SIEMENS

WinCC

组态手册

第二册

订货号：6AV6 392-1CA05-0AH0
C79000-G8276-C164-01

发行：1999 年 9 月

WinCC、SIMATIC、SINEC、STEP 是西门子注册商标。

本手册中所有其它的产品和系统名称是（注册的）其各自拥有者的商标，必须被相应地对待。

（若没有快速写入权限，不允许对本文件或其内容进行复制、传送或使用。
违犯者将要对损坏负责任。保留所有权利，包括由专利授权创建的权利，对实用新型或设计的注册。）

（我们已检查了本手册的内容，使其与硬件和软件所描述的相一致。由于不可能完全消除差错，我们也不能保证完全的一致性。然而，本手册中的数据是经常规检查的，在以后的版本中包括了必要的修正。欢迎给我们提出建议以便改进。）

©Siemens AG 1994 - 1999 保留所有权利　　改变的技术数据

C79000-G8276-C164

目录

1 启动实例 1-1

1.1 下载实例 1-1
1.2 启动实例(单用户项目) 1-3

2 变量的组态(Project_TagHandling) 2-1

2.1 变量的创建、分组和移动 2-2
2.2 递增、递减、按击 2-7
2.2.1 按击 - 更改设定值(实例 01) 2-8
2.2.2 按击 - 通过全局脚本更改设定值(实例 02) 2-9
2.2.3 按击 - 按钮(实例 05) 2-12
2.2.4 按击 - 切换开关(实例 06) 2-16
2.2.5 递增和递减(实例 01) 2-18
2.2.6 通过全局脚本递增和递减(实例 02) 2-21
2.2.7 本主题的其余实例 2-25
2.3 通过 Windows 对象对变量值进行修改 2-26
2.3.1 通过带有直接连接的滚动条进行输入(实例 01) 2-26
2.3.2 通过滚动条和变量连接进行输入(实例 03) 2-29
2.3.3 通过选项组(选项钮)进行输入(实例 02) 2-30
2.3.4 通过复选框进行输入(实例 04) 2-32
2.4 对字中的位进行处理 2-35
2.4.1 直接通过复选框和直接连接进行置位(实例 06) 2-35
2.4.2 选择一个位并更改其状态(实例 01) 2-38
2.4.3 本主题的其余实例 2-41
2.5 变量的间接寻址 2-41
2.5.1 通过直接连接进行间接寻址(实例 01) 2-42
2.5.2 使用间接寻址和 C 动作进行多重显示(实例 02) 2-44
2.5.3 使用 C 动作进行间接寻址(实例 03) 2-46
2.5.4 本主题的其余实例 2-47
2.6 变量的模拟 2-47
2.6.1 通过 C 动作对三角振荡进行模拟(实例 01) 2-48
2.6.2 通过外部程序进行模拟(实例 02) 2-50
2.7 导入/导出变量 2-52
2.8 结构变量的使用 2-54
2.8.1 使用结构变量对阀进行控制(实例 01) 2-54

3 画面组态(Project_CreatePicture) 3-1

3.1 画面布局与画面切换 3-3
3.1.1 画面布局 3-3

3.2 画面切换 3-5
3.2.1 通过直接连接打开画面并显示画面名称(实例 01) 3-5
3.2.2 利用动态向导打开画面(实例 02) 3-9
3.2.3 通过内部函数打开画面(实例 02) 3-11
3.2.4 通过动态向导进行单个画面的切换(实例 03) 3-12
3.2.5 利用直接连接切换单个画面(实例 04) 3-14
3.2.6 通过对象名称和内部函数打开画面(05) 3-16
3.2.7 通过对象名称以及与画面名称显示的变量连接来打开画面(实例 06) 3-18
3.3 显示画面窗口 3-21
3.3.1 从画面窗口外隐藏(撤消选择)和显示(选择)(实例 01) 3-21
3.3.2 从画面窗口外显示(选择)和从画面窗口内隐藏(撤销选择)(实例 02) 3-23
3.3.3 对画面进行时控隐藏(实例 03) 3-24
3.3.4 在按下鼠标右键时对画面窗口进行显示(实例 04) 3-26
3.3.5 使用向导对信息框进行组态(实例 05) 3-27
3.3.6 显示用于文本输入的对话框(实例 06) 3-30
3.4 操作控制允许 3-32
3.4.1 退出运行系统和系统(实例 01) 3-32
3.4.2 操作员控制允许、使用缺省框进行登录(实例 02) 3-33
3.4.3 通过单独对话框进行的操作员控制允许、登录(实例 03) 3-37
3.5 画面缩放 3-39
3.5.1 在两种尺寸之间改变画面几何结构(实例 01) 3-39
3.5.2 连续更改画面几何结构(实例 02) 3-41
3.5.3 通过属性对话框组态可调整的画面几何结构(实例 03) 3-44
3.6 Windows 控制中心 3-45
3.6.1 二进制切换操作(两步控制)(实例 01) 3-45
3.6.2 二进制 S-R 切换操作(两步控制)(实例 02) 3-47
3.6.3 确认的二进制切换操作(实例 03) 3-48
3.6.4 自动输入检查(实例 04) 3-50
3.6.5 增强型自动输入检查(实例 05) 3-52
3.6.6 多重操作(实例 06) 3-56
3.7 动态化 3-60
3.7.1 颜色更改(实例 01) 3-60
3.7.2 文本切换(实例 02) 3-62
3.7.3 移动过程的动画(实例 03) 3-63
3.7.4 使用位判断来显示和隐藏对象(实例 04) 3-64
3.7.5 利用 C 动作的移动过程动画(实例 05) 3-65
3.7.6 使用向导创建移动过程的动画(实例 06) 3-66
3.7.7 通过 C 动作更改颜色(实例 06) 3-68
3.7.8 利用状态显示的移动过程动画(实例 07) 3-69
3.8 语言切换 3-71
3.8.1 运行系统语言切换(实例 01) 3-72
3.8.2 用于运行系统与控制中心语言切换的对话框(实例 02) 3-73
3.9 无鼠标时的操作 3-74

3.9.1 利用 TAB 键或热键进行操作(实例 01) ... 3-74
3.9.2 光标键盘(实例 02) ... 3-82
3.9.3 值的输入、操作的切换(例 03) ... 3-86
3.10 信息的显示与隐藏 ... 3-89
3.10.1 显示和隐藏对象(实例 01) ... 3-89
3.10.2 日期和时间的显示(实例 02) ... 3-91

4 WinCC 编辑器(Project_WinCCEditors) ... 4-1

4.1 变量记录 ... 4-2
4.1.1 周期连续的归档(ex_3_chapter_01.pdl) ... 4-3
4.1.2 周期选择归档(ex_3_chapter_01a.pdl) ... 4-18
4.1.3 如果超出数值就进行归档(ex_3_chapter_01b.pdl) ... 4-27
4.1.4 用户定义的表格布局(ex_3_chapter_01c.pdl) ... 4-40
4.1.5 归档二进制变量(ex_3_chapter_01d.pdl) ... 4-48
4.1.6 以定义的时间进行归档(ex_3_chapter_01e.pdl) ... 4-55
4.1.7 归档的导出(ex_3_chapter_01f.pdl) ... 4-61
4.2 报警记录 ... 4-69
4.2.1 位消息过程(ex_3_chapter_02.pdl) ... 4-70
4.2.2 限制值的监控(ex_3_chapter_02a.pdl) ... 4-83
4.2.3 限制值的监控(续) ... 4-88
4.2.4 消息窗口(ex_3_chapter_02b.pdl) ... 4-102
4.2.5 消息归档(ex_3_chapter_02c.pdl) ... 4-107
4.2.6 组消息(ex_8_generator_00.pdl) ... 4-114
4.3 报表编辑器 ... 4-121
4.3.1 画面文档(ex_3_chapter_03.pdl) ... 4-121
4.3.2 WinCC 资源管理器的报表(ex_3_chapter_03.pdl) ... 4-128
4.3.3 变量记录 CS 的报表(ex_3_chapter_03.pdl) ... 4-131
4.3.4 在运行系统中打印输出趋势窗口(ex_3_chapter_01a.pdl) ... 4-133
4.3.5 在运行系统中打印输出表格(ex_3_chapter_01c.pdl) ... 4-138
4.3.6 消息顺序报表(ex_3_chapter_02b.pdl) ... 4-142
4.3.7 行式打印机上的消息顺序报表 ... 4-145
4.3.8 消息归档报表(ex_3_chapter_02c.pdl) ... 4-147
4.4 与 EXCEL 的 OLE 通讯 ... 4-149
4.4.1 读和写变量数值(ex_3_chapter_04.pdl) ... 4-149
4.5 实例中的附加组态 ... 4-152
4.5.1 画面索引 ... 4-152
4.5.2 索引 ... 4-156
4.5.3 颜色对话框(ex_3_chapter_01c) ... 4-159
4.5.4 棒图显示(ex_3_chapter_01e) ... 4-162

前言

本手册的目的

本手册通过下列章节来介绍 WinCC 的组态选项：

本手册采用印刷版和电子手册形式出版。

目录表和索引可以帮助您快速找到需要的信息。而且，在线文件包含了附加的搜索功能。

使用本手册的先决条件

具有 WinCC 基础知识 (例如，通过学习使用入门) 或具备使用 WinCC 的实际组态经验。

附加支持

如果存在技术问题，请与 当地 Siemens 分公司联系。

此外，您可以拨打热线电话
+49 (911) 895-7000 (传真 7001)

关于 SIMATIC 产品的信息

可以在 CA01 目录中获得关于 SIMATIC 产品的最新信息。可通过下列 Internet 地址访问此目录：

http://www.ad.siemens.de/ca01online/

此外，SIMATIC 客户支持提供了最新的信息并提供下载。从下列 Internet 地址可查找有关技术咨询的解答：

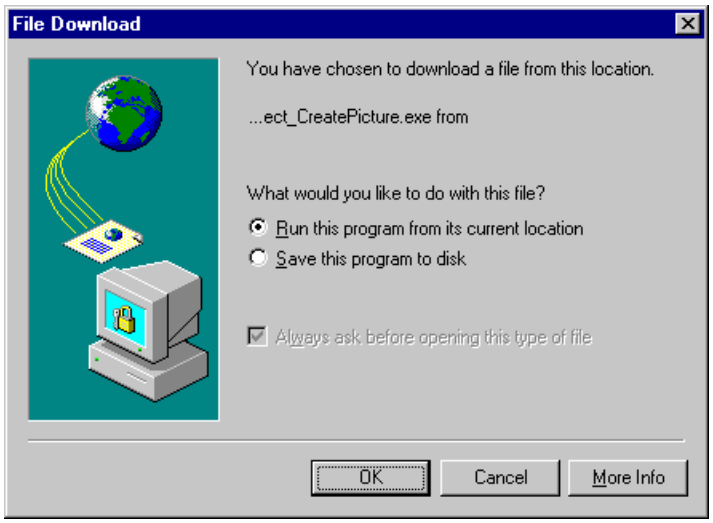
http://www.aut.siemens.de/support/html_00/index.shtml

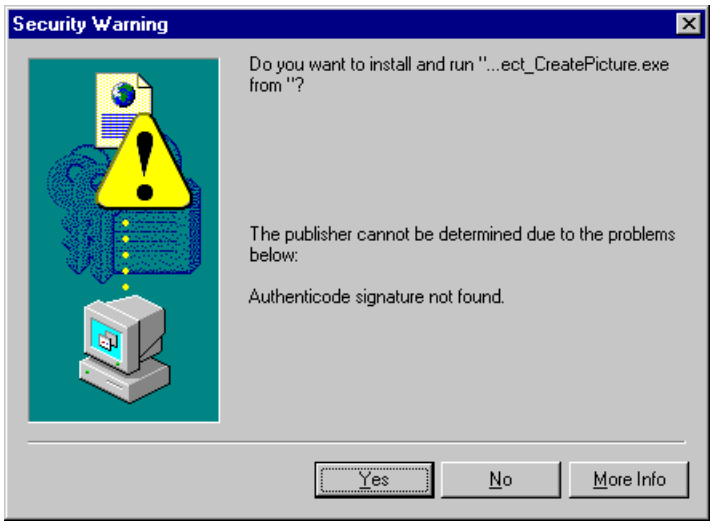
1 启动实例

在本节中将描述基于实例项目的 WinCC 组态步骤。考虑到 WinCC 可以提供许多潜在的应用，下面所描述的项目只是看作可以用 WinCC 组态的实例。
本节中手动创建的 WinCC 项目也可以直接从在线文档复制到用户的硬盘驱动器中。缺省情况下，它们将存储在 *C:\Configuration_Manual* 文件夹中。以下表格中列出了启动 WinCC 项目必需的步骤。

1.1 下载实例

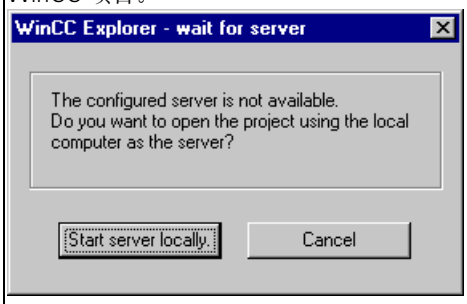
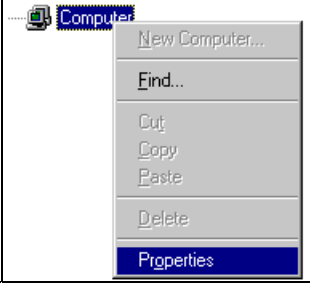
下载实例

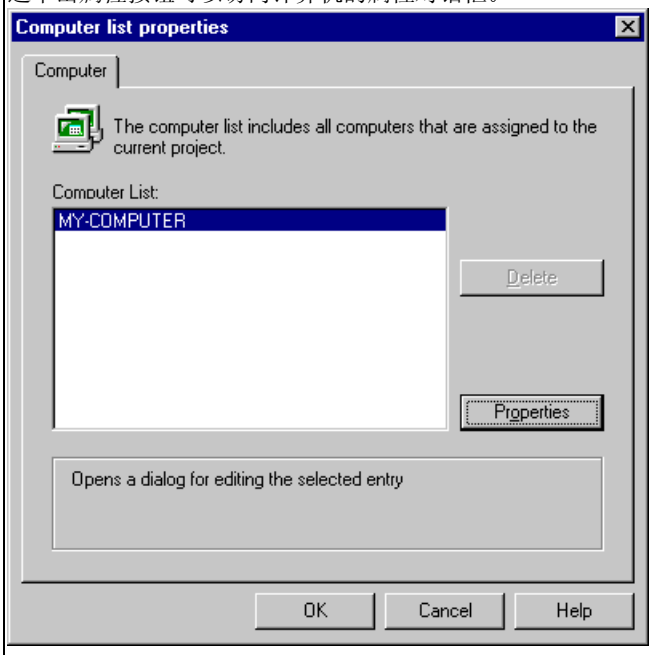
步骤	过程：下载实例
1	下载期望的项目。这可以从在线文档中通过单击以下图标来完成： 项目名称
2	将显示*下载文件*对话框。在此对话框中选择*从该位置执行程序*条目。通过单击*确定*来确认对话框。 

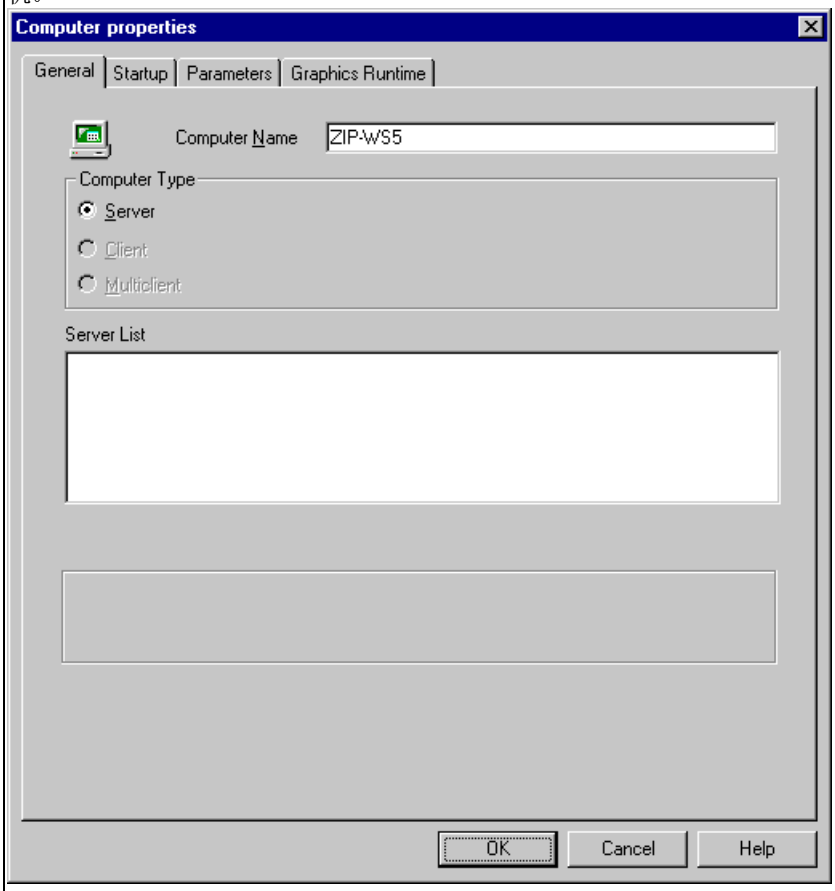
步骤	过程：下载实例
3	将显示*安全警告*对话框。通过单击*是*确认此对话框。 
4	将打开 *WinZip Self-Extractor* 对话框。用户可以指定将项目解压缩至某个文件夹。缺省情况下，项目将解压缩至 *C:\Configuration_Manual* 文件夹中。通过单击 *Unzip* 按钮来启动解压缩过程。
5	解压缩过程结束之后，将出现一个用于确认文件成功解压缩的对话框。通过单击*确定*来确认此对话框。通过*关闭*按钮来关闭 *WinZip Self-Extractor* 对话框。

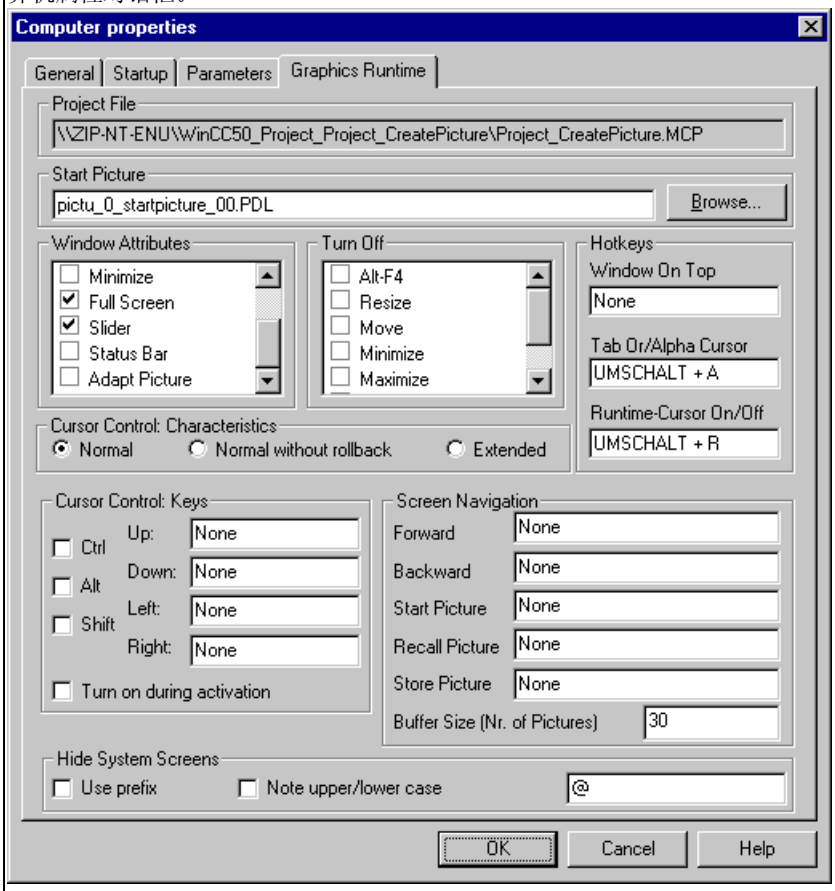
1.2 启动实例(单用户项目)

启动实例(单用户项目)

步骤	过程：启动实例(单用户项目)
1	打开 *WinCC 资源管理器*。打开刚才解压缩好的实例项目。 将会显示一个对话框，指出所组态的服务器不可用。通过*启动本地服务器*打开 WinCC 项目。 
2	为了能够使用项目，必须输入本地服务器的名称作为计算机名。这可以在 *WinCC 资源管理器*中通过以下方法来完成，即 R *计算机*条目，然后从*弹出式菜单*中选择*属性*。 

步骤	过程：启动实例(单用户项目)
3	将打开*计算机列表属性*对话框。计算机列表将显示与项目有关的所有计算机。通过单击*属性*按钮可以访问计算机的属性对话框。 

步骤	过程：启动实例(单用户项目)
4	将打开计算机的属性对话框。在*常规信息*标签中用本地计算机代替输入的计算机。 

步骤	过程：启动实例(单用户项目)
5	在*图形运行*系统标签中确保所有设置都正确。其中包括检查是否已指定启动画面。如果用低于 1024 x 768 的分辨率来显示项目，则必须在窗口属性域中选择*全屏幕*和*滚动条*复选框。通过单击*确定*退出对话框。同样可以通过单击*确定*退出计算机属性对话框。 
6	项目在激活之前必须重新装载。通过文件 → 关闭先将项目关闭，然后再将它打开。

注意：
刚才所描述的步骤可以直接应用于单用户项目。本手册中所描述的多用户项目也可以遵照这些步骤，但是还必须执行一些附加的步骤，它们将在相关的实例中详细说明。

2 变量的组态(Project_TagHandling)

本章中所创建的 WinCC 项目也可直接从在线文档复制到用户的硬盘上。缺省情况下，它将存储在 C:\Configuration_Manual 文件夹中。

Project_TagHandling

在该项目中，将可以找到各种不同的提示，它们使得在 WinCC 中使用变量变得更容易。通常，WinCC 处理三种不同类型的变量。它们是无过程驱动程序连接的内部变量、具有过程驱动程序连接的 WinCC 变量(也称作外部变量)以及在编制的 C 动作、项目函数等中的 C 变量。有关 Project_TagHandling 项目的实例主要处理内部变量。这些变量的常规处理并不是完全不同于 WinCC 变量的处理。

有关该主题的实例在 WinCC 项目 Project_TagHandling 中进行组态。其起始页面如下所示。

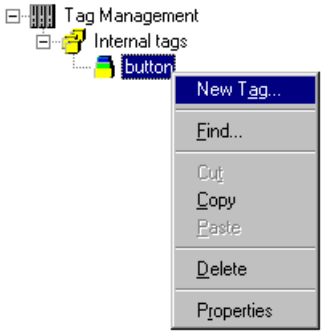
2.1 变量的创建、分组和移动

在 *WinCC 资源管理器*中，可在*变量管理器*条目下创建变量。要区分无过程驱动程序连接的变量(所谓的内部变量)与有过程驱动程序连接的变量(所谓的 *WinCC 变量*或*外部变量*)。对于可组态的内部变量，其最大数目没有任何限制。然而，*WinCC 变量*的最大数目受所获得的软件授权限制。

变量组与变量

处理大量数据从而需要许多变量时，建议将这些变量组织为变量组。只有这样才可以在大型项目中始终注意各种事件。然而，变量组并不保证变量的唯一性。只有通过变量名才可以达到此目的。

步骤	过程：变量组与变量
1	创建*内部变量*的变量组在*变量管理器*中通过以下方法来完成，即 R *内部变量*条目，然后从弹出式菜单中选择*新建组...*。
2	在所显示的对话框中，必须给该组一个合适的名称。在 *WinCC 资源管理器*中，将会显示带刚分配的名称的新组图标。 在实例项目 *Project_TagHandling* 中，已经根据所涉及的章节完成分类成组。

步骤	过程：变量组与变量
3	在变量组中创建变量可以通过以下方法来完成，即 R 相应组的条目，然后从弹出式菜单中选择*新建变量...*。 
4	在所显示的对话框中，在*常规信息*标签内为变量分配一个名称。从下面的列表框中，选择所期望的*数据类型*。不必为内部变量设置*地址*。

注意：
当激活运行系统时，通过工具提示可在 WinCC 资源管理器内显示过程画面中变量的当前值和状态。

移动变量

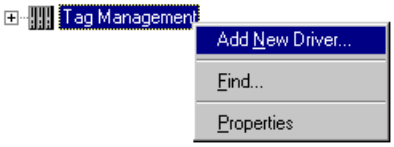
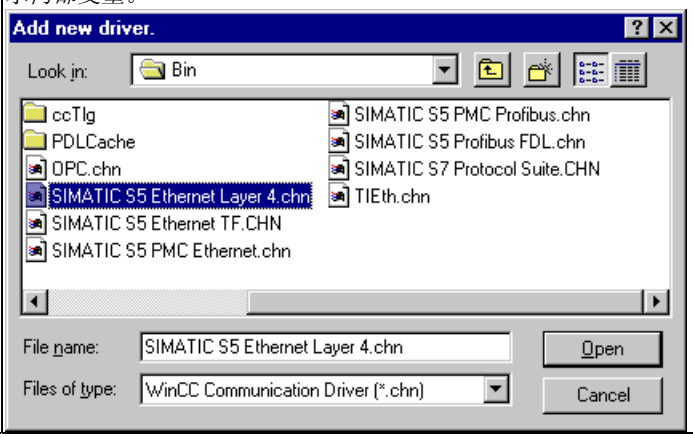
步骤	过程：移动变量
1	在*变量管理器*中，通过 R 变量然后从弹出式菜单中选择*剪切*来移动它。 之后，选择所期望的目标组。在所选的目标组处通过 R 然后从弹出式菜单中选择*粘贴*将变量插入。 同时处理多个变量时也可按照同样的步骤进行。

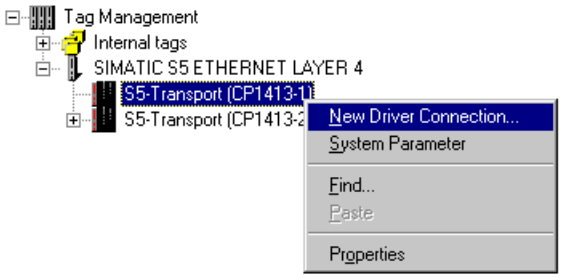
注意：
如果从 *WinCC 资源管理器*中剪切或删除变量，则不得激活运行系统。

如果需要许多具有相同变量名但连续进行编号的变量，则只要创建一个该类型的变量。
通过 R 并从弹出式菜单中选择*复制*可将该变量复制到剪贴板上，然后每当需要时就可以将其插入。变量将以升序方式自动编号。为变量定义名称约定时应该考虑到这种可能性。

WinCC 变量

要在*变量管理器*中创建 *WinCC 变量*，首先必须组态一个与 PLC 的连接。但是，不必安装硬件。安装所期望的通讯驱动程序并组态期望的连接就足够了。

步骤	过程：WinCC 变量
1	安装新的通讯驱动程序。这可以通过 R *变量管理器*并从弹出式菜单中选择*添加新驱动程序...*来完成。 
2	从所显示的对话框中，选择所期望的驱动程序。通过单击打开按钮将驱动程序插入 WinCC 项目中。 *WinCC 资源管理器*现在即可将新驱动程序条目显示在*变量管理器*中，而不是仅显示*内部变量*。 

步骤	过程：WinCC 变量
3	通过 D 新驱动程序条目，可显示一个或几个子条目(所谓的*通道单元*)。 创建一个连接。这通过 R *通道单元*条目并从弹出式菜单中选择*新建驱动程序连接*来完成。 
4	在所显示的对话框中，在*常规信息*标签内为连接分配一个名称。 连接参数可通过单击*属性*按钮来进行设置。
5	通过 R 新添加的连接条目，可按上面所描述的方式添加变量组和变量。
6	当创建 *WinCC 变量*时，除定义*内部变量*所需的设置以外，还必须定义地址和格式调整设置。地址请参考 PLC 中变量的地址。

结构变量

结构变量用于将构成一个逻辑单元的大量不同的变量与变量类型组织成一个组。这样就可使用一个名称对这些变量与变量类型进行寻址。
一个结构变量由许多单个变量组成，这些单个变量可以代表各种不同的数据类型。

步骤	过程：结构变量
1	通过 R *结构类型*条目并从弹出式菜单中选择*新建结构类型*来创建一个新的结构。

步骤	过程：结构变量
2	在所显示的对话框中，通过 R *新结构*条目并从弹出式菜单中选择*重命名*来给结构一个新的名称。 NewStructure Delete Rename
3	通过*新建元素*按钮可以添加新的结构元素。
4	通过 R 新创建的元素，可指定其数据类型和名称。对于每个结构元素，必须对其是*内部变量*还是*外部变量*进行定义。通过单击*确定*按钮结束组态并创建结构类型。 Structure properties General OS Use the formats from unit: No unit NewStructure SHORT NewTag Delete Rename BIT CHAR BYTE SHORT WORD LONG DWORD FLOAT DOUBLE TEXT8 TEXT16 External tag Type Conversion ShortToSignedWord New Element Delete Element AS Length 2 Offset 0 Bit 0 Linear scaling Process Value Range From To Tag Value Range From To Choose OS structure elements. OK Cancel Help

注意：
一旦结构类型创建完毕，以后就不能再对其进行组态。必须再定义完整的结构类型。

创建结构变量的方法与创建所有其它类型变量的方法一样，但是数据类型必须使用所创建的结构类型。所创建的结构变量的各元素名称由创建变量时分配的结构名称和创建结构类型时分配的元素名称组成。这两者在名称中用一个圆点隔开。

Name	Type	Parameters	Last change
STUi_varia_str_00.aktivated	Binary Tag	Internal tag	07/10/97 02:26:14
STUi_varia_str_00.open	Binary Tag	Internal tag	07/10/97 02:26:14
STUi_varia_str_00.closed	Binary Tag	Internal tag	08/21/97 03:45:32
STUi_varia_str_00.error	Binary Tag	Internal tag	07/10/97 02:26:14

2.2 递增、递减、按击

在运行时，有关该主题的实例可以在 Project_TagHandling 项目中通过用选择如上所示的按钮来访问。这些实例组态于 varia_3_chapter_01.pdl 和 varia_3_chapter_01a.pdl 画面中。

定义

- 递增指的是按固定或变化的增量增加一个变量的值。
- 递减指的是按固定或变化的减量减少一个变量的值。
- 按击指的是在按下按钮时执行一个动作，可将其比作按下按钮。对二进制信号而言，这通常表示对设备进行控制。对于模拟值，可通过按击来更改设定值。

2.2.1 按击 - 更改设定值(实例 01)

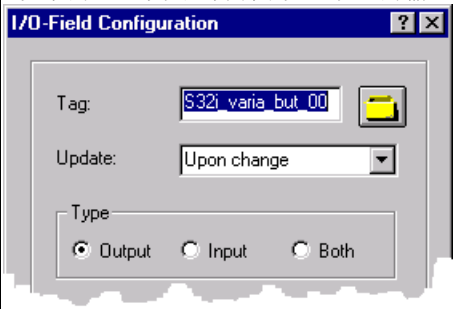
任务定义

按击要用鼠标来执行。
通过单击按钮来以固定的步长对设定值进行修改。这种数值更改要受固定限制值的约束。更改只能在画面中局部进行。

概念的实现

为了实现设定值的更改，使用两个 *Windows 对象* → *按钮*，这样就可以通过事件驱动的按钮来更改设定值。当用按下按钮*按钮*时，*内部变量*的值会增加一个增量。增量是预先指定的，运行期间不可改变。设定值的更改通过一个 *C 动作*来实现。
通过智能对象 → I/O 域显示设定值的变化。I/O 域的输出值与内部变量相连。

图形编辑器中的实现

步骤	过程：图形编辑器中的实现
1	在变量管理器中创建一个有*符号的 32 位数类型*的变量。在本实例中，*使用了变量 S32i_varia_but_00*。
2	在画面中，组态*智能对象* → *I/O 域*。在本实例中，使用 *I/O 域 1* 对象。在*组态对话框中*组态 *I/O 域*期间，设置 *S32i_varia_but_00* 变量。将*更新域*中的缺省值 2 s 更改为*一旦改变*，并将*域类型*设置为*输出*。 
3	在同一画面中，组态 *Windows 对象* → *按钮*。在本实例中，使用*按钮 2* 对象。
4	为了更改设定值，在*事件* → *鼠标* → *按下左键*处创建一个 *C 动作*。每次用单击按钮时该 *C 动作*均要更改变量的值。在 *C 动作*中可指定限制值并对其进行检查。
5	以同样的方式组态设定值的减量。在本实例中，为此使用了*按钮 1* 对象。

按钮 2 的 C 动作

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
DWORD value;

value=GetTagDWord("S32i_varia_but_00"); //get tag value
if (value>1300) (value=1400);           //check limit
else value=value+100;                   //inc value
SetTagDWord("S32i_varia_but_00",value); //set new value
}
```

- 声明 *C 变量值*。
- 使用*内部函数 GetTagDWord* 来读出变量 *S32i_varia_but_00* 的当前值。
- 在 *if* 语句中，检查变量值是否大于 1300。如果是，则将 1400 指定为上限值。如果变量的值小于 1300，则执行 else 分支中的语句，且将值增加 100。
- 然后，*内部函数 SetTagDWord* 将更改后的值写回变量 *S32i_varia_but_00* 中。

常规应用的注意事项

更改变量(内部或外部变量)、限制值和增量之后，两个*按钮*处的 *C 动作*均可使用。

2.2.2 按击 - 通过全局脚本更改设定值(实例 02)

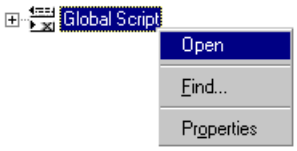
任务定义

按击要用鼠标来执行。
通过单击按钮来以固定的步长对设定值进行修改。这种数值更改要受固定限制值的约束。这可借助*项目函数*来实现。

概念的实现

为了实现设定值的更改，使用两个 *Windows 对象* → *按钮*，这样就可以通过事件驱动的按钮来更改设定值。当用按下按钮*按钮*时，*内部变量*的值会增加一个增量。增量是预先指定的，运行期间不可改变。设定值的更改可通过一个*项目函数*来实现。
通过智能对象 → I/O 域显示设定值的变化。I/O 域的输出值与内部变量相连。

创建项目函数

步骤	过程：创建项目函数
1	在 *WinCC 资源管理器*中通过以下方法启动*全局脚本*编辑器，即通过 OR *全局脚本*条目，然后从弹出式菜单中选择*打开*。 
2	通过*文件* → *新建项目函数*菜单来创建新函数。
3	为函数指定名称 *IncDecValue*，并通过选择*文件* → *另存为* → *IncDecValue.fct* 来保存函数。
4	编写和编译函数。

项目函数 IncDecValue

```
void IncDecValue(DWORD *value,DWORD low,DWORD high,DWORD step,DWORD a)

{
DWORD v;
v=*value; //get current value
switch (a){
   case 0: {
         if (v<step) (v=0); //low limit
         else v=v-step;     //decrement
         } //case 0
         break;
   case 1:{
         if (v>(high-step))
            (v=high);       //high limit
         else v=v+step;     //increment
         }//case 1
         break;
   }//switch
*value=v; //return
}
```

- 函数标题具有项目函数名 *IncDecValue* 和传送参数。递增和递减都使用同一个*项目函数*。
- 变量的声明。
- 在调用函数时，作为传送参数进行传送的不是要处理的变量，而只是其地址。该地址的内容被读入 *C 变量 v* 中。
- 使用 *switch* 语句来判断方向变量 *a* 的信息。
- 在相关的 *case* 分支中，检查限制值，如果超出限制，则指定最大值或最小值。
- 如果没有超出限制，则更改当前值。
- 将当前的设定值传送到要处理的变量的地址中。

图形编辑器中的实现

步骤	过程：图形编辑器中的实现
1	在变量管理器中创建一个*有符号的 32 位数类型*的变量。在本实例中，*使用变量 S32i_varia_but_04*。
2	在画面中，组态*智能对象* → *I/O 域*。 在本实例中，使用 *I/O 域 2* 对象。在*组态对话框*中组态 *I/O 域*期间，设置 *S32i_varia_but_04* 变量。将*更新*域中的缺省值 2 s 改为*一旦改变*，并将*域类型*设置为*输出*。
3	在同一画面中，组态 *Windows 对象* → *按钮*。在本实例中，使用*按钮 7* 对象。
4	为了更改设定值，在*事件* → *鼠标* → *按下左键*处创建一个 *C 动作*。该 *C 动作*调用*项目函数 IncDecValue*，并将所需的参数传送给它。每次用单击*按钮*时，它都要更改变量的值。在调用*项目函数*时，将限制值指定为传送参数。在*项目函数*中执行检查。
5	以同样的方式组态设定值的减量。在本实例中，使用*按钮 6* 对象。

按钮 7 的 C 动作

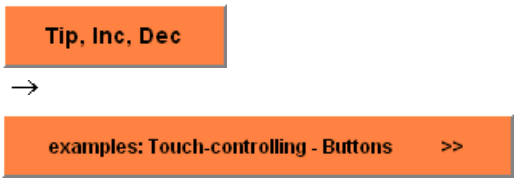
```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
DWORD value;

value=GetTagDWord("S32i_varia_but_04");

//IncDecValue(DWORD *value,DWORD low,DWORD high,DWORD step,DWORD a )
IncDecValue(&value,0,1400,100,1);
SetTagDWord("S32i_varia_but_04",value);
}
```

- 使用*内部函数 GetTagDWord* 来读出*内部变量*的当前值。
- 调用*项目函数 IncDecValue*，并传送参数(指向变量的指针、上限和下限、增量、方向)。
- 使用*内部函数 SetTagDWord* 来将更改后的值传送给*内部变量*。

常规应用的注意事项

不用做进一步更改就可以立即使用该*项目函数*。在用于调用*项目函数*的 *C 动作*中，可以根据自己的要求调整传送参数。

2.2.3 按击 - 按钮(实例 05)



可通过使用 选择如上所示的*按钮*在 *Project_TagHandling* 项目中访问有关该主题的解决方案。它们均在 *pictu_3_chapter_01a.pdl* 画面中组态。

任务定义

按击是通过使用鼠标得以执行的。
单击按钮可以激活设备(电机、阀门)。释放按钮时，激活将被取消。

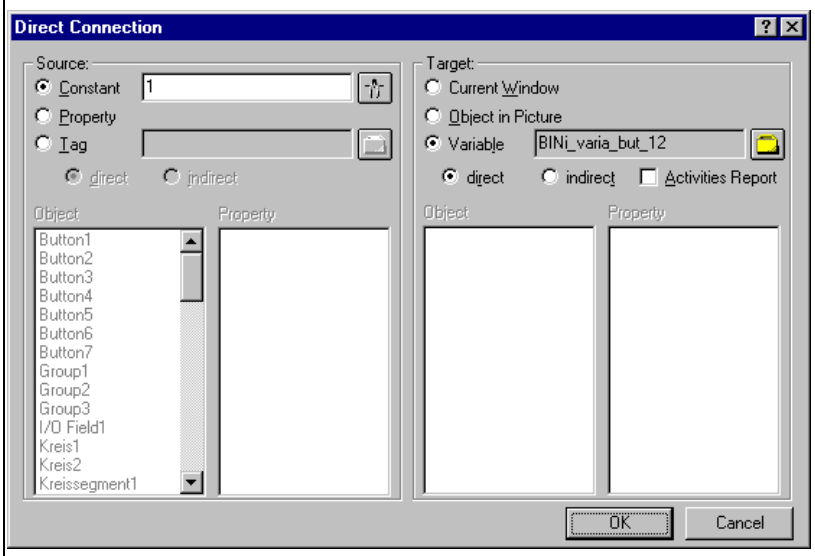
概念的实现

可通过 *Windows 对象* → *按钮*来执行由事件驱动的按钮。
通过一个*直接连接*和一个 *C 动作*可将此执行过程可视化。

注意：
通过*直接连接*来执行按钮将可在运行系统期间提供最佳性能。

图形编辑器中的实现 - 直接连接

步骤	过程：直接连接
1	在变量管理器中创建一个*二进制变量*类型的变量。在本实例中，*使用了 BINi_varia_but_12 变量*。
2	在某个画面中，组态 *Windows 对象* → *按钮*。在本实例中，使用了*按钮 2* 对象。

步骤	过程：直接连接
3	在*事件* → *鼠标* → *按下左键*下为*按钮 2*组态一个*直接连接*。将*源常数* → *1* 与*目标变量* → *BINi_varia_but_12* 连接。单击*确定*按钮即可应用这些设置。在*事件* → *鼠标* → *释放左键*下组态另一个*直接连接*，但这次是为*源常数* → *0*。 
4	通过 *BINi_varia_but_12* 变量对动画进行控制。

在下面将使用一个 *C 动作*对同一任务的执行过程进行说明。上面所概述的使用*直接连接*的执行过程是一种较佳和较为快捷的方式。

图形编辑器中的实现 - C 动作

步骤	过程：C 动作
1	在变量管理器中创建一个*二进制变量类型*的变量。在本实例中，*使用 BINi_varia_but_12 变量*。
2	在画面中组态 *Windows 对象* → *按钮*。在本实例中，使用*按钮 1* 对象。
3	在*事件* → *鼠标* → *按下左键*处，创建一个 *C 动作*，它将 *BINi_varia_but_12* 变量的值设置为 *1*。在*事件* → *鼠标* → *释放左键*处，创建另一个 *C 动作*，它将 *BINi_varia_but_12* 变量的值设置为 *0*。

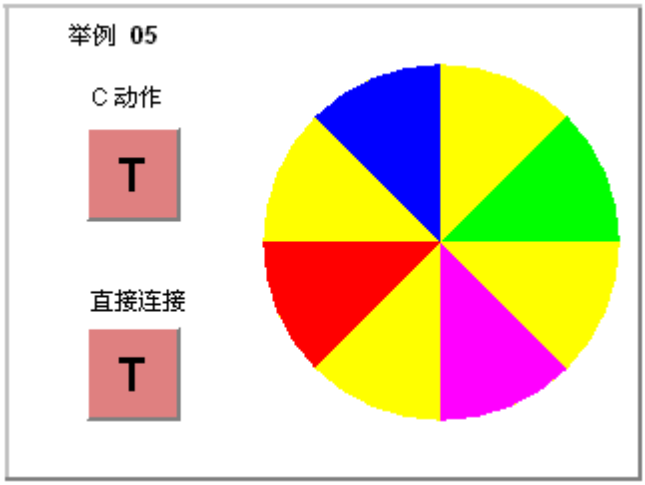
按钮 1 的 C 动作

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
SetTagWord("BINi_varia_but_12",1); //on
}
```

- 使用*内部函数 SetTagDWord* 将变量设置为 *1*。

动画实例

在本实例中，使用按钮将下面的彩色车轮制成动画。



- 彩色车轮由多个*标准对象* → *饼图分割*组成。
- 使用在*属性* → *几何结构* → *起始角度*和*属性* → *几何结构* → *结束角度*下的*动态对话框*使所有对象动态化。

  为了对值进行修改，需要一个动作以固定的时间间隙来修改旋转角度值。通过一个 *C 动作*为*饼图分割 4* 的*属性* → *颜色* → *线段颜色*进行该数值修改。将该动作的触发时间设置为 *250 毫秒*。在这种情况下，没有使*线段颜色*动态化。在该属性使用 *C 动作*的原因在于需要一个触发器用来对数值进行修改。还可以使用该对象的不同属性代替‘线段颜色’。
- 在*内部变量 S32i_vara_but_11* 中对当前的旋转角度进行了修改。

用于动画的 C 动作

```
#include "apdefap.h"
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyN

{
Static DWORD i = 0;

//if button pressed
if (GetTagBit("BINi_varia_but_12")) {
   i=i+10;              //increment of rotation
   if (i==360) (i=0);  //high limit
   SetTagDWord("S32i_varia_but_11",i);
   }//if
return(0x0); //black
}
```

- 将 *C 变量 i* 声明为 *static DWORD*，因为其数值在画面打开时必须保持为常数。
- 如果按下按钮(按钮启动)，则车轮将以 10 度的增量进行旋转，即变量值的增量为 10。
- 每当车轮完全旋转一周(360°)，就对变量 *i* 进行初始化。
- 对用于*内部变量*的旋转角度的新值进行传送。
- 使用 return 返回背景色的组态值。不应对其进行修改。

常规应用的注意事项

在对变量进行修改后，可使用具有*直接连接*的*按钮*。

2.2.4 按击 - 切换开关(实例 06)

可通过使用 选择如上所示的按钮在 *Project_TagHandling* 项目中访问有关该主题的解决方案。在 *pictu_3_chapter_01a.pdl* 画面中进行组态。

任务定义

按击是通过使用鼠标得以执行的。
切换开关的功能是通过按钮来实现的。
按下按钮将打开单元(电机、阀)，而一旦释放按钮，单元将保持打开状态。再次按动按钮，设备将关闭。

概念的实现

通过 Windows 对象 → 按钮实现事件驱动的切换开关。

注意：
通过*直接连接*所实现的切换开关可在运行系统期间提供最佳性能，但需要两个按钮。

图形编辑器中的实现 - 直接连接

步骤	过程：直接连接
1	在变量管理器中创建一个*二进制变量*类型的变量。在本实例中，*使用 BINi_varia_but_16 变量*。
2	在画面中组态两个 *Windows 对象* → *按钮*。在本实例中，使用*按钮 4* 对象来打开，使用*按钮 5* 对象来关闭。
3	在*事件* → *鼠标* → *按下左键*处为*按钮 4* 组态一个*直接连接*将*源常数* → *1* 连接到*目标变量* → *BINi_varia_but_16*。单击确定按钮即应用此设置。为*按钮 5* 组态一个如上所述的*直接连接*，但需使用*源常数* → *0*。
4	在*事件* → *鼠标* → *鼠标动作*上的*直接连接*仅使*按钮 3* 上的设定标签同步化，并且不需要用于切换开关功能。

图形编辑器中的实现 - C 动作

步骤	过程：C 动作
1	在变量管理器中创建一个*二进制变量*类型的变量。在本实例中，*使用 BINi_varia_but_16 变量*。
2	在画面中组态 *Windows 对象* → *按钮*。在本实例中，使用*按钮 3* 对象。
3	在*事件* → *鼠标* → *按下左键*处，创建一个 *C 动作*，它将对 *BINi_varia_but_16* 变量状态求反。

用于切换开关的 C 动作

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
BOOL state;

//flip tag
state = !GetTagBit("BINi_varia_but_16"),
SetTagBit("BINi_varia_but_16",(SHORT)state);
}
```

- *state* 变量的声明。
- 通过*内部函数* *GetTagBit*，可读出*内部变量*的值，对该值求反，然后通过 *SetTagBit* 函数将值返回。

常规应用的注意事项

可在修改变量后使用具有 *C 动作*的按钮。没有如下所示的 *C 变量*也可完成对*内部变量*的求反：

```
SetTagDWord("BINi_varia_but_16",
(SHORT)!GetTagBit("BINi_varia_but_16"));
```

2.2.5 递增和递减(实例 01)

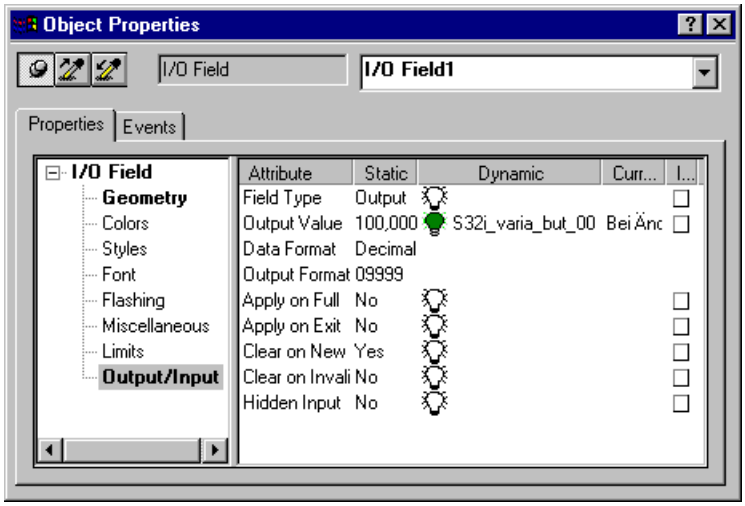
可通过使用 选择如上所示的按钮在 *Project_TagHandling* 项目中访问有关该主题的解决方案。在 *pictu_3_chapter_01a.pdl* 画面中进行组态。

任务定义

变量值将被改变。数值的这种改变将受固定限制值的约束。使用鼠标改变值。按下按钮改变变量值。只有按下按钮时变量的值才被改变。释放按钮时设置的值必须得以保留。

概念的实现

为了执行事件驱动按钮，使用 Windows 对象 → 按钮。
当使用按下按钮时，可根据增量设置增加*内部变量*的值，当使用R 按下按钮时，可根据增量设置减少变量值。只要按住按钮，变量值就一直改变。增量是预先指定的，运行期间不可改变。
为了显示值的改变，使用*智能对象* → *I/O 域*。*I/O 域*的输出值与*内部变量*相连。

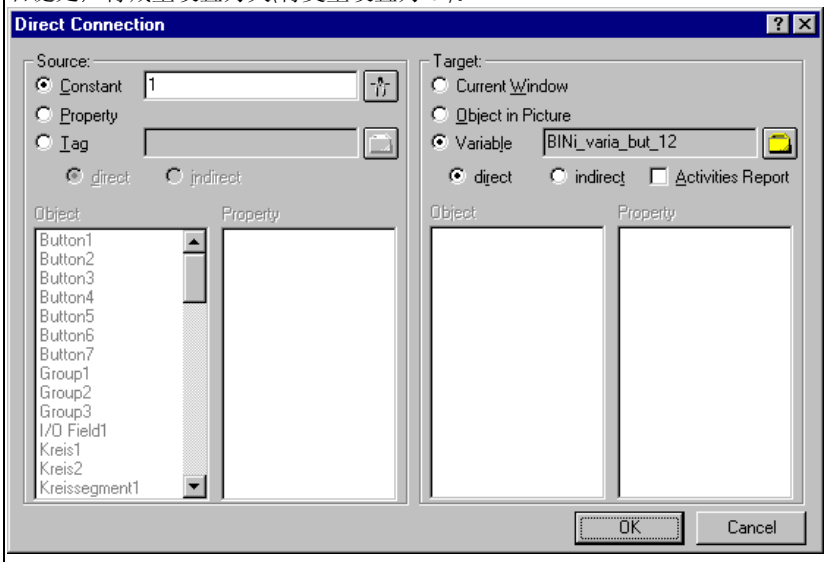


改变数值

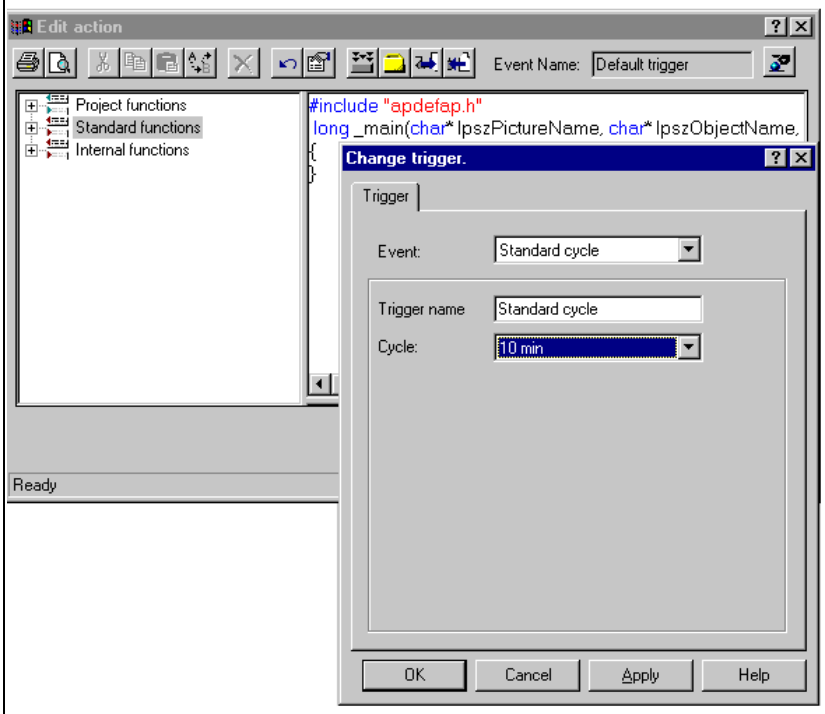
为了改变数值，需要一个动作在固定的时间间隙内改变*内部变量*的值。在 *I/O 域*的*属性* → *几何结构* → *位置 X* 处，用 C 动作直接对值进行修改。将动作的触发时间设置为 *250 毫秒*。我们没有使 *I/O 域*的位置动态化。在该属性使用 *C 动作*的原因是希望直接在此对象上实现值的修改。

在此实例项目中，通过使用*全局动作*已经解决了该问题。

在 WinCC 项目中的实现

步骤	过程：递增、递减
1	在变量管理器中创建变量。在本实例中，*使用 S32i_varia_but_00 及 S08i_varia_but_01 变量*。
2	在画面中对*智能对象* → *I/O 域*进行组态。在本实例中，使用 *I/O 域 1* 对象。在创建 *I/O 域*期间，可在组态对话框中设置 *S32i_varia_but_00* 变量。将*更新域*中的缺省值 2 秒修改为*一旦改变*，并将*域类型*设置为输出。
3	在同一画面中，组态 *Windows 对象* → *按钮*。在本实例中，使用了*按钮 3* 对象。
4	为了通过鼠标的单击对设定值进行修改，可*在该按钮处创建多个直接连接*。每次通过或R 按下按钮时，这些*直接连接*都将修改 *S08i_varia_but_01* 变量的值。 在*事件* → *鼠标* → *按下左键*处，将增量设置为开(将变量设置为 *1*)。在*事件* → *鼠标* → *释放左键*处，将增量设置为关(将变量设置为 *0*)。在*事件* → *鼠标* → *按下右键*处，将减量设置为开(将变量设置为 *2*)，在*事件* → *鼠标* → *释放右键*处，将减量设置为关(将变量设置为 *0*)。 
5	可在对象 *I/O 域 1* 的*属性* → *几何结构* → *位置 X* 下的 *C 动作*中对 *S32i_varia_but_00* 变量的值进行修改。

步骤	过程：递增、递减
6	用来调用 *C 动作*的触发器将被修改成 *250 毫秒*。



用于对值进行修改的 I/O 域上的 C 动作

```
#include "apdefap.h"
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyl

{
DWORD value;
SHORT count;

count = GetTagWord("S08i_varia_but_01");  //inc or dec
if ((count==1) || (count==2)){
   //current value
   value = GetTagDWord("S32i_varia_but_00");

   if (count==1){ //inc
       value++;
       if (value>1400) (value=1400);       //high limit
       SetTagDWord("S32i_varia_but_00",value);
       }//inc
   if (count==2){ //dec
       value--;
       if (value<0) (value=0);             //low limit
       SetTagDWord("S32i_varia_but_00",value);
       }//dec
   }//if count
return(81); //x-pos
}
```

- 声明 *C 变量 value* 和 *count*。
- 判断*按钮*是否按下。如果没有按下*按钮*，则 *C 动作*将结束(以避免不必要的系统负担)。
- 如果按下了*按钮*，则脚本将查询数值是递增还是递减。根据判断结果改变变量值。
- 值改变以后，对限制值进行检查。
- 用 *return* 命令返回为位置 X 组态的值。不应对其进行修改。

常规应用的注意事项

在修改变量之后，可使用具有*直接连接*的按钮，并与 *I/O 域*处的 *C 动作*协同工作。在 *C 动作*中，必须修改限制值和变量。

2.2.6 通过全局脚本递增和递减(实例 02)

任务定义

将对变量的值进行修改。对值的这种修改要受固定限制值的约束。使用鼠标可完成对值的修改。
按下按钮即可对变量值进行修改。只有在按下按钮时，才能对值进行修改。在按钮释放后，所设置的值将得以保留。

概念的实现

为了执行事件驱动按钮，可使用 Windows 对象 → 按钮。
通过一个*全局动作*可完成执行过程。
当使用按下按钮时，可根据增量设置增加*内部变量*的值，当使用R 按下按纽时，可根据增量设置减少变量的值。只要按住按钮，变量值就一直改变。增量是预先指定的，运行期间不可改变。
为了显示值的改变，使用*智能对象* → *I/O 域*。*I/O 域*的输出值与*内部变量*相连。

改变数值

为了改变数值，需要在固定的时间间隙内改变*内部变量*的值。通过一个*全局动作*完成值的改变。
在启动 WinCC 运行系统时，激活该动作，并按设置的周期进行处理。当按下按钮时，仅对实际程序组件进行处理，动作就是按这种方式编程的。
此动作的一个不寻常的特性是它使用了外部 C 变量。在整个 WinCC 运行系统中，对外部 C 变量进行识别，但是必须在函数标题外部对其进行声明。由于在 WinCC 中，这种情况只可能存在于项目函数中，所以必须创建一个独立的项目函数，以便对这些变量进行声明。该项目函数必须在项目启动时执行一次，之后就不再需要它了。

创建项目函数

步骤	过程：创建项目函数
1	在 *WinCC 资源管理器*中，启动*全局脚本*编辑器。
2	通过*文件* → *新建项目函数*菜单来创建新函数。
3	分配 *InitAction* 函数名称，并通过选择*文件* → *另存为* → *InitAction.fct* 来保存函数。
4	编写和编译函数。

项目函数 InitAction

```
//declaration for counter.pas
extern char tagname[30] = " ";
extern SHORT count = 0;
extern DWORD low = 0;
extern DWORD high = 0;
extern DWORD step = 0;

void InitAction()
{
//function is used to generate external tags
}
```

- 外部 *C 变量*的声明。
- 该函数必须在项目启动时执行一次，之后就不再需要它了。建议在*事件* → *其它* → *打开画面*处的启动画面中完成执行过程。

全局动作的创建

步骤	过程：全局动作的创建
1	在 *WinCC 资源管理器*中，启动*全局脚本*编辑器。
2	通过*文件* → *新建动作*菜单来创建新动作。
3	通过选择*文件* → *另存为* → *counter.pas* 来保存文件。
4	编写和编译该动作。
5	设置触发。通过工具栏上的*按钮* 可完成该操作。在*描述*对话窗口中，选择*触发栏*。添加*定时器* → *标准周期* → *250 毫秒*。

全局动作 counter.pas

```
#include "apdefap.h"

int gscAction( void )
{
extern char tagname[30];
extern SHORT count;
extern DWORD low;
extern DWORD high;
extern DWORD step;

DWORD value;

if ((count==1)||(count==2)) {
        //get current value
        value = GetTagDWord(tagname);
        if (count==1){                    //inc
                value = value+step;
                if (value>high) (value=high); //high limit
                }//if
        if (count==2){                        //dec
                value = value-step;
                if (value<low) (value=low);  //low limit
                }//if
        SetTagDWord(tagname,value);
        }//if
return(0);
}
```

- 外部 *C 变量*的声明。
- 判断*按钮*是否按下。如果没有按下*按钮*，则 *C 动作*将结束(以避免不必要的系统负担)。
- 如果按下了*按钮*，则脚本将查询数值是递增还是递减。根据判断的结果，对 *C 变量值*的值进行修改。
- 值改变以后，对限制值进行检查。
- 使用*内部函数 SetTagDWord* 将新的值分配给将要处理的变量。

图形编辑器中的实现

步骤	过程：图形编辑器中的实现
1	在变量管理器中创建变量。在本实例中，*使用 S32i_varia_but_04 变量*。
2	在画面中组态*智能对象* → *I/O 域*。在本实例中，使用对象 *I/O 域 2*。在创建 *I/O 域*期间，可在组态对话框中设置 *S32i_varia_but_04* 变量。将*更新域*中的缺省值 2 秒修改为*一旦改变*，并将*域类型*设置为*输出*。
3	在同一画面中，组态 *Windows 对象* → *按钮*。在本实例中，使用*按钮 8* 对象。
4	为了通过鼠标单击对设定值进行修改，可在该*按钮*上创建几个 *C 动作*。在*事件* → *鼠标* → *按下左键*处，将执行过程设置为开，在*事件* → *鼠标* → *释放左键*处，将执行过程设置为关。在*事件* → *鼠标* → *按下右键*处，将减量过程设置为开，在*事件* → *鼠标* → *释放右键*处，将减量过程设置为关。这些 *C 动作*将为*全局动作 counter.pas* 提供合适的参数。每次通过 或 R 单击按钮时，都将发生这种情况。
5	在*全局动作 counter.pas* 中对 *S32i_varia_but_04* 变量的值进行修改。

用于增量开的按钮 8 上的 C 动作

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
//inc on
extern char tagname[30];
extern SHORT count;
extern DWORD low;
extern DWORD high;
extern DWORD step;

strcpy(tagname,"S32i_varia_but_04");
count = 1;
low = 0;
high =1400;
step = 1;
}
```

用于增量关的按钮 8 上的 C 动作

```
#include "apdefap.h"
void OnLButtonUp(char* lpszPictureName, char* lpszObjectName, char* lpszPrc

{
//inc off
extern SHORT count;
count=0;
}
```

- *C 动作*中对外部 *C 变量*的声明。这些变量由 *InitAction* 项目函数产生。
- 为这些变量提供相关值。这类似于将参数传送给*项目函数*。*count* 变量的内容主要用于对*全局动作*中的程序进行处理。
- 关闭增量过程时，不需要设置所有变量。

常规应用的注意事项

在进行常规应用之前，必须进行下列修改：

- 在 *C 动作*中，对变量进行修改，并相应改编限制值和增量。
- 如果该按钮被传送给另一个项目，择必须用该按钮将*项目函数 InitAction* 和*全局动作 counter.pas* 一起传送。

2.2.7 本主题的其余实例

实例 03

该实例与样本*实例 01* 具有类似的功能。它们之间的根本差别在于运行期间可改变增量。
另一个差别是设置增量时可动态改变增量。如果增量大于 20，则值以 10 为步长改变；如果增量小于 20，则值以 1 为步长改变。

实例 04

该实例的功能是样本*实例 01* 和*实例 02* 相结合的结果。借助于*全局动作 counter.pas* 可对值进行修改。

实例 07

该实例与样本*实例 05* 具有类似的功能。这里的差别在于动画模式的不同。

2.3 通过 Windows 对象对变量值进行修改

在运行系统中，通过使用选择如上所示的*按钮*来访问 *Project_TagHandling* 项目中与该主题有关的解决方案。这些实例在 *varia_3_chapter_02.pdl* 画面中组态。

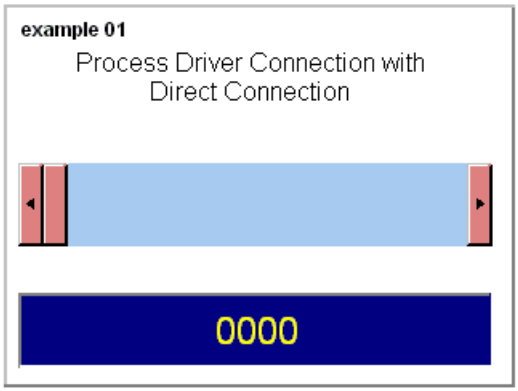
2.3.1 通过带有直接连接的滚动条进行输入(实例 01)

任务定义

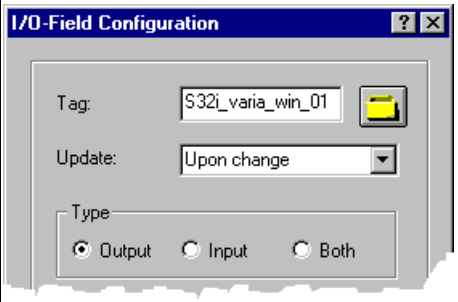
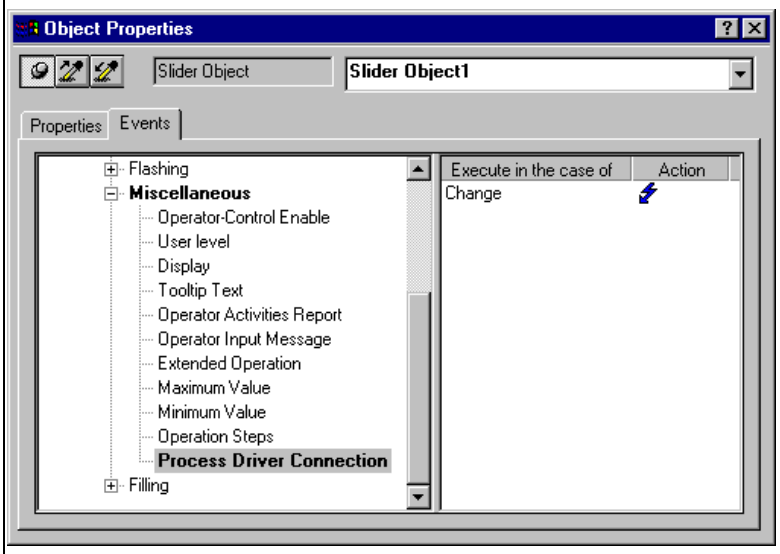
通过滚动条完成对设定值的修改。
数值的改变受到固定限制值的约束。

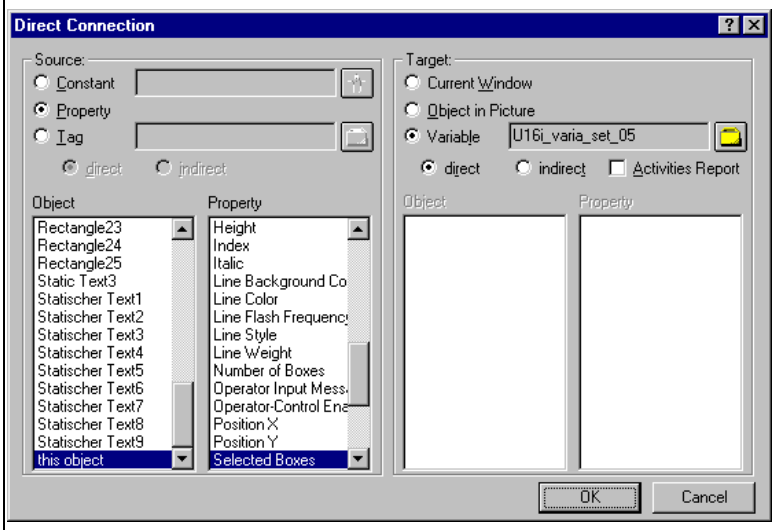
概念的实现

为了实现对设定值的修改，将使用 *Windows 对象* → *滚动条对象*。通过*直接连接*，当滚动条的位置改变时，*内部变量*的值也随之改变。
在智能对象 → I/O 域中将显示设定值的变化。



图形编辑器中的实现

步骤	过程：图形编辑器中的实现
1	在变量管理器中创建变量。在本实例中，*使用 S32i_varia_but_01 变量*。
2	在画面中组态*智能对象* → *I/O 域*。在本实例中，使用 *I/O 域 1* 对象。在创建 *I/O 域*期间，可在组态对话框中设置 *S32i_varia_but_01* 变量。在*更新域*中将缺省值 2 秒修改为*一旦改变*，并将*域类型*设置为*输出*。 
3	在同一画面中，组态 *Windows 对象* → *滚动条对象*。在本实例中，使用*滚动条对象 1*。在*事件* → *其它* → *过程驱动程序连接处*，创建一个*直接连接*。 

步骤	过程：图形编辑器中的实现
4	在*直接连接*对话框中，将*源此对象* → *过程驱动程序连接*与*目标变量* → *S32_varia_win_01* 相连接。单击*确定*按钮即可应用这些设置。 

常规应用的注意事项

在进行常规应用之前，必须进行下列修改：

- 修改*直接连接*变量。
- 可通过*属性* → *其它* → *最大值*和*最小值*来修改*滚动条对象*的取值范围。也可在滚动条的*组态对话框*中进行修改。

2.3.2 通过滚动条和变量连接进行输入(实例 03)

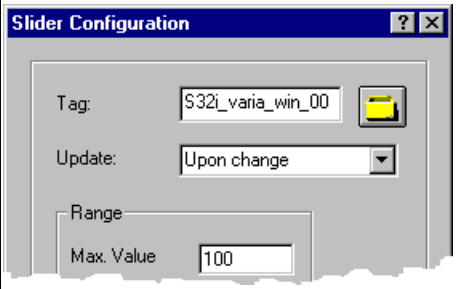
任务定义

通过滚动条对设定值进行修改。
数值的改变受固定限制值的约束。

概念的实现

为了实现对设定值的修改，将使用 *Windows 对象* → *滚动条对象*。通过*变量连接*，当滚动条的位置改变时，*内部变量*的值也随之改变。只有释放滚动条时才可写入变量。
在*智能对象* → *I/O 域*中将显示设定值的变化。

在 WinCC 项目中的实现

步骤	过程：通过滚动条 - 变量连接对设定值进行修改
1	在变量管理器中创建变量。在本实例中，*使用 S32i_varia_but_00 变量*。
2	在画面中组态*智能对象* → *I/O 域*。在本实例中，使用 *I/O 域 3* 对象。在创建 *I/O 域*期间，可在组态对话框中设置 *S32i_varia_but_00* 变量。将*更新域*中的缺省值 2 秒修改为*一旦改变*，并将*域类型*设置为*输出*。
3	在同一画面中，组态一个 *Windows 对象* → *滚动条对象*。在本实例中，使用了*滚动条对象 2*。在创建*滚动条对象*期间，可在*组态对话框*中设置 *S32i_varia_but_00* 变量。将*更新*缺省值由 2 秒修改为*一旦改变*。 

常规应用的注意事项

在进行常规应用之前，必须进行下列修改：

- 在*变量连接*中修改变量。
- 可通过*属性* → *其它* → *最大值*和*最小值*对*滚动条对象*的取值范围进行修改。也可在滚动条的*组态对话框*中进行修改。

2.3.3 通过选项组(选项钮)进行输入(实例 02)

任务定义

从列表中选择指定的、固定的数值可对设定值进行修改。

概念的实现

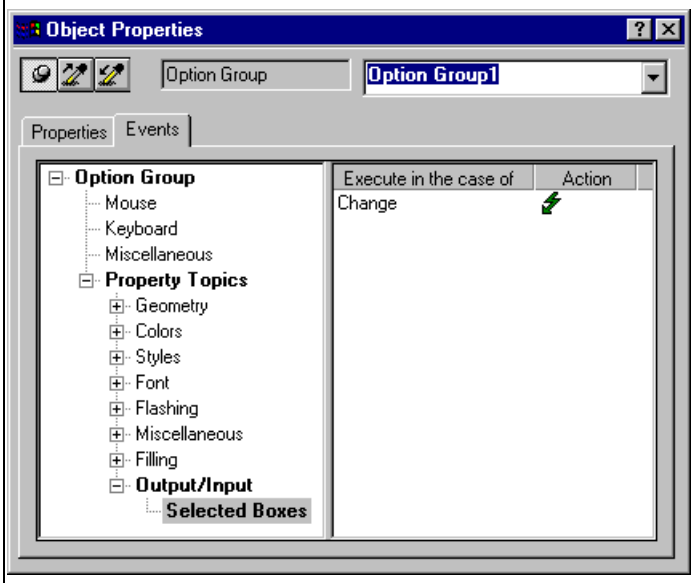
为了实现对设定值的修改，将使用 *Windows 对象* → *选项组*。

在通过选择了某个指定的设定值时，*内部变量*中的值将改变。设定值列表是指定的，运行期间不可更改。

通过 *智能对象* → *I/O 域*将显示设定值的变化。*I/O 域*的输出值与*内部变量*连接。改变设定值可通过一个 *C 动作*来完成。

在 WinCC 项目中的实现

步骤	过程：通过选项组对设定值进行修改
1	在变量管理器中创建变量。在本实例中，*使用了 S32i_varia_but_02 变量*。
2	在画面中组态 *智能对象* → *I/O 域*。在本实例中，使用了 *I/O 域 2* 对象。在创建 *I/O 域*期间，可在*组态对话框*中设置 *S32i_varia_but_02* 变量。将*更新域*中的缺省值 2 秒修改为*一旦改变*，并将域类型设置为*输出*。
3	在同一画面中，组态一个 *Windows 对象* → *选项组*。在实例中是*选项组 1*。在*属性* → *几何结构* → *方框数目*处，将缺省值 3 修改为 *4*。
4	通过*属性* → *字体* → *下标* → *1* 可选择下标 1。在*属性* → *字体* → *文本* → *0* 处为所选下标输入相应的文本。使用同样的方法为其余下标输入组态数值。

步骤	过程：通过选项组对设定值进行修改
5	在事件 → 属性主题 → 输出/输入 → 所选方框处，创建一个 C 动作，它可根据所选的选项钮，将 S32i_varia_win_02 变量设置给某个值。 

选项组的 C 动作

```
Void OnPropertyChanged(char* lpszPictureName, char* lpszObjectNane, char* lpszI

{
//set tag according to selected box
switch(value){
  case 1 : SetTagDWord("S32i_varia_win_02",0);
  break;
  case 2 : SetTagDWord("S32i_varia_win_02",50);
  break;
  case 4 : SetTagDWord("S32i_varia_win_02",100);
  break;
  case 8 : SetTagDWord("S32i_varia_win_02",150);
  break;
  }//switch
}
```

- 根据输入状态，为 *S32i_varia_win_02* 变量赋值。输入状态存储在预定义的 *value* 变量中。

常规应用的注意事项

在对*选项组*进行常规应用前必须进行下列修改：

- 在*事件* → *属性主题* → *输出/输入* → *所选方框*下的 *C 动作*中，可对变量进行修改。

2.3.4 通过复选框进行输入(实例 04)

任务定义

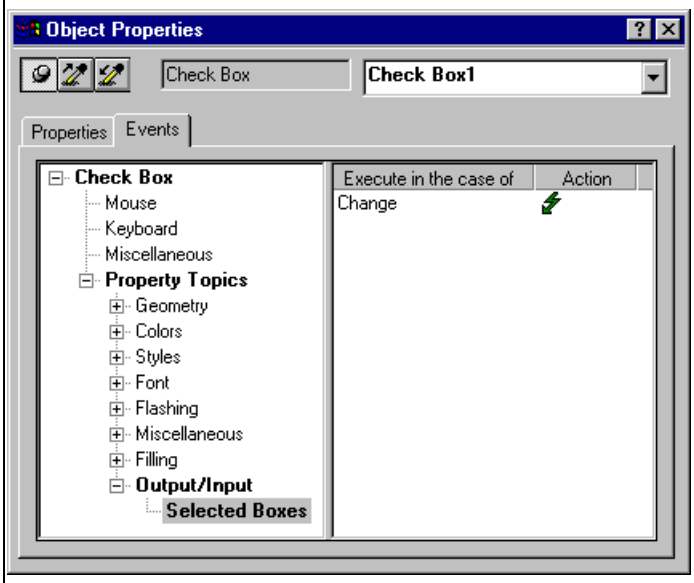
通过复选框，可显示或隐藏各种不同的对象。

概念的实现

可使用一个 *Windows 对象* → *复选框*来实现，它将对变量的各个位进行设置。许多*标准对象* → *多边形*就是根据这些位来显示或隐藏的。为了显示复选框的二进制输出值，使用了一个*智能对象* → *I/O 域*。

