JTEKT

Value & Technology

G C系 列 触 摸 屏

（画面编辑软件 SCREEN CREATOR 5）

嵌入式脚本语言

K-BASIC

参 考 手 册

捷太格特电子(无锡)有限公司
JTEKT ELECTRONICS (WUXI) CO.,LTD.

前　言

感谢您选用捷太格特电子 GC 系列工业触摸式显示器（触摸屏）。我们致力于使我们的资料正确完整，但也因为我们的产品在不断更新和改进，所以我们不可能保证资料完全最新。并且，我们对您使用本产品作如下声明：

1） 我们有权在未经用户允许的情况下根据需要对本手册的任何部分进行修改。

2） 我们热忱欢迎用户对本手册中错误和不当之处提出修改意见，对您表示感谢！

3） 捷太格特电子对正确和不正确使用本手册及 Screen Creator 5 软件所产生的一切直接和间接后果不承担任何法律和经济责任！

4） 在使用本手册和 GC 产品时有任何疑问可与本公司驻当地办事处联系，或直接与我们联系。我们的联系方式是：

地址：江苏省无锡市滨湖区建筑西路 599 号 1 栋 21 层
捷太格特电子（无锡）有限公司营业技术部
联系电话：0510－85167888－8022/8056
传　　真：0510－85161393
homepage：http://www.JTEKTele.com.cn
email：support@JTEKTele.com.cn

GC 专用名词一览

OIP = Operator's Interface Panel	触摸屏
project = system	工　程
screen	画　面
part = component	部　品
control = primitive	控　件
Texture = a collection of figures	构　件
text	文　本
device	设　备
property = setting = attribute	属　性
figure	图　形
pattern	图　案

目 录

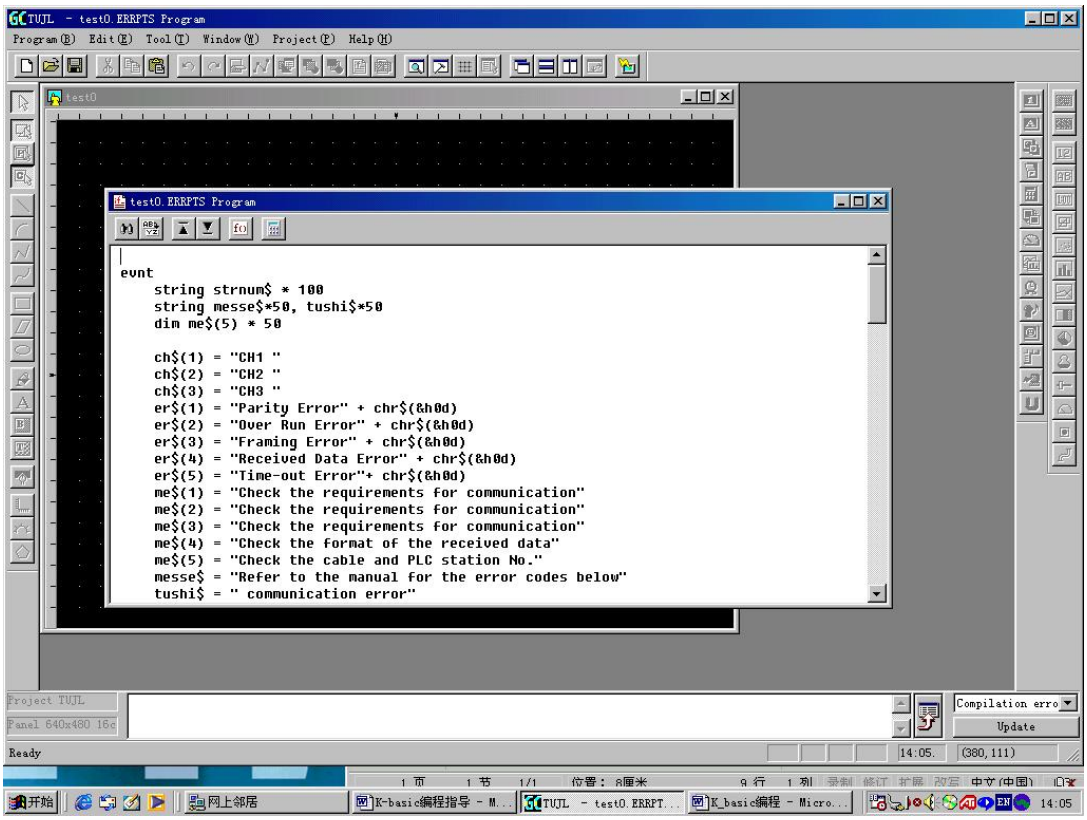
K-basic语言编程基础

第一章 简 介

1、什么是控制程序

当你用手指按操作面板上的按钮的时候，你希望某部品显示数值、字符、或使某个开关运作时，你需要使用K-basic语言编制程序。

我们使用K-basic程序语言来为部品编制程序，使其完成期望的功能。



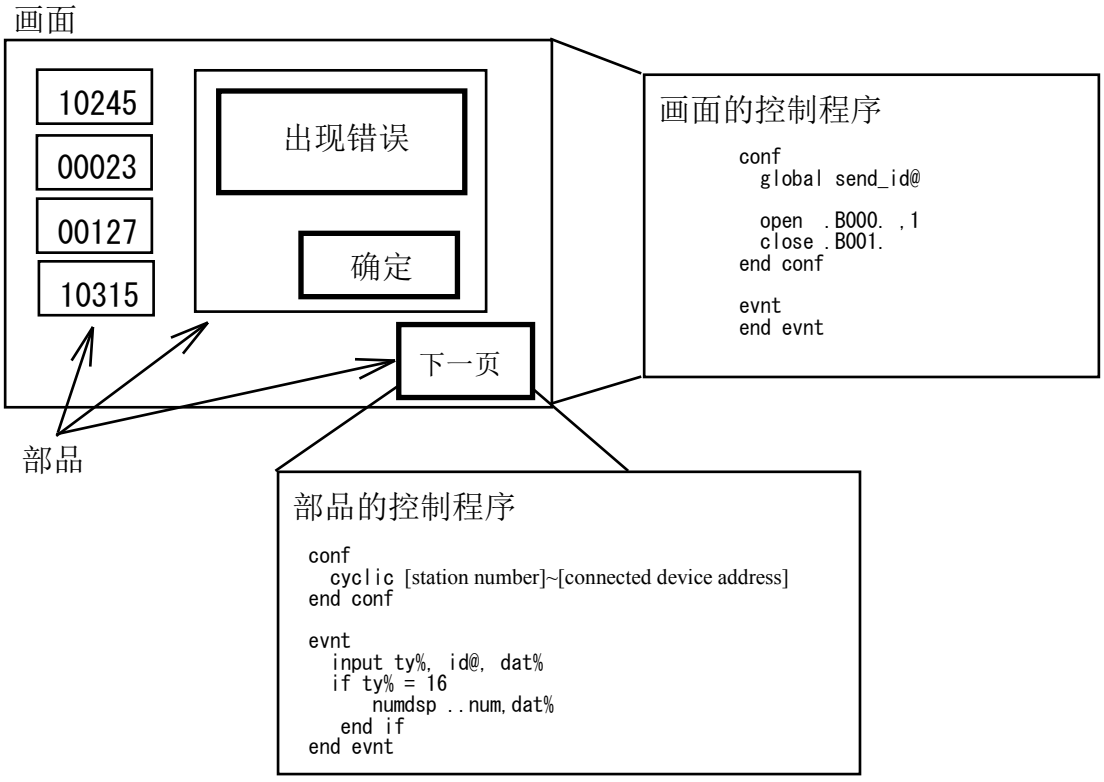
2、K-basic 语言描述的对象

（1）部品的控制程序

我们可以为每个部品单独编制控制程序。

（2）画面的控制程序

同部品一样，你可以为每幅画面编制控制程序。

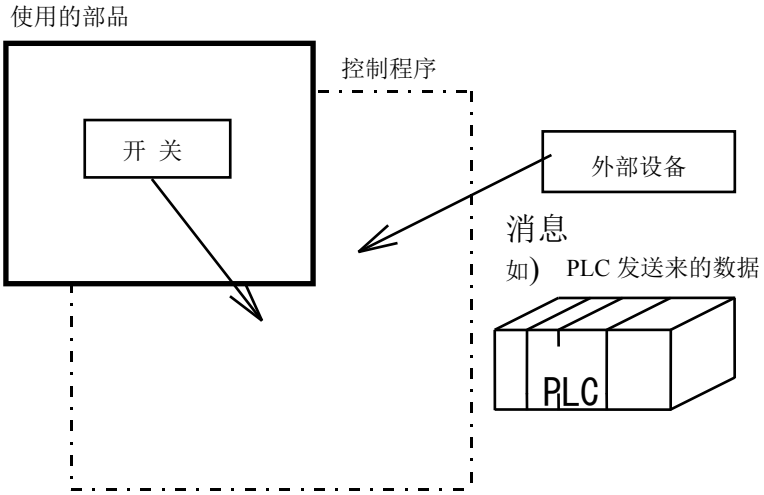


（3）常用术语

a．消息

消息是程序执行的触发信号。部品在接受到发给他的信号之后才开始运作。开关、定时器、PLC等外部设备都能发出消息。

每条消息包括发送设备的 ID（PLC 设备名、部品名称等）、数据类型、数值等。

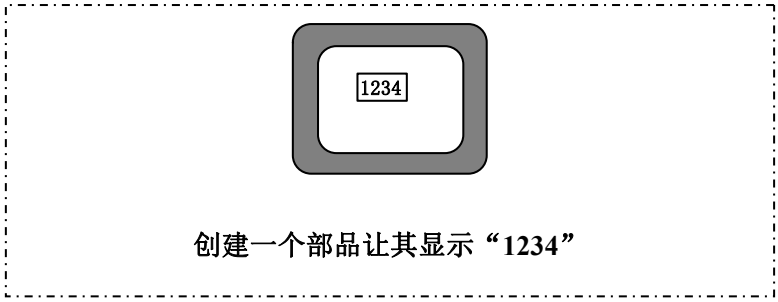


b．其它术语。

画面（Screen）
绘图（Figure）
部品（Parts）
空件（Control）

第二章 编程实例

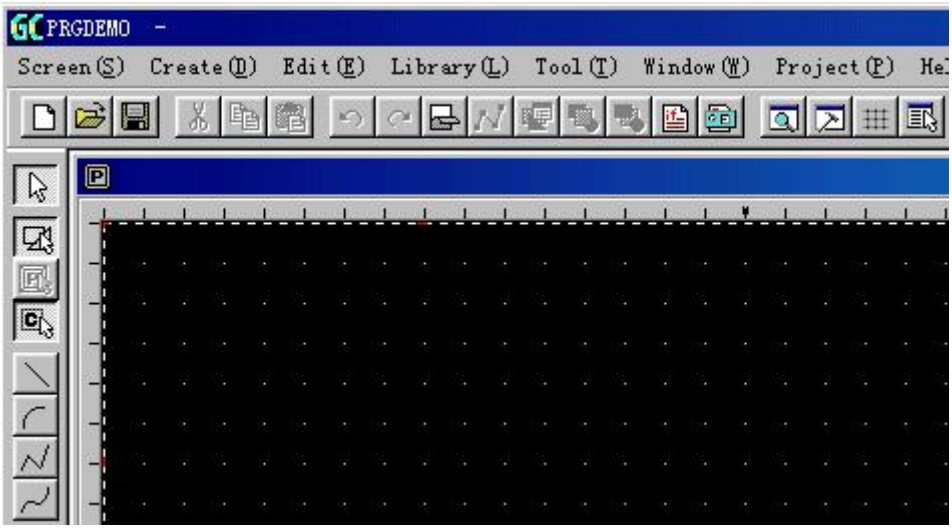
1、创建一个显示数据的部品



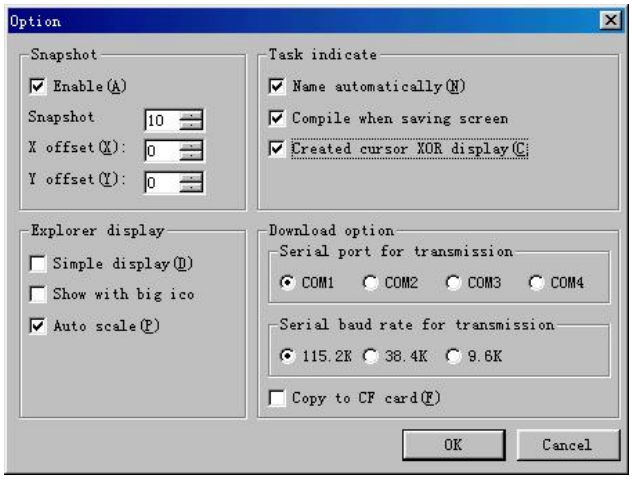
创建一个部品让其显示“1234”

