

# ISaGRAF

版本 3.5

用户指南

ICS Triplex ISaGRAF Inc.

本书内容非常容易在没有通告的情况下被进行修改，**ICS Triplex ISaGRAF Inc.**公司对此修改不承担任何义务。本书中描述的软件——包扩数据库中存放的所有信息——只有在签定了许可证或非泄密协议后，**ICS Triplex ISaGRAF Inc.**公司方可提供，并且需根据许可证或非泄密协议的条款进行使用和拷贝。除非得到许可证或非泄密协议等专门允许，任何对这个软件的拷贝都将视为违法行为。没有**ICS Triplex ISaGRAF Inc.**公司的书面允许，不管为了任何目的，都不得以任何形式或任何方法(电子的或机械的，包括照相复制和录音复制等)，对本书的任何部分进行复制。

(c) 1994 - 2003 年**ICS Triplex ISaGRAF Inc.**国际公司。本公司保留一切权利。

**ICS Triplex ISaGRAF Inc.**公司印制于法国。

**ISaGRAF** 是**ICS Triplex ISaGRAF Inc.**公司的注册商标。

**MS-DOS** 是微软公司的注册商标。

**Windows** 是微软公司的注册商标。

**Windows NT** 是微软公司的注册商标。

**OS-9** 和 **ULTRA-C** 是 **Microware** 公司的注册商标。

**VxWorks** 和 **Tornado** 是风河系统股份有限公司的注册商。

所有其它商标和产品名是它们各自所有者的商标或注册商标

# 目录

<b>A. 用户指南</b>	<b>A-15</b>
<b>A.1 启动</b>	<b>A-16</b>
A.1.1 安装 ISaGRAF	A-16
A.1.2 口令管理	A-20
A.1.3 在线信息的使用	A-24
A.1.4 样板应用程序	A-25
<b>A.2 项目管理</b>	<b>A-31</b>
A.2.1 建立和操作项目	A-32
A.2.2 操作几个项目组	A-34
A.2.3 选项	A-35
A.2.4 工具	A-36
<b>A.3 程序管理</b>	<b>A-37</b>
A.3.1 一个项目的部件	A-37
A.3.2 操作程序	A-40
A.3.3 运行代码生成工具	A-46
A.3.4 其它 ISaGRAF 工具	A-48
A.3.5 向工具菜单中添加命令	A-50
A.3.6 仿真和调试应用程序	A-50
<b>A.4 使用 SFC 编辑器</b>	<b>A-54</b>
A.4.1 SFC 语言主题	A-54
A.4.2 输入 SFC 图	A-58

A.4.3	处理已存在的 SFC 图	A-60
A.4.4	输入层次 2 程序	A-62
A.4.5	使用 SFC 图库	A-67
<b>A.5</b>	<b>使用流程图编辑器</b>	<b>A-69</b>
A.5.1	FC 语言的基本语句	A-69
A.5.2	输入流程图	A-71
A.5.3	在已有的流程图上操作	A-76
A.5.4	输入第二层程序	A-77
A.5.5	用快捷 LD 编程	A-79
A.5.6	显示选项	A-80
<b>A.6</b>	<b>使用快捷 LD 编辑器</b>	<b>A-82</b>
A.6.1	LD 语言基础	A-82
A.6.2	输入 LD 图	A-86
A.6.3	处理已存在的图表	A-91
A.6.4	显示选项	A-92
A.6.5	在线帮助	A-95
<b>A.7</b>	<b>使用 FBD/LD 编辑器</b>	<b>A-96</b>
A.7.1	FBD/LD 语言基础	A-96
A.7.2	输入 FBD 图表	A-100
A.7.3	处理已存在的图表	A-103
A.7.4	显示选项	A-105
A.7.5	样式和修订跟踪	A-108
A.7.6	在线帮助	A-110
A.7.7	打印 FBD 图	A-110

<b>A.8</b>	<b>使用文本编辑器</b>	<b>A-111</b>
A.8.1	(编辑命令)	A-111
A.8.2	句法上色彩	A-112
A.8.3	选项	A-113
<b>A.9</b>	<b>程序编辑器</b>	<b>A-114</b>
A.9.1	调用其它 ISaGRAF 工具	A-114
A.9.2	程序参数	A-115
A.9.3	"文件"菜单中的其它命令	A-116
A.9.4	更新程序日志	A-116
A.9.5	从字典中选择变量	A-117
A.9.6	输出窗口	A-118
<b>A.10</b>	<b>字典编辑器的使用</b>	<b>A-119</b>
A.10.1	字典主窗口	A-122
A.10.2	管理变量	A-124
A.10.3	对象的描述	A-127
A.10.4	快捷声明	A-129
A.10.5	Modbus SCADA 寻址图	A-131
A.10.6	与其他应用程序交换信息	A-132
<b>A.11</b>	<b>输入/输出连接编辑器的使用</b>	<b>A-138</b>
A.11.1	定义输入/输出插板	A-139
A.11.2	设置插板参数	A-142
A.11.3	连接变量	A-142
A.11.4	可直接显示的变量	A-143
A.11.5	编号	A-144
A.11.6	设置单独保护	A-145

<b>A.12</b>	<b>转换表的生成</b>	<b>A-147</b>
A.12.1	主命令	A-147
A.12.2	转换表中的入点	A-148
A.12.3	规则和限制	A-150
<b>A.13</b>	<b>使用代码生成器</b>	<b>A-151</b>
A.13.1	主命令	A-151
A.13.2	编译程序的选项	A-152
A.13.3	产生C源代码	A-157
A.13.4	浏览信息	A-158
A.13.5	定义资源	A-159
<b>A.14</b>	<b>交叉引用</b>	<b>A-168</b>
<b>A.15</b>	<b>图形诊断器的使用</b>	<b>A-171</b>
A.15.1	诊断器窗口	A-172
A.15.2	应用程序的控制	A-173
A.15.3	选项	A-177
A.15.4	“写”命令	A-178
A.15.5	显示锁定状态和设备值	A-181
A.15.6	在线修改	A-182
A.15.7	动态数据交换 (DDE)	A-188
<b>A.16</b>	<b>监视变量表</b>	<b>A-190</b>
<b>A.17</b>	<b>调试 ST 和 IL 程序</b>	<b>A-193</b>
<b>A.18</b>	<b>注视点</b>	<b>A-195</b>
A.18.1	构造图形布局	A-195

A.18.2	列表布局	A-199
A.18.3	定义项样式	A-199
A.18.4	“文件”菜单命令	A-201
A.18.5	ISaGRAF V3.2 用户注意事项	A-202
<b>A.19</b>	<b>上 载</b>	<b>A-203</b>
A.19.1	上载一个项目	A-203
A.19.2	通讯设置	A-204
A.19.3	准备上载项目	A-204
A.19.4	在目标 PLC 中存储压缩源代码	A-206
A.19.5	目标 PLC 的内存需求	A-206
A.19.6	关于已上载的项目	A-207
A.19.7	兼容性问题	A-207
<b>A.20</b>	<b>使用调试工具</b>	<b>A-209</b>
<b>A.21</b>	<b>使用 ISaGRAF 仿真器</b>	<b>A-210</b>
A.21.1	与调试器连接	A-210
A.21.2	I/O 仿真	A-210
A.21.3	库组件	A-212
A.21.4	选项	A-213
A.21.5	保存和恢复输入状态	A-213
A.21.6	循环周期配置器	A-214
A.21.7	仿真脚本	A-216
<b>A.22</b>	<b>使用库管理器</b>	<b>A-228</b>
A.22.1	管理库元件	A-228
A.22.2	I/O 配置	A-232

A.22.3	I/O 综合设备	A-234
A.22.4	I/O 板	A-234
A.22.5	IEC 语言函数和块	A-237
A.22.6	C”函数和功能块	A-239
A.22.7	变换函数	A-241
<b>A.23</b>	<b>档案管理实用程序的使用</b>	<b>A-242</b>
A.23.1	调用档案管理器	A-242
A.23.2	选项	A-243
A.23.3	备份与恢复	A-244
A.23.4	档案文件	A-245
<b>A.24</b>	<b>打印文件</b>	<b>A-247</b>
A.24.1	自定义文件表内容	A-248
A.24.2	选项	A-250
<b>A.25</b>	<b>口令保护</b>	<b>A-253</b>
<b>A.26</b>	<b>高级编程技术</b>	<b>A-258</b>
A.26.1	关于 ISaGRAF 系统工具更多的内容	A-258
A.26.2	I/O 板和虚拟 I/O 板锁定	A-259
A.26.3	PC-PLC 连接确认	A-262
A.26.4	ISaGRAF 系统目录	A-263
A.26.5	应用符号	A-266
A.26.6	ISaGRAF "LARGE" (WDL) workbench 的极限值	A-274
<b>B.</b>	<b>语言参考</b>	<b>B-279</b>
<b>B.1</b>	<b>项目结构</b>	<b>B-280</b>

---

B.1.1	程序	B-280
B.1.2	循环执行、顺序执行	B-281
B.1.3	SFC 子程序和 FC 子程序	B-282
B.1.4	函数和子程序	B-283
B.1.5	功能块	B-285
B.1.6	程序描述语言	B-286
B.1.7	执行规则	B-287
<b>B.2</b>	<b>公共对象</b>	<b>B-289</b>
B.2.1	基本类型	B-289
B.2.2	常数表达式	B-290
B.2.3	变量	B-293
B.2.4	注释	B-300
B.2.5	定义字	B-300
<b>B.3</b>	<b>SFC 语言</b>	<b>B-303</b>
B.3.1	SFC 图的主要格式	B-303
B.3.2	SFC 的基本元件	B-303
B.3.3	分支与会合	B-307
B.3.4	宏步	B-309
B.3.5	步内动作	B-311
B.3.6	与转换相连的条件	B-318
B.3.7	SFC 的动态规则	B-322
B.3.8	SFC 程序的层次结构	B-323
<b>B.4</b>	<b>流程图</b>	<b>B-325</b>
B.4.1	FC 基本部件	B-325
B.4.2	FC 图的开始	B-326

B.4.3	FC 图的结束	B-326
B.4.4	FC 流程的链接	B-327
B.4.5	FC 动作	B-327
B.4.6	FC 条件	B-327
B.4.7	FC 子程序	B-329
B.4.8	FC 输入 / 输出特定动作	B-330
B.4.9	FC 连接符	B-330
B.4.10	FC 注释	B-331
B.4.11	FC 复杂结构举例	B-331
B.4.12	FC 动态行为	B-332
B.4.13	FC 校验	B-333
<b>B.5</b>	<b>FBD 语言</b>	<b>B-334</b>
B.5.1	FBD 图的主要格式	B-334
B.5.2	RETURN 语句	B-336
B.5.3	跳转和标号	B-336
B.5.4	布尔求反运算	B-338
B.5.5	调用 FBD 中的功能或功能块	B-338
<b>B.6</b>	<b>LD 语言</b>	<b>B-340</b>
B.6.1	电源线连接线	B-340
B.6.2	多路连接	B-341
B.6.3	基本的 LD 触点和线圈	B-343
B.6.4	返回语句	B-353
B.6.5	跳转和标号	B-354
B.6.6	LD 中的块	B-355
B.6.7	LD 中的 « 内线 » 功能块	B-357

<b>B.7</b>	<b>ST 语言</b>	<b>B-361</b>
B.7.1	ST 主要语法	B-361
B.7.2	表达式和括号	B-363
B.7.3	功能或功能块的调用	B-364
B.7.4	ST 特殊的布尔操作符	B-367
B.7.5	ST 基本语句	B-371
B.7.6	ST 扩展	B-382
<b>B.8</b>	<b>IL 指令表语言</b>	<b>B-391</b>
B.8.1	IL 的主要语法	B-391
B.8.2	IL 操作符	B-393
<b>B.9</b>	<b>标准运算符，功能块与函数</b>	<b>B-404</b>
B.9.1	标准运算符	B-404
B.9.2	标准功能块	B-438
B.9.3	标准函数	B-467
<b>C.</b>	<b>目标用户指南</b>	<b>C-535</b>
<b>C.1</b>	<b>入门</b>	<b>C-536</b>
<b>C.2</b>	<b>安装</b>	<b>C-538</b>
<b>C.3</b>	<b>ISaGRAF DOS 目标的启动</b>	<b>C-540</b>
C.3.1	运行 ISaGRAF : ISA.EXE	C-540
C.3.2	特性	C-541
<b>C.4</b>	<b>ISaGRAF OS9 目标启动</b>	<b>C-546</b>
C.4.1	运行 ISaGRAF 单任务: isa	C-546

C.4.2	运行 ISaGRAF 多任务: isaker, isatst, isanet	C-547
C.4.3	特性	C-553
<b>C.5</b>	<b>启动 ISaGRAF VxWorks 目标</b>	<b>C-559</b>
C.5.1	系统资源管理: isassr.o	C-559
C.5.2	isa.o, isakerse.o 和 isakeret.o 的公共属性	C-559
C.5.3	运行 ISaGRAF 单任务: isa.o	C-560
C.5.4	运行 ISaGRAF 多任务: isakerse.o 和 isakeret.o	C-564
C.5.5	特殊属性	C-571
<b>C.6</b>	<b>ISaGRAF NT 目标的启动</b>	<b>C-576</b>
C.6.1	运行 ISaGRAF	C-576
C.6.2	选项的概论	C-576
C.6.3	特性	C-583
C.6.4	用户接口	C-590
<b>C.7</b>	<b>"C"语言编程</b>	<b>C-597</b>
C.7.1	概览	C-597
C.7.2	"C" 变换函数	C-600
C.7.3	"C" 函数	C-607
C.7.4	"C" 功能块	C-617
C.7.5	编译和链接技术	C-640
<b>C.8</b>	<b>Modbus 总线连接</b>	<b>C-651</b>
C.8.1	MODBUS 网络和通信协议	C-651
C.8.2	ISaGRAF 的实现过程	C-653
<b>C.9</b>	<b>" 断电" 管理</b>	<b>C-662</b>
C.9.1	基础	C-662

C.9.2	备份“应用变量”	C-663
C.9.3	程序状态备份	C-669
C.10	“错误”信息表和解释	C-671
D.	词汇表	D-688



## A. 用户指南

## A.1 启动

本章内容涉及 ISaGRAF 工作台安装、以及一个 ISaGRAF 应用程序的例子。这个例子可以帮助用户建立 ISaGRAF 软件主要特性的轮廓，以使用户能够较快使用 ISaGRAF 软件。

### A.1.1 安装 ISaGRAF

本节内容涉及 ISaGRAF 工作台的安装，以及如何为开发应用程序而设置计算机。

#### ☐ **硬件要求和软件要求**

ISaGRAF 工作台可以安装在能够运行 Windows3.1 系统的任何计算机上。然而，为了应用程序的开发，我们建议使用下列硬件：

使用 80486 或更高级的微处理器个人计算机

(建议使用奔腾微处理器)

- 8 兆字节常规内存和扩展内存

(建议使用 16 兆字节内存)

- 3.5 英寸(1.44 兆字节)的磁盘驱动器
- 至少具有 20 兆字节的可用空间的硬盘
- 图形 VGA 或 SVGA 适配器及与其兼容的监视器
- 鼠标器(图形开发工具需要)
- 并行端口 LPT1(需要保密钥匙)

在安装 ISaGRAF 工作台之前，系统中应有下列软件：

以 386 增强模式运行的 Windows 3.1 版本  
Windows 95

Windows NT 3.51 版本或 4.00 版本



### 安装程序的使用

可以用 ISaGRAF 软件中的安装程序 INSTALL 来安装 ISaGRAF 工作台。这个安装程序先将 ISaGRAF 软件从 ISaGRAF 光盘或磁盘拷贝到用户硬盘上。INSTALL 安装程序再将"ISaGRAF"程序组加到程序管理器窗口中，并在已安装的 **EXE** 子目录中建立文件名为"ISA.ini"的初始化文件。

INSTALL 安装程序是一个 Windows 程序，必须从 Windows 程序管理器运行这个程序或运行 Windows95/98/2000 的开始菜单命令。为了安装 ISaGRAF，必须完成以下步骤：

在适当的驱动器中插入 ISaGRAF 光盘或 1 号软磁盘

在程序管理器或开始菜单中，运行光盘根文件夹上的"SETUP.EXE"文件或软磁盘上的"A:\INSTALL.EXE"文件。

按照在线命令的引导，完成安装。为避免刚安装的 ISaGRAF 版本的文件与其它 ISaGRAF 版本的文件发生混乱，我们建议将 ISaGRAF 工作台安装在一个新目录上。

INSTALL 安装程序将询问是否需要下列组件：

- ISaGRAF 的可执行程序
- 在线信息和帮助文件
- ISaGRAF 标准库
- ISaGRAF 样板应用程序

我们郑重地建议：在第一次安装 ISaGRAF 时，应安装所有组件。以后通过重新安装 ISaGRAF 工作台，增加更多的组件。

ISaGRAF 主目录缺省名为 "\ISAWIN"，它可以使 ISaGRAF 的 Windows 版本容易与 ISaGRAF 的 MS-DOS 版本安装在同一个硬盘上。请参考手册"高级技术章中的 ISaGRAF 目录"一节，以对 ISaGRAF 硬盘体系结构有更多了解。当所有 ISaGRAF 文件拷贝完毕，在程序管理器窗口中会增加下列组标：



这里有 ISaGRAF 主要的图标：

- 项目**：..... 项目管理
- 库**：..... 库管理
- 书**：..... 有关 ISaGRAF 在线信息
- 调试**：..... 为用户提供的诊断工具
- 自述**：..... 有关 ISaGRAF 新版本的信息
- 报告**：..... 标准故障报告表格

在遇到问题时，请使用标准故障报告表格。先打开表格，然后填写所需的各项，使用文件/另存为菜单命令，将表格存到给定文件名的文件中。然后再将该文件使用传真或电子邮件发到 ICS Triplex ISaGRAF Inc.公司。



## 修改系统文件

安装完成后，在重新启动计算机之前需要对 CONFIG.SYS 文件进行修改。但 ISaGRAF 的目录路径名不能插在 PATH 变量中。

ISaGRAF 软件不使用任何 MSDOS 环境变量。然而，应该将下列语句添加到 CONFIG.SYS 文件中：

```
files=20  
buffers=20
```

ISaGRAF 工作台可使用串行口与 ISaGRAF 目标 PLC 进行通讯。ISaGRAF 缺省的串行口是 COM1。如果鼠标器也使用串行口，则应为鼠标器选择 COM2。这样，缺省的 COM1 串行口对于任何新的 ISaGRAF 应用程序都是有效的。

在 CONFIG.SYS 文件修改完毕后，为了使改变起作用，必须重新启动计算机。

对于 WindowsNT 用户，有一点是很重要的：

当在 WindowsNT3.51 或 4.00 版本上使用工作台时，下列行必须插在 \ISAWIN\EXE 目录下的 ISA.ini 文件的 [WS001] 节内：

```
[WS001]  
NT=1  
Isa=C : \ISAWIN  
IsaExe=C : \ISAWIN\EXE  
IsaApl=C : \ISAWIN\APL1  
IsaTmp=C : \ISAWIN\TMP
```

这对于 RS 通讯是绝对需要的。

### A.1.2 口令管理

您现在安装的 Isagraf 版本允许您建立控制应用。在您要求正式口令之前,您可以使用为期 30 天的临时版本 Isagraf。当没有口令时,您不能输出 IEC 文档 到库文件,不能输出变量,不能 download Workbench,上的源程序到目标板上,也不能从目标版上提取源程序。您必须有口令,才能有完全操作版的产品

您可以用软件或硬件来管理 Isagraf。当使用硬件狗时,硬件狗将被安装在并口或 USB 口,这个狗在发货之前,就被遍程好选定的功能。当您选择软件狗时,您将需要授权。

硬件狗可以连接到任何并口上。如果计算机有一个以上的并口,则最好把硬件狗和打印机装到不同的并口上。对于某些计算机/打印机的配置来说,当它的输出被连接到无线打印机时,硬件狗可能灰不被认出。在这种情况下,要么切断打印机,要么把打印机改成有线连接,然后,再开始 ISaGRAF Workbench。

**注:**当在 Windows NT 系统上使用硬件狗时,您必须安装哨兵驱动器 (Sentinel Driver),确保这只狗可以被看见。通过双击位于哨兵文件夹中 Isagraf CD-ROM 根目录中的"Setup.exe.", 然后屏幕上便出现下列指示:

ISaGRAF 现具有下列两种特性:

- 有限数量的 I/Os, I/Os 数量的范围从 1 到 4095
- 无限数量的 I/Os

两种特性都包括了 ST(结构文本)和 IL(指令清单)的编程语言的使用..

然而,为了使用 Workbench 中现有其它语言,您需要规定下列:

- SFC (序列功能表里一致

- FC (流程图))
- FBD (功能方框图)
- LD (阶梯图)

尽管所有口令均为单机版,但您可以把口令从一台机器转移到另外一台机器上.

到达口令管理员的方法:

开始 Windows 的菜单, 选择程序, 然后进入 **ISaGRAF 3.5**, 然后进入 **Licensing**.

#### **A.1.2.1 增加口令**

为了获得 SaGRAF 的口令:

当注册 ISaGRAF.口令时,您仅需要一套用户代码和注册狗.

1. 在增加 Licensing 菜单上, 从现有的组件清单上选择 ISaGRAF.
2. 移动光标到选定的组件上

接下来您将选择功能: 有限 I/Os, 或无限 I/O 口. 对于有限 I/Os 功能模块来说, 您需要在 1 到 4095.之间指定一特定数量.

A 建立代码, User Code 1, and User Code 2 将出现在它们相应的区域

3. 发出注册口令信信息:

点击 « 发出 »

一个预先设定的 email 地址将出现,将您的联系地址和订单号发出. 同时提供信用卡号

其它所有相关信息也将发 email.

之后,包括原始代码和用户代码, 以及注册口令 1 和注册口令 2 将通过 e-mail.返回.

4. 当收到口令时,确保原始代码和用户代码跟 License Manager 窗口一样, 然后复制并粘贴到它们相应的地方.

5. 点击 **Proceed**.

如果口令正确,在选择组件上将出现 ISaGRAF 灰暗色.

6. 当注册口令结束,停止并从新开始 ISaGRAF.

#### A.1.2.2 转移 Licensing

您可以把口令从一台计算机转移到另一台计算机

为了把口令转移到另一台新的计算机,您需要建立一个口令转移软盘,这个软盘是从新格式化的并是空盘.然后把口另拷贝到软盘上

1. 在新计算机上做如下工作 :

a) 安装 ISaGRAF.

b) 插入软盘

c) 开始 Windows 菜单, 选择 **Programs**, 然后进入 **ISaGRAF 3.5**, 然后进入 **Licensing**.

d) 在转移口令菜单上, 选择驱动软盘, 然后点击“**创建转移盘**”  
这时候,口令转移盘即被创建好.

- e) 从新机器上取下口令转移盘.
- 2. 把口令从当前的机器上转移到新机器上
  - a) 在当前有口令的机器上,插入 口令转移软盘.
  - b) 从 Windows 菜单开始, 选择 **Programs**, 然后进入 **ISaGRAF 3.5**, 然后进入 **Licensing**.
  - c) 在转移口令菜单上, 选择驱动,抓住转移盘, 然后点击**转移口令到软盘上**.

这时从计算机上取下口令,并拷贝到转移盘上.

- d) 从计算机上取下口令转移盘.
- 3. 在新计算机上安装口令:
  - a) 在新机器的软驱上,插入带有口令的转移盘.
  - b) 从 Windows 的菜单开始, 选择 **Programs**, 然后进入 **ISaGRAF 3.5**, 然后进入 **Licensing**.
  - c) 在转移口令菜单上, 选择驱动,抓住转移盘, 然后电击转移完毕.

这时候,口令已成功转移到新机器上,Isagraf3.5 便可以运行了.

### A.1.2.3 删除口令

您可以从计算机上删除合法口令.

按照以下 6 点来删除合法口令:

- 1. 从 Windows 的开始菜单中,选择"程序",然后 ISaGRAF 3.5,然后 License Manager;
- 2. 在删除口令标签上,从口令目录表上,选取您想要删除的口令;
- 3. 点击->删除已选取的内容;

连同用户代码 1 和用户代码 2 一起,建立代码出现在它们对应的领域;

**4. 发送口令信息:**

**a) 点击"发送"**

一个预先设定的 email 地址出现;

**b) 把包括用户代码 1,用户代码 2 及建立代码等所有信息都发出;**

一个用户代码,注册键 1 和注册键 2 将通过 email 返回;

**5. 收到后,把用户代码和两个注册键输入到它们对应的地方,然后点击"进行"**

一个确定代码出现在;

**6. 把这个确定代码,以及您的名字,地址和电话等都回复这个 email.**

### **A.1.3 在线信息的使用**

在线信息随同 ISaGRAF 工作台一起安装, 有下列内容:

ISaGRAF 语言参考手册

完整的用户指南(对于任何 ISaGRAF 工具)

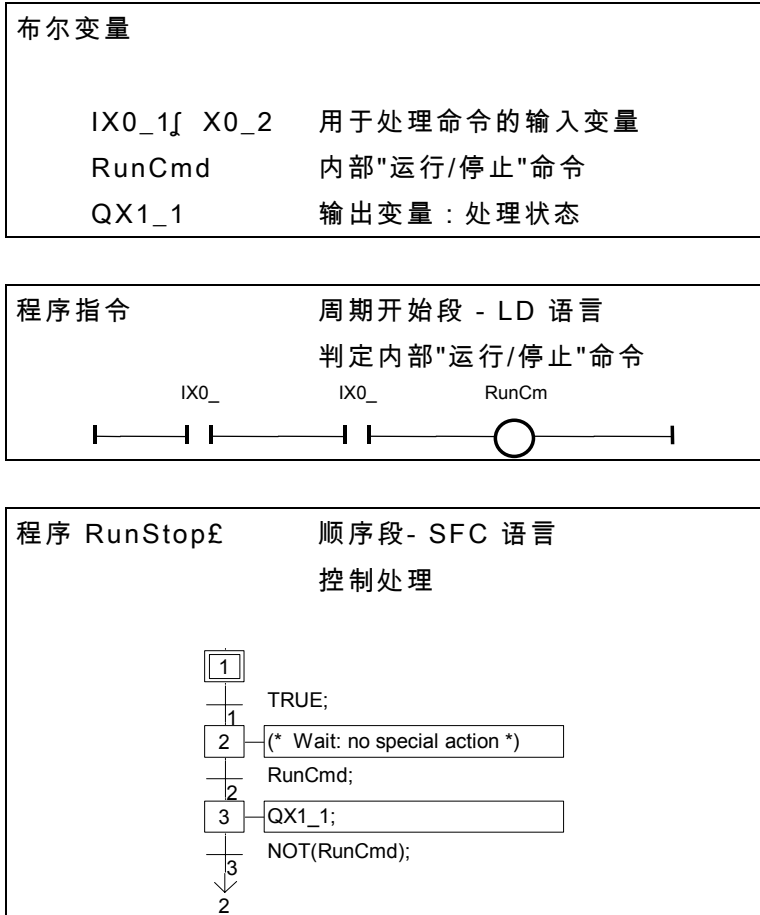
库中元素的技术要点

对于任何一个 ISaGRAF 窗口, 都选择"帮助"菜单中的选项来显示在线信息。

### A.1.4 样板应用程序

本节循序渐进地说明创建、设计、生成和测试短而完整的多语言应用程序所需的全部基本操作。

下面是一个完整地混合使用 LD 和 SFC 的应用程序的规范：



## 开始 *运行 ISaGRAF 工作台*

从 Windows 开始菜单中的"ISaGRAF"组中，运行"项目"命令，这样就可以运行 ISaGRAF 工作台。



### *建立项目*

使用"文件"菜单的"新建"命令或按下"新建"按钮，就可以建立项目(称为"RunStop")。在打开的对话框中：

输入项目名：**"RunStop"**

选择输入/输出配置：**"Sim\_Boo"**

按下**"确认"**按钮。

现在项目已经建立好。



### *打开项目*

打开 ISaGRAF 程序管理器窗口就可以定义项目程序。使用项目管理窗口中的"打开"命令或在项目名上双击鼠标器或使用编辑按钮，就可以打开项目。



### *建立程序*

现在，程序管理器窗口已经打开，并且是空的(没有定义过的程序)。使用"文件"菜单中的"新建"命令或新建按钮建立第一个程序。在打开的对话框中：

输入程序名：**" Command"**。

选择**"快捷 LD"**语言。

选择**"循环开始"**段。

按下**"确认"**按钮建立该程序。

建立第二个程序必须重复同样的操作：

使用"文件"菜单中的"新建"命令或按下新建按钮。在打开的对话框中：

输入程序名："RunStop"。

选择"SFC"语言。

选择"顺序"段。


按下"确认"按钮建立该程序。


现在以上程序已经建立完毕，程序出现在程序管理器窗口中。



### 变量的声明

在输入程序前，编程中所使用的内部变量必须声明。使用"文件"菜单中的"字典"命令或按下字典按钮，可以完成此项功能。当项目建立完毕后，可以对输入/输出变量进行自动声明。

 现在，字典窗口已经打开了。可以使用"文件"菜单中的的"其它"子菜单中的"全局变量"子菜单，和"布尔变量"命令,来选择全局布尔词典。这和按下全局对象按钮和布尔按钮具有同样的功能。

 "编辑"菜单中的"新建"命令可用来建立新的布尔变量，也可以使用插入对象按钮来建立新的布尔变量。打开对话框，输入内部变量的描述：

名称：	RunCmd
注释：	运行/停止命令：内部
属性：	选择"内部"属性
按下"保存"按钮：	该变量建立完毕。
按下"取消"按钮：	退出对话框。

最后退出字典编辑器，保存修改内容输入：使用"文件"菜单中的"退出"命令。单击"是"保存修改内容。



### 编辑快捷 LD 程序

双击程序管理器窗口中的程序名称或使用编辑按钮，可开始编辑 "Command" LD 程序。



现在，ISaGRAF 快捷 LD 编辑器窗口已经打开了。将窗口充满屏幕以增加工作区域。

**F2 F3** 按下 F2 和 F3 键：



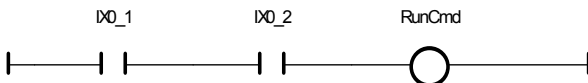
将各个变量和 LD 符号联系在一起：使用键盘箭头键移动光标。将光标位于每一个符号，按下回车键。变量段的对话框打开。

第一个触点，在变量选择框输入：IX0\_1 然后回车。

第二个触点，在变量选择框输入：IX0\_2 然后回车。

线圈，在变量选择框输入：RunCmd 然后回车。

现在程序已经完成。结果如下：



从编辑器退出，保存修改内容输入：使用"文件"菜单中的"退出"命令。单击"是"保存修改内容。



### 编辑 SFC 程序

在程序管理器窗口中双击 SFC 程序名或使用编辑按钮，可开始编辑 SFC 程序"RunStop"。



现在，SFC 编辑器窗口已经打开了。将窗口充满屏幕以增加工作区域：



初始步已经存在并被选中。按下键盘箭头键"↓"键选择空单元，在初始步(0,1)后

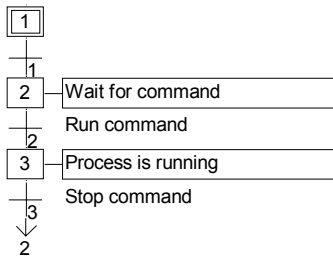
**F4 F3** 按下 F4，然后按下 F3，插入一个程序步和转换。

**F4 F3** 按下 F4，然后按下 F3，插入一个多程序步和转换。

**F5** 按下 F5，插入一个跳转程序步，并选择 GS2 作为跳转的目的。



现在，这个图表已经完成。按下工具条中的"缩放"按钮来增加单元的尺寸，并给出一定的空间以显示二级指令。 以下是这个图表：



使用键盘上的箭头键选择程序名，按下"回车"键可以输入转换"2"的程序。这时第二层编程窗口被打开。可以输入第二层程序：

**RunCmd;**

**^TAB** 按下"Ctrl+Tab"键，把焦点移回到 SFC 图上，移动选择到 3 步上，按下"回车"键，进行第二层文本编辑：

**QX1\_1;**

采用相同的步骤，进行转换 3 的文本输入：

**Not (RunCmd);**

**^F4** 按下"Ctrl+F4"键，关闭第二层窗口。

现在 SFC 程序已经编辑完成。使用"文件"菜单的"退出"命令退出编辑器，单击"是"保存修改。



### **建立应用程序代码**

使用程序管理器窗口的"生成"菜单中的"生成应用程序"命令，可以建立应用程序代码或建立工具条按钮。

当代码生成完成后，将出现对话框，询问用户是**现在**退出代码生成器还是**继续**在代码生成器上工作：按下"退出"按钮。



### **仿真**

使用程序管理器窗口中的"调试"菜单和"仿真"命令，可以运行 ISaGRAF 核心仿真器或工具条上的按钮。


当仿真器窗口出现时，可以进行应用程序的调试。在本例中，输入 1 和输入 2(都是绿色按钮)必须按下，以运行这个处理过程(此时代表输出的红色发光二极管亮)。

关闭诊断器窗口，从仿真模式退出：使用"文件"菜单中的"退出"命令。

## A.2 项目管理

为了运行 ISaGRAF 项目管理工具，在 ISaGRAF 组内，用鼠标双击“项目”图标，“项目管理”窗口被打开。

适合于 PLC 回路的一个项目在一个目标 PLC 上运行。窗口的上部包含已有的项目表。被选择项目的文本说明显示在窗口的下部：

 改变窗口的大小

只需在项目表和项目描述之间的分隔线上单击就可以改变相应窗口的大小。项目说明窗口不能完全被隐藏。它总是含有至少一行文本。



### **插入分隔符**

在任何项目名称的前面可以插入一条分隔线。它允许在项目表的排列图中把与相同应用相关的一些项目组成组。使用“**编辑/切换分隔**”命令在被选择的项目前面插入或删除一个分隔线。



### **移动表中的项目**

为了移动表中的项目，首先你必须选择（高亮显示）它。然后在它的名称上单击，并把它拖动到表中一个新的位置上。当拖动项目时，在左侧边缘上的一个小箭头指示项目将被放置的地方。你也可以使用“**编辑**”菜单的“**移动**”命令，逐条地移动所选择的项目。注意，如

果被选择的项目前面有一个分隔线，那么该分隔线与项目一起移动。

### A.2.1 建立和操作项目

项目管理菜单的命令用于建立新的项目，编辑项目并管理已有的项目。



#### **建立一个新项目**

为了建立新项目，首先要输入它的名称，即建立一个空项目，此时项目中没有对象。新建的项目可以加一个 I/O 配置，但必须在库中定义这个 I/O 配置。ISaGRAF 自动地建立 I/O 连接，并在新项目字典中声明相应的 I/O 变量。当建立或重新命名一个项目时，你必须遵循命名规则：

- 名称不能超过 8 个字符
- 第一个字符必须是一个字母
- 后面的字符可以是字母，数字或下划线符
- 项目的名称与大小写无关

当建立一个项目时，使用“**编辑/设置注释文本**”命令输入文本，该文本与表中的项目名称一起显示。



#### **编辑项目的说明**

“**项目/项目简介**”命令用于编辑项目的文本说明。该文档能从项目表上的其它条款中识别项目。项目简介在项目使用期限内可以用于记录任何备注。



## 编辑项目

“文件 / 打开”命令为选择项目打开程序管理窗口。从这个窗口可以管理项目的所有内容（程序，应用参数...），也可以在一个项目名称上双击，以便编辑它。



## 修改历史记录

ISaGRAF 系统把与项目组件有关的任何修改存储在一个历史文件中。在历史文件中，对每次修改用标题、日期和时间来识别。历史文件包含最后 **500** 次修改。每个项目有一个历史文件。对项目进行修改的历史文件是对附加在项目程序上的“日志”文件的补充。“项目 / 历史记录”命令允许用户察看或打印被选择项目的修改历史。在主表中用户可以选择一个或多个项，并按下面的按钮：

确认.....关闭窗口

打印.....把表中的内容传送给打印机

帮助.....显示关于这个对话框的帮助信息

[删除] 选择从表中清除(删除)所选择的行

[删除] 所有清除整个表

查找.....在表中查找

位于“**查找**”按钮上方的输入框用于输入一个搜索模式。这一功能与大小写无关。当搜索到表的底部时，它便从表的上部到起始位置继续进行搜索。



### **打印一份完整的文档**

“项目 / 打印”命令允许用户建立和打印有关所选择项目的一份完整的文档。这份文档可以把所选择项目的任何部件(程序, 变量, 参数)集合在一起。为了建立一份特殊的(不完整的)文档, 用户只须定义它的内容表。



### **口令保护**

“项目 / 设置口令”命令使用户能够为所选择项目的工具和数据设置口令保护。关于口令的级别和数据保护的更多信息查阅本书指南“**口令保护**”一章。口令仅与所选择的项目有关。它们对其它项目和 ISaGRAF 库没有影响。

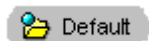
## **A.2.2 操作几个项目组**

一个 ISaGRAF 项目与磁盘上的一个目录有关, 在磁盘上保存所有的项目文件。一个“项目组”部件与项目的目录表有关, 这些目录集合在相同的根目录下。一个项目组用一个名称来识别。作为缺省值, ISaGRAF 建立两个项目组:

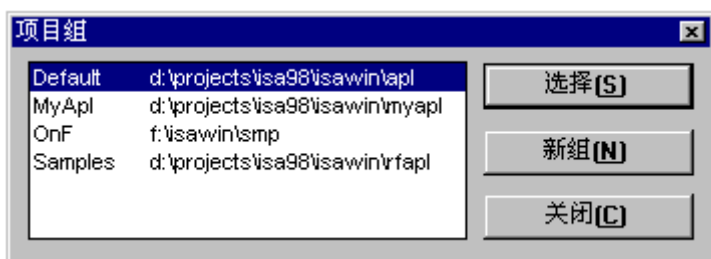
“**缺省**”            在“\ISAWIN\APL”上你的工作区域

“**范例**”            在“\ISAWIN\SMP”上使用 ISaGRAF 工作平台范例应用程序

当前所选择的项目组的名称被写在工具栏内，该工具栏在用于选择一个项目组的按钮旁边



你也可以运行“文件/选择项目组”命令，以便选择一个已有的组或建立一个新的组。下面的对话框被打开：



选择表中的一个组，并按“**选择**”按钮以便在项目管理表中激活它。为了选择它，你也可以在它的名称上双击。使用“**新组**”命令去建立一个新的组，该命令可被用于把一个组的名称指定给一个已有的目录，或者使用一个新的目录去建立一个新的组。

当其它的 ISaGRAF 窗口 (程序管理器，编辑器) 打开时，不可以选择或建立组。

### A.2.3 选项

“**选择**”菜单的命令用于显示或隐藏工具栏，选择文本的字体，以及设置项目管理器的“自动关闭”方式。所

选择的字体用于显示项目的说明，它也被所有的 ISaGRAF 文本编辑器使用。

若清除了“**保持项目管理器打开**”选项，那么当输入一个项目时，项目管理器窗口被自动地关闭。

#### A.2.4 工具

“**工具**”菜单的命令用于运行其它的 ISaGRAF 应用程序。“**工具/档案项目**”命令运行 ISaGRAF 的存档管理器以便备份或再恢复项目。“**工具/档案公共数据**”命令用于备份或再恢复所有项目使用的文件(例如公共被定义字)。

“**工具/库**”命令在一个分隔窗口中运行 ISaGRAF 的库管理器。

“**工具/输入 IL 程序**”命令根据 PLC 打开文件的交换格式输入项目，该项目在一个文本文件中被描述为一个单独的 IL 程序。

## A.3 程序管理

通过可选用的菜单命令,在程序管理窗口中显示应用程序(也称作模块或编程单元)和程序的集合,以便建立项目结构、运行编辑器、编译器和诊断器。该窗口是开发应用程序时的工作平台核心。当在项目管理窗口中运行“打开”命令时,程序管理窗口被打开。

### A.3.1 一个项目的部件

一个项目的部件被称为**程序**。一个程序是一个逻辑的整体,它描述控制执行的一个部分。应用中的任何程序可以使用全局变量(例如输入/输出变量)。局部变量仅可以由一个程序使用。程序以**树状结构**排列,并被分成不同的**逻辑程序段**。窗口显示程序和它们之间的链接。“**最上层**”程序出现在树状结构的左侧。

#### ☐ 最上层程序

最上层程序出现在树状结构的左侧。位于首三个程序段的最上层程序在运行周期内(扫描)总是为激活状态,并按照下面的顺序执行:

- (读输入)
- 执行 **BEGIN** 程序段的最上层程序
- 执行 **SEQUENTIAL** 程序段的最上层程序
- 执行 **END** 程序段的最上层程序
- (刷新输出)

“**Begin**”或“**End**”段的程序描述操作周期。它们并不取决于时间。“**Sequential**”段的程序描述操作顺序，在此，时间变量明显地出现，以便区分基本的操作。“**Begin**”程序段的主程序在每次运行周期的开始有次序地执行程序。“**End**”程序段的主程序在每次运行周期的末尾有次序地执行程序。“**Sequential**”程序段的主程序在**SFC**或**FC**规则的基础上执行程序，该程序必须用**SFC**或**FC**语言编写。循环程序段的程序不能用**SFC**或**FC**语言描述。任何程序段的任何程序可以拥有一个或多个子程序。



### **函数和功能块**

一个项目中的任何程序段的任何程序可以调用“**函数**”程序段的程序。一个函数是一种算法，它根据几个输入值处理一个输出值。一个函数的算法只是操作易失的中间变量，当一个函数又被调用时，其中间变量被删除。这意味着一个函数决不能调用一个函数块。“**函数**”程序段的一个程序不能用**SFC**或**FC**语言描述。与函数不同，“**功能块**”使一个操作输入值的算法与隐藏的静态数据相联系，这些静态数据在功能块的每个不同的应用中被系统拷贝(实例)。一个项目中的任何程序段的任何程序可以调用“**功能块**”程序段的程序。“**功能块**”程序段的程序不能用**SFC**或**FC**语言编程。

## 子程序

子程序是向一个 ( SFC, FC 或其它的 ) 主程序提供某种功能。一个子程序仅可以被它的主程序执行 ( 调用 )。每个程序段的每个程序可以有一个或多个子程序。除了 **SFC** 和 **FC** 以外的任何语言可被用于描述一个子程序。

## 子 SFC 和 FC 程序

一个 **子 SFC 程序** 是一个并行的程序，它可以被它的主程序启动或停止。主程序和子程序都必须用 **SFC** 语言来描述。

当一个主程序启动一个子 **SFC** 程序时，它把一个 **SFC** 标记放入下级程序的每个初始步中。当一个主程序停止一个子 **SFC** 程序时，它清除下级程序步中所有的标记。







顺序程序段上的任何 **FC** 程序可以控制其它的 **FC** 子程序。在执行一个 **FC** 子程序期间，一个 **FC** 主程序被停用 ( 等待 )。 **FC** 主程序和它的其中一个子程序不可能同步执行。

## 程序和子程序之间的链接

在树状结构中，用一根线把子程序和下级程序与它们的主程序链接在一起。一个 **SFC** 程序和一个 **SFC** 下级程序之间的链接用一个箭头结尾。注意，这样一个链接表示 **并行操作**。

## 编程语言

每个程序用唯一的一种语言描述。这种语言是在建立程序时选择的，以后不能改变。然而，**FBD**图可以包含用**LD**编写的部分程序，并且**LD**图可以包含对功能块的调用。可供选用的图形语言是：**SFC**（顺序功能流程图），**FC**（流程图）**FBD**（功能块图）和**LD**（梯形图）。可供选用的文字语言是：**ST**（结构化文本）和**IL**（指令表）。**SFC**和**FC**语言是专供顺序程序段上的主程序和下级程序使用。在程序管理窗口中，每个程序的语言用程序名称旁边的一个图标表示。下面是用于表示语言的图标：

-  **SFC**.....顺序功能流程图
-  **FC**.....流程图
-  **FBD**.....功能块图
-  **LD** .....梯形图(用快捷LD编辑器输入)
-  **ST** .....结构化文本
-  **IL** .....指令表

### A.3.2 操作程序

“文件”菜单把所有用于建立，更新或修改程序的命令集合在一起。它也激活合适的编辑器以便输入应用程序的内容。



## 建立一个新程序

“文件”菜单中的“新建”项允许在每个程序段建立最上层程序、下级程序或子程序。需要输入的第一段信息是按照下列规则命名的新程序的名称：

- 一个名称的最大长度为8个字符
- 第一个字符必须是一个**字母**
- 后面的字符必须是**字母、数字**或'\_'字符
- 一个程序的命名不受字母大小写的影响

下一步是选择用于描述新程序的编辑语言：

SFC.....顺序功能流程图

FC.....流程图

FBD.....功能块图

LD.....梯形图(用快捷LD编辑器输入)

ST.....结构化文本

IL.....指令表

最后，选择程序的一个执行样式：

开始.....“开始”程序段的最上层

顺序.....“顺序”程序段的最上层

结束.....“结束”程序段的最上层

函数.....在“函数”程序段内

功能块.....在“功能块”程序段内

子程序... ..SFC 或一个已有程序的 FC 下级程序或子程序

选择前五个选项中的一个,程序被放在一个**开始,结束,顺序,函数或功能块**程序段的最上层。后面的选项表明新程序是一个**SFC**子程序或一个**FC**子程序或一个子程序。记住,一个最上层的顺序程序必须用**SFC**或**FC**语言描述,并且**SFC**和**FC**语言不能用于描述循环程序和它们的子程序。

## **输入每个程序的注释**

ISaGRAF 允许用户给项目的每个程序附加一段描述文本。这个注释在程序名的旁边用较小的字体显示。使用“**文件/程序注释文本**”命令输入或改变附加给所选程序的注释。



## **编辑一个程序的内容**

这个命令允许对一个程序的内容进行修改。用于输入一个程序的编辑器取决于为该程序所选择的语言。对程序的编辑是在各自的窗口进行的,这样,就有可能通过并行窗口编辑一个以上的程序。按**ENTER**键允许编辑高亮显示的程序。为了编辑程序,用户也可以用鼠标在程序的名称上双击。

## **编辑“日志”文件**

给每个程序都附加一个日志文件。它是一个文本文件,它包括在程序使用期内有关对程序进行修改的所有注解。在任何时候,日志文件可以被编辑,被自由地修改或打印。当离开用于修改一个程序源代码的编辑器时,一个窗口被自动打开,以便为日志表输入注

解。带有修订的日期和时间的注解插入到日志文件中。



### 变量的字典

“文件/字典”命令运行字典编辑器，项目变量在这里被声明。对所选择的程序来说，变量可能是全局型的（可被项目中的任何程序识别）或者是局部型的。字典编辑器也可用于声明定义的字，这些字是语义上的别名，用于代替程序源代码中的一个名称或一个表达式



### 一个函数、子程序或功能块的参数

“文件/参数”命令允许用户定义被选择的子程序，函数或功能块的调用及返回参数。如果在程序管理窗口中选择了“开始”或“结束”程序段中的一个主程序或一个SFC程序，那么这个命令是无效的。

子程序、函数或功能块最多可以有32个参数（输入或输出）。为了符合ST语言的书写习惯，一个函数或子程序总有一个（并且是唯一的一个）返回参数，该参数的名称必须与函数的名称相同。

位于窗口左上方的表按调用方式的顺序显示参数：第一个是调用的参数，最后一个返回参数。窗口的下面部分所显示的是对当前在表中所选择的变量的详细描述。ISaGRAF的任何数据类型可被用于一个参数。在表中，返回参数必须放在调用参数的后面。命名参数必须符合下列规则：

- 一个名称的最大长度为16个字符

- 第一个字符必须是一个字母
- 后面的字符必须是字母、数字或 '\_' 字符
- 一个参数的命名不受字母大小写的影响

“插入”命令用于在被选择的参数前插入一个新参数。  
“删除”命令用于删除被选择的参数。“编排”命令自动地重新排列(排序)参数，以便把返回参数放在表的最后。

### **移动树状结构中的一个程序**

“文件”菜单中的“重命名/移动”命令用于改变一个程序的名称，或把它移到树状结构的另一程序段中。然而一个已有程序的描述语言不能被改变。运行这个命令时，一个与建立程序时所用窗口相同的窗口被打开，窗口的各个区域设置为所选定程序的特征。可以修改一个程序的名称，也可以选择另一个程序段或主程序，以便把它移到树形结构中。

“文件”菜单中的“排列程序”命令被用于在同层次的程序和主程序之间确定一个明确的排列顺序。如果被选择的程序处于最上一层，则该命令用于排列所选程序段的最上一层的程序。如果被选择的程序位于下一层，则该命令只排列 SFC 的下一级程序及与被选程序有着同样主程序的子程序。当“排列程序”对话框打开时，选择你想要移动的程序并按“往上”或“往下”按钮，以便在表内移动它。



## 拷贝程序

为了拷贝一个程序，在程序表中选择源程序，并运行“文件/拷贝”命令。运行该命令时，一个与建立程序时所用窗口相同的窗口被打开，窗口的各个区域会显示所选程序的特征。输入目标程序名和它在树状结构程序段中的位置。如果目标程序不存在，那么它被建立在给定位置上。如果目标程序已经存在，那么它被重写。所有的局部说明语句和定义的字同程序一起被拷贝。目标程序的描述语言必须与源程序所使用的语言相同。按“确认”按钮拷贝源程序。

“文件”菜单的“拷贝到其它项目”命令是把所选择的程序用同一名称拷贝到另一个项目中。所选择的程序的下一级SFC程序和子-程序同它一起被拷贝。被选择的程序和它的下一级程序的名称不必在目标项目中使用。不能使用这个命令重写程序。所有附加的局部说明语句和定义的字与程序一起被拷贝。



## 删除程序

为了删除一个程序，首先从程序表中选择程序，然后运行“文件/删除”命令。含有下一级或子-程序的程序不能被删除。为了删除含有下一级或子-程序的程序，必须首先删除下一级或子-程序。所有局部说明语句和定义的字与程序一起被删除。

### **输入来自库的函数或功能块**

为了把库中用 IEC 语言编写的一个函数或功能块拷贝到打开项目的“函数”或“功能块”程序段上，使用“工具/自库输入”命令。附加到被输入的函数上的局部变量和定义的字与它一起被拷贝。当已经正确地输入完来自库的一个函数时，使用“文件/重命名/移动”命令，可以把它放在树状结构中的另一个程序段或另一个位置上。为了避免命名的不一致，当把函数或功能块输入到项目区域时，必须对它们重新命名。不要忘记对一个函数的返回参数也要重新命名。

### **向库中输出函数或功能块**

“工具/输出到库”命令用于把(在打开项目中的)“函数”或“功能块”程序段的一个程序发送到合适的库中。附加在被输出的函数或功能块上的局部变量和定义的字同它一起被拷贝。在 ISaGRAF 库管理器中，必须重新编译被输出的函数或功能块，以保证它在库的环境中可以被使用。库的函数或功能块不可以使用全局变量。

## **A.3.3 运行代码生成工具**

“制作”菜单的命令用于运行代码发生器，以便输入产生应用代码时使用的选项和附加的数据。关于这些工具的进一步信息查阅本资料中的“使用代码生成器”一章。



### **制作应用代码**

“生成”命令启动项目的代码生成。在运行该命令之前，必须正确地设置目标代码生成的选项。在产生目标代码之前，为了检测语法错误，检查还未被校验的任何程序。ISaGRAF 包含一个增量编译器，它不重新编译已经被编译完的程序。



### **校验被选择的程序**

“校验”命令允许用户校验在表中当前所选择的程序的语法。当校验一个程序且没有检测出错误时，那么在代码生成期间，该程序不再被重新校验，直到程序的内容或相关的定义的字或变量改变。



### **仿真一个修改**

“尝试”命令仿真每个程序的一个修改，结果，在下次代码生成期间，将被再次编译。



### **应用程序运行时间选项**

该命令打开一个对话框，在此输入执行应用程序的主要运行时间参数。其中包括周期编程，运行时间错误管理，启动方式，保持变量的硬件实现。关于这个命令的更多解释，查阅本资料中的“使用代码生成器”一章。

## **编译器选项**

该命令用于建立 ISaGRAF 代码发生器使用的选项，以便产生和优化目标代码。关于这个命令的更多解释查阅本资料中的“使用代码生成器”一章。

## **定义资源**

“资源”是用户定义的数据（例如一个文件），它必须与目标代码合并以便同目标代码一起被下载。关于资源定义文件格式的更多解释，查阅本资料中的“使用代码生成器”一章。

定义的资源接受相关的路径名称（外来数据将并入下载的代码）。您可以使用“.”相关路径来规定位于项目文档中的输入文件。This applies to both TEXTFILE and BINARY FILE resour, for the file specified in the "FROM" statement..

### **A.3.4 其它 ISaGRAF 工具**

“项目”菜单是一组命令的集合，这些命令是为所选择的项目运行 ISaGRAF 工具。关于这些工具的更多信息查阅本资料中的有关章节

## **连接输入/输出变量**

“IO 连接”命令运行 ISaGRAF 输入/输出变量连接编辑器。该工具用于在项目字典中声明的输入/输出变量与相应的输入/输出硬件之间建立关系。



## **运行交互引用编辑器**

“交叉引用”命令允许用户计算，观看，打印项目的交互参考。交互参考向用户显示在整个项目里程序的源代码中每个变量出现的所有地点。这个功能对于检测一个变量或任何全局资源的存取，或者列出源代码中的一个全局变量出现的所有地点是非常有用的。



## **输入项目描述**

“项目简介”命令用于编辑项目的文本描述。该文档能完全识别项目表上的其它项目。项目描述也可用于记录在项目使用期内的任何备注。项目描述在项目管理窗口中显示。



## **打印一份完整的文档**

“打印项目文档”命令允许用户建立和打印有关所选择项目的一份完整的文档。这份文档可以把所选择项目的任何部件(程序,变量,参数)集合在一起。为了建立一份特殊的(非完整的)文档,用户只须定义它的目录表。



## **修改的历史记录**

该命令打开一个对话框，在此显示项目修改的历史记录。关于这个命令的更多解释查阅本资料中的“项目管理”一章。

### A.3.5 向工具菜单中添加命令

ISaGRAF 提供了把其它命令插入“工具”菜单中的方法。用户定义的需要添加的命令在“\ISAWIN\COM\ISA.MNU”文本文件中描述。你最多可以添加 10 个命令。可以把注释插在任何一行上，且用“；”字符开始。按照下面的语法，每个命令用两行文本描述：

```
M=menu_string  
C=command_line
```

菜单字符串是显示在“工具”菜单中的文本。命令行是任何 MS - DOS 或 Windows 的可执行命令，并可加入参数。在命令行中，你可以用“%A”字符代替打开的项目名称，并用“%P”字符替代被选程序的名称。下面的例子是运行“写字板”来编辑被选程序(与 ST 和 IL 程序一起使用)：

```
M=Edit with Notepad  
C=Notepad.exe \isawin\apl\%A\%P.lsf
```

### A.3.6 仿真和调试应用程序

“诊断”菜单的命令用于运行 ISaGRAF 图形诊断器，以仿真模式或实际的连接模式。



## 仿真

“仿真”命令以仿真模式打开诊断器。在这种模式下，将出现被称为仿真器的另一个窗口。当没有连接目标设备时，用该命令去测试任何应用程序是非常有用的。启动仿真器，关闭程序管理窗口。在诊断器和仿真窗口均被打开后，用诊断模式重新打开程序管理窗口。如果目标代码尚未生成，则不能启动仿真器。当下一级窗口(编辑器，生成代码，I/O连接)被打开时，不能启动仿真器。在运行该命令之前，它们中的每个窗口必须关闭。ISaGRAF编辑器菜单中也有这个命令。



## 实诊断

“诊断”命令打开诊断器的主窗口，并关闭程序管理窗口。一旦在诊断器和目标应用程序之间建立起通信，则程序管理窗口以诊断模式再次被打开。如果目标代码尚未生成，则不能启动诊断器。当下一级窗口(编辑器，生成代码，I/O连接)被打开时，不能启动仿真器。在运行该命令之前，它们中的每个窗口必须关闭。ISaGRAF编辑器菜单中也有这个命令。



## 准备诊断工作空间

“诊断/工作空间”命令使你可以为初始工作空间定义一个文档表。这些文档可以是程序、注视点图形、变量表。来自ISaGRAF以前版本的图形和时序图表也和项目文档一起列表。当启动仿真或在线监视时，在初始工作空间内定义的文档被自动打开。



该对话框把已存在的项目文档显示在左侧，为初始工作空间选择的文档显示在右侧。使用“>>”和“<<”按钮把文档从一个表移动到另一个表。每个项目有它自己的初始工作空间文档表。



### 建立链接

“建立链接”命令使用户可以对链接参数进行定义，该链接参数被用于在（PC机主站上的）诊断器和目标ISaGRAF系统之间建立通信。

“从站编号”用于识别目标ISaGRAF系统或任务。该编号可为1至255。要想知道所用目标系统的从站编号，请查阅目标供应商指南。

“通信端口”用于识别ISaGRAF工作平台和目标之间的硬件媒体。它可以是一个串行口的名称或“以太网”，即使用“Winsock”1.1版本的备用TCP-IP通信。

“超时”是留给目标系统的时间，目的是为了在一个诊断问题的结束和该问题开始作出反应之间完成通信操作。这个时间被设置为以毫秒为单位的数。“重试”域是自动试验的次数，诊断程序在检测一个通信错误之前执行的通信操作。

### 建立串行链接

当选择一个串行端口 (COM1..4) 时, “建立” 按钮被用于存取其它的串行链接通信参数。

你可以选择传输的波特率、奇偶校验和格式。当你为“流控制”选择“硬件”选项时, 则 ISaGRAF 工作平台控制 CTS 和 DSR 线, 完成硬件在交换信息时的握手信号。

### 建立以太网链接

当选择“以太网”作通信端口时, 用“建立”按钮为 TCP-IP 通信输入“因特网地址”和“因特网端口”号。

这些区域使用由 Socket 接口定义的标准格式。工作台采用用于 TCP-IP 通信的 WINSOCK 动态连接库版本 1.1. 该文件务必被正确地安装在硬盘上。当运行 ISaGRAF 目标时, 如果没有设定端口号, 则“1100”为缺省的端口号。

## A.4 使用 SFC 编辑器






SFC 语言用来描述顺序过程的操作。它用一种简单的图形表示法描述过程中的操作步，以及允许在活动步间转换的条件。SFC 程序可用 ISaGRAF SFC 图形编辑器输入。SFC 是 IEC1131-3 标准的核心，其它语言通常用来描述步内的操作和进行转换的逻辑条件。ISaGRAF SFC 图形编辑器允许用户输入完全符合 SFC 标准的程序，它组合了图形和文本的编辑功能，从而既能编辑 SFC 表图，也能编辑相应的操作和条件

SFC 语言主题

### A.4.1 SFC 语言主题

SFC 语言用来表示顺序过程。它把一个过程循环分解成若干明晰的连续的步（自我包含状态），步之间以转换分隔。详情请参考 ISaGRAF 语言参考指南有关 SFC 语言的部分。

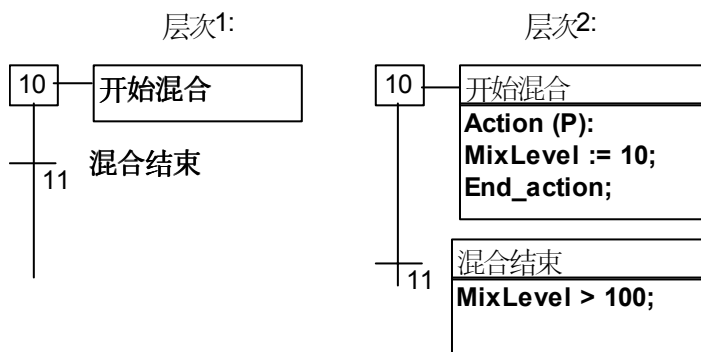
SFC 的部件由有向连接线相连。连接线的缺省方向是由上到下。这些是构成 SFC 图的基本元素：

-  ..... 初始步
-  ..... 步
-  ..... 转换
-  ..... 跳转到步
-  ..... 宏步

□ .....宏初始步

□ .....宏结束步

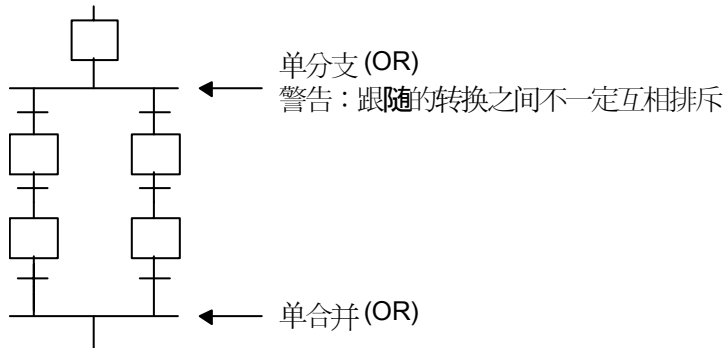
SFC 编程通常可以分为两个层次：层次1显示图形标、步和转换的编号，以及步和转换的有关注释；层次2的程序是用ST或IL编程的步内的操作，或转换的条件。操作或条件也可以是指以其它语言（FBD，LD，ST或IL）编程的子程序。以下是层次1和层次2的编程实例：



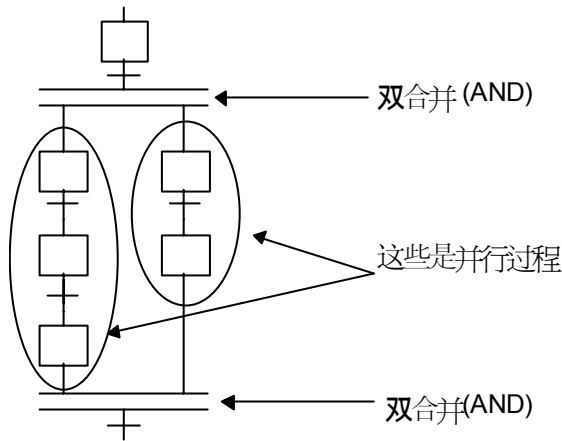
步内的层次2程序是在文本编辑器中输入，它包括用ST或IL编写的操作块。转换的层次2编程则可以使用IL或ST语言，或快捷梯形图编辑器。

## 分支与合并

分支与合并用来表示步与转换间的多重连接。简单的分支与合并表示序列的流向，在过程各子过程间具有不同的可能性。

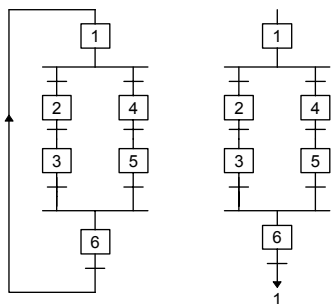


双分支代表了并行过程。



跳转到步

SFC 编辑器只允许用户沿从上到下的方向画连接线。可以用跳转到步的指令表示到图表上部的连接。下列图表是等价的：

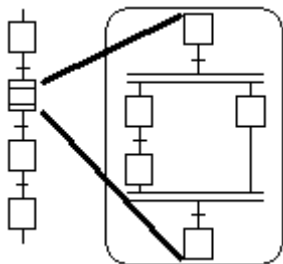


不允许跳转到转换，并且必须明确地表示成一个双（AND）合并。



### 宏步


宏步是将一组步和转换，表示为一个独立步。宏步以初始步开始，以结束步终结。



宏步程序表示法的细节必须与 SFC 程序相同。宏步的符号必须与宏的初始步具有相同的编号。一个宏的描述中可以包括另一个宏步。

## A.4.2 输入 SFC 图



画 SFC 图时，用户只须输入程序的主要构件，所有把元素（水平或垂直地）连接起来的单连接线将由 SFC 编辑器自动画出。用户需要决定所需的位置并从编辑器工具栏上选择 SFC 构件，符号就在当前位置插入。也可使用下列的键盘操作：

F2:  插入一个初始步



F3:  插入一个单步

F4:  插入一个转换

F5:  插入一个跳转



F6:  F7:  插入一个或 (OR) 分支或合并/增加

分支

+F6:  +F7:  插入一个与 (AND) 分支或合并/增加

分支

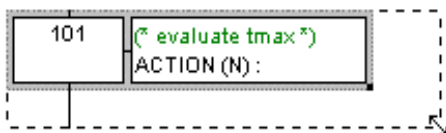
F8:  插入一个宏步

F9:  +F9:  为宏步主体入初始或结束步

(符号“”表示要与 SHIFT 一起使用的组合键)

编辑网格显示矩阵单元。编辑器的选项允许用户在输入程序图表时显示或隐藏网格。网格对初始化程序布局或选择部分表图非常有用。可使用“选项/布局”中的命令显示或隐藏网格。

ISaGRAF SFC 编辑器总显示矩阵中的当前位置，并把当前单元标记为灰色。可自由拖动单元框右下角的小方块以改变其大小，也可以改变宽/高 (X/Y) 比。



## 建立分支和合并

分支和合并总是从左至右画出。分支和合并的左侧支路必须放在图表区内。它们的类型（单线或双线）可用工具栏上的相应按钮选择。

## 建立分支和合并

使用工具栏上的这些按钮，放置附加支路的开始和结束点。分支和合并的左角必须在插入新的支路前存在。分支和合并的右角与主路的左角具有相同的类型。如果没有主路左角存在，右角不能放置。

## 插入宏步

此按钮在主表图中插入一个宏步。宏步的主体内容必须存在于同一SFC程序中的某处。

## 宏步主体

宏步必须用与SFC程序相同的语言表达。宏步必须以初始步开始，以结束步终止。以子程序形式描述的宏操作必须能独立存在。宏的初始步必须与主程序中的宏步符号相同。

### A.4.3 处理已存在的 SFC 图

您可以使用鼠标或键盘在表图中选定矩形区域，整个选定的区域标为灰色。使用“编辑”菜单的命令可以对选定区操作：

#### **剪切/拷贝/删除/粘贴命令**

当选择了编辑器工具栏上的“箭头”按钮后，就可以使用“编辑”菜单中的下列命令了：

剪切.....移动矩形选择区内容到SFC剪贴簿

拷贝.....拷贝矩形选择区内容到SFC剪贴簿

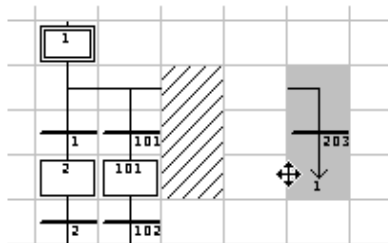
删除.....清除（删除）矩形选择区

粘贴.....将SFC剪贴簿内容插入到当前位置

“编辑/粘贴”命令向屏幕拷贝SFC剪贴簿的内容。拷贝/粘贴命令在SFC表图和层次2的步/转换编程中都能使用，也可以拷贝一个程序中的表图并粘贴到另一个程序中。总是在当前选择的位置之前插入。

#### **移动元素**

在SFC表图中选定元素后，就可以用鼠标拖动选择集并移到另一个位置。当移动选择集时，所选元素的初始位置用阴影显示出来。



移动元素的目标区必须为空。移动 SFC 符号时不允许插入。

### 重编号步和转换

在 SFC 程序表图中用一个逻辑编号识别每个步或转换。“编辑/重编号”命令允许用户让编辑器为当前编辑的 SFC 程序中的任一个步或转换自动设置一个符合数字顺序的引用编号。当改变一个步的编号时，所有跳转到这一步的编号也会自动更新。（这一命令也可应用于宏步和初始步）。

### 直接到达步或转换

“编辑/转到”命令允许用户直接到达一个已存在的步或转换，这时屏幕会自动卷动以显示选择的步或转换。

### 查找和替换文本

“编辑/查找替换”命令可以用来查找和替换整个程序（所有的步和转换）中的文本字符串。查找/替换命令会打开一个用以输入所查找文本的对话框，如果在层

次 2 程序 中 找 到 了 匹 配 的 文 本 ， 会 直 接 打 开 层 次 2 编 程 区。

#### A.4.4 输 入 层 次 2 程 序

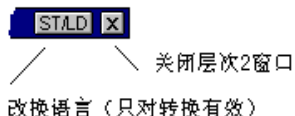
用 户 须 用 鼠 标 双 击 步 或 转 换 的 符 号 ， 以 输 入 层 次 2 的 程 序 内 容 。 层 次 2 编 程 区 显 示 在 SFC 窗 口 的 右 侧 ， SFC 程 序 和 层 次 2 编 程 区 的 分 界 线 可 以 随 意 移 动 调 整 。

您 可 以 同 时 打 开 一 或 两 个 层 次 2 编 程 区 。 可 使 用 键 盘 、 鼠 标 或 “ 编 辑 ” 菜 单 选 择 下 列 命 令 ：

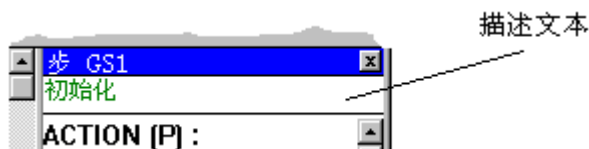
键 盘	鼠 标	“ 编 辑 ” 菜 单
在 最 后 缺 省 窗 口 打 开	Enter	双 击
	编 辑 层 次 2	
在 新 窗 口 中 打 开	Ctrl+Enter	Ctrl + 双 击
	在 新 窗 口 中 编 辑 层 次 2	

当 两 个 层 次 2 窗 口 同 时 可 见 时 ， 它 们 间 的 分 界 线 可 以 自 由 移 动 。 可 以 用 层 次 2 窗 口 标 题 栏 右 侧 的 按 钮 关 闭 窗 口 。

层 次 2 编 程 的 缺 省 语 言 是 ST ( 结 构 文 本 ) 。 对 于 转 换 ， 也 可 以 用 Quick LD ( 快 捷 梯 形 图 ) 编 辑 器 输 入 程 序 。 使 用 层 次 2 窗 口 标 题 栏 上 的 “ ST/LD ” 按 钮 可 以 选 择 当 前 有 效 的 语 言 ， 这 个 命 令 只 在 层 次 2 窗 口 为 空 时 有 效 。



一个单线的编辑框将在层次2窗口顶部出现，它用来输入简短的描述文本。这些文本将在SFC程序中显示为符合IEC标准的注释。这对如“转到...”等其它命令，以及SFC程序的打印输出文件中查找步和转换十分有用。



在层次2窗口打开时可用“选项/刷新”命令，修改后的层次2程序即时更新SFC主程序。



### 插入变量名

使用文本语言编程时，按这个按钮选择一个在项目字典中声明的变量，并将其名称插入在当前的插入标记处。当以Quick LD编程时，按此按钮选择将与所选的接点或块I/O参数联系的变量。



### 在步中插入一个脉冲操作块

当为步进行层次2编程时，按此按钮在当前插入标记处插入一个脉冲操作块的模板。以下是脉冲操作块的格式：

```

Action (P) :
    ST statement;
    ...
End_Action;
    
```

脉冲操作是在步激活时只执行一次的指令。详情参见 ISaGRAF 语言参考。

### **N 在步中插入一个非存储操作**

进行步的层次 2 编程时，按此按钮，在当前插入标记位置插入一个非存储操作块的模板。以下是非存储块的格式：

```

Action (N) :
    ST statement;
    ...
End_Action;
    
```

非存储操作指令在步活动时，每个 PLC 循环周期都要执行。参见 ISaGRAF 语言参考的详解。

### **P0 P1 新的 P0 和 P1 操作限定符**

ISaGRAF 支持新的 P0 和 P1 操作限定符。当输入层次 2 程序时，按此按钮可在当前插入标记处插入 P0 和 P1 操作块。块的格式为：

<pre> Action (P0) :     ST statement;         </pre>	<pre> Action (P1) :     ST statement;         </pre>
--	--

End Action:                      End Action:

P1 操作是当步激活时只执行一次的指令（与脉冲相同）；P0 操作是当步变为非激活时执行一次的指令。详解参见 ISaGRAF 语言参考。

## 二、布尔操作

为在步中对布尔变量进行操作，提供了其它的文本程序语句。这些操作包括把步活动信号连接到一个内部或输出的布尔变量。这是布尔操作的基本语法：

<布尔变量> (N);	将步活动信号赋值给变量
<布尔变量>;	相同的作用 (N属性可选)
/<布尔变量>;	将反相 (非) 的步活动信号赋值给变量

其它命令在步激活时，置位或复位布尔变量。这是置位和复位布尔操作的语法：

<布 尔 变 量> (S);	当步活动信号变为真时置位变 量到真状态
<布 尔 变 量> (R);	当步活动信号变为真时复位变 量到假状态

## 二 SFC 操作

为控制 SFC 子程序的操作提供了其它文本程序语句。  
SFC 操作是一个根据步活动信号的状态而开始或结束

的 SFC 子序列。SFC 操作可具有 N（非存储），S（置位），或 R（复位）限定符。下面是 SFC 操作的基本语法：

<子程序> (N);	步变激活时开始子序列，步变为非激活时结束子序列。
<子程序>;	语法同上（N 属性可选）
<子程序> (S);	步变激活时开始子序列，但步变为非激活时不进行任何操作。
<子程序> (R);	当步变激活时结束子序列，但变为非激活时不进行任何操作。

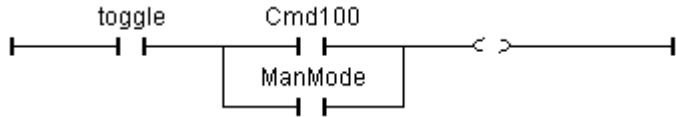
SFC 序列指当前编辑的程序的子 SFC 程序进行的操作，它由 ISaGRAF 程序管理器建立。

## **ST 中的转换**

转换的层次 2 程序是一个布尔表达式。用 ST 语言编程只须根据 ST 语法输入转换的布尔条件。在表达式的结尾处加分号也是可选的。

## **快捷 LD 中的转换**

为编写转换的层次 2 程序提供了快捷 LD 编辑器。在这种情况下，图表仅由一个支路构成，只有一个代表转换的输出线圈。线圈符号不必重复转换的名称。下面是一个用快捷 LD 编程的转换条件示例：



使用快捷LD编程时，可用键盘上的箭头键在编程逻辑网格上移动选择，然后以下列快捷键插入符号：

F2: .....在所选符号后插入触点 / 初始化支路 F3:

在所选符号前插入触点

F4: ..... 插入与所选符号并联的触点

F6: ..... 在所选符号后插入块

F7: ..... 在所选符号前插入块

F8: ..... 插入与所选符号并联的块

您也可以用鼠标单击层次2窗口底部的功能键以代替按键盘上的功能键

选择了触点或块I/O参数后，可以按回车（RETURN）键选择变量或输入常量值。选择了功能块后按RETURN键可以选择功能块的类型。您也可以用鼠标双击一个符号取得相同的结果。

选择了触点后可以按空格（SPACE）键选择触点的类型（直接，反相或脉冲检测）。参见本帮助“使用快捷LD编辑器”一节有关快捷LD功能的详细介绍。

#### A.4.5 使用SFC图库

ISaGRAF SFC编辑器管理着一个SFC图库，它是能被插入SFC表图中的SFC构件的集合。可以将SFC图库的元

素有选择地嵌入到步和转换的层次2程序中。可使用“工具”菜单的下列命令：

拷贝到SFC图库	把选择的元素拷贝到SFC图库
自SFC图库粘贴	在当前位置粘贴SFC图库元素

当向SFC图库拷贝（就是建立新的SFC图库元素）时，您可以选择是否将SFC程序的层次2程序嵌入。

## A.5 使用流程图编辑器

ISaGRAF 流程图的图形编辑器允许用户进入整个 FC (流程图) 程序, 这些程序是用 SL, IL 或快捷 LD 语言编写的动作和测试 (判断)。流程图是一种判断图, 由于它有一些先进的性能, 如无阻碍向后跳转, 所以也可以用它来描述顺序动作。

### A.5.1 FC 语言的基本语句


流程图 (FC) 是用于描述顺序操作的一种图形语言。流程图由动作和测试组成。在动作和测试之间是代表数据流的定向链接。下面是流程图语言的图形组件:





FC 流程图的开始: “开始”符必须出现在一个流程图程序的开头。它是唯一的并且不能省略。当它被激活时, 它表示流程图的初始状态。




FC 流程图的结束: “结束”符必须出现在一个流程图程序的结尾。它是唯一的, 并且不能省略。有时可能没有给“结束”符画连线 (永远循环的流程图), 但不管怎样, 在流程图的底部仍要画出“结束”符。当程序的执行被完成时, 它表示流程图的最后状态。

 FC 流程的链接：一个流程的链接是表示图形两点之间信息流的一条线。一个链接总是以一个箭头结束。两个链接不能与同一个源连接点相连。

 FC 动作：一个动作符号代表将被完成的动作。一个动作用一个号码和一个名称来识别。同一个流程图的两个不同目标不能使用同一名称或逻辑号。一个动作的编程语言可以是 ST, LD 或 IL。一个动作总是与链接相连，其中一个链接表示到达动作，另一个链接表示从动作开始。

 FC 测试：一个测试表示一个布尔量的条件。一个测试用一个号码和一个名称来识别。根据对所附带的 ST, LD 或 IL 表达式所作的判定，程序的流程被指向“YES”或“NO”路径。当用 ST 文本编程时，可供选择的一种方式是在表达式后跟一个分号。用 LD 编程时，那个唯一的线圈表示条件值。

 FC 子程序：本系统能够对 FC 程序的垂直结构进行描述。FC 程序是一个有层次的树形结构。每一个 FC 程序可以调用另一个 FC 程序。这样的程序被称为 FC 程序的子程序，子程序由 FC 程序来调用。调用 FC 子程序的 FC 程序被称为父程序。用一种“父-子”关系把 FC 程序链接在一起，形成一个有层次的树状主干结构。流程图中的一个子程序符号表示对一个流程图子程序的调用。正在调用子程序的 FC 程序被暂停执行，直到子程序执行完毕。



FC 语言的 I/O 特定动作：一个输入/输出特定动作符号表示所要完成的动作。与其它动作一样，一个输入/输出特定动作也是用一个号码和一个名称来识别。在标准动作和输入/输出特定动作上使用相同的语义。输入/输出特定动作的目的仅仅是使流程图更加易读并集中在流程图的不可移动部分。使用输入/输出特定动作是一个供用户选择的功能。输入/输出特殊块与标准动作具有完全相同的特点。



FC 连接符：连接符用于表示未画出的图形两点之间的一个链接。一个连接符用一个圆来表示并被连接到流程的源点。连接符图形是在适当的边上（取决于数据流的方向）通过对目标点（一般是目标符的名称）的识别来完成的。一个连接符总是把一个定义的流程图符号作为目标。目标符用它的逻辑编号来识别。



FC 注释：一个注释框包含有文本，该文本没有流程图语义上的意义。在流程图文件窗口上可以把文本插入到未使用空间的任何地方，并用于给程序提供资料。

### A.5.2 输入流程图

为了进入流程图，你必须把元素（动作，判断检测，连接符）放置到图形区中，并在元素之间画上流向链接线。



## 插入目标

为了在图形中插入一个目标，选择工具栏中相应的按钮，并在图形区中你想要插入目标的位置上单击。你可以在一个空位上放置某个元素，或者在一个流程链接上单击，把元素插入到一个流程。在一个链接上所作的插入只允许用于从上到下的垂直链接。

你可以插入下列基本元素：



用 ST, IL 或快捷 LD 编写的动作



IO 的特殊动作 (高亮显示一个特殊的不可移动的动作)



用 ST, IL 或快捷 LD 编写的测试(判断)



连接符



调用一个 FC 子程序



注释(描述文本)

ISaGRAF 流程图编辑器也向你推荐流程图的分类结构表。这种结构只能插在已有的流程链接上。不能把它们放在一个空位上：



如果 / 那么 / 否则 (双路选择)



重复直到(等待一个条件)



循环(在一个条件为真时循环)



## 选择目标

对大多数编辑命令来说，需要选择图形目标。ISaGRAF FC 图形编辑器能选择图形区内已有的一个或多个目

标。为了选择目标，在编辑器工具栏中必须检查“选择”项(带箭头的按钮)。要想选择一个目标，用户只须在它的符号上单击一下。

要想选择一个目标表，在图中拖动鼠标画一个矩形。该矩形中的所有图形目标被打上“已选择”标记。

一个被选择的目标用深蓝色勾画，在它的图形符号周围有一些小的黑色方块。按住 Shift 或 Ctrl 键并在图形符号上单击，也可以把一个目标添加到一个多重选择中或把它删除。

在做了一项新的选择之后，在此之前所选择的所有目标被删除。要想删除已有的选择，只需用鼠标在一个空位上，即在限定被选目标边界的矩形框外单击。

对于单项选择来说，可以用键盘上的方向键在流程图从一个目标到另一个目标进行选择。也可以选择流程链接。



### **插入注释**

可以把注释插入到图中任何空位上。注释对执行程序没有影响。它们使程序有更高的易读性。想要插入一个注释块，在工具栏中选择相应的按钮，并且在图中必须放置注释的地方单击。想要输入/改变一个注释文本，则双击该注释。当输入一个注释块的文本时，不需要象“(”和“\*)”这样的特殊引导符和结束符。选择一个注释块后，通过拖动它的边框角可以改变其大小。



### 勾画流程链接

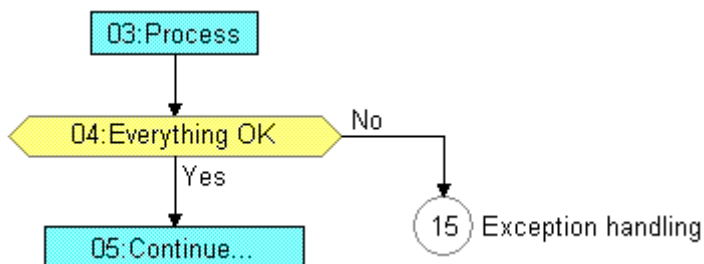
在工具栏中选择该按钮，在已有的元素之间画上流程链接。一个链接必须总是按流向来勾画。想要插入链接，首先选择一个FC元素中的未连接的输出点，并拖动鼠标到某个终点。该终点或者是一个未连接的FC元素的顶端(输入点)，或者是一个已有链接上的任何位置。在流程图中，用灰色的小圆圈表示链接之间的会聚点。为了编排流程图也可以选择 and 移动会聚点。



### 使用连接符

ISaGRAF 流程图编辑器能使用图形连接符，用于代替一个可见的流程链接。连接符对于避免远距离的链接和提高流程图的易读性是非常有用的。一个连接符不能用于与其它的FC程序建立链接。

与其它FC目标一样，一个连接符被放在流程图中。它用一个圆圈表示，圆圈内含有目标元素(流程链接的终点)的参考编号。目标元素的简要描述文本显示在连接符圆圈的旁边。





### **移动目标**

要想移动流程图中的目标，你必须选择目标，并在流程图内拖动鼠标移动它们。你可以移动单个元素或一个多重选择的元素组。当移动它们时，这些元素不能重叠。不能把正在移动的元素与一个已有的链接相连。

当移动单个元素(动作, 检测...)时，ISaGRAF 流程图编辑器自动地把被选择的元素与放置在它下面的并与它相连接的所有目标一起移动。这一功能不能用于多重选择的情况。



### **改变目标大小**

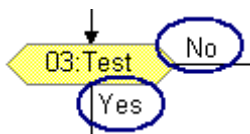
除了“Begin”，“End”符号和连接符以外，流程图的任一图形元素可以自由地改变其大小。要想改变一个元素的大小，你首先必须选择它，然后用鼠标拖动元素边框上画的小方块，以便改变元素尺寸。

当一个元素与一个流程图的链接相连时，在元素的左边框和右边框上水平地改变它的大小，以便当元素的大小被改变时，其元素仍能正确地被连接在链接上。



### **交换一个检测的输出**

你可以在一个检测(判断)上交换YES/NO输出的位置。做这项操作时，只需双击显示在测试符旁边的“ Yes”或“ No”符号即可。



### A.5.3 在已有的流程图上操作

“编辑”菜单上的命令用于改变或完成已有的图。大部分命令作用于图中当前被选择的元素。

#### 修正流程图

DEL 键可用于删除被选择的元素。未确定的链接与所选定的元素一起被删除。在执行 DEL 命令之后用“编辑/复原”命令使元素复原。DEL 命令也可用于在图中选择的一组元素。“编辑”菜单的“剪切”，“拷贝”，“粘贴”命令用于移动或拷贝被选择的元素。

#### 查找和替换

“编辑/查找替换”可用于查找或替换整个程序(用 ST, IL 或快捷 LD 编写的所有动作和测试)的文本字符串。查找/替换对话框用于输入要检索的文本，并直接打开编辑部分，在此查找文本。



### 直接存取一个元素

“编辑到”命令允许用户存取流程图中已有的图形元素。滚动位置被自动连接，以便显示元素。当到达滚动位置时元素即被选中



### 给元素重新编号

“编辑/重新编号”命令用于对流程图的元素重新编号。放在流程图中的任何一个FC元素用一个唯一的参考号来识别。每当插入新元素时，由编辑器配置参考号。“重新编号”命令允许你根据元素在流程图中的位置重新修正元素的编号。编号的增大是按照从顶部到底部和从左到右的顺序进行的。

## A.5.4 输入第二层程序

用户要想输入第二层程序，必须在动作或测试符号上双击。第二层编程显示在FC窗口的右边。你可以自由地移动FC流程图和第二层编程之间的分隔行。你可以同时打开一个或两个第二层区域。你可以从键盘、鼠标或“编辑”菜单上选用下面的命令：

键盘    鼠标    “编辑”菜单

在上次的缺省窗口中打开 Enter    双击

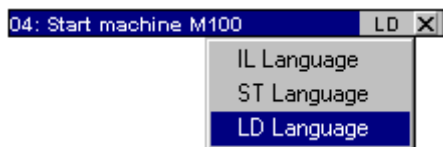
编辑第二层

在单个窗口中打开 Ctrl+Enter    Ctrl + 双击

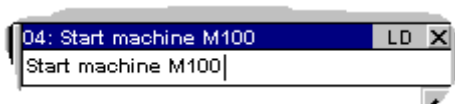
在单个窗口中编辑第二层

当出现两个第二层窗口时，可以自由地移动它们之间的间距。用第二层标题栏右边的按钮关闭第二层窗口。

第二层编程的缺省语言是ST(结构化文本)。编程语言也可以是IL或快捷LD。所选择的语言名称显示在第二层标题栏内的小框里。要想改变激活的语言，从菜单上运行“选项/设置第二层语言”命令或在那个框上单击一下。只有当第二层编程窗为空白时，该命令才生效。



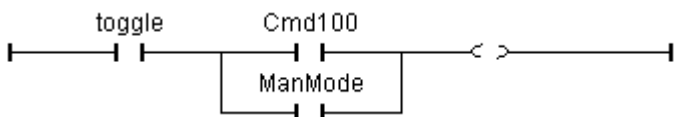
这里有一个单线编辑框出现在第二层窗口的上方。它用于插入一个简短的描述文本。该文本将作为IEC的注释显示在FC符号图中。由于该文本被其它一些命令(如“进入..”)所用，并且在FC打印输出中提供有关FC动作和测试的资料，所以它是非常有用的。



当第二层窗口打开时，任何时候都可以用“选项/刷新”命令去刷新带有被修改过的第二层程序的FC主流程图。

### A.5.5 用快捷LD编程

快捷LD编辑器可用于第二层编程。在作一项判断测试时，LD图仅由一个梯级组成，且只带一个表示判断的线圈。测试名称不会与线圈符号重复。下面是一个用快捷LD编程的测试例子。



当用快捷LD编程时，使用键盘上的方向键在编程的逻辑网格内移动选项，然后用下列快捷键插入符号：

- F2:.....在被选择的符号后面插入一个触点/对梯级进行初始化
- F3:.....在被选择的符号前面插入一个触点
- F4:.....与被选择的符号并联插入一个触点
- F5:.....与被选择的线圈并联添加一个线圈(不是用于测试)
- F6:.....在被选择的符号后面插入一个块
- F7:.....在被选择的符号前面插入一个块
- F8:.....在被选择的符号并联插入一个块
- F9:.....与被选择的线圈并联添加一个跳转符号(不是用于测试)

一个跳转产生一个梯级名称。当选择了梯级标题时，敲ENTER键可输入一个梯级名称。ISaGRAF编辑器对于你已经输入的梯级标记保持记忆，不管它已被规定用作一个梯级名还是用作一个跳转操作。“跳转/标记”

对话框给你提供了这样的可能性：输入一个新标记或选择一个已有的标记。如果你输入一个新的名称，该名称将被自动添加到列表中。“删除”按钮用于从列表中删除被选择的名称，但它不能删除你所选择的梯级图上的标记。操作方法是，当编辑框为空白时按下确认键即可。

你也可以按LD工具栏中的按钮，而不敲功能键。

当选择了一个触点或一个块输入/输出参数时，敲回车键可选择一个变量或输入一个常量。当选择了功能块时，敲回车键可选择功能块的类型。为达到相同的效果，你也可以在一个符号上双击。

当选择了一个触点时，敲击Control + SPACE键，可改变触点或线圈的类型（直接的，求反的）。关于快捷LD功能的详细信息请参阅本文件中“使用快捷LD编辑器”一章。

### A.5.6 显示选项

用“选项/布局”命令打开一个对话框，所有与编辑器工作区和绘图有关的参数和选项在该对话框中集合。用“工作区”集合框中的一些检查框来显示或隐藏编辑器工具栏和状态栏。“文件”集合框中的选项允许你显示或隐藏编辑用的网格线上的点及显示黑白或彩色流程图。



用工具栏中的“显示比例”按钮来改变当前的显示比例。当操作一个附加在一个动作或一个测试上的快捷LD程序时也可使用该命令。



用工具栏中的“网格线”按钮来显示或隐藏用于编辑的网格线上的点。当操作一个附加在一个动作或一个测试上的快捷LD程序时也可使用该命令。

用“选项/字体”命令来选择在所有ISaGRAF文件中使用的字符的字体名称。当从一个ST或IL块中调用字体时，你可以确定字体的大小。当为某个图形选择字体时(FC或快捷LD)，字体的风格和大小是不相关的，因此不必对它们进行设定。ISaGRAF的图形编辑器总是根据当前的放大比例来计算字体的大小。

## A.6 使用快捷 LD 编辑器

LD 语言允许用图形来表示布尔表达式。布尔逻辑的与、或、非 (AND, OR, NOT) 运算符可用图形拓扑明了地表示。输入布尔变量与图形触点相联系；输出布尔变量与图形线圈相联系。使用键盘或鼠标都可以很容易地在 ISaGRAF 快捷 LD 编辑器中输入 LD 梯形图。快捷 LD 编辑器将自动连接图形元素和排列 LD 支路，而不需要用户手工画连接线。快捷 LD 编辑器也会自动排列支路，使图形最优化。

### A.6.1 LD 语言基础





LD 程序是一个在其中排列了触点和线圈的支路的列表。下面是 LD 图的基本元素：

#### 支路头部 (左电源线)

每个支路从左侧的电源线开始，它代表了初始的“真”状态。当用户安放了支路的第一个触点后，ISaGRAF 快捷 LD 编辑器会自动画出左侧的电源线。每个支路可以拥有一个逻辑名称，用作跳转指令的寻址标号。







## 触点

触点根据布尔变量的状态改变布尔数据流。变量的名称显示在触点符号的上方。ISaGRAF 快捷 LD 编辑器支持下列类型的触点：

-  ..... 触点
-  ..... 反相触点
-  ..... 上升沿检测触点
-  ..... 下降沿检测触点

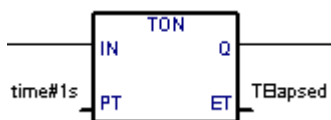
## 线圈

线圈代表了一个操作。用支路的状态（线圈左侧连接的状态）来强制布尔变量。变量的名称显示在线圈符号的上方。ISaGRAF 快捷 LD 编辑器支持下列类型的线圈：

-  ..... 线圈
-  ..... 反相线圈
-  ..... “置位”线圈
-  ..... “复位”线圈
-  ..... 上升沿检测线圈
-  ..... 下降沿检测线圈

## 功能块

LD 图中的一个块可以表示一个功能、功能块、子程序或运算符。它的第一个输入和输出参数总是与支路连接。其它输入和输出参数按逐个排列在块的矩形框外。



## **"内线" 功能块**

在快速 LD 编辑器中, 您可以通过选择功能改变已编辑好的功能块 到"内线"功能块, 然后在工具指令上选择 « 内线 ». 这个指令使得 您可以设置或调置编辑 FB 相关的 « 内线 ». 当 « 内线 »处于设置中, 一小棒将显示 « 内线 »文本

## **—| 支路尾部 ( 右电源线 )**

支路以右电源线结束。使用快捷 LD 编辑器时 , 右电源线是在用户插入线圈之后自动画出的。

## **→L1 跳转符号**

跳转符号总是指向一个支路标号 , 即 LD 图中某处定义的支路名称。跳转符号置于支路的尾部 , 当支路的状态为真时 , 程序的执行就直接跳转到目标支路继续进行。注意向后跳转是危险的 , 有可能导致 PLC 程序循环的阻塞。

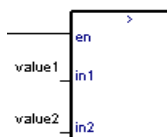
## **return 返回符号**

返回符号置于支路的尾部 , 它指出当支路的状态为真时 , 程序的执行必须终止。



## “EN” 输入

对有些运算符、功能或功能块来说，第一个输入不是布尔类型的数据。由于第一个输入必须总和支路连接，所以就自动为第一位置添加了一个输入，称为“EN”输入。只有EN输入为真时，模块才被执行。以下是一个比较运算符的例子，以及用ST语言表示的等价代码：

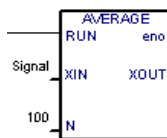


```
IF rung_state THEN
    q := (value1 > value 2);
ELSE
    q := FALSE;
END_IF;
(* continue rung with q state *)
```



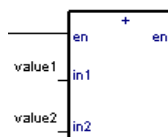
## “ENO” 输出

对有些运算符、功能或功能块来说，第一个输出不是布尔类型的数据。由于第一个输出必须总和支路连接，所以就自动为第一位置添加了一个输出，称为“ENO”输出。ENO输出总是复制与模块第一个输入相同的逻辑状态。以下是一个求平均值的例子，以及用ST表示的等价代码：



```
AVERAGE(rung_state, Signal, 100);
OutSignal := AVERAGE.XOUT;
eno := rung_state;
(* continue rung with eno state *)
```

有些情况下，同时需要EN和ENO。下面是一个算术运算的例子，以及用ST语言表示的等价的代码：



```
IF rung_state THEN
    result := (value1 + value2);
END_IF;
eno := rung_state;
(* continue rung with eno state *)
```

## HHH 快捷LD编辑器的限制

ISaGRAF 快捷 LD 编辑器不支持在线圈的右侧延伸一个支路（插入触点或线圈）。如果一个支路必须有几个输出线圈，则相应的线圈必须画成并联。

### A.6.2 输入 LD 图

所有快捷 LD 编辑器的编辑命令都可以用键盘或鼠标完成。



#### 编辑网格

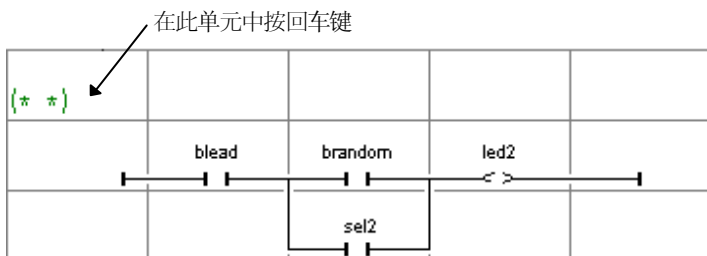
LD 图形是在一个逻辑矩阵中输入的，矩阵的每个单元只能容纳一个 LD 图形符号。使用键盘的箭头键或用鼠标在单元中单击可以移动当前选择。选择到的单元将反相显示。对于一些剪切 / 拷贝 / 粘贴命令来说，只须用鼠标在图中拖动，或在按箭头键时按住 SHIFT 键，就可以同时选择几个单元。

## 开始一个新支路

只要在现存的最后支路之后插入一个触点（按 F2 键或 LD 编辑器工具栏上的相应按钮），就可以在图中添加一个支路。这个支路拥有一个触点和一个线圈。

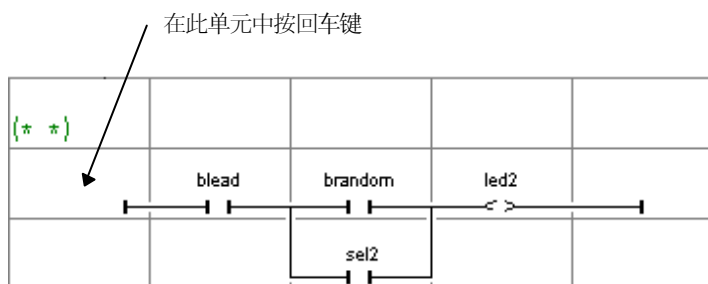
## 输入支路注释

每个支路最多可以有两行注释文本。为输入支路注释，可选择支路上方的单元并按回车键，或在此单元中双击鼠标：



## 输入支路标号

每个支路都可由名称识别，这个名称可用作跳转操作的目的标号。要输入或改变支路标号，只须选择支路头部，然后按回车（ENTER）键，或用鼠标在此单元中双击：



ISaGRAF 快捷 LD 编辑器将保存您输入的标号，而不论它是指定为支路名称或用于跳转操作。使用“ 跳转 / 标号”对话框可以输入或选择已存在的标号。

如果输入了一个新名称，它将被自动添加到列表中。“ 清除”按钮用来从列表中删除所选名称，但它不删除图中所选的支路名称。为完成后者操作，只须当编辑框为空时，按“ 确定”按钮。

## 在支路中插入符号

在已存在的支路内插入符号（触点，线圈，块 ... ..）的操作总是根据当前选择的位置进行。您必须在支路中选取一个有效单元，然后按如下功能键插入符号：

- F2..... 在所选符号之前（左侧）插入触点
- F3..... 在所选符号之后（右侧）插入触点
- F4..... 插入与所选符号并联的触点
- F6..... 在所选符号之前（左侧）插入块
- F7..... 在所选符号之后（右侧）插入块

F8..... 插入与所选符号并联的块

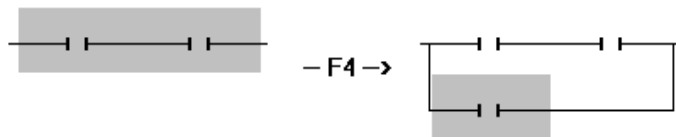
当选择了支路输出（线圈）时，下列命令有效：

F5..... 添加一个与所选线圈并联的线圈

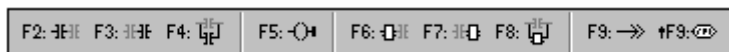
F9..... 为所选线圈添加一个“跳转”符号

Shift +F9..... 为所选线圈添加一个“返回”符号

对于并联插入（F4 / F8）的情况，如果选择了同一支路的几个触点，则新插入的符号与选择的触点成组并联。以下是一个例子：



您也可以用“插入”菜单中的命令插入图形符号。也可以用鼠标单击LD编辑器工具栏上的相应按钮选择要插入的不同类型符号：



## 输入符号

选择触点或线圈并按回车（ENTER）键，或在它们上面双击鼠标，可以为触点和线圈加一个变量符号。弹出一个变量选择对话框。可参照“有关程序编辑器的更多信息”一节有关如何使用这个对话框详细说明。要使功能、功能块或运算符联系到一个块，可以选择

矩形框的内部然后按 ENTER 键。将变量符号关联到输入或输出块参数，必须选择矩形块外的相应位置。

包括变量或块选择列表在内的对话框通常用来进行文本输入。如果在“选项”菜单中选定了“手工键盘输入”模式，则可以在单一文本编辑框内直接输入变量符号和块名。输入新文本后按“Enter”键使其有效；或按“Escape”键退出修改并关闭文本编辑框。“手工键盘输入”模式下的文本编辑框不能使用鼠标关闭。



### **改变触点和线圈类型**

可以用“编辑/改变线圈/触点类型”命令改变所选择的触点或线圈的类型。触点可以是直接、反相、上升沿或下降沿检测型触点；线圈可以是直接、反相、置位或复位、上升沿或下降沿检测型。在选定目标后按空格键具有相同的作用。



### **在图中插入一个支路**

编辑/插入支路”命令在所选择的当前支路前插入一个新的支路。这新支路被初始化为拥有一个触点和一个线圈。

处理已存在的图表

“编辑”菜单中的命令用来改变或完成一个已存在的程序。这些命令中的大部分的都对程序图表中当前选定的元素进行操作。

## A.6.3 处理已存在的图表

### ✎ **更正图表**

DEL 键可用来删除选定的元素。但当支路的唯一输出是一个线圈、跳转或返回符号时，则不能删除它们。使用“编辑/恢复”命令可以恢复被清除的元素。DEL 命令也可应用于图中选定的成组元素。“编辑”菜单中的“剪切”、“拷贝”、“粘贴”命令用来移动和拷贝选定的元素。

### ✎ **拷贝符号**

“编辑”菜单中的“剪切”、“拷贝”、“粘贴”命令用来移动和拷贝选定的元素。这些命令对支路注释不起作用。“编辑/特殊粘贴”命令允许您选择性地粘贴元素

在所选择元素之前（左侧）

在所选择元素之后（右侧）

- 与所选择元素并联

### ✎ **管理整个支路**

当选择了支路的头部（左电源线）后，所有的编辑命令（删除，拷贝，剪切...）都可以对整个支路起作用。这提供了一个只要移动所选择的图中第一列，就可以排列图中整个支路的便捷方法。也可以将

选择集垂直延伸到几个支路头，这种情况下可以对几个完整支路应用编辑命令。

## **查找和替换**

“编辑/查找”和“编辑/替换”命令用来查找和替换程序图中的文本。只有完整的名称可以被找到。查找可作用于触点、线圈、块名、块参数和支路标号。它不能用来查找注释中的字符串。替换命令不能用来改变块的类型。查找/替换可以从当前选择的位置向前或向后进行，在遇到程序边界时，会“循环”继续进行。为快捷查找变量提供了如下快捷方式：

ALT + F2 查找与当前选择的元素具有相同变量名的下一个元素。这一命令也可以应用于功能块和支路标号。

ALT + F5 查找与当前选择的元素具有相同变量名的下一个线圈。这主要在调试模式下使用，以快捷找到输出可疑变量的支路。

## **A.6.4 显示选项**

“选项”菜单中的命令用来定制LD图表在屏幕上的表示方法，以隐藏或显示某一类型的信息。



### 工具提示

使用“选项 / 工具提示”指令来隐藏或显示变化的注释, 这个注释是作为工具提示出现在整个图中。当游标移过变量块的, 注释也将作为工具提示出现。选项在在线和离线状态下均可。



### 支路注释

使用“选项 / 支路注释”命令在整个图表中隐藏或显示支路的注释。由于每个注释占用编辑矩阵的一行, 当查看一个大程序时可能需要隐藏支路注释以获得紧凑的显示。这一选项不影响已存在的注释, 并且可以随时在显示 / 隐藏支路注释间切换。



### 名称和别名

所有与触点、线圈或块 I/O 参数相联系的变量都由其符号名称识别。ISaGRAF 快捷 LD 编辑器为每个变量引入了“别名”的概念。变量的别名就是只取“:”符号之前, 并且限制在 16 个字符以内的变量注释文本。以下是一些例子:

**变量注释:**

short text

long text with no separator

short text: long description

**别名:**

short text

long text with n

short text

别名对 LD 程序的执行没有影响, 并且从语法的角度它应该被看作注释。当从变量列表选取变量时, 变量的别名就从变量注释中被自动地抽取出来。它不能手

工修改。可以使用“选项/触点和线圈”命令选择变量标识的几种显示模式如下：

- 只显示变量名
- 只显示变量别名
- 显示变量名和别名

当在变量字典中改变了变量的别名后，快捷LD编辑器并不自动更新LD文档中的变量别名。需要使用“选项/触点和线圈 / 更新别名”命令来更新当前编辑的程序中所有的变量别名。您也可以在“选项/触点和线圈”中选择“打开时总是更新”选项，使ISaGRAF在每次打开快捷LD程序时自动更新所有用到的变量别名。警告：设置这个选项会显著增加打开程序耗用的时间。

## 画法选项

“选项/布局”命令将打开一个对话框，其中包括有关编辑器工作区和LD梯形图程序图形画法的所有参数和选项。

使用“工作区”中的选择框来显示或隐藏编辑器工具栏，状态栏和LD工具栏。“文档”中选项允许您选择显示或隐藏编辑网格，以及允许/禁止在绘制时使用颜色。



“缩放”选项允许选择一个主要的缩放比例。也可以用编辑器工具栏上的“缩放”按钮与缺省缩放比例间切换。



您也可以定制编辑网格单元的长/宽 (X/Y) 方向的比例。如果您通常使用较短的变量名, 就可以使用这一选项改变缺省的单元宽度。也可以使用编辑器工具栏上的“宽度”按钮改变长/宽比例, 而不用打开布局对话框。

“选项/字体”命令用来选择, 所有 ISaGRAF 图形文档中使用的字体。选择字体时, 字体的样式和大小不须指定, ISaGRAF 图形编辑器总是根据所选的缩放比率计算字体的大小。

## A.6.5 在线帮助

为了从 Quick LD 编辑器上获得关于功能块的帮助

- 在 LD 图中选择功能块
- 按 F1.