在 WinCC 项目中的实现

步骤	过程：通过复选框进行输入
1	在变量管理器中创建一个*有符号的 32 位数*类型的变量。在本实例中，*使用了 S32i_varia_but_03 变量*。
2	*组态几个标准对象* → *多边形*。在本实例中使用了*多边形 1 至多边形 7*。根据*复选框*的选择状态来显示或隐藏这些对象。
3	在同一画面中，组态一个 *Windows 对象* → *复选框*。在本实例中是复选框 1。在*属性* → *几何结构* → *方框数目*中，将缺省值 3 修改为 *7*。
4	通过*属性* → *字体* → *下标* → *1* 可选择下标 1。在*属性* → *字体* → *文本处*为所选下标输入相应的文本。该文本是希望通过选择该复选框进行控制的对象名。使用同样的方法为其余下标输入组态数值。

步骤	过程：通过复选框进行输入
5	在*事件* → *属性主题* → *输出/输入* → *所选方框*处，可创建一个 *C 动作*，它将*复选框 1* 的二进制状态分配给 *S32i_varia_win_03* 变量，并控制各个*多边形*对象的显示。 
6	组态一个*智能对象* → *I/O 域*。在本实例中，使用了 *I/O 域 4* 对象。在*组态对话框*中，设置 *S32i_varia_win_03* 变量。将*更新*缺省值由 2 秒修改为*一旦改变*。在*属性* → *输出/输入*处，将*数据格式*修改为*二进制*，并将输出格式修改为 *01111111*。

复选框中的 C 动作

```
#include "apdefap.h"
void OnPropertyChanged(char* lpszPictureName, char* lpszObjectName, char* l

{
SetTagDWord("S32i_varia_win_03",value);
//first box selected
if (value&1) SetVisible(lpszPictureName,"Polygon1",1);
else SetVisible(lpszPictureName,"Polygon1",0);
//second box selected
if (value&2) SetVisible(lpszPictureName,"Polygon2",1);
else SetVisible(lpszPictureName,"Polygon2",0);
//third box seleted
if (value&4) SetVisible(lpszPictureName,"Polygon3",1);
else SetVisible(lpszPictureName,"Polygon3",0);
//fourth box selected
if (value&8) SetVisible(lpszPictureName,"Polygon4",1);
else SetVisible(lpszPictureName,"Polygon4",0);
//fifth box selected
if (value&16) SetVisible(lpszPictureName,"Polygon5",1);
else SetVisible(lpszPictureName,"Polygon5",0);
//sixth box selected
if (value&32) SetVisible(lpszPictureName,"Polygon6",1);
else SetVisible(lpszPictureName,"Polygon6",0);
//seventh box selected
if (value&64) SetVisible(lpszPictureName,"Polygon7",1);
else SetVisible(lpszPictureName,"Polygon7",0);
}
```

- 将 *S32i_varia_win_03* 变量设置为*复选框*的新的输入状态。
- 按照输入状态控制对象的可见性。将输入状态存储在预定义的 value 变量中。若要读出各个位，必须对相关位执行位屏蔽。

注意：
在*添加动态*一章*实例 4* 的 *Project_CreatePicture* 项目中对类似的实例进行了说明。在那个实例中，通过*动态对话框*对每一个单个对象的可见性进行了查询。

常规应用的注意事项

在对*复选框*进行常规应用前必须进行下列修改：

- 在*事件* → *属性主题* → *输出/输入* → *所选方框*的 *C 动作*中对变量及对象名称进行修改。

2.4 对字中的位进行处理

可通过使用选择如上所示的按钮来访问 Project_TagHandling 项目中与该主题有关的解决方案。这些实例均在 varia_3_chapter_03.pdl 和 varia_3_chapter_03a.pdl 画面中组态。

定义

术语**位处理**是指在一个字中改变位的状态。

2.4.1 直接通过复选框和直接连接进行置位(实例 06)

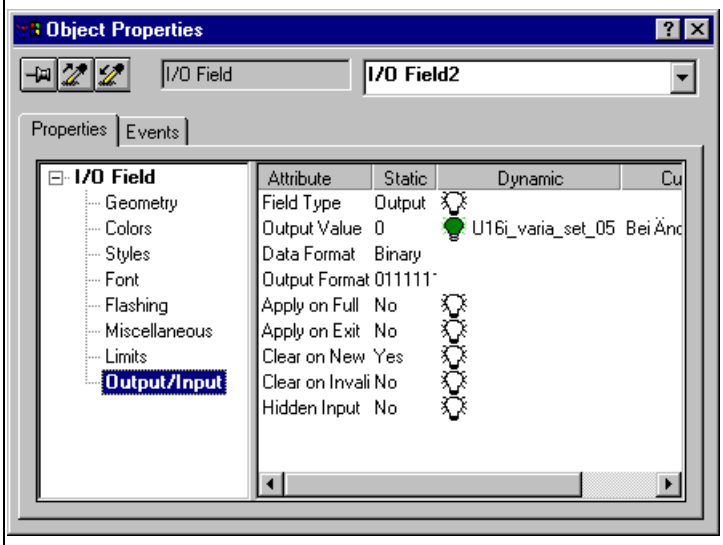
任务定义

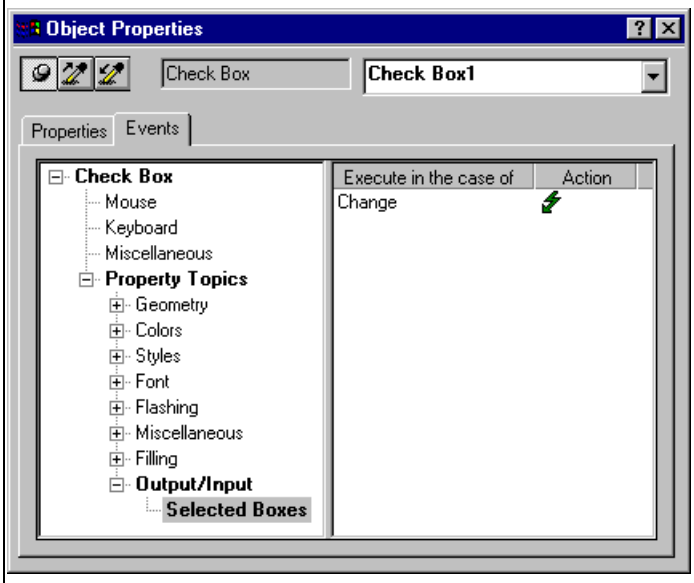
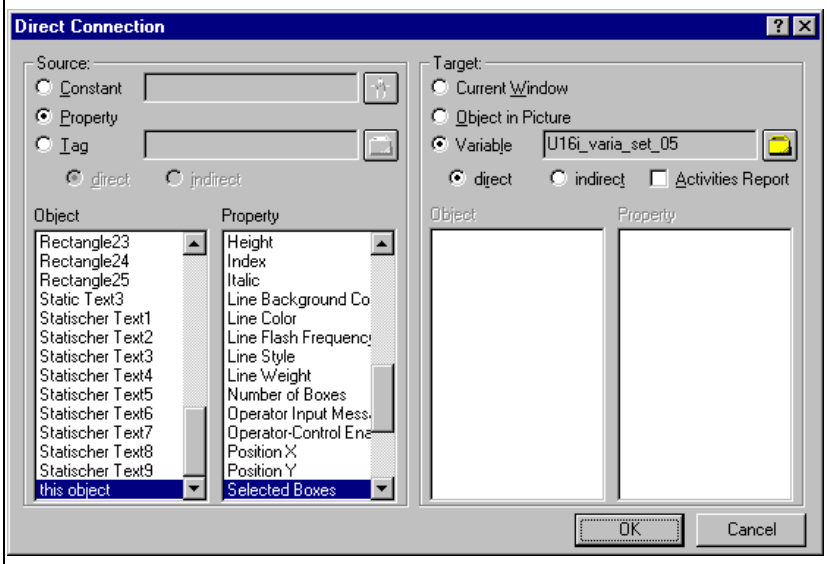
当数据字中的位被选中时，其状态要被改变。希望能够选择若干个位。

概念的实现

为了实现更改位的状态，将使用 *Windows 对象* → *复选框*。如果用选择了其中一个*复选框*域，则在*内部变量*中用*直接连接*更改分配给它的位。
为了显示位模式，使用智能对象 → I/O 域。I/O 域的输出值与内部变量相连。

在 WinCC 项目中的实现

步骤	过程：直接通过复选框和直接连接进行置位
1	在变量管理器中创建一个*无符号的 16 位数*类型的变量。在本实例中，*使用 U16i_varia_set_05 变量*。
2	在画面中，组态*智能对象* → *I/O 域*。在本实例中，使用了 *I/O 域 2* 对象。创建 *I/O 域*期间，在*组态对话框*中设置变量 *U16i_varia_set_05*。将*更新*域中的缺省值 2s 改为*一旦改变*，并将*域类型*设置为*输出*。通过*属性* → *输出/输入*，将*数据格式*改为*二进制*并将*输出格式*改为 *0111111111111111*。 
3	在同一画面中，组态一个 *Windows 对象* → *复选框*。在本实例中，使用了*复选框 1* 对象。在*属性* → *几何结构* → *方框数目*处，将缺省值 3 改为 *16*。
4	通过*属性* → *字体* → *下标* → *1* 来选择下标 1。在*属性* → *字体* → *文本* → *位 0* 处为所选下标输入相应的文本。用同样的方法为其余的下标条目组态文本。

步骤	过程：直接通过复选框和直接连接进行置位
5	*在事件 → 属性主题 → 所选方框*处，用*直接连接*使该事件动态化。 
6	在*直接连接*对话框中，将*源属性 → 本对象 → 所选方框*与*目标变量 → U16i_varia_set_05*相连接。通过单击*确定*按钮即可应用这些设置。 

步骤	过程：直接通过复选框和直接连接进行置位
7	组态两个 *Windows 对象* → *按钮*。在本实例中，使用了对象*按钮 2* 和*按钮 3*。它们将用来对所有的位进行置位和复位。
8	对于*按钮 2*，在*事件* → *鼠标* → *鼠标动作*处创建一个*直接连接*。将*源常数* → *65535* 与*目标画面中的对象* → *复选框 1* → *所选方框*相连接。通过单击*确定*按钮即可应用这些设置。所选择的常数相当于二进制数 1111111111111111。对于*按钮 3*，按同样的方法创建一个*直接连接*，但要使用*源常数* → *0*。

常规应用的注意事项

在常规应用之前，必须进行下列修改：

- 必须在直接连接中修改变量。

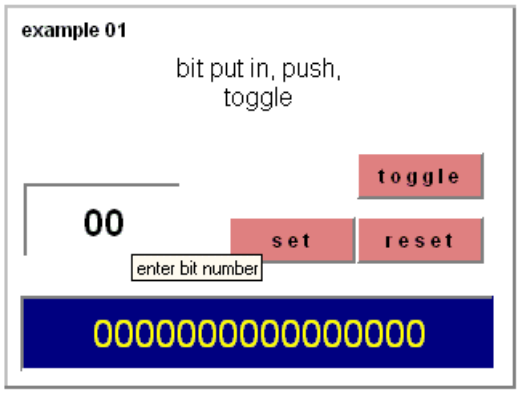
2.4.2 选择一个位并更改其状态(实例 01)

任务定义

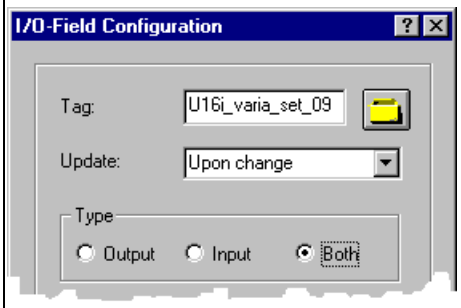
通过输入位编号并按下*按钮*来更改数据字中相应位的状态。它要从 0 转换为 1 或者反过来。

概念的实现

为了实现更改位的状态，使用一个 Windows 对象 → 按钮。
为了输入位编号并显示位模式，使用了一个智能对象 → I/O 域。当输入位编号并用按下按钮时，更改内部变量中所选择的位。这种更改用一个 *C 动作*来实现。



在 WinCC 项目中的实现

步骤	过程：更改数据字中的位
1	在变量管理器中创建两个*无符号的 16 位数*类型的变量。在本实例中，*使用了变量 U16i_varia_set_08* 与 *U16i_varia_set_09*。
2	在画面中，组态*智能对象* → *I/O 域*。在本实例中，使用了 *I/O 域 2* 对象。在其组态期间，将 *I/O 域*连接到 *U16i_varia_set_09* 变量上。将更新的缺省值由 2 s 改为*一旦改变*。将位编号输入此输入域中。 
3	为了显示位的状态，组态第二个 *I/O 域*。在本实例中，使用了 *I/O 域 1* 对象。在其组态期间，将 *I/O 域*连接到 *U16i_varia_set_08* 变量上。将更新的缺省值由 2 s 改为*一旦改变*。将域类型改为*输出*。通过*属性* → *输出/输入*，将*数据格式*改为*二进制*并将输出格式改为 *0111111111111111*。
4	在同一画面中，组态三个 *Windows 对象* → *按钮*。在本实例中，使用了对象*按钮 1*、*按钮 2* 和*按钮 3*。
5	对于*按钮 1*，在*事件* → *鼠标* → *按下左键*处创建一个 *C 动作*。该 *C 动作*在*内部变量*中对从 *I/O 域*内选择的位进行置位。采用同样的方法，为其它按钮创建另外的 *C 动作*，以便对位进行复位和切换。

置位按钮的 C 动作

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
WORD word,pos;

//get word and bit position
pos=GetTagWord("U16i_varia_set_09");
word=GetTagWord("U16i_varia_set_08");

word = (WORD)(word|1<<pos);

SetTagWord("U16i_varia_set_08",word);
}
```

- 声明 *C 变量*
- 使用*内部函数 GetTagWord* 读出所输入位的位置以及变量的当前值
- 位的移位功能
- 将新的值赋给*内部变量*

复位按钮的 C 动作

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
WORD word,pos;

//get word and bit position
pos=GetTagWord("U16i_varia_set_09");
word=GetTagWord("U16i_varia_set_08");

word=(WORD)(word&~(1<<pos));

SetTagWord("U16i_varia_set_08",word);
}
```

- 声明 *C 变量*
- 使用*内部函数 GetTagWord* 读出所输入位的位置以及变量的当前值
- 位的移位功能
- 将新的值赋给*内部变量*

切换按钮的 C 动作

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
WORD word,pos;

//get word and bit position
pos=GetTagWord("U16i_varia_set_09");
word=GetTagWord("U16i_varia_set_08");

word = (WORD)(word^1<<pos);

SetTagWord("U16i_varia_set_08",word);
}
```

- 声明 *C 变量*
- 使用*内部函数 GetTagWord* 读出所输入位的位置以及变量的当前值
- 位的移位功能
- 将新的值赋给*内部变量*

2.4.3 本主题的其余实例

实例 02

该实例与样本*实例 01* 具有类似的功能。主要差别在于选择要切换的位所用的方法。在本实例中，通过选择代表该位的对象来切换位。从对象名可以读出每一个对象所代表的位。

实例 04

该实例与样本*实例 02* 具有类似的功能。差异在于用选中位之后，位立即进行切换。此处对象也通过对象名分配给位。

实例 05

该实例与样本*实例 06* 具有类似的功能。此处差异在于使用了一个选项组(选项钮)。应用这种对象类型意味着在每个数据字中只能设置一个位。

2.5 变量的间接寻址

间接寻址

访问 Project_TagHandling 项目中与该主题相关的解答，可通过使用，选择上面显示的按钮来完成。这些实例均组态在 varia_3_chapter_04.pdl 画面中。

2.5.1 通过直接连接进行间接寻址(实例 01)

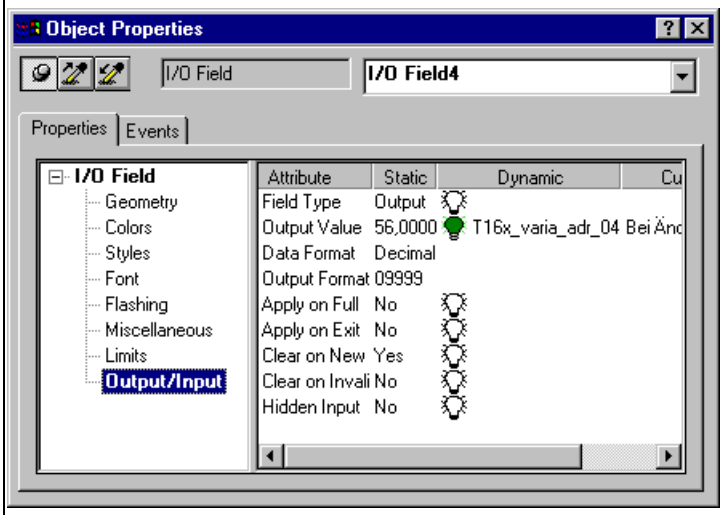
任务定义

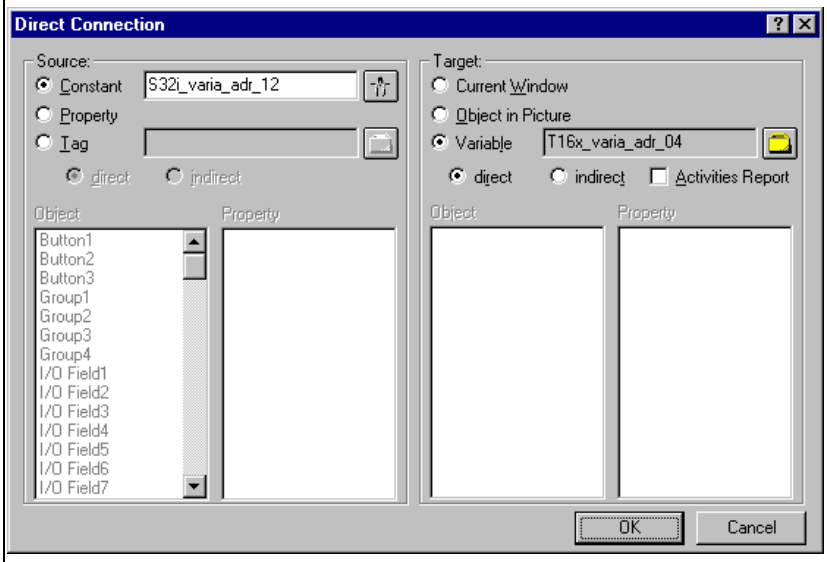
在 *I/O 域*中，各种过程值都将显示。相应的值将通过*按钮*来选择。

概念的实现

为了实现对相应过程值的选择，将使用一个 *Windows 对象* → *按钮*。
为了显示过程值，将使用一个*智能对象* → *I/O 域*和 WinCC 中的间接寻址选项。三个附加的*智能对象* → *I/O 域*创建后，将允许直接输入过程值。

在 WinC

步骤	过程：通过直接连接进行间接寻址
1	在变量管理器中创建三个*有符号的 32 位数*类型的变量。在本实例中，使用了 *S32i_varia_adr_12*、*S32i_varia_adr_13* 和 *S32i_varia_adr_14* 变量。这些变量包含将要显示的过程值。
2	在变量管理器中创建了一个具有*文本变量 16 位字符集*类型的变量。在本实例中，使用了 *T16x_varia_adr_04* 变量。该变量将用作地址变量。
3	在画面中，组态*智能对象* → *I/O 域*。在本实例中，使用了 *I/O 域 4* 对象。在创建 *I/O 域*期间，可在组态对话框中设置 *T16x_varia_adr_04* 变量。将*更新*域中的缺省值 2s 修改为*一旦改变*，并将*域类型*设置为输出。在*属性* → *输出/输入* → *输出值*处，可激活*间接*列中的复选框。 

步骤	过程：通过直接连接进行间接寻址
4	在同一画面中，组态了 3 个附加的 *I/O 域*。在本实例中，使用了对象 *I/O 域 1* 到 *I/O 域 3*。在创建 *I/O 域 1* 期间，在*组态对话框*中设置了 *S32i_varia_adr_12* 变量和触发器*一旦改变*。用同样的方法，组态 *I/O 域 2* 和 *I/O 域 3*，但是要把每个域连接至不同的地址变量。
5	组态一个具有*标准对象* → *静态文本*类型的对象。在本实例中，使用了*静态文本 1* 对象。该对象指示当前显示哪一个过程值。该*按钮*自动提供对象中的文本。
6	在同一画面中，组态三个 *Windows 对象* → *按钮*。在本实例中，使用了*按钮 1*、*按钮 2* 和*按钮 3* 对象。
7	对于按钮 1，在*事件* → *鼠标* → *按下左键*处组态一个*直接连接*。将*源常量* → *S32i_varia_adr_12* 与*目标变量* → *T16x_varia_adr_04* 相连接。单击*确定*按钮即可应用这些设置。 
8	在*事件* → *鼠标* → *鼠标动作*处创建另一个*直接连接*。将*源属性* → *本对象* → *文本*与*目标画面中的对象* → *静态文本 1* → *文本*相连。单击*确定*按钮即可应用这些设置。
9	组态*按钮 2* 与*按钮 3* 的方法与组态*按钮 1* 的方法相同。对于第一个直接连接，必须更改*源*的变量名称。第二个*直接连接*无需改变就可应用。

常规应用的注意事项

为了进行常规应用，必须进行下列修改：

- 必须修改变量名。

2.5.2 使用间接寻址和 C 动作进行多重显示(实例 02)

任务定义

显示容器的三个不同过程值。然而，还可以为若干容器建立同样的显示方法。通过选择相应的容器可显示相关的过程值。

概念的实现

为了实现对相应容器的选择，将使用一个 *Windows 对象* → *选项组*。
为了显示过程值，使用一个*智能对象* → *I/O* 域和 WinCC 中的间接寻址选项。
带有相应数值的容器显示在*实例 04* 中。

在 WinCC 项目中的实现

步骤	过程：使用间接寻址进行多重显示
1	在变量管理器中创建九个*有符号的 32 位数*类型的变量。在本实例中，创建了变量 *S32i_varia_adr_03* 至 *S32i_varia_adr_11*。这些变量含有容器的相应过程值。
2	在变量管理器中创建三个具有*文本变量 16 位字符集*类型的变量。在本实例中，使用了 *T16x_varia_adr_01*、*T16x_varia_adr_02* 和 *T16x_varia_adr_03* 变量。它们将用作 *I/O 域*的地址变量。
3	组态三个*智能对象* → *I/O 域*。本实例中所使用的对象是 *I/O 域 5*、*I/O 域 6* 和 *I/O 域 7*。
4	在创建 *I/O 域 5* 期间，可在*组态对话框*中设置 *T16x_varia_adr_01* 变量。将*更新*改为*一旦改变*，将*域类型*改为*输出*。在*属性* → *输出/输入* → *输出值*处，激活*间接*列中的复选框。
5	采用同样的方法，组态其余的 *I/O 域*，但是要把每个域连接至不同的地址变量。
6	组态一个 *Windows 对象* → *选项组*。在本实例中，使用了*选项组 1* 对象。
7	通过*属性* → *字体* → *下标*可选择下标 1。在*属性* → *字体* → *文本* → *容器 1* 处为所选下标输入合适的文本。用同样的方法为其余的下标值组态文本。
8	在*事件* → *属性主题* → *输出/输入* → *所选方框*处，创建 *C 动作*。根据所选择的域，该动作将被写入地址变量。

选项组的 C 动作

```
#include "apdefap.h"
void OnPropertyChanged(char* lpszPictureName, char* lpszObjectName, char* l

{
char address1[20],address2[20],address3[20];

switch(value) {
        case 2: {
                strcpy(address1,"S32i_varia_adr_03");
                strcpy(address2,"S32i_varia_adr_06");
                strcpy(address3,"S32i_varia_adr_09");
                }//case 2
                break;
        case 4: {
                strcpy(address1,"S32i_varia_adr_04");
                strcpy(address2,"S32i_varia_adr_07");
                strcpy(address3,"S32i_varia_adr_10");
                }//case 4
                break;
        default: {
                strcpy(address1,"S32i_varia_adr_05");
                strcpy(address2,"S32i_varia_adr_08");
                strcpy(address3,"S32i_varia_adr_11");
                }//default
                break;
        }//switch

SetTagChar("T16x_varia_adr_01",address1);
SetTagChar("T16x_varia_adr_02",address2);
SetTagChar("T16x_varia_adr_03",address3);
}
```

- 声明三个 *C 变量*作为一个字符数组。
- 按照输入状态，将变量名称复制给先前所声明的变量。输入状态存储在预定义的 *value* 变量中。
- 将相应的变量名称分配给地址变量。

常规应用的注意事项

在进行常规应用之前，必须进行下列修改：

- 必须修改变量名称。

2.5.3 使用 C 动作进行间接寻址(实例 03)

任务定义

在 *I/O 域*中，各种过程值都将显示。通过*选项组*选择相应的值。

概念的实现

为了实现对相应过程值的选择，将使用一个 *Windows 对象* → *选项组*。为了显示过程值，将使用*智能对象* → *I/O 域*和 WinCC 中的间接寻址选项。

在 WinCC 项目中的实现

步骤	过程：使用 C 动作进行间接寻址
1	在变量管理器中创建三个*有符号的 32 位数*类型的变量。在本实例中，使用了变量 *S32i_varia_adr_00*、*S32i_varia_adr_01* 和 *S32i_varia_adr_02*。这些变量包含要显示的过程值。
2	在变量管理器中创建一个具有*文本变量 16 位字符集*类型的变量。在本实例中，使用了 *T16x_varia_adr_00 变量*。该变量将用作地址变量。
3	在画面中，组态*智能对象* → *I/O 域*。在本实例中，使用了 *I/O 域 8* 对象。在创建 *I/O 域*期间，可在*组态对话框*中设置 *T16x_varia_adr_00* 变量。将*更新*域中的缺省值 2s 修改为*一旦改变*，并将*域类型*设置为*输出*。在*属性* → *输出/输入* → *输出值*处，激活*间接*列中的复选框。
4	在同一画面中，组态一个 *Windows 对象* → *选项组*。在本实例中，使用了*选项组 2* 对象。
5	通过*属性* → *字体* → *下标*选择下标 1。在*属性* → *字体* → *文本* → *填充量*处为所选下标输入合适的文本。用同样的方法为其余的下标值组态文本。
6	在*事件* → *属性主题* → *输出/输入* → *所选方框*处，创建 *C 动作*。根据所选择的域，该动作将被写入地址变量。

选项组的 C 动作

```
#include "apdefap.h"
void OnPropertyChanged(char* lpszPictureName, char* lpszObjectName, char* l

{
char address[40];

//set tag according to input value
switch(value) {
   case 2: strcpy(address,"S32i_varia_adr_01");
   break;
   case 4: strcpy(address,"S32i_varia_adr_02");
   break;
   default: strcpy(address,"S32i_varia_adr_00");
   }//switch
SetTagChar("T16x_varia_adr_00",address);
}
```

- 根据输入状态，将变量名称分配给 T16x_varia_adr_00 地址变量。输入状态存储在预定义的 value 变量中。

常规应用的注意事项

在进行常规应用之前，必须进行下列修改：

- 必须修改变量名称。

2.5.4 本主题的其余实例

实例 04

本实例的功能是显示用在*实例 02* 的过程值。

2.6 变量的模拟

通过用选择如上所示的*按钮*可在 *Project_TagHandling* 项目中访问与该主题相关的解决方案。这些实例均在 *varia_3_chapter_05.pdl* 画面中组态。

定义

术语模拟指的是无需过程驱动程序连接就更改变量的内容。模拟通过实用程序来执行。

2.6.1 通过 C 动作对三角振荡进行模拟(实例 01)

任务定义

创建一个可设置最大值和最小值的三角振荡模拟。输入这些数值的时候要证实它们的真实性。通过按钮，可启动和停止模拟。使用另一个*按钮*可将变量值重新设置为 0。

概念的实现

为了实现模拟的启动/停止以及初始化，使用两个 Windows 对象 → 按钮。为了显示变量值并输入最大值和最小值，使用智能对象 → I/O 域。如果模拟开始，而设置的最大值和最小值一样的话，将会显示一个消息框。

在 WinCC 项目中的实现

步骤	过程：通过 C 动作对三角振荡进行模拟
1	在变量管理器中创建三个*有符号的 32 位数*类型的变量。在本实例中，使用了变量 *S32i_varia_sim_00*、*S32i_varia_sim_02* 和 *S32i_varia_sim_03*。
2	创建两个*二进制*变量类型的变量。在本实例中所使用的变量是 *BINi_varia_sim_01* 与 *BINi_varia_sim_04*。
3	组态三个*智能对象* → *I/O 域*。本实例中所使用的对象是 *I/O 域 1*、*I/O 域 2* 和 *I/O 域 3*。
4	在创建 *I/O 域 1* 期间，在*组态对话框*中设置*变量 S32i_varia_sim_03* 和触发器一旦改变。在*属性* → *输出格式*处，将格式改为 *0999*。采用同样的方式组态 *I/O 域 2*，但要设置变量 *S32i_varia_sim_02*。
5	为了检查 *I/O 域 1* 对象的真实性，在*属性* → *限制值* → *上限值*处组态一个与 *S32i_varia_sim_02* 变量的*变量连接*。采用同样的方法，将 *S32i_varia_sim_03* 变量组态为 *I/O 域 2 的下限值*。
6	创建 *I/O 域 3* 对象期间，在组态对话框中设置变量 *S32i_varia_sim_00*、触发器一旦改变并将*域类型*设置为*输出*。在*属性* → *输出/输入* → *输出格式*处，将格式改为 *0999*。
7	组态一个*智能对象* → *画面窗口*，本实例组态的是*对话框*。通过*属性* → *其它*，将属性*可移动*和*边框*改为*是*，并将*画面名称*改为 *varia_5_window_00*。可以在自己的项目中使用实例项目中的画面；并可对信息文本和标题进行修改，以满足项目要求。
8	组态一个 *Windows 对象* → *按钮*，本实例所组态的是*按钮 2*。在*事件* → *鼠标* → *按下左键*处，创建一个*直接连接*。将*源常数* → *1* 连接至*目标变量* → *BINi_varia_sim_04*。该*按钮*将用于初始化。

步骤	过程：通过 C 动作对三角振荡进行模拟
9	组态另一个 *Windows 对象* → *按钮*。在本实例中，使用了*按钮 1* 对象。在*事件* → *鼠标* → *按下左键*处，创建一个 *C 动作*，它将对 *BINi_varia_sim_01* 变量的状态求反。在*属性* → *几何结构* → *位置 X* 处，创建一个执行变量模拟的 *C 动作*。
10	为了对模拟状态进行显示，组态了一个*智能对象* → *状态显示*。在本实例中使用了*状态显示 1*。在*组态对话框*中，设置变量 *BINi_varia_sim_01* 和触发器“一旦改变”。添加另一个状态。对状态 *0*，设置画面 *glühbirne_2_24Bit.gif*，而对状态 *1*，则设置画面 *glühbirne_1_24Bit.gif*。

用于变量模拟的 C 动作

```
#include "apdefap.h"
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyName)
{
BOOL state;
static DWORD lowstore = 0;
static BOOL statestore = 0;
static DWORD r = 1;
static DWORD i = 0;
static DWORD box = 0;
int high,low;

//if button init was pressed
if (GetTagBit("BINi_varia_sim_04")) {
      (i=lowstore);
      (r=1);
      SetTagDWord("S32i_varia_sim_00",i);
      SetTagDWord("BINi_varia_sim_04",0);
        }

//get simulator state
state=GetTagBit("BINi_varia_sim_01");

if (state!=statestore) (box=0);

statestore=state;

//get limits
high=GetTagDWord("S32i_varia_sim_02");
low=GetTagDWord("S32i_varia_sim_03");

//set low limit store
if (low!=lowstore){
        lowstore = low;
        i=low;
        }//if

//if limits different
if (high!=low) {
        box=0;
        //if simulator is activated
        if (state==TRUE) {
     //inc or dec according to direction
        if (r==1) (i=i+1);
        else (i=i-1);
        //set direction
        if (i==high) (r=0);
        if (i==low) (r=1);
        //init simulator if limit overflow
        if ((i>high)||(i<low)){
                (i=low);
                (r=1);
                }//if
        //set new value
        SetTagDWord("S32i_varia_sim_00",i);
        }//if state
}//if (high!=low)
//set visible message box
if ((high==low)&&(state==1)&&(box==0)){
        box++;
        SetVisible("varia_3_chapter_05.PDL","dialog box",1);
        }
return 80;//x-pos
}
```

- 变量的声明。
- 如果*按钮 2*(初始化)已被按下，则将变量值存储器设置为已存储的最小值，将计数器方向设置为递增方向，将*内部变量 S32i_varia_sim_00* 的值设置为已存储的最小值，并关闭模拟器。
- 在模拟器状态下读取。
- 如果该状态已经改变，则允许弹出消息框。
- 保存状态。
- 在最大值和最小值中读取。
- 当最小值改变时，更新最小值存储器。
- 如果最大值和最小值不一样，则允许弹出消息框，并且打开模拟器时会执行模拟。向上或向下计数依方向变量而定；到达限制值时，方向改变；如果超过限制值，则进行初始化，并且将变量 S32i_varia_sim_00 设置到最小值。
- 如果打开模拟器，激活消息框的显示，且最大值和最小值一致，则将消息框设置为可见。
- 返回值是*按钮 1* 对象的 X 位置。

2.6.2 通过外部程序进行模拟(实例 02)

WinCC 提供自带模拟程序，该模拟程序可以用数种不同的方法模拟变量。安装该模拟程序时，必须使用 Setup.exe 程序，该安装程序位于 WinCC 光盘中的文件夹 SmartTools → CC_Simulator 内。

任务定义

使用 WinCC 变量模拟器模拟变量。

概念的实现

为了实现该操作，将使用多个变量，它们将显示在*智能对象* → *I/O 域*中，其内容将由变量模拟器进行控制。

在 WinCC 项目中的实现

步骤	过程：通过外部程序进行模拟
1	在变量管理器中创建两个具有*有符号的 32 位数*类型的*内部变量*。在本实例中，使用了变量 *S32i_varia_sim_05* 和 *S32i_varia_sim_06*。
2	组态两个*智能对象* → *I/O 域*类型的对象。在本实例中，使用了 *I/O 域 4* 和 *I/O 域 5* 对象。
3	在创建 *I/O 域 4* 期间，可在组态对话框中对变量 S32i_varia_sim_05 进行设置，将触发器设置为一旦改变，将域类型设置为输出。在*属性* → *输出/输入*处，将*输出格式*改为 *0999.999*。采用同样方式，对 *I/O 域 5* 进行设置，但要设置 *S32i_varia_sim_06* 变量。

步骤	过程：通过外部程序进行模拟
4	变量模拟器的启动可通过单击*模拟器*按钮来进行。启动变量模拟器要利用一个 *C 动作*，它位于*事件* → *鼠标* → *按下左键*处。如果还没有按缺省路径安装变量模拟器，则可通过*路径*按钮设置正确的路径。如果使用其它的方法，而非这里所描述的方法启动模拟程序，则必须确保上述项目在运行时模式下运行。
5	在所显示的模拟器中，可通过*编辑* → *新变量菜单*从变量管理器中选择 *S32i_varia_sim_05* 变量。选择 *Inc* 标签，并输入*起始值*和*结束值*。在本实例中，已使用了值 *0* 和 *20*。选择*激活的*菜单项即可启动模拟。变量值从 0 增加到 20，之后以 0 重新启动模拟。
6	采用同样方式处理 *S32i_varia_sim_06* 变量。在本实例中，已选择了*正弦*标签：已将*振幅*设置为 *50*，将*偏移量*设置为 *50* 以及将*振荡时间*设置为 *25*。

模拟器按钮上的 C 动作

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpsz
{
char simdeupath[200];
char program[15];
int result;

if (GetTagBit("BINi_varia_sim_10")) strcpy(program,"\\simeng.exe");
else strcpy(program,"\\simdeu.exe");

//build path to simdeu
strcpy(simdeupath,GetTagChar("T16x_varia_sim_07"));
strcat(simdeupath,program);

//execute simdeu
result=ProgramExecute(simdeupath);

//if not able to execute simdeu set visible message box
if (result<32) {
      SetVisible("varia_3_chapter_05.PDL","error message",1);
      }
}
```

- 变量的声明。
- 如果已对 BINi_varia_sim_10 变量进行了设置，则将英文模拟器程序的名称写入程序变量。否则，将德文模拟器的名称写入程序变量。
- 通过内部函数 GetTagChar，可读出模拟器程序的路径。
- 将启动文件添加给路径。
- 启动模拟器。
- 如果路径指定不正确，则输出一条出错信息。

注意：
在本实例项目中，选择标志图标既可启动英文也可启动德文版本的变量模拟器。

常规应用的注意事项

在进行常规应用之前，必须进行下列修改：

- 必须自定义需要模拟的变量和模拟方法，以便满足各自的要求。

2.7 导入/导出变量

访问 *Project_TagHandling* 项目中与该主题相关的解决方案，可通过使用，选择上面显示的*按钮*来完成。这些实例均组态在 *varia_3_chapter_06.pdl* 画面中。

任务定义

通过实用程序读出变量管理器的内容，并在 MSExcel(电子表格程序)中进行编辑。经过修改的数据可以再次导入 WinCC 项目。这样一来，可以轻松创建大量变量。

概念的实现

为了完成该项目中的实现，使用了两个 *Windows 对象* → *按钮*，以便启动 *var_imex.exe* 导入/导出程序和 *excel.exe* 程序。对这些程序中各个程序的路径进行设置可通过两个*智能对象* → *I/O 域*来进行。

在 WinCC 项目中的实现

步骤	过程：导入/导出变量
1	将正确的路径设置给 *excel.exe* 和 *var_imex.exe* 程序。
2	在运行时期间，启动 *var_exim.exe* 程序可通过单击*按钮导入/导出*来进行。程序也可从 Windows 资源管理器中直接启动，不需要激活运行系统。
3	通过按钮 ....，将路径设置给 *Project_TagHandling* 项目，并选择 *Project_TagHandling.mcp* 文件。
4	选择导出选择域。实例 1.6.2 对用于调用外部程序的 C 动作进行了描述。单击*执行* → *确定*。即可完成变量的导出。该程序将产生一个带有扩展名 vex 的文件，它包含有与该变量相关的信息，程序所产生的第二个文件带有扩展名 cex，它包含有关于与 PLC 连接的信息。程序产生的第三个文件，其扩展名为 *dex*，该文件包含了与*数据结构*类型的变量相关的信息。
5	启动 *Excel*，并打开 *Project_vex.csv* 文件，该文件是刚才通过*文件* → *打开*所产生的。
6	若要创建 100 个无符号的 16 位数类型的变量，则需如下操作。*分配给这些变量的名称，其范围可从 U16i_varia_impex_00 到 U16i_varia_impex_99*。
7	在第一列的第一个空行中输入名称 *U16i_varia_impex_00*。选择该单元格并把鼠标指针指向右下角。在持续按下期间，将鼠标指针向下拖动，即可自动填充所保留的 99 个单元格。

步骤	过程：导入/导出变量
8	在第二列中，输入一个*；在第三列中，输入*内部变量*；在第四列中，输入*impexp*作为组的名称；在第五列中，输入*2*，而在第 6 列中输入*5*作为用于*无符号的 16 位数*的代码。在剩余的列中，输入值*0*。自动填充剩余的 99 行。
9	通过任务栏再次打开 *Var_imex.exe*，并选择*导入*选择域。然后，单击*执行* → *确定*。完成变量导入以后，退出程序。
10	在变量管理器中已创建 100 个新变量。

注意：
在导入和导出变量时，运行不必一定呈激活状态。

2.8 结构变量的使用

结构

访问 *Project_TagHandling* 项目中与该主题相关的解决方案，可通过使用，选择上面显示的*按钮*来完成。这些实例均组态在 *varia_3_chapter_07.pdl* 画面中。

定义

该数据类型允许生成形成逻辑单位的数据结构。结构变量由各种缺省数据类型组成。

2.8.1 使用结构变量对阀进行控制(实例 01)

任务定义

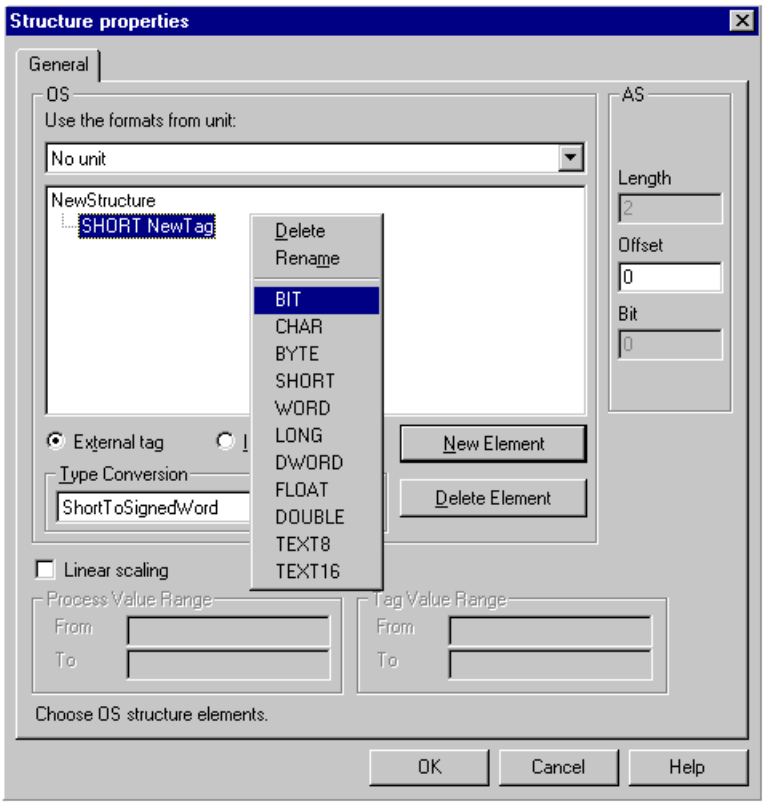
借助结构变量显示阀的不同状态。

概念的实现

为了完成这种实现，可使用两个 *Windows 对象* → *按钮*，用它们可打开和关闭阀，并可模拟故障条件。为了对阀进行显示，使用了*标准对象* → *多边形*。

在 WinCC 项目中的实现

步骤	过程：使用结构变量对阀进行控制
1	在 *WinCC 资源管理器*中定义了一个新的 WinCC 结构变量。在*数据类型*处，通过 R *选择* → *结构类型*，然后从弹出菜单中选择*新的结构*。

步骤	过程：使用结构变量对阀进行控制
2	在下列窗口中，R *新建结构*，然后从弹出菜单中选择*重命名*。在本实例中，使用的名称是*阀*。通过*新建元素*按钮，可添加一个新的结构元素。通过R 刚才所创建的元素，可将其数据类型设置为 *B 位*。 
3	通过*重命名*按钮，可将元素名称设置为*已激活的*，并选择*内部*选项钮。定义下列结构元素： valve BIT aktivated BIT open BIT closed BIT error

步骤	过程：使用结构变量对阀进行控制
4	在变量管理器中创建一个*阀*类型的变量。在本实例中，使用了 *STUi_varia_str_00* 变量。该动作将创建下列*二进制变量*。 Name / Type / Parameters / Last change STUi_varia_str_00.aktivated — Binary Tag — Internal tag — 07/10/97 02:26:14 STUi_varia_str_00.open — Binary Tag — Internal tag — 07/10/97 02:26:14 STUi_varia_str_00.closed — Binary Tag — Internal tag — 08/21/97 03:45:32 STUi_varia_str_00.error — Binary Tag — Internal tag — 07/10/97 02:26:14
5	组态两个 *Windows 对象* → *按钮*，在本实例中，使用了对象*按钮 1* 和*按钮 2*。在*按钮 1* → *事件* → *鼠标动作* → *按下左键*处，创建一个 *C 动作*，用来将阀打开或关闭。采用同样方式，在*按钮 2* 上创建一个 *C 动作*，用来打开或关闭错误位。
6	对于*按钮 1*，创建一个 *C 动作*，该动作位于*属性* → *几何结构* → *位置 X* 上，可对阀上的外部过程进行模拟。
7	然后，创建三个不同的画面，以显示阀的开、关和错误状态。在本实例中，这些画面将包括两个*标准对象* → *多边形*。它们以一个多边形在另一个多边形的顶部的形式放置，并根据阀的状态来决定是显示还是隐藏多边形。

常规应用的注意事项

在进行常规应用之前，必须进行下列修改：

- 结构类型名称及其构成结构元素类型和结构元素名称等均需进行修改。
- 实际应用中，不需要模拟关于阀的外部过程的 *C 动作*。

3 画面组态(Project_CreatePicture)

本章中创建的项目也可以直接从在线文档复制到硬盘驱动器上。缺省情况下，它将存储在 C:\Configuration_Manual 文件夹中。

Project_CreatePicture

项目细节

本项目描述了在 WinCC 内构造、打开画面的各种方法。
画面构造和画面打开取决于以下两个因素：所使用的硬件(带有 OP47 操作面板的工业 PC，或在控制室内的带有鼠标和标准键盘的 PC)和应用程序。不同的用户对 HMI 系统的要求也不同，例如机械制造商和化学制品公司。

WinCC 可提供何种可能性？

WinCC 支持所有 Windows 支持的屏幕分辨率(例如 640x480、800x600、1024x768、1280x1024)。有时，总览画面的显示需要更高的分辨率(例如 1600x1028、2000x1500 等)。
WinCC 允许创建最大分辨率为 4096x4096 像素的画面。如果这些尺寸超过所使用图形系统(监视器的显示卡)的最大分辨率，可以使用滚动条移动这些画面。

假定：

假设所使用的图形系统的分辨率是 1024x768 像素。与该分辨率相适应的是带有 17"监视器的图形系统。
有关该主题的实例将在 *Project_CreatePictureWinCC* 项目中进行组态。

注意：
登录口令是 pictu_00。只需单击窗口标题上的登录，并在口令输入域中输入口令，然后确认刚才的输入即可。

3.1 画面布局与画面切换

本章描述多种构造和打开画面的方法。在其它项目中也使用画面布局的基本元素(起始画面、总览部分和按钮部分)。

3.1.1 画面布局

任务定义

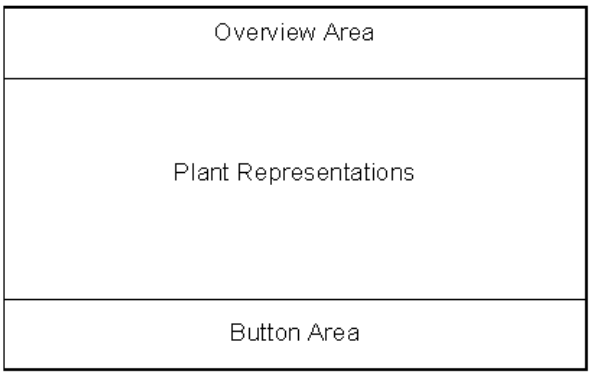
动态按钮设置和总览部分

画面分成三部分：
总览部分、按钮部分和现场画面部分。
可以调整总览和按钮部分。
系统位于控制室内，通过鼠标和键盘进行控制。

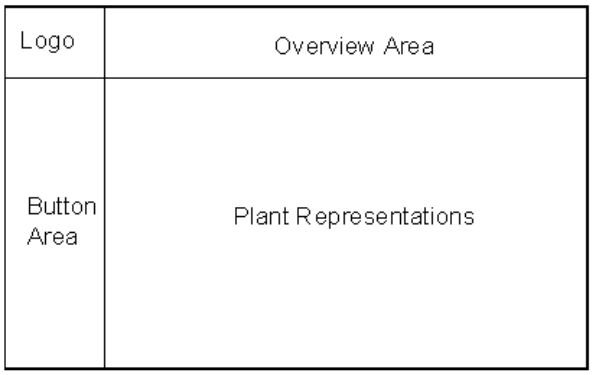
概念的实现

画面分辨率设置为 1024x768 像素。整个画面分成三部分。对于这三个部分有两种不同的布局方法。

画面布局 1



画面布局 2



布局原理

使用一个空白起始画面，然后在其中创建三个画面窗口(总览、按钮、现场)。运行期间，可以根据需要交换这些画面窗口内显示的画面。这就给了我们一种简便而又灵活的修改方法。

总览部分

在总览部分，可组态标志符、画面标题、带有日期和时间的时钟以及报警行。

按钮部分

在按钮部分，组态在每个画面中显示的固定按钮和依靠现场画面显示而显示的按钮。

现场部分

在现场部分，组态各个设备画面。

3.2 画面切换

在运行系统中，访问 *Project_CreatePicture* 项目中与该主题相关的实例，可通过使用选择上面所显示的*按钮*来完成。这些实例均组态在 *pictu_3_chapter_01.pdl* 和 *pictu_3_chapter_01a.pdl* 画面中。

3.2.1 通过直接连接打开画面并显示画面名称(实例 01)

任务定义

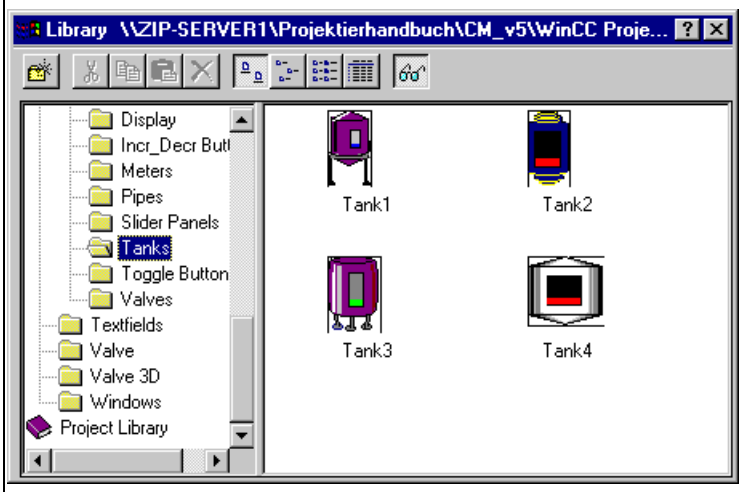
在画面窗口中，通过用鼠标操作的按钮并借助直接连接，可完成画面切换。

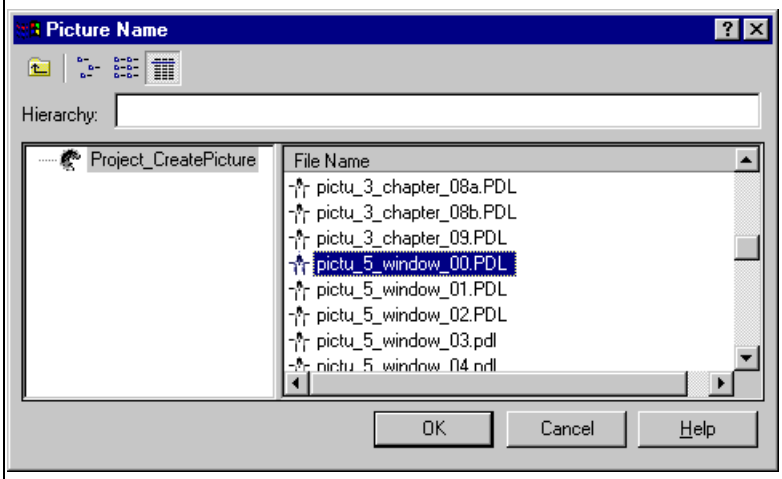
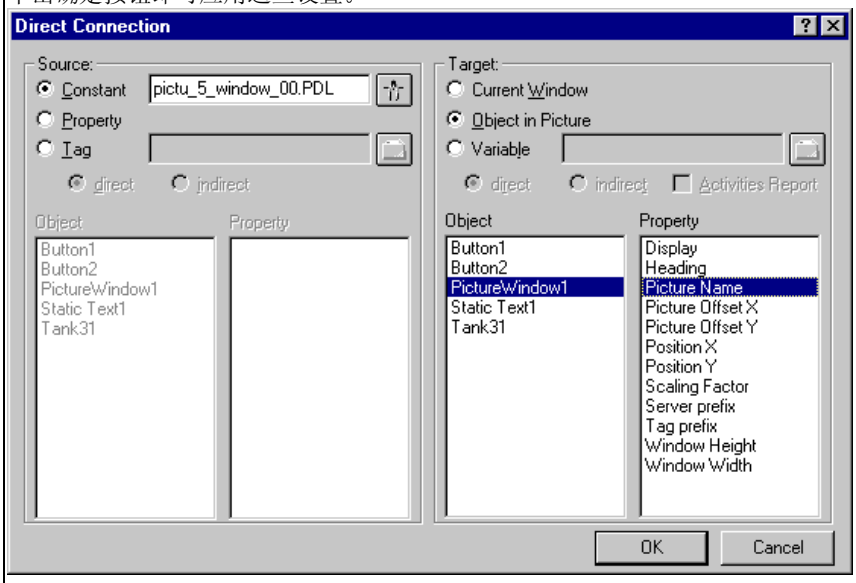
概念的实现

为了实现概念，可使用一个 *Windows 对象* → *按钮*，当使用将其按下时，即可对显示在*智能对象* → *画面窗口*中的画面进行切换。在画面中，使用*标准对象* → *静态文本*可显示画面名称。

在 WinCC 项目中的实现

步骤	过程：通过直接连接打开画面并显示画面名称
1	通过*文件* → *新建*，创建一个新的画面，并通过*文件* → *另存为...*，将其以名称 *pictu_5_window_00.pdl* 进行保存。将*属性* → *几何结构* → *宽度*设置为 *270*，并将*属性* → *几何结构* → *高度*设置为 *280*。
2	在 *pictu_5_window_00.pdl* 画面中，组态一个*标准对象* → *静态文本*。在本实例中，使用了*静态文本 1* 对象。将*属性* → *字体* → *粗体字*设置为*是*。在*属性* → *字体* → *文本*处，将缺省文本从*静态*列中删除。这样可以避免建立画面时输出不正确的文本。 可使用 *C 动作*在*属性* → *字体* → *文本*处动态地制作对象。该 *C 动作*将把当前画面名称作为返回值返回。作为用于 *C 动作*的触发器，使用了*缺省周期* → *1 小时*(低系统负载，不需任何修改)。 Event Name: 1 h

步骤	过程：通过直接连接打开画面并显示画面名称
3	在 *pictu_5_window_00.pdl* 画面中，组态所要显示的信息。在实例中，使用了全局库中的 *tank3* 对象。对库的访问可通过*查看* → *库*或通过工具栏上的按钮来进行。 请确保已通过*按钮*选择了符号查看，以便对单个对象进行预览。 
4	组态两个以上用于画面切换的实例画面，这可通过利用*文件* → *另存为...*以名称 *pictu_5_window_01.pdl* 保存刚才所组态的画面，再以名称 *pictu_5_window_02.pdl* 保存该画面以便得到另一个画面来实现。这样将可获得 *pictu_5_window_00.pdl* 的两份拷贝。现在即可将所期望的内容插入到刚才创建的新画面中。显示画面名称时，不需要改变对象*静态文本 1*。
5	组态一个新的画面可通过*文件* → *新建*来进行。在该画面中，组态了一个*智能对象* → *画面窗口*。在本实例中，使用了*画面窗口 1* 对象。将*画面窗口*的尺寸修改为先前所创建画面的大小可通过*属性* → *几何结构* → *宽度*和*属性* → *几何结构* → *高度*来完成。为了使窗口在运行系统中带边框显示，可将*属性* → *其它* → *边框*设置为*是*。

步骤	过程：通过直接连接打开画面并显示画面名称
6	在*属性* → *其它* → *画面名称*上，设置 *pictu_5_window_00.pdl* 画面。这样在打开画面时，就可通过*画面窗口 1* 对象来将画面设置为显示。 
7	在同一画面中，组态 *Windows 对象* → *按钮*。在本实例中，这就是对象*按钮 1*。在*事件* → *鼠标* → *按下左键*处，创建一个直接连接。 将*常数*选择为*源*，并单击现在已激活的按钮，以便显示所有可用画面的选择列表。选择画面 *pictu_5_window_00.pdl* 和*目标画面中的对象*，对象*画面窗口 1* 和对象属性*画面名称*。 单击*确定*按钮即可应用这些设置。 

步骤	过程：通过直接连接打开画面并显示画面名称
8	使用选择所组态的*按钮 1* 对象，并通过*编辑* → *复制*对其进行复制。再次重复上述过程。现在即可得到两个以上的按钮，*按钮 2* 和*按钮 3*。在*事件* → *鼠标* → *按下左键*处，更改所组态的*直接连接*。对于*按钮 2*，将*源*设置为 *pictu_5_window_01.pdl*，而对于*按钮 3*，则设置为 *pictu_5_window_02.pdl*。

静态文本 1 的 C 动作

```
#include "apdefap.h"
char* _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty

{
char    *name = lpszPictureName;
char    *pdest;
int     ch = ':';

//check if picture path contains char
pdest = strrchr( lpszPictureName, ch );
//read only picture name without path
if ( pdest == NULL ) return lpszPictureName;
else {
        name = strcpy(name,strrchr(name,':')+1);
        return name;
        }//else
}
```

- 声明 *C 变量*。
- 检查 *lpszPictureName* 是否只含有画面名称。使用 *strrchr* 函数即可实现。该函数将通过 *lpszPictureName* 进行搜索。如果画面显示在*画面窗口*中，则 *lpszPictureName* 包含了带有画面完整路径的画面名称。
- 在前一种情况下，直接将 *lpszPictureName* 作为返回值返回。
- 在后一种情况下，将只从画面路径中读出画面名称，并将该名称作为返回值返回。

常规应用的注意事项

在进行常规应用前，必须完成下述修改：

- 可将静态文本 1 对象直接传送给任何其它的画面窗口。该对象亦适合存储在项目库中。这样一来，可以通过拖放的方式把这个对象方便地插入任何画面。
- 对按钮 1 对象处的直接连接，必须对所要显示的画面名称和画面窗口的对象名称进行修改。
- 必须布置好将要显示的画面、画面内容和*画面窗口*以满足各自的需要。画面的高度和宽度应该与*画面窗口*一致。

3.2.2 利用动态向导打开画面(实例 02)

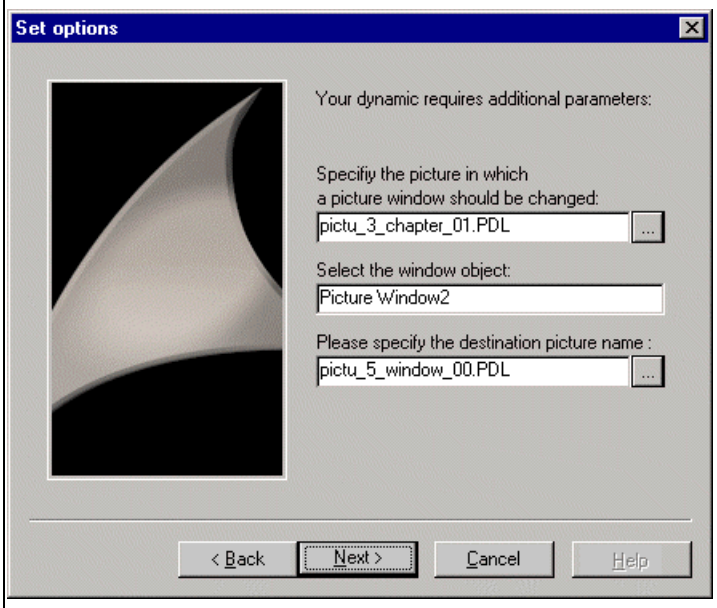
任务定义

在*画面窗口*中，利用一个鼠标控制的*按钮*可完成画面的切换。使用*动态向导*，则可完成组态。

概念的实现

为了实现概念，可使用一个窗口对象 → 按钮，当使用R 将其按下时，即可对智能对象 → 画面窗口所显示的画面进行切换。将使用在前例中已组态的画面。

在 WinCC 项目中的实现

步骤	过程：通过动态向导打开画面
1	在画面中，组态一个*智能对象* → *画面窗口*。在本实例中，这就是*画面窗口 2*。调整*画面窗口*的尺寸以适应屏幕大小，并将*属性* → *其它* → *边框*设置为*是*。通过*属性* → *其它* → *画面名称*，选择 *pictu_5_window_01.pdl* 画面。
2	如果没有显示*动态向导*，则从*查看* → *工具栏*里将其激活。
3	在同一画面中，组态一个 *Windows 对象* → *按钮*。在本实例中，使用了*按钮 4* 对象。在将对象加亮时，选择*画面函数*标签，然后通过从*动态向导*中D，选择*窗口中的画面切换*条目。从*动态向导*的*选择触发器*页面中，选择*鼠标右键*，并且通过单击*下一个*进入下一页。完成*设置选项*页面如下：  通过*按钮*，可访问项目可利用的画面。确认*完成*！页面可通过单击*完成*按钮来实现。

步骤	过程：通过动态向导打开画面
4	组态了两个附加的 *Windows 对象* → *按钮*。在本实例中，它们是对象*按钮 5* 和 *按钮 6*。将*动态向导*也应用于这些按钮。在用于*按钮 5* 的*设置选项*页面中，将 *pictu_5_window_01.pdl* 设置为*目标地址画面名称*，而 *pictu_5_window_02.pdl* 则用于*按钮 6*。

由动态向导生成的 C 动作

```
#include "apdefap.h"
void OnRButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
static char szPicture[22] = "pictu_5_window_00.PDL";
static char* tmp = &szPicture[0];
PDLRTSetPropEx(PDLRT_AM_PICTABS, "pictu_3_chapter_01",
        "Picture Window2",
        "PictureName",VT_LPSTR,&tmp,NULL,NULL,0,NULL, NULL);
}
```

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 为了满足各自的需要，必须对*动态向导*设置进行修改。

3.2.3 通过内部函数打开画面(实例 02)

任务定义

在*画面窗口*中，通过一个鼠标控制的*按钮*，可完成画面的切换。通过一个 *C 动作*，可完成*按钮*上的组态。

概念的实现

为了实现概念，可使用一个*窗口对象* → *按钮*，当使用 R 将其按下时，即可对*智能对象* → *画面窗口*所显示的画面进行切换。将使用由前例所得到的画面。

在 WinCC 项目中的实现

步骤	过程：通过内部函数打开画面
1	在画面中，组态一个*智能对象* → *画面窗口*。在本实例中，这就是*画面窗口 2*。调整*画面窗口*的尺寸以适应屏幕大小，并将*属性* → *其它* → *边框*设置为*是*。通过*属性* → *其它* → *画面名称*，选择 *pictu_5_window_01.pdl* 画面。
2	在同一画面中，组态 *Windows 对象* → *按钮*。在本实例中，使用了对象*按钮 4*。在*事件* → *鼠标* → *按下左键*处，可创建 *C 动作*，用于画面切换。组态了两个*附加的按钮*。在本实例中，它们就是对象*按钮 5* 和*按钮 6*，这些对象上均设置有经适当修改了的 *C 动作*。

按钮 4 的 C 动作

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
SetPictureName("pictu_3_chapter_01.PDL",
               "Picture Window2",
               "pictu_5_window_00");
}
```

- 通过*内部函数 SetPictureName*，可将 *pictu5_window_00.pdl* 画面切换到*画面窗口 2* 对象中。*pictu_3_chapter_01.pdl* 是*画面窗口*所在画面的名称。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 为了满足各自的需要，必须对*内部函数 SetPictureName* 的参数进行修改。

3.2.4 通过动态向导进行单个画面的切换(实例 03)

任务定义

通过鼠标控制的*按钮*，可切换运行系统中所显示的画面。使用*动态向导*可完成组态。

概念的实现

为了实现概念，使用了一个 *Windows 对象* → *按钮*，用单击该按钮，可切换运行系统中所显示的画面。

在 WinCC 项目中的实现

步骤	过程：通过动态向导切换单个画面
1	在本实例中，将画面从 *pictu_0_startpicture_00.pdl* 切换到 *pictu_3_chapter_01a.pdl* 画面即可完成画面的切换。在本实例项目中，总是选择 *pictu_0_startpicture_00.pdl* 画面，并且从这里开始，只在窗口中执行画面的切换。通过使用动态向导生成的 *C 动作*，运行系统中所显示的整个画面系统将被所调用的画面系统代替。切换回 *pictu_0_startpicture_00.pdl* 有如完全重新启动画面项目。
2	在该画面中，组态一个 *Windows 对象* → *按钮*。在本实例中，*使用了对象按钮 7*。 Picture Change via object name >>
3	对象加亮时，选择*画面函数*标签，然后通过从*动态向导*中D，选择*单个画面切换*条目。从*动态向导*的*选择触发器*页面中，选择*鼠标左键列表项*，并通过单击*下一个*按钮对下一页继续操作。完成的*设置选项*页面如下： Please specify the new picture name: pictu_3_chapter_03a.PDL ... 通过*按钮* ...，可显示项目中所有可用画面的列表。确认*完成*！页面可通过单击*完成*按钮来实现。
4	如果完成了实例项目中的画面切换，则单击按钮 << Back 以便返回到项目中。

由动态向导生成的 C 动作

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszF

{
OpenPicture("pictu_3_chapter_01a.PDL");
}
```

- *动态向导*将生成一个 *C 动作*。该 *C 动作*将使用*标准函数 OpenPicture* 来将 *pictu_3_chapter_01a.pdl* 画面切换到运行系统。所生成的 *C 动作*也可由用户来编写。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 为了满足各自的需要，必须对*动态向导*设置进行修改。

3.2.5 利用直接连接切换单个画面(实例 04)

要访问 *Project_CreatePicture* 项目的实例可单击上面所显示的*按钮*。

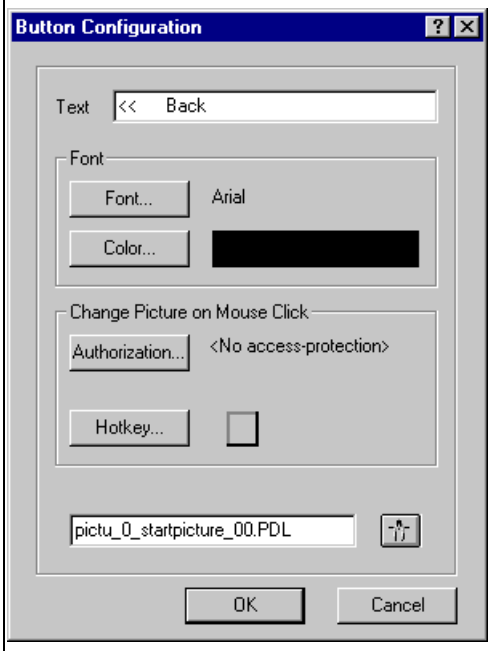
任务定义

与前面的实例相反，单击一个由鼠标控制的*按钮*将切换整个画面。这将不会只是切换画面窗口的内容，而是打开一个新的窗口。

概念的实现

为实现该概念，将使用一个 *Windows 对象* → *按钮*，当使用单击此按钮时，该按钮将执行画面切换。通过*直接连接*来完成该组态。

WinCC 项目中的实现

步骤	过程：通过直接连接切换单个画面
1	在本实例中，将执行从画面 *pictu_3_chapter_01a.pdl* 到画面 *pictu_0_startpicture_00.pdl* 的切换。
2	在画面中，将组态一个 *Windows 对象* → *按钮*。*在本实例中，使用了对象按钮 7*。
3	在*按钮组态*对话框的*单击鼠标切换画面*部分，使用选择按钮来选择 *pictu_0_startpicture_00* 画面。这将在*事件* → *鼠标* → *鼠标动作*处自动生成一个*直接连接*。该连接也可通过*对象属性*对话框生成。 

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 在对象*按钮 7* 处的*直接连接处*，必须对画面名称和该画面窗口的对象名称进行修改。

3.2.6 通过对象名称和内部函数打开画面(05)

Project_CreatePicture 项目中的此实例通过单击如上所示的*按钮*来访问。

任务定义

在*画面窗口*中，画面切换要通过一个用鼠标操作的按钮来执行。*按钮*要通过其对象名称来识别它将调用哪个画面。因此，按钮在复制之后，只有更改了对象名才能再次使用。

概念的实现

为了实现上述任务，将使用一个*窗口对象* → *按钮*，当用按下按钮时，它会切换*智能对象* → *画面窗口*中所显示的画面。这里将使用先前实例中已组态的画面。这些画面的名称由两部分组成：文本部分和画面编号。

在 WinCC 项目中的实现

步骤	过程：通过对象名称和内部函数打开画面
1	在画面中，组态一个*智能对象* → *画面窗口*。在本实例中，使用对象*画面窗口 1*。调整*画面窗口*的尺寸，使其与先前创建的画面相匹配。为了使窗口在运行时带边框显示，将*属性* → *其它* → *边框*设置为*是*。 通过*属性* → *其它* → *画面名称*，选择画面 *pictu_5_window_00.pdl*。
2	在同一画面中，组态一个 *Windows 对象* → *按钮*。在本实例中，使用对象*按钮 0*。在*事件* → *鼠标* → *按下左键*处，组态一个读取*按钮*的名称和编号并根据设置名称约定来显示期望画面的 *C 动作*。
3	复制*按钮 0* 对象两次，并更改*按钮 1* 和*按钮 2* 的对象名。

按钮 0 的 C 动作

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
char    name[30];
int     number;
int     ch = 'n';
char    *pdest;

//check if object name contains character
pdest = strrchr( lpszObjectName, ch );
if ( pdest == NULL )(printf("ObjectNameError"));
else {
   //read object number
   number = atoi(strrchr(lpszObjectName,'n')+1);
   //generate picture name
   sprintf(name,"pictu_5_window_%02d.PDL",number);
   //set picture name
   SetPictureName("pictu_3_chapter_01a.PDL","Picture Window1",name);
   }
}
```

- 声明 *C 变量*。
- 检查对象是否已经按照约定的规则进行命名。对象被给予的名称是[按钮]+[要调用画面的编号]。
- 如果在编号前面没有发现字符(即 n)，则输出一条错误消息。
- 读取按钮名称的编号。*strrchr* 函数向后对名称进行搜索，以寻找字符 n。通过 *atoi* 函数将 *n* 后跟的字符串取出，并将其转换为一个整数值。
- *sprintf* 函数用画面名称和画面编号这两部分生成要通过*按钮*调用的完整画面名称。
- 通过*内部函数 SetPictureName*，要调用的画面被转换为*画面窗口 1* 对象。

常规应用的注意事项

在进行常规应用前，必须完成下列修改：

- *按钮的 C 动作*以及对象名的分配必须根据自己的命名约定进行修改。确保对象名和画面名都严格遵守其命名约定，从而保证访问所期望的画面时不会发生错误。

3.2.7 通过对象名称以及与画面名称显示的变量连接来打开画面(实例 06)

Project_CreatePicture 项目中的此实例通过单击如上所示的*按钮*来访问。

任务定义

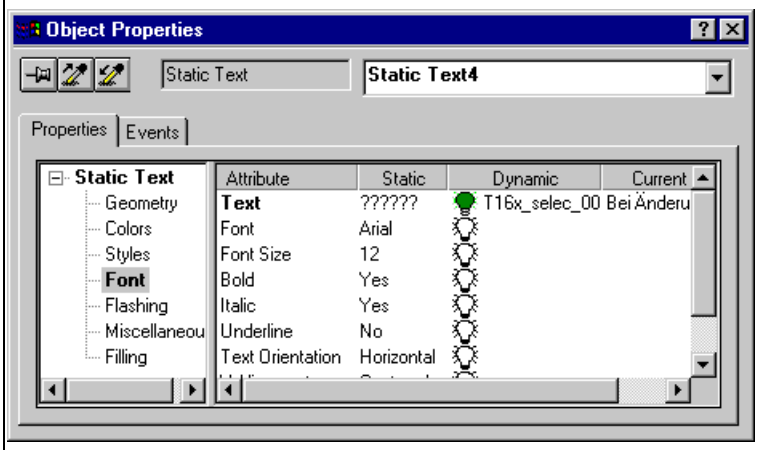
在*画面窗口*中，画面切换要通过一个用鼠标操作的按钮来执行。*按钮*要通过其对象名称来识别它将调用哪个画面。因此，按钮在复制之后，只有更改了对象名才能再次使用。画面名称要存储在文本变量中，并且显示在文本域(它并不在实际画面内)中。

概念的实现

为实现上述任务，将使用一个 *Windows 对象* → *按钮*，当用单击此按钮时，该按钮会切换*智能对象* → *画面窗口*中所显示的画面。这里将使用先前实例中已组态的画面。这些画面的名称由两部分组成：文本部分和画面编号。此外，还使用一个*标准对象* → *静态文本*，用于画面名称的显示。

在 WinCC 项目中的实现

步骤	过程：通过对象名称以及与画面名称显示的变量连接来打开画面
1	在变量管理器中创建一个*文本变量 16 位字符集类型*的变量。在本实例中，使用变量 *T16x_selec_00*。此变量包含画面窗口中所显示画面的名称。
2	打开画面对象 *pic_chapter_01a.pdl* 的属性对话框。 在*事件* → *其它* → *打开画面*处，组态一个将画面名称 *pictu_5_window_01.pdl* 分配给 *T16x_selec_00* 变量的 *C 动作*。这对应于首次打开画面时所要显示的画面。
3	在画面中，组态一个*智能对象* → *画面窗口*。在本实例中，使用*画面窗口 2*。调整*画面窗口*的尺寸，使其与先前创建的画面相匹配。为使窗口在运行时带边框显示，将*属性* → *其它* → *边框*设置为*是*。在*属性* → *其它* → *画面名称*处，选择 *pictu_5_window_01.pdl* 并创建与变量 *T16x_selec_00* 的变量连接。
4	在同一画面中，组态一个 *Windows 对象* → *按钮*。在本实例中，*使用对象按钮_0*。在*事件* → *鼠标* → *按下左键*处，组态一个读取*按钮*的名称和编号并将名称分配给*内部变量 T16x_selec_00* 的 *C 动作*。
5	复制*按钮_0* 对象两次，并更改*按钮_1* 和*按钮_2* 的对象名。

步骤	过程：通过对象名称以及与画面名称显示的变量连接来打开画面
6	在画面中，在*画面窗口画面窗口 2* 上方组态一个*智能对象* → *静态文本*。在本实例中，使用对象*静态文本 4*。将*属性* → *字体* → *粗体字*设置为*是*。在*属性* → *字体* → *文本*处，从*静态*列中删除输入的文本，并创建一个与变量 *T16x_selec_00* 的*变量连接*。将更新设置为*一旦改变*。删除静态条目，以免建立画面时输出不正确的文本。 

打开画面的 C 动作

```
#include "apdefap.h"
void OnOpenPicture(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
SetTagChar("T16x_selec_00","pic_window_01.pdl");
}
```

- 通过*内部函数 SetTagChar* 分配画面名称。

按钮_0 的 C 动作

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
char name[30];
int number;
int  ch = '_';
char *pdest;

//check if object name contains character
pdest = strrchr( lpszObjectName, ch );
if ( pdest == NULL )(printf("ObjectNameErrcr"));
else {
   //read object number
   number = atoi(strrchr(lpszObjectName,'_')+1);
   //generate picture name
   sprintf(name,"pictu_5_window_%02d.PDL",number);
   //set tag which contains picture name
   SetTagChar("T16x_selec_00",name);
   }
}
```

- 声明*内部变量*。
- 检查对象是否已经按照约定的规则进行命名。对象被给予的名称是[按钮]+[_]+[要调用画面的编号]。
- 如果没有找到字符_，则输出一条错误信息。
- 读取按钮名称的编号。函数 *strrchr* 向后对名称进行搜索，以找到字符_。通过 *atoi* 函数将_后跟的字符串取出，并将其转换为一个整数值。
- *sprintf* 函数用画面名称和画面编号这两部分生成要通过*按钮*调用的完整画面名称。
- 通过*内部函数 SetTagChar*，把要调用的画面名称传送给 *T16x_selec_00* 变量。

常规应用的注意事项

在进行常规应用前，必须完成下列修改：

- *按钮*的 *C 动作*以及对象名的分配必须根据自己的命名约定进行修改。确保对象名和画面名都严格遵守其命名约定，从而保证访问所期望的画面时不会发生错误。

3.3 显示画面窗口

Project_CreatePicture 项目中与该主题相关的实例通过用选择如上所示的*按钮*来访问。这些实例均在 *pictu_3_chapter_03.pdl* 画面中组态。

