JTEKT

Value & Technology

GC-A2 系列触摸屏

K-Basic

（画面编辑软件 SCREEN CREATOR ADVANCE2）

参考手册

[第二版]

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

前　言

感谢您选用光洋电子 GC-A2 系列工业触摸屏。本资料是有关 GC-A2 系列触摸屏时画面编辑软件基本操作方法的说明资料。在使用本资料时，请配合阅读参考其他有关 GC-A2 系列触摸屏产品的技术资料。

GC-A2 系列触摸屏相关的技术资料如下。

1、《GC-A2 系列触摸屏入门手册》

有关 SCA2 画面编辑软件的基本操作方法的手册资料。内带有关 GC-A2 硬件规格的说明资料。

2、《SCA2 画面编辑软件使用手册》

有关 SCA2 画面编辑软件各功能/使用方法的详细说明资料。

3、《GC-A2 触摸屏通信连接手册》

有关 GC-A2 和 PLC 及上位计算机通信方法，以及和周边设备通信连接的说明资料。

包括如何在 SCA2 软件中进行各种通信连接的设置的介绍资料。

4、《SCA2 画面编辑软件标准部品手册》

SCA2 中所有标准部品的详细介绍资料。

5、《SCA2 画面编辑软件控件参考手册》

SCA2 部品制作时使用的控件的参考说明手册资料。

6、《SCA2 画面编辑软件 K-BASIC 参考手册》（本手册）

有关画面或部品动作程序（K-BASIC)的编制说明资料。包括所有 K-BASIC 指令的说明。

7、《GC-A2 触摸屏故障处理出错代码手册》

有关在使用 GC-A2 系列触摸屏时出现故障时的出错代码意义说明，处理方法介绍等说明资料。

包括在使用 SCA2 软件时的限制事项。

8、《GC-A2 特殊功能手册》

GC-A2 的一些特殊功能的介绍资料。包括：计算机 RUNTIME 运行，共有存储器，梯形图工具，PLC 的 I/O 监视，远程桌面，备注部品，原来老款触摸屏工程文件的读入等多个便利功能的介绍资料。

我们致力于使我们的资料正确完整，但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，我们可能会在未通知客户的情况下对本手册的任何部分进行修改。

我们努力认真编制本手册资料，但也不排除有错误和不足的地方。我们也热忱欢迎用户对本手册中错误和不当之处提出修改意见，为此对您表示感谢！

我们对您在利用本资料，使用 SCA2 画面编辑软件编制工程画面并使用 GC-A2 产品作如下声明：

1） 我们对 GC-A2 本体和 SCA2 软件拥有完全知识产品或已经付出了许可费用， 请不要随便读取、解析、复制有关内容。

2） 光洋电子对正确和不正确使用 SCA2 软件以及 GC-A2 产品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！

3） 对于利用本手册资料引起的有关工业所有权问题，本公司不承担任何责任。

4） 禁止复制、转载本手册的全部或部分内容。

5） 在使用本手册和 GC-A2 系列产品时有任何疑问，可与本公司本部或当地办事处联系。

6） 技术咨询联系方式：

地址：江苏省无锡市滨湖区建筑西路 599 号 1 栋 21 层

光洋电子（无锡）有限公司

联系电话：0510－85167888－2055/2075

传真：0510－85161393

进行咨询时，为了保证问题有效得到解决，请告知公司名称，使用产品型号、生产批号，详细问题内容（系统构成、现场现象、出错代码、现场环境等）。

关于本手册资料的记号

本手册资料中使用以下记号用于着重提示一些重要的信息.

	**警告**	如果忽视本记号所示内容进行了误操作,有可能会引起死亡、人身重大伤害发生重大的事故。
	**注意**	如果忽视本记号所示内容进行了误操作,有可能会引发人身伤害或发生财损失的事故。
		表示使用上的一般注意事项。
		表示一般的禁止事项。
		表示强调或指示。
注：		解说或补充事项说明。

关于本资料中所用简称

本手册资料说明时使用以下简称。

GC-A2	指GC-A2系列工业触摸屏本体。
SCA2	指画面编辑工具软件 SCREEN CREATOR ADVANCE 2 。
PLC	可变程序控制器简称。
通信连接单元	指连接GC-A2本体和PLC的通信单元。各厂家对该产品的称呼不定相同，本料中统一称为通信连接单元。
功能存储器	PLC 的输入/输出线圈、内部线圈、定时器、计数器、数据寄存器等统称为能存储器。
计算机	本资料中台式计算机、笔记本计算机统称为计算机。

关于本资料中所用专用名词

本资料在介绍说明 GC-A2 和画面编辑工具软件 SCA2 时，会用到如下专用名词。

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

安全注意事项

在使用 GC-A2 系列产品时，请务必注意遵守以下安全注意事项。

【关于使用环境】

	警告
	请不要在有可燃性、爆炸性气体的环境里使用， 否则有可能引发人身事故和产生火灾。
	请不要把本产品用于有关人身安全的用途。 要保证万一出现故障或误动作，也不会对人体产生伤害。
	注意
	请在规格规定的环境（振动、冲击、温度、湿度等）下保管、使用本产品。 超范围使用，有可能引发火灾，损坏产品。
	请在充分了解熟悉产品的基础上，使用本产品。

【关于安装和接线】

	警告
	在设计系统时，要设计完善的外部安全保护回路。以保证即使出现产品故障、程序错误的情况下， 也能确保不出现人身事故以及重大的灾难事故。
	系统设计时，请考虑触摸屏误操作和故障出现情况下的应对方案。
	在使用 GC-A2 触摸屏时，绝对不能制作和人命、重大损伤有关的开关（紧急停止开关等）
	保护接地端请务必以第三种接地方式进行接地。 否则有可能在出现故障或有漏电的时候被电击。
	请不要使用超出电源电压规格的电源供电， 这会成为引起火灾、产生故障的原因。
	请务必不要接错线，这会成为引起火灾、产生故障的原因。
	注意
	请按产品规格规定进行配置、接线，否则可能引起火灾、产生故障。 **详细内容本手册资料中有记载，特别注意点如下。**
	GC-A2 上电前，请务必确保电源电压在规格范围内。否则，可能会损坏产品。
	电线走线时请不要在电线上施加大的应力。否则可能产生感应电或引发火灾。
	请在断电状态下进行接线，否则可能被电击、引发产品故障。

【关于使用方法】

警告
通电中请不要触碰接线端子,否则会因为感应电或误动作引发事故。
触摸屏面板是由玻璃制成，请不要用重物敲打或重力按压面板，以免损坏玻璃面板。
请不要用笔和螺丝刀等顶端尖利的物品点击触摸屏，否则有可能损坏触摸屏或引起故障。
请在规格规定的范围内使用本产品，否则会引发人身事故或设备故障。
在设备运行中，在进行设定值变更操作时一定要小心，如果不小心把本该断开的输出误置位接通的话，可能会引发重大事故。 请由具备资质的人在确保人体、设备安全的情况下，进行操作。
万一 GC-A2 出现了故障，请马上切断电源送修。千万不要带伤工作。
禁止在带有可燃性或易暴性气体或蒸汽的环境下使用本产品。 否则可能引发火灾。
请不要把螺丝刀等金属类物品插入本体背面的散热缝中，否则容易短路，引发故障。
注意
请不要把异物插入本产品上的任何开口部，否则容易产生静电，引发故障。
请不要堵塞本体散热缝。否则，本体内部问题会上升，从而引发火灾或出现故障。

【关于维护保养】

注意
请不要自己分解、修理本产品。否则会引发火灾，产生静电，出现故障。
请在断电状态下对本产品进行维护保养工作。 在通电状态下进行维护保养，可能会引发电击。

【关于报废处理】

警告
报废后的 GC-A2 产品本体包含有一定数量的电子、塑料、金属、液晶等部件，这些部件可能包含有对水、土壤、大气等环境产生一定影响的物质。为了保护环境质量，请您按国家环境保护法律、法规规定以及所在地政府部门有关危险废弃物处理规定妥善处理报废部件。

关于产品使用场合

本公司产品设计为用于一般设备电子控制用途目的。请不要用于和人命直接有关的要求高信赖性的应用。 另外，当用于输送设备（列车、汽车等）的控制和安全性相关单元、交通信号机、防灾/防犯设施等场合，或产品使用的环境/使用条件和一般电子控制设备不同的时候，请事先和本公司销售部门联络确认。

关于产品的质保期和质保范围

[产品质保期]

本产品的质保期为用户购买后的一年间。

[质保范围]

在质保期内由于产品本身的质量问题或本公司的原因而引起产品故障的，本公司负责质保修理或质保调换。但是，由于以下原因而引起产品故障的，不属于本质保范围。

- 由于用户不正当的安装、使用而引起的问题；
- 故障是由于本产品以外的原因引起的；
- 用户自行拆开、改造、修理过的产品；
- 其他由于用户本人的责任引起问题的场合；
- 由于天灾、人祸及其他不可预测的原因而引起的问题。

另外，这儿所承诺的质保，是针对本公司所售出产品的。对于由此而引发的其他损害，本公司恕不承担任何责任。

手册修改履历

如果你有有关本手册的事情需要联系我们，请首先确定手册的名称和版本号。

手册名称：《SCA2 画面编辑软件 K-BASIC 参考手册》。

资料编号	编制日期	内容说明
KEW-M9549A	2019 年 3 月	初稿，根据日文版翻译编辑。
KEW-M9549B	2023 年 6 月	对应至日文版资料 2022.4 修订项
JELWX-M9549B	2024 年 7 月	公司名称更改

目　录

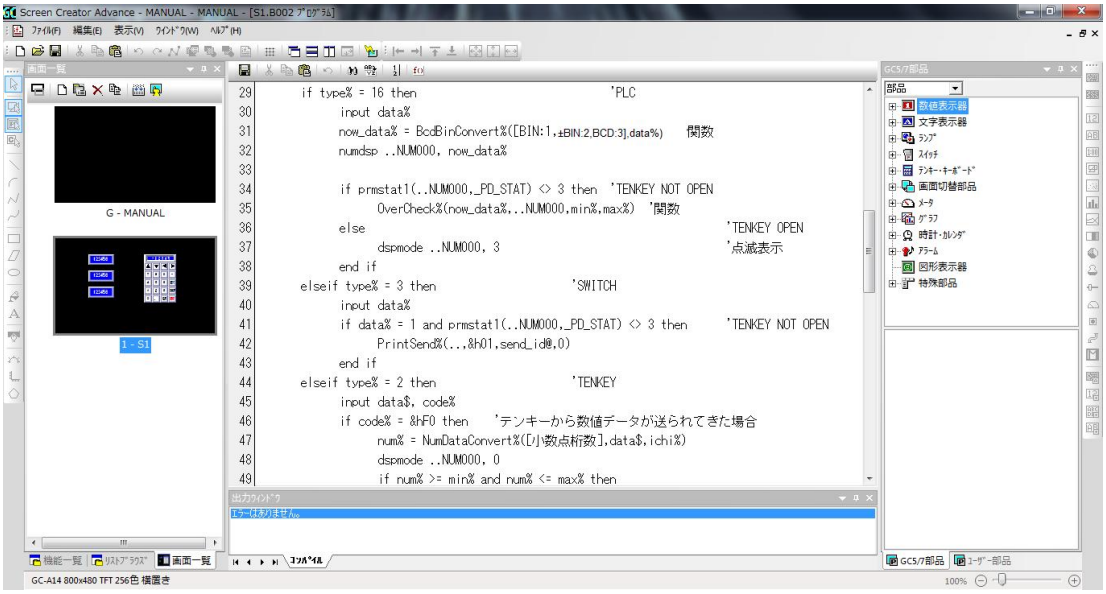
第一章 K-basic 控制程序基本

1-1.K-basic 简 介

1-1-1 什么是控制程序

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

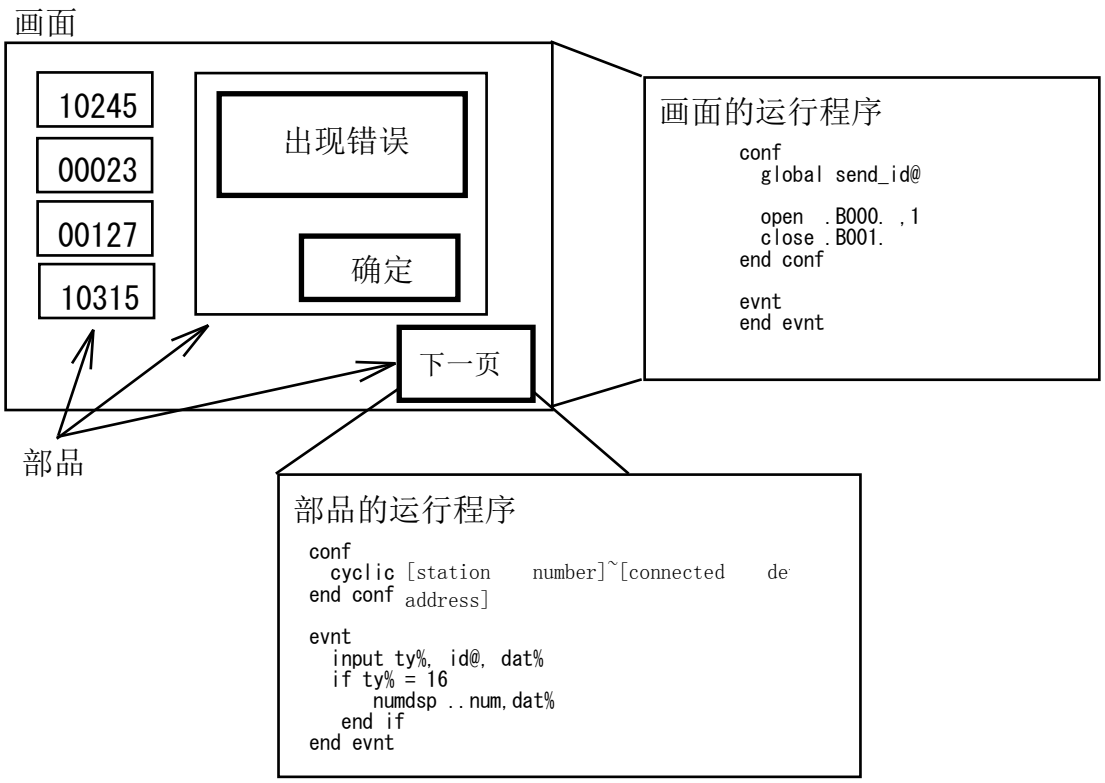
我们使用K-basic程序语言来为部品编制程序，使其完成期望的功能。



1-1-2 K-basic 语言描述的对象

控制程序分部品的控制程序和画面的控制程序 2 种。

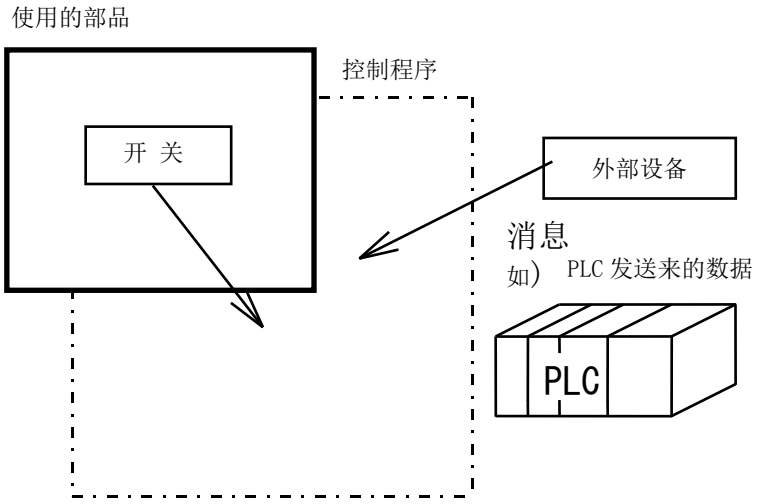
（1）部品的控制程序
我们可以为每个部品单独编制控制程序。
（2）画面的控制程序
同部品一样，你可以为每幅画面编制控制程序。



1-1-3 控制程序的触发执行

消息是程序执行的触发信号。部品在接收到发给它的消息之后才开始控制程序的执行。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。



1-2.用语说明

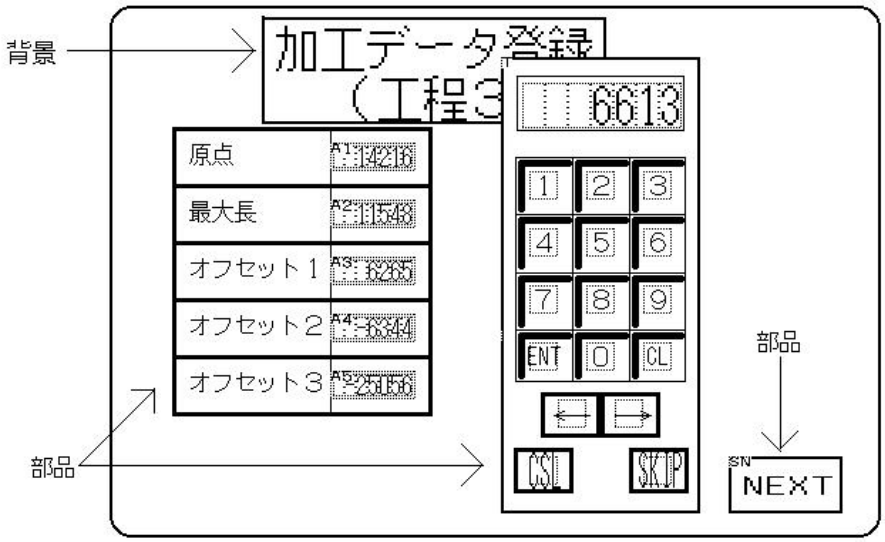
1-2-1.画面

画面是构成工程的组成单位，画面由部品、构件、背景图形、画面控制程序等组成。GC-A2 画面由两种类型的画面组成，它们是：

・全局画面（每个项目有且仅有一幅）

・局部画面

实际显示的画面是以上两种的叠加。

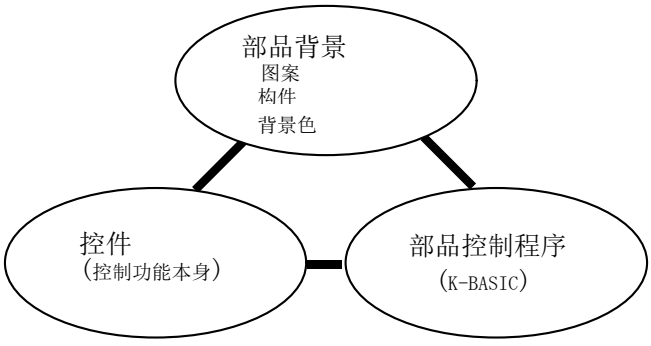


1-2-2. 构件（背景）

用于制作画面、部品以及构件的直线、四边形、圆形、文字等图案要素

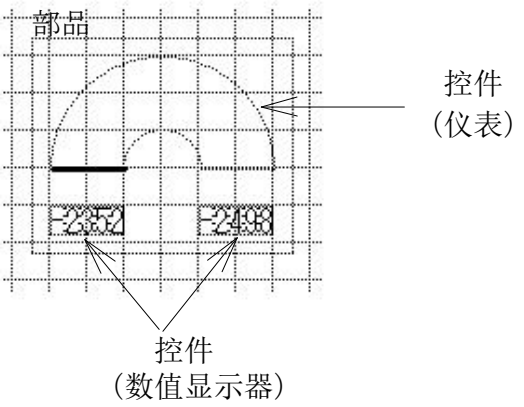
1-2-3.部品

部品是配置到画面上的最小功能部件，SCA2 提供了 13 大类共数百种部品，供用户在制作画面时直接调用。普通的画面，其功能通过直接调用部品一般都能实现。



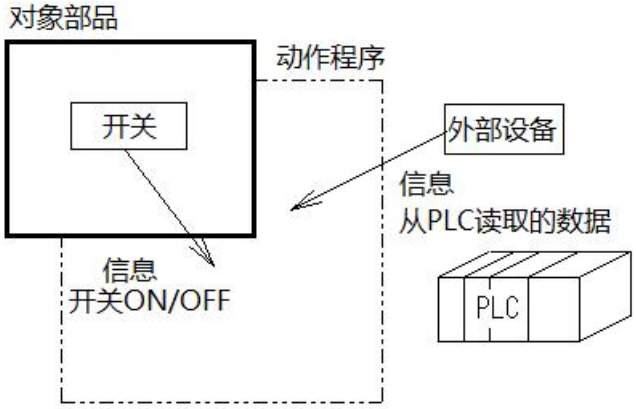
1-2-4.控件

控件是用于构成部品的最小功能单位，在同一部品中可重叠放置多个控件。系统提供 2 大类 16 种控件。控件可以直接与 PLC 功能存储器相连。因此它能直接显示 PLC 功能存储器内的数据，该功能存储器在部品属性界面中设置。



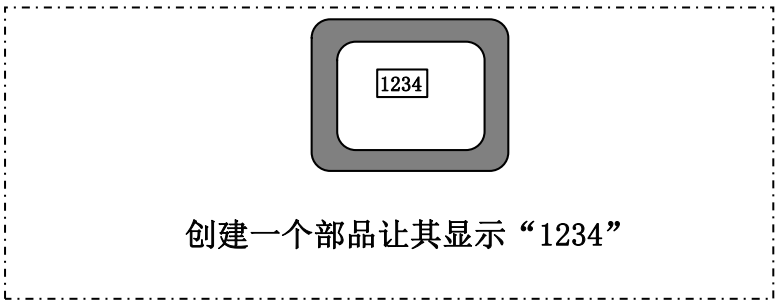
1-2-5．信息

信息一般用于作为控制动作程序的启动条件使用，信息的发生源可以是 PLC 等外部设备的开关量信息、也可以是触摸屏工程内部的其他部品数据发送过来信息。



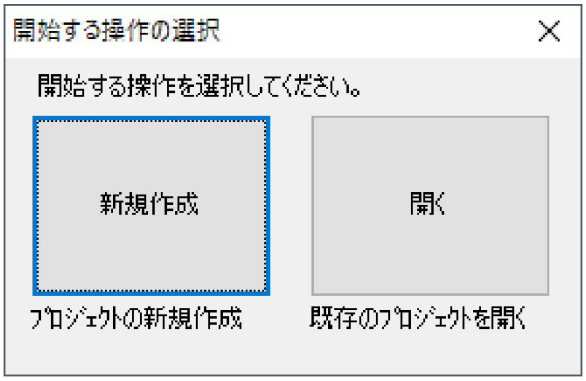
第二章. 编程实例

2-1. 创建一个显示数据的部品

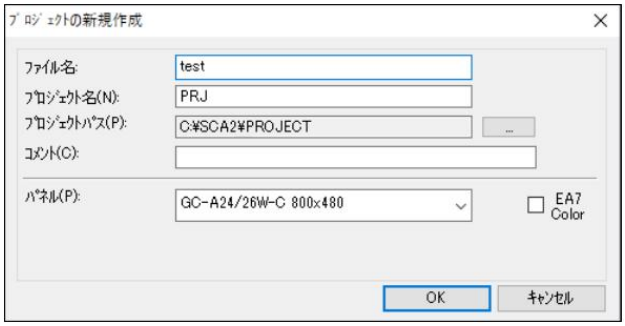


创建一个部品让其显示“1234”

制作在 GC-A2 上显示数值的部件。
在屏幕上创建一个项目。
启动屏幕创建器 2，选择“新建”来创建项目。

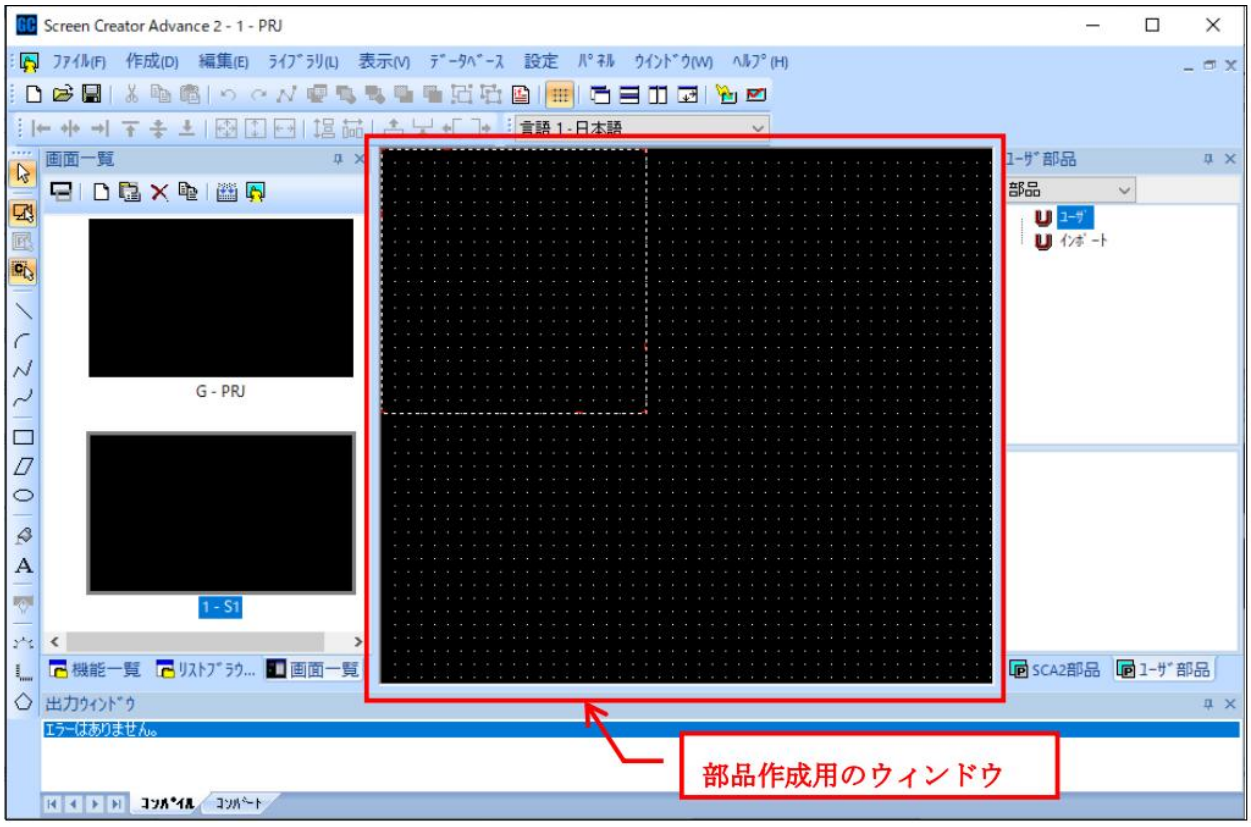


通过以上操作打开“新建项目属性”对话框。
在“项目名”中选择“test”，在“面板”中选择您所使用的模型，
按下“OK” 按钮。



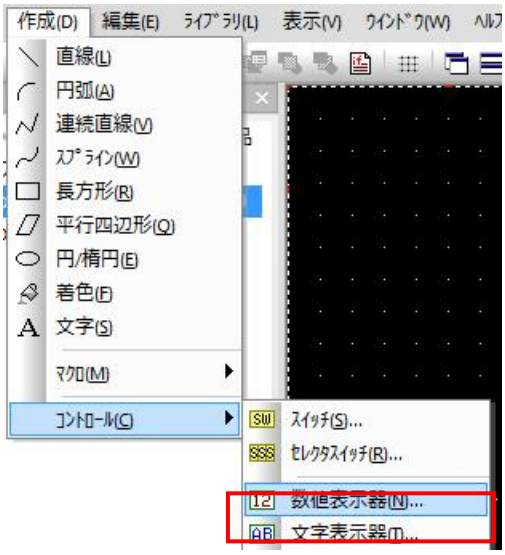
通过以上操作打开“新建项目属性”对话框。
接下来，你可以从屏幕创造者 advance 2 的菜单中找到“库”
选择“制作”中的“零件”。这个打开了制作零件的窗口。


ﾗｲﾌﾞﾗﾘ(L)
表示(V)
ﾃﾞｰﾀﾍﾞｰｽ
設定
ﾊﾟﾈﾙ
ｳｲﾝﾄﾞ
作成(N)
閉じる(C)
保存(S)...
ﾗｲﾌﾞﾗﾘのﾌﾟﾛﾊﾟﾃｨ(P)...
画面から部品を追加(V)...
外部からﾌｧｲﾙを追加(A)
部品(P)
ﾃｸｽﾁｬ(T)


Screen Creator Advance 2 - 1 - PRJ
ﾌｧｲﾙ(F)
作成(D)
編集(E)
ﾗｲﾌﾞﾗﾘ(L)
表示(V)
ﾃﾞｰﾀﾍﾞｰｽ
設定
ﾊﾟﾈﾙ
ｳｲﾝﾄﾞｳ(W)
ﾍﾙﾌﾟ(H)
言語 1 - 日本語
画面一覧
G - PRJ
1 - S1
ﾕｰｻﾞ部品
部品
ﾕｰｻﾞ
ｲﾝﾎﾟｰﾄ
機能一覧
ﾘｽﾄﾌﾞﾗｳ...
画面一覧
SCA2部品
ﾕｰｻﾞ部品
出力ｳｨﾝﾄﾞｳ
ｴﾗｰはありません。
ｺﾝﾊﾟｲﾙ
ｺﾝﾊﾟｰﾄ
部品作成用のウィンドウ

控件的放置

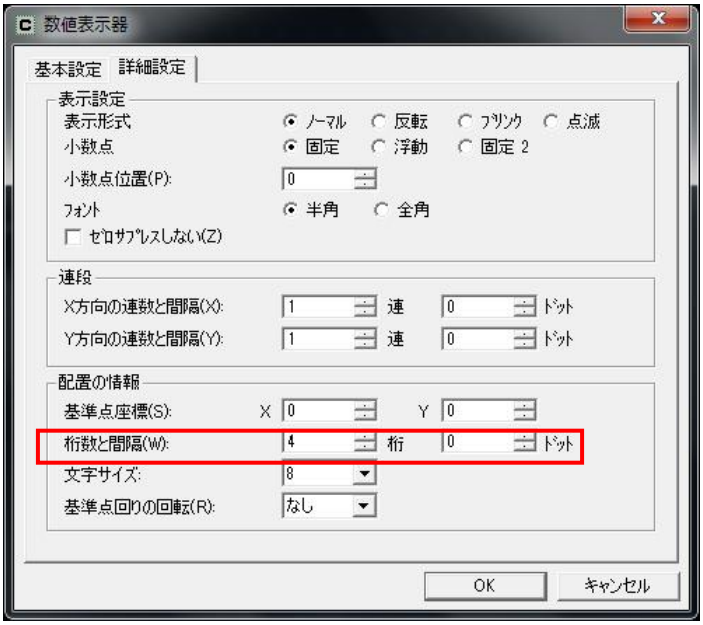
首先选择数字显示控件



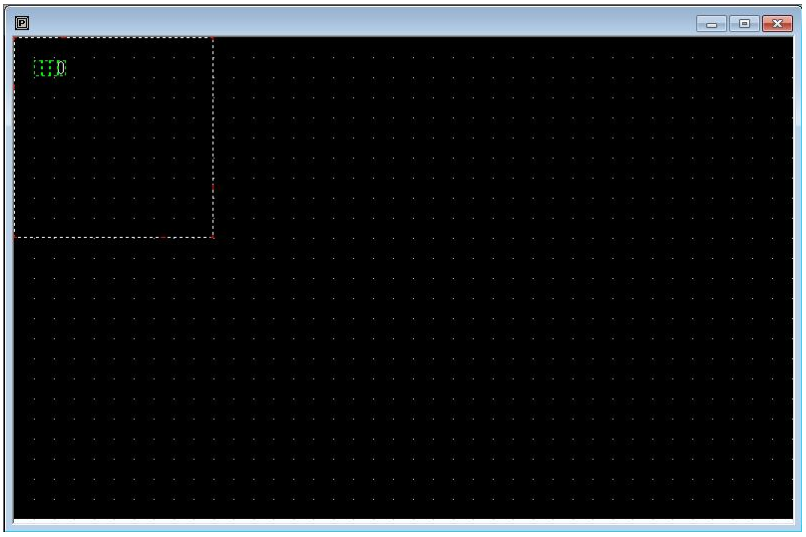
数据显示器属性设置对话框显示如下：

点击“详细设定”页，显示本控件物理属性：

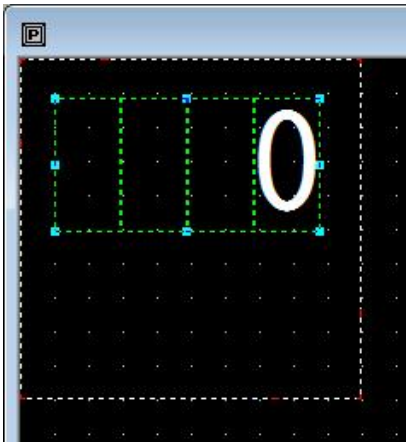
参照下图设定“间隔”与“位数”，并点击“OK”按钮完成设置。



将控件放在界面合适的某地方



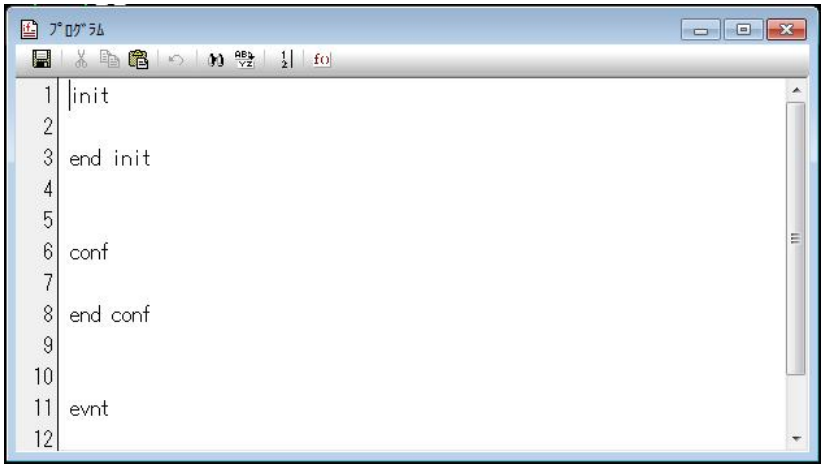
用鼠标拉住控件的右下角出现缩放框，可将其调整到适当的大小。



2-1-2.编程

然后给正在制作的零件输入程序。从菜单的“编辑”选择“零件程序的编辑” 。

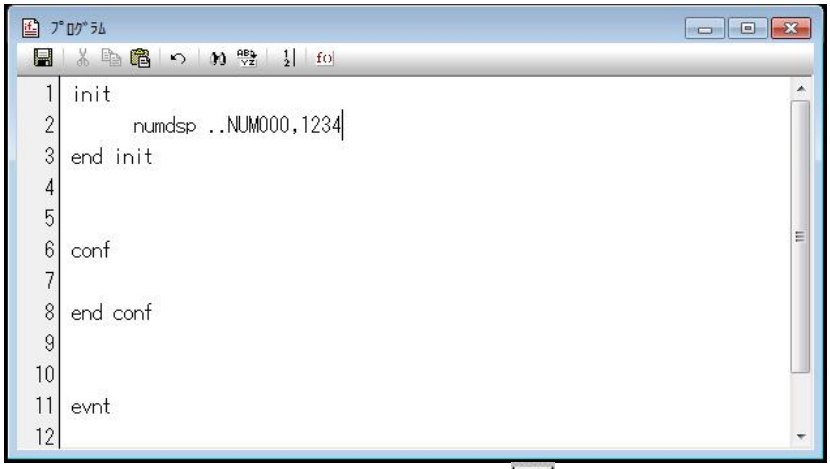
要为部品编制程序，可以选择菜单中的“Edit”，然后选择“Edit Part Programs”，或者直接点击部品右键菜单中的“Edit Part Programs”，会弹出编程界面如下：



