SIEMENS

SIMATIC APT

Programming Reference (Graphics/Math) Manual

Order Number: PPX:APT–8102–10
Text Assembly Number: 2801047–0007
Tenth Edition

⚠ DANGER

DANGER indicates an imminently hazardous situation that, if not avoided, will result in death or serious injury.

DANGER is limited to the most extreme situations.

⚠ WARNING

WARNING indicates a potentially hazardous situation that, if not avoided, could result in death or serious injury, and/or property damage.

⚠ CAUTION

CAUTION used with a safety alert symbol indicates a potentially hazardous situation that, if not avoided, could result in minor or moderate injury.

CAUTION

CAUTION used without the safety alert symbol indicates a potentially hazardous situation that, if not avoided, could result in property damage.

NOTICE

NOTICE indicates a potential situation that, if not avoided, could result in an undesirable result or state.

Copyright 2001 by Siemens Energy & Automation, Inc.
All Rights Reserved — Printed in USA

Reproduction, transmission, or use of this document or contents is not permitted without express consent of Siemens Energy & Automation, Inc. All rights, including rights created by patent grant or registration of a utility model or design, are reserved.

Since Siemens Energy & Automation, Inc., does not possess full access to data concerning all of the uses and applications of customer's products, we do not assume responsibility either for customer product design or for any infringements of patents or rights of others which may result from our assistance.

MANUAL PUBLICATION HISTORY

SIMATIC APT Programming Reference (Graphics/Math) Manual
Order Manual Number: PPX:APT–8102–10
Refer to this history in all correspondence and/or discussion about this manual.

Event	Date	Description
Original Issue	11/88	Original Issue (2601247-0001)
Second Edition	03/90	Second Edition (2601247-0002)
Third Edition	07/91	Third Edition (2601247-0003)
Fourth Edition	11/92	Fourth Edition (2801047-0001)
Fifth Edition	02/94	Fifth Edition (2801047-0002)
Sixth Edition	10/94	Sixth Edition (2801047-0003)
Seventh Edition	05/95	Seventh Edition (2801047-0004)
Eighth Edition	11/96	Eighth Edition (2801047-0005)
Ninth Edition	12/98	Ninth Edition (2801047-0006)
Tenth Edition	4/01	Tenth Edition (2801047-0007)

LIST OF EFFECTIVE PAGES

Pages	Description	Pages	Description
Cover/Copyright	Tenth		
History/Effective Pages	Tenth		
Trademarks/Copyrights	Tenth		
iii — xxiii	Tenth		
1-1 — 1-10	Tenth		
2-1 — 2-45	Tenth		
3-1 — 3-28	Tenth		
4-1 — 4-28	Tenth		
5-1 — 5-7	Tenth		
6-1 — 6-24	Tenth		
7-1 — 7-16	Tenth		
8-1 — 8-25	Tenth		
9-1 — 9-15	Tenth		
10-1 — 10-32	Tenth		
11-1 — 11-72	Tenth		
12-1 — 12-31	Tenth		
13-1 — 13-20	Tenth		
A-1 — A-48	Tenth		
B-1 — B-16	Tenth		
C-1 — C-19	Tenth		
Index-1 — Index-9	Tenth		
Registration	Tenth		

Trademarks and Copyrights

S5™ is a trademark, and STEP®, SIMATIC®, and SINEC® are registered trademarks, of Siemens AG.

PCS™, 386/ATM™, APT™, Series 500™, Series 505™, TISOFT™, TISTAR™, TIWAY™, Thermocouple™, and UNILINK™ are trademarks of Siemens Energy & Automation, Inc.

Motorola® is a registered trademark of Motorola, Incorporated.

Windows®, Windows®95, Windows NT®, Windows®2000, Microsoft®, and MS-DOS® are registered trademarks of Microsoft Corporation.

UNIX® is a registered trademark of Novell, Inc.

Contents

List of Figures

List of Figures (continued)

List of Figures (continued)

List of Tables

List of Tables (continued)

Preface

New Features of APT

SIMATIC *Application Productivity Tool — APT* is a software package that you can use to design and implement a solution to your process control problem. The capabilities of APT have been enhanced in Software Release 1.9A. The documented differences between APT Release 1.9 and Release 1.9A are indicated by change bars in the manual page margins.

