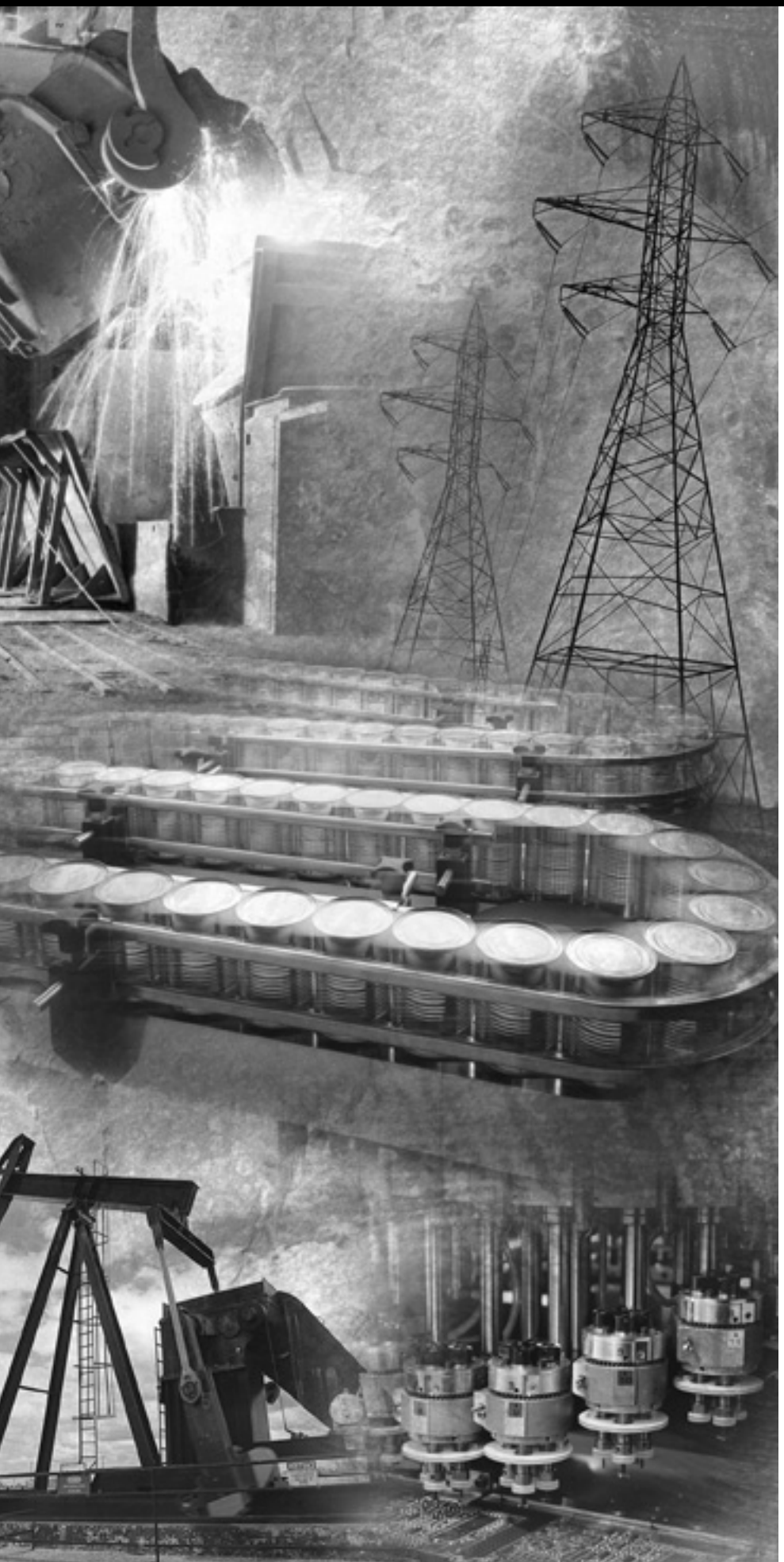


Allen-Bradley

Logix5000™ 控制器过程控制和驱动控制指令集

1756-Lx, 1769-Lx, 1789-Lx, 1794-Lx, PowerFlex 700S

用户手册

Rockwell Automation

重要用户信息

固态设备与机电设备具有不同的运行特性。Safety Guidelines for the Application, Installation and Maintenance of Solid State Controls （固态控制的应用、安装和维护安全准则，出版号为 SGI-1.1，可从当地的 Rockwell Automation 销售处索取或在 http://www.ab.com/manuals/gi 联机查看）说明了固态设备与硬布线的机电设备之间的某些重要差别。由于这些差别以及固态设备的多种应用，因此负责应用此设备的人员必须确保其所有要进行的应用都合乎要求。

在任何情况下， Rockwell Automation, Inc. 对因应用此设备而造成的直接或间接损失均不承担责任。

本手册中的示例和图表只用于说明。由于任何具体安装都存在许多变化和要求，因此 Rockwell Automation, Inc. 无法对基于这些示例与图表的实际应用承担责任和义务。

Rockwell Automation, Inc. 不承担对于使用本手册中说明的信息、电路、设备或软件的专利责任。

未经 Rockwell Automation, Inc. 书面许可，禁止复制本手册的全部或部分内容。

我们在整个手册中以注释来提醒您注意安全考虑事项。

警告

标明可能在危险的环境中引起爆炸 （会导致人员伤亡、财产损失或经济损失）的操作与环境信息。

重要事项

标明成功应用及理解此项产品的关键信息。

注意

标明会导致人员伤亡、财产损失或经济损失的操作及环境信息。“注意”有助您：

- 识别危险
- 避免危险
- 认清后果

有电危险

在驱动器表面或内部可能贴有标签，以提醒人们可能会有危险的电压。

高温危险

在驱动器表面或内部可能贴有标签，以提醒人们表面温度可能有危险。

简介

本文档的这一版本含有新增和更新的信息。要查找新增和更新的信息，请查看本段下面的更新列表。

更新的信息

本文档包含下列更新：

更新：	请参阅：
关于 ALM 死区及变动率计算的附加信息	1-4
PIDE 对 WindupHIn 和 WindupLIn 输入的响应方式的更新	1-46
关于 PIDE 算法的附加信息	1-55
对 TotalTime 等式的更正	2-35
对实时采样的更正	B-9
对面板控件的更新	附录 D

说明：

在何处查找指令

可用本查询表查找有关 Logix 指令的详细参考信息 （可在其他手册中找到灰显的指令）。本查询表还列出可用于下列指令的编程语言。

如果此查询表列出的是：	则指令记录在：
页码	本手册中
基本指令	*Logix5000 控制器指令集参考手册*， 1756-RM003
运动指令	*Logix5000 控制器运动指令集参考手册*， 1756-RM007
相位指令	*Logix5000 控制器 PhaseManager 用户手册*， LOGIX-UM001

指令：	位置：	语言：
ABL 对缓冲区行进行 ASCII 测试	基本指令	梯形图 结构化文本
ABS 绝对值	基本指令	梯形图 结构化文本 功能块
ACB 缓冲区中的 ASCII 字符数	基本指令	梯形图 结构化文本
ACL ASCII 清零缓冲区	基本指令	梯形图 结构化文本
ACOS 反余弦	基本指令	结构化文本
ACS 反余弦	基本指令	梯形图 功能块
ADD 加	基本指令	梯形图 结构化文本 功能块
AFI 恒假指令	基本指令	梯形图
AHL ASCII 握手信号线	基本指令	梯形图 结构化文本
ALM 报警	第 1 章 -2	结构化文本 功能块
AND 按位与	基本指令	梯形图 结构化文本 功能块
ARD ASCII 读取	基本指令	梯形图 结构化文本
ARL ASCII 读取行	基本指令	梯形图 结构化文本
ASIN 反正弦	基本指令	结构化文本

指令：	位置：	语言：
ASN 反正弦	基本指令	梯形图 功能块
ATAN 反正切	基本指令	结构化文本
ATN 反正切	基本指令	梯形图 功能块
AVE 文件均值	基本指令	梯形图
AWA ASCII 追加写入	基本指令	梯形图 结构化文本
AWT ASCII 写入	基本指令	梯形图 结构化文本
BAND 逻辑与	基本指令	结构化文本 功能块
BNOT 逻辑非	基本指令	结构化文本 功能块
BOR 逻辑或	基本指令	结构化文本 功能块
BRK 暂停	基本指令	梯形图
BSL 位左移	基本指令	梯形图
BSR 位右移	基本指令	梯形图
BTD 位域分配	基本指令	梯形图
BTDT 定向位域分配	基本指令	结构化文本 功能块
BTR 消息	基本指令	梯形图 结构化文本
BTW 消息	基本指令	梯形图 结构化文本

指令：	位置：	语言：
BXOR 逻辑异或	基本指令	结构化文本 功能块
CLR 清零	基本指令	梯形图 结构化文本
CMP 比较	基本指令	梯形图
CONCAT 字符串连接	基本指令	梯形图 结构化文本
COP 复制文件	基本指令	梯形图 结构化文本
COS 余弦	基本指令	梯形图 结构化文本 功能块
CPS 同步文件复制	基本指令	梯形图 结构化文本
CPT 计算	基本指令	梯形图
CTD 倒计数	基本指令	梯形图
CTU 正计数	基本指令	梯形图
CTUD 正计数 / 倒计数	基本指令	结构化文本 功能块
D2SD 离散 2 态设备	第 1 章 -7	结构化文本 功能块
D3SD 离散 3 态设备	第 1 章 -16	结构化文本 功能块
DDT 诊断性检测	基本指令	梯形图
DEDT 死区时间	第 1 章 -29	结构化文本 功能块
DEG 转换为角度	基本指令	梯形图 结构化文本 功能块
DELETE 删除字符串	基本指令	梯形图 结构化文本
DERV 微分	第 3 章 -2	结构化文本 功能块
DFF D 触发器	第 6 章 -2	结构化文本 功能块
DIV 除	基本指令	梯形图 结构化文本 功能块
DTOS 将 DINT 转换为字符串	基本指令	梯形图 结构化文本
DTR 数据转换	基本指令	梯形图

指令：	位置：	语言：
EOT 转换结束	基本指令	梯形图 结构化文本
EQU 等于	基本指令	梯形图 结构化文本 功能块
ESEL 增强型选择	第 4 章 -2	结构化文本 功能块
EVENT 触发事件任务	基本指令	梯形图 结构化文本
FAL 文件算术和逻辑	基本指令	梯形图
FBC 文件位比较	基本指令	梯形图
FFL FIFO 装载	基本指令	梯形图
FFU FIFO 卸载	基本指令	梯形图
FGEN 函数发生器	第 1 章 -34	结构化文本 功能块
FIND 查找字符串	基本指令	梯形图 结构化文本
FLL 文件填充	基本指令	梯形图
FOR 循环	基本指令	梯形图
FRD 转换为整数	基本指令	梯形图 功能块
FSC 文件搜索和比较	基本指令	梯形图
GEQ 大于等于	基本指令	梯形图 结构化文本 功能块
GRT 大于	基本指令	梯形图 结构化文本 功能块
GSV 获取系统值	基本指令	梯形图 结构化文本
HLL 上 / 下限	第 4 章 -9	结构化文本 功能块
HPF 高通滤波器	第 3 章 -6	结构化文本 功能块
ICON 输入线连接器	B-1	功能块
INSERT 插入字符串	基本指令	梯形图 结构化文本
INTG 积分器		结构化文本 功能块

指令：	位置：	语言：
IOT 立即输出	基本指令	梯形图 结构化文本
IREF 输入引用	B-1	功能块
JKFF JK 触发器	第 6 章 -4	结构化文本 功能块
JMP 跳至标签	基本指令	梯形图
JSR 跳至子程序	基本指令	梯形图 结构化文本 功能块
JXR 跳至外部程序	基本指令	梯形图
LBL 标签	基本指令	梯形图
LDL2 二阶超前 / 滞后滤波	第 3 章 -12	结构化文本 功能块
LDLG 超前 / 滞后	第 1 章 -38	结构化文本 功能块
LEQ 小于等于	基本指令	梯形图 结构化文本 功能块
LES 小于	基本指令	梯形图 结构化文本 功能块
LFL LIFO 装载	基本指令	梯形图
LFU LIFO 卸载	基本指令	梯形图
LIM 限度	基本指令	梯形图 功能块
LN 自然对数	基本指令	梯形图 结构化文本 功能块
LOG 以 10 为底的对数	基本指令	梯形图 结构化文本 功能块
LOWER 小写	基本指令	梯形图 结构化文本
LPF 低通滤波器	第 3 章 -18	结构化文本 功能块
MAAT 运动应用轴调整	运动指令	梯形图 结构化文本
MAFR 运动轴故障复位	运动指令	梯形图 结构化文本
MAG 运动轴传动	运动指令	梯形图 结构化文本
MAH 运动轴归位	运动指令	梯形图 结构化文本
MAHD 运动应用连接诊断	运动指令	梯形图 结构化文本
MAJ 运动轴点动	运动指令	梯形图 结构化文本
MAM 运动轴移动	运动指令	梯形图 结构化文本
MAOC 运动启用输出凸轮	运动指令	梯形图 结构化文本
MAPC 运动轴定位凸轮	运动指令	梯形图 结构化文本
MAR 运动定位	运动指令	梯形图 结构化文本
MAS 运动轴停止	运动指令	梯形图 结构化文本
MASD 运动轴关闭	运动指令	梯形图 结构化文本
MASR 运动轴关闭复位	运动指令	梯形图 结构化文本
MATC 运动轴时间凸轮	运动指令	梯形图 结构化文本
MAVE 移动均值	第 5 章 -2	结构化文本 功能块
MAW 运动启用查看	运动指令	梯形图 结构化文本
MAXC 获取最大值	第 5 章 -6	结构化文本 功能块
MCCP 运动计算凸轮轮廓	运动指令	梯形图 结构化文本
MCD 运动动态改变	运动指令	梯形图 结构化文本
MCR 主控复位	基本指令	梯形图
MDF 运动定向驱动关闭	运动指令	梯形图 结构化文本
MDO 运动定向驱动打开	运动指令	梯形图 结构化文本
MDOC 运动撤消输出凸轮	运动指令	梯形图 结构化文本
MDR 运动撤消定位	运动指令	梯形图 结构化文本
MDW 运动撤消查看	运动指令	梯形图 结构化文本
MEQ 屏蔽等于	基本指令	梯形图 结构化文本 功能块
MGS 运动组停止	运动指令	梯形图 结构化文本

指令：	位置：	语言：
MGSD 运动组关闭	运动指令	梯形图 结构化文本
MGSP 运动组选通位置	运动指令	梯形图 结构化文本
MGSR 运动组关闭复位	运动指令	梯形图 结构化文本
MID 字符串部分引用	基本指令	梯形图 结构化文本
MINC 获取最小值	第 5 章 -9	结构化文本 功能块
MOD 取模	基本指令	梯形图 结构化文本 功能块
MOV 移动	基本指令	梯形图
MRAT 运动运行轴调整	运动指令	梯形图 结构化文本
MRHD 运动运行连接诊断	运动指令	梯形图 结构化文本
MRP 运动重定位	运动指令	梯形图 结构化文本
MSF 运动伺服关闭	运动指令	梯形图 结构化文本
MSG 消息	基本指令	梯形图 结构化文本
MSO 运动伺服打开	运动指令	梯形图 结构化文本
MSTD 移动标准偏差	第 5 章 -12	结构化文本 功能块
MUL 乘	基本指令	梯形图 结构化文本 功能块
MUX 多路切换器	第 4 章 -12	功能块
MVM 屏蔽传送	基本指令	梯形图
MVMT 定向屏蔽传送	基本指令	结构化文本 功能块
NEG 求反	基本指令	梯形图 结构化文本 功能块
NEQ 不等于	基本指令	梯形图 结构化文本 功能块
NOP 空操作	基本指令	梯形图
NOT 按位非	基本指令	梯形图 结构化文本 功能块

指令：	位置：	语言：
NTCH 陷波滤波器	第 3 章 -24	结构化文本 功能块
OCON 输出线连接器	B-1	功能块
ONS 单触发	基本指令	梯形图
OR 按位或	基本指令	梯形图 结构化文本 功能块
OREF 输出引用	B-1	功能块
OSF 下降沿触发	基本指令	梯形图
OSFI 带输入下降沿触发	基本指令	结构化文本 功能块
OSR 上升沿触发	基本指令	梯形图
OSRI 带输入上升沿触发	基本指令	结构化文本 功能块
OTE 输出激励	基本指令	梯形图
OTL 输出锁存	基本指令	梯形图
OTU 输出解锁	基本指令	梯形图
PATT 连接到设备相位	相位指令	梯形图 结构化文本
PCLF 设备相位故障清零	相位指令	梯形图 结构化文本
PCMD 设备相位命令	相位指令	梯形图 结构化文本
PDET 与设备相位分离	相位指令	梯形图 结构化文本
PFL 设备相位故障	相位指令	梯形图 结构化文本
PI 比例 + 积分	第 2 章 -8	结构化文本 功能块
PID 比例积分微分	基本指令	梯形图 结构化文本
PIDE 增强型 PID	1-42	结构化文本 功能块
PMUL 脉冲倍增器	第 2 章 -20	结构化文本 功能块
POSP 位置比例	1-78	结构化文本 功能块
POVR 设备相位取缔命令	相位指令	梯形图 结构化文本

指令：	位置：	语言：
PPD 设备相位暂停	相位指令	梯形图 结构化文本
PRNP 设备相位新参数	相位指令	梯形图 结构化文本
PSC 相态完成	相位指令	梯形图 结构化文本
PXRQ 设备相位外部请求	相位指令	梯形图 结构化文本
RAD 转换为弧度	基本指令	梯形图 结构化文本 功能块
RES 复位	基本指令	梯形图
RESD 复位优先	第 6 章 -6	结构化文本 功能块
RET 返回	基本指令	梯形图 结构化文本 功能块
RLIM 速率限幅器	第 4 章 -15	结构化文本 功能块
RMPS 斜坡 / 渗透	1-85	结构化文本 功能块
RTO 保持型导通定时器	基本指令	梯形图
RTOR 带复位的保持型导通定时器	基本指令	结构化文本 功能块
RTOS 将 REAL 转换为字符串	基本指令	梯形图 结构化文本
SBR 子程序	基本指令	梯形图 结构化文本 功能块
SCL 比例缩放	1-99	结构化文本 功能块
SCRV S- 曲线	第 2 章 -28	结构化文本 功能块
SEL 选择	第 4 章 -19	功能块
SETD 置位优先	第 6 章 -8	结构化文本 功能块
SFP SFC 暂停	基本指令	梯形图 结构化文本
SFR SFC 复位	基本指令	梯形图 结构化文本
SIN 正弦	基本指令	梯形图 结构化文本 功能块

指令：	位置：	语言：
SIZE 以元素计的大小	基本指令	梯形图 结构化文本
SNEG 选择取反	第 4 章 -21	结构化文本 功能块
SOC 二阶控制器	第 2 章 -37	结构化文本 功能块
SQI 定序器输入	基本指令	梯形图
SQL 定序器装载	基本指令	梯形图
SQO 定序器输出	基本指令	梯形图
SQR 平方根	基本指令	梯形图 功能块
SQRT 平方根	基本指令	结构化文本
SRT 文件排序	基本指令	梯形图 结构化文本
SRTP 脉宽调制	1-103	结构化文本 功能块
SSUM 选择求和	第 4 章 -23	结构化文本 功能块
SSV 将系统值置位	基本指令	梯形图 结构化文本
STD 文件标准偏差	基本指令	梯形图
STOD 将字符串转换为 DINT	基本指令	梯形图 结构化文本
STOR 将字符串转换为 REAL	基本指令	梯形图 结构化文本
SUB 减	基本指令	梯形图 结构化文本 功能块
SWPB 字节重排	基本指令	梯形图 结构化文本
TAN 正切	基本指令	梯形图 结构化文本 功能块
TND 临时截止	基本指令	梯形图
TOD 转换为 BCD	基本指令	梯形图 功能块
TOF 延迟关闭定时器	基本指令	梯形图
TOFR 带复位延迟关闭定时器	基本指令	结构化文本 功能块
TON 延迟导通定时器	基本指令	梯形图

指令：	位置：	语言：
TONR 带复位延迟导通定时器	基本指令	结构化文本 功能块
TOT 累加器	1-109	结构化文本 功能块
TRN 截断	基本指令	梯形图 功能块
TRUNC 截断	基本指令	结构化文本
UID 禁用用户中断	基本指令	梯形图 结构化文本
UIE 启用用户中断	基本指令	梯形图 结构化文本
UPDN 增 / 减累加器	第 2 章 -47	结构化文本 功能块
UPPER 大写	基本指令	梯形图 结构化文本
XIC 检查是否已关闭	基本指令	梯形图
XIO 检查是否已打开	基本指令	梯形图
XOR 按位异或	基本指令	梯形图 结构化文本 功能块
XPY X 的 Y 次幂	基本指令	梯形图 结构化文本 功能块

简介

本手册是介绍 Logix 基本指令的资料之一。

任务 / 目标：	参考资料：
对控制器编程以实现顺序控制应用	*Logix5000 控制器指令集参考手册*，出版号：1756-RM003
对控制器编程以实现过程或驱动控制应用 **您现在位于** 	*Logix5000 控制器过程和驱动指令集参考手册*，出版号：1756-RM006
对控制器编程以实现运动控制应用	*Logix5000 控制器运动指令集参考手册*，出版号：1756-RM007
将文本文件或标记导入项目	*Logix5000 控制器导入 / 导出参考手册*，出版号：1756-RM084
将项目或标记导出到文本文件	
将 PLC-5 或 SLC 500 应用程序转换为 Logix5000 应用程序	*Logix5550 控制器 PLC-5 或 SLC 500 逻辑到 Logix5000 逻辑转换参考手册*，出版号：1756-RM085

以下是介绍 Logix5000 系列控制器的核心文档：

如果您是：	参考资料：
Logix5000 控制器的新用户 该快速入门为您提供了关于配置和运行控制器所需基本步骤的图示分步介绍。	*Logix5000 控制器快速入门* 出版号：1756-QS001
Logix5000 控制器的有经验的使用者 该设计参考提供了规划和实施 Logix5000 控制系统时需考虑的要素。	*Logix5000 控制器设计注意事项参考手册* 出版号：1756-QR107
Logix5000 控制器的有经验的使用者 该系统参考中详细列举了配置信息、控制器特性以及指令 （梯形图、功能块图和结构化文本）。	*Logix5000 控制器系统参考* 出版号：1756-QR107
Logix5000 控制器的任一用户 该通用程序手册讲解了所有 Logix5000 控制器的通用特性和功能。	*Logix5000 控制器通用程序* 出版号：1756-PM001

本手册的使用对象

本手册为程序设计者提供 Logix 系列控制器所有可用指令的详细信息。您首先应已掌握 Logix 系列控制器存储和处理数据的原理。

初学者在使用指令之前应先阅读其相关的详细资料。有经验的程序设计者可通过查阅的方式获取所用指令的详细信息。

本手册的用途

本手册使用以下格式介绍每条指令的相关内容。

区域：	提供以下信息：
指令名称	标识指令 定义指令为输入指令或输出指令
操作数	列举指令的所有操作数  **梯形图中可用的操作数**  **结构化文本中可用的操作数**  **功能块中可用的操作数** **功能块上仅显示其默认引脚。操作数表中列举了功能块的所有可用引脚。**
指令结构	如果有则列出指令的控制状态位及其对应值
说明	描述指令的用法 如适用，则详细说明指令启用和禁用时的所有区别
算术状态标志	说明指令是否影响算术状态标志 请参阅附录 “通用属性”
故障条件	说明指令是否产生次要或主要故障 如果产生，则列出故障类型和代码
执行	详细说明指令执行的细节
示例	为每种可用的编程语言至少提供一个编程示例 针对每个示例加以说明

下列图标有助于识别编程语言的特殊信息：

图标：	表示以下编程语言：
	梯形图
	结构化文本
	功能块

所有指令的通用信息

Logix5000 指令集具有以下通用属性：

相关信息：	参阅附录：
通用属性	通用属性附录中介绍了： • 算术状态标志 • 数据类型 • 关键字
数组	Array Concepts 附录中定义了数组的概念并对控制器处理数组的方式进行了说明
功能块属性	功能块属性附录中介绍了： • 程序和操作员控制 • 定时方式

约定和相关术语

置位和清零

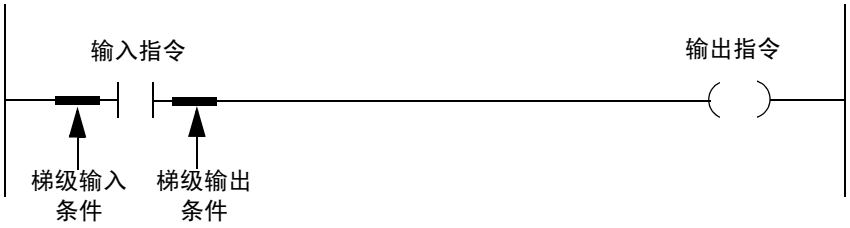
本手册使用置位和清零来定义位 （布尔值）和数值 （非布尔值）的状态：

术语：	含义：
置位	使该位为 1 （开） 使该数值为非零值
清零	使该位为 0 （关） 将该数值中的所有位清零

如果操作数或参数支持多种数据类型，则粗体字的数据类型为最优数据类型。如果指令的所有操作数均使用同一最优数据类型 （通常是 DINT 或 REAL），则指令的执行速度将更快，所需内存也更少。

梯形图梯级条件

控制器基于指令前的梯级条件 （梯级输入条件）计算梯形图指令。然后，控制器根据梯级输入条件和指令对指令后的梯级条件（梯级输出条件）进行设置，通过这种方式依次作用于后续指令。



如果一个输入指令的梯级输入条件为真，则控制器会对该指令进行计算，并根据计算结果设置梯级输出条件。如果该指令的计算结果为真，则输出条件为真；如果该指令的计算结果为假，则输出条件也为假。

控制器还会对指令进行预扫描。预扫描是对控制器中所有例程的一种特殊扫描方式。控制器在预扫描期间扫描所有主例程和子例程，但会忽略可导致跳过指令执行的跳转指令。控制器执行所有 FOR 循环和子例程调用。如子例程被多次调用，则每次调用时都会被执行。控制器通过对梯形图指令进行预扫描来重置非保持型 I/O 和内部值。

在预扫描期间，输入值不是当前值，也不会写入输出数据。以下条件会触发预扫描：

- 从 Program 方式转换到 Run 方式
- 从上电状态自动进入 Run 方式。

在以下情况时不会对程序进行预扫描：

- 当控制器运行时预定程序。
- 当控制器进入 Run 模式时取消程序预定。

功能块状态

控制器根据不同状态条件对功能块指令进行计算。

可能的条件：	说明：
预扫描	对功能块例程的预扫描与梯形图例程中相同。唯一的区别是每个功能块指令的 EnableIn 参数在预扫描期间都会被清零。
指令首次扫描	指令首次扫描指的是指令在预扫描后的首次执行。控制器通过指令首次扫描来读取当前输入值并决定将进入的正确状态。
指令首次执行	指令首次执行指的是指令首次以数据结构的新实例执行。控制器通过指令首次执行来生成系数和其他数据存储，这些内容在初始下载后对于功能块便不再改变。

每个功能块指令均包括 EnableIn 和 EnableOut 参数：

- EnableIn 置位时，功能块指令正常执行。
- EnableIn 清零时，功能块指令将执行预扫描逻辑、后期扫描逻辑或只是跳过正常算法的执行。
- EnableOut 是 EnableIn 的映像，如果功能块执行时检测到溢出条件，EnableOut 也被清零。
- 当 EnableIn 从清零跳转为置位状态时，功能块从断点继续执行。然而有些功能块指令在 EnableIn 从清零跳转为置位状态时可指定特殊功能，如重新初始化。对于具有时基参数的功能块指令，如果其定时方式为过采样，当 EnableIn 从清零跳转为置位状态时，功能块总是从断点继续执行。

如果 EnableIn 参数未连接，指令始终正常执行且 EnableIn 保持置位状态。如果您将 EnableIn 清零，则指令下次执行时该位仍将返回置位状态。

重要事项

由于其内部浮点数运算使用单精度浮点数，在功能块中编程时，工程单位范围限制在 $+/-10^{+/-15}$ 。如果计算结果接近单精度浮点数极限 ($+/-10^{+/-38}$)，使用超出该范围的工程单位会导致精度降低。

目录
过程控制指令（ALM、 D2SD、 D3SD、 DEDT、 FGEN、 LDLG、 PIDE、 POSP、 RMPS、 SCL、 SRTP、 TOT）

第 1 章

驱动控制指令（INTG、PI、PMUL、SCRV、SOC、UPDN）

第 2 章

滤波指令（DERV、HPF、LDL2、LPF、NTCH）

第 3 章

选择 / 限制指令（ESEL、HLL、MUX、RLIM、SEL、SNEG、SSUM）

第 4 章

统计指令（MAVE、MAXC、MINC、MSTD）

第 5 章

移动 / 逻辑指令（DFF、JKFF、RESD、SETD）

第 6 章

通用属性

附录 A

功能块属性

附录 B

结构化文本编程

附录 C

功能块面板控件

索引

附录 D

说明：

过程控制指令
（ALM、 D2SD、 D3SD、 DEDT、 FGEN、 LDLG、 PIDE、 POSP、 RMPS、 SCL、 SRTP、 TOT）

简介

这些过程控制指令可用于：

如要：	所用指令：	可用编程语言：	参见页：
对任意模拟量信号实现报警功能。	报警 (ALM)	结构化文本 功能块	1-2
控制只有两种可能状态 （开 / 关、打开 / 闭合等）的离散设备，如电磁阀、泵和电机。	离散 2 态设备 (D2SD)	结构化文本 功能块	1-7
控制有三种可能状态 （快 / 慢 / 关、正转 / 停止 / 反转等）的离散设备，如具有快 / 慢两档速度的进料器。	离散 3 态设备 (D3SD)	结构化文本 功能块	1-16
使单路输入信号延时。您可以选择死区时间的延迟量。	死区时间 (DEDT)	结构化文本 功能块	1-29
基于分段线性函数对输入信号进行转换。	函数发生器 (FGEN)	结构化文本 功能块	1-34
对输入信号进行相位超前滞后补偿。	超前滞后 (LDLG)	结构化文本 功能块	1-38
使用 PID 算法调节模拟量输出，将过程变量维持在某设定点。	增强型 PID (PIDE)	结构化文本 功能块	1-42
通过脉冲控制信号打开 / 闭合触点来升高 / 降低或打开 / 关闭设备 （如电动阀）。	位置比例 (POSP)	结构化文本 功能块	1-78
根据温度曲线形成交替的升温和保温周期。	升温 / 保温 (RMPS)	结构化文本 功能块	1-85
将未标度的输入数值转换为工程单位的浮点数。	标度 (SCL)	结构化文本 功能块	1-99
根据 PID 回路的输出比率生成一个周期性脉冲，用于驱动加热和制冷的接触器。	拆分范围时间比例 (SRTP)	结构化文本 功能块	1-103
生成模拟量输入对时间的累加值，如流量。	累加器 (TOT)	结构化文本 功能块	1-109

报警 (ALM)

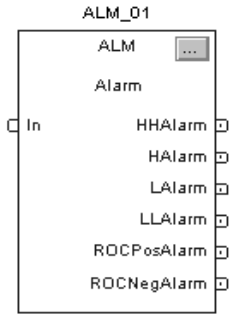
ALM 指令可对任意模拟量信号实现报警功能。

操作数：

```
ALM(ALM_tag);
```

结构文本

操作数：	类型：	格式：	说明：
ALM 标记	ALARM	结构	ALM 结构



功能块

操作数：	类型：	格式：	说明：
ALM 标记	ALARM	结构	ALM 结构

ALARM 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果该位清零，则指令不执行且输出不更新。 如果设置，则指令执行。 缺省状态为设置。 **结构化文本：** 无影响。指令一直执行。
In	REAL	模拟量信号输入。 有效值 = 浮点数 缺省值 = 0.0
HHLimit	REAL	输入信号的最高报警上限。 有效值 = 任意实数值 缺省值 = 最大正数
HLimit	REAL	输入信号的报警上限。 有效值 = 任意实数值 缺省值 = 最大正数
LLimit	REAL	输入信号的报警下限。 有效值 = 任意实数值。 缺省值 = 最大负数
LLLimit	REAL	输入信号的最低报警下限。 有效值 = 任意实数值 缺省值 = 最大负数
Deadband	REAL	最高上限到最低下限的报警死区。 有效值 = 任意实数值 ≥ 0.0 缺省值 = 0.0

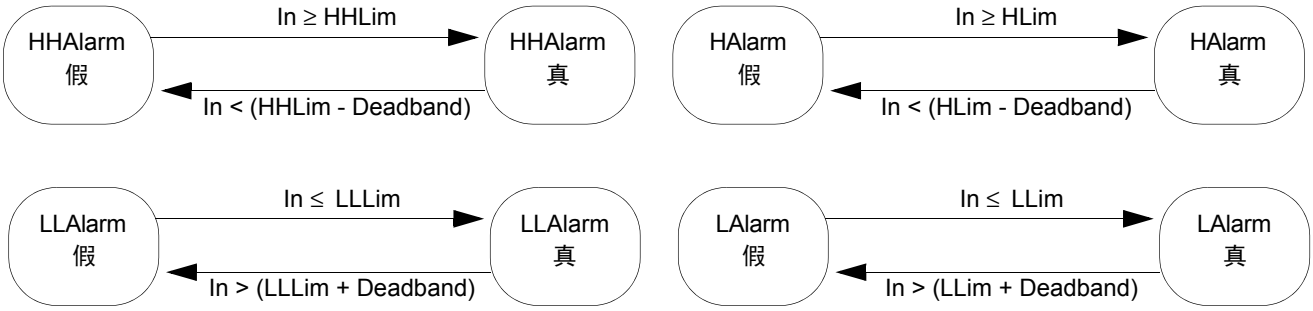
输入参数：	数据类型：	说明：
ROCPosLimit	REAL	输入信号正向 （增加）变化率 （单位 / 秒）的报警极限。设置 ROCPosLimit = 0 可禁用正向 ROC 报警。如果无效，则指令假定值为 0.0 并设置状态字中的相应位。 有效值 = 任意实数值 ≥ 0.0 缺省值 = 0.0
ROCNegLimit	REAL	输入信号反向 （减小）变化率 （单位 / 秒）的报警极限。设置 ROCPNegLimit = 0 可禁用反向 ROC 报警。如果无效，则指令假定值为 0.0 并设置状态字中的相应位。 有效值 = 任意实数值 ≥ 0.0 缺省值 = 0.0
ROCPeriod	REAL	用于计算变化率的时间周期 （单位为秒）。设置 ROCPeriod = 0 可禁用 ROC 报警并将输出 ROC 清零。如果无效，则指令假定值为 0.0 并设置状态字中的相应位。 有效值 = 任意实数值 ≥ 0.0 缺省值 = 0.0

输出参数：	数据类型：	说明：
EnableOut	BOOL	输出使能。
HHAlarm	BOOL	最高上限报警指示。 缺省值 = false
HAlarm	BOOL	上限报警指示。 缺省值 = false
LAlarm	BOOL	下限报警指示。 缺省值 = false
LLAlarm	BOOL	最低下限报警指示。 缺省值 = false
ROCPosAlarm	BOOL	正向变化率报警指示。 缺省值 = false
ROCNegAlarm	BOOL	反向变化率报警指示。 缺省值 = false
ROC	REAL	变化率输出。该输出要设置算术状态标志。
Status	DINT	功能块的状态。
InstructFault (Status.0)	BOOL	指令检测到如下运行错误。该错误不是次要或主要的控制器错误。检查其他状态位以确定错误类型。
DeadbandInv (Status.1)	BOOL	Deadband 值无效。
ROCPosLimitInv (Status.2)	BOOL	ROCPosLimit 值无效。
ROCNegLimitInv (Status.3)	BOOL	ROCNegLimit 值无效。
ROCPeriodInv (Status.4)	BOOL	ROCPeriod 值无效。

说明： ALM 指令可提供报警指示，用于最高上限、上限、下限、最低下限、正向变化率以及反向变化率。最高上限到最低下限值之间的报警有报警死区可用。而且允许使用自定义周期来计算变化率。

最高上限到最低下限值之间的报警

最高上限和最低下限报警算法将输入与报警极限及其加减死区的结果进行比较。



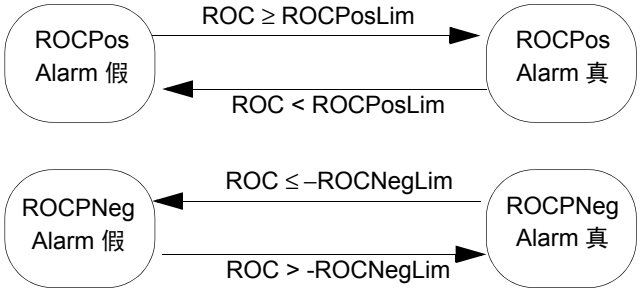
变化率报警

变化率 (ROC) 报警将 ROCPeriod 期间内输入的变化量与变化率报警极限进行比较。 ROCPeriod 规定了变化率报警的死区类型。例如，定义 ROC 报警极限为 2 癮 /s，计算周期为 100ms。如果您使用分辨率为 1 癮 的模拟量输入模块，则每当输入值变化时都会生成 ROC 报警，因为指令计算出的有效变化率为 10 癮 /s。然而，如果 ROCPeriod 为 1 秒，则仅当变化率确实超过 2 癮 /s 的极限时该指令才会报警。

ROC 报警根据以下公式计算变化率：

$$ROC = \frac{In(Now) - In(EndofpreviousROCPeriod)}{ROCPeriod}$$

指令在 ROCPeriod 结束时进行该计算。计算得出 ROC 后，即可通过以下方法确定报警状态：



监控 ALM 指令

ALM 指令具有操作员面板。了解更多详细信息，请参阅附录“功能块面板控件”。

算术状态标志： ROC 输出影响算术状态标志。

错误条件： 无

执行：

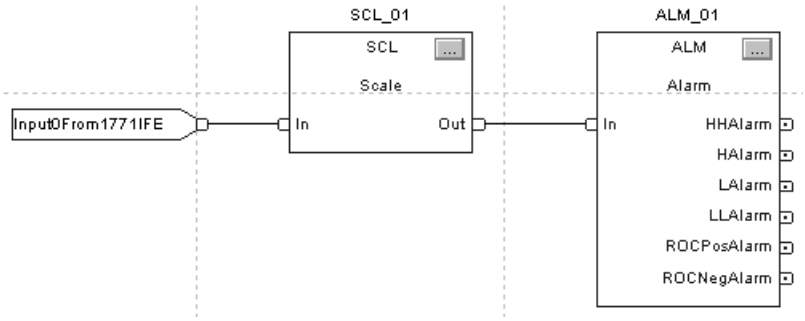
条件：	功能块动作：	结构化文本动作：
预扫描	无动作。	无动作。
指令首次扫描	所有报警输出清零。 累计消耗时间清零。	所有报警输出清零。 累计消耗时间清零。
指令首次执行	所有报警输出清零。 累计消耗时间清零。	所有报警输出清零。 累计消耗时间清零。
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。	不影响
EnableIn 已设置	指令执行。 EnableOut 已设置。	EnableIn 一直保持已设置状态。 指令执行。
后期扫描	无动作。	无动作。

示例： ALM 指令通常用于不支持内置报警功能的模拟量输入模块 （如 1771 I/O 模块）或根据变量计算结果生成报警的应用。在此例中，来自 1771-IFE 模块的模拟量输入先通过 (SCL) 指令标度为工程单位。 SCL 指令的*输出*作为 ALM 指令的输入信号，以决定是否设置报警。报警输出的参数值可用于程序和 （或）显示在操作员界面上。

结构化文本

```
SCL_01.In := Input0From1771IFE;
SCL(SCL_01);

ALM_01.In := SCL_01.Out;
ALARM(ALM_01);
```

功能块



离散型 2 态设备 (D2SD)

D2SD 指令控制只有两种可能状态 （如开 / 关、打开 / 关闭等）的离散设备。

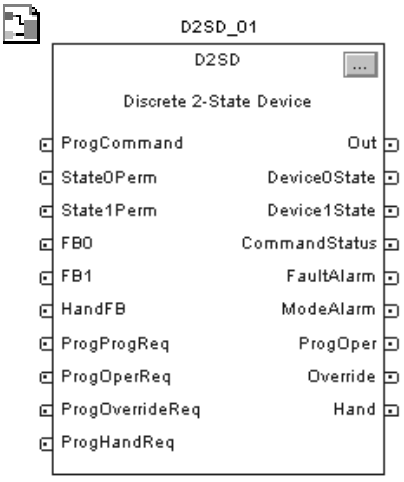
操作数：

```
D2SD(D2SD_tag);
```

结构文本

操作数：	类型：	格式：	说明：
D2SD 标记	DISCRETE_2STATE	结构	D2SD 结构

功能块



操作数：	类型：	格式：	说明：
D2SD 标记	DISCRETE_2STATE	结构	D2SD 结构

DISCRETE_2STATE 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果该位清零，则指令不执行且输出不更新。 如果该位已设置，则指令执行。 缺省状态为已设置。 **结构化文本：** 无效。指令执行。
ProgCommand	BOOL	当设备处于程序控制模式时，用于确定 CommandStatus。该位为已设置时，设备被设置为状态 1 ； 该位清零时，设备被设置为状态 0。 缺省状态为清零。
Oper0Req	BOOL	操作员状态 0 请求。当设备处于操作员控制状态时，通过操作员界面设置该位，使设备处于状态 0。 缺省状态为清零。
Oper1Req	BOOL	操作员状态 1 请求。当设备处于操作员控制状态时，通过操作员界面设置该位，使设备处于状态 1。 缺省状态为清零。

输入参数：	数据类型：	说明：
State0Perm	BOOL	允许状态 0。除非是在手动或越过模式，否则必须设置此输入，设备才能进入状态 0。该输入对已处于状态 0 的设备无效。 缺省状态为已设置。
State1Perm	BOOL	允许状态 1。除非是在手动或越过模式，否则必须设置此输入，设备才能进入状态 1。该输入对已处于状态 1 的设备无效。 缺省状态为已设置。
FB0	BOOL	D2SD 指令可用的第一路反馈输入。 缺省状态为清零。
FB1	BOOL	D2SD 指令可用的第二路反馈输入。 缺省状态为清零。
HandFB	BOOL	手动反馈输入。该输入来自现场站 （手动 / 停止 / 自动），显示对现场设备的状态请求。该位已设置时，请求现场设备进入状态 1 ；该位清零时，请求设备进入状态 0。 缺省状态为清零。
FaultTime	REAL	错误计时值。配置允许设备转换到新的命令状态的时间值（单位为秒）。设置 FaultTime = 0 禁用错误定时器。如果该值无效，则指令假定其值为零并设置状态字中的相应位。 有效值 = 任意浮点数 ≥ 0.0 缺省值 = 0.0
FaultAlarmLatch	BOOL	错误报警锁存输入。该位已设置且出现错误报警时，锁存错误报警。设置 FaultAlmUnlatch 或清零该位可解锁 FaultAlarm。 缺省状态为清零。
FaultAlmUnLatch	BOOL	错误报警解锁输入。当 FaultAlarmLatch 为已设置时，可设置该位解锁 FaultAlarm。指令将该输入清零。 缺省状态为清零。
OverrideOnInit	BOOL	请求在初始化时越过。如果该位已设置，则在指令首次扫描期间， 2 态设备处于操作员控制状态，设置 Override， Hand 清零。如果 ProgHandReq 已设置，则 Override 清零并设置 Hand。 缺省状态为清零。
OverrideOnFault	BOOL	错误时越过请求。如果出现错误报警时要求设备转至越过模式并进入 OverrideState，则对该位进行设置。清除错误报警后， 2 态设备处于操作员控制状态。 缺省状态为清零。
OutReverse	BOOL	对默认输出状态取反。默认输出状态在设备被命令进入状态 0 时为清零，进入状态 1 时为已设置。该位已设置时，则进入状态 0 时为已设置，进入状态 1 时为清零。 缺省状态为清零。
OverrideState	BOOL	越过状态输入。设备处于越过模式时，配置该位以指定设备的状态。该位已设置表示设备应转至状态 1，清零表示应转至状态 0。 缺省状态为清零。
FB0State0	BOOL	0 通道反馈状态 0 输入。配置当设备处于状态 0 时 FB0 的状态。 缺省状态为清零。
FB0State1	BOOL	0 通道反馈状态 1 输入。配置当设备处于状态 1 时 FB0 的状态。 缺省状态为清零。

输入参数：	数据类型：	说明：
FB1State0	BOOL	1 通道反馈状态 0 输入。配置当设备处于状态 0 时 FB1 的状态。 缺省状态为清零。
FB1State1	BOOL	1 通道反馈状态 1 输入。配置当设备处于状态 1 时 FB1 的状态。 缺省状态为清零。
ProgProgReq	BOOL	程序请求进入程序控制模式。通过用户程序设置来请求程序控制。如果 ProgOperReq 已设置，该请求被忽略。保持设置状态并将 ProgOperReq 清零，可锁定指令进入程序控制状态。 缺省状态为清零。
ProgOperReq	BOOL	程序请求进入操作员控制模式。通过用户程序设置来请求操作员控制。保持设置状态可锁定指令进入操作员控制状态。 缺省状态为清零。
ProgOverrideReq	BOOL	程序请求进入越过模式。通过用户程序设置来请求设备进入越过模式。如果 ProgHandReq 已设置，该请求被忽略。 缺省状态为清零。
ProgHandReq	BOOL	程序请求进入手动模式。通过用户程序设置来请求设备进入手动模式。 缺省状态为清零。
OperProgReq	BOOL	操作员请求进入程序控制模式。通过操作员界面设置来请求程序控制。指令将该输入清零。 缺省状态为清零。
OperOperReq	BOOL	操作员请求进入操作员控制模式。通过操作员界面设置来请求操作员控制。指令将该输入清零。 缺省状态为清零。
ProgValueReset	BOOL	复位程序控制值。已设置时，每次执行指令后将所有的程序请求输入清零。 缺省状态为清零。

输出参数：	数据类型：	说明：
EnableOut	BOOL	输出使能。
Out	BOOL	2 态指令的输出。
Device0State	BOOL	设备状态 0 输出。命令设备进入状态 0 且反馈指示设备确实处于状态 0 时，设置该位。
Device1State	BOOL	设备状态 1 输出。命令设备进入状态 1 且反馈指示设备确实处于状态 1 时，设置该位。
CommandStatus	BOOL	命令状态输出。命令设备进入状态 1 时设置该位，命令设备进入状态 0 时该位清零。
FaultAlarm	BOOL	错误报警输出。如果命令设备进入新状态，但在 FaultTime 时限内未收到指示到达新状态的反馈，则设置该位。如果到达命令状态后，反馈突然指示设备不再处于命令状态，此时也会进行设置。
ModeAlarm	BOOL	模式报警输出。如果设备处于操作员控制状态，而某程序命令设备转至与操作员当前所指定不同的状态时，则设置该位。此报警用于指示设备处于操作员控制状态。
ProgOper	BOOL	程序 / 操作员控制指示。处于程序控制状态时为已设置。处于操作员控制时清零。
Override	BOOL	越过模式。设备处于越过时设置。

输出参数：	数据类型：	说明：
Hand	BOOL	手动模式。设备处于手动模式时为已设置。
Status	DINT	功能块的状态。
InstructFault (Status.0)	BOOL	指令检测到如下运行错误。该错误不是次要或主要的控制器错误。检查其他状态位以确定错误类型。
FaultTimeInv (Status.1)	BOOL	FaultTime 值无效。该指令设置 FaultTime = 0。
OperReqInv (Status.2)	BOOL	两个操作员状态请求位均为已设置。

说明： D2SD 指令控制只有两种可能状态 （如开 / 关、打开 / 关闭等）的离散设备。典型的离散设备包括电动机、泵和电磁阀。

监控 D2SD 指令

D2SD 指令具有操作员面板。了解有关更多详细信息，请参阅附录“功能块面板控件”。

算术状态标志： 不影响算术状态标志。

错误条件： 无

执行：

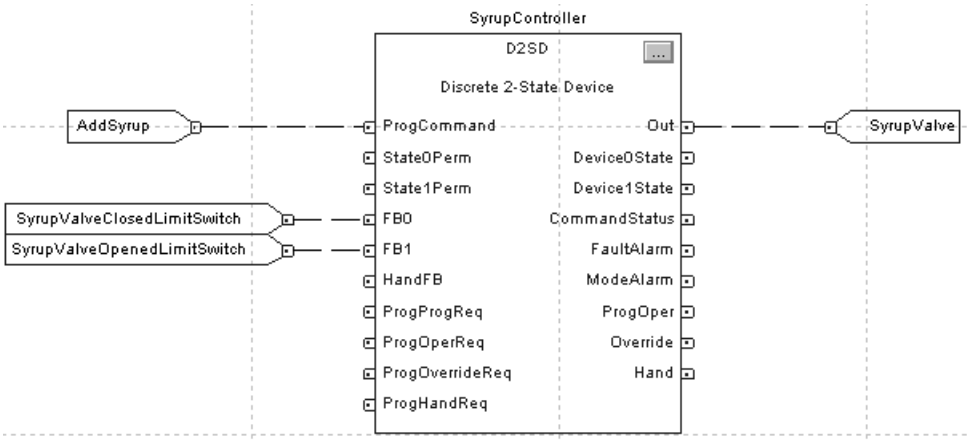
条件：	功能块动作：	结构化文本动作：
预扫描	无动作。	无动作。
指令首次扫描	错误定时器清零。 ModeAlarm 清零。 所有操作员请求输入均清零。 如果 ProgValueReset 已设置，则所有程序请求输入均清零。 OverrideOnInit 为已设置时， ProgOper 清零 （操作员控制）。 如果 ProgHandReq 清零并设置 OverrideOnInit，则 Hand 清零且 Override 已设置 （越过模式）。 如果 ProgHandReq 已设置，则 Hand 已设置且 Override 清零 （手动模式）。	
指令首次执行	ProgOper 和 CommandStatus 清零。	ProgOper 和 CommandStatus 清零。
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。	na
EnableIn 已设置	指令执行。 EnableOut 已设置。	EnableIn 一直保持已设置状态。 指令执行。
后期扫描	无动作。	无动作。

示例： D2SD 指令通常用于控制具有两种可能状态 （开 / 关或打开 / 闭合）的设备，如泵或电磁阀。在此例中， D2SD 指令控制电磁阀向将玉米糖浆注入配料罐。 D2SD 指令处于程序控制状态时，当 AddSyrup 输入为已设置，此时阀门打开。必要时操作员也可通过操作员控制模式来打开或关闭阀门。此例中的电磁阀装有限位开关，用于指示何时阀门完全闭合或打开。限位开关连接至 FB0 和 FB1 反馈输入。这样即可实现当电磁阀在设定的 *FaultTime* 内没有到达所指定的位置时，D2SD 指令便生成一个 *FaultAlarm*。

结构化文本

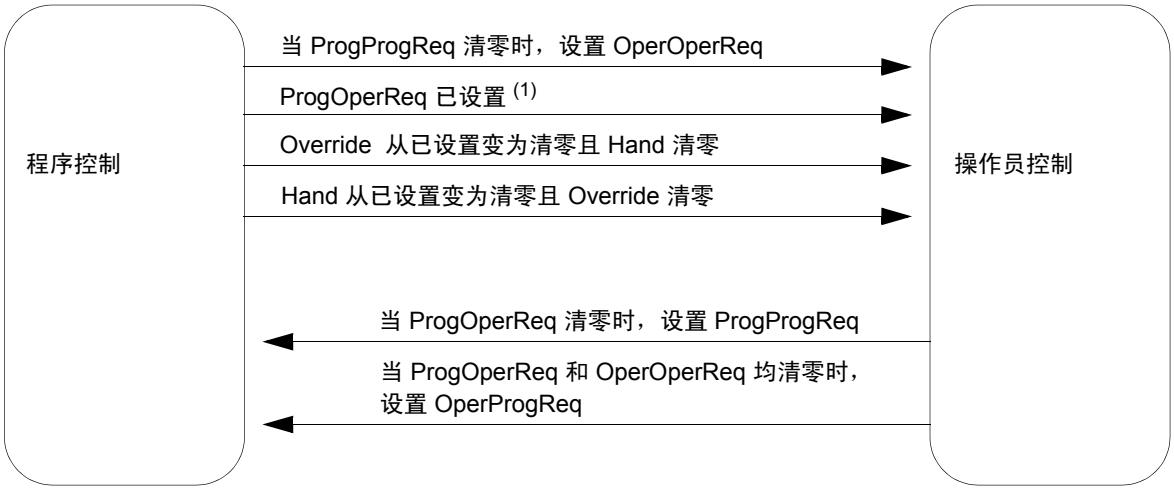
```
SyrupController.ProgCommand := AddSyrup;
SyrupController.FB0 := SyrupValveClosedLimitSwitch;
SyrupController.FB1 := SyrupValveOpenedLimitSwitch;
D2SD(SyrupController);

SyrupValve := SyrupController.Out;
```

功能块



在程序控制和操作员控制之间进行切换

下图演示了 D2SD 指令如何在程序控制和操作员控制之间进行切换。

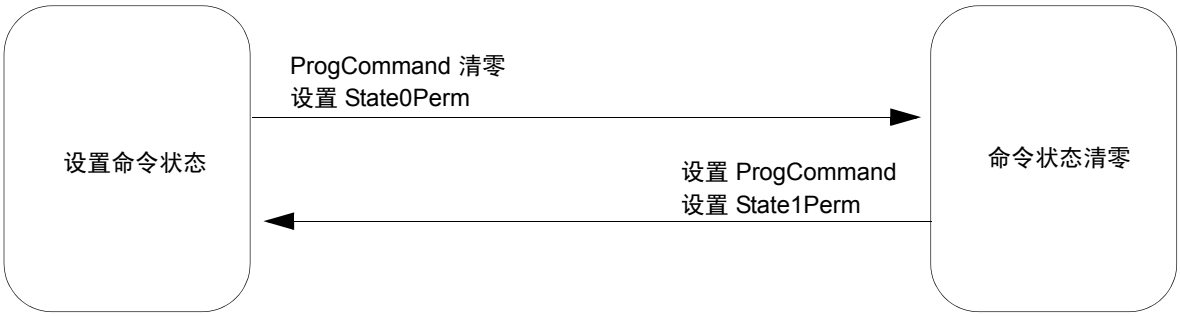


(1) 当 ProgOperReq 为已设置时，指令仍处在操作员控制模式。

了解有关程序控制和操作员控制的更多信息，请参阅第 B-13 页。

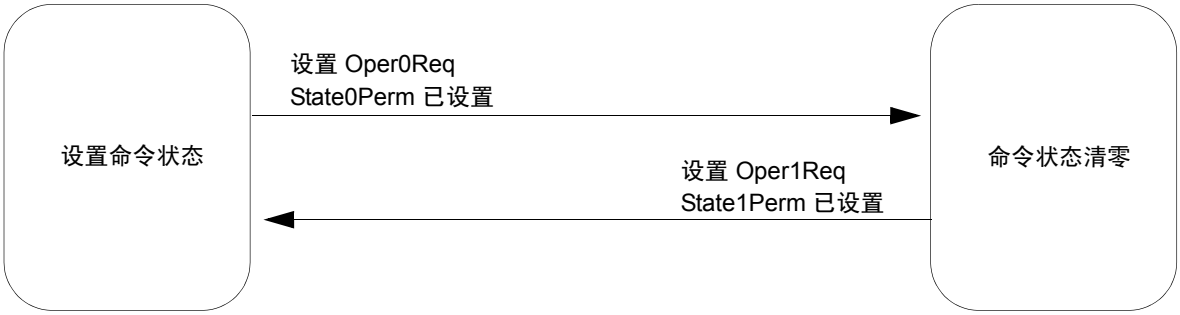
程序控制中的命令状态

下图所示为在程序控制状态下 D2SD 指令如何执行。



操作员控制中的命令状态

下图所示为在操作员控制状态下 D2SD 指令如何执行。



如果 Oper0Req 和 Oper1Req 均已设置：

- 指令设置状态字中的相应位
- 如果 Override 和 Hand 均清零，则指令保持之前的状态。

每次执行指令后，该指令：

- 将所有的操作员请求输入清零
- 如果 ProgValueReset 已设置，则所有程序请求输入均清零

手动或越过模式

下表描述了 D2SD 指令如何确定手动或越过操作模式

ProgHandReq：	ProgOverrideReq：	FaultAlarm 和 OverrideOnFault：	说明：
设置	任一	任一	手动模式 Hand 已设置 Override 清零
清零	设置	任一	越过模式 Hand 清零 Override 已设置
清零	任一	设置	越过模式 Hand 清零 Override 已设置

当指令为越过模式时， CommandStatus = OverrideState

当指令为手动模式时， CommandStatus = HandFB

输出状态

D2SD 输出状态取决于以下命令状态。

命令状态：	输出状态：
清零	如果 OutReverse 清零，则输出清零 如果 OutReverse 已设置，则输出为已设置
设置	如果 OutReverse 清零，则输出为已设置 如果 OutReverse 已设置，则输出清零
清零且 FB0 = FB0State0 且 FB1 = FB1State0	错误定时器已停止并清零 Device0State 已设置
设置，且 FB0 = FB0State1 且 FB1 = FB1State1	错误定时器已停止并清零 Device1State 已设置

错误报警条件

D2SD 指令会检查以下错误报警条件。

错误报警条件源于：	规则：
命令改变设备状态，但是反馈指示在 FaultTime 内设备未达到预期状态。	当 $CommandStatus_n \neq CommandStatus_{n-1}$ 时，启动错误定时器 当错误定时器计时完成且 FaultTime > 0.0 时，设置 FaultAlarm
未经命令，设备意外离开某个状态 （根据反馈指示）。	设置 FaultAlarm，当错误定时器未定时并满足以下条件之一时： CommandStatus 清零且 Device0State 清零 CommandStatus 已设置且 Device1State 清零

满足以下条件之一时， FaultAlarm 清零：

- CommandStatus 清零且 Device0State 已设置
- CommandStatus 已设置且 Device1State 已设置
- FaultTime ≤ 0

FaultAlarmLatch 为已设置时， FaultAlarm 无法清零，除非 FaultAlmUnlatch 已设置且当前无任何错误。

模式报警条件

模式报警提醒操作员还有设备处于操作员控制状态。在操作员控制模式下，仅在程序试图改变设备的操作员控制状态时才生成模式报警。如是操作员配置该设备为操作员模式并改变其状态时，则不会生成模式报警。 D2SD 指令遵循以下规则来检查模式报警条件。

ModeAlarm：	**条件：**
设置	$ProgCommand_n \neq ProgCommand_{n-1}$ **且** $ProgCommand_n \neq CommandStatus$
清零	ProgCommand = CommandStatus **或** 设备处于越过、手动或程序控制模式时

离散 3 态设备 (D3SD)

D3SD 指令控制具有三种可能状态的离散设备，如快 / 慢 / 停、正转 / 停止 / 反转等。

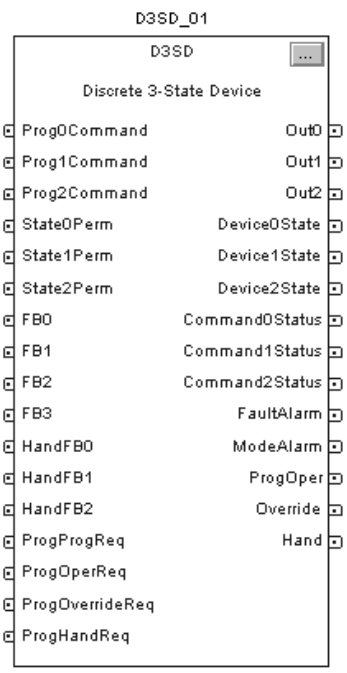
操作数：

结构化文本

```
D3SD(D3SD_tag);
```

操作数：	类型：	格式：	说明：
D3SD 标记	DISCRETE_3STATE	结构	D3SD 结构

功能块



操作数：	类型：	格式：	说明：
D3SD 标记	DISCRETE_3STATE	结构	D2SD 结构

DISCRETE_3STATE 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果该位清零，则指令不执行且输出不更新。 如果该位已设置，则指令执行。 缺省状态为已设置。 **结构化文本：** 无效。指令执行。
Prog0Command	BOOL	程序状态 0 命令。设备处于程序控制时，该输入决定设备状态。如果该位已设置，则命令设备进入状态 0。 缺省状态为清零。
Prog1Command	BOOL	程序状态 1 命令。设备处于程序控制时，该输入决定设备状态。如果该位已设置，则命令设备进入状态 1。 缺省状态为清零。
Prog2Command	BOOL	程序状态 2 命令。设备处于程序控制时，该输入决定设备状态。如果该位已设置，则命令设备进入状态 2。 缺省状态为清零。

输入参数：	数据类型：	说明：
Oper0Req	BOOL	操作员状态 0 请求。当设备处于操作员控制状态时，通过操作员界面设置该位，使设备处于状态 0。 缺省状态为清零。
Oper1Req	BOOL	操作员状态 1 请求。当设备处于操作员控制状态时，通过操作员界面设置该位，使设备处于状态 1。 缺省状态为清零。
Oper2Req	BOOL	操作员状态 2 请求。当设备处于操作员控制状态时，通过操作员界面设置该位，使设备处于状态 2。 缺省状态为清零。
State0Perm	BOOL	允许状态 0。除非是在手动或越过模式，否则必须设置此输入，设备才能进入 0 状态。该输入对已处于状态 0 的设备无效。 缺省状态为已设置。
State1Perm	BOOL	允许状态 1。除非是在手动或越过模式，否则必须设置此输入，设备才能进入 1 状态。该输入对已处于状态 1 的设备无效。 缺省状态为已设置。
State2Perm	BOOL	允许状态 2。除非是在手动或越过模式，否则必须设置此输入，设备才能进入 2 状态。该输入对已处于状态 2 的设备无效。 缺省状态为已设置。
FB0	BOOL	D2SD 指令可用的第一路反馈输入。 缺省状态为清零。
FB1	BOOL	D2SD 指令可用的第二路反馈输入。 缺省状态为清零。
FB2	BOOL	D2SD 指令可用的第三路反馈输入。 缺省状态为清零。
FB3	BOOL	D2SD 指令可用的第四路反馈输入。 缺省状态为清零。
HandFB0	BOOL	手动反馈状态 0。该输入来自现场站 （手动 / 停止 / 自动），显示对现场设备的状态请求。该位已设置表示请求现场设备进入状态 0；清零表示请求现场设备进入其他状态。 缺省状态为清零。
HandFB1	BOOL	手动反馈状态 1。该输入来自现场站 （手动 / 停止 / 自动），显示对现场设备的状态请求。该位已设置表示请求现场设备进入状态 1；清零表示请求现场设备进入其他状态。 缺省状态为清零。
HandFB2	BOOL	手动反馈状态 2。该输入来自现场站 （手动 / 停止 / 自动），显示对现场设备的状态请求。该位已设置表示请求现场设备进入状态 2；清零表示请求现场设备进入其他状态。 缺省状态为清零。
FaultTime	REAL	错误计时值。配置允许设备转换到新的命令状态的时间值 （单位为秒）。设置 FaultTime = 0 禁用错误定时器。如果该值无效，则指令假定其值为零并设置状态字中的相应位。 有效值 = 任意浮点数 ≥ 0.0 缺省值 = 0.0

输入参数：	数据类型：	说明：
FaultAlarmLatch	BOOL	错误报警锁存输入。该位已设置且出现错误报警时，锁存错误报警。 设置 FaultAlmUnlatch 或清零该位可解锁 FaultAlarm。 缺省状态为清零。
FaultAlmUnLatch	BOOL	错误报警解锁输入。当 FaultAlarmLatch 为已设置时，可设置该输入解锁 FaultAlarm。指令将该输入清零。 缺省状态为清零。
OverrideOnInit	BOOL	请求在初始化时越过。如果该位已设置，则在指令首次扫描期间，指令处于操作员控制状态，且 Override 为已置位， Hand 清零。如果 ProgHandReq 已设置，则 Override 清零并设置 Hand。 缺省状态为清零。
OverrideOnFault	BOOL	错误时越过请求。如果出现错误报警时要求设备转至越过模式并进入 OverrideState，则对该位进行设置。清除错误报警后，指令处于操作员控制状态。 缺省状态为清零。
Out0State0	BOOL	输出 0 状态 0 输入。该值决定了设备处于状态 0 时 Output0 的值。 缺省状态为清零。
Out0State1	BOOL	输出 0 状态 1 输入。该值决定了设备处于状态 1 时 Output0 的值。 缺省状态为清零。
Out0State2	BOOL	输出 0 状态 2 输入。该值决定了设备处于状态 2 时 Output0 的值。 缺省状态为清零。
Out1State0	BOOL	输出 1 状态 0 输入。该值决定了设备处于状态 0 时 Output1 的值。 缺省状态为清零。
Out1State1	BOOL	输出 1 状态 1 输入。该值决定了设备处于状态 1 时 Output1 的值。 缺省状态为清零。
Out1State2	BOOL	输出 1 状态 2 输入。该值决定了设备处于状态 2 时 Output1 的值。 缺省状态为清零。
Out2State0	BOOL	输出 2 状态 0 输入。该值决定了设备处于状态 0 时 Output2 的值。 缺省状态为清零。
Out2State1	BOOL	输出 2 状态 1 输入。该值决定了设备处于状态 1 时 Output2 的值。 缺省状态为清零。
Out2State2	BOOL	输出 2 状态 2 输入。该值决定了设备处于状态 2 时 Output2 的值。 缺省状态为清零。
OverrideState	DINT	越过状态输入。在越过模式中，设置该输入以指定设备的状态。 **值：** **指示：** 2 设备应进入状态 2 1 设备应进入状态 1 0 设备应进入状态 0 输入值无效时，将设置状态字中的相应位，防止指令进入越过模式。 有效值 = 0 到 2 缺省值 = 0
FB0State0	BOOL	0 通道反馈状态 0 输入。该值决定了设备处于状态 0 时 FB0 的期望值。 缺省状态为清零。
FB0State1	BOOL	0 通道反馈状态 1 输入。该值决定了设备处于状态 1 时 FB0 的期望值。 缺省状态为清零。

输入参数：	数据类型：	说明：
FB0State2	BOOL	0 通道反馈状态 2 输入。该值决定了设备处于状态 2 时 FB0 的期望值。 缺省状态为清零。
FB1State0	BOOL	1 通道反馈状态 0 输入。当设备处于状态 0 时，该输入值决定了 FB1 的期望值。 缺省状态为清零。
FB1State1	BOOL	1 通道反馈状态 1 输入。该值决定了设备处于状态 1 时 FB1 的期望值。 缺省状态为清零。
FB1State2	BOOL	1 通道反馈状态 2 输入。该值决定了设备处于状态 2 时 FB1 的期望值。 缺省状态为清零。
FB2State0	BOOL	2 通道反馈状态 0 输入。该值决定了设备处于状态 0 时 FB2 的期望值。 缺省状态为清零。
FB2State1	BOOL	2 通道反馈状态 1 输入。该值决定了设备处于状态 1 时 FB2 的期望值。 缺省状态为清零。
FB2State2	BOOL	2 通道反馈状态 2 输入。该值决定了设备处于状态 2 时 FB2 的期望值。 缺省状态为清零。
FB3State0	BOOL	3 通道反馈状态 0 输入。该值决定了设备处于状态 0 时 FB3 的期望值。 缺省状态为清零。
FB3State1	BOOL	3 通道反馈状态 1 输入。该值决定了设备处于状态 1 时 FB3 的期望值。 缺省状态为清零。
FB3State2	BOOL	3 通道反馈状态 2 输入。该值决定了设备处于状态 2 时 FB3 的期望值。 缺省状态为清零。
ProgProgReq	BOOL	程序请求进入程序控制模式。通过用户程序设置来请求程序控制。如果 ProgOperReq 已设置，该请求被忽略。保持该位为已设置并将 ProgOperReq 清零，可锁定指令进入程序控制状态。 缺省状态为清零。
ProgOperReq	BOOL	程序请求进入操作员控制模式。通过用户程序设置来请求设备进入操作员模式。保持该位为已设置，可锁定指令进入操作员控制状态。 缺省状态为清零。
ProgOverrideReq	BOOL	程序越过请求。通过用户程序设置来请求设备进入越过模式。如果 ProgHandReq 已设置，该请求被忽略。 缺省状态为清零。
ProgHandReq	BOOL	程序请求进入手动模式。通过用户程序设置来请求设备进入手动模式。 缺省状态为清零。
OperProgReq	BOOL	操作员请求进入程序控制模式。通过操作员界面设置来请求程序控制。指令将该输入清零。 缺省状态为清零。
OperOperReq	BOOL	操作员请求进入操作员控制模式。通过操作员界面设置来请求操作员控制。指令将该输入清零。 缺省状态为清零。
ProgValueReset	BOOL	复位程序控制值。已设置时，每次执行指令后将所有的程序请求输入清零。 缺省状态为清零。

输出参数：	数据类型：	说明：
EnableOut	BOOL	输出使能。
Out0	BOOL	指令的第一路输出。
Out1	BOOL	指令的第二路输出。
Out2	BOOL	指令的第三路输出。
Device0State	BOOL	设备状态 0 输出。命令设备进入状态 0 且反馈指示设备确实处于状态 0 时，设置该位。
Device1State	BOOL	设备状态 1 输出。命令设备进入状态 1 且反馈指示设备确实处于状态 1 时，设置该位。
Device2State	BOOL	设备状态 2 输出。命令设备进入状态 2 且反馈指示设备确实处于状态 2 时，设置该位。
Command0Status	BOOL	设备命令状态 0。该位在设备被命令进入状态 0 时为已设置，进入其他状态时清零。
Command1Status	BOOL	设备命令状态 1。该位在设备被命令进入状态 1 时为已设置，进入其他状态时清零。
Command2Status	BOOL	设备 2 命令状态。该位在设备被命令进入状态 2 时为已设置，进入其他状态时清零。
FaultAlarm	BOOL	错误报警输出。如果命令设备进入新状态，但在 FaultTime 时限内未收到指示到达新状态的反馈，则设置该位。如果到达命令状态后，反馈突然指示设备不再处于命令状态，此时也会进行设置。
ModeAlarm	BOOL	模式报警输出。如果设备处于操作员控制状态，而某程序命令设备转至与操作员当前所指定不同的状态时，则设置该位。此报警用于指示设备处于操作员控制状态。
ProgOper	BOOL	程序 / 操作员控制指示。处于程序控制状态时为已设置。处于操作员控制时清零。
Override	BOOL	越过模式。设备处于越过模式时为已设置。
Hand	BOOL	手动模式。设备处于手动模式时为已设置。
Status	DINT	功能块的状态。
InstructFault (Status.0)	BOOL	指令检测到如下运行错误。该错误不是次要或主要的控制器错误。检查其他状态位以确定错误类型。

输出参数：	数据类型：	说明：
FaultTimeInv (Status.1)	BOOL	FaultTime 值无效。该指令设置 FaultTime = 0。
OverrideStateInv (Status.2)	BOOL	Override 值超出范围
ProgCommandInv (Status.3)	BOOL	同时设置多个程序状态命令位。
OperReqInv (Status.4)	BOOL	同时设置多个操作员状态请求位。
HandCommandInv (Status.5)	BOOL	同时设置多个手动状态请求位。

说明： D3SD 指令控制具有三种可能状态的离散设备，如快 / 慢 / 停、正转 / 停止 / 反转等。具有此种特性的典型离散设备包括进料系统以及可逆电机等。

监控 D3SD 指令

D3SD 指令具有操作员面板。了解有关更多详细信息，请参阅附录“功能块面板控件”。

算术状态标志： 不影响算术状态标志。

错误条件： 无

执行：

条件：	功能块动作：	结构化文本动作：
预扫描	无动作。	无动作。
指令首次扫描	错误定时器清零。 ModeAlarm 清零。 所有操作员请求输入均清零。 如果 ProgValueReset 已设置，则所有程序请求输入均清零。 OverrideOnInit 为已设置时， ProgOper 清零 （操作员控制）。 如果 ProgHandReq 清零并设置 OverrideOnInit，则 Hand 清零且 Override 已设置 （越过模式）。 如果 ProgHandReq 已设置，则 Hand 已设置且 Override 清零 （手动模式）。	
指令首次执行	ProgOper 和 CommandStatus 清零。	ProgOper 和 CommandStatus 清零。
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。	不影响
EnableIn 已设置	指令执行。 EnableOut 已设置。	EnableIn 一直保持已设置状态。 指令执行。
后期扫描	无动作。	无动作。

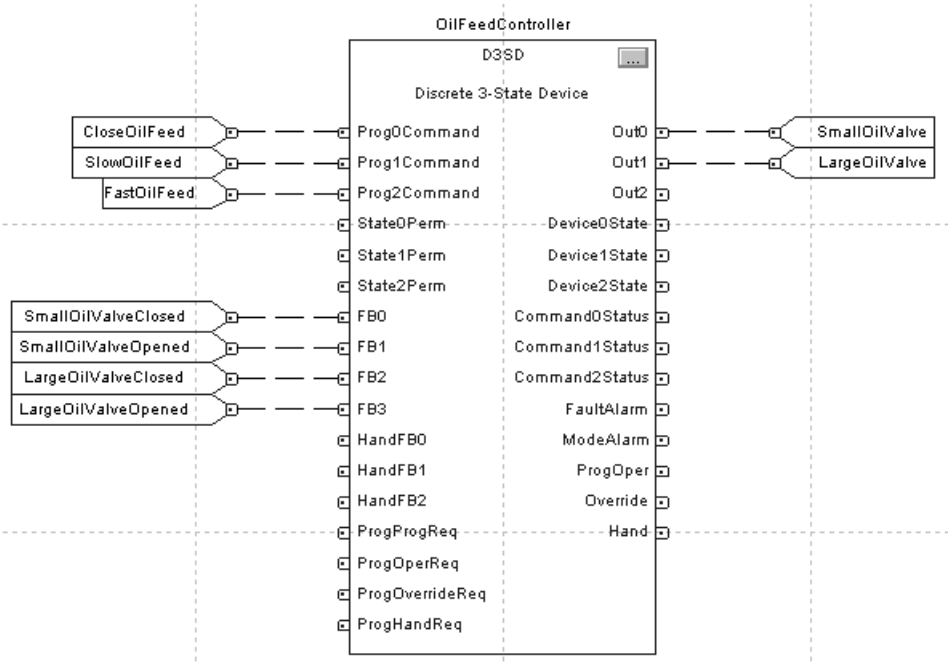
示例： D3SD 指令通常用于控制具有 3 种状态的设备，如进料系统 （高速 / 低速 / 关）。在此例中，D3SD 指令控制由一对电磁阀构成的进料系统将植物油注入配料罐。其中一个阀门连接配料罐上的大口径进料管，另一个阀门接在平行的小口径进料管上。初次注入油时，命令 D3SD 指令控制系统处于高速进料状态 (state 2)，此时两个阀门同时开启。当注油量接近目标值时，命令 D3SD 指令切换系统至低速进料状态 (state 1)，此时大阀门关闭，小阀门保持打开状态。当已注油量达到目标值时，命令 D3SD 指令关闭系统 (state 0)，则两个阀门都将被关闭。

D3SD 指令处于程序控制状态时，阀门的开启取决于 CloseOilFeed、SlowOilFeed 和 FastOilFeed 输入。必要时操作员也可对进料系统实施操作员控制。此例中的电磁阀装有限位开关，用于指示何时阀门完全关闭或打开。限位开关连接到 FB0、 FB1、 FB2 和 FB3 反馈输入。这样即可实现当电磁阀在设定的 *FaultTime* 内没有到达所指定的位置时，D3SD 指令便生成一个 *FaultAlarm*。

结构化文本

```
OilFeedController.Prog0Command := CloseOilFeed;
OilFeedController.Prog1Command := SlowOilFeed;
OilFeedController.Prog2Command := FastOilFeed;
OilFeedController.FB0 := SmallOilValveClosed;
OilFeedController.FB1 := SmallOilValveOpened;
OilFeedController.FB2 := LargeOilValveClosed;
OilFeedController.FB3 := LargeOilValveOpened;
D3SD(OilFeedController);

SmallOilValve := OilFeedController.Out0;
LargeOilValve := OilFeedController.Out1;
```

功能块



在程序控制和操作员控制之间进行切换

下图演示了 D3SD 指令如何在程序控制与操作员控制之间进行切换。

(1) 当 ProgOperReq 为已设置时，指令仍处在操作员控制模式。

了解有关程序控制和操作员控制的更多详细信息，请参阅第 B-13 页。

程序控制中的命令状态

下表描述了 D3SD 指令在程序控制状态下如何执行：

Prog0 Command：	Prog1 Command：	Prog2 Command：	State0 Perm：	State1 Perm：	State2 Perm：	说明：
清零	清零	设置	任一	任一	设置	Command0Status 清零 Command1Status 清零 Command2Status 已设置
清零	设置	清零	任一	设置	任一	Command0Status 清零 Command1Status 已设置 Command2Status 清零
设置	清零	清零	设置	任一	任一	Command0Status 已设置 Command1Status 清零 Command2Status 清零

当多个程序命令输入均为已设置时：

- 指令设置状态字中的相应位
- 如果 Override 和 Hand 均清零，则指令保持之前的状态

操作员控制中的命令状态

下表描述了在操作员控制状态下 D3SD 指令如何执行。

Oper0Req：	Oper1Req：	Oper2Req：	State0 Perm：	State1 Perm：	State2 Perm：	说明：
清零	清零	设置	任一	任一	设置	Command0Status 清零 Command1Status 清零 Command2Status 已设置
清零	设置	清零	任一	设置	任一	Command0Status 清零 Command1Status 已设置 Command2Status 清零
设置	清零	清零	设置	任一	任一	Command0Status 已设置 Command1Status 清零 Command2Status 清零

