DENSO

DENSO机械手

RC7型控制器用

操作盘功能说明书

使用说明书（增补版）

前言

此程序是在电脑上创建多功能教导器的操作盘画面的软件。将零部件配置在画面上，仅通过记述让各零部件进行动作的动作就可以简单地创建操作盘数据。本说明书对该操作盘功能进行说明。

有关全球型控制器的注意事项：

Ver.2.801以前版本的控制器

在 "外部自动" 限定模式下，无法通过外部自动使用 "操作盘功能"。

（参照RC7M型控制器说明书）

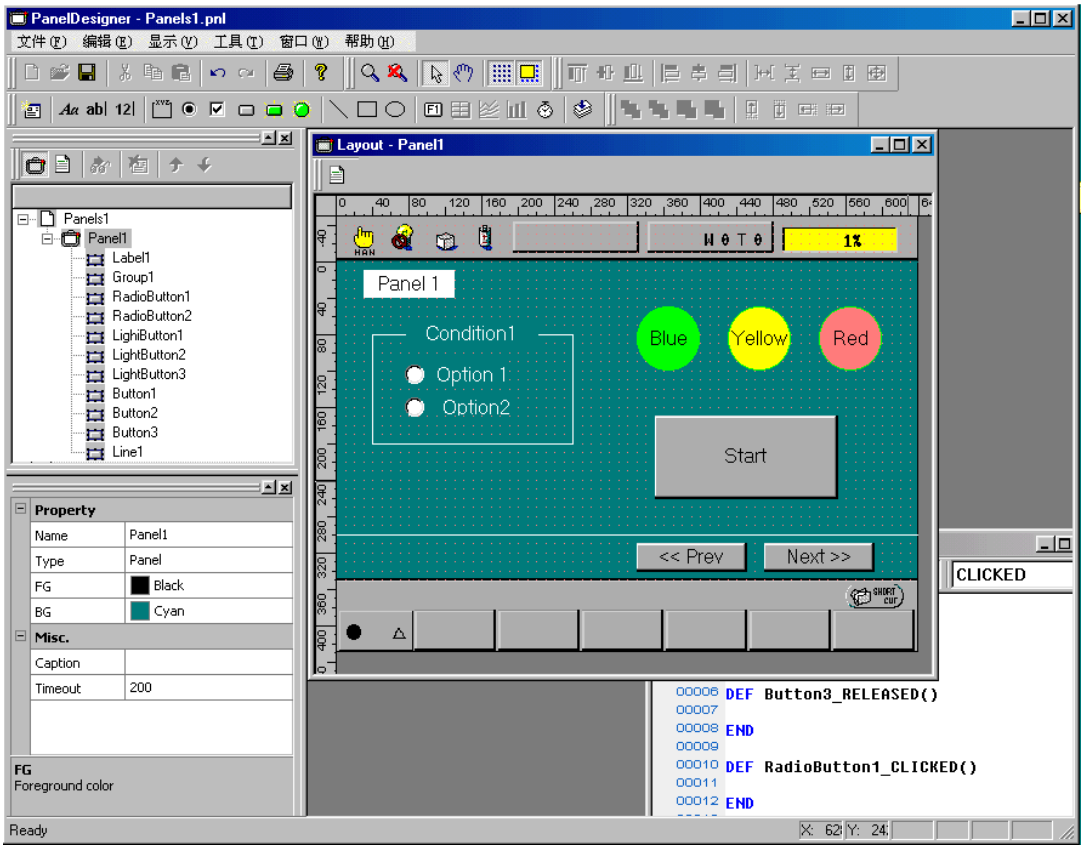
Ver.2.802以后版本的控制器

即使处于外部自动模式也可使用 “操作盘功能”。但是，在外部自动模式的情况下，操作盘记述语言的RUN、CONTINUERUN指令将无法执行。

目录

第1章 操作盘编辑程序

操作盘编辑程序是一个编入到WINCAPSⅢ中的、在电脑上创建多功能教导器操作盘画面的软件。将零部件配置在画面上，仅通过记述让各个零部件进行动作，就可以简单地创建出操作盘数据。在此就其操作方法进行说明。



用操作盘编辑程序创建的操作盘画面示例

1.1　操作盘数据创建步骤的概要内容

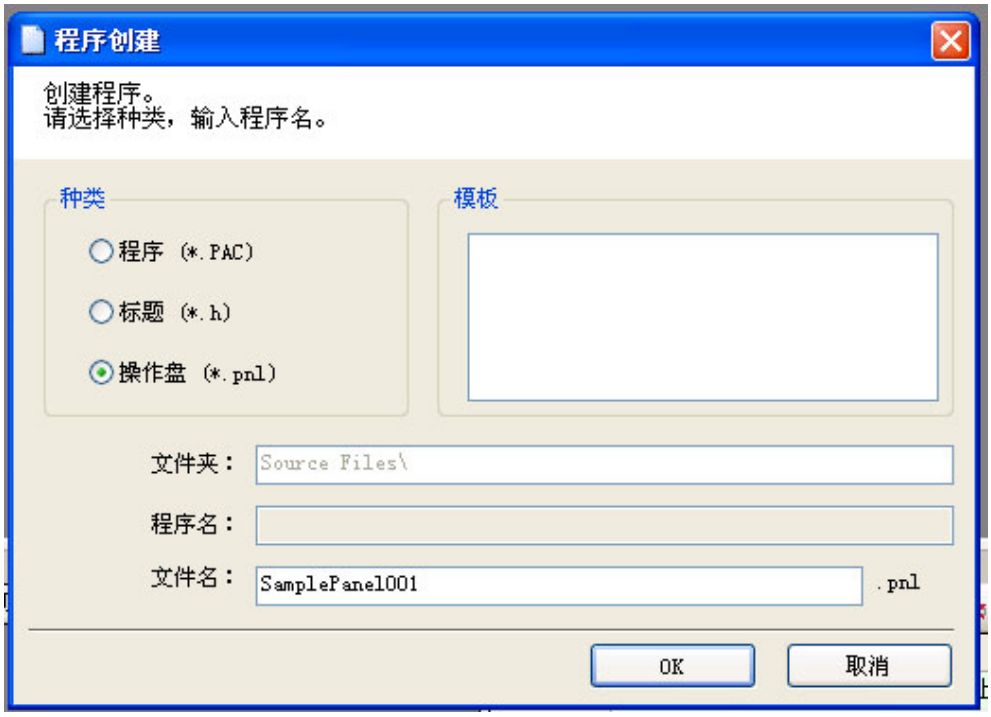
在 (1)～(5) 中表示创建操作盘数据步骤的概要内容。

(1)　操作盘编辑程序的启动

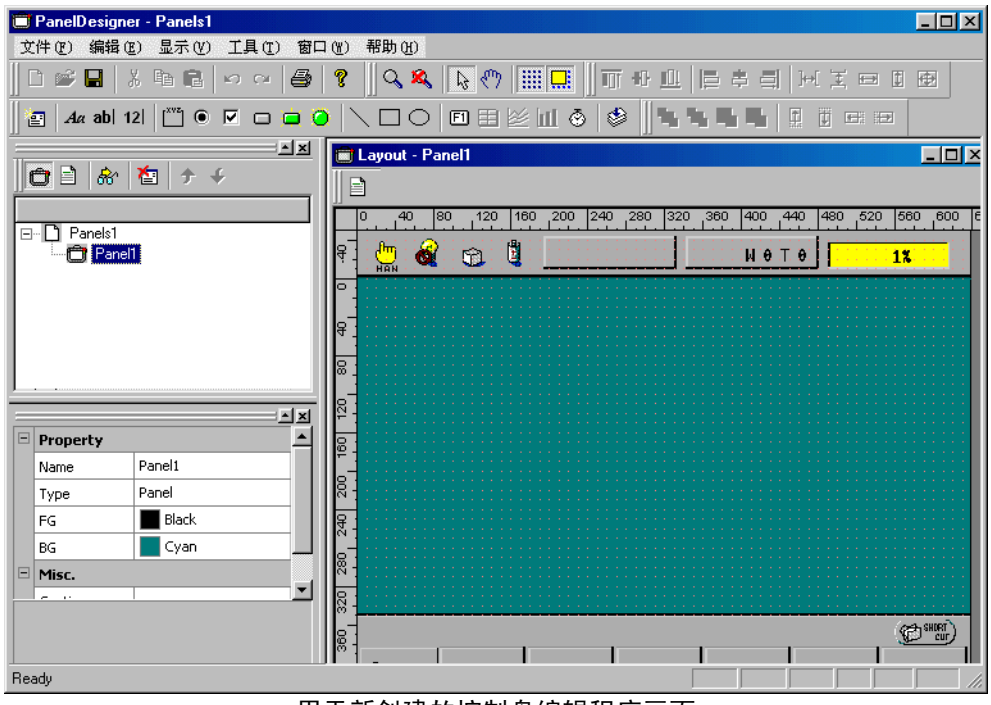
①在WINCAPSⅢ中选择[项目(P)] → [程序创建(P)]，创建程序对话框就会显示出来。

②在种类中选择［操作盘(*.pnl)］，输入文件名，按[OK]，PanelDensigner（操作盘编辑程序）就会启动。

注：当要打开已有的操作盘数据时，请双击程序一览内的操作盘数据。



WINCAPSⅢ创建新程序对话框

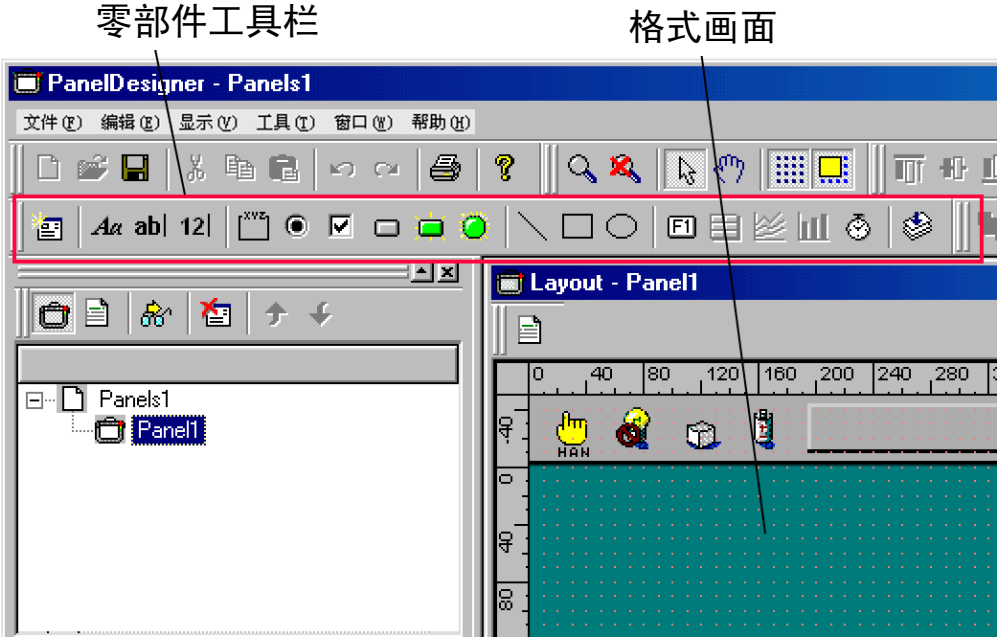


用于新创建的控制盘编辑程序画面

(2) 操作盘画面的创建

从操作盘编辑程序中备有的 "零部件工具栏" 中指定需要的零部件，配置在格式画面上创建操作盘画面。

详细内容请参照 "第2章 用操作盘编辑程序创建操作盘"。

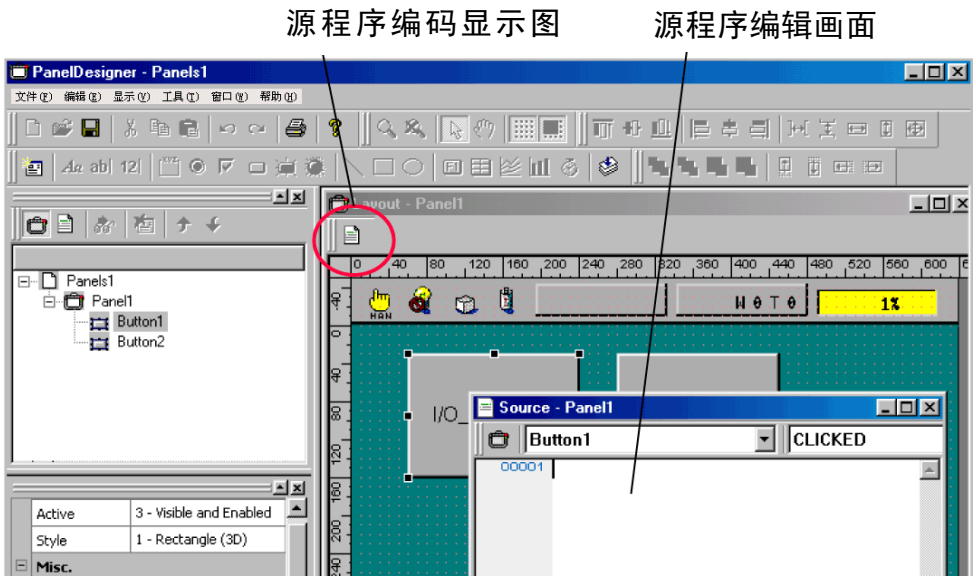


(3) 动作编码的编辑

①点击格式画面的源程序编码显示图标，打开源程序编辑画面。

②在源程序编辑画面上记述零部件被按压时使之进行动作的动作编码。

详细内容请参照 "2.2.2 零部件的动作的指定"。



(4) 编译

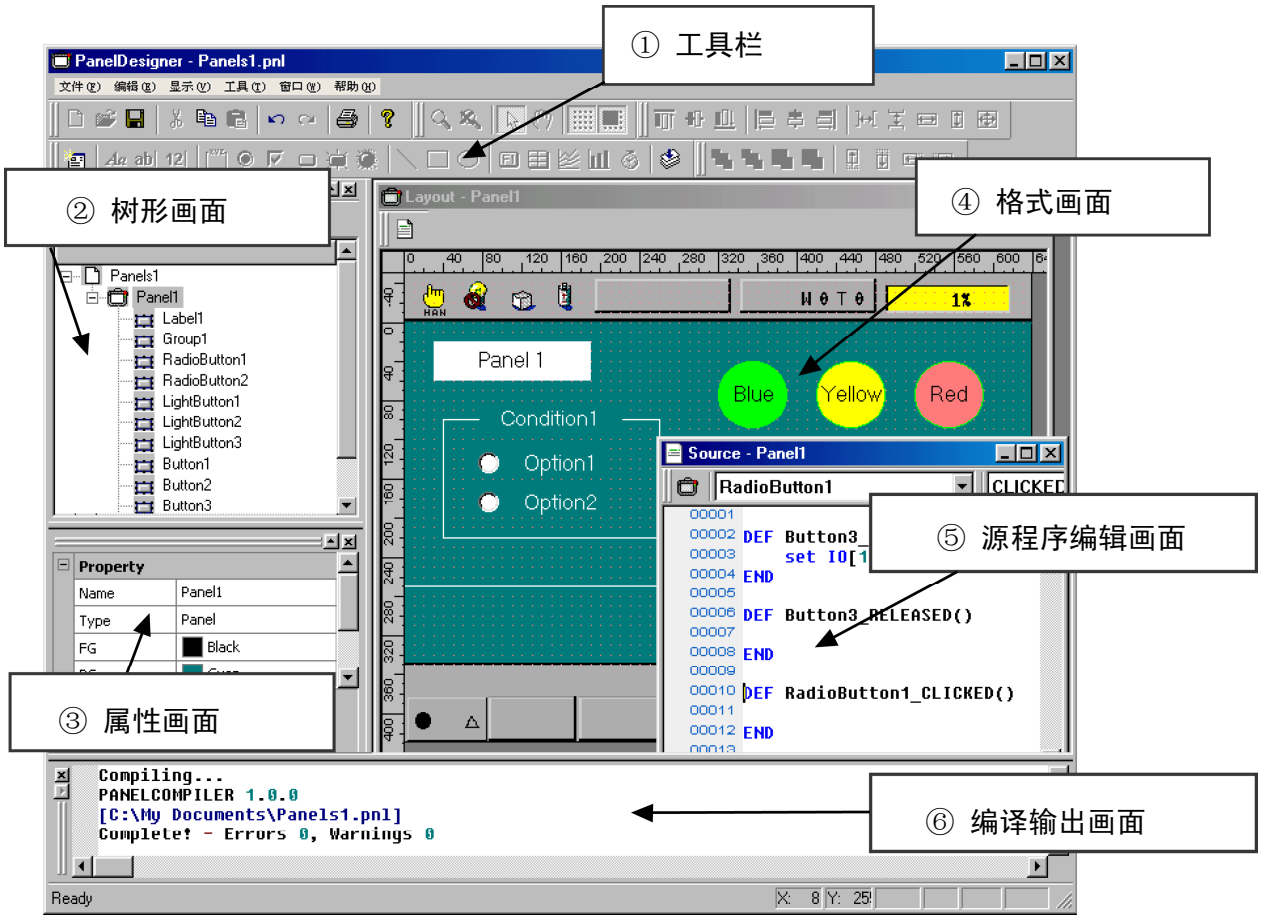
通过编译确认所记述的动作编码中是否有错误。
编译结果被显示在操作盘编辑程序画面下部的输出视窗。

(5) 向控制器发送数据

利用WINCAPSⅢ将所创建的操作盘文件发送给控制器。由此，可以将多功能教导器作为操作盘使用（注）。
注：需要对操作盘功能有效化的多功能教导器进行设定。

1.2 操作盘编辑程序画面的功能说明

在下图表示操作盘编辑程序的格式。关于各个画面的功能如下所述。



操作盘编辑程序的格式

1.2.1 工具栏

在操作盘编辑程序画面中，为了创建操作盘数据，备有非常方便的各种工具栏。

(1) 标准工具栏

操作盘编辑程序的标准工具栏的功能如下所示。

	名称	说明
	新建	创建新的操作盘文件。
	打开	打开已有的操作盘文件。
	保存	将编辑中的文件覆盖原文件保存。
	剪切	剪切选择范围，移动到剪贴板。
	复制	将选择范围复制到剪贴板上。
	粘贴	将剪贴板的内容插入到当前的光标位置。
	无法撤消	将前一步进行的操作退回到原状。
	无法重复	再次执行退回原状的操作。
	打印	打印当前的画面。
	版本信息	显示操作盘编辑程序的版本信息。

(2) 切换 "标尺、网格" 工具栏

切换格式画面的大小变更、网格显示的ON / OFF。

	名称	说明
	标尺	放大、缩小显示选择区域。
	标准倍率	取消标尺，返回到标准显示 (100%)。
	移动	将显示画面向指定的方向移动。
	网格 ON / OFF	切换网格的显示 / 隐藏。
	网格位置对齐	将零部件在网格上定位对齐。

(3) "格式" 工具栏

调合被选择的零部件的位置及大小。

	名称	说明
	上端对齐	将选择的零部件的上端一致。
	中心对齐 – 垂直	将选择的零部件的上下中心一致。
	下端对齐	将选择的零部件的下端一致。
	左端对齐	将选择的零部件的左端一致。
	中心对齐 – 水平	将选择的零部件的左右中心一致。
	右端对齐	将选择的零部件的右端一致。
	水平间隔	将选择的零部件的横间隔一致。
	垂直间隔	将选择的零部件的纵间隔一致。
	宽度调整	将选择的零部件的宽度一致。
	高度调整	将选择的零部件的高度一致。
	大小调整	将选择的零部件的大小一致。

(4) 零部件工具栏

选择配置在格式画面上的零部件。

	名称	说明
	新面板	创建新格式页面。
	零部件的选择	切换格式画面到选择模式。
	标签	零部件按钮：标签
	文本框	零部件按钮：文本框
	数值输入框	零部件按钮：数值输入框
	组框	零部件按钮：组框
	无线电按钮	零部件按钮：无线电按钮
	检查框	零部件按钮：检查框
	Push 按钮	零部件按钮：Push 按钮
	照明式按钮	零部件按钮：照明式按钮
	信号灯	零部件按钮：信号灯
	直线	零部件按钮：直线
	四边形	零部件按钮：长方形
	椭圆	零部件按钮：椭圆
	功能键	零部件按钮：功能键
	计时器	零部件按钮：计时器
	编译	编译所创建的操作盘数据。

(5) 移动工具栏

变更被配置在格式画面上的零部件的位置及上下关系。

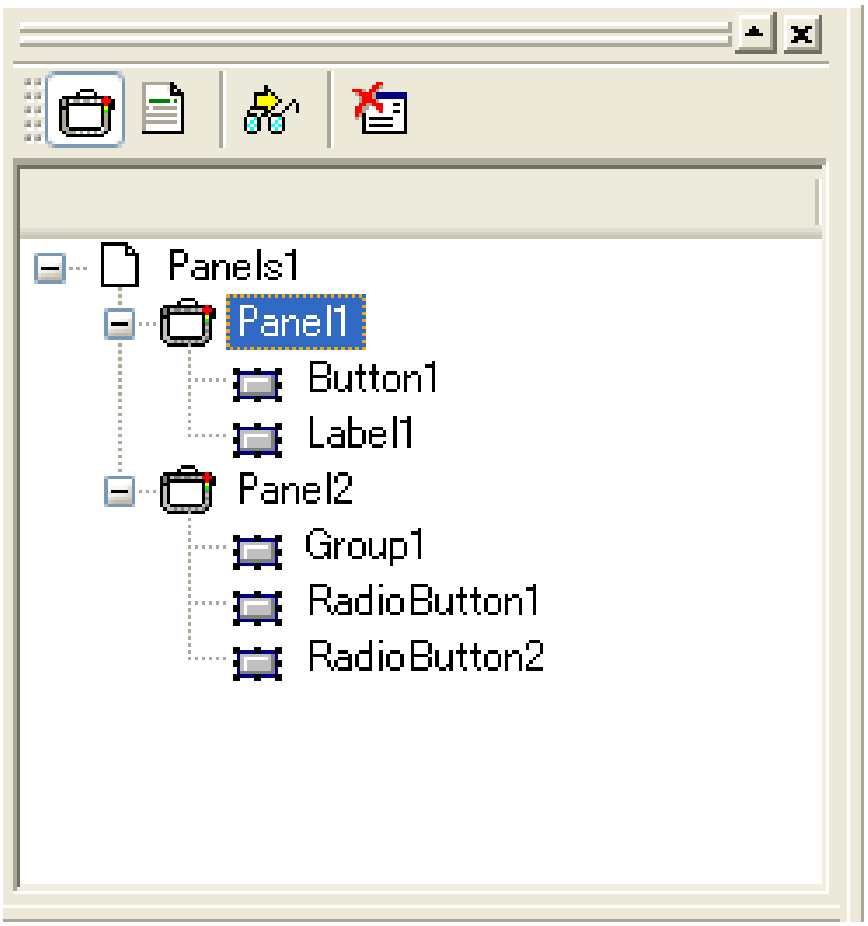
	名称	说明
	至最前面	将选择的零部件配置在最前面。
	至最背面	将选择的零部件配置在最背面。
	至前面	将选择的零部件向前移动一个。
	至背面	将选择的零部件向后移动一个。
	向上	使选择的零部件向上方移动。 若按下 "Shift" 键，则以每 5 点进行移动。
	向下	使选择的零部件向上方移动。
	向左	使选择的零部件向上方移动。
	向右	使选择的零部件向上方移动。

1.2.2 分类目录画面

分类目录显示被格式画面配置的零部件。

(1) 分类目录画面

分类目录画面如下图所示。
若双击零部件，则对应的画面会被显示。



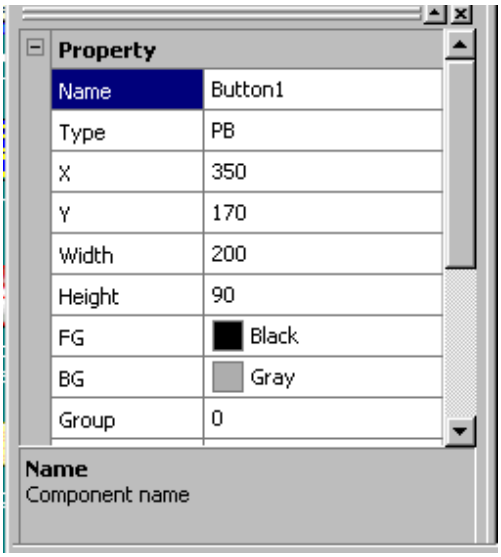
(2) 用于分类目录画面的工具栏

用于分类目录画面的工具栏和功能如下表所示。

	名称	说明
	格式样式	选择格式画面
	源程序样式	指定源程序编辑画面
	面板显示	显示上述选择画面。
	删除面板	删除格式画面。

1.2.3 属性画面

指定零部件的位置、大小等的属性。被显示的项目根据零部件的种类的不同而不同。详细内容请参照 "1.5.1　属性一览"。



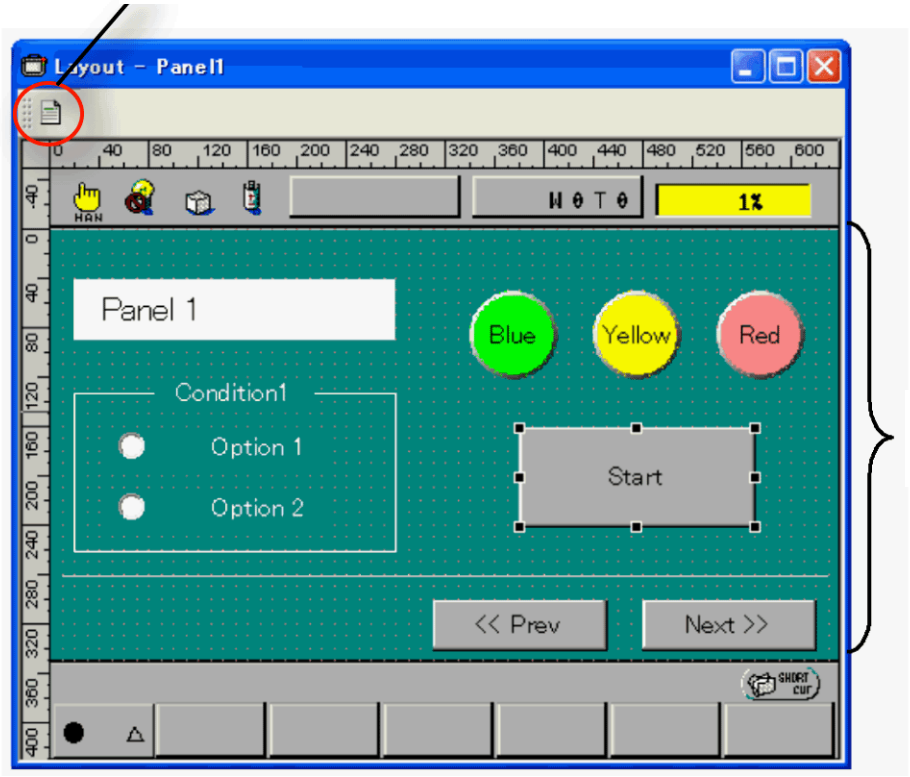
属性画面

1.2.4 格式画面

(1) 是设计被多功能教导器显示的 "操作盘画面" 的画面。
将选择的零部件配置在该画面上。所配置的零部件可以通过拖拽光标键或橡胶带来调整位置及大小。

(2) 若点击用于显示源程序编码的图标，则在画面上会显示所对应的源程序编辑画面。

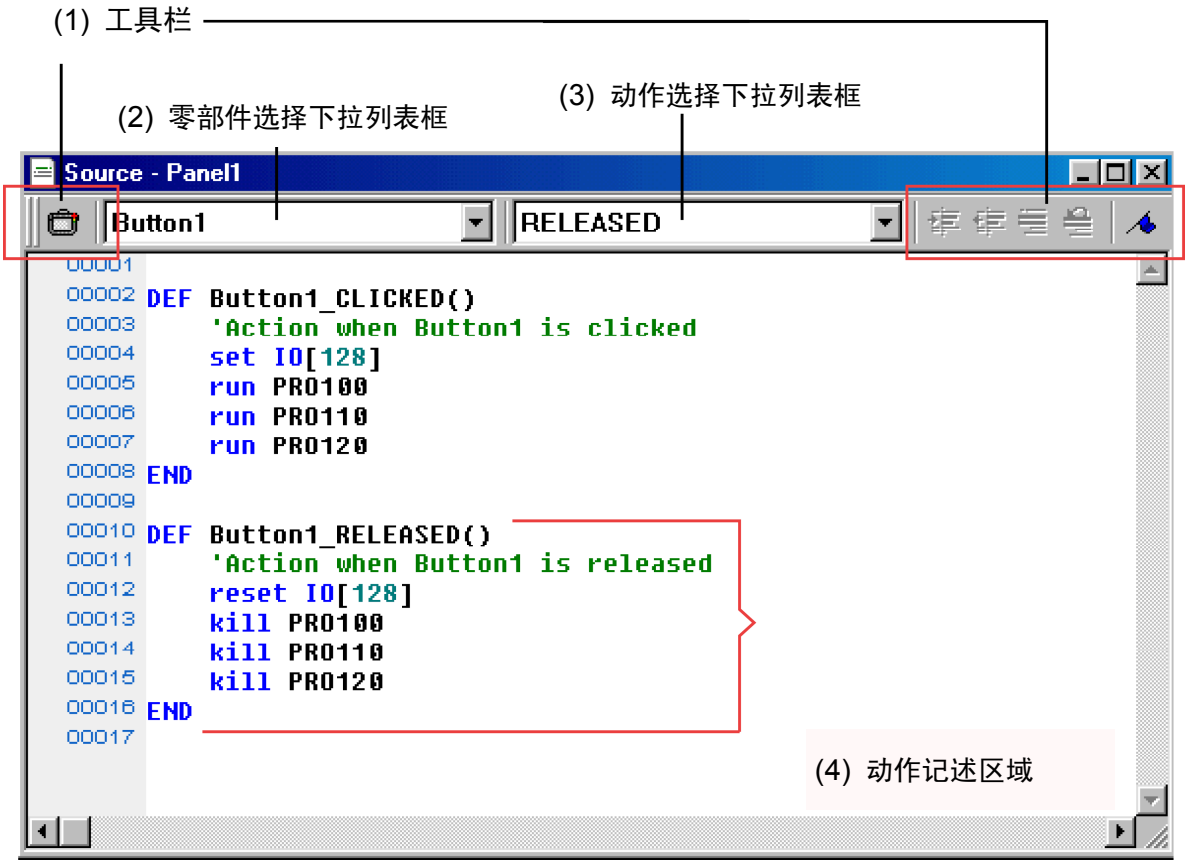
源程序编码显示用图标



格式画面

1.2.5 源程序编辑画面

是记述使配置在布局画面上的零部件进行动作的画面。



源程序编辑画面

(1) 源程序编辑画面的工具栏

	名称	说明
	格式画面	在画面上显示对应的格式画面。
	右缩进	将选择行向右缩进 1Tab 单位。
	左缩进	将选择行向左缩进 1Tab 单位。
	注释	注释选择行。
	非注释程序段	取消选择行的注释。
	书签	进行书签的设定与解除。
	下一个书签	将光标移动到下一个书签。
	上一个书签	将光标移动到上一个书签。
	解除书签	解除所有的书签。
	检索、置换	检索或置换被指定的字符串。

注：若设定书签，则在编码行的左侧会有  显示。

(2) 零部件选择下拉列表框

从下拉列表框中选择记述动作的零部件。

(3) 动作选择下拉列表框

用 "(2) 零部件选择下拉列表框" 中的零部件能够指定的动作一览被显示。
若选择，则在编辑画面上 "(4) 动作记述程序段" 的框架会被自动生成。

```
［例］：被插入的动作记述程序段的框架（Button1 被点击时）
        DEF Button1_CLICKED ()

        END
```

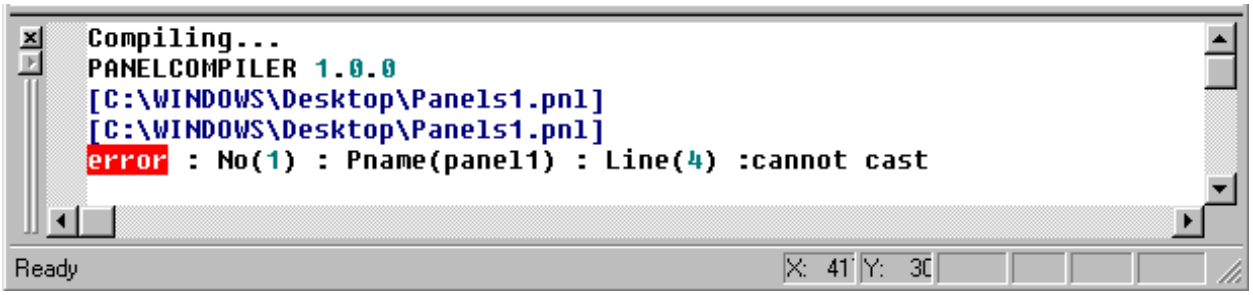
(4) 动作记述程序段

在用 "(3) 动作选择下拉列表框" 被生成的动作记述程序段内，记述使之进行动作的编码。

```
［例］：被插入的动作记述程序段的框架（Button1 被点击时）
        DEF Button1_CLICKED ()
            Set IO [128]      '打开 IO 128 号
            Run PRO100    '启动 PRO100
        END
```

1.2.6 编译输出画面

显示所创建的操作盘数据的编译结果。
通过双击错误行转移到 "源程序编辑画面" 的对应错误的位置。



编译输出画面

1.2.7 菜单说明

在此，就操作盘编辑程序画面的菜单进行说明。

(1) 文件 (F)

菜单		内容
	新建 (N)	创建新的操作盘文件。
	打开 (O) ...	打开已有的操作盘文件。
	关闭 (C)	关闭正在编辑的文件。当文件未被保存时将会显示用于保存的对话框。
	保存 (S)	将编辑的内容保存到现有文件中。 当新创建文件时，会显示用于保存的对话框。
	另保存 (A) ...	将编辑的内容作为另一个文件进行保存。
	打印 (P) ...	打印被选择的 "格式画面" 或 "源程序编辑画面" 的内容。
	打印预览 (V) ...	显示打印出来时的样子。
	打印机的设定 (R) ...	显示打印机的设定对话框。
	引入 (I) ...	从其他的操作盘文件获取 "格式画面"。
	最近使用过的文件	显示过去所保存过的操作盘文件的一览。
	应用程序的结束 (X)	结束操作盘编辑程序。

(2) 编辑 (E)

菜单		内容
	返回原状 (U)	返回到之前的的操作。
	重新进行 (R)	再一次进行返回到原状的操作。
	剪切 (T)	剪切选择范围，移动到系统的剪贴板。
	复制 (C)	将所选择的零部件及字符串复制到系统的剪贴板。
	粘贴 (P)	将剪贴板的内容插入到当前的光标位置。
	删除 (D)	删除所选择的零部件及字符串。
	检索、置换 (F)	显示进行所指定的字符串的检索及置换的对话框。

(3) 显示 (V)

菜单		内容
	工具栏	切换各个工具栏的显示 / 隐藏。
	状态栏 (S)	切换状态栏的显示 / 隐藏。
	分类目录栏 (T)	切换分类目录画面的显示 / 隐藏。
	属性栏 (P)	切换属性画面的显示 / 隐藏。
	面板格式 (A)	显示所选择的格式画面。
	网格 (G)	切换格式网格的显示 / 隐藏。
	网格位置对齐 (S)	切换网格自动位置对齐功能的打开 / 关闭。
	标尺 – 返回到标准 (Z)	以标准倍率显示格式画面。
	标尺 – 放大倍数的指定 (O)	指定格式画面的显示倍率。

(4) 工具 (T)

菜单		内容
	设定 (O)	选择编译器的输出版本。
	编译 (C)	将创建的操作盘数据翻译成执行文件。

(5) 视窗 (W)

菜单		内容
	关闭 (O)	关闭被选择的视窗。
	关闭所有视窗 (L)	关闭正被显示的所有视窗。
	重叠显示 (C)	将打开着的所有视窗以同样大小并且能够看见标题栏地进行重叠显示。
	并列显示 (T)	并列显示打开着的视窗。
	图标的排列 (A)	将被最小化了的视窗排列在主视窗的左下方。
	显示画面一览	显示正被显示的视窗的一览。

(6) 帮助 (H)

菜单		内容
	帮助	显示操作盘编辑程序的帮助。
	版本信息	显示操作盘编辑程序的版本画面。

1.3 创建、变更布局

1.3.1 追加零部件

在格式画面上新追加零部件的步骤如 (1)～(3) 中所示。

(1) 显示格式画面

使其显示要创建、变更的格式画面。

①新创建时

选择 "文件" 菜单中的 "新建" 或 "：新面板" 工具栏。

②已有画面的变更时

在分类目录画面上选择 "：格式样式" 之后，双击使其显示的 "Panel1：格式分类目录" 或点击 "：面板显示" 按钮。

(2) 选择零部件

从零部件工具栏中选择要追加的零部件，选择后在格式画面上的光标位置零部件标记 [ ] 会被显示。

(3) 追加零部件

点击在格式画面上要配置零部件的地方后，零部件以默认的大小被追加。

注：若在追加时进行拖拽，则同时可调整零部件大小。

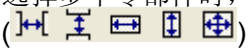
1.3.2 变更零部件的格式

用以下的任意一种方法变更在格式画面上配置了的零部件的位置及大小。

(1) 移动

①通过鼠标拖拽（移动光标正在显示）
②光标键；1
③移动工具栏 ()
④变更属性X、Y项目的值

(2) 变更大小

①拖拽零部件的外框的橡胶带
②变更属性的Width、Height项目的值
③选择多个零部件时，通过格式工具栏可以一括调整间隔、大小
()

(3) 位置对齐

选择多个零部件时，通过格式工具栏可以一括调整对中及对端
()

注：［功能］零部件用 [Index] 属性决定位置、大小。

(4) 变更零部件的顺序（重叠）

选择在格式画面上要变更的零部件后，从右键点击弹出菜单的 "移动" 或 "：顺序工具栏" 选择操作。

注：若变更顺序，则与之相应的分类目录画面的零部件的排列也会被变更。

1.3.3 变更零部件的属性

用属性画面的项目可以变更零部件名称及颜色等的属性。

1.3.4 删除格式画面

在分类目录画面上选择删除的 "  ：格式分类目录" 之后，点击 "  ：面板删除" 按钮。

1.3.5 从其他的操作盘文件引入格式画面

可以从其他的操作盘文件（扩展符 = pnl）以格式画面单位引入数据。

(1) 要引入的操作盘文件的选择

从 "文件" 菜单中的 "引入"，选择引入源的文件。

(2) 格式画面的选择

从文件中包含的格式一览中选择要引入的格式，点击 "Import" 按钮。
引入到零部件分类目录中的格式被追加。

1.4 记述动作编码

使用源程序编辑画面，对操作盘画面上的零部件发生 "被按下时" 及 "被放开时" 等状态变化时所进行的处理进行记述。

1.4.1 动作编码的记述方法

(1) 源程序编辑画面的显示

用以下的任意一种方法显示与记述动作的零部件相对应的 "源程序编辑画面"。

①从格式画面双击零部件。

②在格式画面上选择零部件后，点击 "：格式显示" 按钮。

③在零部件分类目录画面上选择零部件分类目录后，确认 "：源程序样式" 按钮被按，然后点击 "：面板显示" 按钮。

(2) 零部件的选择

确认在源程序编辑画面的零部件选择下拉列表上记述动作的零部件是否被显示。若未被显示，则从下拉列表框中选择。

(3) 动作的选择

在动作选择下拉列表框中可以使用的行动一览被显示。若选择记述动作，则在编辑画面中动作记述程序段的2行会被自动插入。

例：当选择了 Button1 的 CLICKED 时。

```
DEF Button1_CLICKED ()

END
```

(4) 动作编码的记述

在通过 (3) 被插入的程序段中，记述使其进行的动作指令。

例：当选择了 Button1 的 CLICKED 时。

```
DEF Button1_CLICKED ()
    Set IO [128]        '打开 IO128 号
    Run PRO100          '启动 PRO100
    Run PRO200          '启动 PRO200
END
```

