SIEMENS

SINUMERIK

SINUMERIK 828D
Easy XML

编程手册

适用于：

CNC 软件　　版本 4.95

07/2021
6FC5398-3EP40-0RA0

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 Siemens AG 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
Digital Industries
Postfach 48 48
90026 NÜRNBERG
德国

文件订购号: 6FC5398-3EP40-0RA0
Ⓟ 07/2021 本公司保留更改的权利

目录

前言 1

1.1 关于 SINUMERIK

无论是普及型数控机床，还是标准型机床，或者是模块化高端机床，SINUMERIK 数控系统都能为不同类型的机床提供最佳解决方案。无论是单件生产还是批量生产、简单工件还是复杂工件，对于从样品和工具制造、模具制造乃至大批量生产的所有制造领域而言，SINUMERIK 自始至终都是高生产率的自动化解决方案。

详细信息请访问网页 SINUMERIK (https://www.siemens.com/sinumerik)。

1.2 关于本手册

目标使用人群

该手册供以下人员使用：

- 编程人员
- 设计人员

优点

此编程手册可协助目标用户，使用基于 XML 的脚本元素并根据客户和应用的需要进行操作界面的自定义设计。

标准功能范畴

本文档描述了标准功能范畴。该描述可能和交付的系统的功能有所不同。交付的系统的功能仅以订购资料为准。

在系统中也可能会运行本文档中未说明的功能，但这并不表示在交付系统时必须提供这些功能以及相关的维修服务。

为使文档简明清晰，本文档并不包含所有产品类型的所有详细信息，也无法对安装、运行和维护中可能出现的各种情况逐一进行说明。

机床制造商在产品上增添或者更改的功能，由机床制造商进行说明。

第三方网页

本文档可能包含第三方网页链接。西门子对此类网页的内容不承担任何责任，也不会声明或认可此类网页或其内容为西门子所有。西门子并不能控制此类网页上的信息，也不对上述网页的内容和信息负责。使用上述网页的风险由用户承担。

1.3 网上文档

1.3.1 SINUMERIK 828D 文档一览

有关 SINUMERIK 828D 功能的详细文档，自版本 4.8 SP4 起，请参见 828D 文档一览 (https://support.industry.siemens.com/cs/ww/en/view/109766724)。

您可以直接打开文档或者下载 PDF 和 HTML5 格式。

文档分为以下几个类别：

- 用户：操作
- 用户：编程
- 制造商/服务：选型
- 制造商/服务：调试
- 制造商/服务：功能
- 制造商/服务：Safety Integrated
- SINUMERIK Integrate/MindApp
- 介绍和培训

1.3.2 SINUMERIK 操作组件文档一览

有关 SINUMERIK 操作组件的全部文档，请参见 SINUMERIK 操作组件文档一览 (https://support.industry.siemens.com/cs/document/109783841/technische-dokumentation-zu-sinumerik-bedienkomponenten?dti=0&lc=en-WW)。

您可以直接打开文档或者下载 PDF 和 HTML5 格式。

文档分为以下几个类别：

- 操作面板
- 机床控制面板
- 机床按钮面板
- 手持单元/微型手持单元
- 其他操作组件

有关“SINUMERIK”的重要文档、文章和链接，请参见 SINUMERIK 专题页 (https://support.industry.siemens.com/cs/document/109766201/sinumerik-an-overview-of-the-most-important-documents-and-links?dti=0&lc=en-WW)。

1.4 技术文档反馈

对于西门子工业在线支持上发布的任何技术文档，如有疑问、建议或改进意见，请点击文章末尾的链接“发送反馈”。

1.5 mySupport 文档

使用网页版“mySupport 文档”可以自由组合西门子文档内容，创建自己的文档。

在 mySupport 首页 (https://support.industry.siemens.com/cs/cn/zh/my)上点击“我的文档”，便可启动应用：

配置的手册可以 RTF、PDF 或 XML 格式导出。

说明

在链接“配置”下可以查看网页版“mySupport 文档”支持的西门子文档内容。

1.6 服务与支持

产品支持

有关产品的详细信息请访问网址：

产品支持 (https://support.industry.siemens.com/cs/cn/zh/)

在该网址下可以提供：

- 最新产品信息（产品公告）
- FAQ（常见问题与解答）
- 手册
- 下载链接
- 持续提供产品最新信息的新闻。
- “技术论坛”，供全球用户和专家交流经验、分享信息
- “联系人”，提供全球联系人信息，方便查找本地联系人
- “售后服务”，提供现场服务、维修、备件等信息

技术支持

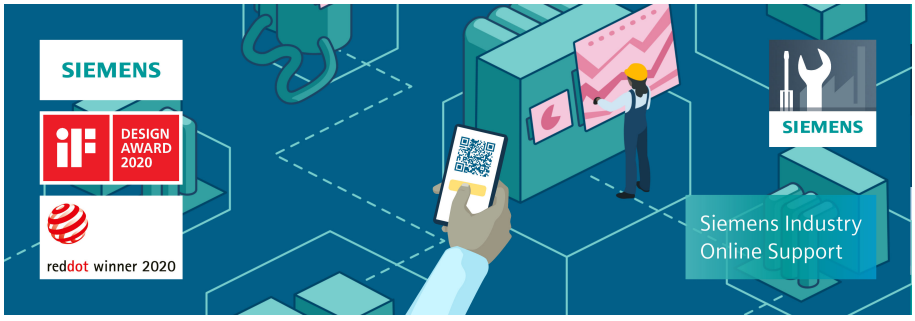
访问网址 (https://support.industry.siemens.com/cs/cn/zh/sc/4868)下的“联系方式”，便可以获取各个国家技术支持的电话号码。

如需咨询技术疑问，请使用“支持请求”一栏下的在线表格。

培训

访问网址 (https://www.siemens.com/sitrain)，可获取有关 SITRAIN 的相关信息。SITRAIN 为西门子的驱动和自动化产品、系统和解决方案提供培训。

无论在何处都能得到最佳的支持



使用荣获大奖的“西门子工业在线支持”App，您可以随时随地查看超过 30 万份的西门子工业领域的产品文件。该应用也可帮助您：

- 解决项目实施中出现的问题
- 排除故障
- 进行设备扩展或重新规划

此外，您还可以登录技术论坛，查看我们的专家为您撰写的其他文章：

- 常见问题与解答
- 应用实例
- 手册
- 证书
- 产品公告等

“西门子工业在线支持”App 提供 Apple iOS 版和安卓版。

铭牌上的二维码

铭牌上的二维码包含了各设备的数据。使用任一智能手机通过“西门子工业在线支持”App 扫描该二维码，便可获取相应设备的技术信息。

1.7 重要产品信息

OpenSSL 的使用

本产品可包含以下软件：

- 由 OpenSSL 项目开发并应用在 OpenSSL Toolkit 中的软件
- 由 Eric Young 开发的加密软件
- 由 Eric Young 开发的软件

详细信息请访问网址：

- OpenSSL (https://www.openssl.org)
- Cryptsoft (https://www.cryptsoft.com)

遵守通用数据保护条例

西门子遵守通用数据保护条例，特别是隐私保护设计（privacy by design）规定。

对于本产品意味着：

产品不会处理或保存个人相关数据，只会处理或保存技术功能数据（例如：时间戳）。用户如果将此类技术功能数据与其他数据（例如：排班表）关联或者将个人相关数据存储在同一介质（例如：硬盘）上而产生个人相关性，则应由用户自行确保遵循数据保护法规。

基本安全说明 2

2.1 一般安全说明

⚠ 警告

其他能源可导致电击危险和生命危险

接触带电部件可能会造成人员重伤，甚至是死亡。

- 只有专业人员才允许在电气设备上作业。
- 在所有作业中必须遵守本国的安全规定。

通常有以下安全步骤：

1. 准备断电。通知会受断电影响的组员。
2. 给驱动系统断电并确保不会再次接通。
3. 请等待至警告牌上说明的放电时间届满。
4. 确认功率接口和安全接地连接无电压。
5. 确认辅助电压回路已断电。
6. 确认电机无法运动。
7. 检查其他所有危险的能源供给，例如：压缩空气、液压、水。将能源供给置于安全状态。
8. 确保正确的驱动系统已经完全闭锁。

结束作业后以相反的顺序恢复设备的就绪状态。

⚠ 警告

连接不合适的电源可导致电击危险

连接不合适的电源会导致可接触部件携带危险电压，从而导致人员重伤，甚至是死亡。

- 所有的连接和端子只允许使用可以提供 SELV(Safety Extra Low Voltage：安全低压) 或 PELV(Protective Extra Low Voltage：保护低压) 输出电压的电源。

⚠ 警告

设备损坏可导致电击危险

未按规定操作会导致设备损坏。设备损坏后，其外壳或裸露部件可能会带有危险电压，接触外壳或这些裸露部件可能会导致重伤或死亡。

- 在运输、存放和运行设备时应遵循技术数据中给定的限值。
- 不要使用已损坏的设备。

⚠ 警告

电缆屏蔽层未接地可导致电击危险

电缆屏蔽层未接地时，电容超临界耦合可能会出现致命的接触电压。

- 电缆屏蔽层和未使用的电缆芯线至少有一侧通过接地的外壳接地。

⚠ 警告

缺少接地可导致电击危险

防护等级 I 的设备缺少安全接地连接或连接出错时，在其裸露的部件上会留有高压，接触该部件会导致重伤或死亡。

- 按照规定对设备进行接地。

注意

使用不合适的螺丝刀可损坏设备

使用不合适的螺丝刀或者采用不恰当的拧紧操作都可能损坏设备上的螺钉。

- 请使用与螺钉头完全匹配的螺丝刀。
- 请使用技术文档中规定的扭矩拧紧螺钉。
- 请使用扭力扳手或者带动态扭矩传感器和转速限制功能的机械式高精度螺丝刀。

⚠ 警告

内置型设备内可引起火灾

发生火灾时，内置型设备的外壳无法避免火苗和烟雾冒出。这可能导致人员重伤或财产损失。

- 将内置型设备安装在合适的金属控制柜中，从而保护人员免受火苗和烟雾伤害，或者对人员采取其他合适的防护措施。
- 确保烟雾只能经所设安全通道排出。

⚠ 警告

无线电设备或移动电话可导致机器意外运动

在设备的无屏蔽范围内使用无线电设备或移动电话，会干扰设备功能。功能异常会对设备功能安全产生影响并能导致人员伤亡或财产损失。

- 大约距离组件 20 cm 时，请关闭无线电设备或移动电话。
- 仅在已关闭的设备上使用“SIEMENS Industry Online Support App”。

⚠ 警告

通风空间不足可引起火灾

通风空间不足会导致过热，产生烟雾，引发火灾，从而造成人身伤害。这可能就是导致重伤或死亡的原因。此外，设备/系统故障率可能会因此升高，使用寿命缩短。

- 组件之间应保持规定的最小间距，以便通风。

注意

安装位置错误可导致过热

安装位置错误时，设备可能会过热并因此损坏。

- 只允许在规定的安装位置上运行设备。

⚠ 警告

安全功能失效可导致机器意外运动

无效的或不适合的安全功能可引起机器意外运动，可能导致重伤或死亡。

- 调试前请注意相关产品文档中的信息。
- 对整个系统和所有安全相关的组件进行安全监控，以确保安全功能。
- 进行适当设置，以确保所使用的安全功能是与驱动任务和自动化任务相匹配并激活的。
- 执行功能测试。
- 在确保了机器的的安全功能能正常工作后，才开始投入生产。

说明

Safety Integrated 功能的重要安全说明

使用 Safety Integrated 功能时务必要注意 Safety Integrated 手册中的安全说明。

⚠ 警告

因参数设置错误或修改参数设置引起机器故障

参数设置错误可导致机器出现故障，从而导致人员重伤或死亡。

- 采取保护措施，防止未经授权的参数设置。
- 采取适当措施（如驻停或急停）处理可能出现的故障。

2.2 静电场或静电放电可导致设备损坏

静电敏感元器件 (ESD) 是可被静电场或静电放电损坏的元器件、集成电路、电路板或设备。

注意

静电场或静电放电可导致设备损坏

电场或静电放电可能会损坏单个元件、集成电路、模块或设备，从而导致功能故障。

- 仅允许使用原始产品包装或其他合适的包装材料（例如：导电的泡沫橡胶或铝箔）包装、存储、运输和发运电子元件、模块和设备。
- 只有采取了以下接地措施之一，才允许接触元件、模块和设备：
  - 佩戴防静电腕带
  - 在带有导电地板的防静电区域中穿着防静电鞋或配带防静电接地带
- 电子元件、模块或设备只能放置在导电性的垫板上（带防静电垫板的工作台、导电的防静电泡沫材料、防静电包装袋、防静电运输容器）。

2.3 应用示例的质保规定

应用示例在组态和配置以及各种突发事件方面对设备没有强制约束力，无需一一遵循。应用示例不会提供客户专用的解决方案，仅在典型任务设置中提供保护。

用户自行负责上述产品的规范运行事宜。应用示例并没有解除您在应用、安装、运行和维护时确保安全环境的责任。

2.4 安全性信息

Siemens 为其产品及解决方案提供了工业信息安全功能，以支持工厂、系统、机器和网络的安全运行。

为了防止工厂、系统、机器和网络受到网络攻击，需要实施并持续维护先进且全面的工业信息安全保护机制。Siemens 的产品和解决方案构成此类概念的其中一个要素。

客户负责防止其工厂、系统、机器和网络受到未经授权的访问。只有在有必要连接时并仅在采取适当安全措施（例如，防火墙和/或网络分段）的情况下，才能将该等系统、机器和组件连接到企业网络或 Internet。

关于可采取的工业信息安全措施的更多信息，请访问 https://www.siemens.com/industrialsecurity (https://www.siemens.com/industrialsecurity)。

Siemens 不断对产品和解决方案进行开发和完善以提高安全性。Siemens 强烈建议您及时更新产品并始终使用最新产品版本。如果使用的产品版本不再受支持，或者未能应用最新的更新程序，客户遭受网络攻击的风险会增加。

要及时了解有关产品更新的信息，请订阅 Siemens 工业信息安全 RSS 源，网址为 https://www.siemens.com/industrialsecurity (https://new.siemens.com/global/en/products/services/cert.html#Subscriptions)。

其他信息请上网查找：

工业安全功能选型手册 (https://support.industry.siemens.com/cs/cn/zh/view/108862708/en)

⚠ 警告

篡改软件会引起不安全的驱动状态

篡改软件（如：病毒、木马、蠕虫等）可使设备处于不安全的运行状态，从而可能导致死亡、重伤和财产损失。

- 总是使用最新版本的软件。
- 将自动化和驱动组件集成到设备或机器上的整套先进工业信息安全方案中。
- 全面考虑整套工业信息安全方案中使用的所有产品。
- 采取相应的保护措施（如：使用杀毒软件）防止移动存储设备中的文件受到恶意软件的破坏。
- 在调试结束后，检查所有和安全相关的设置。

2.5 驱动系统（电气传动系统）的遗留风险

机器或设备制造商在依据相应的本地指令（比如欧盟机械指令）对机器或设备进行风险评估时，必须注意驱动系统的控制组件和驱动组件会产生以下遗留风险：

1. 调试、运行、维护和维修时机器或设备部件意外运行，原因（举例）：
   - 编码器、控制器、执行器和连接器中出现了硬件故障和/或软件故障
   - 控制器和传动设备的响应时间
   - 运行和/或环境条件不符合规定
   - 凝露/导电杂质
   - 参数设置、编程、布线和安装出错
   - 在电子器件附近使用无线电装置/移动电话
   - 外部影响/损坏
   - X 射线辐射、电离辐射和宇宙辐射
2. 在出现故障时，组件内/外部出现异常温度、明火以及异常亮光、噪音、杂质、气体等，原因可能有：
   - 零件失灵
   - 软件故障
   - 运行和/或环境条件不符合规定
   - 外部影响/损坏
3. 危险的接触电压，原因（举例）：
   - 零件失灵
   - 静电充电感应
   - 旋转电机的感应电压
   - 运行和/或环境条件不符合规定
   - 凝露/导电杂质
   - 外部影响/损坏
4. 设备运行中产生的电场、磁场和电磁场可能会损坏近距离的心脏起搏器支架、医疗植入体或其它金属物。
5. 当不按照规定操作以及/或违规处理废弃组件时，会释放破坏环境的物质并且产生辐射。
6. 影响通讯系统，如中央控制发送器或通过电网进行的数据通讯

其它有关驱动系统组件产生的遗留风险的信息见用户技术文档的相关章节。

创建用户对话框 3

3.1 功能范围

概述

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

控制系统提供基于 XML 的脚本语言用于创建用户对话框。

该脚本语言可以在 SINUMERIK Operate 上的操作区 <CUSTOM> 中显示机床专用菜单和对话框窗口。

所有对话框窗口的构成都与语言无关。这种情况下系统会从随附的语言数据库中读取要显示的文本。

使用

已定义的 XML 指令可以实现下列特性：

1. 显示对话框并提供：
   - 软键
   - 变量
   - 文本和帮助文本
   - 图形和帮助画面
2. 通过以下方法调用对话框：
   - 按下（登入）软键
3. 动态重组对话框
   - 修改、删除软键
   - 定义并设计变量栏
   - 显示、更换、删除显示文本（和语言相关或无关）
   - 显示、更换、删除图形
   - 动态创建可执行脚本并集成到脚本处理过程中

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

关于脚本语言的有效单元（标签）的描述可参见章节 “XML 标签” (页 45)。

说明

以下章节“基础配置”对 XML（Extensible Markup Language 可扩展标记语言）的描述并不完整。更多的信息可查阅相应的专业文献。

3.2 配置基础

配置文件

新操作界面的说明存储在配置文件中。这些文件自动编译并显示屏幕上的结果。在供货时并不提供配置文件，因此必须首先创建或下载此文件。

可以使用 XML 编辑器或其他的文本编辑器来创建配置文件。

说明

文件名只能由小写字母构成。

菜单树的原理

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

可以在登入菜单后面通过配置好的登入软键生成更多的菜单树：

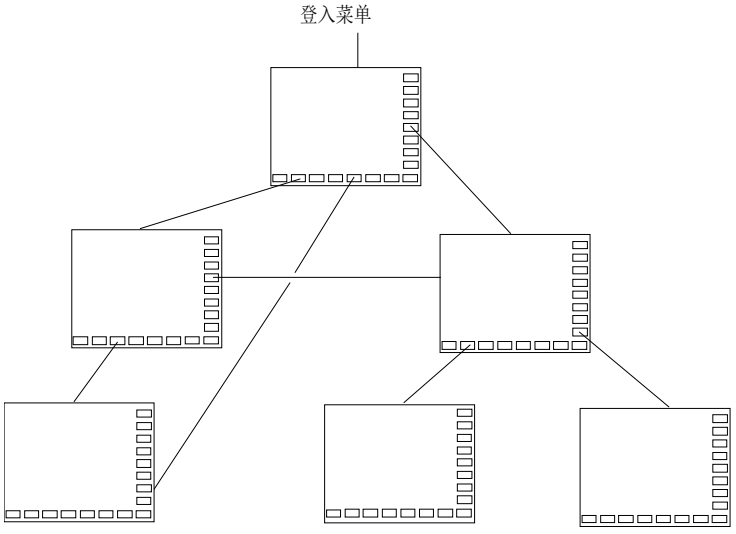


图 3-1 用户对话框菜单树

登入菜单

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

使用“主”菜单可以与加载自身对话框或其他软键条连接在一起，而使用它们又可以执行其他动作。

下图显示了控制系统上的制造商目录“System CF-Card/oem/sinumerik/hm”。

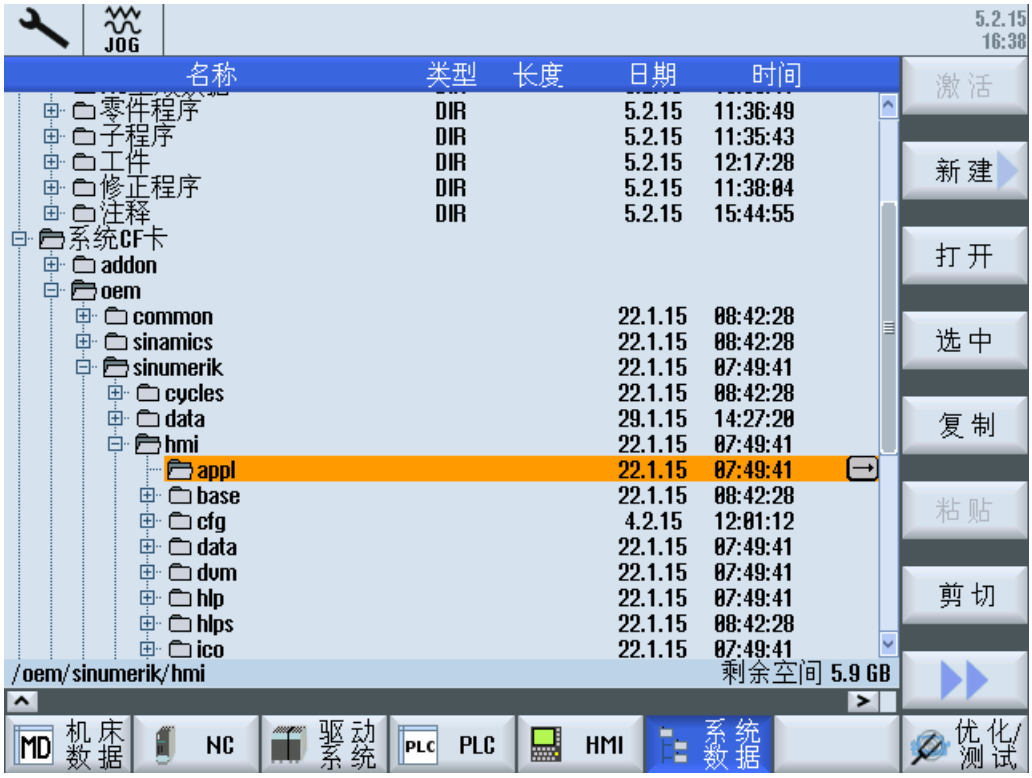


图 3-2　制造商目录

为了进行用户对话框的配置，需要使用控制系统中制造商目录“System CF-Card/oem/sinumerik/hmi”下的以下文件：

表格 3-1　用于配置的文件

文件类型	文件名称	含义	操作区调试 > 系统数据中的存储位置
脚本文件	“xmldial.xml”	该脚本文件通过 XML 标签控制已配置软键菜单的映像以及 SINUMERIK Operate 上操作区 <CUSTOM> 中的对话框屏幕。	制造商目录 > 在用于应用的子目录“appl”中
文本文件	“oem_xml_screens_xxx.ts”	用于用户操作界面的标准文本文件 该文本文件包含有用于单个语言菜单和对话框屏幕的文本。	制造商目录 > 在用于所需语言的子目录“lng”中

文件类型	文件名称	含义	操作区调试 > 系统数据中的存储位置
位图	---	控制系统支持 BMP 格式和 PNG 格式。	制造商目录 > 在子目录"ico"中
XML 文件，在控制文件"xmldial.xml"中插入了 XML 标签"INCLUDE"。	例如"machine_settings.xml"	该文件同样包含用来在 SINUMERIK Operate 上显示对话框窗口和参数的编程指令。	制造商目录 > 在用于应用的子目录"appl"中

Easy XML 连接点

以下连接点可用于 Easy XML 脚本：

硬键/软键	连接点
用户	标准脚本文件名"xmldial.xml"

软键"操作菜单"	标准脚本文件名"xmldial.xml"
	在文件"slamconfig.ini"中激活
	
	示例：
	`[CustomXML]`
	`TextId=SL_AM_CUSTOM`
	`SidescreenTextId=SL_SIDESCREEN_CUSTOM`
	`TextFile=slam`
	`TextContext=SlAmAreaMenu`
	`SoftkeyPosition=12`
	`Visible=true`

用户菜单	标准脚本文件名"menu_user.xml"
功能菜单	标准脚本文件名"menu_function.xml"

硬键/软键	**连接点**

PLC 硬键	脚本文件名基于按键说明。
	
	示例：
	`KEY50.0 = area:=CustomXML, dialog:=SlEECustomDialog, cmdline:="-conf slagmdialog.hmi -mainModule restore.xml -entry cmc2"`
	`KEY51.0 = area:=CustomXML, dialog:=SlEECustomDialog, cmdline:="-conf slagmdialog.hmi -mainModule activate.xml -entry cmc1"`

MMC 指令	脚本文件名基于 NC 指令说明。 **示例：** `MMC("POPUPDLG, PICTURE_ON, TEST.XML, cmd1","A)`

工作区域的预留软键	脚本文件名基于菜单说明。
	
	示例：
	`<MENU name="DgGlobalHu">`
	`<ETCLEVEL id="0">`
	`<SOFTKEYGROUP name="GroupEtc">`
	`<SOFTKEY position="7">`
	`<PROPERTY name="textID" type="QString">DG_SK</PROPERTY>`
	`<PROPERTY name="translationContext" type="QString">EASY_XML</PROPERTY>`
	`<FUNCTION args="-area CustomXML -dialog SlEECustomDialog -mainModule dg.xml -entry main" name="switchToDialog"/>`
	`</SOFTKEY>`
	`</SOFTKEYGROUP>`
	`</ETCLEVEL>`
	`</MENU>`

Sidescreen	脚本文件名基于“slsidescreen.ini”文件的 Sidescreen 说明。
Easy Extend 软键	标准脚本文件名“agm.xml”

快速选择键

同样，任意的用户对话框也可分配给前操作面板上的快速选择键<MENU USER>和<MENU FUNCTION>。为此提供了脚本命名文件"menu_user.xml"和"menu_function.xml"。

用户可将脚本文件从工具盒复制到制造商目录下或者创建自己的命名文件。名称“main”定义为了登入菜单。

3.3 设计文件

前言

下图显示了控制系统上的制造商目录“System CF-Card/oem/sinumerik/hmi”。

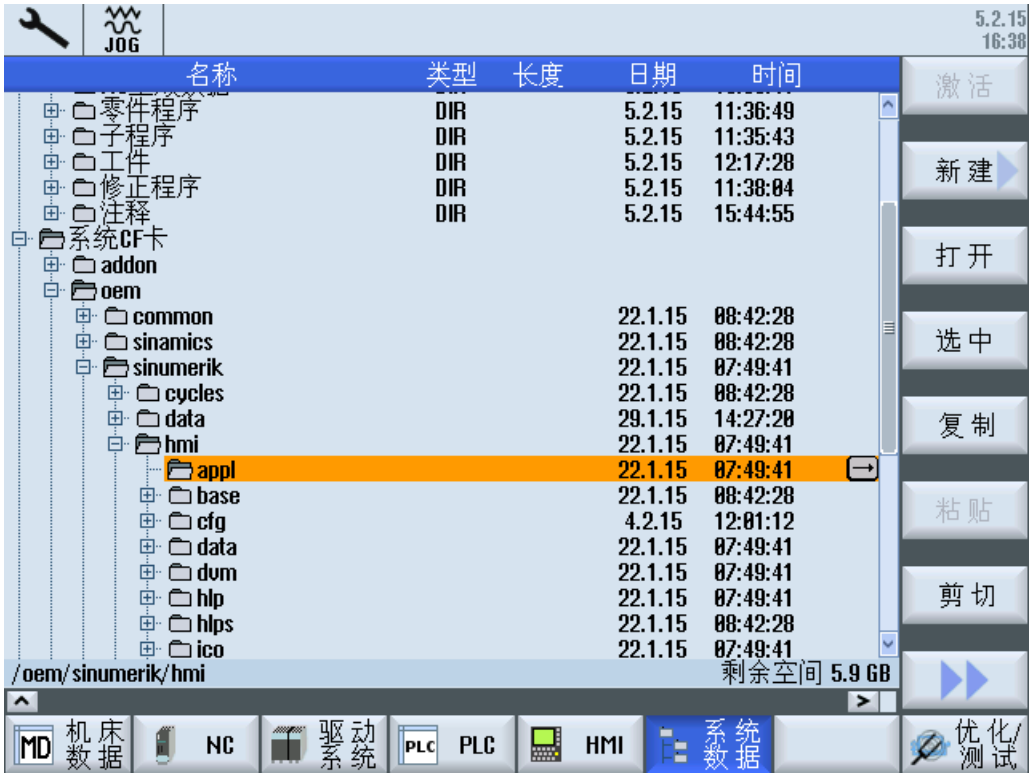


图 3-3 制造商目录

为了进行用户对话框的配置，需要使用控制系统中制造商目录“System CF-Card/oem/sinumerik/hmi”下的以下文件：

表格 3-2　用于配置的文件

文件类型	文件名称	含义	操作区调试 > 系统数据中的存储位置
脚本文件	“xmldial.xml”	该脚本文件通过 XML 标签控制已配置软键菜单的映像以及 SINUMERIK Operate 上操作区 <CUSTOM> 中的对话框屏幕。	制造商目录 > 在用于应用的子目录“appl”中
文本文件	“oem_xml_screens_xxx.ts”	该文本文件包含有用于单个语言菜单和对话框屏幕的文本。	制造商目录 > 在用于语言的子目录“lng”中
位图		控制系统支持 BMP 格式和 PNG 格式。	制造商目录 > 在子目录“ico”中 位图保存在控制系统所属的屏幕分辨率的子目录中。 Bem.：如给定了位图文件的路径，则可以将文件直接保存至该目录中。
XML 文件，在控制文件“xmldial.xml”中插入了 XML 标签 “INCLUDE” 。	例如 “machine_settings.xml”	该文件同样包含用来在 SINUMERIK Operate 上显示对话框窗口和参数的编程指令。	制造商目录 > 在用于应用的子目录“appl”中

文件与用户对话框配置的相关性

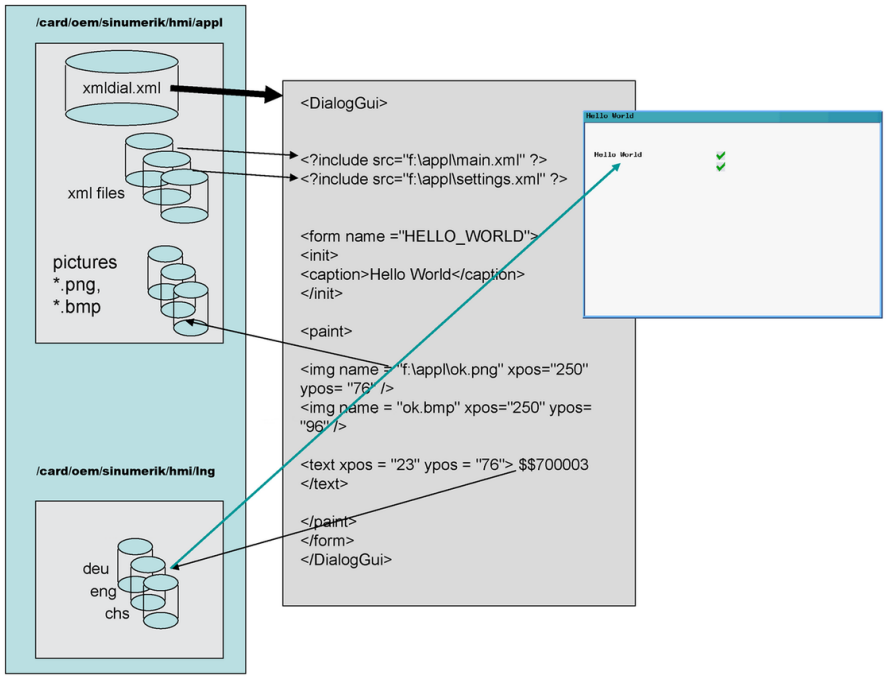


图 3-4　相关性

加载配置

与前面表格“用于配置的文件”的“操作区中的存储位置”一列的说明相同，所创建的文件被复制到制造商目录相应的子目录下。

说明

一旦子目录中有了用于应用的脚本文件“xmldial.xml”，用户就可以在操作区 <CUSTOM> 中启动用户对话框。

第一次复制后必须通过“正常上电启动”对控制系统进行复位。

SINUMERIK Operate 中的用户对话框示例

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

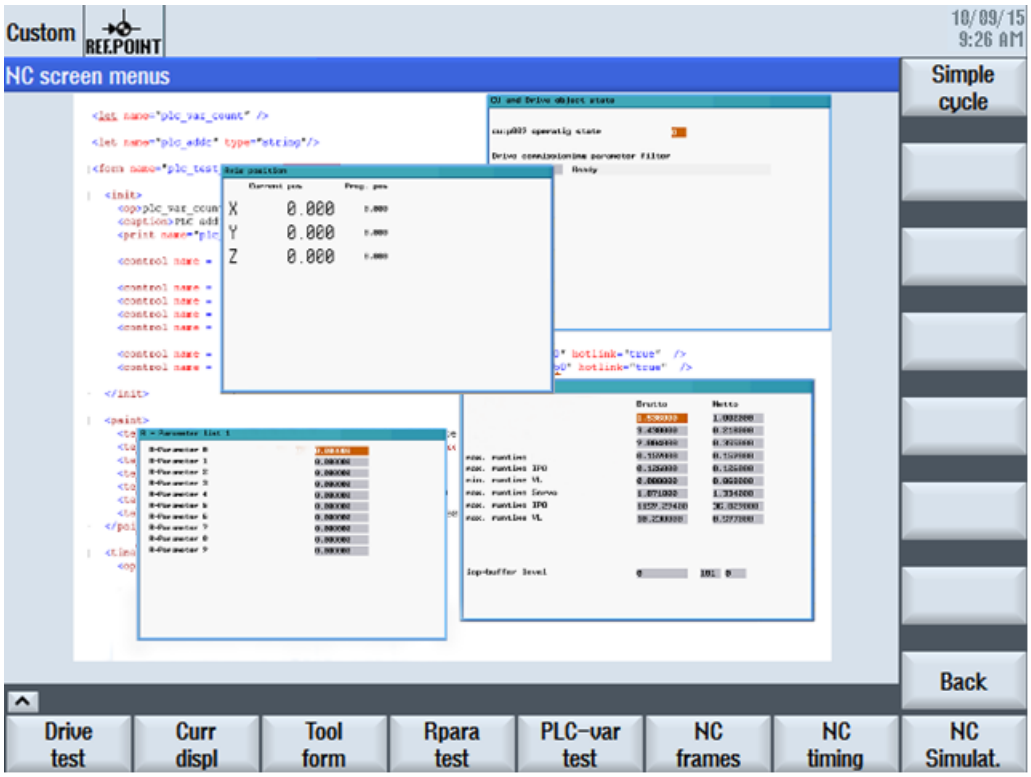


图 3-5　操作区 <CUSTOM> 中用户对话框示例

更多示例参见工具箱

3.4 设计文件的结构

概述

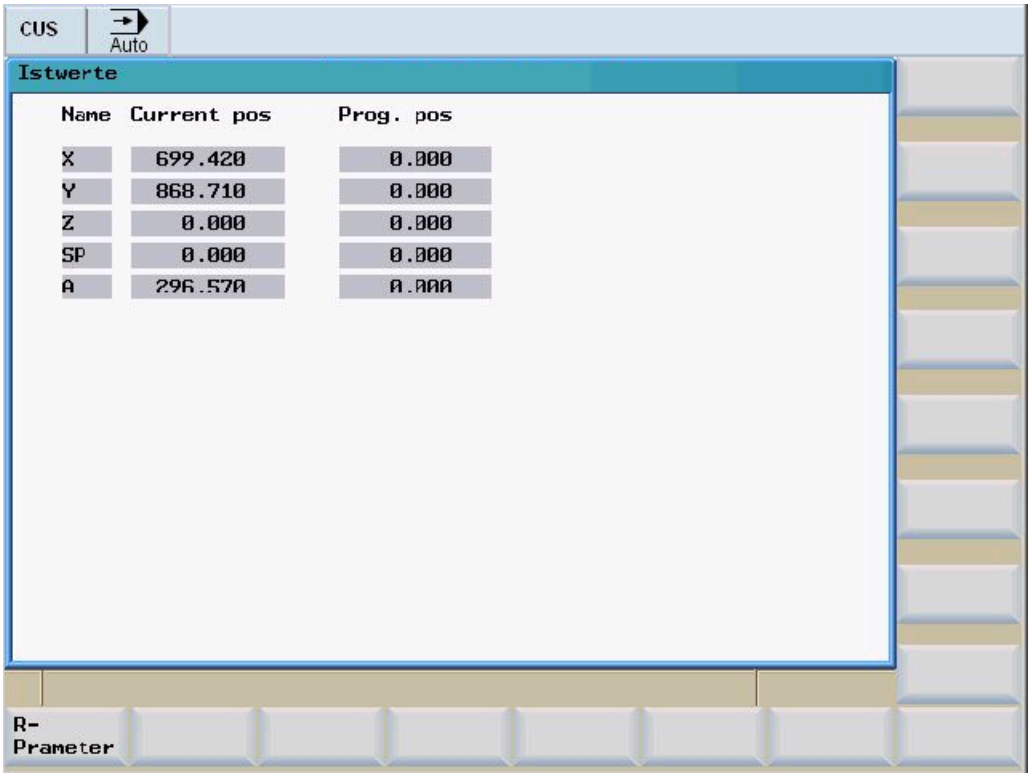
配置文件由以下单元组成：

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

下列示例说明了文件“xmldial.xml”的 XML 脚本以及相应的截屏。

脚本包含有用于显示实际值和剩余行程的对话框，以及一张 R 参数表。

实际值窗口截图



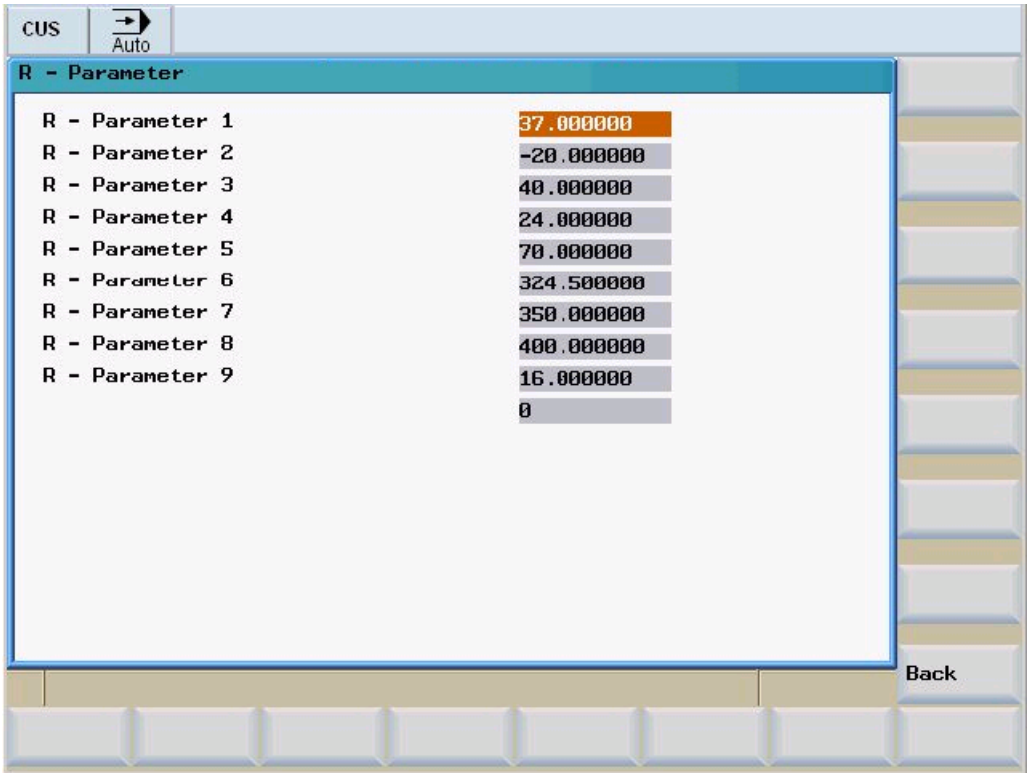
xmldial.xml

```
<DialogGui>

<!--
main menu
It is called by the system software. It starts the application.
The menu tag manages the soft key reactions. One input form can be assigned
to a menu tag.
-->
<menu name = "MAIN">
<OPEN_FORM name = "CURRENT_DISPLAY" />

<softkey POSITION="1">
<caption>R-%nPrameter</caption>
<navigation>MENU_R_PARAMETER</navigation> <!-- opens the menu R parameter --
>
</softkey>


</menu>


<form name = "CURRENT_DISPLAY">

<init>
<caption>Istwerte</caption>

<control name = "label1" xpos = "36" ypos = "56" width = "32"
fieldtype="readonly" refvar="nck/Channel/GeometricAxis/name[0]" />
<control name = "label2" xpos = "36" ypos = "76" width = "32"
fieldtype="readonly" refvar="nck/Channel/GeometricAxis/name[1]" />
<control name = "label3" xpos = "36" ypos = "96" width = "32"
fieldtype="readonly" refvar="nck/Channel/GeometricAxis/name[2]" />
<control name = "label4" xpos = "36" ypos = "116" width = "32"
fieldtype="readonly" refvar="nck/Channel/GeometricAxis/name[3]" />
<control name = "label5" xpos = "36" ypos = "136" width = "32"
fieldtype="readonly" refvar="nck/Channel/GeometricAxis/name[4]" />

<control name = "edit1" xpos = "80" ypos = "56" refvar="nck/Channel/
GeometricAxis/actProgPos[0]" hotlink="true" fieldtype="readonly"
format="%9.3f" time="super fast"/>
<control name = "edit2" xpos = "80" ypos = "76" refvar="nck/Channel/
GeometricAxis/actProgPos[1]" hotlink="true" fieldtype="readonly"
format="%9.3f" time="super fast"/>
<control name = "edit3" xpos = "80" ypos = "96" refvar="nck/Channel/
GeometricAxis/actProgPos[2]" hotlink="true" fieldtype="readonly"
format="%9.3f" time="super fast"/>
<control name = "edit4" xpos = "80" ypos = "116" refvar="nck/Channel/
GeometricAxis/actProgPos[3]" hotlink="true" fieldtype="readonly"
format="%9.3f" time="super fast"/>
```

xmldial.xml

```
<control name = "edit5" xpos = "80" ypos = "136" refvar="nck/Channel/
GeometricAxis/actProgPos[4]" hotlink="true" fieldtype="readonly"
format="%9.3f" time="super fast"/>

<control name = "edit11" xpos = "210" ypos = "56" refvar="nck/Channel/
GeometricAxis/progDistToGo[0]" hotlink="true" fieldtype="readonly"
format="%9.3f" time="super fast"/>
<control name = "edit12" xpos = "210" ypos = "76" refvar="nck/Channel/
GeometricAxis/progDistToGo[1]" hotlink="true" fieldtype="readonly"
format="%9.3f" time="super fast"/>
<control name = "edit13" xpos = "210" ypos = "96" refvar="nck/Channel/
GeometricAxis/progDistToGo[2]" hotlink="true" fieldtype="readonly"
format="%9.3f" time="super fast"/>
<control name = "edit14" xpos = "210" ypos = "116" refvar="nck/Channel/
GeometricAxis/progDistToGo[3]" hotlink="true" fieldtype="readonly"
format="%9.3f" time="super fast"/>
<control name = "edit15" xpos = "210" ypos = "136" refvar="nck/Channel/
GeometricAxis/progDistToGo[4]" hotlink="true" fieldtype="readonly"
format="%9.3f" time="super fast"/>

</init>

<paint>
<text xpos= "36" ypos="30">Name</text>
<text xpos= "80" ypos="30">Current pos</text>
<text xpos= "210" ypos="30">Prog. pos</text>

</paint>
</form>

.....
```

