义软键 VS8：指定图片，软键标签来自语言文件，状态为"可见"

PRESS(HS1)                                     ; 方法开始标记
  HS1.st="计算"                                ; 指定软键标签
...
END_PRESS                                      ; 方式结束标记

PRESS(RECALL)                                  ; 方法开始标记
  LM("屏幕 21")                                ; 载入对话框
END_PRESS                                      ; 方式结束标记
//END                                          ; 软键栏结束标记
```

4.2.1 运行时改变软键属性

说明

文本、访问等级和状态的属性可以在运行期间在方法中改变：

编程

句法：	SK.st = "文本" SK.ac = 访问等级 SK.se = 状态 SK.pa = “软键图对齐” SK.tp = “以软键图为基准的文本对齐”		；软键标签 ；软键访问等级 ;软键状态 ；带图形的软键 ；带图形和标签的软键
说明：	分配属性		
参数：	文本		引号中的标签文本
	访问等级		值范围： 0 ... 7
	状态	1: 2: 3:	可见并可操作 不可操作（灰色标签） 高亮显示（最后操作的软键）
	软键图对齐	0: 1: 2: 3: 4:	左 右 居中 上（默认） 下
	以软键图为基准的文本对齐	0: 1:	文本和图形未对齐 文本和图形已对齐（默认）

示例

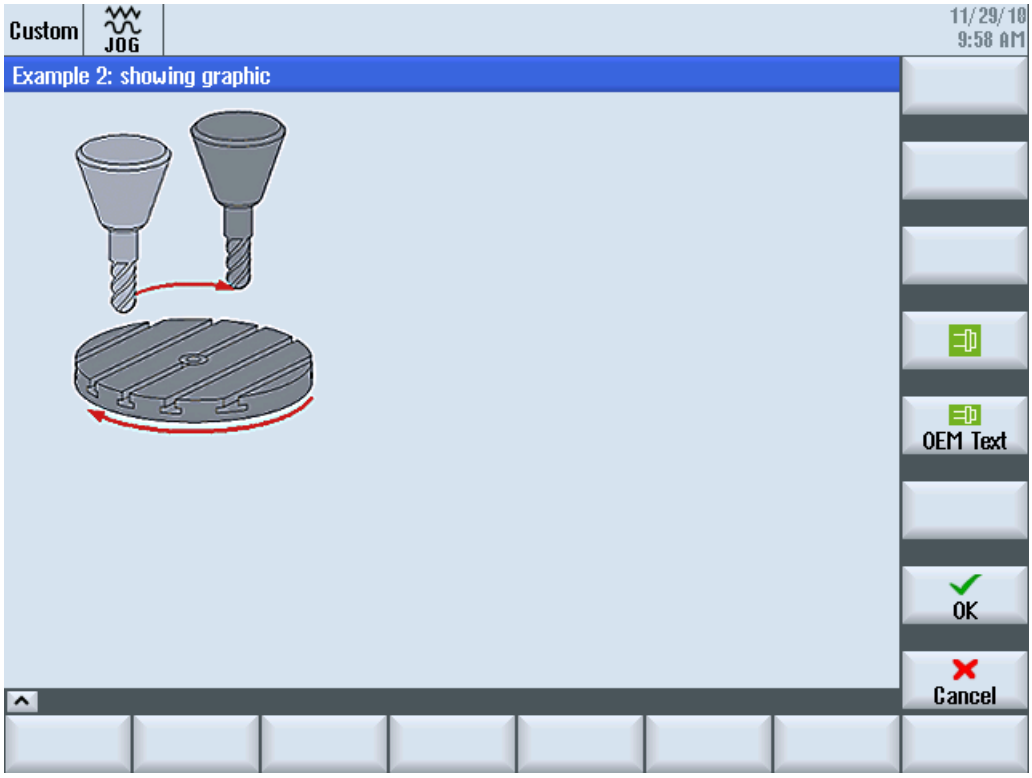


图 4-8　　示例 3： 图形和软键

```
//S(Start)
HS7=("Example", ac7, se1)

PRESS(HS7)
  LM("Maske3")
END_PRESS

//END

//M(Maske3/"Example 2: showing graphic"/"example.png")
HS1=("")
HS2=("")
HS3=("")
HS4=("")
HS5=("")
HS6=("")
HS7=("")
```

```
HS8=("")
VS1=("")
VS2=("")
VS3=("")
VS4=("\\sp_ok.png",,SE1)
VS5=(["\\sp_ok_small.png","OEM Text"],,SE1)
VS6=("")
VS7=(SOFTKEY_OK,,SE1)
VS8=(SOFTKEY_CANCEL,,SE1)
PRESS(VS4)
  EXIT
END_PRESS
PRESS(VS5)
  EXIT
END_PRESS
PRESS(VS7)
  EXIT
END_PRESS
PRESS(VS8)
  EXIT
END_PRESS
//END
```

4.2.2 和语言相关的文本

概述

使用和语言相关的文本：

- 软键标签
- 标题
- 辅助文本
- 其它任意文本

对话框中与语言相关的文本保存在文本文件中。

文本文件位于以下目录中：

- *[系统用户目录]*/lng
- *[系统 oem 目录]*/lng
- *[系统插件目录]*/lng

说明

文本文件必须如同项目文件一样进行保存。

例如：

[系统用户目录]/lng/[文本文件]

[系统用户目录]/proj/[配置文件]

alsc.txt　　用于西门子标准循环、和语言相关的文本

almc.txt　　用于西门子测量循环、和语言相关的文本

程序运行中所使用的文本文件在 easyscreen.ini 文件中给定：

```
[LANGUAGEFILES]
LngFile03 = user.txt      ; ->user<_xxx>.txt（例如：user_eng.txt）
```

文件 user.txt 在这里是作为文本文件的示例。原则上可自由选择名称。根据文件中文本的不同语种，还必须按照下面的句法附加上各语言的缩写。在名称后加上一个下划线以及相应的语种缩写，例如 `user_eng.txt`。

说明

由于 LngFile01 和 LngFile02 已经指定给标准 easyscreen.ini，因而不可用于用户文本。

另见

文件名中的语种缩写表 (页 264)

屏幕专用语言文件

只能在屏幕上使用确定的语言文件，以避免语言文件中所用的编号带重叠。

因此，在屏幕定义行中使用逗号隔开所用的语言文件。最后位置上的文件具有最高的优先级（与 easyscreen.ini 中的逻辑类似）。

所有在屏幕中使用的语言相关的文本提前都在语言文件中查找过了。如果没有找到文本，则接着在 easyscreen.ini 配置的文本文件中查找。

如果屏幕中没有定义语言文件，则只需在 easyscreen.ini 指定的语言文件中查找。

说明

如果当前所选的语言中没有语言文件，则使用英文语言文件（缺省）。

示例：

```
//M(MyMask/$85597///// //"mytexts.txt, general.txt, mask.txt")
DEF ….
```

该操作也能用于登入软键：

```
//S(Start, "mytexts.txt, general.txt, mask.txt")
VS1=($85597)
PRESS(VS1)
   LM("MyMask","masks.com")
END_PRESS
```

文本文件的格式

文本文件必须保存为 UTF-8 编码格式。

说明

保存配置文件和语言文件时请注意，应将您所使用的编辑器中的编码设为 UTF 8。

文本输入形式

句法：	8xxxx 0 0 “文本”		
说明：	文件中文本号和文本的排列		
参数：	xxxx	85000 到 89999	预留于用户的文本识别号范围。号码必须是唯一的。
	“文本”		显示在对话框中的文本
	%n		文本中用于分行的控制符

两个通过空格分开的参数 2 和 3 是用于报警文本输出的控制符。鉴于与报警文本的格式一致性，在任何情况下控制符都必须置零。

和语言相关的文本示例：

85000 0 0 “退回平面”

85001 0 0 “钻削深度”

85002 0 0 “螺距”

85003 0 0 “凹槽半径”

4.3 设计在线帮助

在线帮助

可以 HTML 格式创建对配置的对话框和单元的在线帮助。

创建在线帮助的句法和步骤与 SINUMERIK Operate 相似。

如需为输入栏设计在线帮助，则会使用在线帮助的标准屏幕，以显示针对用户的在线帮助：

```
DEF RFP=(R//1/,"RFP","RFP"///////"sinumerik_md_1.html","9006")
```

说明

LINUX 系统要求 HTML 文件名称必须以小写字母保存！

HTML 文件保存在以下目录中（例如：德语）。

[系统用户目录]/hlp/deu

[系统 oem 目录]/hlp/deu

[系统插件目录]/hlp/deu

其他语言的目录必须根据相应的语种缩写（参见附录）保存。

文档

调试手册“基本软件和操作软件” (IM9)，章节“针对 OEM 的在线帮助”

变量 5

5.1 定义变量

变量值

变量的重要属性即变量值。

可以通过如下方式赋值：

- 定义变量时预设
- 分配系统或者用户变量
- 采取方法

编程

句法：	名称 **val** = 变量值 名称 = 变量值	
说明：	变量值 val (value)	
参数：	名称：	变量名称
	变量值	变量值
示例：	`VAR3 = VAR4 + SIN(VAR5)` `VAR3.VAL = VAR4 + SIN(VAR5)`	

变量状态

通过变量状态的属性可以在运行时查询，变量是否包含有一个有效值。 该属性可通过值 FALSE = 0 读写。

编程

句法：	名称.**vld**
说明：	变量状态 vld (validation)

参数：	名称：	变量名称
	FALSE = TRUE =	查询结果可能是： 无效值 有效值
示例：	IF VAR1.VLD == FALSE VAR1 = 84 ENDIF	

5.2 应用举例

辅助变量

辅助变量是内部计算变量。 计算变量如同变量一样定义，但是另外除了变量值和状态之外没有属性，即辅助变量在对话框中不可见。 辅助变量是 VARIANT 类型。

编程

句法：	DEF *名称*	
说明：	VARIANT 类型的内部计算变量	
参数：	名称：	辅助变量名称
示例：	DEF OTTO；定义一个辅助变量	

句法：	名称 val = *辅助变量值* 名称 = *辅助变量值*	
说明：	辅助变量值在方式中指定。	
参数：	名称：	辅助变量名称
	辅助变量值：	辅助变量内容

示例：

```
LOAD
  OTTO = "Test"              ; 为辅助变量"Test" 赋值 Otto
END_LOAD
LOAD
  OTTO = REG[9].VAL          ; 为辅助变量 Otto 赋寄存器的值
END_LOAD
```

通过变量计算

在每次退出输入/输出栏（通过 ENTER 或者转换键）后计算变量。 计算在 CHANGE 方式中设计并在每次更改值时运算。

通过变量状态可以查询，变量是否包含有效值，例如：

```
IF VAR1.VLD == FALSE
   VAR1 = 84
ENDIF
```

系统变量间接编译地址

系统变量也可以间接编译地址，即和另一个变量相关。

```
PRESS(HS1)
  ACHSE=ACHSE+1
  WEG.VAR="$AA_DTBW["<<ACHSE<<"]"        ; 通过变量编译轴地址
END_PRESS
```

更改软键标签

示例：

```
HS3.st = “新文本”                        ; 更改软键标签
```

5.3 示例 1： 分配变量类型、文本、帮助画面、颜色、提示框

示例 1a

下面的示例中会定义一个变量，并对它的变量类型、文本、辅助画面和颜色属性进行设置。

DEF Var1 = (R///,"实际值",,"mm"//"Var1.png"////8,2)			
	变量类型：		REAL
	文本：		
		短文本：	实际值
		单位文本：	mm
	帮助图形：		Var1.png
	颜色：		
		前景色：	8（棕色）
		背景色：	2（橙色）

示例 1b

下面的示例中会定义一个变量，并对它的变量类型、预设值、文本、提示框、输入模式和短文本位置属性进行设置。

DEF Var2 = (I//5/"","值","","","," 提示框文本"/wr2///20,250,50)			
	变量类型：		INTEGER
	预设值：		5
	文本：		
		短文本：	值（可能的语言文本 ID）
		提示框：	提示框文本
	属性：		
		输入模式	读和写
	短文本位置：		
		到左边缘的间距	20
		到上边缘的间距	250
		宽度：	50

参见

变量参数 (页 79)

5.4 举例 2： 定义变量类型，极限值，属性，短文本位置

示例 2

下面的示例中会定义一个变量，并对它的变量类型、极限值、输入模式、对齐和位置属性进行设置。

DEF Var2 = (I/0,10///wr1,al1/// , ,300)			
	变量类型：		INTEGER
	极限值或者转换栏输入项：		MIN： 0 MAX： 10
	属性：		
		输入模式	只读
		短文本的文本对齐	右对齐
	短文本位置：		
		宽度：	300

参见

变量参数 (页 79)

5.5 举例 3： 定义变量类型、预设、系统或者用户变量、输入/输出栏位置

示例 3

下面的示例中会定义一个变量，并对它的变量类型、预设值、系统变量或用户变量和位置属性进行设置。

DEF Var3 = (R//10////"$R[1]"//300,10,200//)			
	变量类型：		REAL
	预设值：		10
	系统变量或者用户变量：		$R[1] （R 参数 1）
	短文本位置：		相对于输入/输出栏的标准位置
	输入/输出栏位置：		
		到左边缘的间距	300
		到上边缘的间距	10
		宽度：	200

参见

变量参数 (页 79)

5.6 示例 4： 转换栏和列表栏

示例 4a

转换栏中不同的条目：

```
DEF Var1 = (I/* 0,1,2,3)                    ；用于数值转换的简单转换栏
DEF Var2 = (S/* "On", "Off")                ；用于字符串转换的简单转换栏
DEF Var3 = (B/* 1="On", 0="Off")            ；用于数值转换的扩展转换栏，为每个数值分配了一个显示文本
DEF Var4 = (R/* ARR1)                       ；转换栏中要转换的值基于一个数组
```

示例 4b

列表栏与转换栏的配置一样，但要额外设置列表栏的显示类型（变量属性 DT = 4）。

DEF VAR_LISTBOX_Text = (S/*$80000,$80001,$80002,$80003,$80004/$80001// DT4////200,,340,60)		
变量类型：		字符串
极限值/转换栏：		* $80000,$80001,$80002,$80003,$80004 （要显示的与语言相关的文本列表）
预设值：		$80001
属性：		
	显示类型：	4（列表栏）
输入/输出栏位置：		
	左边距：	200
	宽度：	340
	高度：	60
颜色：		
	前景色：	6（蓝色）
	背景色：	10（白色）

5.7 示例 5： 图片显示

示例 5

显示图片，而不是短文本： 图片的大小和位置在“输入/输出栏位置（左侧、上部、宽度、高度）”中规定。

DEF VAR6 = (V///,"\\bild1.png" ////160,40,50,50)		
变量类型：		VARIANT
文本：		
	短文本：	bild1.png
输入输出栏的位置：		
	左边距：	160
	上边距：	40
	宽度：	50
	高度：	50

5.8 示例 6： 进度条

进度条是一种特殊的输入/输出栏显示类型并且只用于显示而不能输入。

原则上有两种进度条类型：

1. 最多可进行两次变色的进度条，例如温度或负载显示（见示例 6a）。
2. Operate 风格的进度显示（不变色）条（见示例 6a）

示例 6a

两次变色的进度条：

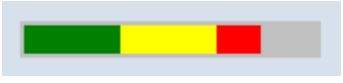


图 5-1 两次变色的进度条

DEF PROGGY0 = (R/0,150,50,100///DT1,DO0//"$R[10]"//,,150/3,4,,,,,,,9,7)		
变量类型：		REAL
极限值/转换栏：		
	MIN：	0
	MAX：	150
	信号值 SVAL1：	50
	信号值 SVAL2：	100
属性：		
	显示模式 DT：	1（进度条）
	显示选项 DO：	0（从左往右（默认））
系统变量或者用户变量：		$R[10]
输入/输出栏位置：		
	宽度：	150
颜色：		
	前景色：	3（深绿）
	背景色：	4（浅灰）
	信号色 SC1：	9（黄色）
	信号色 SC2：	7（红色）

如要使用可变色的进度条应将显示模式 DT (DisplayType)设为 1。

进度显示的方向通过属性“显示选项 DO (DisplayOption)”来确定：

0:从左往右（默认）

1:从右往左

2:从下往上

3:从上往下

进行进度条的显示时，应给定最小值和最大值（示例中：MIN：0，MAX：150）。

在此设置下，取决于变量 PROGGY0 的当前值，进度条将显示前景色 3（= 深绿）和背景色 4（= 浅灰）。

另外还可以选择定义一或两个信号值 SVAL1 和 SVAL2（参数限值）（示例中：SVAL1: 50 以及 SVAL2: 100）。达到这两个信号值时，进度显示的前景色会变化。相应的信号色通过参数 SC1 和 SC2 确定（上例中 SC1: 9（= 黄色）以及 SC2: 7（= 红色））。

进度显示的限值给定适用如下规则：

MIN < SVAL1 < SVAL2 < MAX。

示例 6b

不变色进度条：

图 5-2 进度条

DEF PROGGY0 = (R/0,150///DT2,DO0//"$R[10]"//,,150/6,10)		
变量类型：		REAL
极限值/转换栏：		
	MIN：	0
	MAX：	150
属性：		
	显示类型 DT：	2（进度条）
	显示选项 DO：	0（从左往右（默认））
系统变量或者用户变量：		$R[10]
输入/输出栏位置：		
	宽度：	150
颜色：		
	前景色：	6（蓝色）
	背景色：	10（白色）

如要使用不变色的进度条应将显示模式 DT (DisplayType)设为 2。

进度显示的方向通过属性“显示选项 DO (DisplayOption)”来确定（参见示例 6a 的说明）：

进行进度条的显示时，应给定最小值和最大值（示例中：MIN：0，MAX：150）。

在此设置下，取决于变量 PROGGY0 的当前值，进度条将显示前景色 6（= 蓝色）和背景色 10（= 白色）。

5.9 示例 7： 口令输入模式（星号）

示例 7

如要对输入栏中的输入内容实现隐藏，例如在输入口令时，必须将显示模式 DT 设为 5。所输入的字符会显示为星号。

图 5-3 口令输入模式（星号）

DEF VAR_SET_PWD=(S//""//DT5)			
	变量类型：		STRING
	预设值：		空字符串
	属性：		
		显示模式 DT：	5（口令输入模式）

5.10 变量参数

参数 - 一览

在以下概述中简要说明了变量参数。详细的说明请参见后续章节。

参数	说明	
变量类型 (页 86)	必须规定变量类型。	
	R[x]:	REAL（小数点位数为正数 +）
	I:	INTEGER
	S[x]:	字符串（字符串长度为正数 +）
	C:	字符（单字符）
	B：	BOOL
	V:	VARIANT
极限值 (页 72)	最小极限值，最大极限值 预设置：空 极限值通过逗号隔开。类型 I、C 和 R 的极限值可以使用十进制的格式或者字符 "A"、 "F"定义。	
	对于可变色的进度条显示（变量属性 DT = 1），可选择配置两个信号色 SC1 和 SC2 (Signal Color)，在超出信号值 SVAL1 或 SVAL2 (Signal Value)时会使用相应的进度条前景色。信号值为整型值。 参见进度条 (页 75)应用示例。	
预设值 (页 91)	如果没有定义任何预设值并且没有分配系统或者用户变量给变量，则分配转换栏的第一个单元。如果没有定义转换栏，则不进行预设，即：变量处于状态“未计算”。 预设置：未预设	
转换栏 (页 90)	输入/输出栏中带有预设输入项的列表：列表通过 * 开始，各输入项用逗号隔开。可以赋值输入项。 极限值的输入项视为转换栏的列表。如果只输入一个 *，则建立一个可变的转换栏。 预设置：无	

参数	说明	
文本 (页 70)	顺序已预先规定。也可以显示一个图形代替短文本。 预设置：空	
	长文本：	显示行的文本
	短文本：	对话框单元的名称
	图形文本：	文本参考图形名称。
	单位文本：	对话框单元的单位
	提示框	在设计窗口时，用于为各个显示栏和切换栏提供简短的说明信息。该信息通过纯文本和语言文本 ID 设计。

参数	说明	
属性 (页 72)	属性影响下列特性： ● 显示模式 ● 显示选项 ● 刷新率 ● 转换符号 ● 提示框 ● 输入模式 ● 访问等级 ● 短文本的文本对齐 ● 字体大小 ● 极限值 属性通过逗号隔开，顺序任意每个组件都可以进行定义。	
	显示模式	dt0：标准（输入/输出栏或转换栏）（默认） dt1：可变色的进度条： dt2：Operate 风格的不变色进度条 dt4：列表栏 dt5：口令输入模式（星号）
	显示选项	特别是对于显示模式 dt1 和 dt2（进度条），还需要同时设置显示选项。 do0：从左往右（默认） do1：从右往左 do2：从下往上 do3：从上往下
	刷新速度	使用属性 UR (update rate)可以控制显示的刷新以及相应配置的变量或 Grid 列的 CHANGE 块的处理。根据配置的不同，CPU 负载可能被大幅降低并且可达到更短的用户接口响应时间。 ur0：SlCap::standardUpdateRate()（当前 = 200 ms，默认值） ur1：50 ms ur2：100 ms ur3：200 ms ur4：500 ms ur5：1000 ms

参数	说明	
		ur6：2000 ms ur7：5000 ms ur8：10000 ms
	转换符号	tg0：转换符号关闭（默认） tg1：转换符号打开 如果将属性 TG 设为 1，则在输入栏提示框中还会额外显示转换符号。 示例： `DEF OFFS = (R//123.456/,,,,"My ToolTip"/TG1)`

	提示框	提示框文本在运行期间可通过变量属性 “TT” 来更改。
		示例：
		`PRESS(VS1)`
		`MyVar.TT = "My new ToolTip"`
		`END_PRESS`
		或者
		`DEF MyVar=(R///,,,,"My ToolTip-text")`

	输入模式	wr0:输入/输出栏不可见，短文本可见 wr1:读取（没有输入中心） wr2:读取和写入（行以白色显示） wr3:wr1 带输入中心 wr4:所有变量单元不可见，没有输入中心 wr5:按下任何键立即保存输入的值（和 wr2 相反 - 该模式下，在退出栏或者按下返回键后才开始保存值）。 预设置：wr2
	访问等级 (页 262)	空：总是可以写入 ac0...ac7:保护等级 如果访问等级未达到，行显示为灰色，标准设置：ac7
	短文本的文本对齐	al0:左对齐 al1:右对齐 al2:居中 预设置：al0
	字体大小	fs1:标准字体大小（8 Pt）

参数	说明	
		fs2:双倍字体大小 预设置：fs1 确定行间距。标准字体大小为每对话框 16 行。图形和单位文本只能为标准字体大小。
	极限值	通过极限值可以检查变量值是否在规定的最小极限值和最大极限值之内。 预设置：取决于规定的极限值 li0:没有检查 li1:检查最小极限值 li2:检查最大极限值 li3:检查最大极限值和最小极限值
	打开时的属性	定义变量时规定的 cb 属性，优先于定义对话框时的总定义。多个属性通过逗号隔开记录。（参见对话框打开方式（属性 CB） (页 265)）
	CB0： CB1：	如果变量当前有一个有效值（例如：通过预设值或 NC/PLC 变量），则通过显示屏幕触发 CHANGE 方法。 通过显示屏幕无法显式触发 CHANGE 方法。如果变量有一个配置的 NC/PLC 变量，则一定会调用 CHANGE 方法。
帮助画面 (页 70)	帮助画面文件：	png 文件的名称 预设置：空
	帮助画面文件的名称在双引号中。如果光标移至该变量，则自动显示画面（替代目前的图形）。	
系统或者用户变量 (页 73)	可以将一个来自 NC/PLC 的系统数据或用户数据指定给变量。系统或者用户变量用双引号括起。 文档：参数手册 系统变量，/PGAsl/	
短文本的位置 (页 92)	短文本位置（和左边缘/上边缘的间距、宽度） 位置数据的单位为像素，并且以对话框主体的左上角为基准。数据总是用逗号隔开。	
输入/输出栏的位置 (页 92)	输入/输出栏位置（和左边缘/上边缘的间距、宽度、高度） 位置数据的单位为像素，并且以对话框主体的左上角为基准。数据总是用逗号隔开。如果改变此位置，则短文本、图形文本和单位文本的位置也同时改变。	

参数	说明
颜色 (页 70)	输入/输出栏、短文本、图形文本、单位文本的前景色和背景色及进度条的信号色。 颜色通过逗号隔开。 颜色数据参见章节 颜色表 (页 262)。 输入/输出栏的缺省设置：前景色：黑色，背景色：白色 输入/输出栏的标准颜色取决于写入模式 “wr”： 短文本、图形文本、单位文本的缺省设置：前景色：黑色；背景色：透明色
	对于可变色的进度条显示（变量属性 DT = 1），可选择配置两个信号色 SC1 和 SC2 (Signal Color)，在超出信号值 SVAL1 或 SVAL2 (Signal Value)时会使用相应的进度条前景色。 参见进度条 (页 75)应用示例。
	变量定义中要求的颜色顺序如下： 1. 输入/输出栏的前景色：FC 2. 输入/输出栏的背景色：BC 3. 短文本的前景色：FC_ST 4. 短文本的背景色：BC_ST 5. 图形文本的前景色：FC_GT 6. 图形文本的背景色：BC_GT 7. 单位文本的前景色：FC_UT 8. 单位文本的背景色：BC_UT 9. 信号色 1 10 信号色 2 .

参数	说明
在线帮助文件 (页 65)	在线帮助文件的名称在双引号中。

集成了单位选择框的输入/输出栏	使用集成了单位选择的输入/输出栏可在不同的单位之间进行切换。如果将光标置于输入栏中，集成的单位选择框会被高亮显示（焦点无需显式设置在其上）。此外会显示一个带有转换符号的提示框，其中为该功能的相应说明信息。
	示例：
	`DEF VarEdit=(R/////////200,,100///"VarTgl")`
	`VarTgl=(S/*0="mm",1="inch"/0//WR2////302,,40)`
	或者
	`DEF VarEdit_2={TYP="R", VAL=1.234, X=200, W=100, LINK_TGL="vartgl_2"},`
	`VARTgl_2={TYP="S", TGL="* 0=""mm"", 1=""inch""",WR=2, X=302, W=40}`

变量：更改属性

以格式*名称.属性 = 值* 重新赋值。系统会分析等号右边的表达式并为变量赋值或者分配变量属性。

名称.**ac** = 存取级	(ac:access level)
名称.**al** = 文本对齐	(al:alignment)
名称.**bc** = 输入/输出栏的背景色	(bc:back color)
名称 **bc_gt** = 图形文本的背景色	(bc:back color) (gt:graphic text)
名称.**bc_st** = 短文本的背景色	(bc:back color) (st:short text)
名称.**bc_ut** = 单位文本的背景色	(bc:back color) (ut:unit text)
名称 **do** = 显示选项	(do:display option)
名称.**dt** = 显示模式	(dt:display type)
名称 **.fc** = 输入/输出栏的前景色	(fc:front color)
名称.**fc_gt** = 图形文本的前景色	(fc:front color) (gt:graphic text)

名称.**fc_st** = 短文本的前景色	(fc:front color) (st:short text)
名称.**fc_ut** = 单位文本的前景色	(fc:front color) (ut:unit text)
名称.**al** = 字体大小	(fs:font size)
名称.**gt** = 图形文本	(gt:graphic text)
名称.**hlp** = 帮助图形	(hlp:help)
名称.**li** = 极限值	(li:limit)
名称.**lt** = 长文本	(lt:long text)
名称.**max** = 最大极限值	(max:maximum)
名称.**min** = 最小极限值	(min:minimum)
名称.**sc** = 信号色	(sc:signal color)
名称.**st** = 简要说明	(st:short text)
名称.**tg** = 转换符号	(tg:toggle)
名称.**tt** = 提示框	(tt:提示框
名称.**typ** = 变量类型	(typ:type)
名称.**ur** = 刷新速度	(ur:update rate)
名称.**ut** = 单位文本	(ut:unit text)
名称 **val** = 变量值	(val:value)
名称.**var** = 系统或者用户变量	(var:variable)
名称.**vld** = 变量状态	(vld:validation)
名称.**wr** = 输入模式	(wr:write)

另见

扩展配置句法 (页 38)

5.11 各个变量类型的详细说明

INTEGER 型变量

“ INTEGER”整型变量可以使用下列扩展符号显示在输入/输出栏中并保存在存储器中。

扩展数据类型中的**第 2 个字符**

表示格式	
B	二进制
D	十进制，有符号
H	十六进制
没有数据	十进制，有符号

扩展数据类型中的**第 3 个和第 4 个字符**

存储器保存	
B	字节
W	字
D	双字
BU	字节，无符号
WU	字，无符号
DU	双字，无符号

INTEGER 型数据的字符顺序

1. “I” 主标识符，表示 INTEGER（整型）
2. 表示格式
3. 存储器保存
4. “U”，无符号

有效的整型定义：	
IB	整数变量 32 位二进制表示
IBD	整数变量 32 位二进制表示
IBW	整数变量 16 位二进制表示
IBB	整数变量 8 位二进制表示
I	整数变量 32 位十进制，有符号
IDD	整数变量 32 位十进制，有符号
IDW	整数变量 16 位十进制，有符号
IDB	整数变量 8 位十进制，有符号

有效的整型定义：	
IDDU	整数变量 32 位 十进制，无符号
IDWU	整数变量 16 位 十进制，无符号
IDBU	整数变量 8 位 十进制，无符号
IH	整数变量 32 位十六进制
IHDU	整数变量 32 位十六进制
IHWU	整数变量 16 位十六进制
IHBU	整数变量 8 位十六进制

VARIANT 型变量

VARIANT 型变量通过最后赋值的数据类型确定。 如果分配或输入的值是以 '-'、'+'、'.' 或数字 ('0'-'9') 开头的，该值则被编译为数值，其它情况下为字符串。

它可以通过功能 ISNUM 或者 ISSTR 查询。 VARIANT 型主要用于向 NC 码写入变量名或者数值。

编程

可以检查变量的数据类型：

句法：	**ISNUM** *(VAR)*	
参数：	VAR	待检查其数据类型的变量名称。
	FALSE = TRUE =	查询结果可能是： 没有数值（数据类型 = STRING） 数值（数据类型 = REAL）

句法：	**ISSTR** *(VAR)*	
参数：	VAR	待检查其数据类型的变量名称。
	FALSE = TRUE =	查询结果可能是： 数值（数据类型 = REAL） 没有数值（数据类型 = STRING）
示例：	IF ISNUM(VAR1) == TRUE IF ISSTR(REG[4]+2) == TRUE	

可以改变变量的显示模式：

- 对于 INTEGER 型变量可以改变显示方式。

B	二进制
D	十进制，有符号
H	十六进制
不带正负号	
另加一个 U，表示 unsigned（无符号）	

- 对于 REAL 型变量，只能更改小数点的位数。
  不允许更改基本类型，若更改，则导致文件 easyscreen_log.txt 出现错误报告。
  示例：

```
Var1.typ = "IBW"
Var2.typ = "R3"
```

数字格式

数字可以以二进制、十进制、十六进制或者指数方式描述。

二进制	B01110110
十进制	123.45
十六进制	HF1A9
指数	-1.23EX-3
示例：	
	VAR1 = HF1A9
	REG[0]= B01110110
	DEF VAR7 = (R//-1.23EX-3)

说明

在通过功能 GC 生成代码时计算值仅考虑十进制或者指数描述方式，而不使用二进制和十六进制描述方式。

参见

变量参数 (页 79)

5.12 转换栏的详细说明

说明

通过转换栏扩展可以显示与 NC/PLC 变量有关的文本（在转换栏中的输入项）。只能读取使用转换栏扩展的变量。按下 INSERT 按键，打开转换栏列表。

编程

句法：	**DEF VAR1=(IB/+ $85000/15////"DB90.DBB5")** 或者 **DEF VAR_TGL = (S/* "Hello", "Run", "MyScreens"/"Run")**	
说明：	显示对话框时，在输入/输出栏中显示文本号码 $85015 的内容。在系统变量 DB90.DBB5 输入预设值 15。如果系统变量 DB90.DBB5 中的值改变，在每次改变时显示的文本号码重新生成 $（85000 + <DB90.DBB5>）。	
参数：	变量类型	系统或者用户变量中指定的变量类型
	文本号码	与语言相关的文本号码（基本），该号码作为基本号码使用
	系统或者用户变量	通过最终的文本号码（基本＋补偿）形成的系统或者用户变量（补偿）。

图形与转换栏有关

转换栏用替换的图像覆盖。如果标记字节值为 1，则显示 “bild1.png”；如果标记字节值为 2，则显示 “bild2.png”。

```
DEF VAR1=(IDB/*1="\\bild1.png", 2="\\bild2.png"//,$85000/wr1//"MB[130]"//160,40,50,50)
```

图形的大小和位置在“输入/输出栏位置（左侧、上部、宽度、高度）”中规定。

虚拟转换键

未配置列表的转换栏，例如 `DEF NoTglList=(R/*)`，不带有任何列表。下个元素在按下转换键或运行完相应变量的相应 CHANGE() 方法后才会生成。

此时，当操作触摸屏时，一个虚拟的小键盘会显示在可变转换栏的右侧，该转换栏中只有转换键。

通过配置文件 slguiconfig.ini 中的以下条目也可以强制取消虚拟转换键的显示与触摸屏的关联。

