PC WORX 6

IEC 61131-3编程

PC WORX 6

IEC 61131-3

编程

培训教员: _____
AUTOMATION SERVICE CENTER
自动化服务中心
如果有任何问题或建议，
请致电:+86 025 5210 2908

软件体系结构

第 1 讲

内容

本章节概括介绍了PX WORX 界面。

作为一款组态软件，PC WORX提供了五个自由可配置的工作区，工作区显示可随意显示/隐藏的停驻窗口。根据预先分配，这五个工作区考虑了对主要的PC WORX 功能（总线组态，编程和过程数据链接，工程比较以及现场设备组态）的访问。

工作区的复位可通过*Extras→Options（附加→ 选项 ）菜单*，选择 *General （常规）*标签完成。

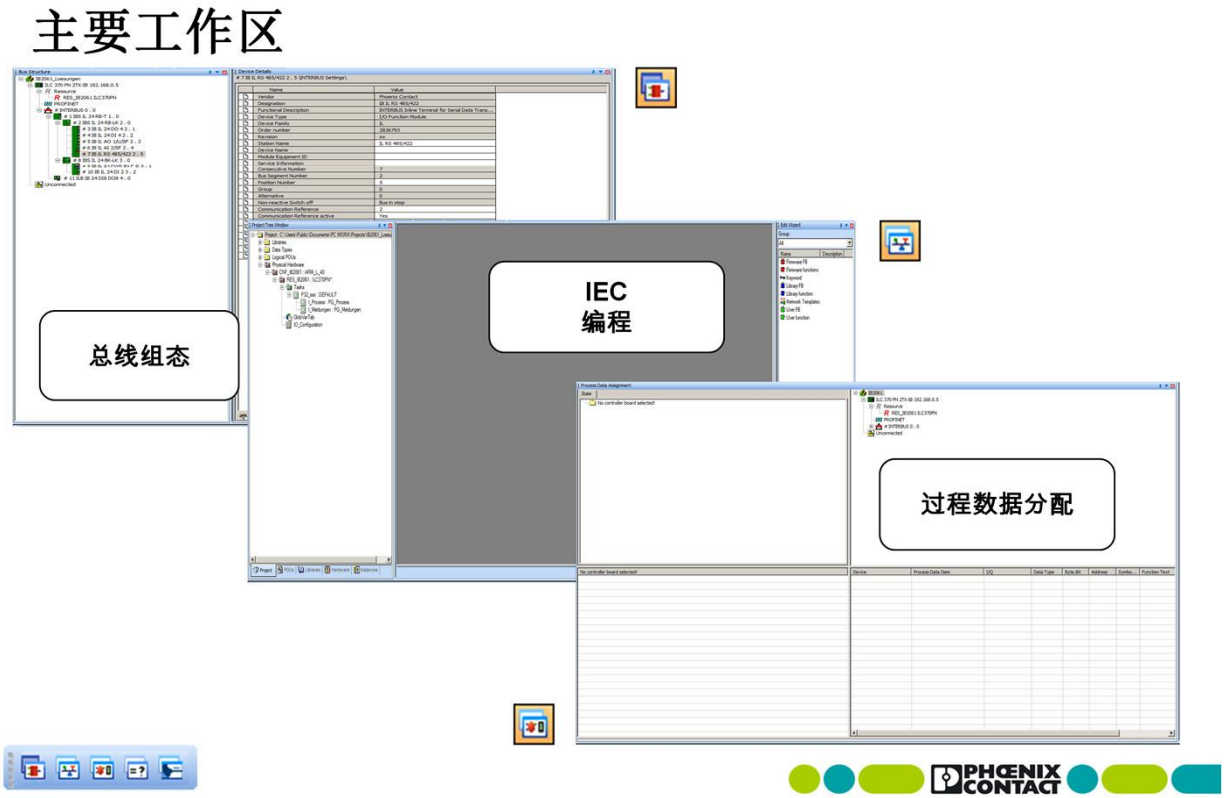
尽管工作区是自由可配置，但还应该清晰地加以组织，而不应当过多地加载显示窗口，并根据当前应用任务，在工作区之间进行切换，以达到有效使用。

注意!

信息

提示

工作区



图中所示的是标准窗口和其命名，与PC WORX的主要功能一一对应。这里所示的是三个主要工作区。菜单栏确保了不同工作区之间的快速切换。

总线组态

创建控制器所支持的总线系统（INTERBUS，PROFINET和AXIO BUS），编辑所使用的设备和常规设备的数据管理。

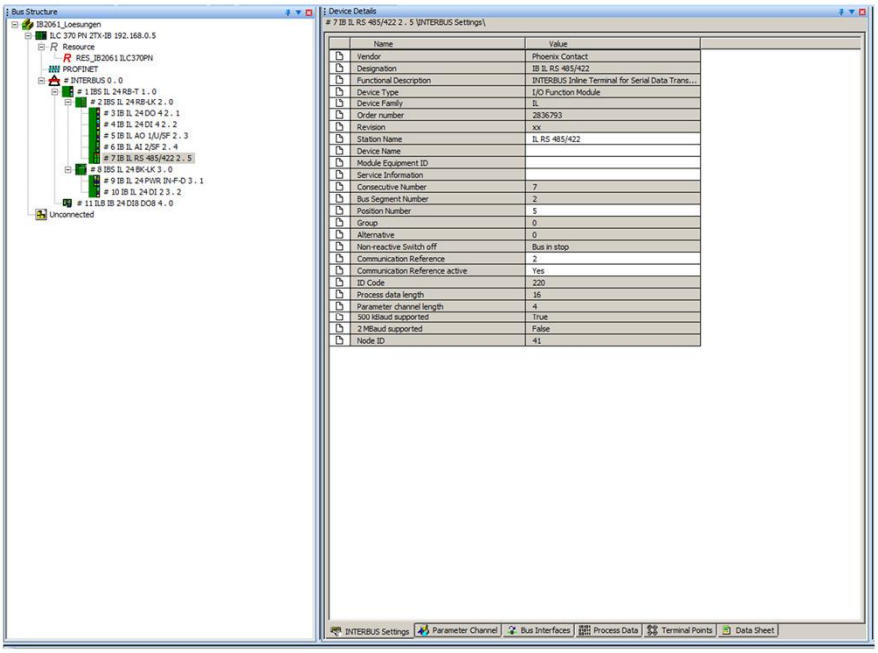
过程数据分配

将总线模块所提供的过程数据项与程序的全局变量相连接，或根据变量命名标准，创建基于上述过程数据对象的全局变量。

IEC编程

基于IEC的编程（程序、功能块和功能），用于创建PLC硬件与任务，声明用户定义的数据类型以及集成库和更多的软件相关功能。

总线结构工作区

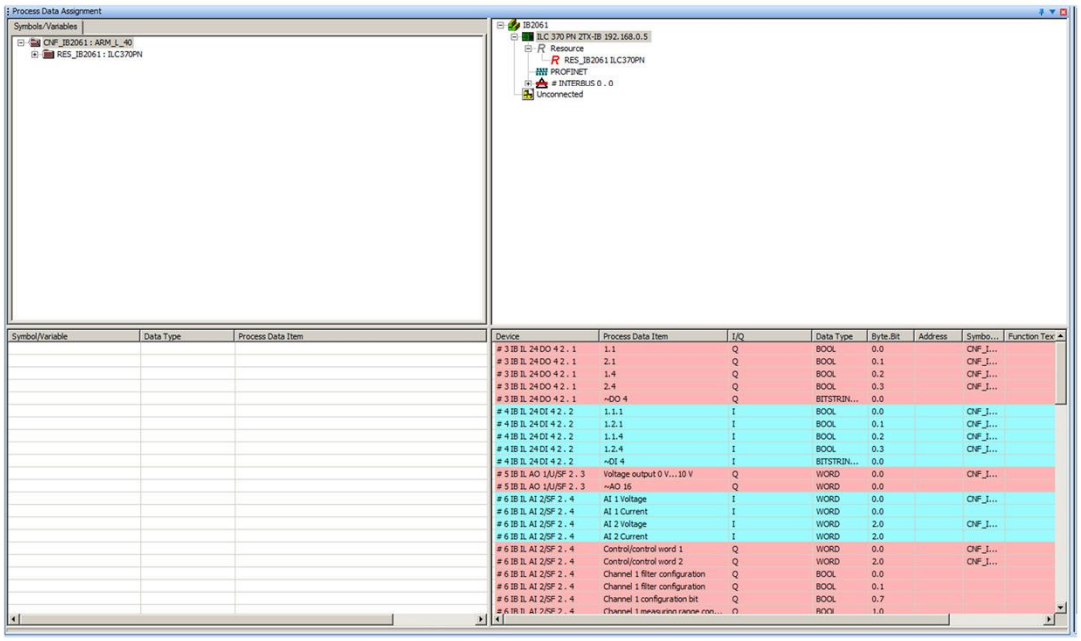


通常，在一个工程中，首要完成的是硬件组态，因此， 对用户而言，*Bus configuration（总线组态*）工作区是最初的工作环境。在标准情况下，显示*Bus configuration* （总线组态），*Device details*（设备细节）和*Device catalog*（设备类别）窗口。由于设备类别只在编辑阶段才需要，这里，有意不显示。

设备细节直接与总线组态相关。在总线组态中所选择的元件在设备细节窗口中显示了其细节。在窗体中，标签页的数量和类型是不同的，这视具体元素类型 （工程、总线目录、总线设备） 而定。页面上要显示的信息是从设备或系统文件中读出。这说明了显示的延时非常短。基于文本的文件被“翻译”，并在设备细节窗体中以一种方便用户的方式显示出来。

完成总线系统组态后，完整的设备信息可以从设备细节窗口中访问到（设备ID，站名等）。

过程数据分配工作区

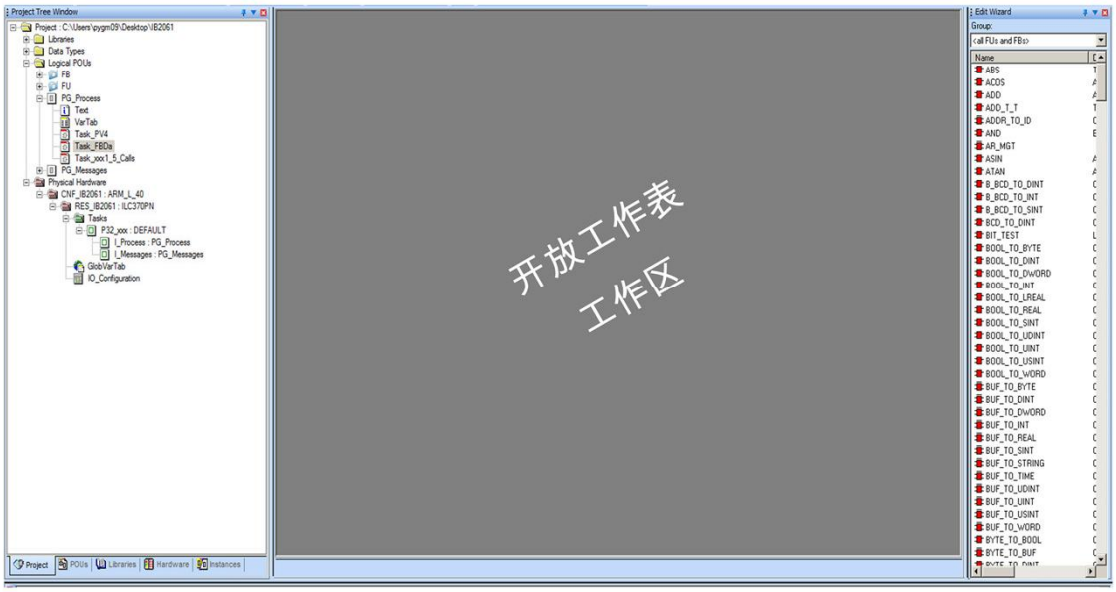


过程数据分配工作区划分为四个区，左侧显示了PLC及其结构。根据所选项，在下方的表中显示了其相应的全局变量。
右侧显示了总线系统硬件。 根据所选项，这里显示了其相应的过程数据项：

- 对于单一设备：仅有所选设备的对象；
- 对于总线终端模块：其分支中所提供的所有过程数据项；
- 对于控制系统：该控制系统中的所有对象。

在此窗口中，编程变量与所连接的总线系统中的过程数据对象项相连接。

IEC编程工作区



该工作区用于PLC基于IEC的编程，它分为工程树和编辑向导。

工程树除了对工程作整体显示外，还为程序组织单元（POU），库，硬件和任务以及程序（实例）的功能调用结构提供专门的显示。

编辑向导是与软件环境相关的。 这意味着，根据正在进行内容，向导为用户自定义数据类型的创建或程序的创建提供帮助。

屏幕截图中灰色工作区用于显示开放工作表（代码工作表和变量表）。与许多其它应用程序一样，这些窗口可以以不同的风格进行排列（参见*Window*（窗口）菜单）。

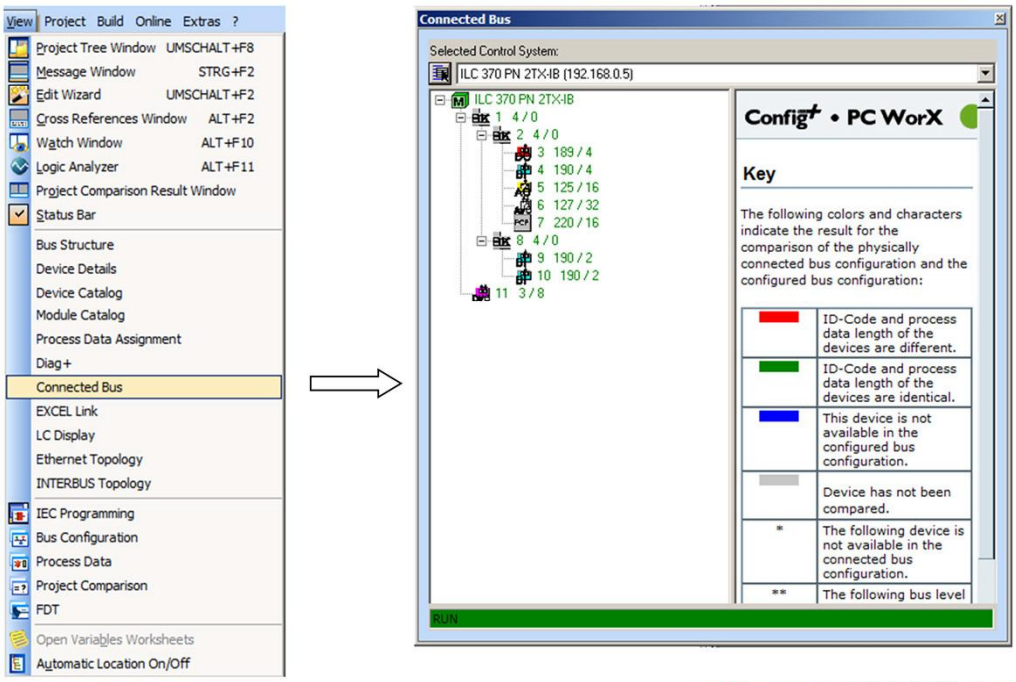
为了使工作表最大化，有效的做法是隐藏在编译的过程中自动跳出的工程树、编辑向导和消息窗口。预定义的快捷方式有：

Shift+F8	隐藏/显示工程树
Ctrl+F2	隐藏/显示消息窗口
Shift+F2	隐藏/显示编辑向导

通过菜单项*Tools → Shortcuts*（工具 → *快捷方式*）可对快捷方式进行调整。

窗口（选择）

连接的总线

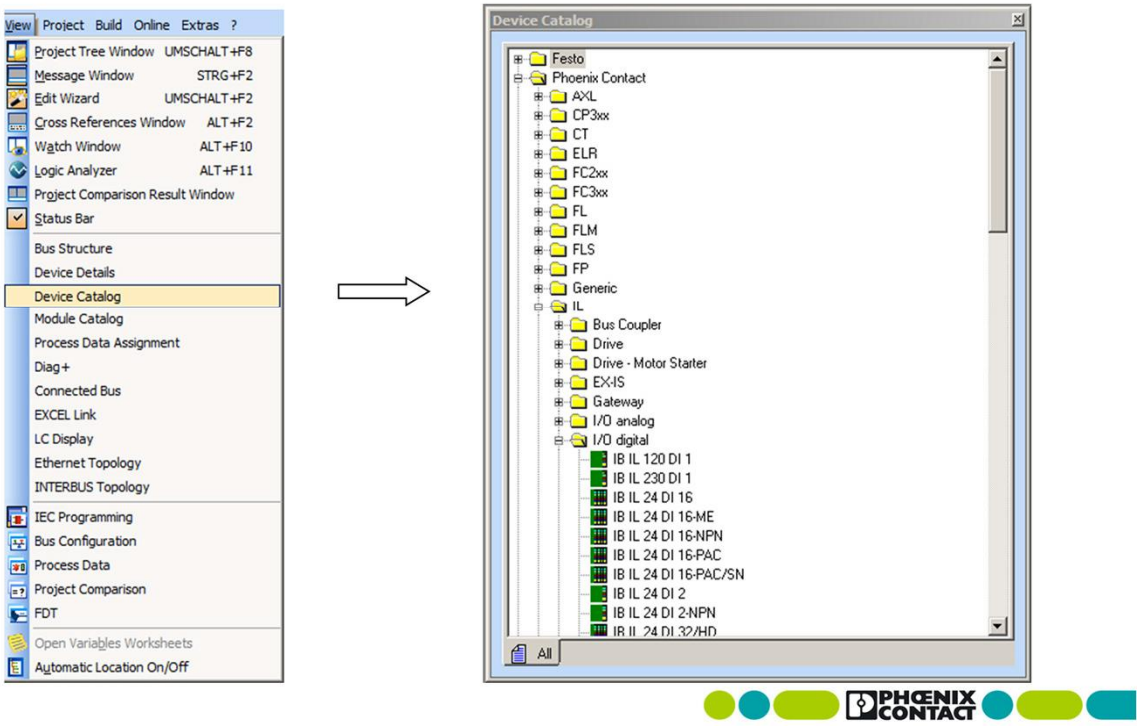


要读入相连接的INTERBUS系统，用户通常只需激活连接总线窗口。该窗口的通信路径可集中设置，并可经*Selected Control System（选取的控制系统）*列表调用。

图例

红	设备的ID代码和过程数据长度不同；
绿	设备的ID代码和过程数据长度相同；
蓝	在配置的总线组态中该设备不可用；
灰	设备没有进行比较。
*	后面的设备在当前总线组态中不可用；
*	后面的总线层在当前总线组态中不可用；
X / Y	ID代码和过程数据长度。

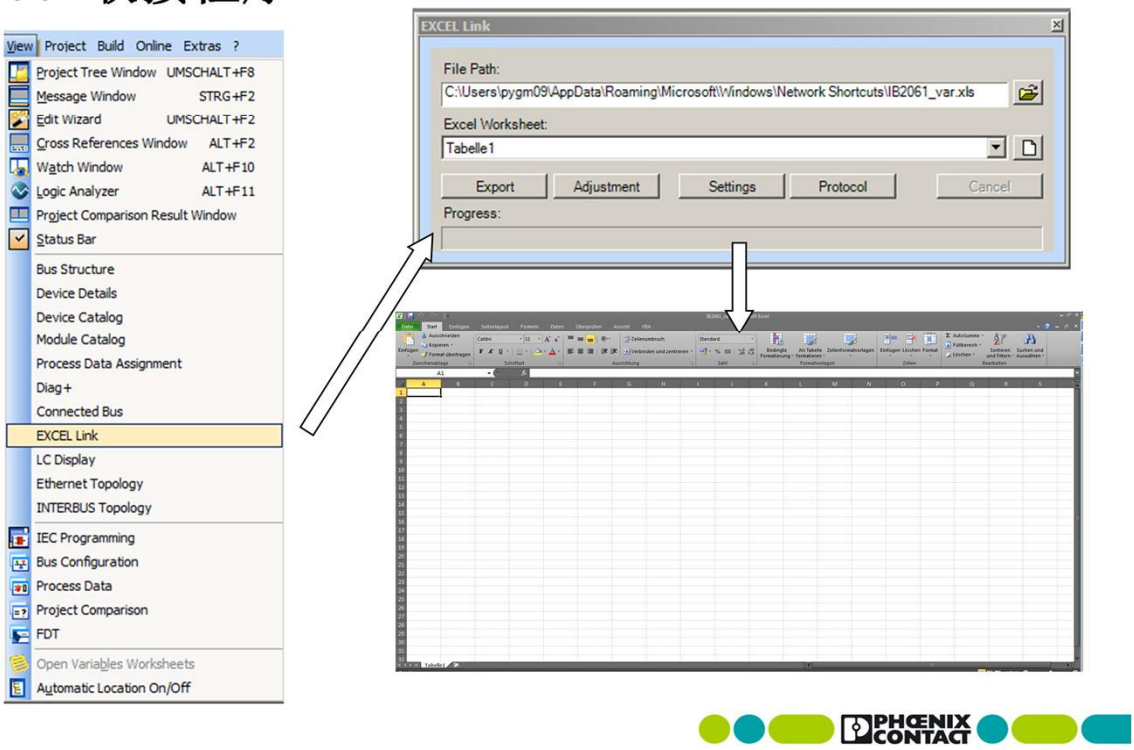
设备类别



*Device Catalog（设备类别）*窗口在总线系统离线组态时需要，必须在操作阶段显示。在在线组态过程中，更正误选设备时也需要。

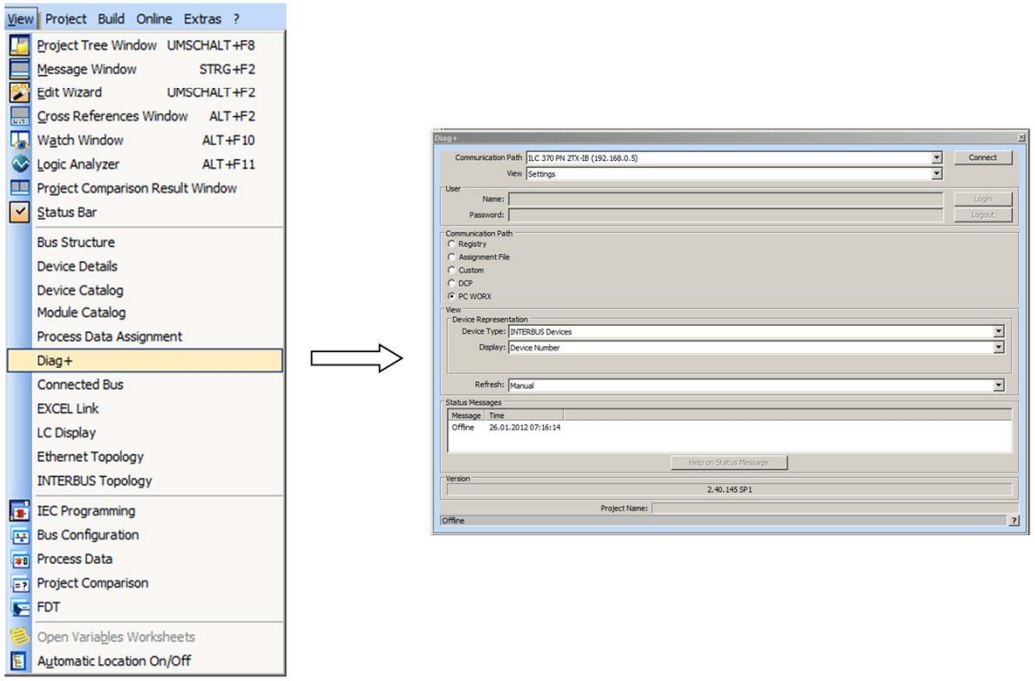
在设备类别中，所有设备描述文件都在*All*（全部）标签中列出。通过右键快捷菜单，用户可调整此工程树结构。

Excel 联接程序



集成在PC WORX 中的Excel联接程序（与MS Excel 兼容）作为一个接口，可使与过程数据项相链接的全局变量成批导入/导出。

Diag+ 2.0



INTERBUS，PROFINET和AXIO BUS是控制系统所支持的总线系统，集成在 PC WORX 6 中的Diag+ 2.0软件为其提供了广泛的诊断功能。与其它的窗口一样，Diag+也是使用集中设置的通信路径。

硬件体系结构

第2讲

内容

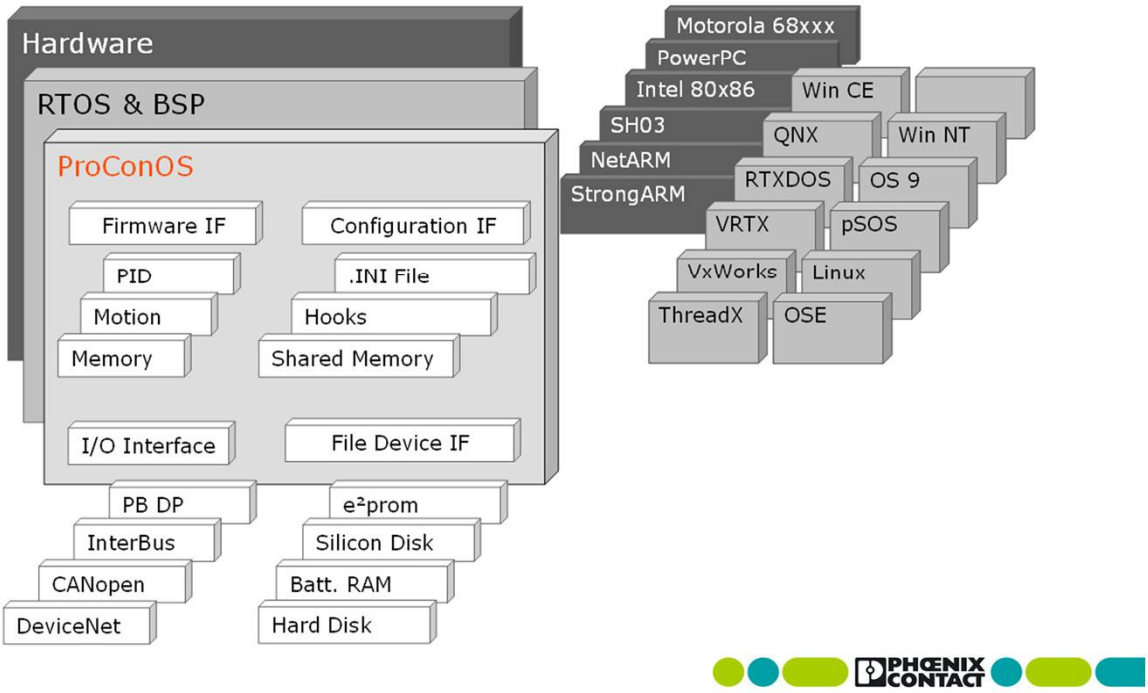
本章节介绍了ProConOS操作系统的功能，在不同的硬件平台上的实现以及采用ProConOS时硬件资源管理的实现。

注意!

信息

提示

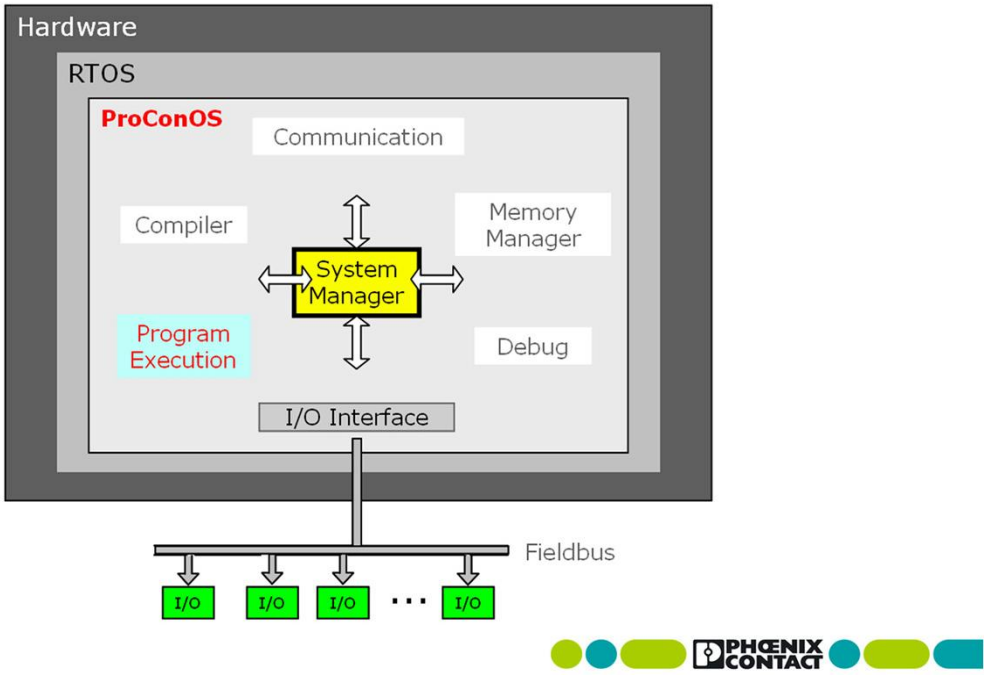
概况



在专用的或标准的硬件平台上，需要ProConOS (**Pro**grammable **Con**troller **O**perating **S**ystem可编程控制器操作系统) 来提供典型的PLC系统服务，包括外部创建的PLC程序的装载和处理以及为PLC控制的机器和系统的编程、安装和维护提供的调试功能。

大多数菲尼克斯控制系统都采用ProConOS作为操作系统，这样的优势在于，所有的控制系统都可以在相同的开发环境下（PC WORX）进行参数化和编程。

多任务处理



ProConOS基于标准的多任务操作系统，根据任务的优先级进行控制。在 ProConOS中，具有三个不同的优先级：

•用户任务优先级；
•ProConOS任务优先级；
•系统管理器任务优先级。

该分类确保了PLC完全可用的处理器计算时间总是有利于应用相关任务而分配的。只有那些应用任务的定时允许，其它系统任务才能接收到它们要求的计算时间。对应用的时间确定性而言，这是ProConOS内在前提条件。这样，在ProConOS系统体系中使用多任务旨在可预测的时间响应，性能优化，即响应时间最小化，以及对运行时错误作具体响应。

用户任务包括循环任务，事件任务和DEFAULT（默认）任务。在一个确定的时间间隔内，循环任务根据用户定义的优先级周期性执行。此外，对非周期性发生的事件（如硬件中断）的响应则在事件任务中执行。

DEFAULT（默认）任务的优先级最低，在没有其它任务激活时执行。 DEFAULT（默认）任务是一个循环任务。

在调整ProConOS以适应不同的菲尼克斯控制平台时，将使用不同的接口来实现控制专用特性。

PC - 控制系统之间的通信

第 3 讲

内容

本章节介绍了编程设备和控制系统之间的通信，描述了根据控制系统类型所提供的不同通信路径。对于基于网络的通信，这里展示了如何给控制系统分配IP地址，正如BootP服务的使用。

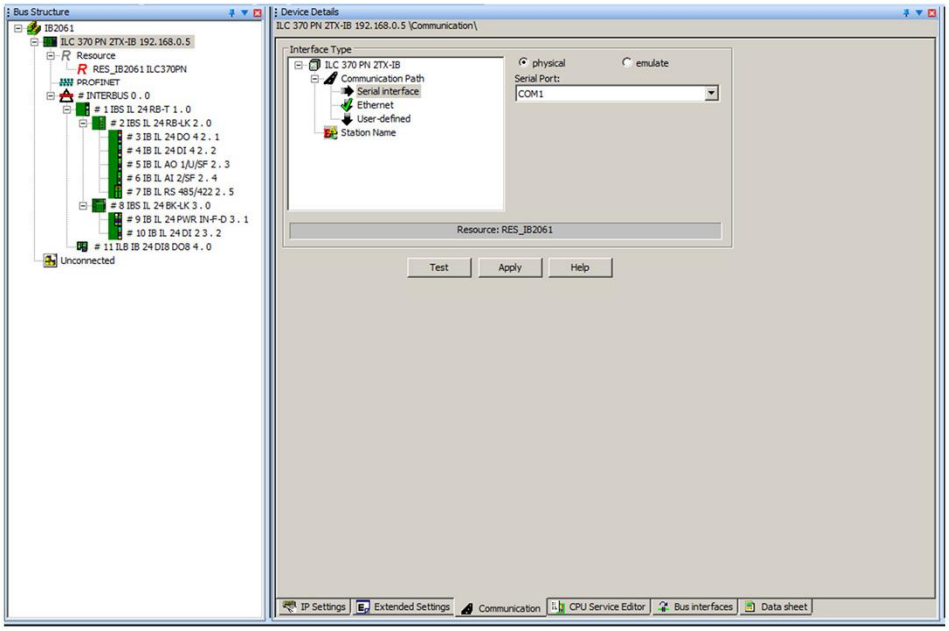
注意！

信息

提示

通信路径

串行接口



选中*控制系统*项，在 *Communication* （通信） 标签上选择连接编程设备和控制系统的通信路径。根据不同的控制系统类型，不同的通信方式可供选择。

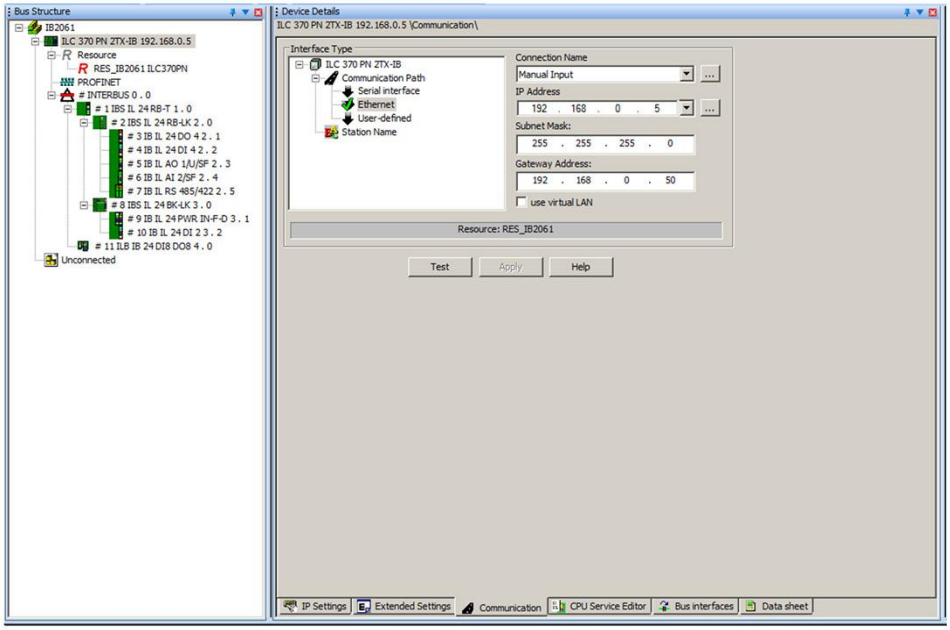
每个控制系统提供了一个串行接口，可使用标准的IBS连接电缆或带有Mini-DIN连接器的连接电缆。如果编程设备不提供串行接口，可使用USB适配器，这可能有必要选择模拟接口。

按下*Test* （*测试*）按钮，激活所建立的连接，并在状态栏中显示。

通过*Apply* （应用）按钮，则在当前的工程中为当前控制系统定义了通信路径。使用这些设置，其它窗口和对话框都可以对控制系统进行持续的访问。

当使用前面配置的控制系统时，由于该连接的一个连接名已保存到控制系统中，因此可能会弹出“ *Connection names do not match* （连接名不匹配）”消息。这将会防止对控制系统的意外访问。这时，不需要对此作应答。

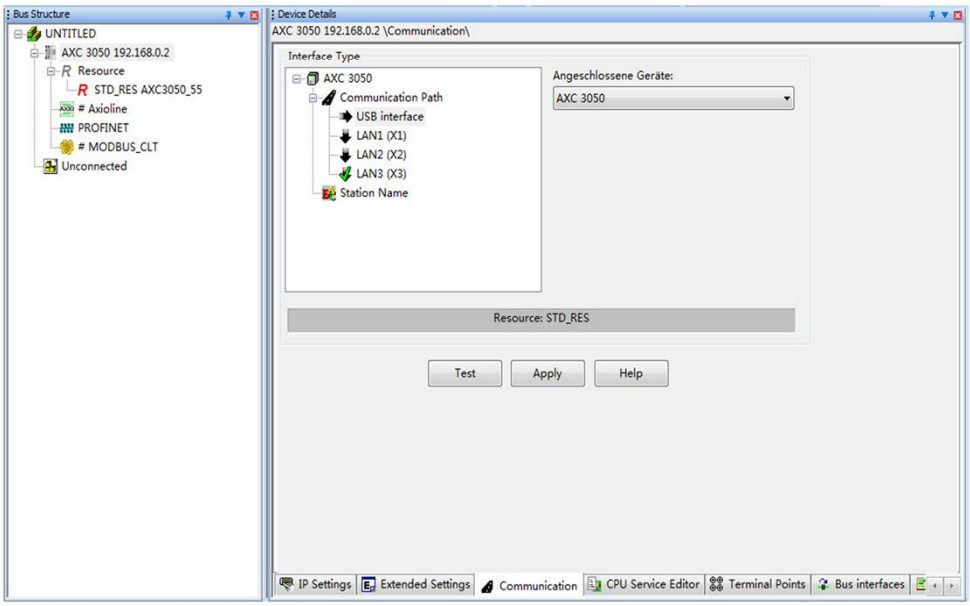
TCP/IP



除了串行接口外，大多控制系统提供了TCP/IP通信选项。

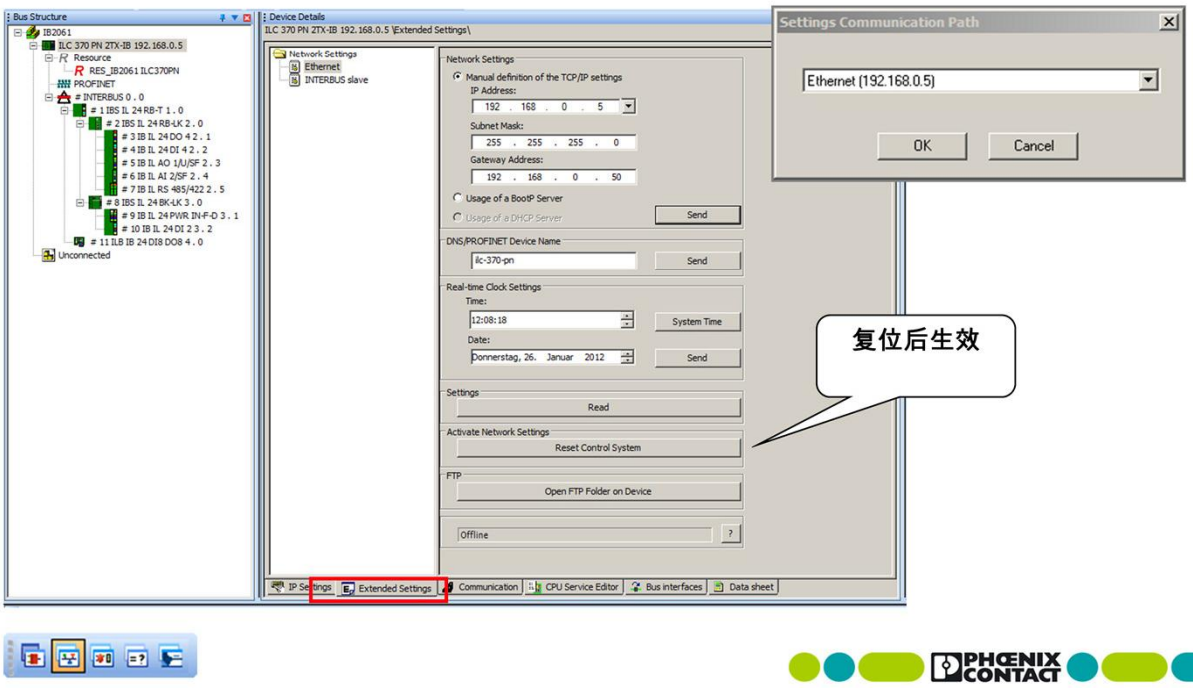
对于这一类型的通信， IP地址必须保存到控制系统中。
此外，该地址必须与PC WORX进行通信。

USB Interface



控制系统网络设置

设置IP地址•实时时钟

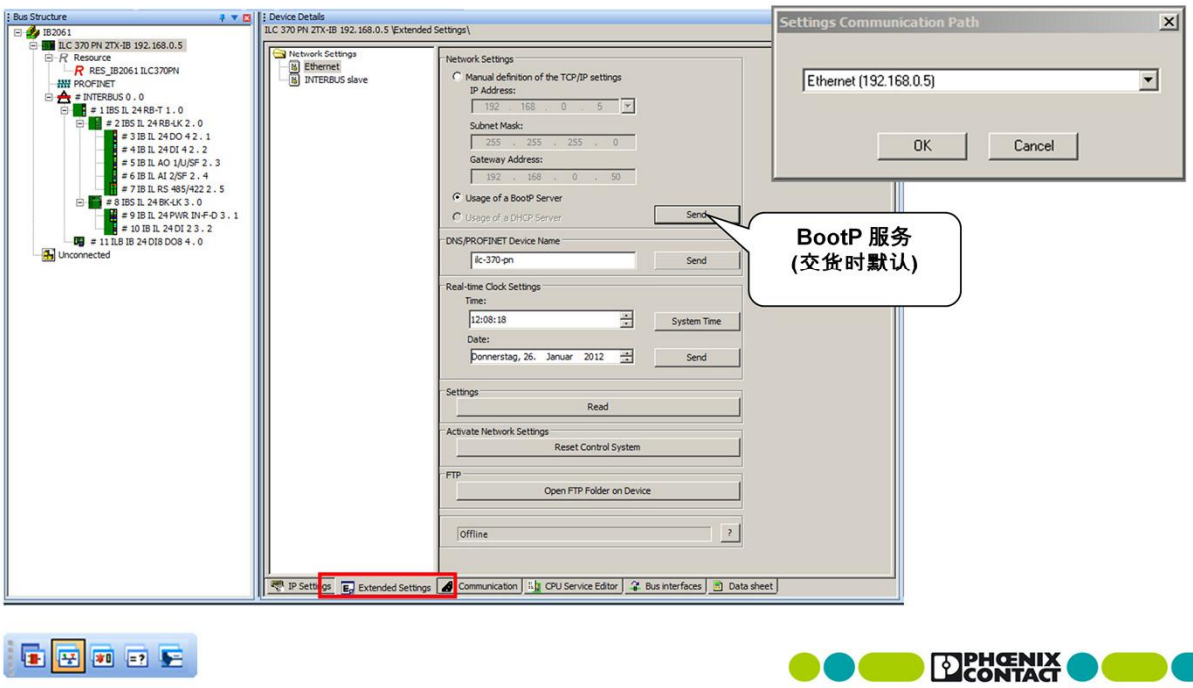


采用含实时时钟的控制系统，选择*控制系统*后，在*Extended Settings*（扩展设置）标签上设置IP地址和时间。 在*Network Settings*（*网络设置*）框中，选择*Read*（读取）功能后，显示控制系统的网络配置（交货时默认： *Usage of BootP server*（使用*BootP服务器*））。为此，需要选择一个准备用于操作的通信路径。通常，这是串行接口。

根据工程需求建立通信。通过调用和发送*System Time*（*系统时间*）调整控制系统的实时时钟。复位控制系统使该调整生效。

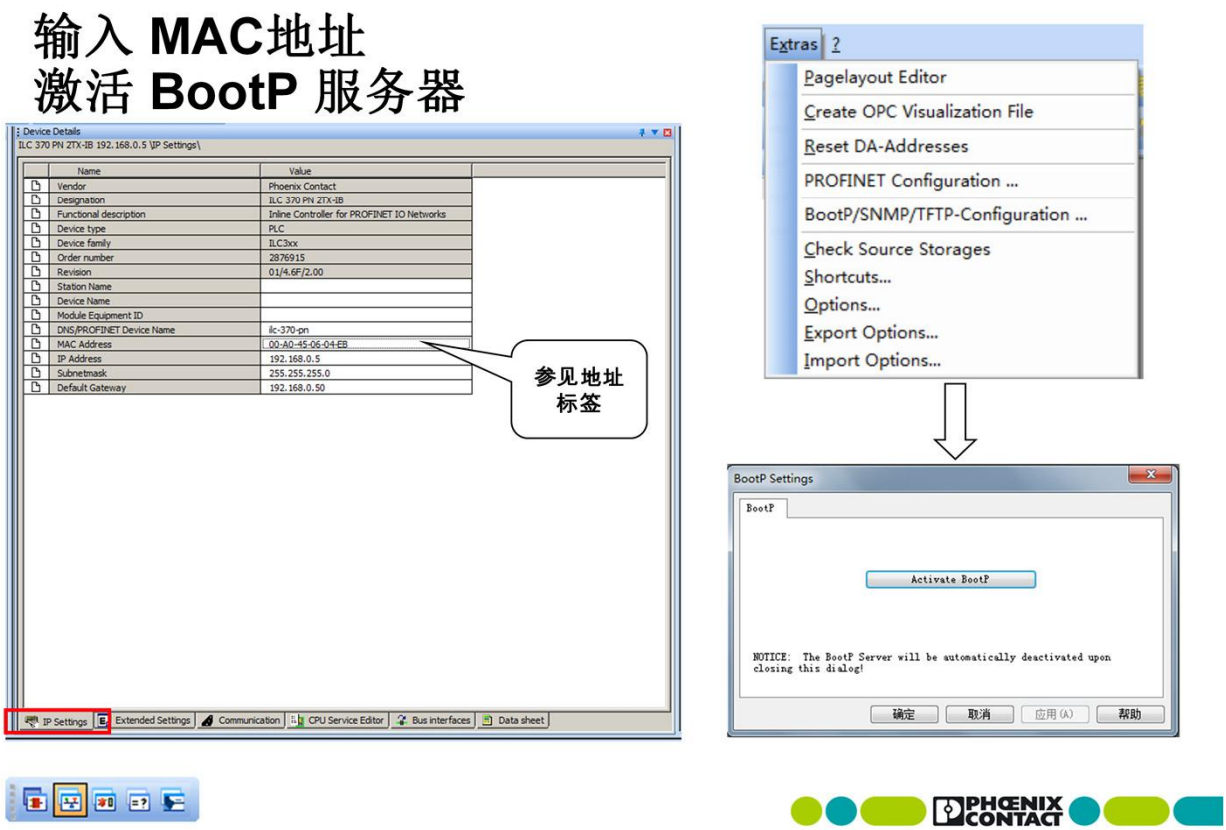
由于控制系统具有一个FTP服务器，借助一个配置成功的网络通信，通过*Open FTP Folder on Device*（*打开设备上的FTP文件夹*）可以启动安装在操作系统下的FTP访问软件。这样，可以显示紧凑型闪存卡中FTP区的内容。

经BootP分配 IP 地址



如果在BootP server启动后（如： PC WORX ）要使控制系统接收到其IP地址，则要在扩展设置中选择*Usage of BootP server*（使用*BootP server* ）选项。正如传输一个新静态IP地址一样，必须发送新组态，并复位控制器，使之生效。

当使用BootP选项时，如果更换控制系统，仅仅更换CF卡还不行，新的MAC地址还必须与BootP server进行通信。

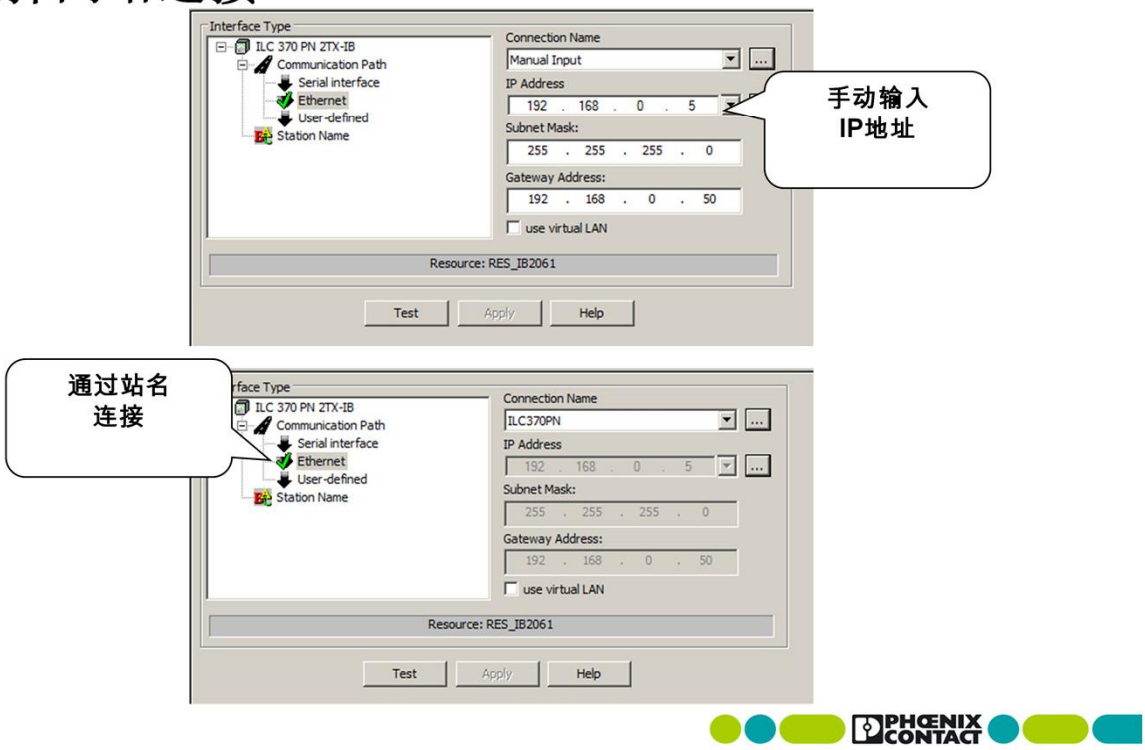


欲通过集成在PC WORX中的BootP server 来使用控制系统的BootP选项，可能有必要经*Tools（工具）*以及 → *BootP/SNMP/TFTP-Configuration（ → BootP/SNMP/TFTP-配置 ）*来激活BootP 服务器。为了对控制系统分配IP地址，则必须在*Extended Settings（扩展设置）*中输入控制系统的MAC地址。

MAC地址（16进制）条形码标签可在外壳的前侧或左侧找到。

网络通信

选择网络连接



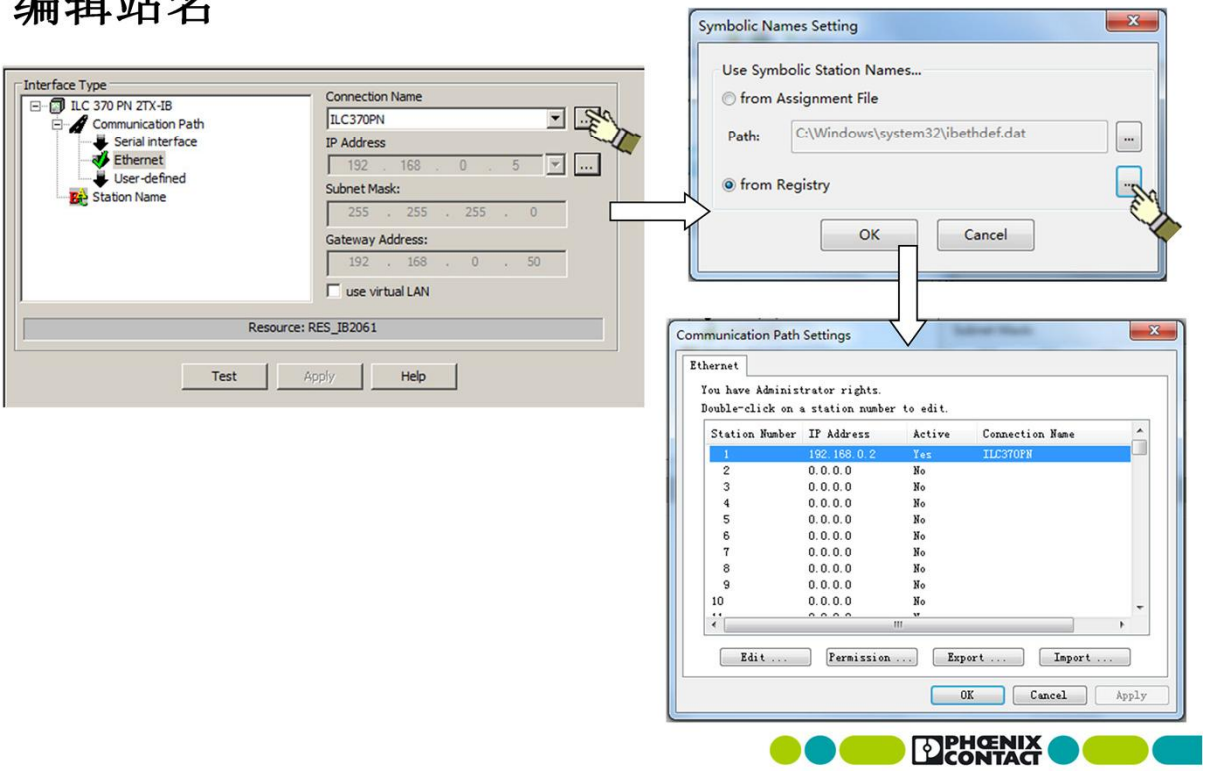
如果控制器已配置了一个有效的IP地址，则该地址得提供给PC WORX。

如果已选择了*TCP/IP (Ethernet/Local host)*（以太网/本地主机） 通信路径，则有两种方式实现：

1. 选择*Manual Input*（*手动输入*）项，在激活的输入框中输入期望的IP地址；

2. 从连接名列表中选择一个站，通过地址文件或注册表的数据项，将站名与IP地址相连接。

编辑站名



通过... *按钮*，可以装载地址文件或更改站名数据项。

创建地址文件最简单的方式是更改注册表中数据项，然后导出该列表。使用文本编辑工具，所创建的文件（*.dat ）可被编辑为以制表符为分隔的文本文件。

如果地址文件或注册表中的数据项被激活，则它们只在*Connection Name* （*连接名*）字段中显示。

IBS 组态

第4讲

内容

本章节概要介绍了连接到控制系统上的INTERBUS系统的组态。
如果一个系统操作准备就绪，它要与控制系统相连接，则可以使用在线组态。而在硬件起动前需要进行修改和配置，则需要离线组态。
本章节还介绍了用户自定义设备类别 （对大规模生产更有意义）的创建和设备描述文件的导入。

于安装了
各
后

对于这两种组态方法，都需要工程中所使用到的设备的设备描述文件。由PC WORX，因此很大一部分设备描述文件在PC机中已经存在。更多的设件（ XML 格式）可以通过*Device Catalog* （设备类别）窗口复制各自目录册。

参见 *PC*

对于在线组态，则需要在PC和控制系统之间存在一个操作准备就绪的通信(- *控制系统之间的通信* 章)。

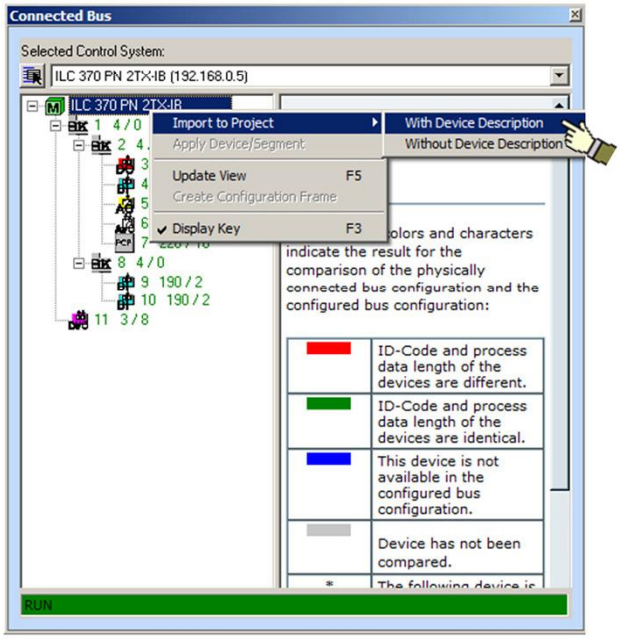
注意!