R 参数窗口截图


CUS
Auto
R - Parameter
R - Parameter 1
37.000000
R - Parameter 2
-20.000000
R - Parameter 3
40.000000
R - Parameter 4
24.000000
R - Parameter 5
70.000000
R - Parameter 6
324.500000
R - Parameter 7
350.000000
R - Parameter 8
400.000000
R - Parameter 9
16.000000
0
Back

xmldial.xml

```
.....

<!-- ************************************************************
Menu R- Parameter
-->

<menu name ="MENU_R_PARAMETER">
<OPEN_FORM name = "R-Parameter" />

<softkey POSITION="16">
<caption>Back</caption>
<navigation>MAIN</navigation>
</softkey>

</menu>

<form name = "R-Parameter">

<init>
<DATA_ACCESS type="true" />
<caption>R - Parameter</caption>

<control name = "edit1" xpos = "322" ypos = "34" refvar="nck/Channel/
Parameter/R[1]" />
<control name = "edit2" xpos = "322" ypos = "54" refvar="nck/Channel/
Parameter/R[2]" />
<control name = "edit3" xpos = "322" ypos = "74" refvar="nck/Channel/
Parameter/R[3]" />
<control name = "edit4" xpos = "322" ypos = "94" refvar="nck/Channel/
Parameter/R[4]" />
<control name = "edit5" xpos = "322" ypos = "114" refvar="nck/Channel/
Parameter/R[5]" />
<control name = "edit6" xpos = "322" ypos = "134" refvar="nck/Channel/
Parameter/R[6]" />
<control name = "edit7" xpos = "322" ypos = "154" refvar="nck/Channel/
Parameter/R[7]" />
<control name = "edit8" xpos = "322" ypos = "174" refvar="nck/Channel/
Parameter/R[8]" />
<control name = "edit9" xpos = "322" ypos = "194" refvar="nck/Channel/
Parameter/R[9]" />
<control name = "edit10" xpos = "322" ypos = "214" refvar="nck/Channel/
Parameter/R[10]" />
<control name = "edit10" xpos = "322" ypos = "214" refvar="plc/mb170" />
</init>

<paint>
<text xpos = "23" ypos = "34">R - Parameter 1</text>
<text xpos = "23" ypos = "54">R - Parameter 2</text>
<text xpos = "23" ypos = "74">R - Parameter 3</text>
```

xmldial.xml

```
<text xpos = "23" ypos = "94">R - Parameter 4</text>
<text xpos = "23" ypos = "114">R - Parameter 5</text>
<text xpos = "23" ypos = "134">R - Parameter 6</text>
<text xpos = "23" ypos = "154">R - Parameter 7</text>
<text xpos = "23" ypos = "174">R - Parameter 8</text>
<text xpos = "23" ypos = "194">R - Parameter 9</text>
</paint>
</form>

</DialogGui>
```

3.5 语言相关性

使用和语言相关的文本：

- 软键标记
- 标题
- 辅助文本
- 其它任意文本

与语言相关的文本保存在文本文件中。

说明

使用该文本文件时需要进行下列操作：

- 提供需要使用的语言。
- 传输到控制系统相应的语言目录中。

3.6 XML 诊断

为了诊断和找出脚本错误，系统提供了“Easy XML 诊断”功能。

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

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

激活 Easy XML 诊断

示例：

```
<DialogGui diagnose="true" >
…
…
</DialogGui >
```

或

显示机床数据

MD9113 $MM_EASY_XML_DIAGNOSE		Easy XML 脚本的诊断与更正辅助支持
= 0	诊断无效	
= 1	语法检查生效	
= 0x100	错误报告会另外保存在文件 error_log.txt 中。	

通过跟踪输出诊断

可通过标签 **debug** 将跟踪输出集成至脚本。只有在标签 **DialogGui** 中的 **debug_msg** 属性值为 **on** 时，保存才能成功。

软键功能一览

对话框	软键	功能
“EasyXML 诊断”主菜单	启动脚本	直接启动加载的项目。
	启动检查	启动语法检查。生成的所有信息都可加以查看。可重新进行检查。
“故障”和“报警”诊断	故障	结果会根据故障和报警进行分类显示。
	报警	
	转到故障	如果发现了故障或报警，则显示该软键。 该软键可从故障列表中打开选中的 XML 文件以便进行编辑。光标被自动置于故障行。
	检查结果	生成的所有信息都可加以查看。
对话框“文档”	保存	保存 XML 文件中的更改。
	取消	XML 文件不更改，然后关闭。检查结果会再次显示。

3.7 XML 标签

3.7.1 通用结构

用于对话框配置的脚本文件结构与指令

所有的对话框配置都保存在 **DialogGui** 标签中。

```
<DialogGui>
…
</DialogGui>
```

示例：

```
<?xml version="1.0" encoding="utf-8"?>
<DialogGui>
…
<FORM name ="Hello_World">
<INIT>
<CAPTION>Hello World</CAPTION>
</INIT>
...
</FORM>

</DialogGui>
```

指令

语言为完成有条件指令与完成程序循环提供有下列指令：

- FOR 循环
- While 循环
- Do-While 循环
- 有条件处理
- Switch 指令和 Case 指令
- 对话框窗口中的操作单元
- 软键说明
- 定义变量

指令的详细说明，请参见"指令/标签说明 (页 46)"一章。

3.7.2 指令/标签说明

为了创建对话框和菜单以及处理程序顺序需要定义下列 **XML 标签**：

说明

引号"<...>"表示属性值，可替换为实际使用的表达式。

示例：

<DATA_LIST action="read/write/append" id="<list name>">
会进行如下的参数设置：

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

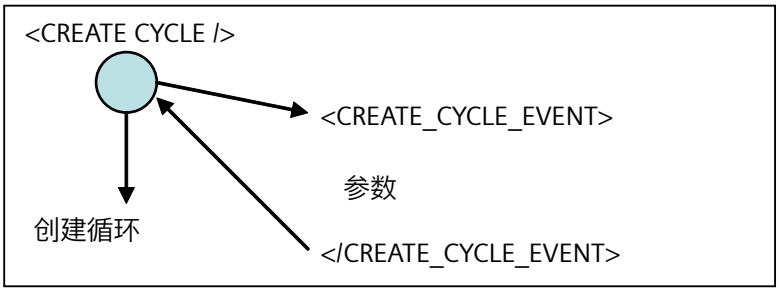
在复制多行 XML 标签和示例时要注意，不要使引号因不必要的换行而断开。

标签名称	含义
BREAK	有条件中断一个循环。

CONDITION	该标签可进行单行编写；也可进行多行编写，此时要使用标签名对条件进行分行。
	
	示例：
	`<if>`
	`<condition>test <= 2</condition>`
	`...`
	
	或
	
	`<if>`
	`<condition>`
	`test <= 2`
	`</condition>`
	`...`

CONTROL	该标签用来创建控件。详细说明请见章节"创建软键菜单和对话框窗口 (页 85)"。

标签名称	含义
CONTROL_RESET	该标签可以重新启动一个或多个控制系统组件。 **语法：** `<CONTROL_RESET resetnc="TRUE" />` **属性：** • **RESETNC = "TRUE"** 重新启动 NC 组件 • **RESETDRIVE = "TRUE"** 重新启动驱动组件。

标签名称	含义
CREATE_CYCLE_EVENT	解析器处理标签 CREATE_CYCLE 时，会首先将信息<CREATE_CYCLE_EVENT>发送到当前 FORM 标签中。在解析器根据参数表和创建规则生成 NC 指令之前，该信息会被用于循环参数的创建准备。  **语法：** <CREATE_CYCLE_EVENT> … … </CREATE_CYCLE_EVENT> **示例：** <SOFTKEY_OK> … <CREATE_CYCLE /> … … <SOFTKEY_OK> … … <FORM> <NC_INSTRUCTION>MY_CYCLE($P1, $P2)</ NC_INSTRUCTION > … … <CREATE_CYCLE_EVENT> <type_cast name="P1" type="int"/> <OP> P1 = P1 * 150 </OP> </CREATE_CYCLE_EVENT> … … </FORM>

标签名称	含义
DATA	该标签用于写入 NC、PLC、GUD 和驱动数据。 地址构成可以查阅章节 "组件地址分配 (页 157)"。 **属性：** • **name** 变量地址 **标签值：** 标签值可以为常量、**变量或 term**。 **语法：** <DATA name="<variable name>"> value </DATA> <DATA name="<variable name>"> <term> </DATA> **示例：** <DATA name = "plc/mb170"> 1 </DATA> … <LET name = "tempVar"> 7 </LET> <!– 局部变量"tempVar"的内容会写入到存储器字节 170 中 → <DATA name = "plc/mb170">$tempVar</DATA>
DATA 续	**示例：** Term 的计算。计算结果会分配给指定的变量。 <let name="a" >#f</let> <let name="b" >7</let> <data name = "b"> a & b</data> <let name="c" type="string"></let> <data name = "c"> _T" test" + _T" and test"</data>

标签名称	含义

DATA_LIST	该标签可以保存或者恢复所列出的驱动数据和机床数据。
	按行方式列出地址。地址构成可以查阅章节 “组件地址分配 (页 157)”。
	可以创建最多 20 个临时数据列表。
	
	属性：
	• **action**
	read - 将列出变量的值存储到一个临时存储器中
	append - 将列出变量的值添加到已有列表中
	write - 将保存的值复制到相应的机床数据中
	• **id**
	标签用于识别临时存储器
	语法：
	`<DATA_LIST action="<read/write/append>"  id="<list name>">`
	`NC/PLC 地址汇编`
	`</DATA_LIST>`
	
	示例：
	`<DATA_LIST action ="read" id="<name>">`
	`nck/channel/parameter/r[2]`
	`nck/channel/parameter/r[3]`
	`nck/channel/parameter/r[4]`
	`$MN_USER_DATA_INT[0]`
	`…`
	`</ DATA_LIST>`
	`<DATA_LIST action ="write" id="<name>" />`

DIALOGGUI	该标签显示了 Easy XML 的 Root 标签。可通过 Easy XML 解析器再编辑所有已完成的指令。 **属性：** 可以选择性输入提及的属性，使能集体函数： • **diagnose** - 值为 **true** 表示激活 XML 诊断 • **debug_msg** - 值为 “on”表示保存跟踪记录 **更多信息**参见章节 “XML 诊断 (页 43)” • **textfile** - 要使用的文本文件名称 • **textcontext** - 要使用的文本内容，只有与属性 textfile 一同使用时才有效 • **textfilelist** - 可以加载含不同文本文件的列表 **更多信息**参见章节 “特定项目的文本文件 (页 284)”

标签名称	含义
DIALOGGUI 续	• **restoresession** - **真值** 如果再次选择 Easy XML 项目，解析器会打开最后选择的菜单以及最后选择的窗体。该属性会持续至重新启动 HMI。 **更多信息**参见章节“恢复会话 (页 292)”
DEBUG_MSG	此属性控制跟踪输出数据的存储。描述参见 DEBUG 标签下的章节“创建软键菜单和对话框窗口 (页 85)”。 若值为**打开**，则 Parser 将所有输出保存在一个文件中。这样有可能导致脚本执行变慢。缺省设置为**关闭**。

DO_WHILE	Do-While 循环
	`DO`
	`  指令`
	`WHILE （测试）`
	
	语法：
	`<DO_WHILE>`
	`指令`
	`…`
	`<CONDITION>…</CONDITION>`
	`</DO_WHILE>`
	Do-While 循环由一个指令块和一个条件构成。首先执行指令块中的代码，然后计算条件。如果条件为真（true），则重新执行代码部分。一直重复，直至条件为假（false）。
	
	示例：
	`<DO_WHILE>`
	`<DATA name = "PLC/qb11"> 15 </DATA>`
	`<CONDITIION> "plc/ib9" == 0 </CONDITION>`
	`</DO_WHILE>`

标签名称	含义
DYNAMIC_INCLUDE	标签包含 XML 脚本文件。 与标签 INCLUDE 相反，在执行相应指令时才进行读取。 在大型项目中该标签的使用能缩短用户使用或循环支持的载入时间。此外，对系统资源的平均需求也会下降，因为在会话过程中不需要一直调用所有的对话框。 **语法：** <DYNAMIC_INCLUDE src="path name"/> **示例：** <SOFTKEY POSITION="3"> <CAPTION>MY_MENU</CAPTION> <DYNAMIC_INCLUDE src="f:\appl\sk3_script.xml"/> <NAVIGATION>MY_MENU</NAVIGATION> </SOFTKEY>
ELSE	满足条件时的指令（**IF（如果）**、**THEN（则）**、**ELSE（否则）**）

标签名称	含义
FOR	FOR 循环 for （初始化；测试；继续）指令 **语法：** <FOR> <INIT>…</INIT> <CONDITION>…</CONDITION> <INCREMENT>…</INCREMENT> 指令 … </FOR> 按下列方式执行 For 循环： 1. 运用表达式 **初始化** （INIT）。 2. 运用表达式 **文本** （CONDITION） 作为布尔型表达式。 如果值为假（false），则结束 For 循环。 3. 执行后续的指令。 4. 运用表达式 **继续** （INCREMENT）。 5. 继续使用 2。 所有在 INIT、CONDITION 和 INCREMENT 分支中使用的变量都要在 For 循环的外部创建。 **示例：** <LET name = "count">0</LET> <FOR> <INIT> <OP> count = 0</OP> </INIT> <CONDITION> count <= 7 </CONDITION> <INCREMENT> <OP> count = count + 1 </OP> </INCREMENT> <OP> "plc/qb10" = 1+ count </OP> </FOR>
FORM	该标签包含对用户对话框的说明。详细说明请见章节“创建软键菜单和对话框窗口 (页 85)”。
HMI_RESET	该标签会触发 HMI 重新启动。 编译在该指令后会被中断。

标签名称	含义
IF	有条件的指令（**IF, THEN, ELSE**） 标签 THEN 和 ELSE 包含在标签 IF 中。 标签 IF 后跟有条件，而条件在标签 CONDITION 中执行。需要通过其他的指令处理来确定运算结果。如果函数结果为真，则执行 THEN 分支而跳过 ELSE 分支。如果函数结果为假，则解析器执行 ELSE 跳转。 **语法：** <IF> <CONDITION> 条件 != 7 </CONDITION> <THEN> 情况说明：条件满足 </THEN> <ELSE> 情况说明：条件不满足 </ELSE> </IF> **示例：** <IF> <CONDITION> "plc/mb170" != 7 </CONDITION> <THEN> <OP> "plc/mb170" = 7 </OP> … </THEN> <ELSE> <OP> "plc/mb170" = 2 </OP> … </ELSE> </IF>

标签名称	含义

IF 续	如果条件中仅使用了局部变量，则可将该条件作为属性输入 IF 标签。
	
	示例：
	`<let name="var1">1</let>`
	`<if condition="var == 1">`
	`<then>`
	`...`
	`...`
	`</then>`
	`</if>`
	
	提示：
	不与参考变量进行关联的控件应分配为整数型。
	特例：
	imagebox 和 **multiline** 类型的 Control 变量分配的数据类型为 **string**。

INCLUDE	指令包含有 XML 说明。 （另见该表格中的 DYNAMIC_INCLUDE ） **属性：** • **src** 包含路径名。 **语法：** `<?INCLUDE src="<Path name>" ?>`

标签名称	含义
LET	指令以给出的名称创建一个局部变量。 **数组：** 通过属性 **dim** （维数）可创建一维或二维的数组。各个数组元素的寻址通过数组下标进行。 对于二维数组，先指定行下标，再指定列下标。 • 一维数组： 下标 0 至 4 0 1 2 3 4 • 二维数组： 行下标 0 至 3，列下标 0 至 5 0,0 0,1 0,2 0,3 0,4 0,5 1,0 1,1 1,2 1,3 1,4 1,5 2,0 2,1 2,2 2,3 2,4 2,5 3,0 3,1 3,2 3,3 3,4 3,5 **属性：** • **name** 变量名称 • **type** 变量类型可以为整数型（INT）、无符号整数型（UINT）、双精度型（DOUBLE）、浮点型（FLOAT）、字符串型（STRING）或STRUCT。此外，可以将通过 **typedef** 创建的结构用作变量类型（参见标签名称 TYPEDEF）。如果未给出类型指令，则系统会创建一个整数型变量。 `<LET name = "VAR1" type = "INT" />` • **permanent** 如果属性为 **true**，则永久保存变量值。该属性仅对全局变量有效。 • **poweroff** 如果属性值为 **true**，则数值会一直保留至控制系统关闭。 **更多信息**参见章节“恢复会话 (页 292)” • **dim** 用来指定数组元素的数量。对于二维数组，用逗号分隔第一维和第二维。对数组元素的存取通过数组下标进行，写在变量名称后的方括号内。 name[index] 或 name[row,column] – 一维数组：dim="<元素数量>"

标签名称	含义
	– 二维数组：dim="<行数>,<列数>" 未赋初值的数组元素用“0”占用。

标签名称	含义

LET 续	**示例：**
	一维数组：
	`<let name="array" dim="10"></let>`
	二维数组：
	`<let name="list_string" dim="10,3" type="string"></let>`
	
	预设值：
	可以对变量进行初始化赋值。
	`<LET name = "VAR1" type = "INT"> 10 </LET>`
	
	如果将 NC 或 PLC 变量的值保存在一个局部变量中，则赋值运算会自动将格式与所读入变量的格式进行匹配。
	• 字符串变量的预设值：
	只要将格式化的文本作为值进行传送，就可以为一个字符串变量赋值多行文本。如果一行要用 line feed <LF>（换行）结尾，应在行尾添加字符 "\\n"。
	`<LET name = "text" type = "string"> F4000 G94\\n`
	`G1 X20\\n`
	`Z50\\n`
	`M2\\n`
	`</LET>>`
	
	数组 (Arrays)：
	`<let name="list" dim="10,3">`
	`  {1,2,3},`
	`  {1,20}`
	`</let>`
	
	`<let name="list_string" dim="10,3" type="string">`
	`  {"text 10","text 11"},`
	`  {"text 20","text 21"}`
	`</let>`
	
	赋值：
	可以使用赋值运算"="将机床数据或子程序中的值赋值给一个变量。
	变量的有效性会一直延伸到最上一级的 XML 数据块。
	需要全局使用的变量应直接在标签 **DialogGUi** 后创建。
	对于对话框须注意以下事项：
	• 处理信息会打开相应的标签。
	• 在信息执行后会关闭标签。
	• 标签中的所有变量会随着关闭而被清零。

标签名称	含义

LET 续	**变量类型 struct：**
	该变量类型包含在使用结构名称时所能响应的变量汇总。一个结构能包含所有变量类型以及结构。
	结构中的变量通过标签“element”进行说明。标签的属性和初始化与 let 说明的属性和初始化相符。
	
	`<let name="name" type="struct">`
	`<element name="<名称>" type="<变量类型>" />`
	…
	…
	…
	`</let>`
	
	可为结构元素分配一个默认值，创建变量时该值会作为初始值使用。
	更多信息参见章节“结构初始化”
	如果在创建变量时指定了初始值，则该默认值会被覆写。

标签名称	含义
LET 续	通过结构和变量名称访问结构变量。两个名称由点隔开。 **语法：** <op> 结构_名称.变量_名称 = 值; </op> **示例：** <let name="info" type="struct"> <element name="id" type="int" /> <element name="name" type="string" /> <element name="phone" type="string" /> </let> <op> info.id = 1; info.name = _T"my name"; info.phone = _T"0034 45634"; </op>

标签名称	含义

LET 续	**结构初始化：**
	通过在结构元素中指定初始值，便可在创建变量时对结构进行初始化。在结构数组中，各个结构之间使用尖括号隔开。
	
	示例：
	`<typedef name="StructRect" type="struct" >`
	`  <element name="left" type="int">22</element>`
	`  <element name="top" type="int">122</element>`
	`  <element name="right" type="int">220</element>`
	`  <element name="bottom" type="int">56</element>`
	`</typedef>`
	`<let name="rect" type="StructRect" >`
	创建变量 **rect** 时，结构元素便会初始化为以下数值：
	`rect.left= 22`
	`rect.top = 122`
	`rect.right = 220`
	`rect.bottom = 56`
	
	如果为结构分配了初始值，这些值便会覆写默认值。
	`<let name="rect" type="StructRect" >`
	`  {11,12}`
	`</let>`
	创建变量 rect 时，结构元素便会初始化为以下数值：
	`rect.left= 11`
	`rect.top = 12`
	`rect.right = 220`
	`rect.bottom = 56`

标签名称	含义
LET 续	**嵌套结构：** 结构是由用户指定类型的变量，因此可作为结构元素在 struct 中使用。 元素的初始化规则如下所述。 **示例：** `<typedef name="StructRectRect" type="struct" >   <element name="r1" type="StructRect">77, 99, 232, 23</element>   <element name="r2" type="StructRect">77, 88, 45, 12</element> </typedef>` `<let name="rect_rect_array" type="StructRectRect" >` 创建变量 **rect_rect_array** 时，结构元素便会初始化为以下数值： `r1.left= 77` `r1.top = 99` `r1.right = 232` `r1.bottom = 23` `r2.left= 77` `r2.top = 88` `r2.right = 45` `r2.bottom = 12` `<let name="rect_rect_array1" type="StructRectRect" >   {{11,12,13,14},{111,188,199,114}} </let>` 创建变量 **rect_rect_array** 时，结构元素便会初始化为以下数值： `r1.left= 11` `r1.top = 12` `r1.right = 13` `r1.bottom = 14` `r2.left= 111` `r2.top = 188` `r2.right = 199` `r2.bottom = 114`
LET 续	**全局字符串变量的初始化：** 如果在初始化全局字符串变量时使用了文本标识符，那么该标识符和要替换的文本则应保存在标准文本文件 **oem_xml_screens_xxx.ts** 或在标签 **DialogGUI** 中指定的文本文件中。在加载应用时，文本标识符便会替换为生效语言的相应文本。如果切换语言，那么在应用激活期间，不会对全局字符串变量重新进行初始化。文本可在消息 **LANGUAGE_CHANGED** 中进行替换。

标签名称	含义
LOCK_OPERATING_AREA	禁止操作区域切换。 使用该标签 **UNLOCK_OPERATING_AREA** 可取消禁止操作区域切换。 **语法：** `<LOCK_OPERATING_AREA />`
MSG	操作组件显示标签中所给出的信息。 如果使用报警编号，则对话框会显示为该编号所保存的文本。 **示例：** `<MSG text ="my message" />`

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

标签名称	含义

OP	该标签会执行给定的操作。
	可以执行章节“运算符 (页 83)”中所列出的各种运算。
	对于 NC/PLC 上的存取和驱动数据要在 **引用字符** 中设置完整的变量名称。地址构成可以查阅章节 “组件地址分配” (页 157) 。
	`PLC: "PLC/MB170"`
	`NC: "NC/Channel/…"`
	
	示例：
	`<LET name = "tmpVar" type="INT"> </LET>`
	`<OP> tmpVar = "plc/mb170" </OP>`
	`<OP> tmpVar = tmpVar *2 </OP>`
	`<OP> "plc/mb170" = tmpVar </OP>`
	
	一个运算标签内可以使用多个等式。用分号标记指令结束。
	
	示例：
	`<op>`
	`x = x+1;`
	`y = y+1;`
	`</op>`
	
	字符串处理：
	运算指令可以对字符串进行处理并将结果字符串赋值给等式中所给出的字符串变量。
	为了标记文本表达式需要在文本前放置标志 _T 。此外还能对变量值进行格式化。格式规定以标志 _F 开始，后面跟有格式指令。接着给定变量地址。
	
	示例：
	`<LET name="buffer" type="string"></LET>`
	`…`
	`…`
	`…`
	`<OP> buffer = _T"unformatted value R0= " + "nck/Channel/Parameter/R[0]" + _T" and " + _T"$$85051" + _T" formatted value R1 " + _F %9.3f"nck/Channel/Parameter/R[1]" </OP>`

标签名称	含义

OPERATION	**Operation**
	一个等式内可以使用一个移位操作。
	向左移动运算符 "<<"
	通过函数 << 向左移动位。可直接或通过一个变量确定待移动值和移动数。如果达到数据格式极限，位会超出极限移动，此时，系统不显示错误信息。
	运算法则
	从左往右算且 '+' 和 '-' 位于 '<<' 或 '>>' 前
	
	示例：
	`<op> idx = idx << 2 </op>`
	`<op> idx = 3 + idx << 2 </op>`
	
	向右移动运算符 ">>"
	通过函数 >> 向右移动位。可直接或通过一个变量确定待移动值和移动数。如果达到数据格式极限，位会超出极限移动，此时，系统不显示错误信息。
	运算法则
	从左往右算且 '+' 和 '-' 位于 '<<' 或 '>>' 前
	
	示例：
	`<op> idx = idx >> 2 </op>`
	`<op> idx = 3+idx >> 2</op>`

标签名称	含义

PASSWORD	在"EasyExtend"功能中，该标签用于启用附加设备。
	此处输入的字符串会写入指定的参考变量中，在 PLC 用户程序中处理。
	
	语法：
	`<PASSWORD refVar ="<variable name>" />`
	
	属性：
	• **refVar**
	参考变量的名称
	• **text**
	属性说明替换标准文本（可选）
	
	示例：
	`<PASSWORD refvar="plc/mw107" />`
	
	可选示例：
	`<password refVar = "plc/MD108" text="Password" />`

POWER_OFF	显示信息要求操作者关闭机床。显示文本固定保存在系统中。

标签名称	含义

PRINT	标签会在对话行中输出文本或者将文本复制到给定的变量中。
	如果文本包含有格式化标志，则会将变量值插入到相应的位置上。
	
	语法：
	`<PRINT name="变量名称" text="text %格式化"> Variable, … </PRINT>`
	`<PRINT text="text %格式化"> Variable, … </PRINT>`
	属性：
	• **name**
	保存文本的变量的名称（可选）
	• **text**
	文本
	
	格式化：
	字符"%"会对作为值给定的变量进行格式化。
	%[标记] [宽度] [.小数点后面的位置] 类型
	• 标记：
	用于确定输出格式的可选字符：
	– 右对齐或左对齐（"-" 用于左对齐）
	– 前面加零（"0"）
	– 使用空格符填充
	• 宽度：
	确定一个非负数最小输出宽度的依据。如果待输出值占用的位置少于依据所确定的位置，则使用空格符填充空缺的位置。
	• 小数点后面的位置：
	使用浮点数时优化参数用来确定小数点后面的位置数。
	• 类型：
	类型字符用来确定打印指令要输出哪些数据格式。该字符必须给定。
	– d:整数值
	– f：浮点数
	– s:字符串

标签名称	含义

PRINT 续	**值：**
	其值应当插入到文本中的变量数量。
	变量类型必须与格式规定的相应类型标识一致并用逗号加以间隔。
	
	示例：
	在信息行中输出文本
	`<PRINT text="信息文本" />`
	使用变量格式输出文本
	`<LET name="trun_dir"></LET>`
	`<PRINT text="M%d">trun_dir</PRINT>`
	在字符串变量中使用变量格式输出文本
	`<LET name="trun_dir"></LET>`
	`<LET name="str" type="string" ></LET>`
	`<print name="str" text="M%d ">trun_dir</print>`

PROGRESS_BAR	标签打开或关闭一根进度条。进度条显示在应用窗口的下方。
	
	语法：
	`<PROGRESS_BAR type="<true/false>"> value </ PROGRESS_BAR>`
	
	属性：
	• **type =** “TRUE” - 打开进度条
	• **type** = “FALSE” - 关闭进度条
	• **min**
	（可选）- 最小值
	• **max**
	（可选）- 最大值
	
	值：
	• **Value**
	进度条的百分比位置
	示例：
	`<PROGRESS_BAR type="true" min="0" max="101" >20< /`
	`PROGRESS_BAR>......<PROGRESS_BAR >50< /`
	`PROGRESS_BAR>......<PROGRESS_BAR type="false" >100< /PROGRESS_BAR>`

标签名称	含义

SEND_MESSAGE	该标签会向有效的 FORM 标签发送含两个参数的信息，该信息在标签“Message”中进行处理。
	
	语法：
	`<SEND_MESSAGE>p1, p2</SEND_MESSAGE>`
	
	示例：
	`<SOFTKEY POSITION="3">`
	`  <caption>Set%nParameter</caption>`
	`  <send_message>1, 0</send_message>`
	`</SOFTKEY>`
	
	`<FORM>`
	…
	…
	`<MESSAGE>`
	`<SWITCH>`
	`<CONDITION>$message_par1</CONDITION>`
	`<CASE value="1">`
	…
	…
	`</CASE>`
	`</SWITCH>`
	`</MESSAGE>`
	…
	…
	
	`</FORM>`

标签名称	含义

SHOW_CONTROL	借助该标签可以指定是否显示控件。
	
	语法：
	`<SHOW_CONTROL name="<name>" type="<type>" />`
	
	属性：
	• **name**
	控件的名称
	• **type =** “TRUE” - 控件可见
	• **type**= “FALSE” - 控件不可见（隐藏）
	
	示例：
	`<SHOW_CONTROL name="myEditfield" type="false" />`
	…
	…
	`<SHOW_CONTROL name="myEditfield" type="true" />`

SLEEP	该标签会在指定的时间区间内中断脚本的执行。中断时间为返回值乘以 50ms 的基准时间。
	
	语法：
	`<SLEEP value="中断时间" />`
	
	示例：
	等待时间 1.5 s。
	`<SLEEP value="30" />`

STOP	解释在该位置中断。

标签名称	含义

SWITCH	指令 **SWITCH** 表示多项选择。表达式运用一次并与常数进行比较。如果表达式与常数一致，则执行 **CASE** 指令中的指令。
	如果常数与表达式不一致，则执行 **DEFAULT** 指令。
	
	语法：
	`<SWITCH>`
	`<CONDITION> 值 </CONDITION>`
	`<CASE value="<常量 1>">`
	指令
	…
	`</CASE>`
	`<CASE value="<常量 2>">`
	指令
	…
	`</CASE>`
	`<DEFAULT>`
	指令
	…
	`</DEFAULT>`
	`</SWITCH>`

标签名称	含义

SWITCHTOAREA	SWITCHTOAREA 标签从用户操作区切换至指定的操作区。
	参数指定为属性值。
	
	语法：
	`<switchToArea name="area" args="argument "/>`
	
	属性：
	name
	以下操作区名称与操作区一致：
	• **AreaMachine** - 机床
	• **AreaParameter** - 参数
	• **AreaProgramEdit** - 编辑器
	• **AreaProgramManager** - 程序管理器
	• **AreaDiagnosis** - 诊断
	• **AreaStartup** - 调试
	args（预留）
	激活与对话框相应的运行时间自变量时派定一个操作区。
	
	示例：
	`<switchToArea name="AreaMachine" />`

SWITCHTODYNAMICTARGET	如果将之前的对话框定义为动态跳转目标，则 SWITCHTODYNAMICTARGET 标签会激活该对话框并结束脚本编辑。 **语法：** `<switchToDynamicTarget />`
THEN	当条件满足时应执行的指令（**IF, THEN, ELSE**）

标签名称	含义
TYPE_CAST	该标签用于控制局部变量的数据类型转换。 **语法：** <type_cast name="variable name" type=" new type" /> **属性：** • **name** 变量名称 • **type** 给变量赋值新的数据类型。 • **convert** 给变量赋值新的数据类型。此外，变量值转换为新的数据类型。

标签名称	含义

TYPEDEF	通过标签可为数据类型确定一个新的名称。这对于结构定义的优势在于只要定义一次数据类型，然后便能在 LET 指令中作为数据类型使用。
	作为属性需要具有名称和类型。
	解析器只支持轮廓定义的确定。
	
	类型定义中的变量通过标签“element”进行声明。标签的属性与 let 命令的属性一致。
	
	`<typedef name="<名称>" type="struct">`
	`  <element name="<名称>" type="<变量类型>" />`
	…
	…
	…
	`</typedef>`
	
	定义之后，名称可作为 LET 命令的数据类型使用。
	
	`<let name="<变量名称>" type="<名称>"></let>`
	
	示例:
	`<typedef name="my_struct" type="struct">`
	`  <element name="id" type="int" />`
	`  <element name="name" type="string" />`
	`  <element name="phone" type="string" />`
	`</typedef>`
	
	`<let name="info" type="my_struct"></let>`
	…
	…
	`<op>`
	`  info.id = 1;`
	`  info.name = _T"my name";`
	`  info.phone= _T"0034 45634";`
	`</op>`

标签名称	含义
TYPEDEF 续	作为调用参数，某些预定义的函数需要使用“结构型”的变量 RECT、POINT 或 SIZE。该结构仅在文件 struct_def.xml 中定义。 **更多信息**参见章节“创建 struct_def.xml 文件 (页 155)” **RECT:** `<typedef name="StructRect" type="struct" > <element name="left" type="int">0</element> <element name="top" type="int">0</element> <element name="right" type="int">0</element> <element name="bottom" type="int">0</element> </typedef>` **POINT:** `<typedef name="StructPoint" type="struct" > <element name="x" type="int">0</element> <element name="y" type="int">0</element> </typedef>` **SIZE:** `<typedef name="StructSize" type="struct" > <element name="width" type="int">0</element> <element name="height" type="int">0</element> </typedef>`
UNLOCK_OPERATING_AREA	取消禁止切换操作区域 **语法：** `<UNLOCK_OPERATING_AREA />`

标签名称	含义

WAITING	标签在 NC 或者驱动复位后等待组件重新启动。
	
	属性：
	• **WAITINGFORNC** = "TRUE" - 等待 NC 重启
	• **WAITINGFORDRIVE** = "TRUE" - 等待驱动重启
	
	语法：
	`<WAITING WAITINGFORNC = "TRUE"/>`
	
	示例：
	...
	`<CONTROL_RESET resetnc = "true" resetdrive = "true"/>`
	`<WAITING waitingfornc = "true" waitingfordrive = "true" />`
	...