```
[VirtualKeyboard]
; Forces easyscreen virtual toggle keyboard function
ForceEasyscreenVirtualToggleKey = true
```

参见

变量参数 (页 79)

5.13 预设值的详细说明

概述

根据变量栏（输入/输出栏或者转换栏）是否分配了一个预设值，一个系统或者用户变量或者两者都分配，得到不同的变量状态。 只有当变量分配了一个有效值时，转换才可行。

预设值生效

条件			栏类型反应
栏类型	预设值	系统或者用户变量	
输入/输出栏	是	是	在系统或者用户变量中写预设值
	否	是	使用系统或者用户变量作为预设值
	错误	是	未计算，系统或者用户变量未描述/未使用
	是	否	预设值
	否	否	未计算
	错误	否	未计算
	是	错误	未计算
	否	错误	未计算
	错误	错误	未计算

条件			栏类型反应
栏类型	预设值	系统或者用户变量	
转换	是	是	在系统或者用户变量中写预设值
	否	是	使用系统或者用户变量作为预设值
	错误	是	未计算， 系统或者用户变量未描述/未使用
	是	否	预设值
	否	否	预设值 =转换栏的第一个单元
	错误	否	未计算
	是	错误	未计算
	否	错误	未计算
	错误	错误	未计算

参见

变量参数 (页 79)

5.14 短文本位置、输入输出栏位置的详细说明

概述

短文本和图形文本以及输入/输出栏和单位文本总是形成一个单元。即短文本的位置数据也对图形文本和输入/输出栏数据以及单位文本上的数据有效。

编程

设计的位置数据覆盖标准值，即也仅能改变一个单独的值。如果下列对话框单元没有设计位置数据，则采用上一对话框单元的数据。

如果对话框单元没有规定位置，则使用预设置。短文本和输入/输出栏的栏宽度在标准情况下各行由栏数和最大栏宽度确定，即：栏宽度=最大行宽/列数。

图像和单位文本宽度是固定的，根据编程支持的请求优化。如果已设计图像和单位文本宽度，则短文本或者输入/输出栏的宽度相应缩短。

短文本和输入/输出栏的顺序可以通过位置数据互换。

输入/输出栏和单位栏之间的间距和单位栏宽度

可以配置输入/输出栏和单位栏之间的间距和单位栏宽度。

在定义行中输入输入/输出位置章节中用逗号隔开的输入/输出栏与单位栏之间的间距（例如 7 像素）和/或单位栏的宽度（例如 60 像素）：

```
DEF VarDT=(R3//0.000/,"DT",,"s"////0,,24/39,,71,,7,60)
```

或者：

```
DEF VarDT={TYP="R3", VAL="0.000", ST="DT", UT="s", TXT_X=0,
TXT_W=24, X=39, W=71, UT_DX=7, UT_W=60}
```

在配置了输入/输出栏/单位栏之间的间距和/或单位栏宽度时，需要注意以下几点：

- 配置的输入/输出区**不**包含单位栏的固定宽度（固定 50 像素）。即直接配置输入/输出栏宽度。
- 如果没有配置单位栏宽度，则宽度默认为 50 像素。
- 如果没有配置输入/输出栏/单位栏之间的间距，则间距默认为 0 像素。

组合转换栏的特性

前提条件：

- 一个变量配置了一个组合转换栏，
- 变量的输入/输出栏和单位栏之间的间距和/或单位栏宽度已经配置且
- 变量无单位文本。

该情况下，组合转换栏定位在变量单位栏的配置位置处。

忽略可能存在的、为组合转换栏的输入/输出部分配置的位置。

示例

以下示例中组合转换栏 **F_Unit** 自动以与变量 **VarF** 的输入/输出栏 **7 像素**的间距定位并设置宽度 **59 像素**。

```
DEF VarF=(R//0.0/,"F",,,////0,,24/39,,85,,7,59///"F_Unit"),
F_Unit = (I/*3="mm/min", 1="mm/U"/3// ////181,,155)
```

参见

变量参数 (页 79)

5.15 使用字符串

字符串链

设计时也可以使用字符串，以动态配置文本显示或者合并代码生成的不同文本。

规则

使用字符串变量时注意以下规定：

- 链接由左向右处理。
- 层叠的表达式由内向外运算。
- 忽略大小写。
- 字符串变量通常左对齐显示。

字符串可以通过一个简单的空字符串指令删除。

字符串在等号右边以运算符 "<<"开始。 字符串中双引号（"）通过两个连续的双引号标记。
字符串可以在 IF 指令中检查相等性。

示例

下列示例预设：

```
VAR1.VAL =“这是一个”
VAR8.VAL = 4
VAR14.VAL = 15
VAR2.VAL = “错误”
$85001 = “这是一个”
$85002 = “报警文本”
```

编辑字符串：

- 合并字符串：
  ```
  VAR12.VAL = VAR1 << “错误”   ;结果： “这是一个错误”
  ```
- 删除一个变量：
  ```
  VAR10.VAL = ""   ;结果： 空字符串
  ```
- 设定一个带有文本变量的变量：
  ```
  VAR11.VAL = VAR1.VAL   ;结果： “这是一个”
  ```
- 数据类型匹配：
  ```
  VAR13.VAL ="这是 " << (VAR14 - VAR8) << ". 错误"
              ;结果： “这是第 11 个错误”
  ```
- 处理数字值：
  ```
  VAR13.VAL = “错误” << VAR14.VAL << ": " << $85001 << $85002
              ;结果： “错误 15： “这是一个报警文本”
  IF VAR15 == “错误”  ; IF 指令中的字符串
    VAR16 = 18.1234
              ;结果： VAR16 等于 18.1234，
       ; 当 VAR15 等于“错误”时
  ENDIF
  ```
- 字符串中的双引号：
  ```
  VAR2=“你好，这是一个“测试”“
              ;结果： 你好，这是一个“测试”
  ```
- 和变量内容有关的系统或者用户变量的字符串：
  ```
  VAR2.Var = "$R[" << VAR8 << "]"   ; 结果： $R[4]
  ```

另见

字符串函数 (页 171)

5.16 变量 CURPOS

说明

通过变量 CURPOS 可以在当前对话框的激活的输入栏中调出或者操纵光标位置。 变量显示光标前有多少个字符。 如果光标位于输入栏开始处，则 CURPOS 接受值为 0。 如果更改 CURPOS 值，则光标停留在输入栏中相应的位置上。

为了可以在变量值更改情况下反应，可以借助于一个 CHANGE 方法监控改变情况。 如果 CURPOS 值改变，则跳转 CHANGE 方法并执行包含的指令。

5.17 变量 CURVER

说明

属性 CURVER（当前版本） 允许匹配编程用于处理不同的版本。 变量 CURVER 仅可读。

说明

在代码生成时自动以最新的版本生成，即使之前已用老的版本反编译。 命令 "GC" 总是生成最新的版本。 在生成代码中，在使用注释当版本 > 0 时插入一个生成的代码的附加标记。

规则

总是显示带有所有变量的最新的对话框。

- 以前的变量不允许改变。
- 新的变量以任意顺序插入在以前（循环）的编程中。
- 不允许从对话框由一个版本到下一个版本去除变量。
- 对话框必须包含所有版本的变量。

举例

```
(IF CURVER==1 ...)                    ; CURVER 在反编译时自动通过反编译代码的版本设置。
```

5.18 变量 ENTRY

说明

通过变量 ENTRY 可以检查如何调用对话框。

编程

句法：	**ENTRY**	
说明：	变量 ENTRY 仅可读。	
返回值：		查询结果可能是：
	0 =	没有编程支持
	1 =	通过软键启动屏幕；还未生成任何代码（预设值，同配置）
	2 =	通过软键启动屏幕；已生成代码 （预设值来自最后由该屏幕生成的代码）
	3 =	重新编译，带使用注释（#行）
	4 =	Code_typ = 0：带使用注释的 NC 代码（#行）
	5 =	Code_typ = 1：无使用注释的 NC 代码（#行）

示例

```
IF ENTRY == 0
  DLGL（“在编程下不调用对话框”）
ELSE
  DLGL（“在编程下调用对话框”）
ENDIF
```

5.19 变量 ERR

说明

通过变量 ERR 可以检查是否已正确执行先前行。

编程

句法：	**ERR**
说明：	变量 ERR 仅可读。

返回值：		查询结果可能是：
	FALSE =	已正确执行先前行
	TRUE =	未正确执行先前行

示例

```
VAR4 = 螺纹[VAR1,"KDM",3]                ；作为数组给出值
IF ERR == TRUE                           ；查询是否在数组中找到值
VAR5 = “数组存取出错”
                                         ；如果未在数组中找到值，则赋值“数组存取出错”给变量。
ELSE
VAR5 = “一切正常”                        ；如果在数组中找到值，则赋值“一切正常”给变量。
ENDIF
```

5.20 变量 FILE_ERR

说明

用变量 FILE_ERR 可以检查前面的 GC 或者 CP 指令是否正确执行。

编程

句法：	**FILE_ERR**
说明：	变量 FILE_ERR 仅可读。

返回值：		可能的结果是：
	0 =	排列中的操作
	1 =	驱动器/路径不存在
	2 =	路径/文件存取故障
	3 =	驱动器未就绪
	4 =	错误的文件名
	5 =	文件已经打开
	6 =	存取失败
	7 =	目标路径不存在或不允许
	8 =	复制源符合目标
	10 =	内部错误： FILE_ERR = 10 时，与一个未分类到其他类别故障有关。

示例

```
CP("D:\source.mpf","E:\target.mpf")
                                        ; 从 source.mpf 向 E:\target.mpf 复制
IF FILE_ERR > 0                         ; 询问是否出现故障
IF FILE_ERR == 1                        ; 询问特定的错误号并输出所属的故障文本
VAR5 = "驱动器/路径不存在"
ELSE
IF FILE_ERR == 2
VAR5 = "路径一/文件存取故障"
ELSE
IF FILE_ERR == 3
VAR5 = "错误文件名"
ENDIF
ENDIF
ENDIF
ELSE
VAR5 = "一切正常"                       ; 如果 CP (或者 GC) 中没有出现错误
                                        , 则输出"一切正常"
ENDIF
```

5.21 变量 FOC

说明

使用变量 FOC 可以控制对话框内的输入中心（当前有效的输入/输出栏）。 已预先固定定义光标左右键、前后键以及 PGUP、PGDN 的反应。

说明

FOC 不允许通过一个导航事件触发。 光标位置只允许在软键 PRESS 方法、CHANGE 方法、...中改变。

带有输入模式 wr = 0 和 wr = 4 的变量以及辅助变量无法实现聚焦。

编程

句法：	**FOC**	
说明：	可以读取和写入该变量。	
返回值：	读取	聚焦变量的名称作为结果输出。
	写入	可以写入一个字符串或者数值。 字符串视为变量名称，数值视为变量索引。

示例

```
IF FOC == "Var1"                ; 读取输入中心
  REG[1] = Var1
ELSE
  REG[1] = Var2
ENDIF

FOC = "Var1"                    ; 分配变量 1 给输入中心。
FOC = 3                         ; 分配第 3 对话框单元， WR ≥ 2 给输入中心。
```

参见

FOCUS (页 114)

5.22 变量 S_ALEVEL

说明

在配置中会通过屏幕属性 S_ALEVEL 询问当前访问级。

编程

句法：	**S_ALEVEL**
说明：	查询当前访问级
返回值：	0: 系统
	1: 制造商
	2: 维修
	3: 用户
	4: 钥匙开关位置 3
	5: 钥匙开关位置 2
	6: 钥匙开关位置 1
	7: 钥匙开关位置 0

示例

```
REG[0] = S_ALEVEL
```

参见

ACCESSLEVEL (页 111)

5.23 变量 S_CHAN

说明

通过变量 S_CHAN 可以确定用于显示或者某个评估的当前通道号码。

编程

句法： **S_CHAN**
说明： 查询当前通道号
返回值： 通道号

示例

```
REG[0] = S_CHAN
```

参见

CHANNEL (页 113)

5.24 变量 S_CONTROL

说明

在配置中会通过屏幕属性 S_CONTROL 询问当前控制系统名称。

编程

句法： **S_CONTROL**
说明： 查询当前控制系统名称
返回值： 控制系统名称为 mmc.ini 中的段落名。

示例

```
REG[0] = S_CONTROL
```

参见

CONTROL (页 113)

5.25 变量 S_LANG

说明

在配置中会通过屏幕属性 S_LANG 询问当前语言。

编程

句法：	**S_LANG**
说明：	查询当前语言
返回值：	resources.xml 中的语言缩写，例如：“deu”、“eng” 等

示例

```
REG[0] = S_LANG
```

参见

LANGUAGE (页 115)

5.26 变量 S_NCCODEREADONLY

说明

只有在编辑器中通过对话框 “Run MyScreens” 反编译循环时，屏幕属性 S_NCCODEREADONLY 才有意义。通过 S_NCCODEREADONLY 确定是否可以修改编辑器中的反编译 NC 代码。

返回值适用条件：

- TRUE：可以修改编辑器中的反编译 NC 代码
- FALSE：由于编辑器中的反编译 NC 代码已经处于预处理状态 (= TRUE)，因此，无法进行修改（只读）。

5.27 变量 S_RESX 和 S_RESY

说明

在配置中会通过屏幕属性 S_RESX 和 S_RESY 询问当前分辨率或其 X 或 Y 分量。

示例

```
REG[0] = S_RESY
```

参见

RESOLUTION (页 120)

编程指令

6

6.1 运算符

概述

在编程时可以使用以下几种运算符：

- 数学运算符
- 比较运算符
- 逻辑（布尔）运算符
- 位运算符
- 三角函数功能

6.1.1 数学运算符

概述

数学运算符	名称
+	加法
-	减法
*	乘法
/	除法
MOD	模数运算
( )	括号
AND	和运算符
OR	或运算符
NOT	非运算符
ROUND	带有小数位的数四舍五入

示例： `VAR1.VAL = 45 * (4 + 3)`

ROUND

在执行对话框设计时，用 ROUND 指令把数值四舍五入直至小数点后面的 12 位。小数点后位置在变量栏不能显示。

使用

ROUND 通过用户的两个参数来控制:

```
VAR1 = 5,2328543

VAR2 = ROUND( VAR1, 4 )
```

结果： `VAR2 = 5.2339`

VAR1 包含在四舍五入的数中。参数"4"给出了结果中的小数点后的位数，保存在 VAR2 中。

三角函数功能

三角函数功能	名称
SIN(x)	正弦 x
COS(x)	余弦 x
TAN(x)	正切 x
ATAN(x, y)	反正切 x/y
SQRT(x)	平方根 x
ABS(x)	绝对值 x
SDEG(x)	换算为度数
SRAD(x)	换算为弧度
CALC_ASIN(x)	反正弦 x
CALC_ACOS(x)	反余弦 x

说明

这些函数处理弧度。为了换算，可以使用函数 SDEG() 和 SRAD()。

示例： `VAR1.VAL = SQRT(2)`

数学函数

数学函数	名称
CALC_CEIL(x)	计算下一个比 x 大的整数（向上取整）
CALC_FLOOR(x)	计算下一个比 x 小的整数（向下取整）
CALC_LOG(x)	计算以 e 为底的 x 的（自然）对数
CALC_LOG10(x)	计算以 10 为底的 x 的对数
CALC_POW(x, y)	计算 x 的 y 次方（x 的 y 次乘方）
CALC_MIN(x, y)	计算 x 与 y 哪个小
CALC_MAX(x, y)	计算 x 与 y 哪个大

随机功能：随机数

句法：	**RANDOM**（*下限值，上限值*）	
说明：	**RANDOM** 功能为提供预先给定范围内的伪随机数。	
参数：	下限值	下限值 >= -32767, 下限值 < 上限值
	上限值	上限值 <= 32767

示例

```
REG[0] = RANDOM(-10,10)          ; 可能结果 = -3
```

常量

常量	
PI	*3.14159265358979323846*
FALSE	*0*
TRUE	*1*

示例： `VAR1.VAL = PI`

比较运算符

比较运算符	
==	相等
<>	不等
>	大于
<	小于
>=	大于等于
<=	小于等于

示例：

```
IF VAR1.VAL == 1
  VAR2.VAL = TRUE
ENDIF
```

条件

层叠深度没有限制。

带有一个命令的条件：

```
IF
...
ENDIF
```

带有两个命令的条件：

```
IF
...
ELSE
...
ENDIF
```

6.1.2 位运算符

概述

位运算符	名称
BOR	位方式 OR
BXOR	位方式 XOR
BAND	位方式 AND
BNOT	位方式 NOT
SHL	位向左移动
SHR	位向右移动

运算符 SHL

通过运算符 SHL(SHIFT LEFT)向左移动位。 此时要移动的值或者移动步数可以直接规定或者作为变量规定。 如果达到数据格式极限，则已超出位，没有出错信息。

使用

句法：	变量 = *值* **SHL** *步数*	
说明：	向左移动	
参数：	值	要移动的值
	步数	移动步数

示例

```
PRESS(VS1)
VAR01 = 16 SHL 2              ; 结果 = 64
VAR02 = VAR02 SHL VAR04       ; VAR02 内容转换为 32 位格式，无符号
                              ;并且位向左移动 VAR04 内容。 接着
                               32 位值重新转换回
                              变量 VAR02 的格式。
END_PRESS
```

运算符 SHR

通过运算符 SHR (SHIFT RIGHT)向右移动位。 此时要移动的值或者移动步数可以直接规定或者作为变量规定。 如果达到数据格式极限，则已超出位，没有出错信息。

使用

句法：	变量 = *值* **SHR** *步数*	
说明：	向右移动	
参数：	值	要移动的值
	步数	移动步数

示例

```
PRESS(VS1)
VAR01 = 16 SHR 2                    ; 结果 = 4
VAR02 = VAR02 SHR VAR04             ; VAR02 内容转换为 32 位格式，无符号
                                    ;并且位向右移动 VAR04 内容。 接着
                                     32 位值重新转换回
                                    变量 VAR02 的格式。

END_PRESS
```

6.2 方法

概述

在对话框和与对话框相关的软键栏中（软键栏由新设计的对话框调用），可以通过不同的事件（退出输入栏，按下软键）触发某些特定的动作。 这些动作设计在方法中。

方法的基本编程按如下方式进行：

说明块	注释	参考章节
PRESS(HS1)	; 方法的开始标识	
LM... LS...	; 功能	参见章节“功能”

说明块	注释	参考章节
`Var1.st = ...`	；改变属性	参见章节“软键栏” 和章节“对话框单元”

`Var2 = Var3 + Var4`
`...`
`EXIT`
	；通过变量计算	参见章节“定义变量”

`END_PRESS`	；方法的结束标识	

6.2.1 ACCESSLEVEL

说明

当在屏幕打开时更改当前访问等级时，就会运行 ACCESSLEVEL 方法。

编程

句法：	ACCESSLEVEL <指令> END_ACCESSLEVEL
说明：	访问等级
参数：	- 无 -

参见

变量 S_ALEVEL (页 101)

6.2.2 CHANGE

说明

当变量值已改变时运行 CHANGE（改变）方法。即在 CHANGE（改变）方法中设计变量改变时立即运行的变量计算。

单元特定的 CHANGE 方法和全局的 CHANGE 方法有所不同：

- 当特定变量值已改变时运行**单元特定的 CHANGE 方法**。 如果系统或者用户变量已分配一个变量，则可以在 CHANGE 方法中循环更新变量值。
- 当改变任意一个变量值且没有设计单元特定的 CHANGE 方法时，运行**全局 CHANGE 方法**。

编程“单元特定”

句法：	CHANGE*(名称)* ... END_CHANGE	
说明：	修改指定变量的值	
参数：	名称	变量名称

示例

```
DEF VAR1=(I//////"DB20.DBB1")          ; Var1 分配一个系统变量
CHANGE(VAR1)
IF VAR1.Val <> 1
  VAR1.st="工具正确!"                  ; 如果系统变量的值 ≠ 1，则变量的简要说明为： 工具正确!
    otto=1
ELSE
    VAR1.st="注意 错误! "              ; 如果系统变量的值 = 1，则变量的简要说明为： 注意 错误!
    otto=2
ENDIF
  VAR2.Var=2
END_CHANGE
```

编程“全局特定”

句法：	CHANGE() ... END_CHANGE
说明：	改变任意变量值
参数：	- 无 -

示例

```
CHANGE()
EXIT                                    ；如果任何一个变量值改变，则退出对话框。
END_CHANGE
```

参见

对话框打开方式（属性 CB） (页 265)

6.2.3 CHANNEL

说明

当在屏幕打开时更改当前通道时，即通道切换时，就会运行 CHANNEL 方法。

编程

句法：	CHANNEL <指令> END_CHANNEL
说明：	通道切换
参数：	- 无 -

参见

变量 S_CHAN (页 101)

6.2.4 CONTROL

说明

当在屏幕打开时更改当前控制系统时，即通常在 1:n 切换时，就会运行 CONTROL 方法。

编程

句法：	CONTROL <指令> END_CONTROL
说明：	系统切换
参数：	- 无 -

参见

变量 S_CONTROL (页 102)

6.2.5 FOCUS

说明

当对话框中聚焦（光标）定位在另一个栏上时，运行 FOCUS 方法。

方法 FOCUS 不允许通过一个导航事件触发。 光标位置只允许在软键 PRESS 方法、CHANGE 方法、...中改变。 光标移动的反应预先固定定义。

说明

在 FOCUS 方法中不允许定位在另一个变量上并且也不允许装载新的对话框。

编程

句法：	FOCUS ... END_FOCUS
说明：	光标：定位
参数：	- 无 -

示例

```
FOCUS
DLGL("聚焦已设定在变量"<< FOC <<"上。")
END_FOCUS
```

参见

变量 FOC (页 100)

6.2.6 LANGUAGE

说明

当在屏幕打开时更改当前语言时，就会运行 LANGUAGE 方法。

编程

句法：	LANGUAGE <指令> END_LANGUAGE
说明：	语言
参数：	- 无 -

参见

变量 S_LANG (页 103)

6.2.7 LOAD

说明

在已编译变量定义和软键定义（DEF Var1= ..., HS1= ...）后运行 LOAD 方法。 此时，对话框还未显示。

编程

句法：	LOAD ... END_LOAD
说明：	装载
参数：	- 无 -

示例

```
LOAD                          ；开始标记
 屏幕窗口 1.Hd =  $85111      ；对话框标题来自文本文件
  VAR1.Min = 0                ；指定变量的最小极限值
  VAR1.Max = 1000             ；指定变量的最大极限值
END_LOAD                      ；结束标记
```

参见

线、分割线、矩形、圆形和椭圆形 (页 183)

6.2.8 UNLOAD

说明

在卸载对话框之前，运行 UNLOAD 方法。

编程

句法：	UNLOAD ... END_UNLOAD
说明：	卸载
参数：	- 无 -

示例

```
UNLOAD
  REG[1] = VAR1             ；保存寄存器的变量
END_UNLOAD
```

6.2.9 OUTPUT

说明

当调用功能 "GC" 时，运行 OUTPUT 方法。 在 OUTPUT 方法中变量和辅助变量作为 NC 代码设计。 代码行各个单元的链接用一个空格符实现。

说明

NC 代码可以用文件功能在一个额外的文件中生成并移向 NC。

编程

句法：	OUTPUT*(名称)* ... END_OUTPUT	
说明：	输出 NC 程序中的变量	
参数：	名称	OUTPUT 方法的名称

程序段号码和隐藏标记

当要继续保留编程支持激活时在零件程序中直接设置的行号和反编译时的隐藏标记时，OUTPUT 方法不允许包含行号和隐藏标记。

在零件程序中通过编辑器更改影响下列行为：

条件	行为
程序段数目保持不变。	程序段号码继续保留。
程序段数目变小。	删除最大的程序段号码。
程序段数目变大。	新的程序段没有程序段号码。

示例

```
OUTPUT(CODE1)
  "CYCLE82(" Var1.val "," Var2.val "," Var3.val ","Var4.val "," Var5.val "," Var6.val ")"
END_OUTPUT
```

6.2.10 PRESS

说明

当已按下相应的软键时，运行 PRESS 方法。

编程

句法：	PRESS*(软键)* ... END_PRESS		
名称：	按下软键		
参数：	软键	软键名称： HS1 - HS8 和 VS1 - VS8	
	回调	按键<RECALL>（回调）	
	ENTER	按键 <ENTER>，参见 PRESS(ENTER) (页 119)	
	转换	按键 <TOGGLE>，参见 PRESS(TOGGLE) (页 120)	
	PU	Page Up	向前翻页
	PD	Page Down	向后翻页
	SL	Scroll Left	光标向左
	SR	Scroll Right	光标向右
	SU	Scroll Up	光标向上
	SD	Scroll Down	光标向下

示例

```
HS1 = ("其他软键栏")
```

```
HS2 = ("没有功能")
PRESS(HS1)
  LS("软键栏 1")                    ; 载入其他软键栏
  Var2 = Var3 + Var1
END_PRESS
PRESS (HS2)
END_PRESS
PRESS(PU)
  INDEX = INDEX -7
CALL("UP1")
END_PRESS
```

6.2.11 PRESS(ENTER)

说明

对于带有输入/输出栏且输入模式为 WR3 或 WR5 的变量，当按下回车(Enter)键时，就会调用 PRESS(ENTER)方法：

- WR3： 定位到栏的位置并按下回车键
- WR5： 在输入模式下，使用回车键接收值

编程

句法：	PRESS(ENTER) <指令> END_PRESS
说明：	回车键已按下
参数：	- 无 -

6.2.12 PRESS(TOGGLE)

说明

与当前处于焦点下的变量无关，只要按下转换键就会调用 PRESS(TOGGLE)方法。

需要时，可借助屏幕属性 FOC 来检测哪个变量当前处于焦点下。

编程

句法：	PRESS(TOGGLE) <指令> END_PRESS
说明：	转换键已按下
参数：	- 无 -

示例

```
PRESS(TOGGLE)
   DLGL("Toggle key pressed at variable " << FOC)        ；借助屏幕属性 FOC 检测哪个变量当前处于焦点下
END_PRESS
```

6.2.13 RESOLUTION

说明

当在屏幕打开时更改当前分辨率时，即通常在 TCU 切换时，就会运行 RESOLUTION 方法。

编程

句法：	RESOLUTION <指令> END_RESOLUTION
说明：	分辨率
参数：	- 无 -

参见

变量 S_RESX 和 S_RESY (页 104)

6.2.14 RESUME

说明

当屏幕（例如）被区域切换中断并要重新显示时，就会调用 RESUME 方法。 此时会重新处理数值变化，必要时启动计时器并执行 RESUME 块。

编程

句法：	RESUME <指令> END_RESUME
说明：	屏幕重新激活
参数：	- 无 -

6.2.15 SUSPEND

说明

当中断屏幕且不卸载时，就会调用 SUSPEND 方法。 例如，当因为简单的区域切换而离开了屏幕，但不显式卸载该屏幕时。 屏幕仍保留在后台，但此时不处理数值变化和可能存在的计时器。 屏幕暂时中止。

编程

句法：	SUSPEND <指令> END_SUSPEND
说明：	屏幕中断
参数：	- 无 -

示例

```
SUSPEND
```

```
   MyVar1 = MyVar1 + 1
END_SUSPEND
```

6.2.16 示例： 带 OUTPUT 方法的版本管理

概述

已有的对话框可以通过扩展补充附加的变量。 附加的变量在定义中变量名后的圆括号内有一个版本识别号： （0 = 原始，未写入），1 = 版本 1，
2 = 版本 2， ...

举例：

```
DEF var100=(R//1)                ; 原始，相当于版本 0
DEF var101(1)=(S//"Hallo")       ; 补充，从版本 1 起
```

在写入 OUTPUT 方法时可以参考某个版本状态，与定义的总体性有关。

举例：

```
OUTPUT(NC1)                      ; 在 OUTPUT 方法中仅提供原始变量。
OUTPUT(NC1,1)                    ; 在 OUTPUT 方法中提供原始变量和带有版本标识符 1 的补充变量。
```

原始版的 OUTPUT 方法不需要版本标识符，然而也可以记为 0。 OUTPUT(NC1) 相当于 OUTPUT(NC1,0)。 OUTPUT 方法中的版本标识符 n 包括所有变量，从原始 0、1、2、... 直至 n。

编程版本标识

```
//M(XXX)                         版本 0 (默认)
DEF var100=(R//1)
DEF var101=(S//"Hallo")
DEF TMP
VS8=("GC")
PRESS(VS8)
GC("NC1")
END_PRESS

OUTPUT(NC1)
```

```
var100",,"var101
END_OUTPUT

; ************ 版本 1，补充的定义 ***************
//M(XXX)
DEF var100=(R//1)
DEF var101=(S//"Hallo")
DEF var102(1)=(V//"HUGO")
DEF TMP
VS8=("GC")
PRESS(VS8)
GC("NC1")
END_PRESS
...

OUTPUT(NC1)                          ; 原始和其它新的版本
var100","var101
END_OUTPUT
...

OUTPUT(NC1,1)                        版本 1
var100","var101"," var102
END_OUTPUT
```

6.3 功能

概述

在对话框和与对话框相关的软键栏中提供不同的功能，这些功能通过事件（例如：退出输入栏，按下软键）触发并在方法中设计。

子程序

重复的或者其它的设计指令，这些指令总结为一个特定的过程，可以在子程序中设计。 子程序可以随时装载到主程序或者其它子程序中并随时进行编辑，即指令不必多次重复设计。 作为主程序适用于对话框描述块或者软键栏。

外部功能

通过外部功能可以引入其它一些针对用户的功能。 外部功能存放在一个 DLL 文件中并通过设计文件定义行中的条目识别。

PI 服务

通过功能 PI_START 可以在由 PLC 在 NC 区中启动 PI 服务（程序实例服务）。

另见

功能 (FCT) (页 145)

PI 服务 (页 160)

6.3.1 读写驱动参数： RDOP, WDOP, MRDOP

说明

使用功能 RDOP、WDOP 和 MRDOP 可以读写驱动参数。

说明

读取驱动参数时不要短于 1 秒的周期，时间长些更佳。
原因：否则与驱动的通讯可能会受到很大干扰，甚至导致故障。

说明

如果在读写驱动参数时出错，则会对屏幕属性 ERR 进行相应的设置。

编程

句法：	**RDOP**（*“驱动对象的名称”，“参数号”*）
说明：	读驱动参数(Drive Object Parameter)

参数：	驱动对象的名称	驱动对象的名称可参考“诊断”操作区（> ETC > HSK 8：驱动系统）“驱动系统诊断”中的“DO Name”一列（见下图）。
	参数号	

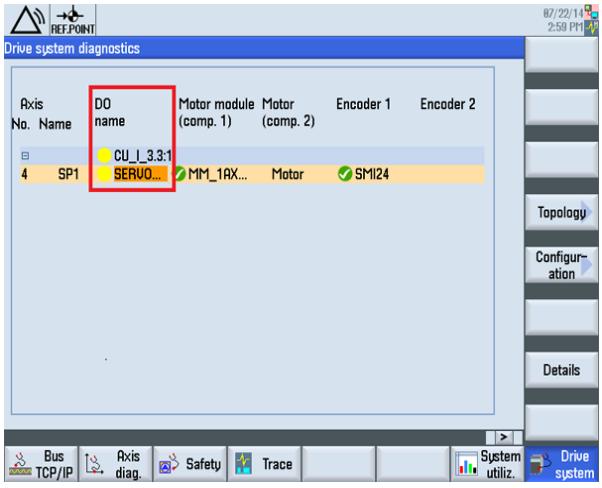


图 6-1　　驱动对象的名称

句法：	**MRDOP**（*“驱动对象的名称”，“参数号 1”*“参数号”[*...]，寄存器索引）*
说明：	多次读取驱动参数。 使用命令 MRDOP 可以通过一次存取向寄存器中传输一个驱动对象的多个驱动参数。这种存取比通过单个存取读取速度明显加快。

参数：	驱动对象的名称	驱动对象的名称可从（例如）“调试”操作区域的基本画面中获取
	参数号 1 ... n	变量名称用“*”作为分隔符。按照命令中变量名称的顺序将值接收到寄存器 REG[寄存器索引]中。
	寄存器索引	第一个变量的值在 REG[寄存器索引] 中。 第二个变量的值在 REG[寄存器索引 + 1] 中。

句法：	**WDOP**（*“驱动对象的名称”，“参数号”，“值”*）	
说明：	写驱动参数(Drive Object Parameter)	
参数：	驱动对象的名称	驱动对象的名称可从（例如）“调试”操作区域的基本画面中获取
	参数号	参数号
	值	待写入的值

示例

读取驱动对象“SERVO_3.3:2”的电机温度 r0035：

```
MyVar=RDOP("SERVO_3.3:2","35") ;
```

读取驱动对象“SERVO_3.3:2”的电机温度 r0035 和转矩实际值 r0080 并从寄存器索引 10 开始保存各个结果：

```
MRDOP("SERVO_3.3:2","35*80",10)
```

6.3.2 子程序调用(CALL)

说明

通过 CALL 功能可以从方法的任意一个位置调用一个装载的子程序。 允许叠加，即由一个子程序调用另一个子程序。

编程

句法：	**CALL***("名称")*	
说明：	调用子程序	
参数：	名称	子程序名称

示例

```
//M(屏幕窗口 1)
DEF VAR1 = ...
DEF VAR2 = ...
CHANGE(VAR1)
  ...
  CALL("MY_UP1")              ; 调用并编辑子程序
  ...
END_CHANGE

CHANGE(VAR2)
  ...
  CALL("MY_UP1")              ; 调用并编辑子程序
  ...
END_CHANGE

SUB(MY_UP1)
                              ;do something
END_SUB
//END
```

6.3.3 定义块(//B)

说明

子程序在程序文件中用块标记 //B 标记并通过 //END 结束。 每个块标记可以定义多个子程序。

说明

必须在调用子程序的对话框中定义子程序所使用的变量。

编程

一个块有下列结构：

句法：	**//B**(*块名称*) **SUB***(名称)* END_SUB [SUB(*名称*) ... END_SUB] ... //END	
说明：	定义子程序	
参数：	块名称	块标记名称
	名称	子程序名称

示例