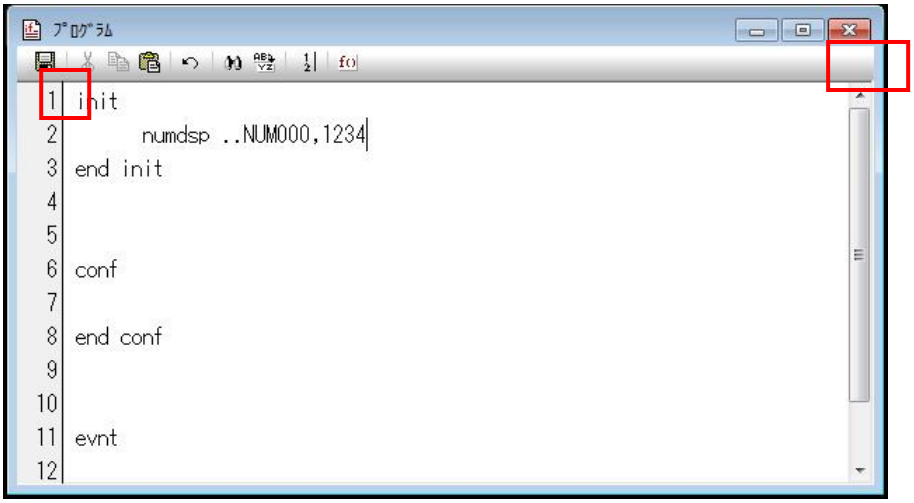
根据要求我们编制程序如下：

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

conf
end conf

evnt
end evnt
```

程序的内容将在后续部分详细介绍！目前先按照上文输入程序：



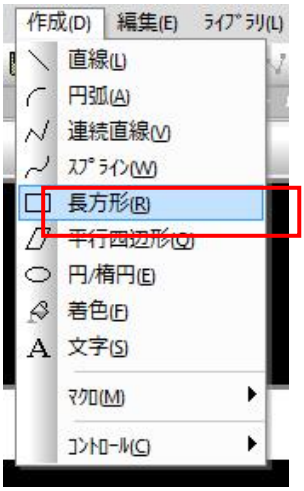
选择“Program”菜单中的“Save”或直接点击  ，保存上述程序并关闭编程界面。



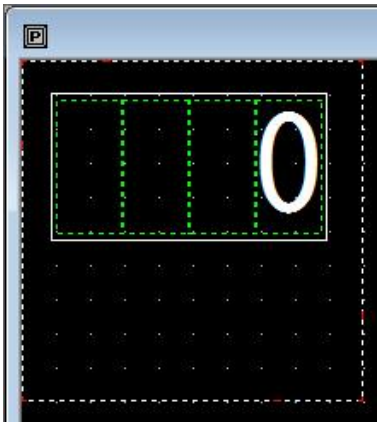
之后我们就回到了部品编辑界面。

2-1-3. 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“Create” 菜单,下拉选择“Rectancle”。当然，在画框之前你可以选择其属性，诸如颜色、线型等。



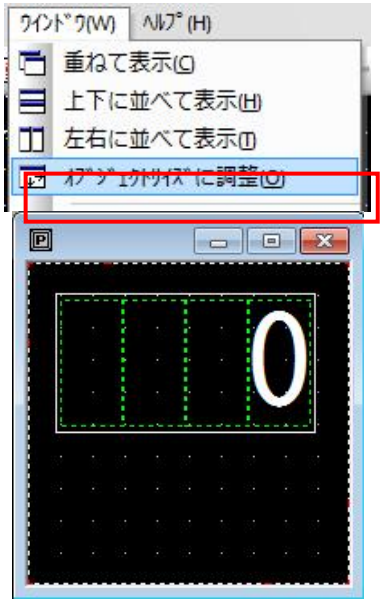
然后用鼠标左键点击数据显示控件的左上角，同时按住左键不放松，沿着显示控件的对角线拖动鼠标，同时会出现一个矩形框，该框的大小随着鼠标拖动的位置改变。当大小合适时放开鼠标左键。



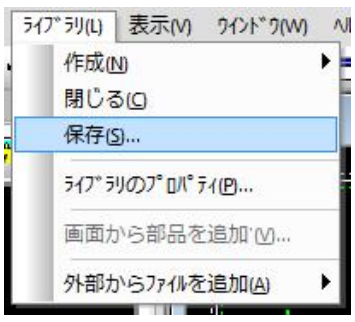
在画完后，鼠标还处于画框模式状态，这时只要点击鼠标右键即可取消。

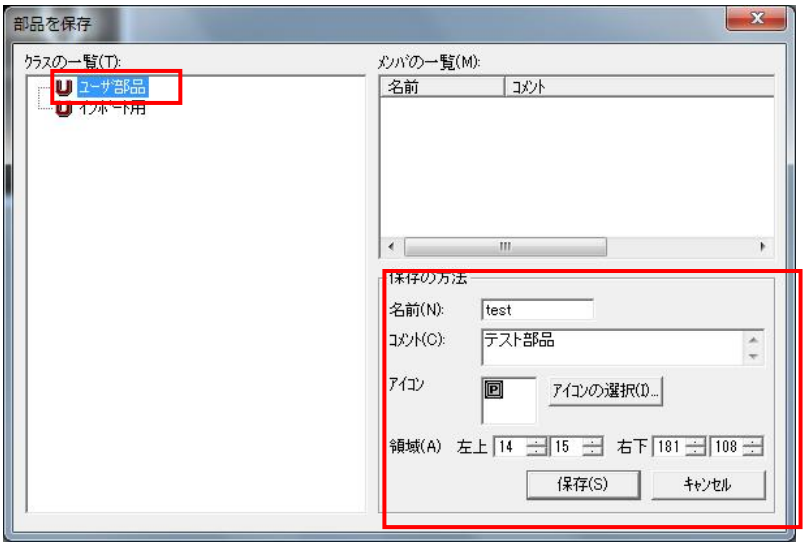
2-1-4. 部品的保存

部品的物理大小是由红白相间虚框的大小决定的，所以为了制作大小适当的部品，应将紧贴界面边框的虚框缩小，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！可以用以下工具调整大小



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。这时出现如下对话界面：可按照下图输入所需信息。



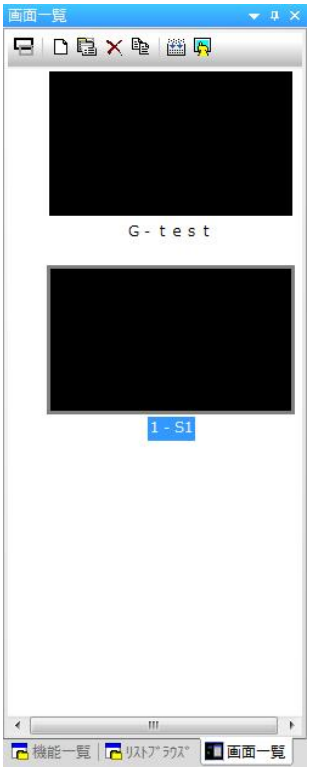


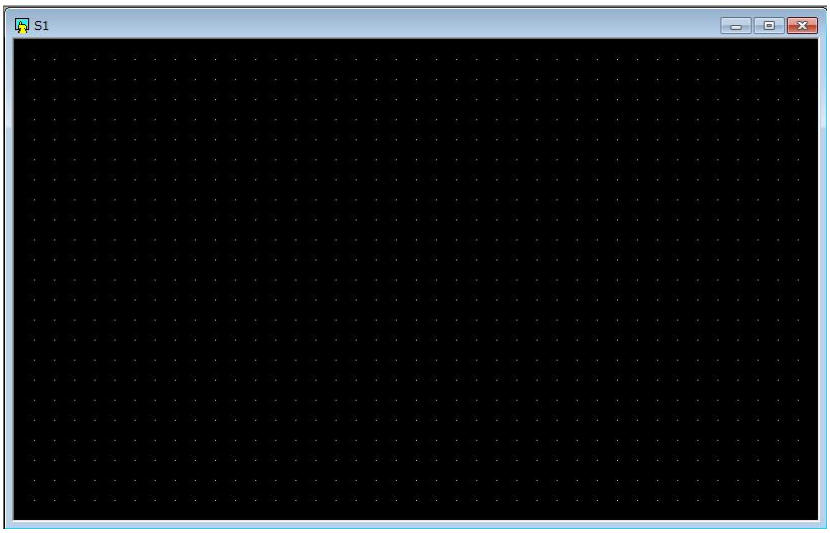
我们将部品保存为用户部品“User parts”，部品名称保存为“test”，在“Comment（注释）”中输入“test part”，点击“save”保存部品。保存后，点击部品界面右上角关闭 按钮，将部品界面关闭。这样，一个部品就做好了！

2-1-5. 创建部品的调用

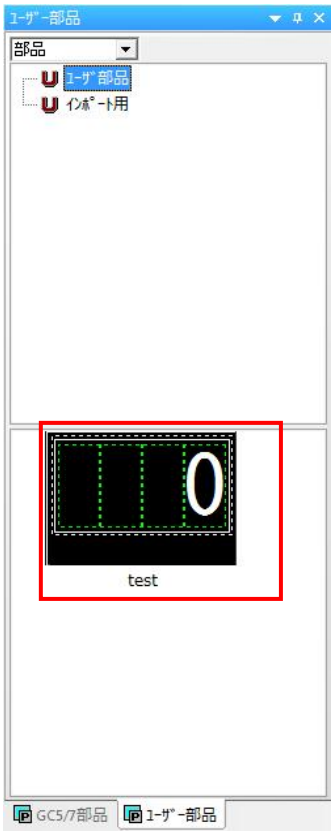
同其它部品的使用一样，在画面上直接调用就行。

首先在画面一览中新建 S1 画面

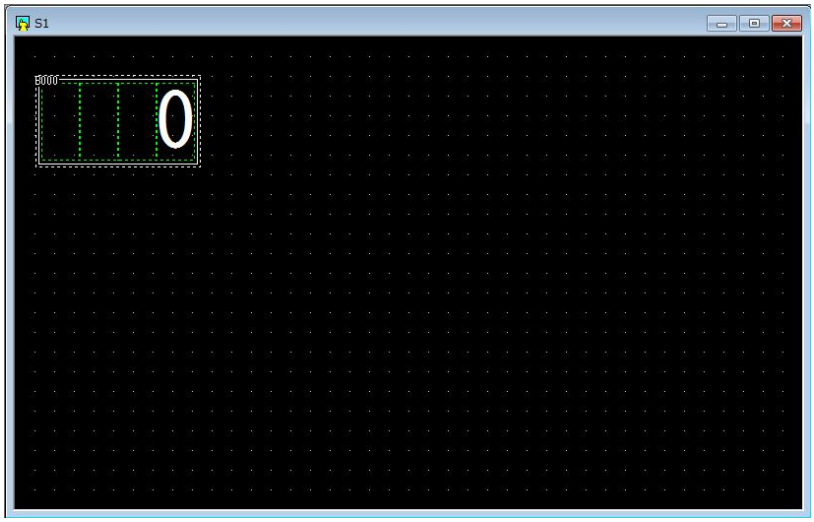




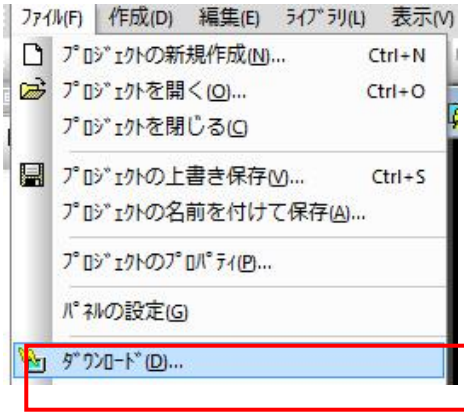
之后将上一节中制作的用户部品放入画面：



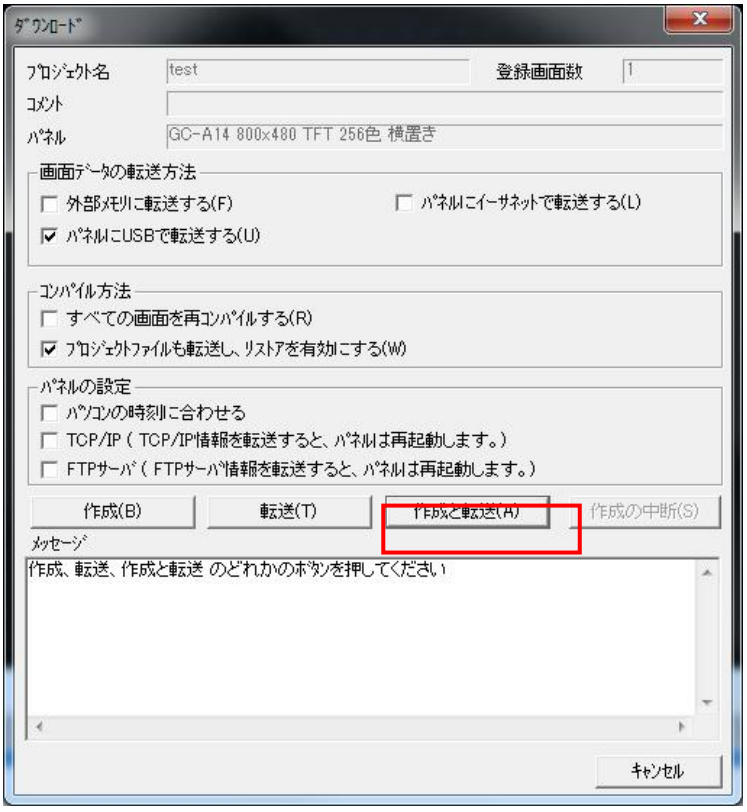
效果见下图：



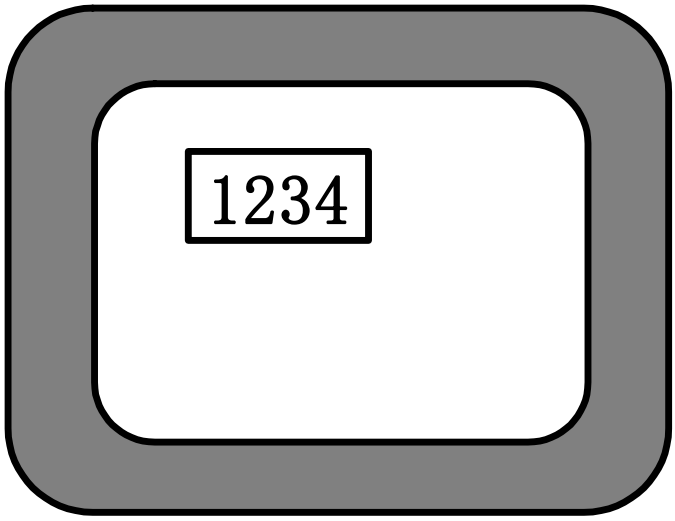
至此，当前工程的作图工作就完成了。现在我们可以尝试把作成的工程下载到触摸屏里去查看一下实际效果。点选“文件”菜单下的“下载”功能。



点击“构建并传送”。



数据传送正常完成后，触摸屏显示用户画面。如下图所示：



如果数据构建过程中发生报错请参照报错信息检查作图的过程是否有误，如传送过程中产生报错则请检查 USB 电缆的连接状态以及下载选项是否正确。

2-1-6.程序解释

刚才使用的程序如下所示

程序：

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

conf
end conf

evnt
end evnt
```

程序解释如下：

① init～end init

这个部分被称为初始化块，在这个部件的程序中最先执行会的。因此一般用于变量的声明和它们的初始化。

② numdsp ..NUM000,1234

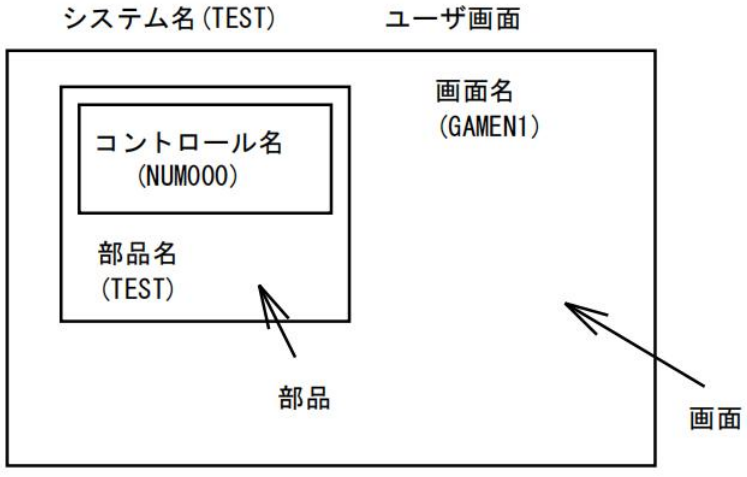
上述语句的功能是将指定的数据显示到指定的数据显示控件。numdsp 命令，是在某个数据显示控件里显示某个数值。..NUM000 用于指出数据显示控件的名称，命名规则如下：

● 控件名及命名规则

表示画面时	GAMEN..
表示 GAMEN 中的零件时	GAMEN. test。
表示在 GAMEN 中的 BUHIN 中的控件时	GAMEN. test . num 000
表示自画像的自己零件的场合(省略)	..
表示自画像、自己零件内的控制的场合	..num000

注意)名字一定要用英文和数字来记述。

像这次的零件一样，控制的地方和写有这个程序的零件一样的情况(在自己零件内的控制时)画面名，零件名可以省略。



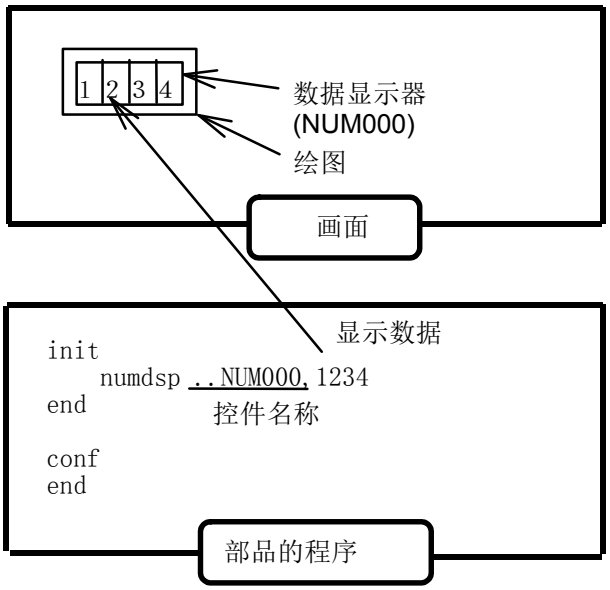
部件(TEST)内的程序控制(NUM000)

指的是。

gamen.test.num 0

或者..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

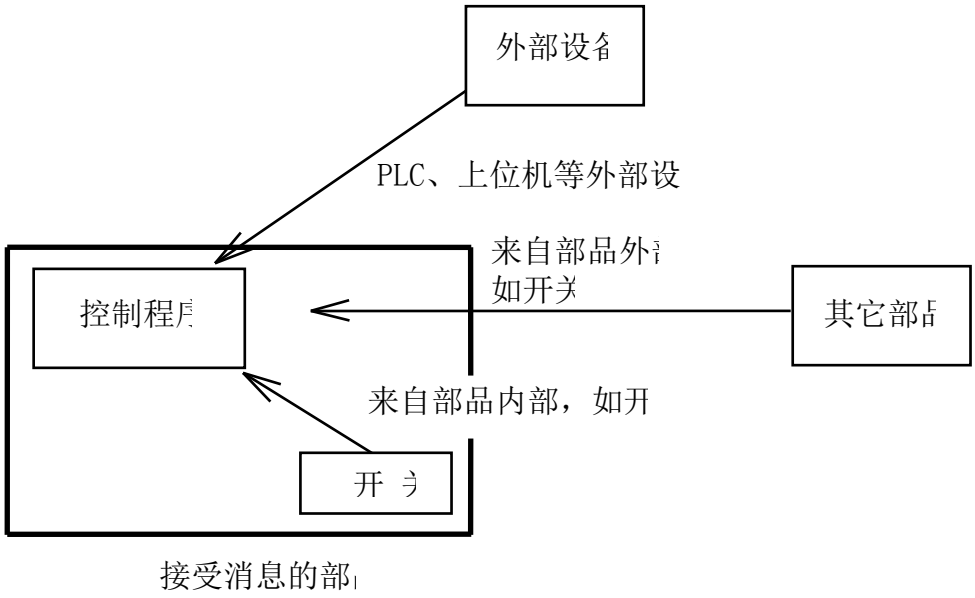
NUM000 控件是本部品里的唯一控件。在触摸屏上显示“1234”

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。

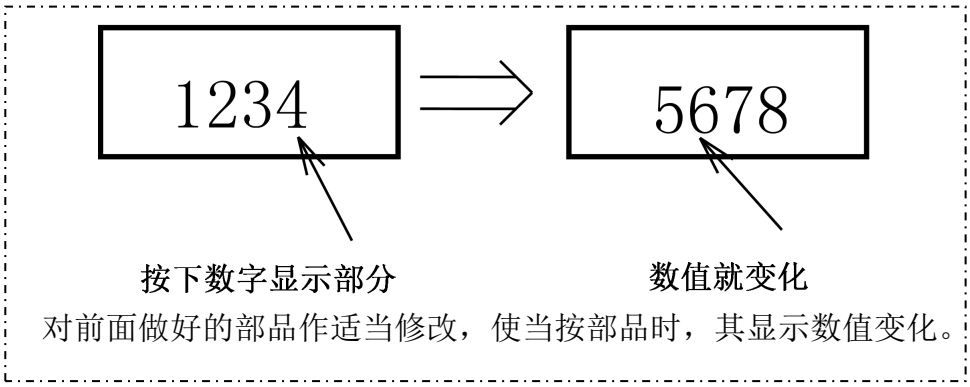
④ Evnt～end evnt

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。部品的消息来自所连接的 PLC 功能存储器或部品等等，如下图所示：



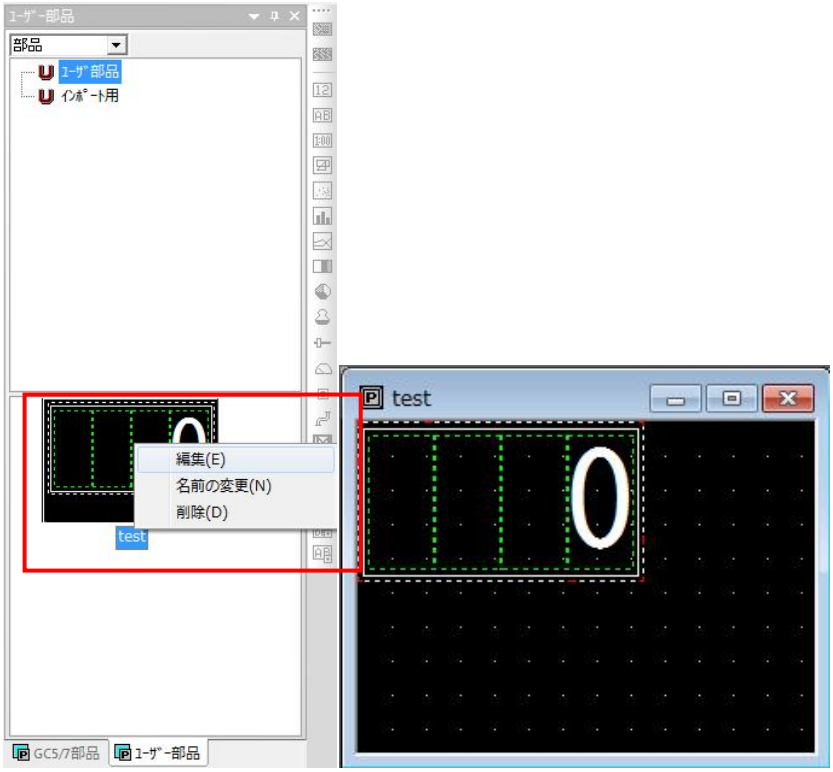
消息的发送

2-1-7.部品的修改

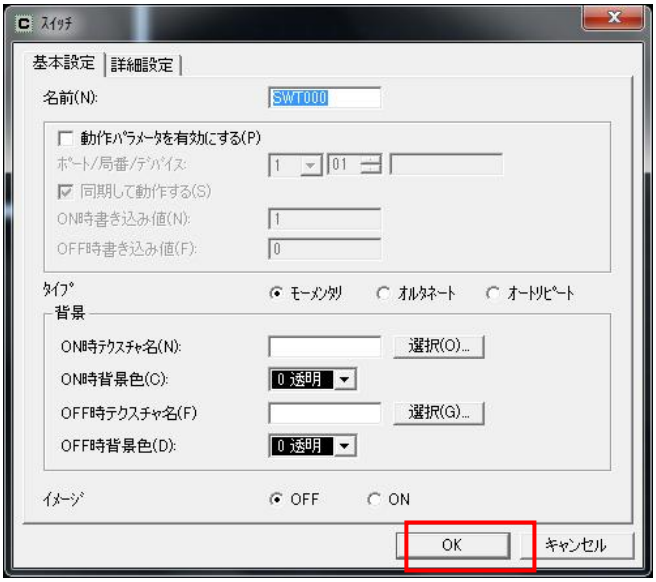


对前面做好的部品作适当修改，使当按部品时，其显示数值变化。

首先，要打开前面做好的部品“Test”。方法是：选择右侧部品列表中“User Parts”下的自定义部品“Test”。

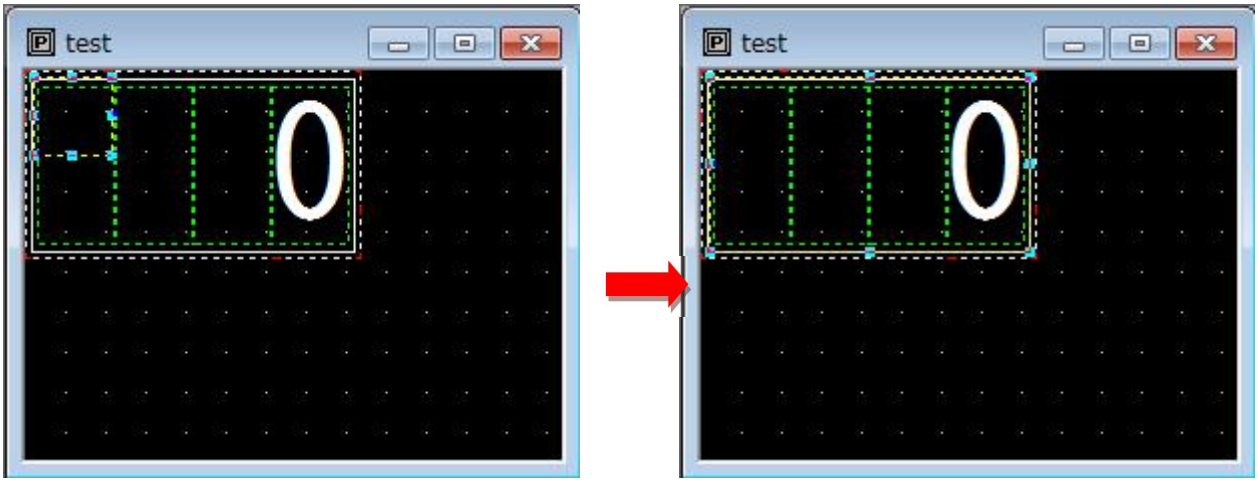


本例中，为了使用触摸面板，要在部品里添加一个开关。通过菜单或通过快捷方式选择“Control”-“Switch”（“控件”—“开关”）。弹出开关控件属性对话框，点击“OK”，



光标变成小鼠形状。点击部品编辑界面，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界面，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
```

追加

追加

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。
我们主要在 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。目前系统共可提供 1~22 共 22 种消息，更详细的情况，请参考《SC5 用户

手册》。

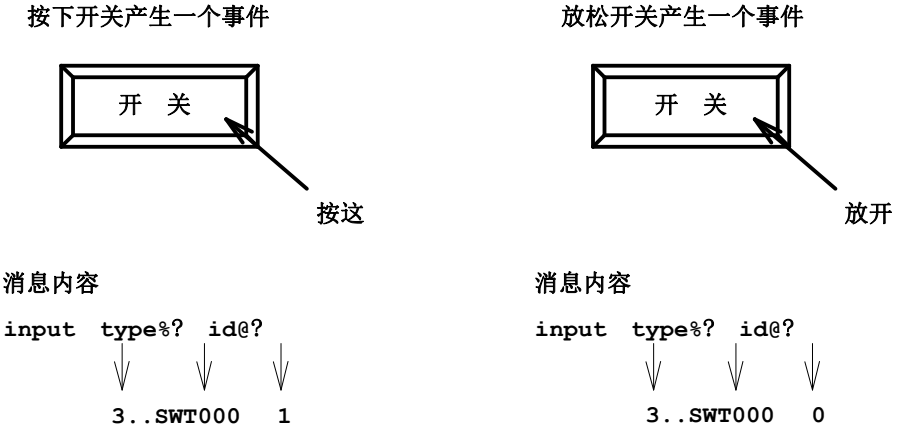
id@：消息发送者身份（ID），如果消息来自开关，则它就是开关控件的名称。ID 的格式是：画面名称（Screen name）、部品名称（Part name）、控件名称（Control name），中间分别用句号（.）隔开。
如：Scr1.test.NUM000
这个 ID 称为 ID 型常量，它是 K-basic 独有的，在处理 ID 时，你可以在其后面加“@”符号，如“ID @”。
注意：消息包括类型（Type），身份代号（ID），和数据（Data）三部分。

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。



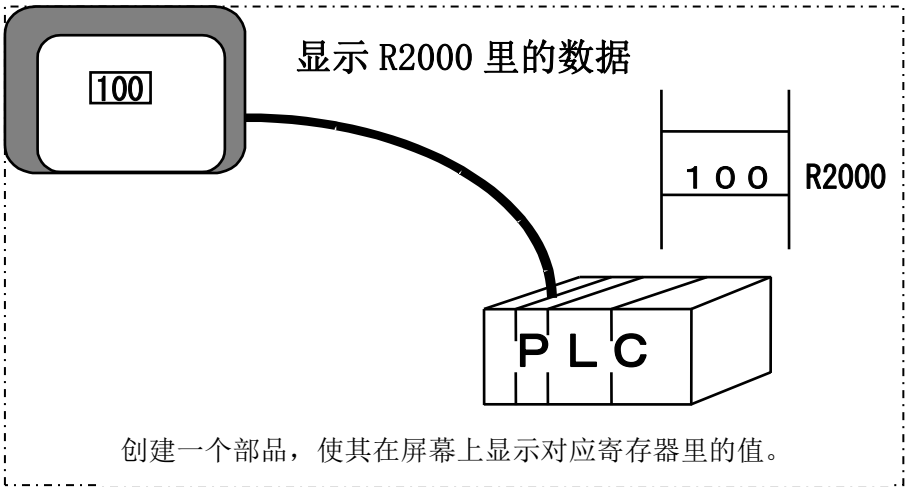
注意：如果开关设置为“momentary”，则开关按下一次将产生两条消息，即按下和放开。

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！

2-2.创建一个与 PLC 设备连接的部品

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

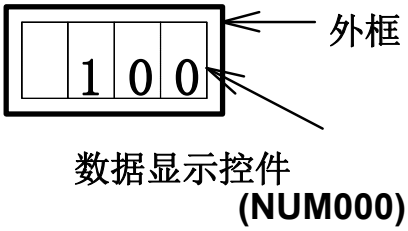
2-2-1.数据显示



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。

程序的流程是：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”功能便可以使这个问题变得非常简单。

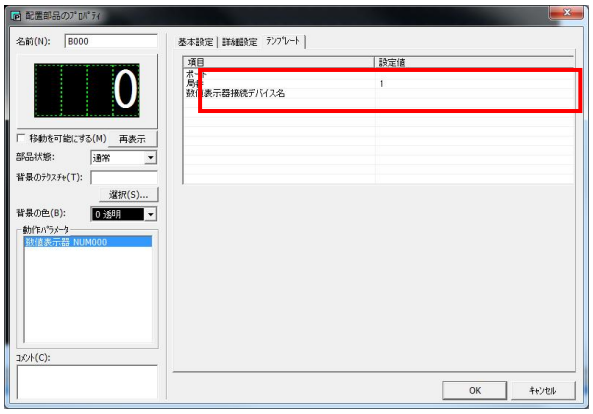
```
Cyclic [局号]~[寄存器号]
```

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。

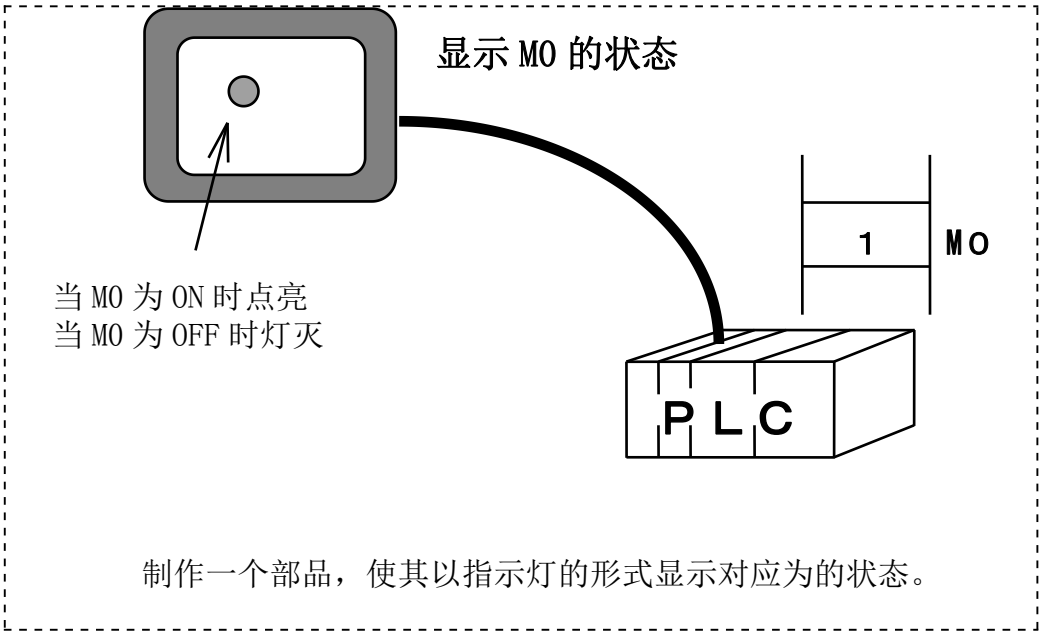
使用模板（template）的程序实例如下：

```
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）。



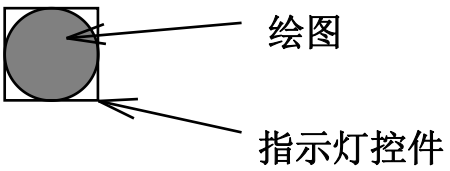
2-2-2.指示灯



使用的控件：

指示灯控件（LAM000）

外形：



注意：指示灯由 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！

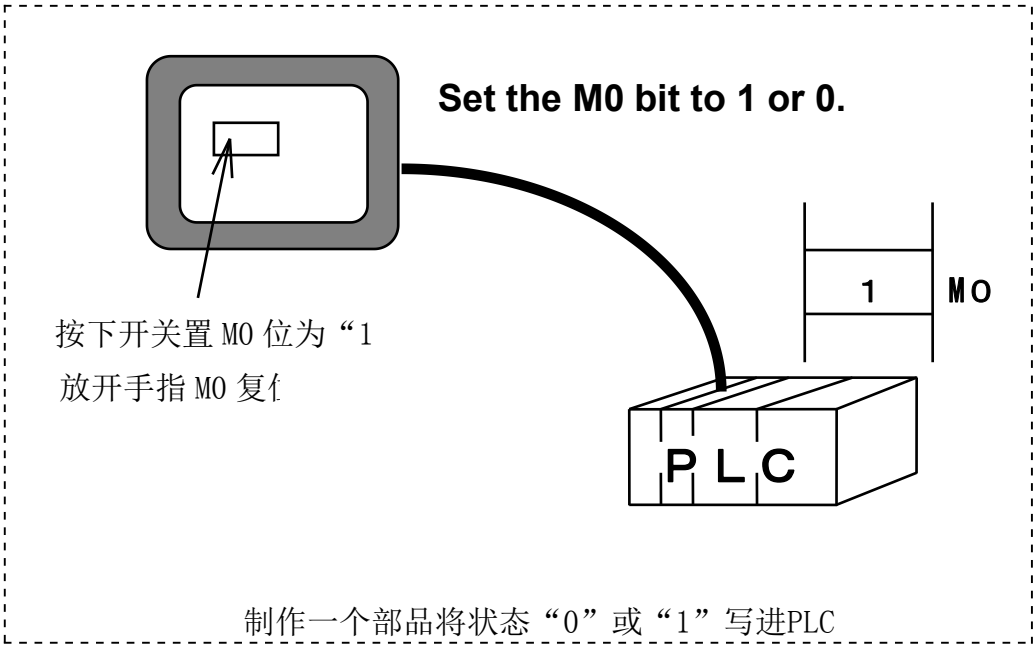
● Event Block

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。

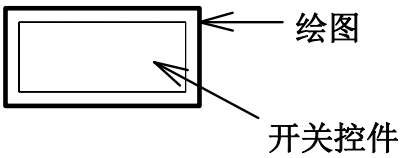
2-2-3.开关



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%
```

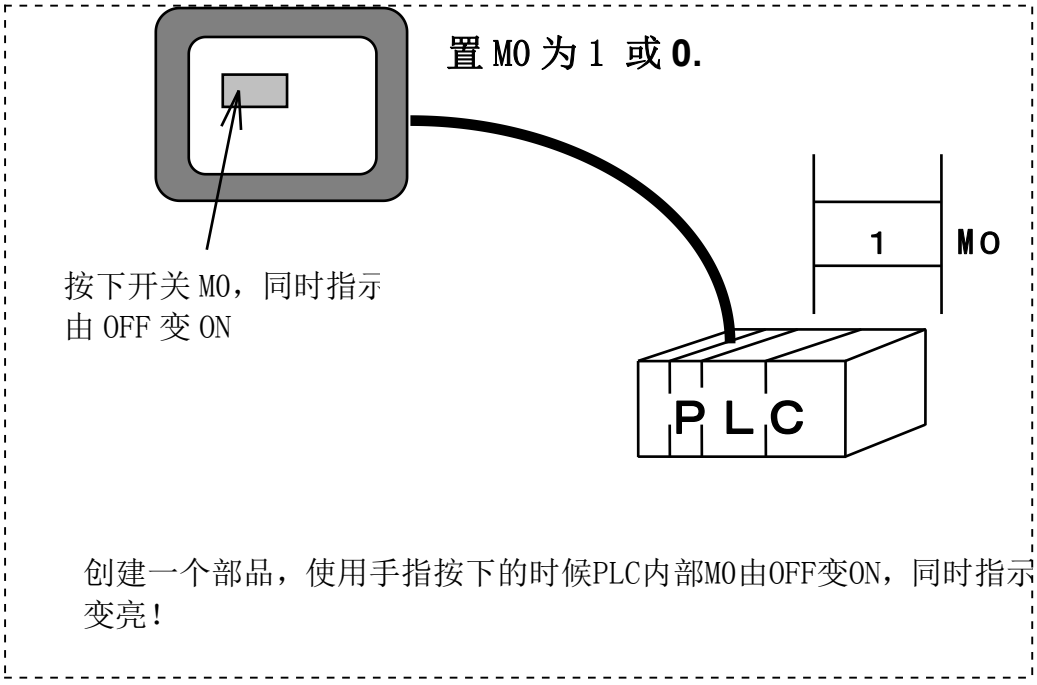
input 指令读取来自开关的消息，读取顺序为“type%=3”、“id@=...SWT000.”、 data%” =ON/OFF (1 or 0)

```
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”。

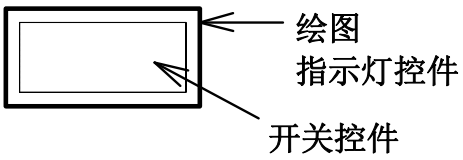
2-2-4.带指示灯的开关



创建一个部品，使用手指按下的时候PLC内部M0由OFF变ON，同时指示变亮！

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
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]=1
    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%

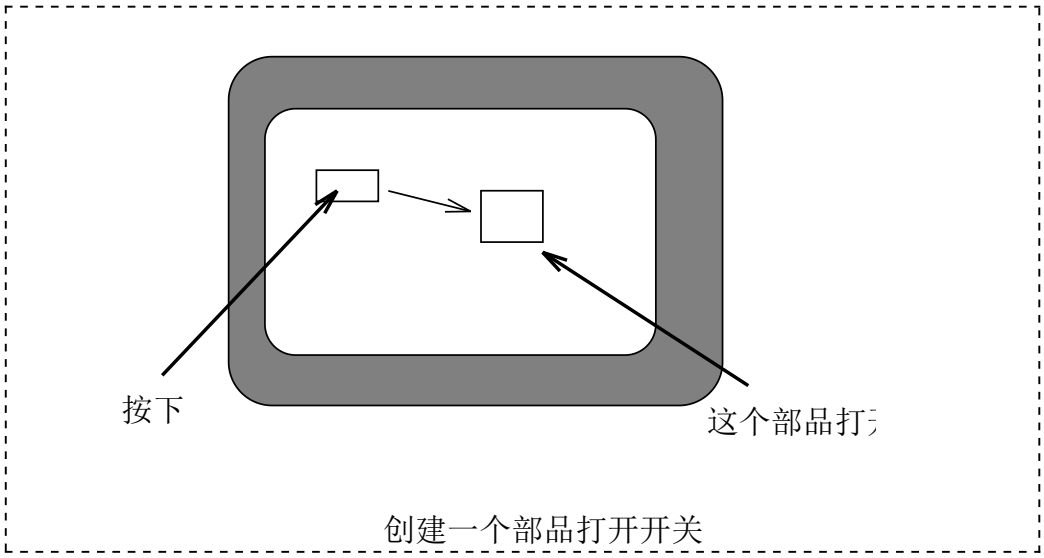
  同前面讲述的一样。

```
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的消息将决定指示灯的ON/OFF状态。

2-3.制作一个部品来控制其它的部品

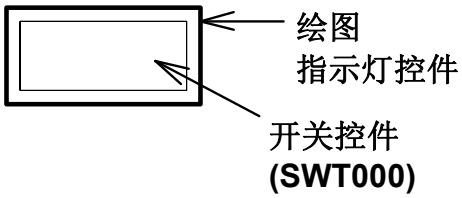
2-3-1.从画面上调用其它部品



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
```

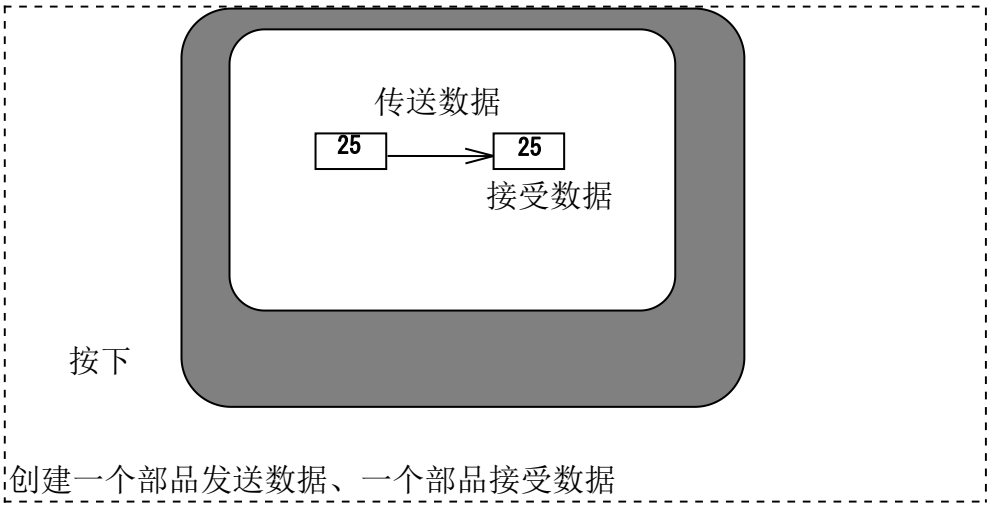
● Initialization Block
除了声明局部变量之外，没作任何处理。

● Configuration Block
input type%,id@,data%
“input”指令从开关控件读取信息。

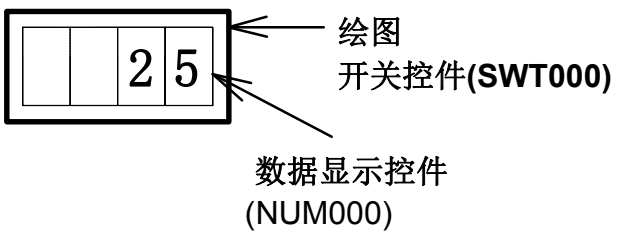
```
if type%=3 and id@=..SWT000 and data%=1 then
      : : : : : : : : : : : : :
end if
```
“if”与“end if”之间的内容，只用在开关被按下时才执行。

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。

2-3-2.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

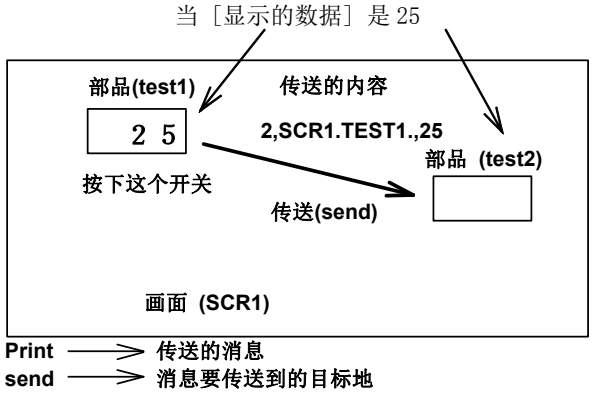
这时，“input”指令要读取3条数据消息，如下：

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]），需要注意的是：“print”和“send”这两个指令需要组合使用。



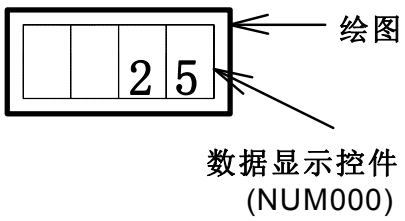
程序到此结束。当开关按下时，程序将包括参数[要显示地数据]、[目标部品名称]在内的消息发送出去。

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!