关于功能块的帮助便会显示出来.在习惯于 "C" 语言或 IEC 功能 或功能块的情况下,显示的帮助是进入到库编辑器中(仅限文本)的 "技术注释".

## A.7 使用 FBD/LD 编辑器

ISaGRAF FBD/LD 图形编辑器允许用户输入全部的 FBD 程序，其中也包括一部分用 LD 语言。它结合了图形和文本编辑能力，所以可以输入程序图表和相应的输入、输出。由于这个编辑器是专供 FBD 语言使用的，输入纯 LD 程序时最好使用 ISaGRAF 快捷 LD 编辑器。

### A.7.1 FBD/LD 语言基础

FBD 语言是许多不同类型的方程式的一种图形表示法。操作符被表示为矩形的功能块。功能的输入被连接到矩形块的左侧；输出连接到块的右侧。图形的输入和输出(变量)通过逻辑连接连接到功能块。一个功能块的输出可以连接到另一个块的输入。

LD 允许用图形表示布尔表达式。布尔逻辑与，或，非运算符可以明确地以图形拓扑表示。布尔输入变量与图形触点联系起来，布尔输出变量连到线圈。触点和线圈被水平短接线连接到左右两侧的电源线上。每条线段都拥有假或真的布尔逻辑状态。所有直接相连的线段具有相同的布尔逻辑状态。所有连接到左侧垂直电源线的线段具有逻辑真的状态。

LD 和 FBD 程序总是从左至右，从上到下解算。有关 LD 和 FBD 语言的细节请参考 ISaGRAF 语言参考指南。以下

是 LD 和 FBD 语言的基本图形元素，并且为 FBD/LD 编辑器所支持：



### 左电源线

左电源线代表了初始的“真”状态，梯级必须与它相连。ISaGRAF FBD 编辑器也允许任何布尔符号连接到左电源线。



### 右电源线

线圈可以连接到右电源线。这是可选的。如果线圈在右侧未连接，则它自身的图中就包含了右电源线。



### LD 垂直“或”连接

LD 垂直线可以在左右两侧同时拥有其它几个连接。每个垂直线右侧的连接相当与左侧连接进行或运算。




### 触点

触点根据布尔变量的状态改变布尔数据流。变量的名称显示在触点符号的上方。ISaGRAF FBD/LD 编辑器支持以下几种触点：

 ..... 直接触点 t





 ..... 反相触点

 ..... 上升沿触点

 ..... 下降沿触点

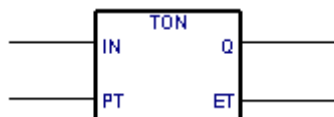
## 线圈

线圈代表了一种操作。它必须与左侧的布尔符号如触点相连。变量的名称显示在线圈符号的上方。ISaGRAF FBD/LD 编辑器支持下列几种类型的线圈：

-  .....直接线圈
-  .....反相线圈
-  .....“置位”操作线圈
-  .....“复位”操作线圈

## 功能块

在FBD图中一个块可以表一个功能、功能块、子程序或运算符。输入和输出端必须连接到变量、触点或线圈、或其它块的输入或输出。形式参数名被标注在矩形内部。



## 标号

标号可以放在程序内的任何地方。标号指明跳转指令的目标，以改变程序执行的顺序。建议将标号置于程序的左侧，以提高程序的可读性。



## 跳转

一个跳转符号总是指向置于程序中某个地方的标号。它的左侧必须连接到一个布尔状态点。当左侧的连接为真时，程序操作就直接跳往目标标号。注意向后的跳转是危险的，在有些情况下会引起PLC程序循环的阻塞



## 返回标号

返回标号要与布尔点连接。它将在支路的状态为真时，中止程序的执行。



## 变量

在程序图表中变量在小的矩形框中表示，它的左侧或右侧连接到图表的其它元素。



## 连接

连接线画在图表程序的各元素之间。连接总是沿着数据流的方向从一个输出到一个输入。



## 带布尔逻辑非的连接

一些布尔连接被表示为在其末端带有一个小圆圈。这代表经过它的信息发生了布尔逻辑反相。



## 用户定义转角

连接可以加上用户定义点。它允许用户人工控制连接的路由。如果没有放置转角，ISaGRAF FBD/LD 编辑器就使用缺省的路由法则。

## A.7.2 输入 FBD 图表

为了输入图表，必须在图形区域中放置各种元素（块、变量、触点、线圈……），然后在它们之间画上连接。



### **插入对象**

选择工具栏上相应的按钮，再用鼠标在要插入的图形区域单击，可以在程序图表中插入对象。



### **选择对象**

大部分的编辑命令都需要选择图形对象。ISaGRAF LD/FBD 图形编辑器允许在图表区中选择一个或多个对象。选择对象先要在编辑器工具栏上按标有箭头的“选择”按钮。选择一个对象时，用户只须用鼠标单击它的符号；选择多个对象时，可用鼠标在图表上拖画出一个矩形区域，凡与矩形相交的图形对象都被标记为“已选择”。已选择的对象显示为外围黑色小方块的图形符号。进行新的选择将取消所有以前的选择。要取消当前的选择，只须用鼠标在选择框外的空白区域单击。



### **插入注释**

图表内的任何位置都可以插入注释。注释不影响程序的执行，只给程序带来更高的可读性。选择工具栏上

的这个按钮，然后用鼠标在要插入注释的区域拖画出一个矩形，再在其中输入注释文本。输入注释区文本时不需要任何如“（\*”和“\*）”的字符做标记。可以通过拖动边框的角来调整注释框的大小。



### 移动对象

先选择程序图表中的对象，然后用鼠标拖动选择区域在图中移动。对于已连接的对象，用户只须简单地拖动图形符号，ISaGRAF LD/FBD 编辑器会根据对象的新位置自动重画它们间的连接。



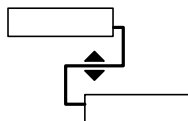
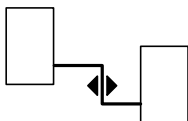
### 画连接

选择工具栏上的这些按钮之一，画已存在的元素间的连接。如果从一个连接点向图中的空区域画连接，它将在连接末端以用户定义转角结束，以便可继续画另一段。



### 改变连接画法

“工具/移动线”用来改变图中所选连接的自动路由。此命令不会影响连接到用户定义角的连接。如果一个连接被画成三段，则此命令将改变其中第二段的位置。举例如下：





## 改变连接类型

您可以在连接的右端双击鼠标，以改变连接的类型（带或不带布尔逻辑非）。



## 画LD支路

为画一个新LD支路，先要插入左电源线，然后放置一个线圈：它将被自动地连接到电源线上。其它触点和垂直或连接可以直接插入到支路线条上，而不必新画任何连接。

当插入一个新的LD触点或线圈到空的编辑区域中时，新的水平支路线条将自动地从新元素画向已存在的左侧和右侧电源线；如果新触点或线圈未被放置在电源线之间，则不会自动画出水平线。新插入的触点或线圈可以在画出的支路上自由移动。插入LD触点或线圈时由编辑器生成的水平线可以被选中和删除。可以在已存在支路的水平线上插入新的LD触点或线圈。编辑器会自动地切断支路并将它连接到新插入的触点或线圈的左右连触点上。



## 多重连接

可以在任何输出点上建立多重连接。这意味着信息被传播到图表中的其它点上，即相同的状态被传播到右侧的每一个点上。输出连接点上能画数目不限的线条。除了下列LD符号之外，两条连接线的右端不能连接到同一个输入点：



.....右电源线

 .....左侧有多重连接的（或）运算符

这些LD符号可以拥有数目不限的输入。

### A.7.3 处理已存在的图表

“编辑”菜单中的命令用来改变或完成一个已存在的程序。这些命令中的大部分，都对程序图表中当前选定的元素进行操作。

#### **更正图表**

DEL键可用来删除选定的元素，悬空的连接也与选定的元素一同删除。使用“编辑/恢复”命令可以恢复被清除的元素。DEL命令也可应用于图中选定的成组元素。“编辑”菜单中的“剪切”、“拷贝”、“粘贴”命令用来移动和拷贝选定的元素。

#### **查找和替换**

“编辑/查找”和“编辑/替换”命令用来查找和替换程序图中的文本。只有完整的名称可以被找到。这些命令可作用于触点、线圈、块名、变量和标号。它们不能用来查找注释中的文本。替换命令不能用来改变块名。查找/替换可以从当前选择的位置向前或向后进行，在遇到程序边界时，会“循环”继续进行。

## **显示执行顺序**

当 FBD 程序包含了反向的循环指令，指令的执行就不能遵照从左至右/从上至下的顺序。为避免引起混乱，使用“工具/显示执行顺序”命令或按 Control + F1 键可以显示在编译时将采用的执行顺序。在能引起操作的元素符号（如线圈、设置变量和功能块）附近将显示从 1 编号到 N 的标记。



## **输入符号和文本**

可以在图形元素上双击鼠标来输入有关的符号和文本。这适用于变量、触点、和线圈、注释文本和标号。当对触点和线圈使用这种操作时，还可以用来改变它们的类型（直接、反相 ... ..）。

包括变量或模块选择列表在内的对话框通常用来输入文本。如果在“选项”菜单中选择了“手工键盘输入”，就可以在一个单一文本编辑框内直接输入变量符号和块名。输入新文本后按“Enter”键确认其有效；或按“Esc”键退出并关闭编辑框。在“手工键盘输入”模式下不能使用鼠标关闭文本编辑框。

如果在“选项”菜单中选择了“自动输入”模式，则在每次插入新的触点或线圈时立即输入变量符号。每当插入变量或标号时都要立即输入符号。



### 选择功能块类型

在一个块上双击鼠标可以用来改变它的类型。块的类型从当前可用的运算符、功能和功能块列表中选择。此命令也允许改变一些可交换的运算符(如 AND, OR, ADD, MUL... ) 的输入点数。



### 获得空白区

当用鼠标右键在 FBD 绘图区中单击时，会显示一个弹出式菜单。它包含一些可用来在当前鼠标指针位置插入或删除空白绘图区：

插入行：此命令将插入一个从打开弹出菜单时的鼠标指针位置开始，横跨4行网格宽的水平空白区。

删除行：此命令删除从鼠标指针处开始的未使用水平空白区（行）。此命令不能用来删除 FBD 元素。

当弹出菜单打开时，会由 FBD 绘图区中的灰色线标出要插入或删除空白区域的位置。

## A.7.4 显示选项

“选项”菜单中的命令用来定制 FBD 图表在屏幕上的表示方法。

## 工具提示

使用“选项 / 工具提示”指令来隐藏或显示变化的注释, 这个注释是作为工具提示出现在整个图中. 当游标移过变量块的, 注释也将作为工具提示出现. 选项在在线和离线状态下均可.

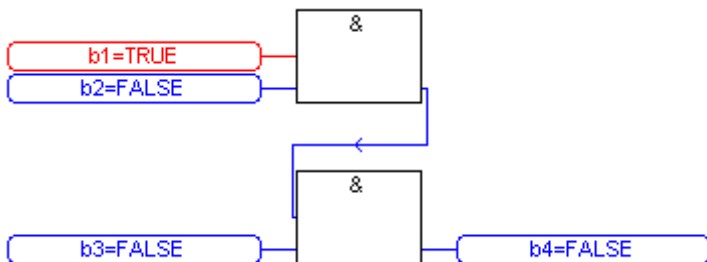
## 定制布局

“选项 / 布局”命令将打开一个对话框, 其中包括了成组的有关编辑器工作区和图形程序的画法的参数和选项. 使用“工作区”分组框中的选择项可显示或隐藏编辑器的工具栏和状态栏. “文档”分组框中的选项允许用户显示或隐藏编辑网格点.

## 使电流调试

仿真过程中 或 LD/FBD 在线 Debugging 程序中, 电流 出现强红色或蓝色, 为了容易跟踪逻辑流. 但是, enabling power flow debugging 影响目标板上的记忆分配. 在离线状态时, 从选项菜单上选择**使电流调试**. 这时您可以在 FBD 编辑器上调试电流流量. 当您安装 Workbench 后, 这个特性可以通过 default 来实现.

按照过程值, FBD 编辑器激活电流(图形连接). 所有 "0" or "错值" 条款出现蓝色. 所有非零值或 "真值" 条款出现红色



## 预览页面边界

打印时,按照选择的打印机和纸张, FBD 图形被对折分.在 FBD 编辑器中,您可以 预览每一个对折纸的边界.在创建您自己的图形时,预览功能可以避免您的一个符号分布在两页纸上..

当显示编辑格子时,在图纸中,这些纸张边界被显示成淡灰色.

FBD 编辑器使用最后选定的打印机配置来显示纸张大小. 在 FBD 编辑器中预览纸张边界之前,您必须激活文件打印机,并正确选择打印机和纸张. 因此,为了预览一个图形,您必须:

1. 运行文件产生器
2. 选择打印机,纸张大小,方向
3. 没有打印任务时,关闭文件产生器.
4. 打开 FBD 程序
5. 显示格子.



### “ 缩放 ”

分组框中的选项允许选用一个主要的缩放比率。您也可以使用编辑器工具栏上的“ 缩放 ”按钮在缺省的比率间切换。

“ 选项 / 字体 ”命令用来选择在所有 ISaGRAF 图形文档中使用的字体。选择字体时,字体的样式和大小不须指定, ISaGRAF 图形编辑器总是根据所选的缩放比率计算字体的大小。

## A.7.5 样式和修订跟踪

ISaGRAF LD/FBD 编辑器允许您给 LD/FBD 程序的每个元素分配一个图形样式。样式主要被定义为特别的图形着色，但 ISaGRAF 也允许用样式为控制程序版本的目的而进行修订跟踪。

注意样式在仿真或在线调试时是不可见的，因为在这些模式下颜色（红和蓝）被用来突出显示所监视的变量的真/假状态。

### ▣ **预定义样式**

下列样式已预定义：

**常用** 缺省画法（黑色）。使用修订跟踪时，具有“常用”样式的元素是原始图形程序的一部分。“常用”样式的元素将在程序执行时正常扫描。

**已修改** 标记为“已修改”的元素显示为粉红色。修订跟踪使用“已修改”样式突出表示自程序的原始版本以来所添加或改变的元素。具有“已修改”样式的元素将在程序执行时正常扫描。

**已删除** 标记为“已删除”的元素显示为灰色和虚线。程序执行时不考虑这些样式的元素。当使用版本控制时，此样式用来追踪记录自原始版本以来所删除的元素。

定制 作为预定义样式的附加，ISaGRAF LD/FBD 编辑器允许您选择任意一种颜色用于图表的一部分。这些元素被认为具有“定制”的样式。“定制”样式的使用对程序的执行没有影响。

使用“编辑”菜单中的“样式”子菜单中的命令，可将样式手工加在所选定的元素。

## **修订跟踪**

样式的使用，特别是“已删除”样式的使用，允许在已存在的程序中进行修订跟踪。使用“编辑/样式”菜单中的“标记修订”命令来使用或禁止修订跟踪。

设置“标记修订”选项后，程序中所有改变或添加的元素将自动使用“已修改”样式。当使用“删除”或“剪切”命令删除的元素，从视觉上并不从程序中消失，而只是被标为“已删除”样式。这就允许用户自动获得程序修订的记录。

使用“编辑/样式/清除所有已删除项目”才真正从LD/FBD程序中删除标记为“已删除”样式的元素。此命令与当前的选择集无关，而总是运用于整个程序。

选定元素并对其应用“常用”、“已修改”、或任何“定制”样式可以“恢复”标记为“已删除”样式的元素。这种操作可能引起无效的连接（同一输入点上

有多于一个的连接)，并将在下次程序校验时得以检测。

### A.7.6 在线帮助

为了从 LD/FBD 编辑器得到关于功能块的帮助:

在 LD/FBD 图上选取已有的功能块或把光标放在此处..

按 F1.

有关功能块帮助的内容便显示出来. 为了照顾"C"语言习惯 或 IEC 功能 或功能块, 帮助的内容为库编辑器中的 "技术注释" (仅限文本).

### A.7.7 打印 FBD 图

"File / Print"指令在打印机上输出 the FBD 图..这个指令 自动运行于 ISaGRAF 文档资料产生器上,从而打印出 FBD 图

## A.8 使用文本编辑器

本章仅描述 ISaGRAF 文本编辑器的性能和命令，特别是描述使用文本编辑器输入 ST 和 IL 程序的源代码。

### A.8.1 (编辑命令

“编辑”菜单的命令被用于操作所编辑的文本。其中大多数命令对图中当前所选择的字符起作用，或者是在插入符的当前位置上完成一个操作。



#### **剪切和粘贴**

DEL 键可用于删除所选择的文本。在执行一个 DEL 命令之后，用“编辑/复原”命令使元素复原。“编辑”菜单中的“剪切”，“拷贝”，“粘贴”命令被用于移动和拷贝程序中的文本，或把在其它应用中拷贝在剪贴板上的文本段落插入其中。



#### **查找和替换**

“编辑/查找”和“编辑/替换”菜单命令用于查找和替换程序中的文本。任何字符串都可以找到。查找时，从插入符的当前位置开始，可以向前或向后搜索。当搜索到程序的最后时，不进行“循环”搜索。

## **进入到某一行**

“编辑/进入到某一行”命令用于把插入符移到一个特定的行号。这对于快捷进入（由 ISaGRAF 编辑器在一个 ST 或 IL 程序中检测到的）含有错误的一行是非常有用的，并且用一个行号作参考标记。

## **插入字典中的符号**

用“编辑/插入变量”命令在插入符所在位置插入项目字典中所声明的一个变量或对象。通过公共变量选择框选择符号。在本文档中“有关程序编辑器的更多信息”一节对公共变量选择框进行了描述。

## **插入文件**

“编辑/插入文件”命令是在插入符的当前位置插入一个文件的全部内容。请注意：只有纯 ASCII 文本文件可以用本命令操作。

### **A.8.2 句法上色彩**

文本编辑器使用几种颜色显示语言中的关键词，变量识别器，常量措词... 这套颜色应用于句法上色彩，不能按当地习惯..

### A.8.3 选项

“选项”菜单中的命令用于显示或隐藏编辑器工具栏，并选择字符的字体。所选择的字符字体用在整个 ISaGRAF 工作台上的任何文本编辑。

当输入一个 ST / IL 程序的源代码时，“选项 / 显示关键词”命令用于显示或隐藏一个工具栏，该工具栏集中了 ST 或 IL 语言中最常用的关键词。在工具栏的某个按钮上单击，即可把相应的关键词或操作符插入到插入符的当前位置。

## A.9 程序编辑器

本章中包括了有关编辑特性的有用信息，这些特性对于所有 ISaGRAF 程序编辑器来说是共同的。本章内容主要涉及其他 ISaGRAF 工具和公共对话框。

### A.9.1 调用其它 ISaGRAF 工具



#### **校验 (编译) 程序**

“文件 / 校验”命令将运行 ISaGRAF 代码生成器来校验当前编辑的程序的语法。在 SFC 语言中，层次 1 和层次 2 都要被检查。语法检查结束后，必须关闭代码生成器窗口才能继续处理程序工作。如果应用程序中只有一个程序（被编辑的这个），而且未检测到语法错误，则会生成此程序的执行代码。“选项 / 编译选项”命令用来设置编译和优化参数。进一步的参考信息请见本帮助文档的“使用代码生成器”一节。



#### **仿真或调试应用程序**

使用“文件 / 仿真”和“文件 / 调试”命令在仿真和实时连接（在线）模式下都可运行 ISaGRAF 图形调试器，并在调试模式下重新打开所编辑的 SFC 程序。在调试模式下，不能对程序进行任何修改。



## 编辑变量字典

“文件/变量字典”命令用来编辑当前应用程序的变量词典。它也能编辑用户定义字。变量的局部声明或用用户定义字与当前编辑的程序有关(与3.2相同)

### A.9.2 程序参数

当正在编辑的程序是一个功能，功能块或子程序，可以用“文件 / 参数”命令来定义它的调用和返回参数。对于正在编辑的SFC程序或来自开始或结束区的最高层程序，此命令没有作用。

子程序，功能或功能块可以由多至32个参数（输入或输出）。功能或子程序总有一个（并且只有一个）与其同名的返回参数，与编写ST语言程序的约定相一致。

参数列表按照它们在调用模型中的顺序排列在窗口的左上角：调用参数在最前，返回参数在最后。窗口的下面显示关于在列表中选定的参数的详细描述。参数可以使用ISaGRAF中的任何数据类型。返回参数必须在调用参数之后。参数命名必须遵照以下的规则：

- 名称的长度不能超过16个字符
- 第一个字符必须为字母
- 随后的字符必须是字母，数字或下划线符号
- 命名与字符大小写无关

“插入”命令用来在选定的参数之前插入一个新参数。“删除”命令用来删除选定的参数。“排列”命令

对参数进行重新排列（排序），返回参数将被置于最后。

### A.9.3 "文件"菜单中的其它命令

在所有程序编辑器的"文件"菜单中，可以得到下列命令：



#### ***打开另一个程序***

"文件/打开"命令可使用户关闭当前正在编辑的程序，开始编辑用同一种语言编写的当前项目的其它程序。但这个功能不能用来编辑用不同语言编写的程序。新选中的程序将取代编辑窗口中的当前程序。



#### ***打印程序***

"文件/打印"命令可将经过编辑的程序输出到打印机。这个命令将自动运行 ISaGRAF 文件生成器，进行输出编辑程序和相应的局部变量的打印。

对于一些图形程序(如 SFC，FBD 和快捷 LD 程序)，用户还可以使用"编辑/拷贝绘图"命令，将剪贴板上的图形拷贝成位文件格式，使之可以粘贴到其它应用程序如字处理程序中去。但对于 SFC 程序，仅有第一层信息(一层图，编号和第一层注释)可以出现在被拷贝的位文件中。

### A.9.4 更新程序日志

当前编辑的程序相联系的日志文件，可以通过使用“文件/日志”命令手动生成。每次编译程序时，都会

用语法检查输出信息更新日志文件，并且都有日期/时间标记。

如果程序编辑器“选项”菜单中的“更新日志”模式被选中，则在每次保存文件到磁盘时会打开“日志”对话框。

如果按下“确定”按钮，输入的文本注释就会被加上当前的日期/时间标记，存储在日志文件的尾部。这一特性对于维护整个的程序非常有用，它为掌握程序的生命周期提供了帮助。

### A.9.5 从字典中选择变量

当编辑文本程序（ST或IL）时，“编辑/插入变量”命令允许选择一个已声明的变量名插入到当前插入符的位置。在编辑LD或FBD程序时，需要选择变量来描述触点、线圈、块I/O参数或FBD变量框。这两种情况下，都将打开“选择”对话框供选择已声明的变量。

“范围”选择框用来选择全局或局部变量。右侧的选择框用来选择数据类型。选择框边沿的小图标按钮可用作选择当前数据类型的快捷方法：



布尔



整型/实型



定时器



信息

用鼠标单击列表中的变量名，它的名称和注释将显示在列表的顶部，然后按“确定”按钮确认。也可以不用选择列表而在编辑控制台直接输入变量名)

### A.9.6 输出窗口

下列命令可以用于所有语言的编辑器“工具”菜单，在编辑窗口的下方的小文本表，显示信息，以及用于浏览程序。

“显示编译器输出”: 所编辑程序的最后一次编译时的出错信息显示在输出窗口。

“找到...”: 在整个所编程序中找到文本出现，在输出窗口列表。对于SFC和FC语言，在两层程序中查找。

“隐含输出窗口”: 关闭输出列表窗口

当出错信息和事件显示在输出窗口时，用鼠标双击一行，可以直接将选择移到出错或事件发生的位置。对于SFC和FC语言程序，  
此命令可以打开相应的第二层程序窗口。



## A.10 字典编辑器的使用

ISaGRAF 字典是对应用程序的内部变量、输入/输出变量、功能块实例和“定义字”进行声明的编辑工具。字典将应用程序中所声明的变量、功能块实例及定义字分别以常量字符串的形式进行分组。




在源代码中使用变量、功能块和定义字之前，必须在字典中对它们进行声明。变量和定义字可以用在一些自动化编程语言中，如：SFC、FBD、LD、ST 和 IL。由于 ISaGRAF FBD 和快捷 LD 编辑器都可以自动对所使用的功能块实例进行声明，所以对于在 FBD 语言中所使用的功能块，不需要加以声明。

### 变量

变量是根据它们的取值范围和类型来排序的。只有同样类型和同样取值范围的变量才可以在同一个输入框中输入。变量的基本取值范围如下：

-  ..... 全局变量      可在当前项目中的任何程序使用的变量
-  ..... 局部变量      只可由一个程序使用的变量

变量的基本类型如下：

-  ..... 布尔变量      真/假两个值
-  ..... 模拟变量      实型值或整型值
-  ..... 计时器变量      时间值

## ..... 信息变量          字符串




变量可由名称、注释、属性、网络地址和其它指定的域来标识。变量的基本属性如下：

内部变量	内存变量
输入变量	连接到一台输入设备上的变量
输出变量	连接到一台输出设备上的变量
常量....	仅可由内部变量(带有初始值)读的变量

注意：计时器变量永远是内部变量，输入变量和输出变量永远具有全局取值范围。

## **定义字**

定义字是一个可以在编程语言中取代文本字符串的别名。被取代的文本字符串可以是变量名、常量表达式或复杂表达式。定义字根据它们自己的取值范围进行排序。只有相同类型和同样取值范围的定义字才可以在同一个输入框中输入。定义字的基本范围如下：

 ..... 公共变量	可由任何项目中的任何程序使用的变量
 ..... 全局变量	可在当前项目中的任何程序使用的变量
 ..... 局部变量	只可由一个程序可用的变量

定义字可由一个名称、一个恰好与经过定义的功能块等价的 ST 字符串和一个空白注释来标识。



## 功能块实例

在 ST 和 IL 语言中使用的功能块实例必须在字典中进行声明。由于功能块中有内部“隐含”数据，因此必须对每个功能块拷贝进行标识。下面的例子中说明了在用于边沿检测的各种不同变量的库文件中定义的功能块“R\_TRIG”（即按上升边沿检测）的情况。功能块的每份拷贝必须用唯一的名称来识别。使用库文件管理器进行功能块类型的命名以及功能块参数的定义。

功能块名称: *R\_TRIG*

参数表:        输入变量 = *CLK*

                输出变量 = *Q*

使用字典编辑器进行功能块实例命名：

实例名称:    *TRIG\_B1*        功能块名称: *R\_TRIG*

实例名称:    *TRIG\_B2*        功能块名称: *R\_TRIG*

经过声明的实例可以应用在 ST 程序中：

*TRIG\_B1 (b1);*

*edge\_b1 := TRIG\_B1.Q;*        (\*b1 变量边沿检测 \*)

*TRIG\_B2 (b2);*

*edge\_b2 := TRIG\_B2.Q;*        (\*b2 变量边沿检测 \*)

经过声明的功能块实例可以是全局型（可在本项目的任何程序内使用）的或局部型（可在一个程序中内使用）的。由于 ISaGRAF FBD 和 LD 编辑器可以自动地对所使用

的功能块实例进行声明，所以应用在 FBD 的功能块不需进行声明。



(\*功能块总是有其在库文件中定义的块名。每次将功能块插入图形时，ISaGRAF FBD 和快捷 LD 编辑器都可以对实例进行自动声明\*)

由 FBD 和快捷 LD 编辑器自动进行声明的功能块实例，对于被编辑的程序来说永远是局部型的。

## 网络地址




网络地址是可选的。一个具有非零网络地址的变量，可以被一个处于运行状态的外部系统（例如一个过程可视化系统）监测到。更明白地讲，网络地址为每一个不能处理符号名的通讯系统提供一个标识机制。当变量正在建立或修改时，可以在每个变量的完整描述中输入网络地址。

### A.10.1 字典主窗口

字典编辑窗口中可以显示具有相同类型和相同取值范围的变量一览表。通常，被编辑变量的类型和取值范围显示在标题栏上。


✎ 编辑窗口中只显示变量描述中的主要域：如变量名称、属性、网络地址和文本注释。选定变量的全部描述总是显示于状态栏上。

使用工具条上的下列按钮，可以选择被编辑的变量取值范围：

-  公共变量.....可由任何项目中的任何程序使用的变量
-  全局变量.....可在当前项目中的任何程序使用的变量
-  局部变量.....只可由一个程序可用的变量

使用“Tab”键，可以控制显示在标题栏上的被编辑变量的类型：

布尔	整型/实型	定时器	信息	FB实例	已定义字
名称	属性	地址	注释		

 使用工具条左边的文本输入域，可以检索变量的前缀名。在这种情况下，根据当前选择，可以从整个一览表开头开始进行重新检索。也可以使用“编辑/查找”，来检索变量名和注释中的文本串，并将选中的字符串移入这个变量。检索通常与大小写无关的。

## A.10.2 管理变量

“文件”菜单中的各个命令，可用于全部选定的变量、功能块实例或定义字上。而“其它”命令则可用于选择被编辑对象的类型和取值范围。



### **打印变量**

使用“文件/打印”命令，可以在标准 Windows 打印设备上打印当前被编辑量或定义字的一览表。打印作业可以使用 ISaGRAF 文件生成器进行。打印输出包括每一个当前被编辑的变量或定义字的全部描述。



### **建立新变量**

“编辑/新建”命令允许用户建立具有选定取值范围和类型的新的变量、功能块实例或定义字。新变量插入在选择栏当前指向的变量之前。当运行“编辑/新建”命令时，先打开一个输入框，输入变量描述。当变量描述输入完毕后，按下“存储”按钮将该变量插入到一览表中。然后输入框自动重新打开，用户能够使用相同的“编辑”命令输入其它变量。按下对话框的“取消”按钮，中断变量的建立过程。



### **修改存在的变量**

“编辑”菜单中的“编辑”命令允许用户修改由选择栏指定的变量描述。当运行“编辑”命令时，先打开一个

输入框，修改变量描述。当变量描述修改完毕后，按下“存储”按钮，使修改有效。用户还可以分别按下“下一个”和“前一个”这两个按钮，将修改命令扩展到相邻的变量。但按下“取消”按钮则关闭对话框，不对任何修改进行存储。



## 剪贴和粘贴

ISaGRAF 字典编辑工具可以进行多行选择。许多命令都可以在正在编辑的变量一览表上工作。下面是可用的“编辑”菜单中的命令：

- 拷贝.....将选定的变量组复制到字典剪贴板上
- 剪切.....复制选定的变量组，并从编辑一览表中删除这个变量组
- 清除.....从编辑一览表中删除选定的变量组
- 粘贴.....在选定的变量之前，插入字典剪贴板的内容

拷贝/剪切/粘贴功能，可以在一个变量一览表到另一个一览表之间使用。但这个命令不能在不同对象类型的一览表之间使用。



## 变量排序

“工具/排序”命令可以对正在编辑的一览表中的变量或定义字进行排序。排序的顺序由变量属性决定：

- 首先是内存变量
- 然后是输入变量

- 最后是输出变量

具有相同属性的变量按字母表的顺序进行排序。定义字也按字母表的顺序进行排序。

## **设置网络地址**

网络地址是可选的。一个具有非零网络地址的变量，可以被一个处于运行状态的外部系统（例如一个过程可视化系统）监测到。当变量正在建立或修改时，可以在每个变量的完整描述中输入网络地址。“工具/地址重编号”命令允许用户为全局变量组建立网络地址。当运行“工具/地址重编号”命令时，这个命令对变量一览表上当前选定的变量组起作用。输入的十六进制基本地址（即该变量组的第一个变量的地址）使该组变量的网络地址设置为连续地址。当输入的基本地址为零时，将所有选定的变量的网络地址重新设置为零。



## **输入布尔型“真/假”字符串**

当编辑定义字时，“工具/输入真/假定义”命令允许用户自动定义作为字典关键字，添加到表示真和假状态的布尔变量上的字符串。这样的字符串通常是为调试格式化而定义的。如果在程序中使用这些的字符串，就必须把它们指定为定义字。在当前选定的定义字的检索中，这个命令可以在相同的范围内进行布尔变量真/假字符串的检索。

### A.10.3 对象的描述

对于每一个变量、功能块实例或定义字必须输入完整的描述。描述域对于每种类型的对象是不同的。下列域对于任何变量来说是相同的：

- 变量名 ..... 变量名：第一个字符必须是字母，以后的字符必须是字母、数字或下划线‘\_’。
- 网络地址 ..... 十六进制网络地址（可选）。当本域为非零值时，变量可被运行中的外部系统监测到。
- 注释 ..... 用于变量描述的注释说明。
- 保持 ..... 该选项表明该变量必须保存在备份内存中。



以下是布尔型变量的其他描述域：

- 特征 ..... 用来指定内存变量、常量、输入变量或输出变量。
- “假”字符串 ..... 在调试时，作为假值使用的字符串。
- “真”字符串 ..... 在调试时，作为真值使用的字符串。
- 设初始值为真 ..... 当选中本选项时，初始值为真，否则初始值为假。



以下是整型或实型变量的其他描述域：

- 特征 ..... 用来指定内存变量、常量、输入变量或输出变量。

- 格式.....用来指定整型变量或实型(浮点)变量。  
在变量调试时所使用的显示格式。
- 单位字符串.....在调试时,用来表示物理单位的字符串。
- 转换.....与变量(仅用于输入变量或输出变量)有关的转换表或变换函数的名称。
- 初始值.....变量的初始值(必须与变量格式相同)。  
如果没有指定,则初始值为0。



以下是计时器变量的其他描述域:

- 特征.....用来指定内存变量、常量。
- 初始值.....变量的初始值(时间值)。如果没有指定,则初始值为时间#0s。



以下是信息型变量的其他描述域:

- 特征.....用来指定内存变量、常量、输入变量或输出变量。
- 最大长度.....用来指定可以存放在该信息变量中的最大字符数。
- 初始值.....变量的初始值(长度不能超过该变量的长度)。如果没有指定,则初始值为空字符串。



以下是定义字的其他描述域:

名称.....在ST源文件中使用的名称：第一个字符必须是字母，以后的字符必须是字母、数字或下划线‘\_’。

定义.....符合ST语法规则的字符串，它在编译时取代定义字。如：Name = PI - Equivalence = 3.14159

注释.....用于所定义的等式的注释说明。



以下是功能块实例的其他描述域：

名称.....在ST源文件中使用的实例名称：第一个字符必须是字母，以后的字符必须是字母、数字或下划线‘\_’。

类型.....在库文件相应的功能块名称。

注释.....功能块实例的注释说明。

#### A.10.4 快捷声明

“工具/快捷声明”命令可使用户同时声明若干个变量。使用快捷声明命令进行声明的变量应使用编号转换进行命名。为了进行变量编号转换，用户应定义以下内容：

- 第一个变量和最后一个变量的索引(号)，
- 在变量字符中，所增加文本的前后编号
- 在变量字符中，用来表示编号的数字位数。

另外，用户还应建立所建立的变量的基本属性(如内部变量、输入变量、输出变量等)。另根据变量的类型，增加一些其它特性(如“保持”属性、整型变量或实型变量格式、信息变量字符串最大长度等)。

由于一个变量不能以数字开头，用户经常需要定义一个字符串在变量编号之前。当“编号位数”设置为“自动”时，ISaGRAF将按照所需的最小编号位数格式化变量编号。当指定编号位数时，ISaGRAF按照指定长度格式化变量编号，变量编号的长度比指定的长度短时，以'0'字符补齐。为变量编号设置固定的编号位数，对于避免发生对变量的不恰当的词典式分类是非常有用的。下面是一些例子。

例 1: 为快捷声明进行设置：

编号：

自：  到：

数字：

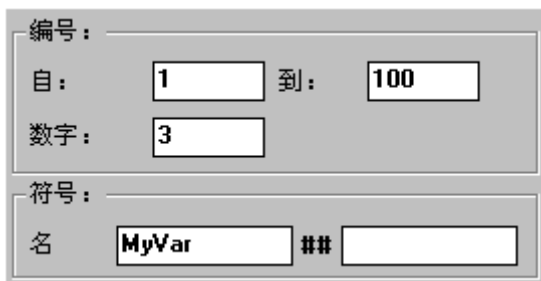
符号：

名  ##

这样，就建立下列三种变量：

Var9xx    Var10xx    Var11xx

例 2: 为快捷声明进行设置:



该对话框用于配置快捷声明。它包含两个主要部分：'编号'和'符号'。在'编号'部分，'自'字后面的输入框中填入数字'1'，'到'字后面的输入框中填入数字'100'，'数字'字后面的输入框中填入数字'3'。在'符号'部分，'名'字后面的输入框中填入文本'MyVar'，紧接着是'##'符号，最后是一个空的输入框。

这样，将建立名称从 MyVar001 到 MyVar100 这样 100 个变量。

### A.10.5 Modbus SCADA 寻址图

ISaGRAF 中的“网络地址”经常用于完成 ISaGRAF 系统与基于 Modbus 总线通讯模式的 SCADA 系统之间的连接。在这种情况下，SCADA 是 Mod 总线的主机，ISaGRAF 目标作为 Mod 总线的从机。网络地址用来为了由 SCADA 控制的所有各个 ISaGRAF 变量建立虚拟 Modbus 总线图。“工具 / Modbus SCADA 寻址图”命令对于快捷产生一个带有应用程序变量的虚拟 Mod 总线图是非常有用的。

制图工具中显示两个一览表。上面的一览表是显示已经编入图内的变量情况(变量都具有网络地址)的 Mod 总线图中的一个段(共 4096 个位置)。下面的一览表用

来显示没有编入图内的变量(即没有定义网络地址的变量)的情况。“0”地址不能用于变量制图。

使用“编辑”菜单中的“制图”命令和“移动”命令，可将变量从一个一览表移动到另一个一览表之中。同样的动作也可由双击一个一览表中的变量符号，并将这个变量符号送到另一个一览表的操作来完成。用户可以随时使用“段”按钮，下拉一览表来观察寻址图中其它段的情况。

“选项”菜单中的命令可用于随时显示十进制或十六进制的网络地址。

“编辑/查找”命令用来检索已声明的变量，不管这个变量是否已经制图。

## **A.10.6 与其他应用程序交换信息**

为了同其它应用程序，比如字处理、电子表格、数据库管理等软件交换信息，ISaGRAF系统的字典编辑工具提供了输入/输出功能。这些输入/输出命令均放置在“工具”菜单中。“输出文本”命令将建立一个纯ASCII文本的格式描述域，用以描述一系列的编辑对象，并将文本存贮在Windows剪贴板或存在一个文件中。这些数据信息可以由其它应用程序使用。“输入文本”命令则以纯ASCII文本格式输入变量的说明描述域，并将

其存贮在 Windows 剪贴板或存在一个文件中，并且用刚输入的说明描述域对正在编辑的一览表进行更新。其它应用程序也生成类似的数据信息。

## 二 输出数据

执行“输出文本”命令时，会出现“输出”对话框，这个对话框使用户可以控制输出结构。

如果“完整列表”选项被选中，表明完整的编辑一览表已经输出。在这种情况下，系统将忽略当前的选择。如果“选择变量”选项被选中，表明只输出高亮的变量。

如果“剪贴板”选项被选中，输出信息将以纯ASCII文本的格式存储到Windows的剪贴板中。这个文本随后可用“粘贴”命令粘到其他应用程序中去。如果“文件”选项被选中，输出的文本将被存到一个ASCII码文件中。应输入这个ASCII码文件的完整的路径名。“浏览”命令可用于寻找已存在的路径名。

然后用户可以选择输出文本的格式。可使用的文本格式在以后的章节中说明。按下“确认”按钮开始执行输出功能。按下“取消”按钮关闭对话框，从输出命令中退出。

所有被选中的描述域，都按照标准的声明顺序，存储在输出文本之中。输出文本的第一行中包含所选中的描述域的名称。每个对象都用输出文本中的一行来描

述。“行结束”分隔符是标准MS-DOS顺序“0d-0a”。按下“关键字”按钮，可以对输出文本的第一行中用于标识描述域的名称进行修改。这个命令将在以后的章节中进行说明。

## 二 输入数据

执行“输入文本”命令时，会出现“输入”对话框，这个对话框使得用户可以控制输入结构。

如果“剪贴板”选项被选中，系统可以从Windows剪贴板上读到纯ASCII文本格式的输入信息。如果“文件”选项被选中，系统将从一个ASCII码文件中读出输出的文本。应输入这个ASCII码文件的完整的路径名。“浏览”命令用来寻找已存在的路径名。

输入函数可以自动识别输入文本所用的格式(即分隔符)。这些可使用的格式将在以后的章节中描述。按下“确认”按钮，执行输入数据。按下“取消”按钮，将关闭对话框，从输入命令中退出。按下“关键字”按钮，可以对输入文本的第一行中用于标识描述域的描述域名称进行修改。这个命令将在以后的章节中进行说明。

输入文本的第一行必须包含描述域的名称，这些描述域按照下列行的顺序排列。每个项目都用输入文本中的一行来描述。“行结束”分隔符是标准MS-DOS顺序“0d-0a”。描述域可按任何顺序出现。如果有些描述域丢失了，系统将自动地以缺省值填补到输入对象描

述域中去。如果输入对象已在编辑一览表中存在，用户应须确认对输入对象的覆写。随后用项目的说明来更新输入描述域。如果有些描述域丢失，在项目说明中不能更新这些描述域。

二      **可用的文本格式**

下面是输出命令可使用的格式一览表。输入命令会自动识别这些格式。

- **制表分隔符**

*描述:*                      描述域由制表符分隔。

*例如:*                      名称      特征              注释  
                                 级别      内存变量    内部计算的水位  
                                 报警 1    输出变量    主报警输出变量

- **逗号分隔符**

*描述:*                      描述域由逗号分隔。

*例如:*                      名称，特征，注释  
                                 级别，内存变量，内部计算的水位  
                                 报警 1，输出变量，主报警输出变量

- **分号分隔符**

*描述:*                      描述域由分号分隔。

例如:                   名称 ; 特征 ; 注释  
                          级别 ; 内存变量 ; 内部计算的水位  
                          报警 1 ; 输出变量 ; 主报警输出变量

- 逗号和引号

描述:                   描述域由逗号分隔。  
                          每个描述域写在引号内。

例如:                   “名称” , “特征” , “注释”  
                          “级别” , “内存变量” , “内部计算水位”  
                          “报警 1” , “输出变量” , “主报警输出变  
量”

## 二 关键字

按下“关键字”按钮，就可以对位于输入文件或输出文件第一行用来标识描述域的描述域名称进行修改。屏幕窗口中显示出与对象域一览表以及与之相关的关键字。用户可在关键字一览表选择一个描述域后，按下“修改”按钮，以修改一个关键字。按下“缺省”按钮，系统将恢复关键字的初始一览表。关键字的命名必须符合下列规则：

- 名称不得超过 16 个字符
- 第一个字符必须是字母
- 以后的字符必须是字母、数字或下划线 ‘\_’ 字符
- 不同的关键字不能采用相同的名称

以下是 ISaGRAF 使用的标准关键字：

对象名 .....	关键字名称
文本注释 .....	注释
网络地址 .....	地址
属性 (内部、输入、输出) .....	特征
布尔 '假' 字符串 .....	假
布尔 '真' 字符串 .....	真
模拟格式 (实型或整型) .....	格式
模拟量单位字符串 .....	单位
模拟转换名 .....	转换
信息型变量最大长度 .....	最大长度
功能块库类型 .....	库
定义字等价运算符 .....	等价
内部属性 .....	内部
输入属性 .....	输入
输出属性 .....	输出
常量属性 .....	常量
实型模拟格式 .....	实型
整型模拟格式 .....	整型 (同 3.2)

## A.11 输入 / 输出连接编辑器的使用

输入 / 输出连接操作的目的是：在应用程序的输入 / 输出变量与安装在目标设备上的输入 / 输出板上的物理通道之间建立逻辑连接。为了建立这样的连接，用户必须检查并且设置目标设备上所有的输入 / 输出插板，将输入 / 输出变量连接在在相应的输入 / 输出通道上。

编辑窗口左边表格中显示出带有插板槽的目标设备的机柜。插板槽可以是空的，也可以插入一块输入 / 输出插板或一个综合设备。每个插板槽用一个序号来标识。一个机柜内最多可插 255 块插板。编辑窗口右边表格中显示输入 / 输出插板的参数和与选定的输入 / 输出插板相关的变量。一块输入 / 输出插板最多可有 128 个输入 / 输出通道。但单个输入 / 输出插板的总数 (包括单个设备和综合设备板) 不能超过 255 个。




### 图标

在前面板上显示的各个图标，是可以连接到插板通道上变量的类型和特征。ISaGRAF 系统不允许同一块插板上的连接不同类型的变量。下面是所用图标的含义：




- ..... 布尔型变量
- ..... 整型 / 实型 (两种可以连接的变量) 变量
- xy² ..... 信息变量
- ⇄ ..... 输入变量 - 没有接通的通道
- ⇄ ..... 输出变量 - 没有接通的通道

- ➔.....输入变量 - 至少连接一个通道
- ➔.....输出变量 - 至少连接一个通道

下面的图标用来表示插在插板槽中的 I/O 设备的类型：

- .....综合输入/输出设备
- .....实际输入/输出插板
- .....虚拟输入/输出插板

下面的图标用来表示参数或通道的情况：

- .....插板参数
- .....空闲通道
- .....被接通的通道

### **在列表中移动插板位置**

使用工具条中的按钮或“编辑/上移板/下移”菜单中的命令，可以将选定的 I/O 插板在主表格中向上移动或向下移动一行。“编辑/插入槽”命令可以在当前位置上再插入一个空插槽。

## **A.11.1 定义输入/输出插板**

“编辑”菜单中包含了定义所选定的插板（即建立插板参数）以及将输入/输出变量与其各自通道相连接的基本命令。



### **选择输入/输出插板的类型**

在把输入/输出变量连接到插板之前，必须输入这块插板的特征。这时可以使用 ISaGRAF 工作台中的预定义插板库。这个库文件被一台或多台输入/输出设备供货商编译过。“编辑/设置板/设备”命令可以用于设立插板标识特征。这个命令也可用于从 ISaGRAF 库中选择一个单一的插板或者综合设备。采用双击某一个插槽，来设置相应的插板或设备，也是可行的。

所有单一插板上的各个通道都具有相同的类型(如布尔变量、整型/实型变量或信息变量等)和方向(如输入或输出)。在输入/输出连接时，不对实型变量和整型变量进行区分。综合输入/输出设备表示它是一台具有不同类型和方向通道的输入/输出设备。一台综合输入/输出设备表示为若干块单一输入/输出插板的一览表。但它仍仅占机架列表中的一个插槽。



### **删除插板**

“编辑/删除插槽”命令用来删除当前已选定的插板或输入/输出设备。如果变量已经连到相应的通道上，当清除板槽时，将自动切断这种连接。



### **实际插板和虚拟插板**

“编辑/实际插板和虚拟插板”命令用来设定所选定的插板或综合输入/输出设备的有效性。下列图标用来显示插板的有效性：

 ..... 实际输入 / 输出插板

 ..... 虚拟输入 / 输出插板

在“实际模式”中，输入/输出变量直接连到相应的输入/输出设备上。应用程序中对输入或输出装置的操作也直接作用于相应实际输入/输出设备的输入或输出环境中。但准确地说，在“虚拟模式”中，输入/输出变量只是作为内部变量来处理的。诊断器可以对这些输入/输出变量进行读操作或更新操作，使用户可以对这些 I/O 变量进行仿真处理，但并非建立起实际的连接。



### **技术说明**

“工具/技术说明”命令显示所选定的插板或综合设备上的在线用户指南。插板技术说明是由输入/输出插板的供应商编写的。技术说明中包括了有关输入/输出插板管理的全部信息，并且说明了输入/输出插板参数的含义。



### **删除被连接的变量**

“工具/删除被连接的变量”命令用于删除全部已经连接到选定插板上的输入/输出变量。



### **为空通道定义注释**

已经声明的输入/输出变量的注释文本与变量名一起被显示在输入/输出板的一览表中。由于 ISaGRAF 系统允许使用直接显示变量（显示变量以 % 为标记），所以

注释也可用于空通道。可以使用“工具/设置通道注释”命令为在当前插板一览表选定的空通道输入注释。但这个命令不能用来改变在项目字典中声明的输入/输出变量的注释。

### A.11.2 设置插板参数

用户可以采用双击屏幕右边表格中该参数名的方法，来设置插板参数值。也可以选择“编辑”菜单的“设置通道/参数”命令来选择参数(使选中的参数变为高亮)。参数在右边一览表的开始处列出。下列图标用于在列表中表示插板参数：

.....插板参数

插板参数的含义和参数的输入格式是由相应的输入/输出板或设备供应商设计的。用户为了获得更多有关插板参数的信息，可以使用“工具/技术说明”命令或参考硬件手册。

### A.11.3 连接变量

用户可以采用双击屏幕右边表格中参数名的方法来设置通道连接参数。也可以选择“编辑/设置通道/参数”命令来选择参数(使选中的参数变为高亮)。下列图标用于表示通道特征：

.....空通道

## ● .....被连接的通道

这个一览表中包含了全部与所选定插板的类型和方向相匹配的变量。只有那些还没有被连接的变量在此列出。“连接”按钮用于将在一览表中选定的变量连接到选定通道上。“断开”按钮则用于从选定的通道删除(切断)变量。“下一个”和“上一个”按钮用于选择输入/输出插板上的其它通道。所选定的通道的位置总是显示在对话框标题内。

### A.11.4 可直接显示的变量

空通道是那些没有与已经声明的输入/输出变量相连接的通道。ISaGRAF 允许用户在源程序中使用可直接显示的变量来描述一个空通道。可直接显示的变量的变量名总是以字符"%"开始。

以下是用于单个插板通道的可直接显示的变量的命名规定。"s"是插板上的板槽号。"c"是通道号。

%IXs.c.....布尔型变量输入插板的空通道  
 %IDs.c.....整型变量输入插板的空通道  
 %ISs.c.....信息型变量输入插板的空通道  
 %QXs.c.....布尔型变量输出插板的空通道  
 %QDs.c.....整型变量输出插板的空通道  
 %QSs.c.....信息型变量输出插板的空通道

以下是用于综合设备通道的可直接显示的变量的命名规定。"s"是设备的板槽号。"b"是在综合设备内独立插板号。"c"是通道号。

%IXs.b.c.....布尔型变量输入插板的空通道

%IDs.b.c.....整型变量输入插板的空通道

%ISs.b.c.....信息型变量输入插板的空通道

%QXs.b.c.....布尔型变量输出插板的空通道

%QDs.b.c.....整型变量输出插板的空通道

%QSSs.b.c.....信息型变量输出插板的空通道

例子如下：

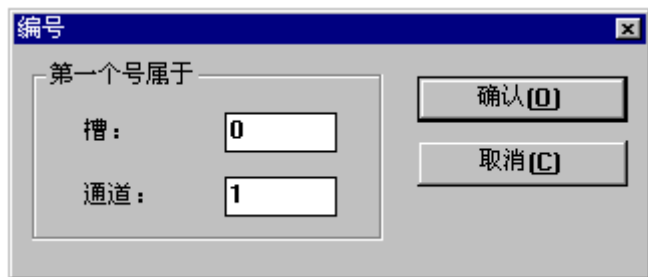
%QX1.6            插板#1(布尔型变量输出)的第六通道

%ID2.1.7        设备#2上插板#1(整型变量输入)的第七通道

可直接显示的变量不能是实型变量。

### A.11.5 编号

使用“选项/编号”命令可以设置编号协议。用户可以在下面的对话框中，指定第一个板槽的编号以及每一块插板的第一个通道所使用的编号：



板槽的编号从0开始，通道的编号从1开始。这是系统的缺省值。

警告：在编号协议已经对可直接显示的变量所使用的符号起作用时，改变这个协议必须非常小心。假如在已存在的程序中使用可直接显示的输入/输出变量，可能会产生编译错误。

### A.11.6 设置单独保护

ISaGRAF 工作台提供了一个完整的以层次口令为基础的数据保护系统。输入/输出连接设置等都可以受到口令的全面保护。另外，ISaGRAF 还允许用户为变量和通道设置单独保护。假设：

各个口令已经在口令定义系统(即使用项目管理器窗口中的“项目/设口令”命令)中定义完毕，这样所对应的各个保护级别就可以用于单独保护。

- 用户可以使用比全局 I/O 具有更高优先级的保护级别，进行变量的单独保护。

当变量和通道设置受到单独保护时，在输入/输出连接窗口中，受保护对象的名字边会有一个小图标。



使用代码窗口或输入/输出连接窗口中“编辑”菜单中的“加保护”和“删除保护”命令，可以对选定的变量或通道进行单独保护的设置或删除。“加保护”和“删除保护”这两条命令要求用户输入一个有效的口令，使保护级别与变量或通道联系起来。因此，在每次用户想要改变带有单独保护的变量或连接通道设置时，必须先输入具有足够优先级别的口令。

**警告：**如果一个变量或通道设置受到保护，而相应的口令从保护系统中被删除，并且没有定义更高级的口令，则这个变量或通道设置不能进行任何改动，除非定义了具有一定级别的新口令。

## A.12 转换表的生成

ISaGRAF 工作平台允许用户生成转换表。转换表是用来定义模拟转换点的集合。它用于模拟输入或输出变量，它的作用是在电量值（即从输入传感器读入的数值或送到输出设备的数值）与物理值（应用程序编程所用的数值）之间建立起比例关系。

转换表可以由 ISaGRAF 字典窗口中的“工具/转换”命令启动的对话框，进行编辑。

一个已定义的转换表可以用来过滤有关选定项目的任何输入模拟变量或输出模拟变量的数值。使用 ISaGRAF 字典，变量声明编辑器中的命令，可以将转换表和变量连接在一起。然后应选定输入或输出模拟变量，编辑变量的参数。但一个变量不能连接到还没有定义的转换表上。

### A.12.1 主命令

“转换表”对话框可以显示已定义的转换表的一览表，也包括了一些主命令按钮，用以进行编辑已存在的转换表（定义表中的输入/输出点）、建立新转换表，转换表重新命名或删除转换表等项操作。

按下“转换表”对话框中的“确认”键可以退出对话框，并将进行过的操作存在盘上。



## 建立新表

“新建”命令可使用户建立新的转换表。每个项目最多可以建立 127 张转换表。只有使用过的转换表(这个转换表与模拟变量相连)才可插入到应用程序的可执行代码之中。命名一个转换表必须按照下列规则：

- 变量名不能超过 16 个字符
- 第一个字符必须是字母
- 后面的字符可以是字母、数字或下划线 '\_'
- 转换表名是大小写无关的



## 修改转换表内容

“编辑”命令用于对一览表中选定的转换点进行输入。也可采用双击转换表名的方法对转换点进行输入。生成一个新转换表时将自动调用“编辑”命令。一张转换表必须至少输入两个转换点。

### A.12.2 转换表中的入点

“编辑”对话框允许用户定义转换表中的转换点。在这个对话框的左侧显示已经定义过的转换点的一览表。对话框的右下角以图形曲线的方式显示已定义的转换表。可以使用对话框中的命令输入各个点。用户必须遵守本节末所讲述的几条点定义编号规则。左侧框中总是包含当前编辑的转换表中已存在点的一览表。左边列显示这些点的电量值(外部值)。右边列显示这些点的物理值(内部值)。为了修改这些点的数值或为了

清除 (删除) 这些点，用户必须在一览表中选择该点。一览表的最末选择 (“...”) 可用于定义一个新点。右下角的框以图形曲线的方式显示当前正在编辑的转换表。由于显示的是曲线的比例关系，所以图形没有显示坐标轴和坐标。在快捷检查曲线定义是否合适时，这种表示方式是很有用的。

## 定义新点

在定义一个新点时，可在转换点的一览表上选择最后输入 (“...”)。开始定义新转换表时，使用缺省模式。用户必须为每个点输入电量值 (外部值) 和物理值 (内部值) 两个数值。这些数值将以单精度浮点数的形式存储起来。请记住：定义一条曲线必须至少输入两个点。当这些点的电量值 (外部值) 和物理值 (内部值) 都输入之后，按下 “存储” 按钮，就可以把这些点添加到转换表中。每一个转换表最多可定义 32 个点。

## 修改转换点

如要修改已存在的转换点，首先要在转换一览表中选择这个点，然后输入点上新的电量值 (外部值) 和物理值 (内部值)。这些数值将以单精度浮点数的形式存储起来。当两个值都输入完毕后，按下 “存储” 按钮，转换表中的该点的数据就被更新了。

## 清除转换点

按下“清除”按钮，可以从转换一览表中清除已被选中的现存点。请记住：定义一条曲线必须至少输入两个点。

### A.12.3 规则和限制

在定义一个转换表时，必须遵守下面的规则。转换表可用来转换输入和输出模拟变量：

- 两个点不能被定义为同一个电量值
- 曲线必须连续上升或连续下降
- 两个点不能被定义为同一个物理值

在为一个项目定义转换表时，也应遵守下面的限制：

- 在同一项目中，定义的转换表不能多于127个。
- 在同一转换表中，定义的点不能多于32个。

## A.13 使用代码生成器

代码生成窗口由其它 ISaGRAF 工作台窗口的“校验”和“制作”命令自动打开。当所需要的代码生成操作结束时，代码生成窗口不会自动关闭，这样用户仍能从窗口菜单使用所有的代码生成命令和选项。

### A.13.1 主命令

“文件”菜单包括检查程序语法和代码生成的命令。

#### **制作应用程序代码**

“制作”命令建立该项目的所有代码。在产生任何代码之前，该命令检查声明语句和程序的语法。在编译单个程序时无法检测到的任何错误在代码生成时被检测到。这种情况适用于变换表，输入/输出变量连接以及与库之间的链接。当检测到错误时，代码生成程序暂停对一个程序的编译。在继续执行代码生成程序之前必须修正该程序。自上一次“校验”操作以来被检查的程序(未检测到错误)和尚未修改的程序不能被重新编译。变量声明语句的校验和应用程序的相关检查总是被处理。在检查程序时，敲击 ESCAPE 键可以停止“制作”操作。

注意: 如果程序的一个局部变量声明已被修改, 则该程序被校验。如果一个全局变量已被修改, 则校验所有的程序。

## ☐ **检查程序语法**

“校验程序”命令允许用户只能校验一个程序。即使从上一次校验以来所选程序未被修改, 该程序仍被编译。  
“校验字典”命令允许用户校验项目的所有变量的声明。

“校验所有程序”命令检查项目的所有程序的语法, 即使它们中的一些程序未被修改。当某个程序被检测出一个错误时, 该命令不中止(执行)。该功能可用于给项目程序中的所有错误生成一个完整的表。敲击 ESCAPE 键可停止该命令的执行。

## ☐ **仿真一个修改**

“尝试”命令仿真所有项目程序的一个修改, 这样, 在下次“制作”操作时所有程序均被校验。“打开”命令用于打开最后被校验的程序。该命令对于直接进入已被检测到语法错误的一个程序是非常有用的。

## **A.13.2 编译程序的选项**

“编译程序选项”命令用于建立由 ISaGRAF 代码生成器使用的主要参数, 以便产生和优化目标代码。该命令

的目的是根据相应的 ISaGRAF 目标选择必须生成的代码类型，并根据所期望的编译时间和应用程序运行时间所提出的要求建立优化参数。

☐ “上载”按钮打开第二个对话框，该对话框带有能够把压缩后的源代码嵌入下载代码的其它选项，以便实现“上载”功能。有关这方面的进一步解释请查阅“上载”资料。

## ☐ 选择目标

上面的表是显示能生成的、可供选用的目标代码表，符号“>>”用于指出所选择的目标。ISaGRAF 代码生成器在同一个编译操作中可以产生 3 种不同的代码。根据你的目标硬件使用“选择”和“不选”按钮设置所需要的目标代码表。以下是标准的 ISaGRAF 目标：

SIMULATE: ...此代码供工作台上的 ISaGRAF 仿真器使用。

如果该目标没有被选择用于产生应用程序代码，则仿真器不能运行。

ISA86M: .....它是一个供安装在 Intel 处理器上 ISaGRAF 内核使用的 TIC 代码(独立于目标的代码)。处理器类型仅与已生成的代码的位顺序有关。

ISA68M: .....它是一个供安装在 Motorola 处理器上 ISaGRAF 内核使用的 TIC 代码(独立于目标的代码)。处理器的类型仅与已生成的代码的位 (byte) 顺序有关。

SCC:.....选择该目标使 ISaGRAF 编译程序产生结构化的“C”语言源代码，该代码将被编译并与 ISaGRAF 目标内核库链接，以便产生一个被嵌入的可执行代码。

CC86M:.....选择该目标使 ISaGRAF 编译程序产生非结构化的“C”语言源代码，该代码将被编译并与 ISaGRAF 目标内核库链接，以便产生一个被嵌入的可执行代码。当不支持结构化“C”代码生成和集成时，该选项被用于与 V3.23 以前的 ISaGRAF 版本相兼容。

如果想知道安装在您的 PLC 上的 ISaGRAF 目标内核的型号，请查阅硬件手册。其它目标型号(机器代码，C 源代码...)在以后出版的 ISaGRAF 工作台上将被支持。



### **SFC 处理**

检查“使用被嵌入的 SFC 生成器”框，以便能够使用 ISaGRAF SFC 生成器。*由于该模式能产生较高的运行时间性能，因此被优先选用。*然而，在 ISaGRAF 目标实现某些特殊功能时，该目标生成器有可能并不存在。这里所说的特殊功能，通常是指基于 ISaGRAF 代码后-处理的用户特定目标。在这种情况下，你也许不得不取消这一选项，并让 ISaGRAF 编译程序把 SFC 流程图转换成低级指令。有关使用该选项的更多信息请查阅你的硬件资料。

## 优化选项

以下是 ISaGRAF 代码生成器用于优化目标代码所使用的参数，这些参数可以从“编译程序选项”对话框中设置。“缺省”按钮被用于删除所有的优化选项，以便减少编译时间。

◆ 当“运行两个优化程序周期”选项被设置时，ISaGRAF 代码优化程序被运行两次。通常在第二个周期所作的优化操作的有效性比第一个周期所作的优化操作要小一些。

◆ 当“判定常量表达式”选项被设置时，常量表达式由编译程序来判定。例如，数字表达式“ $2 + 3$ ”在目标代码中用“5”代替。当该选项未被设置时，常量表达式在运行时间内计算。

◆ 当“取消未使用的标号”选项被设置时，优化程序简化程序的跳转和标号系统，以便取消未使用的目标标号或空跳转。

◆ 当“优化变量拷贝”选项被设置时，暂时使用的变量(用于保存中间结果)被优化。该选项通常与“优化表达式”选项一起使用。当该选项被设置时，优化程序再次使用表达式和子表达式，这些表达式在程序中使用一次以上。

◆ 当“取消未使用的代码”选项被设置时，优化程序取消无效的代码。例如，如果下列语句被编程：

“ var := 1; var := X;” , 所生成的相应代码只有: “ var := X;” .

◆ 当“ 优化算术运算” 选项被设置时, 优化程序根据特殊的操作符简化算术运算。例如, 表达式“ A + 0” 将被“ A” 代替。当“ 优化布尔运算” 选项被设置时, 优化程序根据特殊的操作符简化布尔运算。例如, 布尔表达式“ A & A” 将被“ A” 代替。

◆ 当“ 建立两个判断图” 选项被设置时, 优化程序用一个有条件的跳转操作简化表来替代布尔方程式(把 AND, 或, XOR 和 NOT 操作符混合在一起)。只有当跳转程序所期望的执行时间小于原表达式所期望的执行时间时, 才进行转化。

下列表格概括了所期望的优化及每个参数所需要的编译时间:

	增益 (性能).....	编译时间
运行两个周期	XXXX .....	(*)
优化常量表达式	XXXXXXXXXX .....	XXXX
取消未使用的标号	XXXX .....	XXXXXXXXXX
优化变量拷贝	XXXX .....	XXXXXXXXXX
优化表达式	XXXX .....	XXXXXXXXXX
取消未使用的代码	XXXX .....	XXXXXXXXXX
优化算术运算	XXXXXXXXXX .....	XXXX
优化布尔运算	XXXXXXXXXX .....	XXXX
建立两个判断图	XXXXXXXXXXXXXXXXX .....	XXXXXXXXXXXXXXXXX

(\*) 编译时间也是2的倍数。

### A.13.3 产生 C 源代码

ISaGRAF 工作台能够用“C”语言产生源代码。在这种情况下，应用程序的整个内容（包括SFC流程图描述、数据库定义和代码的顺序）用“C”源代码的格式产生。在此推荐被生成代码的两种风格：

CC86M(C源代码-V3.04) 产生非结构化的“C”源代码。如果你的目标软件是以ISaGRAF 3.23以前的版本为基础，则应该选择这种风格。

SCC(结构化的C源代码) 产生结构化的“C”源代码。如果你的目标软件是以ISaGRAF 3.23或以后的版本为基础，则应该选择这种风格。

以下是在项目目录中产生的两个文件：

APPLI.C      应用程序的通用源代码

APPLI.H      通用的“C”语言定义

在生成结构化“C”源代码的情况下，除了通用的“APPLI.C”和“APPLI.H”文件以外，还为应用程序的每一个程序产生一个“.C”源文件和一个“.H”定义文件。

为了产生最终的可执行代码，这些文件必须被编译并与 ISaGRAF 目标库链接。有关实现上述操作的进一步信息请查阅“ISaGRAF 输入/输出开发工具箱用户指南”。

请注意： 当 ISaGRAF 应用程序用“C”语言编译时，有些诊断功能（比如下载应用程序、在线修改和断点等）已不存在。

#### A.13.4 浏览信息

“编辑”菜单包括用于浏览不同文本文件的命令，这些文件是在代码生成或检查语法操作时在代码生成器窗口里建立的。代码生成窗口是一个文本区，它包含在代码生成或检查语法操作时所产生的信息。所有信息都被存入磁盘，因此可以用“编辑”菜单命令来检查这些信息。

##### ■ **编辑命令**

“清屏”命令用于清除窗口的文本区。在每次代码生成或检查语法操作之前窗口被自动清屏。“拷贝”命令用于拷贝窗口剪切板中所显示的文本，因此它可供其它应用程序（比如 ISaGRAF 文本编辑器）使用。

##### ■ **浏览编译程序的输出信息**

“执行信息”命令把上一次“制作”或“校验”操作时产生的所有信息显示在窗口文本区里。该命令适用于所有的错误信息。

“编辑”菜单的其它选项允许用户监视在语法校验和代码生成时产生的附加文本文件。通常情况下，这些文件不能用于一个常用的 ISaGRAF 项目。

### A.13.5 定义资源

“选项”菜单的“资源”命令允许用户定义资源。一个资源是任一用户定义的任何格式（文件、数值表）的数据（网络配置、硬件的设置...），该数据必须与生成的代码合并，以便和生成的代码一起下载到目标 PLC 中。这样的数据不能由 ISaGRAF 内核直接操作，并且通常供安装在目标 PLC 上的其它软件使用。关于可获得的资源的进一步信息，请查阅你的硬件手册。

资源在一个“资源定义文件”中被定义，该文件与 ISaGRAF 项目的其它文件一起被保存。这是一个单纯的 ASCII 文本文件，由 ISaGRAF 的资源编译程序来处理。当建立应用程序代码时，该编译程序被自动地运行。本段落解释资源定义文件的语法。资源定义文件使用 ST 语言的词法规则。以“ ( \* ”符号开始，以“ \* ) ”符号结束的注释可被插入文本的任何地方。用单撇号限定字符串。关于输入数值时使用的词法格式的进一步解释请查阅本手册的第二部分。

#### 语言参考

以下是资源定义文件中使用的关键词和语句表。

**意义：** 给定一种资源，该资源是一个整型数值表。用无符号的 32 位整数目标代码存储数值。按照资源定义文件中规定的顺序存储数值。数值必须用逗号分开。资源的名称不能超过 15 个字符。

**语法：**     ULONGDATA '<resource\_name>'  
               BEGIN  
                   ...target\_selection...  
                   ...list of values...  
               END

**举例：**     ULongData 'MYDATA'  
               Begin  
                   ...  
                   0, -1, 100\_000,                 (\* 十进制 \*)  
                   16#A0B1, 2#1011\_0101         (\* 十六进制，二进制 \*)  
               End

## VARLIST

**意义：** 表示变量地址表的资源。在资源文件中，变量用名字标识。在目标代码中，以 16 位整数表示变量地址，并且按照资源文件中规定的顺序存储地址。变量必须用逗号分开，名字不能超过 15 个字符。

**语法：**     VARLIST '<resource\_name>'  
               BEGIN

```
...target_selection...  
...list of variable names...  
END
```

**举例:** VarList 'LIST'  
Begin  
...  
Var100, MyParameter, Command, Alarm  
End

## BINARYFILE

**意义:** 给定一种二进制文件资源。源数据被存储在一个MS-DOS文件中。使用一个目标路径名完成对目标资源的定义。行末端的字符不能被ISaGRAF资源编译程序转换。资源的名称不能超过15个字符。

**语法:** BINARYFILE '<resource\_name>'  
BEGIN  
...target selection...  
FROM '<source\_pathname>'  
TO '<destination\_pathname>'  
END

**举例:** BinaryFile 'MYFILE'  
Begin  
...  
From 'c:\user\config.bin'

```
To '/dd/user/appl/config.dat'
End
```

## TEXTFILE

**意义:** 给定一种文本文件资源。源数据被存储在一个 ASCII 文件中。使用一个目标路径名完成对目标资源的定义。根据目标主系统的习惯，行字符的末端由 ISaGRAF 资源编译程序来转换。资源的名称不能超过 15 个字符。

**语法:**

```
TEXTFILE '<resource_name>'
BEGIN
    ...target selection...
    FROM '<source_pathname>'
    TO '<destination_pathname>'
END
```

**举例:**

```
TextFile 'MYFILE'
Begin
    ...
    From 'c:\user\config.bin'
    To '/dd/user/appl/config.dat'
End
```

## TARGET

**意义:** 给定一个目标代码名称，该目标代码必须包含资源。关于所操作目标的进一步信息请查阅前面的章节(编译程序选项)。在同一个资源块中“目标”语句可以出现多次，以便选择几个目标。如果给定了“AnyTarget”语句，则不能使用该语句。

**语法:** TARGET '<target\_name>'

**举例:** BinaryFile 'MYFILE'  
Begin  
    Target 'ISA86M'  
    Target 'ISA68M'  
    ...  
End

## ANYTARGET

**意义:** 请确定：资源必须与所有的由代码生成器产生的目标代码合并在一起。ISaGRAF代码生成器在执行同一个“制作”命令时，可以产生几个目标代码。如果给定了一个或几个“目标”语句，则不能使用该语句。

**语法:** ANYTARGET

**举例:** ULongData 'MYDATA'  
Begin  
    AnyTarget

...

End

## FROM

**意义:** 给定一个 BinaryFile 或 TextFile 资源的源路径名 (在装有 ISaGRAF 工作平台的 PC 上)。用于分隔路径名部件 (驱动器, 目录, 字头, 字尾) 而使用的字符必须遵守 MS-DOS 系统的习惯。

**语法:** FROM '<target pathname>'

**举例:** BinaryFile 'MYFILE'

Begin

...

From 'c:\user\config.dat'

To '/dd/user/appl/config.dat'

End

## TO

**意义:** 给定一个 BinaryFile 或 TextFile 资源的目标路径名 (在目标系统上)。用于分隔路径名部件 (驱动器, 目录, 字头, 字尾) 而使用的字符必须遵守目标主系统的习惯。

**语法:** TO '<target pathname>'

**举例:**     TextFile 'MYFILE'  
               Begin  
                   ...  
                   From 'c:\user\config.dat'  
                   To '/dd/user/appl/config.dat'  
               End

## **举例**

以下是一个资源定义文件的完整例子：

(\* 资源定义文件 \*)

```

ULongData 'DATA1'          (* 数值表 *)
Begin
  Target 'ISA86M'           (* 仅用于该目标 *)
  1, 0, 16#1A2B3C4D, +1, -1 (* 数值 *)
End

```

```

VarList 'VLIST1'           (* 变量表 *)
Begin
  Target 'ISA86M'           (* 仅用于该目标 *)
  Valve1, StateX, Command, Alrm1 (* 变量名 *)
End

```

```

BinaryFile 'FILE1'         (* 二进制文件资源 *)
Begin
  AnyTarget                 (* 供所有目标使用 *)

```

```

From 'c:\user\updatef.bin'    (* 在 PC 上的源文件 *)
To 'updatef.cfg'              (* 在 PLC 上的目标文件 *)
End

TextFile 'FILE2'              (* 文本文件资源 *)
Begin
  Target 'ISA68M'
  From 'c:\nw\nwbd.txt'        (* 在 PC 上的源文件 *)
  To '/nw/dat/nwbd'            (* 在 PLC 上的目标文件 *)
End

```

## 编译资源

如果资源已被输入到资源定义文件中，则在 ISaGRAF 代码生成的末尾将出现一个对话框。按下“启动编译”按钮，以便运行资源编译程序。输出的信息和错误将被显示在主控制中。要想取消资源编译操作，请按下“退出”按钮。在这种情况下，资源将不被添加到 ISaGRAF 代码中。

## 实施

资源的数量、数据行和文件的大小不受 ISaGRAF 的限制。带有一个资源目录的资源被保存在已生成代码的末尾。以下是资源目录的格式(使用 C 符号表示法)：

```

__RESOURCE:
{
    long nbres;                /* 被定义的资源数量 */

```

```
{
    char name[16];           /* 资源的名称 */
    long type;               /* 资源的数据类型 */
    long size;               /* 数据块的准确大小 */
    void *data;
    char *path_offset;       /* 指向一个字符串 */
} /*nb of records */
}
```

以下是“类型”域中可能出现的数值：

1 = 二进制文件

2 = 文本文件

3 = ulong 数据 (在此情况下，不使用 path\_offset 域)

4 = 变量表 (在此情况下，不使用 path\_offset 域)

对于文本文件来说，按照目标系统的习惯，行字符的末尾是由资源编译程序来转换的。所有指针（标明数据项位置的一种标识符）与相关结构地址之间的偏移距离为 32 位。所有的资源名称和路径名是以空字符结尾的字符串。路径名和数据按资源目录来编排 Same as 3.2

## A.14 交叉引用

ISaGRAF 工作台提供交叉引用编辑器，以使用户从整体上查看在项目程序中声明的变量；以及它们在何处被使用。交叉引用的目的是列出在项目中的声明的所有变量，并且在程序源代码定位每个使用到它们的地方。交叉引用对于全面观察一个变量的生命期非常有用。它们有助于确定副作用，并能在维护项目时减少理解时间。交叉引用还被用于全面查看整个项目中的字典变量库，找出未使用的变量，从而项目的复杂性得以衡量。

当执行一个交叉引用,寻找变量时,通过键入名字来寻找指定的变量,然后点击 **OK** ,或点击 **All**来寻找所有变量.