首先创建（或打开）一个工程，如前面章节所述。

然后，点击菜单“Library(L)-New(N)-Parts(P)”，会自动打开一个创建部品的窗口。左上角显示“P”，表示当前窗口是在创建部品（Part）。



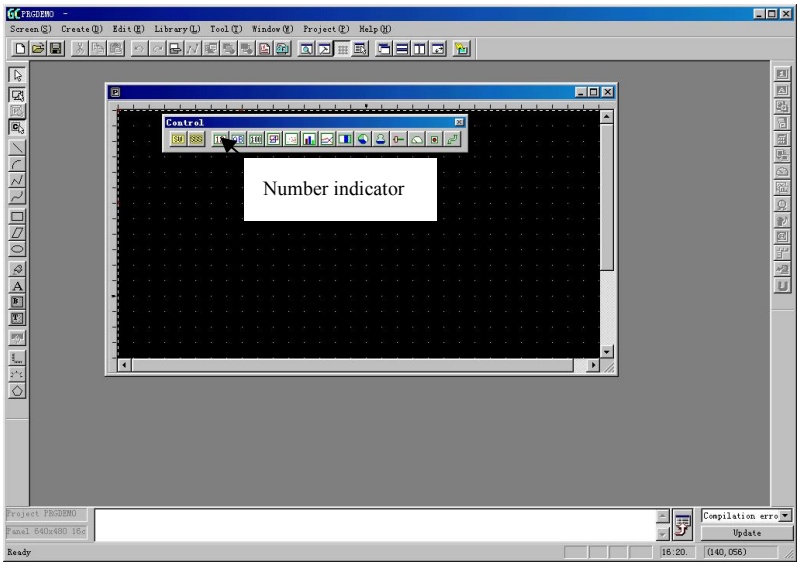
然后点击菜单“Tool－option(O)”，打开选项对话框。



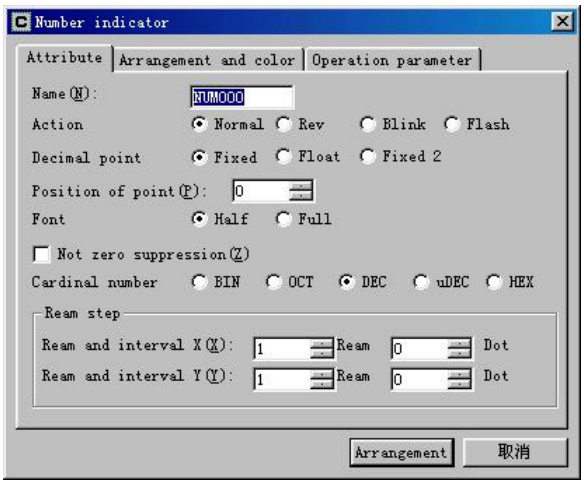
作如上选择，然后点击“OK”。

（1） 控件的放置

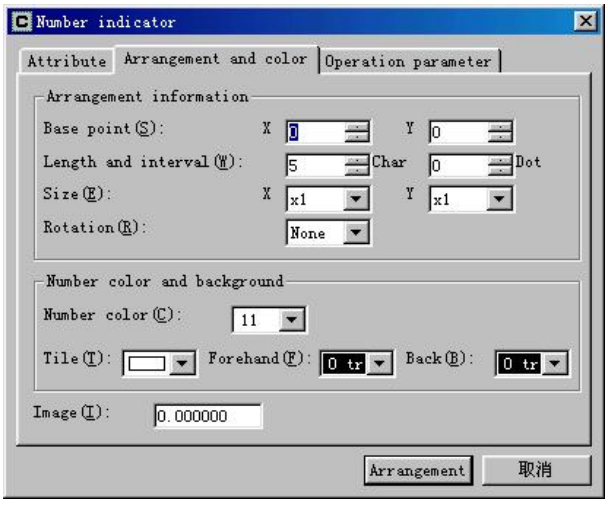
首先选择数字显示控件



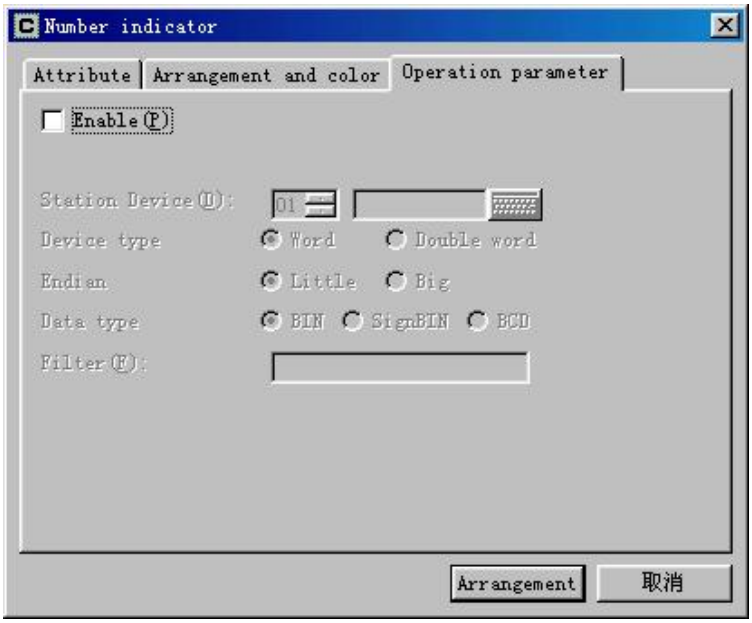
数据显示器属性设置对话框显示如下：



点击“Arrange and color”，显示本控件物理属性（控件放置信息）：

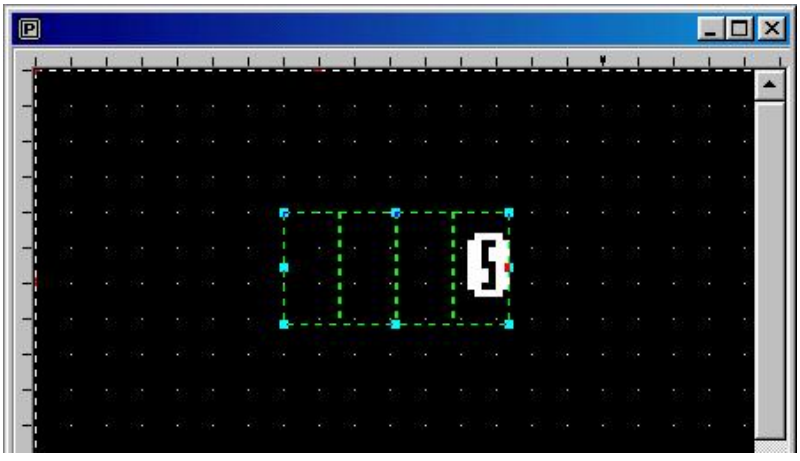


点击“Operation parameter”，不要选定“Enable”，因为其运行要通过程序实现.



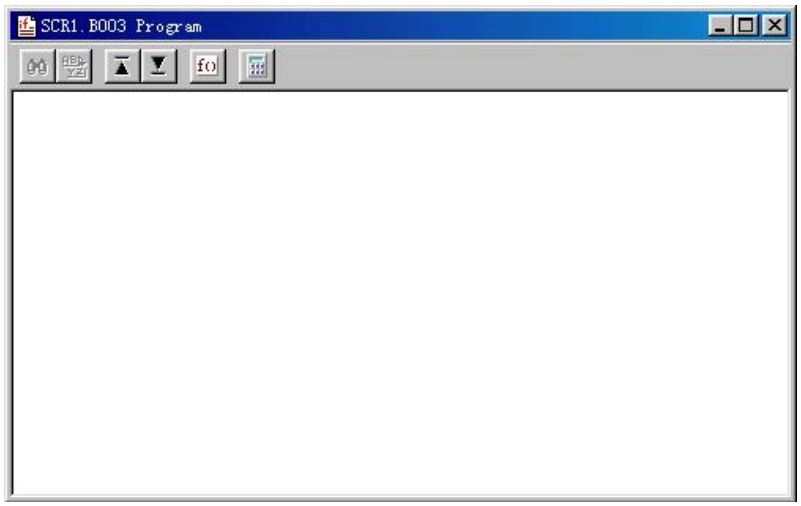
点击“Arrange”，将控件放在窗口的某地方

用鼠标拉住控件的右下角，将其拉到适当大小。



（2）编程

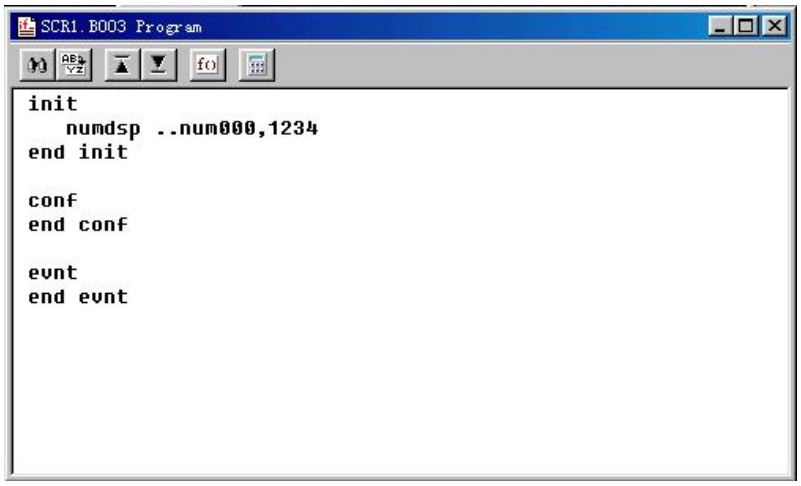
要为部品编制程序，可以选择菜单中的“Edit”，然后选择“Edit Part Programs”，或者直接点击编程的快捷方式  ，会弹出编程窗口如下：



根据要求我们编制程序如下：

```
init
   numdsp ..NUM000,1234 '在数据显示控件..NUM000里显示1234
end init

conf
end conf

evnt

end evnt
```

程序的内容将在后续部分介绍！首先，输入以上程序，程序编辑窗口变得如下：



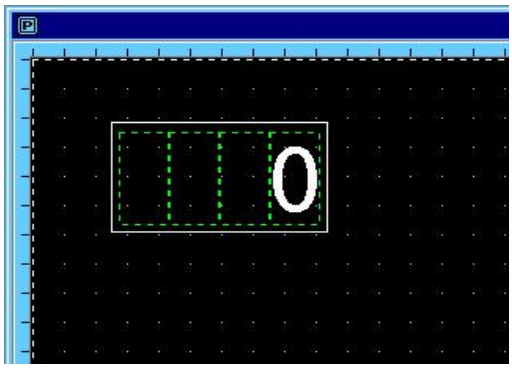
选择“Program”菜单中的“Save”或直接点击  ，保存上述程序。关闭编程窗口，这时部品编辑窗口重新出现在我们面前。

（3）在部品中绘图

为了美观或需要，我们给数据显示器控价画一个长方形的外框。

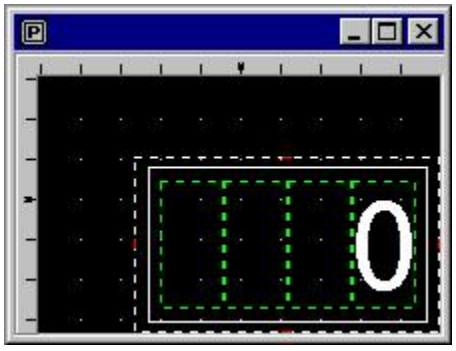
选择的“Create” 菜单,下拉选择“Rectancle”，然后用鼠标左键点击数据显示控件的左上角，同时按住左键不放松，沿着显示控件的对角线拖动鼠标，同时会出现一个矩形框，该框的大小随着鼠标拖动的位置改变。当大小合适时放开鼠标左键。当然，在画框之前你可以选择其属性，诸如颜色、线型等。

在画完后，鼠标还处于画框模式状态，这时只要点击鼠标右键即可取消。

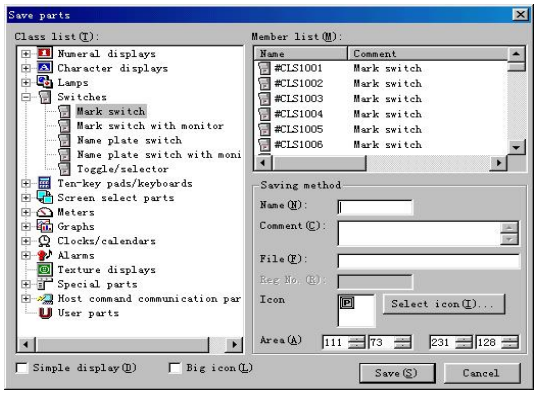


（4）部品的保存

部品的物理大小是由红白相间虚框的大小决定的，所以为了部品占地适当，应将紧贴窗口边框的虚框缩小，缩小的方法是将鼠标放在外框上有红色标记的地方，然后按照鼠标光标箭头方向拖动。注意，虚框不能过小，即必须罩住所有的内容，否则不能正确保存！



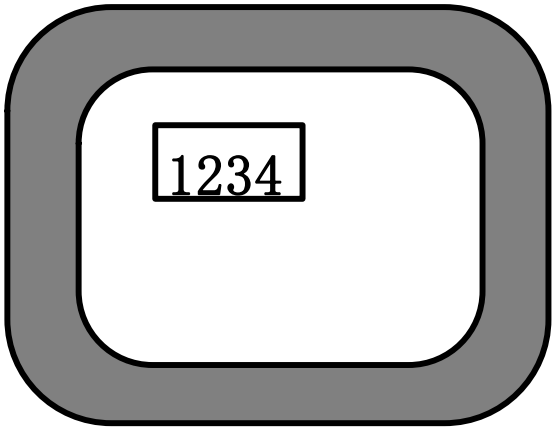
然后，保存部品，这时你既可以通过选择“Library”菜单中的“Save”来保存，也可以通过直接点击快捷方式“  ”来保存。这时出现如下对话窗口：



我们将部品保存为用户部品“User parts”，部品名称保存为“test”，在“Comment（注释）”中输入“test part”，点击“save”保存部品。
保存后，点击部品窗口右上角关闭按钮，将部品窗口关闭。
这样，一个部品就做好了！