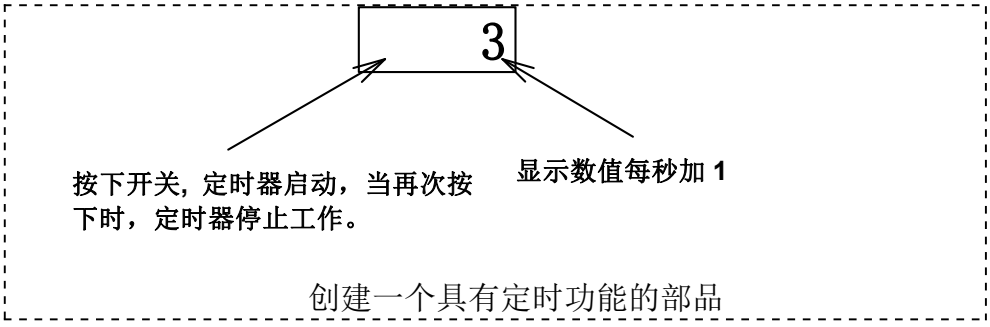
- Event Block
  input type%,id@,data%
  “input”指令前面已讲过，是从开关控件里读取消息。

  ```
  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!

2-4.创建一个使用定时器的部品

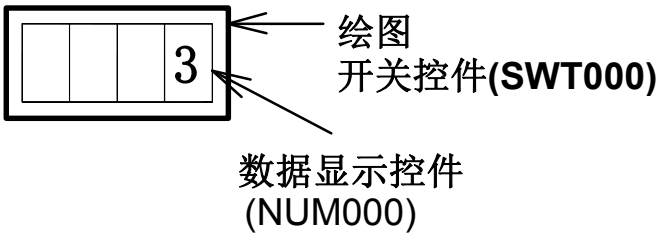


创建一个具有定时功能的部品

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 fthen”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：一个工程中最多可以同时使用16个定时器，其编号为0~15。不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
```

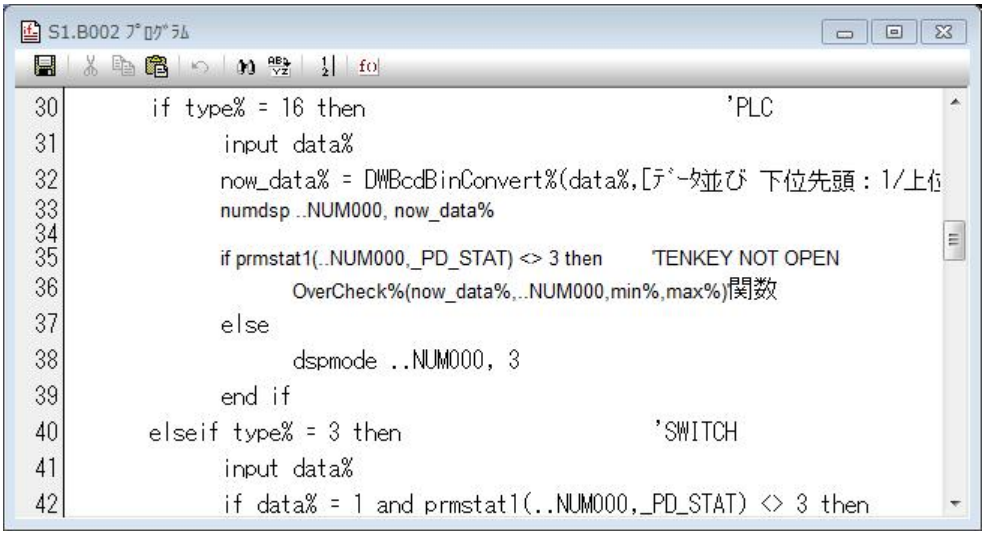
每当定时器发送一次消息，数据显示器单元显示的数值加“1”。

程序到此结束!

2-5.为画面上的部品编程

如果有需要，你可以给任何部品添加控制程序，或修改其原有的控制程序。
进入画面上的部品程序编辑状态的方法是：单击点选部品 → 右键弹出菜单 → 选择“program”进入编程界面。

打开程序编辑界面



此时，你就可以添加或者修改控制程序了。

第三章.编程规则

3-1．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）、半角日文字符（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；对于日文字符，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。

3-2．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
```

&, &O, &H

& 和 &O 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 &O10（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。

```
conf
  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
```

用来分隔表示PLC设备的通讯口号、局号与地址号。
1：1~R2000　　port1、局号1、地址号R2000

3-3. 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。
- 对于Texture(构件) 和Text(文本), 其已注册的设备名称可以作为常量。
- PLC设备如 01~R2000 and 01~M10, 等等。

3-4．常数的声明

在SCA2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，可利用变量对对特定的常数值进行批量修改，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。

3-5．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。

3-5-1.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。

3-5-2.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。

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面程序中使用。

虽然可以使用DIM来声明局部变量，但在SCA2里，应尽量使用LOCAL来定义部变量。但使用DIM可以使 SCA2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（即什也没有）。自动变量的定义方式如下：

AUTO VAR%

3-5-3.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，这些处理和解释是有一些默认规则的，所以如果在编写程序时不了解这些规则，程序可能出现执行动作不合期望的情况。下面将说明一下这些需要了解的默认规则：

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很多时候都是不希望发生的。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，而编程者有时会忽略。这类错误很难发现，即使是程序在编辑时也不容易察觉。

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SCA2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》。

3-5-4.变量初始化

在SCA2 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。

3-6.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“＝”。

3-7. 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。

3-8. 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
```

3-9. 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”。

3-10. 用户自定义函数

SCA2支持用户自定义函数，参数通过调用体输入，然后返回一个值给调用体。

3-10-1.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
```

3-10-2.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SCA2控制下的函数库里。可以被任何程序调用。

3-10-3.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”。

3-10-4.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。

3-11. 程序运行

当 SCA2 程序里指定的部品或画面收到消息后，触摸屏程序开始运行。消息的种类如下：

- 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。

- 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”指令。

- 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”指令

注意：同时接受到复数信息时按时序处理触发程序动作（队列处理），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。

3-12. 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.表示发送者身份的数值　　（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）

- **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**
  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

- **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！

3-13. 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。
  - 作图过程在初始化、Conf 块执行后才会开始，请不要在初始化块、Conf 块使用作图 相关的指令。
- 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。

3-14. 设备和通信

在 K-Basic 程序中，Port 号与局号之间用“：”、局号和设备名之间用“~”连接，如

1：VAR%=01~R2000:表示读取Port为1、局号为01、设备为寄存器R2000里的数据。
1：01~R2000=40:将40写入Port号为1、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，SCA2 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。

3-15. 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。

3-15-1.书写形式

1：00~MTBL(0)：　内部存储器表第0个单元。
1：00~MTBL(2047)：　内部存储器表第2047个单元。
1：00~MTBL(NO%)：以变量 NO% 表示的内部单元。

3-15-2.对某个单元进行读写

- ABC=1：00~MTBL (100)
  将第100个内部单元里的值读进并赋给变量ABC。

- 1：00~MTBL (200) = 23
  将常数23写入内部第200个单元。

- 1：00~MTBL (ABC) = XYZ
  将变量 XYZ的内容写入以变量ABC指出的内部单元里。

3-15-3.同时对多个单元进行读写

- BREAD 1：00~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 1：00~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 1：00~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 1：00~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。

3-16.外部文件的指定方法

当需要控制外部文件时：CG-A2 的外部存储器（USB 存储设备、SD 卡）
驱动器名称规则如下：

盘符	内容
E	SD 卡。
G	USB 存储设备。

目录的指定例：E: ￥ABC, G:/ABC/DEF

文件名例：ABCDE.DOC（包括文件名（最多 8 个字符）和扩展名（3 字符））。

3-17. 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。

第四章.指令详解篇

4-1. 指令检索（按功能检索）

○：可用、×：不可用

指令分类	助记符	型号/部品支持		
		GC5/7	SCA	SCA2
控制指令	CONF…END CONF	○	×	○
	EVNT…END EVNT	○	×	○
	FOR…TO…NEXT	○	×	○
	GOSUB	○	×	○
	GOTO	○	×	○
	IF…THEN…ELSE	○	×	○
	INIT…END INIT	○	×	○
	RETURN	○	×	○
	SELECT CASE…END SELECT	○	×	○
	STOP	○	×	○
	WHILE…WEND	○	×	○
变量声明	AUTO	○	×	○
	BACKUP	○	×	○
	CONST	○	×	○
	DIM	○	×	○
	GLOBAL	○	×	○
	LOCAL	○	×	○
	STATIC	○	×	○
	STRING	○	×	○
消息处理	INPUT	○	×	○
	PRINT	○	×	○
	RUN	○	×	○
	SEND	○	×	○
算数运算	ABS	○	×	○
	ATN	○	×	○
	BITSET	○	×	○
	BITTEST	○	×	○
	CINT	○	×	○
	COS	○	×	○
	EXP	○	×	○
	INT	○	×	○

指令分类	助记符	型号/部品支持		
		GC5/7	SCA	SCA2
算数运算	LOG	○	×	○
	SHIFT	○	×	○
	SIN	○	×	○
	SQR	○	×	○
	TAN	○	×	○
字符串运算	ASC	○	×	○
	CHR$	○	×	○
	CVB	○	×	○
	CVF	○	×	○
	CVI	○	×	○
	CVID	○	×	○
	CVW	○	×	○
	HEX$	○	×	○
	INSTR	○	×	○
	LEFT$	○	×	○
	LEN	○	×	○
	MID$(函数)	○	×	○
	MID$(指令)	○	×	○
	MKB	○	×	○
	MKF	○	×	○
	MKI	○	×	○
	MKID	○	×	○
	MKS	○	×	○
	MKW	○	×	○
	OCT$	○	×	○
	RIGHT$	○	×	○
	STR$	○	×	○
	VAL/VAL2	○	×	○
数值类型转换	BCD2BIN	○	×	○
	BIN2BCD	○	×	○
	GETGID	○	×	○
	GETGNO	○	×	○

指令分类	助记符	型号/部品支持		
		GC5/7	SCA	SCA2
数值类型转换	GETID	○	×	×
	GETOFFSET	○	×	×
	TIMID	○	×	×
	TIMINT	○	×	×
画面/部品控制	CLOSE	○	×	×
	JUMP	○	×	○
	MOVE	○	×	×
	OPEN	○	×	×
	PMODE	○	×	×
	PREVJUMP	○	×	○
	PSTAT	○	×	×
开关控制	SWFIG	○	×	×
	SWMODE	○	×	×
	SWREAD	○	×	×
	SWWRITE	○	×	×
数值显示	NUMCOLOR	○	×	○
	NUMDSP	○	×	○
	NUMDSP2	○	×	×
	NUMFORM	○	×	×
字符串显示	STRCOLOR	○	×	○
	STRDSP	○	×	○
	STRFORM	○	×	×
图形显示	FIGCOLOR	○	×	×
	FIGDSP	○	×	×
	FIGFORM	○	×	×
	ROTATE	○	×	×
绘图显示	PLTCOLOR	○	×	×
	PLTDSP	○	×	×
棒形图显示	BARCOLOR	○	×	×
	BARDSP	○	×	×
	BARSET	○	×	×

指令分类	助记符	型号/部品支持		
		GC5/7	SCA	SCA2
棒形图显示	BARSHIFT	○	×	×
折线图显示	LNECOLOR	○	×	×
	LNEDSP	○	×	×
	LNESET	○	×	×
	LNESHIFT	○	×	×
	LNESHIFT2	○	×	×
	SETLNEPLOT	○	×	×
百分比图显示	BLTCOLOR	○	×	×
	BLTDSP	○	×	×
	BLTSET	○	×	×
	CIRCOLOR	○	×	×
	CIRDSP	○	×	×
	CIRSET	○	×	×
任意图显示	FRECOLOR	○	×	×
	FREDSP	○	×	×
滑动图显示	SLDDSP	○	×	×
仪表显示	MTRCOLOR	○	×	×
	MTRDSP	○	×	○
指示灯显示	LAMPCOLOR	○	×	×
	LAMPDSP	○	×	○
管状图显示	PIPCOLOR	○	×	×
	PIPDSP	○	×	×
控件控制	CLEAR	○	×	○
	DSPMODE	○	×	×
	EXECPRCODE	○	×	×
	PRDSP	○	×	×
	PRMCTL	○	×	○
	PRMSTAT	○	×	○
	RANGE	○	×	×
串口控制	CLOSECOM	○	×	○
	CLOSESIO	○	×	○
	FLUSH	○	×	○

指令分类	助记符	型号/部品支持		
		GC5/7	SCA	SCA2
串口控制	OPENCOM	○	×	○
	OPENSIO	○	×	○
	REOPENCOM	○	×	○
	SETSIO	○	×	○
	WRITESIO/ WRITESIOB	○	×	○
定时/报警控制	CHKTIM	○	×	○
	CLOSETIM	○	×	○
	CONTTIM	○	×	○
	OPENTIM	○	×	○
	OPENTIM2	○	×	○
	OPENTIM3	○	×	○
	READTIM	○	×	○
	RESETALARM	○	×	○
	SETALARM	○	×	○
	SETTIM	○	×	○
	STARTTIM	○	×	○
	STOPTIM	○	×	○
设备通讯	ADDCYC	○	×	○
	ADDCYC2	○	×	○
	ADDCYCID	○	×	○
	BREAD	○	×	○
	BWRITE	○	×	○
	CYCLIC	○	×	○
	CYCLIC2	○	×	○
	DEVRD	○	×	○
	DEVWR	○	×	○
	EVENTWR	○	×	○
绘图	COLOR	○	×	○
	DOT	○	×	○
	LINE	○	×	○
背景灯控制	BLCTL	○	×	○
	BLSTAT	○	×	○

指令分类	助记符	型号/部品支持		
		GC5/7	SCA	SCA2
背景灯控制	GETBLIGHT	○	×	○
	SETBLIGHT	○	×	○
蜂鸣器控制	BEEP	○	×	○
	SETBEEP	○	×	○
时间/日期	DATE$	○	×	○
	GETDATE	○	×	○
	GETTIME	○	×	○
	SETDATE	○	×	○
	SETTIME	○	×	○
	TIME$	○	×	○
文件控制	FCLOSE	○	×	○
	FINPUT	○	×	○
	FOPEN	○	×	○
	FPRINT	○	×	○
	FSEEK	○	×	○
	FWRITE	○	×	○
	KILL	○	×	○
	LINPUT	○	×	○
	LOF	○	×	○
	MEDIACHK	○	×	○
	MKDIR	○	×	○
	ONFERR	○	×	○
	RENAME	○	×	○
系统控制	ERRCTL	○	×	×
	ERRSTAT	○	×	×
	INTERLOCK	○	×	×
	IOCTL	○	×	×
	IOCTL2	○	×	×
	IOSTAT	○	×	×
功能控制	DECLARE	○	×	○
	EXIT FUNCTION	○	×	○
	FUNCTION…END FUNCTION	○	×	○

指令分类	助记符	型号/部品支持		
		GC5/7	SCA	SCA2
编译器控制	LOCALCHECK	○	×	○
程序启动	EXESTART	○	×	○
画面截取	CAPTURE	○	×	○
IME 控制	GETKEY	×	×	×
	GETKEYT	×	×	×
	SETIMEMODE	×	×	×
	S2U	×	×	×
	U2S	×	×	×
语言切换	GETLANG	○	×	○
	SETLANG	○	×	○
EMAIL 发送	SENDMAIL	○	×	○
文件控制	FILEWRITE	○	×	○
	FILEREED	○	×	○
	FINDFILE	○	×	○
	STRSPLIT	○	×	○

※IME 控制暂不支持

ABS

函数

■ **功能** ABS 函数计算一个表达式的绝对值

■ **格式** ABS (数学表达式)

■ **使用范例**

```
AA = ABS (-50)
AA = ABS (Var)
```

■ **说明** ABS函数计算括号里数学表达式的值 (常数, 整型变量、浮点型变量)

■ **相关项目** 无

■ **程序实例：**

```
evnt
   input type% , id@ , data%
   if data% < 0 then data% = abs(data%)
   numdsp ..num000 , data%
end evnt
```

ADDCYC

指令

■ **功能**　　本指令使部品的K-basic程序能直接读取以控件作声明的设备的值。

■ **格式**　　ADDCYC 控件名称

■ **使用范例**　　ADDCYC ..NUM000

■ **说明**

- 当部品中的控件将参数设置为有效时，使用该指令可以使部品程序能够与控件参数中设置的进行通信。
- 设备的数量必须与控件中将要用到的数量相匹配。(即控件中用到的同程序中用到的单元一样多)
- 控件名必须为局部部品中的控件名称。
- 如果该指令指出的控件没有同If the PLC单元（或内部存储器表）进行通信，那么系统将报错。
- 当数字显示器指定为双字时，则本指令也读取双字。

■ **相关项目**　　CYCLIC, CYCLIC2, ADDCYCID

■ **程序实例**

```
conf
   ADDCYC ..NUM000                    用来显示两个连续单元的数据。
end conf
evnt                                  ’在相应的数据显示器里显示相应的数值。
   input type% , id@ , data%
   id1@ = addcycid ( ..NUM000)        ’读取正在使用的控件的 ID 号。
   i% = getoffset (id1@, id@)+1       ’ 读取将要使用的设备相对于第一个设备的偏
移量。
   id1@ = getid(..NUM000, i%)         ’ 读取相应的 ID 号
   numdsp id1@, data%                 ’ 在显示器上显示ID号。
end evnt
```

ADDCYC2

指令

■ **功能**　　ADDCYC2 指令使部品的K-basic程序可以读取在控件参数中声明过的控件的值。

■ **格式**　　ADDCYC2 控件名称

■ **使用范例**　　ADDCYC2 ..NUM000

■ **说明**

- ADDCYC2 指令的功能同 ADDCYC 指令类似。
- 它们之间的唯一区别是：使用ADDCYC2 指令声明过的设备，即使指向该部品的画面没有显示，也可以通信读取数据。(正在显示其它的画面)。不过，通常用来读取指向它的画面正在显示的PLC设备的数据。

■ **相关项**　　ADDCYC, ADDCYCID

■ **程序实例**

```
conf
   ADDCYC2 ..NUM000                    ’ 用来显示两个连续单元的数据。
end conf
evnt
   input type% , id@ , data%           ’在相应的显示器中显示数据
   id1@ = addcycid ( ..NUM000)         ’读取使用设备的ID值。
   i% = getoffset (id1@, id@)+1        ’用来读取相对于第一个设备的偏移量。
   id1@ = getid(..NUM000, i%)          ’ 读取相应显示器的ID值。
   numdsp id1@, data%                  ’ 在显示器上显示ID值。
end evnt
```

ADDCYCID

函数

■ **功能**　　ADDCYCID 函数用来读取控件的ID值，同时使该控件可以被部品程序读取。

■ **格式**　　ADDCYCID (控件名称)

■ **使用范例**　　ID@ = ADDCYCID (..NUM000)

■ **说明**

- ADDCYCID 函数用来读取控件的ID值，同时使该控件可以被部品程序读取。要进行这种操作，首先要将控件的参数设置为有效，并且要预先在操作参数空里设定PLC设备名称。
- 控件名称必须是局部部品里的控件。
- 如果指定的控件没有填写PLC设备（或存储器表）名称，则会有错误信息出现。

■ **相关项目**　　ADDCYC, ADDCYC2

■ **程序实例：**

```
conf
   ADDCYC2 ..NUM000                    ’ 用来显示两个连续单元的数据。
end conf
evnt
   input type% , id@ , data%           ’在相应的显示器中显示数据
   id1@ = addcycid ( ..NUM000)         ’读取使用设备的ID值。
   i% = getoffset (id1@, id@)+1        ’用来读取相对于第一个设备的偏移量。
   id1@ = getid(..NUM000, i%)          ’ 读取相应显示器的ID值。
   numdsp id1@, data%                  ’ 在显示器上显示ID值。
end evnt
```

ASC

函数

■ **功能**　　ASC 函数指定字符串的第一个字节的字符代码。

■ **使用格式**　　ASC (字符串)

■ **使用范例**

```
AA = ASC (“AABCD”)
AA = ASC (MOJI$)
```

■ **说明**

- ASC 函数以十进制数形式指定括号里所示字符串(字符串变量或常量)第一个字节的字符代码。
- ASC 函数指定以汉字开头的字符串表达式的第一个字符代码。

■ **相关项目**　　CHR$

■ **程序实例：**

```
evnt
   input type, id@, data$
   num = ASC (data$)
   numdsp ..NUM000, num
end evnt
```

ATN

函数

■ **功能** 计算数学表达式的反正切值。

■ **格式** ATN (数学表达式)

■ **使用范例** 角度 = ATN (X/Y)

■ **说明** 函数 ATN 用来计算数学表达式的反正切值。结果应在-π/2 到 π/2 之间，单位为弧度。

■ **相关项目** TAN（正弦）

■ **程序实例：**

```
evnt
   ..........
   pi = 3.141592
   angle% = atn( pi/4)
   numdsp ..num000 , angle%
end evnt
```

AUTO

指令

■ **功能**　　AUTO 指令用来声明自动型变量。

■ **格式**　　AUTO 变量名1 [,变量名2 ...]

■ **使用范例**　　AUTO VAR, XYZ(2,3), MOJI$ * 20

■ **说明**

- 使用AUTO进行定义的变量称为自动型（Auto）变量，该变量只能在函数中被定义或引用。
- Auto型变量的值只能在本变量所在的函数被调用并执行时才有效。
- Auto型变量的值仅在函数被调用并执行时才进行初始化。
- Auto变量可以是普通变量、数组变量或字符串变量。
- 在使用Auto对数组或字符串变量进行定义时无需使用DIM或STRING对其进行声明。
- Auto变量是本软件（SCA2）的一个新特征。

■ **相关项目**

■ **使用实例：**

```
function userfunc%(a%, b%)
   AUTO c%
   c% = a% + b%
   userfunc% = c% / 2
end function
```

BACKUP

指令

■ **功能**　　BACKUP 用来声明停电记忆的变量

■ **格式**　　BACKUP 变量名1 [,变量名2...]

■ **使用范例**　　BACKUP VAR, XYZ(2,3), MOJI$*20

■ **说明**

- BACKUP 用来声明停电记忆型变量。除了具有全局变量的特征以外，Backup(停电记忆型)变量还可以进行掉电保护。
- 普通变量、数组变量及字符串变量都可以被指定为Buckup型变量。
- 在对数组变量或字符串变量进行Buckup声明时，不需要对其进行DIM和STRING声明。

■ **相关项目**　　AUTO, DIM, GLOBAL, LOCAL, STATIC, STRING

■ **程序实例：**

```
conf
   BACKUP a , x(2,3) , moji$ * 40
   ......
end conf
```

BARCOLOR

指令

■ **功能**　　BARCOLOR 用来改变棒图颜色及填充属性。

■ **格式**　　BARCOLOR 棒图控件名, 棒图号, 填充-1, 颜色-1, 背景色-1, 填充-2, 颜色-2, 背景色-2

■ **使用范例**　　BARCOLOR ..BAR000, 2, 3, 1, 4, 5, 2, 1

■ **说明**

- BARCOLOR 用来改变整个棒图显示的颜色和填充、背景色和填充属性。
- 控件名是指棒图控件的名称或与其对应的ID变量。
- 棒图号表示棒图中需要进行改变的棒形图的编号。棒图编号可以是常数也可以是变量。棒图编号从1开始。
- 填充-1 表示棒图本身的填充模式，值0～15。
- 颜色-1 表示棒图填充部分的颜色数值代号，也是0～15。
- 背景色-1 表示棒图填充部分背景颜色的数值代号，也是0～15。
- 填充-2 表示背景填充模式的数值代号，也是0～15。
- 颜色-2 表示背景的填充颜色代号，也是0～15。
- 背景色-2 表示棒图背景填充部分的背景颜色，代号是0～15。

■ **相关项目**　　BARDSP, BARSHIFT

■ **程序实例**

```
conf
   static name@
   name@ = ..BAR000
end conf
evnt
   input type%, id@, data%
   if type% = 3 then
      barcolor name@, 2, 2, 3, 1, 4, 5, 2
   end if
end evnt
```

BARDSP

指令

■ **功能**　　BARDSP ——用棒图形式显示数值。

■ **格式**　　BARDSP 控件名称, 棒图号, 显示的数值

■ **使用范例**　　BARDSP ..BAR000, 1, 30

■ **说明**

- BARDSP 表示用棒图方式显示某数值。
- 控件名称表示棒图的控件名称或其能代表它的ID变量名。
- 棒图号指出当不止一根棒图时要使哪一根（第几根）棒图进行显示。棒图起始编号为1。
- 显示的数值表示要让棒图显示的数值。
- 如果在控件中操作参数（operator parameters）被置为有效，则这里的数值无效!

■ **相关项目**　　BARCOLOR, BARSHIFT

■ **程序实例**

```
conf
   static name@
   name@ = ..BAR000
end conf
evnt
   input type%, id@, data%
   bardsp name@, 2, data%
end evnt
```

BARSET

指令

■ **功能**　　BARSET用来设定棒形图显示的数值。

■ **格式**　　BARSET, 控件名称, 棒图号, 显示的数值

■ **使用范例**　　BARSET .BUHIN.GRAPH, 2, 30.0

■ **说明**

- BARSET用于设定要让棒图显示的数值。在设定棒图显示的数值（使用BARSET）之后再使用PRDDSP比在执行BARDSP后再改变所有的棒图显示速度要快。
- BARSET 表示用棒图方式显示某数值。
- 控件名称表示棒图的控件名称或其能代表它的ID变量名。
- 棒图号指出当不止一根棒图时要使哪一根（第几根）棒图进行显示。棒图起始编号为1。
- 显示的数值表示要让棒图显示的数值大小。

■ **相关项目**　　BARDSP, PRDSP

■ **程序实例**

```
evnt
   BARSET  .buhin.gpaph , 3 , 20.1
   var@ = .buhin.graph
   no = 4
   value = 23
   barset  var@ , no , value
   prdsp var@
end evnt
```

BARSHIFT

函数

■ **功能**　　BARSHIFT 函数将棒图显示的数值向左或向右移移到另一个棒图上并显示它。

■ **格式**　　DATA% = BARSHIFT (控件名, 移动方向, 显示数值)

■ **使用范例**　　DATA% = BARSHIFT (..BAR000, 1, 30)

■ **说明**

- 当在一个棒图显示控件中不止一根棒形图时，将整个棒图上各棒的显示数值向左或向右移动到另一个棒图上，然后再显示。
- 当执行BARSHIFT函数时棒图表示的数值向左或向右移到相邻的棒图上。
- 表示棒图的变量或ID以控件名表示。
- 移动方向：向左或向上为1,向右或向下为－1。
- 显示数值表示要移到相邻棒图上显示的数值。

■ **相关项目**　　BARDSP, BARCOLOR

■ **程序实例**

```
evnt
   input type%, id@, data%
   if data% > 0 then
      abc% = barshift ( ..BAR000, 1, 0)
   else
      abc% = barshift ( ..BAR000, -1, 100)
   endif
end evnt
```

BCD2BIN

函数

■ **功能** BCD2BIN 将BCD数转化成二进制数。

■ **格式** BCD2BIN (数学表达式)

■ **使用范例** BINDATA% = BCD2BIN (BCDDATA%)

■ **说明** BCD2BIN 函数将括号里的数值或数学表达式的值变成二进制数值。

■ **相关项目** BIN2BCD

■程序实例

```
conf
   cyclic 00~D10
end conf
evnt
   input type%, id@, data%
   if type% = 16 then
      data% = BCD2BIN(data%)
      numdsp ..NUM000, data%
   endif
end evnt
```

BEEP

指令

■ **功能** BEEP 指令控制蜂鸣器的ON/OFF。

■ **格式** BEEP 命令值

■ **使用范例** BEEP 1

■ **说明**

- BEEP 指令用于启动或停止蜂鸣器。
- 当命令值为1时，蜂鸣器发出声音；当命令值为0时，停止蜂鸣。
- 可以使用SETBEEP 指令来设定蜂鸣时间。

■ **相关项目** SETBEEP

■ **程序实例**

```
conf
   SETBEEP 50,20,3
end conf
evnt
   input type%, id@, data%
   if id@ = ..SWT000 then
      BEEP 1
   else
      BEEP 0
   endif
end evnt
```

BIN2BCD

函数

■ **功能** BIN2BCD将二进制数转化成BCD数。

■ **格式** BIN2BCD (数学表达式)

■ **使用范例** BCDDATA% = BIN2BCD (BINDATA%)

■ **说明**
- BIN2BCD将二进制数转化成BCD数。
- 如果转换成BCD数后大于99999999，则固定为99999999。

■ **相关项目** BCD2BIN

■ **程序实例：**

```
evnt
   input type%, id@, data%
   data% = BIN2BCD ( data% )
   00~D10 = data%
end evnt
```

BITSET

指令

■ **功能**　　BITSET 指令将变量中指定的位置ON或OFF。

■ **格式**　　BITSET 变量名称, 设置位, ON/OFF值

■ **使用范例**　　BITSET VARIABLE%, 10, 1

■ **说明**

- BITSET 指令将变量中指定的位置ON或OFF。
- 变量名称表示要置位的位所在的变量，它必须为整形数或浮点数
- 设置位用来指定要设置的是哪一位，范围是0～31。必须是变量或常数。
- 当ON/OFF值为1时，表示将指定位置1；当ON/OFF值为0时表示将该位置0。它也可以是变量或常数。

■ **相关项目**　　BITTEST

■ **程序实例**

```
conf
end conf
evnt
   input type% , id@ , data%
   numdsp ..NUM000 , data%
   if bittest ( data% , 31 ) = 1 then
      bitset data% , 31 , 0
   else
      bitset data% , 31 , 1
   endif
   numdsp ..NUM000 , data%
end evnt
```

BITTEST

函数

■ **功能**　　BITTEST用来测试变量指定位的状态0或1。

■ **格式**　　BITTEST (变量名称, 测试位)

■ **使用范例**　　ONOFF% = BITTEST (VARIABLE%, 10)

■ **说明**

- BITTEST用来测试变量指定位的状态0或1。当状态为ON时，返回值为1，当状态为OFF时返回值为0。
- 变量名称表示要测试位所在变量的名称。可以是整形或浮点型变量。
- 测试位表示要进行状态测试的是哪一位。可以是0～31。可以是常数也可以是变量。

■ **相关项目**　　BITSET

■ **程序实例**

```
conf
end conf
evnt
   input type% , id@ , data%
   if bittest ( data% , 0 ) = 1 then
      strdsp ..STR000 , “bit is ON”
   else
      strdsp ..STR000 , “bit is OFF”
   endif
end evnt
```

BLCTL

指令

■ **功能** BLCTL 对背光灯的ON/OFF进行控制。

■ **格式** BLCTL 状态（0或1）

■ **使用范例** BLCTL 1

■ **说明**

- BLCTL 用来对背光灯的ON/OFF进行控制。
- 状态表示要将背光灯置ON还是OFF。1表示灯点亮；0表示关闭背光灯。

■ **相关项目** BLSTAT

■ **程序实例：**

```
evnt
   ret =blstat()
   if ret = 0 then BLCTL 1
end evnt
```

BLSTAT

函数

■ **功能**　　BLSTAT 用来读取背景灯的状态。

■ **格式**　　BLSTAT ()

■ **使用范例**

■ **说明**　　BLSTAT 用来读取当前背景灯的状态。
0：表示当前背景灯为OFF；
1：表示当前背景灯为ON。

■ **相关项目**　　BLCTL

■ **程序实例：**

```
conf
   ret = BLSTAT()
   if ret = 0 then blctl 1
end conf
```

BLTCOLOR

指令

■ **功能**　　BLTCOLOR 用于改变带状图显示的填充模式和颜色。

■ **格式**　　BLTCOLOR 控件名称, 带编号, 填充, 显示颜色, 背景颜色

■ **使用范例**　　BLTCOLOR ..BLT000, 2, 1, 2, 3

■ **说明**

- BLTCOLOR用于改变带状图显示的填充模式和颜色。
- 控件名称指棒图名称或表示该棒图的ID变量。
- 带的编号表示要改变颜色和填充的部分。该部分可以用常数或变量指出。起始部分为1。
- 填充表示代表填充模式的数字代号。
- 显示的颜色表示填充颜色的数字代号。显示的颜色代号可以是0～15。
- 背景颜色表示代表填充颜色的代号的。填充颜色代号也可以是0～15中的一种。

■ **相关项目**　　BLTDSP

■ **程序实例：**

```
evnt
   input type%, id@, zone%, tile%
   BLTCOLOR ..BLT000, zone%, tile%, -1, -1
end evnt
```

BLTDSP

指令

■ **功能** BLTDSP 用带状图形式显示数值。

■ **格式** BLTDSP 控件名称, 带编号, 显示数值

■ **使用范例** BLTDSP ..BLT000, 1, 30

■ **说明**

- BLTDSP 指令用来在带状图的指定区域显示指定的数值。
- 控件名称可以是控件名或能够代表它的ID变量。
- 带编号表示在代表100％的棒形图上的位置代号。可以是整型常数或变量，编号从1开始。
- 显示数值表示以百分比棒图显示的数值大小。
- 当在控件里操作参数（“operation parameters”）被置为有效时，这里设置的显示数值无效。

■ **相关项目** BLTCOLOR

■ **程序实例：**

```
conf
   static name@
   name@ = ..BLT000
end conf
evnt
   input type%, id@, zone%, data%
   BLTDSP ..BLT000, zone%, data%
end evnt
```

BLTSET

指令

■ **功能**　　BLTSET 用来设定带状图显示的数值。

■ **格式**　　BLTSET 控件名称, 带编号, 显示数值

■ **使用范例**　　BLTSET .BUHIN.GRAPH, 2, 30.0

■ **说明**

- BLESET 用来设定百分比棒形图显示的数值。在设定百分比棒图显示的数值（使用BLTSET）之后再使用PRDDSP比在执行BLTDSP后再改变所有的百分比显示速度要快。
- 控件名称表示带状图显示器的控件名称，或能够代表它的ID变量。
- 带编号表示要改变的是哪个带状区域的数值。编号只能是整数或整形变量，从1开始。
- 显示数值是表示带状区域所占地百分比数值。可以是整数或浮点数。

■ **相关项目**　　BLTDSP, PRDSP

■ **程序实例：**

```
evnt
   BLTSET  .buhin.gpaph , 3 , 20.1
   var@ = .buhin.graph
   no = 4
   value = 23
   BLTSET  var@ , no , value
   prdsp var@
end evnt
```

BREAD

函数

■ **功能**　　BREAD 用来批量读取指定设备或存储器表的数值。

■ **格式**　　BREAD 设备名称, 数据量, 读取上来的数据存放的数组变量名称。

BREAD 存储器表, 数据量, 读取上来的数据存放的数组名称。

■ **使用范例**

```
BREAD 00~D0001, 10, VARI(2)
BREAD 00~MTBL(5), NUMS, VARI(X)
```

■ **说明**

- BREAD 批量读取指定设备或存储器表的数值。
- 功能是从指定的设备里读取指定数量的数据，并将其存放在指定的数组变量里面。
- 设备名称用来指定要从哪读取数据。（设备名称仅是要读取的数据区域的起始设备地址）。
- 数据量表示要从指定的设备里连续读取得数据量。
- 读取上来的数据存放的数组变量名称。该变量必须是一维数组变量。读取上来的数据从起始单元按顺序依次连续存放。
- 当数组变量小于读取上来的数据个数时，多余的数据将被抛弃！
- 可以一次读取数据的个数取决于plc的型号。详情可以参考通信手册。
- 对于memory link(存储器连接)方式，读取数据的多少可以使用变量表示！

■ **相关项目**　　BWRITE

■ **程序实例：**

```
conf
   cyclic 00~M01
   static PARAM%(10)
end conf
evnt
   input type%, id@, data%
   if id@ = 00~M01 and data% = 1 then
      BREAD 00~D10, 5, PAARAM%(3)
   endif
end evnt
```

BWRITE

函数

■ **功能**　　BWRITE 函数将数据批量送到指定的设备或存储器单元里。

■ **格式**　　BWRITE 目标设备名称，数据量，要写出的数据变量名称
BWRITE 存储器表，数据量，要写出的数据变量名称

■ **使用范例**　　BWRITE 00~D0001, 10, VAR%(1)
BWRITE 00~MTBL(20), NUM, VAA(1)

■ **说明**

- BWRITE函数将数据批量送到指定的设备或存储器单元里。
- 功能是将指定数组变量的里面指定数量的数据写入到指定的设备里。
- 设备名称用来指定要要将数据写到哪。（设备名称仅是要要写出的数据区域的起始设备地址）。
- 数据量表示要连续写出到指定设备的数据量。
- 要写出的数据变量名称表是存放将要写出数据的变量的名称。变量必须是一维数组变量，数据写到从指定设备开始的连续单元里面。
- 当数组变量小于数据量时，将往多余的单元里送0。当数组变量大于大于数据量时，多余的数据将被忽略。
- 可以批量传送的数据个数取决于PLC的种类。详情请参考《通信连接手册》。
- 对于memory link(存储器连接)方式，读取数据的多少可以使用变量表示！

■ **相关项目**　　BREAD

■ **程序实例：**

```
conf
   cyclic 00~M01
   static PARAM%(10)
end conf

evnt
   input type%, id@, data%
   if id@ = 00~M01 and data% = 1 then
      BWRITE 00~D10, 5, PAARAM%(3)
   endif
end evnt
```

CAPTURE

函数

■ **功能**　　CAPTURE 函数将截取当前画面保存到指定的文件路径。

■ **格式**　　CAPTURE（“保存路径”）

■ **使用范例**　　CAPTURE(“G:\”)

■ **说明**

- 捕获正在显示的画面，在外部存储器(USB存储器/SD卡)中保存。
- CAPTURE函数将截取当前画面保存到指定的文件路径。
- 指定保存路径按照下述规则

文件路径指定（USB）：CAPTURE(“G:\”)
文件路径指定（SD卡）：CAPTURE(“E:\”)
•保存的图像文件名为：Cap YYYYMMDD ttmmss.png
YYYY：年、MM：月、DD：日、tt：时、mm：分、ss：秒

■ **相关项目**　　无

■ **程序实例：**

```
evnt
   input type%,id@,data%
   if type%=3 and id@ = ..swt000 then
     if data% = 0 then CAPTURE (“G:\”)
   else if type%=3 and id@ = ..swt001 then
     if data% = 0 then CAPTURE (“E:\test\”)
   end if
end evnt
```

CHDIR

指令

■ **功能** CHDIR 改变文件夹和/或驱动器位置.

■ **格式** CHDIR 文件夹名称

■ **使用范例** CHDIR “C:TEST”

■ **说明**

- CHDIR 指令用于改变当前文件夹和驱动器。
- 使用字符串常量或变量来指定要改变的文件夹。
- 可以从驱动器后开始指定文件夹名称。

■ **相关项目** MKDIR, RMDIR

■ **使用实例：**

```
conf
end conf
evnt
   ......
   CHDIR “C:”              ’ 改变驱动器
   CHDIR “TEST”            ’ 改变文件夹
   CHDIR “E:ABC”           ’ 改变驱动器和文件夹
   ......
end evnt
```

CHKTIM

函数

■ **功能** CHKTIM 检查指定定时器的状态。

■ **格式** RET = CHKTIM (定时器号)

■ **使用范例** RET = CHKTIM (14)

■ **说明**

- CHKTIM 用于检查指定的定时器是否正在被使用，即是否处于Open状态。
- 定时器号用来指出要检测状态的定时器编号，它必须是在0～15之间的某个常数。
- 通过执行本函数，可能返回如下某个值：
  0: 定时器没被使用。
  1: 定时器正被局部程序使用
  2: 定时器正被其它程序使用。

■ **相关项目** CLOSETIM, STARTTIM, STOPTIM, CONTTIM, SETTIM, READTIM, OPENTIM, OPENTIM2

■ **程序实例：**

```
evnt
   for i = 0 to 15
      ret = CHKTIM (i)
      if ret = 0 then i = 15
   next
end evnt
```

CHR$

函数

■ **功能**　　CHR$ 函数将指定的数值(字符代码)转换成相应的字符。

■ **格式**　　CHR$ (字符代码)

■ **使用范例**　　MOJI$ = CHR$(&H30)

■ **说明**

- CHR$函数将指定的数值(字符代码)转换成与该代码相对应的字符（一个字节）。
- 字符代码必须是1～255之间的某个整数。
- 执行本函数后，返回与该数值相对应的字符。

■ **相关项目**　　ASC

■ **程序实例：**

```
evnt
   input type%, id@ data%
   moji$ = CHR$(data%)
   strdsp ..STR000, moji$
end evnt
```

CINT

函数

■ **功能** CINT 函数将实数取整，将其转换成整数。

■ **格式** CINT (数学表达式)

■ **使用范例** A% = CINT (FLOAT)

■ **说明** • CINT将数学表达式的值取整，并返回一个整数。

■ **相关项目** INT

■ **程序实例：**

```
evnt
   input type%, id@, data
   intvar% = CINT ( data )
   numdsp ..NUM000, intvar%
end evnt
```

CIRCOLOR

指令

■ **功能**　　CIRCOLOR　改变饼图显示的颜色和填充。

■ **格式**　　CIRCOLOR　控件名，带位置，填充，显示颜色，背景颜色

■ **使用范例**　　CIRCOLOR ..CIR000, 2, 1, 2, 3

■ **说明**

- CIRCOLOR改变饼图显示的颜色和填充模式。
- 控件名指饼图控件名称或能表示饼图的ID变量。
- 带位置用来指定要改变得是饼图的哪一部分。起始带为1。
- 填充指指定部分的填充模式，可以是0～15。
- 显示颜色填充颜色代号，0 ～15。
- 背景颜色指背景部分的颜色代码。0 ～ 15。

■ **相关项目**　　CIRDSP

■ **程序实例：**

```
evnt
   input type%, id@, zone%, tile%
   CIRCOLOR ..CIR000, zone%, tile%, -1, -1
end evnt
```

CIRDSP

指令

■ **功能**　　CIRDSP 在指定的饼图区域中显示数值。

■ **格式**　　CIRDSP 控件名, 带编号, 显示值

■ **使用范例**　　CIRDSP ..CIR000, 1, 30

■ **说明**

- CIRDSP 在指定的饼图区域中显示数值。
- 控件名为饼图控件名称或能代表该饼图的ID变量。
- 表示带编号的数值表示要在饼图的哪一部分进行显示。起始编号为1。
- 显示值指要在饼图中进行显示的数值大小。
- 但是当饼图控件中的操作参数（operation parameters）有效时，这个地方的设置将变得无效。

■ **相关项目**　　CIRCOLOR

■ **程序实例**