当多个操作员命令输入均为已设置时：

- 指令设置状态字中的相应位
- 如果 Override 和 Hand 均清零，则指令保持之前的状态

每次执行指令后，该指令：

- 将所有的操作员请求输入清零
- 如果 ProgValueReset 已设置，则所有程序请求输入均清零

手动或越过模式

下表描述了 D3SD 指令如何确定手动或越过操作模式

ProgHandReq：	ProgOverrideReq：	FaultAlarm 和 OverrideOnFault：	说明：
设置	任一	任一	手动模式 Hand 已设置 Override 清零
清零	设置	任一	越过模式 Hand 清零 Override 已设置
清零	任一	设置	越过模式 Hand 清零 Override 已设置

当 Override 为已设置时，其优先权高于程序控制和操作员控制。下表描述了越过模式对命令状态的影响。

越过模式：	越过模式状态：	说明：
设置	2	Command0Status 清零 Command1Status 清零 Command2Status 已设置
设置	1	Command0Status 清零 Command1Status 已设置 Command2Status 清零
设置	0	Command0Status 已设置 Command1Status 清零 Command2Status 清零

如果 OverrideState 无效，则指令设置状态字中的相应位且不进入越过状态。

当 Hand 为已设置时，其优先权高于程序控制和操作员控制。下表描述了手动模式对命令状态的影响。

Hand:	HandFB0:	HandFB1:	HandFB2:	说明：
设置	清零	清零	设置	Command0Status 清零 Command1Status 清零 Command2Status 已设置
设置	清零	设置	清零	Command0Status 清零 Command1Status 已设置 Command2Status 清零
设置	设置	清零	清零	Command0Status 已设置 Command1Status 清零 Command2Status 清零

当多个 HandFB 输入均为已设置时，指令设置状态字中的相应位。如果 Hand 为已置位，则指令保持之前的状态。

输出状态

D3SD 输出状态取决于以下命令状态。

命令状态：	输出状态：
Command0Status 已设置	Out0 = Out0State0 Out1 = Out1State0 Out2 = Out2State0
Command0Status 已设置且 FB0 = FB0State0 且 FB1 = FB1State0 且 FB2 = FB2State0 且 FB3 = FB3State0	错误定时器停止并清零 Device0State 已设置
Command1Status 已设置	Out0 = Out0State1 Out1 = Out1State1 Out2 = Out2State1
Command1Status 已设置且 FB0 = FB0State1 且 FB1 = FB1State1 且 FB2 = FB2State1 且 FB3 = FB3State1	错误定时器停止并清零 Device1State 已设置
Command2Status 已设置	Out0 = Out0State2 Out1 = Out1State2 Out2 = Out2State2
Command2Status 已设置且 FB0 = FB0State2 且 FB1 = FB1State2 且 FB2 = FB2State2 且 FB3 = FB3State2	错误定时器停止并清零 Device2State 已设置

错误报警条件

D3SD 指令检查以下错误报警条件。

错误报警条件源于：	规则：
命令改变设备状态，但是反馈指示在 FaultTime 内设备未达到预期状态。	当 $Command0Status_n \neq Command0Status_{n-1}$ 或 $Command1Status_n \neq Command1Status_{n-1}$ 或 $Command2Status_n \neq Command2Status_{n-1}$ 时，启动错误定时器 当错误定时器计时完成且 FaultTime > 0.0 时，设置 FaultAlarm
未经命令，设备意外离开某个状态 （根据反馈指示）。	设置 FaultAlarm，当错误定时器未定时并满足以下条件之一时： Command0Status 已设置且 Device0State 清零 Command1Status 已设置且 Device1State 清零 Command2Status 已设置且 Device2State 清零

如果未出现错误，满足以下条件之一时， FaultAlarm 清零：

- Command0Status 已设置且 Device0State 已设置
- Command1Status 已设置且 Device1State 已设置
- Command2Status 已设置且 Device2State 已设置
- FaultTime ≤ 0

FaultAlarmLatch 为已设置时， FaultAlarm 无法清零，除非 FaultAlmUnlatch 已设置且当前无任何错误。

模式报警条件

模式报警提醒操作员还有设备处于操作员控制状态。在操作员控制模式下，仅在程序试图改变设备的操作员控制状态时才生成模式报警。如是操作员配置该设备为操作员模式并改变其状态时，则不会生成模式报警。 D3SD 指令遵循以下规则来检查模式报警条件。

模式报警：	条件：
设置	Prog2Command ≠ $Prog2Command_{n-1}$ **且** Prog2Command ≠ Command2Status **或** Prog1Command ≠ $Prog1Command_{n-1}$ **且** Prog1Command ≠ Command1Status **或** Prog0Command ≠ $Prog0Command_{n-1}$ **且** Prog0Command ≠ Command0Status
清零	Prog2Command = Command2Status **且** Prog1Command = Command1Status **且** Prog0Command = Command0Status **或** 设备处于越过、手动或程序控制模式时

死区时间 (DEDT)

DEDT 使单路输入信号延时。选择死区时间延迟量。

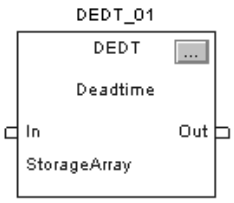
操作数：

结构文本

```
DEDT(DEDT_tag,storage);
```

操作数：	类型：	格式：	说明：
DEDT 标记	DEADTIME	结构	DEDT 结构
存储	REAL	数组	死区时间缓冲区

功能块



操作数：	类型：	格式：	说明：
DEDT 标记	DEADTIME	结构	DEDT 结构
存储	REAL	数组	死区时间缓冲区

DEADTIME 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果该位清零，则指令不执行且输出不更新。 如果该位已设置，则指令执行。 缺省状态为已设置。 **结构化文本：** 无效。指令执行。
In	REAL	输入到指令的模拟量信号。 有效值 = 浮点数 缺省值 = 0.0
InFault	BOOL	输入信号错误指示。如从模拟量输入信号中读取输入值，则 InFault 由该模拟量输入信号的错误状态控制。如果该位已设置，表示输入信号有错误，指令设置状态字中的相应位，且不执行控制算法并使输出保持。 缺省状态为清零。 该位清零表示输入信号正常。
死区时间	REAL	指令的死区时间输入。以秒为单位输入死区时间。如果该值无效，则指令假定其值为零并设置状态字中的相应位。 有效值 = 0.0 到 （StorageArray 长度 * DeltaT） 缺省值 = 0.0
Gain	REAL	指令的增益输入。需将 In 值与该值相乘。可用于对处理增益的仿真。 有效值 = 浮点数 缺省值 = 1.0
Bias	REAL	指令的偏移输入。需将 In 值乘以 Gain 值并与该值相加。可用于对外界条件的仿真。 有效值 = 浮点数 缺省值 = 0.0

输入参数：	数据类型：	说明：
TimingMode	DINT	选择定时执行模式。 **值：** **说明：** 0 周期模式 1 过采样模式 2 实时采样模式 有效值 = 0 到 2 缺省值 = 0 有关定时模式的更多信息，请参阅附录 “功能块属性”。
OversampleDT	REAL	过采样模式下的运行时间。 有效值 = 0 到 4194.303 秒 缺省值 = 0
RTSTime	DINT	实时采样模式下的模块更新周期 有效值 = 1 到 32,767 毫秒 缺省值 = 1
RTSTimeStamp	DINT	实时采样模式下的模块时间戳值。 有效值 = 0 到 32,767 毫秒 缺省值 = 0

输出参数：	数据类型：	说明：
EnableOut	BOOL	输出使能。
Out	REAL	死区时间算法计算得出的输出。该输出要设置算术状态标志。
DeltaT	REAL	更新之间的时间差值。通过控制算法计算过程输出时所用的时间 （单位为秒）。
Status	DINT	功能块的状态。
InstructFault (Status.0)	BOOL	指令检测到如下运行错误。该错误不是次要或主要的控制器错误。检查其他状态位以确定错误类型。
InFaulted (Status.1)	BOOL	输入出错。
DeadtimeInv (Status.2)	BOOL	死区时间值无效。
TimingMode (Status.27)	BOOL	TimingMode 值无效。 有关定时模式的更多信息，请参阅附录 “功能块属性”。
RTSMissed (Status.28)	BOOL	仅用于实时采样模式。ABS | DeltaT - RTSTime | > 1 （0.001 秒）时设置。
RTSTimeInv (Status.29)	BOOL	RTSTime 值无效。
RTSTimeStampInv (Status.30)	BOOL	RTSTimeStamp 值无效。
DeltaTInv (Status.31)	BOOL	DeltaT 值无效。

说明： DEDT 指令使用数据缓冲区存储延时的数据，因此可允许应用任意长度的死区时间。 DEDT 指令可在扫描频率恒定为常数的任务中执行。

要应用 DEDT 指令，需创建一个数组来存储死区时间的缓冲数据，以保持 (In x Gain) + Bias 的采样值。该存储数组的长度应足以容纳所需最大死区时间值，根据以下公式进行计算：

StorageArray 长度 = Deadtime 最大值 （秒） /DeltaT （秒）

维护死区时间缓冲区

运行期间，指令会检测死区时间是否有效。死区时间必须介于 0.0 到（StorageArray 长度 x DeltaT）之间。

如果死区时间无效，指令将设置状态字中的相应位并设置 Out = (In x Gain) + Bias。

死区时间缓冲区以先入先出原则处理数据。死区时间算法每执行一次，会将缓冲区中最早的数据传送到 Out。缓冲区中的剩余数据向下移位，同时， ((In x Gain) + Bias) 值移至死区时间缓冲区的起始位置。存入缓冲区的新数据经过 Deadtime 设定的时间值后会被传送到 Out。

实现预定延时所需的数组元素个数可由 Deadtime 除以 DeltaT 得出。如果 Deadtime 未被 DeltaT 整除，则 DeltaT 保持不变，数组元素个数和延时时间均取整到最接近的增量值。例如：计算当给定 Deadtime = 4.25 秒且 DeltaT = 0.50 秒时，实现预定延时所需的数组元素个数为：

4.25 秒 /0.50 秒 = 8.5
向上取整得出需要 9 个数组元素

此例中实际应用到输入的延时时间为：

数组元素个数 x DeltaT = 预定的延时或
9 x 0.5 秒 = 4.5 秒

如果 Deadtime 或 DeltaT 的数值在移出缓冲区时发生改变，均会导致运行时间随之改变。可以增减实现预定延时所需的元素个数。应用死区时间缓冲区前会进行以下更新：

- 如要增加所需元素个数，则将当前死区时间缓冲区中最早的数据载入新增的缓冲区元素。
- 如要减少所需元素个数，则当前死区时间缓冲区中最早的数据将被删除。

InFault 转换时的指令行为。

当 InFault 为已设置时 （出错），指令暂停执行并保持上一输出状态，同时设置状态字中的相应位。

当 InFault 从已设置转换到清零状态，指令设置 Out 且死区时间缓冲区中所有值均为 In x Gain + Bias。

算术状态标志： Out 输出影响算术状态标志。

错误条件： 无

执行：

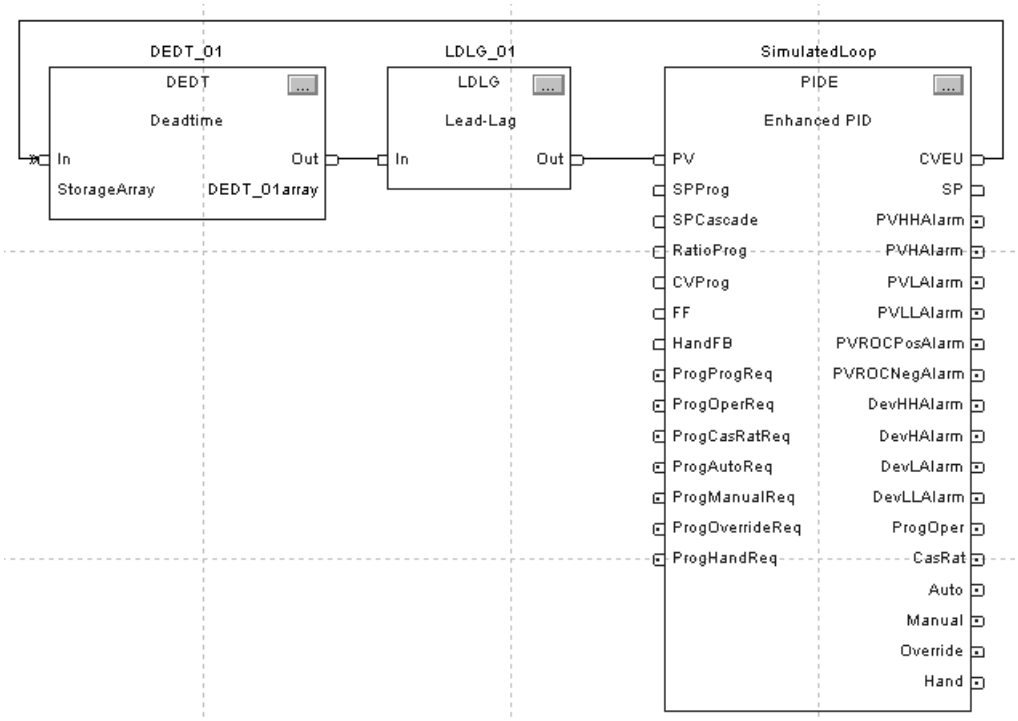
条件：	功能块动作：	结构化文本动作：
预扫描	无动作。	无动作。
指令首次扫描	如果 InFault 清零，则设置 Out 以及死区时间缓冲区的所有值都等于 (In x Gain + Bias)。	
指令首次执行	如果 InFault 清零，则设置 Out 以及死区时间缓冲区的所有值都等于 (In x Gain + Bias)。	
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。	na
EnableIn 已设置	指令执行。 EnableOut 已设置。	EnableIn 一直保持已设置状态。 指令执行。
后期扫描	无动作。	无动作。

示例： 在此例中， DEDT 指令在一个仿真过程中模拟了一次死区时间延时。PIDE 指令的输出通过一次死区时间延时和一个一阶滞后环节以对过程进行仿真。 DEDT_01array 是实型数组，包含 100 个数组元素，因此可支持最多 100 个采样点的死区时间。例如，如此例程以 100 ms 的时间间隔运行，则该数组将支持最长 10 秒的死区时间。

结构化文本

```
DEDT_01.In := SimulatedLoop.CVEU;
DEDT(DEDT_01,DEDT_01array);

LDLG_01.In := DEDT_01.Out;
LDLG(LDLG_01);

SimulatedLoop.PV := LDLG_01.Out;
PIDE(SimulatedLoop);
```

功能块



函数发生器 (FGEN)

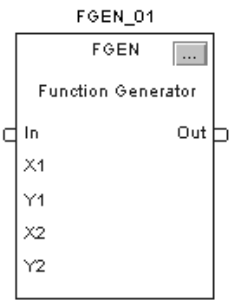
FGEN 指令基于分段线性函数对输入信号进行转换。

操作数：

```
FGEN(FGEN_tag,X1,Y1,X2,Y2);
```

结构文本

操作数：	类型：	格式：	说明：
FGEN 标记	FUNCTION_ GENERATOR	结构	FGEN 结构
X1	REAL	数组	X 轴数组，表 1。结合 Y 轴数组和表 1 来定义第一段线性曲线上的各点。有效值 = 浮点数
Y1	REAL	数组	Y 轴数组，表 1。结合 X 轴数组和表 1 来定义第一段线性曲线上的各点。有效值 = 浮点数
X2	REAL	数组	（可选） X 轴数组，表 2。结合 Y 轴数组和表 2 来定义第二段线性曲线上的各点。有效值 = 浮点数
Y2	REAL	数组	（可选） Y 轴数组，表 2。结合 X 轴数组和表 2 来定义第二段线性曲线上的各点。有效值 = 浮点数



功能块

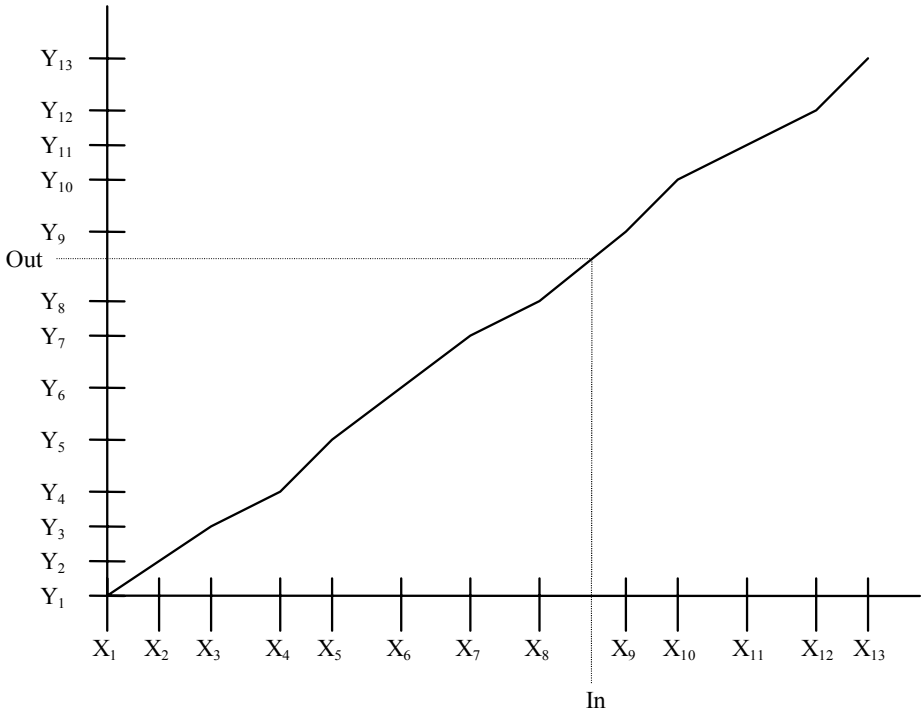
操作数与结构化文本 FGEN 指令的操作数相同。

FUNCTION_GENERATOR 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果该位清零，则指令不执行且输出不更新。 如果该位已设置，则指令执行。 缺省状态为已设置。 **结构化文本：** 无效。指令执行。
In	REAL	输入到指令的模拟量信号。 有效值 = 浮点数 缺省值 = 0.0
XY1Size	DINT	从表 1 中获取的分段线性曲线上点的个数。如果该值小于 1 且 Select 清零，指令设置状态字中的相应位并保持输出不变。 有效值 = 1 到 （X1 和 Y1 数组长度最小值） 缺省值 = 1
XY2Size	DINT	从表 2 中获取的分段线性曲线上点的个数。如果该值小于 1 且 Select 为已设置，指令设置状态字中的相应位并保持输出不变。 有效值 = 0 到 （X2 和 Y2 数组长度最小值） 缺省值 = 0
Select	BOOL	该输入用于确定所选表格。清零时，指令使用表 1。为已设置时，指令使用表 2。 缺省状态为清零。

输出参数：	数据类型：	说明：
EnableOut	BOOL	输出使能。
Out	REAL	指令输出。该输出要设置算术状态标志。
Status	DINT	功能块的状态。
InstructFault (Status.0)	BOOL	指令出错。
XY1SizeInv (Status.1)	BOOL	表 1 的行列设置无效或与数组长度不匹配。
XY2SizeInv (Status.2)	BOOL	表 2 的行列设置无效或与数组长度不匹配。
XisOutofOrder (Status.3)	BOOL	X 参数未排序。

说明： 下图所示为 FGEN 指令如何转换一条 12 段的曲线：



X 轴参数必须满足以下关系式：

X[1] < X[2] < X[3] < ... < X[XY<n>Size]，

其中， XY<n>Size > 1，表示分段线性曲线上点的编号， n 等于 1 或 2，指示所选表格。您必须在 X 数组中创建有序的 X 轴元素。

Select 输入参数用于确定指令选定使用的表。当指令在其中一个表中运行时，您可以修改另一个表中的数据。更改 Select 的状态可在两个表间切换。

计算 Out 值之前，指令先扫描 X 轴参数。如果 X 轴参数未按升序排列，则状态字中的相应位被设置且输出保持不变。同样，如果 XY1Size 或 XY2Size 的值无效，指令也会设置状态中的相应位并保持输出不变。

指令使用以下算法由输入计算输出：

- 当 In ≤ X[1] 时，令 Out = Y[1]
- 当 In > X[XY<n>Size] 时，令 Out = Y[XY<n>Size]
- 当 X[n] < In ≤ X[n+1] 时，计算 Out = ((Y[n+1]-Yn)/(X[n+1]-Xn))*(In-Xn)+Yn

算术状态标志： Out 输出影响算术状态标志。

错误条件： 无

执行：

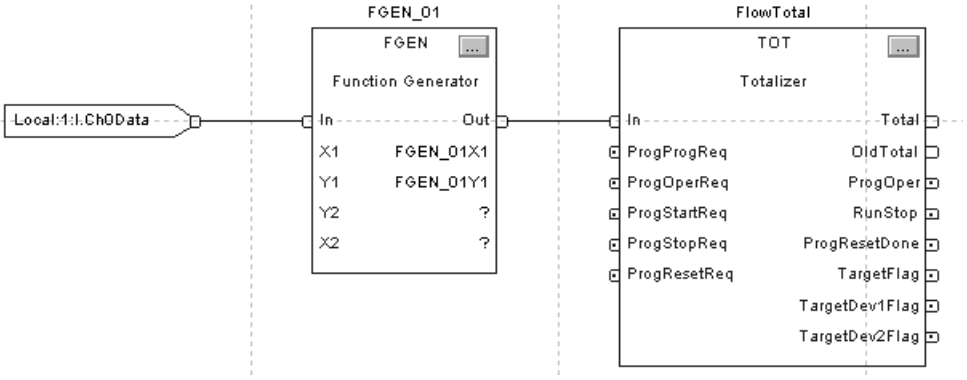
条件：	功能块动作：	结构化文本动作：
预扫描	无动作。	无动作。
指令首次扫描	无动作。	无动作。
指令首次执行	无动作。	无动作。
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。	N/A
EnableIn 已设置	指令执行。 EnableOut 已设置。	EnableIn 一直保持已设置状态。 指令执行。
后期扫描	无动作。	无动作。

示例： 在此例中，FGEN 指令对一条流量信号曲线进行转换，后由 TOT 指令对其求和。 FGEN_01X1 和 FGEN_01Y1 数组均为实型数组，各包含 10 个数组元素，分别支持最多可分 9 段的曲线。您可将任意长度的数组用于根据需要划分段数的曲线。

结构化文本

```
FGEN_01.IN := Local:1:I.Ch0Data;
FGEN(FGEN_01,FGEN_01X1,FGEN_01Y1);

FlowTotal.In := FGEN_01.Out;
TOT(FlowTotal);
```

功能块



超前滞后 (LDLG)

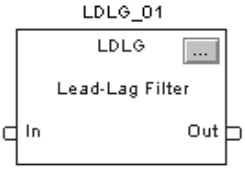
LDLG 指令对输入信号进行相位超前滞后补偿。该指令通常用于前馈 PID 控制或进行过程仿真。

操作数：

```
LDLG(LDLG_tag);
```

结构化文本

操作数：	类型：	格式：	说明：
LDLG 标记	LEAD_LAG	结构	LDLG 结构



功能块

操作数：	类型：	格式：	说明：
LDLG 标记	LEAD_LAG	结构	LDLG 结构

LEAD_LAG 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果该位清零，则指令不执行且输出不更新。 如果该位已设置，则指令执行。 缺省状态为已设置。 **结构化文本：** 无效。指令执行。
In	REAL	输入到指令的模拟量信号。 有效值 = 浮点数 缺省值 = 0.0
Initialize	BOOL	滤波控制算法初始化请求。当 Initialize 为已设置时， Out = (In x Gain) + Bias。 缺省状态为清零。
Lead	REAL	超前时间 （单位为秒）。设置 Lead = 0.0 禁用超前控制算法。如果 Lead < 0.0，则指令设置状态字中的相应位并设置 Lead 等于 0.0。如果 Lead > 最大正浮点数，指令设置状态字中的相应位。 有效值 = 任意浮点数 ≥ 0.0 缺省值 = 0.0
Lag	REAL	滞后时间 （单位为秒）。滞后时间最小值为 DeltaT/2。如果 Lag < DeltaT/2，则指令设置状态字中的相应位并设置 Lag 等于 DeltaT/2。如果 Lag > 最大正浮点数，指令设置状态字中的相应位。 有效值 = 任意浮点数 ≥ DeltaT/2 缺省值 = 0.0
Gain	REAL	过程的增益系数。该值可用于对过程增益的仿真。需将 In 值与该值相乘。 I = (In x Gain) + Bias 有效值 = 浮点数 缺省值 = 1.0

输入参数：	数据类型：	说明：
Bias	REAL	过程的偏移等级。该值可用于对外界条件的仿真。该值需与 In 和 Gain 的乘积相加。 I = (In x Gain) + Bias 有效值 = 浮点数 缺省值 = 0.0
TimingMode	DINT	选择定时执行模式。 **值：** **说明：** 0 周期模式 1 过采样模式 2 实时采样模式 有效值 = 0 到 2 缺省值 = 0 有关定时模式的更多信息，请参阅附录 “功能块属性”。
OversampleDT	REAL	过采样模式下的运行时间。 有效值 = 0 到 4194.303 秒 缺省值 = 0
RTSTime	DINT	实时采样模式下的模块更新周期 有效值 = 1 到 32,767 毫秒 缺省值 = 1
RTSTimeStamp	DINT	实时采样模式下的模块时间戳值。 有效值 = 0 到 32,767 毫秒 缺省值 = 0

输出参数：	数据类型：	说明：
EnableOut	BOOL	输出使能。
Out	REAL	算法计算的输出。该输出要设置算术状态标志。
DeltaT	REAL	更新之间的时间差值。通过控制算法计算过程输出时所用的时间 （单位为秒）。
Status	DINT	功能块的状态。
InstructFault (Status.0)	BOOL	指令检测到如下运行错误。该错误不是次要或主要的控制器错误。检查其他状态位以确定错误类型。
LeadInv (Status.1)	BOOL	Lead < 最小值或 Lead > 最大值。
LagInv (Status.2)	BOOL	Lag < 最小值或 Lag > 最大值。
TimingModeInv (Status.27)	BOOL	TimingMode 值无效。 有关定时模式的更多信息，请参阅附录 “功能块属性”。
RTSMissed (Status.28)	BOOL	仅用于实时采样模式。ABS | DeltaT - RTSTime | > 1 （0.001 秒）时设置。
RTSTimeInv (Status.29)	BOOL	RTSTime 值无效。
RTSTimeStampInv (Status.30)	BOOL	RTSTimeStamp 值无效。
DeltaTInv (Status.31)	BOOL	DeltaT 值无效。

说明： LDLG 指令支持超前环节和滞后环节的串联应用。该指令还允许用户配置增益系数和偏移量。 LDLG 指令指令可在扫描频率恒定为常数的任务中执行。

LDLG 指令使用如下等式：

$$H(s) = \left[\frac{1 + Lead \times s}{1 + Lag \times s}\right]$$

其中参数的极限值如下所示：

参数：	极限值：
Lead	LowLimit = 0.0 HighLimit = 最大正浮点数
	LowLimit = DeltaT/2 （DeltaT 的单位为秒） HighLimit = 最大正浮点数

输出的计算值为无效、 NAN 或 ± INF 时，指令设置 Out 等于该无效值并影响算术溢出状态标志。输出的计算值有效时，指令初始化内部参数并设置输出 Out = (In x Gain) + Bias。

算术状态标志： Out 输出影响算术状态标志。

错误条件： 无

执行：

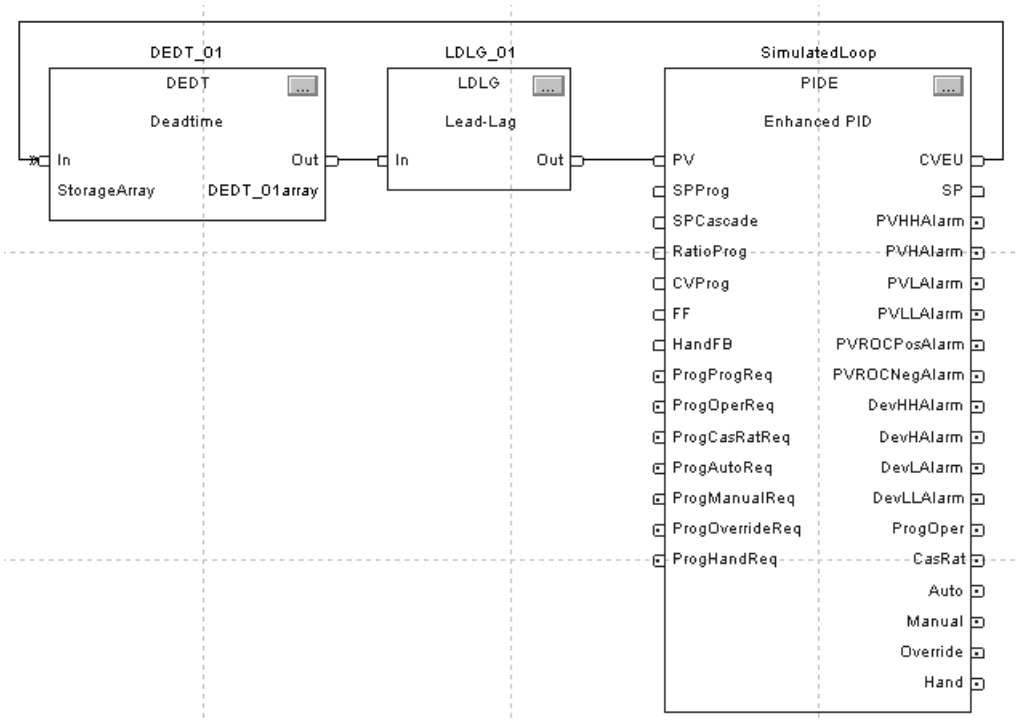
条件：	功能块动作：	结构化文本动作：
预扫描	无动作。	无动作。
指令首次扫描	无动作。	无动作。
指令首次执行	指令设置 Out = (In x Gain) + Bias。 控制算法不执行。	
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。	na
EnableIn 已设置	指令执行。 EnableOut 已设置。	EnableIn 一直保持已设置状态。 指令执行。
后期扫描	无动作。	无动作。

示例： LDLG 指令在此例中将一个一阶滞后环节添加至仿真过程中。您可以选择在 LDLG 指令中输入 Gain 来仿真过程增益，也可输入 Bias 以实现对外界条件的仿真。

结构化文本

```
DEDT_01.In := SimulatedLoop.CVEU;
DEDT(DEDT_01,DEDT_01array);

LDLG_01.In := DEDT_01.Out;
LDLG(LDLG_01);

SimulatedLoop.PV := LDLG_01.Out;
PIDE(SimulatedLoop);
```

功能块



增强型 PID (PIDE)

PIDE 指令提供标准 PID 指令的增强功能。该指令使用 PID 算法的速度公式。“增益”在此处应用于偏差值或 PV 值的变化量，而不是偏差值或 PV 值本身。

操作数：

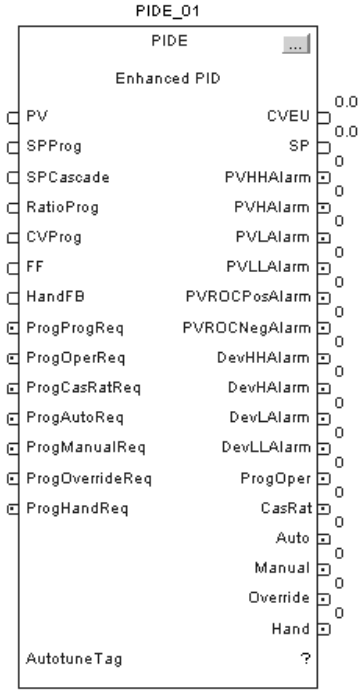
```
PIDE(PIDE_tag);
```

结构化文本

操作数：	类型：	格式：	说明：
PIDE 标记	PIDE_ENHANCED	结构	PIDE 结构

结构化文本不支持功能块中可用的 autotune 标记。

功能块



操作数：	类型：	格式：	说明：
PIDE 标记	PIDE_ENHANCED	结构	PIDE 结构
autotune 标记	PIDE_AUTOTUNE	结构	（可选） 有关 autotune 结构，请参阅第 1-57 页

PID_ENHANCED 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果该位清零，则指令不执行且输出不更新。 如果该位已设置，则指令执行。 缺省状态为已设置。 **结构化文本：** 无效。指令执行。
PV	REAL	标度的过程变量输入。该值通常从模拟量输入模块中读取。 有效值 = 浮点数 缺省值 = 0.0
PVFault	BOOL	PV 错误指示。如果 PV 值从模拟量输入中读取，则 PVFault 由该模拟量输入信号的错误状态控制。当 PVFault 为已设置时，表示输入信号出错。 缺省状态为清零 = 状态正常
PVEUMax	REAL	PV 的最大标度值。与过程变量取值范围中的 100% 相对应的 PV 和 SP 值。 有效值 = PVEUMin < PVEUMax ≤ 最大正浮点数 缺省值 = 100.0
PVEUMin	REAL	PV 的最小标度值。与过程变量取值范围中的 0% 相对应的 PV 和 SP 值。 有效值 = 最大负浮点数 ≤ PVEUMin < PVEUMax 缺省值 = 0.0
SPProg	REAL	SP 程序值，标度为 PV 单位。在程序控制和非级联 / 比例模式下， SP 被设为该值。如果 SPProg 值 < SPLLimit 或 > SPHLimit，则指令设置状态字中的相应位并将 SP 值限定在有效范围内。 有效值 = SPLLimit 到 SPHLimit 缺省值 = 0.0
SPOper	REAL	SP 操作员值，标度为 PV 单位。在操作员控制和非级联 / 比例模式下， SP 被设为该值。如果 SPOper 值 < SPLLimit 或 > SPHLimit，则指令设置状态字中的相应位并将 SP 值限定在有效范围内。 有效值 = SPLLimit 到 SPHLimit 缺省值 = 0.0
SPCascade	REAL	SP 级联值，标度为 PV 单位。如果 CascadeRatio 为已设置且 UseRatio 清零，则 SP = SPCascade。这是典型的主回路 CVEU。如果 CascadeRatio 和 UseRatio 均为已设置，则 SP = (SPCascade x Ratio)。如果 SPCascade 值 < SPLLimit 或 > SPHLimit，则设置状态字中的相应位并将 SP 值限定在有效范围内。 有效值 = SPLLimit 到 SPHLimit 缺省值 = 0.0
SPHLimit	REAL	SP 上限值，标度为 PV 单位。如果 SPHLimit > PVEUMax，则指令设置状态字中的相应位。 有效值 = SPLLimit 到 PVEUMax 缺省值 = 100.0
SPLLimit	REAL	SP 下限值，标度为 PV 单位。如果 SPLLimit < PVEUMin，则指令设置状态字中的相应位。如果 SPHLimit < SPLLimit，则指令设置状态字中的相应位，并使用 SPLLimit 限制 SP 的值。 有效值 = PVEUMin 到 SPHLimit 缺省值 = 0.0

输入参数：	数据类型：	说明：
UseRatio	BOOL	应用比例控制许可。在级联 / 比例模式下，设置该位可启用比例控制。 缺省状态为清零。
RatioProg	REAL	程序比例系数。在程序控制模式下， Ratio 和 RatioOper 被设为与该值相等。如果 RatioProg < RatioLLimit 或 > RatioHLimit，则指令设置状态字中的相应位并将 Ratio 值限定在有效范围内。 有效值 = RatioLLimit 到 RatioHLimit 缺省值 = 1.0
RatioOper	REAL	操作员比例系数。在操作员控制模式下， Ratio 被设置为与该值相等。如果 RatioOper < RatioLLimit 或 > RatioHLimit，则指令设置状态字中的相应位并将 Ratio 值限定在有效范围内。 有效值 = RatioLLimit 到 RatioHLimit 缺省值 = 1.0
RatioHLimit	REAL	比例上限值。限制从 RatioProg 或 RatioOper 获取的 Ratio 值。如果 RatioHLimit < RatioLLimit，则指令设置状态字中相应位，并使用 RatioLLimit 限制 Ratio 的值。 有效值 = RatioLLimit 到最大正浮点数 缺省值 = 1.0
RatioLLimit	REAL	比例下限值。限制从 RatioProg 或 RatioOper 获取的 Ratio 值。如果 RatioLLimit < 0，则指令设置状态字中的相应位，并限定其值为 0。如果 RatioHLimit < RatioLLimit，则指令设置状态字中相应位，并使用 RatioLLimit 限制 Ratio 的值。 有效值 = 0.0 到 RatioHLimit 缺省值 = 1.0
CVFault	BOOL	控制变量错误指示。如果 CVEU 控制模拟量输出，则正常情况下 CVFault 由该模拟量输出错误状态生成。 CVFault 为已设置时指示输出模块出错，且指令将设置状态字中的相应位。 缺省状态为清零 = 状态正常
CVInitReq	BOOL	CV 初始化请求。该信号通常受控于由 CVEU 控制的的模拟量输出模块的“摘机”状态，或来自副 PID 回路的 InitPrimary 输出。 缺省状态为清零。
CVInitValue	REAL	CVEU 初始化值，标度为 CVEU 单位。当 CVInitializing 为已设置时，CVEU = CVInitValue 且 CV 等于相应的百分数值。 CVInitValue 来自 CVEU 控制的模拟量输出的反馈或副回路的设定点。CVFaulted 或 CVEUSpanInv 为已设置时禁用指令初始化。 有效值 = 浮点数 缺省值 = 0.0
CVProg	REAL	CV 程序人工值。在程序人工模式时， CV 等于该值。如果 CVProg < 0 或 > 100，或在 CVManLimiting 已设置的情况下 CVProg < CVLLimit 或 > CVHLimit，则指令设置状态字中的相应位并将 CV 值限定在有效范围内。 有效值 = 0.0 到 100.0 缺省值 = 0.0
CVOper	REAL	CV 操作员人工值。在操作员人工模式时， CV 等于该值。未在操作员人工模式时，每次指令执行结束时 CVOper 被设置为等于 CV。如果 CVOper < 0 或 > 100，或在 CVManLimiting 已设置的情况下 CVOper < CVLLimit 或 > CVHLimit，则指令设置状态字中的相应位并将 CV 值限定在有效范围内。 有效值 = 0.0 到 100.0 缺省值 = 0.0

输入参数：	数据类型：	说明：
CVOverride	REAL	CV 越过值。在越过模式下 CV 等于该值。该值应与 PID 回路的安全状态输出相对应。如果 CVOverride < 0 或 >100，则指令设置状态字中的相应位并将 CV 值限定在有效范围内。 有效值 = 0.0 到 100.0 缺省值 = 0.0
CVPrevious	REAL	CV_{n-1} 值。如果 CVSetPrevious 已设置，则 CV_{n-1} 等于该值。 CV_{n-1} 为指令上次执行的 CV 值。在人工、越过或手动模式下或者 CVInitializing 为已设置时，忽略 CVPrevious。如果 CVPrevious < 0 或 > 100，或在自动或级联 / 比例模式下 CVPrevious < CVLLimit 或 > CVHLimit，则指令设置状态中的相应位并将 CV_{n-1} 值限定在有效范围内。 有效值 = 0.0 到 100.0 缺省值 = 0.0
CVSetPrevious	BOOL	使用 CVPrevious 的请求。如果该位已设置，则 CV_{n-1} = CVPrevious。 缺省状态为清零。
CVManLimiting	BOOL	人工模式中 CV 限制请求。如果在人工模式且 CVManLimiting 已设置，则 CV 值由 CVHLimit 和 CVLLimit 限定。 缺省状态为清零。
CVEUMax	REAL	CVEU 的最大值。对应 100% CV 的 CVEU 值。如果 CVEUMax = CVEUMin，则指令设置状态字中的相应位。 有效值 = 浮点数 缺省值 = 100.0
CVEUMin	REAL	CVEU 的最小值。对应 0% CV 的 CVEU 值。如果 CVEUMax = CVEUMin，则指令设置状态字中的相应位。 有效值 = 浮点数 缺省值 = 0.0
CVHLimit	REAL	CV 上限值。用于设置 CVHAlarm 输出。如果 CVManLimiting 已设置，该指令用于在自动、级联 / 比例模式或人工模式中限制 CV 值。如果 CVHLimit > 100 或 < CVLLimit，则指令设置状态字中的相应位。如果 CVHLimit < CVLLimit，指令使用 CVLLimit 限制 CV 值。 有效值 = CVLLimit < CVHLimit ≤ 100.0 缺省值 = 100.0
CVLLimit	REAL	CV 下限值。用于设置 CVLAlarm 输出。如果 CVManLimiting 已设置，该指令用于在自动、级联 / 比例模式或人工模式中限制 CV 值。如果 CVLLimit < 0 或 CVHLimit < CVLLimit，则指令设置状态字中的相应位。如果 CVHLimit < CVLLimit，则指令使用 CVLLimit 限制 CV 值。 有效值 = 0.0 ≤ CVLLimit < CVHLimit 缺省值 = 0.0
CVROCLimit	REAL	CV 变化率限制，单位为每秒变化的百分数。变化率限制只用于自动或级联 / 比例模式，或当 CVManLimiting 已设置时的人工模式中。输入 0 可禁用 CV 变化率限制。如果 CVROCLimit < 0，则指令设置状态字中的相应位并禁用 CV 变化率限制。 有效值 = 0.0 到最大正浮点数 缺省值 = 0.0

输入参数：	数据类型：	说明：
FF	REAL	前馈值。过零死区限制应用于 CV 后，前馈值与 CV 相加。因此 FF 值的变化总是反映在 CV 的最终输出值上。如果 FF < –100 或 > 100，则指令设置状态字中的相应位并将 FF 值限定在有效范围内。 有效值 = -100.0 到 100.0 缺省值 = 0.0
FFPrevious	REAL	FF_{n-1} 值。如果 FFSetPrevious 为已设置，则指令设置 FF_{n-1} = FFPrevious。FF_{n-1} 是指令上次执行时的 FF 值。如果 FFPrevious < –100 或 > 100，则指令设置状态字中的相应位并将 FF_{n-1} 值限定在有效范围内。 有效值 = -100.0 到 100.0 缺省值 = 0.0
FFSetPrevious	BOOL	使用 FFPrevious 的请求。如果该位已设置，则 FF_{n-1} = FFPrevious。 缺省状态为清零。
HandFB	REAL	CV 手动反馈值。在手动模式下且 HandFBFault 清零 （状态正常）时， CV 等于该值。该值通常来自现场安装的手动 / 自动工作站，用于实现从手动状态下的无扰切换。如果 HandFB < 0 或 > 100，指令设置状态字中的相应位，并将 CV 值限定在有效范围内。 有效值 = 0.0 到 100.0 缺省值 = 0.0
HandFBFault	BOOL	HandFB 值出错指示。如从模拟量输入读取 HandFB 值，则 HandFBFault 通常由模拟量输入通道的状态来控制。 HandFBFault 为已设置时指示输入模块出错，且指令将设置状态字中的相应位。 缺省状态为清零 = 状态正常
WindupHIn	BOOL	积分饱和上限请求。该位已设置时， CV 无法进行正向积分。该信号通常从副回路的 WindupHOut 输出中获得。 缺省状态为清零。
WindupLIn	BOOL	积分饱和下限请求。该位已设置时， CV 无法进行反向积分。该信号通常从副回路的 WindupLOut 输出中获得。 缺省状态为清零。
ControlAction	BOOL	控制动作请求。该位已设置时，按照 E = PV - SP 计算误差；清零时，按照 E = SP - PV 计算误差 。 缺省状态为清零。
DependIndepend	BOOL	独立 / 非独立控制请求。该位已设置时，使用独立形式的 PID 公式；清零时，使用非独立形式的 PID 公式。 缺省状态为清零。
PGain	REAL	比例增益。使用非独立形式的 PID 算法时，该值应输入无单位的比例增益。使用独立形式的 PID 算法时，该值应输入无单位的控制增益。输入 0 可禁用比例控制。如果 PGain < 0，则指令设置状态字中的相应位并使 PGain = 0。 有效值 = 0.0 到最大正浮点数 缺省值 = 0.0
IGain	REAL	积分增益。使用非独立形式的 PID 算法时，该值应输入单位为 1/minute 的积分增益。使用独立形式的 PID 算法时，该值应输入单位为每次重复所用分钟数的积分时间常数。输入 0 可禁用积分控制。如果 IGain < 0，则指令设置状态字的相应位并使 IGain = 0 。 有效值 = 0.0 到最大正浮点数 缺省值 = 0.0

输入参数：	数据类型：	说明：
DGain	REAL	微分增益。使用非独立形式的 PID 算法时，该值应输入单位为分钟的微分增益。使用独立形式的 PID 算法时，该值应输入单位为分钟的微分时间常数。输入 0 可禁用微分控制。如果 DGain < 0，则指令设置状态字中的相应位并使 DGain = 0。 有效值 = 0.0 到最大正浮点数 缺省值 = 0.0
PVEProportional	BOOL	PV 比例控制请求。该位为已设置时，使用过程变量的变化量 (PVPercent) 来计算比例系数 (DeltaPTerm)。清零时，使用误差变化量 (EPercent) 进行计算。 缺省状态为清零。
PVEDerivative	BOOL	PV 微分控制请求。该位为已设置时，使用过程变量的变化量 (PVPercent) 来计算微分系数 (DeltaDTerm)。清零时，使用误差变化量 (EPercent) 进行计算。 缺省状态为已设置。
DSmoothing	BOOL	微分平滑请求。该位为已设置时，微分系数的变化是平滑的。微分平滑产生的对 PV 信号的噪声干扰只会引起微小的输出抖动，而且能有效抑制高微分增益的影响。 缺省状态为清零。
PVTracking	BOOL	SP 跟踪 PV 请求。在人工模式下进行设置，用于使 SP 跟踪 PV。在级联 / 比例或自动模式时该请求被忽略。 缺省状态为清零。
ZCDeadband	REAL	过零死区范围，标度为 PV 单位。用于定义过零死区的范围。输入 0 可禁用过零死区检测。如果 ZCDeadband < 0，则指令设置状态字中的相应位并禁用过零死区检测。 有效值 = 0.0 到最大正浮点数 缺省值 = 0.0
ZCOff	BOOL	禁用过零请求。用于禁用死区计算的过零。 缺省状态为清零。
PVHHLimit	REAL	PV 最高报警上限值，标度为 PV 单位。 有效值 = 浮点数 缺省值 = 最大正浮点数
PVHLimit	REAL	PV 上限报警值，标度为 PV 单位。 有效值 = 浮点数 缺省值 = 最大正浮点数
PVLLimit	REAL	PV 下限报警值，标度为 PV 单位。 有效值 = 浮点数 缺省值 = 最大负浮点数
PVLLLimit	REAL	PV 最低报警下限值，标度为 PV 单位。 有效值 = 浮点数 缺省值 = 最大负浮点数
PVDeadband	REAL	PV 报警极限死区值，标度为 PV 单位。死区是每个 PV 报警极限打开和关闭之间的差值。如果 PVDeadband < 0.0，指令设置状态字中的相应位并，并限定 PVDeadband 值为 0。 有效值 = 0.0 到最大正浮点数 缺省值 = 0.0

输入参数：	数据类型：	说明：
PVROCPosLimit	REAL	PV 正向变化率报警极限。 PV 正向变化 （增长）极限值，标度为 PV 单位 / 秒。输入 0.0 可禁用正向 PVROC 报警检测。如果 PVROCPosLimit < 0.0，则指令设置状态字中的相应位并禁用 PVROC 检测。 有效值 = 0.0 到最大正浮点数 缺省值 = 0.0 PV/ 秒
PVROCNegLimit	REAL	PV 反向变化率报警极限。 PV 反向变化 （减小）极限值，标度为 PV 单位 / 秒。输入 0.0 可禁用反向 PVROC 报警检测。如果 PVROCNegLimit < 0，则指令设置状态字中的相应位并禁用反向 PVROC 检测。 有效值 = 0.0 到最大正浮点数 缺省值 = 0.0
PVROCPeriod	REAL	PV 变化率采样周期。在此时间周期 （单位为秒）内计算得出 PV 的变化率。输入 0 可禁用 PVROC 报警检测。如果 PVROCPeriod < 0.0，则指令设置状态字中的相应位并禁用正向及反向 PVROC 检测。 有效值 = 任意浮点数 ≥ 0.0 缺省值 = 0.0 秒
DevHHLimit	REAL	最高报警偏离量上限值，标度为 PV 单位。偏离量是过程变量 (PV) 和设定点 (SP) 之间的差值。偏离量报警提示操作员过程变量与设定点值之间存在差异。如果 DevHHLimit < 0.0，则该指令设置状态字中的相应位并设置 DevHHLimit = 0.0。 有效值 = 0.0 到最大正浮点数 缺省值 = 最大正浮点数
DevHLimit	REAL	上限报警偏离量极限值，标度为 PV 单位。偏离量是过程变量 (PV) 和设定点 (SP) 之间的差值。偏离量报警提示操作员过程变量与设定点值之间存在差异。如果 DevHLimit < 0.0，则该指令设置状态字中的相应位并设置 DevHLimit = 0.0。 有效值 = 0.0 到最大正浮点数 缺省值 = 最大正浮点数
DevLLimit	REAL	下限报警偏离量极限值，标度为 PV 单位。偏离量是过程变量 (PV) 和设定点 (SP) 之间的差值。偏离量报警提示操作员过程变量与设定点值之间存在差异。如果 DevLLimit < 0.0，则该指令设置状态字中的相应位并设置 DevLLimit = 0.0。 有效值 = 0.0 到最大正浮点数 缺省值 = 最大正浮点数
DevLLLimit	REAL	最低下限报警偏离量极限值，标度为 PV 单位。偏离量是过程变量 (PV) 和设定点 (SP) 之间的差值。偏离量报警提示操作员过程变量与设定点值之间存在差异。如果 DevLLLimit < 0.0，则该指令设置状态字中的相应位并设置 DevLLLimit = 0.0。 有效值 = 0.0 到最大正浮点数 缺省值 = 最大正浮点数
DevDeadband	REAL	偏离量报警极限的死区值，标度为 PV 单位。死区是每个偏离量报警极限打开和关闭之间的差值。如果 DevDeadband < 0.0，则指令设置状态字中的相应位并设置 DevDeadband = 0.0。 有效值 = 0.0 到最大正浮点数 缺省值 = 0.0
AllowCasRat	BOOL	应用级联 / 比例模式许可。改位已设置时，允许用 ProgCascadeRatioReq 或 OperCascadeRatioReq 选择级联 / 比例模式。 缺省状态为清零。

输入参数：	数据类型：	说明：
ManualAfterInit	BOOL	初始化后进入人工模式请求。该位已设置时，如果 CVInitializing 已设置则指令切换到人工模式，除非当前为越过或手动模式。 ManualAfterInit 清零时，如果无其他模式请求，指令模式不发生改变。 缺省状态为清零。
ProgProgReq	BOOL	程序请求进入程序控制模式。通过用户程序设置来请求程序控制。如果 ProgOperReq 已设置，该请求被忽略。保持该位为已设置并将 ProgOperReq 清零，可锁定指令进入程序控制状态。ProgValueReset 已设置时，每次执行指令后都会将输入清零。 缺省状态为清零。
ProgOperReq	BOOL	程序请求进入操作员控制模式。通过用户程序设置来请求操作员控制。保持该位为已设置，可锁定指令进入操作员控制状态。ProgValueReset 已设置时，每次执行指令后都会将输入清零。 缺省状态为清零。
ProgCasRatReq	BOOL	程序级联 / 比例模式请求。通过用户程序设置来请求级联 / 比例模式控制。ProgValueReset 已设置时，每次执行指令后都会将输入清零。 缺省状态为清零。
ProgAutoReq	BOOL	程序自动模式请求。通过用户程序设置来请求自动模式控制。ProgValueReset 已设置时，每次执行指令后都会将输入清零。 缺省状态为清零。
ProgManualReq	BOOL	程序人工模式请求。通过用户程序设置来请求人工模式控制。ProgValueReset 已设置时，每次执行指令后都会将输入清零。 缺省状态为清零。
ProgOverrideReq	BOOL	程序越过模式请求。通过用户程序设置来请求越过模式控制。ProgValueReset 已设置时，每次执行指令后都会将输入清零。 缺省状态为清零。
ProgHandReq	BOOL	程序请求进入手动控制模式。通过用户程序设置来请求手动模式控制。通常从手动 / 自动工作站读取该值作为数字输入。ProgValueReset 已设置时，每次执行指令后都会将输入清零。 缺省状态为清零。
OperProgReq	BOOL	操作员请求进入程序控制模式。通过操作员界面设置来请求程序控制。每次执行指令后将输入清零。 缺省状态为清零。
OperOperReq	BOOL	操作员请求进入操作员控制模式。通过操作员界面设置来请求操作员控制。每次执行指令后将输入清零。 缺省状态为清零。
OperCasRatReq	BOOL	操作员级联 / 比例模式请求。通过操作员界面设置来请求级联 / 比例模式控制。每次执行指令后将输入清零。 缺省状态为清零。
OperAutoReq	BOOL	操作员自动模式请求。通过操作员界面设置来请求自动模式控制。每次执行指令后将输入清零。 缺省状态为清零。
OperManualReq	BOOL	操作员人工模式请求。通过操作员界面设置来请求人工模式控制。每次执行指令后将输入清零。 缺省状态为清零。

输入参数：	数据类型：	说明：
ProgValueReset	BOOL	复位程序控制值。 该位为已设置时，指令每次执行要求清零所有程序请求输入。在操作员模式下为已设置时，指令设置 SPProgram = SP 以及 CVProgram = CV。 缺省状态为清零。
TimingMode	DINT	选择定时执行模式。 **值：** **说明：** 0 周期模式 1 过采样模式 2 实时采样模式 有关定时模式的更多信息，请参阅附录 “功能块属性”。 有效值 = 0 到 2 缺省值 = 0
OversampleDT	REAL	过采样模式下的运行时间。 有效值 = 0 到 4194.303 秒 缺省值 = 0
RTSTime	DINT	实时采样模式下的模块更新周期 有效值 = 1 到 32,767 毫秒 缺省值 = 1
RTSTimeStamp	DINT	实时采样模式下的模块时间戳值。 有效值 = 0 到 32,767 毫秒 缺省值 = 0

输出参数：	数据类型：	说明：
EnableOut	BOOL	输出使能。
CVEU	REAL	控制变量输出标度。根据 CVEUMax 和 CVEUMin 进行标度，其中 CVEUMax 对应 100%， CVEUMin 对应 0%。该输出通常用于控制模拟量输出模块或副回路。该输出影响算术标志。 CVEU = (CV x CVEUSpan / 100) + CVEUMin CVEU 范围计算：CVEUSpan = (CVEUMax - CVEUMin)
CV	REAL	控制变量输出。该值范围为 0 ～ 100%。在自动或级联 / 比例模式，或当 CVManLimiting 已设置时的人工模式下， CV 由 CVHLimit 和 CVLLimit 限制。其他情况时，该值被限制在 0 ～ 100%。该输出影响算术标志。
CVInitializing	BOOL	初始化模式指示。指令首次扫描和将 CVHealth 转换为清零状态 （由出错到正常）期间，如果 CVInitReq 为已设置则设置 CVInitializing。当指令初始化完成且 CVInitReq 清零后， CVInitializing 被清零。
CVHAlarm	BOOL	CV 上限报警指示。当 CV 的计算值 > 100 或 CVHLimit 时设置该位。
CVLAlarm	BOOL	CV 下限报警指示。当 CV 的计算值 < 0 或 CVLLimit 时设置该位。
CVROCAlarm	BOOL	CV 变化率报警指示。当 CV 变化率超过 CVROCLimit 时设置该位。
SP	REAL	当前设定点值。 SP 值用于在自动或级联 / 比例模式下控制 CV。
SPPercent	REAL	以 PV 范围的百分数表示的 SP 值。 SPPercent = ((SP - PVEUMin) x 100) / PVSpan PV 范围计算：PVSpan = (PVEUMax - PVEUMin)

输出参数：	数据类型：	说明：
SPHAlarm	BOOL	SP 上限报警指示。 当 SP > SPHLimit 时设置该位。
SPLAlarm	BOOL	SP 下限报警指示。 当 SP < SPLLimit 时设置该位。
PVPercent	REAL	以百分数表示的 PV 值。 PVPercent = ((PV- PVEUMin) x 100) / PVSpan PV 范围计算：PVSpan = (PVEUMax - PVEUMin)
E	REAL	过程误差。 SP 与 PV 间的差值，标度为 PV 单位。
EPercent	REAL	误差的百分数范围表示。
InitPrimary	BOOL	主回路初始化命令。在非级联 / 比例模式或 CVInitializing 已设置时设置该位。该信号通常用于主 PID 回路的 CVInitReq 输入。
WindupHOut	BOOL	积分饱和上限指示。到达 SP 上限、 CV 上限或 CV 下限时 （取决于控制动作），设置该位。该信号通常由 WindupHIn 输入控制，用于防止主回路 CV 输出积分饱和。
WindupLOut	BOOL	积分饱和下限指示。到达 SP 上限、 CV 上限或 CV 下限时 （取决于控制动作），设置该位。该信号通常由 WindupLIn 输入控制，用于防止主回路 CV 输出积分饱和。
Ratio	REAL	当前比例系数。
RatioHAlarm	BOOL	比例系数上限报警指示。当 Ratio > RatioHLimit 时设置该位。
RatioLAlarm	BOOL	比例系数下限报警指示。当 Ratio < RatioLLimit 时设置该位。
ZCDeadbandOn	BOOL	过零死区指示。该位已设置时，CV 值保持不变。如果 ZCOff 已设置，当 | E | 在 ZCDeadband 范围内时设置 ZCDeadbandOn。如果 ZCOff 清零，当 | E | 过零且仍在 ZCDeadband 范围内时，设置 ZCDeadbandOn。 | E | 超出死区范围或 ZCDeadband = 0 时 ZCDeadbandOn 清零。
PVHHAlarm	BOOL	PV 最高上限报警指示。当 PV ≥ PVHHLimit 时设置该位。 当 PV < (PVHHLimit - PVDeadband) 时清零。
PVHAlarm	BOOL	PV 上限报警指示。当 PV ≥ PVHLimit 时设置该位。 PV < (PVHLimit - PVDeadband) 时清零。
PVLAlarm	BOOL	PV 下限报警指示。当 PV ≤ PVLLimit 时设置该位。 PV > (PVLLimit + PVDeadband) 时清零。
PVLLAlarm	BOOL	PV 最低下限报警指示。当 PV ≤ PVLLLimit 时设置该位。 PV > (PVLLLimit + PVDeadband) 时清零。
PVROCPosAlarm	BOOL	PV 正向变化率报警指示。当 PV 变化率的计算值 ≥ PVROCPosLimit 时设置该位。
PVROCNegAlarm	BOOL	PV 反向变化报警指示。当 PV 变化率的计算值 ≤ (PVROCNegLimit x -1) 时设置该位。
DevHHAlarm	BOOL	偏离量最高上限报警指示。当 PV ≥ (SP + DevHHLimit) 时设置该位。 PV < (SP + DevHHLimit - DevDeadband) 时清零。
DevHAlarm	BOOL	偏离量上限报警指示。当 PV ≥ (SP + DevHLimit) 时设置该位。 PV < (SP + DevHLimit - DevDeadband) 时清零。

输出参数：	数据类型：	说明：
DevLAlarm	BOOL	偏离量下限报警指示。当 PV ≤ (SP - DevLLimit) 时设置该位。 PV > (SP - DevLLimit + DevDeadband) 时清零。
DevLLAlarm	BOOL	偏离量最低下限报警指示。当 PV ≤ (SP - DevLLLimit) 时设置该位。 PV > (SP - DevLLLimit + DevDeadband) 时清零。
ProgOper	BOOL	程序 / 操作员控制指示。处于程序控制状态时为已设置。处于操作员控制时清零。
CasRat	BOOL	级联 / 比例模式指示。处于级联 / 比例模式时为已设置。
Auto	BOOL	自动模式指示。处于自动模式时为已设置。
Manual	BOOL	人工模式指示。处于人工模式时为已设置。
Override	BOOL	越过模式指示。处于越过模式时为已设置。
Hand	BOOL	手动模式指示。处于手动模式时为已设置。
DeltaT	REAL	更新之间的时间差值。通过控制算法计算过程输出时所用的时间 （单位为秒）。
Status1	DINT	功能块的状态。
InstructFault (Status1.0)	BOOL	指令检测到如下运行错误。该错误不是次要或主要的控制器错误。检查其他状态位以确定错误类型。
PVFaulted (Status1.1)	BOOL	过程变量 (PV) 出错。
CVFaulted (Status1.2)	BOOL	控制变量 (CV) 出错。
HandFBFaulted (Status1.3)	BOOL	HandFB 值出错。
PVSpanInv (Status1.4)	BOOL	PV 范围无效。 PVEUMax ≤ PVEUMin。
SPProgInv (Status1.5)	BOOL	SPProg < SPLLimit 或 SPProg > SPHLimit。该指令将已限定的值用于 SP。
SPOperInv (Status1.6)	BOOL	SPOper < SPLLimit 或 SPOper > SPHLimit。该指令将已限定的值用于 SP。
SPCascadeInv (Status1.7)	BOOL	SPCascade < SPLLimit 或 SPCascade > SPHLimit。该指令将已限定的值用于 SP。
SPLimitsInv (Status1.8)	BOOL	极限值无效：SPLLimit < PVEUMin、 SPHLimit > PVEUMax 或 SPHLimit < SPLLimit。如果 SPHLimit < SPLLimit，指令使用 SPLLimit 限制该值。
RatioProgInv (Status1.9)	BOOL	RatioProg < RatioLLimit 或 RatioProg > RatioHLimit。该指令限制 Ratio 的值。
RatioOperInv (Status1.10)	BOOL	RatioOper < RatioLLimit 或 RatioOper > RatioHLimit。该指令限制 Ratio 的值。
RatioLimitsInv (Status1.11)	BOOL	下限 < 0 或上限 < 下限。
CVProgInv (Status1.12)	BOOL	CVManLimiting 为已设置时， CVProg < 0 或 CVProg > 100，或者 CVProg < CVLLimit 或 CVProg > CVHLimit。该指令限制 CV 值。
CVOperInv (Status1.13)	BOOL	CVManLimiting 为已设置时， CVOper < 0 或 CVOper > 100，或者 CVOper < CVLLimit 或 CVOper > CVHLimit。该指令限制 CV 值。

输出参数：	数据类型：	说明：
CVOverrideInv (Status1.14)	BOOL	CVOverride < 0 或 CVOverride > 100。该指令限制 CV 值。
CVPreviousInv (Status1.15)	BOOL	当处于自动或级联 / 比例模式时，CVPrevious < 0 或 CVPrevious > 100，或者 CVPrevious < CVLLimit 或 > CVHLimit。该指令将已限定的值用于 CV_{n-1}。
CVEUSpanInv (Status1.16)	BOOL	CVEU 范围无效。指令取 CVEUMax = CVEUMin 的值。
CVLimitsInv (Status1.17)	BOOL	CVLLimit < 0、CVHLimit > 100 或 CVHLimit < CVLLimit。如果 CVHLimit < CVLLimit，指令使用 CVLLimit 限制 CV。
CVROCLimitInv (Status1.18)	BOOL	CVROCLimit < 0。该指令禁用 ROC 限制。
FFInv (Status1.19)	BOOL	FF < –100 或 FF > 100。该指令将已限定的值用于 FF。
FFPreviousInv (Status1.20)	BOOL	FFPrevious < –100 或 FFPrevious > 100。该指令将已限定的值用于 FF_{n-1}。
HandFBInv (Status1.21)	BOOL	HandFB < 0 或 HandFB > 100。该指令将已限定的值用于 CV。
PGainInv (Status1.22)	BOOL	PGain < 0。该指令使用 PGain = 0 的值。
IGainInv (Status1.23)	BOOL	IGain < 0。该指令使用 IGain = 0 的值。
DGainInv (Status1.24)	BOOL	DGain < 0。该指令使用 DGain = 0 的值。
ZCDeadbandInv (Status1.25)	BOOL	ZCDeadband < 0。该指令禁用过零死区。
PVDeadbandInv (Status1.26)	BOOL	PVDeadband < 0。
PVROCLimitsInv (Status1.27)	BOOL	PVROCPosLimit < 0、 PVROCNegLimit < 0 或 PVROCPeriod < 0。
DevHLLimitsInv (Status1.28)	BOOL	偏离量上下限无效。最低下限 < 0、下限 < 0、上限 < 0 或最高上限 < 0。该指令使用 0 代替已设的无效极限值。
DevDeadbandInv (Status1.29)	BOOL	偏离量死区 < 0。该指令使用 DevDeadband = 0 的值。
Status2	DINT	功能块的定时状态。
TimingModeInv (Status2.27)	BOOL	TimingMode 值无效。 有关定时模式的更多信息，请参阅附录 “功能块属性”。
RTSMissed (Status2.28)	BOOL	仅用于实时采样模式。ABS | DeltaT - RTSTime | > 1 （0.001 秒）时设置。
RTSTimeInv (Status2.29)	BOOL	RTSTime 值无效。
RTSTimeStampInv (Status2.30)	BOOL	RTSTimeStamp 值无效。
DeltaTInv (Status2.31)	BOOL	DeltaT 值无效。

说明： 在级联 / 比例或自动模式时，PID 算法调节 CV 输出以保持 PV 为 SP。

ControlAction 为已设置时，EPercent 和 PVPIDPercent 的计算值需取反后才能用于控制算法。

下表描述了指令如何计算 PID 系数：

PID 系数：	计算公式：
比例	计算比例系数需使用： • PV，当 PVEProportional 为已设置时 • Error，当 PVEProportional 清零时 设置 PGain = 0 以禁用比例控制。
积分	积分系数使用 Error 计算得出。设置 IGain = 0 以禁用积分控制。另外，DependIndepend 已设置时设置 PGain = 0 将禁用积分控制。
微分	计算微分系数需使用： • PV，当 PVEDerivative 为已设置时 • Error，当 PVEDerivative 清零时 设置 DGain = 0 以禁用微分控制。另外， DependIndepend 已设置时设置 PGain = 0 将禁用积分控制。 DSmoothing 已设置时，启用微分平滑；清零时，禁用微分平滑。微分平滑产生的对 PV 信号的噪声干扰只引起微小的 CV 输出抖动且能有效抑制高微分增益的影响。

计算 CV 值

PID 控制算法将 Delta PTerm、 Delta ITerm、 Delta DTerm 及前一次指令执行的 CV 值（即 CV_{n-1}）相加来计算出 CV 值。当 CVSetPrevious 已设置时， CV_{n-1} 等于 CVPrevious。则您可在计算 CV 值前使用指定值预设 CV_{n-1}。

$$CalculatedCV = CV_{n-1} + \Delta PTerm + \Delta ITerm + \Delta DTerm$$

PIDE 算法

PIDE 指令采用与大多数 DCS 系统所用类似的速度公式 PID 算法。速度公式算法的优点包括：

- 无扰自适应增益更改蜒无需对算法进行初始化即可动态更改增益。
- 多回路控制方案蜒通过控制 CV_{n-1} 系数可以实现回路间的交叉限制。

独立增益公式

$$CV_n = CV_{n-1} + K_P\Delta E + \frac{K_I}{60}E\Delta t + 60K_D\frac{E_n - 2E_{n-1} + E_{n-2}}{\Delta t}$$

在算法的此公式中，算法的每个系数 （比例、积分和微分）均有各自的增益。更改一个增益只改变与该增益对应的系数，而不影响其他系数，其中：

PIDE 项：	说明：
CV	控制变量
E	偏差量百分比
t	该回路使用的更新时间 （单位为秒）
Δt	比例增益
K_I	积分增益 （单位为 min^{-1}） 较大的 K_I 值可得到更快的积分响应。
K_D	微分增益 （单位为分钟）

非独立增益公式

$$CV_n = CV_{n-1} + K_C\left(\Delta E + \frac{1}{60T_I}E\Delta t + 60T_D\frac{E_n - 2E_{n-1} + E_{n-2}}{\Delta t}\right)$$

在算法的此公式中，比例增益变为控制器增益。通过更改控制器增益，您可以同时改变三个系数 （比例、积分和微分）的动作，其中：

PIDE 项：	说明：
CV	控制变量
E	范围百分比误差
Δt	该环节使用的更新时间 （单位为秒）
K_C	控制器增益
T_I	积分时间常数 （单位为每次重复所用分钟数） 较大的 T_I 值对应较慢的积分响应 积分系数重复比例系数的动作，用于响应偏差修改步骤需要用 T_I 分钟。
T_D	微分时间常数 （单位为分钟）

确定要使用的算法

PIDE 的 DependIndepend 参数清零时， PGain、 IGain 和 DGain 参数分别表示 K_P、K_I 和 K_D。DependIndepend 为已设置时，PGain、IGain 和 DGain 参数分别表示 K_C、 T_I 和 T_D。

以上 PIDE 公式是 PIDE 指令所用的典型算法。通过控制 PVEProportional 和 PVEDerivative 参数，您可将 PV （单位为百分比范围）变化量替代误差值变化量用于比例和微分系数。默认情况下，PIDE 指令将误差值变化量用于比例系数， PV 变化量用于微分系数。这样即消除了设定点变化时的大量微分峰值。

您可以使用以下公式转换不同 PIDE 算法使用的增益：

- $K_P = K_C$
- $K_I = \frac{K_C}{T_I}$
- $K_D = K_C T_D$

每种算法对相应增益采用相同的控制。有些人习惯使用独立的增益样式，这样可以控制单个增益而不影响其他系数。另一些人则更倾向于使用非独立的增益样式，至少在某种程度上这样可通过改变控制器增益对 PID 回路进行统一修改，而无需单独更改每个增益。

监控 PIDE 指令

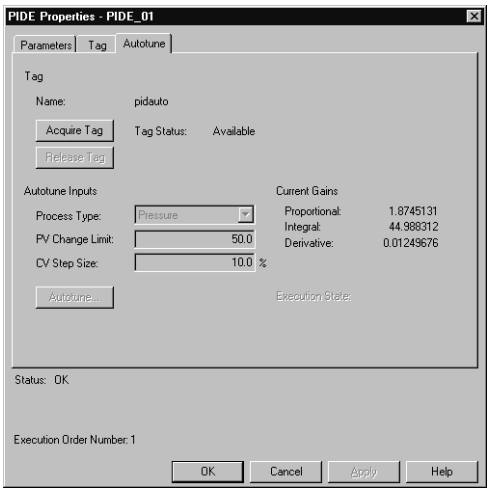
PIDE 指令具有操作员面板。了解有关更多详细信息，请参阅附录“功能块面板控件”。

自动调节 PIDE 指令

RSLogix 5000 PIDE 自动调节器在 PIDE 指令中嵌入开环自动调节器。您可以通过 PanelView 终端或任一其他操作员界面设备 （如 RSLogix 5000 软件）来实现自动调节。如果您希望对 PIDE 功能块进行自动调节，需指定该 PIDE 功能块的 Autotune 标记 （类型为 PIDE_AUTOTUNE）。

PIDE 自动调节器与 RSLogix 5000 软件同时安装，但需要使用激活钥匙将其启用。另外，仅功能块编程支持自动调节器功能；自动调节器在梯形图和结构化文本编程中均不可用。

使用 Autotune 选项卡指定和配置 PIDE 功能块的 autotune 标记。



了解有关自动调节器的更多信息，请参阅 RSLogix 5000 在线帮助或*《PIDE 自动调节器快速入门》*，出版号：PIDE-GR001。

算术状态标志： CV 输出影响算术状态标志。

错误条件： 无

执行：

条件：	功能块动作：	结构化文本动作：
预扫描	InstructionFirstScan 已设置	InstructionFirstScan 已设置
指令首次扫描	如果 CVFault 和 CVEUSpanInv 均已设置，请参阅过程错误 11-77。 如果 CVFault 和 CVEUSpanInv 清零 1. 设置 CVInitializing。 2. 如果 PVFault 已设置，则 PVSpanInv 和 SPLimitsInv 清零。请参阅第 1-77 页的过程错误。 3. 不执行 PID 控制算法。 4. 指令设置 CVEU = CVInitValue 和 CV = 相应的百分数。 CVInitValue 不受 CVEUMax 或 CVEUMin 限制。当指令计算 CV 相应的百分数时，其限制范围为 0-100。 $CVEU = CVInitValue$ $CV_{n-1} = CV = \frac{CVEU - CVEUMin}{CVEUMax - CVEUMin} ¥ 100$ $CVOper = CV$ 5. 当 CVInitializing 和 ManualAfterInit 已设置时，指令禁用自动或级联 / 比例模式。如果当前未处于越过或手动模式，指令转至人工模式。如果 ManualAfterInit 清零，则模式不发生改变。 6. 所有操作员请求输入均清零。 7. 如果 ProgValueReset 已设置，则所有程序输入请求均清零。 8. 所有 PV 上下限、 PV 变化率以及偏离量上下限报警输出均清零。 9. 如果 CVInitReq 清零，则 CVInitializing 清零。	
指令首次执行	ProgOper 清零。 指令更改为人工模式。	ProgOper 清零。 指令更改为人工模式。
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。	不影响
EnableIn 已设置	指令执行。 EnableOut 已设置。	EnableIn 一直保持已设置状态。 指令执行。
后期扫描	无动作。	无动作。

当 CVInitReq 已设置或在指令首次扫描，或 CVFault 转换为清零状态（由出错到正常）期间，指令将 CVEU 和 CV 输出初始化为 CVInitValue 的值。如果定时模式不是过采样模式且 EnableIn 由清零跳变为已设置，则指令初始化 CVEU 和 CV 值。初始化结束且 CVInitReq 清零后，CVInitialization 清零。

CVInitValue 通常从模拟量输出的返回值中获得。CVInitReq 值通常来自 CVEU 控制的模拟量输出上的 “保持”状态位。运行初始化程序的目的是为了避免将启动时产生的干扰信号输出至现场设备。

如果 CVFault 或 CVEUSpanInv 为已设置，则指令不会进行初始化，且 CVEU 和 CV 值不更新。

使用级联的 PID 回路时，当副回路已初始化或离开级联 / 比例模式时，主 PID 回路可进行初始化。这种情况下，将 InitPrimary 输出和 SP 输出从副回路移至主回路上的 CVInitReq 输入和 CVInitValue 输入。

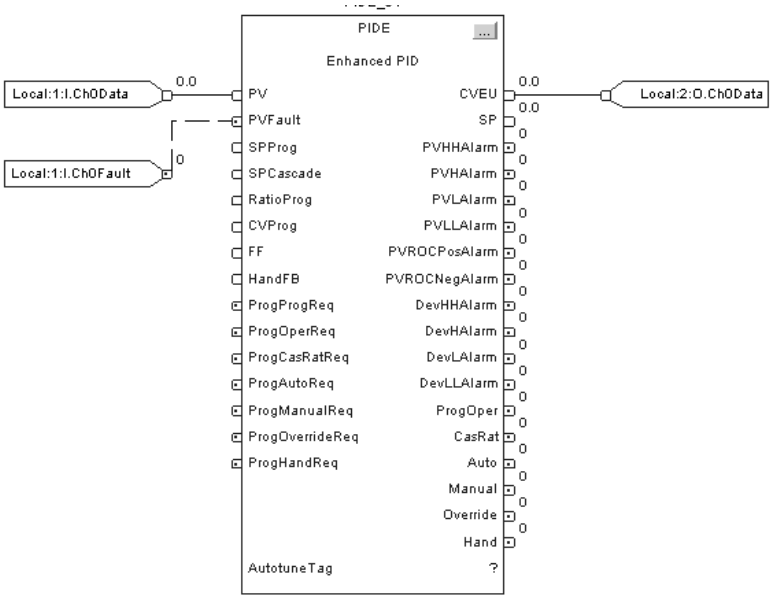
例 1： 应用 PIDE 指令最简单的方法是在周期性任务程序中创建一个功能块例程。PIDE 指令的默认定时模式是周期式。当 PIDE 指令在周期性任务中为周期式定时模式时，该指令自动将周期任务的更新频率用作其 DeltaT 更新时间。您只需将过程变量模拟量输入连接到 PIDE 指令的 PV 参数输入，并将 PIDE 指令的 CVEU 输出连接到控制变量模拟量输出。

您还可选择将模拟量输入错误指示 （如可用）连至 PIDE 的 PVFault 参数。这可实现当模拟量输入出错时强制 PIDE 指令进入人工模式，以防止在 PV 信号不可用时 PIDE CVEU 输出产生超调。

结构化文本

```
PIDE_01.PV := Local:1:I.Ch0Data;
PIDE_01.PVFault := Local:1:I.Ch0Fault;
PIDE(PIDE_01);
Local:2:O.Ch0Data := PIDE_01.CVEU;
```

功能块



例 2： 在外部因素经常对控制变量产生干扰，从而影响您控制所需的过程变量的情况下，级联控制非常有用。例如，通过改变罐外加热套中的蒸汽量来控制罐中液体的温度。当蒸汽流量由于逆流现象而突然减少时，罐中液体的温度可能随之下降，此时 PIDE 指令就会打开蒸汽阀对降温进行补偿。