左侧的列表内显示了在项目中引用的已声明对象（如程序变量和定义字），以及库元件（函数和功能块）。右侧显示了有关左侧列表中当前选择的对象的出现情况。

出现情况描述包括程序名，FC和SFC的步，转换和测试的序号，加上文本语言的行号或LD和FBD程序索引等。对于快捷LD程序，描述海包括支路的编号。如果变量作为输出（在线圈的上方），则支路编号后跟随有（"\*"）字符。

在“选项”菜单中设置“显示未用变量”选项能在主列表中列出应用程序中所有未用的变量。

 **对象类型选择**

由于一个项目可能包括数目巨大的已声明对象，编辑器工具栏提供了一个组合选择框以选择在查看中显示的对象类型，这使用户能有选择地访问信息。

每次重新计算交叉引用后，此选择都会恢复为“所有对象”以提供完整的列表。

 **重新计算交叉引用**

“文件/重新计算”命令任何时候都可用来在使用其它 ISaGRAF 编辑窗口修改程序后更新交叉引用。

 **输出交叉引用**

“工具/输出”命令将交叉引用的完整列表写入到 ASCII 文本文件中。此文可用其它应用程序，例如：Windows 笔记本或字处理程序打开。

 **字典错误**

“文件/字典错误”命令在对话框中，显示加载项目字典时检测到的错误列表。

 **统计**

执行“工具/统计”命令将在一个对话框中根据变量的类型和属性显示项目中已声明对象和变量的数目。此命令的一个特殊的用途是在使用 ISaGRAF 工作台的受

限制的版本时，得知在项目中声明的 I/O 变量数，以确保其能被编译。



### **查找对象列表**

“编辑 / 查找”命令允许用户在编辑器列表中直接选择对象。如果对象没有在列表中真正列出（如使用有选择的显示时），则可能找不到要查找的对象。建议在查找对象前，在工具栏上选择显示“全部”对象。



### **打开程序**

右侧的列表显示了所选择的对象在打开的项目的源代码文件和 I/O 连接出现的情况。“编辑 / 打开程序”命令允许用户直接打开其中出现所选对象的程序，也可以用在右侧列表中出现对象的项上双击鼠标的方法打开引用了对象的相应程序。

## A.15 图形诊断器的使用

ISaGRAF有一套完整的图形和符号诊断器。运行程序管理窗口中的“诊断”命令可以起动诊断器，并将应用程序下载到目标PLC中去。在这种模式下，诊断器通过硬件连接器件与目标系统进行通讯。程序管理窗口的“仿真”命令同时起动诊断器和完整的目标仿真器。这样就使得用户可以在目标输入/输出系统尚未运行完成时，测试自己的应用程序。诊断器窗口中有各种用来控制应用程序的命令。

在诊断器起动时，如果目标PLC中的应用程序与工作台中的应用程序相同，诊断器就自动打开调试模式下的程序管理窗口。这个窗口中的各项命令可以用来打开其它ISaGRAF窗口(如图形和文本编辑器、字典、变量表、输入/输出连接等窗口)。在调试期间打开的所有窗口都以调试模式工作，这意味着不能在此使用编辑命令。显示出来的程序部件(如程序步、转换、变量等)都是当前运行时的状态或数值。在目标应用程序中，双击一个对象，可以改变对象的状态或数值。

当诊断器以仿真模式运行时，诊断器将停止其与ISaGRAF目标系统的通讯。此时，诊断器仅与仿真窗口进行通讯。由于在这种模式下目标系统不存在，因此不能使用诊断器菜单上的“下载”、“停止”或“激活”命令。

## A.15.1 诊断器窗口

诊断器窗口中只包括有关完整应用程序状态的信息。此时，诊断器窗口被链接到另外的能生成一个完全的交互式调试系统的 ISaGRAF 窗口上。在诊断器窗口的底部显示检测到的运行时的错误。“选项”菜单中的各个命令可用来隐藏、显示或清除错误表。

控制面板(诊断器菜单下的区域)可显示目标应用程序的全部状态及有关运行周期内的信息。可能的目标状态列表如下：

登录:.....诊断器建立与目标系统之间的通讯。

断开:.....诊断器不能进行与目标系统之间的通讯，但保证电缆连接和通讯参数都是有效的。

没有应用程序:.....连接正确，但在目标系统中不存在的 ISaGRAF 应用程序。

应下载一个应用程序。

激活应用程序:.....连接正确，并且在目标系统中有可用的应用程序。如果这个应用程序与工作台中的应用程序相同，诊断器将立即建立与目标系统的通讯。

运行:.....目标应用程序处于“实时”模式。

停止:.....目标应用程序处于“循环到循环”模式。

断点:.....由于遇到断点，目标应用程序处于“循环到循环”模式。

致命错误:.....由于出现严重错误,目标应用程序停止。

运行周期中有关计时的信息如下:

允许的:.....程序运行周期。

当前:.....程序完整运行周期最后完成的确切计时。

最大值:.....自从应用程序起动后,检测到的最大计时。

溢出:.....检测到的计时大于允许计时的循环次数。

所有时间值的单位均是毫秒。当诊断器以仿真模式工作时,不显示时间值。

## A.15.2 应用程序的控制

在“文件”和“控制”菜单中,包含了在ISaGRAF目标系统上安装和控制当前编辑ISaGRAF应用程序的有关命令。

**注意:**由于由仿真器处理的应用程序是由ISaGRAF工作台自动安装的,所以这些命令中的一些命令在仿真过程中是不能使用的。



## **停止目标应用程序**

“文件/停止应用程序”命令将停止在当前 ISaGRAF 目标系统中启动的应用程序的运行。



## **启动目标应用程序**

“文件/启动应用程序”命令将启动已存在于目标系统中的应用程序。在下载一个应用程序后，这个应用程序就会自动启动，因此不需要使用“启动”命令。“启动”命令专门用在“停止”命令之后。

注意: 在可能要下载新的应用程序之前，必须停止（即不启动）目标应用程序的运行。



## **下载应用程序**

“文件/下载”命令用于向目标系统下载应用程序代码。应根据目标系统处理器和应用程序选择项，来选择要下载的代码的类型。



## **显示版本号**

“文件/获取版本号”命令用来显示工作台应用程序和目标应用程序的全部标识。工作台应用程序是指当前在 ISaGRAF 工作台打开的程序；目标应用程序是指在目标 ISaGRAF PLC 中运行的程序。这个命令显示下列内容：

版本：.....应用程序代码的版本号。这个版本号是由代码生成器计算而得的。

日期：..... 显示代码生成时的日期和时间。

CRC:..... 由符号表的内容计算而得的校验和。这个数值由代码生成器计算而得。数值取决于变量代码的数量。

注意：在仿真过程中也可使用“获取版本”号命令。实际调试模式中，如果没有连接目标 PLC，则不可使用这个命令。



### **在线修改**

“文件/更新应用程序”命令可以使得用户可以对正在运行的目标应用程序进行“在线修改”。这个命令将在以后有关章中详细介绍。但当诊断器以仿真模式工作时不可使用这个命令。



### **实时模式**

在没有应用程序运行时，不能使用“控制/实时”命令。这个命令可将目标应用程序设为通常的实时模式：即正常模式。程序的运行循环是由编入程序的循环计时来触发的。



### **循环到循环模式**

在没有应用程序运行时，不能使用“控制/循环到循环”命令。这个命令可将目标应用程序设为通常的循环到循环模式。在这个模式中，循环将根据由用户从

诊断器菜单中调用的“运行一个循环”命令，逐一顺序运行。



### **运行一个循环**

当目标应用程序处在循环到循环模式中时，“控制/运行一个循环”命令将启动一个循环的运行。



### **循环计时**

“控制/修改循环计时”命令可以让用户修改编入程序的循环计时。这个时间在诊断器控制条窗口中标为“允许”的。在修改程序循环计时之前，必须将系统置为“循环到循环”模式。循环计时应以毫秒为单位的整数进行输入。



### **删除所有断点**

“控制/删除所有断点”命令可以删除应用程序中现有的所有断点(包括遇到的或仍在活动的)。但在诊断器窗口关闭时,已有的断点不能自动删除。



### **解锁输入/输出变量**

“控制/解锁输入输出变量”命令可以释放所有当前应用程序中被锁定的输入/输出变量。当一个输入/输出变量被锁定时,与之相应的输入/输出设备的输入或输出状态不会发生改变。与输入/输出相连的各个变量可

以由应用程序或诊断器进行写操作。但在诊断器窗口关闭时，当前被锁的输入/输出变量不能自动释放。

### A.15.3 选项

“选项”菜单上有控制显示在诊断器窗口的信息的选项。

#### ☐ **通讯参数**

当诊断器运行时，可以调整通讯计时参数。但此处只能进行“超时”参数的设置。其他各个通讯参数(如波特率、奇偶校验等等)必须在程序管理器窗口的“诊断”菜单中设置。

参数“通讯超时”的数值是指留给目标系统开始应答工作台请求的时间，参数“循环刷新”的数值是指：为了在已打开的窗口中刷新数据，从诊断器送出的读请求到完成数据刷新所需要的时间。

所有的时间值都是以毫秒为单位的整数值进行显示和输入的。在诊断器处在仿真模式时，不能设置通讯计时参数。

#### ☐ **显示选项**

“显示循环时间”选项使得用户可以在诊断器控制条上隐藏或显示循环时间值。当设置这个选项时，所有的循环计时参数(如许可值、当前值、最大值、溢出值等)

都可以显示出来并且不断被刷新。不选择这个选项可以减少诊断器的通讯负担。

当设置“显示错误”选项时，被系统检测到的运行时的错误均被排列在诊断器窗口的底部。不选择这个选项时，系统将关闭错误表。删除这个选项，可以减少诊断器的显示和通讯上的负担。“选项/删除错误”命令用来清除当前显示在诊断器窗口的运行时间错误表。

“选项/缩小窗口”命令可以减小诊断器窗口的尺寸。因此诊断器窗口变成了一个较小的、位于屏幕顶部的画面，这个画面仅可显示应用程序的状态和表示大多数经常使用的命令的图形按钮。

## A.15.4 “写”命令

ISaGRAF 中的符号诊断器提供了许多命令，这些命令可以改变各种应用程序组件中的数值或状态。当诊断器窗口打开时，通过双击在编辑窗口中应用程序组件的名字或者图标，就可以改变对组件的选择。

### 变量

双击下列窗口中的变量名，可以改变变量的状态：

- 代码窗口
- 变量表或时间图表窗口
- LD 或 FBD 程序窗口

- 输入 / 输出连接窗口

调试对话框可提供下列命令：

- 将新值写入变量中
- 锁定变量 (仅用于输入/输出变量)
- 解锁变量 (仅用于被锁定的输入/输出变量)
- 启动或停止计时器变量 (设置自动刷新模式)

用于表示布尔变量“真”和“假”的各个符号是在代码中被定义为专用布尔变量的字符串。由“写”命令指定的模拟量必须根据该变量在代码中的定义，按整型或实型格式输入。由“写”命令指定的信息字符串的长度不能长于在编程代码中附加的变量长度。



### **SFC 对象**

在调试应用程序时，可以使用程序管理器窗口中“文件”菜单中的各个命令，来观察 SFC 程序的控制运行情况。被观察的 SFC 程序应从程序列表中选出。可用下列命令进行：

启动 SFC 程序:.....采用将 SFC 标记置于程序的每个初始程序步的方法，启动选定的程序。

停止 SFC 程序:.....采用清除全部已存在的标记的方法，停止选定的程序，

冻结 SFC 程序:.....清除选定程序中全部已存在的标记，并存储标记的位置。

重新启动 SFC 程序 .....采用更换已由“冻结”命令删除的标记的方法，重新启动一个已经冻结的程序。

对于子程序，这些命令与代码中的"GSTART"、"GKILL"、"GFREEZE"和"GRST"功能相对应。

在调试应用程序时，双击 SFC 编辑窗口中应用程序的图标，就可以在一个 SFC 程序步中看到控制操作的情况。下列命令可以在调试对话框中使用：

- 在激活步中设置一个断点
- 在非激活步中设置一个断点
- 清除加到程序步中的断点

注意： 激活和非激活断点不能加到同一程序步中。

在调试应用程序时，双击 SFC 编辑窗口中应用程序的图标，就可以在一个 SFC 转换，看到控制操作的情况。在调试对话框中，推荐使用下列命令：

- 在转换清除程序步内增加一个断点
- 清除加到转换清除程序步内的断点
- 手动清除转换程序步（如移动或增加标记）

条件的清除：在转换程序步之后的各个程序步上建立一个标记，删除以前程序步中存在的标记。无条件清除：在转换程序步之后的各个程序步上建立一个标记，不删除以前程序步中存在的标记。

### A.15.5 显示锁定状态和设备值

当模拟计算机 I/Os 被锁定时,workbench 读取 "锁定" 状态和实际设备值。实际设备值包括受力值 I/Os 的"锁定"状态被显示在下列编辑器中:

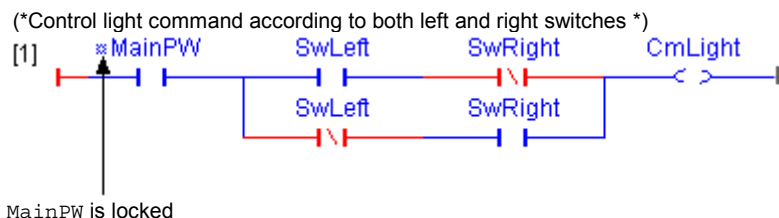
FBD 编辑器

LD 编辑器 (Quick LD)

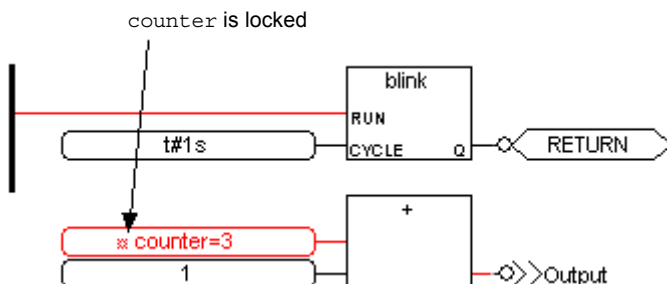
字典

变量清单 (密探清单)

在 FBD 和 LD 编辑器中,当一个 I/O 被锁定时,它的名字会被标上 "※" 符号。下列 LD 例子显示主电源 I/O 被标上锁定符号:

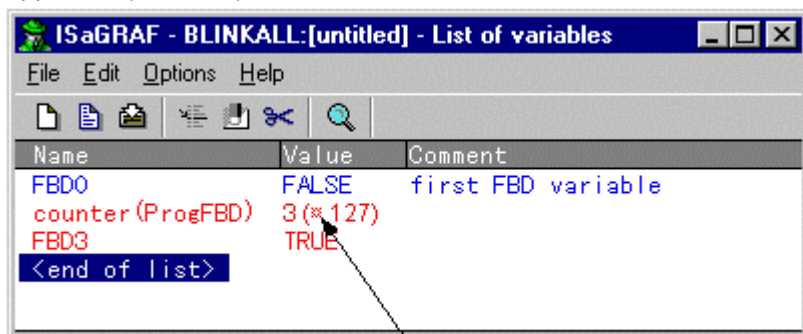


下列 FBD 例子显示计算器 I/O 被标上锁定符号:



在字典中,及变量清单(密探清单), 除显示“锁定”状态,以及显示应用 I/O, 实际设备值也将被显示出来。使用下列句法来显示这信息。

AppValue (x 设备值)。



Counter is locked and forced to 127

## A.15.6 在线修改

"在线修改"特性可以使得用户在进程运行期间修改应用程序。"在线修改"特性对于中断进程运行,可能会危及产品或安全的化学处理过程,有时是很必要的。但使用这项功能时需要非常小心。因为 ISaGRAF 可能检测不到全部可能发生的冲突,而这些冲突正是由用户自己定义的运算过程,而产生的在线修改的结果。

### 代码序列

由于 ISaGRAF 系统为诊断器存取各个变量、程序或输入/输出板中的数据提供了许多种可能,所以这里所描述的“在线修改”功能仅适用于代码序列的修改。代码序列是一连串 ST 代码、IL 代码、LD 代码或 FBD 代码指令的全集。在“循环开始”或“循环结束”程序之间,代码

序列是指写入程序中的全部指令。在 SFC 程序中，代码序列是指一个程序步或转换的二级编程代码。“在线修改”就是在不停止 PLC 运行的情况下，置换一个或多个代码序列。由于控制 SFC 标记是非常关键的，所以修改 SFC 程序的结构，对程序步、转换或 SFC 程序进行添加、改编号或删除等操作，都是不可能的。

## 变量

由于变量数据库是应用程序的一个非常关键的部分，所以其他进程(在多任务 PLC 中)可以随时对变量数据库进行存取操作。用诊断器修改变量值也是可能的。因此，ISaGRAF 不允许用户以在线的方式对变量进行添加、重命名或删除操作。然而，在应用程序中，总有一种可以修改变量的方法。因此应保留应用程序的第一版本中“未使用”过的内部变量或输入/输出变量，以便在将来的修改中可以使用它们。

在 ISaGRAF 目标数据库中有不同种类的变量，对各个变量的限制说明如下：

### - 已声明的变量

已声明的变量是采用 ISaGRAF 字典声明了的变量。不能使用在线修改的方式对这些变量进行改动或重命名操作。建议在应用程序中，对一些附加变量进行声明和初始化，即使现在并没有使用这些变量。这些附加变量可以在不改变应用程序的数据和校验情况下，进行远期的修改。

### - 功能块的实例

每一个由C语言或IEC代码编写的功能块的实例都与存储在ISaGRAF目标实时数据库中的数据相对应。当增加或删除功能块的实例时，使用在线修改是不可能的。因此在快捷LD或FBD图中，较好的ST运行方法是使用在字典中声明了的FB实例，而不是添加功能块（添加就与新的自动声明的实例相一致了）。另外，在ISaGRAF库中的可用功能块定义的修改，将导致不能进行在线修改。

### - 程序步

每一个SFC程序步都与一条存储SFC程序步的动态特性（活动时间和标志）数据对应。增加或删除SFC程序步会改变应用程序的数据库，不得进行在线修改。

### - 由编译器分配的隐含变量

ISaGRAF编译器可以生成“隐含”的临时变量，以便对复杂表达式进行求解。在有些情况下，表达式的改变会产生一个不可见的临时变量的异集，并导致不能进行在线修改。为了避免这种情况的发生，可在ISA.INI文件中增加下列内容，强迫分配给每个程序的临时变量的数量达到最小，即使没有使用应用程序的第一个编辑版本。这里给出的值是例子：

[DEBUG]

MNTVboo=8 ;用于布尔型变量

MNTVana=4 ;用于整型和实型变量

MNTVtmr=4 ;用于计时器变量

MNTVmsg=2 ;用于信息型变量

在 ISA.INI 文件中设置这些内容后，当对应用程序进行重新编译，导致编译器分配的临时变量的数量较大时，编辑器会输出一条警告信息。

## 输入和输出

由于 ISaGRAF 输入/输出系统是一个非常开放的系统，所以它可以使用相关硬件的专用特性，由 OEM 完成所需要的修改。ISaGRAF 系统不允许用户对输入/输出变量进行在线增加、连接或删除；也不允许用户对输入/输出插板的描述进行在线修改。可以用标准 OEM 特性和 OPERATE 功能完成诸如修改输入/输出插板参数和锁定输入/输出通道这样的操作。

## 执行时操作

修改正在运行的应用程序应包括以下操作：

- 在工作台上修改应用程序的源代码
- 生成新的应用程序代码
- 使用上载命令而不是下载命令下载新的应用程序代码

- 使用实际更新命令在 PLC 执行周期内，完成从旧的应用程序到新的应用程序的转换。

这个过程可以保证目标 PLC 上总有一个完整而可靠的正在运行的应用程序，并且使得用户可以以一种非常安全和有效的方式控制采样计时的操作。另外，还可以使用户在每当可能的时候，修改项目程序。无论怎样，“在线修改”与普通的停止、启动和下载命令集从本质上说是一致的。不同点在于“在线修改”没有变量状态的丢失，并且切换时间非常短（通常只有 1 或 2 个周期）。在切换期间，不修改变量，在应用程序修改前后，所有内部变量、输入或输出变量保持原来的数值。在切换期间，系统不发生动作，也不移动 SFC 的标记。

## 内存要求

为了支持“在线修改”特性，目标 PLC 中必须有空闲内存，以存储应用程序代码修改以后的版本。在切换操作过程中，应用程序代码的新旧两个版本都存储在 PLC 内存内。

## 限制因素

如前边所描述的那样，只允许修改代码序列。变量定义、应用程序参数和输入/输出的连接不能进行修改。在下载应用程序的修改版本时，为了对一些不安全的修改处进行检测，ISaGRAF 会在修改过的应用程序和正在运行的应用程序之间进行比较。假如某处修改看起

来可能是危险的或不可能实现的，就会产生一个下载错误。由ISaGRAF完成的安全措施之一比较符号表的校验和，为的是检测变量、程序或SFC元素名的变化。如果在切换时，有一个程序步处于激活状态，步中没有存贮的作用将丢失。也不执行新程序步中的动作。当切换时，如果转换有效，可以进行程序修改。新下载的应用程序代码不会备份到PLC上。备份的是以前用标准下载命令下载的程序版本。



## 操作

为了更新正在运行的应用程序代码，必须完成下列操作：

- 郑重建议：在对正在运行的应用程序进行修改之前，应对当前项目程序，用其他名字作一份备份，在备份上进行修改。
- 在编辑一个程序之前，用户应该检查编辑工具中的“更新日志”选项已经设置，以使得以后的程序维护比较容易。
- 当修改一个或多个代码序列（没有修改SFC的结构和程序层次）时，在下载之前，必须在工作台上生成这个新应用程序的代码。
- 使用旧项目中的诊断器时，用户必须连接到目标PLC上，并完成任何使应用程序的修改更快或更安全的操作。
- 使用新项目中的诊断器时，用户必须连接到目标PLC上。如果应用程序名被修改，则该应用程序不能对目

标数据库进行存取。用户必须运行“文件 / 更新”命令。

- 选择日后更新选项，可以下载修改过的应用程序。但这样向 PLC 传输时，速度稍慢。

- 当下载完成时，用户可以运行“文件 / 实现更新”命令，在最适当的时刻进行切换。切换将持续 1 或 2 个周期。

- 当切换正常完成时，会显示被修改过的的应用程序。如果没有正常完成，则现存的的应用程序按原样保留。

### A.15.7 动态数据交换 ( DDE )

ISaGRAF 诊断器中包含一个 DDE ( 动态数据交换 ) 服务器。为了能够动态地显示非 ISaGRAF 应用程序中变量的当前值，可以在 ISaGRAF 诊断器和其他应用程序之间设置 advise 循环。

ISaGRAF 诊断器 DDE 服务器仅支持“报告”和“发送”业务。只有在 advise 循环中监视到的变量，才可以使用“请求”业务。其它 DDE 的服务如“执行”是不能用的。当对一个变量建立 advise 循环时，在每次客户修改应用程序时，这个变量值都是可以修改的。advise 循环可以监视不同类型的变量。动态连接的标识包括下列名字：

Service name: "ISaGRAF"

Topic name: ISaGRAF 项目名

Item name: 变量名

如果变量对一个程序来说是局部变量，则它的名字必须在其父程序名后，其父程序名写在括号内。语法如下：

`variable_name(program_name)`

ISaGRAF 诊断器 DDE 服务器由诊断器目前监视到的 ISaGRAF 应用程序专用。这个 DDE 服务器最多可以监视 256 个变量。当 ISaGRAF 诊断器以连接模式或仿真模式工作时，可以使用 DDE 服务器。刷新周期就是建立诊断器和 ISaGRAF 目标系统或仿真器之间通讯所用的时间。

## A.16 监视变量表

调试器窗口中“监视”菜单的“监视列表”命令，允许用户建立一个不连续的变量的列表，并且不断用变量的当前值刷新显示。列表在调试应用程序时建立，可以被保存到磁盘，并在其它调试会话中重新打开。一个列表最多可有32个变量，不同类型的变量可以在同一列表中混合显示，全局和局部变量都可以插入到列表中。一个变量列表值用于一个特定的项目。变量列表对于应用程序功能测试非常有用。它允许用户独立于应用程序源代码，监视控制过程中局部的变化。变量列表对调试ST和IL文本程序也很有用，用户可以轻易地在列表中组织用于一个程序的变量，以控制和监视程序指令的执行。

ISaGRAF 显示每个变量的名称，当前值和注释文本。可以用鼠标拖动列表标题栏中的分隔线来改变列表的列宽。

### **保存列表到硬盘**

“文件”菜单中的命令用来建立，打开和保存变量列表。ISaGRAF 不限制一个项目内的列表数目。当为保存到磁盘而命名变量列表时，必须遵守下列规则：

- 名称不能超过8个字符长
- 第一个字符必须是字母
- 随后的字符必须是字母

- 列表名称与大小写无关

列表编辑器不能在同一个窗口中显示一个以上的变量列表，但它可以被运行多次（同时打开几个）以同时监视不同的列表。



### **插入变量到列表**

“编辑/插入”命令将插入另一个变量到列表。变量名可以从项目变量库中的对象列表中选择，这样就不必由用户手工输入变量的标识符。变量将在当前选择的变量之前插入。列表内不能有超过32个变量，每个变量不能出现多于一次。



### **改变所选变量**

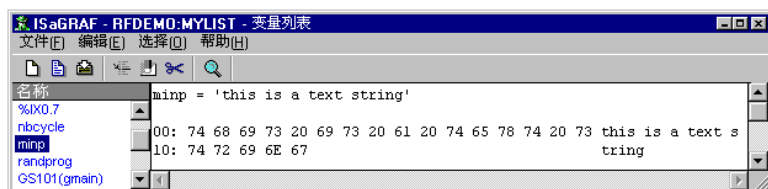
“编辑/修改”命令用来将一个变量替换为另一个。您也能使用“剪切”命令从列表中清除所选择的变量。



### **映射显示**

任何时刻，您都可以在列表和“映射”两种查看模式下切换。也可使用工具栏上的“缩放”按钮或使用“选项/映射”命令切换查看模式。

“映射”模式下只能显示一个变量。它的值将以数字或符号格式在查看上部显示，同时二进制“映射”格式显示。此模式允许您监视变量值的每个字节的十六进制值。



“映射”模式对于监视和理解包含不可打印字符的信息串非常有用。

## A.17 调试 ST 和 IL 程序

在仿真或在线调试 ST 和 IL 程序时，不可能对程序的文本进行修改。

IL 对于 IL 程序，指令在一个表的视图被格式化。指令所使用的变量的当前值显示在同一行上。你可以双击一个指令，以便改变对应变量的数值。

ST 对于 ST 程序，一个监视列表窗口被嵌入到编辑窗口中。用鼠标拖动窗口之间的分隔线可改变视图的大小。

对于表中的每个变量，ISaGRAF 显示变量的名称，变量的当前值和变量的注释文本。用鼠标在表的标题栏中拖动分隔线可改变列的大小。

### **把表保存到硬盘上**

用“文件/保存表”命令把变量表保存到硬盘上，所用的文件名与已编辑的程序名相同。在调试模式下每次打开 ST 或 IL 程序时，该表都自动地重新加载。也可以用监视表工具自由地打开和修改这个表，这里所说的监视表工具是用调试器窗口中的“监视/监视表”命令来运行的。



### **把变量插入到表中**



用“编辑/插入变量”命令把另一个变量插入到表中。从在项目字典里定义的对象表中选择变量名。这一方法使用户不必用手动的方法输入标识符。该变量被插入到表中当前所选变量的前面。变量表中所含有的变量不能多于32个。在同一个表中，同一个变量只能出现一次。



在ST文本中，当一个变量名被高亮显示时，按工具栏中的这个按钮或运行“编辑/监视选择”命令，以便把变量直接发送到已被嵌入的监视表中。



### **改变被选择的变量**

“编辑/改变变量”命令是用另一个变量替换被选择的变量。你也可以使用“剪切变量”命令从表中删除被选择的变量。

## A.18 注视点

ISaGRAF 注视点工具允许用户定义监视列表，在调试时以示意图或列表形式显示。图形项必须与 ISaGRAF 项目的变量相链接。图形总是“在线”定义和演示。

要强制变量的值，用鼠标在图形或列表中双击相应项，或选择它后按 ENTER 键。

您也可以使用“文件/锁定”命令锁住文档（不能进行修改）。锁定文档后，仍然可以用鼠标双击变量符号来强制变量。

### A.18.1 构造图形布局

图表由背景图片（位图或图元文件），以及在调试时活动显示的图形项构成。要输入图表，必须进行下列操作：插入背景图，插入图形项，链接对象到项目变量。



#### **背景图片**

背景图片可以是“位图”（.BMP）或图元（.WMF）文件。在图形布局中的图片数目不限，并可以移动或改变大小。它们不能在列表中显示。图片由其它工具生成，注视点中不包括绘图工具。“选项/背景颜色”命令可用来选择图形空区的填充颜色。

**注意：**位图要消耗大量的内存。请正确设定图形的大小，限制图形框内未使用区域的大小。

123

## 纯文本显示

“纯文本”项就是在矩形框中显示的文本。文本显示相链接的变量的值，这样的项可与信息字符串变量相链接。

显示文本的矩形框可以为颜色所填充或透明。显示文本使用的字体会在改变矩形大小时根据高度调整。

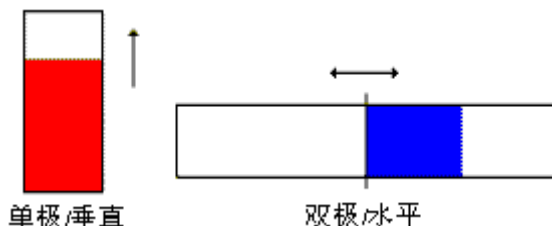


## 单向和双极条形图

条形图就是一个矩形，其中的着色部分表示相关变量的数值，矩形的其余部分可随意着色。条形图可以是水平的，也可以是垂直的。

单向条形图可以向任何方向增长：向上，向下，向左或向右。

双极条形图可以根据联系的变量值向正或负方向增长。对于双极条形图，正向和负向具有相同的最大允许值刻度。

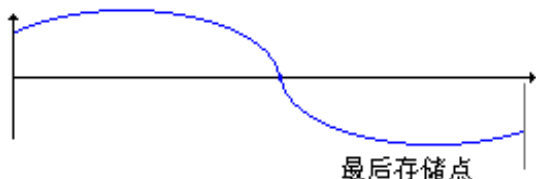




## 曲线

可在文档中插入曲线。曲线显示相关连变量的变化历史。虽然曲线显示不是很精确的度量工具，但它在调试时，可以提供不同变量的变化信息。

曲线可以存储变量的最后 200 个值。改变曲线在图形布局中的显示大小时不改变其采样数目。



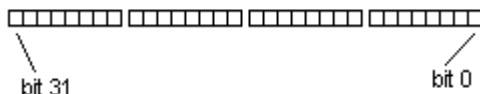
## 布尔图标

布尔图标用于显示二进制状态。一个图标（.ICO）文件被定义为“假”或 0 值；另一个图标定义为其它非零的值。由于注视点不包括图标编辑器，图标文件必须由其它工具预先准备好。



## 位字段

“位字段”以一个 32 位整型值的图形面板的形式显示。最低有效位总是显示在右边。因为可能引起显示信息的混乱，所以不推荐对其它类型数据使用位字段形式，例如实型模拟量。



### **选择，移动或改变项的大小**

大多数布局命令都需要选择图形对象。注视点允许在图表区域内选定一个或多个已存在的对象。选择时，必须先在编辑器工具栏上选用“选择”（有箭头的按钮）。选择一个对象时，用户只须用鼠标单击它的符号；选择多个对象时，要用鼠标在绘图区上拖动出一个矩形，所有与矩形相交的图形对象都被标记为“已选择”。选定的对象外围绕有黑色小方块。

进行新选择时，对上次选择对象的选择也就取消了。也可以用鼠标单击选择区域外的空白区取消选定。

为移动对象，必须先选定它们。然后用鼠标箭头在选择区边沿，拖动它们到其它位置。

为改变对象的大小，必须先选定它们，然后将鼠标箭头放在选择区边沿的小矩形块上，沿适当的方向拖动以改变其大小。图片也可以改变其大小，这种情况下相应的位图或图元图形将按指定的矩形拉伸。



### **项目组合/分解组合**

您可以把多个项组合在一起，从而把它们当作一个项处理。为此可在图中选定图形项，并使用“编辑/组合”命令。“编辑/分解”命令可以将它们恢复到未组合前的状态。

一个组合内可以有图片，组合内也可以包括另一个组合。

项被组合后，它们的样式就不能改变了。组合后的项仍然显示，但不能被用来（双击鼠标）修改相关连变量的值。

一个组合项在列表布局中以一行显示。

## A.18.2 列表布局



任何时刻都可用此按钮在图形和列表布局之间切换。也可使用“选项/列表-图形布局”命令。

在列表布局下，项显示在一个典型的列表框中，每个项的高度根据其绘制样式而定。图片（位图和图元文件）在列表布局中不可见。可用在列表中选择以设置项目样式或改变变量的值。此模式下不提供多重选择和命令。



您可用“编辑/在列表中移动”命令，在列表中对项重新排序，移动前必须先选择项。

## A.18.3 定义项样式

可以在图形区内用鼠标双击项的符号，或在图中选定它，然后使用“编辑/设置样式”命令以设置其样式。在

文档中添加新项目时，会打开“样式”对话框，其中提供了可供用户选择的样式：

图形样式与设置:

可以动态地改变项的显示样式（纯文本，条形图，曲线...）。如果使用了前景和背景颜色，您也可以通过相应的框加以定制。选用“布尔图标”样式时，必须指定相关的 .ICO 文件的路径。使用“ ”按钮可以浏览磁盘上已存在的图标文件。

刻度:

这是在条形图和曲线中显示的最大值。对于双极条形图，正向轴和负向轴都使用相同的刻度绝对值。

变量名:

当“名称”域激活时，按“ ”按钮可以在项目变量库中选定一个已声明的变量。

标题：


标题可以显示在图形项的旁边。您可以定制标题的显示位置（上，下，左或右）及其内容。标题可以由变量名和它的格式化为文本的值任意组合而成。标题的定制对列表布局无影响。


命令变量:


如果选择了“命令变量”选项，用户就可以在调试时，双击图形符号以修改相链接变量的值。

#### A.18.4 “文件”菜单命令

“文件”菜单中的命令使用户可以管理整个文档。

 “文件”菜单的“新建”命令能开始编辑一个新文档。ISaGRAF 不限制为项目定义的文档数目。在编辑新图表之前，必须关闭先前打开的图表，注视点不能一次编辑几个图表，但可以同时打开多个注视点窗口以分别编辑不同的文档。

 “文件”菜单的“打开”命令使用户可以关闭当前编辑的文档并开始编辑当前项目的另一个文档，新选择的文档将在编辑窗口中取代当前的文档。“删除”命令可用来删除已存在的文件以清理项目目录。删除图表时并不在磁盘上删除图表中引用的图标和图元文件。

 “文件”菜单的“保存”命令，将当前编辑的文档保存到磁盘。如果它是一个未命名的新文档，在保存前必须为其命名。命名文档必须遵照下列规则：

- 名称的长度不能超过8个字符
- 第一个字符必须为字母
- 随后的字符必须是字母，数字或下划线符号
- 名称与大小写无关

“文件”菜单的“另存为”目录允许用户以另一个名称保存当前编辑的文档。

### A.18.5 ISaGRAF V3.2 用户注意事项

注视点能读取 ISaGRAF V3.0 和 V3.2 建立的图形和列表时序图。这些文件将在“打开”对话框内与其描述一同出现，并可由注视点读取和修改。

当打开 ISaGRAF V3.2 图形时，文档被自动标记为“锁定”。如果您想修改它，请从“文件”菜单中去掉“锁定”选项。

打开 ISaGRAF 3.2 图形或列表时序图时，注视点总是试图以本地的注视点格式保存它，在关闭这些文档时，将系统地出现“另存为”对话框。

## A.19 上 载

ISaGRAF 支持存储于目标中的上载应用程序。上载过程就是与目标通信，装载内嵌压缩源代码（EZS）并将其在工作台环境中恢复。

如果目标程序版本为 V3.22 或更新，而且压缩源代码已嵌入到应用程序中，连接的 PLC 系统中运行的项目就可以上载。上载压缩源代码是可选特性。

### A.19.1 上 载 一 个 项 目

运行 ISaGRAF 项目管理器的“文件”菜单中的命令，打开“上载”对话框。上载不是指已存在于工作台上的项目。当前在项目管理器列表中选择的项目与上载机制无关。为上载在目标 PLC 中运行的应用程序，您必须：

- 1- 确保与目标 PLC 已正确地连接
- 2- 根据连接设置通讯参数
- 3- 按“运行”按钮

上载内嵌压缩源代码（EZS）和对它们解压缩需要一些时间。对话框中的消息将告诉您上载何时结束，以及是否出现错误。

通讯时在目标 PLC 中读取的名称就是建立 ISaGRAF 项目时用的名称。如果使用此名称的项目已在工作台存

在，将提示您是否覆盖这一项目或选择一个未用的名称。您不能在上载结束后，取消为已上载项目的源代码的注册。现在已上载的项目已经就绪，可以打开了。

## 可能的错误

下列错误可能在上载项目时发生。错误发生时，在“上载”对话框中将显示消息通知您。

- 不能与目标 PLC 建立通讯
- 连接目标 PLC 是 3 22 版本以前的 ISaGRAF 系统
- 在目标 PLC 中无应用程序运行
- 在目标 PLC 中无 EZS 嵌入。

## A.19.2 通讯设置

按“设置”按钮允许用户定义，连接 ISaGRAF 工作台和 ISaGRAF 目标 PLC 系统的上载通讯参数。在进行上载之前，必须确认配置的参数与连接目标 PLC 相匹配。

## A.19.3 准备上载项目

如果您想能在晚些时候进行上载操作，就必须告诉 ISaGRAF 代码生成器，将压缩源代码嵌入到应用程序代码中。为此，可在“编译器选项”对话框中按“上载”按钮，另一个对话框允许您选择压缩源代码的嵌

入。在这种情况下，只嵌入最低需求的源文件。使用其它选择框以嵌入可选文件 **重要注意事项**：库不与嵌入源代码一起下载。这包括了功能，功能块，以及 I/O 板和设备。

为更好地理解上载机制，请参考以下主题：

## □ **可选文件**

作为最低需求的源代码的补充，下列文件也可以嵌入。这将导致对目标 PLC 的额外的内存需求。

### 项目描述

如果未被嵌入，上载后的项目描述只指出上载的日期。

### 口令保护

上载功能并不被口令保护。如果您想上载项目保护，就需要嵌入源代码的口令保护系统。

### 未连接 I/O 通道注释

ISaGRAF 允许为未连接的 I/O 通道添加注释文本。如果您总是用已连接的 I/O 通道工作，请不要选择此选项。

### 修改历史

这是一个项目修改的全面历史记录。

### 日志文件

每个注释文件包含有关程序的用户书写的注释和编译器输出的消息。嵌入日志文件可能消耗目标 PLC 的大量内存。

### 变量列表和时间图

这些是调试中建立的文件，包括用于列表和时间图监视的变量名列表。

图形，图标和位图

这包括 ISaGRAF 图形，加上所有的图标和位图文件，如果它们在项目目录中存在的话。警告：这将大量消耗目标 PLC 的内存。

#### **A.19.4 在目标 PLC 中存储压缩源代码**

内嵌压缩源代码 (EZS) 就是将资源与生成代码在一起，生成的资源被称为“EZS”。如果选择了源代码内嵌，就不能为其它资源选择同一名称。内嵌源代码并不意味着在资源定义方面有任何限制，源代码内嵌不影响用户编写的资源定义文件。

请参考 ISaGRAF 说明文档中代码生成器的有关资源的详细资料。

#### **A.19.5 目标 PLC 的内存需求**

内嵌压缩源代码 (EZS) 需要额外的内存以在目标 PLC 中与应用程序代码一起存储。一个通常的粗略估计是，最小的 EZS (未选择代码内嵌的其它选项) 的大小是可执行代码的一点五倍。这意味着嵌入 EZS 需要把下载代码的大小乘以 2.5 倍。

对于一些内存分段的目标系统可能有特殊的限制。由于EZS将资源与生成代码一同存储，它们必须在同一个数据段内作为应用程序代码存储。

### A.19.6 关于已上载的项目

上载的项目包括了重新编译所需的所有文件和数据。根据先前编译的选项不同，它也可以包括一些附加文件如项目描述和程序日志文件。

您必须在调试或监视它之前编译（生成）项目。警告：由于ISaGRAF使用编译时间标签来比较程序的版本，当工作台和目标应用程序有不同版本号时，您将在打开调试器时得到通知。

**重要注意事项：** 嵌入源代码时并不下载库。您必须确保在重新编译上载的应用程序之前，在ISaGRAF工作台已安装了适当的库功能和功能块。

### A.19.7 兼容性问题

上载为ISaGRAF目标PLC和工作台版本3.22或以上所支持。为此通讯协议已进行了扩展。

由于EZS以标准的资源形式存储应用程序中，在将压缩源代码（EZS）嵌入到基于ISaGRAF系统版本3.03到3.21的目标时，没有特别限制。但是在这种情况下，

因为目标 PLC 不支持所必须的通讯服务，嵌入的信息不能上载。

## A.20 使用调试工具

"诊断工具"是 ISaGRAF 诊断器工具下的一个子集，为了进行检查和控制处理过程，这个子集使最终用户在预先定义过的变量集上工作。ISaGRAF 诊断器是一个包含多种高级功能的强有力的工具。诊断工具提供了一种安全的方法，来控制目标应用程序的最终运行或维护。ISaGRAF 诊断工具可以直接从程序管理器的 ISaGRAF 组中双击下列图标来调用：



调试

在一个对话框中显示出已存在项目的一览表。这个对话框可允许用户在已存在，并已下载的 ISaGRAF 应用程序上运行有限的 ISaGRAF 诊断器。按下"确认"按钮即可在选定的项目上运行有限的诊断器。按下"取消"按钮关闭对话框。"安装"命令用来安装 ISaGRAF 工作台和目标 PLC 之间的通讯连接。请参考本手册的"管理程序"一节，就可以了解到有关本命令更多情况。

**注意** ISaGRAF 诊断工具(有限的诊断器)不能用于对在目标 PLC 上运行的应用程序进行下载，停止或修改。如果在诊断工具对话框中选定的项目与安装在 PLC 上运行的项目不同时，系统不会进行任何操作。

有限的 ISaGRAF 诊断器运行且正确地连接到目标应用程序时，可以使用下列命令：

- 监视变量一览表

用聚光灯监视图形文件

## A.21 使用 ISaGRAF 仿真器

ISaGRAF 内核调试器是在程序管理器窗口中使用“调试”菜单的“仿真”命令而与调试器一同启动的。此内核仿真器是完全的 ISaGRAF 目标系统，支持 ISaGRAF 标准特性和所有由 **ICS Triplex ISaGRAF Inc.** 发布的标准库中的“C”函数以及功能块。I/O 板也在窗口中以图形方式仿真，支持任何类型的 I/O 板仿真。在 I/O 连接中定义的“虚拟板”也能在仿真窗口中显示。


### A.21.1 与调试器连接


内核仿真器支持与 ISaGRAF 调试器进行全面的通讯，仿真中可以使用所有调试手段。内核仿真器总运行在当前的 ISaGRAF 应用程序上。在仿真过程中，调试器的“开始”，“停止”，“下载”或“更新”命令不再有效。如果生成目标代码时未在编译器选项中选定“仿真”，则仿真器不能使用。关闭仿真器窗口意味着调试器窗口（以及在调试会话中打开的任何窗口）都将同时关闭。


### A.21.2 I/O 仿真


I/O 板将在窗口中显示，窗口的标题栏内显示它们的名称和插槽号，并支持处理任何 ISaGRAF 标准 I/O 类型（布


尔，模拟或信息)。输入板的通道与特殊的按钮和数据区一同显示。输出板的旁边显示图形表示的状态灯和数据区。

 布尔输入：布尔输入表示为方形的绿色按钮，I/O按钮的旁边显示通道号。当按下按钮时输入值为“真”，用鼠标在单击按钮可以改变相应的输入值。当按钮按下时可以用鼠标右键设置输入。

 布尔输出：布尔输出表示为一个小圆圈，同时显示其通道号。图形符号高亮显示时，代表输出值为“真”。

 模拟输入：模拟输入通道是单一的数字区，可以在其中输入相应的值。用鼠标单击显示插入符号的文本框就可以输入通道的新值了。输入之后不必按回车键。模拟值可以十进制或十六进制输入。用上/下按钮可以增加和减少当前值。

 模拟输出：模拟输出通道为是一个数字输出区。输出值可以十进制或十六进制显示。用户不能在输出区内进行任何操作。

 信息输入：信息输入通道是一个单一的文本区，用于输入相应的信息值。用鼠标单击显示插入标记的文本框就可以输入通道的新值了。输入之后不必按回车键。

xyz → 信息输出：信息输出通道是文本输出区，用户不能对输出通道进行任何操作。

### A.21.3 库组件

ISaGRAF 仿真器全面支持由**ICS Triplex** ISaGRAF Inc. 发布的标准变换，函数和功能块。下列为支持的对象列表：

▣ **变换函数:**

bcd, scale

▣ **函数:**

abs, acos, ArCreate, ArRead, ArWrite, ascii, asin, atan, char, cos, delete, expt, find, insert, left, limit, log, max, mid, min, mlen, mod, mux4, mux8, odd, rand, replace, right, rol, ror, sel, shl, shr, sin, sqrt, tan, trunc

▣ **功能块:**

average, blink, cmp, ctd, ctu, ctud, derivate, f\_trig, hyster, integral, lim\_almr, r\_trig, rs, sema, sr, stackint, tof, ton, tp

用户定义的变换，“C”函数和功能块通常不集成于ISaGRAF 仿真器内。一般地这些对象被设计为使用目标系统的软件和硬件资源，此类资源通常不为Windows系统所提供。ISaGRAF 仿真器为用户定义的变换，功能和功能块提供了以下标准行为：

- 当一个新的变换由仿真器处理时，它将被一个“空”变换所取代。这意味着模拟变量的物理值等于其电气值（正如在仿真器面板上输入和显示的）。
- 当仿真器运行新的“C”函数和功能块时，它不进行任何操作，也不设置结果值。

#### A.21.4 选项

“选项”菜单中的命令允许用户控制 I/O 信号在仿真器面板上的显示。用户可在调试过程中任意设置或取消这些选项。

▣ 当选中“彩色显示”选项时，I/O 通道可以彩色位图的形式显示。如果在一些 LCD 显示屏上不能辨明色彩，用户须取消这一选项以得到 I/O 通道的纯黑白输入输出图形显示。

▣ 当选中了“变量名”选项后，将在 I/O 通道的旁边显示所连接的 I/O 变量名的标记。取消这一选项可使用户减小仿真器面板的尺寸。

▣ 当选择了“十六进制值”时，任何输入输出模拟通道将以十六进制格式显示或输入。

▣ 当选中“总是在前”选项时，仿真器窗口将总是可见，即便当前的输入在另一个窗口进行。

#### A.21.5 保存和恢复输入状态

使用 ISaGRAF 仿真器，可以用仿真器面板上的切换按钮和编辑控制工具手工强制输入通道。您可以在任何时刻使用“工具”菜单中的命令保存和恢复所有的输入状态：

装载输入方案按照使用“保存输入方案”在磁盘上保存的文件中的输入通道值设置输入值。

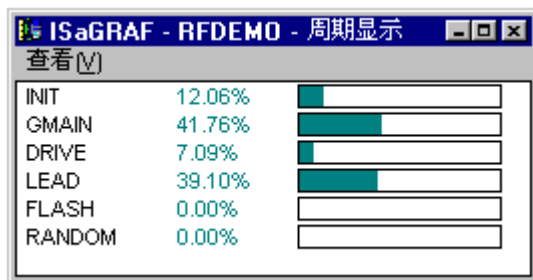
保存输入方案保存输入通道的状态以便以后通过使用“装载输入方案”命令重新恢复。文件存储于工程目录中，从而可以与其它项目文件一同用ISaGRAF档案工具保存。

注意：只有命名后的输入通道（连接了变量的）才被保存到磁盘。

### A.21.6 循环周期配置器

ISaGRAF循环周期配置器是显示分配在不同的程序、函数和功能块中的循环时间的诊断工具。该工具对快捷诊断应用程序的执行性能非常有用，它可将程序员引向需要优化的代码部分。

循环周期配置器可从ISaGRAF仿真器窗口的“工具/循环配置器”命令启动。它显示每个程序，函数或功能块占用的执行时间百分比：



当选中了“视图 / 平均”选项时，显示的信息就是自应用程序开始以来，或自上次使用“视图 / 复位”命令以来的平均百分比值。

如果未选中“视图 / 平均”选项，则显示测量的上一周期的值。您也可以在此应用程序处于“循环 - 循环”模式下运行时，运用此特性测量根据应用程序运行环境的不同而不同的度量值。

使用“视图 / 拷贝”命令可以将程序名和其百分值以 ASCII 格式复制到 Windows 剪贴板中，以便再粘贴到文本文件或电子数据表中。

### **重要注意：**

这些不是精确的测量。百分比计算是基于 TIC 指令以计算不同指令的执行时间的。计算并不包括用于执行“C”函数和功能块的时间。

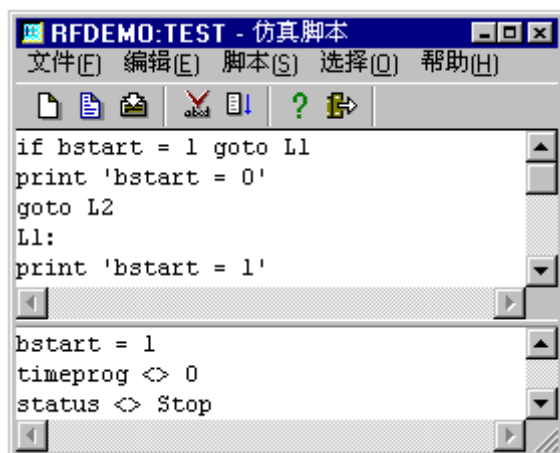
显示的值是在一个应用程序循环周期内对这一某函数或功能块的所有的“调用时间”。

时间计算基于TIC代码，如果实际的应用程序代码由C语言生成并使用了“C”编译器，它就不能提供准确的值。

### A.21.7 仿真脚本

ISaGRAF仿真器包括了建立和运行仿真脚本的工具。脚本以一种类似ST的语言描述，用于ISaGRAF仿真器的自动测试。

仿真脚本编辑器以仿真器窗口中“工具/仿真脚本”命令启动。以下是脚本编辑器的画面：



编辑器的上半窗口用于输入脚本指令。它可以象 ISaGRAF 文本编辑器一样使用，并且支持一些高层次的编辑特性例如用鼠标选择变量符号。可以使用“选项”菜单来设置表格宽度并选择字体。

下半窗口显示脚本运行时的输出信息。窗口件的分界线可以随意拖动以改变窗口大小。输出窗口可以在编辑脚本时关闭，但会在运行脚本时自动打开。

## 编辑脚本

使用“文件”菜单中的命令管理脚本文件：

新建 建立新的未命名脚本

打开 装入已存在的脚本文件

保存 在项目目录中保存脚本文本和输出窗口的内容


另存为 以另一个名称保存脚本

在 ISaGRAF 项目目录中将为每个脚本建立两个文件：

<脚本名称>.SCC 脚本文本（指令）

<脚本名称>.SCO 输出窗口内容

<脚本名称>即为脚本的名称，两个文件可以用其它文本编辑器打开的标准文本文件。

 在编辑脚本时，可以使用“编辑/插入符号”命令选择一个已声明的变量名插入到插入标记处。

## 运行脚本

脚本在运行前必须经过检验和编译。如果必要，将在运行脚本时自动进行语法检查。可以使用下列命令：



检验      检验语法和编译脚本



运行脚本      开始执行当前编辑的脚本

如果脚本未命名，则在检验前须先保存（并为此输入名称）；如果脚本已命名，则在进行语法检查前自动保存到磁盘。

脚本运行时，不能改变其内容。到达脚本的尾部时将显示信息。也可以在脚本运行当中使用“脚本”菜单中的命令中止脚本的执行：



中止脚本      中断脚本的运行

脚本在目标代码循环周期之间执行。如果在脚本中编写了无限循环，ISaGRAF 仿真器会确保总能中断这个循环，以使循环继续执行并且其它的 ISaGRAF 应用程序不致阻塞。如果在同一个循环周期内遇到同一个“标号”多于一次，则 ISaGRAF 脚本解释器会决定中断脚本的执行。脚本的执行也可由“CYCLE”或“WAIT”指令正常终止。

## 脚本描述语言

脚本描述语言是一种类似ST的简单的文本语言，每个指令都在单独的一行内输入，并且不需要以分号结束。使用工具栏上的按钮可以了解可用指令的列表并在插入标记处插入关键词：

### ? 插入指令 ( 关键词及帮助注释 )

有许多类型的指令。首先是赋值（强制）变量：

`:=` 赋值

其它指令允许信息输出到输出窗口：

`Print` 输出文本字符串或变量值

`PrintTime` 输出当前时间标签

还有指令用来与ISaGRAF 循环周期同步：

`Cycle` 使ISaGRAF 仿真器执行一周期

`Wait` 等待一定的时间

其它指令用于控制脚本中的指令流向：

`Labels` 可以置于脚本中的任意位置

`Goto` 无条件跳转到标号

`If goto` 有条件跳转到标号

## End 终止脚本

脚本语言是大小写无关的。可以在任意文本的行末插入注释。注释可以象ST语言的约定那样(置于符号“(”和“\*)”之间)，也可以一字符“;”开始。

## ":=" 赋值 t

**意义：**强制 ISaGRAF 变量值。变量可以是内部变量，输入通道或输出通道。

**语法：** <varname> := <constant\_expression>  
           <varname> = <constant\_expression>

**参数：** <varname> 是应用程序中声明的变量的有效符号，或是以“%”开始的直接表示的 I/O 变量。

<constant\_expression> 是与指定变量相匹配的有效常量表达式。对于布尔变量，可以用“0”和“1”代替“FALSE”和“TURE”；定时器变量的前缀“T#”或“TIME#”可以省略。

**注意：**由脚本强制的输入变量不必锁定。当变量由脚本强制是，相应输入通道的程序会自动更新。

**警告：**因为脚本执行不支持变换功能或表格，所以不要强制与变换相联系的输入或输出模拟变量。

**实例：** MyBooVar := 1      (\* same as TRUE \*)

```
MyIntVar := 1234  
MyRealVar := 1.2345  
MyMsgVar := 'Hello'  
MyTmrVar := t#12s
```

## Print 打印

**意义：**在输出窗口中写字符串或变量值。文本输出显示为在输出窗口底部已有文本之后的新行。

**语法：**  
Print '<text>'  
Print <varname>

**参数：**<text>为任意的在单引号内表示的文本字符串。  
<varname>是应用程序中声明的变量的有效符号，或是以“ %”开始的直接表示的 I/O 变量。

**注意：**输出变量总按 IEC 约定格式化。

**实例：**  
Print 'Hello'  
Print MyBooVar

**输出：**  
Hello  
MyBoovar = TRUE

## PrintTime 打印时间

**意义：**在输出窗口中写入当前时间标签。文本输出显示为在输出窗口底部已有文本之后的新行。

**语法：**PrintTime

**注意：**时间标签的格式依据当前 Windows 系统而定。

**实例：**Print 'Time now is:'  
PrintTime

**输出：**Time now is:  
15:45:22

## Cycle 循环

**意义：**在执行下一个 ISaGRAF 循环周期前暂停。

**语法：**Cycle

**注意：**此脚本指令在 ISaGRAF 循环开始处执行。如果仿真器运行在“循环 - 循环”模式下，“cycle”指令后将紧接着一个循环周期。下列脚本指令将在自调试器执行下一条“执行单周期”命令后执行。

实例： (\* ISaGRAF 程序将 A 复制到 B \*)

A := 0

Cycle

Print B

A := 1

Print B (\* 未执行循环 / B 未被置为 1 \*)

Cycle

Print B

输出： B = 0

B = 0

B = 1

## Wait 等待

意义： 在延时时间内暂停执行脚本。

语法： Wait <delay>

参数： <delay> 符合 IEC 定时器常量表达式的延时时间。可以省略“ T#” 和“ TIME#” 前缀。延时时间必须在 10 毫秒和 1 小时之间。

注意： “ Wait” 指令不很精确，因为它依赖于 Windows 宿主系统，同样地，ISaGRAF 循环周期也会造成延时精度的增减。

当遇到“Wait”指令时，在继续执行脚本之前会中止直到延时时间过去，然后 ISaGRAF 循环也才能继续执行。

实例：        PrintTime  
              Wait 2s  
              PrintTime

输出：        15:45:27  
              15:45:29

## Labels 标号

意义：        标号可以置于脚本内的任何位置。它们作为“Goto”指令的目标，从而允许进行脚本指令的流程控制。

语法：        <labelname>:

参数：        <labelname> 遵守 ISaGRAF 变量命名约定的唯一的名称：限制在 16 个字符内，以字母开头，后随字母或数字或下划线字符。定义时，标号名后必须跟随字符“:”。

注意：        定义标号的行中不能放置指令。  
              标号名不能与 ISaGRAF 变量符号相同。

实例： (\* 一个无限循环的脚本例子 \*)

```
loop:
PrintTime
Wait 1s
Goto loop
```

## Goto 无条件跳转到标号

意义： 无条件跳转到标号

语法： Goto <labelname>

参数： <labelname>为在脚本中定义的标号名

注意： 允许向后跳转。在无限循环的情况下，每次循环时将自动中断，以保持执行 ISaGRAF 循环。

实例： Print 'Before Jump'

```
Goto MyLabel
Print 'Within Jump' (*从未执行的指令*)
MyLabel:
Print 'After Jump'
```

输出： Before Jump  
After Jump

## If Goto 有条件跳转到标号

**意义：** 有条件跳转到标号。条件可以是两个 ISaGRAF 变量间的比较，或一个变量和一个常量间的比较。

**语法：**        If <var1> *test* <var2> Goto <labelname>  
                 If <var1> *test* <constant\_expression> Goto <labelname>

可用的比较测试 ( *tests* )

- =     如果两个值相等为“真”
- <>   如果两个值不等为“真”
- <     第一个值小于第二个时为“真”
- <=   第一个值小于等于第二个时为“真”
- >     第一个值大于第二个时为“真”
- >=   第一个值大于等于第二个时为“真”

**参数：**        <var1> <var2> 是应用程序中声明的变量的有效符号，或是以“ %” 开始的直接表示的 I/O 变量。

<constant\_expression>是与指定变量相匹配的有效常量表达式。对于布尔变量，可以用“ 0” 和“ 1” 代替“ FALSE” 和“ TRUE” ；定时器变量的前缀“ T#” 或“ TIME#” 可以省略。

<labelname>是在脚本中定义的标号。

**注意：** 允许向后跳转。在无限循环的情况下，每次循环时将自动中断，以保持执行 ISaGRAF 循环。

**实例：** (\*此脚本在 MyVar 为“真”之前一直循环 \*)

```
Loop:
If MyVar = TRUE Goto TheEnd
Print MyVar
Goto Loop
TheEnd:
```

## End 结束

**意义：** 终止脚本

**语法：** End

**注意：** 并不需要在脚本尾部特意放置“End”指令。

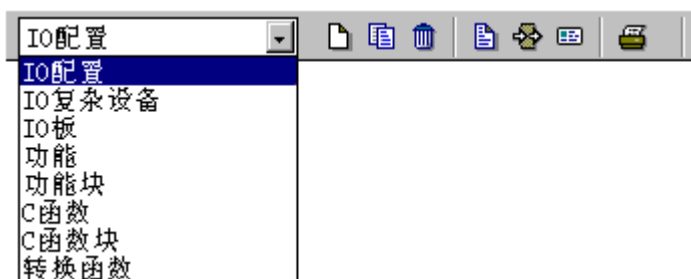
**实例：** (\*此脚本循环直至 MyVar 为“真” \*)

```
Loop:
If MyVar = FALSE Goto Continue
End
Continue:
Print MyVar
```

## A.22 使用库管理器

ISaGRAF 库为自动开发和 ISaGRAF 目标系统的软件或硬件能力间提供了标准接口。对于每个类型的接口都有一个库。ISaGRAF 工作台库管理器专为硬件提供者或软件工程师提供，他可使用库管理器来描述他所建立的对象 ISaGRAF 编程接口。

ISaGRAF 工作台库管理器可用显示某一 ISaGRAF 库内的元件。在窗口的左侧区域内是所选择的库的元件列表；右侧是在元件列表中当前选择的元件的技术注释（用户指南）。库管理器的菜单包括了一些可用来在当前库内建立，定义或修改元件的命令。“文件/其它库”命令允许在 ISaGRAF 库中选择，窗口工具栏左侧的组合框也可用来选择库：



### A.22.1 管理库元件

使用“文件”菜单中的命令在打开的库中建立或处理已存在的元件。



## 新建一个元件

使用“文件”菜单中的“新建”命令在所选择的库中建立一个新元件，其名称要按下列命名规则输入：

- 名称的最大长度是8个字符
- 第一个字符必须是字母
- 随后的字符必须是字母，数字或‘\_’字符
- 库元件的命名与大小写无关

文本注释和每一个库元件相联系。注释在建立元件时输入。当新建一个元件时，下列必须输入：

- 其 I/O 配置定义
- 其 I/O 板参数
- 其函数或功能块的用户接口

当建立“C”转换，“C”函数或“C”函数块时，将自动为其源代码生成完整的框架。



## 处理已存在的元件

“文件/重命名”命令允许用户改变从元件列表中选择元件的名称或注释。“文件/拷贝”命令允许用户复制高亮显示的元件到同一库内的另一个元件，如果目标元件已存在，其内容将被覆盖，如果目标元件不存在，它将自动建立。“文件/删除”命令删除在当前库中选择的元件。元件的以下组成部分可用“重命名”，“拷贝”和“删除”命令处理：

- 技术注释
- I/O 配置的完整定义

- I/O 板或综合设备参数
- 函数或功能块接口定义
- IEC 语言编写的函数或功能块源代码
- 变换，函数或功能块源代码

⚠ 如果元件是“C”转换，“C”函数或“C”函数块，则它在相关源代码内的名称不会为“重命名”或“拷贝”命令所自动更新。

⚠ 如果元件是用 IEC 语言编写的，则它的返回参数名不能被“重命名”或“拷贝”命令改变。

## ☐ **设置口令保护**

“文件/设置口令”命令允许用户在打开的库中为选择的元件定义口令保护。参见本指南第一部分最后的“口令保护”一节以了解口令的层次和数据保护的详情。口令只与所选择的元件有关，对 ISaGRAF 库中的**其它元件**无影响。

## ☐ **编译函数和功能块**

如果选择了用 IEC 语言编写的函数或功能块的库，可以使用“文件”菜单中的“校验（编译）”命令对选择的元件进行语法检查并生成其目标代码。在能够在 ISaGRAF 项目中使用之前，用 IEC 语言编写的函数或功能块必须编译无误。如果选择了其它库，此命令对其无影响。



## 技术注释

“文件/技术注释”命令允许用户为在当前库中选择的元件输入技术注释。输入技术注释时将使用 ISaGRAF 文本编辑器。技术注释就是元件的用户指南，为将其集成到 ISaGRAF 项目中的使用者提供参考。说明如何使用元件的技术注释必须包括其主要功能的描述，编程接口和参数的详情，以及上下文关系和限制。

“工具/标准注释格式”命令允许用户为当前选择的库中的每个元件定义有关标准文本格式。当编辑新元件的技术注释时，此格式可以作为主框架。这使用户能够优化技术注释编辑。



## 参数

参数描述了元件提供的计算机操作与在 ISaGRAF 应用程序中的元件使用之间的接口。对不同类型的库元件，参数具有不同的意义。

I/O 配置参数定义了配置的 I/O 板设置，以及 I/O 通道使用的缺省变量名。I/O 板和综合设备参数定义板的物理和逻辑配置。函数或功能块参数定义根据 ST 语言调用约定协议而定的接口。变换函数使用标准的预定义接口因而不需要参数。



## 源代码

ISaGRAF 工作台允许程序员管理库内的变换，函数或功能块的源代码。用 IEC 语言编写的函数或块的源代码

是与其联系的文本或图形语言程序。使用“C”组件“C”函数，“C”功能块和转换函数)的源代码被存于两个分开的文件中：头文件包含依照元件参数定义的接口的准确定义；源代码文件包含元件的具体操作。

ISaGRAF 工作台在新的库元件建立时生成源代码，它 also 根据参数定义建立和更新头文件。程序员可以使用 ISaGRAF 文本编辑器完成源代码文件。

## **存档库元件**

“工具/存档”菜单命令用于运行 ISaGRAF 档案管理器保存或恢复库元件。在运行“存档命令”之前必须先选择库。档案管理器每次只能显示一个库内的元件列表。

## **A.22.2 I/O 配置**

ISaGRAF I/O 配置库为使用预定义的 I/O 配置，初始化新的 ISaGRAF 项目提供了一种简便方法。I/O 配置要定义：

- I/O 板设置
- I/O 板缺省参数
- I/O 通道缺省名称

当为新 ISaGRAF 项目建立库 I/O 配置时，相应的 I/O 连接是自动设置的，同时与通道名联系的 I/O 变量也在项目字典内自动声明。



I/O 配置定义是用 ISaGRAF I/O 连接工具（与项目内使用的相同）完成的，请参照本指南“ I/O 连接”一节有关如何使用此工具的详细介绍。当在配置中插入一个新 I/O 板时，新板的所有通道都以其标准缺省名声明。I/O 通道的标准缺省名具有如下格式：

`<direction><type><slot_number>_<channel_number>`

第一个字符指出 I/O 通道的方向：

"I".....输入通道

"O".....输出通道

第二个字符指出 I/O 通道的类型：

"X".....布尔

"D".....模拟

"M".....信息

下面是标准 I/O 通道名的例子：

IX0\_7.....布尔输入-板#0-通道#7

QD2\_4.....整型输出-板#2-通道#4

I/O 连接编辑器的“连接 I/O 通道”命令用来修改 I/O 通道的缺省变量名。

### A.22.3 I/O 综合设备

在单一板上的所有通道都具有相同的类型（布尔，模拟或信息）和方向（输入或输出）。综合 I/O 设备是具有不同类型和方向的通道的 I/O 设备。综合 I/O 设备表示为单一 I/O 板的列表，它在 I/O 连接列表上占用一个机架插槽。



为定义综合 I/O 设备，用户必须定义一系列 I/O 单一板，也必须为每个单一板输入详细的参数。单一 I/O 板列表要在一个对话框中输入。

按“添加”按钮使用户能在当前列表末尾增加一个单一板。“插入”按钮用来在当前列表内的选择之前插入一个单一板。按“删除”按钮能从列表中清除当前选择的单一板。“重命名”和“参数”按钮用来改变所选择的单一板的名称和参数。参照下列有关单一板参数的完整说明。综合 I/O 设备可以由最多 16 个单一板组成。（在一个 I/O 设备内）单一板的名称不能超过 8 个字符。

### A.22.4 I/O 板

ISaGRAF I/O 板库定义了应用程序变量和目标硬件之间的标准接口。在应用程序描述中所有的 I/O 变量都与目标 I/O 板的通道相连接。一个 ISaGRAF I/O 板由名称和识别其供应者的“OEM 键码”所定义，其它的 I/O 板参数描述 I/O 板的拓扑特性（通道数，方向和类型），以及硬件或软件配置。



## I/O 板参数

I/O 板有两种不同类型的参数：定义 ISaGRAF 库板的公共参数，以及由硬件供应商提供的板设备的 OEM 参数。公共参数在 I/O 板参数定义对话框的上部输入。这些参数（以及 I/O 板名称）用来识别 ISaGRAF 标准 I/O 板接口。

“OEM 键码”是定义硬件供应者的简单数字。由同一个供应者定义的所有板都具有相同的 OEM 键码。OEM 键码是以十六进制格式输入的 16 位无符号字。为 ICS Triplex ISaGRAF Inc. 保留的 OEM 键码是“1”。

主要参数定义了 I/O 板的拓扑性质。通道数目定义了板上可用的通道数；板的类型就是能被连接到板的通道的变量的类型；方向定义了连接到板的变量是输入还是输出变量。

注意：具有不同类型和方向的 I/O 变量不能在同一个 ISaGRAF I/O 板上编组。这种情况要求使用综合 I/O 设备。



## OEM 参数

OEM 参数将在 I/O 板定义窗口的下面部分输入。这些参数由 I/O 板硬件供应商定义，并且随板设备而不同。一个板可以拥有 16 个 OEM 参数，也可以没有 OEM 参数。ISaGRAF 库管理器允许硬件供应商定义标识符和每个参数的格式，以及自控程序员输入它的方式。

左侧的框内是 OEM 参数列表。每个参数由一个名称和一个逻辑号码识别，号码的值从 0 到 15。右侧的区域显示在列表内选择的参数的详细描述。在列表中选择参数以访问其全部描述。按“清除”按钮可以重新设置参数描述，并将其从参数列表中删除。警告：此命令执行后不能“恢复”。

如果自控操作员必须在连接 I/O 板时定义相应的输入区，可以使用参数名来加以识别。参数名必须遵从下列规则：

- 名称的最大长度不超过 16 个字符
- 名称的第一个字符必须为字母
- 随后的字符必须是字母，数字或‘\_’字符

参数的类型定义了参数的内部格式和它在应用程序 I/O 连接时的输入格式。下面是可用的内部格式：

word .....无符号 16 位字  
 long .....无符号 32 位字  
 word hexa .....无符号 16 位字  
 long hexa .....无符号 32 位字  
 boolean .....符号 16 位字 ( 只用最低位 )  
 character .....无符号 16 位字 ( 只用最低字节 )  
 string .....以 “ null” 终止的 16 字节数组  
 float .....单精度 32 位浮点数值

下面是可用的输入格式：

word .....无符号十进制字  
 long .....十进制长字

word hexa .....无符号十六进制字  
long hexa .....无符号十六进制长字  
boolean .....“真”或“假”  
character .....单字符  
string .....ASCII 码字符串 ( 最长 15 字符 )  
float .....单精度浮点数值


“访问”框规定了最终用户如何访问参数。如果设置了“用户定义”选项，则参数将在连接 I/O 板时显示在输入区内，并使用 OEM 参数的缺省值以备编辑；如果选择了“隐藏”选项，则参数为常量并不在 I/O 板连接框内出现，且用 OEM 参数缺省值作为常量参数。“只读”选项将指出参数对用户是可见的，但不能修改，将使用常量值作为其缺省值。

### A.22.5 IEC 语言函数和块

ISaGRAF 可以处理用 IEC 语言编写的函数和功能块的库。可用于描述这样的函数或功能块的语言包括 FBD (功能块图)，LD (梯形图)，ST (结构文本)，或 IL (指令列表)。注意 LD 和 FBD 语言可以在同一程序中混合使用。SFC (顺序功能图) 不能用来在库中描述函数或功能块。库元件使用的语言在函数建立时就已选定，之后不能改变。

## 编译

在库中定义的函数和功能块，在能够在 ISaGRAF 项目中使用前，必须经过编译（校验）。此外不必在库内对函数或功能块进行任何改变。库元件会自动在项目的 LD/FBD 图形编辑器选择框内出现。

 在库中定义的函数可以调用另一个库内的函数。ISaGRAF 系统在函数调用中不支持递归。用 IEC 语言编写的函数不能调用其它函数（无论是用 IEC 语言或“C”语言编写的）。




## 输入源代码

库函数或功能块的源代码可以用标准的 ISaGRAF 工具输入：LD 或 FBD 程序的图形编辑器，ST 或 IL 程序的文本编辑器等。参见本指南的有关这些工具的章节，以了解详情。可以从图形或文本编辑窗口内直接调用 ISaGRAF 代码生成器，以编译库函数或功能块的源代码。

## 局部变量字典

库函数或功能块可以拥有局部变量，以及局部定义字。编辑源代码时，用户可以用编辑窗口中“文件”菜单的“字典”命令访问变量声明。

 库函数或功能块不能访问全局变量或功能块实例。函数的局部变量必须在函数的内部初始化。

每次在项目中调用 IEC 语言功能块时，局部变量都被拷贝（实例）。同一个实例的局部变量在每次调用时保持其值不变。



### 定义接口

函数或功能块可以拥有最多 32 个参数（输入或输出）。一个函数总有一个（并只有一个）返回参数，返回参数必须与函数同名，以符合 ST 语言编写约定。

在窗口的左上侧，参数列表以调用模式的顺序显示：首先是调用参数，最后是返回参数。窗口的下部分显示当前选择的参数的详细描述。参数可能使用任何 ISaGRAF 数据类型。返回参数在列表中必须置于调用参数之后。

参数的命名必须遵照下列规则：


- 参数名称的长度不能超过 16 个字符
- 第一个字符必须是字母
- 随后的字符必须是字母，数字或下划线符号
- 命名与大小写无关

“插入”命令用来在选定的参数前插入参数。“删除”命令用来清除选择的参数。“排列”命令用于自动重新排列（排序）参数，以使返回参数置于列表的最后。

## A.22.6 C” 函数和功能块

“C”函数和功能块是自控应用程序，依照 ST 语言函数调用接口，调用的计算机功能。

函数是同步过程，在它执行过程中 ISaGRAF 目标应用程序被挂起。功能块把应用程序操作和静态隐藏数据联系起来。例如，“计数器”功能块代表了一种计数操作，同时也表示了操作计数的结果。函数和功能块可以用来完善标准自控语言能力，或用来访问系统资源。

 参数定义框用来定义函数或功能块的调用或返回参数。使用“编辑”菜单定义选定的函数或功能块的参数。有关函数可以有最多 31 个调用参数并总有一个返回参数。功能块可以有 32 个混合的调用和返回参数参数。下面是 ISaGRAF 参数类型和“C”函数和功能块参数类型的对应关系：

BOOLEAN      无符号长型    无符号 32 位字：1 = “真” / 0 = “假”

ANALOG          长型    有符号整型 32 位值

REAL float 浮点      单精度浮点值

TIMER 无符号长型    无符号整型 32 位字（单位为 1 ms）

MESSAGE      字符    字符串。

传递到“C”函数或功能块的信息中不能包括“null”字符，字符串以“null”结束。

参考 ISaGRAF 目标用户指南有关如何管理“C”函数或功能块源代码，以及如何在 ISaGRAF 目标系统中集成新元件的进一步说明。

## A.22.7 变换函数

变换函数是 ISaGRAF I/O 管理器调用的“C”函数，每次调用时，使用此转换函数的模拟变量将输入到或输出自 ISaGRAF 项目。

此函数在变量的（读自输入传感器或发送到输出设备的）电气值和（使用应用程序表达式的）物理值之间建立联系。所以函数被分为两部分：输入变换或输出变换。ISaGRAF 库管理器允许用户控制变换函数的“C”源代码。

变换可以用于整型和实型模拟变量。这意味着与其它变换函数一样，转换函数的接口总定义为浮点值。接口的“C”定义由“TACN0DEF.H”文件定义。

参考 ISaGRAF 目标用户指南有关如何管理变换函数的“C”源代码，以及如何在 ISaGRAF 目标系统中集成新元件的进一步说明。

## A.23 档案管理实用程序的使用

ISaGRAF 的档案管理实用程序可以帮助用户将 ISaGRAF 项目文件和库文件存储到软盘或备份目录上。ISaGRAF 档案管理器是一个能从 ISaGRAF 的项目管理器窗口或库管理器窗口调出的对话框。



我们建议遵照下列规则建立和保存可靠的档案

- 在磁盘的不干胶标签上写出被保存的目标的名称和说明
- 不在同一软盘上存储项目文件和库文件
- 不在同一软盘上存储不同的项目文件

### A.23.1 调用档案管理器

可以从项目管理器窗口的“工具/档案”菜单中调出档案对话框，进行项目文件或公共数据的存储或恢复。

也可从 ISaGRAF 中的库管理器窗口的“工具/档案”命令运行档案对话框，对在库管理器窗口中当前选定的库元素进行存储和恢复。



#### **项目**

项目总是以完整的形式存储的。项目的所有组成部分（程序源文件、目标代码和应用程序执行代码）均存在

同一档案文件中。选择“压缩”选项可减小项目档案的大小。

### ≡ **库元素**

ISaGRAF 的库元素可以单独存储。库元素的所有组成部分(技术说明、定义、接口、源代码等等)均存在同一档案文件中。

### ≡ **公共数据**

项目管理器窗口的“工具/档案/公共数据”命令可以使用户备份或恢复已存在 ISaGRAF 工作台中的“公共”数据。但这条命令对 ISaGRAF 库本身不起作用。下表是用本命令可拷贝的文件一览表：

common.eqv	公共定义字
oem.bat	用户已定义的 MS-DOS 命令文件

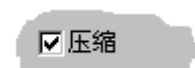
这些文件按它们的原始形式一个接一个地存在档案盘上。相应的档案文件永远不会被压缩。

## **A.23.2 选项**

ISaGRAF 的档案可以显示在对话框的底部。按浏览键就可浏览磁盘，选择其它档案盘和目录。



当选定“压缩”选项时，在“备份”过程中建立的全部档案文件将全部被压缩。这个选项对缩小大项目档案文件的大小是非常有用的，并可将大文件存在一张盘上。对于库文件的组成部分通常是不需要进行档案压缩的。在恢复档案文件时，ISaGRAF档案管理器可以自动识别该文件的状态（被压缩或没有被压缩）。这表明“压缩”选项对于“恢复”过程是不起作用的。



### A.23.3 备份与恢复

安装在硬盘上的“工作台”表（在屏幕的左边）显示出ISaGRAF工作台中的存放的对象文件。“档案”表（在屏幕的右边）显示出存放在指定档案盘或目录上的对象文件。

#### 备份

选择出现在屏幕左边列表中的对象文件（即ISaGRAF工作台表中的对象文件），然后按下“备份”按钮，就可完成将一个目标文件存到档案文件中这一归档过程。在对象表中可以选择多于一个目标文件。当元素是从

屏幕右边的表(恢复模式表)中选定的时,则不能按下“备份”按钮。

## 恢复

选择在屏幕右边列表中的对象文件(即归档对象),然后按下“恢复”按钮,就可以完成从档案文件拷贝对象这一过程。在对象表中可以选择多于一个目标文件。当元素是从屏幕左边的表(备份模式表)中选定的时,不能按下“恢复”按钮。

### A.23.4 档案文件

ISaGRAF 档案管理器对每类存贮对象都会建立一个独立的档案文件。档案文件具有与对象文件相同的名称。名称后缀表明文件类型。下面是所使用的名称后缀:

.pia ... 项目  
.bia ... I/O 板  
.iia .... IEC 语言函数  
.aia ... IEC 语言功能块  
.uia ... C 语言函数  
.fia .... C 语言的功能块  
.cia ... C 语言变换函数  
.ria .... I/O 配置  
.xia ... I/O 设备



## A.24 打印文件

ISaGRAF 文件生成器可以使用户为选定的项目建立并打印一个完整的文件。打印文件可由调用项目管理窗口或程序管理窗口中的“项目/打印”命令来完成。也可以由所有其它 ISaGRAF 编辑器中的“打印”命令调用的文件生成器来打印单独的 ISaGRAF 文件的内容。不过，文件生成器可以为以上两种情况下提供相同的打印功能。

“编辑”菜单的命令用来定义必须插入文件的项目元素。用户只需为所需的文件建立“内容表”。与项目有关的任何信息(如程序、变量、选项、输入/输出连接等)都可以插入到项目文件中。但其它项目的信息或其它 ISaGRAF 库的信息不能出现在这个项目文件中。



“文件/打印”命令可以生成一个文件，并根据指定的内容表，将该文件送到打印机。“打印”作业需要占用几分钟来建立文件并对该文件进行格式化。郑重建议：用户应一直要等到“打印任务”出现在 ISaGRAF 的文件生成器窗口结束之后，再运行 ISaGRAF 工作平台的其它命令。建立整个文件需要很大的硬盘空间。如果硬盘已满，将会显示一个出错信息。在这种情况下，用户必须删除文件以释放硬盘空间，或者减小打印作业的尺寸。在执行“打印”命令的时候，会出现一个对话框。用户可以使用这个对话框，输入对实际打印命令的说明。这些说明存在一个记载文件

中，并打印在将来要打印的任何文件(包括现在要打印的文件)的第一页上。

### A.24.1 自定义文件表内容

“编辑”菜单中包含了定义文件“内容表”的各个命令。对这些命令的不同选择，可以使用户使用缺省表(带有项目的所有组件)、建立指定表(仅有部分组件)或者移动内容表中的条目并且修改这些条目。



#### **缺省一览表**

“编辑”菜单中的“缺省一览表”命令可用于为包括项目全部组件的文件定义一张标准内容表。这个标准表中包括：

- 项目描述
- 层次树(程序之间的连接)
- 程序的源代码文件
- 程序的日志文件
- 公共定义
- 全局定义
- 程序的局部定义
- 全局变量
- 程序的局部变量
- 应用程序选项
- 输入/输出连接
- 变量一览表

- 转换表
- 精减的交叉引用
- 详细的交叉引用
- 声明汇总
- 网址图
- 修改的记载

使用“文件/存储”命令可以将上表的内容存到硬盘上。当ISaGRAF编辑器运行文件生成器，以打印单个文件时，这个命令变成灰色。



### **剪切和粘贴**

为了改变自定义文件表的顺序，可以使用“编辑/剪切”和“编辑/粘贴”命令，在一览表中移动各个条目。文件生成器允许用户进行多重选择，对条目组进行剪切和粘贴。



### **清空表**

使用“编辑/清除”命令可以对文件表进行重新设置，这样就可以使用插入单个条目来进行文件表的重建。



### **在文件表中插入条目**

当执行“编辑/插入”命令时，会出现“添加条目”对话框。用户可以使用这个对话框把各个条目（即项目的组件）插入到文件表中。

使用“程序”复合框可以为与程序有关的条目，选择程序名。按下“添加”按钮，就可以将选中的条目插入到文件表中。但相同的条目在一张表仅能出现一次。

## A.24.2 选项

“选项”菜单中的各个命令可以用来定义和自定义生成文件的格式。

其它选项可从文件生成器窗口中的按钮直接调用：



当“前页”选项被设置时，在文件开始处打印首页，这个首页包含项目标题和打印输出的历程。当这个选项没有被设置时，在第一页的开头打印第一个条目的内容。

当“内容表”选项被设置时，文件表的内容被打印在所生成文件的末尾。

使用 ISaGRAF 编辑器 (如程序、字典等编辑器) 启动“打印”命令时，不对以上这两个选项进行初始检查。

## ☐ SFC 图

“独立的 SFC 级别”选项用于对每个 SFC 程序的打印进行系统管理。先打印 SFC 程序的第一级 (图和注释)，再打印随后的第二级编程内容。在系统没有检测到这个

选项时，程序的第一级和第二级一起同时出现在打印输出上。



### 页面格式

“选项”菜单中的“页面格式”命令可在格式化打印页面时，用来定义由文件生成器使用的主要参数。下列参数应该指定：

- 左边距: (页边距为 1 厘米, 2 厘米或没有)
- 页边框: 当选定这个选项时, 在打印页旁画出边框。



### 页标题模板

“选项”菜单中的“页面标题”命令可以用来定义在页面底部打印的标题框的内容。这个标题框的标准版式如下：

	正文1	ISaGRAF - 项目 'PrName'	日期
	正文2		
	正文3	用户定义的标题	页号

主标题的第一行 (即带有 ISaGRAF 项目名的一行)、当前日期和页号使由文件生成器自动生成，不能修改。

标题框左边的三行文本 (文本 1、文本 2、文本 3) 和主标题的第二行是由用户定义的。用户还可以改变打印在框左边的图标。用户可以采用指定位图的映像文件 (.BMP 文件) 的路径名的方法来使用其它图标。这个映像图的尺寸可以是任意的，可以根据打印纸的尺寸放大

或缩小。单击对话框中的图标区域，就显示新的指定映像图。在执行“打印”命令时，映像文件必须在盘上（在指定目录中并有指定的文件名）。



### 字体的选择

“选项”菜单中的“文本字型”和“标题字型”命令用来在打印文本和文件的条目时，定义所使用的字体。也可用于为文本和标题选定字体大小和样式。字体的选择在由Windows定义的标准对话框中进行。所有文本（如程序文本，图表中的名称等）都将按照所选定的文本字符的大小、样式和字体进行打印。但只有标题按选定的标题字体打印。

如果没有定义字体，系统将使用打印机的标准字体（即使用下列样式）打印文本。

- 用于正文和图表内的名字的“标准”样式
- 用于标题的“黑体”样式

## A.25 口令保护

ISaGRAF 工作台软件系统有一个完整的数据保护系统，这个系统可以使用户用口令来保护项目文件和库元素。一个库元素可以是一个输入/输出的配置、一块输入/输出插板或一个综合设备，用 IEC 代码编写的函数或功能块、C 语言的函数、功能块和变换函数等等。一个口令保护数据库对一个项目文件或一个库元素来说是专用的，不能与其他项目文件和库元素共享。

### 保护级别

对于一个项目文件或库元素来说，用户可以定义多达 16 个存取级别，每个级别都有不同的口令与之相对应。存取级别存贮在一个层次系统中，它们的编号可为 0 到 15，最高的存取级别为 0 级。当用户知道某一级口令时，就可以存取由相应存取级别保护及更低的存取级别保护的数据项。项目文件的基本命令和基本数据或库元素都可由一个存取级别分别进行保护。例如，ISaGRAF 菜单中的生成应用程序代码命令可以受到分别保护。基本数据可以是程序、选项列表、库元素的技术说明等等。

### 定义口令保护

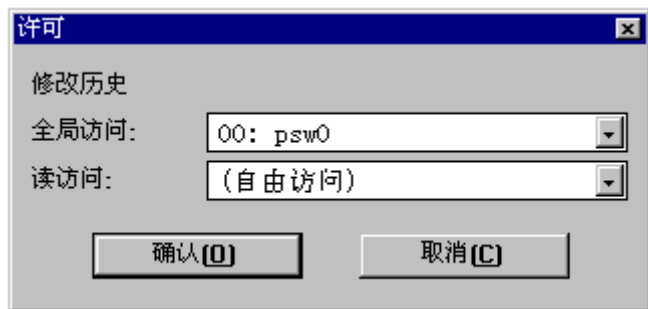
ISaGRAF 菜单中的“设口令”命令可以定义一个项目文件或一个库元素的口令和存取级别。这个命令可以从 ISaGRAF 项目管理器菜单或 ISaGRAF 库管理器菜单中调用。当第一次运行本命令时不需要口令。如果口令已

经定义，用户在存取本命令之前，必须输入他所知道的最高级的口令，以后较高级的口令及受保护的项不能修改。“设口令”命令使用户可以定义与不同存取级别相对应的口令，并用所定义的级别来保护基本命令和数据。

分别对应不同的保护级别的各个口令，可以通过双击屏幕上方表格中的一行的方法来输入。可在下列框中输入口令。

屏幕下方区域中的表格中显示了受到保护的不同的数据项(数据或函数)以及与当前许可进行的“读”方式或“全部存取”方式的保护级别。指定许可的“读”方式的保护级别，可以防止其他用户在没有得到允许的情况下打开或打印你的文件。

双击屏幕下方表格中的一行，可以对选择项或数据进行许可设置。此时，打开下列框：



“读”方式或“全部存取”方式都可设置为任意存取方式或由口令定义的保护级方式两种情况。“全部存取”方式的许可不能加到优先级低于“读”方式的保护级别上。

注意：在使用 ISaGRAF 工作台时，有一些文件（比如项目描述文件）的读存取方式是不能用口令来保护的。只读存取不能由口令保护。

## 保护数据的存取

当启动时，系统不要求输入口令或用户名。但用户每次存取被保护的数据和函数时，则必须在对话框中输入所需要的口令。

在用户输入了所需要的口令（或者更高存取级别的口令）后，就可以继续正常工作了。用户每次输入口令后，这个口令就存放在存储器中，所以用户在以后不必重复输入口令。在每次 ISaGRAF 工具运行其他 ISaGRAF 工具（例如项目管理器运行程序管理器）时，系统都会保留所存储的口令。当关闭最后保留的 ISaGRAF 窗口时，存储的口

令就会自动消失。在进行项目文件编辑、使用库管理器或档案管理器时，输入的口令不能进行共享。如果用户输入了错误的口令，不能运行所选定的功能。

## 二 **链接档案管理器**

在把一个对象文件(项目文件或库元素)存储到档案盘时，应调用被称为“备份”的数据保护项。这个数据保护项与在工作台(硬盘)上连接到对象文件上的数据保护系统是一致的。假如“备份”的数据保护项已经存在于档案盘上，系统就不会进行对象文件的数据保护系统测试。ISaGRAF 档案管理器的命令将对象文件的数据保护信息存储到档案盘上。

当恢复已存在于工作台(硬盘)上的对象文件时，应调用称为“覆盖”的数据保护项。这个数据保护项与在工作台(硬盘)上连接到对象文件上的数据保护系统是一致的。假如覆盖的数据保护项已经存在于档案盘上，系统就不进行对象文件的数据保护系统测试。一旦这个命令生效，恢复存储数据的保护信息将会取代已存到硬盘上数据的保护信息。

## 二 **设置变量和输入/输出通道的单独保护**

ISaGRAF 工作台提供了一个完整的以层次口令为基础的数据保护系统。变量说明和输入/输出连接设置等都可以受到口令的全面保护。另外，ISaGRAF 还允许用户为变量和通道设置单独保护。可以设想：

- 各个口令已经在口令定义系统(使用项目管理器窗口中的“项目/设口令”命令)定义完毕,这样所对应的各个保护级别就可以用于单独保护。

为了对变量和进行单独保护,用户可以使用比全局变量或I/O优先级更高的保护级别。

当变量或I/O通道设置受到单独保护时,在字典或输入/输出连接窗口中,受保护对象的名字旁出现一个小图标。

使用字典窗口或I/O输入/输出连接窗口中“编辑”菜单中的“设保护”和“删除保护”命令,可以对选定的变量或通道进行单独保护的设置或删除。“设保护”和“删除保护”命令要求用户输入一个有效的口令,使保护级别与变量或通道联系起来。因此,在每次用户想要改变带有单独保护的变量或连接通道设置时,必须先输入具有足够优先级别的口令。

注意: 如果一个变量或通道设置受到保护,而相应的口令从保护系统中被删除,并且没有定义更高级的口令,则该变量或通道设置不能进行任何改动,除非定义了具有一定级别的新口令。

## A.26 高级编程技术

本章介绍了更多的有关 ISaGRAF 工作台及目标系统内容。建议用户在阅读本节之前，先熟悉 ISaGRAF 的工具及其使用方法。

### A.26.1 关于 ISaGRAF 系统工具更多的内容

在使用 ISaGRAF 系统的编辑工具时，用户可按下鼠标的右键，打开一个弹出菜单，它包括了主要的编辑命令。这个菜单在光标的当前位置打开。在使用剪切命令和粘贴命令时，对于减少鼠标操作次数是非常有用的。

ISaGRAF 系统工具支持多重操作。虽然不能为编辑同一文件而两次打开同一个工具，但可以采用并行的操作方式，在不同的窗口编辑不同的对象。

在工具栏中可以找到图标信息。双击工具栏的空白区域，就可以以弹出菜单的形式，显示出工具栏的内容。鼠标的光标停留在一个图标上，可以显示与图标相应的文字命令。

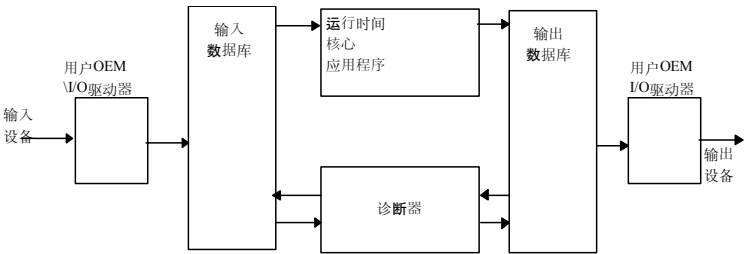
## A.26.2 I/O 板和虚拟 I/O 板锁定

将一块 I/O 板定义为虚拟型，可以切断对物理 I/O 通道的处理。当一块 I/O 板被定义为虚拟型时，ISaGRAF 系统的内核操作并没有改变，所不同的只是不读输入传感器的信号，不更新输出装置。在这种模式下，可以用 ISaGRAF 系统的诊断器修改输入值。虚拟特性适用于整块 I/O 板。在应用代码生成之前，可以编程进行 I/O 板的定义。虚拟特性是一种静态特性，在停止应用程序或重新启动应用程序时，应进行存储。

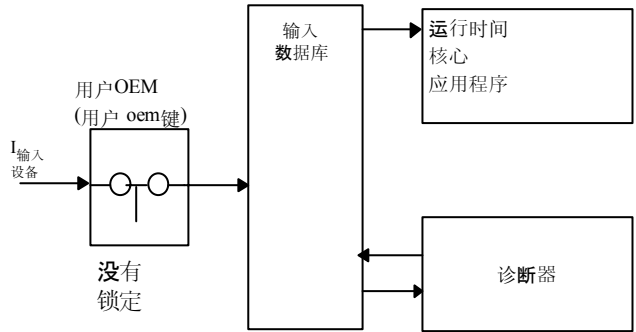
另一种可能是 I/O 变量的锁定。锁定就是切断一台物理装置与相应的 ISaGRAF I/O 变量之间的联系。变量的锁定和解锁可以通过诊断器进行。变量的锁定是一种动态操作，在应用程序启动时，系统不存储变量的锁定情况。在某一时间，只能对一个变量（即一个 I/O 通道）进行锁定。下表是主要的输入/输出控制特性：

	虚拟特性	锁定操作
工具选择	I/O 板连接	诊断器
定义	静态	动态
模式选择	板	变量
应用程序	有效和测试	维护

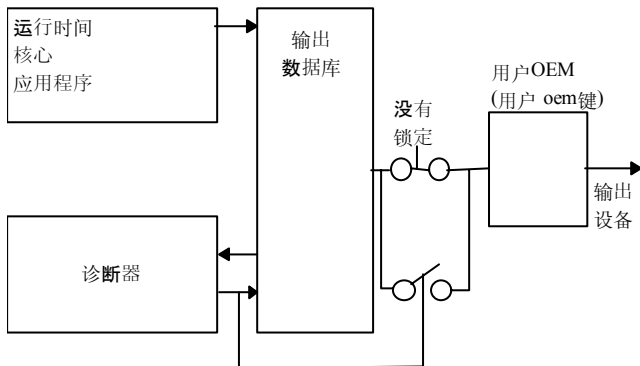
下表说明了在 ISaGRAF 的各项任务之间，I/O 数据流的情况



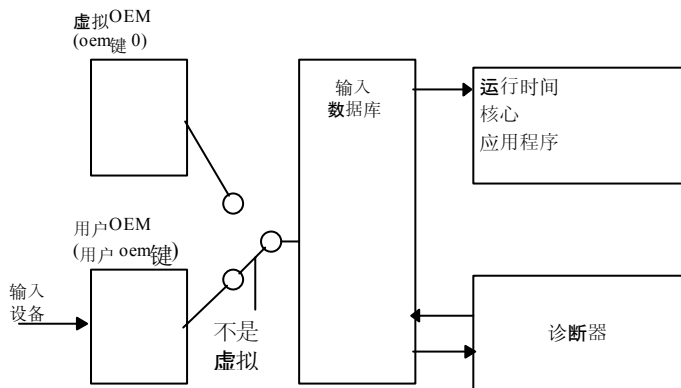
当一个输入变量被锁定时，数据库的存取方式不会发生改变，但输入装置被切断了。输入值可以由诊断器进行设置，并由 ISaGRAF 的内核 进行处理：



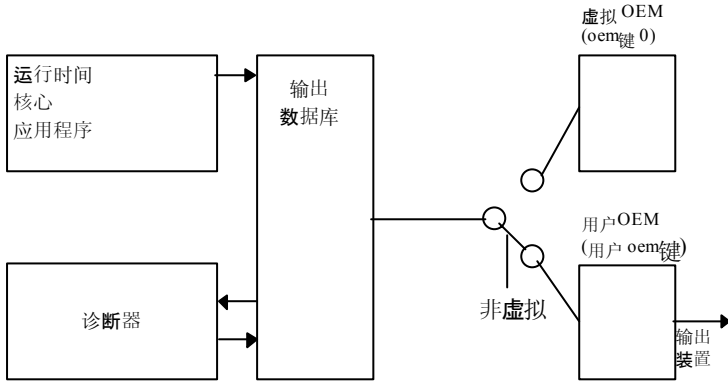
当一个输出变量被锁定时，实时内核与输出装置之间的联系被切断。在这种情况下，使用 ISaGRAF 诊断器，通过输出装置，仍可进行输出装置的存取。



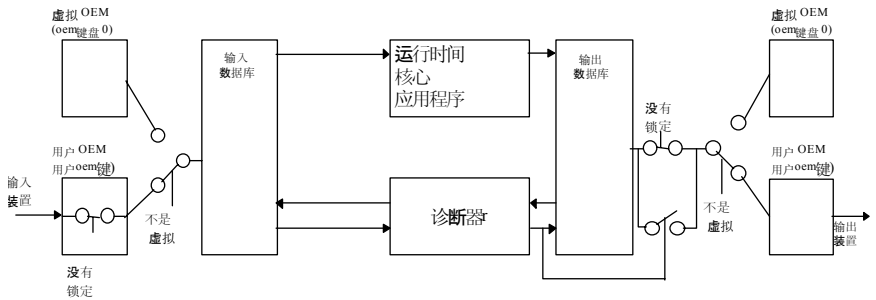
在为一个输入信号进行虚拟特性设置时，输入数据库与相关的输入状之间的联系就被切断了。一个虚拟的 I/O 驱动器取代了实际的驱动器。



对一块输入板或输出板设置虚拟特性，应遵从相同的规则。对于输出板，虽然输出数据库与相应的输出装置之间的联系已经被切断了，但 ISaGRAF 内核仍将更新输出数据库。一个虚拟的 I/O 驱动器取代了实际的驱动器。



综合所有可能的情况:



### A.26.3 PC-PLC 连接确认

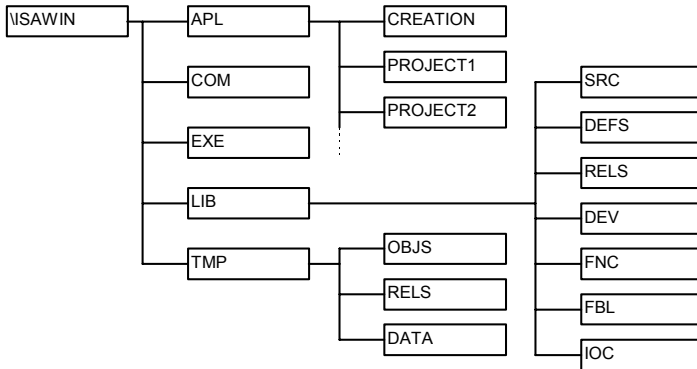
大部分 ISaGRAF 工作台与 PLC 之间通讯不好的问题都可以通过“断开”状态信息反映在诊断器窗口上。在执行诊断测试之前，当 PLC 上有“没有激活的应用程序”信息时，通讯应当确认。使用这种通讯方式，串行连结可以自己确认自己，而与其它相关因素隔离。

用 C 语言编程的变换函数和其它函数可以直接接近目标系统。在这样的软件中，编程错误可能导致系统错误或不正确的 ISaGRAF 系统动作。在使用 ISaGRAF I/O 工具箱进行 I/O 驱动器开发时，就会出现这样的问题。例如：如果把一块 I/O 板连结到一个不正常的总线地址上，就会造成系统错误。下表中为错误诊断综合一览表。

状 态	范 围	诊断结果
"断开"		- 目标没有运行
(在下载之前)		- 没有连结电缆或电缆故障
		- 连结参数有误
		- ISaGRAF 目标安装有误
"断开"	循环到循环	- I/O 配置有误
(在下载之后)	启动模式	- 系统崩溃
	实时	- I/O 配置有误
	启动模式	- 系统崩溃(由于采用 C 语言编程)
"无应用"		- 应用程序没有下载
		- 应用程序没有启动(由于采用 C 语言编程)
		- Intel/Motorola 芯片配合不当
		- 目标软件版本有误

#### A.26.4 ISaGRAF 系 统 目 录

ISaGRAF 工作台在一个指定的磁盘目录下工作。在 ISaGRAF 软件安装时，用户可以指定这种结构的根目录。ISaGRAF 根目录的缺省名是 ISAWIN。以下是由软件安装时产生的标准磁盘结构：



以下是标准 ISaGRAF 软件的子目录

目 录	内 容
-----	-----

A P L	<p>ISaGRAF 项目的根目录</p> <p>每个项目都对应一个包含项目所有数据的子目录。</p> <p>其他的项目组也可以有其他的目录。ISaGRAF 软件安装时，会产生 一个名为"SMP"的目录，将应用的例子存放在这个目录中。</p>
COM	<p>"公共"范围内的数据</p> <p>这些数据可以供任意项目使用</p>
EXE	ISaGRAF 软件的程序文件和帮助文件。
LIB	<p>ISaGRAF 软件的 库文件：</p> <ul style="list-style-type: none"> <li>- 元素一览表</li> <li>- 每个元素的参数或接口</li> <li>- 技术说明</li> </ul>
LIB\IOC	用于 I/O 配置的源代码

LIB\FNC	用 IEC 语言编写的函数源代码
LIB\FBL	用 IEC 语言编写的功能块源代码
LIB\SRC	供变换和 C 语言函数用的源代码
LIB\DEFS	供变换和 C 语言函数用的头函数源代码
LIB\RELS	变换和 C 语言函数的目标文件代码
LIB\DEV	开发"C"语言库文件所使用的命令文件 如 make 文件、链表等
TMP	临时文件：TMP 中的子目录为 ISaGRAF 代码生成器而保留 ，不被删去

子目录可以移动到其他磁盘上。当用户使用非标准的磁盘结构时，子程序的路径名应存放在 ISaGRAF 软件的 EXE 子目录内的初始化文件 ISA.ini 的 WS001 节之中。以下是 WS001 节中记录项的内容：

Isa	ISaGRAF 软件结构的根目录
IsaExe	ISaGRAF 程序文件和帮助文件的根目录
IsaApl	ISaGRAF 项目文件的根目录
IsaTmp	临时文件目录
IsaSrc	库文件源代码目录
IsaDefs	库源代码文件头目录

注：如果用户将 IsaTmp 改变为另外的目录，应在新目录中生成 OBJS、RELS 和 DATA 等三个子目录。在下面的例子中使用 WS001 节，以重新定义一个标准的 ISaGRAF 磁盘结构。

*;file c:\ISAWIN\EXE\ISA.ini*

[WS001]

Isa=c:\isawin

IsaExe=c:\isawin\exe

IsaApl=c:\isawin\apl

IsaTmp=c:\isawin\tmp

IsaSrc=c:\isawin\lib\src

IsaDefs=c:\isawin\lib\defs

当用户试图向 ISaGRAF 系统增加 C 语言的函数或功能块时，必须在 \ISAWIN\LIB\DEV 目录中存放各种供开发用的文件：如命令文件、make 文件、图形文件等等。\\ISAWIN\LIB\RELS 目录还用于存放 C 语言编译时生成的目标文件和 ISaGRAF 进行链接操作所需的 "C" 语言库文件。

### A.26.5 应用符号

一项 ISaGRAF 应用程序中的每个目标都可由一个文件名（在变量声明时输入）和一个由代码生成器计算而得的内部虚拟地址进行访问。变量的虚拟地址与说明变量时输入的网络地址是不同的。虚拟地址用于通讯，也可用于利用 OEM 选项时的特殊 "C" 语言开发的应用程序。在 ISaGRAF 代码生成器工作时，生成器将产生一个为项目的所有目标（变量、程序阶跃等）所用的各个文件名和

虚拟地址之间逻辑响应的 ASCII 码文件。这个文件很容易查询，可根据用户的需要达到 ISaGRAF 静态数据库中有关的信息。这个文件的文件名是 APPLI.TST，存放在 \\SAWIN\\APL\\praname (praname 是项目名) 目录下。本章节将介绍 APPLI.TST 文件的详细格式。下面说明中使用的主要符号：

VA 虚拟地址

ATTR 变量的特征值

USP C 语言的函数

一个变量可能的特征值如下所示。这些值都出现在 attributes 字段中。

+X 内部变量

+C 只读内部变量

+I 输入变量

+O 输出变量

除了虚拟地址以外的所有数字，都可以用十进制的整数来表示。虚拟地址 (VA) 则用四位十六进制数来表示，并在其前边加上 '!' 符号。例如：

123 这是一个十进制数

!A003 这是一个十六进制的虚拟地址

"APPLI.TST"文件的主结构如下所示。文件结构为功能块表。一个块就是一个纪录表。每个纪录都由一个字符行来描述。每个功能块都由一个标题开始，标题占一行。

```
Start block
description blocks
end block
```

一个功能块的一般结构如下所示:

```
@ <block_name> <arguments>
#record...
#record...
...
```

包含应用所需的主要信息的第一个功能块的结构如下所示：

```
@ISA_SYMBOLS,<appli_crc>
#NAME,<appli_name>,<version>
#DATE,<creation_date>
#SIZE,G=<nbprg>,S=<nbstep>,T=<nbtra>,L=0,P=<nbpro>,V=<nbva
r>
#COMMENT, ICS Triplex ISaGRAF Inc.
```

appli_crc	应用程序符号的和校验
appli_name	应用程序的名称
version	ISaGRAF workbench 版本号

creation_date	应用程序的日期
nbprg	程序号
nbstep	SFC 代码的程序步号
nbtra	SFC 代码的转换号
nbpro	所使用的"C"语言的函数号
nbvar	变量的总数

表示文件结束的最后一个功能块的结构如下所示：

@END\_SYMBOLS

用于说明应用程序的功能块的结构如下所示

```
@PROGRAMS,<nbprg>
#<va>,<name>
#...
```

nbprg	本块中定义的程序号
va	程序的虚拟地址
name	程序名

下面给出了用于说明应用 SFC 代码程序步的功能块的结构。注意对于每个非 SFC 程序都定义一个虚拟的步号。

```
@STEPS,<nbsteps>
#<va>,<name>,<father>
```

#...

nbsteps	本功能块中定义的程序步号
va	程序步的虚拟地址
name	程序步名
father	父程序的虚拟地址

下面给出了用于说明应用 SFC 转换的功能块的结构：

```
@TRANSITIONS,<nbtrans>
#<va>,<name>,<father>
#...
```

nbtrans	本功能块中定义的转换块号
va	转换块的虚拟地址
name	转换块名
father	父程序的虚拟地址

下面给出了用于说明应用布尔变量功能块的结构：

```
@BOOLEANS,<nb_boo>
#<va>,<name>,<attr>,<program>,<eq_false>,<eq_true>
#...
```

如果变量值超过 4095：

```
X#(1.<varno>),<name>,<attr>,<program>,<eq_false>,<eq_true>
```

nb_boo	本块中的变量个数
va	变量的虚拟地址
varno	地址的范围(应在布尔数据类型的范围内)
name	变量名
attr	变量的特征
program	主程序的虚拟地址 对于全局变量为"!0000"
eq_false	假值所使用的字符串
eq_true	真值所使用的字符串

下面给出了用于说明应用模拟变量的功能块的结构：

```
@ANALOGS,<nb_ana>
#<va>,<name>,<attr>,<program>,<format>,<unit>
#...
```

and if variable number exceeds 4095:

```
X#(2.<varno>),<name>,<attr>,<program>,<eq_false>,<eq_true>
```

nb_ana	块中的变量个数
va	变量的虚拟地址
varno	地址的范围(应在模拟数据类型范围内)
name	变量名
attr	变量的特征
program	主程序的虚拟地址

	对于全局变量为"!0000"
format	= 整型变量为"I"
	= 实型变量为"F"
unit	单元字符串

下面给出了用于说明应用计时器变量的功能块的结构：

```
@TIMERS,<nb_tmr>
#<va>,<name>,<attr>,<program>
#...
```

如果变量值超过 4095：

```
X#(3.<varno>),<name>,<attr>,<program>,<eq_false>,<eq_true>
```

nb_tmr	本块中变量的个数
va	变量的虚拟地址
varno	地址的范围(应在计时器数据类型范围内)
name	变量名
attr	变量的特征(通常为+X，内部变量)
program	主程序的虚拟地址
	对于全局变量为"!0000"

下面给出了用于说明应用通讯变量的功能块的结构：

```
@MESSAGES,<nb_msg>
#<va>,<name>,<attr>,<program>,< max_len>
```

#...