```
conf
   static name@
   name@ = ..CIR000
end conf
evnt
   input type%, id@, zone%, data%
   CIRDSP ..CIR000, zone%, data%
end evnt
```

CIRSET

指令

■ **功能**　　CIRSET 指令用来设置饼图显示的数据。

■ **格式**　　CIRSET 控件名，扇形编号，显示数据

■ **使用范例**　　CIRSET .BUHIN.GRAPH, 2, 30.0

■ **说明**

- CIRSET 指令用来设置饼图要显示的数据。使用CIRSET指令后在执行PRDSP指令来显示数据比直接使用CIRDSP指令速度要快。
- 控件名指饼图显示控件的名称或能指代它的ID型变量。
- 扇形编号指要改变显示数据的是饼图的哪一部分，编号从1开始。
- 显示数据指扇形编号指定的扇形需要显示的数据。

■ **相关项目**　　CIRDSP, PRDSP

■ **程序实例：**

```
evnt
   CIRSET  .buhin.gpaph , 3 , 20.1
   var@ = .buhin.graph
   no = 4
   value = 23
   CIRSET  var@ , no , value
   prdsp var@
end evnt
```

CLEAR

指令

■ **功能**　　CLEAR 用来清除显示控件中的显示结果。

■ **格式**　　CLEAR 控件名

■ **使用范例**　　CLEAR ..NUM000

■ **说明**

- CLEAR 用于清除控制件内的显示，只设置背景色。
- 当控件为滑动显示器时，该指令清除的是滑动指针。
- 当控件是仪表显示器时，该指令将仪表指针清除。
- 当控件为时钟显示器时，该指令将什么都不清除。
- 控件名指显示器控件名称或能代表该显示器控件的ID变量。

■ **相关项目**　　NUMDSP, STRDSP, FIGDSP, SLDDSP, MTRDSP, FREDSP, PLTDSP, BARDSP, BLTDSP, CIRDSP, LNEDSP

■ **程序实例：**

```
evnt
   input type%, id@, data%
   if data% = 1 then
      CLEAR ..NUM000
   end if
end evnt
```

CLOSE

指令

■ **功能**　　CLOSE 指令关闭指定的部品。

■ **格式**　　CLOSE 部品名称

■ **使用范例**　　CLOSE .B000.

■ **说明**

- CLOSE 关闭屏幕上正在显示的部品，我们未打开状态为关闭状态。
- 当对处于关闭状态（Closed）状态的部品使用该指令时，将不进行任何操作。
- 当处于关闭状态的部品收到消息时，执行该程序。
- 部品名称为部品名称或处于关闭状态的部品的ID号。

■ **相关项目**　　OPEN

■ **程序实例：**

```
evnt
   input type% , id@ , data%
   if pstat(..) = 0 then
      close ..
   endif
end evnt
```

CLOSECOM

指令

■ **功能** CLOSECOM 指令用来临时停止串行口。

■ **格式** CLOSECOM 设备名称

■ **使用范例** CLOSECOM HST

■ **说明**

- CLOSECOM 用来暂时禁止程序从用Opencom指令打开的外部连接设备读取数据。
- 设备名称可以是HST (上位计算机), BCR (条形码阅读机), 或 TKY (十键键盘) 中的一种。

■ **相关项目** OPENCOM

■ **程序实例：**

```
conf
   OPENCOM HST
end conf
evnt
   input type% , id@ , data%
   if type% = 3 and data% = 1 then
      CLOSECOM HST
   else if type% = 3 and data% = 0 then
      REOPENCOM HST
   endif
end evnt
```

CLOSEPARALLEL

指令

■ **功能** CLOSEPARALLEL 用来暂时并行口的数据输入。

■ **格式** CLOSEPARALLEL 输入位

■ **使用范例** CLOSEPARALLEL 3

■ **说明**

- CLOSEPARALLEL 用来暂时禁止程序从用OPENPARALLEL指令打开的外部并行口读取数据信息。
- 输入位表示要禁止数据接收的数据位。

■ **相关项目** OPENPARALLEL, REOPENPARALLEL

■ **程序实例：**

```
conf
   OPENPARALLEL 3
end conf
evnt
   input type% , id@ , data%
   if type% = 3 and data% = 1 then
      CLOSEPARALLEL 3
   else if type% = 3 and data% = 0 then
      REOPENPARALLEL 3
   endif
End evnt
```

CLOSESIO

指令

■ **功能** CLOSESIO 用来关闭无协议通信端口。

■ **格式** CLOSESIO 端口号

■ **使用范例** CLOSESIO 2

■ **说明**

- CLOSESIO 指令用来关闭端口从而中止无协议通信。
- 端口号用来指定要中止无协议通信的端口号。1～3分别代表CH1～CH3。
- 端口号必须是在前面使用OPENSIO指令打开的端口的编号。OPENSIO指令将在后续部分介绍!

■ **相关项目** OPENSIO, SETSIO, WRITESIO, WRITWSIOB, FLUSH

程序实例:

```
conf
   global buf$ * 200
   opensio 2 , 1 , buf$
   setsio 2 , &HD
end conf
evnt
   strdsp ..STR000 , buf$
   CLOSESIO 2
end evnt
```

CLOSETIM

指令

■ **功能**　　CLOSETIM 用来关闭指定的定时器

■ **格式**　　CLOSETIM 定时器号

■ **使用范例**

```
CLOSETIM TIMID@
CLOSETIM VAR
```

■ **说明**

- CLOSETIM 指令将由 OPENTIM, OPENTIM2, 或者OPENTIM3 指令打开的定时器关闭并返回给系统使用。
- 系统最多可以使用16个定时器，不使用的定时器要返回给系统，当使用的定时器数超过16个时，系统将会报错。
- 定时器号指要将其关闭并返回给系统的定时器号，该号是使用整数数值还是使用ID号取决于该定时器的打开方式。(详细情况参考“OPENTIM”, “OPENTIM2”, 和 “OPENTIM3.”)

■ **相关项目**　　OPENTIM, OPENTIM2, OPENTIM3, STARTTIM, STOPTIM, CONTTIM, SETTIM, READTIM

■ **程序实例：**

```
conf
   static timid@
   timid@ = opentim()
   setim timid@, 20, 0
   starttim timid@
end conf
evnt
   input type% , id@ , data%
   if type% = 3 and id@ = ..SWT000 then
      stoptim timid@
   else if id@ = ..SWT001 then
      closetim timid@
   end if
end evnt
```

COLOR

指令

■ **功能**　　COLOR 用于设置直线或点的颜色、大小、及类型。

■ **格式**　　COLOR 显示颜色，线型，线/点的粗细

■ **使用范例**　　COLOR 1, 0, 2

■ **说明**

- COLOR 用于设置直线或点的颜色、大小、及类型。在LINE和DOT指令里设定的值比本指令设定的值有更高的优先级别。
- 显示颜色表示要让直线或点显示的颜色代号，可以是0～15。
- 线型表示线的类型（如实线或虚线）的代号，可以是0～3。
- 点/线的粗细表示点或线的粗细大小的代号，范围是0～2。

命令	種類	太さ		
		0	1	2
Line	0	細線	太線	極太線
	1～3	点線	太線	極太線
dot	0～2	小さい点	中サイズ	大きい点

■ **相关项目**　　LINE, DOT

■ **程序实例：**

```
conf
   color 1 , 0 , 2
end conf
evnt
      ....
   dot 100,200
   dot 100,300
   color 1 , 0 , 0
   line 100,200,100,300
      ....
end evnt
```

CONF ... END CONF

指令

■ **功能**　　CONF ... END CONF 用来声明 Conf 程序块。

■ **格式**

```
CONF
.....
.....
END CONF
```

■ **使用范例**

```
CONF
static VAR%
END CONF
```

■ **说明**

- 画面和部品的Conf程序块仅在画面被显示时执行一次，而在画面没被显示时不执行，本程序在显示其它画面之后跳到本画面时，该程序又执行一次。
- 全局画面及其部品的Conf 块程序仅在系统上电时执行一次。
- 初始化（INIT）程序块用来写初始化处理程序。
- 处于关闭状态的部品其Conf 程序块不执行，只有当它被打开后其Conf 程序块才能被执行。（请参考 “OPEN”指令）

■ **相关项目**　　EVNT ... END EVNT, INIT ... END INIT

■ **程序实例：**

```
CONF
   static moji$
END CONF
evnt
   input ty%, id@, dat$
end evnt
```

CONST

指令

■ **功能**　　CONST 指令用来声明一个常数。

■ **格式**　　CONST 常数名 = 常数

■ **使用范例**　　CONST #MAX#=10

■ **说明**

- 常数名必须是使用一对“＃”符号包围。
- 如果在程序中声明了一个常数，那么在程序中使用到该常数名的地方都将用该常数代替。
- CONST 指令不能用在全局画面程序里!
- 常数声明是SCA2 软件的又一大特性。

■ **相关项目**

■ 程序实例：

```
conf
   global L%
   const #MAXLENGTH#=100
   if L > #MAXLENGTH# then
      L = #MAXLENGTH#
   end if
end conf
```

CONTTIM

指令

■ **功能**　　CONTTIM 重新启动暂停的定时器。

■ **格式**　　CONTTIM 定时器号

■ **使用范例**

```
CONTTIM TIMID@
CONTIM 4
```

■ **说明**

- CONTTIM 用于重新启动使用STOPTIM指令暂停地定时器。在启动后，定时器从停止时的状态继续开始计时。
- 定时器号表示要重新开始的定时器编号。编号为ID型数还是整型数由其被打开的方式决定。(请参考“OPENTIM”，“OPENTIM2”，和 “OPENTIM3.”指令)。

■ **相关项目**　　OPENTIM, OPENTIM2, OPENTIM3, STARTTIM, STOPTIM, CLOSETIM, SETTIM, READTIM

■ **程序实例：**

```
conf
   static timid@
   opentim2(3)
   settim 3, 20, 0
   starttim 3
end conf
evnt
   input type% , id@ , data%
   if type% = 3 and id@ = ..SWT000 then
      stoptim 3
   else if id@ = ..SWT001 then
      conttim 3
   end if
end evnt
```

COPY

指令

■ **功能**　　COPY 用于画面硬拷贝。

■ **格式**　　COPY 颜色代号

■ **使用范例**　　COPY 5

■ **说明**

- COPY 指令用当前显示画面硬拷贝。颜色代号表示将其指定为打印出来后其颜色为黑色的的颜色代号。
- 除了颜色代号0～15外，如果颜色代号选择16，那么所有偶数代号的颜色打印出来后将为黑色；如果颜色代号选择为17，那么所有奇数代号的颜色在打印出来后将为黑色。
- 使用单色打印机时，如果指定的颜色代号为偶数，打印出来后效果将同设置颜色代号2一样；如果指定的颜色代号为奇数，打印出来后效果将同设置颜色代号1相同。
- "颜色代号" 仅在当触摸屏上"System Setup" —"Printer Setup"—"Screen Print Mode"设置为"Select Color"时有效。

■ **相关项目**

■ **程序实例**

```
evnt
   input ty%,id@
   if id@ = ..SWT000 then COPY 8
end evnt
```

COS

函数

■ **功能** COS 用于计算给定数学表达式的余弦值。

■ **格式** COS (数学表达式)

■ **使用范例** X = COS (ANGLE)

■ **说明** COS 函数计算给定数学表达式的余弦值，数学表达式值的单位为弧度。

■ **相关项目** ATN, SIN, TAN

■ **程序实例：**

```
evnt
   angle = 3.141592/3
   x = COS ( angle )
end evnt
```

CURDIR

指令

- **功能** CURDIR 指令将表示当前文件夹路径的字符串写到字符串变量。
- **格式** CURDIR 字符串变量
- **使用范例** CURDIR PATH$
- **说明** 包括磁盘驱动器名在内的整个文件路径都将被写到字符串变量。
- **相关项目** DIR,CHDIR,MKDIR,RMDIR
- **程序实例：**

```
conf
   strdsp ..str, "curdir"
end conf
evnt
   input type%, id@, data%
   if data% = 1 then
      curdir path$
      strdsp .dsp.str, path$
   end if
end evnt
```

CVB

函数

■ **功能** CVB 函数用于从字符串变量的任意位置分配数据。

■ **格式** CVB (字符串变量名, 指定位)

■ **使用范例** VAR% = CVB (MOJI$, 5)

■ **说明**

- CVB 函数从指定字符串变量的指定位读取一个字节的数据，分配的数据作为一个整数对待。
- 指定的位必需是一个整数或整型变量。1表示开始位(最左边位)。

■ **相关项目** MKS, MKB, MKW, MKI, MKF, MKID, CVW, CVI, CVF, CVID

■ **程序实例：**

```
conf

end conf
evnt
   org$ = “1234567”
   data% = CVB ( org$, 3 )  ’从左边数第三位（3）。
   numdsp ..NUM000, data%’   显示 51(&H33).ASCII码为&H33.
end evnt
```

CVF

函数

■ **功能** CVF 函数从指定字符串变量的特定位置读取数据并转换为浮点数。

■ **格式** CVF (字符串变量名，指定位)

■ **使用范例** VAR = CVF (MOJI$, 5)

■ **说明**

- CVF 从指定字符串变量的指定位置开始读取4个字节的数据。读取的数据作为浮点对待。
- 指定位必需为整型或浮点 型变量或常数。1表示字符串的初始位。
- CVF函数返回一个浮点数。
- 读取的数值被转化成一个浮点数.

■ **相关项目** MKS, MKB, MKW, MKI, MKF, MKID, CVB, CVW, CVI, CVID

■ **程序实例：**

```
conf

end conf
evnt
   org$ = “1234567”
   strdsp ..STR000, org$
   mkf org$, 2, 1.23
   strdsp ..STR001, org$          ’ 字符串将不能正确显示。
   data% = CVF ( org$, 2 )
   numdsp ..NUM000, data%         ’ 显示 1.23.
end evnt
```

CVI

函数

■ **功能**　　CVI 函数用来从指定的字符串的任意位置开始读取数据。

■ **格式**　　CVI (字符串变量名, 指定位置)

■ **使用范例**　　VAR% = CVI (MOJI$, 5)

■ **说明**

- CVI 函数从给定字符串变量的指定位置开始读取4个字节的数据。读取的数据被认为是整数。
- 指定的位置必须是整型或浮点型变量或常数。起始位置为1，依次递增。
- 截取的数据自动被转换ASCII码数。

■ **相关项目**　　MKS, MKB, MKW, MKI, MKF, MKID, CVB, CVW, CVF, CVID

■ **程序实例：**

```
conf

end conf
evnt
   org$ = “1234567”
   data% = CVI ( org$, 3 )
   numdsp ..NUM000, data%          ’ 显示的结果为 &H36353433.
end evnt
```

CVID

函数

■ **功能**　　CVID 函数从字符串变量的指定位置读取（截取）数据。

■ **格式**　　CVID (字符串变量名，指定位置)

■ **使用范例**　　VAR@ = CVID (MOJI$, 5)

■ **说明**

- CVID 函数从字符串变量里指定位置开始读取6个字节的数据，读取的数据被视为ID值。
- 指定的位置必须为整型或浮点型变量或常数。1 表示初始位置。
- CVID 函数返回一个ID值。
- 截取的值被转换成一个ASCII码数。

■ **相关项目**　　MKS, MKB, MKW, MKI, MKF, MKID, CVB, CVW, CVI, CVIF

■ **程序实例：**

```
conf

end conf
evnt
   org$ = “1234567”
   data@ = CVID ( org$, 1 )
end evnt
```

CVW

函数

■ **功能**　　CVW函数从字符串变量的指定位置读取（截取）数据。

■ **格式**　　CVW (字符串变量名，指定位置)

■ **使用范例**　　VAR% = CVW (MOJI$, 5)

■ **说明**

- 函数从字符串变量里指定位置开始读取2个字节的数据，读取的数据被视为整数值。
- 指定的位置必须为整型或浮点型变量或常数。1 表示初始位置。
- 截取的值被转换成一个ASCII码数。

■ **相关项目**　　MKS, MKB, MKW, MKI, MKF, MKID, CVB, CVI, CVF, CVID

■ **程序实例：**

```
conf

end conf
evnt
   org$ = “1234567”
   data% = CVW ( org$, 3 )
   numdsp ..NUM000, data%          ’ Displays &H3433.
end evnt
```

CYCLIC

指令

■ **功能**　　CYCLIC 用来声明“循环读取指定设备或存储器表的内容”。

■ **格式**　　CYCLIC 设备名称1, 设备名称2, 设备名称3 *数量
CYCLIC 存储器表1，存储器表2，存储器表3，存储器表4 *数量

■ **使用范例**　　CYCLIC 00~D01, 00~D10 * 5
CYCLIC 00~MTBL(100), 00~MTBL(200) * 10

■ **说明**

- CYCLIC用来使用通信方式循环读取指定设备或存储器表的内容。如果发现当前内容同前一次读取到的内容不一致，证明内容发生了改变，则向控制程序发出一个消息。该指令在控制程序运行过程中不运行!
- 该指令必须在程序中使用该设备之前使用。
- 当从设备里读取数据时（如A=01~R2000），无需用CYCLIC进行声明。
- 当对存储器表使用该操作时，存储器表的大小必须使用整数。
- 在对存储器表使用CYCLIC指令时，当从上位机或控制程序向存储器表写入数据时，会发出消息。（即使是其内容没有发生变化）
- “*数量” 指从指定的设备单元或存储器表头开始连续读取的数据个数。
- 当画面被切换后，将向所有使用了CYCLIC 指令的画面发送一个消息。

■ **相关项目**　　INPUT

■ **程序实例：**

```
conf
   cyclic  00~d01 , 00~d4 * 3
   cyclic  00~mtbl(20), 00~mtbl(100)
end conf
evnt
   input ty%,id@,dat%
   if id@ = 00~mtbl(20) then
     numdsp ..num , dat%
   end if
   ......
end evnt
```

CYCLIC2

指令

■ **功能**　　CYCLIC2 用来声明“以双字为单位循环读取指定设备或存储器表的内容”。

■ **格式**　　CYCLIC2 设备名称1, 设备名称2, 设备名称3, *数量

■ **使用范例**　　CYCLIC2 00~D01, 00~D10 * 5

■ **说明**

- CYCLIC2 除了在读取数据时以双字为单位之外，其它功能与 CYCLIC 相同。
- 设备号大的为高位字。
- 不能用来读取触摸屏内部存储器的内容。
- 当画面被切换后，也将向所有使用了CYCLIC2 指令的画面发送一个消息。

■ **相关项目**　　INPUT, CYCLIC

■ **程序实例：**

```
conf
  cyclic2  00~d01 , 00~d7 * 3
end conf
evnt
  input ty%,id@,dat%
  if id@ = 00~d01 then
    numdsp ..num , dat%
  end if
  ......
end evnt
```

DATE$

函数

■ **功能**　　DATE$ 用来读取系统的当前日期。

■ **格式**　　DATE$

■ **使用范例**　　MOJI$ = DATE$

■ **说明**

- 读取的当前年、月、日分别以两位数字表示，形如YY/MM/DD。
- DATE$ 函数不能用来设定日期。
- 对于带有使用后备电池日期芯片的触摸屏(GC56LC or GC55EM)，一旦当使用SETDATE 命令设置过日期之后，其日期将自动实时更新，即使在系统断电之后。而对于不带有使用后备电池日期芯片的触摸屏(GC53LC or GC53LM)，在系统断电后，其系统日期初始化为98-01-01，系统时钟初始化为00:00:00。

■ **相关项目**　　GETDATE,GETTIME,SETDATE,SETTIME,TIME$

■ **程序实例：**

```
conf
   moji$ = DATE$
   strdsp ..STR000 , moji$
end conf
```

DECLARE

指令

■ **功能**　　DECLARE 用来声明一个函数

■ **格式**　　DECLARE 函数名称 [函数类型声明](变量声明1[, 变量名] ...)

■ **使用范例**　　DECLARE ADD\%(A\%,B\%)

■ **说明**

- DECLARE 用来声明在程序中将要使用到的某种函数（这种声明称为函数原型声明）。
- 函数在声明时采用如下三种方式之一：
  - 局部函数：在非全局画面中定义。
  - 全局函数：在全局画面中定义。
  - 库函数：在函数库里面定义。
- 声明的函数类型(函数原型中定义) 必须同函数本身的类型一致。
- 这是SCA2 的又一个特性。.

■ **相关项目**　　FUNCTION, FUNCTIONCHECK

■ **程序实例：**

```
DECLARE my add(a%,b%)
conf
   global x%,y%
   local sum%
   sum% = my add(x%,y%)
end conf
```

DEVRD

指令

■ **功能**　　DEVRD 用来读取指定设备的内容。

■ **格式**　　DEVRD 设备名, 偏移量, 变量名

■ **使用范例**　　DEVRD 00~D10, 10, VALUE%

■ **说明**

- DEVRD 用来从某个设备里读取数据，该设备为“指定的设备地址＋偏移量”。
- 偏移量为要读取得目标地址到指定设备的距离。DEVRD 指令从与指定的距离相对应的设备单元读取数据。偏移量必须是整型或浮点变量或常数。
- 变量名用来指定读取数据存放存放的变量。它必须是整型或浮点变量。
- DEVRD 指令用在已用Cyclic 指令声明过的连续设备单元，如 CYCLIC 00~mtbl(100) * 10 。
- 如果指定的设备不存在，系统将会报错。
- 该指令要在事件块（event block）中使用。

■ **相关项目**　　CYCLIC, EVENTWR, DEVWR

■ **程序实例**

```
conf
   cyclic  00~mtbl(100) * 5       ’数据量为5
end conf ’从00~mtbl(100).
evnt
   input type% , id@ , data%   ’                  当开关被按下时，就从连续设
   if id@ = ..SWT000 and data% = 1 then           ’    备里读取数据并显示出来。
      for i% = 0 to 4
         id@ = getid ( ..NUM000, i%+1)
         DEVRD 00~mtbl(100), i% , data%
         numdsp ..NUM000, data%
      next
   endif
end evnt
```

DEVWR

指令

■ **功能**　　DEVWR　将数据写到指定的设备。

■ **格式**　　DEVWR 设备名, 偏移量, 写入值

■ **使用范例**　　DEVWR 00~mtbl(100), 10, 5

■ **说明**

- DEVWR 将数据写到离“设备名”距离为“偏移量”的单元里。
- “偏移量”到“设备名”的距离。DEVWR指令将数据写到与“偏移量”相对应设备单元里。偏移量必须是一个整型或浮点型常数或变量。
- 写入值指要写出的值。它必须是一个整型或浮点型变量或常数。
- DEVWR　用于已经用EVENTWR 指令声明过的连续变量（如EVENTWR 00~ mtbl(100) * 10）
- 当指定的设备并不存在时，系统将会报错。

■ **相关项目**　　CYCLIC, EVENTWR, DEVRD

■ **程序实例**

```
conf
    eventwr  00~mtbl(100) * 5    ’声明将数据写到相对mtbl(100)偏移
end conf                         ’量为5之内的单元里面。
evnt
   input type% , id@ , data%       ’ 将10写到相对00~mtbl(100)偏移
   DEVWR 00~mtbl(100), data% , 10  ’ 量为data% 的单元里。
end evnt
```

DIM

指令

■ **功能**　　DIM 指令用来定义一个数组。

■ **格式**　　DIM 变量名（最大下标1 [, 最大下标2] ...）

■ **使用范例**　　DIM ABC$(20), XYZ%(4,4,3), LOC!

■ **说明**

- DIM 将由“变量名”定义一个变量定义为局部变量。
- 局部变量只能被它所声明的程序中引用。如果使用了未被定义的局部变量，系统编译器将会发出警告。每个局部变量当它所在的程序被执行时都要进行初始化。
- 当某个变量带有下标（用括号形式）时，则该变量就是数组变量。
- 括号里“最大下标”的个数表示数组的维数。当为数超过1时，应将各下标用“，”隔开。
- “最大下标”表示元素可以指定的最大下标值，下标从0开始。
- 即使是不用DIM指令，变量也可以作为数组使用。在这种情况下，数组的最大下标只能为10。
- 当将字符数串定义成数组时，可以指定字符串的大小。
- 定义的数组过多可能导致不能使过多的画面，因为这样会使OIP的工作区域变得较小。
- SCA2 一个新的功能就是无需将局部变量和数组变量分开来定义。
- DIM 指令是为了保持同GCSGP3的兼容性，在定义局部变量时使用 LOCAL 而不是DIM。
- 当使用DIM来定义数组变量时，就保持了同GCSGP3的兼容性。

■ **相关项目**　　AUTO, BACKUP, GLOBAL, LOCAL, STATIC, STRING

■ **程序实例**

```
conf
   DIM FLOAT(10),ID@(5),MOJI$(10) * 40
   for i% = 1 to 5
      FLOAT(i%) = i*3
   next
end conf
```

DIR

函数

■ **功能**

DIR 函数将文件夹或文件数据列表送入一个字符串变量，并返回创建的数据量值。

■ **格式**

DIR (文件夹名称，文件属性值，偏移量值，字符串变量)

■ **使用范例**

```
NUM% = DIR("A:SUBDIR", &H20, 6, LIST$)
```

■ **说明**

- 文件夹名称可以是包括驱动器在内的整个路径或者是以当前文件夹名称开头的简化名称。
  Example: A:\SUBDIR1\SUBDIR2　　SUBDIR2\SUBDIR3
- 为了创建单个文件的数据，应该指定一个文件名，而并非文件夹路径名。
- 选择创建数据的文件属性值应该是如下标记的逻辑或值：
  &H01: 只读文件
  &H02: 隐含文件
  &H04: 系统文件
  &H08: 标卷
  &H10: 子文件夹
  &H20: 标准文件
- 为了绕开开始n个数据，必须要指定一个偏移量。
- 创建的数据长度固定为40个字节的记录。后面跟有如下详细数据：

文件名	扩展名	大小	更新日期	更新时间
disk 1	.	<VOL>	98-01-04	15:34
SAMPLE	.EXE	98765	99-09-12	10:09
ABCDE	.	12355	01-08-11	14:45
KBASIC	.	<DIR>	99-04-28	08:59
TEST2	.C	256	97-11-08	13:51
DOWNLOAD	.OIP	<DIR>	96-07-02	17:00
DATA 007	.	32	00-03-19	21:07

本例中，创建的7个数据一共有可以280个字节。
要创建的数据量取决于字符串变量的大小。

■ **相关项目**

DIR,CHDIR,MKDIR,RMDIR

■ **程序实例**

```
conf
   global dname$(13), pname1$(13), pname2$(13)
   global dsel%, p1sel%, p2sel%
   static list$*2000
   strdsp ..str, "dir"
end conf
```

```
evnt
   input type%, id@, data%
   if data% = 1 then
      path$ = dname$(dsel%) + pname1$(p1sel%) + pname2$(p2sel%)
      strdsp .dsp.str, path$
      num% = dir(path$, &H3F, 0, list$)
      strdsp .dsp.str, list$
      numdsp ..num000,num%
   end if
end evnt
```

DINV

指令

■ **功能**　　DINV 将指定屏幕区域上的颜色反转。

■ **格式**　　DINV 左上角X坐标, 左上角Y坐标, 右下角X坐标,右下角Y坐标

■ **使用范例**　　DINV 10, 10, 30, 30

■ **说明**

- 将由其后面各坐标指定的矩形屏幕区域颜色反转。
- 屏幕左上角坐标为(0, 0)，水平方向（向右）为X轴，竖直方向（向下）为Y皱。
- 颜色作如下反转：
  0 — 15 ，1 — 14 … 7 — 8 . . . 即为互补颜色。
  对于单色屏，深色变浅色，浅色变深色，透明色变浅色。
- 如果在INIT或CONF程序块中使用，因为绘图在该程序后执行，所以颜色并不变化。
  该指令在EVNT程序块中使用。

■ **相关项目**　　无

■ **程序实例**

```
evnt
   input ty,id@,dat
   if ty = 3 and id@ = ..SWT000 then
      DINV 0,0,639,399
   endif
end evnt
```

DOT

指令

■ **功能**　　DOT 在屏幕上画一个点。

■ **格式**　　DOT X1, Y1

■ **使用范例**　　DOT 20,300

■ **说明**

- 这是在指定的坐标(X1,Y1)上显示点的语句。
- X1是0 ~ 799,Y1是gc - a24:0 ~ 479,GC-A25/ a26:0 ~ 599的范围。
- 点作为画面的背景直接被显示。
  在点所表示的区域进行零件的打开/关闭、控制的显示,
  在零件的区域内被表示了的点有被清除的情况。
- 被清除的点不会再被表示。
- 点的大小/颜色用COLOR来指定。

■ **相关项目**　　COLOR

■ **程序实例**

```
conf
   color 1 , 0 , 2
end conf
evnt
     ....
   dot 100,200
   dot 100,300
   color 1 , 0 , 0
   line 100,200,100,300
      ....
end evnt
```

DSPMODE

指令

■ **功能**　　DSPMODE 改变控件的显示模式。

■ **格式**　　DSPMODE　控件名, 显示模式

■ **使用范例**　　DSPMODE　..NUM000, 2

■ **说明**

- DSPMODE用于改变控件的显示模式。
- 控件名可以是控件的名称或代表控件的ID变量。
- 控件模式用于指定控件的显示模式。控件模式有如下三种：
  0: 正常模式
  1: 反转模式
  2: 闪烁模式　(显示颜色与背景色交替)
  3: 点灭显示模式　(显示和不显示交替)

■ **相关项目**　　NUMDSP, STRDSP, FIGDSP, SLDDSP, MTRDSP, FREDSP, PLTDSP, BARDSP, BLTDSP, CIRDSP, LNEDSP

■ **程序实例**

```
evnt
   input ty,id@,data
   if id@ = ..SWT000 then
      DSPMODE ..NUM000 , 3
   endif
end evnt
```

EOF

函数

■ **功能**　　EOF 函数用来检测是否到了文件结束。

■ **格式**　　EOF（文件编号）

■ **使用范例**　　AAA = EOF（文件编号）

■ **说明**

- 文件编号用来指出要检测的是哪个文件，本指令用来检测当前是否到了文件的最后位置。文件编号必须同使用FOPEN指令打开的文件编号相同。
- 当返回值为1时，表示已经到达文件的结束位置；当返回0时，表示还没有到达。

■ **相关项目**　　FOPEN, FIELD, FCLOSE, FPUT, FGET

■ **程序实例**

```
conf
   field  5
      global no%
      global moji1$ , moji2$
   end field
   global sum%
   fopen  ``C:TEST'', 2 , 5
   ......
end conf
evnt
   while EOF(5) = 0
      fget 5, i
      numdsp ..NUM000, no%
      strdsp ..STR000, moji1$
      strdsp ..STR001, moji2$
   wend
   fclose (5)
end evnt
```

ERRCTL

指令

■ **功能**　　ERRCTL 用来控制错误代号的显示位置。

■ **格式**　　ERRCTL 模式（mode 位置代号）

■ **使用范例**　　ERRCTL 0

■ **说明**

- ERRCTL用来控制错误代号的显示位置。
- 与错误代号显示位置相对应的有如下三种模式：
  mode= 0: 在屏幕下方显示错误代号
  mode= 1: 使用错误显示部品显示出错代号。
  - 当mode = 1时， 错误代号为4000～4499和5000～5999之间的消息向错误显示部品发送。(部品ERRPTS 在全局画面上).
  - 代号在2000～2999之间的消息只向错误显示部品发送。
  - 根据错误的类型和发送错误的ID，错误代码、出错所在画面、出错画面的注册号等都向错误显示部品发送 （如果是程序出错，部品编号将用-1代替。）
  - 当错误显示部品ERRPTS不在全局画面上，则错误显示在屏幕窗口的最底部。

■ **相关项目**　　ERRSTAT

■ **程序实例**

```
evnt
   input ty%,id@,dat%
   if id@ = ..sw1 then
      if errstat () = 1 then
         errctl 0
      else
         errctl 1
      endif
   endif
end evnt
```

ERRSTAT

函数

■ **功能**　　ERRSTAT 函数用来读取（检测）错误代号显示位置。

■ **格式**　　ERRSTAT

■ **使用范例**　　ERRSTAT()

■ **说明**

- ERRSTAT 函数读取当前的错误显示位置。
- 当函数被执行时，将返回下列某个值：
  当返回值为0时，表示错误显示在屏幕下方。
  当返回值为1时，表示错误显示错误显示部品里。

■ **相关项目**　　ERRCTL

■ **程序实例**

```
evnt
   input ty%,id@,dat%
   if id@ = ..sw1 then
      if errstat () = 1 then
         errctl 0
      else
         errctl 1
      endif
   endif
end evnt
```

EVENTWR

指令

■ **功能**　　EVENTWR 用来声明数据要写入的目标设备。

■ **格式**　　EVENTWR 设备名1， 设备名2，设备名3 *数量

■ **使用范例**　　EVENTWR 00~D01, 00~D10 * 5

■ **说明**

- EVENTWR用来声明数据要写入的目标设备。注意，仅是声明，它本身并不执行数据写入。
- 使用“*数量”可以声明连续的多个设备。但是这并不意味着将所有数据一次写到声明的设备里。
- DEVWR 指令用来将数据写到声明的设备单元里。
- 在执行DEVWR指令之前，一定要预先声明数据将要写入的设备。

■ **相关项目**　　CYCLIC, DEVRD, DEVWR

■ **程序实例**

```
conf
   EVENTWR  00~mtbl(100) * 5          ’    声明：从mtbl(100)开始共5个单元。
end conf
evnt
   input type% , id@ , data%                ’  将10写到相对mtbl(100)偏移量为
   devwr 00~mtbl(100), data% , 10           ’  data%的设备单元里。

end evnt
```

EVNT ... END EVNT

指令

■ **功能**　　EVNT...END EVNT 用来声明程序中的事件（EVNT）程序块。

■ **格式**

```
EVNT
   .....
   .....
END EVNT
```

■ **使用范例**

```
EVNT
   input  ty , id@ , data
   .....
END EVNT
```

■ **说明**

- EVNT程序块（即事件程序块）在接到消息后运行。例如当开关被按下或定时时间到了之后。

■ **相关项目**　　CONF ... END CONF, INIT ... END INIT

■ **程序实例**

```
conf
   static moji$
end conf
evnt
   input ty%, id@, dat$
end evnt
```

EXECPRCODE

指令

■ **功能**　　EXECPRCODE 指令对控件里的数据进行运算。

■ **格式**　　EXECPRCODE 控件名称, 类型, 运算数据, 变量名称

■ **使用范例**　　EXECPRCODE ..NUM000, 0, 20, VAR%

■ **说明**

- 当部品里的控件参数有效时，该指令对设置的部品进行运算。
- 类型通常为0。当指定的控件为plot（绘图）显示器且类型为0时，对X轴数据进行运算；当类型为1时，对Y轴数据进行运算。
- 控件名称必须是当前部品里的控件。
- 运算数据指参见运算的数据，必须是整型或浮点变量或常数。
- 变量名称指运算结果要写入的目标变量，必须是整型或浮点变量。
- 如果没有在指定的控件里没有操作码，则运算数据将被送到指定的变量。

■ **相关项目**　　无

■ **程序实例**

```
conf

end conf
evnt
   input type% , id@ , data%
   EXECPRCODE ..NUM000, 0, data%, data1%
   numdsp ..NUM001, data1%
end evnt
```

EXESTART

指令

■ **功能**　　EXESTART 指令用于启动软件。

■ **格式**　　EXESTART（指令、“软件功能码”）

■ **使用范例**　　EXESTART(4, “-11 -mMON”)

■ **说明**

• 停止用户程序并启动GC-A2上的其他软件。

GC-A2中可使用的其他软件如下：

① 梯形图工具 (功能码: 4)

② PLC 监控工具 (功能码: 6)

指令对软件工具有固定的定义号分配，详情请参考工具软件对应的说明文档

■ **相关项目**　　无

■ **程序实例**

```
evnt
input type% , id@ , data%
if type% = 3 then
  if id@ = ..swt000 and data% = 0 then EXESTART(4, "-11 -mMON")
  if id@ = ..swt001 and data% = 0 then EXESTART(6, "-11 -mMON -s0")
end if
```

EXIT FUNCTION

指令

■ **功能**　　EXIT FUNCTION 指令的功能是强制退出函数

■ **格式**　　EXIT FUNCTION

■ **使用范例**

```
FUNCTION DIV%(A%,B%)
IF B%=0 THEN EXIT 功能
DIV%=A%/B%
END 功能
```

■ **说明**

- EXIT FUNCTION 将强制退出函数块的执行，并将控制权交给调用它的程序。

■ **相关项目**　　DECLARE, FUNCTION, FUNCTIONCHECK

■ **程序实例**

```
declare my div%(a%,b%)
conf
   global x%,y%
   local share%
   share% = my div(x%,y%)
end conf
function my div%(a%,b%)
   if b%=0 then EXIT FUNCTION
   my div%=a%/b%
end function
```

EXP

函数

■ **功能** EXP 函数用来计算自然对数(欧拉(Euler)常数e)为低指数函数的值。

■ **格式** EXP (数学表达式)

■ **使用范例** VAR = EXP (A/2)

■ **说明** EXP 函数返回指数运算结果。

■ **相关项目** LOG

■ **程序实例**

```
evnt
   input ty,id@,data
   if ty = 3 then
      numdsp ..NUM000, EXP(10)
   else
      numdsp ..NUM000, EXP(5)
   endif
end evnt
```

FCLOSE

指令

■ **功能**　　FCLOSE 用于关闭指定的文件

■ **格式**　　FCLOSE 文件编号

■ **使用范例**　　FCLOSE 5

■ **说明**

- FCLOSE关闭由文件编号指定的文件。
- 文件编号编号必须与使用FOPEN指令打开的文件相同。如果不同，系统将会报出错信息，编号在1～16之间。

■ **相关项目**　　FOPEN, FIELD, FPUT, FGET

■ **程序实例**

```
conf
   field  5
      global no%
      global moji1$ , moji2$
   end field
   fopen  “MEMORY”, 2 , 5
   ......
end conf
evnt
   ......