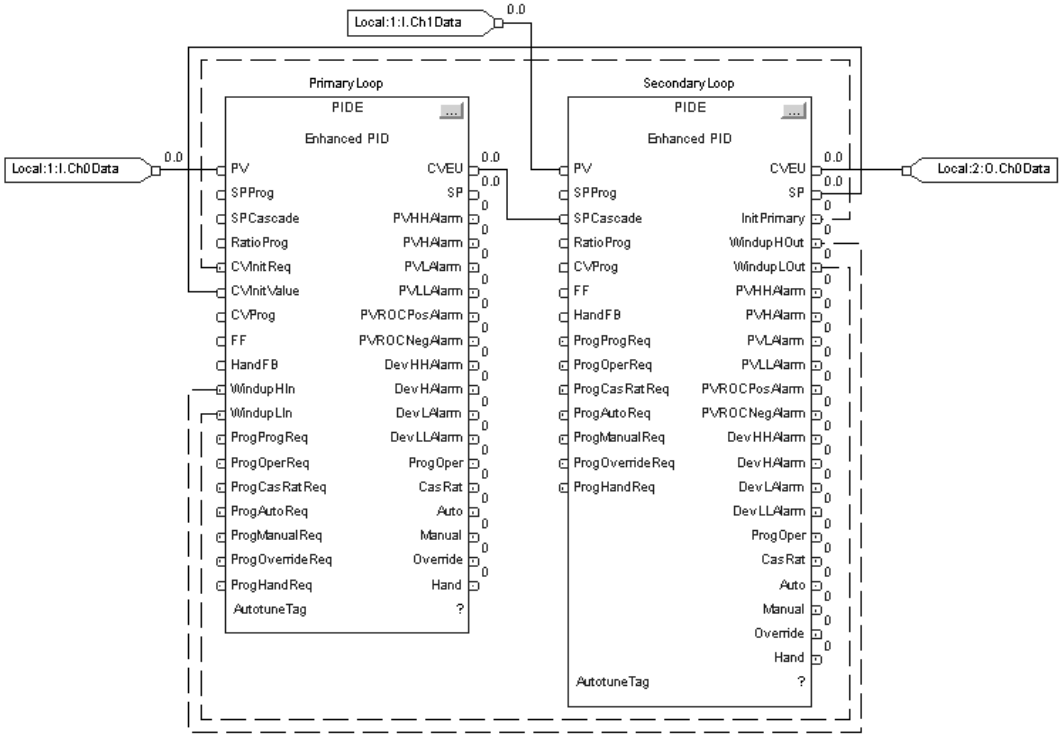
在此例中，级联回路在蒸汽流量减少但罐中液体温度下降*之前*，即可开启蒸汽阀，从而提供更优化的控制。根据来自蒸气流量传送仪的过程变量信号，使用 PIDE 指令控制蒸气阀的开启，即可实现级联回路。此回路为级联对的副回路。另一个 PIDE 指令 （称为主回路）采用液体温度作为过程变量，并将其 CV 输出发送至副回路的设定点。通过这种模式，主温度回路向副蒸气流量回路请求一定量的蒸汽流量。蒸汽流量回路对该请求作出响应，提供温度回路所要求的蒸汽流量以维持液体温度恒定。

结构化文本

```
PrimaryLoop.PV := Local:1:I.CH0Data;
PrimaryLoop.CVInitReq := SecondaryLoop.InitPrimary;
PrimaryLoop.CVInitValue := SecondaryLoop.SP;
PrimaryLoop.WindupHIn := SecondaryLoop.WindupHOut;
PrimaryLoop.WindupLIn := SecondaryLoop.WindupLOut;
```

```
PIDE(PrimaryLoop);

SecondaryLoop.PV := Local:1:I.Ch1Data;
SecondaryLoop.SPCascade := PrimaryLoop.CVEU;

PIDE(SecondaryLoop);

Local:2:O.Ch0Data := SecondaryLoop.CVEU;
```

功能块



对于正常工作的级联对回路，副回路必须具有比主回路更快的过程响应速度。这是因为副回路的过程必须足以在任何干扰影响到主回路之前对其进行补偿。在此例中，如果蒸汽流量减少，副回路控制必须在液体温度下降之前动作，以使蒸汽流量增加。

要建立一对级联的 PIDE 指令，应先在副回路中设置 AllowCasRat 输入参数。即允许副回路进入级联 / 比例模式。然后将主回路的 CVEU 连接到副回路的 SPCascade 参数。当副回路处于级联 / 比例模式时， SPCascade 值被用作其 SP。主回路 CVEU 的工程单位范围必须与副回路的 PV 工程单位范围一致。这使主回路可将其 0-100% 的 CV 标度为与副回路中设定点相匹配的工程单位。

PIDE 指令还支持其他功能以实现更有效的级联控制。将副回路的 InitPrimary 输出连接到主回路的 CVInitReq 输入，并连接副回路的 SP 输出和主回路的 CVInitValue 输入。当副回路处于级联 / 比例模式时，可使主回路的 CVEU 值等于副回路的 SP。还可实现副回路以无扰转换返回级联 / 比例模式。同样，将副回路的 WindupHOut 和 WindupLOut 输出连接到主回路的 WindupHIn 和 WindupLIn 输入。将使主回路的 CVEU 值在副回路达到 SP 或 CV 极限时停止增加或减小，从而避免在这种情况下发生主回路积分饱和。

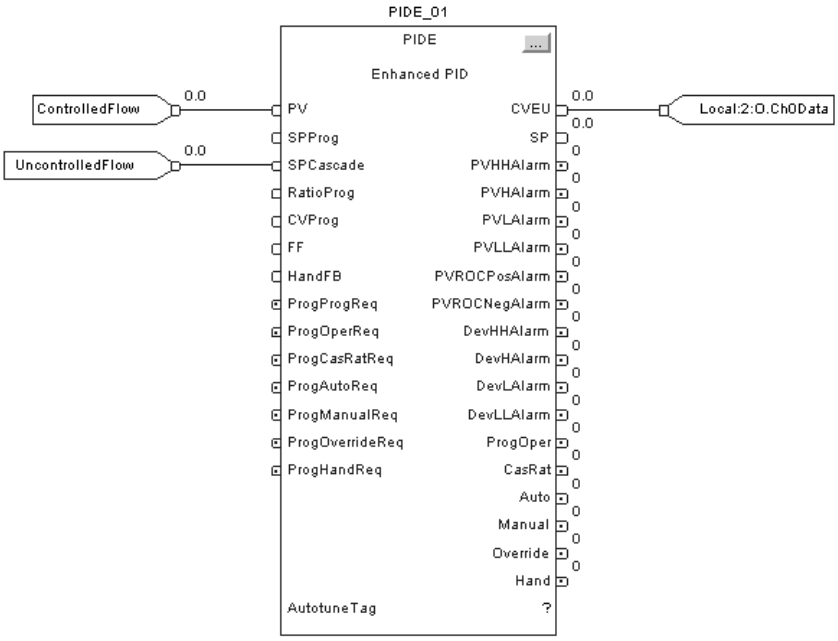
例 3： 比例控制通常用于将一种流体按固定比例添加到另一种流体中。例如：如果您想要将两种反应物 （称为 A 和 B）按一定的比例添加至容器中，由于受某种逆流现象的影响，反应物 A 会随时间发生变化，您可以利用比例控制器自动调节反应物 B 的添加比例。在此例中，反应物 A 称为”非受控 “流，因为它不受 PIDE 指令控制。反应物 B 称为”受控 “流。

若要使用 PIDE 指令进行比例控制，应设置 AllowCasRat 和 UseRatio 输入参数。将非受控流连接到 SPCascade 输入参数。处于级联 / 比例模式时，非受控流乘以 RatioOper （操作员控制状态）或 RatioProg （程序控制状态），所得结果将作为 PIDE 指令的设定点。

结构化文本

```
PIDE_01.PV := ControlledFlow;
PIDE_01.SPCascade := UncontrolledFlow;

PIDE(PIDE_01);

Local:2:O.Ch0Data := PIDE_01.CVEU;
```

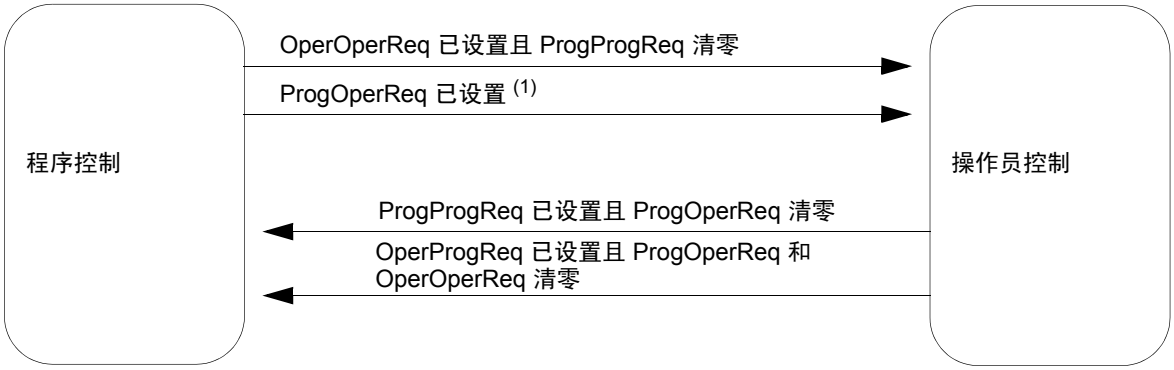
功能块



在程序控制和操作员控制之间进行切换

PIDE 指令通过用户程序或操作员界面进行控制。您可随时改变其控制模式。程序控制和操作员控制使用同一个 ProgOper 输出。当 ProgOper 已设置时，为程序控制状态； ProgOper 清零时，为操作员控制状态。

下图所示为 PIDE 指令如何在程序控制与操作员控制之间进行切换。



(1) 当 ProgOperReq 为已设置时，指令仍处在操作员控制模式。

了解有关程序控制和操作员控制的更多详细信息，请参阅第 B-13 页。

操作模式

PIDE 指令支持以下 PID 模式：

PID 操作模式：	说明：
级联 / 比例	处于级联 / 比例模式时，指令计算 CV 的变化量。指令调节 CV 以保持 PV 为 SPCascade 值或 SPCascade 乘以 Ratio 后的值。SPCascade 来自用于级联控制的主 PID 回路的 CVEU 或比例受控回路的非受控流。 使用 OperCasRatReq 或 ProgCasRatReq 选择级联 / 比例模式： 设置 OperCasRatReq 请求级联 / 比例模式。当 ProgOper、ProgOverrideReq、ProgHandReq、OperAutoReq 或 OperManualReq 为已设置，或当 AllowCasRat 清零时，该请求被忽略。 设置 ProgCasRatReq 请求级联 / 比例模式。当 ProgOper 或 AllowCasRat 清零，或当 ProgOverrideReq、 ProgHandReq、ProgAutoReq 或 ProgManualReq 为已设置时，该请求被忽略。
Auto	处于自动模式时，指令计算 CV 的变化量。指令调节 CV 以保持 PV 为 SP 值。如果处于程序控制模式， SP = SPProg ；处于操作员状态时，SP = SPOper。 使用 OperAutoReq 或 ProgAutoReq 选择自动模式： 设置 OperAutoReq 请求自动模式。当 ProgOper、ProgOverrideReq、 ProgHandReq 或 OperManualReq 为已设置时，该请求被忽略。 设置 ProgAutoReq 请求自动模式。当 ProgOper 清零，或当 ProgOverrideReq、 ProgHandReq 或 ProgManualReq 为已设置时，该请求被忽略。
Manual	处于人工模式时，指令不计算 CV 的变化量。CV 值取决于控制模式。如果处于程序控制模式， CV = CVProg ；处于操作员控制模式时，CV = CVOper。 使用 OperManualReq 或 ProgManualReq 选择人工模式： 设置 OperManualReq 请求人工模式。当 ProgOper、ProgOverrideReq 或 ProgHandReq 为已设置时，该请求被忽略。 设置 ProgManualReq 请求人工模式。 ProgOper 清零或当 ProgOverrideReq 或 ProgHandReq 为已设置时，该请求被忽略。
Override	处于越过模式时，指令不计算 CV 的变化量。 CV = CVOverride，与控制模式无关。越过模式通常是用于设定 PID 回路的 “安全状态”。 使用 ProgOverrideReq 选择越过模式： 设置 ProgOverrideReq 请求越过模式。 ProgHandReq 清零时，该请求被忽略。
Hand	处于手动模式时，PID 算法不计算 CV 的变化量。CV = HandFB，与控制模式无关。手动模式通常用于指示最后控制单元的控制被具有手动 / 自动状态的现场站取代。 使用 ProgHandReq 选择手动模式： 设置 ProgHandReq 请求手动模式。通常从手动 / 自动工作站读取该值作为数字输入。

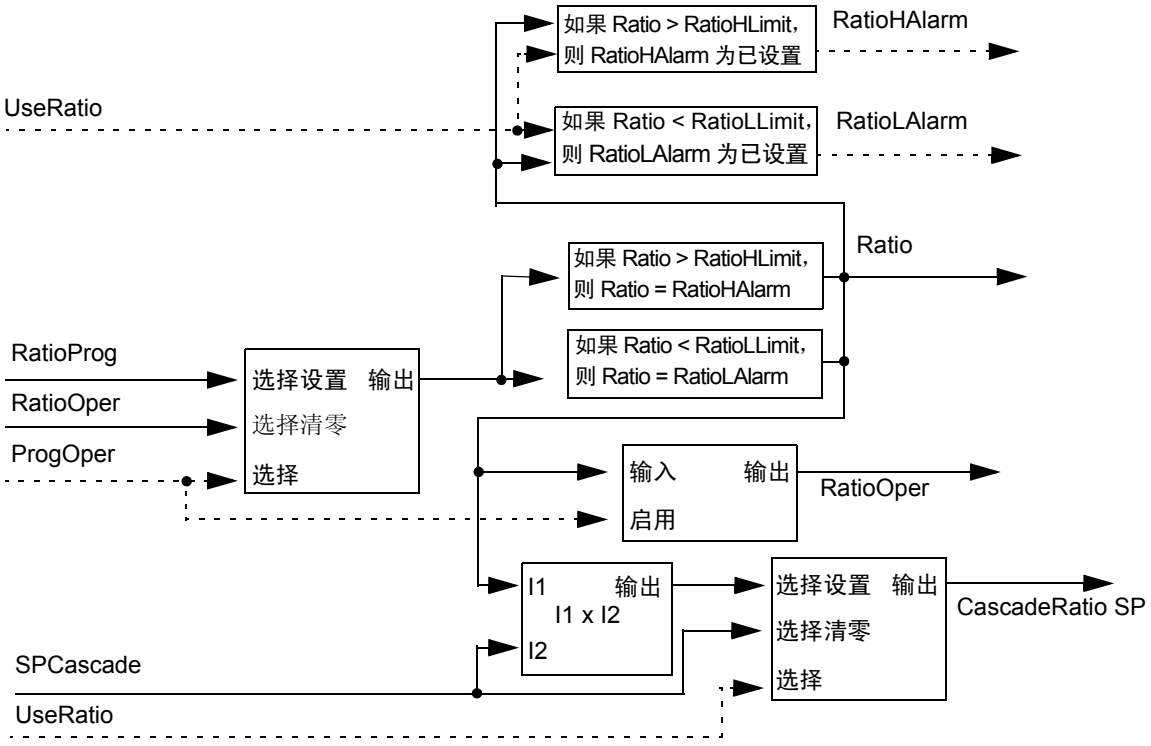
在程序控制模式下，级联 / 比例、自动、人工模式可通过用户程序进行控制；在操作员控制模式下通过操作员界面进行控制。越过或手动模式只能通过用户程序发出模式请求来控制；这些输入操作可由程序和操作员控制。

选择设定点

一旦指令确定了程序或操作员控制及 PID 模式，指令便可得到正确的 SP 值。您可以选择级联 / 比例 SP 或当前 SP。

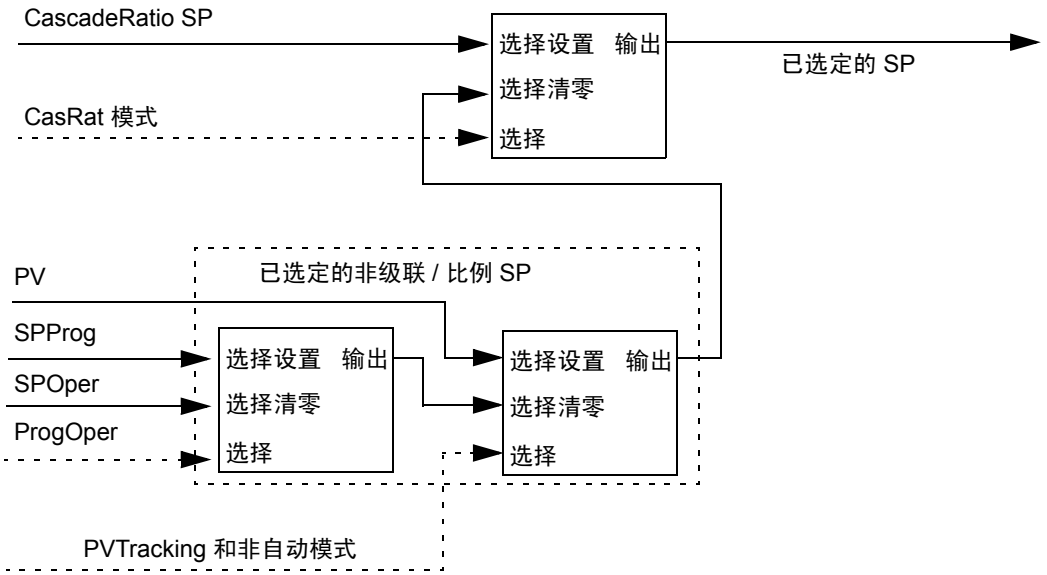
级联 / 比例 SP

级联 / 比例 SP 取决于 UseRatio 和 ProgOper 的值。



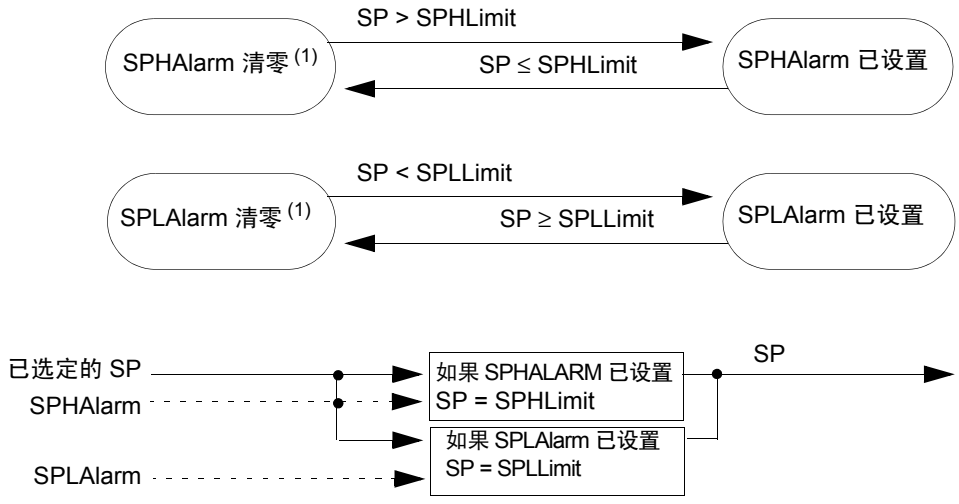
当前 SP

当前 SP 取决于级联 / 比例模式、 PVTracking 值、自动模式以及 ProgOper 值。



SP 上 / 下限

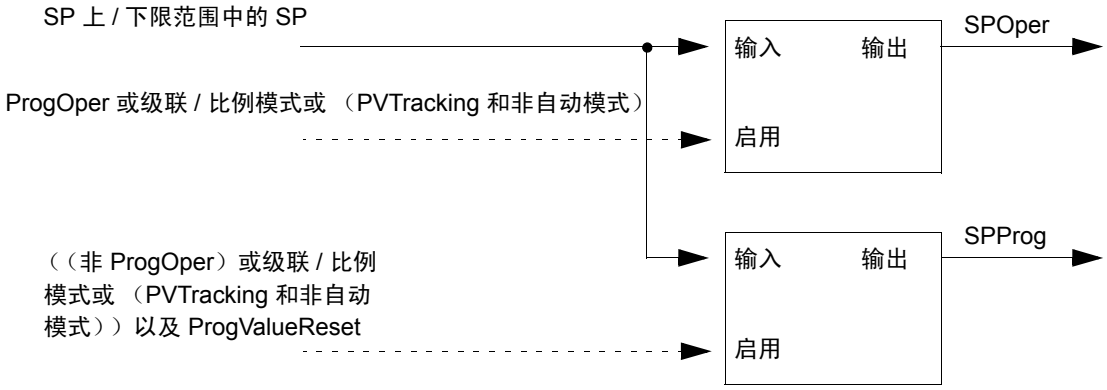
上下限报警算法将 SP 与 SPHLimit 和 SPLLimit 报警极限进行比较。SPHLimit 不能大于 PVEUMax 且 SPLLimit 不能小于 PVEUMin。



(1) 在指令首次扫描期间，指令将 SP 报警输出清零。同时，指令在 PVSpanInv 已设置时将 SP 报警极限清零并禁用报警算法。

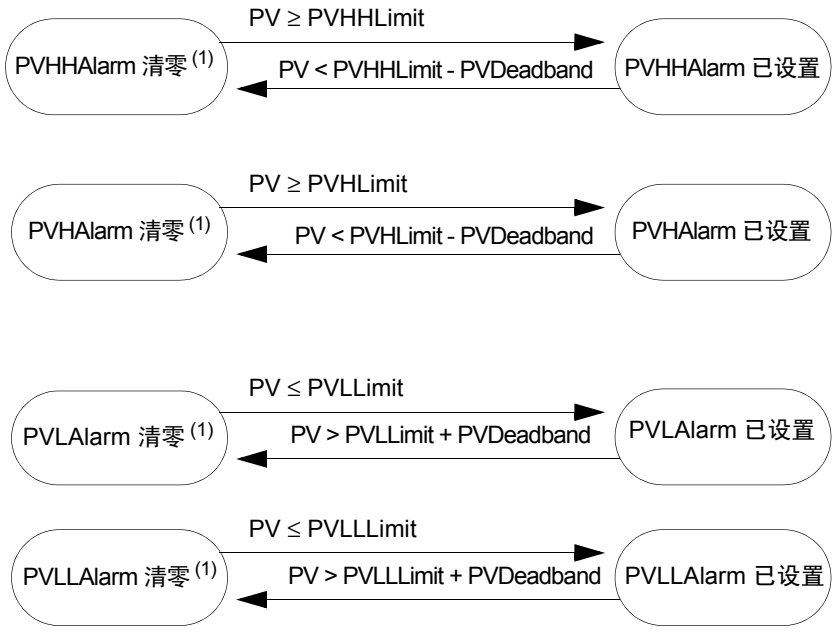
更新 SPOper 和 SPProg 值

PIDE 指令设定 SPOper = SP 或 SPProg = SP，以实现在程序控制和操作员控制之间，或在级联 / 比例模式中的无扰切换控制。



PV 上 / 下限报警

最高上限到最低下限报警算法将 PV 与 PV 报警极限及其加减 PV 报警死区的结果进行比较。



(1) 在指令首次扫描期间，指令将所有 PV 报警输出清零。同时，指令在 PVFaulted 已设置时将 PV 报警输出清零并禁用报警算法。

PV 变化率报警

PV 变化率 (ROC) 报警将 PV 在 PVROCPeriod 周期内的变化量与 PV 正向和反向变化率极限进行比较。PVROCPeriod 规定了变化率报警的死区类型。例如，如果您使用执行周期为 100ms 的 2 瘾 / 秒的 ROC 报警极限，而模拟量输入模块的分辨率为 1 瘾，则每当输入值变化时都会生成 ROC 报警，因为指令计算出的变化率为 10 瘾 / 秒。然而，如果使用不小于 1 秒的 PVROCPeriod，则仅当变化率确实超过 2 瘾 / 秒的极限值时才生成 ROC 报警。

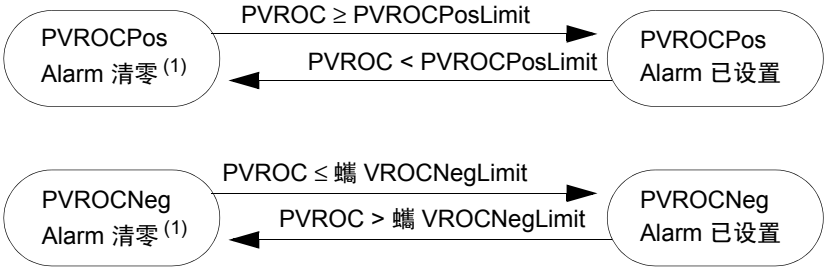
仅当 PVROCPeriod 结束时才执行 ROC 计算。变化率计算公式如下：

$$ElapsedROCPeriod = ElapsedROCPeriod + ElapsedTimeSinceLastExecution$$

如果 ElapsedROCPeriod ≥ PVROCPeriod 则：

值：	公式：
PVROC	$\frac{PV_n - PVROC_{n-1}}{PVROCPeriod}$
$PVROC_{n-1}$	$PVROC_{n-1} = PV_n$
ElapsedROCPeriod	$ElapsedROCperiod = 0$

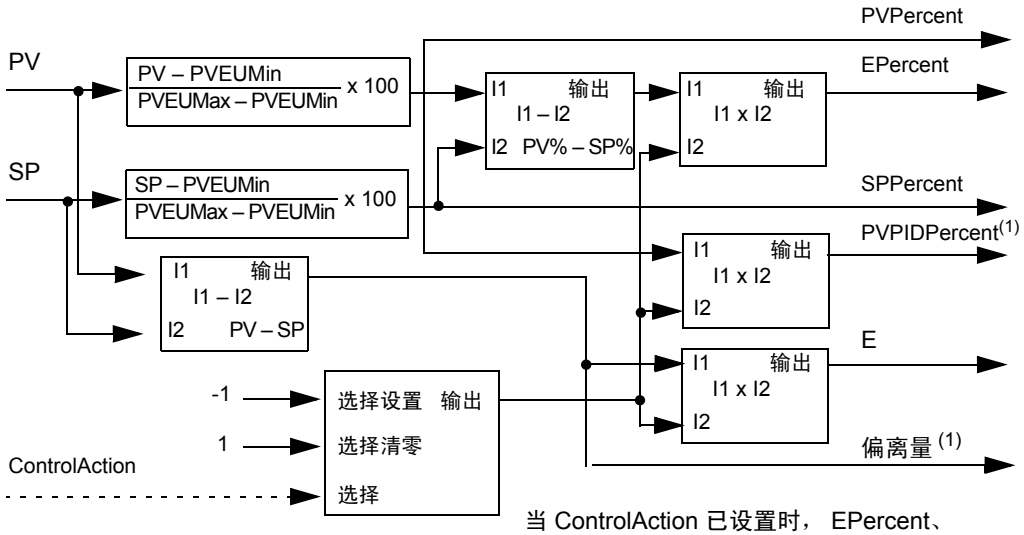
一旦指令计算出 PVROC 值， PV ROC 报警由以下关系决定：



(1) 在指令首次扫描期间，指令将 PV ROC 报警输出清零。同时，指令在 PVFaulted 已设置时将 PVROC 报警输出清零并禁用 PV ROC 报警算法。

转换 PV 和 SP 值为百分数

指令将 PV 和 SP 值转换为百分数并在执行 PID 控制算法前计算误差。该误差是 PV 和 SP 值之间的差值。 ControlAction 为已设置时，EPercent、 E 和 PVPIDPercent 的值需取反后才能用于 PID 算法。

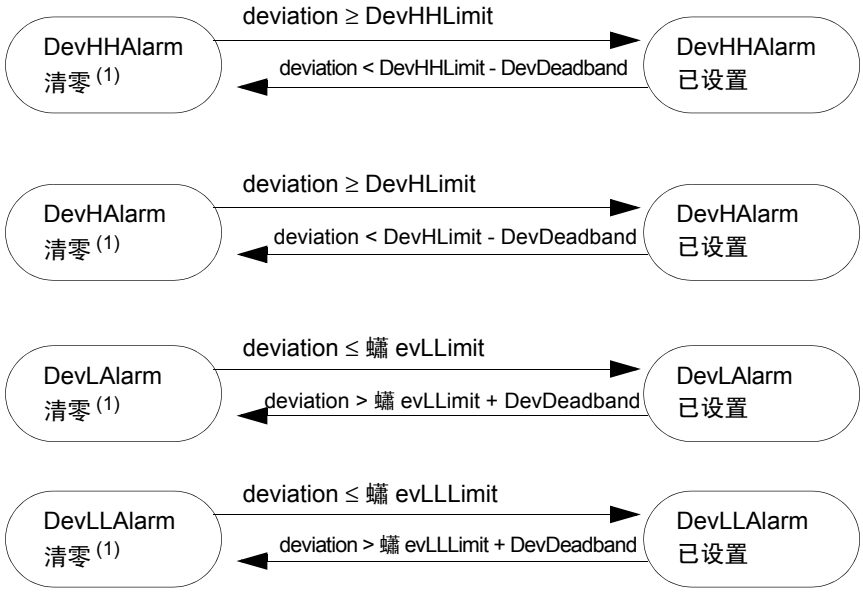


(1) PVPIDPercent 和 Deviation 是用于 PID 控制算法的内部参数。

偏离量上 / 下限报警

偏离量是过程变量 (PV) 与设定点 (SP) 之间的差值。偏离量报警提示操作员过程变量与设定点值之间存在差异。

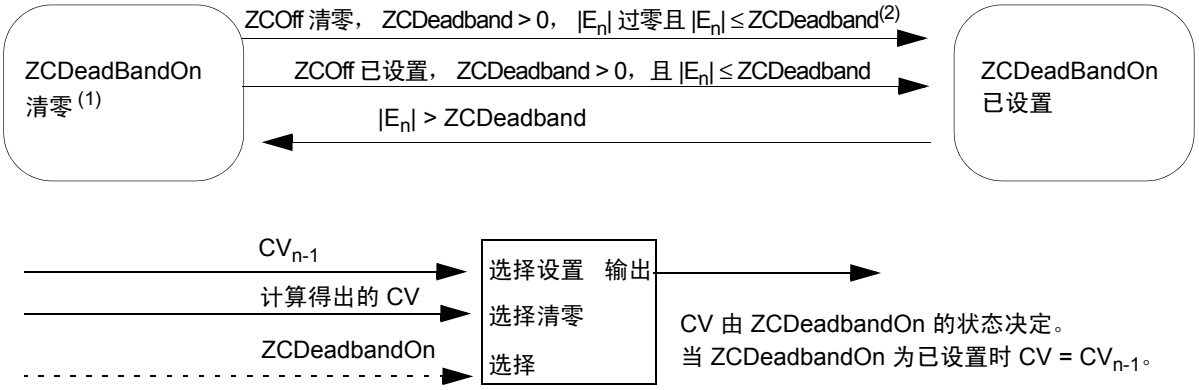
最高上限到最低下限报警算法将偏离量与偏离量报警极限及其加减死区后的结果进行比较。



(1) 在指令首次扫描期间，指令将偏离量报警输出清零。同时，指令在 PVFaulted 或 PVSpanInv 已设置时将偏离量报警输出清零并禁用报警算法。

过零死区控制

当误差处于由 ZCDeadband (| E | ≤ ZCDeadband) 指定的范围内时，您可以设定 CV 值保持不变。



(1) 当 ZCOff 清零时， ZCDeadband > 0，误差第一次过零，（即 $E_n \geq 0$ 且 $E_{n-1} < 0$ 或当 $E_n \leq 0$ 且 $E_{n-1} > 0$ 时），且 $| E_n | \leq$ ZCDeadband，指令设置 ZCDeadbandOn。

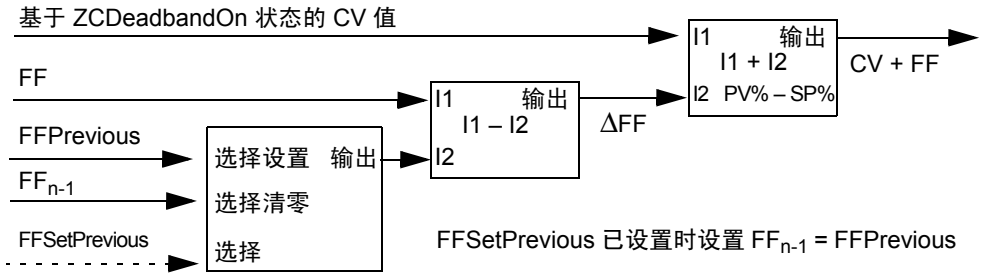
(2) 当指令切换到自动或级联 / 比例模式时，指令设置 $E_{n-1} = E_n$。

在以下条件时，指令禁用过零算法并清零 ZCDeadbandOn：

- 指令首次扫描期间
- ZCDeadband ≤ 0
- 当前模式不是自动或级联 / 比例模式
- PVFaulted 已设置
- PVSpanInv 已设置

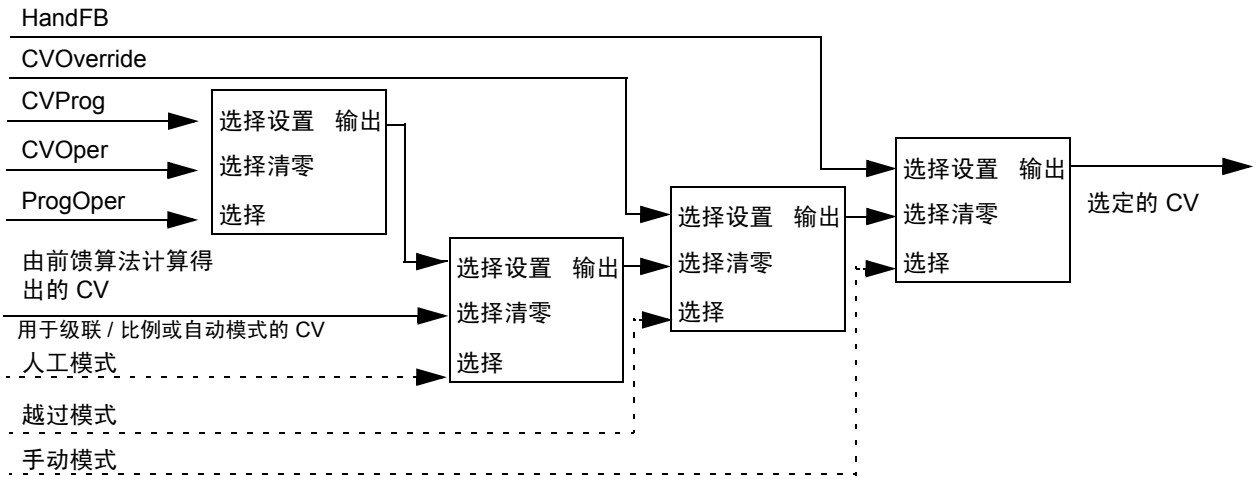
前馈控制

对由过零算法得到的 CV 与 ΔFF 求和来计算 CV 值。其中 ΔFF = FF - FF_{n-1}。当 FFSetPrevious 已设置时， FF_{n-1} = FFPrevious。则您可在指令计算 ΔFF 值前使用指定值预设 FF_{n-1}。



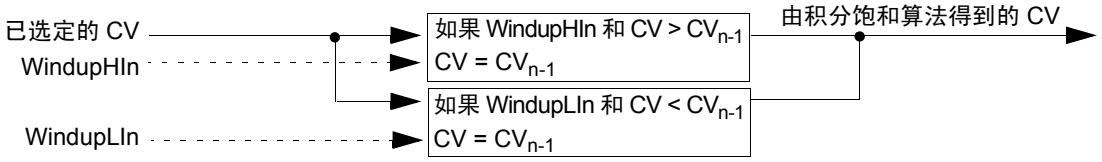
选择控制变量

如 PID 算法已执行，必须根据程序控制或操作员控制以及当前 PID 模式选择 CV 值。



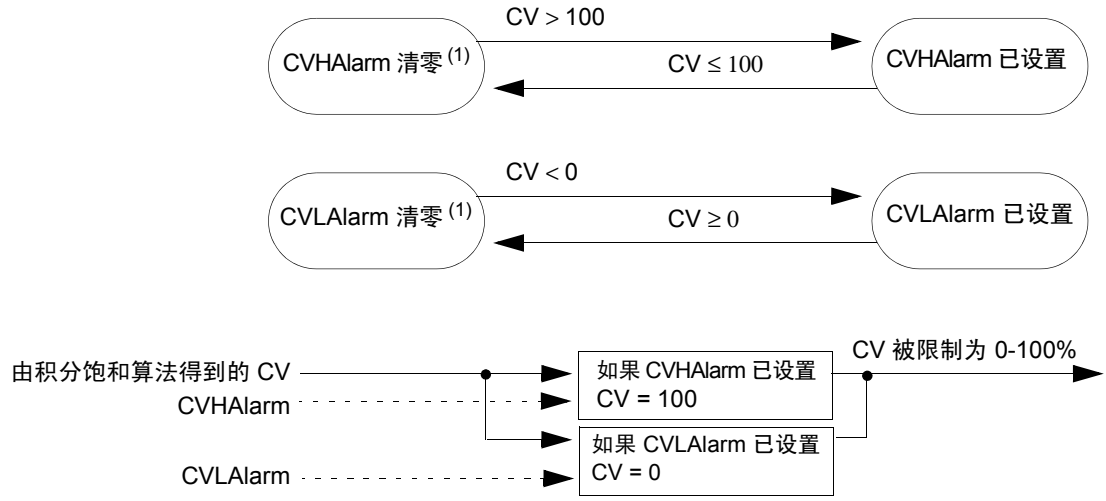
CV 积分饱和限制

当 WindupHIn 已设置时，限制 CV 值不能增加，或当 WindupLIn 已设置时，限制 CV 值不能减小。这些输入通常来自副回路的 WindupHOut 或 WindupLOut 输出。如果 CVInitializing、 CVFault 或 CVEUSpanInv 为已设置，则 WindupHIn 和 WindupLIn 输入将被忽略。



CV 百分数限制

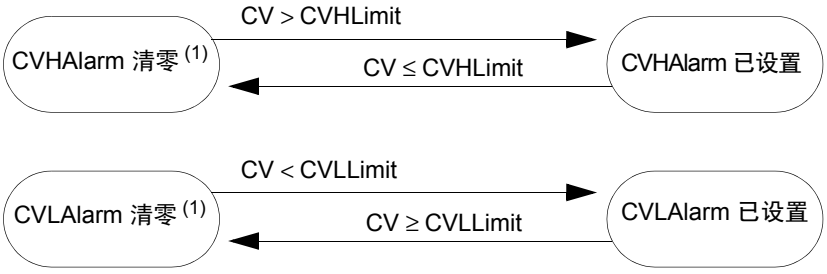
下图所示为指令如何确定 CV 百分数极限。

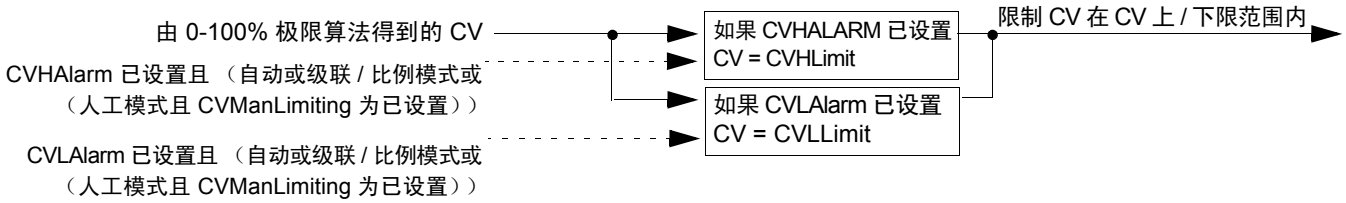


(1) 在指令首次扫描期间，指令将报警输出清零。

CV 上 / 下限

该指令根据 CVHLimit 和 CVLLimit 来执行报警。在自动或级联 / 比例模式下，CV 由 CVHLimit 和 CVLLimit 限制。处于人工模式时，如果 CVManLimiting 已设置，则 CV 由 CVHLimit 和 CVLLimit 限制。否则，CV 极限范围为 0 到 100%。





(1) 在指令首次扫描期间，指令将报警输出清零。

CV 值变化率限制

在自动或级联 / 比例模式，或当 CVManLimiting 已设置时在人工模式中，使用 PIDE 指令限制 CV 的变化率。零值可禁用 CV 变化率限制。

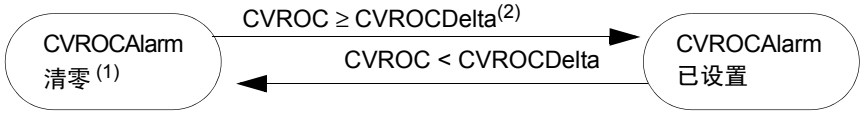
CV 变化率计算公式如下：

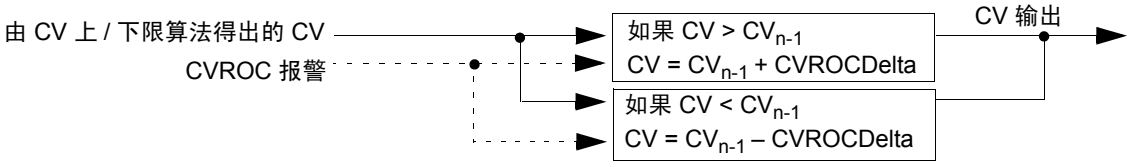
$$CVROC = \left| CV_n - CV_{n-1} \right|$$

$$CVROCDelta = CVROCLimit \times DeltaT$$

其中 DeltaT 以秒为单位。

一旦指令计算出 CV 变化率， CV 变化率报警由以下关系决定：



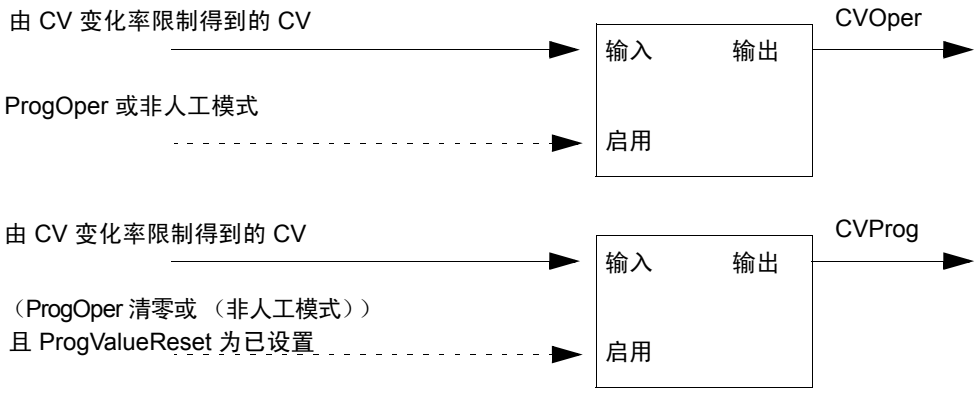


(1) 在指令首次扫描期间，指令将报警输出清零。同时，指令在 CVInitializing 已设置时将报警输出清零并禁用 CV 变化率算法。

(2) 在自动或级联 / 比例模式，或当 CVManLimiting 已设置时在人工模式中，指令限制 CV 的变化量。

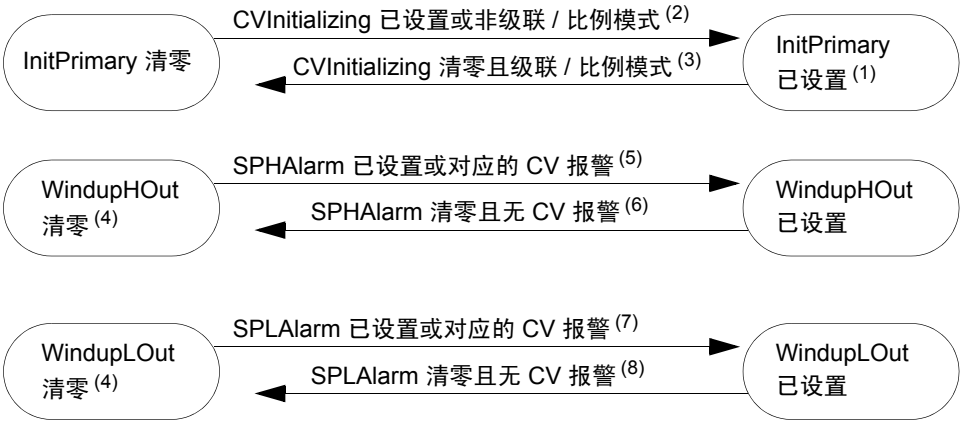
更新 CVOper 和 CVProg 值

如果未处于操作员人工模式，则 PIDE 指令设置 CVOper = CV。这可实现从任意控制模式到操作员人工模式的无扰切换。



主回路控制

在级联 / 比例模式中，主 PID 回路通常使用主回路控制来实现无扰切换和防止再调积分饱和。主回路控制包括初始化主 PID 回路输出和防止输出再调积分饱和。 InitPrimary 输出通常由主 PID 回路的 CVInitReq 输入来控制。积分饱和输出通常用作主回路的积分饱和输入控制来限制其 CV 输出积分饱和。



(1) 在指令首次扫描期间，指令设置 InitPrimary。

(2) 当 CVInitializing 已设置或处于非级联 / 比例模式时，指令设置 InitPrimary。

(3) 当 CVInitializing 清零且处于级联 / 比例模式时，指令清零 InitPrimary。

(4) 在指令首次扫描期间，指令将积分饱和输出清零。同时，指令在 CVFaulted 或 CVEUSpanInv 已设置时将积分饱和输出清零并禁用 CV 积分饱和算法。

(5) 当 SPHAlarm 已设置或 ControlAction 清零且 CVHAlarm 已设置，或当 ControlAction 已设置且 CVLAlarm 已设置时，指令设置 WindupHOut。

SP 和 CV 的限制操作是独立的。SP 上限不限制 CV 值的增加。同样， CV 上下限也不对 SP 值的增加进行限制。

(6) 当 SPHAlarm 清零且非 （ControlAction 清零且 CVHAlarm 已设置），并且非（ControlAction 已设置且 CVLAlarm 已设置）时，指令清零 WindupHOut。

(7) 当 SPLAlarm 已设置或 ControlAction 清零且设置 CVLAlarm，或当 ControlAction 已设置且 CVHAlarm 已设置时，指令设置 WindupLOut。

SP 和 CV 的限制操作是独立的。SP 下限不限制 CV 值的增加。同样， CV 上下限也不对 SP 值的增加进行限制。

(8) 当 SPLAlarm 清零且非 （ControlAction 清零且 CVLAlarm 已设置），并且非（ControlAction 已设置且 CVHAlarm 已设置）时，指令清零 WindupLOut。

错误处理

下表描述了指令如何对错误进行处理：

错误条件：	动作：
CVFaulted 已设置**或** CVEUSpanInv 已设置	• 指令未初始化， CVInitializing 清零 • 计算 PV 和 SP 百分数，计算误差，更新 EPercent 和 PVPIDPercent 的内部参数 • PID 控制算法不执行 • 禁用自动和级联 / 比例模式。如果当前模式不是越过或手动，则设置为人工模式。 • 由程序或操作员控制和模式 （人工、越过或手动）来设置 CV 值。
PVFaulted 已设置	• 禁用自动和级联 / 比例模式。如果当前模式不是越过或手动，则设置为人工模式 • PV 上下限、 PV 变化率、偏离量上 / 下限报警输出均被清零 • PID 控制算法不执行 • 由程序或操作员控制和模式 （人工、越过或手动）来设置 CV 值。
PVSpanInv 已设置**或** SPLimitsInv 已设置	• 禁用自动和级联 / 比例模式。如果当前模式不是越过或手动，则设置为人工模式 • 不计算 PV 和 SP 百分数 • PID 控制算法不执行 • 由程序或操作员控制和模式 （人工、越过或手动）来设置 CV 值。
RatioLimitsInv 已设置**且** CasRat 已设置**且** UseRatio 已设置	• 如果当前不处于越过或手动模式，则设置为人工模式 • 禁用级联 / 比例模式 • 由程序或操作员控制和模式 （人工、越过或手动）来设置 CV 值。
TimingModeInv 已设置**或** RTSTimeStampInv 已设置**或** DeltaTInv 已设置	• 如果当前不处于越过或手动模式，则设置为人工模式

位置比例 (POSP)

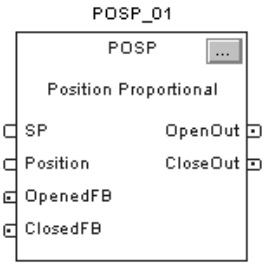
POSP 指令以用户定义的周期时间，使用与期望位置和实际位置之间差距成比例的脉冲宽度，打开或关闭触点，从而打开或关闭设备 （如电动操作阀门）。

操作数：

```
POSP(POSP_tag);
```

结构文本

操作数：	类型：	格式：	说明：
POSP 标记	POSITION_PROP	结构	POSP 结构



功能块

操作数：	类型：	格式：	说明：
块标记	POSITION_PROP	结构	POSP 结构

POSITION_PROP 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果该位清零，则指令不执行且输出不更新。 如果该位已设置，则指令执行。 缺省状态为已设置。 **结构化文本：** 无效。指令执行。
SP	REAL	设定点。这是期望位置值。该值必须使用与 Position 相同的工程单位。 有效值 = 浮点数 缺省值 = 0.0
Position	REAL	位置反馈。该模拟输入来自设备的位置反馈。 有效值 = 浮点数 缺省值 = 0.0
OpenedFB	BOOL	打开的反馈。当设备完全打开时的输入信号。设置时， open 输出不允许开启。 缺省状态为清零。
ClosedFB	BOOL	闭合反馈。当设备完全闭合时的输入信号。设置时， close 输出不允许开启。 缺省状态为清零。
PositionEUMax	REAL	Position 和 SP 最大的标度值。 有效值 = 浮点数 缺省值 = 100.0

输入参数：	数据类型：	说明：
PositionEUMin	REAL	Position 和 SP 最小的标度值。 有效值 = 浮点数 缺省值 = 0.0
CycleTime	REAL	脉冲输出周期，单位为秒。零值对 OpenOut 和 CloseOut 清零。如果该值无效，则指令假定其值为零并设置状态字中的相应位。 有效值 = 正浮点数 缺省值 = 0.0
OpenRate	REAL	设备打开速率，单位为 %/ 秒。零值对 OpenOut 清零。如果该值无效，则指令假定其值为零并设置状态字中的相应位。 有效值 = 正浮点数 缺省值 = 0.0
CloseRate	REAL	设备闭合速率，单位为 %/ 秒。零值对 CloseOut 清零。如果该值无效，则指令假定其值为零并设置状态字中的相应位。 有效值 = 正浮点数 缺省值 = 0.0
MaxOnTime	REAL	脉冲最长打开或关闭时间，单位为秒。如果计算出的 OpenTime 或 CloseTime 大于该值，则被限定为该值。如果该值无效，则指令假定该值为 CycleTime 并设置状态字中的相应位。 有效值 = 0.0 到 CycleTime 缺省值 = 0.0
MinOnTime	REAL	脉冲最短打开或关闭时间，单位为秒。如果计算出的 OpenTime 或 CloseTime 小于该值，则被设为零。如果该值无效，则指令假定其值为零并设置状态字中的相应位。 有效值 = 0.0 到 MaxOnTime 缺省值 = 0.0
死区时间	REAL	克服设备摩擦力的额外脉冲时间，单位为秒。当设备改变方向或处于停止状态时，死区时间就加到 OpenTime 或 CloseTime。如果该值无效，则指令设置状态字中的相应位，并使 Deadtime = 0.0。 有效值 = 0.0 到 MaxOnTime 缺省值 = 0.0

输出参数：	数据类型：	说明：
EnableOut	BOOL	输出使能。
OpenOut	BOOL	打开设备的脉冲输出。
CloseOut	BOOL	闭合设备的脉冲输出。
PositionPercent	REAL	以位置范围百分数表示的位置反馈。该输出要设置算术状态标志。
SPPercent	REAL	以位置范围百分数表示的设定点。
OpenTime	REAL	当前周期的 OpenOutput 的脉冲时间，单位为秒。
CloseTime	REAL	当前周期的 CloseOutput 的脉冲时间，单位为秒。
Status	DINT	功能块的状态。
InstructFault (Status.0)	BOOL	指令检测到如下运行错误。该错误不是次要或主要的控制器错误。检查其他状态位以确定错误类型。
CycleTimeInv (Status.1)	BOOL	CycleTime 值无效。指令使用零值。

输出参数：	数据类型：	说明：
OpenRateInv (Status.2)	BOOL	OpenRate 值无效。指令使用零值。
CloseRateInv (Status.3)	BOOL	CloseRate 值无效。指令使用零值。
MaxOnTimeInv (Status.4)	BOOL	MaxOnTime 值无效。指令使用 CycleTime 值。
MinOnTimeInv (Status.5)	BOOL	MinOnTime 值无效。指令使用零值。
DeadtimeInv (Status.6)	BOOL	死区时间值无效。指令使用零值。
PositionPctInv (Status.7)	BOOL	计算的 PositionPercent 值超出范围。
SPPercentInv (Statius.8)	BOOL	计算的 SPPercent 值超出范围。
PositionSpanInv (Status.9)	BOOL	PositionEUMax = PositionEUMin。

说明： POSP 指令通常从 PID 指令输出接收期望的位置设定点。

位置和设定点值标度

每次指令执行时， PositionPercent 和 SPPercent 输出会更新。如果其中任何一个值超出范围 （小于 0% 或大于 100%），则设置状态字中的相应位。但不限制这些值。该指令通过以下公式计算这些值是否在范围内：

$$PositionPercent = \frac{Position - PositionEUMin}{PositionEUMax - PositionEUMin} \times 100$$

$$SPPercent = \frac{SP - PositionEUMin}{PositionEUMax - PositionEUMin} \times 100$$

POSP 指令如何使用内部周期定时器

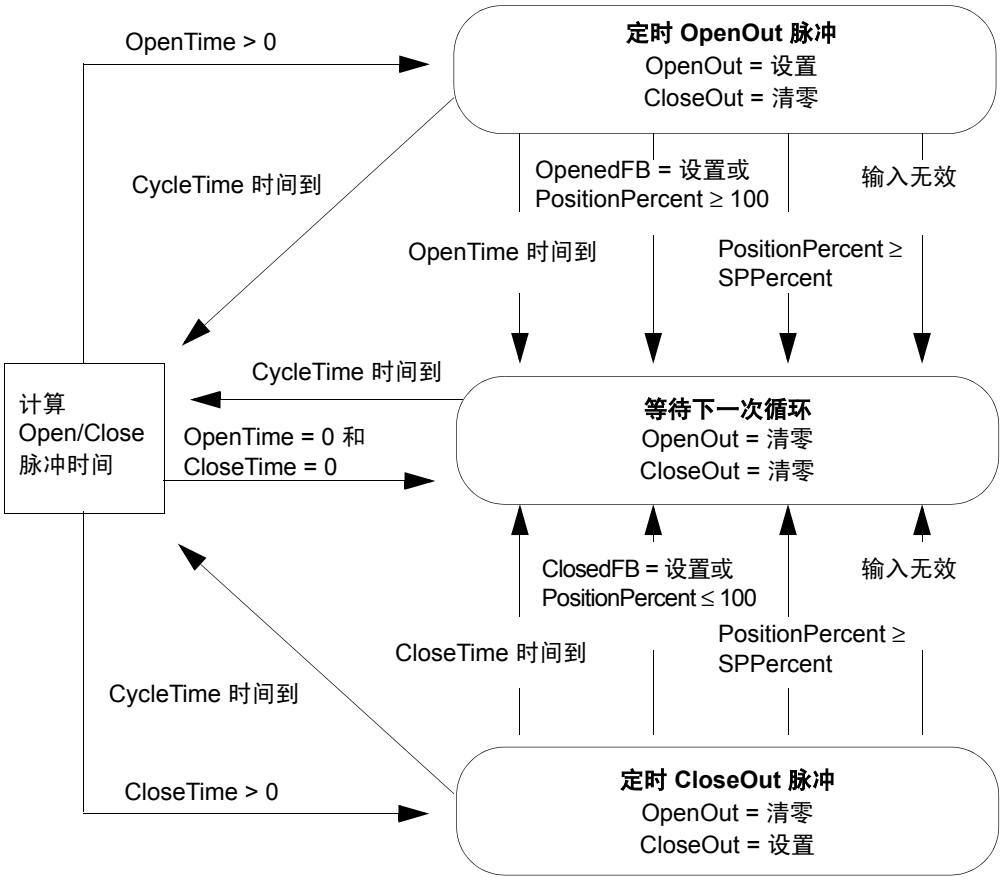
该指令使用 CycleTime 来决定多长时间重新计算 Open 和 Close 输出脉冲的持续时间。内部定时器由 DeltaT 来维护和更新。DeltaT 是自指令上次执行至今的时间。当内部定时器达到或超过程序设定的 CycleTime （周期时间到），则重新计算 Open 和 Close 输出。

用户可以随时改变 CycleTime。

如果 CycleTime = 0，则内部定时器清零， OpenOut 清零且 CloseOut 也清零。

产生输出脉冲

下图显示 POSP 指令的 3 个主要状态。



计算打开和闭合脉冲时间

当 SP > Position 反馈时， OpenOut 有脉冲输出。这时指令设定 CloseTime = 0， OpenOut 打开持续时间计算如下：

$$OpenTime = \frac{SPPercent - PositionPercent}{OpenRate}$$

- 如果 $OpenTime_{n-1}$ < CycleTime，则 OpenTime 加上 Deadtime。
- 如果 OpenTime > MaxOnTime，则设定极限值为 MaxOnTime。
- 如果 OpenTime < MinOnTime，则设定 OpenTime = 0。

如果有以下条件存在， OpenOut 没有脉冲输出且 OpenTime = 0。

- 设置 OpenFB 或 PositionPercent ≥ 100
- CycleTime = 0
- OpenRate = 0
- SPPercent 无效

当 SP < Position 反馈时， CloseOut 有脉冲输出。这时指令设定 OpenTime = 0， CloseOut 打开持续时间计算如下：

$$CloseTime = \frac{PositionPercent - SPPercent}{CloseRate}$$

- 如果 $CloseTime_{n-1}$ < CycleTime，则 CloseTime 加上 Deadtime。
- 如果 CloseTime > MaxOnTime，则设定极限值为 MaxOnTime。
- 如果 CloseTime < MinOnTime，则设定 CloseTime 为 0。

如果有以下条件存在， CloseOut 没有脉冲输出且 CloseTime 将被清零。

- 设置 ClosedFB 或 PositionPercent ≤ 0
- CycleTime = 0
- CloseRate = 0
- SPPercent 无效

如果 SPPercent = PositionPercent，则 OpenOut 和 CloseOut 没有脉冲输出。OpenTime 和 CloseTime 都将清零。

算术状态标志： PositionPercent 输出影响算术状态标志。

错误条件： 无

执行：

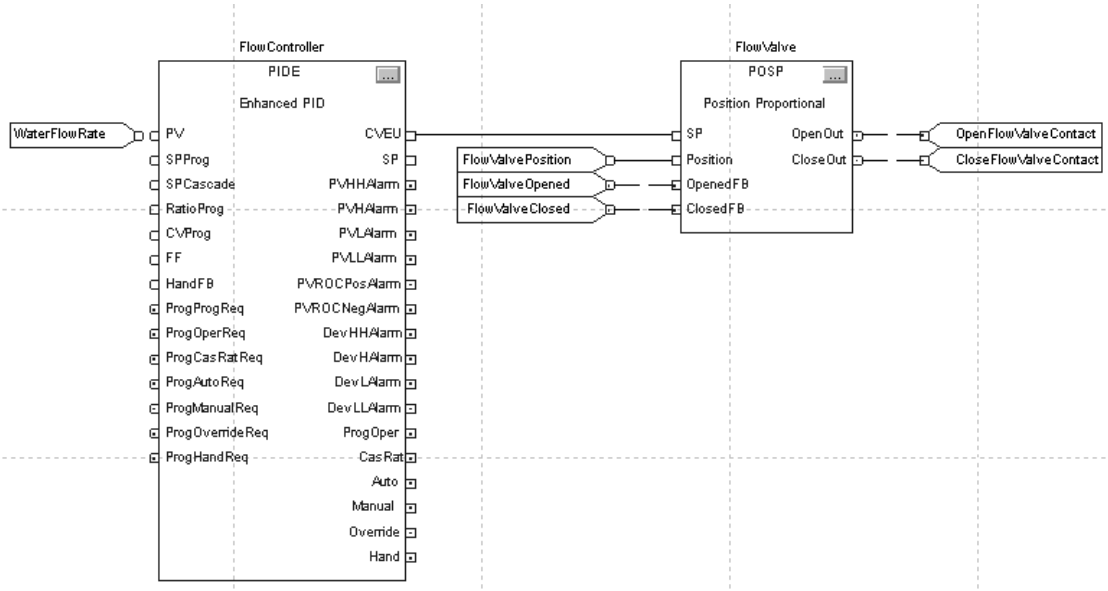
条件：	功能块动作：	结构化文本动作：
预扫描	OpenOut 和 CloseOut 清零。 OpenTime = 0 CloseTime = 0。	OpenOut 和 CloseOut 清零。 OpenTime = 0 CloseTime = 0。
指令首次扫描	内部周期定时器已复位。 指令计算 OpenTime 和 CloseTime。	内部周期定时器已复位。 指令计算 OpenTime 和 CloseTime。
指令首次执行	无动作。	无动作。
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。	不影响
EnableIn 已设置	指令执行。 EnableOut 已设置。	EnableIn 一直保持已设置状态。 指令执行。
后期扫描	无动作。	无动作。

示例： 在此例中，POSP 指令根据 PIDE 指令的 CVEU 输出来打开或关闭一个电动操作阀门。实际阀门位置连接到 Position 输入，用来指示阀门完全打开或完全关闭的限位开关是可选的，连接到 OpenedFB 和 ClosedFB 输入。 OpenOut 和 CloseOut 输出连接到电动操作阀的打开触点和闭合触点。

结构化文本

```
FlowController.PV := WaterFlowRate;
PIDE(FlowController);

FlowValve.SP := FlowController.CVEU;
FlowValve.Position := FlowValvePosition;
FlowValve.OpenedFB := FlowValveOpened;
FlowValve.ClosedFB := FlowValveClosed;
POSP(FlowValve);

OpenFlowValveContact := FlowValve.OpenOut;
CloseFlowValveContact := FlowValve.CloseOut;
```

功能块


FlowController
PIDE
Enhanced PID
WaterFlowRate
PV
SPProg
SPCascade
RatioProg
CVProg
FF
HandFB
ProgProgReq
ProgOperReq
ProgCasRatReq
ProgAutoReq
ProgManualReq
ProgOverrideReq
ProgHandReq
CVEU
SP
PVHHAlarm
PVHAlarm
PVLAlarm
PVLLAlarm
PVROCPosAlarm
PVROCNegAlarm
DevHHAlarm
DevHAlarm
DevLAlarm
DevLLAlarm
ProgOper
CasRat
Auto
Manual
Override
Hand
FlowValve
POSP
Position Proportional
SP
Position
OpenedFB
ClosedFB
OpenOut
CloseOut
FlowValvePosition
FlowValveOpened
FlowValveClosed
OpenFlowValveContact
CloseFlowValveContact

斜坡 / 渗透 (RMPS)

RMPS 指令提供一定数量的交互斜坡和渗透时间段。

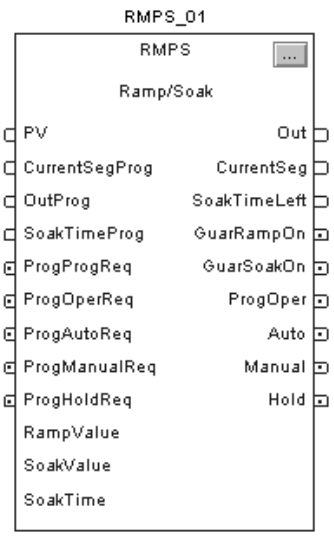
操作数：

```
RMPS(RMPS_tag,RampValue,
     SoakValue,SoakTime);
```

结构化文本

操作数：	类型：	格式：	说明：
RMPS 标记	RAMP_SOAK	结构	RMPS 结构
RampValue	REAL	数组	Ramp 值数组。输入每一段的斜率值（0 到 NumberOfSegs-1）。斜率值作为时间输入 （以分钟为单位），或作为比例输入 （以 “单位 / 分钟” 为单位）。 TimeRate 参数反映出用什么方法来指定斜率。如果斜率值无效，指令将设置状态字中的相应位并切换到操作员人工模式或程序保持模式。数组必须至少与 NumberOfSegs 一样大。有效值 = 0.0 到最大正浮点数
SoakValue	REAL	数组	渗透值数组。输入每一段的渗透值（0 到 NumberOfSegs-1）。数组必须至少与 NumberOfSegs 一样大。有效值 = 浮点数
SoakTime	REAL	数组	渗透时间数组。输入每一段的渗透时间（0 到 NumberOfSegs-1）。渗透时间输入值以分钟为单位。如果渗透值无效，指令将设置状态字中的相应位并切换到操作员人工模式或程序保持模式。数组必须至少与 NumberOfSegs 一样大。有效值 = 0.0 到最大正浮点数

功能块



这些操作数与结构化文本的 RMPS 指令相同。

RAMP_SOAK 结构：

输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果该位清零，则指令不执行且输出不更新。 如果该位已设置，则指令执行。 缺省状态为已设置。 **结构化文本：** 无效。指令执行。
PV	REAL	指令的标度模拟温度信号输入。 有效值 = 浮点数 缺省值 = 0.0
PVFault	BOOL	PV 故障指示。如果设置，则输入无效，指令切换到程序保持模式或操作员人工模式并设置状态字中的相应位。 缺省状态为清零。
NumberOfSegs	DINT	段数。指定指令用到的斜坡 / 渗透段数。RampValue、SoakValue 和 SoakTime 数组应不小于 NumberOfSegs。如果该数值无效，则指令切换到操作员人工模式或程序保持模式并设置状态字中的相应位。 有效值 = 1 到 （RampValue、 SoakValue 或 SoakTime 数组的最小大小） 缺省值 = 1
ManHoldAftInit	BOOL	初始化后进入人工 / 保持模式。设置时，初始化完成后斜坡 / 渗透进入操作员人工模式或程序保持模式。否则，初始化完成后斜坡 / 渗透保持前一状态。 缺省状态为清零。
CyclicSingle	BOOL	周期 / 单次执行。设置时按周期执行，清零时单次执行。周期执行不断地重复执行斜坡 / 渗透配置文件。单次执行则只执行一次斜坡 / 渗透配置文件就停止。 缺省状态为清零。
TimeRate	BOOL	组态时间 / 比例斜坡值。如果 RampValue 参数作为时间输入 （以分钟为单位）以达到渗透温度，则设置。如果 RampValue 参数作为比例输入 （以 “单位 / 分钟”为单位），则清零。 缺省状态为清零。
GuarRamp	BOOL	保障斜率。如果设置且指令处于自动模式，则 PV 与输出差值大于 RampDeadband 时，斜率暂时停止调节。 缺省状态为清零。
RampDeadband	REAL	保障斜率死区。当设置 GuarRamp 时，指定允许的 PV 与输出之间的工程单位差值。如果该值无效，则指令设定 RampDeadband = 0.0 并设置状态字中的相应位。 有效值 = 任意浮点数 ≥ 0.0 缺省值 = 0.0
GuarSoak	BOOL	保障渗透。如果设置且指令处于自动模式，当 PV 与输出差值大于 SoakDeadband 时，渗透定时器清零。 缺省状态为清零。
SoakDeadband	REAL	保障渗透死区。当设置 GuarSoak 时，指定允许的 PV 与输出之间的工程单位差值。如果该值无效，指令设定 SoakDeadband = 0.0 并设置状态字中的相应位。 有效值 = 任意浮点数 ≥ 0.0 缺省值 = 0.0

输入参数：	数据类型：	说明：
CurrentSegProg	DINT	当前程序段。用户程序向该输入中写入所要求的 CurrentSeg 值。该值在斜坡 / 渗透处于程序人工模式时使用。如果该值无效，则该指令设置状态字中的相应位。 有效值 = 0 到 NumberOfSegs-1 缺省值 = 0
OutProg	REAL	输出程序。用户程序向该输入中写入所要求的 Out 值。该值在斜坡 / 渗透处于程序人工模式时使用。 有效值 = 浮点数 缺省值 = 0.0
SoakTimeProg	REAL	渗透时间程序。用户程序向该输入中写入所要求的 SoakTimeLeft 值。该值在斜坡 / 渗透处于程序人工模式时使用。如果该值无效，则该指令设置状态字中的相应位。 有效值 = 0.0 到最大正浮点数 缺省值 = 0.0
CurrentSegOper	DINT	当前段操作员。操作员界面向该输入中写入所要求的 CurrentSeg 值。该值在斜坡 / 渗透处于操作员人工模式时使用。如果该值无效，则该指令设置状态字中的相应位。 有效值 = 0 到 NumberOfSegs-1 缺省值 = 0
OutOper	REAL	操作员输出。操作员界面往该输入中写入所要求的 Out 值。该值在斜坡 / 渗透处于操作员人工模式时使用。 有效值 = 浮点数 缺省值 = 0.0
SoakTimeOper	REAL	操作员渗透时间。操作员界面往该输入中写入所要求的 SoakTimeLeft 值。该值在斜坡 / 渗透处于操作员人工模式时使用。如果该值无效，则该指令设置状态字中的相应位。 有效值 = 0.0 到最大正浮点数 缺省值 = 0.0
ProgProgReq	BOOL	程序请求进入程序控制模式。通过用户程序设置来请求程序控制。如果 ProgOperReq 已设置，该请求被忽略。保持该位为已设置并将 ProgOperReq 清零，可锁定指令进入程序控制状态。 缺省状态为清零。
ProgOperReq	BOOL	程序请求进入操作员控制模式。通过用户程序设置来请求操作员控制。保持该位为已设置，可锁定指令进入操作员控制状态。 缺省状态为清零。
ProgAutoReq	BOOL	程序自动模式请求。由用户程序设置请求斜坡 / 渗透进入自动模式。如果该回路处于操作员控制或已设置 ProgManualReq 或已设置 ProgHoldReq 时，则忽略该请求。 缺省状态为清零。
ProgManualReq	BOOL	程序人工模式请求。由用户程序设置请求斜坡 / 渗透进入人工模式。如果斜坡 / 渗透处于操作员控制或当设置 ProgHoldReq 时，忽略该请求。 缺省状态为清零。
ProgHoldReq	BOOL	程序保持模式请求。由用户程序设置请求停止斜坡 / 渗透，而不改变 Out、CurrentSeg 或 SoakTimeLeft。 PID 回路从斜坡 / 渗透离开级联状态后达到设定点时，也会有用。操作员将斜坡 / 渗透切换到操作员人工模式也可实现相同的功能。 缺省状态为清零。

输入参数：	数据类型：	说明：
OperProgReq	BOOL	操作员请求进入程序控制模式。通过操作员界面设置来请求程序控制。如果 ProgOperReq 已设置，该请求被忽略。指令将该输入清零。 缺省状态为清零。
OperOperReq	BOOL	操作员请求进入操作员控制模式。通过操作员界面设置来请求操作员控制。如果设置 ProgProgReq 且 ProgOperReq 清零，则忽略该请求。指令将该输入清零。 缺省状态为清零。
OperAutoReq	BOOL	操作员自动模式请求。由操作员界面设置请求斜坡 / 渗透进入自动模式。如果该回路处于程序控制或当设置 OperManualReq 时，忽略该请求。指令将此输入清零。 缺省状态为清零。
OperManualReq	BOOL	操作员人工模式请求。由操作员界面设置请求斜坡 / 渗透进入人工模式。如果该回路处于程序控制时，则忽略该请求。指令将此输入清零。 缺省状态为清零。
Initialize	BOOL	初始化程序和操作员数值。如果设置且处于人工模式，则指令设定 CurrentSegProg = 0、CurrentSegOper = 0、SoakTimeProg = SoakTime[0] 及 SoakTimeOper = SoakTime[0]。处于自动或保持模式时，忽略初始化。 指令将该参数清零。 缺省状态为清零。
ProgValueReset	BOOL	复位程序控制值。设置时，指令清零 ProgProgReq、 ProgOperReq、 ProgAutoReq、 ProgHoldReq 和 ProgManualReq。 缺省状态为清零。

输入参数：	数据类型：	说明：
EnableOut	BOOL	输出使能。
Out	REAL	斜坡 / 渗透指令的输出。该输出要设置算术状态标志。
CurrentSeg	DINT	当前段号。显示斜坡 / 渗透周期的当前段号。段号起始于 0。
SoakTimeLeft	REAL	剩余渗透时间。显示当前渗透所剩余的渗透时间。
GuarRampOn	BOOL	保障斜率状态。当使用保障斜率功能且斜率因 PV 与输出差值大于 RampDeadband 暂缓计算时设置。
GuarSoakOn	BOOL	保障渗透状态。当使用保障渗透功能且渗透定时器因 PV 与输出差值大于 SoakDeadband 而清零时设置。
ProgOper	BOOL	程序 / 操作员控制指示。处于程序控制状态时为已设置。处于操作员控制时清零。
Auto	BOOL	自动模式。当斜坡 / 渗透处于程序自动模式或操作员模式时设置。
Manual	BOOL	人工模式。当斜坡 / 渗透处于程序人工模式或操作员人工模式时设置。
Hold	BOOL	保持模式。当斜坡 / 渗透处于程序保持模式时设置。
Status	DINT	功能块的状态。
InstructFault (Status.0)	BOOL	指令检测到如下运行错误。该错误不是次要或主要的控制器错误。检查其他状态位以确定错误类型。
PVFaulted (Status.1)	BOOL	PVHealth 状态不正常。

输入参数：	数据类型：	说明：
NumberOfSegsInv (Status.2)	BOOL	NumberOfSegs 值无效或与数组大小不兼容。
RampDeadbandInv (Status.3)	BOOL	RampDeadband 值无效。
SoakDeadbandInv (Status.4)	BOOL	SoakDeadband 值无效。
CurrSegProgInv (Status.5)	BOOL	CurrSegProg 值无效。
SoakTimeProgInv (Status.6)	BOOL	SoakTimeProg 值无效。
CurrSegOperInv (Status.7)	BOOL	CurrSegOper 值无效。
SoakTimeOperInv (Status.8)	BOOL	SoakTimeOper 值无效。
RampValueInv (Status.9)	BOOL	RampValue 值无效。
SoakTimeInv (Status.10)	BOOL	SoakTime 值无效。

说明： RMPS 指令通常用于为批处理加热过程提供温度配置。该指令的输出通常是 PID 回路的设定点输入。

输出的计算值为无效、 NAN 或 ± INF 时，指令设置 Out 等于该无效值并影响算术溢出状态标志。内部参数未更新。在后续每次扫描中，当输出有效时，使用来自前次扫描的内部参数计算输出。

监控 RMPS 指令

RMPS 指令具有操作员面板。有关更多信息，请参阅附录“功能块面板控件”。

算术状态标志： Out 输出影响算术状态标志。

错误条件： 无

执行：

条件：	功能块动作：	结构化文本动作：
预扫描	无动作。	无动作。
指令首次扫描	所有操作员请求输入均清零。 如果 ProgValueReset 已设置，则所有程序请求输入均清零。 如果当前模式为保持模式，则操作员控制模式设置为人工模式。 请参阅下表。	
指令首次执行	CurrentSegment = 0. 如果 SoakTime[0] 有效，则 SoakTimeProg 和 SoakTimeOper = SoakTime[0]。 模式设置为操作员人工模式。 Out_{n-1} = 0.0。	
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。	na
EnableIn 已设置	指令执行。 EnableOut 已设置。	EnableIn 一直保持已设置状态。 指令执行。
后期扫描	无动作。	无动作。

指令首次扫描应用的初始模式

下表显示最终控制模式 （根据程序请求输入）。

首次扫描初始控制模式：	Prog Oper Req:	Prog Prog Req:	Prog Value Reset:	首次运行：	首次扫描结束时的控制模式：
操作员控制	清零	设置	清零	不影响	程序控制
	不影响	清零	不影响	不影响	操作员控制
程序控制	设置	不影响	清零	清零	操作员控制
	不影响	不影响	设置	设置	
	清零	清零	清零	设置	
	清零	设置	清零	不影响	
	不影响	不影响	设置	清零	
	清零	清零	清零	清零	

下表显示最终控制模式 （根据人工、自动和保持模式请求）。

首次扫描初始控制模式：	Oper Auto Req:	Oper Man Req:	Prog Auto Req:	Prog Man Req:	Prog Hold Req:	Manual Hold After Init:	Prog Value Reset:	首次运行	首次扫描结束时的控制模式：
操作员控制	不影响	不影响	不影响	不影响	不影响	清零	不影响	清零	操作员当前模式
	不影响	不影响	不影响	不影响	不影响	不影响	不影响	设置	操作员人工模式
	不影响	不影响	不影响	不影响	不影响	设置	不影响	不影响	
程序控制	不影响	不影响	清零	清零	清零	清零	不影响	清零	程序当前模式
	不影响	不影响	不影响	不影响	不影响	清零	设置	清零	
	不影响	不影响	设置	清零	清零	清零	清零	不影响	程序自动模式
	不影响	不影响	不影响	设置	清零	清零	清零	不影响	程序人工模式
	不影响	不影响	不影响	不影响	设置	清零	清零	不影响	程序保持模式
	不影响	不影响	不影响	不影响	不影响	设置	不影响	不影响	