WHILE	While 循环
	`WHILE (Test)`
	`  指令`
	
	语法：
	`<WHILE>`
	`<CONDITION>…</CONDITION>`
	指令
	…
	`</WHILE>`
	While 循环用于按顺序多次执行指令，只要满足条件。在处理指令序列前对条件进行检查。
	
	示例：
	`<WHILE>`
	`<CONDITION> "plc/ib9" == 0 </CONDITION>`
	`<DATA name = "PLC/qb11"> 15 </DATA>`
	`</WHILE>`

标签名称	含义
XML_PARSER	标签 “XML_PARSER” 可用于解析 XML 文件。 解析器编译 XML 文件并调用定义的回调函数。每个回调函数都属于一个预定义事件。编程人员可在该函数内部处理 XML 文件。 **预定义事件：** • start document 解析器打开文档并开始解析。 • end document 解析器关闭文档。 • start element 解析器找到了一个元素并创建了一个含所有属性和属性值的列表。 这些列表被转送给回调函数。 • end element 找到元素末尾。 • characters 解析器转送元素的所有字符。 • error 解析器发现了一个语法错误。 出现一个事件时，解析器会调用回调函数并检查函数的返回值。函数返回值“true”时，解析器会继续进程。

标签名称	含义

XML_PARSER 续	**接口**
	属性值 **name** 包含 XML 文件的路径。
	为给回调函数分配事件，须确定以下属性：
	标准
	startElementHandler
	endElementHandler
	charactersHandler
	可选
	errorHandler
	documentHandler
	属性值用于定义回调函数的名称。
	
	示例：
	`<XML_PARSER name="f:\appl\xml_test.xml">`
	`<!-- standard handler -->`
	`<property startElementHandler="startElementHandler" />`
	`<property endElementHandler="endElementHandler" />`
	`<property charactersHandler="charactersHandler" />`
	`<!-- optional handler -->`
	`<property errorHandler="errorHandler" />`
	`<property documentHandler="documentHandler" />`
	`</XML_PARSER>`

标签名称	含义

XML_PARSER 续	此外，解析器还会提供可使回调函数访问事件数据的变量。
	
	startElementHandler:
	函数参数
	tag_name - 标签名称
	num - 找到的属性数
	系统变量
	$xmlAttribute 具有一定数目的通过 num 给定的单元的字符串数组。
	$xmlValue 属性值范围为 0-num 的字符串数组。
	
	示例：
	`<function_body name="startElementHandler" return="true" parameter="tag_name, num">`
	`<print text="attribute name %s"> $xmlAttribute[0]</print>`
	`<print text="attribute value %s"> $xmlValue[0]</print>`
	`</function_body>`
	
	endElementHandler:
	函数参数
	tag_name - 标签名称
	
	示例：
	`<function_body name="endElementHandler" parameter="tag_name">`
	`<print text="name %s"> tag_name </print>`
	`</function_body>`

标签名称	含义

XML_PARSER 续	**charactersHandler:**
	系统变量
	$xmlCharacters 含数据的字符串
	$xmlCharactersStart 始终为 0
	$xmlCharactersLength 字节数
	
	示例：
	`<function_body name="charactersHandler" return="true" >`
	`<print text="chars %s"> $xmlCharacters </print>`
	`</function_body>`
	
	documentHandler:
	函数参数
	state 1 start document，2 end document
	
	errorHandler:
	系统变量
	$xmlErrorString 包含无效行（系统变量）
	
	示例：
	`<function_body name="errorHandler" >`
	`<print text="error %s">$xmlErrorString</print>`
	`</function_body>`
	
	回调结果：
	$return 如果为 1 (true)，分析程序会继续分析文件

3.7.3 颜色代码

颜色属性使用 HTML 语言的颜色代码搭配。

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

R – 红色

G – 绿色

B – 蓝色

#RRGGBB

示例：

color= "#ff0011"

示例颜色：

红色	绿色	蓝色	黄色	白色	黑色
#FF0000	#00FF00	#0000FF	#ffff00	#FFFFFF	#000000

3.7.4 专用 XML 句法

如果要在通用 XML 编辑器中正确显示在 XML 句法中有特殊含义的字符，则必须进行写法转换。

关系到下列字符：

字符	XML 中的表示法
<	&lt;
>	&gt;
&	&amp;
"	&quot;
'	&apos;

3.7.5 XML 特殊字符

为了显示字符或使用控制符，提供了对 HTML 特殊字符的识别。

可以使用代码页“Latin 1”的十进制和十六进制编码。

示例

字符	说明	十进制符号	十六进制符号
"	上引号	"	"
LF	换行（Line Feed）

3.7.6 运算符

运算指令可以处理下列运算：

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

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

3.7.7 系统变量

系统变量是在各个脚本中使用的变量，用于解析器和脚本执行过程之间的数据交换。

下表为自动生成的变量与标签对应关系一览：

变量名称	含义	适用标签
$actionresult	解析器报告，是否要通过解析器处理事件	KEY_EVENT
$focus_name	包含当前所要输入的数组的名称	FOCUS_IN INDEX_CHANGED EDIT_CHANGED
$focus_item_data	包含已分配给数组的数字值 item_data	
$gestureinfo	针对手势操作：解析器会在变量 $gestureinfo 中提供手势信息，并执行标签 gesture_event。	GESTURE_EVENT
$return	变量用于传输间接函数的返回值	FUNCTION_BODY
$message_par1 $message_par2	包含函数 SEND_MESSAGE 的调用参数	MESSAGE
$xmlAttribute	包含已找到的标签属性的列表	XML_PARSER startElementHandler
$xmlValue	包含已找到的标签值的列表	
$xmlCharacters	包含数据流	XML_PARSER charactersHandler
$xmlCharactersStart	包含起始下标	
$xmlCharactersLength	包含数据流中保存的字符的数量	
$mouse_event.type $mouse_event.x $mouse_event.y $mouse_event.id $mouse_event.buttons $mouse_event.button	鼠标事件参数的传输结构	MOUSE_EVENT

3.7.8 创建软键菜单和对话框窗口

欲链接对话框窗口，在 XML 说明中应含有名称为“main”的菜单标签。在激活操作区 <CUSTOM> 后由系统调用该标签。在标签之中可以定义其他的对话框窗口以及激活一个对话框。

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

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

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

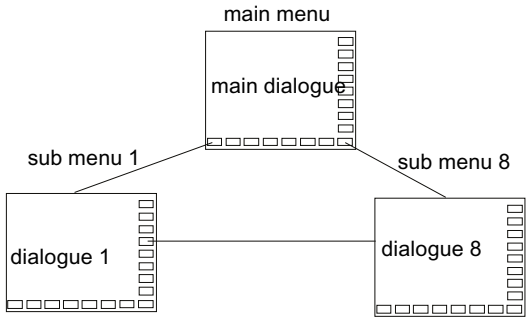


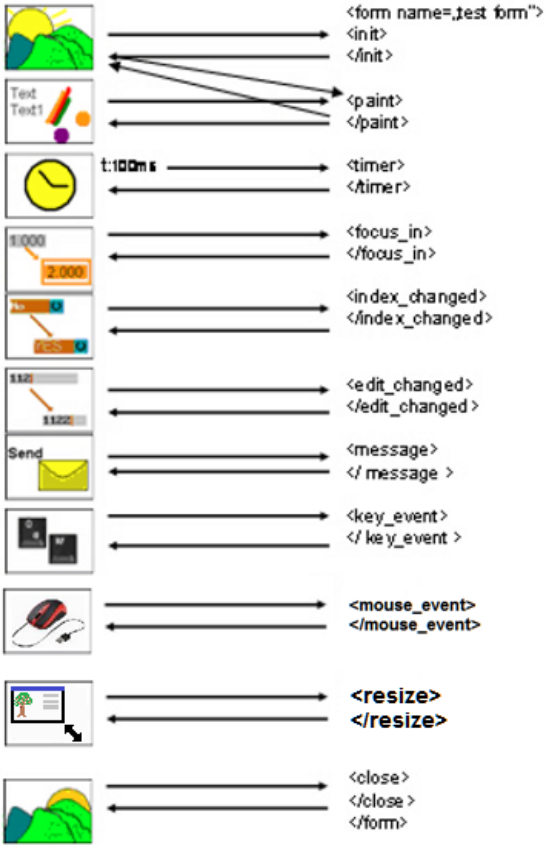
图 3-6　菜单结构

标签名称	含义
FORM	该标签包含对用户对话框的说明。其他相关的标签见章节“创建菜单与对话框窗口”中的说明。 **语法：** `<FORM name="<dialog name>" color="#ff0000">` **属性：** • **color** 对话框背景色 **更多信息**参见章节“颜色代码 (页 81)” – 了解缺省值 • **name** 格式的名称 • **type** 允许的值是 *cycle* ，表示一个用户循环窗口 • **xpos** 对话框左上角的 X 坐标（可选） • **ypos** 左上角的 Y 坐标（可选） • **width** X 方向上的长度（以像素为单位）（可选） • **height** Y 方向上的长度（以像素为单位）（可选）

标签名称	含义
FORM 续	通过 AUTOSCALE_CONTENT 属性可以确定不同屏幕分辨率时 Form 控件的性能。默认情况下，控件自动适应现有的屏幕分辨率。如果想自行控制位置和尺寸，则需将 Form 标签的属性值设为 OFF。 **属性：** **autoscale_content** **值：** • **on** 控件的坐标自动与屏幕分辨率相适应（默认） • **off** 控件的坐标使用时不会变化 **示例：** <form name="main_form" autoscale_content="off"> … … … </form>
FORM 续	属性： • **textfile** 该属性可以指定窗体相关的文本文件。系统使用名称 EASY_XML 作为标准文本内容。 • **textcontext** 该属性可以指定要使用的文本内容的名称。此属性只有与属性 **textfile** 一同使用时才有效。

标签名称	含义

FORM 续	要使用文件 easyscreen.ini 中保存的标准版式，需要使用属性 LAYOUT。此处要给出的值是要使用的版式的名称。
	提示：
	默认设置的更改只对于弹出窗口生效，因为这里调用的窗口未隐去。
	
	属性：
	layout
	
	语法：
	`<form name="窗体名称" layout="版式名称" >`
	…
	…
	`</form>`
	
	示例：
	`<form name="form0" layout="slstandardscreenlayout.`
	`SlStandardScreenLayout.LowerForm" >`
	…
	…
	`</form>`
	
	除了预定义的窗口位置和尺寸之外，还可以在文件 easyscreen.ini 中保存自定义。
	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`
	
	示例：
	`<form name="form0" layout="MyPanel" >`
	…
	…
	`</form>`

标签名称	含义
FORM 续	**对话框信息：** • **INIT** • **PAINT** • **TIMER** • **CLOSE** • **FOCUS_IN** • **INDEX_CHANGED** • **EDIT_CHANGED** • **GESTURE_EVENT** • **KEY_EVENT** • **MESSAGE** • **MOUSE_EVENT** • **RESIZE** • **CHANNEL_CHANGED** • **LANGUAGE_CHANGED**
FORM 续	

标签名称	含义

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

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

标签名称	含义

KEY_EVENT	对话框信息
	标签 KEY_EVENT 可将对键盘事件的评估链接到标签 FORM 中。如果在 FORM 中存在该标签，则系统会将 MF2 键盘代码发送给有效的 FORM。如果变量 **$actionresult** 未置零，系统会接着处理键盘事件。
	键盘代码会作为整型值在变量 **$keycode** 中提供。
	
	示例：
	在变量 **exclude_key** 中输入的信号应从输入电流中剔除。
	
	`<LET name="stream" type="string"/>`
	`<LET name="exclude_key" type="string"/>`
	
	`<FORM name = "keytest_form">`
	`  <INIT>`
	`  <CONTROL name = "p1" xpos = "120" ypos = "84" width ="200" refvar="stream" hotlink="true" />`
	`  <CONTROL name = "p2" xpos = "160" ypos = "104" width ="8" refvar="exclude_key" hotlink="true" />`
	`  </INIT>`
	
	`  <PAINT>`
	`  <text xpos = "8" ypos = "84">data stream</text>`
	`  <text xpos = "8" ypos = "104">exclude key</text>`
	`  </PAINT>`
	
	`  <KEY_EVENT>`
	`  <LET name="excl_keycode" type="string"/>`
	`  <OP>excl_keycode = exclude_key</OP>`
	`  <type_cast name="excl_keycode" type="int" />`
	`  <PRINT text="%d %d">$keycode, excl_keycode</PRINT>`
	
	`  <IF>`
	`  <CONDITION>$keycode == excl_keycode</CONDITION>`
	`  <THEN>`
	`  <op> $actionresult = 0</op>`
	`  </THEN>`
	`  </IF>`
	
	`  </KEY_EVENT>`
	
	`</FORM>`

标签名称	含义

MOUSE_EVENT	该标签可将对鼠标事件的编辑链接到脚本中。具有以下鼠标操作时执行：
	• 某个按键被按下
	• 某个按键被松开
	• 鼠标被移动
	解析器在结构中提供信息并创建带有以下元素的结构变量**$mouse_event**：
	
	结构元素：
	• **type**
	操作的编码
	– 2 - 某个按键被按下
	– 3 - 某个按键被松开
	– 5 - 鼠标被移动
	• **x**
	基于当前屏幕分辨率的光标的 X 位置，单位：像素
	• **y**
	基于当前屏幕分辨率的光标的 Y 位置，单位：像素
	• **id**
	名称
	– -1，该位置没有分配控件时
	– != -1，鼠标光标位于控件内时，提供属性 **idemdata**
	• **button**
	包含事件时间点时按键的状态
	– 0 - 无按键
	– 1 - 左侧按键
	– 2 - 右侧按键
	– 4 - 中间按键
	按键可通过逐位 ODER 运算连接起来。
	
	示例：
	`<MOUSE_EVENT>`
	`  <print text="button %d type %d x %d y %d ">$mouse_event.button, $mouse_event.type, $mouse_event.x, $mouse_event.y</print>`
	`</MOUSE_EVENT>`

标签名称	含义

MESSAGE	**对话框信息**
	如果在脚本中执行指令 Send_message，解析器会执行标签 MESSAGE。值 p1 和 p2 会在变量 $message_par1 和 $message_par2 中提供（参见标签 “SEND_MESSAGE”）。
	
	语法：
	`<MESSAGE>`
	`</MESSAGE>`
	
	示例：
	`<LET name="user_selection" />`
	
	`<SOFTKEY POSITION="3">`
	`<CAPTION>Set%nParameter</CAPTION>`
	`<SEND_MESSAGE>1, 10</SEND_MESSAGE>`
	`</SOFTKEY>`
	…
	…
	
	`<FORM>`
	…
	…
	`<MESSAGE>`
	`<SWITCH>`
	`<CONDITION>$message_par1</CONDITION>`
	`<CASE value="1">`
	`<OP> user_selection = $message_par2 </OP>`
	…
	…
	`</CASE>`
	`</SWITCH>`
	`</MESSAGE>`
	…
	…
	`</FORM>`

标签名称	含义

SEND_MESSAGE	该标签会向有效的 FORM 标签发送含两个参数的信息，该信息在标签“Message”中进行处理（另见 MESSAGE）。
	
	语法：
	`<SEND_MESSAGE>p1, p2</SEND_MESSAGE>`
	
	示例：
	`<LET name="user_selection" />`
	
	`<SOFTKEY POSITION="3">`
	`  <CAPTION>Set%nParameter</CAPTION>`
	`  <SEND_MESSAGE>1, 10</SEND_MESSAGE>`
	`</SOFTKEY>`
	…
	…
	`<FORM>`
	…
	…
	`<MESSAGE>`
	`<SWITCH>`
	`<CONDITION>$message_par1</CONDITION>`
	`<CASE value="1">`
	`<OP> user_selection = $message_par2 </OP>`
	…
	`</CASE>`
	`</SWITCH>`
	`</MESSAGE>`
	…
	…
	`</FORM>`

标签名称	含义
RESIZE	**对话框信息** 该标签可将对 RESIZE 事件的编辑链接到脚本中。结果由动态分辨率切换生成。 

标签名称	含义

RESIZE 续	**示例：**
	`<let name="screen_size" type="StructSize" />`
	`<form name="menu_userscale_form" autoscale_content="off">`
	` <init>`
	` <caption>Use auto scaling</caption>`
	
	` <function name="hmi.get_hmi_resolution">screen_size</function>`
	` <print text="screen size: w %d h: %d">screen_size.width, screen_size.height</print>`
	
	` <data_access type="true" />`
	` <control name="s_c" xpos="8" ypos="140" fieldtype="readonly" width="500" refvar="string_var" hotlink="true"/>`
	` <if>`
	` <condition>screen_size.width == 800</condition>`
	` <then>`
	` <control name="s_1" xpos="508" ypos="140" fieldtype="readonly" width="100" refvar="string_var1" hotlink="true"/>`
	` </then>`
	` </if>`
	` <control name="s_w" xpos="200" ypos="350" fieldtype="readonly" refvar="screen_size.width" hotlink="true"/>`
	` <control name="s_h" xpos="200" ypos="370" fieldtype="readonly" refvar="screen_size.height" hotlink="true"/>`
	` </init>`
	` <paint>`
	` <text xpos ="68" ypos="310">Text</text>`
	` <text xpos ="8" ypos="350">screen width</text>`
	` <text xpos ="8" ypos="370">screen height</text>`
	` </paint>`
	
	` <resize>`
	` <function name="hmi.get_hmi_resolution">screen_size</function>`
	` <print text="resize_event screen size: w %d h: %d">screen_size.width, screen_size.height</print>`
	` <switch>`
	` <condition>screen_size.width</condition>`
	` <case value="640">`
	` <function name="control.delete">_T"s_1"</function>`
	` </case>`
	` <case value="800">`
	` <control name="s_1" xpos="508" ypos="140" fieldtype="readonly" width="100" refvar="string_var1" hotlink="true"/>`
	` </case>`
	` </switch>`
	` </resize>`
	`</form>`

标签名称	含义

CHANNEL_CHANGED	**对话框信息**
	标签使得可以对通道转换事件进行处理。在用户按下通道接通按钮时会启动进程。
	
	语法：
	`<channel_changed>`
	`</channel_changed>`
	
	示例：
	通道更换时，应关闭并再次打开 FORM 来显示已选择的通道的数据。
	`<channel_changed>`
	`  <open_form name = "form1" reopen="true"/>`
	`</channel_changed>`
	
	或
	
	当通道更换时，会激活另一个菜单。
	`<channel_changed>`
	`  <navigation>menu2</navigation>`
	`</channel_changed>`

LANGUAGE_CHANGED	**对话框信息**
	标签使得可以对语言转换要求进行处理。用户在打开对话框时更换语言会启动进程。
	
	语法：
	`<language_changed>`
	`</language_changed>`
	
	示例：
	语言更换时，应关闭并再次打开 FORM 来实现对生效语言的文本元素的控制。
	`<language_changed>`
	`  <open_form name = "form1" reopen="true"/>`
	`</language_changed>`

标签名称	含义

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

PAINT	**对话框信息** 该标签在显示对话框时执行。在此要给定需要在对话框中显示的所有文本与图像。 当系统确定对话框的一些地方需要重新显示时，会继续执行该标签。这可以通过，例如，关闭上一级窗口来实现。
TIMER	**对话框信息** 循环处理该标签。 每个 FORM 都分配了一个 TIMER，大约每 100 ms 就会触发一次 TIMER 标签的执行。

标签名称	含义

CAPTION	该标签包含有对话框的标题。
	在 INIT （初始化）标签内使用该标签。
	标题行可划分为多列。
	为了划分标题行，在输出文本前应使用属性“define_section”对标签“caption”进行编程。
	属性“define_section”确定列数。
	
	通过“property”属性定义每一列的长度、起始位置和文本方向。位置和长度说明是基于表格宽度的百分数值。
	属性值“value”表示列编号（从零开始）
	
	`<caption define_sections="3">`
	`<property value="0" length="20" position="2" alignment="left" />`
	`<property value="1" length="20" position="79" alignment="right" />`
	`</caption>`
	
	然后可以借助列索引指定属性 index 来指定列文本。
	`<caption index=“0”>Spalte1</caption>`
	`<caption index=“1”>Spalte2</caption>`
	
	语法：
	`<CAPTION>Titel</CAPTION>`
	
	示例：
	`<CAPTION>my first dialogue</CAPTION>`

CLOSE	**对话框信息** 在关闭对话框之前处理该标签。

标签名称	含义

CLOSE_FORM	该标签会关闭激活的对话框。
	只有通过 MMC 指令打开对话框且为操作者提供了关闭对话框的软键功能时，该指令才是必须的。一般而言，会自动管理对话框，而无需另外加以关闭。
	
	语法：
	`<CLOSE_FORM/>`
	
	示例：
	`<softkey_ok>`
	`<caption>OK</caption>`
	`<CLOSE_FORM />`
	`<navigation>main_menu</navigation>`
	`</softkey_ok>`

标签名称	含义
CONTROL	该标签用来创建控件。 **语法：** `<CONTROL name = "<control name>" xpos = "<X 坐标>" ypos = "<Y 坐标>" refvar = "<NC 变量>" hotlink = "true" format = "<格式>" />` **提示：** 必须将用于设置属性值的变量声明为全局变量。 **属性：** • **name** 字段名称 该名称会同时显示一个局部变量，并且在 FORM 中不允许重复使用。 • **xpos** 左上角的 X 坐标 • **ypos** 左上角的 Y 坐标 • **fieldtype** 字段类型 如未给定类型，则字段被自动设置为 Edit 型字段。 – **edit** 数据可以修改。 – **readonly** 数据无法修改。 **combobox** 字段显示相应的名称来代替数值。 如果选择了字段类型“**combobox**”，还要为字段设置需要显示的表达式。为此要使用标签 <ITEM> 。 组合框（Combo Box）将当前选定文本的下标保存在控件所属的变量中（参见属性 **refvar**）。 在创建了控件后，其他元素可通过函数 **addItem** 或 **insertItem** 来添加。 – **progressbar** 在值域 0 到 100 内显示进度条。 值域与待显示数据的最小值和最大值相匹配。

标签名称	含义

CONTROL 续	• **fieldtype edit** 和 **readonly**
	数组类型为 **edit** 和 **readonly** 时，可通过属性 **multiline** 进行文本的多行显示。换行可通过字数限制或使用换行符 <LF> 来实现。
	示例：
	`<control name="mulitline_edit" xpos="280" ypos="60" width="200" height="100" type="string">`
	`  <property multiline="true" />`
	`</control>`
	`<control name="c2" xpos="280" ypos="260" width="200" height="100" type="string" fieldtype="readonly">`
	`  <property multiline="true" />`
	`</control>`

标签名称	含义

CONTROL 续	• **fieldtype**
	– **graphicbox**
	该字段类型会生成一个 2 维图形控件。通过该标签 <ITEM> 会向控件中添加一个图形元素。参数 **width** 和 **height** 用来指定图形框的宽度和高度。
	在创建了控件后，其他元素可通过函数 **addItem** 或 **insertItem** 来添加。参数 **itemdata** 在该控件时不进行处理。
	使用控件可分配一个参考变量，在 100 ms 栅格内记录值。最多记录 10000 个值。属性为 **max** 时可以无穷记录数值，达到最大记录时间时，删除最早记录的值。记录时间单位为毫秒。
	在非连续的时间内保存的值以相互连接的直线的形式表示控件。属性 **channel** 使得能记录最多三个其他的参考变量。索引从一开始。索引零表示设置控件标签中分配的变量的属性。
	示例：
	`<control name= "c_gbox1" xpos = "250" ypos="24" width="240" height="356" fieldtype="graphicbox" refvar="val" hotlink="true" COLOR_BK="#ffffff">`
	`  <property max="60000" /> <!-- steps of ms -->`
	`  <property min="0" />`
	`  <property channel="0" color="#00ffff" penwidth="3"/>`
	`  <property ORDINATE="Y sinus" />`
	`  <property ABSCISSA="time *100 ms" />`
	`  <property channel="1" refvar="val2" color="#000000" />`
	`</control>`
	
	`<request name="nck/Channel/Parameter/R[2]" />`
	`<control name= "c_gbox" xpos = "0" ypos="5"  width="260" height="200" fieldtype="graphicbox" refvar="nck/Channel/Parameter/R[1]" hotlink="true" COLOR_BK="#ffffff">`
	`  <property max="400" />`
	`  <!-- steps of  ms  -->`
	`  <property min="0" />`
	`  <property channel="0" color="#00ff00" penwidth="1"/>`
	`  <property ORDINATE="Power" />`
	`  <property ABSCISSA="Time *100 ms" />`
	`  <property channel="1" refvar="nck/Channel/Parameter/R[2]" color="#FF0000" penwidth="1"/>`
	`  <property FIX_TIME_AREA="on" />`
	`</control>`

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

标签名称	含义
CONTROL 续	以下字段类型在“自定义按钮 (页 240)”一章中说明。 • **fieldtype** – **pushbutton** 该字段类型是“按钮”或者“开关”。 – **radiobutton** 该字段类型用于单选或多选。 – **checkbox** 该字段类型用于多选。 – **groupbox** 该字段类型将一组控件框在一起，并设置一个标题。 – **switch** 该字段类型是一种图形元件，借助一个图标指示两个状态中的一个。 – **scrollarea** 该字段类型用于在指定区域内显示控件。

标签名称	含义
CONTROL 续	• **fieldtype** – **listbox** 该字段类型会生成一个空的列表框控件。 通过标签 <ITEM> 会向列表框中添加一个列表框元素。 通过属性 ITEM **value** 为该元素赋唯一的一个值。 该值比如可用于明确标记该元素。 参数 **width** 和 **height** 用来指定列表框的宽度和高度。 在创建了控件后，其他的列表框元素可通过函数 **addItem**、**insertItem** 或 **loadItem** 来添加。 在创建了控件后，其他的列表框元素可通过函数 **addItem**、或 **insertItem** 来添加。参数 **itemdata** 在该控件时不进行处理。 – **itemlist** 该字段类型会生成一个静态控件，取代数值而显示相应的名称。 标签 <ITEM> 可为字段指定一个名称。 • **item_data** 可以使用用户专用的整数值为该属性赋值。该值被提供给 FOCUS_IN 信息，用来识别焦点字段。 • **refvar** 可以与字段互连的参考变量的名称（可选）。 • **hotlink** = "TRUE" 如果参考变量的值发生了变化，该字段会自动进行更新（可选）。 • **format** 该属性定义指定变量的显示格式。 格式定义参见"print-Tag "（可选）。 属性 **format** 使得整型值可以显示在其他进制中，也就是作为数组类型**编辑**和**只读**的布尔值。不支持字符数量和排列的格式化。 可使用下列显示： • **%o** - 八进制的 • **%x** - 十六进制的 • **%n** - 二进制的 • **%b** - 布尔

CONTROL 续	**列的创建示例，属性 property：**
	`<control name="lb1" xpos = "10" ypos = "94" width="200" height="250" fieldtype="listbox" refvar="tmp" hotlink="true">`
	`  <property columns="4" />`
	`  <property column="0" type="width">190</property>`
	`  <property column="1" type="width">100</property>`
	`  <property column="2" type="width">190</property>`
	`  <property column="3" type="width">16</property>`
	`</control>`

标签名称	含义

CONTROL 续	**属性 format 示例：**
	`<let name="val_test">255</let>`
	`<form name="main_form">`
	`<init>`
	`<caption>bool hex octal bin display</caption>`
	`<control name="test_oVal" xpos="250" ypos="60" refvar="val_test" hotlink = "true" />`
	`<control name="test_bool" xpos="250" ypos="90" refvar="val_test" hotlink = "true" format="%b"/>`
	`<control name="test_hex" xpos="250" ypos="120" refvar="val_test" hotlink = "true" format="%x"/>`
	`<control name="test" xpos="250" ypos="150" refvar="val_test" hotlink = "true" format="%o"/>`
	`<control name="test_b" xpos="200" ypos="180" width="300" refvar="val_test" hotlink = "true" format="%n"/>`
	`</init>`
	`<paint>`
	`<text xpos="16" ypos="60">integer value</text>`
	`<text xpos="16" ypos="90">boolean display</text>`
	`<text xpos="16" ypos="120">hexadecimal display</text>`
	`<text xpos="16" ypos="150">octal display</text>`
	`<text xpos="16" ypos="180">binary display</text>`
	`</paint>`
	`</form>`

标签名称	含义
CONTROL 续	**属性：** • **color_bk** 该属性设置控件的背景颜色。 • **color_fg** 该属性设置控件的前景颜色。 **更多信息**参见章节“颜色代码 (页 81)” • **display_format** 该属性定义了指定变量的处理格式。在存取 PLC 浮点型变量时必须使用该属性，因为要通过双字读取来进行存取。 允许使用下列数据格式： – FLOAT – INT – DOUBLE – STRING 向列表框、图形框或组合框分配表达内容（例如要显示的文本或图形）： **语法：** <ITEM>表达</ITEM> <ITEM value ="<值>">表达</ITEM>

标签名称	含义

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

标签名称	含义

CONTROL 续	**示例 itemlist:**
	`<CONTROL name = "itemlist1" xpos = "10" ypos = "10" fieldtype = "itemlist">`
	`<ITEM value = "10">text1</ITEM>`
	`<ITEM value = "20">text2</ITEM>`
	`<ITEM value = "12">text3</ITEM>`
	`<ITEM value = "1">text4</ITEM>`
	`</CONTROL>`

CONTROL 续	控件 Imagebox 控制位图格式或 GIF 格式的图形。如果图形大于所显示的区域，则出现滚动条控件。 系统提供函数 CONTROL.IMAGEBOXSET，用于控制可见区域。 **更多信息**参见章节“预定义功能 (页 172)” **语法：** `<function name="control.imageboxget"> ... </function>`
CONTROL 续	**属性：** • **disable** 该属性禁止/允许在 EDIT 控件中输入。 • **tooltip** 当光标移动到控件上时，显示提示文本。 • **factor** 换算系数 • **font** 指定字体 • **fontpixelsize** 字符高度 - 该值按照像素数指定并以分辨率 640x480 像素为基准。 所显示的字符高度通过实际分辨率与基准分辨率的比值确定。 **示例：** `<control name="c2" xpos="180" ypos="120" fontname="arial" fontpixelsize="26" height="100" type="string" />`

标签名称	含义
CONTROL 续	**属性：** • **fontname** 该属性用于确定要使用的字体。已安装的字体可通过函数 HMI.GET_FONT_FAMILIES 来查看。 **值：** 字体名称 系统中安装了以下西门子字体： • Siemens AD CH Simplified • Siemens AD CH Traditional • Siemens AD JP • Siemens AD KS • Siemens AD Mono • Siemens AD Mono CH Simplified • Siemens AD Mono CH Traditional • Siemens AD Mono JP • Siemens AD Mono KS • Siemens AD Mono TH • Siemens AD Mono VN • Siemens AD Sans • Siemens AD TH • Siemens AD VN • Siemens Logo • Siemens Sans Cond Global • Siemens Sans Cond Mono • Siemens Sans Global 基于微软 Windows 的系统还可安装其他字体。 **示例：** `<control name="c2" xpos="180" ypos="120" fontname="arial" fontpixelsize="26" height="100" type="string" />`

标签名称	含义
CONTROL 续	**创建后修改控件** 可使用控件标签在创建后修改已有控件的控件属性。必须同时给定待修改控件的名称及新属性。该操作可在 "Form" 标签内部进行。例如可修改以下属性： • name • xpos • ypos • width • height • color_bk • color_fg • access level • fieldtype • itemdata • min • max • default • disable • tooltip • font • factor 无法修改参考变量。如果要通过软键“事件”触发来修改属性，则须使用标签 "send message"，以上下文格式传送该要求。使用标签 "message" 采集信息。

标签名称	含义

CONTROL 续	**运行命令中的控件更改**
	在运行时更改控件属性的另一种方式是在运行命令中进行更改。为此需要指明控件的名称以及要分配新值的属性。属性与控件名称通过符号“点”隔开。
	语法：
	<Name>.<Eigenschaft>
	示例：
	`...`
	`...`
	`<let name="value" />`
	`<let name="w" />`
	`<let name="h" />`
	`...`
	`...`
	`<control name="c_move" xpos="$xpos" ypos="124" />`
	`...`
	`...`
	`<op>`
	`c_move.xpos = 300;`
	`value = c_move.xpos;`
	`h = c_move.height;`
	`w = c_move.width;`
	`</op>`

标签名称	含义

HELP_CONTEXT	该标签用来确定所要调用的帮助主题。在 INIT 初始化数据块中进行编程。
	系统 808/802D sl
	在属性中给定的名称会加上前缀 XmlUserDlg_ 并转发到帮助系统中。这样可以从在线帮助主题中提取出帮助文件的关联结构。
	激活帮助系统的过程：
	1. 按下“信息”键。
	2. 对话画面会显示 "my_dlg_help"。
	3. 解析器会将表达式转换为 "XmlUserDlg_my_dlg_help" 。
	4. 帮助系统激活。
	5. 传递查找关键字 "XmlUserDlg_my_dlg_help"。
	系统 840D sl/828D
	有关帮助手册的结构和生成方式的**更多信息**，请参见
	SINUMERIK Operate 调试手册
	
	属性：
	• **name**
	在属性中给定的名称会转发到帮助系统中。需要给出帮助手册中的标签 **entry** 中使用的文件名。
	• **anchor**
	使用此属性可给定关键字。
	
	语法：
	`<HELP_CONTEXT name="<file name>" anchor="key word"/>`
	
	示例：
	`<form name = "form1">`
	` <init>`
	`  <help_context name="chapter_1.html" anchor="Keyword_1" />`
	`...`
	`...`
	`  <control name = "cpwd" xpos = "322" ypos = "64" >`
	`   <property password="true" />`
	`   <property help_context="chapter_1.html" anchor="Keyword_2" />`
	`  </control>`
	`</form>`

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

标签名称	含义

DEBUG	标签用于保存可在脚本中被编程的踪迹输出。为描述要被保存的文本，请使用属性 **text**。此属性和其值符合 **text** 属性和打印指令的值。
	一般来说不会运行调试标签，因为这会使系统运行变慢。若要激活调试输出，需要将标签 **DialogGui** 或 **AGM** 中的属性 **debug_msg** 的值编程为 **on**。
	调试输出和 Parser 报警信息一同存储在下列文件中：
	card/oem/sinumerik/hmi/dvm/log/dvm.log
	
	语法：
	`<debug text="<文本参见打印指令>">变量参见打印指令</debug>`
	
	Easy XML 示例：
	`<DialogGui debug_msg="on">`
	`...`
	`...`
	`<control name="test_bool" xpos="250" ypos="90"`
	`refvar="val_test" hotlink = "true" format="%b"/>`
	`<debug text="main_form control test_bool created`
	`value:%d">test_bool</debug>`
	
	文件 dvm.log 中的条目：
	`date: 10/22/ 2018 time: 09:38:44:351`
	`Debug xmldial.xml 17: main_form control test_bool`
	`created value:255`
	
	Easy Extend 示例 ：
	`<AGM debug_msg="on">`
	`...`
	`...`
	`<control name="test_bool" xpos="250" ypos="90"`
	`refvar="val_test" hotlink = "true" format="%b"/>`
	`<debug text="main_form control test_bool created`
	`value:%d">test_bool</debug>`
	
	文件 dvm.log 中的条目：
	`date: 10/22/ 2018 time: 09:38:44:351`
	`Debug xmldial.xml 17: main_form control test_bool`
	`created value:255`

标签名称	含义

EDIT_CHANGED	**对话框信息**
	当 EDIT 控件的内容发生了变化时，调用该标签。
	为了识别控件，系统将控件的名称复制到变量 **$focus_name** 中并将 **item_data** 的属性值复制到变量 **$focus_item_data** 中。
	示例：
	`<EDIT_CHANGED>`
	`<print text="index changed filed:%s, %d"> $focus_name, $focus_item_data </print>`
	…
	…
	`</EDIT_CHANGED>`

GESTURE_EVENT	该标签用于执行多点触控操作中的手势。 该标签在“多点触控操作 (页 232)”一章中说明。

INDEX_CHANGED	**对话框信息**
	当操作员修改了复合框的选择时，调用该标签。
	为了识别控件，系统将控件的名称复制到变量
	$focus_name 中，将属性 **item_data** 的值复制到变量**$focus_item_data** 中。
	提示：
	分配给控件的参考变量此时还未与控件变量进行匹配，所含的仍是上一次复合框选择的下标。
	示例：
	`<INDEX_CHANGED>`
	`<print text="index changed filed:%s, %d"> $focus_name, $focus_item_data </print>`
	…
	…
	`</INDEX_CHANGED>`

标签名称	含义

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

NAVIGATION	该标签用来确定所要调用的菜单。可以在软键程序块、菜单程序块和 Form 中设置。如果给标签分配了一个变量名称作为值，则解析器会激活变量中的菜单。
	菜单程序块中，导航至说明位置。后续说明将不再执行。
	
	提示：
	在需要通过变量的内容定义导航目标的情况下，必须将此变量声明为全局变量。
	
	语法：
	`<NAVIGATION>menu name</NAVIGATION>`
	
	示例：
	`<menu name = "main">`
	`<softkey POSITION="1">`
	`<caption>sec. form</caption>`
	`<navigation>sec_menu</navigation>`
	`</softkey>`
	`</menu>`
	
	`<menu name = "sec_menu">`
	`<open_form name = "sec_form" />`
	`<softkey_back>`
	`<navigation>main</navigation>`
	`</softkey_back>`
	`</menu>`

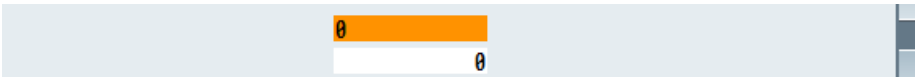
标签名称	含义

OPEN_FORM	该标签可以打开名称下指令的对话框窗口。
	若给定的对话框已经激活，则不会运行此标签。
	对属性 **reopen** 的说明可以强制重复打开对话框。当系统状态改变且预设了窗口内容的重组时，这可能是必要的。
	
	属性：
	• **name**
	对话框窗口的名称
	• **reopen**
	若属性值为 **true**，则在重复调用标签 **open_form** 时会检验这里是否为一个已激活的对话框。如果是，则 Parser 会关闭并再次打开已激活的 Form。
	
	语法：
	`<OPEN_FORM name = "<form name>" />`
	
	示例：
	`<menu name = "main">`
	`<open_form name = "main_form" />`
	`<softkey POSITION="1">`
	`<caption>main form</caption>`
	`<navigation>main</navigation>`
	`</softkey>`
	`</menu>`
	
	`<form name="main_form">`
	`<init>`
	`</init>`
	
	`<paint>`
	`</paint>`
	
	`</form>`

标签名称	含义

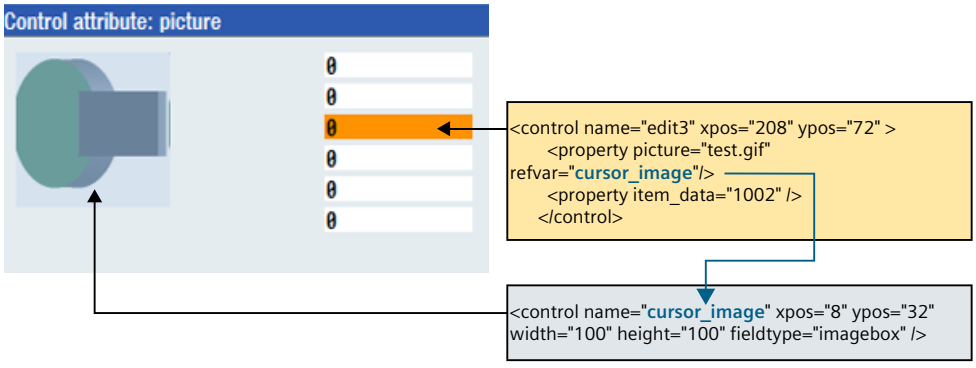
PROPERTY	使用该标签可以确定操作单元的附加特性。该标签嵌入在控件标签中。
	
	属性：
	• **max** = "<最大值>"
	• **min** = "<最小值>"
	• **default** = "<预设值>"
	• **factor** = "换算系数"
	• **color_bk** = "<背景颜色代码>"
	• **color_fg** = "<字体颜色代码>"
	• **password** = "<true>" - 所输入的字符显示为"*"
	• **multiline** = "<true>" - 在 EDIT 控件中允许多行输入
	• **disable** = ""<true/false>" - 在 EDIT 控件禁止/允许输入
	• **transparent** = "位图的透明色"
	更多信息参见章节 "颜色代码 (页 81)"
	• **tooltip** = 当光标移动到控件上时，显示提示文本。
	语法为： <property tooltip="提示文本" />
	• **abscissa** = "第一个坐标轴的名称"（仅允许用于图形框）
	• **ordinate** = "第二个坐标轴的名称"（仅允许用于图形框）
	• **fix_time_area** = "on" - 属性 **min** 和 **max** 用于确定信号的显示时段。所显示的信号从右向左移动。
	示例：
	`<CONTROL name = "progress1" xpos = "10" ypos = "10" width = "100" fieldtype = "progressbar" hotlink = "true" refvar = "nck/Channel/GeometricAxis/actProgPos[1]">`
	`<PROPERTY min = "0" />`
	`<PROPERTY max = "1000" />`
	`</CONTROL>`
	`<CONTROL name = "edit1" xpos = "10" ypos = "10">`
	`<PROPERTY min = "20" />`
	`<PROPERTY max = "40" />`
	`<PROPERTY default = "25" />`
	`</CONTROL>`