   FCLOSE 5
end evnt
```

FGET

指令

■ **功能**　　FGET 从指定的文件中读取数据。

■ **格式**　　FGET 文件编号, 记录代号

■ **使用范例**　　FGET 5, 3

■ **说明**

- FGET 在指定的文件中将指定代号的记录读入到由FIELD...END FIELD 指定的变量组里。
- 文件编号将要从中读取数据的文件的代号。该编号必须与使用FOPEN指令打开的文件的编号相同。
- 记录代号指要首先读取文件中的哪个记录。在这种情况下，使用“记录代号”声明的FIELD中的变量组在使用时作为一个整体。当从文件的开头读取时，记录代号为1。

■ **相关项目**　　FOPEN, FIELD, FCLOSE, FPUT

■ **程序实例**

```
conf
   field  5
      global no%
      global moji1$ , moji2$
   end field
   fopen  “MEMORY”, 2 , 5
   ......
end conf
evnt
   FGET 5 , 3
   numdsp ..NUM000 , no%
   strdsp ..STR000 , moji1$
   strdsp ..STR001 , moji2$
   fclose 5
end evnt
```

FIELD ... END FIELD

指令

■ **功能**　　FIELD ... END FIELD 文件的读写单位。

■ **格式**

```
FIELD 文件编号
变量清单
变量清单
END FIELD
```

■ **使用范例**

```
FIELD 5
global abcd , xyz%
static dddd(10,10)
backup moji$
END FIELD
```

■ **说明**

- FIELD ... END FIELD 用来声明读写单元。在声明后，可以使用FGET指令来读文件，使用FPUT指令来写文件。
- 文件编号指出变量清单所列出的变量将要读出/写入的目标。文件编号必须同使用FOPEN指令打开的文件编号相同。其编号在1～16之间。
- 在FIELD~END FIELD之间可以被写入的变量只能是GLOBAL、STATIC、或者BUCKUP型变量。变量清单的声明方法同BLOBAL、STATIC、BUCKUP变量的声明方法一样。
- 在使用FOPEN指令打开的程序里声明的FIELD块是默认得读写单元。
- 当文件正在被非使用FOPEN打开的部品读写时，使用该部品程序里的FIELD块。当在该部品里没有声明FIELD时，则使用默认得FIELD块。
- 如果在一个程序写有两个或以上的FIELD，则只有最后面那个有效。
- FIELD ... END FIELD 不能写在全局画面程序里。

■ **相关项目**　　FOPEN, FCLOSE, FPUT, FGET

■ **程序实例**

```
conf
   field  5
      global no%
      global moji1$ , moji2$
   end field
   fopen  “MEMORY”, 2 , 5
   ......
end conf
evnt
```

```
    no% = 1
    moji1$ = “product-name”
    moji2$ = “product-number”
    fput 5 , 3
    fclose 5
end evnt
```

FIGCOLOR

指令

■ **功能**　　FIGCOLOR 用于改变图形显示的填充模式和颜色。

■ **格式**　　FIGCOLOR 控件名, 填充, 显示颜色, 背景颜色

■ **使用范例**　　FIGCOLOR .B000.FIG000, 1, 2, 3

■ **说明**

- FIGCOLOR用于改变图形显示的填充模式和颜色。
- 控件名指能代表图形显示器控件的控件名或ID变量。
- 填充指填充模式的数字代号，范围是0～15。
- 显示颜色是指图形显示部分的填充颜色代号，范围0～15。
- 背景颜色是指填充部分的背景颜色代号，范围也是0～15。

■ **相关项目**　　FIGDSP

■ **程序实例**

```
evnt
   input type%, id@, tile%
   FIGCOLOR ..FIG000, tile%, -1, -1
end evnt
```

FIGDSP

指令

■ **功能**　　FIGDSP 用于指定在图形（构件）显示器（控件）中显示的构件（图形）。

■ **格式**　　FIGDSP 控件名, 构件名

■ **使用范例**　　FIGDSP .B000.FIG000, SWFIG

■ **说明**

- FIGDSP 用于在构件显示器中显示指定的图形构件，构件图形必须预先制作好。
- 控件名指能代表图形显示器的控件名称或ID变量。
- 构件名指要在图形显示器中显示的构件的名称或ID变量，也可以是构件的注册号（或登记号，必须是一个整型数）。
- 当控件参数设置为有效时，这里使用的指令就不起作用。

■ **相关项目**　　FIGCOLOR, FIGFORM

■ **程序实例**

```
evnt
   input ty , id@, figno%
   FIGDSP ..FIG000 , figno%
end evnt
```

FIGFORM

指令

■ **功能**　　FIGFORM 用于改变构件显示的显示格式。

■ **格式**　　FIGFORM 控件名称, 调整参数

■ **使用范例**　　FIGFORM ..HYOJIKI, 0

■ **说明**

- 当构件显示器与图形构件的尺寸不一样时，FIGFORM指令用于指定是否采用调整（放大或缩小）的方式使其相匹配。当采用调整方式时，图形大小将自动与图形显示器相匹配。
- 控件名指图形显示器的控件名称或控件的ID变量名。
- 调整参数指是否要对显示时的构件大小进行大小调整的代号。
  0: 不进行大小调整;
  1: 进行大小调整。

■ **相关项目**　　FIGCOLOR, FIGDSP

■ **程序实例**

```
evnt
   input ty , id@, data
   if ty = 3 and data = 1 then
      FIGFORM ..FIG000, 1
   else
      FIGFORM ..FIG000, 2
   endif
   figdsp ..FIG000, figno
end evnt
```

FINPUT

指令

■ **功能** FINPUT 从指定的文件中读取数据。

■ **格式** FINPUT 文件号, 变量, 变量, ...

■ **使用范例** FINPUT 12, VAR% , STRING$

■ **说明**

- FINPUT 将文件号指定的文件读到指定的变量。
- 变量可以是数值型或字符串型。
- 在将数据读入到变量时，可能下列分隔符，它们没有包括在变量里：
  - 只有“,”和回车符（CR）可以用作分隔符。在回车符（CR）后面的空档键（LF）将被忽略。
  - 当指定的是数字变量时，空格也可以用来作分隔符
  - 当指定字符串变量时，只读取引号（“”）中的内容。
  - 当需要写入的数据类型与读取的数据类型不一致时，则变量内容将不能预测。
- 文件号必须同使用FOPEN指令打开的文件号相同。

■ **相关项目** FOPEN, FCLOSE, FPRINT, FWRITE, LINPUT

■ **程序实例**

```
conf
   fopen  “C:TEST”, 2 , 5
end conf
evnt
   var% = -2
   fwrite 5, 123, var%, “ABCD”, “XYZ”
   fseek(5, 0, 0)
   finput 5, VAR1%, VAR2%, VSTR1$, VSTR2$
end evnt
```

要写入指定文件的数据如下：

123,-2,”ABCD”,”XYZ” CR/LF

在数据读入后，变量的内容如下：

VAR1%	123	VSTR1$	ABCD
VAR2%	-2	VSTR2$	XYZ

FLUSH

指令

■ **功能**　　FLUSH 指令将无协议通信接收缓冲区地址返回给变量的开始。

■ **格式**　　FLUSH 端口

■ **使用范例**　　FLUSH 2

■ **说明**

- FLUSH 使接收到的数据能够被写到变量的开始位置，无协议通信接收缓冲区的写入地址已经返回给该变量。
- 表示端口CH1～CH3的编号分别为1～3。
- 在接收到消息接收完成之后，执行本指令。如果不执行本指令，数据接收缓冲区将会写满。它除了返回地址外，并不清除缓冲区的数据。
- 使用该指令的端口，必须预先用OPENSIO指令打开。OPENSIO 指令将在后面介绍。

■ **相关项目**　　OPENSIO, CLOSESIO, WRITESIO, WRITWSIOB, SETSIO

■ **程序实例**

```
conf
   global buf$ * 200
   opensio 2 , 1 , buf$
   setsio  2 , &HD
end conf
evnt
   strdsp ..STR000 , buf$
   FLUSH 2
   closesio 2
end evnt
```

FOPEN

指令

■ **功能**　　FOPEN 指令用来打开指定的文件。

■ **格式**　　FOPEN 文件名，属性，文件号

■ **使用范例**　　FOPEN “MEMORY”, 2, 5

■ **说明**

- FOPEN 指令用来打开需要进行读写的文件。
- 文件名指要打开的目标文件，以括号中文件名为文件名的文件将被打开，括号中文件最多只能有8个字符。当以MEMORY为文件名时，内部存储器作文件处理。
- 属性有如下三种：
  0: 只读
  1: 只写
  2: 可读写
  当文件名为 “MEMORY”时，不管文件属性如何都将被打开。
- 文件号在在文件读写时使用，文件号可以使用1～16的常数表示，也可以是一个变量。
- 要将内部存储器作文件处理，该存储器的容量必须预先在系统模式中设置好。
- 对未进行格式化的文件使用本指令将会导致系统错误。

■ **相关项目**　　FCLOSE, FIELD, FPUT, FGET, FORMAT

■ **程序实例**

```
conf
   field  5
      global no%
      global moji1$ , moji2$
   end field
   FOPEN  “MEMORY”, 2 , 5
   ......
end conf
evnt
   ......
end evnt
```

FOR ... TO ... NEXT

指令t

■ **功能**　　根据计数值循环执行FOR~NEXT之间的指令内容。

■ **格式**　　FOR 变量名 = 起始值 TO 最终值 [单步递增] ...
NEXT

■ **使用范例**

```
FOR I = 1 TO 10
A(I) = 3
NEXT
```

■ **说明**

- FOR后面的变量名指定用来进行循环次数计算，即循环执行FOR~NEXT之间程序的次数。变量名必须是整型或浮点型变量。
- 起始值既初始值。每执行一次FOR~NEXT之间的程序，变量的值增加一次，（注意，增量不可以是负数）。当增加后的变量值超出最终值时，才执行NEXT后面的内容。
- FOR ～ NEXT 指令可以进行嵌套。

■ **相关项目**　　WHILE ... WEND, SELECT CASE

■ **程序实例**

```
conf
   static VAR%(10)
   for i% = 0 to 10
      VAR%(i%) = i% * 3
   next
end conf
```

FORMAT

指令

■ **功能**　　FORMAT 将指定的文件初始化（格式化）

■ **格式**　　FORMAT 文件名

■ **使用范例**　　FORMAT “A:”

■ **说明**

- 文件名只要进行格式化的文件名称。
- "A:", "E:" 或 "MEMORY" 都可以作为驱动器名称。
- 当使用 “MEMORY”作为驱动器名称时，文件的内容都是0。
- 在首次使用文件时，一定要使用FORMAT指令。

■ **相关项目**　　FOPEN, FIELD, FCLOSE, FPUT, FGET

■ **程序实例**

```
conf
   field  5
      global no%
      global moji1$ , moji2$
   end field
   global sum%
   FORMAT  “MEMORY”
   fopen   “MEMORY”, 2 , 5
   ......
end conf
evnt
   no% = 1
   moji1$ =  “product-name”
   moji2$ =  “product-number”
   fput 5 , 3
   fclose 5
end evnt
```

FPRINT

指令

■ **功能**　　FPRINT 将数据写到指定的文件。

■ **格式**　　FPRINT 文件编号，表达式1，表达式2 ，...

■ **使用范例**　　FPRINT 12, 100, “ABCD”, VAR%, STRING$

■ **说明**

- FPRINT 指令将表达式里定义的数值、变量、或字符写到文件号指定的文件里。
- 在表达式可以指定数值、字符、数字或字符变量。
- 数字表达式被转换成数字字符串之后，被写到指定的文件，当被写的数值为正时，数值前面为空；当被写入的数值为负时，在其前面写入“-”号。在数字之后，也为空白。
- 在写入字符串时，不插入分隔符。
- 文件编号必须与使用FOPEN指令打开的文件编号相同。

■ **相关项目**　　FOPEN, FCLOSE, FPUT, WRITE

■ **程序实例**

```
conf
   fopen  “C:TEST”, 2 , 5
end conf
evnt
   var% = -2
   fprint 5, 123, 45, var%, “ABCD”, “XYZ”
end evnt
```

数据以如下格式写到指定的文件：

△123△△45△-2△ABCDXYZ（△ 表示空白）

FPUT

指令

■ **功能**　　FPUT 指令将数据写到指定的文件

■ **格式**　　FPUT 文件编号，数据记录号

■ **使用范例**　　FPUT 5, 3

■ **说明**

- FPUT 指令将FIELD...END FIELD中定义的变量群的内容写到文件中的数据记录中！
- 文件编号只目标文件的编号。该编号必须与使用FOPEN指令打开的文件编号相同。
- 记录号用来指出数据将被写到文件中的哪条记录中。在这种情况下，FIELD中的变量群作为一个整体使用。文件中的第一条记录为1。

■ **相关项目**　　FOPEN, FIELD, FCLOSE, FGET

■ **程序实例**

```
conf
   field  5
      global no%
      global moji1$ , moji2$
   end field
   fopen   “MEMORY”, 2 , 5
   ......
end conf
evnt
   no% = 1
   moji1$ =  “product-name”
   moji2$ =  “product-number”
   FPUT 5 , 3
   fclose 5
end evnt
```

FRECOLOR

指令

■ **功能**　　FRECOLOR 指令用于改变自由图显示的颜色和填充模式。

■ **格式**　　FRECOLOR 控件名, 填充-1, 显示颜色-1, 背景颜色-1, 填充-2, 填充颜色-2, 背景颜色-2

■ **使用范例**　　FRECOLOR ..FRE000, 2, 1, 4, 5, 2, 1

■ **说明**

- FRECOLOR 指令用于改变自由图显示（free graph display）的填充模式和颜色，及背景部分的填充模式和颜色。
- 控件名指自由图控件的名称或能代表它的ID变量名。
- 填充-1 表示显示部分的填充模式数字代号，范围在0～15!
- 显示颜色-1 表示填充颜色的数字代号，范围在0～15 。
- 背景颜色-1 表示自由图显示部分背景颜色的数字代号，范围0～15!
- 填充-2 表示自由图背景部分填充模式的数字代号，范围0～15!
- 显示颜色-2 指背景部分的填充颜色数代号，在0～15之间。
- 背景颜色-2 表示背景部分的背景颜色代号，在0～15之间。

■ **相关项目**　　FREDSP

■ **程序实例**

```
conf
   static name@
   name@ = ..FRE000
end conf
evnt
   input type%, id@, data%
   if type% = 3 then
      FRECOLOR name@, 2, 3, 1, 4, 5, 2
   endif
end evnt
```

FREDSP
指令

■ **功能**　　FREDSP 指令用于：在自由图显示中显示某个数值。

■ **格式**　　FREDSP 控件名, 显示值

■ **使用范例**　　FREDSP .B000.FRE000, 50

■ **说明**

- FREDSP 功能是在自由图显示中显示指定的数值。
- 控件名指自由图显示器的名称或能代表它的ID变量名称。
- 显示值用于指定自由图显示的填充范围。
- 当控件的参数在内部设置为有效时，在这里设定的显示值将不起作用。

■ **相关项目**　　FRECOLOR

■ **程序实例**

```
conf
   static name@
   name@ = ..FRE000
end conf
evnt
   input type%, id@, data%
   FREDSP name@, data%
end evnt
```

FSEEK

函数

■ **功能**　　FSEEK 用于改变读出/写入文件的位置。

■ **格式**　　FSEEK (文件编号, 参考位置, 偏移量)

■ **使用范例**　　AAA% = FSEEK (12, 0, 0)

■ **说明**

- FSEEK 函数将读写位置移到从参考位置开始，以偏移量为距离的地方。
- 文件编号指使用FOPEN指令打开的文件。
- 参考位置可以是0、1和2。当参考位置为0时，将读写位置移到以以文件首位置开始的位置。当参考位置为1时，移到从当前位置开始的位置；当为2时，移到文件的最后位置。
- 偏移量以字节为单位，当将读写位置移到文件结束位置时，应指定正的偏移量。
- FSEEK函数的返回值为一个读写位置。

■ **相关项目**　　FOPEN, FCLOSE, FPRINT, FWRITE, FINPUT

■ **程序实例**

```
conf
   fopen  “C:TEST”, 2 , 5
end conf
evnt
   AAA$ = “12345”
   fwrite 5, AAA$, “ABCD”
   fseek(5, 0, 0)
   finput 5, VSTR$
end evnt
```

FSUM

函数

■ **功能**　　FSUM 函数用来计算指定区域变量群的和。

■ **格式**　　FSUM (文件号)

■ **使用范例**　　SUM = FSUM (5)

■ **说明**

- FSUM 函数用于计算由文件号指出的FIELD里各变量低8位的和。该函数计算的区域应该是在字符串变量中没有字符代码被定义为0。
- FSUM 函数返回一个范围在0～255之间的整数值。
- 如果由文件编号指出的的FIELD不存在，则会报错。

■ **相关项目**　　FOPEN, FIELD, FCLOSE, FPUT, FGET

■ **程序实例**

```
conf
   field  5
      global no%
      global moji1$ , moji2$
   end field
   global sum%
   fopen  “MEMORY”, 2 , 5
   ......
end conf
evnt
   fget 5 , 3
   if sum% = FSUM(5) then
      numdsp ..NUM000 , no%
      strdsp ..STR000 , moji1$
      strdsp ..STR001 , moji2$
   else
      strdsp ..STR002 , “SUM-error”
   fclose 5
end evnt
```

FUNCTION ... END FUNCTION

指令

■ **功能**　　FUNCTION ... END FUNCTION 用于声明对函数块

■ **格式**

```
FUNCTION 函数名[类型声明] 变量1[, 变量2] ...)
...
END FUNCTION
```

■ **使用范例**

```
FUNCTION ADD%(A%,B%)
ADD%=A%+B%
END FUNCTION
```

■ **说明**

- FUNCTION ... END FUNCTION 用于在函数所在的地方进行函数块声明。
- 根据函数声明的地方，定义的函数有如下三种调用方式：
  - 局部函数：在局部画面而非全局画面函数程序中定义。
  - 全局函数：在全局还面函数中定义。
  - 库函数：在函数库中定义。
- 声明的函数类型（函数原型）必须与其本身的类型一致。
- 同变量一样，函数返回一个值，该值的类型由函数类型声明字符（如$，%或!）决定。
  没有类型声明的函数默认为实型函数。
- 函数返回值后将取代函数（包括类型声明字符）所占的位置。
- 变量是指函数的变量参数。
  当变量参数没有声明类型时，默认为实型变量。
- 参数在函数被调用时给出。因此，如果在函数中的参数变量值变化时，以参数形式给出的变量其本身也将发生变化。
- 在需要调用函数块声明的函数时，需要要使用DECLARE进行类型声明。
- 要强行退出函数的执行，可以使用EXIT 功能指令.
- 这也是K-basic语言的又一特性。

■ **相关项目**　　DECLARE, EXIT FUNCTION, FUNCTIONCHECK

■ **程序实例**

```
declare my add%(a%,b%)
conf
   global x%,y%
   local sum%
   sum% = my add(x%,y%)
end conf
FUNCTION my add%(a%,b%)
   my add%=a%+b%
END FUNCTION
```

FWRITE

指令

■ **功能**　　FWRITE 指令用于将数据写到指定的文件。

■ **格式**　　FWRITE 文件号, 表达式1, 表达式2, ...

■ **使用范例**　　FWRITE 12, 100, “ABCD”, VAR%, STRING$

■ **说明**

- FWRITE 指令将表达式里定义的数值或字符写到文件号指定的文件里面。
- 表达式可以是数值、字符，或数字或字符变量。
- 当要写入两个或两个以上的表达式时，中间应该用逗号（，）格开，在表达式的最后加回车符（carriage return简称CR）或空格（line feed 简称LF）。
- 数学表达式被转换成数字字符串然后写入指定文件，当数值为负时，在其前面加“－”。
- 当写入字符串常数时，使用引号（“”）指定。
- 文件号必须同使用FOPEN指令打开的文件编号一致。

■ **相关项目**　　FOPEN, FCLOSE, FPUT, FPRINT

■ **程序实例**

```
conf
   fopen  “C:TEST”, 2 , 5
end conf
evnt
   var% = -2
   fwrite 5, 123, var%, “ABCD”, “XYZ”
end evnt
```

被写入指定文件的数据如下：

123,-2,”ABCD”,”XYZ” CR/LF

GETBLIGHT

指令

■ **功能** GETBLIGHT 指令用于读取背光灯从点亮到熄灭的持续时间。

■ **格式** GETBLIGHT 变量名

■ **使用范例** GETBLIGHT VAR

■ **说明** 变量名指要将背光灯点亮的持续时间赋给的目标变量名。时间单位为分，当值为0时，表示背光灯还没有关闭。

■ **相关项目** SETBLIGHT

■ **程序实例**

```
conf
   GETBLIGHT var
   var = var*2
   setblight var
end conf
```

GETDATE

指令

■ **功能**　　GETDATE 用来读取系统的日期值。.

■ **格式**　　GETDATE 年变量, 月变量, 日变量, 星期变量

■ **使用范例**　　GETDATE YEAR%, MONTH%, DATE%, DAY%

■ **说明**

- GETDATE指令将当前日期读到年变量, 月变量, 日变量, 星期变量四个变量里面。
- 年，读取的是公历年的后两位；月，范围从0～12；日期值可以是从0到31；星期，从周日到周六分别是0~6。
- 注意：这里的变量必须是整型变量。
- 在带有停电记忆日期保持芯片的触摸屏型号里，一旦使用了setdate指令，即使在断电的情况下其日期值也将自动更新。本功能在现在推出的GC系列触摸屏都有，只有早期GC-53LM/LC不能。

■ **相关项目**　　DATE$, GETTIME, SETDATE, SETTIME, TIME$

■ **程序实例**

```
conf
   GETDATE yr%, mt%, d%, dd%
   numdsp ..NUM000, yr%
   numdsp ..NUM001, mt%
   numdsp ..NUM002, d%
end conf
```

GETGID

函数

■ **功能**　　GETGID 函数用于读取当前局部画面的ID号，即画面名称。

■ **格式**　　GETGID()

■ **使用范例**　　VAR@ = GETGID()

■ **说明**

- GETGID 函数读取当前局部画面的ID名称。
- 当然本函数不能用于读取全局画面的ID名称。因为全局画面是一直都显示的。

■ **相关项目**　　GETGNO

■ **程序实例**

```
evnt
   input type%, id@, data%
   if type% = 3 then
      VAR@ = GETGID()
      NO% = GETGNO(VAR@)
      00~D100 = NO%
   end if
end evnt
```

GETGNO

函数

■ **功能**　　GETGNO 用于读取当前显示画面的注册登记号。

■ **格式**　　GETGNO (画面ID值)

■ **使用范例**　　NO = GETGNO (ID@)

■ **说明**

- GETGNO 函数用于读取由“画面ID值”所给出画面的注册登记号。
- 画面ID值可以是画面的名称或能代表它的ID型变量。

■ **相关项目**　　GETGID

■ **程序实例**

```
evnt
   input type%, id@, data%
   if type% = 3 then
      VAR@ = GETGID()
      NO% = GETGNO(VAR@)
      00~D100 = NO%
   end if
end evnt
```

GETID

函数

■ **功能**　　GETID 函数根据离参考点偏移量，读取（算出）ID值。

■ **格式**　　GETID (参照目标ID值, 偏移量)

■ **使用范例**　　ID@ = GETID (VARID@, 10)

■ **说明**

- GETID 函数用于读取离“参照目标ID值”距离为“偏移量”的目标的ID值。
- 参照目标ID值可以是一个ID变量、画面名称、部品名称、注册文本名称或设备名称。
- 偏移量指“参考目标ID值”到“要读取的ID号”的距离。当偏移量为0时，获得的是参照目标ID值本身。

■ **相关项目**　　GETOFFSET

■ **程序实例**

```
conf
   cyclic 00~d0001 * 30
end conf
evnt
   input ty%,id@,dat%
   offset = getoffset(00~d0001,id@)
     ...
     根据偏移量值进行错误处理。
     ...
   id@ = getid (00~d0001,offset)
     ....
end evnt
```

GETKEY

函数

■ **功能**　　向OS传送键盘键码

■ **格式**　　GETKEY（keyCode）

■ **使用范例**　　GETKEY（KeyCode%）

■ **说明**

- 键盘键码是指键盘按键对应的特定码值，可用于模拟键盘按钮输入。对应键码可参照微软公司提供的键码表。
- 延迟时间为固定50ms、超过50ms的场合请使用GETKEYT。
- 返回值为发送键码对应的字符串。

■ **相关项目**　　GETKEYT,SETIMEMODE,S2U,U2S

■ **程序实例**

```
in char% = KeyCode%
if in char% <> 0 then
    bksp$ = chr$(8)
    if in char% = &HD then
        dspchar$ = dspchar$ +  getkey(in char%)
    else
        if ShiftKeyDown% = 1 then
            KeyCode% = &H10
            getkey(KeyCode%)
        endif
        key$    =  getkey(in char%)
        if ShiftKeyDown% = 1 then
            KeyCode% = &H10
            getkey(KeyCode%)
        endif
    end if
end if
```

GETKEYT

函数

■ **功能**　　向OS传送键盘键码

■ **格式**　　GETKEYT（keyCode）

■ **使用范例**　　GETKEYT（KeyCode%）

■ **说明**

- 键盘键码是指键盘按键对应的特定码值，可用于模拟键盘按钮输入。对应键码可参照微软公司提供的键码表。
- 等待时间可设定范围：50~1000 ms。
- 返回值为发送键码对应的字符串。

■ **相关项目**　　GETKEY,SETIMEMODE,S2U,U2S

■ **程序实例**

```
in char% = KeyCode%
if in char% <> 0 then
    bksp$ = chr$(8)
    if in char% = &HD then
        dspchar$ = dspchar$ +  getkeyt(in char%, 50)
    else
        if ShiftKeyDown% = 1 then
            KeyCode% = &H10
            getkeyt(KeyCode%, 50)
        endif
        key$    =  getkeyt(in char%, 50)
        if ShiftKeyDown% = 1 then
            KeyCode% = &H10
            getkeyt(KeyCode%, 50)
        endif
    end if
end if
```

GETLANG

函数

■ **功能**　　读取GC-A2当前设定的语言编号。

■ **格式**　　GETLANG (存放语言号的变量)

■ **使用范例**　　GETLANG (var%)

■ **说明**

- 读取GC-A2当前设定的语言编号。
- 返回的语言编号对应数值为：1~4整型数

■ **相关项目**　　SETLANG

■ **程序实例**

```
if type% = 3 then
    if id@ = ..SWT000 then
        if data% = 1 then
            setlang(1)
        elseif data% = 0 then
            getlang(lang%)
        end if
    end if
end if
```

GETOFFSET

函数

■ **功能** GETOFFSET 函数用于计算参照目标ID到指定ID的距离。

■ **格式** GETOFFSET (参照目标-ID, 指定ID)

■ **使用范例** OFFSET% = GETOFFSET (00~D0001, ID@)

■ **说明**

- GETOFFSET 函数用于计算从参照目标ID到指定ID之间的距离。
- 参照目标-ID可以是一个ID变量、画面名称、部品名称、注册文本名称或设备名称。
- 指定ID 也可以是一个ID变量、画面名称、部品名称、注册文本名称或设备名称。
- 当GETOFFSET 函数用于由fCYCLIC2声明的设备时，偏移量应该是2的倍数。

■ **相关项目** GETID

■ **程序实例**

```
conf
   cyclic 00~d0001 * 30
end conf
evnt
   input ty%,id@,dat%
   offset = GETOFFSET(00~d0001,id@)
     ...
     根据偏移量值进行错误处理。
     ...
   id@ = getid (00~d0001,offset)
     ....
end evnt
```

GETTIME

指令

■ **功能**　　GETTIME 指令用于读取时间数据。

■ **格式**　　GETTIME 时钟变量, 分钟变量, 秒钟变量

■ **使用范例**　　GETTIME HOUR%, MIN%, SEC%

■ **说明**

- GETTIME 指令将当前的时刻数据写到指定的时钟变量、分钟变量、秒钟变量。
- 时钟值为0～23；分钟值为0～59；秒钟值为0～59。
- 三个变量必须都是整型变量。
- 在带有停电记忆日期保持芯片的触摸屏型号里，一旦使用了setdate指令，即使在断电的情况下其日期值也将自动更新。本功能在现在推出的GC系列触摸屏都有，只有早期GC-53LM/LC不能。同时，时钟也一样，不带天电记忆芯片的触摸屏其时钟也不能体电记忆，重新上电后其值初始化为00：00：00。

■ **相关项目**　　DATE$, GETDATE, SETDATE, SETTIME, TIME$

■ **程序实例**

```
conf
   GETTIME H%,M%,S%
   numdsp ..NUM000, H%
   numdsp ..NUM001, M%
   numdsp ..NUM002, S%
end conf
```

GLOBAL

指令

■ **功能**　　GLOBAL 用于声明全局变量。

■ **格式**　　GLOBAL 变量名称 [, 变量名...]

■ **使用范例**　　GLOBAL VAR, XYZ(2,3), MOJI$ * 20

■ **说明**

- GLOBAL用于声明全局变量。全局变量能从所有的程序里读取数据。也能向任意程序中写入数据。全局必须在使用前用Global对其进行声明。这种变量在每次上电时进行初始化，在上电期间其值具有自保功能。
- 一个普通变量，数组变量，字符串变量都可以定义成全局变量。
- 当定义数组或字符串变量时，不需要使用DIM和STRING对其进行声明。而对于非全局变量则需要使用。

■ **相关项目**　　AUTO, BACKUP, DIM, LOCAL, STATIC, STRING

■ **程序实例**

```
conf
   GLOBAL var%, float
   GLOBAL moji$ * 50
   GLOBAL xyz@(10,10)
end conf
```

GOSUB

指令

■ **功能**　　GOSUB 用于执行（或调用）指定的子程序。

■ **格式**　　GOSUB 子程序名

■ **使用范例**　　GOSUB SUB001

■ **说明**

- 在执行该指令后，主程序将控制权交给子程序。
- 子程序可以在全局画面程序里，也可以是在包括GOSUB指令的程序里。使用RETURE指令子程序将控制权交还给主程序。
- 当局部子程序和全局子程序同名时，将调用全局子程序。

■ **相关项目**　　RETURN

■ **程序实例**

```
evnt
   X = 10
   GOSUB SUB001
   numdsp ..NUM000, X
end evnt
SUB001:
   ab = X+3
   RETURN
```

GOTO

指令

■ **功能**　　执行GOTO 指令，控制无条件跳至指定的程序行。

■ **格式**　　GOTO 标签名

■ **使用范例**　　GOTO LABEL1

■ **说明**　　执行GOTO 指令，控制无条件跳至由标签名指定的程序行，然后程序由此往下执行。

■ **相关项目**　　无

■ **程序实例**

```
evnt
   if a = 1 then goto L1
   a = 3
 L1: numdsp ..NUM000 , a
end evnt
```

HEX$

函数

■ **功能** HEX$ 将十进制数字转变成十六进制值。

■ **格式** HEX$ (数学表达式)

■ **使用范例** HEX$ (123)

■ **说明**

- HEX$ 将一个十进制数值转换成一个十六进制数值。
- 当数学表达式为浮点型时，转换时先将其转换成整数，然后再将其转化成十六进制数。
- 转换时十进制数值必须在-2147483648 ～ 2147483647之间。

■ **相关项目** OCT$, VAL

■ **程序实例**

```
evnt
   input type , id@ , data
   moji$ = HEX$(data)
   strdsp ..STR000, moji$
end evnt
```

IF ... THEN ... ELSE

指令

■ **功能**　　先进行条件判断，然后选择下一步将要执行的程序。

■ **格式**

```
IF 条件表达式 THEN 程序1 [ELSE 程序2]
IF 条件表达式1 THEN
     程序1
[ELSEIF 条件表达式2 THEN
     程序2]
[ELSE
     程序3]
END IF
```

■ **使用范例**

```
IF TYPE% = 1 THEN VALUE = 10
```

■ **说明**

- 条件表达式是一个关系运算表达式，当其为真时值为1，否则值为0。
- 当运算结果为真时，执行then后面的程序；当结果为假时，则执行else后面的程序内容。
- ELSE, ELSEIF 及其后面的内容也可以省略。
- 在IF THEN...END IF 之间最多可以使用50个ELSEIF。

■ **相关项目**　　无

■ **程序实例**

```
evnt
   if a = 2 then x = 3
   if x = 5 then
      a = 1
   elseif x = 6 then
      a = 2
   else
      a = 3
   end if
end evnt
```

INIT ... END INIT

声明

■ **功能**　　INIT ... END INIT 用于声明初始化程序块。

■ **格式**

```
INIT
.....
....
END INIT
```

■ **使用范例**

```
INIT
static VAR\%
END INIT
```

■ **说明**

- 本程序块仅在本段程序所在的画面（或部品）程序执行的开始时执行一次。
- 编程时将需要首先处理（如初始化之类的处理）的内容写在本部分。

■ **相关项目**　　CONF .... END CONF,EVNT .... END EVNT

■ **程序实例**

```
INIT
   global moji$
   moji$="initial value"
END INIT
```

INP

函数

■ **功能** 从指定的并行I/O端口读取两字节的数据。

■ **格式** INP (端口号)

■ **使用范例** VAR = INP (0)

■ **说明**

- 本指令用于从指定的并行I/O端口读取数据。
- 指定的端口号取决于插入总线的板子类型，可以是0～3间的一个值。

■ **相关项目** OUT

■ **程序实例**

```
evnt
   var% = inp (0)
   if (var% and 1) = 1 then var% = 0
   OUT 0,var%
end evnt
```

INPBIT

函数

■ **功能**　　INPBIT函数用于从指定的输入端口读取指定位的值。

■ **格式**　　INPBIT (端口号, 位编号)

■ **使用范例**　　DATA% = INPBIT (0,10)

■ **说明**

- INPBIT函数用于从指定的输入端口读取指定位的值。
- 端口号和位编号都是一个整数值。
- 端口的最低位编号为0，一次为1、2 ……，即为连续递增的整数，范围为0～31。
- 如果是不存在地端口或不存在的位，将返回一个0。

■ **相关项目**　　INP, OUT, OUTBIT, OUTBITSTAT, OUTSTAT

■ **程序实例**

```
evnt
   data% = INPBIT(0,3)
   if data% = 0 then
      outbit 0,3,1
   endif
end evnt
```

INPUT

指令

■ **功能**　　INPUT 将发送给画面或部品的数据读入到指定的变量。

■ **格式**　　INPUT 变量名1 [, 变量名2] .....

■ **使用范例**　　INPUT　V1, ID@, DATA

■ **说明**

- INPUT 读取发送给画面或部品的数据。
- 表示数据发送者类型的整型数（或变量）作为第一个变量；表示发送者身份（数据是谁发送的）的ID为第二个变量；后面就是真正接收到的数据。

发送者	类型	ID	数据（data）的内容
Screen（画面）	1	画面名	用PRINT指令发出的内容。
Part（部品）	2	部品名	用PRINT指令发出的内容。
Switch (单)	3	开关控件名	ON 为 1, OFF 为 0 。
Switch (多)	3		开关编号: ON为1, OFF为0
Selector switch（选择开关）	3		处于 ON 状态的开关编号。
Timer（定时器）	4	由opentim打开的定时器号	表示 ON (1) 或 OFF (0) 状态的数值1或0。
Alarm（报警）	5	由SETALARM打开的定时器号	表示状态ON的数值1。
Non-procedual（无协议通信）	7		端口号，状态，和按此顺序接受到的字节个数。
Sampling（采样）	9	控件名	采样值
PLC	16	PLC内部单元名称或存储器表	单元内的数值
Host（上位计算机）	22	HST	数据由用户决定

- 当在进行采样的曲线图、棒形图、等上进行数据显示时，如果数据要发送给K-basic程序，则要使用数据发送者“sampling”。这时，ID为进行采样的控件名，数据为输出的采样值。
- 来自触摸屏内部存储器表（00～mtbl（×××））和PLC单元的消息，类型为16。
- 当数据由无协议通信送来时，端口号为1～2；状态为0表示正常接收，1表示缓冲区满，-1表示通信出错；接收到的字节数即为无协议通信接受到的数据字节总量（即写入数据接收缓冲区的数据量）。对于文本模式，还要读取结束码，当状态为1或－1时，表示正在读取接收到的数据量值。
- 多开关或选择开关的编号从左到右（从上倒到下）依次为1、2、3……。

■ **相关项目**　　PRINT, CYCLIC, OPENPARALLEL, OPENCOM, OPENSIO

■ **程序实例**

```
conf
  global  buffer$
   opensio 2 , 0, buffer$
   setsio  2, 10
end conf
evnt
   input type, id@, port%, status%, bytes%
   if type = 7 then
     moji$ = left(buffer$, bytes% - 1)
     strdsp ..STR000 , moji$
   end if
end evnt
```

INSTR

函数

■ **功能**　　INSTR 函数从指定的字符串中查找指定的字符串。当找到了指定的字符串时，该函数将字符串的起始位置告知系统。

■ 格式　　INSTR（起始位置，查找对象字符串，查找目标字符串）

■ 使用范例　　A = INSTR（10. MOJI1$, MOJI2$）

■ 说明

- INSTR函数从指定的字符串中查找指定的字符串。当找到了指定的字符串时，该函数将字符串的起始位置告知系统。如果没有找到，则返回值为0。
- 当在起始位置为1时，表示从字符串的开始位置起开始查找。
- 查找的字符串对象可以是字符串变量、直接字符串、注册文本名称或文本注册号。

■ 相关项目　　None

■ 程序实例

```
evnt
   a$ = “this is oip.”
   p = instr（1, a$, “company”）     ’   查找对象为字符串变量
   p = instr（1, num,”ab”）      ’   查找对象为文本注册号
end evnt
```

INT

函数

■ **功能**　　INT 对数学表达式取整，从而将实数变为整数。

■ 格式　　INT (数学表达式)

■ 使用范例　　A = INT (30.1)

■ 说明

- INT 对其后面括号内的数学表达式向小的方向取整，从而将实数变为整数。
- 当表达式为负时，也是向下取整。方法如下：
  INT (1.4) → 1
  INT (-1.4) → -2

■ 相关项目　　CINT

■ 程序实例

```
evnt
   input type%, id@, data
   intvar% = INT ( data )
   numdsp ..NUM000, intvar%
end evnt
```

INTERLOCK

指令

■ **功能**　　INTERLOCK 指令控制向系统模式的转换。

■ 格式　　INTERLOCK 模式代号

■ 使用范例　　INTERLOCK 1

■ 说明

- 当模式为1时， INTERLOCK 指令将互锁设置为ON；当模式为0时，INTERLOCK 指令将互锁设置为OFF。
- 当INTERLOCK为ON时，即使是按住左上角和右下角也不能进入系统模式。在上电时按住左上顶角也进入不了系统模式。
- 在互锁激活（INTERLOCK 模式为1）后，必须用程序对其进行复位。所以编程时一定要有能将其安全复位的程序。
- 在带有停电保持的触摸屏内部INTERLOCK能在系统断电后进行停电保持。因此，这种模式设置通常在每次上电时执行的程序里进行。

■ 相关项目　　无

■ 程序实例

```
conf
   INTERCLOCK 1
   end conf
evnt
   input tp%,id@,dat%
   if id@ = ..sw then interlock 0
   .....
end evnt
```

IOCTL

指令

■ **功能** IOCTL 用于控制与OIP连接的I/O设备。s

■ 格式 IOCTL I/O-type, mode (模式)

■ 使用范例 IOCTL 0, 0

■ 说明

- I/O-type指需要控制的I/O设备，在这里I/O设备只能为PLC、开关或无协议通信缓冲区。
- mode 指代表I/O设备如何被控制的整数值。
- 当控制PLC时，mode使用如下其中一个值代替，I/O-type为0。
  0: 可以对PLC进行读写。
  1: 禁止对PLC进行读写。
  - 当在禁止读写时对PLC进行读写操作，则会出现错误。
- 开关的控制方式是：决定I/O-type的数值是&H60。
  - 当多个开关同时被按下时，可以被识别的最多个数可以控制。
  - 指定能够识别同时按下开关的最多个数，在0～640之间。
  - 使用0来使开关禁止输入。这时开关不能使用。因此，要使用其它的方法来使输入禁止复位。
  - 对于带有停电记忆芯片的触摸屏，使用该指令设置的开关能够实现停电保持。所以，最大开关个数的设定应该在每次上电时都执行一次的程序中进行。
- 无协议通信数据发送缓冲区使用如下方式进行清除，决定I/O-type的数值为&H41。
  - 指定一个端口(CH1~CH3) 来清除数据发送缓冲区。代号为1～3。

■ 相关项目 IOSTAT

■ **程序实例：**

```
evnt
   input ty%,id@,dat%
   if id@ = ..sw1 and dat% = 1 then
      ioctl 0,0
   else
      ioctl 0,1
   endif
end evnt
```

IOCTL2

指令

■ **功能** IOCTL2 指令 用于控制PLC的PLC 的cyclic 方式通信。

■ 格式 IOCTL2 设备名称, 代码(code), 数据(data)

■ 使用范例 IOCTL2 00~D10, 0, 0

■ 说明
- 执行 IOCTL2 指令则执行设备名称指出的CYCLIC通信。需要进行CYCLIC通信的设备必须预先使用CYCLIC或CYCLIC2指令进行声明。
- 将code 和 data 设置为0.

■ 相关项目 无

■ 程序实例

```
conf
   cyclic 00~D10
end conf
evnt
   input ty%,id@,dat%
   if id@ = ..sw1 then
      00~D11 = 1
      ioctl2 00~D10 ,0 ,0
   endif
end evnt
```

IOSTAT

函数

■ **功能** IOSTAT 函数用于读取与OIP相连I/O设备的状态。

■ 格式 IOSTAT (I/O-type)

■ 使用范例 IOSTAT (0)

■ 说明

- 用来写入表示状态为I/O-type的I/O设备的整数值。I/O-type指需要控制的I/O设备，在这里I/O设备只能为PLC、开关或无协议通信缓冲区。
- 要读取PLC的状态，应指定I/O-type为0。
  0：允许对PLC进行读写。
  1：禁止对PLC进行读写。
- 要读取开关（Switch）的状态，则指定I/O-type为&H60。
  - 返回值为能够识别同时按下开关的最大个数。在0～640之间。

■ 相关项目 IOCTL

■ 程序实例

```
evnt
   input ty%,id@,dat%
   if id@ = ..sw1 then
      if iostat(0) then
         ioctl 0,0
      else
         ioctl 0,1
      end if
   endif
end evnt
```

JUMP

指令

■ **功能**　　JUMP 从当前画面跳至指定的画面。

■ 格式　　JUMP 画面名

■ 使用范例　　JUMP 10

■ 说明

- 执行JUMP指令，使系统显示由“画面名”指定的画面。
- 画面名指目标画面的画面名称或能代表它的ID变量名。
- 当执行这条指令时，其后面的程序将不执行。
- 当目标画面不存在时，系统将报错。

■ 相关项目　　无

■ 程序实例

```
evnt
   input type , id@ , data
   if type = 3 and id@ = ..SWT000 then
      JUMP GAMEN..
   end if
end evnt
```

KILL

指令

■ **功能** KILL 删除指定的文件

■ 格式 KILL 文件名

■ 使用范例 KILL “G:ABC.DOC”

■ 说明
- KILL 删除文件名指定的文件。
- 文件名中可以带有*符号。

■ 相关项目 无

■ 程序实例

```
conf
end conf

evnt
   .......
   KILL “G:ABC.TXT”
   .......
end evnt
```

LAMPCOLOR

指令

■ **功能**　　LAMPCOLOR 指令用于改变指示灯ON状态的显示颜色。

■ 格式　　LAMPCOLOR 指示灯控件名称, 颜色代号

■ 使用范例　　LAMPCOLOR .BUHIN.GRAPH, 5

■ 说明　　LAMPCOLOR 指令用于改变指示灯ON状态的显示颜色。
- 指示灯控件名或能代表它的ID变量名。
- 颜色代号指状态为ON时指示灯的显示颜色代号。范围 0～15。

■ 相关项目　　LAMPDSP

■ 程序实例

```
conf
   lampdsp .buhin.gpaph , 0
   LAMPCOLOR .buhin.gpaph , 7
   lampdsp .buhin.gpaph , 1
end conf
```

LAMPDSP

指令

■ **功能** LAMPDSP 指令用于改变指示灯的状态（ON或OFF）。

■ 格式 LAMPDSP 控件名, 指示灯状态

■ 使用范例 LAMPDSP .BUHIN.GRAPH, 1

■ 说明 LAMPDSP 用于指定指示灯的显示状态是ON还是OFF。

- 控件名是指指示灯控件的名称或能代表该控件的ID型变量。
- 指示灯状态表示指示灯是ON还是OFF，当为0时表示显示状态为OFF，为1时表示显示状态为ON。
- 当控件内部的控制参数有效时，这里设置的值便不起作用。

■ 相关项目 LAMPCOLOR

■ 程序实例

```
evnt
   input type,id@,data
   var@ = .buhin.graph
   LAMPDSP  var@ , data
end evnt
```

LEFT$

函数

■ **功能**　　LEFT$ 函数从指定的字符串左边开始读取指定长度的字符串。

■ 格式

LEFT$ (字符串，字符个数)
LEFT$ (注册字符串编号，字符个数)
LEFT$ (注册字符串名称，字符个数)

■ 使用范例

```
A$ = LEFT$ (MOJI$, 5)
A$ = LEFT$ (4, 10)
A$ = LEFT$ (TOROKU, 8)
```

■ 说明

- LEFT$ 函数用来从指定的字符串左边开始读取指定长度的一个字符串。
- 字符个数指要从指定字符串读取的字符个数，范围在0～255之间。当字符数为0时，返回一个空字符。
- 字符串可以是直接的字符串也可以是一个字符串变量。
- 注册字符串编号指在SCA2 里面注册的文本编号。
- 注册字符串名称指SCA2 里注册过的文本的名称，或能代替它的ID变量。

■ 相关项目　　MID$, RIGHT$

■ 程序实例