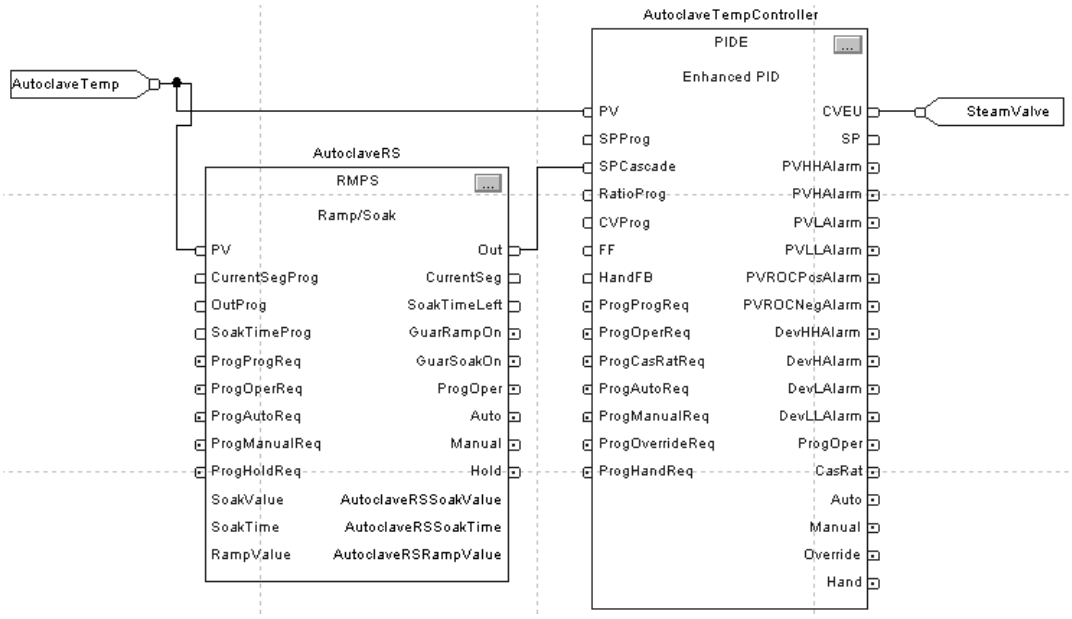
示例： 在此例中， RMPS 指令驱动 PIDE 指令的设定点。当 PIDE 指令处于级联 / 比例模式时， RMPS 指令的输出用作设定点。如果用户想使用保障斜率或保障渗透时，PIDE 指令的 PV 可选择性地连接到 RMPS 指令的 PV 输入上。

在此例中， AutoclaveRSSoakValue、 AutoclaveRSSoakTime 和 AutoclaveRSRampValue 数组都是 10 个元素的 REAL 数组，最多支持 10 段 RMPS 配置。

结构化文本

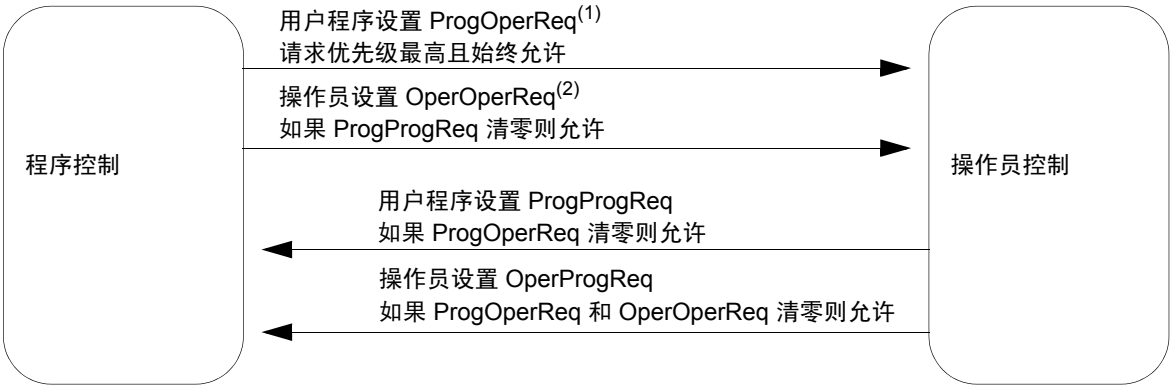
```
AutoclaveRS.PV := AutoclaveTemp;
RMPS (AutoclaveRS,AutoclaveRSRampValue,
    AutoclaveRSSoakValue,AutoclaveRSSoakTime);


AutoclaveTempController.PV := AutoclaveTemp;
AutoclaveTempController.SPCascade := AutoclaveRS.Out;
PIDE(AutoclaveTempController);


SteamValve := AutoclaveTempController.CVEU;
```

功能块



在程序控制和操作员控制之间切换

RMPS 指令可通过用户程序或操作员界面进行控制。可随时改变控制模式。



(1) 保持 ProgOperReq 为设置状态，可将指令锁定为操作员控制模式。

(2) 当 ProgOperReq 清零时保持 ProgProgReq 为设置状态，可将指令锁定为程序控制模式

有关程序控制和操作员控制的更多信息，请参阅第 B-13 页。

当 ProgAutoReq、 ProgManualReq 和 ProgHoldReq 输入清零时，从操作员控制切换到程序控制时，其模式由以下条件决定：

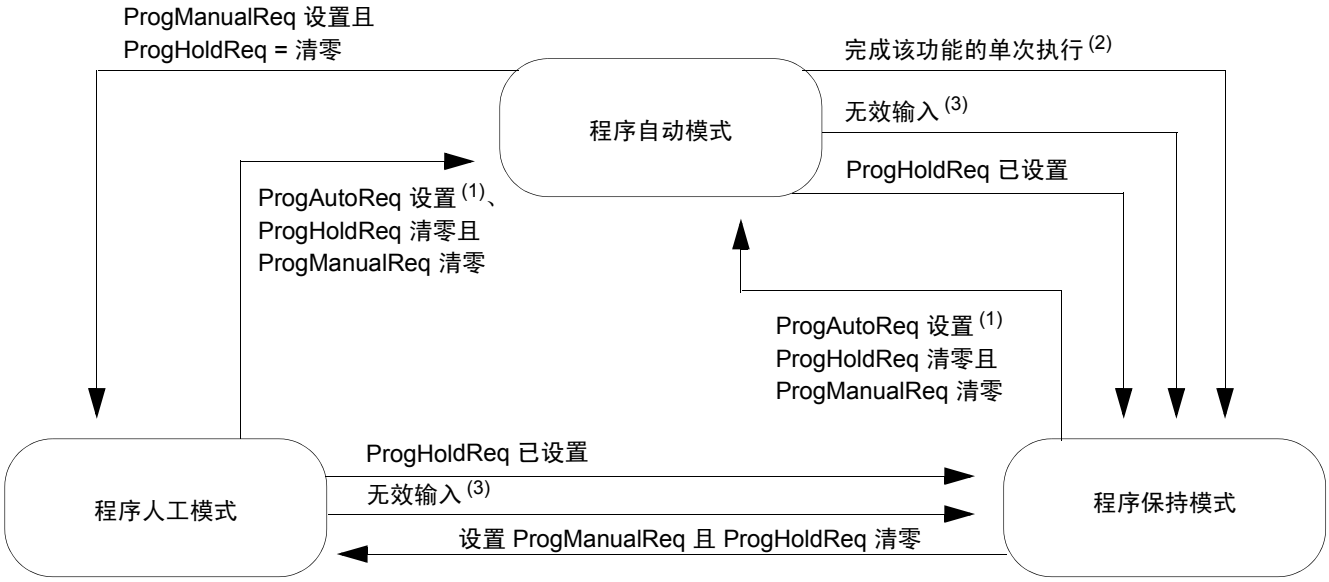
- 如果指令处于操作员自动模式，则切换到程序自动模式。
- 如果指令处于操作员人工模式，则切换到程序人工模式。

当 OperAutoReq 和 OperManualReq 输入清零时，指令从程序控制切换到操作员控制，其模式由以下条件决定：

- 如果指令处于程序自动模式，则切换到操作员自动模式。
- 如果指令处于程序人工或程序保持模式，则切换到操作员人工模式。

程序控制

下图演示了在程序控制模式下， RMPS 指令如何操作。



(1) 在单次 （非周期）执行时，如果已经有一个斜坡 / 渗透执行完成，且用户要执行另一次，则需将 ProgAutoReq 状态由清零转换到设置。

(2) 如果指令被配置为单次执行且自动模式的斜坡 / 渗透功能已完成，则指令进入保持模式。

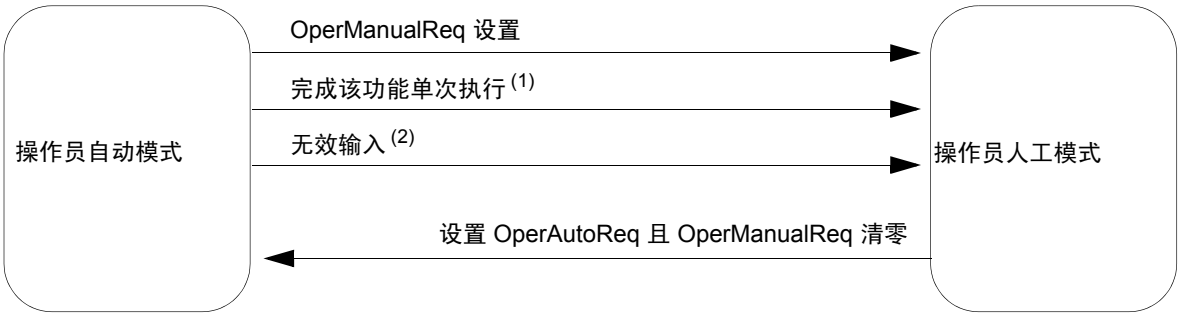
(3) 如果 PVFaulted 设置或者以下任何输入无效，指令将进入保持模式：NumberOfSegs、CurrentSeg、 SoakTimeLeft、 CurrentSegProg 或 SoakTimeProg。

下表说明了可能出现的程序模式。

模式：	说明：
程序自动模式	在自动模式中，指令连续地执行斜坡 / 渗透功能。
程序人工模式	在人工模式中，用户程序直接控制指令输出。 CurrentSegProg、SoakTimeProg 和 OutProg 输入被转化到 CurrentSeg、SoakTimeLeft 和 Out 输出。当指令处于自动模式，斜坡 / 渗透函数从用户程序上次输入值重新开始。如果 CurrentSegProg 和 SoakTimeProg 无效则不切换。 为了实现无冲击地切换到人工模式，当设置 ProgValueReset 且指令不处于程序人工模式时， CurrentSegProg、 SoakTimeProg 和 OutProg 输入不断地被当前的 CurrentSeg、 SoakTimeLeft 和 Out 值所更新。
程序保持模式	处于保持模式时，指令的输出保持其当前值。在此模式下，如果设置 ProgOperReq 来切换为操作员控制，则指令切换到操作员人工模式。

操作员控制

下图演示了在操作员控制模式下， RMPS 指令如何操作。



(1) 如果指令被组态为单次执行，且自动模式的斜坡 / 渗透功能已完成，则指令进入人工模式。

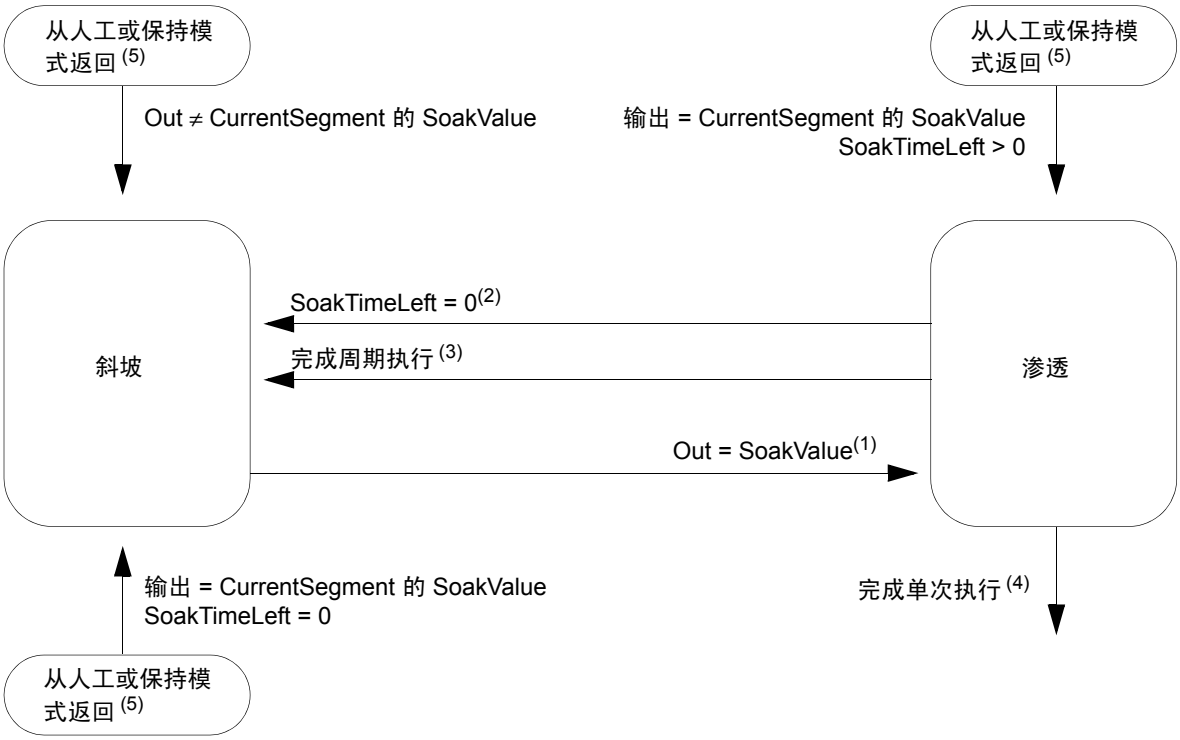
(2) 如果设置 PVFaulted 或以下任何输入无效，指令将进入人工模式：NumberOfSegs、CurrentSeg、 SoakTimeLeft、 CurrentSegOper 或 SoakTimeOper。

下表说明了所有可能的操作员模式

模式：	说明：
操作员自动模式	处于自动模式时，指令连续地执行斜坡 / 渗透功能
操作员人工模式	在人工模式中，操作员直接控制指令的输出。将 CurrentSegOper、SoakTimeOper 和 OutOper 输入转化到 CurrentSeg、SoakTimeLeft 和 Out 输出。当指令处于自动模式，斜坡 / 渗透函数从操作员上次输入值重新开始。如果 CurrentSegOper 和 SoakTime 无效则不切换。 为了实现无冲击切换到人工模式，指令不处于操作员人工模式时，CurrentSegOper、SoakTimeOper 和 OutOper 输入不断地被当前的 CurrentSeg、 SoakTimeLeft 和 Out 值所更新。

执行斜坡 / 渗透功能

下图演示了 RMPS 指令如何执行斜坡 / 渗透功能。



(1) 当 Out = SoakValue 时完成斜坡。如果在斜坡执行期间， Out > SoakValue， Out 被限定为 SoakValue。

(2) 当 Out 保持了当前段 SoakTime 指定的时间后，渗透完成。如果所执行的段不是最后一段，则 CurrentSeg 加 1。

(3) 渗透已经执行完程序设定的最后一段，且指令被配置为周期执行。则指令设置 CurrentSeg = 0.0。

(4) 渗透已经执行完程序设定的最后一段，且指令被配置为单次执行。

(5) 当指令返回自动模式，指令判断是否重新执行斜坡或渗透。下一个步骤取决于当前段 Out、SoakTimeLeft 和 SoakValue 的值。如果 Out = 当前段的 SoakValue 且 SoakTimeLeft = 0，则当前段完成，开始执行下一段。

斜率

斜率循环地将前一段的 SoakValue 数值更新为当前段的 SoakValue 数值。斜率转换的时间是由 RampValue 参数定义的。

当 Out < 当前段的目标 SoakValue 值，斜率为正。如果斜率公式计算出的新 Out 数值超过目标 SoakValue，则 Out 被设置为目标 SoakValue。

如果 Out > 当前段的目标 SoakValue 值，则斜率为负。如果斜率公式计算出的新的 Out 数值小于目标 SoakValue，则 Out 被设置为目标 SoakValue。

每个段都有斜率值。用户可选择以时间单位或比例单位设定斜率。但所有段必须以相同的单位设定。下表说明了斜率的选项：

参数：	说明：
基于时间的斜率	对于基于时间的斜率，设置 TimeRate （以分钟为单位） 计算当前段的变化速率，并对 Out 加、减该速率，直到 Out 达到当前段的渗透值。在以下公式中， DeltaT 是自上次指令执行到现在的时间，单位为分钟。 $$Out = Out \pm \frac{(SoakValue_{CurrentSeg} - RampStart)}{RampValue_{CurrentSeg}} \times \Delta t$$ 其中 RampStart 是当前段开始时的 Out 值。
速率表示的斜率	对于基于速率的斜率，将 TimeRate 清零 （以 “单位 / 分钟” 为单位） 对 Out 加、减程序设定的变化速率，直到 Out 达到当前段的渗透值。在以下公式中， DeltaT 是自上次指令执行到现在的时间，单位为分钟。 $$Out = Out \pm RampValue_{CurrentSeg} \times \Delta t$$

保障斜率

设置输入 GuarRamp 以启用保障斜率。启用时，指令监控 Out 和 PV 之间的差异。如果差值大于程序设定的 RampDeadband，则输出保持不变直到 PV 和 Out 的差值位于死区内。当 Out 因保障斜率生效而保持不变时，设置输出 GuarRampOn。

渗透

渗透是块输出保持不变直到下一个斜坡 / 渗透段开始的时间值。渗透周期在程序设定的时间内保持输出为 SoakValue，然后才开始下一段。输出要渗透的时间在 SoakTime 参数中由程序设定。

每段都有 SoakValue 和 SoakTime。当 Out 达到当前段的 SoakValue 值时开始渗透。 SoakTimeLeft 表示输出渗透所剩余的时间，以分钟为单位。在执行斜率期间， SoakTimeLeft 被设置为当前段的 SoakTime。一旦执行斜率结束， SoakTimeLeft 会减小，它反映当前段的剩余时间，以分钟为单位。当 SoakTime 时间到时， SoakTimeLeft = 0。

保障渗透

设置输入 GuarSoak 以启用保障渗透。启用时，指令监控 Out 和 PV 之间的差异。如果差值大于 SoakDeadband，则渗透周期定时暂停，且内部渗透定时器清零。当 Out 和 PV 之间的差值回到死区以内时，重新开始定时。当定时因保障渗透生效而保持不变时，设置输出 GuarSoak。

标度 (SCL)

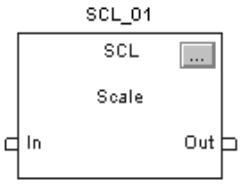
SCL 指令将一个非标度的输入值转换成浮点型工程单位值。

操作数：

```
SCL(SCL_tag);
```

结构文本

操作数：	类型：	格式：	说明：
SCL 标记	SCALE	结构	SCL 结构



功能块

操作数：	类型：	格式：	说明：
SCL 标记	SCALE	结构	SCL 结构

SCALE 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果该位清零，则指令不执行且输出不更新。 如果该位已设置，则指令执行。 缺省状态为已设置。 **结构化文本：** 无效。指令执行。
In	REAL	模拟量信号输入。 有效值 = 任意实数值 缺省值 = 0.0
InRawMax	REAL	指令输入可达到的最大值。如果 InRawMax ≤ InRawMin，则指令设置状态字中的相应位并停止更新输出。 有效值 = InRawMax > InRawMin 缺省值 = 0.0
InRawMin	REAL	指令输入可达到的最小值。如果 InRawMin ≥ InRawMax，则指令设置状态字中的相应位并停止更新输出。 有效值 = InRawMin < InRawMax 缺省值 = 0.0
InEUMax	REAL	与 InRawMax 对应的输入标度值。 有效值 = 任意实数值 缺省值 = 0.0
InEUMin	REAL	与 InRawMin 对应的输入标度值。 有效值 = 任意实数值 缺省值 = 0.0
极限	BOOL	极限选择。如果设置，则输出被限制在 InEUMin 和 InEUMax 之间。 缺省状态为清零。

输出参数：	数据类型：	说明：
EnableOut	BOOL	输出使能。
Out	REAL	表示模拟输入标度值的输出。该输出要设置算术状态标志。 有效值 = 任意实数值 缺省值 = InEUMin
MaxAlarm	BOOL	输入最大上限报警指示。当 In> InRawMax 时设置该值。
MinAlarm	BOOL	输入最小下限报警指示。当 In < InRawMin 时设置该值。
Status	DINT	功能块的状态。
InstructFault (Status.0)	BOOL	指令检测到如下运行错误。该错误不是次要或主要的控制器错误。检查其他状态位以确定错误类型。
InRawRangeInv (Status.1)	BOOL	InRawMin ≥ InRawMax。

说明： SCL 指令用于不支持全分辨率浮点数标度的模拟输入模块。

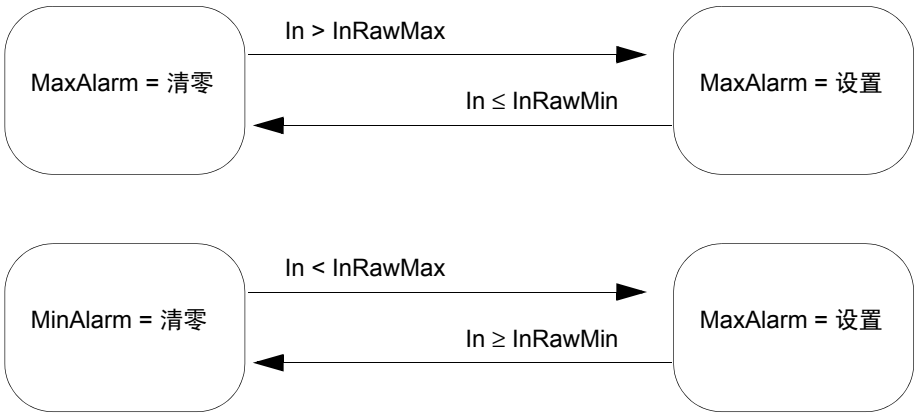
例如：1771-IFE 模块是 12 位的模拟输入模块，只支持整数值标度。如果用一个 1771-IFE 模块读取 0-100 加仑每分钟 (gpm) 的流量值，通常不将模块标度为 0-100，因为这样限制模块分辨率。而是使用 SCL 指令，并配置模块返回一个非标度的 (0-4095) 数值， SCL 指令将其无精度损失地转换为 0-100 gpm （浮点数）。标度后的数值可当作其他指令的输入。

SCL 指令使用以下算法将非标度的输入转换为标度值：

$$Out = (In - InRawMin) \times \left(\frac{InEUMax - InEUMin}{InRawMax - InRawMin} \right) + InEUMin$$

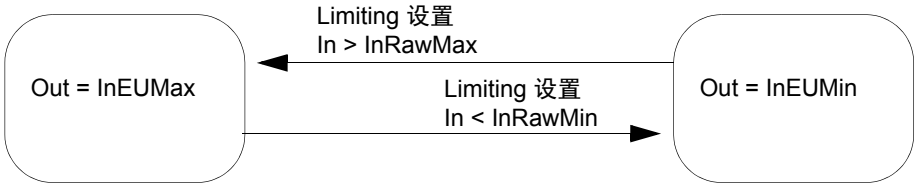
报警

一旦指令计算出 Out 值， MaxAlarm 和 MinAlarm 就由以下关系确定：



极限

设置 Limiting 时，将限定 Out。当 In > InRawMax 时，指令设置 Out = InEUMax。当 In < InRawMin 时，指令设置 Out = InEUMin。



算术状态标志： Out 输出影响算术状态标志。

错误条件： 无

执行：

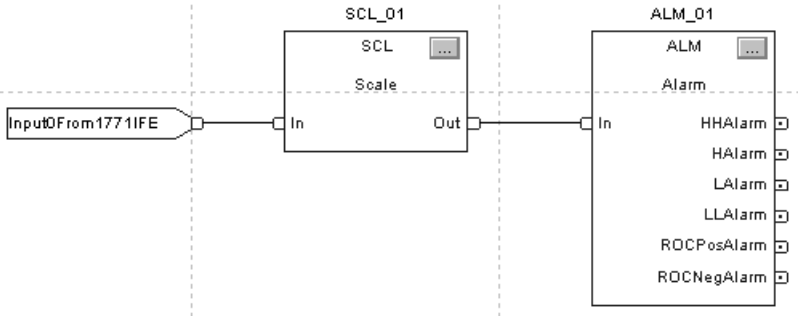
条件：	功能块动作：	结构化文本动作：
预扫描	无动作。	无动作。
指令首次扫描	无动作。	无动作。
指令首次扫描	无动作。	无动作。
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。	不影响
EnableIn 已设置	指令执行。 EnableOut 已设置。	EnableIn 一直保持已设置状态。 指令执行。
后期扫描	无动作。	无动作。

示例： SCL 指令通常与不支持在线浮点数工程单位标度的模拟输入模块配合使用。在此例中，SCL 指令标度来自 1771-IFE 模块的模拟输入。指令将结果放在 Out 中供 ALM 指令使用。

结构化文本

```
SCL_01.In := Input0From1771IFE;
SCL(SCL_01);

ALM_01.In := SCL_01.Out;
ALM(ALM_01);
```

功能块



拆分范围时间比例 (SRTP)

SRTP 指令获取 PID 回路的 0-100% 输出，并以周期性脉冲信号来驱动加热和制冷数字输出触点。该指令可用于冲压机器的料管温度控制等应用。

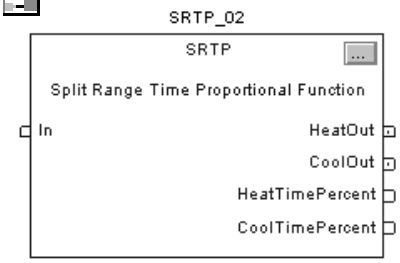
操作数：

```
SRTP(SRTP_tag);
```

结构文本

操作数：	类型：	格式：	说明：
SRTP 标记	SPLIT_RANGE	结构	SRTP 结构

功能块



操作数：	类型：	格式：	说明：
SRTP 标记	SPLIT_RANGE	结构	SRTP 结构

SPLIT_RANGE 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果该位清零，则指令不执行且输出不更新。 如果该位已设置，则指令执行。 缺省状态为已设置。 **结构化文本：** 无效。指令执行。
In	REAL	请求加热或制冷的模拟信号输入。该输入通常来自 PID 回路的 CVEU。 有效值 = 浮点数
CycleTime	REAL	输出脉冲周期，以秒为单位。该值为 0 将关闭加热和制冷输出。如果该值无效，则指令假定其值为零并设置状态字中的相应位。 有效值 = 正浮点数 缺省值 = 0.0
MaxHeatIn	REAL	最大加热输入。该值指定产生最大加热的 In 的百分数。加热 / 制冷回路中通常为 100%。 有效值 = 浮点数 缺省值 = 100.0
MinHeatIn	REAL	最小加热输入。该值指定表示加热开始以及产生最小加热的 In 的百分数。加热 / 制冷回路中通常为 50%。 有效值 = 浮点数 缺省值 = 50.0

输入参数：	数据类型：	说明：
MaxCoolIn	REAL	最大制冷输入。该值指定产生最大制冷的 In 的百分数。加热 / 制冷回路中通常为 0%。 有效值 = 浮点数 缺省值 = 0.0
MinCoolIn	REAL	最小制冷输入。该值指定产生最小制冷的 In 的百分数。加热 / 制冷回路中通常为 50%。 有效值 = 浮点数 缺省值 = 50.0
MaxHeatTime	REAL	最大加热时间，以秒为单位。指定以秒为单位的最大加热脉冲时间。如果指令计算出的 HeatTime 大于该数值，则 HeatTime 被限定为 MaxHeatTime。如果 MaxHeatTime 无效，则指令设定其值为 CycleTime 并设置状态字中的相应位。 有效值 = 0.0 到 CycleTime 缺省值 = 0.0
MinHeatTime	REAL	最小加热时间，以秒为单位。指定以秒为单位的最小加热脉冲时间。如果指令计算出的 HeatTime 小于该数值，则 HeatTime 被设置为零。如果 MinHeatTime 无效，则指令设定其值为零并设置状态字中的相应位。 有效值 = 0.0 到 MaxHeatTime 缺省值 = 0.0
MaxCoolTime	REAL	最大制冷时间，以秒为单位。指定以秒为单位的最大制冷脉冲时间。如果指令计算出的 CoolTime 大于该数值，则 CoolTime 被限定为 MaxCoolTime。如果 MaxCoolTime 无效，则指定设定其值为 CycleTime 并设置状态字中的相应位。 有效值 = 0.0 到 CycleTime 缺省值 = 0.0
MinCoolTime	REAL	最小制冷时间，以秒为单位。指定以秒为单位的最小制冷脉冲时间。如果指令计算出的 CoolTime 小于该数值，则 CoolTime 被设置为零。如果 MinCoolTime 无效，则指令设定其值为零并设置状态字中的相应位。 有效值 = 0.0 到 MaxCoolTime 缺省值 = 0.0

输出参数：	数据类型：	说明：
EnableOut	BOOL	输出使能。
HeatOut	BOOL	加热输出脉冲。指令输出脉冲信号给加热触点。
CoolOut	BOOL	制冷输出脉冲。指令输出脉冲信号给制冷触点。
HeatTimePercent	REAL	以百分数表示的加热输出脉冲时间。该数值表示 HeatingOutput 占当前周期的百分数。这样如果需要用户就可以使用模拟输出指令来加热。该输出要设置算术状态标志。
CoolTimePercent	REAL	以百分数表示的制冷输出脉冲时间。该数值表示 CoolingOutput 占当前周期的百分数。这样如果需要用户就可以使用模拟输出指令来制冷。该输出要设置算术状态标志。
Status	DINT	功能块的状态。
InstructFault (Status.0)	BOOL	指令检测到如下运行错误。该错误不是次要或主要的控制器错误。检查其他状态位以确定错误类型。
CycleTimeInv (Status.1)	BOOL	CycleTime 值无效。指令使用零值。

输出参数：	数据类型：	说明：
MaxHeatTimeInv (Status.2)	BOOL	MaxHeatTime 值无效。指令使用 CycleTime 值。
MinHeatTimeInv (Status.3)	BOOL	MinHeatTime 值无效。指令使用零值。
MaxCoolTimeInv (Status.4)	BOOL	MaxCoolTime 值无效。指令使用 CycleTime 值。
MinCoolTimeInv (Status.5)	BOOL	MinCoolTime 值无效。指令使用零值。
HeatSpanInv (Status.6)	BOOL	MaxHeatIn = MinHeatIn。
CoolSpanInv (Status.7)	BOOL	MaxCoolIn = MinCoolIn。

说明： SRTP 脉冲的宽度与 PID 输出成比例。该指令参数用于加热和制冷应用。

使用内部周期定时器

该指令提供一个始终运行的从 0 到程序设定的 CycleTime 的周期定时器。该内部定时器由 DeltaT 来更新。 DeltaT 是自指令上次执行至今的时间。该定时器决定输出是否需要打开。

用户可以随时改变 CycleTime。如果 CycleTime = 0，则内部定时器清零并且 HeatOut 和 CoolOut 也清零。

计算加热和制冷时间

每次指令执行都计算加热和制冷时间。

HeatTime 是加热输出被打开时间数值，其值小于 CycleTime。

$$HeatTime = \frac{In - MinHeatIn}{MaxHeatIn - MinHeatIn} \times CycleTime$$

- 如果 HeatTime < MinHeatTime，则设置 HeatTime = 0。
- 如果 HeatTime> MaxHeatTime，则限定 HeatTime = MaxHeatTime。

HeatTimePercent 是 HeatOut 占 CycleTime 时间的百分数。

$$HeatTimePercent = \frac{HeatTime}{CycleTime} \times 100$$

CoolTime 是制冷输出被打开时间数值，其值小于 CycleTime。

$$CoolTime = \frac{In - MinCoolIn}{MaxCoolIn - MinCoolIn} \times CycleTime$$

- 如果 CoolTime < MinCoolTime，则设置 CoolTime = 0。
- 如果 CoolTime > MaxCoolTime，则限定 CoolTime = MaxCoolTime。

CoolTimePercent 是 CoolOut 占 CycleTime 时间的百分数。

$$CoolTimePercent = \frac{CoolTime}{CycleTime} \times 100$$

该指令使用以下规则控制加热和制冷输出：

- 如果 HeatTime ≥ 内部周期时间累加器，则设置 HeatOut。
  当内部周期定时器 > HeatTime 时，则 HeatOut 清零。
- 如果 CoolTime ≥ 内部周期时间累加器，则设置 CoolOut。
  当内部周期定时器 > CoolTime 时，则 CoolOut 清零。
- 如果 CycleTime = 0，则 HeatOut 和 CoolOut 清零。

算术状态标志： HeatTimePercent 和 CoolTimePercent 输出影响算术状态标志。

错误条件： 无

执行：

条件：	功能块动作：	结构化文本动作：
预扫描	HeatOut 和 CoolOut 清零。	HeatOut 和 CoolOut 清零。
指令首次扫描	内部周期定时器已复位。	内部周期定时器已复位。
指令首次执行	无动作。	无动作。
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。	不影响
EnableIn 已设置	指令执行。 EnableOut 已设置。	EnableIn 一直保持已设置状态。 指令执行。
后期扫描	无动作。	无动作。

示例： 在此例中， PIDE 指令用于控制缓慢的温度回路，因此在缓慢、优先级较低的任务中执行。 PIDE 指令的输出是控制器范围的标记，因为它将成为 SRTP 指令的输入。 SRTP 指令在高速、优先级较高的任务中执行，脉冲输出更精确。

结构化文本

将 PIDE 指令置于慢速且优先权较低的任务中

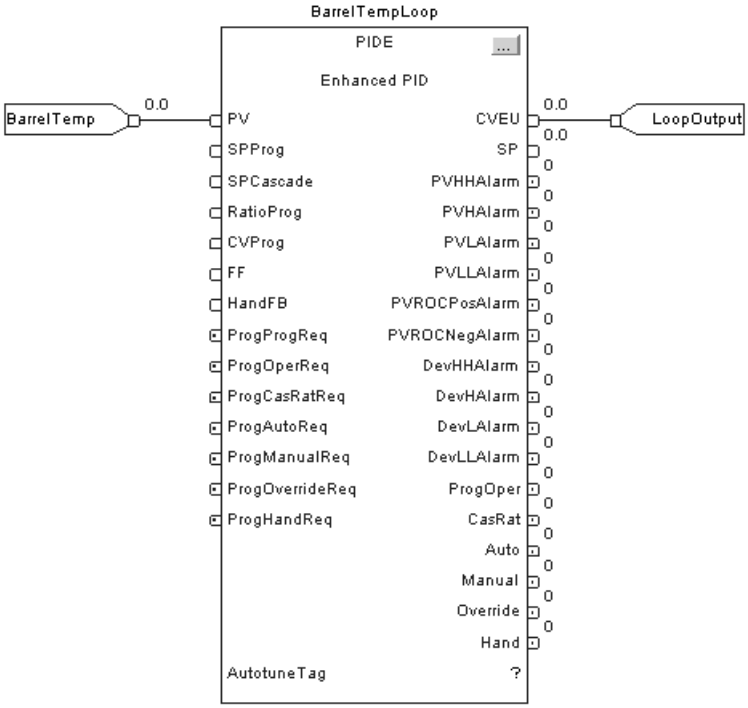
```
BarrelTempLoop.PV := BarrelTemp;
PIDE(BarrelTempLoop);
LoopOutput := BarrelTempLoop.CVEU;
```

将 SRTP 指令置于高速且优先权较高的任务中

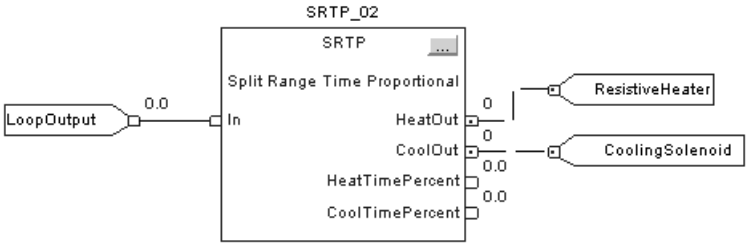
```
SRTP_02.In := LoopOutput;
SRTP(SRTP_02);
ResistiveHeater := SRTP_02.HeatOut;
CoolingSolenoid := SRTP_02.CoolOut;
```

功能块

将 PIDE 指令置于慢速且优先权较低的任务中



将 SRTP 指令置于高速且优先权较高的任务中



累加器 (TOT)

TOT 指令累加一定时间范围内的模拟输入值。

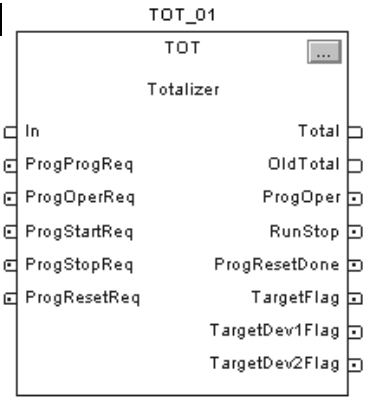
操作数：

```
TOT(TOT_tag);
```

结构文本

操作数：	类型：	格式：	说明：
TOT 标记	TOTALIZER	结构	TOT 结构

功能块



操作数：	类型：	格式：	说明：
TOT 标记	TOTALIZER	结构	TOT 结构

TOTALIZER 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果该位清零，则指令不执行且输出不更新。 如果该位已设置，则指令执行。 缺省状态为已设置。 **结构化文本：** 无效。指令执行。
In	REAL	输入到指令的模拟信号。 有效值 = 浮点数 缺省值 = 0.0
InFault	BOOL	In 故障指示。如果设置，则表示输入信号有错误，指令设置状态字中的相应位，不执行控制算法，且不更新 Total。 缺省状态为清零。

输入参数：	数据类型：	说明：
TimeBase	DINT	时间基输入。基于 In 工程单位的累加时间基。 **值：** **说明：** 0 秒 1 分钟 2 小时 3 天 例如，如果输入单位为 gal/min 则使用 TimeBase = 分钟。如果该值无效，则指令设置状态字中的相应位，且不更新 Total。 有关定时模式的更多信息，请参阅附录 “功能块属性”。 有效值 = 0 到 3 缺省值 = 0
Gain	REAL	累加值的系数。用户可利用 Gain 来转换累加值的单位。例如，使用 Gain 将 gal/min 转换为以料管为单位的累加值。 有效值 = 浮点数 缺省值 = 1.0
ResetValue	REAL	复位值输入。当 OperResetReq 或 ProgResetReq 从清零转换到设置时的 Total 复位值。 有效值 = 浮点数 缺省值 = 0.0
Target	REAL	累加 In 的目标值。 有效值 = 浮点数 缺省值 = 0.0
TargetDev1	REAL	Total 预定目标数值与 Target 之间的较大偏离量。该值反映了与 Target 之间的偏离量。 有效值 = 浮点数 缺省值 = 0.0
TargetDev2	REAL	Total 预定目标数值与 Target 之间的较小偏离量。该值反映了与 Target 之间的偏离量。 有效值 = 浮点数 缺省值 = 0.0
LowInCutoff	REAL	指令低输入截止输入。当 In 等于或低于 LowInCutoff 值时，累加停止。 有效值 = 浮点数 缺省值 = 0.0
ProgProgReq	BOOL	程序请求进入程序控制模式。设置以请求程序控制。如果 ProgOperReq 已设置，该请求被忽略。保持该位为已设置并将 ProgOperReq 清零，可锁定指令进入程序控制状态。 缺省状态为清零。
ProgOperReq	BOOL	程序请求进入操作员控制模式。设置以请求操作员控制。保持该位为已设置，可锁定指令进入操作员控制状态。 缺省状态为清零。
ProgStartReq	BOOL	程序启动请求输入。设置以请求开始累加。 缺省状态为清零。
ProgStopReq	BOOL	程序停止请求输入。设置以请求停止累加。 缺省状态为清零。
ProgResetReq	BOOL	程序复位请求输入。设置以请求 Total 复位为 ResetValue。 缺省状态为清零。

输入参数：	数据类型：	说明：
OperProgReq	BOOL	操作员请求进入程序控制模式。通过操作员界面设置来请求程序控制。 指令将该输入清零。 缺省状态为清零。
OperOperReq	BOOL	操作员请求进入操作员控制模式。通过操作员界面设置来请求操作员控制。 指令将该输入清零。 缺省状态为清零。
OperStartReq	BOOL	操作员启动请求输入。通过操作员界面设置该位请求开始累加。指令将该输入清零。 缺省状态为清零。
OperStopReq	BOOL	操作员停止请求输入。通过操作员界面设置该位请求停止累加。指令将该输入清零。 缺省状态为清零。
OperResetReq	BOOL	操作员复位请求输入。通过操作员界面设置该位请求累加复位。指令将该输入清零。 缺省状态为清零。
ProgValueReset	BOOL	复位程序控制值。设置时，每次指令执行后清零所有程序请求输入。 缺省状态为清零。
TimingMode	DINT	选择定时执行模式。 **值：** **说明：** 0 周期模式 1 过采样模式 2 实时采样模式 有关定时模式的更多信息，请参阅附录 “功能块属性”。 有效值 = 0 到 2 缺省值 = 0
OversampleDT	REAL	过采样模式下的运行时间。 有效值 = 0 到 4194.303 秒 缺省值 = 0
RTSTime	DINT	实时采样模式下的模块更新周期 有效值 = 1 到 32,767 毫秒 缺省值 = 1
RTSTimeStamp	DINT	实时采样模式下的模块时间戳值。 有效值 = 0 到 32,767 毫秒 缺省值 = 0

输出参数：	数据类型：	说明：
EnableOut	BOOL	输出使能。
Total	REAL	In 的累加值。该输出要设置算术状态标志。
OldTotal	REAL	复位前的累加值。用户可监控该值，以读取复位前的确切累加值。
ProgOper	BOOL	程序 / 操作员控制指示。处于程序控制状态时为已设置。处于操作员控制时清零。
RunStop	BOOL	指示累加器的操作状态。 TOT 指令运行时设置。 TOT 指令停止时清零。

输出参数：	数据类型：	说明：
ProgResetDone	BOOL	该位指示 TOT 指令已经完成了一次程序复位请求。当完成 ProgResetReq，指令复位时设置。用户监控该位来判断复位是否成功完成。当 ProgResetReq 清零时，该位清零。
TargetFlag	BOOL	Total 标志。当 Total ≥ Target 时设置。
TargetDev1Flag	BOOL	TargetDev1 标志。当 Total ≥ Target - TargetDev1 时设置。
TargetDev2Flag	BOOL	TargetDev2 标志。当 Total ≥ Target - TargetDev2 时设置。
LowInCutoffFlag	BOOL	指令低输入截止标志输出。当 In ≤ LowInCutoff 时设置。
DeltaT	REAL	更新之间的时间差值。通过控制算法计算过程输出时所用的时间 （单位为秒）。
Status	DINT	功能块的状态。
InstructFault (Status.0)	BOOL	指令检测到如下运行错误。该错误不是次要或主要的控制器错误。检查其他状态位以确定错误类型。
InFaulted (Status.1)	BOOL	In 值有错误。
TimeBaseInv (Status.2)	BOOL	TimeBase 值无效。
TimingModeInv (Status.27)	BOOL	TimingMode 值无效。
RTSMissed (Status.28)	BOOL	仅用于实时采样模式。ABS | DeltaT - RTSTime | > 1 （0.001 秒）时设置。
RTSTimeInv (Status.29)	BOOL	RTSTime 值无效。
RTSTimeStampInv (Status.30)	BOOL	RTSTimeStamp 值无效。
DeltaTInv (Status.31)	BOOL	DeltaT 值无效。如果在过采样定时模式下 OversampleDT 无效，会出现该情况。

说明： 该指令通常根据流量信号，累加在一段时间内添加的材料数量。

TOT 指令支持：

- 可选择秒、分钟、小时或天作为时间基。
- 可指定一个目标数值和最多两个预定目标数值。预定目标值通常用于切换到慢速进料速率。数字标志用来指示达到目标值或预定目标值。
- 支持低流量输入截止，它可用来消除在流量关闭时由于流量计的轻微刻度误差导致的负向累加。
- 操作员或程序控制的启动 / 停止 / 复位。
- 用户自定义复位值。
- 梯形数值积分用来提高精度。
- 以双精度数进行内部累加，提高精度。

监控 TOT 指令

TOT 指令具有操作员面板。有关更多信息，请参阅附录 “功能块面板控件”。

算术状态标志： Total 输出影响算术状态标志。

错误条件： 无

执行：

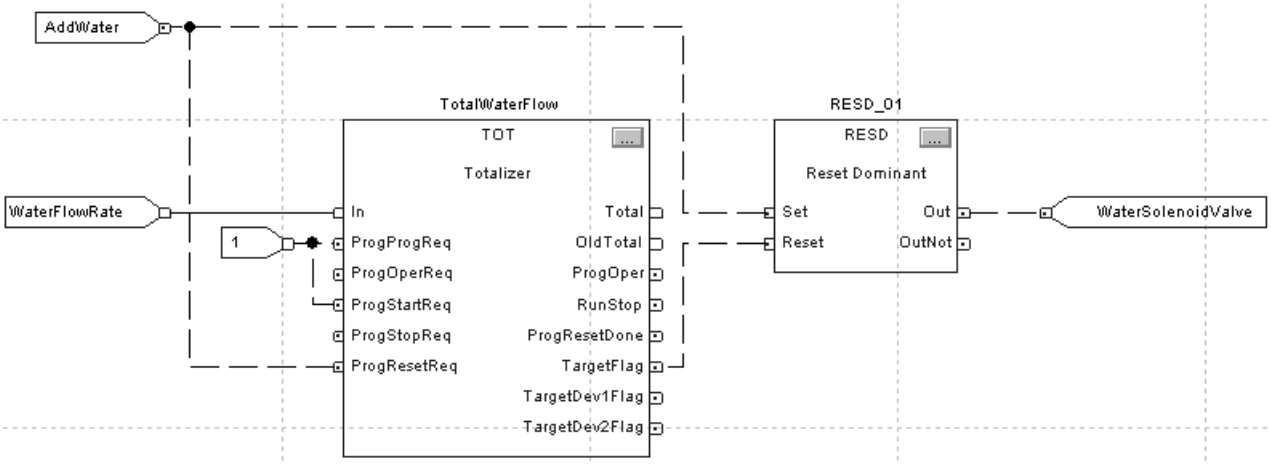
条件：	功能块动作：	结构化文本动作：
预扫描	无动作。	无动作。
指令首次扫描	所有操作员请求输入均清零。 如果 ProgValueReset 已设置，则所有程序请求输入均清零。	
指令首次执行	指令初始化内部参数。 Total = ResetValue。 OldTotal = 0.0。 ProgOper 清零。	
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。	不影响
EnableIn 已设置	指令执行。 EnableOut 已设置。	EnableIn 一直保持已设置状态。 指令执行。
后期扫描	无动作。	无动作。

示例： 在此例中， TOT 指令测量进入罐中的水的量，当水加到适当数量时关闭水流。当按下 AddWater 按钮时， TOT 指令复位并开始累加流入罐中的水量。一旦达到 Target 值， TOT 指令将设置 TargetFlag 输出，可使电磁阀关闭。在此例中，通过设置 ProgProgReq 和 ProgStartReq 输入可使 TOT 指令锁定在程序运行模式。此例使用了该模式，因为操作员始终无需直接控制 TOT 指令。

结构化文本

```
TotalWaterFlow.In := WaterFlowRate;
TotalWaterFlow.ProgProgReq := 1;
TotalWaterFlow.ProgStartReq := 1;
TotalWaterFlow.ProgResetReq := AddWater;
TOT(TotalWaterFlow);

RESD_01.Set := AddWater;
RESD_01.Reset := TotalWaterFlow.TargetFlag;
RESD(RESD_01);

WaterSolenoidValve := RESD_01.Out;
```

功能块



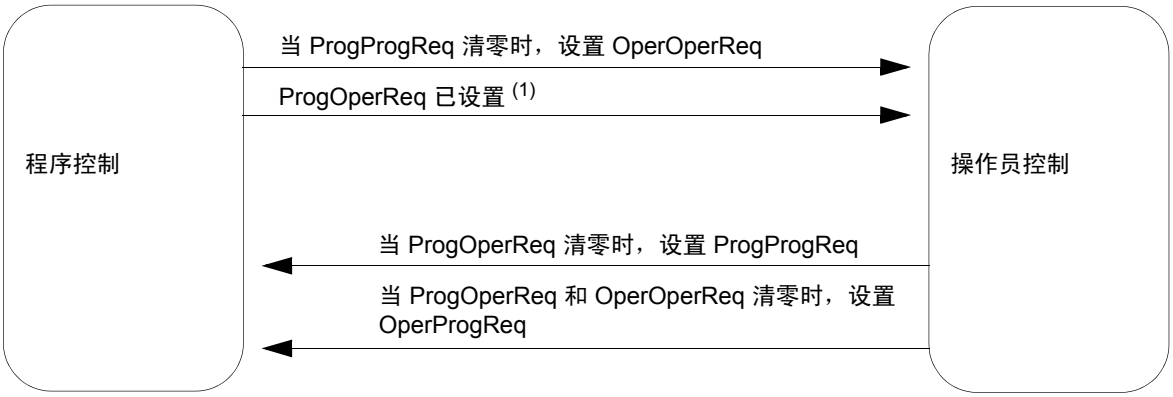
检查低输入截止

如果 (In ≤ LowInCutoff)，则指令设置 LowInCutoffFlag 并使 $In_{n-1} = 0.0$。否则，指令清零 LowInCutoffFlag。

当 LowInCutoffFlag 已设置时，操作模式得以确定，但累加停止。当 LowInCutoffFlag 清零时，累加继续扫描。

操作模式

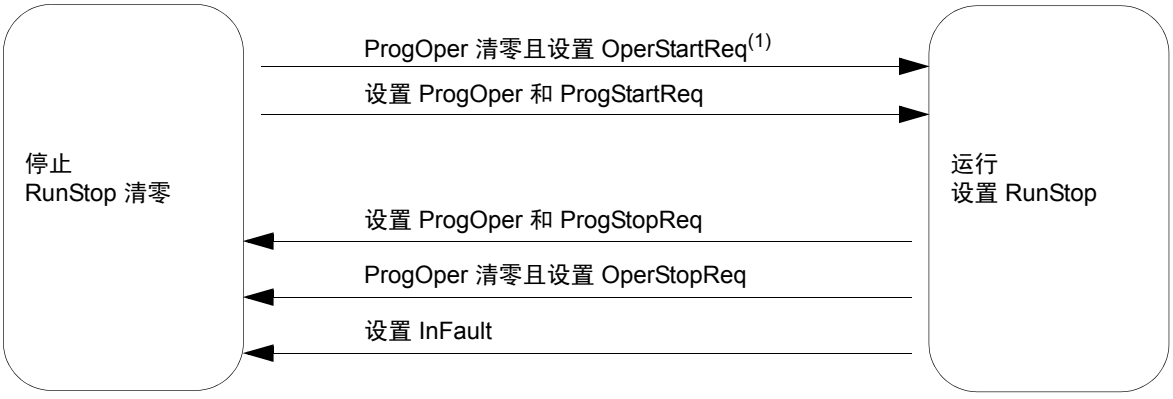
下图演示了 TOT 指令如何在程序控制和操作员控制模式之间切换。



(1) 当 ProgOperReq 为已设置时，指令仍处在操作员控制模式。

有关程序控制和操作员控制的更多信息，请参阅第 B-13 页。

下图说明了 TOT 指令如何在运行和停止模式之间切换。



(1) 停止请求优先权高于启动请求。

(2) 停止后运行首次扫描时，不计算累加值，但更新 In_{n-1}。在下一次扫描时，累加将重新启动。

每次扫描结束时，所有操作员请求输入都清零。如果设置 ProgValueReset，则每次扫描结束时，所有程序请求输入都清零。

复位 TOT 指令

当 ProgOper 已设置时，如果 ProgResetReq 转换为设置，则发生以下动作：

- OldTotal = Total
- Total = ResetValue
- 设置 ProgResetDone

如果 ProgResetReq 清零且设置 ProgResetDone，则 ProgResetDone 清零

当 ProgOper 已清零时，如果 OperResetReq 转换为设置，则发生以下动作：

- OldTotal = Total
- Total = ResetValue

计算累加值

当设置 RunStop 且 LowInCutoffFlag 清零时，可用以下公式来执行累加计算。

$$Total_n = Total_{n-1} + Gain \times \frac{DeltaT}{2 \times TimeBase} \times (In_n + In_{n-1})$$

其中 TimeBase 是：

值：	条件：
1	TimeBase = 0 （秒）
60	TimeBase = 1 （分钟）
3600	TimeBase = 2 （小时）
86400	TimeBase = 3 （天）

判断是否达到目标值

一旦计算出累加值，以下条件就用来判断是否达到目标值或预定目标值：

- 当 Total ≥ Target 时，设置 TargetFlag
- 当 Total ≥ (Target - TargetDev1) 时，设置 TargetDev1Flag
- 当 Total ≥ (Target - TargetDev2) 时，设置 TargetDev2Flag

驱动控制指令
（INTG、 PI、 PMUL、 SCRV、 SOC、 UPDN）

简介

可以使用以下驱动控制指令：

如要：	所用指令：	可用编程语言：	参见页：
执行积分操作。	积分器 (INTG)	结构化文本 功能块	2-2
执行 PI 算法。	比例 + 积分 (PI)	结构化文本 功能块	2-8
通过计算一次扫描与下一次扫描之间的输入变化，从解析器或编码器反馈模块等位置输入模块提供接口到数字系统。	脉冲乘法器 (PMUL)	结构化文本 功能块	2-20
利用附加的速度曲线光滑度执行斜坡函数。	S- 曲线 (SCRV)	结构化文本 功能块	2-28
使用增益项、一阶滞后校正项和二阶超前校正项。	二阶控制器 (SOC)	结构化文本 功能块	2-37
将两个输入相加、相减成累积值。	增 / 减累加器 (UPDN)	结构化文本 功能块	2-47

积分器 (INTG)

INTG 指令实施积分操作。该指令旨在扫描速率恒定的任务中执行。

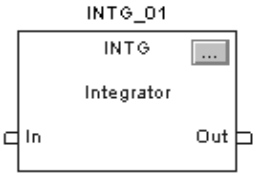
操作数：

结构化文本

```
INTG(INTG_tag);
```

操作数：	类型：	格式：	说明：
INTG 标记	INTEGRATOR	结构	INTG 结构

功能块



操作数：	类型：	格式：	说明：
INTG 标记	INTEGRATOR	结构	INTG 结构

INTEGRATOR 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果该位清零，则指令不执行且输出不更新。 如果该位已设置，则指令执行。 缺省状态为已设置。 **结构化文本：** 无效。指令执行。
In	REAL	输入到指令的模拟信号。 有效值 = 浮点数 缺省值 = 0.0
Initialize	BOOL	请求初始化控制算法。只要 Initialize 已置位，输出 = InitialValue。 有效值 = 浮点数 缺省值 = 0.0
InitialValue	REAL	指令的初始值。只要 Initialize 已置位，输出 = InitialValue。 有效值 = 浮点数 缺省值 = 0.0
IGain	REAL	积分增益乘数。如果 IGain < 0，则指令将设置 IGain = 0.0，设置状态字中相应位， Output 保持不变。 有效值 = 0.0 到最大正浮点数 缺省值 = 0.0
HighLimit	REAL	Out 的上限值。如果 HighLimit ≤ LowLimit，则指令设置 HighAlarm 和 LowAlarm 并设置状态字中相应位，且设置 Out = LowLimit。 有效值 = 浮点数 缺省值 = 最大正浮点数
LowLimit	REAL	Out 的下限值。如果 HighLimit ≤ LowLimit，则指令设置 HighAlarm 和 LowAlarm 并设置状态字中相应位，且设置 Out = LowLimit。 有效值 = 浮点数 缺省值 = 最大负浮点数

输入参数：	数据类型：	说明：
HoldHigh	BOOL	请求保持输出为高。设置后，不允许 Out 的值增加。 缺省状态为清零。
HoldLow	BOOL	请求保持输出为低。设置后，不允许 Out 的值减少。 缺省状态为清零。
TimingMode	DINT	选择定时执行模式。 **值：** **说明：** 0 周期模式 1 过采样模式 2 实时采样模式 有关定时模式的更多信息，请参阅附录 “功能块属性”。 有效值 = 0 到 2 缺省值 = 0
OversampleDT	REAL	过采样模式下的运行时间。 有效值 = 0 到 4194.303 秒 缺省值 = 0
RTSTime	DINT	实时采样模式下的模块更新周期 有效值 = 1 到 32,767 毫秒 缺省值 = 1
RTSTimeStamp	DINT	实时采样模式下的模块时间戳值。 有效值 = 0 到 32,767 毫秒 缺省值 = 0

输出参数：	数据类型：	说明：
EnableOut	BOOL	输出使能。
Out	REAL	算法计算的输出。该输出要设置算术状态标志。
HighAlarm	BOOL	上限报警指示。当 Out ≥ HighLimit 时，设置 HighAlarm 并将输出限制为 HighLimit 值。
LowAlarm	BOOL	下限报警指示。当 Out ≤ LowLimit 时，设置 LowAlarm 并将输出限制为 LowLimit 值。
DeltaT	REAL	更新之间的时间差值。通过控制算法计算过程输出时所用的时间 （单位为秒）。
Status	DINT	功能块的状态。
InstructFault (Status.0)	BOOL	指令检测到以下运行错误之一。该错误不是次要或主要的控制器错误。检查其他状态位以确定错误类型。
IGainInv (Status.1)	BOOL	IGain > 最大值或 IGain < 最小值。
HighLowLimsInv (Status.2)	BOOL	HighLimit ≤ LowLimit。
TimingModeInv (Status.27)	BOOL	TimingMode 值无效。 有关定时模式的更多信息，请参阅附录 “功能块属性”。
RTSMissed (Status.28)	BOOL	仅用于实时采样模式。 ABS | DeltaT - RTSTime | > 1 （0.001 秒）时设置。

输出参数：	数据类型：	说明：
RTSTimeInv (Status.29)	BOOL	RTSTime 值无效。
RTSTimeStampInv (Status.30)	BOOL	RTSTimeStamp 值无效。
DeltaTInv (Status.31)	BOOL	DeltaT 值无效。

说明： INTG 指令旨在用于扫描速率保持不变的任务中。

当 Initialize 清零并且 DeltaT > 0 时， INTG 指令执行该控制算法。

$$Out = IGain \times \frac{In + In_{n-1}}{2} \times DeltaT + Out_{n-1}$$

输出的计算值为无效、 NAN 或 ± INF 时，指令设置 Out 等于该无效值并设置算术溢出状态标志。内部参数不更新。在后续每次扫描中，都使用上一次输出有效时的扫描的内部参数来计算输出。

极限

INTG 指令根据 HoldHigh 和 HoldLow 输入的状态执行结果限制，阻止 Out 改变。如果设置了 HoldHigh 并且 Out > Out_{n-1}，则 Out = Out_{n-1}。如果设置了 HoldLow 并且 Out < Out_{n-1}，则 Out = Out_{n-1}。

INTG 指令还使用 HighLimit 和 LowLimit 执行输出限制。如果 Out ≥ HighLimit，则 Out = HighLimit 且设置 HighAlarm。如果 Out ≤ LowLimit，则 Out = LowLimit 且设置 LowAlarm。

算术状态标志： Out 输出影响算术状态标志。

错误条件： 无

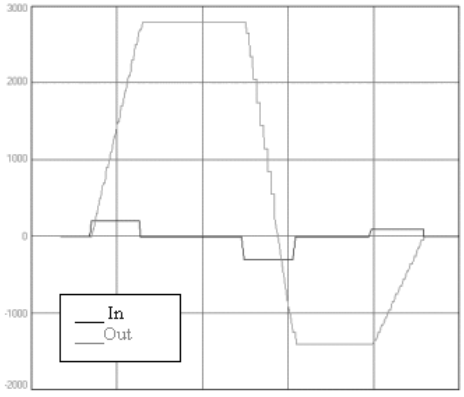
执行：

条件：	功能块动作：	结构化文本动作：
预扫描	无动作。	无动作。
指令首次扫描	将内部参数和 Out 设置为 0。 控制算法不执行。	将内部参数和 Out 设置为 0。 控制算法不执行。
指令首次执行	将内部参数和 Out 设置为 0。 控制算法不执行。	将内部参数和 Out 设置为 0。 控制算法不执行。
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。	na
EnableIn 已设置	指令执行。 EnableOut 已设置。	EnableIn 一直保持已设置状态。 指令执行。
后期扫描	无动作。	无动作。

实例： 许多应用中，为了消除或最小化所调节系统中的错误，闭环调节器设计中包括积分增益元素。正比例调节器一般不会将系统错误调到零。但是，使用比例和积分增益的调节器经过一段时间后，倾向于将错误信号调到零。 INTG 指令使用以下方程式计算输出。

$$Out = IGain \times \frac{In + In_{n-1}}{2} \times DeltaT + Out_{n-1}$$

该图中，对块的输入从 0 变为 +200 单位。这期间，块的输出积分为 2800 单位。随着 In 从 +200 单位变为 0 单位，Out 保持在 2800 单位。当 In 从 0 变为 -300 单位， Out 积分缓慢减为 -1400 单位，直到 In 变回 0。随着 In 从 0 变为 +100，Out 积分变回 0 ；当 Out 达到 0 时， In 同时被设置为 0。



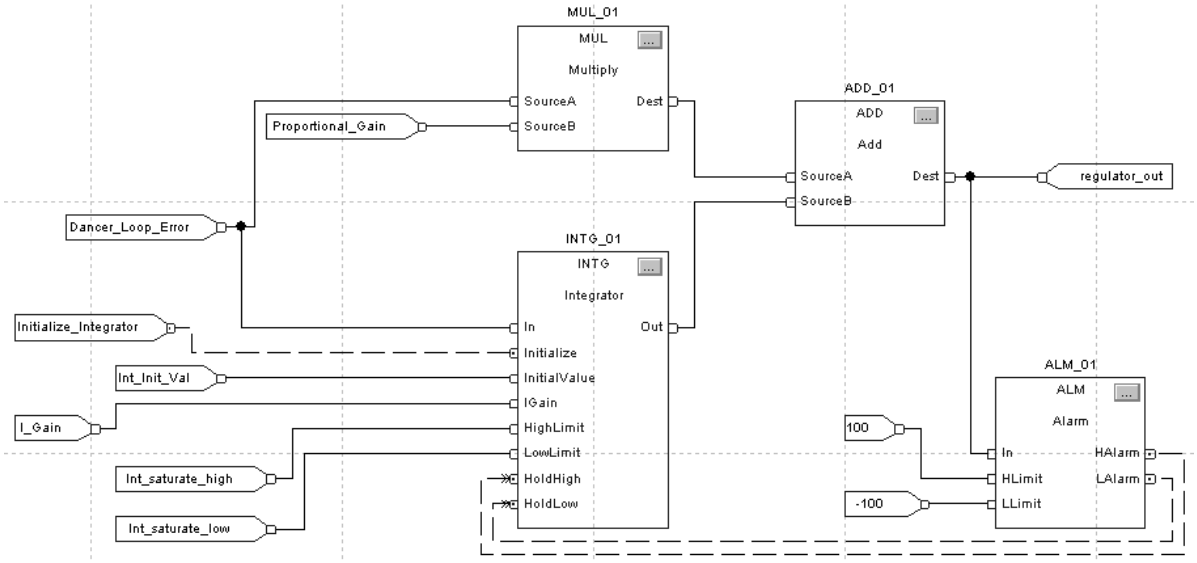
积分器的这个特征蜒在函数存在输入时沿特定方向连续调节，或在输入为零时保持在任一水平蜒导致使用积分增益的调节器经过一段时间后逐渐变为零错误。

以下实例说明了在应用中如何使用 INTG 指令。许多情形下，HighLimit 和 LowLimit 输入限制控制的总百分比，即积分增益元素在调节器总输出功能中所起的控制百分比。另一方面， HoldHigh 和 HoldLow 输入可以用来防止输出在正向或反向上发生进一步变化。本例中，如果调节器输出已饱和在 100%，则 HoldHigh 和 HoldLow 输入将防止 INTG 指令在已超出受控变量限值的方向上 “溢出”。

结构化文本

```
INTG_01.IN := Dancer_Loop_Error;
INTG_01.Initialize := Initialize_Integrator;
INTG_01.InitialValue := Int_Init_Val;
INTG_01.IGain := I_Gain;
INTG_01.HighLimit := Int_saturate_high;
INTG_01.LowLimit := Int_saturate_low;
INTG_01.HoldHigh := ALM_01.HAlarm;
INTG_01.HoldLow := ALM_01.LAlarm;
INTG(INTG_01);


regulator_out := (Dancer_Loop_Error*Proportional_Gain)
    + INTG_01.Out;


ALM_01.In := regulator_out;
ALM_01.HLimit := 100;
ALM_01.LLimit := -100;
ALM(ALM_01);
```

功能块


MUL_01
MUL
Multiply
SourceA
SourceB
Dest
Proportional_Gain
ADD_01
ADD
Add
SourceA
SourceB
Dest
regulator_out
Dancer_Loop_Error
INTG_01
INTG
Integrator
In
Initialize
InitialValue
IGain
HighLimit
LowLimit
HoldHigh
HoldLow
Out
Initialize_Integrator
Int_Init_Val
I_Gain
Int_saturate_high
Int_saturate_low
ALM_01
ALM
Alarm
In
HLimit
LLimit
HAlarm
LAlarm
100
-100

比例 + 积分 (PI)

PI 指令提供两种操作方法。第一种方法采用传统的 PI 算法，在输入信号 （错误）范围内，比例和积分增益保持不变。第二种方法采用非线性算法，在输入信号范围内，比例和积分增益可变。输入信号是程序设定点与反馈之间的偏差。

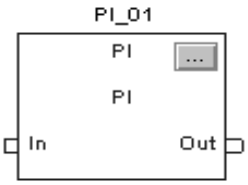
操作数：

结构文本

```
PI(PI_tag);
```

操作数：	类型：	格式：	说明：
PI 标记	PROP_INT	结构	PI 结构

功能块



操作数：	类型：	格式：	说明：
PI 标记	PROP_INT	结构	PI 结构

PROP_INT 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果该位清零，则指令不执行且输出不更新。 如果该位已设置，则指令执行。 缺省状态为已设置。 **结构化文本：** 无效。指令执行。
In	REAL	过程错误信号输入。这是设定点与反馈之间的差值。 有效值 = 浮点数 缺省值 = 0.0
Initialize	BOOL	指令初始化命令。设置时， Out 和内部积分器被设置为与 InitialValue 的值相等。 缺省状态为清零。
InitialValue	REAL	初始化输入值。 Initialize 已设置时， Out 和积分器被设置为 InitialValue 的值。使用 HighLimit 和 LowLimit 限制 InitialValue 值。 有效值 = 浮点数 缺省值 = 0
Kp	REAL	比例增益。这会影响比例和积分控制算法的计算值。如果无效，指令将把 Kp 限制在限值上，并相应地设置状态位。 有效值 = 任意浮点数 > 0.0 缺省值 = 最小正浮点数
Wld	REAL	超前频率 （单位为弧度 / 秒）。这会影响积分控制算法的计算值。如果无效，指令将把 Wld 限制在限值上，并相应地设置状态位。 有效值 = 参阅下面有关有效范围的说明 缺省值 = 0.0

输入参数：	数据类型：	说明：
HighLimit	REAL	上限值。即输出的最大值。如果 HighLimit ≤ LowLimit，则指令设置 HighAlarm 和 LowAlarm 并设置状态字中相应位，且设置 Out = LowLimit。 有效值 = LowLimit < HighLimit ≤ 最大正浮点数 缺省值 = 最大正浮点数
LowLimit	REAL	下限值。即输出的最小值。如果 HighLimit ≤ LowLimit，则指令设置 HighAlarm 和 LowAlarm 并设置状态字中相应位，且设置 Out = LowLimit。 有效值 = 最大负浮点数 ≤ LowLimit < HighLimit 缺省值 = 最大负浮点数
HoldHigh	BOOL	上限保持命令。设置时，不允许增加内部积分器的值。 缺省状态为清零。
HoldLow	BOOL	下限保持命令。设置时，不允许减少内部积分器的值。 缺省状态为清零。
ShapeKpPlus	REAL	正 Kp 整形增益乘数。当 In ≥ 0 时使用。如果无效，指令将把 ShapeKpPlus 限制在限值上，并相应地设置状态位。当 NonLinearMode 被清零时忽略该参数。 有效值 = 0.1 到 10.0 缺省值 = 1.0
ShapeKpMinus	REAL	负 Kp 整形增益乘数。当 In < 0 时使用。如果无效，指令将把 ShapeKpMinus 限制在限值上，并相应地状态位。当 NonLinearMode 被清零时忽略该参数。 有效值 = 0.1 到 10.0 缺省值 = 1.0
KpInRange	REAL	比例增益整形范围。定义 In （错误）的范围，在该范围上，比例增益作为 | In | / KpInRange 比值的函数而增加或减少。当 | In | > KpInRange 时，指令利用所输入的 Kp 整形增益乘以 (In - KpInRange) 来计算比例错误变化。如果无效，指令将把 KpInRange 限制在限值上，并相应地设置状态位。当 NonLinearMode 被清零时忽略该参数。 有效值 = 任意浮点数 > 0.0 缺省值 = 最大正浮点数
ShapeWldPlus	REAL	正 Wld 整形增益乘数。当 In ≥ 0 时使用。如果无效，指令将把 ShapeWldPlus 限制在限值上，并相应地设置状态位。当 NonLinearMode 被清零时忽略该参数。 有效值 = 0.0 到 10.0 缺省值 = 1.0
ShapeWldMinus	REAL	负 Wld 整形增益乘数。当 In < 0 时使用。如果无效，指令将把 ShapeWldMinus 限制在限值上，并相应地设置状态位。当 NonLinearMode 被清零时不使用该参数。 有效值 = 0.0 到 10.0 缺省值 = 1.0
WldInRange	REAL	积分增益整形范围。定义 In （错误）的范围，在该范围上，积分增益作为 | In | / WldInRange 比值的函数而增加或减少。当 |In| > WldInRange 时，计算积分错误时指令将把 In 限制为 WldInRange。如果无效，指令将把 WldInRange 限制在限值上，并相应地设置状态位。当 NonLinearMode 被清零时忽略该参数。 有效值 = 任意浮点数 > 0.0 缺省值 = 最大正浮点数

输入参数：	数据类型：	说明：
NonLinearMode	BOOL	启用非线性增益模式。如果设置，指令将使用 ParabolicLinear 选择的非线性增益模式计算实际的比例和积分增益。如果清零，指令将禁用非线性增益模式，使用 Kp 和 Wld 值作为比例和积分增益。 缺省状态为清零。
ParabolicLinear	BOOL	选择非线性增益模式。增益模式有线性或抛物线式。如果设置，指令将使用抛物线增益方法 $y = a * x^2 + b$ 计算实际比例和积分增益。如果清零，指令将使用线性增益方法 $y = a * x + b$。 缺省状态为清零。
TimingMode	DINT	选择定时执行模式。 **值：** **说明：** 0 周期模式 1 过采样模式 2 实时采样模式 有关定时模式的更多信息，请参阅附录 “功能块属性”。 有效值 = 0 到 2 缺省值 = 0
OversampleDT	REAL	过采样模式下的运行时间。 有效值 = 0 到 4194.303 秒 缺省值 = 0
RTSTime	DINT	实时采样模式下的模块更新周期 有效值 = 1 到 32,767 毫秒 缺省值 = 1
RTSTimeStamp	DINT	实时采样模式下的模块时间戳值。 有效值 = 0 到 32,767 毫秒 缺省值 = 0

输出参数：	数据类型：	说明：
EnableOut	BOOL	输出使能。
Out	REAL	PI 算法的计算输出。该输出要设置算术状态标志。
HighAlarm	BOOL	上限报警指示。当 Out 的计算值 ≥ HighLimit 时设置，并且输出和积分器被限制在 HighLimit。
LowAlarm	BOOL	下限报警指示。当 Out 的计算值 ≤ LowLimit 时设置，并且输出和积分器被限制在 LowLimit。
DeltaT	REAL	更新之间的时间差值。通过控制算法计算过程输出时所用的时间 （单位为秒）。
Status	DINT	功能块的状态。
InstructFault (Status.0)	BOOL	指令检测到以下运行错误之一。该错误不是次要或主要的控制器错误。检查其他状态位以确定错误类型。
KpInv (Status.1)	BOOL	Kp < 最小值或 Kp > 最大值。
WldInv (Status.2)	BOOL	Wld < 最小值或 Wld > 最大值。
HighLowLimsInv (Status.3)	BOOL	HighLimit ≤ LowLimit。

输出参数：	数据类型：	说明：
ShapeKpPlusInv (Status.4)	BOOL	ShapeKpPlus < 最小值或 ShapeKpPlus > 最大值。
ShapeKpMinusInv (Status.5)	BOOL	ShapeKpMinus < 最小值或 ShapeKpMinus > 最大值。
KpInRangeInv (Status.6)	BOOL	KpInRange < 最小值或 KpInRange > 最大值。
ShapeWldPlusInv (Status.7)	BOOL	ShapeWldPlus < 最小值或 ShapeWldPlus > 最大值。
ShapeWldMinusInv (Status.8)	BOOL	ShapeWldMinus < 最小值或 ShapeWldMinus > 最大值。
WldInRangeInv (Status.9)	BOOL	WldInRange < 最小值或 WldInRange > 最大值。
TimingModeInv (Status.27)	BOOL	定时模式无效。 有关定时模式的更多信息，请参阅附录 “功能块属性”。
RTSMissed (Status.28)	BOOL	仅用于实时采样模式。 ABS | DeltaT - RTSTime | > 1 （0.001 秒）时设置。
RTSTimeInv (Status.29)	BOOL	RTSTime 值无效。
RTSTimeStampInv (Status.30)	BOOL	RTSTimeStamp 值无效。
DeltaT (Status.31)	BOOL	DeltaT 值无效。

说明： PI 指令使用 PI 算法的位置形式。这意味着增益项是直接应用在输入信号上，而不是应用在输入信号变化上。 PI 指令旨在扫描速率保持不变的任务中执行。

在非线性算法中，比例和积分增益随着输入信号的振幅改变而改变。PI 指令支持两种非线性增益模式：线性和抛物线式。在线性算法中，增益随着输入的振幅改变而线性改变。在抛物线算法中，增益随着输入的振幅改变而呈抛物线形改变。

PI 指令利用以下方程式计算 Out：

$$Kp \times \frac{s + Wld}{s}$$

输出的计算值为无效、 NAN 或 ± INF 时，指令设置 Out 等于该无效值并设置算术溢出状态标志。内部参数不更新。在后续每次扫描中，都使用上一次输出有效时的扫描的内部参数来计算输出。

线性模式下的操作

线性模式下，非线性增益模式被禁用。 Kp 和 Wld 值是指令所用的比例和积分增益。指令利用以下方程式计算 Out 的值：

值：	方程式：
ITerm	$Kp \times Wld \times \frac{WldInput + WldInput_{n-1}}{2} \times DeltaT + ITerm_{n-1}$ DeltaT 的单位为秒
PTerm	$Kp \times In$
Out	$ITerm + PTerm$

Wld 的限制条件为：

$$\text{LowLimit} > 0.0$$

$$\text{HighLimit} = \frac{0.7\pi}{DeltaT}$$

$$\text{WldInput} = \text{In}$$

非线性模式下的操作

非线性模式下，指令使用 ParabolicLinear 选择的非线性增益模式计算实际比例和积分增益。

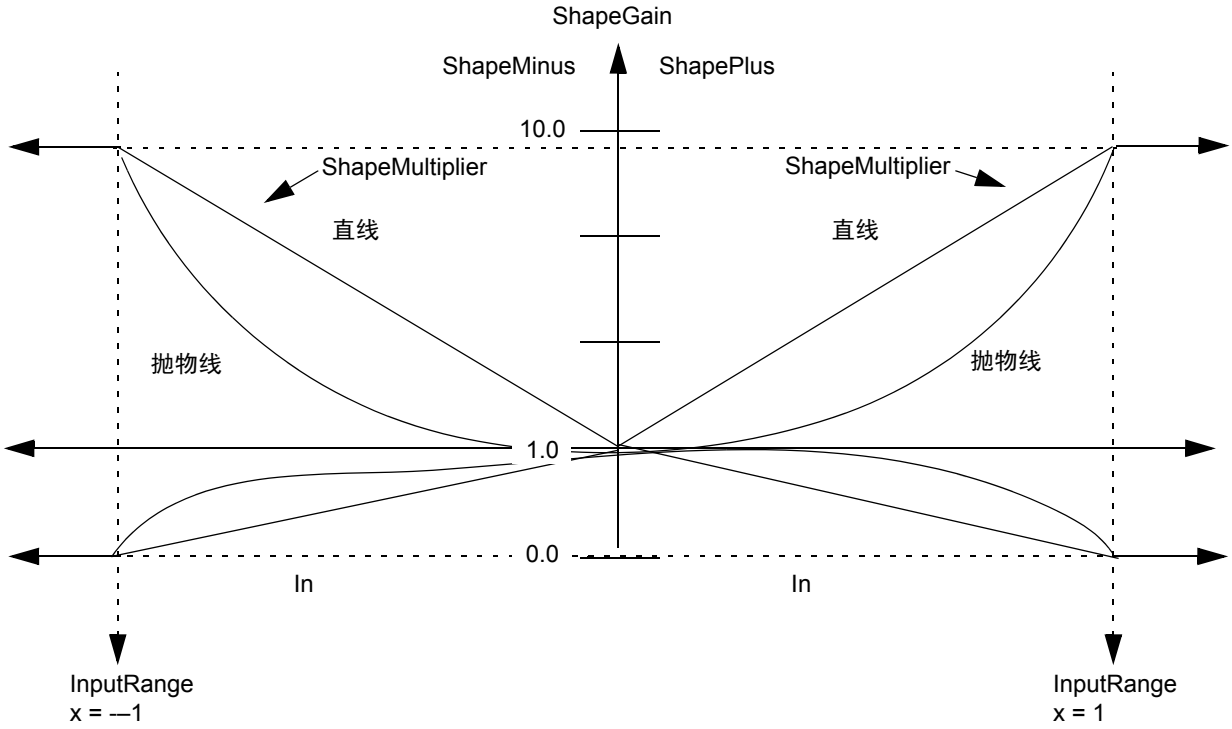
当 In = 0 时， Kp 和 Wld 所指定的增益乘以 1.0。单独的比例和积分算法随着错误振幅的改变而增加或减少比例或积分增益。这些算法利用输入范围和整形增益参数来计算实际的比例和积分增益。输入范围定义增益整形的 In （即错误）范围。输入范围由 KpInRange 和 WldInRange 这两个参数设置。整形增益定义整形增益参数所控制的象限的增益乘数。整形增益由 ShapeKpPlus、 ShapeKpMinus、 ShapeWldPlus 和 ShapeWldMinus 设置。