```
//B(PROG1)                          ; 块开始
SUB(UP1)                            ; 子程序开始
  ...
REG[0] = 5                          ; 寄存器 0 赋值 5
  ...
END_SUB                             ; 子程序结束
SUB(UP2)                            ; 子程序开始
  IF VAR1.val=="Otto"
```

```
    VAR1.val="Hans"
RETURN
ENDIF
  VAR1.val="Otto"
END_SUB                          ; 子程序结束
//END                            ; 块结束
```

6.3.4 检查变量（CVAR）

说明

借助于功能 CVAR（检查变量）可以查询对话框的所有变量或者仅特定变量或者辅助变量是否正确。

在用功能 GC 产生一个 NC 代码之前，查询变量是否包含一个有效值是必要的。

如果变量状态命名符.vld = 1，则变量正确，

编程

句法：	**CVAR(*VarN*)**	
说明：	检查变量的有效内容	
参数：	VarN	要检查的变量列表。 可以最多有 29 个变量，各自之间通过逗号隔开。 最大字符长度为 500。 查询结果可能是：
	1 =	TRUE（真）（所有变量都有有效内容）
	0 =	FALSE（假）（至少一个变量没有有效内容）

示例

```
IF CVAR == TRUE                  ; 检查所有变量
  VS8.SE = 1                     ; 如果所有变量正确，则软键 VS8 可见
ELSE
  VS8.SE = 2                     ; 如果变量包含错误值，则软键 VS8 不可操作
ENDIF
```

```
IF CVAR("VAR1", "VAR2") == TRUE
                                                ; 检查变量 VAR1 和 VAR2
  DLGL ("VAR1 和 VAR2 正确")
                                                ; 如果 VAR1 和 VAR2 无错误，则显示对话框行"VAR1
                                                和 VAR2 正确"
ELSE
    DLGL ("VAR1 和 VAR2 不正确")
                                                ; 如果 VAR1 和 VAR2 错误，则显示对话框行"VAR1
                                                和 VAR2 不正确"
ENDIF
```

6.3.5 CLEAR_BACKGROUND

说明

使用功能 CLEAR_BACKGROUND 可以删除图形元素 LINE、RECT、ELLIPSE、V_SEPARATOR 和 H_SEPARATOR。

参见

线、分割线、矩形、圆形和椭圆形 (页 183)

6.3.6 文件功能"Copy Program (CP，复制程序)"

说明

功能 CP（复制程序）用于在 HMI 文件系统或者 NC 文件系统中复制文件。

编程

句法：	**CP***("源文件", "目标文件")*	
说明：	文件：复制	
参数：	源文件	源文件完整的路径数据
	目标文件	目标文件完整的路径数据

可通过返回值（VAR1 被定义为辅助变量）查询功能是否成功执行：

```
CP("//NC/MPF.DIR/HOHO.MPF","//NC/MPF.DIR/ASLAN.MPF",VAR1)
```

示例

带返回值的应用情况：

```
CP("//NC/MPF.DIR/HOHO.MPF","//NC/MPF.DIR/ASLAN.MPF",VAR1)
CP("CF_CARD:/wks.dir/MESS_BILD.WPD/MESS_BILD.MPF","//NC/WKS.DIR/AAA.WPD/HOHO2.MPF",VAR1)
CP("//NC/MPF.DIR/HOHO.MPF","CF_CARD:/wks.dir/WST1.WPD/MESS.MPF",VAR1)        ; WPD 必须存在
```

无返回值的应用情况：

```
CP("//NC/MPF.DIR/HOHO.MPF","//NC/MPF.DIR/ASLAN.MPF")
CP("CF_CARD:/mpf.dir/MYPROG.MPF","//NC/MPF.DIR/HOHO.MPF")
CP("//NC/MPF.DIR/HOHO.MPF","CF_CARD:/XYZ/MYPROG.MPF")                          ; XYZ 必须存在
```

另见

FILE_ERR 的支持： 变量 FILE_ERR (页 98)

6.3.7 文件功能“Delete Program (DP，删除程序)”

说明

功能 DP（删除程序）删除一个被动的 HMI 文件系统或者主动的 NC 文件系统的文件。

编程

句法：	**DP***("文件")*	
说明：	文件：删除	
参数：	文件	要删除文件的完整路径数据

示例

此功能使用以下句法进行数据管理：

- 带返回值

  ```
  DP("//NC/MPF.DIR/MYPROG.MPF",VAR1)
  DP("//NC/WKS.DIR/TEST.WPD/MYPROG.MPF",VAR1)
  ```

```
DP("//NC/CMA.DIR/MYPROG.SPF",VAR1)
```

VAR1 = 0　　文件已删除。

VAR1 = 1　　文件尚未删除。

- 无返回值：

```
DP("//NC/MPF.DIR/MYPROG.MPF")
DP("//NC/WKS.DIR/TEST.WPD/MYPROG.MPF")
DP("//NC/CMA.DIR/MYPROG.SPF")
```

6.3.8 文件功能“Exist Program (EP，退出程序)”

说明

功能 EP（存在程序）检查 NC 文件系统的特定 NC 程序或者 HMI 文件系统中在规定的路径下面是否存在某个文件。

编程

句法：	**EP**(*"文件"*)	
说明：	检查 NC 程序的存在	
参数：	文件	NC 文件系统或者 HMI 文件系统内文件的完整路径数据
返回值：	待分配查询结果的变量的名称。	

功能 EP 使用新的句法和旧的逻辑（调整句法）。

使用符合要求的名称直接响应文件：

```
//NC/MPF.DIR/XYZ.MPF
```

或

```
CF_CARD: /MPF.DIR/XYZ.MPF
```
（指示 /user/sinumerik/data/prog）

或

```
LOC:
```
（即 CF_CARD）

新句法示例：

```
EP("//NC/WKS.DIR/TEST.WPD/XYZ.MPF",VAR1)
EP("CF_CARD:/mpf.dir/XYZ.MPF",VAR1)
EP("LOC:/mpf.dir/XYZ.MPF",VAR1)
; 带返回值：
; VAR1 = 0 文件存在。
; VAR1 = 1 文件不存在。
```

旧句法示例：

```
EP("/MPF.DIR/CFI.MPF", VAR1)
; 带返回值：
; VAR1 = M 文件位于 HMI 文件系统中。
; VAR1 = N 文件位于 NC 文件系统中。
; VAR1 = B 文件位于 HMI 和 NC 文件系统中。
```

示例

```
EP("/MPF.DIR/GROB.MPF",VAR1)                                    ; 旧句法 - 路径现在采用 /，而不是 \

IF VAR1 == "M"
   DLGL（“文件位于 HMI 文件系统中”）
ELSE
   IF VAR1 == "N"
      DLGL（“文件位于 NC 文件目录中”）
   ELSE
     DLGL（“文件既不在 HMI 文件系统中，也不在 NC 文件系统中”）
   ENDIF
ENDIF
```

6.3.9 文件功能“Move Program (MP，移动程序)”

说明

功能 MP（移动程序）用于在 HMI 文件系统或者 NC 文件系统中复制文件。

编程

句法：	**MP***("源", "目标")*	
	MP("CF_CARD:/MPF.DIR/MYPROG.MPF","//NC/MPF.DIR")	
说明：	移动文件	
参数：	源文件	完整路径
	目标文件	完整路径
返回值	查询结果	

示例

带返回值：

```
MP("//NC/MPF.DIR/123.MPF","//NC/MPF.DIR/ASLAN.MPF",VAR1)          ; 位于 NC 内部
MP("//NC/MPF.DIR/123.MPF",VAR0,VAR1)                              ; 通过变量设定目标
MP(VAR4,VAR0,VAR1)                                                ; 通过变量设定源和目标
MP("CF_CARD:/mpf.dir/myprog.mpf","//NC/MPF.DIR/123.MPF",VAR1)     ; 从 CF 卡至 NC
MP("//NC/MPF.DIR/HOHO.MPF","CF_CARD:/XYZ/123.mpf",VAR1)           ; 从 NC 至 CF 卡

; 带返回值：
; 执行 VAR1 = 0
; 未执行 VAR1 = 1
```

6.3.10 文件功能“Select Program (SP，选择程序)”

说明

功能 SP（选择程序）选择一个主动的 NC 文件系统文件，以对其进行处理。即该文件之前必须就已装载在 NC 中。

编程

句法：	**SP***("文件")*	
名称：	选择程序	
参数：	“文件”	NC 文件的完整路径数据

示例

此功能使用以下句法进行数据管理：

- 带返回值

  ```
  SP("//NC/MPF.DIR/MYPROG.MPF", VAR1)
  ```

  VAR1 = 0　　文件已装载。

  VAR1 = 1　　文件未装载。
- 无返回值：

  ```
  SP("//NC/MPF.DIR/MYPROG.MPF")
  ```

6.3.11　文件存取： RDFILE、WRFILE、RDLINEFILE、WRLINEFILE

说明

采用 INI 句法对文件进行读写访问时可使用功能 RDFILE 和 WRFILE。

对文件中的各行进行读写访问时可使用功能 RDLINEFILE 和 WRLINEFILE。

编程

句法：	**RDFILE**(*“文件名 + 路径”，“段”，“关键字”*)	
说明：	从文件中读	
参数：	文件名 + 路径	路径和文件名
	段	INI 文件中的段落
	关键字	INI 文件中的关键字

句法：	**WRFILE***(数值，“文件名 + 路径”，“段”，“关键字”)*	
说明：	写入文件	
参数：	Value	待写入的值
	文件名 + 路径	路径和文件名
	段	INI 文件中的段落
	关键字	INI 文件中的关键字

句法：	**RDFILE***(“文件名 + 路径”， 行编号)*	
说明：	从文件中读取一行	
参数：	文件名 + 路径	路径和文件名
	行号	行号 第一行从 0 开始编号。

句法：	**WRLINEFILE***(数值， “文件名 + 路径”)*	
说明：	在文件末尾写一行	
参数：	值	待写入的值
	文件名 + 路径	路径和文件名

说明

- 文件不得位于 NC 文件系统（数据维护）中。
- 如果文件不存在或者到达了文件末尾又或者是出现了其他错误，都会相应地设置变量 FILE_ERR 和 ERR。 您可事先使用文件功能 Exist Program (EP)检查一个文件是否存在。
- 处理过的文件会以“UTF-8”编码（无 BOM（Byte Order Mask））。 读取的文件要求以“UTF-8”编码。
- 通过文件功能 Delete Program (DP)可在需要时将文件显式删除。

示例

从 INI 文件中读：

前提条件/假设：

文件 C:/tmp/myfile.ini 的内容：

<…>

```
[MyData]

MyName=Daniel
```

<…>

```
MyVar = RDFILE("C:/tmp/myfile.ini", "MyData", "MyName")
```

结果：

MyVar 现在包含值 “Daniel”。

写入 INI 文件：

前提条件/假设：

VARS=12

```
WRFILE(VARS, "C:/tmp/myfile.ini", "MySession", "NrOfSessions")
```

结果：

文件 C:/tmp/myfile.ini 的内容：

<…>

```
[MySession]
NrOfSessions=12
```

<…>

读取文件的第 4 行：

前提条件/假设：

c:/tmp/myfile.mpf:
R[0]=0 的内容

```
F3500 G1
MYLOOPIDX1: X0 y-50 Z0
  X150 Y50 Z10
  R[0]=R[0]+1
GOTOB MYLOOPIDX1
M30
```

```
MyVar = RDLINEFILE("C:/tmp/myfile.mpf", 4)
```

结果：

MyVar 现在包含值 “R[0]=R[0]+1”

在文件末尾写入：

前提条件/假设：

```
VARX=123
VARY=456

WRLINEFILE("F100 X" << VARX << " Y" << VARY, "C:/tmp/mypp.mpf")
```

结果：

c:/tmp/mypp.mpf 的内容：

<...>

```
F100 X123 Y456
```

6.3.12 对话框行(DLGL)

说明

在对话框的对话框行中可以根据确定的情况给出短文本（信息或者输入帮助）。

标准字体大小时允许的字符数量： 约 50 个字符

编程

句法：	**DLGL***("字符串"*)	
说明：	显示对话框行的文本	
参数：	字符串	显示在对话框中的文本

示例

```
IF Var1 > Var2
                                ；如果变量 1>变量 2，则对话框行中显示文本“值过大！”。
  DLGL("值太大！")
ENDIF
```

6.3.13 DEBUG

说明

通过功能 DEBUG 可在 Run MyScreen 用户屏幕的设计阶段为您提供分析帮助。 采用功能 DEBUG 可对括号中返回的运行时间的表述进行检测。 结果会在日志文件“easyscreen_log.txt”中作为独立条目加以记录。

在每条记录前会附加一个带有方括号的当前时间戳（参见下例）。

建议在对时间要求苛刻的部件上根据情况将 DEBUG 输出文件删除。 建议在设计阶段结束后为 DEBUG 输出文件加上注释。 对不断增多的日志文件 “easyscreen_log.txt” 的写访问可能会导致速度变慢。

编程

句法：	**DEBUG***(表述)*
说明：	在日志文件“easyscreen_log.txt”中进行记录
参数：	待检测的表述，由此在日志文件中生成一条记录

示例

```
IF Var1 > Var2
  DEBUG("Value of ""Var1"": " << Var1)
                                          ; “easyscreen_log_txt” 中的记录：
                                          [10:22:40.445] DEBUG: Value of "Var1":
                                          123.456
ENDIF
```

6.3.14 退出对话框(EXIT)

说明

通过功能 EXIT 可以退出对话框并返回主对话框。 如果不存在主对话框，则退出新配置的操作界面并返回标准应用程序。

编程（没有参数）

句法：	**EXIT**
说明：	退出对话框
参数：	- 无 -

示例

```
PRESS(HS1)
EXIT
END_PRESS
```

说明

如果调用带传输变量的当前对话框，则变量值改变并且返回初始对话框。

变量值分别分配给通过功能"LM"从初始对话框传输到后续对话框的变量。 可以最多传输 20 个变量值，各自之间通过逗号隔开。

说明

变量/变量值的顺序必须根据 LM 功能的传递变量顺序进行，以此达到分配明确。 如果一些变量值没有规定，则这些变量不改变。 改变的传输变量在功能 LM 后立即在初始对话框中生效。

使用传输变量编程

句法：	**EXIT***[(VARx)]*	
说明：	退出对话框，传输一个或者多个变量	
参数：	VARx	计算变量

示例

```
//M(屏幕窗口 1)
...
PRESS(HS1)
```

```
  LM("屏幕窗口 2","CFI.COM",1, POSX, POSY,
直径)
                                          ; 中断屏幕 1 并显示屏幕 2。 传输变量 POSX、POSY
                                          和直径。
DLGL("退出屏幕 2")                        ; 从屏幕 2 返回后在屏幕 1 的对话框行中显示文本:
                                          退出屏幕 2。
END_PRESS
...
//END

//M(屏幕 2)
...
PRESS(HS1)
  EXIT(5, , 计算_直径)
                                          ; 退出屏幕 2 并返回到屏幕 1 的 LM 后的行中。 此
                                          时，赋值 5 给变量 POSX 并分配变量“计算_直径”的
                                          值给变量“直径”。 变量 POSY 包含当前值。
END_PRESS
...
//END
```

6.3.15 转换栏或列表框栏的动态列表操作

说明

功能 LISTADDITEM、LISTINSERTITEM、LISTDELETEITEM 和 LISTCLEAR 用于对转换栏或列表框栏进行动态列表操作。

这些功能只对本身具有列表的变量有效，例如

- “基本”列表
  `DEF VAR_AC1 = (I/* 0,1,2,3,4,5,6,7,8)` 或
- “扩展”列表
  ```
  DEF VAR_AC2 = (I/* 0="AC0", 1="AC1", 2="AC2", 3="AC3", 4="AC4",
  5="AC5", 6="AC6", 7="AC7", 8="AC8").
  ```

当变量指向一个数组时，例如 `DEF VAR_AC3 = (I/* MYARRAY)`，该功能无法使用，否则全局数组会发生变化。

一个变量应至少在 DEF 行中定义了一个值。这样可确定为“基本”或“扩展”型列表。

此后仍允许将列表完全删除以及全部重建。但类型“基本”或“扩展”必须保留或无法动态更改。

编程

句法：	**LISTINSERTITEM***(变量名，位置，ItemValue[, ItemDispValue])*	
说明：	在某个特定位置添加元素	
参数：	变量名	
	位置	列表中需要添加元素的位置
	ItemValue	列表条目的值
	ItemDispValue	应在列表中显示的值

句法：	**LISTADDITEM**(变量名，ItemValue[, ItemDispValue])	
说明：	在列表末尾附加一个元素	
参数：	变量名	
	ItemValue	列表条目的值
	ItemDispValue	应在列表中显示的值

句法：	**LISTDELETEITEM**(变量名，位置)	
说明：	删除某个元素	
参数：	变量名	
	位置	列表中要删除元素的位置

句法：	**LISTCOUNT**（变量名）	
说明：	提供当前列表元素数	
参数：	变量名	

句法：	**LISTCLEAR***(变量名)*	
说明：	删除整个列表	
参数：	变量名	

示例

说明

以下几个示例彼此互为基础，它们的顺序对各个结果的理解起着重要作用。

前提条件/假设：

```
DEF VAR_AC = (I/* 0="Off",1="On"/1/,"Switch"/WR2)
```

在列表末尾附加一个元素 -1="Undefined"：

```
LISTADDITEM("VAR_AC", -1, """Undefined""")
```

结果：0="Off", 1="On", -1="Undefined"

在位置 2 上添加一个元素 99="Maybe"：

```
LISTINSERTITEM("VAR_AC", 2, 99, """Maybe""")
```

结果：0="Off", 1="On", 99="Maybe", -1="Undefined"

确定当前列表元素数：

```
REG[10]=LISTCOUNT("VAR_AC")
```

结果：REG[10] = 4

删除位置 1 上的元素：

```
LISTDELETEITEM("VAR_AC", 1)
```

结果：0="Off", 99="Maybe", -1="Undefined"

删除整个列表：

```
LISTCLEAR("VAR_AC")
```

结果：列表为空

6.3.16 评估(EVAL)

说明

功能 EVAL 评估作出的输出结果，然后执行。 为此可以首先在运行期间建立表达式。 例如用于变量上的显示存取。

编程

句法：	**EVAL***(exp)*	
说明：	评估表达式	
参数：	exp	逻辑表达式

示例

```
VAR1=(S)
VAR2=(S)
VAR3=(S)
VAR4=(S)
CHANGE()
  REG[7] = EVAL("VAR"<<REG[5])   ; 如果 REG[5] 的值为 3， 则括号中的表达式为 VAR3。 然后，分配 REG[7]  VAR3 的值。
  IF REG[5] == 1
  REG[7] = VAR1
ELSE
  IF REG[5] == 2
      REG[7] = VAR2
ELSE
  IF REG[5] == 3
        REG[7] = VAR3
ELSE
  IF REG[5] == 4
          REG[7] = VAR4
ENDIF
ENDIF
ENDIF
ENDIF
END_CHANGE
```

6.3.17 退出装载软键(EXITLS)

说明

通过功能 EXITLS 可以离开当前的操作界面并装载一个定义的软键栏。

编程

句法：	**EXITLS**(*"软键栏"[, "路径名"]*)	
说明：	退出时装载软键栏	
参数：	软键栏	待装载的软键栏名称
	路径名	要装载的软键栏目录路径

示例

```
PRESS(HS1)
    EXITLS( "软键栏 1", "AEDITOR.COM" )
END_PRESS
```

6.3.18 功能 (FCT)

说明

外部功能存放在一个 DLL 文件中，并通过配置文件定义行中的条目识别。

说明

一个外部功能必须至少有一个返回参数。

编程

句法：	**FCT** *功能名称 = ("文件"/返回类型/固定参数的类型/可变参数的类型）*
	FCT InitConnection = ("c:\tmp\xyz.dll"/I/R,I,S/I,S)
说明：	例如可从 LOAD 方法中或在 PRESS 方法中执行外部功能的调用。

参数：	功能名称	外部功能的名称
	文件	DLL 文件的完整路径数据
	返回类型	返回值的数据类型
	固定参数类型	值参数
	可变参数类型	参考参数
	数据类型用逗号隔开。	

例如可从 LOAD 方法中或在 PRESS 方法中执行外部功能的调用。

示例：

```
press(vs4)
RET = InitConnection(VAR1,13,"Servus",VAR2,VAR17)
end_press
```

外部功能的结构

外部功能必须遵循特定的符号：

句法：	extern "C" dllexport void InitConnection (ExtFctStructPtr FctRet, ExtFctStructPtr FctPar, char cNrFctPar)	
说明：	仅在 Windows 环境下执行 DLL 导出 区分符和传输参数为固定设定。 通过传输的结构传送原先的调用参数。	
参数：	cNrFctPar	调用参数的数量 = FctPar 中结构单元的数量
	FctPar	指示某结构单元栏，其包含相应的调用参数（含数据类型）。
	FctRet	指示某结构用于返回功能值（含数据类型）。

传输结构定义

```
union CFI_VARIANT
    (
    char                                b;
    short int                           i;
    double                              r;
```

```
    char*                                   s;
    )
typedef struct ExtFctStructTag
    (
    char                                    cTyp;
    union CFI_VARIANT                       value;
    )ExtFctStruct;
typedef struct ExtFct* ExtFctStructPtr;
```

如需独立于操作平台（Windows、Linux）开发外部功能，则不可使用关键字 __declspec(dllexport)。 只在 Windows 下需要此关键字。 例如在 Qt 下可使用以下宏：

```
#ifdef Q_WS_WIN
       #define MY_EXPORT __declspec(dllexport)
#else
       #define MY_EXPORT
#endif
```

函数声明如下：

```
extern "C" MY_EXPORT void InitConnection
(ExtFctStructPtr FctRet, ExtFctStructPtr FctPar,char cNrFctPar)
```

如需将通过 Run MyScreens 设计的画面用于 NCU 和 PCU/PC，则必须略去二进制文件的扩展名：

```
FCT InitConnection = ("xyz"/I/R,I,S/I,S)
```

在省略了绝对路径信息时，Run MyScreens 会首先在项目索引中搜索二进制文件。

6.3.19 生成代码(GC)

说明

功能 GC（Generate Code）从 OUTPUT 方法中生成 NC 代码。

编程

句法：	**GC**("*名称*"[,"*目标文件*"][,*Opt*],[*Append*])
说明：	生成 NC 代码

参数：	名称	OUTPUT 方法名称，作为代码生成的基础
	目标文件	HMI 或者 NC 文件系统的目标文件路径数据。 如果未规定目标文件（只能在编程支持中），则在该位置上写入代码，在该位置处光标停留在当前打开的文件中。
	可选	可选注释生成
	0:	（缺省设置）生成带有反编译注释的代码（另见 反编译 (页 165)）。
	1:	在生成的代码中不生成注释。 **说明：** 该代码不能反编译（另见 Auto-Hotspot）。
	Append	只有当规定目标文件情况下，该参数才有意义。
	0:	（预设）当文件已存在情况下，删除旧的内容。
	1:	当文件已存在情况下，在文件开始处写入新的代码。
	2:	在文件已存在的情况下，在结束处加入新的代码。

示例

```
//M(测试 GC/"代码生成:")
DEF VAR1 = (R//1)
DEF VAR2 = (R//2)
DEF D_NAME
LOAD
  VAR1 = 123
  VAR2 = -6
END_LOAD
OUTPUT(CODE1)
  "Cycle123(" VAR1 "," VAR2 ")"
  "M30"
END_OUTPUT

PRESS(VS1)
  D_NAME = "\MPF.DIR\MESSEN.MPF"
  GC("CODE1",D_NAME)                ; 将 OUTPUT 方法的 NC 代码写入文件 \MPF.DIR
                                    \MESSEN.MPF 中：
                                    Cycle123(123, -6)
                                    M30

END_PRESS
```

反编译

- **目标文件没有数据：**
  功能 GC 只能在编程支持中使用并待当前编辑器中打开的文件内写入 NC 代码。 NC 代码可以反编译。 如果功能 GC 在没有目标文件数据的情况下在“Run MyScreens”下设计，则此时发出故障信息。
- **目标文件数据：**
  从 OUTPUT 方法中生成的代码输入到目标文件中。 如果不存在目标文件，则在 NC 文件系统中设立。 如果目标文件在 HMI 文件系统中，则该文件存放在硬盘上。 不设立使用注释行（反编译需要的信息），即无法进行反编译。

指定目标文件的特殊性

基本上，指定目标文件有两种方式：

- **NC 格式：** `/_N_MPF_DIR/_N_MY_FILE_MPF`
  该文件存放在 NC 上的目录 MPF 下。
- **DOS 格式：** `d:\abc\my_file.txt` 或者 `\\RemoteRechner\files\my_file.txt`
  文件写入在硬盘上规定的目录中或者指定的计算机上，此时硬盘上的目录或者远程计算机必须已存在。

说明

无效的变量在生成的 NC 代码中产生一个空字符串，如果读取该代码，在日志中出现一条出错信息。

反编译时的特殊情况

在子对话框中无法调用 GC 功能，因为在子对话框中可以使用源自主对话框的变量，然而在直接调用时并不存在。

在通过编辑器手动介入生成的代码时不允许改变由代码生成产生的值的符号数目。 这些改变会妨碍到反编译。

解决方法：

1. 反编译
2. 通过设计的对话框输入修改（例如：99 → 101）
3. GC

6.3.20 口令功能

概述

提供以下可用的口令功能：

- 设置口令
- 删除口令
- 更改口令

HMI_LOGIN 功能：设置新口令

句法：	**HMI_LOGIN***(Passwd)*	
说明：	通过功能 HMI_LOGIN() 可发送一个口令到 NCK，通过该口令设置当前的存取等级。	
参数：	Passwd	口令

示例

```
REG[0] = HMI_LOGIN("CUSTOMER")          ；口令成功设置时，结果 = TRUE，否则为 FALSE
```

HMI_LOGOFF 功能：删除口令

句法：	**HMI_LOGOFF**
说明：	通过功能 HMI_LOGOFF 可复位当前的存取等级。
参数：	- 无 -

示例

```
REG[0] = HMI_LOGOFF                     ；口令成功删除时，结果 = TRUE，否则为 FALSE
```

HMI_SETPASSWD 功能：更改口令

句法：	**HMI_SETPASSWD***(AC, Passwd)*	
说明：	通过功能 HMI_SETPASSWD 可对当前口令级或更低口令级的口令进行更改。	
参数：	AC	访问等级
	Passwd	口令

示例

```
REG[0] = HMI_SETPASSWD(4,"MYPWD")      ; 口令成功更改时，结果 = TRUE，否则为 FALSE
```

6.3.21 装载数组(LA)

说明

通过功能 LA（装载数组）可以从另一个文件装载一个数组。

编程

句法：	**LA***(名称 [, 文件])*	
说明：	从文件装载数组	
参数：	名称	待装载数组的名称
	文件	在该文件中定义数组

说明

如果在当前的配置文件中，一个数组要由另一个配置文件中的数组替代，则数组名称必须相同。

示例

```
                                                ; 文件 maske.com 部分摘录
DEF VAR2 = (S/*ARR5/"关闭", "转换栏")
PRESS(HS5)
  LA("ARR5","arrayext.com")                     ; 从文件 arrayext.com 中
                                                装载数组 ARR5
  VAR2 = ARR5[0]                                ; 代替 "Aus"/"Ein"（"关"/"开"）显示在 VAR2
                                                的转换栏
                                                "上"/"下"/"右"/"左"
END_PRESS
//A(ARR5)
("关"/"开")
//END
                                                ; 文件 arrayext.com 部分摘录
//A(ARR5)
("上"/"下"/"右"/"左")
//END
```

说明

请注意，在通过 LA 功能分配给变量转换栏另一个数组后，必须赋予一个有效值给变量。

6.3.22 装载块（LB）

说明

通过功能 LB（装载块）可以在运行期间内装载带有子程序的块。 首先 LB 要在 LOAD（装载）方法中设计，由此可以随时调用装载的子程序。

说明

子程序也可以直接在对话框中定义，如果那样则不必装载。

编程

句法：	**LB***("块名称"[,"文件"])*	
说明：	运行时装载子程序	
参数：	块名称	块标记名称
	文件	配置文件的路径数据 预设 = 当前的配置文件

示例

```
LOAD
LB("PROG1")                  ; 在当前的配置文件中查找块“PROG1”，接着装载块。
  LB("PROG2","XY.COM")       ; 在配置文件 XY.COM 中查找块“PROG2”，接着装载块。
END_LOAD
```

6.3.23 装载屏幕窗口 (LM)

说明

通过功能 LM 可以载入一个新的对话框。 取决于对话框切换的模式（见下表），还可采用该功能实现信息对话框。

主对话框/子对话框

能够调用其他对话框并且不能自行结束的对话框称为主对话框。 由主对话框调用的对话框是子对话框。

编程

句法：	**LM***("名称"[,"文件"] [,MSx [, VARx]])*
说明：	载入对话框

参数：	名称	待载入对话框的名称
	文件	配置文件的路径数据（HMI 文件系统或者 NC 文件系统）；缺省设置： 当前的配置文件
	MSx	对话框切换模式
	0:	（缺省设置）退出当前对话框，装载并显示新的对话框。 在 EXIT 时（参见“退出”功能）返回到标准应用程序。 通过参数 MSx 可以确定在对话框切换时是否退出当前对话框。 保留当前的对话框，可以将变量传递到新的对话框中。 使用参数 MSx 的优点在于，在切换时无需总是重新初始化对话框，而是保当前对话框的数据和设计，简化数据传输
	1:	自功能 LM 起中断当前对话框，装载并显示新的对话框（例如用于实现信息对话框）。 退出时关闭子对话框并且返回到主对话框的中断位置。 中断时，不处理主对话框中的 UNLOAD（卸载）块。
	VARx	前提条件： MS1 列出可以从主对话框传送到子对话框的变量。 可以最多传输 20 个变量，各自之间通过逗号隔开。

说明

参数 VARx 总是只传递变量值，也就是说变量可以在子对话框中读写，但是不可见。 由子对话框向主对话框返回传递变量可以通过功能 EXIT（退出）进行。

示例

屏幕 1（主对话框）：跳至子对话框屏幕 2（带变量传输）

```
PRESS(HS1)

  LM("屏幕 2","CFI.COM",1, POSX, POSY, 直径)
                                    ; 中断屏幕 1 并显示屏幕 2： 传输变量 POSX、POSY 和直径。
DLGL(“退出屏幕 2”)                  ; 从屏幕 2 返回后在屏幕 1 的对话框行中显示文本： 退出屏幕 2。

END_PRESS
```

屏幕 2（子对话框）：返回主对话框屏幕 1（带变量传输）

```
PRESS(VS8)

  EXIT(POSX, POSY, DURCHMESSER)
                                        ；隐藏屏幕 2 并继续显示屏幕 1，传输修改的变量 POSX、
                                        POSY 和直径。
END_PRESS
```

6.3.24 装载软键(LS)

说明

通过功能 LS 可以显示另一个软键栏。

编程

句法：	**LS***("名称"[, "文件"][, 合并])*	
说明：	显示软键栏	
参数：	名称	软键栏名称
	文件	配置文件的路径数据（HMI 文件系统或者 NC 文件系统） 预设置： 当前的配置文件
	Merge	
	0:	删除所有存在的软键，输入重新设计的软键。
	1:	预设置 仅重新设计的软键覆盖现有的软键，其他软键（= HMI 应用程序的软键）的功能和文本保持不变。

示例

```
PRESS(HS4)
  LS("软键栏 2",,0)     ；软键栏 2 覆盖现有的软键栏，删除显示的软键。
END_PRESS
```

说明

只要编译器还没有显示出对话框，即还没有处理 LM 功能，则在登入软键描述块和软键栏描述块的 PRESS（按压）方法中总是仅设计一个 LS 命令或者 LM 命令，并且没有其它动作。

功能 LS 和 LM 仅允许在软键 PRESS 方法中调用，然而不作为导航键（PU、PD、SL、SR、SU、SD）上的反应。

6.3.25 Load Grid (LG)

说明

表格描述(Grid)可以通过 LG 方法在 LOAD 方法内动态提供。

为此可以通过 LG 方法指定一个表格，必须已将变量定义为 Grid 变量并引用已存在的有效表格。

编程

句法：	**LG**（*栅格名称，变量名称 [,文件名称]*）	
说明：	载入表格	
参数：	栅格名称	表格（栅格）名称，用双引号括起
	变量名	应指定表格的变量名称，用双引号括起
	文件名	文件名称，在该文件中定义表格（栅格），用双引号括起如果表格未在文件中定义，在该文件中也定义变量，则必须指定。

示例

```
//M(MyGridSample/"My Grid Sample")
DEF MyGridVar=(R/% MyGrid1///WR2////100,,351,100)
LOAD
   LG("MyGrid1","MyGridVar","mygrids.com")
END_LOAD
```

mygrids.com 的内容：

```
//G(MyGrid1/0/5)
(I///,"MyGrid1"/wr1//"1"/80/1)
(R3///"LongText1","R1-R4"/wr2//"$R[1]"/80/1)
(IBB///"LongText2","M2.2-M2.5"/wr2//"M2.2"/80/,1)
(R3///"LongText3","R9,R11,R13,R15"/wr2//"$R[9]"/110/2)
//END
```

6.3.26 多次选择 SWITCH

说明

使用 SWITCH 命令可以对变量的不同数值进行检验。

在 SWITCH 命令中设置的表达式会与在接下来的 CASE 命令中设置的所有数值进行比较。

如不相同，则会继续和之后的 CASE 命令进行比较。 如果之后有 DEFAULT 命令并且之前一直没有与在 SWITCH 命令中设置的表达式相同的 CASE 命令，则会执行 DEFAULT 命令之后的指令。

如果相同，则会执行接下来的指令，直到出现 CASE、DEFAULT 或 END_SWITCH 命令为止。

示例