（5）创建部品的调用

同其它部品的使用一样，在画面上直接调用就行。
将其放在画面上下传之后，画面显示如下：



（6）程序解释

程序：

```
init
    numdsp ..NUM000,1234
init init

conf
end conf

evnt
end evnt
```

程序解释如下：

① init～end init

这部分称为初始块（initialization block），这部分程序在程序运行的开始执行，用于对变量的声明和初始化。

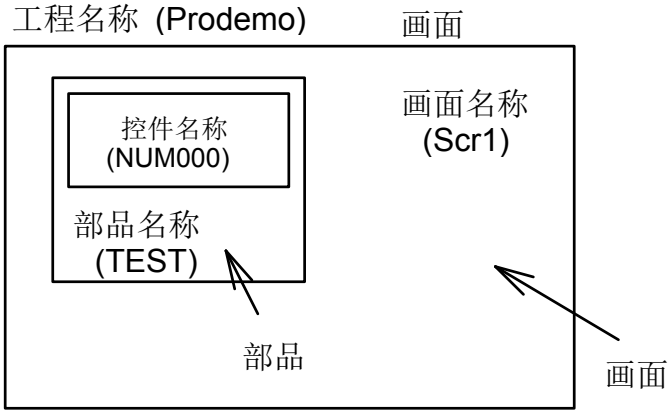
② numdsp ..NUM000,1234

上述语句的功能是将指定的数据显示到指定的数据显示控件。numdsp 命令，是在某个数据显示控件里显示某个数值。..NUM000 用于指出数据显示控件的名称，命名规则如下：

● 控件名及命名规则

Screen（画面）	Scr1..
Scr1 上的部品:	Scr1.Test
画面 Scr1 部品 Test 里的控件：	Scr1.Test.NUM000
当前画面上的当前部品：	..（其它省略）
当前画面上的当前部品里的当前控件：	..NUM000

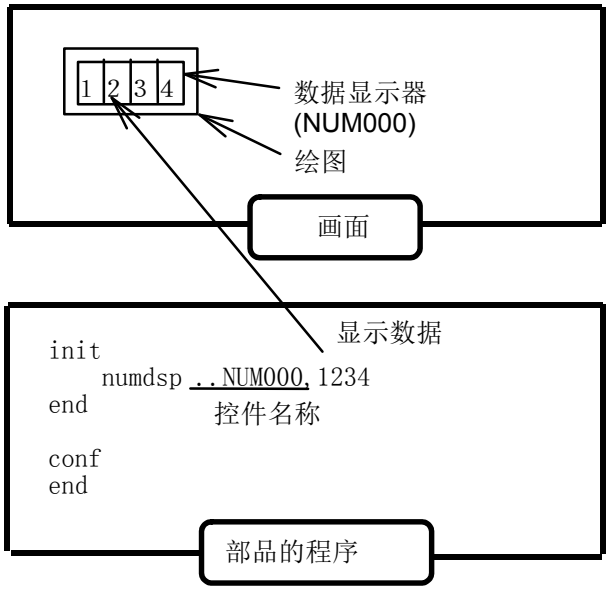
务必注意：部品名称只能是字母和数字，并且只能以字母开头！
同部品一样，如果当前部品上的控件，控件名前面也可以省略部品名称和控件名称。



程序中用于指定控件 NUM000 的是.

Scr1.TEST.NUM000
or ..NUM000

还有一个参数就是需要显示的数据，这里为“1234”，通过改变它，可以变更显示的数据。



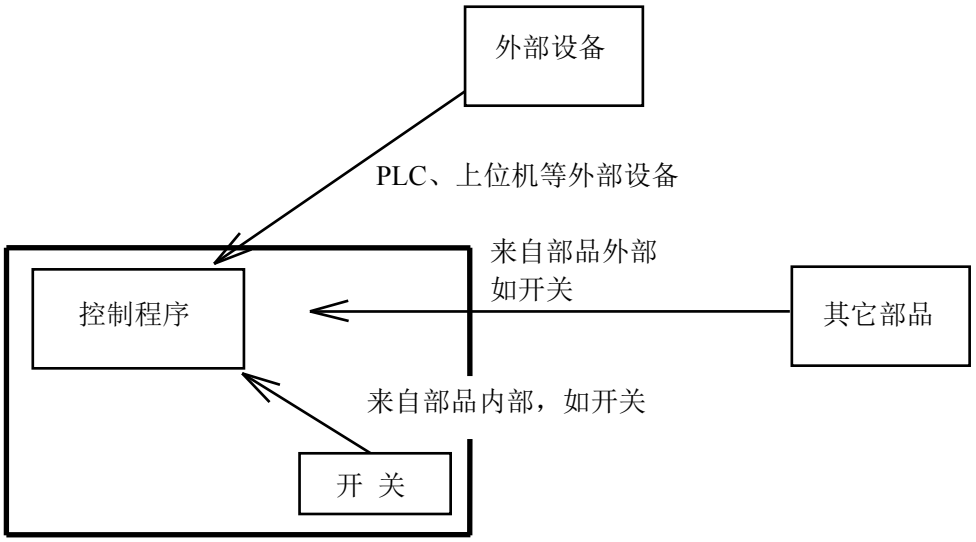
部品中的一些相互关系

NUM000 控件是本部品里的唯一控件。在触摸屏上显示“1234”

③ conf～end conf
本部分称为 Configuration Block(conf 块)。只有当部品已经在画面上显示出来（也就是当前画面的本部品已经打开），本部分程序才执行。本程序中，本部分为空。

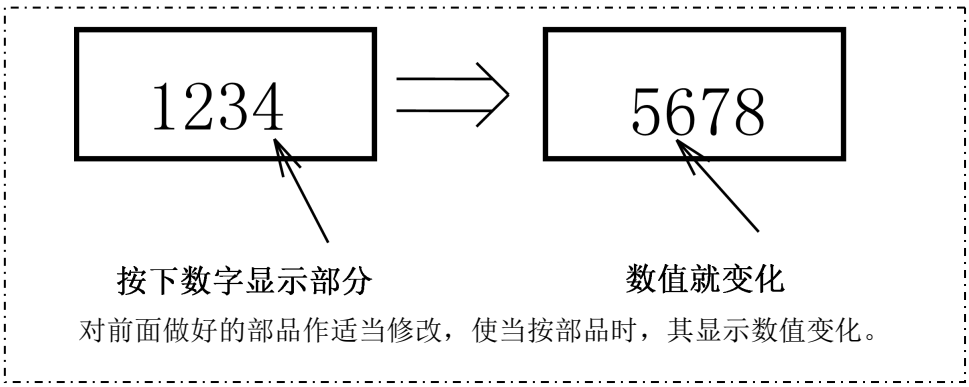
④ Evnt～end evnt
本部分称为 evnt block(事件块)，这部分程序只有在收到相关消息之后才执行。

部品的消息来自设备或部品等等，如下图所示：

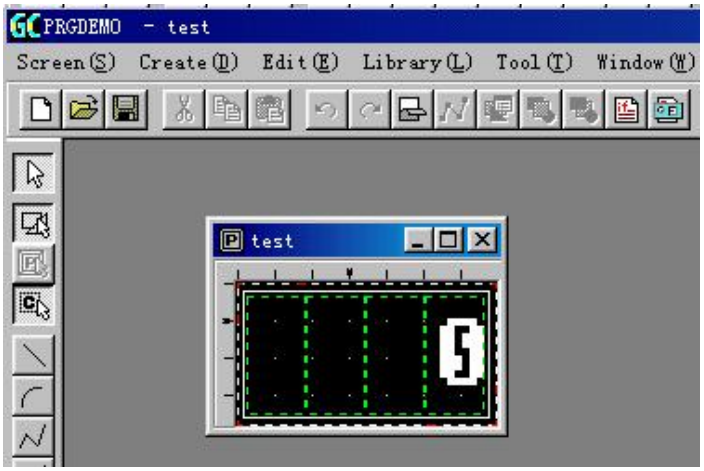


消息的发送

（7）部品的变更

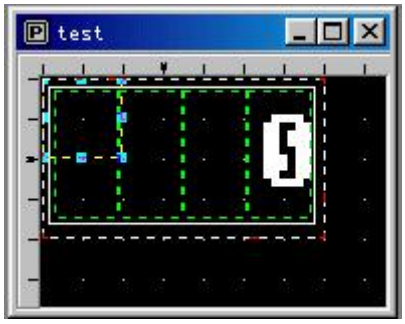


首先，要打开前面做好的部品“Test”。方法是：选择“Library”，选择 Open – Parts，然后打开部品对话框就打开了，选择“User Parts”并打开部品“Test”。



本例中，为了使用触摸面板，要在部品里添加一个开关。通过菜单或通过快捷方式选择“Control”-“Switch”(“控件”－“开关”)。弹出开关控件属性对话框，点击“Arrange”，光标编程小鼠形状。点击部品编辑窗口，光标变成方形开关控件形状，将其放于部品的左上角。然后用鼠标点击本控件，并拖住控件右下角，将其拉至适当大小。注意，因为是开关控件，所以拉大或缩小时只能与 20dots×20dots 为单位。

注意：因为是默认设置，所以该开关是点动开关（即按下时为“ON”，放手时为“OFF”)。



当将开关控件拉到适当大小之后，接下来是添加部品程序。用同上方法打开程序编辑窗口，输入下述程序：

```
init
    local type%, id@, data%
    numdsp ..NUM000,1234
end init

conf
end conf

evnt
    input type%,id@,data%
    if type%=3 and id@=..SWT000 and data%=1 then
        numdsp ..NUM000,5678
    end if
end evnt
```

通过比较可以看出，

Init block 里对在 Evnt Block 里使用的变量作了声明。

我们主要修改了 Evnt block 里添加了一段程序。其中 input 是标准函数，用来接受来自开关的消息；消息包括消息发送者类型（type)，消息发送者（id)，及接受到的数据（data)。如果条件满足“if～end if”之间提出的条件，则显示数据变化。

首先，介绍“input”指令：

使用 Input 指令从消息里获得各种信息。“input”指令的标准使用方法如下：

input type%,id@,data%

本例中，使用本指令可以获得信息解释如下：

type%：表示消息发送者类型，3 表示消息来自开关。如果消息来自 PLC 则为 16。更详细的情况，请参考后面“消息类型”。

id@：消息发送者身份（ID），如果消息来自开关，则它就是开关控件的名称。ID 的格式是：画面名称（Screen name）、部品名称（Part name）、控件名称（Control name），中间分别用半角句号（.）隔开。

如：Scr1.test.NUM000

这个 ID 称为 ID 型常量，它是 K-basic 独有的，在处理 ID 时，你可以在其后面加“@”符号，如“@ID”。

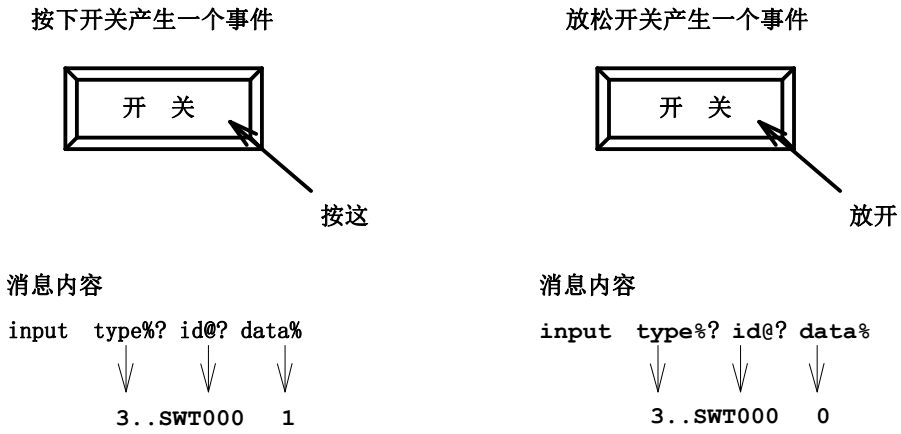
注意：消息包括类型（Type），身份代号（ID），和数据（Data）三部分。

Data%：消息发送者送过来的数据。如开关 ON 时，data%=1，开关 OFF 时，data%=0。

接下来，就是条件判断：

```
if type%=3 and id@=..SWT000 and data%=1 then
        : : : : : : : : : : : : : : : : :
end if
```

表示如果消息来自开关（type%=3），并且消息由SWT000发出（id@=..SWT000），且开关为ON（data%=1）。条件之间用“and”连接，表示条件要同时满足。如果只有type%=3和id@=SWT000，则条件将满足两次，即手指按下时和手指放开时。本例中，只有在手指按下时程序执行。



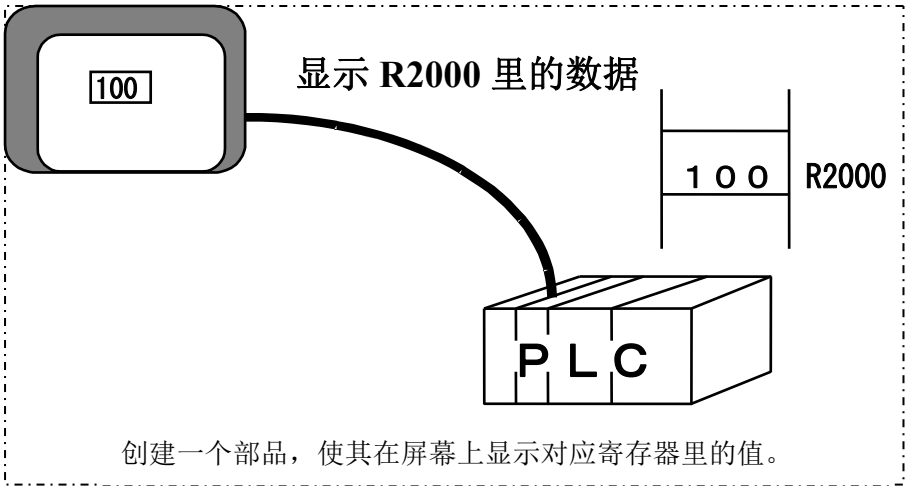
注意：如果开关设置为“momentary”，则开关按下一次将产生两条消息，即按下和放开。