Controller Families

APT continues to support two controller families, the Series 505 and the SIMATIC S5. Most programming tasks, like writing a program, downloading, or debugging, are handled the same way in APT regardless of your controller type. The way APT treats direct memory addressing and I/O is determined by whether you have an S5 or a Series 505 controller.

Using APT Documentation

The APT manual set is organized to make it easy both to use the manuals and to follow the program design process that is appropriate for APT. The APT manual organization is described below.

- *SIMATIC APT User Manual* (Volume 1) is a guide for using the operator interface to enter your program.
- *SIMATIC APT Programming Reference Manual (Tables)* and *APT Programming Reference Manual (Graphics/Math)* (Volumes 2 and 3) provide the information that you need to design your process control solution. These manuals describe the APT programming languages, the characteristics of APT objects, and the tables that you use to configure these objects. Information is presented in the order that provides for the most efficient and logical design of an APT program.
- *SIMATIC APT Applications Manual* (Volume 4) is intended to help you design and write an application program using APT. It includes programming hints, specific examples, and a recommended approach to designing the controls for a factory process.
- *SIMATIC APT MAITT User Manual* (Volume 5) provides the information that you need to design and execute a test program for an application program.
- *SIMATIC APT Release Notes* have important information not included in the manual set.
- The APT manual set is available both in paper form (APT–8200–T) and in electronic form on CD-ROM (APT–8200–CD).

NOTE: Unless otherwise specified, the term “OSx” is used throughout this manual to designate SIMATIC TISTAR Releases 1.x and 2.x in addition to SIMATIC PCS Release 3.x and SIMATIC PCS 7 OSx Release 4.x.

Chapter 1

Continuous Function Charts

1.1 Understanding Continuous Function Charts

Continuous Function Chart Language

The Continuous Function Chart (CFC) Language is a graphics-based language. The CFC Language allows you to define the algorithms that are necessary to provide continuous control of your process.

A Continuous Function Chart consists of one or more boxes, or Continuous Function Blocks (CFBs). Each box represents a pre-defined algorithm that you configure to control objects in APT.

A CFB can represent an analog alarm, a PID loop, a limiter, a selector, etc. To configure a CFB, you define its characteristics by completing the associated form. In addition, you can include Math Language statements that provide additional control within the CFB.

Most CFBs have inputs and/or outputs, which are graphically represented as arrows. These indicate the flow of data to or from the CFB. To indicate that an output of one CFB serves as an input to another CFB, you draw a line to connect the output and input arrows of the respective CFBs.

CFC Scope and Operation

To simplify system design, the structure of APT encourages you to break down large processes into smaller units and to design the operation of each unit independently. For this reason, you need to locate most CFCs at the Unit Content Level of the hierarchy. Because your application may require some global-level continuous control, APT also allows you to place CFCs at the Program Content Level.

One CFB can reference another CFB only if both reside in the same unit. CFBs in the same unit can also communicate with each other through variables and objects that reside at the Program Content Level or at the Unit Content Level for that unit. CFBs in different units must communicate with each other through variables and objects created at the program content level.

Most CFBs can be enabled and disabled to operate only when you need them. You can enable blocks from SFC steps or from other CFBs. Two CFBs, the interlock block and the active math block, run continuously and cannot be disabled.

The three operating states of a CFB are shown in Figure 1-1. You can use the "Not ready" state to interlock the execution of the CFB to some external conditions. When the CFB is Not ready, the CFB is forced to the disabled state. To return the CFB to the enabled state, you must not only set the "Not ready" (**.NRDY**) extension to false, but also re-enable the CFB (set **.ENABL** to true).