```
evnt
   b$ = “12345678”
   a$ = LEFT$(b$ , 3)
   c$ = LEFT$ (no , 3)
   c$ = LEFT$ (id@ , 4)
end evnt
```

LEN

函数

■ **功能**　　LEN 函数以字节为单位返回指定字符串的长度。

■ 格式　　LEN (字符串)
LEN (字符串注册登记号)
LEN (这册字符串名称)

■ 使用范例　　A = LEN (B$)
A = LEN (MOJI)

■ 说明

- LEN 函数用于计算括号里的字符串长度（字节数，即字符个数）。
- 字符串可以是直接给出的字符串，也可以是一个字符串变量。
- 字符串注册登记号为表示在SCA2 里注册登记文本号的数学表达式。
- 注册字符串名称为表示在SCA2 里注册过的文本的名称或能代表它的ID变量。

■ 相关项目　　无

■ 程序实例

```
conf
   a = len (b$)
   a = len (“abcdefg”)
   a = len ( toroku )
   a = le  (1)
end conf
```

LINE

指令

■ **功能** 在屏幕上画直线

■ 格式 LINE X1, Y1, X2, Y2

■ 使用范例 LINE 20,30,100,200

■ 说明

- •这是在指定的两个坐标((X1,Y1)和(X2,Y2))之间用直线连接的状态。
- •X1,X2是0 ~ 799,Y1是gc - a24:0 ~ 479,GC-A25/ a26:0 ~ 599的范围。
- 直线作为画面的背景直接被显示。
  在直线的表示了的区域零件进行开放/关闭，控制的表示，在零件的区域内被表示了的直线有被清除的情况。
  被清除的直线不会再被表示。
- 直线的线型和颜色由COLOR指令决定。

■ 相关项目 COLOR

■ 程序实例

```
conf
   color 1 , 0 , 2
end conf
evnt
....
   dot 100,200
   dot 100,300
   color 1 , 0 , 0
   line 100,200,100,300
....
end evnt
```

LINPUT

指令

■ **功能** LINPUT指令用来从指定的文件中读取数据。

■ 格式 LINPUT 文件编号, 字符串变量

■ 使用范例 LINPUT 12, STRING$

■ 说明
- LINPUT 指令用来从指定的文件中将数据读取到字符串变量中。
- 从当前文件到回车或换行（CR/LF）符之间的数据将被赋给字符串变量。
- 文件编号必须与使用FOPEN指令打开的文件名称一样。

■ 相关项目 FOPEN, FCLOSE, FPRINT, FWRITE, FINPUT

■ 程序实例

```
conf
   fopen  “G:\TEST”, 2 , 5
end conf
evnt
   AAA$ = “12345”
   fwrite 5, AAA$, “ABCD”
   fseek(5, 0, 0)
   linput 5, VSTR$
end evnt
```

文件书写如下：

“12345”,”ABCD” CR/LF

当数据读入后，变量值如下：

VSTR$ “12345”,”ABCD”

LNECOLOR

指令

■ **功能**　　LNECOLOR 指令用于改变曲线图显示的线型和颜色。

■ 格式　　LNECOLOR 控件名称，曲线编号，线型，线颜色，tile, 显示颜色， 背景颜色

■ 使用范例　　LNECOLOR ..LNE000, 1, 2, 1, 4, 5, 2

■ 说明

- LNECOLOR 指令用来改变曲线图表显示的颜色背景等属性。
- 控件名称指曲线显示控件的名称或相应的ID变量。
- 曲线编号指出要改变的是哪条曲线，编号从1开始。
- 线型指线型代码，有4种线型，代号分别为0～3。
- tile indicates the tiling figure of the bar. Specify this tiling figure with a numeric value from 0 to 15.
- display-color is the numeric value indicating the color number of the tile display section. Specify this color number with a numeric value from 0 to 15.
- background-color is the numeric value indicating the color number of the tile background section. Specify this color number with a numeric value from 0 to 15.

■ 相关项目　　LNEDSP, LNESHIFT

■ 程序实例

```
conf
   static name@
   name@ = ..LNE000
end conf
evnt
   input type%, id@, data%
   if type% = 3 then
      LNECOLOR name@, 2, 3, 1, 4, 5, 2
   endif
end evnt
```

LNEDSP

指令

■ **功能** LNEDSP 指令采用线型图表显示数据。

■ 格式 LNEDSP 控件名称，曲线编号，拐点编号，显示数据。

■ 使用范例 LNEDSP .BUHIN.GRAPH, 2, 2, 30.0

■ 说明

- LNEDSP 指令采用线型图表显示数据。
- 控件名称指趋势图显示器的名称或其ID变量。
- 曲线编号指出使用那一条曲线，第一条为1。
- 拐点编号用来指出要改变的是曲线中哪个拐点的显示值。它可以是一个大于等于1的整数值。
- 显示值是指定的拐点显示的值。
- 如果控件中参数设置为有效，则显示值在这里不能改变。

■ 相关项目 LNECOLOR, LNESHIFT

■ 程序实例

```
conf
   static name@
   name@ = ..LNE000
end conf
evnt
   input type%, id@, data%
   lnedsp name@, 2, 2, data%
end evnt
```

LNESET

指令

■ **功能** LNESET 指令用来设置曲线图表显示的数值。

■ 格式 LNESET 控件名称, 曲线编号, 拐点编号, 显示数值

■ 使用范例 LNESET .BUHIN.GRAPH 2, 4, 30.0

■ 说明

- LNESET 指令用来设置曲线显示图表显示的数据。当设置的点数多于两点时，执行本指令后再执行PRDSP指令的显示速度比执行LNEDSP指令要快。
- 控件名称指曲线图表显示器控件的名称或其ID变量。
- 当曲线多于（或等于）两条时，曲线编号用来指出要改变显示数据的是那条曲线。编号为大于或等于1的整数。
- 拐点编号指出要改变显示值的是曲线上哪一点。编号同样为大于或等于1的整数。
- 显示数值指指定的拐点的显示幅度。

■ 相关项目 LNEDSP, PRDSP

■ 程序实例

```
evnt
   lneset  .buhin.graph , 3 , 8 , 20.1
   var@ = .buhin.graph
   no = 4
   value = 23
   point = 4
   LNESET  var@ , no , point, value
   prdsp var@
end evnt
```

LNESHIFT

指令

■ **功能**　　LNESHIFT 指令将曲线的显示值左移或右移。

■ 格式　　LNESHIFT (控件名, 曲线编号, 移动方向, 显示数据)

■ 使用范例　　A = LNESHIFT (..LNE000, 1, 1, 30)

■ 说明

- LNESHIFT 指令将曲线图表显示器中组成曲线的各点向左或向右移动，并显示新的曲线。
- 当该指令执行后，曲线上的各点移动后的值作为移动结果返回。
- 控件名称指曲线图表显示器控件的名称或其ID变量。
- 当曲线多于（或等于）两条时，曲线编号用来指出要改变显示数据的是那条曲线。编号为大于或等于1的整数。
- 移动方向为移动方向代号，向左或向上为1；向右或向下为-1。
- 显示值即曲线上将要移动的值。

■ 相关项目　　LNEDSP, LNECOLOR, LNESHIFT2

■ 程序实例

```
evnt
   input type%, id@, data%
   if data% > 0 then
      abc% = lneshift ( ..LNE000, 1, 1, 0)
   else
      abc% = lneshift ( ..LNE000, 1, -1, 100)
   endif
end evnt
```

LNESHIFT2

指令

■ **功能**　　LNESHIFT2 指令将曲线显示的数据左移或右移。

■ 格式　　LNESHIFT2 (控件名称，曲线编号，移动方向，显示值)

■ 使用范例　　A = LNESHIFT2 (..LNE000, 1, 1, 30)

■ 说明

- 与LNESHIFT 指令不同的是，LNESHIFT2 指令只移动曲线但并不进行显示，要显示移动结果还要执行PRDSP指令。
- LNESHIFT2 指令将曲线图表显示器中组成曲线的各点向左或向右移动。
- 当该指令执行后，曲线上的各点移动后的值作为移动结果返回。
- 控件名称指曲线图表显示器控件的名称或其ID变量。
- 当曲线多于（或等于）两条时，曲线编号用来指出要改变显示数据的是那条曲线。编号为大于或等于1的整数。
- 移动方向为移动方向代号，向左或向上为1；向右或向下为-1。
- 显示值即曲线上将要移动的值。

■ 相关项目　　LNEDSP, LNECOLOR, LNESHIFT, PRDSP

■ 程序实例

```
evnt
   input type%, id@ data%
   if data% > 0 then
      abc% = lneshift2 ( ..LNE000, 1, 1, 0)
   else
      abc% = lneshift2 ( ..LNE000, 1, -1, 100)
   endif
   prdsp ..LNE000
end evnt
```

LOCAL

指令

■ **功能** LOCAL 用来定义局部变量

■ 格式 LOCAL 变量名称 [, 变量名称 ...]

■ 使用范例 LOCAL VAR , XYZ(2,3) , MOJI$ * 20

■ 说明

- LOCAL 指令用来将其后的变量定义成局部变量。
- 局部变量（Local variable）只能被其所声明的程序中引用。如果使用了未定义的局部变量，编译器将会报错。局部变量在每次程序被执行时自动更新。
- 变量名可以是普通变量，也可以是数组变量或字符串变量。A
- 在定义数组变量或字符串变量时不需要使用DIM或STRING指令。
- LOCAL 指令是SCA2 的又一个特性。
- DIM 可以代替 LOCAL使用，但是应该尽量使用LOCAL指令。
- STRING 也可以代替LOCAL使用来指定字符串变量的大小，但是还是应该尽量使用LOCAL指令。

■ 相关项目 AUTO,BACKUP,DIM,FUNCTION,GLOBAL,STATIC,STRING

■ 程序实例

```
conf
   global float(5)
   LOCAL i%
   for i% = 0 to 5
      float(i%) = i%*3
   next
end conf
```

LOCALCHECK

指令

■ **功能**　　LOCALCHECK 指令通过 编译器控制（改变）报警消息的输出等级。

■ 格式　　LOCALCHECK　警告等级

■ 使用范例　　LOCALCHECK　1

■ 说明

- LOCALCHECK 指令用来指出，如果在程序中出现了不明确（未定义）的局部或全局变量、函数或子程序时，是否给以警告。
- 有两种等级如下：
  1:　出示警告
  0:　无警告
- 警告有如下三种类型：
  (1) 如果程序中使用了未声明的变量
  这种情况下，在局部画面中，编译器会默认该变量为局部变量；在全局画面里，该变量默认为全局变量。
  (2) 如果全局变量与局部变量、或全局子程序与局部子程序的名称完全相同
  这种情况下，变量将被视为全局变量；子程序将被视为全局子程序。
  (3) 如果两个或以上的全局、局部（或库）函数的名称完全相同
  这种情况下，如果存在该库函数，则被视为库函数，否则将被视为全局函数。
- 如果不使用LOCALCHECK 指令，警告等级默认为0。
- 程序中，警告等级从LOCALCHECK指令所在的地方开始变化（生效）。

■ 相关项目　　BACKUP,DECLARE,DIM, FUNCTION,GLOBAL,LOCAL

■ 程序实例

```
conf
   local newvar3$
   newvar1$ = "no warning"
   LOCALCHECK 1
   newvar2$ = "warning is given!"
   newvar3$ = "no warning"
end conf
```

LOF

函数

■ **功能** The LOF 函数用来计算文件的大小。

■ 格式 LOF（文件号）

■ 使用范例 AAA = LOF（文件号）

■ 说明
- 文件号必须同前面使用FOPEN指令打开文件的编号相同。
- 文件大小的计算结果以字节（byte）为单位。

■ 相关项目 FOPEN, FIELD, FCLOSE, FPUT, FGET, EOF

■ 程序实例

```
conf
   field  5
      global no%
      global moji1$ , moji2$
   end field
   global sum%
   fopen  “E:\MEMORY”, 2 , 5
   ......
end conf
evnt
   no% = 1
   moji1$ = “product-name”
   moji2$ = “product-number”
   fput 5 , 3
   if  LOF(5) > 100 then
       fclose 5
   end if
end evnt
```

LOG

函数

■ **功能** LOG 函数用来计算数学表达式的自然对数值。

■ 格式 LOG (数学表示)

■ 使用范例 A = LOG (B*C)

■ 说明 • 注意，数学表达式的值必须大于0。

■ 相关项目 EXP

■ 程序实例

```
conf
   la = log ( 10 )
   lb = log ( a * b )
end conf
```

MCPY

指令

■ **功能**　　MCPY 指令用来将某个区域（文件）的内容拷贝到一个字符串变量。

■ 格式　　1:MCPY 文件号，字符串变量
2:MCPY 字符串变量名，文件号

■ 使用范例　　MCPY 5, moji$

■ 说明

- MCPY 指令用来将变量群中的内容拷贝到字符串变量，或将字符串变量中的内容拷贝到FIELD中的变量中。
- 文件号指在FIELD中定义的文件编号。
- 当数组变量或字符串变量被拷贝后，不管谁空间小，都将被占用。

■ 相关项目　　FOPEN, FIELD, FCLOSE, FPUT, FGET, EOF, SOF

■ 程序实例

```
conf
   field  5
      global no%
      global moji1$ , moji2$
   end field
   global buff$ * 50
   opensio 1 , 0 , buff$
   fopen  “MEMORY”, 2 , 5
end conf
evnt
   no% = 1
   moji1$ = “product-name”
   moji2$ = “product-number”
   size% = sof(5)
   MCPY 5 , buff$
   writesiob 1 , size% , buff$
end evnt
```

MEDIACHK

函数

■ **功能**　　MEDIACHK 函数用来检测驱动器上是否存在介质。

■ 格式　　MEDIACHK (驱动器名)

■ 使用范例　　STATUS% = MEDIACHK("E:")

■ 说明

- 返回结果如下：
  0:　不存在
  1:　存在

■ 相关项目　　MEDIASIZE

■ 程序实例

```
conf
   global dname$(13)
   global dsel%
   strdsp ..str, "mediachk"
end conf
evnt
   input type%, id@, data%
   if data% = 1 then
      strdsp .dsp.str, dname$(dsel%)
      num% = mediachk(dname$(dsel%))
      if num% = 1 then
         strdsp ..str, "valid"
      else
         strdsp ..str, "invalid"
      end if
   end if
end evnt
```

MEDIASIZE

函数

■ **功能**　　MEDIASIZE 函数用于检查驱动器的的介质空间大小，返回的是字节数。

■ 格式　　MEDIASIZE (驱动器名，计算方法)

■ 使用范例　　SIZE% = MEDIASIZE("E:", 0)

■ 说明

- 计算方法有如下两种，代号分别如下：
  0:　被占空间
  1:　剩余空间

  当为0时，即计算有文件部分的磁盘空间大小；
  当为1时，即计算磁盘上剩余空间的大小。（以字节为单位）

■ 相关项目　　MEDIACHK

■ 程序实例

```
conf
   global dname$(13)
   global dsel%
   static mode%
   mode% = 0
   strdsp ..str, "mediasize"
   numdsp ..num001, mode%
end conf
evnt
   input type%, id@, data%
   if data% = 1 then
      if mode% = 1 then
         mode% = 0
      else
         mode% = 1
      end if
      numdsp ..num001,mode%
      strdsp .dsp.str, dname$(dsel%)
      num% = mediasize(dname$(dsel%),mode%)
      numdsp ..num000,num%
   end if
end evnt
```

MID$

指令

■ **功能**　　MID$ 指令用另一个字符串代替字符串的一部分。

■ 格式　　MID$ (字符串变量，起始位置，字符数)＝更换字符串

■ 使用范例　　MID$ (x$, 1, 1) = “A”

■ 说明

- MID$ 指令用“更换字符串”来代替“字符串变量”中从“起始位置”开始的“字符数”个字符。
- 如果字符数大于字符串变量的字符数，则只代替变量中有的字符。所以，字符串的长度不变。
- 要被代替字符串的“起始位置”从 1 开始。
- 当字符数为负，或起始位置为0或为负时，系统在编译时会报错。

■ 相关项目　　LEFT$, RIGHT$, MID$

■ 程序实例

```
conf
   static moji$
   moji$ =” ABCDEFG”
end conf
evnt
   input type, id@, data$
   mid$(moji$, 4, 3) = data$
end evnt
```

MID$

函数

■ **功能**　　MID$ 函数返回一个指定长度的字符串。

■ 格式　　MID$ (字符串，起始位置，字符个数)
MID$
(字符串注册号，起始位置，字符数)

■ 使用范例　　MID$
(注册字符串名称，起始位置，字符数)

■ 说明

```
A$ = MID$ (X$, 2, 3)
A$ = MID$ (10, 2, 3)
A$ = MID$ (NAME, 2, 3)
```

■ 相关项目

- MID$ 函数从指定的字符串中的“起始位置”开始读取指定个数的字符。
- 字符串可以是直接字符，也可以由字符串变量给出。
- “字符串注册号“为SCA2里设定的编号。
- “字符串注册名称”可以是在SCA2里面注册文本的名称，或其ID变量名。
- 当“字符个数”为0或“起始位置”大于字符串的字符个数时，将返回一个空字符。

■ 程序实例　　LEFT$, RIGHT$

```
evnt
   input type,id@,data$
   a$ = mid$(data$ , 3 , 3)
   strdsp ..STR000,a$
end evnt
```

MKB

指令

■ **功能**　　MKB 指令将数据存储到字符串变量的任意位置。

■ 格式　　MKB 字符串变量，存放地，数据

■ 使用范例　　MKB MOJI$, 5, VAR

■ 说明

- MKB 指令将数据地低位字节放入字符串变量中的某个位置(由“存放地”指定）。
- 存放低必须是整型或浮点型常数或变量，1表示起始位置。
- 要写入的数据必须是整型或浮点型常数或变量，当为浮点型变量或常数时，先将其转换成整型，在写入时只写入一个字符。

■ 相关项目　　MKS, MKW, MKI, MKF, MKID, CVB, CVW, CVI, CVF, CVID

■ 程序实例

```
conf

end conf
evnt
   org$ = “1234567”
   strdsp ..STR000, org$
   MKB org$, 2, &H39
   strdsp ..STR001, org$
end evnt
```

MKDIR

指令

■ **功能**　　MKDIR 指令用来创建一个文件夹。

■ 格式　　MKDIR 文件夹名称

■ 使用范例　　MKDIR “TEST”

■ 说明

- MKDIR 指令用来创建一个子文件夹
- 文件夹名称可以用字符串常量或变量指定。
- 文件夹名称里可以包括磁盘驱动器名称。

■ 相关项目　　RMDIR, CHDIR, DIR

■ 程序实例

```
conf
end conf
evnt
   ......
   MKDIR “C:TEST”
   ......
end evnt
```

MKF

指令

■ **功能**　　MKF 指令将数据存储到字符串变量的某个位置。

■ 格式　　MKF 字符串变量，存储位置，实数值

■ 使用范例　　MKF MOJI$, 5, VAR

■ 说明

- MKF 指令将4个字节的实数写入字符串变量中的指定位置，位置由存储位置指定。
- 存储位置必须是整型或浮点变量或常数。1 表示变量的开始位置。
- 实数值指要被写入的值，它必须是一个整型或浮点变量或常数。但当是整型变量或常数时，将自动先将其转换成实数。
- 值被转换成ASCII码然后进行保存。

■ 相关项目　　MKS, MKB, MKW, MKI, MKID, CVB, CVW, CVI, CVF, CVID

■ 程序实例

```
conf

end conf
evnt
   org$ = “1234567”
   strdsp ..STR000, org$
   MKF org$, 2, 1.23
   strdsp ..STR001, org$              ’    字符串将不能正确显示
end evnt
```

MKI

指令

■ **功能**　　MKI 指令将数据存储到字符串变量里。

■ 格式　　MKI 字符串变量名，存储位置，整数值

■ 使用范例　　MKI MOJI$, 5, VAR

■ 说明

- MKI 指令将4个字节的整数存储到字符串变量中指定的位置。
- 存储位置必须是整型或浮点变量或常数。1 表示变量的开始位置。
- 整数值指要被写入的值，它必须是一个整型或浮点变量或常数。但当是实型变量或常数时，将自动先将其转换成整型变量或常数。
- 值被转换成ASCII码然后进行保存。

■ 相关项目　　MKS, MKB, MKW, MKF, MKID, CVB, CVW, CVI, CVF, CVID

■ 程序实例

```
conf

end conf
evnt
   org$ = “1234567”
   strdsp ..STR000, org$
   MKI org$, 2, &H39404142
   strdsp ..STR001, org$
end evnt
```

MKID

指令

■ **功能** MKID 指令将数据存储到字符串变量中指定的位置。

■ 格式 MKID 字符串变量名，存储位置，ID-值

■ 使用范例 MKID MOJI$, 5, VAR

■ 说明

- MKID 指令将ID-值的6个字节的数据存储到字符串变量中的指定位置。
- 存储位置必须是整型或浮点变量或常数。1 表示变量的开始位置。
- ID-值指要被写入的值，它必须是一个ID型变量或常数。但当变量或常数是非ID型时，系统将会报错。
- 值被转换成ASCII码然后进行保存。

■ 相关项目 MKS, MKB, MKW, MKI, MKF, CVB, CVW, CVI, CVF, CVID

■ 程序实例

```
conf

end conf
evnt
   input type%, id@, data%
   org$ = “1234567”
   strdsp ..STR000, org$
   MKF org$, 2, id@
   strdsp ..STR001, org$              ’    字符串将不能正确显示
end evnt
```

MKS

指令

■ **功能**　　MKS 指令将字符串数据存储到字符串变量中。

■ 格式　　MKS 字符串变量，存放地，字符串数据

■ 使用范例　　MKS MOJI$, 5, “ABCD”

■ 说明

- MKS 指令将“字符串”存储到“字符串变量”中的“存放地”。
- 存放位置必须是整型或浮点变量或常数。1表示变量的起始位置。
- 字符串就是将要写入的字符，可以是变量或常数。

■ 相关项目　　MKB, MKW, MKI, MKF, MKID, CVB, CVW, CVI, CVF, CVID

■ 程序实例

```
conf

end conf
evnt
   org$ = ``1234567''
   strdsp ..STR000, org$
   MKS org$, 2, ``76543''
   strdsp ..STR001, org$
end evnt
```

MKW

指令

■ **功能**　　MKW 指令将数据存入字符串变量。

■ 格式　　MKW 字符串变量，存放位置，数值

■ 使用范例　　MKW MOJI$, 5, VAR

■ 说明

- MKW 指令将两个字节的数据存放到“字符串变量”中的“存放位置”，存放地点从字符串的开始位置开始计算。
- 存放位置必须是整型或浮点变量或常数。1表示变量的起始位置。
- 数值指将被写入的数值，它必须是整型或浮点型变量或常数。当指定为整数时，浮点变量或常常数首先转换成整型数。执行该指令后，只将数值的最低两位写入字符串变量。
- 值被转换成ASCII码然后进行保存。

■ 相关项目　　MKS, MKB, MKI, MKF, MKID, CVB, CVW, CVI, CVF, CVID

■ 程序实例

```
conf

end conf
evnt
   org$ = “1234567”
   strdsp ..STR000, org$
   MKW org$, 2, &H3940
   strdsp ..STR001, org$
end evnt
```

MOVE

指令

■ **功能**　　MOVE 指令移动指定的部品

■ 格式　　MOVE 部品名称，X方向移动量，Y方向移动量，移动方式

■ 使用范例　　MOVE .BUHIN., 100, 20, 0

■ 说明

- 部品名称指要移动的部品的名称或其ID变量。
- X方向移动量，Y方向移动量 指部品要在屏幕上的移动幅度。屏幕左上角的坐标为（0，0），向右为X轴，向下为Y轴。对于GC56LC2和GC55EM2，X方向移动量范围在0～639之间，Y方向移动量GC56LC2范围在0～479之间，GC55EM2范围在0～399之间。对于GC53LC2和GC53LM2，X方向移动量范围在0～319，Y方向移动量范围在0～239。
- 移动又分为相对移动和绝对移动两种方式，对于绝对移动，移动方式代号为0；相对移动方式代号为1。绝对移动指相对与屏幕左上角要移动的目标位置。相对移动指相对部品当前位置的移动量。

■ 相关项目　　无

■ 程序实例

```
evnt
   input type,id@,data
   if type = 3 then
      buhin@ = .buhin2.
      MOVE buhin@ , 10 , 10 , 0
   endif
end evnt
```

MTRCOLOR

指令

■ **功能**　　MTRCOLOR 指令用来改变模拟仪表指针的颜色。

■ 格式　　MTRCOLOR 控件名称，显示颜色

■ 使用范例　　MTRCOLOR ..MTR000, 1

■ 说明

- MTRCOLOR 指令用来改变模拟仪表指针的颜色。
- 控件名称指仪表显示控件的名称或其ID变量。
- 颜色代号指指针的目标颜色代码，使用0～15来指定颜色。

■ 相关项目　　MTRDSP

■ 程序实例

```
evnt
   input type%, id@,  mcolor%
   MTRCOLOR ..MTR000, mcolor%
end evnt
```

MTRDSP

指令

■ **功能** The MTRDSP 指令 使用仪表显示数据。

■ 格式 MTRDSP 控件名称，显示数据

■ 使用范例 MTRDSP .BUHIN.GRAPH, 30.0

■ 说明
- 这个命令是让仪表显示值的命令。
- 控制件名是仪表的名称，或者是表示仪表的ID型变量。
- 显示数据是仪表上显示的数值数据。
- 通过控制将操作参数设定为有效时，
  即使发布了这个命令也不能改变显示值。

■ 相关项目 MTRCOLOR

■ 程序实例

```
conf
   static name@
   name@ = ..MTR000
end conf
evnt
   input type%, id@, data%
   MTRDSP name@, data%
end evnt
```

NUMCOLOR

指令

■ **功能**　　NUMCOLOR 指令 用来改变数字显示器的显示颜色和背景。

■ 格式　　NUMCOLOR 控件名称，数字显示颜色，填充模式，填充颜色，背景颜色

■ 使用范例　　NUMCOLOR ..GRAPH, 1, 2, 5, 2

■ 说明

- NUMCOLOR 指令用来改变数字显示器的显示颜色和背景。
- 控件名称指数字显示器控件的名称或其ID变量。
- 数字显示颜色指数据的颜色，代号在0～15之间。
- 填充模式指填充所用的图形，代号也在0～15之间。
- 填充颜色指填充部分的颜色，代号也在0～15之间。
- 背景颜色指背景部分的颜色代号，代号在0～15之间。

■ 相关项目　　NUMDSP, NUMFORM

■ 程序实例

```
conf
   static name@
   name@ = ..NUM000
end conf
evnt
   input type%, id@, data%
   if type% = 3 then
      NUMCOLOR name@, 2, -1,-1,-1
   endif
end evnt
```

NUMDSP

指令

■ **功能**　　NUMDSP ： 在数据显示器中显示数据。

■ 格式　　NUMDSP 数据显示器控件，显示数据

■ 使用范例　　NUMDSP .BUHIN.GRAPH, 30.0

■ 说明

- 这个命令是让数值显示器显示值的命令。
- 控制名称是数值指示器的名称，或是表示数值指示器的ID型变量。
- 显示数据是在数值显示器上显示的数值数据。
- 数值显示器成为连级时，若指定数值显示器名，则全部要素相同显示自己的数据。当我们为每个元素设置值时，我们使用GETID求出nutroll的ID，请把这个值作为控件名。
- 在控制中设定工作参数有效的情况下，即使发出这个命令显示值不能变更。

■ 相关项目　　NUMCOLOR, NUMFORM

■ 程序实例

```
conf
   static name@
   name@ = ..NUM000
end conf
evnt
   input type%, id@, data%
   NUMDSP name@, data%
end evnt
```

NUMDSP2

指令

■ **功能**　　NUMDSP2 ：在数据显示器中显示双字数据。

■ 格式　　NUMDSP2　数据显示器控件，显示数据

■ 使用范例　　NUMDSP2 ..NUM000,var%, NUMDSP2 GLOBAL.B001.NUM000,var%(0)

■ 说明

- NUMDSP2 指令用来指定数据显示器中显示双字数据。
- 数据显示控件可以是控件名称或能代表它的ID变量。
- 显示数据可以由要数字或其数学表达式给定。
- 当数据显示控件连续放置时，使用数据显示控件名来指定控件可以使所有的控件中都显示相同的数字。当要为每个显示单元设定数据时，使用GETID函数来读取（计算）控件的ID值，然后用这个ID值代替指令后面的控件名。
- 如果控件的参数设置为有效（effective），则使用本指令设置的显示数据无效。

■ 相关项目　　NUMCOLOR, NUMFORM, NUMDSP

■ 程序实例

```
conf
   static name@
   name@ = ..NUM000
end conf
evnt
   input type%, id@, data%
   NUMDSP2 name@, data%
end evnt
```

NUMFORM

指令

■ **功能**　　NUMFORM 指令用于改变数字的显示形式。

■ 格式　　NUMFORM 控件名称, 显示方式, 小数点位置

■ 使用范例　　NUMFORM ..HYOJIKI, 0, 0

■ 说明

- NUMFORM 指令用于改变数字显示器中数字的显示形式。该指令还可以改变小数点的显示位置。
- 控件名指数学显示器的名称或能代替它的ID型变量。
- 显示方式有如下7 种：

  0: 浮点数显示方式　　4: 二进制表示
  1: 整型数显示方式　　5: 八进制表示
  2: 固定小数点显示方式　　6: 十六进制表示
  3: 二进制固定小数点显示
- 当显示方式为2（固定小数点显示方式）时，小数点位置就表示小数点的显示位置。当要在从数字右边的第一个位置显示小数点时，该值为1，第二个位置时为2。
- 二进制固定小数点显示（方式3）在数据的某个数据位上写入小数点。
- 记住，一定要在本指令后面执行NUMDSP，否则显示的数据将会顺序不对。

■ 相关项目　　NUMCOLOR, NUMDSP

■ 程序实例

```
evnt
   input type , id@,data
   var@ = .buhin.gamen
   NUMFORM var@ , data , 2
   numdsp  var@ , 30.1
end evnt
```

OCT$

函数

■ **功能**　　OCT$ 函数将一个十进制字符串转换成八进制字符串。

■ 格式　　OCT$（数学表达式）

■ 使用范例　　OCT$（134）

■ 说明

- OCT$ 函数将一个十进制字符串转换成八进制字符串。
- 当数学表达式为浮点型时，首先将十进制字符串转换成整数，然后再将其转换成八进制字符串。
- 十进制字符串（数值）范围应该在－2147483648 ～ 2147483647之间。

■ 相关项目　　HEX$, VAL

■ 程序实例

```
evnt
   input type , id@ , data
   moji$ = OCT$(data)
   strdsp ..STR000, moji$
end evnt
```

ONFERR

指令

■ **功能**　　ONFERR 指令用来指定出错消息发送的目标。

■ 格式　　ONFERR 目标

■ 使用范例　　ONFERR .B000.

■ 说明

- ONFERR 指令用来指定文件运行出错信息发送的目标。
- 目标可以由画面或部品（名称），或者画面或部品的ID变量。
- 当通过INPUT指令来接收数据，出错消息发送到的部品或画面可以接收诸如消息类型（type%=8）和数据（data%=出错代号）之类的信息。

■ 相关项目　　FOPEN, FCLOSE, FPRINT, FWRITE, FINPUT

■ 程序实例

```
conf
   ONFERR ..
end conf
evnt
   input ty%, id@, dat1%
   ......
end evnt
```

When an error occurs, 8 is set in ty% and an error code (number) is set in dat1%.

OPEN

指令

■ **功能**　　OPEN 指令用来打开一个处于关闭（Close）状态得部品。

■ 格式　　OPEN 部品, 模式

■ 使用范例　　OPEN .BUHIN., 1

■ 说明

- OPEN 指令用来打开(显示)处于关闭(Close)状态的部品(Part)。
- 部品名称指要打开的部品名，或其ID变量。
- 模式用来指出，部品被打开时是否执行其Conf 程序块的程序。
  0: Conf 程序块不执行；
  1: Conf 程序块执行。

■ 相关项目　　CLOSE

■ 程序实例

```
evnt
   input type% , id@ , data%
   if pstat(.BUHIN.) = 3 then
      OPEN .BUHIN., 0
   endif
end evnt
```

OPENCOM

指令

■ **功能**　　OPENCOM 用来使程序可以从串行口接受数据。

■ 格式　　OPENCOM 逻辑设备名称

■ 使用范例　　OPENCOM HST

■ 说明

- OPENCOM 指令用来OIP串口，使其可以从与之相连接得外设接受数据（注意，当上位机发送数据时，无需使用本指令）。
- 逻辑设备名称指一下某种外设：
  HST: 上位计算机
  BCR: 条形码读入机
  TKY: 十键键盘

■ 相关项目　　CLOSE COM, REOPENCOM

■ 程序实例

```
conf
   OPENCOM HST
end conf
evnt
   input type% , id@ , data%
   if type% = 3 and data% = 1 then
      CLOSECOM HST
   else if type% = 3 and data% = 0 then
      REOPENCOM HST
   endif
end evnt
```

OPENPARALLEL

指令

■ **功能**　　OPENPARALLEL 指令用来打开并行口，使其可以从并行口接收数据。

■ 格式　　OPENPARALLEL 输入位(input bit)，模式（mode）

■ 使用范例　　OPENPARALLEL 3, 1

■ 说明

- OPENPARALLEL指令用来打开并行口，使其可以从并行口接收数据。
- 输入位指从并行口的哪一位接收数据，可以是0～15。
- 模式指出何时接收数据，共有如下3种模式：
  1：在位状态由低电平变高电平时传送数据。
  2：在位状态由高电平变低电平时传送数据。
  3：在位状态由低电平变高电平或相反时传送数据。

■ 相关项目　　CLOSEPARALLEL, REOPENPARALLEL

■ 程序实例

```
conf
   OPENPARALLEL 3
end conf
evnt
   input type% , id@ , data%
   if type% = 3 and data% = 1 then
      CLOSEPARALLEL 3
   else if type% = 3 and data% = 0 then
      REOPENPARALLEL 3
   endif
end evnt
```

OPENSIO

指令

■ **功能**　　OPENSIO 指令用于打开无协议通信端口。

■ 格式　　OPENSIO 端口号, 模式（mode）, 接受缓冲区

■ 使用范例　　OPENSIO 1, 1, moji$

■ 说明

- OPENSIO 指令用来为串行通信打开通信端口。
- 端口号用于指定执行串行通信的通道，CH1~CH3分别用常数1～3指定。
- 模式指无协议通信的数据方式：0表示二进制方式；1表示文本方式。
- 接收缓冲区指存放接收数据的变量（的名称）。变量必须是全局或静态字符串变量。
- 在接收完从外部来的数据之后，当条件成立时，将会向执行该指令的画面或部品发送消息，以表示接收完毕。不能在两个或更多个部品同时执行OPENSIO指令。

  二进制方式：使用二进制方式时，在0～0FFh之间的代码都能被发送和接收；通过指定接收数据的长度，可以进行数据读写。

  文本方式：　在这种方式下，可以传送或接收的数据范围为1～0FFh；在这种方式下，要设置和使用文本的结束码。结束码用于判断接收的数据。

■ 相关项目　　CLOSESIO, SETSIO, WRITESIO, WRITWSIOB, FLUSH, IOCTL

■ 程序实例

```
conf
   global buf$ * 200
   OPENSIO 2 , 1 , buf$
   setsio  2 , &HD
end conf
evnt
   strdsp ..STR000 , buf$
   closesio 2
end evnt
```

OPENTIM

函数

■ **功能** OPENTIM 函数用来读取定时器资源。

■ 格式 OPENTIM ()

■ 使用范例 VAR@ = OPENTIM ()

■ 说明

- OPENTIM 函数用来分配（读取）使用定时器所需的资源。
- OPENTIM 函数必须是一个ID型函数，因为它要返回一个定时器的ID值。
- 被分配来的定时器ID可以用来设置定时器。
- 系统可以使用多大16个定时器，不用的定时器应该及时返回给系统（使用CLOSETIM指令）。
- OPENTIM 函数只能被当前画面或当前画面部品使用。（如果在没显示的画面及部品上执行该指令，系统将会报错）。
- 当画面被切换，被该指令打开的定时器不会自动关闭，如果这时定时器正在EVNT程序块中使用，则定时器消息将被发送到后面（未被显示）的画面。

■ 相关项目 CLOSETIM, STARTTIM, STOPTIM, CONTTIM, WRITETIM, READTIM

■ 程序实例

```
conf
   static timid@
   timid@ = OPENTIM()
   settim timid@, 20, 0
   starttim timid@
end conf
evnt
   input type% , id@ , data%
   if type% = 3 and id@ = ..SWT000 then
      stoptim timid@
   else if id@ = ..SWT001 then
      closetim timid@
   end if
end evnt
```

OPENTIM2

函数

■ **功能** OPENTIM2 函数用来分配（或打开）定时器。

■ 格式 RET = OPENTIM2 (定时器号)

■ 使用范例 RET = OPENTIM2 (14)

■ 说明

- OPENTIM2 函数用来打开由“定时器号”指出的定时器。
- 定时器号用来指出使用的是哪个定时器，有效范围是0～15。
- 当执行OPENTIM2时，返回如下其中一个值：
  0: 定时器可以打开
  1: 定时器不能打开
- OPENTIM2 函数只能在当前画面及其部品中使用，（如果该函数在非当前画面上执行，系统将会发生产生错误）。
- 在画面被切换后，分配的定时器不能自动关闭，所以如果定时器仍在被EVNT块中使用，则消息也会向非当前画面发送。

■ 相关项目 CLOSETIM, STARTTIM, STOPTIM, CONTTIM, SETTIM, READTIM, OPENTIM

■ 程序实例

```
conf
   static timid@
   ret=OPENTIM2(5)
   setim 5, 20, 0
   starttim 5
end conf
evnt
   input type% , id@ , data%
   if type% = 3 and id@ = ..SWT000 then
      stoptim 5
   else if id@ = ..SWT001 then
      closetim 5
   end if
end evnt
```

OPENTIM3

函数

■ **功能** OPENTIM3 函数用来读取（打开）定时器资源。

■ 格式 RET = OPENTIM3 (定时器号)

■ 使用范例 RET = OPENTIM3 (14)

■ 说明

- OPENTIM3 函数用来打开由“定时器号” 指定的定时器。
- 定时器号由0～15中的某个整数来指定，指出要打开哪个定时器。
- 当OPENTIM2 函数执行后，返回如下其中一个值：
  0: 定时器可以打开
  1: 定时器不能打开
- 当定时器所在画面被切换后，本定时器将自动关闭。
- OPENTIM3 函数只能用于当前画面或其部品，当在非当前画面上执行该函数时，系统将会报错。

■ 相关项目 CLOSETIM, STARTTIM, STOPTIM, CONTTIM, SETTIM, READTIM, OPENTIM

■ 程序实例

```
conf
   ret = opentim3 (3)
   settim 3 , 20, 1
   stoptim 3
   closetim 3
end conf
```

OUT

指令

■ **功能** OUT 指令将 2-byte（字节）的数据写入IO端口。

■ 格式 OUT 端口号，输出值

■ 使用范例 OUT 0, &H20

■ 说明
- 目前，数据只能被写到并行IO口。
- 端口号指出往哪个端口输出，（对于彩色/等离子体屏幕，端口号固定为0）。

■ 相关项目 INP

■ 程序实例

```
evnt
   input type,id@,data
   out 0, data
end evnt
```

OUTBIT

指令

■ **功能**　　OUTBIT 指令将数据写到指定端口的某位。

■ 格式　　OUTBIT 端口号，BIT号，写入数据

■ 使用范例　　OUTBIT 0, 10, 1

■ 说明

- OUTBIT 指令将指定的数据写到指定端口的某位。
- 端口号和BIT号由一个整数指出。
- 当写入数据为0时，表示将该位值OFF，当数据为1时，表示将该位置ON。
- 并行口的最低位为0，然后依次为1、2……
- 如果指定了不存在的BIT号或PORT端口，则系统将会报错。

■ 相关项目　　INP, OUT, INPBIT, OUTBITSTAT, OUTSTAT

■ 程序实例

```
evnt
DATA% = INPBIT(0,3)
   if data% = 0 then
      outbit 0,3,1
   endif
end evnt
```

OUTBITSTAT

函数

■ **功能**　　OUTBITSTAT 函数用来读取指定端口指定位的状态。

■ 格式　　OUTBITSTAT (端口号, 位编号)

■ 使用范例　　DATA% = OUTBITSTAT (0,10)

■ 说明

- OUTBITSTAT函数用来读取指定端口指定位的状态。
- 使用整数来指定端口号和位编号。
- 并行口IO的最低位编号为0，次低位为1，一次递增。
- 当指定的位不存在时，返回值为0。

■ 相关项目　　INP, OUT, INPBIT, OUTBIT, OUTSTAT

■ 程序实例

```
evnt
   data% = outbitstat(0,3)
   if data% = 0 then
      outbit 0,3,1
   endif
end evnt
```

OUTSTAT

函数

■ **功能**　　OUTSTAT 函数用来读取指定输出端口的数值。

■ 格式　　OUTSTAT (端口号)

■ 使用范例　　DATA% = OUTSTAT (0)

■ 说明

- OUTSTAT 函数读取指定输出端口的数值。
- 端口号使用整数值指定。
- 当指定的端口号不存在时，返回值为0。

■ 相关项目　　INP, OUT, INPBIT, OUTBIT, OUTBITSTAT

■ 程序实例