ParabolicLinear 输入选择非线性增益模式。如果 ParabolicLinear 清零，则选择线性模式。如果设置 ParabolicLinear，则选择抛物线模式。

如要配置特殊的整形增益曲线，输入整形增益 0.0–10.0 可得到积分整形，输入整形增益 0.1–10.0 可得到比例整形，以及输入整形适用的输入范围。 Kp 和 Wld 乘以所计算的 ShapeMultiplier 得到实际的比例和积分增益。输入整形增益 1.0 将禁用计算该象限的比例或积分增益的非线性算法。

如果 In （错误）的振幅大于 InRange，则 ShapeMultiplier 等于当 | In | 等于 InRange 时计算的值。

下图显示代表抛物线和线性增益方程式的最大和最小增益曲线。



指令利用以下方程式计算 Out 的值：

值：	方程式：
Kp 整形增益乘数	如果 In ≥ 0，则： $KpShapeGain = ShapeKpPlus$ $KpRange = KpInRange$ 否则： $KpShapeGain = ShapeKpMinus$ $KpRange = -KpInRange$
Kp 输入比值	如果 |In| ≤ KpInRange： $KpInputRatio = \|In\| \times \frac{1}{KpInRange}$ 否则： $KpInputRatio = 1$
Kp 比值	如果非抛物线模式： $KpRatio = KpInputRatio \times 0.5$ 如果是抛物线模式： $KpRatio = KpInputRatio^2 \times 0.333$
Kps 整形增益	$Kps = Kp \times (((KpShapeGain - 1) \times KpRatio) + 1)$
比例输出	如果 |In| ≤ KpInRange： $PTerm = Kps \times In$ 否则限制增益： $PTerm = Kps \times KpRange + (In - KpRange) \times KpShapeGain$
Wld 整形增益	如果 In ≥ 0，则： $WldShapeGain = ShapeWldPlus$ 否则： $WldShapeGain = ShapeWldMinus$
Wld 输入	如果 In > WldRange，则： $WldInput = WldInRange$ 否则如果 In < 蠲 ldInRange，则： $WldInput = -WldInRange$ 否则： $WldInput = In$

值：	方程式：
Wld 输入比值	如果 |In| ≤ WldInRange： $WldInputRange = \|In\| \times \frac{1}{WldInRange}$ 否则： $WldInputRange = 1$
Wld 比值	如果非抛物线模式： $WldRatio = WldInputRatio$ 如果是抛物线模式： $WldRatio = WldInputRatio^2$
Wlds 整形增益	$Wlds = Wld \times (((WldShapeGain - 1) \times WldRatio) + 1)$
Wlds 限值	$LowLimit > 0$ $HighLimit = \frac{0.7\pi}{DeltaT}$
积分输出	$ITerm = Kps \times Wlds \times \frac{(WldInput + WldInput_{n-1})}{2} \times DeltaT + ITerm_{n-1}$
Output	$Out = PTerm + ITerm$

极限

指令根据保持输入的状态阻止 ITerm 超出。

条件：	动作：
如果设置了 HoldHigh **并且** $ITerm > ITerm_{n-1}$	$ITerm = ITerm_{n-1}$
如果设置了 HoldLow **并且** $ITerm < ITerm_{n-1}$	$ITerm = ITerm_{n-1}$

指令还根据 HighLimit 和 LowLimit 的值阻止积分器超出。

条件：	动作：
Integrator > HighLimit	Integrator = HighLimit
Integrator < LowLimit	Integrator = LowLimit

指令根据 HighLimit 和 LowLimit 的值限制 Out 的值。

条件：	动作：
HighLimit ≤ LowLimit	Out = LowLimit ITerm = LowLimit HighLowLimsInv 已设置 HighAlarm 已设置 LowAlarm 已设置 WldInput = 0
Out ≥ HighLimit	Out = HighLimit ITerm = $ITerm_{n-1}$ HighAlarm 已设置
ITerm > HighLimit	ITerm = HighLimit
Out ≤ LowLimit	Out = LowLimit ITerm = $ITerm_{n-1}$ LowAlarm 已设置
ITerm < LowLimit	ITerm = LowLimit

算术状态标志： Out 输出要设置算术状态标志。

错误条件： 无

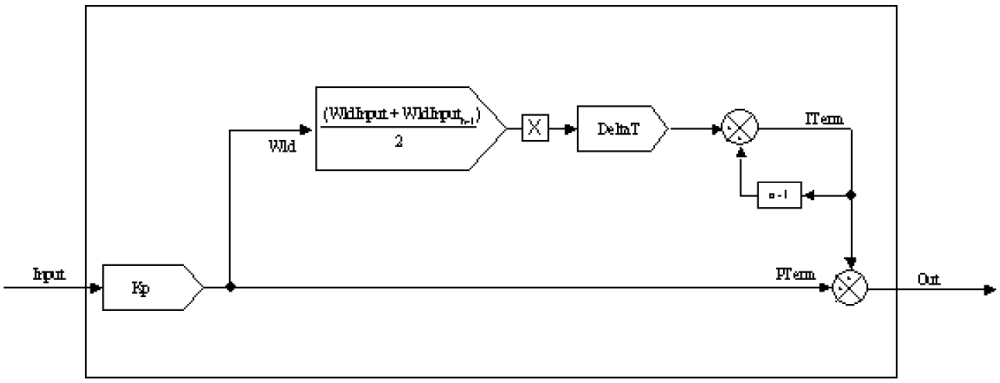
执行：

条件：	功能块动作：	结构化文本动作：
预扫描	无动作。	无动作。
指令首次扫描	Out = 0 控制算法不执行。	Out = 0 控制算法不执行。
指令首次执行	Out = 0 控制算法不执行。	Out = 0 控制算法不执行。
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。	na
EnableIn 已设置	指令执行。 EnableOut 已设置。	EnableIn 一直保持已设置状态。 指令执行。
后期扫描	无动作。	无动作。

实例： PI 指令是带有比例和积分增益元素的调节指令。积分增益元素由用户设置，单位为弧度 / 秒；它设置 PI 调节器的基本频率响应。比例增益设置块的总增益，包括块的比例和积分增益。

除初始化和保持 / 限制功能外，下图显示了 PI 块在线性模式下的基本调节回路。

PI 指令：线性模式

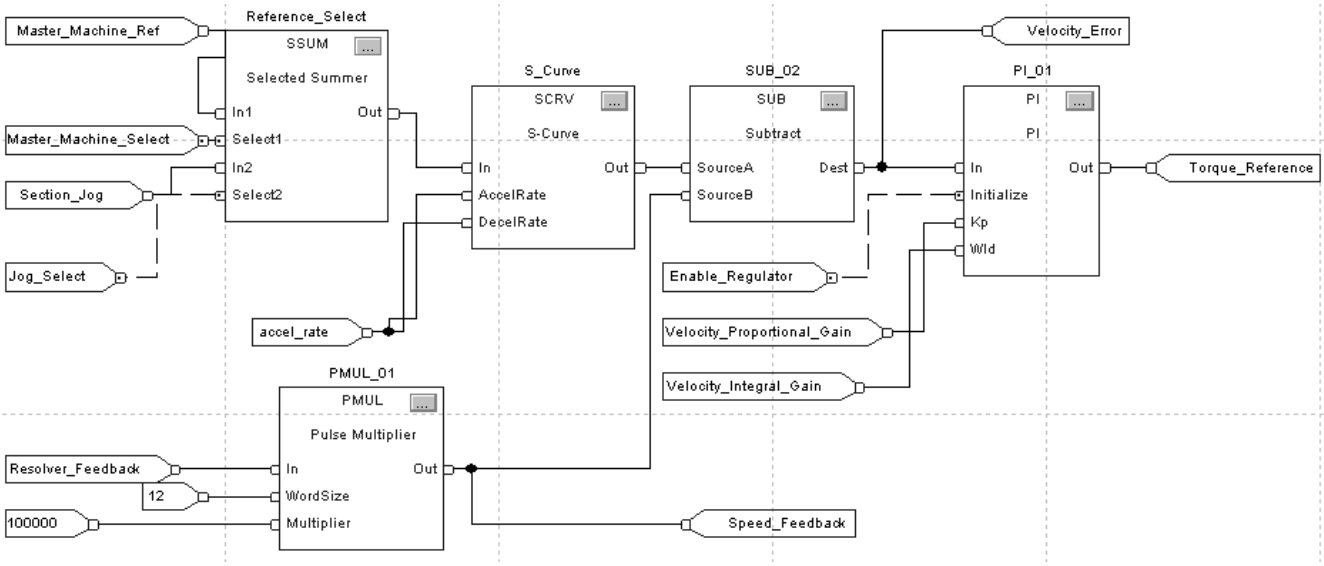


以下示例说明了作为速度调节器的 PI 指令。本例中，速度错误是由系统的速度参考 （通过 SCRV 指令）减去速度反馈信号 （参阅 PMUL 指令实例）产生的。速度错误被直接加入 PI 指令中，它根据上图所示的函数作用在该信号上。

结构化文本

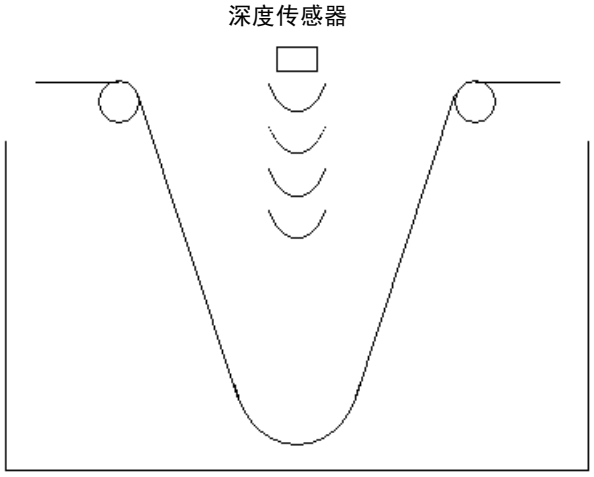
```
Reference_Select.In1 := Master_Machine_Ref;
Reference_Select.Select1 := Master_Machine_Select;
Reference_Select.In2 := Section_Jog;
Reference_Select.Select2 := Jog_Select;
SSUM(Reference_Select);

S_Curve.In := Reference_Select.Out;
S_Curve.AccelRate := accel_rate;
S_Curve.DecelRate := accel_rate;
SCRV(S_Curve);

PMUL_01.In := Resolver_Feedback;
PMUL_01.WordSize := 12;
PMUL_01.Multiplier := 100000;
PMUL(PMUL_01);

Speed_Feedback := PMUL_01.Out;
Velocity_Error := S_Curve.Out - Speed_Feedback;

PI_01.In := Velocity_Error;
PI_01.Initialize := Enable_Regulator;
PI_01.Kp := Velocity_Proportional_Gain;
PI_01.Wld := Velocity_Integral_Gain;
PI(PI_01);

Torque_Reference := PI_01.Out;
```

功能块



非线性模式下， PI 指令的增益可以整形为输入到块中的错误的函数。该函数允许进行适应性增益控制，可以用来模拟调节器机制，以更精确地匹配所调节的过程。可以使用该函数的一个实例是在悬链控制应用中。该应用中，就实际存储的材料量而言，返回自循环坑中的传感器的反馈可能不反映线性信号。这里， PI 调节器的比例增益可以被整形成更精确地模拟该过程，而不使用可能会不断 “上紧” 和 “松弛” 的积分元素。

PI 指令：非线性模式



脉冲乘法器 (PMUL)

PMUL 指令通过计算一次扫描到下一次扫描的输入变化，从位置输入模块，如解析器或编码器反馈模块等提供一个接口到数字系统。通过选择特定的字大小，可以配置 PMUL 指令以连续和线性模式对翻转边界求微分。

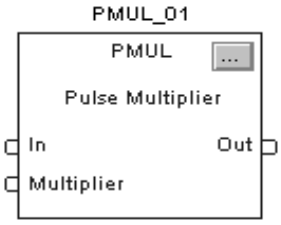
操作数：

结构化文本

```
PMUL(PMUL_tag);
```

操作数：	类型：	格式：	说明：
PMUL 标记	PULSE_MULTIPLIER	结构	PMUL 结构

功能块



操作数：	类型：	格式：	说明：
PMUL 标记	PULSE_MULTIPLIER	结构	PMUL 结构

PULSE_MULTIPLIER 结构

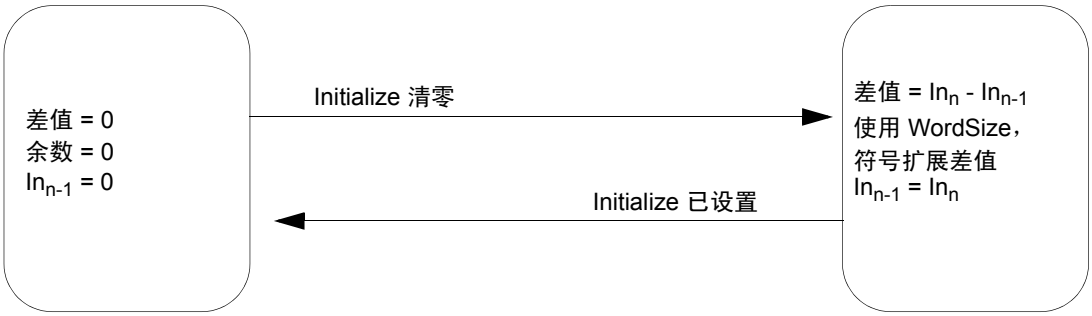
输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果该位清零，则指令不执行且输出不更新。 如果该位已设置，则指令执行。 缺省状态为已设置。 **结构化文本：** 无效。指令执行。
In	DINT	输入到指令的模拟信号。 有效值 = 长整数 缺省值 = 0
Initialize	BOOL	初始化输入。如果设置， Out 将保持在 0.0，所有内部寄存器都被设为 0。在设置到清零转变中， In_{n-1} = InitialValue （对 Absolute 模式无效）。如果清零，指令将正常执行。如果 WordSize 无效，指令将忽略 Initialize。 缺省状态为清零。
InitialValue	DINT	初始化输入值。在 Initialize 的设置到清零转变中， In_{n-1} = InitialValue 有效值 = 长整数 缺省值 = 0

输入参数：	数据类型：	说明：
Mode	BOOL	模式输入。设置将启用 Relative 模式。清零将启用 Absolute 模式。 缺省状态为已设置。
WordSize	DINT	字大小，单位为位。指定在 Relative 模式下计算 (In_n - In_{n-1}) 时使用的位数。 Absolute 模式下不使用 WordSize。当 In 变化大于 1/2 x $2^{(Wordsize - 1)}$ 时，Out 改变符号。如果 WordSize 无效， Out 将保持，指令设置状态字相应位。 有效值 = 2 到 32 缺省值 = 14
Multiplier	DINT	乘数。将该值除以 100,000 来控制 In 与 Out 的比值。如果无效，指令将限制该值，设置状态字相应位。 有效值 = -1,000,000 到 1,000,000 缺省值 = 100,000

输出参数：	数据类型：	说明：
EnableOut	BOOL	输出使能。
Out	REAL	指令的 Out。如果 Out 计算溢出， Out 将被强制为 $^{+}/_{-}\infty$，并设置状态字相应位。该输出要设置算术状态标志。
Status	DINT	功能块的状态。
InstructFault (Status.0)	BOOL	指令检测到以下运行错误之一。该错误不是次要或主要的控制器错误。检查其他状态位以确定错误类型。
WordSizeInv (Status.1)	BOOL	WordSize 值无效。
OutOverflow (Status.2)	BOOL	内部输出计算溢出。
LostPrecision (Status.3)	BOOL	Out < -2^{24} 或 Out > 2^{24}。当指令将 Out 从整数转变成实数时，如果结果大于 |2^{24}|，数据将丢失，因为 REAL 数据类型的最大值为 2^{24}。
MultiplierInv (Status.4)	BOOL	乘数值无效。

说明： PMUL 指令在 Relative 或 Absolute 模式下操作。

在 Relative 模式下，指令的输出是每次扫描输入的微分乘以 (Multiplier/100,000)。Relative 模式下，指令保存一次扫描中除法操作后的任何余数，并在下一次扫描期间加上。这样，在操作过程中，位置信息就不会丢失。



在 Absolute 模式下，指令可以缩放输入，如位置等，从一次扫描到下一次扫描的任何信息都不会丢失。

计算输出和余数

PMUL 指令使用以下方程式计算 Relative 或 Absolute 模式下的 Out：

Ans = ((DiffInput x Multiplier) + INT_Remainder)

INT_Out = Ans / 100,000

INT_Remainder= Ans - (INT_Out * 100,000)

Out = INT_Out

算术状态标志： Out 输出影响算术状态标志。

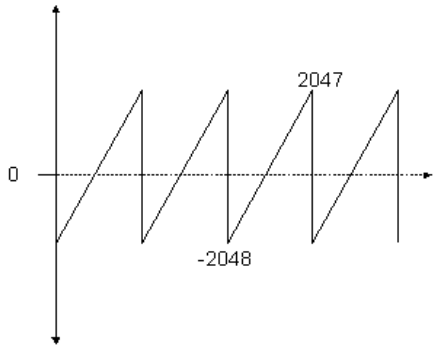
错误条件： 无

执行：

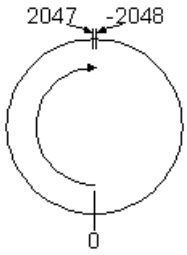
条件：	功能块动作：	结构化文本动作：
预扫描	无动作。	无动作。
指令首次扫描	In_{n-1} = In 余数 = 0	In_{n-1} = In 余数 = 0
指令首次执行	In_{n-1} = In 余数 = 0	In_{n-1} = In 余数 = 0
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。	na
EnableIn 已设置	指令执行。 EnableOut 已设置。	EnableIn 一直保持已设置状态。 指令执行。
后期扫描	无动作。	无动作。

例 1： PMUL 指令的最常见用途是在 Relative 操作模式中。该模式下，PMUL 指令可以实现多个目的。首先，在 Relative 模式下， PMUL 指令对它从每次扫描中在其输入处接收到的信息计算差值。收到数据后，指令输出一次扫描与下一次扫描的输入差值。这意味着，如果在第 “n”次扫描时 In = 500，在第 “n+1”次扫描时 In = 600，则第 “n+1”次扫描时的 Out = 100。

第二，在这种操作模式下， PMUL 指令还能补偿产生自反馈模块的二进制数据的 “翻转”值。例如，解析器反馈模块可具有 12 位分辨率，表示为带符号的二进制值，其范围为 –2048 到 2047。就来自反馈模块的原始数据而言，反馈设备的旋转可表示为下图所示：



本例中，随着反馈数据值从 2047 变到 –2048，有效位置变化相当于位置跳跃 4095 次。但实际上，对于解析器反馈设备的旋转而言，该位置变化仅为 4096 分之一。了解从反馈模块输入的数据的真实字大小后，PMUL 指令以旋转模式对待数据，如下图所示：



知道输入到该块的字大小后，当对该块的输入从 2047 变到 –2048 时，PMUL 指令求得输出为 1 次，而不是数学上计算的 4095。

应用该块时，应特别注意，如果要正确计算旋转方向差值，则从一次扫描到下一次扫描的反馈数据变化不应超过 ½ 字大小。上例中，如果反馈设备沿顺时针方向移动，以致在第 ‘A’次扫描时读取 0，而后在第 ‘B’次扫描时读取 –2000，则实际位置变化相当于沿顺时针方向的 +2096 次。但是，由于这两个值超过 ½ 字大小 （或超过物理设备旋转的 ½ ），因此 PMUL 指令的计算是，反馈设备沿反方向旋转，返回值 –2000，而不是 +2096。

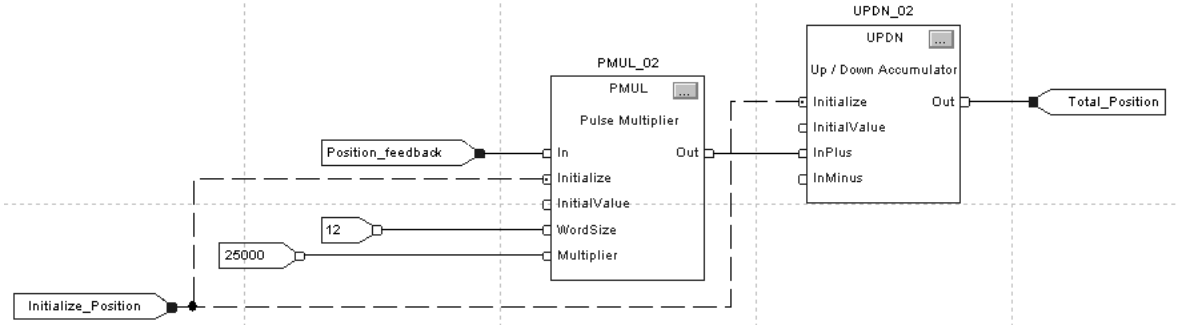
脉冲乘法器块的第三个特性是，它会保留作为 Multiplier/100,000 缩放因子的结果而存在的任何余数的从一次扫描到下一次扫描的分数元素。每次块完成执行后，前一次扫描的余数被加到当前值的总和中，使得全部次数或 “脉冲” 最终都被考虑到，系统不会丢失任何数据。块的输出 Out 始终产生完整的浮点数。

结构化文本

```
PMUL_02.In := Position_feedback;
PMUL_02.Initalize := Initialize_Position;
PMUL_02.WordSize := 12;
PMUL_02.Multiplier := 25000;
PMUL(PMUL_02);
```

```
UPDN_02.Initialize := Initialize_Position;
UPDN_02.InPlus := PMUL_02.Out;
UPDN(UPDN_02);

Total_Position := UPDN_02.Out;
```

功能块



假定 Initial_Position = 0 且 Multiplier = 25000 => (25,000/100,000)：

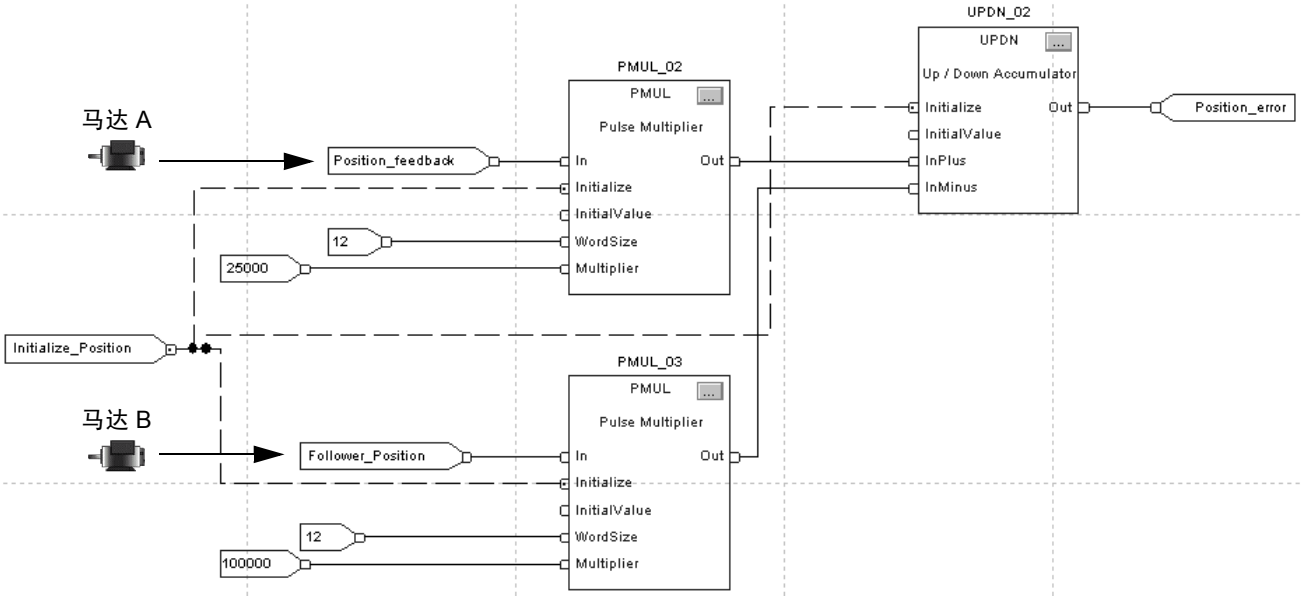
扫描：	Position_Feedback：	PMUL_02.Out：	Total_Position：
n	0	0	0
n + 1	1	0	0
n + 2	2	0	0
n + 3	3	0	0
n + 4	4	1	1
n + 5	5	0	1

例 2： 在该电子中间轴应用中，马达 A 的反馈充当主参考，马达 B 需遵循该主参考。马达 A 的反馈称为 “Position_feedback”。马达 B 的反馈称为 “Follower_Position”。由于两个指令的乘数为 1/4 的倍率，因此马达 A 每转四次，马达 B 需旋转一次，以便维持 UPDN 累加器中的累积零值。 UPDN 指令输出的任何非零值均被视为 Position_error，可以被调节并驱回马达 B，从而维持两个马达之间的相位锁定。

结构化文本

```
PMUL_02.In := Position_feedback;
PMUL_02.Initalize := Initialize_Position;
PMUL_02.WordSize := 12;
PMUL_02.Multiplier := 25000;
PMUL(PMUL_02);

PMUL_03.In := Follower_Position;
PMUL_03.Initalize := Initialize_Position;
PMUL_03.WordSize := 12;
PMUL_03.Multiplier := 100000;
PMUL(PMUL_03);

UPDN_02.Initialize := Initialize_Position;
UPDN_02.InPlus := PMUL_02.Out;
UPDN_02.InMinus := PMUL_03.Out;
UPDN(UPDN_02);

Position_error := UPDN_02.Out;
```

功能块


UPDN_02
UPDN
Up / Down Accumulator
Initialize
InitialValue
InPlus
InMinus
Out
Position_error
PMUL_02
PMUL
Pulse Multiplier
In
Initialize
InitialValue
WordSize
Multiplier
Out
马达 A
Position_feedback
12
25000
Initialize_Position
PMUL_03
PMUL
Pulse Multiplier
In
Initialize
InitialValue
WordSize
Multiplier
Out
马达 B
Follower_Position
12
100000

S- 曲线 (SCRV)

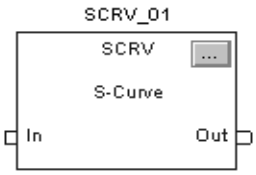
SCRV 指令利用附加的颠簸率执行斜坡函数。颠簸率指用来从输出产生输入的斜率的最大变化率。

操作数：

```
SCRV(SCRV_tag);
```

结构文本

操作数：	类型：	格式：	说明：
SCRV 标记	S_CURVE	结构	SCRV 结构



功能块

操作数：	类型：	格式：	说明：
SCRV 标记	S_CURVE	结构	SCRV 结构

S_CURVE 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果该位清零，则指令不执行且输出不更新。 如果该位已设置，则指令执行。 缺省状态为已设置。 **结构化文本：** 无效。指令执行。
In	REAL	输入到指令的模拟信号。 有效值 = 浮点数 缺省值 = 0.0
Initialize	BOOL	指令的 Initialize 输入。如果设置，指令将保持 Out = InitialValue 缺省状态为清零。
InitialValue	REAL	S- 曲线的初始值。如果设置了 Initialize，则 Out = InitialValue。 有效值 = 浮点数 缺省值 = 0.0
AbsAlgRamp	BOOL	斜坡类型。如果设置，指令将起到绝对值斜坡的作用。如果清零，指令将起到代数斜坡的作用。 缺省状态为已设置。
AccelRate	REAL	加速率，单位为 “输入单位 / 秒^2”。零值将阻止 Out 加速。如果 AccelRate < 0，指令将假定 AccelRate = 0 且设置状态字相应位。 有效值 = 0.0 到最大正浮点数 缺省值 = 0.0
DecelRate	REAL	减速率，单位为 “输入单位 / 秒^2”。零值将阻止 Out 减速。如果 DecelRate < 0，指令将假定 DecelRate = 0 且设置状态字相应位。 有效值 = 0.0 到最大正浮点数 缺省值 = 0.0

输入参数：	数据类型：	说明：
JerkRate	REAL	颠簸率，单位为 “输入单位 / 秒 ^{3} ”。指定从输出产生输入时加速率和减速率的最大变化率。当 (JerkRate *DeltaT) ≥ AccelRate 和 （或） DecelRate 时，加速率和减速率不受限制。这时，指令起到斜坡函数的作用。当 JerkRate < 0 时，指令假定 JerkRate = 0 且设置状态字相应位。 有效值 = 0.0 到最大正浮点数 缺省值 = 0.0
HoldMode	BOOL	S- 曲线保持模式参数。该参数结合 HoldEnable 参数使用。当 HoldEnable 已设置且 Rate = 0 时，指令将保持 Out 不变。这时，一旦设置 HoldEnable，指令就会保持 Out， JerkRate 被忽略， Out 在其轮廓中产生一个 “拐点”。当 HoldEnable 已设置时，如果 HoldMode 清零，指令将使用 JerkRate 使 Out 变为恒定值。当 Rate = 0 时， Out 保持不变。一旦 HoldEnable 已设置，请勿改变 HoldMode，因为指令将忽略该改变。 缺省状态为清零。
HoldEnable	BOOL	S- 曲线保持启用参数。如果设置， Out 将被保持。如果清零， Out 将从其当前值移动，直到等于 In。 缺省状态为清零。
TimingMode	DINT	选择定时执行模式。 **值：** **说明：** 0 周期模式 1 过采样模式 2 实时采样模式 有关定时模式的更多信息，请参阅附录 “功能块属性”。 有效值 = 0 到 2 缺省值 = 0
OversampleDT	REAL	过采样模式下的运行时间。 有效值 = 0 到 4194.303 秒 缺省值 = 0
RTSTime	DINT	实时采样模式下的模块更新周期 有效值 = 1 到 32,767 毫秒 缺省值 = 1
RTSTimeStamp	DINT	实时采样模式下的模块时间戳值。 有效值 = 0 到 32,767 毫秒 缺省值 = 0

输出参数：	数据类型：	说明：
EnableOut	BOOL	输出使能。
S_Mode	BOOL	S_Mode 输出。当 (Jerk * DeltaT) ≤ Rate 且 Rate < Accel 或 Decel 时，会设置 S_Mode。否则 S_Mode 将清零。
Out	REAL	S- 曲线指令的输出。该输出要设置算术状态标志。
Rate	REAL	Out 的内部变化，单位为 “单位 / 秒”。
DeltaT	REAL	更新之间的时间差值。通过控制算法计算过程输出时所用的时间 （单位为秒）。
Status	DINT	功能块的状态。

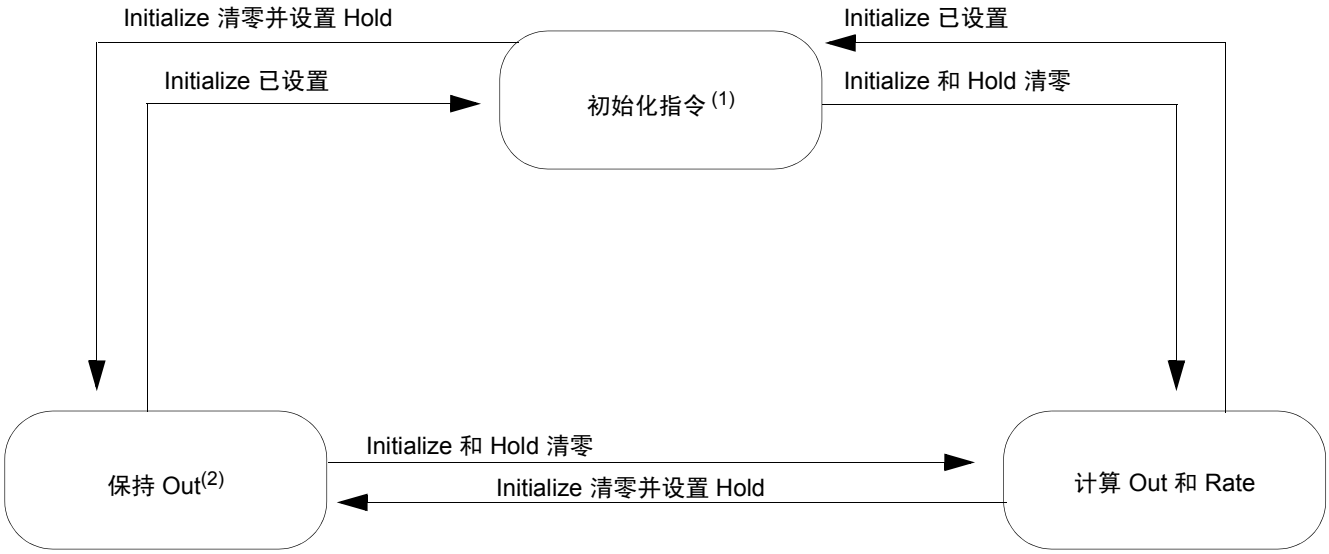
输出参数：	数据类型：	说明：
InstructFault (Status.0)	BOOL	指令检测到以下运行错误之一。该错误不是次要或主要的控制器错误。检查其他状态位以确定错误类型。
AccelRateInv (Status.1)	BOOL	AccelRate 为负。
DecelRateInv (Status.2)	BOOL	DecelRate 为负。
JerkRateInv (Status.3)	BOOL	JerkRate 为负。
TimingModeInv (Status.27)	BOOL	定时模式无效。 有关定时模式的更多信息，请参阅附录 “功能块属性”。
RTSMissed (Status.28)	BOOL	仅用于实时采样模式。 ABS | DeltaT - RTSTime | > 1 （0.001 秒）时设置。
RTSTimeInv (Status.29)	BOOL	RTSTime 值无效。
RTSTimeStampInv (Status.30)	BOOL	RTSTimeStamp 值无效。
DeltaT (Status.31)	BOOL	DeltaT 值无效。

说明： SCRV 指令的首要要求是确保速率变化不会超过指定的颠簸率。

可以配置 SCRV 指令，以便为步进输入产生 S- 曲线轮廓或斜坡轮廓。

SCRV 轮廓：	说明：
S- 曲线轮廓	若要产生 S- 曲线轮廓，设置 JerkRate 使得 (JerkRate * DeltaT) < AccelRate 和 （或） DecelRate。 在 S- 曲线轮廓模式下， SCRV 指令确保速率变化绝不会超过指定的 JerkRate。用来产生 S- 曲线轮廓的算法可以为步进输入产生平滑、对称的 S- 曲线。其中整合了 Out 的梯形积分，进行协助。结果，在轮廓的某些部分中，速率变化将小于 JerkRate。 当输入中出现步进变化时，速率增加到编程设置的 AccelRate 或 DecelRate。 AccelRate 或 DecelRate 将维持到速率必须开始下降的一点，从而在速率达到零时使输出达到输入。 某些情形下，根据加速、减速和颠簸值，在速率必须开始按颠簸率下降之前，可能并未达到加速率或减速率。 对于非常小的步进变化， SCRV 指令不会试图产生 "S" 轮廓。该模式下，整步都会被输出，速率将反映输出变化。如果 Out = In 将出现该现象， In 的下一个步进变化可以按小于或等于编程设置的 JerkRate 的速率输出。 SCRV 指令支持代数斜坡和绝对值斜坡。对于代数斜坡，加速条件是由越来越大的正数输入定义，而减速条件则是由越来越小的负数输入定义。对于绝对值斜坡，加速条件是由偏离零的输入定义，而减速条件则是由偏向零的输入定义。
斜坡轮廓	若要产生斜坡轮廓，设置 JerkRate 使得 (JerkRate * DeltaT) ≥ AccelRate 和 （或） DecelRate。 在斜坡轮廓模式下， SCRV 指令始终产生等于编程设置的 AccelRate 或 DecelRate 的变化率，直到 Out 与 In 之间的差值要求变化率小于 AccelRate 或 DecelRate 以达到端点。 HoldMode = 0 的操作与 HoldMode = 1 的操作相同。当 HoldEnable 已设置时， Out 立即被保持，速率变为零。

下图显示指令如何修正 Out。



(1) 当 Initialize 已设置时，指令设置如下：

Out_n = InitialValue

Out_{n-1}= Out_n

$Rate_n$= 0

$Rate_{n-1}$ = 0

(2) 当 HoldMode 清零时， Out 移向 In，且 HoldEnable 被设置，速率开始以颠簸率减少到零。由于 JerkRate， Out 被保持在当速率达到零时它所具有的值。当 Out 最终保持不变时，它的值不同于设置 HoldEnable 那一刻时它所具有的值。

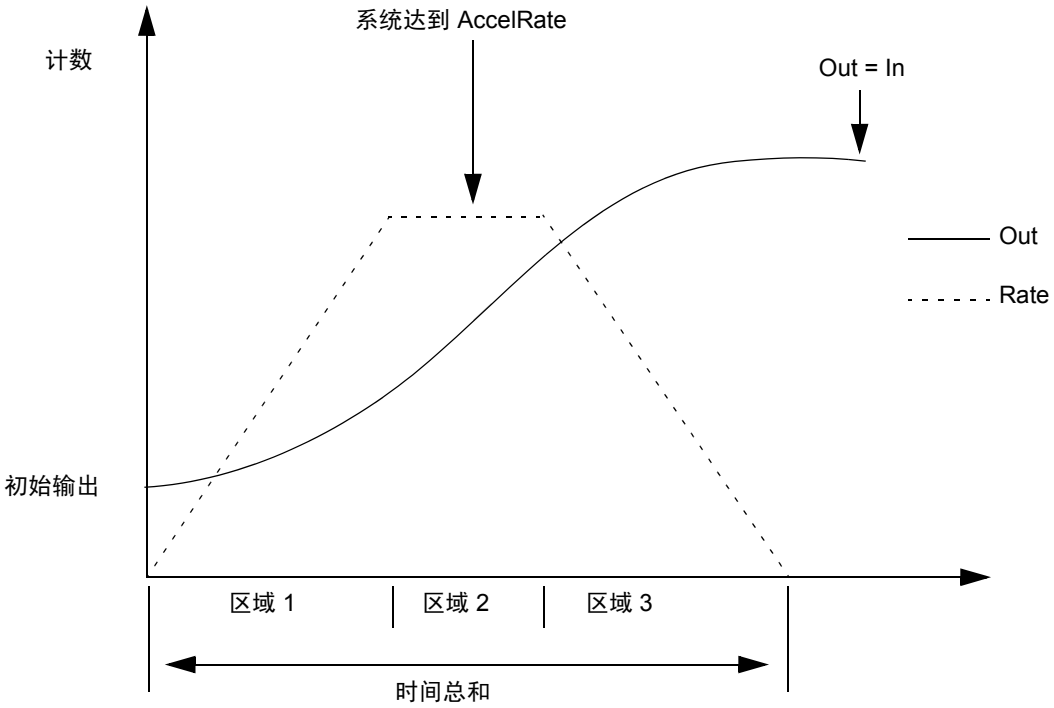
当 HoldMode 已设置时， Out 移向 In，且 HoldEnable 被设置，速率立即被设置为零。Out 保持在 HoldEnable 被设置时它所具有的值。

转变过程中降低 JerkRate 可能会导致 Out 过冲 In。发生过冲是因为强制实行输入的 JerkRate。调谐时小步减少 JerkRate，或者在 Out = In 时 （非转变过程中）改变 JerkRate，可以避免过冲。

Out 等于输入变化所需的时间是 AccelRate、JerkRate 和 In 与 Out 之间差值的函数。

计算输出和速率值

在从初始值到最终值的转变中， Out 经过三个区域。在区域 1 和区域 3，Out 的变化率基于 JerkRate。在区域 2， Out 的变化率基于 AccelRate 或 DecelRate。



各区域中 Out 的计算如下：

$$TotalTime = \frac{FinalOutput - InitialOutput}{AccelRate} + \frac{AccelRate}{JerkRate}$$

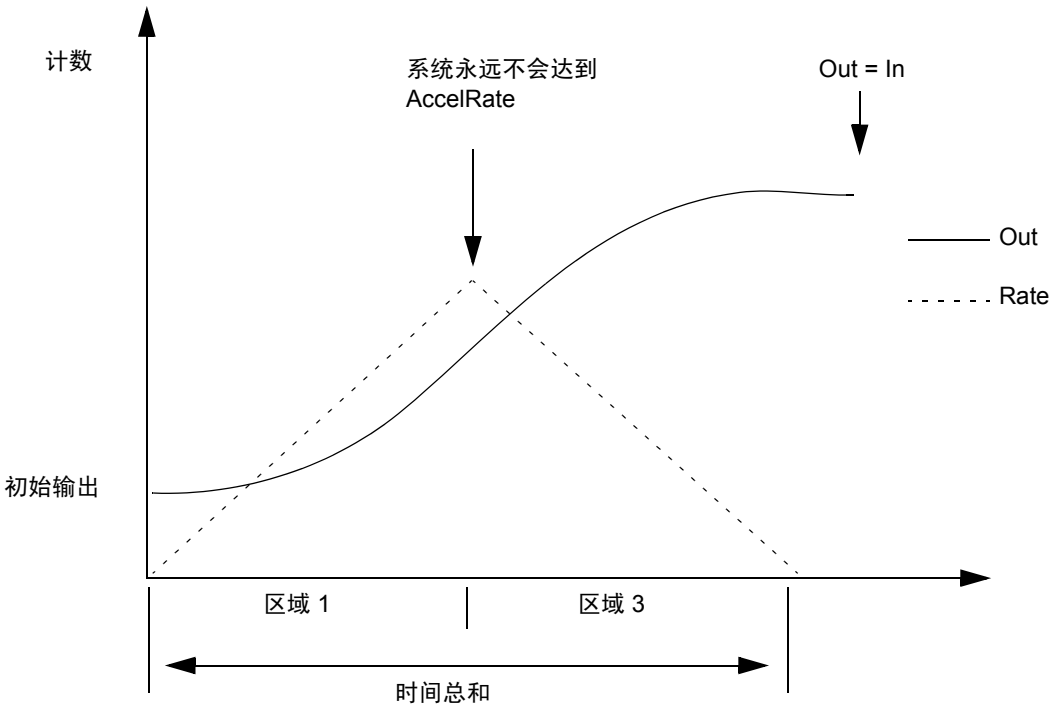
每个区域的方程式如下：

区域：	方程式：
区域 1	$Time_1 = \frac{AccelRate}{JerkRate}$ $Y(Time) = InitialOutput + \frac{1}{2}(JerkRate) \times Time^2$
区域 2	$Time_2 = \frac{JerkRate \times (FinalOutput - InitialOutput) - AccelRate^2}{JerkRate \times AccelRate}$ $Y(Time) = InitialOutput + (AccelRate \times Time) - \frac{AccelRate^2}{2 \times JerkRate}$
区域 3	$Time_3 = \frac{AccelRate}{JerkRate}$ $Y(Time) = FinalOutput - \frac{1}{2}(JerkRate) \times \left(Time - \frac{FinalOutput - InitiaOutput}{AccelRate} - \frac{AccelRate}{JerkRate} \right)^2$

条件：

$$|InitialOutput - FinalOutput| < \frac{AccelRate^2}{JerkRate}$$

SCRV 块不会达到 AccelRate 或 DecelRate。 Out 如下：



图中：

$$TotalTime = 2 \times \sqrt{\frac{|InitialOutput - FinalOutput|}{JerkRate}}$$

算术状态标志： Out 输出影响算术状态标志。

错误条件： 无

执行：

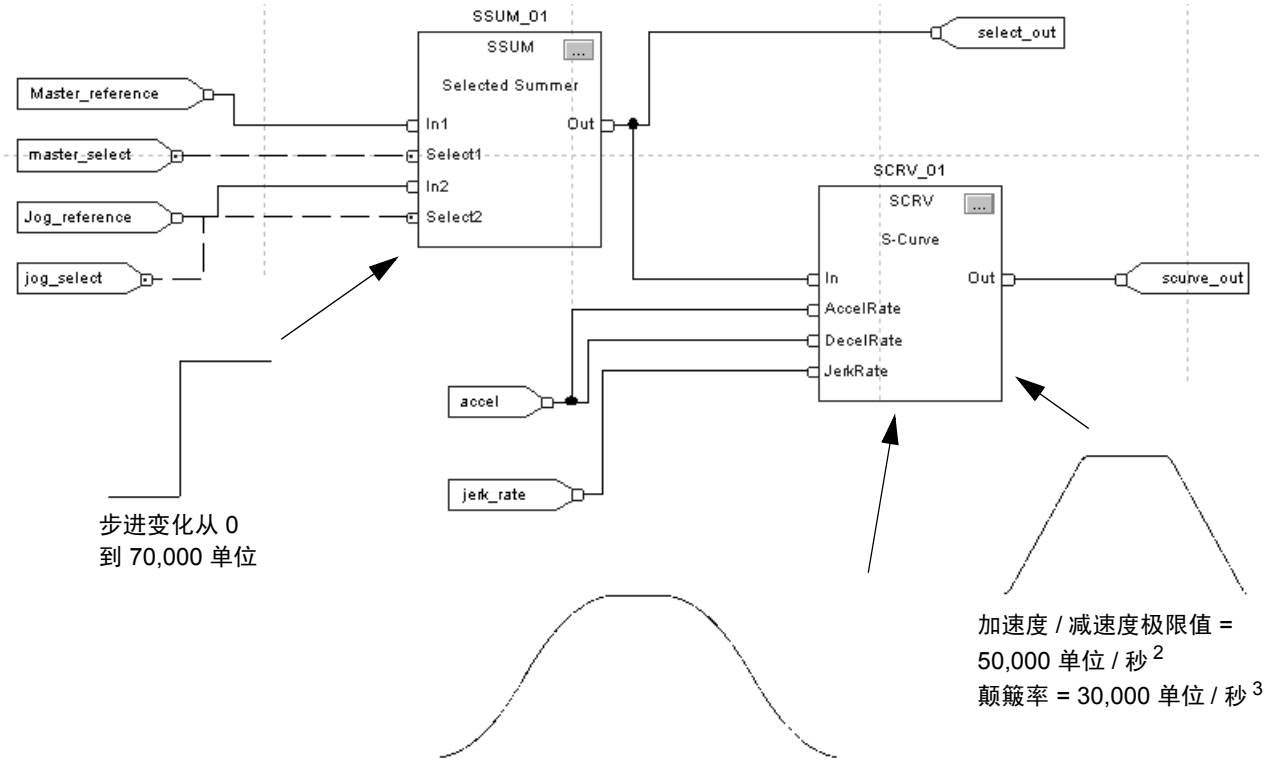
条件：	功能块动作：	结构化文本动作：
预扫描	无动作。	无动作。
指令首次扫描	初始化内部变量。	初始化内部变量。
指令首次执行	无动作。	无动作。
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。	na
EnableIn 已设置	指令执行。 EnableOut 已设置。	EnableIn 一直保持已设置状态。 指令执行。
后期扫描	无动作。	无动作。

实例： 在大多数坐标驱动应用中，主参考决定整组驱动的线速度。随着参考选择的多样化，驱动的速度参考中不能呈现 “步进”变化，因为载荷惯性、马达力矩和调谐的差异不会允许单个驱动部分以协同模式作出反应。 SCRV 指令旨在倾斜和整形到达驱动部分的参考信号，以便控制加速、减速和颠簸 （加速度的导数）。该指令提供一种机制，以允许到达驱动的参考以下述模式达到指定的参考设置点，该模式可以消除过大的力和对相连的机械和设备的过大影响。

结构化文本

```
SSUM_01.In1 := Master_reference;
SSUM_01.Select1 := master_select;
SSUM_01.In2 := Jog_reference;
SSUM_01.Select2 := Jog_Select;
SSUM(SSUM_01);

select_out := SSUM_01.Out;

SCRV_01.In := select_out;
SCRV_01.AccelRate := accel;
SCRV_01.DecelRate := accel;
SCRV_01.JerkRate := jerk_rate;
SCRV(SCRV_01);

scurve_out := SCRV_01.Out
```

功能块


SSUM_01
SSUM
Selected Summer
Master_reference
master_select
Jog_reference
jog_select
In1
Select1
In2
Select2
Out
select_out
SCRV_01
SCRV
S-Curve
In
Out
scurve_out
AccelRate
DecelRate
JerkRate
accel
jerk_rate
步进变化从 0
到 70,000 单位
加速度 / 减速度极限值 = 50,000 单位 / 秒 2
颠簸率 = 30,000 单位 / 秒 3
加速度 / 减速度极限值 = 50,000 单位 / 秒 2
颠簸率 = 3,000 单位 / 秒 3

二阶控制器 (SOC)

SOC 指令旨在以与 PI 指令相似的模式用于闭环控制系统中。 SOC 指令提供了一个增益项，一个一阶滞后校正项和一个二阶超前校正项。

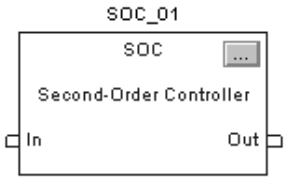
操作数：

结构文本

```
SOC(SOC_tag);
```

操作数：	类型：	格式：	说明：
SOC 标记	SEC_ORDER_CONTROLLER	结构	SOC 结构

功能块



操作数：	类型：	格式：	说明：
SOC 标记	SEC_ORDER_CONTROLLER	结构	SOC 结构

SEC_ORDER_CONTROLLER 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果该位清零，则指令不执行且输出不更新。 如果该位已设置，则指令执行。 缺省状态为已设置。 **结构化文本：** 无效。指令执行。
In	REAL	输入到指令的模拟信号。 有效值 = 浮点数 缺省值 = 0.0
Initialize	BOOL	指令初始化命令。已设置时， Out 和内部积分器被设置为与 InitialValue 的值相等。 缺省状态为清零。
InitialValue	REAL	初始化输入值。 Initialize 已设置时， Out 和积分器被设置为 InitialValue 的值。使用 HighLimit 和 LowLimit 限制 InitialValue 值。 有效值 = 浮点数 缺省值 = 0.0
Gain	REAL	指令的比例增益。如果该值超限，则指令会对其限制并设置状态字中的相应位。 有效值 = 任意浮点数 > 0.0 缺省值 = 最小正浮点数
WLag	REAL	一阶滞后角频率，单位为弧度 / 秒。如果该值超限，则指令会对其限制并设置状态字中的相应位。 有效值 = 参阅下面有关有效范围的说明 缺省值 = 最大正浮点数
WLead	REAL	二阶超前角频率，单位为弧度 / 秒。如果该值超限，则指令会对其限制并设置状态字中的相应位。 有效值 = 参阅下面有关有效范围的说明 缺省值 = 0.0

输入参数：	数据类型：	说明：
ZetaLead	REAL	二阶超前阻尼因子。如果该值超限，则指令会对其限制并设置状态字中的相应位。 有效值 = 0.0 到 10.0 缺省值 = 0.0
HighLimit	REAL	上限值。即输出的最大值。如果 HighLimit ≤ LowLimit，则指令设置 HighAlarm 和 LowAlarm 并设置状态字中相应位，且设置 Out = LowLimit。 有效值 = LowLimit < HighLimit ≤ 最大正浮点数 缺省值 = 最大正浮点数
LowLimit	REAL	下限值。即输出的最小值。如果 HighLimit ≤ LowLimit，则指令设置 HighAlarm 和 LowAlarm 并设置状态字中相应位，且设置 Out = LowLimit。 有效值 = 最大负浮点数 ≤ LowLimit < HighLimit 缺省值 = 最大负浮点数
HoldHigh	BOOL	上限保持命令。已设置时，不允许增加内部积分器的值。 缺省状态为清零。
HoldLow	BOOL	下限保持命令。已设置时，不允许减少内部积分器的值。 缺省状态为清零。
TimingMode	DINT	选择定时执行模式。 **值：** **说明：** 0 周期模式 1 过采样模式 2 实时采样模式 有关定时模式的更多信息，请参阅附录 “功能块属性”。 有效值 = 0 到 2 缺省值 = 0
OversampleDT	REAL	过采样模式下的运行时间。 有效值 = 0 到 4194.303 秒 缺省值 = 0
RTSTime	DINT	实时采样模式下的模块更新周期 有效值 = 1 到 32,767 毫秒 缺省值 = 1
RTSTimeStamp	DINT	实时采样模式下的模块时间戳值。 有效值 = 0 到 32,767 毫秒 缺省值 = 0

输出参数：	数据类型：	说明：
EnableOut	BOOL	输出使能。
Out	REAL	算法计算的输出。该输出要设置算术状态标志。
HighAlarm	BOOL	上限报警指示。当 Out 的计算值 ≥ HighLimit 时设置，输出被限制在 HighLimit。
LowAlarm	BOOL	下限报警指示。当 Out 的计算值 ≤ LowLimit 时设置，输出被限制在 LowLimit。
DeltaT	REAL	更新之间的时间差值。通过控制算法计算过程输出时所用的时间 （单位为秒）。
Status	DINT	功能块的状态。

输出参数：	数据类型：	说明：
InstructFault (Status.0)	BOOL	指令检测到以下运行错误之一。该错误不是次要或主要的控制器错误。检查其他状态位以确定错误类型。
GainInv (Status.1)	BOOL	Gain > 最大值或 Gain < 最小值。
WLagInv (Status.2)	BOOL	WLag > 最大值或 WLag < 最小值。
WLeadInv (Status.3)	BOOL	WLead > 最大值或 WLead < 最小值。
ZetaLeadInv (Status.4)	BOOL	ZetaLead > 最大值或 ZetaLead < 最小值。
HighLowLimsInv (Status.5)	BOOL	HighLimit ≤ LowLimit。
TimingModeInv (Status.27)	BOOL	定时模式无效。 有关定时模式的更多信息，请参阅附录 “功能块属性”。
RTSMissed (Status.28)	BOOL	仅用于实时采样模式。 ABS | DeltaT - RTSTime | > 1 （0.001 秒）时设置。
RTSTimeInv (Status.29)	BOOL	RTSTime 值无效。
RTSTimeStampInv (Status.30)	BOOL	RTSTimeStamp 值无效。
DeltaT (Status.31)	BOOL	DeltaT 值无效。

说明： SOC 指令提供了一个增益项，一个一阶滞后校正项和一个二阶超前校正项。滞后的频率可调，超前的频率和阻尼也可调。二阶超前校正的零对可以为复数 （阻尼 < 一个单位）或实数 （阻尼 ≥ 到一个单位）。SOC 指令旨在用于扫描速率保持不变的任务中。

SOC 指令使用以下 Laplace Transfer 方程式。

$$H(s) = \frac{K\left(\frac{s^2}{\omega_{Lead}^2} + \frac{2 \times \xi_{Lead} \times s}{\omega_{Lead}} + 1\right)}{s\left(\frac{s}{\omega_{Lag}} + 1\right)}$$

参数限制

以下 SOC 参数的有效值有如下限制。

参数：	限值：
WLead	$LowLimit = \frac{0.00001}{DeltaT}$ $HighLimit = \frac{0.07\pi}{DeltaT}$ DeltaT 的单位为秒
WLag	$LowLimit = \frac{0.0000001}{DeltaT}$ $HighLimit = \frac{0.07\pi}{DeltaT}$ DeltaT 的单位为秒
ZetaLead	$LowLimit = 0.0$ $HighLimit = 10.0$

无论何时，如果输出的计算值无效或为 NAN，指令将设置 Out = 该无效值并设置算术溢出状态标志。内部参数不更新。在后续每次扫描中，都使用上一次输出有效时的扫描的内部参数来计算输出。

极限

指令根据 Hold 输入的状态阻止结束。

如果：	目的：
HoldHigh 已设置**并且** Integrator > $Integrator_{n-1}$	Integrator = $Integrator_{n-1}$
HoldLow 已设置**并且** Integrator < $Integrator_{n-1}$	Integrator = $Integrator_{n-1}$

指令还根据 HighLimit 和 LowLimit 值阻止积分器结束。

如果：	目的：
Integrator > IntegratorHighLimit	Integrator = IntegratorHighLimit
Integrator < IntegratorLowLimit	Integrator = IntegratorLowLimit

表中：

$$IntegratorHighLimit = HighLimit \times \frac{Gain \times WLag}{WLead^2}$$

$$IntegratorLowLimit = LowLimit \times \frac{Gain \times WLag}{WLead^2}$$

指令还根据 HighLimit 和 LowLimit 值限制 Out 值。

如果：	目的：
HighLimit ≤ LowLimit	Out = LowLimit Integrator = IntegratorLowLimit HighLowLimsInv 已设置 HighAlarm 已设置 LowAlarm 已设置
Out ≥ HighLimit	Out = HighLimit Integrator = $Integrator_{n-1}$ HighAlarm 已设置
Out ≤ LowLimit	Out = LowLimit Integrator = $Integrator_{n-1}$ LowAlarm 已设置

算术状态标志： Out 输出影响算术状态标志。

错误条件： 无

执行：

条件：	功能块动作：	结构化文本动作：
预扫描	无动作。	无动作。
指令首次扫描	指令设置内部参数并设置 Out = 0。 控制算法不执行。	
指令首次执行	指令设置内部参数并设置 Out = 0。 控制算法不执行。	
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。	na
EnableIn 已设置	指令执行且 EnableOut 被设置。	EnableIn 一直保持已设置状态。 指令执行。
后期扫描	无动作。	无动作。

实例： SOC 指令是一个专门的功能块，用于能量在两个部分之间通过弹簧质量系统转移的应用中。在这类应用中，程序本身的频率响应特征通常可以用下图 A 表示：

SOC 指令实施一个一阶滞后滤波，然后由 PID 控制器利用一个积分、一个二阶零 （超前）和一个一阶极 （滞后）实施一个转移函数。利用该指令可以简化 PID 调谐，因为调节项的安排使得 WLead 和 ZLead 成为 SOC 指令的输入，而不是 Kp、 Ki 和 Kd 值。 SOC 指令的转移函数为：

$$H(s) = \frac{K\left(\frac{s^2}{\omega_{Lead}^2} + \frac{2 \times \xi_{Lead} \times s}{\omega_{Lead}} + 1\right)}{s\left(\frac{s}{\omega_{Lag}} + 1\right)}$$

图 A：程序特征

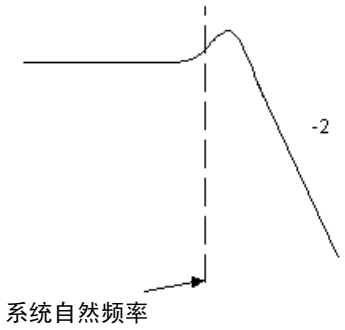
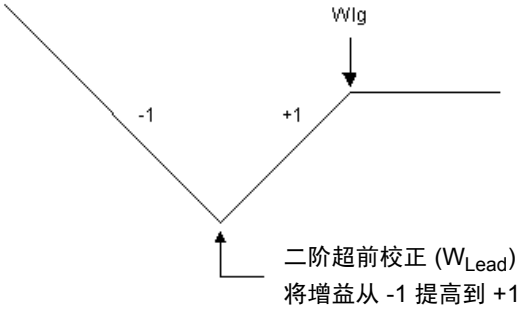
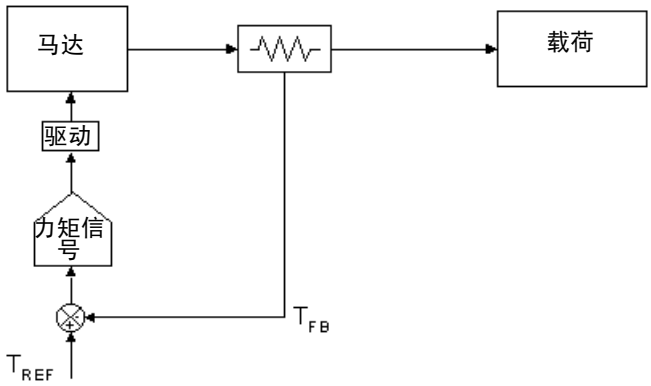


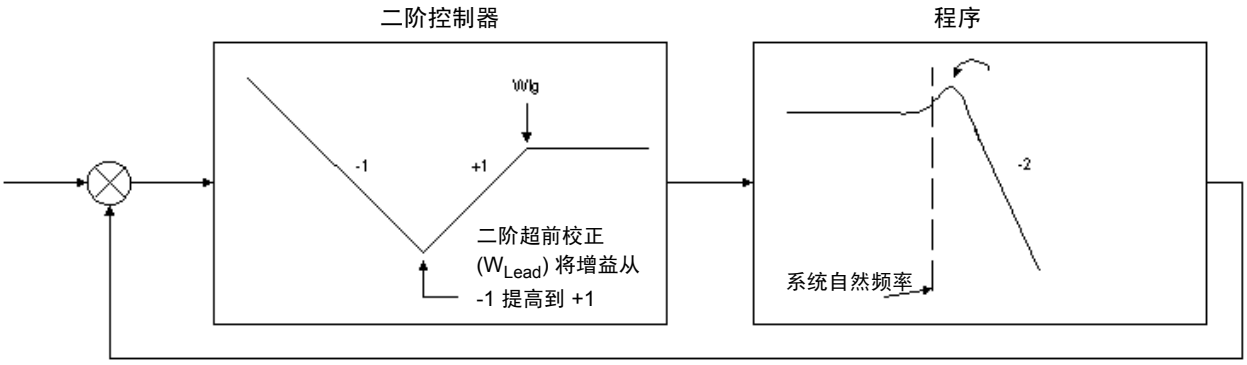
图 B：二阶控制器



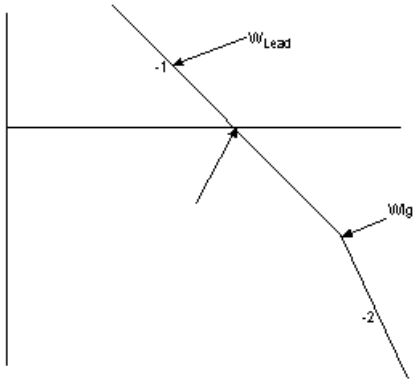
SOC 指令可以用在力矩或张力调节应用中，此类应用中，载荷单元或力变换器用作反馈，调节方案的输出直接作用在驱动器的力矩（当前）小环上。在许多此类应用中，受控系统在机械上可能处于欠阻尼，并具有难以稳定的自然频率，因为该频率通过反馈设备自身被反映。



利用 SOC 指令可以简化 PID 调谐，因为调节项的安排使得 WLead 和 ZLead 成为 SOC 指令的输入，而不是 Kp、Ki 和 Kd 值。这样，控制器 / 调节器的角频率更容易根据实际程序进行调节和设置。启动期间，系统的自然频率和阻尼因子可以根据经验测量或现场测量。然后，可以调整调节器的参数以与程序特征匹配，使得最终程序可以具有更高的增益和更稳定的控制。



在以上的系统中，如果 Wlead 被设置成等于系统自然频率，而 Wlag 被设置成高于所需的交叉频率很多 （> 5 倍的交叉频率），则所得到的系统响应将如下图所示：



实际应用中，使用和设置该指令的步骤为：

1. 识别所控制的程序的类型。如果系统对步进函数的响应引起高度响铃，或者其特征可以用以上的程序曲线表示，则该块可以提供稳定控制所需的调节特征。

2. 确定系统 / 程序的自然频率。这可以通过经验模式办到，或者可以在现场测量。调整 WLead 使得它与程序本身的自然频率相对应或比后者稍微超前。

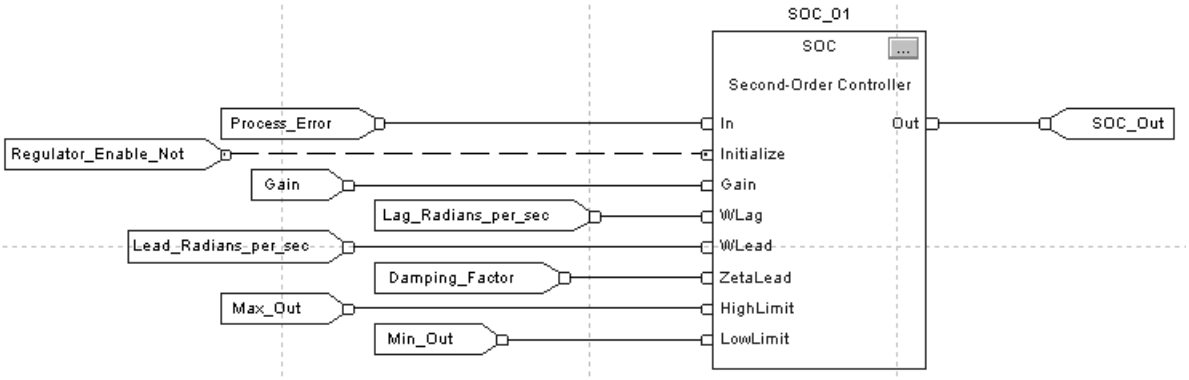
3. 调整阻尼因子 Zlead，使得它抵消系统中的任何过冲。

4. 提高 WLag，使它远远超过系统交叉频率 （>5 倍），并开始增加总体增益以达到所需的系统响应。

结构化文本

```
SOC_01.In := Process_Error;
SOC_01.Initialize := Regulator_Enable_Not;
SOC_01.Gain := Gain;
SOC_01.WLag := Lag_Radians_per_sec;
SOC_01.WLead := Lead_radians_per_sec;
SOC_01.ZetaLead := Damping_Factor;
SOC_01.HighLimit := Max_Out;
SOC_01.LowLimit := Min_Out;
SOC(SOC_01);

SOC_Out := SOC_01.Out;
```

功能块



增 / 减累加器 (UPDN)

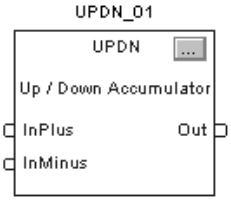
UPDN 指令将两个输入相加和相减成累积值。

操作数：

```
UPDN(UPDN_tag);
```

结构文本

操作数：	类型：	格式：	说明：
UPDN 标记	UP_DOWN_ACCUM	结构	UPDN 结构



功能块

操作数：	类型：	格式：	说明：
UPDN 标记	UP_DOWN_ACCUM	结构	UPDN 结构

UP_DOWN_ACCUM 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果该位清零，则指令不执行且输出不更新。 如果该位已设置，则指令执行。 缺省状态为已设置。 **结构化文本：** 无效。指令执行。
Initialize	BOOL	指令的初始化输入请求。如果设置 Initialize，指令将设置 Out 和内部累加器为 InitialValue。 缺省状态为清零。
InitialValue	REAL	指令的初始化值。 有效值 = 浮点数 缺省值 = 0.0
InPlus	REAL	增加到累加器中的输入。 有效值 = 浮点数 缺省值 = 0.0
InMinus	REAL	从累加器中减去的输入。 有效值 = 浮点数 缺省值 = 0.0
保持	BOOL	指令的保持输入请求。如果 Hold 已设置且 Initialize 清零， Out 将被保持。 缺省状态为清零。
输出参数：	**数据类型：**	**说明：**
EnableOut	BOOL	输出使能。
Out	REAL	指令的输出。该输出影响算术状态滞后校正。

说明： UPDN 指令遵循以下算法。

条件：	动作：
Hold 清零且 Initialize 清零	$AccumValue_n = AccumValue_{n-1} + InPlus - InMinus$ $Out = AccumValue_n$
Hold 已设置且 Initialize 清零	$AccumValue_n = AccumValue_{n-1}$ $Out = AccumValue_n$
Initialize 已设置	$AccumValue_n = InitialValue$ $Out = AccumValue_n$

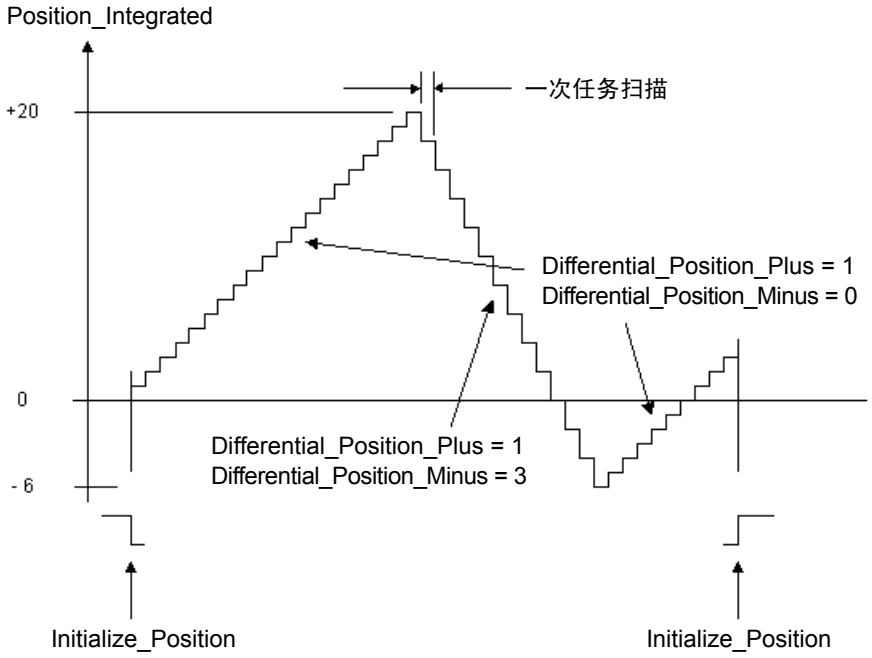
算术状态标志： Out 输出影响算术状态标志。

错误条件： 无

执行：

条件：	功能块动作：	结构化文本动作：
预扫描	无动作。	无动作。
指令首次扫描	无动作。	无动作。
指令首次执行	$AccumValue_{n-1} = 0.0$	$AccumValue_{n-1} = 0.0$
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。	na
EnableIn 已设置	指令执行。 EnableOut 已设置。	EnableIn 一直保持已设置状态。 指令执行。
后期扫描	无动作。	无动作。

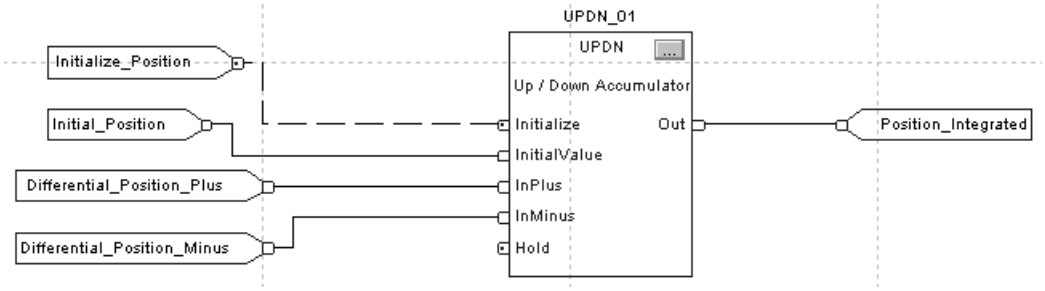
实例： UPDN 指令对从一次输入到下一次输入的计数求积分。该指令可以用于简单的定位应用，或者其他类型的应用，在这些应用中，需要进行简单的积分以从程序的微分反馈信号中产生累积值。在下例中，Initial_Position 被设置成零，而 Differential_Position_Plus 和 Differential_Position_Minus 在一段时间内取不同的值。利用该指令，InPlus 和 InMinus 也可以接受负值。



结构化文本

```
UPDN_01.Initialize := Initialize_Position;
UPDN_01.InitialValue := Initial_Position;
UPDN_01.InPlus := Differential_Position_Plus;
UPDN_01.InMinus := Differential_Position_Minus;
UPDN(UPDN_01);

Position_Integrated := UPDN_01.Out;
```

功能块



滤波指令
（DERV、 HPF、 LDL2、 LPF、 NTCH）

简介

这些滤波指令可用于：

如要：	所用指令：	可用编程语言：	参见页：
计算每秒单位时间内信号的变化量。	微分 (DERV)	结构化文本 功能块	3-2
过滤输入频率中低于截止频率的部分。	高通滤波 (HPF)	结构化文本 功能块	3-6
以极点对和零点对为参数进行滤波。	二阶超前 / 滞后滤波 (LDL2)	结构化文本 功能块	3-12
过滤输入频率中高于截止频率的部分。	低通滤波 (LPF)	结构化文本 功能块	3-18
过滤输入频率中等于陷波频率的部分。	陷波滤波 (NTCH)	结构化文本 功能块	3-24

微分 (DERV)

DERV 指令可计算每秒单位时间内信号的变化量。

操作数：

```
DERV(DERV_tag);
```

结构化文本

操作数：	类型：	格式：	说明：
DERV 标记	DERIVATIVE	结构	DERV 结构

功能块

操作数：	类型：	格式：	说明：
DERV 标记	DERIVATIVE	结构	DERV 结构

DERIVATIVE 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果该位清零，则指令不执行且输出不更新。 如果置位，则指令执行。 默认状态为置位。 **结构化文本：** 无效。指令执行。
In	REAL	输入到指令的模拟信号。 有效值 = 浮点数 默认值 = 0.0
Gain	REAL	微分系数 有效值 = 浮点数 默认值 = 1.0
ByPass	BOOL	算法的旁路请求。ByPass 置位时，指令设置 Out = In。 默认状态为清零。
TimingMode	DINT	选择定时执行方式。 **值：** **说明：** 0 周期方式 1 过采样方式 2 实时采样方式 有关定时方式的更多信息，请参阅附录功能块属性。 有效值 = 0 到 2 默认值 = 0

输入参数：	数据类型：	说明：
OversampleDT	REAL	过采样方式下的运行时间。 有效值 = 0 到 4194.303 秒 默认值 = 0
RTSTime	DINT	实时采样方式下的模块更新周期 有效值 = 1 到 32,767 毫秒 默认值 = 1
RTSTimeStamp	DINT	实时采样方式下的模块时间戳值。 有效值 = 0 到 32,767 毫秒 默认值 = 0

输出参数：	数据类型：	说明：
EnableOut	BOOL	输出使能。
Out	REAL	算法计算出的输出。该输出影响算术状态标志。
DeltaT	REAL	两次更新之间的时间差值。通过控制算法计算过程输出时所用的时间（单位为秒）。
Status	DINT	功能块的状态。
InstructFault (Status.0)	BOOL	指令检测到如下运行错误。该错误不是次要或主要的控制器错误。检查其他状态位以确定错误类型。
TimingModeInv (Status.27)	BOOL	TimingMode 值无效。 有关定时方式的更多信息，请参阅附录功能块属性。
RTSMissed (Status.28)	BOOL	仅用于实时采样方式。ABS | DeltaT - RTSTime | > 1 （0.001 秒）时置位。
RTSTimeInv (Status.29)	BOOL	RTSTime 值无效。
RTSTimeStampInv (Status.30)	BOOL	RTSTimeStamp 值无效。
DeltaTInv (Status.31)	BOOL	DeltaT 值无效。

说明： DERV 指令支持旁路输入，允许您停止计算信号的变化量，直接将信号传至输出。

当 Bypass 为：	指令使用下列公式：
置位	$Out = In_n$ $In_{n-1} = In_n$
清零<u>且</u> DeltaT > 0	$Out = Gain\frac{In_n - In_{n-1}}{DeltaT}$ $In_{n-1} = In_n$ DeltaT 的单位为秒

算术状态标志： Out 输出影响算术状态标志。

故障条件： 无

执行：

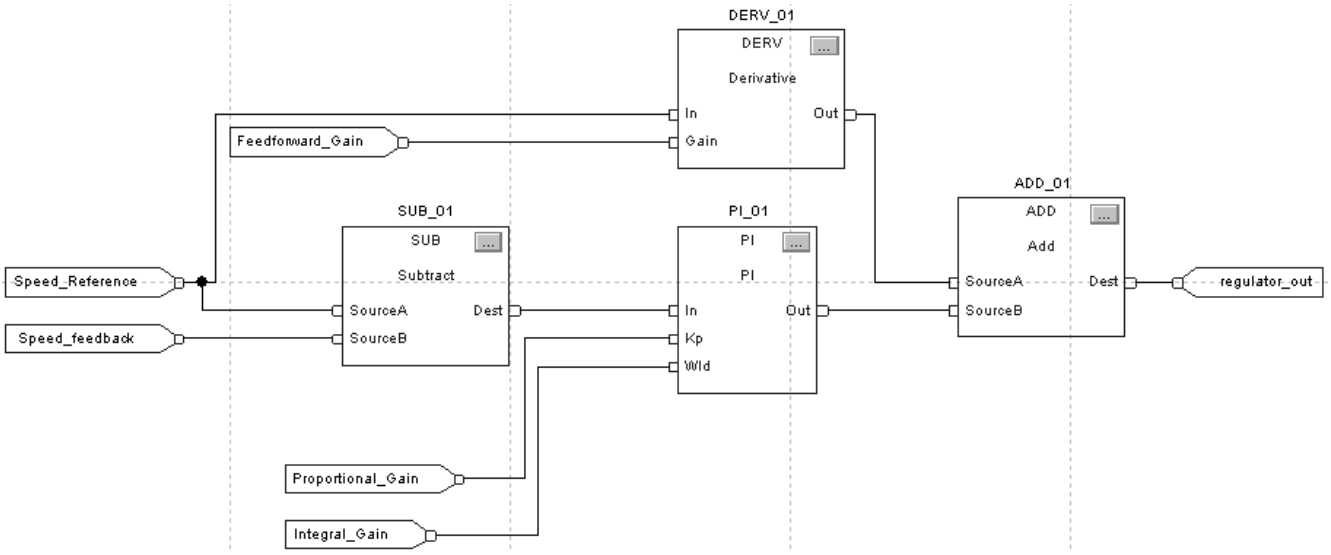
条件：	功能块动作：	结构化文本动作：
预扫描	无动作。	无动作。
指令首次扫描	$In_{n-1} = In_n$	$In_{n-1} = In_n$
指令首次执行	$In_{n-1} = In_n$	$In_{n-1} = In_n$
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。	na
EnableIn 置位	指令执行。 EnableOut 置位。	EnableIn 一直保持置位。 指令执行。
后期扫描	无动作。	无动作。