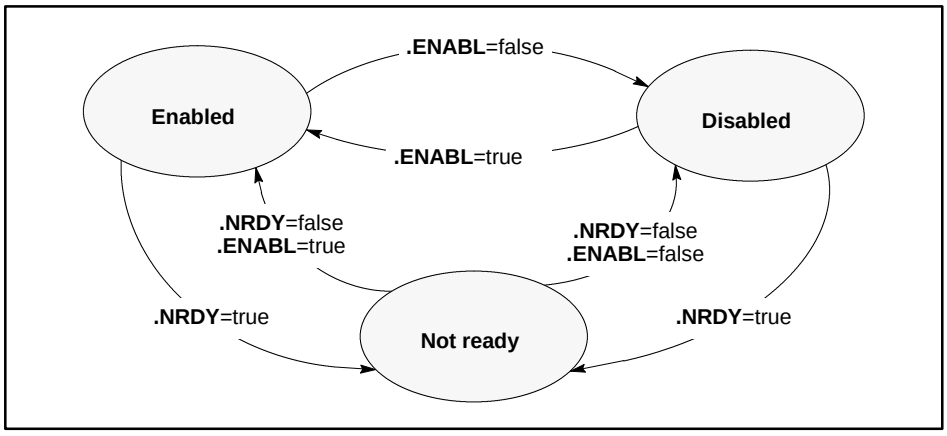


Figure 1-1 CFB Operation

1.2 Continuous Function Blocks

Categories

The 45 continuous function blocks available in APT are grouped into eight categories. Each category represents one basic type of control. The CFB categories follow.

- Standard blocks include basic PID loops, analog alarm, and on/off functions.
- Dynamic blocks include simulator and derivative functions.
- Advanced blocks include special combinations of the standard and dynamic control blocks.
- Limiter blocks include an output limiter and a rate limiter.
- Math blocks include basic arithmetic operations, an interlock, and special math functions.
- Selector blocks include seven types of blocks that allow you to select from multiple inputs.
- Valve blocks include four types of special valve control.
- Other control blocks include miscellaneous functions such as anti-reset windup protection, scaling, and a correlated look-up table.

Availability

Table 1-1 lists all of the CFBs available in APT. All CFBs are supported for Series 505 and S5 controllers. The math block and the interlock block are the only CFBs allowed for the SIMATIC 560/560T CPUs.

Table 1-1 lists all of the CFBs according to type.

Table 1-1 Continuous Function Blocks by Category

Control Type	Continuous Function	
Standard	Analog Alarm On/Off PD Loop PID Loop	I Loop P Loop PI Loop
Dynamic	Dead Time Delay First Order Lag Second Order Lag Integrator	Derivative First Order Lead Lag Second Order Lead Lag
Advanced	Dead Time Compensator Feedforward Output Adjust Feedforward Setpoint Adjust	Dual Mode Ratio Station
Limiter	Output Limiter	Rate Limiter
Selector	Average Selector Inswitch Selector Median Selector Threshold Selector	High Selector Low Selector Outswitch Selector
Valve	Motor Position Control Proportional Time Control	Split Range Valve Sequencer
Math	Absolute Value Interlock [1] Multiplier Square Root Summer	Divider Math [1] Square Subtractor
Other	Correlated Lookup Table Anti-Reset Windup/ (Constraint Type)	Scale Anti-Reset Windup/ (Select Type)

1 These are the only blocks that are supported by the 560/560T controllers.

Configuring CFBs

After you place a CFB on the Continuous Function Chart, you are prompted to enter a name. The name can contain up to eight characters and can be any combination of letters, digits, and underscores. At least one character must be a letter.

The name that you enter in the dialogue window is used to create default variables for the block. For example, the CFB pictured in Figure 1-2 is a split range block named *sr01*. The block has one input and two outputs. The default names for these are *sr01*.**IN**, *sr01*.**OUT1**, and *sr01*.**OUT2**.