标签名称	含义

PROPERTY 续	**属性：**
	alignment - 使文本左对齐或右对齐。默认设置为文本左对齐。
	
	值：
	• **left**
	• **right**
	
	语法：
	`alignment="<right/left>"`
	
	示例：
	`<control name="edit2" xpos="208" ypos="52" >`
	`<property alignment="right"/>`
	`</control>`
	
	 

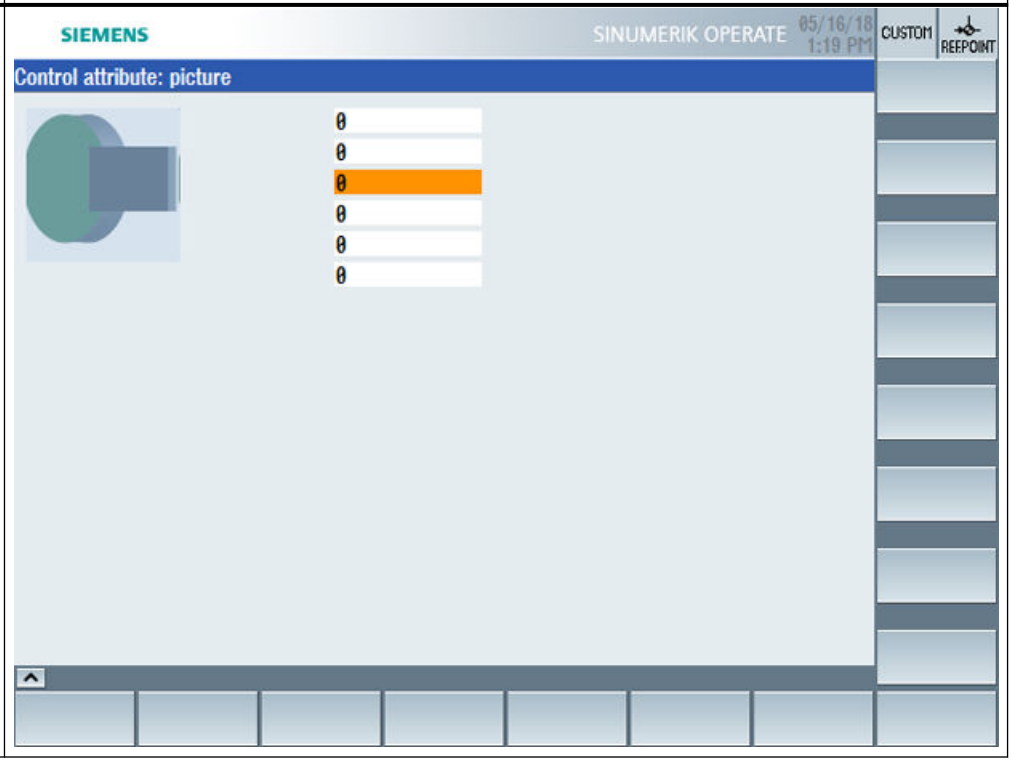
标签名称	含义

PROPERTY 续	**属性：**
	parent - 确定控件的所属关系。默认设置为控件属于窗体。如要将控件分配给滚动视图，则要将其指定为父窗体。
	
	值：
	父窗体名称
	
	示例：
	`...`
	`<control name="area" xpos="8" ypos="72" width="554" height="320" fieldtype="scrollarea" >`
	`<control name="test" xpos="20" ypos="40" width="20" fieldtype="readonly" />`
	`</control >`
	`<control name="test1" xpos="20" ypos="80" width="20" fieldtype="readonly" parent="area"/>`
	
	test
	
	test1
	
	`<init>`
	`<control name="area" xpos="8" ypos="72" width="554" height="120" fieldtype="scrollarea">`
	`<control name="test" xpos="20" ypos="40" width="80" fieldtype="readonly" type="string"/>`
	`</control >`
	
	`<control name="test1" xpos="20" ypos="80" width="80" fieldtype="readonly" type="string" parent="area"/>`
	`<op>test = _T"test"</op>`
	`<op>test1 = _T"test1"</op>`
	`<update_controls type="true" />`
	`...`

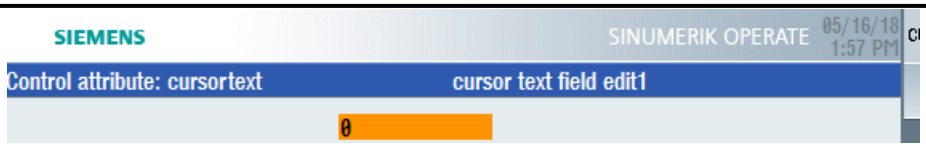
标签名称	含义
PROPERTY 续	**属性：** **picture** - 确定要显示的图片 如果此区域处于输入状态，则显示给定的图片或者将图片名保存在一个变量中。此属性应与 **refvar** 属性合用，以便确定数据宿。 为显示图片或动画，需要给定 **imagebox** 类型的控件名称。此控件可管理图片。 给定一个局部变量名称时，此变量的数据类型必须是 **String**。 图片将一直显示到一个新名称分配到参考变量中为止。 **语法：** `<property picture="<Name der Grafik>" refvar="<Datensenke>"/>`
PROPERTY 续	**数组的关联性示例：** 

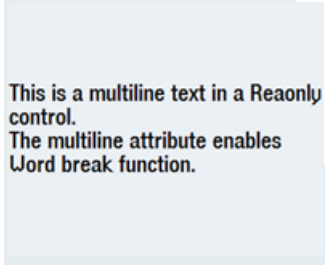
标签名称	含义

PROPERTY 续	**示例：**
	`<let name="cursor_icon" type="string" />`
	` <form name="main_form1">`
	`  <init>`
	`  <caption>Control attribute: picture</caption>`
	`  <control name="cursor_image" xpos="8" ypos="32" width="100" height="100" fieldtype="imagebox" />`
	`  <control name="edit1" xpos="208" ypos="32" >`
	`  <property picture="red_led_off.bmp" refvar="cursor_image"/>`
	`  </control>`
	`  <control name="edit2" xpos="208" ypos="52" >`
	`  <property picture="red_led_on.bmp" refvar="cursor_image"/>`
	`  </control>`
	`  <control name="edit3" xpos="208" ypos="72" >`
	`  <property picture="test.gif" refvar="cursor_image"/>`
	`  </control>`
	`  <control name="edit4" xpos="208" ypos="92" >`
	`  <property picture="red_led_off.bmp" refvar="cursor_image"/>`
	`  </control>`
	`  <control name="edit5" xpos="208" ypos="112" >`
	`  <property picture="red_led_off.bmp" refvar="cursor_icon"/>`
	`  </control>`
	`  <control name="edit6" xpos="208" ypos="132" >`
	`  <property picture="red_led_on.bmp" refvar="cursor_icon"/>`
	`  </control>`
	`  </init>`
	`  <timer><print text="%s">cursor_icon</print></timer>`
	` </form>`

标签名称	含义
	 SIEMENS SINUMERIK OPERATE 05/16/18 1:19 PM CUSTOM REFPOINT Control attribute: picture 0 0 0 0 0 0

标签名称	含义
PROPERTY 续	**属性：** **cursortext** - 此属性确定要显示的光标文本。 当此区域为输入状态时，文本会在标题行输出。 对于此属性，还可以给定两个属性来确定文本和光标文本区域的对齐方式。默认文本左侧对齐。光标文本区域默认占据标题行宽度的 50%。 alignment="<right/left>" - 文本对齐 右对齐/左对齐 length="<宽度>" - 光标文本应当在标题行中占有的百分比 **语法：** <property cursortext="<text>" /> 或 <property cursortext="<text>" alignment="<right/left>" /> 或 <property cursortext="text2" alignment="r="<text>" alignment="<right/left>" length="<比率值>"/> **示例：** <form name="main_form2"> <init> <caption>Control attribute: cursortext</caption> <control name="edit1" xpos="208" ypos="32" > <property cursortext="cursor text field edit1" /> </control> <control name="edit2" xpos="208" ypos="52" > <property cursortext="cursor text field edit2" alignment="right"/> </control> <control name="edit3" xpos="208" ypos="72" > <property cursortext="cursor text field edit3" alignment="right" length="40"/> </control> <control name="edit4" xpos="208" ypos="92" > <property cursortext="cursor text field edit4" /> </control> </init> </form>

标签名称	含义
	

PROPERTY 续	**属性：**
	multiline - 实现 EDIT 或 Readonly 控件中文本的多行输出
	
	语法：
	`<property multiline="true" />`
	
	示例：
	`...`
	`...`
	`<op>`
	`textlist[2] = _T"This is a multiline text in a Reaonly control.\`
	`\nThe multiline attribute enables Word break function.";`
	`</op>`
	`<control name="lstring" xpos="8" ypos="120" width="200" height="160"`
	`fieldtype="readonly" refvar="textlist[2]" >`
	`<property multiline="true" />`
	`</control>`
	
	 

标签名称	含义
PROPERTY 续	**属性：** • **help_context** 此属性给一个控件分配一个帮助主题。需要指定帮助手册中标签 **entry** 中使用的文件名。 • **anchor** 使用此属性可给定关键字。此属性仅在与属性 **help_context** 结合时才生效。 **语法：** `<property help_context="<file name>" anchor="<key word>"/>` **示例：** `<control name = "cpwd" xpos = "322" ypos = "64" > <property password="true" /> <property help_context="chapter_1.html" anchor="Keyword_2" /> </control>`
SOFTKEY	该标签定义了软键的特性与反应。 **属性：** • **position** 软键的编号。1-8 水平软键， 9-16 垂直软键 以下属性生效于： • **type** 定义软键的属性。 user_controled - 软键的显示由脚本确定 toggle_softkey - 软键在交替按下时显示或不按时显示 • **refvar** 只在连接时使用类型 toggle_softkey 。 参考变量，将当前的软键属性复制到其中。 定义了类型为"String"的变量，含有属性“按下”、“不按”或“禁用”（参见标签 state）。 • **picture** 借助该属性，位图会按左对齐方式输出在软键上。必须给出完整的路径。 可显示的文本字符的数量受位图宽度的限制。

标签名称	含义
SOFTKEY 续	可以在软键程序块中确定下列其他操作： • **picturealignment** 通过该属性指定图片的校准。 图片默认位于软键的左侧。 可以设置下列值，用于校准： – **top** 上边沿 – **bottom** 下边沿 – **left** 左边沿 – **right** 右边沿 – **center** 居中 • **caption** 软键文本 • **state** 只在连接时使用类型 user_controlled 。 该标签向系统分配所需的软键显示。 **语法：** <state type="<state>" /> 可以指定以下字符串： – **notpressed** 软键在不按时显示。 – **pressed** 软键在按下时显示。 – **disabled** 软键禁用并显示为灰色。 • **navigation** • **update_controls** • **function**

标签名称	含义

SOFTKEY 续	**语法：**
	标准软键：
	`<state type="<softkey state>" />`
	`<softkey position = "<1>">`
	…
	…
	`</softkey>`
	或
	受脚本控制的软键：
	`<softkey position = "<1>" type="<user_defined>" >`
	`<state type="<softkey state>" />`
	…
	…
	`</softkey>`
	
	或
	切换软键：
	`<softkey position = "<1>" type="<toggle_softkey>" refvar="<variable name>" >`
	…
	…
	`</softkey>`
	
	示例：
	
	`<let name="define_sk_type" type="string">PRESSED</let>`
	`<let name="sk_type">1</let>`
	
	`<softkey POSITION="1" type="user_controled" >`
	`<caption>Toggle%nSK</caption>`
	`<if>`
	`<condition>sk_type == 0 </condition>`
	`<then>`
	`<op> sk_type = 1 </op>`
	`<op> define_sk_type = _T"PRESSED" </op>`
	`</then>`
	`<else>`
	`<op> define_sk_type = _T"NOTPRESSED" </op>`
	`<op> sk_type = 0 </op>`
	`</else>`
	`</if>`
	`<state type="$$$define_sk_type" />`
	…
	…
	
	`</softkey>`

标签名称	含义

SOFTKEY 续	**示例：**
	或
	
	`<let name="curr_softkey_state" type="string">PRESSED</let>`
	`</softkey>`
	
	`<softkey POSITION="3" type="toggle_softkey" refvar="curr_softkey_state">`
	`<caption>Toggle%nSK</caption>`
	…
	
	</softkey>
	 
	
	 

SOFTKEY_OK	该标签定义软键“确认”的反应。
	 
	
	可以在软键程序块中确定下列其他操作：
	• **navigation**
	• **update_controls**
	• **function**
	
	语法：
	`<SOFTKEY_OK>`
	…
	…
	`</SOFTKEY_OK>`

标签名称	含义

SOFTKEY_CANCEL	该标签定义软键“取消”的反应。
	 
	
	可以在软键程序块中确定下列其他操作：
	• **navigation**
	• **update_controls**
	• **function**
	语法：
	`<SOFTKEY_CANCEL>`
	…
	…
	`</SOFTKEY_CANCEL>`

SOFTKEY_BACK	该标签定义软键“返回”的反应。
	 
	
	可以在软键程序块中确定下列其他操作：
	• **navigation**
	• **update_controls**
	• **function**
	语法：
	`<SOFTKEY_BACK>`
	…
	…
	`</SOFTKEY_BACK>`

标签名称	含义

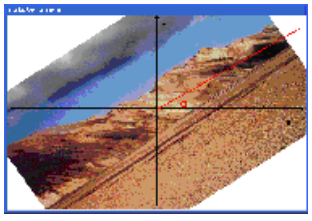
SOFTKEY_ACCEPT	该标签定义软键“接收”的反应。
	 
	
	可以在软键程序块中确定下列其他操作：
	• **navigation**
	• **update_controls**
	• **function**
	语法：
	`<SOFTKEY_ACCEPT>`
	…
	…
	`</SOFTKEY_ACCEPT>`

TEXT	该标签用来在指定的位置上显示文本。 如果使用报警编号，则对话框会显示为该编号所保存的文本。 **语法：** `<TEXT xpos = "<X 坐标>" ypos = "<Y 坐标>"> Text </TEXT>` **属性：** • **xpos** 左上角的 X 坐标 • **ypos** 左上角的 Y 坐标 • **color** 文本颜色 **更多信息**参见章节“颜色代码 (页 81)” **值：** 待显示的文本

标签名称	含义

IMG	该标签用来在指定的位置上显示图像。支持 BMP 和 PNG 图像格式。
	
	语法：
	`<IMG xpos = "<X 坐标>" ypos = "<Y 坐标>" name = "<名称>" />`
	`<IMG … xrot="X 轴角度"yrot="Y 轴角度" zrot="Z 轴角度"/>`
	
	属性：
	• **xpos**
	左上角的 X 坐标
	• **ypos**
	左上角的 Y 坐标
	• **name**
	完整的路径名称。路径名称只能用小写字母。
	• **transparent**
	位图的透明色
	更多信息参见章节“颜色代码 (页 81)”

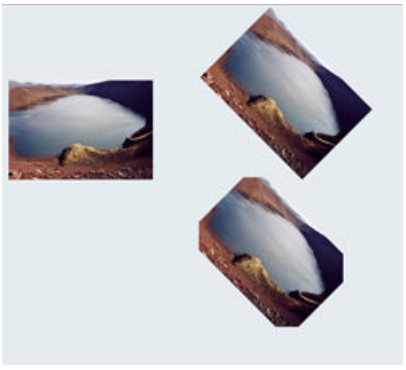
标签名称	含义

IMG 续	**示例：**
	图片绕 Z 轴旋转 34°。
	`<img xpos = "5" ypos = "23" name = "f:/appl/pic3.bmp" height="355" width="550" zrot="34" color_bk="#ffffff" ></img>`
	
	提示：
	驱动器名称根据系统而有所不同。
	 
	
	可选：
	希望修改图片的原始大小时，可以使用属性 **width** 和 **height** 来确定尺寸。
	• **width**
	宽度，单位像素
	• **height**
	高度，单位像素
	
	示例：
	`<IMG xpos = "20" ypos = "40" name = "f:/appl/test.bmp" />`
	`<IMG xpos = "5" ypos = "23" name = "f:/appl/test.bmp" height = "355" width = "550"/>`

标签名称	含义

IMG 续	**属性：**
	AspectRatioMode
	此属性在非比例性的缩放中用于控制图形长宽比。
	
	值：
	• **Ignore**
	位图的长宽比按照给定的高度和宽度进行调整。
	• **Keep** （标准）
	长宽比保留并确定图片缩放为最大限度地填充给定高度和宽度的矩形。
	• **KeepByExpanding**
	长宽比保留并确定图片缩放为超出给定的高度和宽度最少的矩形。
	
	示例：
	`<img xpos = "8" ypos = "30" name = "test.bmp" height="155" width="150" AspectRatioMode="KeepbyExpanding">`
	`  <property transparent="#00ff00" />`
	`</img>`
	`<img xpos = "8" ypos = "200" name = "test.bmp" height="155" width="150" AspectRatioMode="Ignore">`
	`  <property transparent="#00ff00" />`
	`</img>`
	`<img xpos = "200" ypos = "200" name = "test.bmp" height="155" width="150" AspectRatioMode="Keep">`
	`  <property transparent="#00ff00" />`
	`</img>`
	 
	• 上方 - 已设置 **KeepbyExpanding** 属性
	• 左下方 - 已设置 **Ignore** 属性
	• 右下方 - 已设置 **Keep** 属性

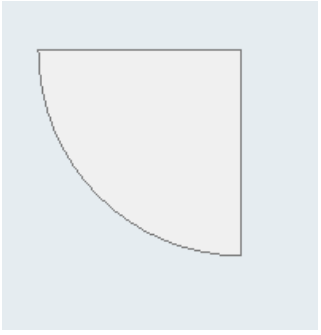
标签名称	含义

IMG 续	**属性：**
	ExpandingForRotation
	长宽比保留。图片缩放为与已旋转和缩放的图形的边框相对应的矩形。
	
	示例：
	`<img xpos = "8" ypos = "100" name = "pic1.bmp" height="155" width="150" >`
	`  <property transparent="#ffffff" />`
	`</img>`
	`<op>`
	`  zrot = 45.0;`
	`</op>`
	`<img xpos = "205" ypos = "23" name = "pic1.bmp" height="155" width="150" zrot="$zrot" AspectRatioMode="ExpandingForRotation">`
	`  <property transparent="#ffffff" />`
	`</img>`
	`<img xpos = "205" ypos = "200" name = "pic1.bmp" height="155" width="150" zrot="$zrot" >`
	`  <property transparent="#ffffff" />`
	`</img>`
	 
	• 左侧 - 图片缩放为 150x155 像素
	• 右上方 - 图片缩放为 150x155 像素且用 **ExpandingForRotation** 属性旋转 45 度
	• 右下方 - 图片缩放为 150x155 像素且旋转 45 度

标签名称	含义
ARC	此标签用于在窗体中绘制圆弧。 **属性：** 以像素为单位指定在窗体中的位置： • **xpos1** - 圆弧起点的 X 坐标 • **ypos1** - 圆弧起点的 Y 坐标 • **xpos2** - 圆弧终点的 X 坐标 • **ypos2** - 圆弧终点的 Y 坐标 以像素为单位指定在窗体中的位置： • **ccx** - 圆弧圆心的 X 坐标 • **ccy** - 圆弧圆心的 Y 坐标 • **radius** - 圆弧半径，像素值 **提示：** **圆心或半径** 指定这两个参数时使用圆弧圆心。
ARC 续	• **penwidth** 以像素为单位设置线条的粗细。 • **color** 线条颜色 • **color_bk** 圆弧的填充色 **更多信息**参见章节“颜色代码 (页 81)” • **style** 设置线条样式： – **"SolidLine"** - 实线（默认设置） – **"DashLine"** - 虚线 – **"DotLine"** - 点线 – **"DashDotLine"** - 点划线 – **"DashDotDotLine"** - 双点划线

标签名称	含义
ARC 续	• **type** – **cw** - 顺时针 – **ccw** - 逆时针 • **arrow** 在圆弧两端显示箭头： – **"P1"** - 箭头位于线条起点 – **"P2"** - 箭头位于线条起点 – **"P1P2"** - 箭头分别位于线条起点和终点 – **"P1_FILLED"** - 实心箭头位于线条起点 – **"P2_FILLED"** - 实心箭头位于线条起点 – **"P1P2_FILLED"** - 实心箭头分别位于线条起点和终点 • **arrow_size** 如果使用属性 arrow，则可以像素为单位指定箭头的长度。

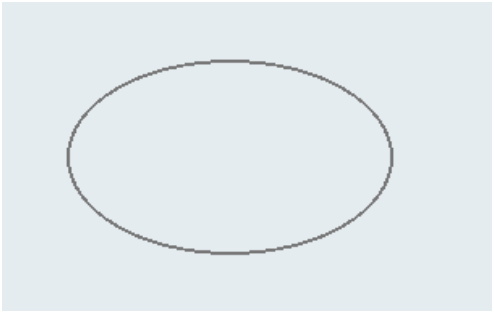
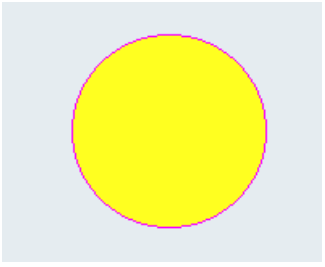
标签名称	含义

ARC 续	**示例：**
	 
	`<arc xpos1="20" ypos1="200" xpos2="180" ypos2="200" ccx="100" ccy="200" type="ccw" PENWIDTH="1" color="#777777" arrow="p1p2_filled"/>`
	
	或
	
	`<arc xpos1="20" ypos1="200" xpos2="180" ypos2="200" radius="80" type="ccw" PENWIDTH="1" color="#777777" arrow="p1p2_filled"/>`
	
	 
	`<arc xpos1="20" ypos1="200" xpos2="100" ypos2="280" radius="80" type="ccw" PENWIDTH="1" color="#777777" color_bk="#f0f0f0"/>`

标签名称	含义
BOX	该标签可以在指定的位置以指定的颜色绘制一个填满的矩形。 **语法：** `<BOX xpos = "<X 坐标>" ypos = "<Y 坐标>" width = "<X 长度>" height = "<Y 长度>" color = "<颜色代码>" />` **属性：** • **xpos** 左上角的 X 坐标 • **ypos** 左上角的 Y 坐标 • **width** X 方向上的宽度（以像素为单位） • **height** Y 方向上的高度（以像素为单位） • **color** 颜色代码 **更多信息**参见章节“颜色代码 (页 81)”

标签名称	含义
ELLIPSE	此标签用于在窗体中绘制椭圆。 **属性：** 以像素为单位指定在窗体中的位置： • **xpos** - 线条起点的 X 坐标 • **ypos** - 线条起点的 Y 坐标 • **width** - 外切矩形的宽度 (bounding box) • **height** - 外切矩形的高度 (bounding box) • **penwidth** 以像素为单位设置线条的粗细。 • **color** 线条颜色 • **color_bk** 椭圆的填充色 **更多信息**参见章节“颜色代码 (页 81)” • **style** 设置线条样式： – **"SolidLine"** - 实线（默认设置） – **"DashLine"** - 虚线 – **"DotLine"** - 点线 – **"DashDotLine"** - 点划线 – **"DashDotDotLine"** - 双点划线

标签名称	含义

ELLIPSE 续	**示例：**
	 
	`<ellipse xpos="302" ypos="160" width="100" height="60" PENWIDTH="2" color="#777777" />`
	 
	`<ellipse xpos="402" ypos="360" width="60" height="60" PENWIDTH="1" color="#f800ff"  color_bk="#ffff20"/>`

标签名称	含义

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

标签名称	含义

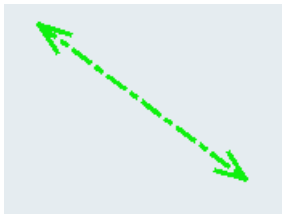
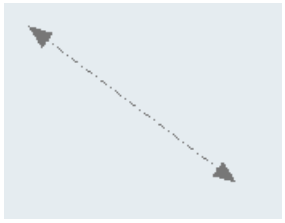
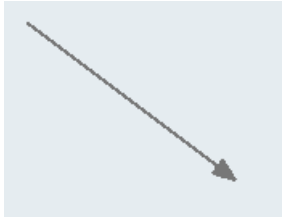
FUNCTION_BODY	**函数体**
	该标签包含有一个子函数的函数体。要在 DialogGui 标签中对函数体进行编程。
	
	属性：
	• **name**= “函数体的名称”
	• **parameter** = “参数列表”（可选）
	属性列出了需要的传输参数。参数相互之间以逗号分隔。
	调用函数体时，函数调用中给定参数的值会复制到所列出的传递参数中。
	• **return** = “true”（可选）
	如果属性设为 true，则创建局部变量 $return。将函数的返回值复制到其中，在退出该函数时该值会继续传输给主调函数。
	
	语法：
	
	无参数的函数体
	`<FUNCTION_BODY name = "<function name>">`
	…
	…
	…
	`</ FUNCTION_BODY>`
	
	带参数的函数体
	`<FUNCTION_BODY name = "<function_name>" parameter = "<p1, p2, p3>">`
	…
	`<LET name = "tmp></LET>`
	`<OP> tmp = p1 </OP>`
	…
	`</FUNCTION_BODY>`
	
	带返回值的函数体
	`<FUNCTION_BODY name = "<function_name>" parameter = "<p1, p2, p3>" return = "true">`
	…
	`<LET name = "tmp></LET>`
	`<OP> tmp = p1 </OP>`
	…
	`<OP> $return = tmp </OP>`
	`</FUNCTION_BODY>`

标签名称	含义
FUNCTION_BODY 续	**示例：** <function_body name = "test" parameter = "c1,c2,c3" return = "true"> <LET name = "tmp">0</LET> <OP> tmp = c1+c2+c3 </OP> <OP> $return = tmp </OP> </function_body> … … … <LET name = "my_var"> 4 </LET> <function name = "test" return = " my_var "> 2, 3,4</function> <print text = "result = %d"> my_var </print> … …
LINE	此标签用于在窗体中绘制直线。 **属性：** 以像素为单位指定在窗体中的位置： • **xpos1 -** 线条起点的 X 坐标 • **ypos1** - 线条起点的 Y 坐标 • **xpos2** - 线条终点的 X 坐标 • **ypos2** - 线条终点的 Y 坐标 • **penwidth** 以像素为单位设置线条的粗细。 • **color** 线条颜色 **更多信息**参见章节 “颜色代码 (页 81)” • **style** 设置线条样式： – **"SolidLine"** - 实线（默认设置） – **"DashLine"** - 虚线 – **"DotLine"** - 点线 – **"DashDotLine"** - 点划线 – **"DashDotDotLine"** - 双点划线

标签名称	含义

LINE 续	• **arrow**
	在直线两端显示箭头：
	– **"P1"** - 箭头位于线条起点
	– **"P2"** - 箭头位于线条起点
	– **"P1P2"** - 箭头分别位于线条起点和终点
	– **"P1_FILLED"** - 实心箭头位于线条起点
	– **"P2_FILLED"** - 实心箭头位于线条起点
	– **"P1P2_FILLED"** - 实心箭头分别位于线条起点和终点
	• **arrow_size**
	如果使用属性 arrow，则可以像素为单位指定箭头的长度。
	
	语法：
	`<line xpos1="<起点 X 坐标>" ypos1="<起点 Y 坐标>" xpos2="<终点 X 坐标>" ypos2="<终点 Y 坐标>"`
	`PENWIDTH="<线条宽度>" color="<线条颜色>" style="<线条样式>" arrow="<箭头样式>" arrow_size="<箭头大小>"/>`

标签名称	含义

LINE 续	**示例：**
	`<line xpos1="20" ypos1="40" xpos2="100" ypos2="100" PENWIDTH="3" color="#0ff00f" style="DashDotLine" arrow="p1p2" arrow_size="12"/>`
	 
	`<line xpos1="20" ypos1="40" xpos2="100" ypos2="100" PENWIDTH="1" color="#777777" style="DashDotLine" arrow="p1p2_filled" arrow_size="9"/>`
	 
	`<line xpos1="20" ypos1="40" xpos2="100" ypos2="100" PENWIDTH="2" color="#777777" arrow="p2_filled" arrow_size="9"/>`
	 

标签名称	含义
REQUEST	借助该标签可以在周期性读取服务（hotlink 热连接）中使用一个变量。这样可以缩短对未与控件相连的变量的存取时间。 如果在值变化时自动调用一个函数，则应将函数的名称作为其它属性加以定义。 该标签仅在 INIT 初始化指令中进行处理。 **属性：** • **name** 地址名称 • **function** 函数名称 **语法：** <REQUEST name = "<NC-Variable>" /> 或 <REQUEST name = "<NC-Variable>" function="<function name>"/> **示例：** <request name ="plc/mb10" /> 或 <function_body name="my_function" > <print text="value changed" /> </function_body> … … … <request name ="plc/mb10" function="my_function"/>

标签名称	含义

RECALL	如要通过 Recall 键执行导航，则可在菜单中使用该标签。如对该标签编程，则会显示 Recall 符号并允许通过解析器对 Recall 键进行处理。
	
	语法：
	`<recall>`
	…
	…
	`</recall>`

UPDATE_CONTROLS	该标签用于比较操作单元和参考变量。
	
	属性：
	• **type**
	该属性可以确定数据比较的方向。
	= TRUE – 从参考变量读取数据并复制到操作单元中。
	= FALSE – 从操作单元读取数据并复制到参考变量中。
	
	语法：
	`<UPDATE_CONTROLS type = "<方向>"/>`
	
	示例：
	`<SOFTKEY_OK>`
	`< UPDATE_CONTROLS type="false"/>`
	`</SOFTKEY_OK>`

3.8 创建用户菜单

3.8.1 创建加工循环窗口

循环支持功能可以通过格式对话框自动创建和反编译循环调用。

提供以下标签实施该功能：

- **NC_INSTRUCTION**
- **CREATE_CYCLE**

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

示例

```
<FORM name = "cycle100_form" type= "CYCLE">
…
…
</FORM>
```

标签 **NC_INSTRUCTION** 含有要执行的循环调用。所有循环参数都通过占位符进行了预留。

示例

```
<FORM name = "cycle100_form" type= "CYCLE">
<NC_INSTRUCTION refvar= "cyc_string" >Cycle100 ($p1, $p2, $p3)</
NC_INSTRUCTION>
…
…
…
</FORM
```

标签 **CREATE_CYCLE** 提供保存在占位符变量中的值并生成 NC 指令。

然后复制到指定的变量中。

标签名称	含义

NC_INSTRUCTION	使用该标签可定义要创建的 NC 指令。
	列出的所有循环参数都会自动作为 **FORM** 字符串变量被创建并可用于 **FORM**。
	
	前提条件：FORM 的属性 **type** 设置为 **CYCLE**。
	
	属性：
	• **refvar**
	如果向标签分配了一个参考变量，则所有参数都会预设为参考变量中保存的 NC 程序段中的值。
	句法：
	`<NC_INSTRUCTION> 带占位符的 NC 指令</ NC_INSTRUCTION>`
	
	示例：
	`<let name="cyc_string" type="string"> Cycle100(0, 1000, 5)</ let>`
	…
	…
	…
	`<FORM name = "cycle100_form" type= "CYCLE">`
	`<NC_INSTRUCTION refvar= "cyc_string" >Cycle100($p1, $p2, $p3)</ NC_INSTRUCTION>`
	…
	…
	…
	`</FORM>`

标签名称	含义

CREATE_CYCLE	该标签会生成一个 NC 程序段，其句法由标签 **NC_INSTRUCTION** 确定。
	在生成 NC 指令前，分析程序会调用标签 FORM 的 CYCLE_CREATE_EVENT。该标签可用于计算循环参数。
	句法：
	`<CREATE_CYCLE/>`
	选项：
	指定了参考变量时，指令会在该变量中复制创建的调用。
	`<create_cycle refvar="name" />`
	属性：
	• **refvar**
	如果向该标签分配了一个参考变量，则 NC 指令会复制到该变量中。
	示例：
	`<LET name="cyc_string" type="string"> Cycle100(0, 1000, 5)</LET>`
	`<SOFTKEY_OK>`
	`<caption>OK</caption>`
	`<CREATE_CYCLE />`
	`<close_form />`
	`<navigation>main_menu</navigation>`
	`</SOFTKEY_OK>`
	或
	`<SOFTKEY_OK>`
	`<caption>OK</caption>`
	`<CREATE_CYCLE refvar= "cyc_string" />`
	`<close_form />`
	`<navigation>main_menu</navigation>`
	`</SOFTKEY_OK>`

3.8.2 替代符号

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

如果标签要以字符串作为属性值或值，则应在变量名称前加上符号 $$$。

示例：

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

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

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

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

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

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

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

3.9 创建 struct_def.xml 文件

步骤

在一些函数中会使用文件 **struct_def.xml** 进行类型定义。如果缺少此 XML 文件，则可以使用下面的示例脚本创建一个文件副本并将其集成到函数项目中。

说明

以 UTF8 格式创建 XML 文件。

脚本

struct_def.xml

```
<!--
Copyright 2015 by Siemens AG and Wolfram Kuhnert
(wolfram.kuhnert@siemens.com)
Permission to use, copy, modify, distribute, and sell this softwareand its
documentation for any purpose is hereby granted without fee,provided that
the above copyright notice appear in all copies and that both that copyright
notice and this permission notice appearin supporting documentation, and
that the names of Siemens AG and Wolfram Kuhnertnot be used in advertising
or publicity pertaining todistribution of the software without specific,
written priorpermission.
Siemens AG and Wolfram Kuhnert make no representations about the
suitability of this software for any purpose. It isprovided "as is" without
express or implied warranty.
Required software version: Operate 4.7 Sp1 HF1
-->

<!--
    typedef struct tagRECT {
     LONG left;
     LONG top;
     LONG right;
     LONG bottom;
    } RECT;
-->

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

<typedef name="StructPoint" type="struct" >
  <element name="x" type="int">0</element>
  <element name="y" type="int">0</element>
</typedef>

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

<typedef name="StructMDLimits" type="struct" >
  <element name="lowerLimit" type="double">0</element>
  <element name="upperLimit" type="double">0</element>
  <element name="arrayDim1" >0</element>
  <element name="arrayDim2" >0</element>
</typedef>
```

3.10 元素定址

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

3.10.1 PLC 定址

PLC 地址分配从路径 **plc** 开始。

表格 3-3　允许使用下列地址：

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

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

表格 3-4　数据格式 **f**：

B	字节
W	字
D	双字

在位地址分配时取消数据格式标识。

地址 **x**：

有效的 S7 200 地址名称

位地址分配：

b – 位号

示例：

```
<data name = "plc/mb170">1</data>
<data name = "i0.1"> 1 </data>
<op> "m19.2" = 1 </op>
refvar="plc/db9062.dbb0:string"
```

3.10.2 NC 变量定址

NC 变量的定址从路径 **nck** 开始。

在该部分之后跟有数据地址，其结构可以查阅 NC 变量和接口信号的参数手册。

示例：

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

3.10.3 通道专用定址

如果未在地址记号中给定通道编号，则会始终访问操作软件的通道 1。

如果需要读取某一特定通道的数据，则可在地址中添加标识 **u** (Unit) 以及所需通道号。

示例：

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

3.10.4 在运行时生成 NC/PLC 地址

支持在运行时生成地址名称。

此时可将字符串变量的内容作为地址用在操作指令以及功能 nc.cap.read 和 nc.cap.write 中。

对于该定址模式应注意以下事项：

- 变量名称写在引号内。
- 在变量名称前标记三个 ‚$‘ 符号。

句法：

```
"$$$variable name"
```

示例：

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

3.10.5 驱动组件的定址

驱动组件的定址从路径 **drive** 开始。

后面跟有驱动设备的详细规格：

CU – Control Unit（控制单元）

DC – Drive Control（电机模块）

CULNK – 扩展模块 (HUBs)

TM – Terminal-Module（端子模块）

LM – Line-Module（电源模块）

该部分要添加上待设置参数。

示例：

```
<LET name="r0002_content"></LET>
<LET name="p107_content"></LET>

<!– 读取 CU 上 r0002 的值 ->

<OP> r0002_content = "drive/cu/r0002" </OP>
<OP> r0002_content = "drive/cu/r0002[CU1]" </OP>

<!– 读取 NX1 上 r0002 的值 ->
<OP> r0002_content = "drive/cu/r0002[CU2]" </OP>

<!– 读取 CU 上 p107[0] 的值 ->
<OP> p107_content = "drive/cu/p107[0]" </OP>
<PRINT text="%d"> p107_content </PRINT>

<!– 读取 CU 上 p107[0] 的值 ->

<OP> p107_content = "drive/cu/p107[0, CU1]" </OP>
<PRINT text="%d"> p107_content </PRINT>

<!– 读取 NX1 上 p107[0] 的值 ->

<OP> p107_content = "drive/cu/p107[0, CU2]" </OP>
<PRINT text="%d"> p107_content </PRINT>
```

驱动对象的地址分配：

为了给单个对象分配地址，要在参数之后的方括号中给出所需的对象。

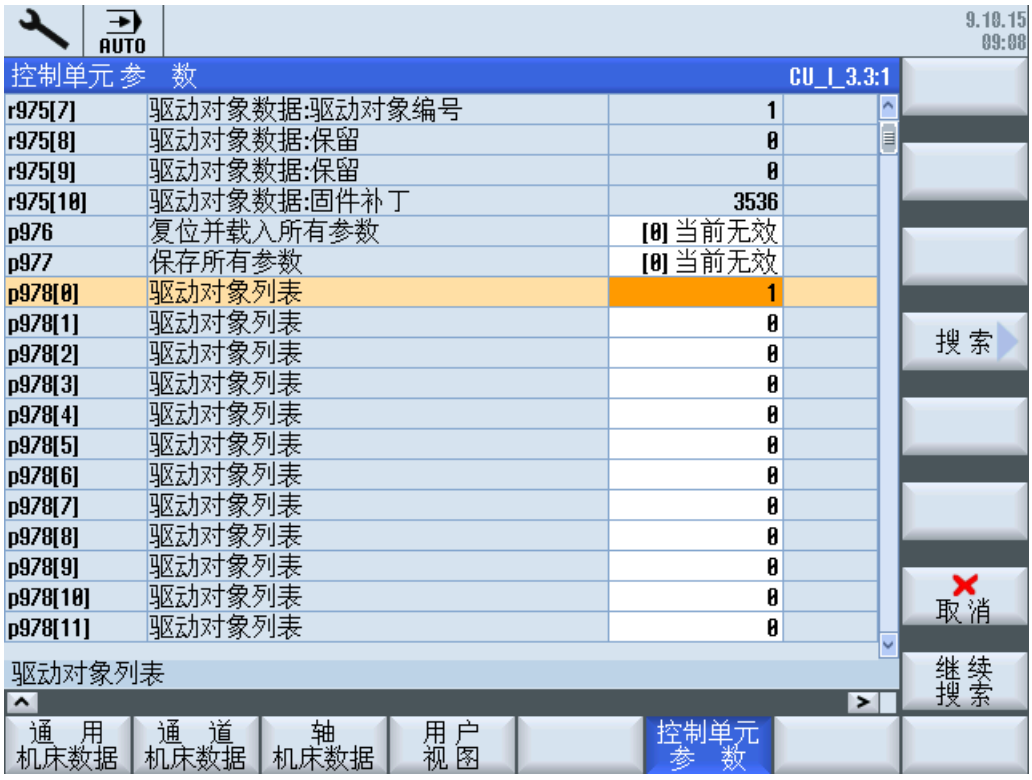


图 3-7 控制系统上的驱动参数 p0978 [...]