3.3.1 从画面窗口外隐藏(撤消选择)和显示(选择)(实例 01)

任务定义

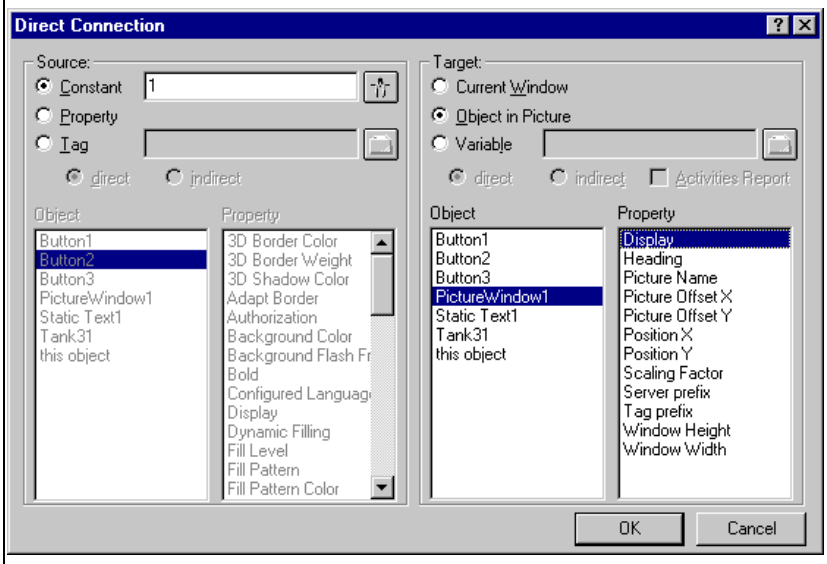
通过两个用鼠标操作的*按钮*来再次显示和隐藏画面窗口。

概念的实现

为实现上述任务，将使用两个 *Windows 对象* → *按钮*，当用按下它们时，将会显示和隐藏*智能对象* → *画面窗口*中的画面。

在 WinCC 项目中的实现

步骤	过程：从画面窗口外隐藏和显示
1	组态要显示和隐藏的画面，例如帮助文本或信息框。在本实例中，使用 *pictu_5_window_07*，即一个不带任何附加控制元素的纯信息框。
2	在另一个画面中，组态一个*智能对象* → *画面窗口*，其几何尺寸与先前创建的画面相同。在本实例中，使用对象*画面窗口 1*。将*属性* → *几何结构* → *宽度*设置为 *246*，并将*属性* → *几何结构* → *高度*设置为 *129*。为使窗口在运行时带边框显示，将*属性* → *其它* → *边框*设置为*是*。为了使窗口能够移动，将*属性* → *其它* → *可移动*设置为*是*。为了在运行时隐藏窗口，将*属性* → *其它* → *显示*设置为*否*。在*属性* → *其它* → *画面名称*处，选择画面 *pictu_5_window_07.pdl*。

步骤	过程：从画面窗口外隐藏和显示
3	在同一画面中，组态两个附加的 *Windows 对象* → *按钮*。在本实例中，它们是对象*按钮 1*和*按钮 2*。对于*按钮 1*，在*事件* → *鼠标* → *按下左键*处组态一个*直接连接*。将*源常数* → *1* 与*目标画面中的对象* → *画面窗口 1* → *显示*相连接。通过单击*确定*按钮即可应用这些设置。 
4	按照为*按钮 1*组态所采用的相同方法，在*事件* → *鼠标* → *按下左键*处为*按钮 2*组态一个*直接连接*。至于*常数*，则指定数值为 *0*。

常规应用的注意事项

在进行常规应用之前，必须完成下列修改：

- 对于对象*按钮 1* 和*按钮 2* 处的*直接连接*，必须修改要显示的画面名称以及*画面窗口*的对象名称。
- 所提供的画面 *pictu_5_window_07* 在修改其标题和信息文本之后可直接传送至另一个项目。

3.3.2 从画面窗口外显示(选择)和从画面窗口内隐藏(撤销选择)(实例 02)

任务定义

通过单击一个用鼠标操作的*按钮*使画面窗口变为可见。通过单击*画面窗口*内的按钮隐藏画面窗口。

概念的实现

为实现上述任务，将使用两个 *Windows 对象* → *按钮*，当用按下它们时，将会显示和隐藏*智能对象* → *画面窗口*中的画面。

在 WinCC 项目中的实现

步骤	过程：从画面窗口外显示(选择)和从画面窗口内隐藏(撤消选择)
1	组态要显示和隐藏的画面，例如帮助文本或信息框。在本实例中，使用画面 *pictu_5_window_08*，即一个具有用于撤销选择画面的附加 *Windows 对象* → *按钮*的信息框。在本实例中，使用对象*按钮 1*。
2	对于按钮 1，在*事件* → *鼠标* → *按下左键*处组态一个*直接连接*。将*源常数* → *0* 与目标*当前窗口* → *显示*相连接。 通过单击*确定*按钮即可应用这些设置。
3	在另一个画面中，组态一个*智能对象* → *画面窗口*，其几何尺寸与先前创建的画面相同。在本实例中，使用对象*画面窗口 2*。将*属性* → *几何结构* → *宽度*设置为 *246*，并将*属性* → *几何结构* → *高度*设置为 *129*。为使窗口在运行时带边框显示，将*属性* → *其它* → *边框*设置为*是*。为使窗口能够移动，将*属性* → *其它* → *可移动*设置为*是*。为了在运行时隐藏窗口，将*属性* → *其它* → *显示*设置为*否*。在*属性* → *其它* → *画面名称*处，选择画面 *pictu_5_window_08.pdl*。
4	在同一画面中，组态一个 *Windows 对象* → *按钮*。在本实例中，它是对象*按钮 3*。对于*按钮 3*，在*事件* → *鼠标* → *按下左键*处组态一个*直接连接*。将*源常数* → *1* 与*目标画面中的对象* → *画面窗口 2* → *显示*相连接。通过单击*确定*按钮即可应用这些设置。

常规应用的注意事项

在进行常规应用之前，必须完成下列修改：

- 对于*按钮 3* 对象的*直接连接*，必须修改所要显示的画面名称和*画面窗口*的对象名称。
- 所提供的 *pictu_5_window_08* 画面在修改其标题和信息文本之后可直接传送至另一个项目。不必对*按钮 1* 的*直接连接*进行任何修改。

3.3.3 对画面进行时控隐藏(实例 03)

任务定义

使用一个鼠标控制的*按钮*可显示和隐藏*画面窗口*。超过设置时间，画面窗口将自动隐藏。

概念的实现

为实现该任务，可使用一个 *Windows 对象* → *按钮*，当使用按下此按钮时，将显示和隐藏*智能对象* → *画面窗口*中的画面。

在 WinCC 项目中的实现

步骤	过程：对画面进行时控隐藏
1	组态将要显示和隐藏的画面，例如帮助文本或信息框。在本实例中，使用了 *pictu_5_window_09* 画面，它是一个不带任何其它控制元素的纯信息框。为了实现对*图形对象 1* 的时控隐藏，在*属性* → *几何结构* → *位置 X* 处组态一个 *C 动作*。该 *C 动作*可放置在任何地方，因为只需要一个*触发器*。将触发设置为 *1 秒*。 Event Name: 1 s
2	在另一个画面中，组态一个*智能对象* → *画面窗口*，它与先前所创建的画面具有相同的几何尺寸。在本实例中，使用了*画面窗口 3* 对象。将*属性* → *几何结构* → *宽度*设置为 *246*，将*属性* → *几何结构* → *高度*设置为 *129*。为使窗口在运行系统中带边框显示，可将*属性* → *其它* → *边框*设置为*是*。为使窗口能够移动，可将*属性* → *其它* → *可移动*设置为*是*。为了隐藏运行系统中的窗口，可将*属性* → *其它* → *显示*设置为*否*。在*属性* → *其它* → *画面名称*处，设置 *pictu_5_window_09.pdl* 画面。
3	组态一个按钮，在本实例中使用了*按钮 4* 对象。在*事件* → *鼠标* → *按下左键*处，为*按钮 4* 组态一个 *C 动作*，用于显示或隐藏*画面窗口*。

图形对象 1 处的 C 动作

```
#include "apdefap.h"
 long _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty

{
static int i = 0;

//count time
i++;
//if maximum time is reached
if (i>5) SetVisible("pictu_3_chapter_03.PDL","Picture Window3",0);
return 0;
}
```

按钮 4 处的 C 动作

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lps
{
//set visibility in complement state
SetVisible(lpszPictureName,"Picture Window3",
        |(SHORT)!GetVisible(lpszPictureName,"Picture Window3"));
}
```

- 通过*内部函数 SetVisible* 将*画面窗口 3* 的可见性分配为当前可见性的相反状态。当前状态则由*内部函数 GetVisible* 来进行查询。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 对于*按钮 4* 对象上的 *C 动作*，必须修改将要显示的画面名称和*画面窗口*的对象名称。
- 所提供的 *pictu_5_window_09* 画面，在对其标题和信息文本进行修改之后，可直接传送给另一个项目。在*图形对象 1* 的 *C 动作*处，隐藏画面的时间可通过修改触发器或修改 *if* 语句中的条件来进行用户定义。

3.3.4 在按下鼠标右键时对画面窗口进行显示(实例 04)

任务定义

在通过 R 按下*按钮*时，*画面窗口*将被显示，如果 R 被释放，则画面窗口将再次隐藏。

概念的实现

为实现该任务，可使用一个 *Windows 对象* → *按钮*，当使用 R 按下此按钮时，将使*智能对象* → *画面窗口*中的画面可见。

在 WinCC 项目中的实现

步骤	过程：在按下鼠标右键时对画面窗口进行显示
1	组态将要显示和隐藏的画面，例如帮助文本或信息框。在本实例中，使用了 *pictu_5_window_07* 画面，它是一个不带任何其它控制元素的纯信息框。
2	在另一个画面中，组态一个*智能对象* → *画面窗口*，它与先前创建的画面具有相同的几何尺寸。在本实例中，使用了*画面窗口 1* 对象。将*属性* → *几何结构* → *宽度*设置为 *246*，将*属性* → *几何结构* → *高度*设置为 *129*。为使窗口在运行时带边框显示，将*属性* → *其它* → *边框*设置为*是*。为使窗口能够移动，可将*属性* → *其它* → *可移动*设置为*是*。在*属性* → *其它* → *画面名称*上，设置 *pictu_5_window_07.pdl* 画面。
3	在同一画面中，组态一个 *Windows 对象* → *按钮*；在本实例中是*按钮 5* 对象。在*事件* → *鼠标* → *按下右键*处，为*按钮 5* 创建一个*直接连接*。将*源常数* → *1* 与*目标画面中的对象* → *画面窗口 4* → *显示*相连接。单击*确定*按钮即可应用这些设置。
4	采用同样的方式，在*事件* → *鼠标* → *释放右键*处创建一个*直接连接*。为常数指定数值 0。

常规应用的注意事项

在进行常规应用之前，必须进行下述修改：

- 对*按钮 5* 的*直接连接*，必须修改要显示的画面名称以及*画面窗口*的对象名称。
- 所提供的 *pictu_5_window_07* 画面，在对其标题和信息文本进行修改之后，可直接传送给另一个项目。

3.3.5 使用向导对信息框进行组态(实例 05)

通过使用 选择如上所示的按钮可访问本实例。该实例在 *pictu_3_chapter_03a.pdl* 画面中组态。

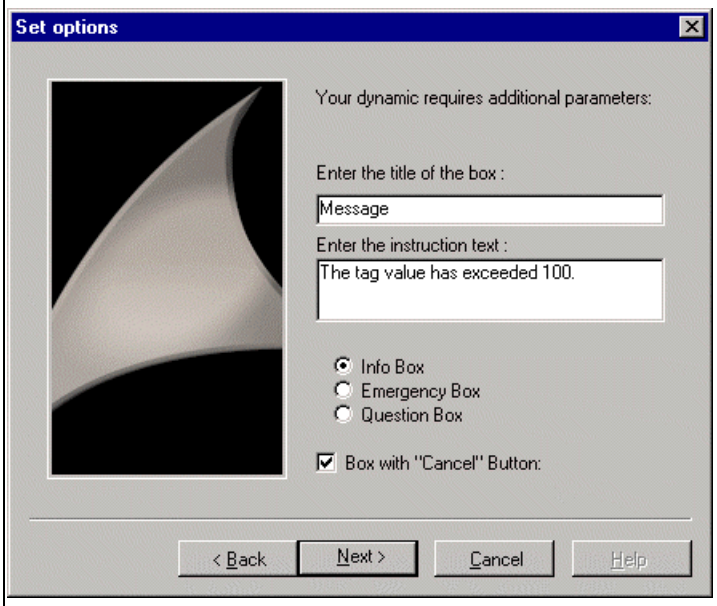
任务定义

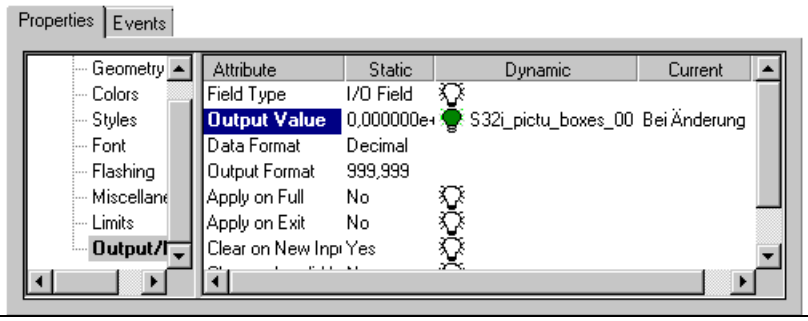
如果变量超过数值 100，则显示一个信息(指令)框；如果该变量值超过 150，则显示紧急框。

概念的实现

为实现该任务，可使用一个 *Windows 对象* → *滚动条对象*以输入变量值和*智能对象* → *I/O 域*，从而显示变量值。

在 WinCC 项目中的实现

步骤	过程：使用向导对信息框进行组态
1	如果没有显示*动态向导*，则从*查看* → *工具*栏里将其激活。
2	在画面中，对*智能对象* → *I/O 域*进行组态。在本实例中，使用了 *I/O 域 1* 对象。在将对象高亮度时，选择*画面函数*标签，然后从动态向导通过 D 来选择*显示信息框条目*。从*动态向导*的*选择触发器*页面中，选择*鼠标左键*列表条目，并通过单击*下一个*按钮对下一页继续操作。完成*设置选项*页面，如下所示： 

步骤	过程：使用向导对信息框进行组态
3	为 *I/O 域 1* 再次使用*动态向导*。在所选的触发页面上，选择*鼠标右键*；在*设置选项*页面，选择*紧急框*选项钮，并输入显示的文本。
4	在变量管理器中创建一个有符号的 32 位数类型的变量。在本实例中，使用了 S32i_pictu_boxes_00 变量。
5	在同一画面中，组态一个 *Windows 对象* → *滚动条滑块对象*。在本实例中是滚动条对象 1。在*事件* → *属性主题* → *其它* → *过程驱动程序连接*处为滚动条对象 1 创建一个*直接连接*。将*源属性* → *滚动条对象 1* → *过程驱动程序连接*与*目标变量* → *S32i_pictu_boxes_00* 相连接。单击*确定*按钮即可应用这些设置。
6	为 *I/O 域 1* 对象在*属性* → *输出/输入* → *输出值*上创建一个*变量连接*给变量 *S32i_pictu_boxes_00*，并一旦改变即进行触发。 
7	在*事件* → *属性主题* → *输出/输入* → *输出值*处为 *I/O 域 1* 对象创建一个 *C 动作*，如果 *S32i_pictu_boxes_00* 变量超出值 100，则它将显示一个信息框，如果超出 150，则显示一个紧急框。由*动态向导*在*事件* → *鼠标处* → *按下左键*和*按下右键*所产生的 *C 动作*可被复制和粘帖到该 *C 动作*中。
8	删除*事件* → *鼠标* → *按下左键*和*按下右键*处的 *C 动作*。

I/O 域 1 的 C 动作

```
#include "apdefap.h"
void OnPropertyChanged(char* lpszPictureName, char* lpszObjectName, char* l

{
int a;
static int i = 0, j = 0;

//get tag value
a=GetTagDWord("S32i_pictu_boxes_00");

//set visible info box
if ((a>100)&&(i==0)) {
        i=1;
        MessageBox(NULL,"Der Variablenwert hat\r\n100 überschritten",
        "Hinweis", MB_OK|MB_ICONEXCLAMATION|MB_SETFOREGROUND);
        }//if
if (a<=100) (i=0);

//set visible emergency box
if ((a>150)&&(j==0)) {
        j=1;
        MessageBox(NULL,"Der Variablenwert hat\r\n150 überschritten",
        "Achtung!!!", MB_OK|MB_ICONSTOP|MB_SETFOREGROUND);
        }//if
if (a<=150) (j=0);
}
```

- 使用*内部函数 GetTagDWord* 读入变量值。
- 如果变量值超出 100，则使用由*动态向导*所产生的 *C 动作*来显示信息框。如果超出 100，则仅将再次关闭信息框，如果低于 100，则*静态 C 变量 i* 复位为 0。
- 如果变量值超出 150，则使用由动态向导所产生的 C 动作来显示紧急框。如果超出 150，则仅将再次关闭紧急框，如果低于 150，则静态 C 变量 j 复位为 0。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 对于 *I/O 域 1* 对象的 *C 动作*，必须对变量名称进行修改。
- 必须根据需要对信息框和紧急框中所显示的文本进行修改。

3.3.6 显示用于文本输入的对话框(实例 06)

使用选择如上所示的按钮可访问本实例。该实例在 *pictu_3_chapter_03a.pdl* 画面中组态。

任务定义

当使用按下*按钮*时，将显示文本输入对话框。输入的文本显示在画面中。

概念的实现

为实现该任务，可使用一个 *Windows 对象* → *按钮*来打开对话框，使用*标准对象* → *静态文本*来显示文本。对于对话框中的文本输入，可使用一个*智能对象* → *I/O 域*和两个 *Windows 对象* → *按钮*来应用或取消输入。

在 WinCC 项目中的实现

步骤	过程：显示用于文本输入的对话框
1	在变量管理器中，创建两个*文本变量 16 位字符集*类型的变量。在本实例中，使用了 *T16i_pictu_win_00* 和 *T16i_pictu_win_01* 变量。
2	组态执行文本输入的画面。在本实例中，使用了 *pictu_5_window_17.pdl* 画面。
3	在该画面中，对*智能对象* → *I/O 域*进行组态。在其*组态对话框*中，选择 *T16i_pictu_win_01* 变量，并将触发器设置为*一旦改变*。将*属性* → *输出/输入* → *数据格式*设置为*字符串*，而将*属性* → *输出/输入* → *退出时应用*设置为*是*。也就是说，不必按下回车键就可以接受输入的文本。
4	在同一画面中，组态 *Windows 对象* → *按钮*。在本实例中，使用了*按钮 1* 对象。使用该按钮来应用所输入的文本。在*事件* → *鼠标* → *按下左键*处，组态一个*直接连接*，它具有*源变量 T16i_pictu_win_01* 和*目标变量 T16i_pictu_win_00*。在*事件* → *鼠标* → *鼠标动作*处，组态一个可隐藏画面的*直接连接*。
5	组态另一个 *Windows 对象* → *按钮*；本实例所组态的是*按钮 2* 对象。该按钮用来取消输入，保留先前输入的文本。在*事件* → *鼠标* → *按下左键*处，组态一个*直接连接*，它具有*源变量 T16i_pictu_win_00* 和*目标变量 T16i_pictu_win_01*。该直接连接将把 *T16i_pictu_win_00* 的内容(包含先前的文本)传送给 *T16i_pictu_win_01*。在*事件* → *鼠标* → *鼠标动作*处，组态一个可隐藏画面的*直接连接*。

步骤	过程：显示用于文本输入的对话框
6	在第二个画面中，组态一个*智能对象* → *画面窗口*。在本实例中，使用了*画面窗口 1* 对象。调整*画面窗口*的大小，使其与刚才创建的画面的大小相匹配。如果要使*画面窗口*带边框显示，则*画面窗口*的*高度*和*宽度*必须比一般画面要大 10 个像素。在*属性* → *其它* → *画面名称*下，输入 *pictu_5_window_17.pdl*。
7	在同一画面中，组态一个 *Windows 对象* → *按钮*。在本实例中，使用了*按钮 1* 对象。在*事件* → *鼠标* → *按下左键*处，创建一个*直接连接*。将*源常数* → *1* 与目标*画面中的对象* → *画面窗口 1* → *显示*相连接。单击*确定*按钮即可应用这些设置。
8	在同一画面中，组态一个*标准对象* → *静态文本*。在本实例中，使用了*静态文本 1* 对象。在*属性* → *字体* → *文本*处，将*变量连接*组态给 *T16i_pictu_win_00* 变量，并一旦改变将其触发。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- *pictu_5_window_17.pdl* 画面可直接用于文本输入，然而，必须对*按钮*处的*C 动作*进行修改，以便与自己的变量名称相匹配。

3.4 操作控制允许

通过使用选择如上所示的*按钮*可访问 *Project_CreatePicture* 项目中与该主题相关的解决方案。这些实例均在 *pictu_3_chapter_02.pdl* 画面中组态。

3.4.1 退出运行系统和系统(实例 01)

任务定义

使用两个鼠标控制的*按钮*来选择两个控制窗口，用于从运行系统或整个系统退出。

概念的实现

为实现该任务，可使用两个 *Windows 对象* → *按钮*，当使用将其按下时，每个按钮都将在*智能对象* → *画面窗口*中显示一个画面。在各个画面中，两个 *Windows 对象* → *按钮*既可调用相应的系统函数又可取消过程。

在 WinCC 项目中的实现

步骤	过程：退出运行系统和系统
1	组态一个画面用于退出运行系统。在本实例中，使用了 *pictu_5_window_04.pdl* 画面。
2	在该画面中，将组态一个 *Windows 对象* → *按钮*；在实例中，使用了*按钮 1* 对象。在将对象高亮度时，选择*系统函数*标签，然后从*动态向导*通过D 选择*退出 WinCC* 或 *Windows* 条目。从*动态向导*的*选择触发*页面中，选择*鼠标左键*条目，然后通过单击*下一步按钮*对下一页继续操作。在*设置选项*页面上，选择*退出 Windows*。确认*完成*！页面可通过单击*完成*按钮。
3	组态另一个 *Windows 对象* → *按钮*。在本实例中，使用了*按钮 2* 对象。该按钮用于取消过程。在*事件* → *鼠标* → *按下左键*处，组态一个可隐藏画面的*直接连接*。
4	组态另一个画面用于关闭系统。在本实例中，使用了 *pictu_5_window_03.pdl* 画面。
5	在该画面中，组态一个 *Windows 对象* → *按钮*。在本实例中，使用了*按钮 1* 对象。在将对象高亮度时，选择*系统函数*标签，然后从*动态向导*通过D 选择*退出 WinCC 运行系统*条目。从*动态向导*的*选择触发*页面中，选择*鼠标左键*条目，然后通过单击*下一步*按钮对下一页继续操作。确认*完成*！页面可通过单击*完成*按钮。

步骤	过程：退出运行系统和系统
6	组态另一个 *Windows 对象* → *按钮*。在本实例中，使用了*按钮 2* 对象。该按钮用于取消过程。在*事件* → *鼠标* → *按下左键*处，组态一个可隐藏画面的*直接连接*。
7	在另一个画面中，组态两个*智能对象* → *画面窗口*；在该画面中，使用了*画面窗口 1* 和*画面窗口 2* 对象来叠加排列。调整*画面窗口*的大小，使其与刚才创建的画面大小相匹配。如果要使*画面窗口*带边框显示，则必须将*画面窗口*的*高度*和*宽度*设置为比一般画面多 10 个像素，以便能够显示整个画面。在*属性* → *其它* → *画面名称*处，输入各个画面名称。将*属性* → *其它* → *显示*设置为*否*。
8	在同一画面中，组态两个 *Windows 对象* → *按钮*。在本实例中，它们是*按钮 1* 和*按钮 2* 对象。在*事件* → *鼠标* → *按下左键*处为*按钮 1* 创建一个*直接连接*。将*源常数* → *1* 与*目标画面中的对象* → *画面窗口 1* → *显示*相连接。单击*确定*按钮即可应用这些设置。用同样的方法为*按钮 2 创建一个直接连接，*但是要将其设置为*目标画面中的对象* → 画面窗口 2 → 显示。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 用于退出系统和运行系统的画面能直接应用于其它项目。
- 在用于调用*画面窗口*的*按钮*处，必须对*直接连接*中的*画面窗口*的对象名称进行修改。

3.4.2 操作员控制允许、使用缺省框进行登录(实例 02)

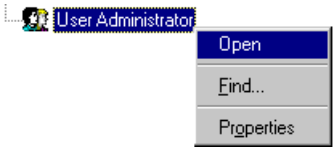
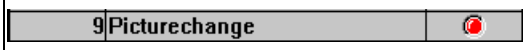
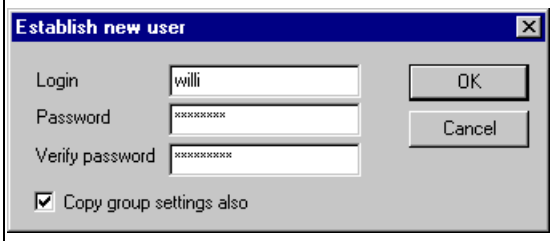
任务定义

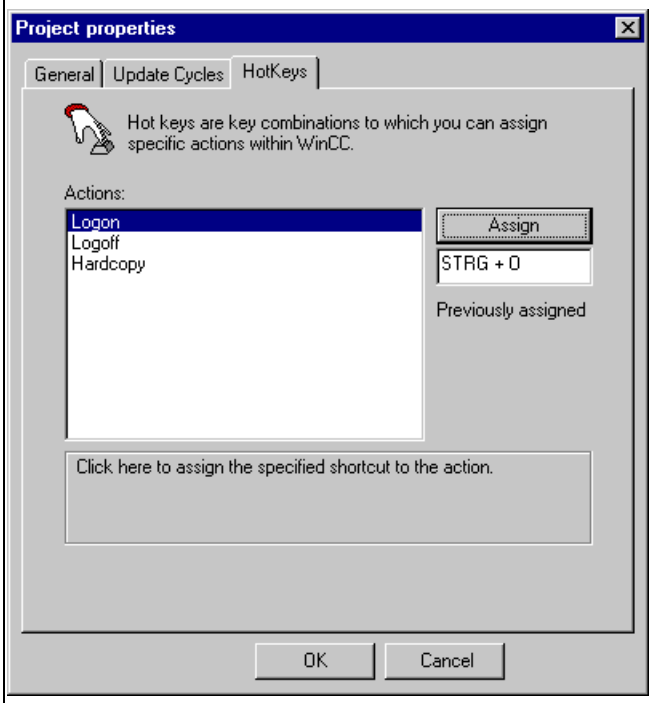
通过两个*按钮*，只有当用户具有相应的授权时，才执行画面切换。

概念的实现

为了实现该概念，将使用两个 *Windows 对象* → *按钮*，当使用将按钮按下时，每个按钮可在*智能对象* → *画面窗口*中显示不同画面。在*用户管理器编辑器*中，可完成对用户权限进行分配所需的设置。

在 WinCC 项目中的实现

步骤	过程：操作员控制允许、使用缺省框进行登录
1	在 *WinCC 资源管理器*中，通过R 然后从弹出式菜单中选择打开将*用户管理器编辑器*打开。 
2	通过按钮，创建一个新的用户组，并为其分配一个名称；在本实例中，将使用名称 service。
3	通过*表* → *添加新的授权等级*菜单，可在第 9 行上定义授权*等级画面切换*。该授权等级将分配给 *service* 组。为此，选择带有的组。在包含*画面切换*行的表中，用D，在*授权*列中选择选项钮。 分配给组或用户的授权等级由靠近*授权*列的红点来识别。 
4	通过*按钮* ，为 *service* 用户组创建一个新的用户。在同一项目中，已创建了一个名称为 *willi* 的用户，该用户带有口令 *Project_CreatePicture*。激活*同时复制组*设置复选框，把可用于这个组的授权等级传送给用户。  通过*文件* → *退出*菜单，可关闭*用户管理器*编辑器。

步骤	过程：操作员控制允许、使用缺省框进行登录
5	在 WinCC 资源管理器中，通过在项目名称上的R 访问项目属性对话框。 在所显示的窗口中，选择热键标签，并完成用于调用登录和退出对话框的期望设置。为了对热键进行分配，可使用，并单击分配按钮。在本实例中，使用组合键 CTRL+O 用于登录，使用 CTRL+F 用于退出。 
6	在某个画面中，组态了两个 *Windows 对象* → *按钮*。在本实例中，使用了对象*按钮 3* 和*按钮 4*。 组态一个*智能对象* → *画面窗口*，通过两个按钮处的*直接连接*可将画面插入到窗口中。
7	对于对象*按钮 3* 与*按钮 4*，选择*画面切换*用户级(位于*属性* → *其它* → *用户级上*)，并将*属性* → *其它* → *操作员控制允许*设置为*否*。
8	在将*按钮 3 对象加亮时*，通过从动态向导中D，*选择标准动态*标签，然后再选择*经授权方可操作*条目。单击*完成*按钮，即可完成*动态向导*。对*按钮 4* 重复同样过程。
9	在变量管理器中，创建*文本变量 16 位字符集*类型的*@CurrentUser* 系统变量。将当前注册的用户名自动分配给该变量。
10	触发*按钮 3* 与*按钮 4* 处的 *C 动作*，该 C 动作是在一旦改变该变量时由*动态向导*所产生的。这意味着 *C 动作*将不再每 2 秒钟执行一次，而是只有在用户名发生变化后才执行。

由动态向导生成的 C 动作

```
#include "apdefap.h"
BOOL _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyN

{
#pragma code ("UseAdmin.DLL")
#include "pwrt_api.h"
#pragma code ()
#define NO_MESSAGEBOX  1
CMN_ERROR err;
DWORD pwlevel = 0;
pwlevel = (DWORD) GetPasswordLevel(lpszPictureName,lpszObjectName);
if (pwlevel==0)
return (TRUE);
else
return(PWRTCheckPermissionOnPicture(pwlevel,lpszPictureName,NO_MESSAGEBOX,&

}
```

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

必须修改用户组和用户的名称、登录名称以及用户口令。

3.4.3 通过单独对话框进行的操作员控制允许、登录(实例 03)

任务定义

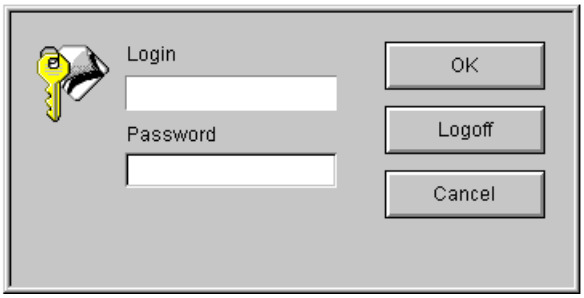
如果用户具有相应的授权，通过按钮才能退出运行系统。通过按钮，还将显示用于进行登录的对话框。

概念的实现

为了实现该概念，可使用两个 *Windows 对象* → *按钮*。对于第一个按钮，在用将其按下时，将显示一个用于登录的*智能对象* → *画面窗口*。第二个按钮则用于退出运行系统。

在 WinCC 项目中的实现

步骤	过程：通过单独对话框进行的操作员控制允许、登录
1	在*用户管理器*编辑器中，创建一个新的用户组并为其命名；在本实例中，使用了名称 user。在第 10 行，定义了一个称为退出运行系统的新授权等级。该授权等级将分配给刚才创建的用户组。为该组创建用户。 在实例项目中，已创建了一个名为 *ulrich* 的用户，该用户带有口令 *Project_CreatePicture*。
2	在一个画面中，组态了两个 *Windows 对象* → *按钮*。在本实例中，使用了对象*按钮 5* 和*按钮 6*。
3	对于*按钮 5*，可对调用用于终止运行时的*智能对象* → *画面窗口*进行组态。在本实例中，使用了对象*画面窗口画面窗口 5*。
4	对于按钮 5 对象，选择退出运行系统用户级(位于属性 → 其它 → 用户级处)，并将属性 → 其它 → 操作员控制允许设置为否。
5	将经*授权方可操作*的动态向导应用于*按钮 5*。将所产生的 C 动作设置为由 *@CurrentUser* 系统变量来触发。

步骤	过程：通过单独对话框进行的操作员控制允许、登录
6	组态一个*智能对象* → *画面窗口*。在本实例中，使用了*画面窗口 4* 对象。将*属性* → *几何结构* → *窗口宽度*设置为 *360*，将*属性* → *几何结构* → *窗口高度*设置为 *180*。将*属性* → *其它可移动的*、*边框*、*标题与前景*均设置为*是*。在*属性* → *其它* → *画面名称*处，选择 *pictu_5_window_18.pdl* 画面。可直接从 *Project_CreatePicture* 项目中获取该画面。 
7	对于*按钮 6* 对象，创建一个*直接连接*用于显示刚才组态的*画面窗口*。
8	在*按钮 6* 对象处，组态一个 *C 动作*，用来根据用户注册与否，将文本分配给按钮标签。该 *C 动作*也可由*@CurrentUser* 变量来触发。

按钮 6 的 C 动作

```
#include "apdefap.h"
 char* _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropert

{
if (strcmp(GetTagChar("@CurrentUser"),""))
        return "Logoff";
else    return "Logon";
}
```

- 如果*@CurrentUser* 变量包含了一个名称，也就是说，如两个文本的比较结果是 *TRUE*，则返回文本*退出*，反之，则返回文本*登录*。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 必须修改用户组和用户的名称、登录名称以及用户口令。

3.5 画面缩放

访问 *Project_CreatePicture* 项目中与该主题相关的实例，可使用，选择上面显示的*按钮*来完成。这些实例均组态在 *pictu_3_chapter_04.pdl* 画面中。

3.5.1 在两种尺寸之间改变画面几何结构(实例 01)

任务定义

通过两个用鼠标操作的按钮再次显示和隐藏画面窗口。打开画面窗口时显示小画面。通过另一个按钮，可调整画面尺寸。

概念的实现

为了实现上述任务，将使用两个 *Windows 对象* → *按钮*，当用将按钮按下时，可显示和隐藏*智能对象* → *画面窗口*中的画面。两个附加的 *Windows 对象* → *按钮*可对画面进行放大和缩小。

在 WinCC 项目中的实现

步骤	过程：在两种尺寸之间改变画面几何结构
1	组态要显示和隐藏的画面。在本实例中，使用画面 *pictu_3_chapter_00* (画面项目 *Project_CreatePicture* 的启动画面)。
2	在另一个画面中，组态一个*智能对象* → *画面窗口*；在本实例中，它就是*画面窗口 1*。将*属性* → *几何结构* → *宽度*设置为 *172*，并将*属性* → *几何结构* → *高度*设置为 *140*。将*属性* → *其它* → *边框*设置为*是*，并将*属性* → *其它* → *适应画面*大小设置为*是*。这样，可使几何尺寸为 859*698 的画面适应画面窗口的尺寸。在*属性* → *其它* → *画面名称*处，选择画面 *pictu_3_window_00.pdl*。将*属性* → *其它* → *显示*设置为*否*。
3	在同一画面中，组态两个附加的 *Windows 对象* → *按钮*。在本实例中，它们是对象*按钮 1* 和*按钮 2*。对于*按钮 1*，在*事件* → *鼠标* → *按下左键*处创建一个*直接连接*。将*源常数* → *1* 与*目标画面中的对象* → *画面窗口 1* → *显示*相连接。通过单击确定按钮即可应用这些设置。

步骤	过程：在两种尺寸之间改变画面几何结构
4	组态两个附加的 *Windows 对象* → *按钮*。在本实例中，它们是对象*按钮 3* 和*按钮 4*。对于*按钮 3*，在*事件* → *鼠标* → *按下左键*处创建一个 *C 动作*，用于放大*画面窗口*、隐藏*按钮 3* 以及显示*按钮 4*。对于*按钮 4*，同样在*事件* → *鼠标* → *按下左键*处创建一个 *C 动作*，用于缩小*画面窗口*、隐藏*按钮 4* 以及显示*按钮 3*。将两个*按钮*的*属性* → *其它* → *显示*均设置为*否*。
5	对于*按钮 1*，在*事件* → *鼠标* → *鼠标动作*处创建一个*直接连接*。将*源常数* → *1* 与*目标画面中的对象* → *按钮 3* → *显示*相连接。 通过单击*确定*按钮即可应用这些设置。对于*按钮 2*，在*事件* → *鼠标* → *按下左键*处组态一个 *C 动作*，用于隐藏*按钮 3* 和*按钮 4*、缩小*画面窗口 1* 画面窗口的尺寸然后隐藏*画面窗口*。
6	将*按钮 3* 与*按钮 4* 重叠放置在一起。

按钮 3 的 C 动作

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
SetHeight(lpszPictureName,"Picture Window1",420);
SetWidth(lpszPictureName,"Picture Window1",516);
SetVisible(lpszPictureName,"Button3",0);
SetVisible(lpszPictureName,"Button4",1);
}
```

- 通过*内部函数 SetHeight* 和 *SetWidth* 更改*画面窗口 1* 的宽度与高度。
- 隐藏放大*按钮(按钮 3)*。
- 显示缩小*按钮(按钮 4)*。

按钮 4 的 C 动作

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
SetHeight(lpszPictureName,"Picture Window1",140);
SetWidth(lpszPictureName,"Picture Window1",172);
SetVisible(lpszPictureName,"Button3",1);
SetVisible(lpszPictureName,"Button4",0);
}
```

- 通过*内部函数 SetHeight* 和 *SetWidth* 更改*画面窗口 1* 的宽度与高度。
- 显示放大*按钮(按钮 3)*。
- 隐藏缩小*按钮(按钮 4)*。

按钮 2 的 C 动作

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
SetVisible(lpszPictureName,"Button3",0);
SetVisible(lpszPictureName,"Button4",0);
SetHeight(lpszPictureName,"Picture Window1",140);
SetWidth(lpszPictureName,"Picture Window1",172);
SetVisible(lpszPictureName,"Picture Window1",0);
}
```

- 隐藏放大按钮(*按钮 3*)和缩小按钮(*按钮 4*)。
- 通过*内部函数 SetHeight* 和 *SetWidth* 更改*画面窗口画面窗口 1* 的高度与宽度。
- 隐藏*画面窗口画面窗口 1*。

常规应用的注意事项

在进行常规应用之前，必须完成下列修改：

- 对于对象*按钮 1* 的*直接连接*，必须修改对象名称。
- 对于对象*按钮 2*、*按钮 3* 和*按钮 4* 的 *C 动作*，必须修改对象名称和要设置的画面尺寸。

3.5.2 连续更改画面几何结构(实例 02)

任务定义

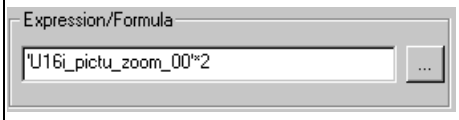
通过两个用鼠标操作的按钮可显示和隐藏画面窗口。此外，通过滚动条对象可连续调整画面尺寸。

概念的实现

为了实现上述任务，将使用两个 *Windows 对象* → *按钮*，以便用将按钮按下时，可显示和隐藏*智能对象* → *画面窗口*中的画面；并将使用一个 *Windows 对象* → *滚动条对象*以便更改画面尺寸。

在 WinCC 项目中的实现

步骤	过程：连续更改画面几何结构
1	组态要显示和隐藏的画面。在本实例中，使用画面 *pictu_5_window_10.pdl*，其宽度与高度之比为 2:1。
2	在另一个画面中，组态一个*智能对象* → *画面窗口*；在本实例中，它是*画面窗口 2*。将*属性* → *几何结构* → *宽度*设置为 *160*，并将*属性* → *几何结构* → *高度*设置为 *80*(宽度:高度也是 2:1)。为使窗口在运行时带边框显示，将*属性* → *其它* → *边框*设置为*是*，并将*属性* → *其它* → *适应画面大小*设置为*是*。这样就可使画面适应*画面窗口*的尺寸。在*属性* → *其它* → *画面名称*处，选择画面 *pictu_5_window_10.pdl*。将*属性* → *其它* → *显示设置*为*否*。
3	在同一画面中，组态两个附加的 Windows 对象 → 按钮。在本实例中，它们是对象*按钮 5*和*按钮 6*。对于*按钮 5*，在*事件* → *鼠标* → *按下左键*处创建一个*直接连接*。将*源常数* → *1*与*目标画面中的对象* → *画面窗口 2* → *显示*相连接。 通过单击*确定*按钮即可应用这些设置。
4	采用同样的方法，在*事件* → *鼠标* → *按下左键*处，为*按钮 6* 创建一个*直接连接*。至于*常数*，则指定数值 *0*。
5	在变量管理器中，创建一个*无符号的 16 位数*类型的变量。在本实例中，使用 *U16i_pictu_zoom_00* 变量。
6	组态一个 *Windows 对象* → *滚动条对象*。在本实例中，它是*滚动条对象 1*。将*属性* → *其它* → *最大值*设置为 *300*。将*属性* → *其它* → *过程驱动程序连接*设置为 80。在*事件* → *属性主题* → *其它* → *过程驱动程序连接*处，创建一个*直接连接*。将*源属性* → *本对象* → *过程驱动程序连接*与*目标变量* → *U16i_pictu_zoom_00* 相连接。 通过单击*确定*按钮即可应用这些设置。
7	对于对象*画面窗口 2*，在*属性* → *几何结构* → *窗口高度*处创建一个*动态对话框*。使用*按钮* ... 来选择*变量* → *U16i_pictu_zoom_00*。在*更改触发器*对话框中使用按钮 将变量 *U16i_pictu_zoom_00* 确认为触发器名称，并将标准周期设置为*一旦改变*。通过单击*确定*按钮确认所作的设置。在*数据类型*域中，选择*直接*选项钮，并通过单击*应用*按钮退出*动态对话框*。

步骤	过程：连续更改画面几何结构
8	对于对象*画面窗口 2*，在*属性* → *几何结构* → *窗口高度处*创建一个*动态对话框*。可根据上面的描述进行设置，但是必须按如下所示完成*表达式/公式*域：  这将使窗口宽度的值是窗口高度的两倍。
9	对于画面对象 *pictu_3_chapter_04*，在*事件* → *其它* → *打开画面*处组态一个 *C 动作*，用于在打开画面时，将变量 *U16i_pictu_zoom_00* 设置为 *80*。如果不进行这种初始化，则直到首次激活*滚动条对象 1* 为止变量值都将保持为 0。如果随后按下按钮 5 对象，则*画面窗口 2* 将会以尺寸 0x0 显示。

打开画面的 C 动作

```
#include "apdefap.h"
void OnOpenPicture(char* lpszPictureName, char* lpszObjectName, char* lpszPa

{
//init tag
SetTagWord("U16i_pictu_zoom_00",80);
}
```

- 将变量 U16i_pictu_zoom_00 设置为 80。

常规应用的注意事项

在进行常规应用之前，必须完成下列修改：

- 对于*滚动条对象 1* 的*直接连接*，必须修改变量名称。
- 对于对象*画面窗口 2* 的*动态对话框*，必须修改变量名称。乘数须符合使用的宽:高比例。

3.5.3 通过属性对话框组态可调整的画面几何结构(实例 03)

任务定义

使用鼠标可将*画面窗口*拖放到任意大小。此外，画面可以移至屏幕的任何位置。通过*按钮*可对画面进行最大化和隐藏。

概念的实现

为了实现该概念，可使用两个 *Windows 对象* → *按钮*，当使用将按钮按下时，可显示和隐藏*智能对象* → *画面窗口*中的画面。必需的画面属性在属性对话框中进行组态。

在 WinCC 项目中的实现

步骤	过程：通过属性对话框组态可调整的画面几何结构
1	组态要显示和隐藏的画面。在本实例中，使用了 pictu_3_chapter_00 画面(画面项目 *Project_CreatePicture* 的起始画面)。
2	在另一个画面中，组态一个*智能对象* → *画面窗口*；在本实例中，它就是*画面窗口 3*。将*属性* → *几何结构* → *宽度*设置为 *147*，将*属性* → *几何结构* → *高度*设置为 *140*。在*属性* → *其它*处，将属性*可缩放*、*可移动*、*边框*、*标题*、*可最大化*、*适应画面大小*以及*可关闭*均设置为*是*。在*属性* → *其它* → *画面名称*处，选择 *pictu_3_chapter_00* 画面。将*属性* → *其它* → *显示*设置为*否*。
3	在同一画面中，组态两个 *Windows 对象* → *按钮*。在本实例中，使用了对象*按钮 7*和*按钮 8*。对于*按钮 7*，可在*事件* → *鼠标* → *按下左键*处创建一个*直接连接*。将*源常数* → *1* 与*目标画面中的对象* → *画面窗口 3* → *显示*相连接。单击*确定*按钮即可应用这些设置。
4	采用同样的方法，可创建一个*直接连接*，用于*事件* → *鼠标* → *按下左键*处的*按钮 8*。然而，输入数值 *0* 作为*常数*。

常规应用的注意事项

在进行常规应用前，必须完成下述修改：

- 对于*按钮 7* 和*按钮 8* 对象处的*直接连接*，必须对将要显示的画面名称以及*画面窗口*的对象名称进行修改。
- 必须对显示在*画面窗口画面窗口 3* 中的画面进行修改。

3.6 Windows 控制中心

Operator Panels

访问 *Project_CreatePicture* 项目中与该主题相关的实例，可通过使用，选择上面显示的*按钮*来完成。这些实例均组态在 *pictu_3_chapter_05.pdl* 画面中。

3.6.1 二进制切换操作(两步控制)(实例 01)

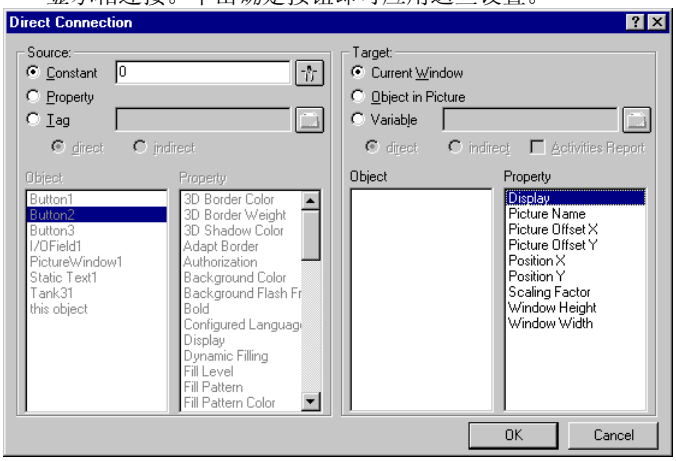
任务定义

通过一个鼠标控制*按钮*对操作面板进行访问。操作面板将包含一个用于打开和关闭阀门的*按钮*和另一个用于关闭面板的*按钮*。

概念的实现

为了实现该任务，可使用一个 *Windows 对象* → *按钮*，当使用按下时，将在*智能对象* → *画面窗口*中显示画面，还可另外使用两个按钮来切换操作和关闭面板。

在 WinCC 项目中的实现

步骤	过程：二进制切换操作(两步控制)
1	在变量管理器中，创建一个*二进制变量*类型的变量。在本实例中，使用了 *BINi_pictu_input_00* 变量。该变量包含数值的当前状态。
2	组态一个具有两个 *Windows 对象* → *按钮*的画面。在本实例中，使用了 *pictu_5_window_11* 画面，它包含了*按钮 1* 和*按钮 2* 对象。可在*事件* → *鼠标* → *按下左键*处为按钮 1 创建一个*直接连接*。将*源常数* → *0* 与目标*当前窗口* → *显示*相连接。单击*确定*按钮即可应用这些设置。 

步骤	过程：二进制切换操作(两步控制)
3	在本实例中的第二个*按钮-按钮 2* 处，组态一个 *C 动作*，用于对二进制变量 *BINi_pictu_input_00* 的状态求反。
4	在另一个画面中，组态一个*智能对象* → *画面窗口*；在本实例中是*画面窗口 1*。将*属性* → *几何结构* → *宽度*设置为 *246*，将*属性* → *几何结构* → *高度*设置为 *129*。为了使窗口在运行期间带边框显示并可移动，将*属性* → *其它* → *边框*设置为*是*，将*属性* → *其它* → *可移动*设置为*是*。在*属性* → *其它* → *画面名称*处，选择 *pictu_5_window_11.pdl* 画面。将*属性* → *其它* → *显示*设置为*否*。
5	在同一画面中，组态一个 *Windows 对象* → *按钮*。在本实例中，这是 *pictu_3_chapter_05.pd 画面中的按钮 1 对象*。*可在事件* → *鼠标* → *按下左键处为按钮 1 创建一个*直接连接。将*源常数* → *1* 与目标*画面中的对象* → *画面窗口 1* → *显示*相连接。单击*确定*按钮即可应用这些设置。

按钮 2 的 C 动作

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
SetTagBit("BINi_pictu_input_00", (SHORT)!GetTagBit("BINi_pictu_input_00"));
}
```

- 可通过*内部函数 GetTagBit* 读取 *BINi_pictu_input_00* 变量的状态并对其求反，然后通过*内部函数 SetTagBit* 进行重新设置。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 对于*按钮 1* 处的*直接连接*，必须对将打开的*画面窗口*的对象名称进行修改。
- 对于操作面板中在*按钮 2* 处的 *C 动作*，必须对变量名称进行修改。

3.6.2 二进制 S-R 切换操作(两步控制)(实例 02)

任务定义

通过一个鼠标控制*按钮*可对操作面板进行访问。该操作面板将包含一个用于打开阀的按钮和另一个用于关闭阀的按钮。通过单击另一个按钮可关闭面板自身。

概念的实现

为了实现该任务，可使用一个 *Windows 对象* → *按钮*，*当使用* *按下时*，*将显示智能对象* → *画面窗口中的画面*，还可*另外使用三个按钮*来切换操作和关闭面板。

在 WinCC 项目中的实现

步骤	过程：二进制 S-R 切换操作(两步控制)
1	在变量管理器中，创建一个*二进制变量*类型的变量。在本实例中，使用了 *BINi_pictu_input_01* 变量。该变量包含数值的当前状态。
2	组态一个具有三个 *Windows 对象* → *按钮*的画面。在本实例中，使用了 *pictu_5_window_12* 画面，它包含了*按钮 1*、*按钮 2* 和*按钮 3* 对象。在*事件* → *鼠标* → *按下左键*处为*按钮 1* 创建一个*直接连接*。将*源常数* → *0* 与目标*当前窗口* → *显示*相连接。 单击确定按钮即可应用这些设置。
3	可在*事件* → *鼠标* → *按下左键*处为*按钮 2* 创建一个*直接连接*。将*源常数* → *1* 与*目标变量* → *BINi_pictu_input_01* 相连接。 单击*确定*按钮即可应用这些设置。
4	采用同样的方法，可在*事件* → *鼠标* → *按下左键*处为*按钮 3* 创建一个*直接连接*。为常数指定数值 *0*。
5	在另一个画面中，组态一个*智能对象* → *画面窗口*；在本实例中是*画面窗口 2*。将*属性* → *几何结构* → *宽度*设置为 *246*，将*属性* → *几何结构* → *高度*设置为 *129*。为了使窗口在运行期间带边框显示并可移动，将*属性* → *其它* → *边框*设置为*是*，将*属性* → *其它* → *可移动*设置为*是*。在*属性* → *其它* → *画面名称*处，选择 *pictu_3_window_12.pdl* 画面。
6	在同一画面中，组态 *Windows 对象* → *按钮*。在本实例中，这是 *pictu_3_chapter_05.pd* 画面中的*按钮 2* 对象。可在*事件* → *鼠标* → *按下左键*处为*按钮 2* 创建一个*直接连接*。将*源常数* → *1* 与*目标画面中的对象* → *画面窗口 2* → *显示*相连接。 单击*确定*按钮即可应用这些设置。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 对于*按钮 2* 处的*直接连接*，必须对将打开的*画面窗口*的对象名称进行修改。
- 对于操作面板中的*按钮 1* 和*按钮 2* 处的*直接连接*，必须对变量名称进行修改。

3.6.3 确认的二进制切换操作(实例 03)

任务定义

通过一个鼠标控制*按钮*对操作面板进行访问。该操作面板将包含一个*按钮*用来打开和关闭阀。实际的切换操作只有在单击一个独立的*确定*按钮之后才生效，同时也关闭了操作面板。

概念的实现

为了实现该任务，可使用一个 *Windows 对象* → *按钮*，当使用按下时，将在*智能对象* → *画面窗口*中显示画面，还可另外使用两个按钮来切换操作和关闭面板。

在 WinCC 中的实现

步骤	过程：确认的二进制切换操作
1	在变量管理器中创建两个*二进制变量*类型的变量。在本实例中，使用了 *BINi_pictu_input_02* 和 *BINi_pictu_input_03* 变量。*BINi_pictu_input_02* 包含阀的当前状态，*BINi_pictu_input_03* 用作缓冲器，在进行确认前可对操作进行切换。
2	组态一个具有两个 *Windows 对象* → *按钮*的画面。在本实例中，使用了 *pictu_5_window_13.pdl* 画面，它包含了*按钮 1* 与*按钮 2* 对象。可在*事件* → *鼠标动作*处为*按钮 1* 创建一个*直接连接*。将*源常数* → *0* 与目标*当前窗口* → *显示*相连接。 单击*确定*按钮即可应用这些设置。 在*事件* → *鼠标* → *按下左键*处，组态另一个*直接连接*。将源 *BINi_pictu_input_02* 与目标 *BINi_pictu_input_03* 相连接。 单击*确定*按钮即可应用这些设置。
3	对本实例中的第二个按钮*按钮 2*，可组态一个 *C 动作*，用于对二进制变量 *BINi_pictu_input_02* 的状态求反。
4	在本实例中的另一个 pictu_3_chapter_05.pdl 画面中，组态一个*智能对象* → *画面窗口*。在本实例中，这是*画面窗口 3 对象*。将*属性* → *几何结构* → *宽度*设置为 *246*，将*属性* → *几何结构* → *高度*设置为 *129*。为了使窗口在运行期间带边框显示并可移动，将*属性* → *其它* → *边框*设置为*是*，将*属性* → *其它* → *可移动*设置为*是*。在*属性* → *其它* → *画面名称*处，选择 *pictu_3_window_13.pdl* 画面。
5	在同一画面中，组态 *Windows 对象* → *按钮*。在本实例中，这是 *pictu_3_chapter_05.pdl* 画面中的*按钮 3* 对象。 在*事件* → *鼠标* → *按下左键*处为*按钮 3* 创建一个*直接连接*。将*源常数* → *1* 与目标*画面中的对象* → *画面窗口 3* → *显示*相连接。单击*确定*按钮即可应用这些设置。

按钮 2 的 C 动作

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
SetTagBit("BINi_pictu_input_02",(SHORT)!GetTagBit("BINi_pictu_input_02"));
}
```

- 通过*内部函数 GetTagBit* 读取 *BINi_pictu_input_02* 变量的状态并对其求反，然后通过*内部函数 SetTagBit* 进行重新设置。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 对于*按钮 3* 处的*直接连接*，必须对将打开的*画面窗口*的对象名称进行修改。
- 对于操作面板中的*按钮 1* 处的*直接连接*，必须对变量名称进行修改。
- 对于操作面板中*按钮 2* 处的 *C 动作*，必须对变量名称进行修改。

3.6.4 自动输入检查(实例 04)

通过用选择如上所示的按钮可在 Project_CreatePicture 项目中访问与该主题相关的实例。这些实例均组态在 pictu_3_chapter_05a.pdl 画面中。

任务定义

将通过鼠标操作的*按钮*来访问操作面板。该操作面板用来把一定量的液体装入容器，液体数量值也将被输入面板。自动检查所输入的数值，确定是否超过容器的最大容量。

概念的实现

为实现该任务，可使用 *Windows 对象* → *按钮*，当用按下按钮时，将在*智能对象* → *画面窗口*中显示画面。此外，还将使用*三个 Windows 对象* → *按钮*来打开或关闭阀以及关闭操作面板。将使用*智能对象* → *I/O 域*来输入填充量。

在 WinCC 项目中的实现

步骤	过程：自动输入检查
1	在变量管理器中，创建一个*二进制变量*类型的变量，它包含阀的当前状态。在本实例中，使用变量 BINi_pictu_input_06。
2	创建两个*无符号的 16 位数*类型的变量。在本实例中，是变量 *U16i_pictu_input_04* 和 *U16i_pictu_input_05*。其中第一个变量包含容器填充量的设定值，第二个变量包含实际值。
3	用三个 *Windows 对象* → *按钮*和一个*智能对象* → *I/O 域*组态画面。在本实例中，使用*按钮 1*、*按钮 2* 和*按钮 3* 以及 *I/O 域 1* 对象。对于这个画面，使用的是 *pictu_5_window_14.pdl*。
4	在 *I/O 域 1* 对象的*组态对话框*中，组态一个与 *U16i_pictu_input_04* 变量的*变量连接*，并且一旦改变就对其触发。
5	假设容器最大填充量是 40 升。因此 *I/O 域*只接受 0 到 40 之间的输入。为此，设置*属性* → *限制值* → *下限值 0*，设置*属性* → *限制值* → *上限值 40*。
6	为*按钮 1*，在*事件* → *鼠标* → *按下左键*处组态一个直接连接以隐藏该画面。
7	为*按钮 2*，在*事件* → *鼠标* → *按下左键*处组态一个*直接连接*，以便将数值 *1* 分配给变量 *BINi_pictu_input_06*。为*按钮 3*，组态一个*直接连接*将数值 *0* 分配给变量。
8	在第二个画面中，组态*智能对象* → *画面窗口*。在本实例中，使用对象*画面窗口 1*。调整*画面窗口*的尺寸以便与刚创建的画面的尺寸相匹配。如果要使*画面窗口*带边框显示，必须将*画面窗口*的*高度*和*宽度*设置得比画面的高度和宽度大 10 个像素。在*属性* → *其它* → *画面名称*处，选择画面 *pictu_5_window_14.pdl*。

步骤	过程：自动输入检查
9	在同一画面中，组态 *Windows 对象* → *按钮*。在本实例中是对象*按钮 1*。在*事件* → *鼠标* → *按下左键*处，创建一个*直接连接*。将*源常数* → *1* 与目标画*面中的对象* → *画面窗口 1* → *显示*相连接。单击*确定*按钮即可应用这些设置。
10	为了显示填充量，已使用了库中的对象 *Tank2*。为了模拟填充过程，在*属性* → *几何结构* → *宽度*处已经创建了 *C 动作*。在*属性* → *变量分配* → *填充量*处，组态与变量 *U16i_pictu_input_05* 的*变量连接*。
11	对于显示填充量的第二种形式，已经使用了*智能对象* → *I/O 域*，在本实例中是 *I/O 域 1*。

模拟填充过程的 C 动作

```
#include "apdefap.h"
 long _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty

{
BOOL state;
SHORT level1,level2;

//get valve state
state=GetTagBit("BINi_pictu_input_06");

if (state==TRUE) {
        level1=GetTagWord("U16i_pictu_input_04");
        level2=GetTagWord("U16i_pictu_input_05");
        level2++;
        if (level2>=level1) {
                SetTagBit("BINi_pictu_input_06",FALSE);
                }//if
        if (level2<=level1) {
                SetTagWord("U16i_pictu_input_05",level2);
                }//if
        }//if
return(80);
}
```

- 读取阀的状态。
- 打开阀时，读取填充量的实际值和设定值。增加实际值。当实际值达到设定值时，关闭阀。设置包含实际值的变量。
- 返回值是对象的宽度。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 在 *pictu_5_window_14.pdl* 画面中，必须修改 *I/O 域*的变量名和限制值以满足要求。

3.6.5 增强型自动输入检查(实例 05)

通过用选择如上所示的按钮可在 Project_CreatePicture 项目中访问与该主题相关的实例。这些实例均组态在 pictu_3_chapter_05a.pdl 画面中。

任务定义

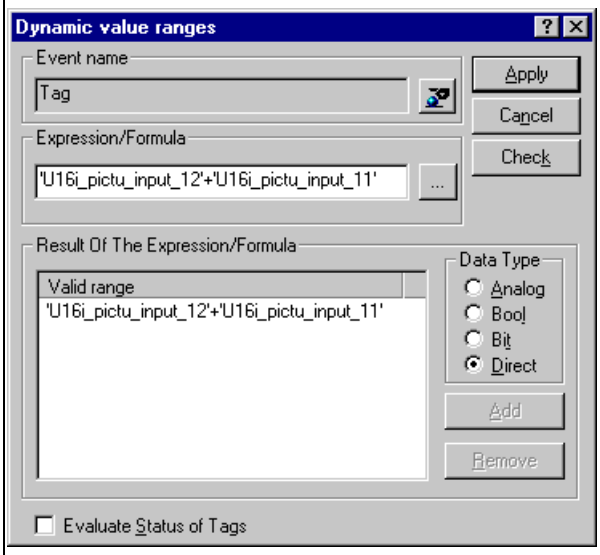
将通过鼠标操作的*按钮*来访问操作面板。使用操作面板把两种液体按一定的比例装入容器中。自动检查输入的两种数值之和，确定其是否超过容器的最大容量。

概念的实现

为实现该任务，可使用 *Windows 对象* → *按钮*，当用按下按钮时，将在*智能对象* → *画面窗口*中显示画面。使用*三个智能对象* → *I/O 域*来输入填充量。此外，将使用*两个 Windows 对象* → *按钮*来应用 *I/O 域*中所做的设置或取消这些设置。

在 WinCC 项目中的实现

步骤	过程：增强自动输入检查
1	在变量管理器中创建两个二进制变量类型的变量，它们包含用于填充容器的阀的当前状态。在本实例中，使用的是变量 *BINi_pictu_input_09* 和 *BINi_pictu_input_10*。
2	创建四个*无符号的 16 位数*类型的变量。在本实例中，它们是变量 *U16i_pictu_input_07* 、 *U16i_pictu_input_08* 、 *U16i_pictu_input_13* 和 *U16i_pictu_input_14*。前两个变量包含容器填充量的设定值，后两个变量包含实际值。
3	创建两个*无符号的 16 位数*类型的变量。在本实例中，它们是变量 *U16i_pictu_input_11* 和 *U16i_pictu_input_12*。它们包含在 *I/O 域*中输入的数值。
4	用两个 *Windows 对象* → *按钮*和三个*智能对象* → *I/O 域*组态画面。在本实例中，使用*按钮 1* 和*按钮 2* 以及 *I/O 域 1*、*I/O 域 2* 和 *I/O 域 3* 对象。至于画面，使用的是 *pictu_5_window_15.pdl*。
5	在 *I/O 域 1* 对象的*组态对话框*中，组态一个与 *U16i_pictu_input_11* 变量的*变量连接*并且一旦改变就对其触发。为 *I/O 域 2* 组态一个与 *U16i_pictu_input_12* 变量的*变量连接*。

步骤	过程：增强自动输入检查
6	为 *I/O 域 3* 在*属性* → *输出/输入* → *输出*值处，组态一个*动态对话框*。在下图中输入设置。将触发设置为一旦改变。 
7	为*按钮 2*，在*事件* → *鼠标* → *按下左键*处组态一个*直接连接*以隐藏该画面。
8	为*按钮 1* 在*事件* → *鼠标* → *按下左键*处创建一个 *C 动作*，以便将输入变量 *U16i_pictu_input_11* 和 *U16i_pictu_input_12* 的内容分配给设定值变量 *U16i_pictu_input_07* 和 *U16i_pictu_input_08*。在*事件* → *鼠标* → *鼠标动作*处，组态一个*直接连接*以关闭画面。
9	在同一个画面中，组态*两个标准对象* → *静态文本*。在本实例中，使用对象*静态文本 5* 和*静态文本 6*。它们用来显示是否超过最大填充量。在包含错误消息的对象*静态文本 5* 处，将*属性* → *其它* → *显示*设置为*否*。
10	为 *I/O 域 3* 在*事件* → *属性主题* → *输出/输入* → *输出值*处创建一个 *C 动作*，可以只有在没有超过最大的填充量时使*按钮 1* 操作，并且可以在超过了最大填充量时显示错误文本。
11	在第二个画面中，组态一个*智能对象* → *画面窗口*。在本实例中，使用*画面窗口 2*。调整*画面窗口*的尺寸以便与刚创建的画面尺寸相匹配。在*属性* → *其它* → *画面名称*上，输入 *pictu_5_window_15.pdl*。
12	在同一画面中，组态一个 *Windows 对象* → *按钮*。在本实例中，这就是对象*按钮 3*。在*事件* → *鼠标* → *按下左键*处，创建一个直接连接。将*源常数* → *1* 与目标*画面中的对象* → *画面窗口 2* → *显示相连接*。单击*确定*按钮即可应用这些设置。

步骤	过程：增强自动输入检查
13	为了显示填充量，已使用了库中的对象 *Tank2*。为了模拟填充过程，在*属性* → *几何结构* → *宽度和属性* → *几何结构* → *高度*处已经创建了 *C 动作*。在*属性* → *变量分配* → *填充量*处，已经创建了*动态对话框*，它返回两个实际数值变量 *U16i_pictu_input_13* 和 *U16i_pictu_input_14* 的总和。
14	对于显示填充量的第二种形式，已经使用了*智能对象* → *I/O 域*，在本实例中是 *I/O 域 2*。

按钮 1 的 C 动作

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszF

{
SHORT tmp1,tmp2;

tmp1=GetTagWord("U16i_pictu_input_11");
tmp2=GetTagWord("U16i_pictu_input_12");

if (tmp1>GetTagWord("U16i_pictu_input_07")){
        SetTagWord("U16i_pictu_input_07",tmp1);
        SetTagBit("BINi_pictu_input_09",TRUE);
        }//if
if (tmp2>GetTagWord("U16i_pictu_input_08")){
        SetTagWord("U16i_pictu_input_08",tmp2);
        SetTagBit("BINi_pictu_input_10",TRUE);
        }//if
}
```

- 读取已在 *I/O 域*中输入的变量值。
- 如果所输入的数值超出了当前的设定值，则将它传送给设定值并使阀打开。

I/O 域 3 的 C 动作

```
#include "apdefap.h"
void OnPropertyChanged(char* lpszPictureName, char* lpszObjectName, char*
{
int a;

a=GetTagWord("U16i_pictu_input_11")+GetTagWord("U16i_pictu_input_12");
if (a<=40) {
      SetOperation(lpszPictureName,"Button1",1);
      SetVisible(lpszPictureName,"Static Text5",0);
      SetVisible(lpszPictureName,"Static Text6",1);
      }//if
else {
      SetOperation(lpszPictureName,"Button1",0);
      SetVisible(lpszPictureName,"Static Text5",1);
      SetVisible(lpszPictureName,"Static Text6",0);
      }//if
}
```

- 读取已在 *I/O 域*中输入的变量值。
- 如果所输入的数值总和超出了容器的最大填充量，则*按钮 1* 变为不可操作，并将显示包含了出错消息的对象*静态文本 5*。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 在 *pictu_5_window_15.pdl* 画面中，必须修改 *I/O 域*的变量名和限制值以满足要求。

3.6.6 多重操作(实例 06)

通过用选择如上所示的*按钮*可在 *Project_CreatePicture* 项目中访问与该主题相关的实例。这些实例在 *pictu_3_chapter_05b.pdl* 画面中组态。

任务定义

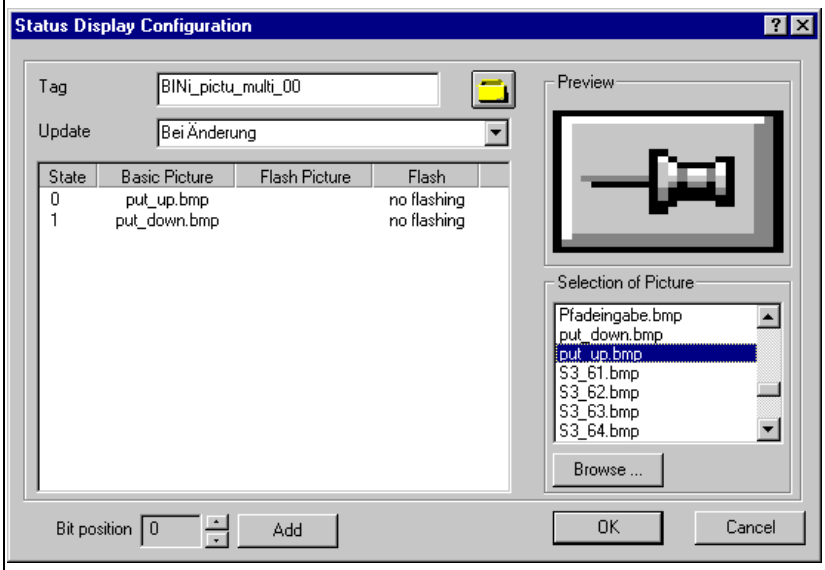
操作面板要通过多个用鼠标操作的*按钮*来访问。如果用一个*按钮*打开了*画面窗口*，则可以控制分配给相应*按钮*的阀。缺省情况下，操作员窗口紧靠着用于调用窗口的*按钮*打开。然而，也可将其定位于任何其它位置。

概念的实现

为了实现上述任务，将使用*窗口对象* → *按钮*，当用按下按钮时，它将把画面显示在*智能对象* → *画面窗口*内。*两个窗口对象* → *按钮*用于控制阀，一个附加的按钮用于关闭窗口。阀的名称及其状态通过两个*标准对象* → *静态文本*来显示。画面的定位借助于*智能对象* → *状态显示*。

在 WinCC 项目中的实现

步骤	过程：多重操作
1	在变量管理器中创建*二进制变量*类型的变量，用于显示阀的当前状态。所需的变量数取决于阀的数目。在本实例中，使用了变量 *BINi_pictu_multi_01*、*BINi_pictu_multi_02*、*BINi_pictu_multi_03* 和 *BINi_pictu_multi_04*。
2	创建一个*文本变量 16 位字符集*类型的变量。在本实例中，它是变量 *T16x_pictu_input_15*。该变量将用作地址变量。
3	创建一个*二进制变量*类型的变量。在本实例中，它是变量 *BINi_pictu_multi_00*。此变量的内容包含有关窗口是否已经定位的信息。
4	组态具有三个 *Windows 对象* → *按钮*的画面。在本实例中，使用了对象*按钮 1*、*按钮 2* 和*按钮 3*。至于画面，使用的是 *pictu_5_window_16.pdl*。
5	对于*按钮 1*，在*事件* → *鼠标* → *按下左键*处创建一个 *C 动作*，用于将画面位置设置在可见区域之外、关闭画面和取消画面定位。
6	对于*按钮 2*，在*事件* → *鼠标* → *按下左键*处创建一个*直接连接*。将源常数 → 1 与目标变量 → T16x_pictu_input_15 相连。选择间接选项钮。通过单击确定按钮即可应用这些设置。这样就设置了间接寻址。用同样的方法为按钮 2 创建一个与源常数 → 0 的直接连接。

步骤	过程：多重操作
7	*组态一个智能对象* → *状态显示*。在本实例中，使用*状态显示 1*。在随后的*组态对话框*中，选择变量 *BINi_pictu_multi_00* 并将触发设置为一旦改变。使用*添加按钮*来添加另一个状态。对于状态 *0*，选择画面 *put_up.gif*；而对于状态 *1*，则选择画面 *put_down.gif*。 
8	对于*状态显示 1*，在*事件* → *鼠标* → *按下左键*处创建一个对变量 *BINi_pictu_multi_00* 的状态求反的 *C 动作*。
9	对于标题，组态一个*标准对象* → *静态文本*。在本实例中，使用了对象*静态文本 1*。在*属性* → *字体* → *文本*处，创建一个从地址变量 *T16x_pictu_input_15* 中读取当前的阀编号并将相应的文本作为返回值来返回的 *C 动作*。
10	组态另一个*标准对象* → *静态文本*来显示阀的状态。在本实例中，使用对象*静态文本 2*。在*属性* → *字体* → *文本*处，创建一个读取当前阀的状态并将相应文本作为返回值来返回的 *C 动作*。在*属性* → *颜色* → *字体颜色*处，创建一个根据当前阀的状态来控制字体颜色的 *C 动作*。
11	在第二个画面中，组态一个*智能对象* → *画面窗口*。在本实例中，使用对象*画面窗口 1*。调整*画面窗口*的尺寸以便与刚创建的画面尺寸相匹配。将*属性* → *其它*可移动和*边框*设置为*是*。在*属性* → *其它* → *画面名称*处，设置画面 *pictu_5_window_16.pdl*。
12	在同一画面中，为每个阀组态一个 *Windows 对象* → *按钮*；在本实例中所用的对象是*按钮 1*、*按钮 2*、*按钮 3* 和*按钮 4*。为每个按钮创建一个 *C 动作*，用于读取*按钮*的编号并将相应的变量名分配给地址变量。根据画面定位与否，或者将画面定位在用于调用它的按钮的右边，或者不是。

关闭按钮(按钮 1)的 C 动作

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
SetLeft("pictu_3_chapter_05b","Picture Window1",-1000);
SetVisible("pictu_3_chapter_05b","Picture Window1",0);
SetTagBit("BINi_pictu_multi_00",FALSE);
}
```

- 将画面位置设置在可见区域之外。
- 隐藏画面。
- 取消画面的定位。

状态显示 1 的 C 动作

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
SetTagBit("BINi_pictu_multi_00",(SHORT)!GetTagBit("BINi_pictu_multi_00"));
}
```

- 对用于画面定位的状态变量求反。

阀控制按钮的 C 动作

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
int x,y;
char name[20];
int number;
int  ch = n;
char *pdest;
//check if odject name contains caracter n
pdest = strrchr( lpszObjectName, ch );
if ( pdest == NULL )(printf("ObjectNameError"));
//read object number
else {
   number = atoi(strrchr(lpszObjectName,n)+1);
   sprintf(name,"BINi_pictu_multi_%02d",number);
   //generate tag name
   SetTagChar("T16x_pictu_input_15",name);
   }
SetVisible(lpszPictureName,"Picture Window1",1);
if (GetTagBit("BINi_pictu_multi_00")==FALSE){
        //get object position
        y=GetTop(lpszPictureName,lpszObjectName);
        x=GetLeft(lpszPictureName,lpszObjectName);
        //set position of picture window
        SetLeft(lpszPictureName,"Picture Window1",-1000);
        SetTop(lpszPictureName,"Picture Window1",y);
        SetLeft(lpszPictureName,"Picture Window1",(x+22));
        }
}
```

- 从对象名中读取对象编号。
- 生成当前状态变量的名称。
- 将地址变量设置为当前的状态变量。
- 显示*画面窗口*。
- 如果*画面窗口*还未定位，则确定*按钮*的位置并将画面的位置设置在紧靠*按钮*的右边。将*画面窗口*设置在可见区域之外，以免第一次更改画面位置时它短暂地显示。

常规应用的注意事项

在进行常规应用之前，必须进行下列修改：

- 根据要求修改对象和变量名称。确保遵守命名约定。按钮编号必须一对一地分配给要切换的变量编号。

3.7 动态化

通过用选择如上所示的*按钮*可在 *Project_CreatePicture* 项目中访问与该主题相关的实例。这些实例在画面 *pictu_3_chapter_06.pdl* 中组态。

3.7.1 颜色更改(实例 01)