信息

提示

在线组态

读入相连的总线系统



对于在线组态，显示*View → Connected Bus*（视图→连接的总线）菜单。该窗口并非嵌入在已打开的窗口中，而是保持非停驻状态。为了使当前使用的工作区（通常是总线组态工作区）布置清晰，该窗口应切换至离线状态，并在读入后隐藏。

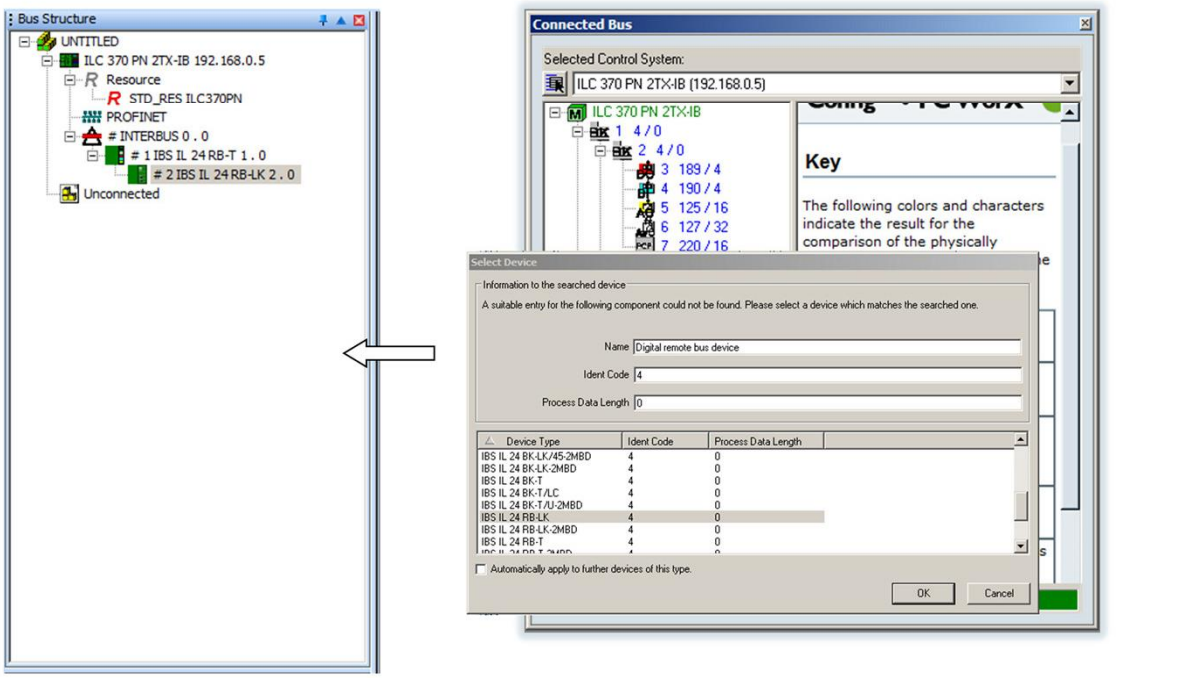
为了要读入相连的INTERBUS系统，则必须从*Selected Control System（选取的控制系统）*列表中选择当前工作的控制系统。然后，PC WORX 将显示保存到控制系统中的INTERBUS 的组态帧，或者是使用了*Ready*（就绪）状态下的系统，则创建一个新的组态帧。

如果系统尚未操作准备就绪或出现故障，则该窗口不能显示组态帧。对于诊断， 则可以使用*Diag+*窗口（具体操作参见*PC WORX 5* 中的*Diag+ 2.0* 章节）。

如果显示了总线系统，通过*Import to Project → With Device Description（导入到工程→ 带设备描述）*功能，可将其从右键快捷菜单复制到*Bus Configuration*（总线组态）窗口中。

请注意当选择该功能后，则放弃了一个已配置好的总线系统。如果仅仅是应用了总线系统中的单个设备（如：当组态不完整时添加设备），则应使用 *Insert Device/Segment*（*插入设备/段*）的功能。

插入检测到的设备



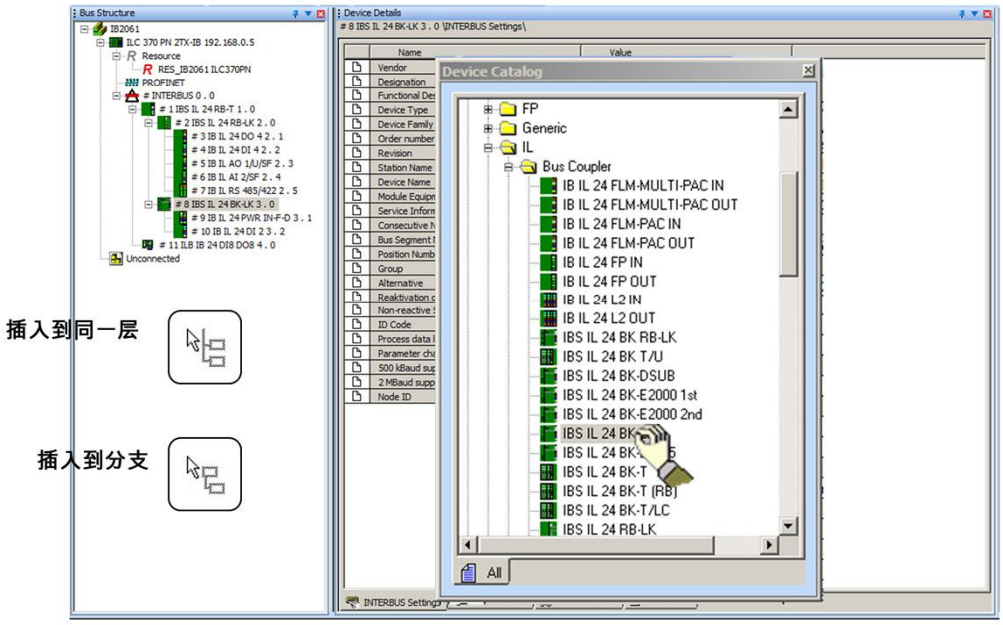
根据当前操作的设备，在*Select Device（选择设备）*对话框中显示在PC WORX 中注册的设备描述文件的清单。此外，根据当前设备的识别代码（ID）和过程数据长度也可以进行选择。

如果在PC WORX中已创建了用户自定义设备类别，在选择设备前，则须选择该目录，在搜索目录过程中，其功能应当能访问到该目录。

如果选择出错，则不应中断该选择过程，而应继续。在多数情况下，经设备类别的后续更正要比反复读入系统所花的时间要少。

离线组态

从设备类别中插入设备



对于离线组态，需要从视图菜单激活*Device Catalog（设备类别）* 窗口。建议将该窗口定位在*Bus configuration（总线组态）*和*Device Details （设备细节）*窗口之间。这样，虽然后者部分不可见，但如*Connected Bus（连接的总线）* 窗口一样，当不需要设备类别时，将不显示设备类别。关于总线组态的目录功能限于三种动作：

1. 将设备插入到相同的层中（位于远程总线设备后的远程总线设备或位于本地总线设备后的本地总线设备 ）。
   这可以通过右键快捷菜单（在目录中*复制设备*，在总线组态中*插入到相同的层中*）或使用鼠标来完成。为此，必须从目录中选择设备，并（鼠标键按下）将设备拖放至总线组态中新设备插入点的前一设备处，鼠标指针显示如下符号：

   插入到相同层时的符号：

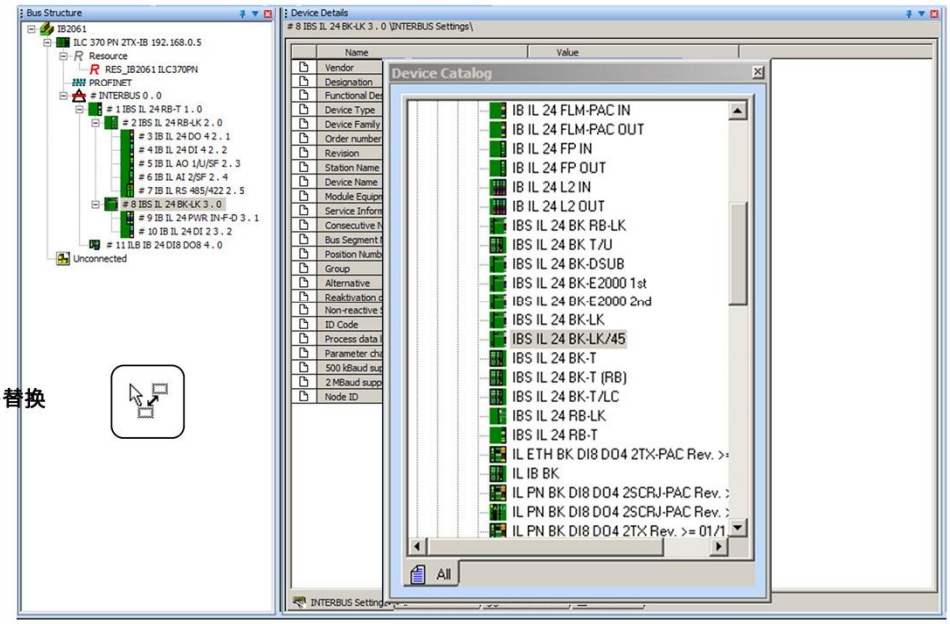
2. 将设备插入到下一层中（分支）（位于远程总线分支中远程总线设备或位于本地总线终端模块后的本地总线设备 ）。

   这也可以通过右键快捷菜单（在目录中*复制设备*，在总线组态中*插入到下一层中*）或使用鼠标来完成。为此，必须从目录中选择设备，并（鼠标键按下）将设备拖放总线组态中，与插入到相同层相对比，鼠标移至新设备插入点的前一设备的右侧，直到显示以下符号：

   插入到下一层时的符号：

   当移动鼠标，并按下shift键，可以在*插入至相同层*和*插入至下一层中*两种可能之间进行切换。

使用设备类别进行设备替换



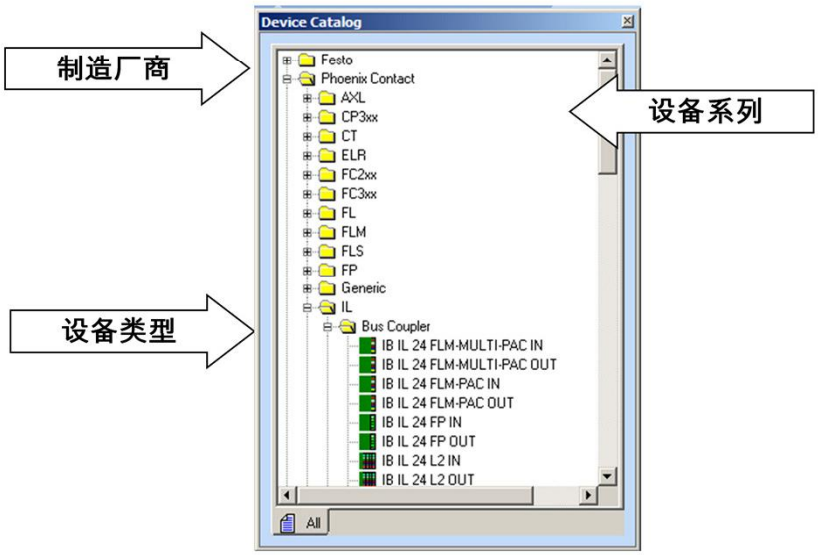
3. 替换设备（原则上设备提供相同的接口才有可能）。首先，通过右键快捷菜单，在设备类别中复制替换设备，然后执行右键快捷菜单功能“*更换*”。

使用鼠标和键盘，在目录中选择需要的设备，按住鼠标键不放，拖放至总线组态中待替换的设备处。虽然是按下了相同的控制键，但由于是替换而不是插入，鼠标指针显示如下符号：

替换过程中的符号：

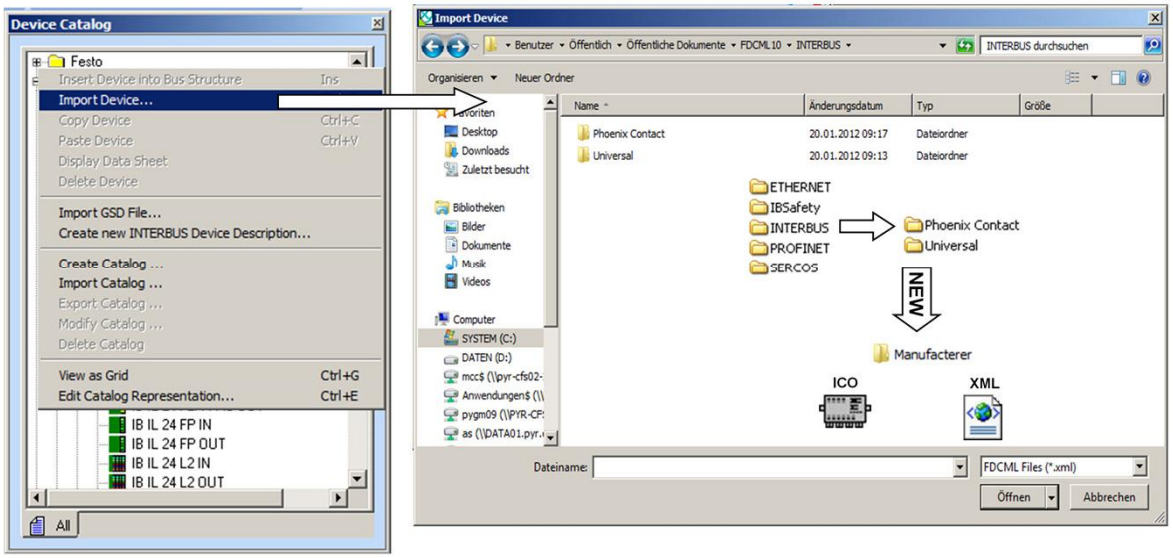
设备类别和设备文件

显示



软件安装后，在系统中目录以树结构呈现，标准的格式如下：供应商 – *设备系列*– *设备类型*。对整个目录而言，这种分类很实用，不过可通过右键快捷菜单中的修改目录（*Modify Catalog*）功能进行调整。对设备系列少的用户自定义目录而言，不采用任何结构可能更有意义。

设备文件的注册（导入）



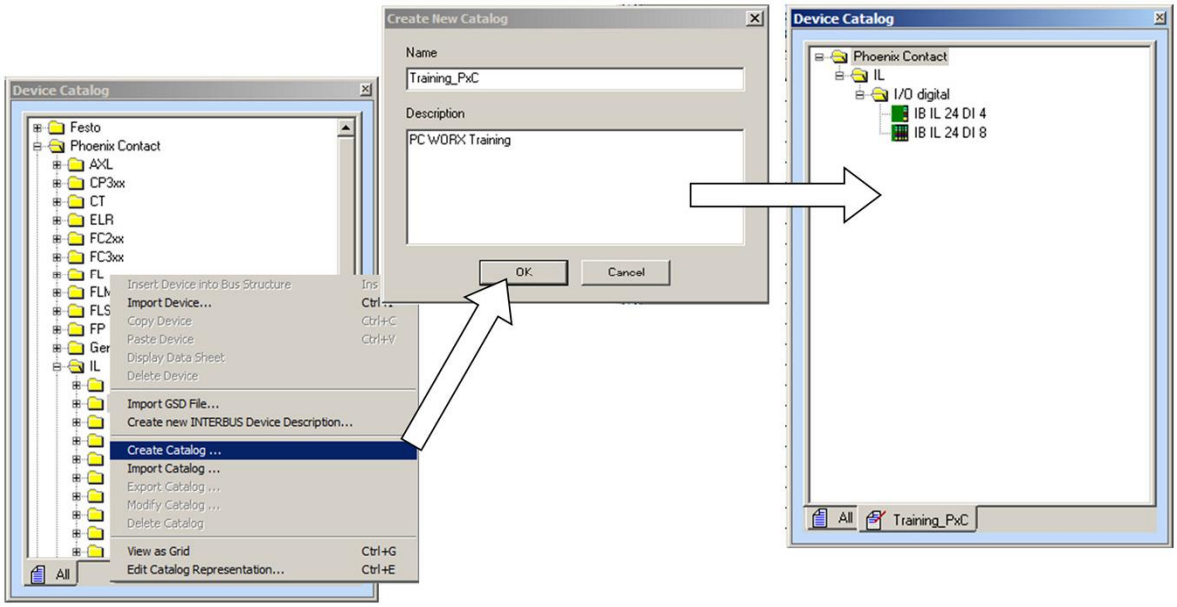
设备描述文件的注册（导入）可使PC WORX 能对其进行访问，这样，随后就可以使用这些设备。

除了一个XML文件外，一个完全的设备描述还包括一个能在XML文件中引用的图标文件。如果没有这样的图标文件， PC WORX 将显示标准的符号。

两个文件都必须保存在相同的目录下。为了能使附加注册的设备能保持总体的概念，在如上图所示的目录级中用设备供应商名来创建文件夹更有意义。该文件夹并不影响设备类别中的布局。例如，供应商并非从保存XML文件的文件夹名读取，而是从XML文件自身中读取。

由于文件不能生成本地副本，所以不能从移动设备 （如：USB设备） 中完成注册/导入。

用户自定义设备类别



如果使用数量有限的设备来实施多种项目，则创建一个用户自定义设备类别将很有意义。在*Device Catalog（设备类别）*窗口中，通过右键快捷菜单，创建一个新目录。一开始，该目录是空的，且没有结构。

使用右键快捷菜单从主目录中复制设备，并粘贴到新目录中，从而填充新目录。也可以如上图所示，通过拖放的方式来完成。

在离线和在线组态过程中，用户自定义目录可以使访问过程更加快速。对于后者，在插入设备时，只显示数量有限的设备。

每个用户自定义设备类别只能包含主目录中的设备，设备描述无差异。

PN 组态

第5讲

内容

本章节介绍了与Ethernet网络相连的PROFINET系统的组态。
在线组态针对的是PROFINET系统安装好的情况下的组态。而在硬件起动前需要进行修改和配置，则需要离线组态。

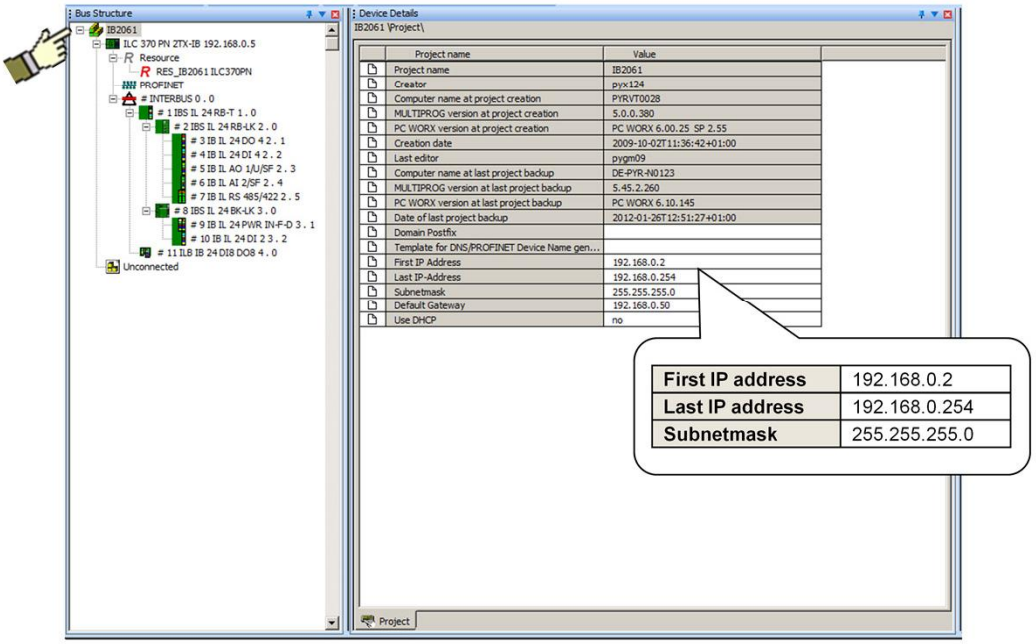
注意!

信息

提示

基本设置

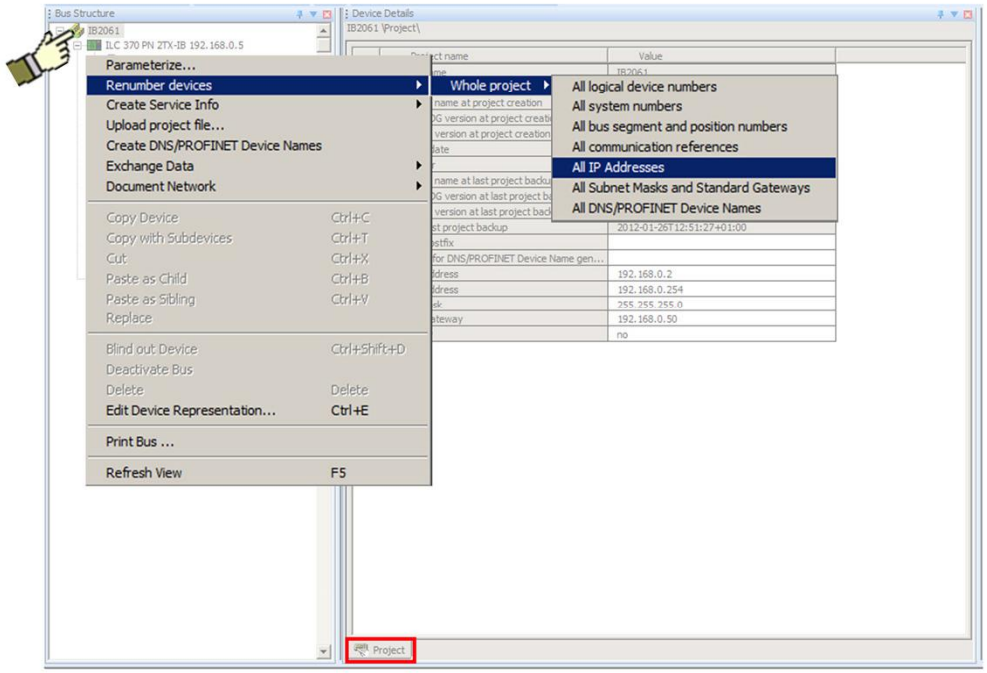
设置地址域



在*Bus Configuration（总线组态）*窗口 中，经工程项，在PC WORX 工程中创建的PROFINET网络的地址范围。对每一个被添加的PROFINET设备， 将自动从该地址域中导入一个IP地址到总线组态中。

在输入IP地址范围时，起始地址不能大于末地址。

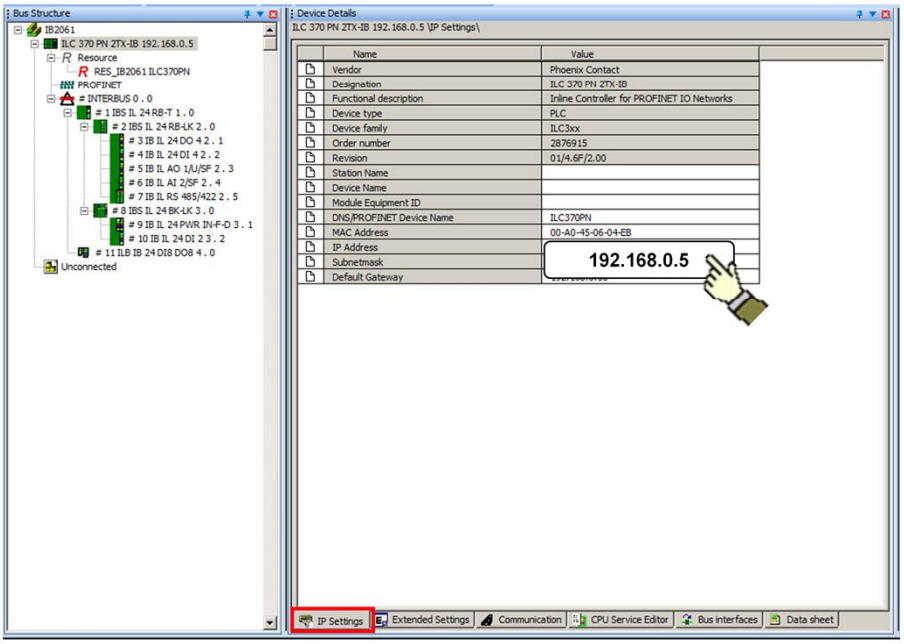
导入地址



通过工程的右键快捷菜单，可以集中调整总线组态中的IP地址，使其与所设置的地址域相适应。

如果更改子网掩码或标准网关，则必须执行*Renumber devices → Whole project → All Subnetmasks and Standard Gateways（设备重编号 → 整个工程 → 所有子网掩码和标准网关）*命令。

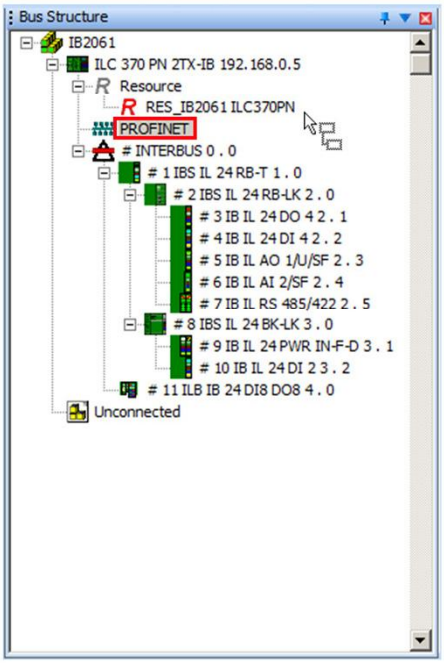
直接IP地址分配

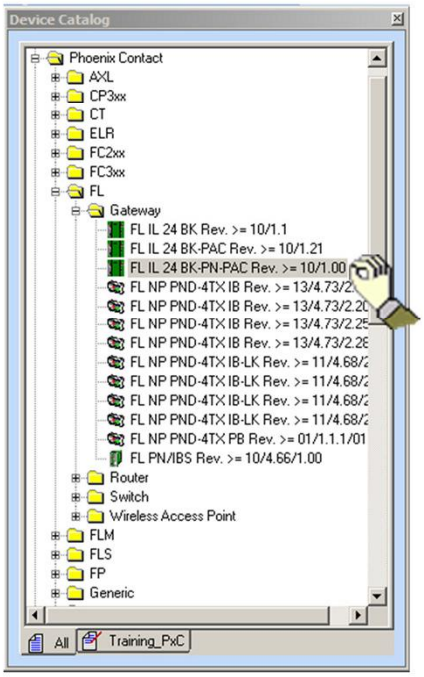


通过设备细节中的*IP Settings*（*IP设置*），可对每个设备进行单独的地址分配。

在线/离线组态

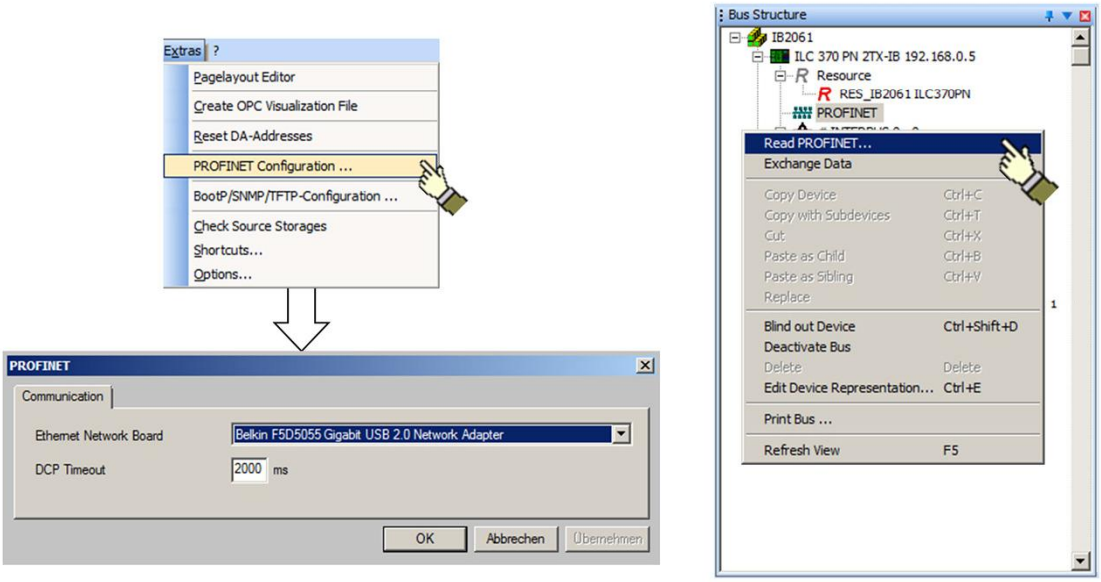
离线组态





PROFINET设备可从设备类别中添加至网络配置中，这可以用拖放的方式完成，或用右键快捷菜单中的复制和粘贴来完成。

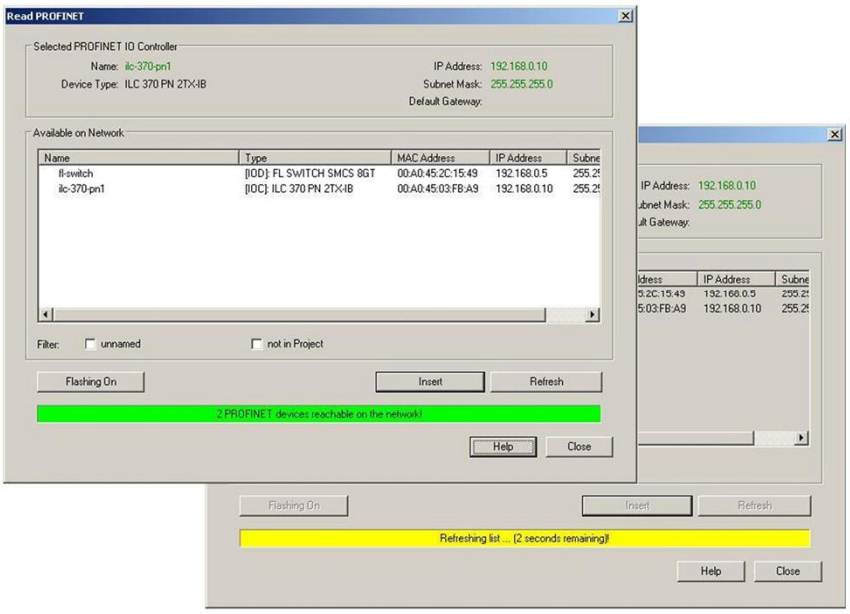
在线组态



如果待配置的设备都已安装完毕，并其设备描述文件也在PC WORX中注册，则PROFINET网络也可以使用PC WORX 进行在线组态。

尽管PROFINET系统已连接，但还是没发现PROFINET设备，如果是这样的话，则可能是在PROFINET 组态中安装了错误的网卡。

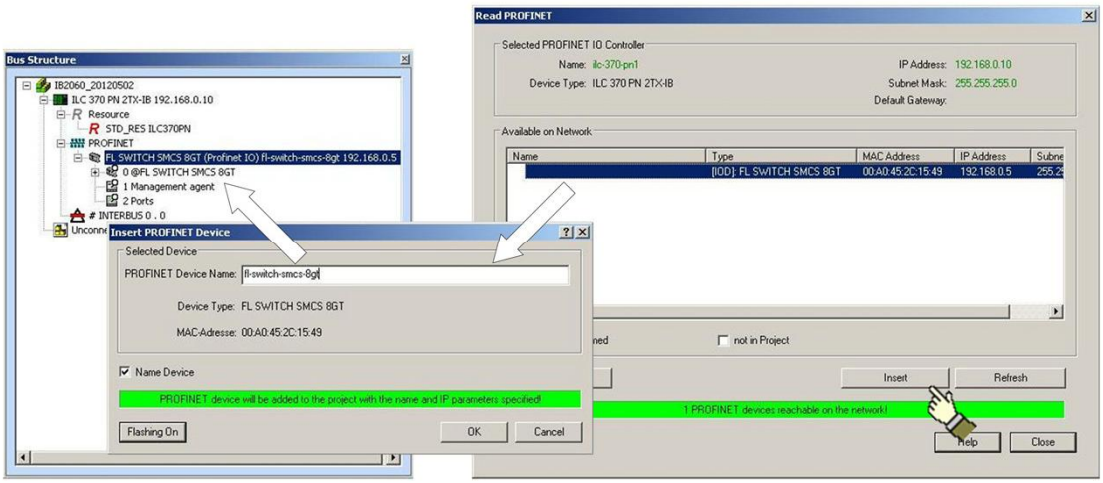
显示可达设备



以上列表显示了网络中所有有效的PROFINET I/O 设备，如果一些已连接的设备未列入，这可能是因为窗口下方的过滤选项造成的。

通过*Insert（插入）*按钮将所选的I/O设备插入到PC WORX 总线组态中。

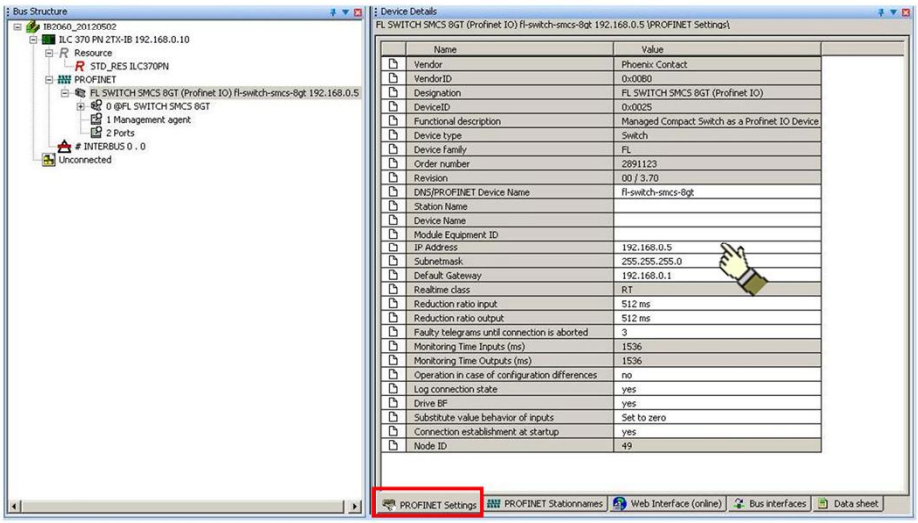
插入设备至工程中



在另一对话框中，可以直接将PROFINET设备名分配给I/O设备。首先，输入的设备名仅与总线组态中的分配有关。实际命名，即将设备名写入到I/O设备本身中，只有在激活*Name Device（设备命名）*控制框才能完成。

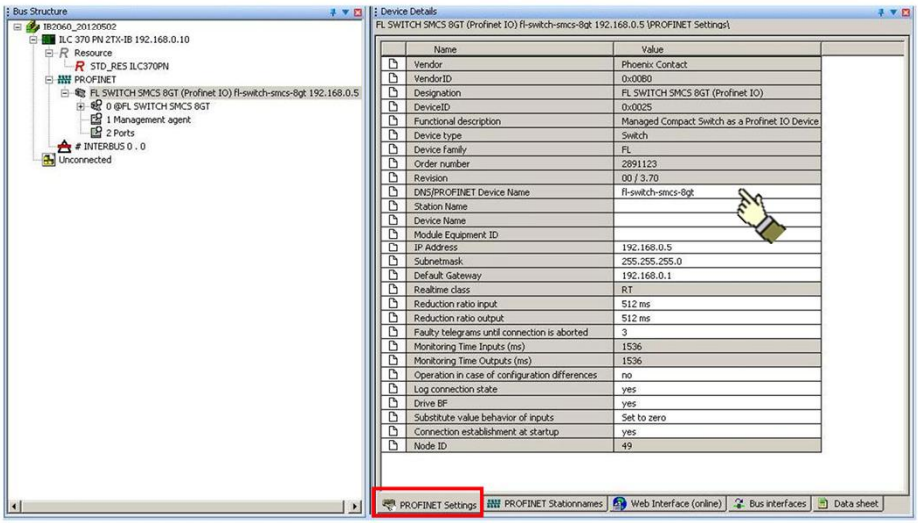
设备设置

设备设置



对于每一个PROFINET I/O 设备，其输入和输出的刷新时间可通过*PROFINET Settings*（*PROFINET设置*）标签设定。
各个I/O设备的刷新时间，以及每个设备的输入和输出的刷新时间可以不同。

设备名



通过*PROFINET Settings*（*PROFINET设置*）页面，可以为PC WORX 工程中的单个I/O 设备 分配设备名。

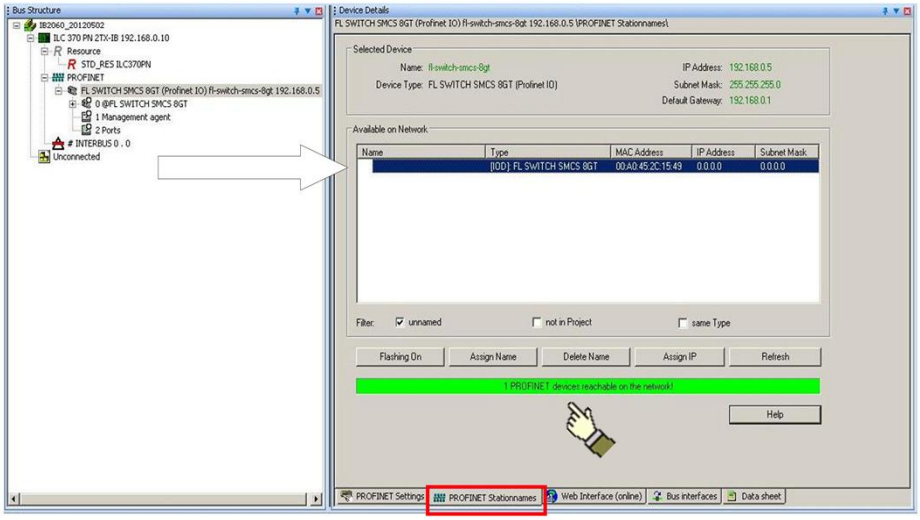
设备名分配时允许的字符集：

字母：　　*a-z*
数字：　　*0-9*
连字符：　　-

设备名分配专指在PC WORX 工程中的分配，实际设备命名必须单独执行。

在PROFINET中，设备名具有最高优先级。它用于识别PROFINET系统中的I/O设备，从而在整个网络中使用时应清晰明了。

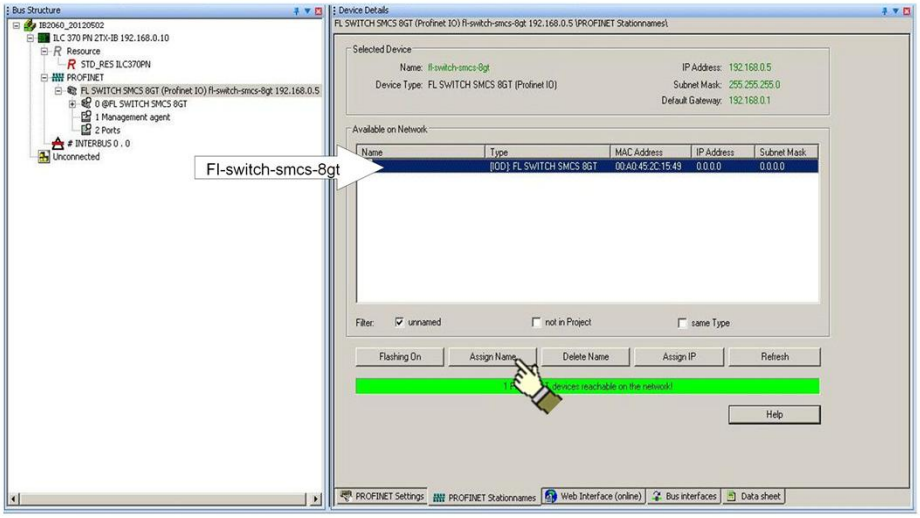
网络中的PROFINET设备



通过*PROFINET Stationnames*（*PROFINET 站名*）标签，读入并列出网络中所有有效的PROFINET I/O 设备，如上图所示。使用该对话框，可以为各个I/O设备写PROFINET设备名和相应的IP 地址。使用*Assign Name*（分配名称）按钮来完成实际设备命名。

在总线组态中只有选择了PROFINET I/O设备，*PROFINET Stationnames*（*PROFINET 站名*）页面才有效。

PROFINET设备命名



在分配名称时，在总线组态中所选择的I/O设备的名称被分配给了*Device Details（设备细节）*窗口中所选择的I/O设备。请确认所选择的设备类型是否相同，设备名将永久保存在各个I/O设备上。

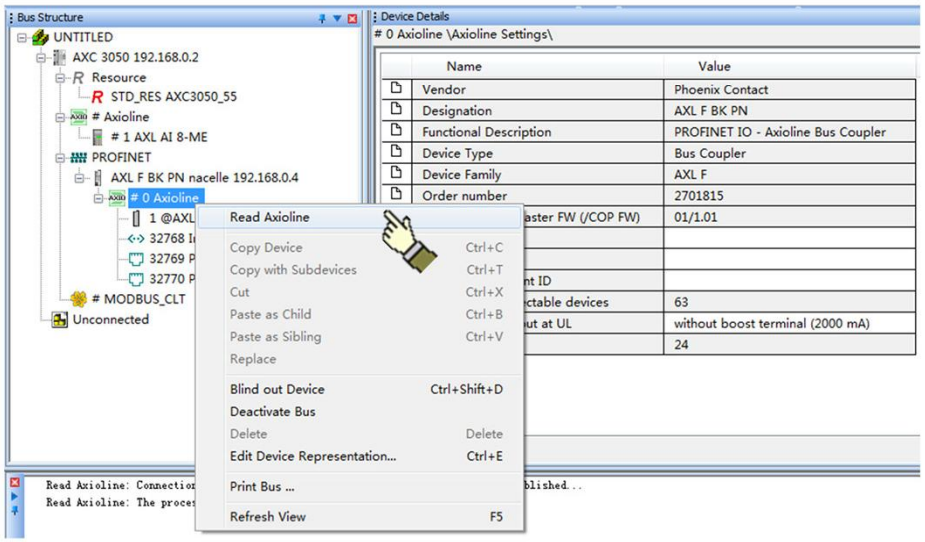
这里所描述的过程通常被称为设备命名。与可变的IP地址不同，在电压复位后，设备名仍将保存在PROFINET I/O设备上。

Axioline 组态

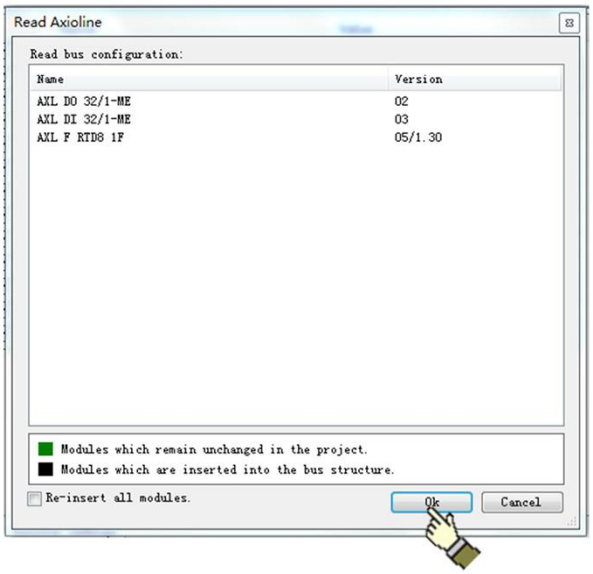
第6讲

在线组态

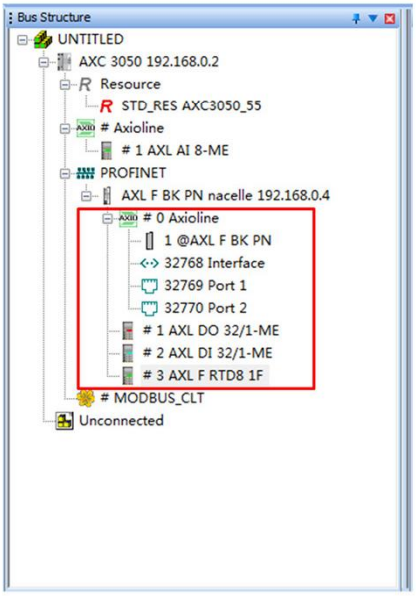
读Axioline


Bus Structure
UNTITLED
AXC 3050 192.168.0.2
Resource
STD_RES AXC3050_55
Axioline
1 AXL AI 8-ME
PROFINET
AXL F BK PN nacelle 192.168.0.4
0 Axioline
MODBUS_CLT
Unconnected
Read Axioline
Copy Device Ctrl+C
Copy with Subdevices Ctrl+T
Cut Ctrl+X
Paste as Child Ctrl+B
Paste as Sibling Ctrl+V
Replace
Blind out Device Ctrl+Shift+D
Deactivate Bus
Delete Delete
Edit Device Representation... Ctrl+E
Print Bus ...
Refresh View F5
Device Details
0 Axioline \Axioline Settings\
Name
Value
Vendor
Phoenix Contact
Designation
AXL F BK PN
Functional Description
PROFINET IO - Axioline Bus Coupler
Device Type
Bus Coupler
Device Family
AXL F
Order number
2701815
01/1.01
63
without boost terminal (2000 mA)
24

读入本地Axioline总线系统

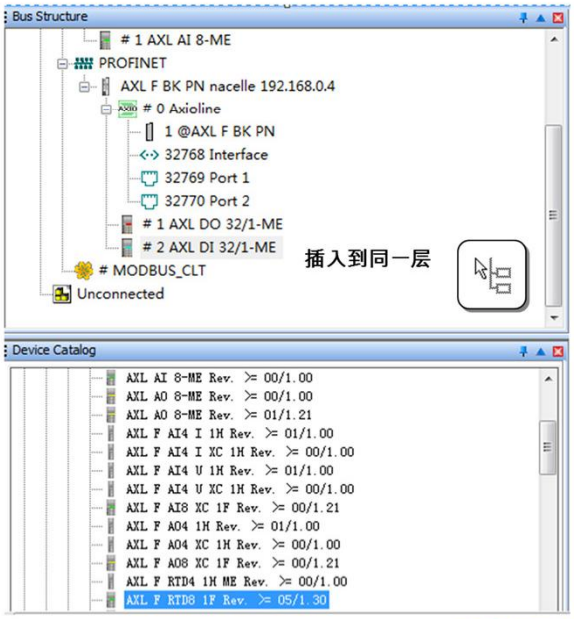


插入Axioline设备至总线结构

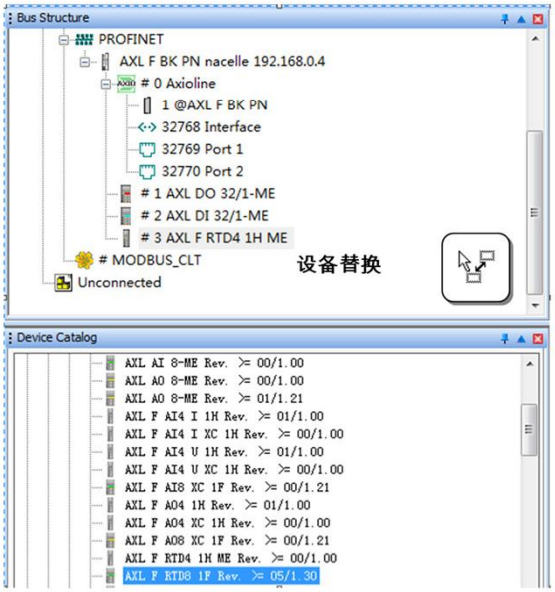


离线组态

从设备类别中插入设备


Bus Structure
1 AXL AI 8-ME
PROFINET
AXL F BK PN nacelle 192.168.0.4
0 Axioline
1 @AXL F BK PN
32768 Interface
32769 Port 1
32770 Port 2
1 AXL DO 32/1-ME
2 AXL DI 32/1-ME
MODBUS_CLT
Unconnected
插入到同一层
Device Catalog
AXL AI 8-ME Rev. >= 00/1.00
AXL AO 8-ME Rev. >= 00/1.00
AXL AO 8-ME Rev. >= 01/1.21
AXL F AI4 I 1H Rev. >= 01/1.00
AXL F AI4 I XC 1H Rev. >= 00/1.00
AXL F AI4 U 1H Rev. >= 01/1.00
AXL F AI4 U XC 1H Rev. >= 00/1.00
AXL F AI8 XC 1F Rev. >= 00/1.21
AXL F AO4 1H Rev. >= 01/1.00
AXL F AO4 XC 1H Rev. >= 00/1.00
AXL F AO8 XC 1F Rev. >= 00/1.21
AXL F RTD4 1M ME Rev. >= 00/1.00
AXL F RTD8 1F Rev. >= 05/1.30

使用设备类别进行设备替换



过程数据作为变量

第7讲

内容

本章节介绍了基于在PC WORX中配置的总线系统过程数据项的过程数据变量（输入和输出）的创建。
第二阶段介绍了如何将程序中有效的全局变量与过程数据项相连接，使得能对总线系统的输入和输出信号进行访问。

对于这两个步骤，需要一个在PC WORX 中配置的总线系统（参见*INTERBUS组态*章节或*PROFINET组态*章节）。

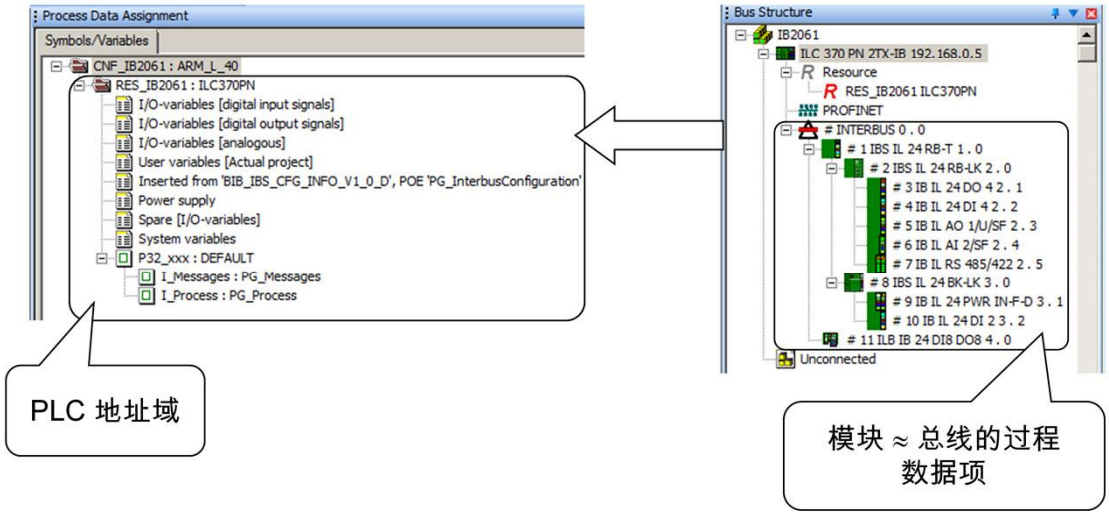
在过程数据分配视图中，通过变量或过程数据的右键快捷菜单，可以重新断开变量和过程数据项之间的连接。但是，在该视图中还不能删除不再需要的变量，只能在相应的全局变量表中删除。

注意!

信息

提示

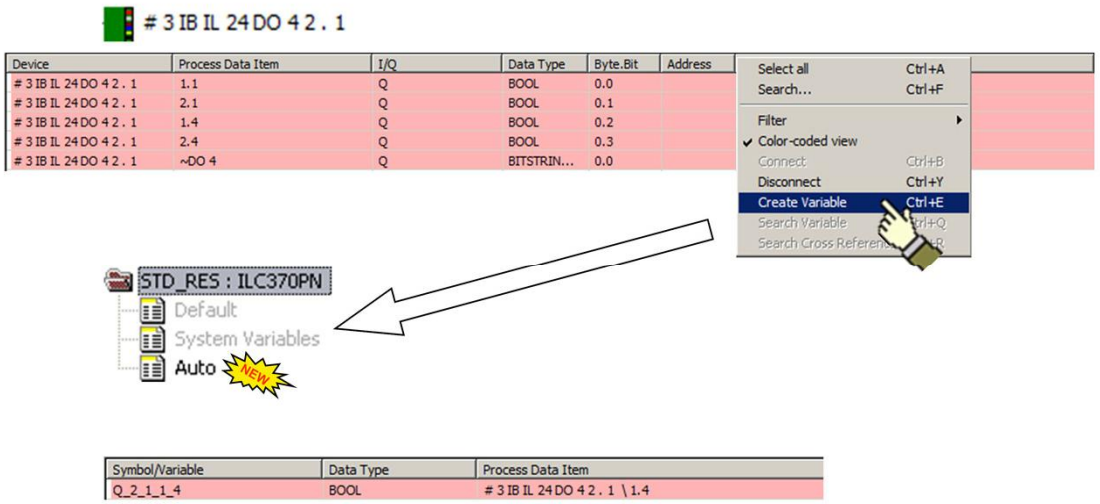
过程数据分配工作区



总线系统（不管是何种系统）通过XML文件提供的过程数据项必须能用于PLC输入信号的采集和输出信号的输出。这意味着从PLC地址域必须能访问到这些对象。采用PC WORX 和其它仅用符号工作系统，这可以通过将变量分配给过程数据项来实现。

在前面与PC WORX界面一起介绍过的PC WORX 过程数据分配工作区，提供了有用的工具来创建基于过程数据项的变量或将已存在的变量与有效的过程数据项相连接。这两种方法将在随后介绍。

创建过程数据变量



要创建过程数据变量，在左上方选择对应于PLC的CPU项。
在基于标准工程模板的工程中，该项的名称是*STD_RES (标准资源)*（PLC的CPU），除非是在工程创建后用户重新对其进行命名。

根据在右上方所选项（总线系统硬件），在右下方则是：

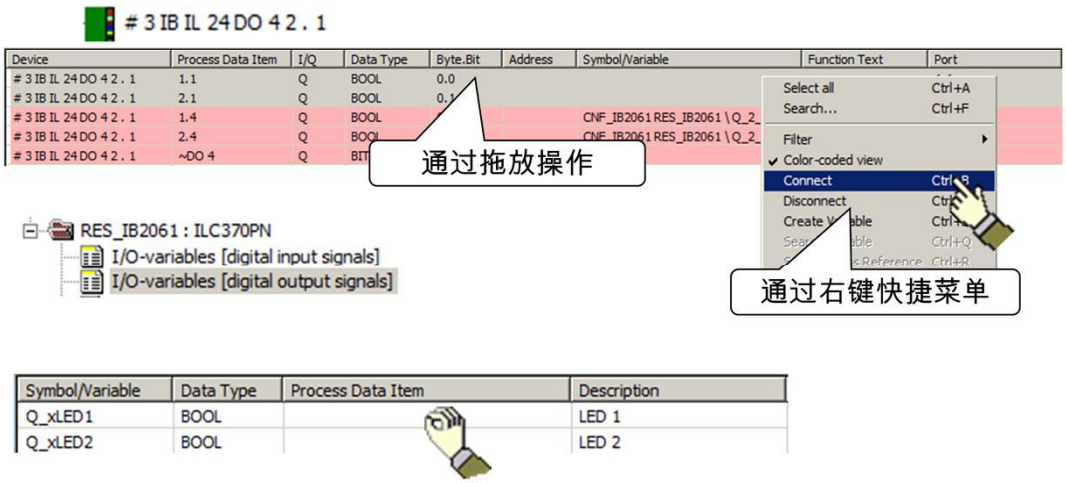
a) 系统文件（ PROFINET, IBS），显示属于总线系统的所有过程数据项；
b) 总线耦合器，显示分配给该总线段的总线模块的所有过程数据项；
c) 单个模块，只显示分配给该模块的过程数据项。

在过滤器中，用户可以选择是否显示标准及/或单个过程数据项，已连接的及/或未连接的过程数据项。
从所选过程数据行的右键快捷菜单中，选择 *Create Variable* （创建变量） 功能创建变量。请确保过程数据项之间不干扰，否则将在工程编译时会出错。所创建的变量总是归入 *Auto* 变量组中，如果 *Auto* 变量组还未创建，则由PC WORX自动创建。

自动创建变量的变量名将根据以下模式进行创建：
<数据方向>_<段>_<位置>_<连接点> (参见以上的例子)

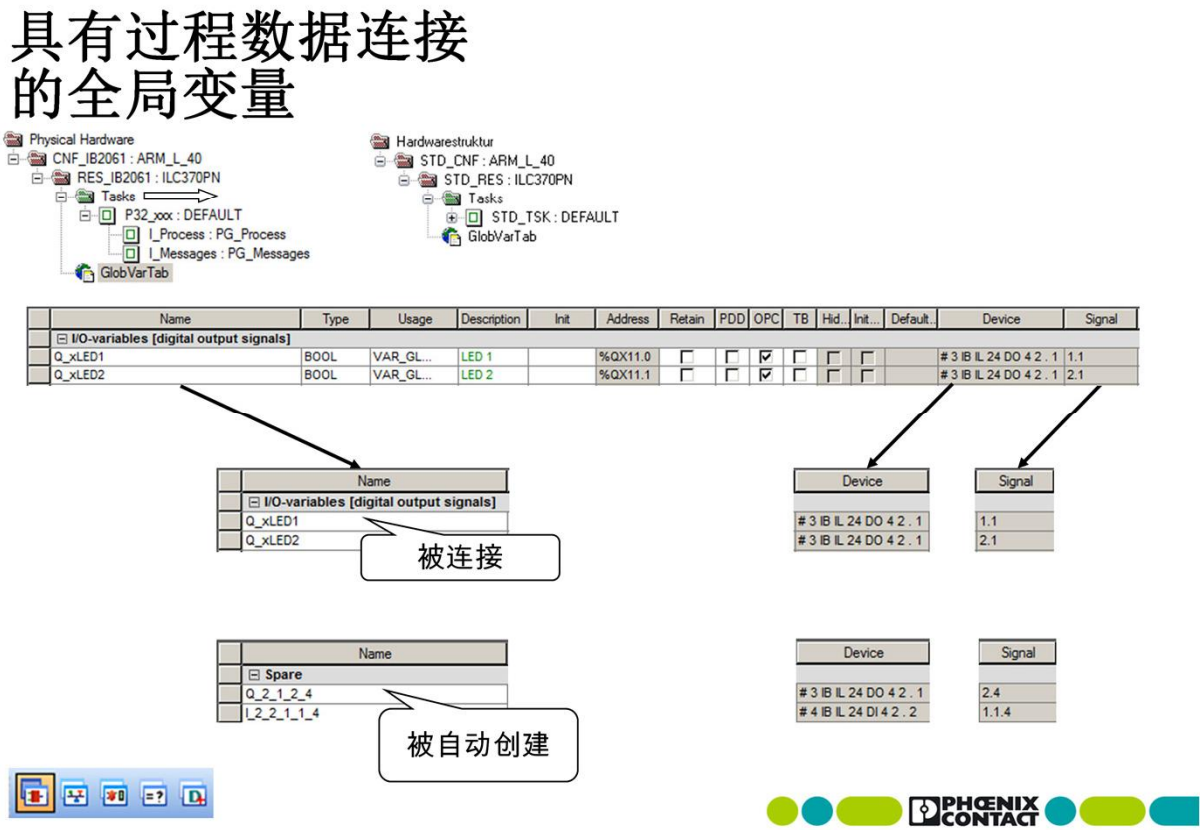
变量名中不允许的字符将被下划线所替代。变量名只能在变量表中修改，而在过程数据分配中则不能修改。

过程数据项与全局变量



欲将过程数据项与已有的全局变量相连接，在左上方选择待连接的变量所在变量组，从右上方选择模块以及待连接的过程数据。
变量和过程数据的右键快捷菜单都提供了“*Connect*（连接）”功能。
另一种方式是，采用拖放的方式建立连接。

如果一个模块的多个过程数据项将与变量相连接，则可以通过拖放方式将多个PD对象与已有的变量相连接，必须顺序正确。



不管选择何种方式访问过程数据变量，其结果都是相同的。只需要根据工程需要修改所创建变量的符号即可。

符合 IEC 61131的软件模型

第8讲

内容

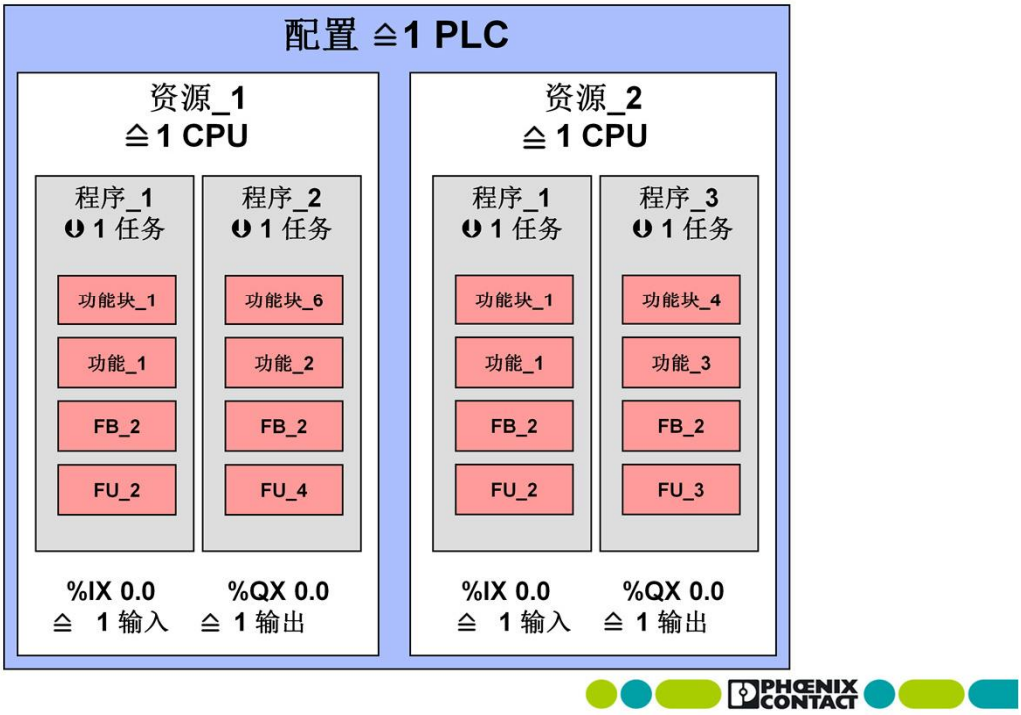
在本章节中，您将学习如何根据IEC 61131对集成的控制系统进行分类，并提出了控制系统的四个性能等级及其可用的处理器和类型。
本章节也介绍了如何根据应用的需求在任务的辅助下对CPU的性能进行组织。

注意!

信息

提示

术语



配置
一个完整的PLC系统的描述（物理和逻辑）。

资源
为执行程序需要的所有特性提供支持，是程序和PLC的物理I/O通道之间的接口。

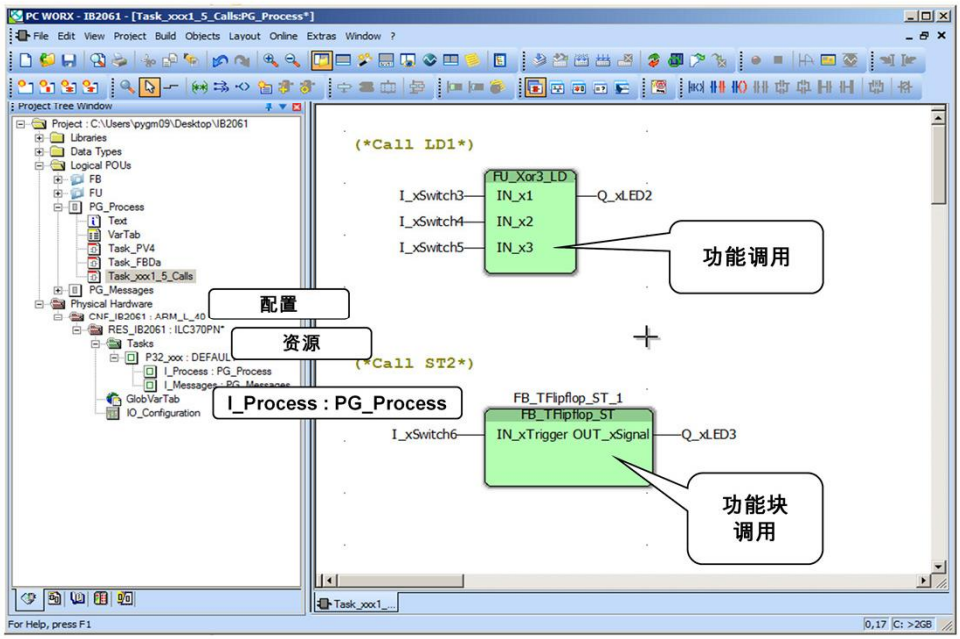
任务
用来控制不同应用部分执行的时间。

程序 (PG)
程序组织单元（POU）被分配给一个任务。

功能块 (FB)
子程序(POU)，包含了静态数据。

功能 (FU)
子程序(POU)，不包含全面数据。

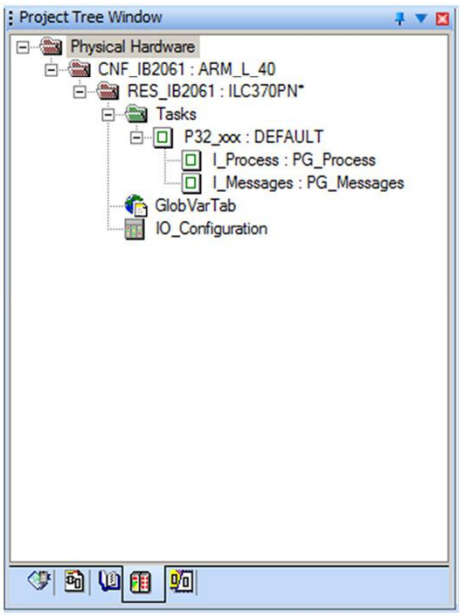
执行



程序*PG_Process*分配给了 实例 *I_Process*。

工作区的左侧显示了符合IEC 61131的工程描述。
工程树的上部分是工程逻辑部分的描述：库，数据类型和逻辑POU。
下部分是物理结构的描述：CPU类型，任务结构，对过程数据的存取。
工作区的右侧详细显示了各元素。

硬件树

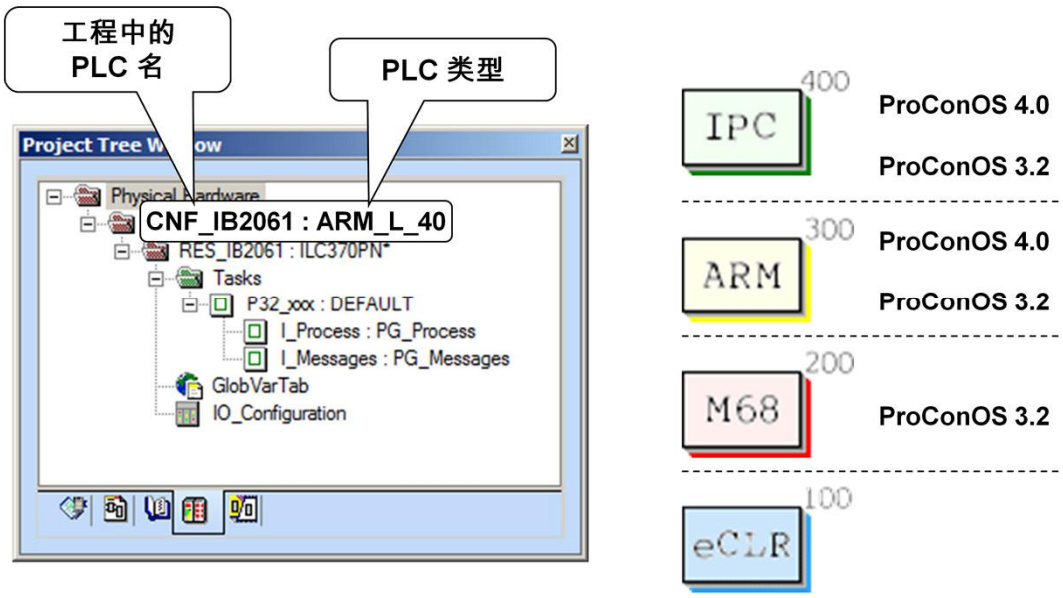


在PC WORX中，PLC硬件结构主要包含PLC机架、PLC处理器和处理器能力的管理。

保存硬件结构的工程树部分可在单独的标签页进行浏览。

配置 • 资源

配置：PLC 性能等级 / 机架



根据IEC 61131，PLC硬件结构中的最高层称为*配置*。对传统的控制系统而言，这对应于PLC机架，用于对性能等级进行划分。

安装后， PC WORX提供了四种[1]不同的性能等级，这些性能等级所使用的处理器有所不同。此外，对两个高性能等级，提供了不同版本的运行时系统。

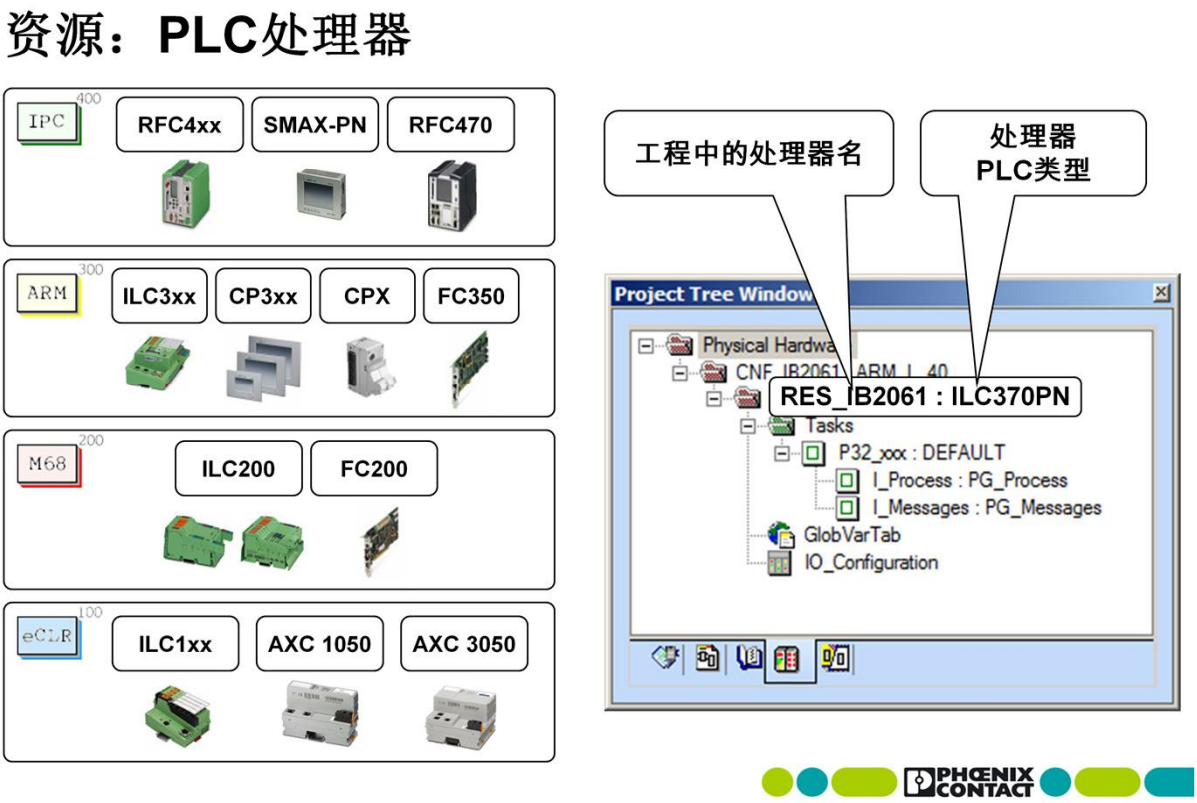
400类采用了INTEL处理器， 300类采用了ARM处理器， 200类控制系统使用了 Motorola处理器，最后的100类控制系统使用了eCLR处理器。