1.4.2 记述了的编码的确认（编译）

检查记述了的动作编码的语法，将结果显示在 "编译输出画面"。若有错误，在视窗内会显示错误内容。

通过双击错误行，"源程序编辑画面" 的对应的位置会被显示。

1.5 其他

1.5.1 属性一览

被属性画面显示的位置、大小等的属性一览如下表所示。

注：被属性画面显示的属性，根据零部件的种类的不同而不同。

名称	内容	备注
Name	名称	零部件的识别名称
Type	零部件的种类	每个零部件是固定的。
X	X 坐标	是在多功能教导器描绘范围内的 X、Y 轴的基准位置。
Y	Y 坐标	
Widht	宽度	以像素单位设定从基准位置起的大小。
Height	高度	
FG	前景色	从下拉列表框中选择颜色。
BG	背景色	
Group	组编号	是零部件所属的组的编号。
Active	有效 / 无效设定	从下拉列表框中选择。
Style	显示形式	从下拉列表框中选择显示样式。
Caption	显示字符串	是在零部件的表面上被显示的字符。 注：当对应多行时用 "Ctrl + Ret" 换行
FSize	字符大小	0: SS、1: S、2: M、3: L
Justify	显示文字位置	0：中央、1：居右、2：居左
Thickness	线宽	用像素单位设定线的粗度。 注：当为 0 时，全部涂抹
MyGroup	管理组编号	组框特有
State	状态	从下拉列表框中选择 ON / OFF 等的状态。
Value	输入值	数值输入框特有
Text	输入文本	文本框特有
Index	功能编号	功能键特有
Interval	间隔时间	定时器特有
Timeout	超时时间	背景特有（面板文件上有 1 个）
Release-Mode	RELEASED 行动发生条件	背景特有（面板文件上有 1 个） 【Ver.2.32 以上版本】

1.5.2 动作一览

所谓动作是指按钮 "被按下"、"被放开" 等的动作。

在下表中列出动作的一览。

注：可以使用的动作，根据零部件的种类的不同而不同。

动作名称	动作内容
CLICKED	零部件被按下
RELEASED	零部件被放开
TIMER	经过了间隔时间
REFRESH	画面被刷新
INITIALIZE	打开了可以初始化的操作画面　　【Ver.2.32 以上版本】
DONE	[OK] 被按下　　【Ver.2.32 以上版本】

1.5.3 动作指令的格式

动作指令通过 "获取 / 变更零部件的状态" 和 "操作盘控制指令" 的组合来记述。

注：可以使用的动作，根据零部件的种类的不同而不同。

(1) 获取 / 变更零部件的状态

零部件的参照以［零部件名称．属性］的格式进行记述。

例 1： 获取无线电按钮 (RadioBtn) 的 ON / OFF 状态。

```
DEFINT iState
IState = RadioBtn.State
```

例 2： 将按钮 (Button) 宽度设定为 200。

```
Button.Width = 200
```

(2) 操作盘控制指令

操作盘控制语言的语法请参照 "第4章"，指令一览请参照 "5.1项"。

1.5.4 向控制器发送数据

利用WINCAPSⅢ将创建的操作盘数据发送给控制器。
此时，WINCAPSⅢ将对已保存的操作盘数据进行编译，并发送。在发送数据之前，请“保存”正在编辑的操作盘数据。

1.5.5 限制事项

无线电按钮的状态 (State) 请仅对默认钮打开。
在操作盘编辑程序中，虽然可以使多个无线电按钮打开，但是若维持该状态向控制器发送，"教导器操作盘画面" 也会显示为多个打开。

第2章 用操作盘编辑程序创建操作盘

在第1章中就被编入到WINCAPSIII的操作盘编辑程序的功能进行了说明。
在本章中将对使用该操作盘编辑程序，实际在多功能教导器上创建操作盘的详细内容进行说明。

在机械手的多功能教导器上实现具有可以指定任意的大小、位置、颜色的各种零部件的通用操作盘。画面设计在电脑上通过鼠标操作进行。
被设计的画面数据（画面数为多个）1个文件夹中只能有1个数据。
在被创建的画面数据中被存放有多个的画面信息和被粘贴在各自画面的零部件（对象）信息。
设定画面设计功能、零部件属性（大小、位置、颜色等）请参照 "第1章 操作盘编辑程序"。

2.1 多功能教导器的设定

2.1.1 操作盘功能的有效化

因为操作盘功能是可选功能，所以需要事先从多功能教导器的基本画面按以下的步骤将功能有效化。

步骤 1

选择［F6 设定］-［F7 选项］-［F8 扩展功能］-［F5 功能追加］，使功能扩展画面显示。

步骤 2

请输入密码 "1453"。

若按下 "OK"，操作盘功能则被追加。

步骤 3

在基本画面上可以确认F5 "各个" 切换到 "操作盘"。

由此，按下［F5：操作盘］，则操作盘画面可以启动。

注：当选择了操作盘功能时，分配给F9的TP简易操作盘功能（RC5兼容的操作盘）将不能使用。

2.1.2　操作盘的启动设定【Ver.2.32 以上版本】

注：【Ver.2.31之前版本】请参照 "2.1.3　操作盘画面的自动显示设定"。

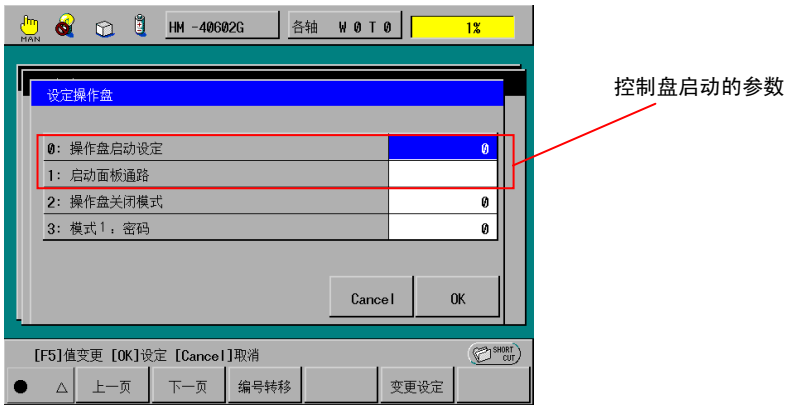
在控制器启动时显示操作盘，或按下多功能教导器的［F5 操作盘］进行显示等，操作盘的启动方法可以如下表所示的4种类型进行设定。

操作盘启动 设定参数	启动面板 路径的指定	操作盘的启动（操作画面的显示）	备考（用途等）
0	—	在控制器启动时不打开操作盘，在按下［F5 操作盘］时打开程序一览的当前文件夹（注）的操作盘。	
1	要	在控制器启动时打开用［启动面板通路］指定的操作盘，或在按下［F5 操作盘］时打开程序一览的当前文件夹的操作盘。	在控制器启动时，想从最初就显示操作盘时
2	要	在控制器启动时不打开操作盘，在按下［F5 操作盘］时打开用［启动面板通路］指定的操作盘。	决定了最初想要打开的操作盘，不进行文件夹的移动而想显示操作盘时
3	要	在控制器启动时打开用［启动面板通路］指定的操作盘，或在按下［F5 操作盘］时打开用［启动面板通路］指定的操作盘。	

注："程序一览的当前文件夹" 是指在程序一览画面上被显示的当前的文件夹。

■设定方法

步骤 1 若按下［F6：设定］→［F7：选项］→［F9：操作盘］，则显示下一个画面。



步骤 2 ［0：操作盘启动设定］的参数由上一页的表在0～3的范围内进行设定。

步骤 3 ［0：操作盘启动设定］的设定为1～3时，指定［1：启动面板通路］。
表示被使用的路径（文件夹的位置）。

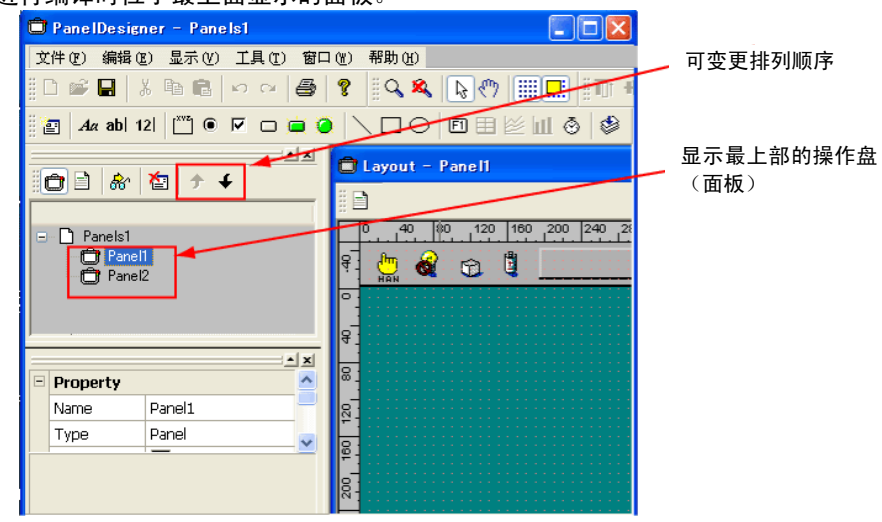
注：启动面板路径的末尾不带＼。

启动面板路径的示例：＼TEST

以＼为分割记述通过 (Pass)。在该示例中，将打开位于TEST文件夹中的操作盘。

当［1：启动面板通路］未被设定时，表示路径（最上层的文件夹）。
路径以空白来指定。（例：＼）

注意1：［1：启动面板通路］可以设定的只是路径。在定义有多个面板时，控制器启动时的最初的操作盘显示会是用PanelDesigner进行编译时位于最上面显示的面板。



注意2：在执行PAGE_CHANGE指令（从某个面板向其他的面板移动的指令）时，当在 "启动面板路径（注）" 内移动时，关闭操作盘再次打开操作盘时，将会记忆移动过的面板，显示最后移动过的面板。
在用PAGE_CHANGE指令向 "启动面板路径（注）" 以外的面板移动后的状态下，当关闭操作盘再次打开操作盘时，与控制器启动时的最初控制盘的显示时相同，在 "启动面板路径（注）" 内的操作盘会被显示。

注：或根据设定是 "程序一览的当前文件夹"

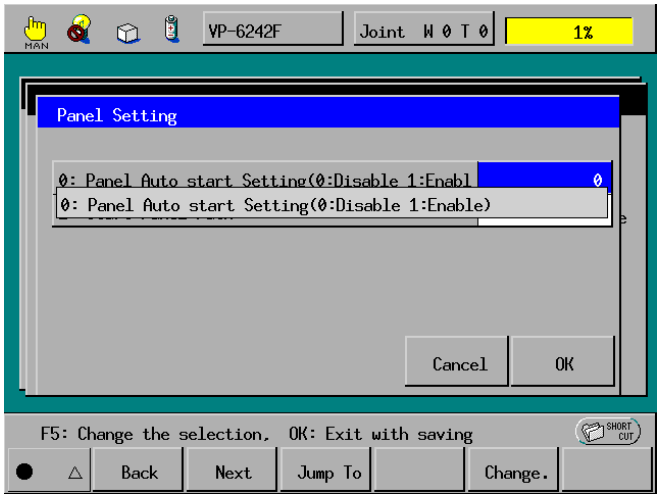
2.1.3　操作盘画面的自动显示设定【Ver.2.31 之前版本】

> 注：【Ver.2.32以上版本】请参照 "2.1.2　操作盘画面的启动设定"。

为了使操作盘画面在控制器启动时自动显示，请用多功能教导器进行以下的设定。

步骤 1

若按下［F6：设定］→［F7：选项］→［F9：操作盘］，以下的画面会被显示。



步骤 2

请将 "操作盘自动启动设定" 设定为［1：有效］。
并且请将具有想要启动画面的文件夹设定为"启动面板通路"（参照上一页的步骤 3）。
通过进行以上步骤在下一次启动时会进行自动显示。

注意：当错码被输出到教导器上时，该功能无效。

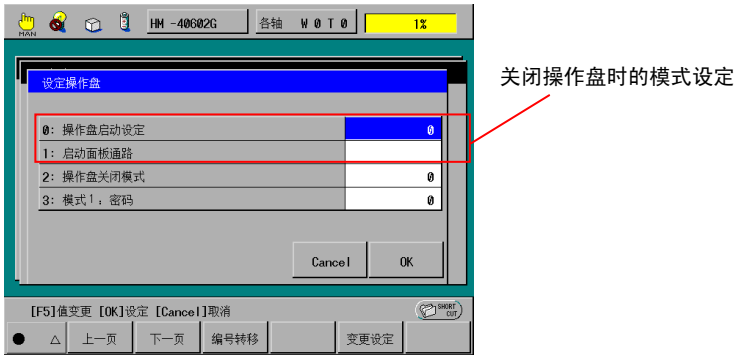
2.1.4　操作盘关闭模式的设定【Ver.2.32 以上版本】

操作盘关闭模式如下表所示进行了设定。
另外，默认被设定为 "SHIFT + CANCEL"。

操作盘关闭模式	内容
0: "SHIFT + CANCEL"（默认）	用 "SHIFT + CANCEL" 关闭操作盘画面。
1: "SHIFT + CANCEL + 密码"	在操作 "SHIFT + CANCEL" 之后，输入密码关闭操作盘画面。 密码用［3：模式1：密码］进行指定。
2: "CANCEL"	仅用 "CANCEL" 关闭操作盘画面。

■设定方法

步骤 1　若按下［F6：设定］→［F7：选项］→［F9：操作盘］，以下的画面会被显示。



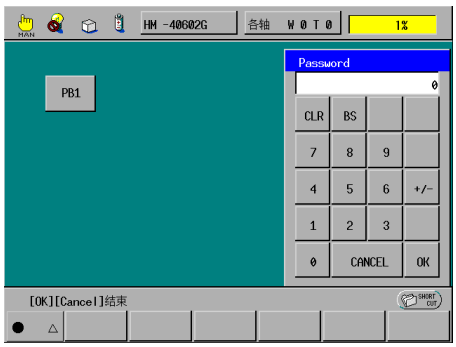
步骤 2　从以下中选择并设定［2：操作盘关闭模式］。

0："SHIFT + CANCEL"
1："SHIFT + CANCEL + 密码"・・・需要步骤 3
2："CANCEL"

步骤 3　(1)　［2：操作盘关闭模式］的设定为1（SHIFT + CANCEL + 密码）时，请对［3：模式1：密码］进行密码的输入设定。

注：密码的范围是-2147483648～2147483647

(2) 用 "SHIFT + CANCEL" 显示密码输入画面。



(3) 若密码与用 (1) 设定的一致，则关闭操作画面。

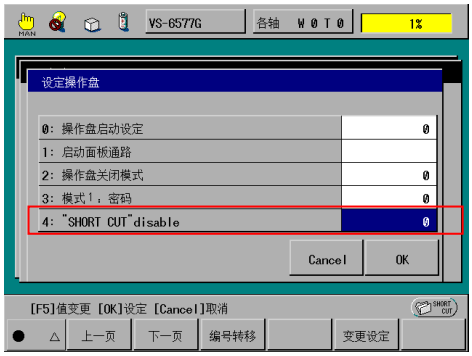
注：当忘记密码时，若输入 273958314 可以关闭操作盘。

2.1.5　快捷按钮的非显示设定【Ver 2.6 以后】

为了避免在显示操作面板时误按快捷按钮，可以进行非显示。

■设定方法

步骤 1　按［F6：设定］→［F7：选件］→［F9：操作面板］，显示［操作面板设定］画面。



步骤 2　选择[4：“SHORT CUT无效化”]，按[F5：设定变更]，变更参数。如果按[OK]，设定即变为有效。

[操作面板设定 4：“SHORT CUT无效化”]参数

参数	设定内容	备考
0	在显示操作面板时显示快捷按钮	默认
1	在显示操作面板时非显示快捷按钮	

2.2 操作盘编辑程序的各个零部件的使用方法

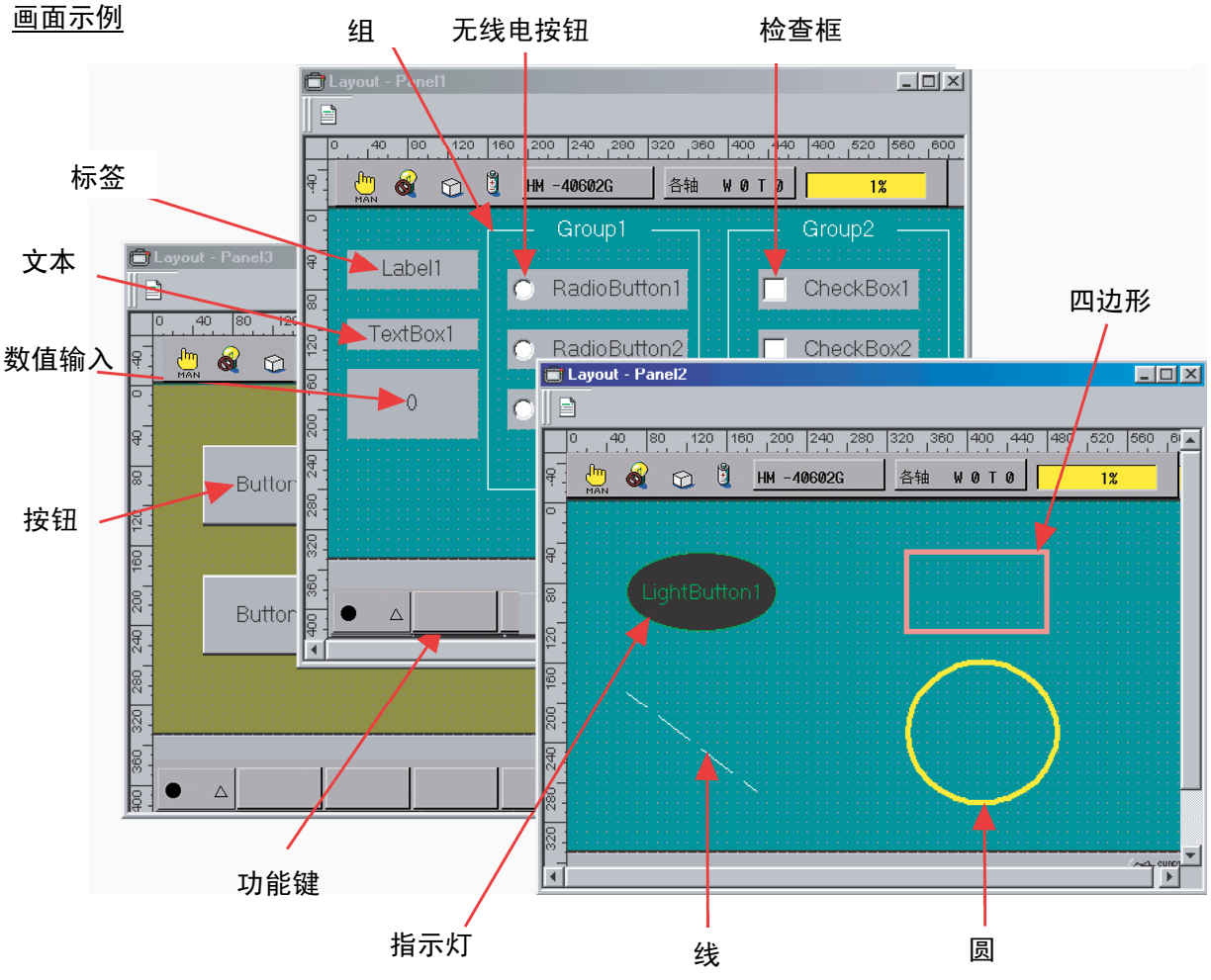
2.2.1 操作盘编辑程序的零部件与功能的一览

可以配置在画面的零部件为下表中的14种。

操作盘编辑程序的零部件

	零部件	功能
1	按钮	作为按压按钮动作。
2	标签	显示字符串。
3	数值输入	通过小数字键盘进行数值输入。
4	文本	通过键盘进行字符串输入。
5	组	作为进行无线电按钮的排他的组指定。
6	无线电按钮	通过组进行排他选择。
7	检查框	进行项目选择。
8	指示灯	作为ON / OFF指示灯动作。
9	线	配置直线。
10	四边形	配置正方形、长方形。
11	圆	配置圆、椭圆。
12	功能键	将功能键 (F1～F12) 作为按压按钮使用。
13	计时器	是可以指定按一定时间单位执行操作的零部件。
14	照明式按钮	是动作作为按钮、显示作为指示灯动作的按钮。

画面示例



2.2.2 零部件的动作的指定

指定操作盘画面零部件的动作（被按下时等的动作）。
进行在操作盘编辑程序画面中按钮被按下时的动作记述、属性变更、获取。

(1) 动作记述形式

动作记述用以下形式仅对想使其进行的动作进行记述。

```
DEF　对象名__动作
 想使其进行的动作
END
```

在操作盘编辑程序中若选择对象名与动作，"DEF　对象名__动作" 和 "END" 的部分因会被自动生成，所以请仅对内部进行记述。
行动里有下表所示的项目，可以使用的动作会根据零部件的种类的不同而不同。

动作名称	动作内容
CLICKED	零部件被按下
RELEASED	零部件被放开（参照 2.2.3 项）
TIMER	经过了间隔时间
REFRESH	画面被刷新
INITIALIZE 【Ver.2.32 以上版本】	打开了可以初始化的操作画面
DONE 【Ver.2.32 以上版本】	[OK] 被按下

(2) 动作记述内容

想使其进行的动作会根据操作盘控制指令和各自的对象的属性变更、获取被进行记述。对象的状态可以以 "对象．属性" 的形式进行变更、获取。

关于可以变更的属性，请参照 "3.3.4　操作盘零部件对象变量" 。
可以使用的变量是I、F、D、S型全局变量和局部 (Local) 变量、文件夹变量。

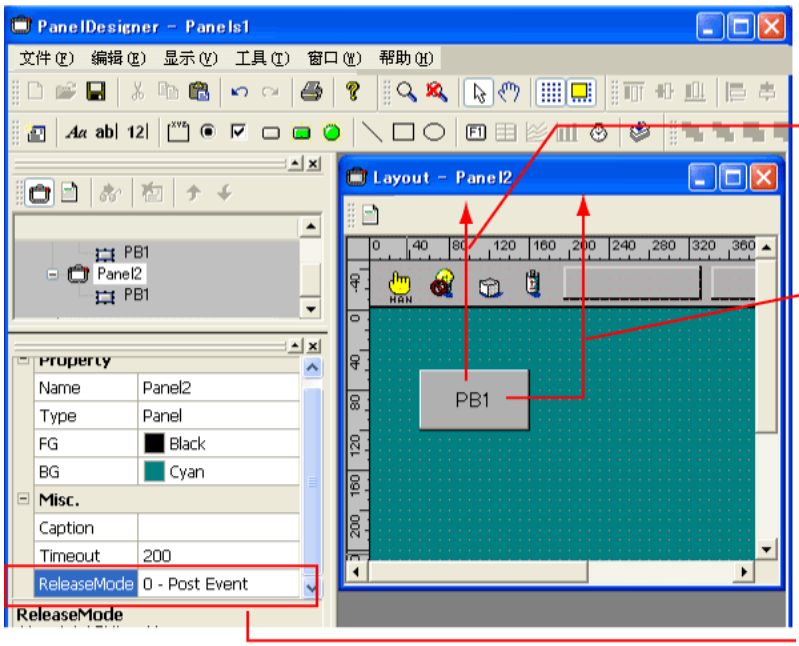
2.2.3　RELEASED 动作发生条件的设定【Ver.2.32 以上版本】

2.2.3.1　Release-Mode 的追加

通过在属性画面中追加Release-Mode，可以设定RELEASED动作发生条件。
在操作盘属性画面的Release-Mode中，可以选择设定以下内容。

属性画面的设定选择	设定内容	备注
0-PostEvent	即使在零部件之外手指放开时也发生RELEASED动作	Ver.2.32以上版本的默认设定
1-No Event	仅在零部件内手指放开时发生RELEASED动作	Ver.2.31以前版本固定于该设定

RELEASED行动发生条件的设定



从零部件被按下的状态切换至零部件内后松开手指。
（1-No 在 Event 设定时）

按下零部件的状态下挪动手指，在零部件外松开手指。
（0-Post 在 Event 设定时）

Release-Mode 设定的属性画面

注：每个面板文件有1个Release-Mode的参数。变更了该参数后，其会被面板内的所有具有RELEASED动作的零部件所适用。

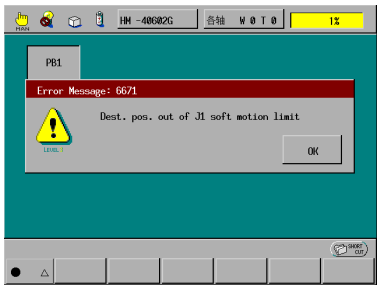
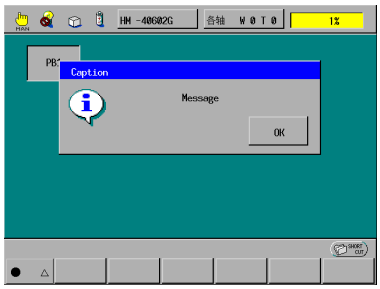
2.2.3.2 RELEASED 动作的注意事项

在当前的操作盘画面上显示其他的画面时RELEASED动作将不被执行。
RELEASED动作不被执行的条件有以下3种，用Push按钮的示例进行说明。

■ RELEASED动作不被执行的条件

在Push按钮被按下的状态时若发生以下的情况，即使手指离开教导器画面RELEASED动作也不被执行。

(1) 发生错误
(2) 用Printmsg指令显示信息
(3) 用Page_change指令切换操作盘画面

按下Push按钮的状态	RELEASED动作不被执行的条件
	(1) 发生错误 
	(2) 用Printmsg指令显示信息 
	(3) 用Page_change指令切换操作画面 （用Timer执行Page_change） 

注：当上述的事例成为像下一页中的问题示例时，请利用程序采取回避措施。

■ 需要回避措施的程序示例

仅在按下Push按钮期间，有IO [128] 打开（ON期间外部设备在动作）的下述的程序时，需要回避措施。

仅在按下Push按钮期间，IO [128] 打开	程序示例
	DEF PB1_CLICKED () set IO [128] END DEF PB1_RELEASED () reset IO [128] END

■ 错误发生时的回避方法的示例

(1) 回避方法 1

在进入特权任务、特权任务扩展的状态下使如下的特权任务程序运行且发生错误时，进行控制使之位于安全一侧（IO [128] 为OFF）。

```
Program TSR1
        DEFINT ERRCODE
INITWAITERR                         '错误检测的初始化
            WHILE 1
            ERRCODE = WAITERROR                 '等待到错误发生。
            IF GETERRLVL (ERRCODE) > 1          '发生的错误大于等级 1 时处理
              RESET IO [128]                    '控制到安全一侧。
              INITWAITERR                       '错误检测的初始化
            ENDIF
        WEND
END
```

(2) 回避方法 2

追加通过特权任务监视双重安全开关，仅在双重安全开关被按下时，IO [128] 可以打开，若放开双重安全开关，则IO [128] 关闭的处理。
由此错误发生时只要放开双重安全开关则可以控制在安全一侧 (IO [128] OFF)。

(3) 回避方法 3

变更为在按下Push按钮时，仅一定间隔期间IO [128] 打开、放开时不做任何的处理。此时是微动的处理。

注意：关于PRINTMSG、PAGE_CHANGE指令，也变更为IO [128] 打开、放开时不做任何的处理。
关于特权任务、各个指令的使用请参照相应手册。

2.2.4 "INITIALIZE" 动作【Ver.2.32 以上版本】

变为打开操作盘画面时，在各个面板只能定义1个被调出的初始化的动作"INITIALIZE"。用于画面结构的初始化等。
操作盘初始化动作被调出的时机是在以下情况。

(1) 用［F5 操作盘］打开了操作盘画面时
(2) 用［控制器启动时的自动启动］打开了操作盘画面时
(3) 用［PAGE_CHANGE指令］切换了操作盘画面时

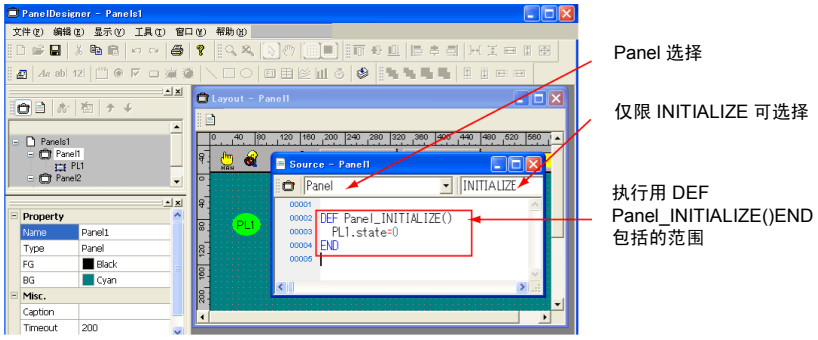
■设定方法

步骤 1

用操作盘若选择Panel，仅可以选择INITIALIZE动作，若选择INITIALIZE，

```
DEF Panel_INITIALIZE ()
END
```

则被自动生成。



步骤 2

记述操作盘初始化动作的内容。

注意：在该记述中不能使用 "Page_Change" 指令。

2.2.5 DONE 动作【Ver.2.32 以上版本】

在操作盘的零部件 "数值输入框" 和 "文本框" 上追加了DONE动作。所谓DONE动作是指在数值输入框，文本框中按下 "OK" 时被执行的动作。

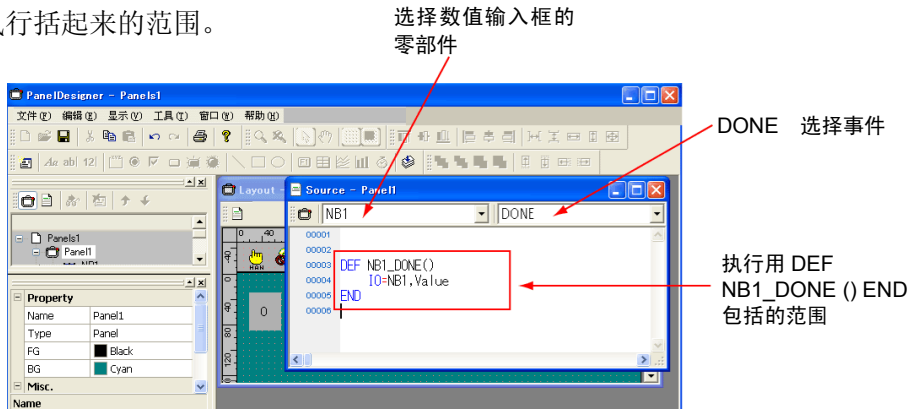
■设定方法（数值输入框的示例）

步骤

选择数值输入框的零部件，并选择DONE动作。

```
DEF NB1_DONE ()
END
```

执行括起来的范围。



如同上例的

```
IO = NB1.Valve
```

按下 [OK] 时， 被设定的零部件的属性的值会设定到全局变量，此动作在这样的用途中使用。

2.2.6　各个零部件的使用方法

(1)　按钮

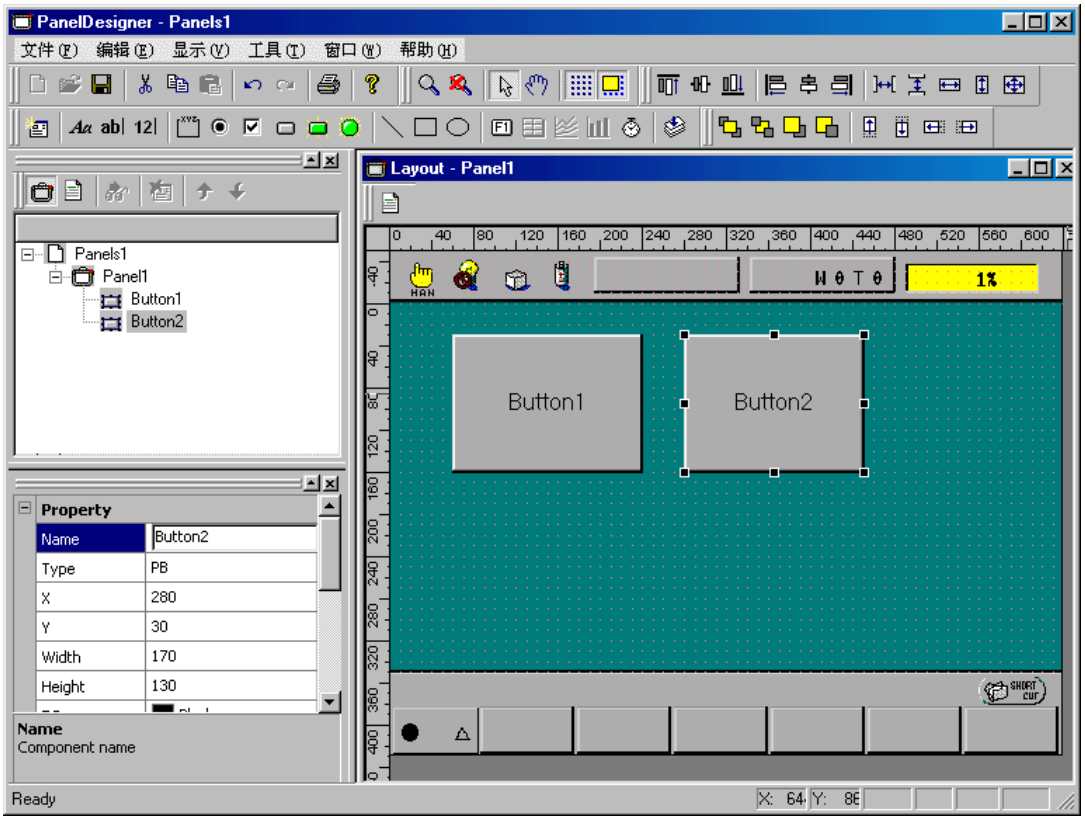
按钮是对于按下 / 放开进行动作的零部件。
在此将对创建在被按下时打开IO24号，被放开关闭的按钮和在被按下时启动程序 (Sample pro) 的按钮的方法进行说明。

■按钮创建的示例

步骤 1

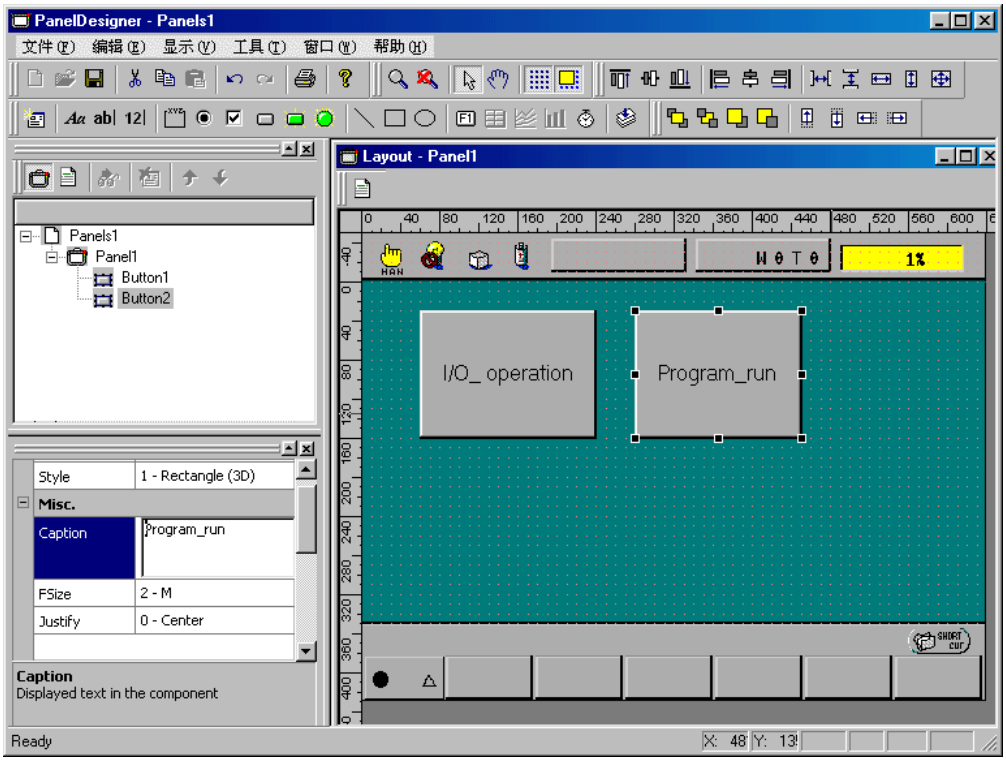
用操作盘编辑程序配置2个按钮。

配置的位置只要是在教导器画面上哪里都可以自由放置。按钮上的显示文字分别为 "IO操作"、"PRO启动"。
由于零部件均通过从自身或同一面板上的其他零部件进行参数变更和状态获取，所以按钮及其以外的所有零部件都通过零部件名被识别。
当为按钮状态时，"Button" + "数字" 为默认值，可以对其进行任意变更。
在此，以默认 ("Button1"、"Button2") 的状态继续操作。



步骤 2

按钮上的显示文字请用各自的按钮的参数 "Caption" 进行变更。

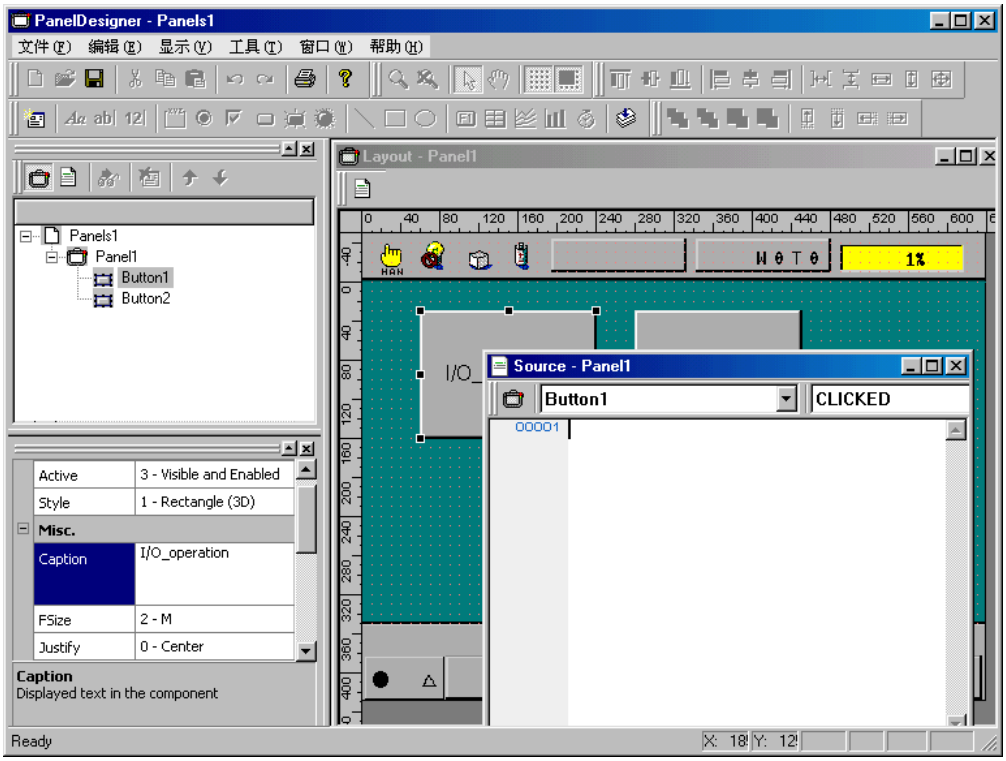


步骤 3

<按钮行动记述>

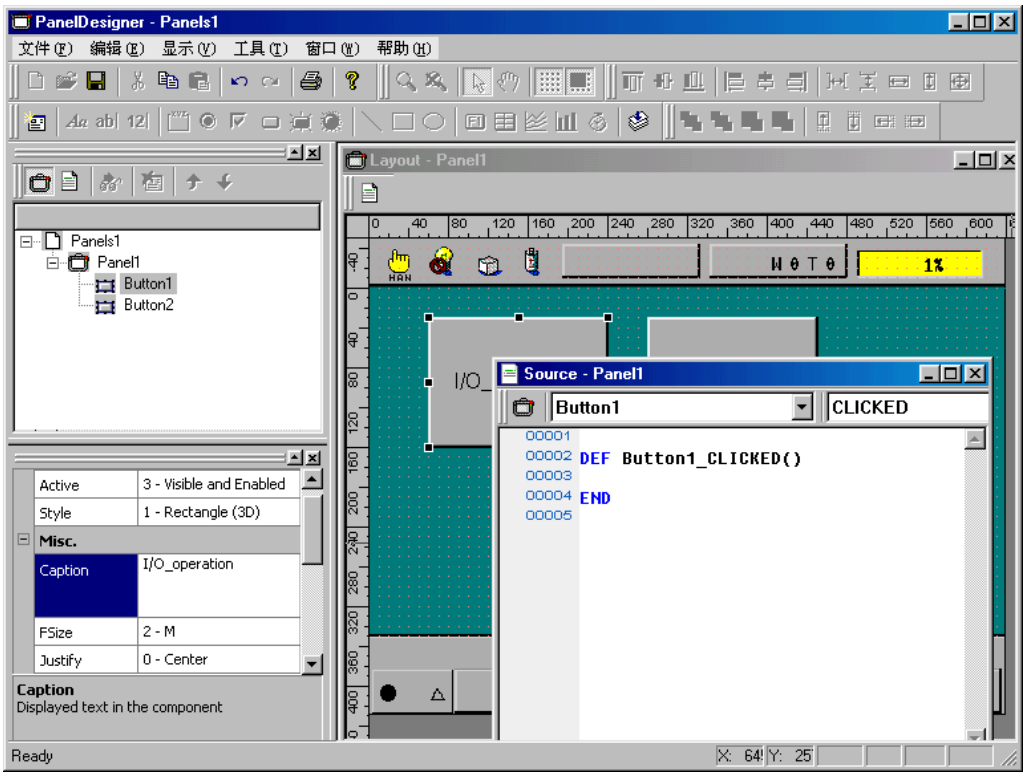
对各个按钮可以自由指定被按下 (Clicked) / 被放开 (Released) 时的动作。
记述按钮被按下时、被放开时的动作的示例如下所示。