```
FOCUS
   SWITCH (FOC)
      CASE "VarF"
         DLGL("Variable ""VarF"" has the input focus.")
      CASE "VarZ"
         DLGL("Variable ""VarZ"" has the input focus.")
      DEFAULT
         DLGL("Any other variable has the input focus.")
   END_SWITCH
END_FOCUS
```

6.3.27 多次读取 NC PLC (MRNP)

说明

用命令 MRNP 可以通过在寄存器中的一次存取输入多个 NC/PLC 变量。 这种存取比通过单个存取读取速度明显加快。 NC/PLC 变量必须在相同区域的 MRNP 命令中。

NC/PLC 变量区域分为如下几种：

- 一般 NC 数据 ($MN..., $SN.., /nck/...)
- 通道专用的 NC 数据 ($MC..., $SC.., /channel/...)
- PLC 数据 (DB..., MB.., /plc/...)
- 相同轴专用的 NC 数据 ($MA..., $SA..)

编程

句法：	**MRNP***(变量名称 1*变量名称 2[*...], 寄存器索引)*
说明：	读取多个变量
参数：	变量名称用“*”作为分隔符。 按照命令中变量名称的顺序，采用寄存器 REG[寄存器索引] 中的值和下列值。 关系到： 第一个变量的值在 REG[寄存器索引] 中。 第二个变量的值在 REG[寄存器索引 + 1] 中 ， 以次类推

说明

请注意，变量列表最多为 500 个字符，寄存器的数量也有一定限制。

示例

```
MRNP("$R[0]*$R[1]*$R[2]*$R[3]",1)      ; REG[1] 至 REG[4] 以变量值 $R[0] 至 $R[3] 描述。
```

NC 变量

系统提供所有的机床数据、设定数据和 R 参数，但是只提供一部分的系统变量（参见 可用的系统变量列表 (页 265)).

可访问所有全局的和通道专用的用户变量（GUD）。 本地和程序全局的用户变量无法编辑。

机床数据	
全局机床数据	$MN_...
轴专用的机床数据	$MA_...
通道专用的机床数据	$MC_...

设定数据	
全局设定数据	$SN_...
轴专用的设定数据	$SA_...
通道专用的设定数据	$SC_...

系统变量	
R 参数 1	$R[1]

PLC 变量

所有 PLC 数据可用。

PLC 数据	
数据模块 x 的字节 y 位 z	DBx.DBXy.z
数据模块 x 的字节 y	DBx.DBBy
数据模块 x 的字 y	DBx.DBWy
数据模块 x 的双字 y	DBx.DBDy
数据模块 x 的实数 y	DBx.DBRy
标记字节 x 位 y	Mx.y
标记字节 x	MBx
标记字 x	MWx
标记双字 x	MDx
输入字节 x 位 y	Ix.y 或者 Ex.y
输入字节 x	IBx 或者 EBx
输入字 x	IWx 或者 EWx

PLC 数据	
输入双字 x	IDx 或者 EDx
输出字节 x 位 y	Qx.y 或者 Ax.y
输出字节 x	QBx 或者 ABx
输出字 x	QWx 或者 AWx
输出双字 x	QDx 或者 ADx
数据模块 x 中长度 z 的字符串 y	DBx.DBSy.z

6.3.28 PI 服务

说明

通过功能 PI_START 可以在由 PLC 在 NC 区中启动 PI 服务（程序实例服务）。

说明

可用的 PI 服务列表请见功能手册之基本功能。

编程

句法：	**PI_START**（“*传输字符串*”）	
说明：	执行 PI 服务	
参数：	“传输字符串”	传输的字符串与 OEM 文献相反，应用双引号括起。

示例

```
PI_START("/NC,001,_N_LOGOUT")
```

说明

通道相关的 PI 服务总是与当前的通道有关。

刀具功能（TO 区）的 PI 服务总是以分配到当前通道的 TO 区为参考。

6.3.29 读取 (RNP) 和写入 (WNP) 系统或用户变量

说明

通过指令 RNP (读取 NC PLC) 可以读取 NC 或者 PLC 变量或者机床数据。

编程

句法：	**RNP** ("*系统或者用户变量",值*)	
说明：	读取 PLC 或者 NC 变量或者机床数据	
参数：	系统或者用户变量	NC 或者 PLC 变量名称
	值	要写入系统或者用户变量中的值。 如果值类型为字符串，则必须以双引号括起。

示例

```
VAR2=RNP("$AA_IN[2]")          ; 读取 NC 变量
```

说明

通过指令 WNP (写入 NC PLC) 可以写入 NC 或者 PLC 变量或者机床数据。

在每次处理功能 WNP 都重新存取 NC 变量或 PLC 变量，即：总是在 CHANGE 方法下存取 NC 变量或 PLC 变量。这在系统或者用户变量经常改变值情况下有意义。 如果只进行一次 NC/PLC 存取，则必须在 LOAD（装载）或者 UNLOAD（卸载）方法下设计。

编程

句法：	**WNP***("系统或者用户变量", 值)*	
说明：	写入 PLC 或者 NC 变量或者机床数据	
参数：	系统或者用户变量	NC 或者 PLC 变量名称
	值	要写入系统或者用户变量中的值。 如果值类型为字符串，则必须以双引号括起。

示例

```
WNP("DB20.DBB1",1)          ; 写入 PLC 变量
```

6.3.30 RESIZE_VAR_IO 和 RESIZE_VAR_TXT

说明

使用命令 RESIZE_VAR_IO() 和 RESIZE_VAR_TXT() 可以更改变量的输入/输出分量或文本分量的几何尺寸。

设置了新的几何尺寸后，屏幕的所有栏都会调整位置，就像屏幕从一开始就是按这样的位置配置的。从而根据配置，各栏会相互重新对齐。

编程

句法：	**RESIZE_VAR_IO***(变量名，[X]，[Y]，[宽度]，[高度])* **RESIZE_VAR_TXT***(变量名，[X]，[Y]，[宽度]，[高度])*	
说明：	更改变量的输入/输出分量或文本分量的几何尺寸。	
参数：	变量名	要更改输入/输出分量或文本分量的几何尺寸的变量的名称。
	X	左上 X 坐标
	Y	左上 Y 坐标
	宽度	宽度
	高度	高度

说明

如未给定 X、Y、宽度或高度的值，则保留当前值。如将其中一个值给定为 -1，则会设为由 “Run MyScreens” 预设的标准位置分量。

示例

```
RESIZE_VAR_IO("MyVar1", 200, , 100)    ; 变量“MyVar1”的 IO 分量偏移至 X 位置 200 像素，宽度设为 100 像素。之前的 Y 位置和高度值被保留。
```

6.3.31 寄存器(REG)

寄存器说明

寄存器用于在两个不同的对话框之间切换数据。 寄存器分配给每个对话框并且在载入第一个对话框时生成，以 0 或空字符串预占。

说明

寄存器不允许直接在 OUTPUT 方法中被用于 NC 代码的生成。

编程

句法：	**REG***[x]*	
说明：	定义寄存器	
参数：	x	寄存器编号，以 x = 0...19 类型： REAL 或者 STRING = VARIANT x ≥ 20 的寄存器由西门子使用。

寄存器值说明

寄存器值的分配在方法中设计。

说明

如果由一个对话框通过功能 LM 生成另一个对话框，则寄存器的内容自动接受入新的对话框中并在第二个对话框中用于其它计算。

编程

句法：	*名称*.val = *寄存器值* 或者 *名称* = *寄存器值*	
说明：		
参数：	名称	寄存器名称
	寄存器值	寄存器的值

示例

```
UNLOAD
  REG[0] = VAR1          ; 寄存器 0 赋值为变量 1 的值
END_UNLOAD

UNLOAD
REG[9].VAL = 84          ; 寄存器 9 赋值为变量 84 的值
END_UNLOAD
                         ; 在下列对话框中该寄存器可以在方法中重新分配本地变量。
LOAD
  VAR2 = REG[0]
END_LOAD
```

寄存器状态说明

通过状态特性可以在设计中查询寄存器是否包含一个有效值。

此外，当一个对话框作为主对话框使用时，可以使用寄存器的状态查询向一个寄存器中仅写入一个值。

编程

句法：	*名称*.vld	
说明：	该属性仅可读。	
参数：	名称	寄存器名称
返回值：		查询结果可能是：
	FALSE =	无效值
	TRUE =	有效值

示例

```
IF REG[15].VLD == FALSE     ; 查询寄存器值的有效性
REG[15] = 84
ENDIF
VAR1 = REG[9].VLD           ; 向 Var1 分配 REG[9] 状态查询的值。
```

6.3.32 RETURN

说明

通过 RETURN 功能可以提前取消当前的子程序处理并返回到最后一次 CALL 命令的跳转位置。

如果子程序中没有设计 RETURN，则执行子程序，直至结束然后返回到跳转位置。

编程

句法：	**RETURN**	
说明：	返回跳转位置	
参数：	- 无 -	

示例

```
//B(PROG1)                ; 块开始
SUB(UP2)                  ; 子程序开始
  IF VAR1.val=="Otto"
    VAR1.val="Hans"
RETURN                    ; 如果变量值 = Otto，则赋值 "Hans" 给变量， 子程序在此位置结束。
ENDIF
  VAR1.val="Otto"         ; 如果变量值 ≠ Otto，则赋值  "Otto" 给变量。
END_SUB                   ; 子程序结束
//END                     ; 块结束
```

6.3.33 反编译

说明

在编程支持中可以**反编译**功能 GC 生成的 NC 代码并在所属输入对话框中的输入/输出栏中再次显示变量值。

编程

来自 NC 代码的变量接受至对话框中。 此时对来自 NC 代码的变量值与来自配置文件的计算的变量值加以比较。 如果存在不一致，则在日志文件中给出错误信息，因为在生成的 NC 代码中数值已改变。

如果一个变量在 NC 代码中多次存在，则在反编译时总是分析该变量最后一次出现的值。另外在日志文件中给出警告。

代码生成时，不在 NC 代码中使用的变量作为使用注释存储。 通过使用注释标记所有反编译时需要的信息。 使用注释不得更改。

说明

NC 代码块和使用注释如果在一行的启始处开始，则只能反编译。

示例：

在程序中有下列 NC 代码：

```
DEF VAR1 = (I//101)
OUTPUT(CODE1)
  "X" VAR1 " Y200"
  "X" VAR1 " Y0"
END_OUTPUT
```

在零件程序中存放下列代码：

```
;NCG#TestGC#\cus.dir\aeditor.com#CODE1#1#3#
X101 Y200
X101 Y0
;#END#
```

在反编译时读取编辑器：

```
X101 Y200
X222 Y0          ; 在零件程序中更改 X 的值 (X101 → X222)
```

在输入对话框中给出下列 VAR1 值： `VAR1 = 222`

参见

生成代码(GC) (页 147)

6.3.34 无注释反编译

说明

在编程支持中，可以对通过 GC 功能生成的 NC 代码执行**无注释反编译**，并在对应输入对话框的输入输出栏中显示变量值。

编程

可以按照以下方式执行 GC 指令，忽略在常规的代码生成中产生的注释行：

```
GC("CODE1",D_NAME,1)
```

产生的代码通常是无法反编译的，但是执行以下操作，便可以反编译生成的循环调用代码：

- **补充文件“easyscreen.ini”**
  在文件“easyscreen.ini”中加入段落[RECOMPILE_INFO_FILES]。在该段落中列出所有 “ini” 文件，这些文件说明了哪些循环可以进行无注释反编译：
  ```
  [RECOMPILE_INFO_FILES]
  IniFile01 = cycles1.ini
  IniFile02 = cycles2.ini
  ```
  可以指定多个“INI”文件，文件名称可自由选择。
- **新建包含循环说明的“INI”文件**
  将含有循环说明的 ini 文件保存到以下路径中：
  [系统用户目录]/cfg
  [系统 oem 目录]/cfg
  [系统插件目录]/cfg
  在该文件中，每个循环需要占用一个单独的说明段落。段落的名称就是循环的名称：
  ```
  [Cycle123]
  Mname = TestGC
  Dname = testgc.com
  OUTPUT = Code1
  Anzp = 3
  Version = 0
  Code_typ = 1
  Icon = cycle123.png
  Desc_Text = This is describing text
  ```

Mname	窗口名称
Dname	对窗口进行定义的文件的名称

OUTPUT	所涉及 OUTPUT 方法的名称
Anzp	需要反编译的窗口的参数数量（指所有用 DEF 创建的变量，也包含辅助变量）
Version	循环的版本，可选
Icon	在工步链程序中显示的图符，格式为“png” 不同图片大小对应的图符分辨率： 640 x 480 mm → 16 x 16 像素 800 x 600 mm → 20 x 20 像素 1024 x 768 mm → 26 x 26 像素 1280 x 1024 mm → 26 x 26 像素 1280 x 768 mm → 26 x 26 像素 保存：*[系统用户目录]*/ico/ico<分辨率> **提示：**分辨率为“1280 x xx mm”的图片保存在用于“1024 x 768 mm”的文件夹下（仅针对工步链程序）。
Desc_Text	工步链程序中显示画面上的说明性文本，可选，字符串最多 17 个字符（仅针对工步链程序）。

示例

```
//M(测试 GC/"代码生成:")
DEF VAR1 = (R//1)
DEF VAR2 = (R//2)
DEF D_NAME
LOAD
  VAR1 = 123
  VAR2 = -6
END_LOAD
OUTPUT(CODE1)
  "Cycle123(" VAR1 "," VAR2 ")"
  "M30"
END_OUTPUT

PRESS(VS1)
  D_NAME = "\MPF.DIR\MESSEN.MPF"
  GC("CODE1",D_NAME)                    ; 将 OUTPUT 方法的 NC 代码写入文件 \MPF.DIR
                                        \MESSEN.MPF 中：
                                        Cycle123(123, -6)
                                        M30
```

```
END_PRESS
```

边界条件

- “无注释反编译”功能不具备“带注释反编译”的全部功能。
  “无注释反编译”支持典型的循环调用，如 MYCYCLE(PAR1, PAR2, PAR3, …)。但在函数调用行中不允许存在任何注释。但是，不在函数调用时传递的以及字符串 S 型的可选参数至少应带有空的引号，如 **“”**。否则，“Run MyScreens” 会尝试使用逗号替代该参数，接着将其反编译为“被替代的循环调用”。
- 字符串型参数不允许待传递的字符串中含有逗号和分号。
- “无注释反编译”时，OUTPUT 方法中包含的所有变量都应位于括号内，这样功能“使用逗号替代缺少的循环参数”才能生效。

示例：

允许：

```
OUTPUT
    "MYCYCLE(" MYPAR1 "," MYPAR2 "," MYPAR3 ")"
END_OUTPUT
```

不允许（变量 MYCOMMENT 位于右括号后面）：

```
OUTPUT
   "MYCYCLE(" MYPAR1 "," MYPAR2 "," MYPAR3 ")" MYCOMMENT
END_OUTPUT
```

6.3.35 向前/后查找(SF, SB)

说明

通过功能 **SF（向前查找）**、**SB（向后查找）**可以在编辑器当前的 NC 程序中从当前光标位置开始查找某个字符串并输出其值。

编程

句法：	**SF***("字符串")*	
名称：	**S**earch **F**orward: 从当前光标位置向前查找	
句法：	**SB***("字符串")*	
名称：	**S**earch **B**ackward: 从当前光标位置向后查找	
参数：	字符串	要查找的文本

查找规则：

- 要查找的字符串和数值的单元在 NC 程序中前后必须有空格。
- 查找对象无法在注释中和字符串内查找。
- 给出的值必须是一个数字表达式，表达式形式“X1=4+5”无法识别。
- 可识别十六进制常数的形式 X1=‘HFFFF'、二进制常数的形式 X1=‘B10010' 和指数常数的形式 X1=‘-.5EX-4’ 。
- 在字符串和数值之间有下列符号，可以给出字符串值：
  - 无
  - 空格键
  - 等号

示例

允许以下的写入方式：

```
X100 Y200                ; 变量 Abc 包含数值 200
Abc = SB("Y")
X100 Y 200               ; 变量 Abc 包含数值 200
Abc = SB("Y")
X100 Y=200               ; 变量 Abc 包含数值 200
Abc = SB("Y")
```

6.3.36 字符串功能

概述

以下的功能允许进行字符串处理：

- 确定字符串长度
- 查找字符串中的一个字符
- 由左提取部分字符串
- 由右提取部分字符串
- 由字符串中间提取部分字符串
- 替换部分字符串
- 字符串的比较
- 将字符串插入到另一个字符串中
- 从字符串中删除一个字符串
- 删除空格符（从左或从右）
- 插入带格式化标识的值或字符串

LEN 功能：字符串长度

句法：	**LEN***(字符串 | 变量名称)*	
说明：	确定一个字符串的字符数目。	
参数：	字符串	每个有效的字符串表达式。对于一个空字符串，会返回零。
	变量名称	每个有效的和表示的变量名称
	仅允许两个可行参数中的一个。	

示例

```
DEF VAR01
DEF VAR02

LOAD
  VAR01="HALLO"
```

```
  VAR02=LEN(VAR01)           ; 结果 = 5
END_LOAD
```

NSTR 功能：查找字符串中的字符

句法：	**INSTR***(开始, 字符串 1, 字符串 2 [,方向])*	
说明：	查找字符	
参数：	开始	从字符串 1 向字符串 2 查找的开始位置。如果从字符串 2 的开头开始查找，则指定为 0。
	字符串 1	要查找的字符。
	字符串 2	在该字符链中查找
	方向（可选）	查找的方向 0:从左向右（预设） 1:从右向左
	如果字符串 2 中没有包含字符串 1，则返回数值 0。	

示例

```
DEF VAR01
DEF VAR02

LOAD
  VAR01="HALLO/WELT"
  VAR02=INST(1,"/",VAR01)        ; 结果 = 6
END_LOAD
```

LEFT 功能：左边字符串

句法：	**LEFT***(字符串, 长度)*	
说明：	LEFT 返回一个字符串，该字符串从字符串左侧开始包含指定的字符数。	
参数：	字符串	字符串或者带有要处理的字符串的变量
	长度	要读取的字符数目

示例

```
DEF VAR01
DEF VAR02
LOAD
  VAR01="HALLO/WELT"
  VAR02=LEFT(VAR01,5)           ; 结果 = “HALLO”
END_LOAD
```

RIGHT 功能：右边字符串

句法：	**RIGHT***(字符串, 长度)*	
说明：	RIGHT 返回一个字符链，该字符链从字符串右侧开始包含指定的字符数。	
参数：	字符串	字符串或者带有要处理的字符串的变量
	长度	要读取的字符数目

示例

```
DEF VAR01
DEF VAR02
LOAD
  VAR01="HALLO/WELT"
  VAR02=LEFT(VAR01,4)          ; 结果 = “WELT”
END_LOAD
```

MIDS 功能：中间字符串

句法：	**MIDS***(字符串, 开始 [, 长度])*	
说明：	MIDS 返回一个字符串，该字符串从字符串的指定位置开始包含指定的字符数。	
参数：	字符串	字符串或者带有要处理的字符串的变量
	开始	开始，从字符链中该处开始读取
	长度	要读取的字符数目

示例

```
DEF VAR01
DEF VAR02
LOAD
  VAR01="HALLO/WELT"
  VAR02=LEFT(VAR01,4,4)            ; 结果 = "LO/W"
END_LOAD
```

REPLACE 功能：替换字符

句法：	**REPLACE** *(字符串, 查找字符串, 替换字符串 [, 开始 [, 计数]])*	
说明：	功能 REPLACE 用另一个字符/字符链替代字符串中的一个字符/字符链。	
参数：	字符串	待通过替换字符串替换查找字符串的字符串。
	查找字符串	需被替代的字符串
	替代字符串	替代字符串（位于查找字符串位置）
	开始	查找和替换的开始位置
	计数器	从开始位置起要开始查找的查找字符串的字符数量
返回值：		
	字符串 = 空字符串	复制字符串
	查找字符串 = 空字符串	复制字符串
	替换字符串 = 空字符串	复制字符串，在该字符串中删除所有出现的查找字符串
	开始 > Len(长度)	空字符串
	计数 = 0	复制字符串

STRCMP 功能：比较字符串

句法：	**STRCMP***(字符串 1，字符串 2 [, CaseInsensitive])*
说明：	STRCMP 将一个字符链与另一个字符链进行比较。

参数：	字符串 1	要与第二个字符链进行比较的字符链
	字符串 2	要与第一个字符链进行比较的字符链
	CaseInsensitive	比较时区分/不区分大/小写： 未说明或 FALSE = 区分大/小写 TRUE = 不区分大/小写

示例

```
REG[0]=STRCMP("Hugo", "HUGO")                ; 结果 = 32
                                             ; (<> 0，字符串不相同)

REG[0]=STRCMP("Hugo", "HUGO", TRUE)          ; 结果 = 0
                                             ; (== 0，字符串相同
```

STRINSERT 功能：插入字符串

句法：	**STRINSERT***(字符串 1，字符串 2，插入位置)*	
说明：	STRINSERT 可在一个字符链的某个特定位置上插入另一个字符链。	
参数：	字符串 1	要向其中插入一个字符链的字符链
	字符串 2	要插入的字符链
	插入位置	插入字符链的位置

示例

```
REG[0]=STRINSERT("Hello!", " world", 5)    ; 结果 = "Hello world!")
```

STRREMOVE 功能：删除字符串

句法：	**STRREMOVE***(字符串 1，删除位置，计数)*	
说明：	STRREMOVE 可从字符链的某一位置上删除一定数量的字符。	
参数：	字符串 1	要从其中删除字符的字符链
	删除位置	从字符链的该位置起删除字符
	计数	要删除的字符数量

示例

```
REG[0]=STRREMOVE("Hello world!", 5, 6)     ; 结果 = "Hello!")
```

TRIMLEFT 功能：从字符串左侧删除空格符

句法：	**TRIMLEFT***(字符串 1)*	
说明：	TRIMLEFT 可从字符链的左侧删除空格符。	
参数：	字符串 1	要从其左侧开始删除空格符的字符链

示例

```
REG[0]=TRIMLEFT("    Hello!")              ; 结果 = "Hello!")
```

TRIMRIGHT 功能：从字符串右侧删除空格符

句法：	**TRIMRIGHT***(字符串 1)*	
说明：	TRIMRIGHT 可从字符链的右侧删除空格符。	
参数：	字符串 1	要从其右侧开始删除空格符的字符链

示例

```
REG[0]=TRIMRIGHT("Hello!    ")             ; 结果 = "Hello!")
```

FORMAT 功能：插入带格式化标识的值或字符串

句法：	**FORMAT***(带格式化标识的文本[,值 1] ... [,值 28])*
说明：	FORMAT 功能可通过使用格式化标识在预设文本的某个位置插入最多 28 个值或字符串。

格式化标识：	句法	%[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”） ● 缺省值：当要输出的值的数位少于[宽度]定义时，使用空格符补齐
	宽度	确定一个非负数最小输出宽度的依据。如果值的数位少于依据所确定的，则使用空格符补齐。
	小数点后的数位	使用浮点数时，该可选参数用来确定小数点后面的位数。
	类型	类型说明符用来确定 PRINT 指令会输出哪些数据格式。 该说明符必须进行给定。 ● d:整型值 ● f：浮点数 ● s:字符串 ● o:八进制 ● x:十六进制 ● b:二进制

示例

```
DEF VAR1
DEF VAR2
LOAD
  VAR1 = 123
VAR2 = FORMAT("Hello %08b %.2f %s!", VAR1 + 1, 987.654321, "world")
                                    ; 结果 = "Hello 01111100 987.65 world!"
END_LOAD
```

参见

使用字符串 (页 94)

6.3.37 WHILE/UNTIL 循环

说明

使用 DO-LOOP 命令可实现一个循环。取决于配置，这个循环会一直执行，直到满足一个条件(WHILE)或一个条件适用(UNTIL)。

取决于配置，由于循环会影响系统的性能，因此请您慎重使用这些循环并且放弃使用循环中费时的动作。

例如，建议使用寄存器(REG[])作为运行变量，因为一般显示变量（尤其是那些使用系统或用户变量连接的）会极其频繁地刷新或写入，这同样会影响系统性能。

借助功能 DEBUG（参见章节 DEBUG (页 139)）可对 “Run MyScreens” 方法的运行时间进行检测。这样还可对循环可能导致的问题（高 CPU 负载、响应性能降低）进行识别。

说明

由于 FOR 循环可被 WHILE 循环替代，因此在 EasyScreen 中不支持 FOR 循环的表达句法。

编程

```
DO
   <指令>
LOOP_WHILE <循环继续执行的条件>

DO
   <指令>
LOOP_UNTIL <循环结束的条件>

DO_WHILE <循环继续执行的条件>
   <指令>
LOOP

DO_UNTIL <循环结束的条件>
   <指令>
LOOP
```

示例

```
REG[0] = 5
DO
   DEBUG("OUTER: " << REG[0])
   REG[0] = REG[0] + 1
   REG[1] = -5

   DO
      DEBUG("INNER: " << REG[1])
      REG[1] = REG[1] + 1
   LOOP_WHILE REG[1] < 0

LOOP_WHILE REG[0] < 10
```

```
REG[0] = 5
DO
   DEBUG("OUTER: " << REG[0])
   REG[0] = REG[0] + 1
   REG[1] = -5

   DO
      DEBUG("INNER: " << REG[1])
      REG[1] = REG[1] + 1
   LOOP_UNTIL 0 <= REG[1]

LOOP_WHILE 10 > REG[0]
```

```
REG[0] = 5
DO_WHILE 10 > REG[0]
   DEBUG("OUTER: " << REG[0])
   REG[0] = REG[0] + 1
   REG[1] = -5

   DO_UNTIL 0 <= REG[1]
     DEBUG("INNER: " << REG[1])
     REG[1] = REG[1] + 1
   LOOP
```

```
LOOP
```

```
REG[0] = 5
DO_WHILE 10 > REG[0]
   DEBUG("OUTER: " << REG[0])
   REG[0] = REG[0] + 1
   REG[1] = -5

   DO
     DEBUG("INNER: " << REG[1])
     REG[1] = REG[1] + 1
   LOOP_UNTIL 0 <= REG[1]

LOOP
```

6.3.38 循环执行脚本： START_TIMER，STOP_TIMER

说明

借助计时器可循环调用 SUB 方法。 为此提供了功能 START_TIMER()和 STOP_TIMER()。

说明

只可为每个 SUB 方法配置一个计时器。

编程

句法：	**START_TIMER***(“SUB 名称”， 间隔)*	
说明：	开始 SUB 方法的循环执行	
参数：	SUB 名称：	要循环调用的 SUB 方法的名称
	间隔：	以 ms 为单位的时间间隔

句法：	**STOP_TIMER***("SUB 名称")*	
说明：	停止 SUB 方法的循环执行	
参数：	SUB 名称：	要停止其计时器的 SUB 方法的名称

示例

```
//M(TimerSample/"My timer")
DEF MyVariable=(I//0/,"Number of cyclic calls:"/WR1)
VS1=("Start%ntimer")
VS2=("Stop%ntimer")

SUB(MyTimerSub)
   MyVariable = MyVariable + 1
END_SUB

PRESS(VS1)
   ; 每 1000 ms 调用一次 SUB "MyTimerSub"
   START_TIMER("MyTimerSub", 1000)
END_PRESS

PRESS(VS2)
   STOP_TIMER("MyTimerSub")
END_PRESS
```

如果为已经分配给一个 SUB 方法的计时器重新调用 START_TIMER，则在有差异时会接收新的间隔。 否则这两次调用都会被忽略。

由于系统的限制，最短间隔为 100 ms。

如果为一个 SUB 方法调用 STOP_TIMER，而当时没有正在运行的计时器，则会忽略此次调用。

图形单元和逻辑单元 7

7.1 线、分割线、矩形、圆形和椭圆形

说明

线、分割线、矩形和椭圆形/圆形在 LOAD 方法中配置：

- 可将填充色设置为系统背景色，以得到透明的矩形。
- 只要高度和宽度设为相同，元素 ELLIPSE 上就可生成圆形。
- 水平和垂直分割线始终与窗口宽度或窗口高度完全一致。分割线不会产生滚动条。
  如果此前一直将元素 RECT 用于显示分割线，例如 RECT(305,0,1,370,0,0,1)，这会被编程支持识别到并自动转换为 V_SEPARATOR(305,1,"#87a5cd", 1)。这同时适用于垂直和水平分割线。

LINE 元素

编程：

句法：	LINE (x1,y1,x2,y2,f,s)	
说明：	定义线	
参数：	x1	起点的 x 坐标
	y1	起点的 y 坐标
	x2	终点的 x 坐标
	y2	终点的 y 坐标
	f	线的颜色
	s	线的样式： 1 = 实线 2 = 虚线 3 = 点线 4 = 点虚线

RECT 元素

编程：

句法：	RECT (x,y,w,h,f1,f2,s)	
说明：	定义矩形	
参数：	x	左上 x 坐标
	y	左上 y 坐标
	w	宽度
	h	高度
	f1	边框颜色
	f2	填充色
	s	边框样式： 1 = 实线 2 = 虚线 3 = 点线 4 = 点虚线

ELLIPSE 元素

编程：

句法：	ELLIPSE(x,y,w,h,f1,f2,s)	
说明：	定义椭圆或圆形	
参数：	x	左上 x 坐标
	y	左上 y 坐标
	w	宽度
	h	高度
	f1	边框颜色
	f2	填充色
	s	边框样式： 1 = 实线 2 = 虚线 3 = 点线 4 = 点虚线

高度与宽度值相同时为圆形。

V_SEPARATOR 元素

编程：

句法：	V_SEPARATOR (x,w,color,pen)	
说明：	定义垂直分割线	
参数：	x	x 位置
	w	线条粗细
	color	颜色
	pen	线的样式： 1 = 实线 2 = 虚线 3 = 点线 4 = 点虚线

H_SEPARATOR 元素

编程：

句法：	H_SEPARATOR (y,h,color,pen)	
说明：	定义水平分割线	
参数：	y	y 位置
	h	线条粗细
	color	颜色
	pen	线的样式： 1 = 实线 2 = 虚线 3 = 点线 4 = 点虚线

另见

LOAD (页 115)

参见

CLEAR_BACKGROUND (页 130)

7.2 定义数组

定义

通过数组可以排列或者保存同一数据类型的数据，从而可以通过索引存取数据。

说明

数组可以是一维或者二维。 一个一维数组可视为带有一行或者一列的一个二维数组。

数组通过标记 //A 定义并通过 //END 结束。 行和列数目任意。 一个数组有下列结构：

编程

句法：	**//A***(名称)* (a/b...) (c/d...) ... //END	
说明：	定义数组	
参数：	名称	数组名称
	a, b, c, d	数组值 STRING 类型的值必须用双引号括起。

示例

```
//A(螺纹)                          ；高度/螺距/底直径
(0.3 / 0.075 / 0.202)
(0.4 / 0.1   / 0.270)
(0.5 / 0.125 / 0.338)
(0.6 / 0.15  / 0.406)
(0.8 / 0.2   / 0.540)
```

```
(1.0 / 0.25  / 0.676)
(1.2 / 0.25  / 0.676)
(1.4 / 0.3   / 1.010)
(1.7 / 0.35  / 1.246)
//END
```

7.2.1 存取数组单元的值

说明

通过属性值（名称.val）可以继续传送一个数组存取值。

行索引（数组的行编号）和列索引（数组的列编号）各自从 0 开始。如果显示数组外的行索引或者列索引，则输出值 0 或者空字符串并且变量 ERR 的值为 TRUE。 当未找到查找关键字时，变量 ERR 同样为 TRUE。

编程

句法：	名称 [Z,[M[,C]]].val 或者 名称 [Z,[M[,C]]]
说明：	存取仅带有一列的一维数组：
句法：	名称 [S,[M[,C]]].val] 或者 名称 [S,[M[,C]]] 或者
说明：	存取仅带有一行的单维数组
句法：	名称 [Z,S,[M[,C]]].val 或者 名称 [Z,S,[M[,C]]]
说明：	存取二维数组

参数：	名称：	数组名称	
	Z:	行值（行索引或者查找关键字）	
	S:	列值（列索引或者查找关键字）	
	M:	存取模式	
		0	直接
		1	按行查找，列直接
		2	按列查找，行直接
		3	查找
		4	查找行索引
		5	查找列索引
	C:	比较模式	
		0	查找关键字必须位于行或者列的值范围内。
		1	查找关键字必须准确找到

示例：		
	`VAR1 = MET_G[REG[3], 1,0].VAL`	`；分配 Var1 数组中的一个值 MET_G`

存取模式

- **存取模式“直接”**
  在存取模式“直接”（M = 0）情况下，数组上的存取通过行索引（以 Z 表示）和列索引（以 S 表示）实现。不评估比较模式 C。
- **存取模式“查找”**
  在存取模式 M = 1、2 或者 3 情况下，查找总是在行 0 或者列 0 中实现。

模式 M	行值 Z	列值 S	输出值
0	行索引	栏索引	行 Z 和列 S 中的值
1	查找关键字： 在列 0 中查找	列索引，从该列中读取值	查找的行和列 S 中的值
2	行索引，从该行中读取返回值	查找关键字： 在行 0 中查找	行 Z 和查找的列中的值

模式 M	行值 Z	列值 S	输出值
3	查找关键字： 在列 0 中查找	查找关键字： 在行 0 中查找	查找的行和查找的列中的值
4	查找关键字： 在列 S 中查找	行索引，在该行中查找	行索引
5	行索引，在该行中查找	查找关键字： 在行 Z 中查找	栏索引

比较模式

在使用比较模式 C = 0 时，查找行或者查找列的内容以升序分类。 如果查找关键字小于第一个元素或者大于最后一个元素，则给出值 0 或者一个空字符并且错误变量 ERR 为真。

在使用比较模式 C = 1 时，查找关键字必须可在查找行或者查找列中找到。 如果没有找到查找关键字，则给出值 0 或者一个空字符并且错误变量 ERR 为真。

7.2.2 举例： 存取数组单元

前提条件

下列定义两个数组是下面例子的前提条件。