参数号[do<DO-index>]

示例：

```
p0092[do1]
```

可以选择利用“替代符号”**$<变量名称>** 从一个局部变量中读取驱动下标。

```
z.B. DO$局部变量
```

示例：

```
<DATA name ="drive/cu/p0092">1</DATA>
<DATA name ="drive/dc/p0092[do1] ">1</DATA>
```

间接地址分配：

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

3.10.6 示例：测定电机模块的 DO 号

11 型电机模块的 DO 号可通过以下方式测定：

所有已连接的驱动对象都根据其接口编号列在相应 CU 的数组 p978 中。同时会在数组 p101 中列出组件编号并在数组 p107 中列出组件类型。

为下列组件类型分别使用一个自己的下标：

CU – Control Units（控制单元）

DC – Drive Controls （电机模块）

CULNK – 扩展模块 (HUBs)

TM – Terminal-Module（端子模块）

LM – Line-Module（电源模块）

定址下标按这样的方式测定，将每个所连接的 CU 的数组 p0107 按升序通过，根据所找到类型的出现次序将类型下标增加一。基准值为一。如果在该数组中找到了 NX 组件，则在当前数组完全通过后，才会对 NX 组件继续进行计数。对 NX 组件和 CU 组件的处理按照发现的顺序进行。

下标测定：CUI 与 NX

CUI		NX1		定址下标	
				DO	CU
p107[0]	[3]SINAMICS				1
p107[2]	[11]SERVO			1	
p107[3]	[11]SERVO			2	
p107[4]	[11]SERVO			3	
p107[5]	[254]CU-LINK				2
p107[6]	[11]SERVO			4	
		p107[1]	[11]SERVO	5	
		p107[2]	[11]SERVO	6	
		p107[3]	[11]SERVO	7	

在该拓扑中含有七个电机模块。下标一至四指向分配给 CU 的电机模块。下标五至七指向 NX 的电机模块。访问 CU 使用下标一。NX 使用下标二来定址。

脚本示例：xmldial.xml 和 drv_sys_hlpfunct.xml

xmldial.xml

```
<DialogGui>

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

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

 <form name="main_form">
  <init>
   <caption>Component arrangement</caption>
    <let name="count" />
    <let name="str" type="string" />
    <let name="do_name" type="string" />
    <let name="cu_name" type="string" />
    <let name="cui_idx" />

    <op>
     cui_idx = 0;
    </op>

    <function name="load_component" />
    <print text="%d CUs found">num_cus</print>

    <function name="calculate_do_index" />

    <control name="list_comp_no_do_idx" xpos="8" ypos="80"
fieldtype="listbox" width="360" height="500" >
     <property item_data="100" />
    </control>

    <op>
     count = 0;
    </op>

    <while>
    <condition>count &lt; 32 && address_idx_map[$count].comp_no != 0</
condition>

    <op>
     do_name= _T"";
    </op>

    <function name="read_do_name_fast" return="do_name">count</function>
```

xmldial.xml

```
    <function name="read_cu_name_fast"
return="cu_name">address_idx_map[$count].cu_idx</function>

    <print name="str" text="%3d %3d %3d %s
%s">address_idx_map[$count].cu_idx, address_idx_map[$count].comp_no,
address_idx_map[$count].do_idx, do_name, cu_name</print>
    <function name="control.additem">_T"list_comp_no_do_idx", str, count</
function>

    <op>
     count = count +1;
    </op>
    </while>

   <paint>
    <text xpos="8" ypos="30">Motor moduls</text>
    <text xpos="8" ypos="60">CU</text>
    <text xpos="40" ypos="60">Comp.</text>
    <text xpos="92" ypos="60">DO Index</text>
    <text xpos="172" ypos="60">Name</text>
  </paint>

 </form>
</DialogGui>
```

drv_sys_hlpfunct.xml

```
<typedef name="components" type="struct">
 <element name="cu_p978" dim="25" />
 <element name="do_p0101" dim="25" />
 <element name="do_p0107" dim="25" />
</typedef>

<typedef name="comp_no_do_idx_map" type="struct">
 <!-- cu index -->
 <element name="cu_idx" />
 <!-- component number -->
 <element name="comp_no" />
 <!-- address index -->
 <element name="do_idx" />
</typedef>

<let name="componentsList" dim="5" type="components" />
<let name="address_idx_map" dim="32" type="comp_no_do_idx_map" />
<let name="num_cus" />
<let name="_drv_sys_comp_array_size">23</let>

<!-- ------------------------------------------------------------

 function: load_components_description
 This function loads the parameter arrays p978, p101 and p107 into a local
memory

 input:
 cuno: CU index 0 based (index 0 == CUI)

 output:
 componentsList structure

 ------------------------------------------------------------------ -->

<function_body name="load_components_description" parameter="cuno">
 <let name="count" />
 <let name="next_nx" >0</let>
 <let name="cuidx"></let>
 <let name="error" />

 <op>
  count = _drv_sys_comp_array_size;
  next_nx = cuno;
  cuidx = cuno+1;
 </op>

 <print text="gather data" />
```

drv_sys_hlpfunct.xml

```
 <function name="ncfunc.cap.read" return="error"
rows="$count">componentsList[$cuno].cu_p978, "drive/cu/p0978[0, cu
$cuidx]"</function>
 <function name="ncfunc.cap.read" return="error"
rows="$count">componentsList[$cuno].do_p0101, "drive/cu/p0101[0, cu
$cuidx]"</function>
 <function name="ncfunc.cap.read" return="error"
rows="$count">componentsList[$cuno].do_p0107, "drive/cu/p0107[0, cu
$cuidx]"</function>

 <print text="gather data finished" />
 <sleep value="20" />

 <op>
  count = 0;
 </op>

 <while>
  <condition>count < _drv_sys_comp_array_size</condition>
  <if>
   <condition>componentsList[$cuno].cu_p978[$count] == 60</condition>
  <then>
   <op>
    next_nx= next_nx +1;
   </op>
  <print text="next nx %d">next_nx</print>
  <function name="load_components_description">next_nx</function>
  </then>
  </if>

 <op>
  count = count+1;
 </op>
 </while>
 <op>
  num_cus = next_nx+1;
 </op>
</function_body>

<!-- ------------------------------------------------------------

 function: calculate_do_index
 This function is looking for components of type 11 (SERVO) and lists these
in the componentsList array.

 input:
 -

 output:
 componentsList array filled
```

drv_sys_hlpfunct.xml

```
 ------------------------------------------------------------------ -->

<function_body name="calculate_do_index">
 <let name="cuno" />
 <let name="count" />
 <let name="do_index" >1</let>
 <let name="map_index" />
 <while>
  <condition>
   cuno < num_cus</condition>
   <op>
    count = 0;
   </op>
  <while>
   <condition>count < _drv_sys_comp_array_size</condition>
   <if>
    <condition> componentsList[$cuno].do_p0107[$count] == 11</condition>
    <then>
     <op>
      address_idx_map[$map_index].cu_idx = cuno+1;
      address_idx_map[$map_index].comp_no =
componentsList[$cuno].do_p0101[$count];
      address_idx_map[$map_index].do_idx = do_index;
      do_index = do_index+1;
      map_index = map_index +1;
     </op>
    </then>
   </if>
   <op>
    count = count +1;
   </op>
  </while>
  <op>
   cuno = cuno +1;
  </op>
 </while>
</function_body>
```

3.10.7 机床数据和设定数据的定址

可以使用 $ 符号标记驱动和设定数据，后面跟有数据名称。

机床数据：

$Mx_<名称[**索引, AX<轴编号>**]>

设定数据：

$Sx_<名称[索引, AX<轴编号>]>

x：

N – 通用的机床或设定数据

C – 通道专用的机床或设定数据

A – 轴专用的机床或设定数据

索引：

参数在该数组中给出数据索引。

AX<轴编号>：

在轴专用数据时对所需要的轴（**<轴编号>**）进行详细说明。

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

例如 AX$ 局部变量

示例：

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

轴的直接地址分配：

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

轴的间接地址分配：

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

3.10.8 通道专用机床数据

如果未在地址记号中给定通道编号，则会始终访问当前设置的操作软件通道。

如果需要读取某一特定通道的数据，则可在地址中使用方括号添加标识 **u** (Unit) 以及所需通道号。使用数组定址时，通道数据是方括号中的最后一个自变量。

示例：

```
$MC_RESET_MODE_MASK
```

或者

```
$MC_AXCONF_GEOAX_ASSIGN_TAB[0]
```

```
$MC_RESET_MODE_MASK[u1]
```

或者

```
$MC_AXCONF_GEOAX_ASSIGN_TAB[0, u1]
$MC_RESET_MODE_MASK
```

3.10.9 用户数据的定址

用户数据的定址路径以 **gud** 开始，之后是对其是全局 GUD 还是通道专用 GUD 的说明，再接着是 GUD 范围和 GUD 名称。

如果是数组型数据，还需要在 GUD 名称后的方括号中指定数组下标。

示例：

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

全局用户数据的地址表达方式

此定址路径以 gud 开始，之后是范围说明 NCK，再接着是具体的 GUD 范围：

GUD 范围	分配
sgud	西门子的 GUD
mgud	机床制造商的 GUD
ugud	用户的 GUD

通道专用用户数据的地址表达方式

此定址路径以 gud 开始，之后是范围说明 CHANNEL，再接着是具体的 GUD 范围。

GUD 范围后面跟着 GUD 名称。
如果是数组型数据，还需要在 GUD 名称后的的方括号中指定数组下标。

示例：

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

多维数组的地址表达方式

在表示多维数组的地址时，行下标后要跟着列下标。两个下标之间用一个点隔开。

对于通道专用的 GUD 而言：

- 通道号的表示方式为：字母 **u**（表示 unit）后加数字，前面或后面加上行下标/列下标。
- 参数之间以逗号分隔。
- 没有指定通道号时，系统访问第一条通道。

示例：

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

或

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

3.11 预定义功能

脚本语言提供有各种字符串运算和标准数学函数。下文列出的函数名称是预留的并且不能改写。

函数名称	含义

Ncfunc cap read	该函数会将指定地址中的值复制到局部变量中。如果读取成功，返回变量中的值为零。
	
	与运算指令相反，该函数在出错时不会终止脚本指令的执行。
	
	属性：
	• **return** - 执行状态
	– 值 = 0 - 正常
	– 值 = 1 - 变量的读取无法进行
	• **rows** - 数组中额外读取的行数（可选）
	如果给定了数组下标的参考变量，该函数可将读取的值从该下标复制到目标变量中。
	
	语法：
	`<function name="ncfunc.cap.read" return="error"> lokale variable, "address"</function>`
	
	示例：
	`<let name="error"></let>`
	`<function name="ncfunc.cap.read" return="error"> 3, "drive/cu/p0009"</function>`
	`<if>`
	`<condition>error != 0</condition>`
	`<then>`
	`<break />`
	`</then>`
	`</if>`
	
	或
	
	`<let name="cu_p978" dim="25" ></let>`
	`...`
	`...`
	`<function name="ncfunc.cap.read" return="error" rows="23">cu_p978, "drive/cu/p0978[0, 1]"`
	`</function>`

函数名称	含义

Ncfunc cap write	该函数会向指定变量写入一个值。如果写入成功，返回变量中的值为零。
	
	与运算指令相反，该函数在出错时不会终止脚本指令的执行。
	
	属性：
	• **return** - 执行状态
	– 值 = 0 - 正常
	– 值 = 1 - 变量的读取无法进行
	• **rows** - 数组中额外写入的行数（可选）
	如果给定了数组下标的参考变量，该函数可将值从该下标复制到目标变量中。
	
	语法：
	`<function name="ncfunc.cap.read" return="error"> local variable or constant, "address"</function>`
	
	示例：
	`<let name="error"></let>`
	`<function name="ncfunc.cap.write" return="error"> 0, "drive/cu/p0009"</function>`
	`<if>`
	`<condition>error != 0</condition>`
	`<then>`
	`<break />`
	`</then>`
	`</if>`
	
	或
	
	`<let name="cu_p978" dim="25" ></let>`
	`...`
	`...`
	`<function name="ncfunc.cap.write" return="error" rows="23">cu_p978, "drive/cu/p0978[0, 1]"`
	`</function>`

函数名称	含义

Ncfunc PI-Service	通过程序实例（PI）服务可将任务传输给 NCK。
	如果服务执行成功，该函数会将值 1 返回到返回变量中。
	
	刀具表的操作：
	_N_CREATO - 创建刀具
	_N_DELETO - 删除刀具
	_N_CREACE - 创建刀沿
	_N_DELECE - 删除刀沿
	
	零点偏移的激活：
	_N_SETUFR - 将当前的用户框架激活
	_N_SETUDT - 将当前的用户数据激活
	
	程序段查找：
	_N_FINDBL - 激活查找
	_N_FINDAB - 取消程序段查找
	
	机床数据的重新配置：
	_N_CONFIG 机床数据和 NEW_CONF 生效级（后续由操作员输入的）会被一同激活。
	
	语法：
	`<function name="ncfunc.pi_service" return="return var"> pi name, var1, var2, var3, var4, var5`
	`</function>`
	
	属性：
	name - 函数名称
	return - 保存执行结果的变量名称：
	• 值 == 1 – 成功执行任务
	• 值 == 0 – 任务失败
	
	标签值：

函数名称	含义
	pi name - PI 服务的名称（字符串） var1 到 var5 - PI 专用自变量

函数名称	含义

Ncfunc PI-Service 续	**自变量：**
	• _N_CREATO
	var1 - 刀具号
	• _N_DELETO
	var1 - 刀具号
	• _N_CREACE
	var1 - 刀具号
	var2 - 刀沿号
	• _N_DELECE
	var1 - 刀具号
	var2 - 刀沿号
	• _N_SETUFR
	无自变量
	• _N_SETUDT
	var1 - 待激活的用户数据区域
	– 1 - 刀具补偿数据
	– 2 - 有效基本框架
	– 3 - 有效可设置框架
	• _N_FINDBL
	var1 - 查找模式
	– 2 - 带轮廓计算的查找
	– 4 - 查找到程序段末尾
	– 1 - 不带计算的查找
	• _N_FINDAB
	无自变量
	• _N_CONFIG
	无自变量
	
	示例：
	创建刀具 - 刀具号 3
	
	`<function name="ncfunc.pi_service">_T"_N_CREATO", 3</function>`
	
	删除刀具 5 的刀沿 1
	
	`<function name="ncfunc.pi_service">_T"_N_DELECE", 5, 1`
	`</function>`

函数名称	含义

ncfunc chan PI-Service	该函数会执行基于通道的 PI 服务。在 PI 服务名称后传输通道编号。然后是所有其他调用参数。
	
	参数：
	channel - 通道编号
	
	语法：
	`<function name="ncfunc.chan_pi_service" return="error">`
	`_T"_N_SETUFR", channel, ...`
	`</function>`
	
	示例：
	`<let name="chan" >1</let>`
	`<function name="ncfunc.chan_pi_service" return="error">`
	`_T"_N_SETUFR", chan</function>`
	
	`<function name="ncfunc.chan_pi_service" return="error">`
	`_T"_N_SETUDT", chan, _T"016", _T"00000", _T"00000"</function>`

Ncfunc display resolution	该函数提供由控制系统确定的浮点数转换规则。提供了字符串变量。
	
	另见显示机床数据 MD203 DISPLAY_RESOLUTION 和
	MD204 DISPLAY_RESOLUTION_INCH
	
	语法：
	`<function name="ncfunc.displayresolution" return="dislay_res" />`
	
	示例：
	`<let name="dislay_res" type="string"></let>`
	...
	`<function name="ncfunc.displayresolution" return="dislay_res" />`
	
	`<control name = "cdistToGo" xpos = "210" ypos = "156" refvar="nck/Channel/GeometricAxis/progDistToGo[2]" hotlink="true" height="34" fieldtype="readonly" format="$$$dislay_res" time="superfast" color_bk="#ffffff"/>`

函数名称	含义

Ncfunc Get drive by axis name	该函数通过给定所属的轴名称来给出电机模块的寻址下标。
	如未识别到相应的分配，函数返回数值 -1。如果未进行轴/驱动的分配，就会出现此情况。
	
	参数：
	str - 轴名称
	
	返回值：
	电机模块下标 - 变量名称，此变量中会写入已获得的下标。
	
	语法：
	`<function name="NCFUNC.GETDRVBYAX_NAME" return="<index>">`
	`<axis name></function>`
	
	示例：
	`<let name="drv_idx" />`
	`<function name="NCFUNC.GETDRVBYAX_NAME" return="drv_idx">_T"X1"`
	`</function>`
	`<control name = "c_do2" xpos = "140" ypos = "114" width = "320"`
	`fieldtype="itemlist" refvar="drive/dc/p10[`**`do$ drv_idx`**`]"`
	`hotlink="true" color_bk="#f1f1f1">`
	`<item value="0">Ready</item>`
	`<item value="1">Quick commissioning</item>`
	`<item value="2">Power unit commissioning</item>`
	`<item value="3">Motor commissioning</item>`
	`<item value="4">Encoder commissioning</item>`
	`<item value="5">Technological application/units`
	`</item>`
	`<item value="15">Data sets</item>`
	`<item value="17">Basic positioning commissioning`
	`</item>`
	`<item value="25">Commissioning the position control</item>`
	`<item value="29">Download</item>`
	`<item value="30">Parameter reset</item>`
	`<item value="95">Safety Integrated commissioning`
	`</item>`
	`</control>`
	`</init>`

函数名称	含义
Ncfunc Get drive by axis index	该函数通过给定所属的轴下标来给出电机模块的寻址下标。 如未识别到相应的分配，函数返回数值 -1。如果未进行轴/驱动的分配，就会出现此情况。 **参数：** **index** - 轴下标 **返回值：** 电机模块下标 - 变量名称，此变量中会写入已获得的下标。 **语法：** <function name="NCFUNC.GETDRVBYAX_IDX" return="<index>"> <axis index></function> **示例：** <let name="drv_idx" /> <function name="NCFUNC.GETDRVBYAX_IDX" return="drv_idx">1</function> <control name = "c_do2" xpos = "140" ypos = "114" width = "320" fieldtype="itemlist" refvar="drive/dc/p10[**do$ drv_idx**]" hotlink="true" color_bk="#f1f1f1"> <item value="0">Ready</item> <item value="1">Quick commissioning</item> <item value="2">Power unit commissioning</item> <item value="3">Motor commissioning</item> <item value="4">Encoder commissioning</item> <item value="5">Technological application/units</item> <item value="15">Data sets</item> <item value="17">Basic positioning commissioning</item> <item value="25">Commissioning the position control</item> <item value="29">Download</item> <item value="30">Parameter reset</item> <item value="95">Safety Integrated commissioning</item> </control> </init>

函数名称	含义

Ncfunc Get drive by drive name	该函数通过给定电机模块名称来给出电机模块的寻址下标。
	如未识别到相应的分配，函数返回数值 -1。如果未进行轴/驱动的分配，就会出现此情况。
	
	参数：
	string - 电机模块名称
	
	返回值：
	电机模块下标 - 变量名称，此变量中会写入已获得的下标。
	
	语法：
	`<function name="NCFUNC.GETDRVBYDRV_NAME" return="<index>">`
	`<drive name></function>`
	
	示例：
	`<let name="drv_idx" />`
	`<function name="NCFUNC.GETDRVBYDRV_NAME"`
	`return="drv_idx">_T"SERVO_3.13:4"</function>`
	`<control name = "c_do2" xpos = "140" ypos = "114" width = "320"`
	`fieldtype="itemlist" refvar="drive/dc/p10[`**`do$ drv_idx`**`]"`
	`hotlink="true" color_bk="#f1f1f1">`
	` <item value="0">Ready</item>`
	` <item value="1">Quick commissioning</item>`
	` <item value="2">Power unit commissioning</item>`
	` <item value="3">Motor commissioning</item>`
	` <item value="4">Encoder commissioning</item>`
	` <item value="5">Technological application/units`
	`</item>`
	` <item value="15">Data sets</item>`
	` <item value="17">Basic positioning commissioning`
	`</item>`
	` <item value="25">Commissioning the position control</item>`
	` <item value="29">Download</item>`
	` <item value="30">Parameter reset</item>`
	` <item value="95">Safety Integrated commissioning`
	`</item>`
	`</control>`
	`</init>`

函数名称	含义

Ncfunc Get drive by bud address	该函数通过给定电机模块总线地址来给出电机模块的寻址下标。
	如未识别到相应的分配，函数返回数值 -1。如果未进行轴/驱动的分配，就会出现此情况。
	
	参数：
	bus - 总线编号
	slave - 从站编号
	component - 组件编号
	
	返回值：
	电机模块下标 - 变量名称，此变量中会写入已获得的下标。
	
	语法：
	`<function name="NCFUNC.GETDRVBYBUS_ADDR" return="<index>"><bus>,<slave>,<component></function>`
	
	示例：
	`<let name="drv_idx" />`
	`<function name="NCFUNC.GETDRVBYBUS_ADDR" return="drv_idx"> 3,13,4 </function>`
	`<control name = "c_do2" xpos = "140" ypos = "114" width = "320" fieldtype="itemlist" refvar="drive/dc/p10[`**`do$ drv_idx`**`]" hotlink="true" color_bk="#f1f1f1">`
	`<item value="0">Ready</item>`
	`<item value="1">Quick commissioning</item>`
	`<item value="2">Power unit commissioning</item>`
	`<item value="3">Motor commissioning</item>`
	`<item value="4">Encoder commissioning</item>`
	`<item value="5">Technological application/units </item>`
	`<item value="15">Data sets</item>`
	`<item value="17">Basic positioning commissioning </item>`
	`<item value="25">Commissioning the position control</item>`
	`<item value="29">Download</item>`
	`<item value="30">Parameter reset</item>`
	`<item value="95">Safety Integrated commissioning </item>`
	`</control>`
	`</init>`

函数名称	含义

Ncfunc Get MD limits	该函数将机床数据限值复制到数据类型为 **StructMDLimits** 的指定变量中。结构定义包含在文件 struct_def.xml 中。
	更多信息参见章节“创建 struct_def.xml 文件 (页 155)”
	
	参数：
	range - 必须输入一个字符串来指定机床数据范围：
	• **displayMD** - 显示机床数据
	• **generalMD** - NC 全局机床数据
	• **channelMD** - NC 通道专用机床数据
	• **axisMD** - NC 轴专用机床数据
	• **generalSettingMD** - NC 全局设定数据
	• **channelSettingMD** - NC 通道专用设定数据
	• **axisSettingMD** - NC 轴专用设定数据
	• **channelGUD** - NC 通道专用用户数据
	• **globalGUD** - NC 全局用户数据
	MD-number - 范围的机床数据编号
	variabel name - 调用该函数之后，该变量包含 Limits 的值
	range/unit - 可选项（如：通道编号）
	
	语法：
	`<function name="NCFUNC.GETMD_LIMITS"><range>, <MD-number, <variabel name>, <range/unit>></function>`
	
	示例：
	`<let name="limits" type="StructMDLimits" /> <function name="NCFUNC.GETMD_LIMITS">_T"generalMD", 19220, _T"limits"</function> <op> upper_BAGS = limits.upperLimit; </op>`

函数名称	含义

Ncfunc bico to int	该函数将指定的 BICO 格式的字符串转换为整型值。（参见 SINAMICS）。
	
	语法：
	`<function name="ncfunc.bicotoint" return="integer variable">`
	`bico-string</function>`
	
	示例：
	`<let name="s_np0480_0" type="string"></let>`
	`<let name="i_p0480_0">0</let>`
	
	`<function name="ncfunc.bicotoint" return="i_p0480_0">s_np0480_0`
	`</function>`

Ncfunc int to bico	该函数将整型值转换为 BICO 格式的字符串（参见 SINAMICS）。
	
	语法：
	`<function name="ncfunc.inttobico"`
	`return="string variable">integer variable</function>`
	
	示例：
	`<function name="ncfunc.inttobico"`
	`return="s_p0480_0">"drive/dc/p0480[0, DO2]"</function>`

Ncfunc is bico str valid	当结果是以 BICO 格式给定的字符串时，该函数会提供零值。（参见 SINAMICS）
	
	语法：
	`<function name="ncfunc.isbicostrvalid"`
	`return="integer variable">string varaible</function>`
	
	示例：
	`…`
	`<let name="s_np0480_0" type="string"></let>`
	`…`
	`…`
	`<control name = "cp0480_0" xpos = "402"`
	`ypos = "76" hotlink="true" refvar="s_np0480_0" >`
	`  <property item_data="4001" />`
	`</control>`
	`…`
	`…`
	`<function name="ncfunc.isbicostrvalid"`
	`return="valid">cp0480_0</function>`

函数名称	含义

Ncfunc password	该函数用来设置或取消 NC 口令等级。
	• 设置口令：
	所需口令等级的口令将作为参数给定。
	• 删除口令：
	空字符串会取消口令等级。
	
	语法：
	`<function name="ncfunc.password">口令`
	`</function>`
	
	示例：
	`<let name="口令"type="string"></let>`
	
	`<function name="ncfunc.password" > 口令`
	`</function>`
	`<function name="ncfunc.password" > _T"CUSTOMER"`
	`</function>`
	
	删除口令：
	`<function name="ncfunc.password" > _T"" </function>`

Control form color	该函数以字符串形式提供对话框的文本颜色或背景颜色。
	更多信息参见章节“颜色代码 (页 81)”
	
	区域：
	• BACKGROUND – 请求背景颜色值
	• TEXT – 请求文本（前景）颜色值
	
	语法：
	`<function name="control.formcolor"`
	`return="variable">_T"区域"</function>`
	
	示例：
	`<let name="bk_color" type="string"></let>`
	
	`<function name="control.formcolor"`
	`return="bk_color">_T"BACKGROUND"</function>`

函数名称	含义

Control local time	该函数将本地时间复制到包含 7 个元素的数组中。
	变量名称将会用于调用参数。
	数组元素中保存以下内容：
	• 下标 0 - 年份
	• 下标 1 - 月份
	• 下标 2 - 星期
	• 下标 3 - 日
	• 下标 4 - 小时
	• 下标 5 - 分钟
	• 下标 6 - 秒
	
	语法：
	`<function name ="control.localtime">_T"time_array"`
	`</function>`
	
	示例：
	`<!-- index`
	`0 = Year`
	`1 = Month`
	`2 = Day of week`
	`3 = Day`
	`4 = Hour`
	`5 = Minute`
	`6 = Second`
	`-->`
	`<let name="time_array" dim="7" />`
	
	`<function name ="control.localtime">_T"time_array"`
	`</function>`

Control exist	如果指定控件存在，该函数返回数值 1。
	
	语法：
	`<function name="CONTROL.EXIST"`
	`return="<return variable>"><control name></function>`
	
	示例：
	`<let name="ret" /> <function name="CONTROL.EXIST"`
	`return="ret">_T"edit"</function>`

函数名称	含义

Control is modified	如果指定的 EDIT 控件的内容已更改，该函数返回数值 1。
	
	语法：
	`<function name="CONTROL.ISMODIFIED" return="<return variable>"><control name></function>`
	
	示例：
	`<let name="ret" /> <function name="CONTROL.ISMODIFIED" return="ret">_T"edit"</function>`

Control is visible	如果指定控件可见，该函数返回数值 1。
	
	语法：
	`<function name="CONTROL.ISVISIBLE" return="<return variable>"><control name></function>`
	
	示例：
	`<let name="ret" /><function name="CONTROL.ISVISIBLE" return="ret">_T"edit"</function>`

Control set modified	该函数会更改指定 EDIT 控件的“更改”标记。
	
	参数：
	control name
	• 值 = 1 - 字段标记为已更改。
	• 值 = 0 - 字段标记为未更改。
	
	语法：
	`<function name="CONTROL.SETMODIFIED" return="<return variable>"><control name>, <Flag></function>`
	
	示例：
	`<let name="b" >1</let>`
	`<let name="ret" />`
	`<control name="edit" xpos="10" ypos="80" width="30" refvar="b" hotlink = "true" />`
	`<function name="CONTROL.SETMODIFIED" return="ret">_T"edit", 1</function>`

函数名称	含义

Control set working area	如果更改了**滚动视图**的内容，例如：删除或添加控件，则要重新计算滚动视图的可见区域。
	调用此函数执行计算。
	
	值：
	滚动视图名称
	
	示例：
	`...`
	`<control name="area" xpos="8" ypos="72" width="554" height="320" fieldtype="scrollarea" >`
	`  <control name="test" xpos="20" ypos="40"  width="20" fieldtype="readonly" />`
	`</control >`
	
	`<control name="test1" xpos="20" ypos="80"  width="20" fieldtype="readonly" parent="area"/>`
	`<control name="test2" xpos="20" ypos="100"  width="20" fieldtype="readonly" parent="area"/>`
	
	`<function name="CONTROL.SETWORKINGAREA">_T"area"</function>`
	`...`

函数名称	含义

String to compare	对两个字符串从书写方式上进行比较。
	当字符串相同时该函数提供返回值零，当第一个字符串小于第二个时返回值小于零或者当第二个字符串小于第一个时返回值大于零。
	参数：
	str1 - 字符串
	str2 - 比较字符串
	
	语法：
	`<function name="string.cmp" return ="<int var>" > str1, str2 </function>`
	
	示例：
	`<let name="rval">0</let>`
	`<let name="str1" type="string">A brown bear hunts a brown dog.</let>`
	`<let name="str2" type="string">A brown bear hunts a brown dog.</let>`
	
	`<function name="string.cmp" return="rval"> str1, str2 </function>`
	结果：
	rval= 0

函数名称	含义

不区分大/小写的字符串比较	在不区分大/小写的情况下对两个字符串从书写方式上进行比较。
	当字符串相同时该函数提供返回值零，当第一个字符串小于第二个时返回值小于零或者当第二个字符串小于第一个时返回值大于零。
	
	参数：
	str1 - 字符串
	str2 - 比较字符串
	
	语法：
	`<function name="string.icmp" return ="<int var>" > str1, str2 </function>`
	
	示例：
	`<let name="rval">0</let>`
	
	`<let name="str1" type="string">A brown bear hunts a brown dog.</let>`
	`<let name="str2" type="string">A brown Bear hunts a brown Dog.</let>`
	`<function name="string.icmp" return="rval"> str1, str2 </function>`
	结果：
	rval= 0

String left	该函数提取字符串 1 中第一个的 nCount 字符并将其复制到返回变量中。
	
	参数：
	str1 - 字符串
	nCount - 字符数量
	
	语法：
	`<function name="string.left" return="<result string>"> str1,  nCount </function>`
	
	示例：
	`<let name="str1" type="string">A brown bear hunts a brown dog.</let>`
	`<let name="str2" type="string"></let>`
	
	`<function name="string.left" return="str2"> str1, 12 </function>`
	结果：
	str2="A brown bear"

函数名称	含义

String right	该函数提取字符串 1 中最后一个 nCount 字符并将其复制到返回变量中。
	
	参数：
	str1 - 字符串
	nCount - 字符数量
	
	语法：
	`<function name="string.right" return="<result string>"> str1, nCount </function>`
	
	示例：
	`<let name="str1" type="string">A brown bear hunts a brown dog.</let>`
	`<let name="str2" type="string"></let>`
	
	`<function name="string.right" return="str2"> str1, 10 </function>`
	结果：
	str2="brown dog."

String middle	该函数从下标 iFirst 开始提取字符串 1 中指定数量的字符并将其复制到返回变量中。
	
	参数：
	str1 - 字符串
	iFirst - 起始下标
	nCount - 字符数量
	
	语法：
	`<function name="string.middle"`
	`return="<result string>"> str1, iFirst, nCount </function>`
	
	示例：
	`<let name="str1" type="string">A brown bear hunts a brown dog.</let>`
	`<let name="str2" type="string"></let>`
	
	`<function name="string.middle" return="str2"> str1, 2, 5 </function>`
	结果：
	str2="brown"

函数名称	含义
String length	该函数可以提供字符串的字符数量。 **参数：** **str1** - 字符串 **语法：** <function name="string.length" return="<int var>"> str1 </function> **示例：** <let name="length">0</let> <let name="str1" type="string">A brown bear hunts a brown dog.</let> <function name="string.length" return="length"> str1 </function> **结果：** length = 31
Strings to replace	该函数可以使用新的字符串替代所有查找到的字符串部分。 **参数：** **string** - 字符串变量 **find string** - 需替换的字符串 **new string** - 新字符串 **语法：** <function name="string.replace"> string, find string, new string </function> **示例：** <let name="str1" type="string">A brown bear hunts a brown dog. </let> <function name="string.replace" > str1, _T"a brown dog" , _T"a big salmon"</function> **结果：** str1 = "A brown bear hunts a big salmon!"

函数名称	含义

Strings to remove	该函数可以删除所有查找到的字符串部分。
	
	参数：
	string - 字符串变量
	remove string - 要删除的字符串部分
	
	语法：
	`<function name="string.remove"> string, remove string </function>`
	
	示例：
	`<let name="index">0</let>`
	
	`<let name="str1" type="string">A brown bear hunts a brown dog. </let>`
	`<function name="string.remove" > str1, _T"a brown dog" </function>`
	结果：
	str1 = "A brown bear hunts"

Strings to insert	该函数可以在指定的下标中插入一个字符串。
	
	参数：
	string - 字符串变量
	index - 下标（从零开始）
	insert string - 要插入的字符串
	
	语法：
	`<function name="string.insert"> string, index, insert string`
	`</function>`
	
	示例：
	`<let name="str1" type="string">A brown bear hunts. </let>`
	`<let name="str2" type="string">a brown dog</let>`
	
	`<function name="string.insert" > str1, 19, str2`
	`</function>`
	结果：
	str1 = "A brown bear hunts a brown dog"

函数名称	含义

String delete	该函数可以从指定的启始位置开始删除确定数量的字符。
	
	参数：
	string - 字符串变量
	start index - 起始下标（从零开始）
	nCount - 要删除的字符数量
	
	语法：
	`<function name="string.delete"> string, start index , nCount </function>`
	
	示例：
	`<let name="str1" type="string">A brown bear hunts. </let>`
	
	`<function name="string.delete" > str1, 2, 5 </function>`
	结果：
	str1 = "A bear hunts"

函数名称	含义
String find	该函数可以在所传输的字符串中查找与部分字符串一致的第一处地方。 如果找到了部分字符串，则该函数会提供到第一个字符的下标（从零开始），否则为 -1。 **参数：** **string** - 字符串变量 **findstring** - 要查找的字符串 **startindex** - 起始下标（可选） **语法：** <function name="string.find" return="<int val>"> str1, find string </function> **示例：** <let name="index">0</let> <let name="str1" type="string">A brown bear hunts a brown dog. </let> <function name="string.find" return="index"> str1, _T"brown" </function> **结果：** 下标 = 2 **或** <function name="string.find" return="index"> str1, _T"brown", 1 </function>

函数名称	含义

String reverse find	该函数可以在所传输的字符串中查找与部分字符串一致的最后一处地方。
	如果找到了部分字符串，则该函数会提供到第一个字符的下标（从零开始），否则为 -1。
	
	参数：
	string - 字符串变量
	find string - 要查找的字符串
	startindex - 起始下标（可选）
	
	语法：
	`<function name="string.reversefind"`
	`return="<int val>"> str1, find string </function>`
	
	示例：
	`<let name="index">0</let>`
	`<let name="str1" type="string">A brown bear hunts a brown dog. </let>`
	`<function name="string.reversefind" return="index"> str1, _T"brown"`
	`</function>`
	
	结果：
	下标 = 21
	
	或
	`<function name="string.reversefind"`
	`return="index"> str1, _T"brown" 10 </function>`
	
	结果：
	下标 = 2

函数名称	含义

String GetAt	该函数可读取特定位置的一个字符。
	参数：
	str - 字符串
	index - 待读取字符的下标，从零开始编号
	
	返回值：
	给出一个字符串变量名，字符应当储存在此变量中。
	
	语法：
	`<function name="string.getat" return="<result string>"><string>, <index></function>`
	
	示例：
	`<let name="resStr" type="string" />`
	`<function name="string.getat"`
	`return="resStr">_T"brown", 2</function>`

函数名称	含义

String pixel height	以像素为单位计算字符串的高度。在指定控件或窗体的当前字体格式设置下进行计算。控件名称作为字符串提供。如果指定的是空字符串，计算则基于窗体进行。
	
	参数：
	control name
	字符串
	字符串可直接指定为文本或指定为字符串变量。
	
	返回值：
	名称为整型变量，其中写入高度值。
	
	语法：
	`<function name="STRING.PIXELHEIGHT"`
	`return="<return variable>"><control name>,`
	`<varaible or text></function>`
	
	示例：
	根据窗体的字体计算宽度
	`<let name="pixelsh" />`
	`...`
	`<function name="STRING.PIXELHEIGHT"`
	`return="pixelsh">_T"", cedit1</function>`
	
	根据控件的字体计算宽度
	`<function name="STRING. PIXELHEIGHT"`
	`return="pixelsh"> cedit1, cedit1</function>`

函数名称	含义

String pixel width	以像素为单位计算字符串的宽度。在指定控件或窗体的当前字体格式设置下进行计算。控件名称作为字符串提供。如果指定的是空字符串，计算则基于窗体进行。
	
	参数：
	control name
	字符串
	字符串可直接指定为文本或指定为字符串变量。
	
	返回值：
	名称为整型变量，其中写入文本宽度值。
	
	语法：
	`<function name="STRING.PIXELWIDTH"`
	`return="<return variable>"><control name>,`
	`<varaible or text></function>`
	
	示例：
	根据窗体的字体计算宽度
	`<let name="pixels" />`
	`...`
	`<function name="STRING. PIXELWIDTH"`
	`return="pixels">_T"", cedit1</function>`
	
	根据控件的字体计算宽度
	`<function name="STRING.PIXELWIDTH"`
	`return="pixels"> cedit1, cedit1</function>`

函数名称	含义
String SetAt	该函数用于在特定位置写一个字符。此下标必须小于文本最大长度。 **参数：** **str** - 字符串 **char** - 应在特定位置写入的字符 **index** - 目标变量字符串的下标，从零开始编号 **语法：** <function name="string.setat" ><destination string>, <character string>, <index></function> **示例：** <let name="str" type="string" >br_wn</string> <function name="string.setat">str, ">_T"o", 2 </function>

函数名称	含义

String Split	该函数将一个字符串分隔为多个子字符串并将其复制到给定的字符串数组中。此分隔在给定的分隔符处发生。分隔符不会存储在子字符串中。
	如果找到的分隔符数量超过既定的数组大小，该函数会自动扩展数组。
	
	参数：
	str - 字符串
	char - 包含分隔符的变量名称
	number - 包含该函数执行完毕后产生的子字符串数量的变量名称。
	
	返回值：
	给出包含该函数执行完毕后的子字符串的字符串数组的名称。
	
	语法：
	`<function name=" string.split"`
	`return="<result string array>"><string>, <char>,`
	`<number></function>`
	
	示例：
	`<let name="strlist" type="string" dim="2"/>`
	`<let name="str_num" />`
	`<function name="string.split" return="strlist">`
	`_T"brown;green;blue;red", _T";", str_num </function>`

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

函数名称	含义

String trim right	该函数可以从字符串中删除后续的空格。
	
	参数：
	str1 - 字符串变量
	
	语法：
	`<function name="string.trimright" > str1 </function>`
	
	示例：
	`<let name="str1" type="string"> test trim right  </let>`
	`<function name="string.trimright" > str1`
	`</function>`
	结果：
	str1 = "test trim right"

Sinus	该函数可以计算所传输值（单位度）的正弦值。
	
	参数：
	double - 角度
	
	语法：
	`<function name="sin" return="<double val>"> double </function>`
	
	示例：
	`<let name= "sin_val" type="double"></let>`
	`<function name="sin" return="sin_val"> 20.0`
	`</function>`