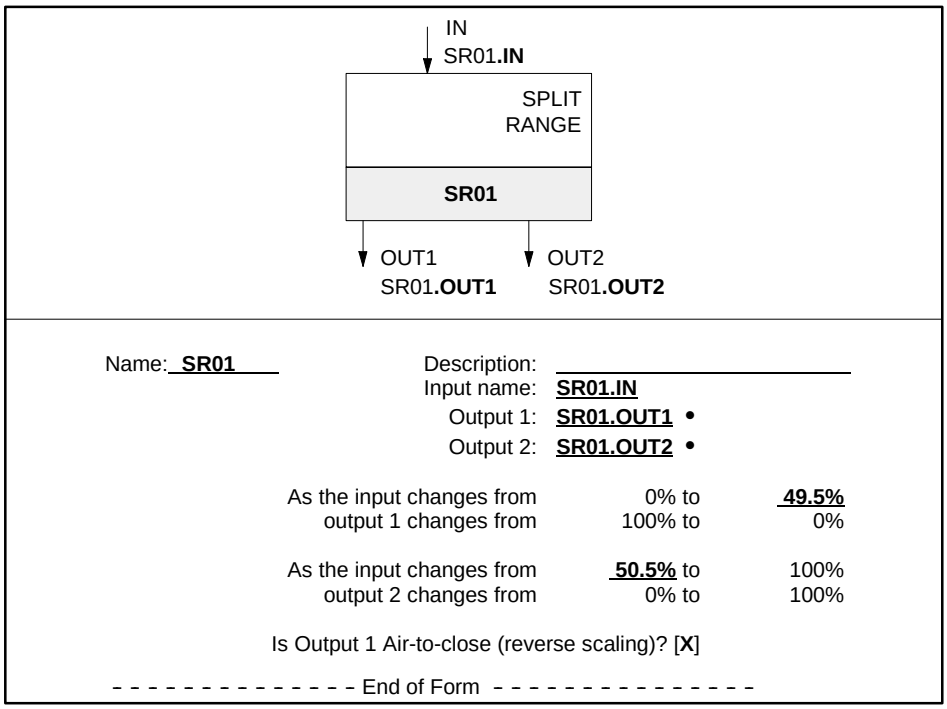


Figure 1-2 A Continuous Function Block (CFB)

When you edit the CFB form, the default names appear in the appropriate fields. The other entry fields depend on the type of block. In some cases, entry fields depend on other selections that you make in the form. For example, if you select “limit output” for a PID loop, entry fields for minimum and maximum values appear on the screen. If you do not select “limit output,” these fields do not appear.

APT allows you to connect the output of one CFB to the input of another CFB. Figure 1-3 shows two CFBs connected by a connection line.

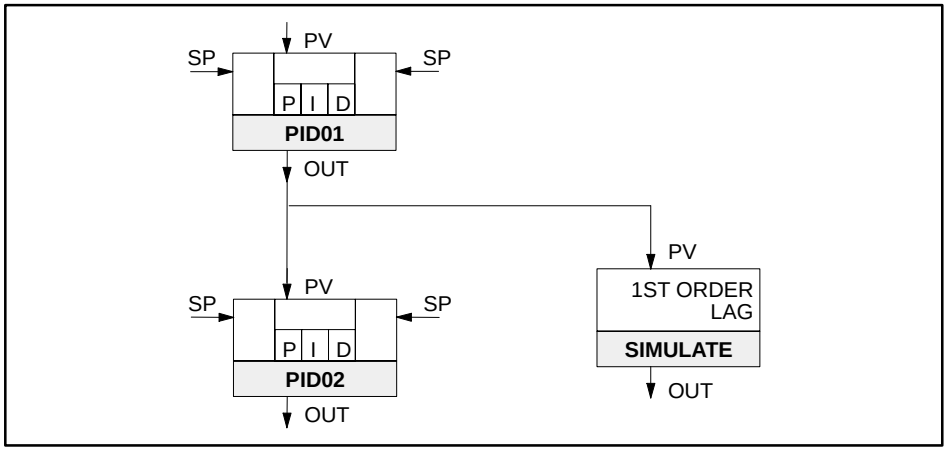


Figure 1-3 Connected CFBs

Graphically, some CFBs (such as the PID blocks in Figure 1-3) appear to have two setpoint (SP) inputs, one on each side of the block; however, these blocks actually have only one setpoint input. The two arrows make it easy for you to connect a line to either side of the block. Because the two input arrows actually represent a single setpoint input, you cannot connect lines to both setpoints.

As long as a connection line begins on an output arrow and terminates on an input arrow, you can draw it anywhere within the boundaries of a CFC. However, do not draw lines through CFB graphics; draw lines through other lines only when unavoidable.

If you draw a connection from the input of one CFB to the output of another CFB, the input changes its name to the same name as the output. As long as a graphic connection exists between the two, the only way to change the name of the input is to edit the output; you cannot edit the input name directly.