```
//A(螺纹)
                (0.3 / 0.075 / 0.202)
                (0.4 / 0.1   / 0.270)
                (0.5 / 0.125 / 0.338)
                (0.6 / 0.15  / 0.406)
                (0.8 / 0.2   / 0.540)
                (1.0 / 0.25  / 0.676)
                (1.2 / 0.25  / 0.676)
                (1.4 / 0.3   / 1.010)
                (1.7 / 0.35  / 1.246)
//END
```

```
//A(Array2)
                ("BEZ" /     "STG" /     "KDM" )
                (0.3 /       0.075 /     0.202 )
                (0.4 /       0.1 /       0.270 )
                (0.5 /       0.125 /     0.338 )
                (0.6 /       0.15 /      0.406 )
                (0.8 /       0.2 /       0.540 )
                (1.0 /       0.25 /      0.676 )
                (1.2 /       0.25 /      0.676 )
                (1.4 /       0.3 /       1.010 )
                (1.7 /       0.35 /      1.246 )
//END
```

示例

- **存取模式 1 示例：**
  在 Z 中有查找关键字。 该关键字总是在列 0 中查找。 通过查找的关键字的行索引给出列 S 中的值：
  `VAR1 = 螺纹[0.5,1,1] ; VAR1 值为 0.125`
  说明:
  数组“螺纹”的列 0 中查找值 0.5 并给出列 1 中查找的相同行的值。

- **存取模式 2 示例：**
  在 S 中有查找关键字。 该关键字总是在行 0 中查找。 通过查找的关键字的列索引给出行 Z 中的值：
  `VVAR1 = ARRAY2[3,"STG",2] ; VAR1 值为 0.125`
  说明:
  在数组“Array2”的行 0 中查找带有内容“STG”的列。 得到查找的列中的值和带有索引 3 的行。

- **存取模式 3 示例：**
  在 Z 和 S 中总是有一个查找关键字。 通过 Z 中的关键字在列 0 中找到行索引和通过 S 中的关键字在行 0 中找到列索引。 通过查找的行索引和列索引输出数组中的值：
  `VAR1 = ARRAY2[0.6,"STG",3]  ;VAR1 值为 0.15`
  说明:
  在数组“Array2”的列 0 中查找带有内容 0.6 的行，在数组“Array2”的行 0 中查找带有内容“STG”的列。 根据 VAR1 给出查找的行和列中的值。

- **存取模式 4 示例：**
  在 Z 中有查找关键字。 在 S 中有要查找的列索引。 给出查找的关键字的行索引：
  `VAR1 = 螺纹[0.125,1,4]  VAR1 值为 2`
  说明:
  在数组“ 螺纹的列 1 中查找值 0.125 并根据 VAR1 给出查找的值的行索引。
- **存取模式 5 示例：**
  在 Z 中有要查找行的行索引。 查找关键字在 S 中。给出查找的关键字的列索引：
  `VAR1 = 螺纹[4,0.2,5,1]  VAR1 值为 1`
  说明:
  在数组“ 螺纹” 的行 4 中查找值 0.2 并根据 VAR1 给出查找的值的列索引。 已选择比较模式 1，因为行 4 的值不是按升序分类的。

7.2.3 查询数组单元的状态

说明

通过属性状态可以查询数组存取是否提供一个有效值。

编程

句法：	*名称 [Z, S, [M[,C]]]*.**vld**
说明：	该属性仅可读。
参数：	名称　　数组名称
返回值：	FALSE =无效值 TRUE = 有效值

示例

```
DEF MPIT = (R///"MPIT",,"MPIT",""/wr3)
DEF PIT  = (R///"PIT",,"PIT",""/wr3)
PRESS(VS1)
  MPIT = 0.6
  IF MET_G[MPIT,0,4,1].VLD == TRUE
    PIT    = MET_G[MPIT,1,0].VAL
    REG[4] = PIT
    REG[1] = "OK"
```

```
ELSE
    REG[1] = "ERROR"
ENDIF
END_PRESS
```

7.3 表格描述(栅格)

定义

与数组相反，表格(栅格)的值会持续更新。 这会涉及（例如）来自 NC 或 PLC 的数值的表格式显示。 表格是按列组织排列的，因此一列中始终为同一类型的变量。

分配

表格描述参考内容收录在变量描述中：

- 变量定义确定显示的值，表格单元定义确定屏幕的外观和布置。 来自变量定义行的输入/输出栏属性将会接收到表格中。
- 表格的可见区域通过输入/输出栏的宽度和高度确定。 如果比可见区域位置存在更多的行和列，则可以通过水平和垂直滚动条进行查看。

表格名称

NCK/PLC 相同类型值的表格名称，它可以通过通道模块编译地址。 表格名称通过一个前置的极限值或者转换栏的 % 符号进行区分。 表格命名符可以通过逗号分隔，还可以跟随一个文件名，该文件名指定定义表格描述的文件。

同类型值的表格命名符，例如来自 NCK 或 PLC 的值。 表格命名符在配置极限值或转换栏的地方给定。 表格命名符前会带有 % 符号。 之后可用逗号隔开并加上一个文件名。 这用于给定在哪个文件中定义表格描述。 表格默认会在配置文件中查找屏幕本身的配置位置。

表格定义也可以通过功能 Load Grid (LG)进行动态加载，例如在 LOAD 方法中。

系统或者用户变量

这些参数保留为空以用于表格，因为详细信息中要显示的变量会在列定义行中进行给定。 表格描述可以动态提供。

示例

变量 MyGridVar 的定义，该变量在您的输入/输出栏（左边距：100，上边距：标准，宽度：350，高度：100）中显示为一个栅格“MyGrid1”。

```
DEF MyGridVar=(V/% MyGrid1///////100,,350,100)
```

另见

变量参数 (页 79)

Load Grid (LG) (页 156)

7.3.1 定义表格(栅格)

说明

表格块由以下部分组成：

- 标题描述
- 1 至 n 列描述

编程

句法：	***//G****（表格名称/表格类型/行数 /* *[固定行属性]，[固定列属性])*
说明：	定义表格(栅格)

参数：	表格名称	这里表格名称不使用前缀 % 符号。 表格名称只能在一个对话框中使用一次。	
	表格类型	0（预设置）	PLC 或者用户数据（NCK 和通道专用的数据）表
		1	备用
	行数	行数包括标题行	
		固定行或者固定列无法滚动显示。 列数由设计的列的数目给定。	
	列标题行的显示	-1:	不显示列标题行
		0:	显示列标题行（默认）
	固定列的数量	0:	无固定列
		1 ... n:	列数应在各列都可见的范围内，即不需水平滚动

示例

```
//G(grid1/0/5/-1,2)
```
列标题行隐藏并有 2 个固定列

```
//G(grid1/0/5/,1)
```
列标题行显示并有 1 个固定列

```
//G(grid1/0/5)
```
列标题行显示并且无固定列

7.3.2 定义列

说明

为表格(栅格)使用带下标的变量非常重要。索引号码对于带有一个或者多个索引的 PLC 或者 NC 变量比较重要。

您可以对表格中显示的值在由属性确定的权限范围和可能已设定的极限范围内直接加以编辑。

编程

句法：	*（类型/极限值/空/长文本,列标题/属性/帮助图形/ 系统或用户变量/列宽/偏移 1、偏移 2、偏移 3）* 另见 扩展配置句法 (页 38)。	
说明：	定义列	
参数：	和变量类似	
	类型	数据类型
	极限值	最小极限值，最大极限值
	长文本，列标题	
	属性	
	帮助图形	
	系统或者用户变量	作为变量在双引号内给出 PLC 或者 NC 变量。
	列宽	参数，单位像素
	偏移	在分配的偏移参数内规定步宽（在该步宽中各个索引应指数运算），以填写该列。 • 偏移 1：第 1 个索引的步宽 • 偏移 2：第 2 个索引的步宽 • 偏移 3：第 3 个索引的步宽

文本文件的列标题

列标题可以规定为文本或者文本号码（$8xxxx），并且同样无法滚动显示。

改变列属性

可动态改变的（可写）的列属性称为：

- 极限值（最大、最小）、
- 列标题(st)、
- 属性(wr, ac 和 li)
- 帮助图形(hlp)和
- BTSS 变量(var)。

通过定义行中的变量命名符和列索引（以 1 开始）改变列属性。

示例：`VAR1[1].st="Spalte 1"`

对于列定义，可以规定属性 wr、ac 和 li。

另见

Load Grid (LG) (页 156)

7.3.3 表格中的焦点控制（栅格）

说明

通过列和行属性可以在表格中设置和确定聚焦：

- 名称 **Row**
- 名称 **Col**

编程

表格的每行都具有属性 Val 和 Vld。

对于行属性写入和读出，除了定义行中的变量命名符之外，还规定一个行索引和列索引。

句法：	名称[行索引，列索引].Vld 或者 名称[行索引，列索引]
说明：	Val 属性
句法：	名称[行索引，列索引].Vld
说明：	Vld 属性

示例

```
Var1[2,3].val=1.203
```

如果没有规定行索引和列索引，则适用于聚焦行的索引，即：

```
Var1.Row =2
Var1.Col=3
Var1.val=1.203
```

7.4 自定义小部件

7.4.1 自定义小部件

说明

通过自定义小部件，可以在对话框内设计用户专用的显示部件。

软件选件

使用自定义小部件功能额外需要下列软件选件：

“SINUMERIK Integrate Run MyHMI /3GL” (6FC5800-0AP60-0YB0)

编程

定义:	**DEF** *（名称）*	
句法：	**(W///"","***（库名称）***.***（类别名称）***"/////***a,b,c,d**);**	
说明：	W	自定义小部件
参数：	名称	自定义小部件的名称，可自由选择
	库名称	可自由选择，dll (Windows)的名称或 so (Linux) 库文件的名称
	类别名称	可自由选择，是前面指出的库的类别功能的名称
	a, b, c, d	小部件的位置和大小

示例

在对话框配置文件中，可以按照以下方式自定义小部件：

```
DEF Cus =
(W///"","slestestcustomwidget.SlEsTestCustomWidget"/////
20,20,250,100);
```

7.4.2 自定义小部件库的结构

说明

自定义小部件库基本上都包含了一个定义的类别，在对话框配置文件中，应在指定库名称后指定该类别的名称。Run MyScreens 根据库名称访问同名 dll 文件，例如：

```
slestestcustomwidget.dll
```

编程

dll 文件的类别定义应为：

```
#define SLESTESTCUSTOMWIDGET_EXPORT Q_DECL_EXPORT

class SLESTESTCUSTOMWIDGET_EXPORT SlEsTestCustomWidget  : public QWidget
{
   Q_OBJECT
....
public slots:
   bool serialize(const QString& szFilePath, bool bIsStoring);
....
}
```

7.4.3 自定义小部件接口的结构

说明

自定义小部件的显示需要在库中增加一个接口。 该接口包含了 Run MyScreens 初始化自定义小部件的宏定义。 接口在 cpp 文件中定义。 文件名称可以自由定义，例如：
`sleswidgetfactory.cpp`

编程

可以按照下面的方式定义接口：

```
#include "slestestcustomwidget.h"      ；将相关自定义小部件的标题文件添加到文件开头
....
//Makros                               ；保持宏定义
```

```
....
WIDGET_CLASS_EXPORT(SlEsTestCustomWi   ；在文件末尾申明相关的自定义小部件
dget)
```

示例

自定义小部件文件“sleswidgetfactory.cpp”的内容，类别名称为“SlEsTestCustomWidget”：

```
#include <Qt/qglobal.h>
#include "slestestcustomwidget.h"

////////////////////////////////////////////////////////////////////////
// MAKROS FOR PLUGIN DLL-EXPORT - DO NOT CHANGE
////////////////////////////////////////////////////////////////////////

#ifndef Q_EXTERN_C
#ifdef __cplusplus
#define Q_EXTERN_C    extern "C"
#else
#define Q_EXTERN_C    extern
#endif
#endif

#define SL_ES_FCT_NAME(PLUGIN) sl_es_create_ ##PLUGIN
#define SL_ES_CUSTOM_WIDGET_PLUGIN_INSTANTIATE( IMPLEMENTATION , PARAM) \
    { \
    IMPLEMENTATION *i = new PARAM; \
    return i; \
    }

#ifdef Q_WS_WIN
#   ifdef Q_CC_BOR
#       define EXPORT_SL_ES_CUSTOM_WIDGET_PLUGIN(PLUGIN,PARAM) \
    Q_EXTERN_C __declspec(dllexport) void* \
    __stdcall SL_ES_FCT_NAME(PLUGIN) (QWidget* pParent) \
    SL_ES_CUSTOM_WIDGET_PLUGIN_INSTANTIATE( PLUGIN,PARAM )
#   else
#       define EXPORT_SL_ES_CUSTOM_WIDGET_PLUGIN(PLUGIN,PARAM) \
    Q_EXTERN_C __declspec(dllexport) void* SL_ES_FCT_NAME(PLUGIN) \
    (QWidget* pParent) \
    SL_ES_CUSTOM_WIDGET_PLUGIN_INSTANTIATE( PLUGIN,PARAM )
#   endif
```

```
#else
#   define EXPORT_SL_ES_CUSTOM_WIDGET_PLUGIN(PLUGIN,PARAM) \
    Q_EXTERN_C void* SL_ES_FCT_NAME(PLUGIN) (QWidget* pParent) \
    SL_ES_CUSTOM_WIDGET_PLUGIN_INSTANTIATE( PLUGIN,PARAM )
#endif

#define WIDGET_CLASS_EXPORT(CLASSNAME)  \
    EXPORT_SL_ES_CUSTOM_WIDGET_PLUGIN(CLASSNAME,CLASSNAME(pParent))

////////////////////////////////////////////////////////////////////////
// FOR OEM USER - please declare here your widget classes for export
////////////////////////////////////////////////////////////////////////

WIDGET_CLASS_EXPORT(SlEsTestCustomWidget)
```

7.4.4 自定义小部件与对话框的互动 - 自动数据交换

自定义小部件与对话框相互影响并能显示或处理值。

条件

自动数据交换会在以下条件下进行：

条件	方向
打开或反编译对话框时	对话框 → 自定义小部件
执行用于生成循环调用的 GC 指令时	自定义小部件 → 对话框

编程

针对互动，以下定义都是必须的：

对话框设计扩展

定义:	**DEF** *(变量)*	
句法：	**(***(类型)***//5/"","***(变量)***","/wr2/)**	
变量类型：	类型	带任意数据类型（无 W）的标准输入区（无栅格或转换）

参数：	变量	用于数据交换的变量的任意名称
输入模式：	wr2	读和写

示例

```
DEF CUSVAR1 = (R//5/"","CUSVAR1",""/wr2/)
```

类定义扩展

在自定义小部件的类定义中必须创建 QProperty，其名称应与在对话框设计中选择的变量名称一致，例如：

```
Q_PROPERTY(double CUSVAR1 READ cusVar1 WRITE setCusVar1);
```

示例

dll 文件的类定义应为：

```
#define SLESTESTCUSTOMWIDGET_EXPORT Q_DECL_EXPORT

class SLESTESTCUSTOMWIDGET_EXPORT SlEsTestCustomWidget  : public QWidget
{
 Q_OBJECT
 Q_PROPERTY(double CUSVAR1  READ cusVar1    WRITE setCusVar1);
....
....
}
```

7.4.5 自定义小部件与对话框的互动 - 手动数据交换

除自动数据交换外，也可进行手动数据交换。 该交换是动态进行的，即在对话框运行期间进行。 可执行以下操作：

- 可读取和写入自定义小部件的属性。
- 可从 Run MyScreens 配置中调用自定义小部件的方法。
- 可以响应一个特定的自定义小部件信号并借此在 Run MyScreens 配置中调用子程序 (SUB)。

7.4.5.1 读取和写入属性

说明

Run MyScreens 配置中提供了 ReadCWProperties 和 WriteCWProperties 函数用于读取和写入自定义小部件的属性。

编程

句法：	**ReadCWProperty**（*“变量名称”，“属性名称”*）	
说明：	读取自定义小部件的属性	
参数：	变量名称	分配到一个自定义小部件的对话框变量的名称
	属性名称	待读取的自定义小部件属性的名称
返回值：	自定义小部件属性的当前值	

句法：	**WriteCWProperty**（*“变量名称”，“属性名称”，“值”*）	
说明：	写入 CustomWidget 的属性	
参数：	变量名称	分配到一个自定义小部件的对话框变量的名称
	属性名称	待写入的自定义小部件属性的名称
	值	应在 CustomWidget 属性中写入的值

示例

示例 1：

读取包含对话框变量 “MyCWVar1” 的自定义小部件属性 “MyStringVar” 并在寄存器 7 中分配值。

CustomWidget 类声明：

```
class SLESTESTCUSTOMWIDGET_EXPORT SlEsTestCustomWidget  : public QWidget
{
   Q_OBJECT
   Q_PROPERTY(QString MyStringVar
              READ myStringVar
              WRITE setMyStringVar);
   ...
```

对话框配置：

```
DEF MyCWVar1 = (W///,"slestestcustomwidget.SlEsTestCustomWidget")
PRESS(VS1)
   REG[7]=ReadCWProperty("MyCWVar1", "MyStringVar")
END_PRESS
```

示例 2：

在包含对话框变量 “MyCWVar1” 的 CustomWidget 属性 “MyRealVar” 中写入计算结果 “3 + sin(123.456)”。

CustomWidget 类声明：

```
class SLESTESTCUSTOMWIDGET_EXPORT SlEsTestCustomWidget  : public QWidget
{
   Q_OBJECT
   Q_PROPERTY(double MyRealVar
              READ myRealVar
              WRITE setMyRealVar);
   ...
```

对话框配置：

```
DEF MyCWVar1 = (W///,"slestestcustomwidget.SlEsTestCustomWidget")
PRESS(VS1)
   WriteCWProperty("MyCWVar1", "MyRealVar", 3 + sin(123.456))
END_PRESS
```

7.4.5.2 执行自定义小部件的方法

说明

Run MyScreens 配置中提供了 CallCWMethod 函数用于执行自定义小部件的方法。

待调用的自定义小部件方法最多可以有 10 个传输参数。

支持以下传输参数数据格式：

- bool
- uint
- int
- double
- QString
- QByteArray

编程

句法：	**CallCWMethod**（*“变量名称”，“方法名称[，自变量 0][，自变量 1 … [，自变量 9]”*）	
说明：	调用 CustomWidget 方法	
参数：	变量名称	分配到一个自定义小部件的对话框变量的名称
	方法名称	待调用的自定义小部件方法的名称
	自变量 0 - 9	CustomWidget 方法的传输参数 不支持的数据格式：参见前面的 **说明：**传输参数将始终传输 “ByVal”，即始终只传输值，不传输变量。
返回值：	自定义小部件方法的返回值 支持以下传输参数数据格式： • void • bool • uint • int • double • QString • QByteArray **提示：** 即使自定义小部件方法的返回值的数据格式为 “void”，也必须将该值分配给一个变量。	

示例

CustomWidget 类声明：

```
class SLESTESTCUSTOMWIDGET_EXPORT SlEsTestCustomWidget  : public QWidget
{
   Q_OBJECT
   public slots:
       void myFunc1(int nValue, const QString& szString, double dValue);
...
```

对话框配置：

```
DEF MyCWVar1 = (W///,"slestestcustomwidget.SlEsTestCustomWidget")
DEF MyStringVar1 = (S)
DEF MyRealVar = (R)

PRESS(VS3)
   REG[9] = CallCWMethod("MyCWVar1", "myFunc1", 1+7, MyStringVar1, sin(MyRealVar) -
8)
END_PRESS
```

提示

自定义小部件必须执行“serialize”方法。采用该方法可以将自定义小部件的内部数据写入指定的文件中或重新创建。当打开的“”屏幕切换至其他操作区域时然后再次返回至该屏幕时，该方法非常有必要。否则的话，内部数据在重新显示时会丢失。

句法：	public slots: bool **serialize**(const QString& szFilePath, bool bIsStoring);	
说明：	内部数据和状态从文件中读出或写入文件中	
参数：	szFilePath	从其中读出或写入自定义小部件的内部数据和状态、带有完整路径说明的文件名称。 必要时，自定义小部件必须自行创建该文件。
	bIsStoring	TRUE = 写入 FALSE = 读出

示例

```
bool SlEsTestCustomWidget::serialize(const QString& szFilePath, bool bIsStoring)
{
   QFileInfo fi(szFilePath);
   bool bReturn = false;
```

```
bool SlEsTestCustomWidget::serialize(const QString& szFilePath, bool bIsStoring)
   QDir dir;
   if (dir.mkpath(fi.canonicalPath()))
   {
      QFile fileData(szFilePath);
      QIODevice::OpenMode mode;

      if (bIsStoring)
      {
         mode = QIODevice::WriteOnly;
      }
      else
      {
         mode = QIODevice::ReadOnly;
      }

      if (fileData.open(mode))
      {
         QDataStream streamData;
         streamData.setDevice(&fileData);

         if (bIsStoring)
         {
            streamData << m_nDataCount << m_dValueX;
         }
         else
         {
            streamData >> m_nDataCount >> m_dValueX;
         }

         streamData.setDevice(0);
            fileData.flush();
            fileData.close();
         bReturn = true;
      }
   }
   return bReturn;
}
```

7.4.5.3 响应一个自定义小部件信号

说明

在 Run MyScreens 中，可以响应一个特定的自定义小部件信号 (invokeSub()) 并借此调用一个子程序 (SUB)。

传输值（自定义小部件信号 -> SUB）时，有 10 个在配置中可与寄存器 (REG) 进行比较的全局变量（即所谓的 SIGARG）可用。系统会保存通过自定义小部件信号传输的值。

支持以下传输参数数据格式：

- bool
- uint
- int
- double
- QString
- QByteArray

编程

调用子程序：

句法：	`void invokeSub(const QString& rszSignalName, const QVariantList& rvntList);`	
说明：	自定义小部件信号，通过其调用一个 Run MyScreens 子程序。	
参数：	rszSignalName	待调用的 Run MyScreens 子程序名称
	rvntList	QVariantList Array，用于传输保存在全局参数 SIGARG 中的参数，可在配置中使用。 最大数量：10 个元素 支持的数据格式：见上 **提示：**传输参数将始终传输 “ByVal”，即始终只传输值，不传输变量。

待调用的子程序：

句法：	SUB(on_<变量名称>_<信号名称>) … END_SUB	
说明：	响应一个自定义小部件信号	
参数：	变量名称	分配到一个自定义小部件的对话框变量的名称。
	信号名称	自定义小部件信号的名称
	SIGARG 0 - 9	自定义小部件方法的传输参数。 支持的数据格式：见上 **提示：**传输参数将始终传输 “ByVal”，即始终只传输值，不传输变量。

示例

“格式错误”，页 210，章节“针对自定义小部件信号的响应”，章节“示例”

此处格式/首行缩进不符。正确的为：

CustomWidget 类声明：

CustomWidget 类声明：

```
class SLESTESTCUSTOMWIDGET_EXPORT SlEsTestCustomWidget : public QWidget
{
   Q_OBJECT
signals:
void invokeSub(const QString& szSubName, const QVariantList& vntList);
…
}
```

CustomWidget 类：

```
QVariantList vntList;
vntList << 123.456;
emit invokeSub("MySub", vntList);
```

对话框配置：

```
DEF MyCWVar1 = (W///,"slestestcustomwidget.SlEsTestCustomWidget")
SUB(on_MyCWVar1_MySub)
   DEBUG("SUB(on_MyCWVar1_MySub) was called with parameter: """ << SIGARG[0] << """")
END_SUB
```

结果 easyscreen_log.txt:

```
[10:22:40.445] DEBUG: SUB(on_MyCWVar1_MySub) was called with parameter: "123.456"
```

7.5 SlEsGraphCustomWidget

7.5.1 SlEsGraphCustomWidget

通用说明

借助于 SlEsGraphCustomWidget 可以显示支持点（例如：测量值，过程）上的几何对象（点、线、矩形、圆角矩形、椭圆、圆弧、文本）和曲线。

对象建立在一个或多个所谓的轮廓中。这些对象可以单独或组合显示或清空、选择或删除。

可以借助当前所选轮廓的功能添加对象并以该顺序显示。如果要显示特定颜色的对象，则添加前必须设置相应的字符颜色。所有以后添加的对象均显示该字符颜色。除了字符颜色外还能影响字符宽度和字符样式。在封闭式矩形、圆角矩形和椭圆中在添加对象前可以设置填充色。

每种轮廓都能采用不同的程序：

- 所有对象类型正常显示。
- 点可以连接至所谓的多线段上并以曲线/图表的格式显示。
- 点、线或圆弧到 X 轴之间的平面可使用相应的填充色填色。例如可以使用该方法显示车削时的剩余材料。
  如果在该模式下的点显示为多线段，则曲线以下至 X 轴的整个平面填满填充色。点、线和圆弧可以处于任意四象限中。

采用特殊的光标可以“顺序控制”一个或多个轮廓。此时设置光标位于轮廓的首个或最后一个点对象上或从当前光标位置延伸至轮廓内部，向前或向后运动点对象。

需要时可以显示带有单独比例缩放的第二个 Y 轴（右）。该轴通过偏移和系数与首个 Y 轴（左）进行耦合。

通过查找功能可以根据指定的 X 坐标查找先前添加至轮廓中的点，需要时也可以在此设置光标。

也可以通过可设定的尺寸将轮廓设置为环形缓冲器。

借助串行化可将 SlEsGraphCustomWidget 的当前状态以二进制格式保存在文件中，也可重新创建。

SlEsGraphCustomWidget 可通过“Pan”（视图移动）和“Pinch”/“Spread”（视图放大/缩小）进行操作。

通过鼠标操作可以移动（左键+移动键）和缩放（鼠标滚轮）视图。

由于性能原因，无法自动更新显示。根据应用情况可通过调用相应的功能触发更新。

示例

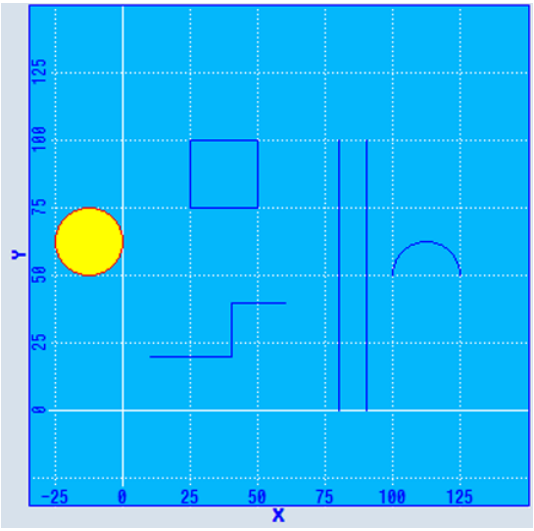


图 7-1 SlEsGraphCustomWidget - 示例

```
//M(MyGraphSampleMask/"SlEsGraphCustomWidget Sample")
DEF MyGraphVar = (W///,"slesgraphcustomwidget.SlEsGraphCustomWidget"/////10,10,340,340/0,0,0,0)
VS1=("Add objects",,se1)
LOAD
   WRITECWPROPERTY("MyGraphVar", "AxisNameX", "X")
   WRITECWPROPERTY("MyGraphVar", "AxisNameY", "Y")
   WRITECWPROPERTY("MyGraphVar", "ScaleTextOrientationYAxis", 2)
   WRITECWPROPERTY("MyGraphVar", "KeepAspectRatio", TRUE)
```

```
//M(MyGraphSampleMask/"SlEsGraphCustomWidget Sample")

   REG[0]= CALLCWMETHOD("MyGraphVar", "addContour", "MyContour", TRUE)
   REG[0]= CALLCWMETHOD("MyGraphVar", "showContour", "MyContour")
   REG[0]= CALLCWMETHOD("MyGraphVar", "setView", -35, -35, 150, 150)
END_LOAD

PRESS(VS1)
   REG[0]= CALLCWMETHOD("MyGraphVar", "addLine", 10, 20, 40, 20)
   REG[0]= CALLCWMETHOD("MyGraphVar", "addLine", 40, 20, 40, 40)
   REG[0]= CALLCWMETHOD("MyGraphVar", "addLine", 40, 40, 60, 40)
   REG[0]= CALLCWMETHOD("MyGraphVar", "addLine", 80, 0, 80, 100)
   REG[0]= CALLCWMETHOD("MyGraphVar", "addLine", 90, 0, 90, 100)
   REG[0]= CALLCWMETHOD("MyGraphVar", "addRect", 25, 100, 50, 75)
   REG[0]= CALLCWMETHOD("MyGraphVar", "setPenColor", "#ff0000");red
   REG[0]= CALLCWMETHOD("MyGraphVar", "setFillColor", "#ffff00");yellow
   REG[0]= CALLCWMETHOD("MyGraphVar", "addCircle", -12.5, 62.5, 12.5)
   REG[0]= CALLCWMETHOD("MyGraphVar", "setFillColor");off/default
   REG[0]= CALLCWMETHOD("MyGraphVar", "setPenColor");off/default
   REG[0]= CALLCWMETHOD("MyGraphVar", "addArc", 100, 37.5, 125, 62.5, 0, 180)
   REG[0]= CALLCWMETHOD("MyGraphVar", "update")
END_PRESS
```

7.5.2 性能说明

根据所用系统的硬件和基本负载情况，在使用 SIEsGraphCustomWidget 时必须注意以下参考值。除了所用硬件和基本负载外，值会根据 SIEsGraphCustomWidget 的配置而变化。

要注意该参考值不能影响到整个系统的响应能力和稳定性。

- 同一时间点能一同配置的 SIEsGraphCustomWidget 数量（例如：最大 1）
- 配置的轮廓数量（例如：最大 6）
- 每个轮廓中可配置的图形数量（例如：最大 1000）
- 要求更新的频率（例如：最大 500 ms）

说明

显示没有实时功能。

7.5.3 读取和写入属性

说明

以下章节中列出的属性可以通过 ReadCWProperty() 功能读取，通过 WriteCWProperty() 功能写入。

示例

读取通过显示变量“MyGraphVar”所连的 SlEsGraphCustomWidget 的属性“CursorX”。结果写入寄存器 0 中。

```
REG[0] = ReadCWProperty("MyGraphVar", "CursorX")
```

将值“MyFirstContour”写入通过显示变量“MyGraphVar”所连的 SlEsGraphCustomWidget 的属性“SelectedContour”中。