采用同样的方法保存程序，然后调用、下载。就可以看到预期效果！

2．创建一个与 PLC 设备连接的部品

本资料的程序以 JTEKT 公司 PLC 为例，如果你使用的是其它公司的产品，应该对局号、设备名称、还有 PLC 类型进行修改，不过，原理上是完全一直的。

（1）数据显示

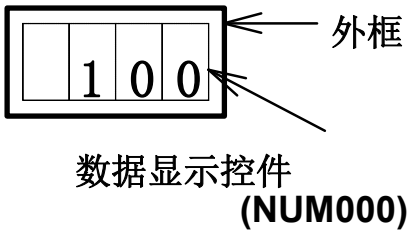


创建一个部品，使其在屏幕上显示对应寄存器里的值。

使用控件：

一个数据显示控件（NUM000）

外观：



程序例如下：

```
init
    local type%, id@, data%
    cyclic 01~R2000
end init

conf
end conf

evnt
    input type%,id@,data%
    if type%=16 and id@=01~r2000 then
       numdsp ..NUM000,data%
    end if
end evnt
```

● Init Block

cyclic 01~R2000 使用"cyclic"命令从“01”号局PLC里的R2000寄存器里读出数据。

“cyclic”指令用来始终监控 PLC 设备的值。

“cyclic”指令循环读取 PLC 设备的值。当 PLC 设备的值发生变化，“cyclic”指令将消息传送给 Evnt Block。Cyclic 后面依次接局号、设备名。中间使用“~”连接。

该指令即使在画面已经发生变化时，也进行数据传送。

● Conf Block

这里不作任何处理。

● Evnt Block

```
input type%,id@,data%
```

“input”指令用来读取传送给部品的消息。消息的按如下顺序格式：“type%”（16）、id@（01~R2000）、data%(PLC 里的值：01~R2000)。

```
if type%=16 and id@=01~r2000 then
   : : : : : : : : : : : : : : : : :
end if
```

条件 type%=16，表示消息来自 PLC，id@=01~r2000 表示发送消息者是 01~r2000。中间用 and 连接表示两个条件同时满足时，后续程序才能执行。

```
numdsp ..NUM000,data%
```

“numdsp”指令的功能是在显示控件“num000”里显示变量“data%”的值。其值就是使用 input 指令从消息里读出的“data%”值。

程序的思想是：在初始块（init block）里使用“cyclic”指令始终监控着 01 号 PLC 里寄存器 R2000 里的值。当里面的值发生变化时，使用 input 指令来接收消息，并将最新的值送到数据显示控件 NUM000 里。

试着使 PLC 里 R2000 的值发生变化，你将看到屏幕上的值也将同时发生变化。

● 如何使部品更容易使用

如果你希望监控 PLC 多个寄存器里的值，你必须重新制作多个部品并分别编程，因为每个寄存器的名称都不一样。你是否觉得这很让费时间呢？其实，若使用“参数”功能变可以使这个问题变得非常简单。

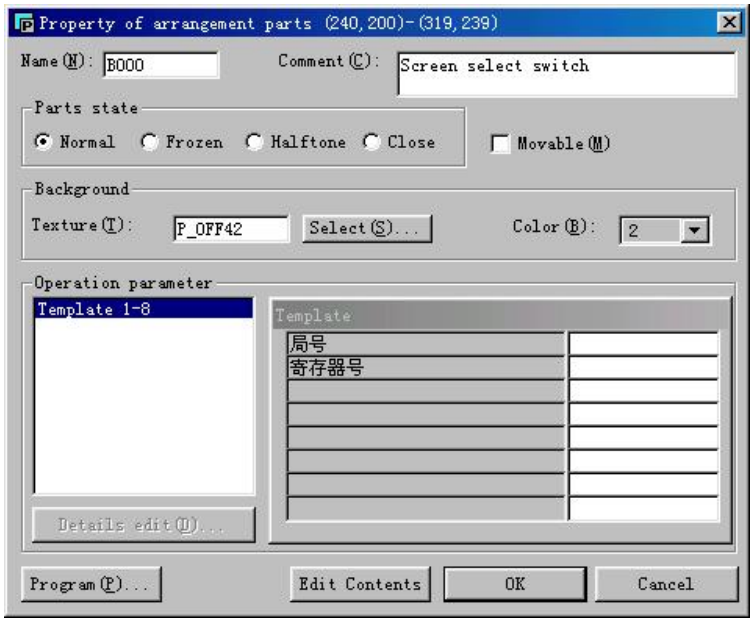
Cyclic [局号]~[寄存器号]

用中括号括起来的字符串称为 template（模板），一个模板里最多可写 32 个半角字符。因为写在括号里的字符串能在相应部品的模板（template）里显示，因此，这些部品也可以像标准部品一样使用。这样，对于不同的 PLC 内部寄存器，你只要在外部改变寄存器号即可，同标准部品一样方便。

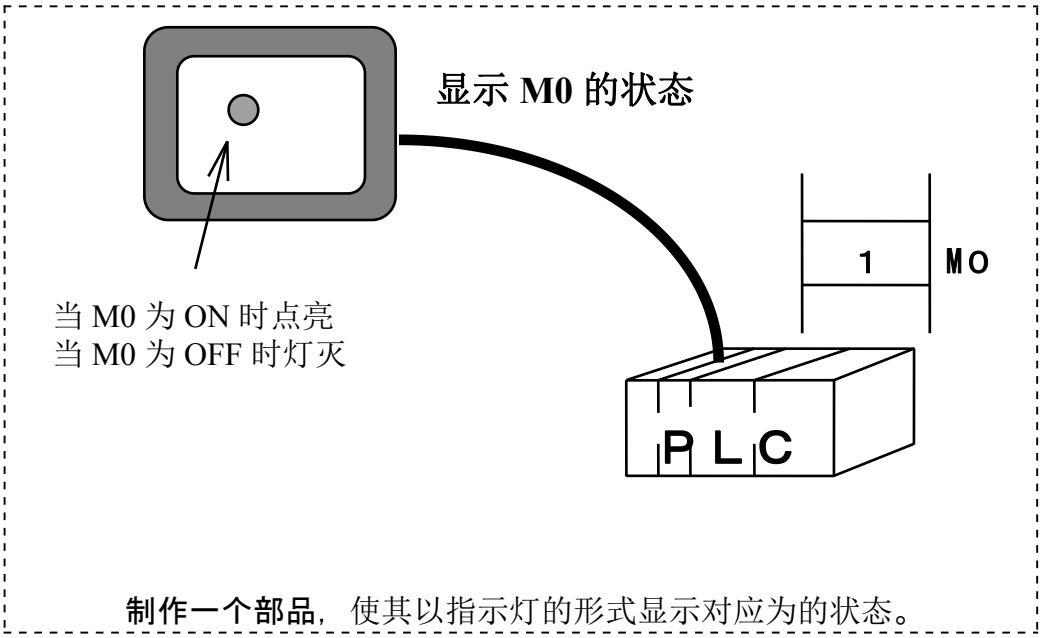
使用模板（template）的程序实例如下：

```
init
    local type%, id@, data%
    cyclic [局号]~[寄存器号]
end init

conf
end conf

evnt
    input type%,id@,data%
    if type%=16 and id@=[局号]~[寄存器号] then
      numdsp ..NUM000,data%
    end if
end evnt
```

“局号”和“寄存器号”作为部品的属性（property of part）显示在部品属性对话框里，在使用时可以为每个放置的部品输入不同的值（寄存器号）。



（2）指示灯

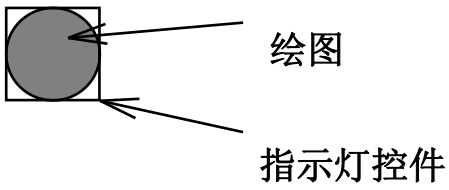


制作一个部品，使其以指示灯的形式显示对应为的状态。

使用的控件：

指示灯控件（LAM000）

外形：



注意：示灯由 OFF 变 ON 时，其颜色也相应地由“OFF 颜色”变成“ON 颜色”。所以给绘图填充颜色时必须用“OFF 颜色”。

与 PLC 相连的指示灯程序如下：

```
init
    local type%, id@, data%
    cyclic [局号]"[与指示灯相连的设备地址]
end init

conf
end conf

evnt
    input type%,id@,data%
    if type%=16 and id@= [局号]"[与指示灯相连的设备地址] then
      lampdsp ..LAM000,data%
    end if
end evnt
```

● Initialization Block

同前面一样，在初始块里使用Cyclic指令。

● Configuration Block

这里什么也不处理！

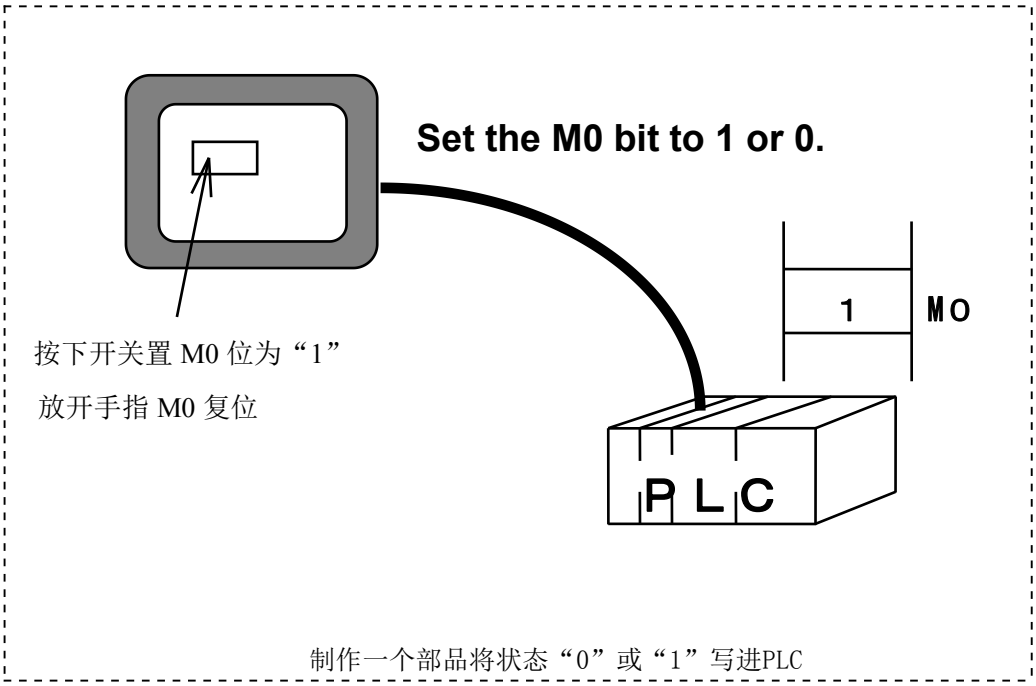
● Event Block

lampdsp ..LAM000,data%

“Lampdsp”指令使用指示灯显示 ON/OFF 状态，当数据是“0”时，指示灯显示“OFF”颜色，当数据为“1”时，指示灯显示“ON”颜色。

试图改变相应内部继电器的状态，观察指示灯颜色的改变。

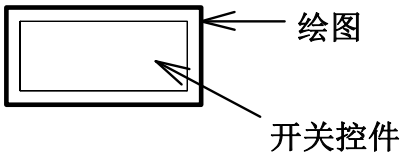
（3）开关



使用控件：

一个开关控件SWT000

外观：



程序如下：

```
init
    local type%, id@, data%
end init

conf
end conf

evnt
   input type%,id@,data%
   if type%=3 and id@=..SWT000 and data%=1 then
      [局号]~[连接设备]=1
   else if type%=3 and id@=..SWT000 and data%=0 then
      [局号]~[连接设备]=0
    end if
end evnt
```