如果变量值超过 4095 :

X#(4.<varno>,<name>,<attr>,<program>,<eq\_false>,<eq\_true>

nb_msg	本块中变量的个数
va	变量的虚拟地址
varno	地址的范围(应在通讯数据类型范围内)
name	变量名
attr	变量的特征(通常为+X)
program	主程序的虚拟地址
	对于全局变量为"!0000"
lemax_len	最大长度(理论容量)

下面给出了用于说明应用中所使用的 C 语言函数的功能块的结构 :

@USP,<nb\_usp>  
#<va>,<name>  
#...

nb_usp	本块中 C 语言函数的个数
va	C 语言函数的虚拟地址
name	C 语言函数名

下面给出了用于说明应用中调用 C 语言函数的功能块的结构：

```
@FBINSTANCES,<nb_fb>
#<va>,<inst_name>,<fb_name>
#...
```

nb\_fb        本块中 C 语言功能块的实例数  
va           C 语言功能块实例的虚拟地址  
inst\_name   C 语言功能块实例名  
fb\_name     C 语言功能块实例参数名

WSHP0ADV -TOPIC5

### A.26.6 ISaGRAF "LARGE" (WDL) workbench 的极限值

各种对象所使用的 ISaGRAF 工作台有以下极限值。当然，由于所使用的计算机的配置（如可使用的内存和硬盘空间的大小）和 ISaGRAF 系统（如可使用的存储器、其他硬件和软件资源）的原因，还有许多其他实用的极限值。以下是一些不能超过的极限值。

对于一个项目：

对象	最大值	说明
程序个数	255	包括主程序、子程序和链接程序
层级数	20	

安装于工作台中的项目个数仅受硬盘内可用空间的限制。

对于名称：

对象	最大值	说明
项目	8 个字符	
程序	8 个字符	
变量	16 个字符	注释行可达 60 个字符
已定义的字标	16 个字符	
已定义的等式	255 个字符	注释行可达 60 个字符
转换表	16 个字符	
变量表	16 个字符	
函数/功能块(库)	8 个字符	适用与 C 语言函数、C 语言功能块或由 IEC 语言编写的函数
功能参数(库)	16 个字符	适用与 C 语言函数、C 语言功能块或由 IEC 语言编写的函数
输入/输出板	8 个字符	
输入/输出板的配置		8 个字符
输入/输出板的 OEM 参数		16 个字符
变换函数	8 个字符	

编辑 ( 对于一个程序 ):

对象	最大值	说明
SFC 代码行	600	
SFC 代码列	20	
SFC 程序步数	4095	指对于整个项目，包括程序步、初始化步、开始步和结束步。
SFC 转换	4095	指对于整个应用
LD/FBD 代码编辑	200 列 2000 行	指在地址单元上的编辑区域的大小 t
快速 LD 代码编辑	无限制	由 PC 的容量确定
IL 标记	251	在同一个 IL 程序内
文本编辑	40KBytes	

对于代码(对于一个项目):

对象	最大值	说明
布尔变量	65535	
模拟变量	65535	包括整型变量和实型变量
计时器	65535	
信息变量	65535	

以上给出的布尔变量、模拟变量和通讯变量的最大极限数包括了全部的内部变量、输入和输出变量。也包括了所有由编辑器配置产生的隐含的临时变量。在代码编辑器中同时编辑的变量个数(同一类型、同一范围中的变

量)不能超过16000个。根据PC机配置的不同,这个限制可能小于16000个。如果一种类型的变量总数超过4095个,应用程序就不能在ISaGRAF V1.21或更早的版本下运行。如果一种类型的变量数超过4095个,适用网络地址的标准"Modbus"总线也不能使用。

已定义的字节	4095	同一个表中(相同的范围)
已定义的字节	255	同一个程序中
转换表	127	在应用程序中使用
一个转换表中的点数	32	在同一个转换表中定义

对于输入/输出连结:

对象	最大值	说明
I/O 板数目	256	在同一个应用程序中定义 (输入/输出板或综合装置)

包括输入/输出板或综合装置在内的I/O板的总数不能超过256块

IO 通道数	128	在同一块输入/输出板上
--------	-----	-------------

对于库文件:

对象	最大值	说明
函数(IEC 代码.)	255	安装在同一个库文件中
功能块		

(IEC 代码.)	255	安装在同一个库文件中
C 语言函数	255	安装在同一个库文件中
C 语言功能块	255	安装在同一个库文件中
功能块		
实例	4095	同一个应用程序中的相同类型
的功能块		
函数输入参数	31	用于 C 语言函数和用 IEC 语言
编写的函数		
功能块参数	32	在输入参数和输出参数之间随
意分布		
		但至少需要一个输出参数
变换函数	128	安装在同一个库文件中
IO 配置函数	255	安装在同一个库文件中
IO 板数	255	安装在同一个库文件中
综合 IO 装置数.	255	安装在同一个库文件中
IO 板 oem 参数	16	

## B. 语言参考

## B.1 项目结构

ISaGRAF 的项目包括数个称作**程序**的编程单元。项目中的各个程序按树状结构连接在一起。各个程序可以用**SFC**、**FC (流程图)**、**FBD**、**LD**、**ST**、**LT**、图形语言或符号语言中的任何一种来描述。

### B.1.1 程序

**程序**是一个逻辑编程单元，这个单元可描述数据处理过程中**变量**之间的执行情况。程序或者描述**顺序**执行情况，或者描述**循环**执行情况。循环程序在每个目标系统循环中执行。顺序程序的执行过程或者遵从**SFC**语言的动态规则，或者遵从**FC**语言的动态规则。

各种程序以层次树的形式连接在一起。控制系统激活位于层次树顶端的程序。子程序（位于层次树较下层的程序）由其上层程序激活。下列的图形语言或文字语言可用于描述程序：

用于高层编程的**顺序功能图 (SFC)**

用于高层编程的**流程图 (FC)**

用于循环的复杂操作的**功能块图 (FBD)**

仅用于布尔操作的**梯形图 (LD)**

用于一些循环操作的**结构化文本 (ST)**

用于低层操作的**指令表 (IL)**

在一个程序中，除了LD语言和FBD语言可以在一张图中混用外，不能几种语言混用。

### B.1.2 循环执行、顺序执行

程序按层次可分为四个主要的程序段或程序组：

**Begin** 在每个循环开始时执行的程序  
**Sequential** 遵从SFC或FC动态规则的程序  
**End** 在每个循环结束时执行的程序  
**Functions** 非专用子程序集

**Begin**或**End**程序用以描述程序的循环执行情况，并与时间无关。**Sequential**程序描述顺序执行情况，在程序的顺序执行中，时间变量明显与基本操作同步。每次循环开始时，控制系统执行**Begin**主程序。每次循环结束时，控制系统则执行**End**主程序。**Sequential**程序则根据SFC或FC语言的动态规则执行。

**Function**程序是一些可由同一项目的其他程序调用的子程序。**Functions**程序也可以调用本段中的其他程序。

**Sequential**主程序和子程序应采用**SFC或FC语言**编写。循环(**Begin**和**End**)程序不能用**SFC或FC语言**编写。各段程序都可以有一个或多个子程序。**Sequential**程序可以有一个或多个**SFC或FC**子程序(按照各自的编程语言)。功能段的子程序不能用**SFC或FC语言**编写。

**Begin** 程序一般用于描述输入设备的初始操作，以建立高层过滤变量，这些变量经常被 **Sequential** 段的程序使用。**End 段** 程序一般用于描述在向输出设备发送变量值之前的安全操作，这些变量也在 **Sequential** 段中使用。

### B.1.3 SFC 子程序和 FC 子程序

**Sequential** 段 **SFC** 程序可以控制其他 **SFC** 程序。后面的这些低层程序称为 **SFC 子程序**。**SFC 子程序** 是一个可以由主程序启动、停止，冻结或重新启动的并程序。主程序和子程序必须使用 **SFC** 语言编写。**SFC** 子程序可以有其局部变量和定义字。

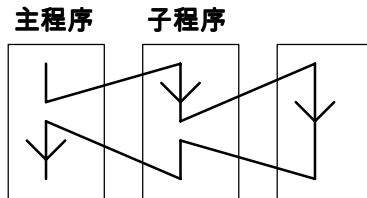
当 **SFC** 主程序启动 **SFC** 子程序时，主程序将 **SFC 标记**（激活）置于子程序的各个初始步中。该条命令为 **GSTART** 语句。当 **SFC** 主程序停止 **SFC** 子程序时，主程序将清除全部存在于子程序各步中的 **SFC** 标记。该条命令为 **GKLL** 语句。

当 **SFC** 主程序冻结 **SFC** 子程序时，主程序清除子程序中的全部 **SFC** 标记，并将标记保留在内存中。该条命令为 **GFREEZE** 语句。当 **SFC** 主程序重新启动被冻结的 **SFC** 子程序时，主程序则恢复全部的子程序被冻结时所清除的全部 **SFC** 标记。该条命令为 **GRST** 语句。

**Sequential** 段 **FC** 程序可以控制其他 **FC** 子程序。**FC** 主程序在 **FC** 子程序运行期间被封锁（等待）。因此 **FC** 主程序和 **FC** 子程序不能同时运行。

### B.1.4 函数和子程序

子程序或函数由其主程序起动。子程序或函数运算结束前，主程序暂时停止执行。



各段程序都可以有一个或多个子程序。一个子程序只能属于一个主程序。子程序可以具有局部变量和局部定义。除 **SFC** 和 **FC** 语言之外的其他语言都可以用来描述子程序。**Functions 段** 程序是一些可由本项目的其他程序调用的子程序，与其他子程序不一样的是：它们不为一个主程序所专有。**Functions 段** 程序可由本段的其他程序调用，函数可存于库文件中。

**警告：** ISaGRAF 系统在函数调用时不支持 **递归**。当 **Functions 段** 程序被自身调用或被 **Functions 段** 程序调用的子程序再次调用时，将会出现运行时间错误提示。

**警告：** 一个函数或子程序不能“存储”其局部变量的变量值，函数或子程序也不能被引用，因此它们也不能调用功能块。

每次子程序调用和返回参数，必须采用**同一符号和唯一名字**，以明确定义子程序的接口。为了支持ST语言约定，返回参数必须与子程序具有相同名字。

下表中表示各种不同语言在子程序中如何确定返回参数的值：

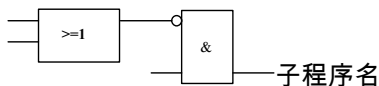
**ST:** 由名字确定返回参数  
(与子程序相同的名字):

子程序名 := <表达式>;

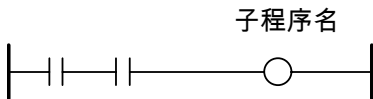
**IL:** 在序列结尾的当前结果的值(IL寄存器)存返回参数

```
LD    10
ADD   20 (* 返回参数的值 = 30 *)
```

**FBD:** 使用名字作为返回参数：

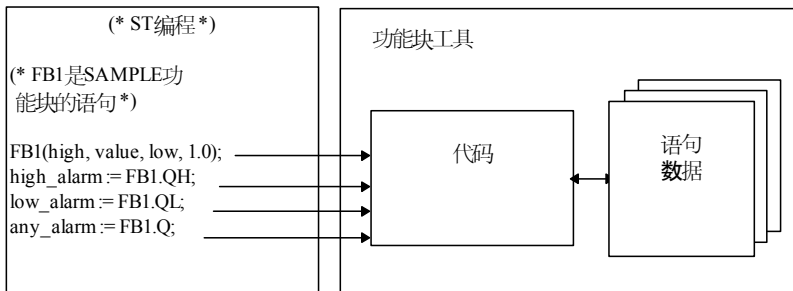


**LD:** 用线圈符号作为返回参数的名字：



### B.1.5 功能块

功能块可用LD、FBD、ST或IL语言。功能块可以被引用。这意味，每个实例都可以复制功能块的局部变量。当程序调用功能块时，是在调用功能块的实例：即调用同一个语句，但所用的数据是实例的数据。实例的变量值从一个循环存到另一个循环。



#### 警告：

- 用IEC语言编写的功能块不能调用其他功能块：即instanciation结构仅可管理自己块内的局部变量。以下标准功能块不能在IEC功能块中使用：

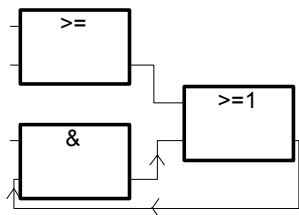
SR, RS, R\_Trig, F\_Trig, SEMA, CTU, CTD, CTUD, TON, TOF, TP, CMP, StackInt, AVERAGE, HYSTER, LIM\_ALARM, INTEGRAL, DERIVATE, BLINK, SIG\_GEN

- 由于相同的原因，不能使用正负触点和线圈、置位或复位线圈等功能。

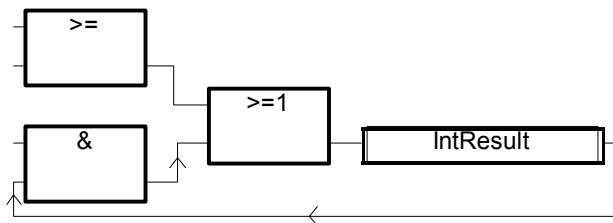
- 启动、停止计时器的TSTART和TSTOP函数不能用于3.0x版本的功能块中，可用于3.20版本之后。

- 当需要在功能块内进行循环时，在做循环之前，必须使用局部变量。请看如下例子：

这样不会工作:



这是正确的:



## B.1.6 程序描述语言

程序可由下列图形语言或文字语言进行描述：

用于高层编程的 **顺序功能图** (SFC)

用于高层编程的 **流程图** (FC)

用于循环的复杂操作的 **功能块图** (FBD)

仅用于布尔操作的 **梯形图** (LD)

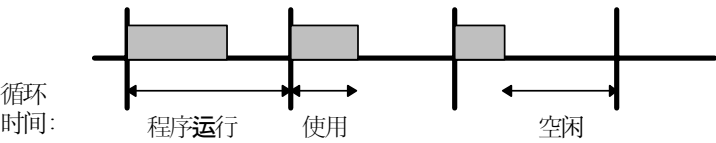
用于一些循环操作的 **结构化文本** (ST)

用于低层操作的 **指令表(IL)**

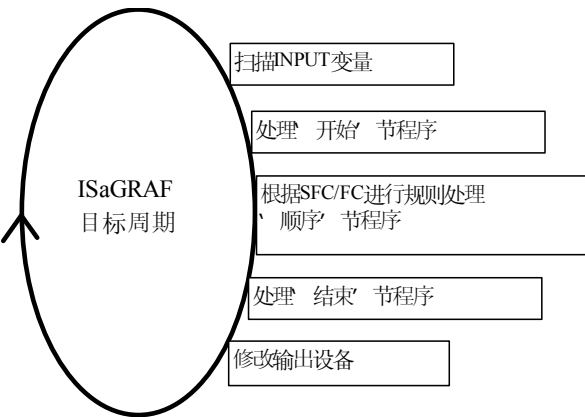
在同一个程序不能混用几种语言。当建立程序时，首先要选定描述该程序的语言，以后不能再改变。唯一的例外是在一个独立的程序中混合使用FBD和LD语言。

**B.1.7 执行规则**

ISaGRAF 是一个 **同步** 系统。所有操作均由时钟触发。时钟的基本持续时间称为循环时间。



在目标周期中的基本运行如下图：



该系统还可以：

- 保证输入变量在一个周期内保持同样的值。
  - 保证输出设备在一个周期内修改不超过一次。
  - 对于不同程序使用的相同全局变量能够安全地工作。
- 估计并控制整个应用程序的响应时间。

## B.2 公共对象

ISaGRAF 编程数据库中有一些主要特性和公共对象。这些对象可以用于由 SFC、FC、FBD、LD、ST 或 IL 语言中的任何一种语言编写的程序。

### 基本类型

常数表达式

### 变量

### 注释

### 定义字

### B.2.1 基本类型

用于一个程序(该程序可由任何一种语言编写)的常数、表达式、变量应该在类型上有明显区别。跟在图形操作和文字语句之后的应是类型说明。下面是编程对象可以使用的基本类型：

<b>布尔型:</b>	逻辑值(真或假)
<b>模拟型:</b>	整型或实型(浮点)连续值
<b>计时器型:</b>	时间值
<b>信息型:</b>	字符串

注意：计时器型的值应小于一天，并且不能用于存储日期值。

### B.2.2 常数表达式

常数表达式与类型有关。同一种符号表示法不能用于表示不同类型的常数表达式。

#### B.2.2.1 布尔常数表达式

只有两种布尔常数表达式：

真	等同于整型值 1
假	等同于整型值 0

"真"和"假"这两个关键字是大小写无关的。

#### B.2.2.2 整型模拟常数表达式

整型常数表达式是带符号长整型值(32位)：

从 **-2147483647** 到 **+2147483647**

整型模拟常量可以表示为以下**基本类型**中的一种。整型常量应以一个标明其类型的前缀开头：

<i>基本类型前缀</i>	<i>举例</i>
十进制数 (none)	-908
十六进制数	"16#" 16#1A2B3C4D
八进制数 "8#"	8#1756402
二进制数 "2#"	2#1101_0001_0101_1101

在二进制整型常量中，下划线('\_')用来分开数字组，下划线没有特别的意义，只是用于提高常数表达式的可读性。

### B.2.2.3 实型模拟常数表达式

实型模拟常数表达式既可用**十进制**来写，也可用**科学计数法**来写。**十进制小数点**('.')将整数和小数部分分隔开。十进制小数点用来区分实型常数表达式和整型常数表达式。科学计数表示法用字母'E'或字母'F'将**对数的尾数**部分和**指数**部分分开。采用科学计数法的实数的指数部分必须是带符号的整型值，数值在**-37**到**+37**。下面是实型模拟常数表达式的例子：

```
3.14159      -1.0E+12
+1.0  1.0F-15
-789.56      +1.0E-37
```

表达式"**123**"不是实型常数表达式。正确的实型常数表达式应为"**123.0**"。

### B.2.2.4 计时器常数表达式

计时器常数表达式表示从**0秒**到**23小时59分59秒999毫秒**的时间值。系统许可的最小时间单位是毫秒。用常数表达式的标准时间单位有：

**小时**        字母“h”跟在小时数后边

<b>分钟</b>	字母“ m” 跟在分钟数后边
<b>秒</b>	字母“ s” 跟在秒数后边
<b>毫秒</b>	字母“ ms” 跟在毫秒数后边

计时器常数表达式必须以 **"T#"** 或 **"TIME#"** 的前缀开头。前缀和单位字母都是大小。可以有一些单位是不显示的。下面是计时器常数表达式的例子：

**T#1H450MS**     1 小时, 450 毫秒

**time#1H3M** 1 小时, 3 分钟

表达式“ 0” 不是时间值, 而是模拟常数。

#### **B.2.2.5 信息字符串常数表达式**

字符串 或 信息常数表达式表示一个字符串。字符串应包括在一对单引号中，例如：

**'THIS IS A 信息'**

警告：单引号不能用在字符串常数表达式之内。一个字符串常数表达式必须写在程序源代码的一行之中。表达式的长度，包括空格，不能超过 255 个字符。

空字符串常数表达式可以用中间无空格或制表符的一对单引号来表示。

**" (\* 这是一个空字符串 \*)**

在一个字符串常数表达式中，以('\$')开头后随其它专用字符用来表示一个不可打印的字符。

字符	含义	ASCII码	举例
\$\$	'\$' 字符	16#24	'I paid \$\$5 for this'
\$'	上撇号	16#27	'Enter \$'Y\$' for YES'
\$L	进给一行	16#0a	'next \$L line'
\$R	回车	16#0d	' Ilo \$R He'
\$N	新一行	16#0d0a	'This is a line\$N'
\$P	新一页	16#0c	'lastline \$P first line'
\$T	制表符	16#09	'name\$Tsize\$Tdate'
\$hh (*) 任意字符		16#hh	'ABCD = \$41\$42\$43\$44'

(\*) “hh” 是用于表示的字符的十六进制 ASCII 码值。

### B.2.3 变量

变量可以是一个局部变量或全局变量。局部变量只能供一个程序使用。全局变量可供本项目的任何程序使用。

变量名必须符合下列规则：

- 变量名不能超过 16 个字符
- 第一个字符必须是字母
- 后面的字符可以是字母、数字或下划线

### B.2.3.1 保留关键字

下面是保留关键字一览表。这些关键字不能用来命名程序、变量或“C”函数或函数块：

A	ANA, ABS, ACOS, ADD, ANA, AND, AND_MASK, ANDN, ARRAY, ASIN, AT, ATAN,
B	BCD_TO_BOOL, BCD_TO_INT, BCD_TO_REAL, BCD_TO_STRING, BCD_TO_TIME, BOO, BOOL, BOOL_TO_BCD, BOOL_TO_INT, BOOL_TO_REAL, BOOL_TO_STRING, BOOL_TO_TIME, BY, BYTE,
C	CAL, CALC, CALCN, CALN, CALNC, CASE, CONCAT, CONSTANT, COS,
D	DATE, DATE_AND_TIME, DELETE, DINT, DIV, DO, DT, DWORD,
E	ELSE, ELSIF, EN, END_CASE, END_FOR, END_FUNCTION, END_IF, END_PROGRAM, END_REPEAT, END_RESSOURCE, END_STRUCT, END_TYPE, END_VAR, END_WHILE, ENO, EQ, EXIT, EXP, EXPT,
F	FALSE, FEDGE, FIND, FOR, FUNCTION,
G	GE, GFREEZE, GKILL, GRST, GSTART, GSTATUS, GT,
I	IF, INSERT, INT, INT_TO_BCD, INT_TO_BOOL, INT_TO_REAL, INT_TO_STRING, INT_TO_TIME,
J	JMP, JMPC, JMPCN, JMPN, JMPNC,
L	LD, LDN, LE, LEFT, LEN, LIMIT, LINT, LN, LOG, LREAL, LT, LWORD,
M	MAX, MID, MIN, MOD, MOVE, MSG, MUL, MUX,
N	NE, NOT,

O OF, ON, OPERATE, OR, OR\_MASK, ORN,  
P PROGRAM  
R R, REDGE, READ\_ONLY, READ\_WRITE, REAL,  
REAL\_TO\_BCD, REAL\_TO\_BOOL, REAL\_TO\_INT,  
REAL\_TO\_STRING, REAL\_TO\_TIME, REDGE, REPEAT,  
REPLACE, RESSOURCE, RET, RETAIN, RETC, RETCN,  
RETN, RETNC, RETURN, RIGHT, ROL, ROR,  
S S, SEL, SHL, SHR, SIN, SINT, SQRT, ST, STN, STRING,  
STRING\_TO\_BCD, STRING\_TO\_BOOL, STRING\_TO\_INT,  
STRING\_TO\_REAL, STRING\_TO\_TIME, STRUCT, SUB,  
SYS\_ERR\_READ, SYS\_ERR\_TEST, SYS\_INITALL,  
SYS\_INITANA, SYS\_INITBOO, SYS\_INITTMR,  
SYS\_RESTALL, SYS\_RESTANA, SYS\_RESTBOO,  
SYS\_RESTTMR, SYS\_SAVALL, SYS\_SAVANA,  
SYS\_SAVBOO, SYS\_SAVTMR, SYS\_TALLOWED,  
SYS\_TCURRENT, SYS\_TMAXIMUM, SYS\_TOVERFLOW,  
SYS\_TRESET, SYS\_TWRITE, SYSTEM,  
T TAN, TASK, THEN, TIME, TIME\_OF\_DAY, TIME\_TO\_BCD,  
TIME\_TO\_BOOL, TIME\_TO\_INT, TIME\_TO\_REAL,  
TIME\_TO\_STRING, TMR, TO, TOD, TRUE, TSTART, TSTOP,  
TYPE,  
U UDINT, UINT, ULINT, UNTIL, USINT,  
V VAR, VAR\_ACCESS, VAR\_EXTERNAL, VAR\_GLOBAL,  
VAR\_IN\_OUT, VAR\_INPUT, VAR\_OUTPUT,  
W WHILE, WITH, WORD,  
X XOR, XOR\_MASK, XORN

所有以下划线('\_')开始的关键字是内部关键字，它们不能被用于文本语句中。

### B.2.3.2 可直接显示的变量

ISaGRAF 允许用户在源程序中使用**可直接显示的变量**来描述一个空通道。空通道是那些没有与已经声明的输入/输出变量相连接的通道。可直接显示的变量的变量名总是以字符“**%**”开始。

以下是用于单一插板通道的可直接显示的变量的命名规定。“**s**”是插板上的板槽号。“**c**”是通道号。

**%IXs.c** 布尔型输入插板空通道

**%IDs.c** 整型输入插板空通道

**%ISs.c** 信息型输入插板空通道

**%QXs.c** 布尔型输出插板空通道

**%QDs.c** 整型输出插板空通道

**%QSS.c** 信息型输出插板空通道

以下是用于综合设备通道的可直接显示的变量的命名规定。“**s**”是设备的板槽号。“**b**”是在综合设备内单一插板索引号。“**c**”是通道号。

**%IXs.b.c** 布尔型输入插板空通道

**%IDs.b.c** 整型输入插板空通道

**%ISs.b.c** 信息型输入插板空通道

<b>%QXs.b.c</b>	布尔型输出插板空通道
<b>%QDs.b.c</b>	整型输出插板空通道
<b>%QSS.b.c</b>	信息型输出插板空通道

例子如下:

%QX1.6            1号插板(布尔型变量输出)的第六通道  
%ID2.1.7 2号设备上1号插板(整型变量输入)的第七通道

可直接显示的变量不能是**实型**变量。

### B.2.3.3 布尔型变量

布尔变量是**逻辑型变量**，布尔变量的取值可以是布尔值(**真或假**)中的一种。布尔变量通常用于布尔表达式。布尔变可以有以下**属性**：

**内部**: 可由程序修改的内存变量

**输入**: 连接到输入设备的变量(由系统刷新)

**输出**: 连接到输出设备的变量

**警告**: 当声明布尔变量后，在调试过程中可以定义字符串取代'真'和'假'值。这些字符串不能用于程序中，除非它们作为'**定义字**'输入。

#### B.2.3.4 模拟型变量

模拟型变量是**连续变量**。模拟型变量可以是带符号的整型或实型(浮点)值。模拟型变量的格式为：

**整型** 32位带符号整型：从 **-2147483647** 到 **+2147483647**

**实型** 标准IEEE 32位浮点值(单精度)

1位符号位+23位底数+8位指数

实型模拟指数值不能小于 **-37** 或大于 **+37**。模拟变量可以有以下**属性**：

**内部** 可由程序修改的内存变量

**输入** 连接到输入设备的变量(由系统刷新)

**输出** 连接到输出设备的变量

注意：当实型变量与I/O设备相连接时，相应的I/O驱动器处理等价的整型值。

警告：整型和实型模拟变量或常数表达式不能在同一个模拟表达式中混用。

#### B.2.3.5 计时器型变量

计时器型变量意味着**时钟**或者**计数器**。该变量有时间值，并通常被用于时间表达式Z中。计时器变量的值不能为负数，并且不能超过**23小时59分钟59秒999毫秒**。计时器变量按32位字存储，并为正的毫秒数。

**警告：** 计时器变量不能有输入或输出属性。

计时器变量可由 ISaGRAF 系统自动刷新。当计时器处于启动状态时，其数值根据目标系统的实际时钟自动增加。ST 语言的下列语句可用来控制计时器：

**TSTART**     开始自动刷新计时器

**TSTOP**      停止自动刷新计时器

#### B.2.3.6 信息字符串型变量

信息或字符串型变量中包含字符串，字符串的长度在数据处理过程中可以被改变。信息变量的长度不能超过该变量声明时指定的容量(最大长度)。信息容量最大为 255 个字符。信息型变量可以有以下属性：

**内部** 可由程序修改的内存变量

**输入** 连接到输入设备的变量(由有系统刷新)

**输出** 连接到输出设备的变量

字符串型变量可以有标准 ASCII 表(即从 0 到 255 的 ASCII 码)中的任何字符。字符串可以有零字符(0)。标准 ISaGRAF 库中的一些 "C" 函数不能正确地处理含有零字符(0)的信息型变量。

### B.2.4 注释

程序注释可以自由地插入**ST**、**IL**等语言中。注释必须以特殊字符“**(\***”开始，以特殊字符“**\*)**”结束。注释可以插在**ST**程序任何位置上，并且可以为一行或多行。

下列是注释的例子:

```
counter := ivalue; (*指定主计时器*)
(*这是一个用两行
表示的注释*)
c := counter (*用户可以将注释放在任何位置上*) +
base_value + 1;
```

不能使用嵌套注释。这表明字符“**(\*\***”不能用于程序注释中。

警告: **IL**语言中，只能在指令行的最后加注程序的注释。

### B.2.5 定义字

ISaGRAF系统允许用户重新定义常数表达式、具有真和假值的布尔表达式、关键字或复杂**ST**表达式。为了做到这点，必须给予相应表达式一个**标识符**名。例如：

<b>YES</b>	是	<b>真</b>
<b>PI</b>	是	<b>3.14159</b>
<b>OK</b>	是	<b>(自动模式 AND NOT (闹钟))</b>

当定义这样的等式时，该标识符可用于ST程序的任何位置上以取代相关的表达式。以下是一个使用“定义字”ST编程的例子：

```
If OK Then
    angle := PI / 2;
    isdone := YES;
End_if;
```

定义字可以是局部定义字、全局定义字或公共定义字。

局部定义字仅可用于一个程序。

全局定义字可用于一个项目的任何程序中。

公共定义字可用于任何项目的任何程序中。

注意公共定义字可用档案管理器分别存贮。

**警告：**当相同标识符被两个不同的ST等式定义两次时，使用最后定义的表达式。例如：

定义:	打开	is	真
	打开	is	假

意义:	打开	is	假
-----	----	----	---

定义字必须符合下列规则：

- 标识符名不能超过16个字符

- 第一个字符必须是**字母**
- 后面的字符可以是**字母、数字**或下划线

警告: 定义字在定义时不能使用定义字，例如，用户不能定义如下：

**PI**        is        **3.14159**

---

**PI2**       is       **PI\*2**

使用常量或变量和运算符写下完整的等式：

**PI2**       is       **6.28318**

## B.3 SFC 语言

顺序功能流程图 (SFC) 是用于描述顺序操作的一种图形语言。顺序用一组定义明确的步表示，这些步通过转换链接起来。每个转换有布尔条件。这些步内的动作可以用其它语言 (ST, IL, LD 和 FDB) 来进行详细的描述。

### B.3.1 SFC 图的主要格式

一个 SFC 程序是由步和转换组成的图形组。通过定向链接把步和转换链接在一起。多个连接的链接用于表示分支和会合。完整程序的某些部分可以被分离并在主流程图中用一个单符号表示，称为宏步。SFC 的基本图形规则如下：

- 步的后面不能紧随另一个步
- 转换的后面不能紧随另一个转换

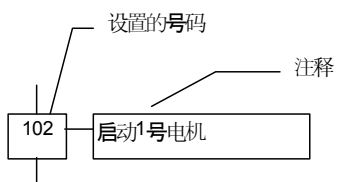
WSHP1SFC-SFC1

### B.3.2 SFC 的基本元件

SFC 的基本元件 ( 图形符号 ) 是：步和初始步，转换，定向链接，以及跳转到一个步。

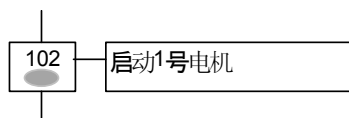
步和初始步

一个步用一个单独的正方形表示。每一个步用写在步的正方形符号内的编号标定，与步符号相链接的矩形内写入步的主要描述。该描述是随意的注释(不属于编程语言的一个部分)。上述信息被称之为步的第一层：

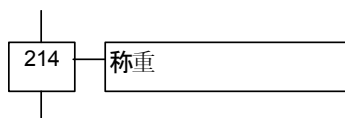


运行时，用一个标记显示这个步是激活的：

激活的步：

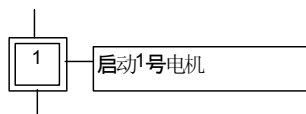


未激活的步：



SFC 程序初始情况用初始步来表示。一个初始步有双边框的图形符号。当启动程序时，标记被自动地放在每个初始步内。

初始步：



一个 SFC 程序必须至少含有一个初始步。以下是步的标志。这些域可被用于任何其它的语言中。:

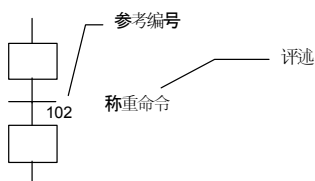
GSnnn.x.....步的动作(布尔值)

GSnnn.t.....步的激活时间(时间值)

(在这里 nnn 是步的参考编号)

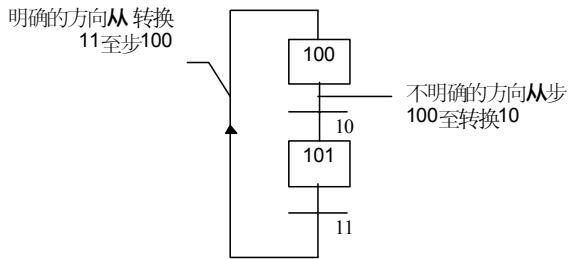
## 转换

转换用一条与链接线交叉的短的水平线段来表示。每一个转换被一个写在其旁边的编号设定。在转换符号的右侧写上有关这个转换的主要描述。该描述是随意的注释(不属于编程语言的部分)。以上信息被称之为转换的第一层:



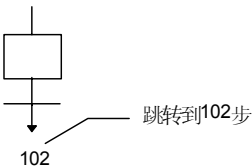
## 定向链接

用单线链接步和转换。它们是定向链接。在没有给出明确的方向时，链接的方向是从顶部到底部。

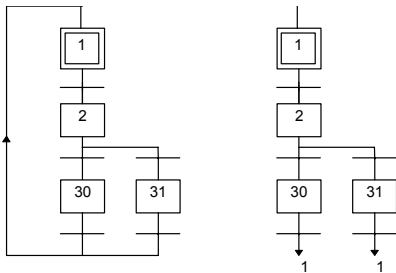


### 跳转到一个步

跳转符号可用于表明从转换至步的连接，此时不必画出连接线。跳转符号用目标步的编号来标记



跳转符号不能用于表示从一个步到一个转换的连接。跳转举例 - 下面的流程图是等效的：

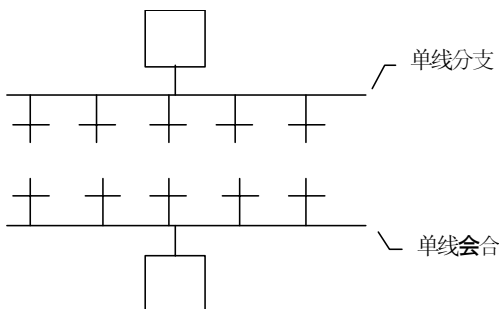


### B.3.3 分支与会合

分支是从一个SFC符号(步或转换)到许多其它的SFC符号的多路连接的链接。会合是从一个以上的SFC符号到一个其它符号的多路连接的链接。分支与会合可以是单线或双线

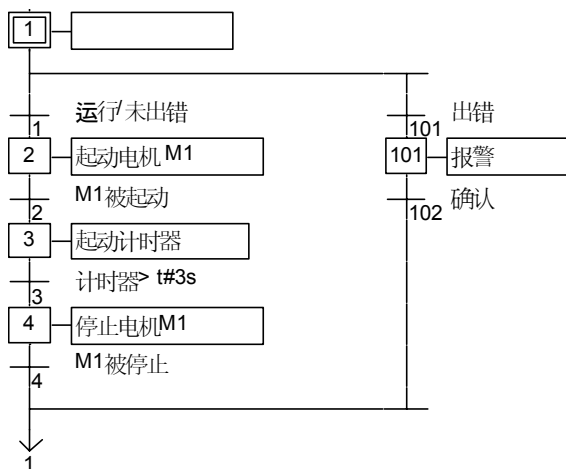
#### 单线分支

单线分支是从一步到许多转换的一个多路链接。它允许激活标记进入许多支路中的一个支路。一个单线会合是从许多转换到同一个步的一个多路链接。一个单线会合通常用于把SFC支路组合起来，它们起始在一个单线分支上。单线支路和会合以单根水平线为表示。



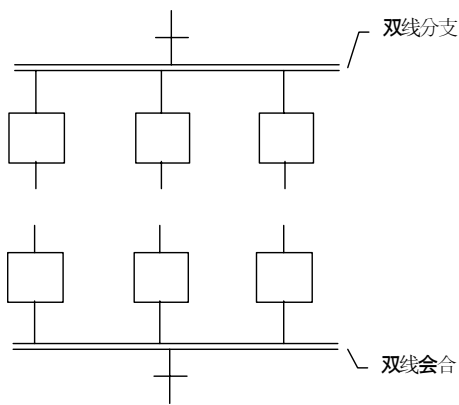
**警告：**在一个单线分支的开始，不同的转换所附带的条件并不正好是唯一的。其唯一性必须在转换条件中得到详尽的描述，以保证在运行时，在分支的一个支路中只有一个标记前进。下面是单线分支和会合的一个例子：

(\*含有单线分支和会合的 SFC 程序\*)



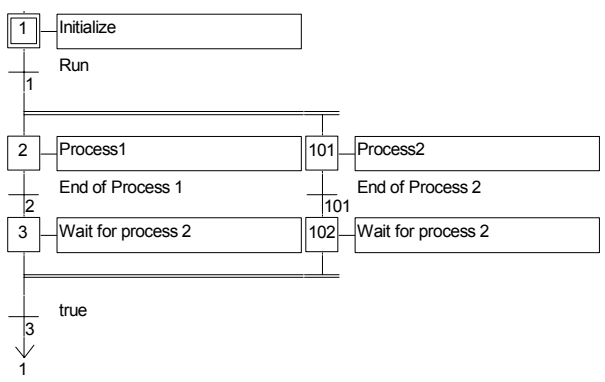
### 双线分支

一个双分支是从一个转换到许多步的一个多路链接。它适合于过程的并行操作。一个双线会合是从许多步到同一个转换的一个多路链接。一个双线会合通常用于把 SFC 支路组合起来，曾在在一个双线分支上起始这些支路。双线分支和会合用双水平线表示。



双线分支和会合的例子：

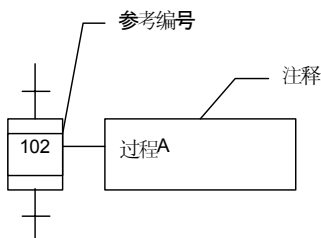
(\*含有双线分支和会合的 SFC 程序 \*)



B.3.4 宏步

一个宏步是对多个步和转换组合的表示法。宏步的主要部分在同一个 SFC 程序的另一个地方被单独描述。在

SFC 的主流程图中，宏步以一个单独的符号出现。以下是用于一个宏步的符号：



宏步符号的参考编号是该宏步体中第一步的参考编号。宏步的主要部分必须从一个起始步开始，并以一个终止步束，流程图必须是独立的。一个起始步没有上面的链接(没有反向转换)。一个终止步没有下面的链接(没有正向转换)。一个宏步符号可被放在另一个宏步的主要部分中。

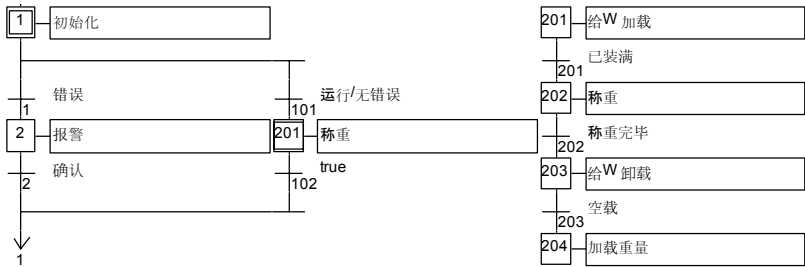
警告：因为宏步是由多个步和转换特别组合，所以在—个SFC程序中不能多次使用相同的宏步。

宏步举例:

(\*主流程图\*)

(\*宏

步的主要部分\*)



### B.3.5 步内动作

一个SFC步的第二层是对步在激活期间所执行的动作的详细描述。该描述利用SFC的文字特性和其它语言如结构化文本(ST)来完成。动作的基本类型是：

布尔动作

脉冲动作

非存储动作

SFC动作

几个动作(有相同的或不同的类型)可在一个相同的步内描述。能使用任何其它语言的特性是：

从一个动作调用功能和功能块

IL语言的规则

布尔动作

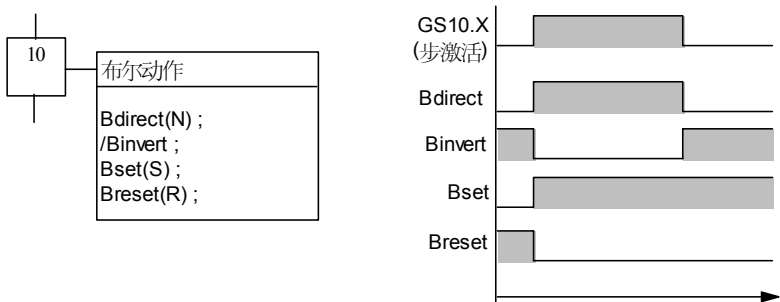
布尔动作是用步的激活赋值给一个布尔变量。布尔变量可以是一个输出或一个内部量。每当步的激活启动或停止时，它就被赋值。以下是基本布尔动作的语法：

<布尔\_变量>(N);     把步的激活信号赋值给变量  
 <布尔\_变量>;相同的结果(N标记是可选的)  
 /<布尔\_变量>;       把步激活的求反信号赋值给变量

当步被激活时，其它一些特性可供用于置位或复位一个布尔变量。以下是置位或复位布尔动作的语法：

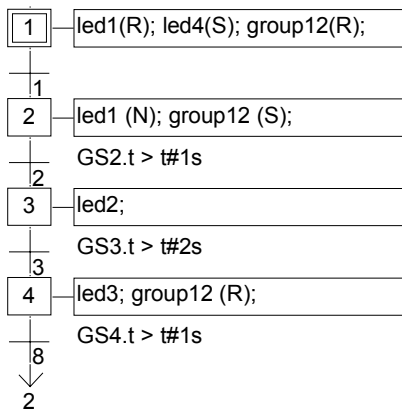
<布尔\_变量>(S);     当步的激活信号变为 TRUE 时，把变量置位为 TRUE  
 <布尔\_变量>(R);     当步的激活信号变为 TRUE 时，把变量复位为 FALSE

布尔变量必须是一个输出或一个内部的量。下面的 SFC 编程导致以下动作：



布尔动作举例：

(\* 使用布尔动作的 SFC 程序 \*)



### 脉冲动作

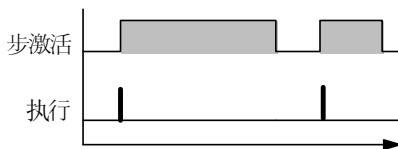
一个脉冲动作是一个 ST 或 IL 指令表，这些指令在步激活状态仅被执行一次。写指令时，应遵照 SFC 语法：

ACTION (P) :

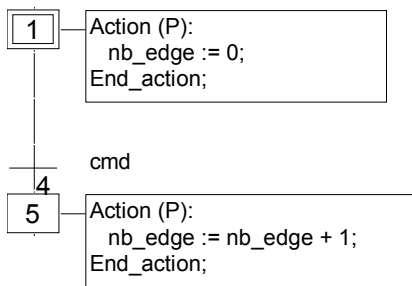
(\* ST 语句 \*)

END\_ACTION ;

以下显示一个脉冲动作的结果:



脉冲动作举例:



### 非存储动作

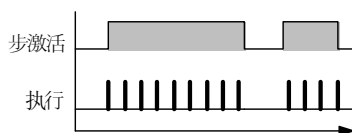
一个非存储(正常的)动作是一个ST或IL指令,这些指令在步的整个激活期间每次循环时被执行。写指令应遵照下面的SFC语法:

ACTION (N) :

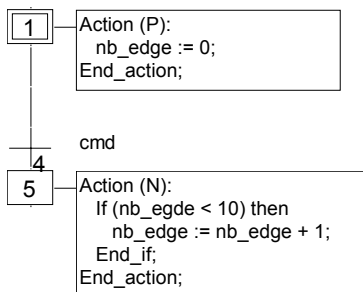
(\* ST 语句 \*)

END\_ACTION ;

以下是一个非存储动作的结果:



非存储动作举例:



### SFC 动作

一个 SFC 动作是一个 SFC 的子程序，它根据步的激活信号的变化启动或停止。一个 SFC 动作可以有 N (非存储)，S (设置)，或 R (复位) 三种判定。以下是 SFC 基本动作的语法：

<child\_prog> (N);      当步变为激活时，启动子程序；当步变为非激活时，停止子程序。

<child\_prog>;      相同的结果 (N 标记是可选项)

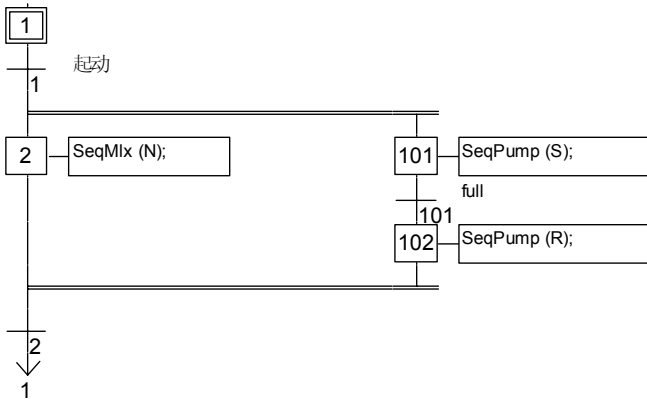
<child\_prog> (S);      当步变为激活时，启动子程序；当步变为非激活时，不动作。

<child\_prog> (R);      当步变为激活时，停止子程序；当步变为非激活时，不动作。

规定为一个动作的 SFC 程序必须是当前正在编辑的节目的一个 SFC 子程序。值得注意的是，用于一个 SFC 动作的 S (设置) 或 R (复位) 判定确实与 GSTART 和 GKILL 语句的效果相同，该语句被编写在一个 ST 脉冲动作中。

下面是一个 SFC 动作的例子。父 SFC 程序命名为 Father，它有两个 SFC 子程序，被命名为 SeqMlx 和 SeqPump 父 SFC 程序的编程如下：

(\* 使用 SFC 动作的 SFC 程序\*)



从一个动作中调用功能和功能块

子程序、功能或功能块(用 ST、IL、LD 或 FBD 语言)或“C”函数和“C”功能块可以直接从一个 SFC 动作块中调用，它们根据下面的语法编写：

用于子程序，功能和“C”函数：

ACTION (P) :

result := sub\_program ( ) ;

END\_ACTION;

或

ACTION (N) :

result := sub\_program ( ) ;

```
END_ACTION;
```

对于用“C”或者ST、IL、LD、FBD语言编写的功能块:

```
ACTION (P) :
```

```
    Fbinst(in1, in2);
```

```
    result1 := Fbinst.out1;
```

```
    result2 := Fbinst.out2;
```

```
END_ACTION;
```

或

```
ACTION (N) :
```

```
    Fbinst(in1, in2);
```

```
    result1 := Fbinst.out1;
```

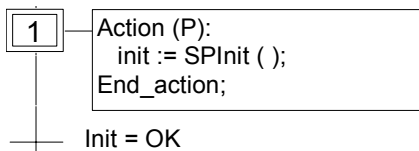
```
    result2 := Fbinst.out2;
```

```
END_ACTION;
```

详细的语法可以在ST语言一节中找到。

在动作块中调用一个子程序的例子：

(\*在一个动作块中调用一个子程序的SFC程序\*)



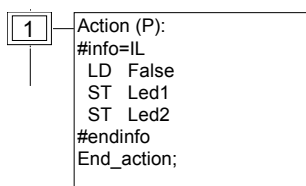
IL 语言规则

指令表 (IL) 的编程可根据下面的语法直接输入到一个 SFC 动作块中：

```
ACTION (P) :      (* or N *)  
#info=IL  
  <instruction>  
  <instruction>  
  ....  
#endinfo  
END_ACTION;
```

特殊关键词“# info = IL”和“# endinfo”必须完全用上面的方法输入，这两个关键词与大小写相关。在关键词的后面或前面不能插入空格或制表符。下面是一个动作块内的一个 IL 程序的例子：

(\* 含有一个 IL 程序的 SFC 程序，IL 程序在动作块内 \*)



### B.3.6 与转换相连的条件

每一个转换都有一个布尔表达式相连，布尔表达式给出了该转换的条件通常是用 ST 语言或 LD 语言 (快捷 LD 编辑

器)来表示。这是转换的第二层。不过也可使用其它结构：

ST 语言规则

LD 语言规则

IL 语言规则

从一个转换调用功能

**警告：**当没有表达式附加给转换时，隐含条件为 TRUE。

#### B.3.6.1 ST 语言规则

结构化文本 (ST) 语言可用于描述一个转换条件。整个表达式必须为布尔类型，并且以一个分号结尾，其语法为：

```
< boolean_expression > ;
```

表达式可以是一个 TRUE 或 FALSE 的常量表达式，信号输入或内部的布尔变量，或是产生布尔值的变量组合。下面是以 ST 编程的转换例子：

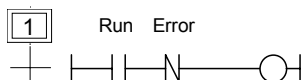
(\* 用 ST 编写转换的 SFC 程序 \*)



## LD 语言规则

梯形图 (LD) 语言可用于描述转换的条件。梯形图由带有一个线圈的一个梯级组成。线圈值代表转换值。

下面是用 LD 编写转换的例子：



## IL 语言规则

指令表 (IL) 语言可直接用于描述一个 SFC 转换，其语法规则为：

```
#info=IL
```

```
    <instruction>
```

```
    <instruction>
```

```
    ....
```

```
#endinfo
```

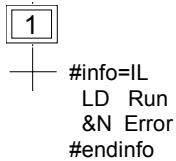
位于 IL 程序末端的当前结果 (IL 寄存器) 所包含的值，使附加给转换的条件产生结果：

current result = 0                      ➔        条件是 FALSE

current result <> 0                   ➔        条件是 TRUE

特殊关键词“ # info = IL” 和“ #endinfo” 必须完全用上面的方法输入，这两个关键词与大小写相关。在关键词的后面 或前面不能插入空格或制表符。下面是为转换用 IL 编程的例子：

(\* 用 IL 编写转换的 SFC 程序 \*)



从一个转换中调用函数

为了判断一个转换的条件，可以调用任何子程序或功能（用 FBD、LD、ST 或 IL 语言编写），或“C”函数，其语法为：

< sub\_program > ( ) ;

子程序或函数返回的值必须是布尔量，并使条件产生结果：

return value = 0	➔	条件是 FALSE
return value <> 0	➔	条件是 TRUE

下面是在转换中调用子程序的例子：

(\* 转换中调用子程序的 SFC 程序\*)



### B.3.7 SFC 的动态规则

SFC 语言的五个动态规则是:

#### 初始状态

初始状态的特征在初始步中体现，初始步被定义为在操作起始时处于激活状态。每个 SFC 程序中必须至少有一个初始步。

#### 清除一个转换

一个转换或为有效或为无效。当所有与相应的转换符链接的之前的步处于激活状态时，被称为有效，否则就是无效。除下列情况以外，一个转换不能被清除:

- 该转换处于有效状态，和
- 有关的转换条件为真(“ true” )。

#### 激活步状态的改变

一个转换的清除同时导致紧跟下面的步为激活状态，并导致前面紧接的步为非激活状态。

#### 转换的同步清除

双线用于表示必须同时被清除的转换。如果这样一些转换被单独显示，那么前面一些步 (GSnnn.x) 的激活状态可被用于表示它们的条件。

#### 一个步同时被激活和被解除激活

如果在操作期间，一个步同时被激活和被解除激活，那么激活将获得优先。

### B.3.8 SFC 程序的层次结构

ISaGRAF 系统能够描述 SFC 程序垂直结构。SFC 程序被按有层次的层状结构组织。每个 SFC 程序可以控制(起动，停止...)其它的 SFC 程序。这样的程序被称为 SFC 程序的子程序，它受 SFC 程序控制。采用一种"父子"关系把各个 SFC 程序链接到主层状结构中。

层次结构所包含的基本规则是：

- 没有父程序的 SFC 程序被称为“父” SFC 程序
- 应用程序启动时，由系统激活父 SFC 程序
- 程序可以有几个子程序
- 子程序不能有一个以上的父程序
- 子程序仅由其父程序控制
- 程序不能控制它自己的子程序的子程序

父 SFC 程序为了控制其子程序而能够采取的基本操作为：

- |    |  |
|----|--|
| 启动 | (GSTART) 启动子程序：激活它的每一个初始步。<br>该子程序的子程序不能自动地启动。 |
| 停止 | (GKILL) 通过解除每个激活步的激活状态使子程序停止。该子程序的所有子程序也停止。    |

冻结	(GFREEZE) 暂停执行程序 (解除每个激活步的激活状态并暂停转换的计算)，存储程序步的状态，以便能重新启动程序。子程序的所有下级程序也被冻结。
重新启动	(GRST) 通过重新启动所有的暂停步，重新启动一个被冻结的 SFC 程序。该程序的子程序不能自动地重新启动。
获得状态	(GSTATUS) 获得一个子程序的当前状态 (激活、非激活或冻结)。

## B.4 流程图

**流程图 (FC)** 是用于描述 **顺序操作** 的一种图形语言。流程图由 **动作** 和 **测试** 组成。在动作和测试之间是代表数据流的 **定向链接**。动作和测试可用 ST, LD, IL 语言描述。可以从动作和测试中调用任何语言 (SFC 除外) 的函数和功能块。一个流程图程序可以调用另一个流程图程序。被调用的 FC 程序是正在进行调用的 FC 程序的 **子程序**。

### B.4.1 FC 基本部件

下面是流程图语言的图形部件：

FC 图的开始

FC 图的结束

FC 流程的链接

FC 动作

FC 条件

FC 子-程序

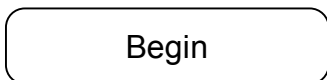
FC 输入输出特有的动作

FC 连接符

FC 注释

### B.4.2 FC 图的开始

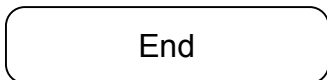
一个“开始”符必须出现在一个流程图程序的开头。它是唯一的并且不能省略。当它被激活时，它表示流程图的初始状态。下面是一个“开始”符的图形:



“开始”符总是与流程图的其它对象处于连接状态(在底部)。如果从“开始”到其它目标没有画连接线，则该流程图无效。

### B.4.3 FC 图的结束

一个“结束”符必须出现在一个流程图程序的结尾。它是唯一的，并且不能省略。有时可能没有给“结束”符画连线(永远循环的流程图)，但不管怎样，在流程图的底部仍要画出“结束”符。当程序的执行被完成时，它表示流程图的最后状态。下面是一个“结束”符的图形:



一般来说，“结束”符与流程图的其它目标处于连接状态(在顶部)。一张流程图与“结束”对象没有连接(永远循环的流程图)。在这种情况下，在流程图的底部仍可看到“结束”目标。

#### B.4.4 FC 流程的链接

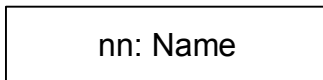
一个流程的**链接**是表示图形两点之间信息流的一条线。一个链接总是以一个箭头结束。下面是一个流程的链接：



两个链接不能与同一个源连接点相连。

#### B.4.5 FC 动作

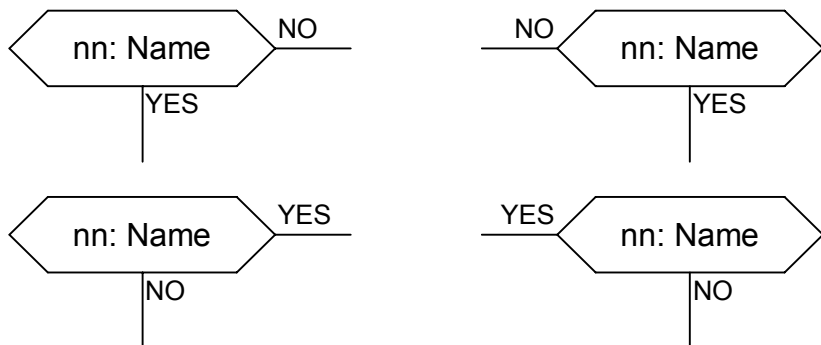
一个**动作**符号代表将被完成的动作。一个动作用一个号码和一个名称来识别。下面是一个"动作"符号的图形：



同一个流程图的两个不同对象不能使用同一名称或逻辑号。一个动作的编程语言可以是ST, LD 或IL. 一个动作总是与链接相连，其中一个链接表示到达动作，另一个链接表示从动作开始。

#### B.4.6 FC 条件

一个**条件**表示一个布尔量的**测试**。一个条件用一个号码和一个名称来识别。根据对所附带的ST, LD 或IL表达式所作的判定，程序的流程被指向"YES" 或"NO" 路径。下面是一个条件符号可能具有的图形：



同一个流程图的两个不同对象不能使用同一名称或逻辑号。

一个测试的编程或是

- 一个ST语言的表达式，或者是

- LD语言的一个单独梯级，在唯一的一个线圈上不附带符号。

或者是

- 几条IL指令。IL寄存器（或当前结果）被用于评判条件。

当用ST文本编程时，可供选择的一种方式是，在表达式后跟一个分号。用LD编程时，那个唯一的线圈表示条件值。一个条件等于：

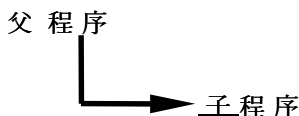
- 0 或 FALSE 把流程引向 NO

- 1 或 TRUE 把流程引向 YES

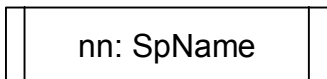
一个测试总是与一个到达的链接相连，并且必须对两个向前的连接进行定义。

### B.4.7 FC 子程序

本系统能够对FC程序的垂直结构进行描述。FC程序是一个**有层次的树形结构**。每一个FC程序可以调用另一个FC程序。这样的程序被称为FC程序的**子程序**，子程序由FC程序来调用。调用FC子程序的FC程序被称为**父程序**。用一种"父-子"关系把FC程序链接在一起，形成一个有层次的树状主干结构：



流程图中的一个**子程序**符号表示对一个流程图子程序的调用。正在调用子程序的FC程序被暂停执行，直到子程序执行完毕。一个流程图的子程序用一个号码和一个名称来识别，作为另外的程序、功能或功能块。下面是一个"子程序调用"符号的图形：



同一个流程图的两个不同的目标不能使用相同的逻辑编号。  
FC层次结构所包含的基本规则是：

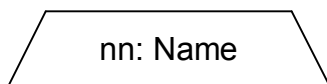
- 没有上级程序的FC主程序
- 当应用程序启动时，由系统激活FC主程序
- 一个程序可以有几个子程序
- 一个程序的子程序不能有一个以上的主程序

- 一个子程序仅由其主程序调用
- 一个程序不能调用它自己的子程序的子程序

同一个子程序可以在主流程图中出现数次。一个流程图调用子程序表示该子流程图被完全执行。在子流程图运行期间，暂停执行主流程图。子程序正在调用的功能块所必须遵循的连接规则与为FC动作所制定的规则相同。

### B.4.8 FC 输入 / 输出 特定动作

一个**输入/输出特定动作**符号表示所要完成的动作。与其它动作一样，一个**输入/输出特定动作**也是用一个号码和一个名称来识别。在标准动作和输入/输出特定的动作上使用相同的语义。输入/输出特定动作的目的仅仅是使流程图更加易读，并集中在流程图的不可移动部分。使用输入/输出特定动作是一个供用户选择的功能。下面是一个“输入/输出特定动作”符的图形：

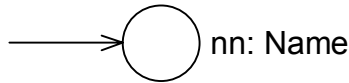


输入/输出特殊块与标准动作具有完全相同的特点，包括相同的属性，相同的ST, LD 或IL编程及相同的连接规则。

### B.4.9 FC 连接符

**连接符**用于表示未画出的图形两点之间的一个链接。一个连接符用一个圆来表示并被连接到流程的源点。连接符图形是在适当的边上（取决于数据流的方向）通过对目标点（一般是

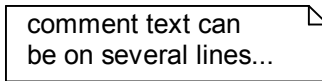
目标符的名称) 的识别来完成的。下面是一个连接符的标准图形:



一个连接符总是把一个定义的流程图符号作为目标。目标符用它的逻辑编号来识别。

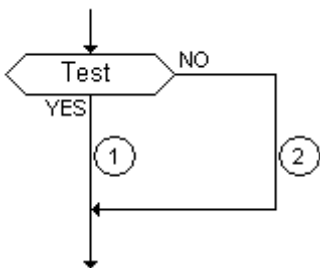
#### B.4.10 FC 注释

一个**注释**框包含有文本，该文本没有流程图语义上的意义。在流程图文件窗口上可以把文本插入到未使用空间的任何地方，并用于给程序提供资料。下面是一个“注释”符的图形：



#### B.4.11 FC 复杂结构举例

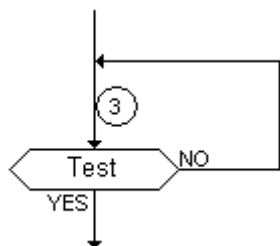
这部分给出了**复杂结构**的例子，可以把它们定义在流程图中。这种结构是链接在一起的基本对象的组合。



#### IF / THEN / ELSE

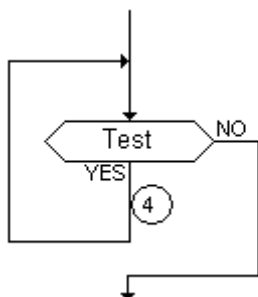
(1) 用于插入 "THEN" 动作的地方

(2) 用于插入 "ELSE" 动作的地方



## REPEAT / UNTIL

(3) 用于插入循环动作的地方



## WHILE / DO

(3) 用于插入循环动作的地方

### B.4.12 FC 动态行为

对流程图的**执行**可作如下解释：

- 开始符号获得一个目标周期
- 结束符号获得一个目标周期并结束流程图的执行。在程序到达该符号后不再执行流程图的动作。
- 在同一目标周期中，如果流程遇到一个已经被执行的动作或判定，流程将被中断。在上述情况下，流程将继续下一个周期。

注意：与 SFC 相反，一个动作不是一个稳定的状态。在动作符号高亮显示期间不重复执行指令。

### B.4.13 FC 校验

除了附加的 ST, LD 或 IL 编程外，其它一些**语法规则**也适用于流程图本身。下面是主要规则表：

- 所有符号的所有“连接”点必须用线连接。（可以省略与“结束”符号的连接）
- 所有符号必须链接在一起（不应出现分隔部分）
- 所有连接符应该有有效的目标

其它小的语法错误告知如下：

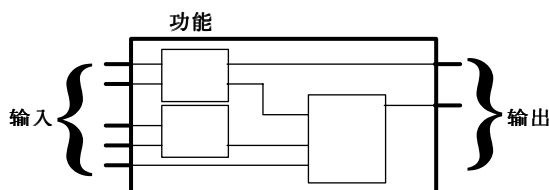
- 空动作（没有编程）被认为是运行时间表内的步
- 空测试（没有编程）被认为“总是真”

## B.5 FBD 语言

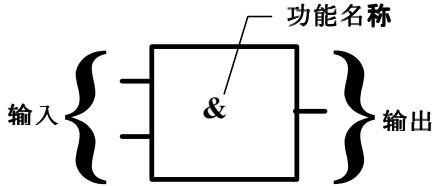
功能块图 (FBD) 是一种图形语言。它允许程序员用 ISaGRAF 库中现有的功能编制复杂的程序，并在框图区域中用线把它们连接在一起。

### B.5.1 FBD 图的主要格式

FBD 图描述一个在输入变量和输出变量之间的功能。一个功能被视为基本功能块的一个集合。连线把输入和输出变量与块相连，功能块的输出又可以与另一功能块的输入相连。



FBD 程序所运行的一个完整的功能是由 ISaGRAF 库中标准的基本功能块组成的。功能块有确定的输入和输出连接点数量。功能块用矩形表示。输入端与矩形的左边框连接，输出端与矩形的右边框连接。基本功能块在它的输入与输出之间完成一个功能。其名称写在矩形符号内。方框的每个输入或输出都有明确的类型。



FBD 程序的输入变量必须与功能块的输入相连。变量的类型与相关输入所要求的类型相同。FBD 程序的输入可以是常数表达式、任意内部的或输入或输出的变量。

FBD 程序的输出变量必须与功能块的输出相连。变量类型与相关输出要求的同类。FBD 程序的输出可以是任意内部的或输出的变量，或者是程序名（只就子程序而言）。当输出是当前所编辑的子程序名时，它代表子程序的返回值（返回到调用程序）。

用连接线把输入和输出变量及功能块的输入和输出连接在一起。单独的线可用于连接框图的两个逻辑点：

- 输入变量和功能块的输入
- 功能块的输出和另一个功能块的输入
- 功能块的输出和输出变量

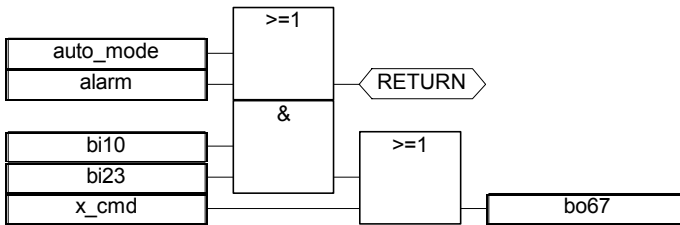
连接具有方向性，这就是说，连接线是从左端向右端传输数据。，连线的左右端具有相同的数据类型。

多路右连接可用于把信息从左端向右端的每一端口进行传输。所有的连接端口必须具有相同的类型。

### B.5.2 RETURN 语句

关键词 "<RETURN >" 可以出现在框图输出位置，它必须与一个功能块的布尔输出连接点相连。RETURN 语句代表程序的一个有条件的结束：如果与返回语句相连的方框的输出，其布尔值为 TRUE，那么程序的结尾（剩余部分）不被执行。

(\* 使用 RETURN 返回语句编写的 FBD 程序举例 \*)



(\* ST equivalence: \*)

```
If auto_mode OR alarm Then
    Return;
End_if;
bo67 := (bi10 AND bi23) OR x_cmd;
```

### B.5.3 跳转和标号

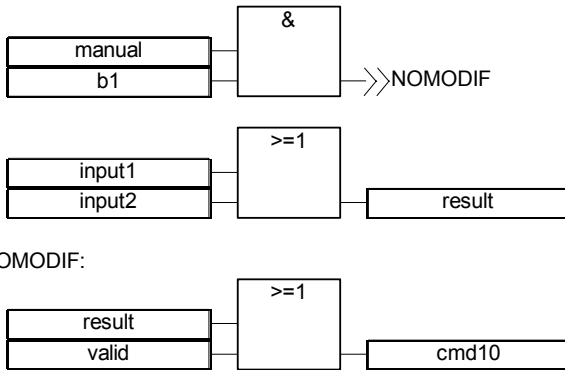
跳转和标号用于控制程序的执行。在一个跳转或标号符的右边不能连接其它。使用的标志如下：

>>LAB ..... 跳转到某个标号 (标号名为 "LAB")

LAB: ..... 某个标号的定义 (标号名为 "LAB")

如果跳转标号左边连接线的布尔状态值是“真”，那么程序的执行直接跳转到相应的标号符后面。

(\* 使用标号和跳转的 FBD 程序举例 \*)



(\* IL Equivalence: \*)

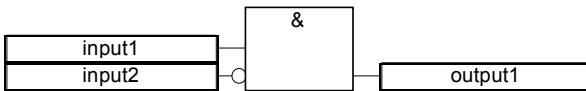
```

      ld      manual
      and     b1
      jmpc    NOMODIF
      ld      input1
      or      input2
      st      result
NOMODIF: ld      result
          and     valid
          st      cmd10
  
```

### B.5.4 布尔求反运算

一根右端与某个功能块的输入相接的连接线可用布尔求反运算结束。求反运算用小圆圈来表示。使用布尔求反运算时，连线左、右端的数据必须具有布尔类型。

(\* 使用标号和跳转的 FBD 程序举例 \*)



(\* ST equivalence: \*)

output1 := input1 AND NOT (input2);

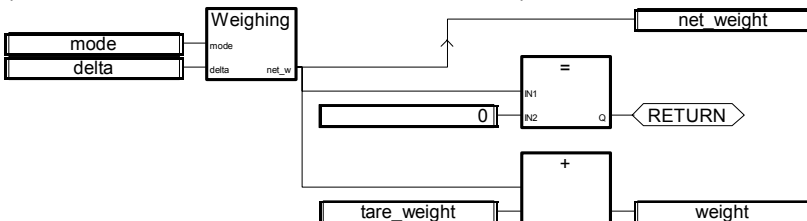
### B.5.5 调用 FBD 中的功能或功能块

FBD 语言能够调用子程序、功能或功能块。一个子程序，功能或功能块用一个功能方块图表示。方块内名称就是子程序或功能或功能块的名称。

对一个子程序或一个功能来说，其返回值是功能方块图的唯一输出。

功能块可以有多个输出。

(\* 使用子程序块的 FBD 程序举例 \*)



(\* ST Equivalence \*)

net\_weight := Weighing (mode, delta); (\* call sub-program \*)

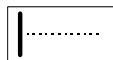
If (net\_weight = 0) Then Return; End\_if;

weight := net\_weight + tare\_weight;

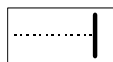
## B.6 LD 语言

梯形图 (LD) 是布尔方程式的一种图形表示法。它把触点 (输入自变量) 与线圈 (输出结果) 组合在一起。LD 语言通过把图形符号放入程序图表内的方法, 使得对测试的描述和对布尔数据的修改成为可能。LD 图表符号在程序图表内恰好组成了一个电气接点图。梯形图的左, 右侧被连接到垂直的电源线。

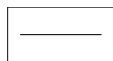
这些是一个梯形图的一些基本图形元件:



左侧垂直电源线



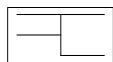
右侧垂直电源线



水平连接线



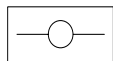
垂直连接线



多路连接线 (所有的线连接在一起)



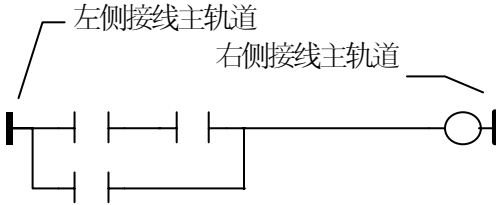
与一个变量相关的触点



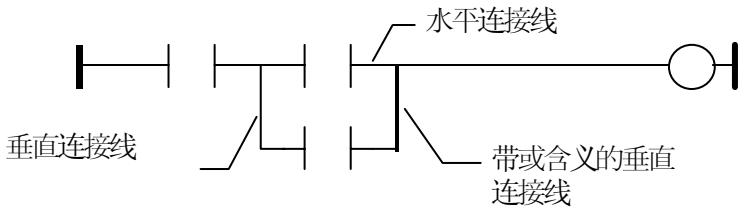
与一个输出变量或一个内部变量相关的线圈

### B.6.1 电源线连接线

一个梯形图的左右两端被两条垂直线所限定, 这两条垂直线被分别命名为左侧电源线和右侧电源线。



用连接线把梯形图的图形符号与电源线或其它符号连接起来。连接线是水平的或垂直的。



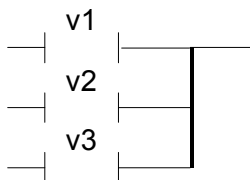
每条线段有一个布尔状态 FALSE 或 TRUE。所有直接连接在一起的线段都有相同的布尔状态。与左侧垂直电源线相连的任一水平线都为 TRUE 状态。

## B.6.2 多路连接

单根水平连接线的布尔状态在线的左右两端是相同的。把水平和垂直连接线组合在一起，就能建立多路连接。在一个多路连接中，各端口的布尔状态遵循逻辑规则。

一个左侧多路连接可把与一根垂直线的左侧相连的多根水平线和一根与垂直线右侧相连的线连接在一起。右端的布尔状态是所有左端口之间的逻辑或 (OR)。

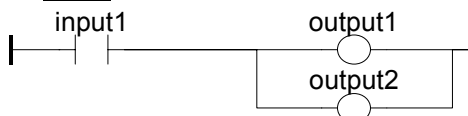
(\* 左侧 多路连接举例 \*)



(\* 右端状态是 (v1 OR v2 OR v3) \*)

一个右侧多路连接可把一根与垂直线的左侧相连的水平线和多根与垂直线右侧相连的线连接在一起。左端的布尔状态被传送到右侧的每一个端口。

(\* 右侧 多路连接举例 \*)



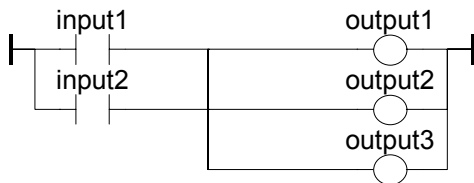
(\* ST equivalence: \*)

output1 := input1;

output2 := input1;

一个左侧和右侧的多路连接可把与一根垂直线的左侧相连的多根水平线和多根与垂直线右侧相连的线连接在一起。右侧每一个端口的布尔状态是所有左端口之间的逻辑或 (OR)

(\* 左右侧 多路连接举例 \*)



(\* ST Equivalence: \*)

output1 := input1 OR input2;

output2 := input1 OR input2;

output3 := input1 OR input2;

### B.6.3 基本的 LD 触点和线圈

::有以下几个符号可供用于输入触点

触点

反向触点

带上升沿检测的触点

带下降沿检测的触点

有以下几个符号可供用于输出触点

线圈

反向线圈

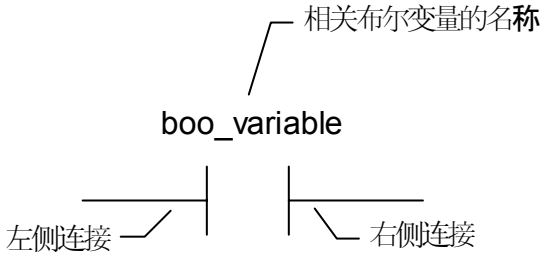
置位线圈

复位线圈

带上升沿检测的线圈

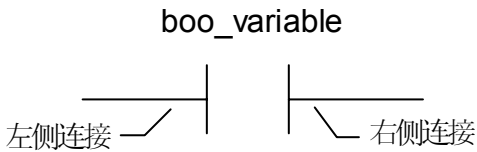
带下降沿检测的线圈

变量名称写在所有这些图形符号的上面:



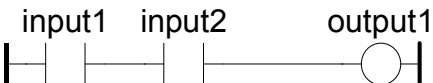
### 触点

一个触点 能在一个连接线状态和一个布尔变量之间实现布尔操作。



触点右侧连接线的状态是左连接线的状态和与触点有关的变量之间的逻辑与。

(\* 使用触点举例\*)

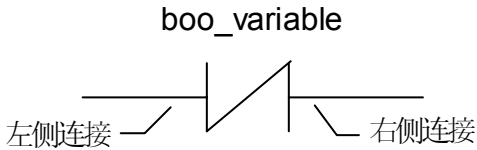


(\* ST Equivalence: \*)

output1 := input1 AND input2;

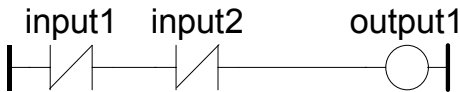
## 反向触点

一个反向触点能在一个连接线状态和一个布尔变量求反之间实现布尔操作。



触点右侧连接线的状态是左连接线的状态和与触点有关的变量值布尔求反之间的逻辑与。

(\* 使用 反向 触点举例 \*)

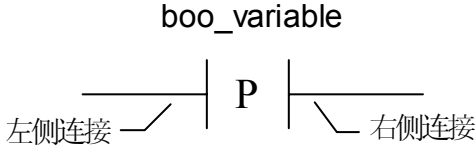


(\* ST Equivalence: \*)

output1 := NOT (input1) AND NOT (input2);

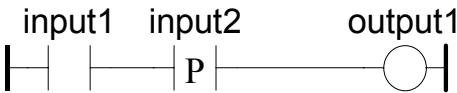
## 带上升沿检测的触点

这个触点(正向能在一个连接线状态和一个布尔变量的上升沿之间实现布尔操作。



当左侧连接线的状态为“真”时，并且有关变量的状态从“假”上升到“真”，触点右侧连接线的状态被设定为“真”。。在所有其它情况下，变量被复位为“假”

(\* 使用 上升沿 触点举例 \*)



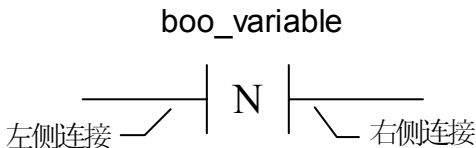
(\* ST Equivalence: \*)

`output1 := input1 AND (input2 AND NOT (input2prev));`

(\* input2prev is the value of input2 at the previous cycle \*)

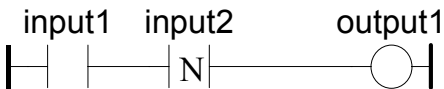
### 带下降沿检测的触点

这个触点(反向)能在一个连接线状态和一个布尔变量的下降沿之间实现布尔操作。



当左侧连接线的状态为“真”时，并且有关变量的状态从“真”下降到“假”，触点右侧连接线的状态被设定为“真”。。在所有其它情况下，变量被复位为“假”。

(\* 使用下降沿触点举例 \*)



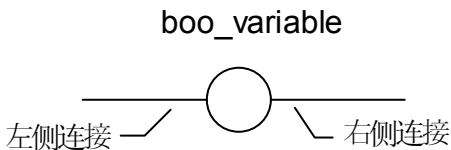
(\* ST Equivalence: \*)

```
output1 := input1 AND (NOT (input2) AND input2prev);
```

(\* input2prev is the value of input2 at the previous cycle \*)

## 线圈

线圈能实现一个连接线布尔状态的一个布尔输出。

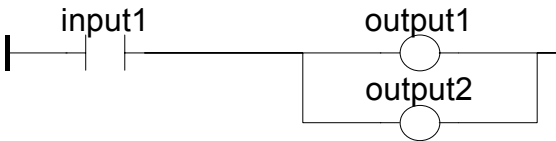


用左侧连接的布尔状态给有关的变量赋值。左侧连接的状态被传送到右侧连接。右侧连接可以与右侧的垂直电源线相连。

有关的布尔变量必须是输出或内部变量。

有关的名称可以是程序名(仅用于子程序)。该名称与子程序返回值所规定的名称一致。

(\* 使用线圈举例 \*)



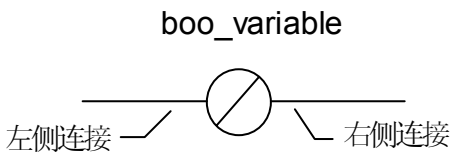
(\* ST Equivalence: \*)

```
output1 := input1;
```

```
output2 := input1;
```

## 反向线圈

反向线圈 能根据一个连接线状态的布尔求反实现一个布尔输出。

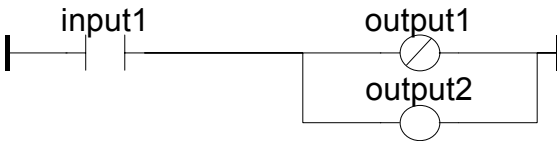


用左侧连接状态的布尔求反给有关的变量赋值。左侧连接的状态被传送到右侧连接。右侧连接可以与右侧的垂直电源线相连。

有关的布尔变量必须是输出或内部变量。

有关的名称可以是程序名(仅用于子程序)。该名称与子程序返回值所规定的名称一致。

(\* 使用反向线圈举例 \*)



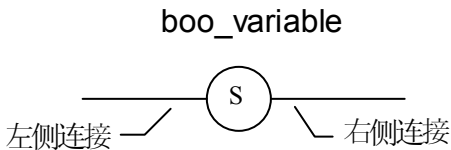
(\* ST Equivalence: \*)

```
output1 := NOT (input1);
```

```
output2 := input1;
```

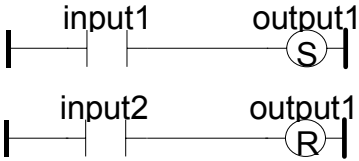
## 置位线圈

"置位"线圈可实现一个连接线布尔状态的一个布尔输出。



当左侧连接的布尔状态变成 TRUE 时, 有关的变量被设置为 TRUE。输出变量一直保持该数值, 直到由一个"复位"线圈产生一个相反的命令。左侧连接的状态被传送到右侧连接。右侧连接可以与右侧的垂直电源线相连。有关的布尔变量必须是输出或内部变量。

(\*使用“置位”和“复位”线圈举例\*)



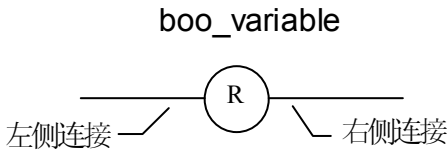
(\* ST Equivalence: \*)

```

IF input1 THEN
    output1 := TRUE;
END_IF;
IF input2 THEN
    output1 := FALSE;
END_IF;
  
```

## 复位线圈

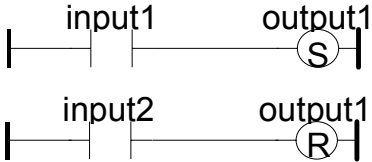
"复位"线圈可实现一个连接线布尔状态的一个布尔输出。



当左侧连接的布尔状态变成TRUE时，有关的变量被复位为FALSE。输出变量一直保持该数值，直到由一个“置位”线圈产生一个相反的命令。左侧连接的状态被传送到右侧连接。右侧连接可以与右侧的垂直电源线相连。

有关的布尔变量必须是输出或内部变量。

(\*使用“置位”和“复位”线圈举例\*)



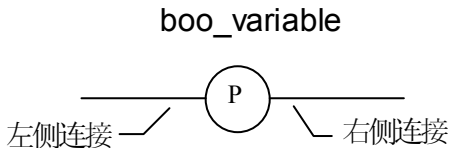
(\* ST Equivalence: \*)

```

IF input1 THEN
  output1 := TRUE;
END_IF;
IF input2 THEN
  output1 := FALSE;
END_IF;
  
```

### 带上升边缘检测的线圈

"正向"线圈可实现一个连接线布尔状态的一个布尔输出。这种类型的线圈只在快捷梯形图编辑器使用。

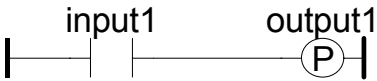


当左侧连接的布尔状态从 FALSE 上升到 TRUE 时，有关的变量被置为 TRUE。在所有其他情况下，输出变量复位为 FALSE。

左侧连接的状态被传送到右侧连接。右侧连接可以与右侧的垂直电源线相连。

有关的布尔变量必须是输出或内部变量。

(\*使用一个"正向"线圈举例\*)



(\* ST Equivalence: \*)

```
IF (input1 and NOT(input1prev)) THEN
```

```
    output1 := TRUE;
```

```
ELSE
```

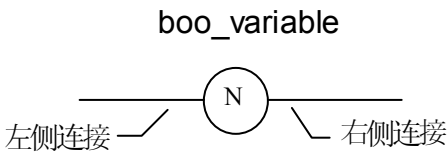
```
    output1 := FALSE;
```

```
END_IF;
```

(\* input1prev is the value of input1 at the previous cycle \*)

### 带下降边缘检测的线圈

"反向"线圈可实现一个连接线布尔状态的一个布尔输出。这种类型的线圈只在快捷梯形图编辑器使用。



当左侧连接的布尔状态从“真”下降到“假”时，有关的变量被置为“真”。在所有其他情况下，输出变量复位为“假”。左侧连接的状态被传送到右侧连接。右侧连接可以与右侧的垂直主线相接。

有关的布尔变量必须是输出或内部变量。

(\* 使用一个 "反向" 线圈举例 \*)



(\* ST Equivalence: \*)

```
IF (NOT(input1) and input1prev) THEN
```

```
    output1 := TRUE;
```

```
ELSE
```

```
    output1 := FALSE;
```

```
END_IF;
```

(\* input1prev is the value of input1 at the previous cycle \*)

## B.6.4 返回语句

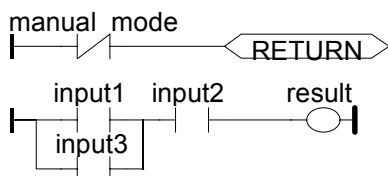
返回标号可被用作一个输出，它表示该程序的一个有条件的结束。在一个返回符号的右侧不能进行连接。

## 返回

如果左侧连接线的布尔状态为“真”，那么程序结束，不执行之后的程序行。

注意: 当LD程序是一个子程序时，为了设置返回值（返回到调用程序），它的名称必须与输出线圈有关。

(\* 使用返回符号举例 \*)



(\* ST Equivalence: \*)

```
If Not (manual_mode) Then RETURN; End_if;
```

```
result := (input1 OR input3) AND input2;
```

### B.6.5 跳转和标号

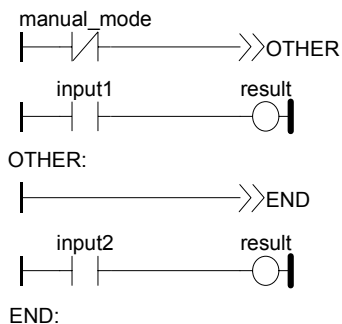
标号，有条件的和无条件的跳转符号，可被用于控制梯形图的执行。在标号和跳转符号的右侧不能进行连接。此时将使用下列符号：

>>LAB ..... 跳转到标号被称为 "LAB"

LAB: ..... 标号被称为 "LAB"

如果跳转符号的左侧连接的布尔状态为“真”，那么在标号符之后的程序被执行。

(\*使用跳转和标号符举例\*)



(\* IL Equivalence: \*)

	<code>ldn</code>	<code>manual_mode</code>
	<code>jmpc</code>	<code>other</code>
	<code>ld</code>	<code>input1</code>
	<code>st</code>	<code>result</code>
	<code>jmp</code>	<code>END</code>
<code>OTHER:</code>	<code>ld</code>	<code>input2</code>
	<code>st</code>	<code>result</code>
<code>END:</code>	(* end of program *)	

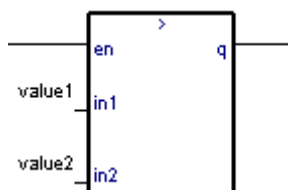
## B.6.6 LD 中的块

使用快捷LD编译器可把功能框与布尔线相连。一个功能实际上可以是一个操作符，一个功能块或一个功能。由于所有的

方块不总是有一个布尔输入和/或一个布尔输出，所以把方块插入梯形图时，会给方块接口附加新的参数EN，ENO。如果你使用FBD/LD编辑器，则不会加入EN，ENO参数，因为你可以连接具有所需类型的变量。

## EN” 输入

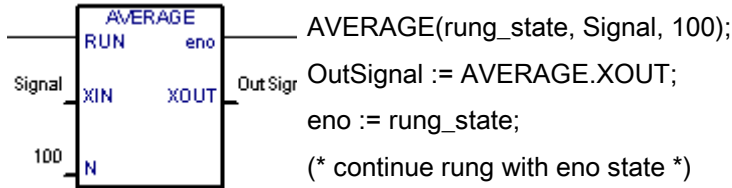
在某些操作符、功能或功能块上，第一个输入不是布尔数据类型。因为第一个输入必须总是与梯级相连，所以被称为“EN”的另一个输入被自动插入到第一个位置。只有EN输入为“真”时，块才被执行。下面是一个比较操作符的例子以及用ST语言表示的等效代码：



```
IF rung_state THEN
    q := (value1 > value 2);
ELSE
    q := FALSE;
END_IF;
(* continue rung with q state *)
```

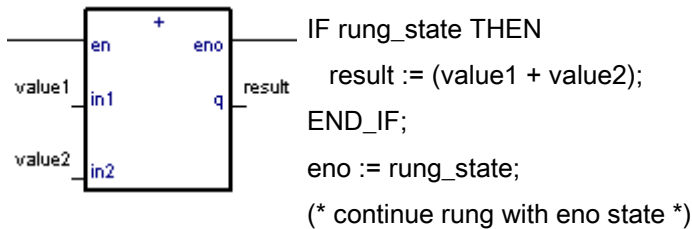
## ENO” 输出

在某些操作符、功能或功能块上，第一个输出不是布尔数据类型。因为第一个输出必须总是与梯级相连，所以被称为“ENO”的另一个输出被自动插入到第一个位置。ENO输出总是与方块的第一个输入处于同样的状态。下面是一个带有平均功能块的例子以及用ST语言表示的等效代码：



同时使用“EN”和“ENO”

在某些情况下需要同时使用EN和ENO。下面是一个带有运算操作符的例子以及用ST语言表示的等效代码：



## B.6.7 LD 中的 « 内线 » 功能块

按照 IEC 句法,“内线”功能块具有与其它功能块相同的定义以及 ISaGRAF 3 一般执行方法:

“内线”功能块具有输入和输出参数. 每一个输入或输出参数都被定义好数据类型. 输入和输出的总数不能超过 32.