示例： 微分指令可计算单位时间 （每秒）内信号的变化量。该指令通常用于闭环控制，在调节器中形成前馈通路以补偿大惯性过程量。

结构化文本

```
DERV_01.In := Speed_Reference;
DERV_01.Gain := Feedforward_Gain;
DERV(DERV_01);

PI_01.In := Speed_Reference - Speed_feedback;
PI_01.Kp := Proportional_Gain;
PI_01.Wld := Integral_Gain;
PI(PI_01);

regulator_out := DERV_01.Out + PI_01.Out;
```

功能块


DERV_01
DERV
Derivative
In
Out
Gain
Feedforward_Gain
ADD_01
ADD
Add
SUB_01
SUB
Subtract
PI_01
PI
PI
SourceA
Dest
SourceB
In
Out
Kp
Wld
Speed_Reference
Speed_feedback
Proportional_Gain
Integral_Gain
regulator_out

高通滤波 (HPF)

HPF 指令可实现对输入频率中低于截止频率的部分进行衰减的滤波。

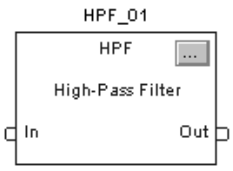
操作数：

结构化文本

```
HPF(HPF_tag);
```

操作数：	类型：	格式：	说明：
HPF 标记	FILTER_HIGH_PASS	结构	HPF 结构

功能块



操作数：	类型：	格式：	说明：
HPF 标记	FILTER_HIGH_PASS	结构	HPF 结构

FILER_HIGH_PASS 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果该位清零，则指令不执行且输出不更新。 如果该位置位，则指令执行。 默认状态为置位。 **结构化文本：** 无效。指令执行。
In	REAL	输入到指令的模拟信号。 有效值 = 浮点数 默认值 = 0.0
Initialize	BOOL	滤波控制算法初始化请求。置位时，指令设置 Out = In。 默认状态为清零。
WLead	REAL	超前频率 （单位为弧度 / 秒）。如果 WLead < 最小值或 WLead > 最大值，则指令置位状态字中的相应位并限制 WLead。 有效值 = 参见下面有关有效范围的说明 默认值 = 0.0
Order	REAL	滤波的阶数。阶数控制截断的锐度。如果 Order 无效，则指令置位状态字中的相应位并设置 Order = 1。 有效值 = 1 到 3 默认值 = 1

输入参数：	数据类型：	说明：
TimingMode	DINT	选择定时执行方式。 **值：** **说明：** 0 周期方式 1 过采样方式 2 实时采样方式 有关定时方式的更多信息，请参阅附录功能块属性。 有效值 = 0 到 2 默认值 = 0
OversampleDT	REAL	过采样方式下的运行时间。 有效值 = 0 到 4194.303 秒 默认值 = 0
RTSTime	DINT	实时采样方式下的模块更新周期 有效值 = 1 到 32,767 毫秒 默认值 = 1
RTSTimeStamp	DINT	实时采样方式下的模块时间戳值。 有效值 = 0 到 32,767 毫秒 默认值 = 0

输出参数：	数据类型：	说明：
EnableOut	BOOL	输出使能。
Out	REAL	算法计算出的输出。该输出影响算术状态标志。
DeltaT	REAL	两次更新之间的时间差值。通过控制算法计算过程输出时所用的时间（单位为秒）。
Status	DINT	功能块的状态。
InstructFault (Status.0)	BOOL	指令检测到如下运行错误。该错误不是次要或主要的控制器错误。检查其他状态位以确定错误类型。
WLeadInv (Status.1)	BOOL	WLead < 最小值或 WLead > 最大值。
OrderInv (Status.2)	BOOL	Order 值无效。
TimingModeInv (Status.27)	BOOL	TimingMode 值无效。 有关定时方式的更多信息，请参阅附录功能块属性。
RTSMissed (Status.28)	BOOL	仅用于实时采样方式。ABS | DeltaT - RTSTime | > 1 （0.001 秒）时置位。
RTSTimeInv (Status.29)	BOOL	RTSTime 值无效。
RTSTimeStampInv (Status.30)	BOOL	RTSTimeStamp 值无效。
DeltaTInv (Status.31)	BOOL	DeltaT 值无效。

说明： HPF 指令使用 Order 参数控制频率截止的锐度。 HPF 指令用于扫描频率为常数的任务。

HPF 指令使用以下公式：

条件：	指令使用下列传递函数：
Order = 1	$\frac{s}{s+\omega}$
Order = 2	$\frac{s^2}{s^2+\sqrt{2}\times s\times\omega+\omega^2}$
Order = 3	$\frac{s^3}{s^3+(2\times s^2\times\omega)+2\times s\times\omega^2+\omega^3}$

其中参数的极限值如下所示 （DeltaT 的单位为秒）：

参数：	极限值：
WLead 一阶 LowLimit	$\frac{0.0000001}{DeltaT}$
WLead 二阶 LowLimit	$\frac{0.00005}{DeltaT}$
WLead 三阶 LowLimit	$\frac{0.001}{DeltaT}$
HighLimit	$\frac{0.7\pi}{DeltaT}$

输出的计算值为无效、 NAN 或 ± INF 时，指令置位 Out 等于该无效值并置位算术溢出状态标志。当计算的输出值有效时，指令初始化内部参数并设置 Out = In。

算术状态标志： Out 输出影响算术状态标志。

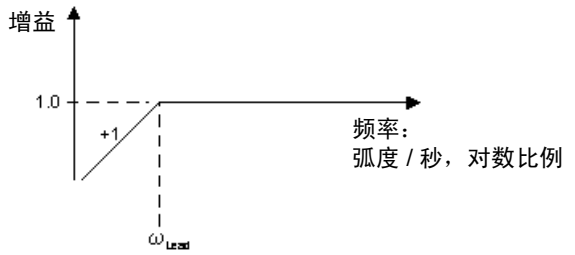
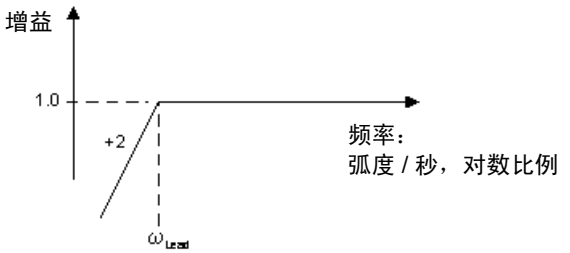
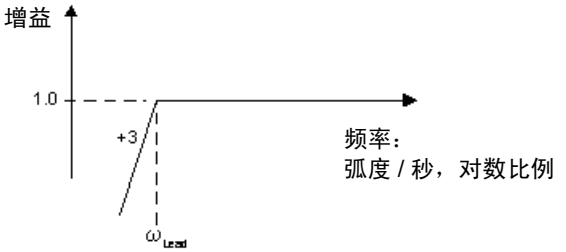
故障条件： 无

执行：

条件：	功能块动作：	结构化文本动作：
预扫描	无动作。	无动作。
指令首次扫描	无动作。	无动作。
指令首次执行	指令设置 Out = In。 控制算法不执行。	指令设置 Out = In。 控制算法不执行。
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。	na
EnableIn 置位	指令执行。 EnableOut 置位。	EnableIn 一直保持置位。 指令执行。
后期扫描	无动作。	无动作。

示例： HPF 指令可对低于设定的截止频率的信号进行衰减。该指令通常用于过滤电气源或机械源产生的低频 “噪声”或扰动。可以选择指定的滤波阶数以获取不同程度的衰减。注意，阶数越高则滤波指令的执行时间越长。

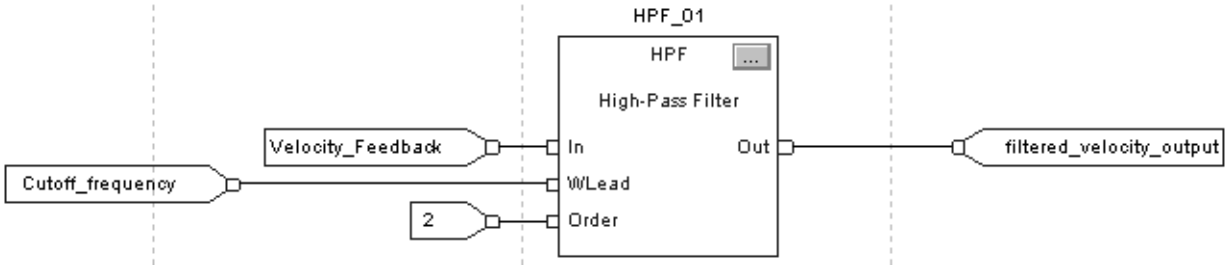
下图所示为给定截止频率的滤波器在不同阶数时的偏差。在图中，理想渐近曲线以增益系数和频率 （对数比例缩放）为坐标。滤波器的实际响应接近这些曲线，但并不与之完全重合。

滤波：	图形：
一阶滤波	
二阶滤波	
三阶滤波	

结构化文本

```
HPF_01.In := Velocity_Feedback;
HPF_01.WLead := Cutoff_frequency;
HPF_01.Order := 2;

HPF(HPF_01);

filtered_velocity_output := HPF_01.Out
```

功能块



二阶超前 / 滞后滤波 (LDL2)

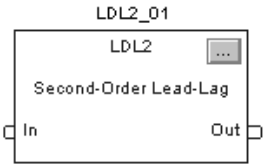
LDL2 指令可实现以极点对和零点对为参数进行滤波。极点和零点对的频率及阻尼可调。极点对或零点对既可以是复数 （阻尼比小于 1），也可以是实数 （阻尼比大于或等于 1）。

操作数：

```
LDL2(LDL2_tag);
```

结构化文本

操作数：	类型：	格式：	说明：
LDL2 标记	LEAD_LAG_SEC_ORDER	结构	LDL2 结构



功能块

操作数：	类型：	格式：	说明：
LDL2 标记	LEAD_LAG_SEC_ORDER	结构	LDL2 结构

LEAD_LAG_SEC_ORDER 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果该位清零，则指令不执行且输出不更新。 如果该位置位，则指令执行。 默认状态为置位。 **结构化文本：** 无效。指令执行。
In	REAL	输入到指令的模拟信号。 有效值 = 浮点数 默认值 = 0.0
Initialize	BOOL	滤波控制算法初始化请求。置位时，指令设置 Out = In。 默认状态为清零。
WLead	REAL	超前角频率，单位为弧度 / 秒。如果 WLead < 最小值或 WLead > 最大值，则指令置位状态字中的相应位并限制 WLead。如果 WLag:WLead > 最大比率，则指令置位状态字中的相应位并限制 WLag。 有效值参见下面有关有效范围的说明 默认值 = 0.0
WLag	REAL	滞后角频率，单位为弧度 / 秒。如果 WLag < 最小值或 WLag > 最大值，则指令置位状态字中的相应位并限制 WLag。如果 WLag:WLead > 最大比率，则指令置位状态字中的相应位并限制 WLag。 有效值参见下面有关有效范围的说明 默认值 = 0.0
ZetaLead	REAL	二阶超前阻尼因子。仅用于 Order = 2 时。如果 ZetaLead < 最小值或 ZetaLead > 最大值，指令将置位状态字中的相应位并限制 ZetaLead 值。 有效值 = 0.0 到 4.0 默认值 = 0.0

输入参数：	数据类型：	说明：
ZetaLag	REAL	二阶滞后阻尼因子。仅用于 Order = 2 时。如果 ZetaLag < 最小值或 ZetaLag > 最大值，指令将置位状态字中的相应位并限制 ZetaLag 值。 有效值 = 0.05 到 4.0 默认值 = 0.0
Order	REAL	滤波的阶数。选择一阶或二阶滤波算法。如果该值无效，指令将置位状态字中的相应位并使 Order = 2。 有效值 = 1 到 2 默认值 = 2
TimingMode	DINT	选择定时执行方式。 **值：** **说明：** 0 周期方式 1 过采样方式 2 实时采样方式 有关定时方式的更多信息，请参阅附录功能块属性。 有效值 = 0 到 2 默认值 = 0
OversampleDT	REAL	过采样方式下的运行时间。 有效值 = 0 到 4194.303 秒 默认值 = 0
RTSTime	DINT	实时采样方式下的模块更新周期 有效值 = 1 到 32,767 毫秒 默认值 = 1
RTSTimeStamp	DINT	实时采样方式下的模块时间戳值。 有效值 = 0 到 32,767 毫秒 默认值 = 0

输出参数：	数据类型：	说明：
EnableOut	BOOL	输出使能。
Out	REAL	算法计算出的输出。该输出影响算术状态标志。
DeltaT	REAL	两次更新之间的时间差值。通过控制算法计算过程输出时所用的时间（单位为秒）。
Status	DINT	功能块的状态。
InstructFault (Status.0)	BOOL	指令检测到如下运行错误。该错误不是次要或主要的控制器错误。检查其他状态位以确定错误类型。
WLeadInv (Status.1)	BOOL	WLead < 最小值或 WLead > 最大值。
WLagInv (Status.2)	BOOL	WLag < 最小值或 WLag > 最大值。
ZetaLeadInv (Status.3)	BOOL	超前阻尼因子 < 最小值或超前阻尼因子 > 最大值。
ZetaLagInv (Status.4)	BOOL	滞后阻尼因子 < 最小值或滞后阻尼因子 > 最大值。
OrderInv (Status.5)	BOOL	阶数值无效。

输出参数：	数据类型：	说明：
WLagRatioInv (Status.6)	BOOL	WLag:WLead 大于最大值。
TimingModeInv (Status.27)	BOOL	TimingMode 值无效。 有关定时方式的更多信息，请参阅附录功能块属性。
RTSMissed (Status.28)	BOOL	仅用于实时采样方式。ABS | DeltaT - RTSTime | > 1 （0.001 秒）时置位。
RTSTimeInv (Status.29)	BOOL	RTSTime 值无效。
RTSTimeStampInv (Status.30)	BOOL	RTSTimeStamp 值无效。
DeltaTInv (Status.31)	BOOL	DeltaT 值无效。

说明： LDL2 滤波指令用于强制参考和强制反馈控制方法。 LDL2 指令用于扫描频率为常数的任务。

LDL2 指令使用以下公式：

条件：	指令使用下列 Laplace 传递函数：
Order = 1	$$H(s) = \frac{\frac{s}{\omega_{Lead}} + 1}{\frac{s}{\omega_{Lag}} + 1}$$
Order = 2	$$H(s) = \frac{\frac{s^2}{\omega_{Lead}^2} + \frac{2 \times \xi_{Lead} \times s}{\omega_{Lead}} + 1}{\frac{s^2}{\omega_{Lag}^2} + \frac{2 \times \xi_{Lag} \times s}{\omega_{Lag}} + 1}$$ 标准化滤波器，使得 $\omega_{Lead} = 1$ $$H(s) = \frac{s^2 + 2 \times \xi_{Lead} \times s + 1}{\frac{s^2}{\omega_{Lag}^2} + \frac{2 \times \xi_{Lag} \times s}{\omega_{Lag}} + 1}$$

其中参数的极限值如下所示 （DeltaT 的单位为秒）：

参数：	极限值：
WLead 一阶 LowLimit	$\frac{0.0000001}{DeltaT}$
WLead 二阶 LowLimit	$\frac{0.00005}{DeltaT}$
HighLimit	$\frac{0.7\pi}{DeltaT}$
WLead:WLag 比率	如果 WLead > WLag，则无极限值
	如果 WLag > WLead： • WLag:WLead 无最小极限值 • 在一阶情况下， WLag:WLead 最大值 = 40:1，超出时指令会限制 WLag 以强制实施该比率 • 在二阶情况下， WLag:WLead 最大值 = 10:1，超出时指令会限制 WLag 以强制实施该比率
ZetaLead （仅用于二阶情况）	LowLimit = 0.0 HighLimit = 4.0
ZetaLag （仅用于二阶情况）	LowLimit = 0.05 HighLimit = 4.0

输出的计算值为无效、 NAN 或 ± INF 时，指令置位 Out 等于该无效值并置位算术溢出状态标志。当计算的输出值有效时，指令初始化内部参数并设置 Out = In。

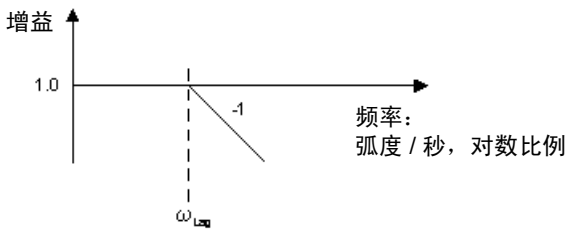
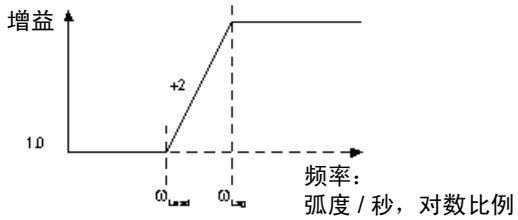
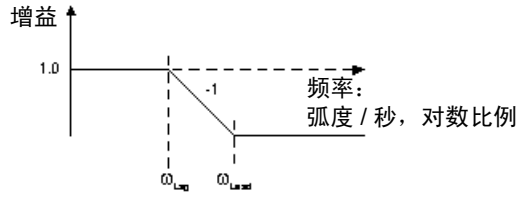
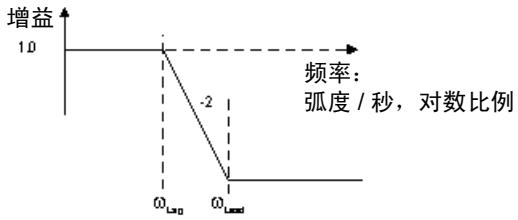
算术状态标志： Out 输出影响算术状态标志。

故障条件： 无

执行：

条件：	功能块动作：	结构化文本动作：
预扫描	无动作。	无动作。
指令首次扫描	无动作。	无动作。
指令首次执行	指令设置 Out = In。 控制算法不执行。	指令设置 Out = In。 控制算法不执行。
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。	na
EnableIn 置位	指令执行。 EnableOut 置位。	EnableIn 一直保持置位。 指令执行。
后期扫描	无动作。	无动作。

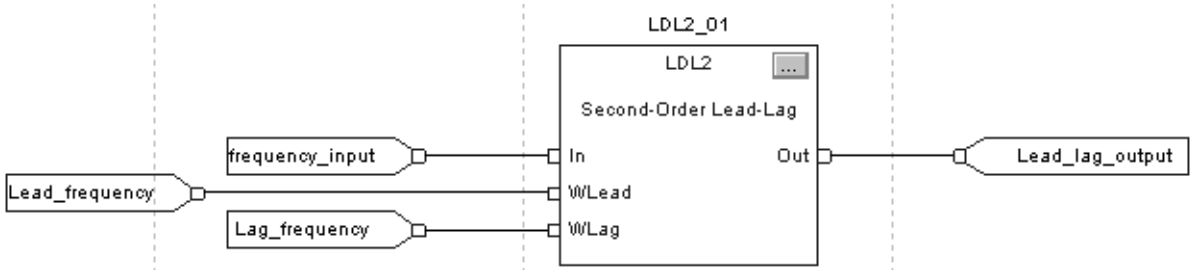
示例： LDL2 指令可对信号中处于两个频率之间的部分进行衰减或放大，具体操作取决于指令的配置。超前和滞后频率的相对大小可任意设置，因此根据二者设定的次序不同，该指令可分别用作超前 - 滞后块或滞后 - 超前块。注意，阶数越高则滤波指令的执行时间越长。

滤波：	图形：
一阶超前 - 滞后 ($\omega_{Lead} < \omega_{Lag}$)	
二阶超前 - 滞后 ($\omega_{Lead} < \omega_{Lag}$)	
一阶超前 - 滞后 ($\omega_{Lag} < \omega_{Lead}$)	
二阶超前 - 滞后 ($\omega_{Lag} < \omega_{Lead}$)	

结构化文本

```
LDL2_01.In := frequency_input;
LDL2_01.WLead := Lead_frequency;
LDL2_01.WLag := Lag_frequency;

LDL2(LDL2_01);

Lead_lag_output := LDL2_01.Out;
```

功能块



低通滤波 (LPF)

LPF 指令可实现对输入频率中高于截止频率的部分进行衰减的滤波。

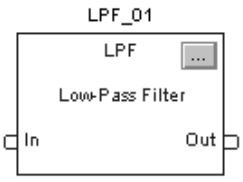
操作数：

结构化文本

```
LPF(LPF_tag);
```

操作数：	类型：	格式：	说明：
LPF 标记	FILTER_LOW_PASS	结构	LPF 结构

功能块



操作数：	类型：	格式：	说明：
LPF 标记	FILTER_LOW_PASS	结构	LPF 结构

FILTER_LOW_PASS 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果该位清零，则指令不执行且输出不更新。 如果该位置位，则指令执行。 默认状态为置位。 **结构化文本：** 无效。指令执行。
In	REAL	输入到指令的模拟信号。 有效值 = 浮点数 默认值 = 0.0
Initialize	BOOL	滤波控制算法初始化请求。置位时，指令设置 Out = In。 默认状态为清零。
WLag	REAL	滞后频率，单位为弧度 / 秒。如果 WLag < 最小值或 WLag > 最大值，则指令置位状态字中的相应位并限制 WLag。 有效值参见下面有关有效范围的说明 默认值 = 最大正浮点数
Order	REAL	滤波的阶数。阶数控制截断的锐度。如果 Order 无效，则指令置位状态字中的相应位并设置 Order = 1。 有效值 = 1 到 3 默认值 = 1

输入参数：	数据类型：	说明：
TimingMode	DINT	选择定时执行方式。 **值：** **说明：** 0 周期方式 1 过采样方式 2 实时采样方式 有关定时方式的更多信息，请参阅附录功能块属性。 有效值 = 0 到 2 默认值 = 0
OversampleDT	REAL	过采样方式下的运行时间。 有效值 = 0 到 4194.303 秒 默认值 = 0
RTSTime	DINT	实时采样方式下的模块更新周期 有效值 = 1 到 32,767 毫秒 默认值 = 1
RTSTimeStamp	DINT	实时采样方式下的模块时间戳值。 有效值 = 0 到 32,767 毫秒 默认值 = 0

输出参数：	数据类型：	说明：
EnableOut	BOOL	输出使能。
Out	REAL	算法计算出的输出。该输出影响算术状态标志。
DeltaT	REAL	两次更新之间的时间差值。通过控制算法计算过程输出时所用的时间（单位为秒）。
Status	DINT	功能块的状态。
InstructFault (Status.0)	BOOL	指令检测到如下运行错误。该错误不是次要或主要的控制器错误。检查其他状态位以确定错误类型。
WLagInv (Status.1)	BOOL	WLag < 最小值或 WLag > 最大值。
OrderInv (Status.2)	BOOL	阶数值无效。
TimingModeInv (Status.27)	BOOL	TimingMode 值无效。 有关定时方式的更多信息，请参阅附录功能块属性。
RTSMissed (Status.28)	BOOL	仅用于实时采样方式。ABS | DeltaT - RTSTime | > 1 （0.001 秒）时置位。
RTSTimeInv (Status.29)	BOOL	RTSTime 值无效。
RTSTimeStampInv (Status.30)	BOOL	RTSTimeStamp 值无效。
DeltaTInv (Status.31)	BOOL	DeltaT 值无效。

说明： LPF 指令使用 Order 参数控制频率截止的锐度。 LPF 指令用于扫描频率为常数的任务。

LPF 指令使用以下公式：

条件：	指令使用下列传递函数：
Order = 1	$\frac{\omega}{s+\omega}$
Order = 2	$\frac{\omega^2}{s^2+\sqrt{2}\times s\times\omega+\omega^2}$
Order = 3	$\frac{\omega^3}{s^3+(2\times s^2\times\omega)+(2\times s\times\omega^2)\times\omega^3}$

其中参数的极限值如下所示 （DeltaT 的单位为秒）：

参数：	极限值：
WLag 一阶 LowLimit	$\frac{0.0000001}{DeltaT}$
WLag 二阶 LowLimit	$\frac{0.00005}{DeltaT}$
WLag 三阶 LowLimit	$\frac{0.001}{DeltaT}$
HighLimit	$\frac{0.7\pi}{DeltaT}$

输出的计算值为无效、 NAN 或 ± INF 时，指令设置 Out 等于该无效值并置位算术溢出状态标志。当计算的输出值有效时，指令初始化内部参数并设置 Out = In。

算术状态标志： Out 输出影响算术状态标志。

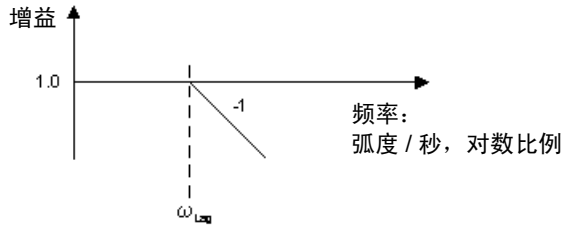
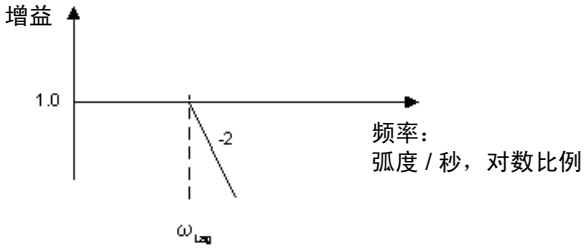
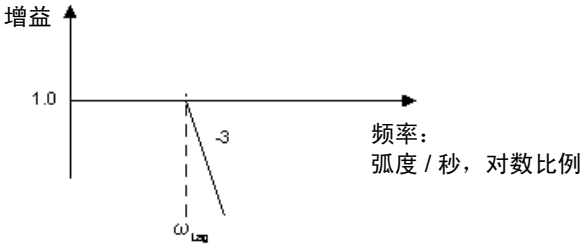
故障条件： 无

执行：

条件：	功能块动作：	结构化文本动作：
预扫描	无动作。	无动作。
指令首次扫描	无动作。	无动作。
指令首次执行	指令设置 Out = In。 控制算法不执行。	指令设置 Out = In。 控制算法不执行。
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。	na
EnableIn 置位	指令执行。 EnableOut 置位。	EnableIn 一直保持置位。 指令执行。
后期扫描	无动作。	无动作。

示例： LPF 指令可对高于设定的截止频率的信号进行衰减。该指令通常用于过滤电气源或机械源产生的高频 “噪声”或扰动。可以选择指定的滤波阶数以获取不同程度的衰减。注意，阶数越高则指令的执行时间越长。

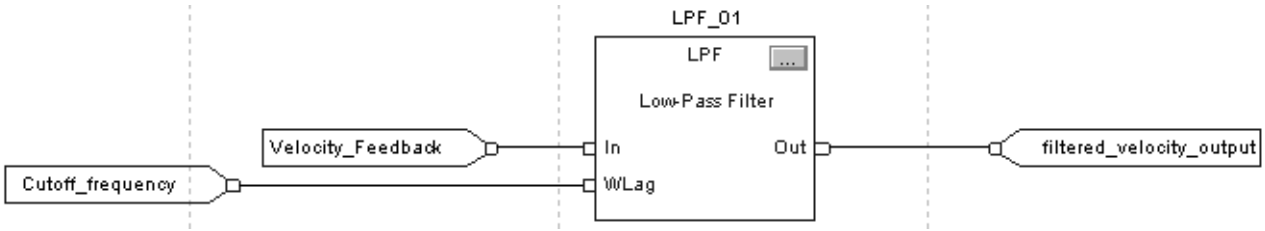
下图所示为给定截止频率的滤波器在不同阶数时的偏差。在图中，理想渐近曲线以增益系数和频率 （对数比例缩放）为坐标。滤波器的实际响应接近这些曲线，但并不与之完全重合。

滤波：	图形：
一阶滤波	
二阶滤波	
三阶滤波	

结构化文本

```
LPF_01.In := Velocity_Feedback;
LPF_01.WLag := Cutoff_frequency;

LPF(LPF_01);

filtered_velocity_output := LPF_01.Out
```

功能块



陷波滤波 (NTCH)

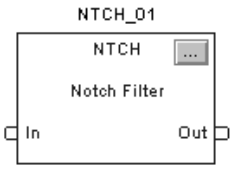
NTCH 指令可实现对输入频率中等于陷波频率的部分进行衰减的滤波。

操作数：

```
NTCH(NTCH_tag);
```

结构化文本

操作数：	类型：	格式：	说明：
NTCH 标记	FILTER_NOTCH	结构	NTCH 结构



功能块

操作数：	类型：	格式：	说明：
NTCH 标记	FILTER_NOTCH	结构	NTCH 结构

FILTER_NOTCH 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果该位清零，则指令不执行且输出不更新。 如果该位置位，则指令执行。 默认状态为置位。 **结构化文本：** 无效。指令执行。
In	REAL	输入到指令的模拟信号。 有效值 = 浮点数 默认值 = 0.0
Initialize	BOOL	滤波控制算法初始化请求。置位时，指令设置 Out = In。 默认状态为清零。
WNotch	REAL	滤波中心频率，单位为弧度 / 秒。如果 WNotch < 最小值或 WNotch > 最大值，指令置位状态字中的相应位并限制 WNotch 值。 有效值参见下面有关有效范围的说明 默认值 = 最大正浮点数
QFactor	REAL	控制宽度和深度比率。设置 QFactor = 1 / （2* 期望的阻尼因子）。如果 QFactor < 最小值或 QFactor > 最大值，指令置位状态字中的相应位并限制 QFactor 值。 有效值 = 0.5 到 100.0 默认值 = 0.5
Order	REAL	滤波的阶数。阶数控制截断的锐度。如果 Order 无效，则指令置位状态字中的相应位并设置 Order = 2。 有效值 = 2 或 4 默认值 = 2

输入参数：	数据类型：	说明：
TimingMode	DINT	选择定时执行方式。 **值：** **说明：** 0 周期方式 1 过采样方式 2 实时采样方式 有关定时方式的更多信息，请参阅附录功能块属性。 有效值 = 0 到 2 默认值 = 0
OversampleDT	REAL	过采样方式下的运行时间。 有效值 = 0 到 4194.303 秒 默认值 = 0
RTSTime	DINT	实时采样方式下的模块更新周期 有效值 = 1 到 32,767 毫秒 默认值 = 1
RTSTimeStamp	DINT	实时采样方式下的模块时间戳值。 有效值 = 0 到 32,767 毫秒 默认值 = 0

输出参数：	数据类型：	说明：
EnableOut	BOOL	输出使能。
Out	REAL	算法计算出的输出。该输出影响算术状态标志。
DeltaT	REAL	两次更新之间的时间差值。通过控制算法计算过程输出时所用的时间（单位为秒）。
Status	DINT	功能块的状态。
InstructFault (Status.0)	BOOL	指令检测到如下运行错误。该错误不是次要或主要的控制器错误。检查其他状态位以确定错误类型。
WNotchInv (Status.1)	BOOL	WNotch < 最小值或 WNotch > 最大值。
QFactorInv (Status.2)	BOOL	QFactor < 最小值或 QFactor > 最大值。
OrderInv (Status.3)	BOOL	阶数值无效。
TimingModeInv (Status.27)	BOOL	TimingMode 值无效。 有关定时方式的更多信息，请参阅附录功能块属性。
RTSMissed (Status.28)	BOOL	仅用于实时采样方式。ABS | DeltaT - RTSTime | > 1 （0.001 秒）时置位。
RTSTimeInv (Status.29)	BOOL	RTSTime 值无效。
RTSTimeStampInv (Status.30)	BOOL	RTSTimeStamp 值无效。
DeltaTInv (Status.31)	BOOL	DeltaT 值无效。

说明： NTCH 指令使用 Order 参数控制频率截止的锐度。QFactor 参数控制陷波的宽度和深度比率。 NTCH 指令用于扫描频率为常数的任务。

NTCH 指令使用以下公式：

$$\frac{(s^2+\omega^2)^i}{\left(s^2+s\times\frac{\omega}{Q}+\omega^2\right)^i}$$

其中 i 是 Order 操作数，参数的极限值如下所示 （DeltaT 的单位为秒）：

参数：	极限值：
WNotch 二阶 LowLimit	$\frac{0.0000001}{DeltaT}$
WNotch 四阶 LowLimit	$\frac{0.001}{DeltaT}$
HighLimit	$\frac{0.7\pi}{DeltaT}$
QFactor	LowLimit = 0.5 HighLimit = 100.0

输出的计算值为无效、 NAN 或 ± INF 时，指令设置 Out 等于该无效值并置位算术溢出状态标志。当计算的输出值有效时，指令初始化内部参数并设置 Out = In。

算术状态标志： Out 输出影响算术状态标志。

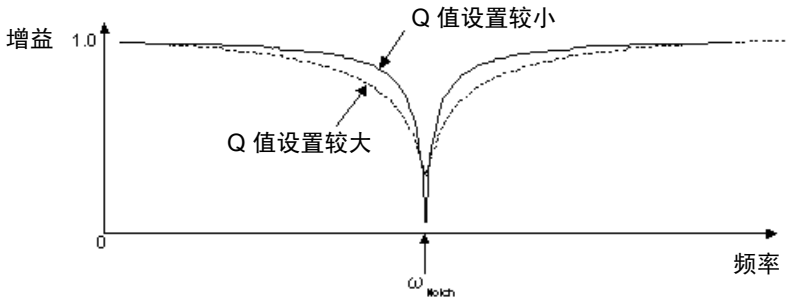
故障条件： 无

执行：

条件：	功能块动作：	结构化文本动作：
预扫描	无动作。	无动作。
指令首次扫描	无动作。	无动作。
指令首次执行	指令设置 Out = In。 控制算法不执行。	指令设置 Out = In。 控制算法不执行。
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。	na
EnableIn 置位	指令执行。 EnableOut 置位。	EnableIn 一直保持置位。 指令执行。
后期扫描	无动作。	无动作。

示例： NTCH 指令可衰减特定的谐振频率。通常，这些谐振频率处于闭环控制系统所调节的响应范围内。它们大多是由于松散的机械联动装置在系统中引起反冲和振动而产生的。尽管最佳的解决方案是校正机器的机械顺从性，但陷波滤波可用来缓解闭环调节方案中谐振频率信号产生的影响。

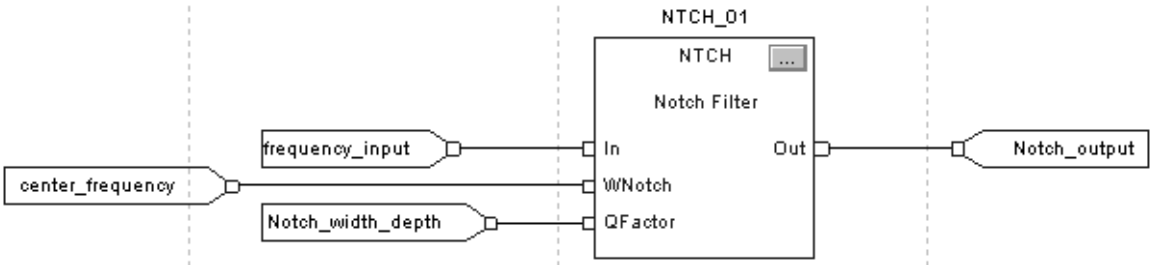
下图所示为指定中心频率和 Q 因子时在某一频率范围中的理想增益曲线。随着 Q 值增大，陷波变得更宽和更浅。Q 值减小，陷波也随之变得更窄且更深。该指令可以设置为 2 阶或 4 阶。阶数越高所需执行时间越长。



结构化文本

```
NTCH_01.In := frequency_input;
NTCH_01.WNotch := center_frequency;
NTCH_01.QFactor := Notch_width_depth;
NTCH(NTCH_01);

Notch_output := NTCH_01.Out;
```

功能块



选择 / 限制指令

（ESEL、 HLL、 MUX、 RLIM、 SEL、 SNEG、 SSUM）

简介

可以使用以下选择 / 限制指令：

如要：	所用指令：	可用编程语言：	参见页：
从最多六个输入中选择一个。	增强型选择 (ESEL)	结构化文本 功能块	4-2
将一个模拟输入限制在两个值之间。	上 / 下限限幅 (HLL)	结构化文本 功能块	4-9
从八个输入中选择一个。	多路切换器 (MUX)	功能块	4-12
限制信号随时间的变化量。	速率限制器 (RLIM)	结构化文本 功能块	4-15
从两个输入中选择一个。	选择 (SEL)	功能块	4-19
在输入值与输入值的反值之间选择。	选择取反 (SNEG)	结构化文本 功能块	4-21
选择要求和的实数输入。	选择求和 (SSUM)	结构化文本 功能块	4-23

增强型选择 (ESEL)

ESEL 指令选择最多六个输入中的一个。选项包括：

- 手动选择 （由操作员或程序）
- 选择最大值
- 选择最小值
- 选择中间值
- 选择平均值

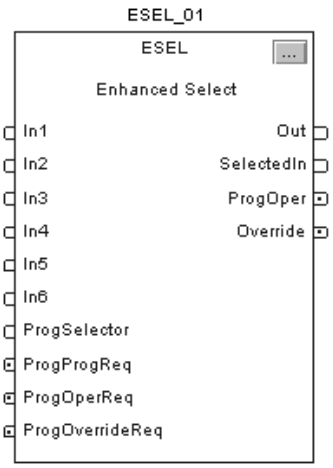
操作数：

结构化文本

```
ESEL(ESEL_tag);
```

操作数：	类型：	格式：	说明：
ESEL 标记	SELECT_ENHANCED	结构	ESEL 结构

功能块



操作数：	类型：	格式：	说明：
ESEL 标记	SELECT_ENHANCED	结构	ESEL 结构

SELECT_ENHANCED 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果该位清零，则指令不执行且输出不更新。 如果该位置位，则指令执行。 默认状态为置位。 **结构化文本：** 无效。指令执行。
In1	REAL	指令的模拟信号输入 1。 有效值 = 浮点数 默认值 = 0.0
In2	REAL	指令的模拟信号输入 2。 有效值 = 浮点数 默认值 = 0.0
In3	REAL	指令的模拟信号输入 3。 有效值 = 浮点数 默认值 = 0.0

输入参数：	数据类型：	说明：
In4	REAL	指令的模拟信号输入 4。 有效值 = 浮点数 默认值 = 0.0
In5	REAL	指令的模拟信号输入 5。 有效值 = 浮点数 默认值 = 0.0
In6	REAL	指令的模拟信号输入 6。 有效值 = 浮点数 默认值 = 0.0
In1Fault	BOOL	In1 错误状态指示。如果 In1 读取自模拟输入，则 In1Fault 一般是由模拟输入上的故障状态控制。如果所有 In_nFault 输入均置位，则指令置位状态字中的相应位，控制算法不执行，且 Out 不更新 默认状态为清零。
In2Fault	BOOL	In2 错误状态指示。如果 In2 读取自模拟输入，则 In2Fault 一般是由模拟输入上的故障状态控制。如果所有 In_nFault 输入均置位，则指令置位状态字中的相应位，控制算法不执行，且 Out 不更新 默认状态为清零。
In3Fault	BOOL	In3 错误状态指示。如果 In3 读取自模拟输入，则 In3Fault 一般是由模拟输入上的故障状态控制。如果所有 In_nFault 输入均置位，则指令置位状态字中的相应位，控制算法不执行，且 Out 不更新 默认状态为清零。
In4Fault	BOOL	In4 错误状态指示。如果 In4 读取自模拟输入，则 In4Fault 一般是由模拟输入上的故障状态控制。如果所有 In_nFault 输入均置位，则指令置位状态字中的相应位，控制算法不执行，且 Out 不更新 默认状态为清零。
In5Fault	BOOL	In5 错误状态指示。如果 In5 读取自模拟输入，则 In5Fault 一般是由模拟输入上的故障状态控制。如果所有 In_nFault 输入均置位，则指令置位状态字中的相应位，控制算法不执行，且 Out 不更新 默认状态为清零。
In6Fault	BOOL	In6 错误状态指示。如果 In6 读取自模拟输入，则 In6Fault 一般是由模拟输入上的故障状态控制。如果所有 In_nFault 输入均置位，则指令置位状态字中的相应位，控制算法不执行，且 Out 不更新 默认状态为清零。
InsUsed	DINT	所用的输入数。定义指令使用的输入数。在最大、最小、中间和平均值选择方式中，指令只考虑 In1 到 $In_{InsUsed}$。如果该值无效，则指令置位状态字中的相应位。如果 InsUsed 无效且指令未使用手动选择方式，则当 Override 清零时，指令将不更新 Out。 有效值 = 1 到 6 默认值 = 1

输入参数：	数据类型：	说明：
SelectorMode	DINT	选择器方式输入。该值决定指令的动作。 **值：** **说明：** 0 手动选择 1 选择最大值 2 选择最小值 3 选择中间值 4 选择平均值 如果该值无效，指令将置位状态字中的相应位，且不更新 Out。 有效值 = 0 到 4 默认值 = 0
ProgSelector	DINT	程序选择器输入。当选择器方式为手动选择且指令处于程序控制方式时，ProgSelector 决定哪个输入 (In1-In6) 移入 Out。如果 ProgSelector = 0，指令将不更新 Out。如果 ProgSelector 无效，指令将置位状态字中的相应位。如果该值无效且指令处于程序控制方式，当选择器方式为手动选择或 Override 置位时，指令将不更新 Out。 有效值 = 0 到 6 默认值 = 0
OperSelector	DINT	操作员选择器输入。当选择器方式为手动选择且指令处于操作员控制方式时，OperSelector 决定哪个输入 (In1-In6) 移入 Out。如果 OperSelector = 0，指令将不更新 Out。如果 OperSelector 无效，指令将置位状态字中的相应位。如果该值无效且指令处于 Operator 控制方式，当选择器方式为手动选择或 Override 置位时，指令将不更新 Out。 有效值 = 0 到 6 默认值 = 0
ProgProgReq	BOOL	程序请求进入程序控制方式。通过用户程序设置来请求程序控制。如果 ProgOperReq 置位，该请求被忽略。保持置位状态并将 ProgOperReq 清零，可锁定指令进入程序控制状态。 默认状态为清零。
ProgOperReq	BOOL	程序请求进入操作员控制方式。通过用户程序设置来请求操作员控制。保持置位状态可锁定指令进入操作员控制状态。 默认状态为清零。
ProgOverrideReq	BOOL	程序请求进入自动方式。通过用户程序设置来请求设备进入自动方式。如果 ProgOper 清零将忽略请求。在 Override 方式下，指令将以手动选择方式工作。 默认状态为清零。
OperProgReq	BOOL	操作员请求进入程序控制方式。通过操作员界面设置来请求程序控制。指令将该输入清零。 默认状态为清零。
OperOperReq	BOOL	操作员请求进入操作员控制方式。通过操作员界面设置来请求操作员控制。指令将该输入清零。 默认状态为清零。
ProgValueReset	BOOL	复位程序控制值。置位时，每次执行指令后将所有的程序请求输入清零。 默认状态为清零。

输出参数：	数据类型：	说明：
EnableOut	BOOL	输出使能。
Out	REAL	算法计算出的输出。该输出影响算术状态标志。
SelectedIn	DINT	选择的输入号。指令使用该值显示当前被置入输出的输入号。如果选择器方式为选择平均值，指令将设置 SelectedIn = 0。
ProgOper	BOOL	程序 / 操作员控制指示。处于程序控制状态时置位。处于操作员控制时清零。
Override	BOOL	自动方式。当指令处于 Override 方式时置位。
Status	DINT	功能块的状态。
InstructFault (Status.0)	BOOL	指令检测到如下运行错误。该错误不是次要或主要的控制器错误。检查其他状态位以确定错误类型。
InsFaulted (Status.1)	BOOL	所有已用 In_n 输入的 In_nFault 输入置位。
InsUsedInv (Status.2)	BOOL	InsUsed 值无效。
SelectorModeInv (Status.3)	BOOL	SelectorMode 值无效。
ProgSelectorInv (Status.4)	BOOL	ProgSelector 值无效。
OperSelectorInv (Status.5)	BOOL	OperSelector 值无效。

说明： ESEL 指令的操作如下：

条件：	动作：
SelectorMode = 0 （手动选择）**或** Override 置位， ProgOper 清零，且 OperSelector ≠ 0	Out = In[OperSelector] SelectedIn = OperSelector
SelectorMode = 0 （手动选择）**或** Override 置位， ProgOper 置位，且 ProgSelector ≠ 0	Out = In[ProgSelector] SelectedIn = ProgSelector
SelectorMode = 1 （选择最大值）**且** Override 清零	Out = In[InsUsed] 的最大值 SelectedIn = 最大输入值的索引
SelectorMode = 2 （选择最小值）**且** Override 清零	Out = In[InsUsed] 的最小值 SelectedIn = 最小输入值的索引
SelectorMode = 3 （选择中间值）**且** Override 清零	Out = In[InsUsed] 的中值 SelectedIn = 中间输入值的索引
SelectorMode = 4 （选择平均值）**且** Override 清零	Out = In[InsUsed] 的平均值 SelectedIn = 0

对于 SelectorMode 1 至 4，任何输入的错误指示将导致该输入在选择中不被考虑。例如，如果 SelectorMode = 1 （选择最大值），且 In6 具有最大值但状态错误，则下一个具有正确状态的最大输入将被置入输出。

对于选择最大值或选择最小值方式，如果两个输入相等且为最大值或最小值，指令将输出最先找到的输入。对于选择中间值方式，中值始终代表选自可用输入的一个值。如果多个值均为中值，指令将输出最先找到的输入。

监控 ESEL 指令

ESEL 指令具有操作员面板。有关更多信息，请参阅附录功能块面板控件。

算术状态标志： Out 输出影响算术状态标志。

故障条件： 无

执行：

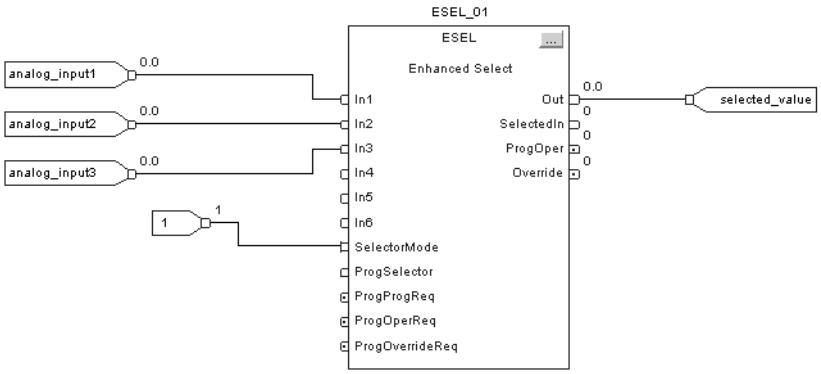
条件：	功能块动作：	结构化文本动作：
预扫描	无动作。	无动作。
指令首次扫描	所有操作员请求输入均清零。 如果 ProgValueReset 置位，则所有程序请求输入均清零。	所有操作员请求输入均清零。 如果 ProgValueReset 置位，则所有程序请求输入均清零。
指令首次执行	将指令设置为操作员控制方式。	将指令设置为操作员控制方式。
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。	na
EnableIn 置位	指令执行。 EnableOut 置位。	EnableIn 一直保持置位。 指令执行。
后期扫描	无动作。	无动作。

示例： 该 ESEL 指令根据 SelectorMode 选择 In1、 In2 或 In3。本例中，SelectorMode = 1，即选择最大值。指令确定哪个输入值最大，然后设置输出 = 最大输入值。

结构化文本

```
ESEL_01.In1 := analog_input1;
ESEL_01.In2 := analog_input2;
ESEL_01.In3 := analog_input3;
ESEL_01.SelectorMode := 1;
ESEL(ESEL_01);

selected_value := ESEL_01.Out;
```

功能块



在程序控制和操作员控制之间切换

下图显示 ESEL 指令如何在程序控制和操作员控制之间切换。

(1) 保持 ProgOperReq 为置位状态，可将指令锁定为操作员控制方式。

(2) 在 ProgOperReq 清零时保持 ProgProgReq 为置位状态，可以将指令锁定为程序控制方式。

有关程序和操作员控制的更多信息，请参阅第 B-13 页。

上 / 下限限幅 (HLL)

HLL 指令将一个模拟输入限制在两个值之间。可以选择上 / 下限、上限或下限。

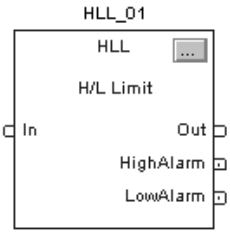
操作数：

结构化文本

```
HLL(HLL_tag);
```

操作数：	类型：	格式：	说明：
HLL 标记	HL_LIMIT	结构	HLL 结构

功能块



操作数：	类型：	格式：	说明：
HLL 标记	HL_LIMIT	结构	HLL 结构

HL_LIMIT 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果该位清零，则指令不执行且输出不更新。 如果该位置位，则指令执行。 默认状态为置位。 **结构化文本：** 无效。指令执行。
In	REAL	输入到指令的模拟信号。 有效值 = 浮点数 默认值 = 0.0
HighLimit	REAL	输入的上限。如果 HighLimit ≤ LowLimit，则指令置位状态字中的相应位并设置 Out = LowLimit。 有效值 = HighLimit > LowLimit 默认值 = 最大正浮点数
LowLimit	REAL	输入的下限。如果 HighLimit ≤ LowLimit，则指令置位状态字中的相应位并设置 Out = LowLimit。 有效值 = LowLimit < HighLimit 默认值 = 最大负浮点数
SelectLimit	DINT	选择输入限幅方式。该输入有三个设置： **值：** **说明：** 0 使用上限和下限 1 使用上限 2 使用下限 如果 SelectLimit 无效，指令将假定 SelectLimit = 0 且置位状态字中的相应位。 有效值 = 0 到 2 默认值 = 0

输出参数：	数据类型：	说明：
EnableOut	BOOL	输出使能。
Out	REAL	算法计算出的输出。该输出影响算术状态标志。
HighAlarm	BOOL	上限报警指示。 In ≥ HighLimit 时置位。
LowAlarm	BOOL	下限报警指示。In ≤ LowLimit 时置位。
Status	DINT	功能块的状态。
InstructFault (Status.0)	BOOL	指令检测到如下运行错误。该错误不是次要或主要的控制器错误。检查其他状态位以确定错误类型。
LimitsInv (Status.1)	BOOL	HighLimit ≤ LowLimit。
SelectLimitInv (Status.2)	BOOL	SelectLimit 的值不是 0、 1 或 2。

说明： HLL 指令使用以下规则决定 Out 的值：

选择：	条件：	动作：
SelectLimit = 0 （使用上限和下限）	In < HighLimit 且 In > LowLimit	Out = In
	In ≥ HighLimit	Out = HighLimit HighAlarm 置位
	In ≤ LowLimit	Out = LowLimit LowAlarm 置位
	HighLimit ≤ LowLimit	Out = LowLimit HighAlarm 置位 LowAlarm 置位 LimitsInv 置位
SelectLimit = 1 （只使用上限）	In < HighLimit	Out = In
	In ≥ HighLimit	Out = HighLimit HighAlarm 置位
SelectLimit = 2 （只使用下限）	In > LowLimit	Out = In
	In ≤ LowLimit	Out = LowLimit LowAlarm 置位

算术状态标志： Out 输出影响算术状态标志。

故障条件： 无

执行：

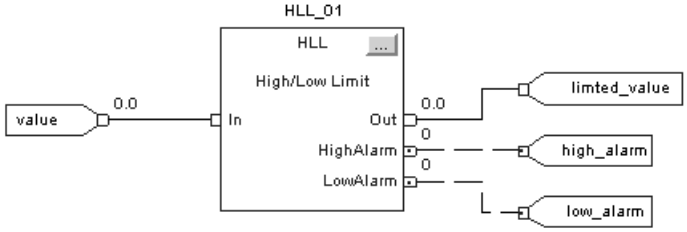
条件：	功能块动作：	结构化文本动作：
预扫描	无动作。	无动作。
指令首次扫描	无动作。	无动作。
指令首次执行	无动作。	无动作。
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。	na
EnableIn 置位	指令执行。 EnableOut 置位。	EnableIn 一直保持置位。 指令执行。
后期扫描	无动作。	无动作。

示例： 该 HLL 指令将输入限制在两个值之间，如果需要，可以设置 HighAlarm 或 LowAlarm。当输入在限幅以外时，指令设置输出等于输入的限值。

结构化文本

```
HLL_01.In := value;
HLL(HLL_01);

limited_value := HLL_01.Out;
high_alarm := HLL_01.HighAlarm;
low_alarm := HLL_01.LowAlarm;
```

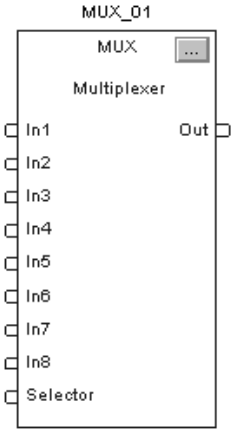
功能块



多路切换器 (MUX)

MUX 指令根据选择器输入选择八个输入中的一个。

操作数：



功能块

操作数：	类型：	格式：	说明：
块标记	MULTIPLEXER	结构	MUX 结构

MULTIPLEXER 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	启用输入。如果该位清零，则指令不执行且输出不更新。 默认状态为置位。
In1	REAL	指令的模拟信号输入 1。 有效值 = 浮点数 默认值 = 0.0
In2	REAL	指令的模拟信号输入 2。 有效值 = 浮点数 默认值 = 0.0
In3	REAL	指令的模拟信号输入 3。 有效值 = 浮点数 默认值 = 0.0
In4	REAL	指令的模拟信号输入 4。 有效值 = 浮点数 默认值 = 0.0
In5	REAL	指令的模拟信号输入 5。 有效值 = 浮点数 默认值 = 0.0
In6	REAL	指令的模拟信号输入 6。 有效值 = 浮点数 默认值 = 0.0

输入参数：	数据类型：	说明：
In7	REAL	指令的模拟信号输入 7。 有效值 = 浮点数 默认值 = 0.0
In8	REAL	指令的模拟信号输入 8。 有效值 = 浮点数 默认值 = 0.0
Selector	DINT	指令的选择器输入。该输入决定哪个输入 (1-8) 移入 Out。如果该值无效（包括 0），指令将置位状态字中的相应位，并使 Out 保持其当前值。 有效值 = 1 到 8 默认值 = 0

输出参数：	数据类型：	说明：
EnableOut	BOOL	输出使能。
Out	REAL	算法的选择输出。该输出影响算术状态标志。
Status	DINT	功能块的状态。
InstructFault (Status.0)	BOOL	指令检测到如下运行错误。该错误不是次要或主要的控制器错误。检查其他状态位以确定错误类型。
SelectorInv (Status.1)	BOOL	Selector 值无效。

说明： 根据 Selector 值， MUX 指令设置 Out 等于八个输入中的一个。

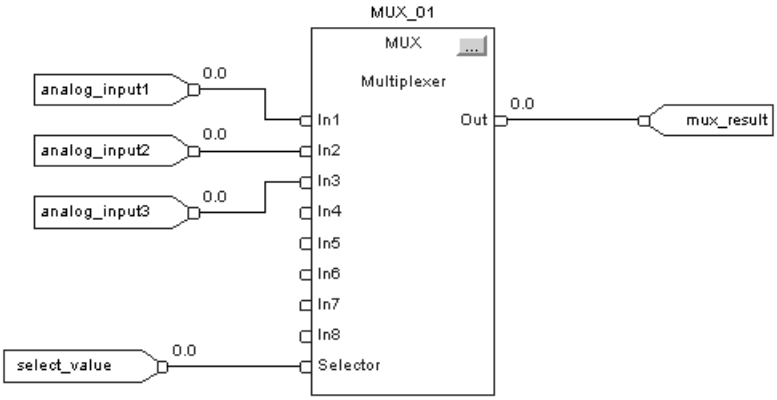
算术状态标志： Out 输出影响算术状态标志。

故障条件： 无

执行：

条件：	功能块动作：
预扫描	无动作。
指令首次扫描	内部参数清零。
指令首次执行	无动作。
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。
EnableIn 置位	指令执行。 EnableOut 置位。
后期扫描	无动作。

示例： 该 MUX 指令根据 Selector 选择 In1、 In2 或 In3。指令设置 Out = In_n。例如，如果 select_value = 2，指令将设置 Out = analog_input2。



速率限幅器 (RLIM)

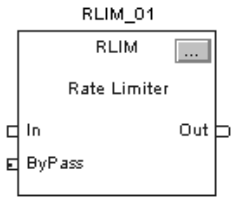
RLIM 指令限制信号随时间的变化量。

操作数：

```
RLIM(RLIM_tag);
```

结构化文本

操作数：	类型：	格式：	说明：
RLIM 标记	RATE_LIMITER	结构	RLIM 结构



功能块

操作数：	类型：	格式：	说明：
RLIM 标记	RATE_LIMITER	结构	RLIM 结构

RATE_LIMITER 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果该位清零，则指令不执行且输出不更新。 如果该位置位，则指令执行。 默认状态为置位。 **结构化文本：** 无效。指令执行。
In	REAL	输入到指令的模拟信号。 有效值 = 浮点数 默认值 = 0.0
IncRate	REAL	最大输出每秒的递增量。如果无效，指令将设置 IncRate = 0.0 且置位状态字中的相应位。 有效值 = 任意浮点数 ≥ 0.0 默认值 = 0.0
DecRate	REAL	最大输出每秒的递减量。如果无效，指令将设置 DecRate = 0.0 且置位状态字中的相应位。 有效值 = 任意浮点数 ≥ 0.0 默认值 = 0.0
ByPass	BOOL	算法的旁路请求。置位时， Out = In。 默认状态为清零。
TimingMode	DINT	选择定时执行方式。 **值：** **说明：** 0 周期方式 1 过采样方式 2 实时采样方式 有关定时方式的更多信息，请参阅附录功能块属性。 有效值 = 0 到 2 默认值 = 0

输入参数：	数据类型：	说明：
OversampleDT	REAL	过采样方式下的运行时间。 有效值 = 0 到 4194.303 秒 默认值 = 0
RTSTime	DINT	实时采样方式下的模块更新周期 有效值 = 1 到 32,767 毫秒 默认值 = 1
RTSTimeStamp	DINT	实时采样方式下的模块时间戳值。 有效值 = 0 到 32,767 毫秒 默认值 = 0

输出参数：	数据类型：	说明：
EnableOut	BOOL	输出使能。
Out	REAL	算法计算出的输出。该输出影响算术状态标志。
DeltaT	REAL	两次更新之间的时间差值。通过控制算法计算过程输出时所用的时间（单位为秒）。
Status	DINT	功能块的状态。
InstructFault (Status.0)	BOOL	指令检测到如下运行错误。该错误不是次要或主要的控制器错误。检查其他状态位以确定错误类型。
IncRateInv (Status.1)	BOOL	IncRate < 0。指令使用 0。
DecRate (Status.2)	BOOL	DecRate < 0。指令使用 0。
TimingModeInv (Status.27)	BOOL	TimingMode 值无效。 有关定时方式的更多信息，请参阅附录功能块属性。
RTSMissed (Status.28)	BOOL	仅用于实时采样方式。 ABS | DeltaT - RTSTime | > 1 （0.001 秒）时置位。
RTSTimeInv (Status.29)	BOOL	RTSTime 值无效。
RTSTimeStampInv (Status.30)	BOOL	RTSTimeStamp 值无效。
DeltaTInv (Status.31)	BOOL	DeltaT 值无效。

说明： RLIM 指令提供单独的每秒递增速率和递减速率。利用 ByPass 输入可以停止速率限幅，将信号直接传递到输出。

条件：	动作：
ByPass 置位	$Out_n = In_n$ $Out_{n-1} = In_n$
ByPass 清零且 DeltaT > 0	$Slope = \frac{In_n - Out_{n-1}}{DeltaT}$ 如果 Slope ≤ 蠕 ecRate，则 YSlope = 蠕 ecRate 如果 蠕 ecRate ≤ Slope ≤ IncRate，则 YSlope = Slope 如果 IncRate ≤ Slope，则 YSlope = IncRate $Out_n = Out_{n-1}$ + DeltaT x YSlope $Out_{n-1} = Out_n$ DeltaT 的单位为秒

算术状态标志： Out 输出影响算术状态标志。

故障条件： 无

执行：

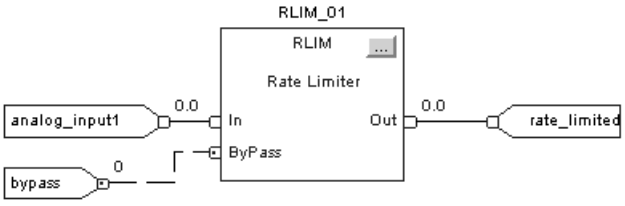
条件：	功能块动作：	结构化文本动作：
预扫描	无动作。	无动作。
指令首次扫描	$Out_{n-1} = In_n$	$Out_{n-1} = In_n$
指令首次执行	$Out_{n-1} = In_n$	$Out_{n-1} = In_n$
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。	na
EnableIn 置位	指令执行。 EnableOut 置位。	EnableIn 一直保持置位。 指令执行。
后期扫描	无动作。	无动作。

示例： RLIM 指令通过 IncRate 限制输入。如果 analog_input1 以大于 IncRate 值的速率变化，指令将限制输入。指令设置输出等于输入的速率限制值。

结构化文本

```
RLIM_01.In := analog_input1;
RLIM_01.Bypass := bypass;
RLIM(RLIM_01);

rate_limited := RLIM_01.Out;
```

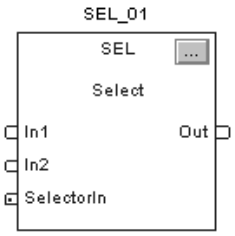
功能块



选择 (SEL)

SEL 指令使用数字输入选择两个输入中的一个。

操作数：



功能块

操作数：	类型：	格式：	说明：
SEL 标记	SELECT	结构	SEL 结构

SELECT 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	启用输入。如果该位清零，则指令不执行且输出不更新。 默认状态为置位。
In1	REAL	指令的模拟信号输入 1。 有效值 = 浮点数 默认值 = 0.0
In2	REAL	指令的模拟信号输入 2。 有效值 = 浮点数 默认值 = 0.0
SelectorIn	BOOL	在 In1 和 In2 之间选择一个输入。 默认状态为清零。
输出参数：	**数据类型：**	**说明：**
EnableOut	BOOL	输出使能。
Out	REAL	算法计算出的输出。该输出影响算术状态标志。

说明： SEL 指令的操作如下：

条件：	动作：
SelectorIn 置位	Out = In2
SelectorIn 清零	Out = In1

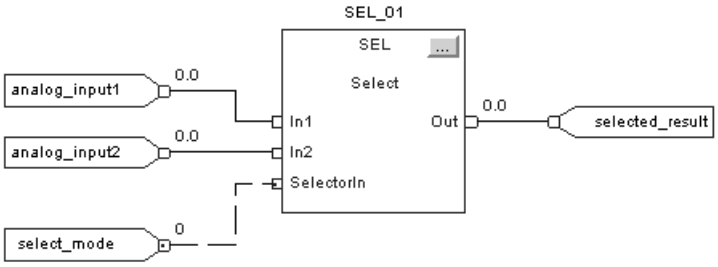
算术状态标志： Out 输出影响算术状态标志。

故障条件： 无

执行：

条件：	功能块动作：
预扫描	无动作。
指令首次扫描	无动作。
指令首次执行	无动作。
EnableIn 清零	EnableOut 清零。
EnableIn 置位	指令执行。 EnableOut 置位。
后期扫描	无动作。

示例： SEL 指令根据 SelectorIn 选择 In1 或 In2。如果 SelectorIn 置位，指令将设置 Out = In2。如果 SelectorIn 清零，指令将设置 Out = In1。



选择取反 (SNEG)

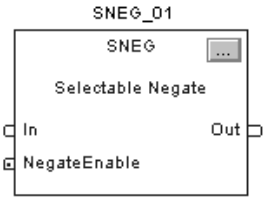
SNEG 指令使用数字输入在输入值与输入值的反值之间进行选择。

操作数：

```
SNEG(SNEG_tag);
```

结构化文本

操作数：	类型：	格式：	说明：
SNEG 标记	SELECTABLE_NEGATE	结构	SNEG 结构



功能块

操作数：	类型：	格式：	说明：
SNEG 标记	SELECTABLE_NEGATE	结构	SNEG 结构

SELECTABLE_NEGATE 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果该位清零，则指令不执行且输出不更新。 如果该位置位，则指令执行。 默认状态为置位。 **结构化文本：** 无效。指令执行。
In	REAL	输入到指令的模拟信号。
NegateEnable	BOOL	启用取反。如果 NegateEnable 置位，指令将设置 Out 为 In 的反值。 默认状态为置位。
输出参数：	**数据类型：**	**说明：**
EnableOut	BOOL	输出使能。
Out	REAL	算法计算出的输出。该输出影响算术状态标志。

说明： SNEG 指令的操作如下：

条件：	动作：
NegateEnable 置位	Out = 鑾 n
NegateEnable 清零	Out = In

算术状态标志： Out 输出影响算术状态标志。

故障条件： 无

执行：

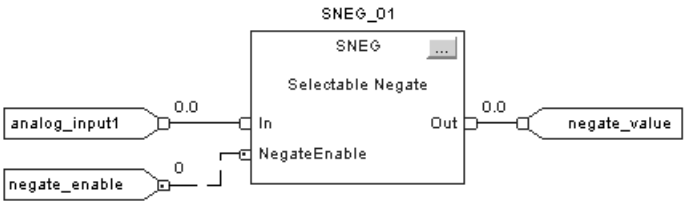
条件：	功能块动作：	结构化文本动作：
预扫描	无动作。	无动作。
指令首次扫描	无动作。	无动作。
指令首次执行	无动作。	无动作。
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。	na
EnableIn 置位	指令执行。 EnableOut 置位。	EnableIn 一直保持置位状态。 指令执行。
后期扫描	无动作。	无动作。

示例： 标记 negate_enable 决定是否对 In 取反。如果 NegateEnable 清零，指令将设置 Out = In。如果 NegateEnable 置位，指令将设置 Out = -In。

结构化文本

```
SNEG_01.In := analog_input1
SNEG_01.NegateEnable := negate_enable;
SNEG(SNEG_01);

negate_value := SNEG_01.Out;
```

功能块



选择求和 (SSUM)

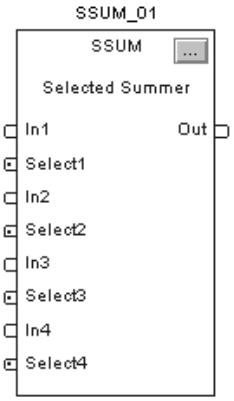
SSUM 指令使用布尔输入选择要进行代数求和的实数输入。

操作数：

```
SSUM(SSUM_tag);
```

结构化文本

操作数：	类型：	格式：	说明：
SSUM 标记	SELECTABLE_SUMMER	结构	SSUM 结构



功能块

操作数：	类型：	格式：	说明：
SSUM 标记	SELECTABLE_SUMMER	结构	SSUM 结构

SELECTABLE_SUMMER 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果该位清零，则指令不执行且输出不更新。 如果该位置位，则指令执行。 默认状态为置位。 **结构化文本：** 无效。指令执行。
In1	REAL	要求和的第一个输入。 有效值 = 浮点数 默认值 = 0.0
Gain1	REAL	第一个输入的增益。 有效值 = 浮点数 默认值 = 1.0
Select1	BOOL	第一个输入的 Selector 信号。 默认状态为清零。
In2	REAL	要求和的第二个输入。 有效值 = 浮点数 默认值 = 0.0
Gain2	REAL	第二个输入的增益。 有效值 = 浮点数 默认值 = 1.0
Select2	BOOL	第二个输入的 Selector 信号。 默认状态为清零。

输入参数：	数据类型：	说明：
In3	REAL	要求和的第三个输入。 有效值 = 浮点数 默认值 = 0.0
Gain3	REAL	第三个输入的增益。 有效值 = 浮点数 默认值 = 1.0
Select3	BOOL	第三个输入的 Selector 信号。 默认状态为清零。
In4	REAL	要求和的第四个输入。 有效值 = 浮点数 默认值 = 0.0
Gain4	REAL	第四个输入的增益。 有效值 = 浮点数 默认值 = 1.0
Select4	BOOL	第四个输入的 Selector 信号。 默认状态为清零。
In5	REAL	要求和的第五个输入。 有效值 = 浮点数 默认值 = 0.0
Gain5	REAL	第五个输入的增益。 有效值 = 浮点数 默认值 = 1.0
Select5	BOOL	第五个输入的 Selector 信号。 默认状态为清零。
In6	REAL	要求和的第六个输入。 有效值 = 浮点数 默认值 = 0.0
Gain6	REAL	第六个输入的增益。 有效值 = 浮点数 默认值 = 1.0
Select6	BOOL	第六个输入的 Selector 信号。 默认状态为清零。
In7	REAL	要求和的第七个输入。 有效值 = 浮点数 默认值 = 0.0
Gain7	REAL	第七个输入的增益。 有效值 = 浮点数 默认值 = 1.0
Select7	BOOL	第七个输入的 Selector 信号。 默认状态为清零。
In8	REAL	要求和的第八个输入。 有效值 = 浮点数 默认值 = 0.0

输入参数：	数据类型：	说明：
Gain8	REAL	第八个输入的增益。 有效值 = 浮点数 默认值 = 1.0
Select8	BOOL	第八个输入的 Selector 信号。 默认状态为清零。
Bias	REAL	Bias 信号输入。指令将 Bias 添加到输入的总和中。 有效值 = 浮点数 默认值 = 0.0

输出参数：	数据类型：	说明：
EnableOut	BOOL	输出使能。
Out	REAL	算法计算出的输出。该输出影响算术状态标志。

说明： SSUM 指令的操作如下：

条件：	动作：
未选择输入	$Out = Bias$
已选择输入	$Out = \sum_{n=1}^{8} (In_n \times Gain_n) + Bias$

算术状态标志： Out 输出影响算术状态标志。

故障条件： 无

执行：

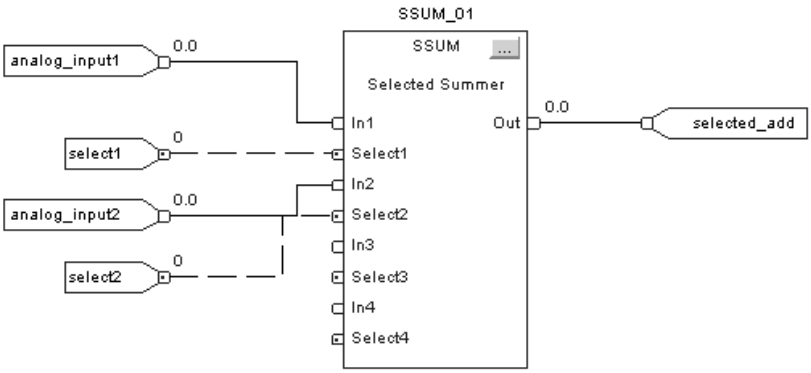
条件：	功能块动作：	结构化文本动作：
预扫描	无动作。	无动作。
指令首次扫描	无动作。	无动作。
指令首次执行	无动作。	无动作。
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。	na
EnableIn 置位	指令执行。 EnableOut 置位。	EnableIn 一直保持置位状态。 指令执行。
后期扫描	无动作。	无动作。

示例： Select1 和 Select2 的值分别决定是否选择 analog_input1 和 analog_input2。指令然后将所选输入相加并将结果置入 Out。

结构化文本

```
SSUM_01.In1 := analog_input1;
SSUM_01.Select1 := select1;
SSUM_01.In2 := analog_input2;
SSUM_01.Select2 := select2;
SSUM(SSUM_01);

selected_add := SSUM_01.Out;
```

功能块



统计指令
（MAVE、 MAXC、 MINC、 MSTD）

简介

可使用以下统计指令：

如要：	可用指令：	可用编程语言：	参见页：
计算时间均值。	移动均值 (MAVE)	结构化文本 功能块	5-2
及时发现最大信号。	获取最大值 (MAXC)	结构化文本 功能块	5-6
及时发现最小信号。	获取最小值 (MINC)	结构化文本 功能块	5-9
计算移动标准偏差。	移动标准偏差 (MSTD)	结构化文本 功能块	5-12

移动均值 (MAVE)

MAVE 指令计算信号 In 的时间均值。该指令有选择地支持用户指定的加权。

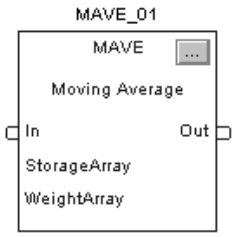
操作数：

结构化文本

```
MAVE(MAVE_tag,storage,weight)
```

操作数：	类型：	格式：	说明：
MAVE 标签	MOVING_AVERAGE	结构	MAVE 结构
存储	REAL	数组	保持移动均值样本。数组必须至少与 NumberOfSamples 一样大。
weight	REAL	数组	（可选） 用于加权均值。数组必须至少与 NumberOfSamples 一样大。元素 [0] 用于最新的样本；元素 [n] 用于最旧的样本。

功能块



操作数与结构化文本 FGEN 指令的操作数相同。

MOVING_AVERAGE 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果该位清零，则指令不执行且输出不更新。 如果该位已设置，则指令执行。 缺省状态为已设置。 **结构化文本：** 无效。指令执行。
In	REAL	输入到指令的模拟信号。 有效值 = 浮点数 缺省值 = 0.0
InFault	BOOL	输入信号错误指示。如果从模拟输入中读取 In， InFault 通常由模拟输入的错误状态字控制。设置时， InFault 表示输入信号有误，指令将设置状态字中相应位，并使 Out 保持其当前值。 InFault 由已设置转换为清零时，指令初始化均值算法并继续执行。 缺省状态为清零。

输入参数：	数据类型：	说明：
Initialize	BOOL	指令的初始化输入。设置时，指令保持 Out = In，但如果 InFault 已设置，指令将使 Out 保持其当前值。 Initialize 由已设置转换为清零时，指令将初始化均值算法并继续执行。 缺省状态为清零。
SampleEnable	BOOL	启用输入采样。设置时，指令将 In 值输入存储数组并计算一个新的 Out 值。 SampleEnable 清零且 Initialize 清零时，指令使 Out 保持其当前值。 缺省状态为已设置。
NumberOfSamples	DINT	计算中使用的采样点数。如果这个值无效，指令将设置状态字中相应位，并使 Out 保持其当前值。 NumberOfSamples 再次生效时，指令初始化均值算法并继续执行。 Valid = 1 到 （StorageArray 或 WeightArray 的最小大小蜒如果使用的话） 缺省值 = 1
UseWeights	BOOL	指令的均值方案输入。设置时，指令使用加权法计算 Out。清零时，指令使用统一法计算 Out。 缺省状态为清零。

输出参数：	数据类型：	说明：
EnableOut	BOOL	输出使能。
Out	REAL	算法计算的输出。该输出影响算术状态标志。
状态	DINT	功能块的状态。
InstructFault （Status.0）	BOOL	指令检测到如下运行错误。该错误不是次要或主要的控制器错误。检查其他状态位以确定错误类型。
InFaulted （Status.1）	BOOL	In 有错误 （InFault 已设置）。
NumberOfSampInv （Status.2）	BOOL	NumberOfSamples 无效或与数组大小不匹配。

说明： MAVE 指令计算输入信号的加权或未加权移动均值。
Number Of Samples 指定移动均值范围的长度。 SampleEnable 已设置时，每次扫描该块，指令都将 In 值移入存储数组并丢弃最旧的值。每个 In_n 都有一个用户配置的 $Weight_n$，用于 UseWeights 已设置时。

MAVE 指令使用以下方程式：

条件：	方程式：
加权均值法 UseWeights 已设置	$Out = \sum_{n=1}^{NumberOfSamples} Weight_n \times In_n$
统一均值法 UseWeights 已清零	$Out = \frac{\sum_{n=1}^{NumberOfSamples} In_n}{NumberOfSamples}$

指令不能将无效的 ln 值 （NAN 或 ± INF）输入存储数组。 In 无效时，指令设置 Out = In 并设置算术溢出状态标志。 In 变为有效时，指令初始化均值算法并继续执行。

可在运行时更改 NumberOfSamples 参数。如果增加参数，指令将逐渐把新的数据从当前样本尺寸平均为新的样本尺寸。如果减少参数，指令将重新计算从样本数组开始到新的 NumberOfSamples 值的均值。