- 初始块（Initialization Block）
  因为是将值写入PLC，所以没有必要使用“Cyclic”指令。

- Configuration Block
  没有处理任务。

- Event Block

```
input type%, id@, data%
```

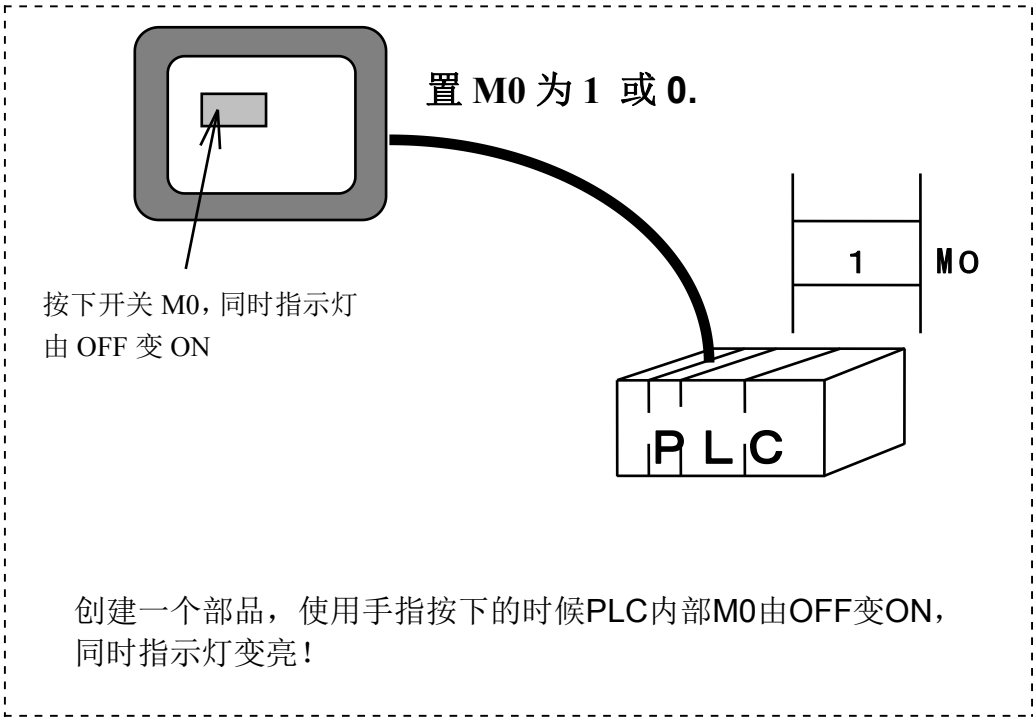
input指令读取来自开关的消息，读取顺序为“type%=3”、 “id@=...SWT000.”、 data%” =ON/OFF (1 or 0)

```
if type%=3 and id@=..SWT000 and data%=1 then
     : : : : : : : : : : : : : :
else if type%=3 and id@=..SWT00 and data%=0 then
     : : : : : : : : : : : : : :
end if
```

解释同前面所述。

[局号]~[连接设备]=1，[局号]~[连接设备]=0
表示将1或0写进“PLC”。

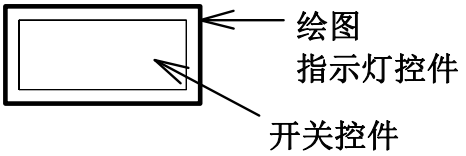
（4）带指示灯的开关



使用控件：

开关控件SWT000和指示灯控件LAM000

外观：



指示灯控件和开关控件相互重叠。

程序如下：

```
init
    local type%, id@, data%
    cyclic[局号]~[连接设备地址]
end init

conf
   cyclic [局号]~[连接设备地址]
end conf

evnt
   input type%,id@,data%
   if type%=3 and id@=..SWT000 and data%=1 then
      [局号]~[连接设备地址] = 1
```

```
        else if type%=3 and id@=..SWT000 and data%=0 then
           [局号]~[连接设备地址]=0
        else if type%=16 and id@=[局号]~[连接设备地址] then
            lampdsp ..NUM000,data%
        end if
    end evnt
```

- Initialization Block

本程序使用cyclic指令监控PLC内部继电器的状态，内部继电器的状态将决定指示灯的状态是ON还是OFF。

- Configuration Block

没有处理内容。

- Event Block

```
    input type@,id@,data%
```

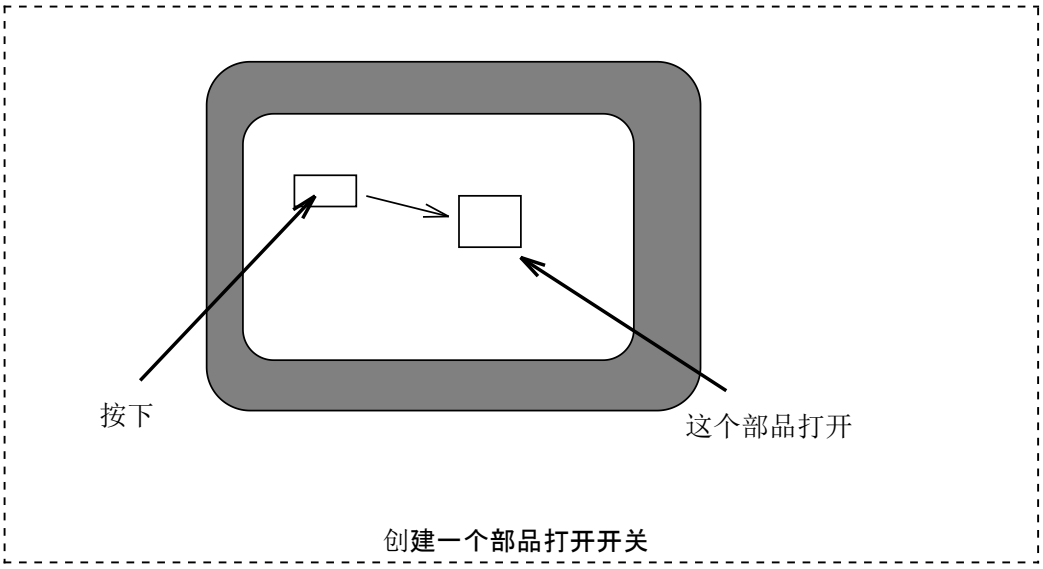
同前面讲述的一样。

```
    if type%=3 and id@=..SWT00 and data%=1 then
         : : : : : : : : : : : : : :
    else if type%=3 and id@=..SWT000 and data%=0 then
         : : : : : : : : : : : : : :
    else if type%=16 and id@=[station-number]~[connected-device-address]
         : : : : : : : : : : : : : :
    end if
```

在本部分，来自开关的消息写进PLC设备，同时来自PLC的信号将决定指示灯的ON/OFF状态。

4. 制作一个部品来控制其它的部品

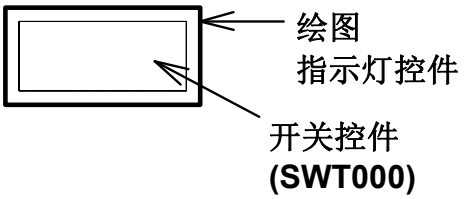
（1）从摸屏上调用其它部品



使用的控件：

一个开关控件SWT000

外观：



从触摸屏上调用其它的部品的程序如下：

```
init
    local type%, id@, data%
end init

conf
end conf

evnt
   input type%,id@,data%
   if type%=3 and id@=..SWT000 and data%=1 then
      open .[要打开的部品名称].,1
   end if
end evnt
```

● Initialization Block

除了声明局部变量之外，没作任何处理。

● Configuration Block

```
input type%,id@,data%
```

“input”指令从开关控件读取信息。

```
if type%=3 and id@=..SWT000 and data%=1 then
     : : : : : : : : : : : : :
end if
```

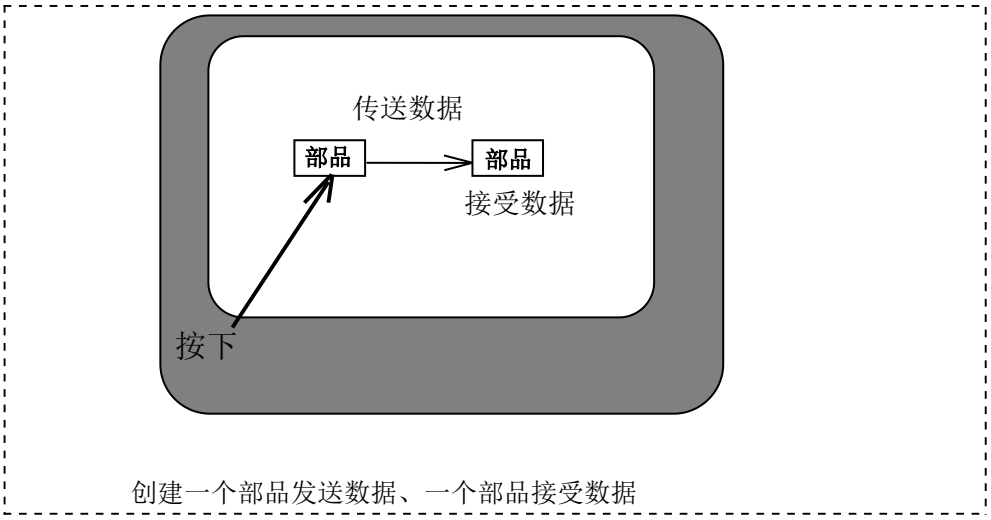
“if”与“end if”之间的内容，只用在开关被按下时才执行。

```
open .[要打开的部品名称].,1
```

“open”指令将由“ID”指出的部品由“Close”状态打开。

如果部品名称后面是“1”，则被打开部品的Configuration Block在部品被打开的时候被执行；反之不执行。被打开的部品名称采用参数形式使部品调用时更加方便。

（2）从（向）其它部品接收（发送）数据

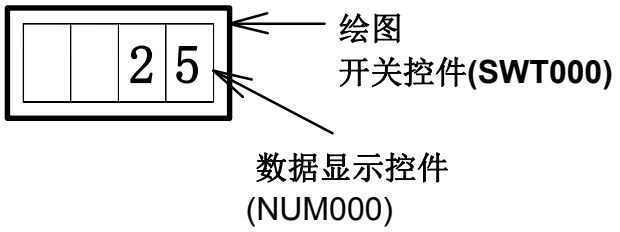


首先创建一个发送数据的部品：

使用的控件：

一个数据显示控件（NUM000）和一个开关控件（SWT000）

外观：



部品用于数据发送的程序如下：

```
init
    local type%, id@, data%
conf
   numdsp ..NUM000,[要显示的数据]
end init

conf
end conf

evnt
   input type%,id@,data%
   if type%=3 and id@=..SWT000 and data%=1 then
      print [要显示的数据]
      send .[目标部品名称].
   end if
end evnt
```

- Initialization Block

  numdsp ..NUM000,[要显示的数据]

  “numdsp”为数据显示命令。

- Configuration Block

  没有处理内容！

- Event Block

```
input type%,id@,data%
```

“input”指令前面已讲过，是从开关控件里读取消息。

```
if type%=3 and id@=..SWT000 and data%=1 then
        print [要显示的数据]
        send .[目标部品名称].
End if
```

“print”和“send”指令在开关被按下时执行。

print [要显示的数据]

“print”指令用于将消息传送给其它部品，消息包括“type”、“ID”、“显示的数值”。如果需要传送两个或两个以上的数据，可以将它们连写并用逗号隔开，如：

例：print 123,234,345

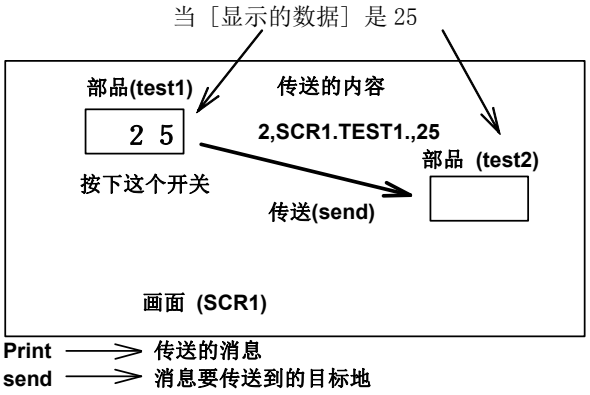
这时，“input”指令要读取3条数据消息，如下：

input type%,id@,data1%,data2%,data3%

其中，data1%读取“123”，data2%读取“234”，data3%读取“345”。

send .[目标部品名称].

Send指令将print指令传送来的数据送给指定的部品（[目标部品名称]），记住：“print”和“send”指令要同时使用。



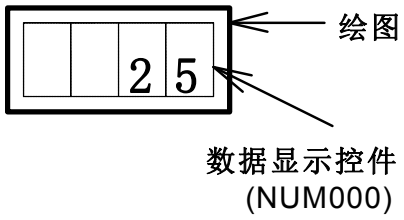
程序到此结束。当开关按下时，程序将包括参数[要显示地数据]、[目标部品名称]在内的消息发送出去。

然后，创建一个接收数据的部品：

使用的控件：

一个数据显示控件（NUM000）

外观：



部品用于数据接收的程序如下：

```
init
    local type%, id@, data%
end init

conf
end conf

evnt
   input type%,id@,data%
   if type%=2 then
      numdsp ..NUM000,data%
   end if
end evnt
```

- Initialization Block

  定义一个局部变量

- Configuration Block

  没有处理内容！

- Event Block

```
input type%,id@,data%
```

“input”指令前面已讲过，是从开关控件里读取消息。

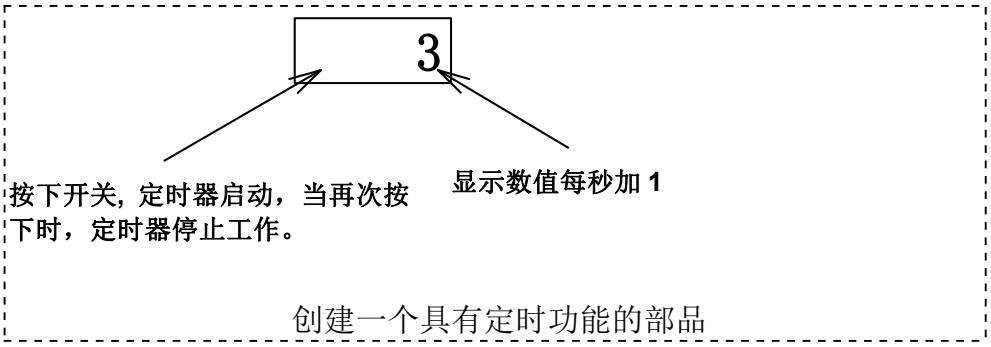
```
if type%=2 then
   numdsp ..NUM000,data%
end if
```