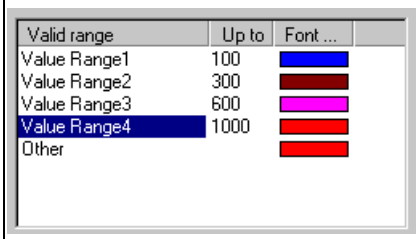
任务定义

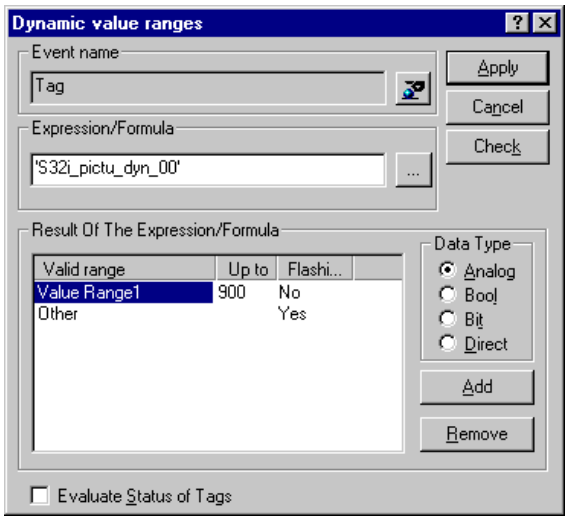
文本颜色将根据变量值更改为各种颜色。

概念的实现

为了实现上述任务，将使用一个 *Windows 对象* → *滚动条对象*来更改变量的值。文本显示借助于*标准对象* → *静态文本*。

在 WinCC 项目中的实现

步骤	过程：颜色更改
1	在变量管理器中创建一个*有符号的 32 位数*类型的变量。在本实例中，使用变量 *S32i_pictu_dyn_00*。
2	组态一个 *Windows 对象* → *滚动条对象*。在本实例中，使用*滚动条对象 1*。在*组态对话框*中，将*最大值*设置为 *1000*，并将*最小值*设置为 *0*。在*事件* → *属性主题* → *其它* → *过程驱动程序连接*处，创建一个与变量 *S32i_pictu_dyn_00* 的*直接连接*。
3	组态一个*标准对象* → *静态文本*。在本实例中，使用对象*静态文本 5*。在*属性* → *字体* → *文本处*，创建一个 *C 动作*，用于输出带相应变量值的文本。一旦变量改变就会触发该 C 动作。
4	在*属性* → *颜色* → *字体颜色*处，创建一个*动态对话框*。在*表达式/公式*域中设置变量 *S32i_pictu_dyn_00*，并且一旦改变就触发它。选择*数据类型模拟*并通过*添加*按钮添加 4 个数值范围。设置如下所示的数值范围： 

步骤	过程：颜色更改
5	在*属性* → *闪烁* → *闪烁背景激活*处，创建一个*动态对话框*。在*表达式/公式*域中设置变量 *S32i_pictu_dyn_00*，并且一旦改变就触发它。选择*数据类型模拟*并通过*添加*按钮添加一个数值范围。设置如下所示的数值范围： 

静态文本的 C 动作

```
#include "apdefap.h"
 char* _main(char* lpszPictureName, char* lpszObjectName, char* lpszPrope
{
char text[100];
DWORD temp;

//get tag value
temp = GetTagDWord("S32i_pictu_dyn_00");
//generate text
switch (GetLanguage())
{
case 0x407:
      sprintf(text,"Die Kesseltemperatur beträgt\r\n%d Grad",temp);
      return text;
case 0x409:
      sprintf(text,"Container Temperature is\r\n%d degree",temp);
      return text;
case 0x40C:
      sprintf(text,"La température de chaudière est\r\nde %d degré",temp);
      return text;
default:
      sprintf(text,"Container Temperature is\r\n%d degree",temp);
      return text;
```

- 读取变量值。
- 用 *sprintf* 函数生成由文本段和数值段组成的文本。这根据当前设置的运行系统语言来执行。
- 返回值是生成的文本。

常规应用的注意事项

在进行常规应用之前，必须完成下列修改：

- 在*动态对话框*中，必须修改所使用的数值范围和变量。

3.7.2 文本切换(实例 02)

任务定义

根据变量的状态自动切换与不同对象相关的文本。工具提示文本同样被切换。

概念的实现

为了实现该任务，将使用一个 *Windows 对象* → *按钮*，它将用于打开和关闭阀。使用*标准对象* → *静态文本*来显示阀是打开的还是关闭的。

在 WinCC 项目中的实现

步骤	过程：文本切换
1	在变量管理器中创建一个*二进制变量*类型的变量。在本实例中，使用变量 *BINi_pictu_dyn_01*。
2	组态一个 *Windows 对象* → *按钮*。在本实例中，使用*按钮 1* 对象。在*事件* → *鼠标* → *按下左键*处创建一个 *C 动作*，用于对 *BINi_pictu_dyn_01* 变量的状态求反。
3	在*属性* → *其它* → *工具提示文本*处创建一个*动态对话框*。在*表达式/公式*域中设置 *BINi_pictu_dyn_01* 变量，并且一旦改变就触发它。选择*数据类型布尔型*，并在有效范围*是/真*中输入文本*关闭*，在有效范围*否/假*中输入文本*打开*。
4	组态一个*标准对象* → *静态文本*。在本实例中，使用*静态文本 7* 对象。在*属性* → *字体* → *文本*处创建一个*动态对话框*。在*表达式/公式*域中设置 *BINi_pictu_dyn_01* 变量，并且一旦改变就触发它。选择*数据类型布尔型*，并在有效范围*是/真*中输入文本阀*打开*，在有效范围*否/假*中输入文本*阀关闭*。

常规应用的注意事项

在进行常规应用之前，必须进行下述修改：

- 在*动态对话框*中，根据需要修改所使用的文本和变量。

3.7.3 移动过程的动画(实例 03)

任务定义

根据变量值将对象移动到屏幕上的指定位置。

概念的实现

为了实现该任务，将使用一个*智能对象* → *画面窗口*，其位置由变量控制。使用 *Windows 对象* → *滚动条对象*来更改变量的值。

在 WinCC 项目中的实现

步骤	过程：移动过程的动画
1	在变量管理器中创建一个*有符号的 32 位数*类型的变量。在本实例中，使用变量 *S32i_pictu_dyn_03*。
2	组态一个 *Windows 对象* → *滚动条对象*；在本实例中，使用*滚动条对象 2*。在*组态对话框*中，将*最大值*设置为 *300*，并将*最小值*设置为 *0*。在*事件* → *属性主题* → *其它* → *过程驱动程序连接处*，创建一个与变量 *S32i_pictu_dyn_03* 的*直接连接*。
3	组态一个*智能对象* → *画面窗口*。在本实例中，使用对象*画面窗口 1*。将*属性* → *其它边框*与*适应画面*大小设置为*是*。在*属性* → *其它* → *画面名称*处，设置 *pictu_3_chapter_00.pdl* 画面。
4	在*属性* → *几何结构* → *位置 X* 处，创建一个*动态对话框*。在*表达式/公式*域中，输入表达式 *((S32i_pictu_dyn_03*2)+90)*。将触发设置为一旦变量 *S32i_pictu_dyn_03* 改变。选择*数据类型*直接。
5	在*属性* → *几何结构* → *位置 Y* 处，创建一个*动态对话框*。在*表达式/公式*域中，输入表达式 *(400-S32i_pictu_dyn_03)*。将触发设置为一旦变量 *S32i_pictu_dyn_03* 改变。选择*数据类型*直接。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 在*动态对话框*中，必须根据需要修改用于计算位的位置的表达式。
- 同样还必须修改变量名。

3.7.4 使用位判断来显示和隐藏对象(实例 04)

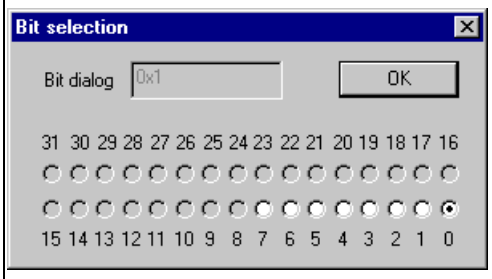
任务定义

根据变量值中指定位的位置来显示和隐藏对象。

概念的实现

可使用一个 *Windows 对象* → *复选框*来实现，它将对变量的各个位进行设置。
许多*标准对象* → *多边形*根据这些位来显示或隐藏。

在 WinCC 项目中的实现

步骤	过程：使用位判断来显示和隐藏对象
1	在变量管理器中，创建一个*无符号的 8 位数*类型的变量。在本实例中，使用了变量 *U08_pictu_dyn_02*。
2	组态一个 *Windows 对象* → *复选框*；在本实例中，使用了对象*复选框 1*。在*属性* → *几何结构* → *方框数目*处，设置所要切换的对象数目；在本实例中，该数目为 *7*。在*属性* → *字体* → *文本处*，为每个索引值输入要由相应位进行切换的对象名称。在 → *事件* → *属性主题* → *输出/输入* → *所选方框* → *更改*处，创建一个与*源属性* → *复选框 1* → *所选方框*和目标*变量 U08i_pictu_dyn_02* 的直接连接。
3	组态多个*标准对象* → *多边形*。在本实例中，使用了对象*多边形 1* 至*多边形 7*。
4	对*多边形 1* 对象，在*属性* → *其它* → *显示*处创建一个*动态对话框*。在*表达式/公式*域中，设置 *U08i_pictu_dyn_02* 变量并且一旦改变就将其触发。选择*数据类型位*。使用 ... 按钮来打开位选择对话框，并选择第一个位。 
5	对其余*多边形*对象按同样方法进行处理，但需修改每个对象的位编号。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 在*动态对话框*中，变量名与画面位置均必须进行修改，以满足需要。

3.7.5 利用 C 动作的移动过程动画(实例 05)

通过使用单击如上所示的*按钮*，可在 *Project_CreatePicture* 项目中访问“添加动态”章节的下列实例。这些实例均组态在 *pictu_3_chapter_06a.pdl* 画面中。

任务定义

通过单击一个*按钮*使一个对象沿某个方向移动，通过单击另一个*按钮*使其沿另一个方向移动。

概念的实现

为了实现上述任务，将使用一个*智能对象* → *状态显示*来显示两个画面。用两个 *Windows 对象* → *按钮*来沿两个不同的方向移动此状态显示。

在 WinCC 项目中的实现

步骤	过程：利用 C 动作的移动过程动画
1	在变量管理器中，创建三个*二进制变量*类型的变量；在本实例中，使用变量 BINi_pictu_dyn_05、BINi_pictu_dyn_06 与 BINi_pictu_dyn_07。
2	组态一个*智能对象* → *状态显示*。在本实例中，使用“状态显示 1”对象。在*组态对话框*中，设置 *BINi_pictu_dyn_05* 变量并将触发设为“一旦改变”。添加另一个状态。为状态 0 设置 *Ferrari1.gif* 画面，为状态 *1* 设置 *Ferrari2.gif* 画面。
3	在*属性* → *状态* → *基本的画面透明颜色*处，为两种状态(*1* 与 *0*)设置颜色*白色*，并将*画面透明颜色打开*设置为*是*。也就是说画面不以白色背景来显示。
4	组态一个 *Windows 对象* → *按钮*。在本实例中，使用*按钮 1* 对象。在*事件* → *鼠标* → *按下左键*处，创建一个将 BINi_pictu_dyn_07 变量设置为 *1* 的*直接连接*，并在*事件* → *鼠标* → *按下右键*处，创建一个将同一变量复位为 *0* 的*直接连接*。
5	在第二个 *Windows 对象* → *按钮*处，按照上述相同方法创建两个与变量 BINi_pictu_dyn_06 的*直接连接*。在本实例中，使用*按钮 2* 对象。
6	对于*状态显示 1* 对象，在*属性* → *几何结构* → *位置 X* 处创建一个 *C 动作*，用于根据所按下的*按钮*来执行移动过程的动画。将该动作的触发设置为 *250 ms*。

移动过程动画的 C 动作

```
#include "apdefap.h"
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyN

{
static int a = 90;

//forward
if (GetTagBit("BINi_pictu_dyn_07")&&(a<652)) {
        a+=20;
        SetTagBit("BINi_pictu_dyn_05",(SHORT)!GetTagBit("BINi_pictu_dyn_05"
        }
//rewind
if (GetTagBit("BINi_pictu_dyn_06")&&(a>-0)) {
        a-=10;
        SetTagBit("BINi_pictu_dyn_05",(SHORT)!GetTagBit("BINi_pictu_dyn_05"
        }
//return x-position
return a;
}
```

- 定义一个 *static int* 类型的变量，并用对象的当前 X 位置对其进行初始化。
- 检查*按钮 1* 是否已按下以及 X 位置是否小于 652。如果是，则将包含 X 位置的值增加 20。然后改变*状态显示 1* 中显示的画面。
- 检查*按钮 2* 是否已按下以及 X 位置是否大于-200。如果是，则将包含 X 位置的值减少 10。然后改变*状态显示 1* 中显示的画面。
- 返回值就是新的 X 位置。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 可以传送动画的基本原理。

3.7.6 使用向导创建移动过程的动画(实例 06)

任务定义

变量改变时对象的屏幕位置随之改变。X 和 Y 轴位置使用不同的变量。通过*动态向导*执行组态。

概念的实现

为了实现这种操作，可使用一个*标准对象* → *圆*，它将在屏幕上进行移动。使用了两个 *Windows 对象* → *滚动条对象*用于变量输入。

在 WinCC 项目中的实现

步骤	过程：使用向导创建移动的动画
1	在变量管理器中创建两个*无符号的 32 位数*类型的变量。在本实例中，使用了变量 *S32i_pictu_dyn_10* 与 *S32i_pictu_dyn_11*。
2	组态两个 *Windows 对象* → *滚动条对象*；在本实例中，使用了*滚动条对象 1* 与*滚动条对象 2*。为*滚动条对象 1* 创建一个*直接连接*。将*源属性* → *滚动条对象 1* → *过程驱动程序连接*与变量 S32i_pictu_dyn_10 相连。采用同样方法，为*滚动条对象 2* 创建一个至变量 *S32i_pictu_dyn_11* 的直接连接。
3	在*滚动条对象*的*组态对话框*中，将最大值设置为 255。
4	组态一个*标准对象* → *圆*。在本实例中，使用了对象*圆 1*。在将对象加亮时，选择*标准动态*标签，然后通过 D 从*动态向导*中选择*移动对象*条目。对于触发器，则选择*变量*。在*设置选项*页面上，选择用于 X 方向的变量 S32i_pictu_dyn_10 和用于 Y 方向的变量 S32i_pictu_dyn_11。分别输入 *0* 与 *255*，用作格式化的下限值与上限值。在下一页上，指定移动对象所属的画面区域。单击*完成*按钮结束向导。
5	在由*动态向导*生成的 *C 动作*中，为在*属性* → *几何结构* → *位置 X 与属性* → *几何结构* → *位置 Y* 处所使用的变量将触发器设置为一旦改变。

由向导在位置 X 处生成的 C 动作

```
#include "apdefap.h"
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyN

{
long i,j,k;
i=GetTagWord("S32i_pictu_dyn_10");
j=((i-0)*100/(255-0));
k=min((((j*(690-490))/100)+490),690);
return max(490,k);

}
```

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 必须根据需要修改*动态向导*中对移动过程的动画进行的设置。

3.7.7 通过 C 动作更改颜色(实例 06)

任务定义

随着变量值的改变，对象颜色要逐渐由深变浅。

概念的实现

为了实现上述任务，将使用一个标准对象 → 圆，其颜色随变量值的变化而变化。为了输入变量值，使用一个 Windows 对象 → 滚动条对象。

在 WinCC 项目中的实现

步骤	过程：通过 C 动作更改颜色
1	在变量管理器中，创建一个*无符号的 32 位数*类型的变量；在本实例中，使用变量 S32i_pictu_dyn_10。
2	组态一个 *Windows 对象* → *滚动条对象*。在本实例中，使用*滚动条对象 1*。在*滚动条对象 1* 的*事件* → *属性主题* → *其它* → *过程驱动程序连接*处，创建一个*直接连接*。将*源属性* → *滚动条对象 1* → *过程驱动程序连接*与变量 S32i_pictu_dyn_10 相连接。
3	在*滚动条对象 1* 处，将*属性* → *其它* → *最大值*设置为 *255*。
4	组态一个*标准对象* → *圆*；在本实例中，使用*圆 1* 对象。在*属性* → *颜色* → *背景色*处创建一个 *C 动作*，用于根据 *S32i_pictu_dyn_10* 变量来提供颜色值。一旦该变量改变就触发此动作。

用于更改颜色的 C 动作

```
#include "apdefap.h"
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyN

{
return (GetTagDWord("S32i_pictu_dyn_10")<<8);
}
```

- 该动作将读取的变量 *S32i_pictu_dyn_10* 向左移动 8 位后作为返回值返回。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 颜色值通过指定红、绿、蓝的值来进行编码。在 24 位颜色值中为每个数值保留 8 位。在本实例中，变量值向左移动了 8 位，因此代表绿色数值。否则，颜色将从黑色变为红色；如果变量移动 16 位，则颜色从黑色变为蓝色。

3.7.8 利用状态显示的移动过程动画(实例 07)

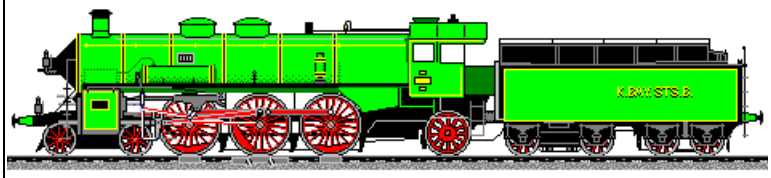
任务定义

通过打开*智能对象* → *状态显示*中的不同画面来模拟移动。

概念的实现

为了实现该任务，可使用一个*智能对象* → *状态显示*，在通过另一个*智能对象* → *状态显示*将显示打开之后，可将不同的画面一个接一个地显示在其中。

在 WinCC 项目中的实现

步骤	过程：利用状态显示的移动过程动画
1	在变量管理器中，创建一个*二进制变量*类型的变量。在本实例中，使用了 BINi_pictu_dyn_09 变量。
2	组态一个*智能对象* → *状态显示*。在本实例中，使用的是对象*状态显示 3*。在*组态对话框*中，设置 *BINi_pictu_dyn_09* 变量并将触发器设为*一旦改变*。添加另一个*状态*。为*状态 0* 设置 Smili.gif 画面，为状态 1 设置 Ohh.gif 画面。
3	为对象*状态显示 3*，在*事件* → *鼠标* → *按下左键*处，创建一个 *C 动作*，用于对变量 BINi_pictu_dyn_09 的状态求反。
4	组态另一个*智能对象* → *状态显示*。在本实例中，使用的对象是*状态显示 4*。通过*属性* → *状态* → *当前状态*，添加具有相应画面的七个附加状态。对各个状态，将属性*基本的画面透明颜色设置为白色*，并将属性*基本的画面透明颜色打开*设置为*是*。从 *0* 到 *7* 的状态均对应分配给从 *S3_61.gif* 到 *S3_68.gif* 的某个画面。 
5	对于对象*状态显示 4*，在*属性* → *状态* → *当前状态*处创建一个 *C 动作*，用于启动当前状态 0 到 7 的运行。将用于该动作的触发器设置为 *250 ms*。

状态显示 4 的 C 动作

```
#include "apdefap.h"
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyN

{
static int a = 0, b = 0;

if (GetTagBit("BINi_pictu_dyn_09")) {
        if (b==0) a++;
        else a--;
        if (a==7) b=1;
        if (a==0) b=0;
        }
return a;
}
```

- 声明两个 *static int* 类型的变量，并将它们初始化为 0。
- 如果激活动画，则运行 0 至 7 的变量，然后再从 0 开始。
- 将此变量作为返回值返回。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 可以传送动画的基本原理。
- 如果修改了状态画面和变量名称，则可将对象*状态显示 3* 以切换对象的形式集成到其它项目中。

3.8 语言切换

访问 *Project_CreatePicture* 项目中与该主题相关的实例，可通过用选择如上所示的*按钮*来完成。这些实例均组态在 *pictu_3_chapter_07.pdl* 画面中。

3.8.1 运行系统语言切换(实例 01)

任务定义

通过为每一种语言设置所分配的按钮对运行系统语言进行切换。

概念的实现

为了实现该任务，可使用三个 *Windows 对象* → *按钮*，这些按钮可从已组态的库中获取。

在 WinCC 项目中的实现

步骤	过程：运行系统语言切换
1	在*图形编辑器*中，可用某种语言对任何画面进行组态。通过*查看* → *语言*菜单，可选择将要组态的下一种语言，并将所有的文本均翻译成该语言。使用位于 WinCC 光盘中的 *language.exe* 程序，项目中所使用的所有文本均可作为 *csv 文件*导出。然后可以翻译文本并将它们导入项目。
2	可通过*查看* → *库*菜单将库打开。从文件夹*全局库* → *按钮语言*中，选择对应的*按钮*。通过单击所期望的对象，并在持续按下 期间，将其拖动到工作域，则可很容易地完成上述操作。
3	如果所需语言不属于库的一部分，则必须在*事件* → *鼠标* → *按下左键*处为 *Windows 对象* → *按钮*创建一个 *C 动作*，它可将语言切换到对应的另一种语言。也可利用*动态向导*来生成相应的 *C 动作*。为了应用向导，可加亮*按钮*，选择*系统功能*标签，然后通过 D 从*动态向导*中选择*语言切换*条目。在*动态向导*中，选择所期望的语言。

德语按钮的 C 动作

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszF

{
SetLanguage(0x407);    //Rückgabe-Typ :BOOL
}
```

- 通过输入相应的语言代码，可使用 *SetLanguage* 函数来修改语言设置。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 在*动态向导*中，完成所期望的语言设置。

3.8.2 用于运行系统与控制中心语言切换的对话框(实例 02)

任务定义

通过*按钮*调用一个对话框，在其中可选择某种设置语言。

概念的实现

为了实现上述任务，将使用一个 *Windows 对象* → *按钮*，用于显示或隐藏一个*智能对象* → *画面窗口*。可直接从 Project_CreatePicture 项目中获取该对话框。

在 WinCC 项目中的实现

步骤	过程：用于运行系统和控制中心语言切换的对话框
1	在*图形编辑器*中，用某种语言组态任一画面。通过*查看* → *语言*菜单，选择要组态的下一种语言，并将所有的文本均翻译成该语言。
2	组态一个*智能对象* → *画面窗口*。在本实例中，使用*画面窗口 1* 对象。在*属性* → *几何结构*处，将*窗口宽度*设置为 *230* 并将*窗口高度*设置为 *214*。在*属性* → *其它*处，将*可移动*、*边框*、*标题*以及*可被关闭*条目设置为*是*。在*属性* → *其它* → *画面名称*处，设置 *pictu_5_window_19.pdl* 画面。该画面位于 *Project_CreatePicture* 项目中，且无需任何修改即可使用。将*属性* → *其它* → *显示*设置为*否*。
3	组态一个 *Windows 对象* → *按钮*。在本实例中，使用对象*按钮 4*。在*事件* → *鼠标* → *按下左键*处，组态一个使*画面窗口 1* 对象可见的*直接连接*。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 画面 *pictu_5_window_19.pdl* 无需进行任何修改即可在另一个项目中再次使用。

3.9 无鼠标时的操作

访问 *Project_CreatePicture* 项目中与该主题相关的实例，可通过使用选择如上所示的*按钮*来完成。这些实例均组态在 *pictu_3_chapter_08.pdl*、*pictu_3_chapter_08a.pdl* 和 *pictu_3_chapter_08b.pdl* 画面中。

3.9.1 利用 TAB 键或热键进行操作(实例 01)

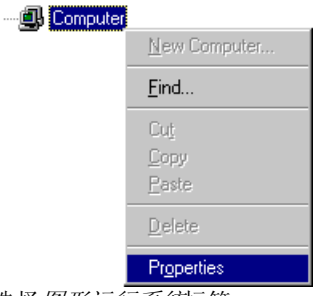
任务定义

使用各种不同的对话框完成对文本的格式化。其字体颜色与各种不同的字体属性，例如字体大小等均要进行设置。此外，所进行的设置要能重新被设置为缺省值。
画面中的所有元素的操作仅通过键盘就能完成。

概念的实现

为了完成该任务，可使用四个 *Windows 对象* → *按钮*。它们能使对话框可见。如果已打开运行系统光标，则可使用键盘来操作对话框。对将要操作的按钮的选择可通过 TAB 键完成。此外，可将热键分配给任何一个按钮。
为了显示对话框，使用了三个*智能对象* → *画面窗口*。

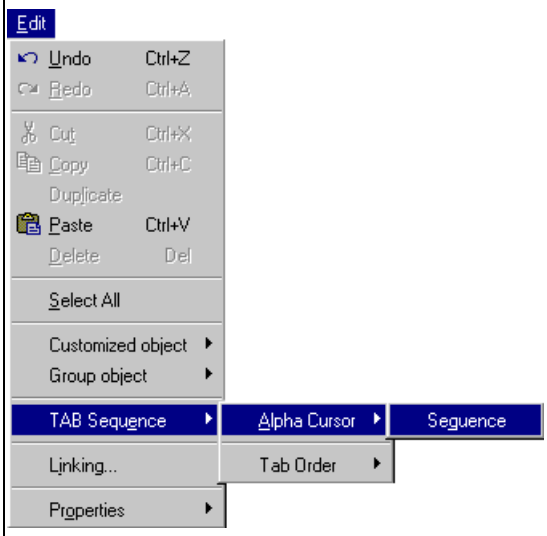
光标控制的组态

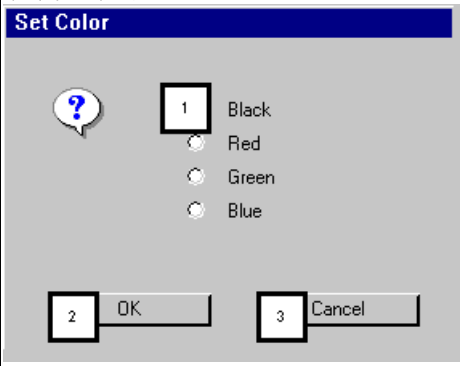
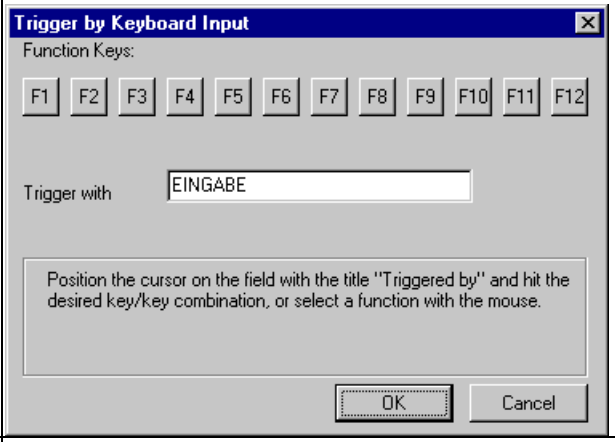
步骤	过程：光标控制的组态
1	在*控制中心*内，为光标控制制定设置。R *计算机*条目，然后从弹出式菜单中选择*属性*。在随后的*计算机列表属性*对话框中，单击按钮*属性*。  选择*图形运行系统*标签。

步骤	过程：光标控制的组态
2	按以下方式进行热键设置。对于*窗口置前*命令，没有配置任何热键，因为在实例中，可通过 *C 动作*设置操作焦点。 为了在 *Tab 顺序/Alpha 光标*之间进行切换，可设置 *SHIFT+A*，为了使运行系统光标打开/关闭，可设置 *SHIFT+R*。
3	在*光标控制*：*键*域中，无需设置任何键。如果实例中需要这些设置，那么可使用一个 API 函数来实现设置。在本实例中将实现该操作，以对各种不同的操作概念进行演示。在通常情况下，可为某个项目选择一种操作概念，并在此处进行设置。

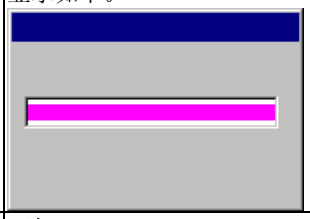
在 WinCC 项目中的实现

步骤	过程：组态热键操作
1	在变量管理器中，创建三个*无符号 16 位数*类型的变量，该变量包含字体属性的设置。在本实例中，使用了变量 *U16i_pictu_cursor_00* 至 *U16i_pictu_cursor_02* 变量。
2	在 *pictu_3_chapter_08.pdl* 画面中，组态了四个 *Windows 对象* → *按钮*类型的对象。在本实例中，它们是对象*按钮 1*、*按钮 2*、*按钮 3* 与*按钮 4*。它们可用于显示对话框以及复位所作的设置。 此外，还组态了一个*标准对象* → *静态文本*，通过对话框可对其字体属性进行设置。在本实例中，使用了对象*静态文本 1*。
3	组态另一个画面，将其用作对设置颜色的对话框。在本实例中，这就是 *pictu_5_window_23.pdl* 画面。 在该画面中，组态一个 *Windows 对象* → *选项组*。在本实例中，这就是对象*选项组 1*。将*属性* → *几何结构* → *方框数目*设置为 *4*。这将允许选择四种不同的颜色。 在*属性* → *输出/输入* → *所选方框*处，创建一个至变量 *U16i_pictu_cursor_00* 的*变量连接*。 在*属性* → *几何结构* → *位置 X* 处，创建一个 *C 动作*，用于将焦点设置给刚才所创建的对象。该 *C 动作*每 *1 h* 触发一次。在激活运行系统光标时，如果*选项组*具有操作焦点，那么它将带边框显示。如果与背景色具有同一颜色的长方形置于边框之上，则对象的边框可隐藏。

步骤	过程：组态热键操作
4	在同一画面中，组态两个 *Windows 对象* → *按钮*。在本实例中，它们是对象*按钮 1* 与*按钮 2*。 *按钮 1* 用作*确定按钮*。在*事件* → *鼠标* → *鼠标动作*处，创建一个 *C 动作*，用于根据 *U16i_pictu_cursor_00* 变量的值修改文本颜色，然后隐藏该对话框。 *按钮 2* 用作*取消按钮*。在*事件* → *鼠标* → *鼠标动作*处，创建一个可使窗口成为不可见窗口的*直接连接*。
5	对键盘操作进行设置。 设置 Tab 顺序。完成该操作可通过*编辑* → *TAB 顺序* → *Tab 顺序* → *顺序*菜单。 

步骤	过程：组态热键操作
	可操作的各个对象现在即可带编号显示。编号顺序代表 tab 顺序。这就是按下 TAB 键时，访问对象的顺序。使用鼠标，单击各个编号，可改变这种顺序。 顺序设置如下：  可通过*光标键*完成*选项组*中的选择。可通过*空格键*完成对颜色的选择。该 *TAB 键*用于在操作元素中进行切换。可通过*空格键*对按钮进行操作。 此外，两个*按钮*均分配有热键。通过*属性* → *其它* → *热键*，可打开用于组态热键的对话框。对于*确定按钮*，设置 *ENTER 键*，对于*取消按钮*，设置 *ESC 键*。 
6	在 *pictu_3_chapter_08.pdl* 画面中，组态一个*智能对象* → *画面窗口*，可在其中显示刚才组态的画面。在本实例中，这就是*对象画面窗口 1*。在*属性* → *其它* → *画面名称*处，设置 *pictu_3_window_23.pdl* 画面。将*属性* → *其它* → *显示*设置为*否*。

步骤	过程：组态热键操作
7	对于*按钮 1*，创建一个 *C 动作*，查询文本当前所设置的颜色，并根据结果将其写入变量 *U16i_pictu_cursor_00* 中。完成该动作后，可将对话框的*选项组*中的选择设置为当前设置值。此外，还将显示对象*画面窗口 1*。 对于*按钮 1*，可在*属性* → *几何结构* → *位置 X* 处创建一个 *C 动作*，用于给该对象设置操作焦点。该 *C 动作*每 *1 小时*触发一次，但焦点只在第一次操作时进行设置。 **Set Color F9** 此外，还将一个热键分配给该*按钮*。在本实例中，这就是功能键 *F9*。
8	组态一个画面，用作对各种不同字体属性进行设置的对话框。在本实例中，这就是 *pictu_5_window_24.pdl* 画面。 在该画面中，组态一个 *Windows 对象* → *复选框*。在本实例中，使用了对象*复选框 1*。将*属性* → *几何结构* → *方框数目*设置为 *4*。可以选择*粗体字*、*斜体字*、*下划线*与*边框*属性。 在*属性* → *输出/输入* → *所选方框*处，创建一个至变量 *U16i_pictu_cursor_01* 的*变量连接*。 在*属性* → *几何结构* → *位置 X* 处，创建一个 *C 动作*，用于给该对象设置焦点。使用与*选项组*中所述相同的过程，可隐藏选择页。 正如 *pictu_5_window_23.pdl* 画面一样，可组态两个 *Windows 对象* → *按钮*。如果激活*确定按钮*，则可读出 *U16i_pictu_cursor_01* 变量，并将对应的设置应用于文本。 与键盘操作相关的设置可采用与 *pictu_5_window_23.pdl* 画面同样的方法来完成。
9	在 *pictu_3_chapter_08.pdl* 画面中，组态另一个*智能对象* → *画面窗口*，可在其中显示刚才组态的画面。在本实例中，这就是对象*画面窗口 2*。在*属性* → *其它* → *画面名称*处，设置 *pictu_3_window_24.pdl* 画面。将*属性* → *其它* → *显示*设置为*否*。
10	对于*按钮 2*，创建一个 *C 动作*，用于对文本当前所要修改的设置字体属性进行询问，并根据结果将其写入变量 *U16i_pictu_cursor_01* 中。完成该动作后，可将对话框的*复选框*中的选择设置为当前的设置值。此外，还将显示对象*画面窗口 2*。 此外，还将一个热键分配给该*按钮*。在本实例中，这就是功能键 *F10*。 **Format F10**

步骤	过程：组态热键操作
11	组态另一个画面，用作设置字体大小的对话框。在本实例中，这就是 *pictu_5_window_25.pdl* 画面。 在该画面中，组态一个*智能对象* → *I/O 域*。在本实例中，使用对象 *I/O 域 1*。 在*属性* → *输出/输入* → *输出值*处，创建一个至变量 *U16i_pictu_cursor_02* 的*变量连接*。 在*属性* → *几何结构* → *位置 X* 处，创建一个 *C 动作*，用于给该对象设置焦点。将一个*图形对象*放置在 *I/O 域*的上面，可隐藏选择页。在本实例中，使用了*图形对象 1*。由*图形对象*所显示的位图在将要显示的 *I/O 域*处的区域中具有一种确定的颜色。*图形对象*中的该颜色在*属性* → *画面* → *画面透明颜色*处进行设置。而且，要将*属性* → *画面* → *画面透明颜色打开*设置为*是*。所使用的位图显示如下。 
12	正如 *pictu_5_window_23.pdl* 画面一样，可组态两个 *Windows 对象* → *按钮*。如果按下了*确定按钮*或*取消按钮*，则对话框将关闭。 然而，将把两个*按钮*从 TAB 顺序中删除。可通过*编辑* → *TAB 顺序* → *Tab 顺序* → *顺序菜单*完成该操作。按下并保持 *CTRL+SHIFT* 组合键，然后使用可将对象从 TAB 顺序中删除。取代编号的是一个 a * 号，它将显示在白色矩形中。 *按钮*操作将只能通过热键 *ENTER* 与 *ESC* 来完成。然而，如果按下 *ENTER* 键，则输入到 *I/O 域*中的值将传送给变量 *U16i_pictu_cursor_02*。 对于 *pictu_3_chapter_08.pdl* 画面中的对象*静态文本 1*，可在*属性* → *字体* → *字体大小*处创建一个至变量 *U16i_pictu_cursor_02* 的*变量连接*。
13	在 *pictu_3_chapter_08.pdl* 画面中，组态另一个*智能对象* → *画面窗口*，可在其中显示刚才组态的画面。在本实例中，这就是对象*画面窗口 3*。在*属性* → *其它* → *画面名称*处，设置 *pictu_3_window_25.pdl* 画面。将*属性* → *其它* → *显示*设置为*否*。
14	对于*按钮 3*，创建一个 *C 动作*，用于显示对象*画面窗口 3*。 此外，还将一个热键分配给该按钮。在本实例中，这就是功能键 *F11*。

步骤	过程：组态热键操作
15	对于*按钮 4*，创建一个 *C 动作*，用于将对象*静态文本 1* 的可设置属性复位为缺省值。将*按钮*分配给热键 *F12*。 **Reset F12**
16	通过*编辑* → *TAB 顺序* → *TAB 顺序* → *顺序菜单*，按相应的顺序，设置对象*按钮 1* 至*按钮 4*。从 TAB 顺序中删除所有其它对象。 对*按钮*进行确认可通过按下*空格键*或对应的热键来完成。

设置焦点的 C 动作

```
#include "apdefap.h"
 long _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropert

{
Static BOOL  bFirst = FALSE;

//set focus in first run
if (bFirst==FALSE)
  Set_Focus(lpszPictureName,lpszObjectName);

BFirst=TRUE;
Return 100;
}
```

- 在函数的初始调用期间，可将焦点设置给其自身对象。每小时调用一次 *C 动作*。然而，焦点只设置一次。
- 在画面 *pictu_5_window_25.pdl* 的对象*选项组 1* 的*属性* → *几何结构* → *位置 X* 处组态该 *C 动作*。其执行周期为 1 小时。

用于设置字体颜色的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert

{
WORD wValue;

//get tag value
wValue=GetTagWord("U16i_pictu_cursor_00");

//set text color
switch (wValue) {
  case 1: SetForeColor("pictu_3_chapter_08.PDL","Static Text1",CO_BLACK);br
  case 2: SetForeColor("pictu_3_chapter_08.PDL","Static Text1",CO_RED);  br
  case 4: SetForeColor("pictu_3_chapter_08.PDL","Static Text1",CO_GREEN);br
  case 8: SetForeColor("pictu_3_chapter_08.PDL","Static Text1",CO_BLUE); br
  default:SetForeColor("pictu_3_chapter_08.PDL","Static Text1",CO_BLACK);br
  }
//close window
SetVisible("pictu_3_chapter_08.PDL","Picture Window1", FALSE);
}
```

- 对*静态文本 1*对象的属性*字体颜色*进行设置需根据变量 *U16i_pictu_cursor _00* 的值来进行。
- 在 *pictu_5_window_23.pdl* 画面中的*确定按钮*按下之后，即可执行该 *C 动作*。

打开画面的 C 动作

```
#include "apdefap.h"
void OnOpenPicture(char* lpszPictureName, char* lpszObjectName, char* lpszPr

{
//load DLL
#pragma code ("pdlrtapi.dll")
#include <pdlrtapi.h>
#pragma code ()

PDLRTSetCursorKeys(255,255,255,255,0,0,NULL, (LPVOID)1,NULL);
}
```

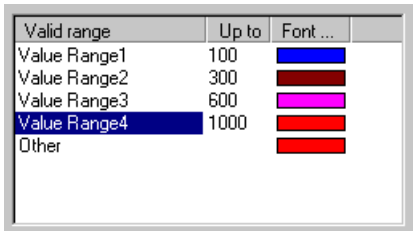
- 如果选择了 *pictu_3_chapter_08.pdl* 画面，则光标键可通过 API 函数 *PDLRTSetCursorKeys* 设置。函数的前 4 个参数包含了用于上下左右移动所需要的键代码。
- 在本实例中，WIN 键用于关闭键盘光标选项的所有光标方向。因此，运行系统光标只能通过 TAB 顺序设置里的 TAB 键来移动。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 如果使用了多种窗口，则必须在控制中心里对其中用于切换的组合键进行定义。按照通过键盘在各个对话框中间进行切换的方法实现本实例中的操作概念是不可能的，也是不必要的。
- 可对组合键和热键进行修改，以满足各自的需要。
- 已设计了本实例，它可只通过 TAB 键，不使用任何特殊的方向键即可移动运行系统光标。然而，对于选项组与复选框的操作，通过缺省值即可使用光标键。

3.9.2 光标键盘(实例 02)



通过组合键 *CTRL+W* 或用 选择如上所示的按钮可以从 *pictu_3_chapter_08.pdl* 画面中访问本实例。它组态在 *pictu_3_chapter_08a.pdl* 画面中。

任务定义

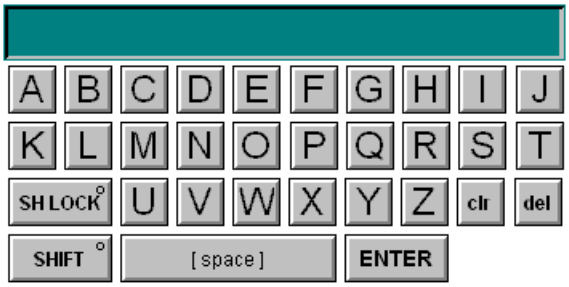
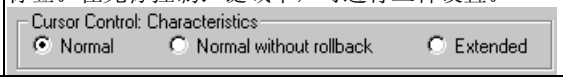
通过画面中所组态的光标键与键盘来输入文本。通过光标键来选择各个字符。
在运行系统中光标特性通过一个对话框来进行设置。该对话框只有在按下热键后才会显示。

概念的实现

为了实现上述任务，使用一个库中预先准备好的键盘。可根据自己的需要对该键盘进行修改。

对话框在*智能对象* → *画面窗口*中显示。为了显示对话框，使用一个已为其分配热键的 *Windows 对象* → *按钮*。按钮本身并不在运行系统中显示。

在图形编辑器中的实现

步骤	过程：在图形编辑器中的实现
1	在变量管理器中，创建两个*无符号 16 位数*类型的变量，它们将存储光标特性设置。在本实例中，使用变量 *U16i_pictu_cursor_04* 和 *U16i_pictu_cursor_05*。 此外，创建一个将输入文本传送给它的*文本变量 16 位字符集*类型的变量。在本实例中，使用变量 *T16i_pictu_cursor_00*。
2	通过工具栏上的按钮打开库。 从键盘文件夹中，选择键盘字符对象并将其拖放到画面中。在本实例中，它是画面 pictu_3_chapter_08a.pdl。解释性的对象可以删除，只需要如下所示的元素： 
3	在*事件* → *鼠标* → *鼠标动作*处为 *ENTER* 按钮创建一个 C 动作，用于将输入的文本写入文本变量中。该变量的名称为 ConnectedVarChar。将该名称改为 T16i_pictu_cursor_00。 在本实例中，变量 T16i_pictu_cursor_00 的内容在具有画面标题的静态文本中显示。这通过与该变量的变量连接来执行。
4	进行光标控制的设置。除了键盘的键以外，将所有对象从 TAB 顺序中删除。TAB 顺序本身不必更改，因为键盘操作要通过光标键来执行，而不是 TAB 键。 光标控制的光标键在*事件* → *其它* → *打开画面*处的 *C 动作*中进行设置。
5	光标特性可通过一个对话框进行设置。 通常情况下，这在*控制中心*内执行。R *计算机*条目，然后从弹出式菜单中选择*属性*。在随后的*计算机列表属性*对话框中，单击按钮*属性*。选择*图形运行系统*标签。在*光标控制：键*域中，可进行三种设置。 

步骤	过程：在图形编辑器中的实现
6	创建一个用作对话窗口的新画面。在本实例中，它是 *pictu_5_window_26.pdl* 画面。 在该画面中，组态三个*智能对象* → *状态显示*。在本实例中，它们是对象*状态显示 1*、*状态显示 2* 与*状态显示 3*。通过*组态对话框*，为用于显示按钮按下状态以及按钮未按下状态的*状态显示*设置位图。*状态 1* 表示已按下的按钮，而*状态 0* 则表示未按下的按钮。 在*属性* → *状态* → *当前状态*处各创建一个*动态对话框*，用于根据变量 *U16i_pictu_cursor_05* 控制当前状态。该变量包含光标特性的临时设置。 在*事件* → *键盘* → *按下*处创建一个 *C 动作*，用于将值写入代表某个选择的 *U16i_pictu_cursor_05* 变量中。这些值是： *0...常规*、*1...常规(不能前移)*、*10...扩充*
7	在同一画面中，组态两个 *Windows 对象* → *按钮*。在本实例中，它们是对象*按钮 1* 与*按钮 2*。 *按钮 1* 用作*确定按钮*。在*事件* → *鼠标* → *鼠标动作*处创建一个 *C 动作*，用于将 *U16i_pictu_cursor_05* 变量的值写入 *U16i_pictu_cursor_04* 变量中。该变量代表当前的光标特性。之后，API 函数 *PDLRTSetCursorKeys* 将转换光标特性。存储在 *U16i_pictu_cursor_04* 变量中的值对应于函数所期望的代表某个光标特性的数值。此外，将焦点设置给键盘的 *A 按钮*，并使窗口不可见。 *按钮 2* 用作*取消按钮*。在*事件* → *鼠标* → *鼠标动作*处创建一个 *C 动作*，用于将焦点设置给键盘的 A 按钮并隐藏窗口。
8	在画面 *pictu_3_chapter_08a.pdl* 内组态一个*智能对象* → *画面窗口*，在其中显示刚才组态的画面。在本实例中，它是*画面窗口 1* 对象。在*属性* → *其它* → *画面名称*处，设置 *pictu_3_window_26.pdl* 画面。将*属性* → *其它* → *显示*设置为*否*。
9	在 *pictu_3_chapter_08a.pdl* 画面中，创建一个 *Windows 对象* → *按钮*。在本实例中，它是对象*按钮 5*。 为*按钮 5* 创建一个 *C 动作*，用于将当前设置的光标特性值写入变量 *U16i_pictu_cursor_05* 中，并显示*画面窗口 1* 对象。 此外，还为该*按钮*分配一个热键。在本实例中，它就是组合键 *CTRL+E*。将*属性* → *其它* → *显示*设置为*否*。这将隐藏该按钮，但所组态的热键仍然保持激活。

按键事件的 C 动作

```
#include "apdefap.h"
void OnKeyDown(char* lpszPictureName, char* lpszObjectName, char* lpszPrc

{
if (nChar==VK_SPACE)
        SetTagWord("U16i_pictu_cursor_05",0);
}
```

- 如果运行系统光标定位在“状态显示”上面，则按键时就将执行该 C 动作。变量 nChar 包含相应键的键代码。如果它是空格键，则光标特性的对应值将被写入该变量。在本实例中，它是常规光标特性的值。
- 该 C 动作必须在按键事件处组态，因为对象不是一个按钮，而是一个状态显示。否则，可使用鼠标动作事件。

为了设计用鼠标操作的对象，必须在鼠标动作事件处创建另一个 C 动作，以省略对键代码的询问。

确定按钮的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert

{
//load DLL
#pragma code ("pdlrtapi.dll")
#include <pdlrtapi.h>
#pragma code ()

//set selected cursor mode to tag
SetTagWord("U16i_pictu_cursor_04",
        GetTagWord("U16i_pictu_cursor_05"));

//set cursor mode
PDLRTSetCursorKeys(VK_UP,VK_DOWN,VK_LEFT,VK_RIGHT,0,
        GetTagWord("U16i_pictu_cursor_04"),NULL,
        (LPVOID)1,NULL);

//set focus to A-button
Set_Focus("pictu_3_chapter_08a.PDL","Button92");

//close window
SetVisible("pictu_3_chapter_08a.PDL","Picture Window1" ,FALSE);
}
```

- 装载包含 *PDLRTSetCursorKeys* 函数的 DLL。
- 所选择的光标特性存储在 *U16i_pictu_cursor_04* 变量中。
- 通过 API 函数 *PDLRTSetCursorKeys*，进行光标设置。函数的前四个参数包含用于上下左右移动所需键的键代码。第六个参数用于把期望的光标特性传送给函数。变量 *U16i_pictu_cursor_04* 中已包含编码正确的该参数。
- 焦点将重新设置给键盘的 *A 按钮*，并且关闭对话框。
- 在打开画面事件处，也调用了 *PDLRTSetCursorKeys* 函数，并将光标特性设置为常规。此时，已经装载了 DLL。不需要再次装载它。然而，为了保持完整性，将再次提到这一点。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 必须根据需要修改所使用的组合键和热键。
- 库包含两个附加的键盘：数字键盘与数字/字符键盘。可按本实例所描述的相同方法使用它们。

3.9.3 值的输入、操作的切换(例 03)

在 *pictu_3_chapter_08a.pdl* 画面中通过使用组合键 *CTRL+W* 或使用选择如上所示的按钮来对该实例进行访问。其组态在 *pictu_3_chapter_08b.pdl* 画面中。

任务定义

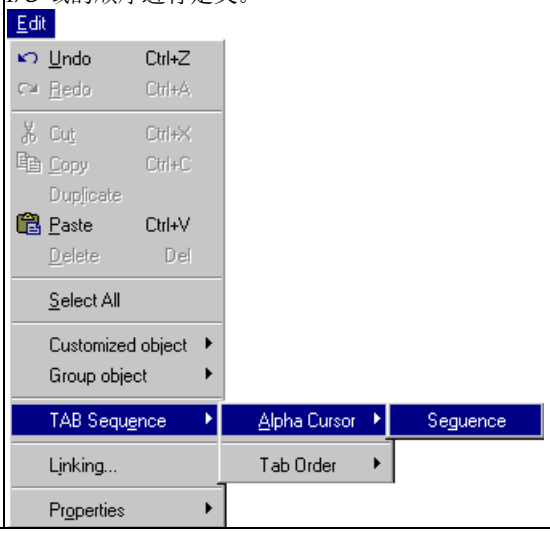
在设备画面中不需使用鼠标即可完成各种不同的控制动作。输入数值，执行若干切换操作。

概念的实现

为了实现该操作，可使用 *Windows 对象* → *按钮*来分配热键。将在*智能对象* → *I/O 域*中进行值的输入和阀的打开与关闭。

图形编辑器中的实现

步骤	过程：图形编辑器中的实现
1	在变量管理器中，创建六个*有符号的 16 位数*类型的变量，用于创建条目和随后对其进行存储。在本实例中，它们是 *S16i_pictu_cursor_00* 至 *S16i_pictu_cursor_05* 变量。
2	在画面中，组态三个*智能对象* → *I/O 域*，用于将填充量设定值输入其中。在本实例中，它们是 *I/O 域 1*、*I/O 域 2* 与 *I/O 域 3* 对象。 对 *I/O 域 1*，在*组态对话框*中创建一个*变量连接*给 *S16i_pictu_cursor_00* 变量，并将*上限值*设置为 *4999*，将*下限值*设置为 *0*。 对其余的 *I/O 域*采用同样的方法进行操作，但要分别设置 *S16i_pictu_cursor_01* 或 *S16i_pictu_cursor_02* 变量。对 *I/O 域 3*，将*上限值*设置为 *9999*。 0000
3	组态三个 *Windows 对象* → *按钮*，用于对 *I/O 域*中所输入的值进行应用。在本实例中，使用了*按钮 F6*、*按钮 F7* 与*按钮 F8* 对象。 对*按钮 F6*，在*事件* → *鼠标* → *鼠标动作*处创建一个 *C 动作*，用于将在 *S16i_pictu_cursor_00* 变量中输入的值写入到 *S16i_pictu_cursor_03* 变量中。通过*属性* → *其它* → *热键*，可为该*按钮*设置 *F6* 热键。 对其余的*按钮*采用同样的方法进行处理。 对*按钮 F6*，可在*属性* → *几何结构* → *宽度*处创建一个 *C 动作*，用于为该按钮设置焦点。 F6

步骤	过程：图形编辑器中的实现
4	组态三个*标准对象* → *矩形*，用于代表所输入的值。在本实例中，它们是*矩形 9*、*矩形 10* 与*矩形 11* 对象。 将*属性* → *填充* → *动态填充*设置为*是*。在*属性* → *填充* → *填充量*处，为每个矩形创建一个*动态对话框*，用于将变量值转化为一个填充量。 对于容器的图形显示，可使用多个成组的*标准对象*。
5	另外组态四个 *Windows 对象* → *按钮*。这些按钮用于打开和关闭阀。在本实例中，它们是*按钮 F9*、*按钮 F10*、*按钮 F11* 与*按钮 F12* 对象。 在*事件* → *鼠标* → *鼠标动作*处，为每个按钮组态一个 *C 动作*，用于对表示数值状态的二进制变量求反。为每个*按钮*分配一个热键。
6	在画面中，组态四个与相应二进制变量相连接的阀。在*信息的显示与隐藏*一章的实例对象的显示与隐藏(实例 01)中可找到有关如何创建阀的详细描述。
7	除已为其分配热键的*按钮*外，将从 TAB 序列中删除所有对象。 通过*编辑* → *TAB 序列* → *Alpha 光标* → *序列*菜单，可对使用 TAB 键选择 *I/O 域*的顺序进行定义。 

步骤	过程：图形编辑器中的实现
8	在*控制中心*中定义一个组合键，用于在 TAB 顺序与 Alpha 光标之间进行切换。在*计算机*条目上使用R，然后从弹出式菜单中选择*属性*。在下列*计算机列表属性*对话框中，单击按钮*属性*。选择*图形运行系统*标签。 为了在 *TAB 顺序/Alpha 光标*之间进行切换，设置组合键 *SHIFT+A*。此外，设置组合键 *SHIFT+R*，以便打开和关闭运行系统光标。 Hotkeys Window On Top Kein Tab Or/Alpha Cursor STRG + ALT + A Runtime-Cursor On/Off UMSCHALT + R

注意：

通过下列按钮或 *ESC* 键，可退出刚才所描述的实例：

ESC

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 必须对所使用的组合键和热键进行修改以满足需求。
- 只有颜色为红色的控制单元设置有函数。其它所有元素均没有函数。整个画面是操作面板 *Simatic OP47* 的图解性表示。

3.10 信息的显示与隐藏

信息

可通过使用选择上面显示的*按钮*来访问 *Project_CreatePicture* 项目中与该主题相关的实例。这些实例均组态在 *pictu_3_chapter_09.pdl* 画面中。

3.10.1 显示和隐藏对象(实例 01)

在许多设备画面内，某些信息条目在画面中不会一直显示，但可以在需要或指定事件发生时显示。

任务定义

用户可以隐藏画面中的某些对象或对象组。

概念的实现

使用显示多个阀的画面来执行控制动作。给每个阀分配一个 *Windows 对象* → *按钮*来控制这个阀，一个*标准对象* → *静态文本*来显示阀的名称和表示阀的状态的一组对象。此外，画面还描述了容器，其填充量通过*智能对象* → *I/O 域*来显示。通过三个 *Windows 对象* → *按钮*，可以显示和隐藏所有的 *I/O 域*、*所有的按钮*和所有的*静态文本*。

在 WinCC 项目中的实现

步骤	过程：显示和隐藏对象
1	在变量管理器中，创建三个*二进制类型*的变量，它们可控制各种对象组的可见性。在本实例中，使用变量 *BINi_pictu_info_12*、*BINi_pictu_info_13* 和 *BINi_pictu_info_14*。
2	在变量管理器中创建*二进制变量*类型的其它变量，它们包含阀的当前状态。所需要的变量数依阀的数目而定。在本实例中，将变量 *BINi_pictu_info_1* 到 *BINi_pictu_pictu_11* 用于总共 11 个阀。
3	为了显示打开的阀，组态一个*标准对象* → *多边形*，它具有阀的外形。在*属性* → *颜色* → *背景色*处，将颜色设置为*深绿色*。
4	为了显示关闭的阀，组态一个*标准对象* → *多边形*，它具有阀的外形。
5	组态两个完全相同的*标准对象* → *矩形*，并在画面的*属性* → *颜色* → *背景色*处设置画面的背景色。*矩形*应比阀略大一些，以便隐藏阀。

步骤	过程：显示和隐藏对象
6	将*矩形*与打开的阀重叠放置，并通过单击*按钮*将打开的阀设置为前景。通过*编辑* → *组对象* → *组菜单*将两个对象构成组。为所生成的*组对象*，在*属性* → *其它* → *显示*处，组态一个至变量 *BINi_pictu_info_1* 的*变量连接*。
7	将关闭的阀定位于第二个*矩形*上并设置为前景。然后将步骤 6 生成的*组对象*定位于阀上并将此设置为前景。现在编组这三个对象。为组态其余的阀，可以复制这个新的*组对象*。必须修改的只有*变量连接*。
8	为每个阀组态一个 *Windows 对象* → *按钮*，并在*事件* → *鼠标* → *按下左键*处创建一个 *C 动作*来对相应的变量值求反。
9	为每个阀组态一个*标准对象* → *静态文本*，它包含阀的名称。
10	组态多个容器，通过*智能对象* → *I/O 域*显示其填充量。
11	组态三个 *Windows 对象* → *按钮*。在本实例中，使用对象*按钮 12*、*按钮 13* 和*按钮 14*。在*事件* → *鼠标* → *按下左键*处，为*按钮 12* 创建一个 *C 动作*来对变量 *BINi_pictu_info_12* 的数值求反。对于其余*按钮*，以同样的方式为变量 *BINi_pictu_info_13* 和 *BINi_pictu_info_14* 创建 *C 动作*。
12	为了通过*按钮 12* 显示或隐藏所有的对象，可创建一个至变量 *BINi_pictu_info_12* 的*变量连接*。对于其它对象执行同样的过程。在本实例中，*按钮 12* 使 *I/O 域*可见，*按钮 13* 使*静态文本*可见，*按钮 14* 使*按钮*可见。

常规应用的注意事项

在进行常规应用前，必须完成下述修改：

- 可以采用显示和隐藏对象的基本方法。
- 可以直接采用显示阀的方法。

3.10.2 日期和时间的显示(实例 02)

任务定义

提供显示日期和时间的不同方法。

概念的实现

为了实现该任务，将使用 OCX 对象。此外，使用两个标准对象 → 静态文本来显示日期和时间。

在 WinCC 项目中的实现

步骤	过程：日期和时间的显示
1	从对象选项板的控件选择菜单中，选择 *WinCC 数字/模拟时钟控件*。这会生成时间显示，用户只要根据需要调整显示的尺寸和类型。
2	组态一个*标准对象* → *静态文本*。在本实例中，使用对象*静态文本 22*。在*属性* → *字体* → *文本*处，创建一个读取当前计算机时间并将其作为返回值来返回的*C 动作*。为该动作设置的触发是 1 秒。
3	组态一个附加的*标准对象* → *静态文本*。在本实例中，使用对象*静态文本 23*。在*属性* → *字体* → *文本*处，创建一个读取当前日期并将其作为返回值来返回的*C 动作*。

读取时间的 C 动作

```
#include "apdefap.h"
char* _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty

{
time_t timer;
struct tm *ptm;
char *p;

time(&timer);
ptm=localtime(&timer);
p=SysMalloc(9);
sprintf(p,"%02d:%02d:%02d",ptm->tm_hour,ptm->tm_min,ptm->tm_sec);
return(p);
}
```

- time(timer)以秒为单位返回当前的系统时间。
- localtime(timer)返回一个指向系统时间结构的指针。
- SysMalloc 保留一个存储区域。
- sprintf 生成由静态段和多个数字段组成的文本。

读取日期的 C 动作

```
#include "apdefap.h"
char* _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty

{
time_t timer;
struct tm *ptm;
char *p;