函数名称	含义

Cosinus	该函数可以计算所传输值（单位度）的余弦值。
	
	参数：
	double - 角度
	
	语法：
	`<function name="cos" return="<double val>"> double </function>`
	
	示例：
	`<let name= "cos_val" type="double"></let>`
	`<function name="cos" return="cos_val"> 20.0`
	`</function>`

Tangens	该函数可以计算所传输值（单位度）的正切值。
	
	参数：
	double - 角度
	
	语法：
	`<function name="tan" return="<double val>"> double </function>`
	
	示例：
	`<let name= "tan_val" type="double"></let>`
	`<function name="tan" return="tan_val"> 20.0`
	`</function>`

ARCSIN	该函数可以计算所传输值（单位度）的反正弦值。
	
	参数：
	double - x 取值范围 -PI/2 至 +PI/2
	
	语法：
	`<function name="arcsin" return="<double val>"> double </function>`
	
	示例：
	`<let name= "arcsin_val" type="double"></let>`
	`<function name="arcsin" return="arcsin_val"> 20.0`
	`</function>`

函数名称	含义
ARCOS	该函数可以计算所传输值（单位度）的反余弦值。 **参数：** **double -** x 取值范围 -PI/2 至 +PI/2 **语法：** <function name="arcos" return="<double val>"> double </function> **示例：** <let name= "arccos_val" type="double"></let> <function name="arccos" return="arccos_val"> 20.0 </function>
ARCTAN	该函数可以计算所传输值（单位度）的反正切值。 **参数：** **double** - 反正切 y/x **语法：** <function name="arctan" return="<double val>"> double </function> **示例：** <let name= "arctan_val" type="double"></let> <function name="arctan" return="arctan_val"> 20.0 </function>
文件处理	

函数名称	含义

读取文件	该函数可以将指定文件的内容读到一个字符串变量中。
	或者可以将待读取的字符数量指定为第二参数。
	
	属性：
	name - 文件名只能用小写字母。
	需要访问另一个目录中的文件时，可采用相对路径，即目录从 appl 或 dvm 开始。
	return - 局部变量的名称
	
	参数：
	progname - 文件名
	number of characters - 待读取的字符数量，字节（可选）
	
	语法：
	`<function name="doc.readfromfile"`
	`return="<string var>"> progname, number of characters`
	`</function>`
	
	示例：
	`<let name = "my_var" type="string" ></let>`
	
	NC 文件系统
	`<function name="doc.readfromfile"`
	`return="my_var"> _T"n:\mpf\test.mpf"`
	`</function>`
	
	CF 卡
	`<function name="doc.readfromfile"`
	`return="my_var"> _T"f:\appl\test.mpf"`
	`</function>`
	
	或
	
	`<function name="doc.readfromfile"`
	`return="my_var"> _T".\test.mpf"`
	`</function>`

函数名称	含义

写文件	该函数可以将一个字符串变量的内容写入指定文件中。
	文件名只能用小写字母。
	需要访问另一个目录中的文件时，可采用相对路径，即目录从 appl 或 dvm 开始。
	
	参数：
	progname - 文件名
	str1 - 字符串
	
	语法：
	`<function name="doc.writetofile" > progname, str1 </function>`
	
	示例：
	`<let name = "my_var" type="string" > file content </let>`
	
	NC 文件系统
	`<function name="doc.writetofile">_T"n:\mpf\test.mpf", my_var </function>`
	
	CF 卡
	`<function name="doc.writetofile">_T"f:\appl\test.mpf", my_var </function>`
	
	或
	
	`<function name="doc.writetofile">_T".\test.mpf", my_var </function>`

函数名称	含义

删除文件	该函数可以从目录中删除指定的文件。
	文件名只能用小写字母。
	需要访问另一个目录中的文件时，可采用相对路径，即目录从 appl 或 dvm 开始。
	
	参数：
	progname - 文件名
	
	语法：
	`<function name="doc.remove" > progname </function>`
	
	示例：
	NC 文件系统
	`<function name="doc.remove">_T"n:\mpf\test.mpf"`
	`</function>`
	
	CF 卡
	`<function name="doc.remove">_T"f:\appl\test.mpf"`
	`</function>`
	
	或
	
	`<function name="doc.remove">_T".\test.mpf"`
	`</function>`

函数名称	含义

提取脚本文件	该函数将零件程序中包含的对话框说明复制到指定的局部变量中。
	
	指定用于保存主菜单名称的对话框名称和参数的程序名称用作调用参数。如果在零件程序中找到了对话框说明名称，则 Return 变量中会包含该说明。如果变量内容保存在文件中，则可以通过间接调用执行脚本文件。
	
	系统提供脚本文件，可从激活的零件程序中提取对话框说明并激活对话框。可通过 MMC 指令调用该脚本文件，从而激活属于零件程序的对话框。
	
	语法：
	`<function name="doc.loadscript" return="<name of script variable>">progname, _T"dialog part name", main menu </function>`
	
	属性：
	return - 保存有所提取的脚本的变量
	
	参数：
	progname - 程序中的完整路径（该路径名称可传输至 DOS 记录函数中。）
	main menu - 找到的菜单名称复制到该变量中
	dialog part name - 包含对话框说明的标签名称
	
	示例：
	`<function name="doc.loadscript" return="contents">prog_name, _T"main_dialog", entry</function>`

函数名称	含义
提取脚本文件 续	**示例程序：**

NC 程序

```
;<main_dialog entry="rpara_main">
; <let name="xpos" />
; <let name="ypos" />
; <let name="field_name" type="string" />
; <let name="num"  />


; <menu name="rpara_main">
;          <open_form name="rpara_form"/>
;          <softkey_back>
;                    <close_form />
;          </softkey_back>
;
;</menu>
;
;<form name="rpara_form">
;</form>
;
;</main_dialog>

G94 F100
MMC("CYCLES,PICTURE_ON,XMLDIAL_EMB.XML,
main","A")

;...
;...
;...
MMC("CYCLES,PICTURE_OFF,XMLDIAL_EMB.XML,
main","A")
G4 F2

M2
```

标准对话框

```
<DialogGui>

 <let name="menu_name" type="string">main</let>
 <let name="script_loaded">0</let>

 <menu name = "main">
  <if>
   <condition>script_loaded == 0</condition>
   <then>
    <function name ="load_current_program" />
    <dynamic_include src="__tmp.xml" />
        <op>
        script_loaded = 1;
        </op>
    <navigation>$$$menu_name</navigation>
   </then>
  </if>

  <softkey_back>
   <close_form />
  </softkey_back>
 </menu>

 <function_body name ="load_current_program" >
  <let name="prog_name" type= "string"/>
  <let name="contents" type= "string"/>
  <let name="entry" type= "string"/>
  <let name="len" />

  <op> prog_name = "nck/Channel/ProgramInfo/
workPandProgName"</op>

   <function name="DOC.LOADSCRIPT"
return="contents">prog_name,  _T"main_dialog",
entry</function>

   <function name="string.length"
return="len">contents</function>
   <if>
    <condition>len > 0</condition>
    <then>
        <op>
        menu_name = entry;
        </op>
     <function name="doc.writetofile" >
_T"F:\appl\__tmp.xml", contents</function>
    </then>
   </if>
  </function_body>
</DialogGui>
```

函数名称	含义

Exist	如文件存在，则函数提供值 1。
	文件名只能用小写字母。
	需要访问另一个目录中的文件时，可采用相对路径，即目录从 appl 或 dvm 开始。
	
	参数：
	progname - 文件名
	
	语法：
	`<function name="doc.exist" return="<int_var>" > progname </function>`
	
	示例：
	`<let name ="exist">0</let>`
	
	NC 文件系统
	`<function name="doc.exist" return="exist">_T"n:\mpf\test.mpf" </function>`
	
	CF 卡
	`<function name="doc.exist" return="exist">_T"f:\appl\test.mpf" </function>`
	
	或
	
	`<function name="doc.exist" return="exist">_T".\test.mpf" </function>`

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

函数名称	含义

设置位	该函数用于设置给定变量的各个位。
	可以置位或复位。
	
	语法：
	`<function name="ncfunc.bitset" refvar="address" value="set/reset" > bit0, bit1, … bit9 </function>`
	
	属性：
	refvar - 指定要写入的位组合所在变量的名称
	value – 位的取值为 0 和 1
	
	值：
	作为函数值，位号从 0 开始给定。
	每次调用最多可以修改 10 个位。
	
	示例：
	`<function name="ncfunc.bitset" refvar="nck/Channel/Parameter/R[1]" value="1" > 0, 2, 3, 7 </function>`
	
	`<function name="ncfunc.bitset" refvar="nck/Channel/Parameter/R[1]" value="0" > 1, 4 </function>`

删除控件	该函数可删除指定的控件。
	
	语法：
	`<function name="control.delete"> control name </function>`
	
	属性：
	name – 函数名称
	
	值：
	control name – 控件的名称
	
	示例：
	`<function name="control.delete"> _T"my_editfield" </function>`

函数名称	含义

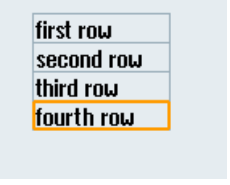
Add Item	该函数可在列表末尾添加一个新元素。
	提示：
	该函数只能用于控件类型“listbox”和“graphicbox”。
	
	语法：
	`<function name="control.additem"> control name, item </function>`
	
	属性：
	name – 函数名称
	
	值：
	control name – 控件名称
	item - 要添加的表达式
	itemdata - 整数值；由用户确定
	
	示例：
	`<let name ="itemdata">1</let>`
	…
	…
	…
	`<op> item_string = _T"text1" </op>`
	`<function name="control.additem">_T"listbox1", item_string, itemdata </function>`

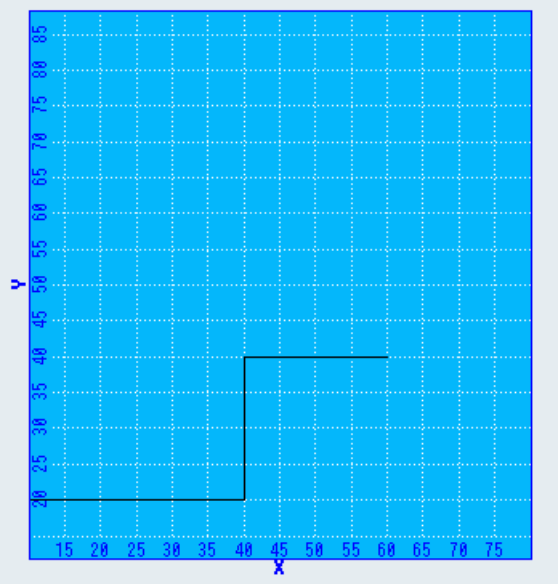
函数名称	含义

Insert Item	该函数可在指定位置添加一个新元素。
	提示：
	该函数只能用于控件类型“listbox”。
	
	语法：
	`<function name="control.insertitem"> control name, index, item, itemdata </function>`
	
	属性：
	name – 函数名称
	
	值：
	control name – 控件名称
	index – 位置从零开始
	item - 要添加的表达式
	itemdata - 整数值；由用户确定
	
	示例：
	`<let name ="itemdata">1</let>`
	…
	…
	…
	`<op> item_string = _T"text2" </op>`
	`<function name="control.insertitem">_T"listbox1", 1, item_string, itemdata </function>`

函数名称	含义
Delete Item	该函数可删除指定位置上的元素。 提示： 该函数只能用于控件类型“listbox”。 **语法：** <function name="control.deleteitem"> control name, index </function> **属性：** **name** - 函数名称 **值：** **control name** – 控件名称 **index**- 下标从 0 开始 **示例：** <function name="control.deleteitem">_T"listbox1", 1</function>

函数名称	含义

Load Item	该函数可向控件中添加一个表达式列表。
	该函数只能用于控件类型"listbox"和"graphicbox"。
	
	语法：
	`<function name="control.loaditem"> control name, list </function>`
	
	属性：
	name – 函数名称
	
	值：
	control name – 控件名称
	list 字符串变量
	在变量中包含的字符串必须符合下面描述的结构。
	
	列表结构：
	列表包含一定数量的表达式，它们之间用 \\n 分隔。
	
	示例：
	在本例中，内容通过函数 control.loaditem 分配给了列表框。
	控制符\\n 分开了可列为不同的行的表达式。
	`<CONTROL name = "list" xpos = "160" ypos = "32"`
	`width ="100" height="200" fieldtype="listbox"/>`
	`<let name="lb_list" type="string" />`
	`<op>`
	`lb_list = _T"first row\\n";`
	`lb_list = lb_list + _T"second row\\n";`
	`lb_list = lb_list + _T"third row\\n";`
	`lb_list = lb_list +_T"fourth row\\n";`
	`</op>`
	`<function name="control.loaditem">_T"list",`
	`lb_list</function>`
	 

函数名称	含义
Load Item 续	**示例：** 在本例中，内容通过函数 control.loaditem 分配给了图形框。 控制符\\n 分开了不同的图表元素。 `<control name= "c_gbox" xpos = "6" ypos="32" width="328" height="356" fieldtype="graphicbox" /> <let name="plot_list" type="string" /> <op> plot_list = _T"l;10;20;40;20\\n"; plot_list = plot_list + _T"l;40;20;40;40\\n"; plot_list = plot_list + _T"l;40;40;60;40\\n"; plot_list = plot_list +_T"l;80;0;80;100\\n"; </op> <function name="control.loaditem">_T"c_gbox", plot_list</function>` 

函数名称	含义

Empty	该函数可清除指定列表框或图形框的内容。
	
	语法：
	`<function name="control.empty"> control name,`
	`</function>`
	
	属性：
	name – 函数名称
	
	值：
	control name – 控件名称
	
	示例：
	`<function name="control.empty">_T"listbox1"`
	`</function>`

Get focus	该函数可提供当前正处在输入状态的控件的名称。
	
	语法：
	`<function name="control.getfocus" return="focus_name" />`
	
	属性：
	name – 函数名称
	return – 给定字符串变量，将控件的名称复制到其中。
	
	示例：
	`<let name>="focus_field" type="string"></let>`
	`<function name="control.getfocus" return="focus_field"/>`

函数名称	含义

Set focus	该函数可将指定的控件设置到输入状态。
	控件的名称将以函数的文本表达方式传输。
	
	语法：
	`<function name="control.setfocus"> control name`
	`</function>`
	
	属性：
	name - 函数名称
	
	值：
	control name - 控件名称
	
	示例：
	`<function name="control.setfocus" ">_T"listbox1"`
	`</function>`

Get cursor selection	该函数可提供列表框中光标所在的下标。
	控件的名称将以函数的文本表达方式传输。
	
	语法：
	`<function name="control.getcurssel" return="var">control name`
	`</function>`
	
	示例：
	`<let name>="index"></let>`
	`<function name="control.getcurssel" return="index">_T"listbox1"`
	`</function>`

函数名称	含义

Set cursor selection	该函数可在列表框中将光标放置在相应的行中。
	控件的名称将以函数的文本表达方式传输。
	
	语法：
	`<function name="control.setcurssel" > control name, index </function>`
	
	示例：
	`<let name>="index">2</let>`
	`<function name="control.setcurssel" ">_T"listbox1",index </function>`

Get Item	该函数可将列表框中所选择行的内容复制到指定的变量中。
	提供了字符串变量作为参考变量。
	控件的名称将以函数的文本表达方式传输。
	
	语法：
	`<function name="control.getitem" return="var"> control name, index </function>`
	
	示例：
	`<let name>="index">2</let>`
	`<let name>="item" type="string"></let>`
	
	`<function name="control.getitem" return="item" ">_T"listbox1",index </function>`

Get Item Data	该函数可将列表框中用户自定义的元素值复制到指定的变量中。
	在 EDIT 控件中，该函数可将用户自定义的值（item_data）复制到指定的变量中。
	提供了整型变量作为参考变量。
	控件的名称将以函数的文本表达方式传输。
	
	语法：
	`<function name="control.getitemdata" return="var"> control name, index </function>`
	
	示例：
	`<let name>="index">2</let>`
	`<let name>="itemdata"></let>`
	
	`<function name="control.getitemdata" return="itemdata" ">_T"listbox1",index</function>`

函数名称	含义
Image Box Set	该函数用于 **Imagebox** 和 **Graphicbox** 控件可见区域的控制。可以指定控件名称、控制指令和相应的值作为调用参数。 **语法：** `<function name="control.imageboxset">control name, command, command parameter</function>` **调用参数：** • **x_offs** 该函数将图像区移至指定的 X 位置。指定左边角度的坐标为其他参数。 • **y_offs** 该函数将图像区移至指定的 Y 位置。指定上边角度的坐标为其他参数。 • **xy_offs** 该函数将图像区移至指定的 X/Y 位置。指定左边和上边角度的坐标为其他参数。 • **followCursor** 图像区遵循设置的光标位置。 • **zoomplus** 图像以 0.12 的系数放大。 • **zoomminus** 图像以 0.12 的系数缩小。 • **autozoom** 图像根据显示区域自动缩放。 • **Update** 控件重新标记。 • **SetBkColor** 该指令用于设置指定的背景颜色。指定颜色代码为其他参数。 • **SetCursorRect** 指定为矩形的图像区移至可见区域。 • **SetAnimationState** （仅适用于 Imagebox 控件） – **start** - 指令启动动画。 – **stop** - 指令停止动画。

函数名称	含义
Image Box Set 续	**示例：** ... ... <softkey POSITION="1"> <caption>zoom%nin</caption> <function name="control.imageboxset">_T"c_gbox", _T"zoomplus"</function> </softkey> ... ... <form name = "main_form"> <init> <caption> graphicbox</caption> <control name= "c_gbox" xpos = "6" ypos="32" width="328" height="356" fieldtype="graphicbox" /> <let name="plot_list" type="string" /> <op> plot_list = _T"1;10;20;40;20\\n"; plot_list = plot_list + _T"1;40;20;40;40\\n"; plot_list = plot_list + _T"1;40;40;60;40\\n"; plot_list = plot_list +_T"1;80;0;80;100\\n"; </op> <function name="control.loaditem">_T"c_gbox", plot_list</function> </init> </form>
Image Box Get	该函数获取 Imagebox 控件属性。可以指定控制指令和相应的值作为调用参数。 **语法：** <function name="control.imageboxget">control name, command, command parameter</function> **调用参数：** **GetViewSize** 提供显示区尺寸 指定 SIZE 结构类型的变量作为其他参数。 **示例：** <let name="view_size" type="size" /> <function name="control.imageboxget">_T"topoview", _T"**GetViewSize**", view_size</function>

函数名称	含义

Bitmap Dim	该函数将位图尺寸重新复制到 SIZE 结构类型的变量中。将 struct_def.xml in 文件整合至项目中进行类型定义。
	更多信息参见章节“创建 struct_def.xml 文件 (页 155)”
	
	语法：
	`<function name="bitmap.dim" >name, variable type`
	`</function>`
	
	参数：
	name - 文件路径
	variable type - SIZE 类型变量的名称
	
	示例：
	`<let name="bmp_size" type="size" />`
	
	`<function name="bitmap.dim" >_T"test.bmp", bmp_size`
	`</function>`

Screen Resolution	该函数将系统的绝对屏幕分辨率重新复制到 SIZE 结构类型的变量中。将 struct_def.xml in 文件整合至项目中进行类型定义。 **更多信息**参见章节“创建 struct_def.xml 文件 (页 155)” **语法：** `<function name="hmi.screen_resolution" >varialble type </function>`
Get HMI Resolution	该函数将 SINUMERIK Operate 所用的屏幕分辨率重新复制到 SIZE 结构类型的变量中。将 struct_def.xml in 文件整合至项目中进行类型定义。 **更多信息**参见章节“创建 struct_def.xml 文件 (页 155)” **语法：** `<function name="hmi.get_hmi_resolution" >variable type </function>`

函数名称	含义
Get Caption Height	该函数提供标题栏高度，单位：像素。 **语法：** `<function name="hmi.get_caption_height" return="<return var>" />` **属性：** **return** - 整形变量

Get Font Families	该函数提供已安装字体的列表。字体名称通过换行符相互隔开。返回值为字符串变量的名称。
	
	语法：
	`<function name="HMI.GET_FONT_FAMILIES"`
	`return="< String-Variable>" />`
	
	示例：
	`<init>`
	`...`
	` <let name="families" type="string" />`
	` <function name="HMI.GET_FONT_FAMILIES" return="families" />`
	` <control name="lb" xpos="8" ypos="40" width="260" height="140"`
	`fieldtype="listbox">`
	` </control>`
	`<function name="control.loaditem">_T"lb", families</function>`
	`...`
	`<update_controls type="true" />`
	`</init>`

Abs	该函数提供给定数值的绝对值。
	
	语法：
	`<function name="abs" return="var"> value`
	`</function>`

SDEG	该函数可将给定值换算为度。
	
	语法：
	`<function name="sdeg" return="var"> value`
	`</function>`

函数名称	含义

SRAD	该函数可将给定值换算为弧度。
	
	语法：
	`<function name="srad" return="var"> value`
	`</function>`

SQRT	该函数可以计算给定值的根。
	
	语法：
	`<function name="sqrt" return="var"> value`
	`</function>`

ROUND	该函数可以将所传输的数值的小数位保留到指定的位数。如果未指定要保留的小数点后的位数，则该函数会保留到小数点后第一位。
	
	语法：
	`<function name="round" return="var"> value, nDecimalPlaces`
	`</function>`

FLOOR	该函数提供小于等于所传输值的最大整数值。
	
	语法：
	`<function name="floor" return="var"> value`
	`</function>`

CEIL	该函数提供大于等于所传输值的最小的整数值。
	
	语法：
	`<function name="ceil" return="var"> value`
	`</function>`

LOG	该函数可以计算给定值的对数。
	
	语法：
	`<function name="log" return="var"> value`
	`</function>`

LOG10	该函数可以计算给定值的十进制对数。
	
	语法：
	`<function name="log10" return="var"> value`
	`</function>`

函数名称	含义
POW	该函数计算“a^b”的值。 **语法：** `<function name="pow" return="var"> a, b </function>`
MIN	该函数会比较所传输的值并将较小的值返回。 **语法：** `<function name="min" return="var"> value1, value2 </function>`
MAX	该函数会比较所传输的值并将较大的值返回。 **语法：** `<function name="max" return="var"> value1, value2 </function>`
RANDOM	该函数提供伪随机数。 **语法：** `<function name="random" return="var" </function>`

函数名称	含义
MMC	**函数：** 通过 MMC 指令可以在 SINUMERIK Operate 上从零件程序中显示用户自定义对话框窗口。对话框窗口的式样由纯文本配置确定（制造商目录中的 XML 文件），系统软件保持不变。 Operate-Base 系统中发生变化后，参数如下： XML → CYCLES 或 POPUPDLG XML_ON → PICTURE_ON XML_OFF → PICTURE_OFF **语法：** MMC （“操作区、指令、XML 脚本文件、菜单名称、预留、预留、显示时间或者应答变量、预留”，“应答模式”） **含义：** • MMC 在 SINUMERIK Operate 上从零件程序中调用交互式对话框窗口。 • CYCLES 或 POPUPDLG 应从零件程序中启动的用户对话框的标识。 • PICTURE_ON 或 PICTURE_OFF 指令：对话框选择或对话框取消选择 • XML 文件 XML 脚本文件的名称 • 菜单 管理待显示对话框的菜单标签的名称 • 预留 预留用于 SINUMERIK Operate • 预留 预留用于 SINUMERIK Operate • 时间 应答模式“N”中的对话框显示时间 • 预留 预留用于 SINUMERIK Operate • 应答模式 “S” 同步，通过软键“OK”应答 “N” 异步，在规定的时间后关闭对话框

函数名称	含义

MMC 续	**同步调用示例：**
	Operate-Base 系统中发生变化后，参数如下：
	XML → CYCLES 或 POPUPDLG
	XML_ON → PICTURE_ON
	XML_OFF → PICTURE_OFF
	NC 指令
	`MMC("POPUPDLG or CYCLES",PICTURE_ON,mmc_cmd.xml,cmd1,,,,,","S")`
	文件： mmc_cmd.xml
	
	`<menu name = "cmd1">`
	`<open_form name = "cmd1_form" />`
	`<softkey_ok>`
	`<close_form />`
	`</softkey_ok>`
	`</menu>`
	
	`<form name = "cmd1_form" xpos ="12" ypos="100" width="500" height="240">`
	`<init>`
	`</init>`
	`<paint>`
	`</paint>`
	`</form>`
	
	异步调用示例（无应答）：
	Operate-Base 系统中发生变化后，参数如下：
	XML → CYCLES 或 POPUPDLG
	XML_ON → PICTURE_ON
	XML_OFF → PICTURE_OFF
	NC 指令
	`MMC("POPUPDLG or CYCLES",PICTURE_ON,mmc_cmd.xml,cmd1,,,10,,","N")`
	
	文件： mmc_cmd.xml `<menu name = "cmd1">`
	
	`<open_form name = "cmd1_form" />  <softkey_ok>  <close_form />`
	`</softkey_ok>`
	`</menu>`
	
	`<form name = "cmd1_form" xpos ="12" ypos="100" width="500" height="240">`
	`<init>`
	`</init>`
	`<paint>`

函数名称	含义
	</paint> </form>

函数名称	含义

MMC 续	**从零件程序中提取脚本文件示例**
	Operate-Base 系统中发生变化后，参数如下：
	XML → CYCLES 或 POPUPDLG
	XML_ON → PICTURE_ON
	XML_OFF → PICTURE_OFF
	NC 指令
	`MMC("POPUPDLG, PICTURE_ON, XMLDIAL_EMB.XML,main","S")`
	文件：xmldial_emb.xml
	
	`<DialogGui>`
	`<let name="menu_name" type="string">main</let>`
	`<let name="script_loaded">0</let>`
	`<menu name = "main">`
	`<if>`
	`<condition>script_loaded == 0</condition>`
	`<then>`
	`<function name ="load_current_program" />`
	`<dynamic_include src="__tmp.xml" />`
	`<op>`
	`script_loaded = 1;`
	`</op>`
	`<navigation>$$$menu_name</navigation>`
	`</then>`
	`</if>`
	`<softkey_back>`
	`<close_form />`
	`</softkey_back>`
	`</menu>`
	`<function_body name ="load_current_program" >`
	`<let name="prog_name" type= "string"/>`
	`<let name="contents" type= "string"/>`
	`<let name="entry" type= "string"/>`
	`<let name="len" />`
	`<op> prog_name = "nck/Channel/ProgramInfo/workPandProgName"</op>`
	`<function name="DOC.LOADSCRIPT"`
	`return="contents">prog_name, _T"main_dialog", entry`
	`</function>`
	`<function name="string.length" return="len">contents</function>`
	`<if>`
	`<condition>len > 0</condition>`
	`<then>`
	`<op>`
	`menu_name = entry;`
	`</op>`
	`<function name="doc.writetofile" >_T"__tmp.xml", contents`
	`</function>`
	`</then>`

函数名称	含义
	</if> </function_body> </DialogGui>

函数名称	含义
MMC 续	程序示例 ;<main_dialog entry="rpara_main"> ; <let name="xpos" /> ; <let name="ypos" /> ; <let name="field_name" type="string" /> ; <let name="num" /> ; <menu name="rpara_main"> ; <open_form name="rpara_form"/> ; <softkey_back> ; <close_form /> ; </softkey_back> ;</menu> ; ;<form name="rpara_form"> ; <init> ; <caption>test mask</caption> ; <let name="count" >0</let> ; <op> ; xpos = 120; ; ypos = 34; ; ; "nck/Channel/Parameter/R[10]" = 10; ; </op> ; <!-- load the number of controls --> ; <op> ; num = "nck/Channel/Parameter/R[10]"; ; </op> ; ; <while> ; <condition>count < num</condition> ; <print name="field_name" text="edit%d">count</print> ; ; <control name = "$field_name" xpos = "$xpos" ypos = "$ypos" refvar="nck/Channel/Parameter/R[$count]" hotlink="true" /> ; <op> ; ypos = ypos +24; ; count = count +1; ; </op> ; </while> ; ; </init> ; ; <paint> ; <op> ; xpos = 8; ; ypos = 36; ; count = 0; ; </op> ; <while>

函数名称	含义
	; <condition>count < num</condition> ; <print name="field_name" text="R-Parameter%d">count </print> ; ; <text xpos = "$xpos" ypos = "$ypos" >$$$field_name</text> ; <op> ; ypos = ypos +24; ; count = count +1; ; </op> ; </while> ; ; </paint> ; ;</form> ; ;</main_dialog> … G94 F100 MMC("POPUPDLG, PICTURE_ON, XMLDIAL_EMB.XML,main","A") … … … MMC("POPUPDLG, PICTURE_OFF, XMLDIAL_EMB.XML,main","A") G4 F2 M2
ISNUMERIC	该函数用于检查字符串内容是否可作为数字分析。空格和表格均被忽略。此外，符号和小数点均被视为有效字符。如果满足条件，该函数返回数值 1。 **参数：** **string** - 字符串 **语法：** <function name="isnumeric" return="result"><string> </function>
ROOT	该函数用于从数字中调用第 n 位根。 **参数：** **value** - 数字 **n** - 根说明 **语法：** <function name="ROOT" return="result"><value>, <n></function>

3.12 多点触控操作

3.12.1 多点触控功能

概述

随着多点触控显示屏的投入使用，屏幕操作也需要根据相应的功能扩展来加以调整。例如取消了位置固定的软键，由可自由定位的按钮取代，或作为对软键的补充。鉴于按钮设计方案的多样性，编程时仅使用面向操作软件外观设计的控件已无法再满足需要。比如：仅识别两个状态的切换控件可通过开关替代，该开关借助图标显示相应的状态并对轻击（Tap）或滑动（Flick）手势作出响应。

显示的图形能够对拖拽（Pan）手势作出响应。为此对 **imagebox** 控件进行相应扩展。

说明

以 Paint 标签输出的图形不会对手势作出响应。

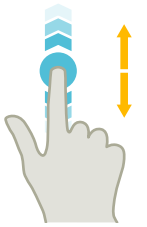
除了解析器所提供的控件外，还可在脚本中对手指手势进行处理，以便在对话框中提供新的导航策略。手指手势可用于放大/缩小对话框内容。

Easy XML 解析器支持下列手指手势：

手指手势

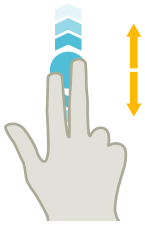
轻击 (Tap)

- 选择窗口
- 选择对象（例如 NC 程序段）
- 激活输入栏
  - 输入或覆盖值
  - 重新轻击以修改值



用 1 根手指垂直滑动 (Flick)

- 在列表中滚动（例如：程序、刀具、零点）
- 在文件中滚动（例如：NC 程序）



用 2 根手指垂直滑动 (Flick)

- 按页滚动列表（例如：NPV）
- 按页滚动文件（例如：NC 程序）

用 3 根手指垂直滑动 (Flick)

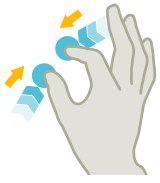
- 在列表开头或结尾滚动
- 在文件开头或结尾滚动

用 1 根手指水平滑动 (Flick)

- 在多栏列表中滚动

放大 (Spread)

- 放大图形内容（例如：模拟、模具制造视图）



缩小 (Pinch)

- 缩小图形内容（例如：模拟、模具制造视图）

用 1 根手指移动 (Pan)

- 移动图形内容（例如：模拟、模具制造视图）
- 移动列表内容

使用标签 **gesture_event** 和变量 **$gestureinfo** 来操纵手指手势。其实现针对经解码之手指手势的程序动作。

3.12.2 手指手势编程

标签 gesture_event

说明

仅当操作面板支持多点触控时，才执行此标签。

若操作软件识别出手指手势，则解析器会在变量 **$gestureinfo** 中提供手势信息，并执行标签 **gesture_event**。该变量具有数据类型 **GestureInfoStruct**，并包含以下属性：

属性	值	含义
type	3	移动手势
	4	缩小手势
	5	轻击手势
flag	1	缩放系数已改变
	2	旋转角已改变
state	1	已启动
	2	已更新
	3	已结束
item_data		生效控件的识别
	-1	手势未分配给控件
	!= -1	手势已在控件中执行，创建时已将该值分配给此控件。
point		手势的位置
	point.x	手势的 X 位置
	point.y	手势的 Y 位置
delta		手势的开始与当前事件之间的差
	<缩放差>	在缩放系数改变时
	<角度差>	在旋转角改变时

这些属性在文件 **struct_def.xml** 中预定义。

更多信息参见章节“创建 struct_def.xml 文件 (页 155)”

3.12.3 图形的手势控制

控制变量 imagebox

针对图形的手势控制，在控制变量 **Imagebox** 中有三个扩展：

属性	含义/特性
rotationangle	此属性值给出显示图形时采用的角度。
setrotationmode	属性值 **true** 允许对捏合（Pinch）手势的处理
setzoommode	属性值 **true** 允许在显示区域中放大/缩小以及移动图形。缺省状态下，此属性被设置为 **false**。

句法

```
<property rotationangle="角度" />
<property setrotationmode="true/false" />
<property setzoommode="true/false" />
```

示例：旋转位图

将位图顺时针旋转 30.5 度：

```
<let name="image_box_pict_name" type="string" >pic1.bmp</let>
…
…
…
 <control name="image_view" xpos="100" ypos="123" width="300"
height="200" fieldtype="imagebox" refvar="image_box_pict_name"
hotlink="true" >
 <property item_data="1000" />
 <property rotationangle="30.5" />
 <property setrotationmode ="true" />
```

```
</control>
```

示例：放大位图

允许位图的放大：

```
<control name="image_view" xpos="100" ypos="123" width="300"
height="200" fieldtype="imagebox" refvar="image_box_pict_name"
hotlink="true" >
 <property item_data="1000" />
 <property setzoommode="true" />
 </control>
```

控制功能 imageboxset

还可通过 **control.imageboxset** 功能设置特性。

可供使用的指令如下：

指令	含义/特性
SetRotationAngle	使图形旋转在 **lparam1** 中给定的角度。
SetRotationMode	**lparam1** 的值定义对捏合（Pinch）手势的与旋转有关的分析。 值等于 1 - 手势由控制项处理
SetZoomMode	若 **lparam1** 的值等于 1，则对于用于放大/缩小以及移动的捏合（Pinch）手势进行分析。

3.12.4 手势处理

标签 message

若缩放系数大于 1.0，则在显示区域中移动图片。在系数为 1.0 的情况下，例如可使用这个手指手势来在图片的列表中翻页。在此情形下，分析程序将一个可在标签 **message** 中分析的消息发送至表格。

消息参数 **$message_par1** 包含控制项的 Item_Data 值。消息参数 **$message_par2** 提示手势的运动方向。

$message_par2 可取下列值：

- -1，向左滚动
- 1，向右滚动
- -2，向上滚动
- 2，向下滚动

示例

```
…
…
<let name="image_box_pict_name" type="string" ></let>
<let name="pictindex" />
<let name="image_box_list" type="string" dim="4">
 "pic1.bmp",
 "pic2.bmp",
 "pic3.bmp",
 "pic4.bmp",
</let>

…
…
<control name="image_view" xpos="100" ypos="123" width="300"
height="200" fieldtype="imagebox" refvar="image_box_pict_name"
hotlink="true" >
 <property item_data="1000" />
 <property setzoommode="false" />
</control>
…
…
…
<!--
 -1，向左滚动
1，向右滚动
-2，向上滚动
2，向下滚动
```

```
 -->
<message>
 <if condition="$message_par1 == 1000">
  <then>

   <switch>
    <condition>$message_par2</condition>
    <case value="1">
     <op>
      pictindex = pictindex+1;
     </op>
     <if condition="pictindex > 3">
      <then>
       <op>
        pictindex = 0;
       </op>
      </then>
     </if>
    </case>
    <case value="-1">
     <op>
      pictindex = pictindex-1;
     </op>
     <if condition="pictindex < 0">
      <then>
       <op>
        pictindex = 3;
       </op>
      </then>
     </if>
    </case>
   </switch>
    <op>
     image_box_pict_name = image_box_list[$pictindex];
    </op>
   </then>
  </if>
 </message>
…
…
```

3.13 自定义按钮

按钮作为动作按钮或选择按钮嵌入表格。

3.13.1 下压按钮

标签 property

下压按钮是一个可用作按键或开关（带锁定功能的按键）的控制元件。可通过属性定义以“软键外观”风格以及以用户专用风格来设计按钮。相关的属性需要通过标签 **property** 定义。

可通过属性 **color_fg**、**color_bk**、**color_fg_pressed** 和 **color_bk_pressed** 修改前景色和背景色。

状态**按下/锁定**或者**松开**通过值一或零表达。为对按键的状态变化进行分析，需要指定处理程序功能。在 Control 标签中通过属性 **function** 指定处理程序功能。

可通过读取控制项变量或者分配的参考变量来确定开关的状态。

在触控操作中，当手指手势“松开”结束后，才提交手指手势“按下”和“松开”。

句法

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

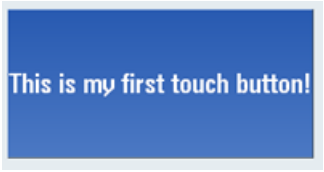
或者采用处理程序功能

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

或

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

 <property checkable="true" />
</control>
```

关闭



接通，锁定

示例

```
<control name="name" xpos="x position" ypos="y position"
fieldtype="pushbutton" refvar="button_state" hotlink="true"
color_fg="#ff00ff" color_bk="#000eee">
 <property checkable="true" />
  <caption></caption>
 <property picture="sk_circ_grind_cg.png" alignment="left"
TextAlignedToPicture="true" scaled="true"/>
</control>
```

3.13.2 按钮的功能

3.13.2.1 下压按钮的子标签

标签 caption

借助标签 **caption**，编程人员定义待为未按下状态显示的按钮文本。在缺省设置下，文本居中显示。可通过属性 **alignment** 修改文本的对齐。可给定下列属性值：

属性值	对齐
left	左对齐
right	右对齐
top	上
bottom	下
center	居中

若需为按下状态显示另一文本，则需要编写具有属性 **pressed** 和值 **true** 的第二 Caption 指令。

句法

居中显示

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

左对齐显示

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

按下显示

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

</control>
```

3.13.2.2 下压按钮的属性

通过标签 **property** 为控制项分配附加属性。

定义按键属性

借助属性 **checkable** 能够将按钮用作开关。为此将该属性的值设置为 **true**。

句法

```
<control name="name" xpos="x position" ypos="y position"
fieldtype="pushbutton" >
 <property checkable="true/false" />
</control>
```

禁用按键

属性 **disabled** 控制按钮的可操作性。若值为 **true**，则不对操作进行处理。

因分配的参考变量所引起的状态变化会导致按钮的更新。

句法

```
<property disabled="true/false" />
```

示例

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

分配图标

通过指定专有的图标或按键颜色，能够对预定义的设计进行修改。针对状态“未按下”、“按下”和“禁用”，可分别为按钮分配一个图标。若未针对状态“按下”或“禁用”进行分配，则会显示针对“未按下”状态的图标。

其他属性对各图标的对齐、缩放以及透明度加以控制。

属性	含义/特性
Picture	前景中的图标 针对“未按下”状态的图标的名称
PicturePressed	前景中的图标 针对“按下”状态的图标的名称

属性	含义/特性
PictureDisabled	前景中的图标 针对“禁用”状态的图标的名称
BackgroundPicture	背景中的图标 针对“未按下”状态的图标的名称
BackgroundPicturePressed	背景中的图标 针对“按下”状态的图标的名称
BackgroundPictureDisabled	背景中的图标 针对“禁用”状态的图标的名称