```
evnt
   data% = outstat(0)
   if data% = 0 then
      out 0,&hffff
   endif
end evnt
```

PIPCOLOR

指令

■ **功能**　　PIPCOLOR 指令用来改变管道状态分别为OFF, ON1, 和 ON2 时的显示颜色。

■ 格式　　LAMPCOLOR 控件名，状态代号，颜色代号

■ 使用范例　　LAMPCOLOR .BUHIN.GRAPH, 5

■ 说明　　PIPCOLOR 指令用来改变管道状态分别为OFF, ON1, 和 ON2 时的显示颜色。

- 控件名可以是控件的名称或其 ID 变量。
- 状态代号指代表状态OFF, ON1, ON2 的0、1、2 。.
- 颜色代码可以为0～15。

■ 相关项目　　PIPDSP

■ 程序实例

```
conf
   pipdsp .buhin.graph , 0
   PIPCOLOR .buhin.graph ,1 ,7
   lampdsp .buhin.graph , 1
end conf
```

PIPDSP

指令

■ **功能**　　PIPDSP 指令在管道（Pipe）显示器中显示数据（ON/OFF状态）。

■ 格式　　PIPDSP 控件名，数据（状态代号）

■ 使用范例　　PIPDSP .BUHIN.GRAPH, 1

■ 说明　　PIPDSP 指令将管道（pipe）显示器设置成OFF、ON1或ON2。

- 控件名指管道显示控件的名称或其ID变量。
- 表示状态OFF、ON1或ON2的代号分别为0、1、2。
- 如果在管道控件的中的参数设置成有效（“effective”），则使用本指令设置的数据就无效。

■ 相关项目　　PIPCOLOR

■ 程序实例

```
conf
   pipdsp .buhin.pip , 0
   PIPCOLOR .buhin.pip ,1 ,7
   pipdsp .buhin.pip , 1
end conf
```

PLTCOLOR

指令

■ **功能**　　PLTCOLOR 指令用来改变区域坐标图显示的颜色及其背景。

■ 格式　　PLTCOLOR 控件名称，点颜色，Tile ，显示颜色（Display color），背景颜色(background color)

■ 使用范例　　PLTCOLOR ..GRAPH, 1, 1, 2, 1

■ 说明

- PLTCOLOR指令用来改变区域坐标图显示的颜色及其背景。
- 控件名指显示器的名称或其ID变量。
- 点颜色指坐标点的颜色，可以为0～15。
- tile 指显示器的背景图（填充模式），可以从0～15。，
- 显示颜色(display-color)指填充背景图的颜色。可以为0～15。
- 背景颜色设置与上述相同。

■ 相关项目　　PLTDSP

■ 程序实例

```
conf
   static name@
   name@ = ..PLT000
end conf
evnt
   input type%, id@, data%
   if type% = 3 then
      PLTCOLOR name@, 2, 3, 1, 4
   endif
end evnt
```

PLTDSP

指令

■ **功能**　　PLTDSP 指令：在某区域（plot）内显示数据。

■ 格式　　PLTDSP 显示控件，X轴坐标，Y轴坐标

■ 使用范例　　PLTDSP .BUHIN.GRAPH, 15, 30

■ 说明

- PLTDSP 指令在区域图中用坐标点显示数据。
- 显示控件为控件名称或其ID变量名称。
- X轴坐标和Y轴坐标表示要在图中显示的坐标数值。
- 如果控件原来的参数设置为有效，则这里设置的坐标值无效。

■ 相关项目　　PLTCOLOR

■ 程序实例

```
conf
   static name@
   name@ = ..PLT000
end conf
evnt
   input type%, id@, x%,y%
   if type% = 3 then
      PLTDSP name@, x%, y%
   endif
end evnt
```

PMODE

指令

■ **功能** PMODE 指令用于改变指定部品的状态。

■ 格式 PMODE 部品名称, 模式

■ 使用范例 PMODE .BUHIN., 3

■ 说明
- 部品名称指需要改变状态的部品的名称或能够代表它的ID变量。
- 表示状态（模式）的代号如下：
  0: 正常状态（或模式）；
  1: 禁止开关输入；
  2: 半透明状态。

■ 相关项目 PSTAT

■ 程序实例

```
evnt
   input type% , id@ , data%
   if pstat(.BUHIN.) = 0 then
      PMODE .BUHIN., 1
   endif
end evnt
```

PRDSP

指令

■ **功能** PRDSP 指令指令重新显示指定的控件。

■ 格式 PRDSP 控件名

■ 使用范例 PRDSP .BUHIN.PRIM

■ 说明 • 控件名指指用于显示的控件的名称或其ID变量名。

■ 相关项目 BARSET, CIRSET, BLTSET, LNESET

■ 程序实例

```
evnt
   lneset  .buhin.graph , 3 , 8 , 20.1
   lneset  .buhin.graph , 3 , 8 , 20.1
   PRDSP .buhin.graph
end evnt
```

PREVJUMP

指令

■ **功能**　　执行PREVJUMP指令跳到当前画面的前一显示画面。

■ 格式　　PREVJUMP

■ 使用范例　　PREVJUMP

■ 说明

- PREVJUMP 指令，根据画面跳转路径记录，执行本指令后跳到此前显示的一幅画面。
- 最多可以记录30 幅画面，PREJUMP指令不能跳到登记号在30 幅画面以前的画面。

■ 相关项目　　JUMP

■ 程序实例

```
conf
end conf
evnt
   input type% , id@ , data%
   if id@ = ..SWT000 then PREVJUMP
end evnt
```

PRINT

指令

■ **功能**　　PRINT 用来发送消息。

■ 格式　　PRINT 表达式1 [, 表达式2] ......

■ 使用范例　　PRINT 23, “ABCD”, XYZ, MOJI$

■ 说明

- PRINT 指令用来向画面、部品、串行口、并行口等写入消息。
- 当写入的消息不止一条时，消息之间用逗号（“,”）隔开。
- 当执行PRINT指令时，消息并不马上输出，而是要在执行其后面的SEND指令之后才开始输出。
- 当消息被发送给上位机时，要用逗号将数据隔开。

■ 相关项目　　INPUT, SEND

■ 程序实例

```
evnt
   input type% , id@ , data%
   if type% = 3 then
      PRINT “ABCD”, data%
      send .B000.
   endif
end evnt
```

PRMCTL

指令

■ **功能**　　PRMCTL 用于改变控件的属性。

■ 格式

PRMCTL1 控件名, 要求代码，控件值-1
PRMCTL2 控件名称, 要求代码，type-1, 控件值-1
PRMCTL3 控件名称, 要求代码，type-1, 控件值-2
PRMCTL4 控件名称, 要求代码，type-1, type-2, 控件值-1

■ 使用范例

```
PRMCTL1 ..NUM000,  PD STAT, 3
PRMCTL2 ..NUM000,  PD DCOLOR, 3, 4
PRMCTL3 ..LNE000,  PD RANGE, 0, 2.5
PRMCTL4 ..BAR000,  PD PTRN, 1, 0, 12
```

■ 说明

・变更被指定的控件的属性。
变更有PRMCTL1 ~ PRMCTL4的4种。
・控件名中有表示控件的常数，
或者是表示控件ID的ID型变量。
・要求代码指定进行怎样的属性变更。
从下一页开始展示那个种类。
・种类1，种类2根据请求码的不同值也不同。
・控制值1设定与请求代码对应的值。
整数型的常数，或者变量。
・控制值2设定与请求代码对应的值。
浮点型的常数，或者变量。

■ 相关项目　　PRMCTL1 PRMCTL2, PRMCTL3, PRMCTL4, PRMSTAT1 PRMSTAT2, PRMSTAT3, PRMSTAT4

■ 程序实例

```
conf
end conf
evnt
   status% = prmstat1(..NUM000,  PD STAT)
   if status% = 0 then
      PRMCTL1 ..NUM000,  PD STAT, 2
   endif
end evnt
```

• 可以被PRMCTL1 指令使用的“要求代码”的类型和用法解释如下：

1. PD STAT

功能: PD STAT 用于改变控件的显示形式(正常/反转/闪烁/点灭)

使用范围： PD STAT 适用于所有的控件

控件值： 表示显示形式的代码如下：

0: 正常显示
1: 反转显示
2: 闪烁显示
3: 点灭显示

2. PD DSPFMT

功能: PD DSPFMT 用于改变控件的显示形式。

使用范围： PD DSPFMT 适用于数字和字符显示控件。

控件值： 控件值决定于是使用数字显示控件还是字符显示器

数字显示器	字符显示器
0: 浮点数表示	0: 左边对其显示
1: 整数表示	1: 居中显示
2: 固定小数位置表示	2: 右边对齐显示
3: 二进制不定小数位置显示	
4: 二进制表示	
5: 八进制表示	
6: 十六进制表示	

3. PD PTPOS

功能: PD PTPOS 改变小数点位置

使用范围： PD PTPOS仅对数字显示器有效

控件值： 设置一个表示小数点位置的值，当该值为负数时，小数位置将被强制为0。

4. PD ZSPRS

功能: PD ZSPRS 压缩 0 操作。

使用范围： PD ZSPRS 仅适用于数字显示控件。

控件值： 当不压缩时设置为0，要进行压缩时设置为1。

5. PD FIGMD

功能: PD FIGMD 用来设置显示的图形是否采用放缩的方法使其大小同图形显示器相匹配。

使用范围： PD FIGMD 仅适用于图形显示控件。

控件值： 当不需要时为0，需要时为1。

6. PD WSIZ

功能: PD WSIZ 用于改变点的大小和线的粗细。

使用范围: PD WSIZ 适用于点坐标显示（plot），仪表显示（meter），管道显示（pipe displays）。

控件值: 对于点坐标显示（plot display），用0～2来设置点的大小；对于仪表显示，使用0～2来设置线的宽度；对于管道显示器，使用0～3来表示管道的粗细（1，3，5，7）。

7. PD PIPSTAT

功能: PD PIPSTAT 用于改变指示灯或管道显示的ON/OFF状态。

使用范围: PD PIPSTAT 仅适用于指示灯和管道显示控件。

控件值: ON / OFF 状态的的数字表示如下：

指示灯	管道显示
0: OFF	0: OFF
1: ON	1: ON1
	2: ON2

8. PL FIRST

功能: PL FIRST 用于改变图形显示或文本显示的起始注册号。

使用范围: PL FIRST 适用于文本和图形显示器控件。

控件值: 将值为你希望的起始注册号。

9. PL SMPMSG

功能: PL SMPMSG 用来设置当控件进行采样时，控件是否向其所在的部品发送消息。

使用范围: PL SMPMSG 适用于坐标图显示（plot display）、棒图显示（bar graph）、和趋势图显示控件。

控件值: 当要发送消息时，将其设置为1，否则设置为0。

10. PL SMPCTL

功能: PL SMPCTL 控制采样（停止、启动、复位）（“Stop”，“start”，和 “reset”）

使用范围: PL SMPCTL 适合于坐标图显示（Plot garph）、棒形图（bar graph）、趋势图（line chart）显示器。

控件值: “Stop” 停止采样， “Start” 从停止状态启动开始采样，“Reset” 将所有的结果清除，从头开始采样。

0: 停止采样

1: 启动采样

2: 采样复位

11. PL SMPTME

功能: PL SMPTME 改变采样时间（周期）。

使用范围: PL SMPTME适合于坐标图显示（Plot garph）、棒形图（bar graph）、趋势图（line chart）显示器。

控件值: 设置表示采样时间的数值（采样时间为：设置值×0.5s），当采样时间变化时，采样在复位后重新启动。

12. PL DIRECT

功能: PL DIRECT 用于改变趋势图的显示方向。

使用范围: PL1 DIRECT 仅适用于趋势图显示控件。

控件值: 当显示方向由右向左时，设定为0；当显示方向由左向右移动时，设置为1。当没进行采样时，移动方向就没有什么意义了。当显示方向变化时，采样在复位后重新启动。

13. SW RACT

功能: SW RACT 用来设置当开关ON时是否进行翻转操作。

使用范围: SW RACT 仅适用于开关和选择开关控件。

控件值: 当要求开关翻转时设定为1，否则设置为0。

14. SW BZER

功能: SW BZER 用来设置当开关按下时蜂鸣器是否发出声音。

使用范围: SW BZER 仅适用于开关和选择开关控件。

控件值: 当要求发出声音时设定为1，否则将其设定为0。

15. SW STAT

功能: SW STAT 用来改变开关的状态。(正常操作状态/ 禁止输入状态/ 半透明状态)。

使用范围: SW STAT 仅适用于开关和选择开关控件。.

控件值: 开关的状态代号如下：

0: 正常状态；

1: 禁止输入状态；

2: 半透明状态。

16. SW BMODE

功能: SW BMODE 改变开关的背景颜色和显示方法。

使用范围: SW BMODE仅 适用于开关和选择开关控件。

控件值: 当为直接显示时，设置为0，如果要采用替代方式显示时，将其设置为1。

17. SW ONCOLOR

功能: SW ONCOLOR 用来设置开关的ON背景。

使用范围: SW ONCOLOR 仅适用于开关和选择开关控件。

控件值: 开关背景代号为0～15。

18. SW OFFCOLOR

功能: SW OFFCOLOR 用来设置开关OFF时的背景颜色。

使用范围: SW OFFCOLOR适用于开关和选择开关控件。

控件值: 开关背景代号为0～15。

19. SW ONOFF

功能: SW ONOFF 用来改变开关的ON/OFF状态(注意:在选中“synchronous and operation” 时执行本指令，系统会出现错误）。

使用范围: SW ONOFF 适用于开关和选择开关控件。

控件值: 对于普通开关，当要将开关变为OFF时，可以将其设置为0，置ON时将其设置为1;对于选择开关,要将所有的开关设置为OFF时设置为0，要将某个开关设置为ON时，只要将相应的号码。

• 可以被PRMCTL2 指令使用的“要求代码”的类型和用法解释如下：

1. PD DCOLOR

功能: PD COLOR 用来改变控件的显示颜色。

使用范围: PD DCOLOR 本指令适用于数字显示、字符显示、时钟显示、坐标图显示、自由图显示、仪表显示和指示灯显示控件。

类型: 指定如下一种：

0: 外形（figure）变化。

1: 前面颜色变化

2: 后面颜色变化

3: 显示颜色变化

控件值: 颜色代号可以为0～15中的一个。

2. PD BCOLOR

功能: PD BCOLOR 用来改变控件的背景颜色

使用范围: PD BCOLOR本指令适用于数字显示、字符显示、时钟显示、坐标图显示、棒图显示、趋势图显示、自由图显示控件。

类型: 指定为如下一种：

0: 外形（figure）变化。

1: 前面颜色变化

2: 后面颜色变化

控件值: 颜色代号可以为0～15中的一个。

3. PD PIPCOLOR

功能: PD PIPCOLOR 用来改变管道和指示灯显示的内部颜色。

使用范围: PD PIPCOLOR 适用于管道显示器和指示灯显示器控件。

类型: 指定为下面一种：

0: 改变OFF 时的显示颜色。

1: 改变ON1时的显示颜色。

2: 改变ON2 时的显示颜色。(仅对管道显示器)

控件值: 颜色代号可以为0～15中的一种。

4. PD BSLNE

功能: PD BSLNE 用来改变基准线或参考线的类型。

使用范围: PD BSLNE 适用于棒形图和趋势图显示。

类型: 指定如下一种：

0: 改变基准线

1: 改变参考线1

2: 改变参考线2

控件值: 线型代号为0～3。

5. PD BSCOLOR

功能: PD BSCOLOR 用于改变基准线或参考线的颜色。

使用范围: PD BSCOLOR 适用于棒形图和线状趋势图控件。

类型: 指定如下一种：

0: 改变基准线颜色

1: 改变参考线1颜色

2: 改变参考线2颜色

控件值: 颜色代号可以为0～15中的一种。

6. SW ONFIG

功能: SW ONFIG 用于改变开关ON时的背景图形。

使用范围: SW ONFIG适用于开关或选择开关。

类型: 对于普通开关，指定为1。对于选择开关，指定需要改变ON时背景图形的开关位置编号，起始为1。

控件值: 指定希望的构件注册号。

7. SW OFFFIG

功能: SW OFFFIG 用于改变开关OFF时的背景图形。

使用范围: SW OFFFIG适用于开关或选择开关。

类型: 对于普通开关，指定为1。对于选择开关，指定需要改变OFF时背景图形的开关位置编号，起始为1。

控件值: 指定希望的构件注册号。

8. PD PLOTRNG

功能: PD PLOTRNG 用于改变折线图显示的起始节点/终了节点。

使用范围: PD PLOTRNG 适用于折线图。

类型: 指定如下一种：

0: 以起始节点为基准

1: 以终了节点为基准

控件值: 可指定的节点号范围为：0~（折线图最大节点数-1）。

• 可以被PRMCTL3 指令使用的“要求代码”的类型和用法解释如下：

1. PD Range
   功能: PD Range 用来设置控件的显示范围
   使用范围: PD range 适用于棒图、趋势图、自由图、滑动图（Slide）、仪表和坐标显示图（Plot display）控件。
   类型: 当使用坐标显示图（Plot display）控件时，可用0（X轴最小值变动）、1（X轴最大值变动）、2（Y轴最小值变动）、3（Y轴最大值变动）。其它控件使用2（最小值变动）、3（最大值变动）。
   控件值: 要改变的值的范围（显示范围）。

2. PD BSVAL
   功能: PD BSVAL 改变参考线和基准线的值。
   使用范围: PD BSVAL 仅对棒形图和趋势图控件有效。
   类型: 0（基准线变化），1（参考线1变化），2（参考线2变化）.
   控件值: 设置目标值

• 可以被PRMCTL4 指令使用的“要求代码”的类型和用法解释如下：

1. PD PTRN
   功能: PD PTRN 设置控件的显示颜色
   使用范围: PD PTRN 适用于棒图、百分比棒形图、和饼图显示控件。
   Type-1: 指定棒图或饼图扇形编号。
   Type-2: 要改变的属性：
   0: 背景图
   1: 前面颜色变化
   2: 后面颜色变化
   控件值: 颜色范围在0～15。

2. PD LNE
   功能: PD LNE 改变趋势图显示的颜色。
   使用范围: PD LNE 仅适合于趋势图显示控件。
   Type-1: 指定要改变颜色的曲线编号。
   Type-2: 指定如下某个值：
   0: 线型变化
   1: 颜色变化
   控件值: 颜色范围在0～15。

PRMSTAT

函数

■ **功能**　　PRMSTAT 函数用于读取指定控件的属性。

■ 格式

返回值-1 = PRMSTAT1 (控件名称, 要求代码)
返回值-1 = PRMSTAT2 (控件名称, 要求代码，type-1)
返回值-2 = PRMSTAT3 (控件名称, 要求代码，type-1)
返回值-1 = PRMSTAT4 (控件名称, 要求代码，type-1, type-2)

■ 使用范例

```
VAL% = PRMSTAT1 (..NUM000,  PD STAT)
VAL% = PRMSTAT2 (..NUM000,  PD DCOLOR, 3)
VALF! = PRMSTAT3 (..LNE000,  PD RANGE, 0)
 VAL% = PRMSTAT4 (..BAR000,  PD PTRN, 1, 0)
```

■ 说明

・读取指定的控件属性。
变更分为PRMSTAT1 ~ PRMSTAT4四种。
・控件名中有表示控件的常数或控件的ID
的ID型变量。
・要求代码指定进行什么样的属性的读入。
从下一页开始展示那个种类。
・种类1，种类2根据请求码的不同值也不同。
・返回值1是对应请求代码的函数的返回值。
整数型的常数，或者变量。
・返回值2是对应请求代码的函数的返回值。
浮点型的常数，或者变量。

■ 相关项目　　PRMCTL1, PRMCTL2, PRMCTL3, PRMCTL4, PRMSTAT1, PRMSTAT2, PRMSTAT3, PRMSTAT4

■ 程序实例

```
conf
end conf
evnt
   status% = prmstat1(..NUM000,  PD STAT)
   if status% = 0 then
      PRMCTL1 ..NUM000,  PD STAT, 2
   endif
end evnt
```

- 可以被PRMCTL1 指令使用的“要求代码”的类型和用法解释如下：

1. PD NUMS

   功能: PD NUMS 读取控件元素的编号。

   使用范围: PD NUMS 适用于所有的控件

   返回值-1: 设置表示显示形式的数值。当显示形式不是连续控件方式时，该返回值恒为1。

2. PD ROTATE

   功能: PD ROTATE 读取控件的旋转角度。

   使用范围: PD ROTATE 适用于所有的显示控件。

   返回值-1: 当旋转角度为0时，返回值为0；当旋转角度为90度时，返回值为1；当旋转角度为180度时，返回值为2；当旋转角度为270度时，返回值为3。对于饼图、仪表图、指示灯、管道显示图，返回值通常为0。

3. PD STAT

   功能: PD STAT 读取控件的显示形式（正常/反色/闪烁/点灭）。

   使用范围: PD STAT 适用于所有的控件。

   返回值: 返回值为如下之一：

   0: 正常显示
   1: 反色显示
   2: 闪烁显示
   3: 点灭显示

4. PD DSPFMT

   功能: PD DSPFMT 读取控件的显示形式。

   使用范围: PD DSPFMT 适用于数字和文本显示控件。

   返回值-1: 返回值取决于是使用数值显示器还是文本显示器：

数值显示器	文本显示器
0: 浮点型表示	0: 靠左边显示
1: 整数表示	1: 居中显示
2: 固定小数点表示	2: 靠右边显示
3: 二进制方式固定小数点表示	
4: 二进制表示	
5: 八进制表示	
6: 十六进制表示	

5. PD DATFMT

   功能: PD DATFMT 显示数据的形式

   使用范围: PD DATFMT 适用于除时钟显示控件以外的所有控件。

   返回值: 对于实数，返回值为0；对于整数，返回值为1；对于无符号整数，返回值为2；对于BCD码数，返回值为3。对于指示灯控件，返回值通常为2。

6. PD FONT

功能: PD FONT 读取控件上显示的字体。

使用范围: PD FONT 适用于数字和时钟显示控件。

返回值: 对于半角文字显示，返回值为0；对于全角文字显示，返回值为1。

7. PD XFSZ

功能: PD XFSZ 用来读取控件上显示文字在水平方向上的大小。

使用范围: PD XFSZ 适用于数字、文字及时钟显示控件。

返回值: 放大倍数为1时，返回值为0；放大倍数为2时，返回值为1；放大倍数为4时，返回值为3；放大倍数为16时，返回值为4。

8. PD YFSZ

功能: PD YFSZ 用来读取控件上显示文字在竖直方向上的大小。

使用范围: PD YFSZ 适用于数字、文字及时钟显示控件。

返回值: 放大倍数为1时，返回值为0；放大倍数为2时，返回值为1；放大倍数为4时，返回值为3；放大倍数为16时，返回值为4；当放大倍数为32时，返回值为5。

9. PD PTPOS

功能: PD PTPOS 读取小数点的位置。

使用范围: PD PTPOS 仅适用于数字显示控件。

返回值: 返回值为小数点的位置值。

10. PD ZSPRS

功能: PD ZSPRS 读取是否对零进行压缩。

使用范围: PD ZSPRS 仅适用于数字显示控件。

返回值: 当不对零进行压缩时，返回值为0，当不对零进行压缩时，返回值为0，当对零进行压缩时，返回值为1。

11. PD XNUM

功能: PD XNUM 读取水平方向的显示位数。

使用范围: PD XNUM 适用于数字和文本显示控件。

返回值: 返回值为水平方向可以显示的半角字符个数。

12. PD YNUM

功能: PD YNUM 读取竖直方向的显示位数。

使用范围: PD YNUM 仅适用于字符显示控件。

返回值: 返回值为竖直方向可以显示的字符个数。

13. PD DIRECT

功能: PD DIRECT 读取字符的显示方向。

使用范围: PD DIRECT 仅适用于字符串显示控件。

返回值: 当为水平书写时，返回值为0；当为竖直书写时，返回值为1。

14. PD PLTNUM

功能: PD PLTNUM 读取控件最多可以显示的坐标点数。

使用范围: PD PLTNUM 适用于坐标显示图和趋势图显示控件。

返回值: 返回值为可以显示的最多点数。

15. PD LNENUM

功能: PD LNENUM 读取棒图或趋势图可以显示的棒条或曲线的数量。

使用范围: PD LNENUM 适用于棒形图显示和趋势图显示控件。

返回值: 返回值为棒条数或曲线条数。

16. PD ZNNUM

功能: PD ZNNUM 读取控件中的区域数。

使用范围: PD ZNNUM 适用于饼图或百分棒图显示控件。

返回值: 返回值为控件可以显示的区域数量。

17. PD FIGMD

功能: PD FIGMD 读取内容为是否采用放缩的方式使图形大小自动与图形显示器大小相匹配。

使用范围: PD FIGMD 仅适用于图形显示控件（Texture display）。

返回值: 当自动放缩时值为1，如果不需要时，返回值为0。

18. PD WSIZ

功能: PD WSIZ 读取控件中显示点/线的大小。

使用范围: PD WSIZ 适用于坐标显示控件、仪表显示控件和管道显示控件。

返回值: 对于坐标显示控件，表示点大小的数值为0～2；对于仪表显示控件，表示指针粗细的数值为0～2；对于管道显示控件，表示管道粗细的数值为0～3（分别表示1、3、5、7）。

19. PD PIPSTAT

功能: PD PIPSTAT 读取指示灯或管道显示器的 ON / OFF 状态。

使用范围: PD PIPSTAT 适用于指示灯控件和管道显示控件。

返回值: 表示指示灯和管道ON/OFF状态的值对应如下：

指示灯显示器	管道显示器
0: OFF	0: OFF
1: ON	1: ON1
	2: ON2

20. PL NUMS

功能: PL NUMS 读取正在使用的设备数量。

使用范围: PL NUMS 适用于除了时钟显示控件之外的所有控件。

返回值: 返回值为正在使用的设备数量。（当数字显示器为双字时，设备数量要翻倍）

21. PL FIRST

功能: PL FIRST 读取显示图形（构件,Texture）或显示文本（Text）的其实注册号。

使用范围: PL FIRST 适用于文本显示控件和图形显示控件。

返回值: 返回值为要显示内容的起始注册号。

22. PL DVTYP

功能: PL DVTYP 读取控件正在使用的的设备的类型。

使用范围: PL DVTYP 仅适用于数字显示控件。

返回值: 对于双字，返回值为0；对于单字，返回值为1。

23. PL ENDI

功能: PL ENDI 读取双字的显示方式。

使用范围: PL ENDI 仅适用于数字显示控件。

返回值: 当双字的显示是从下往上时，返回值为0；当从上往下时，返回值为1。

24. PL SMPMSG

功能: PL SMPMSG 用来读取“当放在部品上的控件进行采样时，是否向部品发送消息”。

使用范围: PL SMPMSG 适用于坐标显示控件、棒图显示控件、和趋势图显示控件。

返回值: 当采样时要发送消息时，返回值为1，否则为0。

25. PL SMPTME

功能: PL SMPTME 用来读取采样时间。

使用范围: PL SMPTME 适用于坐标显示（Plot display）、棒图显示(Bar graph display)、和趋势图显示(Line graph display)控件。

返回值: 返回值为表示采样间隔（值*0.5秒）的设定值。

26. PL DIRECT

功能: PL DIRECT 读取趋势图的移动方向。

使用范围: PL1 DIRECT 仅适合于趋势图显示控件。

返回值: 当趋势图从左向右移动时，返回值为0；当显示从右向左时，返回值为1。

27. SW NUMS

功能: SW NUMS 用来读取开关中开关元素的数量。

使用范围: SW NUMS 适用于开关和选择开关。

返回值: 对于普通开关，返回值为1；对于选择开关，返回值为开关元素的数量。

28. SW TYPE

功能: SW TYPE 读取开关的类型。

使用范围: SW TYPE 适用于开关和选择开关。

返回值: 对于点动开关，返回值为0；对于翻转开关，返回值为1；对于自动翻转开关，返回值为2；对于选择开关，返回值为3。

29. SW ONCOLOR

功能: SW ONCOLOR 读取开关ON时的背景颜色。

使用范围: SW ONCOLOR 适用于开关和选择开关。

返回值: 返回值为0～15之间的某个值。

30. SW OFFCOLOR

功能: SW OFFCOLOR 读取开关OFF时的背景颜色。

使用范围: SW OFFCOLOR 适用于开关和选择开关。

返回值: 返回值为0～15之间的某个值。

31. SW BMODE

功能: SW BMODE 读取开关背景颜色的显示方法。

使用范围: SW BMODE 适用于开关和选择开关。

返回值: 当开关的背景颜色显示方法为“直接显示（direct display）”时，返回值为0；当显示方法为“替换显示（replacement display）”时，返回值为1。

32. SW RACT

功能: SW RACT 读取当开关为ON时是否进行翻转操作。

使用范围: SW RACT 适用于开关和选择开关。

返回值: 当开关为ON时翻转，则返回值为1；否则为0。

33. SW BZER

功能: SW BZER 读取当开关按下时是否发出声音。

使用范围: SW BZER 适用于开关和选择开关。

返回值: 当按下开关时蜂鸣器发出声音，返回值为1，如果不发出声音，返回值为0。

34. SW STAT

功能: SW STAT 读取开关的状态(正常操作 / 禁止输入 / 半透明)。

使用范围: SW STAT 适用于开关和选择开关。

返回值: 返回值为下列之一：

0: 正常操作状态

1: 禁止开关输入状态

2: 半透明状态

35. SW ONOFF

功能: SW ONOFF 读取开关的ON/OFF状态。

使用范围: SW ONOFF 适用于开关和选择开关。

返回值: 当开关处于OFF状态时，返回值为0。当开关处于ON状态时，返回值为1。当所有的开关都处于OFF状态时，返回值为0，当某个开关处于ON状态时，返回值为相应的开关编号。

36. SL SYNC

功能: SL SYNC 读取开关显示的同步性。

使用范围: SL SYNC 适用于开关和选择开关。

返回值: 当不同步时返回值为0，当同步时返回值为1。

37. SL BORW

功能: SL BORW 读取开关的写入方法。

使用范围: SL BORW 适用于选择开关。

返回值: 当开关设备的写入方法为位写入时,返回值为0;当为字写入方式时,返回值为1。

- 可以被PRMCTL2 指令使用的“要求代码”的类型和用法解释如下：

1. PD DCOLOR
   功能: PD COLOR 读取控件的显示颜色。
   使用范围: PD DCOLOR 适用于数字显示、文本显示、时钟显示、坐标图显示、自由图显示、仪表显示和指示灯显示控件。
   类型: 类型为下列其一：
   0: 读取背景图形（Figure）。
   1: 读取前色
   2: 读取背景色
   3: 读取显示颜色
   返回值: 返回的颜色值可能为0～15中的一个。

2. PD BCOLOR
   功能: PD BCOLOR 读取控件的背景颜色。
   使用范围: PD BCOLOR 适用于数字显示、文本显示、时钟显示、坐标图显示、自由图显示、仪表显示和指示灯显示控件。
   类型: 指定如下一个：
   0: 读取背景图形（Figure）。
   1: 读取前色
   2: 读取背景色
   返回值: 返回的颜色值可能为0～15中的一个。

3. PD PIPCOLOR
   功能: PD PIPCOLOR 读取管道或指示灯的内部颜色。
   使用范围: PD PIPCOLOR 适用于管道或指示灯显示控件。
   类型: 指定如下一个：
   0: 读取OFF时的颜色（仅对管道和指示灯控件有效）。
   1: 读取 ON1 时的颜色。（仅对管道和指示灯控件有效）。
   2: 读取 ON2 时的显示颜色 (仅对管道显示控件有效)
   返回值: 返回的管道内部颜色可能为0～15中的某个。

4. PD BSLNE
   功能: PD BSLNE 读取基准线和参考线的线型。
   使用范围: PD BSLNE 适用于棒图显示控件和趋势图显示控件。
   类型: 指定要读取的对象：
   0: 读取基准线线型
   1: 读取参考线1线型
   2: 读取参考线2线型
   返回值: 返回值为线型代号0～3。

5. PD BSCOLOR
   功能: PD BSCOLOR 读取基准线和参考线的颜色。
   使用范围: PD BSCOLOR 适用于棒图显示控件和趋势图显示控件。
   类型: 指定要读取的对象：
   0: 读取基准线颜色
   1: 读取参考线1颜色
   2: 读取参考线2颜色

返回值:　　返回值为颜色代号可能为0～15中的某个。

6. SW ONFIG

功能:　　SW ONFIG 读取开关为 ON 时背景的显示构件

使用范围:　　SW ONFIG 适用于开关和选择开关。

类型:　　对于开关，指定为1；对于选择开关，为状态为1的开关的编号。编号从1开始。

返回值:　　返回值为背景构件编号。

7. SW OFFFIG

功能:　　SW OFFFIG读取开关为 OFF 时背景的显示构件

使用范围:　　SW OFFFIG适用于开关和选择开关。

类型:　　对于普通开关，指定为1。对于选择开关，指定为状态为OFF的开关编号。

返回值:　　返回值为背景构件编号。

8. SL WRITE

功能:　　SL WRITE 读取开关写入时的值。

使用范围:　　SL WRITE 仅适用于开关控件。

类型:　　要读取当开关为ON时的写入值时，设定为1，否则设定为0。

返回值:　　返回开关写入时的值。

9. PD PLOTRNG

功能:　　PD PLOTRNG 读取趋势图显示曲线的起始点和终点。

使用范围:　　PD PLOTRNG 仅适用于趋势图显示控件。

类型:　　指定方法如下：

0:　表示读取显示的起始点。

1:　表示读取显示的终点。

- 可以被PRSTAT3 指令使用的“要求代码”的类型和用法解释如下：

1. PD Range
   功能: PD Range 读取控件的显示范围。
   使用范围: PD Range 适用于棒图显示控件、趋势图显示控件、自由图显示控件、滑动图显示、仪表显示和坐标显示控件。
   类型: 当读取X轴的最小值时，指定为0；读取X轴的最大值时，指定为1；当读取Y轴的最小值时，指定为2；读取Y轴显示最大值时，指定为3。
   返回值: 返回值为指定的显示范围。

2. PD BSVAL
   功能: PD BSVAL 读取基准线和参考线的设定值。
   使用范围: PD BSVAL 适用于棒图显示和趋势图显示控件。
   类型: 当改变基准线时，指定为0；当改变参考线1时指定为1；当改变参考线2时指定为2。
   返回值: 返回值为表示显示范围的数值。

- 可以被PRMSTAT4 指令使用的“要求代码”的类型和用法解释如下：

1. PD PTRN
   功能: PD PTRN 读取控件的显示颜色。
   使用范围: PD PTRN适用于棒形图、百分比棒图、和饼图显示控件。
   Type-1: 指定要改变部分的编号。
   Type-2: 指定值为如下某个：
   0: 背景构件读取
   1: 前面色读取
   2: 后面色读取
   返回值: 返回值为表示构件和颜色的代号。

2. PD LNE
   功能: PD LNE 读取趋势图的显示颜色。
   使用范围: PD LNE 适用于趋势图显示控件。
   Type-1: 指定要改变颜色的曲线的编号。
   Type-2: 设定值如下：
   0: 读取线型
   1: 读取曲线颜色
   返回值: 返回值为曲线线型或曲线颜色。

PSTAT

函数

■ **功能** PSTAT 函数读取指定部品的状态模式。

■ 格式 PSTAT (部品名称)

■ 使用范例 MODE = PSTAT (.BUHIN.)

■ 说明

- PSTAT 函数用于括号里部品的状态。
- 部品名称即要读取状态的部品的名称或能表示它的ID变量名。
- 状态模式代号如下：
  1: 禁止输入状态
  2: 半透明状态
  3: 关闭状态

■ 相关项目 PMODE

■ 程序实例

```
evnt
   input type% , id@ , data%
   if PSTAT(.BUHIN.) = 0 then
      pmode .BUHIN., 1
   endif
end evnt
```

RANGE

指令

■ **功能**　　使用范围 指令用于改变数值显示控件显示数值的最大最小值范围。.

■ 格式　　使用范围 控件名称, area-1, area-2, area-3, area-4

■ 使用范例　　使用范围 ..GRAPH, 0, 0, 100, 100

■ 说明

- 控件名称可以是图形显示器名称也可以是能代表图形显示器的ID变量名。
- 控件里能表示的最大最小值范围设定方法如下表所示：

	Area 1	Area 2	Area 3	Area 4
绘图显示	横向最小值	横向最大值	纵向最小值	纵向最大值
棒图显示	最小值	最大值	基准值	-
折线图显示	最小值	最大值	-	-
自由图显示	最小值	最大值	-	-
滑动图形显示	最小值	最大值	-	-
仪表显示	最小值	最大值	-	-

“-”：无

■ **相关项目**　　None

■ **程序实例**

```
evnt
   input type% , id@ , min%,max%
   if type% = 3 then
      RANGE ..MTR000 , min%, max%,0,0
   endif
end evnt
```

READTIM

函数

■ **功能**　READTIM 函数读取指定定时器的当前值。

■ 格式　READTIM (定时器号)

■ 使用范例

```
DD = READTIM (TNO@)
DD = READTIM (VAR)
```

■ 说明

- READTIM 函数读取定时器的当前计时时间，单位为0.1s（即100ms）。
- 定时器号为0～15之间的一个整数。

■ 相关项目　OPENTIM, STARTTIM, STOPTIM, CLOSETIM, CONTTIM, WRITETIM

■ 程序实例

```
conf
   static timid@
   timid@ = OPENTIM()
   settim timid@, 20, 0
   starttim timid@
end conf
evnt
   input type% , id@ , data%
   if type% = 3 then
      tim% = READTIM(timid@)
      numdsp ..NUM000,tim%*100
   end if
end evnt
```

RENAME

指令

■ **功能** RENAME 指令用于改变文件名或文件夹名称。

■ 格式 RENAME 原文件名, 新文件名

■ 使用范例 RENAME "E:\SUBDIR\FILE1", "FILE2"

■ 说明
- 原文件名可以由包括驱动器名在内的整个文件路径指出，也可以直接由当前文件夹指定。
  例如: E:\SUBDIR\FILE1
- 新文件名不允许包括路径名称。
- 要改变文件夹名称，则用文件夹名称代替文件名。

■ 相关项目 FOPEN,KILL,MKDIR,

■ 程序实例

```
conf
   global dname$(13), pname1$(13), pname2$(13), pname3$(13)
   global dsel%, p1sel%, p2sel%, p3sel%
   strdsp ..str, "rename"
end conf
evnt
   input type%, id@, data%
   if data% = 1 then
      path$ = dname$(dsel%) + pname1$(p1sel%) + pname2$(p2sel%)
      strdsp .dsp.str, path$
      rename path$, pname3$(p3sel%)
   end if
end evnt
```

REOPENCOM

指令

■ **功能** REOPENCOM 指令将临时关闭的串口打开。

■ 格式 REOPENCOM 设备逻辑名称

■ 使用范例 REOPENCOM HST

■ 说明

- REOPENCOM 指令使得被CLOSECOM指令临时关闭的串口可以重新从外设接受数据。
- 设备逻辑名称指如下外设之一：
  - HST: Host computer（上位机）
  - BCR: Bar code reader（条形码读入机）
  - TKY: Ten-key pad （十键键盘）

■ 相关项目 OPENCOM, CLOSECOM

■ 程序实例

```
conf
   OPENCOM HST
end conf
evnt
   input type% , id@ , data%
   if type% = 3 and data% = 1 then
      CLOSECOM HST
   else if type% = 3 and data% = 0 then
      REOPENCOM HST
   endif
end evnt
```

REOPENPARALLEL

指令

■ **功能** REOPENPARALLEL 指令将临时关闭的并行口打开。

■ 格式 REOPENPARALLEL 输入位

■ 使用范例 REOPENPARALLEL 3

■ 说明
- REOPENPARALLEL将使用CLOSEPARALLEL指令暂时关闭的并行口重新打开，使之可以接收数据。
- 输入位指要重新启动数据输入的输入位。输入位同在这以前使用CLOSEPARALLEL指令关闭的位相同。

■ 相关项目 OPENPARALLEL, CLOSEPARALLEL

■ 程序实例

```
conf
   OPENPARALLEL 3
end conf
evnt
   input type% , id@ , data%
   if type% = 3 and data% = 1 then
      CLOSEPARALLEL 3
   else if type% = 3 and data% = 0 then
      REOPENPARALLEL 3
   endif
end evnt
```

RESETALARM

指令

■ **功能** RESETALARM 指令将指定的报警复位。

■ 格式 RESETALARM 报警编号

■ 使用范例 RESETALARM (NO@)

■ 说明
- 报警编号指使用SETALARM指令设置的报警编号，它必须是一个ID型变量。
- 当指定的时间到了之后，本指令将已经设置为ON的报警复位。

■ 相关项目 SETALARM

■ 程序实例

```
conf
   static alid@
   alid@ = setalarm(10,0)
end conf
evnt
   input type% , id@ , data%
   if type% = 3 then
      RESETALARM(alid@)
   end if
end evnt
```

RETURN

指令

■ **功能** RETURN 指令将子程序的控制权交还给主程序。

■ 格式 RETURN

■ 使用范例 RETURN

■ 说明 • RETURN 指令将控制权交还给子程序调用指令GOSUB后面的程序。

■ 相关项目 GOSUB

■ 程序实例

```
evnt
   X = 10
   GOSUB SUB001
   numdsp ..NUM000, X
end evnt
SUB001:
   X = X+3
RETURN
```

RIGHT$

函数

■ **功能**

RIGHT$ 函数用来返回一个指定长度的字符串，该字符串从指定字符串的右边开始。

■ 格式

RIGHT$（字符串名，字符数）
RIGHT$（字符串注册号，字符数）

■ 使用范例

RIGHT$（注册字符串名，字符数）

■ 说明

A$ = RIGHT$ (MOJI$, 5)
A$ = RIGHT$ (4, 10)
A$ = RIGHT$ (TOROKU, 8)

■ 相关项目

- RIGHT$ 函数从指定字符串的右边开始返回一个字符串。
- 字符数指要从指定字符串返回字符的数量，字符数以整形数表示，范围在0～255之间。当为0时，返回一个空字符。
- 字符串可以是一个直接字符串或者一个字符串变量名。
- 字符串注册号指在SCA2 注册过的字符串登记号，或数学表达式。
- 注册字符串名可以是注册过的字符串的名称，也可以是能表示该字符串的ID变量。

■ 程序实例

MID$, LEFT$