time(&timer);
ptm=localtime(&timer);
p=SysMalloc(9);
sprintf(p,"%02d:%02d:%02d",ptm->tm_mday,ptm->tm_mon,ptm->tm_year);
return(p);
}
```

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- *WinCC 数字/模拟时钟控件*可以直接传送给另一个项目。
- 在*标准对象* → *静态文本*处的 *C 动作*可以直接传送给另一个项目。

4 WinCC 编辑器(Project_WinCCEditors)

本章中创建的项目也可以直接从在线文档复制到硬盘驱动器上。缺省情况下，它将存储在 *C:\Configuration_Manual* 文件夹中。

Project_WinCCEditors

本项目提供了有关变量记录、报警记录和报表编辑器的若干实例。

有关该主题的实例在 Project_WinCCEditorsWinCC 项目中进行组态。

4.1 变量记录

在运行系统中，有关该主题的实例通过用选择如上所示的按钮来访问。这些实例在画面 *ex_3_chapter_01.pdl* 至 *ex_3_chapter_01f.pdl* 中组态。

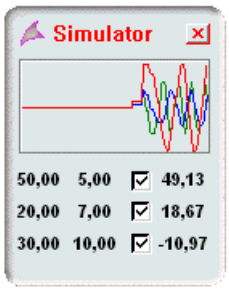
常规信息

*变量记录*包含用于申请来自外部与内部 WinCC 变量数据的函数。这种数据可以用各种方法进行归档。数据在运行系统中可以按趋势或表格的形式进行显示。

过程值的模拟

实例项目提供了一个项目内部的模拟器，用于模拟由变量*记录归档*的过程值。该模拟器通过按下按钮栏上相应的按钮来激活。

该模拟器可以用正弦波图来模拟三个不同的内部变量。另一个变量提供了各变量值的总和。变量值图形在较小的趋势窗口中显示。



趋势窗口下面是由各种输入和输出元素组成的三行。每一行分配给一个趋势图。在第一个 I/O 域中，可以更改趋势的振幅。在第二个 I/O 域中，可以设置每分钟趋势振荡的频率。通过复选框可以停止相应的趋势模拟。在最后一个 I/O 域中，显示当前的趋势振幅。为变量 G64_ex_tlg_01、G64_ex_tlg_02 和 G64_ex_tlg_03 提供数值。给变量 G64i_ex_tlg_04 所赋的值为这三个变量值的总和。如果取消激活该模拟器，则所有变量值都复位为零。

4.1.1 周期连续的归档(ex_3_chapter_01.pdl)

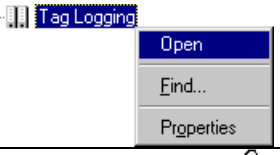
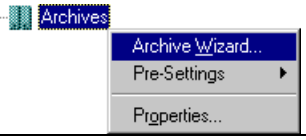
任务定义

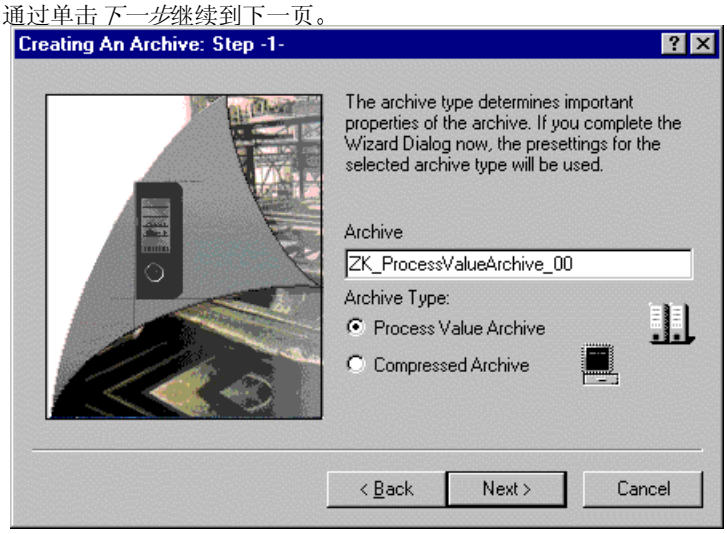
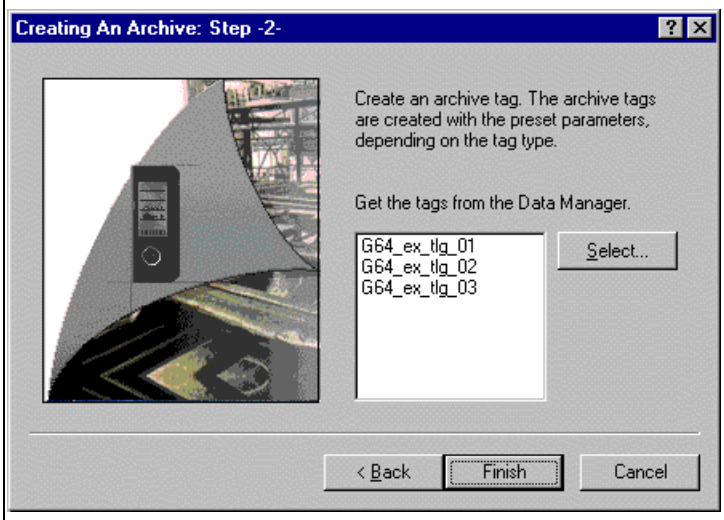
不同的过程值将以设定的周期连续存储在一个归档中。所存储的数据将在运行系统中使用趋势进行图形显示。

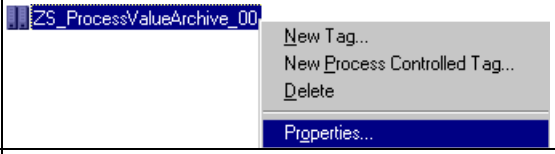
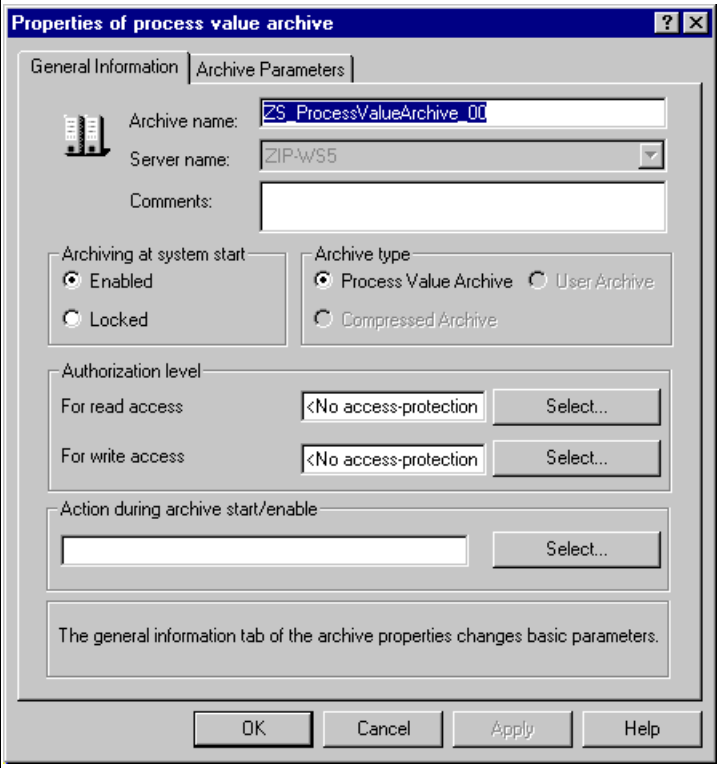
概念的实现

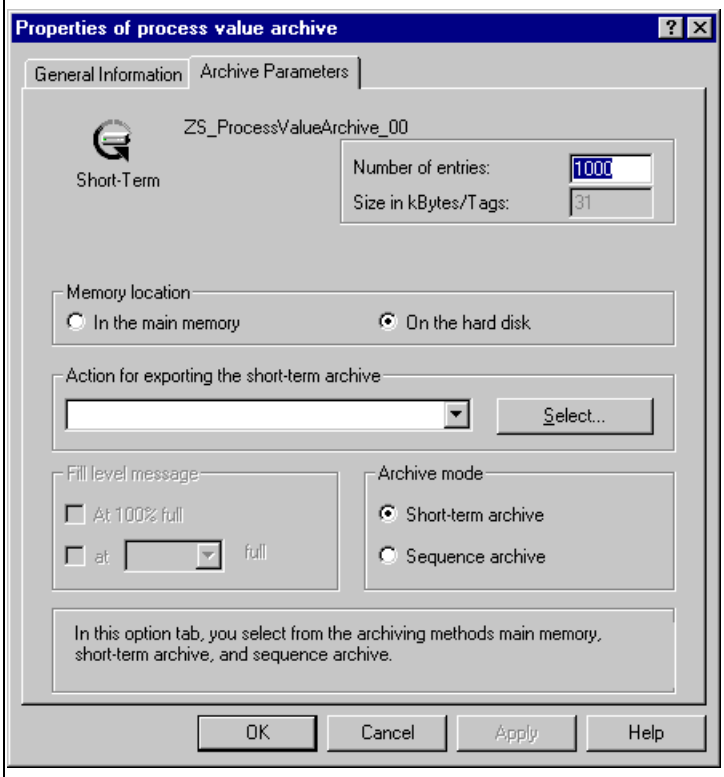
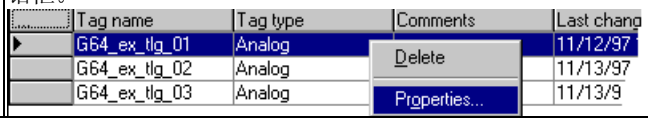
为了对所要显示的数据进行归档，在*变量记录*编辑器中创建一个*周期连续的过程值归档*。
在运行系统中，通过特定的*控件*显示归档。该控件以趋势形式显示数据。

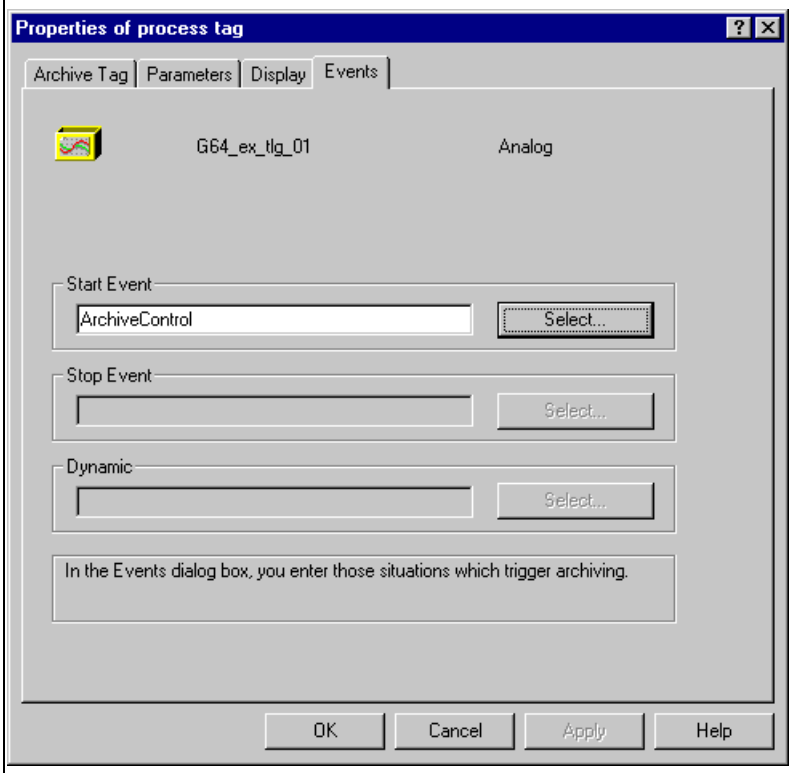
创建过程值归档

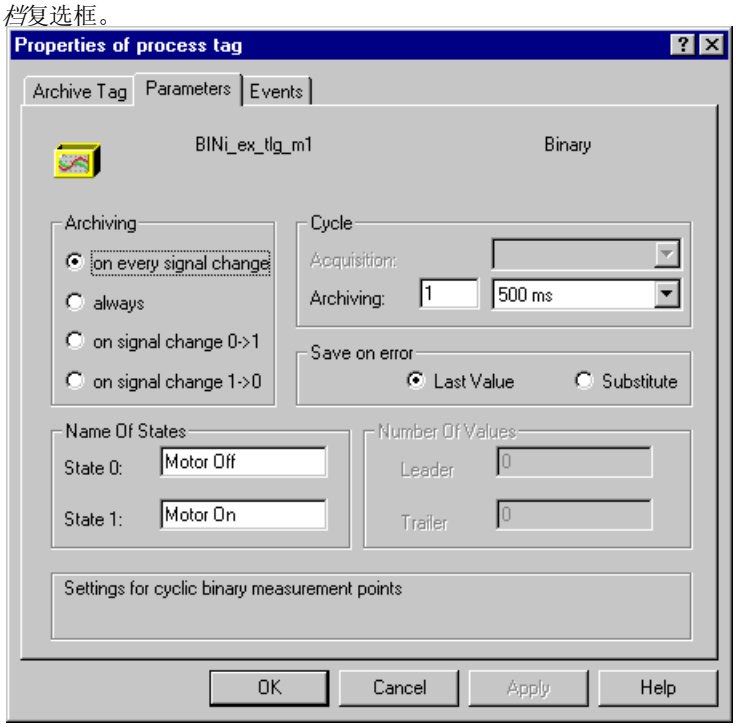
步骤	过程：创建过程值归档
1	在变量管理器中创建要进行归档的变量。 在本实例中，对 *G64_ex_tlg_01*、*G64_ex_tlg_02* 和 *G64_ex_tlg_03* 变量进行归档，它们由模拟器提供数值。
2	打开*变量记录*编辑器。这可从 *WinCC 资源管理器*中通过 R *变量记录*条目，然后从弹出式菜单中选择*打开*来完成。 
3	创建一个新归档。通过 R *归档*条目，然后从弹出式菜单中选择*归档向导*来启动向导。该向导将指导用户创建新归档。 

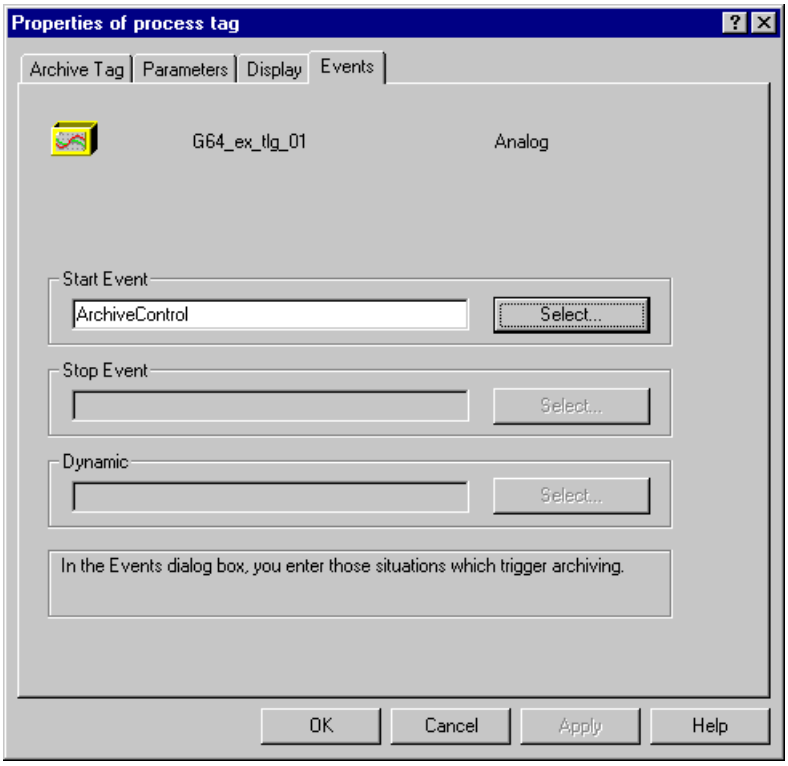
步骤	过程：创建过程值归档
4	通过单击 *下一步*按钮退出起始页。 在下一页中，将*归档类型*设置为*过程值归档*选项。输入*归档名称*。在本实例中，归档名称为 *ZK_ProcessValueArchive_00*。 通过单击 *下一步*继续到下一页。 
5	在向导的第三页中，定义要进行归档的变量。这可通过*选择按钮*来完成。在本实例中，*使用变量 G64_ex_tlg_01*、*G64_ex_tlg_02* 和 *G64_ex_tlg_03*。 通过单击完成关闭此向导页。 

步骤	过程：创建过程值归档
6	在窗口的右边将显示新创建的归档条目。 通过 R 该条目然后选择*属性*来打开该归档的属性对话框。 
7	在*常规信息*标签中，可设置基本的归档参数。*归档名称*和*归档类型*已在*归档向导*中指定。*归档类型*不能再更改。 *系统启动*时*归档*是*激活*的。在系统启动后将直接启动归档。不需要通过一个单独的功能来激活归档。 在*授权等级域*中，将读访问和写访问设置为*无访问保护*。该数据可被任何用户使用，而不需要进行特殊的访问保护。 在启动归档时，不执行特殊动作。这类动作可用来获取例如有关归档状态的信息。 

步骤	过程：创建过程值归档
8	在*归档参数*标签中，还可设置归档的其它属性。 将归档大小设置为 *1000* 条数据记录。为*存储位置*选择*在硬盘上*。为*归档类型*选择*短期归档*。可为用于*导出短期归档的动作*设置一个函数，如果短期归档已满将自动执行该函数。本实例没有指定任何动作。 使用这些设置，将有 1000 条数据记录被归档到硬盘上。如果超出了数据记录的最大数，则最前面的归档条目将被删除并由新的条目取代。 单击*确定*关闭归档的属性对话框。 
9	指定各归档变量的属性。 为此，R 底部的表格窗口，从弹出式菜单中选择*属性*以打开归档变量的属性对话框。 

步骤	过程：创建过程值归档
10	在*归档*变量标签中，可对基本变量属性进行设置。 相应的过程变量已在*归档向导*中指定。可为其分配一个名称作为*归档变量名称*; 在本实例中，使用相应过程变量的名称。 在*提供变量*域中，选择*系统*选项钮。在*系统启动时归档*域中，选择*允许*选项钮。 为*归档类型*设置*周期连续*。这些设置表示数据采集在系统启动时开始，并在恒定的时间间隔内连续进行直到系统关机。 已归档的值不会写入变量中。 

步骤	过程：创建过程值归档
11	在*参数*标签中进行其它设置。 在*周期*域中，输入 *500ms* 作为*采集周期*，输入 *1*500ms* 作为*归档周期*。在*处理*域中，选择*实际值*选项钮。 没有指定*单元*。在出错的情况下，将保存最后的值。没有选择*一旦改变*就进行*归档*复选框。 

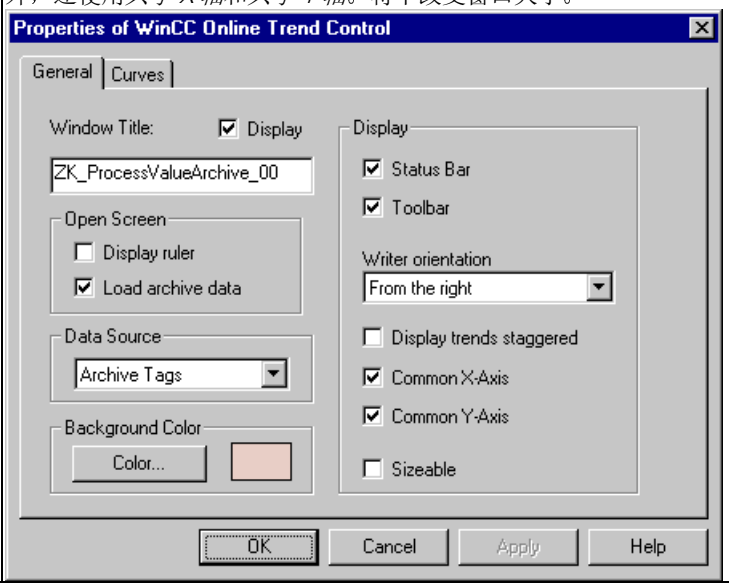
步骤	过程：创建过程值归档
12	在*显示*标签中，指定变量进入归档的可接受范围。 在本实例中，选择*无显示限制*选项钮。 

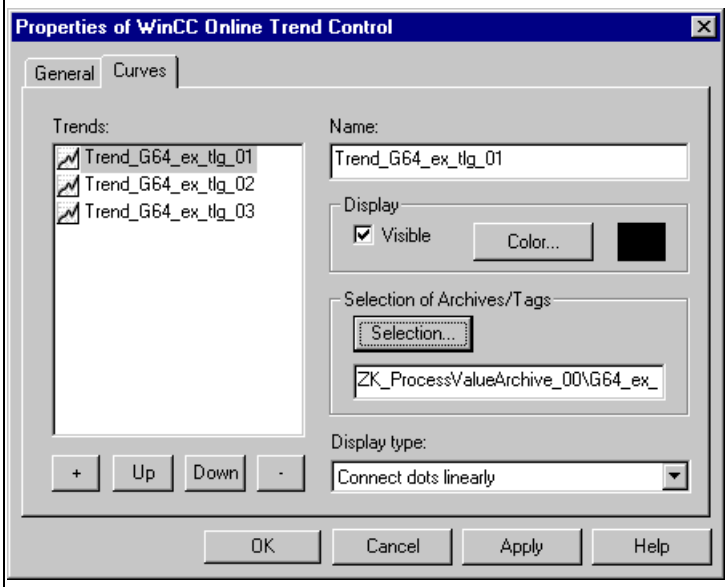
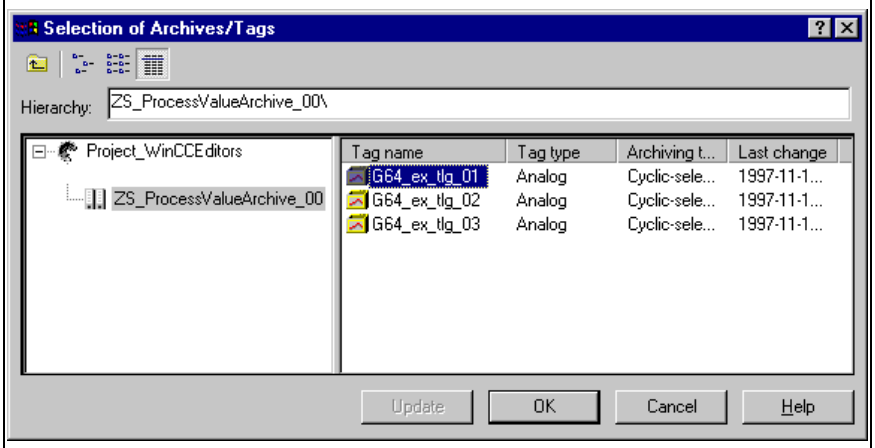
步骤	过程：创建过程值归档
13	在*事件*标签内，本实例没有在*动态*域中输入改变归档周期的动作。 单击*确定*关闭归档变量的属性对话框。
14	还必须指定另两个归档变量的属性。为此，必须执行步骤 9 至 13。

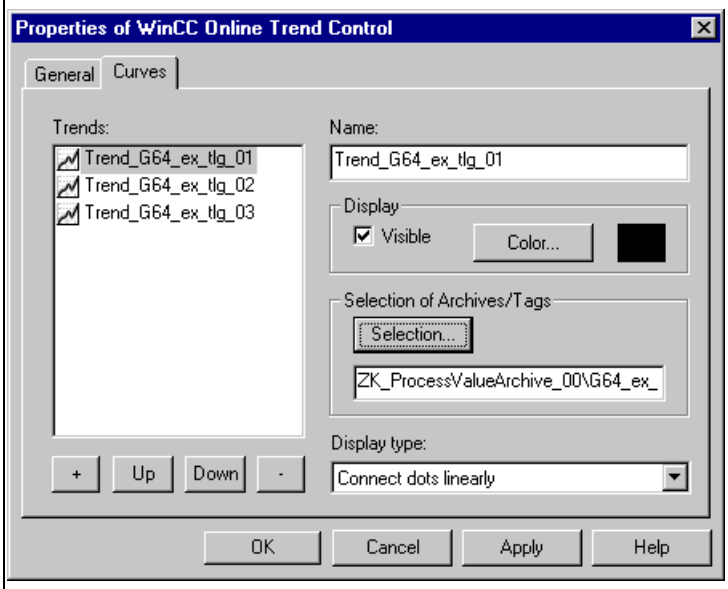
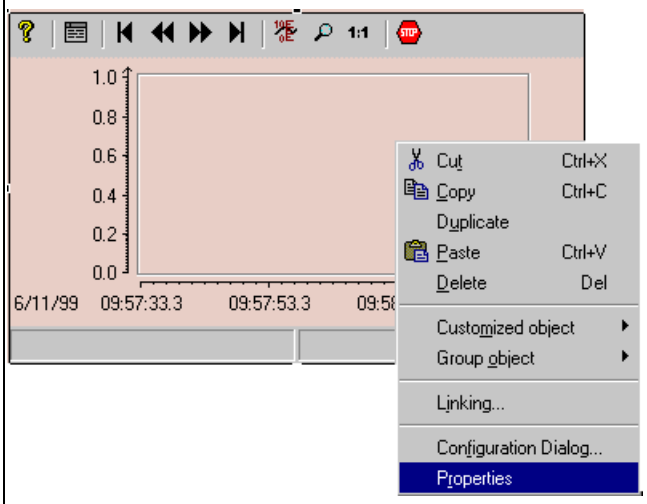
注意：
在生成过程值归档和相应的归档变量期间，由归档向导所作的预设置可由用户通过*归档* → *预设置* → *过程归档*和*归档* → *预设置* → *模拟变量*来更改。如果要创建大量类似的归档，这将很有用。

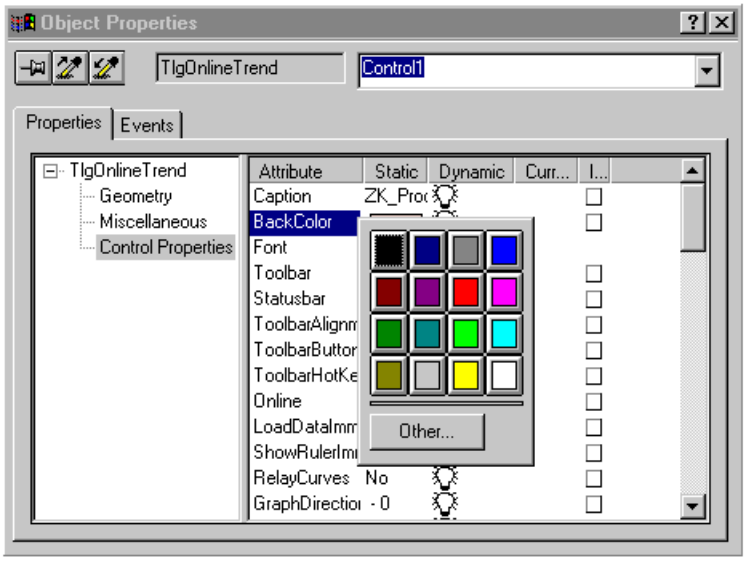
组态趋势显示

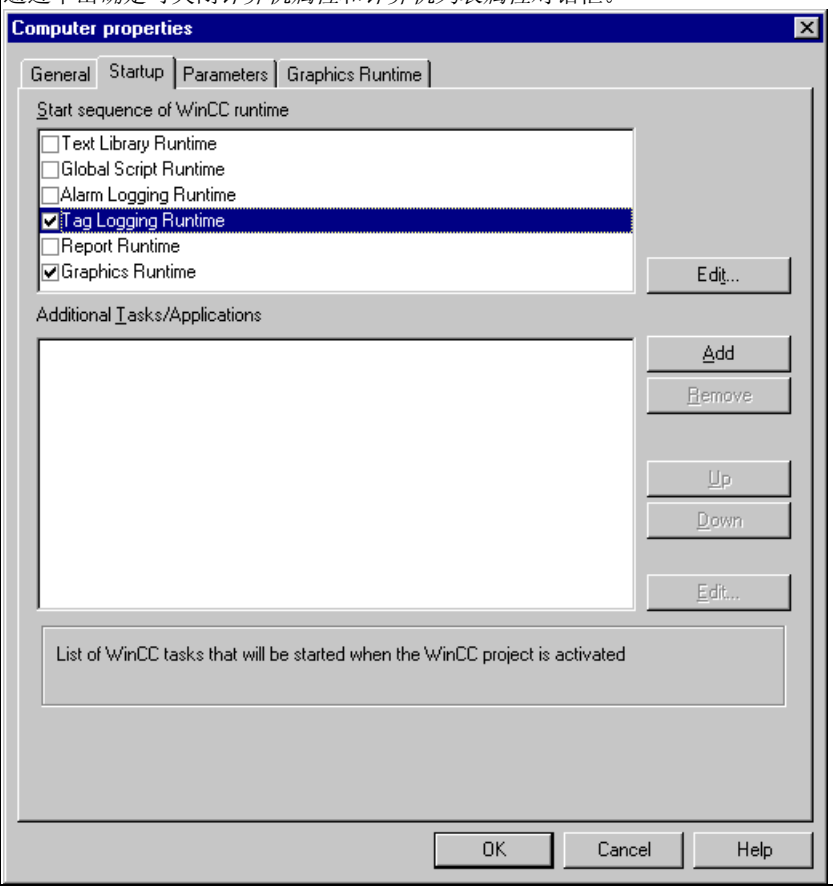
步骤	过程：组态趋势显示
1	在*图形编辑器*中创建一个新画面。在本实例中，它是画面 *ex_3_chapter_01.pdl*。
2	组态用于显示趋势图的控件。它是 *WinCC 在线趋势控件*。从*对象选项板的控件*选择菜单中选择该控件，然后将其置于画面中。

步骤	过程：组态趋势显示
3	将控件置于画面中之后，将会自动打开其组态对话框。 在*常规信息*标签中，可以指定控件的标题以及它如何进行标记。在本实例中，选择*显示*复选框，并输入先前创建的归档名 *ZK_ProcessValueArchive_00* 作为*窗口标题*。 在*打开画面*域中，可指定当打开画面时已显示标尺窗口。在本实例中没有对其进行指定。因此，如果需要的话，标尺窗口必须通过相应的工具栏按钮打开。选择*从归档装载数据*复选框。如果没有选择该复选框，则在画面打开后该控件将只是显示已归档的值。 在*数据源*域中，可选择要显示*归档变量*还是*在线变量*的图。如果选择*在线变量*，则也可以显示没有进行归档的变量的趋势图。本实例中，设置*归档变量*。 通过*颜色*按钮，可指定趋势窗口的*背景色*。如果要使整个调色板都可用，则必须按步骤 7 中所描述的那样，在*控件 1* 对象的*对象属性*对话框中进行设置。 在“显示”域中，本实例规定显示*工具栏*和*状态栏*。为写入方向选择从右边。此外，还使用*共享 X 轴*和*共享 Y 轴*。将不改变窗口大小。 

步骤	过程：组态趋势显示
4	在*趋势*标签中，详细规定要显示的趋势图。 已经创建了一条趋势。在本实例中，将该趋势重命名为 *Trend_G64_ex_tlg_01*。 将*显示类型*设置为*用直线连接点*。 通过*选择*按钮，可以把要显示的归档变量分配给该趋势。 
5	显示*归档/变量选择*对话框。 在左边窗口中，选择所期望的 *ZK_ProcessValueArchive_00* 归档。在右边窗口中，选择所期望的在该归档中可用的归档变量 *G64_ex_tlg_01*。 通过单击*确定*按钮退出对话框。 

步骤	过程：组态趋势显示
6	另外创建两条趋势来显示其余*归档变量*。 通过单击+按钮在*趋势*标签中添加一条新趋势。 按步骤 4 至 5 中所描述的过程来设置其属性。然而，使用的*归档变量*是 *G64_ex_tlg_02 和 G64_ex_tlg_03*。 可以通过单击*确定*按钮关闭控件的属性对话框。 
7	设置趋势窗口的背景色。为此，OR 然后从弹出式菜单中选择*属性*来打开*控件 1* 对象的*对象属性*对话框。 

步骤	过程：组态趋势显示
	在本实例中，*背景色*与项目中所采用的颜色方案相匹配。 在此处还可以进行 *WinCC 用户归档表格控件属性*对话框的所有设置。然而，这对于有些设置可能没用。 

步骤	过程：组态趋势显示
8	*激活变量记录运行系统。* 为此，在 *WinCC 资源管理器*中 R *计算机*条目，然后从弹出式菜单中选择*属性*来打开*计算机列表属性*对话框。单击*属性*按钮来打开本地计算机的属性对话框。在*启动*标签中，选择要激活的运行系统应用程序。必须选择*变量记录运行系统*复选框。 通过单击*确定*可关闭*计算机属性*和*计算机列表属性*对话框。 

有关属性对话框的注意事项：

要设置 *WinCC 在线趋势控件*的属性，可使用三种不同的对话框。

- **组态对话框：**该对话框将在创建控件时自动打开。它向用户提供了允许快速组态控件的最重要的设置选项。这是在上述描述中主要使用的对话框。可在按住 SHIFT 键的同时 D 控件来将其打开。
- **控件的属性对话框：**该对话框的内容更为全面。它允许根据用户的需要对控件进行更为精确的调整。此对话框可通过 D 控件来打开。
- **对象属性对话框：**这是*图形编辑器*的缺省属性对话框。它可通过 R 控件，然后从弹出式菜单中选择*属性*来打开。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 必须根据需要修改要归档的变量。
- 仅当要显示快速变化的数值图时，本实例中所选的这个高(快)归档周期才有意义。在通常情况下，只需要较低的周期。较高的归档周期会加重系统的负担。

4.1.2 周期选择归档(ex_3_chapter_01a.pdl)

任务定义

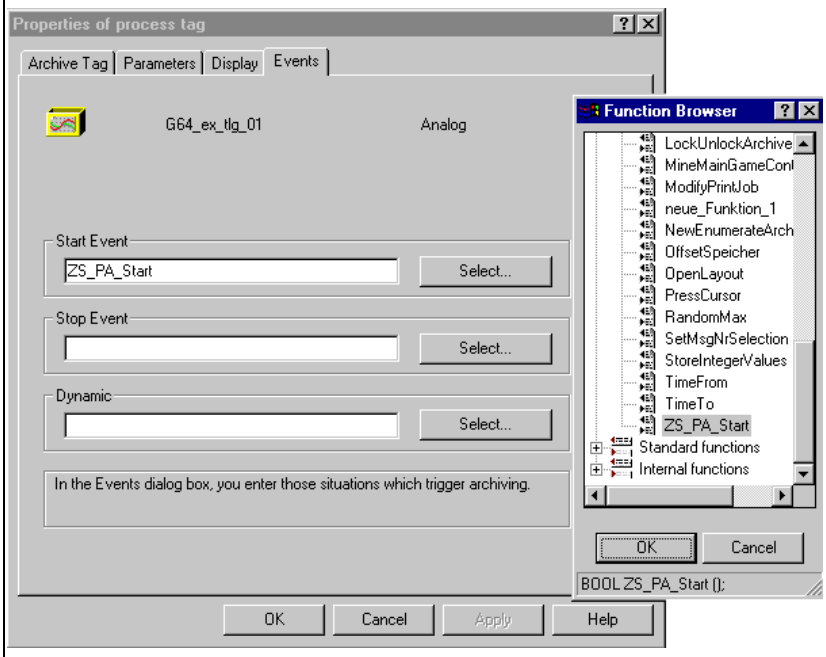
不同的过程值将以设定的周期连续存储在一个归档中。通过按钮来启动和停止归档。所存储的数据将在运行系统中用趋势进行图形显示。要组态具有已定义对象的工具栏和状态栏。

概念的实现

为了对所要显示的数据进行归档，在*变量记录*编辑器中创建一个*周期选择的过程值归档*。
在运行系统中，归档通过特定的*控件*来显示。该控件以趋势形式显示数据。所需的工具栏用各种*按钮*、*状态显示*和*图形对象*来实现。状态栏用两个*按钮*来实现。
为了控制归档，需要一个*项目函数*来启动和停止归档。

创建过程值归档

步骤	过程：创建过程值归档
1	在变量管理器中创建要进行归档的变量。 在本实例中，对 *G64_ex_tlg_01*、*G64_ex_tlg_02* 和 *G64_ex_tlg_03* 变量进行归档，它们由模拟器提供数值。 创建一个*二进制变量*类型的附加变量，它将存储归档的当前状态。在本实例中，它是变量 *BINi_ex_tlg_00*。
2	在*全局脚本*编辑器中创建一个*项目函数*来启动和停止归档。 在本实例中，它是函数 *ZS_PA_Start*。其功能在本表格后进行描述。
3	在*变量记录*编辑器中创建一个*过程值归档*。 这通过*归档向导*来完成。在本实例中，归档已经命名为 *ZS_ProcessValueArchive_00*。选择变量 *G64_ex_tlg_01*、*G64_ex_tlg_02* 和 *G64_ex_tlg_03* 进行归档。
4	设置*过程值归档*的属性。 在*归档参数*标签中将归档大小设置为 *1000* 条数据记录。其余选项保留缺省设置。

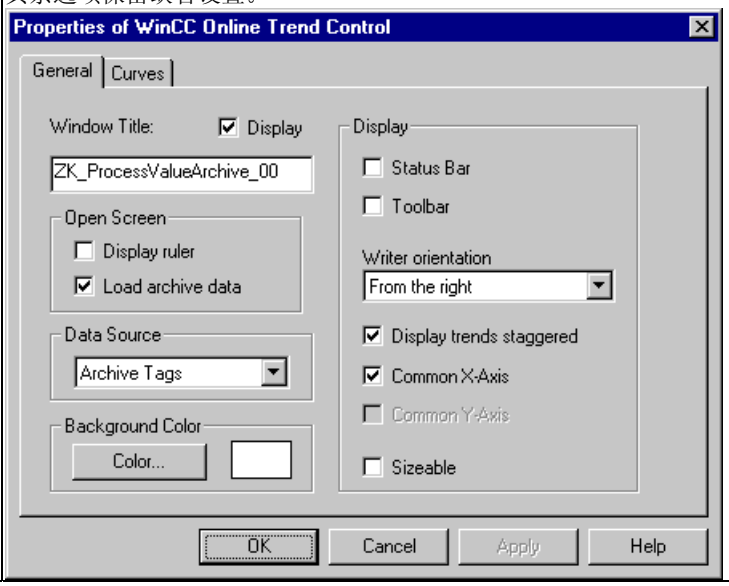
步骤	过程：创建过程值归档
5	设置*归档变量*的属性。 在*归档变量*标签中，三个变量都选择*周期选择*作为*归档类型*。 这种归档类型使用户可以选择设置*事件*标签中的*启动事件*和*停止事件*。在本实例中，将先前创建的*项目函数 ZS_PA_Start* 设置为*启动事件*。 其余选项保留缺省设置。 

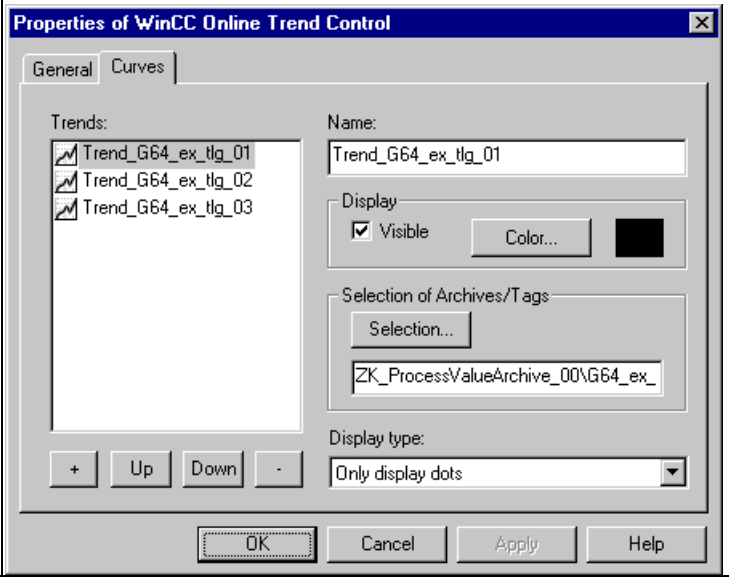
项目函数 ZS_PA_Start

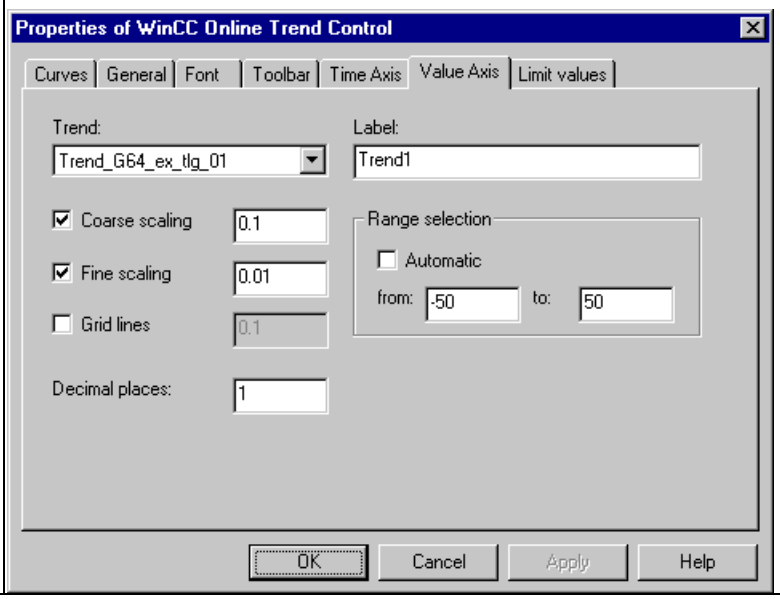
```
BOOL ZS_PA_Start()
{

if (GetTagBit("BINi_ex_tlg_00"))
{
      return TRUE;
}
else
{
      return FALSE;
}

}
```

- 该函数根据二进制变量 *BINi_ex_tlg_00* 的状态，返回数值 *TRUE* 或 *FALSE*。
- 在每个归档周期中，由*变量记录*调用该函数。通过返回值，决定是否执行归档。返回值 *TRUE* 启动归档。

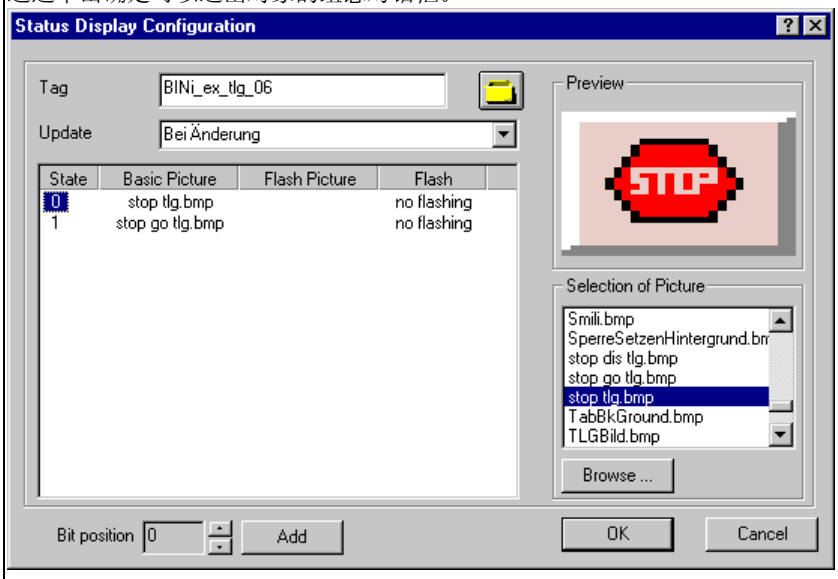
组态趋势显示

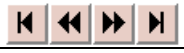
步骤	过程：组态趋势显示
1	在*图形编辑器*中创建一个新画面。在本实例中，它是画面 *ex_3_chapter_01a.pdl*。
2	组态用于显示趋势图的控件。它是 *WinCC 在线趋势控件*。从*对象选项板的控件*选择菜单中选择它，然后将其置于画面中。将控件置于画面中之后，其组态对话框将自动打开。 在*常规信息*标签中，可以指定控件的标题以及它如何进行标记。在本实例中，撤消选定*显示*复选框。*窗口标题*仍然输入。在稍后创建的 *C 动作*中，该窗口标题用于注明相应的控件。使用先前创建的归档名称 *ZS_ProcessValueArchive_00*。 通过*颜色*按钮，将趋势窗口的*背景色*设置为白色。 在“显示”域内，本实例规定不显示*工具栏*和*状态栏*。选择*交错趋势*复选框。也就是说每个趋势用单独的图来显示。 其余选项保留缺省设置。 

步骤	过程：组态趋势显示
3	在*趋势*标签内，详细规定要显示的趋势图。 创建三个趋势。将归档 *ZS_ProcessValueArchive_00* 的变量 *G64_ex_tlg_01* 至 *G64_ex_tlg_03* 分配给这些趋势。将三个趋势的颜色设置为黑色，并将*显示类型*设置为*只显示点*。 其余选项保留缺省设置。可以通过单击*确定*按钮关闭控件的属性对话框。 

步骤	过程：组态趋势显示
4	各趋势的特殊属性设置。为此，可以使用一个扩充的属性对话框。该对话框通过 D 控件来打开。另一方面，先前所描述的属性对话框在按下 CTRL 键的同时通过 D 控件来打开。 扩充的属性对话框除了包含已经提及的*常规信息*和*趋势*标签外，还包含五个附加的标签。在本实例中，只在*数值轴*标签内进行设置。 在*趋势*域中，设置条目 *Trend_G64_ex_tlg_01* 来定义该趋势的属性。在*标签*域内，输入文本*趋势 1*。*范围选择*并不是自动执行，而是设置为从*-50* 至 *50*。其余选项保留缺省设置。 其余趋势的属性用刚才所述的相同方法进行设置。可以通过单击*确定*按钮关闭控件的属性对话框。 

组态工具栏和状态栏

步骤	过程：组态工具栏和状态栏
1	在变量管理器中，创建一个二进制变量类型的内部变量。在本实例中，它是变量 *BINi_ex_tlg_06*。
2	为了控制更新，组态一个*智能对象* → *状态显示*。在本实例中，使用对象*状态显示 5*。 通过其*组态对话框*，将对象连接到变量 *BINi_ex_tlg_06* 上并且一旦改变就触发。创建状态 *0* 和 *1*。在本实例中，将位图 *stoptlg.bmp* 和 *stopgotlg.bmp* 分配给这些状态。 通过单击*确定*可以退出对象的*组态对话框*。 
3	在*事件* → *鼠标* → *按下左键*处，为刚组态的对象*状态显示 5* 创建一个 *C 动作*。该 *C 动作*模拟按下控件的标准工具栏的停止/执行按钮。此外，对变量 *BINi_ex_tlg_06* 的状态求反以显示已改变的控件更新的状态。变量值为 0 对应于更新已激活。 由于当画面打开时总是会激活趋势窗口的更新，所以在打开画面时变量 *BINi_ex_tlg_06* 的状态总是为零。这通过画面对象 *ex_3_chapter_01a.pdl* 的*事件* → *其它* → *打开画面*处的直接连接来实现。这样就将该变量的状态设置为 *0*。

步骤	过程：组态工具栏和状态栏
4	按步骤 2 的描述，组态第二个*智能对象* → *状态显示*。在本实例中，它是对象*状态显示 6*。该对象用于控制归档。 将该对象与前一部分中所创建的变量 *BINi_ex_tlg_00* 相连接。相应地使用不同位图*(Archive.bmp / Archive inv.bmp)*。 在*事件* → *鼠标*按下左键处，创建一个 *C 动作*。该动作对变量 *BINi_ex_tlg_00* 求反。该变量用于显示归档已改变的状态，并且通过*项目函数 ZS_PA_Start* 将该信息传送给归档。
5	为了当更新停止时在归档中进行浏览，需要复制四个控件的标准工具栏的浏览按钮。 为了实现上述任务，组态四个 *Windows 对象* → *按钮*；在本实例中，它们是对象*按钮 4*、*按钮 7*、*按钮 8* 和*按钮 11*。 为其中的每个对象，在*事件* → *鼠标* → *鼠标动作*处创建一个 *C 动作*。这些动作模拟按下标准工具栏上的按钮。 此外，需要一个*智能对象* → *图形对象*来将其本身置于这些按钮上，并使它们在启动更新时不可操作。在本实例中，它是*图形对象 2*。此对象显示四个处于不可操作状态的按钮*(Pfeile dis.bmp)*。在*属性* → *其它* → *显示*处，创建一个*动态对话框*。此对话框根据变量 *BINi_ex_tlg_06* 控制对象的可见性，该变量包含有关控件更新的信息。 
6	为了显示状态栏，组态两个 *Windows 对象* → *按钮*；在本实例中，它们是对象*按钮 5* 和*按钮 6*。 对于文本显示，使用*按钮*，因为它们可以很容易配备 3D 边框。因此，无需任何附加的对象。 为*按钮 5*，在*属性* → *字体* → *文本*处创建一个 *C 动作*。该动作根据变量 *BINi_ex_tlg_00* 将*文本归档已启动或归档已停止*返回给属性。利用 *C 动作*而不是等效的动态对话框来实现语言的切换。 按刚才描述的相同方法对*按钮 6* 与变量 *BINi_ex_tlg_06* 进行处理。

停止/执行按钮对象(状态显示 5)的 C 动作

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszI
{

TlgTrendWindowPressStartStopButton("ZS_ProcessValueArchive_00");

SetTagBit("BINi_ex_tlg_06",(SHORT)!GetTagBit("BINi_ex_tlg_06"));

}
```

- 调用标准函数 *TlgTrendWindowPressStartStopButton* 与按下控件的标准工具栏上的*停止/执行按钮*具有相同的效果。将文本分配给该函数，以便允许它识别要访问的控件。该文本就是在组态控件时已指定的窗口标题。在本实例中，它是文本 ZS_ProcessValueArchive_00。
- 对变量 *BINi_ex_tlg_06* 求反来存储控件更新的当前状态。

浏览按钮开始(按钮 4)的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProper
{

TlgTrendWindowPressFirstButton("ZS_ProcessValueArchive_00");

}
```

- 调用该标准函数与按下控件的标准工具栏上的*第一条数据记录*按钮具有相同的效果。其它按钮所使用的函数是：
- *TlgTrendWindowPressPrevButton*
- *TlgTrendWindowPressNextButton*
- *TlgTrendWindowPressLastButton*

注意：
对于 *WinCC 在线趋势控件*的标准工具栏上的每个按钮，可以使用模拟按下按钮的相应标准函数。

显示状态栏文本(按钮 5)的 C 动作

```
#include "apdefap.h"
 char* _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

char start[40] = "";
char stop [40] = "";

switch(GetLanguage())
{
case 1031 : strcpy(start,"   Archivierung gestartet... ");
            strcpy(stop, "   Archivierung gestoppt... ");
            break;
case 1033 : strcpy(start,"   Archiving started... ");
            strcpy(stop, "   Archiving stoped... ");
            break;
case 1036 : strcpy(start,"   Archivage démarré...");
            strcpy(stop, "   Archivage arrêté...");
            break;
default   : strcpy(start,"   Archivierung gestartet... ");
            strcpy(stop, "   Archivierung gestoppt... ");
            break;
}

if (GetTagBit("BINi_ex_tlg_00"))
{
  return start;
}
else
{
  return stop;
}

}
```

- 创建两个文本变量。根据当前设置的语言，在这些变量中输入归档启动和归档停止状态的文本。当前设置的语言通过函数 *GetLanguage()*来确定。
- 根据变量 *BINi_ex_tlg_00*，将 *start* 变量或 *stop* 变量中的文本返回给属性。一旦变量 *BINi_ex_tlg_00* 改变，就触发该动作。

> 注意：
> 在以后的实例中，也实现带有独立对象的状态栏。然而为了比较直观，使用*动态对话框*而不是 *C 动作*来控制状态栏。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 必须根据需要修改要归档的变量。
- 仅当要显示快速变化的数值图时，本实例中所选的这个高(快)归档周期才有意义。在通常情况下，只需要较低的周期。较高的归档周期会加重系统的负担。
- 可以根据某个事件(并不一定要按下某个按钮)来启动归档和结束归档。
- 可以根据自己的需要修改所需元素的外观。这同样适用于状态栏。
- 已选择的显示类型能较好地显示不进行归档的时间间隔。在所有其它显示类型中，链接了所有归档的点。这就意味着不进行归档的时间间隔用线跨接起来。

4.1.3 如果超出数值就进行归档(ex_3_chapter_01b.pdl)

任务定义

当超出某个限制值时，过程值要在归档中存储一次。所存储的数值要在表格中显示。此过程值的时间顺序进程要作为趋势显示。要组态具有已定义对象的工具栏和状态栏。

概念的实现

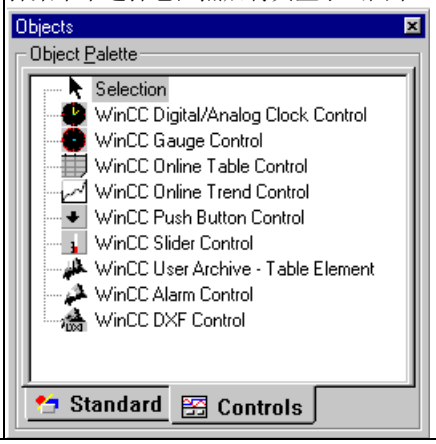
为了对所要显示的数据进行归档，在变量记录编辑器中创建一个*非周期的过程值归档*。
在运行系统中，归档通过特定的控件来显示。该控件以表格形式显示数据。过程值的趋势图通过另一个控件来显示。所需的工具栏用各种按钮、状态显示和*图形对象*来实现。状态栏用按钮来实现。
为了对归档进行控制，创建一个*项目函数*。如果过程值超出某个限制值，则此函数会触发归档操作。

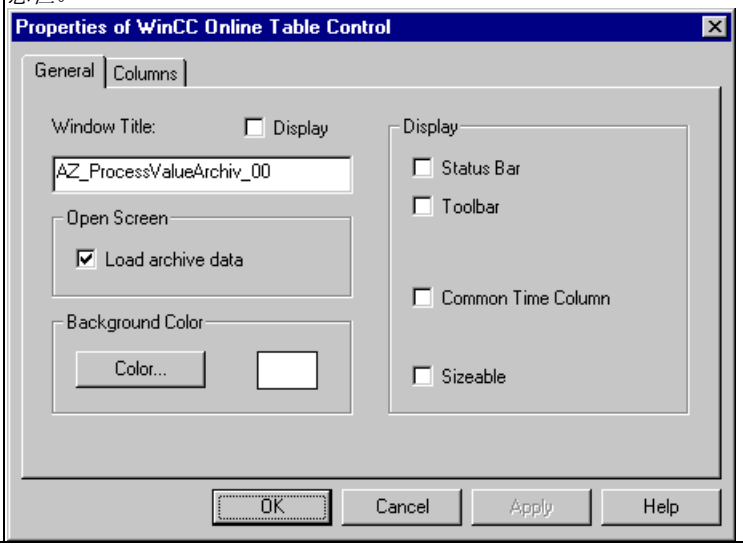
创建过程值归档

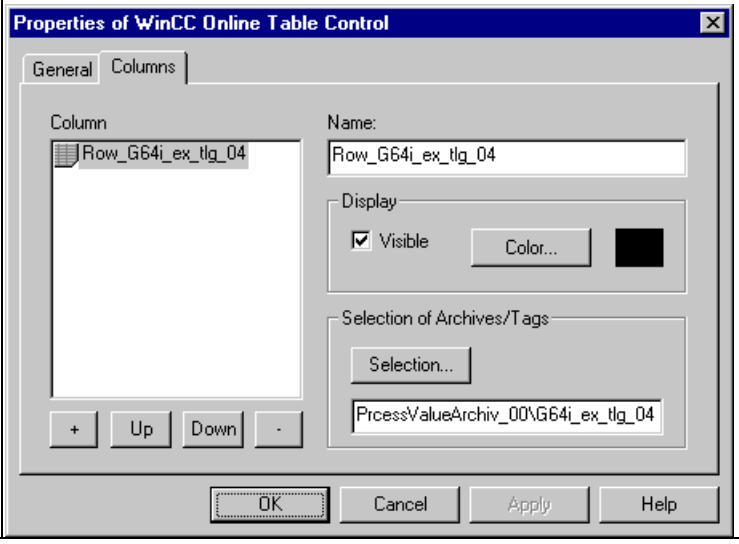
步骤	过程：创建过程值归档
1	在*变量管理器*中，创建两个变量。 一个变量包含由模拟器提供的数值总和。在本实例中，它是变量 *G64I_ext_lgl_04*。另一个变量用于在超出限制值时进行归档。在本实例中，它是变量 *G64i_ex_tlg_08*。
2	在*变量记录*编辑器中创建一个*过程值归档*。 这通过*归档向导*来完成。在本实例中，归档已经命名为 *AZ_ProcessValueArchive_00*。选择变量 *G64i_ex_tlg_08* 进行归档。
3	设置过程值归档的属性。 在*归档参数*标签中将归档大小设置为 *25* 个数据记录。其余选项保留缺省设置。

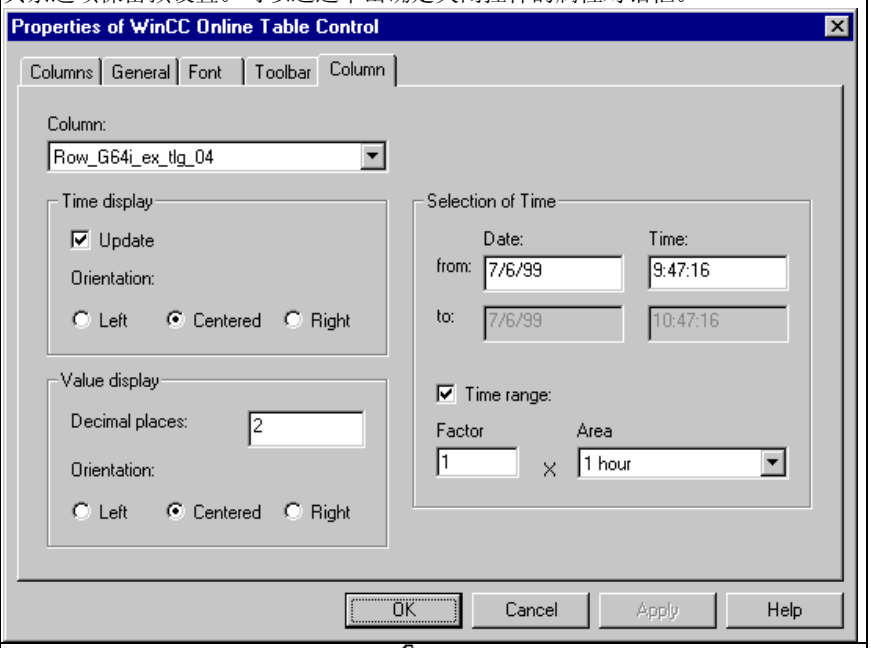
步骤	过程：创建过程值归档
4	设置*归档变量*的属性。 在*归档变量*标签中，选择*非周期*作为*归档类型*。 这种归档类型在每次归档变量改变时进行归档。 其余选项保留缺省设置。

表格显示的组态

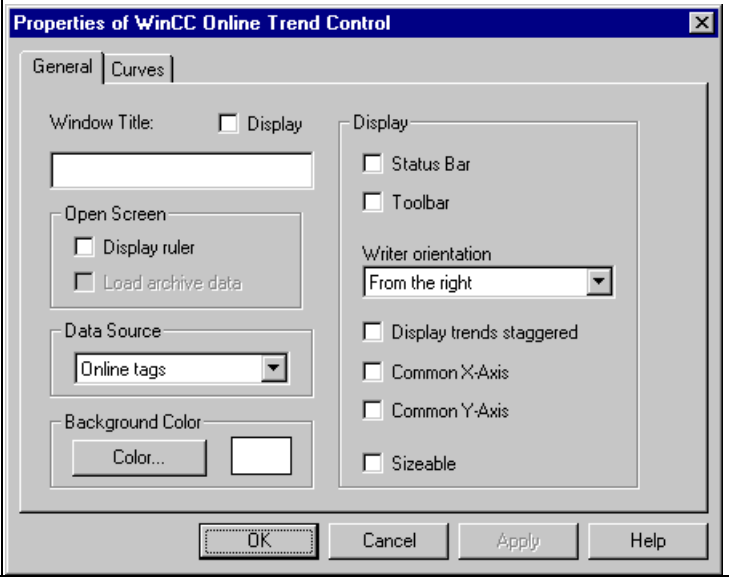
步骤	过程：表格显示的组态
1	在 *图形编辑器*中创建一个新画面。在本实例中，它是画面 *ex_3_chapter_01b.pdl*。
2	组态用于显示表格的控件。它是 *WinCC 在线表格控件*。从*对象选项板的控件*选择菜单中选择它，然后将其置于画面中。 

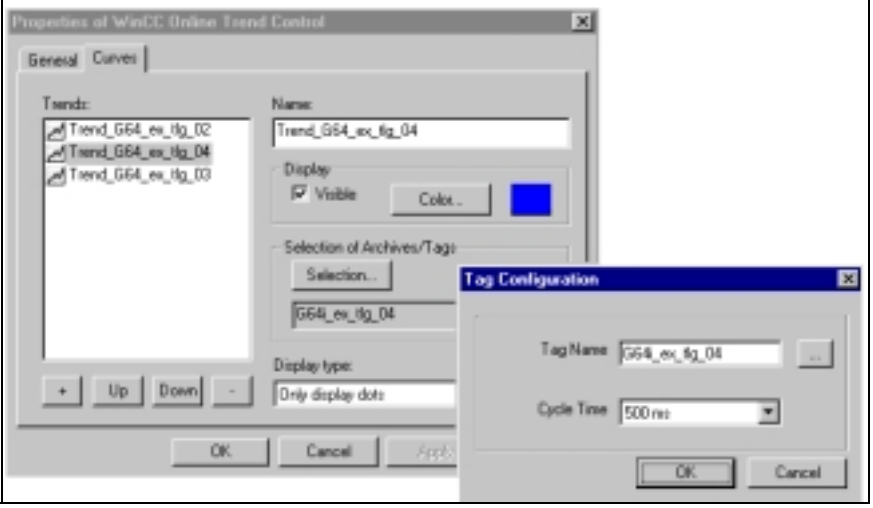
步骤	过程：表格显示的组态
3	将控件置于画面中之后，会自动打开 *WinCC 在线表格控件属性*对话框。 在*常规信息*标签中，可以指定控件的标题以及它如何进行标记。在本实例中，撤消选择*显示*复选框。仍然输入*窗口标题*。在稍后创建的 *C 动作*中，此窗口标题用于注明相应的控件。使用先前创建的归档的名称 *AZ_ProcessValueArchive_00*。 通过颜色按钮，将表格窗口的*背景色*设置为白色。 在本实例中，撤消选择*显示*域中的所有复选框。因此，不会显示任何*工具栏*与*状态栏*。 

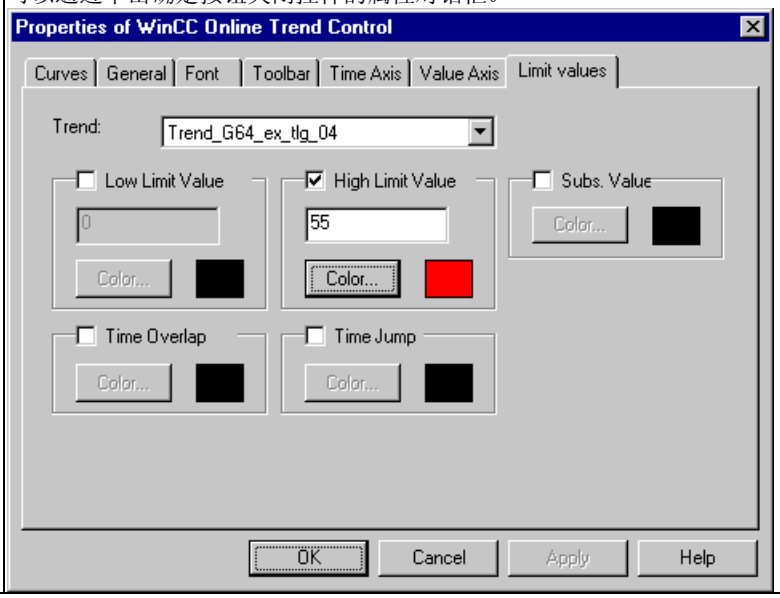
步骤	过程：表格显示的组态
4	在*列*标签中，具体指定要显示的列。对于本实例，只需要一列。 有一列已经创建好。将此列重命名为 *Row_G64i_ex_tlg_08*。 通过*选择*按钮，可以把要显示的*归档变量*分配给该列。在本实例中，将先前创建的 *AZ_ProcessValueArchive_00* 归档的*归档变量 G64i_ex_tlg_08* 分配给该列。 可以通过单击*确定*关闭控件的属性对话框。 

步骤	过程：表格显示的组态
5	列的特殊属性设置。为此，可以使用一个扩充的属性对话框。此对话框通过 D 控件来打开。另一方面，先前描述的属性对话框通过在按下 CTRL 键的同时利用 D 控件来打开。 扩充的属性对话框除了包含已经提及的*常规信息*与*列*标签外，还包含三个附加的标签。在本实例中，只在*列*标签内进行设置。 将*时间显示*的*格式*更改为 *hh:mm:ss*。将*时间显示*与*数值显示*的*方向*设置为*居中*。在*时间选择*域中，*时间范围*复选框仍然选中，但是将*范围*更改为 *1 X 1 小时*。 其余选项保留预设置。可以通过单击*确定*关闭控件的属性对话框。 
6	如果超出设置值就写入变量。这通过 R → *属性* → *几何结构* → *位置 X* 处的控件的 C 动作来完成。由于需要一个触发，所以已经在属性处创建了 C 动作。属性本身并没有动态化。

趋势显示的组态

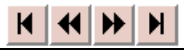
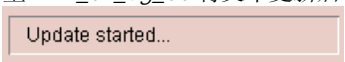
步骤	过程：趋势显示的组态
1	组态用于显示趋势图的控件。它是 *WinCC 在线趋势控件*。从*对象选项板的控件*选择菜单中选择它，然后将其置于画面中。将控件置于画面中之后，将会自动打开其组态对话框。 在*常规信息*标签中，指定控件不带标题栏显示。为此，撤消选择*显示*复选框。 在*数据源*域中，设置*在线变量*。这样就不必创建单独的归档来显示变量按时间顺序的进程。只要设置期望的内部变量或外部变量。显示所需的变量值的临时缓冲由控件本身来完成。 通过*颜色*按钮，将趋势窗口的*背景色*设置为白色。 在*显示*域中，撤消选择所有的复选框。 

步骤	过程：趋势显示的组态
2	在*趋势*标签中，详细规定要显示的趋势图。 将趋势重命名为 *Trend_G64i_ex_tlg_04*。将趋势的*颜色*设置为*蓝色*。至于*显示类型*，则设置*填充区域内插趋势*。 通过*选择*按钮打开*变量组态*对话框。在此对话框中，设置要显示的变量。此处，它不是*归档变量*，而是所谓的*在线变量*(内部变量或外部变量)。此外，还设置*周期时间*。在本实例中，设置周期为 *500 ms* 的变量 *G64i_ex_tlg_04*。 可以通过单击*确定*关闭控件的属性对话框。 

步骤	过程：趋势显示的组态
3	超出限制值用颜色进行标识。这只能在扩充的属性对话框中进行。此对话框通过 D 控件来打开。 扩充的属性对话框除了包含已经提及的*常规信息*和*趋势*标签外，还包含五个附加的标签。 在*数值*轴标签内，撤消选择*范围选择*域中的*自动*复选框。将范围固定设置为从-*100* 至 *100*。 在*限制值*标签内，组态*上限值*。将该值设为 *55*。将限制值的颜色设为红色。 可以通过单击*确定*按钮关闭控件的属性对话框。 

工具栏和状态栏的组态

步骤	过程：工具栏和状态栏的组态
1	在变量管理器中，创建两个*二进制变量*类型的内部变量。在本实例中，使用变量 *BINi_ex_tlg_06* 与 *BINi_ex_tlg_07*。
2	为了控制更新，组态一个*智能对象* → *状态显示*。在本实例中，使用对象*状态显示 3*。 通过其*组态对话框*，将对象与变量 *BINi_ex_tlg_06* 相连接并且一旦改变就触发。 创建状态 *0* 和 *1*。在本实例中，将位图 *stop tlg.bmp* 和 *stop go tlg.bmp* 分配给这些状态。通过单击*确定*按钮可以退出对象的*组态对话框*。 对于刚组态的对象*状态显示 3*，在*事件* → *鼠标* → *按下左键*处创建一个 *C 动作*。该 *C 动作*模拟从控件的标准工具栏上按下停止/执行按钮。此外，对变量 *BINi_ex_tlg_06* 的状态求反以显示控件更新后已改变的的状态。变量值为 0 对应一个已激活的更新。 由于当画面打开时总是会激活趋势窗口的更新，所以在打开画面时变量 *BINi_ex_tlg_06* 的状态总是为零。这通过在画面对象 *ex_3_chapter_01b.pdl* 的*事件* → *其它* → *打开画面*处的 *C 动作*来实现。这样就将该变量的状态设置为 *0*。
3	按步骤 2 的描述，组态第二个*智能对象* → *状态显示*。在本实例中，使用对象*状态显示 2*。此状态显示控制表格的编辑能力。 将该对象与变量 *BINi_ex_tlg_07* 相链接。相应地使用不同的位图(*Edit.gif / Edit inv.gif*)。 在*事件* → *鼠标* → *按下左键*处，创建一个 *C 动作*。该 C 动作模拟按下控件的标准工具栏上的编辑按钮。此外，对变量 *BINi_ex_tlg_07* 的状态求反，以显示已改变的表格编辑能力的状态。变量值为零对应于取消激活的编辑能力。 由于当画面打开时表格窗口的编辑能力总是被取消激活，所以在打开画面时变量 *BINi_ex_tlg_07* 的状态总是为零。这通过在画面对象 *ex_3_chapter_01b.pdl* 的*事件* → *其它* → *打开画面*处扩展 C 动作来实现。插入一个语句，用于将变量的状态设置为 *0*。

步骤	过程：工具栏和状态栏的组态
4	为了当更新停止时在归档中进行浏览，需要复制四个控件的标准工具栏的浏览按钮。 为了实现上述任务，组态四个 *Windows 对象* → *按钮*；在本实例中，它们是对象*按钮 4*、*按钮 7*、*按钮 8* 和*按钮 11*。 为其中的每个对象，在*事件* → *鼠标* → *鼠标动作*处创建一个 *C 动作*。这些动作模拟按下标准工具栏上的按钮。 此外，需要一个*智能对象* → *图形对象*来将其本身置于这些按钮上，并使它们在启动更新时不可操作。在本实例中，它就是*图形对象 2*。此对象显示四个处于不可操作状态的按钮*(Pfeile dis.bmp)*。在*属性* → *其它* → *显示*处，创建一个*动态对话框*。此对话框根据变量 *BINi_ex_tlg_06* 控制对象的可见性，该变量包含有关控件更新的信息。 
5	需要一个附加的*智能对象* → *图形对象*。此对象用于在激活表格的编辑能力的情况下使停止/执行按钮不可操作。在本实例中，使用*图形对象 1*。 在*属性* → *其它* → *显示*处创建一个*动态对话框*，用于在变量 *BINi_ex_tlg_07* 接收到状态 *TRUE*(即表格可以编辑)时使对象可见。至于要显示的画面，本实例使用 *stop dis tlg.bmp*。该对象必须恰好放置在停止/执行按钮之上。
6	为了显示状态栏，组态一个 *Windows 对象* → *按钮*。在本实例中，它是对象*按钮 10*。 对于文本显示，使用按钮，因为它可以很容易配备 3D 边框。因此，无需任何附加的对象。 为*按钮 10*，在*属性* → *字体* → *文本*处创建一个*动态对话框*。该对话框根据变量 *BINi_ex_tlg_06* 将文本*更新启动*或*更新停止*返回给属性。 

WinCC 在线表格控件(控件 1)的 C 动作

```
#include "apdefap.h"
 long _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty
{

if  (GetTagDouble("G64i_ex_tlg_04") >= 55)
      {
      SetTagDouble("G64i_exTlg_08",GetTagDouble("G64i_ex_tlg_04"));
      }

return 50;
}
```

- 读取变量 G64I_ex_tlg_04，并检查当前值是否大于 55。
- 如果值大于 55，则将值写入变量 D64I_ex_tlg_08 中。

编辑按钮(状态显示 2)的 C 动作

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszl
{

TlgTableWindowPressEditRecordButton("AZ_ProcessValueArchive_00");

SetTagBit("BINi_ex_tlg_07",(SHORT)!GetTagBit("BINi_ex_tlg_07"));

SetTagBit("BINi_ex_tlg_06",TRUE);

}
```

- 调用标准函数 *TlgTableWindowPressEditButton* 与按下控件的标准工具栏上的编辑按钮具有相同的效果。将文本分配给该函数，以便允许它识别要访问的控件。该文本就是在组态控件时已指定的窗口标题。在本实例中，它是文本 *AZ_ProcessValueArchive_00*。
- 对变量 *BINi_ex_tlg_07* 求反来存储表格编辑能力的当前状态。
- 将变量 *BINi_ex_tlg_06* 设置为真，以便停止更新。

浏览按钮启动(按钮 4)的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

TlgTableWindowPressFirstButton("AZ_ProcessValueArchive_00");

}
```

- 调用此标准函数与按下标准工具栏上的*第一个数据记录*按钮具有相同的效果。其它按钮所使用的函数是：
- *TlgTrendWindowPressPrevButton*
- *TlgTrendWindowPressNextButton*
- *TlgTrendWindowPressLastButton*
- *TlgTableWindowPressOpenTimeSelectDlgButton*
- *TlgTableWindowPressStartStopButton*

注意：
对于 *WinCC 在线表格控件*的标准工具栏上的每个按钮，可以使用相应的用于模拟按下按钮的标准函数。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 必须根据需要修改要归档的变量。
- 启动归档操作的事件必须由用户进行定义。为此，必须创建一个项目函数。
- 所需元素的外观可以根据各自的需要进行修改。这同样适用于状态栏。

4.1.4 用户定义的表格布局(ex_3_chapter_01c.pdl)

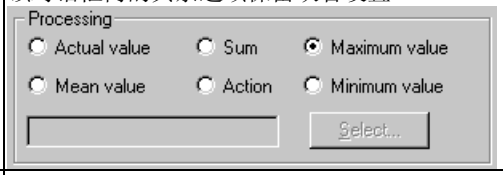
任务定义

周期性获取一个过程值。对于持续 10 秒的时间间隔，将确定平均值、最大值和最小值。这些数值将存储在项目内部的归档中。
所存储的值将在表格中显示。该表格在*图形编辑器*中是用户定义的。如果使用*变量记录*标准工具不能实现表格布局但却需要表格布局时，就需要这样做。

概念的实现

为了归档数据，在*变量记录*编辑器中创建周期连续的过程值归档。
为了进行图形显示，可根据要显示信息的类型，为每个表格行使用一个*标准对象* → *静态文本*或一个*智能对象* → *I/O 域*。
通过 API 函数，可从相应归档的数据库表格中读取数据。

创建过程值归档

步骤	过程：创建过程值归档
1	在变量管理器中，创建三个*浮点数 64 位 IEEE 754* 类型的变量。在本实例中，它们是变量 *G64i_ex_tlg_05*、*G64i_ex_tlg_06* 和 *G64i_ex_tlg_07*。并将已归档的数值写入这些变量中。
2	使用归档向导创建一个新的*过程值归档*。在本实例中，归档名称为 *ZK_ProcessValueArchive_02*。 作为将要归档的变量，变量 *G64i_ex_tlg_04* 已被选择了三次。
3	在过程值归档的属性对话框中，将归档的大小设置为 *100* 个数据记录。其余选项保留缺省设置。
4	在第一个过程变量的属性对话框中，输入*最大值*作为*常规信息*标签中的*归档变量的名称*。在同时将*已归档的数值写入变量*域内，设置变量 *G64i_ex_tlg_07*。通过该变量，可以使用 *C 动作*来影响数值的归档。在变量一旦改变时，这也可通过触发该 *C 动作*来完成。 在*参数*标签中，将*周期*域内的*采集*设置为 *500 毫秒*并且将*归档*设置为 *20*500 毫秒*。在*过程*域内，选择*最大值*选项钮。也就是说每 500 毫秒采集一次所选变量并且每 10 秒进行一次归档。已归档的数值是在 10 秒周期内所出现的最大数值。 该对话框内的其余选项保留缺省设置。 
5	对于另两个归档变量，根据步骤 3 和 4 进行设置。 然而，对于*处理*，选择*最小值*和*平均值*选项钮。相应地分配*归档变量的名称*。

图形编辑器中的实现

<table>
<tr><th>步骤</th><th>过程：图形编辑器中的实现</th></tr>
<tr><td>1</td><td>在变量管理器中，创建两个二进制变量类型的内部变量。在本实例中，它们是变量 BINi_ex_tlg_06 和 FLAG_TableGetOutputValue。</td></tr>
<tr><td>2</td><td>创建一个项目函数，它指示变量记录将归档数据传送给其它函数(回叫功能)。该函数由每个数据记录调用一次，并以特殊结构类型的形式包含了该数据记录的信息。所传送的数据被存储在该结构类型的一个静态数组中。
在本实例中，使用函数 Enumerate Super Archive Data 和 Get Archive Data Callback。
本实例使用两个外部 C 变量。
• extern int dwSize
• extern WORD wOffset
必须在项目开始时创建这些变量。为此，使用一个独立的项目函数。在起始画面 ex_0_startpicture_00.PDL 的事件 → 其它 → 打开画面处的 C 动作中插入对该函数的调用。在本实例中，该函数称为 CreateExternal。</td></tr>
<tr><td>3</td><td>创建一个新的画面，在本实例中它是画面 ex_3_chapter_01b.pdl。
用户定义将能显示 10 行。对于第一列中数据的显示，使用 10 个标准对象 → 静态文本来显示日期和时间。对于其余列，使用智能对象 → I/O 域。
对于第一列中的对象名，使用名称静态文本 1 至静态文本 10，其中的编号定义了对应的行。由底部至顶部进行编号，因为最后一行包含了最新的数据。
I/O 域使用数字代码作为名称。第一位数字表示列编号，第二位数字表示行编号。
<table>
<tr><th>Date/Time</th><th>Mean Value</th><th>Minimum</th><th>Maximum</th></tr>
<tr><td>06.07.1999 10:17:27</td><td>000,00</td><td>000,00</td><td>000,00</td></tr>
<tr><td>06.07.1999 10:17:37</td><td>000,00</td><td>000,00</td><td>000,00</td></tr>
<tr><td>06.07.1999 10:17:47</td><td>000,00</td><td>000,00</td><td>000,00</td></tr>
<tr><td>06.07.1999 10:17:57</td><td>000,00</td><td>000,00</td><td>000,00</td></tr>
<tr><td>06.07.1999 10:18:07</td><td>000,00</td><td>000,00</td><td>000,00</td></tr>
<tr><td>06.07.1999 10:18:17</td><td>000,00</td><td>000,00</td><td>000,00</td></tr>
<tr><td>06.07.1999 10:18:27</td><td>000,00</td><td>000,00</td><td>000,00</td></tr>
<tr><td>06.07.1999 10:18:37</td><td>000,00</td><td>000,00</td><td>000,00</td></tr>
<tr><td>06.07.1999 10:18:47</td><td>000,00</td><td>000,00</td><td>000,00</td></tr>
<tr><td>06.07.1999 10:18:57</td><td>000,00</td><td>000,00</td><td>000,00</td></tr>
</table></td></tr>
</table>

步骤	过程：图形编辑器中的实现
4	对于每个*静态文本*，在*属性* → *字体* → *文本*处，创建一个 *C 动作*。该动作将根据其自身的对象编号，从回叫函数中读取所要显示的日期。一旦改变变量 *FLAG_TableGetOutputValue* 就触发该函数。如果归档接收到新的数据且该数据已被读出，则该变量的状态将发生改变。 在*属性* → *输出/输入* → *输出值*处，以相同的方式为每个 *I/O 域*创建一个 *C 动作*。该动作也具有从回叫函数中读取分配给对象的数据记录的任务。
5	为了控制更新，组态一个*智能对象* → *状态显示*。在本实例中，使用对象*状态显示 3*。 通过其*组态对话框*，将对象与变量 *BINi_ex_tlg_06* 相连接并且一旦改变就触发。创建状态 *0* 和 *1*，并将相应的画面分配给每个状态。在本实例中，使用位图 *stop go tlg.gif* 和 *stop tlg.gif*。
6	在*事件* → *鼠标* → *按下左键*处，创建一个 *C 动作*对变量 *BINi_ex_tlg_06* 的状态求反。状态 *TRUE* 表示更新已经启动。 由于画面打开时总是启动表格窗口的更新，所以在画面打开时该变量的状态为 *TRUE*。这通过在画面对象 *ex_3_chapter_01c.pdl* 的*事件* → *其它* → *打开画面*处的 *C 动作*来实现。该 *C 动作*将变量的状态设置为 *TRUE* 并一次读取归档。
7	为 *Statusdisplay3*，在*属性* → *几何结构* → *宽度*处创建一个 *C 动作*。该动作根据变量 *BINi_ex_tlg_06* 的状态读取归档，并对变量 *FLAG_TableGetOutputValue* 求反来触发表格的更新。一旦改变变量 *G64i_ex_tlg_07* (已归档的数值存储在其中)，则触发该 *C 动作*。如果该数值与先前的数值相同，则该动作也会影响下一个归档。为此，将变量 *G64i_ex_tlg_07* 设置为一个数值，在每次运行 *C 动作*后所要归档的过程值不能达到该数值。
8	当更新停止时为了在归档中进行浏览，需要浏览按钮。 为了实现上述任务，组态四个 *Windows 对象* → *按钮*；在本实例中，它们是对象*按钮 4*、*按钮 7*、*按钮 8* 和*按钮 11*。
9	在*事件* → *鼠标* → *鼠标动作*处，创建 *C 动作*。这些动作将新的数值写入外部 C 变量 *dwOffset*。此外，对触发变量 *FLAG_TableGetOutputValue* 求反来获得显示的更新。 此外，还需要一个*智能对象* → *图形对象*来将其本身置于这些按钮上，并使它们在启动更新时不可操作。在本实例中，这可通过*图形对象 2* 来实现。由该对象所显示的位图表示处于非运行状态的四个按钮*(Pfeile dis.bmp)*。在*属性* → *其它* → *显示*处，创建一个至一旦改变即会触发的变量 *BINi_ex_tlg_06* 的*变量连接*。

用于读取归档的项目函数

```
#include "apdefap.h"
BOOL EnumerateSuperArchiveData()
{

extern DWORD   dwSize;
BOOL           fRet;
TLG_GETARCHIVDATA      GAD;
CMN_ERROR      Error;
LPTSTR  lpszArchivName = "ZK_ProcessValueArchive_02";
LPTSTR  lpszVarName3 = "MaximumValue" ;
LPTSTR  lpszVarName2 = "MinimumValue" ;
LPTSTR  lpszVarName1 = "MeanValue" ;
SYSTEMTIME     sysFrom;
SYSTEMTIME     sysTo;
time_t  Time;
struct tm*     TimeStruct;

time(&Time);
TimeStruct = localtime(&Time);

sysTo.wYear   = (WORD)(TimeStruct->tm_year+1900);
sysTo.wMonth  = (WORD)(TimeStruct->tm_mon+1);
sysTo.wDay    = (WORD)(TimeStruct->tm_mday);
sysTo.wHour   = (WORD)(TimeStruct->tm_hour);
sysTo.wMinute = (WORD)(TimeStruct->tm_min);
sysTo.wSecond = (WORD)(TimeStruct->tm_sec);

sysFrom.wYear   = 1997;
sysFrom.wMonth  = 1;
sysFrom.wDay    = 1;
sysFrom.wHour   = 0;
sysFrom.wMinute = 0;
sysFrom.wSecond = 0;

Call(&GAD,(PVOID)0);

if (TLGConnect( NULL , &Error )==FALSE) {
        printf("Error: %s\r\n",Error.szErrorText);
        return FALSE;
        }
else {
        fRet=TLGGetArchivData(lpszArchivName,lpszVarName1,
                sysFrom,sysTo,GetArchiveDataCallback,
                (PVOID)1,0,&Error);
        if (fRet==FALSE)
                printf("Error: %s\r\n",Error.szErrorText);

        fRet=TLGGetArchivData(lpszArchivName,lpszVarName2,
                sysFrom,sysTo,GetArchiveDataCallback,
                (PVOID)2,0,&Error);
        if (fRet==FALSE)
                printf("Error: %s\r\n",Error.szErrorText);

        fRet=TLGGetArchivData(lpszArchivName,lpszVarName3,
                sysFrom,sysTo,GetArchiveDataCallback,
                (PVOID)3,0,&Error);
        if (fRet==FALSE)
                printf("Error: %s\r\n",Error.szErrorText);

        Call(&GAD,(PVOID)4);
        dwSize=GAD.dwFlags;
        TLGDisconnect( NULL );
        return TRUE;
        }
}
```

- 定义启动与结束时间的数值，在这两个时间值之间从归档中读取数据。为启动时间设置一个固定的时间，将结束时间设置为当前系统时间。
- 回叫函数的初始化是通过*调用*帮助函数来实现的。该函数调用其参数 *lpUser* 为数值 0 的函数 *GetArchiveDataCallback*。
- 建立与*变量记录*的连接。如果未成功，则函数中止。
- 通过函数 *TLGGetArchiveData* 从归档变量*最大值*、*最小值*和*平均值*中读取数值。
- 确定所读取数值的编号。这是通过帮助函数*调用*来完成的，该函数调用其参数 *lpUser* 为数值 4 的函数 *GetArchiveDataCallback*。
- 中止与*变量记录*的连接。

回叫函数

```
BOOL GetArchiveDataCallback (PTLG_GETARCHIVDATA      lpGAD, PVOID   lpUser)
{

static int i1 = 0;
static int i2 = 0;
static int i3 = 0;

WORD wRecordNumber;
WORD wColumnNumber;

static TLG_GETARCHIVDATA GAD1[100];
static TLG_GETARCHIVDATA GAD2[100];
static TLG_GETARCHIVDATA GAD3[100];

int User;
User=(int)lpUser;

if ((User==1)||(User==2)||(User==3))
{

switch(User)
{
case 1 : if (i1<=100) {GAD1[i1]=*lpGAD; i1++;} break;
case 2 : if (i2<=100) {GAD2[i2]=*lpGAD; i2++;} break;
case 3 : if (i3<=100) {GAD3[i3]=*lpGAD; i3++;} break;
}//switch

}//if ((User==1)||(User==2)||(User==3))

if (User==0)
{

i1=0;
i2=0;
i3=0;

memset(&GAD1[0],0,sizeof(TLG_GETARCHIVDATA)*100);
memset(&GAD2[0],0,sizeof(TLG_GETARCHIVDATA)*100);
memset(&GAD3[0],0,sizeof(TLG_GETARCHIVDATA)*100);

}//if (User==0)

if (User==4)
{
lpGAD->dwFlags=i1;
}//if (User==4)

if (User==7)
{

wRecordNumber=lpGAD->stTime.wMonth;
wColumnNumber=lpGAD->stTime.wDay;

switch(wColumnNumber)
{
case 0  : lpGAD->stTime.wYear = GAD1[wRecordNumber].stTime.wYear;
          lpGAD->stTime.wMonth = GAD1[wRecordNumber].stTime.wMonth;
          lpGAD->stTime.wDay = GAD1[wRecordNumber].stTime.wDay;
          lpGAD->stTime.wHour = GAD1[wRecordNumber].stTime.wHour;
          lpGAD->stTime.wMinute = GAD1[wRecordNumber].stTime.wMinute;
          lpGAD->stTime.wSecond = GAD1[wRecordNumber].stTime.wSecond;
          break;
case 1  : lpGAD->doValue=GAD1[wRecordNumber].doValue;
          break;
case 2  : lpGAD->doValue=GAD2[wRecordNumber].doValue;
          break;
case 3  : lpGAD->doValue=GAD3[wRecordNumber].doValue;
          break;
default : break;
}//switch

}//if (User==7)

return TRUE;
}
```

- 三个静态计数器变量的声明：i1、i2 和 i3。
- 声明三个包含了 TLG_GETARCHIVEDATA 类型结构的静态数组。归档值将被存储在这些数组中。
- 如果传递参数 *lpUser* 包含数值 1、2 或 3，则该函数已由变量记录所调用。在这种情况下，将所传送的结构 *lpGAD* 存储在相应的数组中。
- 如果传递参数 *lpUser* 数值为 0，则表示这是初始化运行。计数器变量被复位至 0，并为数组保留了存储空间。
- 如果传递参数 *lpUser* 数值为 4，则请求所存储的数值的编号。于是此编号作为结构成员 *dwFlags* 存储在所传递的结构中。
- 如果传递参数 *lpUser* 数值为 7，则请求 *I/O 域*或*静态文本*中所存储的变量值。从表格的哪个位置请求变量，将在所传送的结构中由结构成员 *stTime.wMonth* 和 *stTime.wDay* 来决定。

静态文本的 C 动作

```
#include "apdefap.h"
 char* _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

int nRecordNumber;
WORD wColumnNumber;
extern WORD wOffset;
char szObject[5];
TLG_GETARCHIVDATA GAD;
PVOID lpUser7 = 7;
char szTime[20] = "";
int nObjectNumber;
extern DWORD dwSize;

wColumnNumber=0;

nObjectNumber=atoi(lpszObjectName+15);
nRecordNumber=(dwSize-nObjectNumber-wOffset);

if (nRecordNumber<0) return "";

GAD.stTime.wMonth=(WORD)nRecordNumber;
GAD.stTime.wDay=wColumnNumber;

GetArchiveDataCallback(&GAD,lpUser7);

sprintf(szTime,"%02d.%02d.%d %02d:%02d:%02d",
GAD.stTime.wDay,GAD.stTime.wMonth,GAD.stTime.wYear,GAD.stTime.wHour,
GAD.stTime.wMinute,GAD.stTime.wSecond);

return szTime;

}
```

- 为将要传送的 *GAD* 结构的结构成员 *stTime.wMonth* 和 *stTime.wDay* 提供列编号或计算得出的数据记录编号。对象名包含了有关数据记录编号的信息。
- 由传递参数 *lpUser* 的数值 7 调用函数 *GetArchiveDataCallback*，即请求一个数值。
- 日期值将存储在所传送的结构 *GAD* 的结构成员 *stTime* 中。由此形成所要显示的文本。通过 *return* 将该文本返回至属性。

I/O 域的 C 动作

```
#include "apdefap.h"
 double _main(char* lpszPictureName, char* lpszObjectName, char* lpszProper
{

extern dwSize;
int nRecordNumber;
WORD wColumnNumber;
extern WORD wOffset;
char szObject[5];
TLG_GETARCHIVDATA GAD;
PVOID lpUser = 7;

strcpy(szObject,"");
sprintf(szObject,"%c",lpszObjectName[0]);
wColumnNumber=(WORD)atoi(szObject);

nRecordNumber=(dwSize-atoi(lpszObjectName+1)-wOffset);

if (nRecordNumber<0) return 0;

GAD.stTime.wMonth=(WORD)nRecordNumber;
GAD.stTime.wDay=wColumnNumber;

GetArchiveDataCallback(&GAD,lpUser);

return GAD.doValue;