属性 alignment

在标签中可指定其他属性，其值基于列举的图标分配的 **alignment**。

属性值	对齐
left	左对齐
right	右对齐
top	上
bottom	下
center	居中
stretch	背景图标： 若使用值 **stretch**，分析程序会将图标缩放至按钮的矩形范围。 借助通过属性 **transparent** 指定透明色，能够产生按钮的任意轮廓。

3.13.2.3 下压按钮的控制项变量

通过为下面列出的属性变量赋予新的值，能够追加改变按钮的属性。在运算指令中通过给定控制项名称以及跟随于其后控制项变量名称来进行赋值。两个名称必须通过点相互分隔。

句法

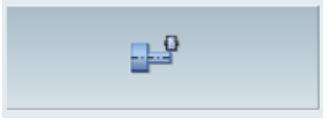
<控制项名称>.<控制项变量>

控制项变量	含义/特性
backgroundpicture	变量类型：字符串 背景图标的名称
backgroundpicturedisabled	变量类型：字符串 针对禁用状态的背景图标的名称
backgroundpicturepressed	变量类型：字符串 针对按下状态的背景图标的名称
picture	变量类型：字符串 前景图标的名称
picturedisabled	变量类型：字符串 针对禁用状态的前景图标的名称
picturepressed	变量类型：字符串 针对按下状态的前景图标的名称
picture_textalignedtopicture	变量类型：Bool 值： 1 = true 0 = false 在图标上将按钮文本对齐
picturedisabled_textalignedtopicture	变量类型：Bool 值： 1 = true 0 = false 在“禁用状态”图标上将按钮文本对齐

控制项变量	含义/特性
picturepressed_textalignedtopicture	变量类型：Bool 值： 1 = true 0 = false 在“按下状态”图标上将按钮文本对齐
backgroundpicture_keepaspectratio	变量类型：Bool 值： 1 = true 0 = false 保持或忽略背景图标的长宽比
backgroundpicturedisabled_keepaspectratio	变量类型：Bool 值： 1 = true 0 = false 保持或忽略“禁用状态”背景图标的长宽比
backgroundpicturepressed_keepaspectratio	变量类型：Bool 值： 1 = true 0 = false 保持或忽略“按下状态”背景图标的长宽比
picture_keepaspectratio	变量类型：Bool 值： 1 = true 0 = false 保持或忽略图标的长宽比
picturedisabled_keepaspectratio	变量类型：Bool 值： 1 = true 0 = false 保持或忽略“禁用状态”图标的长宽比

控制项变量	含义/特性
picturepressed_keepaspectratio	变量类型：Bool 值： 1 = true 0 = false 保持或忽略“按下状态”图标的长宽比
backgroundpicture_scaled	变量类型：Bool 值： 1 = true 0 = false 根据按钮的尺寸调整背景图标
backgroundpicturedisabled_scaled	变量类型：Bool 值： 1 = true 0 = false 根据按钮的尺寸调整“禁用状态”背景图标
backgroundpicturepressed_scaled	变量类型：Bool 值： 1 = true 0 = false 根据按钮的尺寸调整“按下状态”背景图标
picture_scaled	变量类型：Bool 值： 1 = true 0 = false 根据按钮的尺寸调整图标
picturedisabled_scaled	变量类型：Bool 值： 1 = true 0 = false 根据按钮的尺寸调整“禁用状态”图标
picturepressed_scaled	变量类型：Bool 根据按钮的尺寸调整“按下状态”图标

控制项变量	含义/特性
backpicture_alignment	变量类型：字符串 参见属性 alignment
backgroundpicturedisabled_alignment	
backgroundpicturepressed_alignment	
picture_alignment	
picturedisabled_alignment	
picturepressed_alignment	
caption	变量类型：字符串 “未按下状态”按钮文本
captionpressed	变量类型：字符串 “按下状态”按钮文本
caption_alignment	变量类型：字符串 参见属性 alignment
captionpressed_alignment	
captiondisabled_alignment	
disabled	变量类型：Bool 值： 1 = true - 下压按钮被禁用 0 = false - 下压按钮可操作



Picture

PicturePressed

PictureDisabled

BackgroundPicture + Picture

属性 TextAlignedToPicture

若此属性的值为 **true**，则基于图标将文本对齐。

```
<property picture="sk_circ_grind_cg.png" alignment="left"
TextAlignedToPicture="true" />
```

属性 scaled

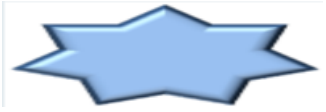
若此属性的值为 **true**，则根据表格中按钮的尺寸调整图片的尺寸，使得按钮的高度或宽度的最多 50 % 可供图形使用。

```
<property picture="sk_circ_grind_cg.png" alignment="left"
TextAlignedToPicture="true" scaled="true"/>
```

属性 stretch（仅对背景图标生效）

若此属性的值为 **true**，则根据按钮的尺寸调整图片的尺寸。

```
<property backgroundpicture="f:\appl\pbutton.png"
alignment="stretch" />
```



```
<property backgroundpicture="f:\appl\pbutton.png"
alignment="stretch" transparent="#ffffff"/>
```

```
<caption alignment="left" >This is my first touch button!</
caption>
 <property picture="sk_circ_grind_cg.png" alignment="left"
TextAlignedToPicture="true" scaled="true"/>
 <property backgroundpicture="pbutton.png" alignment="stretch"
transparent="#ffffff"/>
```

3.13.3 开关 on/off

开关控制项是一种图形元件，其借助一个图标指示两个状态中的一个。此控制项可以通过触控或鼠标来进行操作。

进入的状态可以存储在参考变量中，或者，可以在分配给该控制项的功能中确定状态切换。在 Control 标签中通过属性 **function** 进行分配。

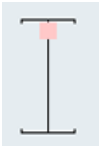
属性 **alignment** 实现按钮的垂直或水平对齐。

此控制项不具有缺省设计。若未为该控制项分配图标，则仅显示边界框和开关位置（通过点显示）。

句法

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

属性 alignment



开关垂直：值 **vr**

开关水平：值 **hr**

3.13.4 开关的功能

3.13.4.1 开关的属性

标签 property

可通过标签 **property** 为该控制项分配下列开关位置：

属性	含义/特性
left_position	在开关按钮向左运动的情况下，为控制项变量分配这个属性的值。
right_position	在开关按钮向右运动的情况下，为控制项变量分配这个属性的值。
upper_position	在开关按钮向上运动的情况下，为控制项变量分配这个属性的值。
lower_position	在开关按钮向下运动的情况下，为控制项变量分配这个属性的值。
disabled	此属性控制开关的可操作性。 若值为 **true**，则不对操作进行分析。 因分配的参考变量所引起的状态变化会导致开关状态的更新。

需要通过给定其他属性来定义最终的开关设计。这些属性必须与属性 **left_position**、**right_position**、**upper_position** 或 **lower_position** 一起定义。

借助通过属性 **transparent** 指定透明色，能够产生开关的任意轮廓。

属性	含义/特性
picture	前景中的图标，针对“未按下”状态的图标的名称
pictureDisabled	前景中的图标，针对“禁用”状态的图标的名称
transparent	该属性值包含透明色的颜色值。 此颜色值被用于所有与开关对应的图标。
caption	该属性值包含归属于开关位置的文本。此文本会显示在元件上，并受开关尺寸限制。

3.13.4.2 开关的控制项变量

通过为下面列出的控制项变量赋予新的值，能够追加改变开关的属性。在运算指令中通过给定控制项名称以及跟随于其后的控制项变量名称来进行赋值。两个名称必须通过点相互分隔。

句法

<控制项名称>.<控制项变量>

控制项变量	含义/特性
Left_position	变量类型：Int 在开关按钮向左运动的情况下，为控制项变量分配这个属性的值。
Right_position	变量类型：Int 在开关按钮向右运动的情况下，为控制项变量分配这个属性的值。
Lower_position	变量类型：Int 在开关按钮向下运动的情况下，为控制项变量分配这个属性的值。
Upper_position	变量类型：Int 在开关按钮向上运动的情况下，为控制项变量分配这个属性的值。
Picture_left_position	变量类型：字符串 针对左侧位置的图标的名称
Picture_right_position	变量类型：字符串 针对右侧位置的图标的名称
Picture_upper_position	变量类型：字符串 针对上部位置的图标的名称
Picture_lower_position	变量类型：字符串 针对下部位置的图标的名称
Picturedisabled_left_position	变量类型：字符串 针对左侧位置“禁用状态”的图标的名称
Picturedisabled_right_position	变量类型：针对右侧位置“禁用状态”的图标的名称

控制项变量	含义/特性
Picturedisabled_upper_position	变量类型：字符串 针对上部位置“禁用状态”的图标的名称
Picturedisabled_lower_position	变量类型：字符串 针对下部位置“禁用状态”的图标的名称
Caption_left_position	变量类型：字符串 左侧位置上的文本
Caption_right_position	变量类型：字符串 右侧位置上的文本
Caption_upper_position	变量类型：字符串 上部位置上的文本
Caption_lower_position	变量类型：字符串 下部位置上的文本
disabled	变量类型：Bool 值： 1 = true - 开关被禁用 0 = false - 开关可操作

水平显示

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

垂直显示

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

或者分配处理程序功能

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

示例

icon_left.bmp

icon_right.bmp

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

通过控制项变量分配属性

```
<control name="main_switch" xpos = "450" ypos="340" width="20"
height="46" fieldtype="switch" refvar="switch_statebf"
hotlink="true" alignment="vr">
 <property upper_position="1" />
 <property lower_position="0" />
</control>
…
…
…
<op>
  main_switch.transparent = #ffffff;
  main_switch.picture_upper_position = _T"switch_up.png";
  main_switch.picture_lower_position = _T"switch_bottom.png";
  main_switch.disable= 1;
</op>
```

3.13.5 单选按钮

借助单选按钮能够从多个选项中选择一个。必须为每个选项创建一个按钮。所有归属于表格、分组框或者滚动区域的单选按钮相互交互作用，故仅一个按钮被标记为选中。按钮状态被传输至控制项变量。

句法

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

```
 <caption>button text</caption>
</control>
```

3.13.6 复选框

借助复选框能够选择多个选项。必须为每个选项创建一个按钮。复选框的状态被传输至控制项变量。

句法

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

3.13.7 分组框

就外观而言，分组框将一组控制项包围，具有框架和标题。其用于将逻辑关联的控制项分组，这些控制项仅需要在该组内进行交互。嵌入的控制项的坐标以标题行下方的左上角为基准。

所有归属于组的控制项被作为子控制项嵌入 Control 标签。

在给定嵌入的控制项名称和值 **Item_Data** 时需要注意的是，就所有归属于表格的控制项而言，这些设定必须具有唯一性。

通过标签 **caption** 定义分组框的标题。

句法

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

3.13.8 滚动区域

滚动区域用于显示定义的区域内的控制项。嵌入的控制项的坐标以滚动区域的左上角为基准。若在可见区域外创建控制项，则实现可见区域的移动。所有归属于区域的控制项必须作为子控制项嵌入 Control 标签。

在给定嵌入的控制项名称和值 **Item_Data** 时需要注意的是，就所有归属于表格的控制项而言，这些设定必须具有唯一性。

句法

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

触控操作

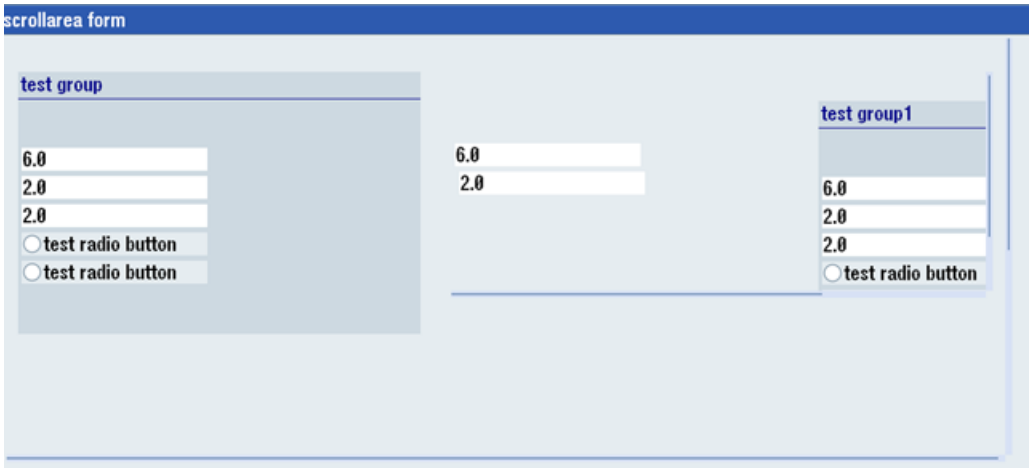
若将焦点置于当前不完全可见的控制项上，则使可见区域移动，直至能够完全显示该控制项。

轻击手势

在无法将此手势分配给嵌入的控制项的情况下，将此手势传输至脚本。

示例

嵌套的滚动区域



```
<let name="CB1" >1</let>
<let name="RB1" />

<form name="scrollarea_form">
 <init>
  <caption>scrollarea form</caption>
   <data_access type="true" />
    <control name= "c_scroll" xpos = "2" ypos="24" width="520"
height="300" fieldtype="scrollarea" >
    <control name= "c_group" xpos = "6" ypos="24" width="208"
height="186" fieldtype="groupbox" >
    <caption>test group</caption>
    <control name = "c18" xpos = "2" ypos = "54" refvar="nck/
Channel/Parameter/R[1]" hotlink="true" />
    <control name = "c19" xpos = "2" ypos = "74" refvar="nck/
Channel/Parameter/R[2]" hotlink="true" />
    <control name = "c20" xpos = "2" ypos = "94" refvar="nck/
Channel/Parameter/R[3]" hotlink="true" />
    <control name="radiobg" xpos = "2" ypos="114"
fieldtype="radiobutton" >
    <caption>test radio button</caption>
    <property backgroundpicture="f:\appl\cycle_my1.png" />
   </control>
   <control name="radiobg1" xpos = "2" ypos="134"
fieldtype="radiobutton" >
    <caption>test radio button</caption>
    <property backgroundpicture="f:\appl\cycle_my1.png" />
   </control>
  </control>
 <control name= "c_scroll_emb" xpos = "230" ypos="24" width="280"
height="160" fieldtype="scrollarea" >

 <control name = "c18emb" xpos = "2" ypos = "54" refvar="nck/
Channel/Parameter/R[1]" hotlink="true" />
 <control name = "c19emb" xpos = "4" ypos = "74" refvar="nck/
Channel/Parameter/R[2]" hotlink="true" />
 <control name = "c20emb" xpos = "3" ypos = "194" refvar="nck/
Channel/Parameter/R[3]" hotlink="true" />
 <control name= "c_group1" xpos = "190" ypos="24" width="208"
height="186" fieldtype="groupbox" >
 <caption>test group1</caption>
 <control name = "c18gemb" xpos = "2" ypos = "54" refvar="nck/
Channel/Parameter/R[1]" hotlink="true" />
 <control name = "c19gemb" xpos = "2" ypos = "74" refvar="nck/
Channel/Parameter/R[2]" hotlink="true" />
 <control name = "c20gemb" xpos = "2" ypos = "94" refvar="nck/
Channel/Parameter/R[3]" hotlink="true" />
 <control name="radiobggemb" xpos = "2" ypos="114"
fieldtype="radiobutton" >
  <caption>test radio button</caption>
  <property backgroundpicture="f:\appl\cycle_my1.png" />
</control>
```

```
<control name="radiobg1emb" xpos = "2" ypos="134"
fieldtype="radiobutton" >
 <caption>test radio button</caption>
  <property backgroundpicture="f:\appl\cycle_my1.png" />
 </control>
 </control>
 </control>
<control name = "c22" xpos = "2" ypos = "400" refvar="nck/Channel/
Parameter/R[12]" hotlink="true" />
 <control name="ChB1" xpos="208" ypos="430" width="50"
height="30" fieldtype="checkbox" refvar="PLC/M100.7" hotlink =
"true" color_bk="#00ffff">
  <caption>Check1</caption>
 </control>
  <control name="ChB2" xpos="208" ypos="460" width="58"
height="30" fieldtype="checkbox" refvar="PLC/M100.6" hotlink =
"true" color_bk="#00ffff">
  <caption>Check2</caption>
 </control>
  <control name="Radio1" xpos="208" ypos="490" width="40"
height="30" fieldtype="radiobutton" refvar="PLC/M100.0" hotlink =
"true" color_bk="#00ffff">
   <caption>Radio1</caption>
  </control>
   <control name="Radio2" xpos="208" ypos="520" width="45"
height="30" fieldtype="radiobutton" refvar="PLC/M100.1" hotlink =
"true" color_bk="#00ffff">
    <caption>Radio2</caption>
  </control>
   <control name="Radio3" xpos="208" ypos="550" width="50"
height="30" fieldtype="radiobutton" refvar="PLC/M100.2" hotlink =
"true" >
    <caption>Radio3</caption>
   </control>
    <control name="VChB1" xpos="8" ypos="430" width="50"
height="30" refvar="PLC/MB100" hotlink = "true"
color_bk="#00ffff"/>
    <control name="VChB2" xpos="8" ypos="465" width="53"
height="30" refvar="PLC/MB100" hotlink = "true"
color_bk="#00ffff"/>
   </control>
   <control name="ChB3" xpos="208" ypos="340" width="60"
height="30" fieldtype="checkbox" refvar="CB1" >
    <caption>Check3</caption>
   </control>
   </init>
  <timer>
  </timer>
</form>
```

3.14 Sidescreen 的使用

3.14.1 Sidescreen 中的 Easy XML

Easy XML 脚本语言同样可以用于在 Sidescreen 区域中创建对话框。对话框的显示由 Sidescreen 部件“页面”（**Page**）和“微件”（**Widget**）实现。对话框的集成由文件 **slsidescreen.ini** 实现。Sidescreen 部件会随着操作软件的启动、关闭而自动启动和关闭，也就是说，脚本解析器会随着 Sidescreen 部件的首次激活一次性加载对应的脚本。

Sidescreen 区域的尺寸由文件 **slsidescreen_<屏幕分辨率>.ini** 中的布局参数（layout）来确定。Easy XML 脚本不受该尺寸的影响，可以使用的宽度为 276 个像素，Easy XML 对话框可以使用的高度取决于使用的 Sidescreen 部件：页面型部件可以使用 768 个像素；微件型部件可以使用的高度由 Sidescreen 管理项决定，可以查看 **GETHMIResolution** 得知。

脚本解析器会查找一个可以激活某个“窗体”（form）或切换到另一个窗体的菜单。主菜单必须始终包含名称 **main**。Sidescreen 不支持软键标记，因此必须通过窗体的操作单元才能切换到另一个窗体。

脚本解析器不限制 Sidescreen 页面或 Sidescreen 微件的数量。

3.14.2 集成 Sidescreen 对话框

页面或微件的集成通过文件 **slsidescreen.ini** 实现。

页面应在段落 **[Sidescreen]** 中创建，每个页面的定义从关键字 **PAGE** 开始，后面跟一个连续编号的数字以及必要的一些属性。属性 **name** 确定页面的对象名称，属性 **implementation** 确定设计库和类名称，这两个属性要用逗号隔开。类型为 **SlSideScreenPage** 的页面可以集成 Sidescreen 元素，具体是通过 **ELEMENT** 条目实现的。创建一条元素行 **ELEMENTxxx** 需要遵循的规定和创建页面一样。如果希望页面上只包含由 Easy XML 设计的对话框，必须在属性 **implementation** 中输入 **slagmforms.SlEESideScreenPage** 。

微体可以添加到段落 **[Element_<name>]** 中的一个 Sidescreen 元素中。

创建一条微体行 **WIDGETxxx** 需要遵循的规定和创建页面一样。

需要激活由 Easy XML 设计的 Sidescreen 微体时，必须在属性 **implementation** 中输入 **slagmforms.SlEESideScreenWidget**。

语法

```
ELEMENT002= name:=<name>, implementation:=SlSideScreenElement

…
…

[Element_<name>]
WIDGET001= name:=<Widget Name>, implementation:=
slagmforms.SlEESideScreenWidget
```

默认设置中，微体的名称会用于组成主模块的名称。

示例

```
PAGE004= name:=sidescreen_proginfluence,
implementation:=slagmforms.SlEESideScreenPage
```

主模块的名称：sidescreen_proginfluence.xml

其他属性

在文件 **slsidescreen.ini** 的段落 **PAGE_<pagename>**、**ELEMENT_<elmentname>** 或 **WIDGET_<widgetname>** 中有更多属性用于页面或微体的设计。

以下属性可用于设计页面、元素或微体：

属性	含义/特性
TextId	文本标识符，和语言无关
TextFile	文本文件名称
TextContext	文本上下文 EASY_XML
Icon	图标名称

属性	含义/特性
maskPath	该属性确定要使用的主模块名称。
maskName	该属性确定要使用的主菜单名称。
focusable	该属性确定是否可以在该元素上放置输入焦点。

示例

```
[Page_sidescreen_proginfluence]
PROPERTY001= name:=focusable, type:=bool, value:="true"
Icon=sidescreen_proginfluence.png

[PAGE_sidescreen_proginfluence]
Icon= sidescreen_proginfluence.png
PROPERTY001= name:=focusable, type:=bool, value:="true"
PROPERTY002= name:=maskPath, type:=QString, value:="
sidescreen_proginfluence.xml"
PROPERTY003= name:=maskName, type:=QString, value:="main"
```

3.14.3 语言和文本管理

您可以为每个部件指定一份文本文件来管理和语言无关的文本。该文本文件的名称由部件名称、语言版本和后缀名".ts"构成。

文本上下文请使用 **EASY_XML**。

语法

<name>_<language>.ts

示例

```
PAGE004= name:=sidescreen_proginfluence,
implementation:=slagmforms.SlEESideScreenPage
```

文本文件名称：sidescreen_proginfluence_**eng.ts**

没有指定文本文件时，解析器会在标准文本文件 **oem_xml_screens_xxx.ts** 中查找文本标识符。

3.14.4 Sidescreen 部件

3.14.4.1 Sidescreen 元素

页面关联了标准设计时，便可以为该页面指定元素。为此要创建段落 **[Page_<page_name>]**，然后列出所有页面下的元素。

示例

```
[Sidescreen]
PAGE001= name:=<page_name>, implementation:=SlSideScreenPage

[Page_<page_name>]

ELEMENT<nummer>= name:=<Element name>,
implementation:=SlSideScreenElement
```

每个元素都需要用一个单独的段落 **[Element_<element name>]** 来定义，在该段落中列出属性以及对应的微体。

```
[Element_<element_name>]
WIDGET001= name:=<widget_name>, implementation:=
<Implementierung>
```

3.14.4.2 Sidescreen 微体

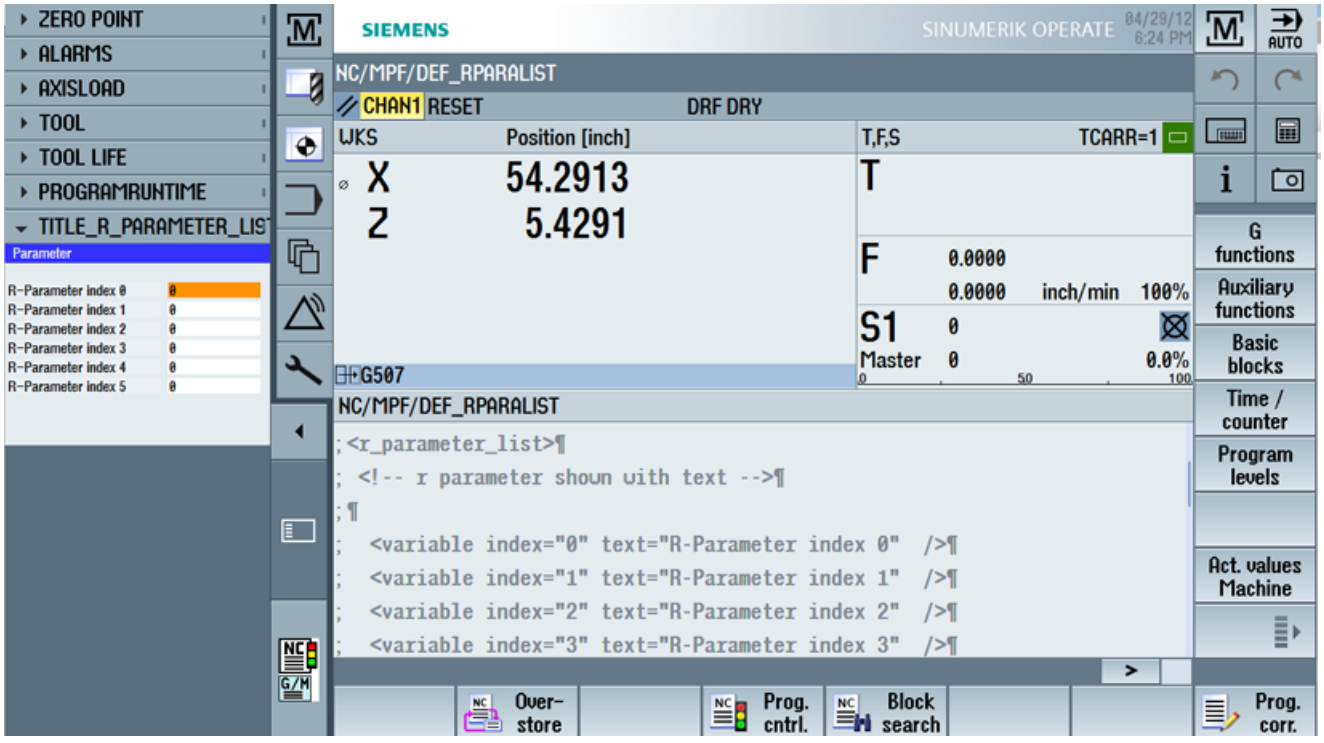
“微体”（Widget）是 Sidescreen 管理项指定的、用于显示设计的窗体的某个区域。默认设置中，微体没有输入焦点，因此无法编辑各字段。该情况可以通过属性 **focusable** 修改。随着微体的打开，解析器会一同打开属于主菜单的窗体。在该窗体内，可以通过导航指令转入其他菜单。

示例

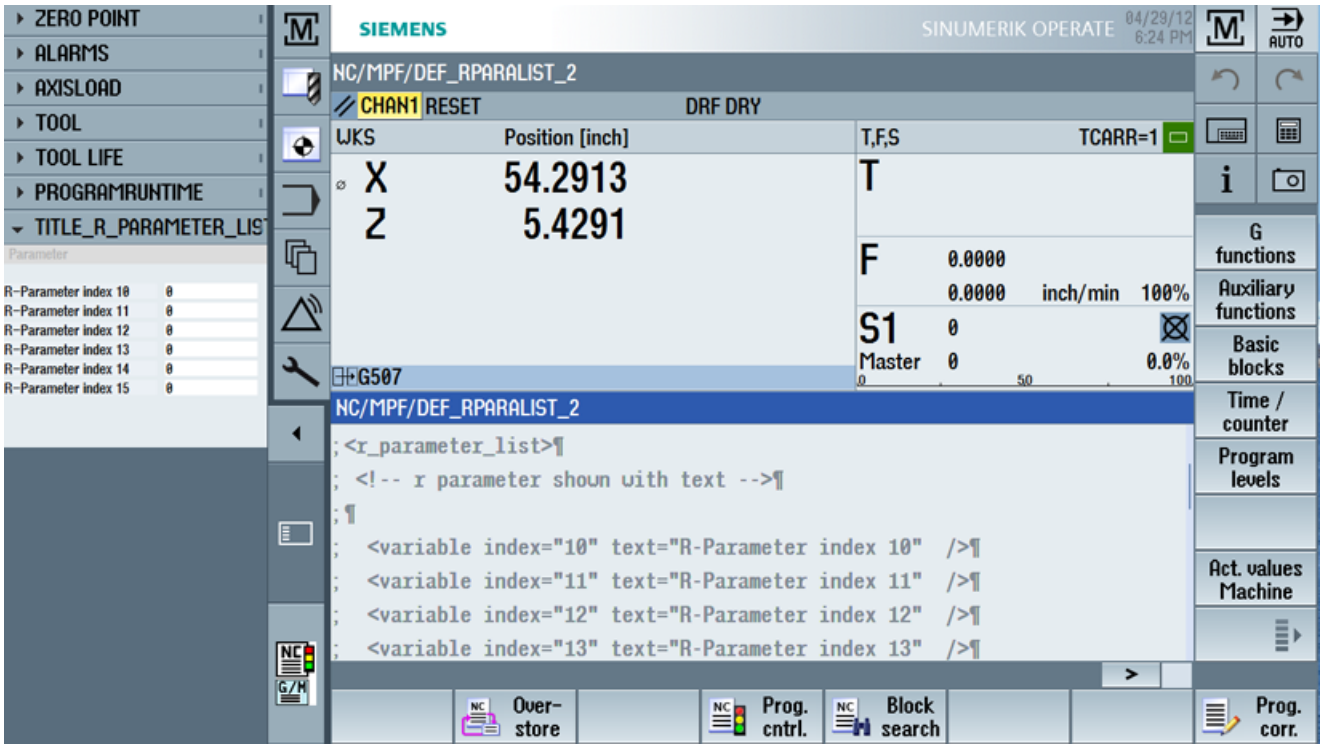
Easy XML 脚本在当前的零件程序内查找对于微体数据内容的说明。具体的表现形式有：脚本查找需要显示的、R 参数列表的说明。每个参数都可以有一条说明文本。

Toolbox 示例

Custom Screen Sample\
side_screen_examples\sidescreenwidget\displayRparameter\rparameter_widget.xml



选择了其他一个零件程序时，微体会自动更新。


ZERO POINT
ALARMS
AXISLOAD
TOOL
TOOL LIFE
PROGRAMRUNTIME
TITLE_R_PARAMETER_LIS
Parameter
R-Parameter index 10 0
R-Parameter index 11 0
R-Parameter index 12 0
R-Parameter index 13 0
R-Parameter index 14 0
R-Parameter index 15 0
SIEMENS
SINUMERIK OPERATE
04/29/12
6:24 PM
NC/MPF/DEF_RPARALIST_2
CHAN1 RESET
DRF DRY
WKS
Position [inch]
X
54.2913
Z
5.4291
T,F,S
TCARR=1
T
F
0.0000
0.0000
inch/min
100%
S1
0
Master
0
0.0%
G507
NC/MPF/DEF_RPARALIST_2
;<r_parameter_list>¶
; <!-- r parameter shoun uith text -->¶
;¶
; <variable index="10" text="R-Parameter index 10" />¶
; <variable index="11" text="R-Parameter index 11" />¶
; <variable index="12" text="R-Parameter index 12" />¶
; <variable index="13" text="R-Parameter index 13" />¶
AUTO
G functions
Auxiliary functions
Basic blocks
Time / counter
Program levels
Act. values Machine
Over-store
Prog. cntrl.
Block search
Prog. corr.

3.14.4.3 Sidescreen 页面

通过设计 **slagmforms.SIEESideScreenPage** 启动的“页面”（page）指供窗体使用的整个 Sidescreen 区域。它没有标题行，因此无法用指令 Caption 来设计标题。默认设置中，页面没有输入焦点，因此无法编辑各字段。该情况可以通过属性 **focusable** 修改。随着页面的打开，解析器会一同打开属于主菜单的窗体。在该窗体内，可以通过导航指令转入其他菜单。

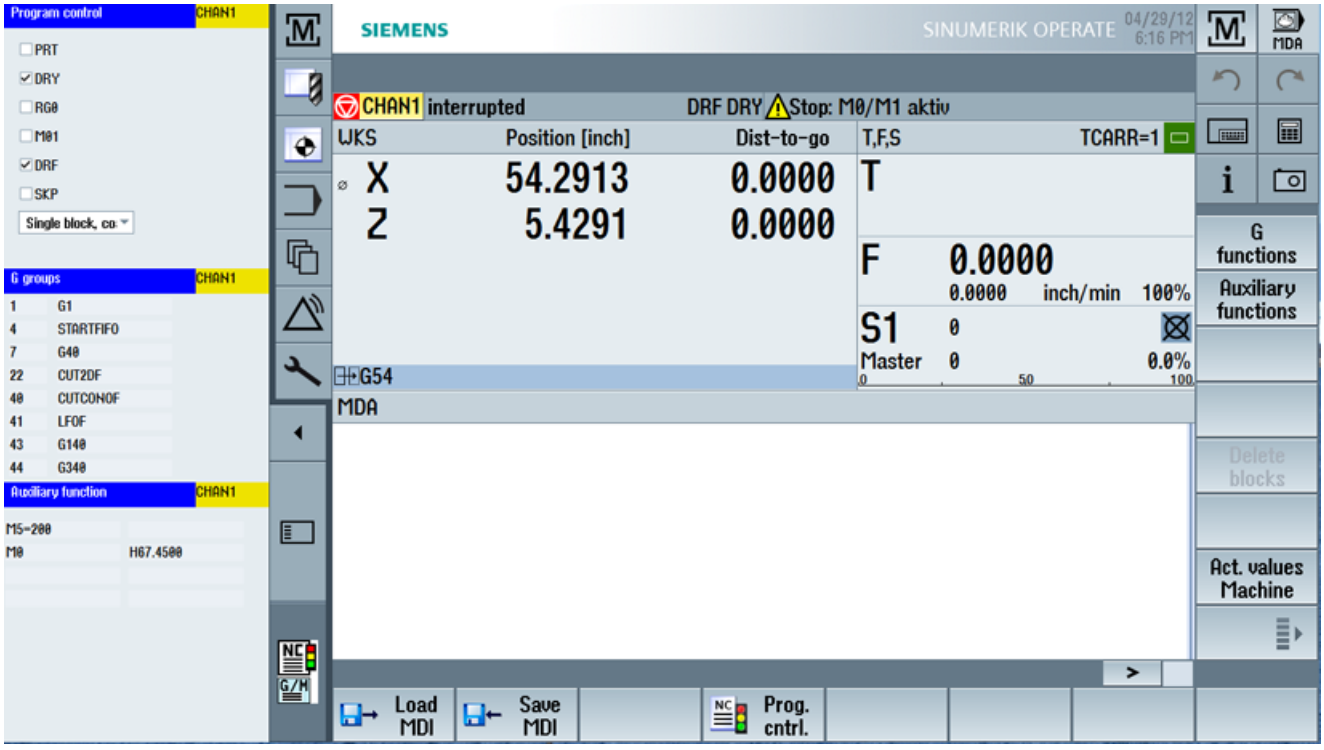
示例

Sidescreen 页面要设计为一幅只显示一个 Easy XML 窗体的页面。

本例中，Sidescreen 区域中显示的是附加信息。

Toolbox 示例

Custom Screen Sample\side_screen_examples\sidecreenpage
\sidescreen_proginfluence.xml




```
[Sidescreen]
PROPERTY001= name:=minimizable, type:=bool, value:="true"
PROPERTY002= name:=buttonBarVisible, type:=bool, value:="true"
```

```
PAGE001= name:=WidgetsPage, implementation:=SlSideScreenPage
PAGE004= name:=sidescreen_proginfluence,
implementation:=slagmforms.SlEESideScreenPage

[Page_sidescreen_proginfluence]
PROPERTY001= name:=focusable, type:=bool, value:="true"
Icon=sidescreen_proginfluence.png
```

文本文件：sidescreen_proginfluence_deu.ts

3.15 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	MACHINE	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" 无功能
HK5	ALARM	选择操作区“诊断”，对话框“报警列表”
HK6	CUSTOM	选择操作区“自定义”
HK7	MENU SELECT	选择“初始菜单”
HK8	MENU FUNCTION	选择“功能”操作区
HK9	MENU USER	选择“用户”操作区

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，该文件位于 user 级目录下的“/cfg”中和 oem 级目录下的“/cfg”中。

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 接口上已经为选择硬键预留了范围，该范围位于 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：

```
; 配置操作面板硬键 (KEY1-KEY9) 和
; PLC 硬键(KEY50-KEY254)
[keyconfiguration]
; “加工”键（硬键区）
KEY1.0 = area:=AreaMachine, dialog:=SlMachine
; “程序”键（硬键区）
KEY2.0 = area:=AreaProgramEdit
; “参数”键（硬键区）
KEY3.0 = area:=AreaParameter
; “程序管理”键（硬键区）
KEY4.0 = area:=AreaProgramManager
; “报警”键（硬键区）
KEY5.0 = area:=AreaDiagnosis, dialog:=SlDgDialog,
cmdline:="-slGfwHmiScreen SlDgAeAlarmsScreen"
; “自定义”键（硬键区）
KEY6.0 = area:=Custom
; “加工”键（可选，Shift + F10）
KEY7.0 = area:=AreaMachine, dialog:=SlMachine,
cmdline:="-MKey"
; 用户菜单（硬键区，Ctrl + F10）
KEY10.0 = area:=MenuUser
; 功能菜单（硬键区，Ctrl + Shift + F10）
KEY11.0 = area:=MenuFunction
KEY50.0 = area:=CustomXML, dialog:=SlEECustomDialog
```

各区域和对话框名称请查看“systemconfiguration.ini”文件，它位于 siemens 级目录下的“/cfg”中。

如果只指定了对话框名称“**SlEECustomDialog**”，系统的语法分析程序会在应用目录中查找文件“**xmldial.xml**”以及名为“**main**”的菜单标签。可以通过插入“**cmdline**”来表明其他的文件名称或“Main”之外的菜单名称。参数“**-mainModule**”指定待加载的 XML 说明。在该脚本中，系统的语法分析程序会查找“Main”菜单。参数“**-entry**”指定“Main”菜单的名

称。如果该参数为空，在功能启动时，语法分析程序会使用“**main**”。参数“**-conf**”必须始终保持值“**slagmdialog.hmi**”。

示例：

```
KEY50.0 = area:=CustomXML, dialog:=SlEECustomDialog,
cmdline:="-conf slagmdialog.hmi -mainModule restore.xml -entry
cmc2"
KEY51.0 = area:=CustomXML, dialog:=SlEECustomDialog,
cmdline:="-conf slagmdialog.hmi -mainModule activate.xml
-entry cmc1"
```

操作界面

“systemconfiguration.ini”文件中的以下条目可以使 Easy XML 实现上文描述的功能。

```
AREA0111= name:=CustomXML, dialog:=SlEECustomDialog,
panel:=SlHdStdHeaderPanel
DLG056= name:=SlEECustomDialog,
implementation:=slagmdialog.SlEECustomDialog,
process:=SlHmiHost1, preload:=false, terminate:=true,
cmdline:="-conf slagmdialog.hmi"
```

3.16 将用户窗口分配给预留软键

在所有的标准操作区域中都为扩展当前功能而预留了一个软键。为激活和连接带有用户项目的 OEM 软键，需要在以下系统目录中保存相应的文件：

/siemens/sinumerik/hmi/template/cfg

这些文件包含了特定操作区域的 XML 描述，这些描述在操作软件启动时会被编译并集成到操作区域的菜单树中。

为了集成自定义应用程序，需要将模板目录中与相应操作区域相关的文件复制到以下目录中并进行修改：

/oem/sinumerik/hmi/template/cfg

3.16.1 确定无需区分语种的软键文本

标签 TEXTFILE 包含了属于 OEM 区域的文本文件的文件名。文件名不需包括所使用语种的缩写和文件扩展名。

语法

```
<TEXTFILE>oemtextfile</TEXTFILE>
```

示例

oem_xml_screens_deu.ts

```
<TEXTFILE>oem_xml_screens</TEXTFILE>
```

在标签 **property** 中 **softkey** 中管理要显示的软键文本。OEM 值可用希望使用的文本 ID 来替代。属性 **translationContext** 指向文本文件的一个区域，文本分配在此区域中。在 easyXML 中，需要给定 EASY_XML 上下文。