“内线”功能块具有的本地变量(在 ISaGRAF 中定义的 )被复制到图形中每一个功能块中.

“内线”功能块的主要特性是,在其应用中,功能块的代码可以被复制到每一个应用 (每一个例子). 不像其它的功能块,呼叫系统 和数据传递机制不可以被用于“内线”功能块. 到达例子的呼叫被功能块的实际代

码所取代. 这个取代由 Quick LD 编码器来实现. 由于即时机制有编码器来完成, 对"内线" 功能块的编网功能没有限制,. 因此, "内线" 功能块可以叫做另一个 "内线" FB 或 一个标准 或"C" 功能块

准则 :

让我们想象一下具有一个输入, 一个输出及一个内部即时变量的功能块. 这个例子是一个边缘检测功能块.

Name:

FB1

Inputs:

IN (boolean) = input signal

Outputs:

Q (boolean) = set to TRUE only when IN changes from FALSE TO TRUE

Internal instance data (to be duplicated on each call):

PREV (boolean) = state of IN signal at the previous cycle

LD programming of the FB1 block:

```

      IN          PREV          Q
|----] [-----] \ [----- ( ) -|
      IN          PREV
|----] [----- ( ) -|

```

Equivalent code in ST language:

```

Q := IN and not PREV;
PREV := IN;

```

Below, is a Quick LD program that calls the function block:

```

      B1          +-----+          B2
|----] [--|  FB1  |----- ( ) -|
      +-----+
      B3          +-----+          B4
|----] [--|  FB1  |----- ( ) -|
      +-----+

```

按照输入和输出上的变量情况, "FB1" 代码将被复制 .. 当编辑母程序时, 以下是代码产生的 ST 等式 :

```
(* code of the first call *)  
B2 := B1 and not PREV1;  
PREV1 := B1;  
(* code of the second call *)  
B4 := B3 and not PREV2;  
PREV2 := B3;
```

按照这个例子所显示的,编辑器必须为每一个呼叫的 FB 本地变量配置内部变量 (在这个例子中是 PREV1 and PREV2 ). 由于内部变量的自动从新命名机制, 因此,在 « 内线 » 功能块的编网上没有限制.

当使用 "内线" FBs 来代替标准 FBs 时,应用代码(TIC 代码)的大小将增加. 同时,, "内线" FBs 提供更好的执行时间,这是因为当数据传输代码是不需要呼叫功能

The nesting of "In Line" FBs also supports the instantiation of edge detection contacts and coils (P and N), the use of "C" and standard function blocks, and local jumps and labels.

权限:

"内线" FBs 仅有 Quick LD .这意味着:

- "内线" FBs 必须在 Quick LD 上书写
- 所有呼叫"内线" FBs 的程序必须被写在 Quick LD 内.
- 在 SFCs and 流程图内的 Quick LD 可以呼叫 "内线" FBs

当呼叫 FBs 时受下列条款限制:

- "内线" FBs 可以呼叫其它 "内线" FBs
  - "普通" FBs 不能呼叫 "内线" FBs
  - "内线" FBs 不能呼叫"普通" FBs
  - 不允许重复呼叫
- ( 在 ISaGRAF 3.3 版本中"普通" 称着 FBs)

编辑时呼叫错误可被检测出来. 同时,当您运行 « 校对 » 或 « 做 » 指令编辑器将会按照呼叫曲线图自动保证 « 内线 » 功能块被编辑.

"内线" 功能块只能在一个项目中定义. ISaGRAF 库不支持这些功能块.

## B.7 ST 语言

ST (结构化文本) 是为自动化过程设计的一种高级结构化语言。这种语言主要用于完成那些不能简单地用图形语言来表示的复杂程序。在对步内动作和SFC语言中的转换条件进行描述时, ST是默认的语言。

### B.7.1 ST 主要语法

ST程序是ST的语句表。每条语句用一个分号(“;”)分隔符结束。在源代码中(变量标识符、常量、语言的关键词...)中所使用的名称文字用非激活分隔符(空格符、行末标记或制表符)分开或用激活分隔符分开, 激活分隔符有一个意义明确的含义(例如, “>”分隔符表示一个“大于”比较)。注释可被自由地插入文本中。一个注释必须用符号“(“开始和用符号“)”结束。每条语句用一个分号(“;”)分隔符结束。ST语句的基本类型是:

- 赋值语句 (变量 := 表达式;)
- 子程序或函数调用
- 功能块调用
- 选择语句 (IF, THEN, ELSE, CASE...)
- 循环语句 (FOR, WHILE, REPEAT...)
- 控制语句 (返回, EXIT...)
- 用于与其它语言如SFC进行链接的特殊语句

非激活分隔符可自由地被插入激活分隔符、常量表达式和标识符之间。ST的非激活分隔符是：空格符、制表符和行结束符。与讲究行格式的语言(如IL)不同，ST语言可在程序中的任何地方插入行的结束符。当使用非激活分隔符以提高ST程序的易读性时，应遵循以下规则：

- 在一行上不要写一条以上的语句
- 使用制表符对复杂的语句进行缩进处理
- 插入注释以提高行或段落的易读性

源程序可读性的举例：

可读性差

可读性好

<pre> imax := max_ite; cond := X12; if not(cond (* alarm *)) then return; end_if; for i (* index *) := 1 to max_ite do if i &lt;&gt; 2 then Spcall(); end_if; end_for; (* no effect if alarm *) </pre>	<pre> (* imax : number of iterations *) (* i: FOR statement index *) (* cond: process validity *) imax := max_ite; cond := X12; if not (cond) then     return; end_if;  (* process loop *) for i := 1 to max_ite do     if i &lt;&gt; 2 then         Spcall ();     end_if; end_for; </pre>
--	---

## B.7.2 表达式和括号

ST表达式是ST操作符和变量或常量操作数的组合。对每个单独的表达式(操作数与一个ST操作符组合)来说,操作数的类型必须是相同的。这个单独的表达式同它的操作数的类型是一样的,可被用在较复杂的表达式。例如:

(boo_var1 和 boo_var2)	为 BOO 布尔型
not (boo_var1)	为 BOO 布尔型
(sin (3.14) + 0.72)	为 实数模拟型
(t#1s23 + 1.78)	是一个无效的表达式

括号用于分隔表达式的各个分项，并清楚地安排操作的优先级。对一个复杂的表达式来说，当没有给出括号时，操作顺序由ST操作符之间的缺省优先级给出。例如：

$2 + 3 * 6$                       等于  $2+18=20$             因为乘法操作数有较高的优先级

$(2+3) * 6$                       等于  $5*6=30$             由括号给出优先级

警告: 在一个表达式中，可被嵌套的括号层数最多为8层。

### B.7.3 功能或功能块的调用

标准的ST调用功能可用于以下每个对象：

- 子程序
- 用IEC语言编写的库功能和功能块
- “C”功能和功能块
- 类型转换功能

调用子程序或函数

调用功能块

---

## 调用子程序或函数

名称: 用 IEC 语言或“C”语言编写的被调用的子程序或库函数的名称

含义: 调用一个 ST、IL、LD 或 FBD 子程序或函数或一个“C”函数并得到它的返回值。

语法: `<变量> := <subprog> (<par1>, ... <parN>);`

操作数: 返回值的类型和调用的参数必须遵循为子程序所定义的接口。

返回值: 由子程序返回的数值

子程序可以在任何表达式中被调用，也可以在一个 SFC 的转换中被调用。

例 1: 子程序的调用

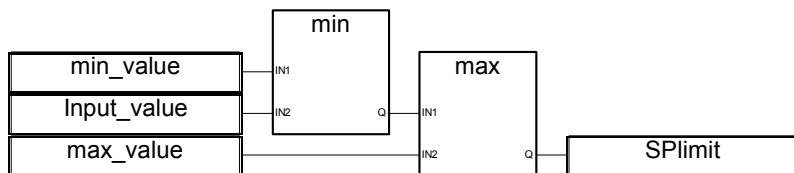
(\* 主 ST 程序 \*)

(\* 得到一个模拟值并把它转换成一个被限定的时间值 \*)

`ana_timeprog := SPLimit ( tprog_cmd );`

`appl_timer := tmr (ana_timeprog * 100);`

(\* 调用被称为 ‘ Splimit’ 的 FBD 程序 \*)



例 2: 功能调用

(\* 在一个复杂表达式中调用的 min, max, right, mlen 和 left 标准 “ C ” 函数 \*)

```
limited_value := min (16, max (0, input_value) );
```

```
rol_msg := right (message, mlen (message) - 1) + left (message, 1);
```

## 调用功能块

名称: 功能块实例的名称

含义: 从 ISaGRAF 库中或从用户的库中调用一个功能块并访问它的返回参数

语法: (\* 功能块的调用 \*)  
 <功能块名称> ( <p1>, <p2> ... );  
 (得到它的返回参数\*)  
 <结果> := <功能块名称>. <ret\_param1>;  
 ...  
 <结果> := <功能块名称>. <ret\_paramN>;

操作数: 参数是与该功能块的参数类型相对应的表达式

返回值:           参见语法中的得到返回参数部分

为了找到每个功能块参数的含义和类型，请查阅 ISaGRAF 库。  
功能块实例 (拷贝的名称) 必须在字典中声明。

举例:

(\* ST 程序调用一个功能块 \*)

(\* 在字典中声明功能块的实例: \*)

(\* trigb1 : 块 R\_TRIG -上升沿检测 \*)

(\*从 ST 语言激活功能块 \*)

trigb1 (b1);

(\* 存取返回参数 \*)

if (trigb1.Q) Then nb\_edge := nb\_edge + 1; End\_if;

## B.7.4 ST 特殊的布尔操作符

下列布尔操作符是专门用于 ST 语言的:

REDGE           上升沿检测

FEDGE           下降沿检测

其它的标准布尔操作符，例如可以使用下列操作符:

NOT           布尔求反

AND (&)       逻辑与

OR                      逻辑 或

XOR                    逻辑 异-或

有关对它们的描述可在 '标准操作符、功能块和功能' 一节中找到。

## **“” REDGE” 操作符**

名称:                  REDGE

含义:                  判断一个完整的布尔表达式的上升沿

语法:                  <edge>                    :=                  REDGE                  (                  <boo\_expression>,<memo\_variable> );

操作数:                  第一个操作数是任何布尔变量或综合表达式  
                            第二个操作数是一个内部的布尔变量，用于保存表达式的最后状态

返回值:                  当表达式从假 (FALSE) 变为真 (TRUE) 时，返回值为真 (TRUE)  
                            在所有其它情况下，返回值为假 (FALSE)

使用 REDGE 操作符时，在同一个执行周期中，一个表达式的上升沿不能检测一次以上。这个操作数可用于描述附加给 SFC 转换的条件。

警告: 用于保存表达式最后状态的“记忆式”布尔变量不能被当作一个不同表达式沿的触发脉冲来使用。

当表达式是一个被命名为“xxx”的布尔变量时，应声明一个名为“EDGE\_xxx”的唯一的内部变量，并在这个布尔变量的REDGE表达式中使用它。用这种方法可保证在执行其它REDGE时，记忆变量不被改写。

举例:

(\* 使用 REDGE 操作符的 ST 程序\*)

(\* 该程序是对一个布尔输入的上升沿进行计数 \*)

(\* Bi120 是一个输入布尔变量 \*)

(\* Edge\_Bi120 变量是 Bi120 变量状态的记忆 \*)

```
if REDGE (Bi120, Edge_Bi120) Then
    Counter := Counter + 1;
End_if;
```

注意: 这个操作符不在 IEC1131-3 标准内。您最好使用 R\_TRIG 标准块。由于兼容性方面的原因，它已被保留。

## □ **"FEDGE" 操作符**

名称: FEDGE

含义: 判断一个布尔表达式的下降沿

语法: <edge> := FEDGE ( <boo\_expression>, <memo\_variable> );

操作数:            第一个操作数是任何布尔变量或综合表达式  
                     第二个操作数是一个内部的布尔变量，用于保存表达式的最后状态

返回值:            当表达式从假 (FALSE) 变为真 (TRUE) 时，返回值为真 (TRUE)  
                     在所有其它情况下，返回值为假 (FALSE)

使用 FEDGE 操作符时，在同一个执行周期中，一个表达式的下降沿不能检测一次以上。这个操作数可用于描述附加给一个 SFC 转换的条件。

警告: 用于保存表达式最后状态的“记忆式”布尔变量不能被当作一个不同表达式沿的触发脉冲来使用。

当表达式是一个被命名为“xxx”的布尔变量时，应声明一个名为“EDGE\_xxx”的唯一的内部变量，并在这个布尔变量的 FEDGE 表达式中使用它。用这种方法可保证在执行其它 FEDGE 时，记忆变量不被改写。

举例：

(\* 使用 FEDGE 操作符的 ST 程序 \*)

(\* 该程序是对一个布尔输入的下降沿进行计数\*)

(\* Bi120 是一个输入布尔变量 \*)

(\*Edge\_Bi120 变量是 Bi120 变量状态的记忆\*)

if FEDGE (Bi120, Edge\_Bi120) Then

Counter := Counter + 1;

End\_if;

注意：这个操作符不在 IEC1131-3 标准内。你最好使用 F\_TRIG 标准块。由于兼容性方面的原因，它已被保留。

## B.7.5 ST 基本语句

ST 语言的基本语句是：

赋值语句

返回语句

IF-THEN-ELSIF-ELSE 结构

CASE 情况语句

WHILE 循环语句

REPEAT 循环语句

FOR 循环语句

EXIT 退出语句

### 赋值语句

名称:                   :=

含义:                    把一个变量赋值给一个表达式

语法:                    <variable> := <any\_expression> ;

操作数:                    变量必须是内部的或输出变量  
                              变量和表达式须有相同的类型

表达式可以是对一个子程序或者 ISaGRAF 库中函数的调用

举例:

(\* 有赋值语句的 ST 程序 \*)

(\* 变量 <=< 变量 \*)

bo23 := bo10;

(\* 变量 <=< 表达式 \*)

bo56 := bx34 OR alrm100 & (level >= over\_value);

result := (100 \* input\_value) / scale;

(\* 有子程序返回值的赋值语句 \*)

rc := PSelect ( );

(\* 有函数调用的赋值语句 \*)

limited\_value := min (16, max (0, input\_value) );

---

## 返回语句

名称: RETURN

含义: 终止当前程序的执行

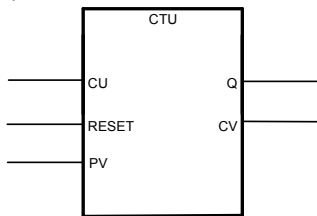
语法: RETURN ;

操作数: (无)

在一个SFC动作块中,返回语句仅表示该动作块的执行结束。

举例:

(\* 程序的FBD规范: 可编程计数器 \*)



(\* ST 程序中, 使用返回语句 \*)

```
if not (CU) then
    Q := false;
    CV := 0;
    返回; (* 终止该程序 *)
end_if;
```

```
if R then
    CV := 0;
else
    if (CV < PV) then
        CV := CV + 1;
    end_if;
end_if;
Q := (CV >= PV);
```

## ≡ **IF-THEN-ELSIF-ELSE 语句**

名称: IF ... THEN ... ELSIF ... THEN ... ELSE ... END\_IF

含义: 根据一个布尔表达式的值选择  
执行两个ST语句表中的其中一个

语法: IF <boolean\_expression> THEN  
    <语句>;  
    <语句>;  
    ...  
ELSIF <boolean\_expression> THEN  
    <语句>;  
    <语句>;  
    ...  
ELSE

```
<语 句> ;  
<语 句> ;  
...  
END_IF;
```

ELSE 和 ELSIF 语句是可选择的。IF 没有编写 ELSE 语句，那么当条件为假 (FALSE) 时，没有指令被执行。

举例:

(\* 使用 IF 语句的 ST 程序 \*)

```
IF manual 和 not (alarm) THEN  
    level := manual_level;  
    bx126 := bi12 OR bi45;  
ELSIF over_mode THEN  
    level := max_level;  
ELSE  
    level := (lv16 * 100) / scale;  
END_IF;
```

(\* 没有 ELSE 的 IF 结构 \*)

```
IF overflow THEN alarm_level := true;  
END_IF;
```

## ☐ **CASE 语句**

名称: CASE ... OF ... ELSE ... END\_CASE

含义: 根据一个整型表达式选择执行几个ST语句表中的其中一个

语法: CASE <integer\_expression> OF  
    <value> : <语句> ;  
    <value> , <value> : <语句> ;  
    ...  
ELSE  
    <语句> ;  
END\_CASE;

CASE值必须是一个整型的常量表达式。用逗号隔开的几个值可以产生相同的语句表。ELSE语句是可选择的。

举例:

(\*使用CASE语句的ST程序\*)

```
CASE error_code OF
    255: err_msg := 'Division by zero';
        fatal_error := TRUE;
    1:   err_msg := 'Overflow';
    2, 3: err_msg := 'Bad sign';
ELSE
    err_msg := 'Unknown error';
```

END\_CASE;

## WHILE 语句

名称: WHILE ... DO ... END\_WHILE

含义: 用于一组 ST 语句的循环结构  
在任一循环之前判断“继续”的条件

语法: WHILE <boolean\_expression> DO  
    <语句>;  
    <语句>;  
    ...  
END\_WHILE;

警告: 因为 ISaGRaF 是一个同步系统，所以在 WHILE 的循环期间，不刷新输入变量。一个输入变量状态的变化不能用于描述一个 WHILE 语句的条件。

举例:

(\*使用 WHILE 语句的 ST 程序\*)

(\*这个程序使用特殊的“C”函数读字符\*)

(\*在一个串行口上\*)

string := ""; (\*空字符串\*)

```
nbchar := 0;
```

```
WHILE ((nbchar < 16) & ComIsReady ( )) DO  
    string := string + ComGetChar ( );  
    nbchar := nbchar + 1;  
END_WHILE;
```

## **REPEAT 重复语句**

名称: REPEAT ... UNTIL ... END\_REPEAT

含义: 用于一组 ST 语句的循环结构  
在任一循环之后判断“继续”的条件

语法: REPEAT  
    <语句>;  
    <语句>;  
    ...  
    UNTIL <boolean\_condition>  
END\_REPEAT;

警告: 因为 ISaGRAF 是一个同步系统，所以在 WHILE 的循环期间，不刷新输入变量。一个输入变量状态的变化不能用于描述一个 REPEAT 语句的条件。

举例:

(\* 使用 REPEAT 语句的 ST 程序 \*)

(\* 这个程序使用特殊的“C”函数读字符 \*)

(\* 在一个串行口上 \*)

string := ""; (\* 空字符串 \*)

nbchar := 0;

IF ComIsReady ( ) THEN

    REPEAT

        string := string + ComGetChar ( );

        nbchar := nbchar + 1;

    UNTIL ( (nbchar >= 16) OR NOT (ComIsReady ( )) )

    END\_REPEAT;

END\_IF;

## == **FOR 语句**

名称:               FOR ... TO ... BY ... DO ... END\_FOR

含义:               使用一个整形的模拟索引变量执行一个受限的循环数

语法:               FOR <index> := <mini> TO <maxi> BY <step> DO  
                      <语句>;  
                      <语句>;  
                      END\_FOR;

操作数:	index:	在任一循环中增加的内部模拟变量
	mini:	索引的初始值(在第一个循环之前)
	maxi:	索引的最大允许值
	步:	每次循环时的索引增量

[ BY step ] 语句是可选的。如果没有具体规定，则增量 step 为 1

警告: ISaGRAF 是一个同步系统，所以在 FOR 循环期间不刷新输入变量。

这是一个与 FOR 语句等效的“ while” 语句:

```
index := mini;
while (step <= maxi) do
    <语句>;
    <语句>;
    index := index + step;
end_while;
```

举例:

(\*使用 FOR 语句的 ST 程序\*)

(\* 该程序提取字符串中的数字符 \*)

```
length := mlen (message);
```

```
target := ""; (* 空字符串 *)
FOR index := 1 TO length BY 1 DO
    code := ascii (message, index);
    IF (code >= 48) & (code <= 57) THEN
        target := target + char (code);
    END_IF;
END_FOR;
```

## EXIT 退出语句

名称: EXIT

含义: 从一个 FOR, WHILE 或 REPEAT 循环语句中退出

语法: EXIT;

操作数: (无)

EXIT 通常用于一个 IF 语句内, 并在一个 FOR, WHILE 或 REPEAT 块内。

举例:

(\* 使用 EXIT 语句的 ST 程序 \*)

(\* 该程序在一个字符串中搜索一个字符 \*)

```
length := mlen (message);
found := NO;
```

```
FOR index := 1 TO length BY 1 DO
    code := ascii (message, index);
    IF (code = searched_char) THEN
        found := YES;
        EXIT;
    END_IF;
END_FOR;
```

## B.7.6 ST 扩展

以下功能是对 ST 语言的扩展：

TSTART-TSTOP：计时器控制

下列语句和功能可用于控制 SFC 子程序的执行。它们可在 ACTION(), .END\_ACTION 内部使用；也可用于 SFC 步内的块。

GSTART	启动一个 SFC 程序
GKILL	终止一个 SFC 程序
GFREEZE	冻结一个 SFC 程序
GRST	重新启动一个冻结的 SFC 程序
GSTATUS	获取一个 SFC 程序的当前状态

**警告：**这些功能不在 IEC 1131-3 标准内。

在 SFC 步内使用下面的语法可以为 GSTART 和 GKILL 找到等效结果：

child\_name(S); (\* 与 GSTaRT(child\_name) 等效; \*)

child\_name(R); (\* 与 GKILL(child\_name) 等效; \*)

下列符号组可用于访问一个 SFC 步的状态:

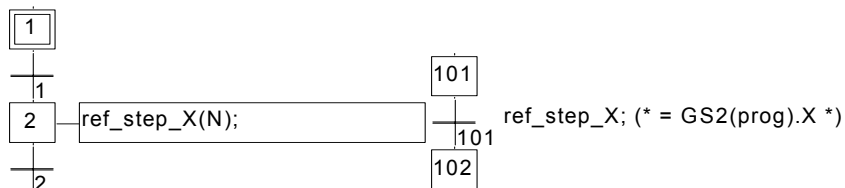
GSnnn.x                      代表步被激活的布尔值  
GSnnn.t                      从步的最后一次激活起所化的时间  
                                (“ nnn” 是 SFC 步的参考号)

对于在另一个 SFC 程序中声明的一个步的激活状态也可以进行测试, 此时应使用下列语法:

GSnnn(progname).x

**警告:** 在参考另一个程序的一个步时, 该语法不在 IEC 1131-3 标准中。实现同一操作而又遵守 IEC 规则, 有一个简易的方法是, 在字典中声明一个全局布尔变量, 它表示需要测试的那个步的激活状态 (例如 ref\_step\_X)。然后你把带 N 判定的变量 (ref\_步\_X(N);) 插入其中。接着在准备对步的激活状态进行测试的程序中使用该变量。

Prog 程序                      需要 Prog 程序的步激活状态的另一个程序



## ≡ TSTART 语句

名称:                      TSTART

**含义:** 一个计时器变量开始计数

TSTART 命令不会改变计时器的值，也就是说，计数是从计时器的当前值开始。

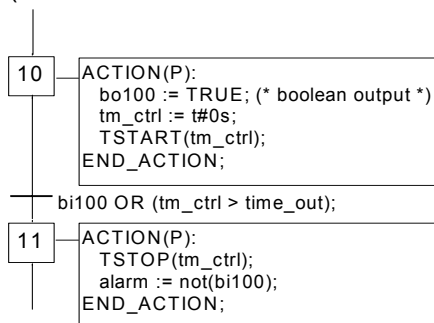
**语法:** TSTART ( <timer\_variable> );

**操作数:** 任一非激活的计时器变量

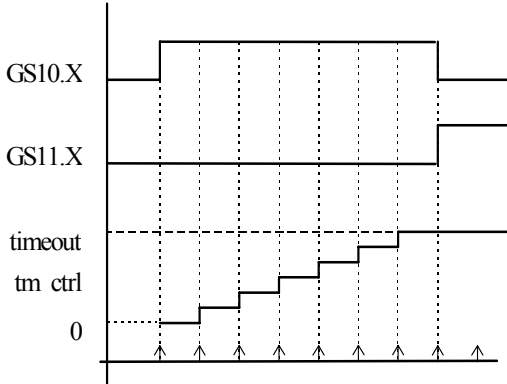
**返回值:** (无)

**举例:**

(\*使用 TSTART 和 TSTOP 语句的 SFC 程序\*)



如果 bi100 总是 FALSE，则时间图如下:



在一个循环里计时器保持相同值。

## □ **TSTOP 语句**

名称: TSTOP

含义: 停止更新一个计时器变量  
TSTOP 命令不能修改计时器的值

语法: TSTOP ( <timer\_variable> );

操作数: 任一激活的计时器变量

返回值: (无)

举例: 参见 (上面所描述的) TSTART 功能

## □ **GSTART 语句**

名称: GSTART

含义: 通过把一个标记放入程序的每个初始步中来启动一个子程序。

语法: GSTART ( <child\_program> );

操作数: 给定的 SFC 程序必须是一个子程序，且在它的主程序中写有上述语句。

返回值: (无)

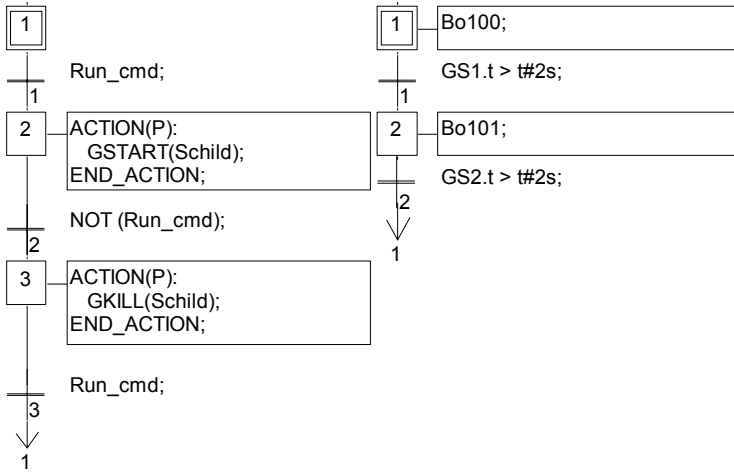
子程序的子程序不能自动地被 GSTART 语句启动。

注意：因为 GSTART 语句不在 IEC 1131-3 标准中，最好使用 S 判定，并且使用下面的语法来启动一个 SFC 子程序：

Child\_name(S);

举例：GSTART 和 GKILL 的使用

(\* ‘ Sfather’ 序列\*) (\* ‘ Schild’ 序列\*)



## **GKILL 语句**

名称: GKILL

含义: 通过去除程序步内当前存在的标记终止一个 SFC 子程序

语法: GKILL ( <child\_程序> );

操作数: 给定的 SFC 程序必须是一个子程序，且在它的主程序中写有上述语句

返回值: (无)

子程序的子程序被给定的程序自动地停止。

注意: 因为 GKILL 语句不在 IEC 1131-3 标准中，最好使用 R 判定，并且使用下面的语法来终止一个 SFC 子程序:

Child\_name(R);

举例：参见 (上面所描述的) GSTART 功能

## GFREEZE 语句

名称: GFREEZE

含义: 暂停一个 SFC 程序的执行  
冻结的程序可由 GRST 语句重新启动。

语法: GFREEZE ( <child\_program> );

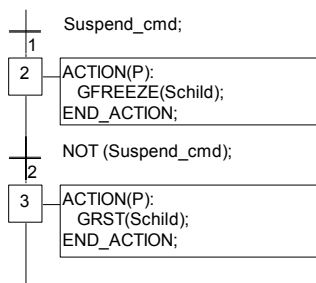
操作数: 给定的 SFC 程序必须是一个子程序，且在它的主程序中写有上述语句

返回值: (无)

子程序的子程序和给定的程序一起被自动冻结。

注意：GFREEZE 语句不在 IEC 1131-3 标准中。

举例：



**▢ GRST 语句**

名称: GRST

含义: 重新启动一个由 GFREEZE 语句冻结的 SFC 子程序

语法: GRST ( <child\_program> );

操作数: 给定的 SFC 程序必须是一个子程序，且在它的主程序中写有上述语句

返回值: (无)

子程序的子程序被 GRST 语句自动地重新启动。

注意: GRST 语句不在 IEC 1131-3 标准中。

举例: 参见 (上面所描述的) GFREEZE 功能

**▢ GSTATUS 语句**

名称: GSTATUS

含义: 返回一个 SFC 程序的当前状态

语法: <ana\_var> := GSTATUS ( <child\_program> );

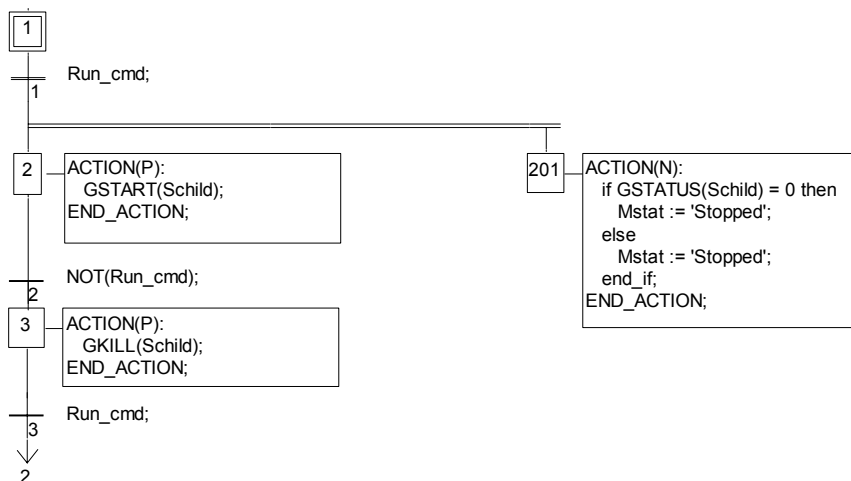
**操作数:** 给定的 SFC 程序必须是一个子程序，且在它的主程序中写有上述语句

**返回值:**

- 0 = 程序非激活状态 (被终止)
- 1 = 程序为激活状态 (被启动)
- 2 = 程序被冻结

**注意：** GFREEZE 语句不在 IEC 1131-3 标准中。

**举例:**



## B.8 IL 指令表语言

) 指令表语言 (Instruction List 或称 IL) 是一种低级语言, 它对于较小的应用程序或某个应用程序的部分优化是非常有效的。指令总是与当前结果 (或 IL 寄存器) 有关。操作符表示当前值和操作数间的操作。操作结果再被存入当前结果中。

### B.8.1 IL 的主要语法

IL 程序是一个指令表。每条指令从新的一行开始, 且含有一个操作符, 与可选的修改符形成一个整体。对于特殊操作, 必要时增加一个或多个操作数, 并用逗号 (‘,’) 将操作数分开。标号后面跟随冒号 (‘:’), 放在指令的前面。如果指令中附有注释, 它必须是在一行的最后。注释以 ‘( \*’ 开始, 用 ‘\*)’ 结束。指令之间可以输入空行。注释可放在空行上。

以下是指令行的例子:

标号	操作符	操作数	注释
Start:	LD	IX1	(* push button *)
	ANDN	MX5	(* command is not forbidden *)
	ST	QX2	(* start motor *)

## ≡      **标号**

标号后随冒号(‘:’), 可以放在指令的前面。标号可放在空行上。对某些操作(比如跳转)来说, 标号被用作操作符。命名标号须符合下列规则:

- 名称不能超过 16 个字符
- 第一个字符必须是一个字母
- 随后的字符必须是字母、数字或‘\_’ 字符

在同一 IL 中, 标号不可重名, 但可以和变量的名称相同。

## ≡      **操作修改符**

可使用的操作修改符如下所示。修改符的字符使操作符的名称完整, 字符间不能有空格:

N      操作数的布尔求反

(      延时操作

C      有条件的操作

‘N’ 修改符表示操作数布尔求反, 指令 ORN IX12 表示为:  
result := result := result OR NOT (IX12).

括号‘(’ 修改符表示指令的求值必须被延时至出现表示结束的括号‘)’ 操作符。

‘C’ 修改符表示只有在当前结果为布尔值“真”(不同于非布尔值 0) 时, 才执行相关指令。为了表示只有在当前结果为布

尔值“假”(或非布尔值0)时才执行指令, 可把 'C' 修改符与 'N' 修改符结合在一起。

## == 延时操作

因为只有一个 IL 寄存器 (当前值, 一些操作不得被延时, 这样就可以改变执行的顺序或指令。括号被用于表示延时操作:

'(' 是一个修改符 表示操作被延时

)' 是一个操作符 执行被延时的操作

括号 '(' 修改符表示指令的求值必须被延时, 直至出现表示结束的括号 ')' 操作符。例如下面的程序:

```
AND(   IX12
OR     IX35
)
```

被理解为:

```
result := result AND ( IX12 OR IX35 )
```

## B.8.2 IL 操作符

下面的列表概括了 IL 语言的标准操作符

操作符	修改符	操作数	描述
-----	-----	-----	----

LD	N	变量, 常量	装入操作数
ST	N	变量	保存当前结果
S		BOO 变量	置位为 TRUE
R		BOO 变量	复位为 FALSE
CAL	C N	FB 状态名称	调用一个功能块
JMP	C N	标号	跳转到标号处
RET	C N		从子程序返回
)			执行被延时的操作
Function			调用函数或子程序
AND	N (	BOO	布尔与运算
&	N (	BOO	布尔与运算
OR	N (	BOO	布尔或运算
XOR	N (	BOO	异-或运算
ADD	(	变量, 常量	加
SUB	(	变量, 常量	减
MUL	(	变量, 常量	乘
DIV	(	变量, 常量	除
GT	(	变量, 常量	检测: >
GE	(	变量, 常量	检测: >=
EQ	(	变量, 常量	检测: =
LE	(	变量, 常量	检测: <=
LT	(	变量, 常量	检测: <

NE ( 变量, 常量 检测: <>

在以下段落中，只描述与IL语言中的操作符，其它标准操作符可在“标准操作符、功能块和功能”段落中找到。

## LD 操作符

操作 将一个数值载入到当前结果中

允许的修改符 N

操作数 常量表达式  
内部的、输入或输出变量

举例：

(\* LD 操作示例 \*)

```
LDex:  LD    false    (* result := FALSE boolean constant *)
        LD    true     (* result := TRUE boolean constant *)
        LD    123      (* result := integer constant *)
        LD    123.1    (* result := real constant *)
        LD    t#3ms    (* result := time constant *)
        LD    boo_var1 (* result := boolean variable *)
        LD    ana_var1 (* result := analog variable *)
        LD    tmr_var1 (* result := timer variable *)
        LDN   boo_var2 (* result := NOT (boolean variable) *)
```

## ▣ **ST 操作符**

操作                    把当前结果存到一个变量中  
                          当前结果没有被该操作所修改

允许的修改符   N

操作数                内部的或输出变量

举例:

( \* ST 操作示例\* )

```
STboo:   LD      false
          ST      boo_var1 (* boo_var1 := FALSE *)
          STN     boo_var2 (* boo_var2 := TRUE *)
STana:   LD      123
          ST      ana_var1 (* ana_var1 := 123 *)
STtmr:   LD      t#12s
          ST      tmr_var1 (* tmr_var1 := t#12s *)
```

## ▣ **S 操作符**

操作                    如果当前结果为布尔值 TRUE，则把布尔值  
                          TRUE 存到一个布尔变量中。如果当前结果是  
                          FALSE，则不操作处理。当前结果没有被该操  
                          作所修改。

允许的修改符: (无)

操作数:            输出或内部的布尔变量

举例:

(\* S 操作示例 \*)

```
SETex:  LD   true      (* current result := TRUE *)
        S    boo_var1 (* boo_var1 := TRUE *)
                        (* current result is not modified *)
        LD   false     (* current result := FALSE *)
        S    boo_var1 (* nothing done, boo_var1 unchanged *)
```

## ❏ **R 操作符**

操作            如果当前结果为布尔值 TRUE，则把布尔值 FALSE 存到一个布尔变量中。如果当前结果是 FALSE，则不操作处理。当前的结果没有被该操作所修改。

允许的修改符    (无)

操作数            输出或内部的布尔变量

举例:

(\* R 操作示例\*)

```
RESETex: LD   true      (* current result := TRUE *)
          R    boo_var1 (* boo_var1 := FALSE *)
                        (* current result is not modified *)
```

```
ST      boo_var2 (* boo_var2 := TRUE *)
LD      false    (* current result := FALSE *)
R       boo_var1 (* nothing done, boo_var1 unchanged *)
```

## **JMP 操作符**

操作                      跳转到给定的标号

允许的修改符    C   N

操作数                      在相同的 IL 程序中定义的标号

举例：

(\* 下例通过测试模拟选择器的值为 0 或者为 1 或者为 2)

(\* 来设置 3 个布尔输出值。值是否为 0 决定是否进行 JMPC 操作

```
JMPex:      LD      selector
             BOO
             JMPC    test1
             LD      true
             ST      bo0
             JMP     JMPend
test1:      LD      selector
             SUB     1
             BOO
             JMPC    test2
             LD      true
             ST      bo1
             JMP     JMPend
test2:      LD      true
             ST      bo2
JMPend:
```

## RET 操作符

**操作**                      结束当前的指令表。如果 IL 是一个子程序，则当前结果返回到调用程序。

**允许的修改符**    C   N

**操作数**                      (none)

**举例：**

```
JMPex:      LD      selector
             BOO
             JMPC    test1
             LD      true
             ST      bo0
             RET

test1:       LD      selector
             SUB     1
             BOO
             JMPC    test2
             LD      true
             ST      bo1
             LD      1
             RET

test2:       RETNC
             LD      true
             ST      bo2
             LD      2
```

## )" operator

**操作符**执行一个延时操作。用 '(' 符号通知延时操作

**允许的修改符**    (none)

**操作数**                      (none)

举例：

```
(* res = a1 + (a2 * (a3 - a4) * a5) + a6; *)
```

```
Delayed:      LD          a1
               ADD(       a2
               MUL(       a3
               SUB        a4
               )
               MUL        a5
               )
               ADD        a6
               ST          res
```

## 调用子程序或函数

从 IL 程序中调用一个子程序 或一个函数 (用 IL、ST LD、FBD 或“C”语言中的任何一种语言编写)，此时把它的名称用作一个操作符。

**操作**                      执行一个子程序或一个函数 - 由子程序或函数返回的数值被存入 IL 的当前结果中。

允许的修改符 (无)

**操作数**                      第一个调用的数据必须在调用之前存到当前的结果中。随后的参数在操作数区域中表示，并用逗号分开。

举例：

(\* 调用程序：将一个模拟量转换为一个时间量\*)

```

Main:      LD      bi0
           SUBPRO   bi1,bi2
           ST      result
           GT      vmax
           RETC
           LD      result
           MUL     1000
           TMR
           ST      tmval

           LD      in2
           ANA
           MUL     2
           ST      temporary
           LD      in1
           ANA
           ADD     temporary
           MUL     2
           ST      temporary
           LD      in0
           ANA
           ADD     temporary
           ST      SUBPRO

```

## 调用功能块：CAL 操作符

操作调用一个功能块

允许的修改符 C N

操作数                  功能块实例名

在使用 LD / ST 操作顺序调用功能块之前，必须给功能块的输入参数赋值。

这样，在使用时，输出参数是已知的。

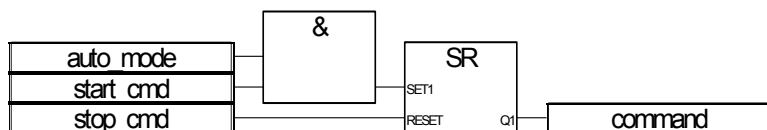
例 1:

(\* 调用功能块 SR：SR1 是 SR 的一个实例\*)

```

LD      auto_mode
AND     start_cmd
ST      SR1.set1
LD      stop_cmd
ST      SR1.reset
CAL     SR1
LD      SR1.Q1
ST      command
    
```

(\* FBD 等价 \*)



## 例 2

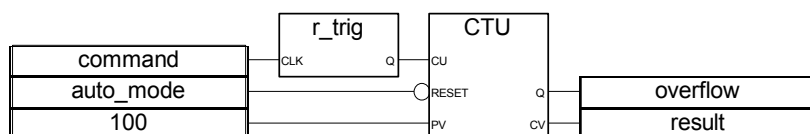
(\* 假设 R\_TRIG1 是 R\_TRIG 块的一个实例，CTU1 是 CTU 块的一个实例 \*)

```

LD      command
ST      R_TRIG1.clk
CAL     R_TRIG1
LD      R_TRIG1.Q
ST      CTU1.cu
LDN     auto_mode
ST      CTU1.reset
LD      100
ST      CTU1.pv
    
```

CAL	CTU1
LD	CTU1.Q
ST	overflow
LD	CTU1.cv
ST	result

(\* FBD 等价: \*)



## B.9 标准运算符，功能块与函数

### B.9.1 标准运算符

下列是符合 IEC 标准语言的标准运算符。

数据处理:	赋值, 模拟量求反
布尔运算:	布尔逻辑与
	布尔逻辑或
	布尔逻辑异或
算术运算:	加
	减
	乘
	除
逻辑运算:	模拟量逐位与屏蔽
	模拟量逐位或屏蔽
	模拟量逐位异或屏蔽
	逐位求反
比较测试运算:	小于
	小于或等于
	大于
	大于或等于
	等于
	不等于
数据变换:	变换为布尔型
	变换为整型模拟量

变换为实型模拟量

变换为定时器型

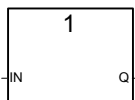
变换为信息

其它:.....信息串联

系统访问

操作 I/O 通道

## 1 gain 赋值



参数:

IN            任何类型

Q            任何类型

说明:

将一个变量赋值给另一个变量

此模块对于将图表的输入直接连接到图表的输出非常有用。它也可用于转化图表输出的状态(需要一个布尔求反的连接)。

(\* 赋值块的 FBD 例子 \*)

(\* 等价的 ST 语言程序: \*)

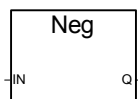
```
ao23 := ai10;
```

```
bo100 := NOT (bi1 AND bi2);
```

(\* 等价的 IL 语言程序: \*)

```
LD      ai10
ST      ao23
LD      bi1
AND     bi2
STN     bo100
```

## NEG 模拟量求反



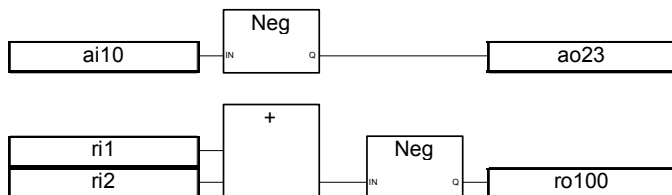
参数:

IN	整型-实型	输入和输出的类型必须相同
Q	整型-实型	

说明:

对变量求反

(\*求反模块的 FBD 例子 \*)



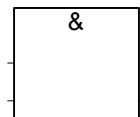
(\*等价的 ST 语言程序: \*)

```
ao23 := - (ai10);
ro100 := - (ri1 + ri2);
```

(\*等价的 IL 语言程序:\*)

```
LD      ai10
MUL     -1
ST      ao23
LD      ri1
ADD     ri2
MUL     -1.0
ST      ro100
```

## & AND 布尔逻辑与



注意：这个运算符的输入可以扩展到两个以上。

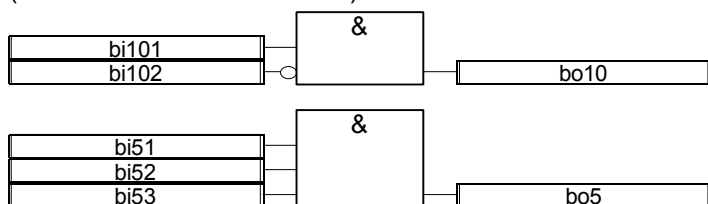
参数：

(输入)	布尔
输出	布尔      输入项的布尔逻辑与

说明：

对两个或更多的输入进行布尔逻辑与运算。

(\*“ AND” 块的 FBD 例子 \*)



(\*等价的 ST 语言程序: \*)

bo10 := bi101 AND NOT (bi102);

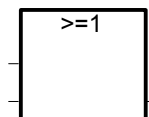
bo5 := (bi51 AND bi52) AND bi53;

(\*等价的 IL 语言程序 \*)

LD	bi101	(* 当前结果 := bi101 *)
ANDN	bi102	(* 当前结果 := bi101 AND not(bi102) *)
ST	bo10	(* bo10 := 当前结果 *)
LD	bi51	(* 当前结果 := bi51;
&	bi52	(* 当前结果 := bi51 AND bi52 *)
&	bi53	(* 当前结果 := (bi51 AND bi52) AND
		bi53 *)
ST	bo5	(* bo5 := 当前结果 *)

## & OR 布尔逻辑或

>=1 OR 布尔逻辑或



注意：此运算的输入可以扩展到两个以上。

参数：

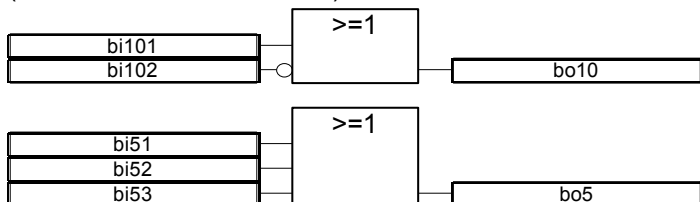
(输入) 布尔

输出 布尔 输入的布尔逻辑或

说明：

Boolean OR of two or more terms.

(\*“OR”块的FBD例子\*)



(\*等价的ST语言程序:\*)

```
bo10 := bi101 OR NOT (bi102);
```

```
bo5 := (bi51 OR bi52) OR bi53;
```

(\*等价的IL语言程序:\*)

```
LD      bi101
```

```
ORN     bi102
```

```
ST      bo10
```

```
LD      bi51
```

```
OR      bi52
```

```
OR      bi53
```

```
ST      bo5
```

## =1 XOR 布尔逻辑异或



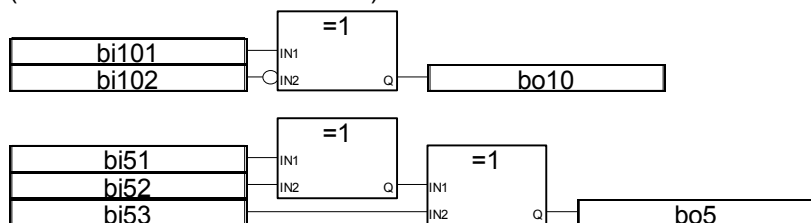
参数:

IN1	布尔	
IN2	布尔	
Q	布尔	两个输入项的布尔逻辑异或

说明:

对两个输入求布尔逻辑异或。

(\*“ XOR” 块的 FBD 例子 : \*)



(\*等价的 ST 语言程序: \*)

```
bo10 := bi101 XOR NOT (bi102);
```

```
bo5 := (bi51 XOR bi52) XOR bi53;
```

(\*等价的 IL 语言程序: \*)

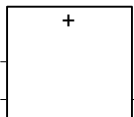
```
LD      bi101
```

```
XORN    bi102
```

```
ST      bo10
```

LD	bi51
XOR	bi52
XOR	bi53
ST	bo5

## +加



注意：此运算的输入可以扩展到两个以上。

参数:

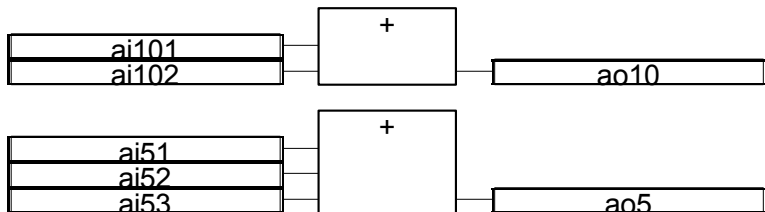
(输入)      实型-整型    可以是整型或实型数据  
(所有输入必须类型相同)

输出 实型-整型 输入的带符号加

说明:

对两个或以上的模拟变量做加法。

(\* 加法模块的 FBD 例子: \*)



(\* 等价的 ST 语言程序: \*)

```
ao10 := ai101 + ai102;
ao5 := (ai51 + ai52) + ai53;
```

(\* 等价的 IL 语言程序: \*)

```
LD      ai101
ADD     ai102
ST      ao10
LD      ai51
ADD     ai52
ADD     ai53
ST      ao5
```

## -减



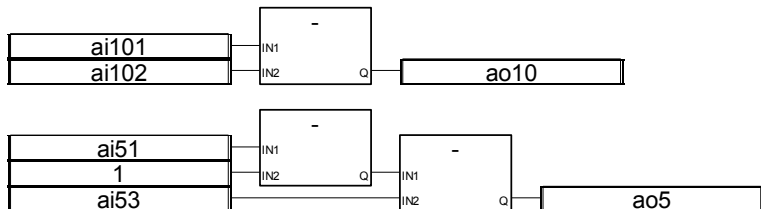
参数:

IN1	整型-实型 可以是整型或实型数据
IN2	整型-实型 ( IN1 和 IN2 必须同类型 )
Q	整型-实型 减法结果 ( 第一个 - 第二个 )

说明:

对两个模拟变量做减法 ( 第一个 - 第二个 )。

(\* 减法模块的 FBD 例子: \*)



(\* 等价的 ST 语言程序: \*)

```
ao10 := ai101 - ai102;
```

```
ao5 := (ai51 - 1) - ai53;
```

(\* 等价的 IL 语言程序: \*)

```
LD      ai101
```

```
SUB     ai102
```

```
ST      ao10
```

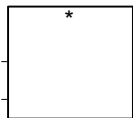
```
LD      ai51
```

```
SUB     1
```

```
SUB     ai53
```

```
ST      ao5
```

## \*乘



注意：此运算的输入可以扩展到两个以上。

参数:

(输入)      整型实型可以是整型或实型数据

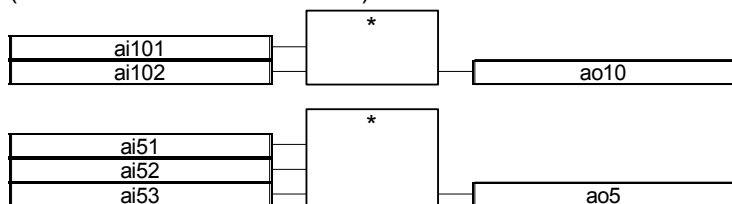
(所有输入必须类型相同)

输出            整型-实型    输入的带符号乘法

说明:

两个或以上模拟变量的乘积

(\* 乘法模块的FBD例子: \*)



(\*等价的 ST 语言程序: \*)

```
ao10 := ai101 * ai102;
```

```
ao5 := (ai51 * ai52) * ai53;
```

(\*等价的 IL 语言程序: \*)

```
LD      ai101
```

```
MUL     ai102
```

```
ST      ao10
```

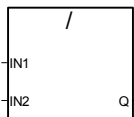
```
LD      ai51
```

```
MUL     ai52
```

```
MUL     ai53
```

```
ST      ao5
```

/除



参数:

IN1            整型-实型可以是整型或实型 ( 被除数 )

IN2            整型-实型非零的模拟值 ( 除数 )

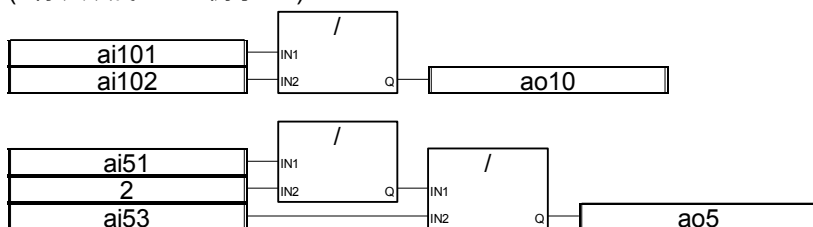
( IN1 和 IN2 必须同类型 )

Q             整型-实型IN1 被 IN2 带符号除

说明:

对两个变量做除法 ( 第一个被第二个除 ) 。

(\* 除法块的 FBD 例子 : \*)



(\* 等价的 ST 语言程序: \*)

```
ao10 := ai101 / ai102;
```

```
ao5 := (ai5 / 2) / ai53;
```

(\* 等价的 IL 语言程序 : \*)

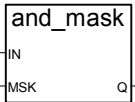
```
LD        ai101
```

```
DIV       ai102
```

```
ST        ao10
```

LD            ai51  
DIV           2  
DIV           ai53  
ST            ao5

## AND\_MASK 模拟量逐位与屏蔽



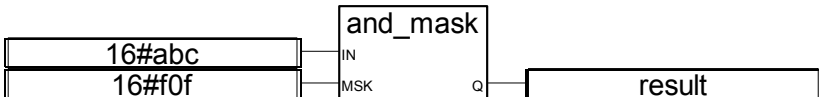
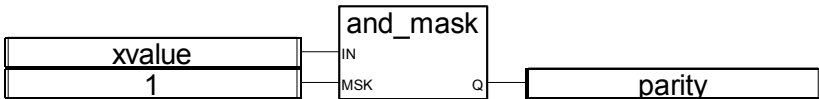
参数:

IN        整型    必须是整型格式  
MSK        整型    必须是整型格式  
Q        整型    IN 和 MSK 之间逐位逻辑与运算

说明:

整型模拟值进行逐位的与运算。

(\*AND\_MASK 块的 FBD 例子 : \*)



(\* 等价的 ST 语言程序 : \*)

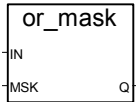
parity := AND\_MASK (xvalue, 1); (\*如果 xvalue 为 奇数 则为 1 \*)

result := AND\_MASK (16#abc, 16#f0f); (\*等于 16#a0c \*)

(\*等价的 IL 语言程序:\*)

```
LD          xvalue
AND_MASK 1
ST          parity
LD          16#abc
AND_MASK 16#0f
ST          result
```

### OR\_MASK 模拟量逐位或屏蔽



参数:

IN      整型      必须是整型格式

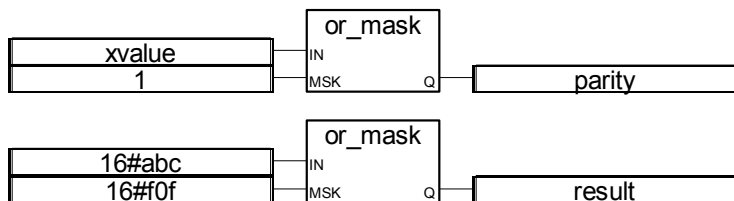
MSK      整型      必须是整型格式

Q      整型      IN 和 MSK 之间的逐位逻辑或

说明:

对整型模拟值进行逐位或运算。

(\*OR\_MASK 块的 FBD 例子:\*)



(\*等价的 ST 语言程序: \*)

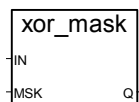
```
is_odd := OR_MASK (xvalue, 1); (* makes value always odd *)
```

```
result := OR_MASK (16#abc, 16#f0f); (* equals 16#fbf *)
```

(\*等价的 IL 语言程序\*)

```
LD      xvalue
OR_MASK 1
ST      is_odd
LD      16#abc
OR_MASK 16#f0f
ST      result
```

## XOR\_MASK 模拟量逐位异或屏蔽



参数:

IN      整型      必须是整型格式

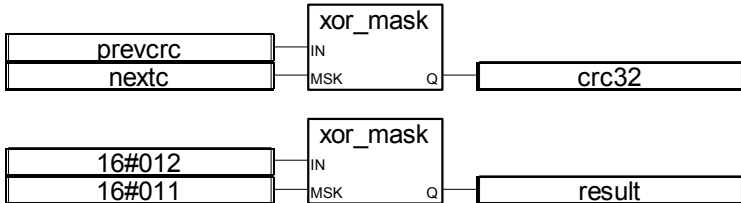
MSK      整型      必须是整型格式

Q      整型      IN 和 MSK 之间逐位逻辑异或

说明：

对整型模拟值逐位进行异或的位屏蔽运算。

(\* XOR\_MASK 块的 FBD 例子：\*)



(\*等价的 ST 语言程序:\*)

```
crc32 := XOR_MASK (prevcrc, nextc);
```

```
result := XOR_MASK (16#012, 16#011); (* equals 16#003 *)
```

(\* 等价的 IL 语言程序:\*)

```
LD      prevcrc
```

```
XOR_MASK nextc
```

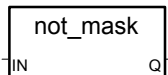
```
ST      crc32
```

```
LD      16#012
```

```
XOR_MASK 16#011
```

```
ST      result
```

## NOT\_MASK 逐位求反



参数：

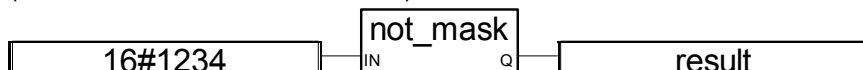
IN      整型    必须是整型格式

Q                      整型                      IN 的 32 位数据逐位求反

说明:

整型模拟值的逐位求反。

(\*NOT\_MASK 块的 FBD 例子: \*)



(\*等价的 ST 语言程序 \*)

```
result := NOT_MASK (16#1234);
```

(\* result is 16#FFFF\_EDCB \*)

(\* 等价的 IL 语言程序: \*)

```
LD            16#1234
```

```
NOT_MASK
```

```
ST            result
```

<小于



参数:

IN1      整型-实型-

定时器信息

IN2                      整型-实型

定时器-信息

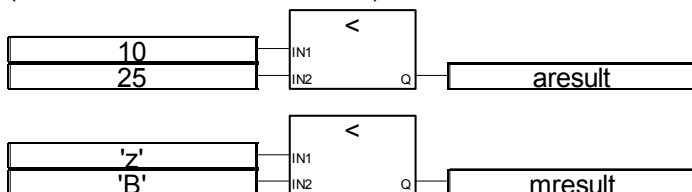
两个输入的类型必须相同

Q                    如果  $IN1 < IN2$  , 布尔状态为真

说明:

测试一个值是否小于另一个 ( 模拟 , 定时器或信息 )

(\*“ 小于” 块的 FBD 例子 : \*)



(\*等价的 ST 语言程序: \*)

areult := (10 < 25); (\* areult is “ 真 ” \*)

mresult := ('z' < 'B'); (\* mresult is “ 假 ” \*)

(\*等价的 IL 语言程序 : \*)

```

LD      10
LT      25
ST      areult
LD      'z'
LT      'B'
ST      mresult
  
```

**<= 小于等于**



参数:

IN1      整型-实型-信息

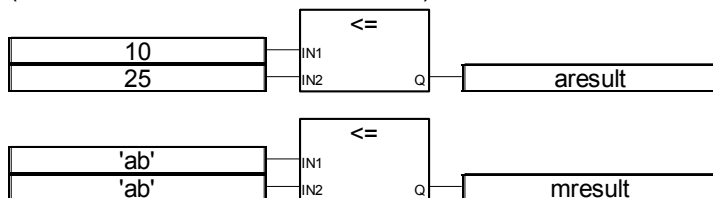
IN2              整型-实型-信息      两个输入的类型必须相同

Q              如果  $IN1 \leq IN2$  , 布尔状态为真

说明 :

测试一个值是否小于或等于另一个 ( 模拟 , 或信息 )

(\*“ 小于或等于” 块的 FBD 例子 : \*)



(\*等价的 ST 语言程序 \*)

areult := (10 <= 25); (\* areult is “ 真 ” \*)

mresult := ('ab' <= 'ab'); (\* mresult is “ 真 ” \*)

(\* 等价的 IL 语言程序 : \*)

LD          10

LE          25

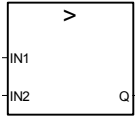
ST          areult

LD          'ab'

LE          'ab'

ST          mresult

## &gt;大于



参数:

IN1 整型-实型

定时器-信息

IN2 整型-实型-

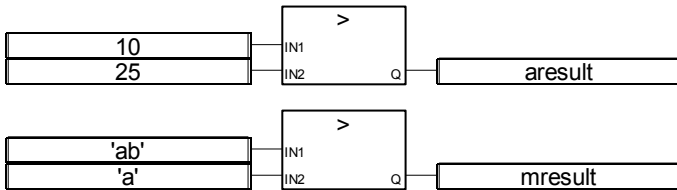
定时器-信息 两个输入的类型必须相同

Q 如果  $IN1 > IN2$  , 布尔状态为真

说明:

测试一个值是否大于或等于另一个 ( 模拟量 , 定时器或信息 )

(\*“ 大于” 块的 FBD 例子 : \*)



(\*等价的 ST 语言程序 : \*)

arestult := (10 > 25); (\* arestult is “ 假 ” \*)

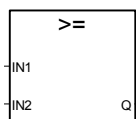
mresult := ('ab' > 'a'); (\* mresult is “ 真 ” \*)

(\* 等价的 IL 语言程序 : \*)

LD 10

GT	25
ST	areresult
LD	'ab'
GT	'a'
ST	mresult

## >= 大于或等于



参数:

IN1 整型-实型-信息

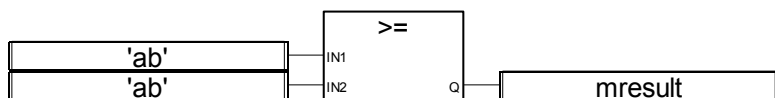
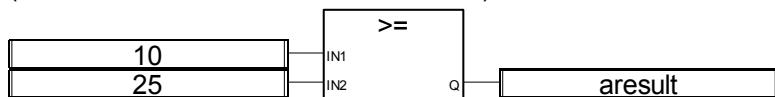
IN2 整型-实型-信息 两个输入的类型必须相同

Q 如果  $IN1 \geq IN2$ ，布尔状态为真

说明:

测试一个值是否大于或等于另一个（模拟量，或信息）

(\*“大于或等于”块的FBD例子：\*)



(\*等价的ST语言程序：\*)

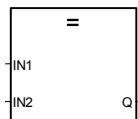
areresult := (10 >= 25); (\* areresult is FALSE \*)

```
mresult := ('ab' >= 'ab'); (* mresult is TRUE *)
```

(\*等价的 IL 语言程序:\*)

```
LD      10
GE      25
ST      aresult
LD      'ab'
GE      'ab'
ST      mresult
```

**= 等于**



参数:

**IN1**      整型-实型-信息

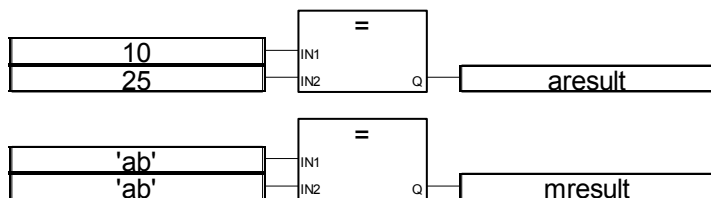
**IN2**              整型-实型-信息      两个输入的类型必须相同

**Q**                  如果  $IN1 \geq IN2$  , 布尔状态为真

说明:

测试一个值是否大于或等于另一个 ( 模拟量 , 或信息 )

(\*“等于”块的 FBD 例子:\*)



(\*等价的 ST 语言程序 \*)

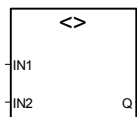
areult := (10 = 25); (\* areult is 假 \*)

mresult := ('ab' = 'ab'); (\* mresult is 真 \*)

(\*等价的 IL 语言程序：\*)

```
LD      10
EQ      25
ST      areult
LD      'ab'
EQ      'ab'
ST      mresult
```

## <> 不等于



## 参数

IN1 整型-实型-信息

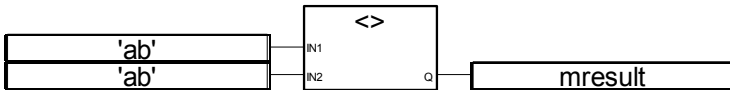
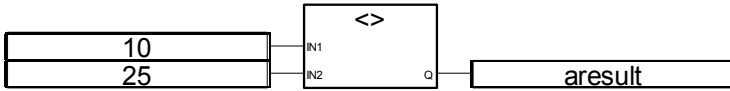
IN2 整型-实型-信息 两个输入的类型必须相同

Q 如果第一个 <> 第二个，布尔状态为真

## 说明

测试一个值是否不等于另一个 ( 模拟量 , 或信息 )

(\*“ 不等于 ” 块的 FBD 例子 : \*)



(\*等价的 ST 语言程序 : \*)

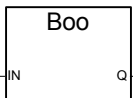
areult := (10  $\neq$  25); (\* areult is “ 真 ” \*)

mresult := ('ab'  $\neq$  'ab'); (\* mresult is “ 假 ” \*)

(\*等价的 IL 语言程序 : \*)

```
LD      10
NE      25
ST      areult
LD      'ab'
NE      'ab'
ST      mresult
```

## BOO 变换为布尔型



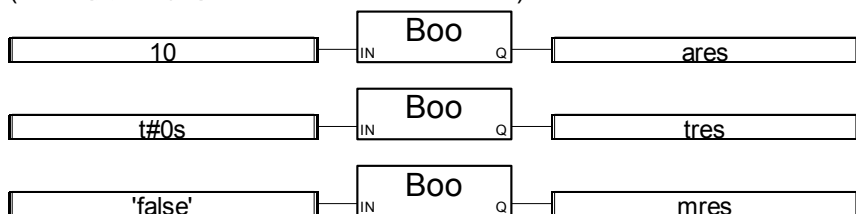
## 参数:

IN      任意    任意非布尔型的值  
       Q            布尔        对非零数值为真  
                       对零值为假  
                       对“真”信息为真  
                       对“假”信息为假

## 说明:

把任意类型的变量变换为布尔型

(\*“变换到布尔型”块的FBD例子:\*)



(\*等价的 ST 语言程序 :\*)

```

ares := BOO (10);           (* ares 为 “ 真 ” *)
tres := BOO (t#0s);        (* tres 为 “ 假 ” *)
mres := BOO ('false');     (* mres 为 “ 假 ” *)
  
```

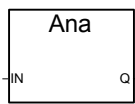
(\*等价的 IL 语言程序 :\*)

```

LD      10
BOO
ST      ares
LD      t#0s
BOO
  
```

ST            tres  
LD           'false'  
BOO  
ST           mres

ANA 变换为整型模拟量



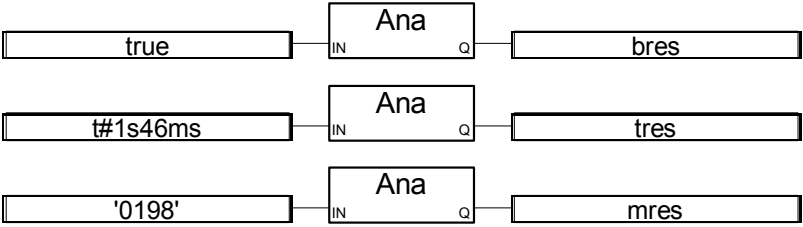
参数:

IN        任意    任意非整型的模拟值  
Q            整型        IN为假则为零 / IN为真则为 1  
                              以毫秒表示的定时器值  
                              实型模拟值的整数部分  
                              以字符串表示的十进制数

说明:

将任意类型的变量变换为整型模拟量

(\*“ 变换为模拟量” 块的 FBD 例子: \*)



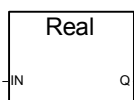
(\*等价的 ST 语言程序: \*)

```
bres := ANA (true);           (* bres is 1 *)
tres := ANA (t#1s46ms);      (* tres is 1046 *)
mres := ANA ('0198');        (* mres is 198 *)
```

(\*等价的 IL 语言程序:\*)

```
LD      true
ANA
ST      bres
LD      t#1s46ms
ANA
ST      tres
LD      '0198'
ANA
ST      mres
```

## REAL 变换为实型



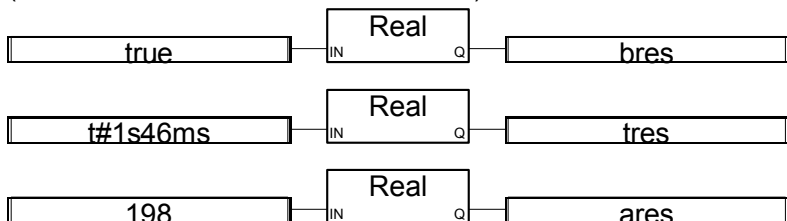
参数:

IN	布尔-整型-	定时器	任意非实型的模拟值 (非信息)
Q	实型	IN 为假则为 0.0 / IN 为真则为 1.0	
		以毫秒计的定时器数值	
		相等的整型数模拟值	

说明:

## 将任意类型的变量变换为实型

(\*“变换为实型”块的FBD例子:\*)



(\*等价的 ST 语言程序:\*)

```

bres := 实型(true);           (* bres 为 1.0 *)
tres := 实型(t#1s46ms);       (* tres 为 1046.0 *)
ares := 实型(198);            (* ares 为 198.0 *)

```

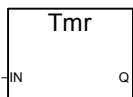
(\*等价的 IL 语言程序:\*)

```

LD      true
REAL
ST      bres
LD      t#1s46ms
REAL
ST      tres
LD      198
实型
ST      ares

```

TMR 变换为定时器型



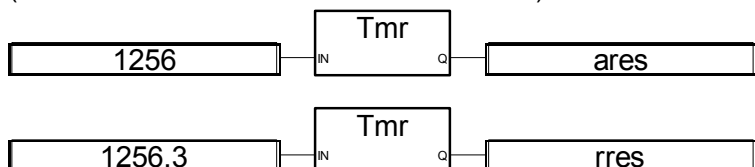
参数:

IN	整型-实型	任意非定时器的值
		IN (若 IN 为实型, 则取 IN 的整数部分)
		为毫秒数
Q	定时器	IN 代表的延时值

说明:

### 将任意的模拟变量变换为定时器型

(\* “变换为定时器型”块的FBD例子：\*)



(\*等价的 ST 语言程序 : \*)

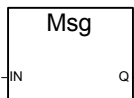
```
ares := TMR (1256);           (* ares := t#1s256ms *)
rres := TMR (1256.3);         (* rres := t#1s256ms *)
```

(\* 等价的 IL 语言程序: \*)

LD	1256
TMR	
ST	ares
LD	1256.3
TMR	

ST            rres

## MSG 变换为信息



参数:

IN        布尔-

整型-实型 任意非信息的值

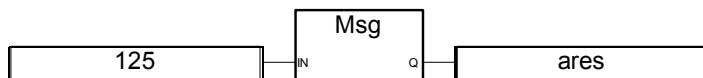
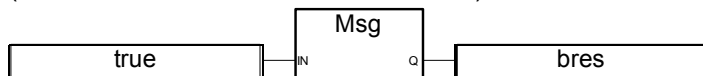
Q            信息        布尔逻辑的“假”或“真”状态

如果IN为模拟量，则为十进制表示的数字

说明:

将任意变量变换为信息

(\*“变换为信息”块的FBD例子:\*)



(\*等价的ST语言程序:\*)

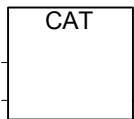
bres := 信息(true); (\* bres 为 ‘真’ \*)

ares := 信息(125); (\* ares 为 ‘125’ \*)

(\*等价的IL语言程序:\*)

LD            true  
MSG  
ST            bres  
LD            125  
MSG  
ST            ares

### CAT 信息串联



注意：此运算的输入可以扩展到两个以上。

参数:

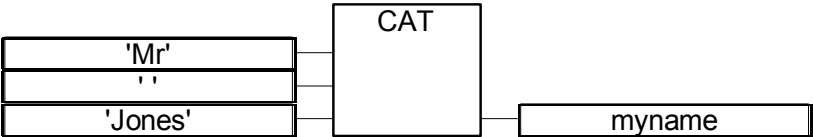
(输入) 信息 (所有输入信息相加后的长度不得超过输出信息的最大长度)

输出            信息            输入信息的串联

说明:

将几个信息串成一个

(\*“ 信息串联” 块的 FBD 例子 : \*)



(\*等价的 ST 语言程序：使用 + 运算符 \*\*)

```
myname := ('Mr' + ' ') + 'Jones';
```

(\* means: myname := 'Mr Jones' \*)

(\*等价的 IL 语言程序：\*)

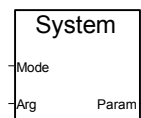
```
LD      'Mr'
```

```
ADD     ''
```

```
ADD     'Jones'
```

```
ST      myname
```

## SYSTEM 系统访问



参数:

**Mode 整型** 确定系统参数和访问模式

**Arg 整型** 定时器“写”访问的新值

**Param 整型** 所访问参数的值

说明:

存取系统参数

下列是 SYSTEM 函数的变量命令（预定义的关键词）列表：

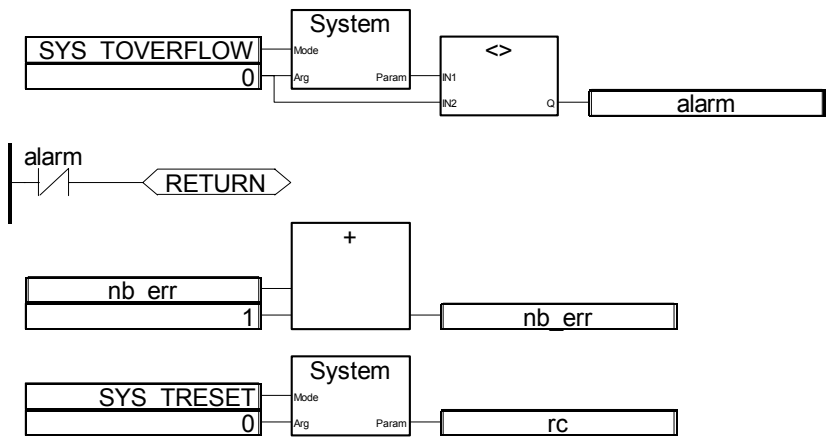
命令	意义
----	----

SYS_TALLOWED	读允许的循环周期
SYS_TCURRENT	读当前循环周期
SYS_TMAXIMUM	读最大循环周期
SYS_TOVERFLOW	读循环周期溢出
SYS_TRESET	复位周期计数器
SYS_TWRITE	改变循环周期
SYS_ERR_TEST	检查运行周期错误
SYS_ERR_READ	读最早的运行周期错误

下列是执行SYSTEM函数的预定义函数所需的参数：

命令	参数	返回值
SYS_TALLOWED	0	允许的循环周期
SYS_TCURRENT	0	当前循环周期
SYS_TMAXIMUM	0	检测到的最大循环周期
SYS_TOVERFLOW	0	循环周期超出次数
SYS_TRESET	0	0
SYS_TWRITE	新的允许 循环周期	写入的时间值
SYS_ERR_TEST	0	如果未检测到错误则为0
SYS_ERR_READ	0	最早错误代码

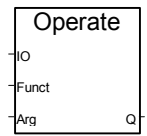
(\*“系统访问”块的FBD例子：\*)



(\*等价的 ST 语言程序 : \*)

```
alarm := (SYSTEM (SYS_TOVERFLOW, 0) <> 0);  
If (alarm) Then  
    nb_err := nb_err + 1;  
    rc := SYSTEM (SYS_TRESET, 0);  
End_If;
```

## OPERATE 操作



参数:

- IO 任意 输入或输出变量
- Funct 整型 执行的操作
- Arg 整型 I/O 操作的参数
- Q 整型 返回校验

说明:

访问 I/O 通道

OPERATE 参数的意义取决于 I/O 界面设备。请参看您的硬件手册或相应的 I/O 板技术注释以了解有关 OPERATE 函数的进一步情况。

## B.9.2 标准功能块

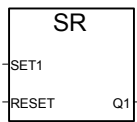
ISaGRAF 统支持预定义的，不用在库中声明的标准功能块。

布尔 .....	SR	置位优先双稳态触发器
	RS	复位优先双稳态触发器
	R_Trig	上升沿检测
	F_Trig	下降沿检测
	SEMA	信号传送器
计数器 .....	CTU	上升计数器
	CTD	下降计数器
	CTUD	计数器
计时器	TON	接通延时
	TOF	断开延时
	TP	脉冲延时
整型模拟 .....	CMP	完全比较器
	StackInt	整型模拟堆栈
实型模拟 .....	AVERAGE	连续均值
	HYSTER	滞后
	LIM_ALRM	超限报警

INTEGRAL 积分  
DERIVATE 微分  
信号发生器.....BLINK 闪烁信号  
SIG\_GEN 信号发生器

注意: 当 "C" 函数创建后，可以从 FBD 语言程序调用。.