```
WriteCWProperty("MyGraphVar ", "SelectedContour",
"MyFirstContour")
```

7.5.4 属性

概述

属性	说明
AxisNameX	轴名称 X 轴
AxisNameY	轴名称 Y 轴
AxisNameY2	轴名称第二 Y 轴（右）
AxisY2Visible	显示/隐藏第二 Y 轴（右）
AxisY2Offset	第二 Y 轴（右）偏移
AxisY2Factor	第二 Y 轴（右）系数
ScaleTextEmbedded	比例缩放文本的定位
ScaleTextOrientationYAxis	Y 轴比例缩放文本校准
KeepAspectRatio	保持页面比例
SelectedContour	当前所选轮廓的名称

属性	说明
BackColor	背景色
ChartBackColor	字符平面的背景色
ForeColor	文本的前景色和缺省字符颜色
ForeColorY2	第二 Y 轴（右）的文本前景色
GridColor	栅格线颜色
GridColorY2	第二 Y 轴（右）水平栅格线的颜色
CursorColor	光标十字线的颜色
ShowCursor	显示/隐藏光标
CursorX	光标 X 坐标
CursorY	光标 Y 坐标
CursorY2	基于第二 Y 轴（右）的光标 Y 位置
CursorStyle	光标显示方式
ViewMoveZoomMode	缩放和移动时的特性

AxisNameX – 轴名称 X 轴

句法：	ReturnValue = ReadCWProperty*(GraphVarName, "***AxisNameX***")* WriteCWProperty*(GraphVarName, "***AxisNameX***", Value)*	
说明：	读取或设置 X 轴的名称。如果没有给出名称，则需充分利用可用的字符平面。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	已读取的属性值（QString）
	Value	待设置的值（QString）

AxisNameY – 轴名称 Y 轴

句法：	ReturnValue = ReadCWProperty*(*GraphVarName, "**AxisNameY**") WriteCWProperty*(GraphVarName, "***AxisNameY***", Value)*
说明：	读取或设置 Y 轴的名称。如果没有给出名称，则需充分利用可用的字符平面。

参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	已读取的属性值（QString）
	Value	待设置的值（QString）

AxisY2Visible - 显示/隐藏第二 Y 轴（右）

句法：	ReturnValue = ReadCWProperty*(GraphVarName, "AxisY2Visible")* WriteCWProperty*(GraphVarName, "AxisY2Visible", Value)*	
说明：	显示/隐藏第二 Y 轴（右）	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	已读取的属性值（bool）
	Value	待设置的值（bool）：TRUE 或 FALSE

AxisY2Offset - 第二 Y 轴（右）偏移

句法：	ReturnValue = ReadCWProperty*(GraphVarName, "AxisY2Offset")* WriteCWProperty*(GraphVarName, "AxisY2Offset", Value)*	
说明：	第二 Y 轴（右）基于第一 Y 轴（左）的偏移。 参见 Property AxisY2Factor 示例。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	已读取的属性值（double）
	Value	待设置的值（double）

AxisY2Factor – 第二 Y 轴（右）系数

句法：	ReturnValue = ReadCWProperty*(GraphVarName, "AxisY2Factor")* WriteCWProperty*(GraphVarName, "AxisY2Factor", Value)*
说明：	第二 Y 轴（右）基于第一 Y 轴（左）的系数。

参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	已读取的属性值（double）
	Value	待设置的值（double）

与“AxisY2Offset”属性一起时可以以单独的缩放比例显示第二 Y 轴（右）。比例通过偏移和系数与首个 Y 轴（左）进行耦合。

Y2 转换为 Y 的公式：

Y = Y2 / 系数 - 偏移

Y 转换为 Y2 的公式：

Y2 = (Y + 偏移) * 系数

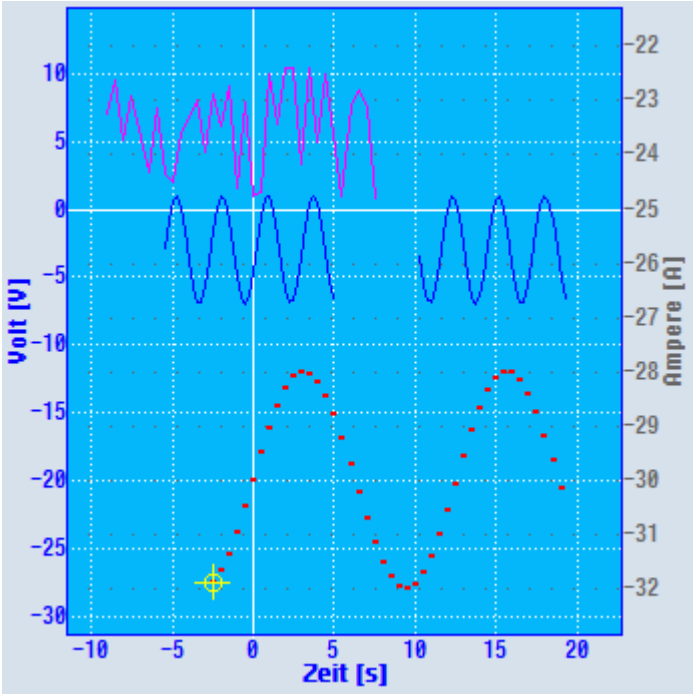


图 7-2　　具有单独比例缩放的第二 Y 轴示例

示例

Y：0 到 200 时，Y2 相应为 -25 到 25。

- 偏移：-100
- 系数：0.25

计算示例：

Y2 = -30

Y = -30 / 0.25 – (-100) = -20

计算示例：

Y = 0

Y2 = (0 + (-100)) * 0.25 = -25

两个坐标系统上的 X 轴都是一样的。

可通过 setForeColorY2() 和 setGridColorY2() 设置颜色（参见颜色）。

通过 setAxisNameY2() 设置轴的标记（参见轴名称）。

ScaleTextEmbedded - 比例缩放文本的定位

句法：	ReturnValue = ReadCWProperty*(GraphVarName, "ScaleTextEmbedded")* WriteCWProperty*(GraphVarName, "ScaleTextEmbedded", Value)*	
说明：	通过该属性可以确定比例缩放文本是定位于字符平面内部还是外部。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	已读取的属性值（bool）
	Value	待设置的值（bool）：TRUE 或 FALSE

示例

字符区外的比例缩放文本：

ScaleTextEmbedded = FALSE

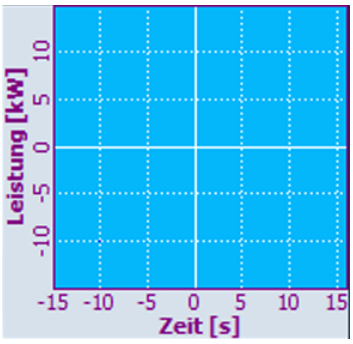


图 7-3　　字符区外的比例缩放文本

字符区内的比例缩放文本：

ScaleTextEmbedded = TRUE

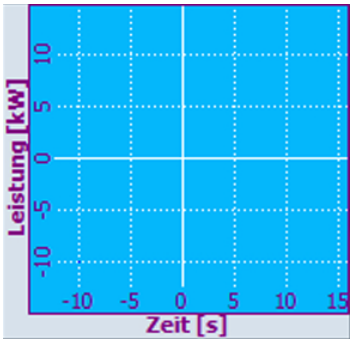


图 7-4　　字符区内的比例缩放文本

ScaleTextOrientationYAxis - Y 轴比例缩放文本校准

句法：	ReturnValue = ReadCWProperty*(GraphVarName,* **'ScaleTextOrientationYAxis'***)* WriteCWProperty*(GraphVarName,* **'ScaleTextOrientationYAxis'***,* *Value)*	
说明：	通过该功能垂直校准 Y 轴的比例缩放文本。	
参数：	GraphVarName	包含 SIEsGraphCustomWidget 的显示变量的名称
	Return Value	已读取的属性值（int）
	Value	待设置的值（int）：1 (= 水平) 或 2 (= 垂直)

示例

字符区内的比例缩放文本：

- ScaleTextEmbedded = TRUE

水平文本校准：

- ScaleTextOrientationYAxis = 1

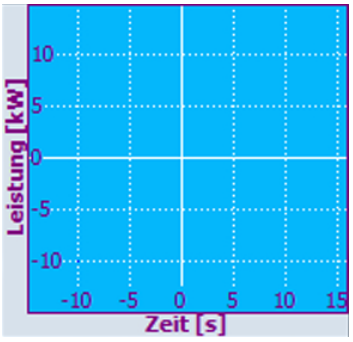


图 7-5　　字符区内的比例缩放文本，水平文本校准

垂直文本校准：

- ScaleTextOrientationYAxis = 2

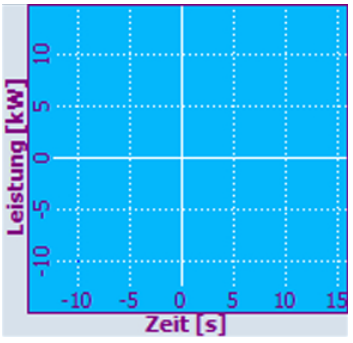


图 7-6　　字符区内的比例缩放文本，垂直文本校准

字符区外的比例缩放文本：

- ScaleTextEmbedded = FALSE

水平文本校准：

- ScaleTextOrientationYAxis = 1

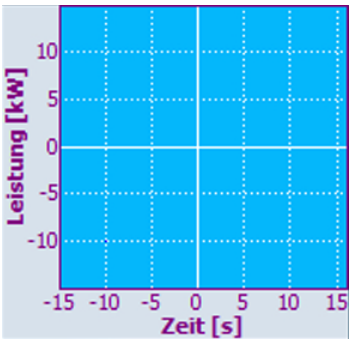


图 7-7　　字符区外的比例缩放文本，水平文本校准

垂直文本校准：

- ScaleTextOrientationYAxis = 2

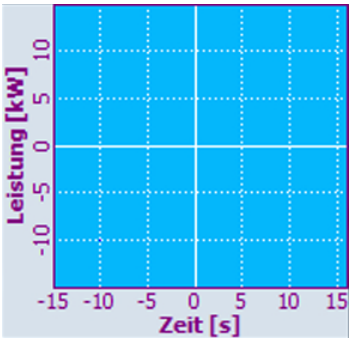


图 7-8　　字符区外的比例缩放文本，垂直文本校准

KeepAspectRatio – 保持页面比例

句法：	ReturnValue = ReadCWProperty*(GraphVarName,* **"KeepAspectRatio"***)* WriteCWProperty*(GraphVarName,* **"KeepAspectRatio"***, Value)*
说明：	通过该属性确定，由 setView() 确定的视图是否自动校准、调整或扩展，X 轴与 Y 轴的页面比例是否一直相同。该特性在显示几何数据时尤其重要。这样一来，圆弧、正方形以及其他和椭圆或矩形便不会扭曲显示。

参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	已读取的属性值（bool）
	Return Value	待设置的值（bool）：TRUE 或 FALSE

示例

```
setView(-15, -15, 15, 15)
setFillColor("#A0A0A4")
addCircle(0, 0, 5)
KeepAspectRatio = TRUE
```

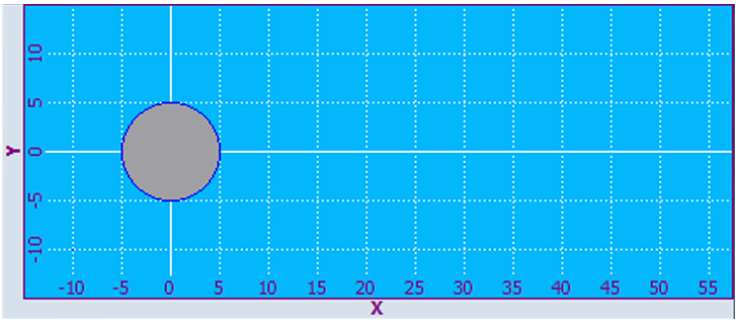


图 7-9　　X 轴与 Y 轴的页面比例保持相同

```
KeepAspectRatio = FALSE
```

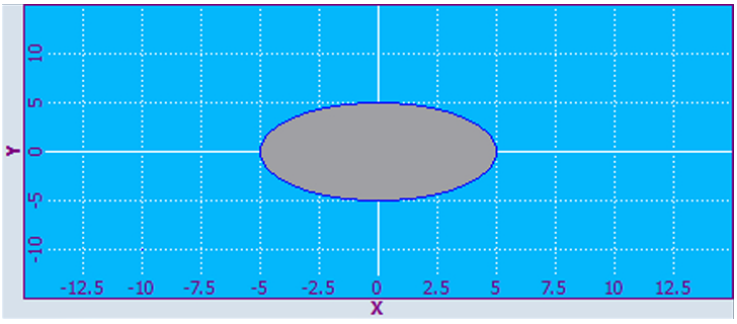


图 7-10　　X 轴与 Y 轴的页面比例扭曲

SelectedContour – 当前所选轮廓的名称

句法：	ReturnValue = ReadCWProperty*(GraphVarName, "*SelectedContour*")* WriteCWProperty*(GraphVarName, "*SelectedContour*", Value)*	
说明：	读取或设置当前轮廓的选择。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	已读取的属性值（QString）
	Value	待设置的值（QString）

说明

为了能在轮廓上进行操作，例如：添加图形对象，必须提前选择。

BackColor – 背景颜色

句法：	ReturnValue = ReadCWProperty*(GraphVarName, "*BackColor*")* WriteCWProperty*(GraphVarName, "*BackColor*", Value)*	
说明：	轴标记的背景颜色，取决于属性 ScaleTextInsideChartArea 和比例缩放文本。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	已读取的属性颜色（QString）
	Value	待设置的值（QString），作为“#RRGGBB”格式的 RGB 值，例如“#04B7FB”

ChartBackColor – 字符平面的背景色

句法：	ReturnValue = ReadCWProperty*(GraphVarName, "*ChartBackColor*")* WriteCWProperty*(GraphVarName, "*ChartBackColor*", Value)*
说明：	字符区的背景颜色。

参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	已读取的属性颜色（QString）
	Value	待设置的值（QString），作为“#RRGGBB”格式的 RGB 值，例如“#04B7FB”

ForeColor - 标记的前景颜色和缺省字符颜色

句法：	ReturnValue = ReadCWProperty*(GraphVarName, "ForeColor")* WriteCWProperty*(GraphVarName, "ForeColor", Value)*	
说明：	文本的前景颜色和缺省字符颜色	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	已读取的属性颜色（QString）
	Value	待设置的值（QString），作为“#RRGGBB”格式的 RGB 值，例如“#04B7FB”

ForeColorY2 - 第二 Y 轴（右）的标记前景色

句法：	ReturnValue = ReadCWProperty*(GraphVarName, "ForeColorY2")* WriteCWProperty*(GraphVarName, "ForeColorY2", Value)*	
说明：	第二 Y 轴（右）的文本前景色。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	已读取的属性颜色（QString）
	Value	待设置的值（QString），作为“#RRGGBB”格式的 RGB 值，例如“#04B7FB”

GridColor - 栅格线颜色

句法：	ReturnValue = ReadCWProperty*(GraphVarName, "GridColor")* WriteCWProperty*(GraphVarName, "GridColor", Value)*
说明：	栅格线颜色。

参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	已读取的属性颜色（QString）
	Value	待设置的值（QString），作为“#RRGGBB”格式的 RGB 值，例如“#04B7FB”

GridColorY2 - 第二 Y 轴（右）水平栅格线的颜色

句法：	ReturnValue = ReadCWProperty*(GraphVarName, "GridColorY2")* WriteCWProperty*(GraphVarName, "GridColorY2", Value)*	
说明：	第二 Y 轴（右）水平栅格线的颜色	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	已读取的属性颜色（QString）
	Value	待设置的值（QString），作为“#RRGGBB”格式的 RGB 值，例如“#04B7FB”

CursorColor - 光标颜色

句法：	ReturnValue = ReadCWProperty*(GraphVarName, "CursorColor")* WriteCWProperty*(GraphVarName, "CursorColor", Value)*	
说明：	光标颜色。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	已读取的属性颜色（QString）
	Value	待设置的值（QString），作为“#RRGGBB”格式的 RGB 值，例如“#04B7FB”

ShowCursor – 显示/隐藏光标

句法：	ReturnValue = ReadCWProperty*(GraphVarName, "ShowCursor")* WriteCWProperty*(GraphVarName, "ShowCursor", Value)*
说明：	显示/隐藏光标。

参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	已读取的属性值（bool）
	Value	待设置的值（bool）：TRUE 或 FALSE

说明

该功能会自动更新显示。

CursorX – 光标 X 坐标

句法：	ReturnValue = ReadCWProperty*(GraphVarName, "CursorX")* WriteCWProperty*(GraphVarName, "CursorX", Value)*	
说明：	光标的 X 坐标。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	已读取的属性值（double）
	Value	待设置的值（double）

说明

该功能会自动更新显示。

CursorY – 光标 Y 坐标

句法：	ReturnValue = ReadCWProperty*(GraphVarName, "CursorY")* WriteCWProperty*(GraphVarName, "CursorY", Value)*	
说明：	光标的 Y 位置。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	已读取的属性值（double）
	Value	待设置的值（double）

说明

该功能会自动更新显示。

CursorY2 – 基于第二 Y 轴（右）的光标 Y 位置

句法：	ReturnValue = ReadCWProperty*(GraphVarName, "CursorY2")* WriteCWProperty*(GraphVarName, "CursorY2", Value)*	
说明：	基于第二 Y 轴（右）的光标的 Y 位置。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	已读取的属性值（double）
	Value	待设置的值（double）

说明

该功能会自动更新显示。

CursorStyle – 光标显示方式

句法：	ReturnValue = ReadCWProperty*(GraphVarName, "CursorStyle")* WriteCWProperty*(GraphVarName, "CursorStyle", Value)*	
说明：	光标显示方式。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	已读取的属性值（int）
	Value	待设置的值（int）： 0 (= 十字) 1 (= 十字线) 2 (= 垂直线) 3 (= 水平线) 4 (= 水平和垂直线))

ViewMoveZoomMode – 用手势缩放和移动时的特性

句法：	ReturnValue = ReadCWProperty*(GraphVarName, "ViewMoveZoomMode")* WriteCWProperty*(GraphVarName, "ViewMoveZoomMode", Value)*	
说明：	可通过手势修改所显示的 SlEsGraphCustomWidget 视图。通过该属性确定允许哪种方式。例如可在特定应用情况下确定仅允许水平移动或缩放视图（测量值过程显示时）。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	已读取的属性值（int）
	Value	待设置的值（int）： 0 (= 关闭) 1 (= 仅水平) 2 (= 仅垂直) 3 (= 水平和垂直)

7.5.5 功能

可通过 CallCWMethod() 调用以下功能。

示例

设置通过显示变量“MyGraphVar”连接的 SlEsGraphCustomWidget 的视图。

结果写入寄存器 0 中。

```
REG[0] = CallCWMethod("MyGraphVar", "setView", -8, -5, 11- 20)
```

概述

功能	说明
setView	设置坐标系
setMaxContourObjects	设置轮廓的最大对象数量（环形缓冲器）
addContour	添加轮廓
showContour	显示轮廓

功能	说明
hideContour	隐藏轮廓
hideAllContours	隐藏所有轮廓
removeContour	删除轮廓
clearContour	删除轮廓中的图形对象
fitViewToContours	视图自动适配
fitViewToContour	视图自动适配
findX	在轮廓中查找
setPolylineMode	显示轮廓的点为多线段/曲线
setIntegralFillMode	显示轮廓的点为多线段/曲线
repaint, update	更新视图
addPoint	添加轮廓中的点
addLine	添加轮廓中的线
addRect	添加轮廓中的矩形
addRoundedRect	添加轮廓中的圆角矩形
addEllipse	添加轮廓中的椭圆
addCircle	添加轮廓中的圆
addArc	添加轮廓中的圆弧
addText	添加轮廓中的文本
setPenWidth	设置字符宽度
setPenStyle	设置字符样式
setPenColor	设置字符颜色
setFillColor	设置填充色
setCursorPosition	光标定位
setCursorPositionY2	基于第二 Y 轴（右）定位光标
setCursorOnContour	定位轮廓上的光标
moveCursorOnContourBegin	定位轮廓上第一个图形对象的光标
moveCursorOnContourEnd	定位轮廓上最后一个图形对象的光标
moveCursorOnContourNext	定位轮廓上下一个图形对象的光标
serialize	保存/恢复当前状态

setView – 设置坐标系

句法：	ReturnValue = CallCWMethod*(GraphVarName, "**setView**", x1, y1, x2, y2)*	
说明：	通过该功能确定坐标系的大小，在该坐标系要在 SlEsGraphCustomWidget 内显示。另见 KeepAspectRatio 属性。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	错误代码（bool）：TRUE = 成功
	x1	左边沿（double）
	y1	上边沿（double）
	x2	右边沿（double）
	y2	下边沿（double）

说明

该功能会自动更新显示。

示例

```
setView(-8, -5, 11, 20)
AxisNameX = "X"
AxisNameY = "Y"
ScaleTextOrientationYAxis = 1
ScaleTextEmbedded = false
KeepAspectRatio = false
update
```

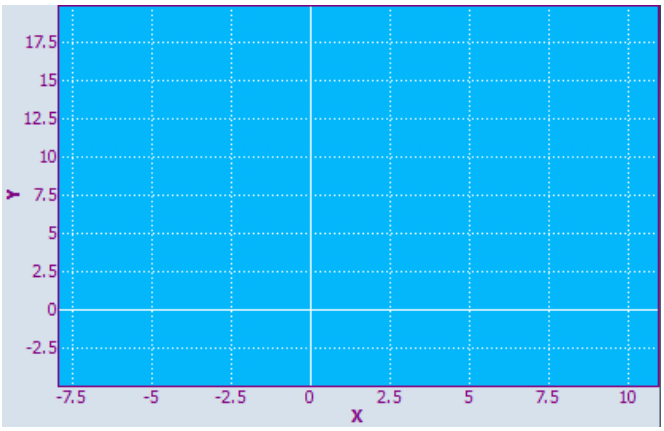


图 7-11　　示例 - setView

setMaxContourObjects - 设置轮廓的最大对象数量（环形缓冲器）

句法：	ReturnValue = CallCWMethod*(GraphVarName,* **"setMaxContourObjects"***, Value)* ReturnValue = CallCWMethod*(GraphVarName,* **"setMaxContourObjects"***, Value, ContourName)*	
说明：	通过该功能确定轮廓环形缓冲器的大小。如果轮廓添加的对象超出了该限值，则会删除最旧的对象。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	错误代码（bool）：TRUE = 成功
	Value	图形对象的最大数目（unit）
	ContourName	轮廓名称（QString）。 如果没有指定轮廓，则自动调用当前所选的轮廓。

说明

该功能会自动更新显示。

addContour – 添加轮廓

句法：	ReturnValue = CallCWMethod*(GraphVarName, "addContour", ContourName)* ReturnValue = CallCWMethod*(GraphVarName, "addContour", ContourName, Selected)* ReturnValue = CallCWMethod*(GraphVarName, "addContour", ContourName, Selected, AssignedY2)*	
说明：	使用该功能创建新轮廓。 或者也可以分配第二 Y 轴（右）的坐标系轮廓。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	错误代码（bool）：TRUE = 成功
	ContourName	轮廓名称（QString）。 如果没有指定轮廓，则自动调用当前所选的轮廓。
	Selected	选择轮廓（bool）：TRUE 或 FALSE
	AssignedY2	应分配至第二 Y 轴（右）的轮廓 （bool）：TRUE 或 FALSE

showContour – 显示轮廓

句法：	ReturnValue = CallCWMethod*(GraphVarName, "showContour")* ReturnValue = CallCWMethod*(GraphVarName, "showContour", ContourName)*	
说明：	该功能用于显示轮廓。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	错误代码（bool）：TRUE = 成功
	ContourName	轮廓名称（QString）。 如果没有指定轮廓，则自动调用当前所选的轮廓。

hideContour – 隐藏轮廓

句法：	ReturnValue = CallCWMethod(*GraphVarName, "hideContour")* ReturnValue = CallCWMethod*(GraphVarName, "hideContour", ContourName)*	
说明：	该功能用于隐藏轮廓。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	错误代码（bool）：TRUE = 成功
	ContourName	轮廓名称（QString）。 如果没有指定轮廓，则自动调用当前所选的轮廓。

showAllContours - 显示所有轮廓

句法：	ReturnValue = CallCWMethod*(GraphVarName, "showAllContours")*	
说明：	该功能用于显示所有轮廓。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	错误代码（bool）：TRUE = 成功

hideAllContours – 隐藏所有轮廓

句法：	ReturnValue = CallCWMethod*(GraphVarName, "hideAllContours")*	
说明：	该功能用于隐藏所有轮廓。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	错误代码（bool）：TRUE = 成功

removeContour – 隐藏轮廓

句法：	ReturnValue = CallCWMethod*(GraphVarName, "removeContour")* ReturnValue = CallCWMethod*(GraphVarName, "removeContour", ContourName)*
说明：	删除轮廓。

参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	错误代码（bool）：TRUE = 成功
	ContourName	轮廓名称（QString）。 如果没有指定轮廓，则自动调用当前所选的轮廓。

clearContour – 删除轮廓中的图形对象

句法：	ReturnValue = CallCWMethod*(GraphVarName, "clearContour")* ReturnValue = CallCWMethod*(GraphVarName, "clearContour", ContourName)*	
说明：	删除轮廓中包含的所有图形对象。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	错误代码（bool）：TRUE = 成功
	ContourName	轮廓名称（QString）。 如果没有指定轮廓，则自动调用当前所选的轮廓。

fitViewToContours – 视图自动适配

句法：	ReturnValue = CallCWMethod*(GraphVarName, "fitViewToContours")* ReturnValue = CallCWMethod*(GraphVarName, "fitViewToContours", OnlyVisible)*	
说明：	通过该功能视图自动与轮廓进行适配。根据传输参数确定，是否只显示可显示的轮廓还是所有轮廓。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	错误代码（bool）：TRUE = 成功
	OnlyVisible	待设置的值（bool）：TRUE 或 FALSE

fitViewToContour – 视图自动适配

句法：	ReturnValue = CallCWMethod*(GraphVarName, "fitViewToContour", ContourName)*	
说明：	通过该功能视图自动适配，仅显示一个特定的轮廓。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	错误代码（bool）：TRUE = 成功
	ContourName	轮廓名称（QString）

findX – 在轮廓中查找

句法：	ReturnValue = CallCWMethod*(GraphVarName, "findX", x)* ReturnValue = CallCWMethod*(GraphVarName, "findX", x, ContourName)* ReturnValue = CallCWMethod(*GraphVarName, "findX", x, ContourName, SetCursor)*	
说明：	通过该功能可在轮廓中通过确定的 X 坐标查找之前定义的点。获得 Y 坐标作为结果。或者也可以在该点上同时定位光标。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	Errorcode (QString):成功时提供相应的 Y 值
	x	待查找的 X 值（double）
	ContourName	轮廓名称（QString）。 如果没有指定轮廓，则自动调用当前所选的轮廓。
	SetCursor	设置光标（bool）：TRUE 或 FALSE

说明

通过该功能设置光标时，显示自动更新。

setPolylineMode – 显示轮廓的点为多线段/曲线

句法：	ReturnValue = CallCWMethod*(GraphVarName, "setPolylineMode", SetPoly)*	
说明：	当前轮廓上所有在该功能调用后添加的点连接至多线段/曲线。该功能是轮廓专用的可逐段打开和关闭。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	错误代码（bool）：TRUE = 成功
	SetPoly	PolylineMode (bool):TRUE = 开

示例

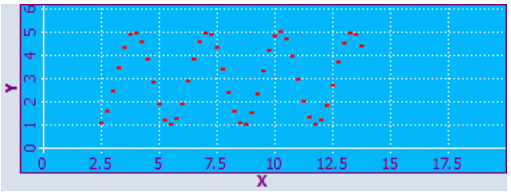


图 7-12　　示例 - PolylineMode:关

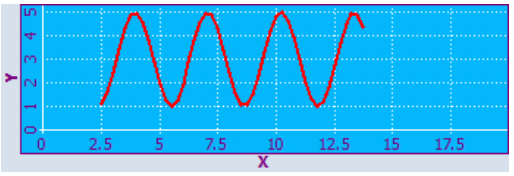


图 7-13　　示例 - PolylineMode:开

setIntegralFillMode – 给点、线和圆弧及 X 轴之间的平面填色

句法：	ReturnValue = CallCWMethod*(GraphVarName, "setIntegralFillMode", SetIntFill)*	
说明：	点、线和圆弧及 X 轴间的平面用当前填充色进行填充（不管点是 +Y 还是 –Y）。 该功能是轮廓专用的可逐段打开和关闭。	

参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	错误代码（bool）：TRUE = 成功
	SetIntFill	IntegralFillMode (bool):TRUE = 开

示例

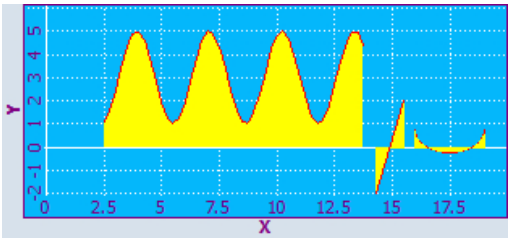


图 7-14　　示例 - setIntegralFillMode:开

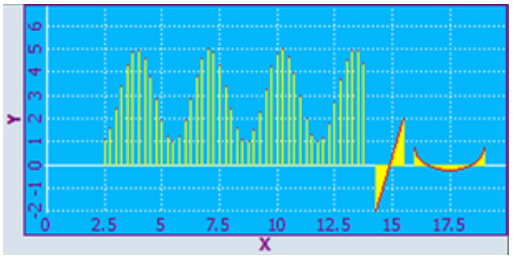


图 7-15　　示例 - setIntegralFillMode:关

repaint, update – 更新视图

句法：	ReturnValue = CallCWMethod*(GraphVarName, "repaint")* ReturnValue = CallCWMethod*(GraphVarName, "update")*	
说明：	通过该功能可以手动更新显示。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	错误代码（bool）：TRUE = 成功

- update()
  该功能可以更新视图，前提条件是未取消小部件的更新且小部件未隐藏。优化该功能，确保保留应用的性能且视图不会因频繁更新而闪烁。
- repaint()
  调用该功能后直接更新视图。因此只有在迫切需要立即更新时（例如：动画）才使用 repaint 功能。

建议始终与 update() 功能一同处理。SlEsGraphCustomWidget 内部始终与 update() 功能一同处理，例如：setView() 上。

addPoint – 添加轮廓中的点

句法：	ReturnValue = CallCWMethod*(GraphVarName, "addPoint", x, y)* ReturnValue = CallCWMethod*(GraphVarName, "addPoint", x, y, DummyPoint)*	
说明：	添加当前所选轮廓上的点。此外还可以设置假点，该点虽然不会显示，但仍可通过光标定位。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	错误代码（bool）：TRUE = 成功
	x	x 坐标（double）
	y	y 坐标（double）
	DummyPoint	不可见的点（bool）:TRUE = 是

addLine – 添加轮廓上的线

句法：	ReturnValue = CallCWMethod*(GraphVarName, "addLine", x1, y1, x2, y2)*
说明：	添加当前所选轮廓上的线

参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	错误代码（bool）：TRUE = 成功
	x1	起点的 x 坐标（double）
	y1	起点的 y 坐标（double）
	x2	终点的 x 坐标（double）
	y2	终点的 y 坐标（double）

addRect – 添加轮廓上的矩形

句法：	ReturnValue = CallCWMethod*(GraphVarName, "**addRect**", x1, y1, x2, y2)*	
说明：	添加当前所选轮廓上的矩形	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	错误代码（bool）：TRUE = 成功
	x1	左边沿（double）
	y1	左边沿（double）
	x2	右边沿（double）
	y2	下边沿（double）

addRoundedRect – 添加轮廓中的圆角矩形

句法：	ReturnValue = CallCWMethod*(GraphVarName, "**addRoundedRect**", x1, y1, x2, y2, r)*
说明：	添加当前所选轮廓上的圆角矩形

参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	错误代码（bool）：TRUE = 成功
	x1	左边沿（double）
	y1	上边沿（double）
	x2	上边沿（double）
	y2	上边沿（double）
	r	半径（double）

addEllipse – 添加轮廓中的椭圆

句法：	ReturnValue = CallCWMethod*(GraphVarName, "addEllipse", x1, y1, x2, y2, radius)*	
说明：	添加当前所选轮廓上的椭圆。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	错误代码（bool）：TRUE = 成功
	x1	左边沿（double）
	y1	上边沿（double）
	x2	右边沿（double）
	y2	下边沿（double）
	半径	半径（double）

addCircle – 添加轮廓中的圆

句法：	ReturnValue = CallCWMethod*(GraphVarName, "addCircle", x, y, radius)*
说明：	添加当前所选轮廓上的圆。

参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	错误代码（bool）：TRUE = 成功
	x	左边沿（double）
	y	上边沿（double）
	半径	半径（double）

addArc – 添加轮廓中的圆弧

句法：	ReturnValue = CallCWMethod*(GraphVarName, "addArc", x1, y1, x2, y2, StartAngle, SpanAngle*	
说明：	添加当前所选轮廓上的圆弧。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	错误代码（bool）：TRUE = 成功
	x1	左边沿（double）
	y1	上边沿（double）
	x2	右边沿（double）
	y2	下边沿（double）
	StartAngle	起始角，单位：度（double）
	SpanAngle	张角，单位：度（double）

addText –添加轮廓中的文本

句法：	ReturnValue = CallCWMethod*(GraphVarName, "addText", x, y, Text)*	
说明：	添加当前所选轮廓上的文本。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	错误代码（bool）：TRUE = 成功
	x	左边沿（double）
	y	上边沿（double）
	Text	文本（QString）

setPenWidth – 设置字符宽度

句法：	ReturnValue = CallCWMethod*(GraphVarName, "*setPenWidth*")* ReturnValue = CallCWMethod*(GraphVarName, "*setPenWidth*", width)*	
说明：	确定从功能调用时间点开始随后在当前轮廓中添加的对象的字符宽度。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	错误代码（bool）：TRUE = 成功
	width	字符宽度（double）。 如果没有指定字符宽度，则设置缺省字符宽度。

setPenStyle – 设置字符样式

句法：	ReturnValue = CallCWMethod*(GraphVarName, "*setPenStyle*")* ReturnValue = CallCWMethod*(GraphVarName, "*setPenStyle*", style)*	
说明：	确定从功能调用时间点开始随后在当前轮廓中添加的对象的字符式样。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	错误代码（bool）：TRUE = 成功
	style	1 :实线 (___) 2 :虚线 (_ _) 3 :点线 (...) 4 :线点线 (_._) 5 :线点点 (_..)

setPenColor – 设置字符颜色

句法：	ReturnValue = CallCWMethod*(GraphVarName, "*setPenColor*")* ReturnValue = CallCWMethod*(GraphVarName, "*setPenColor*", Color)*
说明：	确定从功能调用时间点开始随后在当前轮廓中添加的对象的字符颜色。

参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	错误代码（bool）：TRUE = 成功
	Color	待设置的值（QString），作为“#RRGGBB”格式的 RGB 值，例如“#04B7FB”

setFillColor – 设置填充色

句法：	ReturnValue = CallCWMethod*(GraphVarName, "setFillColor")* ReturnValue = CallCWMethod*(GraphVarName, "setFillColor", Color)*	
说明：	确定从功能调用时间点开始随后在当前轮廓中添加的对象的填充色。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	错误代码（bool）：TRUE = 成功
	Color	待设置的值（QString），作为“#RRGGBB”格式的 RGB 值，例如“#04B7FB”

setCursorPosition – 光标定位

句法：	ReturnValue = CallCWMethod*(GraphVarName, "setCursorPosition", x, y)*	
说明：	在规定的位置上设置光标。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	错误代码（bool）：TRUE = 成功
	x	x 坐标（double）
	y	y 坐标（double）

说明

该功能会自动更新显示。

setCursorPositionY2Cursor – 基于第二 Y 轴（右）定位光标

句法：	ReturnValue = CallCWMethod*(GraphVarName, "setCursorPositionY2", x, y2)*	
说明：	基于第二 Y 轴（右），在规定的位置上设置光标。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	错误代码（bool）：TRUE = 成功
	x	x 坐标（double）
	y2	基于第二 Y 轴（右）的 Y 坐标（double）

说明

该功能会自动更新显示。

setCursorOnContour – 定位轮廓上的光标

句法：	ReturnValue = CallCWMethod*(GraphVarName, "setCursorOnContour")* ReturnValue = CallCWMethod*(GraphVarName, "setCursorOnContour", ContourName)*	
说明：	在轮廓内设置最后光标位置上的光标。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	错误代码（bool）：TRUE = 成功
	ContourName	轮廓名称（QString）。 如果没有指定轮廓，则自动调用当前所选的轮廓。

说明

该功能会自动更新显示。

示例

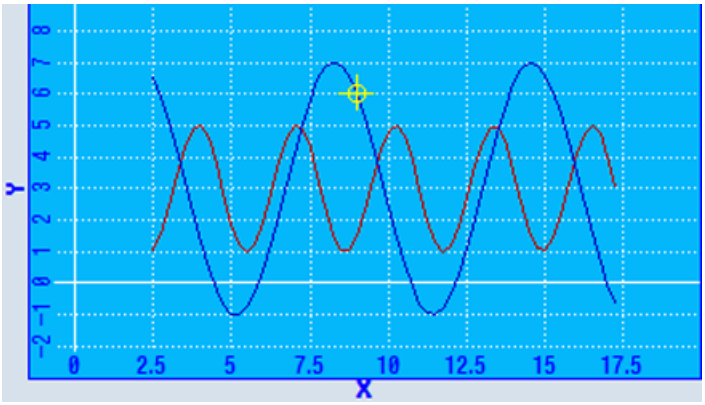


图 7-16　　示例 - setCursorOnContour

moveCursorOnContourBegin – 定位轮廓上第一个图形对象的光标

句法：	ReturnValue = CallCWMethod*(GraphVarName,* **"moveCursorOnContourBegin"***)* ReturnValue = CallCWMethod*(GraphVarName,* **"moveCursorOnContourBegin"***, ContourName)*	
说明：	从轮廓内的当前光标位置开始，可以通过该功能将光标导航至光标的第一个点对象。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	错误代码（bool）：TRUE = 成功
	ContourName	轮廓名称（QString）。 如果没有指定轮廓，则自动调用当前所选的轮廓。

说明

该功能会自动更新显示。

moveCursorOnContourEnd – 定位轮廓上最后一个图形对象的光标

句法：	ReturnValue = CallCWMethod*(GraphVarName, "moveCursorOnContourEnd")* ReturnValue = CallCWMethod*(GraphVarName, "moveCursorOnContourEnd", ContourName)*	
说明：	从轮廓内的当前光标位置开始，可以通过该功能将光标导航至光标的最后一个点对象。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	错误代码（bool）：TRUE = 成功
	ContourName	轮廓名称（QString）。 如果没有指定轮廓，则自动调用当前所选的轮廓。

说明

该功能会自动更新显示。

moveCursorOnContourNext – 定位轮廓上下一个图形对象的光标

句法：	ReturnValue = CallCWMethod*(GraphVarName, "moveCursorOnContourNext")* ReturnValue = CallCWMethod*(GraphVarName, "moveCursorOnContourNext", ContourName)*	
说明：	从轮廓内的当前光标位置开始，可以通过该功能将光标导航至光标的下一个点对象。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	错误代码（bool）：TRUE = 成功
	ContourName	轮廓名称（QString）。 如果没有指定轮廓，则自动调用当前所选的轮廓。

说明

该功能会自动更新显示。

moveCursorOnContourBack – 定位轮廓上前一个图形对象的光标

句法：	ReturnValue = CallCWMethod*(GraphVarName, "moveCursorOnContourBack")* ReturnValue = CallCWMethod*(GraphVarName, "moveCursorOnContourBack", ContourName)*	
说明：	从轮廓内的当前光标位置开始，可以通过该功能将光标导航至光标的前一个点对象。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	错误代码（bool）：TRUE = 成功
	ContourName	轮廓名称（QString）。 如果没有指定轮廓，则自动调用当前所选的轮廓。

说明

该功能会自动更新显示。

serialize – 保存/恢复当前状态

句法：	ReturnValue = CallCWMethod*(GraphVarName, "serialize", FilePath, IsStoring)*	
说明：	通过该功能可以在需要时将 SlEsGraphWidget 的当前状态和内容以二进制格式写入文件或数据流中或恢复。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	Return Value	错误代码（bool）：TRUE = 成功
	FilePath	包含文件名称的完整路径（QString）
	IsStoring	保存/恢复（bool）：TRUE = 保存

说明

该功能会自动更新显示。

7.5.6 信号

以下描述的信号 ViewChanged 可在配置时拦截并作出相应的响应。

概述

功能	说明
ViewChanged	视图变化

ViewChanged – 视图变化

句法：	SUB*(on_<GraphVarName>*_ViewChanged*)* … END_SUB	
说明：	如果视图发生变化，则会发送信号“ViewChanged”。可在配置中以特定的 SUB 方法对此进行响应。相应地设置 SIGARG 参数。	
参数：	GraphVarName	包含 SlEsGraphCustomWidget 的显示变量的名称
	SIGARG[0]	左边沿（double）
	SIGARG[1]	上边沿（double）
	SIGARG[2]	右边沿（double）
	SIGARG[3]	下边沿（double）

示例