语法

```
<PROPERTY name="textID" type="QString">OEM</PROPERTY>
<PROPERTY name="translationContext" type="QString">OEMContext</
PROPERTY>
```

示例

```
<PROPERTY name="textID" type="QString">MY_TEXT1</PROPERTY>
<PROPERTY name="translationContext" type="QString">EASY_XML</
PROPERTY>
```

3.16.2 Easy XML 区域分配

在标签 **softkey** 中可进一步确定导航目标。为此，需要在标签 **area** 中将导航目标 **CustomXML** 输入属性 **name** 中。需要删除属性 **dialog** 和 **screen**，因为这些参数是自动确定的。

语法

```
<NAVIGATION target="area">
<AREA name="OEMArea" dialog="OEMDialog" screen="OEMScreen"/>
</NAVIGATION>
```

示例

```
<NAVIGATION target="area">
<AREA name="CustomXML" />
</NAVIGATION>
```

如果使用导航目标 **area**，那么 Parser 会以文件 xmldial.xml 作为启动模块，以菜单 **main** 作为菜单管理的进入点。为了同时提供多个应用，可使用 **switchToDialog** 功能。为此，用 FUNCTION 标签取代 NAVIGATION 标签。**CustomXML** 区域、**SIEECustomDialog** 对话框以及主菜单和启动模块的名称都通过此功能来调用。

通过 **switchToArea** 标签退出 easyXML 区域，返回标准操作区域。

语法

```
<switchToArea name="<Area>"/>
...
<FUNCTION args="-area CustomXML -dialog SlEECustomDialog -
mainModule <Filename> -entry <Menuname>" name="switchToDialog"/>
```

项目示例

sldgarea_oem.xml

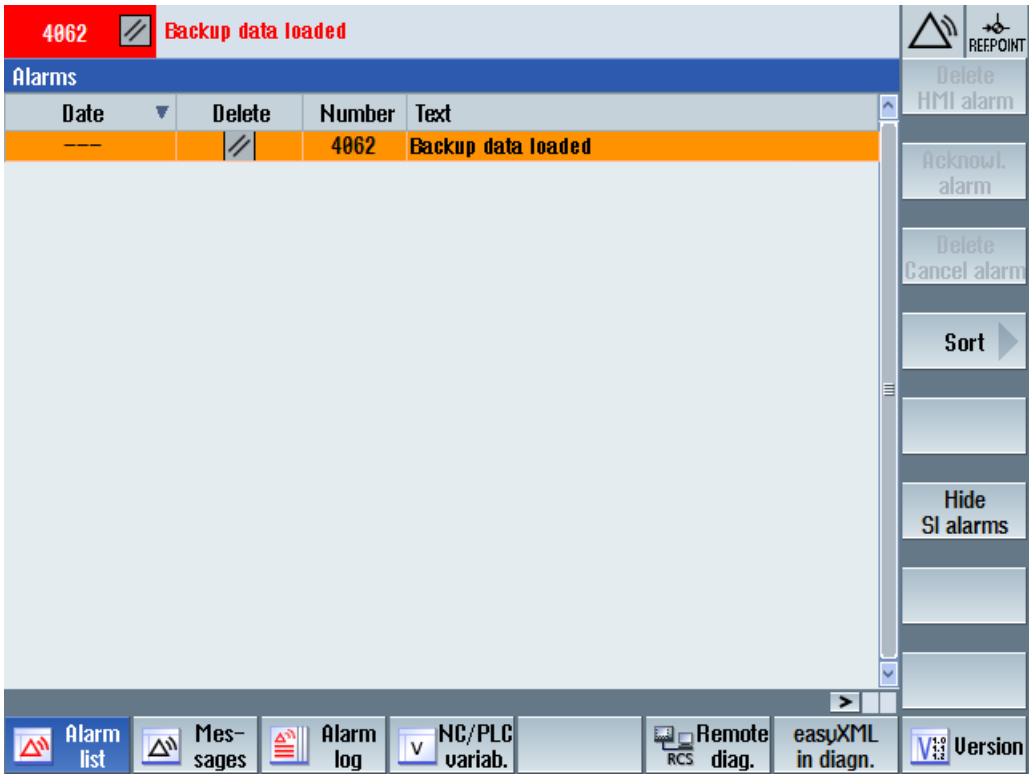
```
<!DOCTYPE DIALOG>
<DIALOG>
<TEXTFILE>oem_xml_screens</TEXTFILE>
<!-- ========================================================================
-->
<!-- OEM softkey number 7 on first horizontal main menu in area diagnosis --
>
<!-- ========================================================================
-->
<MENU name="DgGlobalHu">
 <ETCLEVEL id="0">
 <SOFTKEYGROUP name="GroupEtc">
 <SOFTKEY position="7">
  <PROPERTY name="textID" type="QString">DG_SK</PROPERTY>
  <PROPERTY name="translationContext" type="QString">EASY_XML</PROPERTY>
  <NAVIGATION target="area">
   <AREA name="CustomXML" />
  </NAVIGATION>
 </SOFTKEY>
</SOFTKEYGROUP>
</ETCLEVEL>
</MENU>
</DIALOG>
```

或

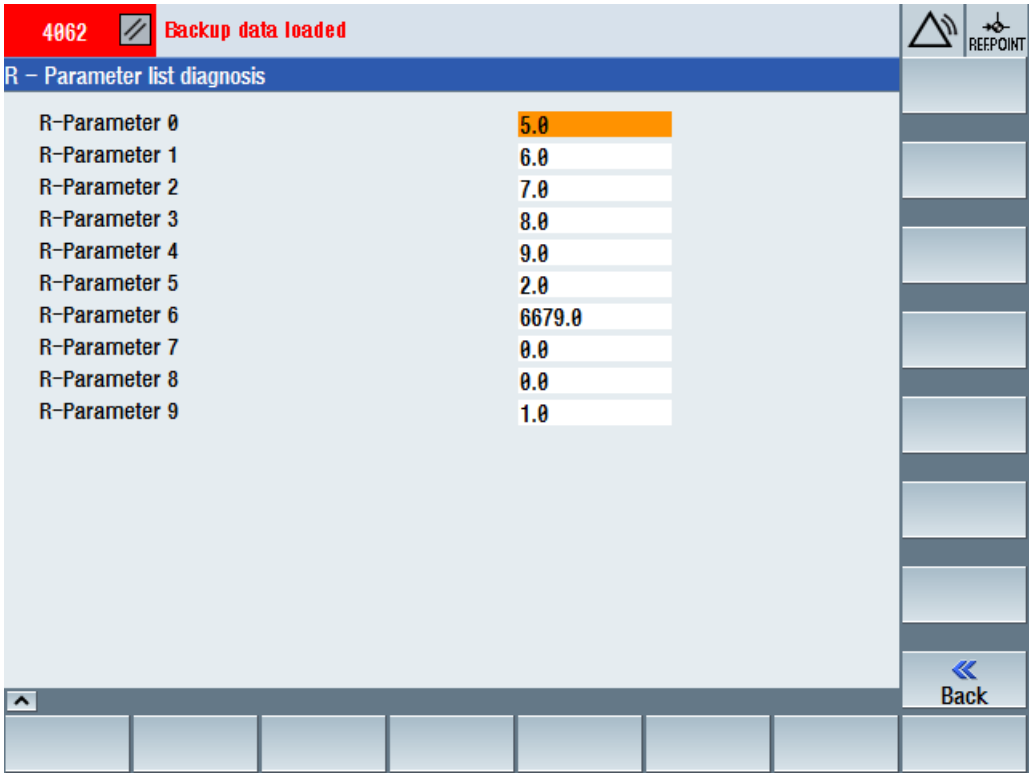
```
<!DOCTYPE DIALOG>
<DIALOG
<!-- ========================================================================
-->
<!-- OEM softkey number 7 on first horizontal main menu in area diagnosis --
>
<!-- ========================================================================
-->
<TEXTFILE>oem_xml_screens</TEXTFILE>
<MENU name="DgGlobalHu">
 <ETCLEVEL id="0">
 <SOFTKEYGROUP name="GroupEtc">
 <SOFTKEY position="7">
  <PROPERTY name="textID" type="QString">DG_SK</PROPERTY>
  <PROPERTY name="translationContext" type="QString">EASY_XML</PROPERTY>
  <FUNCTION args="-area CustomXML -dialog SlEECustomDialog -mainModule
dg.xml -entry main" name="switchToDialog"/>
 </SOFTKEY>
</SOFTKEYGROUP>
</ETCLEVEL>
</MENU>
</DIALOG>
```

dg.xml

```
<DialogGui>
<menu name = "main">
 <open_form name = "R_PARAMETER_LIST" />
 <softkey_back>
  <switchToArea name="AreaDiagnosis" dialog="SlDgDialog"/>
 </softkey_back>
</menu>
<!-- This form shows a R - parameter list -->
<form name = "R_PARAMETER_LIST">
 <init>
 <caption>R - Parameter list diagnosis</caption>
 <data_access type="true" />
  <control name = "c1" xpos = "322" ypos = "34" refvar="nck/Channel/
Parameter/R[0]" hotlink="true" />
  <control name = "c2" xpos = "322" ypos = "54" refvar="nck/Channel/
Parameter/R[1]" hotlink="true" />
  <control name = "c3" xpos = "322" ypos = "74" refvar="nck/Channel/
Parameter/R[2]" hotlink="true" />
  <control name = "c4" xpos = "322" ypos = "94" refvar="nck/Channel/
Parameter/R[3]" hotlink="true" />
  <control name = "c5" xpos = "322" ypos = "114" refvar="nck/Channel/
Parameter/R[4]" hotlink="true" />
  <control name = "c6" xpos = "322" ypos = "134" refvar="nck/Channel/
Parameter/R[5]" hotlink="true" />
  <control name = "c7" xpos = "322" ypos = "154" refvar="nck/Channel/
Parameter/R[6]" hotlink="true" />
  <control name = "c8" xpos = "322" ypos = "174" refvar="nck/Channel/
Parameter/R[7]" hotlink="true" />
  <control name = "c9" xpos = "322" ypos = "194" refvar="nck/Channel/
Parameter/R[8]" hotlink="true" />
  <control name = "c10" xpos = "322" ypos = "214" refvar="nck/Channel/
Parameter/R[9]" hotlink="true" />
 </init>
 <paint>
  <text xpos = "23" ypos = "34">R-Parameter 0</text>
  <text xpos = "23" ypos = "54">R-Parameter 1</text>
  <text xpos = "23" ypos = "74">R-Parameter 2</text>
  <text xpos = "23" ypos = "94">R-Parameter 3</text>
  <text xpos = "23" ypos = "114">R-Parameter 4</text>
  <text xpos = "23" ypos = "134">R-Parameter 5</text>
  <text xpos = "23" ypos = "154">R-Parameter 6</text>
  <text xpos = "23" ypos = "174">R-Parameter 7</text>
  <text xpos = "23" ypos = "194">R-Parameter 8</text>
  <text xpos = "23" ypos = "214">R-Parameter 9</text>
 </paint>
</form>
</DialogGui>
```


4062
Backup data loaded
REFPOINT
Alarms
Date
Delete
Number
Text

4062
Backup data loaded
Delete HMI alarm
Acknowl. alarm
Delete Cancel alarm
Sort
Hide SI alarms
Alarm list
Mes-sages
Alarm log
NC/PLC variab.
Remote diag.
easyXML in diagn.
Version


4062
Backup data loaded
REFPOINT
R – Parameter list diagnosis
R-Parameter 0 5.0
R-Parameter 1 6.0
R-Parameter 2 7.0
R-Parameter 3 8.0
R-Parameter 4 9.0
R-Parameter 5 2.0
R-Parameter 6 6679.0
R-Parameter 7 0.0
R-Parameter 8 0.0
R-Parameter 9 1.0
Back

3.16.3 标签描述

软键标签

除了 **softkey** 标签，软键状态也可在已使用 POSITION 属性时通过 **state** 属性进行设置。给出软键号作为属性值，以设定软键的状态。

示例

```
<menu name = "menu_sktest">

 <state type="$$$define_sk_type" POSITION="2"/>

<softkey POSITION="2" type="user_controlled" picture="$$
$bmp_name">
 <caption>Toggle%nSK</caption>
 <if>
 <condition>sk_type == 0 </condition>
 <then>
  <op> sk_type = 1 </op>
  <op> define_sk_type = _T"PRESSED" </op>
  <op>bmp_name = _T"f:\appl\red_led_on.bmp"</op>
  <state type="$$$define_sk_type" />
 </then>
 <else>
  <op> define_sk_type = _T"NOTPRESSED" </op>
  <op>bmp_name = _T"f:\appl\red_led_off.bmp"</op>
  <op> sk_type = 0 </op>
  <state type="$$$define_sk_type" />
 </else>
 </if>
  <print text="%s">sk_type_name</print>
  <navigation>menu_sktest</navigation>
</softkey>
...
...
```

typedef 标签

说明

示例中的元素预设值应当删除。

在类型定义中，使用 **element** 标签声明一个变量。标签的属性对应于 **let** 指令的属性。元素预设置可在创建变量时进行。

示例

RECT:

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

POINT:

```
<typedef name="StructPoint" type="struct" >
<element name="x" type="int" />
<element name="y" type="int" />
</typedef>
```

SIZE:

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

let 标签

用属性 **dim** 可给定数组元素的数量。对于二维数组，用逗号分隔第一维和第二维。对数组元素的存取通过数组下标进行，写在变量名称后的方括号内。字符串的维度可直接定义或通过一个变量的内容来定义。

示例

```
 <let name="d1" >3</let>

<let name="list_string" dim="3" type="string" />
```

或

```
<let name="list_string" dim="$d1" type="string" />
...
...
<op>
  list_string[2] = _T"test";
</op>
```

或

```
<let name="d1" >3</let>
<let name="d2" >6</let>

<let name="list_string3" dim="3, $d2" type="string" />
```

```
<op>
  list_string3[2, 5] = _T"test";
</op>
```

3.17 创建无需区分语种的文本

通过在对话框中使用文本标签，所有在对话框中使用的文本都会被替代为目前所选语种的语言，因而管理对话框中使用的文本时可不用考虑语种。此文本会和已提供的每个语种的文本标签一同保存在一个 UTF8 格式的文本文件中。

oem_xml_screens_<文件默认会作为文本文件归入 Easy XML 功能的语种缩写>.ts 中。给定 EASY_XML 标签作为文本上下文。

为辨识文本标签，在脚本中文本标签前会添加两个美元符号。

结构

oem_xml_screens_deu.ts

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

示例

```
<text xpos ="120" ypos="158">$$MY_TEXT1</text>
```

3.18 特定项目的文本文件

3.18.1 创建特定项目的文本文件

为分开管理各个项目，可为项目、窗体和菜单各自分别使用一个单独的文本文件。通过 Attribut 属性在相对应的标签中进行发布。

给出不含语言标签和文件扩展名的文本文件名作为属性值。在窗体或菜单关闭前，一直可以访问文本。如果加载带有 **DialogGui** 标签的文本文件，其内容在退出项目之前可以一直访问。如果多次使用了一个文本内容，则会在最后一次加载的文件中查找文本名称。每个文本文件可使用一个文本内容。

示例

main_form_screens_deu.ts

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

激活

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

或

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

或

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

对话框示例

main_form_screens_deu.ts

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

main2_sktext_screens_deu.ts

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE TS>
<TS>
<context>
    <name>EASY_XML</name>
    <message>
    <source>SK1</source>
    <translation>Softkey1</translation>
    </message>
</context>
</TS>
```

form2_screens_deu.ts

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE TS>
<TS>
<context>
    <name>EASY_XML</name>
    <message>
    <source>FORM2_TITLE</source>
    <translation>Form2 Title</translation>
    </message>
    <message>
    <source>FORM2_TEXT</source>
    <translation>Form2 text from text file</translation>
    </message>
</context>
</TS>
```

对话框脚本

xmldial.xml

```
<DialogGui textfile="main_form_screens">
<menu name = "main">
    <open_form name = "main_form" />
    <softkey POSITION="1">
    <caption>Menu 2</caption>
    <navigation>menu_2</navigation>
    </softkey>
</menu>
<menu name = "menu_2" textfile="main2_sktext_screens">
    <open_form name = "form2" />
    <softkey POSITION="1">
    <caption>$$SK1</caption>
    </softkey>
    <softkey_back>
    <navigation>main</navigation>
    </softkey_back>
</menu>
<form name="main_form" >
 <init>
    <data_access type = "true" />
    <caption>$$MAIN_FORM_TITLE</caption>
 </init>
 <paint>
    <text xpos="5" ypos="30">$$MAIN_FORM_TEXT</text>
 </paint>
</form>
<form name="form2" textfile="form2_screens">
 <init>
    <data_access type = "true" />
    <caption>$$FORM2_TITLE</caption>
 </init>
 <paint>
    <text xpos="5" ypos="30">$$FORM2_TEXT</text>
 </paint>
</form>
</DialogGui>
```

3.18.2 给定文本上下文

在一个文本文件内，可通过自行确定的区域名将文本进行分组。在标签 **context** 中，通过标签 **name** 确定区域标签。

语法

```
<context>
```

```
 <name>name</name>
</context>
```

示例

```
<context>
 <name>my context 1</name>
...
...
</context>
<context>
 <name>my context 2</name>
...
...
</context>
```

如要使用自定义的上下文，应通过 TEXTCONTEXT 属性为项目、窗体或菜单给定上下文名称以及文本文件名称。在设定新的上下文之前，在一个上下文之中的文本可以一直访问。也就是说，一个设置在项目上的上下文会替换为一个设置在窗体或菜单上的上下文。

项目示例

form2_screens_deu.ts

```
?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE TS>
<TS>
<context>
 <name>EASY_XML</name>
  <message>
   <source>FORM2_TITLE</source>
   <translation>Form2 Title</translation>
  </message>
  <message>
   <source>FORM2_TEXT</source>
   <translation>Form2 text from text file</translation>
  </message>
</context>
<context>
 <name>MY_FORM_CONTEXT</name>
  <message>
   <source>FORM3_TITLE</source>
   <translation>Title from the text context MY_FORM_CONTEXT</translation>
  </message>
  <message>
   <source>FORM3_TEXT</source>
   <translation>Form3 text from text file</translation>
  </message>
</context>
<context>
 <name>MY_SOFTKEY_CONTEXT</name>
  <message>
   <source>MENU3_SK1</source>
   <translation>SK%n1</translation>
  </message>
</context>
</TS>
```

using_textcontext.xml

```
...
<menu name = "menu3" textfile="form2_screens"
textcontext="MY_SOFTKEY_CONTEXT">
 <open_form name = "menu3_form" />
 <softkey POSITION="1">
  <caption>$$SK1</caption>
 </softkey>
 <softkey_back>
  <navigation>main</navigation>
 </softkey_back>
</menu>
<form name="menu3_form" textfile="form2_screens"
textcontext="MY_FORM_CONTEXT">
 <init>
  <data_access type = "true" />
  <caption>$$FORM3_TITLE</caption>
 </init>

 <paint>
  <text xpos="5" ypos="30">$$FORM3_TEXT</text>
 </paint>
</form>
...
...
```

3.18.3 给定文本文件列表

如要在一个项目中使用多个文本文件，可通过属性 **textfilelist** 将所需文本文件的名称以列表的形式提交给解析器。

文本文件名称的标记逐行定义，并通过换行符加以结束。文本文件的名称不要带有语言扩展名和文件扩展名。

示例

文件 `form2_screens_deu.ts` 应列在带 `form2_screens` 的列表中。

textfilelist.ini

```
form2_screens
...
...
```

激活

```
<DialogGui textfilelist="txtfilelist.ini">
...
...
</DialogGui>
```

3.19 恢复会话

关闭 Easy XML 项目时，所有本地变量都会被释放，脚本指令会被卸载。如果想要在关闭控制系统后仍保留变量数据，则需要预先将这些变量的属性设为 **permanent**。只需保留至控制系统关闭时的数据应将属性设为 **poweroff**。

如果再次选择 Easy XML 应用程序，则应在主菜单中设置重新选择应用程序后要激活哪个菜单或哪个窗体。编程人员必须确保所有必需数据均可用。

如果在标签 **DialogGUI** 中设置了属性 **restoresession**，则分析程序会重新选择上次打开的菜单和窗体。编程人员负责准备必需的数据。

创建调试对话框

4

4.1 功能概述

使用目的

使用"Easy Extend"能够非常简单地调试、激活、取消激活或测试附加设备。控制系统中有一张显示所有可用设备和设备状态的清单。系统可以最多管理 64 个设备。

设备的激活或取消由软键操作实现。

功能"Easy Extend"位于"参数"操作区域中。

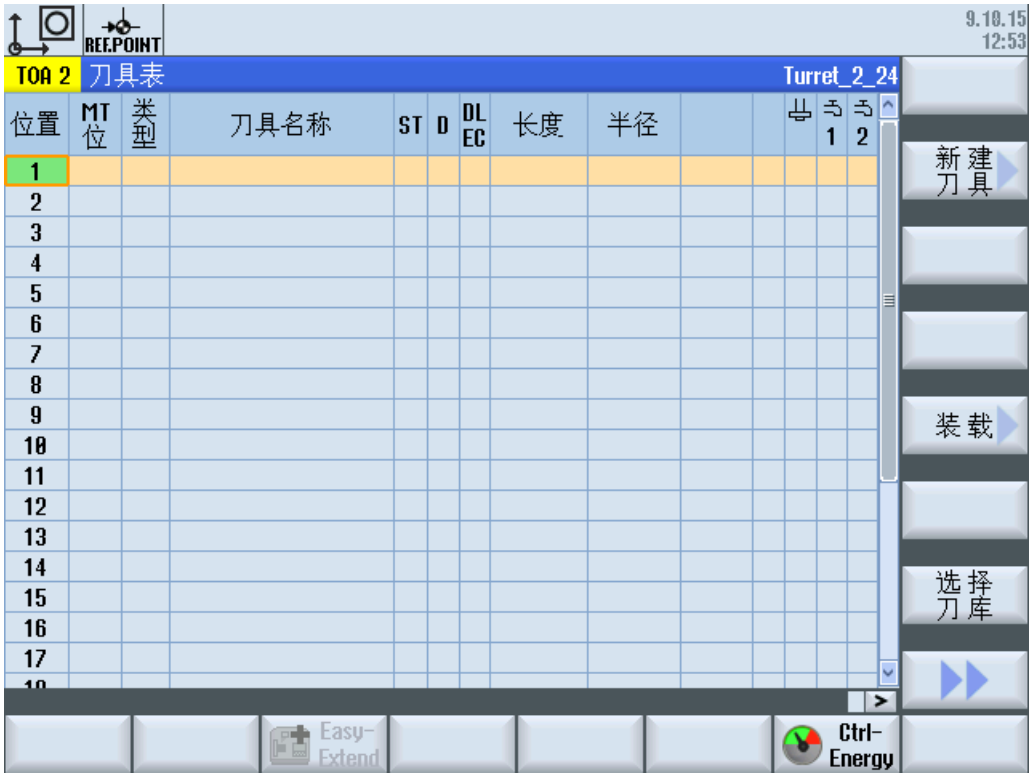


图 4-1 基本画面"参数"

配置

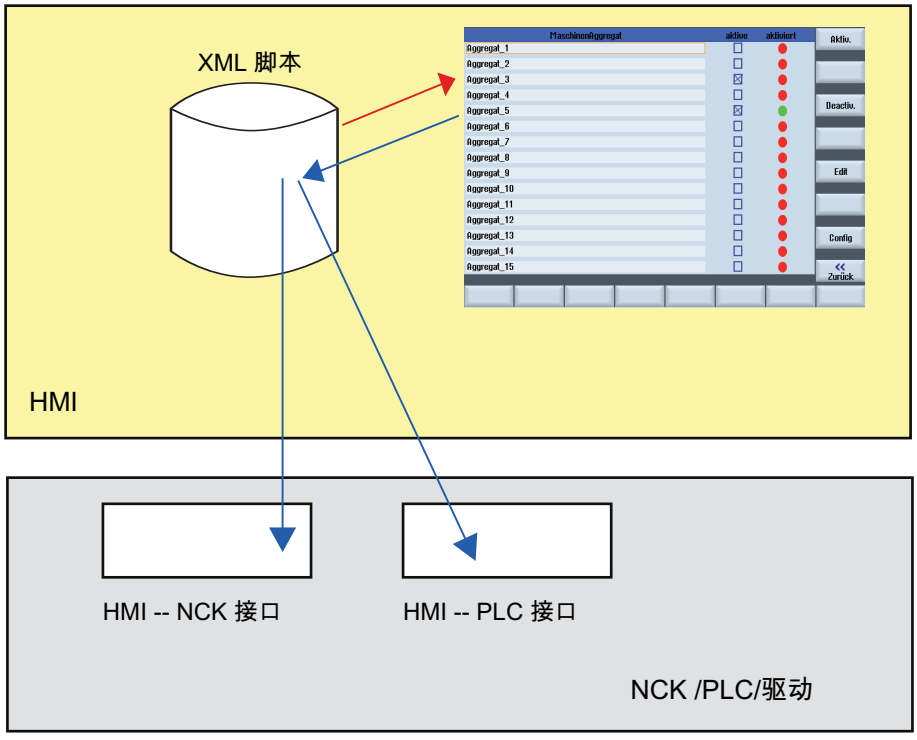


图 4-2 “Easy Extend”的功能

机床制造商首先定义以下功能，才可以使用“Easy Extend”功能：

- **PLC ↔ HMI 的接口**
  可选设备是通过操作界面和 PLC 之间的接口来管理的。
- **脚本编辑**
  机床制造商应在一个指令脚本中定义如何测安装、激活、取消和测试设备的流程。
- **参数窗口（可选）**
  参数窗口用于显示在脚本文件中定义的设备信息。

文件保存

和“Easy Extend”相关的文件保存在系统 CF 卡的目录“dvm”（机床制造商）中。

文件	名称	目标目录
文本文件	oem_aggregate_xxx.ts	\oem\sinumerik\hmi\lng
脚本文件	agm.xml	\oem\sinumerik\hmi\dvm

文件	名称	目标目录
存档文件	任意	\oem\sinumerik\hmi\dvm\archives
PLC 用户程序	任意	PLC

4.2 PLC 用户程序中的配置

载入配置

首先将制造商创建的配置和脚本文件、文本文件一起传送到控制系统的制造商目录中。另外，还需要载入对应的 PLC 用户程序。

设备的编程

操作组件和 PLC 之间是通过 PLC 用户程序中的数据块 DB9905 通讯的，该数据块中有 128 个字预留用于设备的管理。有关数据块标识符的说明请见章节“PLC_INTERFACE (页 308)”。

首先从设备 1 开始，指定 PLC 数据字：

数据块	设备名称
DB9905.DBB0	设备 1
DB9905.DBB4	设备 2
...	...
DB9905.DBB192	设备 49
DB9905.DBB196	设备 50

每个设备可以占用四个字节，含义如下：

字节	位	说明	
0	0	== 1	设备已经启动（HMI 反馈信息）
	1	== 1	请激活设备（HMI 请求）
	2	== 1	请取消激活设备（HMI 请求）
	3-7	预留	
1	0-7	预留	
2	0	== 1	设备已激活（PLC 反馈）
	1	== 1	设备出错
	2-7	预留	
3	0-7	设备的唯一标识	

文件保存

文件保存在系统 CF 的目录“oem”（机床制造商）和“oem_i”（个人）下。

说明

文件名只能由小写字母构成。

文件	名称	目标目录
文本文件	oem_aggregate_xxx .ts	/oem/sinumerik/hmi/lng/ /oem_i/sinumerik/hmi/lng/
脚本文件	agm.xml	/oem/sinumerik/hmi/dvm /oem_i/sinumerik/hmi/dvm
存档文件	任意	/oem/sinumerik/hmi/dvm/archives /oem_i/sinumerik/hmi/dvm/archives
PLC 用户程序	任意	PLC

一般操作顺序

机床制造商首先执行以下操作，提供所需数据：

1. 创建 PLC 用户程序，该程序中包含了由 PLC 控制的设备。
2. 调试“标准机床”，接着将数据备份到批量调试文档中。
3. 安装、调试设备，将数据作为差别批量调试文档读出。

说明

修改机床配置

需要修改驱动机床数据时，首先在控制系统中执行修改。然后在所有设置和装置上再次修改。

添加轴

在需要为机床增加机床数据时，请按照固定顺序添加驱动对象(DO)，因为批量调试文档包含了机床制造商参考机床的结构，一旦顺序被改动，便无法使用。

我们推荐为控制系统组件选择以下设置：

- NC 数据
- PLC 数据
- 驱动数据
  – ACX　格式（二进制）

4.3 操作界面上的显示画面

操作界面上的窗口

以下窗口用于“Easy Extend” 功能：

- 控制系统提供了一个**自定义窗口**，其中可显示可用设备。
- 如果还没有开展首次调试，系统会打开**调试窗口**。

如果已经为设备编写了一个调试流程（XML 指令：“START_UP”），而设备还没有经过调试，系统会启动调试流程。

此时，在读取脚本文件中保存的批量调试文档前，会完整备份数据。

出错时调试人员可以决定，是否取消调试，或手动排除可能出现在机床配置中的错误。

- 按下“取消”，可以中断调试。系统随后会将取消调试数据的备份。

调试成功结束后需要断开机床时，可以编写 XML 指令“POWER_OFF”，控制系统上便会输出对应的信息。

4.4 创建和语言相关的文本

文本文件结构

包含和语言相关的文本的 xml 文件应以 UTF8 格式创建：

示例

oem_aggregate_eng.ts

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE TS>
<TS>
<context>
    <name>EASY_EXTEND</name>
    <message>
    <source>DEVICE_ONE</source>
    <translation>Device one</translation>
    </message>
    <message>
    <source>DEVICE_TWO</source>
    <translation>Device two</translation>
    </message>
</context>
</TS>
```

oem_aggregate_deu.ts

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE TS>
<TS>
<context>
    <name>EASY_EXTEND</name>
    <message>
    <source>DEVICE_ONE</source>
    <translation>Erstes Gerät</translation>
    </message>
    <source>DEVICE_TWO</source>
    <translation>Zweites Gerät</translation>
    </message>
</context>
</TS>
```

4.5 功率部件应用示例

激活驱动对象

需要激活的驱动对象已经过调试，并由机床制造商重新取消激活，用于将轴作为选件投入市场。

必须执行以下步骤激活轴：

- 通过 p0105 激活驱动对象。
- 在通道机床数据中激活第 2 根轴。
- 通过 p0971 备份驱动机床数据。
- 等待数据写入。
- 重启 NCK 和驱动。

编程：

```
<DEVICE>
  <list_id>1</list_id>
  <name> "激活驱动"</name>

  <SET_ACTIVE>
    <data name = "drive/dc/p105[DO5]">1</data>
    <data name = "$MC_AXCONF_MACHAX_USED[4]">5</data>
    <data name = "drive/dc/p971[DO5]">1</data>
    <while>
      <condition> "drive/dc/p971[DO5]" !=0 </condition>
    </while>
    <control_reset resetnc ="true" resetdrive = "true"/>
  </SET_ACTIVE>

  <SET_INACTIVE>
    <data name = "drive/dc/p105[DO5]">0</data>
    <data name = "$MC_AXCONF_MACHAX_USED[4]">0</data>
    <data name = "drive/dc/p971[DO5]">1</data>
    <while>
      <condition> "drive/dc/p971[DO5]" !=0 </condition>
    </while>
    <control_reset resetnc ="true" resetdrive = "true"/>
  </SET_INACTIVE>

</DEVICE>
```

激活由 PLC 控制的设备

设备通过输出字节 10 应答，并且通过输出字节 9 向 PLC 发送运行就绪反馈信息。

输出字节被设置为固定的编码用于激活设备。接着，While 循环等待设备运行准备就绪。

编程：

```
<SET_ACTIVE>
  <DATA name = "plc/qb10"> 8 </DATA>
  <while>
    <condition> "plc/ib9" !=1 </condition>
  </while>
</SET_ACTIVE>
```

4.6 脚本语言

说明

在功能 创建用户对话框 (页 25) 中描述的所有脚本元素形成了“Easy Extend”功能的基础。为管理附加装置，还定义了其它的脚本元素。

脚本的程序段落

脚本由以下段落组成：

- “Easy Extend” - 标签
- 用于确定设备可执行操作的框架
- 用于设备的标签
- 用于设备调试的标签
- 用于设备激活的标签
- 用于设备取消的标签
- 用于设备测试的标签
- 参数窗口的标签

下面的章节详细说明各个标签。

说明

标签 <tag>	含义
AGM	“调试向导”功能的标签
DEVICE	用于描述设备的标签。 **属性：** • **option_bit** 设备指定有一个固定的位编号，用于管理选件。
NAME	该标签定义在窗口中显示的设备名称。 如果使用参考文本，窗口中会显示该标签下定义的文本。
START_UP	该标签包含调试设备所需流程的说明。

标签 <tag>	含义
SET_ACTIVE	该标签包含激活设备所需流程的说明。 **属性：** • **timeout** 该属性指定超时时间，单位秒。如果脚本在该时间后仍未结束，系统会中断处理。
SET_INACTIVE	该标签包含取消设备所需流程的说明。 **属性：** • **timeout** 该属性指定超时时间，单位秒。如果脚本在该时间后仍未结束，系统会中断处理。
TEST	该标签包含设备测试所需的指令。 **属性：** • **timeout** 该属性指定超时时间，单位秒。如果脚本在该时间后仍未结束，系统会中断处理。
UID	用于区分 PLC ↔ HMI 接口中设备的唯一数字标识码。
VERSION	用于版本的标签

应答功能执行（negative）

通过自动提供的变量"$actionresult"系统可以向 XML-Parser 传递一个"negative"的执行结果。该值为零时，Parser 会中断执行功能。

示例

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE AGM>
<AGM>                                  “调试向导”的标签
<DEVICE>
  <NAME> 设备 1 </NAME>                用于设备的标签
  <START_UP>                           用于设备调试的标签
  …
  </START_UP>
  <SET_ACTIVE>                         用于设备激活的标签
  …
  </SET_ACTIVE>
```

```
    <SET_INACTIVE>                    用于设备取消的标签
    …
    </SET_INACTIVE>
    <TEST>                            用于设备测试的标签
    …
    </TEST>
  </DEVICE>
  …
  </AGM>
```

4.6.1 CONTROL_RESET

说明

该标签可以重新启动一个或多个控制系统组件。 只有当控制系统恢复周期运行时，才可以继续执行脚本。

编程

标签：	**CONTROL_RESET**	
句法：	<CONTROL_RESET resetnc="TRUE" />	
属性：	resetnc="true"	重新启动 NC 组件。
	resetdrive="true"	重新启动驱动组件。

4.6.2 FILE

说明

该标签可以读入或创建标准文档。

- 读入文档：
  请输入文档名称。
- 创建文档：
  定义了属性 create= "true" 后，该功能会以指定名称创建一个标准文档(*.arc)，并保存在目录 .../dvm/archives 中。

编程

标签：	**FILE**	
句法：	<file name ="<文档名称>" />	
	<file name =“<文档名称>” create=“true” class=“<数据类别>”	
	group=“<范围>” />	
属性：	name	用于文件名称的标签

create	以指定的名称在目录 …/dvm/archives/ 下创建一个调试文档。
group	指定文档包含的数据组。 如果需要备份多个数据组，请用空格隔开。 文档可以包含以下数据组： • NC • PLC • HMI • DRIVES

示例

```
<!-- 创建数据级文档 -->
<file name="user.arc" create="true"
group="nc plc hmi" />

<!–将文档读入控制系统->
<file name="user.arc" />
```

4.6.3 OPTION_MD

说明

该标签允许重新定义选件机床数据。 在出厂时，系统使用的是 MD14510 $MN_USER_DATA_INT[0] ~ $MN_USER_DATA_INT[3]。

如果由 PLC 用户程序管理选件，请在一个数据块或 GUD 中提供对应的数据字。

数据按位管理。 从位 0 开始，各个位按照设备清单上的顺序固定指定给各个设备，即：位 0 指定给设备 1；位 1 指定给设备 2，如此类推。如果需要管理多于 16 个设备，请通过区域索引将地址标识指定给设备组 1-3。

说明

换算取值范围

MD14510 $MN_USER_DATA_INT[i] 的取值范围是 -32768 ~ +32767。为了在机床数据窗口中按位使能各个设备，需要将位组换算成十进制格式。

编程

标签：	**OPTION_MD**	
句法：	区域 0：	
	<option_md name = "数据的地址标识" />	
	或者是	
	<option_md name = "数据的地址标识" index= "0"/>	
	区域 1~3：	
	<option_md name = "数据的地址标识" index= "区域索引"/>	
属性：	name	地址标识，例如：$MN_USER_DATA_INT[0]
	index	用于区域索引的标签：
		0 （缺省设置）： 设备 1 ~ 16
		1: 设备 17 ~ 32
		2: 设备 33 ~ 48
		3: 设备 49 ~ 64

4.6.4 PLC_INTERFACE

说明

该标签允许重新定义 PLC ↔ HMI 接口。系统提供 128 个可编址的数值。

预设置：DB9905

编程

标签：	**PLC_INTERFACE**	
句法：	<plc_interface name = "数据的地址标识" />	
属性：	name	地址标识，例如“plc/mb170”

示例：`plc/mb170`

4.6.5 POWER_OFF

说明

该标签用于设定一条要求操作员关闭机床的信息。 显示文本固定保存在系统中。

编程

标签：	**POWER_OFF**
句法：	<power_off />
属性：	--

4.6.6 WAITING

说明

该标签用于指定：在 NC 或驱动复位后，等待各个组件重新启动。

编程

标签：	**WAITING**	
句法：	<WAITING WAITINGFORNC ="TRUE" />	
属性：	waitingfornc="true"	等待 NC 重新启动。
	waitingfordrive="true"	等待驱动重新启动。

4.6.7 用于窗口的 XML 标签

用于设置参数的窗口

可以为每个设备设计一个窗口，用于设置或输出运行时间所需的附加参数。按下软键“附加参数”后，会显示该窗口。

在创建对话框时可以使用章节 创建用户对话框 (页 25) 中描述的所有脚本元素。

示例

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE AGM>
<AGM>
<DEVICE>
  <NAME> 设备 1 </NAME>
  <START_UP>
  …
  </START_UP>
  <SET_ACTIVE>
  …
  </SET_ACTIVE>
…
  <FORM name="<对话框名称>">                用于用户窗口的标签
    <INIT>
      <CONTROL name = "编辑 1" .../>       用于输入栏的标签
    </INIT>
    <PAINT>                                 用于文本显示或图片显示的标签
      <TEXT>大家好! </TEXT>
    </PAINT>
  </FORM>

</DEVICE>
…
</AGM>
```

4.6.8 SOFTKEY_OK, SOFTKEY_CANCEL

说明

标签 SOFTKEY_OK 的用途在于，在按下软键"OK"关闭窗口时覆盖标准属性。 标签 SOFTKEY_CANCEL 的用途在于，在按下软键"CANCEL"关闭窗口时覆盖标准属性。

在该标签内，可以执行以下功能：

- 数据复制
- 有条件处理
- 循环处理

编程

标签：	**SOFTKEY_OK**
句法：	<SOFTKEY_OK>
	...
	</SOFTKEY_OK>
标签：	**SOFTKEY_CANCEL**
句法：	<SOFTKEY_CANCEL>
	...
	</SOFTKEY_CANCEL>

索引