The algorithm that controls object download affects the execution flow of CFBs, not the apparent order indicated by the graphical connections. Although the exact order in which blocks are executed is not predictable, the order does not change from download to download.

Commands

CFBs have associated commands to manipulate the blocks from the parallel section of an SFC step, or from a math block that generates RLL (Series 505) or STL (S5) code. Table 1-2 lists the CFB commands with the corresponding blocks. These commands are explained in subsequent chapters, within the description of each block.

Table 1-2 CFB Commands

Commands	Continuous Function Blocks	
ENABLE (ENA) DISABLE (DIS)	Analog Alarm Anti-Reset Windup/Constraint Anti-Reset Windup/Select Average Selector Correlated Lookup Table Control Rate Limiter Scale Second Order Lag Second Order Lead Lag Split Range Threshold Selector Valve Sequencer	Math Median Selector Motor Position Control Output Limiter Proportional Time Dead Time Delay Derivative First Order Lag First Order Lead Lag High Selector Integrator Low Selector
AUTO MANUAL (MAN) CASCADE (CAS)	Dead Time Compensator Feedforward Setpoint Adjust I Loop On/Off P Loop	PD Loop PI Loop PID Loop Ratio Station
AUTO MANUAL (MAN) CASCADE (CAS) ENABLE (ENA) DISABLE (DIS)	Feedforward Output Adjust	Dual Mode
ENABLE (ENA) DISABLE (DIS) SWITCH	Inswitch Selector	Outswitch Selector

The commands that are associated with the CFB control the extension variables that are associated with the CFB, that is, they control the APT flags and boolean mode variables.

Extensions

When you create a CFB, APT automatically creates a set of extension variables that are associated with that CFB. The name of each variable consists of the name of the CFB followed by a period and the extension.

For example, the **.NRDY** (not ready) extension is a read/write boolean variable that you can set to true to disable the block automatically.

Some extensions are classified as "read-only": you can read the value of the extension variable, but you cannot change its value. Other extensions are classified as "read/write": you can both read and change the value of that extension variable.

You can control the mode of the CFB in the following ways:

Method	Example
Use a command in an SFC	**Auto** *pid_1*;
Use a command in Math	**Auto** *pid_1*;
Use an APT flag	**On** *pid_1*.**RAUTO;**
Use a boolean extension	*pid_1*.**GAUTO := true;**

The command manipulates the APT flag and the boolean extensions in order to control the CFB. Use the command, rather than the extensions, to control the mode of the block, because APT can overwrite the boolean mode extensions. Use the APT flags and boolean extensions to monitor the status of the block. If you use an APT flag extension, you must use the LATCH, ON, and CLEAR math procedures, described in Chapter 11, "Math Functions and Procedures."

NOTE: S5 users are familiar with the term "flag" used to refer to a single digital point. However, an APT flag is more than this; it is a construct of several points, with logic that you can manipulate by using the LATCH, ON, and CLEAR commands.

The CFB extensions are listed with the description of each block in Chapters 2 through 9 of this manual. The "OSx Tag Translation" appendix in the *SIMATIC APT Applications Manual* contains information about how these extensions are translated to OSx.

Error-Code Extensions

All CFBs have extensions that provide information about any errors that can occur. When an error occurs in a Series 505 controller, it is automatically stored in these error extensions. Although the extensions are also present for CFBs programmed into an S5 controller, they contain no meaningful information.

- If an execution error occurs, the **.ECODE** extension contains a non-zero value.
- The **.IID** extension identifies the SFPGM associated with the block.
- The **.SNUM** extension contains the SFPGM line number that contains the error.

Table 1-3 lists the error codes that are provided by a Series 505 controller when it executes SFPGM code, and the associated number that appears in the **.ECODE** extension for Series 505 controllers.