在操作盘编辑程序上双击 "IO 操作" 按钮。之后源程序编辑画面被打开。



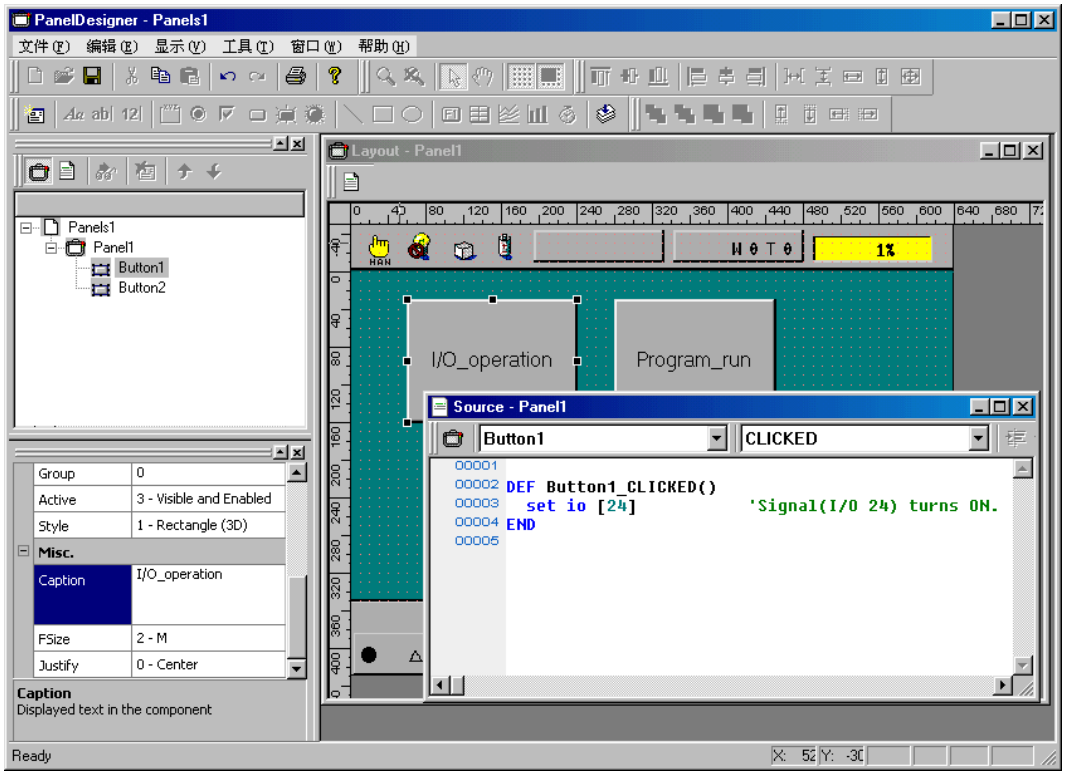
步骤 4

首先，记述在被按下时打开IO24号的程序。若从源程序编辑画面上部的下拉列表框中选择零部件名 "Button1"、动作 "Clicked"（被按下时），会在程序外部自动生成。



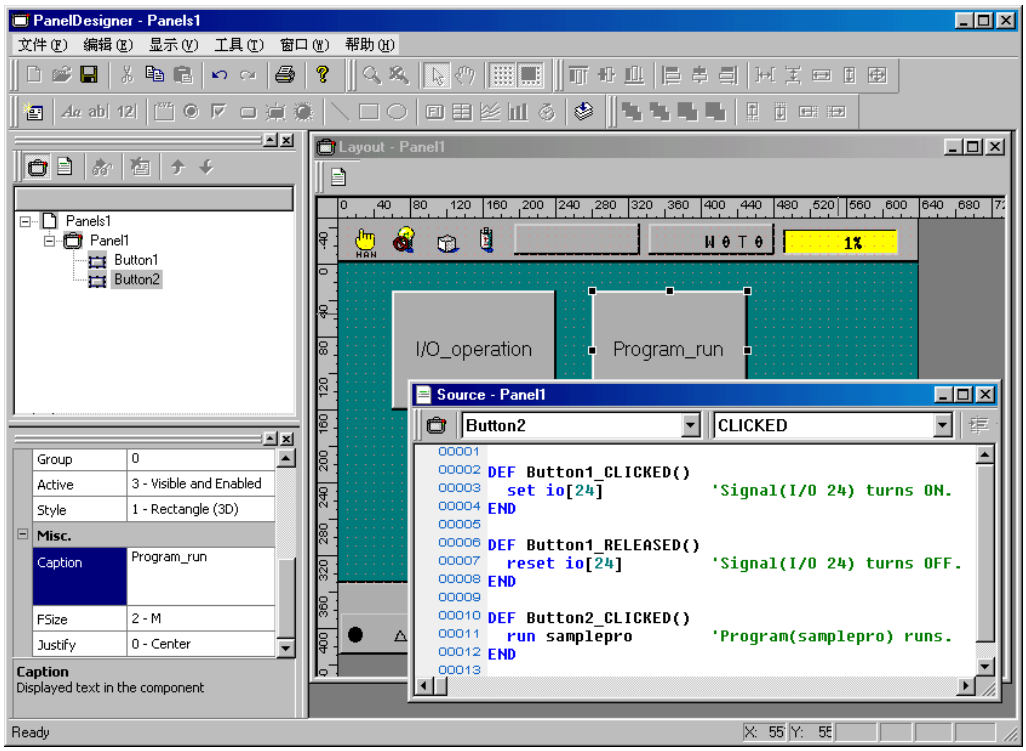
步骤 5

在其中记述动作。



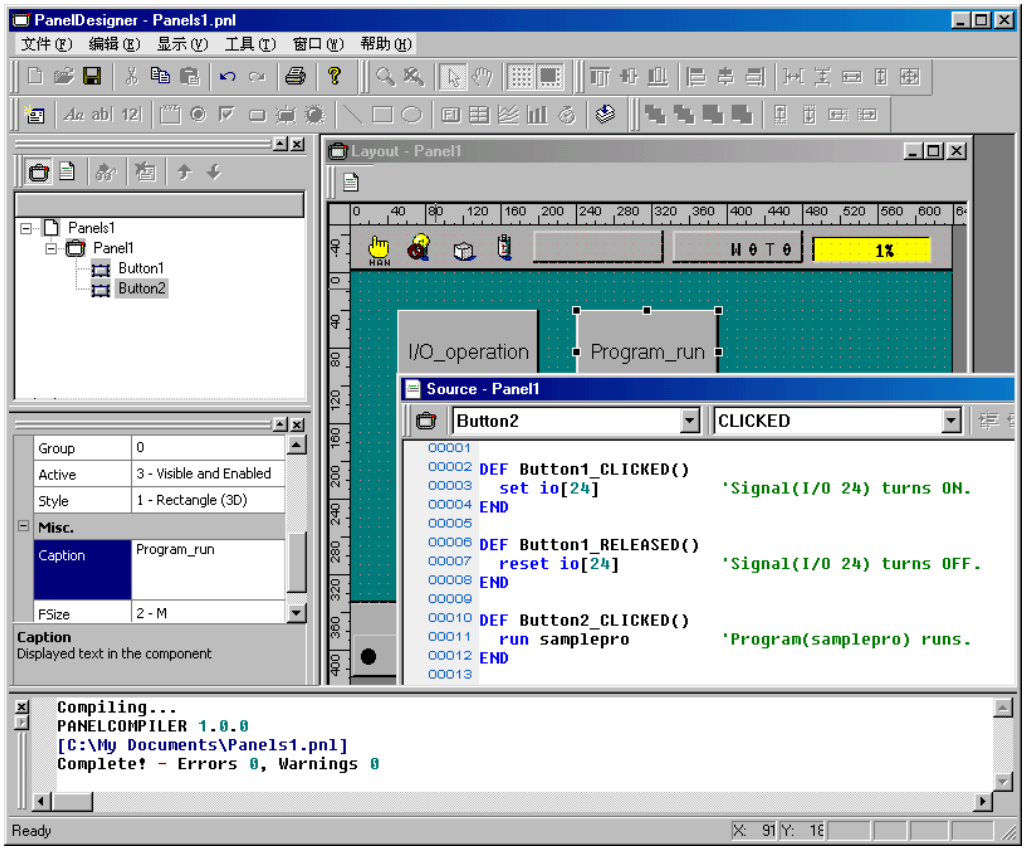
步骤 6

同样的，记述在被放开时关闭IO24号的程序（零部件名 "Button1"、动作 "Released"：被放开时）、在被按下时启动程序 (samplepro) 的程序（零部件名 "Button2"、动作 "Clicked"）。



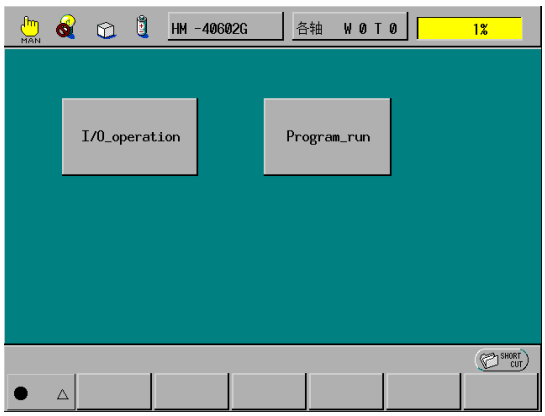
步骤 7

结束之后，请保留程序，并实施编译进行语法检查。



步骤 8

若编译成功，请用 WINCAPSⅢ向控制器进行转发。



步骤 9

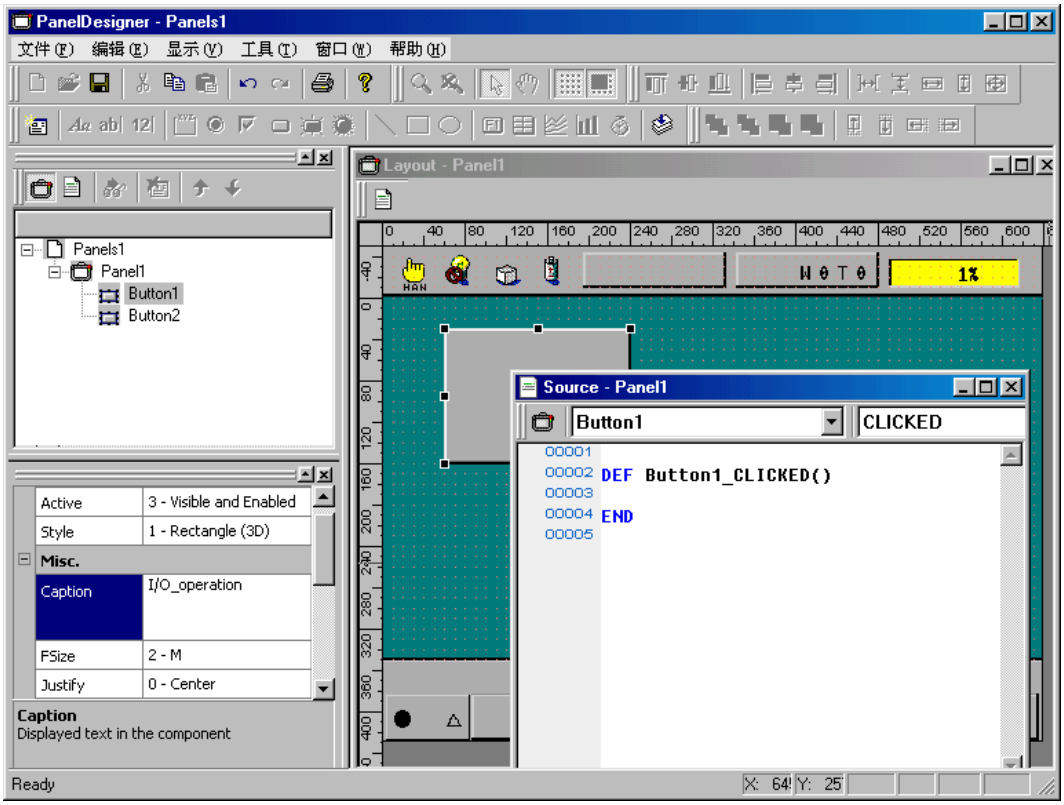
<按钮参数变更>

颜色、位置等的按钮参数可以使用零部件名从零部件自身（按钮）、同一面板上的其他零部件进行变更 / 获取。

按钮参数可以用 "零部件名．属性" 的形式进行访问。各个零部件的参数、可以取得的值的详细内容请参照 "3.3.4　操作盘零部件对象变量"。

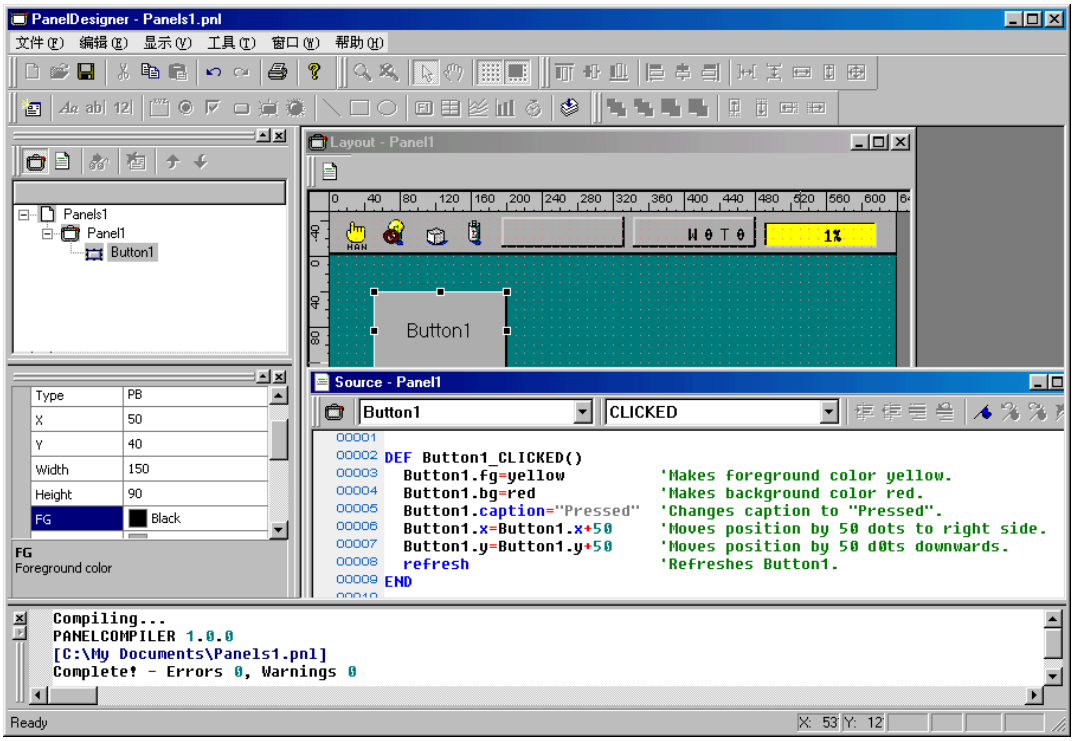
在此示出变更零部件的颜色（Fg：前景色、Bg：背景色）、显示文字 (Caption)、位置（X：横位置、Y：纵位置）的示例。

如刚才那样启动操作盘编辑程序，并请进行按钮的配置和源程序编辑画面的启动。



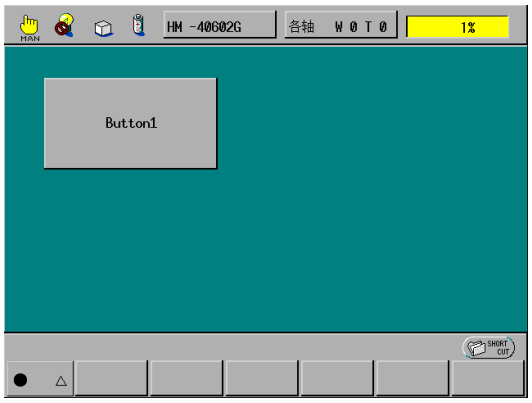
步骤 10

程序如下所示进行记述。

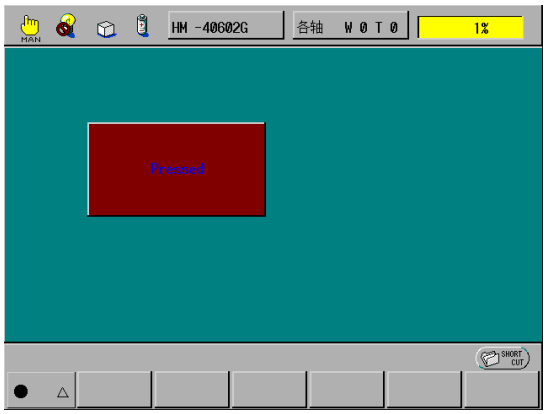


步骤 11

请与刚才一样保留、编译、向控制器进行转发。



按钮按下后的教导器图

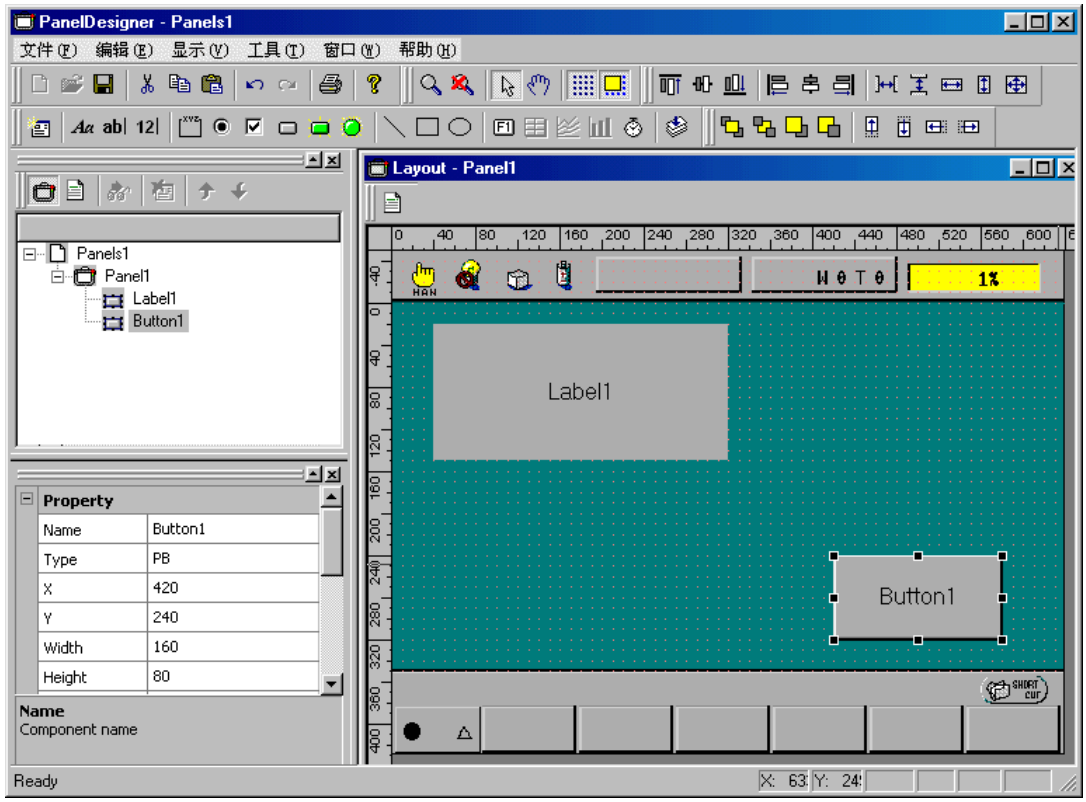


(2) 标签

标签是显示字符串的零部件，不能进行动作记述。以下为创建标签、在同一画面中的按钮被按下时变更该标签参数的示例。

■标签创建的示例

步骤 1 与按钮一样启动操作盘编辑程序，在画面上配置1个标签和1个按钮。



步骤 2

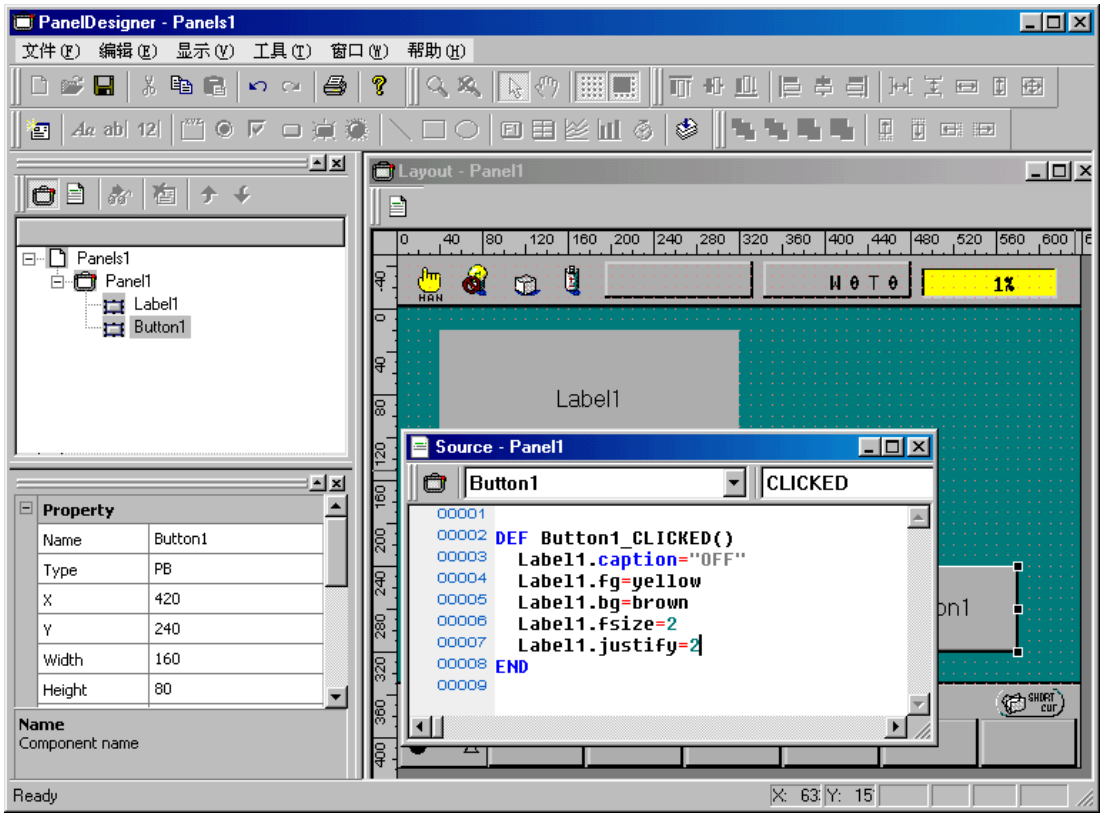
<标签参数变更>

变更各自标签的显示字符串、颜色、字体大小、字符位置。向参数的访问与按钮一样用"零部件名．属性"进行。

要将零部件名Label1的显示字符串变更为"消灯中"可记述为"Label1.caption = "消灯中"。同样要想将标签的前景色为黄色，背景色为茶色，字体大小为大，文字位置为居左，按照如下所示进行记述。

Label1.fg = yellow ：前景色为黄色
Label1.bg = brown ：背景色为茶色
Label1.fsize = 2 ：字体大小为大
Label1.justify = 2 ：文字位置为居左

将以上内容记述在源程序编辑画面上用Button1的Clicked被创建的程序之中。



步骤 3

若将其发送给控制器，按下按钮时被显示为如下。

按下按钮之前

按下按钮之后

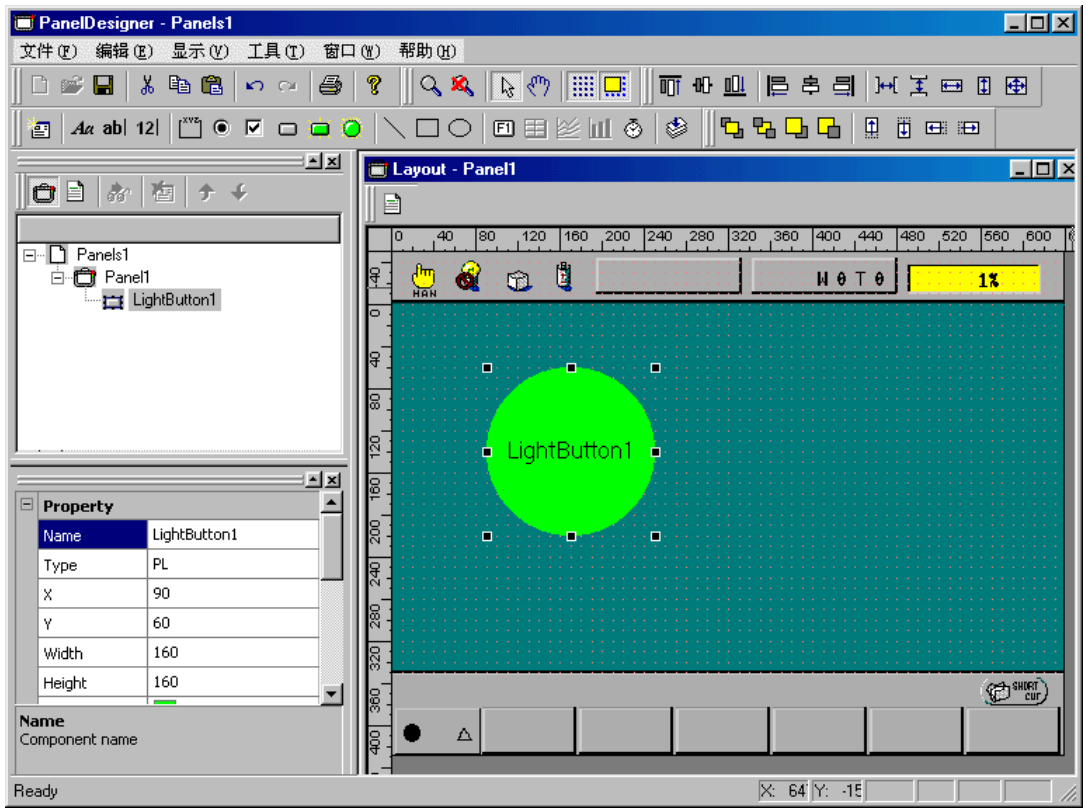
(3) 指示灯

指示灯是显示ON / OFF状态的零部件，定期性地进行动作 (refresh)。
使用该动作进行状态监视、显示。
在此示出监视I / O状态，并显示其状态的示例。

■指示灯创建的示例

步骤 1

与按钮一样启动操作盘编辑程序，在画面上配置 1 个指示灯。



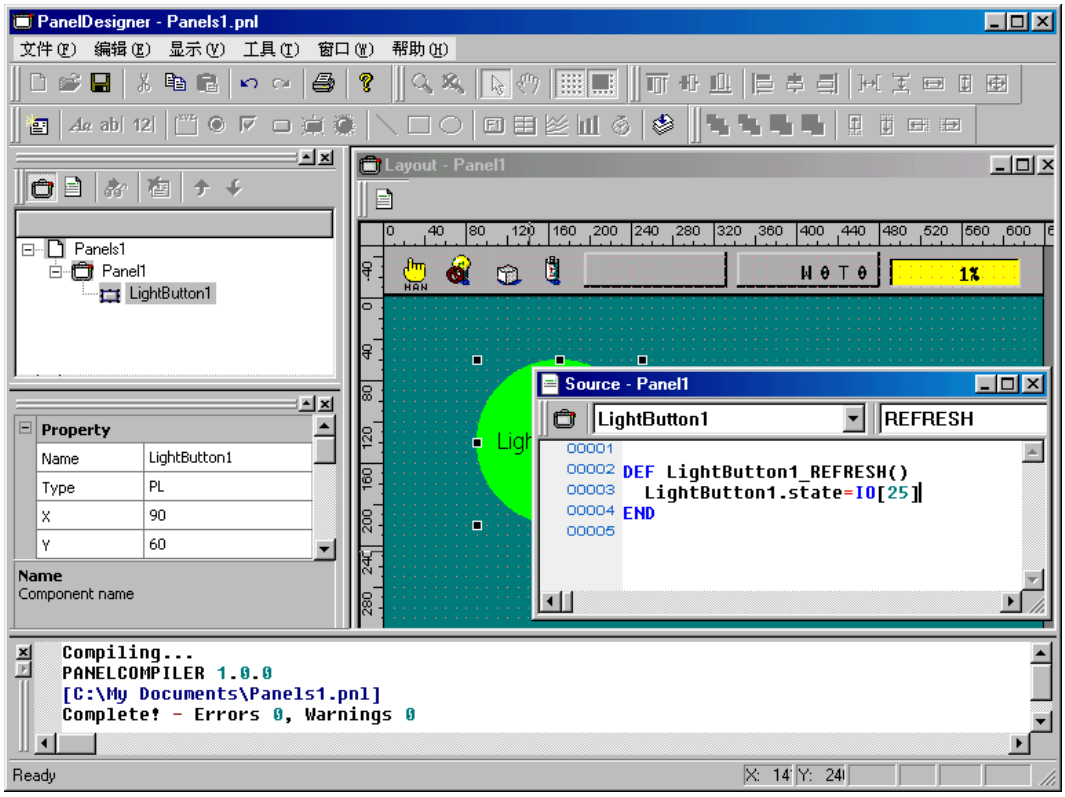
步骤 2

<指示灯动作记述>

指示灯会定期性地进行refresh动作。为了使该动作将I / O 25号的状态作为指示灯本身的状态进行显示，要在用源程序编辑画面上选择指示灯的refresh动作，在创建的程序内按照如下所示进行记述。

```
LightButton1.state = IO [25]
```

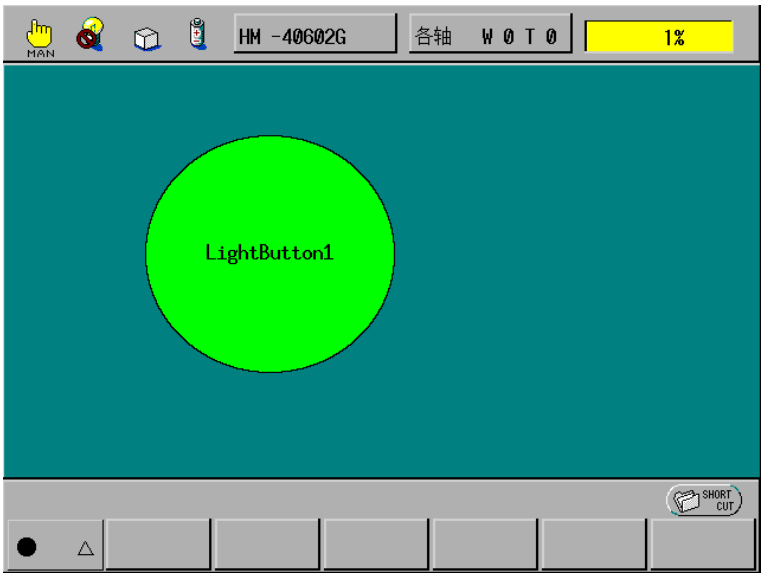
将指示灯的状态与IO [25] 的状态相一致。



步骤 3

若将其发送给控制器，则会显示如下。

亮灯时



熄灯时

步骤 4

<指示灯参数变更>

指示灯的参数和按钮一样是可以被访问的。

(4) 数值输入框

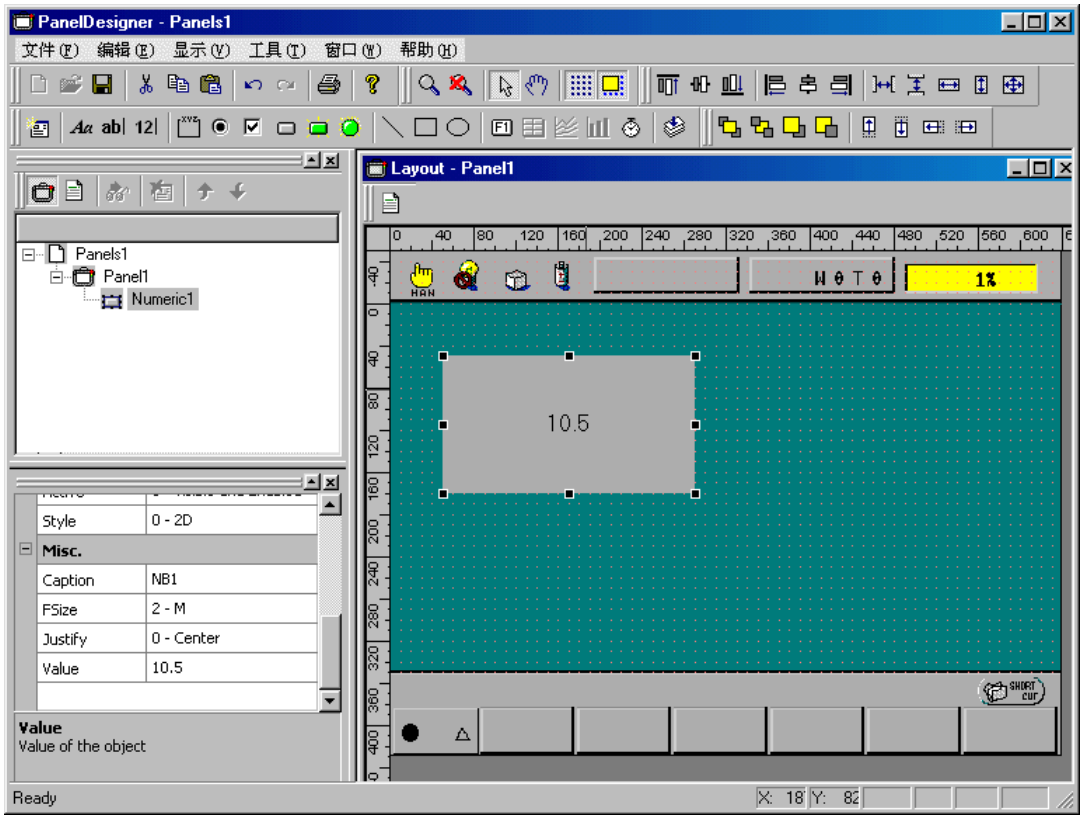
数值输入框是按钮本身带有实数值并显示该值的零部件。可以按下按钮显示数值输入小键盘，也可以从教导器操作画面上直接输入数值。此外与按钮相同，可以分别对被按下的动作，被放开的动作进行记述。

> 注：【Ver.2.32以上版本】，在数值输入框中被追加了当 [OK] 被按下时的 "DONE动作"。
> 请参照 "2.2.5 DONE动作"。

■数值输入框创建的示例

步骤 1

与按钮一样启动操作盘编辑程序，在画面上配置1个数值输入框。
接着设定数值输入框所具有的值的初始值（可省略）。



步骤 2

<数值输入框行动记述>

数值输入框所具有的动作和按钮一样可以进行记述。

步骤 3

<数值输入框参数变更>

数值输入框和按钮一样具有颜色、位置等，但作为该零部件特有的参数还具有数值（value：实数值）、样式（style：显示形式10进制、16进制等）。

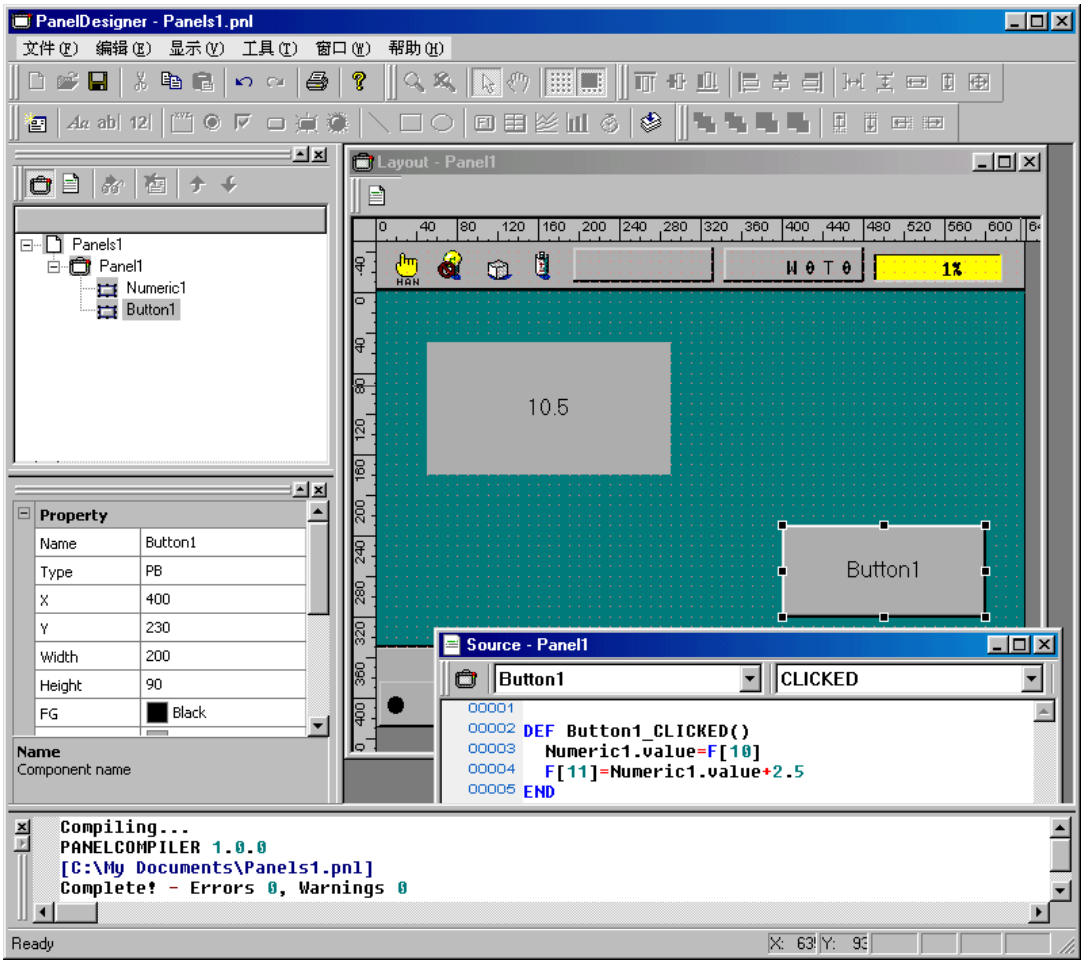
在此，示出当同一画面上的按钮被按下后，将F型全局变量10号的值输入到数值输入框的数值中，并将该值加上2.5后存放到F型全局变量11号中的示例。

用操作盘编辑程序创建数值输入框和按钮，打开源程序编辑画面，记述当按钮被按下时的处理内容。对参数的访问是和其他零部件一样。

选择Button1，Clicked，创建程序的外部，并按如下进行记述。

```
Numeric1.value = F [10]

F [11] = Numeric1.value + 2.5
```



(5) 文本框

文本框是按钮本身带有字符串并显示该值的零部件。可以按下按钮显示键盘，也可以从教导器操作画面上直接输入字符串。此外与按钮相同，可以分别对被按下的动作，被放开的动作进行记述。

> 注：【Ver.2.32以上版本】，在文本框中被追加了当 [OK] 被按下时的 "DONE动作"。
> 请参照 "2.2.5 DONE动作"。

■文本框创建的示例

步骤 1

与按钮一样启动操作盘编辑程序，在画面上配置文本框。接着设定文本框所具有的字符串的初始值（可省略）。

步骤 2

<文本框动作记述>

文本框所具有的动作和按钮一样具有Clicked，Released，所以可以分别就其处理进行记述。

步骤 3

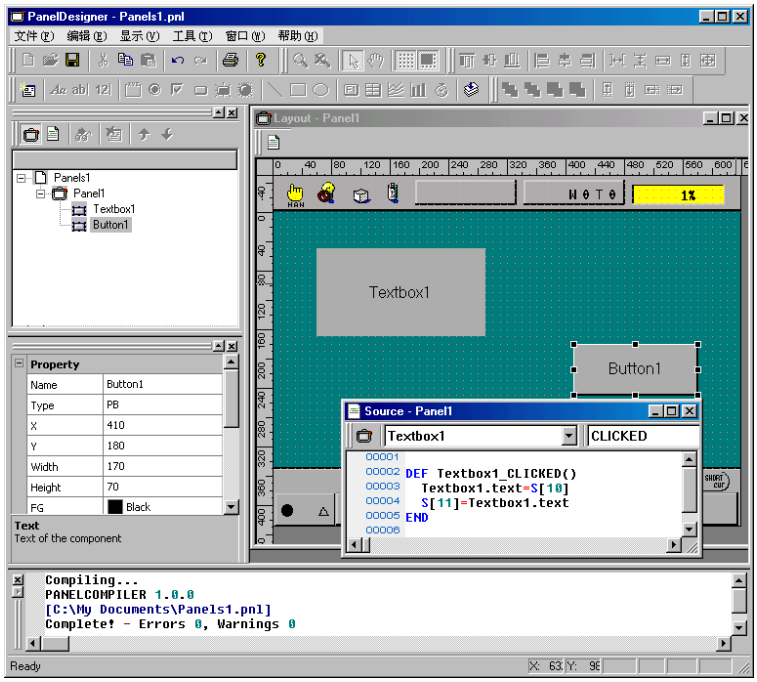
<文本框参数变更>

文本框和按钮一样具有颜色、位置等，但作为该零部件特有的参数还具有字符串（text：字符串值）。在此，示出当同一画面上的按钮被按下后，将S型全局变量10号的值输入到文本框中，并将该值再次存放到S型全局变量11号中的示例。