操作系统3.2版本和4.0版本的主要区别是：在PLC操作过程中编辑选项的程度。

由于可以对多个控制系统进行操作，因此，除了控制系统的性能等级外，还必须进行配置名分配。作为标准，PLC工程模板使用了配置名*STD_CNF* (标准配置)。

这里所使用的*配置*这一术语并非与INTERBUS 组态相对应。

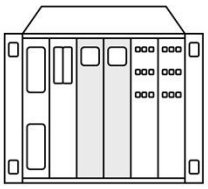
[1] 从 PC WORX 5.10.22起

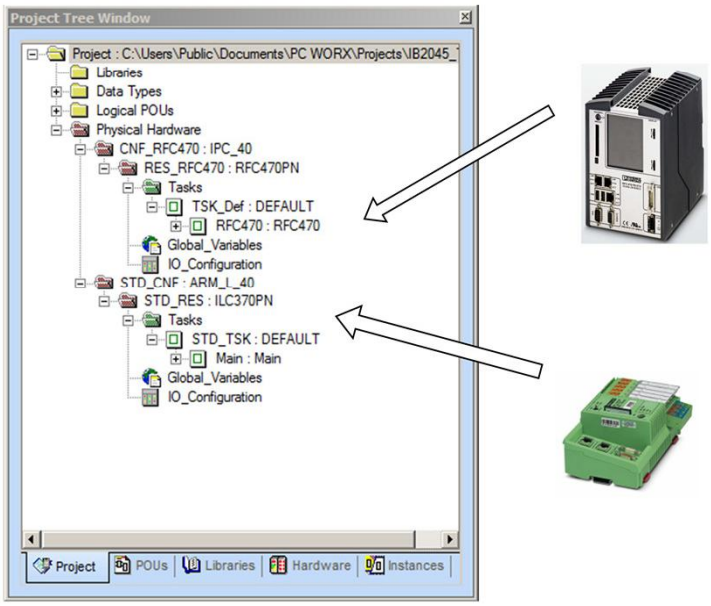


所使用的处理器及其相匹配的控制系统取决于性能等级。上图显示了控制系统与性能等级的对应关系。一些控制系统提供了两个ProConOS版本： 3.2 和 4.0。

与性能等级一样，除了输入处理器类型外，还需要为*资源*分配资源名。作为标准，PLC工程模板使用了资源名STD_RES（ 标准资源） 。

分布式资源

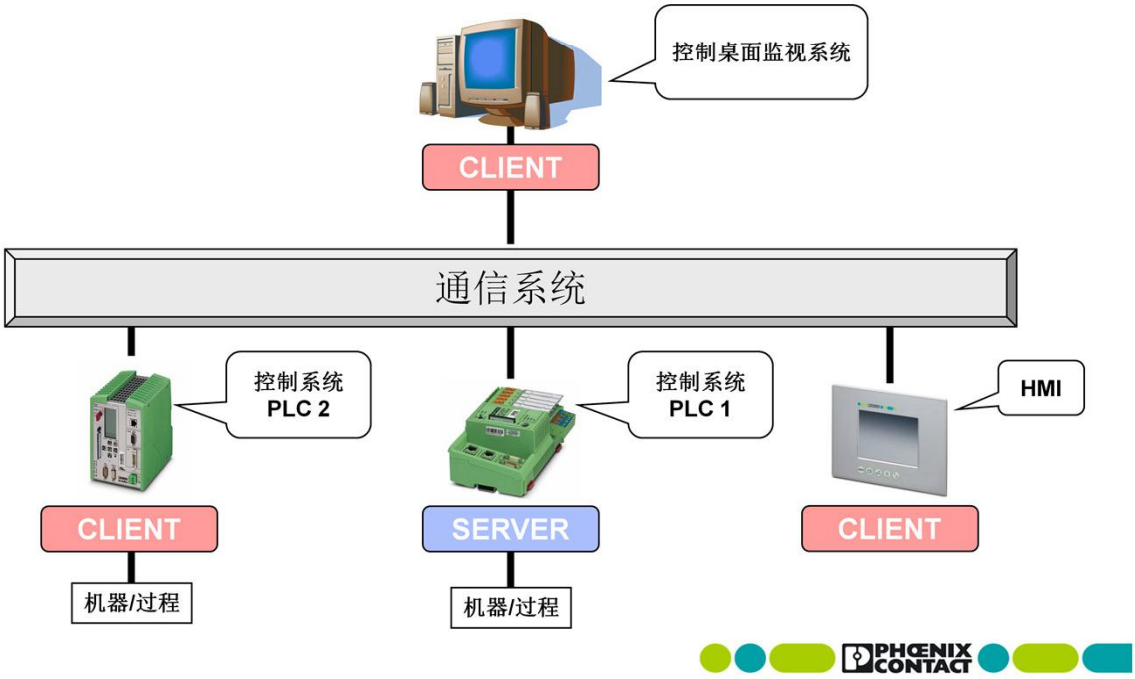




就总线组态中的硬件和PLC而言， PC WORX 支持多个系统。但在接下来的部分，只就每个工程一个控制系统展开讨论。

在工程树上部分所创建的程序可被分配给不同的、分布式配置及控制系统。控制系统虽使用相同的算法，但单独进行处理。

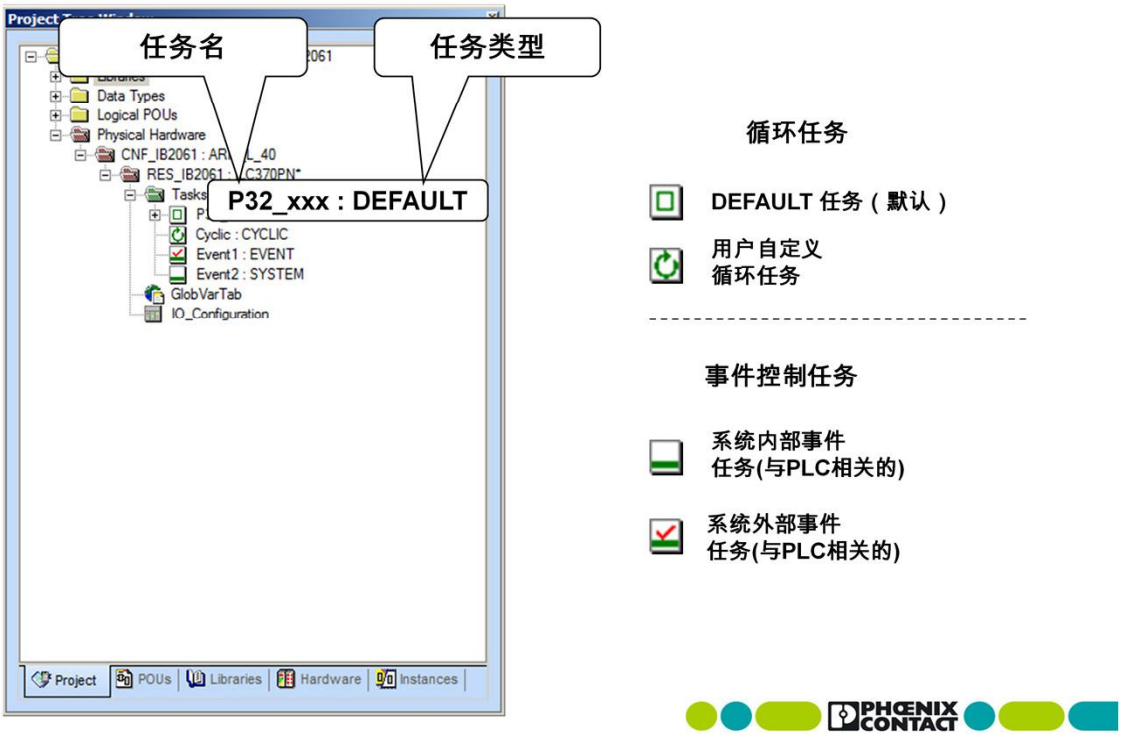
符合IEC 61131-5的通信模型



如果配置了多个控制系统，可通过标准化模块进行数据交换。在IEC 61131第5部分，通信模型描述了多个自动化工程设备之间的数据通信。它对应于通常所说的客户机/服务器体系结构，这将在PC WORX 6 通信基础课程中展开。

任务

任务： 处理器管理



在PC WORX中，正是因为有了任务管理，处理器能力才能够得到很好的使用，并转化为应用。

任务可以划分为两类：循环任务和事件控制任务。

PLC中的过程是以循环执行为特征的。DEFAULT（默认）任务作为一个循环任务可用于大多数应用中。其具有最低优先级，最小的循环时间为tmin=2*系统时钟。对IPC和ARM处理器而言，系统时钟设为1ms， M68处理器设为5ms。 DEFAULT任务只能创建一次，由于其优先级低，它可被其它任何任务所中断。
与DEFAULT任务相对比，用户自定义任务(循环任务) 可以在技术条件允许的条件内自由选择优先级和循环时间。优先级设在*0* (最高) 和*31* (最低)之间。

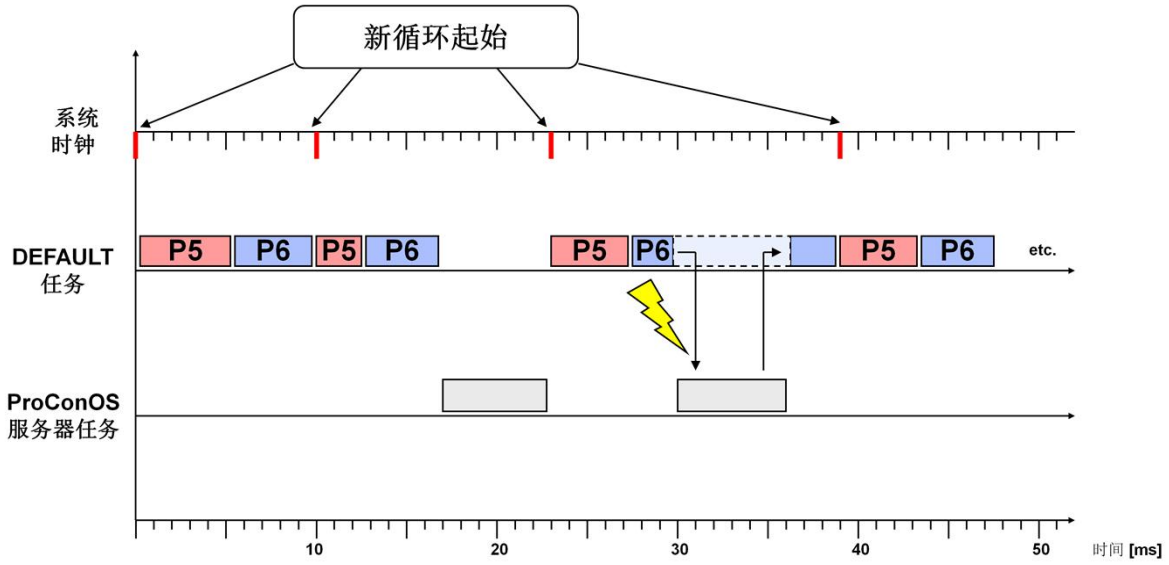
第二类任务涵盖了系统和事件任务。系统任务能对PLC内部事件进行响应（如：冷启动，热启动，除以0等）。这包括了控制系统的常规的和非常规的运行状态。
这取决于所使用的控制系统，其外部事件可由事件任务得到。一个例子是直接输入，其状态的变化定义为ILC的一个事件。当连接INTERBUS时一个周期的结束就是一个事件。

事件任务和系统任务的事件可在PC WORX 对话框中通过文本选择进行设置。

在任务中不执行程序。根据配置，任务触发程序实例（参见*硬件体系结构*章节）。

通过分配与特性相对应的名称，确保硬件结构清晰。过程数据预处理任务名必须是PDP。

PLC 定时:
DEFAULT 任务



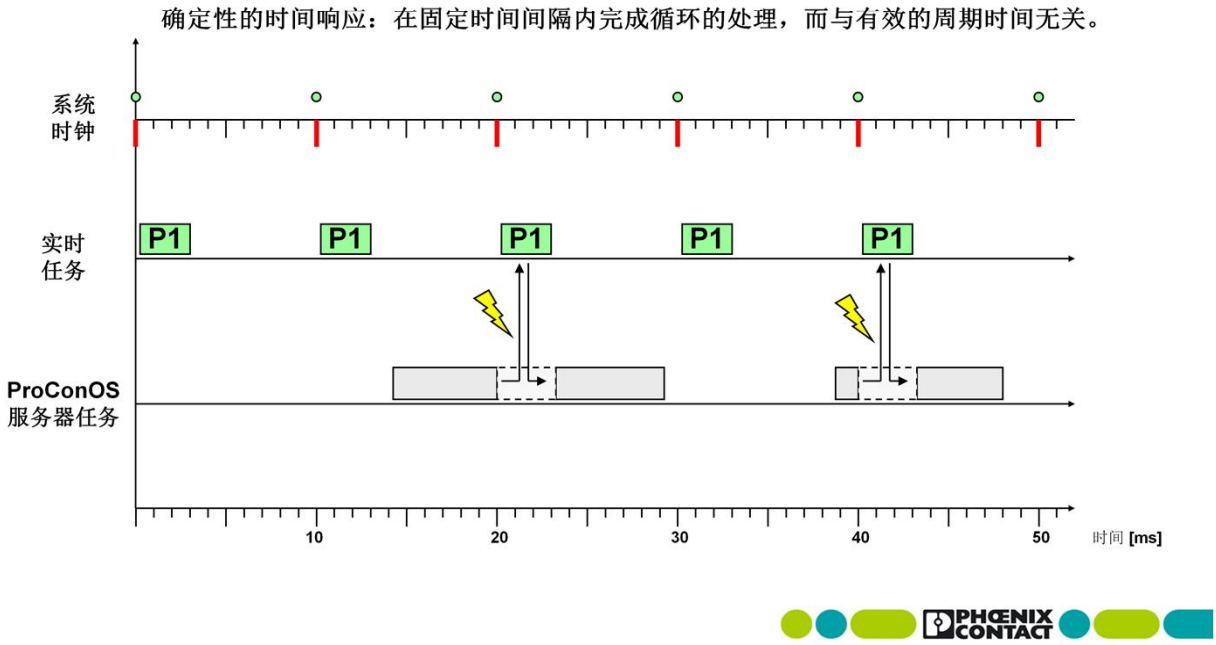
DEFAULT任务的定时与大多传统控制系统的时间响应相对应：在一个循环执行后，直接启动下一个循环，它们之间无固定间隔。由于程序的原因，循环时间可能波动很大。由服务器任务（调试或通信）产生的默认任务的中断也会造成循环时间的波动。

DEFAULT 任务：在每个循环后（ P5 和 P6代码），立即启动下一个循环。

看门狗错误：代码不能在t=500ms（默认设置—可以更改） 内执行。

调试会影响程序执行的时间响应。

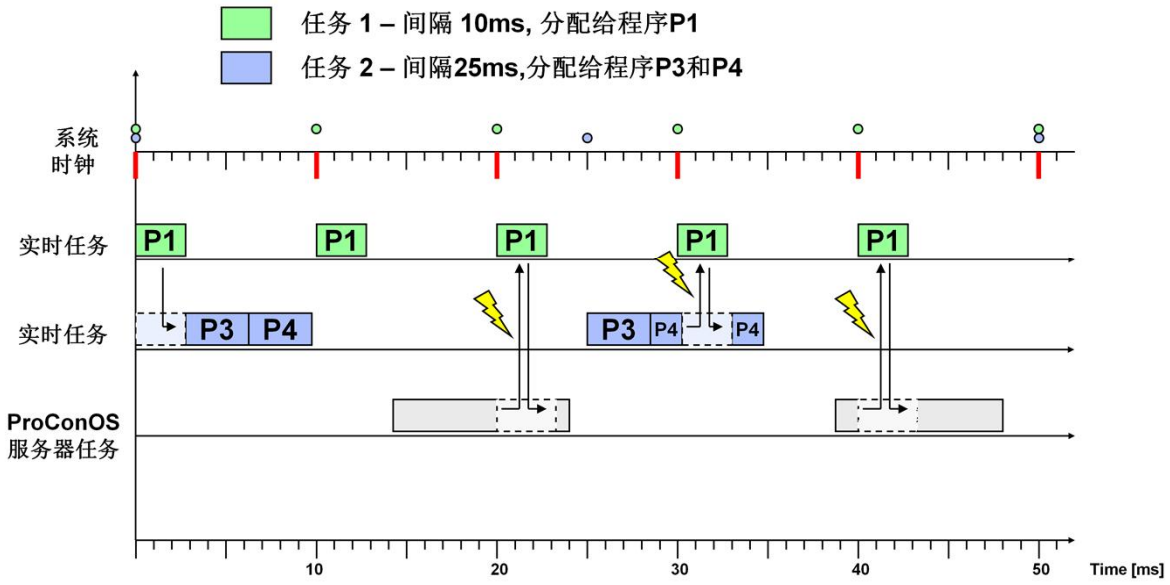
PLC 定时:
实时任务



实时任务是以时间确定性为特征的。在固定的时间内分配给任务的代码(P1) 仅执行一次。因此，可计算出响应时间。只有当实时任务不需要计算时间时，才激活服务器任务。

时间监控（看门狗）确保了可计算响应的执行，即使是所分配的代码不能在时间间隔中完成（例如，由于代码含有过多的处理器命令）。要么是控制系统切换至停止操作状态，要么执行了用户自定义响应。

PLC定时:
多个实时任务



程序组织单元

第9讲

内容

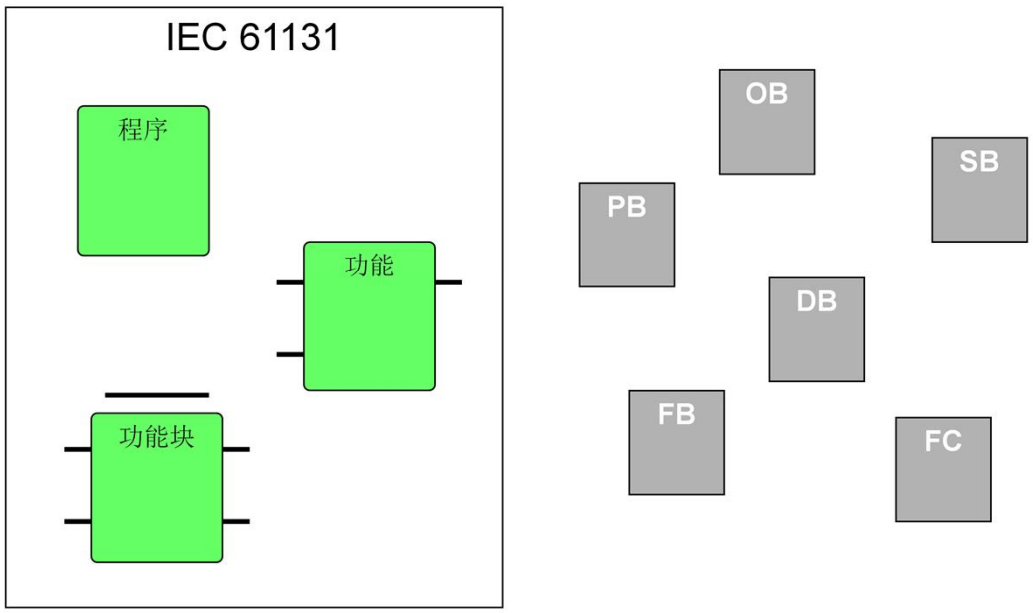
本章节介绍了工程中软件模块，根据IEC 61131，它们被称为程序组织单元（POU）。
本章节还介绍了三种类型的POU的特征，在工程中如何进行模块调用，以及各个模块之间如何进行数据交换。


注意!


信息


提示

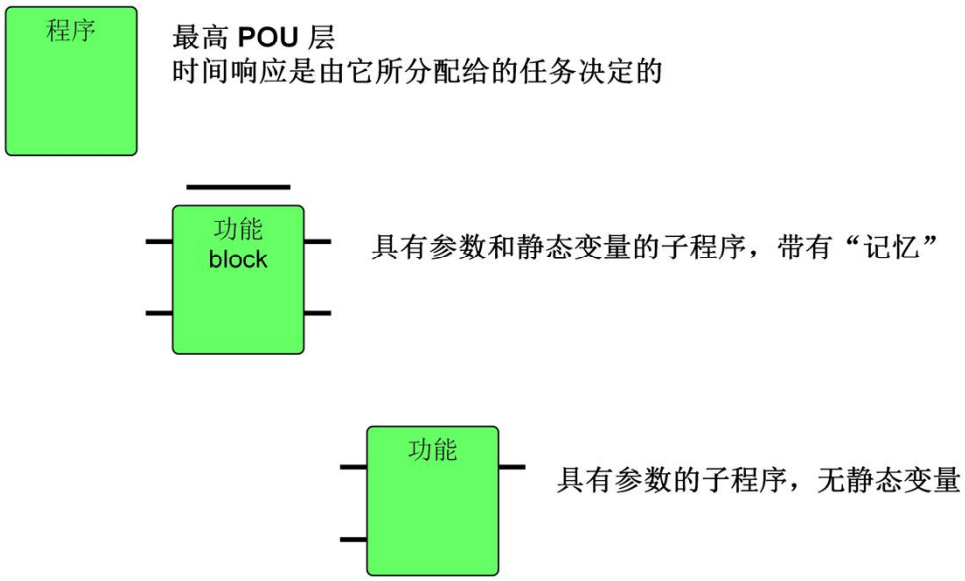
POU 类型 1



根据IEC 61131，定义了三种不同的POU类型：程序，功能块和功能。通常，编程可以是命令无结构化的排列。但是，这三类POU的使用使工程中的清晰度得到很大程度的提高，更重要的是在编辑阶段提供了有效编程的能力。

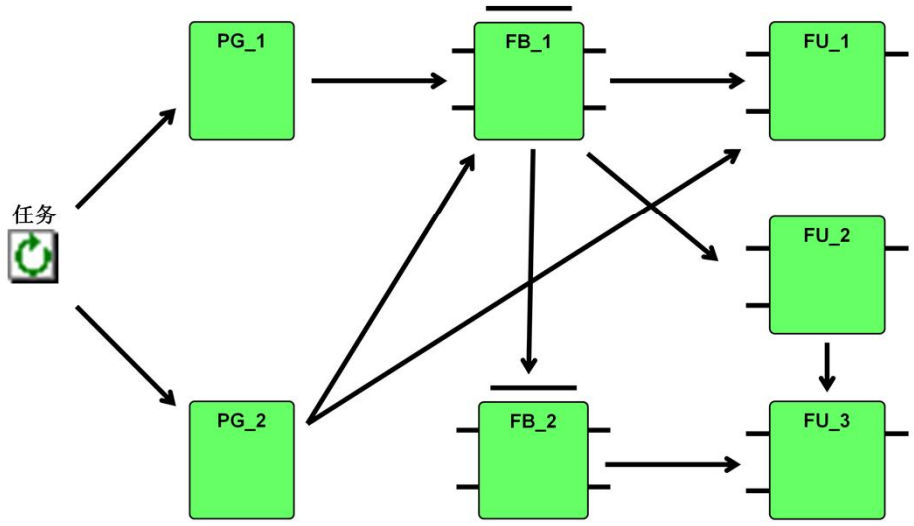
下面将介绍各个POU类型的特征及其应用领域，以及所配置的POU之间如何进行数据交换。

POU 层次结构



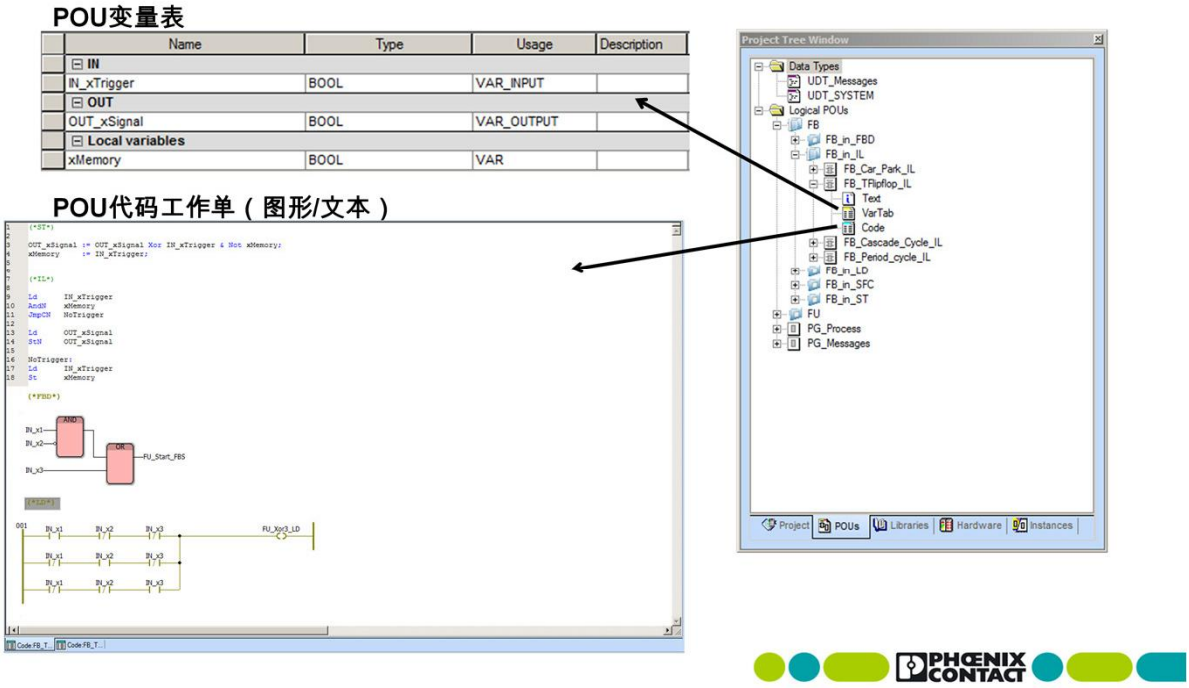
概要显示了三种POU 的层次结构及其基本特性。

POU 调用结构



调用结构是层次化的。
POU调用的复杂程度取决于存储堆栈。
安全起见，不允许递归调用。

POU 元素



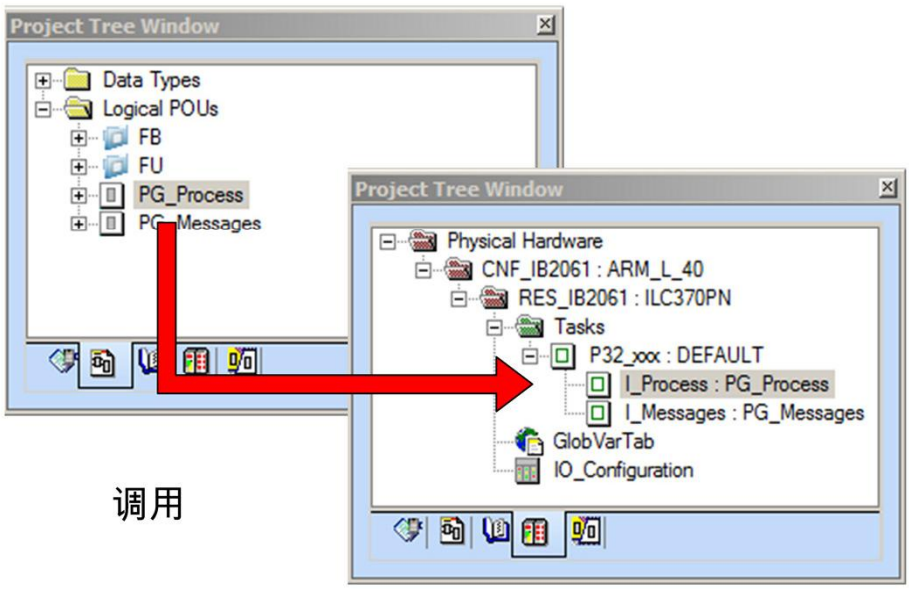
每一个POU（不管是何种形式）都提供了一个或多个注释工作表，供程序员保存执行功能描述以及版本信息。

仅有一个局部变量表可供保存POU所需的变量。

一个或多个图形或文本代码工作表可供程序员完成所期望的算法。

程序

程序的使用



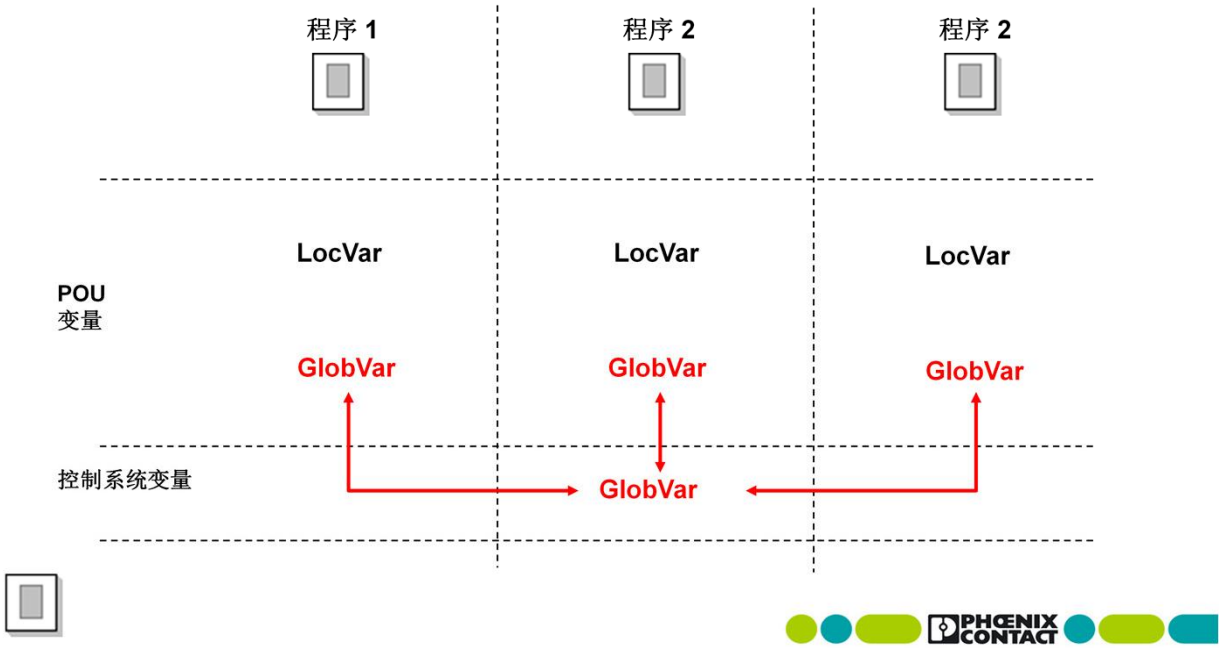
调用

程序是最高层的POU，在一个工程中，至少需要一个这一类型的POU。
程序以及其它的POU保存在逻辑POU文件夹下，在此处，对自动化工程无影响。只有当程序以程序实例的形式在任务中使用时，保存在程序中的代码才会被PLC执行。

程序实例是CPU的存储区域，根据任务的定义，执行作为程序的代码。
通过将程序实例分配给任务来释放CPU程序以供执行，称之为程序实例化（与功能块实例化相对照）。

PC WORX支持程序的多重实例化。由此引申出来的内容将在*PC WORX IEC 61131 通信*课程中展开。

POE类型 *程序*
数据交换



程序的一个主要任务是将硬件信号与程序相链接。

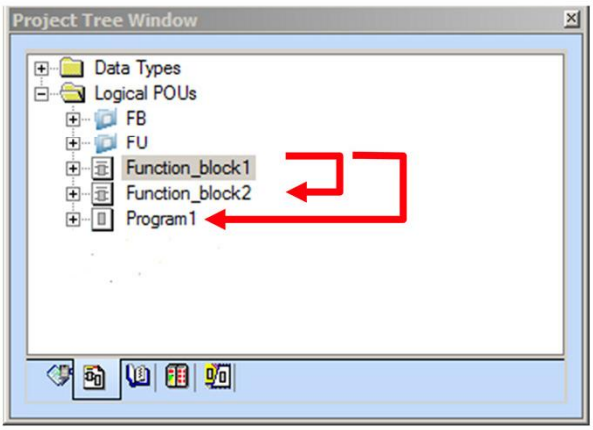
全局变量是建立链接的唯一途径。它们用于交换关于控制系统的全局信息。

对于程序，全局变量是唯一可用来与其它的程序单元进行数据交换的方式。

与其它类型POU的一样，只要标记的值对整个工程并不重要，就被声明为局部变量。

功能块

POE类型 *功能块*



调用

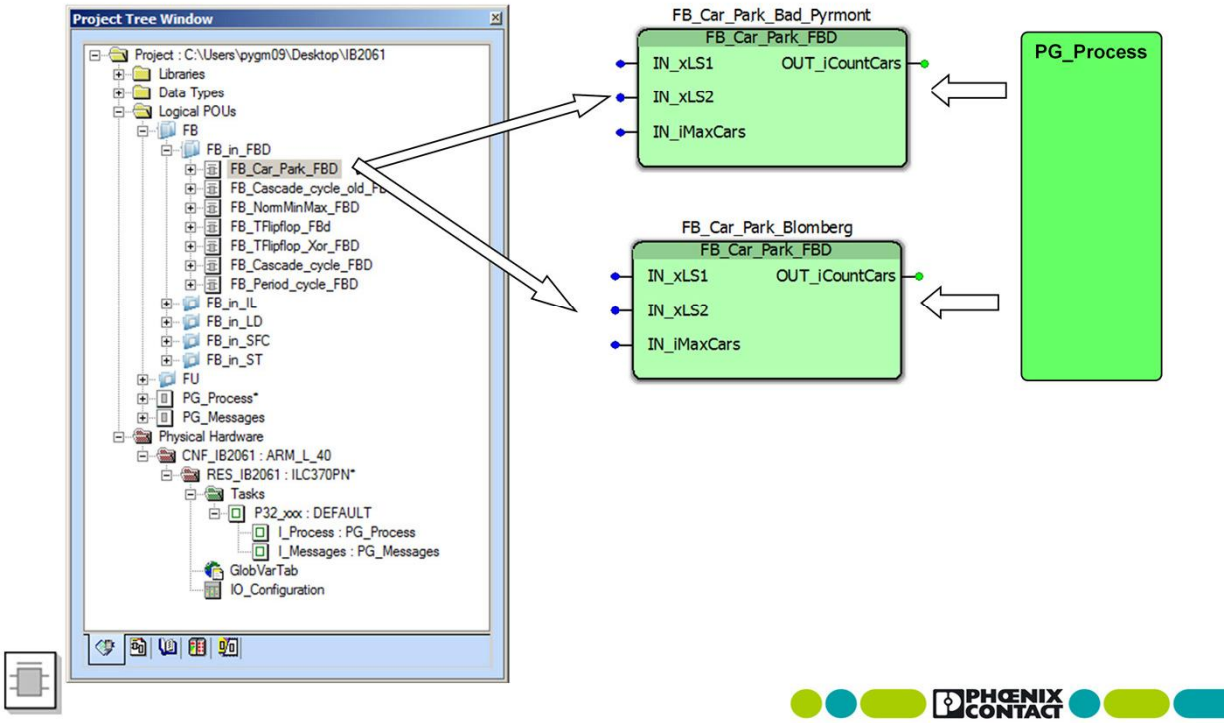
作为参数化单元，功能块可在程序及其它功能块中进行调用。

功能块（功能）的创建基本上针对两件事。第一，一旦程序代码创建完毕，它可以简单反复使用。第二，参数化POU也提供了封装的优点。这意味着(例如，用基于文本语言) 创建了一个复杂的程序。如果调用只包含很少几个输入和输出值，则可以用图形语言来实现，这可以提供更高的清晰度。

如屏幕截图所示，功能块可在其它功能块及程序中被调用，递归调用则不允许。

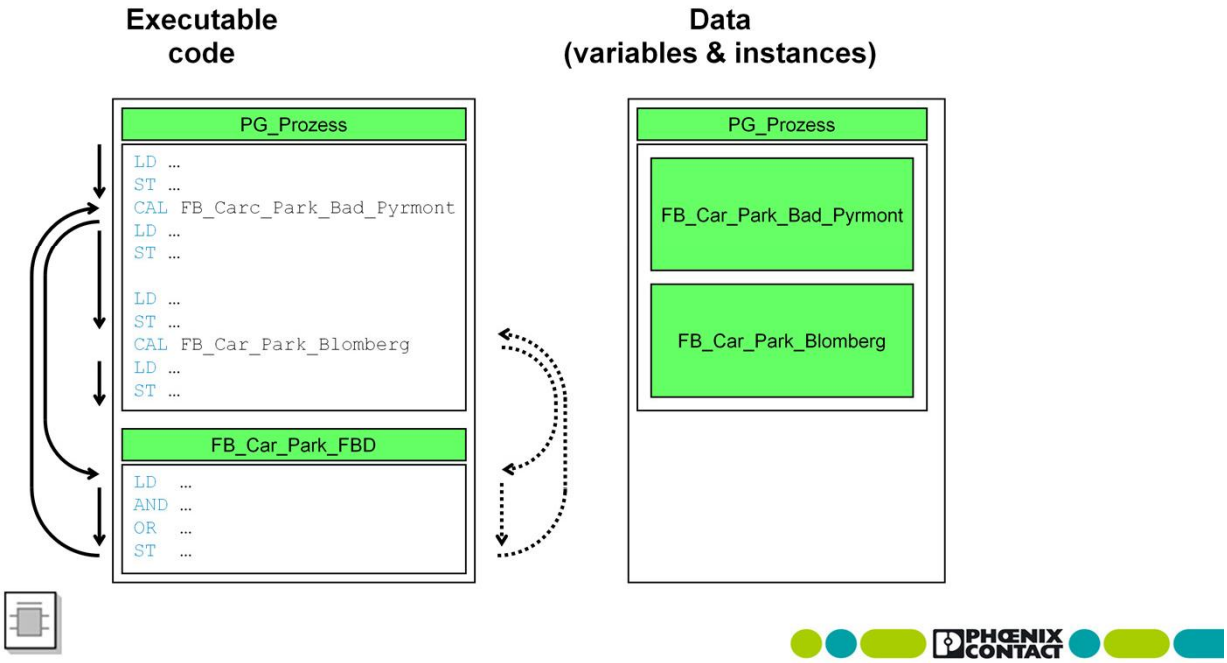
由于功能块在多个程序循环中可以保持（记忆）值，对功能块的每一次调用，在控制系统上则必须提供单独的存储器。该存储器称为功能块实例（与程序实例相对比），它可以比作一个复杂的变量。功能块实例在调用功能块的POU变量表中列出。

功能块的实例化



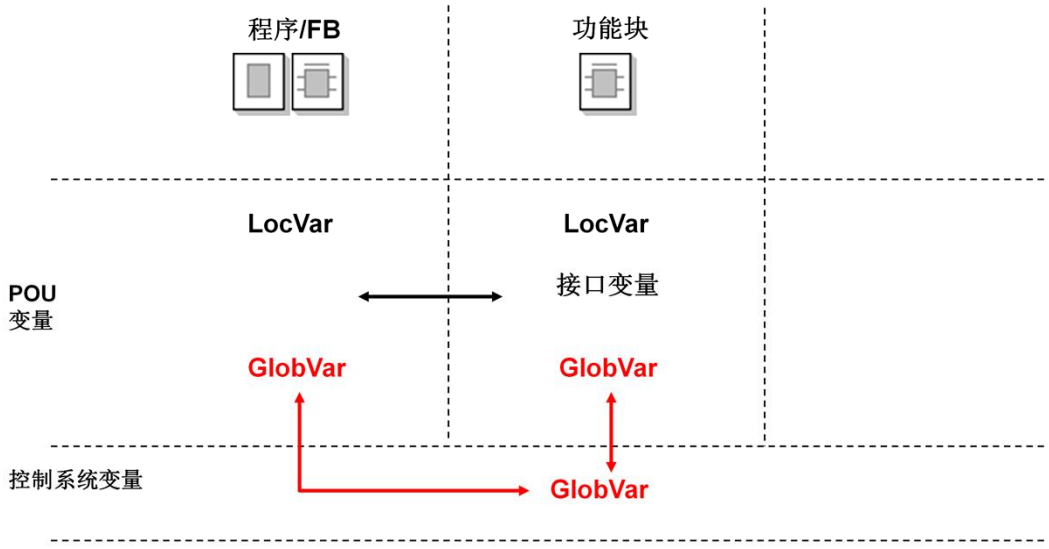
相同的算法类型(FB_Counter)被调用二次。程序PG_Process为不同的任务使用Car_Park功能块两次。为这两个任务提供了单独的存储器（分别命名为FB_Car_Park_Bad_Pyrmont 和 FB_Car_Park_Blomberg）。

FB实例：代码和数据



以上显示了FB结构在不同、不连续存储器上的两次执行。

POE类型 *功能块*：数据交换

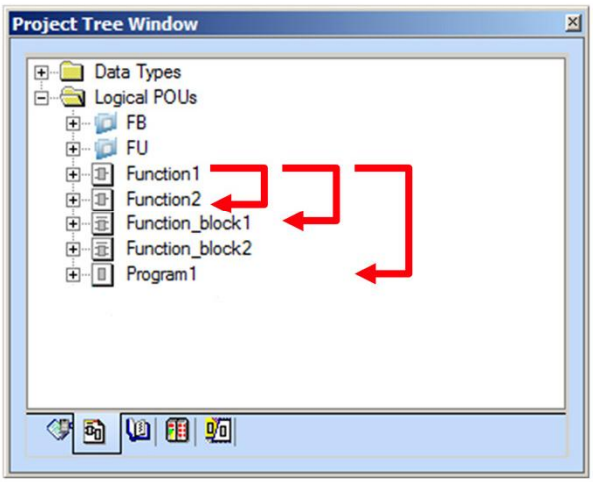


就数据交换而言，功能块提供了最大的灵活性。根据传统的方法，数据交换只能通过输入和输出参数（接口变量）来完成。

但是，对于特殊的应用，通过“后门”，即通过全局变量与功能块进行数据交换将更有意义。对程序员而言，这种方法的缺点在于在调用过程中就被处理的数据而言无直接透明性。

功能

功能的使用



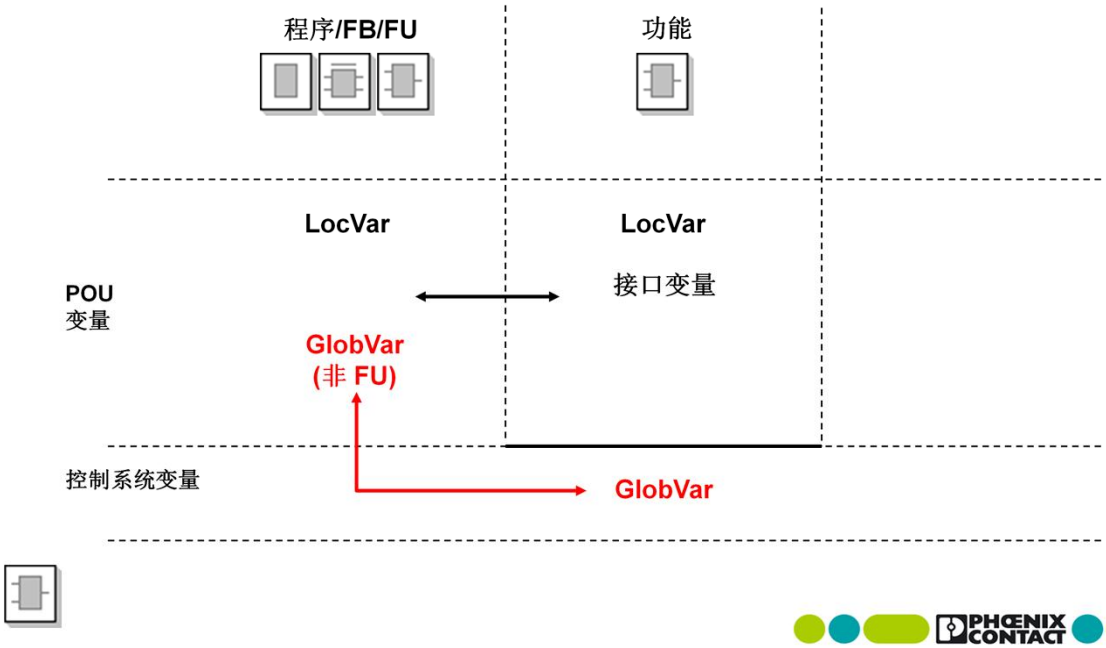
调用

功能是第二类参数化的POU。如图所示，它们可用于其它的功能、功能块和程序中。与功能块一样，不允许有递归调用。

与功能块不同的是，功能有一定的限制，它们不能在多个PLC循环中保存数据。如果一个功能模块正好必须实现这一功能，则就不能用功能来实现。由于功能只能返回一个简单的结果，因此，如果功能要持有多个输出参数也是不行的。

与功能块相比，功能的优越性在于功能无需实例化。对于多次使用，这意味着在执行时反复使用相同的存储器。

POE类型 *功能*：数据交换



至于数据交换，功能可以清晰地定义。功能必须至少提供一个输入参数， 并严格返回一个结果。由于在工程树中创建了功能，因此，在内部，功能的返回值必须保存在已有的参数中（参数名=功能名）。

功能不能访问全局变量。

数据管理

第10讲

内容

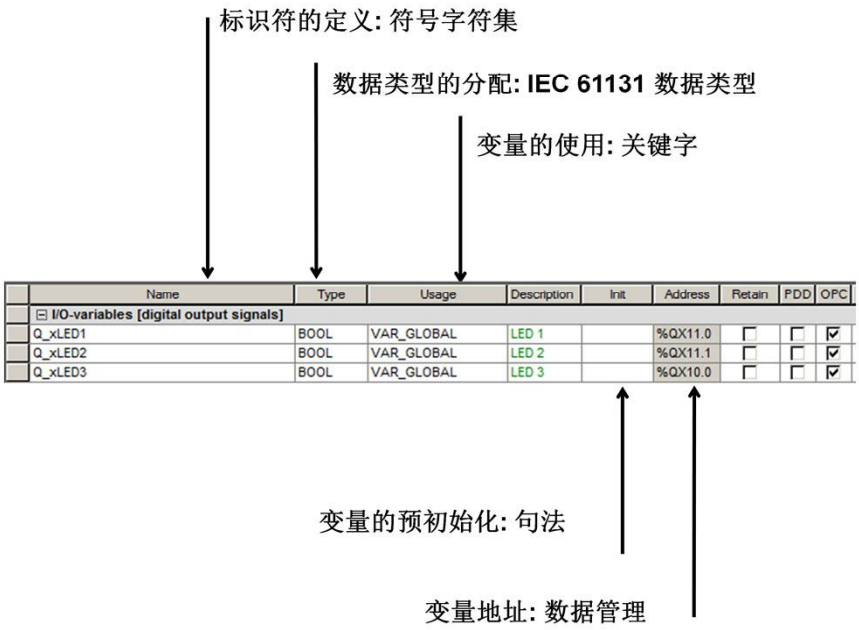
本章节介绍了确定一个变量的参数和以及可在符合IEC 61131的程序组织单元中使用的变量应　用。此外，还介绍了局部变量，局部使用的全局变量和全局参考变量之间的连接，以及在功能和功能块中可用的应用。
变量组和变量对话框可用于管理和组织。

Attention!

Information

Tip

变量声明原则



根据IEC 61131，所有的大小写字母（非变元音或ß），所有的数字以及作为分隔符或首个符号的下划线，都可以用作变量名[1]的组成元素。

一个重要的变量参数是数据类型。在PC WORX安装后，可以使用IEC 61131 定义的多种数据类型，用户自定义数据类型在声明后也可以使用。

变量的使用显示了其与PLC相关的有效性，以及它所具有哪些功能可用于POU与其它POU及PLC之间的数据交换。

变量的描述可选，并可包含所有字符。

控制系统上的物理地址给出了变量与外围设备的交换点。为此， IEC 61131 定义了一个固定的句法：

抽象句法：
%<数据方向><能力前缀><字节偏移量[.位的位置]>
例如： %QX4.0 (输出位 4.0)

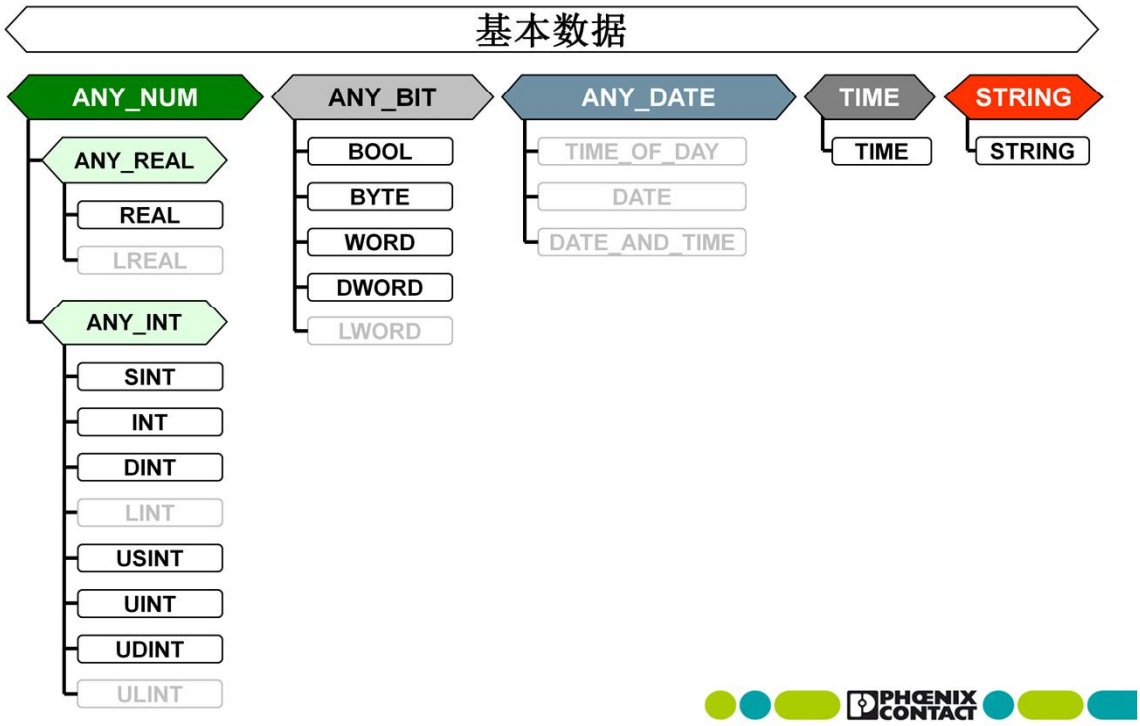
变量可以预初始化为一个值（标准为0）
（句法请参考常量声明）。

[1] PC WORX可以使用DIN命名规则。当采用它时，以结构化文本编写的POU的编程必须是唯一的。

Extras → Options → sheet *Code → Enable DIN-identifiers（ 附加→ 选项 → 表代码 → 启用DIN-标识符）*菜单

IEC 61131中的基本数据类型

数据类型的层次



IEC 61131定义的大部分数据类型在PC WORX中都作为标准提供使用。这些数据类型被划分 组。某些功能和功能块（带有过多的输入和输出参数）不一定要具备某数据类型的变量，而只要具备一组中的一种数据类型的变量（例如：使用ANY_BIT的布尔逻辑AND, OR, XOR, 和NOT 功能块）。

数字数据类型

关键字	数据类型	大小 [位]	值的范围
SINT	8 位整数 (带符号位)	8	-128..127
INT	整数(带符号位)	16	-32768..32767
DINT	双整数 (带符号位)	32	-2.147.483.648..2.147.483.647
LINT	64位整数 (带符号位)	64	-9.223.372.036.854.775.808.. 9.223.372.036.854.775.807
USINT	8 位整数 (无符号位)	8	0..255
UINT	整数(无符号位)	16	0..65.535
UDINT	双整数 (无符号位)	32	0..4.294.967.295
ULINT	64 位整数 (无符号位)	64	0..18.446.744.073.709.551.615
REAL	32-位浮点数	32	+/- 1.5*10^-45 .. +/- 3.4*10^38
LREAL	64-位浮点数	64	

基于位的数据类型

关键字	数据类型	大小 [位]	值的范围
BOOL	布尔	1	$0..1_{HEX}$ (可也以是 false / true)
BYTE	位串8	8	$0..FF_{HEX}$
WORD	位串16	16	$0..FFFF_{HEX}$
DWORD	位串32	32	$0..FFFF\ FFFF_{HEX}$
LWORD	位串64	64	$0..FFFF\ FFFF\ FFFF\ FFFF_{HEX}$

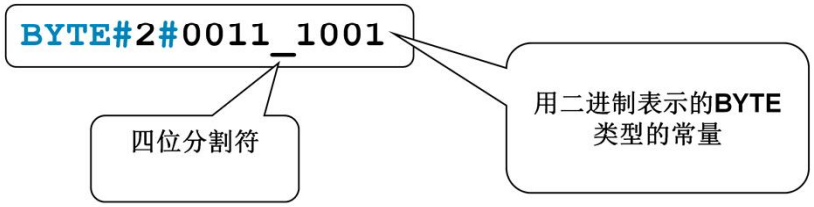
扩展数据类型

关键字	数据类型	大小 [位]	值的范围
TIME	持续时间	32	0..4.294.967.295ms (对照 UDINT)
DATE	日期		
TOD	一天中的时间		
DT	日期和时间		
STRING	字符串 (标准)	80 [字节]	

常量 – 字面值

常量的语法

<数据类型>#<基本>#<常量值><单位>

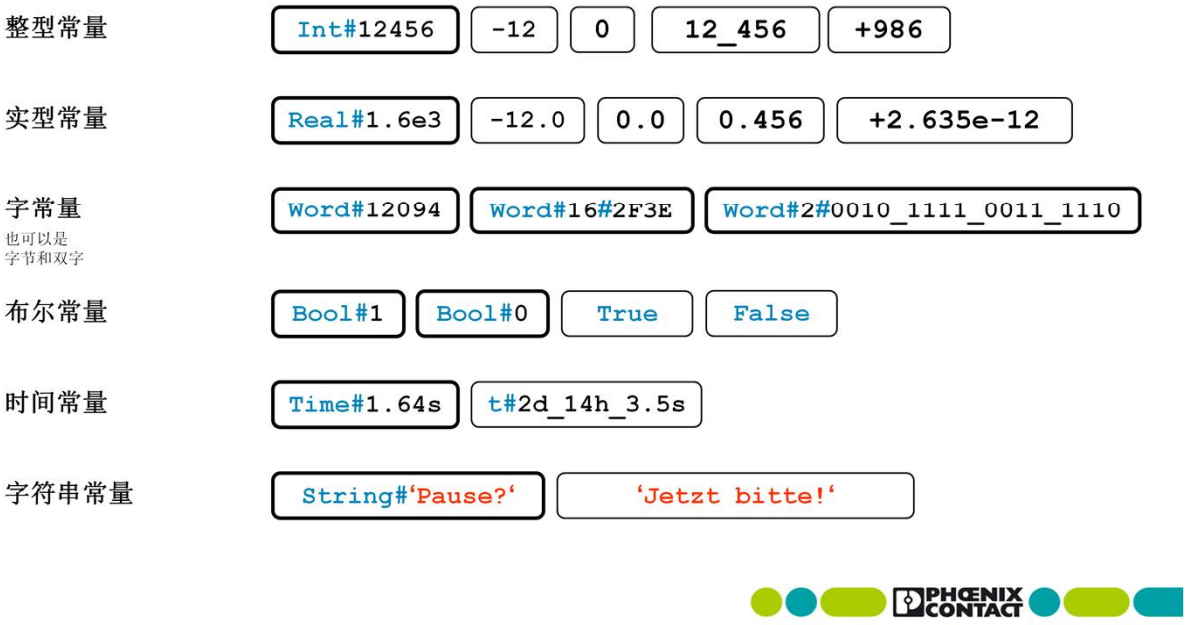


常量声明的正确的形式句法（根据IEC 61131，常量的正式名是“字面值”）如下：

•数据类型
•基本 (如果 不等于 *10*　　*2* 表示二进制
8 表示八进制
16 表示十六进制编码常量)
• 常量值
• 单位 (只是TIME数据类型)，如：*s*, *ms*

如例子所示，下划线可增强其可读性，并不对常量值产生影响。

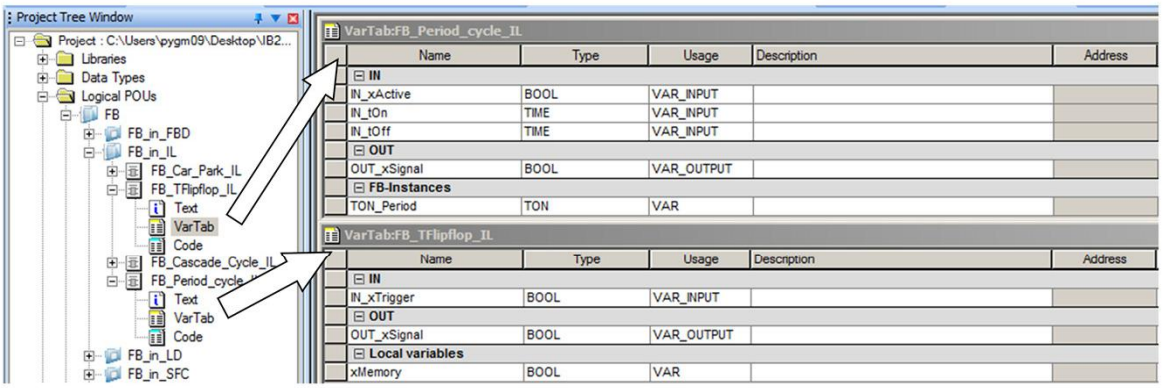
常量表达示例



本页显示了各种形式的常量，请注意对于一些数据类型而言，允许使用简化句法，这常见于编程中。缩写形式（例如：整型数据类型的），对整型组中的其它成员并不允许。
只有在特殊的编程范围内，才会发现是布尔量*1* 和 *0*，而不是*True* 和 *False*。

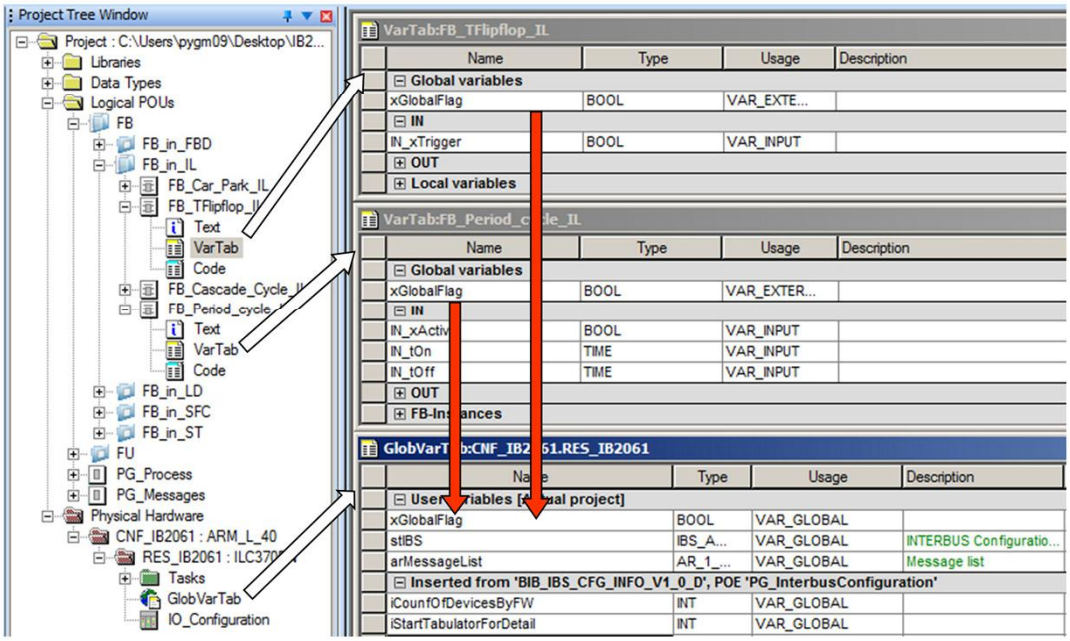
变量的使用

局部数据

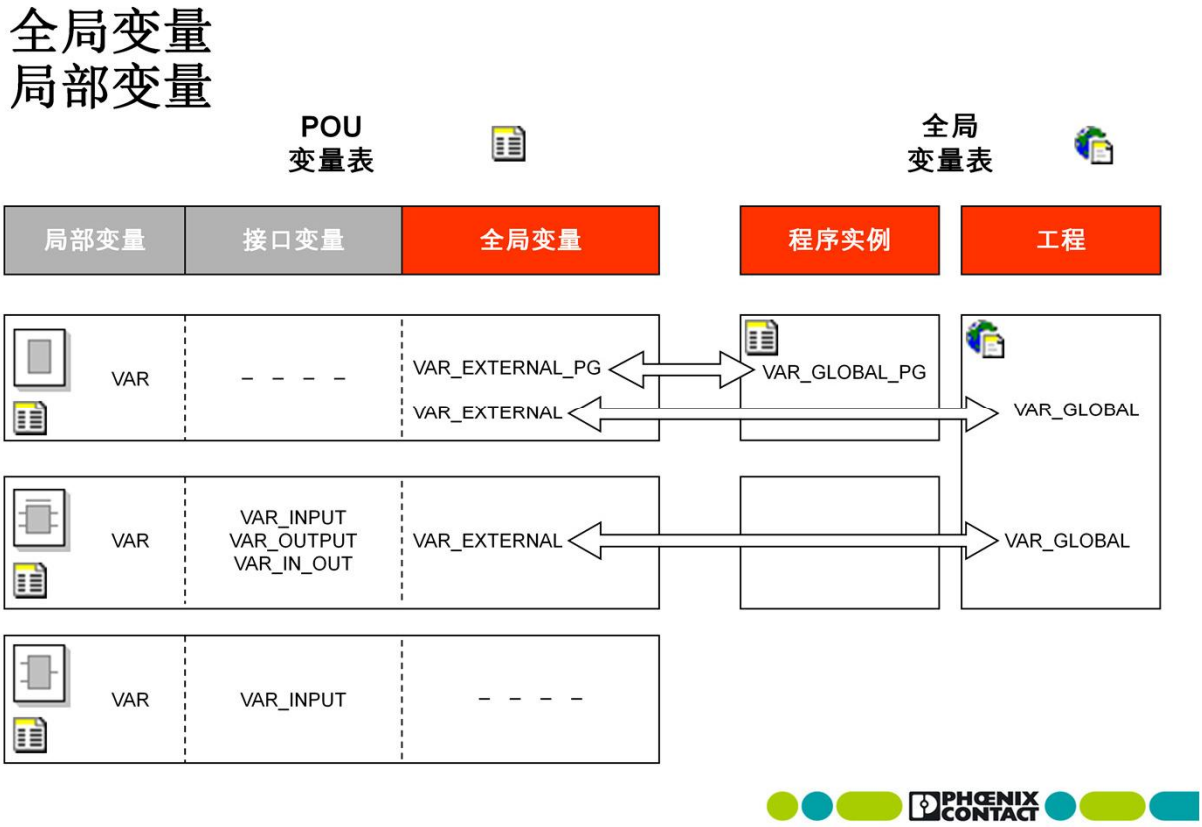


IEC 61131中的数据封装与高级语言的组织示例一致。使用此方法，具有相同名的两个变量可以在两个不同的POU中处理，因而完全独立。

全局数据



全局数据集中存放在全局变量表中，数据保存仅在此处完成，局部使用这种变量将引用该全局参考。



到现在为止，本文档都限于POU区域局部变量和全局变量之间的区别。
归纳起来，全局参考变量的名称为VAR_GLOBAL，而在编程中使用的变量以及在POU输入的变量的名称为VAR_EXTERNAL。
这样，后面要对一个已创建了的变量进行更改，只有在全局变量表中的中央数据存储区中完成更改才有意义（PC WORX支持更改后POU中变量的同步性）。就组织而言，变量表分配给资源，并成为硬件结构的一部分。

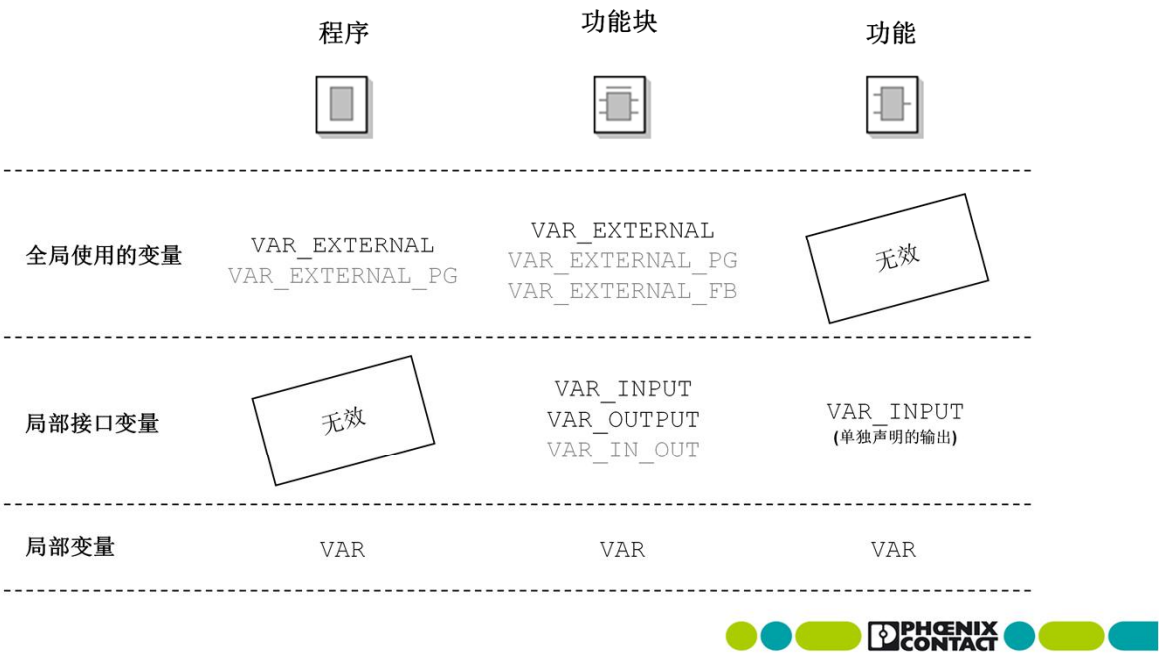
在局部变量表中，输入POU程序中相互链接的局部和全局变量。
因此，*局部变量表*并非仅指一个局部变量表，而是指属于POU的存放局部变量和局部使用的全局变量的局部表。