}
```

- 列编号与行编号均从对象名称中确定。为将要传送的 *GAD* 结构的结构成员 *stTime.wMonth* 和 *stTime.wDay* 提供列编号或计算得出的数据记录编号。
- 由传递参数 *lpUser* 的数值 7 调用函数 *GetArchiveDataCallback*，即请求一个数值。
- 变量值存储在所传送结构 *GAD* 的结构成员 *doValue* 中并用作返回值。

注意：
工具栏中包含了如下显示的这个用于设置表格参数的按钮。通过此按钮，访问用于设置各种表格元素颜色的对话框。与此有关的简短描述可参见实例颜色对话框 (ex_3_chapter_01c)。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 必须修改要显示的归档数据以满足需求。
- 修改表格布局以满足需求。如果需要一个不同的列编号或行编号，则也必须修改 C 动作与项目函数。

4.1.5 归档二进制变量(ex_3_chapter_01d.pdl)

任务定义

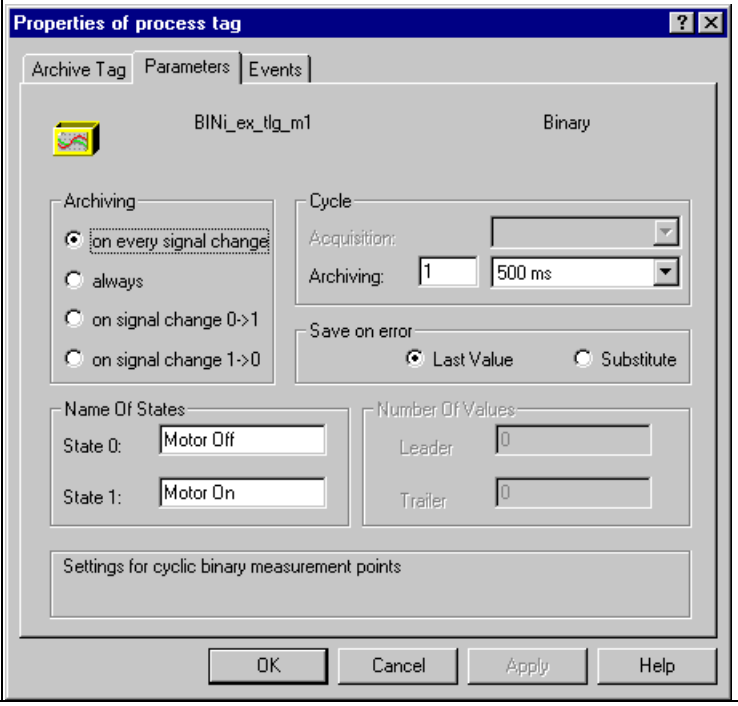
三台电机的开关操作要存储在一个归档中。如果用选择了一台电机，则用表格来显示一天中最后几次开关操作。该表格只显示所选电机的状态。

概念的实现

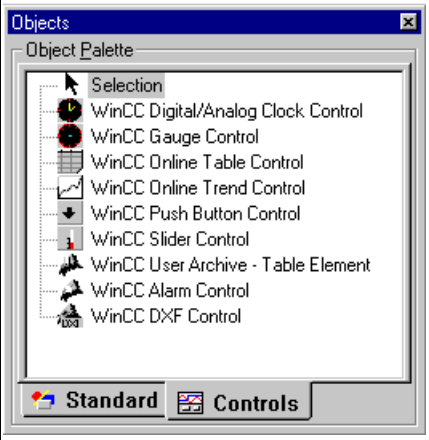
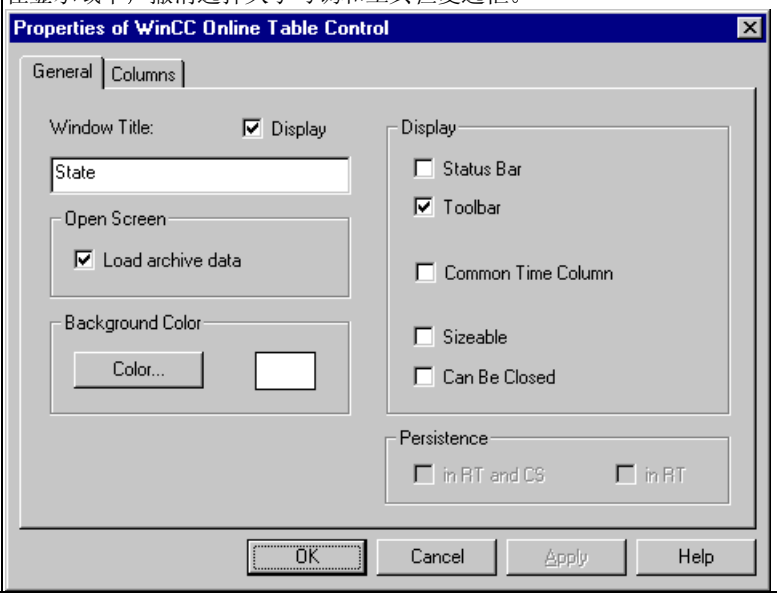
为了对所要显示的数据进行归档，在*变量记录*编辑器中创建一个周期连续的过程值归档。每个变量在单独的列中显示。
为了实现图形显示，用 WinCC 在线表格控件根据所选电机显示相应的列。

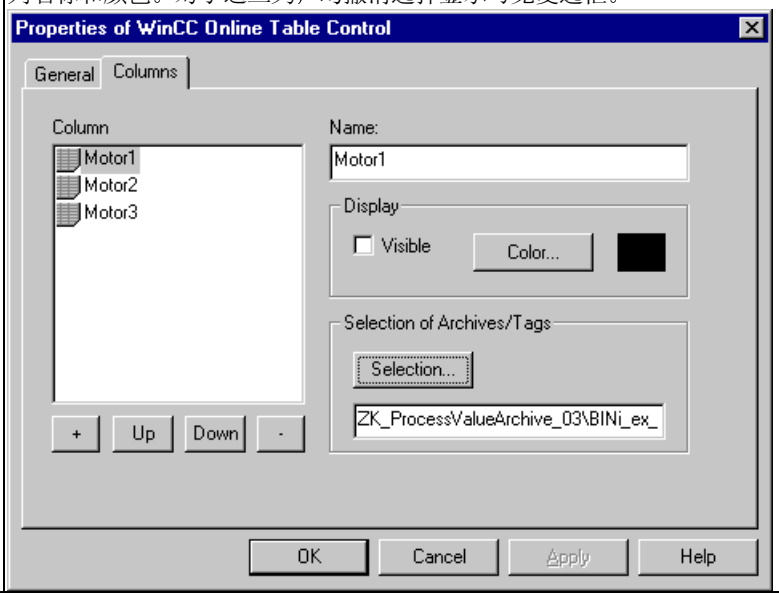
创建过程值归档

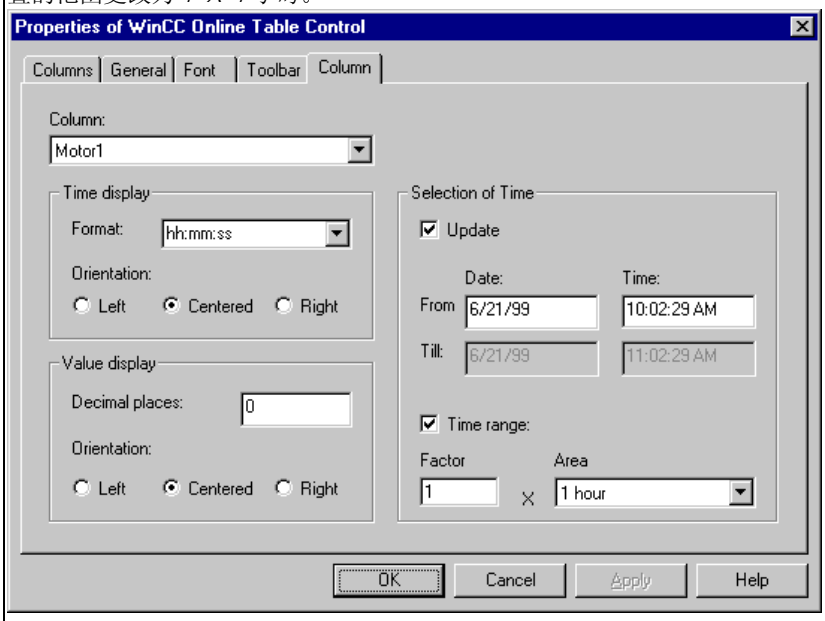
步骤	过程：创建过程值归档
1	在变量管理器中，创建四个变量。在本实例中，它们是*二进制变量*类型的变量 *BINi_ex_tlg_m1*、*BINi_ex_tlg_m2* 与 *BINi_ex_tlg_m3*，以及无符号的 8 位数类型的变量 *U08i_ex_tlg_00*。
2	用归档向导创建一个新的*过程值归档*。在本实例中，归档已经命名为 *ZK_ProcessValueArchive_03*。 对于归档，选择变量 *BINi_ex_tlg_m1*、*BINi_ex_tlg_m2* 和 *BINi_ex_tlg_m3*。
3	在过程值归档的属性对话框中，将归档大小设置为 *40* 个数据记录。其余选项保留缺省设置。

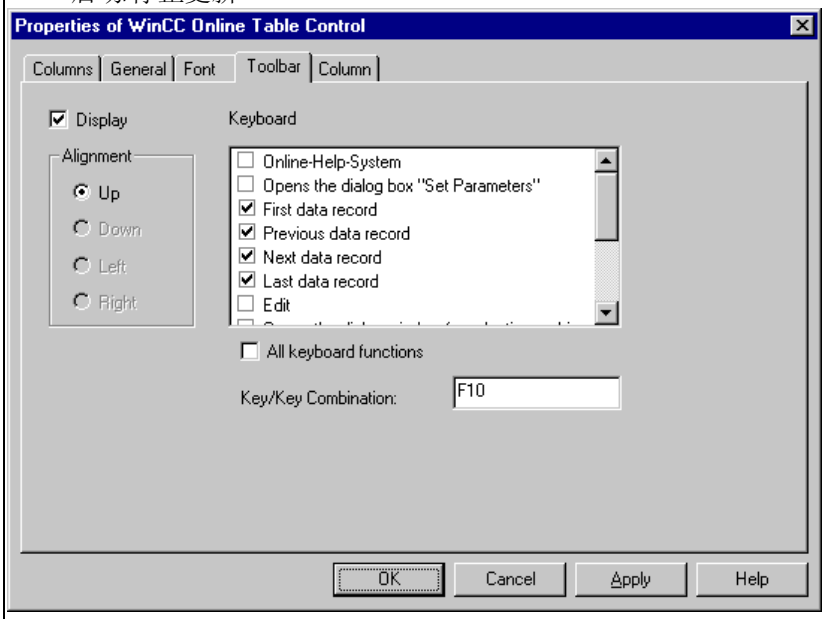
步骤	过程：创建过程值归档
4	在第一个过程变量的属性对话框中，在*参数*标签的*归档*域内选择条目*每次信号改变*。在*状态的名称*域中，为*状态 0* 输入*电机关闭*并为*状态 1* 输入*电机打开*。至于周期，设置*归档 1*500 ms*。 其余选项保留缺省设置。 其余的归档变量执行相同的设置。 

表格显示的组态

步骤	过程：表格显示的组态
1	创建一个新画面，在实例中它是画面 *ex_3_chapter_01d.pdl*。
2	组态用于显示表格的控件。它是 *WinCC 在线表格控件*。从*对象选项板的控件*选择菜单中选择它，然后将其置于画面中。 
3	将控件置于画面中之后，会自动打开 *WinCC 在线表格控件属性*对话框。 在*常规信息*标签中，可以指定控件的标题以及它如何进行标记。输入“状态”作为窗口标题。通过*颜色*按钮，将表格窗口的*背景色*设置为白色。 在*显示*域中，撤消选择*大小可调*和*工具栏*复选框。 

步骤	过程：表格显示的组态
4	在*列*标签中，具体指定要显示的列。本实例需要三列。 有一列已经创建好。将其重命名为电机 1。 通过*选择*按钮，可以把要显示的*归档变量*分配给该列。在本实例中，将先前创建的归档 *ZK_ProcessValueArchive_03* 的*归档变量 BINi_ex_tlg_m1* 分配给该列。 再添加两列。将变量 *BINi_ex_tlg_m2* 与 *BINi_ex_tlg_m3* 分配给它们，并修改其列名称和颜色。*对于这三列*，均撤消选择*显示可见*复选框。 

步骤	过程：表格显示的组态
5	列的特殊属性设置。为此，可以使用一个扩充的属性对话框。此对话框通过 D 控件来打开。另一方面，先前描述的属性对话框通过在按下 CTRL 键的同时 D 控件来打开。 扩充的属性对话框除了包含已经提及的*常规信息*与*列*标签外，还包含三个附加的标签。在*列*标签中，进行下列设置。 将*时间显示*的*格式*设置为 *hh:mm:ss*。将*时间显示*与*数值显示*的*方向*设置为*居中*，并将小数位设为 0。在*时间选择*域中，仍然选中*时间范围*复选框，但是将设置的范围更改为 *1 X 1 小时*。 

步骤	过程：表格显示的组态
6	在*工具栏*标签内，在*键功能*子条目处选择下列复选框： • 第一条数据记录 • 前一条数据记录 • 下一条数据记录 • 最后一条数据记录 • 选择时间范围 • 启动/停止更新 
7	由于只显示一列，所以可以在*字体*标签中将字体大小设置为 13.5，以便增加清晰度。 其余选项保留预先设置。可以通过单击*确定*按钮关闭控件的属性对话框。

图形编辑器中的实现

步骤	过程：图形编辑器中的实现
1	所要显示的电机均由*标准对象* → *圆*、*标准对象* → *多边形*和*标准对象* → *静态文本*组成。圆的背景色根据电机的状态通过*动态对话框*来更改。 对这三个对象进行编组。在本实例中，这就使得要创建*对象组 1*、*组 2* 和*组 3*。其中每个对象都在*事件* → *鼠标* → *鼠标动作*处有一个*直接连接*，用于将电机的编号写入变量 *U08i_ex_tlg_00* 中；并且在*事件* → *鼠标* → *按下左键*处有一个 *C 动作*，用于使当前列可见(或使其它列不可见)。
2	为每个电机分配两个 *Windows 对象* → *按钮*。这些按钮通过*直接连接*控制分配给各电机的变量。
3	为了标识当前所选的电机，为每个电机分配一个*标准对象* → *矩形*。 在*属性* → *样式* → *线条样式*处选择虚线，并在*属性* → *样式* → *填充模式*处选择*透明*模式。 在*属性* → *其它* → *显示*处，每个电机都用*动态对话框*来使*矩形*只有当变量 *U08i_ex_tlg_00* 的内容与自己的对象编号一致时才可见。

电机 1 (组 1)的 C 动作

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpsz
{


//show column 1
SetPropWord(lpszPictureName,"Control1","Index",0);
SetPropBOOL(lpszPictureName,"Control1","ItemVisible",TRUE);


//hide column 2
SetPropWord(lpszPictureName,"Control1","Index",1);
SetPropBOOL(lpszPictureName,"Control1","ItemVisible",FALSE);

//hide column 3
SetPropWord(lpszPictureName,"Control1","Index",2);
SetPropBOOL(lpszPictureName,"Control1","ItemVisible",FALSE);

}
```

- 通过 SetPropWord 函数，为控件 1 对象设置下标 0。这对应于第一列。然后通过 SetPropBOOL 将此列设置为可见。
- 使用相同的过程将其它列设置为不可见。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 必须根据需要修改要归档的变量。
- 必须根据需要修改对象的图形显示。

4.1.6 以定义的时间进行归档(ex_3_chapter_01e.pdl)

任务定义

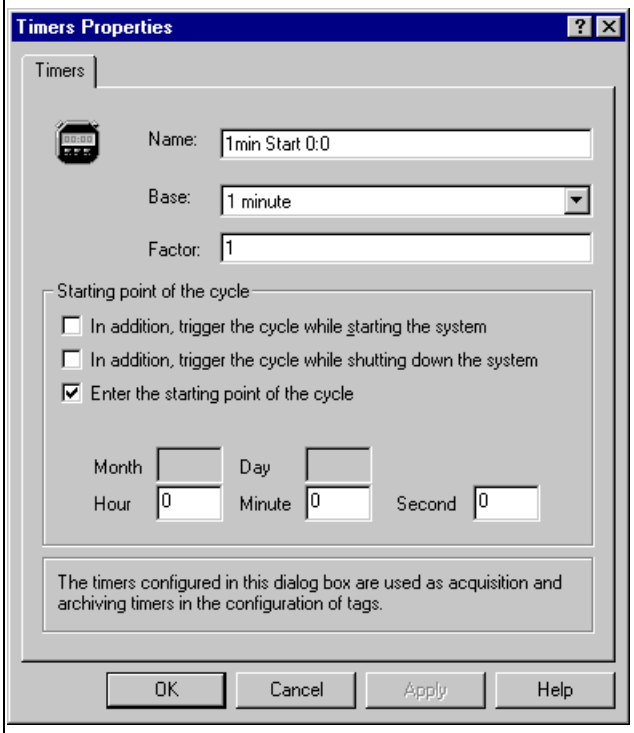
周期连续的过程值归档用于在一秒的周期内获取过程值。每满一分钟，就对数值的总和进行归档。
所归档的数值以表格形式显示，工具栏和状态栏用标准工具来完成。

概念的实现

为了对所要显示的数据进行归档，在*变量记录*编辑器中创建一个周期连续的过程值归档。为了每次满一分钟时进行归档，创建一个新的*定时器*。该定时器按定义的时间启动。归档由该定时器触发。
为了显示数据，创建一个 WinCC 在线表格控件。

创建新的定时器

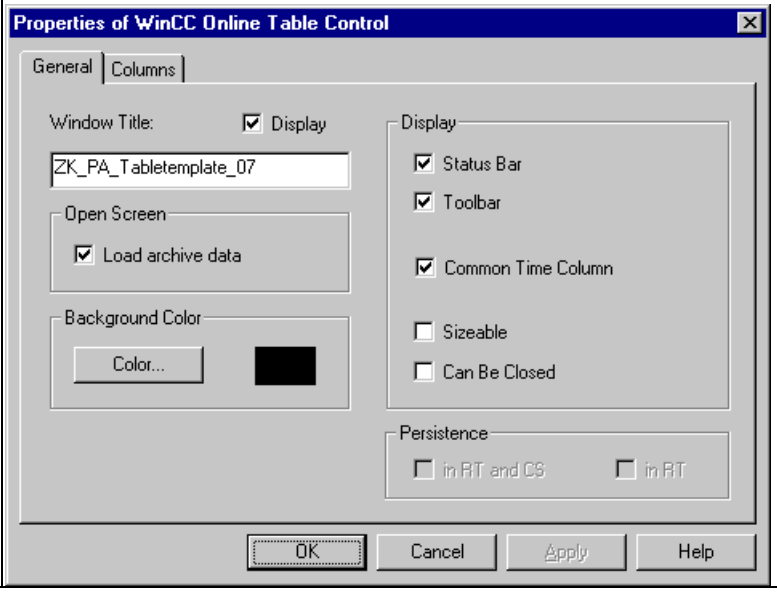
步骤	过程：创建新的定时器
1	从 *WinCC 资源管理器*中打开*变量记录*编辑器。 通过 R 浏览窗口中相应的条目，创建一个新的*定时器*。

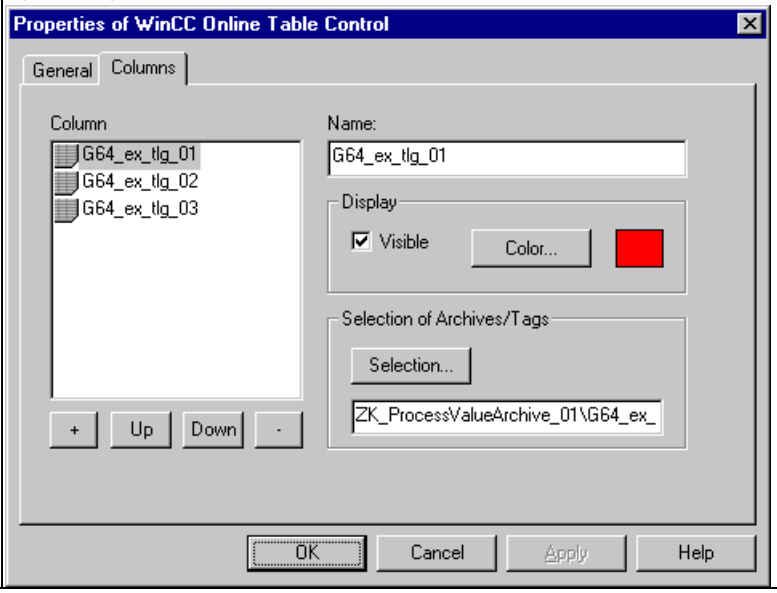
步骤	过程：创建新的定时器
2	显示新*定时器*的属性对话框。 本实例使用 *1min Start 0:0* 作为*定时器*的*名称*。选择带有*系数 1* 的 *1 分钟*，作为基准。*系数*允许用户将定时器组态为 4 或 6 分钟。在*周期的起始点*域内选择*输入周期的起始点*复选框。在周期的每个输入域内，输入 *0*。这样会使周期在运行系统经过第一分钟之后被触发。如果输入具体的时间，那么一旦达到设定时间就会首次启动该周期。 通过单击*确定*退出对话框。 
3	在右边窗口中，除了缺省*定时器*之外还将显示新创建的定时器符号。 Timer name / Time base / Time factor / Last change 500 ms / 500 ms / 1 / 10/22/97 01:14:28 PM 1 Sekunde / 1 second / 1 / 10/22/97 01:14:28 PM 1 Minute / 1 minute / 1 / 10/22/97 01:14:28 PM 1 Stunde / 1 hour / 1 / 10/22/97 01:14:28 PM 1 Tag / 1 day / 1 / 10/22/97 01:14:28 PM 1min Start 0:0 / 1 minute / 1 / 02/03/98 09:08:36 AM 1min Start 10:30 / 1 minute / 1 / 02/03/98 09:10:01 AM 1hour Start 0:0 / 1 hour / 1 / 02/04/98 03:59:48 PM

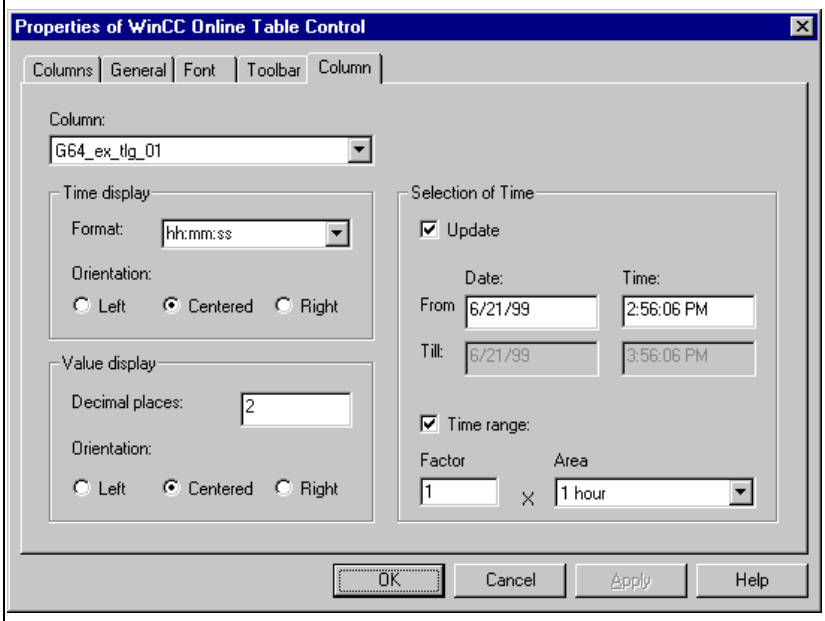
创建过程值归档

步骤	过程：创建过程值归档
1	用归档向导创建一个新的*过程值归档*。在本实例中，归档已命名为 *ZK_ProcessValueArchive_01*。 已选择 *G64_ex_tlg_01*、*G64_ex_tlg_02* 和 *G64_ex_tlg_03* 作为要归档的变量。在本实例项目中，它们将由模拟器提供数值。 在过程值归档的属性对话框中，保留由向导所做的设置。
2	在第一个过程变量的属性对话框中，在*参数*标签的*周期*域内输入 *1 秒*作为*采集周期*。至于*归档*周期，设置 *1 * 1min Start 0:0*。为*处理*选择*求和*选项钮。 第一个变量的其余选项保留缺省设置。 其余的归档变量按照刚才已执行的方法进行相同的设置。

在图形编辑器中的实现

步骤	过程：表格显示的组态
1	创建一个新画面，在实例中它是画面 *ex_3_chapter_01e.pdl*。
2	组态用于显示表格的控件。它是 *WinCC 在线表格控件*。从*对象选项板的控件*选择菜单中选择它，然后将其置于画面中。
3	将控件置于画面中之后，会自动打开 *WinCC 在线表格控件属性*对话框。 在 *常规信息*标签中，可以指定控件的标题以及它如何进行标记。输入 *ZK_ProcessValueArchive_01* 作为窗口标题。通过颜色按钮，将表格窗口的背景色设置为黑色。 本实例在*显示*域内撤消选定*大小可调*复选框，并选择*共享时间列*复选框。 

步骤	过程：表格显示的组态
4	在*列*标签中，具体指定要显示的列。本实例需要三列。 有一列已经创建好。将此列重命名为 *G64_ex_tlg_01*。 通过*选择*按钮，可以把要显示的*归档变量*分配给该列。在本实例中，将先前创建的 *ZK_ProcessValueArchive_01* 归档的*归档变量 G64_ex_tlg_01* 分配给该列。 再添加两列。将变量 *BINi_ex_tlg_m2* 与 *BINi_ex_tlg_m3* 分配给它们，并修改其列名称和颜色。 

步骤	过程：表格显示的组态
5	列的特殊属性设置。为此，可以使用一个扩充的属性对话框。此对话框通过 D 控件来打开。另一方面，先前描述的属性对话框通过在按下 CTRL 键的同时 D 控件来打开。 扩充的属性对话框除了包含已经提及的*常规信息*与*列*标签外，还包含三个附加的标签。在*列*标签中，进行下列设置。 将*时间显示*的*格式*设置为 *hh:mm:ss*。将*时间显示*与*数值显示*的*方向*设置为*居中*。将时间范围设置为 *1 x 1 小时*。 其余选项保留预先的设置。可以通过单击*确定*按钮关闭控件的属性对话框。 

注意：
在*图形编辑器*中为 *ex_3_chapter_01e.PDL* 画面所执行的组态的简短描述可参见棒图显示 *(ex_3_chapter_01e)*一章。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 根据所描述的组态，可以用某个确定的时间启动归档。此外，数值可以每满一分钟(或一小时等)后存储在归档中。
- 必须根据所用的起始时间和时间基准修改本实例中所创建的归档周期，以满足需要。

4.1.7 归档的导出(ex_3_chapter_01f.pdl)

任务定义

一旦归档的数据记录达到最大数目，具有周期连续过程值的归档就作为 CSV 文件导出。归档在系统启动时将被锁住，且只能在按下按钮后才能被激活。
所归档的数值将以表格形式显示，并且需要一个用户定义的工具栏和状态栏。
通过消息对话框来告诉用户有关导出的时间。

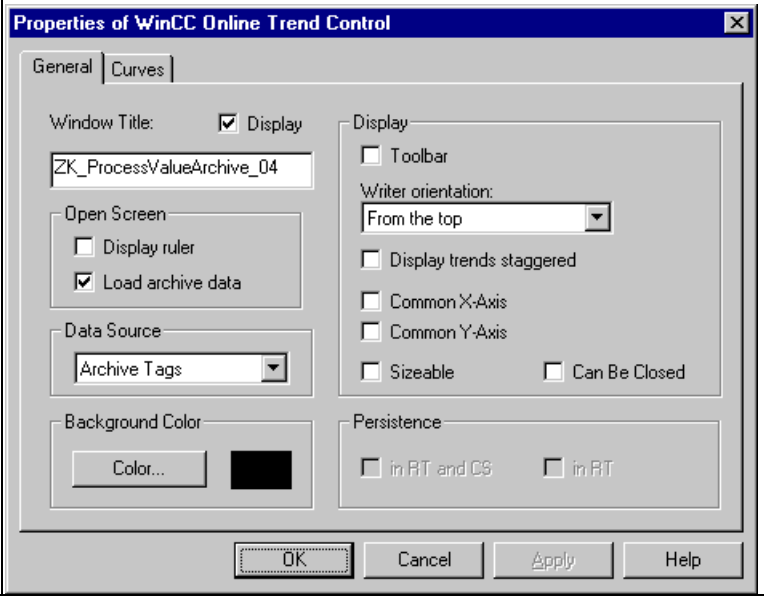
概念的实现

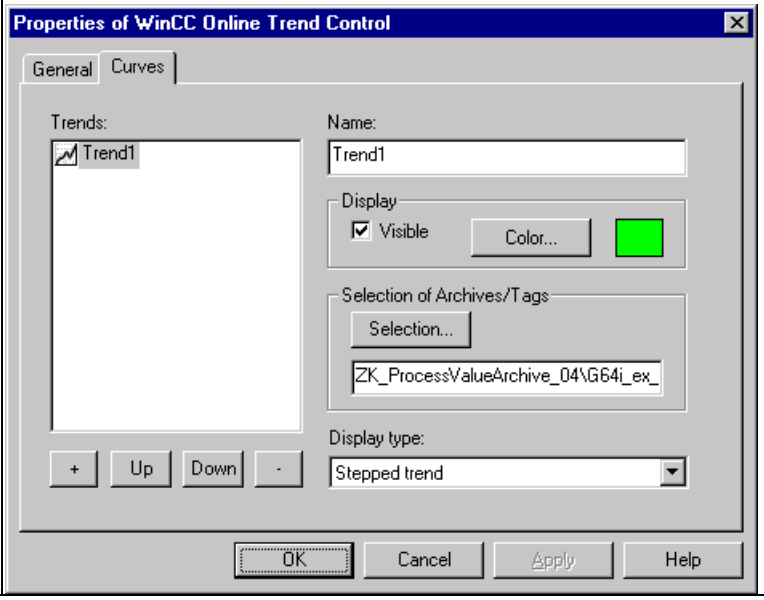
为了对所要显示的数据进行归档，在*变量记录*编辑器中创建一个周期连续的过程值归档。为了导出归档以及锁住/释放归档，创建了项目函数。
为了对数据进行显示，使用了 WinCC 在线趋势控件。工具栏由多个 *Windows 对象* → *按钮与智能对象* → *状态显示*所组成。

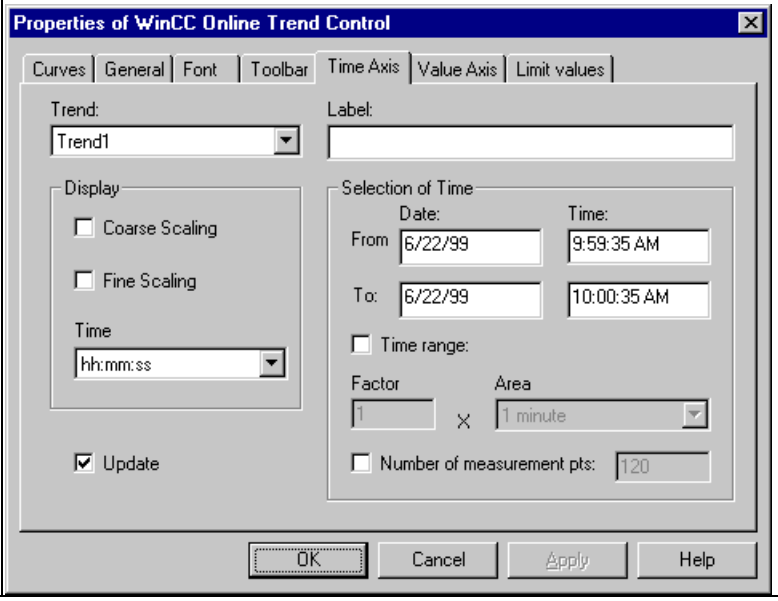
创建过程值归档

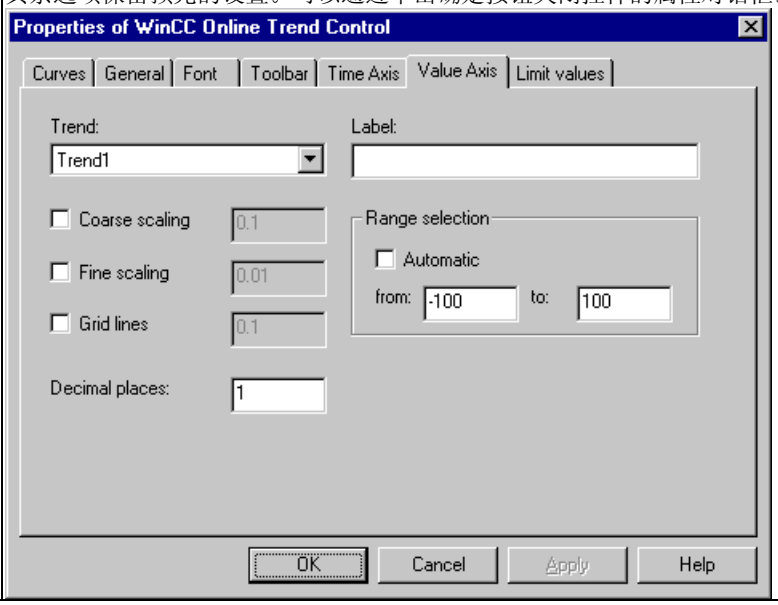
步骤	过程：创建过程值归档
1	用归档向导创建一个新的*过程值归档*。在本实例中，归档已命名为 *ZK_ProcessValueArchive_04*。 对于就要归档的变量，可选择 *G64i_ex_tlg_04* 变量。在实例项目中，由模拟器为该变量提供了三个趋势图形的总和。
2	在过程值归档的属性对话框中，将归档大小设置为 *200* 个数据记录。至于用于*导出短期归档的动作*，则设置了项目函数 *ActionForExportingArchive*。 其余选项保留缺省设置。 Action for exporting the short-term archive ActionForExportingArchive Select...
3	在过程变量的属性对话框中，保留缺省设置。

趋势显示的组态

步骤	过程：趋势显示的组态
1	创建一个新的画面，在实例中它是画面 *ex_3_chapter_01f.pdl*。
2	组态用于显示趋势的控件。它是 *WinCC 在线趋势控件*。从*对象选项板的控件*选择菜单中选择它，然后将它放置在画面中。
3	将控件置于画面中之后，将会自动打开其组态对话框。 在*常规信息*栏中，可以指定控件的标题以及它如何进行标记。输入 *ZK_ProcessValueArchive_04* 作为窗口标题。通过*颜色*按钮，将趋势窗口的*背景色*设置为黑色。 在*显示*域中，撤消所有复选框的选定，并将文本定位设置为*从最上*面开始。 

步骤	过程：趋势显示的组态
4	在*趋势*栏内，详细规定将要显示的趋势。对于本实例，只需要一个趋势。 已经创建了一条趋势。将其重新命名为*趋势 1*。 通过*选择*按钮，可以把要显示的*归档变量*分配给该趋势。在本实例中，将先前创建的 *ZK_ProcessValueArchive_04* 归档的*归档变量 G64_ex_tlg_04* 分配给该趋势。将趋势的显示类型设置为*步进趋势*。 

步骤	过程：趋势显示的组态
5	指定趋势的属性设置。为此，可以使用一个扩充的属性对话框。此对话框通过 D 控件来打开。另一方面，先前描述的属性对话框通过在按下 CTRL 键的同时使用 D 控件来打开。 扩充的属性对话框除了包含已经提及的*常规信息*和*趋势栏*外，还包含五个附加的栏。 在*时间坐标轴*趋势中，进行下列设置。撤消对*粗网格*与*细网格*复选框的选定，并将*时间格式*设置为 *hh:mm:ss*。在*时间选择*域中，撤消对*时间范围*复选框的选定。 

步骤	过程：趋势显示的组态
6	在*数值坐标轴*栏中，撤消*趋势*域中所有选定的复选框。在*区域选择*域中，撤消选定*自动*复选框，并输入*从–100 到 100* 的数值。 其余选项保留预先的设置。可以通过单击*确定*按钮关闭控件的属性对话框。 

用于导出归档的项目函数

```
void ActionForExportingArchive (LPTSTR lpszArchivNameReturn, LPTSTR lpszVar
{
BOOL fRet;
int  iTlgCon = 0;
CMN_ERROR Error;
char szProj[MAX_PATH];
char szFile[MAX_PATH];
LPTSTR  lpszArchivName =
      "PDE#HD#ZK_ProcessValueArchive_04#G64i_ex_tlg_04" ;
char    szFileName[MAX_PATH] = "";
LPTSTR  lpszFileName;
TLG_IO_BACKUP_SELECT   ibs;
DWORD dwSize;
time_t         Time;
struct tm*     TimeStruct;
int nPathLen,nFileLen;

DMGetRuntimeProject( szProj, MAX_PATH, &Error);

nPathLen=strlen(szProj);
nFileLen=strlen((strrchr(szProj,\\)+1));

strncat(szFile,szProj,nPathLen-nFileLen);
sprintf(szFileName,"%s%s",szFile,"ArchiveBackUp.CSV");
lpszFileName=&szFileName[0];

time(&Time);
TimeStruct = localtime(&Time);

ibs.sysFrom.wYear   = 1997;
ibs.sysFrom.wMonth  = 1;
ibs.sysFrom.wDay    = 1;
ibs.sysFrom.wHour   = 0;
ibs.sysFrom.wMinute = 0;
ibs.sysFrom.wSecond = 0;

ibs.sysTo.wYear   = (WORD)(TimeStruct->tm_year+1900);
ibs.sysTo.wMonth  = (WORD)(TimeStruct->tm_mon+1);
ibs.sysTo.wDay    = (WORD)(TimeStruct->tm_mday);
ibs.sysTo.wHour   = (WORD)(TimeStruct->tm_hour);
ibs.sysTo.wMinute = (WORD)(TimeStruct->tm_min);
ibs.sysTo.wSecond = (WORD)(TimeStruct->tm_sec);

fRet = TLGConnect( NULL, &Error );
if (fRet==FALSE)       printf("Error in TLGConect(...)\r\n");

fRet=TLGGetBackupSize (lpszArchivName, &dwSize, &ibs, TLG_BACKUP_EVACUATE,
if (fRet==FALSE) printf("Error in TLGGetBackupSize(...)[%s]\r\n",Error.szEr
else SetTagWord("U16i_ex_tlg_00",(WORD)dwSize);

fRet=TLGBackup (lpszArchivName, lpszFileName, &ibs, TLG_BACKUP_EXPORT, TLG_
if (fRet==FALSE) printf("Error in TLGBackup(...) [%s]\r\n",Error.szErrorTex
else SetTagBit("BINi_ex_tlg_09",TRUE);

TLGDisconnect( NULL );
}
```

- 项目路径的确定。
- 生成归档所要导入的文件名。此文件名同时包含了路径。
- 系统时间的确定。
- 对启动与结束时间进行指定，所要导出的数据的归档时间将放置在它们中间。
- 使用 *TLGConnect* 函数建立与*变量记录*的连接。
- 确定所要导出的数据大小可使用 *TLGBackupSize* 函数。该数值存储在内部变量中。
- 使用 *TLGBackup* 函数导出归档，并对二进制变量 *BINi_ex_tlg_09* 进行设置，这将使导出成为可见。

图形编辑器中的实现

步骤	过程：图形编辑器中的实现
1	在变量管理器中创建三个*二进制变量*类型的变量。在本实例中，使用了变量 *BINi_ex_tlg_06*、*BINi_ex_tlg_08* 和 *BINi_ex_tlg_09* 变量。此外，还需要*无符号的 16 位数*类型的变量 *U16i_ex_tlg_00*。
2	在当数值超出时的归档(ex_3_chapter_01b.pdl)实例中，已对用户定义的工具栏的执行过程进行了详细的描述。本章将只描述新添加的控制元件。
3	为了对归档进行控制，组态了一个 *Windows 对象* → *按钮*。在本实例中，它是对象*按钮 16*。 在*事件* → *鼠标* → *按下左键*处，创建一个 *C 动作*，用于对 *BINi_ex_tlg_08* 变量的状态求反，并调用项目函数 *LockUnlockArchive*。二进制变量用于存储归档的当前状态。
4	组态一个新的画面，在归档导出时，将显示该画面。在本实例中，这就是画面 *ex_5_window_03.PDL*。 该画面包含了一个*标准对象* → *静态文本*，它可通过 *C 动作*来显示文本。该文本由一个固定部分和 *U16i_ex_tlg_00* 变量的数字值所组成。该变量包括所要导出的数据的大小。此外，画面还包含一个 *Windows 对象* → *按钮*与一个*智能对象* → *图形对象*，通过*直接连接在事件* → *鼠标* → *鼠标动作*处，这两个对象可将常数 *0* 切换为变量 *BINi_ex_tlg_09*。
5	在初始画面中，组态一个*智能对象* → *画面窗口*；在本实例中，它就是对象*画面窗口 1*。在*属性* → *其它* → *画面名称*处，设置 *ex_5_window_03.PDL* 画面。 在*属性* → *其它* → *显示*处，创建一个与 *BINi_ex_tlg_09* 变量的变量连接。
6	为了显示状态栏，组态两个 *Windows 对象* → *按钮*；在本实例中，它们是对象*按钮 14* 和*按钮 17*。 为*按钮 17*，在*属性* → *字体* → *文本*处创建一个*动态对话*，根据变量 *BINi_ex_tlg_06*，该对话框可将文本*更新启动*或*更新停止*返回给属性。 为*按钮 14*，在*属性* → *字体* → *文本*处创建一个*动态对话*，根据变量 *BINi_ex_tlg_08*，该对话框可将文本*归档激活*或*归档锁住*返回给属性。

用于锁住和激活归档的项目函数

```
void LockUnlockArchive (BOOL bLock)
{
  BOOL fRet;
  CMN_ERROR Error;
  LPTSTR       lpszArchivName = "ZK_ProcessValueArchive_04";

fRet = TLGConnect( NULL, &Error );

if (fRet==FALSE)
        printf("Error in TLGConnect(...)\r\n");
else
{
        fRet=TLGLockArchiv (NULL,lpszArchivName,bLock,&Error );
        if (fRet==FALSE) printf("Error in TLGLockArchive(...) [%s]\r\n",
        Error.szErrorText);
        TLGDisconnect( NULL );
}

}
```

- 建立与*变量记录*的连接。
- 调用函数 *TLGLockArchive*。传送参数 *bLock* 将决定是锁住还是激活归档。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 必须根据需要修改要归档的变量。
- 导出文件的最大归档大小、路径以及文件名均需进行修改。

4.2 报警记录

在运行系统中，可通过用选择如上所示的按钮来访问有关该主题的实例。这些实例在画面 *ex_3_chapter_02.pdl* 至 *ex_3_chapter_02d.pdl* 中组态。

常规信息

*报警记录*编辑器负责消息的获取和归档。编辑器所包含的函数可用于传送过程中的消息、处理消息、显示消息、确认消息以及归档消息。
这样，可使报警记录支持用户查找错误原因。

4.2.1 位消息过程(ex_3_chapter_02.pdl)

任务定义

由*报警记录*对四台电机进行监控。电机故障可通过在分配给每台电机的变量中设置不同的位来显示。电机的消息状态存储在内部变量中。电机的显示根据消息状态而改变。
消息将显示在消息窗口中。

概念的实现

在*报警记录*中，必须创建多个与所监控的四台电机相关的单个消息。
可使用一个 WinCC 报警控件在图形编辑器中执行消息窗口。可使用多个*标准对象*来显示各台电机。电机在不同消息状态下的显示变化可通过使用 *C 动作*来实现。

所需变量的创建

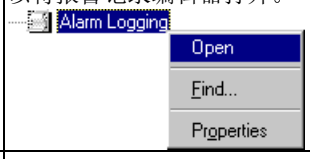
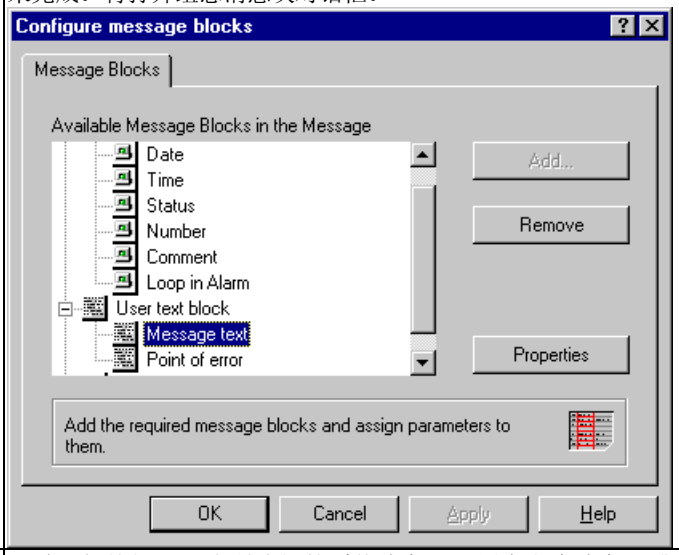
步骤	过程：所需变量的创建
1	在变量管理器中共创建十二个*无符号 16 位数*类型的变量。 其中四个变量用作事件变量。在本实例中，它们是 *U16i_ex_alg_00*、*U16i_ex_alg_03*、*U16i_ex_alg_06* 与 *U16i_ex_alg_09* 变量。另外四个变量用作状态变量；在本实例中，它们是 *U16i_ex_alg_02*、*U16i_ex_alg_05*、*U16i_ex_alg_08* 与 *U16i_ex_alg_11* 变量。其余变量，即本实例中的 *U16i_ex_alg_12*、*U16i_ex_alg_13*、*U16i_ex_alg_14* 与 *U16i_ex_alg_15* 变量，将用作确认变量。

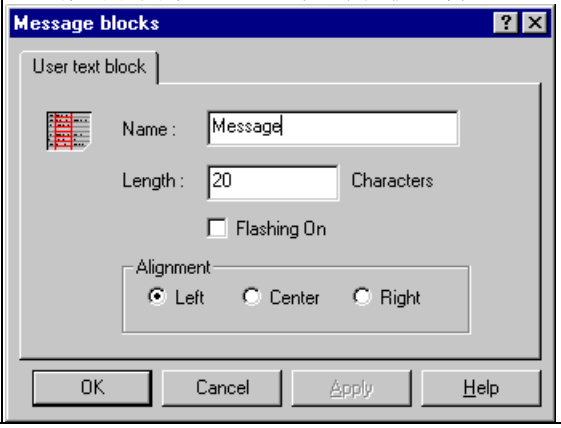
消息块

一条消息由多个不同的消息块组成。可将其归类为三个方面：

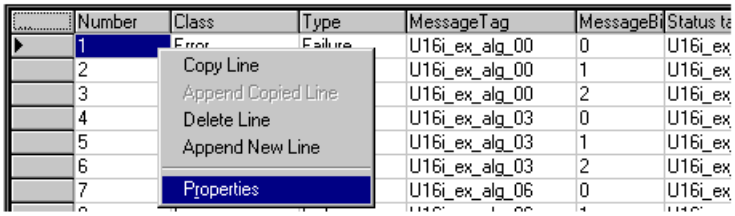
- **系统块：**这些块包含了由*报警记录*所分配的系统数据。包括日期、时间、报表标识等。
- **过程值块：**这些块包含了由过程返回的数值，例如临界填充量、温度等。
- **用户文本块：**用于常规信息与解释的文本，例如出错解释、消息原因、出错查找等。

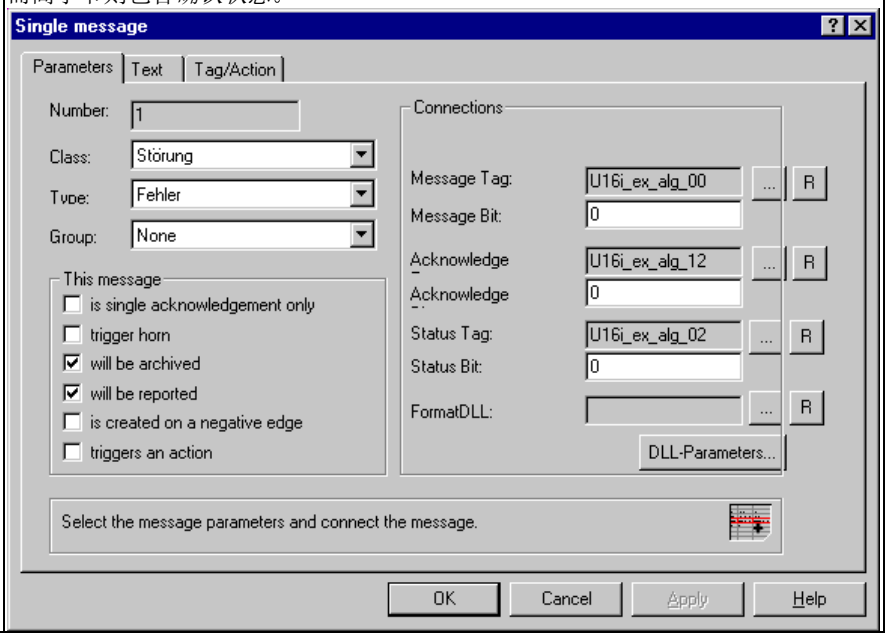
消息块的组态

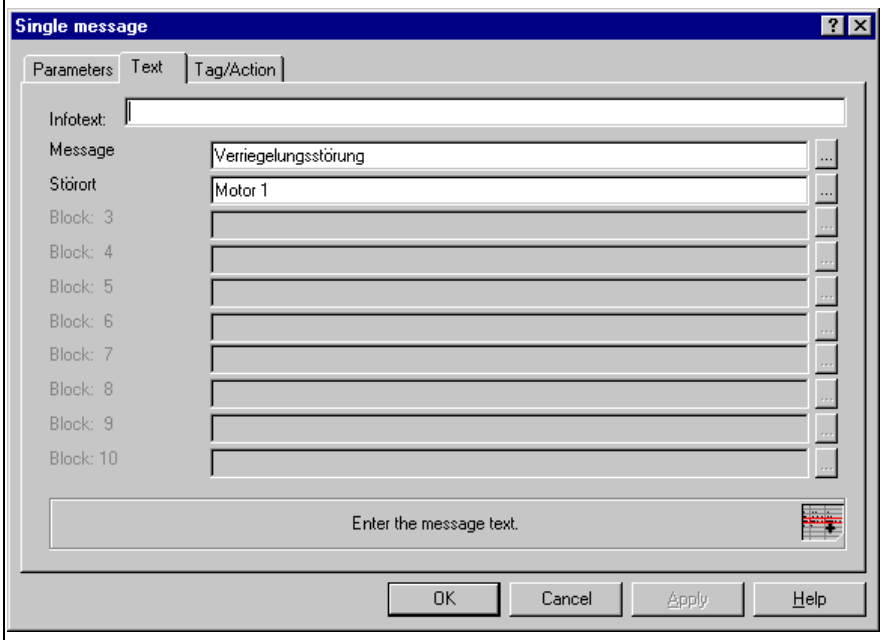
步骤	过程：消息块的组态
1	在 *WinCC 资源管理器*中，通过 R *报警记录*编辑器然后从弹出菜单中选择打开，以将报警记录编辑器打开。 
2	选择所期望的消息块。这可通过 R 消息块条目，然后从弹出菜单中选择消息块来完成。将打开组态消息块对话框。 
3	通过添加按钮，可为所选择的系统块条目、用户文本块条目或过程值块条目打开用于添加块的对话框。 如果用在*组态消息块*对话框中选择一个块，则按钮*删除*与*属性*将可操作。 第一个按钮允许用户删除所选择的消息块，第二个按钮允许用户组态各个消息块的属性。

步骤	过程：消息块的组态
4	在本实例中，保留*名称*，并将*长度*设置为 *20* 个字符。 单击*确定*按钮以完成在*消息块*对话框中所做的设置。 *组态消息块*对话框也可通过单击*确定*按钮来关闭。 

单个消息的创建

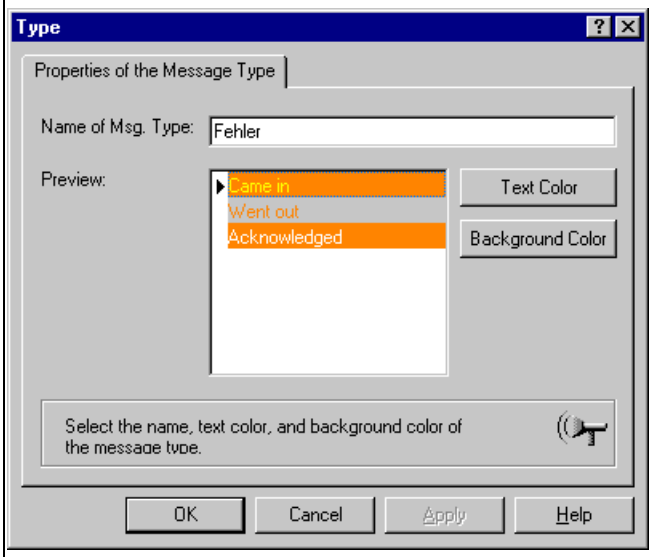
步骤	过程：单个消息的创建
1	在*报警记录*编辑器中，表格窗口位于较底部区域中。在该区域中，组态单个消息，并显示一个已经组态的消息。通过使用 R，可添加新的行。 对于该实例，总共创建了 12 个不同的单个消息。 每个消息对应于表格窗口中的一行，并由多列组成。在各列中可以直接进行设置。然而，在本实例中，是通过*单个消息*对话框来进行设置的。可通过 R 相关消息行来打开该对话框。 

步骤	过程：单个消息的创建
2	按照先前所描述的步骤打开第一行的*单个消息*对话框。 在*参数*标签中，选择消息级别*错误*与消息类型*故障*。 在*消息*域中，选择复选框*将进行归档*与*将进行报告*。 在*连接*域中，选择变量 *U16i_ex_alg_00* 作为*事件变量*。输入 *0* 作为*事件位*。也就是说，如果变量设置的第一位为状态 1，将会产生消息。 对于*确认变量*，选择变量 *U16i_ex_alg_12*，对于*确认位*，则输入 *0*。换句话说，如果在运行系统中确认消息，则变量设置的第一位将设为 1。 对于*状态变量*，选择变量 *U16i_ex_alg_12*，对于*状态位*，则输入 *0*。该设置表示变量设置的第一位代表了消息的*到达/消失状态*。如果消息是未决的，该位将被设置为 1；如果消息不再是未决的，则重新设定该位。该变量的第九位包含消息的*确认状态*。如果没有对其确认，则该位的状态为 1，如果对其进行了确认，则状态为 0。 一个 16 位的状态变量可以代表 8 个单个消息的状态。低字节包含*到达/消失状态*，而高字节则包含*确认状态*。 

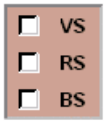
步骤	过程：单个消息的创建
3	在*文本*标签中，输入*消息文本锁定错误*与*出错点电机 1*。没有使用任何*信息文本*。 在本实例中不可填写*变量/动作*标签。 单击*确定*按钮应用所作的设置。 
4	刚才所创建的消息监控四台电机的第一台。必须为第一台电机创建超过二行的消息行。 按照步骤 2 至 3 所描述的那样进行设置，但要修改*事件位*、*确认位*与*状态位*。同时还使用了*消息文本反馈错误*与*双金属错误*。
5	对于另三个电机，还可分别创建三个消息行。 此处必须对*事件变量*、*确认变量*和*状态变量*以及用于*出错点*的文本进行修改。

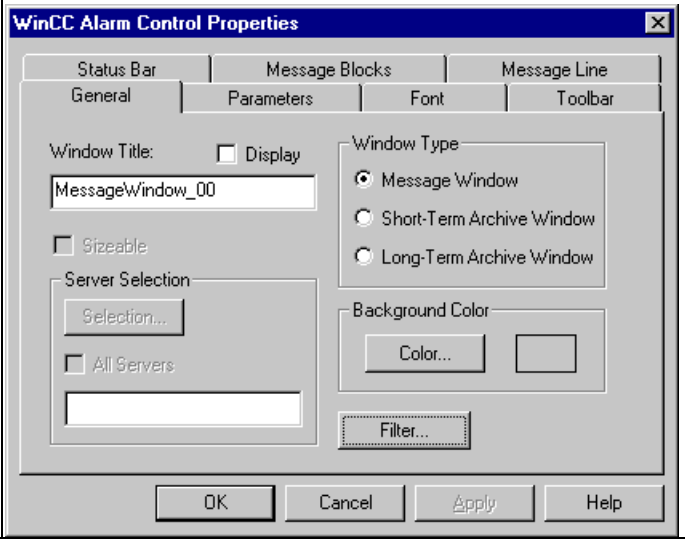
...	Number	Class	Type	MessageTag	MessageBi	Status tag	Status bit	Message text
▶	1	Error	Failure	U16i_ex_alg_00	0	U16i_ex_alg_02	0	Lock Error
	2	Error	Failure	U16i_ex_alg_00	1	U16i_ex_alg_02	1	Feedback Error
	3	Error	Failure	U16i_ex_alg_00	2	U16i_ex_alg_02	2	Bimetal Error
	4	Error	Failure	U16i_ex_alg_03	0	U16i_ex_alg_05	0	Lock Error
	5	Error	Failure	U16i_ex_alg_03	1	U16i_ex_alg_05	1	Feedback Error
	6	Error	Failure	U16i_ex_alg_03	2	U16i_ex_alg_05	2	Bimetal Error
	7	Error	Failure	U16i_ex_alg_06	0	U16i_ex_alg_08	0	Lock Error
	8	Error	Failure	U16i_ex_alg_06	1	U16i_ex_alg_08	1	Feedback Error
	9	Error	Failure	U16i_ex_alg_06	2	U16i_ex_alg_08	2	Bimetal Error
	10	Error	Failure	U16i_ex_alg_09	0	U16i_ex_alg_11	0	Lock Error
	11	Error	Failure	U16i_ex_alg_09	1	U16i_ex_alg_11	1	Feedback Error
	12	Error	Failure	U16i_ex_alg_09	2	U16i_ex_alg_11	2	Bimetal Error

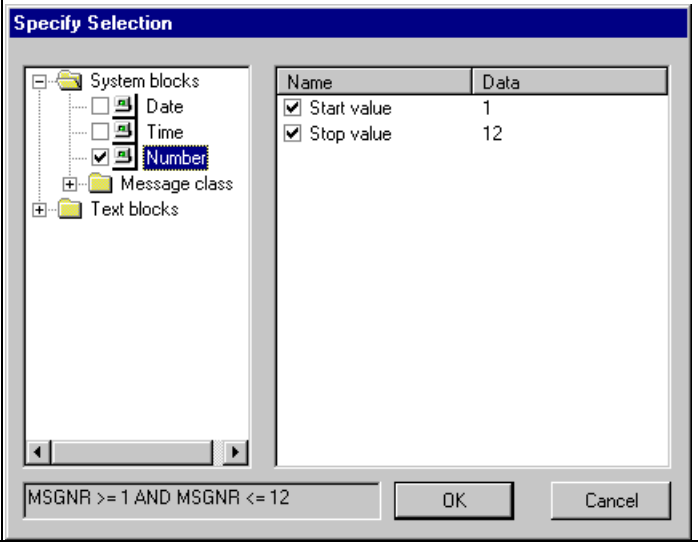
对消息颜色方案的组态

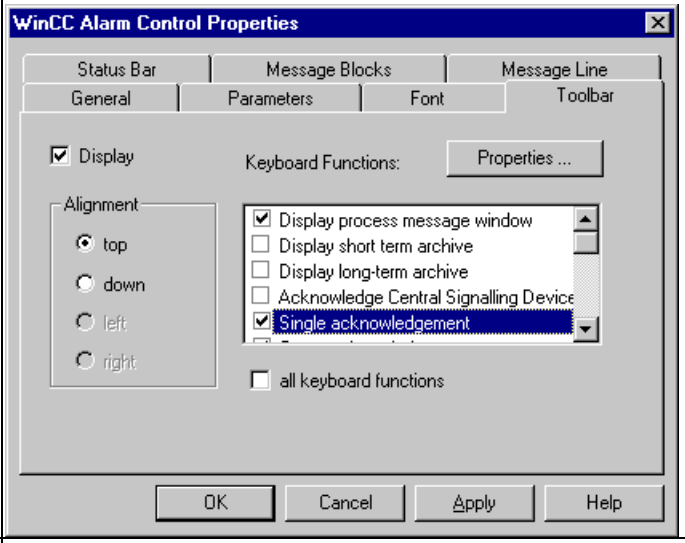
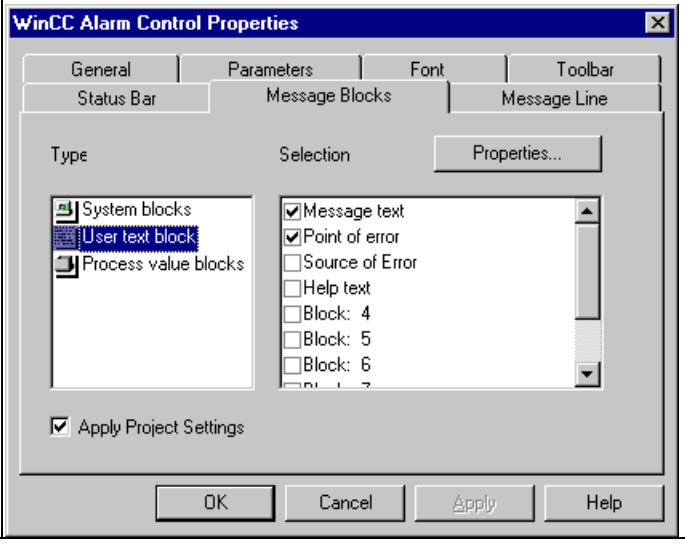
步骤	过程：对消息颜色方案的组态
1	所组态的单个消息的消息等级为*错误*，消息类型为*故障*。 通过*消息等级*条目，可将所有可用的消息等级显示在右边窗口中。通过使用D 消息等级*错误*的图标，可显示所有与该等级相关的消息类型。通过使用D 消息类型*故障*的图标或通过R，然后从弹出菜单中选择属性，可打开*类型*对话框。
2	在类型对话框中，可以为每个消息状态创建颜色方案。 在本实例中，使用了下列颜色方案： 到达：文本 = 黄色，背景色 = 橙色 消失：文本颜色 = 橙色，背景色 = 淡灰色 确认：文本 = 白色，背景色 = 橙色 
3	在*报警记录*中进行的组态可通过*文件* → *保存*菜单来保存。

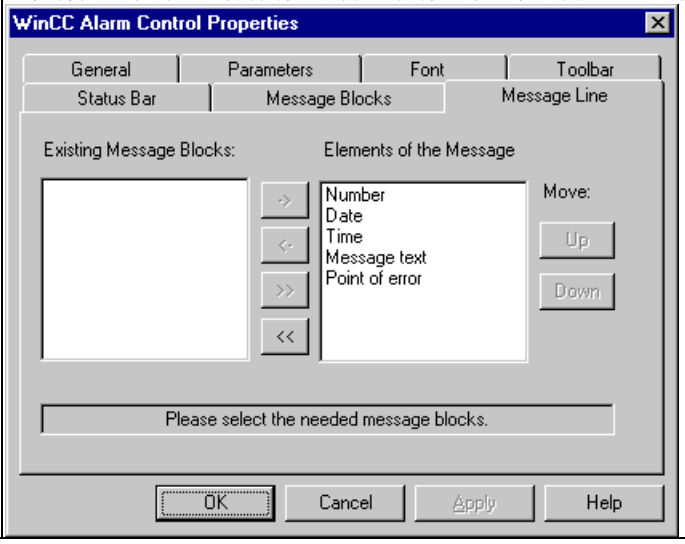
图形编辑器中的实现

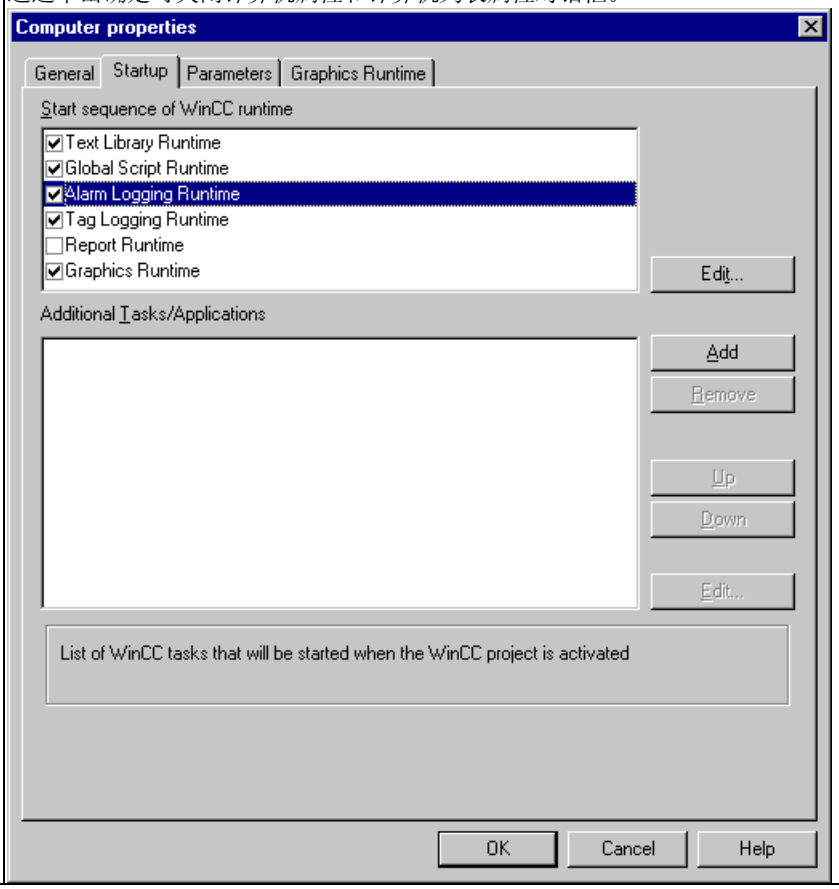
步骤	过程：图形编辑器中的实现
1	创建一个新的画面；在本项目中是 *ex_3_chapter_02* 画面。
2	各个电机可通过*标准对象* → *圆*、*标准对象* → *静态文本*和*标准对象* → *多边形*来显示。 当发生错误或该消息被确认时，电机就将修改其颜色方案。该颜色方案对应于消息状态的到达、消失和确认。 为此，在*属性* → *颜色* → *字体颜色*处，为*静态文本*创建一个 *C 动作*，该动作将根据电机的状态变量的当前状态来修改字体颜色。 同样，在*属性* → *颜色* → *背景色*处，为*圆*创建一个 *C 动作*来完成同样的任务。
3	通过 *Windows 对象* → *复选框*对发生在电机处的错误进行模拟。 在*属性* → *几何图形* → *方框数*中，输入 *3*。 在*属性* → *输出/输入* → *所选方框*处，创建一个与电机的相关事件变量的*变量连接*。 

步骤	过程：图形编辑器中的实现
4	为了显示在*报警记录*中组态的消息，使用了一个 *WinCC 报警控件*。从对象选项板的*控件*选择菜单中选择它，然后将其置于画面中。
5	将控件置于画面中之后，将会自动打开其*组态对话框*。单击*确定*关闭对话框。 打开控件的*属性*对话框。可通过 D 控件来显示该对话框。在*常规信息*标签中，使用*选择*按钮来选择在报警记录中创建的将由控件显示的单个消息。 

步骤	过程：图形编辑器中的实现
6	通过系统块*编号 2*，将在右边窗口中显示两个复选框。通过D 缺省值 *0* 将*起始值*修改为 *1*，将*停止值*修改为 *12*。也就是说，控件只能显示编号 1 至 12 的单个消息。 

步骤	过程：图形编辑器中的实现
7	在*工具栏*标签中，选择将在运行系统中显示的按钮。在本实例中，所需要的按钮如下： *单个确认*、*组确认*、*自动滚动开/关*、*列表开始*、*列表结束*、*下一个消息*以及*前一个消息*。 
8	在*消息块*标签中，选择以后将显示在消息行中的列。在本实例中，使用在*类型*域中选择系统块。在右边窗口中，选择*日期*、*时间*和*编号*。对于用户文本块条目，选择*消息文本*与*出错点*。 

步骤	过程：图形编辑器中的实现
9	在*消息行*标签中，将先前所选的*消息块*分配给消息行。*可用的消息块*域将列出可用的列。通过按下->按钮，可将各个消息块分别添加到消息行。通过>>按钮，可一次将窗口中列出的所有消息块分配给消息行。单击*确定*退出属性对话框。 

步骤	过程：图形编辑器中的实现
10	*激活报警记录运行系统。* 为此，在 *WinCC 资源管理器*中 R *计算机*条目，然后从弹出式菜单中选择*属性*来打开*计算机列表属性*对话框。单击*属性*按钮来打开本地计算机的属性对话框。在*启动*标签中，选择要激活的运行系统应用程序。必须选择*报警记录运行系统*复选框。 通过单击*确定*可关闭*计算机属性*和*计算机列表属性*对话框。 

圆(圆 1)处的 C 动作

```
#include "apdefap.h"
 long _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty
{
DWORD state;

state=GetTagDWord ("U16i_ex_alg_02");

if ((state&1)||(state&2)||(state&4))
        return 0x80FF;
else
        return 0xFFFFFF;

}
```

- 该 *C 动作*使分配给第一台电机的*圆*的背景色属性动态化。
- 读取分配给第一台电机的状态变量 *U16i_ex_alg_02*。如果该变量的低字节包含有消息状态到达/消失，即如果该变量的第一、第二或第三位被置为 1，则消息就是未决的，圆的背景色被设置为橙色(十六进制 80ff)。如果消息消失，则背景色将被设置为白色(十六进制 ffffff)。
- 一旦改变状态变量 *U16i_ex_alg_02* 即触发该 *C 动作*。

静态文本(静态文本 1)处的 C 动作

```
#include "apdefap.h"
 long _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty
{
DWORD state;

State=GetTagDWord ("U16i_ex_alg_02");

if ( ((state&1)&&(state&256)) || ((state&2)&&(state&512)) ||
        ((state&4)&&(state&1024)) )
        return 0xFFFF;
else if ((state&1)||(state&2)||(state&4))
        return 0xFFFFFF;
else if ((state&256)||(state&512)||(state&1024))
        return 0x80FF;
else    return 0x800000;

}
```

- 该 *C 动作*将使分配给第一台电机的*静态文本*的字体颜色属性动态化。
- 读取分配给第一台电机的状态变量 *U16i_ex_alg_02*。该变量的低字节包含消息状态到达/消失，高字节则包含已确认的消息状态。如果是未经确认的未决的消息，则字体颜色将被设置为黄色(十六进制 ffff)；如果是已确认的消息，则字体颜色设置为白色(十六进制 ffffff)；如果是未经确认但已消失的消息，则字体颜色设置为橙色(十六进制 80ff)。在通常情况下，字体颜色为深蓝色(十六进制 800000)。
- 一旦改变状态变量 *U16i_ex_alg_02* 即触发该 *C 动作*。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 必须对所需消息块进行修改以满足用户需求。
- 必须将 WinCC 报警控制修改为所期望的显示类型。
- 必须修改事件、状态和确认变量以及它们的位以满足用户需求。

4.2.2 限制值的监控(ex_3_chapter_02a.pdl)

任务定义

通过*报警记录*对三个容器中的压力和温度值进行监控。如果所监控的模拟值接近临界值，就会产生警告。如果它们达到了临界范围，也会产生报警。报警的发生也将在*图形编辑器*中从视觉和听觉方面进行报告。
使用了大型的用户定义的消息窗口布局。

概念的实现

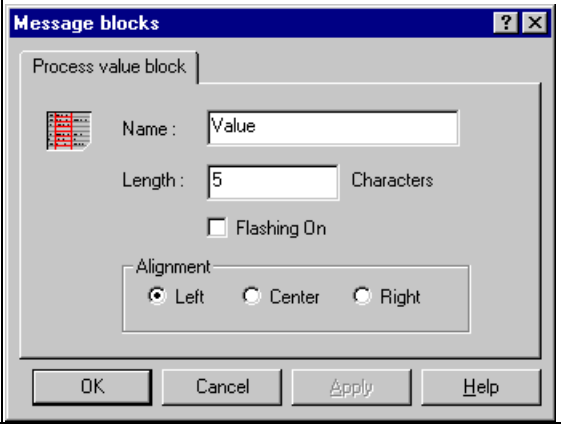
在*报警记录*中，必须创建多个单个消息，用于引用所监控的三个包容器。
利用 WinCC 报警控件在*图形编辑器*中创建消息窗口。工具栏由多个 *Windows 对象* → *按钮与智能对象* → *状态显示*所组成。

所需变量的创建

步骤	过程：所需变量的创建
1	在变量管理器中总共创建 6 个*无符号 16 位数*类型的变量。其中三个变量包含单个包容器的温度值。在本实例中，它们是变量 *U16i_ex_alg_t1*、*U16i_ex_alg_t2* 与 *U16i_ex_alg_t3*。剩余的三个变量均包含压力值。在本实例中，它们是变量 *U16i_ex_alg_p1*、*U16i_ex_alg_p2* 与 *U16i_ex_alg_p3*。 需要三个*无符号 16 位数*类型的附加变量，用作状态变量。在本实例中，它们就是变量 *U16i_ex_alg_01*、*U16i_ex_alg_04* 与 *U16i_ex_alg_07*。 需要一个用于控制中心指示器的*无符号 16 位数*类型的变量；在本实例中，它就是变量 *U16i_ex_alg_10*。 此外，还需要两个二进制变量类型的变量。在本实例中，它们就是变量 *BINi_ex_alg_00* 与 *BINi_ex_alg_03*。

注意：
在先前实例的*组态消息块*表中进行的组态已全面考虑，不再单独进行描述。

新的消息窗口模板的建立

步骤	过程：新的消息窗口模板的建立
1	打开*报警记录*编辑器。 如果消息到达，该消息窗口应显示所监控变量的当前值。为此，必须创建新的过程值块。 可通过 R *消息块*条目来打开*组态消息块*对话框。在对话框中，选择*过程值块*列表条目，并单击添加按钮打开*添加过程值块...*对话框。在该对话框中，将添加一个新的过程值块。通过单击*确定*关闭对话框。 通过 D *过程值块*列表条目，将显示一个新建块。如果选中了该块，则通过属性按钮可对其进行访问。在本实例中，输入*数值*作为块的*名称*，并将 *5* 个字符作为其*长度*。 单击*确定*，应用*消息块*与*组态消息块*对话框中所作的设置。 

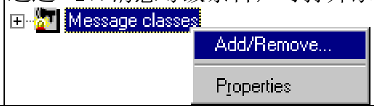
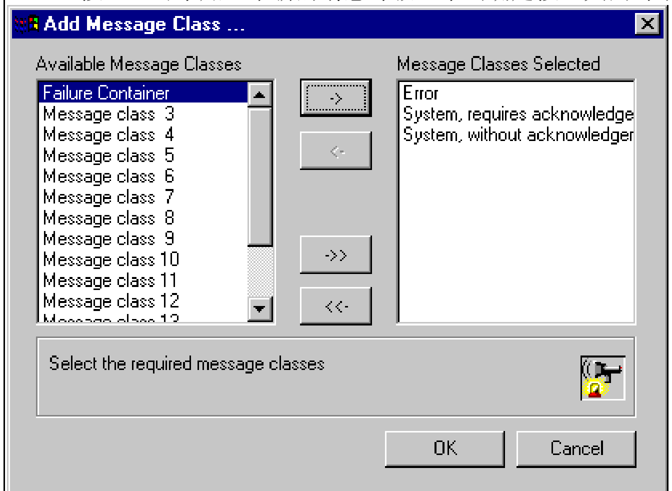
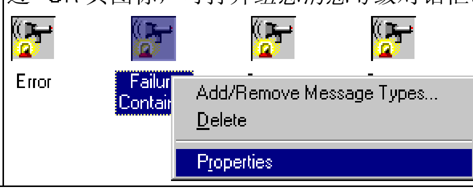
常规信息