Table 1-3 CFB Error Codes (Series 505 only)

Code		Meaning
Hex	Decimal	
02	02	Address out of range
03	03	Requested data not found
09	09	Incorrect amount of data sent with request
11	17	Invalid data
43	67	Control block does not exist
4A	74	Attempt to access an integer variable as a real
4B	75	Attempt to access a real variable as an integer
4E	78	Attempt to write to a read-only variable
4F	79	Invalid variable data type for this operation
52	82	Invalid returned value
53	83	Attempt to use lead-lag procedure in event or continuous math block
58	88	Stack overflow while evaluating IF...THEN statement
5A	90	Arithmetic overflow
5B	91	Invalid operator in an IF...THEN statement
5D	93	Attempt to divide by zero
60	96	Invalid data type code (usually in IF...THEN statement)

Chapter 2

Standard Control Blocks

2.1 Understanding Standard Control Blocks

Overview

APT provides three basic types of standard control blocks:

- PID blocks provide standard combinations of proportional, integral, and derivative control.
- On/off blocks incorporate a high-gain PID loop and convert the output to a boolean value.
- Analog alarm blocks provide a range check on an analog input.

Availability

All of the standard control blocks are supported for Series 505 and S5 controllers. The standard control blocks are not supported for the 560/560T controllers.

Loop Control

Figure 2-1 shows the basic operation of a continuous process loop.

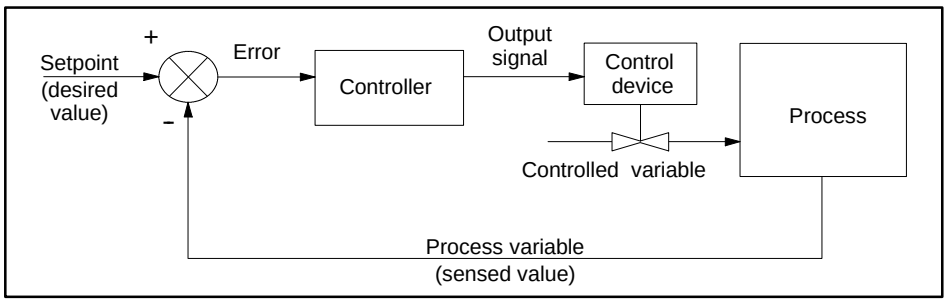


Figure 2-1 Process Control Loop

In a typical loop operation, a measured process variable is compared to a setpoint, or desired value. The difference between these two values is the error signal that the controller uses to calculate an output to the control device. The control device, in turn, manipulates the controlled variable that serves as an input to the process.

2.2 PID Blocks

Overview

The PID loops provide a combination of proportional, integral, and derivative control to calculate the value of an output. The output is based on the error (the relationship between the setpoint and the process variable).

- In proportional control, the output is proportional to the error and is based on the formula:

$$M = K_c E + M_x$$

where

$$K_c = gain = \frac{\Delta output}{\Delta input}$$

$M = output$, $E = error = setpoint - process\ variable$

- In integral control, the output is based on the integral of the error with respect to time. Integral mode can be combined with proportional control or with proportional and derivative control. Integral control uses the formula:

$$M = \frac{T_s}{T_i} \sum_{J=0}^{n} E_j$$

where $T_s = sample\ time$, $T_i = reset\ time$, $E = error\ at\ time\ J$

- In derivative control, the output is based on the rate of change of the process variable. Derivative control is never used alone, but always in combination with proportional and/or integral control:

$$M = - \frac{T_d}{T_s} \frac{dPV}{dt}$$

where $T_s = sample\ time$, $T_d = rate\ time$, $PV = process\ variable$

Availability

PID loops are available for both Series 505 and S5 controllers; however, for the Series 505, the PID loop is part of the firmware, while for the S5 the loop has been implemented through code. Consequently, the S5 PID loop has several distinctive features:

- The Series 505 S-memory mapping feature is not supported for S5.
- The process variable for S5 controllers must be a real number, for instance, a scaled analog input; the PID output must be a real number, for instance, an analog output.
- The PLC Reserved Address feature is not supported for S5.
- The broken transmitter signal is an AI extension and not a loop extension for S5.

PID Blocks (continued)

For all standard control blocks, you must specify a sample time that determines how often the loop is updated. This sample time is a real number between 0.1 and 1.67772×10^6 seconds for Series 505 controllers, and for S5 controllers it is a real number between 0.1 and 3276.7 seconds.

APT provides the following two algorithms that are available with PID blocks.