进一步信息

对于全局变量，使用VAR_EXTERNAL_PG和VAR_GLOBAL_PG的变量是一种特例，这些变量具有全局变量所有的特征，因此，它们可以与总线系统的过程点相连接。但是，对含有这类变量的程序而言，都需要为其每个程序实例保留单独的地址区域。这样，达到与硬件的差异耦合。

关于*程序的多重实例化*的详细信息和示例将在*PC WORX IEC 61131 通信*课程中展开。

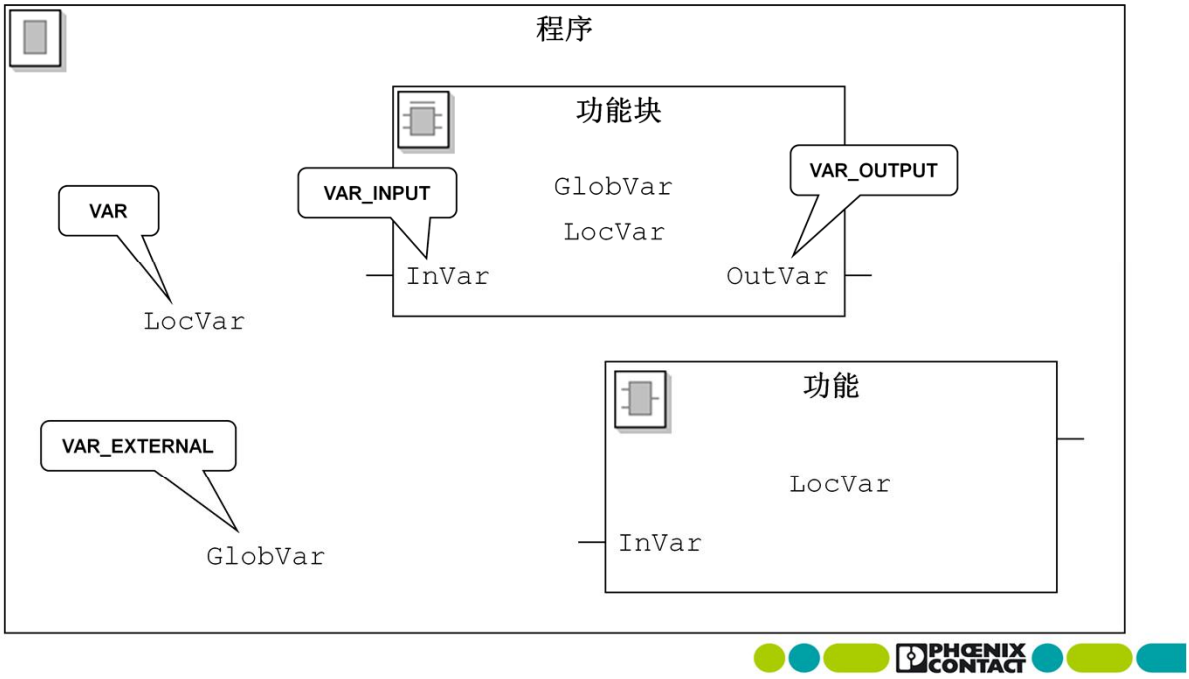
变量使用概况



除了已经描述的应用外，该图显示了在功能和功能块中使用的可能性。

*VAR_IN_OUT*对功能块呈灰色显示，由于采用*VAR_IN_OUT*不必进行数据复制，只需执行指针操作，因此它可特别应用于大量数据（字段和结构体变量）的传输。

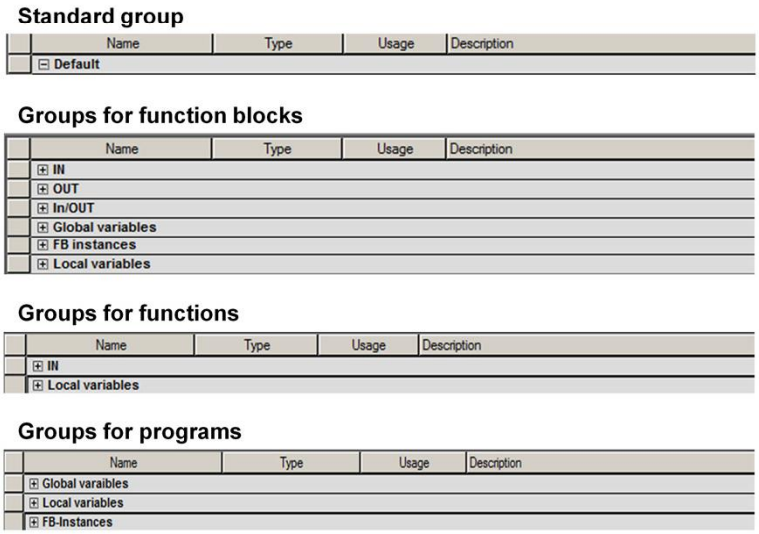
变量使用概况（图形）



该图显示了大多数公共使用。

变量管理

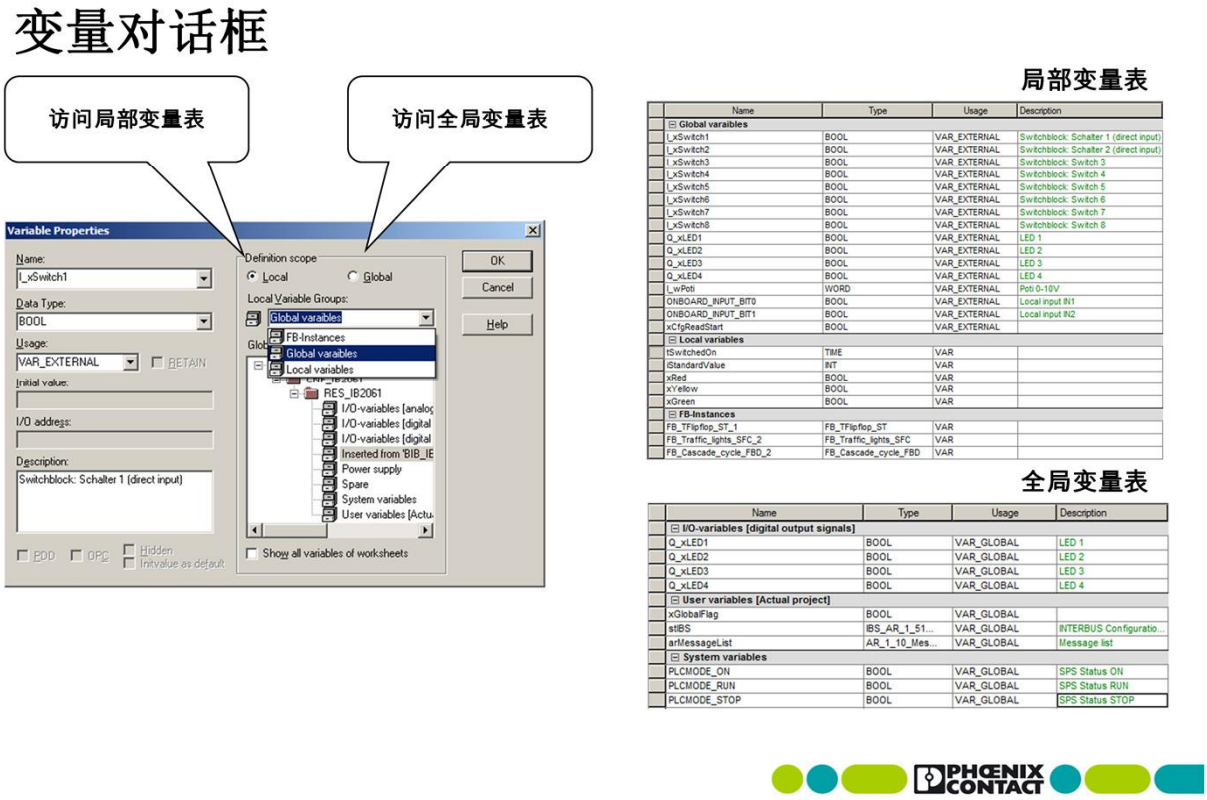
变量组



为了使变量表中的变量结构更为清晰，建议使用变量组。它们并不影响到编程，只是实现了对变量更好的组织，变量操作时效率更高。

在工作表创建时， *Default* 组是作为标准提供的。根据POU类型，屏幕截图上显示的组被证明是一个良好的基础。在更为复杂的工程中，则使用深层次的、与功能相关的组将会更有意义。

这类分组不仅适用于局部变量表，而且也适用于全局变量表。包含了系统变量的*System Variables（系统变量）*组也可以被更名。



该变量对话框允许对相应局部变量表和全局变量表中的变量进行存取。该对话框也能对已有变量进行查看和更改并可以声明一个新的变量。

如果要查看和声明全局变量，则除了选择局部变量组外，还必须选择全局变量组。

符合IEC 61131-3的编程语言

第11讲

内容

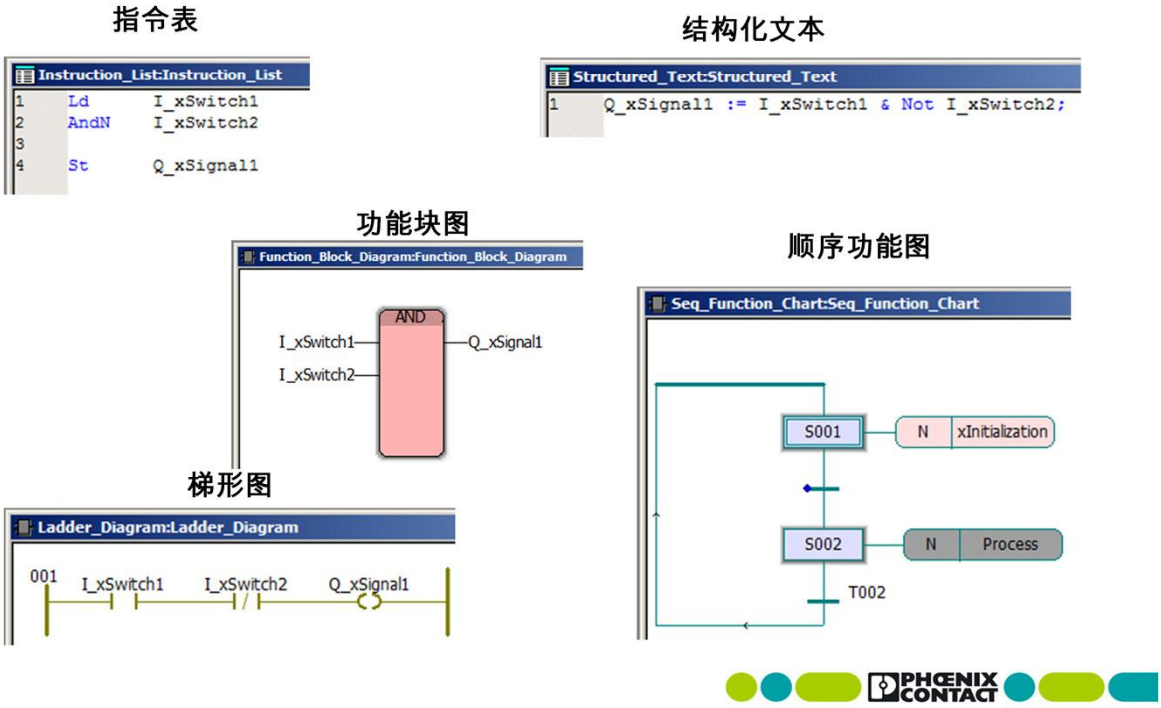
本章节简要介绍符合IEC 61131 的五种语言及其特有的优点。

注意!

信息

提示

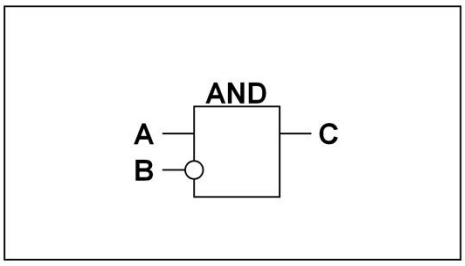
IEC 61131-3语言



作为标准， PC WORX 提供了在IEC 61131中描述的五种语言。其它的机器顺序功能图语言和FFLD（固定格式的梯形图 ）则不提供。

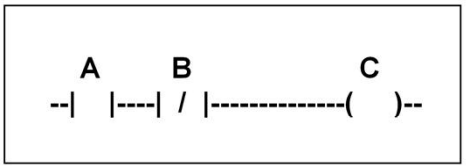
功能块图 (FBD)

- 图形化语言，广泛使用于欧洲；
- 编程元素以功能块的形式提供；
- 功能块可被“布线”，与电路图相似；
- 使用于各种负责**控制系统**组件之间信息流的应用中。



梯形图 (LD)

- 继电器控制系统编程符号的标准化有限集；
- 基于北美编程风格，与**US**电路图绘制标准相似。



指令表(IL)

- 汇编模型，使用一个累加器
- 每行允许一条命令，如保存数值于累加器中

```
LD    A
ANDN  B
ST    C
```

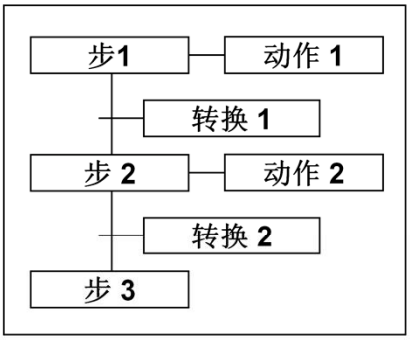
结构化文本 (ST)

- 高级语言，通过子程序实现结构化
- 语法类似于**PASCAL**
- 复杂和嵌套的指令

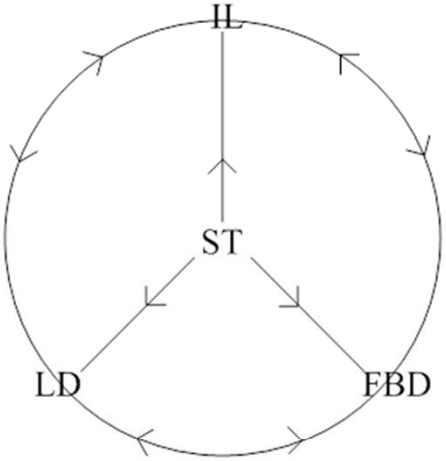
```
C := A AND NOT B;
```

顺序功能图(SFC)

- 强大的图形化编程语言，用于描述控制程序的顺序行为；
- 用于构造控制程序；
- 布置清晰的编程语言，考虑了快速诊断；
- 基本元素：含动作块的步和转换；
- 支持选择和并行顺序。

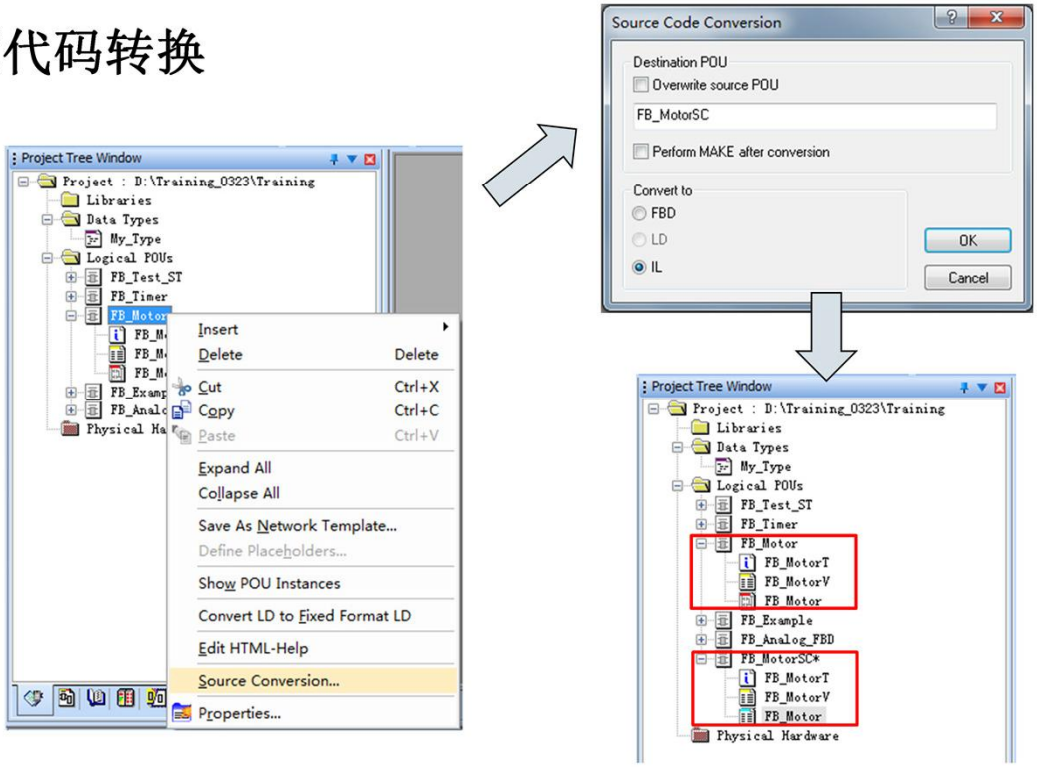


IEC 61131-3五种编程语言之间的相互装换关系



SFC

源代码转换


Project Tree Window
Project : D:\Training_0323\Training
Libraries
Data Types
My_Type
Logical POUs
FB_Test_ST
FB_Timer
FB_Motor
FB_Example
Physical Ha
Insert
Delete Delete
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Expand All
Collapse All
Save As Network Template...
Define Placeholders...
Show POU Instances
Convert LD to Fixed Format LD
Edit HTML-Help
Source Conversion...
Properties...
Source Code Conversion
Destination POU
Overwrite source POU
FB_MotorSC
Perform MAKE after conversion
Convert to
FBD
LD
IL
OK
Cancel
Project Tree Window
Project : D:\Training_0323\Training
Libraries
Data Types
My_Type
Logical POUs
FB_Test_ST
FB_Timer
FB_Motor
FB_MotorT
FB_MotorV
FB_Motor
FB_Example
FB_Analog_FBD
FB_MotorSC*
FB_MotorT
FB_MotorV
FB_Motor
Physical Hardware

采用PC WORX 编程

第12讲

内容

本章节介绍了PC WORX中工程结构的基本信息，包括程序组织单元的插入和处理以及IEC 61131 标准功能和功能块。

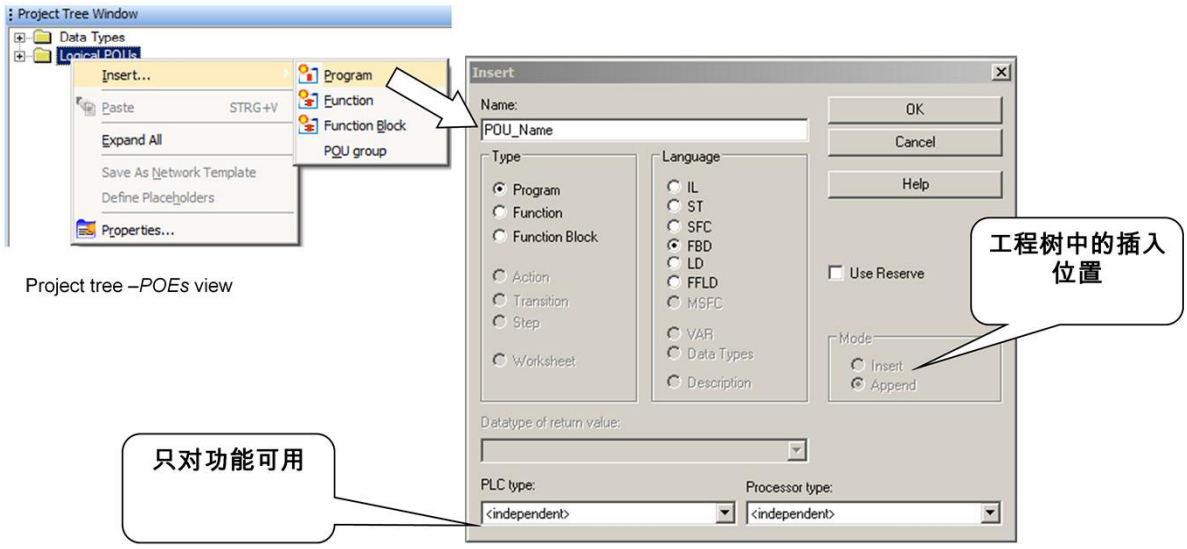
注意!

信息

提示

程序组织单元

插入POU



Project tree –*POEs* view

如果不使用通过工程模板自动插入的*Main* 程序，创建程序的第一步是插入一个*程序*类型的POU。当选择*逻辑POU*项或一个已有的POU时，通过右键快捷菜单或菜单栏插入一个POU。

根据待插入元素（*类型*框），提供多个选项。例如，并非每个POU类型都可选择所有语言（功能不支持顺序功能图，程序不支持机器顺序功能图语言）。

*返回值的数据类型*选项只对功能有效。正如在符合*IEC 61131的变量结构*章节中所描述的，功能只有一个输出参数。该参数的数据类型是通过选择确定的，其参数名是通过POU名确定的。

如果使用了PLC- 或处理器专门的功能和功能块，则 PLC和处理器类型的选择<*independent（独立的）*> 才要改变。如果使用了该选项，该POU只能在确定类型的PLC或处理器上进行操作。

POU在工程树中的位置并不影响执行顺序。程序是通过任务以程序实例的形式进行调用的，代码工作表中的功能和功能块的调用决定了程序顺序。

POU 属性

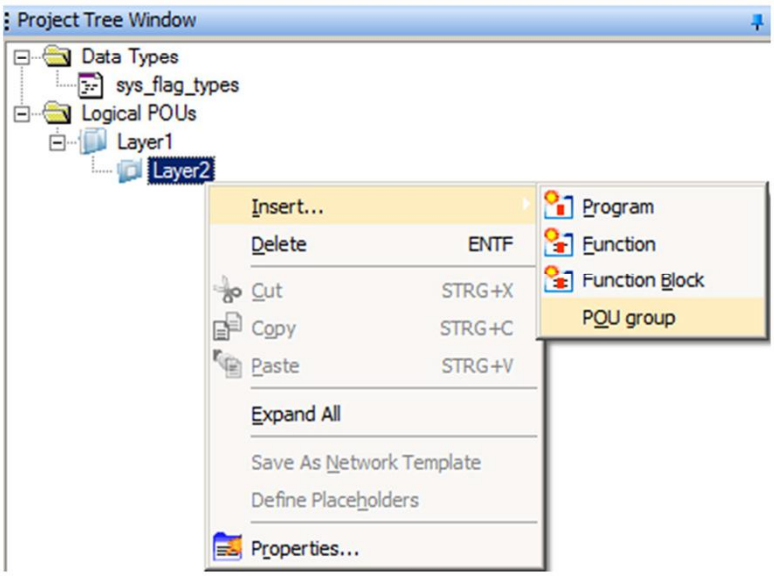
除了语言外， POU属性随后可在POU属性对话框中进行修改。

改变类型是可能的，但是，这意味着对工程而言需要花费更多的精力进行修改，这视POU使用的频率而定。保留是指在下载后来的更改的过程中，为POU保留存储单元。作为标准，设置资源（硬件结构），使存储单元对每个POU都有效，而不必进行单独保留。

关于*PLC/Processor*（*PLC/处理器*）标签页
*只读*属性用于多用户功能，但也可以用于防止无意变更。

POU的安全设置考虑了编程专门保护，通过菜单*File → Enter password*（*文件→ 输入密码*）输入密码，才能激活该保护。

POU组



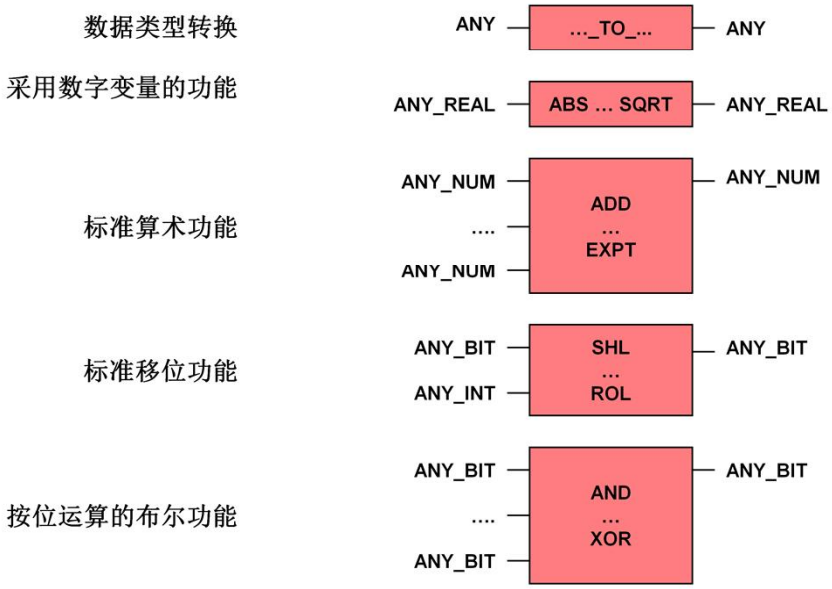
为了能使您所创建的程序组织单元能在工程中保持组织清晰，一旦工程达到一定的规模，则建议使用POU组。

POU组并不影响程序分配，只是一种用于更好进行组织的方法，并可随意创建。

POU组总是位于*逻辑POU*文件夹的最顶层，同层中按字母排序。

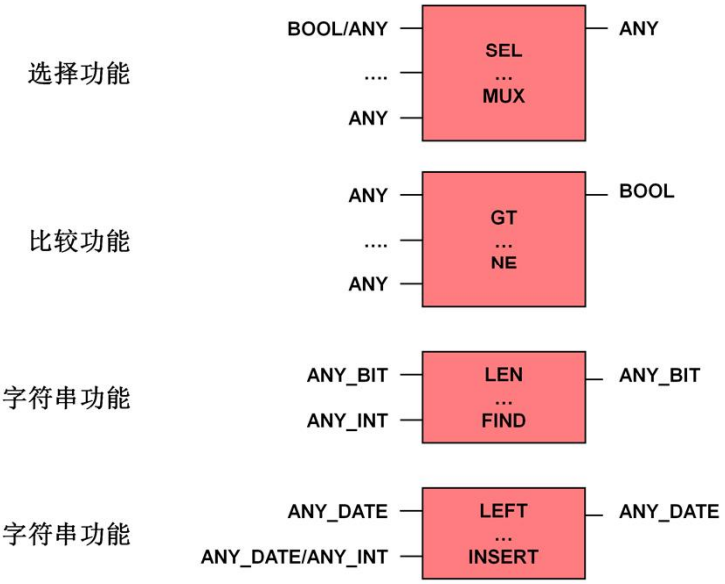
IEC 61131的标准功能

标准功能 1



PC WORX安装后，可提供多种功能，可通过编辑向导查看到，它们是由IEC 61131定义的。由于存在多种数据类型，类型转换功能组尤其大。

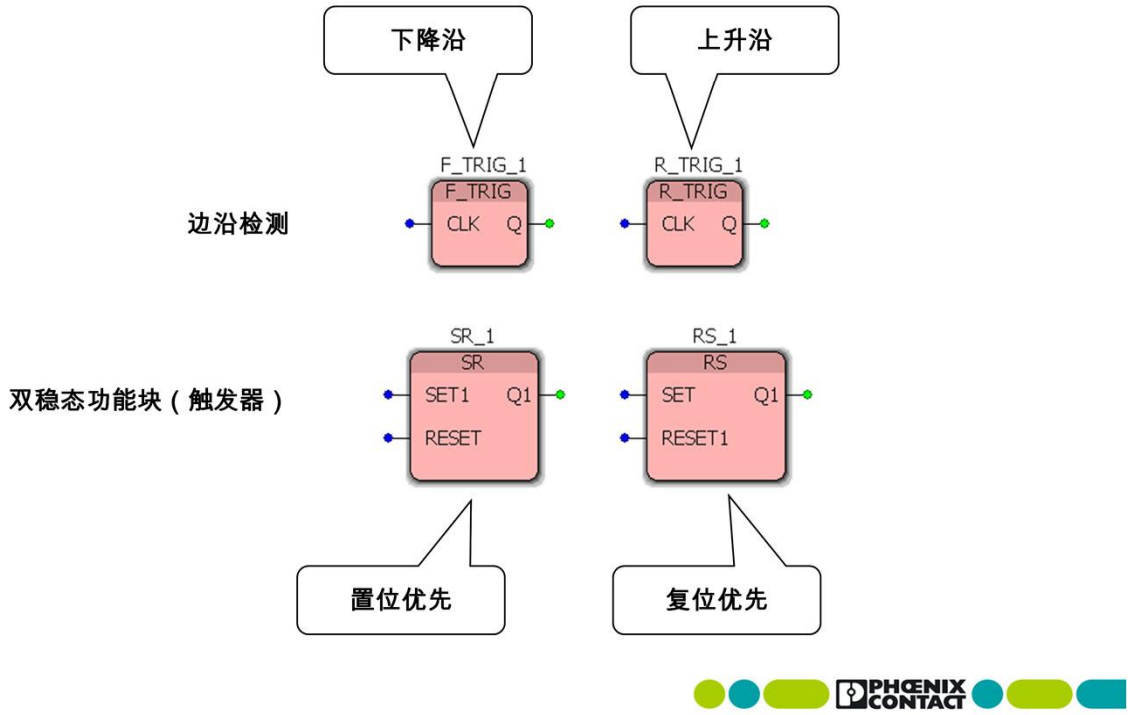
标准功能 2



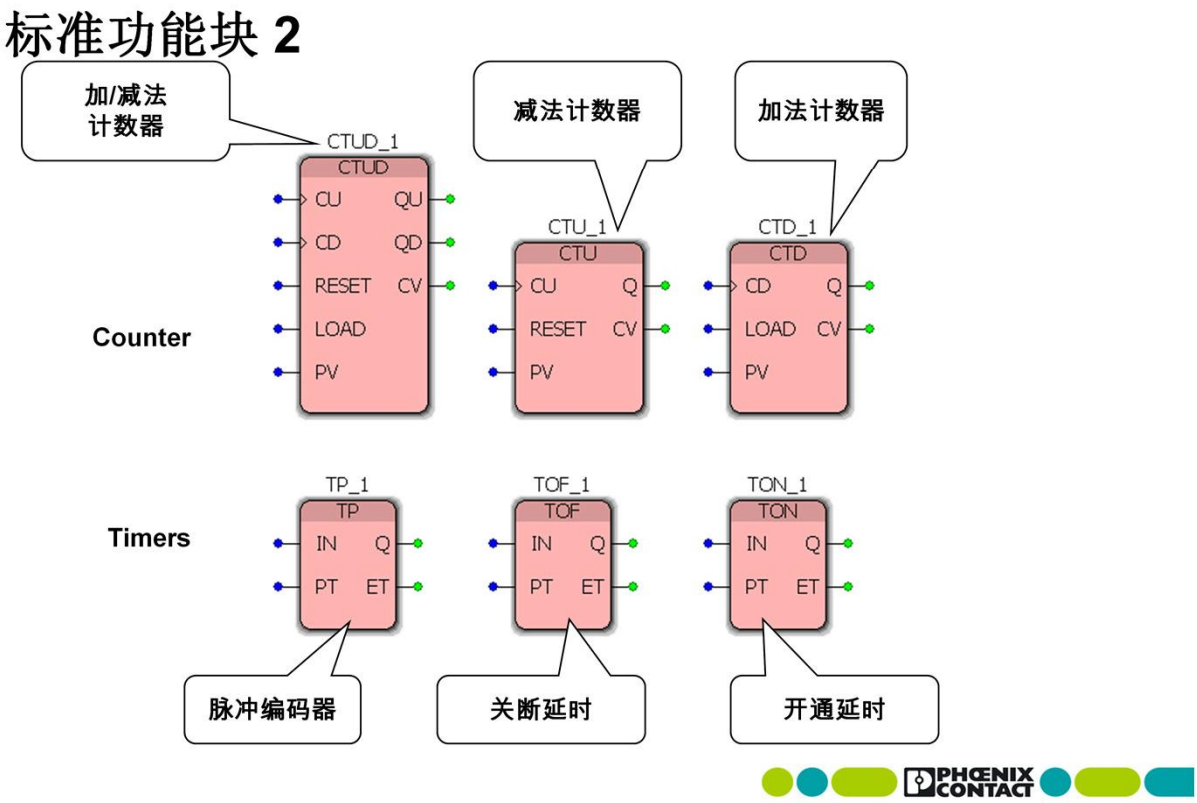
块的连接及功能的细节可在PC WORX中每个块的HTML帮助中查找得到。

IEC 61131标准功能块

标准功能块 1



IEC 61131定义了如上所示的四组功能块。就功能而言，它们是从传统的PLC编程系统的块中派生而来的。其中一个例子就是RS触发器是复位优先，而SR触发器则是置位优先。



有关功能块和功能的接线及功能的详细信息可在PC WORX HTML帮助中查找得到。

FBD - 功能块图

第13讲

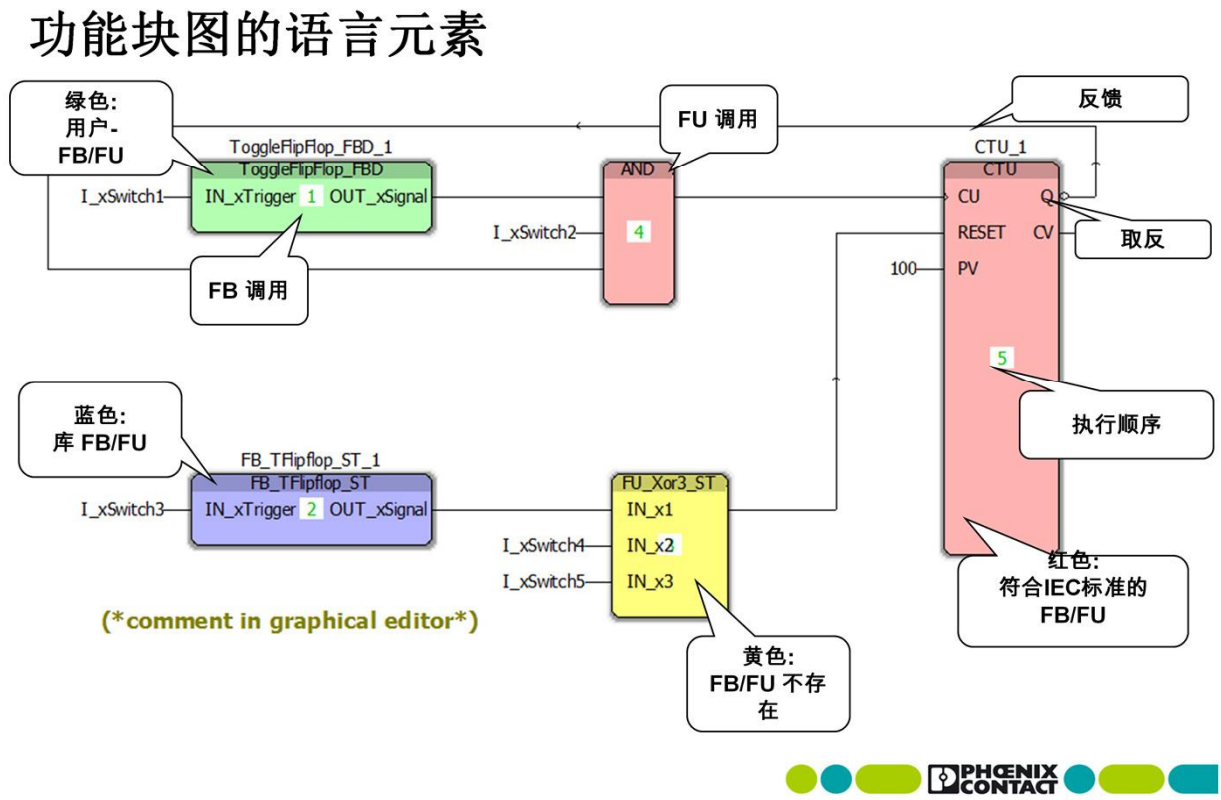
内容

本章节介绍了功能块图编程，包括功能和功能块的插入，以及不同有效性的变量的存取。

注意!

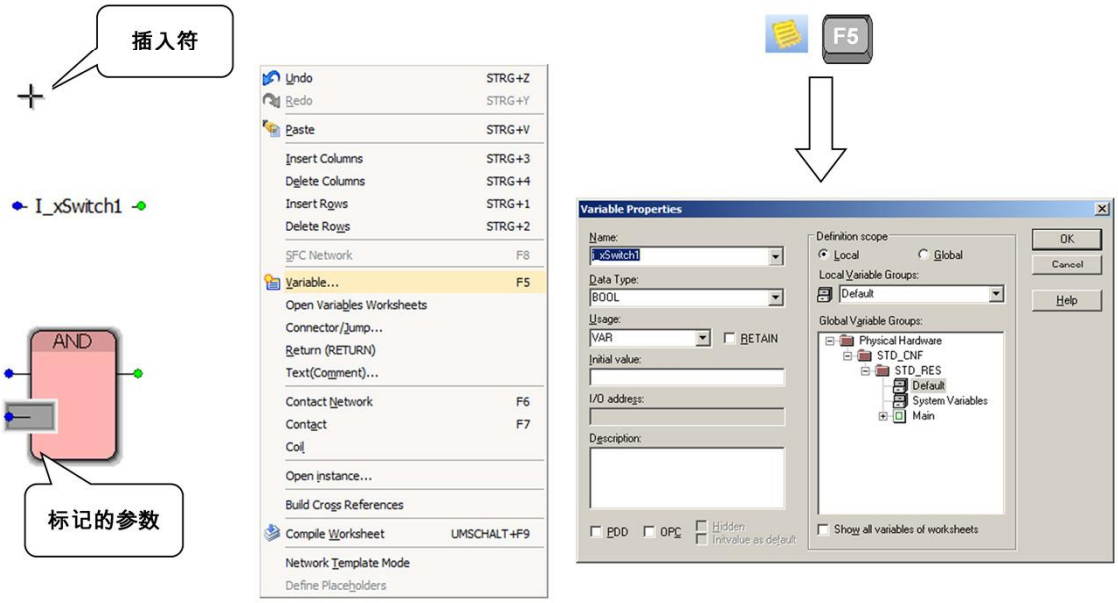
信息

提示



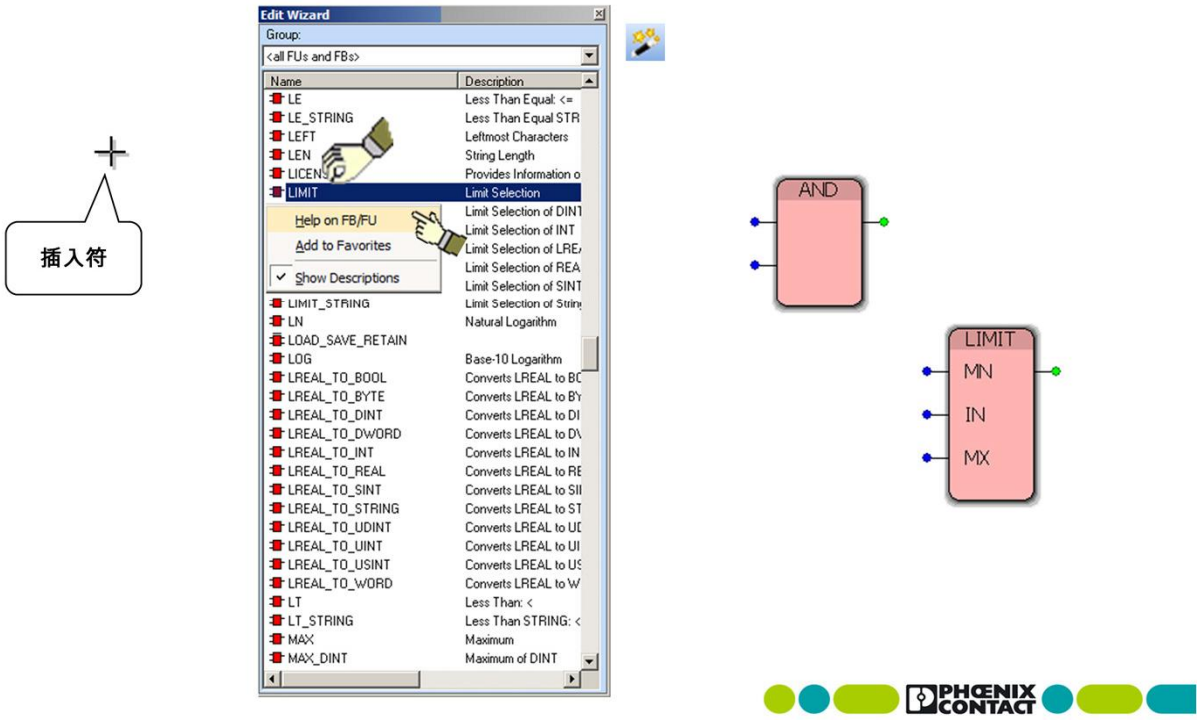
以上概括介绍了在功能图中可以使用的语言元素，基本上，是块和变量。

存取变量



在FBD中，通过变量对话框来完成变量的插入。该对话框可用来存取局部和全局变量。
在空白的FBD编辑器中，在显示对话框前，首先必须放置插入标识或对功能或功能块的连接点加以标识。
该对话框可通过右键快捷菜单，上图所示的按钮或通过F5功能按钮（默认设置）激活。

存取功能

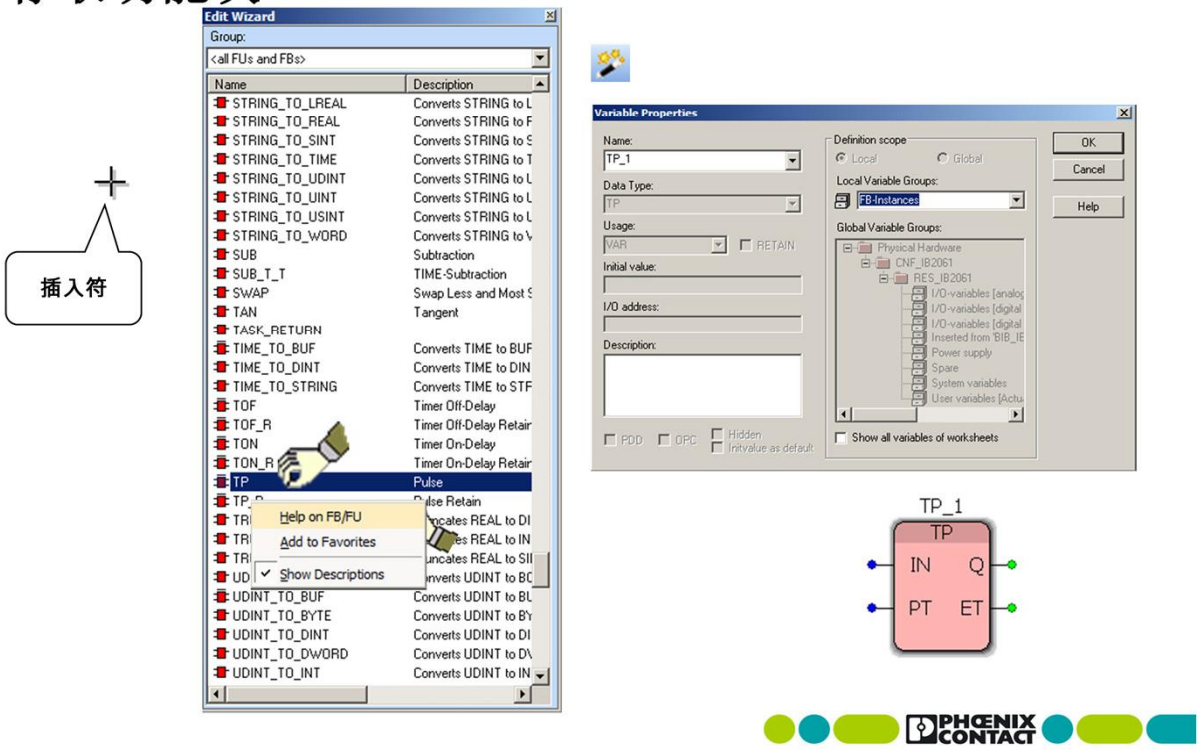


要通过编辑向导存取功能，首先必须在图形工作单中放置插入标识。双击所选的功能完成插入。另一种方式，也可以使用拖放操作来插入一个功能。

每个标准块都提供了HTML帮助，也可以通过右键快捷菜单调出。

通过按钮 或标准组合键Shift+F2，可显示/隐藏编辑向导。

存取功能块

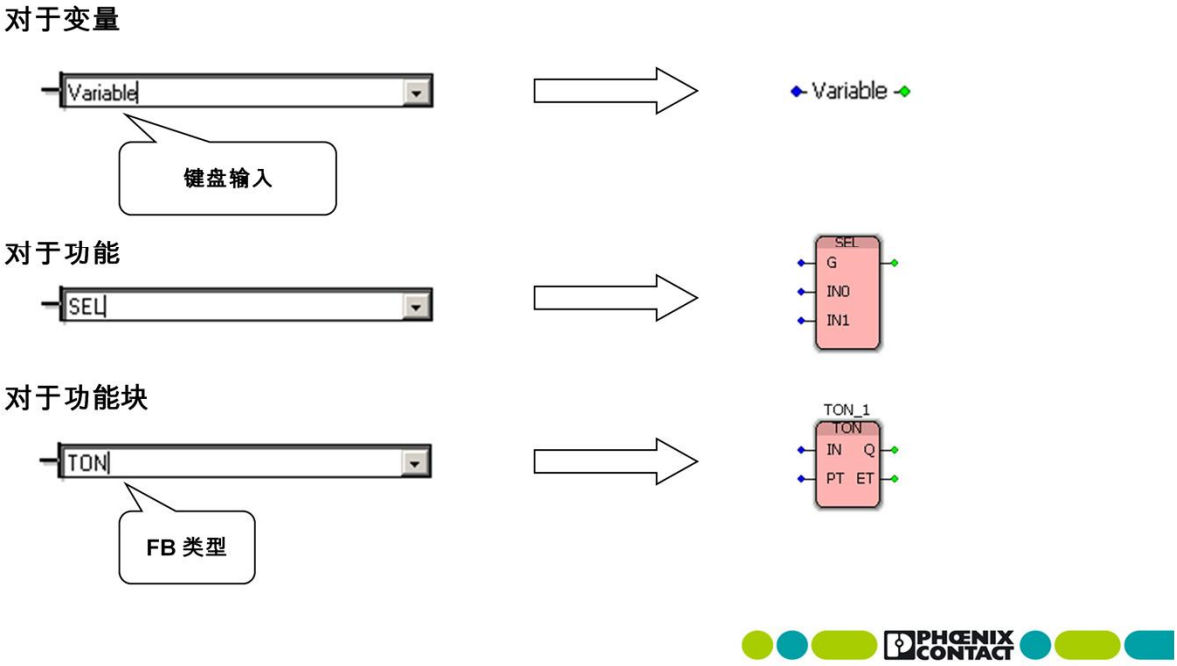


功能块的插入方法与插入功能一样。但是，在插入功能块之前，必须对其实例进行声明。关于实例名， PC WORX 建议采用功能名加上一个增量，之间用一个下划线进行连接。该名可以由用户单独（即与功能相关）选择。

每个标准块都提供了HTML帮助，也可以通过右键快捷菜单调出。

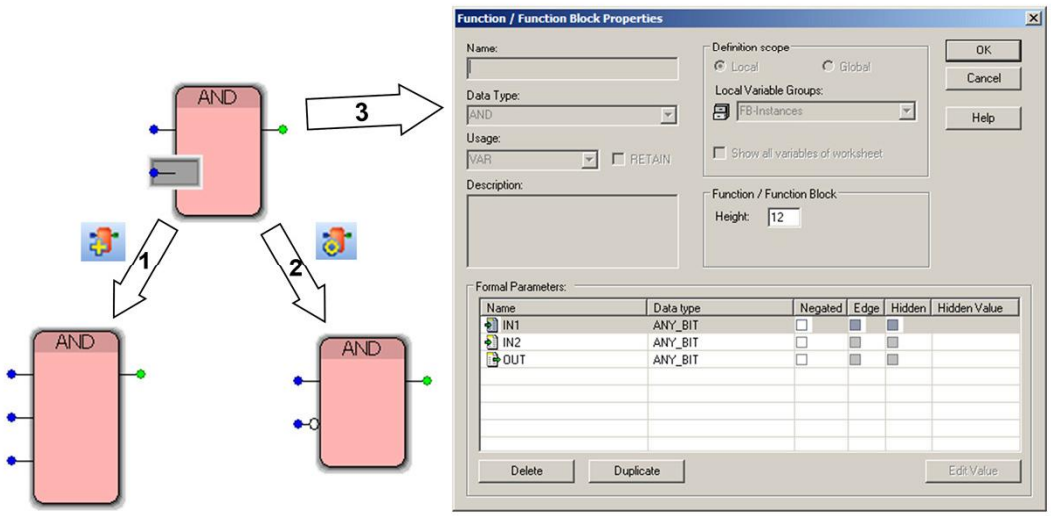
上例显示了工程的插入对话框，其中已经为FB实例创建了一个局部变量组。否则 ，只 *ult* 组。

通过键盘实现简单存取



PC WORX中的图形编辑器可以直接通过键盘输入实现编程元素的存取，而不用通过对话框存取。在这之前，必须将一个形式参数选中或者在工作单中插入一个插入符，这样就可以使用键盘直接输入要插入的元素的名称。对于插入功能块而言，要输入的必须是功能块的类型，而不是功能块实例所需要的名称。

编辑形式参数

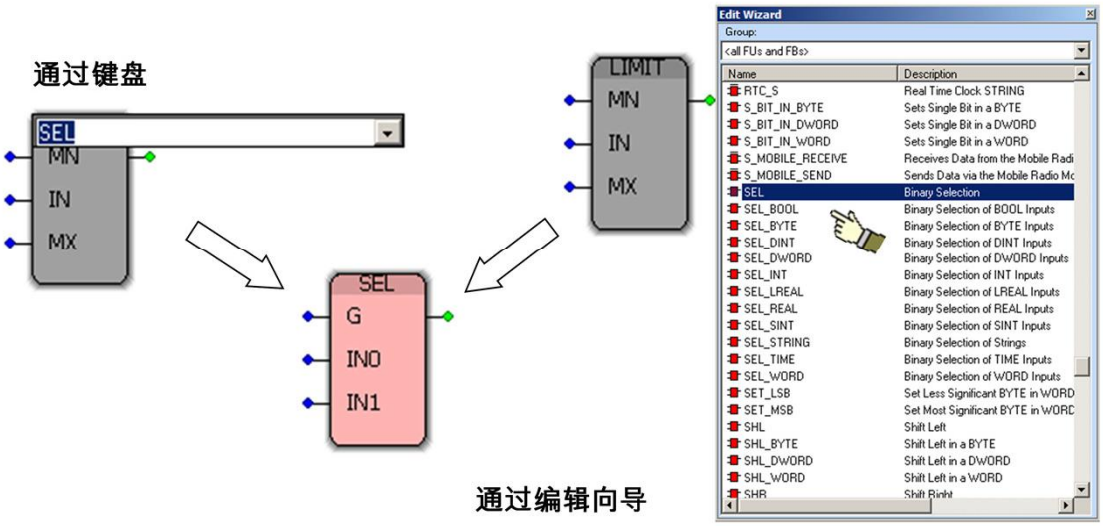


本页显示了编辑块（运算符、功能和功能块）的形式参数的几种不同的操作。

这包括（1）添加输入参数（通常只被运算符支持）和（2）对基于位参数的输入和输出参数进行取反（并非所有的块都支持）。当选择一个参数时，上图所示的按钮被释放。

双击可激活标准块的（3）属性对话框，对于用户自定义块，可通过右键快捷菜单中的*属性*菜单项激活。

替换块



一个功能或一个功能块可以通过编辑向导使用拖放操作进行替换，或者如果该块是被选中的状态，可以通过双击一个要替换的块来实现。另一种方式，可以通过键盘输入来实现对块的简单存取。

用户编制的功能块和功能

第14讲

内容

本章节介绍如何创建功能和功能块，在创建时必须考虑的因素，如何在当前工程中使用这些块以及如何供其它工程使用。

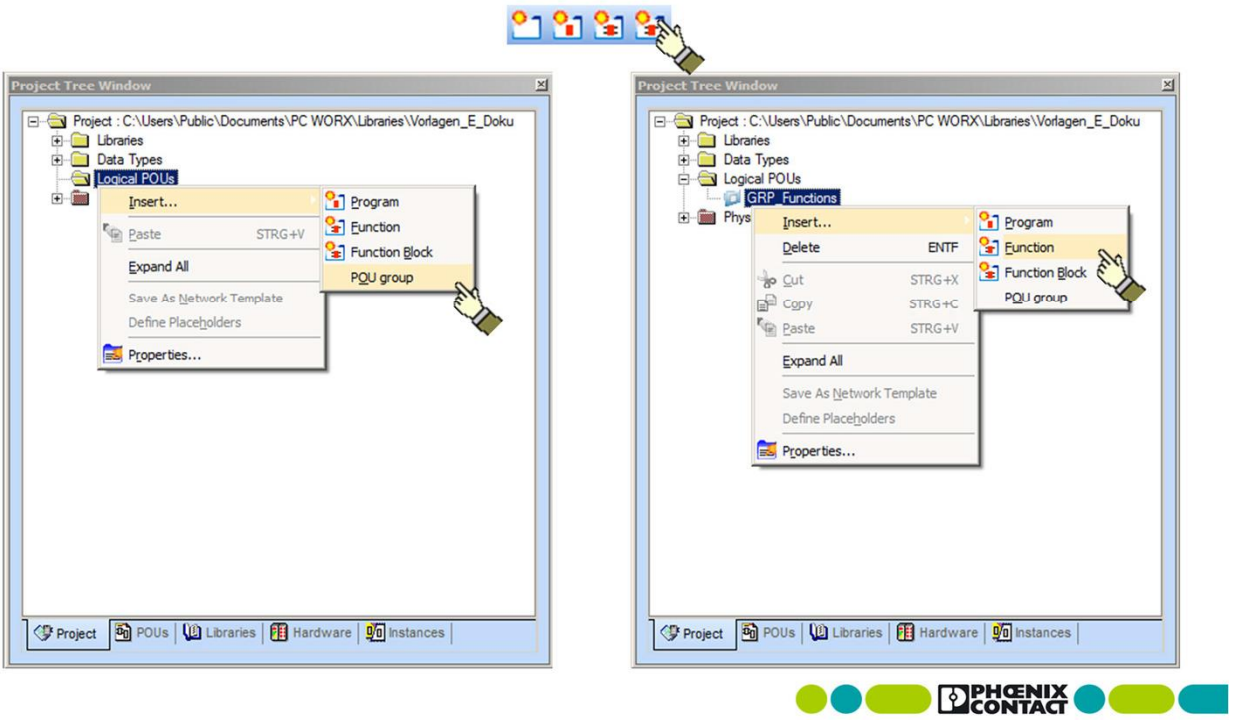
注意!