```
DEF MyGraphVar = (W///,"slesgraphcustomwidget.SlEsGraphCustomWidget"/////
10,10,340,340/0,0,0,0)
```

```
SUB(on_MyGraphVar_ViewChanged
     DLGL("Current view: " << SIGARG[0] << ", " << SIGARG[1] << ", " <<   SIGARG[2]
<< ", " << SIGARG[3]
END_SUB
```

操作区“Custom（定制）” 8

8.1 这样激活操作区“自定义”

激活操作区“Custom”

“自定义”操作区在交付时未激活。

1. 首先将文件 slamconfig.ini. 从 *[系统西门子目录]*\cfg 目录下复制到 *[系统 oem 目录]*\cfg 下或者复制到相应的 *[系统插件目录]*\cfg 或 *[系统用户目录]*\cfg 下。
2. 为激活操作区“Custom”，必须进行以下输入：
   ```
   [Custom]
   Visible=True
   ```

结果

激活后，操作区“Custom”的软键位于 HSK4 (= 预设置) 扩展菜单栏的主菜单(F10)中。

操作区“Custom”显示为带有可设置标题的空窗口。 所有的水平和垂直软键都是可设置的。

8.2 这样设计“自定义”软键

设计用于操作区“Custom”的软键

在文件 slamconfig.ini 中设计操作区“Custom”软键的标签和位置。

登入软键的设计如下：

1. 为了使用**语言相关的文本**替代软键上的标签，则在[Custom]段中必须有以下输入项：
   ```
   TextId=MY_TEXT_ID
   TextFile=mytextfile
   TextContext=mycontext
   ```
   在该示例中软键显示语言相关的文本，文本 ID 为“MY_TEXT_ID”保存在文本文件 mytextfile_xxx.qm 中的“MyContext”下（xxx 代表语种缩写）。
2. 为了使用**与语言无关的文本**替代软键上的标签，则在[Custom]段中必须有以下输入项：
   ```
   TextId=HELLO
   TextFile=<empty>
   TextContext=<empty>
   ```
   在该示例中操作区“Custom”软键在每种语言中都显示为“HELLO”。

3. 除了文本外，软键上还可以显示**图标**。
   为此在[Custom]段中必须有以下输入项：
   ```
   Picture=mypicture.png
   ```
   这样软键就会显示 mypicture.png 文件中的图标。 图形和位图保存在以下路径下：*[系统 oem 目录]*/ico/ico<分辨率>。 根据显示器的分辨率来使用相应目录下的图片。
4. 此外还可以设置软键的**位置**。 为此在[Custom]段中必须有以下输入项：
   ```
   SoftkeyPosition=12
   ```
   位置 12 是预设置。 这是操作区菜单扩展栏上的 HSK4。 位置 1-8 是菜单栏上的 HSK1 到 HSK8，位置 9-16 是菜单扩展栏上的 HSK1 到 HSK8。

8.3 这样设计操作区"自定义"

设计用于操作区"Custom"的软键

进行操作区设计需要文件 easyscreen.ini 和 custom.ini。两个文件的模板位于目录 *[系统西门子目录]*/templates/cfg 下。

1. 首先将这些文件复制到目录 *[系统 oem 目录]*/cfg 下，再在这里进行修改。
2. 在文件 easyscreen.ini 中已经包含了操作区"自定义"的定义行：
   ```
   ;StartFile02 = area := Custom, dialog := SlEsCustomDialog,
   startfile := custom.com
   ```
   一行开头的"； "代表是注释内容。 这一行即是注释并因此无效。 要变其状态必须删除"； "。
   通过该行中的属性"startfile"来定义，在选择操作区"自定义"时输入项参考设计文件 custom.com。
3. 将 **Projektdatei custom.com** 保存到目录 *[系统 oem 目录]*/proj 下。 其中包括模拟操作区"程序"的 aeditor.com 文件的设计。 所设计的登入软键于是显示在操作区"自定义"中。
4. 在文件 custom.ini 中设计用于对话框标题行的**与语言无关的文本**。
   为此在模板中必须有以下输入项：
   ```
   [Header]
   Text=Custom
   ```
   这些文本可以替换成用户自定义的文本。

5. 为了设计操作区“自定义”的**开始画面**，模板中必须有以下输入项：
   ```
   [Picture]
   Picture=logo.png
   ```
   Logo.png 是在操作区“自定义”的开始对话框中显示的开始画面的名称。 此处可以显示公司徽标或其他图片。 文件保存在相应分辨率的文件夹下：*[系统 oem 目录]*ico/ …
6. 如要在首次显示“自定义”操作区时直接显示某个 “Run MyScreens” 屏幕，则在模板中应存在以下记录：
   ```
   ; Mask shown with startup of area "custom"
   [Startup]
   ;Startup = Mask:=MyCustomStartupMask, File:=mycustommasks.com
   ```
   典型的应用场合，例如在启动 SINUMERIK Operate 时直接以某个 “Run MyScreens” 屏幕启动。
   为此，在配置文件 systemconfiguration.ini 中应按以下方式设置段“[miscellaneous]”关键字“startuparea”：
   ```
   [miscellaneous]
   ;name of the area to be shown at system startup
   startuparea = Custom
   ```

8.4 操作区“自定义”的编程示例

示例

图 8-1 软键“ Start example”示例

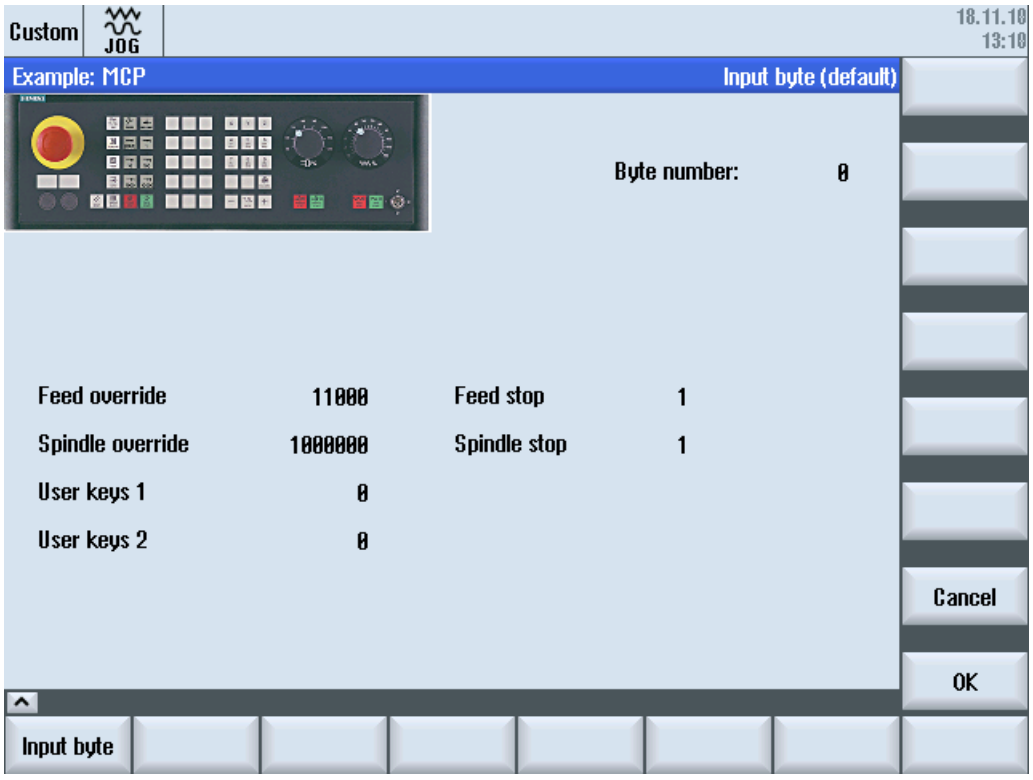


图 8-2　　位图和文本栏示例

文件概览

需要以下文件：

- custom.com
- easyscreen.ini

编程

文件 custom.com 的内容：

说明

本例中链接的图形文件 mcp.png 仅用于示范。 如果希望改编本示例，可以使用自己的图形文件。

```
//S(Start)
   HS7=("Start example", se1, ac7)
```

```
   PRESS(HS7)
      LM("Maske4")
   END_PRESS
//END
//M(Maske4/"Example: MCP"/"mcp.png")
   DEF byte=(I/0/0/"Input byte=0 (default)","Byte number:",""/wr1,li1///380,40,100/480,40,50)
  DEF Feed=(IBB//0/"","Feed override",""/wr1//"EB3"/20,180,100/130,180,100), Axistop=(B//0/"","Feed
stop",""/wr1//"E2.2"/280,180,100/380,180,50/100)
   DEF Spin=(IBB//0/"","Spindle override",""/wr1//"EB0"/20,210,100/130,210,100), spinstop=(B//
0/"","Spindle stop",""/wr1//"E2.4"/280,210,100/380,210,50/100)
   DEF custom1=(IBB//0/"," User keys 1",""/wr1//"EB7.7"/20,240,100/130,240,100)
   DEF custom2=(IBB//0/"","User keys 2",""/wr1//"EB7.5"/20,270,100/130,270,100)
   DEF By1
   DEF By2
   DEF By3
   DEF By6
   DEF By7

   HS1=("Input byte", SE1, AC4)
   HS2=("")
   HS3=("")
   HS4=("")
   HS5=("")
   HS6=("")
   HS7=("")
   HS8=("")
   VS1=("")
   VS2=("")
   VS3=("")
   VS4=("")
   VS5=("")
   VS6=("")
   VS7=("Cancel", SE1, AC7)
   VS8=("OK", SE1, AC7)

   PRESS(VS7)
      EXIT
   END_PRESS

   PRESS(VS8)
      EXIT
   END_PRESS

   LOAD
      By1=1
```

```
      By2=2
      By3=3
      By6=6
      By7=7
   END_LOAD

   PRESS(HS1)
      Byte.wr=2
   END_PRESS

   CHANGE(Byte)
      By1=byte+1
      By2=byte+2
      By3=byte+3
      By6=byte+6
      By7=byte+7
      Feed.VAR="EB"<<By3
      Spin.VAR="EB"<<Byte
      Custom1.VAR="EB"<<By6
      Custom2.VAR="EB"<<By7
      Axisstop.VAR="E"<<By2<<".2"
      Spinstop.VAR="E"<<By2<<".4"
      Byte.wr=1
   END_CHANGE

   CHANGE(Axis stop)
      IF Axistop==0
         Axistop.BC=9
      ELSE
         Axistop.BC=11
      ENDIF
   END_CHANGE

   CHANGE(Spin stop)
      IF Spinstop==0
         Spinstop.BC=9
      ELSE
         Spinstop.BC=11
      ENDIF
   END_CHANGE
//END
```

对话框选择

9

9.1 通过 PLC 软键选择对话框

设计

步骤说明：

- 在 systemconfiguration.ini 中有 [keyconfiguration] 一段。 此记录定义了指定 PLC 软键的动作。
- 设定动作编号，如其大于等于 100 则为 Run MyScreens 调用。
- 在文件 easyscreen.ini 中必须创建一个段用于定义需要执行的动作，其名称由操作区域名称和对话框名称组成（参见 [keyconfiguration] 下的记录 → Area:=..., Dialog:=...）→ [<Area>_<Dialog>] → 例如 [AreaParameter_SlPaDialog]
- 在此段中定义动作编号（已在 systemconfiguration.ini 中设定 → 参见 Action:=...）。 此处涉及两个指令：
  1. LS("Sofkeyleiste1","param.com") ... 装载软键栏
  2. LM("Maske1","param.com") ... 装载屏幕

通过 PLC 软键选择软键栏

在 Run MyScreens 中可通过 PLC 软键选择 Run MyScreens 软键栏和 Run MyScreens 对话框。 为此在设计相关 PLC 软键时候，必须为属性“action”设定大于等于 100 的赋值。

PLC 软键的设计在文件 `systemconfiguration.ini` 中的 [keyconfiguration] 段中进行：

```
[keyconfiguration]
KEY75.1 = Area:=area, Dialog:=dialog, Screen:=screen, Action:=
100,
```

对于相应 PLC 软键，需要执行的 LM 和 LS 指令在文件 easyscreen.ini 中的段中进行，段的名称按照以下规定创建。

[areaname_dialogname]	名称的第一部分“areaname”指示操作区域，第二部分“dialogname”指示对话框，段中设计的指令适用于此对话框。
[AreaParameter_SlPaDialog] 100.screen1 = LS("Softkey1","param.com") 101.screen3 = LM("Maske1","param.com")	必须使用在文件 systemconfiguration.ini 中为操作区域和对话框设定的名称。 对话框的设定是可选项。 特别是对于通过单个对话框执行的操作区域，可省略此设定，见旁边的例子。 如果在通过对话框 SlPaDialog 执行的操作区域 AreaParameter 中显示“screen1”，则在出现值为 100 的“action”时执行指令 "LS("Softkey1","param.com")"。
action.screen=指令	通过两个属性“action”和“screen”将执行指定指令的时间设定为唯一值。 “screen”的设定是可选项。 允许的指令为： LM (LoadMask) LS (LoadSoftkeys)

9.2 通过 PLC 硬键选择对话框

使用范围

PLC 可以启动操作软件中的以下功能：

- 选择操作区域，
- 选择操作区内的某个屏幕
- 执行软键上标注的功能

硬键

所有按键，包括 PLC 按键在下文都称为“硬键”。可以最多定义 254 个硬键，具体可以分为：

按键号	使用
按键 1 - 9	操作面板上的按键
按键 10 - 49	保留
按键 50 - 254 按键 50 - 81	PLC 按键： 预留给 OEM 使用
按键 255	由控制信息预定义。

硬键 1 - 9 的预定义如下：

按键标识		动作/作用
HK1	加工	选择操作区“加工”，回到上一个对话框
HK2	Program	选择操作区“程序”，回到上一个对话框或上一个程序
HK3	Offset	选择操作区“参数”，回到上一个对话框
HK4	Program Manager	选择操作区“程序”，初始画面“程序管理器”
HK5	Alarm	选择操作区“诊断”，对话框“报警列表”
HK6	Custom	操作区“自定义”
HK7 1)	Menu Select	选择“初始菜单”
HK8 1)	功能菜单	选择“功能菜单”
HK9 1)	用户菜单	选择“用户菜单”

1) 只用于 828D

配置

按键的定义在配置文件“systemconfiguration.ini”的章节[keyconfiguration]中进行。每一行定义了一个“硬键事件”。“硬键事件”指第几次按下某个硬键。比方说第二次按下某个硬键和第三次按下该硬键产生的动作就不一样。

配置文件“systemconfiguration.ini”中的条目可以载入用户自定义的设置，可以使用目录 *[系统用户目录]*/cfg 和 [系统 oem 目录]/cfg。

配置文件中用于定义硬键事件的行的结构为：

```
KEYx.n = area:=area, dialog:=dialog, screen:=screen,
forms:=form, menus:=menu,
action:=menu.action, cmdline:=cmdline
KEYx.n = area:=area, dialog:=dialog, cmdline:=cmdline, action:=
action
```

x:硬键的编号，值域：1 – 254

n:事件号 - 相当于第几次按下硬键，值域：0 – 9

前提条件

PLC 用户程序必须首先符合以下前提条件：

每次总是只执行一个硬键功能。因此只有当操作软件应答了前面的一个请求后，才可以处理新的请求。当 PLC 用户程序将 MCP 按键转成硬键时，必须有足够大的中间缓存，这样即使在快速操作中也不会丢失某个按键事件。

PLC 接口

在 PLC 接口上对用于选择硬键的范围进行了规定。该范围定义在 DB19.DBB10 中。在该范围中，PLC 可以直接指定 50 到 254 之间的按键值。

操作软件分两步应答硬键事件。这样操作软件便可以将两个连续的相同按键代码看成两个单独的事件进行处理。在第一步中，控制信息 255 被写入到字节 DB19.DBB10 中。借助该虚拟的按键操作可以明确地看出 PLC 的每个按键序列。控制信息对于 PLC 程序没有作用，不允许被修改。在第二步中，操作软件删除 DB19.DBB10，真正地向 PLC 发出应答。从此刻起，PLC 用户程序便成功指定了一个新硬键。与此同时，操作软件开始处理新的硬键请求。

示例

配置文件：

```
; PLC 硬键(KEY50-KEY254)
[keyconfiguration]
KEY50.0 = name := AreaMachine, dialog := SlMachine
KEY51.0 = name := AreaParameter, dialog := SlParameter
KEY52.0 = name := AreaProgramEdit, dialog := SlProgramEdit
```

```
KEY53.0 = name := AreaProgramManager, dialog := SlPmDialog
KEY54.0 = name := AreaDiagnosis, dialog := SlDgDialog
KEY55.0 = name := AreaStartup, dialog := SlSuDialog
KEY56.0 = name := Custom, dialog := SlEsCustomDialog
```

区域和对话框名称请从 *[系统西门子目录]*/cfg 目录下的 systemconfiguration.ini 文件中获取。

9.3 通过 NC 选择对话框

HMI Operate 中的 MMC 指令

可按下述方法使用 MMC 指令：

1. **MMC 指令的定义**
   标准文件 systemconfiguration.ini 中含有以下组合：
   地址:=MCYCLES --> 指令:=LM
   地址:=CYCLES --> 指令:=PICTURE_ON
   该区别用于区分测量循环和其他循环。即：
   - LM 只用于测量循环，PICTURE_ON 只用于非测量循环
   - 新定义的 MMC 指令不允许称为 PICTURE_ON 和 LM
2. **“Run MyScreens” 授权**
   所有 “Run MyScreen” 打开的对话框均需获得 “Run MyScreens” 授权，若无授权，只能使用有限数量的对话框。
   使用 test.com (Run MyScreens) 的调用示例：

```
g0 f50
MMC("CYCLES,PICTURE_ON,test.com,maske1","A")
m0
MMC("CYCLES,PICTURE_OFF","N")
M30
```

Referenz - Listen

A

A.1 登入软键表

A.1.1 车床登入软键表

车床的程序操作区

HSK1	HSK2	HSK3	HSK4	HSK5	HSK6	HSK7	HSK8
编辑	**钻削**	**车削**	**车削轮廓**	**铣削**	**其它**	**模拟**	**NC 选择**
HSK9	HSK10	HSK11	HSK12	HSK13	HSK14	HSK15	HSK16
--	直线 圆弧 （仅限 ShopTurn 程序）	--	--	测量工件	测量刀具	**OEM**	--

车削

在下表中列出了车削工艺中可能的登入软键。 各登入软键的分配根据不同的系统可能会有所不同。 给定的 OEM 软键允许用于 Run MyScreens。

programGUIDE（G 代码）登入软键：

	钻削	**车削**	**车削轮廓**		**铣削**		**其它**	
	HSK2	**HSK3**	**HSK4**		**HSK5**		**HSK6**	
VSK1	钻中心孔	轮廓车削	轮廓	--	端面铣削	轮廓	设置	高速设定
VSK2	钻削铰孔	凹槽	轮廓车削	--	型腔	轨迹	平面回转	平行轴
VSK3	深孔钻削	退刀槽	余料车削	--	多边形凸台	预钻削	刀具回转	--
VSK4	镗孔	螺纹	槽式车削	--	槽	型腔	--	--

VSK5	螺纹	切断	余料槽式车削	--	螺纹铣削	型腔余料	--	--
VSK6	**OEM**	--	往复车削	--	雕刻	凸台	子程序	--
VSK7	位置	**OEM**	余料往复车削	**OEM**	**OEM**	凸台余料	--	**OEM**
VSK8	重复位置	--	**>>**	**<<**	轮廓铣削	**<<**	**>>**	**<<**

ShopTurn 登入软键:

	钻削	**车削**	**车削轮廓**		**铣削**		**其它**		
	HSK2	**HSK3**	**HSK4**		**HSK5**		**HSK6**		**HSK10**
VSK1	中心钻孔	轮廓车削	新建轮廓	--	端面铣削	新建轮廓	设置	高速设定	刀具
VSK2	钻中心孔	凹槽	轮廓车削	--	型腔	轨迹	平面回转	平行轴	直线
VSK3	钻削铰孔	退刀槽	余料车削	--	多边形凸台	预钻削	刀具回转	重复程序	圆心
VSK4	深孔钻削	螺纹	槽式车削	--	槽	型腔	副主轴	--	圆半径
VSK5	螺纹	切断	余料槽式车削	--	螺纹铣削	型腔余料	转换	--	极坐标
VSK6	**OEM**	--	往复车削	--	雕刻	凸台	子程序	--	逼近/返回
VSK7	位置	**OEM**	余料往复车削	**OEM**	**OEM**	凸台余料	--	**OEM**	--
VSK8	重复位置	--	**>>**	**<<**	轮廓铣削	**<<**	**>>**	**<<**	--

参见

定义登入软键 (页 30)

A.1.2 铣床登入软键表

铣床的程序操作区

HSK1	HSK2	HSK3	HSK4	HSK5	HSK6	HSK7	HSK8
编辑	**钻削**	**铣削**	**轮廓铣削**	**车削**（仅限 G 代码程序）	**其它**	**模拟**	**NC 选择**
HSK9	HSK10	HSK11	HSK12	HSK13	HSK14	HSK15	HSK16
--	直线 圆弧 （仅限 ShopMill 程序）	--	--	测量工件	测量刀具	**OEM**	--

铣削

在下表中列出了铣削工艺中可能的登入软键。 各登入软键的分配根据不同的系统可能会有所不同。 给定的 OEM 软键允许用于 Run MyScreens。

programGUIDE（G 代码）登入软键：

	钻削	**铣削**	**轮廓铣削**		**车削**		**其它**	
	HSK2	**HSK3**	**HSK4**		**HSK5**		**HSK6**	
VSK1	钻中心孔	端面铣削	轮廓	--	轮廓车削	轮廓	设置	--
VSK2	钻削铰孔	型腔	轨迹	--	凹槽	轮廓车削	平面回转	平行轴
VSK3	深孔钻削	多边形凸台	预钻削	--	退刀槽	余料车削	刀具回转	--
VSK4	镗孔	槽	型腔	--	螺纹	槽式车削	高速设定	--
VSK5	螺纹	螺纹铣削	型腔余料	--	切断	余料槽式车削	--	--
VSK6	**OEM**	雕刻	凸台	--	--	往复车削	子程序	--
VSK7	位置	**OEM**	凸台余料	**OEM**	**OEM**	余料往复车削	--	**OEM**
VSK8	重复位置	--	**>>**	**<<**	车削轮廓	**<<**	**>>**	**<<**

ShopMill 登入软键:

	钻削	**铣削**	**轮廓铣削**		**其它**		**直线圆弧**
	HSK2	**HSK3**	**HSK4**		**HSK6**		**HSK10**
VSK1	钻中心孔	端面铣削	新建轮廓	--	设置	--	刀具
VSK2	钻削铰孔	型腔	轨迹	--	平面回转	平行轴	直线
VSK3	深孔钻削	多边形凸台	预钻削	--	刀具回转	重复程序	圆心
VSK4	镗孔	槽	型腔	--	高速设定	--	圆半径
VSK5	螺纹	螺纹铣削	型腔余料	--	转换	--	螺旋线
VSK6	**OEM**	雕刻	凸台	--	子程序	--	极坐标
VSK7	位置	**OEM**	凸台余料	**OEM**	--	**OEM**	--
VSK8	重复位置	--	**>>**	**<<**	**>>**	**<<**	--

A.2 访问等级列表

不同的访问等级有以下不同的含义：

保护等级	**禁用密码**	**区域**
ac0	预留用于西门子	
ac1	密码	机床制造商
ac2	密码	维修
ac3	密码	用户
ac4	钥匙开关位置 3	编程员，调试员
ac5	钥匙开关位置 2	合格的操作员
ac6	钥匙开关位置 1	受过培训的操作员
ac7	钥匙开关位置 0	学过相关内容的操作员

A.3 颜色表

系统颜色

提供统一的颜色表用于对话框设计（各个标准颜色的色块）。对于单元（文本、输入栏、背景等等）可以从 0 到 133 号的颜色中选择一种。

除了预定义的颜色，您也可以使用 RGB 值(“#RRGGBB”)来给定颜色。

索引	图标	颜色	颜色描述
1		黑色	
2		桔黄色	
3		深绿色	
4		浅灰	
5		深蓝色	
6		蓝色	
7		红色	
8		棕色	
9		黄色	
10		白色	
126		黑色	当前处于焦点下的输入/输出栏的文本颜色
127		浅橙色	当前处于焦点下的输入/输出栏的背景色
128		桔黄色	聚焦系统颜色
129		浅灰	背景颜色
130		蓝色	标题颜色（激活）
131		黑色	标题字体颜色（激活）
132		蓝绿色	转换栏的背景色
133		淡蓝色	列表框的背景色

A.4 文件名中的语种缩写表

支持的语言

标准语言：

语言	文件名中的缩写
简体中文	chs
德语	deu
英语	eng
法语	fra
意大利语	ita
西班牙语	esp

其他语言：

语言	文件名中的缩写
繁体中文	cht
丹麦语	dan
芬兰语	fin
印度尼西亚语	ind
日语	jpn
韩语	kor
马来西亚语	msl
荷兰语	nld
波兰语	plk
葡萄牙语（巴西）	ptb
罗马尼亚语	rom
俄语	rus
瑞典语	sve
斯洛伐克语	sky
斯洛文尼亚语	slv
泰语	tha
捷克语	csy

语言	文件名中的缩写
土耳其语	trk
匈牙利语	hun
越南语	vit

A.5 可用的系统变量列表

文档

参数手册 系统变量 /PGAsl/

A.6 对话框打开方式（属性 CB）

下表列出了在哪些条件下调用 CHANGE 方法。

属性 CB 的适用条件：

CB0

如果变量当前有一个有效值（例如：通过预设值或 NC/PLC 变量），则通过显示屏幕触发 CHANGE 方法。

CB1（默认）

通过显示屏幕无法显式触发 CHANGE 方法。如果变量有一个配置的 NC/PLC 变量，则一定会调用 CHANGE 方法。

条件				响应
类型	系统或者用户变量	预设值	编辑 CHANGE 方法	
输入/输出栏	是	是	是	有了配置的 NC/PLC 变量，系统至少会通过当前 NC/PLC 变量值自动调用一次 CHANGE 方法。此时，CB 的预设值没有用。
			否	
		否	是	
			否	
	否	是	是	通过预设值调用 CHANGE 方法。
			否	不调用 CHANGE 方法。
		否	是	由于当前没有可以用来调用 CHANGE 方法的有效值，因此不调用。
			否	
转换	是	是	是	有了配置的 NC/PLC 变量，系统至少会通过当前 NC/PLC 变量值自动调用一次 CHANGE 方法。此时，CB 的预设值没有用。
			否	
		否	是	
			否	
	否	是	是	通过预设值调用 CHANGE 方法。
			否	不调用 CHANGE 方法。
		否（转换列表中的第一个值默认为预设值）	是	通过转换列表中的第一个值（预设值）来调用 CHANGE 方法。
			否	不调用 CHANGE 方法。

参见

CHANGE (页 111)

提示和技巧

B

B.1 一般提示

- 可视情况将 BTSS 变量用作系统变量($变量)。
  原因：
  这样可避免可能较为费时的名称分析。
  示例：
  最好使用“/通道/参数/R[u1,10]”替代“$R[10]”。
- 避免循环（同类）设置，例如屏幕和变量属性 HLP。 比较之前要设置的值与当前值，不相同时再真正设置该值。
  原因：
  避免在查找与分辨率相关的辅助画面上费时。
  示例：

```
DEF MyVar=(R3///,"X1",,"mm"/WR1//"$AA_IM[0]")
CHANGE(MyVar)
   IF MyVar.VAL < 100
      HLP="mypic1.png"
   ELSE
      HLP="mypic2.png"
   ENDIF
END_CHANGE
```

建议：

```
CHANGE(MyVar)
   IF MyVar.VAL < 100
      IF HLP <> "mypic1.png"
         HLP="mypic1.png"
      ENDIF
   ELSE
      IF HLP <> "smypic2.png"
         HLP="mypic2.png"
```

```
      ENDIF
    ENDIF
  END_CHANGE
```

- 使用一个 MRNP()函数替代多个连续的 RNP()函数。
  使用一个 MRDOP()函数替代多个连续的 RDOP()函数。
  原因：
  降低通讯负载并提升性能。
- 读取驱动参数时不要短于•1•秒的周期，时间长些更佳。
  原因： 否则与驱动的通讯可能会受到很大干扰，甚至导致故障。
- 避免随之发生的与系统或用户变量的计算。 为此请使用，例如寄存器(REG[x])或(不可见)辅助变量。
  原因： 每次赋值都会导致向所连接的系统或用户变量中写入。
- 出于对清晰、可维护和性能（显示屏幕时）的考虑，不同块中所使用的同类代码应放在一个 SUB 块中。 这可在相应位置上使用函数 CALL()进行调用。
- 通过对对话栏（在 slguiconfig.xml 中设置）中 CPU 负载的查看可以确定屏幕上的哪些更改会在多大程度上影响性能。

B.2 DEBUG 的提示

- 函数 DEBUG()用于进行诊断。 调用函数 DEBUG()时返回的字符串会写入 easyscreen_log.txt 文件中。 函数 DLGL()对向对话栏输出信息也会有帮助。
  屏幕开发阶段结束后，出于对性能的考虑应将该函数删除或取消。
- 日志文件 easyscreen_log.txt 在屏幕开发完成后始终为空。

B.3 CHANGE 方法的提示

- CHANGE 方法总是只持续很短的时间，特别是那些与系统或用户变量相连并且频繁变化的变量。
  原因：
  提升屏幕性能。
- 视情况请不要在 CHANGE 方法中设置 RNP()函数。 而是最好与要读取的系统或用户变量一起创建一个不可见的变量并使用。
  原因：
  每次调用都会强制取消 RNP()函数。 否则会直接存取已存在的当前值。

示例：

此处会在轴运行每次变化时通过 RNP()进行名称分析，以读取通道专用的机床数据：