“type%=2”的意思是从部品接收消息，显示的数据由“data%”给出。

程序结束。当test1将数据送过来时，本部品显示test1送来的数据。

同时使用这两个部品，并恰当填写参数，按下开关运行时，两者显示相同的数据！

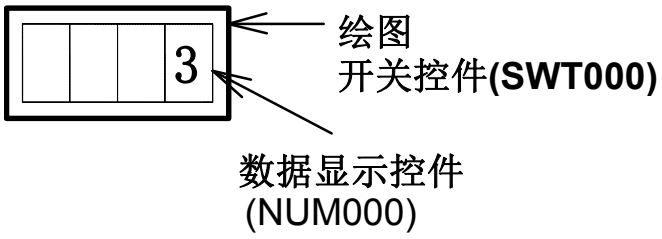
5. 创建一个使用定时器的部品



使用的控件：

一个数据显示控件（NUM000）和一个开关控件（SWT000）

外观：



使部品显示的数据自动加1的程序如下：

```
init
    local type%, id@, data%
    static timeid@
    static flag%
    static number%
    flag%=0
    numdsp ..NUM000,0
end init

conf
end conf

evnt
   input type%,id@,data%
   if type%=3 and id@=..SWT000 and data%=1 then
      if flag%=0 then
         timeid@=opentim()
         settim timeid@,10,1
         starttim timeid@
```

```
            flag%=1
        else if flag%=1 then
            stoptim timeid@
            closetim timeid@
            flag%=0
        end if
    else if type%=4 then
        number%=number%+1
        numdsp ..NUM000,number%
    end if
end evnt
```

- Initialization Block

在本程序块中定义了局部变量和静态变量。

```
static timeid@
static flag%
static number%
```

使用“static”指令使在程序执行的过程中变量的内容得以保留。本例中，本指令保留定时器的ID、定时器的ON/OFF标志、显示的数值。也可以将许多“static”指令合并书写。即将各参数连写，中间用逗号隔开。如下：

```
例：static timeid@,flag%,number%
```

```
flag%=0
```

“flag%=0”是将定时器ON/OFF标志初始化。

```
numdsp ..NUM000,0
```

开始时，在数据显示器里显示“0”。

- Configuration Block

没有处理内容！

- Event Block

```
input type%,id@,data%
```

“input”指令从开关和定时器读取消息。

```
if type%=3 and id@=..SWT000 and data%=1 then
        : : : : : : : :   ← 当开关按下的时候执行
else if type%=4 then
        : : : : : : : :   ← 当读到来自定时器的消息时执行
end if
```

当开关按下或当接收到来自定时器的消息分别执行“then”后面的操作。每接到一次来自定时器的消息，显示器计数一次。（消息每秒钟传送一次。）

当按下开关时执行如下程序：

```
if type%=3 and id@=..SWT000 and data%=1 then
    if flag%=0 then
        timeid@=opentime()
        settim timeid@,10,1
        starttim timeid@
        flag%=1
    else if flag%=1 then
        stoptim timeid@
        closetim timeid@
        flag%=0
    end if
```

当定时器停止（flag%=0）时，紧接“if flag%=0 then”后的操作执行。

```
timeid@=opentim()
```

本函数是获得定时器的ID，定时器的ID用“timeid@”标适。

```
settim timeid@,10,1
```

设置定时时间。定时时间的最小单位为100ms，10表示定时时间为100ms×10＝1s。其后面的“1”表示定时器能反复触发时间。

```
starttim timeid@
```

启动定时器。

以上三条指令需一块联合使用。

```
flag%=1
```

Flag用来标识定时器的状态，并保持其状态。flag%=1表示定时器正在运行。

“else if”之后的程序用于停止定时器的运行。

```
stoptim timeid@
```

使定时器停止向上计数。

```
closetim timeid@
```

关闭“opentim”打开的定时器，并将其返回给系统。

注意：最多可以使用16个定时器。不必使用的定时器应及时返回给系统。

```
        flag%=0
```

因为定时器已经停止计时，所以将标志定时器状态的标识置“0”.

当收到来自定时器的消息时，执行如下程序：

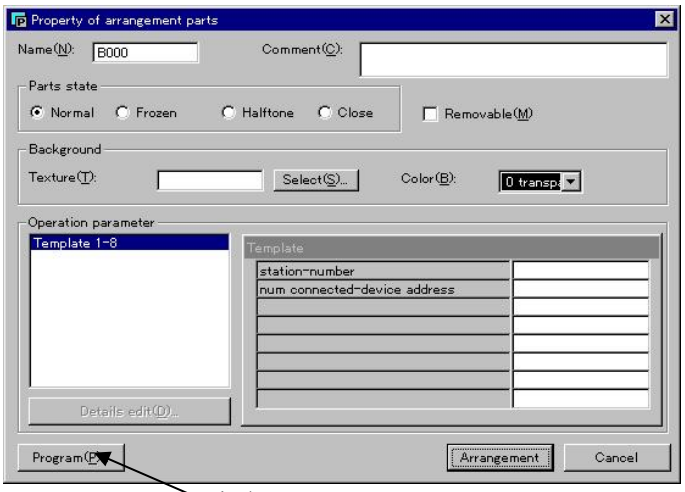
```
    else if type%=4 then
        number%=number%+1
        numdsp ..NUM000,number%
    end if
```

每当定时器发送一次消息，数据显示器单元显示的数值加“1”。

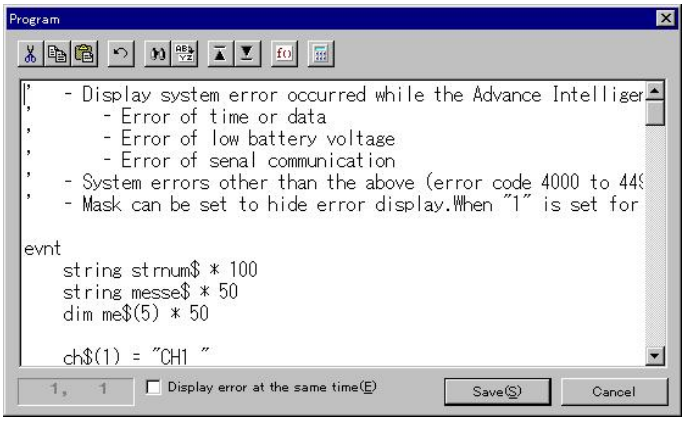
程序到此结束!

6．为画面上的部品编程

进如画面上的部品程序编辑状态的方法是：双击部品 → 弹出部品属性窗口 → 选择“program”进入编程窗口。



打开程序编辑窗口



第三章 编程规则

1．可用字符

程序中可以使用如下字符：半角字母字符（0x20～0x7f ASCII码）、半角Kara字符（0xa0～0xdf ASCII码）、全角字符（两字节代码）。对于全角字符，用双引号括起来的为合法字符；对于Kara字符，设备名称或被双引号括起来才为合法。字母字符可以是大写或小写。然而，当作为字符使用时，大小写各不相同。

大小写不分的情况是，K-basic程序里使用的变量名称、函数名称、和子程序名称。如

Labal 与 LABAL 相同

Variable 与variable 相同。

2．特殊字符

许多字符在K-basic语言里有特殊的含义，这些字符称为特殊字符。特殊字符列表如下：

句号 “.” 将画面、部品、控件名称隔开。也用来做小数点。

将画面、部品、控件名称隔开例：

SCR1.test1.control1

Test1.

小数点例：

1.23, 0.01

&, &0, &H & 和 &0 用来表示一个八进制数

&H 用来表示一个十六进制数

&7 (八进制) 表示十进制里的7。

&10 和 &010 (八进制) 表示十进制里的8。

&H20 (十六进制) 表示十进制里的32。

%, $, !, @ 用来表示变量或函数的类型。 使用时放在变量或函数的最后面。

%: 表示整型变量 (VAR%)

$: 表示字符串变量 (MOJI$)

!: 表示浮点型变量 (FLOAT!)

@: 表示 ID型变量 (ID@)（为K-basic语言特有）

波浪号 “～” Used to delimit a station number and a PLC device name.

01~R2000: 01 表示局号；R2000表示PLC内部寄存器地址。

“[”,“]” 用于引入可以在外部更改的参数

```
conf
  cyclic [局号]~[设备名称]
end conf
```

单引号 “’” 注释的开始。从本符号开始到本行结束之间的内容为注释内容。

```
  global var(3,2)    ’ 全局变量声明
end conf
```

冒号“:” 用来标识一段程序。 标识（Label）用来标识 GOTO 要跳转的目的地或子程序。

```
evnt
  if var% = 0 then goto LABEL
  aa% = bb% + 1
LABEL: aa% = 10
end evnt
```

3．常数

K-basic语言里使用的常量有字符串常量、整型常量、浮点常量、ID常量。

字符串常量　使用双引号括起来的字符串称为字符串常量。一个字符串（一对双引号里）最多可以包含80各半角字符。
“ABCDEF” 和 “1234”,”欢迎光临” 等, 都是字符串常量

整型常量　整型常量可以是八进制、十进制或十六进制的。

&123,&66 (八进制)	& 或 &O 加在八进制的0～7前面。
100,322 (十进制)	可以是 -2147483648 ~ 2147483647 之间的数据。
&H123,&HFF	&H 加在十六进制的0～F前面。

浮点型常量　浮点型常量可以表示从 1.70141E+38 至+1.70141E+38之间的数，有效位最多为6。
浮点型常量可以书写如下：1.23,0.001,-2,3E-4。E-4 表示10的-4次幂。

ID型常量　画面名称、部品名称、控件名称、逻辑设备名称、构件名称、文本名称、PLC设备名称都可以是ID型常量。

- 画面名称、部品名称、控件名称
  画面名称书写如：SCREEN.. ；部品名称书写如：SCREEN.PART。
  控件名称书写如：SCREEN.PART.CTRL。
- 逻辑设备名称
  在触摸屏内部， HST (上位计算机), PRN (打印机), BCR (条形码读入机), MCR (磁卡读入机), TKY (十键键盘), ICC (存储卡), 和 SIO (串行口) 都可以写作逻辑设备。 逻辑设备可以与OIP相连。
- 对于Texture(构件) 和Text(文本), 其已注册的名称可以作为常量。
- PLC设备如 01~R2000 and 01~M10, 等等。

4．常数的声明

在Screen Creator 5 里可以声明变量。对于在程序里要频繁使用的变量，为了减小书写量，可以用一个简单的符号来代替它。在程序中用到该常数时用这个符号代替即可。在程序中，用许用户每次对常数值进行修改。同时也加强了程序的可读性。

常数的声明格式如下：

```
const 常数名称 = 常数值
```

常数名称也是用字符串书写，不过要使用双“#”号标示，如：

```
const #pai# = 3.1415926
```

这时，在程序中所有的pai都代表3.1415926。

注意：常量的声明只能在普通画面中进行，而不能在全局画面的运行程序中声明。否则，在程序编译时会出现语法错误。

5．变量

变量名称里可以使用的字符为字母和下划线“_”，（变量名称里字母部分大小写）注意：变量名称不能以数字开头。变量名称最多不能超过20个字符。
变量必需有自己的类型，所以在变量名称里务必加上$、%、!、@ 之一。实数型常量除外，在其后面不用添加任何声明字符。

（1）变量的分类

字符串变量	用来存放字符串的变量。“$”用来表示变量类型为字符串类型变量。 默认的字符串变量长度最多为20个字符，当需要增加字符串长度时，可以使用STRING命令。
整型变量	用来存放整型数的变量。用“%”标识变量类型。
浮点型变量	用来存放浮点数的变量。该类型变量以“!”符号结束。不以！结束的变量当作浮点型字符变量处理。
ID型变量	用来表示画面名称、部品名称、控件名称、逻辑设备名称、构件名称、文本名称、PLC设备名称的变量都可以是ID型比变量。
数组变量	在字符型、整型、浮点型、ID型变量后加上括号并在里面填写数字，称为数组变量。 数组型变量可以使用DIM命令来声明其大小。一般的声明方法是： GLOBAL VAR$(2,3), VAR1%(10) 数组元素的下标值通常用圆括号形式标注，下标从0开始。VAR1%(10) 表示该数组变量为整型，数组元素为11个，即VAR1%(0)～VAR1%(10)。变量的维数可以是1、2、3….