用操作盘编辑程序创建文本框和按钮，打开源程序编辑画面，记述当按钮被按下时的处理内容。对参数的访问是和其他零部件一样的。

选择Button1，Clicked，创建程序的外部，并按如下这样进行记述。

```
Textbox1.text = S [10]
S [11] = Textbox1.text
```



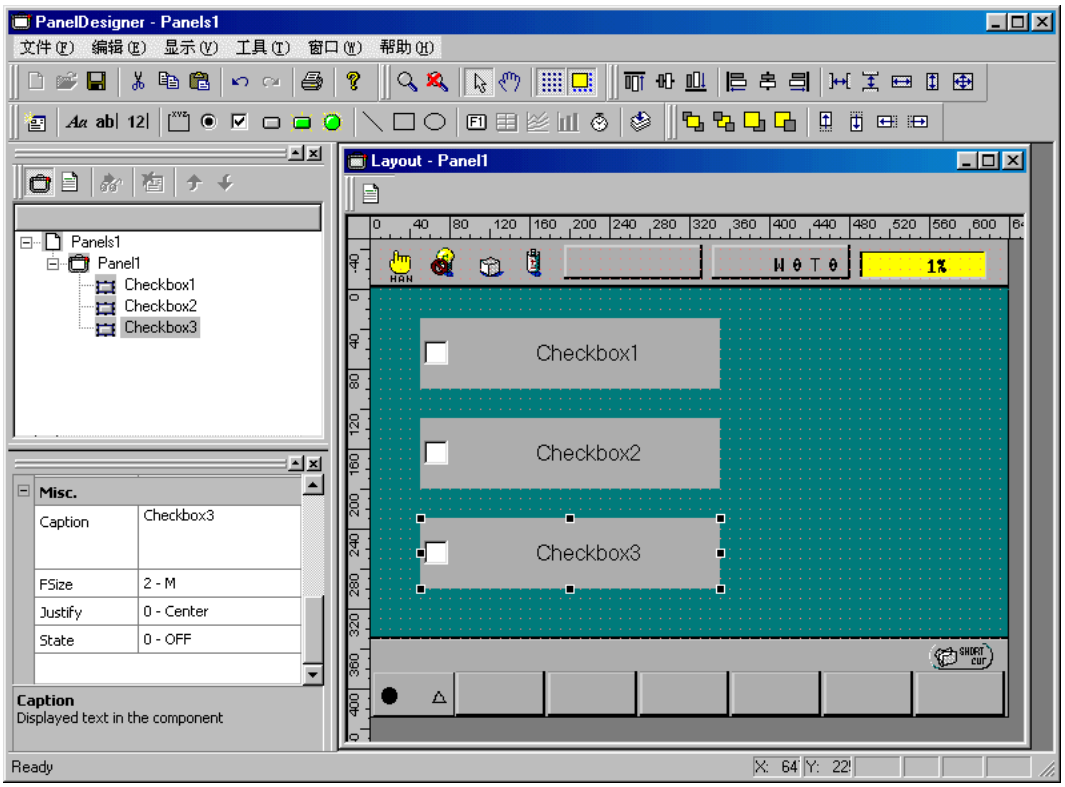
(6) 检查框

检查框是通过被按下切换ON / OFF状态的零部件。也可以作为触发按钮来使用。
此外与按钮相同，可以分别对被按下的动作，被放开的动作进行记述。
ON / OFF状态可以通过参数 (state) 来访问。

■检查框创建的示例

步骤 1

与按钮一样启动操作盘编辑程序，在画面上配置检查框。



步骤 2

<检查框动作记述>

检查框所具有的动作和按钮一样具有Clicked，Released，所以可以分别就其处理进行记述。

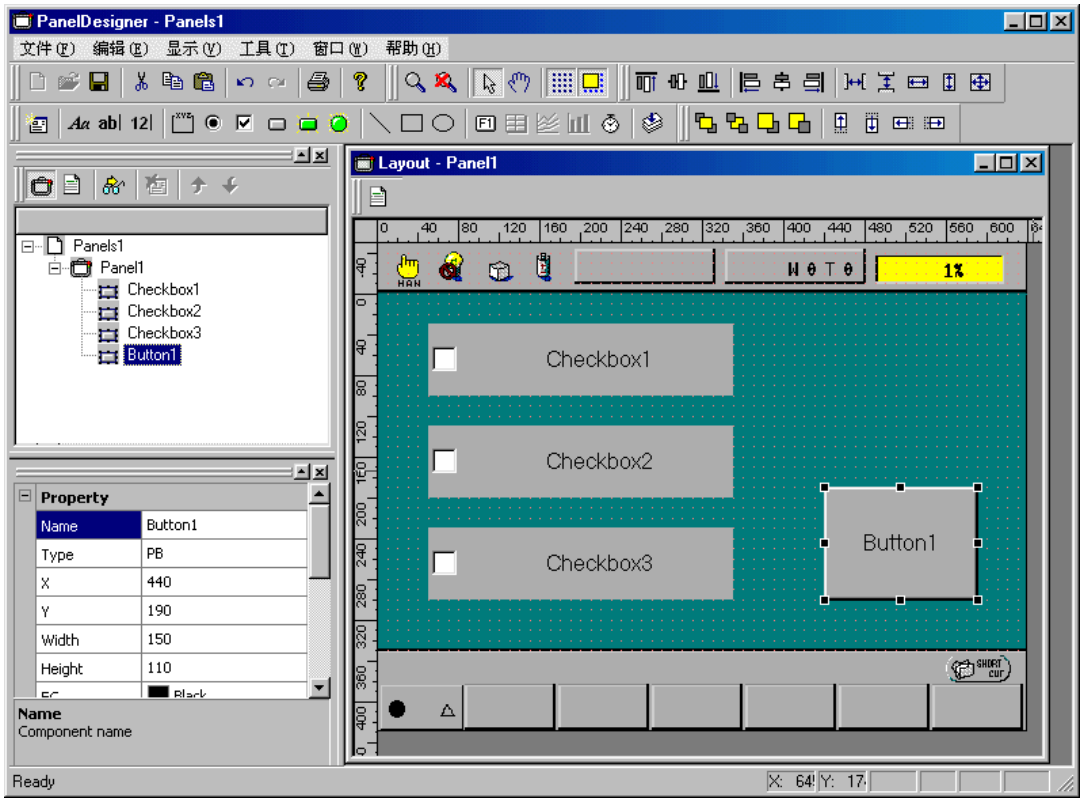
步骤 3

<检查框参数获取、变更>

检查框具有与按钮、标签一样的参数。

在此，记述当同一画面中的按钮被按下时，将检查框的状态（ON / OFF）输出到 IO [24]～IO [26] 的示例。

首先配置各个零部件。

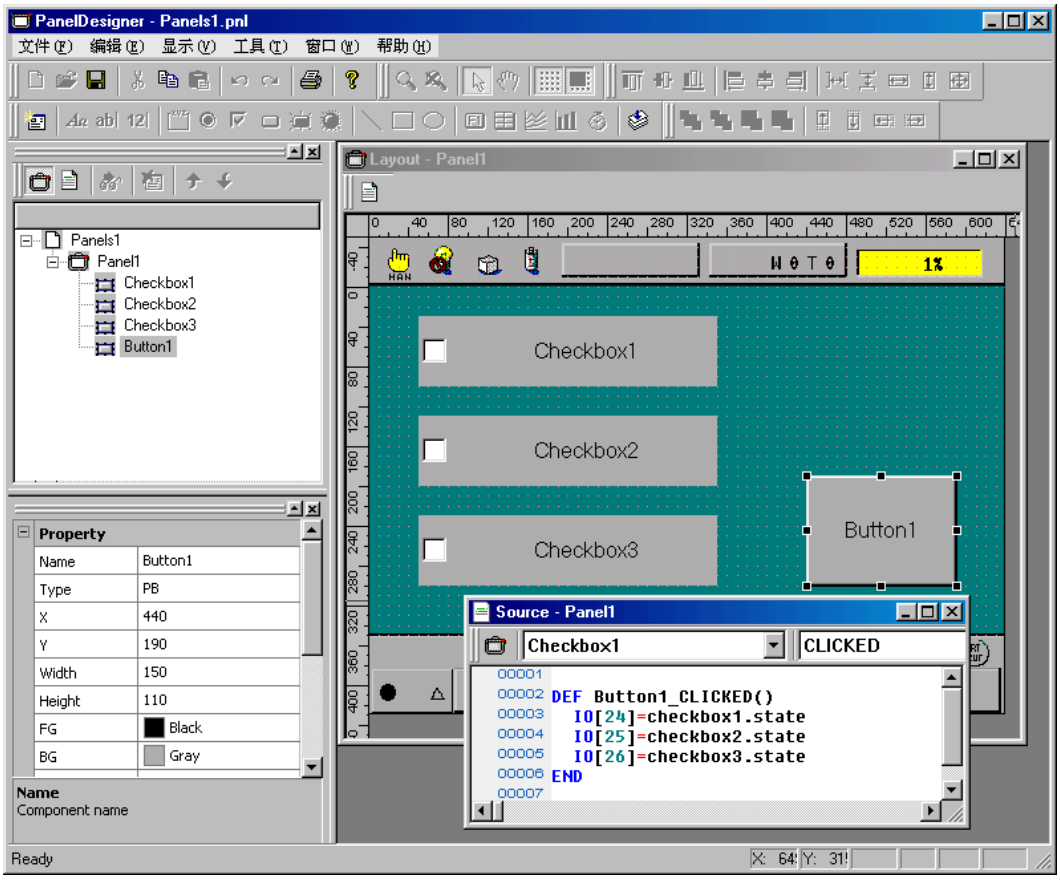


步骤 4

记述当按钮被按下时的处理（选择Button1，Clicked）。
用state参数获取检查框的状态的处理如下所示。

```
IO [24] = checkbox1.state
IO [25] = checkbox2.state
IO [26] = checkbox3.state
```

通过将其转发给控制器，可以实现上述的功能。



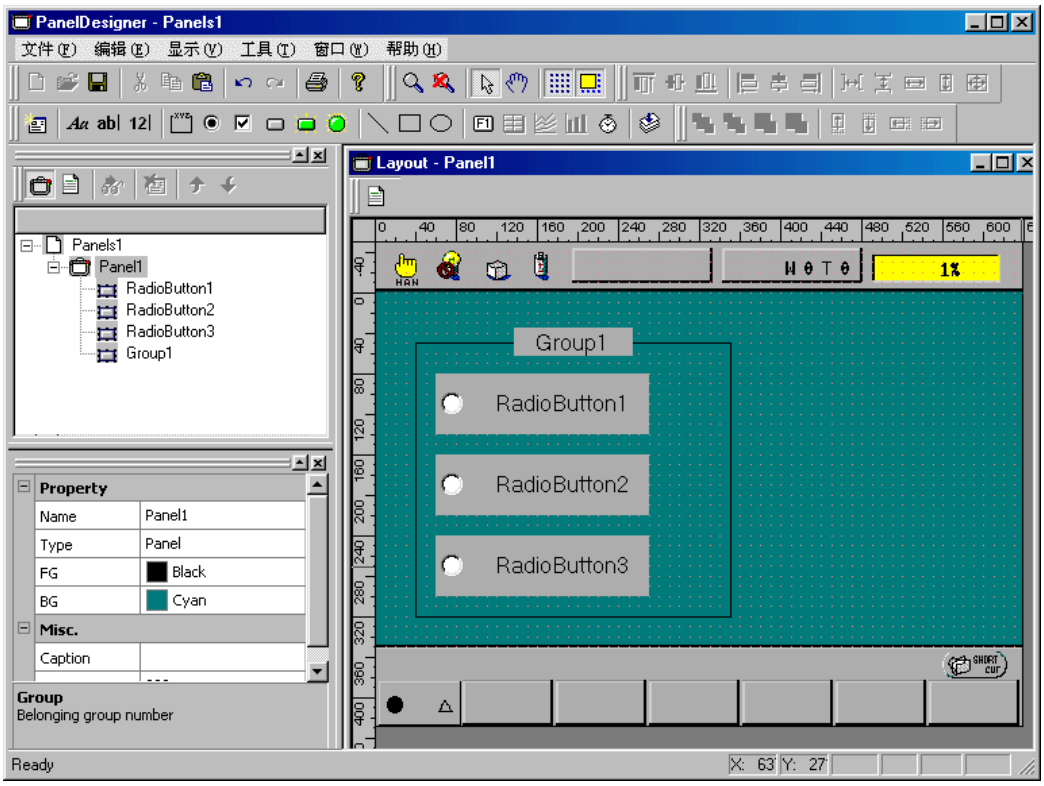
(7) 无线电按钮

无线电按钮是通过用后边介绍的零部件组编组化由此实现同一组内的无线电按钮之间的排他的零部件。与按钮相同，可以分别对被按下的动作，被放开的动作进行记述。和指示灯、检查框相同，ON / OFF状态可以通过参数 (state) 进行访问。

■无线电按钮创建的示例

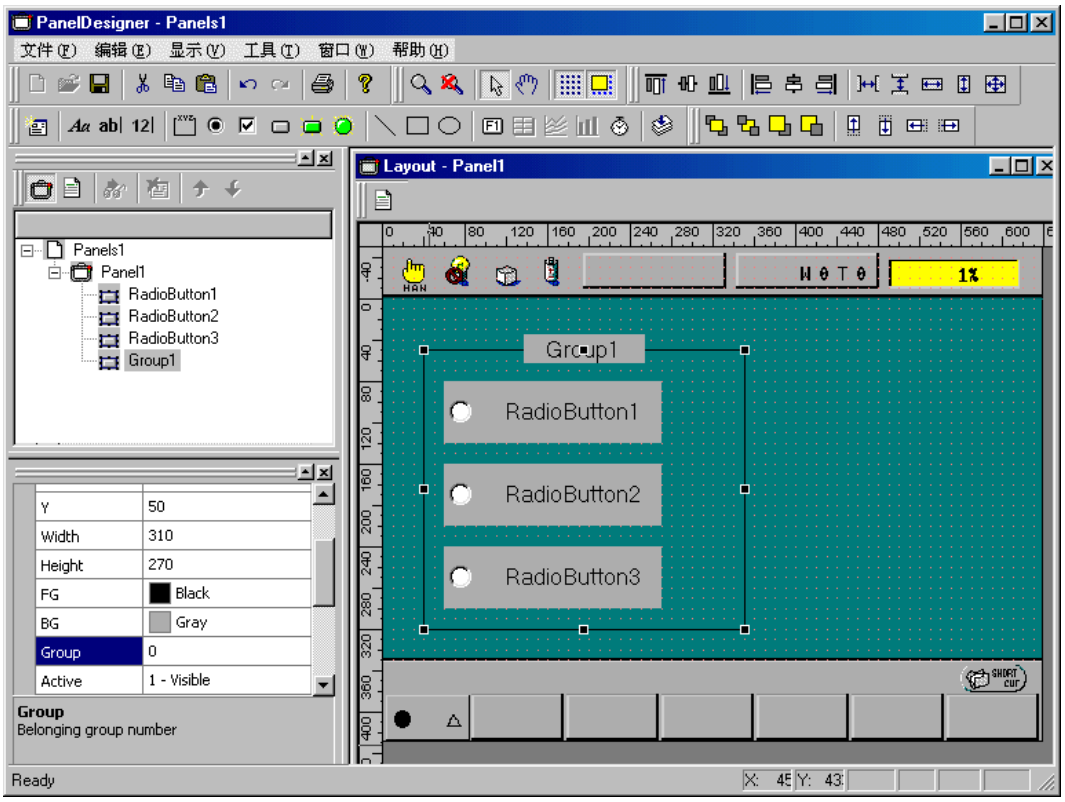
步骤 1

通过3个无线电按钮进行排他处理的示例。
用操作盘编辑程序配置3个无线电按钮和1个组。



步骤 2

请设定所有的配置了无线电按钮的参数 (group) 的零部件组的组编号 (group)。
（这次设定为0）
由此可以实现无线电按钮的排他。



步骤 3

<无线电按钮动作记述>

无线电按钮所具有的动作和按钮一样具有Clicked，Released，所以可以分别就其处理进行记述。

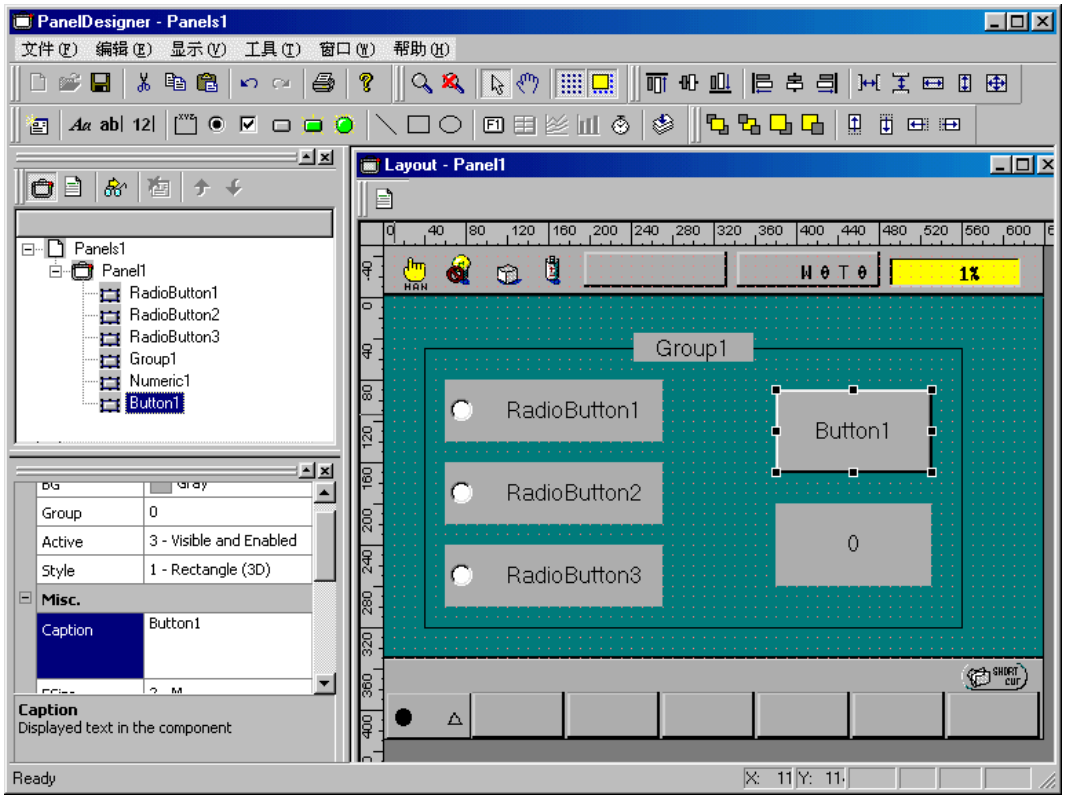
步骤 4

<无线电按钮参数变更>

无线电按钮具有与按钮、标签一样的参数。

在此记述当同一画面上的按钮被按下时，将无线电按钮的状态（ON / OFF）输出到IO [24]～IO [26]，当RadioButton1被选择时将F [10] 的值、当RadioButton2被选择时将F [11] 的值、当RadioButton3被选择时将F [12] 的值存放到同一画面上的数值输入框的示例。

首先配置各个零部件。

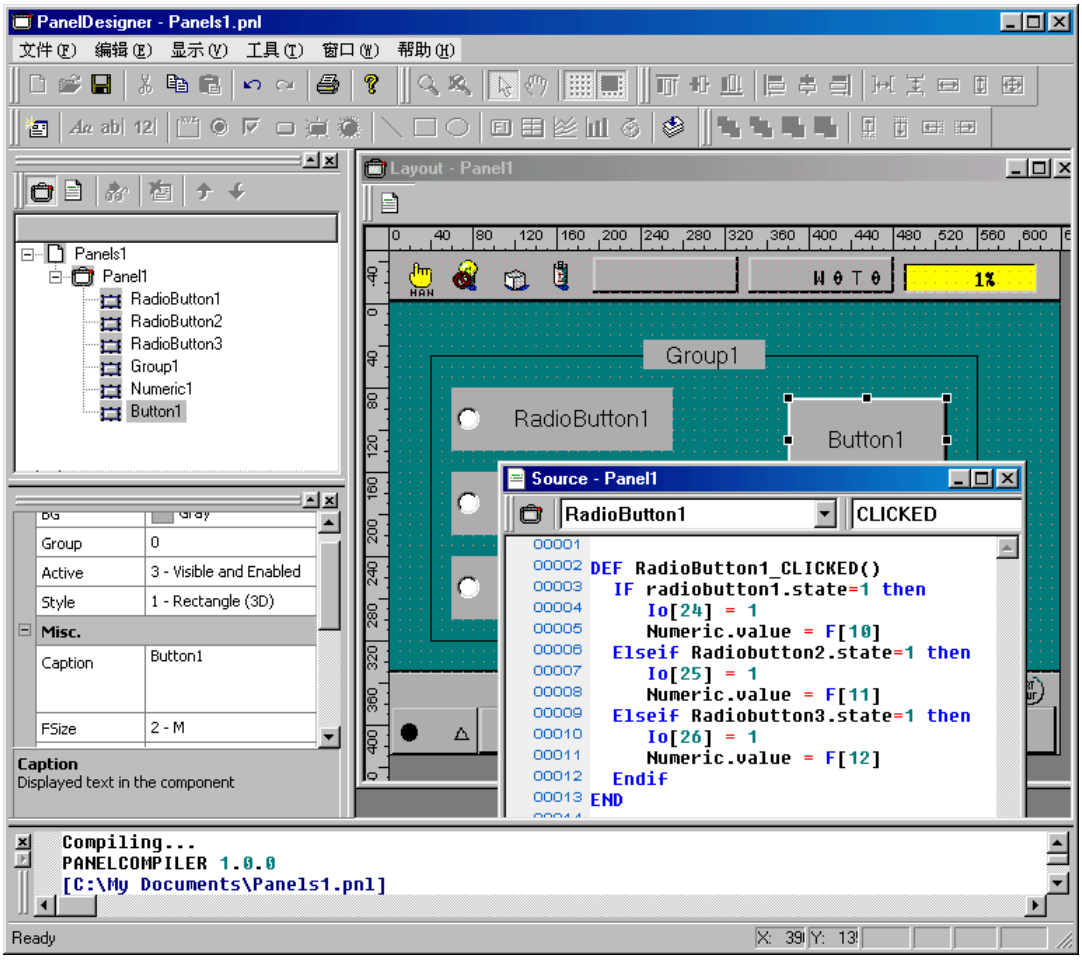


步骤 5

记述当按钮被按下时的处理（选择Button1，Clicked）。

用state参数获取无线电按钮的状态，并用IF语句进行分支的处理如下所示。

```
If radiobutton1.state = 1 then
   Io [24] = 1
   Numeric1.value = F[10]
Elseif radiobutton2.state = 1 then
   Io [25] = 1
   Numeric2.value = F [11]
Elseif radiobutton3.state = 1 then
   Io [26] = 1
   Numeric1.value = F [12]
End if
```



通过将其转发给控制器，可以实现上述的功能。

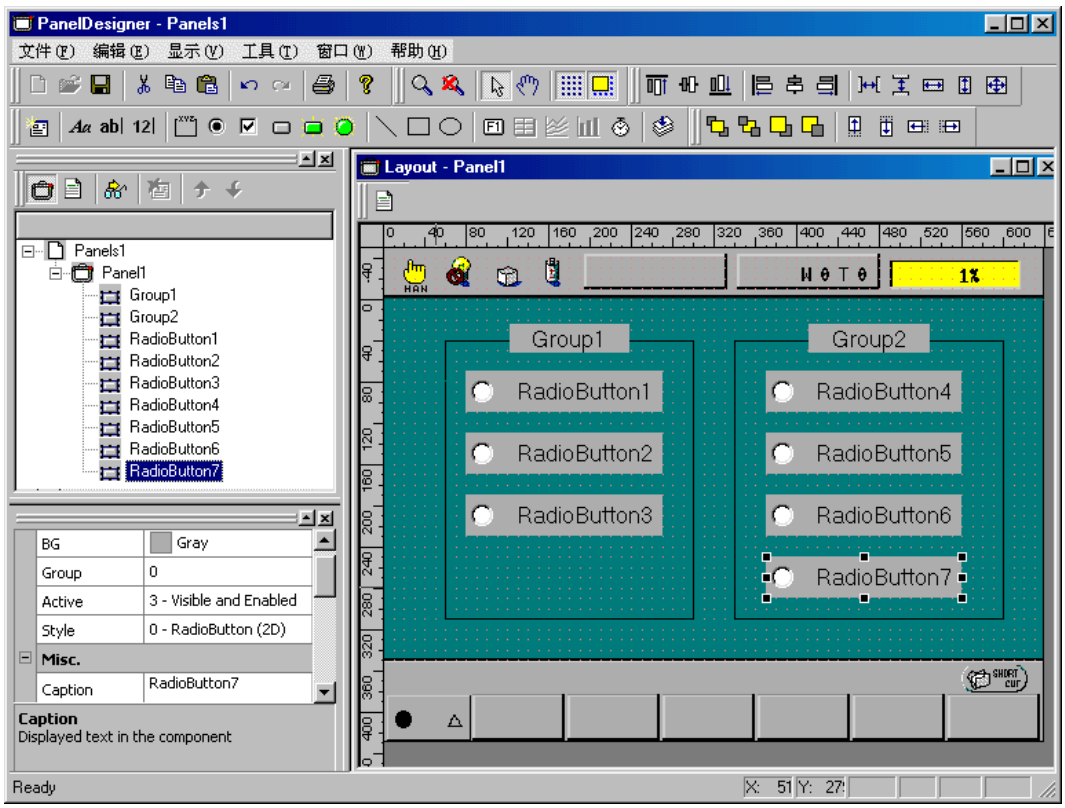
(8) 组

组是为了实现无线电按钮的排他而进行无线电按钮的编组化的零部件。

■组创建的示例

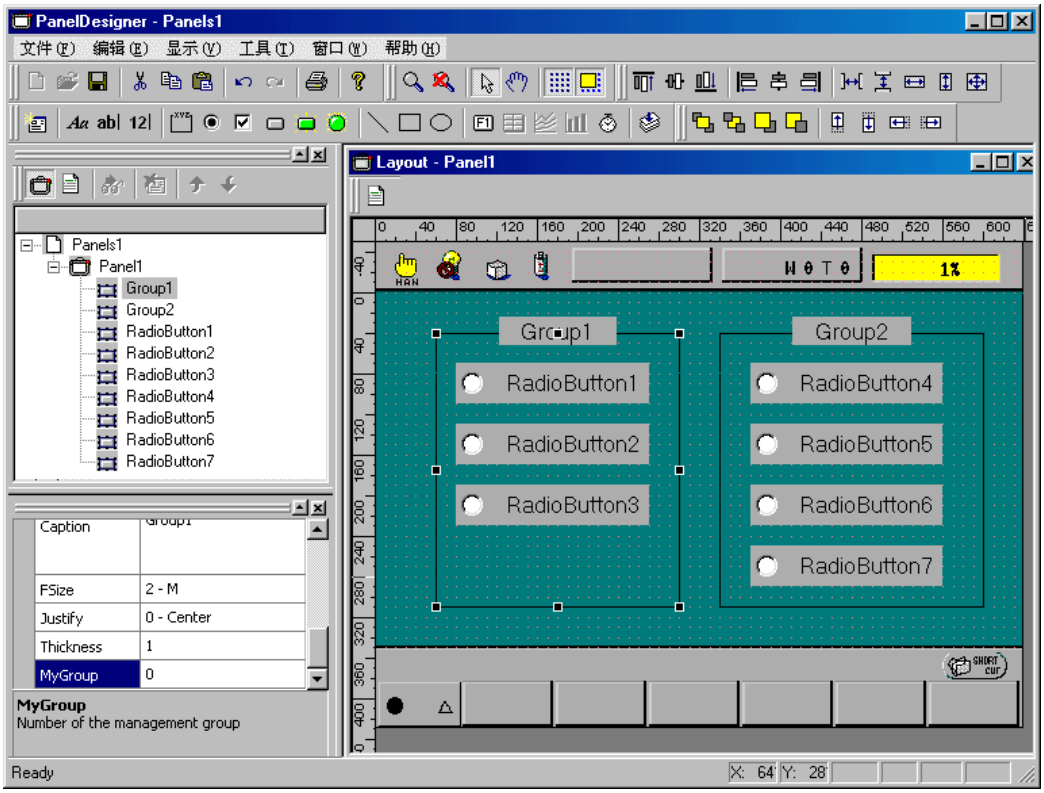
步骤 1

在此创建如下的画面，创建2个组使其分别具有无线电按钮，并用各自的组进行排他处理。用操作盘编辑程序配置2个组和分别3个、4个的无线电按钮。



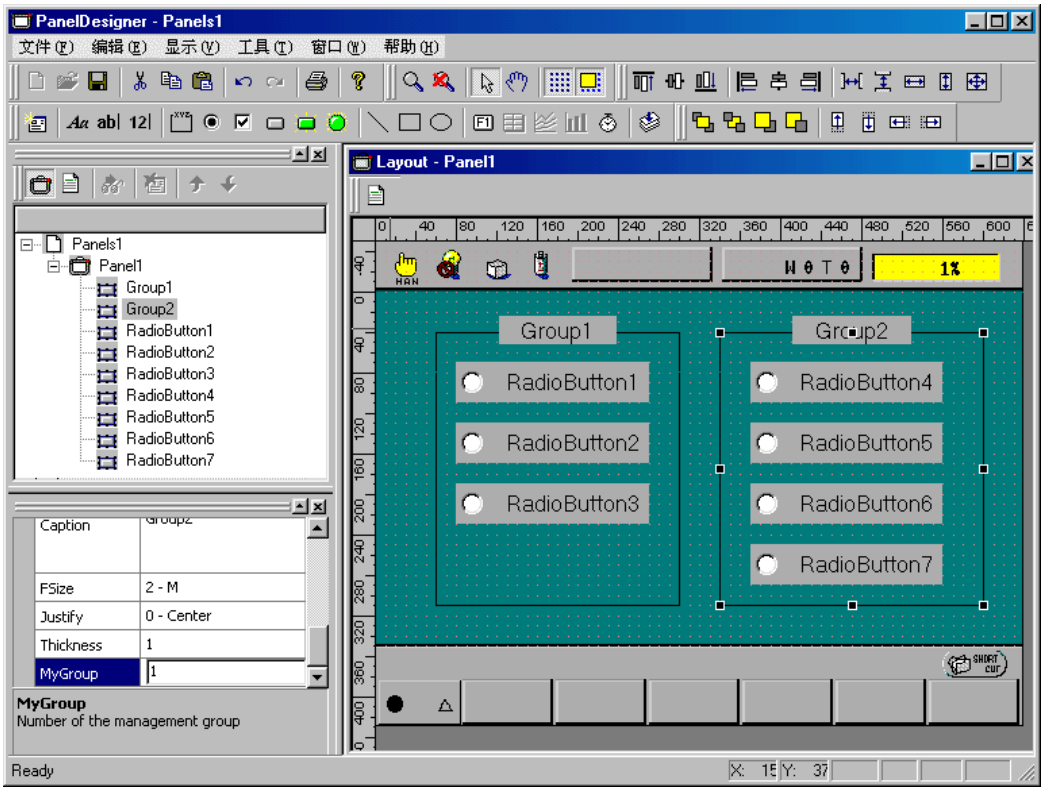
步骤 2

Group1 的编号设为 0，Group2 的编号设为 1。



步骤 3

设定为包含各自所包含的无线电按钮的参数Group的组的编号。



由此可以实现不同无线电按钮的组的排他。

(9) 功能键

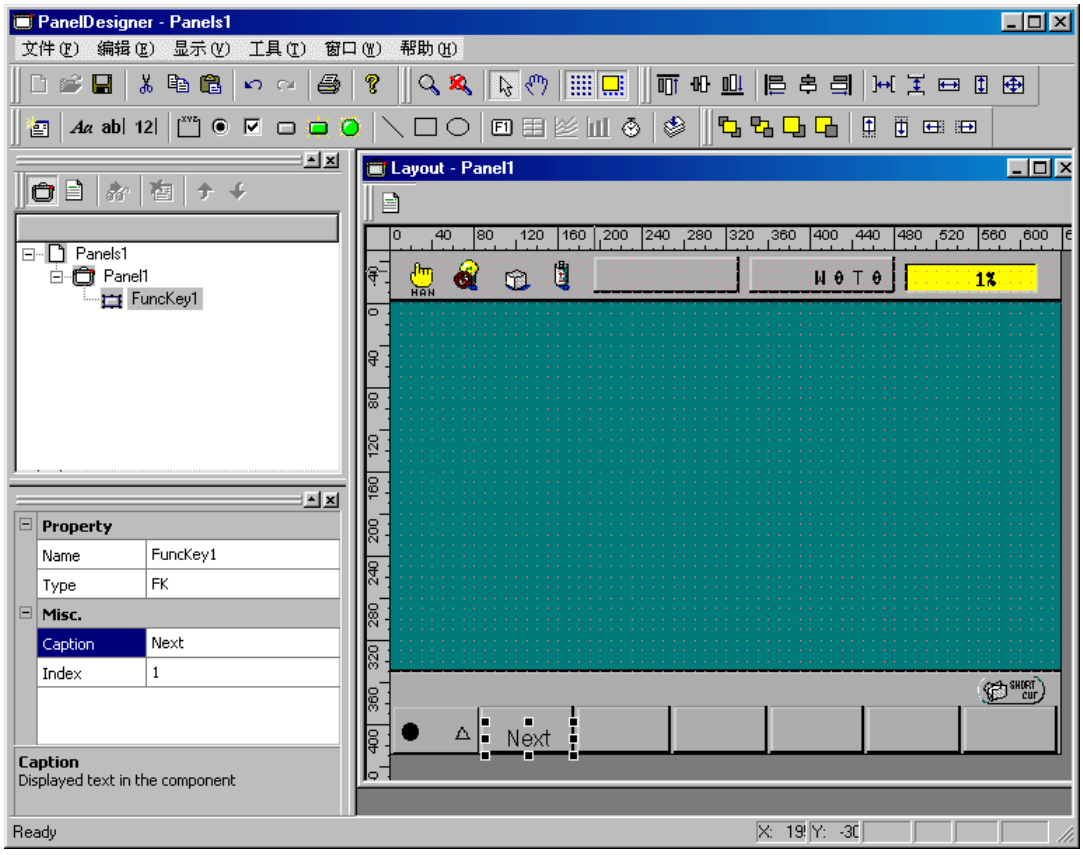
功能键是可以记述当教导器的功能键和按钮一样被按下时的处理，并且可以粘贴字符串的零部件。但是，由于被配置在功能键上，因此不能指定与其他零部件不同的位置。

注：【Ver.2.32以上版本】，在功能键上被追加了 "RELEASED行动"。

■功能键创建的示例

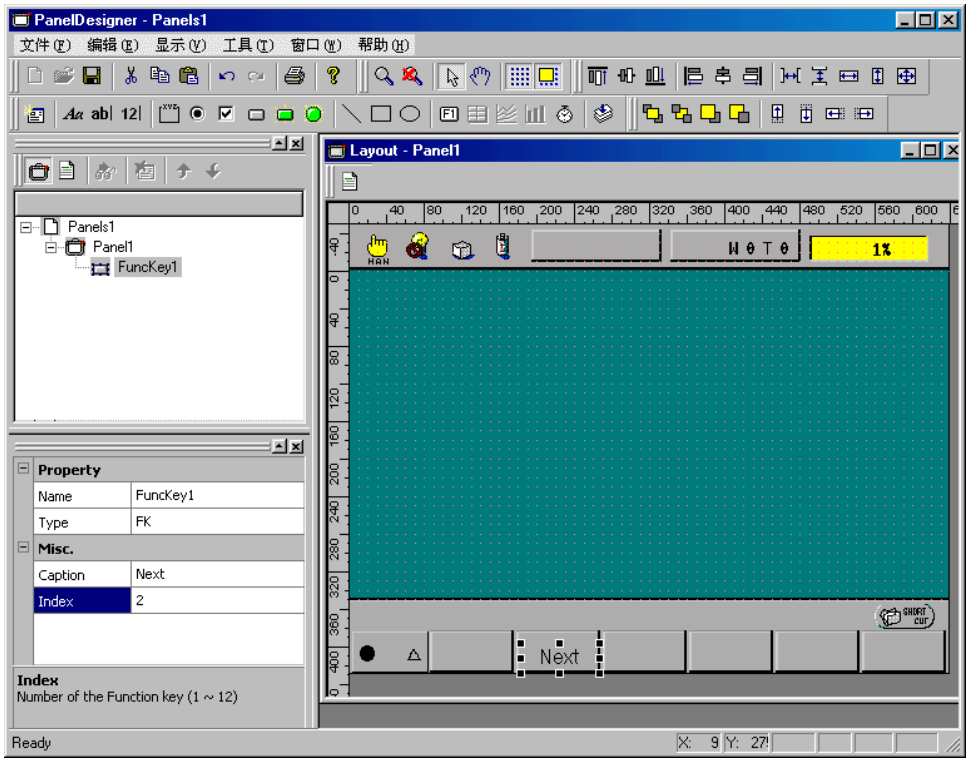
步骤 1

与按钮一样启动操作盘编辑程序，在画面上配置功能键。虽然功能键可以配置在操作盘编辑程序上的任意的位置，但是实际上是被配置在功能键上。因为功能键的显示字符串要设为"下一页"，所以请将Caption变更为 "下一页"。