```
evnt
   b$ = “12345678”
   a$ = RIGHT$(b$ , 3)
   c$ = RIGHT$ (no , 3)
   c$ = RIGHT$ (id@ , 4)
end evnt
```

RMDIR

指令

■ **功能** RMDIR 指令用来删除一个目录

■ 格式 RMDIR 目录名

■ 使用范例 RMDIR “TEST”

■ 说明
- RMDIR 指令用来删除一个子目录。
- 要删除的目录名称用字符串常量或变量表示。
- 要删除的目录名称前面可以加上驱动器名。

■ 相关项目 MKDIR, CHDIR

■ 程序实例

```
conf
end conf
evnt
   ......
   RMDIR “C:TEST”
   ......
end evnt
```

ROTATE

指令

■ **功能**　　ROTATE 将图形显示器（控件）里面的图画旋转一个角度。

■ 格式　　ROTATE 控件名, 旋转角度

■ 使用范例　　ROTATE ..FIG000, 2

■ 说明

- 控件名指图形显示器的名称或能表示该控件的ID型变量。
- 旋转角度指下列表示旋转角度的代号之一：
  0: 旋转 0　度
  1: 旋转 90 度
  2: 旋转 180度
  3: 旋转 270度

■ 相关项目　　FIGDSP

■ 程序实例

```
evnt
   input ty , id@, fig%
   ROTATE ..FIG000 , fig%
end evnt
```

RSTAT

函数

■ **功能** RSTAT 函数用于注册目标的状态。

■ 格式 RSTAT (registration-name,type,option) (注册名称, 类型, 选值)

■ 使用范例 VAR@ = RSTAT (GAMEN1.., 0, 1)

■ 说明

- RSTAT指令用于读取离registration-name 距离为option的画面的注册登记号。
- registration-name(注册名称)可以是代表画面、注册文本名称、注册图形（构件）的变量或常数。
- type 在此固定为0。
- 当“选值”为正时，RSTAT指令往注册号递增的方向检测；当“选值”为负时，往注册号减小的方向检查。
- 如果没有符合条件的目标，则函数返回值为-1。

■ 相关项目 GETGID, GETGNO

■ 程序实例

```
conf
end conf
evnt
   id@ = getgid()
   no% = RSTAT ( id@, 0, 1)              ’ 检测下一注册画面编号。
   if no% <> -1 then jump no%            ’
end evnt
```

RUN

指令

■ **功能**　　RUN 指令用于运行其后面指定的程序。

■ 格式　　RUN 执行部品 / 画面

■ 使用范例　　RUN .BUHIN.

■ 说明

- RUN 指令将消息发送到“执行部品 / 画面”指出的部品或画面，并执行画面或部品的程序。(发送的消息包括消息的type、ID，然而它并不包括要发送的数据)。
- 在发送RUN命令时并不执行程序，只有当RUN指令所在的程序结束后才运行画面或部品程序。
- “执行部品/画面”可以是画面部品名称，或者ID变量。

■ 相关项目　　INPUT, PRINT, SEND

■ 程序实例

```
evnt
   input ty , id@, fig%
   if ty = 3 and id@ = ..SWT000 then
      RUN .B000.
   endif
end evnt
```

SELECT CASE ... END SELECT

指令

■ **功能**　　执行满足条件的程序。

■ 格式

```
SELECT CASE
CASE
    程序1
CASE
    程序2
CASE ELSE
    程序n
END SELECT
```

■ 使用范例　　见下面的 “程序实例” 。

■ 说明

- SELECT CASE 指令执行程序表中满足条件表达式的程序。
- 当CASE 、CASE ELSE、 和 END SELECT在满足给定条件的程序被执行之后出现时，本指令执行END SELECT之后的程序。
- 条件判断可以执行多达50次。

■ 相关项目　　IF ... THEN ... ELSE

■ 程序实例

```
evnt
   input ty , id@, dat%
   select case dat%
      case 1                    ’   When dat% is 1
         aaa = 1
      case 2,3                  ’   When dat% is 2 or 3
         aaa = 2
      case 4 to 10              ’   When dat% is 4 to 10
         aaa = 3
      case else                 ’   When dat% is another value
         aaa =4
   end select
end evnt
```

SEND

指令

■ **功能** SEND 指令将数据送到指定的画面、部品、或连接的逻辑设备。

■ 格式 SEND 发送目标名称

■ 使用范例 SEND .BUHIN.

■ 说明

- SEND 将PRINT指令写入的数据送到指定的目标。
- 发送目标名称是指数据将要发送到的目标画面或部品的名称。它可以是表示名称的ID变量，或如下逻辑设备：
  HST: 上位计算机
  PRN: 打印机
- 画面或部品程序在发送SEND指令时并不执行，只有当SEND指令所在程序结束后才执行。

■ 相关项目 RUN, PRINT

■ 程序实例

```
evnt
   input ty , id@, dat%
   if ty = 3 and id@ = ..SWT000 then
      print “BUHIN1”,dat%
      send .B0000.
   endif
end evnt
```

SENDMAIL

指令

■ **功能**　　通过CG-A2发送电子邮件（EMAIL）。

■ 格式　　SENDMAIL 发送文字

■ 使用范例　　SENDMAIL(2)，SENDMAIL(Alarm1)，SENDMAIL(“TEST MAIL”)，SENDMAIL(MAIL_STRING$)

■ 说明

- 向已登录的邮箱地址发送相应内容的电子邮件。
- 可发送的文字内容如下：
  1.注册文本　注册号
  2.注册文本　注册名
  3.字符串
  4.字符串变量

■ 相关项目

■ 程序实例

```
evnt
    input type%,id@,data%
    if type% = 3 and ..swt000 then
        SENDMAIL(“异常 1 电源电压低下”)
    endif
end evnt
```

SETALARM

指令

■ **功能**　　SETALARM 指令用来设置报警时间。

■ 格式　　SETALARM hour , minute (时钟, 分钟)

■ 使用范例　　ID@ = SETALARM (13, 30)

■ 说明

- SETALARM 指令设置OIP时钟的报警时间（时刻）。当设定的时间到达之后，标示结果的数据将被送往设定的画面和部品。
- hour (时钟) 取值范围在0～23。
- minute （分钟）取值范围在0～59。
- 当执行 SETALARM 函数后，返回报警器编号。返回的报警器编号是一个ID型变量。
- 该函数可以在当前画面或当前画面的部品上使用。

■ 相关项目　　RESETALARM

■ 程序实例

```
conf
   static alid@
   alid@ = SETALARM(10,0)
end conf
evnt
   input type% , id@ , data%
   if type% = 3 then
      resetalarm(alid@)
   end if
end evnt
```

SETBEEP

指令

■ **功能** SETBEEP 指令用来指定蜂鸣器的声音。

■ 格式 SETBEEP ON-time, OFF-time, 发声次数

■ 使用范例 SETBEEP 10, 5, 3

■ 说明
- SETBEEP 用来设置由BEEP命令触发的蜂鸣器的音质。
- ON-time 指定蜂鸣器连续蜂鸣的时间，时间以100ms为单位。
- OFF-time 指定蜂鸣器连续停顿的时间，时间以100ms为单位。
- 发声次数表示蜂鸣器发声、停止互相交替的次数。本参数不能为0。
- 当OFF-time 为0时，蜂鸣器会叫个不停。

■ 相关项目 BEEP

■ 程序实例

```
conf
   SETBEEP 50,20,3
end conf
evnt
   input type%, id@, data%
   if id@ = ..SWT000 then
      BEEP 1
   else
      BEEP 0
   endif
end evnt
```

SETBLIGHT

指令

■ **功能** SETBLIGHT 用来设置背光灯的点亮时间。

■ 格式 SETBLIGHT OFFTIME(关闭时间)

■ 使用范例 SETBLIGHT 20

■ 说明

- 关闭时间用来设置背光灯在关闭前的延时时间。它是一个整形数或一个整型变量。单位为分钟，当关闭时间为0时，背光灯不关闭。

■ 相关项目 GETBLIGHT

■ 程序实例

```
conf
   getblight var
   var = var*2
   SETBLIGHT var
end conf
```

SETDATE

指令

■ **功能** SETDATE 指令用来设置系统的内部时钟。

■ 格式 SETDATE 年, 月, 日

■ 使用范例 SETDATE 02, 5, 8

■ 说明

- 年，取公历年的后两位数，范围在00～99之间。
- 月，范围在1 ～ 12。
- 日期，范围在1 ～ 31。
- 如果设定了不存在地年、月、日，系统将会报错。
- 星期，会根据设定的系统时间自动生成。
- 日期一旦设定，因为触摸屏内部有停电记忆日历芯片，所以即使在断电之后系统时钟也能自动更新。（除早期的GC-53LM外，其它早期型号和最新型号都有停电记忆功能）。

■ 相关项目 DATE\$, GETDATE, GETDATE, SETTIME, TIME\$

■ 程序实例

```
evnt
   input type,id@,dat
   if type = 3 then
      y = 94
      m = 12
      d = 1
      setdate y, m, d
   endif
end evnt
```

SETIMEMODE

函数

■ **功能**　　IME输入法的半角/全角模式切换。

■ 格式　　SETIMEMODE

■ 使用范例　　SETIMEMODE（0）

■ 说明　　• 输入模式全角/半角对应数值：
0：全角模式
1：半角模式
此函数无返回值。

■ 相关项目　　GETKEY,GETKEYT,S2U,U2S

■ 程序实例

```
if type% = 3
  input data%
    if data% = 1 then
      select case id@
        case ..SW_MODE     'IME输入模式
          if ShiftKeyDown% = 1 then
            ShiftKeyDown% = 0
            id@ = ..SW_SHIFT
            SWwrite id@ , 0
          end if
          setimemode(0)    'IME输入模式(全角)
        case ..SW_SHIFT'SHIFT
          if ShiftKeyDown% = 1 then
            ShiftKeyDown% = 0
          end if
      end select
   end if
```

SETLANG

指令

■ **功能**　　用于设定GC-A2显示的语言。

■ 格式　　SETLANG（设定的语言编号）

■ 使用范例　　SETLANG（3）

■ 说明
- 设定GC-A2显示的语言。
- 设定范围为：1~4（依次对应GA-A2可预置4种语言1~4）

■ 相关项目　　GETLANG

■ 程序实例

```
if type% = 3 then
   if id@ = ..SWT000 then
    if data% = 1 then
        setlang(3)
    elseif data% = 0 then
        getlang(lang%)
    end if
   end if
end if
```

SETLNEPLOT

指令

■ **功能**　　SETLNEPLOT 用来设置曲线图的显示范围。

■ 格式　　SETLNEPLOT 显示起始点, 显示终点

■ 使用范例　　SETLNEPLOT 10, 50

■ 说明

- SETLNEPLOT 指令用来设置曲线图的显示范围。在设定完显示范围之后，执行LNEDSP, LNESHIFT, 或PRDSP指令，则在指定范围内显示数值。
- 在执行LNEDSP, LNESHIFT, 或 PRDSP指令之后，设定的显示范围释放，整个范围的显示状态就设定了。
- 指定为闪烁（“blink”）或点灭（“on-and-off”）的曲线图在设定的整个范围内显示。
- 当在100ms之内为两个或两个以上曲线图设定了范围时，可能只有后面一个会有效。

■ 相关项目　　LNEDSP, LNESHIFT, PRDSP

■ 程序实例

```
evnt
   input type,id@,data
   SETLNEPLOT 20, 30
   lneshift (..lnegraph , 1,1, 40)
end evnt
```

SETSIO

指令

■ **功能**　　SETSIO 指令用来设置无协议通信端口的接收方式。

■ 格式　　SETSIO 端口号, 值

■ 使用范例　　SETSIO 2 , &HD

■ 说明

- SETSIO：当接收无协议通信数据时，用来设置端口在向部品/画面程序发送消息时的状态。
- 端口号指设置成无协议通信模式的端口编号。
- 当端口模式为二进制模式时，要指定从所连接设备接收的字节数，（当然也可以是0）。当为文本模式时，要指定一个结束码（1～0FF）。
- 在二进制码传输模式时，指定从所连接设备接收的字节数，在接收到指定字节的数据之后，向部品或画面发送消息。
- 在以文本形式传输时，当检测到字符串结束码（0h）时，向部品或画面发送消息。结束码只能为一个字节。
- 端口号必须是预先使用OPENSIO指令将其打开的端口编号。

■ 相关项目　　OPENSIO, CLOSESIO, WRITESIO, WRITWSIOB, FLUSH, IOCTL

■ 程序实例

```
conf
   global buf$ * 200
   opensio 2 , 1 , buf$
   SETSIO  2 , &HD
end conf
evnt
   strdsp ..STR000 , buf$
   closesio 2
end evnt
```

SETTIM

指令

■ **功能**　　SETTIM 指令设定指定定时器的计时上限。

■ 格式　　SETTIM 定时器号, 定时时间, 定时器类型

■ 使用范例　　SETTIM ID@, 100, 0
SETTIM VAR, 200, 1

■ 说明

- SETTIM 指令用来设定指定定时器的计时时间。在设定这个值时定时器必须处于停止（stop）状态。
- 定时器号指要进行定时设置的定时器ID号或ID变量，该值必须为整形数或整型变量。
- 在使用STARTTIM指令启动定时器后，定时器以100ms为单位开始计数。
- 定时器类型指定时器的定时类型，有两种：一种为标准型，一种为间隔型。标准型在到了设定的时间后停止计时；而间隔型在到达设定的时间后又从头开始计时。
  0: 标准型
  1: 间隔型
- 当定时间隔等于或小于1秒时，因为消息堆积太多可能导致系统报错。

■ 相关项目　　OPENTIM, STARTTIM, STOPTIM, CLOSETIM, CONTTIM, READTIM

■ 程序实例

```
conf
   static timid@
   timid@ = opentim()
   SETTIM timid@, 20, 0
   starttim timid@
end conf
evnt
   input type% , id@ , data%
   if type% = 3 then
      tim% = readtim(timid@)
      numdsp ..NUM000,tim%*100
   end if
end evnt
```

SETTIME

指令

■ **功能**　　SETTIME 指令用来设置系统内部时钟。

■ 格式　　SETTIME 时钟,分钟, 秒钟

■ 使用范例　　SETTIME 12, 0, 0

■ 说明

- 时钟为0 ～ 23之间的一个数值。
- 分钟为0 ～ 59.
- 秒钟为0 ～ 59.
- 如果设置值超出上述范围，系统会报错。
- 一旦使用SETTIME设置了系统时钟后，系统时钟能停电保持并实时更新（除GC-53LC/LM外新型号都可以）。

■ 相关项目　　DATE\$, GETDATE, GETDATE, SETDATE, TIME\$

■ 程序实例

```
evnt
   input type% , id@ , h%, m%, s%
   settime h%, m%, s%
end evnt
```

SHIFT

指令

■ **功能** SHIFT 将指定变量的内容向左或向右移位。

■ 格式 SHIFT 变量名, 移动量

■ 使用范例 SHIFT VARIABLE% , 1

■ 说明

- SHIFT 将指定变量的内容（转换成二进制码数）向左或向右移动指定的位数。
- 移动后空出来的位以0代替。
- 变量名指要进行数据移位的变量的名称。它必须是一个整型变量。
- 移动量指数据要移动的位数。范围在-31~31之间，大于0表示向左移，小于0表示向右移位。

■ 相关项目 无

■ 程序实例

```
conf
end conf
evnt
   input type% , id@ , data%
   numdsp ..NUM000 , data%
   shift data% , 1
   numdsp ..NUM000 , data%
end evnt
```

SIN

函数

■ **功能** SIN 计算数学表达式的正弦值。

■ 格式 SIN (数学表达式)

■ 使用范例 X = SIN (ANGLE)

■ 说明

- SIN 函数用于计算指定数学表达式的正弦值，数学表达式值的单位为弧度。

■ 相关项目 ATN, COS, TAN

■ 程序实例

```
evnt
   angle = 3.141592/3
   x = SIN ( angle )
   numdsp ..num000,x
end evnt
```

SLDDSP

指令

■ **功能** SLDDSP 以滑动显示图形式显示数据。

■ 格式 SLDDSP 控件名, 显示值

■ 使用范例 SLDDSP .BUHIN.GRAPH, 30.0

■ 说明
- 控件名（控件名称）指控件的名称或能代表它的ID变量。
- 显示数据（display-data）指要在滑动显示图中显示出来的数据。
- 如果内部控件参数有效，则这里设定的显示值无效。

■ 相关项目 None

■ 程序实例

```
evnt
   input type,id@,data
   SLDDSP ..SLD000, data
end evnt
```

SOF

函数

■ **功能**　　SOF 函数用于计算区域的大小。

■ 格式　　SOF（文件编号）

■ 使用范例　　AAA = SOF（文件编号）

■ 说明

- 文件编号指在FIELD声明中定义的文件号。计算出来的文件大小将成为后面读写的数据量。
- 文件的大小以字节计算。

■ 相关项目　　FOPEN, FIELD, FCLOSE, FPUT, FGET, EOF

■ 程序实例

```
conf
   field  5
      global no%
      global moji1$ , moji2$
   end field
   global buff$ * 50
   opensio 1 , 0 , buff$
   fopen  “C:TEST’ , 2 , 5
end conf
evnt
   no% = 1
   moji1$ = “product-name”
   moji2$ = “product-number”
   size% = SOF(5)
   mcpy 5 , buff$
   writesiob 1 , size% , buff$
end evnt
```

SQR

函数

■ **功能** SQR 计算平方根。

■ 格式 SQR (数学表达式)

■ 使用范例 X = SQR (Y)

■ 说明 • SQR 用来计算数学表达式的平方根。

■ 相关项目 无

■ 程序实例

```
evnt
   x = SQR ( a^2 + b^2)
   numdsp ..NUM000, X
end evnt
```

STARTTIM

指令

■ **功能**　　STARTTIM 用于启动定时器

■ 格式　　STARTTIM 定时器号

■ 使用范例　　STARTTIM ID@
STARTTIM VAR

■ 说明
- STARTTIM 启动指定的定时器（从零开始计时，单位为0.1s）。
- 定时器号可以是表示定时器的ID变量、或是在0～15之间的某个整型数。

■ 相关项目　　OPENTIM, STOPTIM, CONTTIM, CLOSETIM, SETTIM, READTIM

■ 程序实例

```
conf
   static timid@
   timid@ = opentim()
   settim timid@, 20, 0
   STARTTIM timid@
end conf
evnt
   input type% , id@ , data%
   if type% = 3 then
      tim% = readtim(timid@)
      numdsp ..NUM000,tim%*100
   end if
end evnt
```

STATIC

指令

■ **功能**　　STATIC 用来定义静态变量

■ 格式　　STATIC 变量名1 [, 变量名2 ...]

■ 使用范例　　STATIC VAR, XYZ(2,3), MOJI$ * 20

■ 说明

- STATIC 用来声明静态变量。静态变量只能在其所生明的程序中引用。该类型的变量在每次上电时初始化一次，其值在系统有电期间能自保持。
- 普通变量、数组变量、字符串变量都可以定义成静态变量。
- 在定义数组或字符串变量时，不需要使用DIM和STRING指令。

■ 相关项目　　AUTO, BACKUP, DIM, GLOBAL, LOCAL, STRING

■ 程序实例

```
conf
   STATIC var%, float
   STATIC moji$ * 50, moji2(10) * 3
   STATIC xyz@(10,10)
end conf
```

STOP

指令

■ **功能** STOP 指令中止程序的执行。

■ 格式 STOP

■ 使用范例 STOP

■ 说明 • 执行本指令将停止本指令后面程序的执行。

■ 相关项目 RUN

■ 程序实例

```
evnt
   input type , id@, data
   if type = 3 and data = 0 then STOP
   numdsp ..NUM000, data
end evnt
```

STOPTIM

指令

■ **功能**　　STOPTIM 指令使定时器计时值停止增加。

■ 格式　　STOPTIM 定时器号

■ 使用范例　　STOPTIM ID@
STOPTIM VAR

■ 说明
- STOPTIM使定时器计时值停止增加。
- 定时器号指定时器的ID变量或定时器数字代号0～15。

■ 相关项目　　OPENTIM, STARTTIM, CONTTIM, CLOSETIM, SETTIM, READTIM

■ 程序实例

```
conf
   static timid@
   timid@ = opentim()
   settim timid@, 20, 0
   starttim timid@
end conf
evnt
   input type% , id@ , data%
   if type% = 3 and data% = 1 then
      tim% = readtim(timid@)
      numdsp ..NUM000,tim%*100
   else
     STOPTIM timid@
   end if
end evnt
```

STR$

函数

■ **功能**　　本函数将数字常量转换成字符串常量。

■ 格式　　STR$ (数学表达式)

■ 使用范例　　A$ = STR$(123)

■ 说明

- 数学表达式可以是整型和浮点型表达式。
- 当数字常量为负值时，转换的结果是在字符串的前面添加一个“－”号。

■ 相关项目　　VAL

■ 程序实例

```
evnt
   input type, id@,data
   a$ = STR$ ( data )
   strdsp ..hyojiki , a$
end evnt
```

STRCOLOR

指令

■ **功能**　　STRCOLOR 用于改变字符串控件的显示背景和颜色

■ 格式　　STRCOLOR 控件名称, 字符颜色, 填充, 填充颜色, 背景颜色

■ 使用范例　　STRCOLOR ..GRAPH, 1, 2, 5, 2

■ 说明

・这个语句进行文字显示器的显示色的背景的瓷砖/颜色的变更
・在指定颜色或瓷砖时，指定-1 表示该项目保持现状不变。
・控件名是字符指示器的名字或表示字符指示器的 ID 型变量。
・文字显示颜色是表示显示文字的颜色。是 0 ~ 15 的色号。
瓷砖表示文字显示器的背景的排列图案。模式是 0 ~ 15 的范围
・表示颜色是表示瓷砖表示部分的色号的数值。色号是 0 ~ 15
范围。
・背景色是表示瓷砖背景部分的色号的数值。色号是 0 ~ 15 的
围。

■ 相关项目　　STRDSP, STRFORM

■ 程序实例

```
conf
   static name@
   name@ = ..STR000
end conf
evnt
   input type%, id@, data%
   if type% = 3 then
      STRCOLOR name@, 2, -1,-1,-1
   endif
end evnt
```

STRDSP

指令

■ **功能**　　在指定的字符串显示控件里显示指定字符。

■ 格式　　STRDSP 控件名称，显示数据

■ 使用范例　　STRDSP .BUHIN.GRAPH, “ABCDEF”

■ 说明

- STRDSP 在字符串显示器中显示字符数据。
- 控件名称表示分配给字符串显示控件的名称或能表示该显示控件的ID型变量。
- 显示数据指要显示的字符串。
- 如果字符串控件本身的内部参数有效，则这里设置将不起作用。

■ 相关项目　　STRCOLOR, STRFORM

■ 程序实例

```
conf
   static name@
   name@ = ..STR000
end conf
evnt
   input type%, id@, data$
   STRDSP name@, data$
end evnt
```

STRFORM

指令

■ **功能**　　STRFORM 指令用于改变字符串显示器的显示方式。

■ 格式　　STRFORM 控件名称, 显示方式

■ 使用范例　　STRFORM ..HYOJIKI, 0

■ 说明

- STRFORM指令用于改变字符串显示器的显示方式。
- 控件名称表示分配给字符串显示器的名称或能表示该字符串显示控件的ID型变量。
- 显示方式指代表如下三种显示方式的数字代号：
  0: 左边对其显示
  1: 居中显示
  2: 右边对其显示

■ 相关项目　　STRCOLOR, STRDSP

■ 程序实例

```
evnt
   input type , id@,data
   var@ = .buhin.moji
   STRFORM var@ , data
   strdsp  var@ , "ABCDEFG"
end evnt
```

STRING

指令

■ **功能**　　STRING 用于指定字符串变量的长度。

■ 格式　　STRING 变量1 * 长度1 [, 变量2 * 长度2 ......]

■ 使用范例　　STRING MOJI$ * 50

■ 说明

- STRING指令用来指定局部字符串变量的长度（字符个数）。
- 本指令主要是为了保持同GCSGP软件的兼容性，在SCA2中使用LOCAL指令即可实现。
- 字符串变量的默认最大长度为20个字符。当超过20个时，要使用该指令进行声明。同时，在使用字符串变量之前也需要进行声明。
- 变量名后面必须接$符号，以表示字符串变量。
- 字符串变量的长度以整形数表示。
- 定义的多个字符串之间使用“，”号隔开。

■ 相关项目　　GLOBAL, STATIC, BACKUP, LOCAL

■ 程序实例

```
conf
   string xxx$ * 40
   string moji$ * 50
end conf
```

SWFIG

指令

■ **功能**　　SWFIG 指令用来设置开关状态改变（ON/OFF）时显示的图形。

■ 格式　　SWFIG 开关名称, 显示图形, 状态, 子开关的ID

■ 使用范例　　SWFIG ..SW1, FIG3, 0, 0

■ 说明

- 指令用来设置开关状态改变（ON/OFF）时显示的图形。选择开挂和普通开关都可以使用。
- 开关名称表示分配给开关的名称或能表示该开关的ID型变量。
- 显示图形表示在不同状态下显示图形的名称或ID名称。
- 状态是指该图形是在OFF状态下显示还是在ON状态下显示。
  0: 表示该图形是在开关状态为OFF时显示。
  1: 表示该图形是在开关状态为ON时显示。
- 当使用选择开关时，要注明需要显示该图形的子开关ID号。当使用普通开关时，该值为0。

■ 相关项目　　None

■ 程序实例

```
conf
   static figid@,subid,onoff
   figid@ = FIG03
   subid = 3
   onoff = 1
end conf
evnt
   input type,id@,data
   if type = 3 and id@ = ..SWT000 then
      SWFIG id@ , figid@ , onoff , subid
   endif
end evnt
```

SWMODE

指令

■ **功能** SWMODE 指令改变开关的模式

■ 格式 SWMODE 开关名称, 模式

■ 使用范例 SWMODE ..SW1, 2

■ 说明
- 开关名称表示分配给开关的名称或能表示该开关的ID型变量。
- 表示开关模式的数据分别如下：
  0: 普通模式
  1: 输入禁止模式
  2: 半透明模式

■ 相关项目 无

■ 程序实例

```
evnt
   input type,id@,data
   if type  = 3 then
      SWMODE ..sw2 , 1
      SWMODE var@ ,2
   end if
end evnt
```

SWREAD

函数

■ **功能**　　SWREAD 函数用于读取指定开关的状态。

■ 格式　　SWREAD (开关控件名)

■ 使用范例　　STATE = SWREAD (..SW1)

■ 说明

- SWREAD 函数用于读取开关控件的ON/OFF状态。
- 开关名称表示分配给开关的名称或能表示该开关的ID型变量。
- 当开关控件的参数有效时，全局画面及其部品不能使用CONF程序块。
- SWREAD 不能读取未显示画面上开关的状态。
- 普通开关的状态由下列数据表示：
  0: OFF 状态
  1: ON 状态
- 执行此指令后，选择开关的状态表示方式如下：
  0:　　所有开关状态均为OFF
  其它值:　　状态为ON的开关的编号。(子开关从左往右递增编号，即左上角编号为1)

■ 相关项目　　SWWRITE

■ 程序实例

```
evnt
   input type,id@,data
   id@ = ..SW2
   state = SWREAD (ID@)
   if state = 0 then
      swwrite id@,1
   endif
end evnt
```

SWREV

指令

■ **功能**　　SWREV 指令的功能是：当开关状态改变时，其显示结果是否反转。

■ 格式　　SWREV 开关名称, 操作

■ 使用范例　　SWREV ..SW2, 0

■ 说明

- 执行SWREV指令后，确定开关状态改变之后，其面板上的开关显示结果是否反转。
- 开关名称是指系统自动分陪配（或人为指定）的开关控件名称或能代表它的ID型变量。
- 操作表示是否进行反转：
  0：表示显示结果不反转
  1：显示结果反转

■ 相关项目　　无

■ 程序实例

```
evnt
   input type,id@,data
   if type = 3 and id@ = ..SWT000 then
      id@ = ..SW2
      SWREV id@,1
   endif
end evnt
```

SWWRITE

指令

■ **功能**　　SWWRITE 指令改变指定开关的状态。

■ 格式　　SWWRITE 开关名称, 状态

■ 使用范例　　SWWRITE ..SW1, 1

■ 说明

- 即使在不按下面板上的开关，使用SWWRITE 指令可以改变开关的状态（ON / OFF）。当状态改变时，标示状态的数据被传送到开关控件所在部品程序。
- 开关名指系统分配的开关名称或指定的ID变量名。
  对于多路开关，要首先要设置开关的偏移量编号，并使用GETID指令读取开关的ID。
- 表示普通开关状态的数据如下：
  0: OFF 状态
  1: ON 状态
- 选择开关的状态表示方式如下：
  0:　　所有开关状态均为OFF
  其它值:　　状态为ON的开关的编号。（子开关从左往右递增编号，即左上角编号为1）
- 多路开关和选择开关的编号从左往右依次为1、2、3……。 其下方的开关也采用同样的方式计数。
- 当执行本指令时，同开关被按下一样，也将发送出一个消息。
- 当在按下开关的同时执行本指令时，因为冲突，系统将会报错。.
- SWWRITE 指令对于点动开关无效，即仅对翻转开关有效。

■ 相关项目　　GETID,SWREAD

■ 程序实例

```
evnt
   input type,id@,data
   id@ = ..SW2
   state = swread (ID@)
   if state = 0 then
      SWWRITE id@,1
   endif
end evnt
```

S2U

指令

■ **功能** 将SJIS编码的字符串转换成对应的Unicode编码字符串。

■ 格式 S2U（字符串）

■ 使用范例 S2U（moji$）

■ 说明
- 不可使用除了字符串或者字符串变量以外的对象
- 返回值为Unicode字符串

■ 相关项目 SWREAD

■ 程序实例

```
evnt
  input type%,id@
  if type% = 16 then
      input data%                'SJIS Code → Unicode 编码转换
      pos% = getoffset([port]:[局号]~[字符串存放起始地址],id@)
      if data% > 0 then
          mid$(moji$,pos%+1,1) = chr$(data%)
      else
          mid$(moji$,pos%+1,1) = " "
      end if
      moji1$ = S2U(moji$)
      strdsp ..STR000, moji1$
  end if
end evnt
```

TAN

函数

■ **功能** TAN 用于计算数学表达式的正切值。

■ 格式 TAN (数学表达式)

■ 使用范例 X = TAN (角度表达式)

■ 说明

- TAN 函数计算其后面括号里数学表达式的正切值。数学表达式值的单位为弧度。

■ 相关项目 ATN, SIN, COS

■ 程序实例

```
evnt
   angle = 3.141592/3
   x = TAN ( angle )
   numdsp ..num000,x
end evnt
```

TIME$

指令

■ **功能** TIME$ 读取当前时间（时刻）

■ 格式 TIME$

■ 使用范例 A$ = TIME$

■ 说明
- TIME$ 以H:M:S格式字符串形式读取当前时刻。
- 本指令不能用于设置当前时间。要设置当前系统时间可以使用SETTIME指令。

■ 相关项目 DATE\$, GETDATE, GETTIME, SETDATE, SETTIME

■ 程序实例

```
conf
   moji$ = TIME$
   strdsp ..STR000 , moji$
end conf
```

TIMID

函数

■ **功能** TIMID 将以整型数表示的定时器编号转换成定时器ID值。

■ 格式 TIMID (定时器号)

■ 使用范例 AA@ = TIMID (VAR)

■ 说明

- TIMID 将以整型数表示的定时器编号转换成定时器ID值。定时器号可以是整形数，也可以是一个整型变量。

■ 相关项目 TIMINT, OPENTIM2

■ 程序实例

```
conf
   opentim2(2)
   settim 2 , 20, 0
   starttim 2
end conf
evnt
   input type,id@
   if id@ = timid(2) then
      .......
   end if
end evnt
```

TIMINT

函数

■ **功能** TIMINT 函数将ID型数表示的定时器转换成整形数表示的定时器。

■ 格式 TIMINT (ID号)

■ 使用范例 VAR = TIMINT (ID@)

■ 说明 • ID号为需要作转换的定时器的ID号或ID变量。

■ 相关项目 TIMID, OPENTIM

■ 程序实例

```
evnt
   .....
   id@=opentim()
   no = TIMINT (id@)
   chktim ( no )
   ....
end evnt
```

U2S

指令

■ **功能**　　将Unicode编码字符串转换成对应的SJIS编码的字符串。

■ 格式　　U2S（字符串）

■ 使用范例　　U2S（moji$）

■ 说明
- 不可使用除了字符串或者字符串变量以外的对象
- 返回值为SJIS字符串

■ 相关项目　　GETKEY,GETKEYT,SETIMEMODE ， S2U

■ 程序实例

```
if type% = 2 then           'KEY BOARD 发送指令
    input data$, code%
    if code% = &hF0 then    '&F0 处理指令
                            'Unicode → SJIS Code 数据转换 & PLC 数据写入
        s_data$ = U2S(data$)
    if len(s_data$) <= [PLC 读取字符串] then
                            'PLC 写入 SJIS Code 转换数据
    else
                            '键盘再次输入处理
    end if
end if
```

VAL/VAL2

函数

■ **功能**　　VAL/VAL2 将以数字字符串表示的值转换成以数字表示的值。

■ 格式　　VAL（字符串）
VAL2（字符串）

■ 使用范例　　A = VAL（“123”）
A = VAL2（“123.45”）

■ 说明

- 当字符串的开始字符不是+,－，0～9，E和. 时，函数返回值为0。
- 当字符串内出现不可转换的字符时，函数只将该字符前面的字符转换。
- VAL函数的结果是将以数字字符串表示的值转换成以数字表示的值，结果为实数。
- VAL函数的结果是将以数字字符串表示的值转换成以数字表示的值，结果为整数。

■ 相关项目　　STR$

■ 程序实例

```
conf
   var = VAL ( “234”)
   numdsp ..NUM000 , var
end conf
```

WHILE ... WEND

指令

■ **功能**　　当条件表达式为真（成立）时，执行WHILE…WEND之间的内容。

■ 格式

```
WHILE 条件表达式
......
WEND
```

■ 使用范例

```
WHILE   X > 0
......
WEND
```

■ 说明

- 当条件表达式为真时，执行WHILE…WEND之间的内容；
- 当表达式条件不成立时才执行WEND后面的内容。

■ 相关项目　　IF ... THEN ... ELSE

■ 程序实例

```
conf
   static var(10)
   WHILE i% < 10
      var(i%) = i% * 5
   WEND
end conf
```

WRITESIO/WRITESIOB

指令

■ **功能**　　WRITESIO 和 WRITESIOB 将要发送的数据传送到无协议通信数据发送缓冲区。

■ 格式　　WRITESIO 端口号, 变量名
WRITESIOB 端口号, 传送字节数, 变量名

■ 使用范例　　WRITESIO 2 , moji$
WRITESIOB 2 , 20 , moji$

■ 说明

- WRITESIO 指令以文本的形式将数据写到无协议通信缓冲区（串行口）。WRITESIOB 则以二进制码形式将数据写到相同的地方。
- 端口号指通信串口数字编号，CH1～CH3 编号为1～3。
- 传送字节数指要传输的数据量（仅在数据以二进制码方式传送时有效）。
- 变量名指数据要传输到的目标变量名。
- 在以文本形式传输时，数据连续传送，直到检测到字符串结束码（0h）。（也就是数据可以是0～0FFh，结束码不会自动添加到数据流中）
- 以二进制码传输时，数据可以是0～0FFh中的任何数。
- 端口号必须是预先使用OPENSIO指令将其打开的端口编号。

■ 相关项目　　OPENSIO, CLOSESIO, WRITESIO, WRITWSIOB, SETSIO

■ 程序实例

```
conf
   global buf$ * 200
   opensio 2 , 1 , buf$
   setsio  2 , &HD
end conf
evnt
   sendbuf$ = “ABCDEFG”
   WRITESIO 2 , sendbuf$
   closesio 2
end evnt
```

FILEWRITE

指令

■ **功能**　　文件数据写入。

■ 格式　　写入行数 = FILEWRITE（类型、文件名、写入数据、写入行数）

写入行数	整数型	1 以上：写入行数 -1 以下：写入报错
类型	整数型	0：新建、覆盖保存 1：添加到文件末行
文件名	文字型	
写入数据	整数型/实数型/字符型	
写入行数	整数型	

■ 使用范例　　ret% = FileWrite(Type%, Path$, SaveData$, LineSize%)

■ 说明

•向文件名指定的文件写入指定行数的数据。
•如果指定的文件不存在则新建文件。文件已存在则覆盖保存
•写入的结果作为返回值。

■ 相关项目　　FILEWRITE,FILEREAD,FINDFILE,STRSPLIT

■ 程序实例

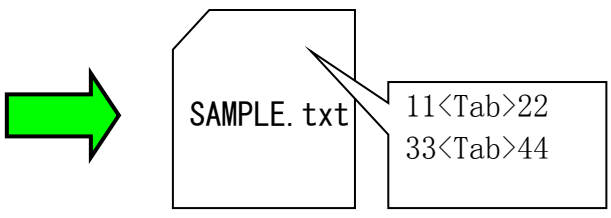
将两行数据覆盖保存在 C 驱动器下方的“SAMPLE.txt”文件中。

```
   init
static SaveData$(10)
end init
evnt
    input type%,id@ data%
    ·····(中略)
    SaveData$(0) = “11” + chr$(&H09) + “22”
    SaveData$(1) = “33” + chr$(&H09) + “44”
    ret% = FileWrite (0,"G:\SAMPLE.txt",SaveData$(0), 0)
end evnt
```

（K-BASIC 执行结果）

字符串型数组	
SeadData$(0)	“11”<Tab>” 22”
SeadData$(1)	“33”<Tab>” 44”
・・・	・・・
SeadData$(10)	“100”<Tab>” 110”

FILEWRITE



FILEREAD

指令

■ **功能**　　从指定文件读取数据保存到数组变量。

■ 格式　　读取行数 = FileRead(文件名、读取数据保存路径)

读取行数	整数型	1 以上：行数 -1 以下：读取失败
文件名	字符型	
读取数据保存路径	整数型/字符型	ReadData$:保存为字符型 ReadData%:保存为整数型

■ 使用范例　　ret = FileRead(Path$, ReadData$(0))

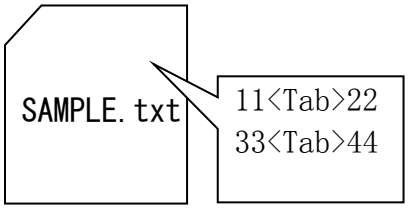
■ 说明　　・从指定文件读取数据保存到数组变量。
・读取写过作为返回值。

■ 相关项目　　FILEWRITE,FILEREAD,FINDFILE,STRSPLIT

■ 程序实例

从 C 盘“SAMPLE.txt”文件读取数据存放到数组变量。

```
init
   static ReadData$(10)
end init
evnt
   input type%,id@ data%
   ・・・・・・・・・・・・・(中略)
   ret% = FileRead ("C:\ SAMPLE.txt",RaveData$(0))
end if
```



FILEREAD

字符串型数组	
SeadData$(0)	“11”<Tab>” 22”
SeadData$(1)	“33”<Tab>” 44”
・・・	・・・
SeadData$(10)	“100”<Tab>” 110”

FINDFILE

指令

■ 功能　　根据指定的名称搜索文件或者文件夹。

■ 格式　　返回值 = FindFile（类型，文件夹/文件名）

返回值	整数型	0：文件夹/文件不存在 1：文件夹/文件存在
类型	整数型	0：搜索文件夹 1：搜索文件
文件名	字符型	文件夹/文件名

■ 使用范例　　ret% = FindFile(Type%, Path$)

■ 说明

- 搜索指定名称的文件或者文件夹
- 搜索结果作为返回值

■ 相关项目　　FILEWRITE,FILEREAD,FINDFILE,STRSPLIT

■ 程序实例

在 C:盘内搜索指定的文件或者文件夹。

```
evnt
    input type%,id@ data%
    ·····(中略)
    ret1% = FindFile(0, "G:\TEMP")
    ret2% = FindFile(1, "G:\TEMP\SAMPLE.txt")
end evnt
```

STRSPLIT

指令

■ **功能**　　将字符数列根据分隔符分割保存。

■ 格式　　返回值 = StrSplit(原始数组、保存的数组、分隔符)

原始文本	字符型/整数型	0：文件夹/文件不存在 1：文件夹/文件存在
保存的文本	字符型/整数型	0：搜索文件夹 1：搜索文件
分隔符	整数型	文件夹/文件名

■ 使用范例　　ret% = StrSplit(ReadData$, data$, code%)

■ 说明　　• 根据指定的分隔符分割字符数组并保存到指定的数组。

■ 相关项目　　FILEWRITE,FILEREAD,FINDFILE,STRSPLIT

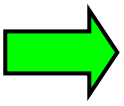
■ 程序实例

根据特定分隔符(字符代码09H)分割数组（ReadData$）后保存到指定的数组（data$）。

```
init
   Static ReadData$(2), data$(2)(2)
   ReadData$(0) = “11” + chr$(&H09) + “22”
   ReadData$(1) = “33” + chr$(&H09) + “44”
end init
evnt
   input type%,id@ data%
   ・・・・・・・・・・・・・・・(中略)
   ret = StrSplit ( ReadData$, data$(0,0), 9 )
end if
```

STRSPLIT

分割源数组	
ReadData$(0)	“11”<Tab> “22”
ReadData$(1)	“33”<Tab> “44”



分割后保存的数组		
Data$	(0)	(1)
(0)	“11”	“22”
(1)	“33”	“44”

JTEKT

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-M9549B

2024 年 7 月