注意：虽然变量名称相同，但因为后面接上不同的符号（*!, @, %, 和 $*），所以将被当作不同的变量来处理。数组变量也是同样的道理。
例如：***VAR!, VAR@, VAR%, VAR$, VAR!(5), VAR@(5), VAR%(5), VAR$(5)*** 都是不同的变量。

（2）变量的类型

根据存储方法和变量在程序中的有效范围的不同，可以分成不同的类型。

全局变量（Global variables）

带有全局声明的变量。全局变量是所有全局程序中都可以引用的变量。触摸屏上电时，全局变量只初始化一次。整型和浮点型变量初始化为0，字符型和ID型变量初始化为空（即什么也没有）。全局变量的定义方式如下：

GLOBAL VAR%

当在一个工程的多个程序中多次定义时，它们指的是同一个变量。

静态变量（Static variables）

带有静态声明（static）的变量。静态变量只能在声明的程序中使用。 触摸屏上电时，静态变量只初始化一次。整型和浮点型变量初始化为0，字符型和ID型变量初始化为空（即什么也没有）。 静态变量的定义方法如下：

STATIC VAR%

停电记忆型变量（Backup variables）

除了即使OIP断电其内容也能保持之外，停电记忆型变量具有全局变量的几乎所有特性。停电记忆型变量的值即使重新上电也不再次初始化。然而，当画面数据下载后首次运行时，其值初始化为0。停电记忆型变量的定义方法如下：

BACKUP VAR%

当在一个工程的多个程序中多次定义时，它们指的是同一个变量。停电记忆型变量仅对带有内置存储器的触摸屏有效。对于不带内部存储器的触摸屏，其功能将同全局变量（Global variables）一样，即重新上电后变量仍然重新初始化。

触摸屏的停电记忆变量和RAM文件（MS-DOS文件系统和内存文件）是通过使用停电记忆存储器来实现的。因此，使用于停电记忆变量和RAM文件的内存总和不能超过触摸屏内部停电记忆存储器大小。使用于RAM文件的存储器大小通过触摸屏上系统设置（system setup）下的RAM文件设置（RAM file setup）来进行。

型　号	停电记忆内存大小	停电记忆可否
GC-56LC/ GC-55EM	63K	可以
GC-56LC2/GC-55EM2	255K	可以
GC-53LC /GC-53LM	没有	不能
GC-53LC2/GC-53LM2	255K	可以
GC-53LC3/GC-53LM3	255K	可以

局部变量
（Local variables）

使用LOCAL声明或未被声明的变量，该变量只能在局部画面而不能在全局画面程序中使用。
虽然可以使用DIM来声明局部变量，但在Screen Creator 5里，应尽量使用LOCAL来定义局部变量。但使用DIM可以使 Screen Creator 5程序同GCSGP3程序相兼容。
局部变量在每次程序执行时都初始化一次。整型和浮点型变量初始化为0，字符型和ID型变量初始化为空（即什么也没有）。局部变量的定义方法如下：

LOCAL VAR%

自动变量
（Auto variables）

自动变量通过AUTO调用。
自动变量只能在函数中定义并只能由函数引用，在每次程序执行时都被初始化。整型和浮点型变量初始化为0，字符型和ID型变量初始化为空（即什么也没有）。自动变量的定义方式如下：

AUTO VAR%

（3）编辑时的变量检查和解释

当画面数据创建以后，编辑器对程序语法进行分析处理。如果程序中包含由全局变量、静态变量、或其它特殊声明，处理将根据这些声明来处理。某些情况下，这些处理和解释是很严格的，所以如果在编写程序时不严格遵守，程序可能不能如愿。下面说以下那些解释是严格的：

① 被全局画面引用但未被声明是全局、静态、停电记忆型，这种变量将被系统解释成全局变量。
② 非全局画面或部品程序中未被声明为全局、静态、停电记忆、局部或自动型，这种变量将自动被解释成局部变量。

以上也是普通BASIC语言的通用特性。然而，这些特性对许多编程者来说都是不希望的。例如，在程序中使用了不正确的变量名，将自动生成局部或全局变量，而编程者有时觉察不到。这类错误很难发现，即使是程序在编辑时也不容易。
为了避免这类问题的产生，在程序编辑时当发现未声明的变量时，希望Screen Creator 5能给出错误提示。在程序的中加入“LOCAL CHECK”声明。关于“LOCAL CHECK”用法，请参考《命令手册》。

（4）变量初始化

在Screen Creator 5 中，变量可以声明时进行初始化，变量初始化的方法如下所示：

例：STATIC VAR% = 12

对于数组变量，初始化相对复杂。可以使用大括号“{}”列出。对于一维数组，初始化从下标为0的元素开始。

例：GLOBAL ARRAY%(5) = {0, 1, 2, 3, 4, 5}

对于多维数组，书写时确保下标从右递增。

例：GLOBAL ARRAY%(2, 3) = {{0, 1, 2, 3 }, {4, 5, 6, 7}, {8, 9, 10, 11}}

GLOBAL ARRAY%(1, 2, 3) ={{{0, 1, 2, 3 }, {4, 5, 6, 7}, {8, 9, 10, 11}},
{{12, 13, 14, 15}, {16, 17, 18, 19}, {20, 21, 22, 23}}}

如果初始化的值的类型与变量类型不一致，初始化时以声明的类型为准。

ID变量也可以初始化，即初始化适合与所有类型的变量。然而，如果对停电记忆型变量进行初始化，则停电记忆功能将打不到预期效果。

变量初始化的位置取决于变量的类型和声明的地方。全局变量、静态变量和停电记忆型变量，在应程序块执行前初始化；对于局部变量，在变量声明时初始化。因此，如果局部变量是在CONF BLOCK程序块或EVNT BLOCK内部声明，则变量将在每次本程序块执行时进行初始化。自动（AUTO）变量在每次本变量所在函数被调用时执行初始化。

6．表达式和运算符

变量和常数之间可以使用下列运算符：

● **算术运算符**

^ (指数运算符)	指数运算式写法为 X ^ Y，表示x的y次幂。
- (负号)	-100,-VAR! 整数和浮点数都转化成负数。
* (乘号)	VAR1*VAR2 VAR1 乘以VAR2.
/ (除号)	VAR1/VAR2 VAR1 除以 VAR2.
¥ (除号)	VAR1¥VAR2 VAR1 除以VAR2，并将商取整。
+ (加号)	VAR1+VAR2 VAR2加VAR1.
- (减号)	VAR1-VAR2 VAR1减VAR2
MOD (取余)	VAR1 MOD VAR2 取VAR1 除以VAR2的余数.

● **关系运算符**

关系运算符用于两个数值的比较。比较结果为1（真）或0（假）。

= (等于)	= 用法如 VAR1=VAR2。 当两值相等时结果为真，否则为假。
<> (不等于)	<>用法如VAR1<>VAR2. 当两值不相等时结果为真，相等时结果为假。
< (小于)	< 用法如 ：VAR1<VAR2. 当前者小于后者时结果为真。
> (大于)	> 用法如： VAR1>VAR2. 当前者大于后者时，结果为假。
<= (小于等于)	<=用法如： VAR1<=VAR2. 当前者小于或者等于后者时结果为真。
>= (大于等于)	>= 用法如： VAR1>=VAR2. 当前者大于或者等于后者时结果为真。

● **逻辑运算符**

NOT 逻辑非	NOT 用法如： NOT VAR%. 逻辑非用于数学表达式（变量或常数）前。意思是将数值的各位取反。
AND 逻辑与	AND 用法如： VAR1% AND VAR2%. VAR1% and VAR2% 将两者的各位进行与运算。
OR 逻辑或	OR用法如：VAR1% OR VAR2%. VAR1% and VAR2% 将二者的各位进行或运算。
XOR 异或	XOR 用法如： VAR1% XOR VAR2%. VAR1% and VAR2% 将二者的各位进行异或运算。

后三者是对两个数值的各对应位（bit）进行运算，NOT是对一个数值的各位进行运算。

● **字符运算符**

-用于字符之间的连接

+	+ 用法如： VAR1$+VAR2$. + 用于将两个字符串连接起来。VAR$=VAR1$+VAR2$表示将后两个字符串合并并赋给前一个变量。

-用于字符串间的比较

两个字符串进行比较，结果为真（1）或为假（0）。

=	=用法如： VAR1$=VAR2$. 用于判断两个字符串是否相同。相同时比较结果为1，否则为0。
<>	<> 用法如：VAR1$<>VAR2$. 用于判断两字符串是否不相同，不相同时比较结果为1，否则为0。
<	< 用法如： VAR1$<VAR2$. 当 VAR1$ 小于VAR2%时，结果为真。
>	> 用法如： VAR1$>VAR2$. 当VAR1$大于VAR2%, 结果为真。
<=	<=用法如： VAR1$<=VAR2$. 当VAR1$ 小于或者等于VAR2$, 比较结果为真。
>=	>= 用法如： VAR1$>=VAR2$. 当VAR1$ 大于或等于VAR2$，比较结果为真。

两个字符串以字节为单位依次进行比较，当发现有不同的字符时，就要比较其大小。当在比较时发现某字符串比另一个要短时，则判断该者为小。

● **运算符的优先级**

根据级别高低排列如下：

表达式	圆括号内的表达式
函数	系统定义的函数或用户函数
-	负号
^	指数运算符
*, /, ¥	乘除
+, -	加减
MOD	取余
=, <>	关系运算符
NOT	逻辑非
AND, OR, XOR	连接、或、 异或

注意：ID变量和常数之间比较时只能使用“＝”。

7．类型的转换

当整型变量和浮点变量进行运算时，或将整数或浮点数赋给不同的变量时，就会发生类型的转变。

● 指定转换

以下例子将浮点数转换成整型数：

```
VAR1% = 2.45
VAR2% = 2.56
```

这时，指定VAR1%为2，VAR2%为3。

实数在转化成整数时自动进行圆整，圆整后的值就是整型数。

● 逻辑运算

浮点数在进行逻辑运算时，先转化成整数然后再进行运算。如：

```
VAR% = 23
FLOAT! = 12.35
    VAR%  AND  FLOAT!
```

经计算后结果应是 23 AND 12。

● 其它

当将整数值转化成浮点数值，然后再次转换成整数，这时会产生有效位丢失。在OIP里面，有效位数为6。如：

```
VAR% = 99999999
FLOAT! = VAR%
    VAR% = FLOAT!
```

执行的结果是VAR% =100000000。

8．标签（Label）

标签用来标注“程序的跳转目的地”或“子程序名称”等。标签名地指定同变量名一样。与正式程序间用冒号（：）隔开。使用方法如下：

```
evnt
   input ty%,id@,dat%
   if dat% = 1 then goto LABEL1
   gosub SUBNAME
   ........
   .......
LABEL1: dat% = 20
end evnt

SUBNAME:
  dat% = 10
  return
```

9．子程序

子程序是写在Evnt block外并由其调用的一段程序。子程序从标签（Label）名称开始，以“Return”结束。注意，在标签名称行，不能写程序！在同一个程序种可允许写有多个子程序。如下：

```
conf
   ....
   Description of configuration block
   gosub SUB1
   ....
end conf
evnt
   ....
   Description of event block
   gosub SUB10
   ....
end evnt
SUB1:
    ....          Subroutine body

    RETURN
SUB10:
    ....          Subroutine body

    RETURN
```

同变量一样，子程序也分局部子程序和全局子程序。

- 全局子程序

  全局子程序写在全局画面程序里，全局子程序可由可画面和部品程序调用。全局子程序可以使用的变量为全局变量和静态变量。当局部画面而非全局画面程序要调用全局子程序时，只有声明了“global”或“static”的变量才能作为全局子程序的变量。

- 局部子程序

  写在局部画面而非全局画面程序里的子程序。局部子程序只能在本程序里调用。如果全局子程序和局部子程序同名，当调用局部子程序时，实际上将调用全局子程序。为了确保当局部子程序和全局子程序同名时系统给出提示，可声明“LOCAL CHECK”。

10．用户函数

Screen Creator5 支持用户函数，参数通过调用体输入，然后返回一个值给调用体。

（1）用户函数的定义

用户函数的定义方法如下：

```
function 函数名称 [类型声明] (参数1, 参数2, .... )
      函数功能程序
end function
```

function～end function 之间的程序称为“函数块”。同INIT Block、Conf Block、Evnt Block一样，它也是程序的一个部分。在其它的程序块中，也可以拥有“函数块”。
函数名的写法同变量名一样，在函数名后应加上表示函数类型的符号$、%、!或@，以表示函数返回值的类型。实数函数例外，无需注明。
参数1, 参数2, ...圆括号里的参数由调用体给出。参数的类型由类型声明字符给出，如果没有，参数将被视为实数。函数调用体可使用变量、常数、计算表达式等作为参数。
如果参数是变量，则函数将自动代替变量的值，那么参数可能会是变动的。在这种情况下，即使外面对其没施加影响，调用体的参数也在变化。也就是说，使用这种初始变量作为参数，虽然称为参数，但不能称为变量。
如果参数是参数或计算表达式，则将这个值代替参数然后执行函数。即使外面对其没施加影响，如果用值代替参数，参数也将发生变化。换句话说，函数认为这时带有默认值的变量（也就是，自动变量）。
当函数返回值后，它又可以作为其它函数的变量或参数。
当程序执行到“end function”后，它将把处理权重新交给调用体。在程序中使用“exit function”可以结束函数的处理。

以下是一个用户函数实例：

```
function my_div%(a%, b%)
   if b% = 0 then
      if a% < 0 then
         my_div% = -217483648
      else
         my_div% = 217483647
      end if
      exit function
   end if
   my_div% = a% / b%
end function
```

（2）用户函数的定义位置和其有效范围

可以调用用户函数的程序类型和函数的有效范围随着函数的定义位置不同而各异。用户函数分为如下三类：

- 全局函数
  它写在全局画面函数里。任何画面和部品程序都可以调用。
- 局部函数
  写在局部画面程序里。只能被其所在的程序调用。
- 库函数
  这类函数写在Screen Creator 5控制下的函数库里。可以被任何程序调用。

（3）用户函数的调用

在所有的函数块（Init block）之前，要先声明要调用函数的类型（即函数模型声明）。函数声明的格式如下：

函数名称[函数类型符号]（参数1，参数2，……）

如上面（1）里用户函数的声明如下：

```
declare my_div% (a%, b%)
```

函数调用的优先级为：库函数－全局函数－局部函数，如果有几个不同类型的函数同名，则按上述优先级调用。所以，应尽量避免函数同名。为了在编译时能由系统检查出同名函数的存在，可在程序的开头声明“LOCAL CHECK”。