步骤 2

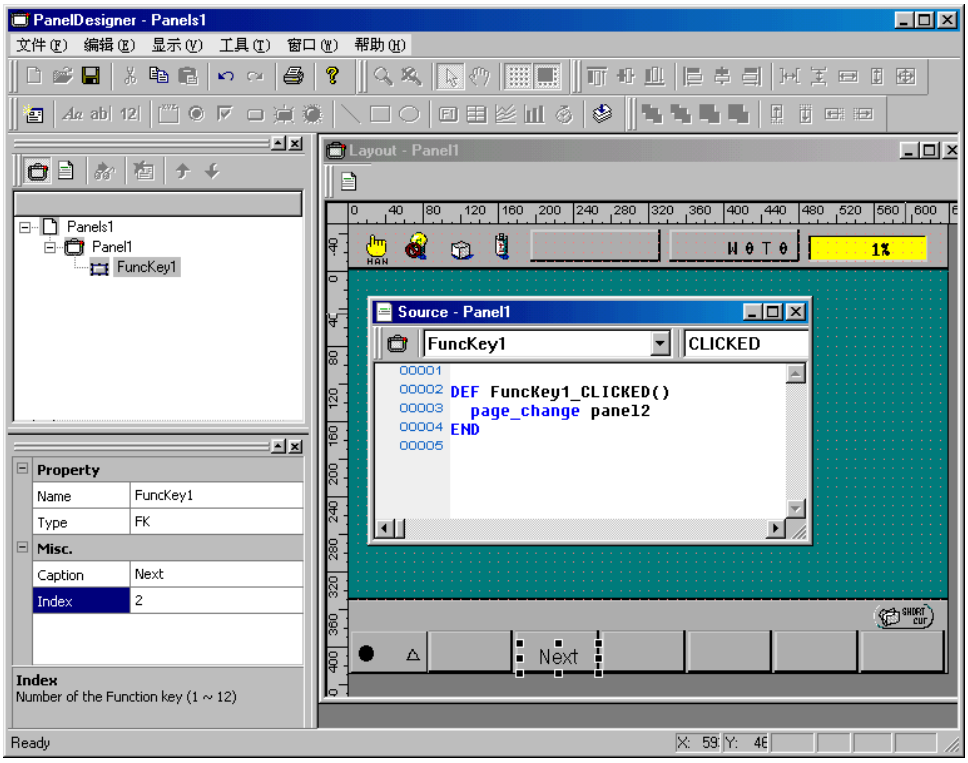
设定想要分配的功能键编号 (0～9)。这次设定为2号。



步骤 3

<功能键动作记述>

功能键与按钮等不同，仅可以对被按下动作的处理进行记述。以下示例记述当被按下时向面板名 "Panel2" 的面板移动。



步骤 4

<功能键参数变更>

功能键的参数比其他零部件要少，访问的参数仅有caption，但是访问的方法与其他零部件相同。

(10) 计时器

计时器是每经过指定时间就发生动作，并可以记述对该动作进行处理的零部件。能够记述处理的动作是用计时器 (TIMER)，通过参数可以指定间隔 (interval)。

■计时器创建的示例

步骤 1

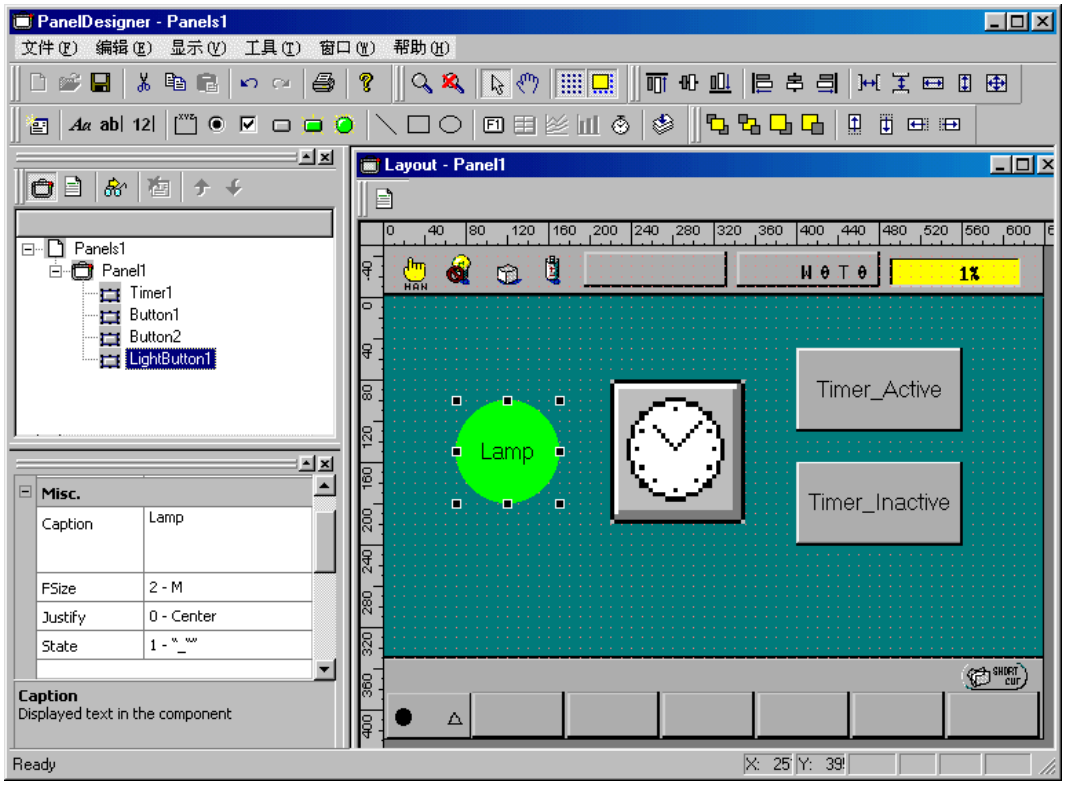
与按钮一样启动操作盘编辑程序，在画面上配置计时器。虽然计时器可以配置在操作盘编辑程序上的任意的位置，但是实际上在教导器上并不进行显示。

步骤 2

<计时器参数变更>

计时器的参数主要是指定有效 / 无效的Active，指定计时器间隔的Interval。使用这些参数，用按钮设定定时器的有效 / 无效，用计时器的TIMER动作将指示灯ON / OFF的示例做如下所示。

首先，用操作盘编辑程序配置零部件（计时器、按钮2个、指示灯）。



步骤 3

<计时器动作记述>

接下来，记述计时器的TIMER动作处理，按钮的Clicked动作处理。记述在TIMER动作中，如指示灯为ON则切换到OFF，如为OFF则切换到ON的处理。

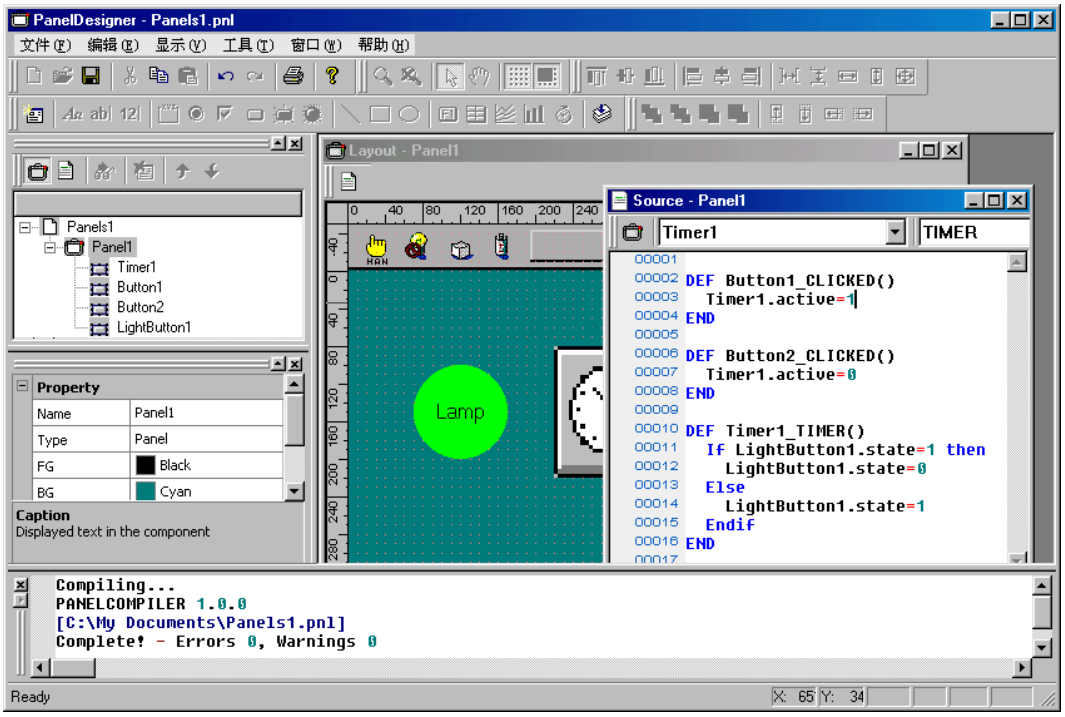
```
If Lightbutton1.state = 1 then
   Lightbutton1.state = 0
Else
   Lightbutton1.state = 1
End if
```

在Button1（caption：计时器有效）的Clicked动作中将计时器置为有效，

```
Timer1.active = 1
```

在Button2（caption：计时器无效）的Clicked动作中将计时器置为无效。

```
Timer1.active = 0
```



(11) 线

线零部件在画面上用指定的式样描绘线。与之后介绍的圆、四边形相同，这些只是用来描绘的零部件不会动作。参数变更是由同一画面上的其他零部件进行的。

■线创建的示例

步骤 1

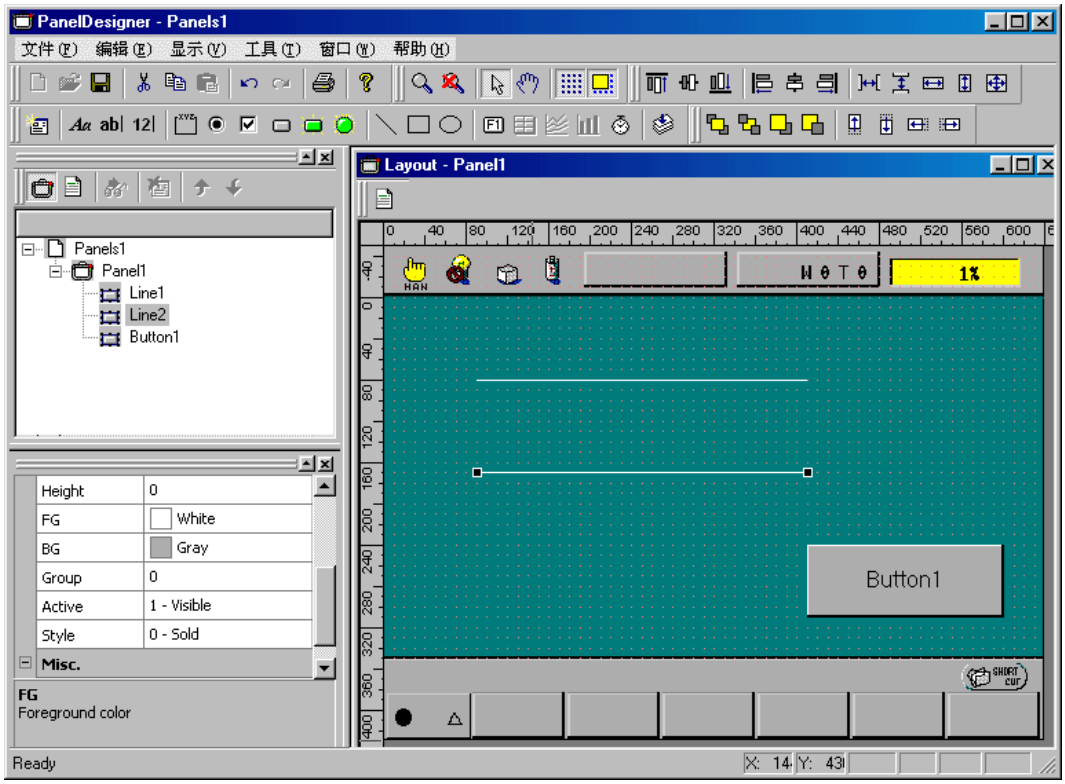
与按钮一样启动操作盘编辑程序，并在画面上描绘线。

步骤 2

<线参数变更>

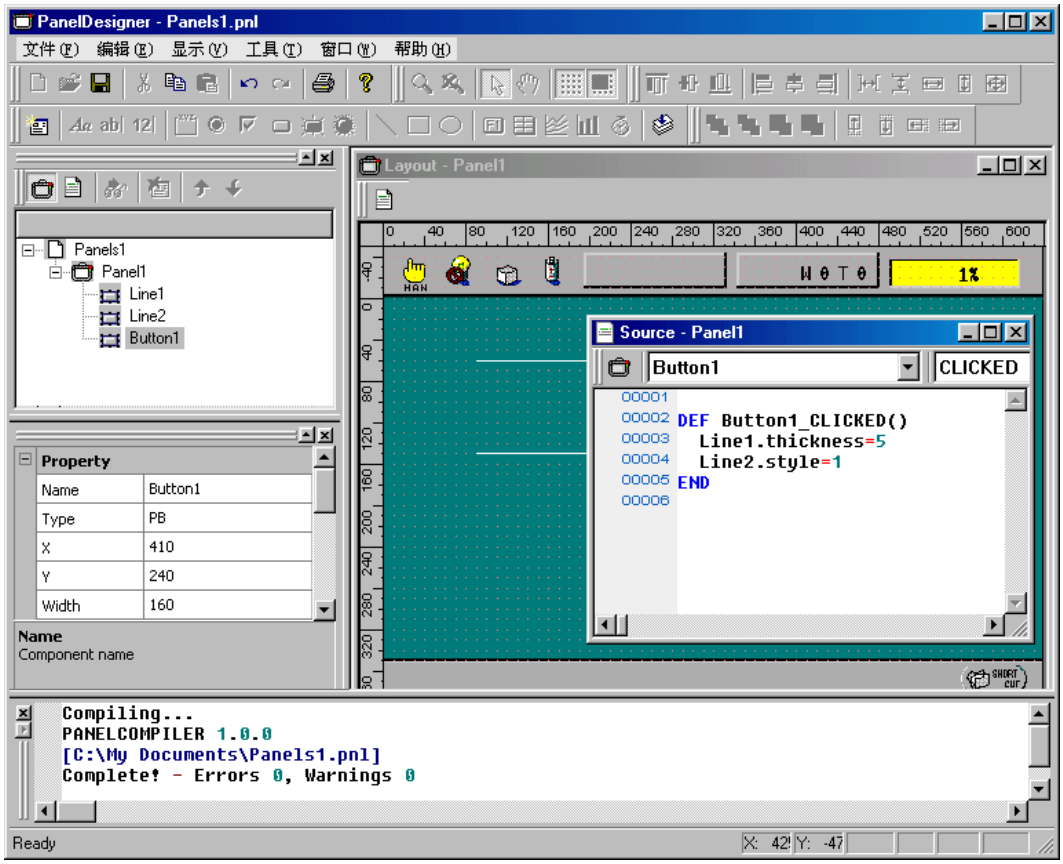
线零部件里主要有线型 (style) 和线的粗细。以下为当按钮被按下时变更线的粗细和线型的示例。

用操作盘编辑程序配置2根线和按钮。



步骤 3

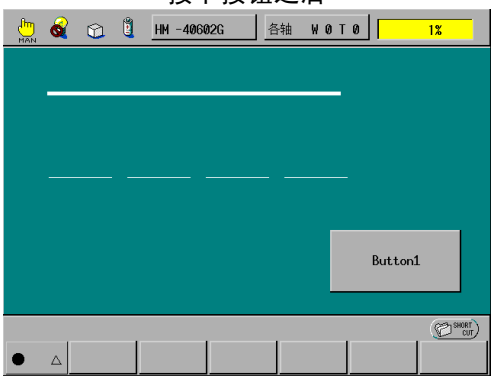
在源程序编辑画面上记述当按钮被按下时，将线 1 的粗细置为 5 点，将线 2 的线型置为虚线。



步骤 4

由此，可以创建当按下按钮，线的粗细和线型会变化的画面。

按下按钮之后



(12) 圆

圆零部件在画面上用指定的式样描绘圆。与线、四边形相同，这些只是用来描绘的零部件不会动作。参数变更是由同一画面上的其他零部件进行的。

■圆创建的示例

步骤 1 与按钮一样启动操作盘编辑程序，并在画面上描绘圆。

步骤 2 <圆参数变更>

圆零部件里与线零部件相同，主要有线型 (style) 和线的粗细。变更、参照可以和线零部件一样进行。

(13) 四边形

四边形零部件在画面上用指定的式样描绘四边形。与圆、线相同，这些只是用来描绘的零部件不具有动作。参数变更是由同一画面上的其他零部件进行的。

■四边形创建的示例

步骤 1 与按钮一样启动操作盘编辑程序，并在画面上描绘四边形。

步骤 2 <四边形参数变更>

四边形零部件里与线零部件相同，主要有线型 (style) 和线的粗细。
变更、参照可以和线零部件一样进行。

(14) 照明式按钮

照明式按钮的操作与通常的按钮相同，是显示具有指示灯功能的特殊零部件。因此，可以分别对被按下的动作、被放开的动作、被刷新的动作进行记述。
和指示灯、复选框相同，ON / OFF状态可以通过参数 (state) 进行访问。

■照明式按钮创建的示例

步骤 1

配置与通常的按钮相同，可以在操作盘编辑程序上进行。

步骤 2

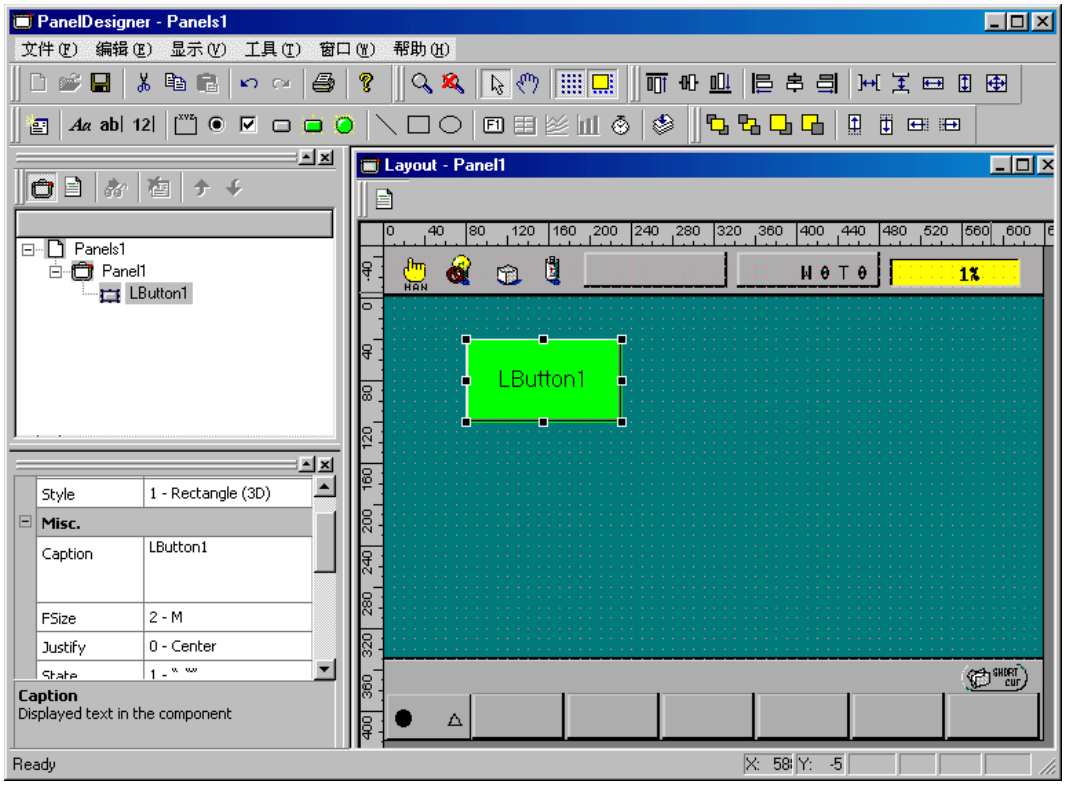
<照明式按钮行动记述>

照明式按钮可能的动作因具有Clicked，Released，Refresh，所以可以分别就其进行记述处理。

步骤 3

<照明式按钮参数变更>

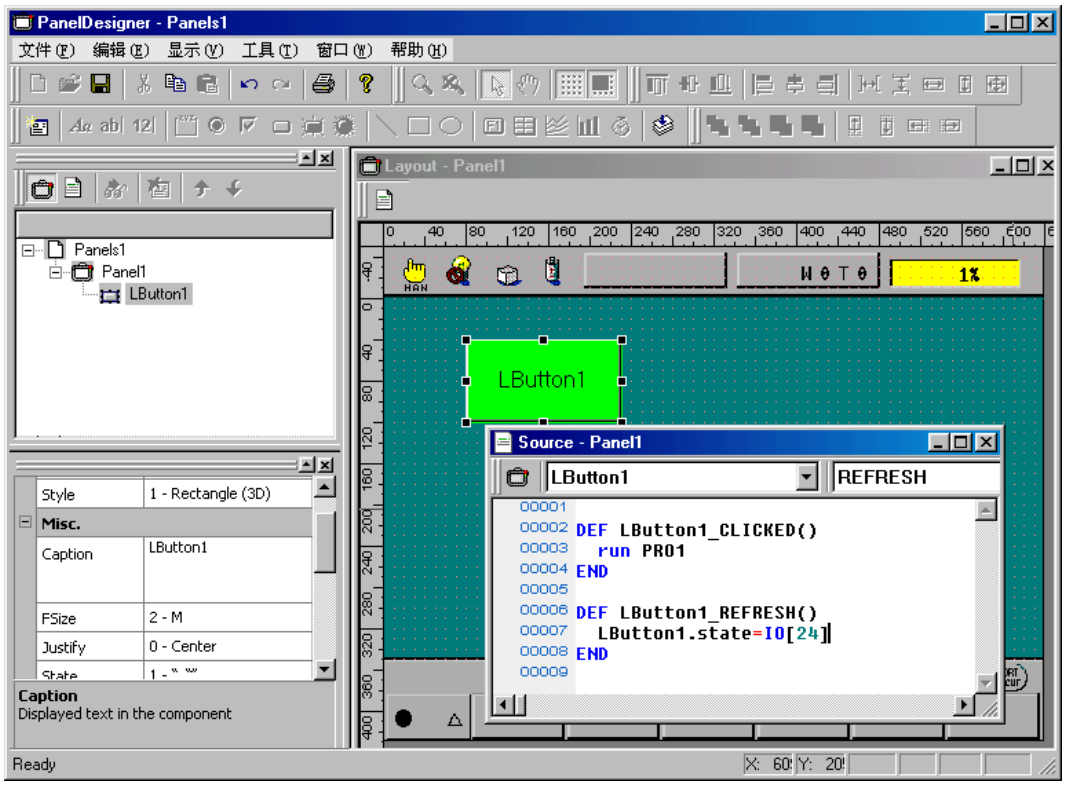
在此示出通过照明式按钮进行程序启动和I / O状态的显示的示例。
当照明式按钮被按下时，启动同一文件夹下的程序（启动后2秒打开IO [24]），并将IO [24]的状态反映到照明式按钮。
首先配置各个零部件。



步骤 4

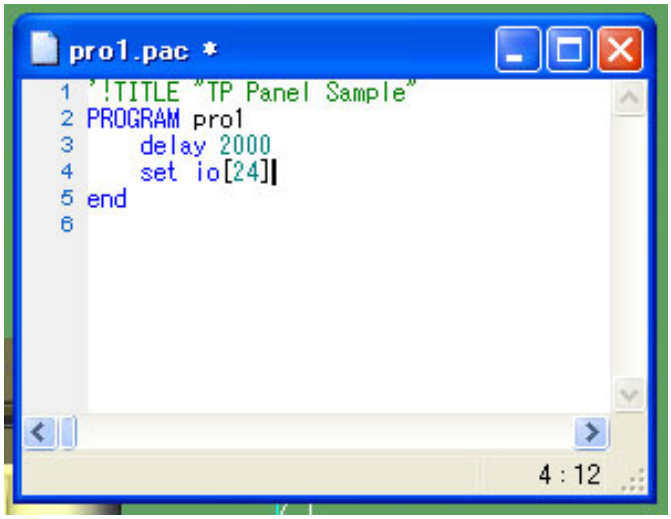
对用于实现上述功能的零部件动作进行记述。

```
lbutton1.state = io [24]      '使 io [24] 的状态反映到 lbutton1 的状态
```



步骤 5

接着，用WINCAPSIII记述将启动的程序。



通过将其编译并转发给控制器，可以实现上述功能。

2.3　PAC 语言，与系统的界面

PAC语言与操作盘之间可以通过全局变量、文件夹变量来进行数据交换。
此外与系统的界面可以通过sysstate指令、I / O变量来实现。

2.3.1　PAC 变量的获取、显示

虽然PAC变量中存在全局变量、局部 (Local) 变量、文件夹变量，但从操作盘可以访问的是全局变量和文件夹变量。从操作盘调出时，全局变量无需定义就可使用，但是文件夹变量若不用EXTERN进行定义，就不能使用。以下为将各种变量值显示在操作盘上的示例。

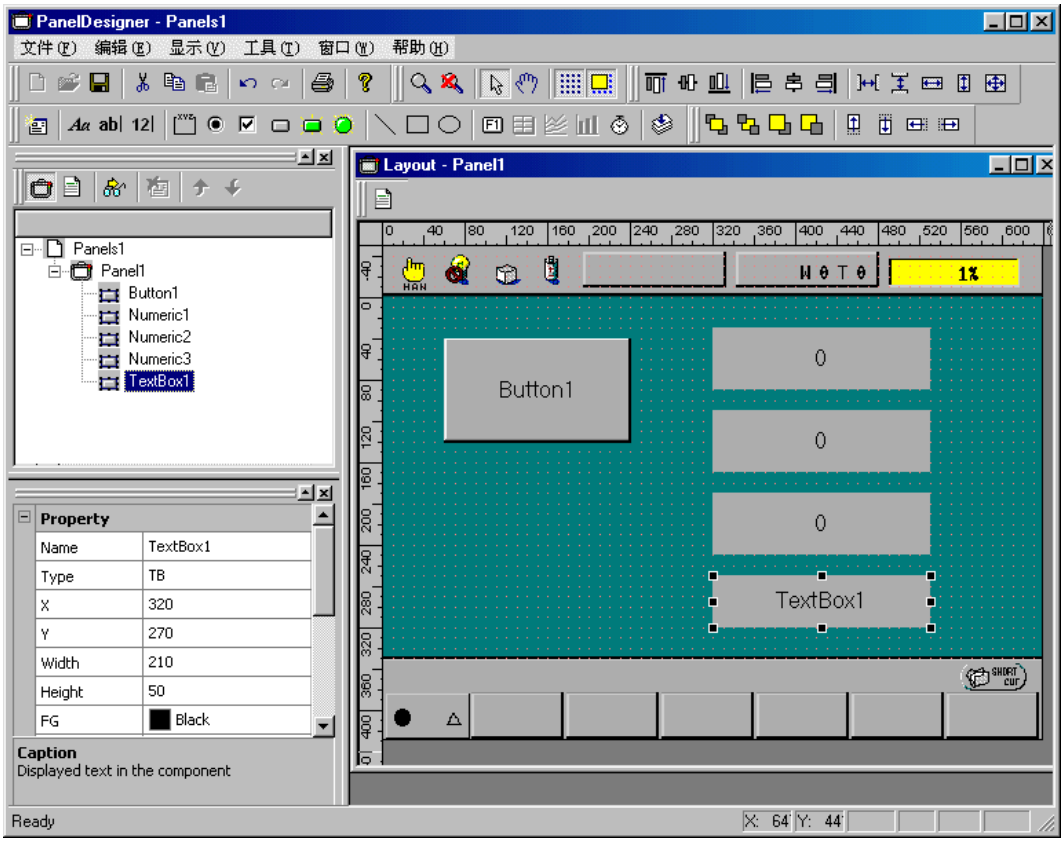
■将全局变量显示在操作盘上的示例

对全局变量的访问
　用变量种类（I：整数、F：单精度实数、D：双精度实数、S：字符串）的形式可以执行。
例如，要访问全局整数变量的10号可以记述为I [10]。

在操作盘上当按钮被按下时，获取I、F、D、S型的全局变量，并将其分别在数值输入框、文本框进行显示的示例做如下所示。

步骤 1

首先，用操作盘编辑程序配置记述数据获取处理的按钮；配置将所获取的I、F、D型全局变量进行显示的数值输入框；配置将S型全局变量进行显示的文本框。

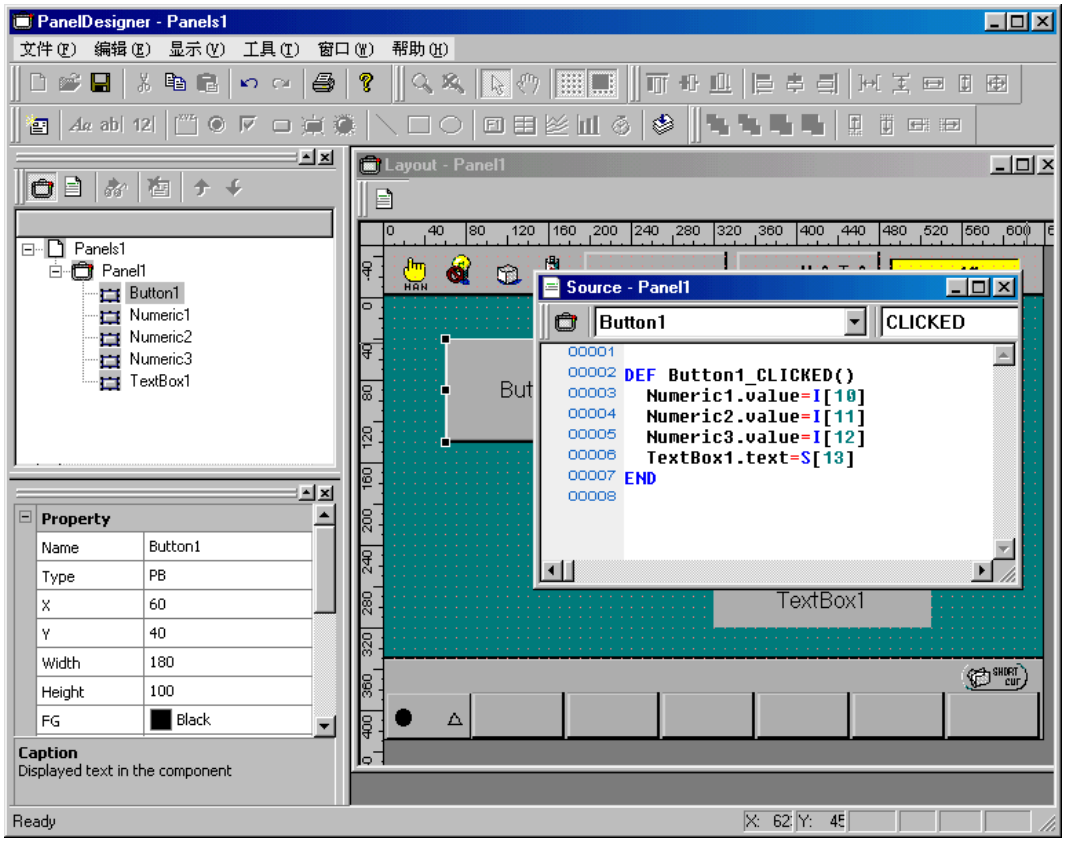


步骤 2

在源程序编辑画面上记述当按钮被按下时的处理。

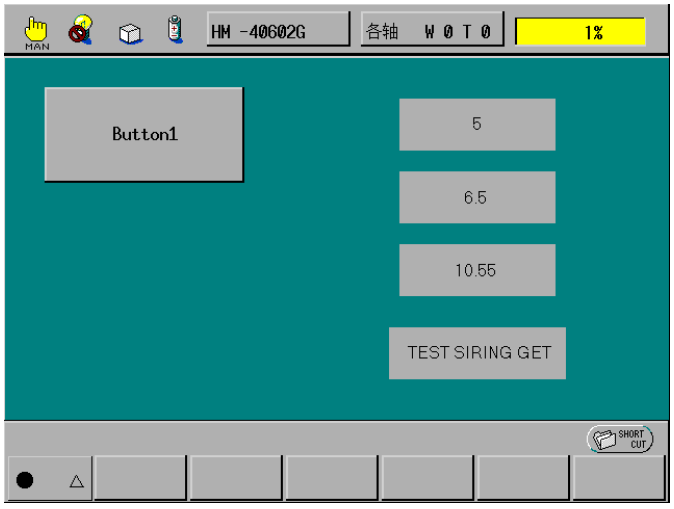
在此，进行如下指定，将I、F、D、S型全局变量的各个10号、11号、12号、13号赋值到数值输入框、文本框的值中。

```
Numeric1.value = I [10]
Numeric2.value = F [11]
Numeric3.value = D [12]
Textbox1.text  = S [13]
```



步骤 3

将转发给控制器按下按钮时的结果做如下所示。



■将文件夹变量显示在操作盘上的示例

对文件夹变量的访问可以通过用在按钮等的动作程序内的EXTERN型定义(DEFINT, DEFSNG, DEFDBL, DEFSNG) 文件夹变量的形式进行记述。
例如，要访问文件夹整数变量的itest时，在动作程序内进行 "EXTERN DEFINT itest" 定义，并在程序内记述 "itest"。

例如，在操作盘上当按钮被按下时，获取整数、单精度实数、双精度实数、字符串型的文件夹变量，并将其分别在数值输入框、文本框中进行显示的示例做如下所示。

步骤 1

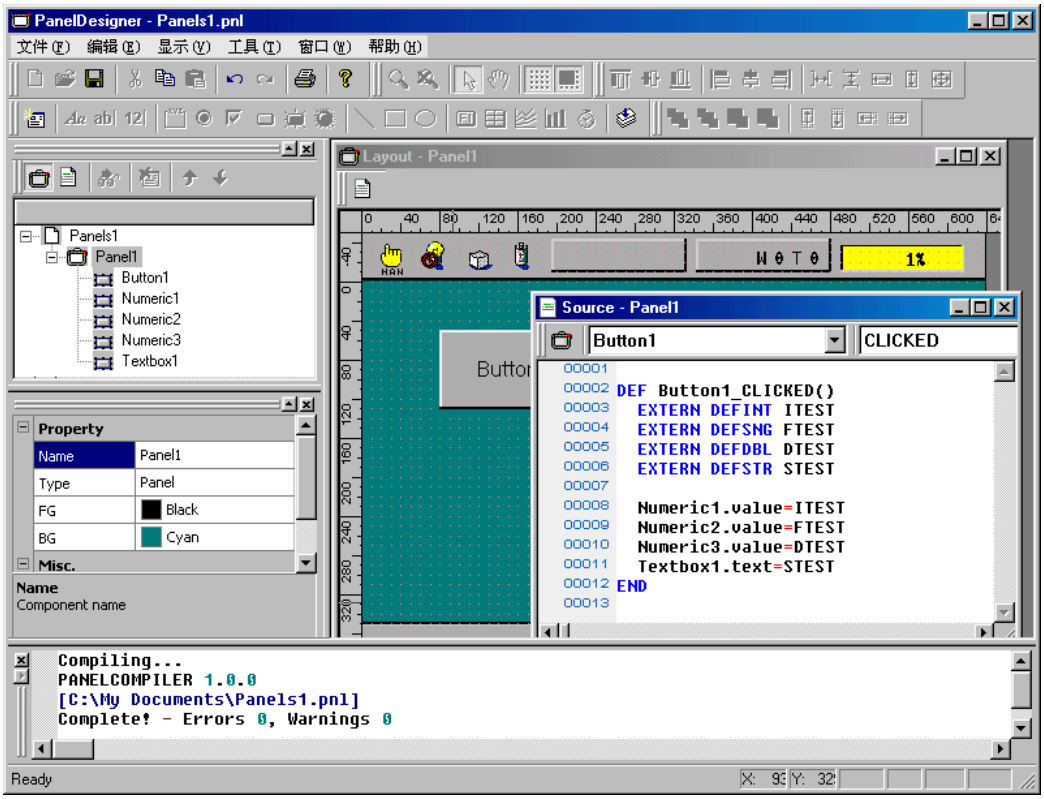
首先，用操作盘编辑程序配置记述数据获取处理的按钮；配置将所获取的整数、单精度实数、双精度实数型文件夹变量进行显示的数值输入框；配置将字符串型文件夹变量进行显示的文本框。
与在全局变量中所示的示例相同，用操作盘编辑程序配置零部件。

步骤 2

在源程序编辑画面上记述当按钮被按下时的处理。

在此，进行如下指定，将整数、单精度实数、双精度实数、字符串型文件夹变量的各个itest, ftest, dtest, stest赋值到数值输入框、文本框的值中。

```
EXTERN DEFINT ITEST
EXTERN DEFSNG FTEST
EXTERN DEFDBL DTEST
EXTERN DEFSTR STEST
Numeric1.value = ITEST
Numeric2.value = FTEST
Numeric3.value = DTEST
Textbox1.text  = STEST
```



2.3.2　PAC 变量的变更

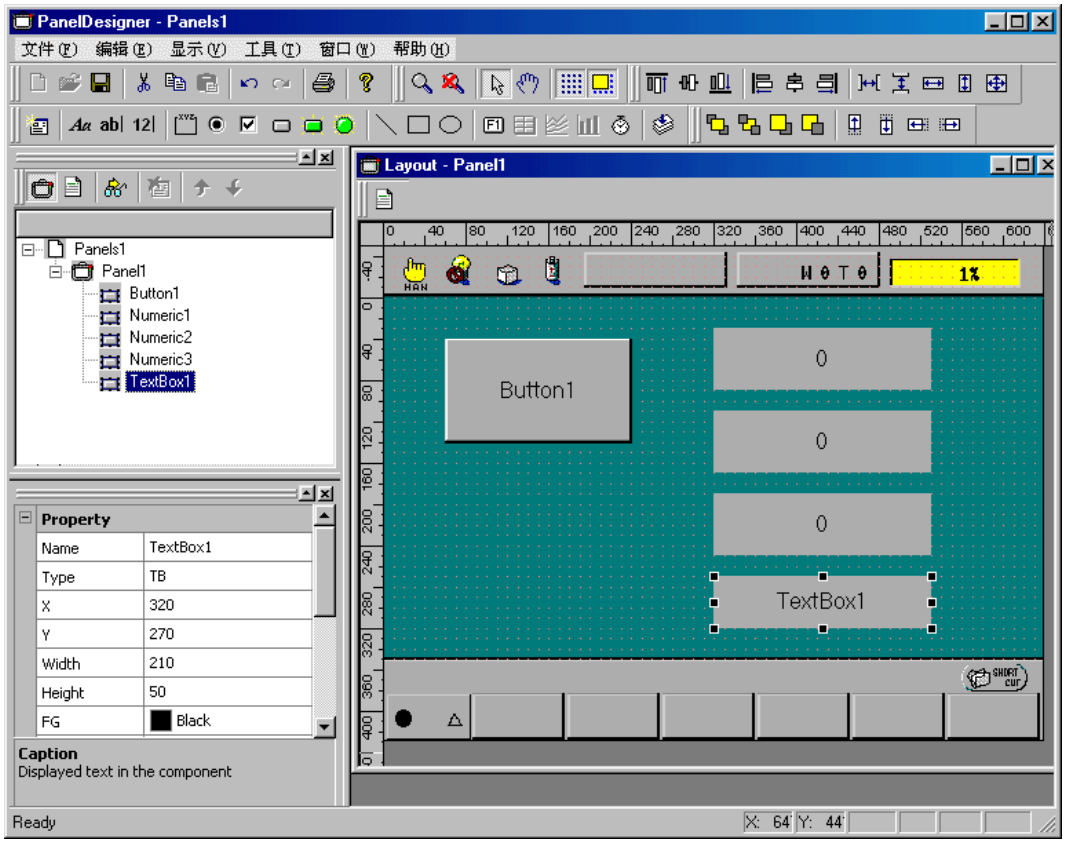
关于变更也可以以与 "2.3.1　PAC变量的获取、显示" 相同的形式进行访问。

■全局变量的示例

在此，在操作盘上当按钮被按下时，将数值输入框、文本框的值存放到I、F、D、S型的全局变量中的示例做如下所示。

步骤 1

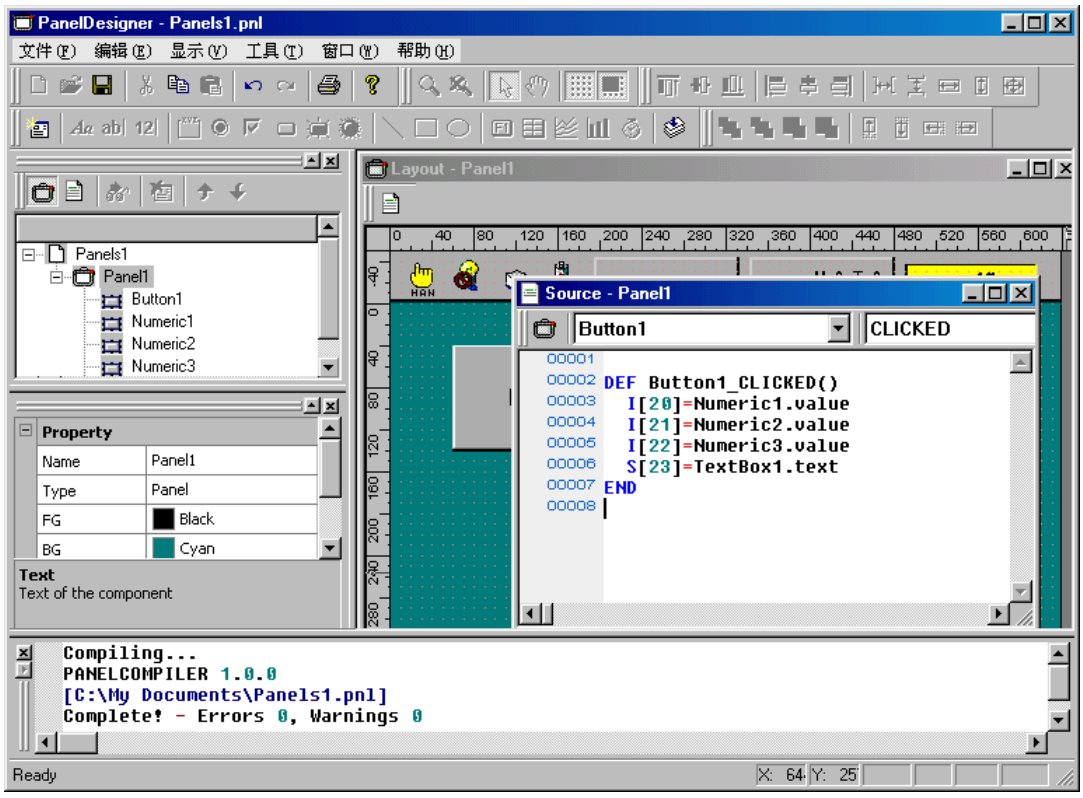
首先，用操作盘编辑程序配置记述数据存放处理的按钮；配置具有向I、F、D型全局变量写入的数据的数值输入框；配置具有向S型全局变量写入的数据的文本框。