SR 置位双稳态触发器



参数:

SET1 布尔 如果为“真”，将 Q1 置位为“真”（优先）  
RESET 布尔 如果为“真”，将 Q1 复位为“假”  
Q1 布尔 布尔存储状态

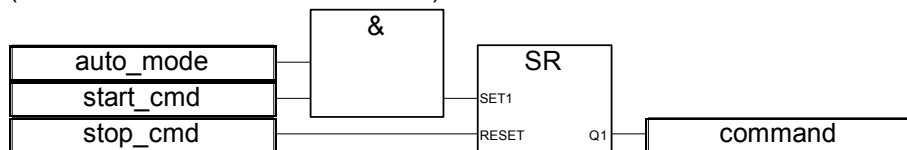
说明:

置位双稳态触发器；参看下列真值表：

<i>Set1</i>	<i>Reset</i>	<i>Q1</i>	<i>输出结果 Q1</i>
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1

1	1	0	1
1	1	1	1

(\* 使用“SR”块的FBD例子：\*)



(\*等价的ST语言程序：我们假定SR1为SR块的一个实例\*)

```
SR1((auto_mode & start_cmd), stop_cmd);
```

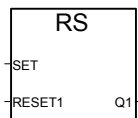
```
command := SR1.Q1;
```

(\* 等价的IL语言程序: \*)

```

LD      auto_mode
AND     start_cmd
ST      SR1.set1
LD      stop_cmd
ST      SR1.reset
CAL     SR1
LD      SR1.Q1
ST      command
  
```

## RS复位双稳态触发器



SET 布尔 如果为“真”，将Q1置位为“真”

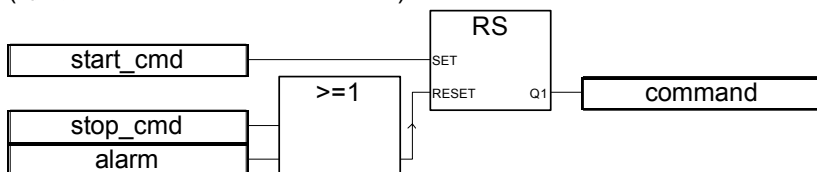
RESET1	布 尔	如 果 为 “ 真 ” ， 将 Q1 复 位 为 “ 假 ” （ 优 先 ）
--------	-----	---

说明:

复位双稳态触发器：参看下列真值表：

<i>Set</i>	<i>Reset1</i>	<i>Q1</i>	输出结果 <i>Q1</i>
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

(\*使用“RS”块的FBD例子\*)



(\*等价的 ST 语言程序：我们假定 RS1 为 RS 块的一个实例\*)

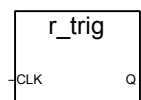
```
RS1(start_cmd, (stop_cmd OR alarm));
```

```
command := RS1.Q1;
```

(\*等价的 IL 语言程序:\*)

```
LD      start_cmd
ST      RS1.set
LD      stop_cmd
OR      alarm
ST      RS1.reset1
CAL     RS1
LD      RS1.Q1
ST      command
```

## R\_TRIG 上升沿检测



参数:

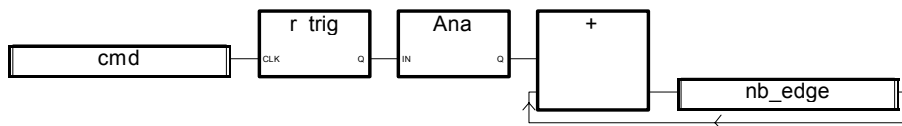
CLK 布尔 任何布尔变量

Q 布尔 当 CLK 从“假”变为“真”时，输出  
为“真”  
其它情况下为“假”

说明:

检测布尔变量的上升沿跳变

(\*“ R\_TRIG” 块的 FBD 例子:\*)



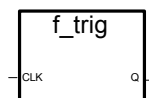
(\*等价的 ST 语言程序：我们假定 R\_TRIG1 是 R\_TRIG 块的一个实例\*)

```
R_TRIG1(cmd);
nb_edge := ANA(R_TRIG1.Q) + nb_edge;
```

(\*等价的 IL 语言程序:\*)

```
LD      cmd
ST      R_TRIG1.clk
CAL     R_TRIG1
LD      R_TRIG1.Q
ANA
ADD     nb_edge
ST      nb_edge
```

## F\_TRIG 下降沿检测



参数:

CLK 布尔 任何布尔变量

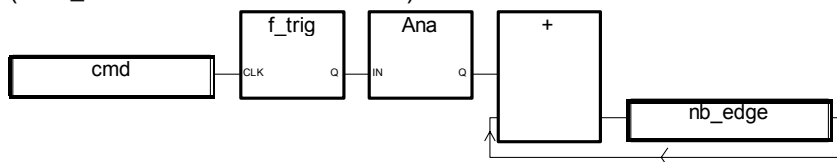
Q 布尔 当 CLK 从“真”变为“假”时，输出为“真”

其它情况下为“假”

说明:

检测布尔变量的下降沿

(\*“ F\_TRIG” 块的 FBD 例子 : \*)



(\*等价的 ST 语言程序 : 我们假定 F\_TRIG1 是 F\_TRIG 块的一个实例 \*)

```
F_TRIG1(cmd);
```

```
nb_edge := ANA(F_TRIG1.Q) + nb_edge;
```

(\* 等价的 IL 语言程序: \*)

```
LD      cmd
ST      F_TRIG1.clk
CAL     F_TRIG1
LD      F_TRIG1.Q
ANA
ADD     nb_edge
ST      nb_edge
```

SEMA 信号发送器



参数:

CLAIM 布尔 “测试和置位”命令

RELEASE 布尔 释放信号

BUSY 布尔 信号状态

说明:

(\*"x"是一个初始化为“假”的布尔变量\*)

busy := x;

If claim Then

    x := True;

Else

    If release Then

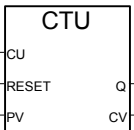
        busy := False;

        x := False;

    End\_if;

End\_if;

## CTU 上升计数器



参数:

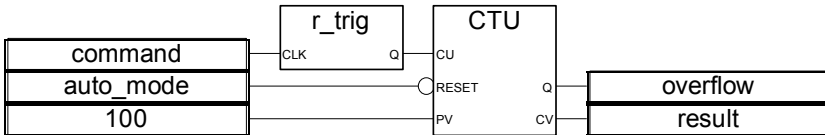
CU	布尔	计数输入 ( 当 CU 为 ‘ 真 ’ 时计数 )
RESET	布尔	复位命令 ( 优先 )
PV	整型	可编程的最大值
Q	布尔	溢出 : 当 $CV = PV$ 时为 “ 真 ”
CV	整型	计数器当前结果

**警告：** CTU 块不能检测计数输入 ( CU ) 的上升或下降沿，它必须与 “ R\_TRIG ” 或 “ R\_TRIG ” 块组合起来才能成为一个脉冲计数器。

说明:

从 0 开始以 1 为步长计 ( 整型 ) 数

(“ CTU ” 块的 FBD 例子 : \*)



(\*等价的 ST 语言程序：我们假定 R\_TRIG1 是 R\_TRIG 块的一个实例；CTU1 是 CTU 块的一个实例\*)

```
CTU1(R_TRIG1(command),NOT(auto_mode),100);
```

```
overflow := CTU1.Q;
```

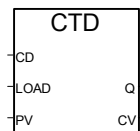
```
result := CTU1.CV;
```

(\*等价的 IL 语言程序: \*)

```
LD      command
ST      R_TRIG1.clk
CAL     R_TRIG1
```

LD	R_TRIG1.Q
ST	CTU1.cu
LDN	auto_mode
ST	CTU1.reset
LD	100
ST	CTU1.pv
CAL	CTU1
LD	CTU1.Q
ST	overflow
LD	CTU1.cv
ST	result

## CTD 下降计数器



参数:

CD 布尔 计数输入

(当 CD 为‘真’时下降计数)

LOAD 布尔

装入命令 ( 优先 )

(当 LOAD 为‘真’时，使 CV = PV)

PV 整型

可编程的初始值

Q 布尔

下溢：当 CV = 0 时为“真”

CV 整型

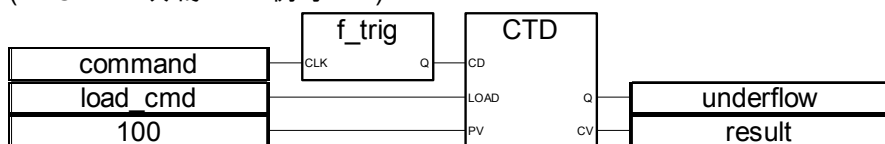
计数结果

**警告：**CTD块不能检测计数输入（CD）的上升或下降沿，它必须与“R\_TRIG”或“F\_TRIG”块组合起来才能成为一个脉冲计数器。

说明:

从给定值开始向下以1为步长计（整型）数

(\*“CTD”块的FBD例子：\*)



(\*等价的ST语言程序：我们假定F\_TRIG1是F\_TRIG块的一个实例；CTD1是CTD块的一个实例\*)

```
CTD1(F_TRIG1(command),load_cmd,100);
```

```
underflow := CTD1.Q;
```

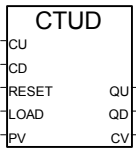
```
result := CTD1.CV;
```

(\*等价的IL语言程序：\*)

```
LD      command
ST      F_TRIG1.clk
CAL     F_TRIG1
LD      F_TRIG1.Q
ST      CTD1.cd
LD      load_cmd
ST      CTD1.load
LD      100
ST      CTD1.pv
```

CAL	CTD1
LD	CTD1.Q
ST	underflow
LD	CTD1.cv
ST	result

### CTUD 上升 - 下降计数器



#### 参数:

CU	布尔	上升计数 (当 CU 为 ‘真’ 时)
CD	布尔	下降计数 (当 CD 为 ‘真’ 时)
RESET	布尔	复位命令 (优先) (当 RESET 为 ‘真’ 时, CV = 0)
LOAD	布尔	装入命令 (当 LOAD 为 ‘真’ 时, 使 CV = PV)
PV	整型	可编程最大值
QU	布尔	溢出: 当 CV = PV 时为 “真”
QD	布尔	下溢: 当 CV = 0 时为 “真”
CV	整型	计数结果

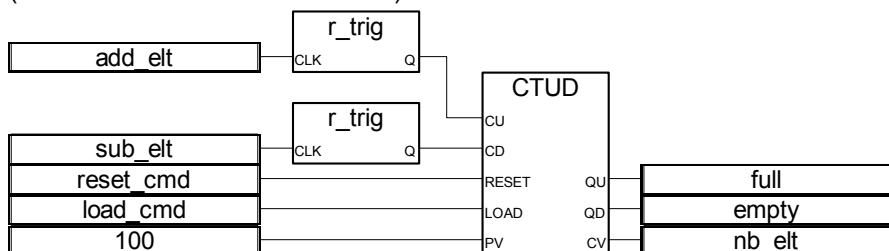
**警告:** CTUD 块不能检测计数输入 (CU 和 CD) 的上升或下降沿, 它必须与 “R\_TRIG” 或 “R\_TRIG” 块组合起来才能成为一个脉冲计数器。

说明:

从0开始以1为步长向上计 ( 整型 ) 数

或从一个给定值开始向下, 以1为步长计数

(\*“ CTUD” 块的 FBD 例子 : \*)



(\* 等价的 ST 语言程序 : 我们假定 R\_TRIG1 和 R\_TRIG2 是 R\_TRIG 块的两个实例 ; CTUD1 是 CTUD 块的实例\*)

```
CTUD1(R_TRIG1(add_elt), R_TRIG2(sub_elt), reset_cmd, load_cmd,100);
```

```
full := CTUD1.QU;
```

```
empty := CTUD1.QD;
```

```
nb_elt := CTUD1.CV;
```

(\*等价的 IL 语言程序: \*)

```
LD      add_elt
ST      R_TRIG1.clk
CAL     R_TRIG1
LD      R_TRIG1.Q
ST      CTUD1.cu
LD      sub_elt
ST      R_TRIG2.clk
CAL     R_TRIG2
```

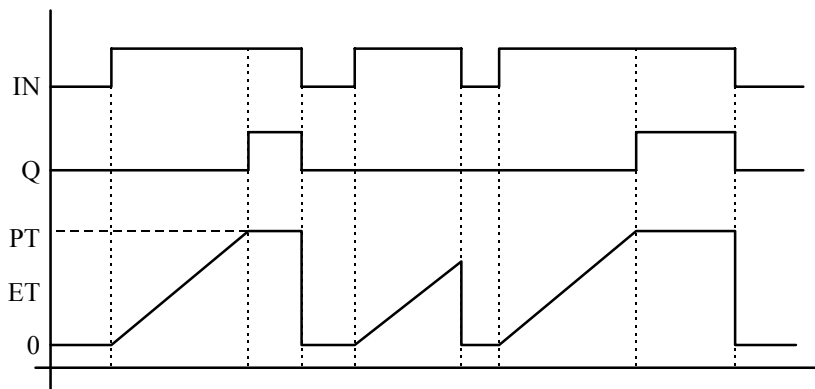


Q	布尔	如果延时到达，则为“真”
ET	定时器	当前经过的延时时间

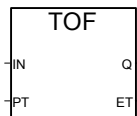
说明:

对内部定时器进行增计时，直到到达设定的延时时间。

时序图:



## TOF 关延时定时



参数:

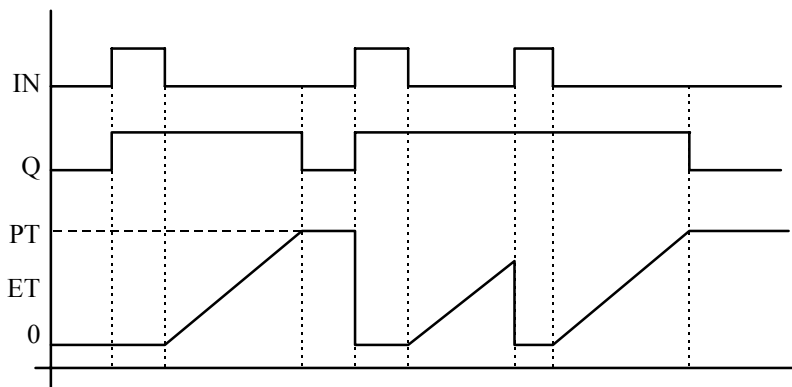
IN	布尔	如果是下降沿，则启动内部定时器
		如果是上升沿，则停止并复位定时器
PT	定时器	最大可变程时间
Q	布尔	延时未结束时为“真”

ET                      定时器    当前经过的延时时间

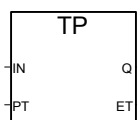
说明:

内部定时器进行升计时直到设定值。

时序图:



## TP 脉冲定时



参数:

IN    布尔    如果为上升沿，启动内部上升定时器（如果并非正在上升）

如果为“假”，并且定时时间已到达，  
复位内部定时器

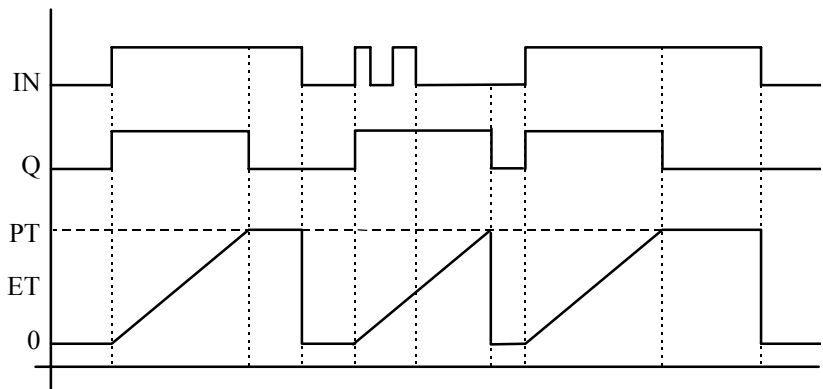
计时的过程中IN的任何变化都无影响

PT	定时器	最大可编程时间
Q	布尔	定时器正在计时时为“真”
ET	定时器	当前经过的延时时间

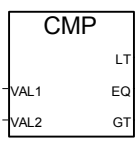
说明:

内部定时器增计时直到设定值。

时序图:



CMP 比较器



参数:

VAL1 整型 任意有符号整型模拟值

VAL2 整型 任意有符号整型模拟值

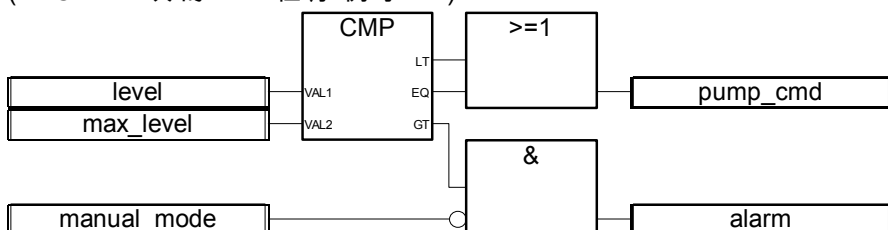
LT 布尔 如果 val1 小于 val2，则为“真”

EQ	布尔	等于 val2, 则为“真”
GT	布尔	如果 val1 大于 val2 则为“真”

说明:

比较第一个值是否等于, 小于或大于第二个值。

(\*“CMP”块的FBD程序例子: \*)



(\*等价的ST语言程序: 假定CMP1为CMP块的一个实例\*)

```
CMP1(level, max_level);
```

```
pump_cmd := CMP1.LT OR CMP1.EQ;
```

```
alarm := CMP1.GT AND NOT(manual_mode);
```

(\*等价的IL语言程序: \*)

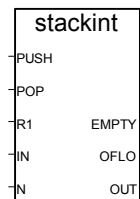
```

LD      level
ST      CMP1.val1
LD      max_level
ST      CMP1.val2
CAL     CMP1
LD      CMP1.LT
OR      CMP1.EQ
ST      pump_cmd
LD      CMP1.GT

```

ANDN      manual\_mode  
ST          alarm

## STACKINT 整型堆栈



参数:

PUSH 布尔 压入命令 ( 只在上升沿时 )

将 IN 的值加入到堆栈顶部

POP 布尔 弹出命令 ( 只在上升沿时 )

删除堆栈的最后一次压入的 ( 顶部 )  
一个值

R1 布尔 将堆栈复位为空状态

IN 整型 所压入堆栈的值

N 整型 应用程序定义的堆栈大小

EMPTY 布尔 如果堆栈为空则为“真”

OFLO 布尔 溢出: 如堆栈溢出则为“真”

OUT 整型 堆栈顶部的数值

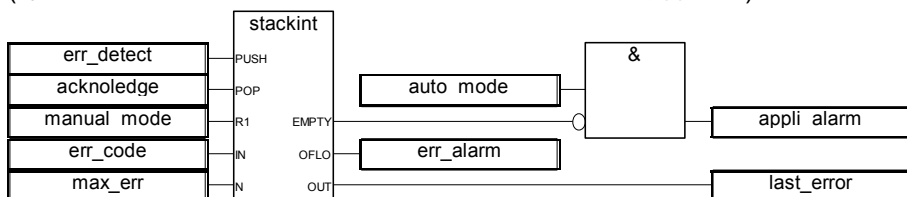
说明:

管理整型数值堆栈。

“STACKINT”块在执行PUSH和POP命令时包含了上升沿检测。堆栈的最大容量为128，所以由应用程序定义的堆栈大小必须在1和128之间。

注意OFLO值只在复位后有效(R1至少为“真”一次并返回“假” )

(\*使用“STACKINT”块的FBD程序例子：错误管理\*)



(\*等价的ST语言程序：假定STACKINT1是STACKINT块的一个实例\*)

```

STACKINT1(err_detect, acknowledge, manual_mode, err_code, max_err);
appli_alarm := auto_mode AND NOT(STACKINT1.EMPTY);
err_alarm := STACKINT1.OFLO;
last_error := STACKINT1.OUT;
  
```

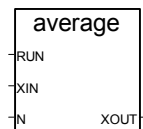
(\*等价的IL语言程序：\*)

```

LD      err_detect
ST      STACKINT1.push
LD      acknowledge
ST      STACKINT1.pop
LD      manual_mode
ST      STACKINT1.r1
  
```

LD	err_code
ST	STACKINT1.IN
LD	max_err
ST	STACKINT1.N
CAL	STACKINT1
LD	auto_mode
ANDN	STACKINT1.empty
ST	appli_alarm
LD	STACKINT1.OFLO
ST	err_alarm
LD	STACKINT1.OUT
ST	last_error

### AVERAGE 连续均值



#### 参数:

RUN	布尔	运行 = “真” / 复位 = “假”
XIN	实型	任意实型模拟变量
N	整型	应用程序定义的采样数目
XOUT	实型	求出XIN的平均值

#### 说明:

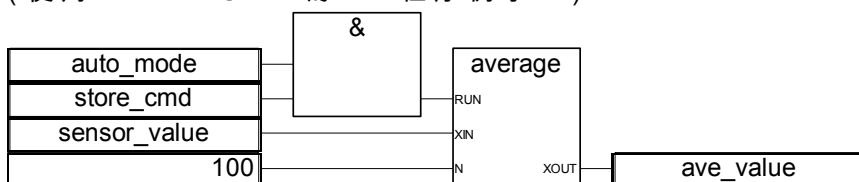
每周期存储一个值，并计算所有存储值的平均值。只存储 N 个值。

采样数目 N 不能超过 128.

如果“运行”命令为“假”（复位状态）

存储的值达到 N 值时，第一个存储的值会被最后一个值所取代。

(\*使用“ AVERAGE ” 的 FBD 程序例子：\*)



(\*等价的 ST 语言程序: AVERAGE1 是 AVERAGE 块的一个实例\*)

```
AVERAGE1((auto_mode & store_cmd), sensor_value, 100);
```

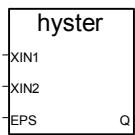
```
ave_value := AVERAGE1.XOUT;
```

(\*等价的 IL 语言程序：\*)

```
LD      auto_mode
AND     store_cmd
ST      AVERAGE1.run
LD      sensor_value
ST      AVERAGE1.xin
LD      100
ST      AVERAGE1.N
CAL     AVERAGE1
LD      AVERAGE1.XOUT
```

ST                      ave\_value

HYSTER 滞后



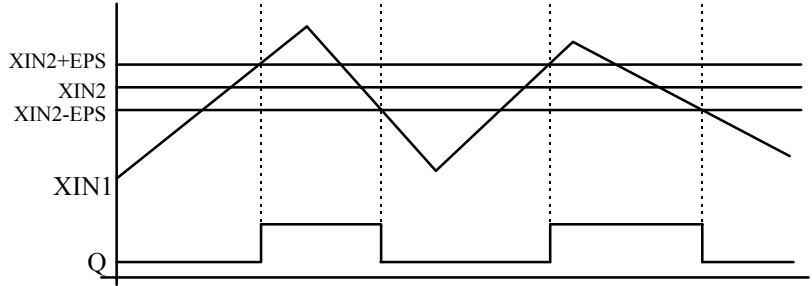
参数:

- XIN1    实型    任意实型模拟量值
- XIN2    实型    测试 XIN1 是否超过 XIN2+EPS
- EPS    实型    滞后值 ( 必须大于零 )
- Q    布尔    当 XIN1 超出 XIN2+EPS 并且不小于 XIN2-EPS 时为“ 真”

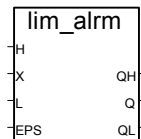
说明:

实型值高限幅的滞后操作

时序图例子 :



## LIM\_ALARM 超限报警



参数:

H      实型    高限幅值

X            实型    输入: 任意实型模拟值

L            实型    低限幅值

EPS          实型    滞后值 ( 必须大于零 )

QH           布尔    “ 高 ” 报警: 当 X 超出高限幅 H 时为“真”

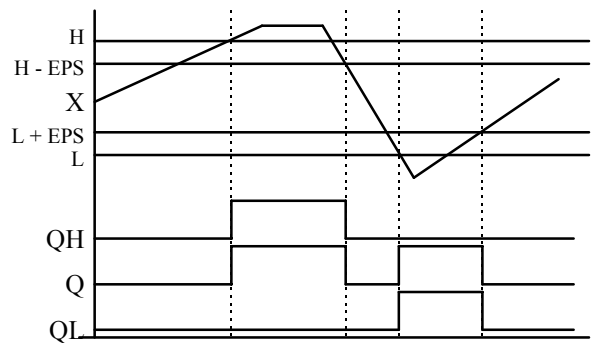
Q            布尔    报警输出: 当 X 超出限幅时为“ 真 ”

QL           布尔    “ 低 ” 报警: 当 X 低于低限幅 L 时为“真”

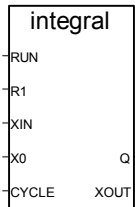
说明:

实型值高低限幅的滞后操作

无论高低限幅的滞后差值都为 EPS 参数的一半。以下是一个示例的时序图:



INTEGRAL 积分



参数:

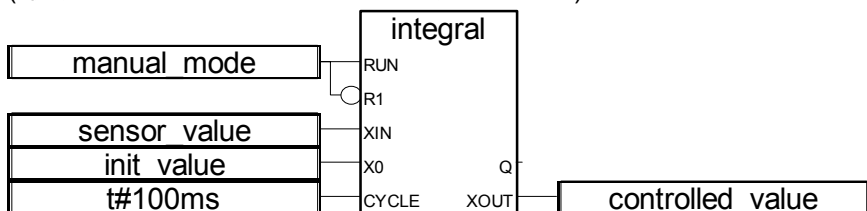
RUN	布尔	模式: 为“真”时积分 / 为“假”时保持
R1	布尔	主导复位
XIN	实型	输入: 任意实型模拟值
X0	实型	初始值
CYCLE	TMR	采样周期
Q	布尔	R1 的非
XOUT	实型	积分输出

说明:

对实型值的积分

如果“ CYCLE ” 参数值小于 ISaGRAF 应用程序的循环周期，则采用程序的循环周期。

(\*使用“ INTEGRA ” 块的 FBD 程序例子：\*)



(\*等价的 ST 语言程序： INTEGRAL1 是 INTEGRAL 块的一个实例\*)

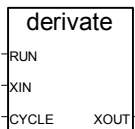
```
INTEGRAL1(manual_mode, NOT(manual_mode), sensor_value, init_value,
t#100ms);
```

```
controlled_value := INTEGRAL1.XOUT;
```

(\* IL Equivalence: \*)

```
LD      manual_mode
ST      INTEGRAL1.run
STN     INTEGRAL1.R1
LD      sensor_value
ST      INTEGRAL1.XIN
LD      init_value
ST      INTEGRAL1.X0
LD      t#100ms
ST      INTEGRAL1.CYCLE
CAL     INTEGRAL1
LD      INTEGRAL1.XOUT
ST      controlled_value
```

## DERIVATE 微分



参数:

RUN 布尔 模式：“真”为正常/“假”为复位

XIN 实型 输入：任意实型模拟值

CYCLE 定时器 采样周期

XOUT 实型 微分输出

说明:

对实型值求微分。

如果“CYCLE”参数小于ISaGRAF应用程序的循环周期，则采用程序的循环周期。

(\*使用“DERIVATE”块的FBD程序例子：\*)



(\*等价的ST语言程序：DERIVATE1是DERIVATE块的一个实例\*)

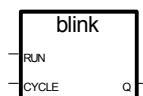
DERIVATE1(manual\_mode, sensor\_value, t#100ms);

```
derivated_value := DERIVATE1.XOUT;
```

(\*等价的 IL 语言程序:\*)

```
LD      manual_mode
ST      DERIVATE1.run
LD      sensor_value
ST      DERIVATE1.XIN
LD      t#100ms
ST      DERIVATE1.CYCLE
CAL     DERIVATE1
LD      DERIVATE1.XOUT
ST      derivated_value
```

## BLINK 闪烁信号



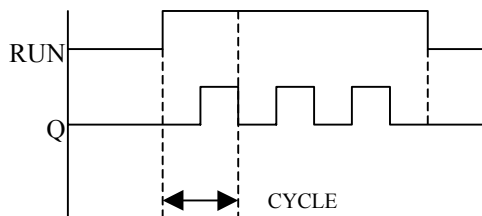
参数:

RUN 布尔 模式:“真”为闪烁/“假”为复位  
 CYCLE 定时器 闪烁周期  
 Q 布尔 输出闪烁信号

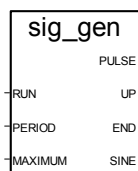
说明:

发生闪烁的信号。

时序图:



## SIG\_GEN 信号发生器



## 参数:

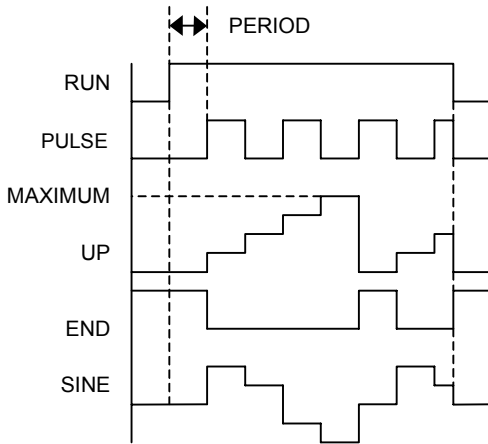
RUN	布尔	模式：“真”为运行/“假”为复位
PERIOD	定时器	采样持续时间
MAXIMUM	整型	最大计数值
PULSE	布尔	每次采样后翻转
UP	整型	上升计数器，每次采样后增加
END	布尔	上升计数终止后为“真”
SINE	实型	正弦信号（周期等于计数周期）

## 说明：

发生多种信号：闪烁的布尔信号，整型上升计数，实型正弦波。

当计数到达最大值时将从0开始，所以END只在一个周期内为“真”。

时序图:



### B.9.3 标准函数

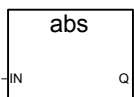
以下是 ISaGRAF 系统支持的标准函数。这些函数已经预定义，并不需要在库中声明。

数学 .....	ABS	绝对值
	EXPT, POW	指数
	LOG	对数
	SQRT	平方根
	TRUNC	实数舍位
三角函数: .....	ACOS, ASIN,	反余弦, 反正弦。
.....	ATAN	反正切
	COS, SIN,	余弦, 正弦。
	TAN	正切
寄存器控制:	ROL, ROR	向左循环移位, 向右循环移位

	SHL, SHR	向左移位, 向右移位
数据处理:.....	MIN, MAX,	最小值, 最大值。
	LIMIT	限幅
	MOD	余数
	MUX4, MUX8	多路选择器 (4 输入 或 8 输入)
	SEL	二进制选择器
	ODD	奇校验
	RAND	随机值
数据变换:.....	ASCII	字符 → ASCII 码
	CHAR	ASCII 码 → 字符
字符串管理: .....	MLEN	取得串长度
	DELETE	删除子串
	INSERT	插入串
	FIND,	查找子串。
	REPLACE	替换子串
	LEFT, MID	提取字符串左边字符, 中间字符
	RIGHT	提取字符串右边字符
	DAY_TIME	日期时间
数组处理:	ARCREATE	建立整型值数组
	ARREAD	读数组元素
	ARWRITE	写数组元素
二进制文件管理.....	F_OPEN	在读模式下打开二进制文件
	F_WOPEN	在写模式下打开二进制文件
	F_CLOSE	关闭二进制文件
	F_EOF	测试二进制文件尾

FA\_READ 从二进制文件中读取模拟值  
 FA\_WRITE 向二进制文件中写入模拟值  
 FM\_READ 从二进制文件中读取信息  
 FM\_WRITE 向二进制文件中写入信息

## ABS 绝对值



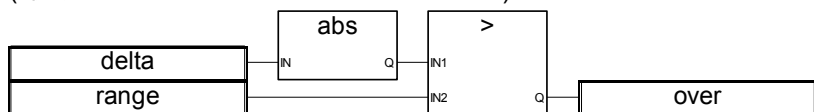
参数:

IN	实型	任意带符号的实型模拟值
Q	实型	绝对值 ( 总为正 )

说明:

给出实型值的绝对值。

(\*使用“ABS”块的FBD程序例子:\*)



(\*等价的 ST 语言程序 : \*)

```
over := (ABS (delta) > range);
```

(\*等价的 IL 语言程序 : \*)

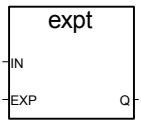
```
LD      delta
```

ABS

GT            range

ST            over

EXPT 指数



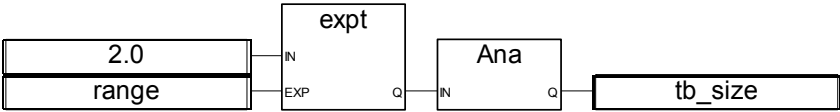
参数:

IN	实型	任意带符号的实型模拟值
EXP	整型	整型模拟指数
Q	实型	(IN EXP)

说明:

给出 [base (底) exponent (指数)] 运算的实型结果；以底为第一个参数，指数为第二个参数。

(\*使用“ EXPT” 块的 FBD 程序例子:\*)



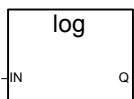
(\*等价的 ST 语言程序:\*)

tb\_size := ANA (EXPT (2.0, range) );

(\*等价的 IL 语言程序 : \*)

```
LD      2.0
EXPT    range
ANA
ST      tb_size
```

## 对数



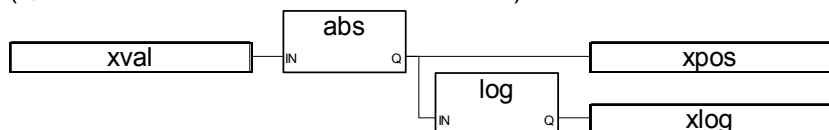
参数:

IN	实型	必须大于零
Q	实型	输入值的以 10 为底的对数

说明:

计算实型值的以 10 为底的对数。

(\*使用“LOG”块的 FBD 程序例子 : \*)



(\*等价的 ST 语言程序 : \*)

```
xpos := ABS (xval);
xlog := LOG (xpos);
```

(\*等价的 IL 语言程序:\*)

LD            xval  
ABS  
ST            xpos  
LOG  
ST            xlog

POW 乘方



参数:

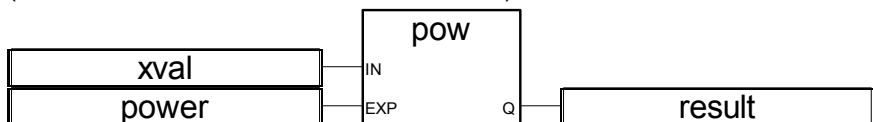
IN	实型	将自乘的模拟值
EXP	实型	幂 ( 指数 )
Q	实型	(IN EXP)

1.0 如果 IN 非 0.0 并且 EXP 为 0.0  
0.0 如果 IN 为 0.0 并且 EXP 为负数  
0.0 如果 IN 和 EXP 都为 0.0  
0.0 如果 IN 为负数并且Y不对应一个整数

说明:

给出 [ base (底) exponent (指数) ] 运算的实型结果；以‘底’为第一个参数，‘指数’为第二个参数。指数为实型值。

(\*使用“ POW” 块的 FBD 程序例子：\*)



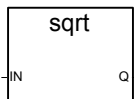
(\*等价的 ST 语言程序：\*)

```
result := POW (xval, power);
```

(\*等价的 IL 语言程序：\*)

```
LD      xval
POW     power
ST      result
```

## SQRT 平方根



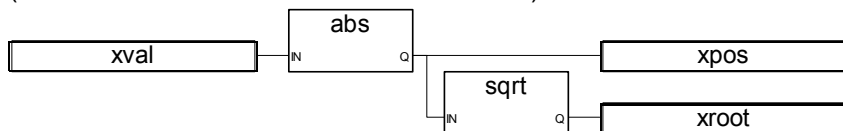
参数:

IN	实型	必须大于或等于零
Q	实型	输入值的平方根

说明:

计算实型值的平方根

(\*使用“SQRT”块的FBD程序例子:\*)



(\*等价的ST语言程序:\*)

xpos := ABS (xval);

xroot := SQRT (xpos);

(\*等价的IL语言程序:\*)

LD xval

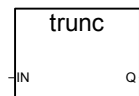
ABS

ST xpos

SQRT

ST xroot

## TRUNC 实数舍位



参数:

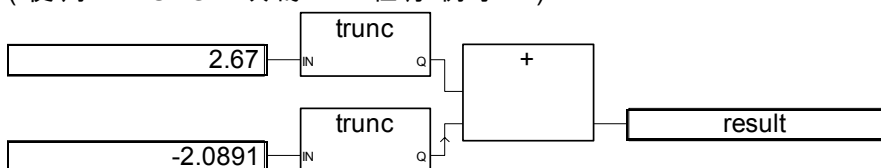
IN 实型 任意 实型模拟值

Q	实型	如果 $IN > 0$ , 则为小于或等于输入的最大整数 如果 $IN < 0$ , 则为大于或等于输入的最小整数
---	----	--

说明:

截取实型数值的整数部分。

(\*使用“ TRUNC” 块的 FBD 程序例子 : \*)



(\*等价的 ST 语言程序 : \*)

result := TRUNC (+2.67) + TRUNC (-2.0891);

(\* means: result := 2.0 + (-2.0) := 0.0; \*)

(\*等价的 IL 语言程序 : \*)

LD 2.67

TRUNC

ST temporary (\*第一个 TRUNC 的中间结果\*)

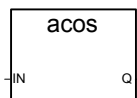
LD -2.0891

TRUNC

ADD temporary

ST result

## ACOS 反余弦



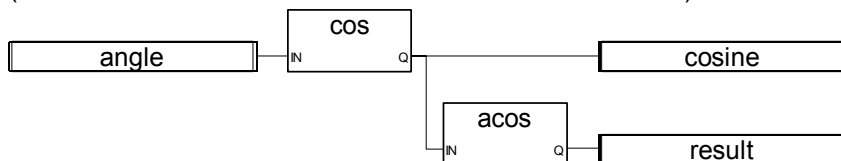
参数:

IN	实型	必须属于集合 $[-1.0 .. +1.0]$
Q	实型	输入值的反余弦 (在集合 $[0.0 .. \text{PI}]$ 中) = 0.0 , 当非法输入时

说明:

计算实型数值的反余弦。

(\*使用“COS”和“ACOS”块的FBD程序例子: \*)



(\*等价的ST语言程序: \*)

cosine := COS (angle);

result := ACOS (cosine); (\*结果等于角度\*)

(\*等价的IL语言程序: \*)

LD            angle

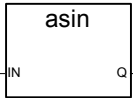
COS

ST            cosine

ACOS

ST      result

ASIN 反正弦 {XE "ASIN"} {XE "Arc sine"}



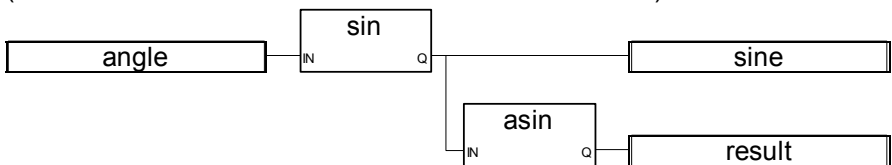
参数:

IN	实型	必须属于集合 $[-1.0 .. +1.0]$
Q	实型	输入值的反正弦 (在集合 $[-\pi/2 .. +\pi/2]$ 中) = 0.0 如果输入非法

说明:

计算实型数值的反正弦。

(\*使用“ SIN” 和“ ASIN” 块的 FBD 程序例子: \*)



(\*等价的 ST 语言程序: \*)

sine := SIN (angle) ;

result := ASIN (sine) ; (\* 结果等于角度 \*)

(\*等价的 IL 语言程序: \*)

LD            angle

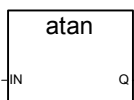
SIN

ST            sine

ASIN

ST        result

## ATAN 反正切



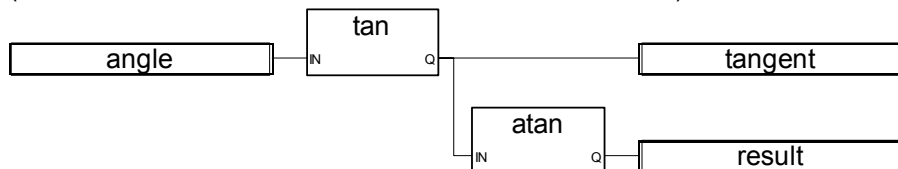
参数：

IN	实型	任意实型模拟值
Q	实型	输入值的反正切（在集合 $[-\pi/2 .. +\pi/2]$ 中） = 0.0，如果输入非法

说明：

计算实型值的反正切。

(\* 使用“TAN”和“ATAN”块的FBD程序例子：\*)



(\* 等价的 ST 语言程序: \*)

```
tangent := TAN (angle);
```

```
result := ATAN (tangent); (* 结果等于 angle*)
```

(\* 等价的 IL 语言程序: \*)

```
LD          angle
```

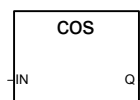
```
TAN
```

```
ST          tangent
```

```
ATAN
```

```
ST    result
```

## COS 余弦



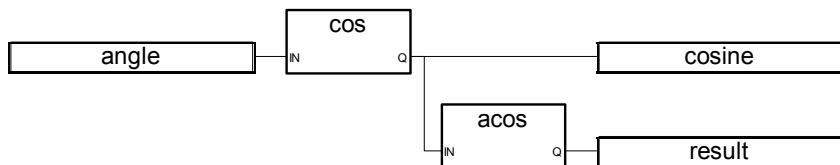
参数:

IN	实型	任意实型模拟值
Q	实型	输入值的余弦 ( 在集合 [-1.0 .. +1.0] 之间 )

说明:

计算实型值的余弦。

(\* 使用 “ COS ” 和 “ ACOS ” 块的 FBD 程序例子: \*)



(\* 等价的 ST 语言程序 : \*)

cosine := COS (angle);

result := ACOS (cosine); (\* 结果 等于 angle \*)

(\* 等价的 IL 语言程序 : \*)

LD            angle

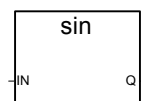
COS

ST            cosine

ACOS

ST            result

## SIN 正弦



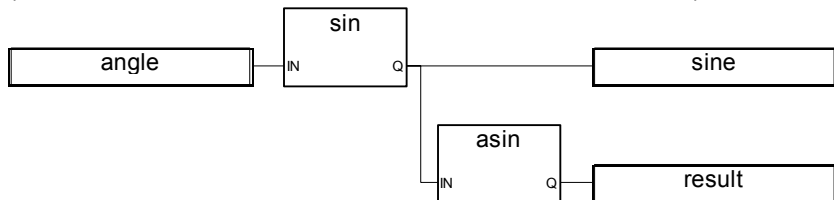
参数 :

IN	实型	任意 实型 模拟值
Q	实型	输入值的正弦 (在集合 [-1.0 .. +1.0] 之间)

说明 :

计算实型值的正弦。

(\* 使用 “ SIN ” 和 “ ASIN ” 块的 FBD 程序例子 : \*)



(\* 等价的 ST 语言程序 : \*)

```
sine := SIN (angle);
```

```
result := ASIN (sine); (* 结果 等于 angle *)
```

(\* 等价的 IL 语言程序 : \*)

```
LD      angle
```

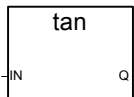
```
SIN
```

```
ST      sine
```

```
ASIN
```

```
ST      result
```

## TAN 正切



参数 :

IN

实型

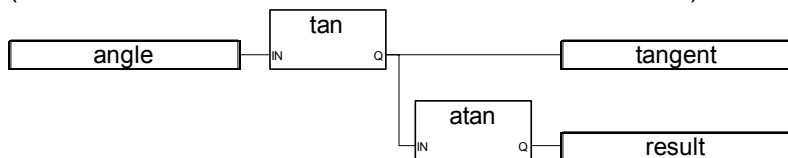
不能 等于 PI/2 至 PI

Q                      实型                      输入值的正切  
= 1E+38, 如果输入无效

说明:

计算实型值的正切

(\* 使用“ TAN” 和“ ATAN” 块的 FBD 程序例子: \*)



(\* 等价的 ST 语言程序: \*)

tangent := TAN (angle);

result := ATAN (tangent); (\* result is equal to angle\*)

(\* 等价的 IL 语言程序: \*)

LD                      angle

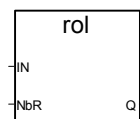
TAN

ST                      tangent

ATAN

ST                      result

## ROL 向左循环移位

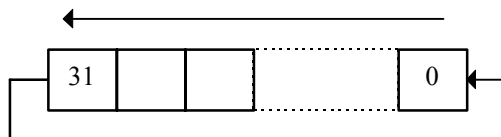


参数：

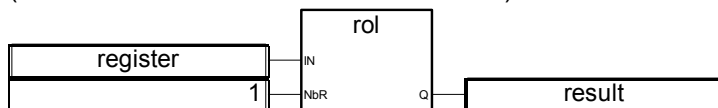
IN	整型	任意整型模拟值
NbR	整型	循环移位的位数 ( 在区间 [1..31] 之中 )
Q	整型	向左移位后的值
NbR ≤ 0 时无操作		

说明：

使整型数值向左循环移位。循环有 32 位同时进行：



(\* 使用“ ROL” 块的 FBD 程序例子：\*)



(\* 等价的 ST 语言程序：\*)

```
result := ROL (register, 1);
```

```
(* register = 2#0100_1101_0011_0101*)
```

```
(* result = 2#1001_1010_0110_1010*)
```

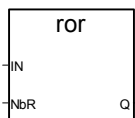
(\*等价的 IL 语言程序：\*)

```
LD      register
```

```
ROL     1
```

```
ST      result
```

ROR 向右循环移位



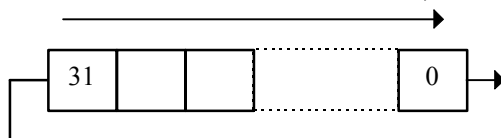
参数：

IN	整型	任意整型数值
NbR	整型	循环移位的位数 (在 [1..31] 之间)
Q	整型	向右移位的值

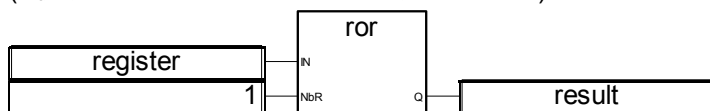
如果 NbR ≤ 0，则无操作

说明：

使整型数值向右循环移位，移位由 32 位同时进行。



(\* 使用 “ ROR ” 块的 FBD 程序例子：\*)



(\* 等价的 ST 语言程序：\*)

```
result := ROR (register, 1);
```

```
(* register = 2#0100_1101_0011_0101 *)
```

```
(* result = 2#1010_0110_1001_1010 *)
```

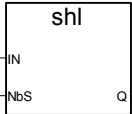
(\*等价的 IL 语言程序：\*)

```
LD      register
```

ROR 1

ST result

## SHL 向左移位



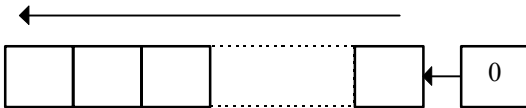
参数：

IN	整型	任意整型模拟值
NbS	整型	移位的位数（在 [1..31] 之间）
Q	整型	向左移位后的值

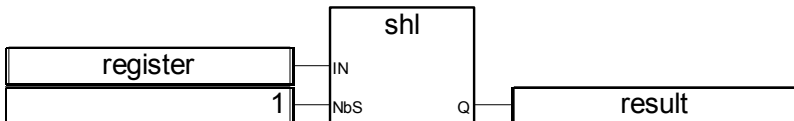
如果 NbS ≤ 0 则无操作  
最低位由 0 替换

说明：

使整型数值向左移位，32 位同时进行：



(\* 使用 “SHL” 块的 FBD 程序例子：\*)



(\* 等价的 ST 语言程序：\*)

```

result := SHL (register,1);
(* register = 2#0100_1101_0011_0101 *)
(* result   = 2#1001_1010_0110_1010 *)

```

(\* 等价的 IL 语言程序: \*)

```

LD      register
SHL     1
ST      result

```

## SHR 向右移位

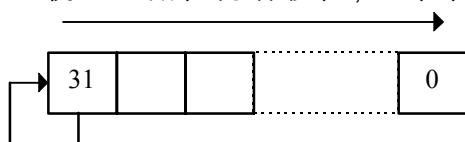


参数：

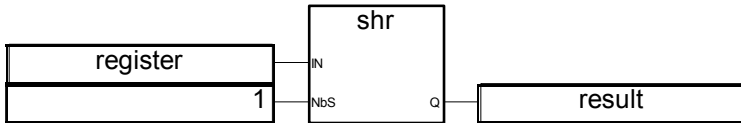
IN	整型	任意整型模拟值
NbS	整型	移位的位数 ( 在 [1..31] 之间 )
Q	整型	向右移位后的值
		如果 NbS ≤ 0 则无操作
		最高位在每次移位时向低位复制

说明：

使整型数值向右移位，32 位同时进行：



(\* 使用“ SHR ”块的 FBD 程序例子 : \*)



(\* 等价的 ST 语言程序 : \*)

```
result := SHR (register,1);
```

```
(* register = 2#1100_1101_0011_0101 *)
```

```
(* result   = 2#1110_0110_1001_1010 *)
```

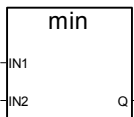
(\* 等价的 IL 语言程序 : \*)

```
LD      register
```

```
SHR     1
```

```
ST      result
```

## MIN 最小值



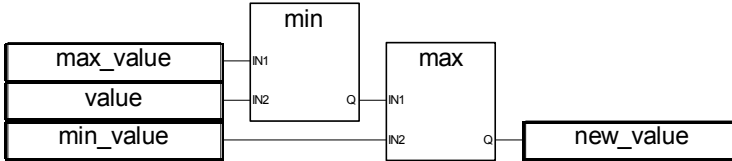
参数 :

IN1	整型	任意带符号的整型模拟值
IN2	整型	( 不能是实型数 )
Q	整型	两个输入值中的最小值

说明 :

给出两个整型值间的最小值。

(\* 使用 “ MIN ” 和 “ MAX ” 块的 FBD 程序例子 : \*)



(\* 等价的 ST 语言程序 : \*)

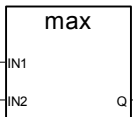
```
new_value := MAX (MIN (max_value, value), min_value);
```

(\* bounds the value to the [min\_value..max\_value] set \*)

(\* 等价的 IL 语言程序 : \*)

```
LD      max_value
MIN     value
MAX     min_value
ST     new_value
```

## MAX 最大值



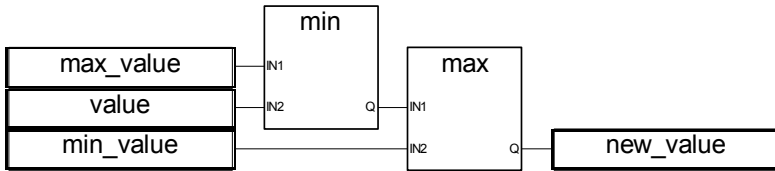
参数 :

IN1	整型	任意带符号的整型模拟值
IN2	整型	( 不能是实型)
Q	整型	两个输入值中的最大值

说明：

给出两个整型数值间的最大值。

(\* 使用 “ MIN ” 和 “ MAX ” 块的 FBD 程序例子：\*)



(\* 等价的 ST 语言程序：\*)

```
new_value := MAX (MIN (max_value, value), min_value);
```

(\* bounds the value to the [min\_value..max\_value] set \*)

(\*等价的 IL 语言程序：\*)

```
LD      max_value
MIN      value
MAX      min_value
ST      new_value
```

## LIMIT 限幅



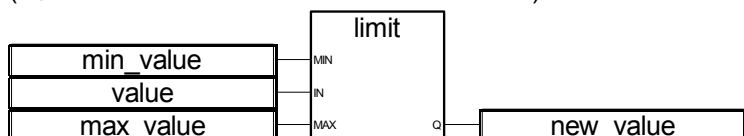
参数：

MIN	整型	最小允许值
IN	整型	任意带符号的模拟值
MAX	整型	最大允许值
Q	整型	被允许范围限制的输入值

说明：

将输入整型值限制在一定范围之内。输入值在限制范围之内时保持原值；超出最大值时即变为最大值；低于最小值时即变为最小值。

(\* 使用“LIMIT”块的FBD程序例子：\*)



(\* 等价的 ST 语言程序：\*)

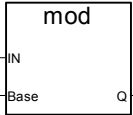
```
new_value := LIMIT (min_value, value, max_value);
```

(\* bounds the value to the [min\_value..max\_value] set \*)

(\* 等价的 IL 语言程序：\*)

```
LD      min_value
LIMIT   value, max_value
ST      new_value
```

## MOD 余数



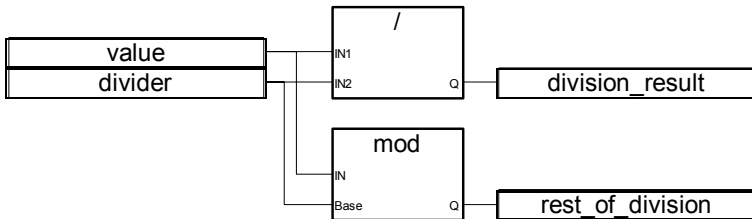
参数：

IN	整型	任意带符号的整型模拟值
Base	整型	必须大于零
Q	整型	求余数计算（以输入的 base 为底） 如果 Base ≤ 0 则返回 -1

说明：

计算整型值的余数。

(\* 使用“MOD”块的 FBD 程序例子：\*)



(\* 等价的 ST 语言程序：\*)

division\_result := (value / divider); (\* 整数除法\*)

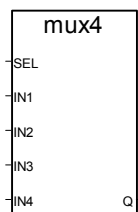
rest\_of\_division := MOD (value, divider); (\* 除法的余数 \*)

(\* 等价的 IL 语言程序：\*)

```
LD      value
DIV     divider
ST      division_result
LD      value
```

MOD          divider  
ST          rest\_of\_division

## MUX4 4 路 输 入 选 择 器



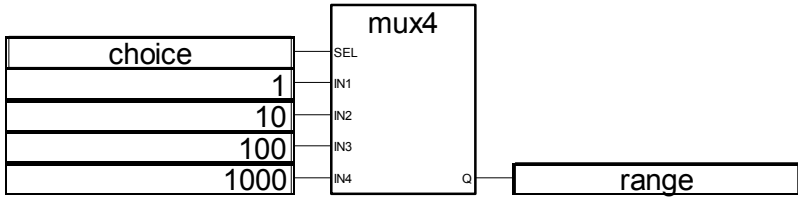
参 数：

SEL	整 型	选 择 器 整 型 值 ( 必 须 在 [0..3] 之 间 )
IN1..IN4	整 型	任 意 的 整 型 模 拟 值
Q	整 型	= 值 IN1 当 SEL = 0 时 = 值 IN2 当 SEL = 1 时 = 值 IN3 当 SEL = 2 时 = 值 IN4 当 SEL = 3 时 = 0 当 SEL 为 所 有 其 它 值 时

说 明：

4 输 入 多 路 选 择 器：在 4 个 整 型 值 中 选 择 一 个。

(\* 使 用 “ MUX4 ” 块 的 FBD 程 序 例 子：\*)



(\* 等价的 ST 语言程序 : \*)

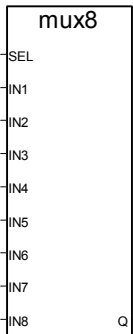
```
range := MUX4 (choice, 1, 10, 100, 1000);
```

(\* 从 4 个预定的范围 ( range ) 中选定一个, 例如, 当 choice 为 1 时, range 将为 10 \*)

(\* 等价的 IL 语言程序 : \*)

```
LD      choice
MUX4    1,10,100,1000
ST      range
```

## MUX8 8 路输入选择器



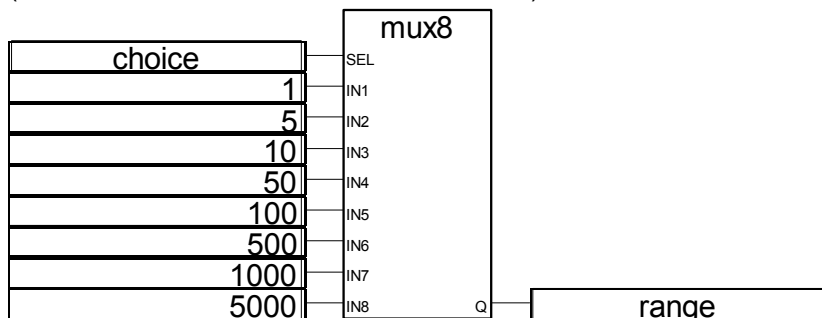
参数：

SEL	整型	选择器整型值（必须在 [0..7] 之间）
IN1..IN8	整型	任意整型模拟值
Q	整型	= 值 IN1 当 SEL = 0 时 = 值 IN2 当 SEL = 1 时 ... = 值 IN8 当 SEL = 7 时 = 0 当 SEL 为其它任何值时

说明：

8 输入多路选择器：从 8 个整型值中选择一个。

(\*使用“MUX8”块的 FBD 程序例子：\*)



(\*等价的 ST 语言程序：\*)

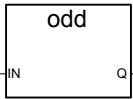
```
range := MUX8 (choice, 1, 5, 10, 50, 100, 500, 1000, 5000);
```

(\*从 8 个预定义的范围 (range) 中选择一个，例如 choice 为 3 时，range 将为 50 \*)

(\*等价的 IL 语言程序：\*)

LD            choice  
 MUX8        1,5,10,50,100,500,1000,5000  
 ST        range

## ODD 奇 校 验



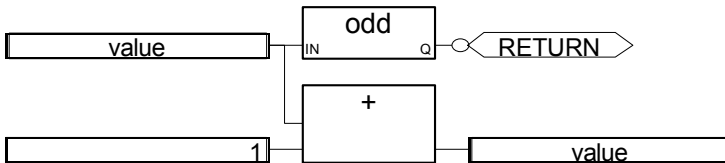
参 数 :

IN	整 型	任意带符号的整型模拟值
Q	布 尔	输入值为奇时，为“真” 输入值为偶时为“假”

说 明 :

测试整型值的奇偶，结果为奇或偶。

(\* 使用“ ODD” 块的 FBD 程序 例子 : \*)



(\* 等价的 ST 语言 程序 : \*)

```
If Not (ODD (value)) Then Return; End_if;
```

```
value := value + 1;
```

(\* 使 值 总 为 偶 \*)

(\* 等价的 IL 语言程序: \*)

LD           value

ODD

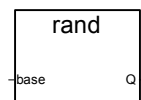
RETNC

LD           value

ADD          1

ST     value

## RAND 随机值



参数:

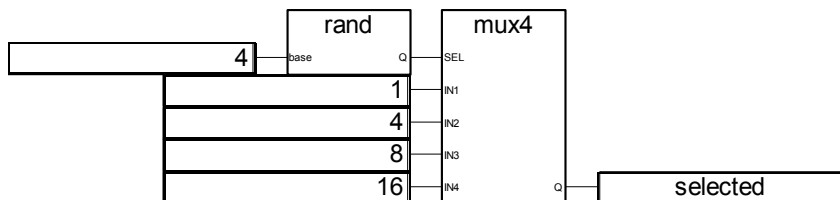
base           整型       定义数值的允许区间

Q              整型       在区间 [0..base-1] 之间的随机值

说明:

给出一个在给定范围内的随机整型数值。

(\* 使用 “ RAND ” 块的 FBD 程序例子: \*)



(\* 等价的 ST 语言程序 : \*)

```
selected := MUX4 ( RAND (4), 1, 4, 8, 16 );
```

(\*

自 4 个预定义数值中随机选择一个

RAND 调用的结果是在区间 [0..3] 中的数值 ,

所以 MUX4 的选择结果 ' selected' 将为 ' 随机地' 取得的值

1 , 当 RAND 的结果为 0 时

4 , 当 RAND 的结果为 1 时

8 , 当 RAND 的结果为 2 时

16 , 当 RAND 的结果为 3 时

\*)

(\* 等价的 IL 语言程序 : \*)

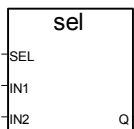
```
LD      4
```

```
RAND
```

```
MUX4    1,4,8,16
```

```
ST      selected
```

## SEL 二进制选择器



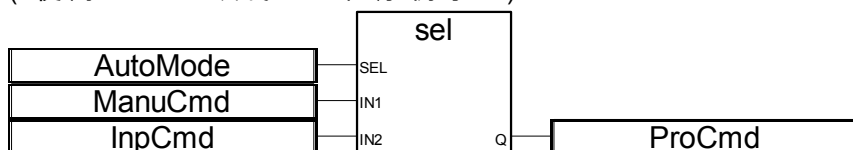
参数：

SEL	布尔	指出所选择的值
IN1, IN2	整型	任意整型模拟值
Q	整型	= IN1 , 当 SEL 为“假”时 = IN2 , 当 SEL 为“真”时

说明：

二进制选择器：从两个整型输入值中选择一个。

(\* 使用“SEL”块的FBD程序例子：\*)



(\* 等价的 ST 语言程序: \*)

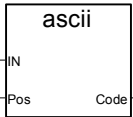
```
ProCmd := SEL (AutoMode, ManuCmd, InpCmd);
```

(\* 加工命令选择 \*)

(\* 等价的 IL 语言程序：\*)

```
LD      AutoMode
SEL      ManuCmd, InpCmd
ST      ProCmd
```

## ASCII 字符 ASCII 码



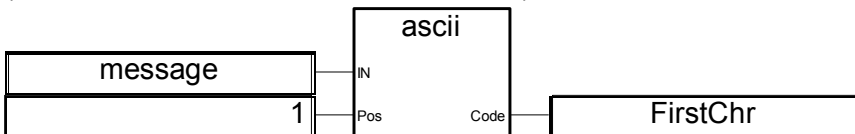
参数：

IN	信息	任意非空的串
Pos	整型	所选择字符的位置 在区间 [1.. len] 中 (len 为 IN 信息的长度)
Code	整型	所选择字符的编码 (在区间 [0 .. 255] 内) 如果 Pos 的值处于串外，则返回 0

说明：

给出在信息串中的字符的 ASCII 码

(\* 使用“ASCII”块的 FBD 程序例子：\*)



(\* 等价的 ST 语言程序：\*)

```
FirstChr := ASCII (message, 1);
```

(\* FirstChr 是 message 串的第一个字符的 Ascii 码 \*)

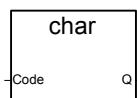
(\* 等价的 IL 语言程序：\*)

```
LD      message
```

ASCII 1

ST FirstChr

## CHAR ASCII 码字符



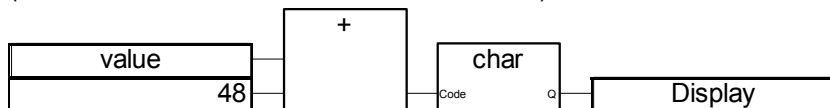
参数：

Code	整型	在区间 [0 .. 255] 内的码值
Q	信息	单字符的串
		具有 Code 给出的 ASCII 码的字符 (ASCII 码在 256 之内)

说明：

从给定的 ASCII 码求出单字符信息串。

(\* 使用“CHAR”块的 FBD 程序例子：\*)



(\* 等价的 ST 语言程序：\*)

Display := CHAR ( value + 48 );

(\* value 在区间 [0..9] 内 \*)

(\* 48 是 ' 0 ' 的 ASCII 码值 \*)

(\* 结果是范围在 ' 0 ' 至 ' 9 ' 的一个单字符串\*)

(\* 等价的 IL 语言程序 : \*)

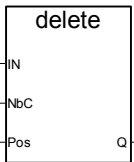
LD            value

ADD          48

CHAR

ST        Display

## DELETE 删除子串



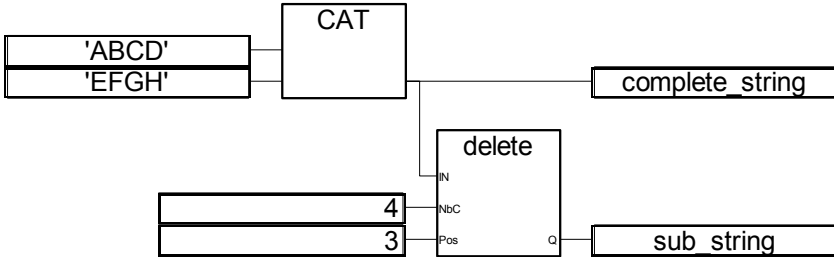
参数：

IN	信息	任意非空的信息串
NbC	整型	将删除的字符数
Pos	整型	将删除的第一个字符的位置 (串内的第一个字符的位置为 1)
Q	信息	修改后的串 Pos < 1 如果 Pos < 1, 返回空串 如果 Pos > IN 串长度, 返回初始串 如果 NbC <= 0, 返回初始串

说明：

删除部分信息串。

(\*使用“DELETE”块的FBD程序例子:\*)



(\*等价的ST语言程序:\*)

complete\_string := 'ABCD' + 'EFGH'; (\* complete\_string is 'ABCDEFGH' \*)

sub\_string := DELETE (complete\_string, 4, 3); (\* sub\_string is 'ABGH' \*)

(\*等价的IL语言程序:\*)

```

LD      'ABCD'
ADD     'EFGH'
ST      complete_string
DELETE  4,3
ST      sub_string
  
```

## FIND 查找子串



参数:

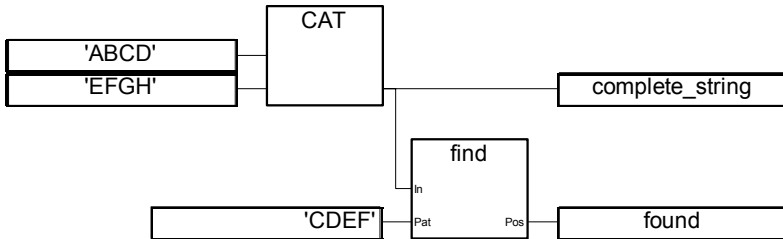
In	信息	任意信息串
Pat	信息	任意非空的串 ( 样板 )

Pos            整 型        = 0 , 如果未找到子串 Pat  
                               = 子串 Pat 第一次出现时其第一个字符  
                               的位置  
                               (可能的第一个位置是 1)  
                               此函数的结果与大小写有关

说明：

在信息串中查找子串，给出子串在串中的位置。

(\* 使用“ FIND” 块的 FBD 程序例子：\*)



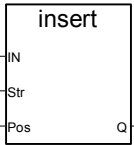
(\* 等价的 ST 语言程序: \*)

```
complete_string := 'ABCD' + 'EFGH'; (* complete_string is 'ABCDEFGH' *)
found := FIND (complete_string, 'CDEF'); (* found is 3 *)
```

(\* 等价的 IL 语言程序: \*)

```
LD      'ABCD'
ADD     'EFGH'
ST      complete_string
FIND    'CDEF'
ST      found
```

## INSERT 插入串



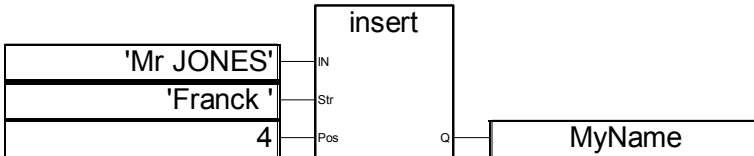
参数：

IN	信息	初始字符串
Str	信息	将插入的串
Pos	整型	插入的位置 插入到插入位置之前 (第一个有效的位置是 1)
Q	信息	修改后的串 如果 Pos ≤ 0 则返回空串 如果 Pos 大于 IN 串的长度，则返回两个串的串联体

说明：

在信息串的指定位置插入一个子串。

(\* 使用“INSERT”块的 FBD 程序例子：\*)



(\* 等价的 ST 语言程序：\*)

MyName := INSERT ('Mr JONES', 'Frank ', 4);

(\* MyName is 'Mr Frank JONES' \*)

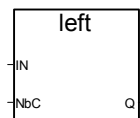
(\* 等价的 IL 语言程序: \*)

LD           'Mr JONES'

INSERT      'Frank ',4

ST      MyName

## LEFT 提取字符串左边字符



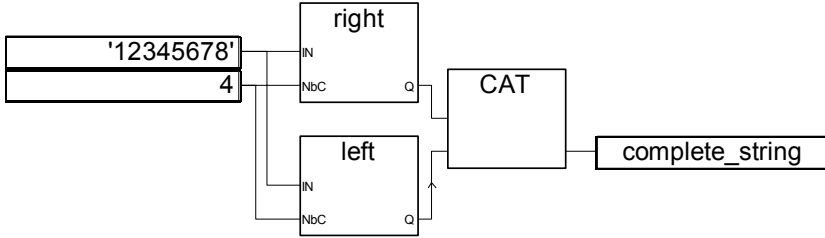
参数：

IN	信息	任意非空的串
NbC	整型	要提取的字符数 不能大于 IN 串长度
Q	信息	IN 串的左边部分 ( 长度 = NbC ) 如果 NbC <= 0 则返回空串 如果 NbC >= IN 串长度则返回完整的 IN 串

说明：

提取信息串的左边部分，提取的字符数是指定的。

(\* 使用“ LEFT” 和“ RIGHT” 块的 FBD 程序例子: \*)



(\* 等价的 ST 语言程序: \*)

```
complete_string := RIGHT ('12345678', 4) + LEFT ('12345678', 4);
```

(\* complete\_string is '56781234'

RIGHT 调用的结果为 '5678'

LEFT 调用的结果为 '1234'

\*)

(\* 等价的 IL 语言程序: 首先完成 LEFT 调用\*)

```
LD      '12345678'
```

```
LEFT    4
```

```
ST      sub_string      (* 中间结果 *)
```

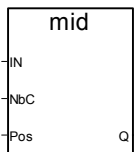
```
LD      '12345678'
```

```
RIGHT   4
```

```
ADD     sub_string
```

```
ST      complete_string
```

MID 提取字符串中间字符



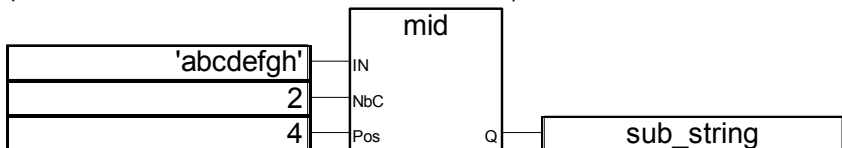
参数：

IN	信息	任意非空串
NbC	整型	要提取的字符数 不能大于 IN 串的长度
Pos	整型	子串的位置 Pos 指向子串的第一个字符 (第一个有效的位置为 1)
Q	信息	串的中间部分 (其长度 = NbC) 如果参数无效则返回空串

说明：

提取信息串的一部分，给定提取的字符数和起始位置。

(\* 使用“MID”块的 FBD 程序例子：\*)



(\* 等价的 ST 语言程序：\*)

```
sub_string := MID ('abcdefgh', 2, 4);
```

(\* sub\_string is 'de' \*)

(\* 等价的 IL 语言程序：\*)

LD            'abcdefgh'

MID          2,4

ST        sub\_string

MLEN 取得串长度



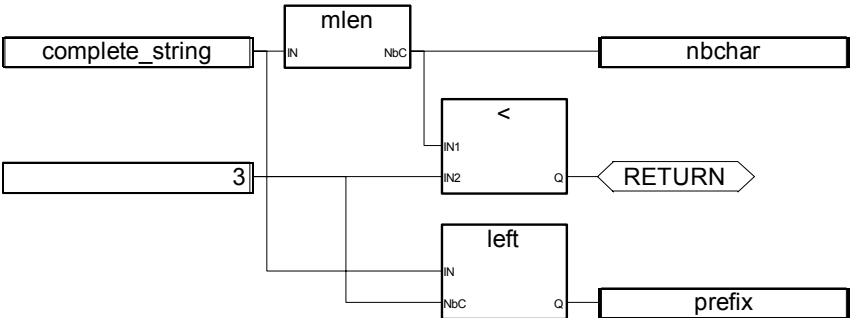
参数：

IN	信息	任意信息串
NbC	整型	IN 串的字符数

说明：

计算信息串的长度。

(\* 使用“ MLEN” 块的 FBD 程序例子：\*)



(\* 等价的 ST 语言程序：\*)

```
nbchar := MLEN (complete_string);  
If (nbchar < 3) Then Return; End_if;
```

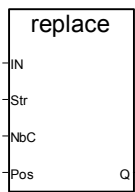
```
prefix := LEFT (complete_string, 3);
```

(\* 此程序提取字符串左侧的 3 个字符并将结果赋值给信息变量  
prefix  
如果串长度小于 3 则不进行操作 ( 返回 ) \*)

(\* 等价的 IL 语言程序 : \*)

```
LD          complete_string
MLEN
ST          nbchar
LT          3
RETC
LD          complete_string
LEFT       3
ST      prefix
```

REPLACE 替换子串



参 数 :

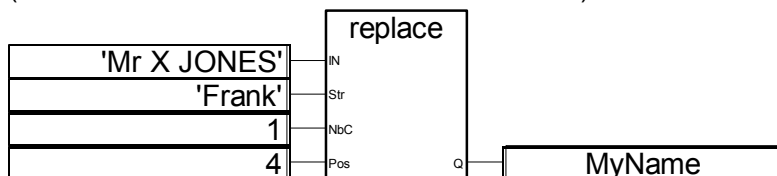
IN	信 息	任 意 字 符 串
Str	信 息	将 插 入 的 字 符 串 ( 替 换 NbC 指 定 的 字 符 )

NbC	整型	将删除的字符数
Pos	整型	要修改的第一个字符的位置 (第一个有效位置为 1)
Q	信息	修改后的字符串: - 从由 Pos 指定的位置起删除 NbC 个字符 - 再插入子串 Str 到此位置 如果 Pos <= 0 则返回空串 如果 Pos 大于 IN 串的长度则返回串联的字符串 (IN+Str) 如果 NbC <= 0 则返回初始串 IN

说明:

以一组新字符替换信息串的一部分。

(\* 使用 “ REPLACE ” 块的 FBD 程序例子: \*)



(\* 等价的 ST 语言程序: \*)

```
MyName := REPLACE ('Mr X JONES', 'Frank', 1, 4);
```

(\* MyName is 'Mr Frank JONES' \*)

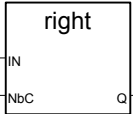
(\* 等价的 IL 语言程序: \*)

```
LD      'Mr X JONES'
```

REPLACE 'Frank',1,4

ST MyName

## RIGHT 提取字符串右边字符



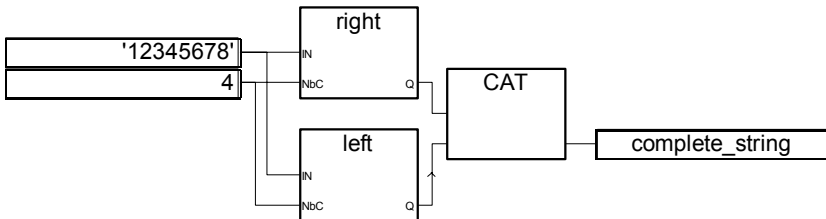
参数：

IN	信息	任意非空的串
NbC	整型	不能大于 IN 串的长度
Q	信息	串的右边部分（长度 = NbC） 如果 NbC ≤ 0 则返回空串 如果 NbC ≥ 串长度，则返回整个串

说明：

提取信息串的右边部分。提取的字符数目已给定。

(\* 使用“LEFT”和“RIGHT”块的 FBD 程序例子：\*)



(\* 等价的 ST 语言程序：\*)

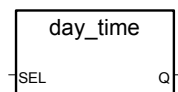
complete\_string := RIGHT ('12345678', 4) + LEFT ('12345678', 4);

```
(* complete_string is '56781234'
RIGHT 调用的结果为 '5678'
LEFT 调用的结果为 '1234'
*)
```

(\* 等价的 IL 语言程序：首先完成 LEFT 调用\*)

```
LD      '12345678'
LEFT    4
ST      sub_string      (* 中间结果 *)
LD      '12345678'
RIGHT   4
ADD     sub_string
ST      complete_string
```

## DAY\_TIME 日期时间



参数：

SEL	整型	选择输出结果
		0= 取当前日期
		1= 取当前时间
		2= 取星期值
Q	信息	以字符串表示的时间 / 日期
		如果 SEL = 0 则为 'YYYY/MM/DD'

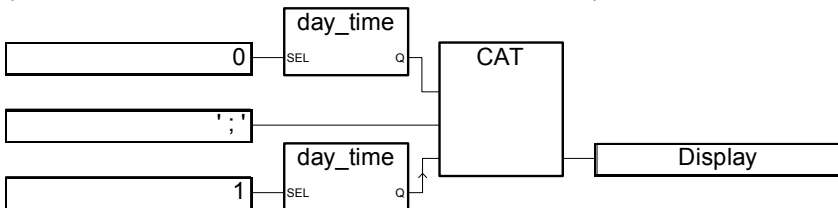
如果 SEL = 1 则为 ‘ HH:MM:SS’

如果 SEL = 2 则为 星期名 ( 例如 : ‘  
Monday’ )

说明 :

以字符串的形式给出日期和时间。

(\* 使用 “ DAY\_TIME” 块的 FBD 程序例子 : \*)



(\* 等价的 ST 语言程序 : \*)

Display := Day\_Time (0) + ';;' + Day\_Time (1);

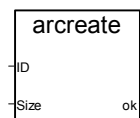
(\* Display 的文本格式为 : 'YYYY/MM/DD ; HH:MM:SS' \*)

(\* 等价的 IL 语言程序 : 首先完成 day\_time(1) 调用\*)

```

LD      1
DAY_TIME
ST      hour_str (* 中间结果 *)
LD      0
DAY_TIME
ADD     ';;'
ADD     hour_str
ST      Display
  
```

## ARCREATE 建立整型值数组



参数：

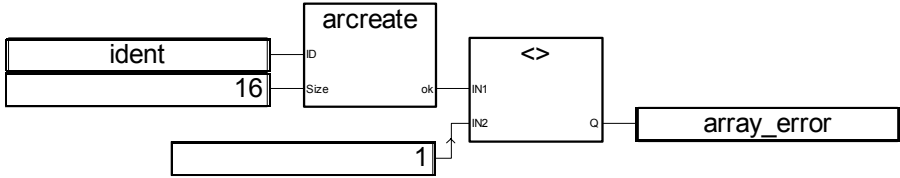
ID	整型	数组的标识符(必须在区间[0..15]内)
Size	整型	数组元素的个数
ok	整型	执行状态： 1 = 数组已成功建立 2 = 无效的数组标识符或数组已存在 3 = 无效数组大小 4 = 无足够内存

说明：

建立整型数组。

**警告：**应用程序中最多可有16个数组，数组内总包含整型模拟值。在执行动态内存分配时，当数组大小接近于可用内存的数量时，此函数有可能引起系统出错。

(\* 使用 FBD 程序建立一个整型数组 \*)



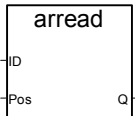
(\* 等价的 ST 语言程序 : \*)

```
array_error := (ARCREATE (ident, 16) <> 1);
```

(\* 等价的 IL 语言程序 : \*)

```
LD      ident
ARCREATE 16
NE      1
ST      array_error
```

## ARREAD 读数组元素



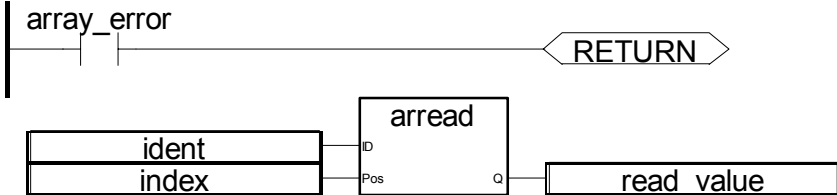
参数 :

ID	整型	数组的标识符 ( 必须在区间 [0..15] 内 )
Pos	整型	数组内元素的位置 必须在区间 [0 .. size-1]
value	整型	所读元素的值 如果参数无效则返回 0

说明：

读整型数组的元素值。

(\* 使用数组管理块的 FBD 程序例子：\*)



(\* 等价的 ST 语言程序：\*)

```

If (array_error) Then Return; End_if;
read_value := ARREAD (ident, index);
(* array_error 来自 ARCREATE 调用*)

```

(\* 等价的 IL 语言程序：\*)

```

LD      array_error
RETC

LD      ident
ARREAD  index
ST      read_value

```

## ARWRITE 写数组元素



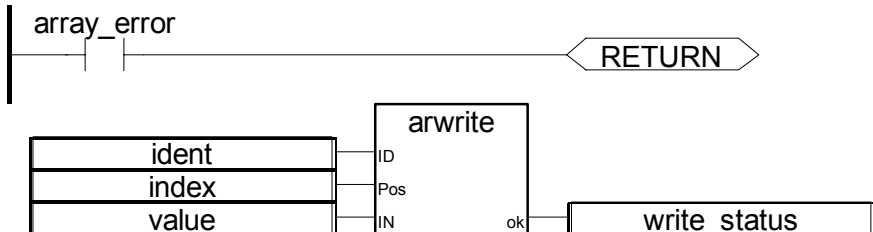
参数：

ID	整型	数组标识符（必须在区间 [0..15] 内）
Pos	整型	数组内元素的位置 必须在区间 [0 .. size-1] 内
IN	整型	元素的新值
ok	整型	执行状态： 1 = 已成功写入 2 = 无效数组标识符 3 = 无效索引

说明：

存储（写入）数值到整型数组。

(\* 使用数组管理块的 FBD 程序例子：\*)



(\* 等价的 ST 语言程序：\*)

```

If (array_error) Then Return; End_if;
write_status := ARWRITE (Ident, Index, value);
(* array_error 来自 ARCREATE 调用 *)

```

(\* 等价的 IL 语言程序：\*)

LD            array\_error

RETC

LD            ident

ARWRITE    index,value

ST          write\_status

## F\_OPEN 在读模式下打开二进制文件



### 参数：

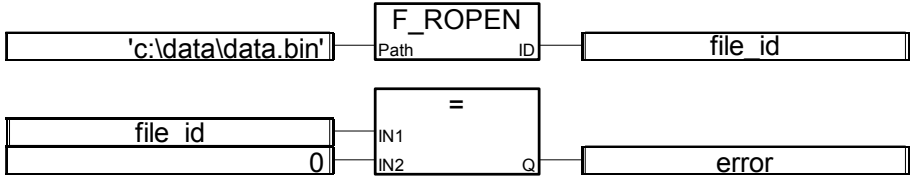
Path	信息	文件名 可以使用 / 或 \ 符号指定访问路径以增加应用程序的可移植性。/ 和 \ 符号等效。 。
ID	整型	文件号 如果出错则为 0：文件不存在。

### 说明：

以读模式打开二进制文件。与 FX\_READ 和 F\_CLOSE 一起使用。  
。

此函数不包括在 ISaGRAF 仿真器中。

(\* 使用文件管理块的 FBD 程序例子：\*)



(\* 等价的 ST 语言程序 : \*)

```

file_id := F_ROPEN('c:\data\data.bin');
error := (file_id=0);

```

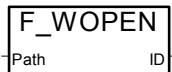
(\*等价的 IL 语言程序 : \*)

```

LD      'c:\data\data.bin'
F_ROPEN
ST      file_id
EQ      0
ST      error

```

### F\_WOPEN 在写模式下打开二进制文件



参数 :

Path	信息	文件名
		可以使用 / 或 \ 符号指定访问路径以增加应用程序的可移植性。/ 和 \ 符号等效。
ID	整型	文件号

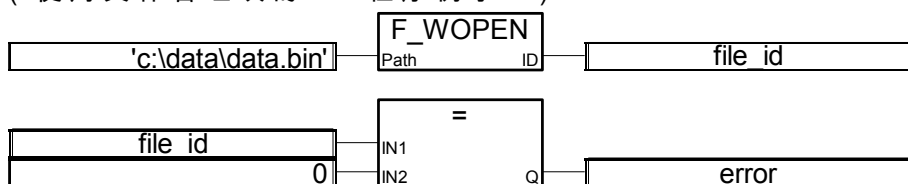
出错时为 0，如果文件已存在，它将被覆盖

说明：

以写模式打开二进制文件。与 FX\_WRITE 和 F\_CLOSE 一起使用。

此函数不包括在 ISaGRAF 仿真器中。

(\* 使用文件管理块的 FBD 程序例子：\*)



(\* 等价的 ST 语言程序: \*)

```

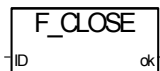
file_id := F_WOPEN('c:\data\data.bin');
error := (file_id=0);
  
```

(\* 等价的 IL 语言程序: \*)

```

LD      'c:\data\data.bin'
F_WOPEN
ST      file_id
EQ      0
ST      error
  
```

F\_CLOSE 关闭二进制文件



参 数:

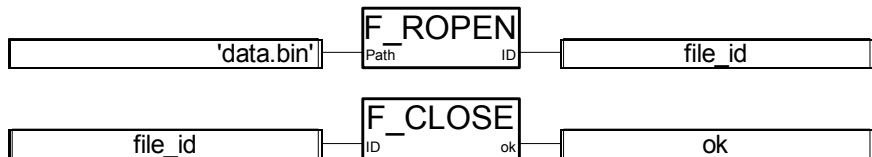
ID	整 型	文 件 号 : 由 F_ROPEN 或 F_WOPEN 返 回
ok	布 尔	返 回 状 态 :
		“ 真 ” , 如 果 文 件 成 功 关 闭
		“ 假 ” , 如 果 发 生 错 误

说 明 :

关 闭 由 F\_ROPEN 或 F\_WOPEN 函 数 打 开 的 文 件。

此 函 数 不 包 括 在 ISaGRAF 仿 真 器 中。

(\* 使用文件管理块的FBD程序例子: \*)



(\* 等价的 ST 语言程序: \*)

```
file_id := F_ROPEN('data.bin');
```

```
ok := F_CLOSE(file_id);
```

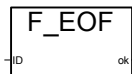
(\* 等价的 IL 语言程序: \*)

```
LD      'data.bin'
```

```
F_ROPEN
```

ST            file\_id  
F\_CLOSE            (\* file\_id 已存在于当前的 IL 结果中 \*)  
ST            ok

## F\_EOF



### 参数：

ID	整型	文件号：由 F_ROPEN 或 F_WOPEN 返回
ok	布尔	文件尾指示器

当最后一次写或读调用过程中遇到文件尾时为“真”

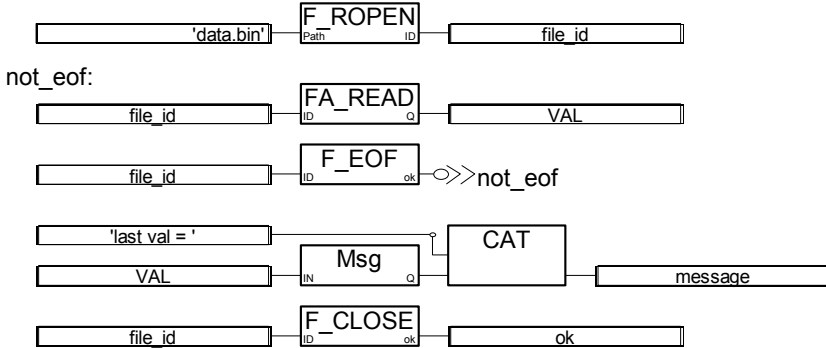
与 FM\_READ 一同使用时，如果最后一个字符不是字符串结束符，最后读出的信息可能不正确

### 说明：

测试是否到达文件尾。

此函数不包括在 ISaGRAF 仿真器中。

(\* 使用文件管理块的 FBD 程序例子：\*)



(\* 等价的 ST 语言程序 : \*)

```

file_id := F_ROPEN('data.bin');
WHILE not(F_EOF(file_id))
VAL := FA_READ(file_id);
END_WHILE;
MESSAGE := 'last val = ' + msg(VAL);
ok := F_CLOSE(file_id);

```

(\* 等价的 IL 语言程序 : \*)

```

LD      'data.bin'
F_ROPEN
ST      file_id
LD      file_id
F_EOF
JMPC    END_OF_FILE
NOT_EOF: LD      file_id
FA_READ
ST      VAL
LD      file_id

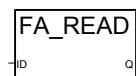
```

```

        F_EOF
        JMPNC NOT_EOF      (* 如果不是文件尾, 继续读 *)
END_OF_FILE:    LD      VAL
               信息
               ST      val_msg (* 将 VAL 变换为 信息 *)
               LD      'last val = '
               ADD     val_msg
               ST      MESSAGE
               LD      file_id
               F_CLOSE
        ST      ok

```

#### FA\_READ 从二进制文件中读取模拟值



#### 参数：

ID	整型	文件号，由 F_ROPEN 返回
Q	整型	自文件读出的整型模拟值

#### 说明：

从二进制文件中读模拟变量。与 F\_ROPEN 和 F\_CLOSE 一起使用。

此过程从先前的位置开始对文件进行顺序访问。

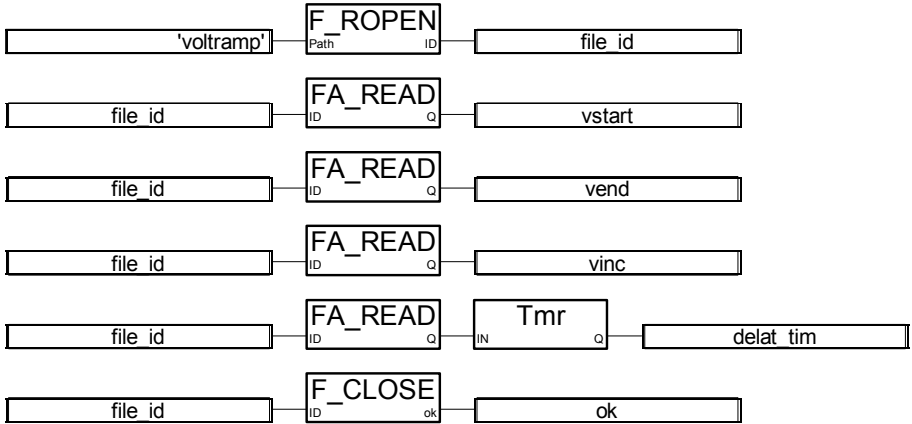
第一次调用 F\_ROPEN 读取文件的起始 4 字节。

每次调用都将读指针压入堆栈。

要检测是否到达文件尾，使用 F\_EOF.

此函数不包括在 ISaGRAF 仿真器中。

(\* 使用文件管理块的 FBD 程序例子: \*)



(\* 等价的 ST 语言程序: \*)

```

file_id := F_ROPEN('voltramp.bin');
vstart := FA_READ(file_id);
vend := FA_READ(file_id);
vinc := FA_READ(file_id);
delta_tim := tmr(FA_READ(file_id));
ok := F_CLOSE(file_id);
  
```

(\* 等价的 IL 语言程序: \*)

```

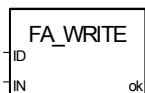
LD      'voltramp.bin'
F_ROPEN
ST      file_id
  
```

```

FA_READ                      (* 读 vstart *)
ST      vstart
LD      file_id
FA_READ                      (* 读 vend *)
ST      vend
LD      file_id
FA_READ                      (* 读 vinc *)
ST      vinc
LD      file_id
FA_READ                      (* 读 delta_tim *)
TMR                      (* 变换为定时器*)
ST      delta_tim
LD      file_id
F_CLOSE
ST      ok

```

### FA\_WRITE 向二进制文件中写入模拟值



参数：

ID	整型	文件号：由 F_WOPEN 返回
IN	整型	将被写入文件的整型模拟值
OK	布尔	执行状态：如果成功为“真”

说明：

向二进制文件内写入模拟变量。

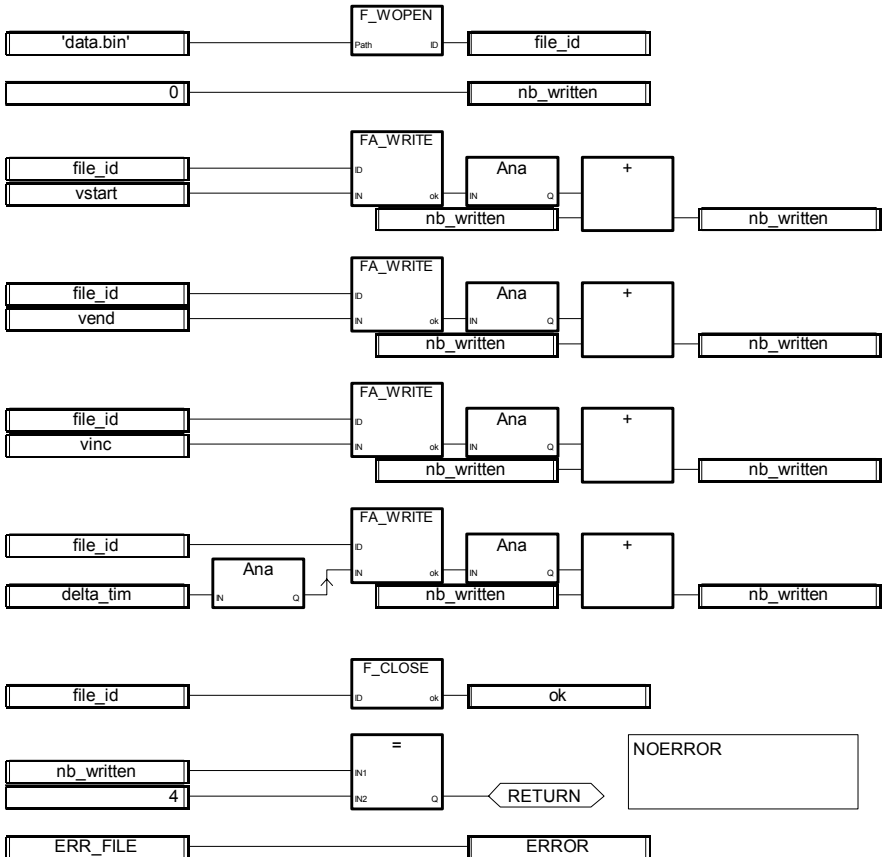
此过程从先前的位置开始对文件进行顺序访问。

第一次调用F\_WOPEN后写入文件的起始4字节。

每次写入都将写指针压入堆栈。

此函数不包括在ISaGRAF仿真器中。

(\* FBD 程序 \*)



(\* 等价的 ST 语言程序 : \*)

```
file_id := F_WOPEN('voltramp.bin');
nb_written := 0;
nb_written := nb_written + ana(FA_WRITE(file_id,vstart));
nb_written := nb_written + ana(FA_WRITE(file_id,vend));
nb_written := nb_written + ana(FA_WRITE(file_id,vinc));
nb_written := nb_written + ana(FA_WRITE(file_id,ana(delta_tim)));
ok := F_CLOSE(file_id);
IF ( nb_written <> 4) THEN
    ERROR := ERR_FILE;
END_IF;
```

(\* 等价的 IL 语言程序: \*)

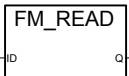
```
LD      'voltramp.bin'
F_ROPEN
ST      file_id
LD      0
ST      nb_written
LD      file_id    (* 写 vstart *)
FA_WRITE      vstart
ANA
ADD      nb_written
ST      nb_written
LD      file_id    (* 写 vend *)
FA_WRITE      vend
ANA
ADD      nb_written
ST      nb_written
```

```

LD      file_id    (* 写 vinc *)
FA_WRITE      vinc
ANA
ADD      nb_written
LD              (* 写 delta_tim *)
ANA              (* 变换为整型*)
ST      ana_delta_tim
LD      file_id
FA_WRITE      ana_delta_tim
ANA
ADD      nb_written
ST      nb_written
F_CLOSE
ST      ok
LD      nb_written
EQ      4
RETC              (* 等于 4 时返回 *)
LD      ERR_FILE    (* else 错误 *)
ST      ERROR

```

### FM\_READ 从二进制文件中读取信息



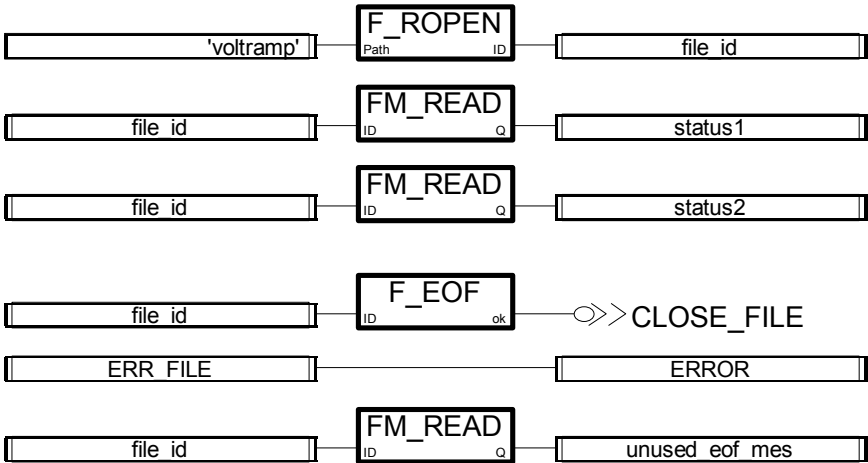
参数：

ID	整型	文件号：由 F_ROPEN 返回
Q	信息	从文件读取的信息值

说明：

自二进制文件中读取信息变量。  
与 F\_ROPEN 和 F\_CLOSE 一起使用。  
此过程从先前的位置开始对文件进行顺序访问。  
第一次 F\_ROPEN 调用读取文件的第一个字符串。  
每次调用将读指针压入堆栈。  
一个字符串以零符号 (0)，行尾 ('\n') 或回车 ('\r') 结束。  
使用 F\_EOF 检测是否到达文件尾。  
此函数不包括在 ISaGRAF 仿真器中。

(\* 使用文件管理块的 FBD 程序例子：\*)



CLOSE\_FILE:



(\* 等价的 ST 语言程序：\*)

```

file_id := F_ROPEN('voltramp.bin');
status1 := FM_READ(file_id);
status2 := FM_READ(file_id);
IF (F_EOF(file_id)) THEN
    ERROR := ERR_FILE;
    unused_eof_mes := FM_READ(file_id);
END_IF;
ok := F_CLOSE(file_id);

```

(\* 等价的 IL 语言程序 : \*)

```

        LD      'voltramp.bin'
        F_ROPEN
        ST      file_id
        FM_READ                      (* 读 status1 *)
        ST      status1
        LD      file_id
        FM_READ                      (* 读 status2 *)
        ST      status2
        LD      file_id
        F_EOF
        JMPNC   CLOSE_FILE (* 如果到达文件尾则不跳转 *)
        LD      ERR_FILE
        ST      ERROR
        LD      file_id
        FM_READ                      (* 读 unused_eof_mes *)
        ST      unused_eof_mes
CLOSE_FILE      LD      file_id

```

F\_CLOSE

ST      ok

### FM\_WRITE 向二进制文件中写入信息



参数：

ID	整型	文件号：由 F_WOPEN 返回
IN	信息	message value to be written in the file
ok	布尔	执行状态： 如果成功则为“真”

说明：

向二进制文件中写入信息变量。.