初始化均值算法

有些情况要求指令初始化移动均值算法，比如指令初次扫描时及指令初次执行时。发生这些情况时，指令认为样本数组为空，并逐渐从 1 到 NumberOfSamples 值将样本进行平均。例如：

NumberOfSamples = 3， UseWeights 已设置

扫描 1：Out = In_n*$Weight_1$

扫描 2：Out = (In_n*$Weight_1$) +(In_{n-1}*$Weight_2$)

扫描 3：Out = (In_n*$Weight_1$) +(In_{n-1}*$Weight_2$) +(In_{n-2}*$Weight_3$)

NumberOfSamples = 3， UseWeights 已清零

扫描 1：Out = In_n/1

扫描 2：Out = (In_n+In_{n-1})/2

扫描 3：Out = (In_n+In_{n-1}+In_{n-2})/NumberOfSamples

算术状态标志： Out 输出影响算术状态标志。

错误条件： 无

执行：

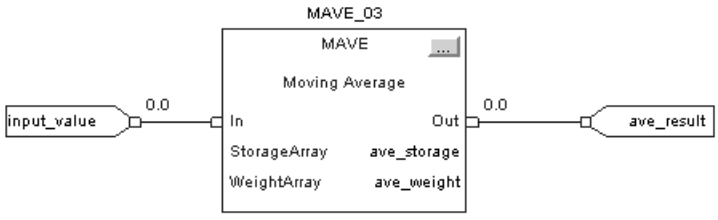
条件：	功能块动作：	结构化文本动作：
预扫描	无动作。	无动作。
指令首次扫描	如果 InFault 清零，则指令初始化算法并继续。	如果 InFault 清零，则指令初始化算法并继续。
指令首次执行	如果 InFault 清零，则指令初始化算法并继续。	如果 InFault 清零，则指令初始化算法并继续。
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。	N/A
EnableIn 已设置	指令执行。 EnableOut 已设置。	EnableIn 一直保持已设置状态。 指令执行。
后期扫描	无动作。	无动作。

示例： 每次扫描时指令都将 input_value 放入数组存储。指令计算数组存储里的均值，有选择地使用数组加权里的加权值，然后将结果放入 Out。

结构化文本

```
MAVE_03.In := input_value;
MAVE(MAVE_03,ave_storage,ave_weight);
ave_result := MAVE_03.Out;
```

功能块



获取最大值 (MAXC)

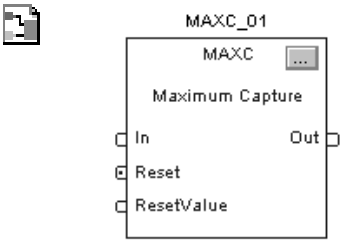
MAXC 指令过后发现 Input 信号的最大值。

操作数：

```
MAXC(MAXC_tag);
```

结构化文本

操作数：	类型：	格式：	说明：
MAXC 标签	MAXIMUM_CAPTURE	结构	MAXC 结构



功能块

操作数：	类型：	格式：	说明：
MAXC 标签	MAXIMUM_CAPTURE	结构	MAXC 结构

MAXIMUM_CAPTURE 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果该位清零，则指令不执行且输出不更新。 如果该位已设置，则指令执行。 缺省状态为已设置。 **结构化文本：** 无效。指令执行。
In	REAL	输入到指令的模拟信号。 有效值 = 浮点数 缺省值 = 0.0
Reset	BOOL	复位控制算法请求。只要 Reset 已设置，指令设置 Out = ResetValue。 缺省状态为清零。
ResetValue	REAL	指令的复位值。只要 Reset 已设置，指令设置 Out = ResetValue。 有效值 = 浮点数 缺省值 = 0.0
输出参数：	**数据类型：**	**说明：**
EnableOut	BOOL	输出使能。
Out	REAL	算法计算的输出。该输出影响算术状态标志。

说明： MAXC 指令执行以下算法：

条件：	动作：
Reset 已设置	Out_{n-1} = ResetValue Out = ResetValue
Reset 清零	当 In > Out_{n-1} 时， Out = In 当 In ≤ Out_{n-1} 时， Out = Out_{n-1} Out_{n-1} = Out

算术状态标志： Out 输出影响算术状态标志。

错误条件： 无

执行：

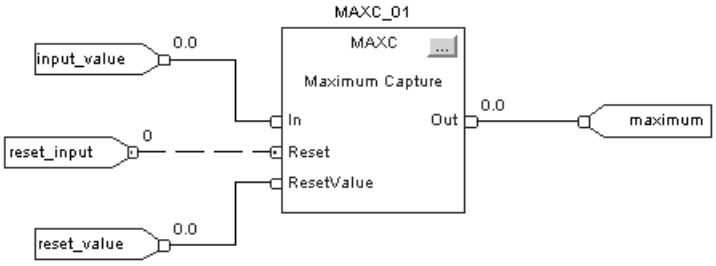
条件：	动作：	动作：
预扫描	无动作。	无动作。
指令首次扫描	Out_{n-1} = In	Out_{n-1} = In
指令首次执行	Out_{n-1} = In	Out_{n-1} = In
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。	na
EnableIn 已设置	指令执行。 EnableOut 已设置。	EnableIn 一直保持已设置状态。 指令执行。
后期扫描	无动作。	无动作。

示例： 如果 Reset 已设置，则指令设置 Out = ResetValue。如果 Reset 清零，指令在 In > Out_{n-1} 时设置 Out = In。否则，指令设置 Out = Out_{n-1}。

结构化文本

```
MAXC_01.In := input_value;
MAXC_01.Reset := reset_input;
MAXC_01.ResetValue := reset_value;
MAXC(MAXC_01);
maximum := MAXC_01.Out;
```

功能块



获取最小值 (MINC)

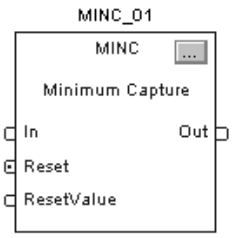
MINC 指令过后发现 Input 信号的最小值。

操作数：

```
MINC(MINC_tag);
```

结构文本

操作数：	类型：	格式：	说明：
MINC 标签	MINIMUM_CAPTURE	结构	MINC 结构



功能块

操作数：	类型：	格式：	说明：
MINC 标签	MINIMUM_CAPTURE	结构	MINC 结构

MINIMUM_CAPTURE 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果该位清零，则指令不执行且输出不更新。 如果该位已设置，则指令执行。 缺省状态为已设置。 **结构化文本：** 无效。指令执行。
In	REAL	输入到指令的模拟信号。 有效值 = 浮点数 缺省值 = 0.0
Reset	BOOL	复位控制算法请求。只要 Reset 已设置，指令设置 Out = ResetValue。 缺省状态为清零。
ResetValue	REAL	指令的复位值。只要 Reset 已设置，指令设置 Out = ResetValue。 有效值 = 浮点数 缺省值 = 0.0
输出参数：	**数据类型：**	**说明：**
EnableOut	BOOL	输出使能。
Out	REAL	算法计算的输出。该输出影响算术状态标志。

说明： MINC 指令执行以下算法：

条件：	动作：
Reset 已设置	Out_{n-1} = ResetValue Out = ResetValue
Reset 清零	当 In < Out_{n-1} 时， Out = In 当 In ≥ Out_{n-1} 时， Out = Out_{n-1} Out_{n-1} = Out

算术状态标志： Out 输出影响算术状态标志。

错误条件： 无

执行：

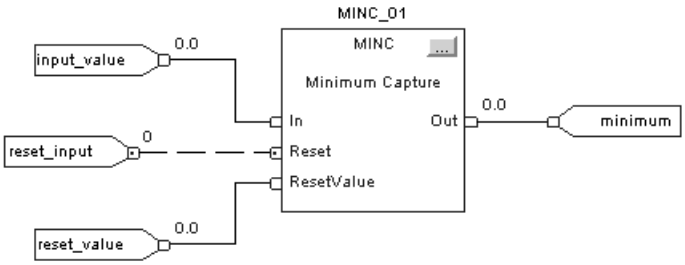
条件：	功能块动作：	结构化文本动作：
预扫描	无动作。	无动作。
指令首次扫描	Out_{n-1} = In	Out_{n-1} = In
指令首次执行	Out_{n-1} = In	Out_{n-1} = In
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。	na
EnableIn 已设置	指令执行。 EnableOut 已设置。	EnableIn 一直保持已设置状态。 指令执行。
后期扫描	无动作。	无动作。

示例： 如果 Reset 已设置，则指令设置 Out = ResetValue。如果 Reset 清零，指令在 In < Out_{n-1} 时设置 Out = In。否则，指令设置 Out = Out_{n-1}。

结构化文本

```
MINC_01.In := input_value;
MINC_01.Reset := reset_input;
MINC_01.ResetValue := reset_value;
MINC(MINC_01);
minimum := MINC_01.Out;
```

功能块



移动标准偏差 (MSTD)

MSTD 指令计算 In 信号的移动标准偏差及均值。

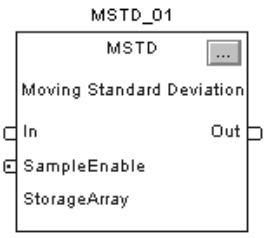
操作数：

结构文本

```
MSTD(MSTD_tag,storage);
```

操作数：	类型：	格式：	说明：
MSTD 标签	MOVING_STD_DEV	结构	MSTD 结构
storage	REAL	数组	保持 In 采样值。数组必须至少与 NumberOfSamples 一样大。

功能块



操作数：	类型：	格式：	说明：
MSTD 标签	MOVING_STD_DEV	结构	MSTD 结构
storage	REAL	数组	保持 In 采样值。数组必须至少与 NumberOfSamples 一样大。

MOVING_STD_DEV 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果该位清零，则指令不执行且输出不更新。 如果该位已设置，则指令执行。 缺省状态为已设置。 **结构化文本：** 无效。指令执行。
In	REAL	输入到指令的模拟信号。 有效值 = 浮点数 缺省值 = 0.0
InFault	BOOL	输入信号错误指示。如果从模拟输入中读取 In， InFault 通常由模拟输入的错误状态字控制。设置时， InFault 表示输入信号有误，指令将设置状态字中相应位，并使 Out 和 Average 保持其当前值。 InFault 由已设置转换为清零时，指令初始化均值算法并继续执行。 缺省状态为清零。

输入参数：	数据类型：	说明：
Initialize	BOOL	指令的初始化输入。设置时，指令设置 Out = 0.0 并且 Average = In，但当 InFault 已设置时，指令会使 Out 和 Average 都保持其当前值。 Initialize 由已设置转换为清零时，指令初始化标准偏差算法并继续执行。 缺省状态为清零。
SampleEnable	BOOL	启用输入采样。设置时，指令将 In 值输入存储数组，并计算新的 Out 和 Average 值。 SampleEnable 清零且 Initialize 也清零时，指令使 Out 和 Average 保持其当前值。 缺省状态为清零。
NumberOfSamples	DINT	计算中使用的采样点数。如果这个值无效，指令将设置状态字中相应位，并使 Out 和 Average 保持其当前值。当 NumberOfSamples 再次生效时，指令初始化标准偏差算法并继续执行。 Valid = 1 到存储数组大小 缺省值 = 1

输出参数：	数据类型：	说明：
EnableOut	BOOL	输出使能。
Out	REAL	算法计算的输出。SampleEnable 清零时，指令使 Out 保持其当前值。该输出影响算术状态标志。
Average	REAL	算法计算的均值。
Status	DINT	功能块的状态。
InstructFault (Status.0)	BOOL	指令检测到如下运行错误。该错误不是次要或主要的控制器错误。检查其他状态位以确定错误类型。
InFaulted (Status.1)	BOOL	输入出错。 InFault 已设置。
NumberOfSampInv (Status.2)	BOOL	NumberOfSamples 无效或与数组大小不匹配。

说明： MSTD 指令支持任何输入队列长度。如果 SampleEnable 已设置，每次扫描时指令会将 In 值输入存储数组。存储数组已满时，每个新的 In 值都会导致最旧的条目被删除。

MSTD 指令使用以下方程式计算输出：

条件：	等于：
均值	$$Average = \frac{\sum_{n=1}^{NumberOfSamples} In_n}{NumberOfSamples}$$
Out	$$Out = \sqrt{\frac{\sum_{n=1}^{NumberOfSamples} (In_n - Average)^2}{NumberOfSamples}}$$

指令不能将无效的 In 值 （NAN 或 ± INF）输入存储数组。当 In 无效时，指令设置 Out = In、 Average = In，并设置算术溢出状态标志。当 In 变得有效时，指令初始化标准偏差算法并继续执行。

可在运行时更改 NumberOfSamples 参数。如果增加参数，指令逐渐将新的数据从当前的样本尺寸处理为新的样本尺寸。如果减少参数，指令会重新计算从样本数组开始到新的 NumberOfSamples 值的标准偏差。

初始化标准偏差算法

某些情况要求指令初始化标准偏差算法，比如指令初次扫描时及指令初次执行时。当这些情况发生时，指令认为样本数组为空并逐渐从 1 到 NumberOfSamples 值，对样本进行处理。例如：

NumberOfSamples = 3

扫描 1：Average = $In_n/1$

Out = $(((In_n\text{-Average})^2)/1)$ 的平方根

扫描 2：Average = $(In_n+In_{n-1})/2$

Out = $(((In_n\text{-Average})^2+(In_{n-1}\text{-Average})^2)/2)$ 的平方根

扫描 3： Average = $(In_n+In_{n-1}+In_{n-2})$/NumberOfSamples

Out = $(((In_n\text{-Average})^2+(In_{n-1}\text{-Average})^2+(In_{n-2}\text{-Average})^2)$/NumberOfSamples) 的平方根

算术状态标志： Out 输出影响算术状态标志。

错误条件： 无

执行：

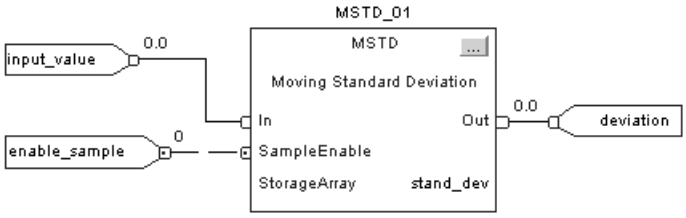
条件：	功能块动作：	结构化测试动作：
预扫描	无动作。	无动作。
指令首次扫描	如果 InFault 清零，则指令初始化算法并继续。	如果 InFault 清零，则指令初始化算法并继续。
指令首次执行	如果 InFault 清零，则指令初始化算法并继续。	如果 InFault 清零，则指令初始化算法并继续。
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。	na
EnableIn 已设置	指令执行。 EnableOut 已设置。	EnableIn 一直保持已设置状态。 指令执行。
后期扫描	无动作。	无动作。

示例： SampleEnable 已设置时，每次扫描指令都将 In 值输入数组存储，计算数组存储中的值的标准偏差，并把结果放入 Out。

结构化文本

```
MSTD_01.In := input_value;
MSTD_01.Sample_enable := enable_sample;
MSTD(MSTD_01,stand_dev);
deviation := MSTD_01.Out;
```

功能块



说明：

移动 / 逻辑指令
（DFF、 JKFF、 RESD、 SETD）

简介

可以使用以下移动 / 逻辑指令：

如要：	所用指令：	可用编程语言：	参见页：
在 Clock 输入跳转时，将 Q 输出为 D 输入的状态。	D 触发器 (DFF)	结构化文本 功能块	6-2
当 Clock 输入跳转时，将 Q 和 QNot 输出取反。	JK 触发器 (JKFF)	结构化文本 功能块	6-4
当 Reset 输入优先于 Set 输入时，使用 Set 和 Reset 输入控制锁存输出。	复位优先 (RESD)	结构化文本 功能块	6-6
当 Set 输入优先于 Reset 输入时，使用 Set 和 Reset 输入控制锁存输出。	置位优先 (SETD)	结构化文本 功能块	6-8

D 触发器 (DFF)

在 Clock 输入从清零到置位的转变中， DFF 指令将 Q 输出设置为 D 输入的状态。 QNot 输出被设置为 Q 输出的相反状态。

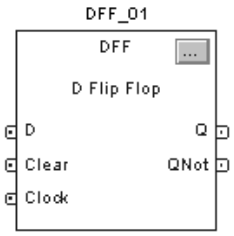
操作数：

```
DFF(DFF_tag);
```

结构化文本

操作数：	类型：	格式：	说明：
DFF 标记	FLIP_FLOP_D	结构	DFF 结构

功能块



操作数：	类型：	格式：	说明：
DFF 标记	FLIP_FLOP_D	结构	DFF 结构

FLIP_FLOP_D 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果该位清零，则指令不执行且输出不更新。 如果该位置位，则指令执行。 默认状态为置位。 **结构化文本：** 无效。指令执行。
D	BOOL	指令的输入。 默认状态为清零。
Clear	BOOL	指令输入清零。置位时，指令将 Q 清零并置位 QNot。 默认状态为清零。
Clock	BOOL	指令的时钟输入。 默认状态为清零。
输出参数：	**数据类型：**	**说明：**
EnableOut	BOOL	输出使能。
Q	BOOL	指令的输出。
QNot	BOOL	Q 输出值取反。

说明： Clear 置位时，指令将 Q 清零并置位 QNot。否则如果 Clock 置位且 $Clock_{n-1}$ 清零，指令将设置 Q = D 并设置 QNot = NOT (D)。

每次扫描时指令设置 $Clock_{n-1}$ = Clock。

算术状态标志： 不影响

故障条件： 无

执行：

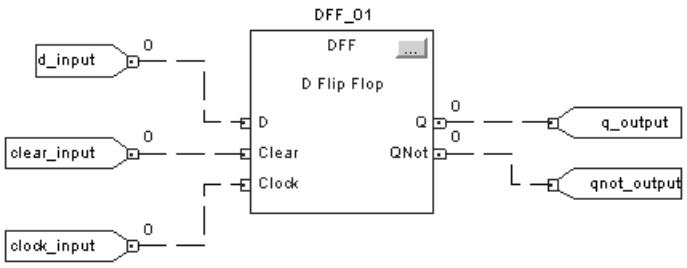
条件：	功能块动作：	结构化文本动作：
预扫描	无动作。	无动作。
指令首次扫描	$Clock_{n-1}$ Q 清零 QNot 置位	$Clock_{n-1}$ Q 清零 QNot 置位
指令首次执行	$Clock_{n-1}$ Q 清零 QNot 置位	$Clock_{n-1}$ Q 清零 QNot 置位
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。	na
EnableIn 置位	指令执行。 EnableOut 置位。	EnableIn 一直保持置位状态。 指令执行。
后期扫描	无动作。	无动作。

示例： 当 Clock 从清零变为置位时， DFF 指令设置 Q = D。当 Clear 位置时，Q 清零。 DFF 指令将 QNot 设置为 Q 的相反状态。

结构化文本

```
DFF_01.D := d_input;
DFF_01.Clear := clear_input;
DFF_01.Clock := clock_input;
DFF(DFF_01);
q_output := DFF_01.Q;
qnot_output := DFF_01.QNot;
```

功能块



JK 触发器 (JKFF)

当 Clock 输入从清零变为置位时， JKFF 指令将 Q 和 QNot 输出取反。

操作数：

```
JKFF(JKFF_tag);
```

结构化文本

操作数：	类型：	格式：	说明：
JKFF 标记	FLIP_FLOP_JK	结构	JKFF 结构

功能块

操作数：	类型：	格式：	说明：
JKFF 标记	FLIP_FLOP_JK	结构	JKFF 结构

FLIP_FLOP_JK 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果该位清零，则指令不执行且输出不更新。 如果该位置位，则指令执行。 默认状态为置位。 **结构化文本：** 无效。指令执行。
Clear	BOOL	指令输入清零。置位时，指令将 Q 清零并置位 QNot。 默认状态为清零。
Clock	BOOL	指令的时钟输入。 默认状态为清零。
输出参数：	**数据类型：**	**说明：**
EnableOut	BOOL	输出使能。
Q	BOOL	指令的输出。
QNot	BOOL	Q 输出值取反。

说明： Clear 置位时，指令将 Q 清零并置位 QNot。否则，如果 Clock 置位且 $Clock_{n-1}$ 清零，指令将翻转 Q 和 QNot。

每次扫描时指令设置 $Clock_{n-1}$ = Clock。

算术状态标志： 不影响

故障条件： 无

执行：

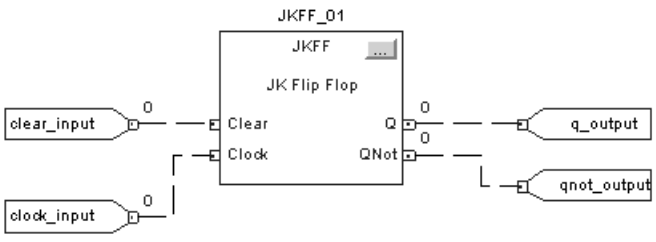
条件：	功能块动作：	结构化文本动作：
预扫描	无动作。	无动作。
指令首次扫描	$Clock_{n-1}$ Q 清零 QNot 置位	$Clock_{n-1}$ Q 清零 QNot 置位
指令首次执行	$Clock_{n-1}$ Q 清零 QNot 置位	$Clock_{n-1}$ Q 清零 QNot 置位
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。	na
EnableIn 置位	指令执行。 EnableOut 置位。	EnableIn 一直保持置位状态。 指令执行。
后期扫描	无动作。	无动作。

示例： 当 Clock 从清零变为置位时， JKFF 指令翻转 Q。如果 Clear 置位，Q 始终为清零。 JKFF 指令将 QNot 设置为 Q 的相反状态。

结构化文本

```
JKFF_01.Clear := clear_input;
JKFF_01.Clock := clock_input;
JKFF(JKFF_01);
q_output := JKFF_01.Q;
qnot_output := JKFF_01.QNot;
```

功能块



复位优先 (RESD)

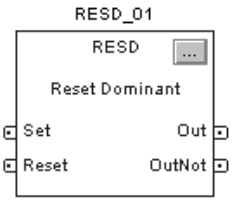
RESD 指令使用 Set 和 Reset 输入控制锁存输出。Reset 输入优先于 Set 输入。

操作数：

```
RESD(RESD_tag);
```

结构化文本

操作数：	类型：	格式：	说明：
RESD 标记	DOMINANT_RESET	结构	RESD 结构



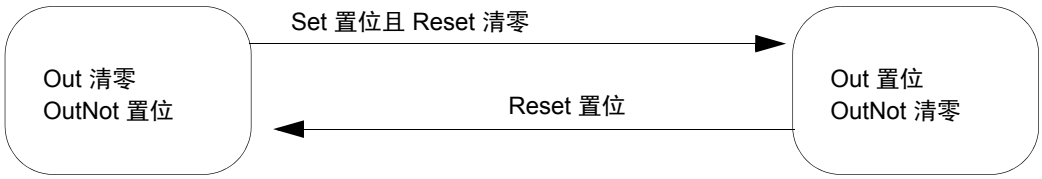
功能块

操作数：	类型：	格式：	说明：
RESD 标记	DOMINANT_RESET	结构	RESD 结构

DOMINANT_RESET 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果该位清零，则指令不执行且输出不更新。 如果该位置位，则指令执行。 默认状态为置位。 **结构化文本：** 无效。指令执行。
Set	BOOL	置位指令输入。 默认状态为清零。
Reset	BOOL	复位指令输入。 默认状态为清零。
输出参数：	**数据类型：**	**说明：**
EnableOut	BOOL	输出使能。
Out	BOOL	指令的输出。
OutNot	BOOL	指令的取反输出。

说明： 下图显示 RESD 指令的操作



算术状态标志： 不影响

故障条件： 无

执行：

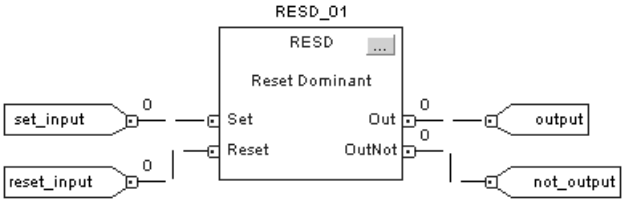
条件：	功能块动作：	结构化文本动作：
预扫描	无动作。	无动作。
指令首次扫描	无动作。	无动作。
指令首次执行	Out 清零。 OutNot 置位。	Out 清零。 OutNot 置位。
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。	na
EnableIn 置位	指令执行。 EnableOut 已置位。	EnableIn 一直保持置位状态。 指令执行。
后期扫描	无动作。	无动作。

示例： Set 置位时， Out 置位； Reset 置位时， Out 清零。 Reset 优先于 Set。

结构化文本

```
RESD_01.Set := set_input;
RESD_01.Reset := reset_input;
RESD(RESD_01);
output := RESD_01.Out;
not_output := RESD_01.OutNot;
```

功能块



置位优先 (SETD)

SETD 指令使用 Set 和 Reset 输入控制锁存输出。Set 输入优先于 Reset 输入。

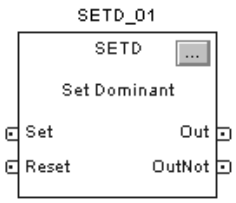
操作数：

结构化文本

```
SETD(SETD_tag);
```

操作数：	类型：	格式：	说明：
SETD 标记	DOMINANT_SET	结构	SETD 结构

功能块



操作数：	类型：	格式：	说明：
SETD 标记	DOMINANT_SET	结构	SETD 结构

DOMINANT_SET 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果该位清零，则指令不执行且输出不更新。 如果该位置位，则指令执行。 默认状态为置位。 **结构化文本：** 无效。指令执行。
Set	BOOL	置位指令输入。 默认状态为清零。
Reset	BOOL	复位指令输入。 默认状态为清零。
输出参数：	**数据类型：**	**说明：**
EnableOut	BOOL	输出使能。
Out	BOOL	指令的输出。
OutNot	BOOL	指令的取反输出。

说明： 下图显示 SETD 指令的操作

算术状态标志： 不影响

故障条件： 无

执行：

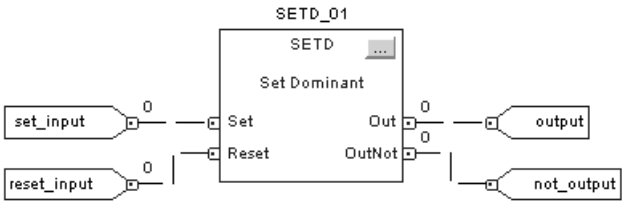
条件：	功能块动作：	结构化文本动作：
预扫描	无动作。	无动作。
指令首次扫描	无动作。	无动作。
指令首次执行	Out 置位。 OutNot 清零。	Out 置位。 OutNot 清零。
EnableIn 清零	EnableOut 清零，指令不执行且输出不更新。	na
EnableIn 置位	指令执行。 EnableOut 置位。	EnableIn 一直保持置位状态。 指令执行。
后期扫描	无动作。	无动作。

示例： Set 置位时， Out 置位； Reset 置位时， Out 清零。 Set 优先于 Reset。

结构化文本

```
SETD_01.Set := set_input;
SETD_01.Reset := reset_input;
SETD(SETD_01);
output := SETD_01.Out;
not_output := SETD_01.OutNot;
```

功能块



说明：

通用属性

简介

本附录说明的是 Logix 指令的通用属性。

关于以下信息：	参见页：
立即数	A-1
数据转换	A-1

立即数

每当以十进制 （如 -2、 3）格式输入立即数 （常数）时，控制器都会使用 32 位进行存储。如果使用非十进制基数 （如二进制或十六进制）输入值，并且未对所有 32 位都进行指定，则控制器会将零添加到未指定的位中 （补零）。

示例 立即数的补零

如果输入：	则存储器将存储：
-1	16#ffff ffff (-1)
16#ffff (-1)	16#0000 ffff (65535)
8#1234 (668)	16#0000 029c (668)
2#1010 (10)	16#0000 000a (10)

数据转换

在编程中混用多种数据时将发生数据转换：

使用下列语言编程时：	在下列情况中将发生转换：
梯形图逻辑	在一个指令内混用多种数据类型的参数
功能块	在两个数据类型不同的参数之间通信

如果指令的所有操作数均采用以下数据类型，则指令会执行得更快，所需的内存也更少：

- 相同的数据类型
- 最优数据类型：
  - 在本手册中每一个指令的 “操作数”部分内，加粗的数据类型表示最优数据类型。
  - DINT 和 REAL 数据类型通常是最优数据类型。
  - 对于其操作数，大多数功能块指令仅支持一种数据类型（最优数据类型）。

如果混用多种数据，并且使用的标记不属于最优数据类型，则控制器将按下列规则转换数据

- 操作数中是否有 REAL 值？

如果：	则会将输入操作数 （如源数据、表达式中的标记、限度值）转换为：
是	REAL
否	DINT

- 执行指令后，如有必要，则会将结果 （DINT 或 REAL 值）转换为目标数据类型。

无法在处理整数或 REAL 数据类型的指令中指定 BOOL 标记。

由于数据转换要耗费更多时间和内存，因此可以通过下列方式提高程序的效率：

- 在同一指令中始终使用相同的数据类型
- 尽量少使用 SINT 或 INT 数据类型

换句话说，要在指令中全部使用 DINT 标记和立即数，或全部使用 REAL 标记和立即数。

下一节说明在使用 SINT 或 INT 标记或混用多种数据时将如何转换数据。

将 SINT 或 INT 转换为 DINT

对于将 SINT 或 INT 值转换为 DINT 值的指令，本手册中的 “操作数” 部分指明了转换方法。

以下转换方法：	通过下列方式转换数据：
符号扩充	将最左边的位 （值的符号）的值添加到现有位左边的每一位中，直到满 32 位为止。
补零	将零添加到现有位的左边，直到满 32 位为止

下例所示的是使用符号扩充和补零转换值的结果。

对于值	2#1111_1111_1111_1111	(-1)
通过符号扩充转换为	2#1111_1111_1111_1111_1111_1111_1111_1111	(-1)
通过补零转换为	2#0000_0000_0000_0000_1111_1111_1111_1111	(65535)

因为立即数始终都经过补零，所以 SINT 和 INT 值的转换可能会产生意外的结果。在下例中，由于源数据 A （INT 值）通过符号扩充进行转换，而源数据 B （立即数）是通过补零转换的，因此其比较结果是不相同。

如果在通过符号扩充转换数据的指令中使用 SINT 或 INT 标记和立即数，则使用下列方法之一来处理立即数：

- 使用十进制基数指定立即数
- 如果使用非十进制基数输入值，则要指定立即数的所有 32 位。为此，将最左边的位的值输入到其左边的每一位中，直到满 32 位为止。
- 对每个操作数创建标记并在整个指令中使用相同的数据类型。也可通过以下方法指定常数值：
  - 将其输入某个标记中
  - 添加 MOV 指令，以便将该值移到某个标记中。
- 使用 MEQ 指令仅检查所需的位

下例所示的是混用立即数和 INT 标记的两种方式。两个示例都是检查 1771 I/O 模块的位以确定是否已启用所有的位。由于 1771 I/O 模块的输入数据字为 INT 标记，因此使用 16 位常数值最为简单。

示例

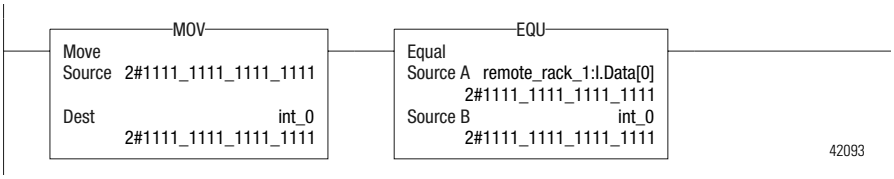
混合使用 INT 标及和立即数

由于 remote_rack_1:I.Data[0] 是 INT 标记，因此据其检查 remote_rack_1:I.Data[0] 的值也是用 INT 标记输入的。

示例

混合使用 INT 标记和立即数

由于 remote_rack_1:I.Data[0] 是一个 INT 标记，因此会将据其检查 remote_rack_1:I.Data[0] 的值先移到 int_0（也是 INT 标记）中。EQU 指令随后将比较这两个标记。



将整数转换为 REAL

控制器以 IEEE 单精度浮点数值格式存储 REAL 值。该格式的值符号占用一位，基值占用 23 位，指数占用 8 位（共 32 位）。如果在同一指令中混用整数标记（SINT、 INT 或 DINT）和 REAL 标记作为输入，则控制器会在执行指令之前将整数值转换为 REAL 值。

- SINT 或 INT 值始终可以转换为相同的 REAL 值。
- DINT 值可能无法转换为相同的 REAL 值：
  - REAL 值的基值最多可以占用 24 位（23 个存储位和一个“隐藏”位）。
  - DINT 值最多占用 32 位（符号占用一位，值占用 31 位）。
  - 如果 DINT 值要求的有效位多于 24，则其可能不会被转换为相同的 REAL 值。如果不能将其转换为相同的 REAL 值，控制器会使用 24 个有效位将其四舍五入到最接近的 REAL 值。

将 DINT 转换为 SINT 或 INT

将 DINT 值转换为 SINT 或 INT 值时，如有必要，控制器会截断 DINT 的高位部分并将溢出状态标志置位。下例所示的是 DINT 到 SINT 或 INT 的转换结果。

示例 DINT 到 INT 和 SINT 的转换

对于 DINT 值：	可以转换为下列较小的值：	
16#0001_0081 (65,665)	INT：	16#0081 (129)
	SINT	16#81 (-127)

将 REAL 转换为整数

要将 REAL 值转换为整数，控制器会对小数部分进行四舍五入并截断非小数部分的高位。如果丢失了数据，则控制器将溢出状态标志置位。数值的四舍五入如下：

- 将除 x.5 以外的数值四舍五入为最接近的整数。
- 将 X.5 四舍五入为最接近的偶数。

下例所示的是 REAL 值到 DINT 值的转换结果。

示例 REAL 值到 DINT 值的转换

下列 REAL 值：	可以转换为下列 DINT 值：
-2.5	-2
-1.6	-2
-1.5	-2
-1.4	-1
1.4	1
1.5	2
1.6	2
2.5	2

重要事项 算术状态标志是根据所存储的值来置位的。如果因混用多种数据类型的指令参数而发生类型转换，则通常不会影响算术状态关键字的指令也可能看似有此影响。类型转换过程可以将算术状态关键字置位。

功能块属性

简介

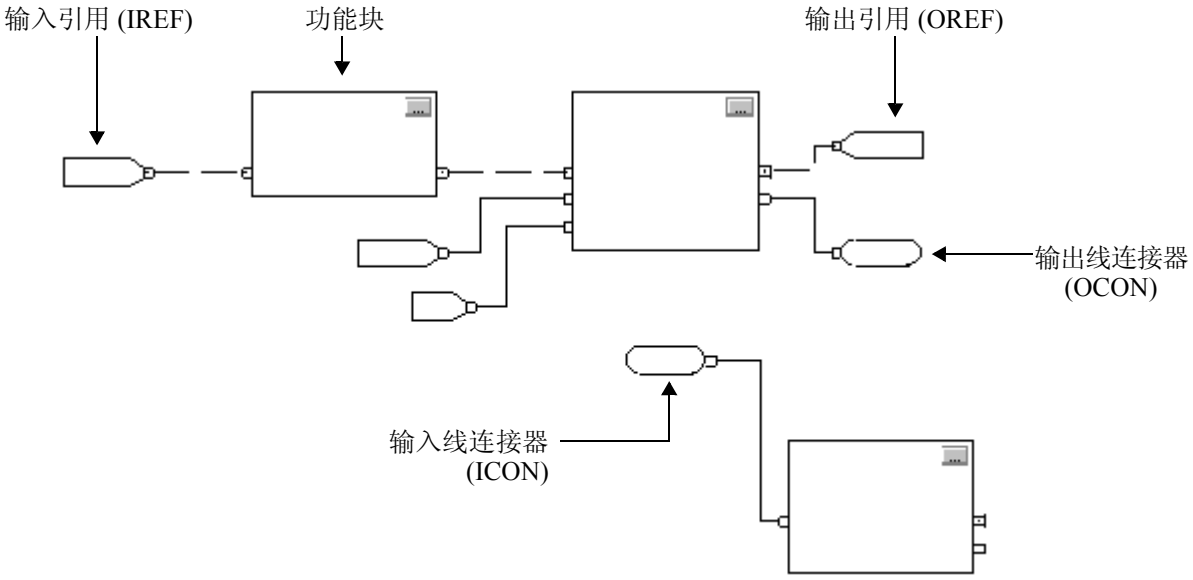
本附录说明的是功能块指令特有的问题。请查看本附录中的信息以确保了解功能块程序的运行方式。

重要事项

由于其内部浮点数运算使用单精度浮点数，在功能块中编程时，工程单位范围限制在 $+/-10^{+/-15}$ 。如果计算结果接近单精度浮点数极限 ($+/-10^{+/-38}$)，使用超出该范围的工程单位会导致精度降低。

选择功能块元素

要控制设备，请使用以下元素：

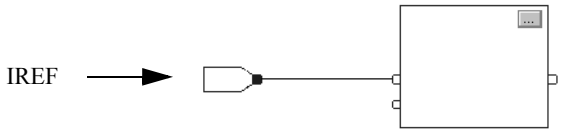


使用下表选择功能块元素：

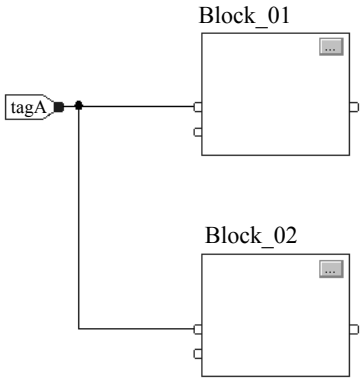
如要：	则使用：
从输入设备或标记中提供值	输入引用 (IREF)
向输入设备或标记发送值	输出引用 (OREF)
对一个或多个输入值执行操作并生成一个或多个输出值	功能块
当功能块处于以下状态时在其间传送数据： • 在同一表内但相距很远 • 在同一程序的不同表内	输出线连接器 (OCON) 和输入线连接器 (ICON)
将数据分送到程序中的若干点	单个输出线连接器 (OCON) 和多个输入线连接器 (ICON)

锁存数据

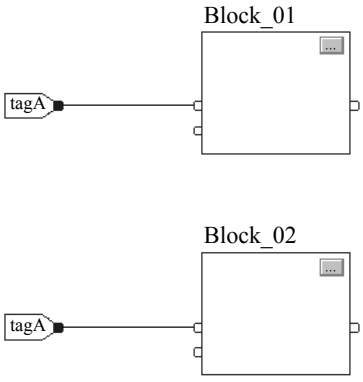
如果使用 IREF 对功能块指令指定输入数据，则该 IREF 中的数据会被锁存以供功能块程序扫描。 IREF 会锁存程序范围和控制器范围内的标记中的数据。控制器会在每次开始扫描时更新所有的 IREF 数据。



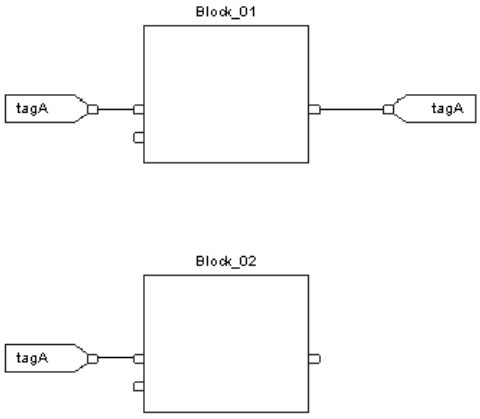
在此例中， tagA 的值是开始执行程序时存储的。执行 Block_01 时会使用存储的值。执行 Block_02 时也会使用同一存储值。如果 tagA 的值在程序执行期间发生变化，则在下一次执行程序之前， IREF 中存储的 tagA 值不会改变。



此例与前一个示例相同。 tagA 的值仅会在开始执行程序时存储一次。程序会在整个执行过程中使用该存储值。



从 RSLogix 5000 软件 11 版开始，您可以在同一程序的多个 IREF 和一个 OREF 内使用同一标记。由于每次扫描程序时 IREF 中的标记值会被锁存，因此，即使 OREF 在执行程序过程中获取了不同的标记值，所有的 IREF 也将使用相同的值。在此例中，如果在程序开始执行此次扫描时 tagA 的值为 25.4，并且 Block_01 将 tagA 的值改为 50.9，则在 Block_02 执行此次扫描时，连接到 Block_02 的第二个 IREF 使用的值仍是 25.4。在下次扫描开始之前，此程序中的所有 IREF 都不会使用 tagA 的新值 50.9。



执行顺序

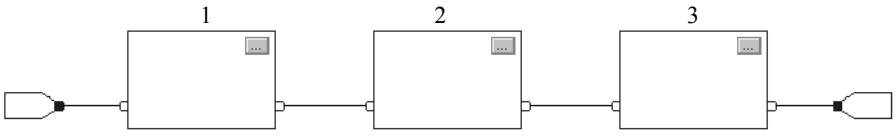
在执行以下操作时， RSLogix 5000 编程软件会自动确定功能块的执行顺序：

- 验证功能块程序
- 验证含有功能块程序的项目
- 下载含有功能块程序的项目

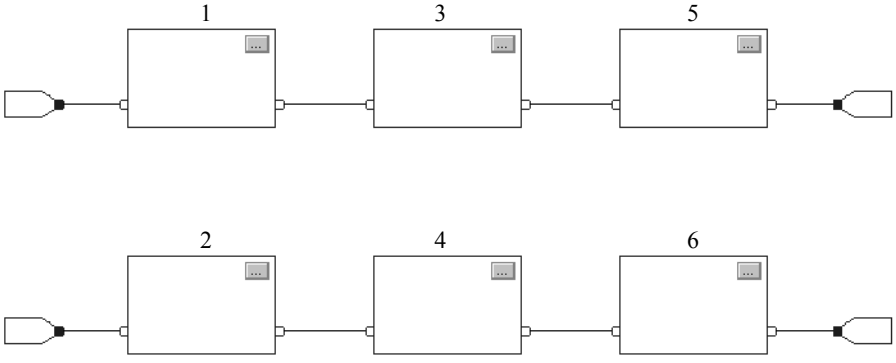
如有必要，可以通过将功能块连接在一起并指明所有反馈线的数据流来指定执行顺序。

如果未将功能块连接在一起，则先执行哪个功能块都无关紧要。功能块之间没有数据流。

如果按顺序连接功能块，则执行顺序为从输入到输出。在控制器执行功能块之前，功能块的输入必须有可用数据。例如，功能块 2 必须先于功能块 3 执行，因为功能块 2 的输出就是功能块 3 的输入。

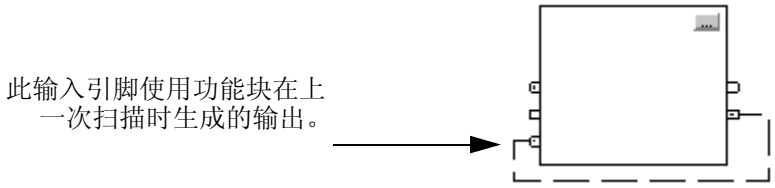


执行顺序仅与连接在一起的功能块有关。以下示例的接法正确，因为未将两组功能块连接在一起。特定组内的功能块的执行顺序只与该组中的功能块相关。

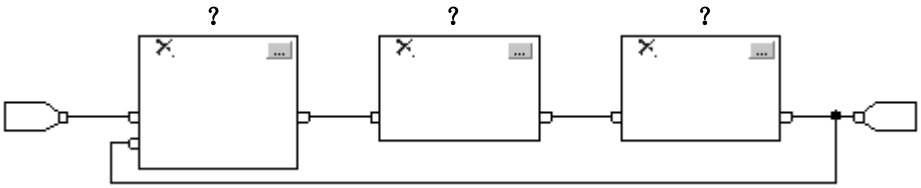


解析回路

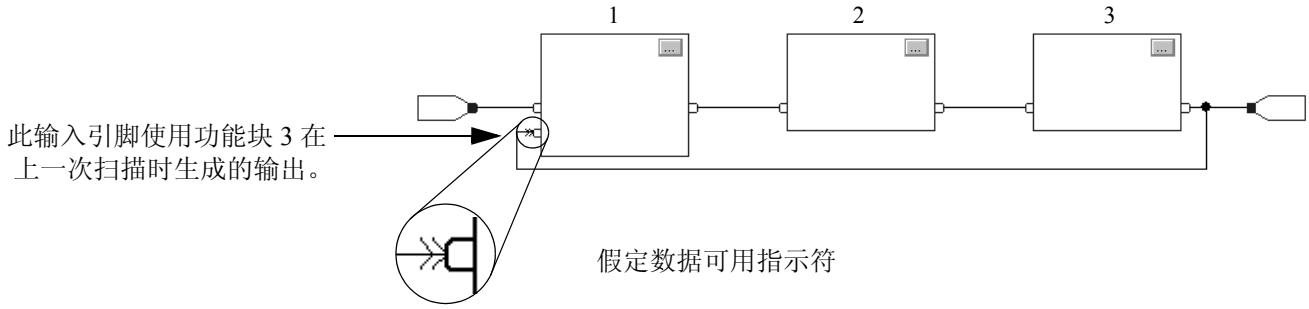
要在功能块周围建一个反馈回路，请将功能块的输出引脚与同一功能块的输入引脚相连。下例接法正确。回路仅含有一个功能块，所以执行顺序无关紧要



如果回路中含有一组功能块，则控制器无法确定首先执行的功能块。换句话说，控制器无法解析回路。

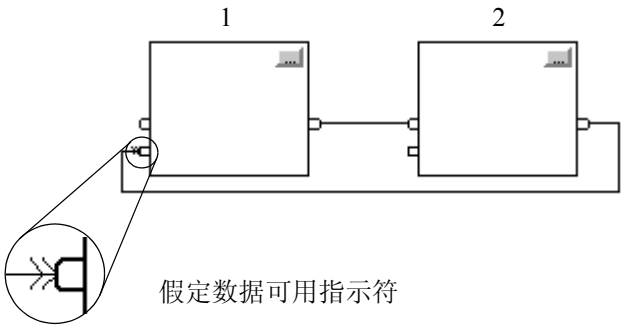
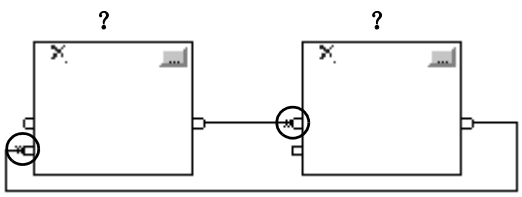


要确定首先执行的功能块，可使用 “假定数据可用”指示符对用于创建回路（反馈线）的输入线作标记。在下例中，功能块 1 使用功能块 3 在上一次程序执行中生成的输出。



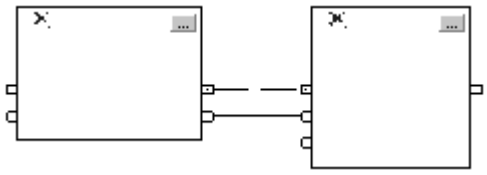
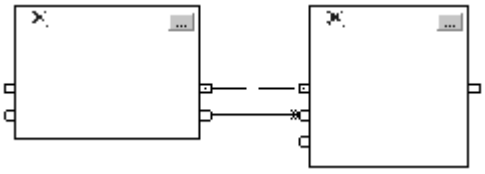
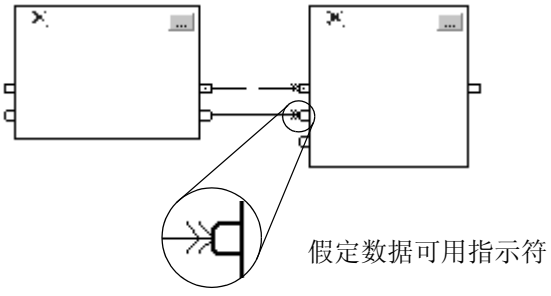
“假定数据可用”指示符用于确定回路中的数据流。箭头表示数据用作回路中第一个功能块的输入。

请勿使用 “假定数据可用”指示符对回路的所有连接线作标记。

正确接法	错误接法
 *假定数据可用* 指示符用于确定回路中的数据流。	 控制器无法解析回路，因为所有连接线都使用了*假定数据可用* 指示符。

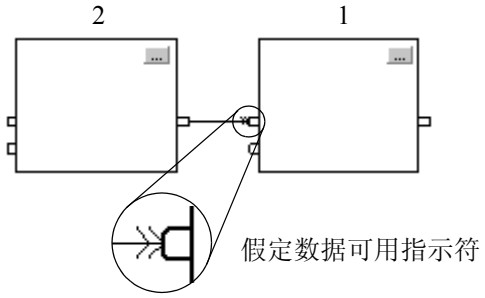
解析两个功能块之间的数据流

如果使用两条或更多条线连接两个功能块，则可将同一数据流指示符用于这两个功能块之间的所有连接线。

正确接法	错误接法
 两条连接线均不使用*假定数据可用* 指示符。	 一条线使用*假定数据可用* 指示符，而另一条线不使用。
 两条连接线均使用*假定数据可用* 指示符。	

创建一次扫描延迟

要在功能块之间生成一次扫描延迟，请使用 “假定数据可用”指示符。在下例中，首先执行的是功能块 1。它使用功能块 2 在上一次程序扫描中生成的输出。



总结

总之，功能块程序的执行顺序如下：

1. 控制器锁存 IREF 中的所有数据值。
2. 控制器按顺序 （由功能块的连接方式决定）执行其他功能块。
3. 控制器将输出写入 OREF。

功能块对溢出条件的响应

一般情况下，维护历史记录的功能块指令不会在发生溢出时使用 ± NAN 或 ± INF 值更新记录。每个指令会对溢出条件作出以下一种响应：

响应 1：		**响应 2：**	**响应 3：**	
功能块执行其算法并检查结果是否为 ±NAN 或 ±INF。如果是 ±NAN 或 ±INF，则功能块将输出 ±NAN 或 ±INF。		有输出限制的功能块执行其算法并检查结果是否为 ±NAN 或 ±INF。输出极限由 HighLimit 和 LowLimit 输入参数决定。如果是 ±INF，功能块输出受限的结果。如果是 ±NAN，则不受限制，功能块将输出 ±NAN。	溢出状况不适用。这些指令通常具有布尔输出。	
ALM	NTCH	HLL	BAND	OSRI
DEDT	PMUL	INTG	BNOT	RESD
DERV	POSP	PI	BOR	RTOR
ESEL	RLIM	PIDE	BXOR	SETD
FGEN	RMPS	SCL	CUTD	TOFR
HPF	SCRV	SOC	D2SD	TONR
LDL2	SEL		D3SD	
LDLG	SNEG		DFF	
LPF	SRTP		JKFF	
MAVE	SSUM		OSFI	
MAXC	TOT			
MINC	UPDN			
MSTD				
MUX				

定时方式

以下过程控制和驱动控制指令支持各种定时方式。

DEDT	LDLG	RLIM
DERV	LPF	SCRV
HPF	NTCH	SOC
INTG	PI	TOT
LDL2	PIDE	

共有三种不同的定时方式：

定时方式：	**说明：**
周期	周期方式是默认的方式，适于大多数控制应用。我们建议将使用此方式的指令放入在周期性任务中执行的程序内。指令的增量时间 (DeltaT) 按以下方式确定：
	如果指令在以下任务中执行： 则 **DeltaT 等于：**
	周期性任务 任务周期
	事件或连续性任务 自上一次执行后经过的时间 控制器会将经过的时间毫秒数 (ms) 取整。例如，如果经过的时间 = 10.5 ms，控制器会设置 DeltaT 为 10 ms。
	过程输入的更新必须与任务执行同步，或其采样频率比任务执行快 5-10 倍，以便将输入和指令之间的采样误差降到最低。
过采样	在过采样方式中，指令使用的增量时间 (DeltaT) 是写入指令的 OversampleDT 参数中的值。如果过程输入具有时间戳值，请改用实时采样方式。 将逻辑添加到程序中以控制指令的执行时间。例如，可以使用设置为 OversampleDeltaT 的定时器，以使用指令中的 EnableIn 输入来控制执行。 过程输入的采样频率必须比指令执行快 5-10 倍，才能将输入和指令之间的采样误差降到最低。
实时采样	在实时采样方式中，指令使用的增量时间 (DeltaT) 是相应于过程输入更新的两个时间戳值之间的差值。当过程输入具有与其更新相关联的时间戳并且需要精确协调时，请使用此方式。 时间戳值从对指令的 RTSTimeStamp 参数输入的标记名称中读取。此标记名称通常是与过程输入相关联的输入模块上的一个参数。 指令会将配置的 RTSTime 值 （预计更新周期）与计算的 DeltaT 进行比较，以确定指令是否读取了过程输入的每一次更新。如果 DeltaT 与配置时间之差超过 1 毫秒，则指令会置位 RTSMissed 状态位，以指示读取模块上输入的更新时出了问题。

基于时间的指令要求 DeltaT 为常数值，以使控制算法正确计算过程输出。如果 DeltaT 不一致，过程输出会发生中断。中断的严重程度取决于指令和 DeltaT 的变化范围。在以下情况中将发生中断：

- 未在扫描过程中执行指令。
- 在某次任务期间指令执行多次。
- 正在执行任务，但任务扫描速率或过程输入的采样时间改变。
- 用户在执行任务时更改时基方式。
- 执行任务时更改过滤器块上的 Order 参数。更改 Order 参数会在指令中选择不同的控制算法。

定时方式的通用指令参数

支持时基方式的指令具有以下输入和输出参数：

输入参数

输入参数：	数据类型：	说明：
TimingMode	DINT	选择定时执行方式。 **值：** **说明：** 0 周期方式 1 过采样方式 2 实时采样方式 有效值 = 0 到 2 默认值 = 0 当 TimingMode = 0 且为周期性任务时会启用周期定时，并且会将 DeltaT 设置为等于任务扫描速率。当 TimingMode = 0 且为事件或连续性任务时会启用周期定时，并且会将 DeltaT 设置为等于自上次执行指令后经过的时间。 当 TimingMode = 1 时会启用过采样定时，并且会将 DeltaT 设置为等于 OversampleDT 参数的值。 当 TimingMode = 2 时会启用实时采样定时，并且会将 DeltaT 置位为当前和上次时间戳值之间的差值，这两个时间戳值是从与输入相关联的模块中读取的。 如果 TimingMode 无效，指令会在状态中置位相应的位。

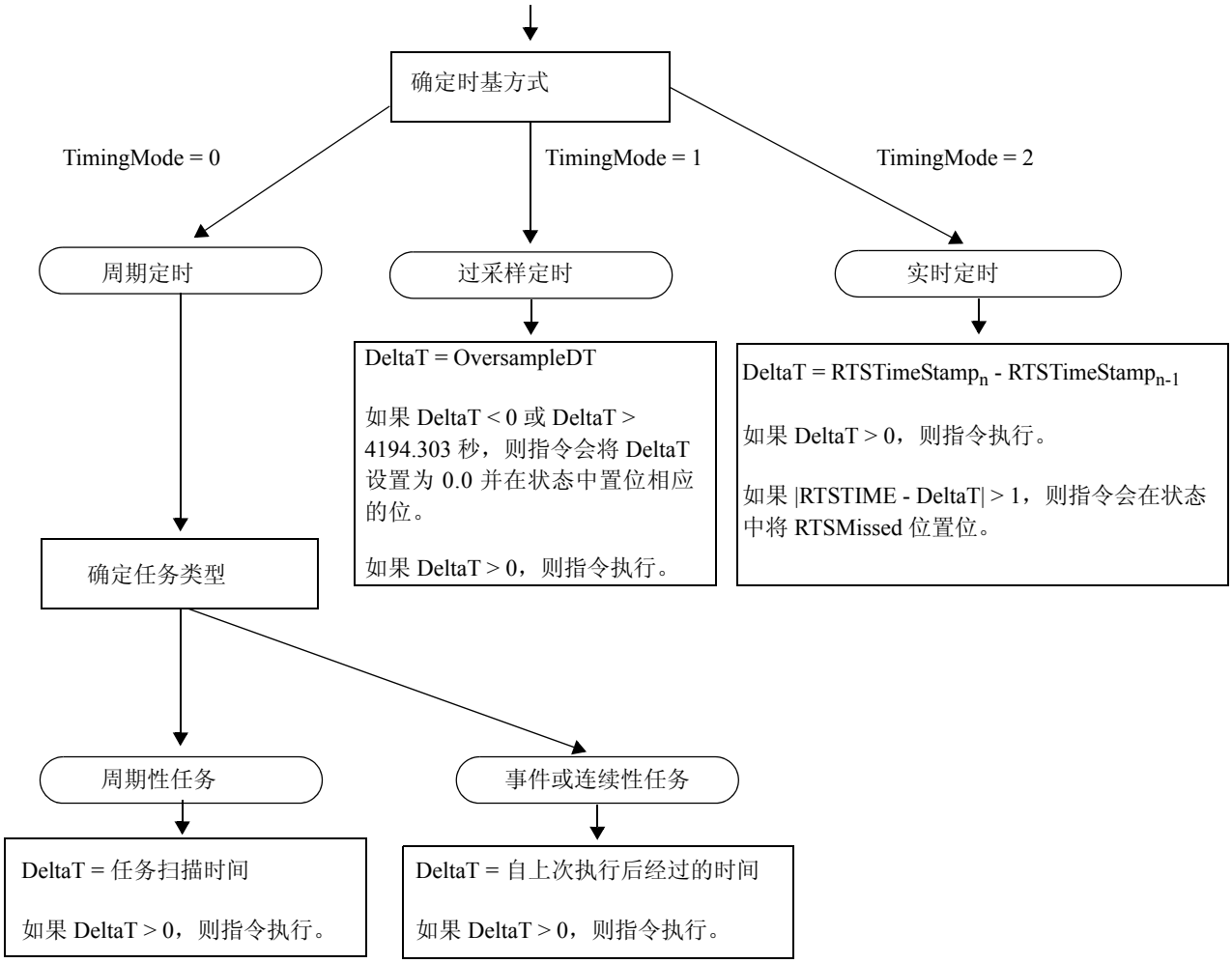
输入参数：	数据类型：	说明：
OversampleDT	REAL	过采样定时的执行时间。 DeltaT 值以秒为单位。如果 TimingMode = 1，则 OversampleDT = 0.0 会禁止控制算法的执行。如果无效，指令会设置 DeltaT = 0.0 并在状态中置位相应的位。 有效值 = 0 到 4194.303 秒 默认值 = 0.0
RTSTime	DINT	实时采样定时的模块更新周期。预计的 DeltaT 更新周期以毫秒为单位。更新周期通常为用于配置模块更新时间的值。如果无效，指令会在状态中置位相应的位并禁止 RTSMissed 检查。 有效值 = 1 到 32,767 毫秒 默认值 = 1
RTSTimeStamp	DINT	实时采样定时的模块时间戳值。上一次输入信号更新的相应时间戳值。此值用于计算 DeltaT。如果无效，指令会在状态中置位相应的位，并禁止执行控制算法和 RTSMissed 检查。 有效值 =1 到 32,767 毫秒 （从 32767 返回到 0） 1 次计数 = 1 毫秒 默认值 = 0

输出参数

输出参数：	数据类型：	说明：
DeltaT	REAL	更新之间的时间差值。通过控制算法计算过程输出时所用的时间 （单位为秒）。 周期方式：如果为周期性任务，则 DeltaT = 任务扫描速率，如果为事件或连续性任务，则 DeltaT = 自上一次执行指令后经过的时间。 过采样方式：DeltaT = OversampleDT 实时采样方式：$DeltaT = (RTSTimeStamp_n - RTSTimeStamp_{n-1})$
Status	DINT	功能块的状态。
TimingModeInv (Status.27)	BOOL	TimingMode 值无效。
RTSMissed (Status.28)	BOOL	仅用于实时采样方式。ABS | DeltaT - RTSTime | > 1 （0.001 秒）时置位。
RTSTimeInv (Status.29)	BOOL	RTSTime 值无效。
RTSTimeStampInv (Status.30)	BOOL	RTSTimeStamp 值无效。
DeltaTInv (Status.31)	BOOL	DeltaT 值无效。

定时方式概述

下图所示的是指令确定相应定时方式的方法。



程序 / 操作员控制

一些指令支持程序 / 操作员控制。这些指令包括：

- 增强型选择 (ESEL)
- 累加器 (TOT)
- 增强型 PID (PIDE)
- 斜坡 / 浸透 (RMPS)
- 离散型 2 态设备 (D2SD)
- 离散型 3 态设备 (D3SD)

通过程序 / 操作员控制可以从用户程序和操作界面设备中同时控制这些指令。在程序控制方式中，指令由程序输入控制，在操作员控制方式中，指令由操作员输入控制。

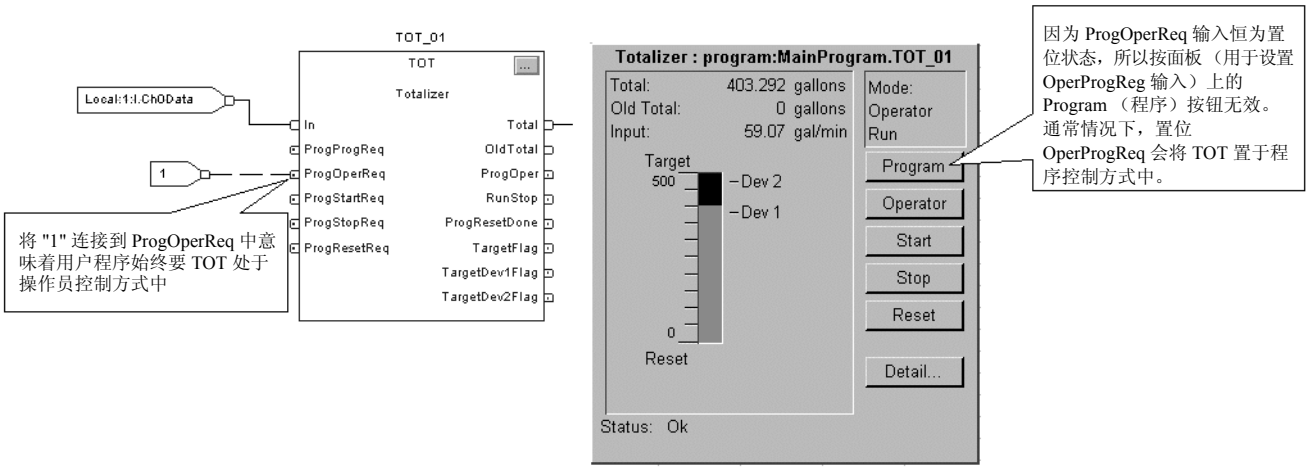
通过以下输入决定是由程序控制还是操作员控制：

输入：	说明：
.ProgProgReq	程序请求进入程序控制方式。
.ProgOperReq	程序请求进入操作员控制方式。
.OperProgReq	操作员请求进入程序控制方式。
.OperOperReq	操作员请求进入操作员控制方式。

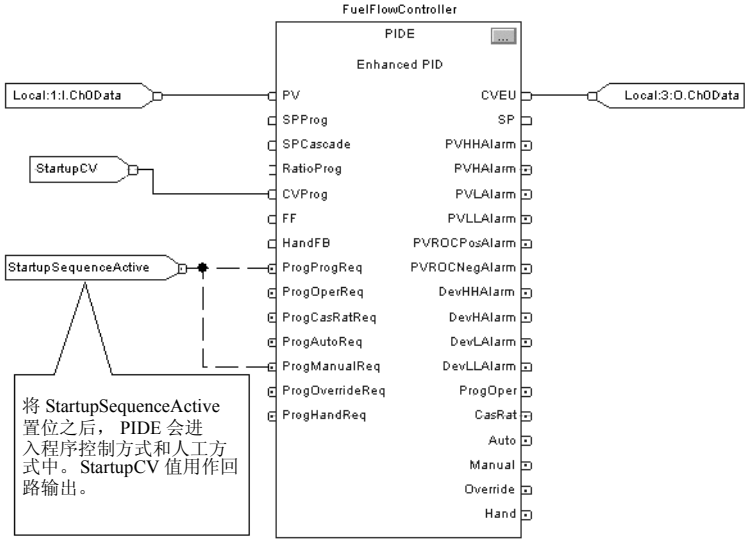
要确定指令是处于程序控制方式还是操作员控制方式，可检查 ProgOper 输出。如果置位 ProgOper，则指令处于程序控制方式；如果已将 ProgOper 清零，则指令处于操作员控制方式。

如果已同时置位两个输入请求位，则操作员控制优先于程序控制。例如，如果已同时置位 ProgProgReq 和 ProgOperReq，则指令受操作员控制。

程序请求输入优先于操作员请求输入。这样就能使用 ProgProgReq 和 ProgOperReq 输入将指令 “锁定”在所需的控制方式。例如，假定累加器指令始终用在操作员控制方式中，而用户程序绝不会控制累加器的运行和停止。在这种情况下，您可以将文本值 1 与 ProgOperReq 相连。这可以永远防止操作员通过从操作界面设备置位 OperProgReq 来将累加器设置成程序控制方式。



同样，始终置位 ProgProgReq 可以将指令 “锁定”在程序控制方式中。如果希望不必担心操作员会无意中控制指令，而由程序控制指令的操作，则这种方法可以用于自动启动序列。在此例中，您让程序在启动期间置位 ProgProgReq 输入，然后在启动完毕时将 ProgProgReq 输入清零。将 ProgProgReq 输入清零之后，在收到更改请求之前指令将一直处于程序控制方式中。例如，操作员可以从面板中置位 OperOperReq 输入，以便取得该指令的控制权。下例所示的是如何将指令锁定在程序控制方式中。



在执行指令时，对该指令的操作员请求输入始终会被清零。这使操作员界面可以通过仅置位所需方式的请求位来使用这些指令。您不必对操作员界面进行编程以复位请求位。例如，如果操作员界面置位对于 PIDE 指令的 OperAutoReq 输入，则在执行 PIDE 指令时，它会确定应作何种适当的响应并将 OperAutoReq 清零。

程序请求输入通常不会由指令清零，因为这些输入通常是作为指令的输入连接的。如果指令将这些输入清零，则输入会被连接的输入重新置位。在有些情况下您可能希望由指令将程序请求清零，并同时希望使用其他逻辑置位程序请求。在这种情况下，您可以将 ProgValueReset 输入置位，而指令会始终在其执行时将程序方式请求输入清零。

在此例中，另一程序中梯形逻辑的一个梯级用于在按某个按钮时将 ProgAutoReq 以单发方式锁存到 PIDE 指令。因为 PIDE 指令可以自动将程序方式请求清零，所以不必编写任何梯形图逻辑在执行程序后将

ProgAutoReq 清零，每当按下按钮时 PIDE 指令只会收到一个转到 Auto 的请求。

按 TIC101AutoReq 按钮时，会以单触发方式对 PIDE 指令 TIC101 锁存 ProgAutoReq。因为已将 TIC101 配置为 ProgValueReset 输入置位，所以在执行 PIDE 指令时，它会自动将 ProgAutoReq 清零。

结构化文本编程

简介

本附录描述了结构化文本编程特有的问题。仔细阅读这部分内容有助于您理解结构化文本程序的执行方式。

结构化文本语法

结构化文本是一种使用语句来定义执行内容的文本方式编程语言。结构化文本不区分字母大小写。结构化文本包含以下元素：

术语：	定义：		实例：
赋值 （参阅第 C-3 页）	使用赋值语句为标记赋值。 赋值运算符为 :=。 赋值语句以分号 “;” 结束。		*tag := expression;*
表达式 （参阅第 C-5 页）	表达式是完整的赋值或结构语句的一部分。表达式的计算结果为数值 （数值表达式）或真假状态（布尔表达式）。 表达式包含：		
	标记	用于存储数据的已命名内存区域（BOOL、 SINT、 INT、 DINT、 REAL、 string）。	*value 1*
	立即数	常数。	*4*
	运算符	在表达式中指定运算的符号或助记符。	*tag1 + tag2* *tag1 >= value1*
	函数	函数执行后会生成一个值。函数的操作数放在括号中。 函数和指令的语法类似，二者的区别在于函数只能用于表达式中。而指令不能用在表达式中。	*function(tag1)*

术语：	定义：	实例：
指令 （参阅第 C-12 页）	指令是独立的语句。 指令的操作数放在括号中。 不同的指令可能有 0、 1 或多个操作数。 指令执行后会生成一个或多个值，这些值是数据结构一部分。 指令以分号 “;”结束。 指令和函数的语法类似，二者的区别在于指令不能用在表达式中。而函数只能用于表达式中。	*function();* *instruction(operand);* *instruction(operand1, operand2,operand3);*
结构 （参阅第 C-13 页）	用于触发结构化文本代码的条件语句 （即其它语句）。 结构以分号 “;”结束。	IF...THEN CASE FOR...DO WHILE...DO REPEAT...UNTIL EXIT
注释 （参阅第 第 C28 页 页）	用于解释或阐述一段结构化文本代码功能的文本。 • 使用注释可提高结构化文本的可读性。 • 注释不影响结构化文本的执行。 • 注释可写在结构化文本中的任何位置。	*// 注释* （* *注释开始. . . 注释结束* *） */* 注释开始. . . 注释结束 *** /

结构化文本语法中允许输入空格。空格不影响结构化文本的执行。例如，以下两条语句的执行结果相同：

```
Tag_B:=Tag_A

Tag_B := Tag_A
```

赋值语句

可通过赋值语句更改标记内存储的值。赋值语句的语法如下：

tag := *expression*;

其中：

元素：	说明：	
tag	表示获得新值的标记 标记必须为 BOOL、 SINT、 INT、 DINT 或 REAL	
:=	赋值符号	
expression	表示要赋予标记的新值	
	如果 *tag* 的数据类型为：	**使用以下类型表达式：**
	BOOL	布尔表达式
	SINT INT DINT REAL	数值表达式
;	结束赋值	

*标记*将保持所赋值，直到该值被另一赋值语句改变。

表达式可以很简单，如立即数或其它标记名；也可能比较复杂，包括多个运算符和（或）函数。有关详细信息，请参阅第 C-5 页的“表达式”一节。

进行非保持性赋值

非保持性赋值语句与上述普通赋值语句的不同之处在于，每当控制器处于以下状态时，非保持性赋值语句中的标记将复位为 0：

- 进入 RUN 方式
- 如果 SFC 配置为自动复位，离开该 SFC 的步时 （仅适用于在步动作中嵌入赋值语句或通过 JSR 指令调用结构化文本程序的情况。）

非保持性赋值语句的语法如下：

tag [:=] *expression*;

其中：

元素：	说明：	
tag	表示获得新值的标记 标记必须为 BOOL、 SINT、 INT、 DINT 或 REAL	
[:=]	非保持性赋值符号	
expression	表示要赋予标记的新值	
	如果 *tag* 的数据类型为：	**使用以下类型表达式：**
	BOOL	布尔表达式
	SINT INT DINT REAL	数值表达式
;	结束赋值	

将 ASCII 字符赋值给字符串

通过赋值运算符将 ASCII 字符赋给字符串标记中 DATA 成员的元素。要进行字符赋值，需指定该字符的值或指定标记名称、 DATA 成员和字符的元素。例如：

正确语法：	***错误*** **语法：**
`string1.DATA[0]:= 65;`	`string1.DATA[0]:= A;`
`string1.DATA[0]:= string2.DATA[0];`	`string1 := string2;`

要向字符串标记中添加或插入一串字符，可使用下列任一 ASCII 字符串指令：

目的：	**所用指令：**
向字符串末尾添加字符	CONCAT
在字符串中插入字符	INSERT

表达式

表达式可以是标记名、等式或比较运算。使用以下任一元素书写表达式：

- 用于存储值 （变量）的标记名
- 您直接输入到表达式中的数值 （立即数）
- 函数，如：ABS、 TRUNC
- 运算符，如：+、 -、 <、 >、 And、 Or

书写表达式时应遵循以下规则：

- 可使用任意大小写字母组合。例如，“AND”的以下 3 种变体均可使用：AND、 And、 and。
- 对于较复杂的要求，可使用括号在表达式内嵌套多个表达式。这样能提高整个表达式的可读性，并使其按照期望的顺序执行。参阅第 第 C11 页 页的 “确定运算顺序”。

在结构化文本中可使用以下两种类型的表达式：

布尔表达式：生成布尔值 1 （真）或 0 （假）的表达式。

- 布尔表达式中使用布尔标记、关系运算符以及逻辑运算符进行数值比较或判断条件真假。例如，标记 1>65。
- 简单的布尔表达式可以是单个 BOOL 标记。
- 通常情况下使用布尔表达式作为其它逻辑的执行条件。

数值表达式：计算整数或浮点数值的表达式。

- 数值表达式使用算术运算符、算术函数和位运算符。例如，标记 1+5。
- 大多数情况下您可将数值表达式嵌套于布尔表达式中。例如，（标记 1+5） >65。

在下表中选择用于表达式的运算符：

如要：	目的：
计算算术值	“使用算术运算符和函数”，第 C-7 页。
比较两个值或字符串	“使用关系运算符”，第 C-8 页。
判断条件真假	“使用逻辑运算符”，第 C-10 页。
数值的位比较	“使用位运算符”，第 C-11 页。

使用算术运算符和函数

您可以在算术表达式中使用多个运算符和函数。

算术运算符用于计算新值。

目的：	运算符：	最优数据类型：
加	+	DINT、 REAL
减 / 求反	-	DINT、 REAL
乘	*	DINT、 REAL
指数 （x 的 y 次幂）	**	DINT、 REAL
除	/	DINT、 REAL
取模	MOD	DINT、 REAL

算术函数用于执行数学运算。可对函数指定常数、非布尔标记或表达式。

目的：	所用函数：	最优数据类型：
绝对值	ABS (*numeric_expression*)	DINT、 REAL
反余弦	ACOS (*numeric_expression*)	REAL
反正弦	ASIN (*numeric_expression*)	REAL
反正切	ATAN (*numeric_expression*)	REAL
余弦	COS (*numeric_expression*)	REAL
弧度转换为角度	DEG (*numeric_expression*)	DINT、 REAL
自然对数	LN (*numeric_expression*)	REAL
以 10 为底的对数	LOG (*numeric_expression*)	REAL
角度转换为弧度	RAD (*numeric_expression*)	DINT、 REAL
正弦	SIN (*numeric_expression*)	REAL
平方根	SQRT (*numeric_expression*)	DINT、 REAL
正切	TAN (*numeric_expression*)	REAL
截整	TRUNC (*numeric_expression*)	DINT、 REAL

例如：

使用此种格式：	实例：	
	适用于下列情况：	应写为：
value1 operator value2	如果 *gain_4* 和 *gain_4_adj* 均为 DINT 标记，且要求：“*gain_4* 加 15 并将结果存储到 *gain_4_adj* 中”。	`gain_4_adj := gain_4+15;`
operator value1	如果 alarm 和 *high_alarm* 均为 DINT 标记，且要求：“对 *high_alarm* 求反并将结果存储到 *alarm* 中。”	`alarm:= -high_alarm;`
function(numeric_expression)	如果 *overtravel* 和 *overtravel_POS* 均为 DINT 标记，且要求：“计算 *overtravel* 的绝对值并将结果存储到 *overtravel_POS* 中。”	`overtravel_POS := ABS(overtravel);`
value1 operator (function((value2+value3)/2)	如果 *adjustment* 和 *position* 均为 DINT 标记，*sensor1* 和 *sensor2* 均为 REAL 标记，且要求：“计算 *sensor1* 和 *sensor2* 的平均值的绝对值，加 *adjustment*，并将结果存储到 *position* 中。”	`position := adjustment + ABS((sensor1 + sensor2)/2);`

使用关系运算符

关系运算符用于比较两个值或字符串以得到真或假的结果。关系运算的结果是布尔值：

比较关系为：	结果：
true	1
false	0

使用以下关系运算符：

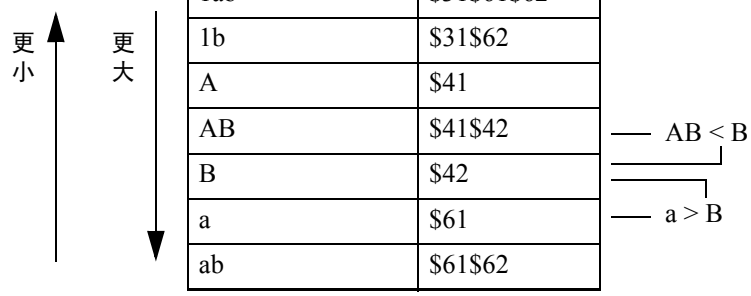
比较关系：	运算符：	最优数据类型：
等于	=	DINT、REAL、string
小于	<	DINT、REAL、string
小于等于	<=	DINT、REAL、string
大于	>	DINT、REAL、string
大于等于	>=	DINT、REAL、string
不等于	<>	DINT、REAL、string

例如：

使用此种格式：	实例：	
	适用于下列情况：	应写为：
value1 operator value2	如果 *temp* 是 DINT 标记，且要求：“如果 *temp* 小于 100，则 ...	`IF temp<100 THEN...`
stringtag1 operator stringtag2	如果 *bar_code* 和 *dest* 均为字符串标记，且要求：“如果 *bar_code* 等于 *dest*，则……”	`IF bar_code=dest THEN...`
char1 operator char2 要向表达式中直接输入 ASCII 字符，需输入字符的十进制值。	如果 *bar_code* 是字符串标记，且要求：“如果 *bar_code.DATA[0]* 等于 ‘A’，则……”	`IF bar_code.DATA[0]=65 THEN...`
bool_tag := *bool_expressions*	如果 *count* 和 *length* 均为 DINT 标记，*done* 是 BOOL 标记，且要求“果 *count* 大于或等于 *length*，计算完成。”	`done := (count >= length);`