步骤 2

在源程序编辑画面上记述当按钮被按下时的处理。
在此，指定将数值输入框、文本框的值代入到对应I、F、D、S型全局变量的20号、21号、22号、23号中。

```
I [20] = Numeric1.value
F [21] = Numeric2.value
D [22] = Numeric3.value
S [23] = Textbox1.text
```



■文件夹变量的示例

对文件夹变量的访问与 "2.3.1　PAC变量的获取、显示" 的情况相同。在操作盘上当按钮被按下时，获取数值输入框、文本框的值，并将其分别存放到整数、单精度实数、双精度实数、字符串型的文件夹变量中的示例做如下所示。

步骤 1

首先，用操作盘编辑程序配置记述数据获取处理的按钮；配置将所获取的整数、单精度实数、双精度实数型文件夹变量进行显示的数值输入框；配置将字符串型文件夹变量进行显示的文本框。

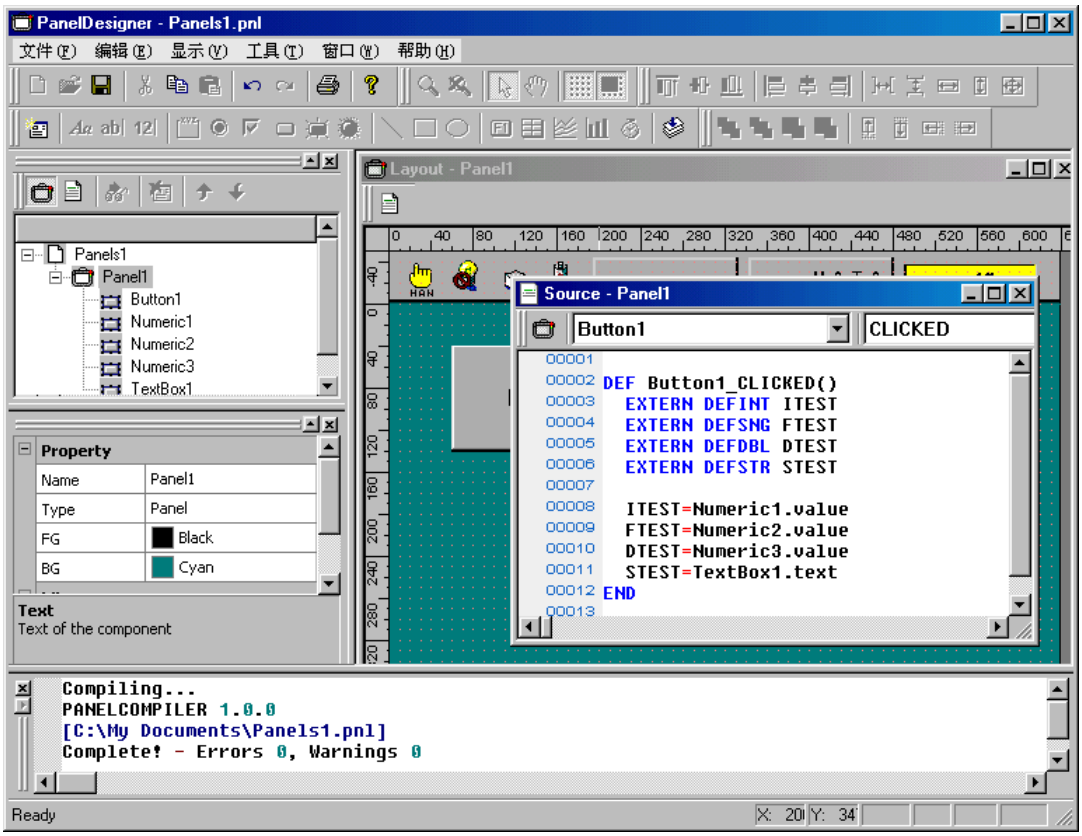
与在全局变量中所示的示例相同，用操作盘编辑程序配置零部件。

步骤 2

在源程序编辑画面上记述当按钮被按下时的处理。
在此，进行如下指定，将整数、单精度实数、双精度实数、字符串型文件夹变量的各个itest、ftest、dtest、stest赋值到数值输入框、文本框的值中。

```
EXTERN DEFINT ITEST
EXTERN DEFSNG FTEST
EXTERN DEFDBL DTEST
EXTERN DEFSTR STEST

ITEST = Numeric1.value
FTEST = Numeric2.value
DTEST = Numeric3.value
STEST = Textbox1.text
```



2.3.3　I / O 状态的获取

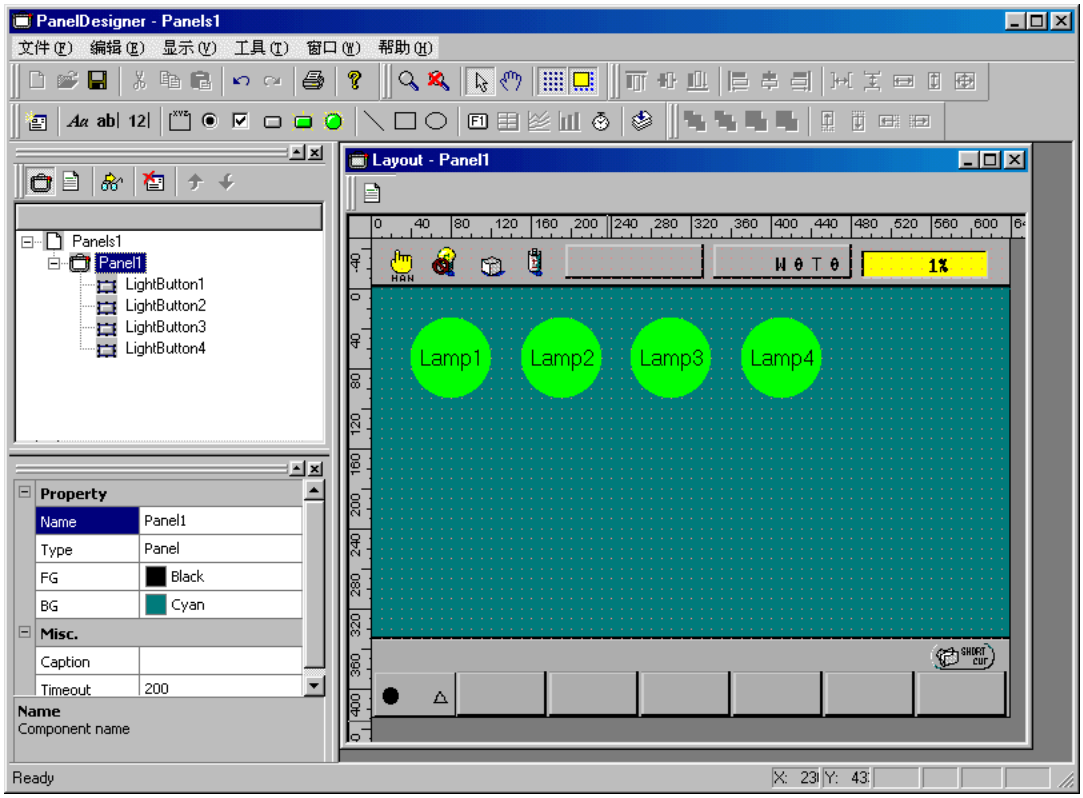
为了获取机械手控制器的I / O状态，有使用I / O全局变量的方法和通过DEFIO定义使用局域I / O变量的方法。
在此，先就使用I / O全局变量的方法进行说明。

■I / O全局变量的示例

对I / O全局变量的访问通过以下的形式进行。
　IO［IO编号］
例如，定期地获取I / O编号24～27的状态，并将其作为指示灯的状态显示。

步骤 1

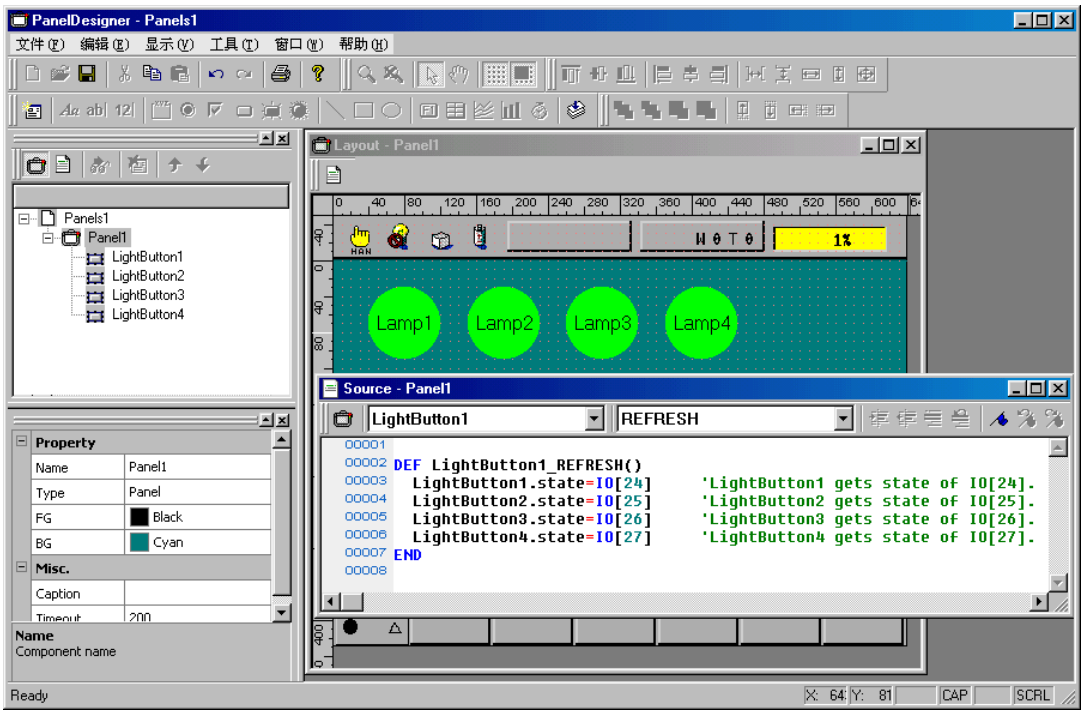
用操作盘编辑程序配置4个显示I / O状态的指示灯。



步骤 2

在源程序编辑画面上记述获取I / O状态、并将其反映到指示灯的处理。
为了定期地进行处理，因此使用指示灯的REFRESH动作。
在此，从源程序编辑画面的下拉列表中选择LightButton1和REFRESH，创建程序的外部，并将处理记述到内部。

```
LightButton1.state = IO [24]      '获取I / O 24号的状态，反映在LightButton1中
LightButton2.state = IO [25]      '获取I / O 25号的状态，反映在LightButton2中
LightButton3.state = IO [26]      '获取I / O 26号的状态，反映在LightButton3中
LightButton4.state = IO [27]      '获取I / O 27号的状态，反映在LightButton4中
```



2.3.4 I / O 状态的变更

要变更系统的I / O状态可以使用SET / RESET指令，具体指令形式如下。

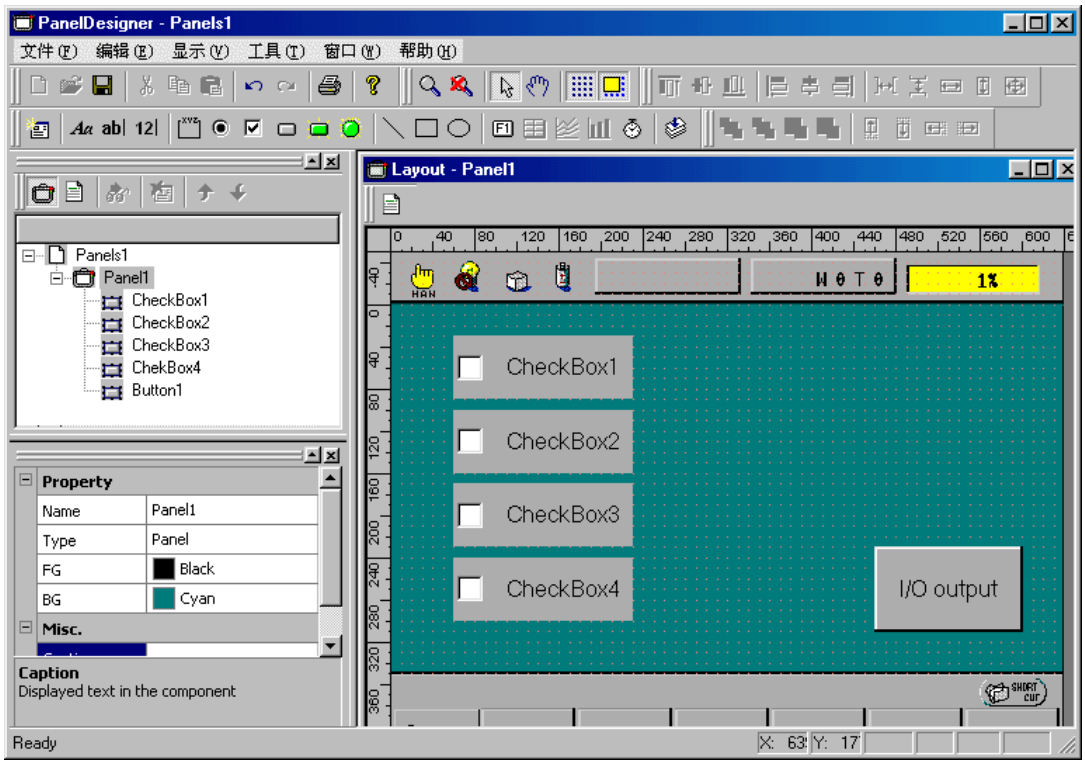
打开时： SET IO［IO编号］

关闭时： RESET IO［IO编号］

■操作盘中的I / O状态的变更示例

例如，当按钮被按下时，将检查框的状态反映到系统的I / O状态（I / O编号28～31）。

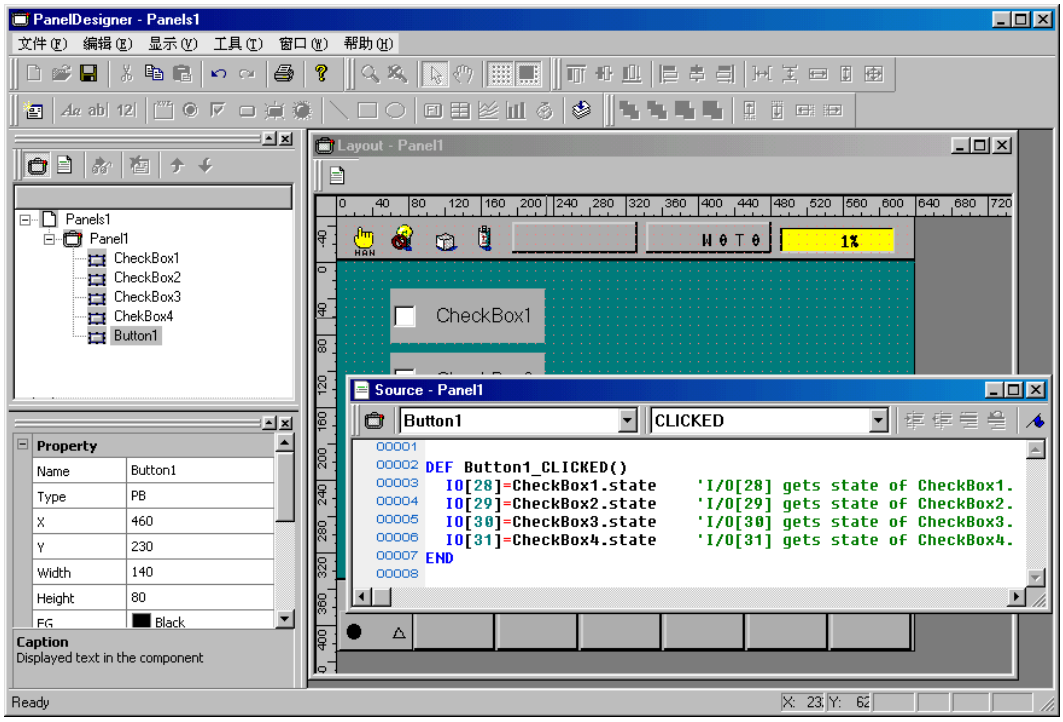
步骤 1 用操作盘编辑程序配置4个检查框和当被按下时向I / O进行数据输出的按钮。



步骤 2

在源程序编辑画面上记述获取检查框状态、并向I / O输出的处理。

```
IO [28] = Checkbox1.state        'Get Checkbox1 state, reflect to I / O 28
IO [29] = Checkbox2.state        'Get Checkbox2 state, reflect to I / O 29
IO [30] = Checkbox3.state        'Get Checkbox3 state, reflect to I / O 30
IO [31] = Checkbox4.state        'Get Checkbox4 state, reflect to I / O 31
```



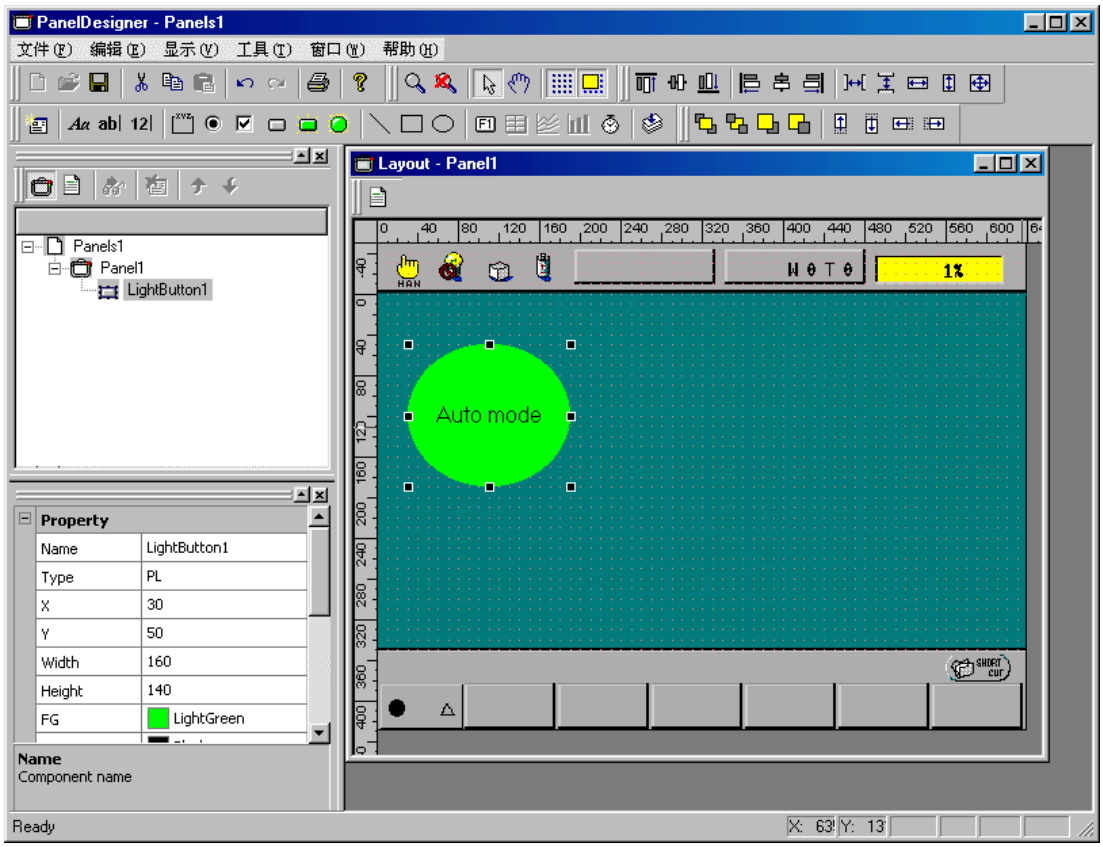
2.3.5　系统状态的获取

要获取系统状态，请使用SYSSTATE指令。
关于SYSSTATE指令的详细内容请参照第5章中的指令介绍。

■操作盘中的系统状态的获取示例

以下为获取控制器的自动模式状态，并使其反映到指示灯状态的示例。

步骤 1　用操作盘编辑程序配置指示灯。

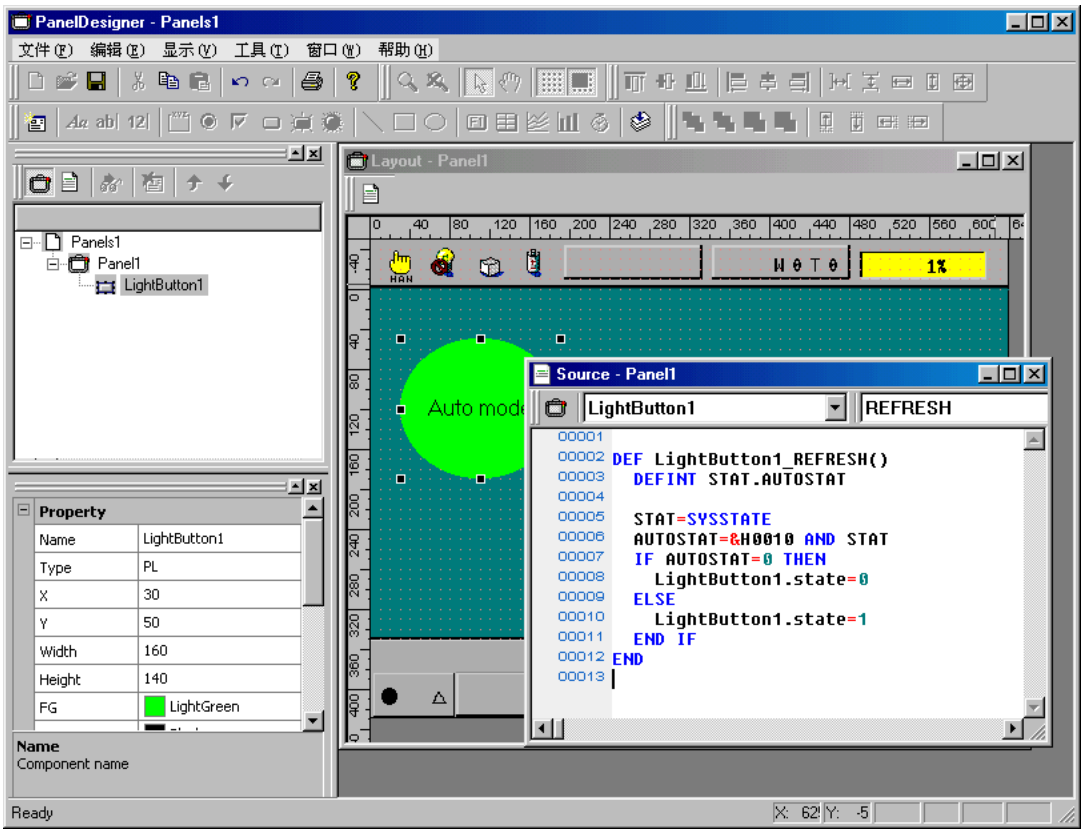


步骤 2

用指示灯的REFRESH动作记述获取自动模式状态、并将其显示的处理。
在此使用LightButton1的REFRESH行动，进行获取控制器状态，模式信息的抽取，指示灯的亮灯、熄灯。

```
DEFINT STAT, AUTOSTAT

STAT=SYSSTATE
AUTOSTAT = &H0010 AND STAT
IF AUTOSTAT = 0 THEN
   LIGHTBUTTON1.state = 0
ELSE
   LIGHTBUTTON1.STATE = 1
END IF
```



2.4 换页

换页用PAGE_CHANGE指令进行。通过该指令，也可以进行向同一文件夹内的其他画面移动和向其他文件夹的画面移动。
用以下的格式进行指定。

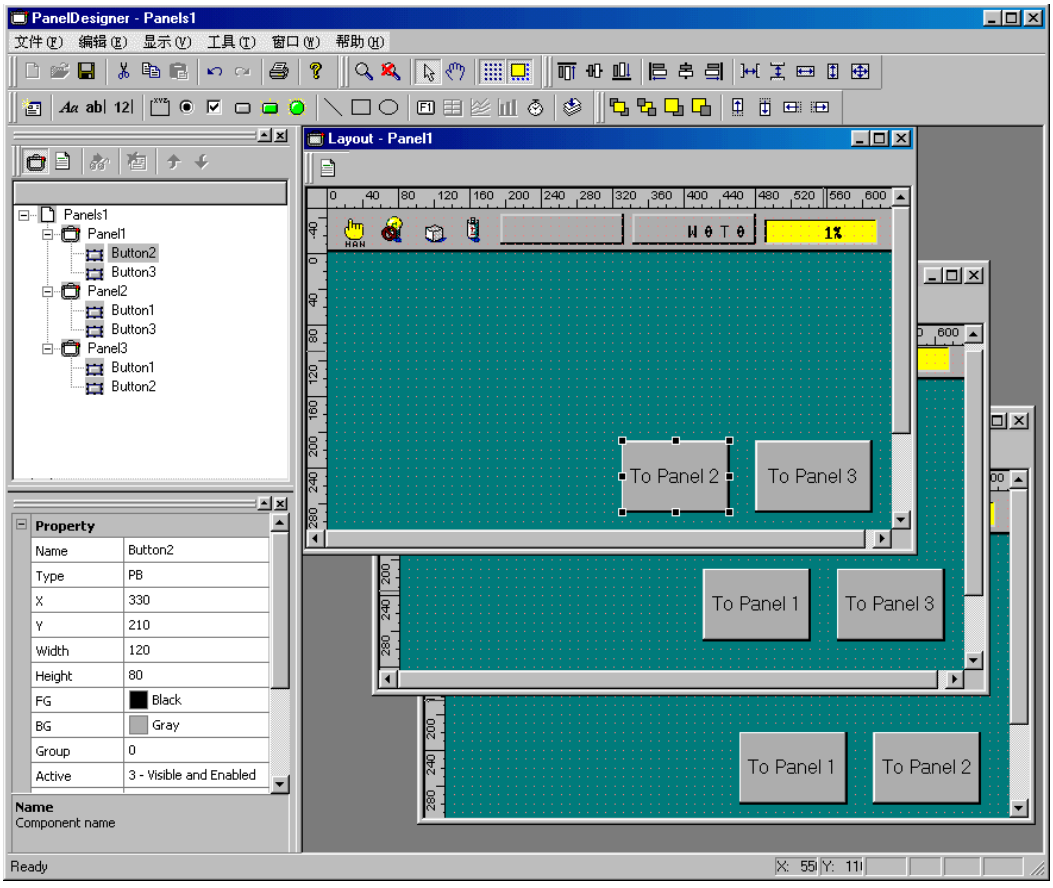
在同一文件夹内的移动： PAGE_CHANGE 面板名称
向其他文件夹的移动： PAGE_CHANGE 路径名．面板名称
移动到路径文件夹： PAGE_CHANGE ¥面板名称

2.4.1 页面移动的示例

以下为当按钮被按下时进行同一文件夹内3个画面的移动的示例。
在各自的画面上创建成有2个向其他画面进行移动的按钮。

步骤 1

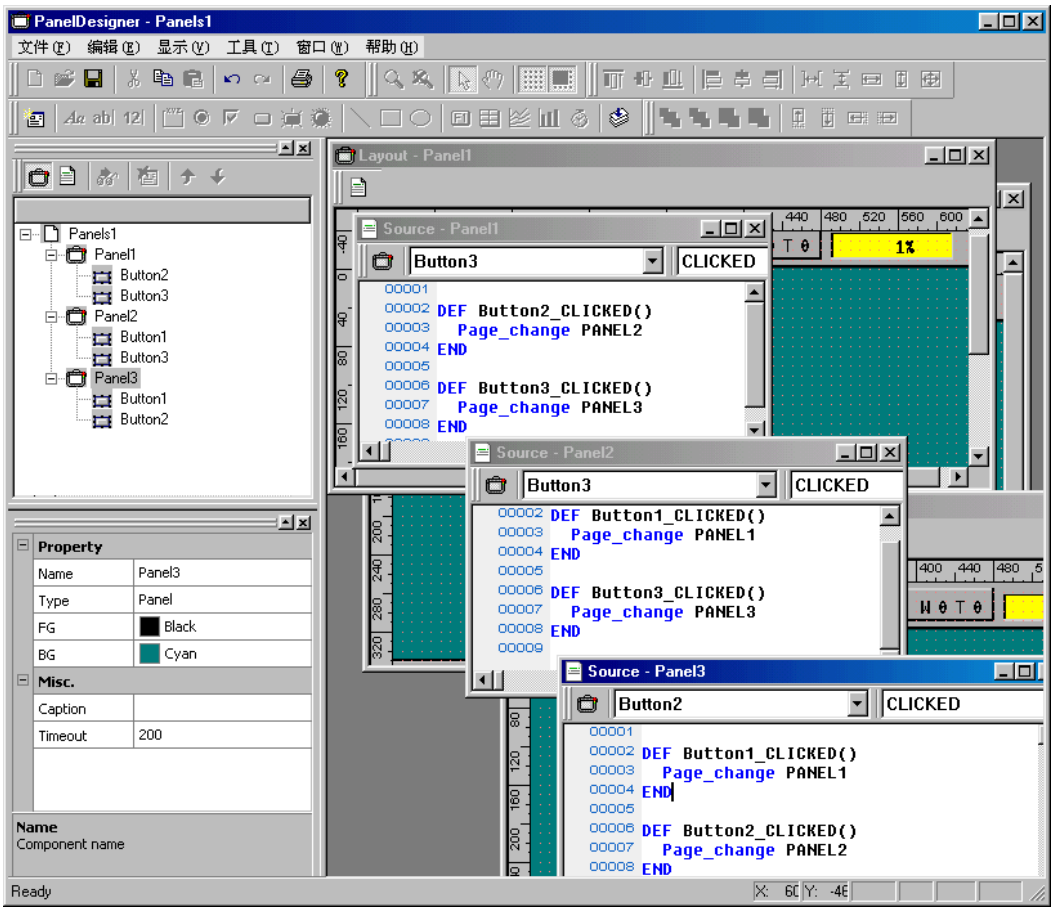
用操作盘编辑程序配置3个画面和每个画面上各2个按钮，并将按钮的显示字符串分别置为其他画面的名称等。



步骤 2

在源程序编辑画面上记述各个按钮被按下时向其他画面移动的处理。

```
PAGE_CHANGE PANEL1      '向面板名称 PANEL1 的画面移动
PAGE_CHANGE PANEL2      '向面板名称 PANEL2 的画面移动
PAGE_CHANGE PANEL3      '向面板名称 PANEL3 的画面移动
```



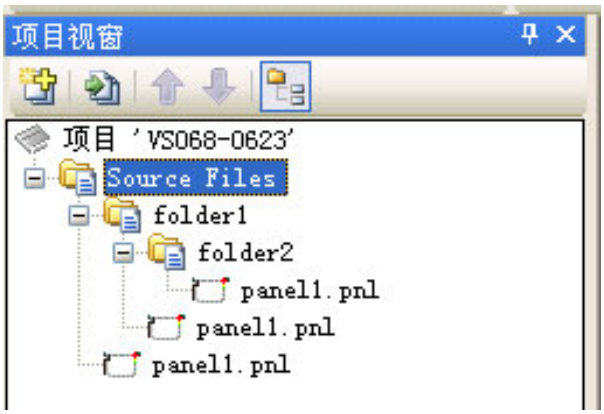
由此可以进行3个画面之间的移动。

2.4.2 文件夹之间移动的示例

以下为当按钮被按下时进行3层结构文件夹的画面移动的示例。
在各自的画面上创建成具有向其他画面移动的按钮。

步骤 1

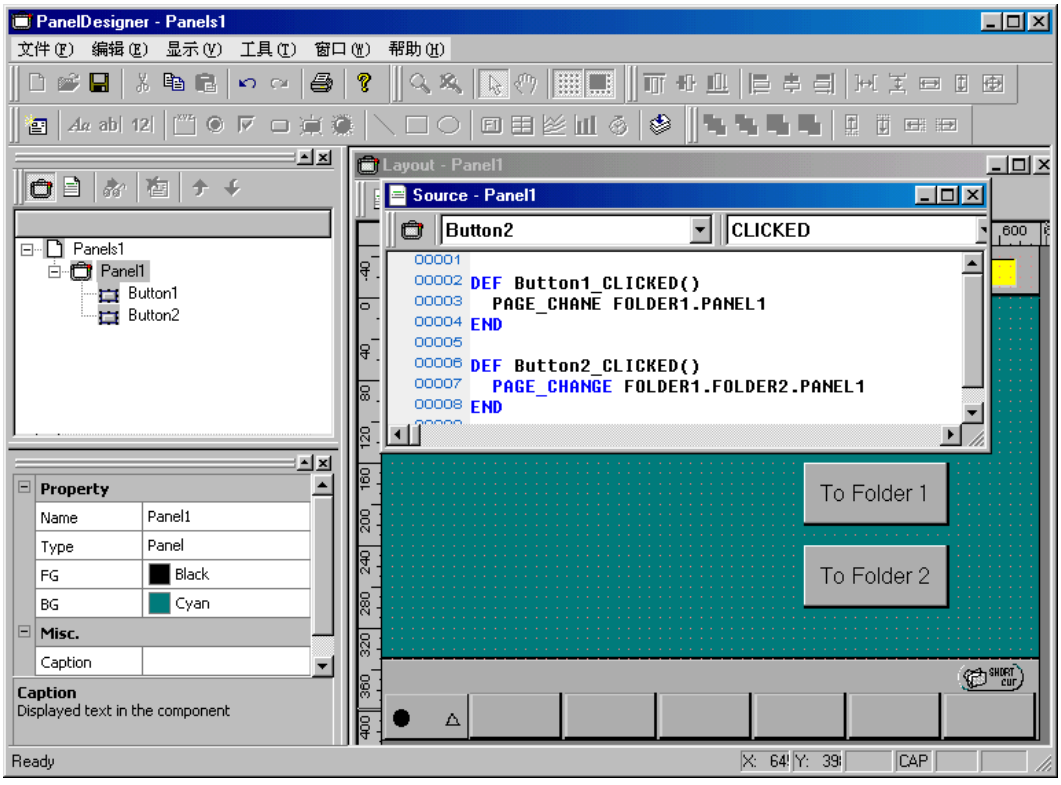
用WINCAPSIII创建3层结构的文件夹，用操作盘编辑程序在各个文件夹中创建画面，在各个画面上配置各2个按钮，并将按钮的显示字符串分别置为其他画面的名称等。



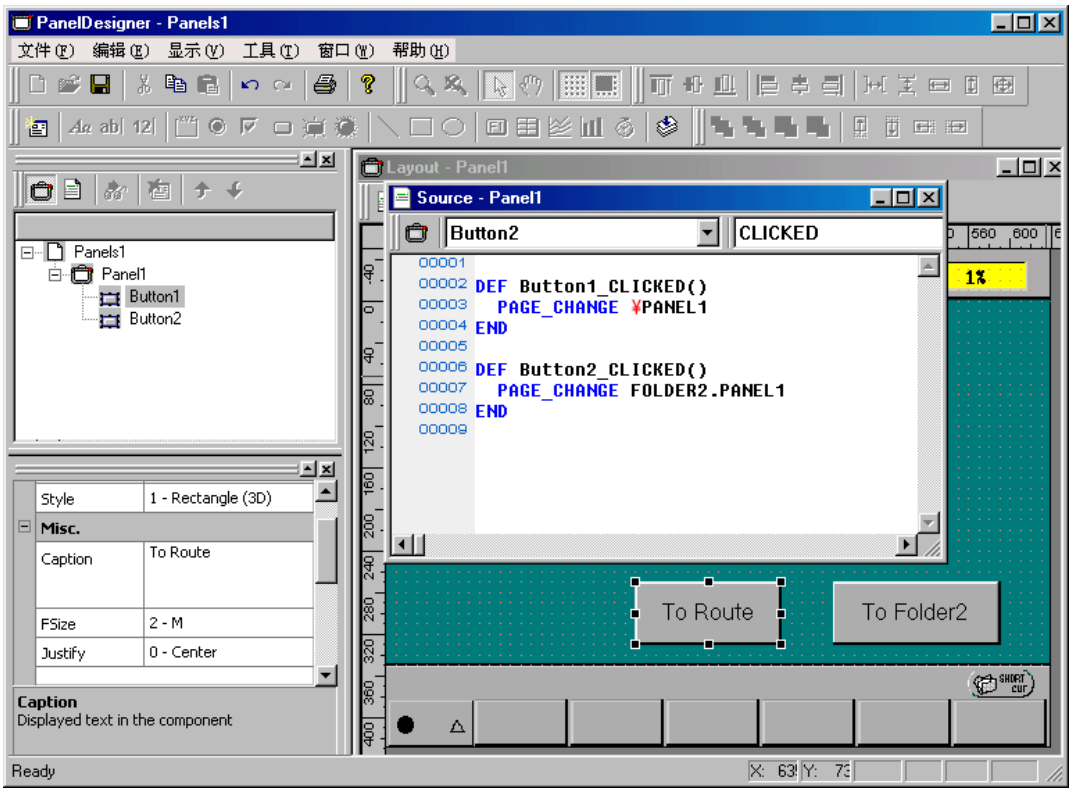
步骤 2

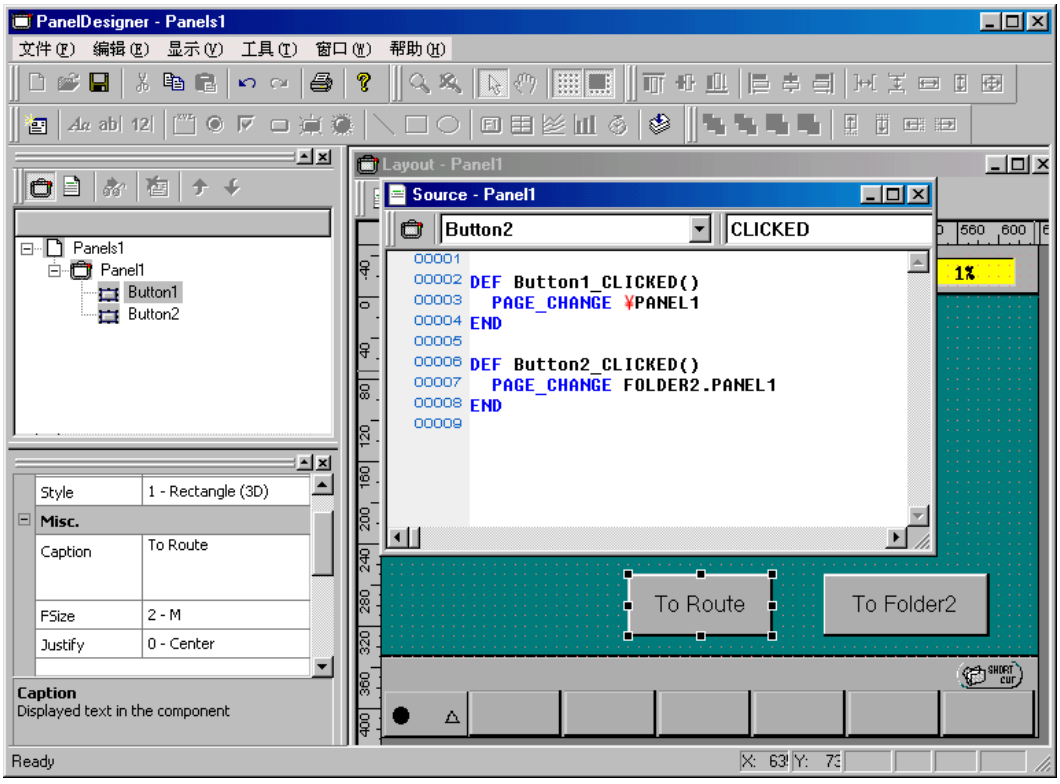
在源程序编辑画面上记述当按钮被按下时的处理（换页）。

```
PAGE_CHANGE FOLDER1.PANEL1          '向 FOLDER1 的 PANEL1 的画面相对移动
PAGE_CHANGE FOLDER2.PANEL1          '向 FOLDER2 的 PANEL1 的画面相对移动
PAGE_CHANGE FOLDER1.FOLDER2.PANEL1      '向 FOLDER1、FOLDER2 的
                                        'PANEL1 的画面相对移动
PAGE_CHANGE \PANEL3                 '向路径文件夹的面板名为 PANEL1
                                    '的画面绝对移动
```