信息

提示

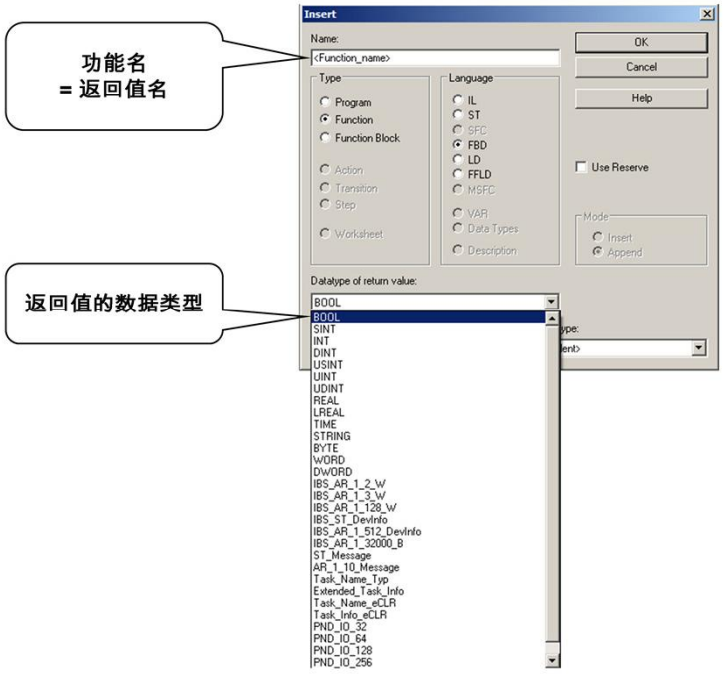
创建功能

插入到工程树中



选中逻辑*POU文件夹*或POU组，就可以在工程树中插入功能。根据工程的规模，将所创建的块组织成POU组将更有意义，这些块也可以在后面添加到组中。

设置功能属性



要插入一个功能，只有少数参数需要设置。

功能名

之后，系统将列出块的功能名，以供在编辑向导中选择使用。此外，该名将用作该块内部程序的返回值。该名必须符合变量名规定。

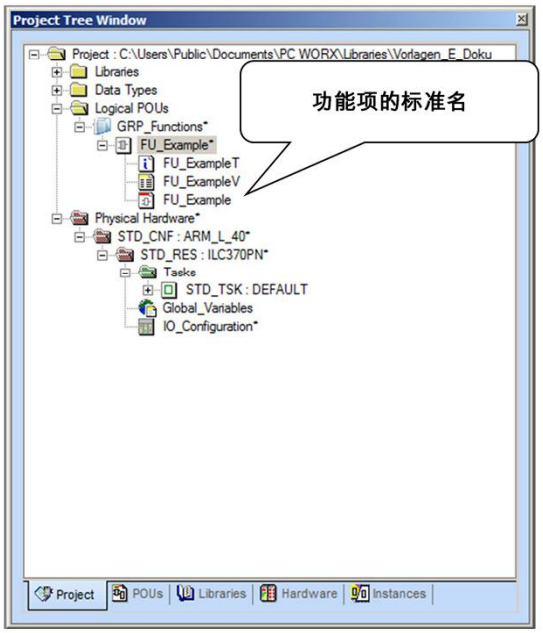
返回值

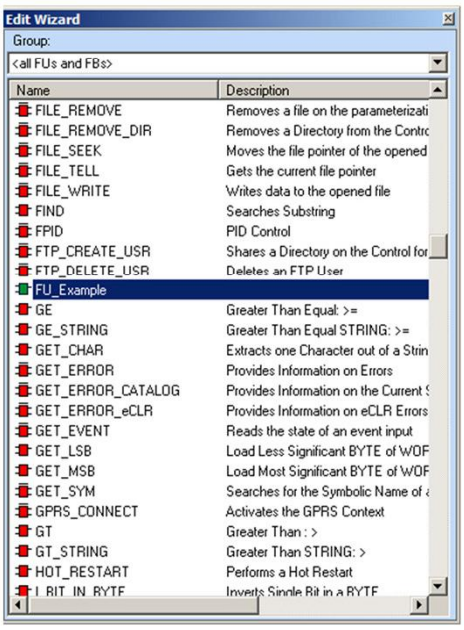
功能的单一输出值的类型是通过返回值的数据类型确定的。

PLC和处理器类型

如果要访问所选类型相应的专用块，则必须要确定PLC和处理器类型。由于大多数专用块都是功能块，这样就不能被调用，因此，这不大可行。

工程树中的功能





当功能插入到工程树中后，则以选用的名称和三个基本项的名称列出，其中基本项是以首个名字为基础的。

<功能名>T 用于注释；

<功能名>V 用于变量表；

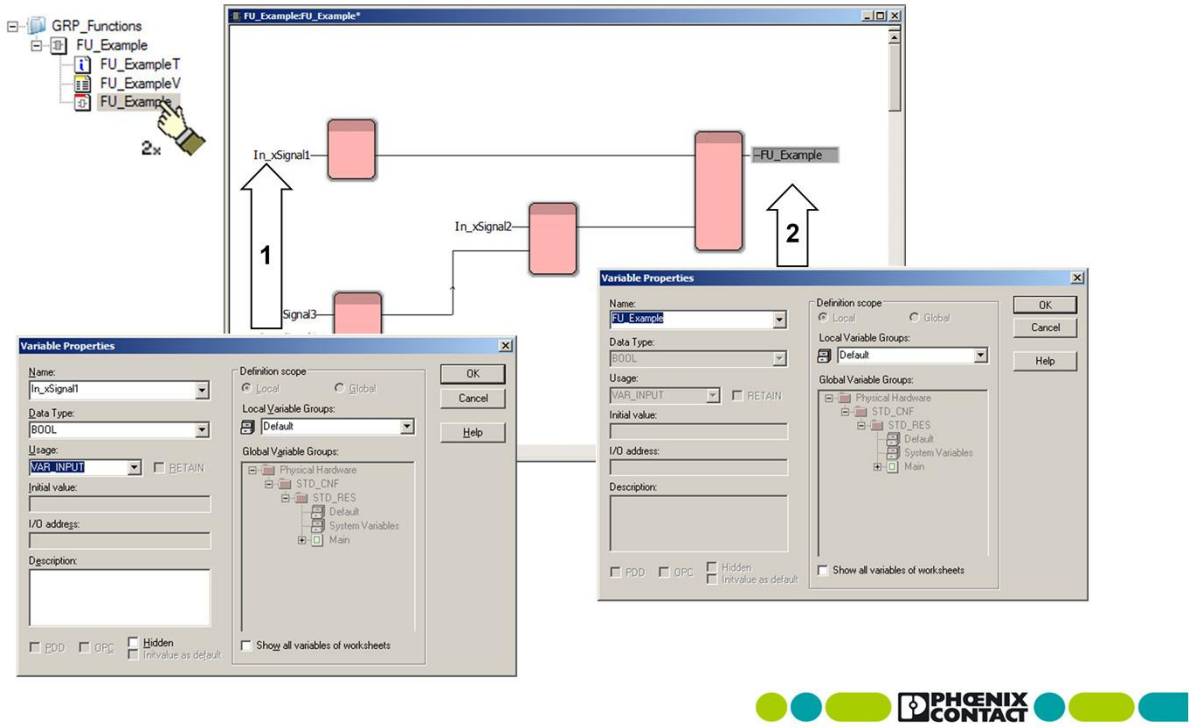
<功能名>用于首个代码工作表。

除此之外，在编辑向导中，此功能在<*所有的FU和 FB*>组中以及以当前工程名命名的组中出现。

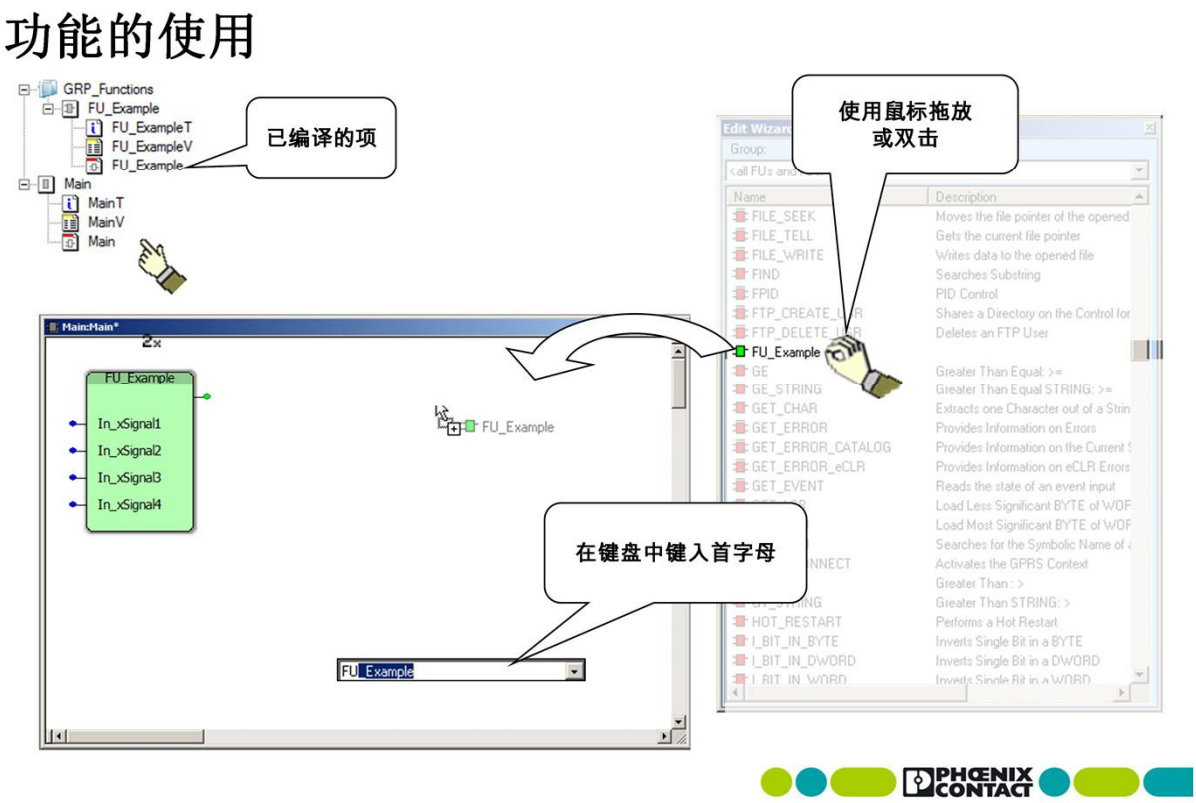
虽然用户创建的功能属于功能，但是它们还并不属于<*功能*>组。

此时，该功能还不能被调用。变量表和代码工作表后面的星号显示了它们还未编译。

编辑功能



功能首先是按照标准的编程程序在其工作单中进行编辑。如上图所示，必须至少要声明一个（1）输入参数，此外，利用工程可用的功能算法，计算出其（2）返回值（名=功能名）。



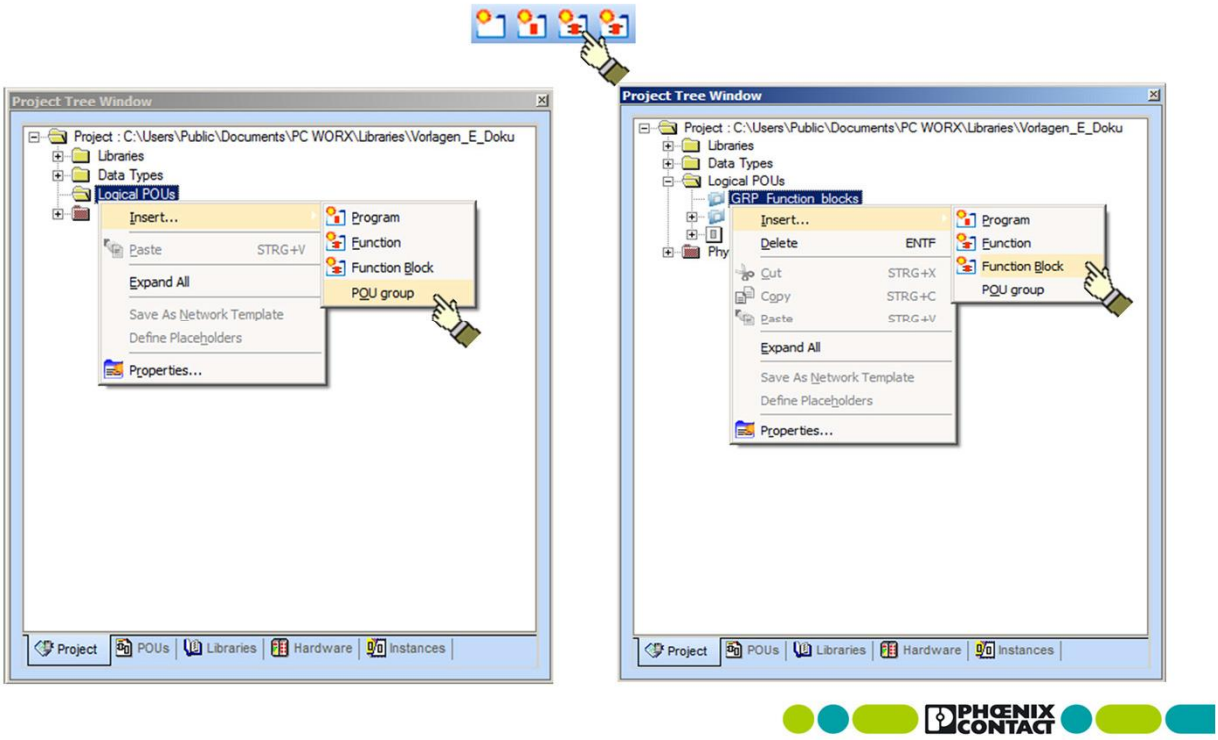
变量表的编译是通过关闭表来完成的。

通过关闭代码单或明确使用相应的按钮或组合键（标准的是Shift+F9 ）可对代码表进行编译。

变量单成功编译后，功能可在其它的POU中调用使用。

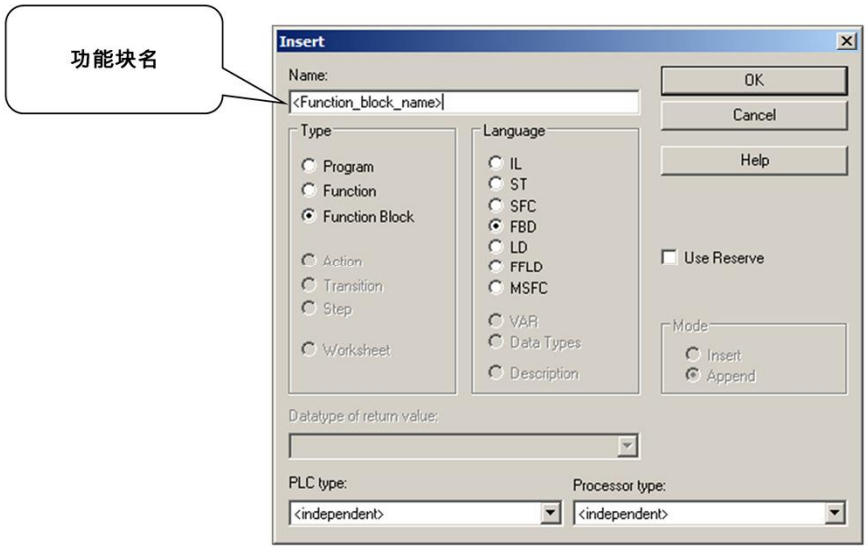
创建功能块

插入到工程树中



当逻辑*POU文件夹*或POU被选中时，则可以在工程树中插入功能块。根据工程的规模，将所创建的块组织成POU组将更有意义，这些块也可以在后面添加到组中。

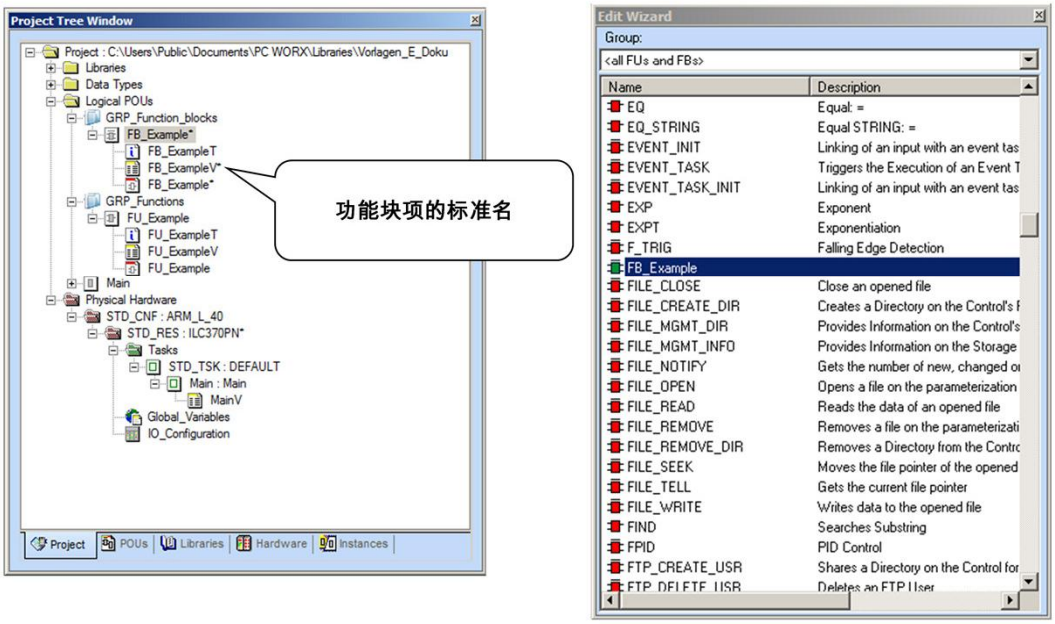
设置功能块属性



对功能块的插入而言，可使用的语言种类比功能的插入要多。也不必定义返回值的数据类型。

之后，系统将列出块的功能块名，以供在编辑向导中选择使用。该名不能在FB的内部使用，且必须符合变量名规定。

工程树中的功能块



当功能块插入到工程树中后，则以选用的名称和三个基本项的名称列出，其中基本项的名称是以首个名字为基础的。

<功能块名>T 用于注释；

<功能块名>V 用于变量表；

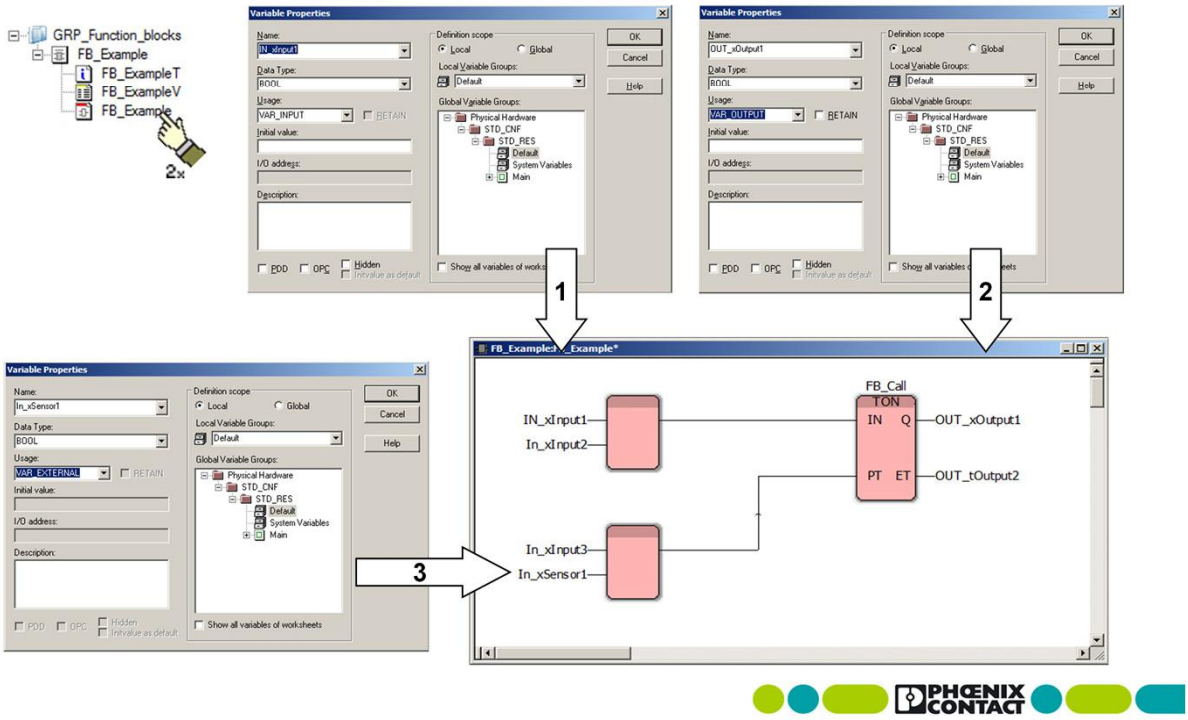
<功能块名>用于首个代码个工作表。

除此之外，在编辑向导中，此功能块在<*所有的FU和 FB*>组中 以及以当前工程名命名的组中出现。

虽然用户创建的功能块属于功能块，但是它们还并不属于<*功能块*>组。

此时，该功能块还不能被调用。变量表和代码工作表后面的星号显示了它们还未编译。

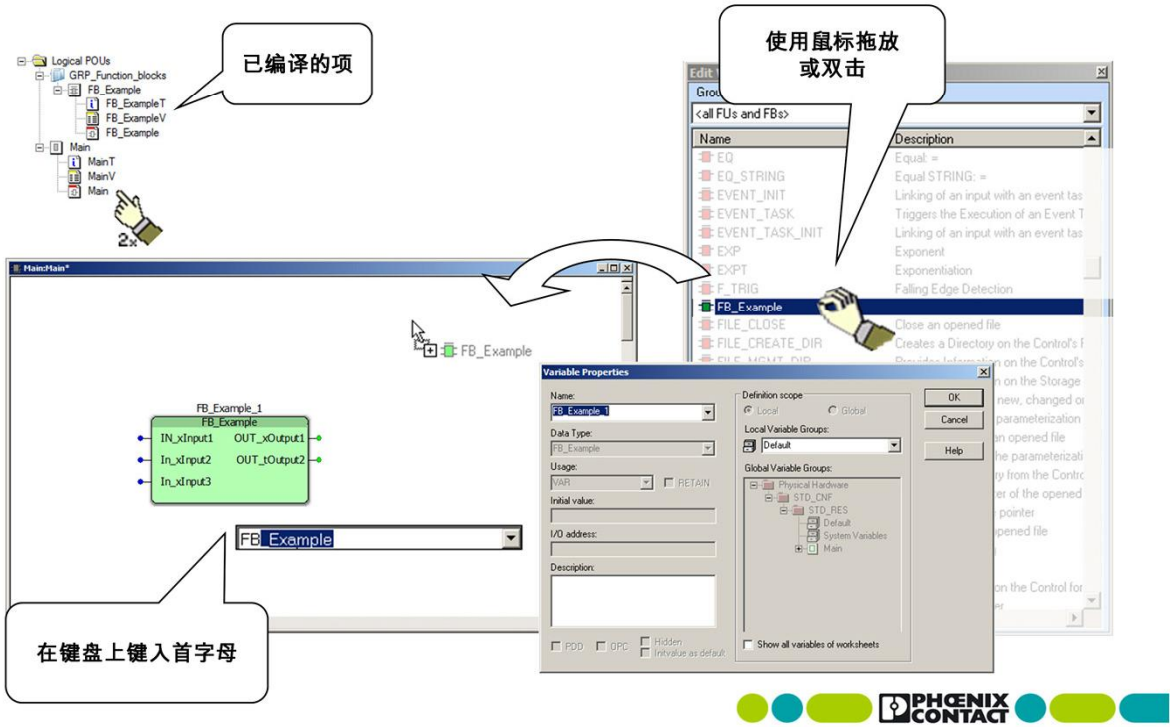
编辑功能块



功能块首先是按照标准的编程程序在其工作表中进行编辑。与功能不同的是，就设计而言，自由度非常高。可以声明任意多个输入（1）和输出（2）参数，功能块也可以对全局变量（3）存取。

唯一要记住的是在对功能块进行内部编程的时候功能块名不应用作变量名。

功能块的使用



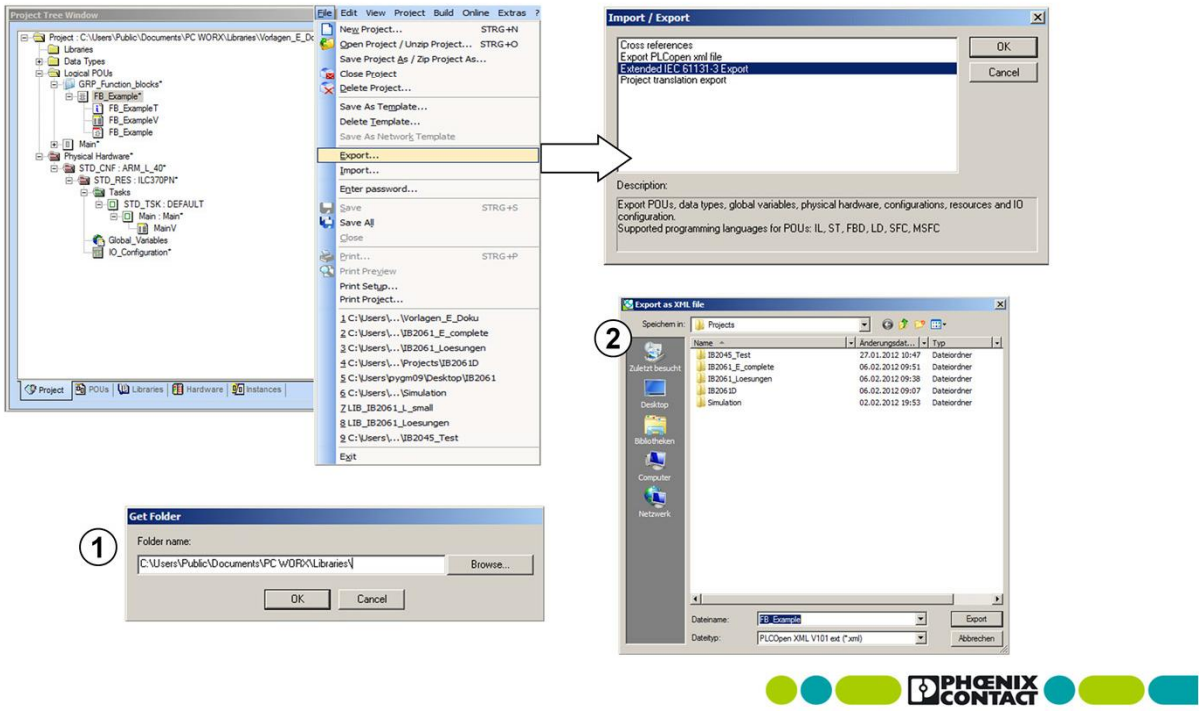
变量表的编译是通过关闭表来完成的。

通过关闭代码单或明确使用相应的按钮或合键（标准的是Alt+F9 ）可对代码表进行编译。

变量单成功编译后，功能块可在其它的POU中调用使用。

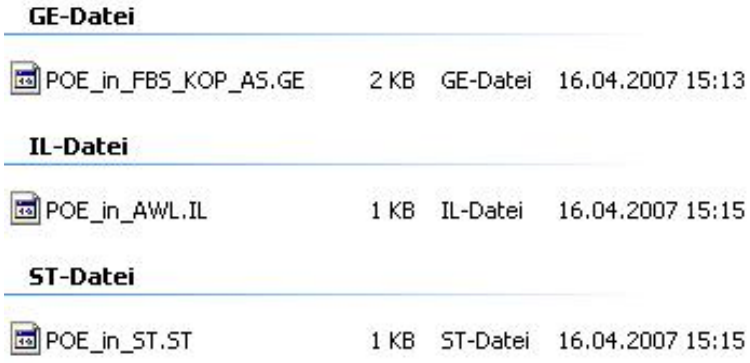
功能块的导出和导入

块的导出



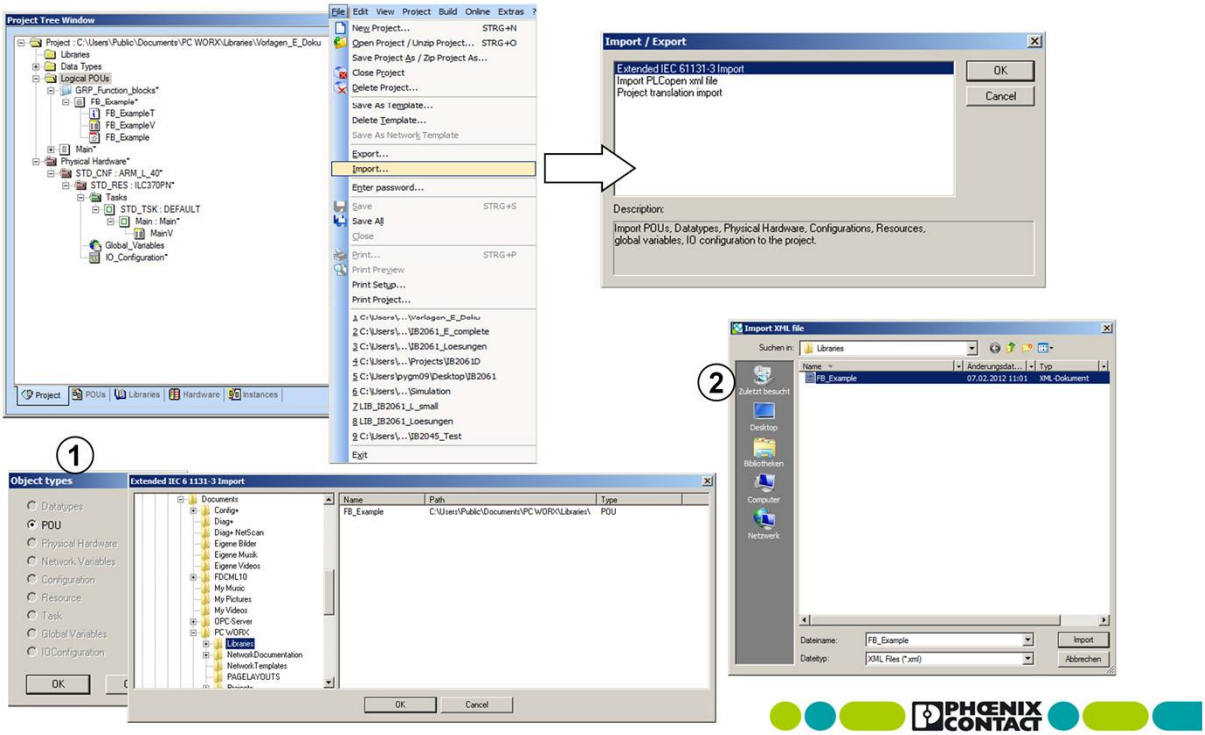
在块能被导出，保存为文件或可供其它工程使用之前，必须在工程树中选中该块。在*File（文件）*菜单中，选择*Export（导出）*，您可以在保存POU为（1） IEC 61131-3 文件或（2）PLC开放的XML文件之间进行选择。

IEC 61131-3 文件



PLC 开放 XML文件 (图形编辑器)

块的导入



为了能从（1）一个IEC 61131-3 文件或一个（2） PLC 开放 XML 文件中导入一个块，必须选中*逻辑POU文件夹*或其中一项。在*File（文件）*菜单中选择*Import （导入）*，打开对话框选择待导入的类型。

IL - 指令表

第15讲

内容

本章节介绍了在PC WORX中如何使用指令表进行编程。除了简单的指令外，如操作符和功能外，还介绍了如何调用功能块，以及如何通过跳转jump和返回return有条件地执行程序。

注意!

信息

提示

指令表中的语言元素

赋值和操作符

```
Ld    I_xInput1
AndN  I_xInput2
Not
St    Q_xAccess1
```

FU 调用和自变量

```
Ld    iMin
Limit iBasis, iMax
St    iCorrected
```

条件处理

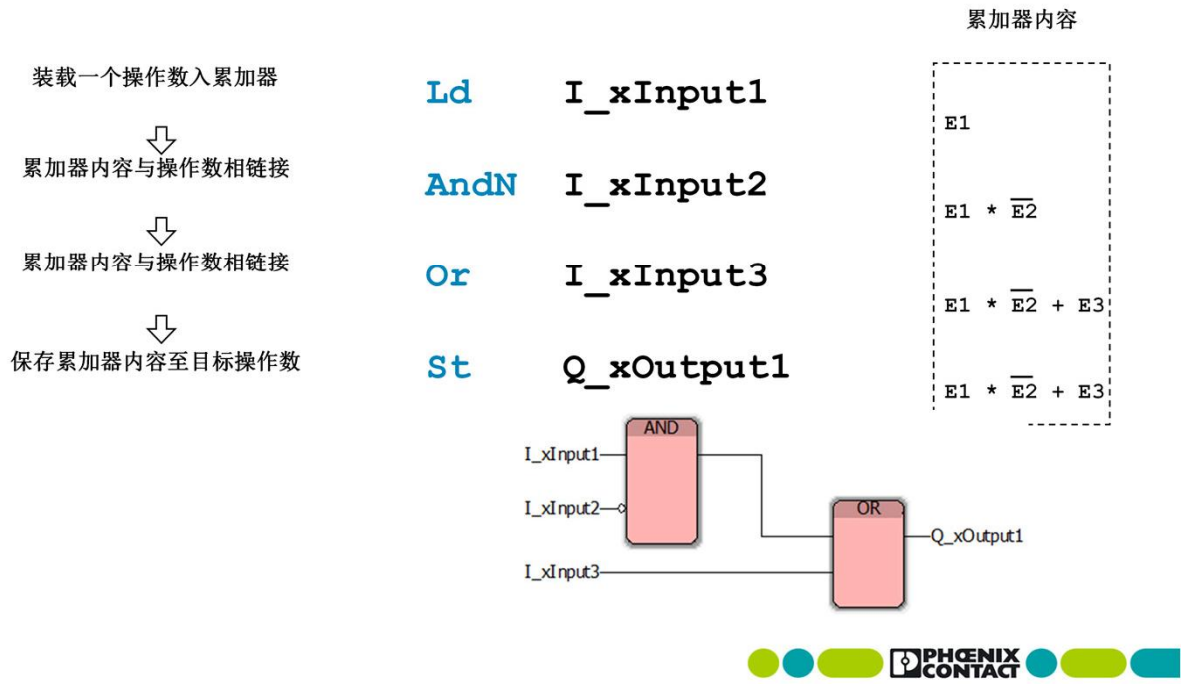
```
Ld    I_xInput1
JmpC  JumpFlag
…
Ret
JumpFlag:
Ld    I_xInput2
St    Q_xAccess1
```

FB 调用和自变量

```
Ld    xPulse
St    CTU_Access.CU
Ld    iMaximumvalue
St    CTU_Access.PV
Ld    xStop
St    CTU_Access.RESET

Cal   CTU_Access

Ld    CTU_Access.QV
St    iDisplay
Ld    CTU_Access.Q
St    xFull
```

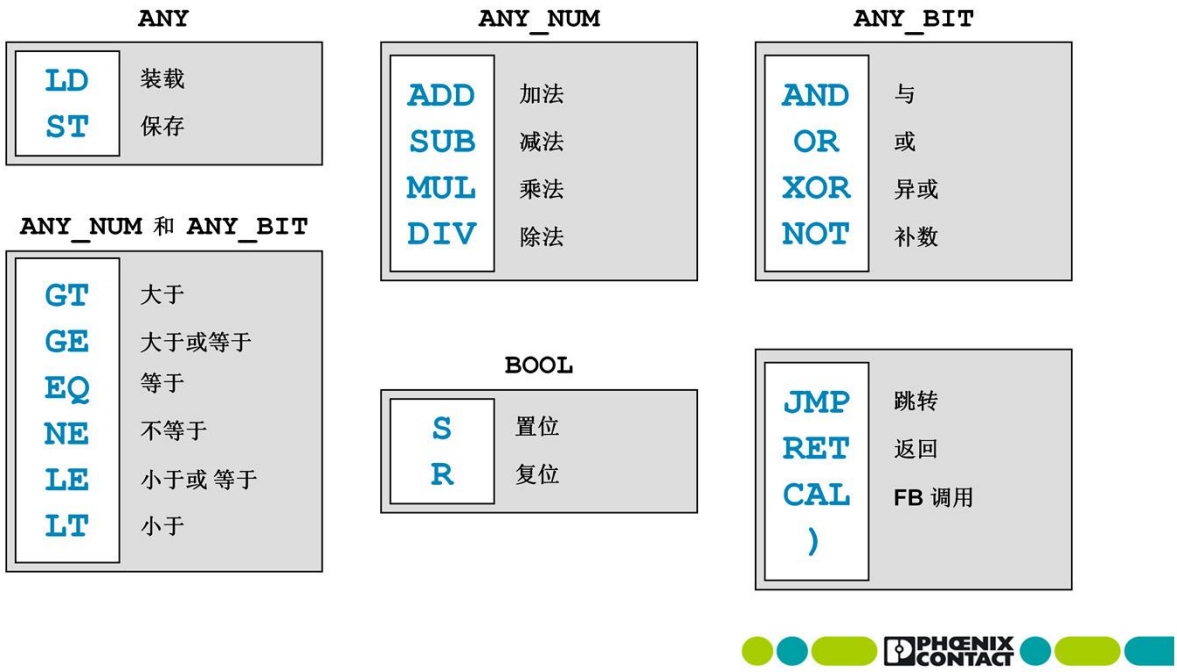
对IL中可用的语言元素的访问大多由编辑向导支持。除了*功能*和*功能块*组外，在指令表中还提供*操作符*组。

赋值和操作符



PC WORX中的指令表使用了一个累加器，对所有的数据类型都采用了统一的一对装载和保存命令。

指令表中的操作符



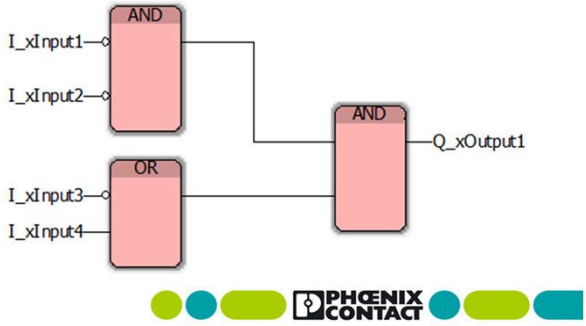
本图概括了可使用的数据类型变量和数据类型组的操作符。 指令表中指令格式都是相同的。

其中的一些特例是补数*NOT*，功能块和*RETURN*和闭括号，使用它们时在同一行中可以不带操作数。

在传统的PLC编程中作为标准，根据正链接，操作符*S*和*R*一次性将布尔量1或0写到连接的操作数中。

修改操作符

```
LdN    I_xInput1
AndN   I_xInput2
And(   I_xInput3
Not
Or     I_xInput4
)
St     Q_xOutput1
```

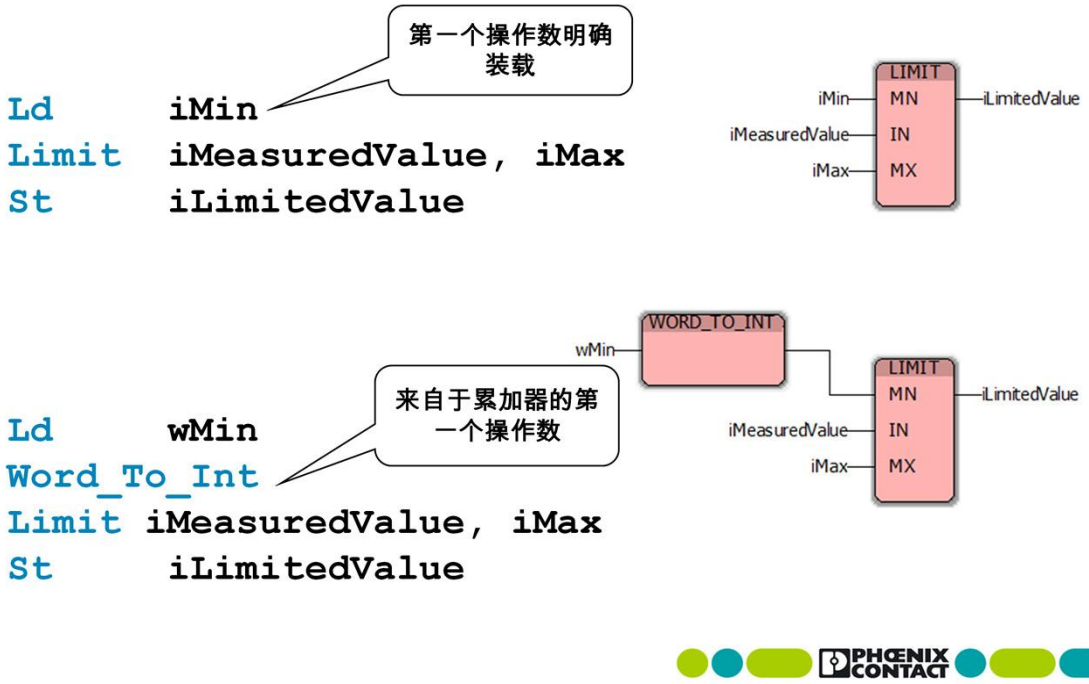


通过修饰符*N*，存取存储器的操作符*LD* 和*ST*则可对基于位的操作数执行取反运算。这同样适用于使用布尔量操作数的操作符。

开括号作为一个修饰符，使括号内的代码优先处理。

闭括号作为一个操作符，独自出现在一指令行中，并触发优先代码的处理。

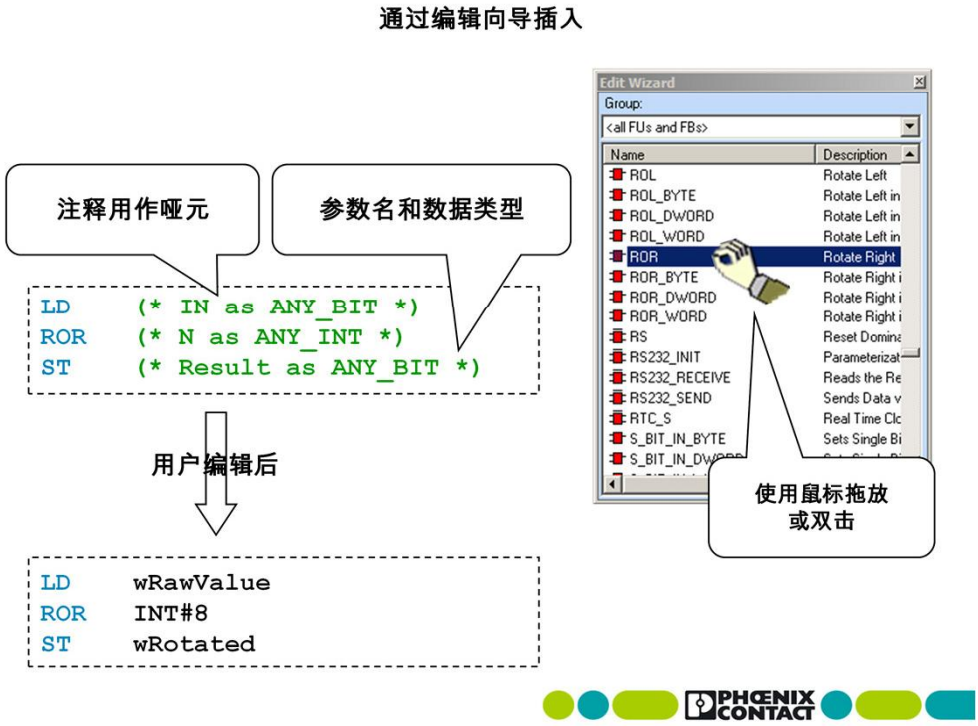
功能调用



指令表中的功能调用只是与标准操作句法有细微的差别。首个操作数可通过*LD*装载或从累加器中取出。

如果使用的功能有多个输入，差别才很明显。第二个参数后的绝对参数并不单独起一行，而是在功能名后，按正确的顺序，用逗号隔开。

在PC WORX中编辑

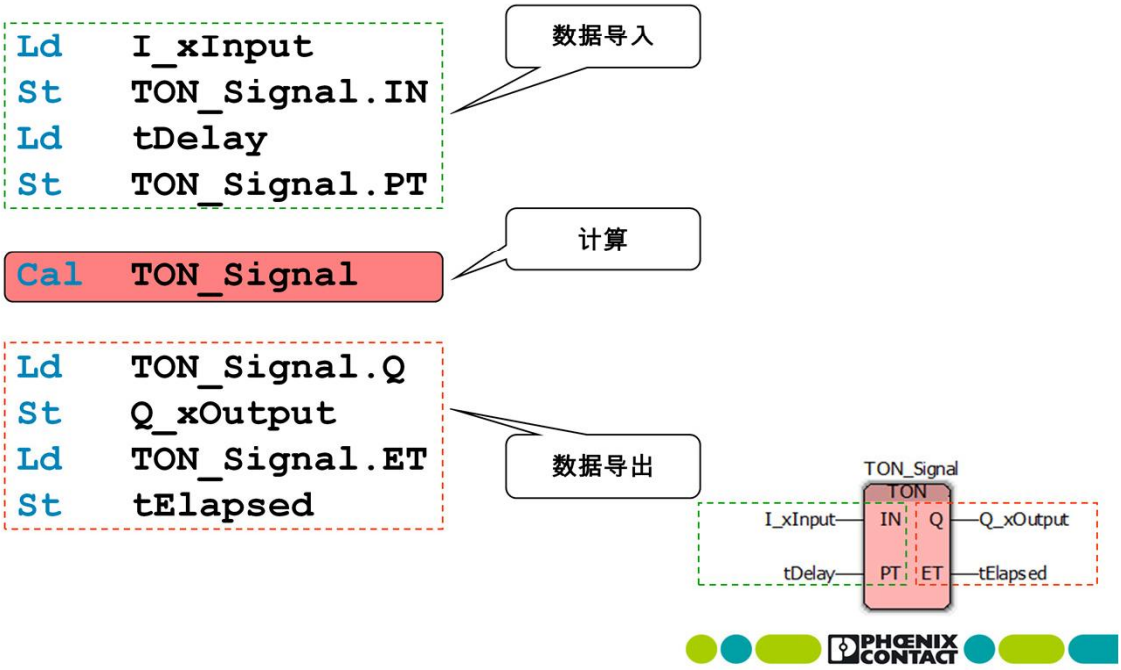


在PC WORX中编辑功能可通过键盘输入功能句法来完成。如果不知道参数、功能的拼写或变量的顺序，则可通过编辑向导在程序中插入句法模板。

目标变量和待传递的变量被编辑向导作为注解插入，且必须被变量替代。

特别是使用编辑向导不能完成功能的嵌套。那么在空行中查看句法，然后才进行实际编程，这被证明是有助于操作的。

功能块调用

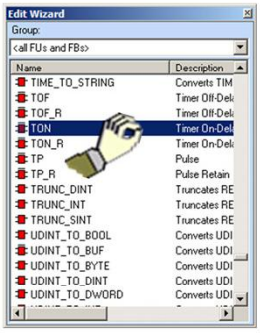


用指令表实现功能块的调用与其他所有语言一样，分成三个步骤：

1. 为输入参数提供数值（数据导入）；
2. 执行块的功能性，如必要的话，使用保存了的数据（计算）；
3. 通过输出参数将计算得到的数值保存在所创建的变量上（数据导出）。

在PC WORX中编辑

通过编辑向导插入




```
LD   (* BOOL *)
ST   TON_Signal.IN
LD   (* TIME *)
ST   TON_Signal.PT
CAL  TON_Signal
LD   TON_Signal.Q
ST   (* BOOL *)
LD   TON_Signal.ET
ST   (* TIME *)
```

用户编辑后

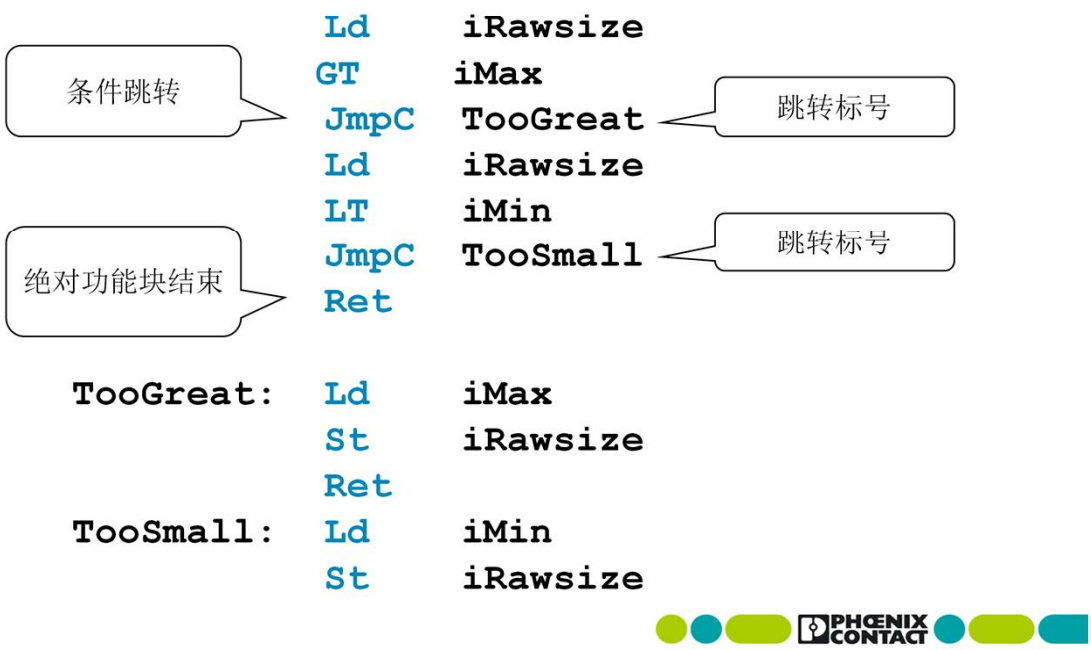
```
LD    xInput
ST    TON_Signal.IN
LD    TIME#250ms
ST    TON_Signal.PT

CAL   TON_Signal

LD    TON_Signal.Q
ST    xOutput
LD    TON_Signal.ET
ST    tElapsed
```

与功能一样，在PC WORX中编辑功能块可通过键盘输入FB句法来完成。但必须牢记一点，即功能块不同于功能，必须要实例化。这可以通过变量表或通过变量声明对话框手动完成。FB类型可确定为数据类型。

如果用编辑向导添加一个FB至程序中，则与采用图形化语言一样，打开声明对话框执行实例的声明。至于功能，需要用变量替代的调用的句法部分以注释插入。

通过编辑向导不能执行嵌套FB的调用。向导只显示单个调用的句法。

条件代码执行
JMP | RET



跳转操作符*JMP*和功能块结束操作符*RET*考虑了与标准IL顺序相偏离的执行顺序，这样就考虑了代码的条件执行。对*JMP*而言，必须确定跳转地址。跳转标号无需声明。作为目标，它可单独成一行，或如上所示，放置于指令的前面。

修饰符*C* 考虑了操作符*JMP, RET* 和 *CAL*的条件执行。

修饰符*N* 可以实现对执行条件取反。

LD - 梯形图

第16讲

内容

本章节介绍了用梯形图进行编程时所需要的语言元素及其应用。这包括触点和线圈以及电源轨线。

用梯形图进行编程时，如果需要使用非布尔量数据类型，则变量，功能和功能块的使用与在FBD中一样。有关编辑的信息，请参考*FBD-功能块图*章节。

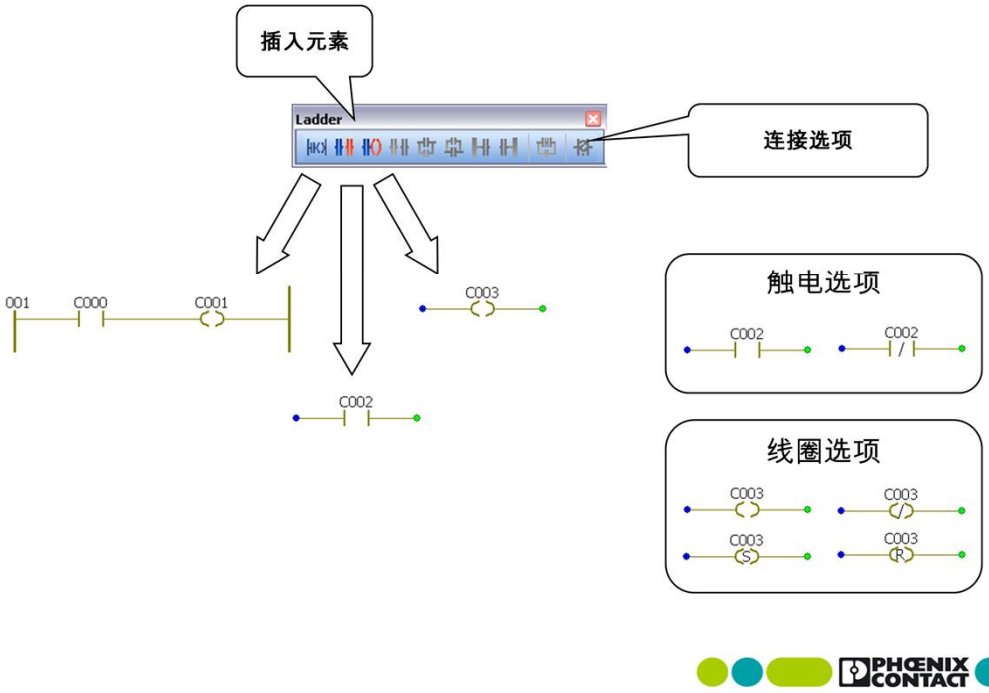
注意!

信息

提示

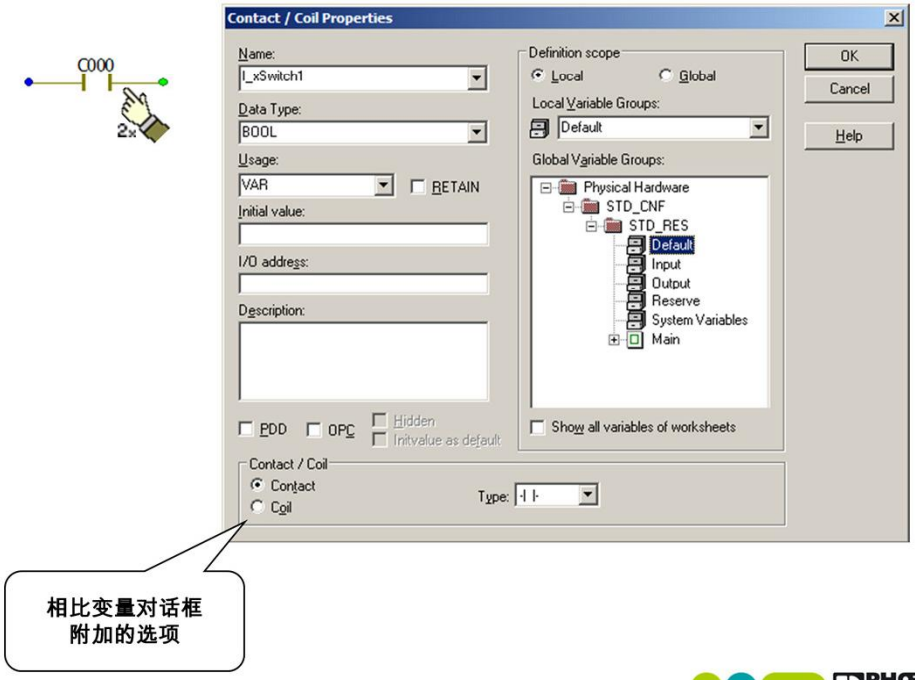
基本元素

梯形图编辑



梯形图基本元素可通过LD菜单栏插入和编辑。使用这些按钮，也可以扩展和完成已有的网络。根据工作表中所选元素，启用相应的按钮。

触点 / 线圈对话框

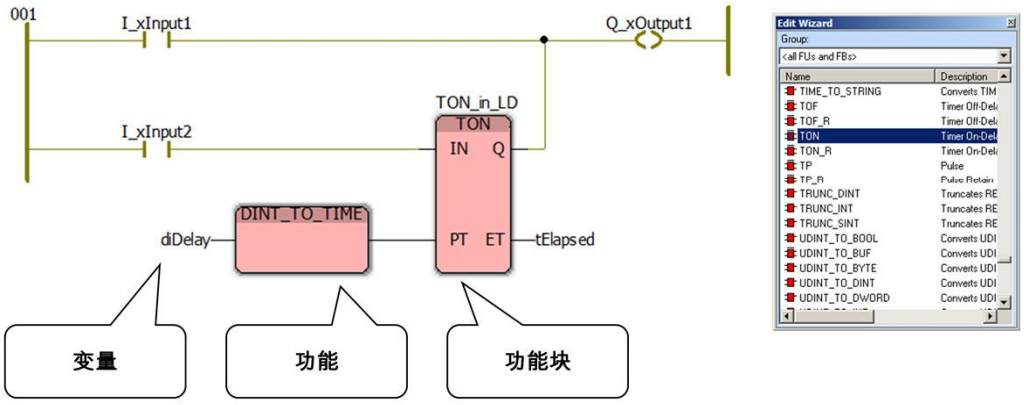


设置触点/线圈属性的对话框与标准变量对话框只有细微的差别。此外，还提供了设置LD元素的控制元素。此外，只能选择那些对布尔参数存取的数据类型。

除了*Bool*数据类型，这也包括了其它基于位的数据类型，对于一个隐式位的存取的情况，如：`I_wFeedback.x13`，这样则是存取*Feedback*输入过程数据字的第13位。也可以使用用户自定义的数据类型的元素（参见*用户自定义数据类型*章节）。

扩展编辑

在梯形图中的功能和功能块



由于是使用相同的编辑器，功能块图的元素（变量，功能和功能块）可以在梯形图中使用，而不存在任何问题。有关功能块图的编辑和处理详细信息，请参考*FBD – 功能块图*章节。

两种语言过多地掺杂在一个网络中将会导致不能被编译器清楚地理解，从而产生错误消息。

SFC - 顺序功能图

第 17 讲

内容

本章节介绍了使用IEC 61131 语言——*顺序功能图*在PC WORX 中完成图形步链编程。首先，介绍了基本结构，之后介绍了编程的基本元素（步，动作和转换）以及编写它们的不同的方 法。

需 

转换和动作的编程是用其它的IEC 61131 语言来实现的。因此，用户至少掌握一种编写这些顺序功能图元素的语言。

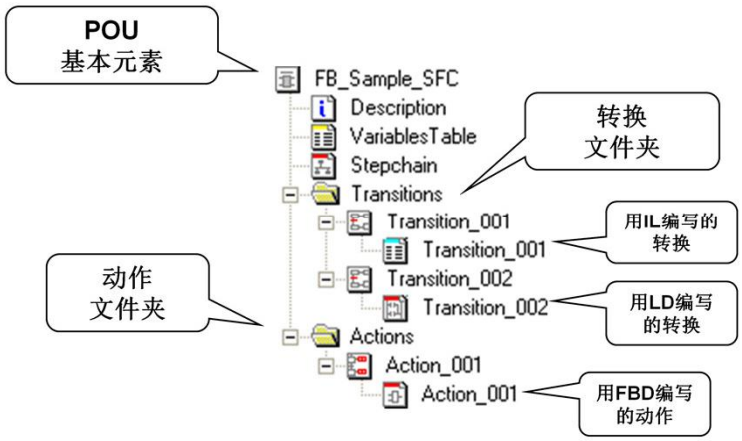
注意!

信息

提示

顺序功能图的基本结构

项目树中的顺序功能图 POU

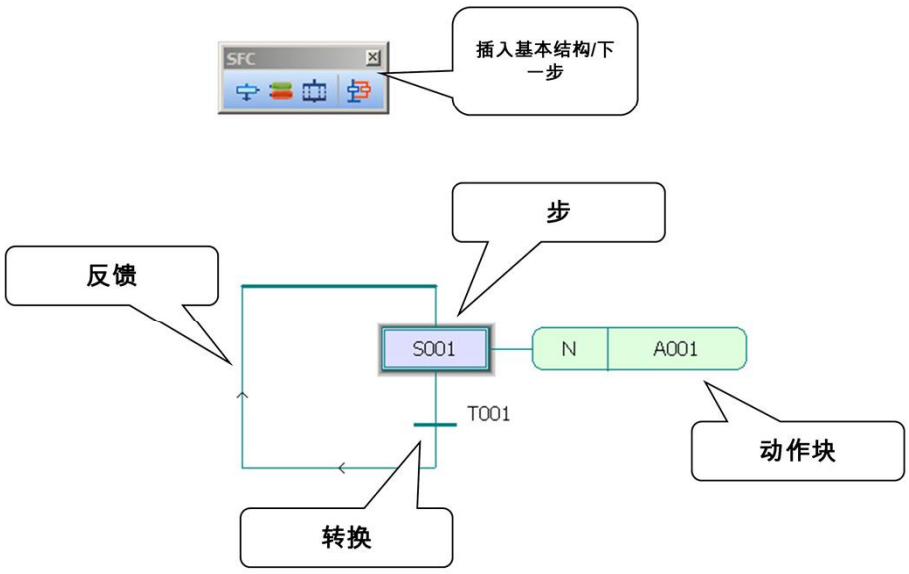


上例显示了在工程树中用顺序功能块编写的功能块。

步链不再是POU类型， 顺序功能图可以用作类型为*功能块*和*程序*的POU的编程。

与采用其它IEC 61131 语言的POU结构不同， 顺序功能图包含了其它的元素。除了在SFC中编写实际步链的文本表，变量表和代码工作单外，还有两个包含已有转换和动作的文件夹。

基本结构

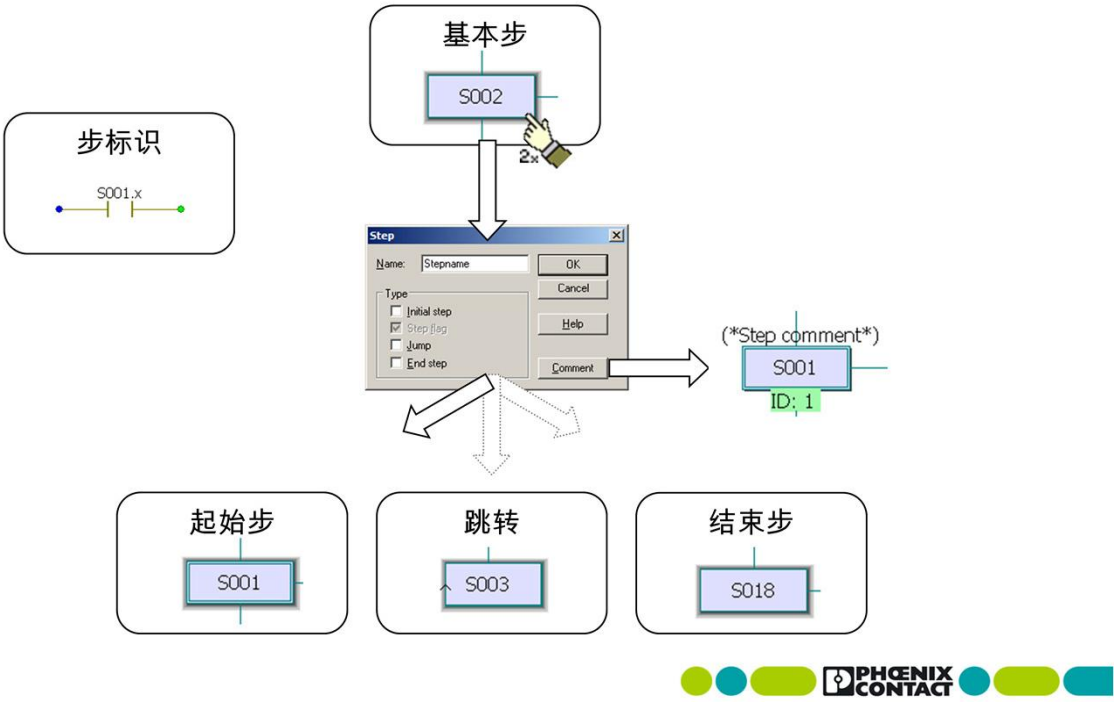


在设置插入标识后，通过上图所示按钮将基本结构插入到步链工作表中，基本结构是由两个基本元素组成：步（这里：S001 ）和转换（ 这里： T001 ），其中有一个动作块（这里： A001 ）分配给了该步。所有元素名都可以改变。

一个步链也可以由一系列的步和转换构成，而不需要动作（没有动作的步即为等待步）。

基本元素

步



第一步是基本结构的一部分，它是自动插入的。之后，系统自动只插入标准步。通过步对话框，可为每个步输入注释。

但是，只有步标志<步名>.x 是自动为步链中的步声明的。这个参数表示状态，即步的行为，并可读出，在特殊需求下，也可写入（根据IEC 61131 是不允许的），它在一个POU内有效。

通过步对话框，也可以对步进行设置，以满足特殊要求所需的版本。

初始步

每一个步链（当超出一张工作表时）必须至少有一个定义了的初始步。如果起始是从并行分支执行的，则每一个并行分支需要一个初始步。动作块可以分配给初始步。

跳转

与其它类型的步不同，跳转并不表示一个过程状态。它得看作是直接跳转到跳转名上所示的目标步。对于跳转，它必须与另一个步同名，此例外在SFC中是适用的。

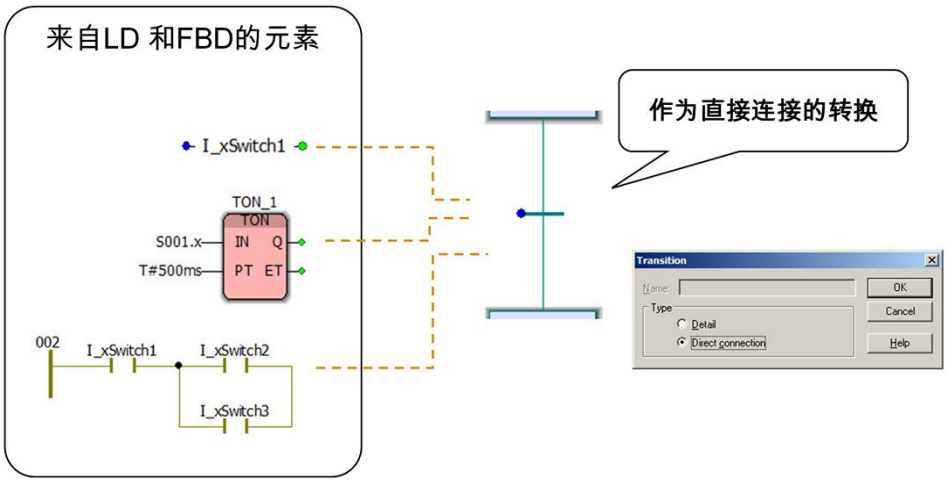
结束步

对于程序序列，此跳转表示一个死胡同。如果一旦到达，步链的重新执行只能通过操作（置位/复位步标志）步链来达到。

以结束步结束的步链可用于初始化，动作块可分配给结束步。

对于跳转及结束步而言，步链结构的连接得中断，该变化在后面不可逆。

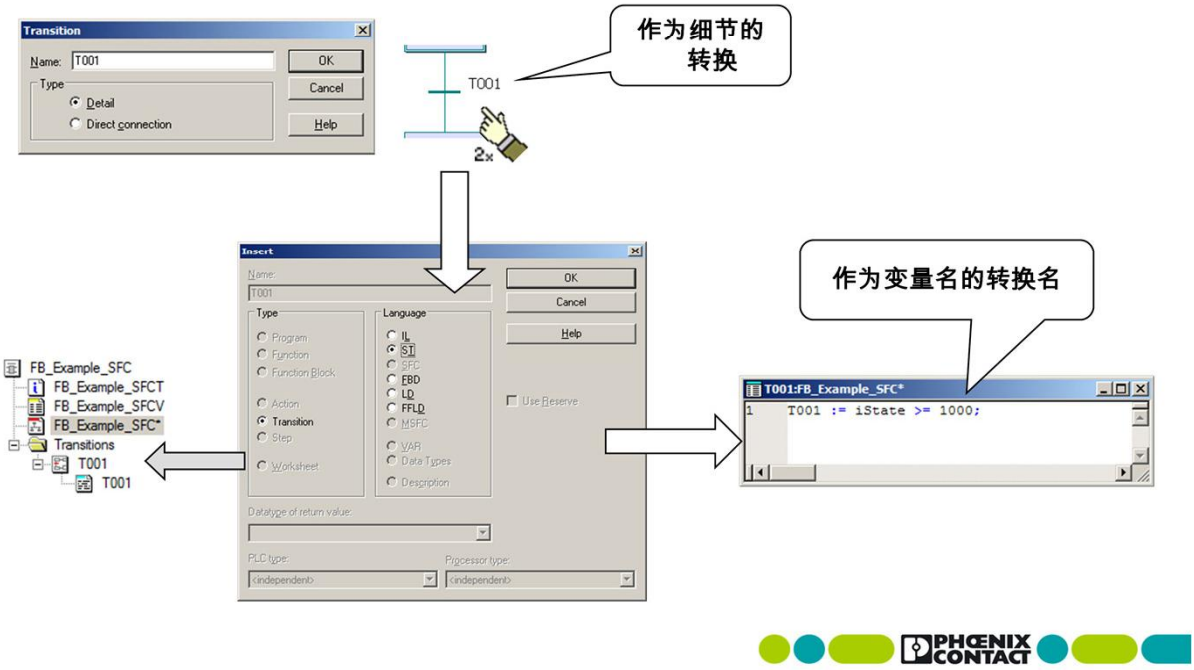
转换



直接连接

对于步链接口上的这类编程，通过属性对话框将转换切换至*Direct connection（直接连接）*。该转换不再命名，但收到一个与功能和功能块相连的终端点，该点可分配一个布尔参数以执行转换需求。为此，如上所示，可使用梯形图和功能块图元素（变量，触点，功能和功能块）。

转换细节



在步链中，转换执行了从一个过程状态转入到下一个状态的条件。有两种方式编写这些条件。

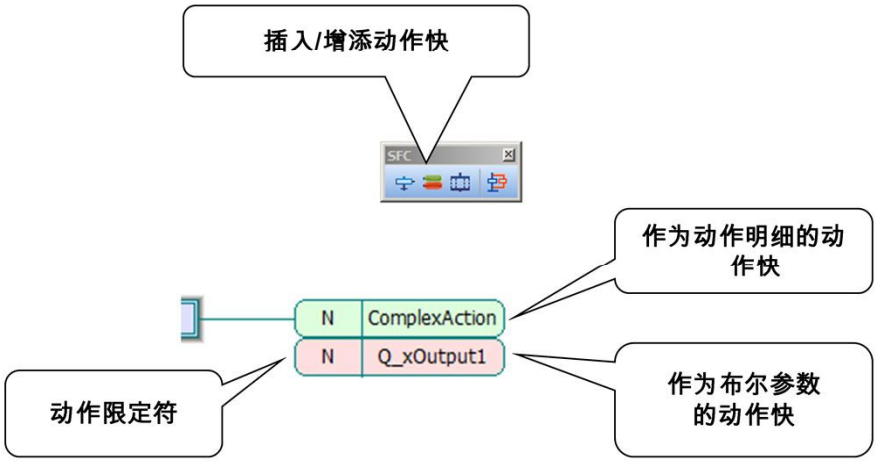
转换细节

转换细节表示了用一种IEC 61131语言编程（除了SFC）的一个或多个工作表。在修改了转换名后，双击转换名打开新元素插入对话框。当选择了语言，转换就被插入到POU转换文件夹中。同时，打开第一张工作表。转换中的编程须创建转换需求，这类似于功能的返回值，将转换名保存在一个变量中。与功能名不同，该参数并不是已保存在变量对话框中，而是要手工输入。系统不过是识别出正确输入的名，然后阻止通常为变量声明而启用的控制元素。

对于后面的调整，转换明细可从步链中打开，或通过工程树中相应的元素打开。

在工程树中可复制明细，以用于转换明细相似的编程。在改名之后所需要做的仅是调整转换名和对程序进行细微改动。

动作块



动作块影响过程状态，正如步表示过程状态。 它们可以作为动作明细执行 （标准颜色：淡绿）。这意味着它们可以在单独工作单中进行编程，或者用于控制布尔参数（标准颜色：浅粉色）。该使用可在动作块的属性对话框中设置。

除了步的状态外，动作限定符对动作的执行非常重要。除了*N*这个用得比较多的字符外（不存储，只要步激活，动作就激活），根据IEC 61131 3 ，所有其它可用的字符有：