与 F\_WOPEN 和 F\_CLOSE 一起使用。

写入到文件的信息以 null 字符结束。

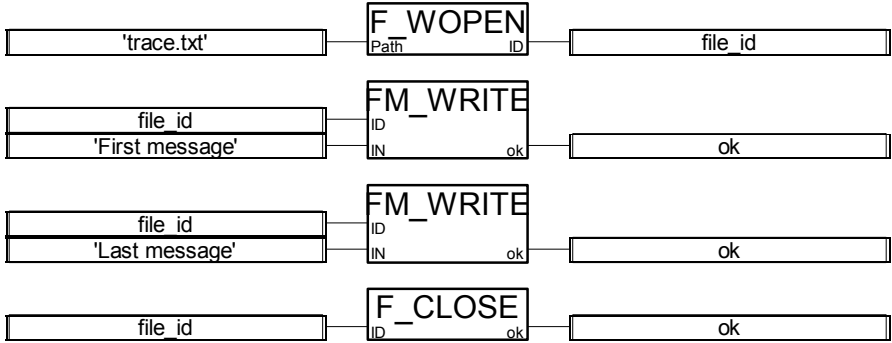
此过程从先前的位置开始对文件进行顺序访问。

第一次调用 F\_WOPEN 时向文件中写入第一个字符串。

每次调用都将写指针压入堆栈

此函数不包括在 ISaGRAF 仿真器中。

(\* 使用文件管理块的 FBD 程序例子：\*)



(\* 等价的 ST 语言程序 : \*)

```
file_id := F_WOPEN('trace.txt');
ok := FM_WRITE(file_id,'First message');
ok := FM_WRITE(file_id,'Last message');
ok := F_CLOSE(file_id);
```

(\* 等价的 IL 语言程序: \*)

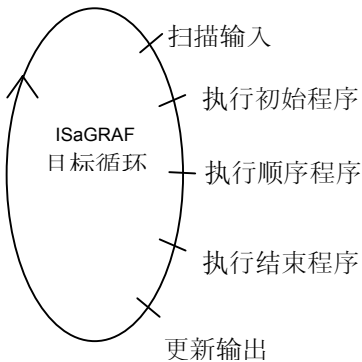
```
LD      'trace.txt'
F_WOPEN
ST      file_id
FM_WRITE      'First message' (* write first 信息 *)
ST      ok
LD      file_id
FM_WRITE      'Last message' (* write second 信息 *)
ST      ok
LD      file_id
F_CLOSE
ST      ok
```



## C. 目标用户指南

## C.1 入门

ISaGRAF 目标是在你的工业计算机或模板上运行的实时软件,ISaGRAF 应用程序按照下图所示方式循环：



目标循环由几个步骤组成：首先扫描过程的物理输入，然后按 ISaGRAF 工作台的应用程序处理应用数据，最后进行输出的更新。

- 本章的第一部分讲述了如何在特殊目标系统中启动。这些操作系统分别是 DOS, OS-9, VxWorks 和 NT。对每一个不同操作系统，首先了解：ISaGRAF 目标如何运行，然后获得其特殊属性信息。例如：上电时的目标启动，“出错”管理，一般工作状况。...

- 第二部分讲解了用户自定义的“C”函数，功能块，以及变换函数。这些方法将增强 ISaGRAF 目标的实现能力。

- 第三部分提供了 Modbus 总线与 ISaGRAF 实现的信息，描述了不同功能代码的框架格式。

- 第四部分介绍了处理“断电管理”和目标再启动的实用工具。

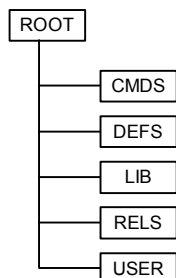
1 此指南假定用户已经熟悉 ISaGRAF 工作台。

## C.2 安装

需要大约一兆的磁盘剩余空间用于安装。Install.bat 批处理文件提供了在你的 PC 机特有平台上安装所需的文件。

示例: `a:\install a: c:\path`

将从磁盘驱动器 a: 上安装到 c: 盘的指定路径的目录，使用下列目录结构:



根目录包括一些工具和 README 自述文件

CMDS 目录放置可执行文件

DEFS 目录放置头定义文件

LIB 目录放置库文件

RELS 目录放置可重定位 ( 对象 ) 文件

USER 目录放置用户自定义的“ C” 过程，用于“ C” 函数，功能块和变换函数文件 ( 源和头文件 )

你可以在所安装的平台启动了。

## C.3 ISaGRAF DOS 目标的启动

### C.3.1 运行 ISaGRAF : ISA.EXE

在 MS-DOS 环境下，目标启动仅需运行单个程序：ISA.EXE

可以在 CMDS 目录下运行帮助命令 isa - ?

在此系统平台上操作是有限度的，例如：为了保持系统性能，不能使通讯连接过载。目标程序不妨碍运行的中断驱动子程序。

☞ 通讯连接和配置：-t 选项

ISaGRAF 目标与调试器间采用串行通讯连接。用-t 选项确定端口名。通讯接口设计为与任何机器都兼容，可使用 COM1，COM2 或 COM3 端口，这取决于 BIOS 版本。

没有默认值：如不选此项，不能与目标通讯。此时显示 7 号“出错”。

DOS 平台下 ISaGRAF 目标不能使用以太网通讯。为了特别实现，可询问供货者。

启动 ISaGRAF 之前，先设定通讯参数，用户可以任意使用他所需要的参数。使用工作台调试器时，应当保证工作台通讯参数与目标的参数匹配。

( 参见用户指南：“程序管理器” )

示例：

MODE COM1 : 9600 , N , 8 , 1

设置通讯参数为如下值：

波特率 9600

无奇偶校验

8 位数据

1 位停止位

注意：在一些 BIOS 版本，不能使用工作台的默认值 19200 波特。

ICS Triplex ISaGRAF Inc. 提供了 ISAMOD.EXE 实用工具，用于设置工作台的参数。

## ISAMOD COM1

等同于 MODE COM1 : 19200 , N , 8 , 1

☞ 从站号 : -s 选项

此选项确定目标从站号。 , 可采用除 13 ( \$0D ) 外的 1 至 255 , 从站号用于通讯链接协议。主要用于连接多个目标从站时 , 从站之间的相互区分。当使用工作台调试器时 , 确保工作台的从站参数与从站号相匹配 ( 见用户指南 : 程序管理器 ) 。

默认值 : 默认从站号为 1 ( 等同于工作台的那个 )

☞ 示例 :

isamod COM1 配置 COM1 端口 , 19200 波特 , 无奇偶校验 , 8 位数据位 , 1 停止位。

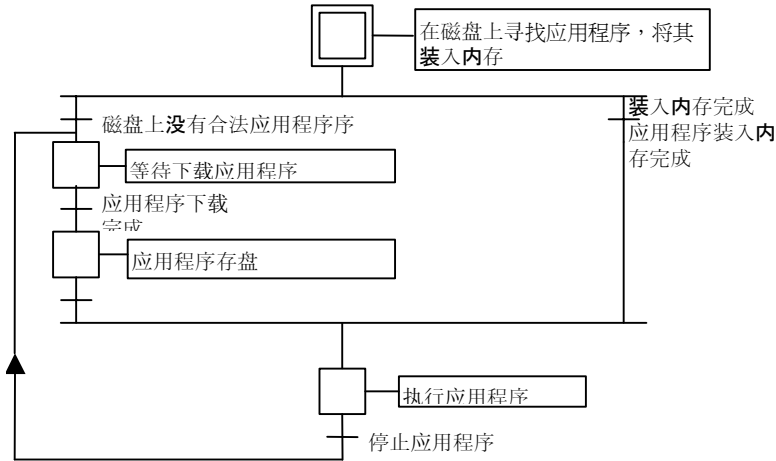
isa - t =COM1 在默认从站号 1 , 启动 ISaGRAF 目标。以 COM1 为通讯端口。

isa -s=3 -t=COM1 在 3 号从站启动 ISaGRAF 目标 , 以 COM1 为通讯端口。

## C.3.2 特性

☞ ISaGRAF 启动

当目标启动时 , 执行下列算法。



## • 定义

生成基于二进制数据的应用程序代码，从工作台下载，然后在目标执行。还可以添加符号表。

在工作台产生和下载 ASCII 数码的应用程序符号表。此表链接符号对象和内部目标对象。除非用户使用特殊的符号管理，在目标不需要此表。关于符号表的进一步信息，请参阅用户指南中“ 先进编程技术” 一章。

## • 应用程序备份

当新的应用程序从工作台调试器下载到目标，应用代码保存在目标当前目录，使用文件名：

ISAx1 ISaGRAF 应用代码备份文件 (x 是从站号)

如果事先下载了应用符号表,它用下列文件名保存在目标的当前目录:

ISAx6 ISaGRAF 应用符号备份文件 (x 是从站号)

当 ISaGRAF 启动后，在当前目录中找寻应用程序代码和符号文件，并将其装入内存。

如果没有合法的符号表文件，符号就不装入内存，目标开始运行应用程序代码

如果没有合法的应用程序代码，目标就等待下载应用程序。

为了在上电时，不使用调试器链接，在目标启动特殊应用程序。将这些文件从工作台所在的同一台 PC 的同一磁盘拷贝到目标当前目录。或者使用软盘，如果目标机器没有磁盘，就用虚拟盘。

如果 ISaGRAF 工作台安装在标准的 \ISAWIN 目录：

MYPROJ 项目的应用程序代码文件是：

\ISAWIN\APL\MYPROJ\appli.x8m

MYPROJ 项目的应用程序符号表文件是：

\ISAWIN\APL\MYPROJ\appli.tst

示例：

从 isa.exe 所在目录，输入下列命令：

copy \ISAWIN\APL\MYPROJ\appli.x8misa11

找到 isa.exe，执行 ‘ myproj’ 应用程序。

所有命令可以组成批处理文件，从工作台“ 工具” 菜单启动 (见用户指南“ 程序管理” )。

## “ 出错” 管理和输出信息

ISaGRAF 目标软件集成了故障监测管理。在附录可以找到“ 警告出错” 列表以及解释。

“ 出错” 监测过程如下：

- 将错误名和参数送到 ISaGRAF“ 出错” 程序
- 如果在工作台的“ 制作” 选项中将“ 出错监测” 标志置位，“ 出错” 将被处理。否则，信息将丢失，出错管理结束。

当处理时：

- 在默认的 stdout 输出显示出错号 ( 十进制 ) 和参数 ( 十六进制 ) 。

- 为了今后能够检索处理，将出错号和参数一起推入环型 FIFO 寄存器。在工作台的“制作”选项设置此寄存器的尺寸大小。当一个新的出错到达时，最先出现的那个将丢失。
- 当程序运行时，使用系统调用；或者从调试器可以获得“出错”信息。  
(见用户指南)

当调试器监测到一个“出错”时，出错信息的描述出现在“出错”窗口。取决于应用状况（运行/停止），调试器可以显示对象的名称（变量或程序），在何处出现的错误，或者将“出错参数”（十进制）放入[X]内，表示同一出错号下的不同意义。

当目标启动时，如果出现错误，欢迎信息和出错号将显示在默认的 stdout 窗口。如果不希望显示在标准输出通道，转向命令为

```
isa -t=COM1 -s=1 >NUL
```

### ≡ **系统时钟**

ISaGRAF 目标被设计可以运行在任何系统，用于循环同步和计时器变量更新的时基标准节拍为 55 毫秒。

这样一来，不可能有比 55ms 更精确的计时变量了。同样，特定循环周期小于 55MS 或与零的差值将产生循环周期溢出错误（出错 62），没有脉冲触发，循环将停止。

不修改系统节拍的好处是：任何现存的应用程序，或者“C”函数，功能块集成到应用程序中时，不会发生干扰 ISaGRAF 的执行。

如果需要更高的精度，对此询问你的供货者。

### ≡ **离开键**

当使用台式 PC,在非工业条件下测试应用程序时，用户可能想停止

ISaGRAF: 按组合键可以防止意外的停止，键的顺序是:

Shift +ctrl+alt

当然，当按下一个键后，不想让工业应用停止，应当使此组合键失能。这种快速离开的危险的副效应是，IO 板的接口还没有关闭。停止 ISaGRAF 目标的方法是：

- 从调试器停止运行应用程序，（ 将关闭 I/O 板 ）
- 从键盘停止 ISaGRAF 目标。

### **应用程序字节量**

ISaGRAF MS - DOS 目标被设计成英特尔实时模式，最大数据量为 64K。这样，从工作台下载的应用代码不能超过此数。在很少的情况下，内部的结构分派可以超出此极限，当下载后可引起冲突。整个可能的内存限定在不久的将来是 640K 常规内存。

如果你的应用需要更多的内存，询问你的供货者。

## C.4 ISaGRAF OS9 目标启动

首先使用任意传送工具，将文件（至少将执行文件从 CMDS 目录）传送到 OS-9 目标。

然后，为了启动，在 OS-9 系统 CMDS 目录下运行帮助命令：

```
isa -?  
isaker -?  
isatst -?  
Isanet -?
```

### C.4.1 运行 ISaGRAF 单任务: isa

ISaGRAF 目标能作为单任务运行。但是如此配置,操作可能有限度。为了保证系统的良好特性,不要使通讯过载。在 OS9 多任务系统,只要从站号和端口号不同，不同的单任务目标可以运行在同一 CPU 上。单任务实现主要被设计用在低性能硬件平台,例如：低价模板或者 MS-DOS PC' s 或者与新平台连接时的试验。因此,应当优先采用多任务目标实现。ISaGRAF 单任务不妨碍后台处理或中断驱动程序。

#### ☐ **通讯链接和配置: -t 选项**

ISaGRAF 单任务与调试器间采用串行通讯连接。用-t 选项设定描述名。

没有默认值：如不选此项，不能与目标通讯。此时显示 7 号“出错”。

单任务实现的 ISaGRAF，目标不能使用以太网通讯。

串接设备采用二进制数据传输模式（没有控制字，没有 XON/XOFF）。

启动 ISaGRAF 之前，先设定其它通讯参数，这样一来，用户可以任意使用他所需要的参数。使用工作台调试器时，应当保证工作台通讯参数与目标的参数匹配（参见用户指南：“程序管理器”）

示例:

xmode /t0 波特=19200

设置通讯波特率 19200 , on /t0 设备

### ≡ **从站号: -s 选项**

此选项确定目标从站号,可采用除 13 ( \$0D ) 外的 1 至 255,从站号用于通讯链接协议。用于连接多个目标从站时,从站之间的相互区分。当使用工作台调试器时,确保工作台的从站参数与存在的目标从站相匹配 ( 见用户指南: 程序管理 )

默认值: 默认从站号为 1 ( 与工作台的相同 )

### ≡ **示例:**

isa -t=/t0 启动 ISaGRAF 单任务目标,默认从站号(1),使用 /t0 作为通讯端口。

isa -s=3 -t=/t1 启动 ISaGRAF 单任务目标从站号 3 ,使用 /t1 作为通讯端口。

isa -t=/t0 &

isa -s=3 -t=/t1 启动两个 ISaGRAF 单任务目标: 一个默认从站号(1),使用 /t0 作为通讯端口。另一个从站号 3 ,使用 /t1 作为通讯端口。

## **C.4.2 运行 ISaGRAF 多任务: isaker, isatst, isanet**

运行多任务为了改善目标内核和通讯连接的响应时间,目标从执行应用 ( isaker 内核任务 ) 分出两个单独的通讯任务 ( isatst,或 isanet 通讯 )。这样的结构更加灵活,可以使用户与同一个内核任务建立多个通讯任务连接,或者多至 4 个内核使用一任务个通讯任务。

可以容易的实现过程可视化系统和工作台调试器连接于同一个应用程序,或者通过同一物理端口与 4 个不同应用单链接。

内核与通讯任务是互相独立和分别不同的分支。仅要求内核任务首先启动,初始化系统环境。然后,通讯任务可以与它连接。

ISaGRAF 多任务目标不妨碍后台运行处理或中断驱动程序

#### C.4.2.1 运行内核任务: isaker

##### ☞ **从站号: -s 选项**

此选项确定目标内核从站号,可采用除 13 ( \$0D)外的 1 至 255,从站号用于通讯链接协议。通讯任务与内核链接,用于连接多个目标从站时,从站之间的相互区分。

默认值: 默认从站号为 1 (与工作台的相同)

#### C.4.2.2 运行串行通讯任务: isatst

##### ☞ **通讯链接和配置: -t Option**

目标通讯任务 isatst 与调试器间采用串行通讯连接。用-t 选项设定描述名。

没有默认值: 如不选此项,不能与目标通讯。此时显示 7 号“出错”。

isatst 任务实现,不能使用以太网通讯。

串接设备采用二进制数据传输模式(没有控制字,没有 XON/XOFF)。启动 ISaGRAF 之前,先设定其他通讯参数,这样,用户可以任意使用他所需要的参数。使用工作台调试器时,应当保证工作台通讯参数与目标的相匹配(参见用户指南:“程序管理”)

## 示例

xmode /t0 baud=19200

设置通讯波特率 19200 baud on /t0 设备

### ⇒ **从站号: -s 选项**

此选项确定通讯任务链接的目标内核从站号。 可采用除 13 ( \$0D)外的从 1 至 255。从站号用于通讯链接协议。用于连接多个目标从站时，从站之间的相互区分。当使用工作台调试器时，确保工作台的从站参数与存在的目标相匹配（内核和通讯任务）（见用户指南：程序管理）

默认值： 默认从站号为 1 （与工作台的相同）

### ⇒ **通讯任务逻辑号: -c 选项**

此选项确定通讯任务逻辑号。 用于同时管理多个通讯任务。可采用从 1 至 255，每个通讯任务的逻辑号必须不同。

默认值： 使用 -s 选项。默认值可以兼容过去的 ISaGRAF (3.0)版本

## C.4.2.3 运行以太网通讯任务: isanet

### ⇒ **通讯链接和配置: -t 选项**

目标通讯任务 isanet 用于标准以太网与调试器的通讯。 用 -t 选项确定通讯端口号。

没有默认值：如不选此项，不能与目标通讯。此时显示 7 号“ 出错” 。当使用工作台调试器时，确保工作台的通讯参数与目标相匹配。（见用户指南：程序管理）

对于 ISaGRAF, OS-9 目标是服务器, 调试器是连接到特定端口号的客户。当开始以太网第一次调试前, 你应当保证 OS-9 以太网设备正确地配置。可以发送一个 ping 到 OS-9 系统。

### ☐ **从站号: -s 选项**

此选项确定通讯任务链接的目标内核从站号, 可采用除 13 ( \$0D ) 以外的 1 至 255, 从站号用于通讯链接协议。用于连接多个目标从站时, 从站之间的相互区分。当使用工作台调试器时, 确保工作台的从站参数与存在的目标从站相匹配(内核和通讯任务) ( 见用户指南: 程序管理器 )

默认值: 默认从站号为 1 ( 与工作台的相同 )

### ☐ **通讯任务逻辑号: -c 选项**

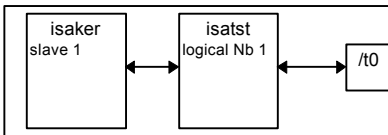
此选项确定通讯任务逻辑号。用于同时管理多个通讯任务。可采用从 1 至 255, 每个通讯任务的逻辑号必须不同。

默认值: 使用 -s 选项。默认值可以兼容过去的 ISaGRAF (3.0)版本

#### C.4.2.4 示例:

isaker &

isatst -t=/t0



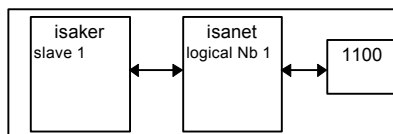
启动:

一个 ISaGRAF 内核任务, 其默认从站号为 (1)。

一个 ISaGRAF 串行通讯任务, 在 /t0 com 端口, 与默认从站号(1)链接, 使用默认逻辑号(最后确定的从站号 = 默认 = 1)。

isaker &

isanet -t=1100



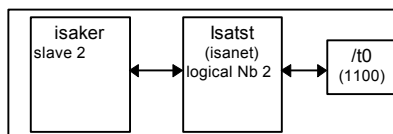
启动:

一个 ISaGRAF 内核任务，其默认从站号为 (1)。

一个 ISaGRAF 以太网通讯任务，在 端口号 1100, 与默认从站号(1)链接,使用默认逻辑号(最后确定的从站号 = 默认 = 1)。

isaker -s=2 &

isatst -t=/t0 -s=2 (分别 isanet -t=1100 -s=2)



启动:

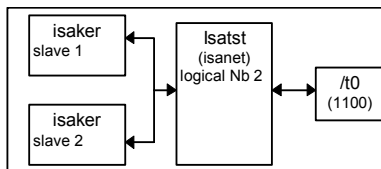
一个 ISaGRAF 内核任务，其从站号为 (2)。

一个 ISaGRAF 串行 (以太网) 通讯任务，在 /t0 com 端口 (端口号 1100)，与从站号(2)链接,使用默认逻辑号(最后确定的从站号 = 2)。

Isaker -s=1 &

isaker -s=2 &

isatst -t=/t0 -s=1 -s=2 (分别 isanet -t=1100 -s=1 -s=2)



启动:

一个 ISaGRAF 内核任务，其从站号为 (1)。

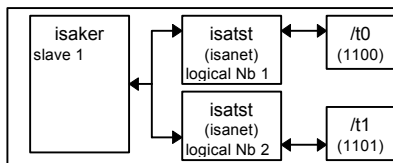
一个 ISaGRAF 内核任务，其从站号为 (2)

一个 ISaGRAF 串行 (以太网) 通讯任务, 在/t0 com 端口 (端口号 1100), 与从站号(2)和 (1)链接,使用默认逻辑号(最后确定的从站号 = 2)。

Isaker -s=1 &

isatst -t=/t0 -s=1 -c=1 & (分别 isanet -t=1100 -s=1 -c=1 &)

isatst -t=/t1 -s=1 -c=2 (分别 isanet -t=1101 -s=1 -c=2)



启动:

一个 ISaGRAF 内核任务，其从站号为 (1)。

一个 ISaGRAF 串行 (以太网) 通讯任务, 在/t0 com 端口 (端口号 1100), 与从站号(1)链接,使用逻辑号 (1)。

一个 ISaGRAF 串行 (以太网) 通讯任务, 在/t0 com 端口 (端口号 1101), 与从站号(1)链接,使用逻辑号(2)。

注意:

可以混合使用串接和以太网通讯任务。

### C.4.3 特性

#### ☐ 通讯链接

由于 OS-9 串行字符管理器非常灵活，几乎任何支持 Microware 的双向物理设备都可以使用：

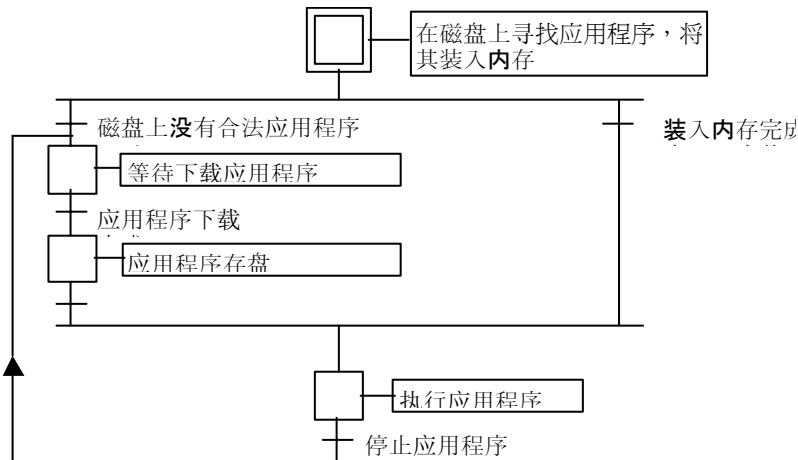
示例：

串行链接可以通过网络路径至另一个 CPU 的物理端口。使用 -t 选项的示例如下：-t=/nr/MASTER/t0

通讯链接到在 ramnet 网络上称为 MASTER 的 CPU，使用的物理端口是 /t0。

#### ☐ ISaGRAF 启动

当目标启动时，执行下列算法。



- 定义

生成基于二进制数据的应用程序代码，从工作台下载，然后在目标执行。还可以添加符号表。

在工作台产生和下载 ASCII 数码的应用程序符号表。此表链接符号对象和内部目标对象。除非用户使用特殊的符号管理，目标不需要此表。关于符号表的进一步信息，请参阅用户指南中“先进编程技术”一章。

- ISaGRAF OS-9 对象和多应用程序

每个 ISaGRAF 公共对象名以 ISAxn 开始，x 是内核从站号，n 有是特殊意义的号，除了 ISAy3，y 是多任务实现的通讯任务逻辑号。只要具有不同的通讯任务逻辑号和分别不同的从站号，不同的应用(内核和通讯任务)可以同时运行于一个 CPU。尽管如此，当运行不同的应用时，用户必须注意：一些应用对象共享存取这些 I/O 板。例如：不同的应用(内核)分别使用物理板，而不是通过 I/O 驱动实现同类 I/O 服务器或信号发生器。

OS-9 对象名:

磁盘文件:

ISAx1	ISaGRAF 应用程序代码备份文件
ISAx6	ISaGRAF 应用程序符号备份文件

内存模块:

ISAx0	ISaGRAF 内核系统数据
ISAx1	ISaGRAF 应用程序代码
ISAx2	ISaGRAF 内核实时数据库
ISAy3	ISaGRAF 通讯数据交换寄存器
ISAx4	ISaGRAF 在线修改 1 应用程序代码
ISAx5	ISaGRAF 在线修改 2 应用程序代码
ISAx6	ISaGRAF 应用程序符号

对此，用户必须小心不要使用同一对象名称。

- 应用程序备份

当新应用程序从工作台调试器下载到目标，应用程序代码被存在目标当前目录下，使用文件名：

ISAx1 ISaGRAF 应用程序代码备份文件 (x 是从站号)

如果事先下载了应用符号表,它用下列文件名保存在目标的当前目录:

ISAx6 ISaGRAF 应用程序符号备份文件 (x 是从站号)

当 ISaGRAF 启动后，在当前目录中找寻应用程序代码和符号文件，并将其装入内存，作为有同一名称的数据模块。

如果在内存中没有合法的符号文件，符号就不装入内存，目标开始运行应用程序代码。

如果在内存中没有合法的应用程序代码，目标就等待下载应用程序。

为了在上电时，不使用调试器链接，启动带有特殊应用程序的目标。

- 第一种方法是：使用任何一种文件传送工具，从工作台所在的主 PC 机，直接将这此文件拷贝到目标当前目录磁盘。可以使用工作台的“工具”菜单去方便此操作(见用户指南“管理程序”)。
- 第二种方法是：从工作台所在的主 PC 机，使用工具将这此文件的应用程序代码（必要时，还有应用符号表）存储到非易失的内存中，( PROM 或 EPROM)。

当系统上电时，如有需要（例如：快速存取，或断点管理），你可以使用工具，从 PROM 内装载应用程序代码(必要时：还有应用符号表)到 RAM。作为 ISAx1（必要时，ISAx6）内存数据模块。

**警告：**

如果应用程序代码模块不能被“写”访问，ISaGRAF 调试器的断点管理就不能正确的运行。当你事先进行了全面测试，将不会发生此问题。

在 PC 主站, 如果 ISaGRAF 工作台安装在标准的 \ISAWIN 目录下:

MYPROJ 项目的应用程序代码是:

\ISAWIN\APL\MYPROJ\appli.x6m (对应目标中的 isax1)。

MYPROJ 项目的应用程序符号表是：

\ISAWIN\APL\MYPROJ\appli.tst (对应目标中的 isax6)。

## “出错”管理和输出信息

ISaGRAF 目标软件集成了故障监测管理。在附录可以找到“警告出错”列表以及解释。

出错监测过程如下:

- 将错误名和参数送到 ISaGRAF “出错”程序
- 如果在工作台的“制作”选项中将“出错监测”标志置位，“出错”将被处理。否则，信息将丢失，“出错”管理结束。

当处理时：

- 在默认的 stdout 输出显示出错号（十进制）和参数（十六进制）

- 为了今后能够检索处理，将出错号和参数一起推入环型 FIFO 寄存器。在工作台的“制作”选项设置此寄存器的尺寸大小。当一个新的“出错”到达时，最先出现的那个将丢失。
- 当程序运行时，使用系统调用；或者从调试器可以获取“出错”信息。（见用户指南）

当调试器监测到一个出错时，“出错”信息的描述出现在出错窗口。取决于应用场合（运行/停止），调试器可以显示对象的名称（变量或程序），在何处出现的错误，或者将出错参数（十进制）放入[X]内表示每个同一出错号的不同意义。

当目标启动时，如果出现错误，欢迎信息和出错号将显示在默认的 stdout 窗口。如果不希望显示在标准输出通道，转向命令为

```
isa -t=COM1 -s=1>NUL
```

### ⇒ **循环周期, 任务工况, 任务优先级**

- ISaGRAF 循环结束时, 启动下一个之前, 执行下列算法:

如果循环时间特别设定, (从工作台: 见用户指南: 程序管理) CPU 将放弃一段保留的时间段 (特定循环时间 - 当前使用的)。如果此保留时间为负值, 将出现溢出。CPU 将放弃一个节拍, 以便强制同步准时。

如果没有特设循环时间, 或者保留时间小于或等于 1 个节拍, 或等于 0, CPU 将放弃一个节拍, 强制同步准时。

目标计时精度对应于 OS-9 系统节拍。

特定循环时间一般用于触发循环, 或者使 CPU 顾及运行在 OS-9 系统的其它任务。

- 当没有数据到达通过通讯链接，通讯任务处于休眠状态。当需要时，此任务与内核任务一起通过问/答通讯协议获取运行的应用程序信息。通讯任务向内核任务提出问题。循环结束时，（为了同步应用映像）内核应答通讯任务。

ISaGRAF 任务不修改它们的优先权，用户可以按照上述 ISaGRAF 任务工况和整个应用的要求，调整这些优先权。

例如：为了保证 ISaGRAF 不被低级任务强占优先权，OS9 的任务管理器的 MIN\_AGE 和 MAX\_AGE, 参数可被修改。

## 终端模式

目标串行通讯协议承认连续的 3 个回车字符(\$0D)，然后启动 OS9 shell 任务,如果许可的话，在串接设备上，可以使用 ISaGRAF 目标串接，在任一终端得到 OS-9shell 提示符号。

### 示例:

从主 PC:

- 关闭 ISaGRAF 调试器。
- 启动 Windows 终端仿真器(访问组)，使用正确的参数。
- 击三下 return 回车键

你现在已经在 OS-9 Shell 登录

- 输入 logout，退出终端模式

### 警告:

总是使用 logout 退出终端模式, 否则，下一次与工作台连接将不成功。

## C.5 启动 ISaGRAF VxWorks 目标

为了启动 ISaGRAF 目标,在 VxWorks 系统需要执行一些命令。以便于设置配置环境和最终作用于 ISaGRAF 目标。从脚本文件开始这些命令。以下章节描述它们。

### C.5.1 系统资源管理: isassr.o

在任何 ISaGRAF 目标配置中,都需要此模块。而且必须是首先下载到目标的模块。它可以实现多目标运行的系统资源管理。

### C.5.2 isa.o, isakerse.o 和 isakeret.o 的公共属性

为了运行 ISaGRAF,可以下载 这些模块之一。

isa.o:            允许启动单任务目标(仅串行通讯连接)

Isakerse.o:    。 允许启动多任务目标 ( 仅串行通讯连接)

Isakeret.o:     允许启动多任务目标(串行或者/和以太网连接)

下一章描述其细节。

#### ⇨ 串行通讯连接配置

ISaGRAF 目标基本使用串行与调试器通讯连接,对于 ISaGRAF 目标的串行连接设备,一般不需要进行配置。用户可以任意使用他所需要的参数。尽管如此,还需要一个二进制数据传输模块(RAW 模块)。对此提供了一个 *ISAMOD* ()子程序

uchar ISAMOD

(

```
char *desc, /* 串行设备名称 */
uint32 baudrate /* 波特率 */
)
```

#### 描述

特殊波特率,二进制数据的串行连接设备的配置

#### 返回值

如果成功=0,如果出错= BAD\_RET

使用工作台调试器时,应当保证工作台通讯参数与目标一致。

(参见用户指南: 程序管理)

#### 系统时钟频率

全局变量 CLKRATE (uint32)用于初始化 VxWorks 系统时钟频率,此时可以使用:

```
CLKRATE = sysClkRateGet ()
```

默认值 CLKRATE 是 60Hz。

### C.5.3 运行 ISaGRAF 单任务: isa.o

ISaGRAF 目标允许作为单任务运行。 如此配置,操作可能是处于临界态。推荐: 为了保证系统的良好特性,不要使通讯过载。在 VxWorks 多任务系统,只要从站号和端口号不同,不同的 ISaGRAF 单任务目标可以运行在同一 CPU 上。

单任务实现主要被设计用在低性能硬件平台,例如: 低价模板或者 MS-DOS PC' s 或者与新平台连接时的样机。因此,应当优先采用多任务目标实现。单任务不妨碍后台处理或中断程序驱动

## ☐ 从站登录

ISaGRAF 目标以其从站号为标识特征,可以是除了 13 ( \$0D)之外的从 1 至 255 号码。从站号适用于全部通讯协议。当多个从站目标运行时,需要从中互相区别。因此,启动 ISaGRAF 目标之前,先要登录。对此提供了 *isa\_register\_slave()*程序

```
uchar isa_register_slave
(
    uchar slave /* 从站号 */
)
```

### 描述

新加一个从站登录到多任务目标管理系统。

### 返回值

如果成功=0,如果出错= BAD\_RET

## ☐ 应用备份文件存储单元

全局变量 TSK\_FUNIT (char \*) 可被初始化为,包含应用文件备份路径的字符串。ISaGRAF 目标使用 fopen, fread, fwrite, fclose , 用于应用文件备份的程序。其默认值是一个空串(""),表示没有存储单元。

### 示例:

```
TSK_FUNIT = "host name:/C:/ISaGRAF/target/apl/"
```

设定 在 *host name* PC 的 C 盘根目录下 ISaGRAF\target\apl\ ,作为应用文件的备份目录。注意:不要忘记最后的斜杠,否则,备份到 ISaGRAF\目标\目录下,以 APL 为前缀的文件名下。

如果需要, 对每个目标,每次作用前, 变量可以被设到不同的路径单元。在应用程序备份一章,可以找到更多的关于备份应用程序文件,和相关特性的详细信息。

### ☐ **系统控制结束**

变量 TSK\_NBTCKSCHED(uint 32)能被设定:当 ISaGRAF 目标循环结束时,延时几个节拍。默认值为 0(与任务安排同优先级)。每次作用前,每个目标启动,如果需要,此变量可被设为不同的值。在任务特性和任务优先权一章,可以找到关于特殊属性和循环周期的更详细信息。

### ☐ *ISaGRAF 目标作用*

只要配置环境设定完成,最后一步是: 作用 ISaGRAF 目标: isa\_main

```
uchar isa_main
(
    uchar slave, /* 从站编号*/
    char *com     /* 串行设备名称 */
)
```

### 描述

启动 ISaGRAF 目标任务

### 返回值

如果出错,返回一个非 0 值

从站号与在从站登录一章所讨论的相同。只要具有不同的从站号和端口号，可以启动多于一个目标。使用工作台调试器时,应当保证工作台通讯参数与存在的目标一致。(参见用户指南:管理程序)

### 示例

此例显示了：用/tyCo/1 设备串接的从站 1 号 ISaGRAF 单任务目标如何启动。目标安装的目录是当前的主目录。

装入 isassr.o 模块

```
ld < RELS/isassr.o
```

装入 isa.o 模块

```
ld < CMDS/isa.o
```

串口通讯配置

```
ISAMOD ("/tyCo/1", 19200)
```

系统时钟频率

```
CLKRATE = sysClkRateGet ()
```

从站登录

```
isa_register_slave (1)
```

文件存储单元 (采用默认值，可以跳过)

```
TSK_FUNIT = ""
```

循环结束控制 (采用默认值，可以跳过)

```
TSK_NBTCKSCHED = 0
```

ISaGRAF 目标作用

sp (isa\_main, 1, "/tyCo/1")

### C.5.4 运行 ISaGRAF 多任务: isakerse.o 和 isakeret.o

运行多任务是为了改善目标内核和通讯连接的响应时间,目标从应用程序执行(内核任务)中分出单独的通讯任务。这样的结构更加灵活,可以使用户与同一个内核任务建立多个通讯任务连接,或者多至 4 个内核使用一个通讯任务。可以容易的实现“过程可视化”系统和工作台“调试器”连接于同一个应用程序;或者通过同一个物理端口,与多至 4 个不同的应用程序单链接。

内核与通讯任务是互相独立和分别不同的作用。仅要求内核任务首先启动,初始化系统环境。然后,通讯任务可以与它连接。

多任务目标不妨碍后台运行处理或中断驱动程序。

取决于通讯硬件的能力,可以提供两种模式

— 内核与串行连接: isakerse.o

此模式可以启动内核任务和串行通讯任务。

— 内核与串行或/和以太网连接: isakeret.o

此模式允许启动内核任务和串行任务或/与以太网连接。

当启动 ISaGRAF 通讯任务: tst\_main\_ex(见下式)

对于 isakerse.o 和 isakeret.o 模块启动 ISaGRAF 的方法相同,启动 isakeret.o 通讯任务时,可以定义,不是一个串行设备名称,就是一个以太网端口号,作为通讯设备名参数。

对于 ISaGRAF 系统，VxWorks 目标是服务器,连接到特定端口号的调试器为客户。

## ≡ 内核登录

ISaGRAF 内核以其从站号为特征识别,可以是除了 13 ( \$0D)之外的从 1 至 255 号码。从站号适用于全部通讯协议，由通讯任务与内核链接。当多个从站目标运行时,需要从中互相区别。因此,启动 ISaGRAF 目标之前,先要登录。对此提供了 *isa\_register\_slave()*程序

```
uchar isa_register_slave  
(  
    uchar slave    /* 从站号 */  
)
```

### 描述

加一个新内核从站登录到多目标管理系统。

### 返回值

如果成功=0,如果出错 =BAD\_RET

## ≡ 通讯任务登录

ISaGRAF 通讯任务由它的逻辑号识别,同时可以管理多于一个通讯任务。可以取从 1 到 255，但是每个通讯任务的编号应当不同。启动 ISaGRAF 通讯任务之前，应当先登录它们。对此，提供 *isa\_register\_com()*程序

```
uchar isa_register_com  
(
```

```
uchar com_id /* com. 任务标识 */  
)
```

### 描述

加一个新通讯任务登录到多目标管理系统。

### 返回值

如果成功=0,如果出错=BAD\_RET

## ≡ **应用文件备份存储单元**

全局变量 TSK\_FUNIT(char \*) 可被初始化为,包含应用文件备份路径的字符串。ISaGRAF 目标使用标准文件管理程序: fopen, fread, fwrite, fclose, 用于应用文件备份。其默认值是一个空串(""), 确定没有存储单元。

### 示例:

```
TSK_FUNIT = "host name:/C:/ISaGRAF/target/apl/"
```

设定 *host name* PC, 在 C 盘根目录下,ISaGRAF\target\apl,作为应用文件的备份目录。注意:不要忘记最后的斜杠,否则,备份到 ISaGRAF\目标\目录下,以 APL 为前缀的文件名下。

如果需要,对每个目标,每次作用前,变量可以被设到不同的路径单元。在应用程序备份一章,可以找到更多的关于备份应用程序文件,和相关特性的详细信息。

## ≡ **循环控制结束**

变量 TSK\_NBTCKSCHED (uint32)能被设定:当目标循环结束时,延时几个节拍。默认值为 0(与任务安排同优先级)。

每次内核作用前,对于每个目标,如果需要,此变量可被设为不同的值,在任务特性和任务优先权一章,可以找到关于特殊属性和循环周期的更详细信息。

---

### **ISaGRAF 内核作用**

只要配置环境设定完成,最后一步由 ISaGRAF 内核作用 isa\_main 组成

```
: uchar isa_main
(
    uchar slave, /* 从站编号 */
    char *com     /* 未用的空串 OK */
)
```

#### 描述

启动 ISaGRAF 内核任务。

#### 返回值

如果出错,返回一个非 0 值。

从站号与在从站登录一章所讨论的相同。

只要具有不同的从站号,可以启动多于一个内核。

### **ISaGRAF 通讯任务作用**

只要配置环境设定完成,最后一步由 ISaGRAF 通讯作用任务: tst\_main\_ex. 组成。

```
uchar tst_main_ex
(
```

```
char *com, /* 通讯设备名 */
uchar *slave, /* 4 Bytes 字节域位置，内核从站与之链接
*/
uchar com_id /* 通讯任务标识符 */
)
```

### 描述

启动 ISaGRAF 通讯任务。

### 返回值

如果出错,返回一个非 0 值。

4 Bytes 字节域确定通讯任务与之链接的内核从站,。如果所需内核少于 4 个,此域用 0 补齐。当任务启动后,将不再需要此域。

通讯设备名称与用于通讯的串行设备名对应。

只要具有不同的任务识别号,可以启动多于一个通讯任务。使用工作台调试器时,应当保证工作台通讯参数与存在的目标匹配。(内核和通讯任务)(参见用户指南: 程序管理)

### **示例:**

此例显示如何启动: 带有从站 1 号的 ISaGRAF 内核任务。

ISaGRAF 通讯任务标识号 1, 与内核从站 1 号链接, 使用 /tyCo/1 设备串接。

ISaGRAF 通讯任务标识号 2, 与内核从站 1 号链接, 并且使用端口号 1100 以太网链接。

目标所安装的目录是当前主目录。

装载 isassr.o 模块

ld < RELS/isassr.o

装载 isakeret.o 模块 (当不需要以太网时, 可以装载 isakerse.o 模块)

ld < CMDS/isakeret.o

串行通讯配置

ISAMOD ("/tyCo/1", 19200)

系统时钟频率

CLKRATE = sysClkRateGet ()

从站登录

isa\_register\_slave (1)

通讯登录

isa\_register\_com (1)

isa\_register\_com (2)

文件存储单元 (使用默认值, 可以跳过)

TSK\_FUNIT = ""

循环结束控制 (使用默认值, 可以跳过)

TSK\_NBTCKSCHED = 0

ISaGRAF 内核作用

sp (isa\_main, 1, "")

通讯任务，链接从站

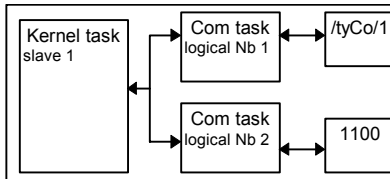
SlavesLink = 0x01000000

ISaGRAF 通讯任务作用

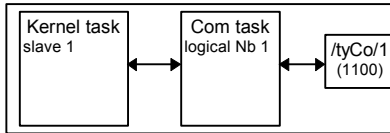
sp (tst\_main\_ex, "/tyCo/1", &SlavesLink, 1)

sp (tst\_main\_ex, "1100", &SlavesLink, 2)

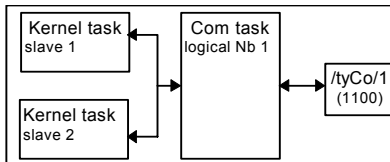
启动对应下图。



也可以选择下列基本配置。



常见的基本配置，串接（以太网），一个内核任务与通讯任务相连。



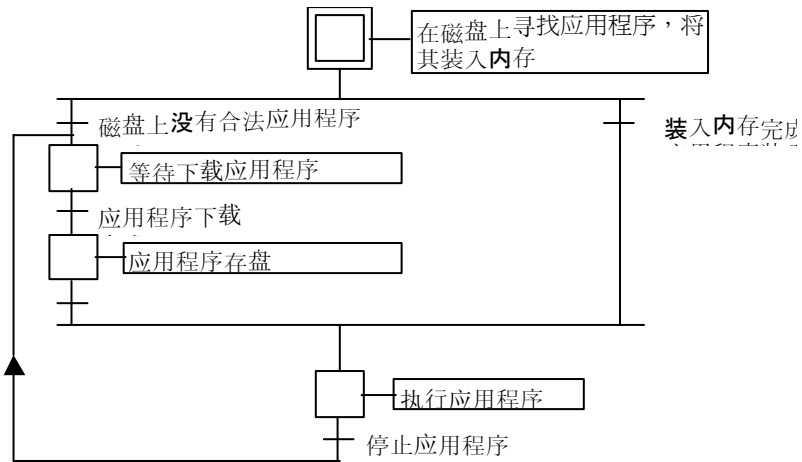
一个另外的配置是：2 个内核与一个通讯任务相连 采用串口（以太网）。

此时：从链接 = 0x01020000。

## C.5.5 特殊属性

### ISaGRAF 启动

当目标启动时，执行下列算法。



- 定义

在工作台产生和下载二进制数据的应用程序代码，然后在目标中执行。可以补加上符号表。

在工作台产生和下载 ASCII 数码的应用程序符号表。此表链接符号对象和内部目标对象。除非用户使用特殊的符号管理，在目标不需要此表。关于符号表的进一步信息，请参阅用户指南中“先进编程技术”一章。

使用全局变量 TSK\_FUNIT (默认值 = "" 确定没有磁盘文件单元)，在目标启动时，设定的磁盘文件单元目录。

- ISaGRAF 的多应用程序

只要不同的应用（内核和通讯任务）具有不同的从站号和通讯任务逻辑号，就能同时运行在同一 CPU。尽管如此，当运行不同应用程序时，用户必须注

意：一些应用程序对象共享访问 I/O 板。例如：除了同类 I/O 服务器或信号器，通过 I/O 驱动实现外，不同应用程序使用不同的“物理”板。

- 备份应用程序

当一个新的应用程序从工作台的调试器下载到目标，应用程序代码存在目标(目标使用标准文件管理程序：fopen,...)，使用下列文件名。

*path*ISAx1        ISaGRAF 应用程序代码备份文件(X 是从站号 )

如果此前下载了应用程序符号表，它也存在目标的当前目录下使用下列文件名

*path*ISAx6 ISaGRAF 应用程序符号备份文件(X 是从站号 )

使用全局变量 TSK\_FUNIT，在目标启动时，确定路径。空串("")表示没有磁盘文件单元(默认值)。

当 ISaGRAF 启动后，在当前目录中找寻应用程序代码和符号文件，并将其装入内存。

如果内存中，没有合法的符号文件。符号就不装入内存，目标开始运行应用程序代码。

如果在内存中，没有合法的应用程序代码，目标就等待下载应用程序。

为了在上电时，不使用调试器链接。启动带有特殊应用程序的目标。

- 第一种方法是：使用任何一种文件传送工具，从工作台所在的主 PC 站，直接将这些文件拷贝到应用程序备份存储单元。可以使用工作台的

“工具”菜单方便此操作(见用户指南：管理程序)。

- 第二种方法是：从工作台所在的主 PC 站，使用工具将这些文件的应用程序代码（必要时，还有应用符号表）存储到非易失的内存中，（PROM 或 EPROM），。

当系统上电时，如需要（例如：快速存取，或断点管理）你可以使用工具，从 PROM 内，装载应用程序代码到 RAM。

然后，ISaGRAF 启动（在任务作用之前），你必须设定内存中的应用程序放置地址(必要时也有符号表) 对此你需要初始化 SSR 全局变量如下：

SSR[x][1].space =应用程序代码位置地址

必要时：

SSR[x][6].space =应用程序符号表位置地址

对此你可以写一个小过程，SSR 全局变量声明为 str\_ssr 结构类型，被定义在 tasy0ssr.h 文件。

#### 警告：

如果应用程序代码不能被“写”访问，ISaGRAF 调试器的断点管理就不能正确的运行。当你事先进行了全面测试，将不会发生此问题。

在 PC 主站, 如果 ISaGRAF 工作台安装在标准的 \ISAWIN 目录下:

MYPROJ 项目的应用程序代码文件是:

\ISAWIN\APL\MYPROJ\appli.x6m ( 对应目标中的 isax1 )。

MYPROJ 项目的应用程序符号文件是：

\ISAWIN\APL\MYPROJ\appli.tst (对应目标中的 isax6 )。

## ❏ “ 出错” 管理和输出信息

ISaGRAF 目标软件集成了故障监测管理。在附录可以找到“ 警告出错” 列表及其解释。

出错监测过程如下:

- 将错误名和参数送到 ISaGRAF “ 出错” 程序
- 如果在工作台的“ 制作” 选项中将“ 出错监测” 标志置位, “ 出错” 将被处理。否则, 信息将丢失, “ 出错” 管理结束。

当处理时:

- 在默认的 stdout 输出显示出错号 ( 十进制 ) 和参数 ( 十六进制 )
- 为了今后能够检索处理, 将出错号和参数一起推入环型 FIFO 寄存器。在工作台的“ 制作” 选项设置此寄存器的尺寸大小。当一个新的出错到达时, 最先出现的那个将丢失。
- 当程序运行时, 使用系统调用; 或者从调试器可以获得“ 出错” 。( 见用户指南 )

当调试器监测到一个“ 出错” 时, 出错信息的描述出现在“ 出错” 窗口。取决于应用状态 ( 运行/停止 ), 调试器可以显示对象的名称 ( 变量或程序 ), 在何处出现的错误, 或者将出错参数 ( 十进制 ) 放入[X]内, 表示对于同一出错号的代表不同意义。

在目标, 当监测到出错, 出错值被显示在默认的 stdout 输出。这样一来显示可以直接用于 VxWorks 程序, 例如:

*ioGlobalStdSet()*

或 *ioTaskStdSet()*

后一种情况, 不能完成内核与通讯任务生成“ 出错” 的重定向。

## ❏ 循环周期, 任务工况, 任务优先级

- ISaGRAF 循环结束时, 启动下一个之前, 执行下列算法:

如果循环时间特别设定，(从工作台:见用户指南：程序管理)CPU 将放弃一段保留的时间段 (特定循环时间 - 当前使用的)。如果此保留时间为负值，将出现溢出。CPU 将放弃 TSK\_NBTCKSCHED(在 ISaGRAF 启动时设的变量)个节拍，以便强制同步准时。

如果没有特设循环时间，或者保留时间小于 1 个节拍，或等于 0，CPU 将放弃 TSK\_NBTCKSCHED 个节拍，强制同步准时。

目标计时精度对应于 VxWorks 系统节拍。

特定时间一般用于触发循环，或者使 CPU 顾及到运行在 VxWorks 系统的其它任务。

- 当没有数据到达通过通讯链接，通讯任务处于休眠状态。当需要时，此任务与内核任务一起，通过问/答通讯协议获取运行的应用程序信息。通讯任务向内核任务提出问题。循环结束时，(为了同步应用映像)内核给以通讯任务回答。

ISaGRAF 任务不修改它们的优先权，用户可以按照上述 ISaGRAF 任务工况和整体应用的要求，调整这些优先权。

## **C.6 ISaGRAF NT 目标的启动**

### **C.6.1 运行 ISaGRAF**

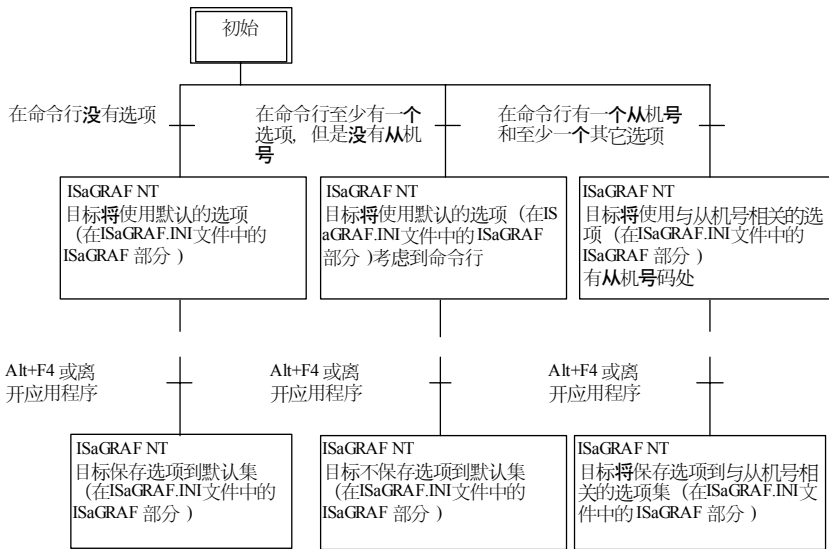
在 NT 系统的实现,目标运行单一程序: WISAKER.EXE, 启动需要一段时间。允许有所期望数量的 ISaGRAF NT 目标, 以及每个实例有不同的从站号。

目标程序不妨碍运行的中断驱动程序。

WISAKER 软件被设计运行在 Windows NT 3.51 或更高的版本。

### **C.6.2 选项的概论**

按下列图表保存和检索选项:



注意：ISAGRAF.INI 被保存在当前工作目录。

### 从站号: -s 选项

此选项确定目标的从站号。可以是除了 13 (\$0D)。之外的从 1 到 255。此从站号被用在全部通讯协议。当在同一 PC 中有多于一个目标时，或者有多于一个目标与同一主站工作台相连接时，用来区分不同的从站。当使用工作台调试器时，应当保证工作台的从站设置与目标相匹配。（参见用户指南“程序管理”）

默认值： 默认的从站号是 1，或是 定义在 ISaGRAF.INI 文件中的那个。

示例:

WISAKER.EXE -s=2

用户接口： 执行 ISaGRAF NT 目标主窗口的“ 选项/从站” 命令后的对话框。



使用鼠标或者上下箭头键可以改变此选项数值。 为了使其生效，必须重新启动 ISaGRAF NT 目标。

#### **☐ 通讯链接和设置：-t 选项**

可以使用串行链接或者以太网链接 ISaGRAF 目标与调试器的通讯。用 -t 选项来确定端口的名称。设计的通讯接口与任何机器相兼容,端口 COM1, COM2, COM3 或 COM4 能被用于串口通讯, 从 1100 始的端口号可用于以太网的通讯。

默认值： 以太网的默认端口号是 1100 ， 串口是 COM1 ；或是定义在 ISaGRAF.INI 文件中的那个

提示: 默认的通讯链接是以太网。

示例:

WISAKER -t=COM2

WISAKER -t=1101

串口设定:

只有当 -t=COMx 选项设定之后。才可以使用其它选项。

这是串口连接配置的选项:

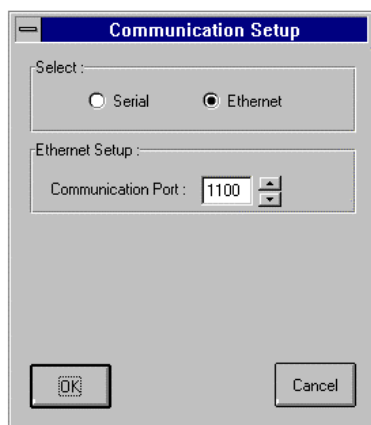
选项	值	定义
波特率	600	波特率
	1200	
	2400	
	4800	
	9600	
	19200	
校验	n	无
	e	偶
	o	奇
数据	7 or 8	位数
停止位	1 or 2	停止位长
流	h	硬件控制
	n	不控制

默认值是 19200, 无奇偶校验, 1 停止位, 不控制流

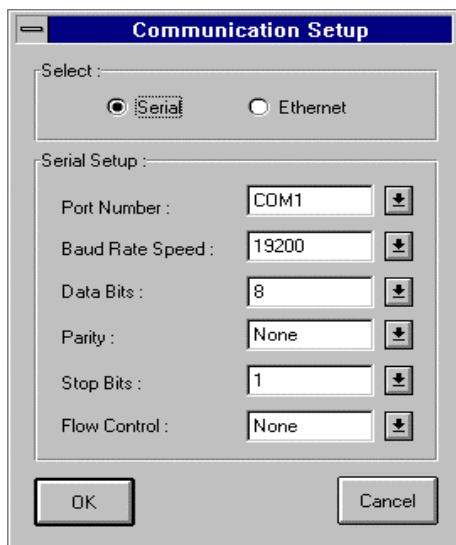
示例:

WISAKER -t=COM1 波特率=1200 数据=8 奇偶校验=n 停止位=2

用户接口: 运行 ISaGRAF NT 目标主窗口的“选项/通讯”命令后的对话框。



可以选择串行通讯或是以太网通讯，以太网通讯可以修订端口号，网口号应等同于工作台的 PC - PLC 链接设定。



当选择了串行通讯，出现设置窗口，设置应等同于工作台的 PC - PLC 链接设定

#### 虚拟板的图形仿真：- X 选项

如果选择了此选项，将板声明为虚拟，在 I/O 连接编辑器(见 A 部分)，将进行仿真。可选择 0 或 1，0 表示无仿真，1 表示有仿真。

默认值：为 0，或为在 ISaGRAF.INI 文件中所设定。

#### 示例：

WISAKER -x=1, 将仿真“虚拟板”，

用户接口：按选项的不同，此菜单项将选定或不选定。仿真板将出现在图形面板。

#### ISaGRAF NT 目标的优先级：- P 选项

如果目标运行于 NT 系统，确立任务优先级是有意义的。例如：目标中对时间要求紧迫的任务(应用程序)定为高优先级，其它的低优先级任务运行在后台。

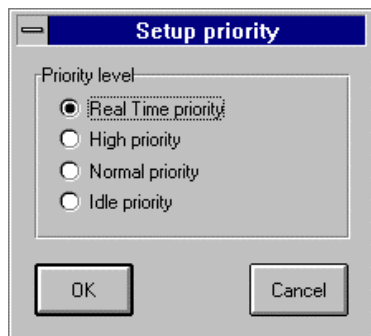
取值范围为 0，1，2，3，0 是最高优先权，3 是最低优先权。

#### 示例：

WISAKER -p=0

WISAKER -p=1

用户接口：此窗口显示在 ISaGRAF NT 系统主窗口的“ 选项/优先权” 命令



实时任务为最高级，闲置任务为最低级

0：实时优先权

1：高优先权

2：普通优先权

3：闲置优先权

☐ 示例：

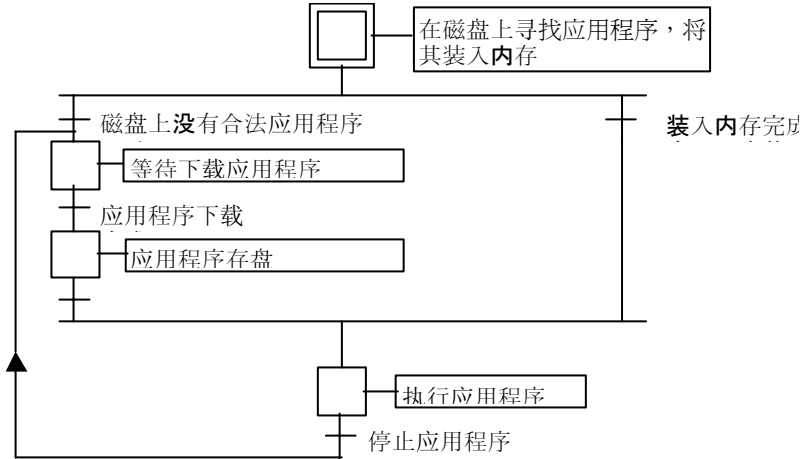
wisaker -t=COM1 启动带以 1 号默认从站的 ISaGRAF 目标，以 COM1 串口为通讯端口。

wisakers=3t=COM1 启动带有 3 号从站的 ISaGRAF 目标，以 COM1 串口为通讯端口。

### C.6.3 特性

#### ISaGRAF 启动

当目标启动后，执行下列算法。



- 定义

在工作台产生和下载二进制数据的应用程序代码，然后在目标中执行。可以补加上符号表。

在工作台产生和下载 ASCII 数码的应用符号表。此表链接符号对象和内部目标对象。除非用户使用 DDE 特性，或是使用带符号名特性的 I/O 仿真的特殊符号管理，在目标不需要此表。关于符号表的进一步信息，请参阅用户指南中“ 先进编程技术”一章。

- ISaGRAF 的多应用

只要不同的应用具有不同的从站号和通讯任务逻辑号，就能同时运行在同一 CPU。尽管如此，当运行不同应用程序时，用户必须注意：一些应用程序对

象共享访问同一 I/O 板。除了同类 I/O 服务器或信号器通过 I/O 驱动实现外，不同应用使用不同的“物理”板。

- 备份应用程序

当一个新的应用程序从工作台的调试器下载到目标，应用程序代码存在目标的当前目录，使用下列文件名：

ISAx1 ISaGRAF 应用程序代码备份文件(X 是从站号)

如果此前下载了应用程序符号表，它也存在目标的当前目录下使用下列文件名：

ISAx6 ISaGRAF 应用程序符号备份文件(X 是从站号)

当 ISaGRAF 启动后，在当前目录中找寻应用程序代码和符号文件，并将其装入内存。

如果没有合法的符号文件，符号就不装入内存，目标开始运行应用程序代码

如果没有合法的应用程序代码，目标就等待下载应用程序。

为了在上电时在目标启动特殊应用程序，如果工作台位于同一台 PC，或是使用软盘，可以直接从同一磁盘将这些文件拷贝到目标的当前目录。

如果 ISaGRAF 工作台安装在标准的\ISAWIN 目录

MYPROJ 项目的应用程序代码文件是：

\ISAWIN\APL\MYPROJ\appli.x8m

MYPROJ 项目的应用程序符号文件是：

\\ISAWIN\APL\MYPROJ\appli.tst

### 示例：

如果下列命令已输入在 WISAKER.EXE 安装的目录：

copy \\ISAWIN\APL\MYPROJ\appli.x8m isa11

这样，将找到 WISAKER.EXE 文件，且执行‘ MYPROJ’ 应用程序。

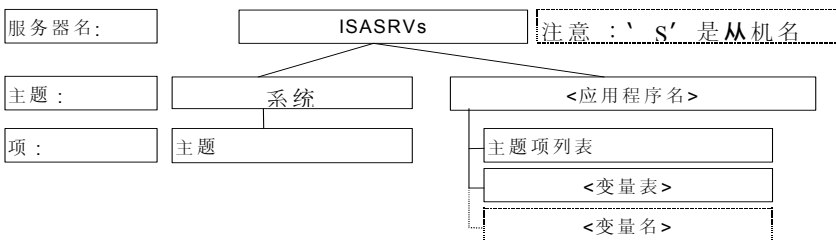
所有这些命令可以合并为一批处理文件，从工作台的“ 工具” 菜单下启动(参见用户指南的“ 程序管理” )

### ☐ DDE 规范

ISaGRAF NT 目标是 DDE 服务器(动态数据交换)，任何软件都能是客户，可以与目标相连接，以交换变量。例如，使用 MSEXCEL 可以以图表方式显示，借助 DDE 从 ISaGRAF 目标获取的数值。

使用 DDE 特性需要目标中的应用程序符号表

DDE 的要点定义如下



《 ISASRVs 》是 DDE 服务器名，‘ S’ 是从站名

《SYSTEM》是给予访问《TOPICS》项的标准主题

《TOPICS》。当前已定义主题的列表：系统和运行在 ISaGRAF NT 目标的应用程序名

《APPLICATION NAME》应用程序名。

《TOPICITEMLIST》当前主题下的变量项列表，可以借助 DDE 访问的变量列表。。

《VARIABLE NAME》变量名

ISaGRAF NT 目标的 DDE advise 循环率：- D 选项

通常：DDE 客户在它每次需要时询问变量，如果变量数量大，就耗费大量时间。有另外一种被称为 advise 方式(advise loop)，采用此方式后，服务器仅发送发生变动的变量，这样通讯耗时少且高效。此时，服务器周期性的查询被标识为“advise”的变量，以便决定应当发送哪些变量，此周期称为 DDE advise 循环率。

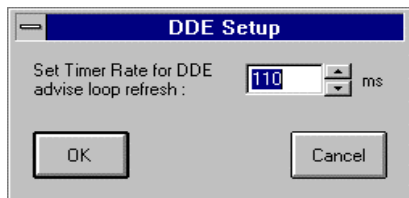
此选项，可以确定从(MS)为单位的 DDE advise 循环率

默认值：为 1000MS，或为被设置在 ISaGRAF.INI 文件中的值

示例：

WISAKERd=100

用户接口：此窗口为 ISaGRAFNT 目标主窗口的“选项/DDE”命令



## “ 出错” 管理和输出信息

ISaGRAF 目标软件集成了故障监测管理。在附录可以找到“ 警告出错” 列表以及解释。

“ 出错” 检测过程如下：

- “ 出错” 由被发送到 ISaGRAF“ 出错” 程序的“ 出错” 和原因号组成。
- 如果在工作台“ 制作” 选项中，置位“ 出错” 检测标志，“ 出错” 将被处理；如果未置位，“ 出错” 信息将丢失，“ 出错” 管理将结束。

处理：

- 当调试器检测到一个“ 出错” ，“ 出错” 解释将显示在“ 出错” 窗口取决于应用程序(运行或停止)的场合，调试器可以显示产生“ 出错” 的对象(变量或程序)的名称，或是“ 参数出错” (10 进制值)在括号内，这表示每个“ 出错” 的不同定义。出错” 号(10 进制值)和“ 参数” (16 进制值)将显示在输出 ( WISAKER.EXE 窗口 )
- 为了今后的检索，“ 出错” 号和“ 参数” 将被推入一个环形 FIFO(先进先出)“ 出错” 暂存器。在工作台“ 制作” 选项中设置此“ 出错” 暂存器尺寸。如果暂存器已满，当再推入一个新“ 出错” 时，那么最先的那个“ 出错” 信息将丢失。

- 或是从调试器，或是从运行应用程序中，使用了系统调用(见用户指南)“  
出错”。

当目标启动后，在输出显示一条欢迎信息。信息由从站号，通讯配置和 DDE 服务器名组成

## ☐ **系统时钟**

ISaGRAF NT 目标设计为运行在任一系统，用于循环同步和时间变量刷新的标准节拍为 10 微秒。这样，就不可能有精度高于 10MS 的时间变量。同样，当特定的循环周期小于或等于 10MS，以及对零的偏差将产生一个循环周期溢出“出错”（“出错” 62），为得到详情，请参见下述章节。

如果你的应用需要更高的精度，对此特殊要求，请咨询供货商。

## ☐ **循环周期和目标特性**

一个 ISaGRAF 循环结束，启动一个新循环之前，执行下列算法：

### **如果循环周期被设定(从工作台：见用户指南：程序管理)**

CPU 将放弃保留的时间周期 (特定的循环时间 – 当前应用的)。如果这个保留的时间周期为负值，将产生溢出，CPU 将放弃一个周期以便强制同步。

如果没有指定循环周期,或者保留的时间小于或等于系统节拍,或是等于零，CPU 让出一个周期来强制与系统同步

目标的时间精度对应于 Windows NT 的节拍。

特定的循环周期一般用来触发循环，或者使 CPU 与运行在 Windows NT 系统的其它过程协调。

## 离开键

当使用台式 PC, 在非工业条件下测试应用程序时，用户可能想停止 ISaGRAF：按组合键可以防止不期望的停止，键的顺序是：

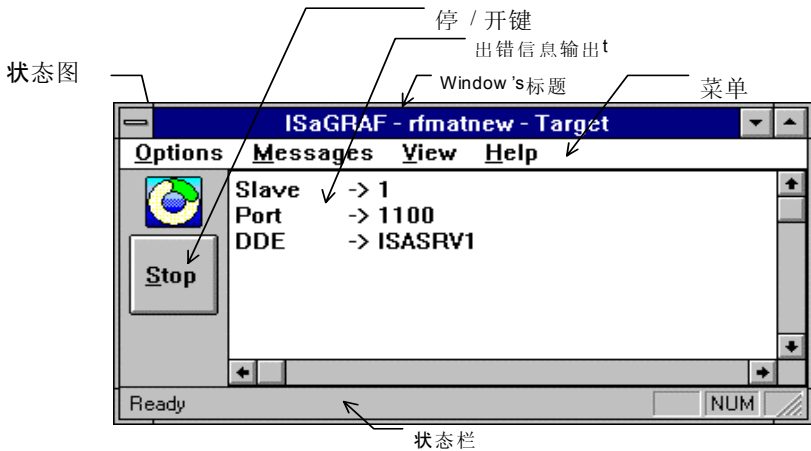
alt + F4

这种快速离开的危险的副效应是，IO 板的接口还没有关闭。这样，停止 ISaGRAF 的方法是：

- 从调试器停止运行应用程序，或者使用停/开按键（将关闭 I/O 板）
- 从 ISaGRAF 系统菜单停止。

### C.6.4 用户接口

这是 ISaGRAF NT 目标的用户接口界面：



主项：

窗口标题

菜单栏

运行状态栏

停开键

出错和信息输出

状态栏

窗口标题 《 ISaGRAF – 应用程序名称 - target 》

其中 name\_of\_appli 是运行的“ 应用程序名” ， 如果没有运行的应用程序，  
将仅显示 《 ISaGRAF - - Target 》 。

### ■ **ISaGRAF NT 目标菜单栏:**

菜单栏有四个菜单:

选项

信息

查看

帮助

#### ● "选项" 菜单

(参见 NT 的第一部分:选项)

"选项" 菜单可以访问运行的选项。 可以有列选项:

从站可以修改从站号，仅在下次启动目标时，修改了的选项才能生效。如果目标已经带有至少一个在命令行的选项启动，此特性将不再有效。

通讯可以修改通讯配置，仅在下次启动目标时，修改了的选项才能生效。如果目标已经带有至少一个不同于-S 选项启动，此特性将不再有效。

DDE 可以修改 DDEadvise 循环率，仅在下次启动目标时，修改了的选项才能有效。如果目标已经带有至少一个不同于-S 选项启动，此特性将不再生效。

仿真 I/O 可以反映此选项的是否选中的状态，仅在下次启动目标时，修改了的选项才能生效

优先权 可以修改优先权，修改将立即有效。

默任的选项

仅可对下列项恢复为默认选项。

通讯

DDE

窗口屏幕的坐标

仅在下次启动目标时，修改了的选项才能有效。

如果目标已经带有至少一个不同于-S 选项启动，此特性将不再有效。

- “ 信息”菜单

“ 信息”菜单是输出的管理，包括下列两个指令：

应答 停止在发生出错或信息时的频闪。

全清 全部清除输出。





 **ISaGRAF NT 目标图标:**

此图标反映目标的状态:

- 应用程序运行，图标转动。
- 没有应用程序，（或者应用程序停止），图标停止转动。

- 在输出窗口有出错或者提示信息时，图标中心将发出红色的频闪，为了停止频闪，可以选择信息菜单的《应答》项；或者同一菜单的《全清》项（此项将全部清除输出窗口）。在“出错”管理和输出信息一章，可以获得更多的信息。

下列图表总结了目标的不同状态：

	没有出错	出错或者信息 (中心为红色)
应用程序正在运行		
没有应用程序		

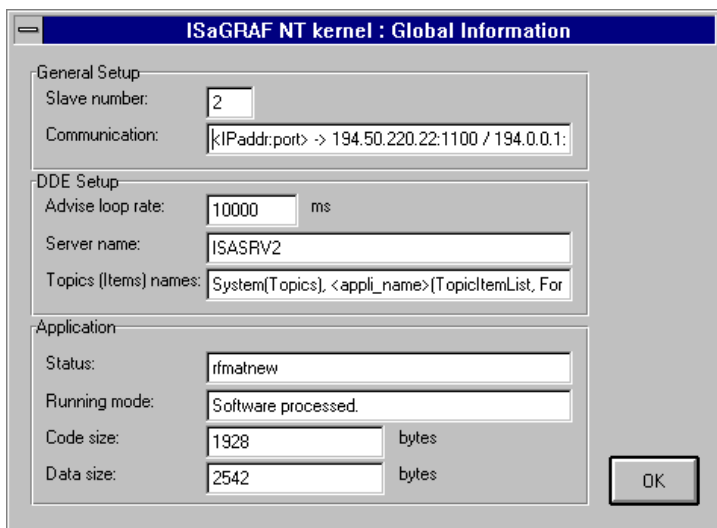
#### ISaGRAF NT 目标停止 / 开始 按键:

**停止/开始 按键完全等同于调试器的开/停功能。按键上的文字表明了应用程序的运行状态。如果应用程序运行，其文字为《停止》。如果应用程序**

**停止（或者没有应用程序），其文字为《开始》（注意当没有应用程序，又请求启动时，按键先切换到停止模式，然后又返回到启动模式）**

#### ISaGRAF NT 目标，总信息

通过“查看/信息”命令，下列对话框给出了目标配置和运行的应用程序的总信息：



The image shows a Windows-style dialog box titled "ISaGRAF NT kernel : Global Information". It is divided into three sections: "General Setup", "DDE Setup", and "Application".

- General Setup:**
  - Slave number: 2
  - Communication: kIPAddr:port> -> 194.50.220.22:1100 / 194.0.0.1:
- DDE Setup:**
  - Advise loop rate: 10000 ms
  - Server name: ISASRV2
  - Topics (Items) names: System(Topics), <appli\_name>(TopicItemList, For
- Application:**
  - Status: rformatnew
  - Running mode: Software processed.
  - Code size: 1928 bytes
  - Data size: 2542 bytes

An "OK" button is located at the bottom right of the dialog box.

有三项主题:

a) 总设置:

- 从站号
- 通讯配置 (如果采用以太网连接,除了端口号之外,还有当前 NT 系统上有效的 IP 地址)

b) DDE 设置

- Advise 循环率
- DDE 服务器名
- DDE 主题和项名。这仅是一般信息,并不反应实际值,事实上,在括号<>内的域由实际值取代。

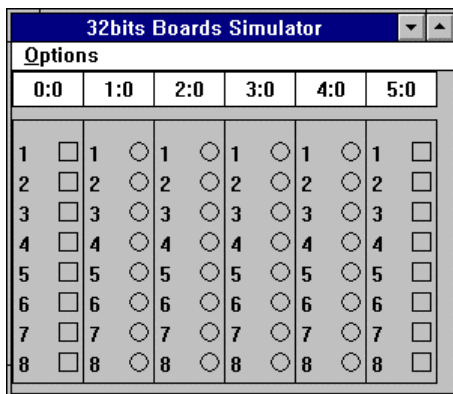
c) 应用程序

- 当有应用程序运行时,它的名称就是程序的状态。当没有程序运行时,出现字符串“没有应用程序运行”。

- 应用程序运行模式,如果应用程序运行在软件处理器,此时出现《软件处理》字符串。或者应用程序已被编译器所编译完,出现字符串《C 编译了》。如果没有应用程序运行出现字符串:《没有应用程序运行》。
- 以字节为代码的尺寸。如果运行方式是《C 编译了》,这个域为空。
- 以字节为代码的尺寸。这是实时内部变量和变量数据库的汇总。

### ISaGRAF NT 目标的虚拟板仿真:

当选定了《仿真 I/O》选项之后,下一次应用程序启动时,显示下列窗口



取决于你的 I/O 连接配置,将有不同数量和种类的板和变量。每个板上方的号码《S:T》表示槽的标识(S)和板的标识(T)。从零开始计数,不能修改。

“32 位板的仿真器”窗口工作与应用程序的停开状态同步。如果带有虚拟板的(或使用仿真板)的应用程序运行,并且《仿真 I/O》标志置位,

此窗口将出现。反之，当按下停止按键时，它将关闭。此窗口随着 I/O 调用而工作。

“选项”菜单可以有两项：

如果符号表已经在代码之前下载，变量名将显示变量名称。

十六进制 每个整型量将不以十进制格式，而以十六进制值显示。

变量名如下图所示：

32bits Boards Simulator											
Options											
0:0		1:0		2:0		3:0		4:0		5:0	
1	<input type="checkbox"/> ROW0	1	<input type="radio"/> LED00	1	<input type="radio"/> LED10	1	<input type="radio"/> LED20	1	<input type="radio"/> LED30	1	<input type="checkbox"/> COL0
2	<input type="checkbox"/> ROW1	2	<input type="radio"/> LED01	2	<input type="radio"/> LED11	2	<input type="radio"/> LED21	2	<input type="radio"/> LED31	2	<input type="checkbox"/> COL1
3	<input type="checkbox"/> ROW2	3	<input type="radio"/> LED02	3	<input type="radio"/> LED12	3	<input type="radio"/> LED22	3	<input type="radio"/> LED32	3	<input type="checkbox"/> COL2
4	<input type="checkbox"/> ROW3	4	<input type="radio"/> LED03	4	<input type="radio"/> LED13	4	<input type="radio"/> LED23	4	<input type="radio"/> LED33	4	<input type="checkbox"/> COL3
5	<input type="checkbox"/>	5	<input type="radio"/>	5	<input type="radio"/>	5	<input type="radio"/>	5	<input type="radio"/>	5	<input type="checkbox"/>
6	<input type="checkbox"/>	6	<input type="radio"/>	6	<input type="radio"/>	6	<input type="radio"/>	6	<input type="radio"/>	6	<input type="checkbox"/>
7	<input type="checkbox"/>	7	<input type="radio"/>	7	<input type="radio"/>	7	<input type="radio"/>	7	<input type="radio"/>	7	<input type="checkbox"/>
8	<input type="checkbox"/>	8	<input type="radio"/>	8	<input type="radio"/>	8	<input type="radio"/>	8	<input type="radio"/>	8	<input type="checkbox"/>

## C.7 "C"语言编程

### C.7.1 概览

本指南的对象是：对 ISaGRAF 概念和工作台工具已有经验的用户。当使用 ICS Triplex ISaGRAF Inc.标准库的变换函数，“C”函数和功能块开发自动化应用程序之后，可以开发用户自定义的变换函数，“C”函数和功能块。建立新库，可使用户增强 ISaGRAF 目标 PLC 的能力，发挥出硬件平台和柔性工作站的<sup>1</sup>最大输出能力。

借助“C”语言开发系统，以及过去使用“C”语言编程的经验，本指南用户能使它的 ISaGRAF 目标 PLC 达到更加适应于它本身特点的最佳控制。这些开发改善了目标 PLC 的特性，使用 ISaGRAF 工作台的自动化技术编程员将能更加方便的完成高质量的开发。

#### ☐ 标准 ISaGRAF 工作台特性

此文档的信息不是专用于某一专门目标系统。但是一些特性(例如：多任务能力)并不适于某些单任务系统。

ISaGRAF 工作台提供了很多功能，用于在自动化开发方面，管理“C”组件库。对自动化技术编程而言，“C”变换函数或功能块是一个黑箱，仅需定义它的接口。

ISaGRAF 库管理器用于在已有的库里增填新的组件，并且定义“C”实现和在 ST/FBD 程序使用它的组件之间的接口。ISaGRAF 库管理器还提供了自动生成变换函数，函数，功能块的 C 源码框架，以及编辑此类“C”源码文件

的工具。请参见 ISaGRAF 用户指南，以获取更多关于“库管理器”功能的信息。

### ☐ “C” 语言开发

ISaGRAF 工作台并不包括“C”编译器和交叉编译工具。用户必须有自己的编译器，用于 ISaGRAF 目标系统，集成他的“C”部件到 ISaGRAF 内核。

当使用交叉编译器时，ISaGRAF 工作台提供用户入口，在 WINDOWS 的 DOS 窗口下，运行用户自定义的 MS - DOS 批命令文件 ( BAT )。交叉编译器必须运行在 WINDOWS 下的 DOS 窗口，如果不是这样，在运行编译器和链接在纯 MS - DOS 前，必须关闭 WINDOWS。

### ☐ 技术说明

在 ISaGRAF 库管理器，用户可以写对于每一个库组件的文字说明，此技术说明是“C”组件开发的用户指南。将有益于自动化技术程序员，描述如何在 ISaGRAF 应用程序中使用变换函数，函数或功能块。

必须在技术说明中精确的定义变换“C”函数或功能块。使自动化技术程序员能真正将它当成集成的 ISaGRAF 函数使用。对于“C”函数，技术说明应描述

☐函数详细的功能

☐调用参数的完整描述

- 返回值的意义
- 调用参数和返回值的具体类型
- 应用场合

对于“ C” 功能块的技术说明包括

- 块的具体功能
- 调用参数的完整描述
- 返回参数的意义
- 调用及返回参数的具体类型
- 应用场合

变换函数的技术说明应描述

- 对输入变量使用时变换的定义
- 对输出变量使用时变换的定义
- 变换所处理值的极限

技术说明还应包括下述信息

- 变换函数，函数和功能块的完整标识
- 关于更新和维护的所有信息
- 所支持的目标系统
- 多任务特性
- 所需的系统服务、内存和驱动

## C.7.2 "C" 变换函数

ISaGRAF 工作台包括一个线性变换实用工具, 在 ISaGRAF 目标 PLC 运行时, 进行简单的模拟量 I/O 的变换。此实用工具不需要任何的 "C" 开发, 它仅用于连续的增减函数。参见用户指南关于此工具的详细描述。

变换函数使用户采用 "C" 语言的特殊操作, 完成复杂变换。一般讲, 变换函数用于输入和输出。即使没有使用其中的之一。为了保护系统不由于错误的调用而崩溃, 在集成到 ISaGRAF 内核前, 必须进行测试和实现。

用 "C" 语言写的变换函数, 编译并且与 ISaGRAF 内核链接。在 ISaGRAF 项目中使用新的变换函数前, 必须将增加后的内核安装在 ISaGRAF 目标 PLC。新的变换函数不能集成到 ISaGRAF 仿真器。在插入新的非标准变换函数前, 必须先仿真 ISaGRAF 应用程序。

由 ICS Triplex ISaGRAF Inc. 编写的标准变换函数 "C" 源代码, 已安装在 ISaGRAF 工作台。它们可以作为编写新的函数的范例。推荐不要改动标准函数, 以使它们可以用于任何 ISaGRAF 应用程序。ISaGRAF 仿真器支持由 ISaGRAF 工作台提供的标准变换函数。

**警告:** 变换函数是同步操作, 在程序循环时的输入和输出阶段, 由 ISaGRAF I/O 处理器激活。变换函数的执行时间包括在 ISaGRAF 应用程序循环时间内。用户必须保证: 在变换函数内没有“等待操作”, 致使 ISaGRAF 循环周期没有不必要的延长。

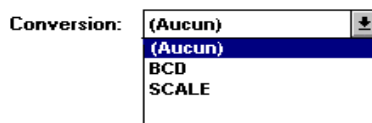
## 在 ISaGRAF 库中增加函数

使用工作台的 ISaGRAF 库管理器,将变换函数加入到 ISaGRAF 库,当变换函数库选定后,使用“文件”菜单的“新建”命令。不用在工作台定义参数,因为变换函数使用标准的预定义接口。

当创建了变换函数后,必须写它的技术说明。ISaGRAF 库管理器为新的变换函数,自动生成变换函数的“C”源代码框架。

## 在 ISaGRAF 项目中使用变换函数

所选项目的输入输出模拟变量的过滤,能由定义了的变换函数完成。为了将变换函数用在变量上,运行变量声明编辑器,选定一个输入或输出模拟变量,然后编辑它的参数。模拟变量声明对话框的“变换”域,设置用于 I/O 模拟变量的变换函数。



变换函数和表同时出现在列表中。这表明:函数和表不能同名。一个变量没有定义或集成在 ISaGRAF 内核之前,不能连接变换函数。

## 标准“C”接口

变换函数的接口总有同一格式。调用和返回参数通过结构传递,在“TACN0DEF.h”文件中定义结构:

```
/* 名称: tacn0def.h 目标变换定义文件*/
```

```
#define DIR_INPUT 0          /* 方向 = 输入变换 */
```

```
#define DIR_OUTPUT 1          /* 方向 = 输出变换 */

typedef int32  T_ANA;         /* integer ANA type      */
typedef float  T_REAL;        /* real ANA type          */

typedef struct {              /* 变换结构              */
    uint16 number;            /* 变换号 (保留) */
    uint16 direction;         /* 变换方向            */
    T_REAL *before;           /* 变换前值            */
    T_REAL *after;            /* 变换后值            */
} str_cnv;

#define ARG_BEFORE (*(arg->before))
#define ARG_AFTER  (*(arg->after))
#define DIRECTION  (arg->direction)

/* eof */
```

"str\_cnv"结构完整的描述了接口。"C" 变换函数的参数是指向结构的指针 "号码"域是变换函数逻辑号 (在 ISaGRAF 库),在编程中不使用。

"direction"域表示:是对输入还是输出变量变换, DIR\_INPUT 值用于输入 变换;DIR\_OUTPUT 值用于输出变换。

"before"域指向变换前的值。 对于输入和输出的含义不同。当“ direction”有 DIR\_INPUT 值。 对于输入变换表示电量 (从输入读入); direction 有 DIR\_OUTPUT 值,表示输出变换的物理量值(使用在程序的等式中)

"after" 域指向变换之后的值,它对于输入和输出变换有不同的意义。当 direction 域采用 DIR\_INPUT 值,它表示输入变换的物理值(用在编程的等式)。当 direction 域采用 DIR\_OUTPUT 值,。它表示输出变换的电量值(送到输出设备)。

程序员使用 "ARG\_BEFORE" 和 "ARG\_AFTER" 定义去直接访问传送 "c" 变换函数结构的 before 和 after 域。处理的量值是单精度浮点数,对于一个整型模拟变量,其结果转成长整型。这意味着,同一变换可以用于实型或整型模拟 I/O 变量

## ≡ 源代码

*因为变换函数能被用于输入和输出的模拟量,函数源代码被分为两部分:*

**输入变换和输出变换。结构接口的 direction 域用来选择变换方向。当创建变换函数时, ISaGRAF 库管理器自动生成完整的函数的框架。这包括主选择“ IF”结构。**

**下面是一个变换函数的标准框架**

/\*变换函数

名称: sample

\*/

#include <tasy0def.h>

#include <tacn0def.h>

void CNV\_sample (str\_cnv \*arg)

```
{  
    if (DIRECTION == DIR_INPUT) { /*INPUT CONV*/  
  
    }  
    else { /*OUTPUT CONV*/  
  
    }  
}
```

/\* 下面函数显示使用函数名与 ISaGRAF I/O 管理器链接,此函数完全由 ISaGRAF 库管理器生成。 \*/

```
UFP cnvdef_sample (char *name)  
{  
    sys_strcpy (name, "SAMPLE"); /* 给出变换名称 */  
    return (CNV_sample); /* 返回实现函数 */  
}
```

完成函数的特定部分的最好方法是：分别为输入和输出变换写两个局部函数。主算法将调用它们，如在上例的注释显示的，采用主 IF 结构。

包括来自 ISaGRAF 内核的文件 "TASY0DEF.H"，用于系统相关的定义。它也包括 UFP 类型定义，它表示指向空函数的指针，用于声明函数。

### □ **项目和 "C" 实现间的链接**

使用变换的名称，达到变换函数的实现与在 ISaGRAF 项目中使用的变换函数的逻辑链接。将声明函数加到变换函数的源代码。当应用程序启动时，仅

调用此函数一次。使 ISaGRAF I/O 管理器获得变换名称，对应于函数的实现。这是声明函数的标准格式：

```
UFP cnvdef_xxx (char *name)
{
    strcpy (name, "XXX");    /* 变换的名称 */
    return (CNV_xxx);        /* 返回实现函数 */
}
/* (xxx 是变换的名称) */
```

strcpy 语句使用的函数的名称必须大写。

变换实现函数和声明函数的名称必须小写。

对实现函数和定义函数使用 "CNV\_" 和 "cnvdef\_" 前缀，使用户可以用 "C" 语言的保留关键字，或者已存在的 "C" ISaGRAF 库的函数名为变换的名称。

为了实现与此变换相关的初始化操作，可以加其它的语句到声明函数。当应用程序启动时，ISaGRAF 系统许可用户调用此函数一次（仅一次）。

即使声明函数没有用于 ISaGRAF 应用程序，任何集成的变换函数将调用它。如果在应用程序中使用的变换没有事先集成到内核，ISaGRAF 内核将出现致命错误，而崩溃。

新函数与内核链接之前，用户必须写一个名为 "GRCN0LIB.C" 的另外的源文件。将它（保留的变换函数）插入到用于链接器的文件列表中。"GRCN0LIB.C" 仅包括声明函数的数组。此数组在应用程序初始化时读到的，为了与用 "C" 语言写的变换函数动态链接。下为这类文件的例子：

/\* 文件 "GRCN0LIB.c" – 标准库变换的示例 \*/

```
#include <tasy0def.h>          /*定义类型 */
```

```
extern UFP cnvdef_scale (char *name);/* 声明 , SCALE 变换函数 */
```

```
extern UFP cnvdef_bcd (char *name); /* 声明。BCD 变换函数 */
```

```
UFP_LIST CNVDEF[] = {        /* 声明函数数组 */
```

```
    /* 集成的变换函数 */
```

```
    cnvdef_scale,
```

```
    cnvdef_bcd,
```

```
    NULL};
```

```
/* 文件结束 */
```

CNVDEF 数组是 NULL 空指针结束的。当此条件不满足，将发生冲突。如果事先没有定义 CNVDEF 数组，链接一个新的 ISaGRAF 内核时，产生“未定义”出错。

在写此文件时，包括所有已存在变换在内，生成新内核。，当插入一个仅用于一个项目的变换到 CNVDEF 数组，将建立一个专用内核。当应用程序代码已经生成，ISaGRAF 代码生成器自动产生 "GRCN0LIB.C" 文件。此文件放置在 ISaGRAF 项目目录，仅包括在本项目中用到的变换。

## 限制

ISaGRAF 库可以包括至多 128 变换函数。在变换函数中可以处理任何一种类型的操作。提醒：ISaGRAF 循环中调用此函数采用同步方式，所以执行函数将直接影响到系统循环时间。

### C.7.3 "C" 函数

"C" 函数用于提高 ST 和 FBD 语言性能，它们可用来实现任何特殊计算，系统调用，通讯，或建立在 ISaGRAF 应用程序和其他任务间的一系列对话服务。用 "C" 语言写的函数，与 ISaGRAF 内核相链接，以及编译。在 ISaGRAF 项目使用新函数之前，必需将增加了的内核下载到 ISaGRAF 目标 PLC 中。

新函数不能集成到 ISaGRAF 仿真器。在使用非标准函数前，必须仿真 ISaGRAF 应用程序。

注意：函数是同步操作，被 ISaGRAF 内核在应用程序循环运行时激活。执行函数所耗费的时间也包括在 ISaGRAF 应用程序循环周期内。用户必须确保没有“等待操作”编程在函数中，这样，ISaGRAF 循环周期，就不会有不必要的延长。

#### ■ 在 ISaGRAF 库中增加一个函数

在工作台，使用 ISaGRAF 库管理器，将一个新的“C”函数加到 ISaGRAF 库中，当选中一个新函数库时，使用“文件”菜单的“新建”命令。当创建一个新函数时，必须写它的技术说明。新函数的“C”源代码框架将由 ISaGRAF 库管理器自动生成。

“编辑”菜单的“参数”命令用来定义新函数的调用和返回参数。

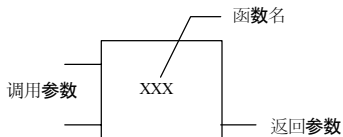
#### ■ 在 ISaGRAF 项目中使用“C”函数

在 ISaGRAF 项目的程序中，任何集成的“ C” 函数当成标准函数使用。可以在 ST 和 FBD 语言程序中调用“ C” 函数，使用 SFC 语言的特殊语句也可以调用。

从 ST 语言程序调用“ C” 函数，应遵照调用函数的语言规则。在函数名称之后，在括号之间，由逗号隔开，写入函数的调用参数。表达式表示函数的返回值。可以将“ C” 函数调用插入任一目标值语句，或者复杂表达式中。这是一个在赋值语句中调用“ C” 函数的示例：

```
result := ProcName (par1, par2, ... parN);
```

FBD 程序可以调用任一“ C” 函数。函数表示为一个标准功能方框。调用参数连接到功能方框的左侧。返回参数连接的方框的右侧。这是一个标准的功能方框图



可以从 SFC 语言的任一动作块，或是一个变换的任一布尔表达式中调用“ C” 函数。

#### 定义“ C” 函数的接口

“编辑”菜单的“参数”命令，用来定义一个新函数的调用和返回参数。一个函数可以有至多 31 个调用参数，和仅有一个返回参数。

窗口的上边显示“ C” 函数的参数列表，遵照函数调用规则样式：先是调用参数，最后是返回参数。窗口下部显示表中当前选中的参数的详细描述。

- 参数名称
- 参数方向(调用/返回)
- (参数类型)

任何 ISaGRAF 数据类型都可以用于参数：布尔，整型模拟，实型模拟，计时器或信息。必须区分整型和实型模拟量。

下面是“ C” 类型和 ISaGRAF 类型对照表

BOOLEAN	unsignedLong	无符号的 32 位字: 1=真 / 0=假
ANALOG	long	有符号的整型 32 位字
REAL	float	单精度浮点值
TIMER	unsignedLong	无符号的整型 32 位字(单位为微秒)
MESSAGE	char *	字符串。