步骤2
（续）





由此可以进行不同文件夹的3个画面之间的移动。

2.5　流程控制

在操作盘控制语言中作为流程控制语句可以使用进行条件分支的IF～END IF、IF THEN ELSE、进行重复的FOR～NEXT。
各示例如下。

2.5.1　条件分支

■IF～END IF的示例

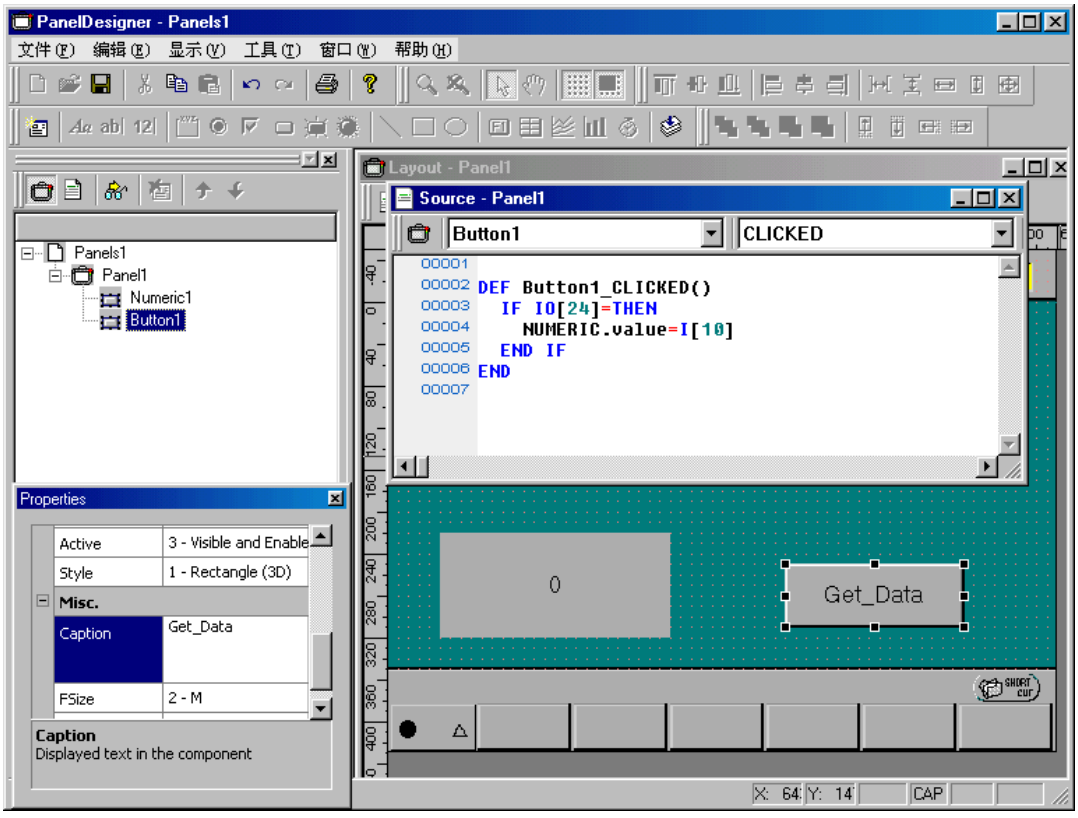
以下为使用IF语句，通过I / O状态有选择地进行将全局变量的值向数值输入框写入的示例。

步骤 1

用操作盘编辑程序配置数值输入框和指定输入动作的按钮。

步骤 2

在源程序编辑画面上记述：当按钮被按下时若I / O 24号为ON则输入全局Ｉ型变量1 / 0号，若为OFF则不做处理。



由此可以实现有选择地输入数据的功能。

■SELECT～CASE的示例

在此，使其选择作为使用SELECT～CASE语句有选择地进行处理的示例，对应数值输入框的值使之启动的程序。

步骤 1

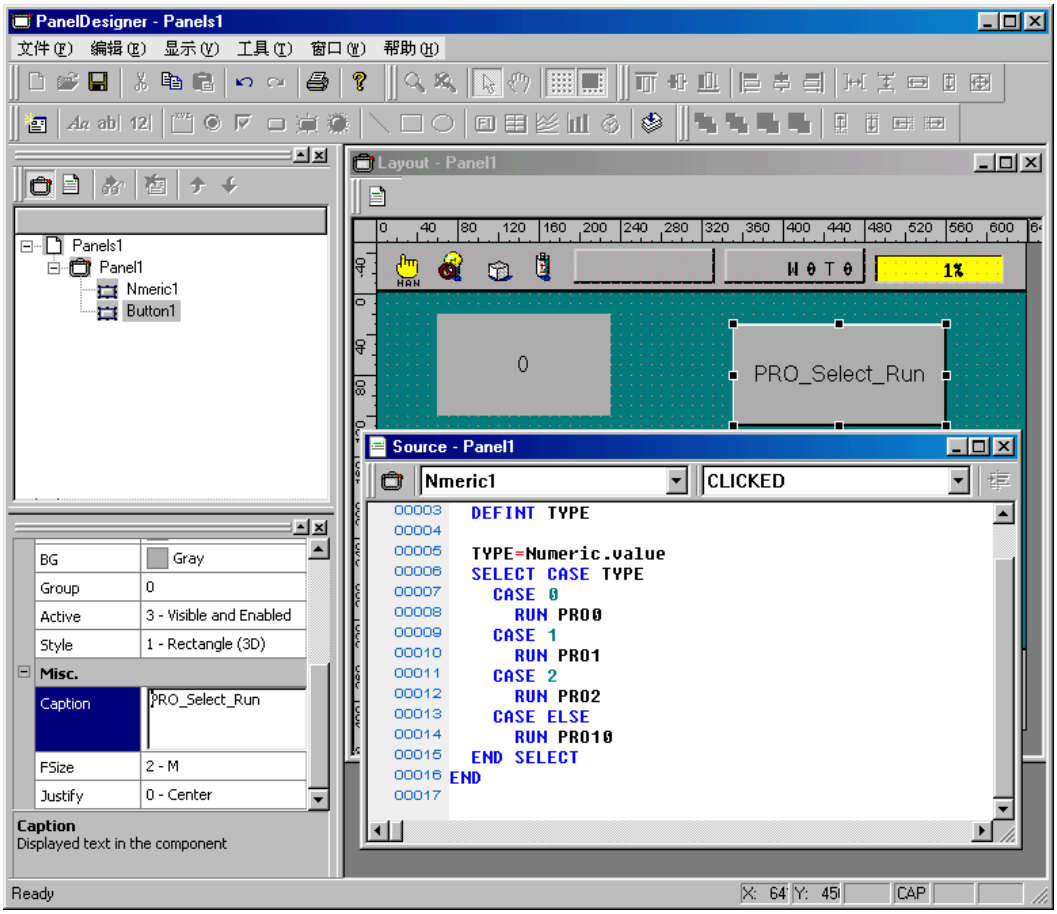
用操作盘编辑程序配置当被按下时有选择地使程序启动的按钮和数值输入框。

步骤 2

接下来，在源程序编辑画面上记述按下按钮时的处理。

```
DEFINT TYPE

TYPE = Numeric1.value
SELECT CASE TYPE
  CASE 0
    RUN PRO0
  CASE 1
    RUN PRO1
  CASE 2
    RUN PRO2
  CASE ELSE
    RUN PRO10
END SELECT
```



由此可以实现有选择地启动程序的功能。

2.5.2　反复

■FOR～NEXT 的示例

以下为使用FOR～NEXT语句，统计全局I型变量的0号～99号之中有几个0，并将其显示在数值输入框中的示例。

步骤 1

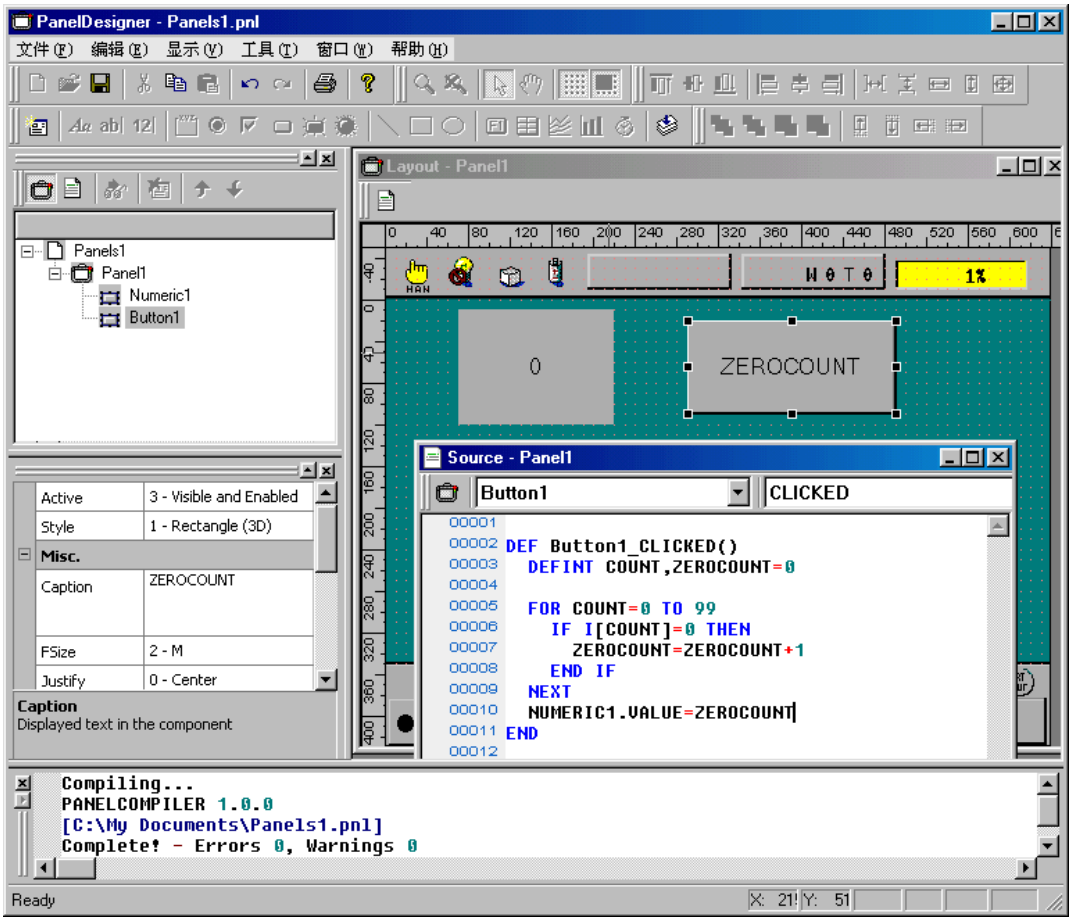
用操作盘编辑程序配置当被按下时获取数据，并统计0的个数的按钮和显示计数结果的数值输入框。

步骤 2

接下来，在源程序编辑画面上记述按下按钮时的处理。

```
DEFINT COUNT, ZEROCOUNT = 0

FOR COUNT = 0 TO 99
   IF I[COUNT] = 0 THEN
      ZEROCOUNT = ZEROCOUNT + 1
   END IF
NEXT
NUMERIC1.VALUE = ZEROCOUNT
```



由此可以实现计数全局变量中的0的个数的功能。

2.6 局部 (Local) 变量

在操作盘控制语言中作为局部 (Local) 变量可以使用的有整数型、单精度实数型、双精度实数型、字符串型、I / O型。
局部 (Local) 变量仅在进行定义的各个零部件的事件程序内进行访问。

```
DEF Button1_CLICKED ()
   DEF Button1_CLICKED ()
   DEFINT COUNT, ZEROCOUNT = 0

   FOR COUNT = 0 TO 99
      IF I [COUNT] = 0 THEN
         ZEROCOUNT = ZEROCOUNT + 1
      END IF
   NEXT
   NUMERIC1.VALUE = ZEROCOUNT


   END
```

COUNT，ZEROCOUNT
的有效范围

■局部 (Local) 变量的使用示例

以下为当按钮被按下时，将全局变量输入到局部 (Local) 变量，进行数据处理后再将数据写回全局变量功能的示例。

步骤 1 用操作盘编辑程序配置当被按下时进行数据处理的按钮。

步骤 2

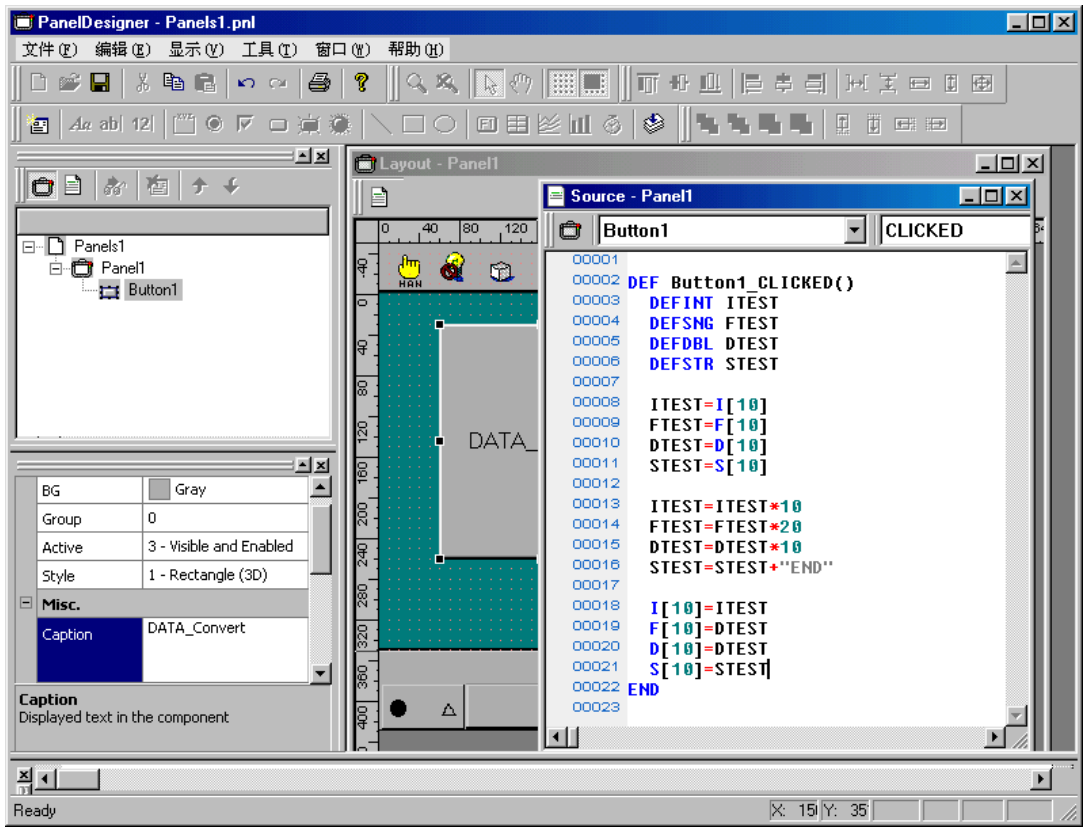
在源程序编辑画面上记述：将全局变量I型10号的数据输入到局域整数变量，扩大10倍后写回到全局变量Ｉ型10号的处理；将全局变量F型10号的数据输入到局域单精度实数型变量，扩大20倍后写回到全局变量F型10号的处理；将全局变量D型10号的数据输入到局域双精度实数变量，扩大30倍后写回到全局变量D型10号的处理；将全局变量S型10号的数据输入到局域字符串型变量，连接字符串 "end" 后写回到全局变量S型10号的处理。

```
DEFINT ITEST
DEFSNG FTEST
DEFDBL DTEST
DEFSTR STEST

ITEST = I [10]
FTEST = F [10]
DTEST = D [10]
STEST = S [10]

ITEST = ITEST * 10
FTEST = FTEST * 20
DTEST = DTEST * 10
STEST = STEST + "END"

I [10] = ITEST
F [10] = FTEST
D [10] = DTEST
S [10] = STEST
```



由此可以实现使用局部 (Local) 变量进行数据处理的功能。

第3章 操作盘控制语言的构成要素

3.1 语言要素

构成操作盘控制语言的要素中有以下内容。

●标识符.....为了识别构成要素的名称
●变　量.....用于暂时存储数据
●常　量.....具有固定数值的数据
●运算符.....在2个值之间进行计算的符号
●　式　.....求取值所需的构成要素的组合
●指　令.....为了执行处理而编入PAC语言的命令

3.2 名称

在操作盘记述语言中有为了识别程序中的各种各样的要素的规则。
在本章中，就该规则进行说明。而且，表示指令、变量的名称需遵守以下的规则。

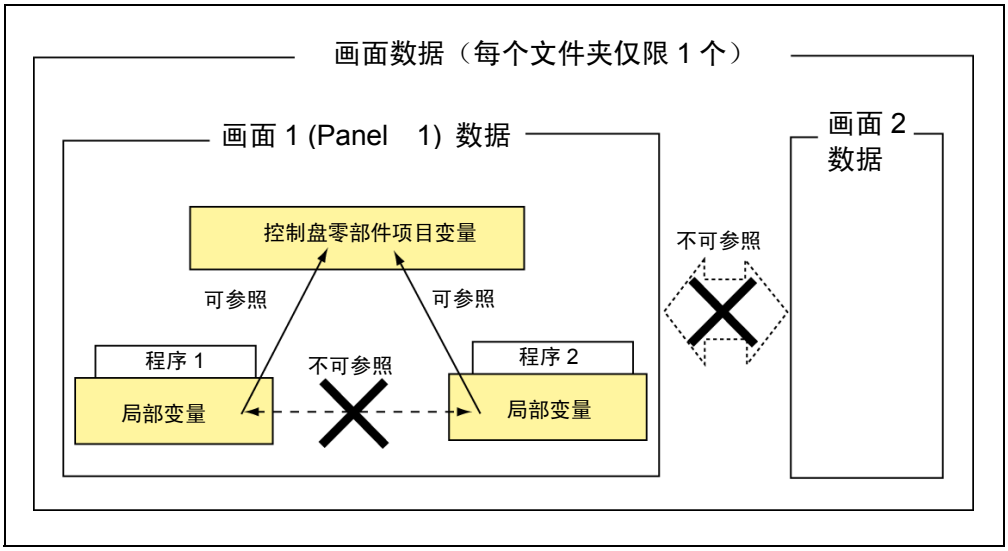
●名称必须用字母（半角英文字母，没有大小写的区分）或者规定的符号开头。
●名称里可以使用字符、数字、下划线。
　名称的开头字符必须是字母。
●不能使用句号、前括号、后括号、空格、冒号、分号、单引号、双引号、星号。
●不能使用+、-、*、/、(、) 等作为运算符使用的字符。
●为了将名称和其他的语言识别开，要在名称与其他语言之间插入空格字符。
●名称所能使用的字符数最大为64字符。

3.3 标识符变量

3.3.1 变量

变量是指将程序中使用的数据暂时存放。有全局变量和局部 (Local) 变量、操作盘零部件对象变量。
全局变量是从任何一个操作盘程序都能共同使用的变量。
局部 (Local) 变量是只在一个程序中有效的变量。
在同时被执行的其他程序中，即使有名称相同的局部 (Local) 变量，也会在各自所属的程序中运行，互不影响。
操作盘零部件对象变量是仅在同一个操作盘文件中有效的变量。

零部件对象与程序的关系如图所示。



操作盘零部件对象变量

3.3.2 全局变量

全局变量的名称，在表示类型的字母 (I、F、D、S、IO) 之后加上括号 [ ] 来表示。
只有I / O变量为2个字母 (IO)。
变量名因在系统中被预约，所以无需定义即可使用。
在全局变量中可以使用以下所示的类型。

● I型：整数型（范围：-2147483648～+2147483647）
例）I [1]
● F型：单精度实数型 (-3.402823E + 38~3.402823E + 38)
例）F [1]
● D型：双精度实数型
(-1.7976931348623157E+308～1.7976931348623157E + 308)
例）D [1]
● S型：字符串型（最大为243个字符）
例）S [1]
● IO型：I / O型
例）IO [1]

3.3.3 局部 (Local) 变量

和全局变量相同，局域变量可以使用以下类型的变量。

- I型：整数型（范围：-2147483648～+2147483647）
- F型：单精度实数型 (-3.402823E+38～3.402823E+38)
- D型：双精度实数型(-1.7976931348623157E + 308～1.7976931348623157E + 308）
- S型：字符串型（最大为243个字符)
- IO型：I / O型

局部 (Local) 变量使用类型定义命令，在类型定义之后才可以使用。

注：与PAC语言不同，间接参照、后置词是不能使用的。

3.3.4 操作盘零部件对象变量

用于变更参照在操作盘编辑程序中创建的操作盘画面的零部件属性的变量按如下所示使用。

对象变量以 对象名．属性 的形式进行变更、获取。

例：

①将零部件名 Button1 的字幕变更为 "按钮"。

```
Button1.caption = "按钮"
```

②将零部件名 LightButton1 的 状态输入 I [1]。

```
I [1] = LightButton1.state
```

各个零部件所具有的变量和可以取得的值、类型如下表所示。

操作盘零部件对象变量

零部件种类	变量名称 / 类型	意思	备注
按钮	X / 整数	左上 X 坐标	描绘区域的 X 轴基准位置：TP 描绘范围内
动作	Y / 整数	左上 Y 坐标	描绘区域的 Y 轴基准位置：TP 描绘范围内
CLICKED RELEASED	Width / 整数	宽度	• 以 X 位置为基准左侧 (-)、右侧 (+) • Y 位置+高度处于 Y 位置范围内
	Height / 整数	高度	• 以 Y 位置为基准上 (-)、下 (+) • X 位置+高度处于 X 位置范围内
	Fg / 整数	前景色	-1: White, 0: Black, 1: Blue, 2: Green, 3: Cyan, 4: Red, 5: Magenta, 6: Brown, 7: LightGray, 8: Gray, 9: LightBlue, 10: LightGreen, 11: LightCyan, 12: LightRed, 13: LightMagenta, 14: Yellow
	Bg / 整数	背景色	↑（与 Fg 相同）
	Group / 整数	组编号	所属的组编号
	Active / 整数	有效 / 无效设定	0：非显示、无效、1：显示、无效、2：非显示、有效、3：显示、有效 注：仅限 "3" 发生动作
	Style / 整数	显示形式	0：2D 四边形、1：3D 四边形、2：2D 圆形、3：3D 圆形
	Caption / 字符串	显示字符串	字符串：到半角 80 个字符为止
	Fsize / 整数	字体大小	0：极小、1：小、2：标准、3：大
	Justify / 整数	显示文字位置	0：中央、1：居右、2：居左

零部件种类	变量名称 / 类型	意思	备注
标签	X / 整数	左上 X 坐标	描绘区域的 X 轴基准位置：TP 描绘范围内
	Y / 整数	左上 Y 坐标	描绘区域的 Y 轴基准位置：TP 描绘范围内
	Width / 整数	宽度	• 以 X 位置为基准左侧 (-)、右侧 (+) • Y 位置+高度处于 Y 位置范围内
	Height / 整数	高度	• 以 Y 位置为基准上 (-)、下 (+) • X 位置+高度处于 X 位置范围内
	Fg / 整数	前景色	-1: White, 0: Black, 1: Blue, 2: Green, 3: Cyan, 4: Red, 5: Magenta, 6: Brown, 7: LightGray, 8: Gray, 9: LightBlue, 10: LightGreen, 11: LightCyan, 12: LightRed, 13: LightMagenta, 14: Yellow
	Bg / 整数	背景色	↑（与 Fg 相同）
	Group / 整数	组编号	所属的组编号
	Active / 整数	有效 / 无效设定	0：非显示、1：显示
	Caption / 字符串	显示字符串	字符串：到半角 80 个字符为止
	Fsize / 整数	字体大小	0：极小、1：小、2：标准、3：大
	Justify / 整数	显示文字位置	0：中央、1：居右、2：居左
指示灯	X / 整数	左上 X 坐标	描绘区域的 X 轴基准位置：TP 描绘范围内
动作 REFRESH	Y / 整数	左上 Y 坐标	描绘区域的 Y 轴基准位置：TP 描绘范围内
	Width / 整数	宽度	• 以 X 位置为基准左侧 (-)、右侧 (+) • Y 位置+高度处于 Y 位置范围内
	Height / 整数	高度	• 以 Y 位置为基准上 (-)、下 (+) • X 位置+高度处于 X 位置范围内
	Fg / 整数	前景色	-1: White, 0: Black, 1: Blue, 2: Green, 3: Cyan, 4: Red, 5: Magenta, 6: Brown, 7: LightGray, 8: Gray, 9: LightBlue, 10: LightGreen, 11: LightCyan, 12: LightRed, 13: LightMagenta, 14: Yellow
	Bg / 整数	背景色	↑（与 Fg 相同）
	Group / 整数	组编号	所属的组编号
	Active / 整数	有效 / 无效设定	0：非显示、1：显示
	Style / 整数	显示形式	0：2D 四边形、1：3D 四边形、2：2D 圆形、3：3D 圆形
	Caption / 字符串	显示字符串	字符串：到半角 80 个字符为止
	Fsize / 整数	字体大小	0：极小、1：小、2：标准、3：大
	Justify / 整数	显示文字位置	0：中央、1：居右、2：居左 注：当样式为 2、3 时无效。
	State / 整数	状态	0：熄灯、1：亮灯

零部件种类	变量名称 / 类型	意思	备注
线	X / 整数	左上 X 坐标	描绘区域的 X 轴基准位置：TP 描绘范围内
	Y / 整数	左上 Y 坐标	描绘区域的 Y 轴基准位置：TP 描绘范围内
	Width / 整数	宽度	• 以 X 位置为基准左侧 (-)、右侧 (+) • Y 位置+高度处于 Y 位置范围内
	Height / 整数	高度	• 以 Y 位置为基准上 (-)、下 (+) • 位置+高度处于 X 位置范围内
	Fg / 整数	前景色	-1: White, 0: Black, 1: Blue, 2: Green, 3: Cyan, 4: Red, 5: Magenta, 6: Brown, 7: LightGray, 8: Gray, 9: LightBlue, 10: LightGreen, 11: LightCyan, 12: LightRed, 13: LightMagenta, 14: Yellow
	Bg / 整数	背景色	↑（与 Fg 相同）
	Group / 整数	组编号	所属的组编号
	Active / 整数	有效 / 无效设定	0：非显示、1：显示
	Style / 整数	显示形式	0 ：实线、 1～7 ：DashStyle (dashOnOff)・PenDash (1)～(7)、 8～14 ：DashStyle (dashDouble)・PenDash (1)～(7)
	Thickness / 整数	线的粗细	"0" 的时候线宽为 "2"
数值输入 按钮 动作 CLICKED RELEASED	X / 整数	左上 X 坐标	描绘区域的 X 轴基准位置：TP 描绘范围内
	Y / 整数	左上 Y 坐标	描绘区域的 Y 轴基准位置：TP 描绘范围内
	Width / 整数	宽度	• 以 X 位置为基准左侧 (-)、右侧 (+) • Y 位置+高度处于 Y 位置范围内
	Height / 整数	高度	• 以 Y 位置为基准上 (-)、下 (+) • X 位置+高度处于 X 位置范围内
	Fg / 整数	前景色	-1: White, 0: Black, 1: Blue, 2: Green, 3: Cyan, 4: Red, 5: Magenta, 6: Brown, 7: LightGray, 8: Gray, 9: LightBlue, 10: LightGreen, 11: LightCyan, 12: LightRed, 13: LightMagenta, 14: Yellow
	Bg / 整数	背景色	↑（与 Fg 相同）
	Group / 整数	组编号	所属的组编号
	Active / 整数	有效 / 无效设定	0：非显示、无效、1：显示、无效、2：非显示、有效、3：显示、有效 注：仅限 "3" 发生动作
	Style / 整数	显示形式	0：2D、1：3D
	Caption / 字符串	显示字符串	字符串：到半角 80 个字符为止
	Fsize / 整数	字体大小	0：极小、1：小、2：标准、3：大
	Justify / 整数	显示文字位置	0：中央、1：居右、2：居左
	Value / 实数	输入值	实数：与 D 形同等

零部件种类	变量名称 / 类型	意思	备注
圆（椭圆）	X / 整数	左上 X 坐标	描绘区域的 X 轴基准位置：TP 描绘范围内
	Y / 整数	左上 Y 坐标	描绘区域的 Y 轴基准位置：TP 描绘范围内
	Width / 整数	宽度	• 以 X 位置为基准左侧 (-)、右侧 (+) • Y 位置+高度处于 Y 位置范围内
	Height / 整数	高度	• 以 Y 位置为基准上 (-)、下 (+) • X 位置+高度处于 X 位置范围内
	Fg / 整数	前景色	-1: White, 0: Black, 1: Blue, 2: Green, 3: Cyan, 4: Red, 5: Magenta, 6: Brown, 7: LightGray, 8: Gray, 9: LightBlue, 10: LightGreen, 11: LightCyan, 12: LightRed, 13: LightMagenta, 14: Yellow
	Bg / 整数	背景色	↑（与 Fg 相同）
	Group / 整数	组编号	所属的组编号
	Active / 整数	有效 / 无效设定	0：非显示、1：显示
	Style / 整数	显示形式	0 ：实线、 1～ 7 ： DashStyle (dashOnOff)・PenDash (1)～(7)、 8～14 ： DashStyle (dashDouble)・PenDash (1)～(7)
	Thickness / 整数	线的粗细	"0" 时为全部涂抹
四边形	X / 整数	左上 X 坐标	描绘区域的 X 轴基准位置：TP 描绘范围内
	Y / 整数	左上 Y 坐标	描绘区域的 Y 轴基准位置：TP 描绘范围内
	Width / 整数	宽度	• 以 X 位置为基准左侧 (-)、右侧 (+) • Y 位置+高度处于 Y 位置范围内
	Height / 整数	高度	• 以 Y 位置为基准上 (-)、下 (+) • X 位置+高度处于 X 位置范围内
	Fg / 整数	前景色	-1: White, 0: Black, 1: Blue, 2: Green, 3: Cyan, 4: Red, 5: Magenta, 6: Brown, 7: LightGray, 8: Gray, 9: LightBlue, 10: LightGreen, 11: LightCyan, 12: LightRed, 13: LightMagenta, 14: Yellow
	Bg / 整数	背景色	↑（与 Fg 相同）
	Group / 整数	组编号	所属的组编号
	Active / 整数	有效 / 无效设定	0：非显示、1：显示
	Style / 整数	显示形式	0 ：实线、 1～7 ： DashStyle (dashOnOff)・PenDash (1)～(7)、 8～14 ： DashStyle (dashDouble)・PenDash (1)～(7)
	Thickness / 整数	线的粗细	"0" 时为全部涂抹

零部件种类	变量名称 / 类型	意思	备注
文本框	X / 整数	左上 X 坐标	描绘区域的 X 轴基准位置：TP 描绘范围内
动作 CLICKED RELEASED	Y / 整数	左上 Y 坐标	描绘区域的 Y 轴基准位置：TP 描绘范围内
	Width / 整数	宽度	• 以 X 位置为基准左侧 (-)、右侧 (+) • Y 位置+高度处于 Y 位置范围内
	Height / 整数	高度	• 以 Y 位置为基准上 (-)、下 (+) • X 位置+高度处于 X 位置范围内
	Fg / 整数	前景色	-1: White, 0: Black, 1: Blue, 2: Green, 3: Cyan, 4: Red, 5: Magenta, 6: Brown, 7: LightGray, 8: Gray, 9: LightBlue, 10: LightGreen, 11: LightCyan, 12: LightRed, 13: LightMagenta, 14: Yellow
	Bg / 整数	背景色	↑（与 Fg 相同）
	Group / 整数	组编号	所属的组编号
	Active / 整数	有效 / 无效设定	0：非显示、无效、1：显示、无效、2：非显示、有效、 3：显示、有效、 注：仅限 "3" 发生动作
	Style/整数	显示形式	0：2D、1：3D
	Caption/字符串	显示字符串	字符串：到半角 80 个字符为止
	Fsize / 整数	字体大小	0：极小、1：小、2：标准、3：大
	Justify / 整数	显示文字位置	0：中央、1：居右、2：居左
	Text / 整数	输入文本	字符串：与 S 形同等 注：请注意虽然可以进行 2 字节 (Byte) 码的显示，但因在教导器上不能进行编辑，所以在教导器上编辑时 2 字节 (Byte) 码不能被正确显示。
组	X / 整数	左上 X 坐标	描绘区域的 X 轴基准位置：TP 描绘范围内
	Y / 整数	左上 Y 坐标	描绘区域的 Y 轴基准位置：TP 描绘范围内
	Width / 整数	宽度	• 以 X 位置为基准左侧 (-)、右侧 (+) • Y 位置+高度处于 Y 位置范围内
	Height / 整数	高度	• 以 Y 位置为基准上 (-)、下 (+) • X 位置+高度处于 X 位置范围内
	Fg / 整数	前景色	-1: White, 0: Black, 1: Blue, 2: Green, 3: Cyan, 4: Red, 5: Magenta, 6: Brown, 7: LightGray, 8: Gray, 9: LightBlue, 10: LightGreen, 11: LightCyan, 12: LightRed, 13: LightMagenta, 14: Yellow
	Bg / 整数	背景色	↑（与 Fg 相同）
	Group / 整数	组编号	所属的组编号
	Active / 整数	有效 / 无效设定	0：非显示、1：显示
	Caption / 字符串	显示字符串	字符串：到半角 80 个字符为止
	Fsize / 整数	字体大小	0：极小、1：小、2：标准、3：大
	Justify / 整数	显示文字位置	0：中央、1：居右、2：居左
	Thickness / 整数	线的粗细	"0" 的时候线宽为 "2"
	Mygroup / 整数	所管理的组	所管理的组编号

零部件种类	变量名称 / 类型	意思	备注
文本框	X / 整数	左上 X 坐标	描绘区域的 X 轴基准位置：TP 描绘范围内
动作 CLICKED RELEASED	Y / 整数	左上 Y 坐标	描绘区域的 Y 轴基准位置：TP 描绘范围内
	Width / 整数	宽度	• 以 X 位置为基准左侧 (-)、右侧 (+) • Y 位置+高度处于 Y 位置范围内
	Height / 整数	高度	• 以 Y 位置为基准上 (-)、下 (+) • X 位置+高度处于 X 位置范围内
	Fg / 整数	前景色	-1: White, 0: Black, 1: Blue, 2: Green, 3: Cyan, 4: Red, 5: Magenta, 6: Brown, 7: LightGray, 8: Gray, 9: LightBlue, 10: LightGreen, 11: LightCyan, 12: LightRed, 13: LightMagenta, 14: Yellow
	Bg / 整数	背景色	↑（与 Fg 相同）
	Group / 整数	组编号	所属的组编号
	Active / 整数	有效/无效设定	0：非显示、无效、1：显示、无效、2：非显示、有效、 3：显示、有效 注：仅限 "3" 发生动作
	Style / 整数	显示形式	0：2D 检查、1：3D 检查、2：3D 按钮、3：3D 按钮 注：2D 肘节因不能描绘所以为 3D。
	Caption / 字符串	显示字符串	字符串：到半角 80 个字符为止 注：请注意当被 Caption 存放的字符串的长度较长时， 按钮上的字符串会重叠。
	Fsize / 整数	字体大小	0：标准、1：小、2：大
	Justify / 整数	显示文字位置	0：中央、1：居右、2：居左
	State / 整数	状态	0：OFF、1：ON

零部件种类	变量名称 / 类型	意思	备注
无线电按钮	X / 整数	左上 X 坐标	描绘区域的 X 轴基准位置：TP 描绘范围内
动作 CLICKED RELEASED	Y / 整数	左上 Y 坐标	描绘区域的 Y 轴基准位置：TP 描绘范围内
	Width / 整数	宽度	• 以 X 位置为基准左侧 (-)、右侧 (+) • Y 位置+高度处于 Y 位置范围内
	Height / 整数	高度	• 以 Y 位置为基准上 (-)、下 (+) • X 位置+高度处于 X 位置范围内
	Fg / 整数	前景色	-1: White, 0: Black, 1: Blue, 2: Green, 3: Cyan, 4: Red, 5: Magenta, 6: Brown, 7: LightGray, 8: Gray, 9: LightBlue, 10: LightGreen, 11: LightCyan, 12: LightRed, 13: LightMagenta, 14: Yellow
	Bg / 整数	背景色	↑（与 Fg 相同）
	Group / 整数	组编号	所属的组编号
	Active / 整数	有效 / 无效设定	0：非显示、无效、1：显示、无效、2：非显示、有效、 3：显示、有效 注：仅限 "3" 发生动作
	Style / 整数	显示形式	0：2D 无线电、1：3D 无线电、2：3D 按钮、3：3D 按钮 注：2D 肘节因不能描绘所以为 3D。
	Caption / 字符串	显示字符串	字符串：到半角 80 个字符为止 注：请注意当被 Caption 存放的字符串的长度较长时， 按钮上的字符串会重叠。
	Fsize / 整数	字体大小	0：极小、1：小、2：标准、3：大
	Justify / 整数	显示文字位置	0：中央、1：居右、2：居左
	State / 整数	状态	0：OFF、1：ON
功能键配置	Caption / 字符串	显示字符串	字符串
动作 CLICKED	Index / 整数	功能键编号	功能键的编号从 1 至 12 注：请注意在程序中虽然可以变更， 但是获取、变更是不能的。
计时器	X / 整数	左上 X 坐标	描绘区域的 X 轴基准位置：TP 描绘范围内
动作 TIMER	Y / 整数	左上 Y 坐标	描绘区域的 Y 轴基准位置：TP 描绘范围内
	Group / 整数	组编号	所属的组编号
	Active / 整数	有效 / 无效设定	0：无效、1：有效
	Interval / 整数	定时器间隔	设定发生行动的间隔。 单位：msec

零部件种类	变量名称 / 类型	意思	备注
照明式按钮	X / 整数	左上 X 坐标	描绘区域的 X 轴基准位置：TP 描绘范围内
动作 CLICKED RELEASED REFRESH	Y / 整数	左上 Y 坐标	描绘区域的 Y 轴基准位置：TP 描绘范围内
	Width / 整数	宽度	• 以 X 位置为基准左侧 (-)、右侧 (+) • Y 位置+高度处于 Y 位置范围内
	Height / 整数	高度	• 以 Y 位置为基准上 (-)、下 (+) • X 位置+高度处于 X 位置范围内
	Fg / 整数	前景色	-1: White, 0: Black, 1: Blue, 2: Green, 3: Cyan, 4: Red, 5: Magenta, 6: Brown, 7: LightGray, 8: Gray, 9: LightBlue, 10: LightGreen, 11: LightCyan, 12: LightRed, 13: LightMagenta, 14: Yellow
	Bg / 整数	背景色	↑（与 Fg 相同）
	Group / 整数	组编号	所属的组编号
	Active / 整数	有效 / 无效设定	0：非显示、无效、1：显示、无效、2：非显示、有效、3：显示、有效 注：仅限 "3" 发生 CLICKED、RELEASED 动作
	Style / 整数	显示形式	0：2D 四边形、1：3D 四边形、2：2D 圆形、3：3D 圆形
	Caption / 字符串	显示字符串	字符串：到半角 80 个字符为止
	Fsize / 整数	字体大小	0：极小、1：小、2：标准、3：大
	Justify / 整数	显示文字位置	0：中央、1：居右、2：居左 注：当样式为 2、3 时无效。
	State / 整数	状态	0：熄灯、1：亮灯

3.3.5 文件夹变量

从操作盘程序可以进行文件夹变量的参照、变更。
但是，需要事先在同一文件夹内的PAC程序中定义文件夹变量。

为了使用文件夹变量附加上EXTERN定义来进行类型定义。

```
例：EXTERN   DEFINT   AAA          '定义变量名AAA的文件夹变量。
```

文件夹变量的值参照、值变更与通常的变量一样进行。

```
例：AA   =   LightButton1.state     '将指示灯LightButton1的状态输入
                                    '文件夹变量AAA。
      I[2]   =   AAA                '将文件夹变量AAA的值存放到全局
                                    '变量I [2]。
```

3.4 程序

操作盘程序以以下的形式仅对想使其进行的动作进行记述。

```
DEF　对象名__动作
 想使其进行的动作
END
```

在操作盘编辑程序中若选择对象名与行动，"DEF　对象名__动作" 和 "END" 的部分因会被自动生成，所以请仅对内部进行记述。（参照2.2.2　零部件的动作）
动作里有下表所示的项目，可以使用的动作会根据零部件的种类的不同而不同。

动作名称	动作内容
CLICKED	零部件被按下
RELEASED	零部件被放开
TIMER	经过了间隔时间
REFRESH	画面被刷新

此外，程序中的局部 (Local) 变量从其他的程序不能进行参照。

3.5 数据型

在操作盘控制语言中使用的数据里有字符串、数值、I / O的类型。
以下就这些的数据类型进行说明。

(1) 字符串

字符串 (String) 型数据也叫做S型。
最大可以包含到243个字符的字符串。

(2) 数值

在数值型数据中有以下3种类型。
I型：整数型（范围：-2147483648～+2147483647）
F型：单精度实数型 (-3.402823E+38～3.402823E+38)
D型：双精度实数型 (-1.79769313486231E+308～1.79769313486231E+308)

(3) I / O (ON / OFF)

I / O型的数据，是将I / O端口的状态（ON或OFF）作为数值持有。

3.6 数据型的转换

在不同的数据型之间，可以转换类型。

(1) 数值

数值数据可以按照以下的规则进行类型转换。

●若将数值数据赋值不同类型的数值变量，则数值会顺着变量的类型被转换。
●若类型不同的数值之间进行运算，则会顺着精度高的一方的类型进行运算。
●在逻辑运算中，数值转换为整数进行运算，结果为整数。
●当实数被转换为整数时，被取整为不超过该实数值的最大整数。
●双精度实数被赋值单精度实数的情况下，其结果被取为7位有效数字。