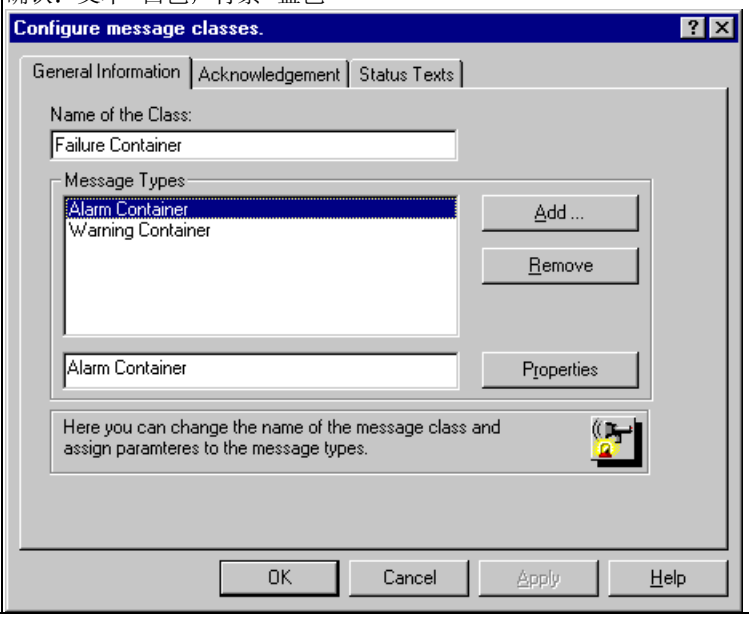
利用消息等级的帮助，

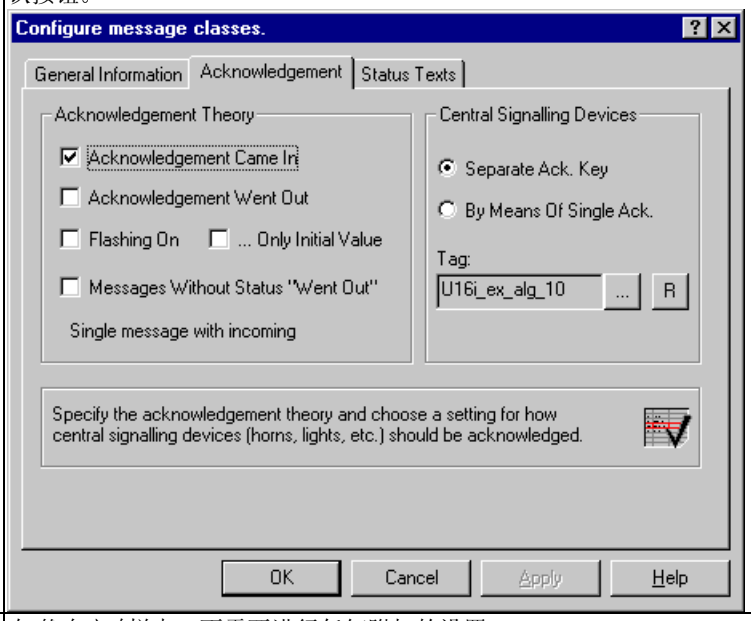
- 对确认类型
- 对应的状态文本
- 声响/视觉信号的输出

进行指定，用于属于消息等级的所有消息类型。

创建新的消息等级

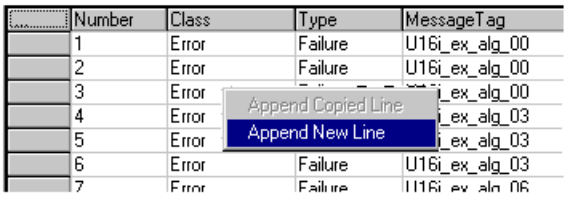
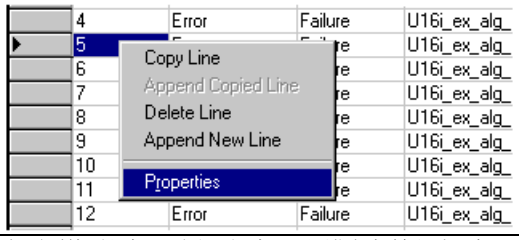
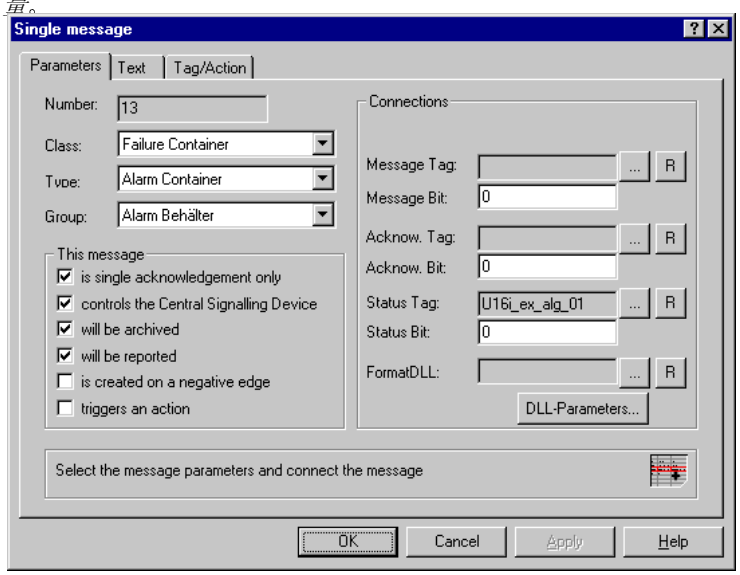
步骤	过程：创建新的消息等级
1	通过R *消息等级*条目，可打开*添加消息等级*...对话框。 
2	通过->按钮，可添加一个新的消息等级。单击*确定*按钮关闭对话框。 
3	通过*消息等级*条目，可显示所有已创建的消息等级，即使是新添加的等级。通过R 其图标，可打开*组态消息等级*对话框。 

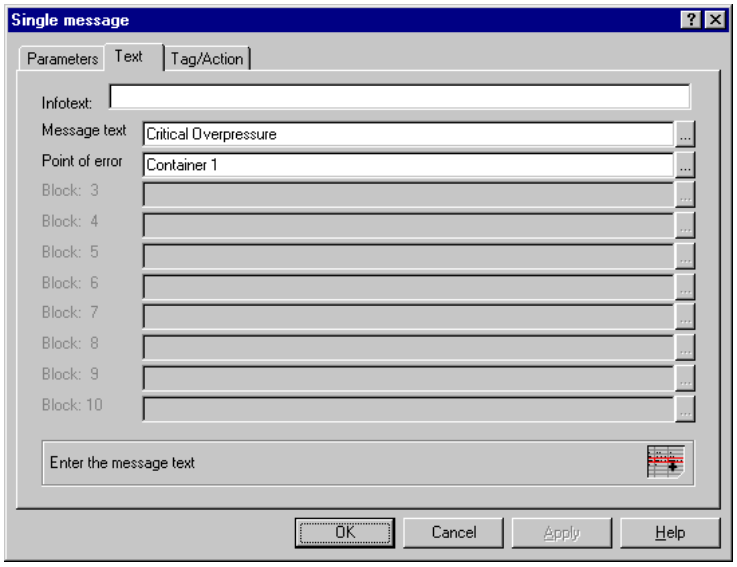
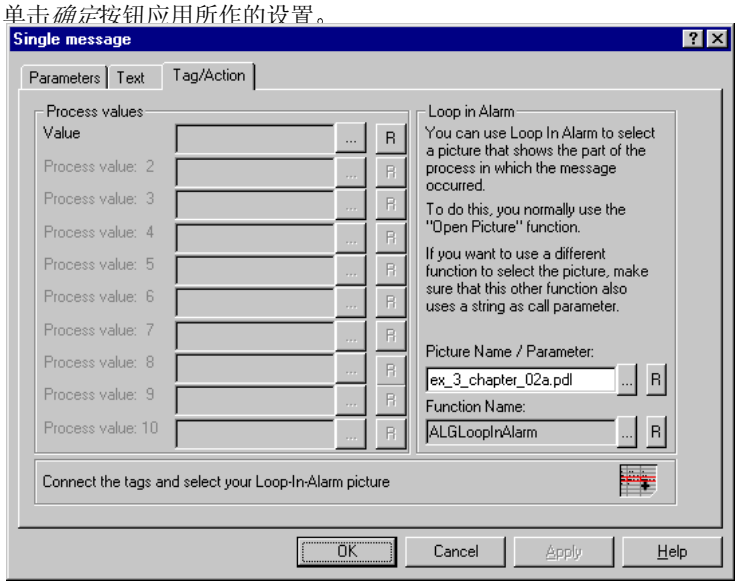
步骤	过程：创建新的消息等级
4	在*常规信息*栏中，输入*包容器出错*，以此作为*等级的名称*。 通过添加按钮访问*添加消息类型*...对话框。在该对话框中，可通过->按钮将两种消息类型从左窗口移到右窗口。单击*确定*按钮关闭对话框。 如果在*消息类型*域中选择了新添消息类型中的一个，则通过属性按钮可将其属性对话框打开。 输入*包容器报警*作为第一个消息类型的名称。单个消息状态的颜色图看起来如下： 到达：文本=黑色，背景=红色 消失：文本=黑色，背景=绿色 确认：文本=黑色，背景=橙色 输入*包容器警告*作为第二个消息类型的名称。单个消息状态的颜色图看起来如下： 到达：文本=黄色，背景=蓝色 消失：文本=蓝色，背景=RGB (207、163、146) 确认：文本=白色，背景=蓝色 

步骤	过程：创建新的消息等级
5	在*确认*栏中，从*确认原理*域中选择*确认到达*复选框。 在*中心指示的确认*域中，选择*单独的确认键*选项钮。作为*变量*，对*U16i_ex_alg_10* 变量进行设置。该变量控制中心指示器。为了确认该指示器，必须在工具栏中组态一个单独的按钮。如果组态了标准工具栏，则它就是*蜂鸣器确认*按钮。 
6	在*状态文本*栏中，不需要进行任何附加的设置。 单击*确定*按钮，关闭对话框。

4.2.3 限制值的监控(续)

单个消息的创建

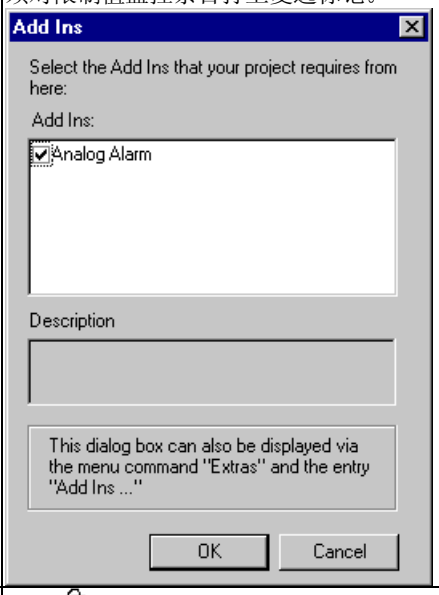
步骤	过程：单个消息的创建
1	通过 R 表格窗口，添加 12 个新的行。  通过选择新添加行中的第一行。通过 R 该行，打开单个消息对话框。 
2	在*参数*标签中，选择*包容器出错*消息等级与*容器报警*消息类型。在*本消息*域中，选择仅*单个确认*、*控制蜂鸣器*、*将被归档*与*将被报告*等复选框。在*连接*域中，将变量 *U16i_ex_alg_01* 选作*状态变量*。作为*状态位*，可输入 *0*。不需要设置任何*事件变量*，因为消息是由限制值监控所产生的。同样，也不需要设置任何*确认变量*。 

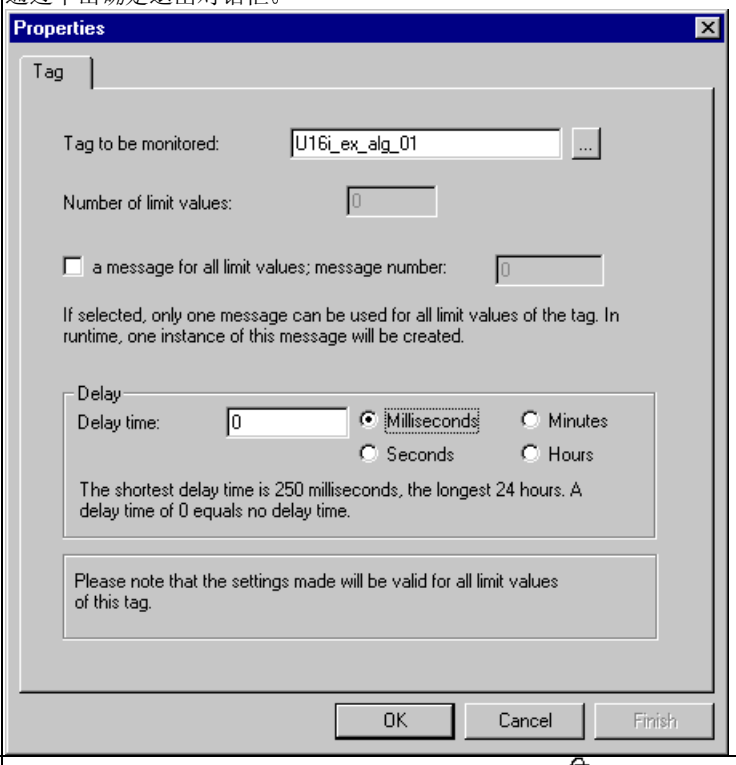
步骤	过程：单个消息的创建
3	在*文本*标签中，输入*消息文本临界超压*与*出错点包容器 1*。输入*容器 1 中的已超出临界值的压力*作为*信息文本*。 
4	在*变量/动作*标签内，可设置一个变量，用于*数值*过程值块。然而，如果消息是由限制值的监控所产生的，则将自动为消息行的第一个过程值块提供引起消息触发的限制值。 单击*确定*按钮应用所作的设置。 

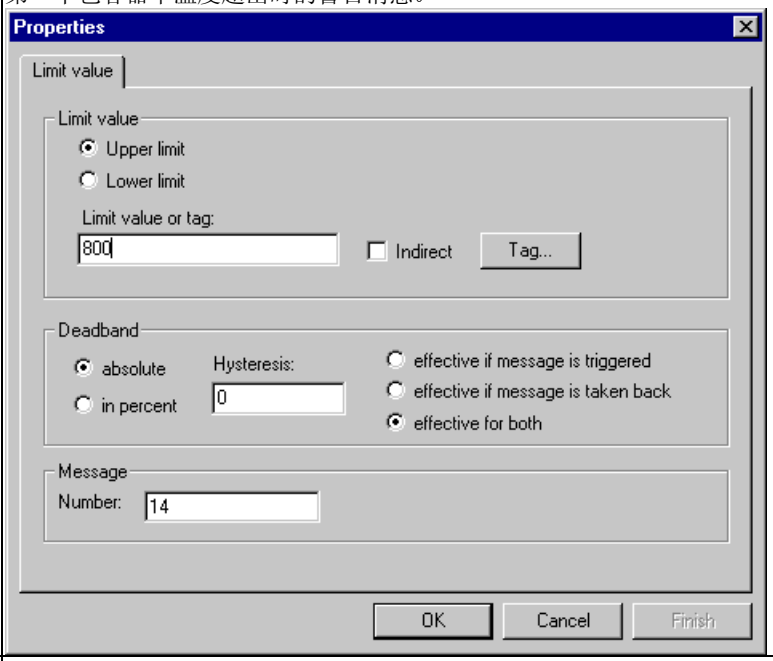
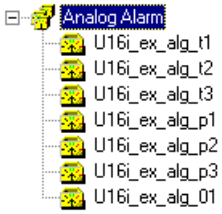
步骤	过程：单个消息的创建
5	刚创建的消息将对三个包容器中的第一个包容器中的压力进行监控。必须为第一个包容器创建三个以上的消息行。 按照步骤 2 的描述进行设置，然而，为*包容器出错*消息类型的附加消息，输入*消息文本临界温度*和已进行了相应修改的*信息文本*。此外，还创建两个*包容器警告*消息类型的消息，它具有消*息文本压力警告*与*温度警告*。对于这些警告，*参数*标签的*单个消息*对话框中的*本消息*域内所有的复选框均被撤消选定。对于与包容器 1 相关的所有消息，使用了同样的状态变量，但带有经修改的状态位。
6	对于其它两个包容器，也可为其分别创建四个消息。 这里，必须根据各个包容器对*状态*变量与用于*出错点*的文本进行修改。

Number	Class	Type	MessageTag	MessageBi	Status tag	Status bit	Message text	Point of err
5	Error	Failure	U16i_ex_alg_03	1	U16i_ex_alg_05	1	Feedback Error	Motor 2
6	Error	Failure	U16i_ex_alg_03	2	U16i_ex_alg_05	2	Bimetal Error	Motor 2
7	Error	Failure	U16i_ex_alg_06	0	U16i_ex_alg_08	0	Lock Error	Motor 3
8	Error	Failure	U16i_ex_alg_06	1	U16i_ex_alg_08	1	Feedback Error	Motor 3
9	Error	Failure	U16i_ex_alg_06	2	U16i_ex_alg_08	2	Bimetal Error	Motor 3
10	Error	Failure	U16i_ex_alg_09	0	U16i_ex_alg_11	0	Lock Error	Motor 4
11	Error	Failure	U16i_ex_alg_09	1	U16i_ex_alg_11	1	Feedback Error	Motor 4
12	Error	Failure	U16i_ex_alg_09	2	U16i_ex_alg_11	2	Bimetal Error	Motor 4
13	Failure Containe	Alarm Contain		0	U16i_ex_alg_01	0	Critical Overpressure	Container 1
14	Failure Containe	Alarm Contain		0	U16i_ex_alg_01	1	Critical Temperature	Container 1
15	Failure Containe	Alarm Contain		0	U16i_ex_alg_04	0	Critical Overpressure	Container 2
16	Failure Containe	Alarm Contain		0	U16i_ex_alg_04	1	Critical Temperature	Container 2
17	Failure Containe	Alarm Contain		0	U16i_ex_alg_07	0	Critical Overpressure	Container 3
18	Failure Containe	Alarm Contain		0	U16i_ex_alg_07	1	Critical Temperature	Container 3
19	Failure Containe	Warning Cont		0	U16i_ex_alg_01	2	Warning pressure	Container 1
20	Failure Containe	Warning Cont		0	U16i_ex_alg_01	3	Warning temperature	Container 1
21	Failure Containe	Warning Cont		0	U16i_ex_alg_04	2	Warning pressure	Container 2
22	Failure Containe	Warning Cont		0	U16i_ex_alg_04	3	Warning temperature	Container 2
23	Failure Containe	Warning Cont		0	U16i_ex_alg_07	2	Warning pressure	Container 3
24	Failure Containe	Warning Cont		0	U16i_ex_alg_07	3	Warning temperature	Container 3

限制值监控的组态

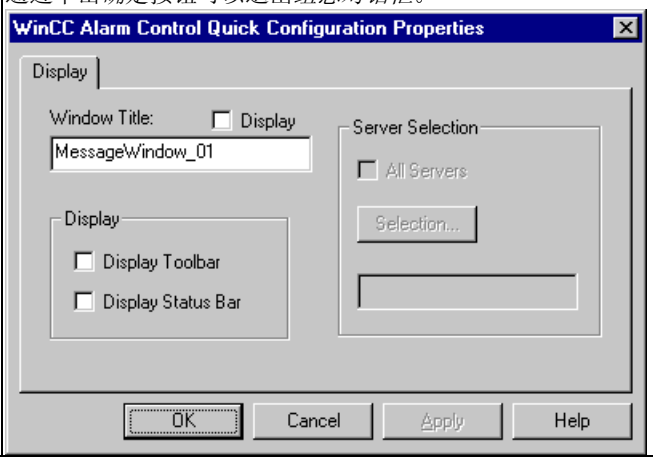
步骤	过程：限制值监控的组态
1	如果浏览窗口中没有显示*限制值监控*(模拟报警)条目，则必须将其装载上去。可通过*报警记录*中的*选项* → *添加插入*菜单完成该操作。在所显示的对话框中，必须对限制值监控条目打上复选标记。 
2	通过 OR *限制值监控*条目，然后从弹出菜单中选择新建，则可访问变量的*属性*对话框。在该对话框中，可设置一个用于限制值监控的新变量。

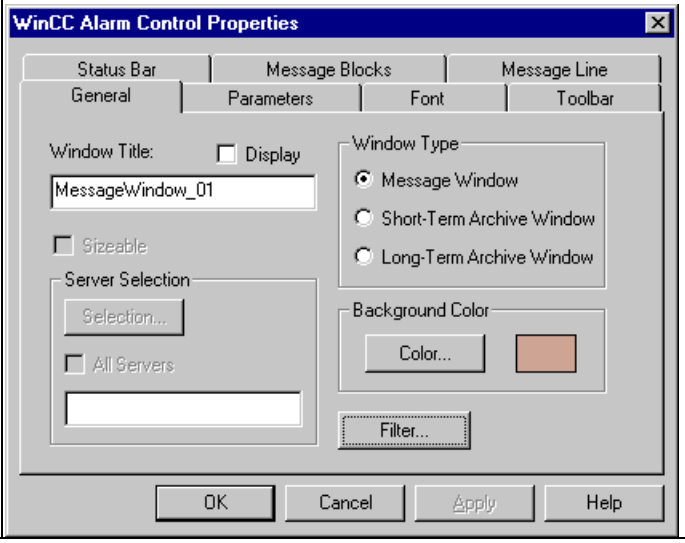
步骤	过程：限制值监控的组态
3	在该对话框中，将包含有第一个包容器的温度值的变量 *U16i_ex_alg_t1* 设置为*将要监控的变量*。复选框*所有限制值的消息*则不需选择。至于*延迟时间*，则保持为 *0*。 通过单击*确定*退出对话框。 
4	在右窗口中，将会显示所要监控的变量的图标。通过 OR 该图标，然后从弹出菜单中选择新建，可打开限制值的*属性*对话框。在该对话框中，可以将新的限制值分配给变量。

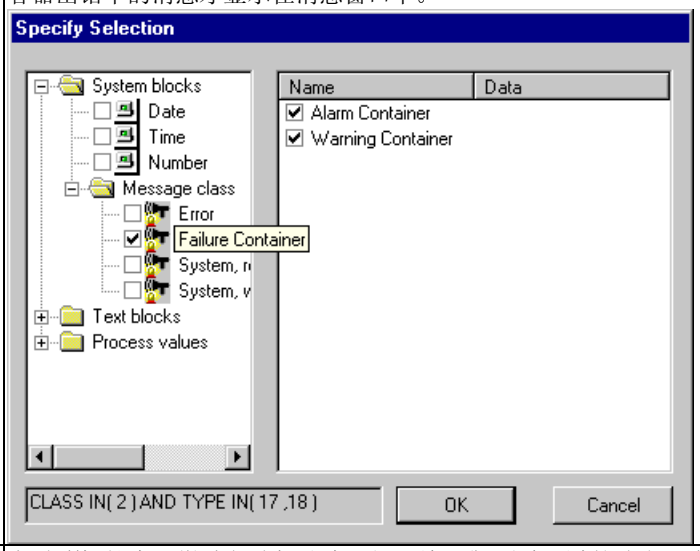
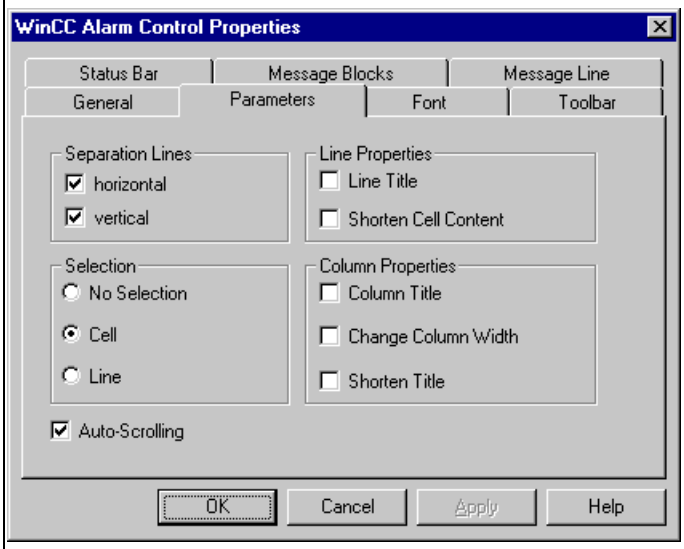
步骤	过程：限制值监控的组态
5	在*限制值*域中，设置*上限*。在*限制*值或*变量*域中，输入一个值为 800 的限制值。至于*滞后*，则保持为 *0*。对于*消息*，输入*编号 14*。这就是在第一个包容器中温度超过时的报警消息。 通过单击*确定*退出对话框。 对于同样的变量，可指定第二个限制值。在*限制值*域中，再设置一个*上限*。然而，在*限制值*或*变量*域中，输入限制值 *500*。对于*消息*，输入*编号 20*。这就是在第一个包容器中温度超出时的警告消息。 
6	按照步骤 2 和 3 所描述的那样创建其余 5 个受监控的变量，每个变量具有两个已组态的限制值。 通过双击*限制值监控*条目，则将显示所有已创建的变量。 

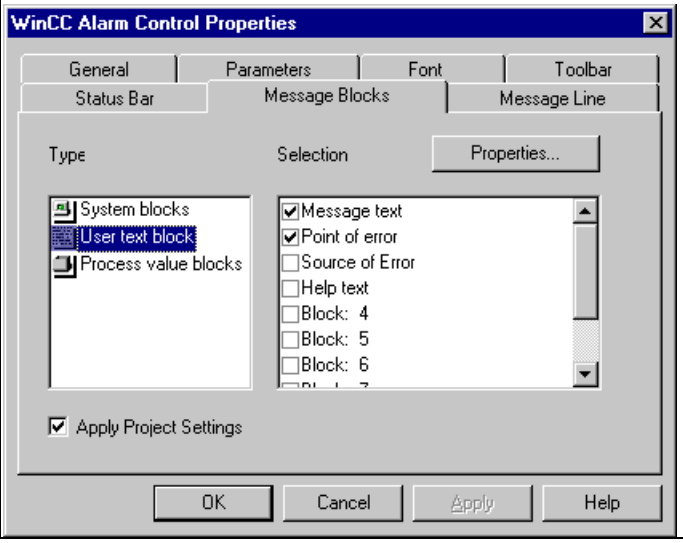
图形编辑器中的实现

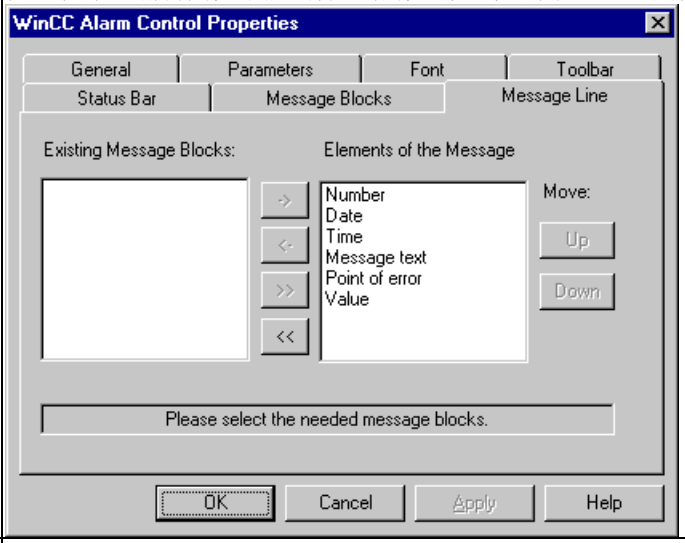
步骤	过程：图形编辑器中的实现
1	实现所要监控的过程值的模拟可通过各个 *Windows 对象* → *滑块对象*。在实例中，它们是*滑块对象 1* 到*滑块对象 6*。 对*滑块对象 1*，在*事件* → *属性主题* → *其它* → *过程驱动程序连接*处，创建一个*直接连接*，用于将*滑块*的当前过程值切换成变量 *U16i_ex_alg_t1*。这个*滑块*模拟第一个包容器中的温度值。采用同样的方法，为其余变量组态滑块。 为了在画面打开时，使滑块的位置与当前变量值匹配，可在*事件* → *其它* → *打开画面*处创建一个 *C 动作*，用于完成该任务。
2	此外，将一个*智能对象* → *I/O 域*分配给各个*滑块*，以便显示当前的变量值。在实例中，它们是 I/O 域 1 到 I/O 域 6 等对象。 对于 *I/O 域 1*，在*属性* → *其它* → *画面名称*处创建一个至变量 *U16i_ex_alg_t1* 的变量连接，并且一旦改变则将其触发。这就是分配给第一个*滑块*的 *I/O 域*。采用同样的方式，将 *I/O 域*分配给其余的各个*滑块*。
3	单个包容器的显示可通过标准库中的*智能对象 Tank4* 来实现。在本实例中，它们是对象 *Tank41*、*Tank42* 与 *Tank43*。 这些对象仅用于显示用途，且不会接收到任何动态信号。
4	可将*智能对象* → *状态显示分配给每个表示警告灯的包容器*。在该实例中，它们是对象状态显示 1 到状态显示 3。 本实例中，在当*前状态 0* 的*属性* → *状态*处，将位图 *Blinker blinkt nicht.gif* 设置为*基本画面*，而将位图 *Bliker blinkt.gif* 设置为*状态显示 1 的闪烁画面*。将*属性* → *状态* → *正在闪烁*的*闪烁画面已激活*设置为*否*。对于同样的属性，可创建一个 *C 动作*，如果对应包容器的报警消息是未决的，则它可以用来激活闪烁。采用同样的方式组态其它两个*状态显示*。

步骤	过程：图形编辑器中的实现
5	组态一个附加的*智能对象* → *状态显示*，*用于显示蜂鸣器*。在实例中，这就是对象*状态显示 4*。在*当前状态 0* 的*属性* → *状态*处，将位图 *Hupe hupt nicht.gif* 设置为*基本画面*，而将位图 *Hupe hupt.gi* 设置为该对象的*闪烁画面*。将*属性* → *状态* → *正在闪烁的闪烁画面已激活*设置为*否*。对于同样的属性，创建一个 *C 动作*，用于有关三个包容器当中某一个的报警消息到达时，就接通闪烁，也就是说，如果变量设置在用于*包容器出错*消息级别的*报警记录*中，用于控制中心指示器，则呈现为状态 1。在实例中，这就是变量 *U16i_ex_alg_10*。 在*属性* → *几何结构* → *宽度*处，创建一个 *C 动作*，用于在对象闪烁时，发出声音信号。
6	为了显示在*报警记录*中组态的消息，使用了一个 *WinCC 报警控件*。从对象工具板的*控件*选择菜单中选择它，然后将其置于画面中。
7	将控件置于画面中之后，将会自动显示其*组态对话框*。 输入 *MessageWindow_01*，将其作为*窗口标题*。*显示*复选框仍保持撤消选定状态。在稍后创建的 C 动作中，此窗口标题用于注明相应的控件。 *工具栏*与*状态栏*复选框均处于撤消选定状态。 通过单击*确定*按钮可以退出*组态对话框*。 

步骤	过程：图形编辑器中的实现
8	打开控件的*属性*对话框。该对话框通过 OD 控件来显示。在*常规信息*标签中，使用选择按钮来使背景色与现存的项目颜色表相匹配。 *选择*按钮用于选择预先在报警记录中创建的单个消息，并通过控件来显示。 

步骤	过程：图形编辑器中的实现
9	通过 *系统块消息级别* → *包容器出错*，可将 2 个复选框显示在右窗口中。选择复选框*包容器报警*与*包容器警告*。也就是说，在运行系统中，只有消息级别包容器出错中的消息才显示在消息窗口中。 
10	在*参数*标签中，撤消复选框*行标题*、*列标题*与*改变列宽*的选定。在*选择*域中，选择*表元*单选按钮。 

步骤	过程：图形编辑器中的实现
11	在*消息块*标签中，选择以后将显示在消息行中的列。在本实例中，使用在*类型*域中选择*系统块*。在右窗口中，选择*日期*、*时间*与*编号*。对于*用户文本块*条目，选择*消息文本*与*出错点*。对于*过程值块*条目，选择*数值*。 

步骤	过程：图形编辑器中的实现
12	在*消息行*标签中，将先前所选择的*消息块*分配给消息行。*可用的消息块*域将列出可用的列。通过按下->按钮，可将各个消息块添加到消息行。通过>>按钮，可将窗口中列出的所有消息块随时分配给消息行。单击*确定*，退出属性对话框。 
13	对于工具栏，可组态多个 *Windows 对象* → *按钮*，用于通过特定标准函数模拟单个按钮的按下动作。
14	组态一个用于对消息进行单个确认的*按钮*。如果触发了蜂鸣器，该按钮也可对其进行确认。相关的标准函数是： *AXC_OnBtnSinglAckn*(lpszPictureName，lpszObjectName) *AXC_OnBtnHornAckn(lpszPictureName，lpszObjectName)*
15	组态附加的按钮。一个用于组确认的*按钮*与一个用于调用信息文本对话框的*按钮*。相关的标准函数是： *AXC_OnBtnVisibleAckn*(lpszPictureName，lpszObjectName) *AXC_OnBtnInfo*(lpszPictureName，lpszObjectName)

步骤	过程：图形编辑器中的实现
16	可使用*智能对象* → *状态显示*作为自动滚动功能开/关按钮的替换对象。在本实例中，它是对象*状态显示 6*。 在*属性* → *状态* → *当前状态*处，创建一个变量 *BINi_ex_alg_00* 的*变量连接*。该变量包含是否接通或关闭自动滚动的信息。在*事件* → *鼠标* → *按下左键*处，创建一个 *C 动作*，用于对变量 *BINi_ex_alg_00* 的状态求反，并调用标准函数 *AXC_OnBtnScroll*(lpszPictureName,lpszObjectName)。在画面打开时，如果重新选择消息窗口，那么由于自动滚动接通，则变量 *BINi_ex_alg_00* 将设置为 *0*。

状态显示 1 的 C 动作

```
#include "apdefap.h"
 BOOL _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{
WORD state;

state=GetTagWord("U16i_ex_alg_01");

if ((state&1)||(state&2)) return TRUE;
else return FALSE;
}
```

- 第一个包容器状态变量的读取。如果报警消息是未决的，则将 *TRUE* 返回给属性，且警告灯闪烁。
- 根据第一个包容器的状态变量触发该 *C 动作*。

状态显示 4 的 C 动作

```
#include "apdefap.h"
 BOOL _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{
if (GetTagWord("U16i_ex_alg_10")&1) return TRUE;
else return FALSE;
}
```

- 如果触发了中心指示器，则将 *TRUE* 返回给属性，且蜂鸣器以视觉形式进行显示。
- 一旦控制中心指示器的变量改变就触发该 *C 动作*。

用于产生声响信号的 C 动作

```
#include "apdefap.h"
 long _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty
{
#pragma code ("winmm.dll")
BOOL PlaySound(LPCSTR pszSound,HMODULE hMod,DWORD fdwSound);
#define SND_FILENAME 0x00020000L
#define SND_ASYNC 0x0001
#pragma code ()

char szProjectName[MAX_PATH];
CMN_ERROR Error;
char szSoundFilePath[MAX_PATH] = "";
char szSoundFile[MAX_PATH] = "Hupe.wav";

if (GetFlashFlashPicture(lpszPictureName,lpszObjectName)) {
        if (DMGetRuntimeProject( szProjectName, MAX_PATH, &Error)) {
                strncat(szSoundFilePath,szProjectName,
                strlen(szProjectName)-strlen(strrchr(szProjectName,\\)+1));
        strcat(szSoundFilePath,szSoundFile);
        //MessageBeep((WORD)-1);
        PlaySound(szSoundFilePath,NULL,SND_FILENAME| SND_ASYNC);

        }
}

return 56;
}
```

- 装载 DLL *winmm.dll*。这个 DLL 包含用于执行声音文件的函数。
- 如果对象*状态显示 4* 闪烁，则执行 *Hupe.wav* 文件，该文件位于项目文件夹中。为此，必须通过 *DMGetRuntimeProject* 函数来确定项目文件夹，且要对文件路径进行设置。
- 调用函数 *PlaySound*。
- 以一秒的周期执行该 *C 动作*。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 为了满足用户需要，必须对已创建的消息级别进行修改。
- 必须根据需要对消息窗口的显示类型进行修改。

4.2.4 消息窗口(ex_3_chapter_02b.pdl)

任务定义

通过消息窗口监控多个过程。如果消息到达，工具栏上的按钮将允许跳转到产生错误的窗口。
使用*报警记录*的标准工具创建消息窗口，将使用标准工具栏和标准状态栏。

概念的实现

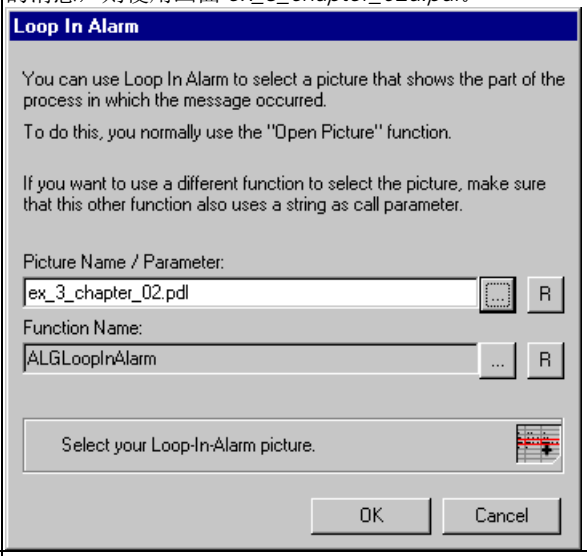
本实例将使用前面实例中所创建的消息和画面。如果按下工具栏上的*报警回路*按钮，需要一个项目函数执行画面切换。
可使用 WinCC 报警控件在*图形编辑器*中创建消息窗口。无需其它对象。

注意：
在前面两个实例中所作的组态将被视为已完成。它们将不再单独进行解释，然而本实例是以它们为基础。

实例的实现

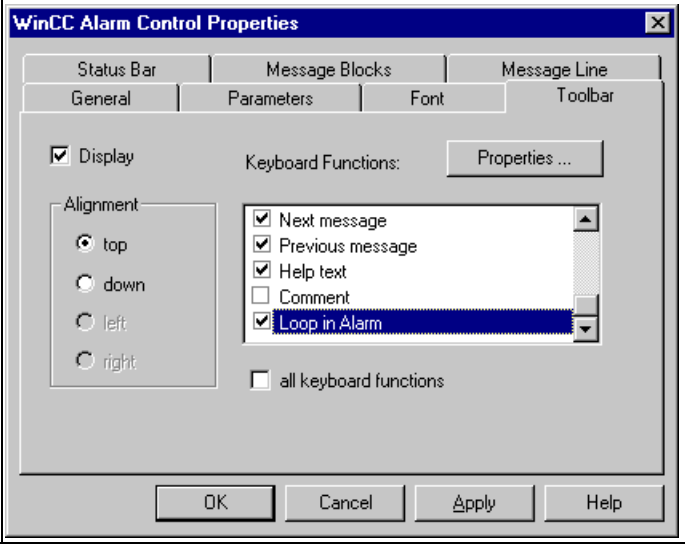
步骤	过程：实例的实现
1	从 *WinCC 资源管理器*中打开*报警记录*编辑器。
2	对于每一个单个消息，必须设置*报警回路*。该函数可以直接将一个画面切换至消息的对应画面。在缺省状态下，可将 *OpenPicture* 设置为执行画面切换的函数。然而，在本实例中，必须创建独立的函数以在画面窗口中执行画面的切换。该函数的参数调用由*报警记录*进行预定义。在本实例中，*ALGLoopInAlarm* 函数已在*全局脚本*编辑器中创建。
3	在*报警记录*的表格窗口中，D *报警回路*列，以便打开所选择的单个消息的*报警回路*对话框。

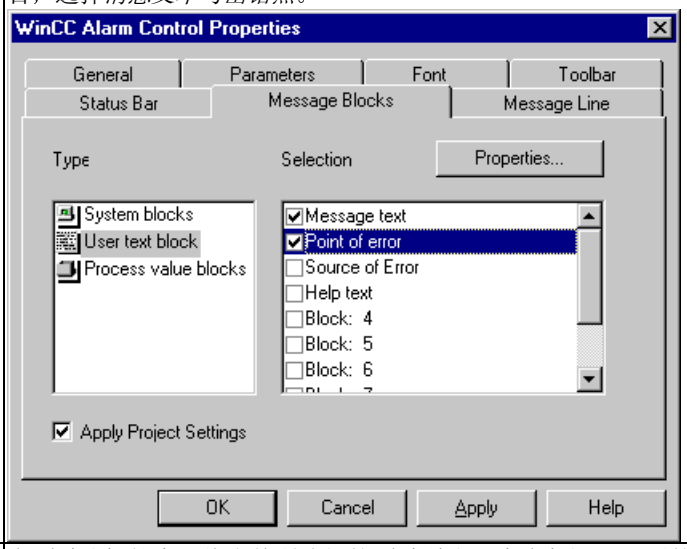
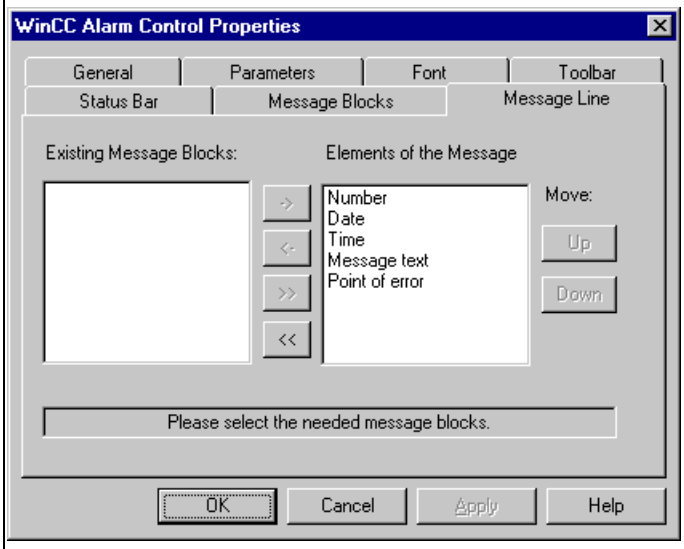
	Acknowledgement bit	Loop in Alarm	Group	F
	0	Not set		
	0	Not set		
	0	Not set		
	0	Not set		
	0	Not set		
	0	Not set		
	0	Not set		
	0	Not set		

步骤	过程：实例的实现
4	将使用函数 *ALGLoopInAlarm* 作为*函数名称*。对于与第一个实例中的电机有关的消息，画面 *ex_3_chapter_02.pdl* 将用作*画面名称/调用参数*，而对第二个实例中的消息，则使用画面 *ex_3_chapter_02a.pdl*。 
5	还可在单个消息属性对话框的*变量/动作*标签中的*报警回路*域处对*报警回路*函数进行组态。 保存在报警记录中所进行的组态。

图形编辑器中的实现

步骤	过程：
1	打开图形编辑器，并创建一个新的画面。在本实例中，它是画面 *ex_3_chapter_02b.pdl*。
2	为了显示在*报警记录*中组态的消息，可使用 *WinCC 报警控件*。从对象选项板的控件*选择*菜单中选择该控件，然后将其置于画面中。
3	将控件置于画面中之后，将会自动打开其*组态对话框*。 通过单击*确定*按钮可以退出*组态对话框*。 打开控件的*属性*对话框。通过 D 控件来显示此对话框。 在*常规信息*标签中，可进行所有的设置。不需要进行选择，因为所有可能出现的单个消息均将显示。

步骤	过程：
4	在工具栏标签中，选择下列复选框： • 单个确认 • 组确认 • 自动滚动开/关 • 报表函数 • 列表的开始 • 列表的结束 • 下一个消息 • 前一个消息 • 信息文本 • 报警回路 

步骤	过程：
5	在*消息块*标签中，选择以后将显示在消息行中的列。在本实例中，可使用在*类型*域中选择*系统块*。在右边窗口中，选择*日期*、*时间*与*编号*。对于*用户文本块*条目，选择*消息文本*与*出错点*。 
6	在*消息行*标签中，将先前所选择的*消息块*分配给消息行。*可用的消息块*域将列出可用的列。通过按下->按钮，可将各个消息块分别添加到消息行。通过>>按钮，可一次将窗口中列出的所有消息块分配给消息行。单击*确定*，退出属性对话框。 

项目函数 ALGLoopInAlarm

```
void ALGLoopInAlarm(char* PictureName)
{
SetPictureName("ex_0_startpicture_00.pdl","workspace", PictureName);
}
```

- 调用 *SetPictureName* 函数来完成画面的切换。由于调用参数的数目和类型与指定的不匹配，该函数不能直接在*报警记录*中使用。

注意：
在 WinCC 报警控件的工具栏中，提供了一个用于报表函数的按钮。在*报表编辑器*一章的消息顺序报表(ex_3_chapter_02b.pdl)实例中对消息顺序报表及其激活的执行过程进行了描述。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 必须对所组态的用于单个消息的*报警回路*函数进行修改以满足用户需要。
- 必须对消息窗口的显示类型进行修改以满足用户需要。

4.2.5 消息归档(ex_3_chapter_02c.pdl)

任务定义

将消息归档创建为一个用于 200 条消息的短期归档。所有消息将在消息窗口内显示。
消息窗口由用户定义的工具栏控制。该工具栏应包含两个专门的选择按钮，允许用户从实例 1 或实例 2 中显示消息。

概念的实现

本实例将使用前面实例中所创建的消息。此外，组态一个消息归档。
可使用 WinCC 报警控件在*图形编辑器*中创建消息窗口。可使用多个 *Windows 对象* → *按钮*、*智能对象* → *状态显示*与*智能对象* → *图形对象*执行该工具栏。
如果按下了选择按钮，则需要一个项目函数以便在消息窗口中进行选择。

所需变量的创建

步骤	过程：所需变量的创建
1	总共创建三个*二进制*变量类型的变量。在本实例中，它们是 *BINi_ex_alg_00*、*BINi_ex_alg_01* 与 *BINi_ex_alg_02* 变量。

注意：

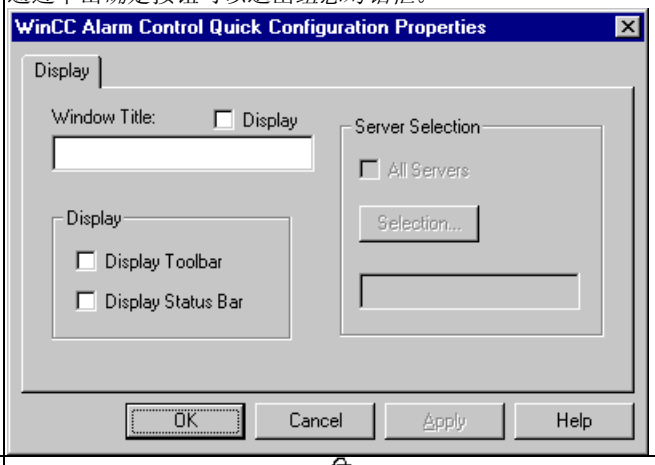
在第一和第二个实例中所作的组态将被视为已完成。它们将不再单独进行解释，然而本实例是以它们为基础。

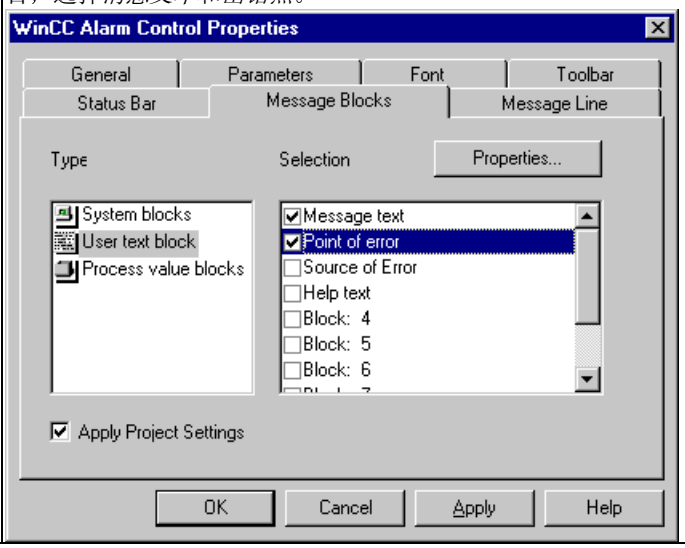
在报警记录中的实现

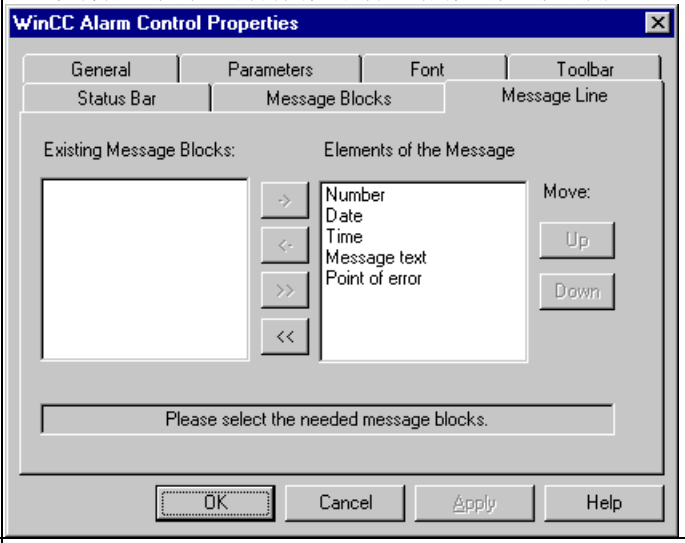
步骤	过程：在报警记录中的实现
1	从 *WinCC 资源管理器*中打开*报警记录*编辑器。
2	通过 OR *归档*条目，可打开*归档参数*对话框。

步骤	过程：在报警记录中的实现
3	在该对话框中，选择短期归档复选框。
4	在右边窗口中，将显示短期归档图标。通过 R 该图标，可打开短期归档的属性对话框。
5	归档将保存在硬盘上。在*条目编号*输入域中，指定 *200*。不用执行*选择*。

图形编辑器中的实现

步骤	过程：图形编辑器中的实现
1	打开*图形编辑器*并创建一个新的画面。在本实例中，它是画面 *ex_3_chapter_02c.pdl*。
2	将控件置于画面中之后，将会自动显示其*组态对话框*。 输入 *MessageWindow_04* 作为*窗口标题*。*显示*复选框保持未选择状态。在后面创建的 C 动作中，此窗口标题用于注明相应的控件。 *工具栏*与*状态栏*复选框处于未选择状态。 通过单击*确定*按钮可以退出*组态对话框*。 
3	打开控件的*属性*对话框。通过 D 控件来显示此对话框。 在*常规信息*标签中，可进行所有的设置。无需进行选择，因为所有可能出现的单个消息均将显示。

步骤	过程：图形编辑器中的实现
4	在*消息块*标签中，选择以后将显示在消息行中的列。在本实例中，可使用在*类型*域中选择*系统块*。在右边窗口中，选择*日期*、*时间*和*编号*。对于*用户文本*块条目，选择*消息文本*和*出错点*。 

步骤	过程：图形编辑器中的实现
5	在*消息行*标签中，将先前所选择的*消息块*分配给消息行。*可用的消息块*域将列出可用的列。通过按下->按钮，可将各个消息块分别添加到消息行。通过>>按钮，可一次将窗口中列出的所有消息块分配给消息行。单击*确定*退出属性对话框。 
6	对于工具栏，可组态多个 *Windows 对象* → *按钮*，用于通过特定的标准函数模拟单个按钮的按下动作。
7	组态一个用于调用选择对话框的*按钮*和一个用于调用信息文本对话框的*按钮*。相关的标准函数是： *ACX_OnBtnInfo()ACX_OnBtnSelect*()
8	可使用*智能对象* → *状态显示*来代替具有打开和关闭自动滚动功能的按钮。在本实例中，使用对象*状态显示 3*。 在*属性* → *状态* → *当前状态*处，创建一个至 *BINi_ex_alg_00* 变量的*变量连接*。该变量包含了有关打开还是关闭自动滚动功能的信息。在*事件* → *鼠标* → *按下左键*处，创建一个 *C 动作*，用于对 *BINi_ex_tlg_00* 变量的状态求反，并调用标准函数 *ACX_OnBtnScroll*()。在打开画面时，*BINi_ex_alg_00* 变量设置为 *0*，这是因为如果重新选择消息窗口，自动滚动功能被转换。

步骤	过程：图形编辑器中的实现
9	如果自动滚动关闭，则通过四个特定的按钮实现消息窗口中的浏览。使用下列标准函数可使这些按钮替代标准工具栏上的相应按钮： *ACX_OnBtnMsgFirst()ACX_OnBtnMsgLast()ACX_OnBtnMsgNext()ACX_OnBtnMsgPrev()* 当打开自动滚动时，通过将*智能对象* → *图形对象*放置在这些按钮上来使这些按钮不可操作。可通过*属性* → *其它* → *显示*处的与 *BINi_ex_alg_00* 变量的*变量连接*完成此操作。
10	通过两个*智能对象* → *状态显示*，可实现显示类型*消息窗口*与*短期归档窗口*之间的切换。消息窗口的当前状态存储在 *BINi_ex_alg_01* 变量中，由于该消息窗口在重新打开时将作为*短期归档窗口*显示，所以在打开画面时该变量必须设置为零。对于*状态显示 1*，在*属性* → *状态* → *当前状态*处创建一个与 *BINi_ex_alg_01* 变量的*变量连接*。在*属性* → *其它* → *操作员控制允许*处，创建一个*动态对话框*，它可使对象只有在消息窗口显示短期归档时才能操作，也就是说 *BINi_ex_alg_01* 变量具有状态 0。在*事件* → *鼠标* → *按下左键*处，创建一个 *C 动作*，用于模拟在工具栏上按下相关按钮，并对 *BINi_ex_alg_01* 变量取反。采用同样方式对*状态显示 2* 对象进行组态。使用下列标准函数： *ACX_OnBtnMsgWin()ACX_OnBtnArcShortt()*
11	通过另两个 *Windows 对象* → *按钮*，在消息窗口中进行直接选择。可选择查看与电机或容器相关的消息。可通过*全局脚本*编辑器中所创建的项目函数完成该选择。消息编号(所显示的消息位于其中)将被传送给该函数。在本实例中，该函数称为 *SetMsgNrSelection*。

用于设置选择的项目函数

```
BOOL SetMsgNrSelection(DWORD dwFrom, DWORD dwTo, LPSTR MsgTem)
{

   PCMN_ERROR          pError;
   BOOL                fRet;
   MSG_FILTER_STRUCT   Filter;

   memset(&Filter, \0,  sizeof( MSG_FILTER_STRUCT ) );
   strcpy( Filter.szFilterName, MsgTem);
   Filter.dwFilter = MSG_FILTER_NR_FROM|MSG_FILTER_NR_TO;
   Filter.dwMsgNr[0] = dwFrom;
   Filter.dwMsgNr[1] = dwTo;

   fRet = MSRTSetMsgWinFilter( &Filter, pError );

   if (fRet == FALSE)
   {
        printf("Error MSRTSetMsgWinFilter\r\n" );
        return FALSE;
   }
   else
        return TRUE;

}
```

- 为已创建的 *Filter* 过滤器结构保留存储空间。
- 将数值分配给与该应用相关的过滤器结构的结构成员。对于 *szFilterName*，必须使用与过滤器有关的消息窗口模板名称。在 *dwMsgNr* 数组中，输入所选择的消息编号的开始值和结束值。这些数值在函数调用时作为传送参数提供。通过将过滤器结构识别为编号过滤器的方式来设置 *dwFilter* 开关。
- 调用 API 函数 *MSRTSetMsgWinFilter*，该函数可将已创建的过滤器应用到所选择的消息窗口模板中。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 必须对消息窗口的显示类型进行修改以满足需要。
- 必须修改工具栏的外观和元素以满足需要。

4.2.6 组消息(ex_8_generator_00.pdl)

在运行系统中，访问与该主题相关的实例可通过使用选择上面所显示的*按钮*来进行。通过在画面中标记为激活的复选框，打开消息生成器。该消息生成器将以 10 秒的周期创建不同的消息。

任务定义

在画面中，将显示警告信息以指出出现了某种类型的消息。
这些信息已在位消息过程(ex_3_chapter_02.pdl)与限制值的监控(续)实例中进行了组态，且已应用于该实例。可以指出容器画面中的未决警告和报警以及出现在电机画面上的错误。报警的优先级将高于故障与出错信息。如果消息是未决的，则可通过按钮，跳转到相应的画面中。

概念的实现

所监控的单个消息已连接成一个组消息。只要生成一个单个消息，就会生成组消息。将给该组消息分配一个状态变量和一个状态位。
通过*智能对象* → *状态显示*，可对该变量的当前状态进行评估，并显示一个适当的符号。

注意：
第一个和第二个实例中所作的组态被看作是完整的。它们将不能再分开进行解释，然而，本实例将以它们为基础。

所需变量的创建

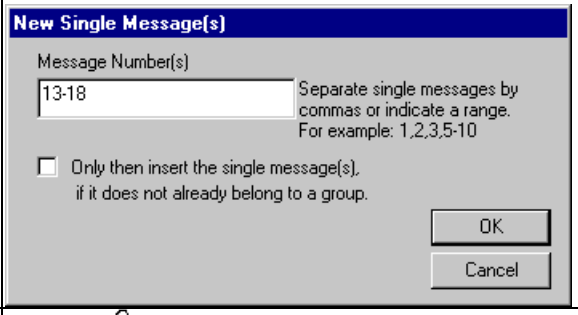
步骤	过程：所需变量的创建
1	在变量管理器中总共创建 3 个*无符号 16 位数*类型的变量。在本实例中，它们是变量 *U16i_ex_alg_20*、*U16i_ex_alg_21* 与 *U16i_ex_alg_22*。它们用作状态、锁定和确认变量。

常规信息

如果创建了新的消息等级，用于该消息等级的组消息也将自动创建。这个消息等级内的全部消息都将传送给组消息。组消息中消息等级和消息类型的属性可以独立改变，因此可链接到各种状态、锁定和确认变量上。
然而，在本实例中，假定使用同一消息等级的其它画面均在项目中存在。也就是说，由于相应的组消息也必须识别其中所出现的画面，所以不可使用自动生成的组消息。
这意味着必须创建由用户定义的组消息。

新组消息的创建

步骤	过程：新组消息的创建
1	打开*报警记录*编辑器。 D *组消息*条目以便将其展开，将显示 2 个子条目。它们是*消息等级*与*用户定义*的点。通过R *用户定义*的条目，可访问*新建组消息*对话框。 Group messages Message Class User defined
2	在所显示的对话中，将*报警包容器*作为*名称*输入。将先前所创建的变量设置为*状态*、*锁住*与*确认变量*。在每种情况下，均使用 *0* 作为位编号。 通过单击*确定*退出对话框。 Properties General Information Parent : User defined Name : Alarm Container Status Tag: U16i_ex_alg_20 ... R Status Bit: 0 Lock Tag: U16i_ex_alg_21 ... R Lock Bit: 0 Acknowledge Tag: U16i_ex_alg_22 ... R Acknowledge Bit: 0 OK Cancel Apply Help
3	采用同样的方式，可创建 2 个附加的组消息。它们使用同样的*状态*、*锁住*与*确认变量*，但分别使用位编号 1 和 2。在右边窗口中，将显示新创建的组消息图标。 Alarm Container Failure Motor Warning Container

步骤	过程：新组消息的创建
4	通过这些图标中的一个，即可打开*新建的单个消息*对话框。对于各个组消息，可输入相应单个消息的消息编号，然后单击*确定*将对话框关闭。 
5	通过使用D 浏览器窗口中的*自定义*的条目，可显示作为子条目的单个组消息。如果使用选择了这些条目中的某一个，则右窗口将显示所有添加的单个消息的图标。

图形编辑器中的实现

步骤	过程：图形编辑器中的实现
1	在*图形编辑器*中创建一个新的画面。在本实例中，它就是画面 *ex_8_generator_00*。 在该画面中，组态了一个*智能对象* → *状态显示*，它可显示组消息的当前状态。在本实例中，它就是对象*状态显示 1*。对下列组态，存储组消息的状态可通过变量 *U16i_ex_alg_20* 中的*报警记录*来进行。 对于每种状态，均必须设计一个对应的位图。也就是说，需要可用于三个未确认状态、三个已确认状态和无消息未决状态的位图。在*属性* → *状态* → *当前状态*处，创建了一个 *C 动作*，用于根据变量 *U16i_ex_alg_20* 与所需要的优先级来控制状态。

步骤	过程：图形编辑器中的实现
2	此外，还组态了一个 *Windows 对象* → *按钮*，用于在显示组消息的情况下，完成将画面切换到消息起始画面的过程。在本实例中是对象*按钮 1*。 通过*事件* → *鼠标* → *鼠标动作*处的 *C 动作*，可请求组消息的当前状态，并完成对应的画面切换。如果没有任何未决的消息，则可将一个附加的 *Windows 对象* → *按钮*放置在刚才所描述的按钮上。该按钮将使其它按钮不能操作，且将它显示为可视的。在*属性* → *其它* → *操作员控制允许*处，将该按钮设置为*否*。
3	组态另一个 *Windows 对象* → *按钮*，使用它可对当前所显示的组消息进行确认。在本实例中，这就是对象*按钮 3*。 通过*事件* → *鼠标* → *鼠标动作*处的 *C 动作*，可决定是否需要对组消息进行确认，如果需要，是哪一个。如果需要对消息进行确认，则应对所组态的确认变量 *U16i_ex_alg_22* 中的对应位进行设置，然后，立即重新进行设定。如果没有任何未确认的消息是未决的，则将附加的 *Windows 对象* → *按钮*放置在刚才所描述的按钮上，以便使其失去作用，并将其显示为可视的。在*属性* → *其它* → *操作员控制允许*处，将该按钮设置为*否*。
4	组态另一个画面；在本实例中，它就是画面 *ex_8_generator_01*。 在该画面中，组态了 3 个 *Windows 对象* → *复选框*。在本实例中，它们是对象*复选框 1*、*复选框 2* 与*复选框 3*。 在*事件* → *属性主题* → *输出/输入* → *所选方框*处，为每个复选框创建一个 *C 动作*，用于锁住或激活相应的组消息。可通过 *U16i_ex_alg_21* 变量中*报警记录*，根据组态对各自的锁定状态进行存储。由于锁定也可从其它侧面进行设置，因此，必须创建一个 *C 动作*，它位于*属性* → *输出/输入* → *所选方框处*。一旦改变变量 *U16i_ex_alg_21*，该动作就将触发，并检查由对应的*复选框*控制的锁定的状态是否改变。

步骤	过程：图形编辑器中的实现
5	在起初创建的画面 *ex_8_generator_00* 中，在*属性* → *其它* → *画面名称*处创建一个*智能对象* → *画面窗口*，画面 *ex_8_generator_01* 设置在其中。将*属性* → *其它* → *显示*设置为*否*。 还需要另一个*窗口对象* → *按钮*，它将使先前所组态的画面窗口成为可见，这可通过*事件* → *鼠标* → *鼠标动作*处的一个*直接连接*来完成。

用于确定当前状态的 C 动作

```
#include "apdefap.h"
 long _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty
{
WORD state;

state = GetTagWord("U16i_ex_alg_20");

if ((state&1)&&(state&256)) return 6;
else if ((state&2)&&(state&512)) return 5;
else if (state&1) return 3;
else if ((state&4)&&(state&1024)) return 4;
else if (state&2) return 2;
else if (state&4) return 1;
else return 0;
}
```

- 通过*报警记录*读取所写入的状态变量。
- 根据该变量设置当前状态。如果多个组消息都未决，由所定义的优先级顺序将决定显示哪一个消息。本例中，从最高优先级水平开始的优先级顺序如下：
- 容器报警
- 电机故障
- 确认的容器报警
- 容器警告
- 确认电机故障
- 确认的容器警告

执行画面切换的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{
Int value;

Value = GetIndex(lpszPictureName,"Status Display1");

If ((value==2)||(value==5))
        SetPictureName("ex_0_startpicture_00.PDL",
        "workspace","ex_3_chapter_02.PDL");
else if (value>0)
        SetPictureName("ex_0_startpicture_00.PDL",
        "workspace","ex_3_chapter_02a.PDL");

}
```

- 确定状态显示的当前显示状态。
- 根据所显示的状态，完成画面的切换。如果状态为 0，则没有动作执行。

用于确认所显示的消息的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{
WORD state;

state = GetTagWord("U16i_ex_alg_20");

if ((state&1)&&(state&256)){
        SetTagWord("U16i_ex_alg_22",
                (WORD)(1|GetTagWord("U16i_ex_alg_22")));
        SetTagWord("U16i_ex_alg_22",
                (WORD)(~1&GetTagWord("U16i_ex_alg_22")));
        }
else if ((state&2)&&(state&512)){
        SetTagWord("U16i_ex_alg_22",
                (WORD)(2|GetTagWord("U16i_ex_alg_22")));
        SetTagWord("U16i_ex_alg_22",
                (WORD)(~2&GetTagWord("U16i_ex_alg_22")));
        }
else if ((state&4)&&(state&1024)){
        SetTagWord("U16i_ex_alg_22",
                (WORD)(4|GetTagWord("U16i_ex_alg_22")));
        SetTagWord("U16i_ex_alg_22",
                (WORD)(~4&GetTagWord("U16i_ex_alg_22")));
        }
}
```

- 读入组消息的当前状态。
- 如果将被确认的消息是未决的，它将被确认。如果将被确认的多个消息都是未决的，将确认具有最高优先级的一条消息。

用于设置锁定的 C 动作

```
#include "opdcfop.h"
void OnPropertyChanged(char* lpszPictureName, char* lpszObjectName, char* l
{
DWORD   dwServiceID;
MSG_RTGROUPSET_STRUCT mGroup;
CMN_ERROR Error;
BOOL fRet;
time_t Time;
struct tm*     TimeStruct;

time(&Time);
TimeStruct = localtime(&Time);

mGroup.stTime.wYear=(WORD)(TimeStruct->tm_year+1900);
mGroup.stTime.wMonth=(WORD)(TimeStruct->tm_mon+1);
mGroup.stTime.wDay=(WORD)(TimeStruct->tm_mday);
mGroup.stTime.wHour=(WORD)(TimeStruct->tm_hour);
mGroup.stTime.wMinute=(WORD)(TimeStruct->tm_min);
mGroup.stTime.wSecond=(WORD)(TimeStruct->tm_sec+1);

mGroup.fIDUsed=FALSE;
strcpy(mGroup.szName,"Alarm Behälter");
mGroup.dwData=value;

MSRTStartMsgService(&dwServiceID,NULL,NULL,
                        MSG_NOTIFY_MASK_ALL,(LPVOID)0,&Error);

fRet=MSRTLockGroup (dwServiceID,&mGroup,&Error);
if (fRet==FALSE)
        printf("Error in MSRTLockGroup(:::) %s\r\n",Error.szErrorText);
else  printf("Executed MSRTLockGroup(:::) \r\n");


MSRTStopMsgService (dwServiceID,&Error );
}
```

- 定义所需要的变量。*mGroup* 是一种必须传送给函数的结构，它主要负责设置锁定。
- 确定当前系统时间。该数值被传送给 SYSTEMTIME 类型的 *stTime* 结构成员。
- *fIDUsed* 结构组元指明了所期望的组消息(将被锁定或激活)是否利用其 ID 的名称来进行定义。数值 FALSE 指定组消息通过其名称来识别。
- *szName* 包含所期望的组消息名。
- *dwDate* 指示是否进行设置或锁定。将当前状态传送到复选框。
- 消息服务的启动可通过函数 *MSRTStartMsgService* 来进行。
- 调用用于锁定或激活组消息 *MSRTLockGroup* 的函数。
- 消息服务的结束可通过函数 *MSRTStopMsgService* 来进行。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 构成组消息的单个消息必须进行修改，以满足需要。
- 组消息的显示、显示优先级以及所要完成的画面切换均必须进行修改，以满足需要。

4.3 报表编辑器

在运行系统中，可通过使用选择如上所示的按钮来访问与该主题有关的实例。这些实例组态在画面 *ex_3_chapter_03.pdl* 中。其余实例分布于整个实例项目中。

常规信息

*报表编辑器*是 WinCC 基本工具包的一部分，它提供了报表的创建和输出功能。该创建与*报表编辑器*的组态系统中的报表布局的创建相关，而输出与报表的打印相关。

注意：
所提供的系统布局可直接使用或复制，然后根据用户规定进行修改。系统布局和系统打印作业的名称通常以字符@开头。

4.3.1 画面文档(ex_3_chapter_03.pdl)

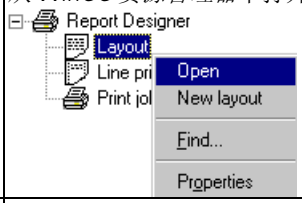
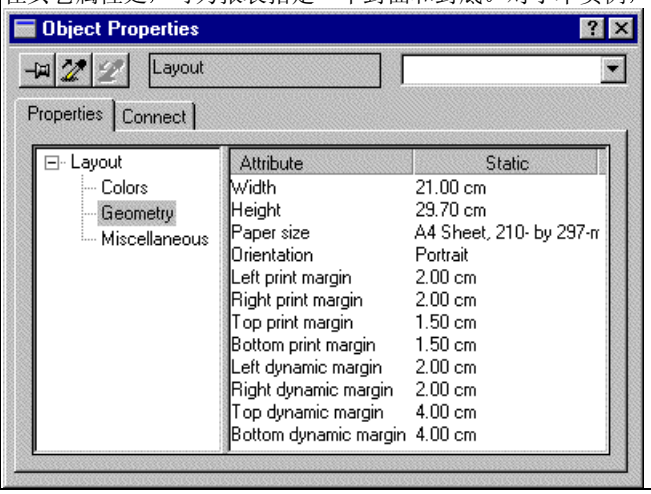
任务定义

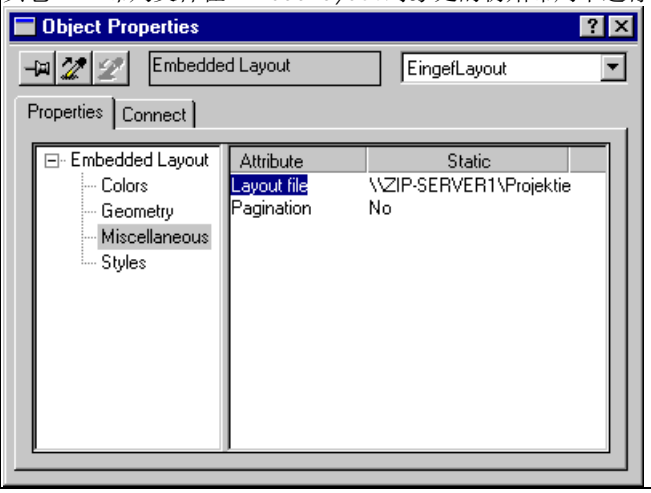
将要创建一个在项目中包含的所有画面的综合文件。该文件应该包括每个画面的图形显示、常规信息、所有对象列表和所有设置的画面属性的列表。

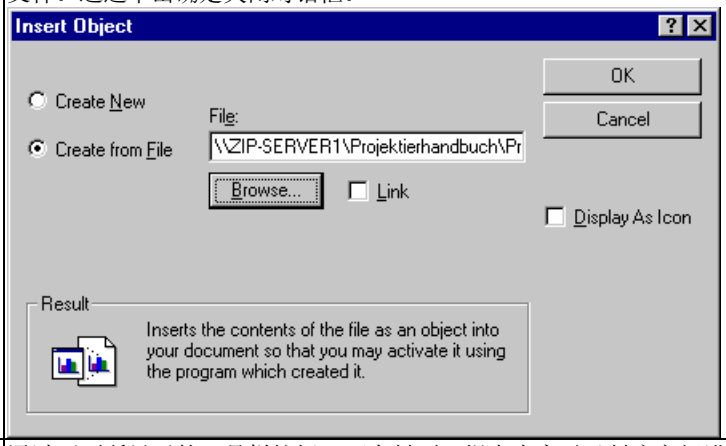
概念的实现

可使用一种符合要求的系统布局。它就是布局@PDLpicture(compact).rpl。可以复制并修改该布局来满足用户需要。

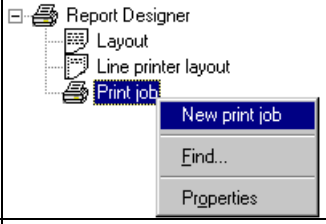
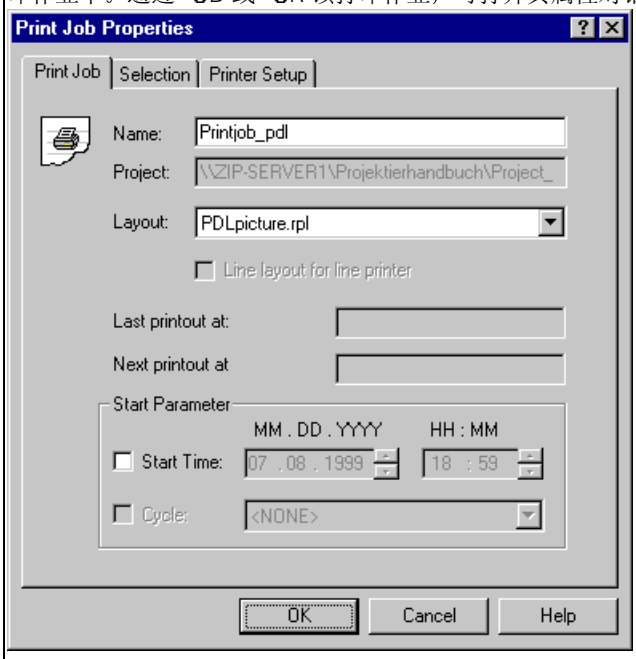
在报表编辑器中的实现

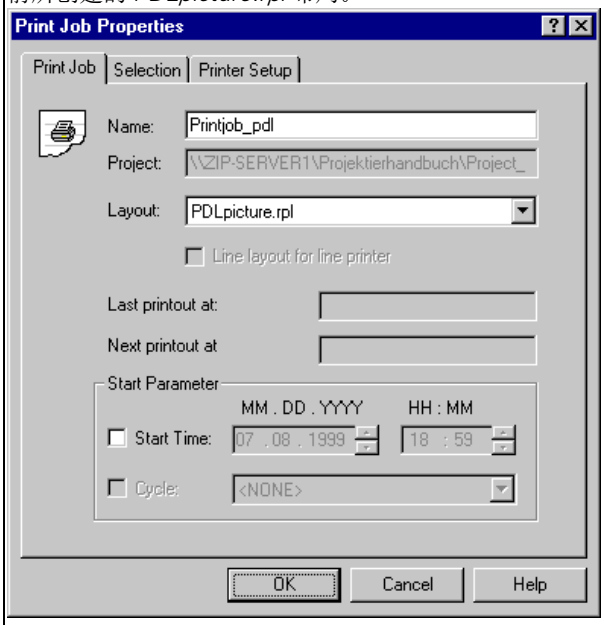
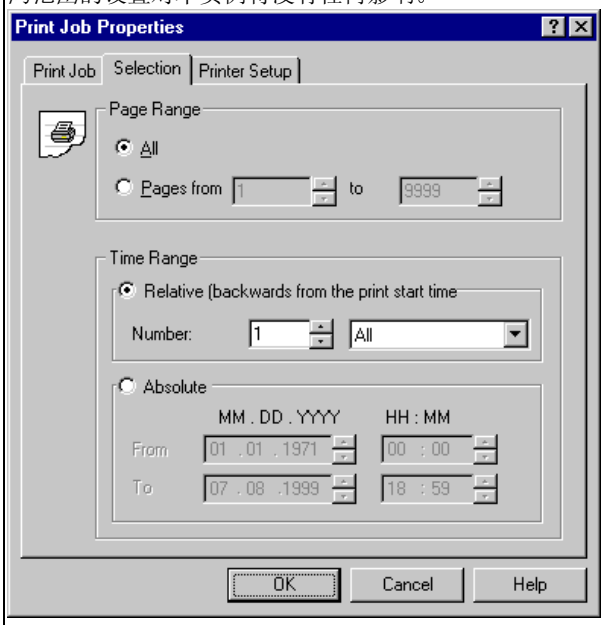
步骤	过程：在报表编辑器中的实现
1	从 *WinCC 资源管理器*中打开*报表编辑器*。 
2	通过*文件* → *打开*...菜单打开系统布局@PDLPic.rpl，并使用*文件* → *另存为*...菜单，将其以另外的名称进行保存。在本实例中，使用了名称 *PDLpicture.rpl*。
3	通过 R 布局空白处，可打开其属性对话框。 在*属性*标签中，可在*几何图形*属性处进行常规几何属性的设置。 在*其它*属性处，可为报表指定一个封面和封底。对于本实例，指定了一个封面。 