如何对字符串进行计算 一个字符串是否小于或大于另一字符串由其 ASCII 字符对应的十六进制数值决定。

- 当两个字符串以类似电话簿中的方式进行排序时，字符串的排列顺序会决定其大小。



- 所有字符均匹配的字符串相等。
- ASCII 字符区分大小写。大写 “A”($41) 不等于小写 “a”($61)。

字符的十进制数值和十六进制代码，请参阅本手册的封底。

使用逻辑运算符

通过逻辑运算符可判断多个条件的真假。逻辑运算的结果是布尔值：

比较关系为：	结果：
true	1
false	0

使用以下逻辑运算符：

目的：	运算符：	数据类型：
逻辑与	&, AND	BOOL
逻辑或	OR	BOOL
逻辑异或	XOR	BOOL
逻辑非	NOT	BOOL

例如：

使用此种格式：	实例：	
	适用于下列情况：	应写为：
BOOLtag	如果 *photoeye* 是 BOOL 标记且要求：“如果 *photoeye_1* 为真，则……”	`IF photoeye THEN...`
NOT *BOOLtag*	如果 *photoeye* 是 BOOL 标记且要求：“如果 *photoeye* 为假，则……”	`IF NOT photoeye THEN...`
expression1 & *expression2*	如果 *photoeye* 是 BOOL 标记， *temp* 是 DINT 标记，且要求： “如果 *photoeye* 为真，而且 *temp* 小于 100ϒϑ，则……”。	`IF photoeye & (temp<100) THEN...`
expression1 OR *expression2*	如果 *photoeye* 是 BOOL 标记， *temp* 是 DINT 标记，且要求： “如果 *photoeye* 为真，或 *temp* 小于 100ϒϑ，则……”。	`IF photoeye OR (temp<100) THEN...`
expression1 XOR *expression2*	如果 *photoeye1* 和 *photoeye2* 均为 BOOL 标记，且要求：”如果： • *photoeye1* 为真而 *photoeye2* 为假，或 • *photoeye1* 为假而 *photoeye2* 为真， 则……”	`IF photoeye1 XOR photoeye2 THEN...`
BOOLtag := *expression1* & *expression2*	如果 *photoeye1* 和 *photoeye2* 均为 BOOL 标记， *open* 也是 BOOL 标记，且要求：“如果 *photoeye*1 和 *photoeye2* 均为真，则将 *open* 设置为真”。	`open := photoeye1 & photoeye2;`

使用位运算符

位运算符基于两个数值对其中一个数值进行位操作。

目的：	运算符：	最优数据类型：
位与	&, AND	DINT
位或	OR	DINT
位异或	XOR	DINT
按位求补	NOT	DINT

例如：

使用此种格式：	实例：	
	适用于下列情况：	应写为：
value1 operator value2	如果 *input1*、*input2* 和 *result1* 均为 DINT 标记，且要求：“计算 *input1* 和 *input2* 的位与结果。将结果存储到 *result1* 中。”	`result1 := input1 AND input2;`

确定运算顺序

写入表达式中的运算以预定的顺序执行，而非一定自左向右执行。

- 同级运算自左向右执行。
- 如果表达式中包含多个运算符或函数，可使用括号 “( )”将条件分组。这样可确保正确的执行顺序，并能提高表达式的可读性。

顺序：	运算：
D：1.	()
D：2.	函数 (…)
D：3.	**
D：4.	– （求反）
D：5.	NOT
D：6.	*、 /、 MOD
D：7.	+、 -（减）
D：8.	<、 <=、 >、 >=
D：9.	=, <>
D：10.	&, AND
D：11.	XOR
D：12.	OR

指令

结构化文本语句也可以是指令。参阅本手册开头部分的速查表了解结构化文本中可用的指令。每次扫描结构化文本指令都会使其执行。当结构的条件为真时，会执行结构中的结构化文本指令。如果结构的条件为假，则不会扫描其包含的语句。无需梯级条件或状态转换触发指令执行。

这与使用 EnableIn 触发执行的功能块指令不同。结构化文本指令的执行中相当于 EnableIn 始终处于已设置状态。

这与使用输入梯级条件触发执行的梯形图指令也是不同的。一些梯形图指令仅当梯级输入条件从 false 变为 true 时才会执行。这样的指令称为瞬变触发型梯形图指令。在结构化文本中，除非预先规定了指令的执行条件，否则每次对其扫描都会引发指令执行。

例如， ABL 指令是瞬变触发的梯形图指令。在此例中， ABL 指令仅在 tag_xic 从清零跳变到置位状态时执行。如果 tag_xic 保持置位状态或将 *tag_xic* 清零时， ABL 指令不执行。

在结构化文本中，如果您将此例写为：

```
IF tag_xic THEN ABL(0,serial_control);

END_IF;
```

则 ABL 指令在 tag_xic 置位时执行，而不仅是 tag_xic 从清零跳变到置位时才执行。

如果您希望仅当 tag_xic 从清零跳变到置位时 ABL 指令执行，必须设定结构化文本指令条件。使用上升沿触发执行。

```
osri_1.InputBit := tag_xic;
OSRI(osri_1);

IF (osri_1.OutputBit) THEN
     ABL(0,serial_control);
END_IF;
```

结构

结构可单独编写或嵌套在其它结构中。

如要：	使用结构：	可用编程语言：	参见页：
满足特定条件时执行某项操作	IF...THEN	结构化文本	C-14
基于数值选择要执行的操作	CASE...OF	结构化文本	C-17
对某项操作执行指定次数后再执行其它操作	FOR...DO	结构化文本	C-19
只要特定的条件为真，即重复执行某项操作	WHILE...DO	结构化文本	C-22
在条件为真之前，重复执行某项操作	REPEAT...UNTIL	结构化文本	C-25

一些关键字留作备用

以下结构不可用：

- GOTO
- REPEAT

RSLogix 5000 软件不支持以上结构。

IF...THEN

使用 IF...THEN 语句在满足特定条件时执行某项操作。

操作数：

```
IF bool_expression THEN
    <statement>;
END_IF;
```

结构文本

操作数：	类型：	格式：	输入：
bool_ expression	BOOL	标记 表达式	计算出布尔值（布尔表达式）的布尔型标记或表达式

说明： 其语法如下：

```
IF bool_expression1 THEN
      <statement>;          ◄── 当 bool_expression1 为真时执行的语句
                .
                .
                .
可选 ┐ ELSIF bool_expression2 THEN
      <statement>;          ◄── 当 bool_expression2 为真时执行的语句
                .
                .
                .
可选 ┐ ELSE
      <statement>;          ◄── 两个表达式均为假时执行的语句
                .
                .
                .
END_IF;
```

使用 ELSIF 或 ELSE 时遵循以下准则：

D：1. 要从多个语句组中进行选择，需添加一个或多个 ELSIF 语句。

- 每个 ELSIF 表示一条备选路径。
- 根据需求指定 ELSIF 路径个数。
- 控制器执行首个条件为真的 IF 或 ELSIF 语句，并跳过其余 ELSIF 和 ELSE 语句。

D：2. 要在所有 IF 或 ELSIF 条件均为假时执行某项操作，需添加一条 ELSE 语句。

下表总结了 IF、 THEN、 ELSIF 和 ELSE 语句的多种组合。

如要：	并且：	则使用以下结构
如果条件为真，执行语句	如果条件为假，不执行任何语句	IF...THEN
	如果条件为假，执行其它语句	IF...THEN...ELSE
根据输入条件从备选语句（或语句组）中选择	如果条件为假，不执行任何语句	IF...THEN...ELSIF
	如果所有条件均为假，执行缺省语句	IF...THEN...ELSIF...ELSE

例 1： IF...THEN

如要：	输入结构化文本：
如果 rejects > 3，则作如下设置：	`IF rejects > 3 THEN`
关闭传送带 (0)	`conveyor := 0;`
开启报警 (1)	`alarm := 1;`
	`END_IF;`

例 2： IF...THEN...ELSE

如要：	输入结构化文本：
如果传送带方向触点为前进 (1)，则作如下设置：	`IF conveyor_direction THEN`
指示灯灭	`light := 0;`
否则，指示灯亮	`ELSE`
	`light [:=] 1;`
	`END_IF;`

[:=] 要求控制器将 light 清零，前提是当控制器处于以下状态时：

- 进入 RUN 方式
- 如果 SFC 配置为自动复位，离开该 SFC 的步时 （仅适用于在步动作中嵌入赋值语句或通过 JSR 指令调用结构化文本程序的情况。）

例 3： IF...THEN...ELSIF

如要：	输入结构化文本：
如果糖料的下限开关为已超限 (on)，糖料的上限开关未超限 (on)，则作如下设置：	`IF Sugar.Low & Sugar.High THEN`
打开入口阀 (on)	`Sugar.Inlet [:=] 1;`
直到糖的上限开关超限 (off)	`ELSIF NOT(Sugar.High) THEN`
	`Sugar.Inlet := 0;`
	`END_IF;`

[:=] 要求控制器将 Sugar.Inlet 清零，前提是当控制器处于以下状态时：

- 进入 RUN 方式
- 如果 SFC 配置为自动复位，离开该 SFC 的步时 （仅适用于在步动作中嵌入赋值语句或通过 JSR 指令调用结构化文本程序的情况。）

例 4： IF...THEN...ELSIF...ELSE

如要：	输入结构化文本：
如果罐的温度 > 100	`IF tank.temp > 200 THEN`
则设置泵为慢速运行	`pump.fast :=1; pump.slow :=0; pump.off :=0;`
如果罐的温度 > 200	`ELSIF tank.temp > 100 THEN`
则设置泵为快速运行	`pump.fast :=0; pump.slow :=1; pump.off :=0;`
否则，关闭泵	`ELSE`
	`pump.fast :=0; pump.slow :=0; pump.off :=1;`
	`END_IF;`

CASE...OF

使用 CASE 基于数值选择要执行的操作。

操作数：

```
CASE numeric_expression OF
     selector1: statement;
     selectorN: statement;
ELSE
     statement;
END_CASE;
```

结构文本

操作数：	类型：	格式：	输入：
numeric_ expression	SINT INT DINT REAL	标记 表达式	计算出数值 （数值表达式）的标记或表达式
selector	SINT INT DINT REAL	立即数	与 *numeric_expression* 的类型相同

重要 使用 REAL 数值时需为选择器指定一个数值范围，因为 REAL 值在更多情况下是在一定范围内的数，而不是某个特定的数值。

说明： 其语法如下：

```
CASE numeric_expression OF
    selector1:  <statement>;   ← 当 numeric_expression = selector1 时执行的语句
                    .
                    .
                    .
    selector2:  <statement>;   ← 当 numeric_expression = selector2 时执行的语句
                    .
                    .
                    .
    selector3:  <statement>;   ← 当 numeric_expression = selector3 时执行的语句
                    .
                    .
                    .
ELSE                                                    （可选）
                <statement>;   ← 当 numeric_expression ≠ 任一选择器时执行的语句
                    .
                    .
                    .
END_CASE;
```

根据需求指定选择器的值 （路径）

获取有效的选择器数值，请参阅下页表格。

用于输入选择器数值的语法如下：

当选择器为：	输入：
单个值	*value*: *statement*
多个分散的值	*value1*, *value2*, *valueN* : <*statement*> 每个值之间使用逗号 (,) 隔开。
数值范围	*value1..valueN* : <*statement*> 使用两个句点 (..) 标识范围。
分散的值外加一个数值范围	*valuea, valueb, value1..valueN* : <*statement*>

实例

如要：	输入结构化文本：

如果配方号 = 1，则作如下设置：
成分 A 出口 1 = 打开 (1)
成分 B 出口 4 = 打开 (1)
如果配方号 = 2 或 3，则作如下设置：
成分 A 出口 4 = 打开 (1)
成分 B 出口 2 = 打开 (1)
如果配方号 = 4、 5、 6 或 7，则作如下设置：
成分 A 出口 4 = 打开 (1)
成分 B 出口 2 = 打开 (1)
如果配方号 = 8、11、12 或 13，则作如下设置：
成分 A 出口 1 = 打开 (1)
成分 B 出口 4 = 打开 (1)
否则，关闭所有出口 (0)
	`CASE recipe_number OF`
	`1: Ingredient_A.Outlet_1 :=1;`
	`Ingredient_B.Outlet_4 :=1;`
	`2,3: Ingredient_A.Outlet_4 :=1;`
	`Ingredient_B.Outlet_2 :=1;`
	`4..7: Ingredient_A.Outlet_4 :=1;`
	`Ingredient_B.Outlet_2 :=1;`
	`8,11..13 Ingredient_A.Outlet_1 :=1;`
	`Ingredient_B.Outlet_4 :=1;`
	`ELSE`
	`Ingredient_A.Outlet_1 [:=]0;`
	`Ingredient_A.Outlet_4 [:=]0;`
	`Ingredient_B.Outlet_2 [:=]0;`
	`Ingredient_B.Outlet_4 [:=]0;`
	`END_CASE;`

[:=] 要求控制器将 outlet 标记清零，前提是当控制器处于以下状态时：

- 进入 RUN 方式
- 如果 SFC 配置为自动复位，离开该 SFC 的步时 （仅适用于在步动作中嵌入赋值语句或通过 JSR 指令调用结构化文本程序的情况。）

FOR...DO

使用 FOR...DO 循环对某项操作执行指定次数后再执行其它操作。

操作数：

```
FOR count:= initial_value TO
final_value BY increment DO
    <statement>;
END_FOR;
```

结构文本

操作数：	类型：	格式：	说明：
count	SINT INT DINT	标记	在 FOR...DO 执行时用于存储计数位置
initial_value	SINT INT DINT	标记 表达式 立即数	必须是一个数值 指定计数的初始值
final_value	SINT INT DINT	标记 表达式 立即数	指定计数的结束值，决定何时退出循环
increment	SINT INT DINT	标记 表达式 立即数	（*可选*）每循环一次的计数增量 如果您未指定增量数值，计数增量默认为 1。

重要

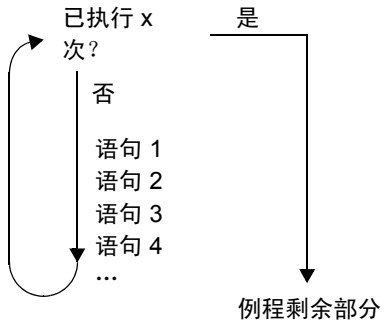
确保在单次扫描中重复循环的次数不能过多。

- 控制器在完成循环之前不执行程序中的任何其它语句。
- 如果循环完成所用时间超出该任务加密狗定时器的值，则产生一个主要错误。
- 考虑使用不同的结构，例如 IF...THEN。

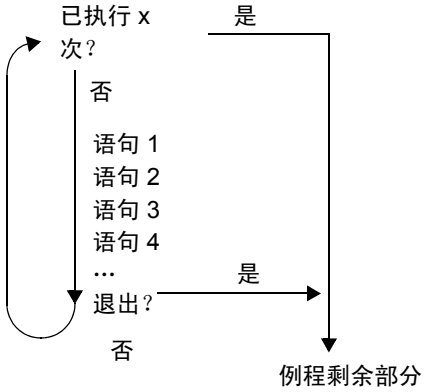
说明： 其语法如下：

```
FOR count := initial_value
    TO final_value
可选 { BY increment                    如果您未指定增量数值，循环增量默认为 1。
    DO
    <statement>;
    IF bool_expression THEN
可选     EXIT;                         如果需要提前退出循环，可使用其它语句（例如 IF...THEN 结构）作为 EXIT 语句退出条件。
    END_IF;
END_FOR;
```

下图所示为 FOR...DO 循环的执行方式，以及如何使用 EXIT 语句提前退出循环。



FOR...DO 循环执行指定的次数。



使用 EXIT 语句在计数值到达结束值前退出循环。

例 1：

如要：	输入结构化文本：

将布尔型数组的 0 - 31 位清零：
D：1. 初始化 *subscript* 标记为 0。
D：2. 将 *array[subscript]* 清零。例如，当 *subscript* = 5 时，将 *array[5]* 清零。
D：3. *subscript* 加 1。
D：4. 如果 *subscript* ≤ 31，重复步骤 2 和 3。否则，停止。
	`For subscript:=0 to 31 by 1 do`
	`  array[subscript] := 0;`
	`End_for;`

例 2：

如要：	输入结构化文本：

使用用户自定义数据类型 （结构）存储以下库存物品信息：
• 物品的条形码 ID （字符串数据类型）
• 物品的库存量 （DINT 数据类型）
 	
以上结构的数组中包含的元素可表示库存中每个不同的物品。要在数组中搜索指定产品 （使用其条形码）并确定其库存量。
D：1. 获取 *Inventory* 数组的大小 （物品数）并将结果存储在 *Inventory_Items* 中 （DINT 标记）。
D：2. 初始化 *position* 标记为 0。
D：3. 如果 *Barcode* 与数组中某个物品的 ID 相匹配，则：
C： a. 设置 *Quantity* 标记等于 *Inventory[position].Qty*。得到该物品的库存量。
C： b. 结束。
Barcode 是字符串标记，用于存储您查找的物品的条形码。例如，当 *position* = 5 时，对 *Barcode* 和 *Inventory[5].ID* 进行比较。
D：3. *position* 加 1。
D：4. 如果 *position* ≤ (*Inventory_Items -1)*，重复步骤 3 和 3。由于元素号为从 0 开始，因此最末元素的编号比数组的元素个数少 1。
否则，停止。
	`SIZE(Inventory,0,Inventory_Items);`
	`For position:=0 to Inventory_Items - 1 do`
	`  If Barcode = Inventory[position].ID then`
	`  Quantity := Inventory[position].Qty;`
	`  Exit;`
	`  End_if;`
	`End_for;`

WHILE...DO

只要特定的条件为真，使用 WHILE...DO 循环即重复执行某项操作。

操作数：

```
WHILE bool_expression DO
    <statement>;
END_WHILE;
```

结构文本

操作数：	类型：	格式：	输入：
bool_ expression	BOOL	标记 表达式	计算出布尔值的布尔型标记或表达式

重要

确保在单次扫描中重复循环的次数不能过多。

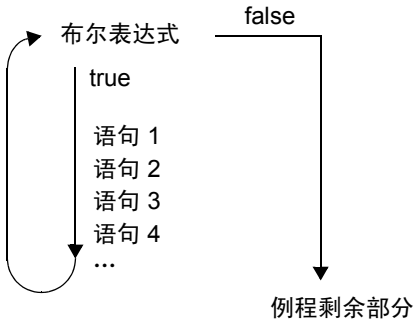
- 控制器在完成循环之前不执行程序中的任何其它语句。
- 如果循环完成所用时间超出该任务看门狗定时器的值，则产生一个主要错误。
- 考虑使用不同的结构，例如 IF...THEN。

说明： 其语法如下：

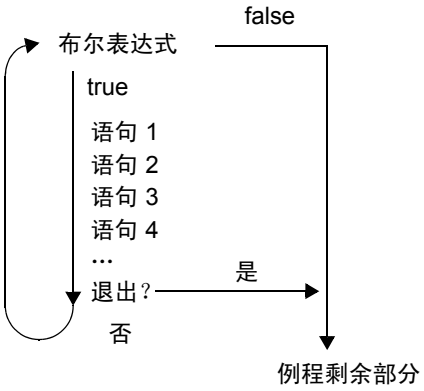
```
WHILE bool_expression1 DO
   <statement>;          <-- 当 bool_expression1 为真时执行的语句
   IF bool_expression2 THEN
     EXIT;               <-- 如果需要提前退出循环，可使用其它语句（例如 IF...THEN 结构）作为 EXIT 语句退出条件。
   END_IF;
END_WHILE;
```

可选

下图所示为 WHILE...DO 循环的执行方式，以及如何使用 EXIT 语句提前退出循环。



bool_expression **为真时，控制器仅执行 WHILE...DO 循环内的语句。**



使用 EXIT 语句在条件为真之前退出循环。

例 1：

如要：	输入结构化文本：

WHILE...DO 循环首先对其条件进行判断。
如果条件为真，则控制器执行循环中的语句。
这与 REPEAT...UNTIL 循环不同，REPEAT...UNTIL 循环首先执行结构中的语句，
再次执行该语句之前需确定条件是否为真。
REPEAT...UNTIL 循环中的语句至少会执行一次。
WHILE...DO 循环中的语句可能从不执行。
	`pos := 0;`
	`While ((pos <= 100) & structarray[pos].value <> targetvalue)) do`
	`pos := pos + 2;`
	`String_tag.DATA[pos] := SINT_array[pos];`
	`end_while;`

例 2：

如要：	输入结构化文本：

将 ASCII 字符从 SINT 数组移动到字符串标记中。（在 SINT 数组中，每个元素包含一个字符。）到达结束符时停止。
D：1. 将 *Element_number* 初始化为 0。
D：2. 计算 *SINT_array*（包含 ASCII 字符的数组）中元素的个数并将结果存储在 *SINT_array_size*（DINT 标记）中。
D：3. 如果 *SINT_array[element_number]* 对应的字符为 13（结束符的十进制数值），则停止。
D：4. 设置 *String_tag[element_number]* 等于 *SINT_array[element_number]* 对应的字符。
D：5. *element_number* 加 1。可使控制器检查 *SINT_array* 中的下一个字符。
D：6. 设置 *String_tag* 的长度值为 *element_number*。（该值记录 *String_tag* 中现有字符数。）
D：7. 如果 *element_number* = *SINT_array_size*，则停止。（已执行到数组末尾但无结束符。）
D：8. 转至步骤 3。
	`element_number := 0;`
	`SIZE(SINT_array, 0, SINT_array_size);`
	`While SINT_array[element_number] <> 13 do`
	`  String_tag.DATA[element_number] := SINT_array[element_number];`
	`  element_number := element_number + 1;`
	`  String_tag.LEN := element_number;`
	`  If element_number = SINT_array_size then`
	`  exit;`
	`  end_if;`
	`end_while;`

REPEAT...UNTIL

使用 REPEAT...UNTIL 循环重复执行某项操作，直至条件为真。

操作数：

```
REPEAT
    <statement>;
UNTIL bool_expression
END_REPEAT;
```

结构文本

操作数：	类型：	格式：	输入：
bool_ expression	BOOL	标记 表达式	计算出布尔值 （布尔表达式） 的布尔型标记或表达式

重要

确保在单次扫描中重复循环的次数不能过多。

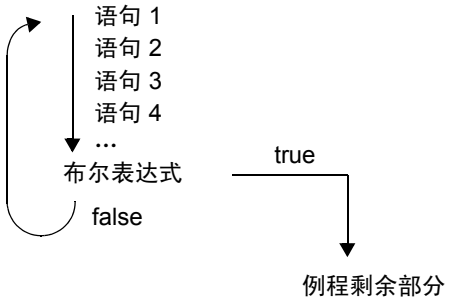
- 控制器在完成循环之前不执行程序中的任何其它语句。
- 如果循环完成所用时间超出该任务加密狗定时器的值，则产生一个主要错误。
- 考虑使用不同的结构，例如 IF...THEN。

说明： 其语法如下：

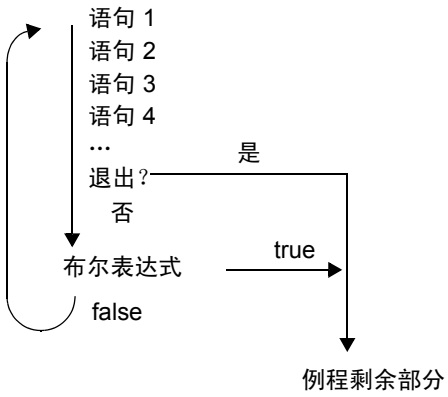
```
REPEAT
   <statement>;          ← 当 bool_expression1 为假时执行的语句
   IF bool_expression2 THEN
     EXIT;               ← 如果需要提前退出循环，可使用其它语句 （例如 IF...THEN 结构） 作为 EXIT 语句退出条件。
   END_IF;
UNTIL bool_expression1
END_REPEAT;
```

可选

下图所示为 REPEAT...UNTIL 循环的执行方式，以及如何使用 EXIT 语句提前退出循环。



bool_expression **为假时，控制器仅执行 REPEAT...UNTIL 循环内的语句。**



使用 EXIT 语句在条件为假之前退出循环。

例 1：

如要：	输入结构化文本：

REPEAT...UNTIL 循环首先执行结构中的语句，再次执行该语句之前需确定条件是否为真。
这与 WHILE...DO 循环不同，因为 WHILE...DO 循环先对其执行条件进行判断。如果条件为真，则控制器执行循环中的语句。REPEAT...UNTIL 循环中的语句至少会执行一次。WHILE...DO 循环中的语句可能从不执行。
	`pos := -1;`
	`REPEAT`
	`  pos := pos + 2;`
	`UNTIL ((pos = 101) OR (structarray[pos].value = targetvalue))`
	`end_repeat;`

例 2：

如要：	输入结构化文本：

将 ASCII 字符从 SINT 数组移动到字符串标记中。（在 SINT 数组中，每个元素包含一个字符。）到达结束符时停止。
D：1. 将 *Element_number* 初始化为 0。
D：2. 计算 *SINT_array*（包含 ASCII 字符的数组）中元素的个数并将结果存储在 *SINT_array_size*（DINT 标记）中。
D：3. 设置 *String_tag[element_number]* 等于 *SINT_array[element_number]* 对应的字符。
D：4. *element_number* 加 1。可使控制器检查 *SINT_array* 中的下一个字符。
D：5. 设置 *String_tag* 的长度值为 *element_number*。（该值记录 *String_tag* 中现有字符数。）
D：6. 如果 *element_number* = *SINT_array_size*，则停止。（已执行到数组末尾但无结束符。）
D：7. 如果 *SINT_array[element_number]* 对应的字符为 13（结束符的十进制数值），则停止。否则转至步骤 3。
	`element_number := 0;`
	`SIZE(SINT_array, 0, SINT_array_size);`
	`Repeat`
	`  String_tag.DATA[element_number] := SINT_array[element_number];`
	`  element_number := element_number + 1;`
	`  String_tag.LEN := element_number;`
	`  If element_number = SINT_array_size then`
	`  exit;`
	`  end_if;`
	`Until SINT_array[element_number] = 13`
	`end_repeat;`

注释

为使结构化文本更具可读性，您可以在其中添加注释。

- 通过注释用简单的文字来描述结构化文本的工作方式。
- 注释不影响结构化文本的执行。

在结构化文本中添加注释：

添加注释：	使用以下格式：
单行	// *注释*
结构化文本的某一行末尾	(*comment*) /*comment*/
结构化文本的某一行中	(*comment*) /*comment*/
多行注释	(* *注释开始...注释结束* *) /* *注释开始...注释结束* */

例如：

格式：	实例：

// *注释*	**行起始**
	// 检查传送带方向
	IF conveyor_direction THEN...
	行尾
	ELSE // 如果传送带不动，报警灯亮
	light := 1;
	END_IF;

(**comment**)	Sugar.Inlet[:=]1;(* 打开入口 *) IF Sugar.Low (* 低电平 LS*)& Sugar.High (* 高电平 LS*)THEN... (* 控制循环泵的速度。该速度取决于罐内温度。 *) IF tank.temp > 200 THEN...
/**comment**/	Sugar.Inlet:=0;/* 关闭入口 */ IF bar_code=65 /*A*/ THEN... /* 获取 Inventory 数组中的元素个数，并将其存储到 Inventory_Items 标记中 */ SIZE(Inventory,0,Inventory_Items);

功能块面板控件

简介

RSLogix 5000 编程软件包括某些功能块指令的面板 （控件）。这些面板是 Active-X 控件，可以在 RSView SE 或 RSView32 软件或其他任何可以用作 Active-X 容器的应用程序中使用。对于 RSView SE 软件，面板通过 RSLinx Enterprise 软件与控制器进行通讯。对于 RSView 32 或其他第三方应用程序，面板则通过 RSLinx OPC 服务器与控制器进行通信。

重要事项 RSLogix 5000 编程软件无法作为面板的容器。您必须使用 RSView SE 或 RSView 32 之类的软件作为面板的容器。

下列指令具有面板：

指令：	参见页：
报警 (ALM)	D-6
增强型选择 (ESEL)	D-8
累加器 (TOT)	D-9
升温 / 保温 (RMPS)	D-11
离散 2 态设备 (D2SD)	D-14
离散 3 态设备 (D3SD)	D-16
增强型 PID (PIDE)	D-18

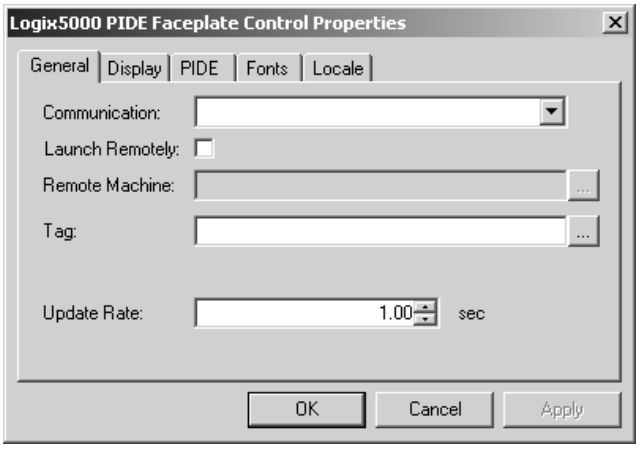
可通过在容器应用程序 （如 RSView SE 或 RSView32 软件）中打开的属性页对面板进行配置。

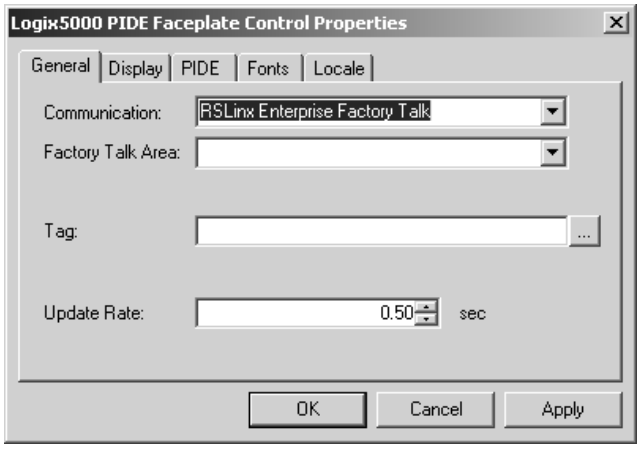
所有面板均有四个属性页。

- general （常规）
- display （显示）
- server （服务器）
- fonts （字体）

配置常规属性

常规属性页可以决定控件的运行方式。





属性页要素：	说明：
Communication （通信）	选择 RSLinx Classic OPC Server 或 RSLinx Enterprise Factory Talk。 如果选择 RSLinx Classis OPC Server，则还需指定： • 是否远程启动 • 远程计算机的访问路径 如果选择 RSLinx Enterprise Factory Talk，则还需指定： • Factory Talk 区域
Tag （标记）	指定用于连接该控件的功能块指令。
Update Rate （更新速率）	该选项用于配置控件的更新速率 （单位为秒）。使用数值调节钮控件以 0.25 秒为增量修改速率。 默认值 = 1.00 秒

重要事项

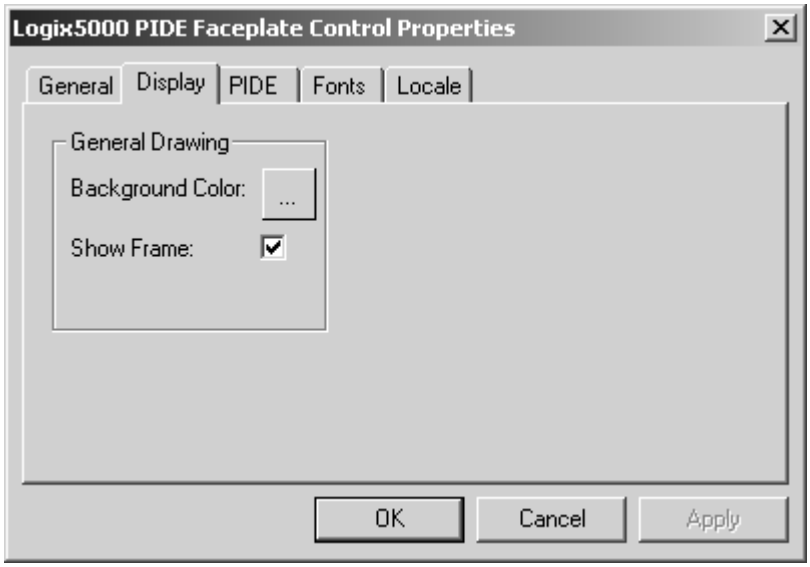
以上屏幕中，块标记示例显示的是控制器级的标记名称。默认情况下，功能块在插入后会自动指定一个程序级块标记。要指定名为 PID1 的程序级块标记，输入：

program: *program_name*.PID1

其中 *program_name* 为程序名。

配置显示属性

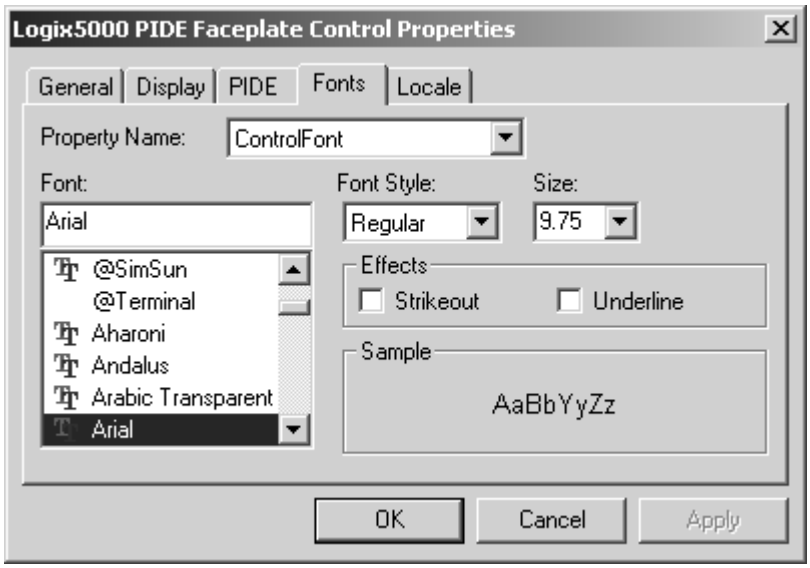
显示属性页可以设定常规屏幕属性。



属性页要素：	说明：
Background Color （背景色）	此按钮用于设置面板的背景色。 默认值 = 浅灰
Show Frame （显示边框）	此选项可显示或隐藏控件的三维边框。这使用户能够将该控件与显示器上的其他项目区分开来。 默认值 = 选中

配置字体属性

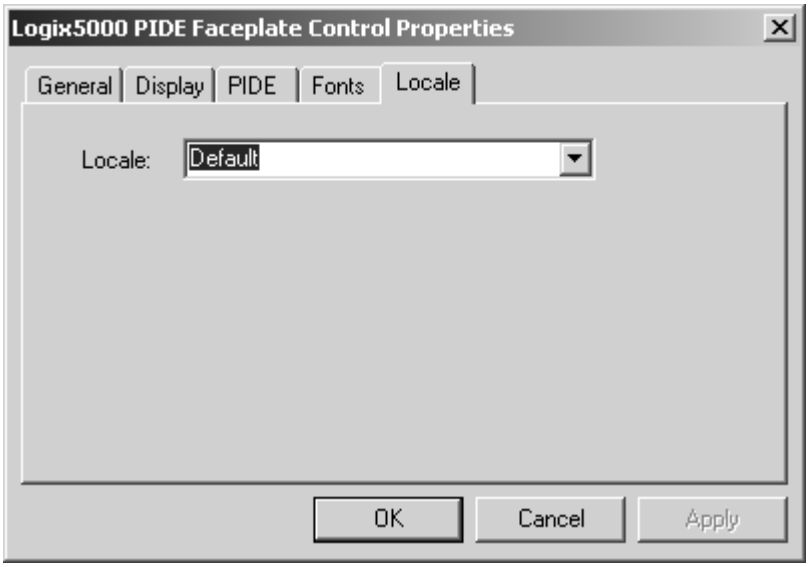
字体属性页可以设定显示在面板上的字体。配置将用于面板的主要部分的 ControlFont 以及用于面板刻度和其他次要部分的 MinorFont。



属性页要素：	说明：
Property Name （属性名）	使用此下拉列表选择要配置的字体。可选择 ControlFont 或 MinorFont。 默认值 = ControlFont
Font （字体）	选择控件的字体。列表中包含系统的所有可用字体。 默认值 = Arial
Size （字号）	配置字体的大小。 ControlFont 默认值 = 10.5 磅 MinorFont 默认值 = 8.25 磅
Effects （效果）	选择是否对字体添加 underline （下划线）和 （或） strikeout （删除线）。 默认值 = 均未选中

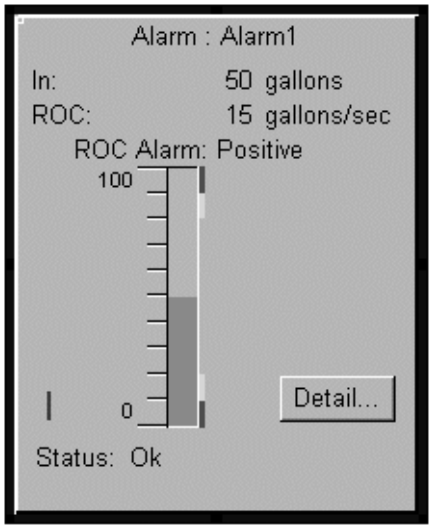
配置地区属性

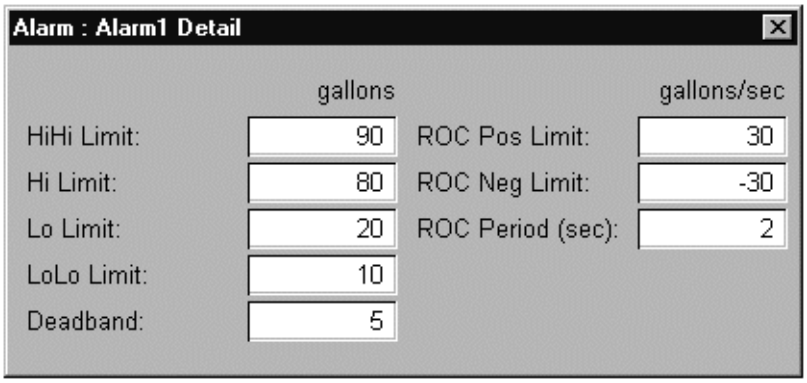
地区属性页可以确定语言要求。



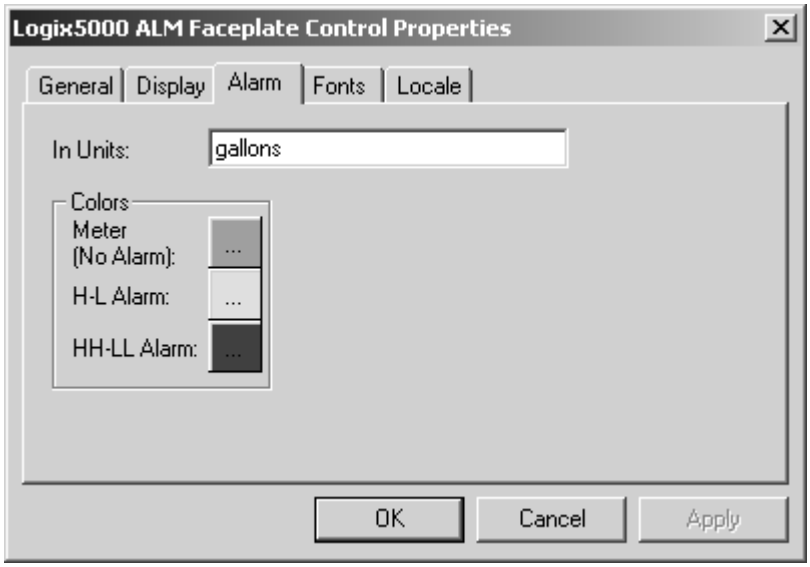
属性页要素：	说明：
Locale （地区）	选择语言： • 英语 • 葡萄牙语 • 法语 • 意大利语 • 德语 • 西班牙语

ALM 控件



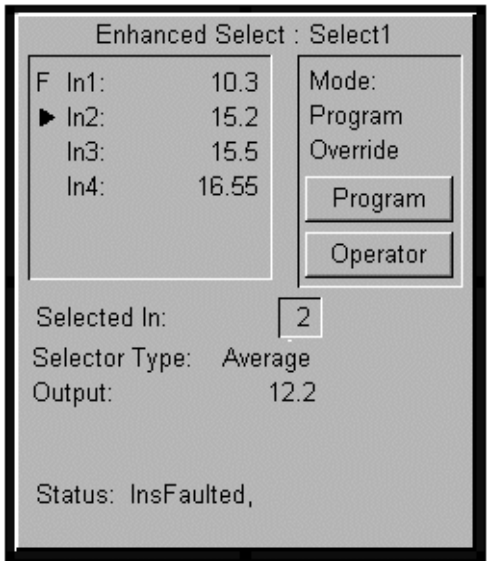
控制面板要素：	用于显示：
In （输入）	当前的输入值。
ROC （变化率）	变化率的值。 如果 ROCPosAlarm 或 ROCNegAlarm 置位，则文本会变成红色。将光标指向控件时会显示工具提示 "Rate Of Change" （变化率）。
报警计	功能块的输入值与其报警极限值的关系。 如果 HAlarm 或 LAlarm 值置位，报警条为黄色。同样，如果 HHAlarm 或 LLAlarm 值置位，报警条为红色。若无报警，则报警条为绿色。
报警标记条	HHLim、 HLim、 LLim 和 LLLim 的值。 HHLim 和 LLLim 报警条为红色， HLim 和 LLim 报警条为黄色。
报警计刻度	报警条的刻度。 刻度上限 = HHLim + Deadband。刻度下限 = LLLim – Deadband。
Detail （详细信息）按钮	弹出详细信息对话框。 
Status （状态）	功能块中所有已置位的状态位。 如果所有位均未置位，则状态显示 "OK" （正常）。

ALM 控件还具有以下附加属性页。



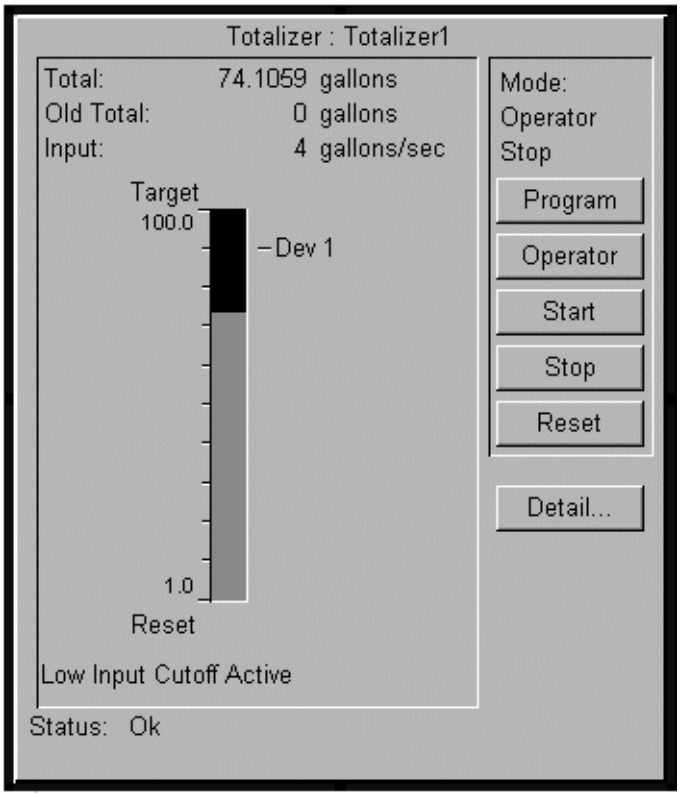
配置属性：	指定：
In Units （输入单位）	控件上 In （输入）字段的单位。
Meter Color （报警计颜色）	当前没有报警时的报警条颜色。
H-L Color （H-L 颜色）	指令处于下限或上限报警状态时的报警条颜色。
HH-LL Color （HH-LL 颜色）	指令处于最低下限或最高上限报警状态时的报警条颜色。

ESEL 控件

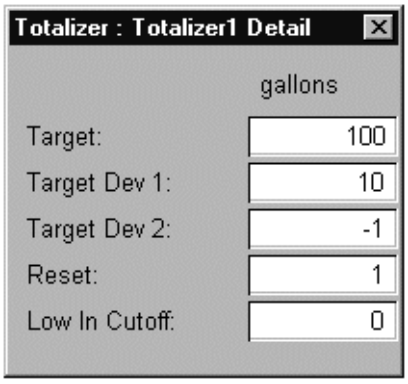


控制面板要素：	显示：
Mode （方式）	功能块的方式。
Input （输入）	功能块的输入。 显示数目 (1-6) 取决于可用输入通道的个数。
故障指示	特定输入出现故障时，在输入的左侧显示字母 "F"。
已选择指示	在输入的左侧显示三角形符号，指示已选择的输入。
Program （程序）按钮	单击此按钮将 OperProgReq 置位。
Operator （操作员）按钮	单击此按钮将 OperOperReq 置位。
Selected In （已选输入）	SelectedIn 的值。
Selector Type （选择器类型）	选择方式。
Output （输出）	输出值。
Status （状态）	功能块中所有已置位的状态位。 如果所有位均未置位，则状态显示 "OK" （正常）。

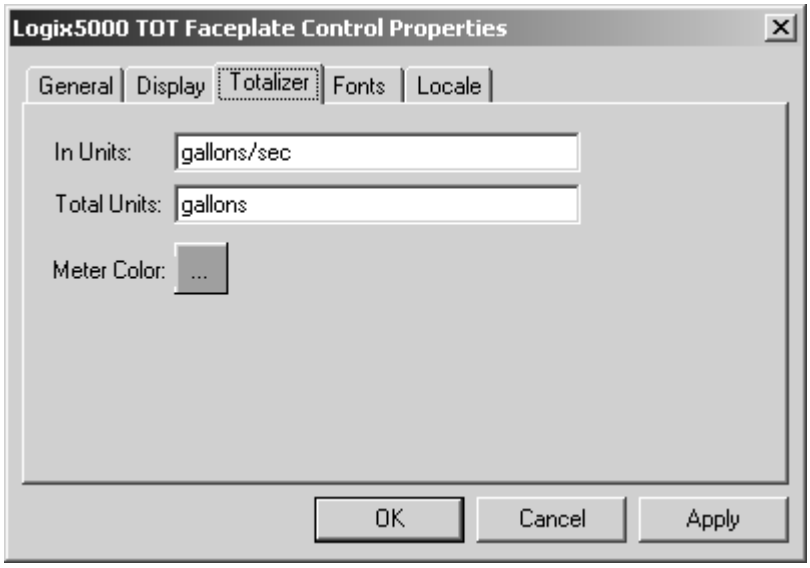
TOT 控件



控制面板要素：	**显示：**
Mode （方式）	功能块的方式。
Total （累加值）	累加值。
Old Total （上次累加值）	上次累加值。
Input （输入）	输入值。
总量程	累加值的范围。
Target Dev1 （目标偏离量 1）和 Dev2 （偏离量 2）刻度	TargetDev1 和 TagetDev2 的值。
总刻度	总量程的刻度。 刻度上限 = Target。刻度下限 = Reset。
Program （程序）按钮	单击此按钮将 OperProgReq 置位。
Operator （操作员）按钮	单击此按钮将 OperOperReq 置位。
Start （启动）按钮	单击此按钮将 OperStartReq 置位。
Stop （停止）按钮	单击此按钮将 OperStopReq 置位。
Reset （复位）按钮	单击此按钮将 OperResetReq 置位。

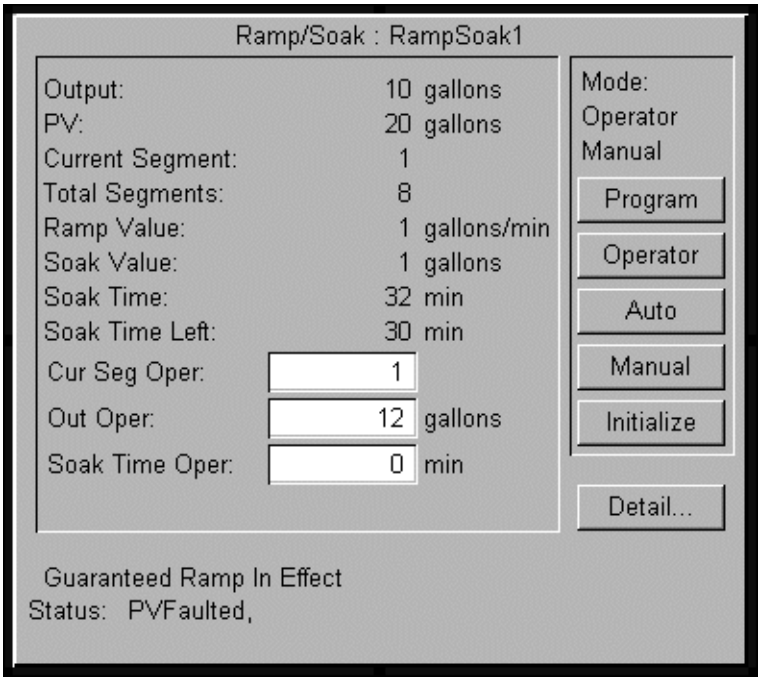
控制面板要素：	显示：
Detail （详细信息）按钮	弹出详细信息对话框。 
Low Input Cutoff Active （低输入截止激活）	仅在 LowInCutoffFlag 置位时才会显示 "Low Input Cutoff Active" （低输入截止激活）。
Status （状态）	功能块中所有置位的状态位。 如果所有位均未置位，则状态显示 "OK" （正常）。

TOT 控件还具有以下附加属性页。

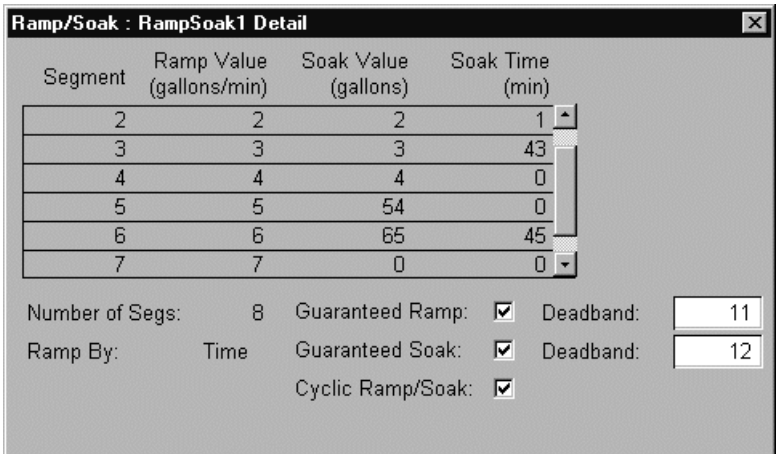


配置属性：	指定：
In Units （输入单位）	控件上 In （输入）字段的单位。
Total Units （累加值单位）	控件上 Total （累加值）和 Old Total （上次累加值）字段的单位。
Meter Color （报警计颜色）	报警条的颜色。

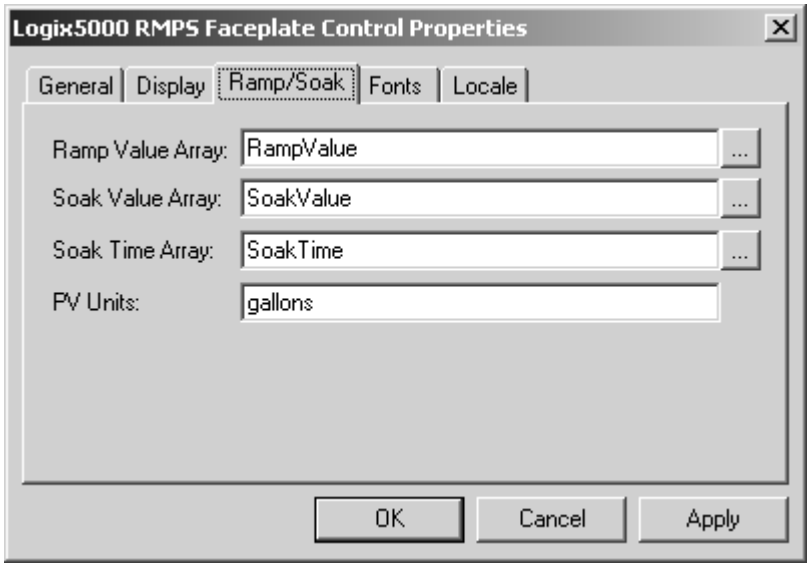
RMPS 控件



控制面板要素：	显示：
Mode （方式）	功能块的方式。
Output （输出）	输出值。
PV （过程变量）	PV （过程变量）值。
Current Segment （当前段）	当前段的值。
Ramp Value （升温值）	当前段的 RampValue[ ] 值。
Soak Value （保温值）	当前段的 SoakValue[ ] 值。
Soak Time （保温时间）	当前段的 SoakTime[ ] 值。
Soak Time Left （剩余保温时间）	SoakTimeLeft 的值。
Program （程序）按钮	单击此按钮将 OperProgReq 置位。
Operator （操作员）按钮	单击此按钮将 OperOperReq 置位。
Auto （自动）按钮	单击此按钮将 OperAutoReq 置位。
Manual （人工）按钮	单击此按钮将 OperManualReq 置位。
Initialize （初始化）按钮	单击此按钮将 Initialize 置位。 仅当功能块处于 Operator Manual （操作员人工）方式时才会启用此按钮。

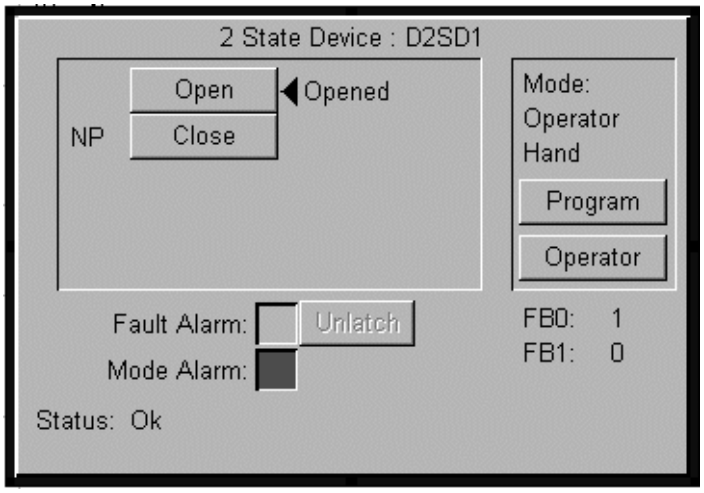
控制面板要素：	显示：
Detail （详细信息）按钮	弹出详细信息对话框。 
Cur Seg Oper （当前操作员段数）	CurrentSegOper 的值。
Out Oper （操作员输出）	OutOper 的值。
Soak Time Oper （操作员设定保温时间）	SoakTimeOper 的值。
Guaranteed Ramp or Soak in Effect （确保升温或保温有效）	相应的 GuarRamp 或 GuarSoak 位置位时显示 Guaranteed Ramp in Effect （确保升温有效）或 Guaranteed Soak in Effect （确保保温有效）。
Status （状态）	功能块中所有置位的状态位。 如果所有位均未置位，则状态显示 "OK" （正常）。

RMPS 控件还具有以下附加属性页。



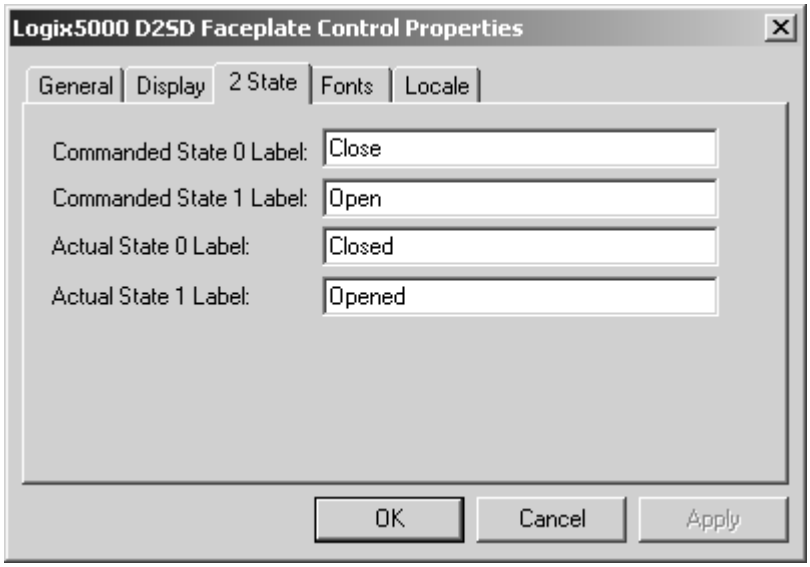
配置属性：	指定：
Ramp Value Array （升温值数组）	控制器中包含升温值的数组。
Soak Value Array （保温值数组）	控制器中包含保温值的数组。
Soak Time Array （保温时间数组）	控制器中包含保温时间的数组。
PV Units （过程变量单位）	显示在控件中的单位。

D2SD 控件



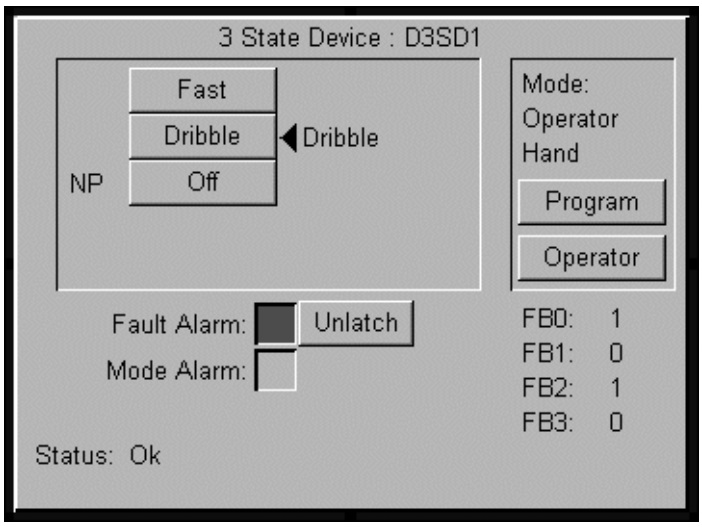
控制面板要素：	显示：
Mode （方式）	功能块的方式。
State （状态）按钮	在控制面板属性页中定义的 Commanded State Label （命令状态标签）的打开或关闭状态。 上方的按钮可将 Oper1Req 置位。下方的按钮可将 Oper0Req 置位。单击按钮时，设置 OperReq 字段为特定状态。
命令状态指示	指向请求的按钮得到其状态的命令状态值。
实际状态指示	实际状态值。 如果 DeviceStatus 置位，则实际状态为预定状态。
非允许指示	如果该状态的 StatePerm 未置位，则按钮左侧显示 “NP”。
Fault Alarm （故障报警）指示	指示 FaultAlarm 是否置位。
Mode Alarm （方式报警）指示	指示 ModeAlarm 是否置位。
Unlatch （解锁）按钮	FaultAlmUnlatch 的状态。 单击按钮将 FaultAlmUnlatch 置位。仅当 FaultAlarm 和 FaultAlmLatch 置位时才会启用此按钮。
Program （程序）按钮	单击此按钮将 OperProgReq 置位。
Operator （操作员）按钮	单击此按钮将 OperOperReq 置位。
FB1	FB1 的值。
FB0	FB0 的值。
Status （状态）	功能块中所有置位的状态位。 如果所有位均未置位，则状态显示 “OK”（正常）。

D2SD 控件还具有以下附加属性页。



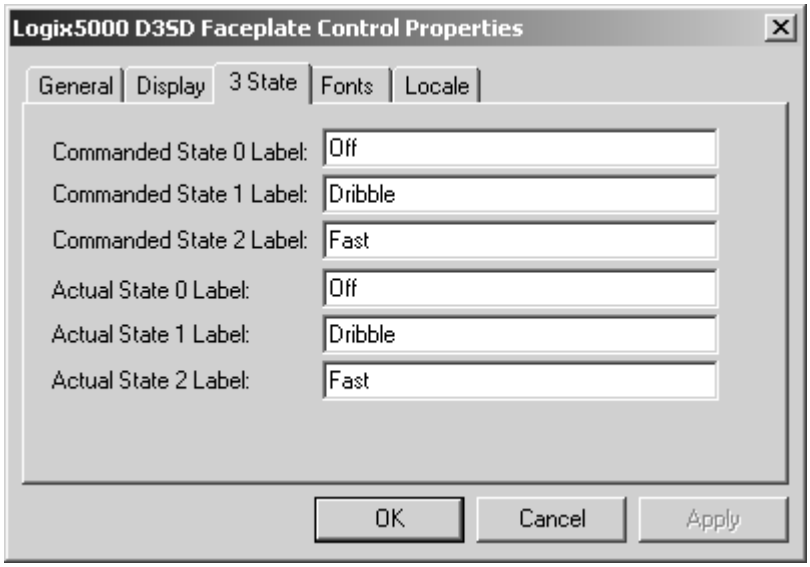
配置**属性：**	**指定：**
Commanded State 0 Label （命令状态 0 标签）	命令状态 0 的标签。
Commanded State 1 Label （命令状态 1 标签）	命令状态 1 的标签。
Actual State 0 Label （实际状态 0 标签）	实际状态 0 的标签。
Actual State 1 Label （实际状态 1 标签）	实际状态 1 的标签。

D3SD 控件



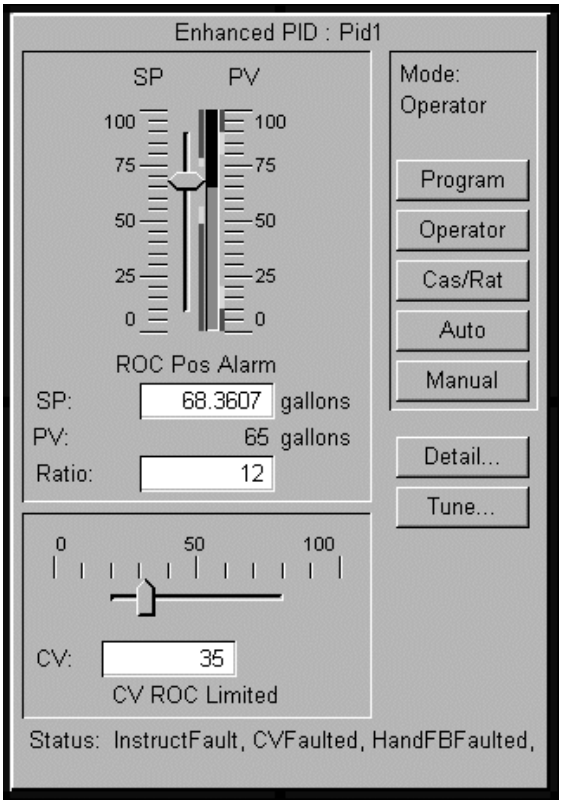
控制面板要素：	显示：
Mode （方式）	功能块的方式。
State （状态）按钮	在控制面板属性页中定义的 Commanded State Label （命令状态标签）的打开或关闭状态。 上方的按钮将 Oper2Req 置位。中间的按钮将 Oper1Req 置位。下方的按钮将 Oper0Req 置位。单击按钮时，将 OperReq 字段置位为特定状态。
命令状态指示	指向请求的按钮得到其状态的命令状态的值。
实际状态指示	实际状态值。 如果 DeviceStatus 置位，则实际状态为预定状态。
非允许指示	如果该状态的 StatePerm 未置位，则按钮左侧显示 "NP"。
Fault Alarm （故障报警）指示	指示 FaultAlarm 是否置位。
Mode Alarm （方式报警）指示	指示 ModeAlarm 是否置位。
Program （程序）按钮	单击此按钮将 OperProgReq 置位。
Operator （操作员）按钮	单击此按钮将 OperOperReq 置位。
FB3	FB3 的值。
FB2	FB2 的值。
FB1	FB1 的值。
FB0	FB0 的值。
Unlatch （解锁）按钮	FaultAlmUnlatch 的状态。 单击按钮将 FaultAlmUnlatch 置位。仅当 FaultAlarm 和 FaultAlmLatch 置位时才会启用此按钮。
Status （状态）	功能块中所有置位的状态位。 如果所有位均未置位，则状态显示 "OK" （正常）。

D3SD 控件还具有以下附加属性页。

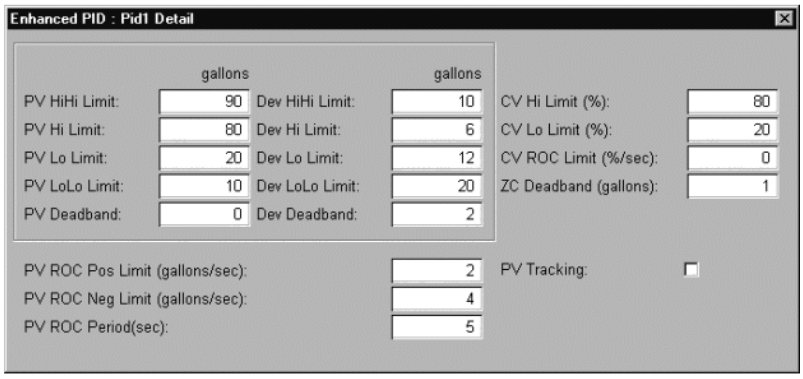
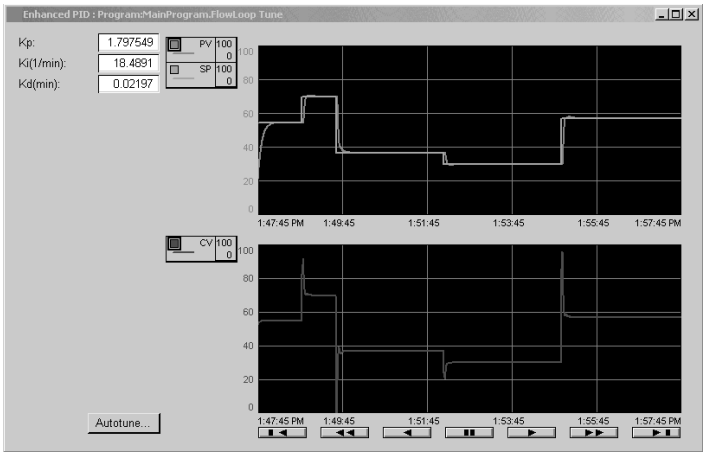
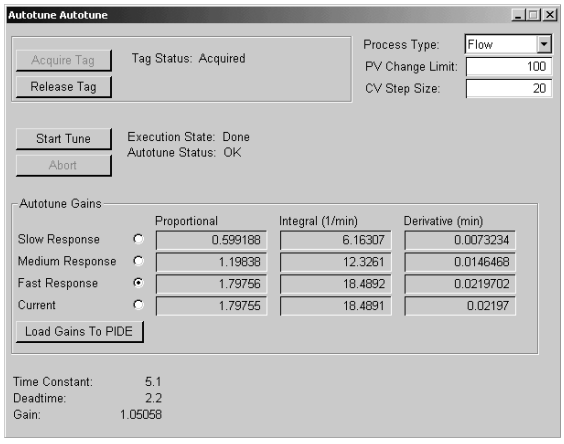


配置属性：	指定：
Commanded State 0 Label （命令状态 0 标签）	命令状态 0 的标签。
Commanded State 1 Label （命令状态 1 标签）	命令状态 1 的标签。
Commanded State 2 Label （命令状态 2 标签）	命令状态 2 的标签。
Actual State 0 Label （实际状态 0 标签）	实际状态 0 的标签。
Actual State 1 Label （实际状态 1 标签）	实际状态 1 的标签。
Actual State 2 Label （实际状态 2 标签）	实际状态 2 的标签。

PIDE 控件

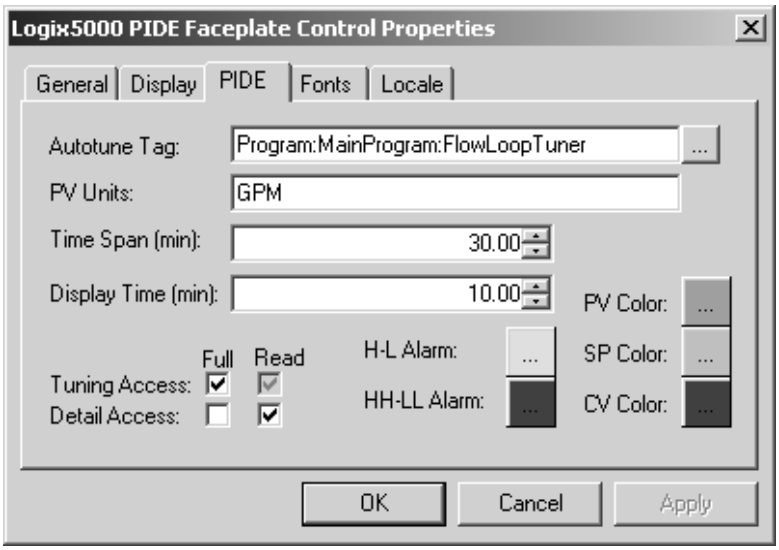


控制面板要素：	显示：
Mode （方式）	功能块的方式。
PV 条形指示计	PV 值。条形指示计的极限值为 PVEUMax 和 PVEUMin。
报警条	偏离量和 PV 极限报警的极限值。 左侧的偏离量报警条随 SP 值变化。 PV 极限报警条位于右侧并且始终保持静止状态。
SP 滑块	SP 的值。 滑块的极限值为 PVEUMax 和 PVEUMin。滑块由其轨道限制在 SPHLimit 和 SPLLimit 之间，可能未完全覆盖 PV 的范围。
ROC Alarm （变化率报警）指示	PVROCPosAlarm 和 PVROCNegAlarm 的状态。
Ratio （速率）	Ratio 值。 仅在 AllowCasRat 和 UseRatio 均置位的情况下才会显示此内容。
SP	SP 值。用户也可以在此编辑框中输入新的 SP。
PV	PV 值。
CV 滑块	CV 值。 滑块的极限值为 0% 到 100%。
CV	CV 值。
Program （程序）按钮	单击此按钮将 OperProgReq 置位。
Operator （操作员）按钮	单击此按钮将 OperOperReq 置位。

控制面板要素：	显示：
Cas/Rat （级联 / 比例）按钮	单击此按钮将 OperCasRatReq 置位。
Auto （自动）按钮	单击此按钮将 OperAutoReq 置位。
Manual （人工）按钮	单击此按钮将 OperManualReq 置位。
Detail （详细信息）按钮	弹出详细信息对话框。 
Tune （调节）按钮	弹出调节对话框。 
Autotune （自动调节）按钮	弹出自动调节对话框 （可从上面所示的调节对话框中进入）。 

控制面板要素：	显示：
Status （状态）	功能块中所有置位的状态位。 如果所有位均未置位，则状态显示 "OK" （正常）。

PIDE 控件还具有以下附加属性页。



配置属性：	指定：
Autotune Tag （自动调节标记）	PIDE 标记
PV Units （PV 单位）	控制面板上用作 PV 和 SP 单位的字符串。
Time Span （时间范围）	为趋势保留值的时间长度。
Display Time （显示时间）	在趋势中显示值的时间长度。
PV、 SP 和 CV 颜色	各个参数趋势曲线的颜色。
Tuning Access （调节访问） Detail Access （详细信息访问）	访问权限级别：full （完全）或 read-only （只读）。
报警颜色	报警颜色表示颜色条中的颜色。 报警颜色 1 表示下限或上限报警。报警颜色 2 表示最低下限或最高上限报警。

说明：

T

U

Z

说明：

ASCII 字符代码

字符	十进制	十六进制
[ctrl-@] NUL	0	$00
[ctrl-A] SOH	1	$01
[ctrl-B] STX	2	$02
[ctrl-C] ETX	3	$03
[ctrl-D] EOT	4	$04
[ctrl-E] ENQ	5	$05
[ctrl-F] ACK	6	$06
[ctrl-G] BEL	7	$07
[ctrl-H] BS	8	$08
[ctrl-I] HT	9	$09
[ctrl-J] LF	10	$l ($0A)
[ctrl-K] VT	11	$0B
[ctrl-L] FF	12	$0C
[ctrl-M] CR	13	$r ($0D)
[ctrl-N] SO	14	$0E
[ctrl-O] SI	15	$0F
[ctrl-P] DLE	16	$10
[ctrl-Q] DC1	17	$11
[ctrl-R] DC2	18	$12
[ctrl-S] DC3	19	$13
[ctrl-T] DC4	20	$14
[ctrl-U] NAK	21	$15
[ctrl-V] SYN	22	$16
[ctrl-W] ETB	23	$17

字符	十进制	十六进制
[ctrl-X] CAN	24	$18
[ctrl-Y] EM	25	$19
[ctrl-Z] SUB	26	$1A
ctrl-[ESC	27	$1B
[ctrl-\] FS	28	$1C
ctrl-] GS	29	$1D
[ctrl-^] RS	30	$1E
[ctrl-_] US	31	$1F
SPACE	32	$20
!	33	$21
“	34	$22
#	35	$23
$	36	$24
%	37	$25
&	38	$26
‘	39	$27
(	40	$28
)	41	$29
*	42	$2A
+	43	$2B
,	44	$2C
-	45	$2D
.	46	$2E
/	47	$2F
0	48	$30
1	49	$31
2	50	$32
3	51	$33
4	52	$34
5	53	$35
6	54	$36
7	55	$37
8	56	$38
9	57	$39
:	58	$3A

字符	十进制	十六进制
;	59	$3B
<	60	$3C
=	61	$3D
>	62	$3E
?	63	$3F
@	64	$40
A	65	$41
B	66	$42
C	67	$43
D	68	$44
E	69	$45
F	70	$46
G	71	$47
H	72	$48
I	73	$49
J	74	$4A
K	75	$4B
L	76	$4C
M	77	$4D
N	78	$4E
O	79	$4F
P	80	$50
Q	81	$51
R	82	$52
S	83	$53
T	84	$54
U	85	$55
V	86	$56
W	87	$57
X	88	$58
Y	89	$59
Z	90	$5A
[	91	$5B
\	92	$5C
]	93	$5D
^	94	$5E
_	95	$5F

字符	十进制	十六进制
‘	96	$60
a	97	$61
b	98	$62
c	99	$63
d	100	$64
e	101	$65
f	102	$66
g	103	$67
h	104	$68
i	105	$69
j	106	$6A
k	107	$6B
l	108	$6C
m	109	$6D
n	110	$6E
o	111	$6F
p	112	$70
q	113	$71
r	114	$72
s	115	$73
t	116	$74
u	117	$75
v	118	$76
w	119	$77
x	120	$78
y	121	$79
z	122	$7A
{	123	$7B
|	124	$7C
}	125	$7D
~	126	$7E
DEL	127	$7F

Rockwell Automation 支持

为了协助您使用我们的产品， Rockwell Automation 在网站上提供了技术信息。您可以在 http://support.rockwellautomation.com 找到技术手册、常见问题知识库、技术和应用说明、软件服务包的样本代码和链接，并且可以自定义 MySupport 功能以充分利用这些工具。

有关安装、配置和故障排除的其他技术电话支持，我们推出了 TechConnect 支持程序。要获得更多信息，可以联系当地分销商或 Rockwell Automation 代表，或者访问 http://support.rockwellautomation.com。

安装协助

如果在安装硬件模块后 24 小时内遇到问题，请查看本手册中的信息。也可以拨打专用客户支持电话获取使模块正常运行的初步帮助：

美国	1.440.646.3223 星期一 — 星期五，东部时间上午 8 点 — 下午 5 点
美国以外的国家或地区	请就任何技术支持问题联系当地的 Rockwell Automation 代表。

因新产品不满意而退货

Rockwell 对所有的产品都进行测试，以确保从制造工厂发出后就能完全正常运行。然而，如果您的产品无法运行并需要退货：

美国	联系您的分销商。为了完成退货过程，您必须向分销商提供客户支持事件号 （使用上面的电话号码获取一个）。
美国以外的国家或地区	请联系当地 Rockwell Automation 代表了解退货规程。

www.rockwellautomation.com

Corporate Headquarters
Rockwell Automation, 777 East Wisconsin Avenue, Suite 1400, Milwaukee, WI, 53202-5302 USA, Tel: (1) 414.212.5200, Fax: (1) 414.212.5201

Headquarters for Allen-Bradley Products, Rockwell Software Products and Global Manufacturing Solutions
Americas: Rockwell Automation, 1201 South Second Street, Milwaukee, WI 53204-2496 USA, Tel: (1) 414.382.2000, Fax: (1) 414.382.4444
Europe: Rockwell Automation SA/NV, Vorstlaan/Boulevard du Souverain 36-BP 3A/B, 1170 Brussels, Belgium, Tel: (32) 2 663 0600, Fax: (32) 2 663 0640
Asia Pacific: Rockwell Automation, 27/F Citicorp Centre, 18 Whitfield Road, Causeway Bay, Hong Kong, Tel: (852) 2887 4788, Fax: (852) 2508 1846

Headquarters for Dodge and Reliance Electric Products
Americas: Rockwell Automation, 6040 Ponders Court, Greenville, SC 29615-4617 USA, Tel: (1) 864.297.4800, Fax: (1) 864.281.2433
Europe: Rockwell Automation, Brühlstraße 22, D-74834 Elztal-Dallau, Germany, Tel: (49) 6261 9410, Fax: (49) 6261 17741
Asia Pacific: Rockwell Automation, 55 Newton Road, #11-01/02 Revenue House, Singapore 307987, Tel: (65) 351 6723, Fax: (65) 355 1733

出版号 1756-RM006D-ZH-P - 5 月 2005 PN 957974-79
替代出版号 1756-RM006C-EN-P - 2003 年 6 月