动作限定符

S **置位** 该动作将持续直至其明确被复位（即使相应步变为不活动了）；

R **复位** 终止前面用S字符启动的动作的执行；

L **时间限定** 在激活相应步后，动作将执行持续一个设定的时间或至该步变为为不活动状态；

D **时间延迟** 当步激活时启动延迟，当设定的时间到达后，执行该动作直至步被取消激活，如果在设定时间到达之前，该步变为不激活状态，则该动作不执行；

P **脉冲** 当步变为激活/不激活时启动，并执行一次；

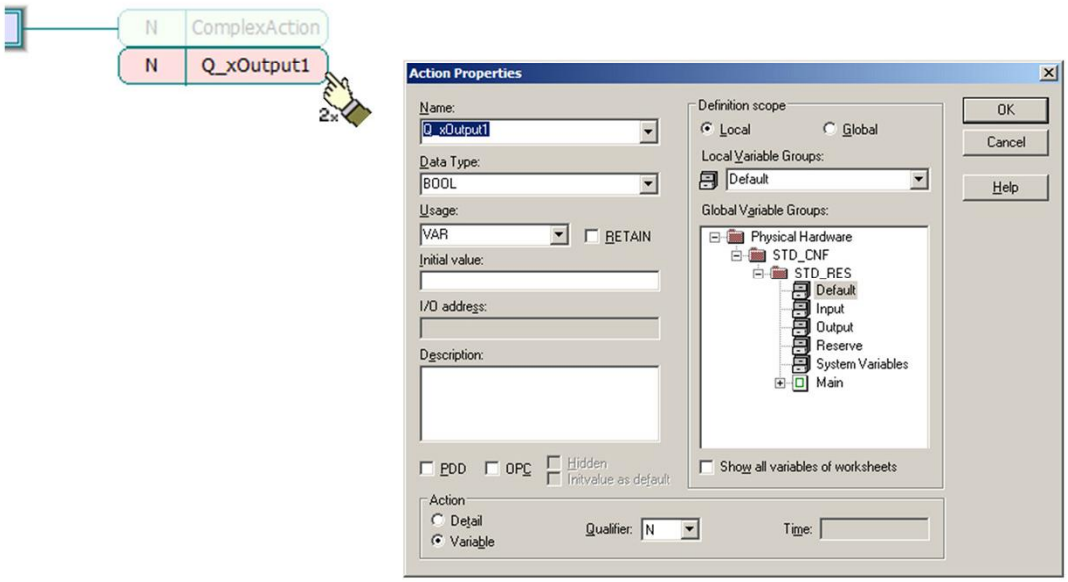
SD **保存** 动作执行直至复位，但当延迟时间到达后才执行。即使是步激活的时间少于时间延迟也执行；
+时间延迟

DS **时间延迟** 动作执行直至复位，但当延迟时间到达后才执行。与前面的符号不同的是，如果时间延迟到达之前步就变为不激活状态，则动作不可能执行。
+保存

SL **保存** 该动作执行持续一个设定的时间，即使步激活的时间较短。用字符*R*复位该动作。
+时间限定

时间须根据时间常量的标准设定。

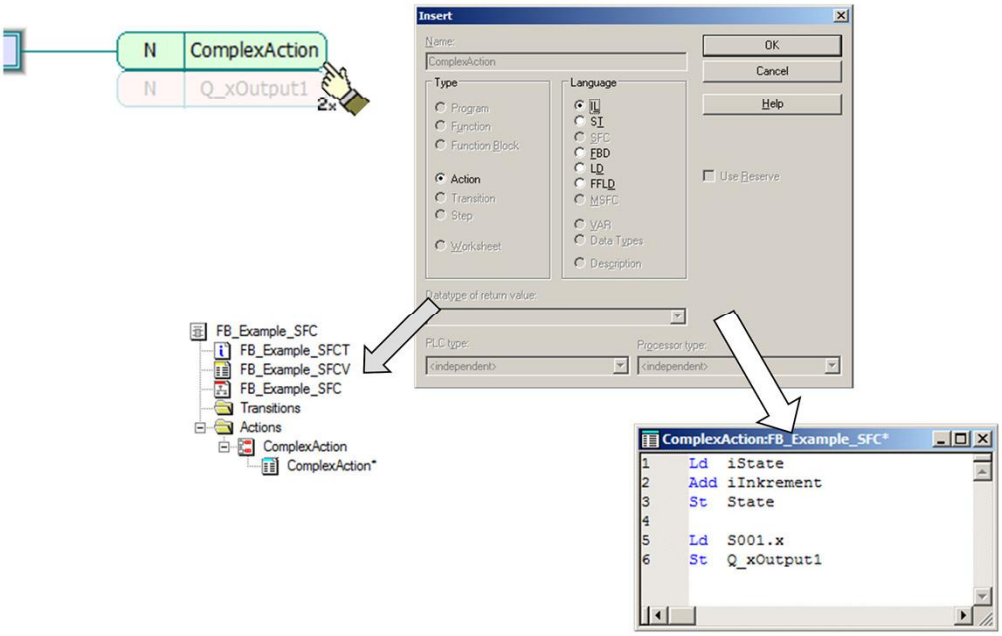
动作变量



一旦动作作为变量添加到步时或动作在动作块属性菜单中被设置为变量，则必须根据已知的程序对变量进行声明或调用。

为此，如在梯形图中一样，布尔变量或布尔参数可被隐式存取单个位使用，如在字、字节或双字变量中的位或结构和字段变量的布尔参数。

动作明细



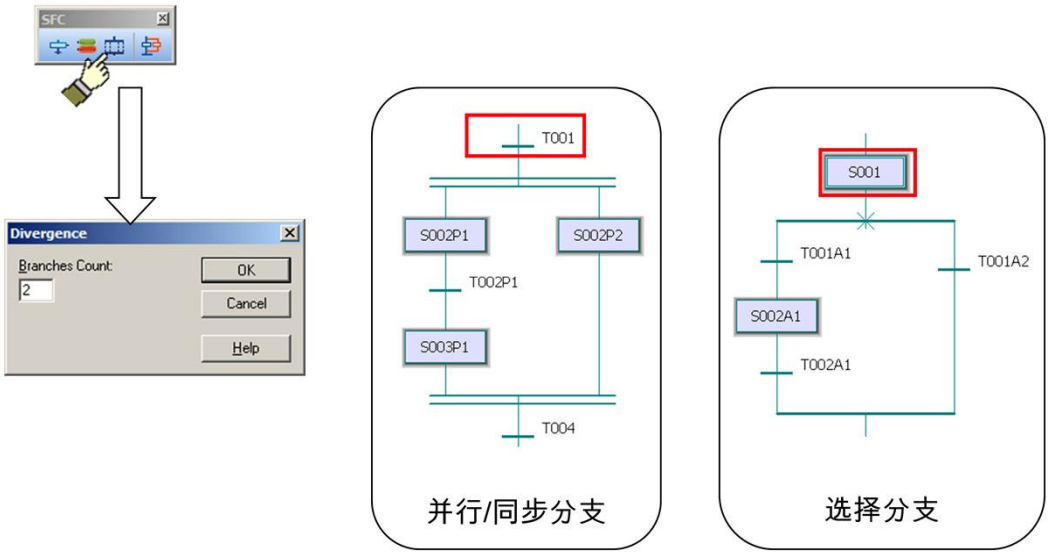
如果动作块用作动作明细，复合动作名须在属性对话框中通过右键快捷菜单进行编辑。这里，复合动作是指超过控制单个布尔变量的所有动作。

一旦分配了名，明细可通过双击创建。在打开的对话框中，只有编程语言可供选择。提供了顺序功能图除外的所有IEC 61131语言。随着首个动作明细工作表的打开， 在POU*动作*文件夹中创建了相应元素。现在，可以在工程树中选择工作单，或者使用动作块直接从步链中选择。

在工程树中可复制明细，以用于动作明细相似的编程。在改名之后所需要做的仅是调整动作名和对程序进行细微改动。

对于以上的明细，步标志*S001.x* 用于激活一个布尔变量，使该步进入活动周期。也可以使用一个带有相应布尔量和动作限定符*N*的附加动作块。

分支



IEC 61131-3 顺序功能图考虑了同步和选择分支的创建。

同步分支

要插入一个同步分支，必须选择开始分支前的步。可以插入的分支的数量视工作表的大小而定。分支独立执行。如果所有分支的结束步都激活，且满足成组转换的切换需求，这才成组执行。

在PC WORX中，在同步分支中插入跳转和结束步被视为不合法的编程而被拒绝。

选择分支

要插入一个同步分支，必须选择开始分支前的步。可以插入的分支的数量视工作表的大小而　　定。初始转换条件满足了的分支才会被执行。如果同时有多个条件得到满足，则根据在图形工作表中的执行顺序进行，即执行的优先级自左向右排列。

在选择分支中可以将跳转和结束步集成在一起。

也可以顺序地插入同步分支与选择分支。对于选择分支，所要插入的分支下面的两个转换必须都被选中；对于同步分支而言，是两个步必须被全部选中。使用SFC工具栏来插入分支。

用户自定义数据类型

第18讲

内容

PC WORX可使用很多IEC 61131-定义的数据类型来声明变量，此外，IEC还为兼容的编程系统提供用户自定义数据类型声明。这些数据类型（UDT）是标准数据类型的汇总。
本章节介绍了这些数据类型的不同分类，如何进行声明，以及在程序中如何使用基于这些数据类型的变量。

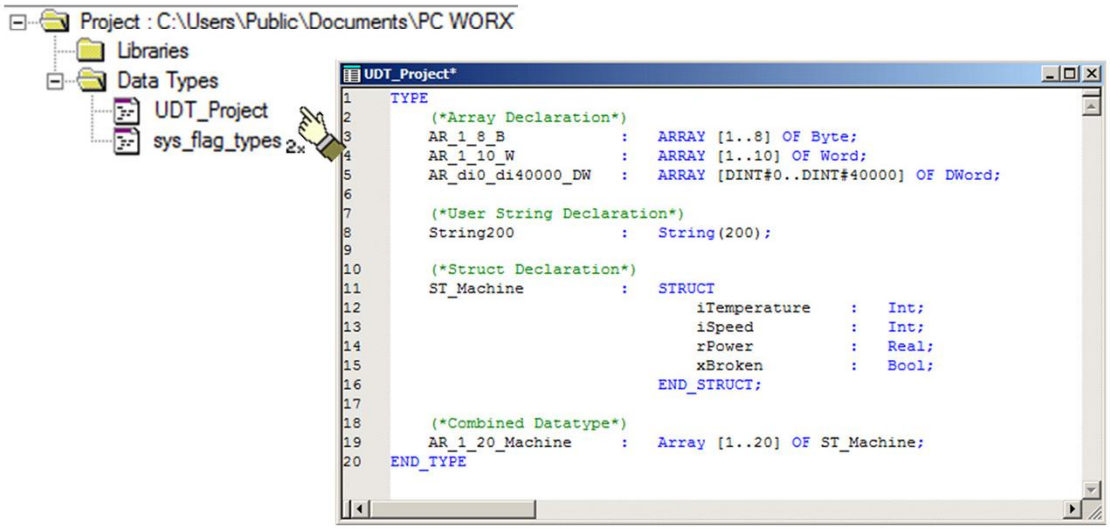
用户自定义数据类型的声明以及由此而形成的复杂变量的使用是有效的，尤其是与结构化文本高级专门语言程序结构相结合时更为有效。尽管如此，介绍了所有语言的编程示例。

注意!

信息

提示

用户自定义数据类型工作单



用户自定义数据类型声明是在*Data Types（数据类型）*文件夹的工作单中完成的。在PC WORX中，该工作表 sys_flag_types 已含在每一个工程模板中，且是控制系统系统数据类型所需要的。工作单本身可以重新命名(*SYSTEM*)，但包含在其中的数据类型声明决不能被修正。

基本上，一个工程中的所有声明都可以在一个工作单中完成。但是，欲使任何元素都排列有序，建议创建与功能相关的单独的工作表。

数组

原则

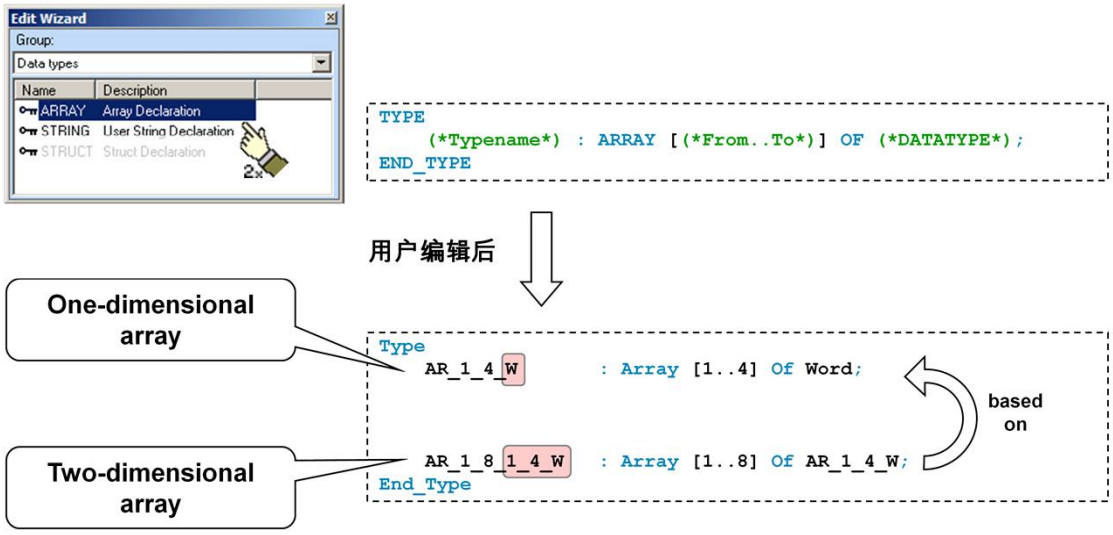
命名示例

AR_<下限>_<上限>_<基本类型>

*arrays （数组）*类数据类型的声明将一个基本数据类型元素集中在一起。该示例显示了数据类型Word的四个变量集中在一个基于新数据类型的数组变量中。

新数据类型名可以自由选择。但是，正如所有其它编程元素一样，建议使用用户自定义数据类型命名规定。上图所示为一示例。

声明

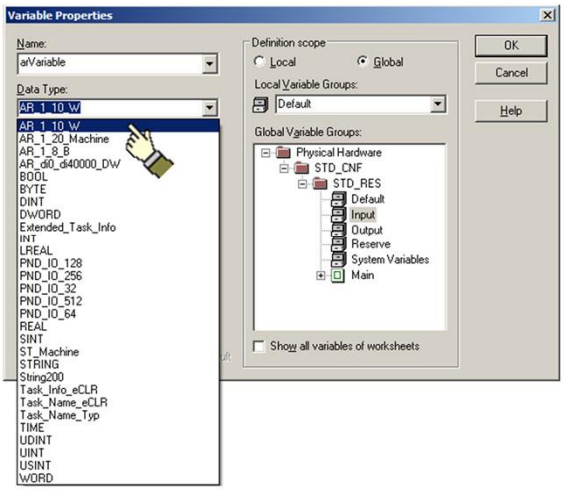


要对数组进行声明，必须使用特定的格式，它可经编辑向导以数据类型声明相应的帮助形式加以调用。必须输入该数据类型名，数组的下限和上限（用方括号以及正整数）和基本数据类　型。

·对多维数组的声明通常是遵循上一页中所描述的程序。但是，该示例显示了该声明以及后面对这些数据类型变量的使用就清晰性而言都会造成困难。

对于基于另一个的数据类型，必须遵循在工作表中的声明顺序（在PC WORX错误消息）。

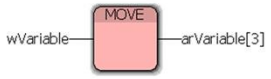
在编程中使用



两个相同维的数组变量的赋值。

```
arVariable1 := arVariable2;
```

用一个常量将一个数值
赋给数组变量中的元素



用一个下标变量将一个数值
赋给数组变量中的元素

```
Ld wVariable
St arVariable[iIndex]
```

在数据类型工作表（ Alt+F9 ）关闭或编译后，新声明的数据类型将在数据类型选择列表中列　出。两个相同数据类型的变量，这同样也适用于数组，可以相互间进行赋值。

在POU中对数组变量进行声明后，使用一个整型常量或使用一个整型变量作为数组的下标变　量，可实现对单个元素的存取。

如果变量用于访问单个元素，必须确保编程时下标不能超越数组元素数量极限。

结构体

原则

命名示例

ST_<功能>

*Structures （结构）*类数据类型的声明将一个或多个基本数据类型元素集中在一起。该示例显示了不同数据类型的四个变量集中在一个基于新数据类型的结构变量中。

新数据类型名可以自由选择。但是，正如所有其它编程元素一样，建议使用用户自定义数据类型命名规定。上图所示为一示例。

声明


```
TYPE
        (*Typename*)    :
        STRUCT
                (*Element 1 Name*)   :   (*DATATYPE*);
                (*Element 2 Name*)   :   (*DATATYPE*);
                (*Element 3 Name*)   :   (*DATATYPE*);
                (*       .           :          .   *);
                (*       .           :          .   *);
                (*Element n Name*)   :   (*DATATYPE*);
        END_STRUCT;
END_TYPE
```

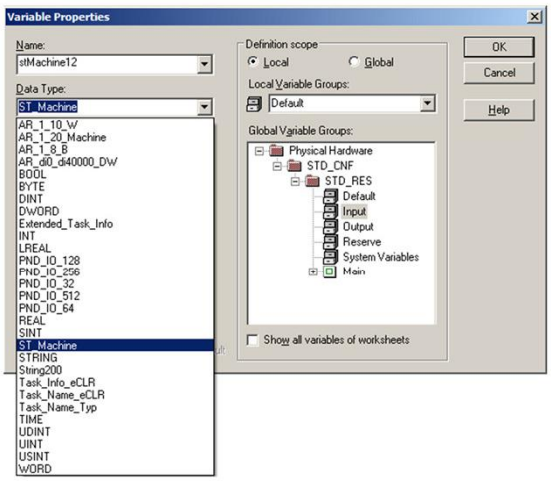
用户编辑后

```
Type
    ST_Machine  :   Struct
                      iTemperature      : Int;
                      iSpeed            : Int;
                      rPower            : Real;
                      xBroken           : Bool;
                    End_Struct;
End_Type
```

要对结构体进行声明，必须使用特定的格式，它可经编辑向导以数据类型声明相应的帮助形式加以调用。必须输入该数据类型名并要列出以上所显示的指定元素。系统也支持多维结构体的声 明。

对于基于另一种数据类型的数据类型，必须遵循在工作表中的声明顺序（在PC WORX错误消息）。

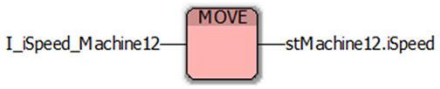
在编程中使用



两个相同维的结构变量的赋值

```
stPump12 := stPump13;
```

用指定参数将一个数值
赋给结构变量中的元素

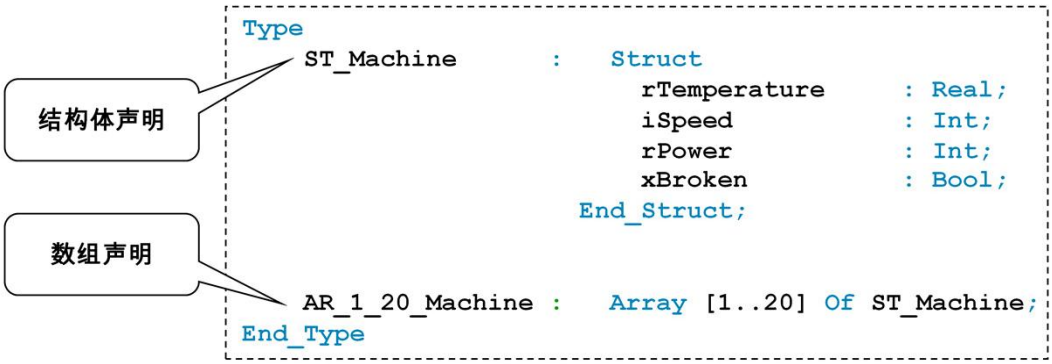


在数据类型工作表（ Alt+F9 ）关闭或编译后，新声明的数据类型将在数据类型选择列表中列　出。两个相同数据类型的变量，这同样也适用于结构体，可以相互间进行赋值。

在POU中对结构体变量进行声明后，使用命名参数可实现对单个元素的存取。

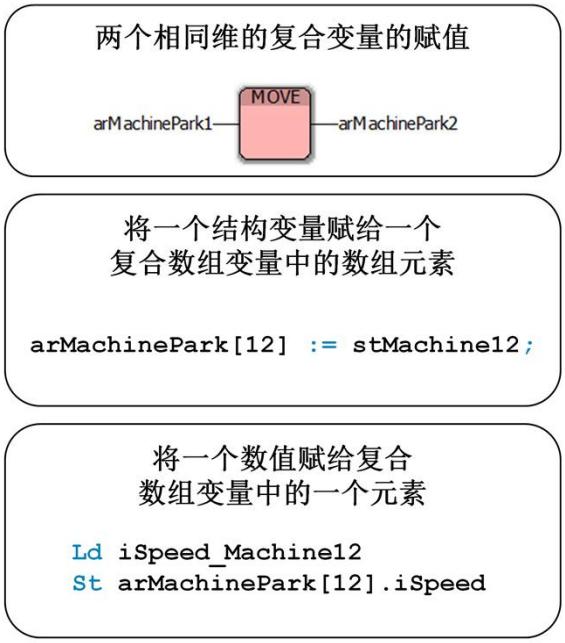
组合型用户自定义数据类型

声明和使用



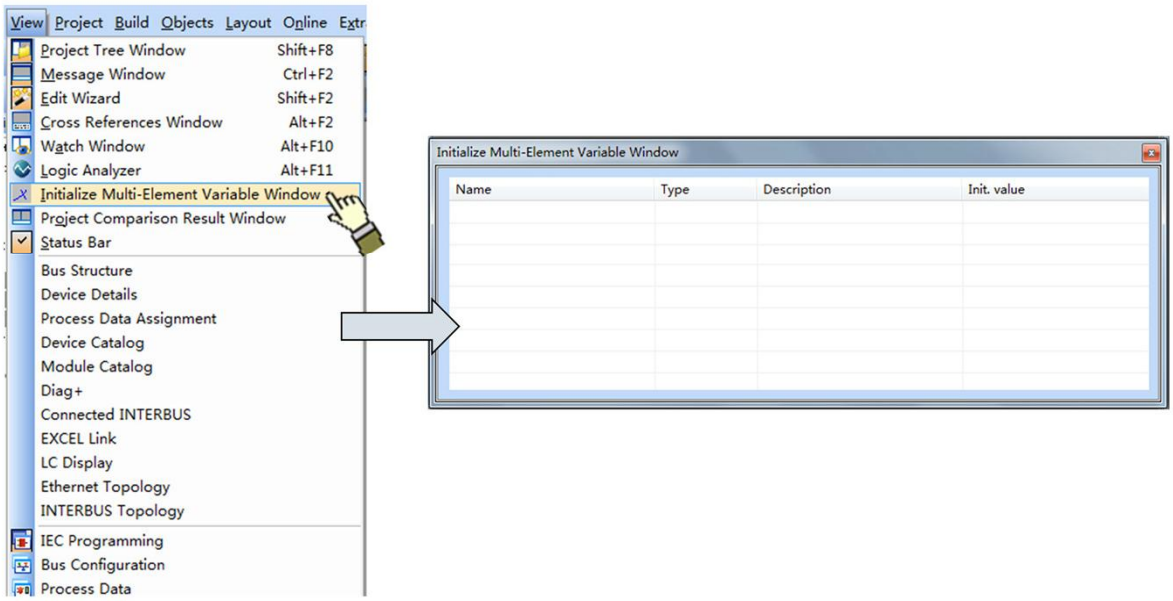
在许多工程中，数组和结构体的数据类型声明是组合在一起的。该示例显示了一数据类型的声　　明，它包含了采集泵数据的变量。数组的声明是基于前面的声明。该数组包含了泵结构的20个元素，下标从1至20。

在编程中使用

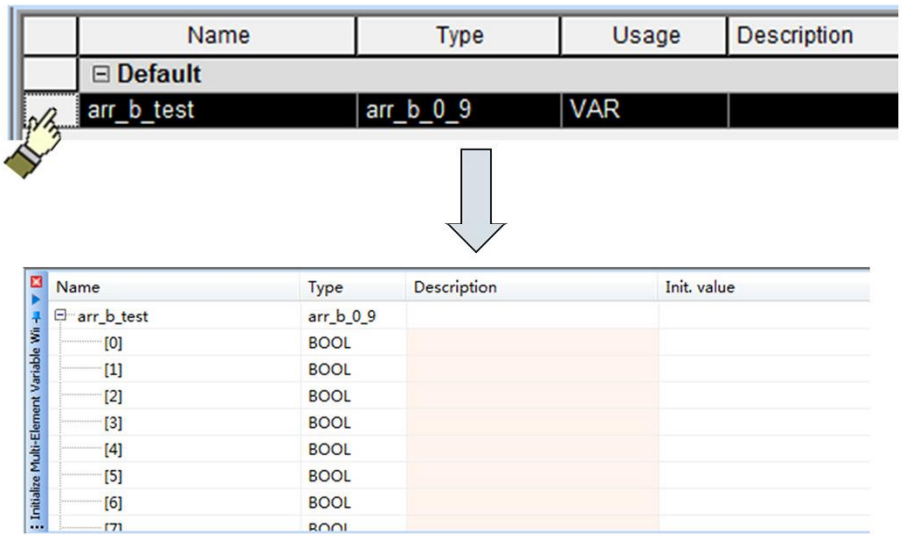


要对基于泵数组变量中的元素进行存取，则遵循“从大到小”的原则。这意味着首先要指定在数组中要访问哪个泵（方括号中的 INT 变量或INT 常量），然后指定该泵的参数（通过命名参　　数）。

初始化多元素变量窗口


View Project Build Objects Layout Online Extr
Project Tree Window Shift+F8
Message Window Ctrl+F2
Edit Wizard Shift+F2
Cross References Window Alt+F2
Watch Window Alt+F10
Logic Analyzer Alt+F11
Initialize Multi-Element Variable Window
Project Comparison Result Window
Status Bar
Bus Structure
Device Details
Process Data Assignment
Device Catalog
Module Catalog
Diag+
Connected INTERBUS
EXCEL Link
LC Display
Ethernet Topology
INTERBUS Topology
IEC Programming
Bus Configuration
Process Data
Initialize Multi-Element Variable Window
Name
Type
Description
Init. value

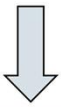
初始化数组变量


Name
Type
Usage
Description
Default
arr_b_test
arr_b_0_9
VAR
Initialize Multi-Element Variable Wi
Name
Type
Description
Init. value
arr_b_test
arr_b_0_9
[0]
BOOL
[1]
BOOL
[2]
BOOL
[3]
BOOL
[4]
BOOL
[5]
BOOL
[6]
BOOL

键入初值

Initialize Multi-Element Variable Window

Name	Type	Description	Init. value
arr_b_test	arr_b_0_9		[true, false, true, false, true, false, true, false, true, false]
[0]	BOOL		true
[1]	BOOL		false
[2]	BOOL		true
[3]	BOOL		false
[4]	BOOL		true
[5]	BOOL		false
[6]	BOOL		true
[7]	BOOL		false
[8]	BOOL		true
[9]	BOOL		false



Name	Type	Usage	D...	Ad...	Init
Default					
arr_b_test	arr_b_...	VAR			[true, false, true, false, true, false, true, false, true, false]

ST - 结构化文本

第19讲

内容

本章节介绍了用结构化文本编程的可能性，包括一些在其它语言中所提供的元素，如操作符，功能和功能块，也有用于条件和循环编程的高级语言中特有的命令。

注意!

信息

提示

结构化文本的语言元素

赋值，操作符

```
Q_xOutput1 := False;
Q_xOutput2 := I_xInput2 & Not I_xInput4;
```

请求

```
If I_xInput1 & I_xInput2 Then
   Q_xOutput1 := True;
ElsIf I_xInput2 & I_xInput3 Then
   Q_xOutput2 := True;
End_If;
```

FU 调用和参数

```
iScaled := Limit(iMin, iBase, iMax);
```

FB 调用和参数

```
CTU_Output (CU     := xPulse,
            PV     := iMaxValue,
            RESET  := xStop);

iValue             := CTU_Output.QV;
xFull              := CTU_Output.Q;
```

循环

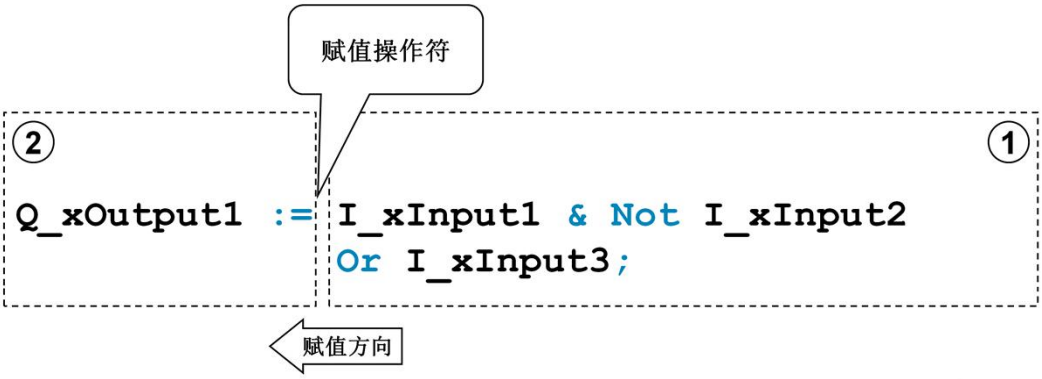
```
Repeat
   iLoop := iLoop + 1;
Until iLoop = 100
End_Repeat;
```

对结构化文本中可用的语言元素的访问大多为编辑由导所支持。除了*功能*和*功能块*组外，在结构化文本还提供*关键字*组。

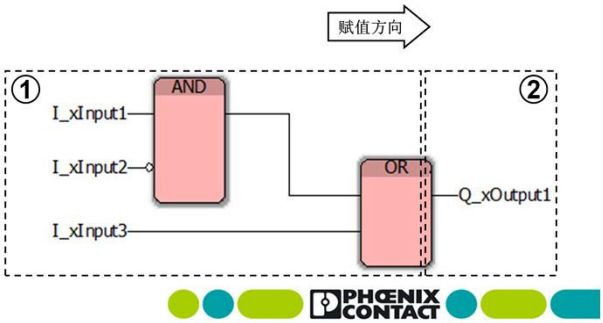
基本编程

操作符，功能和功能块

赋值和操作符



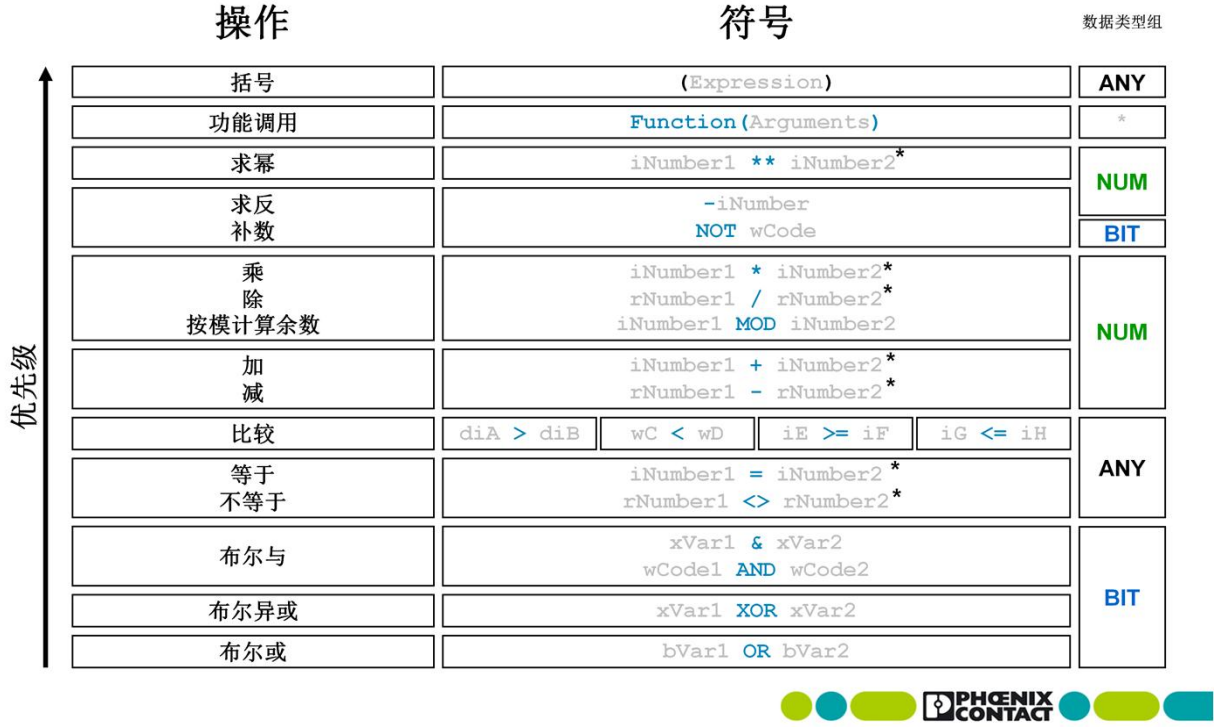
① 计算
② 赋值



图形和文本语言的主要差别在于表达式的赋值（常量，变量和计算）上，图形语言是自左向右执行，而文本语言则是自右向左执行。就其命令集、操作符和语言元素而言，这两类编程语言之间并无差别。

赋值的基础性关键元素也是基本组成总是赋值操作符:= 和编辑指令的分号。如上例所示，指令可跨越多行，以增强编程的清晰性。

操作符层次结构

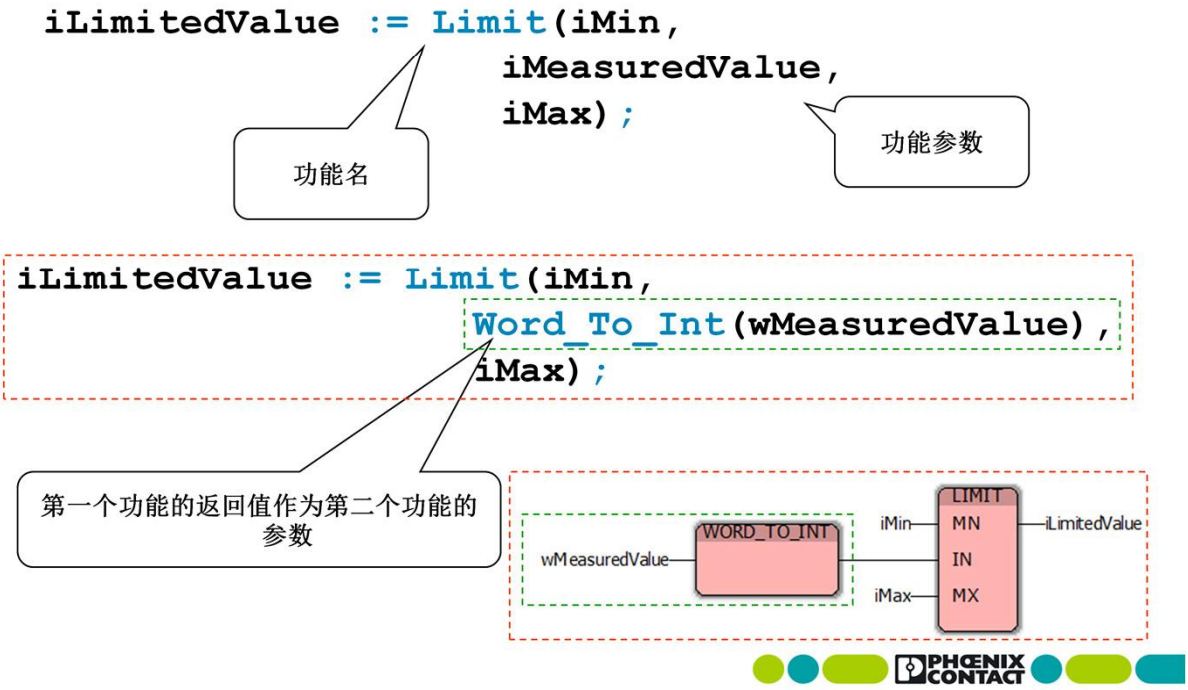


在结构化文本中所提供的操作符是按一定的顺序执行的。如果在一个表达式中存在多个操作符，这将非常重要。使用括号，用户总是能改变它们的优先级的。但是，括号也可以用于提高编程及相互关系的清晰性。

上列表中的*数据类型组*列说明了哪类数据类型或数据类型组的操作符可以使用。

•如果通过*Tools → Options → Code*激活*Enable DIN*标识符选项，那么必须输入空格键来对源代码做一个清楚的注释。

功能调用

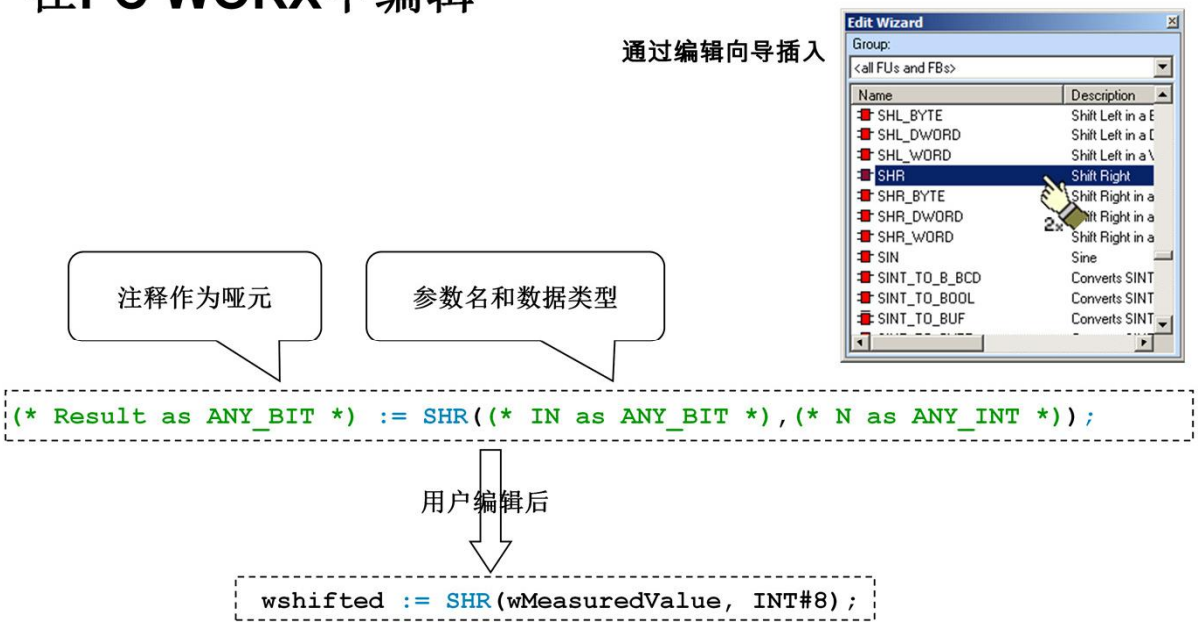


在结构化文本中功能的使用所应遵循的规则与其它语言一样。输入参数必须按参数的正确顺序进行赋值。
在功能名后，待传递的参数在后面的括号中，并以逗号隔开。如果可以排在一行中，建议按上面的模式——一个参数一行。

在大多简单的应用中，通过赋值操作符将功能的返回值赋给目标变量。

第二个例子显示了功能的嵌套。在本例中， *Word_To_Int*的结果用作*Limit*功能的第二个参数。

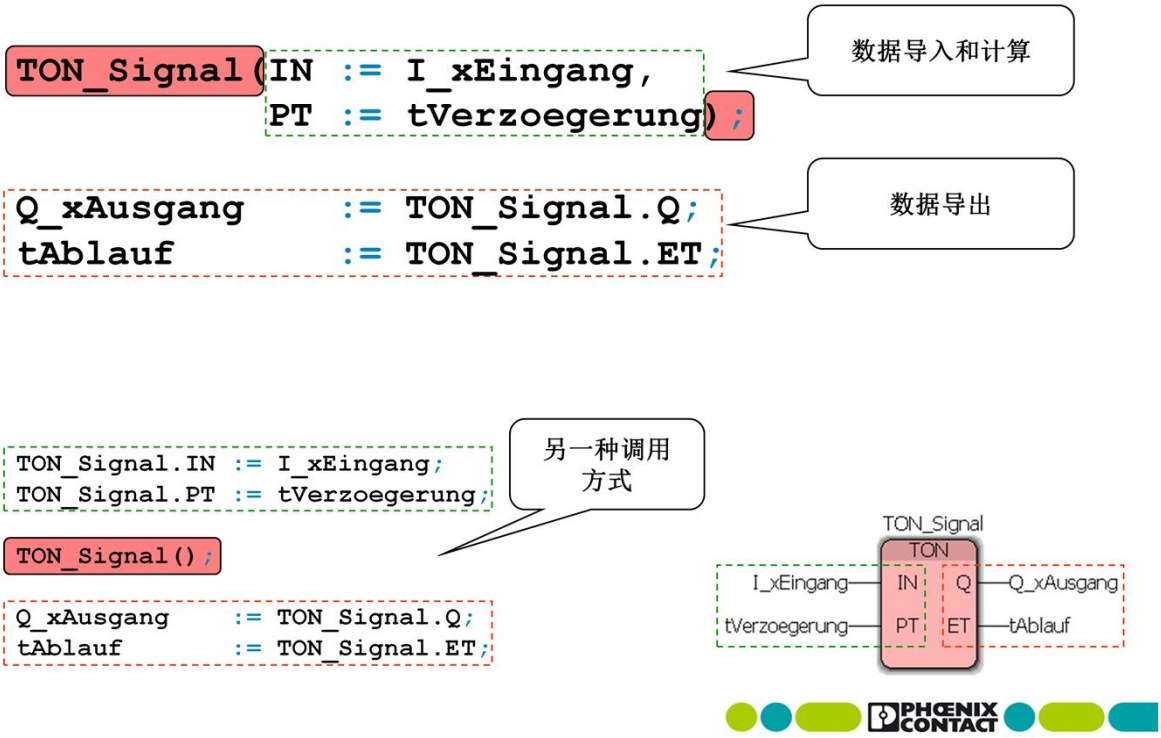
在PC WORX中编辑



在PC WORX中编辑功能可通过键盘输入功能句法来完成。如果不知道参数、功能的拼写或变量的顺序，则可通过编辑向导在程序中插入句法模板。
目标变量和待传递的变量被编辑向导作为注解插入，且必须被变量替代。

特别是使用编辑向导不能完成功能的嵌套。那么在空行中显示句法，然后复制到实际编 程中，将更具实践性。

功能块调用

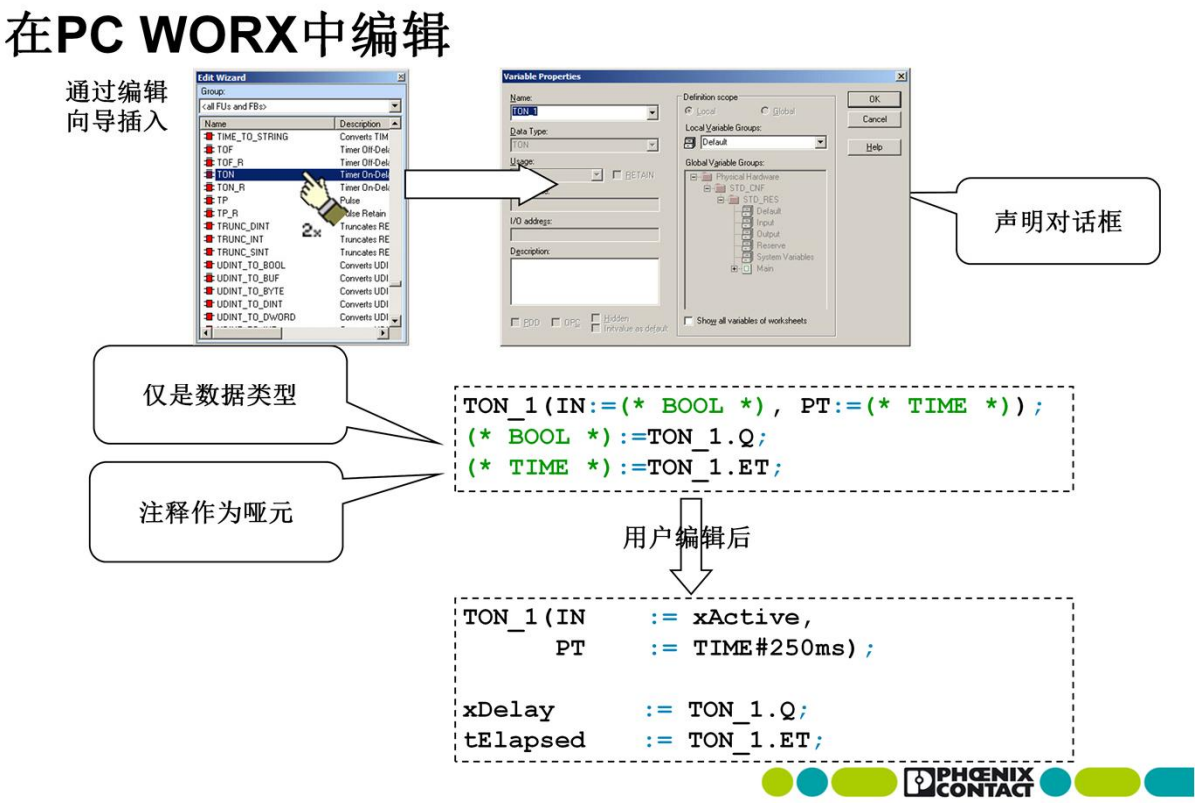


用结构化文本实现功能块的调用与其他所有语言一样，分成三个步骤：

1. 为输入参数提供数值（数据导入）；
2. 执行块的功能性，如必要的话，使用保存了的数据（计算）；
3. 通过输出参数将计算得到的数值保存在所创建的变量上（数据导出）。

采用第一个调用版本，括号内的实参与功能块实例的形参相连接。因此，实例名不必位于它们之前，这正是另一种调用方式的情形。

无论如何，在块的调用过程中，必须输入括号，即使是数据导入单独完成（参见另一种调用方式）。

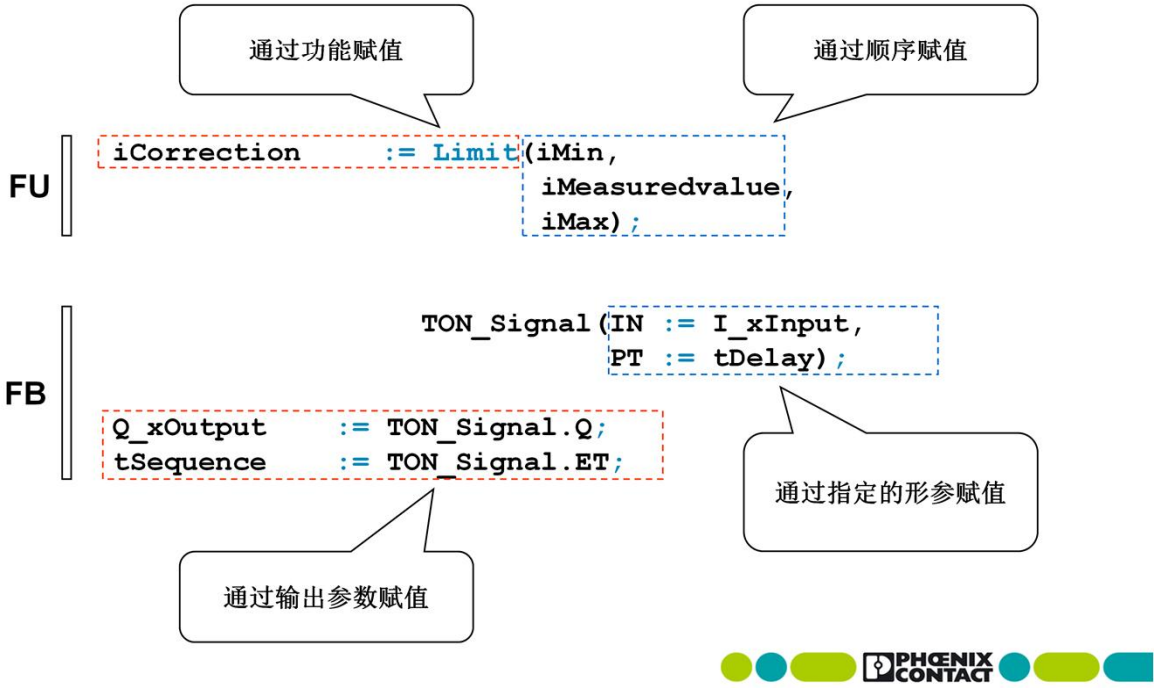


与功能一样，在PC WORX中编辑功能块可通过键盘输入FB句法来完成。但必须牢记一点，即功能块不同于功能，必须进行实例化。这可以通过变量表或通过变量声明对话框手动完成。FB类型可确定为数据类型。

如果用编辑向导添加一个FB至程序中，则与采用图形化语言一样，打开声明对话框执行实例的声明。至于功能，调用的句法中需要用变量替代的部分以注释插入。

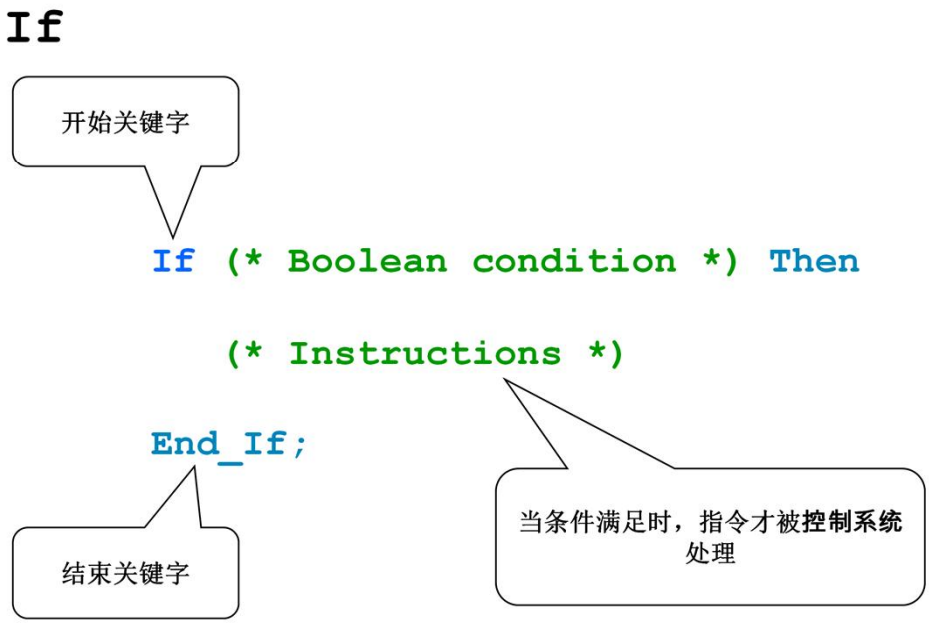
通过编辑向导不能执行嵌套FB的调用。尽管如此，使用编辑向导可插入单个调用的句法，FB实例进行声明。

比较FU调用和FB调用



上图对功能块和功能结构调用进行了比较。功能就其连接可能性是非常有限制的，参数是通过顺序进行赋值的。功能块的各种可能性都是基于对命名的输入和输出参数的存取。

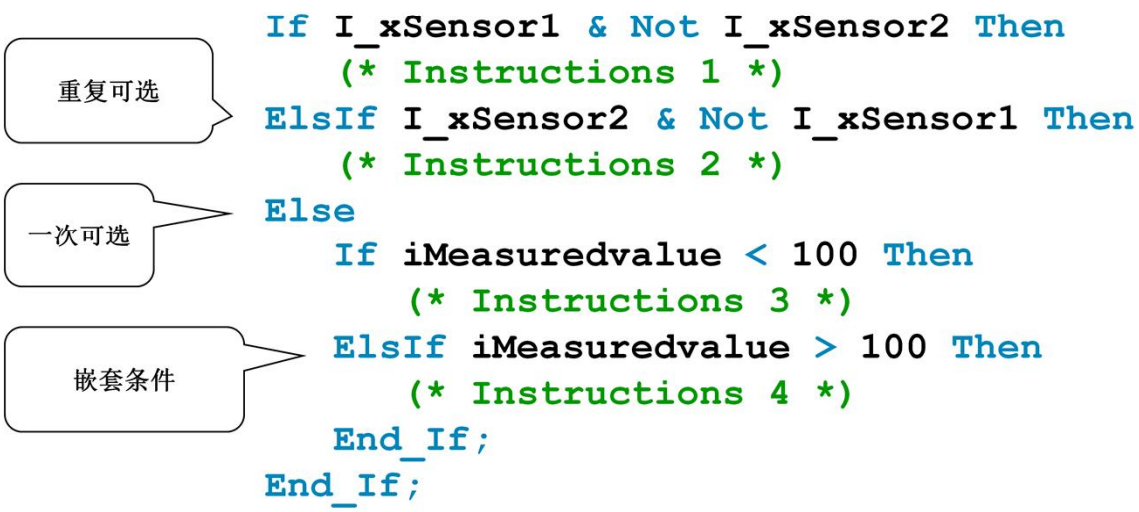
高级语言元素1
请求



关键字*If*的结构使得程序员可以根据布尔条件来执行代码。

该结构是对其它语言元素，如指令、功能和功能块的重要补充。但是，它不能用来替代 其他语言。

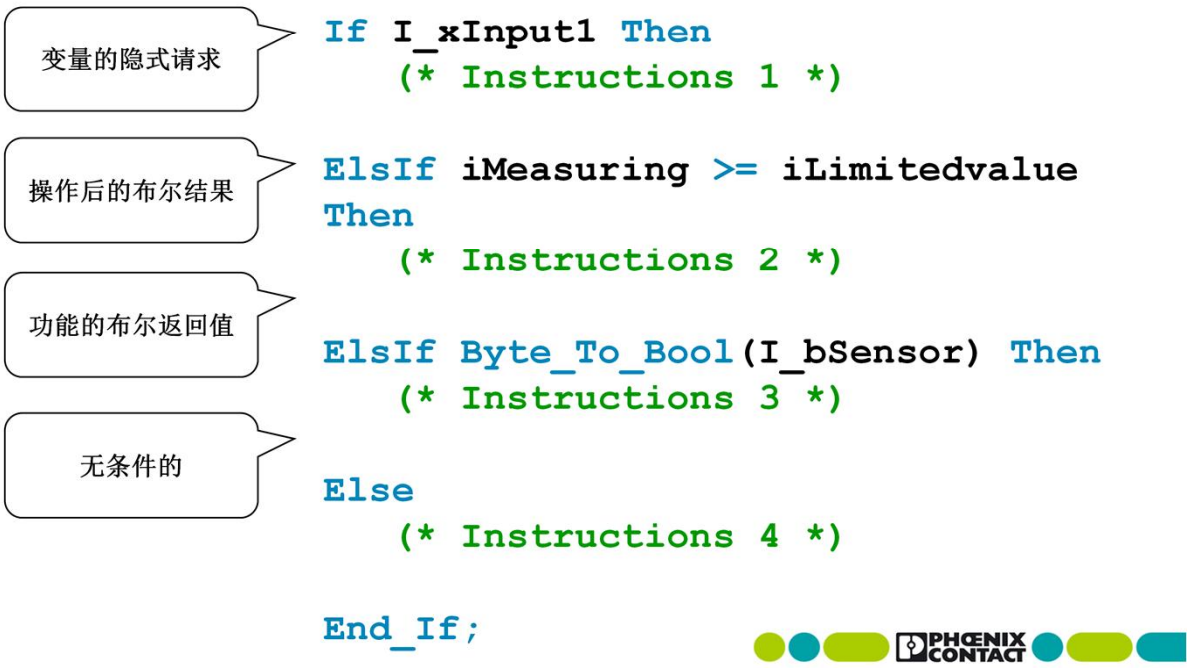
If 扩展



使用两个关键字可扩展基本结构：
如果开始的条件*If*不满足，*ElsIf*允许一个可选择的、有条件的请求。该结构可使用多次。但是，在一个*If*结构中，只有满足第一个条件的指令才能执行。
Else，与*ElsIf*不同，只能使用一次。如果在一个*If* 结构中， 经*If* 和*Elsif*，无请求完成，依赖于*Else*的指令才执行。

上图也显示了有一个嵌套*If*结构的例子。只有当*Else*激活时，第二个*If* 块才执行，而后遵循独立*If*结构的规则。

If 条件

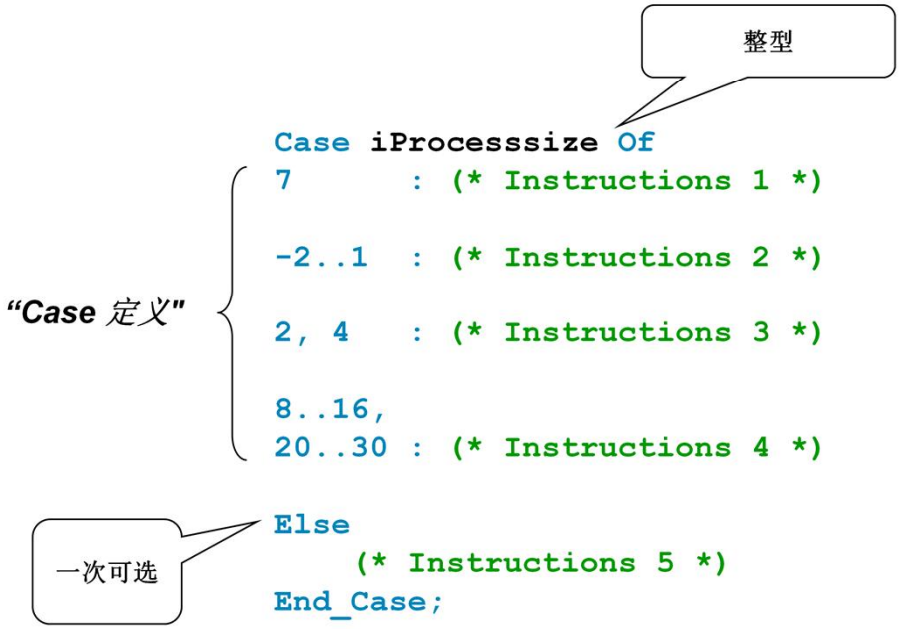


参照前面，各种编程都可用作元素和扩展的条件：

•布尔变量（使用*NOT* 进行取反请求）；
•操作的布尔结果（也可以是嵌套）；
•功能的布尔返回值；
•*Else*无条件（在一个*If* 结构中只可能使用一次）。

Case

用于过程值



由关键字*Case*构成的请求结构对*整型*数据类型的变量值加以请求，上例显示了各个值，由逗号分割的值，值的范围和后两个的组合可用作case定义。

与*If* 结构一样，待执行的指令块要求其条件（值）对应于请求的变量值。与*If* 结构不同的是， *If* 结构不必互斥（如果使用 *If* … *ElsIf* 结构），而在*Case*结构中case定义上的重叠将作为错误读出。

如果变量值与请求的值不匹配，则执行依赖于可选*Else*的指令块。

所有的值都应从INT数据类型中选择（-32.768 至 32.767 ）。

Case

用于控制值

```
Case iProcessstep Of
0     : (* Initialize *)
        If xInit_completed Then iProcessstep := 10;
        End_If;

10    : (* Execute Prozess 1 *)
        If xProcess1_completed Then iProcessstep := 20;
        End_If;

20    : (* Execute Prozess 2 *)
        If xProcess2_completed Then iProcessstep := 30;
        End_If;

(* etc. *)

500   : (* Exceptionalhandling *)
        iProcessstep := 0;
End_Case;
```

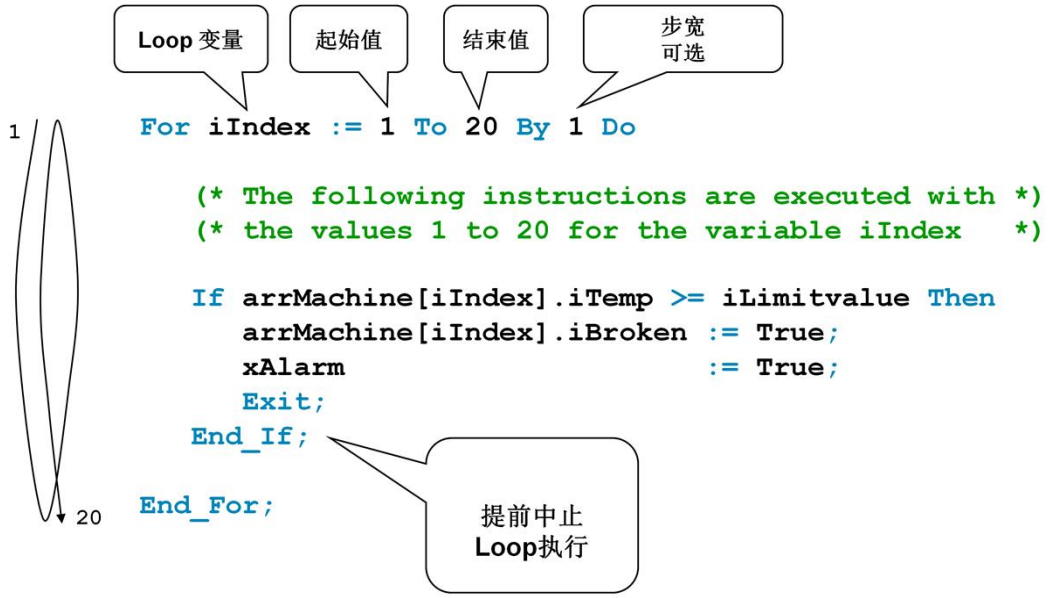
如果*Case*结构用于构建一个文本步链编程，则请求的变量可用于控制序列。其状态显示了当前的过程步骤。

在上例中，当PLC启动时，变量*iProcessstep*初始化为0，执行该值所分配的指令块直至过程通过布尔变量*xInit_completed*报告初始化序列结束。结果是， *iProcessstep*置为10，在下一个(!) PLC周期中，执行新值所分配的指令块。

这类编程的一大优点是通过结束条件将步变量置为一个值，这也可用于异常处理。

高级语言元素 2
循环

For



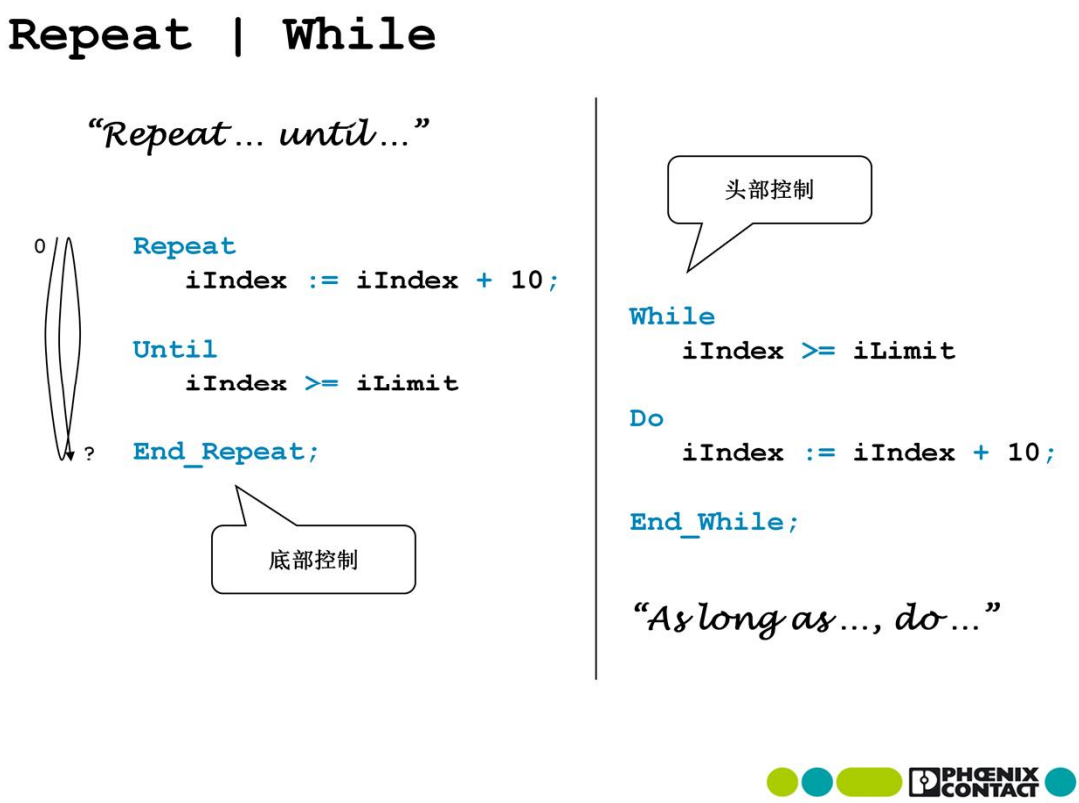
*For*循环是在一个PLC周期中反复执行指令。循环变量（这里是`iIndex`）是从起始值到结束值，按指定步宽增加。

所有值（循环变量，起始值，结束值，步宽）可从INT数据类型组中自由选择。步宽只能选择正值。

如果没指定步宽，则设置为标准值INT#1，然后，其它的值必须是INT数据类型。

一旦循环已经运行，循环变量接收到结束值加上步宽（上例： INT#21）。

循环运行的提前中止可通过*EXIT* 指令实现。



*Repeat*和*While*循环，与*For*循环不同，不是一个预确定的循环。根据一个布尔量，决定是否再次执行程序。这里，请求的次数在*底部控制的Repeat* 循环和头部控制的*While*循环之间有所区别。在*Repeat*循环中，首先是执行请求；而在*While*循环中，虽然不满足条件，但在循环中止之前，附属的指令已执行了一次。

由于PLC不能继续执行，因此，在处理和编程中必须确保循环不会不明确地执行。 此外，运行时监控，将触发一个看门狗错误。

工程管理

附录 A1

内容

本章节概括介绍了在PC WORX中的工程文件，控制器的存储器概念以及不同代码格式和使用不同控制系统的工程处理。

这篇文档描述了使用ProConOS作为操作系统的控制。100系列的控制 有与相出入的对话框与存储的概念。请阅读其用户手册中不同的部分。

注意!