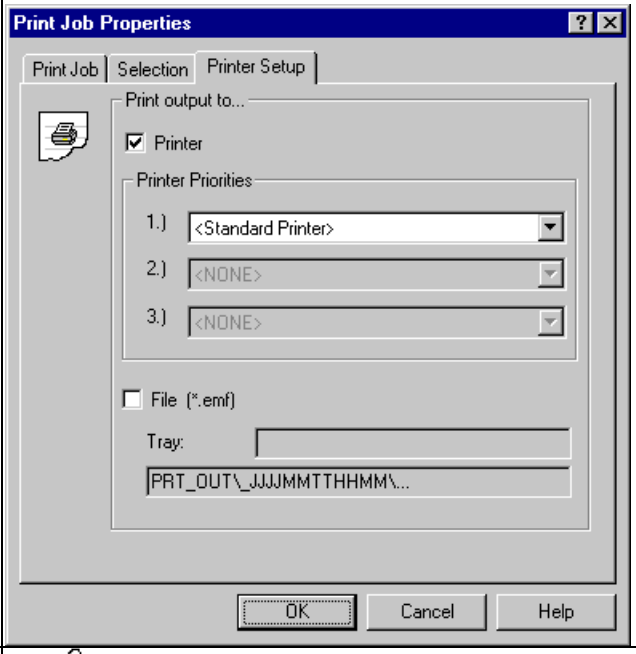
步骤	过程：在报表编辑器中的实现
4	通过下面所显示的工具栏按钮，可对报表的静态与动态部分进行编辑。 Static Part Dynamic Part
5	报表的动态部分包括一个*动态对象* → *嵌入布局*。在本实例中，它就是 *EmbedLayout* 对象，必须对其进行修改。通过 R 报表的动态部分，在*属性* → *其它* → *布局文件*处选择*@PDLpicture (compact).rpl* 布局。 可打开该布局并修改其元素以满足用户需要。然而，建议首先复制该布局，然后在复制件中进行修改。如果已进行了此操作，则新创建的布局必须通过*属性* → *其它* → *布局文件*在 *EmbedLayout* 对象处的初始布局中进行设置。 

步骤	过程：在报表编辑器中的实现
6	报表的静态部分包含页眉和页脚。 页脚包括*系统对象日期/时间*、*页码*、*项目名称*和*布局名称*。 页眉包括两个*静态对象* → *静态文本*以及一个*系统对象* → *项目名称*。此外，可使用*静态对象* → *OLE 元素*来显示标识符。在本实例中，将 *StatText1* 对象的文本修改为*画面文档*。为了显示用户自己的标识符，需将现有的 *OLEElement1* 对象删除。现在，即可组态一个新的*静态对象* → *OLE 元素*。在将对象放入报表后所显示的*插入对象*对话框中，选择*从文件创建*选项钮，并指定包含标识符的位图文件。通过单击*确定*关闭对话框。 
7	通过下面所显示的工具栏按钮，可在封面、报表内容以及封底之间进行切换。 在本实例中，封面包括两个静态对象 → *静态文本*、*一个系统对象* → *项目名称*和一个*静态对象* → *静态图元文件*。
8	保存在报表编辑器中进行的修改，然后退出报表编辑器。

打印作业的创建

步骤	过程：打印作业的创建
1	在 *WinCC 资源管理器*中，通过 R 创建一个新的打印作业。 
2	在右边的窗口中，具有缺省名称*打印作业 001* 的新打印作业将被添加到现有的打印作业中。通过 D 或 R 该打印作业，可打开其属性对话框。 

步骤	过程：打印作业的创建
3	在*打印作业*标签中，缺省名称将由 *Printjob_pdl* 来代替。至于*布局*，则可指定先前所创建的 *PDLpicture.rpl* 布局。 
4	在*选择*标签中，可指定所要打印的范围。在*页面范围*域内，选择*全部*选项钮。*时间范围*的设置对本实例将没有任何影响。 

步骤	过程：打印作业的创建
5	在*打印机设置*标签中，可指定所要使用的打印机。也可将数据打印至一个文件。 
6	通过 R，可从 *WinCC 资源管理器*中启动打印作业。采用同样的方式可显示打印预览。
7	在实例项目中，可通过 *Windows 对象* → *按钮*来激活打印作业的预览。它就是 *ex_3_chapter_03.pdl* 画面中的*按钮 13* 对象。 在*事件* → *鼠标* → *鼠标动作*处，创建一个 *C 动作*，用于启动相应打印作业的预览。

用于启动打印作业的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

RPTJobPreview("PrintJob_pdl");

}
```

- 调用 *RPTJobPreview* 标准函数。使用打印作业名称来作为传送参数。

注意：
如果嵌入在报表布局中的布局在*报表编辑器*中打开，则不能启动预览与打印作业。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 所创建的布局不用修改即可使用。报表所显示的标识符与信息必须进行修改。可根据*报表编辑器中的实现*的步骤 3 与步骤 4 的描述来完成这些修改。

4.3.2 WinCC 资源管理器的报表(ex_3_chapter_03.pdl)

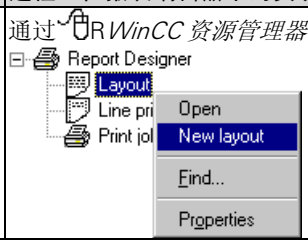
任务定义

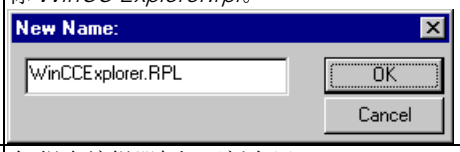
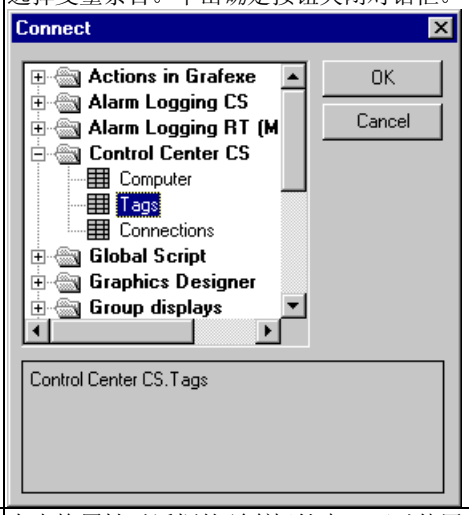
在项目中使用的特定变量组的所有变量将被记录成文档。该文档应包含变量名称、变量类型、变量组、变量参数和与变量过程连接有关的信息。

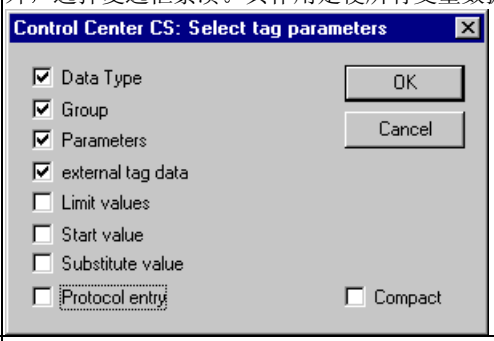
概念的实现

在*报表编辑器*中组态一个独立的布局。此布局不基于任何现有的布局。

在报表编辑器中的实现

步骤	过程：在报表编辑器中的实现
1	通过 R *WinCC 资源管理器*中的*页面布局*条目创建新的布局。 

步骤	过程：在报表编辑器中的实现
2	在右边窗口中，新布局将以缺省名称 *NewRPL00.RPL* 添加到现有的页面布局中。通过R，可将该布局重命名为一个更有意义的名称。在本实例中，使用名称 *WinCC Explorer.rpl*。 
3	在*报表编辑器*中打开新布局。 在布局的属性对话框(通过R 报表页面的空白空间可访问到)中，在*其它属性*处指定*封面*。保留其余设置。
4	在报表的静态部分，为页眉和页脚组态各种*静态对象*和*系统对象*。 封面设计仅是对个人自行设计的建议。
5	在报表的动态部分，组态了*动态对象* → *动态表格*。在本实例中，这是对象 *DynTable1*。 在将对象放入报表中后，将显示*连接*对话框。从 *WinCC 资源管理器*文件夹中，选择*变量*条目。单击*确定*按钮关闭对话框。 
6	在表格属性对话框的*连接*标签中，可以使用若干个选择项。 Tag parameter selection Tag group selection Tag selection Format

步骤	过程：在报表编辑器中的实现
7	通过 D 其中一个条目，可访问相关对话框进行数据选择。如果进行了选择，则用红色复选标记表示。
8	在变量参数选择对话框中，选择复选框*数据类型*、*组*、*参数*和*外部变量数据*。此外，选择复选框紧凑。其作用是使所有变量数据在一行上显示。 
9	对于变量组选择，在本实例中选择了组 *AlarmLogging1* 和 *AlarmLogging2*。该选择只有在未选择复选框*所有变量组*时才允许。 为应用变量组的选择，必须在变量选择对话框中取消对*所有变量*复选框的选择。 保存*报表编辑器*中所作的设置。

注意：
从 *WinCC 资源管理器*和运行系统中创建新的打印作业并将其激活的过程在*报表编辑器*这章的第一个实例中创建打印作业处进行了详细描述。可以用相同的方式来进行设置。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 已创建的布局无需改变就能使用。

4.3.3 变量记录 CS 的报表(ex_3_chapter_03.pdl)

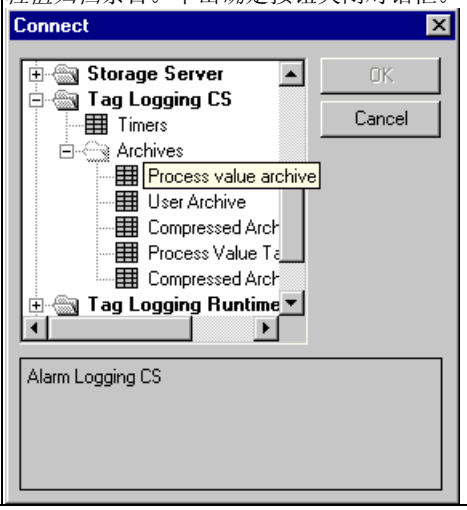
任务定义

在项目中创建的所有过程值归档的组态数据要记录成文档。该文档应包含常规归档数据和各归档变量的组态数据。

概念的实现

在*报表编辑器*中组态一个独立的布局。此布局不基于任何现有的布局。

在报表编辑器中的实现

步骤	过程：在报表编辑器中的实现
1	通过 R *WinCC 资源管理器*中的*页面布局*条目来创建新的布局。 在右边窗口中，新布局将以缺省名称 *NewRPL00.RPL* 添加到现有的页面布局中。通过 R 该名称，将其重命名为 *tlg_cs.rpl*。
2	在*报表编辑器*中打开新布局。 在通过 R 报表页面的空白区域来访问的布局属性对话框内，在*其它*属性处指定*封面*。保留其余的设置。
3	在报表的静态部分，为页眉和页脚组态各种*静态对象*和*系统对象*。 封面设计仅是对个人设计的一种建议。
4	在报表的动态部分，组态*动态对象* → *动态表格*。在本实例中，它是对象 *DynTable1*。 将对象置于报表中之后，将显示*连接*对话框。从*变量记录 CS* 文件夹中，选择*过程值归档*条目。单击*确定*按钮关闭对话框。 

步骤	过程：在报表编辑器中的实现
5	在表格属性对话框的*连接*标签中，有多个选择项可用。 Time range Tag Selection
6	通过 D 其中一个条目，可访问用于数据选择的相应对话框。如果做出了选择，则会以红色复选标记来表示。 在用于选择期望归档的对话框中，选择复选框*所有*归档。在用于选择期望的归档数据的对话框中，所有列出的数据选择都有复选标记。 保存*报表编辑器*中所作的设置。

注意：

创建新的打印作业以及从 *WinCC 资源管理器*与运行系统中激活它的过程在报表编辑器一章的第一个实例中的创建打印作业处进行了详细描述。可以用相同的方法进行设置。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 已创建的布局无需更改就能使用。

4.3.4 在运行系统中打印输出趋势窗口(ex_3_chapter_01a.pdl)

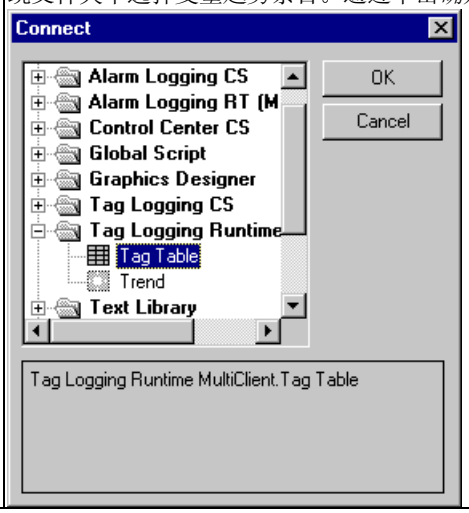
任务定义

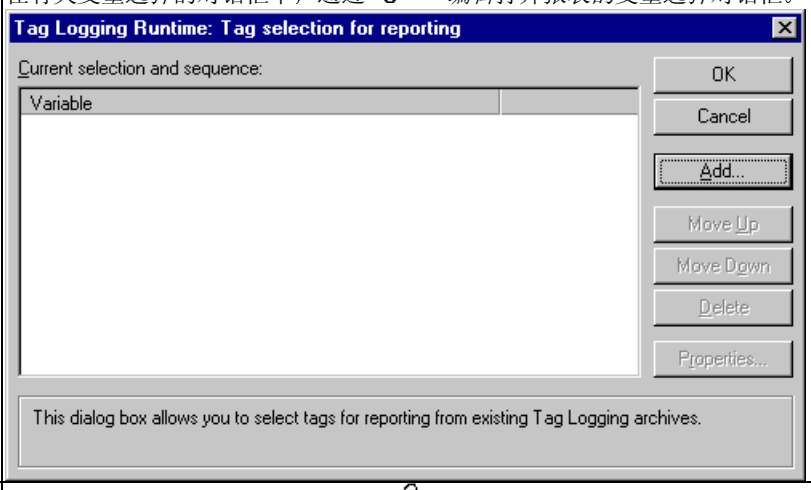
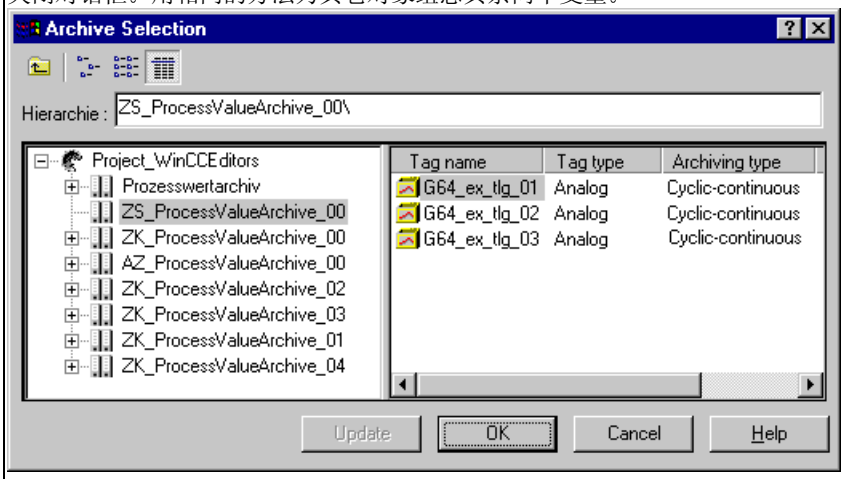
在运行系统中打印输出趋势窗口。可以为要打印的数据设置时间范围。
本实例基于*变量记录*一章中的实例周期选择归档(ex_3_chapter_01a.pdl)。它用于打印输出该实例中显示的表格。

概念的实现

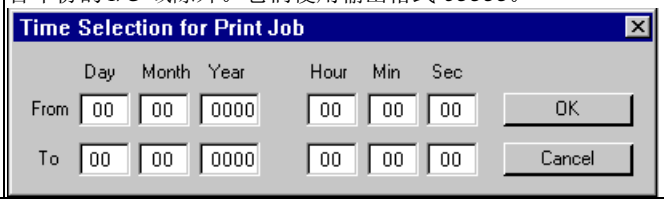
在*报表编辑器*中组态一个独立的布局。时间选择不是在布局中进行，而是通过项目函数在运行系统中进行。该函数将直接在打印作业中进行时间选择。

在报表编辑器中的实现

步骤	过程：在报表编辑器中的实现
1	通过 R *WinCC 资源管理器*中的*页面布局*条目来创建新的布局。 在右边窗口中，新布局被添加到现有布局中。通过 R 将该布局的缺省名称更改为 *tlg_ZS_PA_00.rpl*。
2	在*报表编辑器*中打开新布局。 在报表的静态部分，为页眉和页脚组态各种*静态对象*与*系统对象*。
3	在报表的动态部分，组态三个*动态对象* → *动态图元文件*。在本实例中，它们是对象 *DynMetafile1*、*DynMetafile2* 与 *DynMetafile3*。 把对象置于报表中之后，将会显示*连接*对话框。为这三个对象从*变量记录运行系统文件夹*中选择*变量趋势*条目。通过单击*确定*来关闭对话框。 

步骤	过程：在报表编辑器中的实现
4	在*动态图元文件*属性对话框的*连接*标签中，有多个选择项可以使用。 Time range Tag Selection Format 通过 D 其中一个条目，可访问用于数据选择的相应对话框。如果做出了选择，则会以红色复选标记来表示。 没有进行*时间选择*。 在有关*变量选择*的对话框中，通过 → *编辑*打开*报表的变量选择*对话框。 
5	保存*报表编辑器*中所作的设置。通过 → *添加*来打开归档数据选择对话框。选择归档 *ZS_ProcessValueArchive_00* 和变量 G64_ex_tlg_01。可以用确定按钮来关闭对话框。用相同的方法为其它对象组态其余两个变量。 

创建打印作业

步骤	过程：创建打印作业
1	在 *WinCC 资源管理器*中，通过 R *打印作业*条目来创建一个新的打印作业。 在右边窗口中，该新的打印作业将以缺省名称*打印作业 001* 添加到现有的打印作业中。通过 D 或 R 该打印作业，打开其属性对话框。 在*打印作业*标签中，输入*名称 Printjob_ZS_PA_00*。至于*布局*，则指定先前创建的布局 *tlg_ZS_PA_00.rpl*。 在*选择*标签中，设置打印范围。在*页面范围*域中，选择*全部*选项钮。在*时间范围*域中，选择*绝对*选项钮。不设置指定的时间范围，它将稍后在运行系统中完成。 在*打印机设置*标签中，指定所要使用的打印机。
2	在实例项目中，打印作业预览可以通过一个 *Windows 对象* → *按钮*来激活。它是画面 *ex_3_chapter_01a.pdl* 中的对象*按钮 13*。 在*事件* → *鼠标* → *鼠标动作*处，创建一个启动相应打印作业预览的 *C 动作*。
3	为了进行时间选择，需要一个设置对话框。此对话框组态为一个单独的画面，在本实例中它是画面 *ex_5_window_02.PDL*。 在该画面中，为开始时间条目和结束时间条目各组态 6 个*智能对象* → *I/O 域*。它们是用于开始时间条目的对象 I/O 域 1 至 I/O 域 6，以及用于结束时间条目的对象 I/O 域 7 至 I/O 域 12。 为了缓冲所作的条目，在变量管理器中为每个 *I/O 域*组态一个*无符号的 16 位数*类型的变量。在本实例中，它们是用于开始时间的变量 *U16i_ex_rep_f1* 至 *U16i_ex_rep_f6*，以及用于结束时间的变量 *U16i_ex_rep_t1* 至 *U16i_ex_rep_t6*。 对于每个 *I/O 域*，在*属性* → *输出/输入* → *输出值*处创建与其相应变量的*变量连接*。 在*属性* → *输出/输入* → *输出格式*处，为所有 *I/O 域*将格式设置为 *099*，将包含年份的 *I/O 域*除外。它们使用输出格式 *09999*。 

步骤	过程：创建打印作业
4	在画面 *ex_5_window_02.PDL* 的*事件* → *其它* → *打开画面*处，创建一个为设置时间的变量提供预定义时间值的 *C 动作*。将当前系统时间设定为结束时间，将当前系统时间减去一分钟设定为开始时间。
5	在画面 *ex_5_window_02.PDL* 中，组态两个 *Windows 对象* → *按钮*。在本实例中，它们是对象*按钮 1* 与*按钮 2*。 *按钮 2* 是取消按钮。在*事件* → *鼠标* → *鼠标动作*处创建一个*直接连接*，用于将*常数 0* 转换为当前窗口的显示。 *按钮 1* 是确定按钮。它也在*事件* → *鼠标* → *鼠标动作*处接收一个*直接连接*，用于关闭窗口。在*事件* → *鼠标* → *按下左键*处，组态一个 *C 动作*。该动作调用一个预先创建的用于为某个打印作业选择时间的项目函数。打印作业的名称从一个 *16 位字符集文本变量*类型的变量中读取，该变量必须在*变量管理器*中创建。在本实例中，它是变量 *T16i_ex_rep_00*。
6	为了在画面 *ex_3_chapter_01a.PDL* 中显示已创建的画面，必须创建一个*智能对象* → *画面窗口*。在本实例中，它是对象*画面窗口 1*。在*属性* → *其它* → *画面名称*处，设置先前创建的画面 *ex_5_window_02.PDL*。将*显示*属性设置为*否*。
7	为了使*画面窗口*可见，需要一个附加的 *Windows 对象* → *按钮*。在本实例中，它是对象*按钮 12*。该对象在*事件* → *鼠标* → *鼠标动作*处有一个 *C 动作*，用于把要处理的打印作业的名称写入变量 *T16i_ex_rep_00* 中，并使对象*画面窗口 1* 可见。

确定按钮的 C 动作

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszF
{

ModifyPrintJob(TimeFrom(),
        TimeTo(),
        GetTagChar("T16i_ex_rep_00"));

}
```

- 调用项目函数 *ModifyPrintJob*。该项目函数需要两个 SYSTEMTIME 结构形式的时间值作为传送参数。它们由两个项目函数根据存储时间值的内部变量来确定*(TimeFrom()与 TimeTo())*。此外，还需要进行处理的打印作业的名称。该名称存储在变量 *T16i_ex_rep_00* 中。

项目函数 ModifyPrintJob

```
BOOL ModifyPrintJob(SYSTEMTIME st1,SYSTEMTIME st2,char jobname[200])
{
BOOL            fRet;
PCMN_ERROR      pError;
HPROPERTIES     hProp;
LPVOID  ptr1,ptr2;
DWORD   typ;
DWORD   dwVal;
char            propname1[200],propname2[200];
TCHAR           g_szProj[MAX_PATH+1];

typ = VT_DATE;
strcpy( propname1, "ABSOLUTESELECTIONFROM");
strcpy( propname2, "ABSOLUTESELECTIONTO");
ptr1 = (LPVOID)&st1;
ptr2 = (LPVOID)&st2;

//-----get project path
if( !DMGetRuntimeProject( g_szProj, MAX_PATH, pError)) {
        printf("Error DMGetRuntimeProject(...)\r\n");
        return FALSE;
        }
//-----create property handle
hProp = RPJCreatePropertyHandle ( g_szProj, pError );
if( !hProp) {
        printf("Error RPJCreatePropertyHandle(...)\r\n");
        return FALSE;
        }
//-----get job properties
if ( !RPJGetJobProps ( hProp, jobname, pError )) {
        printf("Error RPJGetProps(...)\r\n");
        RPJDeletePropertyHandle ( hProp, pError);
        return FALSE;
        }
//-----set property
if ( !RPJSetProperty ( hProp, propname1, ptr1,
                (VARTYPE) typ, 200, pError )) {
        printf("Error RPJSetProperty(...)\r\n");
        RPJDeletePropertyHandle ( hProp, pError);
        return FALSE;
        }
//-----save job properties
if ( !RPJSetJobProps ( hProp, jobname, pError)) {
        printf("Error RPJSetProps(...)\r\n");
        RPJDeletePropertyHandle ( hProp, pError);
        return FALSE;
        }
//-----get job properties
if ( !RPJGetJobProps ( hProp, jobname, pError )) {
        printf("Error RPJGetProps(...)\r\n");
        RPJDeletePropertyHandle ( hProp, pError);
        return FALSE;
        }
//-----set property
if ( !RPJSetProperty ( hProp, propname2, ptr2,
                (VARTYPE) typ, 200, pError )) {
        printf("Error RPJSetProperty(...)\r\n");
        RPJDeletePropertyHandle ( hProp, pError);
        return FALSE;
        }
//-----save job properties
if ( !RPJSetJobProps ( hProp, jobname, pError)) {
        printf("Error RPJSetProps(...)\r\n");
        RPJDeletePropertyHandle ( hProp, pError);
        return FALSE;
        }
//-----delete property handle
fRet = RPJDeletePropertyHandle ( hProp, pError);
return TRUE;
}
```

- 作为 *st1* 与 *st2* 传送参数，该函数会接收以 SYSTEMTIME 结构形式设定的开始和结束时间。
- 用 *DMGetRuntimeProject* 函数确定项目路径。
- 设置并保存开始时间。这是属性 *ABSOLUESELECTIONFROM*。
- 设置并保存结束时间。这是属性 *ABSOLUESELECTIONTO*。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 在用于打印趋势图的布局中，必须修改要制成报表的归档以及要打印的归档变量。
- 时间选择对话框不用修改就可以应用。为了进行操作，需要项目函数 *ModifyPrintJob*、*TimeFrom* 与 *TimeTo*。用于缓冲时间值的变量必须使用相同的名称来创建。否则，必须修改项目函数 *TimeFrom* 与 *TimeTo*。为了使用用于多个打印作业的对话框，建议创建用于存储打印作业名称的文本变量。

4.3.5 在运行系统中打印输出表格(ex_3_chapter_01c.pdl)

任务定义

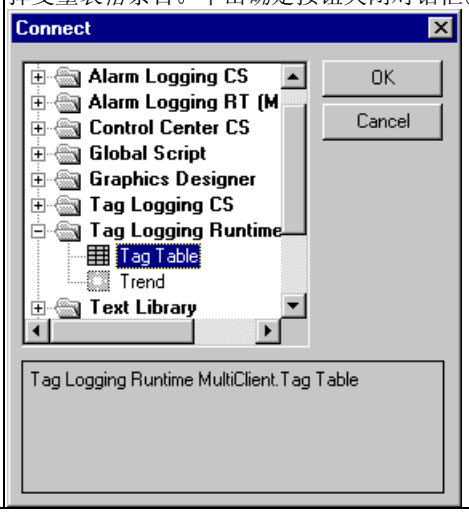
在运行系统中打印输出表格。可以为要打印的数据设置时间范围。
本实例基于*变量记录*一章中的实例自定义的表格布局(ex_3_chapter_01c.pdl)。
它用于打印输出该实例中显示的表格。

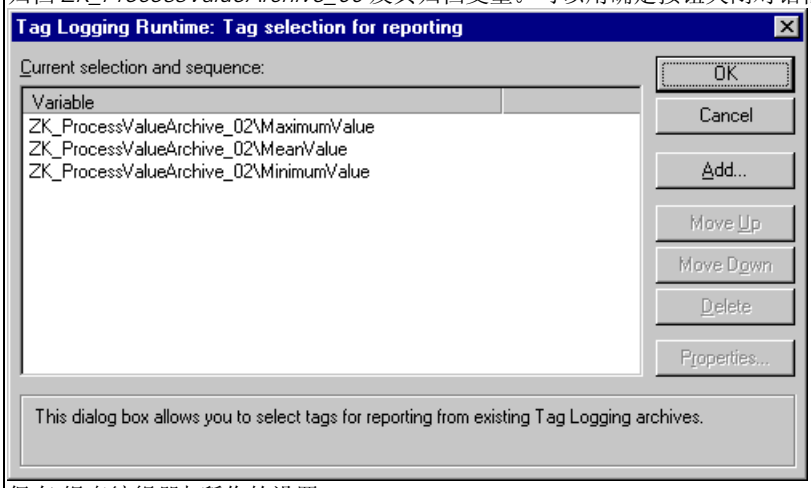
概念的实现

在*报表编辑器*中组态一个新的页面布局。时间选择并不在布局中进行，而是在运行系统中通过项目函数来进行。该函数将直接在打印作业中执行时间选择。
在运行系统中进行时间选择的步骤在先前实例的*创建打印作业*处进行了详细描述。

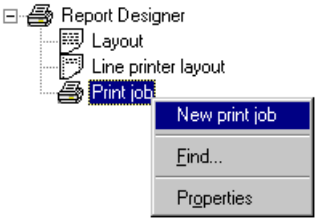
在报表编辑器中的实现

步骤	过程：在报表编辑器中的实现
1	通过 R *WinCC 资源管理器*中的*页面布局*条目来创建新的布局。 在右边窗口中，新布局被添加到现有布局中。通过 R 将布局的缺省名称更改为 *tlg_ZK_PA_02.rpl*。
2	在*报表编辑器*中打开新布局。 在报表的静态部分，为页眉和页脚组态各种*静态对象*与*系统对象*。

步骤	过程：在报表编辑器中的实现
3	在报表的动态部分，组态*动态对象* → *动态表格*。在本实例中，它是对象*DynTable1*。 将对象置于报表中之后，将显示*连接*对话框。从*变量记录运行*系统文件夹中，选择*变量表格*条目。单击*确定*按钮关闭对话框。 

步骤	过程：在报表编辑器中的实现
4	在*动态图元文件*属性对话框的*连接*标签中，有多个选择项可以使用。 Time range Tag Selection 通过D 其中一个条目，可访问用于数据选择的相应对话框。如果做出了选择，则会以红色复选标记来表示。 不指定*时间范围*。 通过 → *编辑* → *添加来归档用于归档选择的对话框。在该对话框中*，选择归档 *ZK_ProcessValueArchive_00* 及其归档变量。可以用确定按钮关闭对话框。  保存*报表编辑器*中所作的设置。

创建打印作业

步骤	过程：创建打印作业
1	在 *WinCC 资源管理器*中，通过 R 来创建新的打印作业。  在右边窗口中，新的打印作业将以缺省名称*打印作业 001* 添加到现有的打印作业中。通过 D 或 R 该打印作业，打开其属性对话框。 在*打印作业*标签中，输入*名称 Printjob_ZK_PA_02*。为*布局*指定先前已创建的布局 *tlg_ZK_PA_02.rpl*。 在*选择*标签中，设置打印范围。在*页面范围*域中，选择*全部*单选按钮。在*时间范围*域中，选择*绝对*单选按钮。不设置指定的时间范围，它将稍后在运行系统中完成。 在*打印机设置*标签中，指定所要使用的打印机。
2	在运行系统中进行时间选择的步骤在先前实例的创建打印作业处进行了详细描述。 对于如下所示的按钮，必须修改*事件* → *鼠标* → *鼠标动作*处的 *C 动作*。 必须为文本变量 *T16i_ex_rep_00* 提供刚创建的打印作业的名称。 对于启动打印预览的按钮，也必须更改*事件* → *鼠标* → *鼠标动作*处 *C 动作*的函数调用中的打印作业名称。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 修改归档选择之后，可以直接使用设计的布局。

4.3.6 消息顺序报表(ex_3_chapter_02b.pdl)

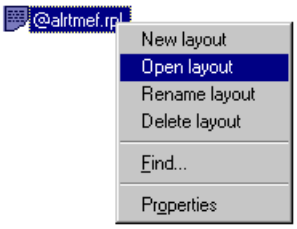
任务定义

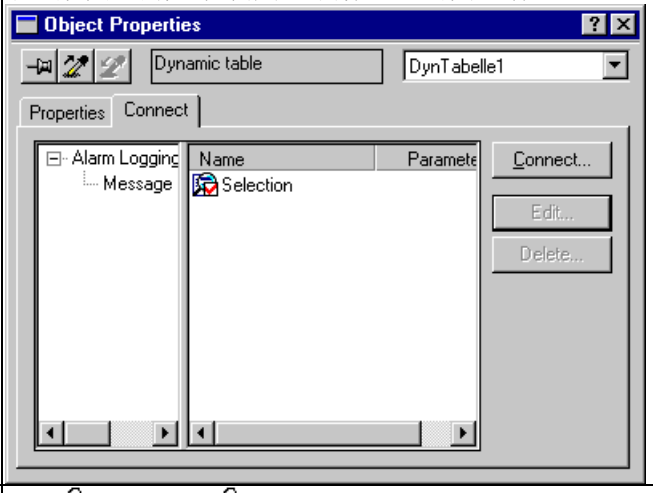
将创建消息顺序报表。在完整填写了布局页面后，消息顺序报表将自动打印。
本实例基于*报警记录*一章中的实例消息窗口(ex_3_chapter_02b.pdl)。在该实例中，在所使用的消息窗口中已包含了用于报表功能的工具栏按钮，并且消息顺序报表已经被激活。

概念的实现

可以使用一个系统布局和符合规定要求的系统打印作业。这是*@alrtmef.rpl* 布局和*@Report Alarm Logging RT Message sequence* 打印作业。复制并修改该布局以满足用户需要。至于打印作业，必须使用系统打印作业，只有系统打印作业所使用的布局才可改变。

在报表编辑器中的实现

步骤	过程：在报表编辑器中的实现
1	在 *WinCC 资源管理器*中通过 R 系统布局*@alrtmef.rpl* 的名称来将其打开。  通过*文件* → *另存为...*菜单将系统布局以另一个不同的名称保存。在本实例中使用名称 *alg_mef.rpl*。
2	报表的静态部分包含页眉和页脚。 可以修改静态部分的元素以满足用户需求。

步骤	过程：在报表编辑器中的实现
3	报表的动态部分包括*动态对象* → *动态表格*。在本实例中是对象 *DynTable1*。 打开 *DynTable1* 对象的属性对话框，并选择*连接*标签。该表格已经连接到*报警记录运行系统的消息顺序报表*。同样也已进行了选择。 
4	通过 → *编辑*或 D *选择*条目，打开用于消息块选择的对话框。在该对话框中，已选择了系统块*日期*、*时间*、*编号*和*报警中的循环*。在本实例中，通过下面所显示的按钮选择所有其余的消息块。 通过单击*确定*来关闭对话框。保存该布局。 ->>

打印作业的修改

步骤	过程：打印作业的修改
1	系统打印作业*@Report Alarm Logging RT Message sequence* 的打开可以通过 D 或在 *WinCC 资源管理器*中 R 它的名称实现。
2	在*打印作业*标签中，指定刚创建的*布局 alrtmef.rpl*。此外，在*打印机设置*标签中设置所期望的打印机。无需做其它修改。单击确定按钮关闭对话框。
3	消息顺序报表必须在*报警记录*编辑器中激活。为此，打开该编辑器。通过 R *报表*条目，打开对话框*分配报表参数*。 在该对话框中，选择用于消息顺序报表的*复选框*。
4	如果用户没有手动激活打印作业，则一旦布局页面已满，将会自动打印消息顺序报表。 为了在任何时侯都允许用户启动消息顺序报表，必须在组态消息窗口模板时包括工具栏按钮。该按钮功能名称为*报表功能*。在实例项目中，该功能已在 *ex_3_chapter_02b.pdl* 画面中所使用的 *MessageWindow_04* 模板中选择。 保存在*报警记录*中所作的设置。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 在已创建的布局中，必须为消息顺序报表修改所期望的消息块。

4.3.7 行式打印机上的消息顺序报表

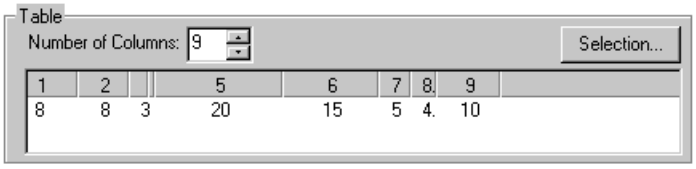
任务定义

要设计适合行式打印机输出的消息顺序报表。如果要生成报表的消息到达，则它将自动打印。

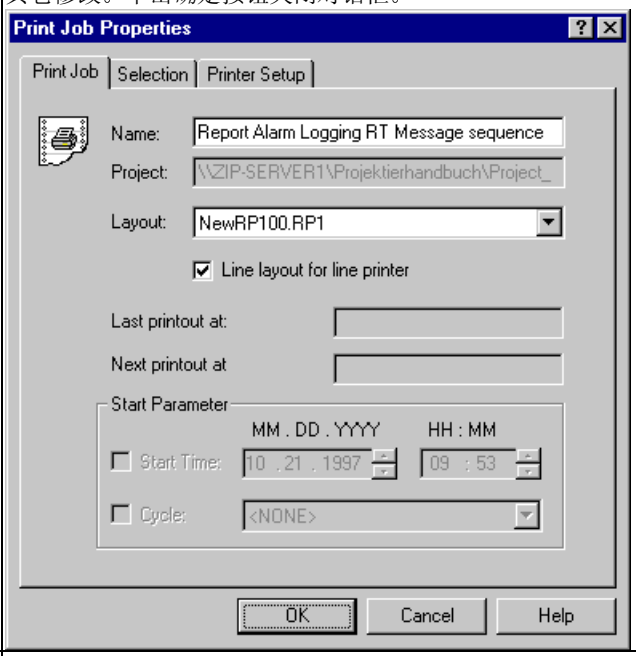
概念的实现

创建行式布局。于是将为系统打印作业*@Report Alarm Logging RT Message sequence* 指定该布局。

创建行式布局

步骤	过程：创建行式布局
1	通过 R *WinCC 资源管理器*中相应的条目来创建新的行式布局。
2	创建一个带有缺省名称 *NewRP100.RP1* 的新的行式布局。保留该名称用于实例项目。用 D 右边窗口中新行式布局的名称，打开*行式布局编辑器*。 在本编辑器中可以进行关于页边距、页眉、页脚等的常规设置。 在*表格域*中，通过 → *选择*打开消息顺序报表中用于选择所期望的消息块的对话框。在本实例中，选择所有可用的消息块。
3	列数及其宽度将自动与所选择的消息块和顺序相匹配。  保存所作的设置并关闭行式布局编辑器。

打印作业的修改

步骤	过程：打印作业的修改
1	可以通过 D 或 R *WinCC 资源管理器*中系统打印作业*@Report Alarm Logging RT Message sequence* 的名称来将其打开。
2	在*打印作业*标签中，选择用于*行式打印机的行式布局*复选框并指定刚创建的*布局 NewRP100.RP1*。此外，在*打印机设置*标签中设置所期望的行式打印机。无需做其它修改。单击*确定*按钮关闭对话框。 
3	消息顺序报表必须在*报警记录*编辑器中激活。为此，打开该编辑器。通过 R *报表*条目，打开对话框*分配报表参数*。在该对话框中，选择用于消息顺序报表的*复选框*。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 要打印的期望的页面设置和消息块必须在行式布局编辑器中进行修改。

4.3.8 消息归档报表(ex_3_chapter_02c.pdl)

任务定义

创建消息归档报表。打印作业将由用户通过按钮激活。
本实例基于*报警记录*一章中的实例消息归档(ex_3_chapter_02c.pdl)。在该实例中，已经包括了用于短期(周期性)归档窗口的报表功能的工具栏按钮，并已激活了消息顺序报表。

概念的实现

可以使用一个系统布局和符合规定要求的系统打印作业。它们是布局*@alrtuma.rpl* 和打印作业*@Report Alarm Logging RT Revolving archive*。复制并修改该布局以满足用户需要。至于打印作业，必须使用系统打印作业，只有系统打印作业所使用的布局才可改变。

在报表编辑器中的实现

步骤	过程：在报表编辑器中的实现
1	在 *WinCC 资源管理器*中通过 R 右边窗口中的系统布局*@alrtuma.rpl* 的名称来将其打开。 通过*文件* → *另存为...*菜单将系统布局保存为另一个不同的名称。在本实例中，使用名称 *alg_uma*。
2	报表的静态部分包含页眉和页脚。 可以修改静态部分的元素以满足用户需求。
3	报表的动态部分包括*动态对象* → *动态表格*。在本实例中是 *DynTable1* 对象。 打开 *DynTable1* 对象的属性对话框，并选择*连接*标签。该表格已经连接到*报警记录运行系统的短期归档报表*上。也已进行了选择。
4	通过 → *编辑*按钮或 D 选择条目，打开用于消息块选择的对话框。在该对话框中，已选择了系统块*日期*、*时间*和*编号*。在本实例中，通过下面所显示的按钮选择所有其余的消息块。 通过单击*确定*来关闭对话框。保存该布局。 ->>

打印作业的修改

步骤	过程：打印作业的修改
1	可以通过 D 或 R *WinCC 资源管理器*中系统打印作业*@Report Alarm Logging RT Revolving archive* 的名称来将其打开。
2	在*打印作业*标签中，指定刚创建的*布局 alrtuma.rpl*。此外，在*打印机设置*标签中设置所期望的打印机。无需做其它修改。单击*确定*按钮关闭对话框。
3	归档报表必须在*报警记录*编辑器中激活。为此，打开该编辑器。通过 R *报表*条目，打开对话框*分配报表参数*。 在该对话框中，选择用于归档报表的*复选框*。保存在*报警记录*中所作的设置。
4	为了在任何时侯都允许用户启动消息顺序报表，必须在组态消息窗口模板时包括工具栏按钮。如果使用了用户定义的工具栏，必须使用相应的标准函数模拟该按钮的按下。它就是标准函数 *ACX_OnBtnPrint()*。在实例项目中，该按钮已经组态在画面 *ex_3_chapter_02c.pdl* 中。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 在已创建的布局中，必须为归档报表修改所期望的消息块。

4.4 与 EXCEL 的 OLE 通讯

在运行系统中，有关该主题的实例可通过使用选择如上所示的按钮来访问。这些实例组态在画面 *ex_3_chapter_04.pdl* 和 Excel 文件夹 *OLE_Communication.xls* 中。

4.4.1 读和写变量数值(ex_3_chapter_04.pdl)

任务定义

各种类型的内部变量值将写入 Excel 电子表格中。在电子表格的第二列，为这些变量输入设定值。然后将这些数值写入 WinCC 项目。

概念的实现

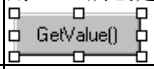
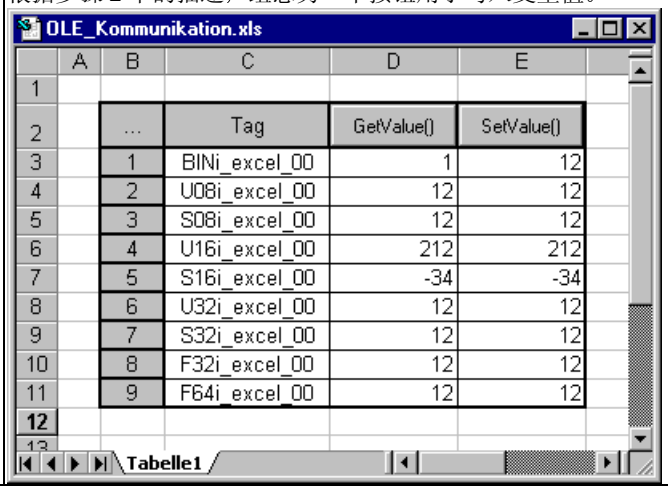
在画面中，为每个变量组态一个 I/O 域，将显示变量值并可为它分配一个数值。

在 Excel（版本 8.0）中创建电子表格。在电子表格的其中一列，输入要读取和写入的变量名称。将两个按钮添加到电子表格中。将宏指令分配给这些按钮，读取要处理的变量名称，并且读取相关的变量值或将设定值分配给这些变量。

在 WinCC 项目中的实现

步骤	过程：在 WinCC 项目中的实现
1	在变量管理器中，创建多个不同类型的变量。在本实例中使用下列变量： Name / Type BINi_excel_00 — Binary Tag F32i_excel_00 — Floating-point number 32-bit IEEE 754 F64i_excel_00 — Floating-point number 64-bit IEEE 754 S16i_excel_00 — Signed 16-bit value S32i_excel_00 — Signed 32-bit value S08i_excel_00 — Signed 8-bit value U16i_excel_00 — Unsigned 16-bit value U32i_excel_00 — Unsigned 32-bit value U08i_excel_00 — Unsigned 8-bit value
2	在画面中，为每个变量创建一个*智能对象* → *I/O 域*。在*属性* → *输出/输入* → *输出值*处，为每个变量创建一个*变量连接*。

在 Excel (版本 8.0)中的实现

步骤	过程：在 Excel 中的实现
1	创建一个新的 Excel 文件夹。在本实例中，将文件夹命名为 *OLE_Communication.xls*。 将要处理的变量的名称填入电子表格的其中一列。
2	组态一个用于读取变量值的按钮。 为此，通过*查看* → *工具栏*菜单激活*控制工具箱*工具栏，如果它还尚未激活。选择命令按钮元素并且将它放进表格中。
3	可在通过如下所示的按钮所访问到的对话框中设置该控制元素的属性。本实例使用的对象名称为 *GetValue*，其标题为 *GetValue()*。
4	用 D 所创建的按钮，将打开 Visual Basic 编辑器并且可以输入要执行的过程。 
5	根据步骤 2 中的描述，组态另一个按钮用于写入变量值。 

...	Tag	GetValue()	SetValue()
1	BINi_excel_00	1	12
2	U08i_excel_00	12	12
3	S08i_excel_00	12	12
4	U16i_excel_00	212	212
5	S16i_excel_00	-34	-34
6	U32i_excel_00	12	12
7	S32i_excel_00	12	12
8	F32i_excel_00	12	12
9	F64i_excel_00	12	12

读取变量值的过程

```
Rem Read Tag Values in WinCC-Project

Private Sub GetValue_Click()

    Dim mcp As Object
    Dim var As String
    Dim value As Variant
    Dim cell As Variant
    Dim i As Integer

    Set mcp = CreateObject("WinCC-Runtime-Project")

    Cell = "C3"
    i = 1

    Do While Not Range(cell) = ""
        var = Range(cell)
        value = mcp.GetValue(var)
        Range("D" & 2 + i).value = value
        cell = "C" & 3 + i
        i = i + 1
    Loop

End Sub
```

- 存储在 *mcp* 变量中的 WinCC 对象的生成。
- 在循环中，读取包含了要处理的变量名称的这一列中的各个单元格的内容。在 WinCC 项目中，用函数 *GetValue()*读取变量并将其数值写入相邻列。重复该循环过程直至到达第一个空白的单元格。

写入变量值的过程

```
Rem Set Tag Values in WinCC-Project

Private Sub SetValue_Click()

    Dim mcp As Object
    Dim var As String
    Dim value As Variant
    Dim cell As Variant
    Dim i As Integer
    Dim bRet As Integer

    Set mcp = CreateObject("WinCC-Runtime-Project")

    Cell = "C3"
    i = 1

    Do While Not Range(cell) = ""
        var = Range(cell)
        value = Range("E" & 2 + i).value
        bRet = mcp.SetValue(var, value)
        cell = "C" & 3 + i
        i = i + 1
    Loop

End Sub
```

- 存储在 mcp 变量中的 WinCC 对象的生成。
- 在循环中，读取包含了要处理的变量名称的这一列中的各个单元格的内容。此外，还读取包含将要设置的变量值的一列。用函数 *SetValue()*将这些变量写入 WinCC 项目。重复该循环过程直至到达第一个空白的单元格。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 使用 GetValue()和 SetValue()函数在 WinCC 和 Excel 之间进行数据交换。可根据所期望的要求在 Excel 中处理该数据。

4.5 实例中的附加组态

本章描述某些画面中使用的附加元素。因为它们与现有主题并没有直接关联，所以它们将在相关的实例中进行详尽的描述。本章将完成实例项目的描述。

4.5.1 画面索引

任务定义

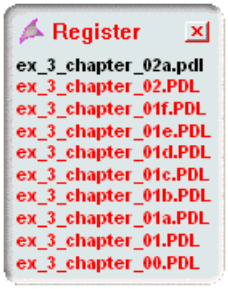
对项目中最后 10 幅所选画面的顺序进行存储。通过返回按钮，可以相反的顺序选择这些画面。通过下一个按钮，可按正常顺序往下移动到当前画面。
在一幅单独的画面中，索引中的所有画面均将按正确的顺序进行显示。从该画面中，也可直接选择画面。

概念的实现

画面顺序将存储在项目函数的 10 个静态 C 变量中。每执行一次画面切换，就调用该项目函数一次。它通过下一个和返回按钮来控制画面选择，同时，也可对直接画面选择进行控制。

在图形编辑器中的实现

步骤	过程：在图形编辑器中的实现
1	使用 2 个*二进制变量*类型的变量。它们是变量 *BINi_ex_org_00* 与 *BINi_ex_org_01*。这些变量使下一个与返回按钮成为可操作的按钮。 此外，还使用了一个*无符号 16 位数*类型的变量。在实例中，这就是变量 *U16i_ex_org_00*。该变量将当前位置存储在画面索引中。 此外，还使用了一个 *16 位字符集文本变量*类型的变量。在实例中，这就是变量 *T16x_ex_org_00*。该变量记录了当前画面名称。
2	可利用一个用于控制画面索引的项目函数。这就是函数 *CreatePictureSequence*。每执行一次画面切换，就调用该函数一次。完成该操作可通过 *ex_0_startpicture_00.pdl* 画面中对象*工作空间*的*事件* → *属性主题* → *其它* → 画面名称处的 C 动作来进行。每次调用该函数时，新画面名称均将存储在索引中，且现有名称将向后移动一个位置。

步骤	过程：在图形编辑器中的实现
3	在 *ex_2_keyboard_00.PDL* 键盘画面中，组态了 2 个用于向后和向前滚动的控制单元。 如果按下这些对象中的一个，则也要调用项目函数 *CreatePictureSequence*。然后，该函数将完成画面切换。通过两个*智能对象* → *图形对象*，可使这两个控制单元成为非操作单元。
4	通过下列按钮，可调用一个用于显示当前画面索引的画面。在实例中，这就是 *ex_9_register_00.PDL* 画面。
5	在该画面中，组态了 10 个*标准对象* → *静态文本*。缺省文本将从所有对象中删除。在画面对象的*属性* → *几何形状* → *画面宽度*处，创建一个 *C 动作*，用于调用项目函数 *CreatePictureSequence*。该动作将提供所存储的画面名称作为*静态文本*。一旦变量 *T16x_ex_org_00* 发生变化，就将触发该 *C 动作*。这将导致一旦画面切换则显示将更新。 对各个*静态文本*，在*事件* → *鼠标* → *按下左键*处均创建一个 *C 动作*，用于调用项目函数，并完成将画面切换为所显示画面的操作。当前所选画面用高亮度显示。完成该操作可通过用于各个静态文本的*属性* → *颜色* → *字体颜色*处的*动态对话框*来进行。 
6	通过某个参数，项目函数可识别出它是从何处调用的。对于该参数，已在 *APDEFAP.H* 文件中定义了几个不同的常数。该文件位于项目文件夹的*库*子文件夹内。 定义的常数如下： • #define REG_INSERTPICTURE 0 • #define REG_BACK 1 • #define REG_FORWARD 2 • #define REG_DIRECT 3 • #define REG_SHOWREGISTER 4

用于控制画面索引的项目函数

```
#include "APDEFAP.H"
#define MAX_REG 10
void CreatePictureSequence(char* PicName,int nFlag,int nPos)
{
static char PictureName[MAX_REG][40] = {"","","","","","","","","","","}};
int i;
static int pos = 0;
static int st = 0;
static int biz = 0;

if (nFlag==REG_INSERTPICTURE){
 if (st == 0 ){
  pos = 0;
  if (biz < MAX_REG) biz++;
  for ( i=(MAX_REG-1) ; i>0 ; i-- ){
   strcpy(PictureName[i],PictureName[i-1]);
   }
  strcpy(PictureName[0],PicName);
  }
 else st=0;
 }
if (nFlag==REG_BACK){
 pos++;
 if ( pos > (MAX_REG-1) ) pos=(MAX_REG-1);
 st = 1;
 SetPictureName("ex_0_startpicture_00.PDL",
                "workspace",PictureName[pos]);
 }
if (nFlag==REG_FORWARD){
 pos--;
 if ( pos < 0 ) pos=0;
 st = 1;
 SetPictureName("ex_0_startpicture_00.PDL",
                "workspace",PictureName[pos]);
 }
if (nFlag==REG_SHOWREGISTER){
 SetText("ex_9_register_00.PDL",
         "Static Text1",PictureName[0]);
 SetText("ex_9_register_00.PDL",
         "Static Text2",PictureName[1]);
 SetText("ex_9_register_00.PDL",
         "Static Text3",PictureName[2]);
 SetText("ex_9_register_00.PDL",
         "Static Text4",PictureName[3]);
 SetText("ex_9_register_00.PDL",
         "Static Text5",PictureName[4]);
 SetText("ex_9_register_00.PDL",
         "Static Text6",PictureName[5]);
 SetText("ex_9_register_00.PDL",
         "Static Text7",PictureName[6]);
 SetText("ex_9_register_00.PDL",
         "Static Text8",PictureName[7]);
 SetText("ex_9_register_00.PDL",
         "Static Text9",PictureName[8]);
 SetText("ex_9_register_00.PDL",
         "Static Text10",PictureName[9]);
 }
if (nFlag==REG_DIRECT){
 st=1;
 pos=nPos;
 }
if ((nFlag!=REG_SHOWREGISTER) && (nFlag!=REG_DIRECT)){
 if (pos<(biz-1))      SetTagBit("BINi_ex_org_00",FALSE);
 else                  SetTagBit("BINi_ex_org_00",TRUE);

 if (pos>0)            SetTagBit("BINi_ex_org_01",FALSE);
 else                  SetTagBit("BINi_ex_org_01",TRUE);
 }
SetTagWord("U16i_ex_org_00",(WORD)pos);
}
```

- 如果传送参数 *nFlag* 具有值 *REG_INSERTPICTURE*，则表示已通过一个 *C 动作*对函数进行了调用，该动作位于*事件* → *属性主题* → *其它* → *画面名称*处，属于 *ex_0_startpicture_00.pdl* 画面中的*工作空间*对象。*工作空间*对象是种可在其中显示所有实例画面的画面窗口。如果画面切换没有输入到索引中，则在先前的函数调用期间，必须将变量 st 设置为 1。索引本身由具有 10 个文本变量的静态数组组成。
- 如果传送参数 *nFlag* 的值为 *REG_BACK*，则表示已按下了返回按钮。画面切换由函数本身执行，但不输入到索引中。
- 如果传送参数 *nFlag* 的值为 *REG_FORWARD*，则表示已按下了下一个按钮。画面切换由函数本身执行，但不输入到索引中。
- 如果传送参数 *nFlag* 的值为 *REG_SHOWREGISTER*，则对 *ex_9_register_00.pdl* 画面中的所有*静态文本*进行更新。如果已选择了索引画面，或在索引画面打开时完成了画面的切换，就会发生这种情况。
- 如果传送参数 *nFlag* 的值为 *REG_DIRECT*，则表示已通过静态文本完成了一个直接画面选择。完成画面切换可通过*静态文本*的 *C 动作*来进行，然而，这种切换将不输入到索引中。

常规应用的注意事项

在进行常规应用之前，必须完成下述修改：

- 可直接应用所作的组态。为了进行操作，必须创建 5 个要用的变量，必须应用项目函数并且必须插入控件单元。
- 如果不需要直接画面选择和画面索引显示，则段 *REG_DIRECT* 与 *REG_SHOWREGISTER* 可从项目函数中省略。
- 为了修改所存储画面的数目，必须对项目函数中的用于最大画面数目的 *MAX_REG* 定义进行修改。

4.5.2 索引

实例项目的索引可以通过用选择如上所示的按钮来进行访问。

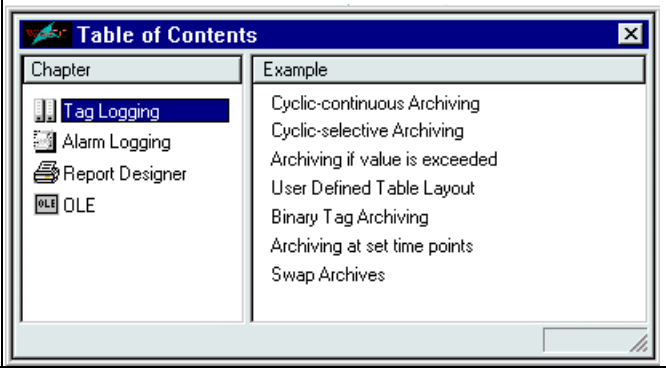
任务定义

通过一个对话框来显示项目的索引。在一个窗口中，将显示主要的章节。在另一个窗口中，将显示与所选章节相关的实例。
也可以直接选择一个实例。这种选择要通过双击来进行。

概念的实现

索引对话框的选择可通过总览栏上的按钮来完成。该对话框用一个画面窗口来显示。对话框包含一个附加画面窗口，用于根据所选章节来显示画面。

在图形编辑器中的实现

步骤	过程：在图形编辑器中的实现
1	使用两个*无符号 16 位数*类型的变量。在实例中，它们是变量 *U16i_ex_con_00* 与 *U16i_ex_con_01*。这些变量存储当前所选择的章节编号和实例编号。
2	可使用一个在其中已创建对话框布局的画面。它就是画面 *ex_9_register_01.PDL*。为每章组态一个*标准对象* → *静态文本*以及一个*智能对象* → *图形对象*。 最初，在选择画面时没有选择任何章节，即变量 *U16i_ex_con_00* 的值为 0。如果用选择一个*静态文本*，则相应的章节编号将被写入到该变量中。使用各种不同的*动态对话框*，可更改所选*静态文本*的颜色设置。 

步骤	过程：在图形编辑器中的实现
3	为每章组态一个单独的画面，它将根据所选章节的编号显示在一个*智能对象* → *画面窗口*中。如果没有选择任何章节，则画面窗口不会显示。 为各章节中的每个实例，组态一个*标准对象* → *静态文本*。如果选择一个章节，则最初将不会显示任何实例。如果用选择一个*静态文本*，则相应的实例编号将被写入到变量 *U16i_ex_con_01* 中。使用各种不同的*动态对话框*，可更改所选*静态文本*的颜色设置。
4	为了通过双击实现画面选择，创建了三个外部 C 变量(如下所示)。它们均在项目函数 *CreateExternal* 中创建。该函数在项目启动时执行一次。 • externBOOLbPress1,bPress2 • extern int nButtonID 通过在画面的*属性* → *几何结构* → *画面宽度*处的 *C 动作*，可按 500 毫秒的周期对是否执行了双击进行查询。如果执行了双击，则根据变量 *nButtonID* 执行一个动作。

实例文本的 C 动作

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP
{
extern BOOL bPress1,bPress2;
static BOOL bToggle = FALSE;
extern int nButtonID;

nButtonID=1;

if (bToggle) bPress1=TRUE;
else bPress2=TRUE;

bToggle=!bToggle;

SetTagWord("U16i_ex_cont_01",(WORD)nButtonID);

}
```

- 为外部 C 变量提供*静态文本*的标识号。该标识号用于确定所要执行的动作。
- 对于每个鼠标动作，*bPress1* 与 *bPress2* 的其中之一将设置为 TRUE。

确定双击的 C 动作

```
#include "apdefap.h"
 long _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty
{
extern BOOL bPress1,bPress2;
extern int nButtonID;

if ((bPress1)&&(bPress2))
 switch(nButtonID){
 case1:SetPictureName("ex_0_startpicture_00.PDL",
                      "workspace","ex_3_chapter_01.PDL");
       break;
 case2:SetPictureName("ex_0_startpicture_00.PDL",
                      "workspace","ex_3_chapter_01a.PDL");
       break;
 case3:SetPictureName("ex_0_startpicture_00.PDL",
                      "workspace","ex_3_chapter_01b.PDL");
       break;
 case4:SetPictureName("ex_0_startpicture_00.PDL",
                      "workspace","ex_3_chapter_01c.PDL");
       break;
 case5:SetPictureName("ex_0_startpicture_00.PDL",
                      "workspace","ex_3_chapter_01d.PDL");
       break;
 case6:SetPictureName("ex_0_startpicture_00.PDL",
                      "workspace","ex_3_chapter_01e.PDL");
       break;
 case7:SetPictureName("ex_0_startpicture_00.PDL",
                     "workspace","ex_3_chapter_01f.PDL");
       break;
 case8:SetPictureName("ex_0_startpicture_00.PDL",
                      "workspace","ex_3_chapter_01g.PDL");
       break;
 case9:SetPictureName("ex_0_startpicture_00.PDL",
                      "workspace","ex_3_chapter_01i.PDL");
       break;
 case10:SetPictureName("ex_0_startpicture_00.PDL",
                      "workspace","ex_3_chapter_01j.PDL");
        break;
 default:break;
 }

bPress1=FALSE;
bPress2=FALSE;

return 242;
}
```

- 对两个外部 C 变量都进行查询。如果两个都具有状态 *TRUE*，则在过去的 500 毫秒内已双击了*静态文本*。这就是说，每 500 毫秒执行一次 *C 动作*，在每次调用之后，两个外部 C 变量都重新设置为 *FALSE*。
- 如果识别出双击，则根据变量 *nButtonID* 执行一个动作。该变量包含选择*静态文本*的依据。
- 返回画面宽度。

4.5.3 颜色对话框(ex_3_chapter_01c)

可通过在 *ex_3_chapter_01c* 画面中使用选择如上所示的按钮来访问在本实例中所描述的颜色对话框。

任务定义

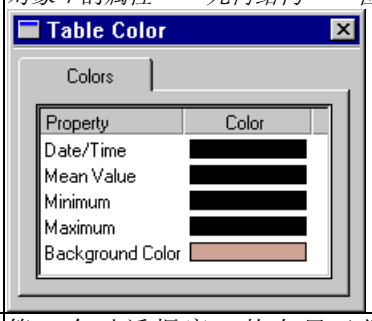
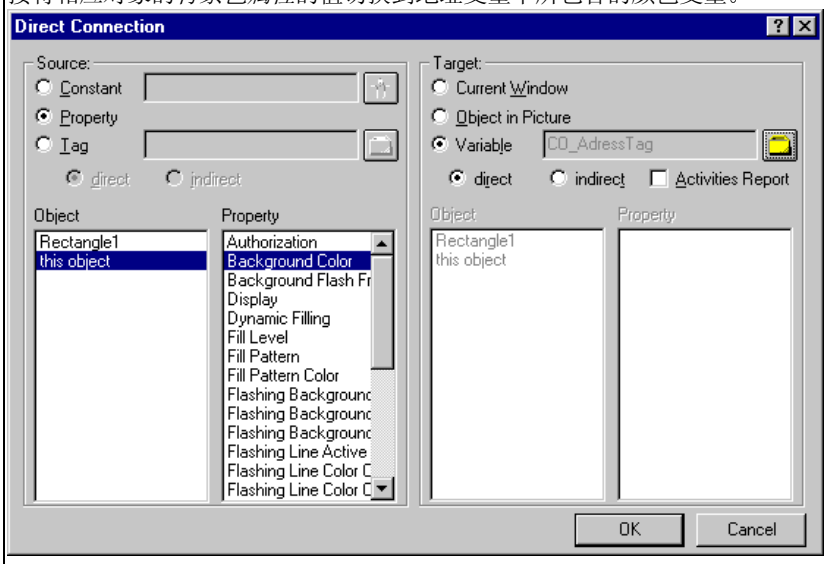
使用各种不同的对话框窗口，对*变量记录*这章的自定义的表格布局(ex_3_chapter_01c.pdl)实例中所描述的表格的颜色设置进行修改。对表格的背景色以及每列的字体颜色进行编辑。

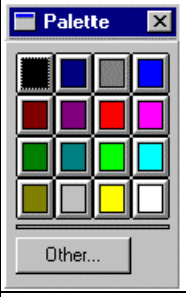
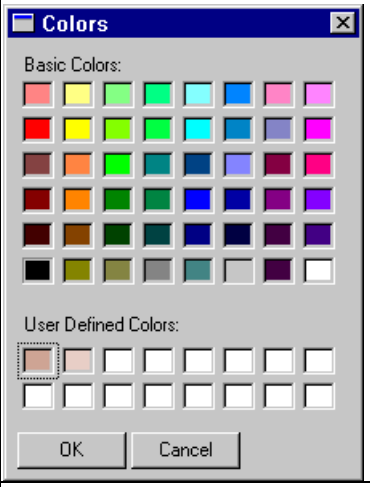
概念的实现

总共使用三个画面来完成颜色对话框。在选择上面所显示的按钮之后所访问的第一幅画面包括了一个显示当前颜色设置的对话框。从该对话框中，可对包含有 16 种基本颜色的颜色选择对话框进行访问，这可通过D 每个可设置的表格属性来完成。从该对话框中，也可通过一个按钮对包含有 50 种颜色的另一个颜色对话框进行访问。

在图形编辑器中的实现

步骤	过程：在图形编辑器中的实现
1	使用了五个*无符号 32 位数*类型的变量。在本实例中，它们是 C*O_TIME*、*CO_MAX*、*CO_MIN*、*CO_MEAN* 与 *CO_BACK* 变量。这些变量存储当前颜色值。 此外，还使用了一个*无符号 32 位数*类型的变量。在本实例中，这就是 *CO_TEMP* 变量。该变量用作颜色值的缓冲区，可通过*确定按钮*予以应用。 使用了一个 *16 位字符集文本变量*类型的变量作为地址变量。所要处理的颜色变量的名称存储在该变量中。在实例中，这就是 *CO_AdressTag*。
2	在 *ex_3_chapter_01c.pdl* 画面中组态的表格由多种单个对象组成。这些将要进行修改的对象的所有属性均具有一个与某个颜色变量的*变量连接*。

步骤	过程：在图形编辑器中的实现
3	可使用一个画面，在其中创建第一个对话框窗口的布局。这就是 *ex_10_FD_00.PDL* 画面。为所要设置的每个属性组态一个*标准对象* → *静态文本*与一个*标准对象* → *矩形*。 矩形显示了每个属性的当前设置颜色。这可通过一个与相关颜色变量的*变量连接*来执行。通过*事件* → *鼠标* → *按下左键处*的*矩形和静态文本的 C 动作*，可将相应颜色变量的名称写入到地址变量中。 双击其中一个矩形，可打开从 16 种基本颜色中进行颜色选择的对话框。在*图形对象 1* 的*属性* → *几何结构* → *位置 X* 处双击。 
4	第二个对话框窗口的布局已经在另一个画面中进行了组态。这就是 *ex_10_FD_01.PDL* 画面。为每种可选颜色组态了一个*标准对象* → *矩形*，其背景色对应于所要设置的颜色。 在*事件* → *鼠标* → *按下左键*处，创建了一个用于各个矩形的*直接连接*。该连接将相应对象的*背景色属性*的值切换到地址变量中所包含的颜色变量。 

步骤	过程：在图形编辑器中的实现
5	在*事件* → *鼠标* → *鼠标动作*处，为每个*矩形*创建了一个*直接连接*，以使当前窗口不可见。 通过一个单独的 *Windows 对象* → *按钮*，可打开下一个对话框。 
6	第三个对话框窗口的布局已经在另一个画面中进行了组态。这就是 *ex_10_FD_03.PDL* 画面。正如步骤 4 所述，可组态一个*标准对象* → *矩形*，用于每种可选择的颜色。然而，*事件* → *鼠标* → *按下左键*处的*直接连接*，并没有描述由地址变量所指示的颜色变量，而只是描述了临时颜色变量 *CO_TEMP*。只有在按下确定按钮后，该变量中所包含的值才会写入所要处理的颜色变量中。 在画面中，已组态了一个*智能对象* → *图形对象*，它显示了当前包含在 *CO_TEMP* 变量中的颜色值。在实例中，这就是*选择*对象。一旦通过*事件* → *鼠标* → *按下左键*处的 *C 动作*选择了一个矩形，该对象的位置就将发生变化。 
7	在 *ex_3_chapter_01c.pdl* 画面中，为每个对话框组态了一个*智能对象* → *画面窗口*。通过一个*窗口对象* → *按钮*来打开包含在第一个对话框中的*画面窗口*。

4.5.4 棒图显示(ex_3_chapter_01e)

本实例中所描述的棒图显示已在*变量记录*一章的以定义的时间进行归档(ex_3_chapter_01e.pdl)实例中使用。

任务定义

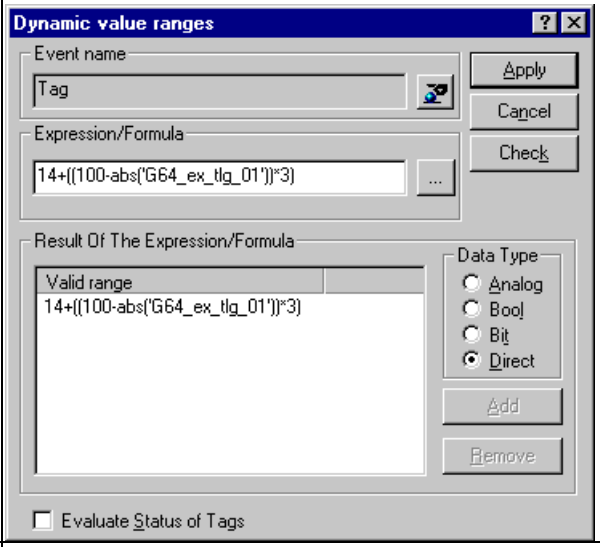
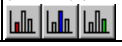
使用三个棒图，对归档在相应画面中的变量当前值进行显示。各个棒图可通过按钮分别激活。

概念的实现

每个棒图由一个代表棒图前景色的*智能对象* → *状态显示*以及一个在其中显示棒图背景色的*智能对象* → *画面窗口*所组成。通过一个*动态对话框*，可根据所要显示的变量的值对*画面窗口*的高度进行控制。

在图形编辑器中的实现

步骤	过程：在图形编辑器中的实现
1	有三个画面可用，每个画面均只由一个*智能对象* → *图形对象*组成。这些画面是 *ex_10_BH_00.pdl*、*ex_10_BH_01.pdl* 与 *ex_10_BH_02.pdl*。在各个图形对象中，所显示的位图代表棒图背景。
2	在 *ex_3_chapter_01e.pdl* 画面中组态三个*智能对象* → *状态显示*，它们显示各个棒图的前景。如果棒图没有激活，则它们显示棒图的背景。

步骤	过程：在图形编辑器中的实现
3	在各个*状态显示*上面放置一个*智能对象* → *画面窗口*，它们具有步骤 1 中所描述的在*属性* → *其它* → *画面名称*处设置的画面。 在*属性* → *几何结构* → *窗口高度*处，各创建一个*动态对话框*，它可根据所要显示的变量值来控制画面窗口的高度。 只有正值才能显示。因此，使用 *abs()*函数来生成变量的绝对值。要显示的最大值为 100。由于*画面窗口*代表棒图的背景，所以必须用最大值减去变量值，以获得背景高度。位图中棒图的一个单位等于三个像素，因此计算的棒图背景高度必须乘以三。位图中在棒图背景上面的 14 个像素保留为空白，所以必须将其添加到画面高度中。 
4	通过工具栏上的三个*智能对象* → *状态显示*，可取消激活棒图。这通过各自在*事件* → *鼠标* → *按下左键*处的 *C 动作*来完成。该 *C 动作*切换*画面窗口*的可见性、切换包含棒图状态显示的画面并且切换由其自身对象来显示的画面。 

索引

字母

API, 4-107
  用于消息过滤器, 4-107
CSV, 4-61
  自动导出归档, 4-61
DLL, 4-88
  集成, 4-88
OCX
  使用, 3-38
OLE
  与 MS Excel 的连接, 4-149
SysMalloc, 3-38
WinCC
  结束, 3-32

A

按击操作, 2-7

B

报警
  创建, 4-70
  归档, 4-107
  回路, 4-102
  位消息操作步骤, 4-70
  限制值监控, 4-83
  行, 3-3
报警回路, 4-102
变量
  归档, 4-48
  模拟, 2-47
  文档, 4-128
表格, 4-40
  显示过程值, 4-27
  自定义, 4-40
表格布局, 4-40

C

初始化, 4-40
  带调用的回调函数, 4-40
  模拟, 2-48
  在项目中, 3-41
创建
  变量, 2-2
  单个消息, 4-70
  过程值归档, 4-3
  消息等级, 4-83
  组消息, 4-114

D

打印作业, 4-133
  时间选择, 4-133
导出
  过程值归档, 4-61
导入
  变量, 2-52
动态
  报表的一个部分, 4-142
动态对话框, 4-48
  使用, 4-48
短期归档, 4-107
  创建, 4-107
  打印消息, 4-147

F

附注, 4-88

G

更新
  趋势窗口, 4-18
归档, 4-61
  报警, 4-107
  导出, 4-61
  非周期性, 4-27
  每次满一分钟, 4-55
滚动条, 2-26
过程值归档, 4-3
  创建, 4-3

H

画面, 3-24
  布局, 3-3
  窗口, 3-3
  创建画面索引, 4-152
  调整大小, 3-39
  改变, 3-5
  改变大小, 3-41
  几何结构, 3-39
  时间控制的取消选择, 3-24
  文档, 4-121
  显示画面名称, 3-5
  项目, 3-1
回调函数
  使用, 4-40

J

键
  用于消息过滤器, 4-107
结束
  从 WinCC, 3-32

K

可操作
  操作员控制允许, 3-32
控制
  无鼠标, 3-21
控制窗口, 3-45
口令, 4-27
  在项目 example_01 中, 4-27

L

连接
  与 MS Excel, 4-149

M

模拟, 2-47
模拟值
  画面, 4-83

N

内容
  来自 example_01, 4-156

Q

切换操作, 3-45
  二进制, 3-45
切换开关, 2-16
区域
  画面, 3-3
趋势, 4-2
  过程值的显示, 4-2
取消选择, 3-24
  画面, 3-24
  画面窗口, 3-23
确认, 4-83
  蜂鸣器, 4-83

R

热键
  组态, 3-21
任务
  消息顺序报表, 4-145

S

设定值, 2-8
  更改, 2-26
声音, 4-88
时间, 3-38
  显示, 3-38
时间选择, 4-133
授权等级, 3-33
输入
  检查, 3-52
  通过复选框, 2-32
  通过滚动条, 2-26
  通过选项钮, 2-30
数据
  归档, 4-40

T

通讯
带 MS Excel 的 OLE, 4-149

W

位处理, 2-35
位模式, 2-35
位消息操作步骤, 4-70
文本输入, 3-30
文档, 4-121
变量, 4-128
画面, 4-121
趋势窗口, 4-133
无鼠标, 3-21
操作, 3-21

X

系统时间, 4-61
确定, 4-61
显示, 3-21
画面窗口, 3-21
向导
用于过程值归档, 4-3
项目
启动, 2-21
项目库, 3-5
消息
打印, 4-147
归档, 4-107
位消息操作步骤, 4-70
指定颜色, 4-70
组态, 4-70
信息
显示, 3-11
隐藏, 3-36
信息框, 3-27
组态, 3-27

Y

颜色, 3-15
改变, 3-15
隐藏
信息, 3-11
语言
在运行系统中, 3-19
运行系统
打印表格, 4-138
打印趋势窗口, 4-133
结束, 3-32
启动、停止归档, 4-18
语言切换, 3-19

Z

增量, 2-18
指针化显示, 2-44
注册, 3-37
状态显示, 3-16
总览, 3-3
画面, 3-3
组
变量管理器, 2-2
组合键, 3-21
窗口切换, 3-21
登录、退出, 3-33
组态
报警回路, 4-102
消息的颜色方案, 4-70
组消息, 4-114