```
DEF AXIS_POSITION_X =  (R///,""///"$AA_IM[X]")

CHANGE(AXIS_POSITION_X)
      DLGL("Axis """ << RNP("$MC_AXCONF_GEOAX_NAME_TAB[0]") <<
""" has moved: "
<< AXIS_POSITION_X)
END_CHANGE
```

借助不可见变量保持通道专用机床数据的最新状态，每个值变化都会复制到临时变量中，例如寄存器。

该临时变量可以在轴位置变化的 CHANGE 方法中使用，无需每次都进行机床数据的名称分析以及相应的读访问：

```
DEF AXIS_POSITION_X =  (R///,""///"$AA_IM[X]")
DEF AXIS_NAME_X = (S///,""/WR0//"$MC_AXCONF_GEOAX_NAME_TAB[0]")

CHANGE(AXIS_NAME_X)
      REG[0] = AXIS_NAME_X
END_CHANGE

CHANGE(AXIS_POSITION_X1)
```

```
   DLGL("Axis """ << REG[0] << """ has moved " <<
AXIS_POSITION_X)
END_CHANGE
```

- 刷新率以及与频繁变化的系统或用户变量相连接的变量的 CHANGE 方法的执行速度会由于变量属性 UR 的使用而降低，例如与轴实际值耦合的变量。
  原因：
  由此，值变化时相应的 CHANGE 方法将在固定设置的光栅中执行。

B.4 DO-LOOP 循环的提示

取决于配置，由于循环会影响 SINUMERIK Operate 的性能，因此请您慎重使用这些循环并且视情况放弃使用循环中费时的动作。

同时应使用寄存器(REG[])作为运行变量，因为一般显示变量（特别是那些使用 BTSS 连接的）会极其频繁地刷新或写入，这同样会影响系统性能

请注意，程序块中执行的脚本行的数量目前限制为 10000 行。 如果达到该数量，文件 easyscreen_log.txt 中的相应记录会中断脚本执行。

原因：

- 防止无限循环
- “Run MyScreens” 必须保持可操作

动画元素

C

C.1 简介

简介

本手册介绍的是通过 X3D 浏览器进行作业，借助该浏览器可将动态和静态图形场景（动画元素）- 以下称作辅助画面 - 集成到 SINUMERIK Operate（V4.7 SP1 起）的图形界面中。

您可在上下文关联的辅助画面中显示目标运动过程和参数。 这样可以提升应用的可操作性并设计相应的用户界面。

从实施理念到完成辅助画面共包含以下几步：

- 创建用于最后可显示在 X3D 浏览器中的辅助画面的图形元素和 3D 模型（参见章节 建模 (页 272)）。
- 创建场景说明文件，在该文件中图形文件的模型数据被分配给待显示的场景和动画（参见章节 场景说明文件的结构 (页 279)）。
- 将 X3D 和 XML 文件转换为 HMI 文件（参见章节 转换为 hmi 文件 (页 284)）。
- 将 X3D 浏览器以 C++ 语言格式集成到自身应用中（参见章节 执行示例 (页 286)）。
- 在 Run MyScreens 中使用（参见章节 在 Run MyScreens 中显示 (页 287)）。

文件格式一览

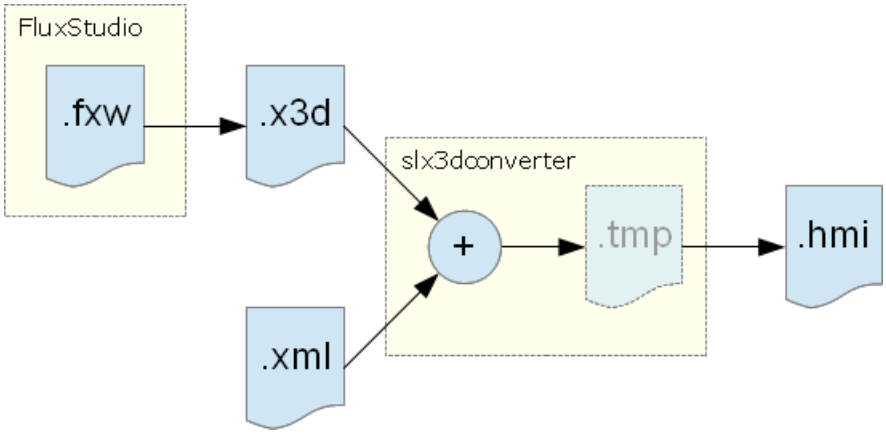


图 C-1 文件格式

文件格式	说明
.fxw	Flux Studio 的图形文件 (3D)
.x3d	交换格式的模型文件
.xml	动画和场景的定义文件
.hmi	X3D 浏览器的输出文件

C.2 建模

C.2.1 前提条件

建模的目的在于创建一个含生成的 3D 模型（文件格式为 .x3d）和一个 3D 内容官方标准的文件。

建模在 **Flux Studio Web3D Authoring Tool** 中进行。 Flux Studio Web3D Authoring Tool 是一个可以提供多种导出和导入方式的免费 3D 建模工具。

前提条件

- 您已安装了 Flux Studio Web3D Authoring Tool V 2.1。
- 知道如何进行建模和使用 Flux Studio。
- 本说明中没有对文件格式 .x3d 作进一步说明，需提前作相应的了解。

说明

您可通过以下链接下载 Flux Studio Web3D Authoring Tool。

http://mediamachines.wordpress.com/flux-player-and-flux-studio (http://mediamachines.wordpress.com/flux-player-and-flux-studio)

C.2.2 建模规则

建模时请注意以下规则。 之后处理辅助画面文件时需遵循这些规则。

建模规则

1. TimeSensor 的名称必须是“动画”。
2. 所有配套元素都必须被分配到一个组中。
3. 所有组都必须设为以下位置：
   x = 0，y = 0，z = 0
4. 如需隐藏组，则须按如下方式设置翻译值：
   x = 1000000，y = 1000000，z = 1000000
5. 下列元素在 Flux Studio 图形模型和 XML 场景定义（对比 一览 (页 279) 一章）中的名称必须相同：
   – 刀具
   – 摄像机
6. 如需创建一个 .x3d 文件，则须在对话框“导出选项”中进行以下设置：

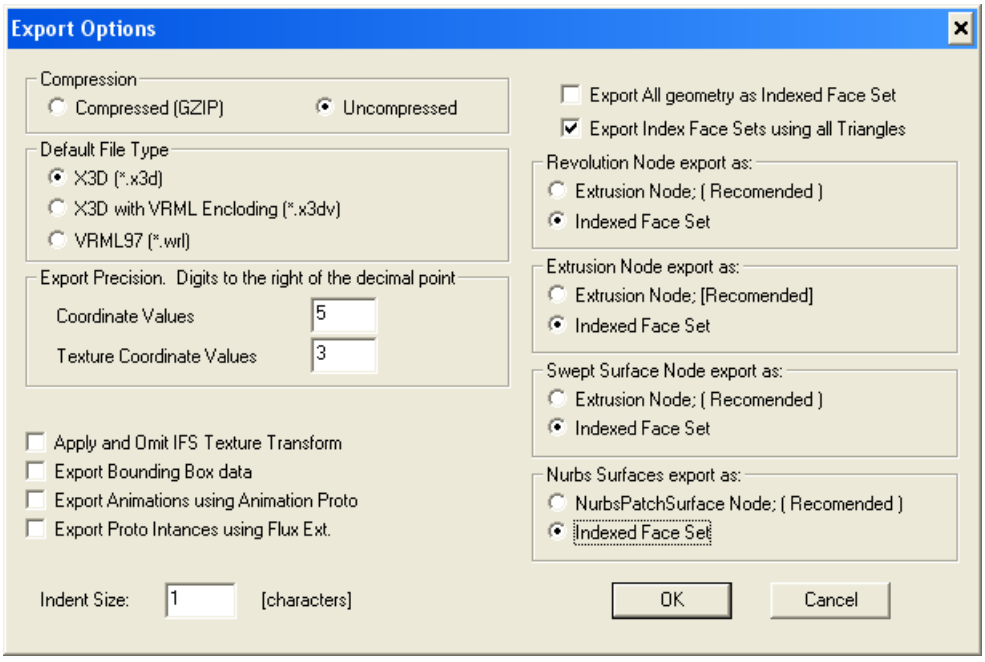


图 C-2　　对话框“导出选项”中的设置

7. 对于文本建议采用以下设置（另见下列对话框）：
   - 文本必须始终中心对齐。
   - 文本大小应设为 0.2。 便于文本进行精准的定位。 文本输出是通过 X3D 浏览器进行的，与操作界面中的字体大小无关。

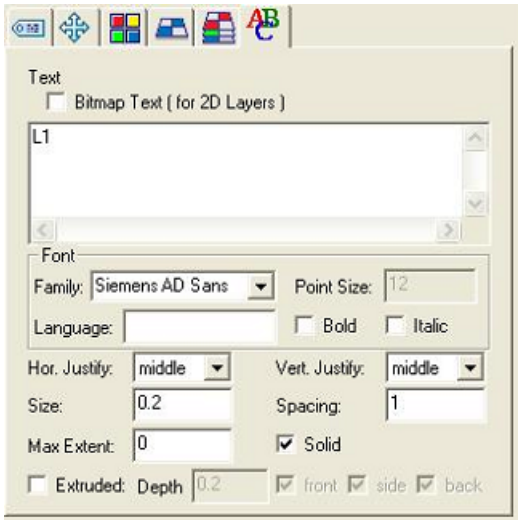


图 C-3　　文本设置

8. 如需创建一个阴影线，应将文件 schnitt.png 用作结构。 阴影线的走线方式始终是左下到右上，成 45°度角。 两个尺寸中的缩放比例都应设为 3。 旋转取决于结构形式且必须手动调整。

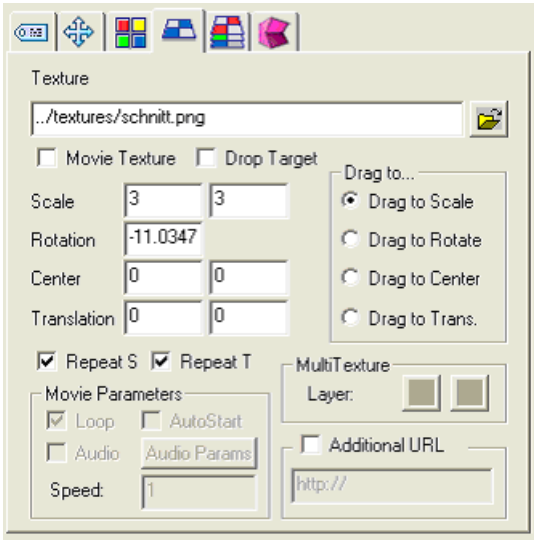


图 C-4　　阴影线设置

9. 毛坯的大小/尺寸建议按如下方式设置：
   – 用于车削加工的圆柱体：
     长度 = 5.5；半径 = 1.4（对比模板 *turning_blank.fxw*）
   – 用于铣削加工的长方六面体：
     x = 4.75；y = 3；z = 3（对比模板 *milling_blank.fxw*）

参见

XML 指令 (页 279)

C.2.3　导入图形（模型）

在 Flux Studio 中可以导入外部图形（模型）。 常用模型（如刀具）可作为 .x3d 或 .hmi 文件集中保存并作为完成的元素在其他辅助画面中重复使用。 随附的建模模板列表请见 建模模板 (页 277) 一章。

复杂的对象也可通过其他建模工具（如 Cinema4D）创建和导入。

下文介绍了如何重复使用这些模型。

作为 “Inline” 导入

Flux Studio 提供了对象 “Inline” 用于导入 .x3d 文件。 这样可以关联外部元素，而不增加这些模型的 3D 数据。

通过菜单项 **Create -> Create Inline** 插入对象。 之后在对象属性中输入文件名。

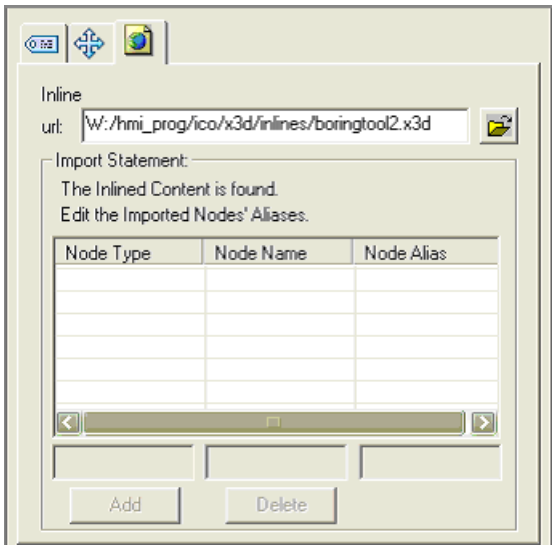


图 C-5　　通过对象 “Inline” 导入

导入 3D 模型数据

按如下步骤导入一个文件中的模型数据。

1. 通过菜单项 **File -> Import Other Format** 打开对话框 “Flux Studio Accutrans3D Translation Utility”。
2. 通过下拉菜单 “File Types” 选择相应的格式。 例如：如需导入一个 Cinema4D 文件，则须选择文件类型 “3D Studio”。
3. 然后通过 **Select Files** 选择所需的文件并开始导入。

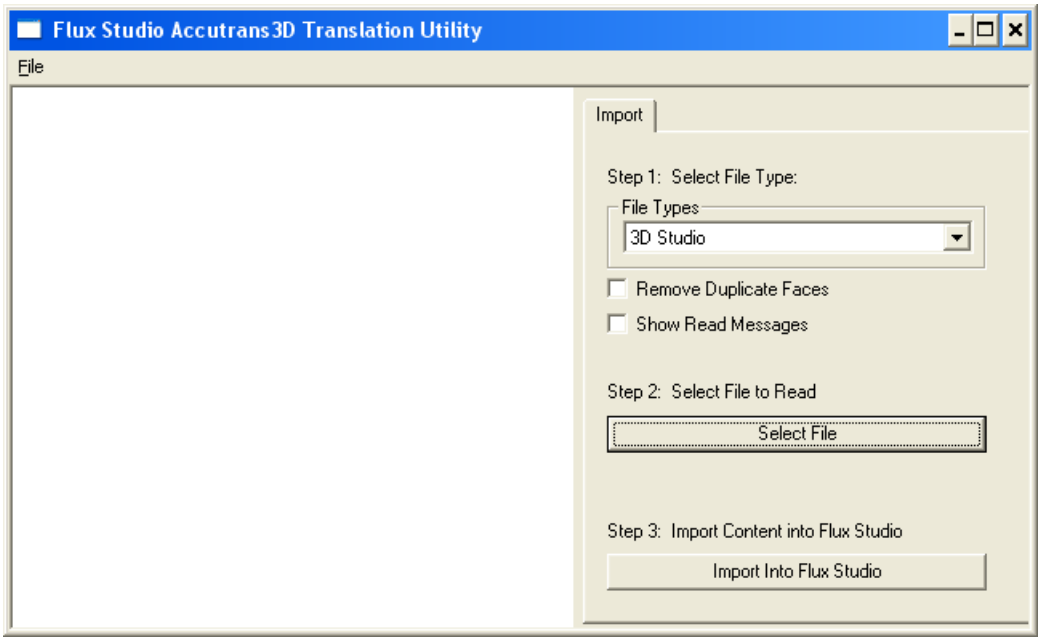


图 C-6　　导入 3D 模型数据

将 Cinema4D 文件中的模型数据作为整个组插入。 删除一并导入的所有摄像机和其他信息。只需要 Cinema4D 文件的图形元素。

C.2.4　建模模板

本章节中列举了一个作为外形、颜色和尺寸基础的图形元素列表。 为确保操作界面中的“外观”一致，建议使用模板格式或模板创建自有图形。

概述

模板	说明
intersection_texture.png	截面结构
rapid_traverse_line_hori.fxw	用于阐明刀具运行路径的水平线（快速模式）
rapid_traverse_line_vert.fxw	用于阐明刀具运行路径的垂直线（快速模式）
feed_traverse_line_hori.fxw	用于阐明刀具路径的水平线（进给模式）
feed_traverse_line_vert.fxw	用于阐明刀具路径的垂直线（进给模式）
dimensioning_lines_hori.fxw	水平尺寸辅助线
dimensioning_lines_vert.fxw	垂直尺寸辅助线
z1_inc.fxw	水平尺寸线 Z1
x1_inc.fxw	垂直尺寸线 X1
3d_coordinate_origin.fxw	3D 坐标原点
3d_zero_point.fxw	3D 零点

车削

模板	说明
turning_blank.fxw	用于车削工艺的毛坯：一个简易圆柱体
turning_centerline.fxw	中心线
turning_centerpoint.fxw	用于显示 WCS 中零点的坐标原点
turning_refpoint.fxw	加工参考点
turning_machining_area.fxw	加工面的边界线

铣削

模板	说明
milling_blank.fxw	用于车削工艺的毛坯：一个简易长方六面体
milling_centerline.fxw	中心线
milling_refpoint.fxw	加工参考点

C.3 XML 指令

C.3.1 一览

为了使 X3D 浏览器能够访问图形，需要场景说明文件（结尾 .xml）。 在场景说明文件中分配从配置中调用的场景名称及图形所在的 .x3d 文件 (TimeSensor) 中的时间。

C.3.2 场景说明文件的结构

下文显示了场景说明文件的结构并对各个 XML 指令进行了说明：

结构和说明

<!-- 在车削加工时处理平面相关的文本 -->

```
<TextPlane plane="G18_ZXY" />
```

说明

<TextPlane plane="G18_ZXY" />	在车削加工时处理平面相关的文本（轴名称）。 如果使用一个标记为“Z” 的文本（G18 第一根几何轴），相应的控制系统几何轴名称则会显示在辅助画面中（例如：“Z”）。

<!--文本对齐 -->

```
<TextPosition center='true' />
```

说明

<TextPosition center='true' />	文本对中的标记。 这有助于在旋转画面中显示文本。 推荐使用此项设置。

<!--调整刀具速度 -->

```
<ToolSpeedFactors planeSpeedFactor="0.4" rapidSpeedFactor="0.7"
reducedSpeedFac¬tor="1.0"/>
```

说明

<ToolSpeedFactors	如需调整刀具速度，可添加该项。
planeSpeedFactor="0.4"	进给速度系数 (0.4 = 40%)
rapidSpeedFactor="0.7"	快速速度系数 (0.7 = 70%)
reducedSpeedFactor="0.1" />	第三方速度系数 (0.1 = 10%)

<!--动画定义-->

```
<SceneKey name='BoringAnimation' masterRotationSpeed="-64.0"
maxRotAngle="45.0" begin-Time='0.110 endTime='0.118 view='camiso'
speedMaster='boringtool' Type="VIEW_3D_TURN_CYL">
```

说明

<SceneKey name='BoringAnimation'	动画名称
masterRotationSpeed="-64.0"	刀具的旋转方向和旋转速度。 该设置为可选项。
maxRotAngle="45.0"	避免混淆效应（刀具似乎按错误的方向旋转）。 该值通常是四分之一的刀具对称。 该设置为可选项。
beginTime='0.110	Timesensor 上的动画起始时间点
endTime='0.118'	Timesensor 上的动画结束时间点
view='camiso'	用于动画的摄像机
speedMaster='boringtool'	动画刀具名称
type="VIEW_3D_TURN_CYL">	视图类型确定视图（参见章节 视图类型 (页 283)）

<!-- 元素 -->

```
<Element name='1' time='0.110' feed='rapid' dwell=’2’/>
<Element name='2' time='0.112' feed='rapid'/>
<Element name='3' time='0.114' feed='rapid'/>
<Element name='4' time='0.116' feed='plane'/>
<Element name='5' time='0.118' feed='plane'/>
```

说明

<Element name='1'	表示针对的是第一个动画元素。
time='0.110'	Timesensor 上第一个元素的时间点
feed='rapid'	元素的运行速度 plane = 加工进给 rapid = 快速 reduced = 减速，速度低于 “plane”
dwell=’2’/>	动画运行暂停 旋转的刀具保持旋转状态。

<!-- 动画定义结束 -->

```
</SceneKey>
```

说明

</SceneKey>	动画定义结束

<!-- 现有动画作为新动画的来源 -->

```
<SceneKey source=''BoringAnimation' name=''BoringAnimationRear' mirror='
MirrorScreenX ' />
```

说明

<SceneKey source="BoringAnimation'	用作另一个动画来源的动画的名称。
name="BoringAnimationRear'	基于来源的新动画的名称
mirror=' MirrorScreenX ' />	按 X 轴方向镜像，在新动画中输出。

<!-- 静态场景（画面）定义（4 个示例） -->

```
<SceneKey name='Default' time='0.026' view='camiso'
type="VIEW_3D_DRILL_CUT"/>
<SceneKey name='Cut' time='0.046' view='camside' type="VIEW_SIDE"/>
<SceneKey name='Z1ink' time='0.056' view='camside' type="VIEW_SIDE"
 highLightedGroup='dad_Z1ink'/>
<SceneKey name='Z1abs' time='0.066' view='camside' type="VIEW_SIDE"
 highLightedGroup='dad_Z1abs'/>
```

说明

<SceneKeyname=’Z1abs'	画面名称
time='0.066'	Timesensor 上的画面时间点
view='camside'	用于画面的摄像机
type="VIEW_SIDE"	视图类型确定视图（参见章节 视图类型 (页 283)）
highLightedGroup='dad_Z1abs'/>	需要加亮的组的名称。 组在 FluxStudio 中已被命名为 “Z1abs”。 前缀 “dad_” 由 Flux 自动添加。

<!-- end of xml file –!>

C.3.3 镜像和旋转

通过指令 **mirror** 确定画面/模型的镜像或旋转。

说明

mirror="RotateScreenX"	画面围绕 X 轴旋转
mirror="RotateScreenY"	画面围绕 Y 轴旋转
mirror="RotateScreenZ"	画面围绕 Z 轴旋转
mirror="MirrorScreenX”	画面按 X 轴方向镜像
mirror="MirrorScreenY"	画面按 Y 轴方向镜像
mirror="MirrorScreenZ"	画面按 Z 轴方向镜像

mirror="RotatePieceX”	模型围绕 X 轴旋转
mirror="RotatePieceY”	模型围绕 Y 轴旋转
mirror="RotatePieceZ”	模型围绕 Z 轴旋转
mirror="MirrorPieceX”	模型按 X 轴方向镜像
mirror="MirrorPieceY”	模型按 Y 轴方向镜像
mirror="MirrorPieceZ”	模型按 Z 轴方向镜像

所有旋转和镜像方式都可组合使用。 旋转需要确定旋转角度。

示例

mirror="RotatePieceZ=180”

mirror="RotatePieceZ=-90 MirrorScreenX"

mirror="RotateScreenY=90"

mirror="MirrorPieceX MirrorPieceY”

mirror="MirrorPieceZ RotatePieceX=-90”

C.3.4 视图类型

通过视图类型确定视图。 视图以适用于显示对象的经验值和规定为基础。
借此确保对象的显示与操作界面坐标系（MD 52000 $MCS_DISP_COORDINATE_SYSTEM 或 MD 52001 $MCS_DISP_COORDINATE_SYSTEM_2）的设置相匹配。

说明

“VIEW_STATIC”	无需转换的直接视图
“VIEW_3D_TURN_CYL”	车削加工的 3D 视图（圆柱体）
“VIEW_3D_MILL_CUBE”	铣削加工的 3D 视图（长方六面体）
“VIEW_3D_DRILL_CUT”	钻削加工的 3D 视图（剖视图）
“VIEW_SIDE”	钻削加工的 2D 视图（剖视图）
“VIEW_TOP_GEO_AX_1”	第 1 根几何轴方向上的 2D 视图
“VIEW_TOP_GEO_AX_2”	第 2 根几何轴方向上的 2D 视图
“VIEW_TOP_GEO_AX_3”	第 3 根几何轴方向上的 2D 视图

C.4 转换为 hmi 文件

说明

在 HMI 启动时将含有配套 XML 文件的 X3D 文件转换为 HMI 文件。

必须为每个 X3D 文件创建一个相应（同名）的 XML 文件。 因此，必须将 X3D 文件和 XML 文件保存到目录 `Suchpfad des HMI\ico\x3d\turning` 或 `milling` 中。 步骤与 .ts 文件类似。

C.5 在 Create MyHMI /3GL 中显示

C.5.1 X3D 浏览器

如需在自有 OA 应用中显示目标辅助画面，则须将 X3D 浏览器小部件集成到自有 OA 应用中。

X3D 浏览器小部件提供了可在 HMI 内部显示 X3D 内容的接口。

类 SlX3dViewerWidget 用于显示图形场景。 该类的定义是在全局 GUI Include 目录 `\hmi_prog\gui\include` 下的标题文件 slx3dviewerwidget.h 中进行的。

C.5.2 类 SlX3dViewerWidget

该类提供了一个可灵活使用的小部件，该部件可以独立显示指定模型文件中的内容，必要时还可运行动画。

类接口由构造函数、析构函数及两种图形输出控制方法组成。

提供了功能全面的接口用于直接推导 Qt 类 QWidget（例如：show(), hide()和 resize(...)，但此处不进行进一步说明，详细信息请参考 Qt 文档）。

C.5.3 公共方法

SlX3dViewerWidget (QWidget* pParent = 0)

X3D 浏览器小部件的构造函数。

参数	含义
pParent	将参数传送给 Qwidget 的构造函数。

~SlX3dViewerWidget ( )

X3D 浏览器小部件的析构函数。

参数	含义
-	-

C.5.4 公共槽

void viewSceneSlot (const QString& rsFileName, const QString& rsScene, const QString& rsAnimationScene, int nChannel, int nPlane, SlStepTechnology nTechnology)

借助 viewSceneSlot 方法，X3D 浏览器可指定载入和显示 rsFileName 文件中的静态场景 rsScene 和动画场景 rsAnimationSzene。

正确的显示方式是先显示静态场景，再显示动画场景（按固定时间循环）。

如果未指定静态场景，则立即显示动画；相应的也可能未指定动画场景。

使用通道号、平面和工艺使场景转动到正确的位置上（取决于设置的机床坐标系）。

参数	含义
rsFileName	含待显示场景的文件的名称。
rsScene	静态场景名称。
rsAnimationScene	动画场景名称。
nChannel	通道号

参数	含义
nPlane	平面
nTechnology	工艺 针对计数类型 SlStepTechnology 定义了以下常量： • SL_STEP_NO_TECHNOLOGY • SL_STEP_MILLING • SL_STEP_TURNING • SL_STEP_SURFACE_GRINDING • SL_STEP_CIRCULAR_GRINDING

void viewSceneSlot (const QString& rsFileName, const QString& rsScene, const QString& rsAnimationScene)

一种简易化 viewSceneSlot 方法，X3D 浏览器可借助其指定载入和显示 rsFileName 文件中的静态场景 rsScene 和动画场景 rsAnimationSzene。

参数	含义
rsFileName	含待显示场景的文件的名称。
rsScene	静态场景名称。
rsAnimationScene	动画场景名称。

C.5.5 库

如需在自有项目中使用 X3D 浏览器，则须为库相关性列表扩展项 'slx3dviewer.lib'。

C.5.6 执行示例

执行示例请参考 Create MyHMI/3GL 程序包中的 **\examples\GUIFrameWork\SlExGuiX3D**。

C.5.7 机床数据

显示 MD 9104 $MM_ANIMATION_TIME_DELAY

至使用辅助画面中的动画的延时，单位：秒。 该设置对只进行了动画处理的辅助画面无效。

该设置对 SINUMERIK Operate 的所有动画都有效。

C.5.8 使用说明

- 如果 X3D 浏览器小部件消失，动画则会中断。 此时无需选中无动画场景选择。
- 考虑到性能和存储容量，应避免经常或多次实例化 X3D 浏览器小部件。 此类应用场景下推荐使用（执行）X3D 浏览器小部件。
- X3D 浏览器也可在信号槽方案中使用。
- 如果出错（例如：文件未找到或场景名称不可识别），X3D 浏览器小部件则会显示一个信息区。 一旦操控另一个辅助画面文件，该信息区则会再次自动消失。

C.6 在 Run MyScreens 中显示

本章节面向有经验的 “Run MyScreens” 开发人员。 可在对应的文档中查阅相关背景知识。

除了使用 .bmp 或 .png 格式的图片外，也可通过 X3D 浏览器显示动画辅助画面。

通常使用用于辅助画面的 Run MyScreens 接口进行集成。

如需输出 .bmp 或 .png 格式的图片，则应在定义对话框时在属性中输入数据 XG0 或清空参数。 如需将 X3D 辅助画面集成到对话框中，则须在属性中输入数据 **XG1**。

```
//M(MASK_F_DR_O1_MAIN/$85407////52,80/XG1)
```

操控单个辅助画面需要以下参数，其含义参见章节 5.3 中的公开槽：

1. 文件名
2. 静态场景（可选）
3. 动画场景（可选）
4. 工艺（可选）
5. 平面（可选）

参数按此顺序以字符串的形式排列汇总，用逗号隔开。

```
Hlp = "Dateiname,StaticScene,AnimationScene,Technologie,Ebene"
```

示例

- 默认辅助画面可通过指定变量 Hlp 设置。
  下面的示例是从文件 “MyDlgHelp.hmi” 中输出动画 “MyAnimation”。 未指定静态场景，即不会输出静态场景。 也没有指定工艺和平面数据，即使用默认值。

```
Hlp = "MyDlgHelp.hmi,,MyAnimation"
```

- 对于输入窗口的单个参数，可将对应变量上的属性 hlp 设为一个特定的辅助画面。
  下面的示例是从文件 “MyDlgHelp.hmi” 中输出平面 G17 中的静态场景 “MyParam” 和动画 “MyAnimation”。 未指定工艺数据，即使用默认值。

```
VarMyParam.hlp = "MyDlgHelp.hmi,MyParam,MyAnimation,,G17"
```

词汇表

PI 服务

在 NC 上执行固定操作的功能。 PI 服务可以由 PLC 和 HMI 系统调用。

PLC 硬键

PLC 硬键通过 HMI 软件的 PLC 接口提供，如同热键。 由操作界面触发的功能都是可设计的。

该按键可以是机床操作面板上的按键或者是 PLC 用户程序中的 PLC 信号连接端。 因此，它们也被称作“虚拟按键”。

编程支持

提供对话框以支持 → 建立零件程序，组件版本较高

编辑器

ASCII 编辑器可以通过符号在文件中输入和编辑。

编译器

编译器自动将 → 配置文件中定义的代码转换成 → 对话框并控制其使用。

变量

内存位的标记，可以通过指定 → 属性显示在 → 对话框中，可在其中输入数据和计算结果。

参数

参数是编程语句的可改变部分，在 → 配置文件中可用另一个字 / 符号替换。

操作树

多个相互连接的 → 对话框

登入软键

启动第一次新建的 → 对话框的软键。

定义行

定义 → 变量和 → 软键的程序段。

对话框

→ 操作界面的显示

- **和对话框相关的软键栏**
  由一个新设计的对话框调用的软键栏。
- **和对话框无关的软键**
  不由对话框调用的软键，即由第一个新对话框设计的登入软键和软键栏。

反编译

由 → 编程支持 → 对话框的输入栏可以在 → 零件程序中生成数控程序代码段。 反编译指的是相反的过程。 它会将数控程序代码段再次转为输入栏，然后显示在初始对话框中。

方法

如果出现相应的 → 事件，则执行编程的步骤。

访问等级

权限分级，即根据用户的不同权限来使用操作界面上的功能。

辅助变量

没有 → 属性并因此不显示在 → 对话框中内部计算变量。

块

用于 → 配置文件的装载单元

栏索引

数组栏索引

零件程序

规定轴的运动过程以及各种特殊动作的、以数控语言编制的程序。

模拟

模仿 → 零件程序过程，但没有实际的机床轴运动。

配置文件

包含定义和指令，并确定 → 对话框的外观和 → 功能的文件。

热键

OP 010、OP 010C 和带有热键块的 SINUMERIK 键盘上的 6 个按键，按下此键直接选择一个操作区。 也可以选择 2 个其它的按键作为热键操作。

软键标签

软键在屏幕上的文本/图形。

软键栏

所有水平软键或者所有垂直软键

事件

所有触发处理→方法的事件： 输入字符，按下 → 软键， ...

输入/输出栏

即 I/O 栏： 用来输入或者输出变量值。

数组

通过数组可以归类、保存同一数据类型的数据，从而可以通过索引存取数据。

行索引

数组行号码

用户变量

由用户在 → 零件程序或者数据块中定义的变量。

属性

属于某个对象（→ 对话框或者 → 变量）的 → 属性。

对象的特征（例如：→ 变量）

属性

属于某个对象（→ 对话框或者 → 变量）的 → 属性。

对象的特征（例如：→ 变量）

转换栏

→ 在输入 / 输出栏中列明多个数值；通过转换栏检查： 某栏中的输入必须与列明值中的一个相符合。

索引

V

帮

保

报

背

比

变

表

不

操

长

常

单

导