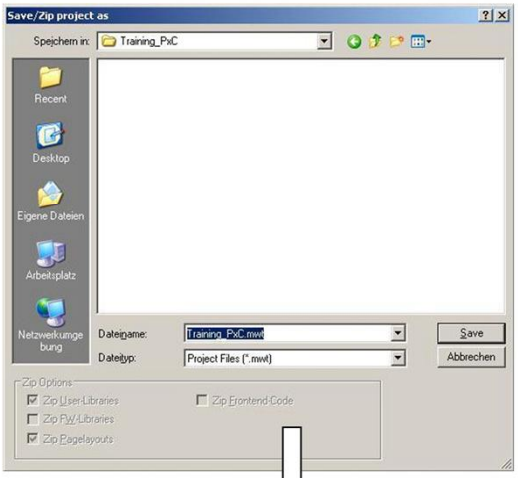
信息

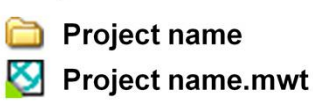
提示

工程文件

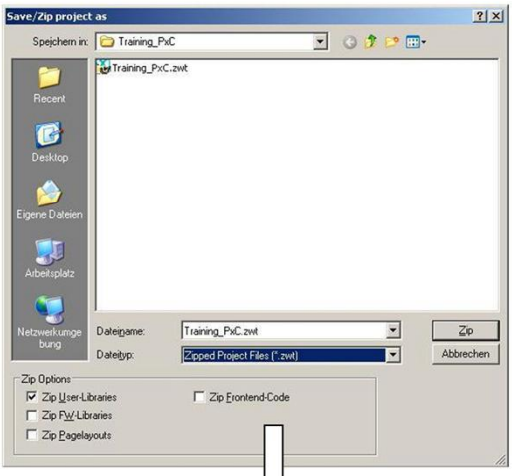
保存... / 存档

保存为 ...





存档



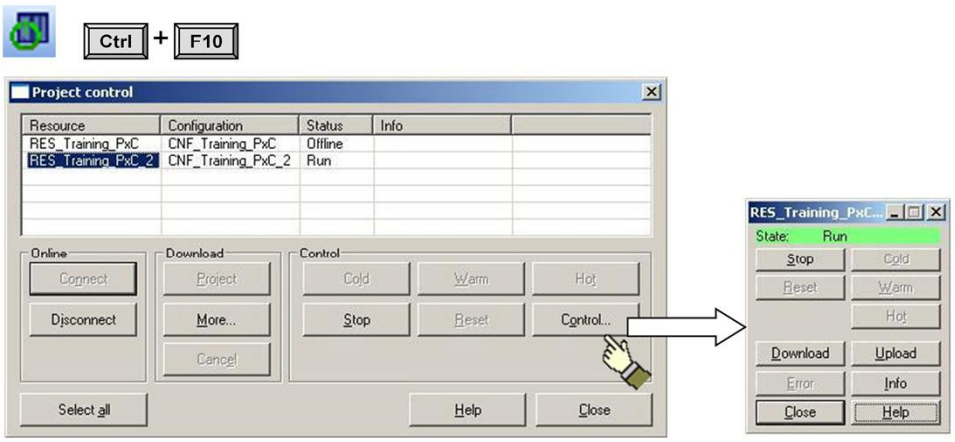
欲在PC WORX中进行处理，工程文件是解包文件，保存在PC硬盘中所选的存储位置。除了扩展名*.mwt的头文件外，这些文件包含了一个相同名的目录。

当选择了*保存为*...的选项，当前在PC WORX正在处理的工程将按所选择的名保存，这些文件将用于进一步的处理。

与其相对比，在归档时，头文件和目录组合成一个zwt文件（如果必要的话，可使用新名），但是，使用最初的工程（和名）继续在PC WORX中操作。

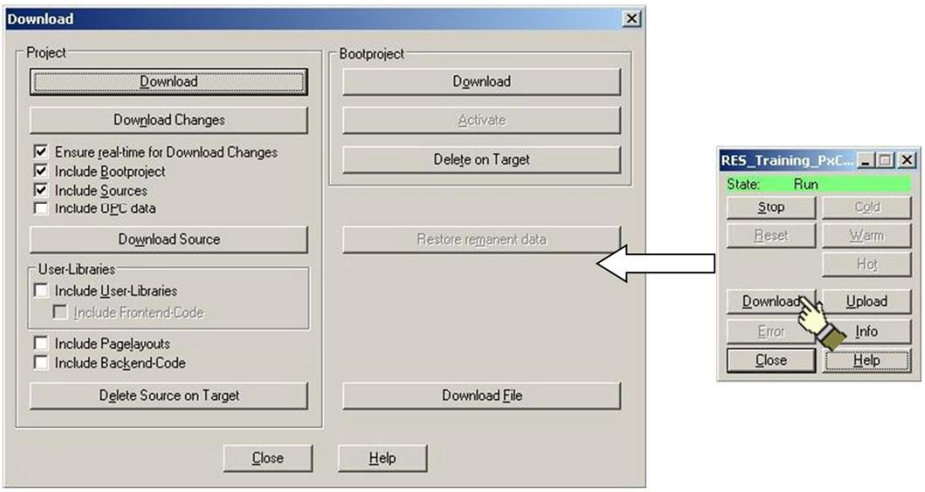
CPU控制对话框

CPU选择



在控制工程（总线组态和控制程序）成功编译后，可通过工程控制对话框将代码下载到控制系统中。如果一个工程配置了多个控制系统，则经上述对话框选择控制系统。这样，就不能同时对所有系统执行下载。只有经*Connect（连接）*建立与控制系统之间的连接，才能调用控制对话框。

临时工程 – 引导工程



工程

下载
编译了的机器码（ 总线组态和应用程序）写入易失主存储器中。

下载修改部分
在运行过程中，将程序变更（不是对总线组态的修改或是对硬件结构，包括任务的修改）传送给主存储器中（如果必要的话，如果相应选项激活具有实时性）。

包括工程
在下载到主存储器后， 在单独的传送中，引导工程写入到参数化存储器中。

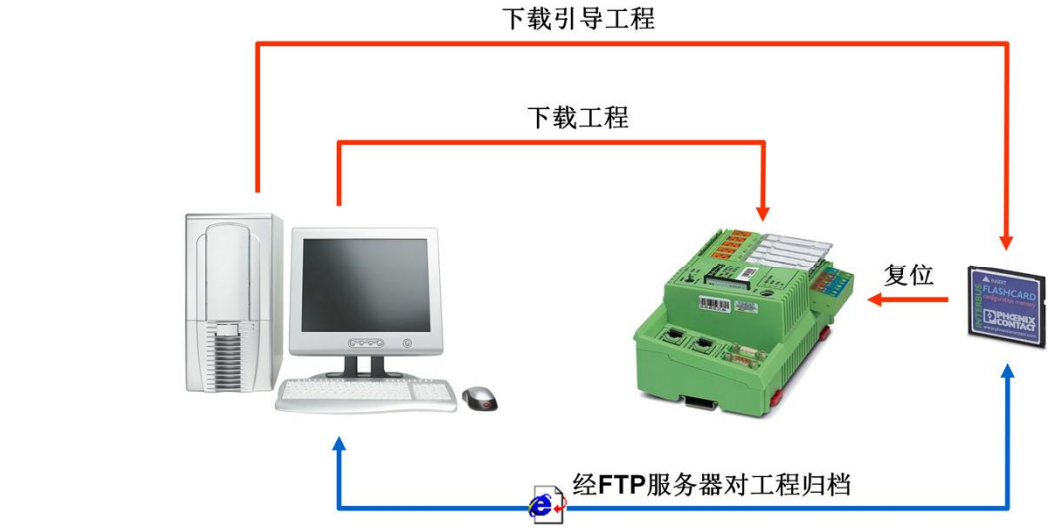
包括工程源代码
在下载到主存储器后， 相应的源代码写入到参数化存储器中。

包括OPC数据
采用合适的控制系统，在下载到主存储器后， OPC配置可被写入到参数化存储器中。

只有在源代码成功下载后，工程才能从参数化存储器中读回，不可能对机器码进行解码读取。

工程代码

存储器概念



在代码处理时，需要对两类代码加以区分：

编译了的机器码

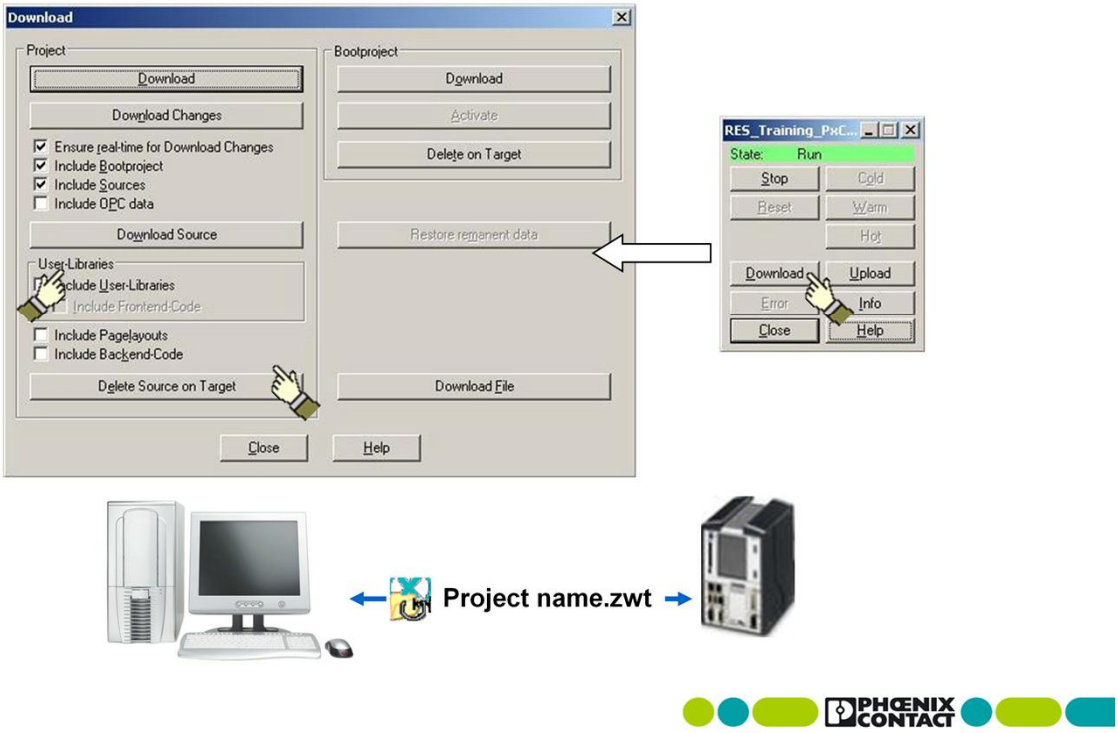
该代码可被控制系统解释或直接执行。采用PC WORX，它不能进行解码。以引导仿真的形式，使用*下载工程*，将其作为可执行代码直接下载到控制系统中。此外，它也可以作为引导工程保存在参数化存储器中。如果控制系统复位或使用PC WORX手动激活机器码，机器码可从参数化存储器中上载，并在控制系统中执行。

可解码的源代码

该代码对应于保存在编程设备硬盘上的归档工程（(<工程名>.zwt）。该代码不能被控制系统解释，仅能作为控制系统上的工程备份。采用最新的控制系统，这些文件的上载和下载都可通过FTP服务器完成。

如果使用旧的控制系统，则必须遵循下一页所描述的使用标准化的归档名保存源　代码

标准归档



标准归档可用于将源代码保存至所有的控制系统中（甚至是一些不含集成FTP服务器的控制系统）。

通过PC WORX中的*Download source（下载源代码）*命令，当前工程将自动归档，并以*ZipFile.zwt*名写入到参数化存储器中。

如果归档文件*ZipFile.zwt*已存在，则激活*Delete source on target（删除目标系统上的源代码）*按钮以删除标准归档，激活*Upload（上载）*按钮以进行上载，并可以在编程设备中打开工程。

对于早期的PC WORX版本/早期的控制器中，归档曾经是作为ZipFile.zwt文件下载。

库

附录 A2

内容

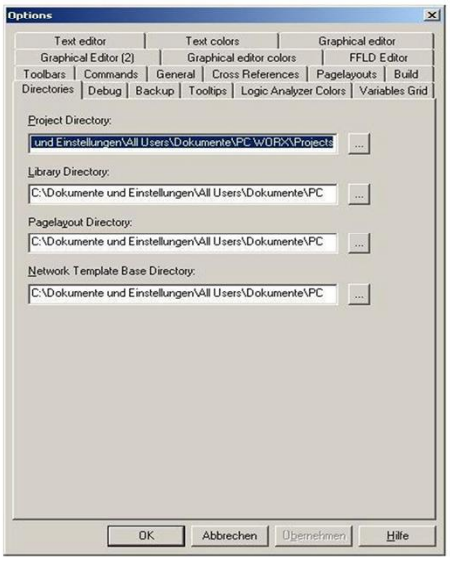
本章节介绍了在PC WORX中库的重要性，如何创建库，库包含了哪些元素，如何在新工程中重新使用库。

注意!

信息

提示

库目录

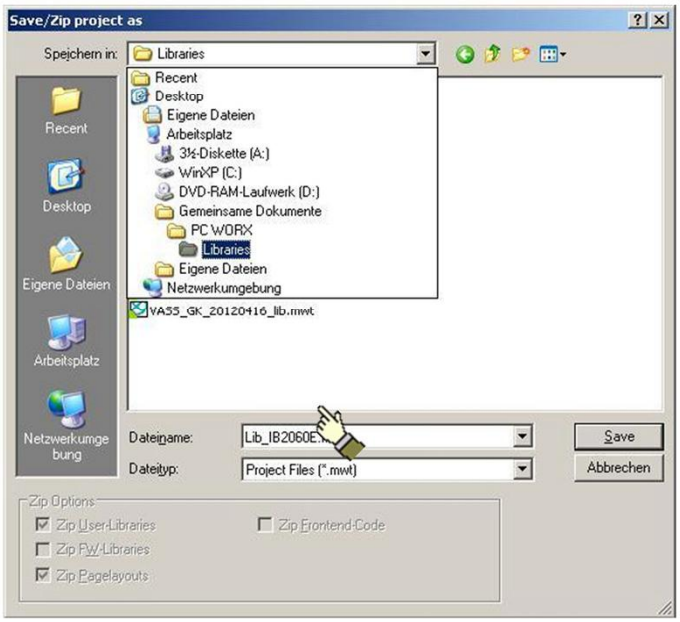


在PC WORX中，库这一术语是指一个工程，它包含了在今后工程中将要使用的程序组件。每个PC WORX工程都可用作一个库，即，集成在一个新工程中。

库被PC WORX 自动管理，并保存在硬盘目录中，通过选项框，修改该目录。

创建库

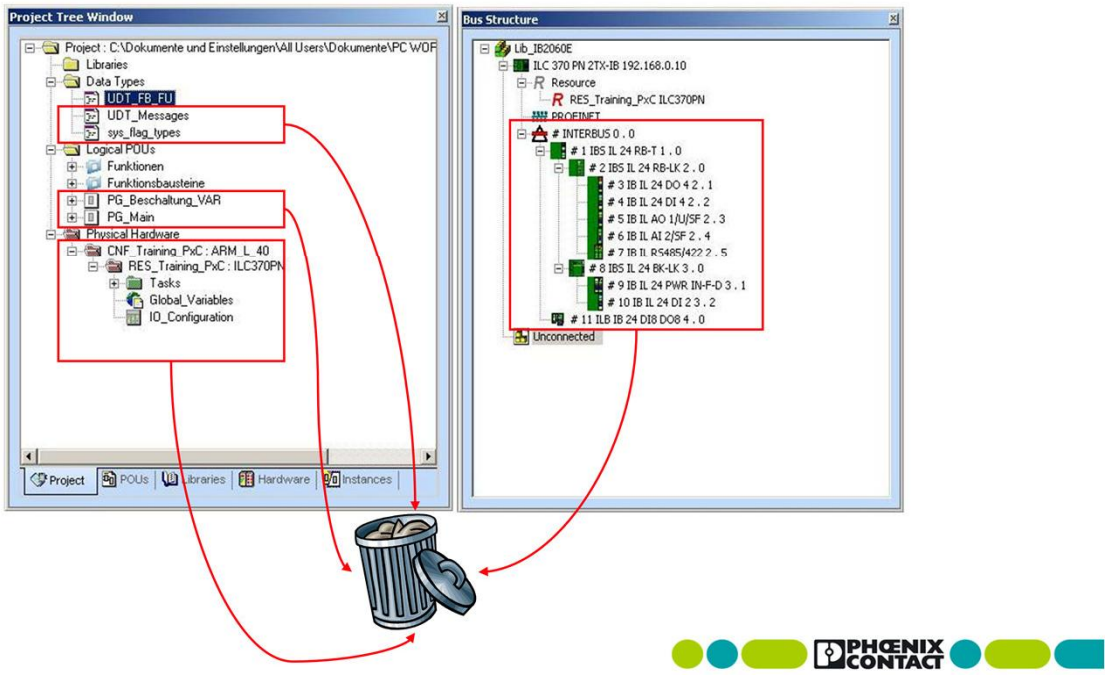
保存



如果当前工程的功能和功能块将通过库为其它工程所用，则必须将该工程以描述性文件名保存在库文件下。由于库是标准工程，因此使用工程保存的标准程序。
为了能在后面通过插入对话框的形式访问库，之前通过选项框选择的目录将作为存储位置。

如果在以新名保存前，对工程进行更改，则删除的内容将永久丢失。

删除和编译



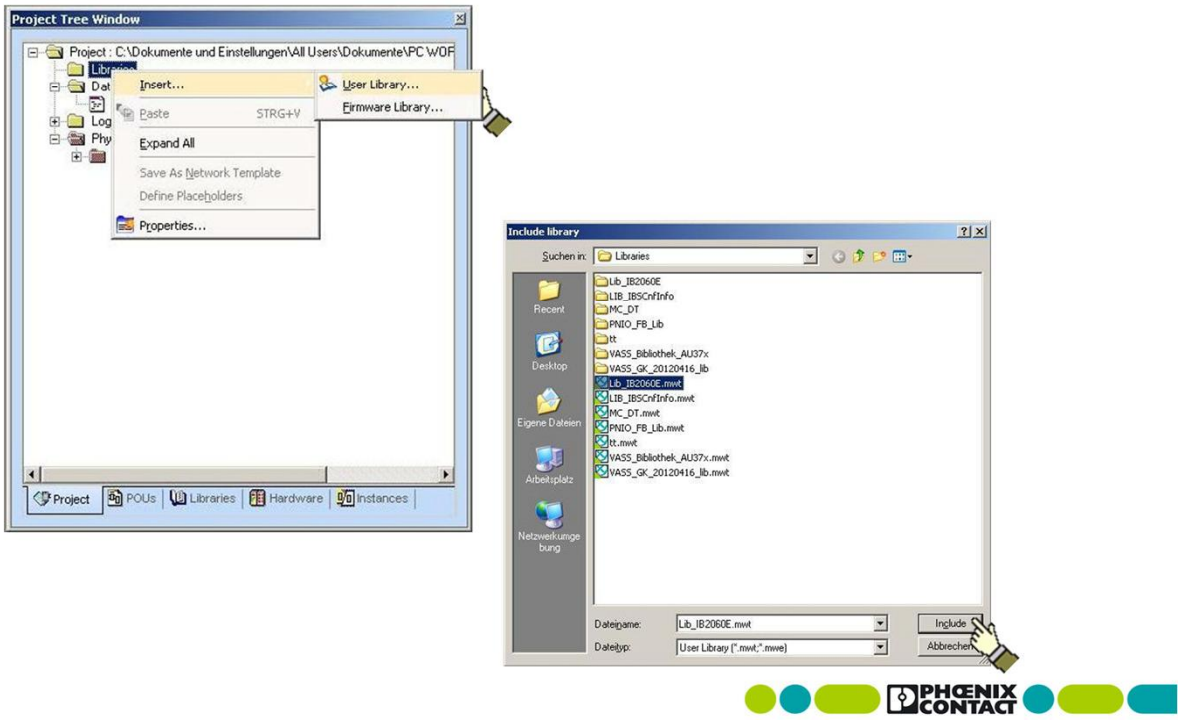
原则上，无需特意创建一个库。每个PC WORX 工程都可用作一个库，但是，在多数情况下，从一个工程中删除非库专用组件是有益的，包括总线组态以及工程树的控制器和硬件结构。

由于库的主要任务是提供功能、功能块和相应的用户自定义数据类型，因此，删除程序POU也是有益的。

删除后，应该能够编译工程而没有错误消息或警告出现，而后，使用库的工程也能进行 编

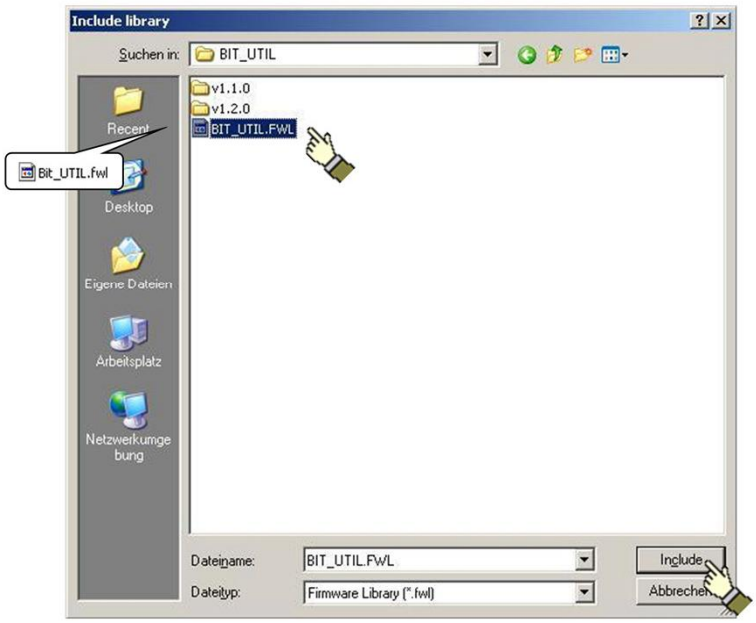
使用库

包含用户库



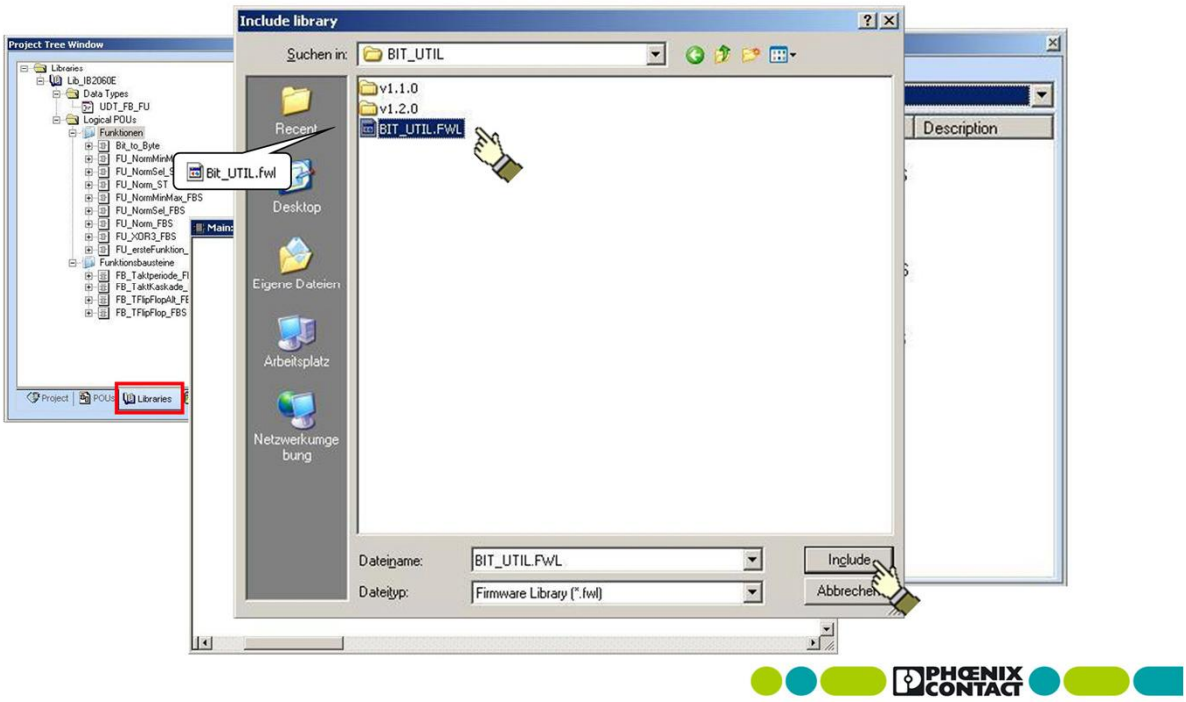
通过工程树的右键快捷菜单，可在当前工程中包含一个库，可以是用户自定义库和固件库。
当插入一个用户库时，系统将访问提取工程(*.mwt)所在的库目录。

包含固件库



当包含一个固件库时， PC WORX 将访问固件库附加目录所在的固定目录。

用户工程中的库



所包含的库可通过工程树中的库标签页中查看到，每个库在编辑向导中以组的形式提供。作为标准，在工程中，来自库的功能将显示为蓝色。

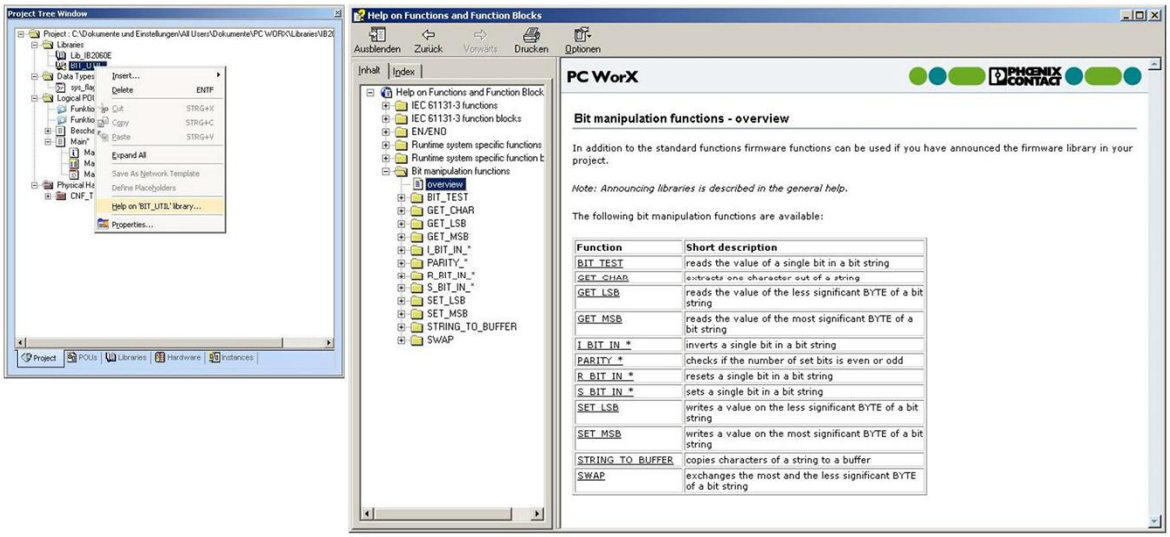
使用带库的工程

库工程成功编译后才能包含于一个工程中，如果必要的话，必须打开并编译库目录中的相应工程。

当打开一个集成了库的工程归档时，也必须选择相同的程序。在工程树中，后面带星号的有关项，通常表示其还没有编译过。这对库也适用。

如果打开一个工程归档时，其中的一个库已经作为一个工程存在于库目录中，则必须通过一个选择对话框对如何处理这两个元素进行决策。
可以使用库的属性对话框检查哪些库包含在工程中。

有关固件库的帮助



与用户自定义库不同，固件库有一个中心帮助功能，可通过右键快捷菜单进行调用。

测试 & 调试

附录 A3

内容

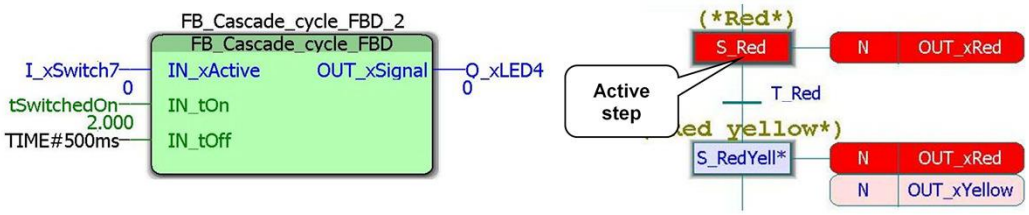
本章节概括介绍了观察和控制一个PC WORX 工程以及写数据的方法。

注意!

信息

提示

工作单中的状态

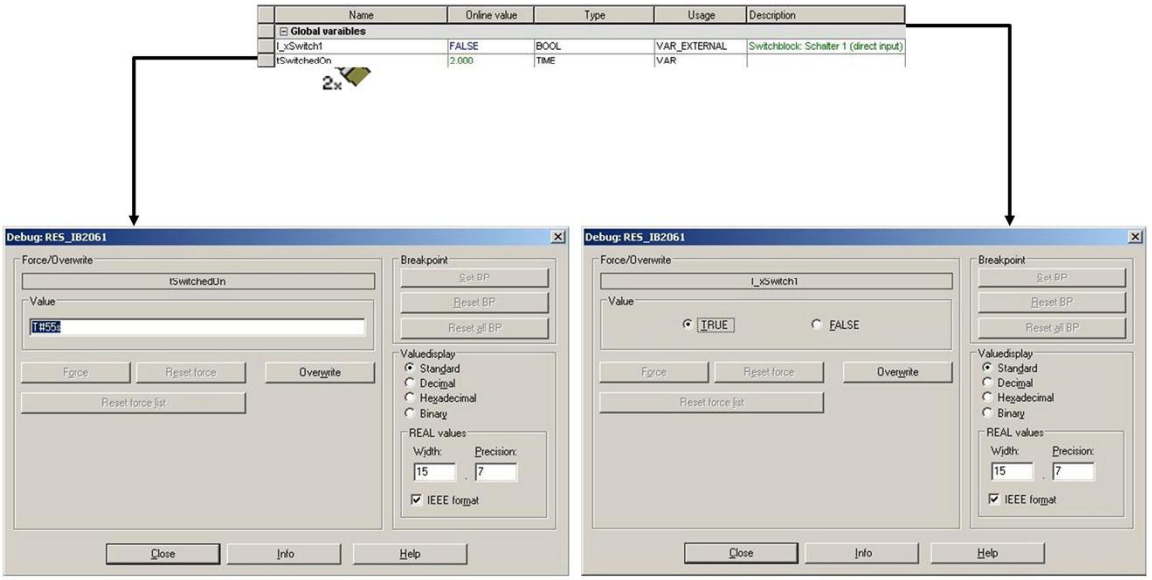



```
30    25.04.2012    strDate :=  Concat(strDay, '.');
31    25.04.2012    strDate :=  Concat(strDate, strMonth);
32    25.04.2012    strDate :=  Concat(strDate, '.');
33    25.04.2012    strDate :=  Concat(strDate, strYear);
34
35      10:45:12    strTime     :=  Concat(strHour, ':');
36      10:45:12    strTime     :=  Concat(strTime, strMinute);
37      10:45:12    strTime     :=  Concat(strTime, ':');
38      10:45:12    strTime     :=  Concat(strTime, strSecond);
```

Name	Online value	Type	Usage	Description	Init
⊞ I/O-variables [digital output signals]					
⊟ User variables [Actual project]					
xGlobalFlag	FALSE	BOOL	VAR_GLOBAL		
stIBS		IBS_AR_1_51...	VAR_GLOBAL	INTERBUS Configuratio...	
arMessageList		AR_1_10_Mes...	VAR_GLOBAL	Message list	
⊞ System variables					
⊟ Inserted from 'BIB_IBS_CFG_INFO_V1_0_0', POE 'PG_InterbusConfiguration'					
iCounfOfDevicesByFW	0	INT	VAR_GLOBAL		
iStartTabulatorForDetail	10	INT	VAR_GLOBAL		10
xCfgReadStart	FALSE	BOOL	VAR_GLOBAL		

在状态视图中，通过调试对话框选择所有工作单的显示格式。数值、基于字符串和位的数据类型之间是存在区别的。

覆盖/强制



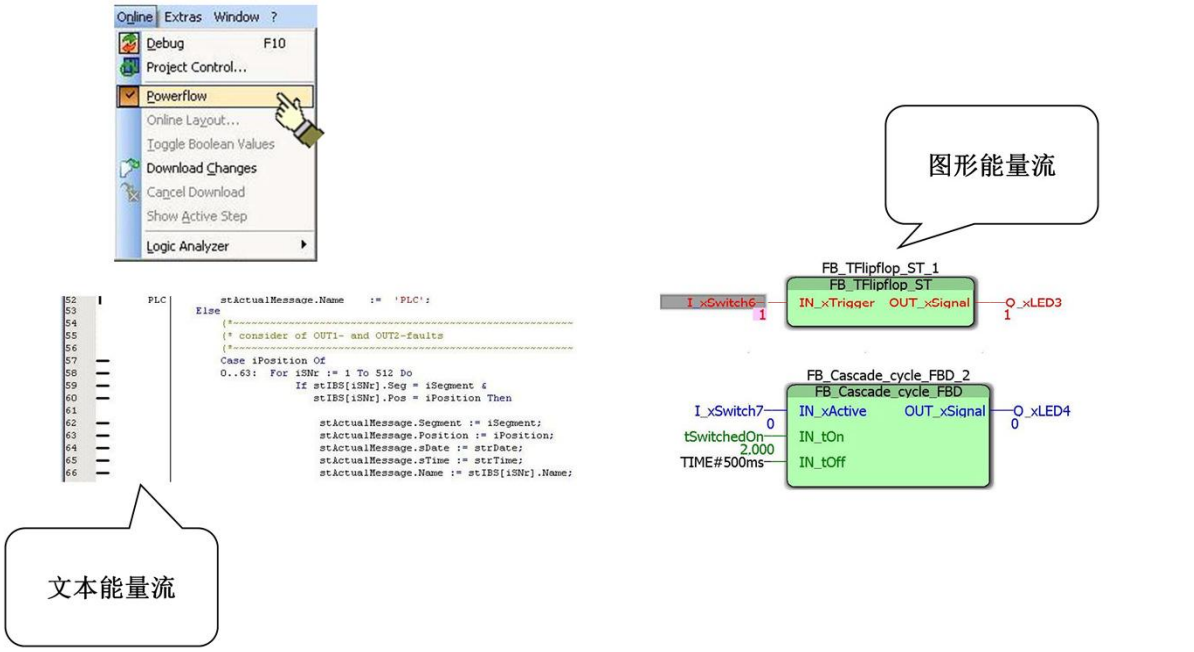
当变量没有预连接逻辑或没有连接到输入过程数据时，变量的重写才有效。重写的结果是一次性将一个值写到变量中。根据格式标准，在此过程中应使用文字的输入格式。

变量的强制只是对与过程数据对象相连的变量支持。

通过传感技术强制一个值在控制程序中被认为是一个严重的干扰性操作。

控制系统中的被强制变量的状态能够通过系统变量中的PLC_DEBUG_FORCE变量 和CPU的控制对话框读出。

能量流 (通过地址状态)

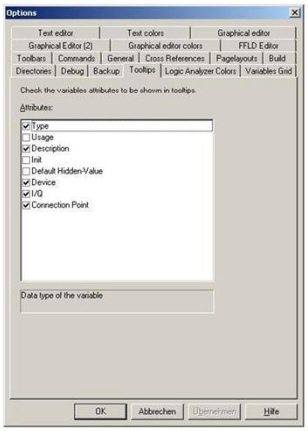


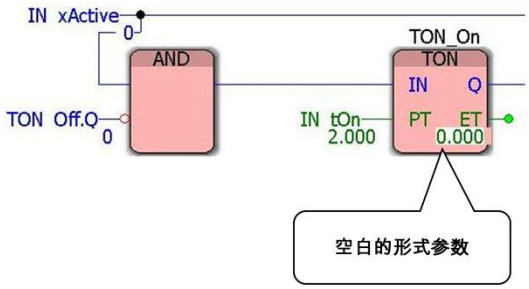
激活地址状态，能量流可在在线的图形和文本代码工作表中执行。如果相应的程序部分正在执行，显示水平或竖直栏。此外，还显示各种数据源写入的变量临时值（“中间结果暂存器”）。

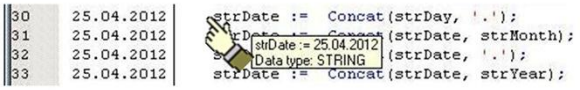
启用地址状态将使循环时间明显增长，并将增加处理器负荷。在操作过程中，由于地址状态的启用会导致PLC停机，因此观察警告信息是绝对必要的。

有关鼠标指针工具提示的信息

工具提示







除了图形和文本编辑器在工作表状态中所提供的显示外，当鼠标掠过相应元素时，会显示进一步的信息。通过选项框，可以选择待显示的变量或FB实例的细节。

为了能显示所有选择的细节，有必要更新交叉参照表（标准：功能按钮F12）。

Tasks for the PC WORX 6 - IEC 61131 Programming Course

Reference: PC WORX 6.00.25 SP3.73

Overview of tasks:

	POU-Type_Function_Language	FBD	LD	IL	SFC	ST	UDT/BIB
1	FU_Xor3_*						
2	FB_Tflipflop_*						
3	FB_Cascade_cycle_*						
4[a]	FU_Norm_*						
4[b]	FU_NormSel_*						
4[c]	FU_NormMinMax_*						
5	FB_Traff_light_*						
6[a]	UDT declaration						
6[b]	PG_Messages						
6[c]	Library integration						

Symbol key:

Function
Function block
Program
Data type worksheet
Library

Tasks regarding communication path setting: TCP/IP

Com 1 *Checking the IP address*

To establish a communication connection via TCP/IP, it is required that both communication partners (i.e., PC WORX and the controller used) are assigned a valid IP address (including the subnet mask). If this is not (yet) the case or not known for the controller used, the IP address can **always** be checked **via the serial interface located on the controller** and be set, if required.

First, set the IP address and the subnet mask on your controller according to the specifications given by your speaker.

Documentation: Page 3-11

After transmitting the desired IP address, ensure that it will be “enabled” on the controller.

Com 2 *Setting up the communication path*

To exchange data between PC WORX and the corresponding controller, it is required to select a communication path for each controller. Even though the serial interface is available for all controllers, it is recommended to set the **TCP/IP** communication path, ensuring in this way a faster data exchange.
Therefore, select **TCP/IP** as the communication path for the controller **within the project**. Enter the IP address manually by selecting “Manual Input” from the “Connection Name” dropdown menu (see *COM 1* task) and test the connection.

Documentation: Pages 3-7, 3-16

Com 3 *Using symbolic connection names*

Once the connection has been established successfully, create a new symbolic connection name in the “ibethdef.dat” assignment file for communication with your controller and select it afterwards. Then test the connection to your controller again.

Documentation: Page 3-17

Tasks regarding INTERBUS bus configuration

IBS1 *Bus configuration (online)*

Use the controller to view the connected bus configuration via the *Connected Bus* dialog box. Copy this configuration (bus devices on basis installation platform) by selecting the corresponding devices from the range of available devices.

Documentation: Pages 4-6ff.

IBS2 *Extending the bus configuration (offline) (optional)*

After connecting the devices that are directly connected to the controller to the remaining remote bus devices, use the device catalog to also extend the bus configuration in the software. The devices to be added are shown in the screenshot below.

Documentation: Page 4-10

Tasks regarding PROFINET bus configuration

PN1 *Bus configuration (online)*

Read in the connected PROFINET devices using the training computer. Copy the devices shown in the dialog box and make the configuration settings as specified by your speaker.

Documentation: Pages 5-6ff.

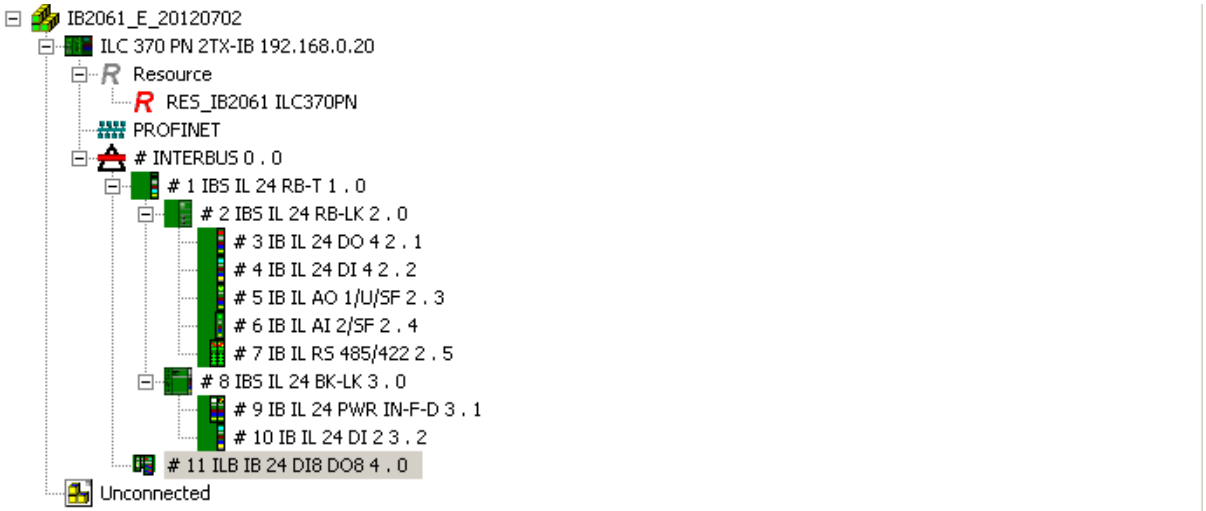
Tasks regarding configuration and variables

PV1 *Creating process data variables*

In the *Process Data Assignment* view, you will create global variables based on the process data objects that are provided by the configured devices. These automatically created variables should be saved to “Resource”.

Documentation: Page 6-6
Context menu for the process data objects of the modules

What is the relation between the automatically generated variable name and the process data object used?



Device	Process Dat...	I/Q	Data ...	Byte.Bit	Port	Symbol/Variable
# 11 ILB IB 24 DI8 DO8 4 . 0	OUT0	Q	BOOL	0.0	O1 1.1	CNF_IB2061 RES_IB2061 \ Q_xLED3
# 11 ILB IB 24 DI8 DO8 4 . 0	OUT1	Q	BOOL	0.1	O1 2.1	CNF_IB2061 RES_IB2061 \ Q_xLED4
# 11 ILB IB 24 DI8 DO8 4 . 0	OUT2	Q	BOOL	0.2	O1 1.4	CNF_IB2061 RES_IB2061 \ Q_4_0_OUT2
# 11 ILB IB 24 DI8 DO8 4 . 0	OUT3	Q	BOOL	0.3	O1 2.4	CNF_IB2061 RES_IB2061 \ Q_4_0_OUT3
# 11 ILB IB 24 DI8 DO8 4 . 0	OUT4	Q	BOOL	0.4	O2 1.1	CNF_IB2061 RES_IB2061 \ Q_4_0_OUT4
# 11 ILB IB 24 DI8 DO8 4 . 0	OUT5	Q	BOOL	0.5	O2 2.1	CNF_IB2061 RES_IB2061 \ Q_4_0_OUT5
# 11 ILB IB 24 DI8 DO8 4 . 0	OUT6	Q	BOOL	0.6	O2 1.4	CNF_IB2061 RES_IB2061 \ Q_4_0_OUT6
# 11 ILB IB 24 DI8 DO8 4 . 0	OUT7	Q	BOOL	0.7	O2 2.4	CNF_IB206
# 11 ILB IB 24 DI8 DO8 4 . 0	IN0	I	BOOL	0.0	I1 1.1	CNF_IB206
# 11 ILB IB 24 DI8 DO8 4 . 0	IN1	I	BOOL	0.1	I1 2.1	CNF_IB206
# 11 ILB IB 24 DI8 DO8 4 . 0	IN2	I	BOOL	0.2	I1 1.4	CNF_IB206
# 11 ILB IB 24 DI8 DO8 4 . 0	IN3	I	BOOL	0.3	I1 2.4	CNF_IB206
# 11 ILB IB 24 DI8 DO8 4 . 0	IN4	I	BOOL	0.4	I2 1.1	CNF_IB206
# 11 ILB IB 24 DI8 DO8 4 . 0	IN5	I	BOOL	0.5	I2 2.1	CNF_IB206
# 11 ILB IB 24 DI8 DO8 4 . 0	IN6	I	BOOL	0.6	I2 1.4	CNF_IB206
# 11 ILB IB 24 DI8 DO8 4 . 0	IN7	I	BOOL	0.7	I2 2.4	CNF_IB206
# 11 ILB IB 24 DI8 DO8 4 . 0	~DI 8	I	BYTE	0.0		
# 11 ILB IB 24 DI8 DO8 4 . 0	~DO 8	Q	BYTE	0.0		

PV2 I/O control for process data variables

Once you have created variables for the process data objects to be used, compile the project:

Click the icon above or the F9 function key.

After successful compilation, transmit the project to the main memory of your controller.

Documentation: Page A1-10

Change to *Programming View* (if not already done) and open the global variables table. The variables that you created yourself have automatically been added to the *Auto* group. Minimize all other variable groups (*Default*, *System Variables*) and activate the debug mode (status indicator) of the controller.

Click the icon above or the F10 function key.

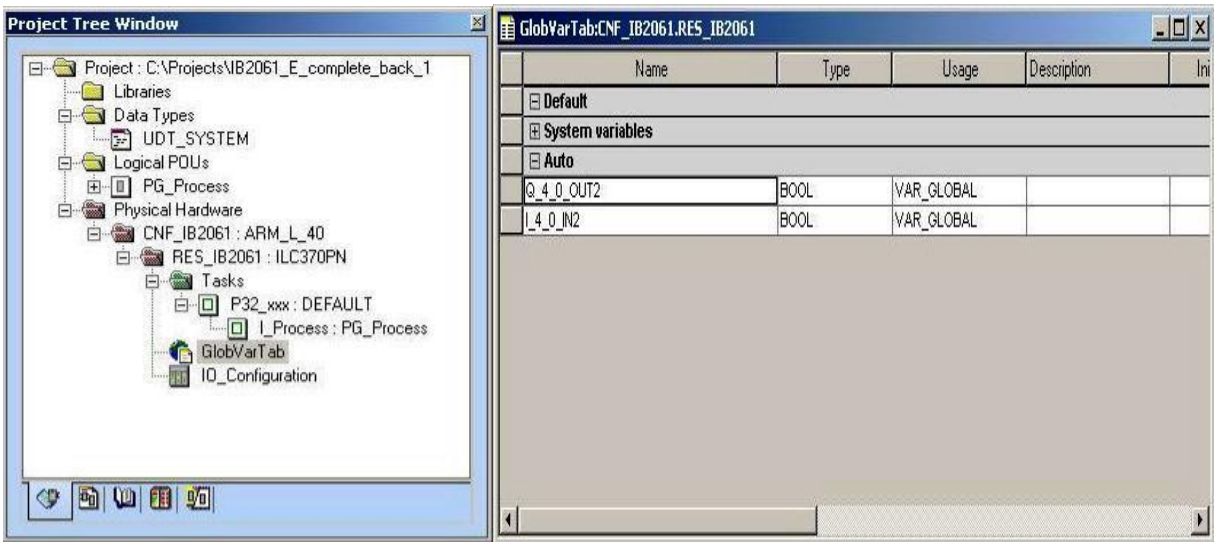
Check the states of your input variables and click the button to the left of the variable names to open the debug dialog box for individual variables. Check the functionality of your outputs by controlling them.

PV3 *Adapting variable names*

In the global variables table, modify the names of the automatically generated variables so that they can easily be related to hardware connection.

Create a new *Process Variables* variables group and shift the variables to this group without losing the connection to process data.

Context menu for a variables group



In variables tables, contents can simply be overwritten after selecting the cell. When a cell is selected, the cursor jumps to position 1 of the text when using the *Pos1* key, and to the end when using the *End* key.

To mark a row, the gray button in variables tables located left to the variable name can be used. If a row is marked, it can be moved exactly using a red target line (without losing process data connection).

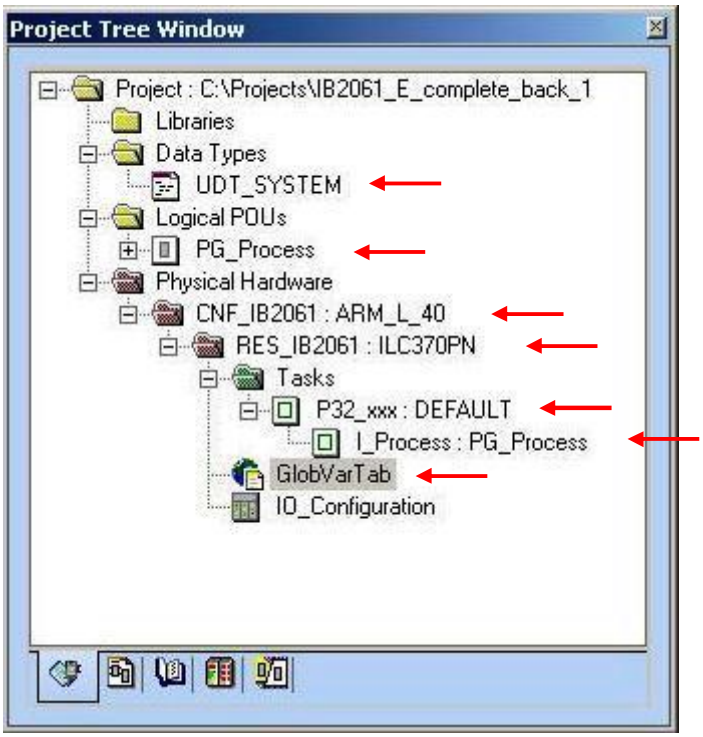
You will find a complete list of all key functions when searching for “Shortcuts” in the help system and selecting the “Shortcuts” hit in the variables worksheet given in tabular form.

PV4 Use of direct inputs (optional)

In the global variables table, direct controller inputs and outputs are available as system variables. For reasons of compatibility, these variables must not be renamed. However, to make them available to the project in a customized manner, it makes sense to copy these signals to individually named variables in the program called first within a PLC cycle.

PV5 Customization of the project

If a project template is used, the elements in the project tree always receive the same standard names. Adapt these to the course standard using the Element Properties dialog box.



The Element Pproperties dialog box of the project tree can be called via the key combination *Alt+Enter* or via the context menu.

Tasks regarding programming in function block diagram

FBDa *First programming*

In the worksheet of the **PG_Course** program, develop programming in such a way that the requirements below will be met.

	Switch1	Switch2	Switch3	Light1
Combination 1	True	True	True	True
Combination 2	False	True	True	True
Combination 3	True	False	True	True
Combination 4	False	False	True	True
Combination 5	True	True	False	False
Combination 6	False	True	False	True
Combination 7	True	False	False	False
Combination 8	False	False	False	False

To parameterize your logic connection, use your automatically created and renamed variables instead of the names predefined in the table.

Documentation: Page 12-6
Documentation: Section 10
Block help from the context menu/edit wizard/appendix

Instead of using a NOT block, some blocks (e.g., those the Boolean logic can be used for) allow input and output parameters to be inverted.

FBDb *First function*

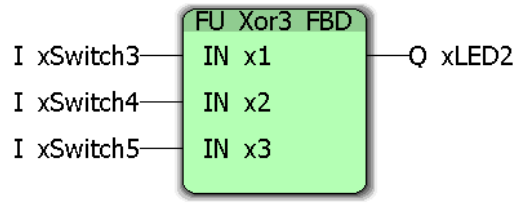
Add a **Functions** POU group to your project tree. To this group, add a new function with the name **FU_Start_FBD** using function block diagram as the programming language.

This function should provide the same logic as the previous programming and should be called in the program (instead of the previous programming).
It is now required to internally use variables of the “VAR_INPUT usage” and the output parameters of the function instead of global variables.

When assigning names to input parameters of function and function blocks, use more general names such as IN1 and IN2 instead of Switch1 and Switch2. The selection of a name should be based on the parameter function and not on the process variable that is accidentally connected to the parameterizable block within a project.

Documentation: Section 10 and pages 13-5ff.

FBD1 *Function FU_Xor3_FBD*



Add a function with the name **FU_Xor3_FBD** to your project tree. This function should meet the following requirements:

	IN1	IN2	IN3	FU_Xor3_FBD
Combination 1	True	True	True	False
Combination 2	False	True	True	False
Combination 3	True	False	True	False
Combination 4	False	False	True	True
Combination 5	True	True	False	False
Combination 6	False	True	False	True
Combination 7	True	False	False	True
Combination 8	False	False	False	False

Block help from the context menu/edit wizard/appendix

Instead of using a NOT block, some blocks (e.g., those the Boolean logic can be used for) allow input and output parameters to be inverted.

For a systematic approach, check the cases where the output parameter should have the value `True`. Represent these cases separately.

FBD2 *Function block FB_TFlipflop_FBD*

Add a **Function blocks** POU group to your project tree. To this goup, add a new function block with the name **FB_TFlipflop_FBD**
using function block diagram (FBD) as the programming language.

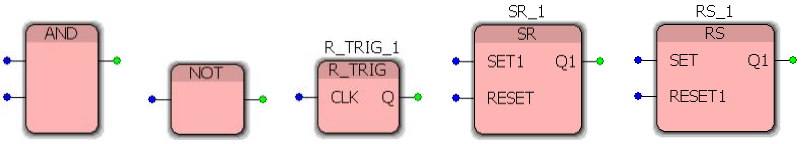
The function block should behave as follows:

If *a rising edge is detected at the IN input parameter,* **then** *the OUT output parameter is to be inverted.*

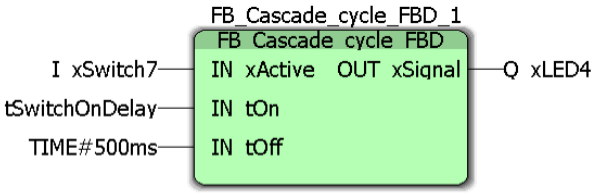
Block help from the context menu/edit wizard/appendix

Divide the task in two subtasks and combine the solutions afterwards. Formulate the task for yourself.

Blocks that can be helpful for creation (not all of them must be used):



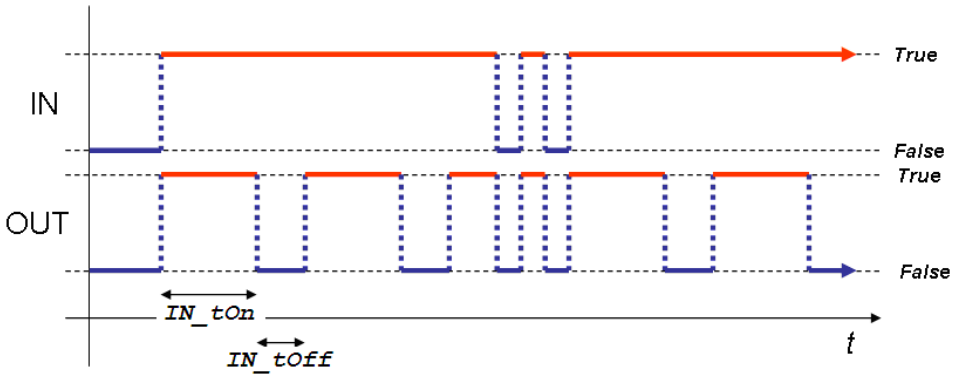
FBD3 *Function block FB_Cascade_cycle_FBD*



Add a function block with the name **FB_Cascade_cycle_FBD** to your project tree using function block diagram as the programming language.

The function block should behave as follows:
If the IN_xActive input parameter is set to True, the OUT_xSignal output parameter should immediately follow and be set to True. As long as IN_xActive is set to True, the OUT_xSignal should be set to True for the time specified for IN_tOn and then switch to False for the time specified for IN_tOff. This switching shall continue until IN_xActive is set to False. In this case, the OUT_xSignal should also change to False immediately.

This behavior is shown in the following diagram:

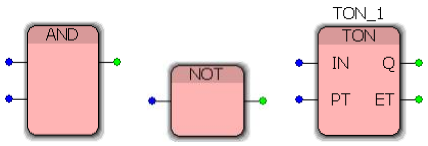


Block help from the context menu/edit wizard/appendix

Use the time diagram to develop programming step by step, i.e. section by section.

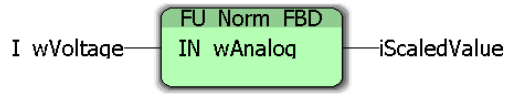
An alternative option is based on the idea of two overlapping signals: the continuous activity signal and an alternating interference signal.

Blocks that can be helpful for creation:



FBD4 Functions for analog value processing

[a] Basic functions: FU_Norm_FBD



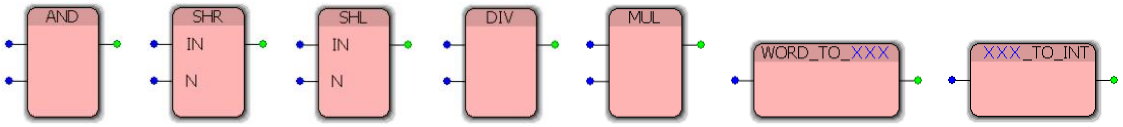
Add a function with the name **FU_Norm_FBD** to your project tree. This function is to scale an analog value provided by an analog input module according to the following scheme:

Analog input value	Representation	Scaled value
0-10V	x 12Bit-Analog-Value x x x	$0..100_{decimal}$

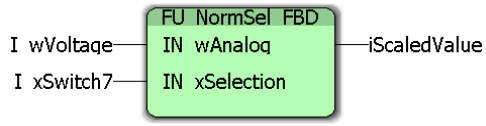
Block help from the context menu/edit wizard/appendix
Device data sheet

Please note that a conversion in data types is required for scaling. These data types are intended for parameterizing arithmetic basic functions and provide a value range sufficient for calculations.

Blocks that can be helpful for creation (not all of them must be used):



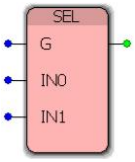
[b] Extension by binary range selection: FU_NormSel_FBD



In the project tree, create a copy of the function programmed for task [a]. Change the POU name to **FU_NormSel_FBD** and add the `IN_xSelection` parameter to the programming. This parameter causes the scaled value to be indicated in percent (with `IN_xSelection` = `False`) or in per mil (with `IN_xSelection` = `True`).

Analog input value	Scale	Representation	Scaled value
0-10V	`False`	x 12Bit Analog Value x x x	$0..100_{decimal}$
0-10V	`True`	x 12Bit Analog Value x x x	$0..1000_{decimal}$

Additional blocks that can be helpful for creation:

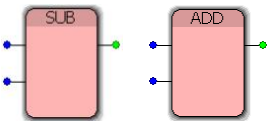


[c] Extension for user-defined scaling: FU_NormMinMax_FBD

In the project tree, create a copy of the function programmed for task [a]. Change the POU name to **FU_NormMinMax_FBD** and add two INT input parameters - `IN_iMax` and `IN_iMin` - to the programming. These allow the user to dynamically adjust the upper and lower value of the scaled value via the defined values. Scaling should be linear.

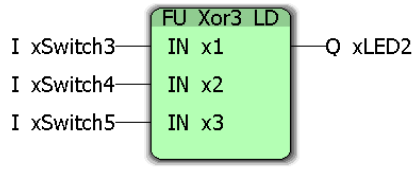
Analog input value	Min..Max	Representation	Scaled value
0-10V	`-20..80`	x 12Bit Analog Value x x x	$-20..80_{decimal}$

Additional blocks that can be helpful for creation:



Tasks regarding programming in ladder diagram

LD1 *Function FU_Xor3_LD*



Add a function with the name **FU_Xor3_LD** to your project tree. This function shall perform the same task as the function of task FBD1.

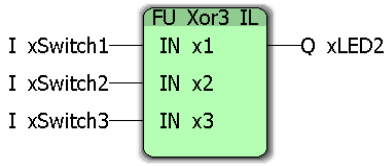
LD2 *Function block FB_TFlipflop_LD*

Add a function block with the name **FB_TFlipflop_LD** to your project tree. This block shall perform the same task as the function block of task FBD2.

Consider how you can realize and program an edge evaluation using the elements of ladder diagram and, in addition to this, implement the inversion via contacts and coils, in order to not require the use of function blocks.

Tasks regarding programming in instruction list

IL1 *Function FU_Xor3_IL*

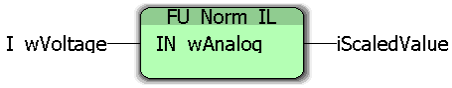


Add a function with the name **FU_Xor3_IL** to your project tree. This function shall perform the same task as the function of task FBD1.

Documentation: Page 14-6

IL4 Functions for analog value processing

[a] Basic functions: FU_Norm_IL

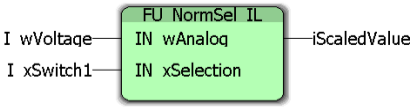


Add a function with the name **FU_Norm_IL** to your project tree. This function shall perform the same task as the function of task FBD4[a].

Documentation: Page 14-10

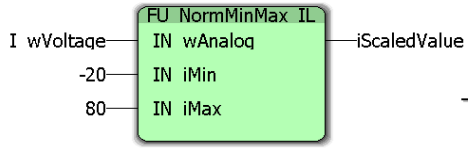
For branches to be executed in parallel, use brackets or local variables (intermediate markers) in instruction list.

[b] Extension by binary range selection: FU_NormSel_IL



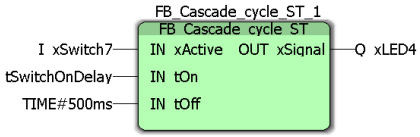
Add a function with the name **FU_NormSel_IL** to your project tree. This function shall perform the same task as the function of task FBD4[b].

[c] Extension for user-defined scaling: FU_NormMinMax_IL



Add a function with the name **FU_NormMinMax_IL** to your project tree. This function shall perform the same task as the function of task FBD4[c].

IL3 *Function block FB_Cascade_cycle_IL*



Add a function block with the name **FB_Cascade_cycle_IL** to your project tree. This block shall perform the same task as the function block of task FBD3.

Documentation: Page 14-12

IL2 *Function block FB_TFlipflop_IL*

Add a function block with the name **FB_TFlipflop_IL** to your project tree. This block shall perform the same task as the function block of task FBD2.

Consider how you can realize and program an edge evaluation using the elements of instruction list and, in addition to this, implement the inversion via conditional execution, in order to not require the use of function blocks.

Documentation: Page 14-15

Tasks regarding programming in sequential function chart

SFC5 Function block FB_Traff_light_SFC

[a] Basic functions

Add a function block with the name **FB_Traff_light_SFC** to your project tree. This function block is to be programmed in sequential function chart (SFC).

In the first step, the basic functions of a single signal device are implemented, i.e. the sequence *Red* | *Red-Yellow* | *Green* | *Yellow*.