3.7 常量

常量是具有固定值的公式。
操作盘控制语言的常量可以进行以下分类。

数值数据	I型：整数型常量 F型：单精度实数型常量 D型：双精度实数型常量
字符串数据	S型：字符串型常量（最大243个字符）

以下对各个类型的常量进行说明。

(1) 整数型常量

是基于从-2147483648到+2147483647范围内的整数的常量。
整数型常量里有十进制、二进制的两种标示形式。

十进制形式

是用十进制数表示的整数型常量。
例：32767
-125
+10

二进制形式

是用二进制数表示的整数型常量。
在数值的前头加上 "&B"，用0或1的排列表示。
用二进制形式标示整数型常量的数值的范围的话，
会是&B0～&B11111111111111111111111111111111
（32比特的范围内）。

例：&B110
&B0011

(2) 单精度实数型常量

是具有7位有效位精度的实数型的常量。
数值的范围是-3.402823E+38～3.402823E+38。
单精度实数型常量里有以下两种标示形式。

●使用了E的指数形式
●没有上述指定，7位以下的实数
例：1256.3
-9.345E-06

(3) 双精度实数型常量

是具有15位有效位精度的实数型的常量。
数值的范围是-1.79769313486231E+308～1.79769313486231E+308。
双精度实数型常量里有以下标示形式。

●超过7位且15位以下的实数
例：1256.325468
-9.345E-06

(4) 字符串常量

字符串型常量是表示字符串的常量。
字符串用 " " 将前后括起来标示。
字符串的长度必须在128个字符以内。

例："PAC"

3.8 式与运算符

进行值返回（结果）的表现方式称为式。如常量及变量那样单独的具有值的式和通过运算符相结合的由多个要素构成的式。PAC语言的所有数据型的数值通过式表示。
若使用在以下部分说明的运算符，则可以在式中进行运算。

(1) 赋值运算符

向变量赋值，要在赋值语句使用赋值运算符 = 进行。赋值运算符的右侧的值被赋值左侧。

(2) 算术运算符

在算术运算中，使用下表所示的运算符。
运算按照表中所示的优先顺序进行。

算术运算符

算术运算符	运算内容	运算的优先顺序
^	指数（幂乘）运算	高
-	负符号	↕
*、/	乘法、除法	
MOD	余数	
+、-	加法、减法	低

算术运算符

除数 / 被除数	+	0	-
+	+	错误	+
0	0	错误	0
-	-	错误	+

(3) 关系运算符

关系运算符，用于比较2个数值。结果由布尔值（真为 "1"、伪为 "0"）取得，
用于流程控制语句的条件式等。

关系运算符

关系运算符	运算内容
=	等于
<>	不等于
<	小于
>	大于
<=	小于或等于
=.	等于（近似比较）
>=	大于或等于

(4) 逻辑运算符

逻辑运算符进行比特运算。
整数型以外的数值，要先转换成整数型后运算。

逻辑运算符

逻辑运算符	运算内容
NOT	否定
AND	逻辑积
OR	逻辑和
XOR	排他的逻辑和

例：比特运算

I1 = &B1100　XOR　&B0101

这种情况的结果为&B1001。

(5) 字符串运算符

字符串可以通过字符串运算符 "+" 进行连接。

例：A = "ABC" + "DEF"　　'A为 "ABCDEF"。

(6) 算术运算符、逻辑运算符、关系运算符的优先顺序

运算符之间的优先顺序请参照如下。

运算符	运算内容	优先顺序
^	指数（幂乘）	高
-	负符号	↑
*, /	乘法、除法	
MOD	余数	
+, -	加法、减法	
NOT	否定	
AND	逻辑积	
OR	逻辑和	
XOR	排他的逻辑和	↓
=, <>, <, >, <=, >=	关系运算符	低

优先顺序相同的情况下，公式从左向右被进行评价。
用（　）括上的情况下括弧内的运算优先被处理。（优先顺序提高）

第4章 操作盘控制语言的语法

4.1 语句和行

操作盘记述语言的程序由若干行构成。
在任意的行中都可以记述一个语句。
1行的长度最大为255字节 (Byte)。
语句是记述PAC语言处理的最小单位，由一个指令构成。
指令由指令的名称和给予指令的信息（参数）构成。

4.2 字符组

在操作盘控制语言上可以使用的字符是英文字母（不分大小写）、数字、特殊符号。
特殊记号除了算术运算符（+, -, *, / ）之外，有以下种类。

●逗号（, ）
　用于参数并列时的隔开。
●单引号（' ）
　作为REM指令的代用指令使用。
●双引号（" ）
　通过将任意的字符串的前后括起来，用于字符串数据的指定。
●空格
　在命令的名称前后是必须的。

4.3 保留字

操作盘控制语言在对指令的名称及运算符等进行处理的基础上,确定使用方法的语句称为保留字。

以下 "操作盘保留字一览" 所示的保留字不能作为变量名、面板名进行使用。

<操作盘保留字一览>

if,then,else,elseif,while,do,return,print,add_widget,msgbox,page_change, set,reset,run,kill,suspend,suspendall,killall,caption,fg,bg,timeout, defint,defsng,defdbl,defstr,defio,in,out,break,continue,var,def,pend, for,refresh,extern,begin,end,wend,next,endif,status,str$,continuerun, io,i,f,d,s,sysstate,curoptmode,time$,date$,timer,select,case,is,to, deadmanstate,sprintf$,releasemode,pnlccver,chr$,step

4.4 定义语句

诸如变量及常量、函数等需要确定名称及类型的，在程序中使用之前要用定义语句进行定义。
定义语句大致划分有以下的种类。

(1) 类型定义

定义变量以及常量的类型。

类型定义命令

使用以下的类型定义的指令，可以定义变量的类型。

类型定义命令

种类	指令	使用示例
整数型	DEFINT	DEFINT AA,AB
单精度实数型	DEFSNG	DEFSNG BA,BB
双精度实数型	DEFDBL	DEFDBL CA,CB
字符串型	DEFSTR	DEFSTR DA,DB

在这些指令中定义类型的同时，可以进行变量的初始化。

使用示例：

```
DEFINT AA = 1      '将AA作为整数型赋值1。
DEFSNG BB (10)   '将BB设为要素数10的单精度实数型。
```

(2) 排列定义

在为排列定义的定义语句中通过使用类型定义命令，指定要素数，可以对I / O变量以外的所有类型进行排列。
但是，在定义时不能进行排列的初始化。
排列的追加字必须是0以上。排列的维数最大是3维。
排列的要素总数以32767为上限。

排列定义的示例：

```
DEFINT CC (3,3,3)   '将CC进行整数型的3维排列。
```

(3) I / O 变量定义

将变量名称和特定的I / O端口相对应。

I / O变量定义

种类	指令	使用示例
I / O变量定义	DEFIO	DEFIO PORT = BYTE, 104

4.5 赋值语句

赋值语句在各个类型的变量上设定值。
大致划分为数值赋值语句、字符串赋值语句两种。

(1) 数值赋值语句

数值赋值语句是将值赋值数值变量。

例：D[2] = 3.14　　'将3.14赋值D[2]。

(2) 字符串赋值语句

字符串赋值语句是将字符串赋值字符串型变量。

例：S[2] = "DENSO"　　'将 "DENSO" 赋值S[2]。

4.6 流程控制语句

为了控制程序的各个语句的执行顺序，使用流程控制语句。
流程控制大致划分为条件分支、选择、重复3种。

(1) 条件分支

使用IF～THEN～ELSE语句或IF～END IF语句，根据条件是否成立，
决定分支目的地。
在IF之后所记述的关系式如果是真 (TRUE (1))，则执行THEN之后的程序，
如不是，则执行ELSE以后的程序。

(2) 选择

根据所指定的式的值，选择要执行的处理。

在SELECT CASE语句中，在SELECT行的CASE之后加入运算式，将满足该运算式值的CASE行以后的进行执行到下面的CASE行或END SELECT行。所有的CASE行都不相等时，将CASE ELSE以后执行到END SELECT行。

(3) 反复

根据所指定的条件，控制是否进行重复。

在FOR～NEXT语句上，在FOR行的FOR之后加入重复的条件，在满足该条件之前，重复在所对应的NEXT之前进行处理。

4.7 输出入控制语句

输入输出控制语句里有DI / DO控制语句、操作盘控制语句的3种。

(1) DI / DO 控制语句

DI / DO控制语句控制I / O端口的输入输出。

DI / DO控制指令

动作的种类	指令
数据读取	IN
数据写入	OUT
I / O端口ON	SET
I / O端口OFF	RESET

(2) 多功能教导器控制语句

多功能教导器控制语句对多功能教导器的输入输出进行设定。

多功能教导器控制指令

动作的种类	指令
信息画面输出	MSGBOX
页显示	PAGE_CHANGE
画面的再描绘	REFRESH

4.8 多项任务控制语句

多任务控制语句里有任务控制语句。

(1) 任务控制语句

任务控制语句对该任务控制语句所属任务以外的任务进行控制。

任务控制指令

动作的种类	指令
任务的生成、启动	RUN
任务的中断	SUSPEND
任务的删除	KILL
所有任务的中断	SUSPENDALL
所有任务的删除	KILLALL
连续启动	CONTINUERUN

4.9 函数

通过以下的指令对字符串进行控制。

字符串函数

动作的种类	函数
向字符串的转换	STR$
字符编码的输入	CHR$

4.10 系统信息

系统信息通过以下所示的指令可以取得。

系统信息指令

动作的种类	指令
获取程序的状态	STATUS
动作模式的获取	CUROPTMODE
控制器状态的获取	SYSSTATE

4.11 预处理器

在将程序翻译（编译）成执行形式时，预处理器语句控制字符串的替换，或文件的输入。

预处理器指令

动作的种类	指令
将常量、宏名称置换为字符串	#define
文件的取得	#include

第5章 指令参考

5.1 操作盘控制指令一览

功能区分	指令	功能	4 轴	6 轴
定义语句				
局部 (Local) 变量 I 型	DEFINT	定义整数型变量。 整数的范围为-2147483648～2147483647。	◎	◎
F 型	DEFSNG	定义单精度实数型变量。 单精度实数的范围为-3.4E-38～3.4E+38。	◎	◎
D 型	DEFDBL	定义双精度实数型变量。 双精度实数的范围为-1.7D+308～1.7D+308。	◎	◎
S 型	DEFSTR	定义字符串型变量。字符串的长度为最大 243 文字。	◎	◎
I / O 型	DEFIO	定义与输出入端口相对应的 I / O 变量。	◎	◎
流程控制语句				
反复	FOR～NEXT	循环执行位于 FOR～NEXT 之间的一系列的指令。	◎	◎
条件分支	IF～END IF	执行 IF～END IF 之间的条件判断表达式中的条件判断。	◎	◎
	SELECT CASE	执行多个条件判断。	◎	◎
输出入控制语句				
I / O 端口	IN	从通过 I / O 变量表示的 I / O 端口读入数据。	◎	◎
	OUT	向通过 I / O 变量表示的 I / O 端口输出数据。	◎	◎
	SET	将 I / O 端口置为 ON。	◎	◎
	RESET	将 I / O 端口置为 OFF。	◎	◎
TP 操作盘	MSGBOX	显示信息框。	◎	◎
	PAGE_CHANGE	进行页面的显示。	◎	◎
	REFRESH	进行画面的重新描绘。	◎	◎
多项任务控制语句				
任务控制	RUN	与其它程序同时启动。	◎	◎
	KILL	强制结束任务。	◎	◎
	SUSPEND	暂停任务。	◎	◎
	SUSPENDALL	停止特权任务以外的所有的程序。	◎	◎
	KILLALL	强制结束特权任务以外的所有的任务。	◎	◎
	DEADMANSTATE	获取双重安全开关的状态。	◎	◎
常量				
内置常量	OFF	赋予 OFF (0) 值。	◎	◎
	ON	赋予 ON (1) 值。	◎	◎
	PI	赋予π值。	◎	◎
	FALSE	赋予布尔值的假 (0) 的值。	◎	◎
	TRUE	赋予布尔值的真 (1) 的值。	◎	◎
函数				
字符串	STR$	进行向字符串的转换。	◎	◎
	CHR$	输入字符编码。	◎	◎
	SPRINTF$	转换为指定了表达式的格式，并作为字符串返回。	◎	◎
时间 / 日期控制				
时间 / 日期	DATE$	得出当前的日期。	◎	◎
	TIME$	得出当前的时间。	◎	◎
	TIMER	得出经过时间。	◎	◎
系统信息				
动作模式	STATUS	得出程序的状态。	◎	◎
	CUROPTMODE	获取动作模式。	◎	◎
	SYSSTATE	获取控制器的状态。	◎	◎

功能区分	指令	功能	4 轴	6 轴
预处理器				
符号常量、宏定义	#define	使用指定字符串替换程序中所指定的常量或宏名。	◎	◎
文件取得	#include	读入预处理程序。	◎	◎

5.2 定义语句

DEFINT（语句）

功能 定义 I 型变量（整数型变量）。整数的范围为-2147483648～2147483647。

格式 DEFINT <变量名>[=<常量>][,<变量名>[=<常量>]…]

说明 将在 <变量名> 上指定的变量作为整数型的变量进行定义。
通过在 <变量名> 之后接着写等号和常量，可以在定义的同时进行初始化。
通过用 "," 断开，可以一次定义多个变量名。

相关项目 DEFDBL、DEFSNG、DEFSTR

应用示例

```
DEFINT Iix, Iiy, Iiz          '将 Iix, Iiy, Iiz 作为Ⅰ型变量进行定义。
DEFINT Iix = 1                '将 Iix 作为Ⅰ型变量进行定义，赋予初始值 = 1。
```

DEFSNG（语句）

功能 F 型变量（定义单精度实数型变量）。单精度实数的范围为-3.4E-38～3.4E+38。

格式 DEFSNG <变量名>[=<常量>][,<变量名>[=<常量>]…]

说明 将在 <变量名> 上指定的变量作为单精度实数型进行定义。通过在 <变量名> 之后接着写等号和常量，可以在定义的同时进行初始化。
通过用 "," 断开，可以一次指定多个变量名。

相关项目 DEFDBL、DEFINT、DEFSTR

应用示例

```
DEFSNG Ifx, Ify, Ifz          '将 Ifx, Ify, Ifz 作为 F 型变量进行定义。
DEFSNG Ifx = 1.0              '将 Ifx 作为 F 型变量进行定义，并给予初始值 = 1.0。
```

DEFDBL（语句）

功能　定义 D 型变量（双精度实数型变量）。双精度实数的范围是
-1.7D+308 ～ 1.7D+308。

格式　DEFDBL　<变量名>[=<常量>][,<变量名>[=<常量>]…]

说明　将在 <变量名> 上指定的变量作为双精度实数型进行定义。通过在 <变量名> 之后接着写等号和常量，可以在定义的同时进行初始化。
通过用 "," 断开，可以一次指定多个变量名。

相关项目　DEFINT、DEFSNG、DEFSTR

应用示例

```
DEFDBL ldx, ldy, ldz        '将 ldx, ldy, ldz 作为 D 型变量进行定义。
DEFDBL ldx = 1.0            '将 ldx 作为 D 型变量进行定义，并给予初始值 = 1.0。
```

DEFSTR（语句）

功能　定义 S 型变量（字符串型变量）。字符串的长度为最大 247 文字。

格式　DEFSTR <变量名>[=<常量>][,<变量名>[=<常量>]…]

说明　将在 <变量名> 上指定的变量作为字符串型进行定义。通过在 <变量名> 之后接着写等号和常量，可以在定义的同时进行初始化。

通过用 "," 断开，可以一次指定多个变量名。

相关项目　DEFDBL、DEFINT、DEFSNG

应用示例

```
DEFSTR lsx, lsy, lsz        '将 lsx, lsy, lsz 作为 S 型变量进行定义。
DEFSTR lsx = "DENSO"        '将 lsx 作为 S 型变量进行定义，
                            '并给予初始值 = "DENSO"。
```

DEFIO（语句）

功能

定义对应输出入端口的 IO 变量（I / O 变量）。

格式

DEFIO <变量名> = <I / O 变量的类型>,<端口地址>[,<掩码信息>]

说明

将 <变量名> 上所指定变量作为 I / O 变量进行定义。

<I / O 变量的类型>	选择 I / O 变量的类型。I / O 变量的类型有 BIT、BYTE、WORD、INTEGER、SINGLE。BIT 型指定 1 比特、BYTE 型指定 8 比特、WORD 型指定 16 比特、INTEGER 型指定 32 比特、SINGLE 型指定 32 比特的范围。 SINGLE 型在 Ver.3.2 不足时不能使用。
<端口地址>	指定输入输出端口上的开始编号。
<掩码信息>	输入端口的情况下，取输入数据和掩码信息的 AND。 输出端口的情况下，取输出数据和掩码信息的 AND 进行输出，但掩码未被设定的比特的输出状态不变化。 当为 SINGLE 时，掩码信息无效。只要记述掩码信息，编译就会出错。

相关项目

IN、OUT、SET、RESET

应用示例

```
DEFIO samp1 = BIT, 1                    '将 samp1 作为从端口 1 开始的 BIT 型 I / O 变量进行
                                        '定义。samp1 的反馈值表示端口 1 的状态，
                                        '为 0 或 1 的 1 比特整数。
DEFIO samp2 = BYTE, 10, &B00010000      '将 samp2 添加从端口 10 开始的 BYTE 型 I / O 变量屏蔽
                                        '信息之后进行定义。Samp2 的反馈值表示端口 10 的
                                        '状态，为 0 或 16 的 8 比特整数。
DEFIO samp3 = WORD, 15                  '将 samp3 作为从端口 15 开始的 WORD 型 I / O 变量
                                        '进行定义。Samp3 的反馈值表示从端口 15 到 30 的状态，
                                        '为 0～&Hffff 的 16 比特整数。
DEFIO samp4 = INTEGER, 1                '将 samp4 作为从端口 1 开始的 INTEGER 型 I / O 变量
                                        '进行定义。Samp4 的反馈值表示从端口 1 到 32 的状态，
                                        '为 0～&Hffffffff 的 32 比特整数。
DEFIO samp6=128                         '将 samp6 作为从端口 128 开始的 SINGLE 型 I / O 变量
                                        '进行定义。Samp6 的反馈值表示从端口 128 到 159 的
                                        '状态，为-3.4E+38～3.4E+38 的 32 比特单精度实数。
```

注意事项

・当为 WORD、INTEGER 时，相当于最高比特的端口为符号比特。
SINGLE 的表现方式为 IEEE754 式。
数值的范围和相对于各种比特的端口编号如下所示。

WORD	数值范围：-32768～32767 最高比特端口编号：开始端口地址 + 15
INTEGER	数值范围：-2147483648～2147483647 最高比特端口编号：开始端口地址 + 31
SINGLE	数值范围：-3.4E+38～3.4E+38 符号部端口编号：开始端口地址 + 31 指数部端口编号：开始端口地址 + 30 ～ 开始端口地址 + 23 尾数部端口编号：开始端口地址 + 22 ～ 开始端口地址

・WINCAPSⅢ不支持显示 SINGLE 型的数值（查看等）。
请用多功能教导器来参照 SINGLE 型的数值。

5.3 流程控制语句

FOR～NEXT（语句）

功能　循环执行位于 FOR～NEXT 之间的一系列的指令。

格式

```
FOR <变量名> = <初始值> TO <最终值> [STEP <增量>]
  :
NEXT [<变量名>]
```

说明

根据 FOR 行指定的条件，反复执行位于 FOR～NEXT 区间内的一系列命令。

<初始值> 设定 <变量名> 中指定的变量的初始值。

<最终值> 设定 <变量名> 中指定的变量的最终值。

<增量> 设定初始值和最终值之间的增量。如果省略 STEP，则增量被视为 1。
此外，<增量> 里不能指定负的数值。

在 1 个 FOR～NEXT 中，可以再放置 1 个 FOR～NEXT（称为程序套结构、嵌套结构）。

在这种情况下，在各个 <变量名> 中必须使用不同的。

此时，1 个 FOR～NEXT 必须完全在其他的 FOR～NEXT 的内部。

应用示例

```
DEFINT li1
 FOR li1 = 1 TO 5          '反复进行 5 次 FOR～NEXT 的处理。

NEXT                       '反复进行。
```

IF～END IF（语句）

功能

执行 IF～END IF 之间的条件判断表达式中的条件判断。

格式

```
IF <条件式> THEN
    :
[ELSEIF <条件式> THEN]
    :
[ELSE]
    :
END IF
```

说明

根据 <条件式> 的条件控制程序的执行。

如果 IF 语句的 <条件式> 为真（0 以外的数），则执行 IF～ELSEIF 语句之间的语句，
如果 <条件式> 为假 (0)，则判断 ELSEIF 语句的 <条件式>。
同样，与 ELSEIF～ELSE，ELSE～END IF 执行。

相关项目

IF～THEN～ELSE

应用示例

```
DIM li1 As Integer
IF li1 = 0 THEN                 'li1 为 0 的情况。
PAGE_CHANGE   PANEL1            '移动到页面名 PANEL1。
ELSEIF li1 = 1 THEN             'li1 为 1 的情况。
PAGE_CHANGE   PANEL2            '移动到页面名 PANEL2。
ELSEIF li1 = 2 THEN             'li1 为 2 的情况。
PAGE_CHANGE   PANEL3            '移动到页面名 PANEL3。
ELSE                            'li1 为其他的情况。
PAGE_CHANGE   PANEL4            '移动到页面名 PANEL4。
END IF                          '定义 IF 语句。
```

SELECT CASE（语句）

功能

执行多个条件判断。

格式

SELECT CASE <式>
　CASE <项目>[,<项目>...]
　　:
　[CASE ELSE]
END SELECT

说明

当 <式> 的值与 CASE 语句的 <项目> 一致时，执行该 CASE 之后的一系列命令。
在 <式> 中，可以指定算式或字符串。
在 <项目> 中,可以指定变量、常量、公式及条件公式。
条件式可如下所示进行指定。

- <算式 1> TO <算式 2>

  调查 <式> 的结果是否是在 <算式 1> 以上、还是在 <算式 2> 以下。
  为字符串时不能使用。

- IS <比较运算符> <算式>

  比较 <式> 的结果和 <算式> 的值。

为字符串时，<比较运算符> 仅为 "="。
CASE ELSE 语句在与所有 CASE 语句都不一致时被执行。
同时，CASE ELSE 语句必须放在 END SELECT 语句前。

相关项目

IF～END IF

应用示例

```
SELECT CASE Index                    'Index 的值与 CASE 语句的值一致时，
该指令被执行。
        CASE 0                       'Index 为 0 的情况。
          Button1.caption = "0"
        CASE 1                       'Index 为 1 的情况。
          Button1.caption = "1"
        CASE 2                       'Index 为 2 的情况。
          Button1.caption = "2"
        CASE 3                       'Index 为 3 的情况。
          Button1.caption = "3"
        CASE 4                       'Index 为 4 的情况。
          Button1.caption = "4"
        CASE 5                       'Index 为 5 的情况。
          Button1.caption = "5"
        CASE 6 TO 8                  'Index 为 6～8 的情况。
          Button1.caption = "6-8"
        CASE IS ≧ 9                  'Index 为 9 以上的情况。
          Button1.caption = "9-"
END SELECT                           '定义多个条件判断语句的结束。
```

5.4 输出入控制语句

IN（语句）

功能 从通过 I / O 变量表示的 I / O 端口中读取数据。

格式 IN <算术变量名> = <I / O 变量>

说明 将用<I / O 变量>表示的 I / O 端口的数据赋值用 <算术变量名> 指定的变量。

<I / O 变量>是用 DEFIO 语句被定义的变量，或是 I / O 型变量。

相关项目 OUT、DEFIO

应用示例

```
DEFINT Li1, Li2
DEFIO samp1 = INTEGER, 220 '将 samp1 作为从端口 220 开始的 INTEGER 型 I / O
                           '变量进行定义。
IN Li1 = samp1             '将 samp1 的数据赋值到 Li1。
IN Li2 = IO[240]           '将端口 240 的数据赋值到 Li2。
OUT samp1 = Li1            '将 Li1 的数据从以 samp1 定义的端口输出。
OUT IO[240] = Li2          '将 Li2 的数据从端口 240 输出。
```

注意事项

在 SINGLE 下定义的 I/O 型变量内容，无法作为单精度实数表示时，将出现 777F 错误（实数变换失败）。

OUT（语句）

功能 向通过 I / O 变量表示的 I / O 端口输出数据。

格式 OUT <I / O 变量>=<输出数据>

说明 将 <输出数据> 的值输出在以 <I / O 变量> 表示的端口地址上。

<I / O 变量> 是用 DEFIO 语句被定义的变量，或是 I / O 型变量。

相关项目 IN、DEFIO

应用示例

```
DEFINT Li1, Li2
DEFIO samp1 = INTEGER, 220 '将 samp1 作为从端口 220 开始的 INTEGER 型 I / O
                           '变量进行定义。
IN Li1 = samp1             '将 samp1 的数据赋值到 Li1。
IN Li2 = IO[240]           '将端口 240 的数据赋值到 Li2。
OUT samp1 = Li1            '将 Li1 的数据从以 samp1 定义的端口输出。
OUT IO[240] = Li2          '将 Li2 的数据从端口 240 输出。
```

SET（语句）

功能　　将 I / O 端口置为 ON。

格式　　SET<I / O 变量>

说明　　将在<I / O 变量>中指定的端口置于 ON。

相关项目　　RESET、DEFIO

应用示例

```
SET IO[240]                 'BIT 型端口 240 置于 ON。
SET IO[SOL1]                'I / O 变量 SOL1 所表示的端口置于 ON。
SET IO[104 TO 110]          'BIT 端口 104～110 置于 ON。
IF IO[242] THEN
   RESET IO[240]            'BIT 型端口 240 置于 OFF。
   RESET IO[SOL1]           'I / O 变量 SOL1 所表示的端口置于 OFF。
   RESET IO[104 TO 110]     'BIT 型端口 104～110 置于 OFF。
ENDIF
```

注意事项

在 SINGLE 下定义的 I/O 型变量无法使用。如果执行，将出现 777F 错误（实数变换失败）。

RESET（语句）

功能　　将 I / O 端口置为 OFF。

格式　　RESET <I / O 变量>

说明　　将在<I / O 变量>中指定的端口置于 OFF。

相关项目　　SET、DEFIO

应用示例

```
SET IO[240]                 'BIT 型端口 240 置于 ON。
SET IO[241], 40             将 BIT 型端口 241 在 40ms 之间置于 ON。
SET IO[SOL1]                'I / O 变量 SOL1 所表示的端口置于 ON。
SET IO[104 TO 110]          'BIT 端口 104～110 置于 ON。
IF IO[242] THEN
   RESET IO[240]            'BIT 型端口 240 置于 OFF。
   RESET IO[SOL1]           'I / O 变量 SOL1 所表示的端口置于 OFF。
   RESET IO[104 TO 110]     'BIT 型端口 104～110 置于 OFF。
ENDIF
```

MSGBOX（语句）

功能　　将信息显示在多功能教导器的彩色液晶上。

格式　　MSGBOX<信息字符串>

说明　　将指定的信息显示在多功能教导器的彩色液晶触屏上。

可以显示的信息字符串最多为 60 字符。

应用示例

```
MSGBOX   "Hello World !"
```

注意事项

请注意即使在具有弹出视窗的零部件（数值输入框、文本框）的 CLICKED 动作中记述 MSGBOX 也不被执行。

PAGE_CHANGE（语句）

功能　　进行页面的显示。

格式　　PAGE_CHANGE　<页面名>　[,<文件夹返回数>]

<页面名>　　在操作盘中配置的页面的名称

<文件夹返回数>　从当前所在的文件夹向上层返回的文件夹数

说明　　将用页面名指定的编号的页面显示在操作盘画面上。

应用示例

```
page_change panel1          '指定画面的显示
page_change panel1,2        '将文件夹向上层移动 2 层，并显示该文件夹的 panel1
```

5.5 多项任务控制语句

RUN（语句）

功能 与其它程序同时启动。

格式 RUN <程序名>[(<自变量 (argument)>[,< 自变量 (argument)>…])][,<RUN 选项>]

说明 从当前正在执行的程序中并行启动在 <程序名> 上所指定的程序。
不能将自程序进行 RUN。

在 <自变量 (argument)> 中，只能使用传值。当指定了传址提交时，自动成为传值，但不能使用局域排列。

在 <RUN 选项> 中，有 PRIORITY（或者 P）、CYCLE（或者 C）。

PRIORITY（或者 P）

指定程序的优先度。如果省略，则作为默认值 128 进行处理。数值越小优先度越高。
设定范围在 102～255 之间。

注：特权任务的优先顺序不能变更。

CYCLE（或者 C）

指定切换周期（反复启动程序时的每个周期的间隔）。单位是 msec。
设定范围在 1～2147483647 之间。

带有自变量 (argument) 的程序，添加 CYCLE 选项不能启动。

应用示例

```
DEFINT Li1 = 1, Li2 = 2, Li3 = 3
RUN samp1, C = 1000            '将 samp1 进行并行 (C = 1000) 启动。
RUN samp2(Li1)                 '在 samp2 上添加 Li1 的自变量 (argument) 并行启动
                               。
RUN samp3(Li1, Li2),PRIORITY = 129
                               '在 samp3 上添加 Li1, Li2 的自变量 (argument)
                               '并行 (P = 129) 启动。
RUN samp4(Li1, Li2),  PRIORITY = 150
                               '在 samp4 上添加 Li1, Li2 的自变量 (argument) 并行
                               '(P = 150) 启动。
RUN samp5(Li1, Li2, Li3),  PRIORITY = 120
                               '在 samp5 上添加 Li1, Li2, Li3 的自变量 (argument)
                               '并行 (P = 120) 启动。
```

KILL（语句）

功能　强制结束任务。

格式　KILL <程序名>

说明　强制结束用 <程序名> 指定的任务（程序）的执行。

对自程序不能进行 KILL。如果对自程序进行 KILL，则会发生错误。
为了强制结束自程序，请使用 STOP 命令。

相关项目　SUSPEND

应用示例

```
RUN samp1                '并行启动 samp1。
  .
  .
KILL samp1               '结束 samp1。
```

SUSPEND（语句）

功能　暂停任务。

格式　SUSPEND <程序名>

说明　暂停所指定的任务。

对自程序不能进行 SUSPEND。

相关项目　KILL

应用示例

```
SUSPEND samp1            '将 samp1 的任务的执行暂停。
```

SUSPENDALL（语句）

功能　　停止特权任务以外的所有的程序。

格式　　SUSPENDALL

说明　　停止特权之外的所有任务，使停止中的任务处于可以从停止步骤转换为让其动作的状态，即 "连续停止" 状态，并将机械手运行过程中的输出置于 OFF。

相关项目　　SUSPEND、KILLALL

应用示例

```
SUSPENDALL                    '所有任务瞬时停止　进入连续停止状态
```

KILLALL（语句）

功能　　强制结束特权任务以外的所有的任务。（相当于程序清零）

格式　　KILLALL

说明　　强制结束机械手当前正在执行的特权任务以外的所有任务，将机械手运行过程中的输出置为 OFF。

相关项目　　KILL、SUSPENDALL

应用示例

```
KILLALL                       '所有任务停止　程序清零状态
```

CONTINUERUN（语句）

功能　　再次启动连续停止过程中的所有的任务。

格式　　CONTINUERUN

说明　　机械手让当前连续停止过程中的所有任务再次启动。

相关项目　　KILL、SUSPENDALL

应用示例

```
CONTINUERUN                '再次启动所有任务
```

DEADMANSTATE（语句）

功能　　获取双重安全开关的状态。
0: OFF, 1: ON

格式　　DEADMANSTATE

说明　　获取当前的双重安全开关的状态。

应用示例

```
IO = DEADMANSTATE                  '当前的双重安全开关
                                   ' (Deadman Switch)  的状态进入到 I / O。
```

5.6 常量

OFF（内置常量）

功能 赋予 OFF (0) 值。

格式 OFF

说明 对于式赋予 OFF (0) 的值。

相关项目 ON

应用示例

```
IF I0 = TRUE THEN              '赋予布尔值的真 (1) 的值。
  I1 = ON                      '将 ON (1) 的值赋值整数型变量。
ELSEIF I0 = FALSE THEN         '赋予布尔值的假 (0) 的值。
  I1 = OFF                     '将 OFF (0) 的值赋值整数型变量。
ELSE
  D1 = PI                      '将π赋值实数型变量。
ENDIF
```

ON（内置常量）

功能 赋予 ON (1) 值。

格式 ON

说明 对于式赋予 ON (1) 的值。

相关项目 OFF

应用示例

```
IF I0 = TRUE THEN              '赋予布尔值的真 (1) 的值。
  I1 = ON                      '将 ON (1) 的值赋值整数型变量。
ELSEIF I0 = FALSE THEN         '赋予布尔值的假 (0) 的值。
  I1 = OFF                     '将 OFF (0) 的值赋值整数型变量。
ELSE
  D1 = PI                      '将π赋值实数型变量。
ENDIF
```

PI（内置常量）

功能

赋予π值。

格式

PI

说明

用倍精度型返回π的值。

应用示例

```
IF I0 = TRUE THEN              '赋予布尔值的真 (1) 的值。
  I1 = ON                      '将 ON (1) 的值赋值整数型变量。
ELSEIF I0 = FALSE THEN         '赋予布尔值的假 (0) 的值。
  I1 = OFF                     '将 OFF (0) 的值赋值整数型变量。
ELSE
  D1 = PI                      '将π赋值实数型变量。
ENDIF
```

FALSE（内置常量）

功能

赋予布尔值的假 (0) 的值。

格式

FALSE

说明

对于式赋予布尔值的假 (0) 的值。

相关项目

TRUE

应用示例

```
IF I0 = TRUE THEN              '赋予布尔值的真 (1) 的值。
  I1 = ON                      '将 ON (1) 的值赋值整数型变量。
ELSEIF I0 = FALSE THEN         '赋予布尔值的假 (0) 的值。
  I1 = OFF                     '将 OFF (0) 的值赋值整数型变量。
ELSE
  D1 = PI                      '将π赋值实数型变量。
ENDIF
```

TRUE（内置常量）

功能

赋予布尔值的真 (1) 的值。

格式

TRUE

说明

对于式赋予布尔值的真 (1) 的值。

相关项目

FALSE

应用示例

```
IF I0 = TRUE THEN              '赋予布尔值的真 (1) 的值。
  I1 = ON                      '将 ON (1)的值赋值整数型变量。
ELSEIF I0 = FALSE THEN         '赋予布尔值的假 (0) 的值。
  I1 = OFF                     '将 OFF (0) 的值赋值整数型变量。
ELSE
  D1 = PI                      '将π赋值实数型变量。
ENDIF
```

5.7 时间 / 日期控制

DATE$（系统变量）

功能 得出当前的日期。

格式 DATE$

说明 当前的日期以 "yyy / mm / dd"（年 / 月 / 日）的形式被存放。

相关项目 TIME$

应用示例

```
defstr   ls1
ls1 = DATE$                    '将当前的日期赋值 ls1。
```

TIME$（系统变量）

功能 得出当前的时间。

格式 TIME$

说明 当前的时间以 "hh:mm:ss"（时:分:秒）的形式存放。
时刻的显示为 24 小时方式。

相关项目 DATE$

应用示例

```
defstr ls1
ls1 = TIME$                    '将当前的时间赋值 ls1。
```

TIMER（系统变量）

功能 得出经过时间。

格式 TIMER

说明 以接通控制器的电源时刻为基准 (0)，获取以毫秒计时所经过的时间。

> 注意：当经过时间超过2147483647毫秒时，以-2147483648毫秒为基准返回经过时间。

应用示例

```
DEFINT li1, li2, li3
li1 = TIMER                    '将从基准时间开始所经过的时间赋值 li1。
```

5.8 字符串函数

STR$（函数）

功能 将数值转换为字符串。

格式 STR$(<算式>)

说明 将指定在 <算式> 上的值转换为字符串。

相关项目 CHR$, SPRINTF$

应用示例

```
DEFSTR ls1, ls2
ls1 = STR$(20)                  '将 20 转换为字符串赋值 ls1。
ls2 = STR$(li1)                 '将 li1 转换为字符串赋值 ls2。
```

CHR$（函数）

功能 将 ASCII 编码转换为文字。

格式 CHR$(<算式>)

说明 得到具有在 <算式> 上所指定的值的字母编码的字符。

相关项目 STR$

应用示例

```
DEFSTR ls1, ls2
ls1 = CHR$(49)          '将具有 49 的字母编码的字符赋值 ls1。
ls2 = CHR$(&H4E)        '将具有&H4E 的字母编码的字符赋值 ls2。

PBI.caption="程序"+ CHR$（13）+ CHR$（10）+"启动"
                        '想要使显示文字改行时
```

SPRINTF$（函数）

功能 转换为指定了表达式的格式，并作为字符串返回。

格式 SPRINTF$（<格式>,<表达式>）

相关项目 STR$

应用示例

```
SO = SPRINTF$("% d",123)            '字符串 123 进入到 SO。
```

5.9 系统信息

CUROPTMODE（语句）

功能 获取动作模式。
1：手动、2：教导检查、3：内部自动、4：外部自动

格式 CUROPTMODE

说明 获取当前的动作模式（手动、教导检查、内部自动、外部自动）的信息。

应用示例

```
I [1]  =  CUROPTMODE  '动作模式获取
```

SYSSTATE（语句）

功能 获取控制器的状态。

格式 SYSSTATE

说明 获取控制器状态。根据 I / O 配置的设定，有效的数据发生变化。
可以获取的数据如下所示。

位	0	机械手运行中
	1	机械手异常
	2	伺服 ON 状态
	3	机械手初始化完成（选择 I / O 标准模式时） / 机械手电源已投入（选择 I / O 互换模式时）
	4	自动模式
	5	外部模式
	6	电池耗尽警告
	7	机械手警告
	8	连续开始许可
	9	SS 模式
	10	机械手停止
	11	自动运行允许
	12～15	预约
	16	程序开始清零（选择 I / O 互换模式时）
	17	CAL 完成（选择 I / O 互换模式时）
	18	正在教导（选择 I / O 互换模式时）
	19	1 循环完成（选择 I / O 互换模式时）
	20～23	预约
	24	指令处理完成（选择 I / O 标准模式时）
	25～31	预约

应用示例

```
I [1] = SYSSTATE          '获取系统状态
```

STATUS（函数）

功能

得出程序的状态。

格式

STATUS(<程序名>)

说明

<程序名> 所指定的程序的状态以整数被存放。

值	状　　态	
1	running	正在执行
2	stopping	停止状态
3	suspend	暂停状态
4	delay	延迟状态
5	pending	等待状态
6	step stopped	步骤停止状态

应用示例

```
defint   li1
li1 = STATUS(samp1)          '将 samp1 的程序状态以整数赋值 li1。
```

注意事项

不能获取自身的 STATUS。

5.10 预处理器

#define（预处理器语句）

功能　使用指定字符串替换程序中所指定的常量或宏名。

格式　#define <符号常量><字符串>

或

#define <宏名(自变量 (argument))><包括自变量 (argument) 的字符串>

说明　使用指定字符串替换程序中的 <符号常量> 或 <宏名>。为宏名时，包括自变量 (argument) 也可置换。

程序中的 <符号常量> 或 <宏名> 不能置换以双引号 " " 括起的字符串。

#define 语句必须写在 1 行内。

在 <符号常量> 与 <字符串> 之间必须插入 1 个以上的空格。

与括起宏名和自变量 (argument) ( ) 之间，不能插入空格。

<符号常量> 和 <宏名> 必须在 64 个字符以内。

1 个程序最多可 2048 个宏名。此外，宏函数的自变量 (argument) 数量没有限制。

应用示例

```
#DEFINE NAME "Denso Corporation"
                                    '为 NAME 的符号常量赋予 "Denso Corporation"。
S1 = NAME                           '将 "Denso Corporation" 赋值 S1。
```

#include（预处理器语句）

功能

读入预处理程序。

格式

#include　"[路径]文件名"

#include <[路径]文件名>

说明

在保存#include 语句的位置，装入预处理器程序文件。如果省略文件的路径，则 " " 时，以现用户位置目录、系统目录的顺序搜索文件。在< >时，只从系统目录搜索文件。若以全路径 指定路径，则只搜索该目录。

在用#include 语句指定的文件中，可以进一步包含#include 语句，能够最多进行 8 次嵌套。

可以指定的文件的扩展符是 H。

应用示例

```
#include "samp1.h"              '将 samp1.h 文件在这一行展开。
```

RC7 型控制器用

操作盘功能说明书

使用说明书 增补版

初　版　2008 年　1 月
第 2 版　2009 年　4 月
第 3 版　2011 年　9 月

DENSO WAVE INCORPORATED

9N**C