当一个“ 信息” 值被传递到“ C” 函数，它不能包含空字符。字符串传递到“ C” 代码是空字符结束的。不要忘记，返回参数必须位于列表最后。在参数命名时，必须遵守下列规则：

- 名称长度不能超出 16 个符号
- 首字符必须是字母
- 随后字符可以是字母，数字，或下划线
- 名称不区分大小写

一个函数的参数不能重名。调用参数和返回参数不能重名。同一名称可用于不同函数的参数。返回参数的默认名是“ Q” 。此名称可以任意修改。参数名称用于在“ C” 源码中识别参数。

“ 插入” 命令用于在当前选中参数之前，插入一个新参数。“ 删除” 命令用来删除当前选中的参数。“ 重排” 命令用来将列表中的参数重新排序，返回参数置于列表的最后。按下“ 确认” 按钮，存储函数接口的定义，并且关闭对话框。按下“ 取消” 按钮，关闭对话框，但不保存函数接口定义的变动。

### 函数“ C” 语言接口

函数的接口取决于它所定义的参数。调用和返回参数通过结构传递。结构在“ GRUSONNN.H” 文件中定义，其中 NNN 是函数在 ISaGRAF 库中的逻辑号。这是一个正弦函数(SIN)的“ C” 语言接口的示例：

```
/*文件：GRUS0255.h - 函数 "示例" */

typedef long          T_BOO;
typedef long          T_ANA;
typedef float         T_REAL;
typedef long          T_TMR;
typedef char          *T_MSG;

typedef struct {
    /* CALL */          T_REAL _param1;
    /* RETURN */       T_REAL _param2;
} str_arg;
```

```
#define PARAM1          (arg->_param1)
#define PARAM2          (arg->_param2)
```

```
/* end of file */
```

如下显示，ISaGRAF 与“C”类型的关系，ISaGRAF 类型在函数的定义文件中被定义为“C”类型

布尔	T_BOO	long (32 bits)
整型模拟	T_ANA	long
实型模拟	T_REAL	float (32 bits – 单精度)
计时器	T_TMR	long
信息	T_MSG	char * (32 bits - char pointer)

“STR\_ARG”结构的每一个域对应于函数的一个参数，返回参数位于结构最后。出现在结构中的调用参数的顺序与函数定义中所建的一样。大写的标识符被定义为可直接访问结构中的传递函数 C 实现的参数。标识符名称是在 ISaGRAF 库管理器定义函数时输入的。

每次使用 ISaGRAF 库管理器，对函数的接口做出变动，其“C”定义文件也随之更新。这就保证了“函数实现”和“使用在 ISaGRAF 应用程序”之间的完全匹配。

## 源代码

下面是“C”函数实现的标准框架

```
/* 用户自定义函数示例 - 号码是“ 255” - 名称是“ 示例” */
```

```
#include "tasy0def.h"          /* 公共定义 ISaGRAF 内核*/
```

```
#include "grus0255.h"          /* 函数 255 的接口定义 */
```

```
void USP_sample (str_arg *arg)
```

```
{
```

```
    /* 函数体 */
```

```
}
```

/\* 下列函数用于初始化函数和声明它的实现。使用函数的名称，完成了与 ISaGRAF 内核的链接。此函数在 ISaGRAF 库管理器已完全生成。 \*/

```
UFP uspdef_sample (char *name)
```

```
{
```

```
    strcpy (name, "SAMPLE");    /* 函数命名 */
```

```
    return (USP_sample);        /* 返回实现函数 */
```

```
}
```

```
/* 文件结束 */
```

"TASY0DEF.H" 中包括来自 ISaGRAF 内核的文件，用于系统相关的定义。它也包括 UFP 类型的定义，代表了指向空函数的指针，用于声明函数。

## 项目与"C"实现间的链接

用函数名来逻辑链接 "C"函数实现和它在 ISaGRAF 项目的程序中的使用。将一个声明函数加到函数的"C"源代码。当应用程序启动时，此函数仅被调用一

次,对 ISaGRAF 内核表明"C"函数名,代表它所实现的函数。这是声明函数的标准格式:

```
UFP uspdef_xxx (char *name)
{
    strcpy (name, "XXX");    /* 函数命名 */
    return (USP_xxx);        /* 返回实现函数 */
}
/* (xxx 是函数名) */
```

用于 strcpy 语句的 "C"函数名必须大写。在声明函数和实现函数的 "C"函数名必须小写。对于声明函数和实现函数使用“ USP” 和“ uspdef”前缀,可以使用户使用"C”语言的保留关键字作为函数名,或者使用“ C” ISaGRAF 库的已经存在的函数名。

为了实现与此变换相关的初始化操作,可以加其它的语句到声明函数。当应用程序启动时, ISaGRAF 系统许可用户调用此函数一次 (仅一次)。

即使声明函数没有用于 ISaGRAF 应用程序,任何集成的变换函数将调用它。如果在应用程序中使用的变换没有事先集成到内核, ISaGRAF 内核将出现致命错误,而崩溃。

新函数与内核链接之前,用户必须写一个名为“ GRCN0LIB.C”的另外的源文件。将它 (保留的变换函数)插入到用于链接器的文件列表中。“ GRCN0LIB.C”仅包括声明函数的数组。此数组在应用程序初始化时读到的,为了与用“ C”语言写的变换函数动态链接。下为这类文件的例子:

/\*文件 "GRUS0LIB.c" – 使用三角函数的例子 \*/

#include <tasy0def.h>                      /\* 定义类型 \*/

extern UFP uspdef\_fc1 (char \*name);   /\* 声明函数 \*/

extern UFP uspdef\_fc2 (char \*name);

extern UFP uspdef\_fc3 (char \*name);

extern UFP uspdef\_fc4 (char \*name);

UFP\_LIST USPDEF[ ] = {              /\* 定义函数的数组 \*/

    /\* 集成的函数 \*/

    uspdef\_fc1,

    uspdef\_fc2,

    uspdef\_fc3,

    uspdef\_fc4,

NULL};

/\* 文件结束 \*/

USPDEF 数组由空指针结束。当此条件不满足时，将发生冲突。当链接新的 ISaGRAF 内核前，如果还没有定义 USPDEF。将出现错误。当写此文件时，产生包括所有已有函数的新内核。将仅用于该项目的函数插入到 USPDEF 数组，也可以产生用于一个项目的内核。应用程序的代码生成后，ISaGRAF 代码生成器自动产生"GRUS0LIB.C" 文件。此文件放在 ISaGRAF 项目目录，与仅用于这个项目的函数成一组。

## 限制

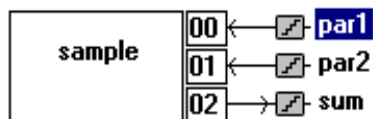
ISaGRAF 库可包括至多 255 个"C" 函数。在函数里可以进行任何类型的操作。应当记住，在 ISaGRAF 循环中，函数的调用是同步的，因此函数的执行将直接影响到循环时间。

## 完整的范例

下面是加法函数编程的完整的范例，函数的技术注释：

名称:	SAMPLE
描述:	整型模拟量加法
创建日期:	1st July 1992
作者:	ICS Triplex ISaGRAF Inc.
调用:	par1, par2: integer operands
返回:	integer sum
样式:	sum := sample (par1, par2);

函数的接口:



下面是函数的"C" 源代码的头:

```
/* 文件: GRUS0255.h – 用户 C 函数定义- 名: sample */
```

```
/*定义标准 ISaGRAF 数据类型 */
```

```
typedef long T_BOO;  
typedef long T_ANA;  
typedef float T_REAL;  
typedef long T_TMR;  
typedef char *T_MSG;
```

```
/* 定义调用和返回参数类型 */
```

```
typedef struct {  
    T_ANA _par1;          /* 调用参数 #1 */  
    T_ANA _par2;          /* 调用参数 #2 */  
    T_ANA _sum;           /*返回参数 */  
} str_arg;
```

```
/* 用于访问调用和返回参数的标识*/
```

```
#define PAR1          (arg->_par1)  
#define PAR2          (arg->_par2)  
#define SUM           (arg->_sum)
```

```
/* 文件结束 */
```

下面是函数的"C"源代码。“C”程序员仅输入黑体字部分。

```
/* 文件: GRUS0255.c – 用户 C 函数 – 名: SAMPLE */
```

```
#include "tasy0def.h"          /* 类型定义 */
```

```
#include "grus0255.h"          /* C 函数源文件头 */
```

```
/* C 主服务: 计算加法 */
```

```
void USP_sample (str_arg *arg)
{
    SUM = PAR1 + PAR2;
}
```

```
/* 与 ISaGRAF 内核动态链接的声明服务 */
```

```
UFP uspdef_sample (char *name)
{
    strcpy (name, "SAMPLE");
    return (USP_sample);
}
```

```
/* 文件结束 */
```

#### C.7.4 "C" 功能块

"C"功能块关联操作和静态数据。因为它可以处理静态对象，所以完善了 "C" 函数。它们一般用于提高 ST 和 FBD 语言的标准功能。与处理值的函数不

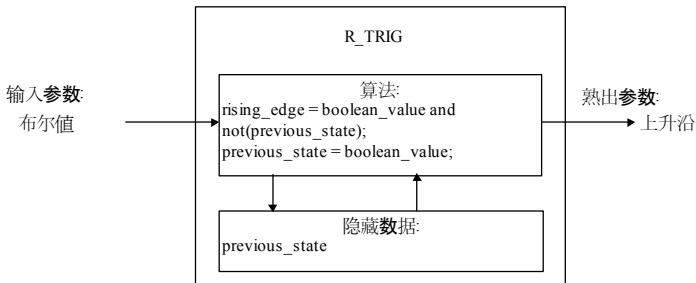
同, 功能块能处理静态数据。这意味着功能块算法可以管理数据变量 (连续处理)。

用 "C" 语言写的功能块, 编译, 与 ISaGRAF 内核链接。在 ISaGRAF 项目中使用新的功能块之前, 必须将增加了的内核安装到 ISaGRAF 目标 PLC。新的功能块不能集成到仿真器, 在使用非标准的功能块之前, 先仿真 ISaGRAF 应用程序。

**警告:**调用功能块是同步操作, 在应用程序循环期间, 由 ISaGRAF 核激活。用于功能块激活和读服务的时间包括在 ISaGRAF 应用循环周期内。用户必须保证不能将“等待”语句编入功能块中, 以使 ISaGRAF 循环周期超过最大允许。

## 二 声明功能块实例

功能块是操作和静态数据的组合对象。下面是一个 "R\_TRIG" 功能块的示例, 用于检测布尔表达式的上升沿。这里是块的功能描述:



隐含的静态变量 "previous\_state" 用于计算边沿。每次在程序中使用“功能块” TRIG”。时, 此变量必是不同的。在用 ST 语言中, 必须先字典中声明, 再被用于功能块的实例。因为功能块有内部隐含数据, 功能块的每一个

实例 ( 拷贝)用一个特定名称来识别。使用库管理器进行块类型的命名，使用字典编辑器进行实例的命名。

在 FBD 语言中使用功能块不用声明，因为 ISaGRAF FBD 编辑器自动声明，所用功能块的实例。由 FBD 编辑器自动声明。所用功能块的实例总是所编程程序局部的。

### ■ **将功能块加到ISaGRAF 库**

使用 ISaGRAF 库管理器将新的"C"功能块加到工作台的 ISaGRAF 库，当选择了“ C” 功能块库，用“ 文件” 菜单下的“ 新建” 命令完成。创建一个新的功能块后，必须写它的技术说明。新功能块的 "C" 源代码框架将由 ISaGRAF 库管理器自动生成。编辑菜单下的参数命令用于定义新功能块的调用和返回参数。

### ■ **在ISaGRAF 项目中使用“ C” 功能块**

任何集成的"C" 功能块都能用于 ISaGRAF 项目的程序中。"C" 功能块可由 ST 和 FBD 语言的程序调用。

在 ST 语言程序中调用功能块，应当遵从该语言的规则。块的调用参数写在功能名称的后面，在括号之间；由逗号分开。一个接一个的访问返回参数。每一个返回参数由名称表示，它由块实例的名称和参数名称组和而成名称的，组成部分由点号分开。例如: 名称

FBINSTNAME.parname

用来表示返回参数名为"parname", 功能块实例名为 "FBINSTNAME"。

用在 ST 语言程序的功能块实例必须在字典声明。功能块的每个拷贝

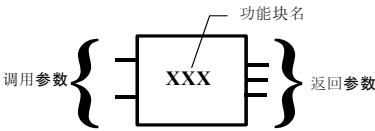
(实例。)必须有一个独特的识别名称，下面是实例在 ISaGRAF 字典声明的示例:

instance:	TRIG1	type:	R_TRIG
	TRIG2		R_TRIG

下面是在 ST 程序中使用声明了的实例的示例:

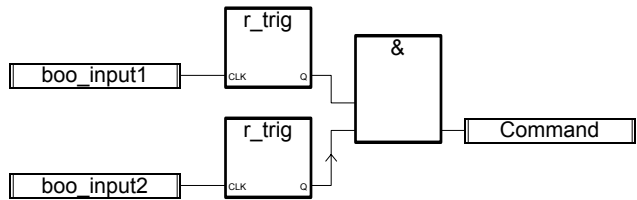
```
TRIG1 (boo_input1);  
TRIG2 (boo_input2);  
Command := (TRIG1.Q & TRIG2.Q);
```

FBD 程序能调用任何 "C" 功能块。功能块使用标准功能框。它的调用和返回参数分别连接在框的左右侧。下为一个功能框的标准格式：



在 FBD 语言程序的功能块不必声明，因为 ISaGRAF FBD 编辑器自动声明了所用块的实例。由 FBD 编辑器自动声明所用功能块的实例总是所编程序局部的。

下面是一个过去的示例 用 FBD 语言编程:



## 定义"C" 功能块的接口

文件菜单下的参数命令，用于定义新功能块的调用和返回参数。一个功能块可以有 32 个任意安排的调用或返回参数。不同于 "C"函数，一个功能块可以有多个返回参数。

窗口上方显示“ C” 功能块的参数，它基于功能调用原型的顺序：先是调用参数，然后是返回参数。窗口下方是：当前在表中选定的参数的详细描述：

- 参数名
- 参数的方向（调用/返回）
- 参数类型

任何 ISaGRAF 数据类型可以用于参数：布尔,整型模拟,实型模拟，计时器或信息。必须区分整型和实型模拟。。下为 ISaGRAF 类型和"C" 类型之间的关系：

BOOLEAN	unsigned long	无符号的 32 位字: 1=真 / 0=假
ANALOG	long	带符号的 32 位整型字:
REAL	float	单精度浮点数
TIMER	Unsigned long	无符号的 32 位整型字: (毫秒为单位)
MESSAGE	char *	字符串。

当一个信息值传送到 "C" 函数, 它不能包含空字符。字符串传送到 "C" 代码是以空字符结束的。必须记住，返回参数位于表的末尾。给参数命名时，遵守下列规则：

- 名称不能超过 16 个字符

- 首字符必须是字母
- 随后字符必须是字母，数字或 '\_' 符号
- 名称不分大小写

一个功能块的参数不能重名，调用参数名称不能同于返回参数。同一名称可以用于不同的功能块的参数。使用的参数名称应与源代码参数一致。

“插入”命令用于在当前选中参数之前，插入一个新参数。“删除”命令用来删除当前选中的参数。“重排”命令用来将列表中的参数重新排序，返回参数置于列表的最后。按下“确认”按钮，存储功能块接口的定义，并且关闭对话框。按下“取消”按钮，关闭对话框，但不保存功能块接口定义的变动。

### 功能块"C"接口

函数的接口取决于它所定义的参数。调用和返回参数通过结构传递。结构在“GRFB0nnn.H”文件中定义，其中 NNN 是功能块在 ISaGRAF 库中的逻辑号。返回参数由一个定义 GRFB0nnn.H 文件的逻辑号表示，这是一个“C”语言接口

的示例，用于 "LIM\_ALARM" 功能块（在极限时报警）：

```
/* 功能块接口 – 名称: sample */
```

```
/* 标准 ISaGRAF 数据类型 */
```

```
typedef long T_BOO;  
typedef long T_ANA;  
typedef float T_REAL;  
typedef long T_TMR;
```

```
typedef char *T_MSG;
```

```
/* 调用的参数结构 */
```

```
typedef struct {
    /* CALL */      T_BOO _par1;
    /* CALL */      T_BOO _par2;
} str_arg;
```

```
/* 访问 str_arg structure 域*/
```

```
#define PAR1 (arg->_par1)
#define PAR2 (arg->_par2)
```

```
/* 返回参数逻辑号 */
```

```
#define FBLPNO_Q1 0
#define FBLPNO_Q2 1
```

```
/* 文件尾 */
```

下面显示 ISaGRAF 类型和"C"类型的关系。在函数的定义文件中 ISaGRAF 类型被定义为 "C" 类型。

布尔	T_BOO	long (32 bits)
模拟	T_ANA	long

实型	T_REAL	float (32 bits - single precision)
计数器	T_TMR	long
信息	T_MSG	char * (32 bits - char pointer)

在 "str\_arg" 结构的每一个域对应一个功能块的调用参数。在结构中的参数出现的顺序与功能块定义时一样。定义为大写的标识符，可以直接访问结构参数，此参数 传送到功能块激活服务的 "C" 实现。用 ISaGRAF 库管理器定义功能块时，一次输入标识名称。返回参数的编号顺序与功能块定义时的相同，返回参数的第一个逻辑号总是 0。

定义的标识符不应使用数字值，而采用 "C"源码程序的返回参数表示。 这样可以保证：当修改接口定义之后，可以容易的重编译源文件。

使用 ISaGRAF 库管理器，每次变更了功能块的接口后，“ C” 定义文件都要更新。这保证了功能块的实现和它在 ISaGRAF 应用程序中使用的相关一致。

## 源代码

功能块的"C"语言实现分为三个不同的入口：

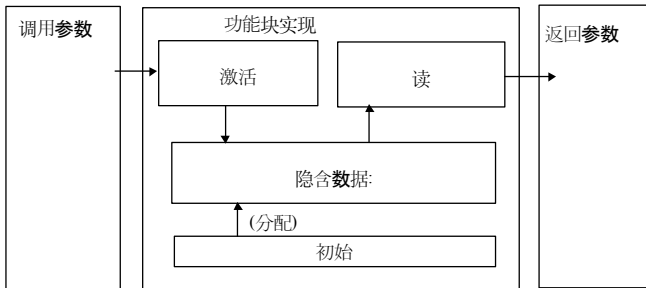
- 初始化服务
- 激活 服务 – 调用参数的处理
- 读返回参数服务

同一功能块的每个实例使用同一代码, 并且不能复制。每个实例关联一个静态数据。此数据不能由 ISaGRAF 程序直接访问,并且包含功能块实例的隐含变量。

每个块的每个实例，在每个目标循环都调用一次激活服务。它处理调用参数，更新关联数据，代表了功能块的主算法。

ISaGRAF 内核调用读服务，读取一个实例的返回参数的当前值，在此服务中不进行特别的计算。它仅进行隐含数据和 ISaGRAF 应用程序间的传送。

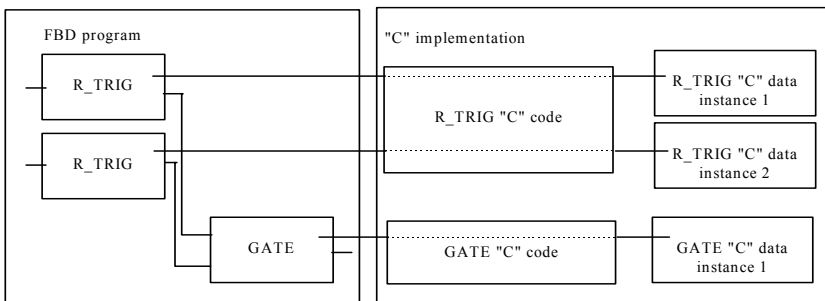
功能图：



#### ● 功能块的静态数据

功能块关联操作和静态数据。同一功能块的每个实例关联一个数据结构每次 ST 或 FBD 程序中的功能块中，对应于一个实例有一个数据结构。下面的一个功能块实例在 FBD 程序中对对应于一个“C”数据结构。

示例：



由 ISaGRAF 系统分配每个实例所需的数据结构内存,当应用程序启动时，关联实例数据结构的指针传送到“激活”和“读”服务。

ISaGRAF 库管理器自动生成定义数据结构的"C"源码框架。数据结构类型被称为 "str\_data"。程序员不应改变此名称，以保证与服务头的兼容。隐含数据一般由内部变量和返回参数的映像组成。“读”服务功能块仅用于访问返回参数,不应用于执行其它操作。

### ● 初始化服务

当应用程序启动时，由 ISaGRAF 内核调用功能块的初始化服务。它允许程序员询问系统，为实例分配内存。下为一个初始化服务的标准程序：

```
uint16 FBINIT_xxx (uint16 hinstance)
/* "xxx" is the name of the f. block */
{
    return (sizeof (str_data));
}
```

"hinstance" 参数是实例的逻辑号。它被保留为 ISaGRAF 内部操作,不应用于服务的编程。初始化服务返回一个实例的数据所需要的内存字节数量。所需内存总字节 (返回值) 不能超过 64 Kbytes。此服务不应用于其它操作。当功能块创建时，由 ISaGRAF 库管理器自动生成此服务的“C”源代码。

### ● 激活服务

为每一个在应用程序中使用的功能块实例，每次目标循环调用激活服务，此服务处理调用参数，且运行主功能块算法，更新隐含静态数据和返回参数。下为激活服务的标准框架：

```
void FBACT_xxx (
uint16 hinstance,          /* "xxx"是功能块名*/
```

```

                                /* 实例的逻辑号 */
str_data *data,                /* 数据: 指向实例数据结构 */
str_arg *arg                   /* 指向调用参数结构 */
)
{
}

```

hinstance" 参数是实例的逻辑号。它被保留为 ISaGRAF 内部操作, 不应用于服务的编程。"data" 参数是一个长指针指向相关实例的数据结构。"arg" 参数是一个长指针指向包括调用参数值的结构。程序员可以使用定义在功能块 "C" 头的标识符, 访问 "arg" 结构的域。

激活算法处理调用参数 (存在 "arg" 结构), 并且更新 "data" 结构的域。下例显示 TRIG (上升沿检测) 功能块的激活服务:

```

/* 在功能块 "C" 头存储定义 */

typedef struct {                /* 调用参数 */
    T_BOO _clk;                 /* 触发输入 */
} str_arg;

#define CLK    (arg->_clk)

/* 功能块实例数据结构 */

typedef struct {
    T_BOO prev_state;           /* 触发输入在过去状态 */

```

```
T_BOO edge_detect;    /* 边沿值: 返回参数映像 */
} str_data;

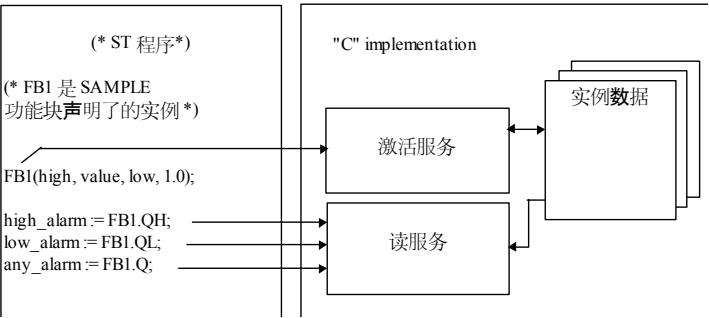
/* 激活服务*/

void FBACT_trig (uint16 hinstance, str_data *data, str_arg *arg)
{
    data->edge_detect = (T_BOO)(CLK && !data->prev_state);
    data->prev_state = CLK; /* 调用参数 */
}
```

当功能块创建时，此服务的 "C" 源代码框架将由 ISaGRAF 库管理器自动生成。

• 读返回参数

在 ST 或 FBD 程序中。每次引用功能块实例的返回参数时都调用读服务。它用于获取一个返回参数值。下例显示：在 ST 程序运行时调用读服务：



因为读服务可以在同一循环中多于一次调用, 对于同一返回参数或同一功能块实例, 在此服务中, 不进行特别计算, 仅进行隐含数据和 ISaGRAF 应用程序间的传送操作。下一个标准读服务的框架:

```
/* cast 操作用于拷贝返回参数的值 */
```

```
#define BOO_VALUE      ((T_BOO *)value)
#define ANA_VALUE      ((T_ANA *)value)
#define REAL_VALUE     ((T_REAL *)value)
#define TMR_VALUE      ((T_TMR *)value)
#define MSG_VALUE      ((T_MSG *)value)
```

```
/* 返回参数读服务: 每个返回参数调用*/
```

```
void FBREAD_xxx (          /* "xxx" 是功能块名*/
uint16 hinstance,         /* 实例的逻辑号 */
str_data *data,           /* 指向实例数据结构指针 */
uint16 parno,             /* 读参数的逻辑号 */
void *value)              /* 拷贝参数值的寄存器 */
{
    switch (parno) {
        case FBLPNO_XX: /* ... */ break;
        case FBLPNO_YY: /* ... */ break;
        /* .... */
    }
}
```

hinstance" 参数是实例的逻辑号。它被保留为 ISaGRAF 内部操作,不应用于服务的编程。"data" 参数是一个长指针指向相关实例的数据结构。

"parno" 参数是返回参数的逻辑号，需要此值。使用在功能块“ C” 头文件中定义的标识号，来标识返回参数。这类标识号以"FBLPNO\_" 为前缀，"value" 参数是一个长指针，指向放置可访问的返回参数当前值的寄存器。此参数指向的数据类型，取决于 ISaGRAF 返回参数的类型。下列表格给出了 ISaGRAF 类型和寄存器 "C" 数据类型:

布尔	long	32 bit 无符号字 (0=false / 1=true)
模拟	long	32 bit 有符号字
实数	float	32 bit 单精度浮点数
计时器	long	32 bit 无符号字 (1ms 为单位)
信息	char *	字符数组

下列宏用于访问拷贝寄存器，，按照访问返回参数的类型:

```
#define BOO_VALUE      ((T_BOO *)value)
#define ANA_VALUE      ((T_ANA *)value)
#define REAL_VALUE     ((T_REAL *)value)
#define TMR_VALUE      ((T_TMR *)value)
#define MSG_VALUE      ((T_MSG *)value)
```

这是一般使用的编程操作，用于将参数值拷贝到 ISaGRAF 寄存器

```
/* 布尔型参数: */
*BOO_VALUE = parameter_value;
```

---

```
/* 整型模拟 型参数: */
```

```
    *ANA_VALUE = parameter_value;
```

```
/* 实整 型参数: */
```

```
    *REAL_VALUE = parameter_value;
```

```
/* 计时器 型参数: */
```

```
    *TMR_VALUE = parameter_value;
```

```
/* 字符串 型参数: */
```

```
    strcpy (*MSG_VALUE, parameter_value);
```

当功能块创建时，ISaGRAF 库管理器自动生成此服务的"C"源代码

- "C" 源文件的例子

下面是 "C"功能块实现的标准框架

```
/* 功能块 (xxx 是功能块名称) */
```

```
#include <tasy0def.h>
```

```
#include <grfb0nnn.h> /* nnn 是功能块在库中的号 */
```

```
/* 每个块实例的隐藏数据结构/
```

```
typedef struct {
```

```
    /* 域定义 */
```

```
} str_data;
```

```
/* 初始化服务: 返回所需要的隐藏数据 */
```

```
word FBINIT_xxx (uint16 hinstance)
```

```
{
```

```
    return (sizeof (str_data));
```

```
}
```

```
/*激活服务: 处理调用的参数 */
```

```
void FBACT_xxx (uint16 hinstance, str_data *data, str_arg *arg)
{
    /* ... */
}
```

```
/* cast 操作 , 用于拷备返回参数的值 */
```

```
#define BOO_VALUE ((T_BOO *)value)
#define ANA_VALUE ((T_ANA *)value)
#define REAL_VALUE ((T_REAL *)value)
#define TMR_VALUE ((T_TMR *)value)
#define MSG_VALUE ((T_MSG *)value)
```

```
/* 读返回参数服务: 被每个返回参数调用 */
```

```
void FBREAD_xxx (uint16 hinstance, str_data *data, uint16 parno, void
*value)
{
    switch(parno)
    {
        case FBLPNO_XX: *???_VALUE = ...; break;
        case FBLPNO_YY: *???_VALUE = ...; break;
        ....
    }
}
```

/\*下列函数用于初始化功能块和它的实现声明。使用功能块名，完成了与 ISaGRAF 内核的链接，此服务由 ISaGRAF 库管理器完全生成。\*/

```
ABP fbldf_xxx (char *name, IBP *initproc, RBP *readproc)
{
    strcpy (name, "XXX");
    *initproc = (IBP)FBINIT_xxx;
    *readproc = (RBP)FBREAD_xxx;
    return ((ABP)FBACT_xxx);
}

/* 文件结束 */
```

包括来自 ISaGRAF 内核文件的"TASK0DEF.H"，用于系统相关的定义。它也包括数据类型定义指向实现服务的长指针。

## == "C" 实现和项目的链接

使用函数的名称，完成 "C" 功能块的实现和它在 ISaGRAF 项目的程序中使用的逻辑链接。声明服务加到功能块的 "C" 源代码。当应用程序启动时，调用此服务一次，专用于 ISaGRAF 内核的 "C"功能块名对应于实现服务。这是声明服务的标准格式：

```
ABP fbldf_xxx (char *name, IBP *initproc, RBP *readproc)
{
    strcpy (name, "XXX");          /* 功能块的名称*/
    *initproc = (IBP)FBINIT_xxx;   /* 初始化 服务*/
```

```
*readproc = (RBP)FBREAD_xxx;          /* 读服务 */
return ((ABP)FBACT_xxx);                /* 激活服务 */
}
/* xxx 是功能块的名称 */
```

用于 strcpy 语句的功能块名称必须大写。实现和声明服务的名称必须小写。使用 "FBACT\_", "FBINIT\_", "FBREAD\_" 和 "fbldf\_" 为前缀，用于实现和定义服务。可以使用户使用"C" 语言的保留关键字，或者使用 "C" ISaGRAF 库中的已经存在的函数名为功能块名。其它的语句不用加到声明服务。

任何集成的"C" 功能块调用声明服务，即使它未被 ISaGRAF 应用程序使用。如果在应用程序使用"C" 功能块还未集成到内核，ISaGRAF 内核将检测到一个致命的错误。

在链接新的功能块和内核之前，用户必须写一个另外的 "C"源文件,名为 "GRFB0LIB.C", 将它插入到链接文件列表内(与保留的功能块)。  
"GRFB0LIB.C" 仅包括声明服务的数组。在应用程序启动时，读入此数组，用于与"C" 语言写的功能块进行动态链接。下面是文件的例子：

```
/* File: grfb0lib.c – 实现功能块 */
#include <tasy0def.h>

extern ABP fbldef_fb1(char *name, IBP *init, RBP *read);
extern ABP fbldef_fb2(char *name, IBP *init, RBP *read);

FBL_LIST FBLDEF[ ] = {
```

```
    fblddef_fb1,  
    fblddef_fb2,  
    NULL };  
/* 文件结束 */
```

FBLDEF 数组用 NUll 指针结束。如果条件不相符，可能产生一些冲突。如果没有定义 FBLDEF 数组，当链接 ISaGRAF 核心时，将产生“未定义”错误。

当写此文件时，可构成一个新的核心，包括所有存在的功能块。当仅将用于此项目的功能块插入 FBLDEF 数组时，专用于这个项目的核心将能构成。当一个应用程序代码已生成，ISaGRAF 代码生成器自动产生“GRFBOLIB.C”文件。此文件放置在 ISaGRAF 项目目录，仅由用于此项目的功能块组成。

## 限制

ISaGRAF 库可以包含至多 255 个“C”功能块。在一个函数当中可以处理任何类型操作。同一项目中任何类型的功能块可被复制(实例)至多 255 次。应当记住，在一个 ISaGRAF 循环中，调用功能块是同步的，这样，执行处理功能块将直接影响到循环耗时。

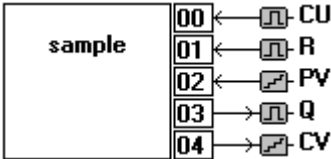
## 完整的示例

下面是使用功能块编程的完整 "sample" 示例,一个上升计数器。

下为功能块的技术说明:

名称:	SAMPLE
描述:	上升计数器
创建日期:	01 February 1994
作者:	ICS Triplex ISaGRAF Inc.
调用:	CU : 输入 R : 复 位 PV : 编程的最大值
返回:	Q : 检测到最大值 CV : 当前计数结果
示例:	<pre>SAMPLE ( count, reset_command, maximum_value); max_detect := SAMPLE.Q; count_result := SAMPLE.CV;</pre>

下为功能块的接 口:



下为功能块的 "C" 源代码的头:

```
/* 功能块的接 口: -名称 : SAMPLE */
```

---

```
/* 定义标准 ISaGRAF 数据类型 */
```

```
typedef long T_BOO;  
typedef long T_ANA;  
typedef float T_REAL;  
typedef long T_TMR;  
typedef char *T_MSG;
```

```
/* 定义调用参数结构 */
```

```
typedef struct {  
    T_BOO _cu;  
    T_BOO _r;  
    T_ANA _pv;  
} str_arg;
```

```
/* 访问调用参数的标识*/
```

```
#define CU                (arg->_cu)  
#define R                 (arg->_r)  
#define PV                (arg->_pv)
```

```
/* 返回参数的逻辑号 */
```

```
#define FBLPNO_Q          0  
#define FBLPNO_CV        1
```

/\* 文件结束 \*/

下为功能块的 "C" 源代码。程序员仅需手工输入黑体字行。

/\* 功能块 - 名称: SAMPLE \*/

#include <tasy0def.h>                   /\* 定义数据类型 \*/

#include <grfb0255.h>               /\* 功能块的 "C" 源代码的头: \*/

/\* 定义包括实例数据的结构 \*/

```
typedef struct {  
    T_BOO overflow;           /* 真: 计数结果 >= 编程的最大值*/  
    T_ANA value;             /* 当前计数结果 */  
} str_data;
```

/\* 初始化:请求实例数据内存 \*/

```
word FBINIT_sample (uint16 hinstance)  
{  
    return (sizeof (str_data));  
}
```

/\*激活: 上升计数器算法 \*/

```
void FBACT_sample (uint16 hinstance, str_data *data, str_arg *arg)  
{
```

```

    if (R) data->value = 0;
    else if (CU && data->value < PV) (data->value)++;
    data->overflow = (data->value >= PV) ? (T_BOO)1 : (T_BOO)0;
}

```

/\* cast 操作需要拷贝参数 到 ISaGRAF 寄存器 \*/

```

#define BOO_VALUE ((T_BOO *)value)
#define ANA_VALUE ((T_ANA *)value)
#define REAL_VALUE ((T_REAL *)value)
#define TMR_VALUE ((T_TMR *)value)
#define MSG_VALUE ((T_MSG *)value)

```

/\* 读： 从返回参数中取值 \*/

```

void FBREAD_sample (uint16 hinstance, str_data *data, uint16 parno, void
*value)
{
    switch (parno) {
        case FBLPNO_Q : *BOO_VALUE = data->overflow; break;
        case FBLPNO_CV : *ANA_VALUE = data->value; break;
    }
}

```

/\* 与 ISaGRAF 内核动态链接，使用声明服务 \*/

```

ABP fbldef_sample (char *name, IBP *initproc, RBP *readproc)

```

```
{
    strcpy (name, "SAMPLE");
    *initproc = (IBP)FBINIT_sample;
    *readproc = (RBP)FBREAD_sample;
    return ((ABP)FBACT_sample);
}
```

/\* 文件结束 \*/

## C.7.5 编译和链接技术

ISaGRAF 工作台不包括任何"C"编译或链接器。尽管如此,本章介绍了轻松使用 ISaGRAF 库管理器创建的文件的主要技术,以及传递它到其它编译或链接器工具。

### “C” 源文件

变换函数” , “ 函数” , 或功能块的 "C" 源文件被 ISaGRAF 库管理器放在 ISAWIN\LIB\DEFS 和 ISAWIN\LIB\SRC 目录。源文件的名称中包括在 ISaGRAF 库中变换函数” , “ 函数” , 或功能块的号码。下面是文件名:

\isawin\lib\defs\TACN0DEF.H	任何变换函数的定义文件
\isawin\lib\src\GRCN0nnn.H	变换函数的源文件
\isawin\lib\defs\GRUS0nnn.H	函数的定义文件
\isawin\lib\src\GRUS0nnn.C	函数的源文件
\isawin\lib\defs\GRFB0nnn.H	功能块的定义文件
\isawin\lib\src\GRFB0nnn.C	功能块的源文件

(nnn 是变换” , “ 函数” , 或功能块的号码)

注意：

当更名或复制库元素时，它的文本和编程行在 ISaGRAF 库管理器中，按照新元素名和逻辑名，并不更新。必须在“ C” 源文件中手工更新。

ISAWIN\LIB\USPNUMS 文件对存在于 ISaGRAF 库的“ C” 函数，给出了名称和逻辑号之间的联系。下面为一文件示例：

```
1    funct_A
10   funct_B
16   funct_C
```

\ISAWIN\LIB\FBLNUMS 文件对存在于 ISaGRAF 库的“ C” 功能块，给出了名称和逻辑号之间的联系。下面为一文件示例：

```
0    fbl_A
1    fbl_B
2    fbl_C
```

\ISAWIN\LIB\CNVNUMS 文件对存在于 ISaGRAF 库的“ C” 变换 函数 ，给出了名称和逻辑号之间的联系。此文件的内容是标准库的变换。

```
0    SCALE
1    BCD
```

每次“ 变换函数” ，“ 函数” ，或功能块，建立，改名，复制或删除时，这些文件在 ISaGRAF 库管理器中自动更新。当生成一个应用程序时，ISaGRAF 代码生成器自动生成下列文件：

\\isawin\\apl\\ppp\\GRCN0LIB.C	被声明为使用在项目中的所有变换函数功能的数组
\\isawin\\apl\\ppp\\GRUS0LIB.C	被声明为使用在项目中的所有函数功能的数组
\\isawin\\apl\\ppp\\GRFB0LIB.C	被声明为使用在项目中的所有功能块功能的数组
.	

(ppp 是 ISaGRAF 项目名称)

在链接操作时，这些文件用于建立专用项目的新 ISaGRAF 核心，它仅包括用于此项目的“变换”，“函数”和功能块。

#### 下载源文件至本地系统

如果 ISaGRAF 目标系统支持本地编译工具，在 ISaGRAF 库管理器创建的“C”源文件和定义文件可以下载至目标 ISaGRAF 系统，可以使用带视窗的标准 TERMINAL ( Windows 所提供的终端仿真 ) 工具。

当源文件被管理在目标系统中时，使用 ISaGRAF 库管理器，每次修订功能接口时，定义文件必须用一个下载操作而更新。

下载文件的命令行可以合并为批处理文件，然后从工作台“工具”菜单启动(见用户指南“程序管理”一章)

## 使用交叉编译器

可以直接在你的 PC 上管理源文件，如果目标是一个 PC，或是交叉编译器可以运行在 PC 上，并且产生用于目标系统的代码。

在这种情况下，用户可以运行 ISaGRAF 库管理器去完善和修改“变换函数”，“函数”和功能块的源码。

运行编译器和链接的命令行可以合并为批处理文件，然后从工作台“工具”菜单启动(见用户指南“程序管理”一章)

当在 PC 上已编译了“变换”功能，“函数”和功能块，用户在运行应用程序之前，将新产生的 ISaGRAF 核心(与新的组件相链接)下载至目标系统。如果目标是另一台 PC，新产生的 ISaGRAF 核心可以通过软磁盘或通过网络，装载入目标机器。

## 与 ISaGRAF 核心库链接

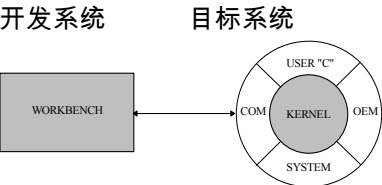
**警告：**以下为可能与你的目标系统不是完全对应的一般信息。在此情况下，你可以参阅目标盘上提供的 README 自述文件和.TXT 文本文件

ISaGRAF 目标盘上包括很多实用工具文件，用于编译和链接 ISaGRAF 核心库与“变换”，“函数”和功能块。

存在二种实现：

- 单任务 ISaGRAF：在同一程序中执行所有功能。
- 多任务 ISaGRAF：一个分别的任务(或进程)是通讯

在两种情况下，“C”组件都合并在同一库中：对程序员来说单任务或多任务是无区别的，对于单任务版，用户“C”库与单任务链接，(通常被称为 ISA)。对多任务版：“库”与“内核任务”相链接(通常被称为 ISAKER)



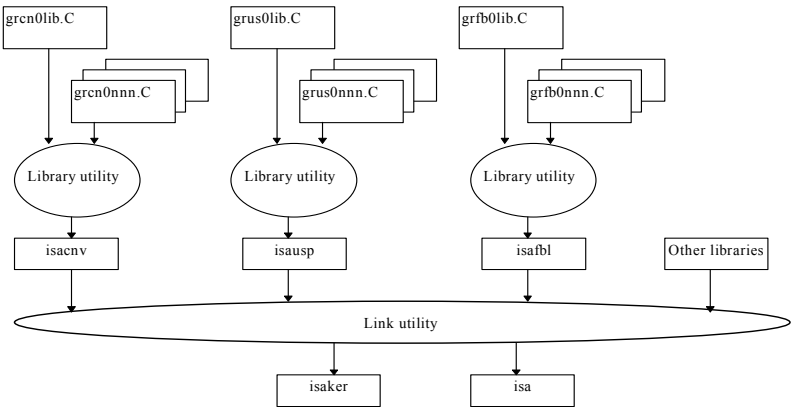
ISaGRAF 软件的内部是与硬件无关的。它执行 IEC 语言并具有自己的变量数据库。

建立与内核链接的第一步是建立所有“变换函数”，“函数”和功能块的库，它用于专用项目。

库	内容
ISAUSP	- GRUS0LIB 对象文件 (所申明函数的数组) - 每个集成函数的对象文件
ISAFBL	- GRFB0LIB 对象文件 (所申明函数的数组) - 每个集成功能块的对象文件

ISACNV	- GRCN0LIB 对象文件 (所申明函数的数组) - 每个集成“ 变换” 函数的对象文件
--------	---

程序员必须链接这些新库与其它对象文件和 ISaGRAF 内核的库。下列图表显示了，用户“ C” 开发集成的不同阶段。



这是对象模块的精确列表和在链接时必须链接的库

为了生成 isaker:

对象模块:                   tast0mai

对象模块:                   tats0com

内核库:                    isaker

内核库:                    isaoem

用户库:	isausp	用户自定义函数
用户库:	isafbl	用户自定义功能块
用户库:	isacnv	用户自定义变换函数功能
内核库:	isasy	
系统库:	(参见你的“ C” 编译器手册)	

为了生成 ISA:

对象模块:	tast0mai	
对象模块:	tast0com	
内核库:	isaker	
内核库:	isatst	
内核库:	isaoem	
用户库:	isausp	用户自定义函数
用户库:	isafbl	用户自定义功能块
用户库:	isacnv	用户自定义变换函数功能
内核库:	isasy	
系统库:	(参见你的“ C” 编译器手册)	

程序员必须遵守上图所示的库和对象模块的严格规则。按照目标系统，对象模块和库标准的后缀(".lib", ".obj", ".l", ".r"...)

## 编译和链接选择项

可以在编译和链接时选择方便选项.他们取决于处理“变换函数”，“函数”和功能块时操作的类型.一些操作在链接时，需要其它的系统库(数学，图形...)ISaGRAF 内核的所有“C”源文件是采用大(LARGE)内存模式编译的.程序员也必须使用同种模式，编译“变换函数”，“函数”和功能块  
编译“C”库组件时，必须定义一个专用常数.它表示目标系统和处理器的类型。这样“变换函数”，“函数”和功能块的源码可以具有系统无关性。下为这些常数值的名称：

DOS .....基于 DOS 系统 (INTEL 处理器)  
ISAWNT .....基于 Windows-NT 系统 (INTEL 处理器)  
OS9 .....基于 OS9 系统 (MOTOROLA 处理器)  
VxWorks .....基于 VxWorks 系统 (MOTOROLA 处理器)

编译和链接实用工具指令文件，使用 ISaGRAF 处理器目标软件提供了怎样在编译器的指令行，定义方便常量值。

## 支持的编译器

下列编译器支持“变换函数”，“函数”，“功能块”以及与 ISaGRAF 内核链接的开发：

Microsoft MSC 7.00 compiler	基于 MS-DOS 的目标
-----------------------------	---------------

Microsoft MSVC 4.00 compiler	基于 Windows-NT 的目标
Microware ULTRA-C compiler	基于 OS-9 的目标
Tornado 1.0; GNU Toolkit 2.6	基于 VxWorks 的目标

使用其它编译器请与**ICS Triplex ISaGRAF Inc.**联系

☐ 总结：

当开发新的变换，函数或功能块时，必须完成的操作步骤汇总如下：

- ⇒1. 使用 ISaGRAF 库管理器，创建新元素：命名和注释文本  
“ C” 源文件的框架将自动生成.
- ⇒2. 如果元素是函数或是功能块，使用 ISaGRAF 库管理器描述其 接口[调用(call)和返回(return)参数]“ C” 源头文件 将自动生成.
- ⇒3. 使用 ISaGRAF 库管理器，输入元素的详细的技术说明文本(用户指南)
- ⇒4. 用 ISaGRAF 库管理器，输入“ 变换函数”，“ 函数” 或功能块的算法的 C 语言程序，完成“ C” 源文件。元件的源代码现已完备。可以使用其它编辑器。
- ⇒5. 选择库管理器的“ 显示逻辑号” 选项，可得知：哪些逻辑号附属于新元素。这些号码用于相应的“ C” 和“ H” 源文件的路径名称。

- ⇒6. 拷贝/下载“ C” 和“ H” 文件到你的目标(如果有本地编译器)或是到相应环境(如果是交叉编译器)。在哪里 ISaGRAF 目标库和任务已经安装完毕。
- ⇒7. 对新的源文件运行“ C” 编译器，修改可能的任何 语法错误。
- ⇒8. 将新元素的申明服务名称插入“ GR ? ? OLIB.C” 源文件， 定义所插入元素的数组。
- ⇒9. 运行“ C” 编译器，编译“ GR ? ? OLIB.C” 文件。
- ⇒10. 将对象模块的名称插入到对象文件列表，产生相应的库。
- ⇒11. 运行“ C” 库生成器，运行“ C” 链接器，生成新的内核。
- ⇒12. 安装新创建的内核到你的目标
- ⇒13. 编写一个 ISaGRAF 应用程序示例，用于检测新元素的接口和激活



## C.8 Modbus 总线连接

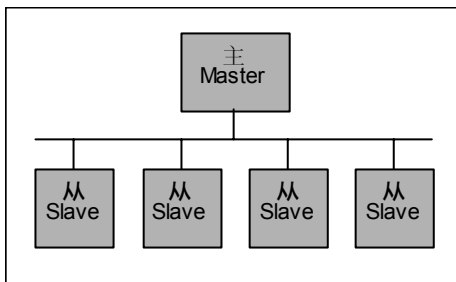
应用程序开发和测试完成后，你可将它接入“过程可视”系统。

ISaGRAF 是一个开放系统，提供了多样的联网能力。MODBUS 是采用 MODICON 公司标准通信协议的简单工业网络；可用于大多数过程可视系统，仅需要采用串行连接(RS232,RS485,电流环)。

ISaGRAF 通信调试协议与 MODBUS 兼容，可由 MODBUS 主站读写访问变量。

### C.8.1 MODBUS 网络和通信协议

MODBUS 网络仅支持单主站(通常为过程可视系统)，和一个或多个从站(多为 PLC)



当主站向一个从站(使用从站号)发送一个请求，在发送下一请求之前，等待从站回答，其它无关从站不用回答。

每一个框架包括：一个从站号，请求号，和相关数据，以及一个 16 位(bit)的“校验和”(csc)。

如果过了“超时”期还未收到“回答”，在主站申明从站为“断开”前，可重复发送一定次数请求。

在主站可以调整“超时”的值，以及“重发”次数，以满足从站的需求(取决于实际应用的情况)

在请求过程中出错了，从站发出“出错”信息，而不发送期待的回答框架包。

MODBUS 总线采用 MODICON 通信协议，不是国际标准。有很多‘MODBUS’兼容的通信协议，它们在下列几点有区别，例如：

- 实现功能码表
- 地址映射
- RTU(二进制码)或 ASCII 协议
- 其它...

### C.8.2 ISaGRAF 的实现过程

#### 访问应用程序变量

ISaGRAF 通信连接识别五种 MODBUS 功能码

1	读 N 位
3	读 N 字
5	写 1 位
6	写 1 字
16	写 N 字

如果变量已经在工作台字典中被定义，通过它们的网络地址，可以访问 ISaGRAF 应用变量。这些变量是：

- 布尔或模拟变量
- 输入，输出或内部变量
- 局部或全局变量

为了写一个布尔变量，可使用功能 5，6 或 16，写一个真值是所有非零值。

为了读一个布尔变量，可用功能 1 或 3。功能 1 以位形式，功能 3 以字节形式检索(真值对应于 0xFFFF)。

为了写一个模拟变量，可用功能 6 或 16。其值为 16 位整型，值域为(-32768 至+32767)(ISaGRAF 目标变量为 32 位)。

使用功能 3，读一个模拟变量。所读值为 16 位整型，值域为(-32768 至 +32767)。在目标侧，模拟量为 32 位，因此目标侧超出 16 位的(正或负)，将仅被以 16 位范围最大值(正或负)读出。

MODBUS 请求不能访问实型变量

警告：

ISaGRAF 执行过程不能管理像“unkown Modbus address” (不详的总线地址)这样的出错代码。

表示符号：

slv      从站号  
 nbw      “字”数量  
 nbb      “字节”数量  
 nbi      “位”数量  
 addH    网址(高字节)  
 addL    网址(低字节)  
 vH      值(高字节)  
 vL      值(低字节)  
 V      字节值  
 bfd      (位域)(nbb 字节)  
 crcH    校验和(高字节)  
 crcL    校验和(低字节)

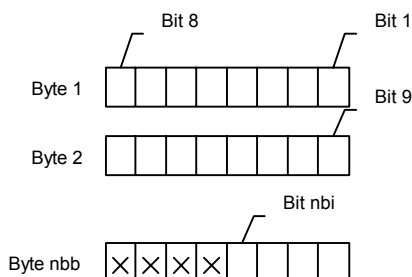
功能 1：读 N 位

从网址 ADDH/ADDL 起读 NBI 位

请求	slv	01	add H	add L	00	nbi	crcH	crcL
----	-----	----	----------	----------	----	-----	------	------

回答	slv	01	nbb	bfd	。。 .		crcH	crcL
----	-----	----	-----	-----	---------	--	------	------

bfd 是下列格式的一个位域



Bit 1 是在网址 ADDH/ADDL 处的变量值

Bit nb1 是在网址 ADDH/ADDL + nbi-1 处的变量

X 水定义的值.

功能 3：读 N 字

从网址 ADDH/ADDL 起读 nbw 字

请求	slv	03	addH	add L	00	nbw	crcH	crcL
----	-----	----	------	----------	----	-----	------	------

回答	slv	03	nbb	vH	vL	...	crcH	crcL
----	-----	----	-----	----	----	-----	------	------

nbb 对应于 VH，VL 字节数量

功能 5：写 1 位

在网址 ADDH/ADDL 处写一布尔位

请求	slv	05	addH	add L	vH	00	crcH	crcL
----	-----	----	------	----------	----	----	------	------

回答	slv	05	addH	add L	vH	00	crcH	crcL
----	-----	----	------	----------	----	----	------	------

功能 6：写 1“ 字”

在网址 ADDH/ADDL 处写一个“ 字”

请求	slv	06	addH	add L	vH	vL	crcH	crcL
----	-----	----	------	----------	----	----	------	------

回答	slv	06	addH	add L	vH	vL	crcH	crcL
----	-----	----	------	----------	----	----	------	------

功能 16：写 N“ 字”

从网址 ADDH/ADDL 起写 nbw 字 ( nbb=2nbw )

Questi on	slv	10	add H	add L	00	nbw	nbb	vH	vL	...	crc H	crc L
--------------	-----	----	----------	----------	----	-----	-----	----	----	-----	----------	----------

Answer	slv	10	add H	add L	00	nbw	crc H	crc L
--------	-----	----	----------	----------	----	-----	----------	----------

示例：

— 功能 1：在从站 1，从网址 0×1020 起读 15 位

Questi on	0	0	1	2	0	0	7	0
	1	1	0	0	0	F	9	4

Answer	0	0	0	0	1	3	F1
r	1	1	2	0	2	9	

在 0×0012 处读的值为 00000000 00010010

此例中仅变量 0×1029 和 0×102C 为真，其余为假。

- 功能 16：在从站 1，地址 0×2100 处写 2 个“字”，值为 0×1234 和 0×5678

Question	0	1	2	0	0	0	0	1	3	5	7	1	C
	1	0	1	0	0	2	4	2	4	6	8	C	A

Answer	0	1	2	0	0	0	4	F
r	1	0	1	0	0	2	B	4

□ 传送文件

与现代现场总线相比，如果不由专用制造商扩展功能码，MODBUS 通信协议将提供很差的维护性能。

在我们的情况下，ISaGRAF 基于功能强大和零活的计算机，MODBUS 协议有两条限制：

- 仅能访问 ISaGRAF 变量。
- 执行大容量数据快速传输有困难。

这就是为什么 ISaGRAF 提供了一组文件传送的类似 MODBUS 请求，或称为远程文件管理通信协议。它可以执行：

- 下载二进制或 ASCII 文件
- 上载二进制或 ASCII 文件

- 通过虚拟或物理共享文件，实现动态数据交换

借助 ISaGRAF 通信连接，所有与 ISaGRAF 独立的应用，可以方便的与远程目标相通讯。通信协议基于下列概念在 ISaGRAF 目标侧的文件称为“远程文件”

在主计算机的文件称为本地文件访问文件的每个字节，由 32 位的基址加上 16 位字节地址。有请求用于选择远程文件名，选择基址，使用 16 位字节地址去读或写远程文件的数据。

#### 功能 17：写数据

nbb 相当于 VH，VL 字节的数号

请求	slv	11	add H	add L	00	nbb	nbb	vH	vL	...	crc H	crc L
----	-----	----	----------	----------	----	-----	-----	----	----	-----	----------	----------

回答	slv	11	add H	add L	00	nbb	crc H	crc L
----	-----	----	----------	----------	----	-----	----------	----------

按地址范围 ADDH/ADDL 区别请求的意义。

- 0xF000:初始化远程文件名 nbb 代表文件名的字符数定义在 VH，VL 区域 (在此高和低无意义)，以及包括\O 用于字符串的结束。如果文件不存在，建立时加上可写“+”可读“+”“可执行”属性。
- 0xF002:改变特定值的基址，nbb 应当等于 4，第一个 VH/VL 字节代表特定值的高“字”，可访问所有 32 位值。所有今后的读或写请求将使用此基址，。当未使用此请求，默认的基址值是零。

- 0xF004:删除文件 nbb 应当等于零，如果文件存在和有可能，将删除此文件。

### 大于 0xF004：保留

- 小于 0xF000：写入字节

特定在 ADDH/ADDL 的地址处，写入字节。此地址小于 F000，特定的字节将被写入(nbb 字节确定在 VH, VL 范围，高和低不再有意义)以由左至右的顺序，相对于以前所选择的远程文件名。写入的起始地址，等于特定地址加上过去所选择的基址。如果访问地址结果超出当前文件大小，文件将扩展，你不能缩小文件的大小。

### 功能 18：读数据

请求	slv	12	add	add	00	nbb	crc	crc
			H	L			H	L

回答	slv	12	nbb	V	V	...	crc	crc
							H	L

在 ADDH/ADDL 定义读入字节的地址。它必须小于 F000，从过去所选择的远程文件名，读入 nbb 个字节，从特定地址开始(任何 16 位值的 ADDH/ADDL)，加上过去所选择的基地址。

以(以 V 区域从左至右)的顺序在文件中读出的顺序检索这些值。

示例：

选择远程文件名：' target.fil'

请求	01	11	F0	00	00	0B	0B	74	...	00	25	9F
----	----	----	----	----	----	----	----	----	-----	----	----	----

回答	01	11	F0	00	00	0B	8F	0E
----	----	----	----	----	----	----	----	----

选择基址：0x10000.

请求	01	11	F0	02	00	04	04	00	01	00	00	76	11
----	----	----	----	----	----	----	----	----	----	----	----	----	----

回答	01	11	F0	02	00	04	6E	CA
----	----	----	----	----	----	----	----	----

写入 4 字节：绝对地址 0x107D0, 值 01,02,03,04.

请求	01	11	07	D0	00	04	04	01	02	03	04	28	6F
----	----	----	----	----	----	----	----	----	----	----	----	----	----

回答	01	11	07	D0	00	04	FC	87
----	----	----	----	----	----	----	----	----

读出 4 字节：绝对地址 0x107D0.

请求	01	12	07	D0	00	04	B8	87
----	----	----	----	----	----	----	----	----

回答	01	12	04	01	02	03	04	58	7D
----	----	----	----	----	----	----	----	----	----

## C.9 “断电”管理

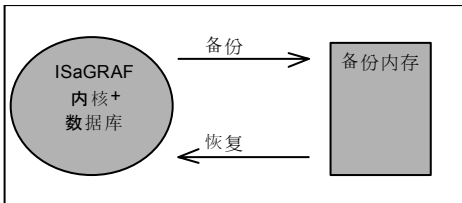
### C.9.1 基础

在应用程序当中，有些场合对断电的处理措施是至关重要的，有如下三个原因：

- 取决于特殊过程要求
- 取决于硬件的性能
- 取决于编程方法

当然 ISaGRAF 对“断电管理”问题并不能提供一个完全通用的方法而只是一系列原则，方法和工具，将其组合在一起，可对每个实际应用场合提供一个专门的处理，至少在硬件方面。

为了使过程控制系统在断电后，能正确的再启动，有三个问题必须解决：



- 进行数据备份
- 当启动时，必须探明在这之前曾经发生“断电”
- 恢复备份了数据

第二个问题不可能有标准的软件方案，但系统供货者可以提供必要的工具，从 ISaGRAF 应用程序或 C 语言程序去访问，获得硬件状态。

进一步，重要的是决定：哪些数据必须被保存和检索，我们定义了二类数据

– 应用变量

过程变量：例如，处理的条目号，发生断电的日期，应用参数的值，等...

像计数器，计时器，中间变量值和标志这样的程序变量。

– 程序状态，

例如，当前激活步号，每个 C 程序状态，等...

下章由二个实例分析，可以了解 ISaGRAF 如何解决问题的。

## C.9.2 备份“应用变量”

### ■ 可保持的变量

工作台变量编辑器提供了对每个内部变量(非 I/O 变量)选择“保持”属性的能力。

每一次循环周期结束时，保留变量的值被复制到特定内存区，此内存区通常为有后备电池 RAM，每次启动时，如果至少有一个变量具有“保持”属性，ISaGRAF 将找寻这个保持变量：

- 如果同一应用程序在此之前运行过，ISaGRAF 查到“储存”数据，并将其赋给每个保持数据。
- 如果之前运行的是另一应用程序，或在此之前没有运行过程序，ISaGRAF 将认为保持变量为无效，将其值复位为零。

在工作台对不同类型变量设定不同的内存区，在“制作”菜单：应用程序选项；保持变量。特定字符串有如下格式：

boo_add , boo_size , ana_add , ana_size , tmr_add , tmr_size , msg_add , msg_size
--

boo\_add: 用于存放布尔变量的 16 进制地址，不能为零。

boo\_size : 地址上的以字节为单位的 16 进制尺寸，每个布尔变量的  
存储需要一个字节。

ana\_add: 用于存放模拟变量的 16 进制地址，不能为零。

ana\_size: 此地址上的以字节为单位的 16 进制尺寸。至少四个字节，  
然后加上存储每个模拟变量的 4 个字节

tmr\_add: 用于存储计时器变量的 16 进制地址，不能为零，

tmr\_size: 每个计时器变量的存储需要 5 个字节。

msg\_add: 用于存储“ 信息” 变量的 16 进制地址，不能为零。

msg\_size: 每个信息变量的存储需要 256 字节。

要求：

- 即使你不需要对变量的所有类型进行备份，还是需要设定所有类型的所有的域，在此情况下对于不要求的变量类型你只要将其尺寸定为零[除了模拟量(你需要 4 个字节)以及每个地址不能为零]。

示例：

假设需要备份

20 布尔变量

0 模拟变量

0 计时器变量

3 信息变量

备份用内存区位于 16 进制地址 0xA2F200

设想：

布尔变量存储在地址 0xA2F200，以及 20 字节的内存区。

模拟量至少需要 4 字节，将存储于地址 0xA2F214。

计时器的名义地址为 0xA2F200，尺寸为零

信息变量将存储在地址 0xA2F218，将需要内存区 256×3 字节

在工作台输入的字符串为：

A2F200,14,A2F214,4,A2F200,0,A2F218,300
--

## 系统功能调用

如果大多数应用变量需要存储，将使用系统功能特性来处理完整一系列变量。(关于系统功能的更多信息请参见用户指南) 提示：备份和恢复将在应用程序层由程序员管理。

首先，你要确定备份内存区是对某种类型变量，还是所有类型变量：

<new\_address> := SYSTEM(SYS\_INITxxx,<address>);

这里

–<地址>是备份内存地址(16#值使用 16 进制格式)，必须是偶地址，否则操作失败。

– SYS\_INITxxx 能是:

- \*SYS\_INITBOO      确定用于布尔变量的备份内存位置
- \*SYS\_INITANA      确定用于模拟变量的备份内存位置
- \*SYS\_INITTMR      确定用于计时器变量的备份内存位置
- \*SYS\_INITALL      所有布尔，模拟，计时器内存位置

– <new\_address> 获得下一个自由地址，按照 SYS\_INITXXX 以字  
为单位，<地址> + 备份变量尺寸

这可以检查备份所需的内存区尺寸。如果操作失败，<NEW - ADDRESS>  
变为零

这时你可以进行备份了，可在应用程序的任意时刻调用此过程，在当前循环结束时执行备份。仅进行一次。如果当断电发生时硬件可提供一个输入或一个“C”函数，通知用户或者至少在系统崩溃之前一个周期延时，当断电被探测到，才可以进行备份。

<error> :=SYSTEM(SYS\_SAVxxx,0);

这里

– SYS\_SAVxxx 可以是:

- \*SYS\_SAVBOO      询问所有布尔变量备份
- \*SYS\_SAVANA      询问所有模拟变量备份
- \*SYS\_SAVTMR      询问所有计时器变量备份
- \*SYS\_SAVALL。      询问所有布尔，模拟和计时器变量备份

– <error>当操作失败后，获得非零的“出错”状态)。

最后，你可以恢复变量，可以在应用程序中，在任何时刻调用此过程。在当前循环结束时，完成恢复且仅做一次。为了保证数据备份有效，将模拟变量置为一个常量。

```
<error> := SYSTEM(SYS_RESTxxx,0);
```

这里

–SYS\_RESTxxx 可以是:

\*SYS\_RESTBOO 恢复所有布尔量。

\*SYS\_RESTANA 恢复所有模拟量。

\*SYS\_RESTTMR 恢复所有计时器量。

\*SYS\_RESTALL 恢复所有布尔，模拟和计时器变量量。

–<error> >当操作失败后，获得非零的“ 出错” 状态 )。(SYS\_INITxxx has not been done)。

下为用于管理备份变量的系统功能指令汇总:

指令		意义
予定义关键字	值	
SYS_INITBOO	16#20	初始化布尔量备份
SYS_SAVBOO	16#21	保存布尔量
SYS_RESTBOO	16#22	恢复布尔量
SYS_INITANA	16#24	初始化模拟量备份
SYS_SAVANA	16#25	保存模拟量
SYS_RESTANA	16#26	恢复模拟量
SYS_INITTMR	16#28	初始化计时器备份
SYS_SAVTMR	16#29	保存计时器
SYS_RESTTMR	16#2A	恢复计时器
SYS_INITALL	16#2C	初始化所有类型备份

SYS_SAVALL	16#2D	保存所有类型
SYS_RESTALL	16#2E	恢复所有类型

命令(予定义的关键字)	参数	返回值
SYS_INITxxx	内存地址	下一个未占用地址
SYS_SAVxxx	0	如果正常为零
SYS_REStxxx	0	如果正常为零

## 用户自定义的实现

最后，使用“C”函数或功能块，在应用程序的任何时刻，你可以开发特殊的用户过程，，用于访问电池备份内存，存储和恢复变量。

示例：

### 1) 应用程序的过程申明

\_backup, restore\_temp, restore\_date, restore\_cpt would be C user' s procedures

backup(temperature, date, cnt); 存储 3 个关键数据

温度:= restore\_temp(); 恢复温度

日期:= restore\_date(); 恢复日期

计数器:= restore\_cnt(); 恢复计数器

### 2) 通用过程

backup\_init, backup, backup\_link, restore would be C user' s procedures。

save\_id := backup\_init(address, size);            分配备份数组内存。

backup(save\_id, cpt1, 3);            保存 CPT1 为第三个元素。

rest\_id := backup\_link(address, size) 链接备份内存

cpt1 := restore(rest\_id, 3);            恢复 CPT1 的备份值

### C.9.3 程序状态备份

可以存储每个应用程序的每个状态，但是恢复每个程序为最后一次备份的状态，看起来有些危险。有至少三条理由：

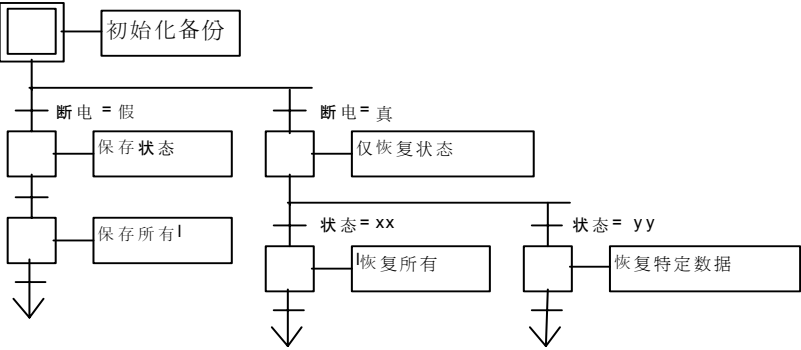
- 有些过程在再启动之前，需要特殊的操作
- 处理全部应用程序的每个状态是乏味的
- 有些资源，例如“C”程序，外设，等等，是不能自动重新启动的。

最好的解决方法是：由程序员决定，为了描述过程状态备份“模拟量”或“布尔量”，这样在重新启动过程中将能使用这些信息。

这样可以从一个不完整的但是理智的过程映像中起动，停止或冻结 SFC 程序，并且初始化变量，将应用程序置于一个等价状态。

ISaGRAF 不能提供自动启动过程。

示例



## C.10 “错误”信息表和解释

号	信息	类型
1	不能为运行数据库分配内存	系统
2	不正确的应用程序数据库或 CRC 错误	应用
3	不能分配通讯邮箱	系统
4	不能连接内核数据库	系统
5	向内核发送询问时超时	系统
6	等待内核响应时超时	系统
7	不能初始化通讯	系统
8	不能为保持变量分配内存	应用
9	应用程序中止	应用
10	同时进行的 N 或 P 动作太多	应用
11	同时进行的置位动作太多	应用
12	同时进行的复位动作太多	应用
13	未知的 TIC 指令	应用
16	不能响应读数据请求	系统
17	不能响应写数据请求	系统
18	不能响应调试器会话请求	系统
19	不能响应 MODBUS 请求	系统
20	不能响应调试器应用请求	系统
21	不能响应调试器	系统
23	未知的请求代码	系统

24	以太网通信错误	系统
25	通讯同步错误	系统
28	不能为应用程序分配内存	系统
29	不能为应用程序更新分配内存	系统
30	未知 OEM 码	应用
31	不能初始化布尔输入板	应用
32	不能初始化模拟输入板	应用
33	不能初始化信息输入板	应用
34	不能初始化布尔输出板	应用
35	不能初始化模拟输出板	应用
36	不能初始化信息输出板	应用
37	不能从布尔板输入	应用
38	不能从模拟板输入	应用
39	不能从信息板输入	应用
40	不能输出布尔变量	应用
41	不能输出模拟变量	应用
42	不能输出信息变量	应用
43	不能操作布尔变量	应用
44	不能操作模拟变量	应用
45	不能操作信息变量	应用
46	不能打开工作板	应用
47	不能关闭工作板	应用
50	不能重写布尔输出变量	程序
51	不能重写模拟输出变量	程序
52	不能重写信息输出变量	程序
61	未知的系统请求码	程序

62	采样周期超值	程序
63	用户函数未执行	应用
64	整型数被零除	程序
65	变换函数未执行	应用
66	功能块未执行	应用
67	标准函数未执行	应用
68	实型数被零除	程序
69	无效的操作参数	应用
72	不能修改应用程序符号	应用
73	不能更新，不同组的布尔变量	应用
74	不能更新，不同组的模拟变量	应用
75	不能更新，不同组的定时器变量	应用
76	不能更新，不同组的信息变量	应用
77	不能更新，不能发现新的应用程序	应用
> 100	特殊的 OEM 出错码，向供货商询问详情	

下面为三类较特别的错误信息：

– 系统出错：

这些问题源由于目标系统软、硬件，与应用设置和程序执行无关。

将目标硬件复位(断电)试着运行其它应用程序技术

将故障信息汇报给 ISaGRAF 支持。

– 应用出错：

这些问题源于应用参数，尺寸和内容

当装载入已知的和已经证明有效的应用后，这些出错将消失。如果再次出，问题将是上述系统出错

– 程序出错：

由于程序的实际顺序不当而产生

当将应用在循环至循环方式下启动，或是有问题的程序停止时，这类问题将消失。

出错解释：

1。 不能为运行数据库分配内存	系统
-----------------	----

没有有效的内存，检查硬件

2. 不正确的应用程序数据库或 CRC 错误	应用
------------------------	----

未能正确的下载或备份应用文件，当文件过时或是为 INTEL 而生成，下载至 MOTOROLA 时产生(和反之)

3. 不能分配通讯邮箱	系统
-------------	----

如果不能将位置 3 分配给内部通信任务，执行通讯任务将出此错

4. 不能连接内核数据库	系统
--------------	----

如果不能找到，带有运行时在命令行有特定从站号的内核时，通信任务将出此错

5. 向内核发送询问时超时	系统
---------------	----

通信任务不能向内核发送，内核可能未运行或“忙”

6. 等待内核响应时超时	系统
--------------	----

通信任务不能从内核接收到回答，核心可能未运行或“忙”

7. 不能初始化通讯	系统
------------	----

当通信层不能初始化物理链接产生些警告，如果未指定通讯路径也出此警告

这不能使目标运行不正常，但是不能通讯

## 8. 不能为保持变量分配内存

应用

ISaGRAF 不能管理保持变量.产生此问题可能有两种原因：

- 作为参数传递到主目标的字符串语法不正确。
- 分配给每个块的内存量不够。

你必须检查“保持变量”参数的语法。你可以试着减少“保持变量”的数量

## 9. 应用程序中止

应用

。

每次由于调试器引起应用程序停止，产生此警告

## 10. 同时进行的 N 或 P 动作太多

应用

在一个目标循环中，执行了太多“非存储”“脉冲触发动作”或者“循环块”，就产生此“出错”。可以切换到 CC 模式，以查清“出错”位置。每个 SFC 程序同时执行动作的最大值为  $2 + 4$ 。

## 11. 同时进行的置位动作太多

应用

在一个目标循环中，执行了太多置位动作”，就产生此“出错”。可以切换到 CC 模式，以查清“出错”位置。每个 SFC 程序同时执行动作的最大值为  $2 + 4$  置位动作(当步变为激活时执行)

## 12. 同时进行的复位动作太多

应用

在一个目标循环中，执行了太多复位动作”，就产生此“ 出错”。可以切换到 CC 模式，以查清“ 出错” 位置。每个 SFC 程序同时执行动作的最大值为 2 + 4。执行复位动作(当步变为非激活时执行)

13. 未知的 TIC 指令	应用
----------------	----

内核检测到应用程序代码中有误(调用了与目标无关的代码)可能有两种原因：

- 外部程序可能写入应用程序代码。切换到 CC 模式，以查清系统崩溃原因。确保所有 I/O 接口，参数正确。
- 你的目标有一个缩小了的指令集。应用程序中使用了非授权的指令或变量类型

16. 不能响应读数据请求	系统
---------------	----

MODBUS 请求功能码 18(读文件)时通讯出错，检查目标和主站双方的连接和系统配置

17. 不能响应写数据请求	系统
---------------	----

MODBUS 请求功能码 17(读文件)时通讯出错，检查目标和主站双方的连接和系统配置。

18. 不能响应调试器会话请求	系统
-----------------	----

。

回答调试器请求时通讯出错，检查目标和主站双方的连接和系统配置。

19. 不能响应 MODBUS 请求	系统
--------------------	----

回答 MODBUS 请求时通讯出错，检查目标和主站双方的连接和系统配置。

20. 不能响应调试器应用请求	系统
-----------------	----

回答调试器请求时通讯出错，检查目标和主站双方的连接和系统配置。

21. 不能响应调试器	系统
-------------	----

回答调试器请求时通讯出错，检查目标和主站双方的连接和系统配置。。

23. 未知的请求代码	系统
-------------	----

。

一个调试器请求没有回音

24. 以太网通信错误	系统
-------------	----

每次当调试器关闭时，连接也关闭了；系统工作正常，另外这意味着以太网通讯“出错”，检查目标和主站双方的连接和系统配置。

另外有可能：

1. 接收或发送时“出错”
2. 建立接口时“出错”
3. 连接或收听接口时“出错”
4. 接受一个新从站时“出错”

## 25. 通讯同步错误

系统

主站和从站(目标)通讯任务同步不好，检查双方的连接和系统配置(通讯参数)

## 28. 不能为应用程序分配内存

系统

。没有适用的内存，按照应用程序尺寸检查硬件

## 29. 不能为应用程序更新分配内存

系统

没有适用的内存，按照应用程序尺寸检查硬件。

## 30. 未知的 OEM 码

应用

应用程序使用的板的制造商代码不被目标所承认。在工作台检查 I/O 连接，使用“虚拟”属性查明不正确的板。工作台库文件与目标的版本不对应。

## 31. 不能初始化布尔输入板

应用

初始化布尔输入板失败，在工作台检查 I/O 连接，以及你的布尔输入板的参数

## 32. 不能初始化模拟输入板

应用

初始化模拟输入板失败，在工作台检查 I/O 连接，以及你的模拟输入板的参数

33. 不能初始化信息输入板	应用
----------------	----

初始化信息输入板失败，在工作台检查 I/O 连接，以及你的信息输入板的参数

34. 不能初始化布尔输出板	应用
----------------	----

初始化布尔输出失败，在工作台检查 I/O 连接，以及你的布尔输出的参数

35. 不能初始化模拟输出板	应用
----------------	----

初始化模拟输出板失败，在工作台检查 I/O 连接，以及你的模拟输出的参数

36. 不能初始化信息输出板	应用
----------------	----

。

初始化信息输出板失败，在工作台检查 I/O 连接，以及你的信息输出板的参数

37. 不能从布尔板输入	应用
--------------	----

当刷新布尔输入板时，检测到出错，检查工作台的 I/O 连接和板参数

38. 不能从模拟板输入	应用
--------------	----

当刷新模拟输入板时，检测到出错，检查工作台的 I/O 连接和板参数

39. 不能从信息板输入
--------------

应用
----

。当刷新信息输入板时，检测到出错，检查工作台的 I/O 连接和板参数

40. 不能输出布尔变量
--------------

应用
----

当刷新信息输入板时，检测到出错，检查工作台的 I/O 连接和板参数

41. 不能输出模拟变量
--------------

应用
----

当更新模拟输出变量时时，检测到出错，检查工作台的 I/O 连接和板参数

42. 不能输出信息变量
--------------

应用
----

当更新信息输出变量时时，检测到出错，检查工作台的 I/O 连接和板参数

43. 不能操作布尔变量
--------------

应用
----

。当执行一个布尔变量的操作调用时，检测到“ 出错” 。检查你的操作参数和板的用户说明。

44. 不能操作模拟变量	应用
--------------	----

。

当执行一个模拟变量的操作调用时，检测到“ 出错”。检查你的操作参数和板的用户说明。

45. 不能操作信息变量	应用
--------------	----

当执行一个信息变量的操作调用时，检测到“ 出错”。检查你的操作参数和板的用户说明。

46. 不能打开工作板	应用
-------------	----

应用使用了一个对于目标未知的板参照，在工作台检查 I/O 连接。你的工作台库不对应目标版本

47. 不能关闭工作板	应用
-------------	----

应用使用了一个对于目标未知的板参照，在工作台检查 I/O 连接。

50. 不能重写布尔输出变量	程序
----------------	----

同一目标循环中两个 SFC 顺序写了同一个布尔输出变量。这样将发生冲突的情况下，优先权将给予在层次中最高的程序。如果两个 SFC 程序处于同一层次，结果将不可预料。

51. 不能重写模拟输出变量	程序
----------------	----

在同一目标循环中，两个 SFC 程序写了同一个模拟输出变量。见一上条解释。

52. 不能重写信息输出变量
----------------

程序
----

在同一目标循环中，两个 SFC 程序写了同一个信息输出变量。见一上条解释。

61. 未知的系统请求码
--------------

程序
----

程序使用了带非法代码的系统调用

62. 采样周期超值
------------

程序
----

目标循环周期大于在工作台菜单的设定。在多任务系统中，这意味着，没有足够的 CPU 时间来执行循环。即使“当前循环周期”小于设定的周期在单位任务系统，这表示在一个目标循环周期中有过多的操作。

为了消除此条警告，有很多途径：

- 减少当警告检测到的时刻所处理的操作数量。
- 减少标记(令牌)的数量，有效的转换的数量，优化复杂的处理。
- 
- 使用动态的循环周期修正以适应不同处理阶段的周期变化。
- 设置循环时间接近为零，以使内核尽快运转，取消所有溢出检测。

63. 用户函数未执行
-------------

应用
----

程序使用了目标不能辨认的“ C” 函数，工作台库文件与目标版本不一致，

64. 整型数被零除	程序
------------	----

。

程序中出现整型量除零，在你的应用程序中应防止出现，产生不可预见的事件。当出现此情况时，ISaGRAF 将最大的模拟量作为结果。

当操作数为负值时，结果取反。

65. 变换函数未执行	应用
-------------	----

程序使用了目标不能辨认的“ C” 变换函数，工作台库文件与目标版本不一致，当次情况发生，，ISaGRAF 不能变换值

66. 功能块未执行	应用
------------	----

程序使用了目标不能辨认的“ C” 功能块，工作台库文件与目标版本不一致

。

67. 标准函数未执行	应用
-------------	----

。程序使用了目标不能辨认的功能块。尽管推测它适用于大多数目标，询问你的从货商

## 68. 实型数被零除

程序

。

程序中出现实型模拟量除零，在你的应用程序中应防止出现，产生不可预见的事件当出现此情况时，ISaGRAF 将最大的实型模拟量作为结果。

当操作数为负值时，结果取反。

## 69. 无效的操作参数

应用

应用程序使用错误参数调用操作。它通常被编译器所过滤。它可能是一个计时器参数，或者不是输入输出变量

## 72. 不能修改应用程序符号

应用

当更新应用程序时，因为符号不同，所以修改后的应用程序不能启动。与当前的应用程序相比，一个或多个变量，或者功能块实例可能被增，减或修改了

## 73. 不能更新，不同组的布尔变量

应用

与当前程序相比，修改后的程序因为其布尔变量发生了变化(增或减)，不能启动

## 74. 不能更新，不同组的模拟变量

应用

。

与当前程序相比，修改后的程序因为其模拟变量发生了变化(增或减)，不能启动

75. 不能更新，不同组的定时器变量	应用
--------------------	----

。

与当前程序相比，修改后的程序因为其计时器变量发生了变化(增或减)，不能启动

76. 不能更新，不同组的信息变量	应用
-------------------	----

与当前程序相比，修改后的程序因为其信息变量发生了变化(增或减)，不能启动。

77. 不能更新，不能发现新的应用程序	应用
---------------------	----

内存中找不到修订后的应用程序，下载时“ 出错” 。



## D. 词汇表

**动作 (FC)** 流程图表的符号。一个动作代表当动态流遇到动作符号时所完成的一系列指令。

**动作** 当一个 SFC 程序的程序步处于激活状态时，执行的一系列语句或赋值命令。

**程序步的激活** 由 SFC 标记设定的程序步的属性。当激活时，执行程序步的动作。

**模拟** 变量类型。模拟变量是连续的整型变量或实型变量。

**属性** 变量分类。可用的变量属性有内部变量、输入变量或输出变量。

**开始段** 在每个目标循环开始时，执行的循环程序组。

**开始步** 宏程序步的第一步。开始步不连接到任何以前的转换上。

**布尔动作** SFC 动作：由于步的激活，给一个布尔变量赋值。

**布尔** 变量类型。布尔变量仅可取真值或假值。

**断点** 用户在调试时，在 SFC 程序步或转换上设置的标记。当 SFC 标记移到断点时，目标系统停止。

- C 函数** 用"C"语言编写的函数。C 函数可由 ISaGRAF 程序(由其它语言编写的)以同步方式调用。C 函数可由 ICS Triplex ISaGRAF Inc. 公司提供, 或可由用户开发。
- C 语言** 用于描述计算机操作(如 C 函数和变换函数)的高级文字语言。
- C 源代码** 包含有函数和变换函数的 "C"源代码的文本文件。
- C 源头文件** 包含有函数和变换函数编程所需的"C" 定义和类型的文本文件。
- 单元** 用于图形语言(如 SFC、FBD 或 LD)的图形点阵的基本区域。
- SFC 子程序** 由称为主程序的另外 SFC 程序控制的 SFC 程序。
- 清除转换** 实时操作: 所有存在于前面程序步中的标记都被删除。在每个下一程序步中设立一个标记。
- 线圈** LD 程序中的图形部件。用于表示对输出变量的赋值。
- 注释** 包含在程序中的文本。这个文本不影响程序的执行。
- 注释 (SFC)** 与 SFC 程序步或转换有关的文本。这个文本不影响程序的执行 (SFC) 。
- 公共** 定义字的范围。这样的对象可以用于任何项目中的任何程序。
- 条件(用于 SFC 转换)** 与 SFC 转换有关的布尔表达式。当条件为假时, 转换不会被清

转换) 除。

连接符 FC 的图形组件，可以代表从流程图的一点到 FC 的动作或测试的 (FC) 连接。跳转的图形符号是一个小圆圈，为目的元素的编号。

常量表达式 用于描述常量值的文字表达式。常量表达式有专用类型。

触点 LD 程序的图形组件。它代表输入变量的状态。

变换函数 "C"语言编写的描述变换函数。这样的变换可以与任何输入或输出模拟变量相连。

转换表 定义线性(由段)转换点的集合。转换可适用于任何输入或输出模拟变量。

转换 连接到一个输入模拟变量或输出模拟变量的过滤器。转换每次自动使用的变量是输入变量或输出变量。

交叉引用 ISaGRAF 工作台计算得到的变量词典的信息，有关这些变量用在一个项目何处。

当前结果 (IL) 在 IL 程序中一条指令的结果。当前结果可由指令修改，或用来设置变量。

循环计时 目标执行周期的持续时间。

循环到循环 执行模式：在这个模式中，循环按照诊断器用户给定的顺序依次

---

环模式	执行。
循环	一个总在运行的程序的属性。
判定 (FC)	(判定也称为测试)与一个布尔表达式相连的流程图符号。这个流程根据表达式的状态直接输出“ 是” 或“ 否” 符号。
定义字	用于取代程序中的任何表达式的唯一标识符
延迟操作 (IL)	IL 程序中的一项操作，在程序中出现“(”指令时延时执行。
日志	包含在一个程序中所做的所有改变的要点的文本文件。 每个要点带有编辑日期。
词典	用于一个项目的各个程序内的内部变量、输入变量或输出变量和定义字的集合。
边沿	布尔变量的变化。 上升沿是指从假变到真，下降沿是指从真到假。
结束段	在每个目标周期结束时执行的循环程序组。
结束步	SFC 宏程序步的最后步。结束步不连接到以后的转换上。
表达式	描述处理值的操作符和标识符的集合。
SFC 主程	控制其他 SFC 程序(称为子程序)的 SFC 程序。

## 序

FBD            功能块图表语言。

功能块        表示 ISaGRAF 库标准基本功能的 FBD 语言的图形组件。

功能块图表    图形语言：由 ISaGRAF 库标准基本功能连接在一起的图形构成。  
表            。

全局           变量或定义字的取值范围。这样的对象可以用于任何项目中的任何程序。

层次           一个分为若干个程序的项目体系结构。层次树表示主程序和子程序之间的连接。

输入/输出    硬件资源。输入/输出插板具有类型和方向(输入或输出)的特性。

插板           输入/输出插板的参数在 ISaGRAF 库中描述。

输入/输出通道    输入/输出插板上的单独连接点。一个输入/输出通道可以与一个输入/输出变量相连接。

输入/输出连接    对应用程序的变量与已存在于目标系统中的插板上的通道之间的连接的定义。

输入/输出变量    连接到输入或输出设备上的变量。输入/输出变量必须连接到输入/输出插板上的通道上。

识别符        在编程中，用于代表变量或常量表达式的唯一字。

IL	指令列表语言;
初始环境	一个 SFC 程序的初始步的集合。这些初始步描述程序启动时程序的状态内容。
初始步	SFC 程序的特定步，该步在程序启动时被激活。
输入	变量的属性。这种变量可连接到输入设备上。
指令列表	低级文字语言，作为基本操作的序列列表输入。
指令	IL 程序的基本操作，文本输入在一个基本操作的顺序列表
整型	模拟变量类，按带符号整型 32 位格式存储。
内部	变量属性，该种变量不能连接到输入或输出设备上。
跳转到	SFC 的图形组件，表示从转换到步的连接。转移的图形符号是一个箭头，使用目的步的号码。
关键字	语言的保留字。
标号(IL)	位于在 IL 指令行开始的标识符，这个标识符可以识别指令，可用做 JMP 操作的操作数。
梯形图	由触点和线圈组成的、用于布尔表达式设计的图形语言。

LD            梯形图语言。

第 一 层 SFC 程序的主要描述。第一层分为图形(程序步与转换)以及相关 SFC 的注释。

第 二 层 SFC 程序的详细描述。描述各步之间的动作和与转换有关的布尔 SFC 条件。

库            硬件和软件资源的集合，这些资源可以直接插入到任何应用程序中。

局部          变量或定义字的范围。这样的对象仅可用在一个项目的一个程序之中。

锁 定 的 输 入/输出    由用户从诊断器发出的"锁"命令，将相应的输入/输出设备与输入或输出变量逻辑断开。

宏程序步    SFC 图形组件。宏程序步是在主图中的唯一描述的符号的程序步和转换的独立组，分别描述。

点阵          矩形单元编辑区域的逻辑分区，用以编辑一个图形语言程序。

信息          变量类型。这样的变量中包含有变量长度字符串。

Mod 总线    主从协议。在一个完整的体系结构中，为了同外部系统(如监视系统)连接，ISaGRAF 目标系统可以作为 Mod 总线的从站。

**修 改 符** 位于 IL 操作关键字后面的单个字符，该字符可以改变操作的方式 (IL) 。

**网络地址** 为每个变量自由定义的可选的十六进制地址。由 Mod 总线协议在目标系统与外部系统连接时使用的地址。

**非 存 储 动 作** SFC 动作：当相应程序步被激活时，在每个目标周期中执行一次的一组语句。

**OEM 关键 入/输出插板)** 为 ISaGRAF 库中每块输入/输出插板定义的 16 位十六进制代码字代码 (输入/输出插板)。OEM 代码由插板的供应商提供。

**OEM 参数 插板)** 输入/输出插板的参数，这个参数由插板的设计者定义。它可以是 (输入/输出插板) 常数或变量，参数由用户在输入/输出连接时输入。

**操作数(IL)** 由初级 IL 指令处理的变量或常量表达式。

**操作(IL)** IL 语言的基本指令。操作通常作用到指令中的相关作数。

**输出** 变量的属性。这样的变量可连接到目标机器的输出设备上。

**参数(C 函 数)** 提供给"C"函数作为输入的数值。参数有类型的特征。

**参数(输入/ 标准输入/输出插板的由用户定义的参数或常数参数。用户定义的输出插板)** 参数由程序员在输入/输出的连接过程中输入。

**主程序** 用任何语言编写的程序，这个程序可以控制其它称为子程序的非 SFC 程序。调用子程序。

**电源轨道** 位于梯形图的末端，左右两条主要的垂直电源线。

**程序** 一个项目的基本编程单元。程序用一种语言编写，位于项目层次结构的一定位置。

**项目** 用于一个 ISaGRAF 应用程序的所有信息（如程序、变量、目标代码等）集合的编程区域。

**脉冲动作** SFC 动作：仅在相应程序步被激活时，执行一次的语句组。

**范围** 能够用于一个对象的程序的集合。预定义的 ISaGRAF 的范围为公共、全局和局部三种。

**实型插板** 实际连接到一台目标机器输入输出设备上的输入输出插板。

**实时模式** 运行时间正常模式：由可编程循环计时来触发目标循环。

**实型** 模拟变量类型。按浮点 IEEE 单精度 32 位格式存储。

**参 考 数 (SFC)** 用于识别 SFC 程序中 SFC 程序步或转换的十进制数(从 1 到 65535)。

寄存器 一个 IL 序列的当前结果。

(IL)

子程序的返回值 由子程序执行结束时返回的数值。返回值被用于上级主程序的操作。

运行时间错误 由 ISaGRAF 目标系统在运行时检测到的应用程序错误。

段 按照相同的动态规则执行的一组程序。

分隔符 用来在一种文字语言中分隔标识符的一个特殊字符(或一组字符)。一个分隔符可以表示一种操作。

顺序功能图 图形语言：即与转换相连的各个程序步的集合。各个动作都与各个程序步相关。转换则与布尔条件相关。

顺序段 一个项目程序的组合。这些程序都是按照 SFC 语言的动态规则执行的。

SFC 顺序功能图语言

语句 基本的 ST 完整操作。

程序步 SFC 语言的基本图形组件。一个程序步代表数据处理过程中的一个稳定状态，画成一个矩形。一个程序步由一个数字注明。一个

	程序步的激活可用来控制相应动作的执行。
串	存储在信息变量中的一系列字符。
结构文本	由赋值语句、高级结构化语言如 f/Then/Else 和功能调用组成的高级结构化文字语言。
ST	结构化文本语言。
子程序	用除了 SFC 之外的语言编写的程序。它可由上级其它程序调用。
目标周期	每次 ISaGRAF 目标系统被启动时执行的操作序列。循环由可编程循环计时触发执行。
目标	由 ISaGRAF 核心软件支持的 ISaGRAF 目标机器。
技术说明	ISaGRAF 库的每一个元素(如 C 函数、功能块、转换功能或输入/输出板等)的用户指南。技术说明由元素的设计者编写。
测试 (FC)	(也可称为判定) 与布尔表达式相关的流程图符号。根据表达式的状态,流程图直接输出：“是”或“不”。
计时器	变量类型。计时器变量为时间值，可由 ISaGRAF 系统在运行时自动刷新。
标记	用来表示 SFC 程序的激活程序步的图形符号。

## (SFC)

工具框	一个图形编辑窗口中的小窗口。这个小窗口有用于图形部件选择的主要按钮的分组。
顶级程序	位于层次树的顶级的程序。顶级程序由系统启动。
转换	基本的 SFC 组件。一个转换代表不同的 SFC 程序步之间的条件。转换由数字定位。每个转换都与一个布尔条件相关。
类型	具有同一格式的变量的归类。基本类型有布尔变量、模拟变量、计时器变量和信息变量。
转换变量	转换变量的属性。在所有以前的程序步都是活动的时,转换才是有效的。
变量	在目标程序中处理的基本元素的唯一代表。
虚拟插板	没有实际连接到一台目标机器输入输出设备上的输入输出插板。