The input parameters (*Time* data type) are used to provide the block with the phase lengths. The individual light controls are called via the output parameters.

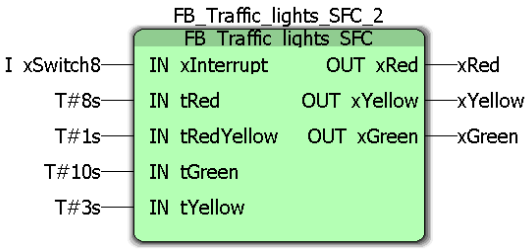
The output parameters should be created as action variables (without using action zooms). The transitions, however, should be programmed as transition zooms (any IEC 61131 language).

The sequence of the corresponding light phase is used as a transition condition.

Documentation: Section 16

[b] Extended functions with error flashing

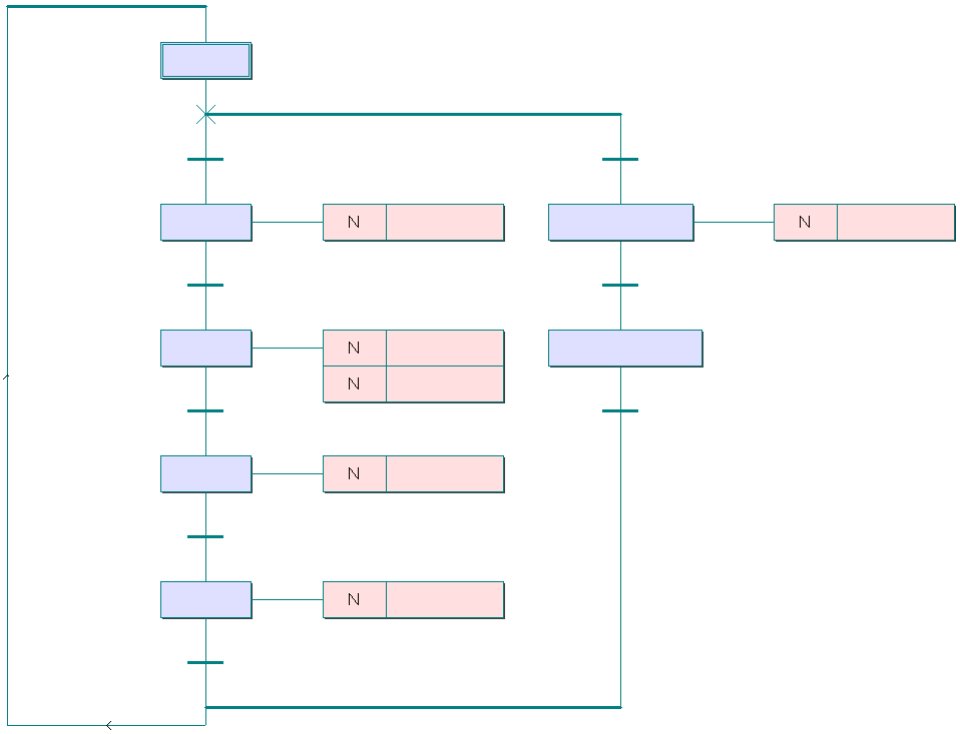
Add the IN_xInterrupt input parameter to the function block created for task [a]. Controlling this input should enable **direct** jumping to an alternative branch which has been inserted in parallel to the four previous action steps. Its sequence triggers a flashing yellow light (f = 1 Hz). This error mode is to be replaced by normal operation after resetting the IN_xInterrupt input parameter.



The deactivation of the currently active step (all steps except for the error mode steps) and the activation of the first error step is to be implemented via the **StepName.x** step flag (e.g. **S_RedYellow.x**) of the steps.

As each transition zoom is executed in each cycle, the required error logging can be implemented in each zoom. It would make sense to implement this programming in the zoom of the initial step.

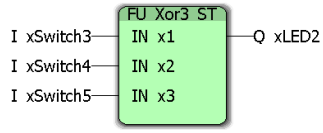
Use the following step scheme as an orientation help for programming:



The step chain shown above uses four steps for traffic light control in normal operation, two steps for error flashing and an additional step as the initial step.

Tasks regarding programming in structured text

ST1 *Function FU_Xor3_ST*

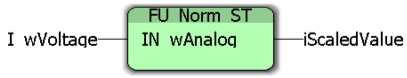


Add a function with the name **FU_Xor3_ST** to your project tree. This function shall perform the same task as the function of task FBD1.

Documentation: Page 18-6

ST4 Functions for analog value processing

[a] Basic functions: FU_Norm_ST

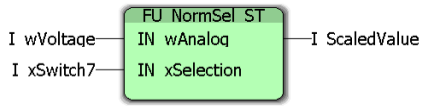


Add a function with the name **FU_Norm_ST** to your project tree. This function shall perform the same task as the function of task FBD4[a].

Documentation: Page 18-8

For functions to be executed in parallel and serially, you have to use brackets or local variables (intemediate markers) in structured text.

[b] Extension by binary range selection: FU_NormSel_ST

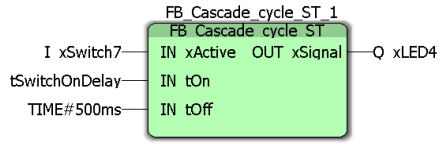


Add a function with the name **FU_NormSel_ST** to your project tree. This function shall perform the same task as the function of task FBD4[b].

[c] Extension for user-defined scaling: FU_NormMinMax_ST

Add a function with the name **FU_NormMinMax_ST** to your project tree. This function shall perform the same task as the function of task FBD4[c].

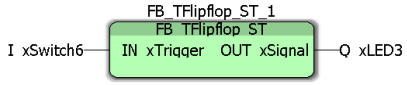
ST3 Function block FB_Cascade_cycle_ST



Add a function block with the name **FB_Cascade_cycle_ST** to your project tree. This block shall perform the same task as the function block of task FBD3.

Documentation: Page 18-10

ST2 Function block FB_TFlipflop_ST (optional)



Add a function block with the name **FB_TFlipflop_ST** to your project tree. This function shall perform the same task as the function block of task FBD2.

Consider how you can do without function blocks when using high-level language structures.

Documentation: Page 18-17

Re-read the formulation of functionality in task FBD2.

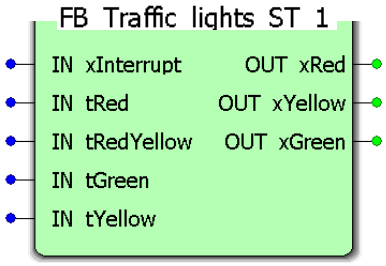
ST5 *Function block FB_Traffic_lights_ST (optional)*

[a] Add a function block with the name **FB_Traffic_lights_ST** to your project tree. This block shall perform the same task as the function block of task SFC5.

Consider how you can implement the step chain following the example of the alternative solution for FBD3.

Documentation: Page 18-23

[b] Extend programming by an IN_xInterrupt input parameter, in the same way as with sequential function chart.



ST6 Logging of INTERBUS errors with FIFO memory

The target of the complex task ST6 is to log bus errors reported by the INTERBUS controller (including date, time, segment, position and plain text information on the respective device). This error message should be recorded in a FIFO (First In First Out) memory using ten elements. Element 1 should always be used to output the current error message.

[a] Data type definition

1.	Definition of a structure data type according to the message requirements

ST_Message
- Date — Type: `String`
- Time — Type: `String`
- Segment — Type: `Int`
- Position — Type: `Int`

2.	Definition of an array data type according to the message list requirements

AR_1_10_Message

Type: `Array [1..10] Of ST_Message`

[b] Program for error logging:

Add a program with the name **PG_Messages** in structured text to your project tree. Once completed, instantiate the program by calling the cyclic task available in your project under the name **I_Messages**.

For information on functions required for the following tasks, please refer to the last pages of the task in hand.

The program should perform the following tasks in five steps:

1. Processing values provided by the system variables and saving them to local variables.

Use two-digit values for day, month, hour, minute, and second. Use a four-digit value for the year.

Variable	Usage	Data type
strDay	VAR	String
strMonth	VAR	String
strYear	VAR	String
strHour	VAR	String
strMinute	VAR	String
strSecond	VAR	String
iSegment	VAR	String
iPosition	VAR	String

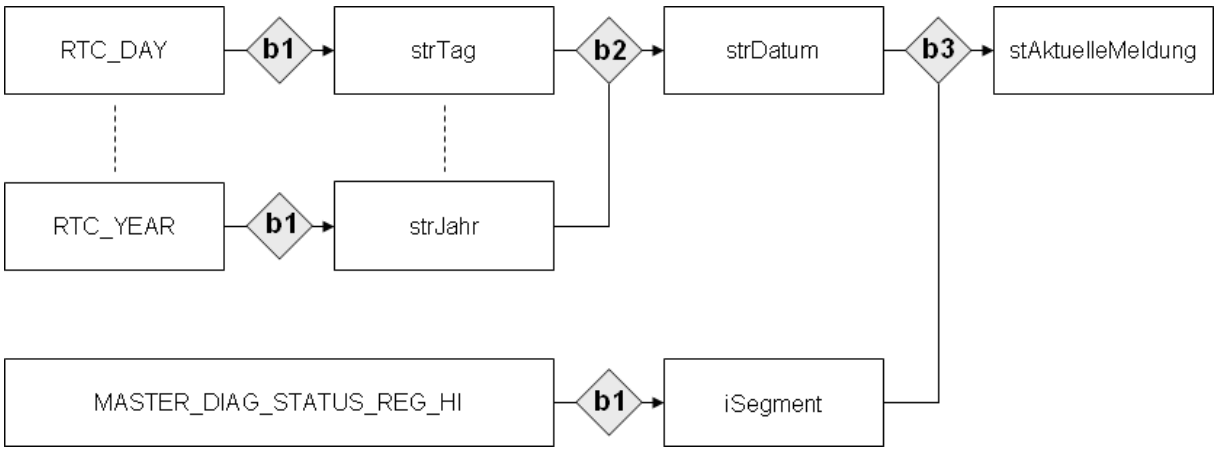
2. Combining time and date values to string variables of the following format:

Use the standard formats for date and time:
Example for date: '26.01.2007'
Example for time: '14:17:52'

Variable	Usage	Data type
strDate	VAR	String
strTime	VAR	String

3. Entering the compiled information of the previous steps in a local variable.

Variable	Usage	Data type
stCurrentMessage	VAR	ST_Message

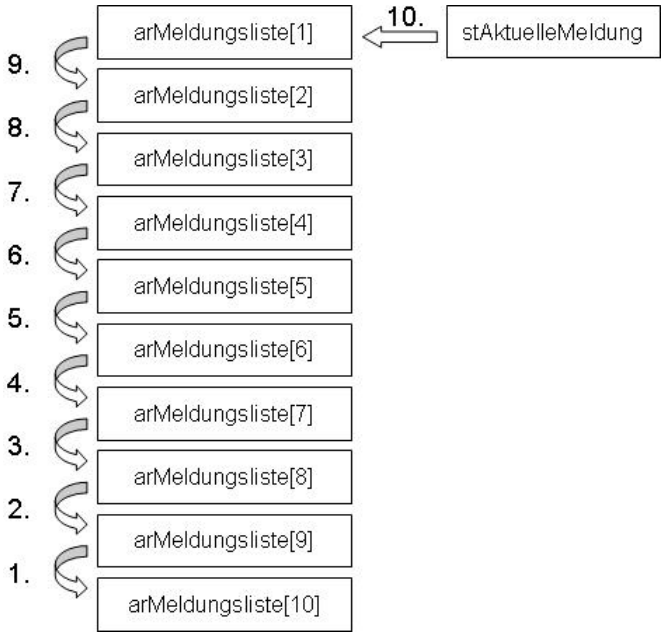


Graphical overview of conversions and assignments (selection)

4.	Creating a global variable for entering the current message. This variable represents the message list.

Variable	Usage	Data type
arMessageList	VAR_GLOBAL	AR_1_10_Message

To update the message list, first copy the content of element 9 to element 10, then from element 8 to element 9, etc. Finally, the current message is entered in element 1.



This update can be done manually by means of ten separate statements or implemented via a For loop and an additional assignment. The For loop is not able to count down the control variable. This represents a special challenge when programming the For loop.

5.	Conditional execution of the entry routine

The entry routine should only be executed, if the falling edge of the error-detecting bit and the pending bus error bit of the INTERBUS master are detected by the program. With regard to detection, use the system variables provided by the controller (additional information can be found in the appendix).

[c] Device name in plain text

The following tasks enable the respective device name to be shown in plain text.

1.	Extending the ST_Message data type by one parameter.

ST_Message
- Date — Type: `String`
- Time — Type: `String`
- Segment — Type: `Int`
- Position — Type: `Int`
- Name — Type: `String`

2.	Integrating a user library.

Integrate the *BIB_IBS_CFG_INFO_V1_0_D.zwt* library in the current project.

The library provides the following elements:

Data types:

IBS_ST_DevInfo
- Name — Type: `IBS_String20`
- Segment — Type: `Int`
- Position — Type: `Int`
- Active — Type: `Bool`
- Jumpered — Type: `Bool`
- PF — Type: `Bool`

IBS_AR_1_512_DevInfo
- `ARRAY [1..512] OF IBS_ST_DevInfo`

IBS_ST_CfgInfo
- DevCnt — Type: `Int`
- DevInfoList — Type: `IBS_AR_1_512_DevInfo`

Programs:

PG_InterbusConfiguration

The program reads the active INTERBUS configuration from the INTERBUS master using the *Read_Complete_Configuration* firmware service and determines the number of devices within the configuration. The data acquired is entered in the **stIBS** structure variables (module segment and position, active status and jumpered status, peripheral fault occurred). The structure variables are sorted according to the consecutive number of the modules. In addition, the program reads the Diag32 file saved on the memory card of the controller by sending the boot project. In the **stIBS** variable, the station names for the IBS modules which are saved in this file (with the initial value for the variable `iStartTabulatorForDetail` = 10) are added to the already acquired module information.

3. Program call and parameterization

Instantiate the program provided by the library and call the program via the cyclic task available in your program.

In the context menu of the project tree, select the function for transforming VAR_EXTERNAL into VAR_GLOBAL.

For the `iStartTabulatorForDetail` variable, assign the value specified in the description.

4. Using the device name from the configuration structure variable.

Check the stIBS variable for entries with regard to segment and position and use the entered name from the variable, on condition that both parameters match the values currently provided by the IBS master.

Consider the following situations: The IBS master is reported as a faulty segment (segment: 0) or an OUT1 or OUT2 device interface error has occurred. In the event of interface errors, the segment variable indicates the segment of the reporting remote bus device, the position variable has the value 80_{hex} for OUT1 or 81_{hex} for OUT2.

Support for task ST6

For [b] 1. System variables, realtime clock, INTERBUS diagnostics

The following system variables provide information on the system time of the ILC:

RTC_Year	INT (four-digit)
RTC_Month	INT
RTC_Day	INT
RTC_Hours	INT (24h-indication)
RTC_Minutes	INT
RTC_Seconds	INT

The following variables provide information on the bus error, active bus error diagnostics and the respective device:

MASTER_DIAG_PARAM_REG_HI	BYTE (Segment)
MASTER_DIAG_PARAM_REG_LOW	BYTE (Position)
MASTER_DIAG_STATUS_REG_BUS	BOOL (IBS bus error)
MASTER_DIAG_STATUS_REG_DTC	BOOL (IBS diagnostics active)

For [b] 1. Data type conversion Word_To_Int

When converting a byte variable into an integer value, please note that the most significant bit is the sign bit. If a byte without sign bit is to be converted, it is recommended to use the word data type in a roundabout way. The call structure is then as follows:

```
IntVar := Word_to_Int(Byte_To_Word(ByteVar));
```

This is the nested call of functions, as it is typical for structured text. Example for conversion with sign bit:

```
Byte#2#1001_0101 → Byte_To_Int →                  -107
Byte#2#1001_0101 → Byte_To_Word → Word_To_Int →    149
```

For [b] 1. Data type conversion Int_To_String

INT_TO_STRING is the function for converting an integer variable in a string variable. This function requires two input parameters, i.e. the integer to be converted (of integer type) and the format (of string type) into which the string is to be converted. The call in structured text is as follows:

```
String := Int_To_String(INT_Variable, Format);
```

Example:

INT_Value : 4
Format: '%02d'
String: '04'
(The number is indicated with two digits and leading zeros.)

INT_Value: 2007
Format: '%04d'
String: '2007'
(The number is indicated with four digits and leading zeros.)

For [b] 2. Concat string concatenation

CONCAT is the function for combining two strings to a new string. It is used as follows:

```
New string := Concat(string1, string2);
```

Example:

String 1 : 'PC WORX'
String 2: ' is awesome!'
New string: 'PC WORX is awesome!'

Spaces in the string (as in front of 'is') are also used. Please consider that only two values can be combined using CONCAT. To set up date and time, CONCAT has to be called four times in each case.

Solutions for the PC WORX IEC 61131 Programming Course

Latest version: 5. Februar 2014
Reference: PC WORX 6.00.25 SP3.73

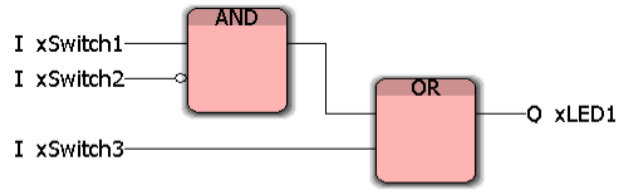
Solutions regarding programming in function block diagram

FBDa *First programming*

Variables table

	Name	Type	Usage	Description
	Global varaibles			
	I_xSwitch1	BOOL	VAR_EXTERNAL	Switchblock: Schalter 1 (direct input)
	I_xSwitch2	BOOL	VAR_EXTERNAL	Switchblock: Schalter 2 (direct input)
	I_xSwitch3	BOOL	VAR_EXTERNAL	Switchblock: Switch 3
	Q_xLED1	BOOL	VAR_EXTERNAL	LED 1

Programming

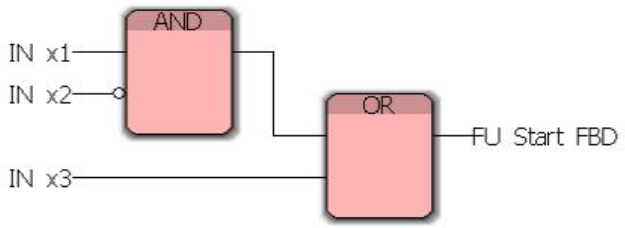


FBDb *First function*

Variables table

	Name	Type	Usage	Description
	IN			
	IN_x1	BOOL	VAR_INPUT	
	IN_x2	BOOL	VAR_INPUT	
	IN_x3	BOOL	VAR_INPUT	

Programming

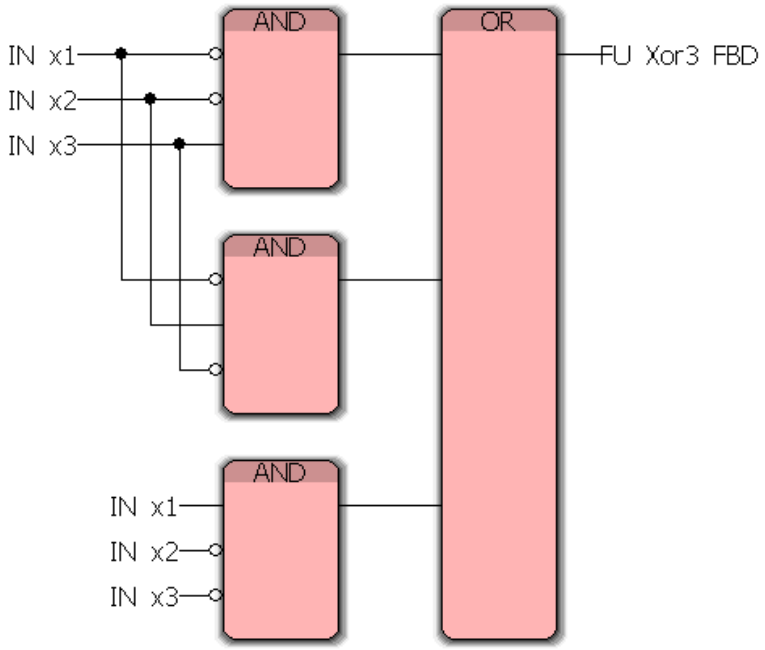


FBD1 *FU_Xor3_FBD*

Variables table

	Name	Type	Usage	Description
	⊟ IN			
	IN_x1	BOOL	VAR_INPUT	
	IN_x2	BOOL	VAR_INPUT	
	IN_x3	BOOL	VAR_INPUT	

Programming

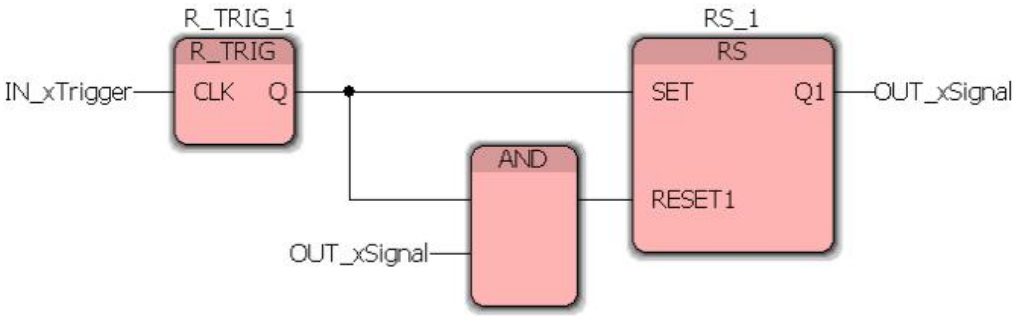


FBD2 *FB_TFlipflop_FBD*

Variables table

	Name	Type	Usage	Description
	IN			
	IN_xTrigger	BOOL	VAR_INPUT	
	OUT			
	OUT_xSignal	BOOL	VAR_OUTPUT	
	FB-Instnces			
	R_TRIG_1	R_TRIG	VAR	
	RS_1	RS	VAR	

Programming



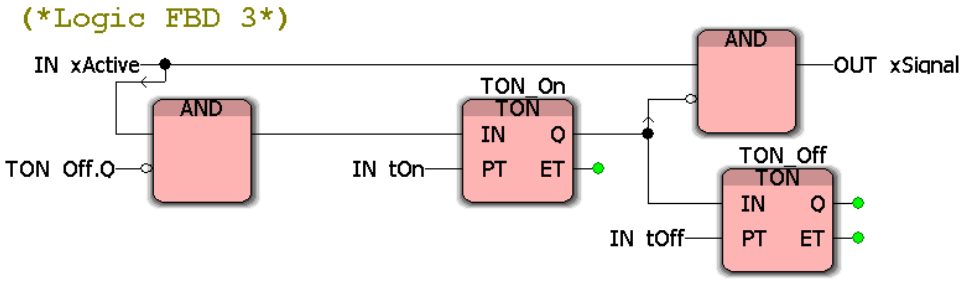
Alternative programming

FBD3 FB_Cascade_cycle_FBD

Variables table

	Name	Type	Usage	Description
	IN			
	IN_xActive	BOOL	VAR_INPUT	
	IN_tOn	TIME	VAR_INPUT	
	IN_tOff	TIME	VAR_INPUT	
	OUT			
	OUT_xSignal	BOOL	VAR_OUTPUT	
	TON_Off	TON	VAR	
	FB-Instances			
	TON_On	TON	VAR	

Programming

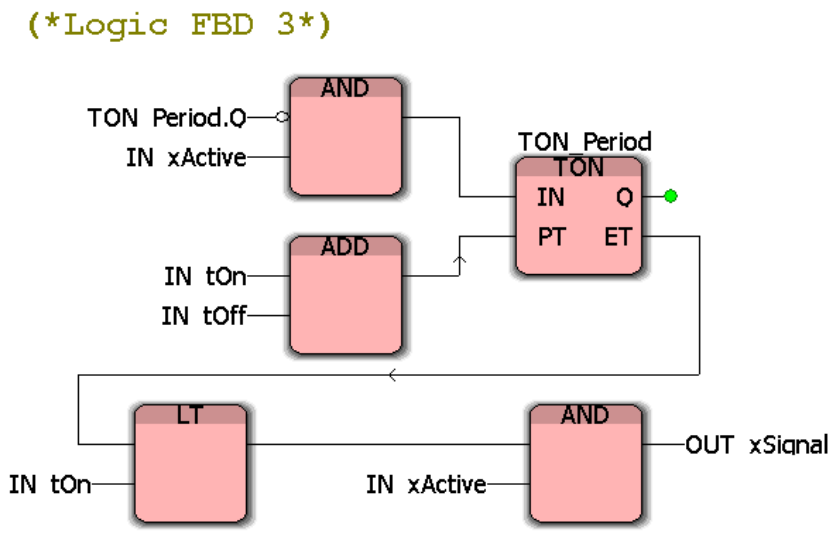


FBD3 FB_Cascade_cycle_FBD (alternative solution)

Variables table

	Name	Type	Usage	Description
	IN			
	IN_xActive	BOOL	VAR_INPUT	
	IN_tOn	TIME	VAR_INPUT	
	IN_tOff	TIME	VAR_INPUT	
	OUT			
	OUT_xSignal	BOOL	VAR_OUTPUT	
	FB-Instances			
	TON_Period	TON	VAR	

Programming

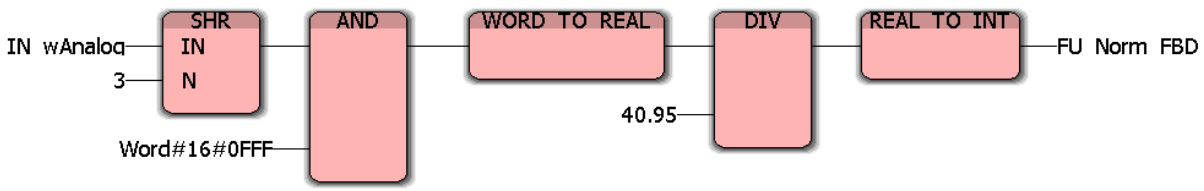


FBD4[a] FU_Norm_FBD

Variables table

	Name	Type	Usage	Description
	IN			
	IN_wAnalog	WORD	VAR_INPUT	12Bit-Analog value

Programming

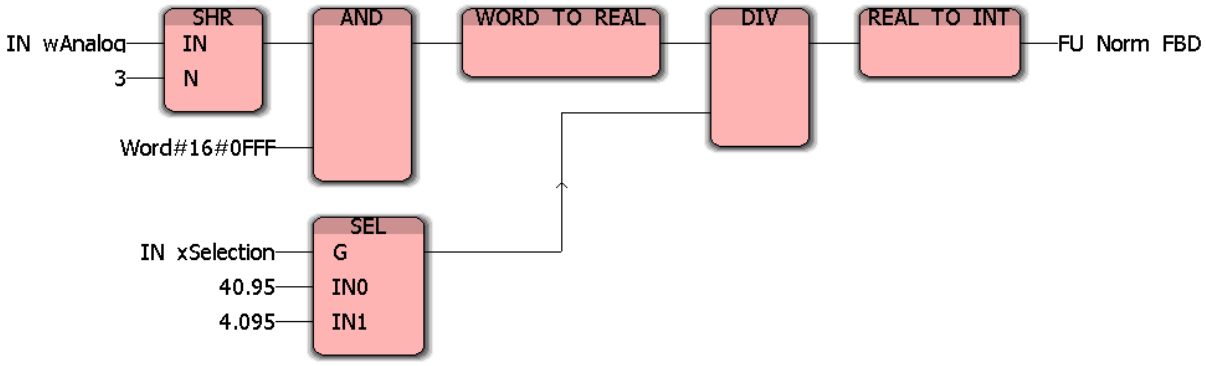


FBD4[b] FU_NormSel_FBD

Variables table

	Name	Type	Usage	Description
	⊟ IN			
	IN_wAnalog	WORD	VAR_INPUT	12Bit-analog value
	IN_xSelection	BOOL	VAR_INPUT	Switching percent/thousandth
	FU_Norm_FBD	STRING	VAR	

Programming

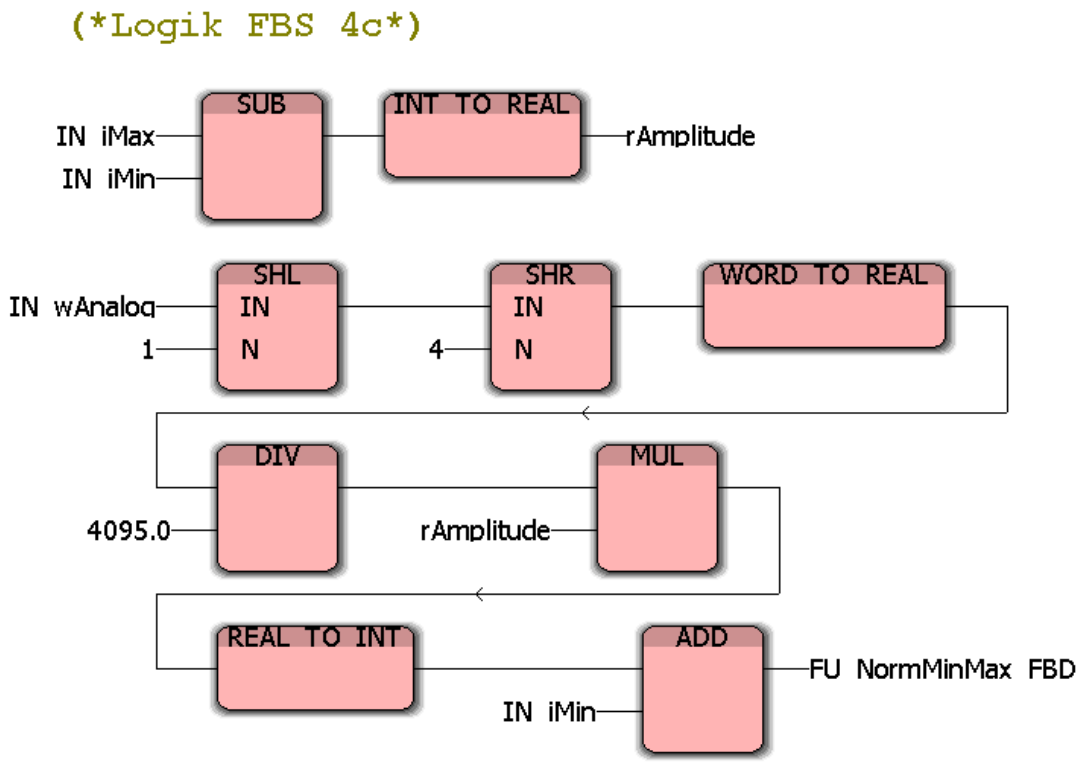


FBD4[c] *FU_NormMinMax_FBD*

Variables table

Name	Type	Usage	Description
IN			
IN_wAnalog	WORD	VAR_INPUT	12Bit-analog value
IN_iMin	INT	VAR_INPUT	Min value
IN_iMax	INT	VAR_INPUT	Max value
Local Variables			
rAmplitude	REAL	VAR	Amplitude of the scaled signal

Programming



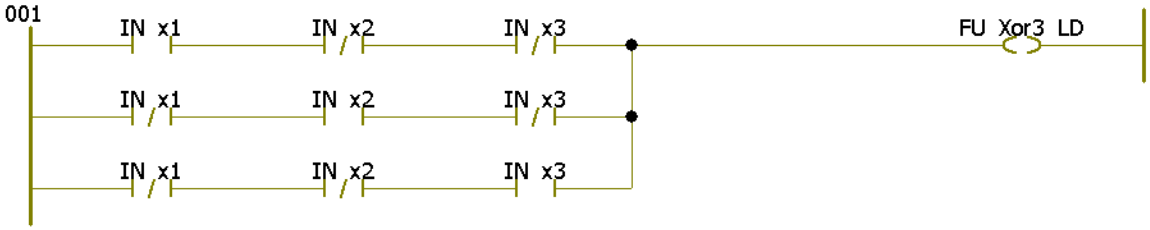
Solutions regarding programming in ladder diagram

LD1 *FU_Xor3_LD*

Variables table

	Name	Type	Usage	Description
	⊟ IN			
	IN_x1	BOOL	VAR_INPUT	
	IN_x2	BOOL	VAR_INPUT	
	IN_x3	BOOL	VAR_INPUT	

Programming

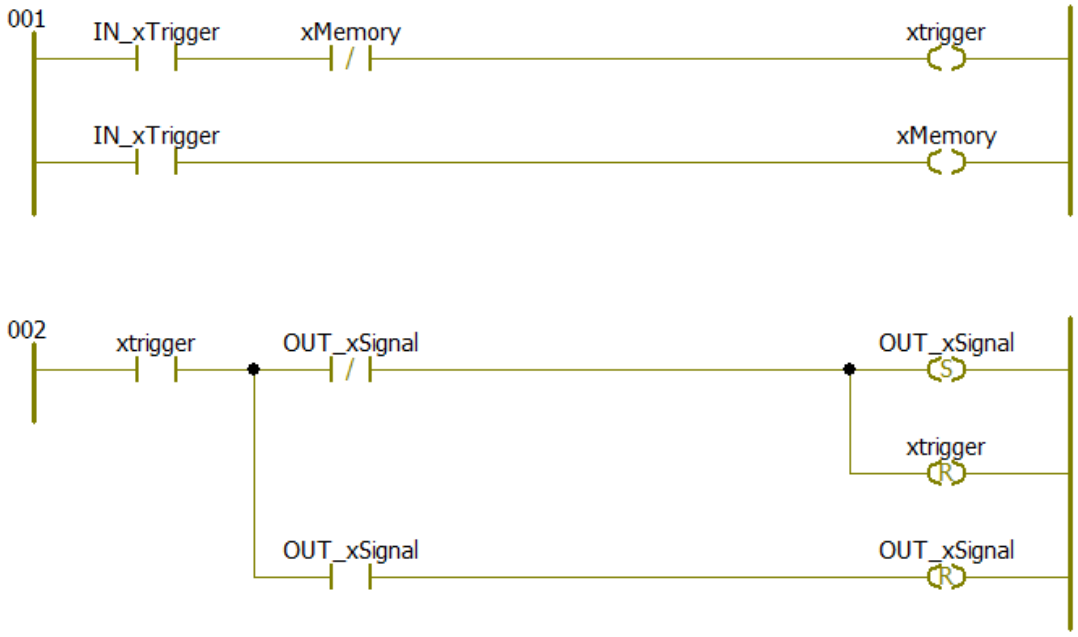


LD2 *FB_TFlipflop_LD*

Variables table

	Name	Type	Usage	Description
	IN			
	IN_xTrigger	BOOL	VAR_INPUT	
	Local variables			
	xtrigger	BOOL	VAR	
	xMemory	BOOL	VAR	
	OUT			
	OUT_xSignal	BOOL	VAR_OUTPUT	

Programming



Solutions regarding programming in instruction list

IL1 *FU_Xor3_IL*

Variables table

	Name	Type	Usage	Description
	IN			
	IN_x1	BOOL	VAR_INPUT	
	IN_x2	BOOL	VAR_INPUT	
	IN_x3	BOOL	VAR_INPUT	

Programming

```
1
2     Ld        IN_x1
3     AndN      IN_x2
4     AndN      IN_x3
5     Or(       IN_x1
6     Not
7     And       IN_x2
8     AndN      IN_x3
9     )
10    Or(       IN_x1
11    Not
12    AndN      IN_x2
13    And       IN_x3
14    )
15    St        FU_Xor3_IL
```

IL4[a] *FU_Norm_IL*

Variables table

	Name	Type	Usage	Description
	⊟ IN			
	IN_wAnalog	WORD	VAR_INPUT	12Bit-analog value

Programming

```
1
2     Ld        IN_wAnalog
3     Shr       3
4     And       Word#16#0FFF
5     Word_To_Real
6     Div       40.95
7     Real_To_Int
8     St        FU_Norm_IL
```

IL4[b] *FU_NormSel_IL*

Variables table

	Name	Type	Usage	Description
	⊟ IN			
	IN_wAnalog	WORD	VAR_INPUT	12Bit-analog value
	IN_xSelection	BOOL	VAR_INPUT	Switching percent/thousandth

Programming

```
1
2     Ld        IN_wAnalog
3     Shr       3
4     And       Word#16#0FFF
5     Word_To_Real
6     Div(      IN_xSelection
7     Sel       40.95, 4.095
8     )
9     Real_To_Int
10    St        FU_NormSel_IL
```

IL4[c] FU_NormMinMax_IL

Variables table

	Name	Type	Usage	Description
	IN			
	IN_wAnalog	WORD	VAR_INPUT	12Bit-analog value
	IN_iMin	INT	VAR_INPUT	Min value
	IN_iMax	INT	VAR_INPUT	Max value
	Local variables			
	rAmplitude	REAL	VAR	Amplitude of the scaled signal

Programming

```
1
2     Ld        IN_iMax
3     Sub       IN_iMin
4     Int_To_Real
5     St        rAmplitude
6
7     Ld        IN_wAnalog
8     Shl       1
9     Shr       4
10    Word_To_Real
11    Div       4095.0
12    Mul       rAmplitude
13    Real_To_Int
14    Add       IN_iMin
15    St        FU_NormMinMax_IL
16
```

IL3 FB_Cascade_cycle_IL

Variables table

	Name	Type	Usage	Description
	IN			
	IN_xActive	BOOL	VAR_INPUT	
	IN_tOn	TIME	VAR_INPUT	
	IN_tOff	TIME	VAR_INPUT	
	OUT			
	OUT_xSignal	BOOL	VAR_OUTPUT	
	FB-Instances			
	TON_On	TON	VAR	
	TON_Off	TON	VAR	

Programming

```
1     (* Logic IL 3 *)
2
3     Ld       IN_xActive
4     AndN     TON_Off.Q
5     St       TON_On.IN
6
7     Ld       IN_tOn
8     St       TON_On.PT
9
10    Cal      TON_On
11
12    Ld       TON_On.Q
13    St       TON_Off.IN
14
15    Ld       IN_tOff
16    St       TON_Off.PT
17
18    Cal      TON_Off
19
20    Ld       IN_xActive
21    AndN     TON_On.Q
22    St       OUT_xSignal
```

IL3 FB_Period_cycle_IL (alternative solution)

Variables table

Name	Type	Usage	Description
Global variables			
IN			
IN_xActive	BOOL	VAR_INPUT	
IN_tOn	TIME	VAR_INPUT	
IN_tOff	TIME	VAR_INPUT	
OUT			
OUT_xSignal	BOOL	VAR_OUTPUT	
FB-Instances			
TON_Period	TON	VAR	

Programming

```
1     (* Logic IL 3 *)
2
3     LdN      TON_Period.Q
4     And      IN_xActive
5     St       TON_Period.IN
6
7     Ld       IN_tOn
8     Add      IN_tOff
9     St       TON_Period.PT
10
11    Cal      TON_Period
12
13    Ld       TON_Period.ET
14    Lt       IN_tOn
15    And      IN_xActive
16    St       OUT_xSignal
```

IL2 *FB_TFlipflop_IL*

Variables table

	Name	Type	Usage	Description
	IN			
	IN_xTrigger	BOOL	VAR_INPUT	
	OUT			
	OUT_xSignal	BOOL	VAR_OUTPUT	
	Local variables			
	xMemory	BOOL	VAR	

Programming

```
1     (* Logic IL 2 *)
2
3     Ld       IN_xTrigger
4     AndN     xMemory
5     JmpCN    NoTrigger
6
7     Ld       OUT_xSignal
8     StN      OUT_xSignal
9
10    NoTrigger:
11    Ld       IN_xTrigger
12    St       xMemory
13
```

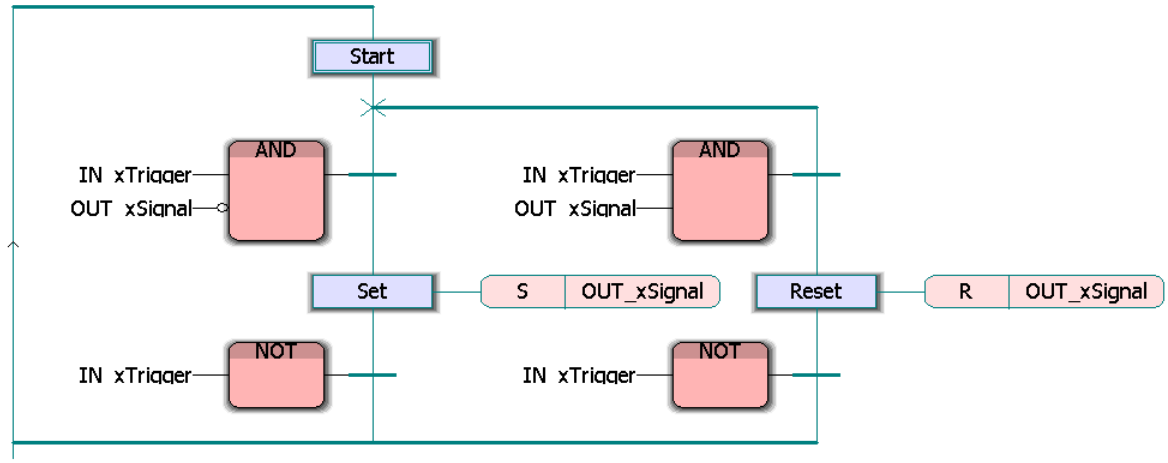
Solutions regarding programming in sequential function chart

SFC2 *FB_TFlipflop_SFC (no task assigned)*

Variables table

	Name	Type	Usage	Description
	IN			
	IN_xTrigger	BOOL	VAR_INPUT	
	OUT			
	OUT_xSignal	BOOL	VAR_OUTPUT	

Programming

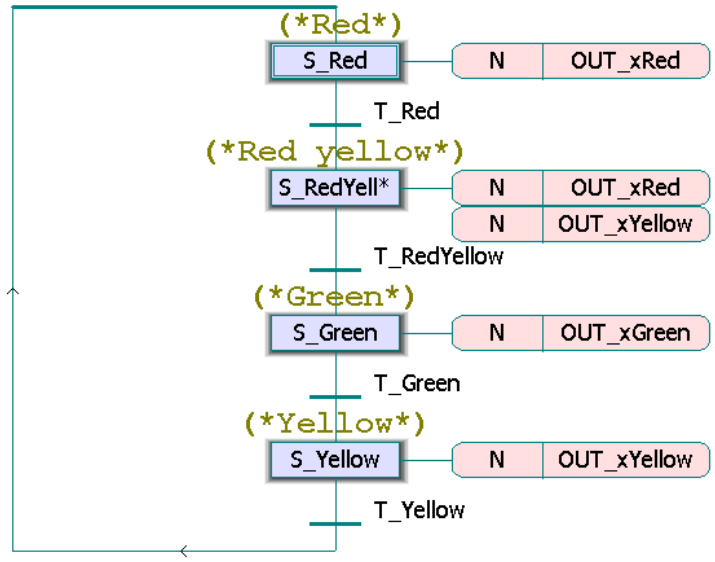


SFC5[a] FB_Traff_light_SFC

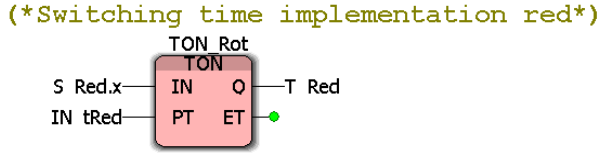
Variables table

	Name	Type	Usage	Description
	IN			
	IN_tRed	TIME	VAR_INPUT	
	IN_tRedYellow	TIME	VAR_INPUT	
	IN_tGreen	TIME	VAR_INPUT	
	IN_tYellow	TIME	VAR_INPUT	
	OUT			
	OUT_xRed	BOOL	VAR_OUTPUT	
	OUT_xYellow	BOOL	VAR_OUTPUT	
	OUT_xGreen	BOOL	VAR_OUTPUT	
	FB-Instances			
	TON_Red	TON	VAR	
	TON_RedYellow	TON	VAR	
	TON_Green	TON	VAR	
	TON_Yellow	TON	VAR	

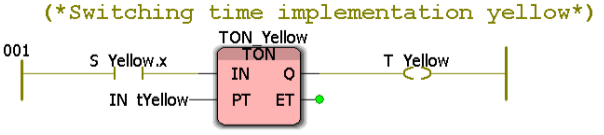
Programming



Detail programming




```
1    (* Switching time implementation red yellow *)
2
3    Ld      S_RedYellow.x
4    St      TON_RedYellow.IN
5    Ld      IN_tRedYellow
6    St      TON_RedYellow.PT
7
8    Cal     TON_RedYellow
9
10   Ld      TON_RedYellow.Q
11   St      T_RedYellow
```



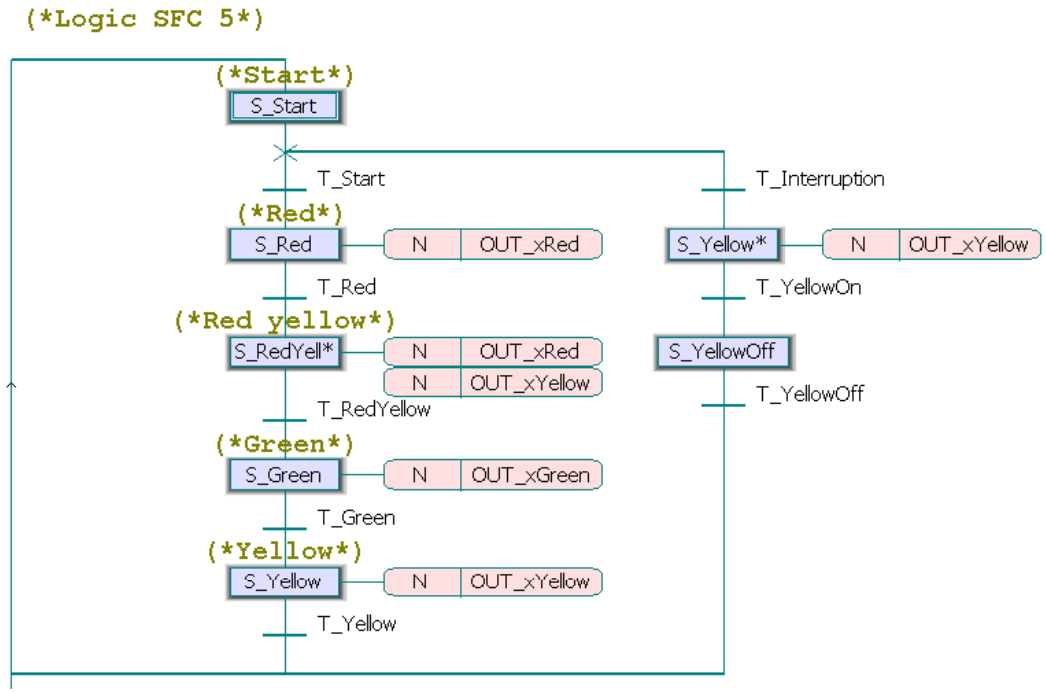

```
1    (* Switching time implementation green *)
2
3    TON_Green(IN    := S_Green.x,
4              PT    := IN_tGreen);
5
6    T_Green         := TON_Green.Q;
```

SFC[b] FB_Traffic_lights_SFC

Variables table

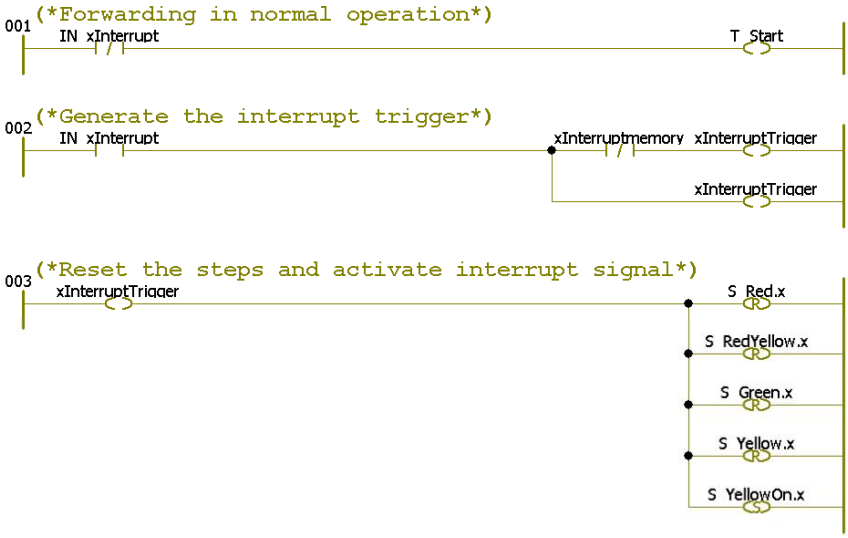
Name	Type	Usage	Description
IN			
IN_xInterrupt	BOOL	VAR_INPUT	
IN_tRed	TIME	VAR_INPUT	
IN_tRedYellow	TIME	VAR_INPUT	
IN_tGreen	TIME	VAR_INPUT	
IN_tYellow	TIME	VAR_INPUT	
OUT			
OUT_xRed	BOOL	VAR_OUTPUT	
OUT_xYellow	BOOL	VAR_OUTPUT	
OUT_xGreen	BOOL	VAR_OUTPUT	
Local Variables			
xInterruptTrigger	BOOL	VAR	
xInterruptmemory	BOOL	VAR	
FB-Instances			
TON_Red	TON	VAR	
TON_RedYellow	TON	VAR	
TON_Green	TON	VAR	
TON_Yellow	TON	VAR	
TON_YellowOn	TON	VAR	
TON_YellowOff	TON	VAR	

Programming



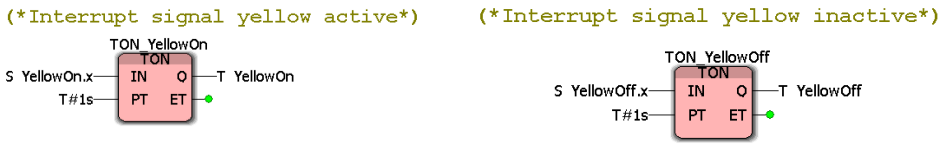
Detail programming (in addition to part [a])

T_Start



T_Interruption

T_YellowOn/YellowOff



Solutions regarding programming in structured text

ST1 *FU_Xor3_ST*

Variables table

	Name	Type	Usage	Description
	⊟ IN			
	IN_x1	BOOL	VAR_INPUT	
	IN_x2	BOOL	VAR_INPUT	
	IN_x3	BOOL	VAR_INPUT	

Programming

```
1
2     FU_Xor3_ST  :=      IN_x1 & Not IN_x2 & Not IN_x3
3                   Or    IN_x2 & Not IN_x3 & Not IN_x1
4                   Or    IN_x3 & Not IN_x1 & Not IN_x2;
```

ST4[a] FU_Norm_ST

Variables table

	Name	Type	Usage	Description
	IN			
	IN_wAnalog	WORD	VAR_INPUT	12Bit-analog value
	Local variables			
	rFormatedValue	REAL	VAR	

Programming

```
1    rFormatedValue          := Word_To_Real(Shr(IN_wAnalog, 3) & Word#16#0FFF);
2    FU_Norm_ST              := Real_To_Int(rFormatedValue / 40.95);
```

ST4[b] FU_NormSel_ST

Variables table

	Name	Type	Usage	Description
	IN			
	IN_wAnalog	WORD	VAR_INPUT	12Bit-analog value
	IN_xSelection	BOOL	VAR_INPUT	Switching percent/thousandth
	Local variables			
	rDivisor	REAL	VAR	
	rFormatedValue	REAL	VAR	

Programming

```
1    rFormatedValue          := Word_To_Real(Shr(IN_wAnalog, 3) & Word#16#0FFF);
2    rDivisor                := Sel(IN_xAuswahl, 40.95, 4.095);
3    FU_Norm_ST              := Real_To_Int(rFormatedValue / rDivisor);
```

ST4[c] FU_NormMinMax_ST

Variables table

	Name	Type	Usage	Description
	IN			
	IN_wAnalog	WORD	VAR_INPUT	12Bit-analog value
	IN_iMin	INT	VAR_INPUT	Min value
	IN_iMax	INT	VAR_INPUT	Max value
	Local variables			
	rAmplitude	REAL	VAR	Amplitude of the scaled signal

Programming

```
1    rAmplitude              := Int_to_Real(IN_iMax - IN_iMin);
2
3    rFormatedValue := Word_To_Real(Shr(IN_wAnalog, 3) & Word#16#0FFF);
4    FU_NormMinMax_ST        := Real_To_Int(rFormatedValue * 4095.0 * rAmplitude) + IN_iMin;
```

ST3 FB_Cascade_cycle_ST

Variables table

Name	Type	Usage	Description
IN			
IN_xActive	BOOL	VAR_INPUT	
IN_tOn	TIME	VAR_INPUT	
IN_tOff	TIME	VAR_INPUT	
OUT			
OUT_xSignal	BOOL	VAR_OUTPUT	
FB-Instances			

Programming

```
1
2
3     TON_ON(IN    := IN_xActive & Not TON_Off.Q,
4            PT    := IN_tOn);
5
6     TON_Off(IN   := TON_On.Q,
7            PT    := IN_tOff);
8
9     OUT_x        := IN_xActive & Not TON_On.Q;
```

ST3 FB_Period_cycle_ST (alternative solution)

Variables table

Name	Type	Usage	Description
IN			
IN_xActive	BOOL	VAR_INPUT	
IN_tOn	TIME	VAR_INPUT	
IN_tOff	TIME	VAR_INPUT	
OUT			
OUT_xSignal	BOOL	VAR_OUTPUT	
FB-Instances			
TON_Period	TON	VAR	

Programming

```
1
2
3     TON_Period(IN    := Not TON_Period & IN_xActive,
4                PT    := IN_tOn + IN_tOff);
5
6     OUT_xSignal      := (TON_Period.ET > IN_tOn) & IN_xActive;
```

ST2 FB_TFlipflop_ST

Variables table

	Name	Type	Usage	Description
	IN			
	IN_xTrigger	BOOL	VAR_INPUT	
	OUT			
	OUT_xSignal	BOOL	VAR_OUTPUT	
	Local Variables			
	xMemory	BOOL	VAR	

Programming

```
1
2
3     If IN_xTrigger & Not xMemory Then
4         OUT_xSignal := Not OUT_xSignal;
5     End_If;
6
7     xMemory := IN_xTrigger;
```

Alternative solution

```
1
2
3     OUT_xSignal := OUT_xSignal Xor IN_xTrigger & Not xMemory;
4     xMemory     := IN_xTrigger;
```

ST5 FB_Traffic_lights_ST

Variables table

	Name	Type	Usage	Description
	IN			
	IN_xInterrup	BOOL	VAR_INPUT	
	IN_tRed	TIME	VAR_INPUT	
	IN_tRedYellow	TIME	VAR_INPUT	
	IN_tGreen	TIME	VAR_INPUT	
	IN_tYellow	TIME	VAR_INPUT	
	OUT			
	OUT_xRed	BOOL	VAR_OUTPUT	
	OUT_xYellow	BOOL	VAR_OUTPUT	
	OUT_xGreen	BOOL	VAR_OUTPUT	
	Local Variables			
	tEndRedPhase	TIME	VAR	
	tEndRedYellowPhase	TIME	VAR	
	tEndGreenPhase	TIME	VAR	
	tEndYellowPhase	TIME	VAR	
	FB-Instances			
	TON_Phase	TON	VAR	
	TON_Yellow	TON	VAR	

Programming

```
(*****************************************************************************)
(* Resetting of the signals controlled by the textual step chain             *)
(*****************************************************************************)
OUT_xRed            := False;
OUT_xYellow         := False;
OUT_xGreen          := False;

(*****************************************************************************)
(* Period creation for standard mode                                         *)
(*****************************************************************************)
TON_Phase(IN        := Not IN_xInterrup & Not TON_Phase.Q,
          PT        := IN_tRed + IN_tRedYellow + IN_tGreen + IN_tYellow);

(*****************************************************************************)
(* Period creation for error mode                                            *)
(*****************************************************************************)
TON_Yellow(IN       := IN_xInterrup & Not TON_Yellow.Q,
           PT       := T#500ms);

(*****************************************************************************)
(* Creation of the switching times                                           *)
(*****************************************************************************)
tEndRedPhase        := IN_tRed;
tEndRedYellowPhase  := IN_tRed + IN_tRedYellow;
tEndGreenPhase      := IN_tRed + IN_tRedYellow + IN_tGreen;
tEndYellowPhase     := IN_tRed + IN_tRedYellow + IN_tGreen + IN_tYellow;

(*****************************************************************************)
(* Signal control after time elapsed and error state                         *)
(*****************************************************************************)
If IN_xInterrup Then

    OUT_xYellow     := TON_Yellow.ET < DIV_T_AI(TON_Yellow.PT, 2);

Else
    If TON_Phase.ET < tEndRedPhase Then

            OUT_xRed    := True;

    ElsIf TON_Phase.ET >= tEndRedPhase &
          TON_Phase.ET < tEndRedYellowPhase Then

            OUT_xRed    := True;
            OUT_xYellow := True;

    ElsIf TON_Phase.ET >= tEndRedYellowPhase &
          TON_Phase.ET < tEndGreenPhase Then

            OUT_xGreen  := True;

    ElsIf TON_Phase.ET >= tEndGreenPhase &
          TON_Phase.ET < tEndYellowPhase Then

            OUT_xYellow := True;
    End_If;
End_If;
```

ST6[a] Data type declaration

Worksheet for user-defined data types

```
1     Type
2         ST_Message       :    Struct
3                                 sDate        :    String;
4                                 sTime        :    String;
5                                 Segment      :    Int;
6                                 Position     :    Int;
7                                 Name         :    String;
8                             End_Struct;
9
10        AR_1_10_Message :    Array [1..10] Of ST_Message;
11    End_Type
```

ST6[b+c] PG_Messages

Local variables table

Name	Type	Usage	Description
Global variables			
stIBS	IBS_AR_1_512_DevInfo	VAR_EXTERNAL	
arMessageList	AR_1_10_Message	VAR_EXTERNAL	
Local variables [Date]			
strDate	STRING	VAR	
strYear	STRING	VAR	
strMonth	STRING	VAR	
strDay	STRING	VAR	
Locla variables [Time]			
strTime	STRING	VAR	
strHour	STRING	VAR	
strMinute	STRING	VAR	
strSecond	STRING	VAR	
Local variables [Messages]			
stActualMessage	ST_Message	VAR	
iSegment	INT	VAR	
iPosition	INT	VAR	
Local variables			
xFaultSearchEnd	BOOL	VAR	
xFaultSearchActive	BOOL	VAR	
iSNr	INT	VAR	
iIndex	INT	VAR	
iTarget	INT	VAR	
iSource	INT	VAR	
System variables			
MASTER_DIAG_PARAM_REG_HI	BYTE	VAR_EXTERNAL	
MASTER_DIAG_PARAM_REG_LOW	BYTE	VAR_EXTERNAL	
MASTER_DIAG_STATUS_REG_BUS	BOOL	VAR_EXTERNAL	
MASTER_DIAG_STATUS_REG_DTC	BOOL	VAR_EXTERNAL	
RTC_HOURS	INT	VAR_EXTERNAL	
RTC_MINUTES	INT	VAR_EXTERNAL	
RTC_SECONDS	INT	VAR_EXTERNAL	
RTC_DAY	INT	VAR_EXTERNAL	
RTC_MONTH	INT	VAR_EXTERNAL	
RTC_YEAR	INT	VAR_EXTERNAL	

Global variables table (extract)

User variables [Actual project]				
stIBS	IBS_AR_1_512_DevInfo	VAR_GLOBAL	INTERBUS Configuratio...	
arMessageList	AR_1_10_Message	VAR_GLOBAL	Message list	
Inserted from 'BIB_IBS_CFG_INFO_V1_0_D', POE 'PG_InterbusConfiguration'				
iCounfOfDevicesByFW	INT	VAR_GLOBAL		
iStartTabulatorForDetail	INT	VAR_GLOBAL		10
xCfgReadStart	BOOL	VAR_GLOBAL		

Programming

```
(*****************************************************************************************)
(* Troubleshooting ended at INTERBUS                                                     *)
(*****************************************************************************************)
xFaultSearchEnd :=  MASTER_DIAG_STATUS_REG_BUS
                        & Not MASTER_DIAG_STATUS_REG_DTC
                        & xFaultSearchActive;

If xFaultSearchEnd Then

    (*************************************************************************************)
    (* Convert the BYTE values to INT for the location                                   *)
    (*************************************************************************************)
    iSegment    :=  Word_To_Int(Byte_To_Word(MASTER_DIAG_PARAM_REG_HI));
    iPosition   :=  Word_To_Int(Byte_To_Word(MASTER_DIAG_PARAM_REG_LOW));

    (*************************************************************************************)
    (* Convert the INT values of the real time clock to STRING                           *)
    (*************************************************************************************)
    strDay      :=  Int_To_String(RTC_DAY, '%02d');
    strMonth    :=  Int_To_String(RTC_MONTH, '%02d');
    strYear     :=  Int_To_String(RTC_YEAR, '%02d');

    strHour     :=  Int_To_String(RTC_HOURS, '%02d');
    strMinute   :=  Int_To_String(RTC_MINUTES, '%02d');
    strSecond   :=  Int_To_String(RTC_SECONDS, '%02d');

    (*************************************************************************************)
    (* Assemble the converted STRING values                                          *)
    (*************************************************************************************)
    strDate :=  Concat(strDay, '.');
    strDate :=  Concat(strDate, strMonth);
    strDate :=  Concat(strDate, '.');
    strDate :=  Concat(strDate, strYear);

    strTime     :=  Concat(strHour, ':');
    strTime     :=  Concat(strTime, strMinute);
    strTime     :=  Concat(strTime, ':');
    strTime     :=  Concat(strTime, strSecond);

    (*************************************************************************************)
    (* Insert date and Time into the Message structure                                   *)
    (*************************************************************************************)
    stActualMessage.sDate := strDate;
    stActualMessage.stime := strTime;

    (*************************************************************************************)
```

```
47      (* Read the name from the IBS configuration structure                                    *)
48      (*************************************************************************************)
49      If iSegment = 0 and iPosition = 0 Then
50          stActualMessage.Segment := iSegment;
51          stActualMessage.Position := iPosition;
52          stActualMessage.Name    :=  'PLC';
53      Else
54          (*************************************************************************************)
55          (* consider of OUT1- and OUT2-faults                                                 *)
56          (*************************************************************************************)
57          Case iPosition Of
58          0..63:  For iSNr := 1 To 512 Do
59                      If stIBS[iSNr].Seg = iSegment &
60                         stIBS[iSNr].Pos = iPosition Then
61
62                          stActualMessage.Segment := iSegment;
63                          stActualMessage.Position := iPosition;
64                          stActualMessage.sDate := strDate;
65                          stActualMessage.sTime := strTime;
66                          stActualMessage.Name := stIBS[iSNr].Name;
67
68                          Exit;
69                      End_If;
70                  End_For;
71
72          128:    For iSNr := 1 To 512 Do
73                      If stIBS[iSNr].Seg = iSegment &
74                         stIBS[iSNr].Pos = 0 Then
75
76                          stActualMessage.Segment := iSegment;
77                          stActualMessage.Position := 0;
78                          stActualMessage.sDate := strDate;
79                          stActualMessage.sTime := strTime;
80                          stActualMessage.Name := Concat(stIBS[iSNr].Name, '-OUT1');
81
82                          Exit;
83                      End_If;
84                  End_For;
85
86          129:    For iSNr := 1 To 512 Do
87                      If stIBS[iSNr].Seg = iSegment &
88                         stIBS[iSNr].Pos = 0 Then
89
90                          stActualMessage.Segment := iSegment;
91                          stActualMessage.Position := 0;
92                          stActualMessage.sDate := strDate;
93                          stActualMessage.sTime := strTime;
94                          stActualMessage.Name := Concat(stIBS[iSNr].Name, '-OUT2');
95
96                          Exit;
97                      End_If;
98                  End_For;
99          End_Case;
100     End_If;
101
102     (*************************************************************************************)
103     (* Update the message list                                                           *)
104     (*************************************************************************************)
105     For iIndex := 1 To 9 Do
106         iSource := 10 - iIndex;
107         iTarget := iSource + 1;
108
109         arMessageList[iTarget]  := arMessageList[iSource];
110     End_For;
111
112     (*************************************************************************************)
113     (* enter the actual message into the message list                                    *)
114     (*************************************************************************************)
115     arMessageList[1]    := stActualMessage;
116 End_If;
117
118 (*************************************************************************************)
119 (* Update the fault search memory                                                    *)
120 (*************************************************************************************)
121 xFaultSearchActive := MASTER_DIAG_STATUS_REG_DTC;
```