（4）用户函数里的变量声明和有效外部变量

在程序里可以声明自动变量，也可以声明其它类型。
只要对本主程序有效，全局变量和包含在本程序中的局部变量，对其都是有效。换句话说，如果在程序中作了声明，全局变量、静态变量和停电记忆型变量都是有效的。还有，声明或未声明的局部变量都为有效。注意，除非在程序中声明为自动变量，否则不能引用库函数。

11．程序运行

当程序里指出的部品或画面收到消息后，触摸屏程序开始运行。Screen Screator 5 提供如下消息：

● 部品和画面消息
- 部品和画面程序可以执行 SEND 指令来向部品或画面发送消息。

● 开关消息
- 当放在部品里的开关控件置 ON 或 OFF 时会发出消息。
- 开关控件被触摸时也发出消息。

● 内部定时器消息
- 当设定的时间过了之后，会发出消息。
- 程序中必须打开内部定时器才能发送消息（参考 OPENTIM）。

● 报警消息
- 当到了设定的时刻时，发出消息。
- 如何启动报警，详见“SETALARM”指令详解。

● PIO 消息
- 当并行输入状态变化时，发出消息。
- 要接收来自并行输入的消息，使用哪个 PIO 位必须在程序中预先声明。详见“OPENPARALLEL”。

● 无协议通信消息
- 当无协议通信数据接收完毕时发出消息。

● 采样消息
- 当进行采样的控件读取数据时发出消息。

● PLC 消息
- PLC 设备的值作为消息来传送。在 OIP 与 PLC 通信时，当设备值发生变化时发出消息。
- 如果 PLC 内部许多设备的值都发生变化，这些变化只有在 OIP 与 PLC 通信后才能被检测到。因此，消息并不一定按照设备值的变化顺序来发送。
- 要接收来自 PLC 的消息，应预先在程序声明使用 PLC 内部的哪个设备。详见“CYCLIC”。

● 条形码/十键键盘消息
- 当上述设备同部品或画面通信启动时发送消息。消息的内容为传送的数据。
- 为了能接收上述消息，程序必须预先声明。详见“OPENCOM”

● 上位机消息
- 当上位机同部品或画面的通信开始时发送消息。消息的内容为传送的数据。
- 为了能接收来自上位机的消息，程序必须预先声明。详见“OPENCOM”

注意：消息也可以被发送到隐藏画面，控制程序在接收到消息后也同样运行。

12．消息的格式

消息是触摸屏程序执行的触发器。每条消息包括发送者、身份号（ID）、和发送的数据三个要素。注意，发送的数据可能不止一条！

三种类型如下：

a.表示发送者类型的数值 （整型）

b.表示发送者身份的值 （ID 型）

c.数据本身 （要发送数据的类型）

程序使用“input”指令来读取消息。例如，数值 10 来自画面 SCREEN 上的 PART 部品，则 INPUT 指令读取消息的格式如下：

```
INPUT TYPE% , ID@ , DATA%
```

TYPE%：值为 2。表示消息来自部品时 TYPE%＝2

ID@： SCREEN.PART.。表示发送消息的是画面 SCREEN 上的 PART。

DATA%： 数据本身，这里为“10”。

现对不同的消息分述如下：

- **Screens（画面）**

  发送者类型： 1

  发送者ID： 画面名称

  数据： 用“PRINT”指令指出的数据

- **Parts（部品）**

  发送者类型： 2

  发送者ID： 部品名称

  数据： 用“PRINT”指令指出的数据

- **Switchs（开关）**

  发送者类型： 3

  发送者ID： 开关控件名称

  数据：（单开关） 1（ON时），0（OFF时）

  数据：（多开关） 开关号 1（ON时），0（OFF时）

  数据：（选择开关）开关号

  选择开关的开关号表示处于ON状态的的开关号。如果为0，表示所有的开关都处于OFF状态。

- **Timer（定时器）**

  发送者类型： 4

  发送者ID： 使用OPENTIM打开的定时器的ID

  数据： 1（固定）

- **Alarms（部品）**

  发送者类型： 5

  发送者ID： 使用OPENALRM打开的报警器的ID

  数据： 1（固定）

- **PIO**

  发送者类型： 6
  发送者ID： 表示并行口的ID
  数据： 1 满足OPENPARALLEL设定条件的位数
  2 位状态（1：ON；0：OFF）
  3 PIO通道编号（0～3）

- **无协议通信**

  发送者类型： 7
  发送者ID： —
  数据： 1 端口号
  2 位状态（1：ON；0：OFF）
  3 接收到的字节数

- **Sampling（采样）**

  发送者类型： 9
  发送者ID： 进行采样的控件的ID
  数据： 采样值

- **PLC和Memory Link**

  发送者类型： 16
  发送者ID： PLC设备或内部存储器表
  数据： 设备或存储器里的值

- **条形码读入机**

  发送者类型： 18
  发送者ID： 逻辑名称“BAR”
  数据： 来自条形码读入机的字符串

- **十键键盘（Ten-key pad）**

  发送者类型： 20
  发送者ID： 逻辑名称“BAR”
  数据： 来自键盘的字符串

- **上位机（指令通信）**

  发送者类型： 22
  发送者ID： 逻辑名称“HST”
  数据： 来自上位机的阿数据

记住：程序中使用 INPUT 指令读取消息！

13. 程序块（Block）

K-Basic程序包括如下程序块：初始化块（INITIALIZATION（INIT ～ END INIT)), CONF Block (CONF ～ END CONF), 事件块(Event Block(EVNT ～ END EVNT)), 子程序(标签: ～ RETURN)和函数(FUNCTION ～ END FUNCTION).下面对各部分的结构和功能解释如下：

```
declare func%(a%, b%)          ’函数声明
init                           ’初始化程序块
   static var1% = 10
   global var2% = 20
end init
conf                           ’Conf block
   var2% = 30
end conf
evnt                           ’Event block
   input type% , id@ , data%
   if type% = 3 then
      var1% = func%(data, var2%)
      ...............
      ...............
   endif
end evnt
SUB1:                          ’子程序块
    ....
    RETURN
function func%(a%, b%)         ’函数块
...............
...............
end function
```

- Initialization block (INIT ~ END INIT)
  - INIT 块只在 Conf 和 Evnt 块首次执行时执行一次。
    用于对 Conf 和 Evnt 块中将要使用的变量进行声明或初始化。
- Configuration block (CONF ~ END CONF)
  - 画面或部品程序的 Conf 块只在画面或部品开始显示时执行一次。在它们显示的过程中本程序快并不执行。当显示不同的画面时，再执行一次。
  - 全局画面及其部品程序的 CONF 块仅在系统开始运行时执行一次。
  - Conf 程序块进行初始化等处理。
  - 处于 CLOSE 状态的部品程序的 CONF 块不处理，只有当该部品被打开时处理一次。
- Event block (EVNT END ~ EVNT)
  - 程序在收到消息后执行本程序块。在收到消息后进行何种处理由程序给出。
  - 全局画面程序里不能有本程序块！

注意：如果消息被送到尚未显示的画面，则画面程序的 CONF 块不执行，而 EVNT 块执行。这时写在 CONF 块里的初始化就无效。因此，尽量在 INIT 块里进行初始化。

14. 设备和通信

在 K-Basic 程序中，局号和设备名之间用“~”连接，如

VAR%=01~R2000:表示读取局号为01、设备为寄存器R2000里的数据。
01~R2000=40: 将40写入局号为01号PLC的R2000寄存器里。

通信就是用来对设备内容进行读写，Screen Creator 5 提供如下两种通信方法：

- Cyclic 通信
  - OIP 要经常同 PLC 通信从设备里读取数据，当要读取的单元内容发生了变化时，将会发出一个消息。
  - Cyclic 通信即使在K-Basic程序没有被执行时也照样进行。
  - Cyclic 通信要在 INIT 块中用 CYCLIC 指令进行声明。
  - Cyclic 通信不能用于写操作。

- Event 通信
  - Event 通信在接到消息后进行。
  - Event 通信要通过程序执行来实现。
  - Event 通信可以用于数据的读写。

在触摸屏里，总是全局画面和某个局部画面重叠显示。这时，全局画面同局部画面之间的通信如下：

- 全局画面通信
  - 全局画面上的 Cyclic 通信的执行与局部画面无关。

- 局部画面通信
  - 只有当前画面上声明的 Cyclic 通信才能使用。
  - Event 通信在当前画面对设备的内容进行读写时进行。
  - 如果非当前画面的程序激活，并在对设备进行读写，数据可能从非程序指定的设备读/或写进非程序指定的设备。为避免这种情况的发生，编程时不要让数据的读写在未显示的画面上进行。如，不要将定时、报警、图形采样等消息送给未显示画面。如果却有必要，应在主画面上处理。

15. 内部存储器表（Memory Tables）

内部存储器表用于上位机与 Memory Link 之间的通信。该表以字为单位（2 字节），共有 2048 个内存单元（地址从 0～2047）。

下面讲述用 K-Basic 程序如何对其进行访问。

（1）书写形式

00~MTBL(0): 内部存储器表第0个单元。
00~MTBL(2047): 内部存储器表第2047个单元。
00~MTBL(NO%): 以变量 NO% 表示的内部单元。

（2）对某个单元进行读写

- ABC=00~MTBL (100)
  将第100个内部单元里的值读进并赋给变量ABC。

- 00~MTBL (200) = 23
  将常数23写入内部第200个单元。

- 00~MTBL (ABC) = XYZ
  将变量 XYZ的内容写入以变量ABC指出的内部单元里。

（3）同时对多个单元进行读写

- BREAD 00~MTBL (100), 20, ABCD (XY)
  将从第100个单元开始的20个内部单元里的数据读入到以XY为下标、变量名为ABCD的数组里。

- BREAD 00~MTBL (START), NUMS, ABCD (XY)
  从内部单元读取数据，起始于第100个单元，共NUMS个单元，读到数组ABCD (XY)里。

- BWRITE 00~MTBL (100), 20, ABCD (XY)
  从以XY为下标、名为ABCD的数组里读取20个数据到以第100个内部单元开始的20个单元里。

- BWRITE 00~MTBL (START), NUMS, ABCD (XY)
  从以XY为下标、名为ABCD的数组里读取NUMS个数据到以第START个内部单元开始的NUMS个单元里。

16.文件系统

本部分讲述文件系统的特性及访问文件系统的命令。
在 OIP 里停电记忆存储器和外部存储器卡都可以作为 MS-DOS 兼容性文件系统来使用。这些称为“MS-DOS 文件系统”。

驱动器名称的指定：

驱动器名称	说明
Drive A:	是停电记忆存储器的一部分，可以用作 MS-DOS 文件系统。
Drive E:	存储器卡驱动器，同存储器卡串行连接。

另外，还可以使用“Memory files(内存文件)”。在内存文件里，内存映象被从系统内存读取或被写入系统存储器。要访问存储器文件，使用文件名“Memory”。

在停电记忆存储器上创建的文件系统或存储器文件也称为 RAM 文件,只有带有内部停电记忆存储器的 OIP 才能使用 RAM 文件。停电记忆存储器上存储的数据即使在停电后也不会丢失。

（1）文件系统注意事项

- **存储器（内存）**
  内存文件不是以文件系统的形式进行管理。虽然使用相同的系统内存，但 Drive A 和 Memnory 不能同时使用。

- **将停电记忆存储器用于 RAM 文件注意事项：**
  用于停电记忆变量的停电记忆存储器容量同用于 RAM 文件的停电记忆存储器容量之和不能超过内部停电记忆存储器容量。

- **更改 Backup 的设置**
  当存储器设置发生变化时，停电记忆的数据将被清除。

- **文件格式化**
  在使用文件系统之前，务必将文件格式化。使用“FORMAT”指令对文件进行格式化。

- **存储器卡格式化**
  OIP 存储器卡使用 MS-DOS 兼容性格式创建文件，上面的数据可以通过运行在个人电脑上的 MS-DOS 系统来读写。然而，最好创建一个目录，以使文件路径名和文件名的总长度不超过 128 个字符。
  OIP 存储器卡不支持过长的文件名，（Windows 系统支持）。所以文件名不宜太长!

（2）文件的指定

驱动器的指定：A:,E:
目录的指定：A:\ABC, A:/ABC/DEF
文件名：ABCDE.DOM（包括文件名（最多 8 个字符）和扩展名（3 字符））。

17. 注意事项

在使用 K-Basic 编程时，请注意如下事项：

- **颜色和填充模式代号**
  用于改变图形或显示的颜色和填充模式。

- **画面切换时注意 1：**
  当画面被切换到另一幅画面时，本画面上的瞬态开关（Momentary Switch）如果为 ON，将被强制置 OFF，而不管开关的模式（输入允许、输入禁止或半色调）。

- **画面切换时注意 2：**
  当进行 CYCLIC 通信的画面显示时，所有执行 CYCLIC 通信的设备都将发送消息。

- 消息发送给未显示画面的部品时，程序在背后执行。如果企图执行一个不可执行的指令，则会出现错误！

- 如果程序中产生了无限循环，开关功能及通信将停止。

- 放有开关控件的部品在移动时只能以栅格（grid）为单位。

捷太格特电子(无锡)有限公司

JTEKT ELECTRONICS (WUXI) CO.,LTD.

地址：江苏省无锡市滨湖区建筑西路 599 号 1 栋 21 层　邮编：214072

电话：0510-85167888　传真：0510-85161393

网址：**https://www.jtektele.com.cn**

JELWX-M9017A