Standard PID Algorithm (Position Algorithm)

In a position algorithm, the position of the device being controlled is computed based on the error. The algorithm computes the value of the output for each loop calculation. For example, if the position algorithm calculates that the valve needs to be 50% open, the output is 50%.

The position algorithm is the most commonly used and is based on the formula:

$$M_n = K_c \left[E_n + \frac{T_s}{T_i} \sum_{J=0}^{n} E_j - \frac{T_d}{T_s}(PV_n - PV_{n-1}) \right] + M_x$$

where M_n = output at time n, K_c = gain, E_n = error, E_j = error at time j
T_s = sample time, T_i = reset time, T_d = rate time
M_x = bias, PV = process variable

Velocity Algorithm

A velocity algorithm computes the change in the device position based on the error. The algorithm generates a change in output based on the current position. For example, if you want the valve 50% open and the current position is 40%, the output is 10%.

The velocity algorithm uses the formula:

$$\Delta M_n = M_n - M_{n-1} = K_c \left[E_n - E_{n-1} + \frac{T_s}{T_i} E_n - \frac{T_d}{T_s}(PV_n - 2PV_{n-1} + PV_{n-2}) \right]$$

Use the velocity algorithm only when you have a good understanding of the digital control problem. You can only use the velocity algorithm with actuators that move by percentage. For example, an output of zero means that you hold to your current position.

APT provides blocks for the following combinations: PI, P, PID, I, and PD. In the following discussion of the PID block, PID refers to any of these combinations.

PID Loop CFB and Form

Figure 2-2 shows the PID Loop CFB graphic.

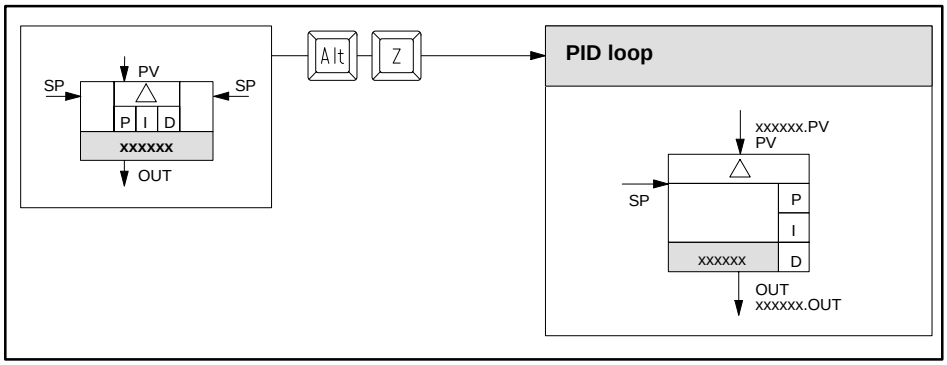


Figure 2-2 PID Loop CFB

Figure 2-3 shows the PID form with default values and default names that are based on the name that you enter when you create the CFB graphic. The shaded entries in the form depend on a previous selection and are not a part of the initial display.

The details of each entry are explained on page 2-9 for process variable information, page 2-11 for process setpoint information, page 2-13 for output information, page 2-15 for controller options, page 2-17 for associated math, and page 2-19 for tuning parameters.

The various fields in the PID form are described below and on subsequent pages.

Sample time: The sample time in seconds determines how often the loop, the deviation alarm bits, and the associated math are evaluated and updated; default is 1; the value is measured in tenth-of-a-second increments, between 0.1 and 1.67772×10^6 for Series 505 controllers, or between 0.1 and 3276.7 for S5 controllers.

Algorithm type: POSITION or VELOCITY; default is POSITION.

PLC address (automatic or user assigned): (Series 505 only) Default is automatic. Automatic allows APT to assign a loop address. User assigned allows you to specify the loop address, e.g., %loop1.

Reserved address: (Series 505 only) This is the starting controller address for the specified loop, e.g., %loop1. You do not have to reserve Series 505 S-memory, but you must reserve a loop in the Compiler Control file.

Process variable: For Series 505, the name of the analog input, integer, scaled integer, or real input that represents the process variable. For S5, the name of the scaled analog input or real input that represents the process variable. Integer inputs are not allowed for S5.

PID Blocks (continued)

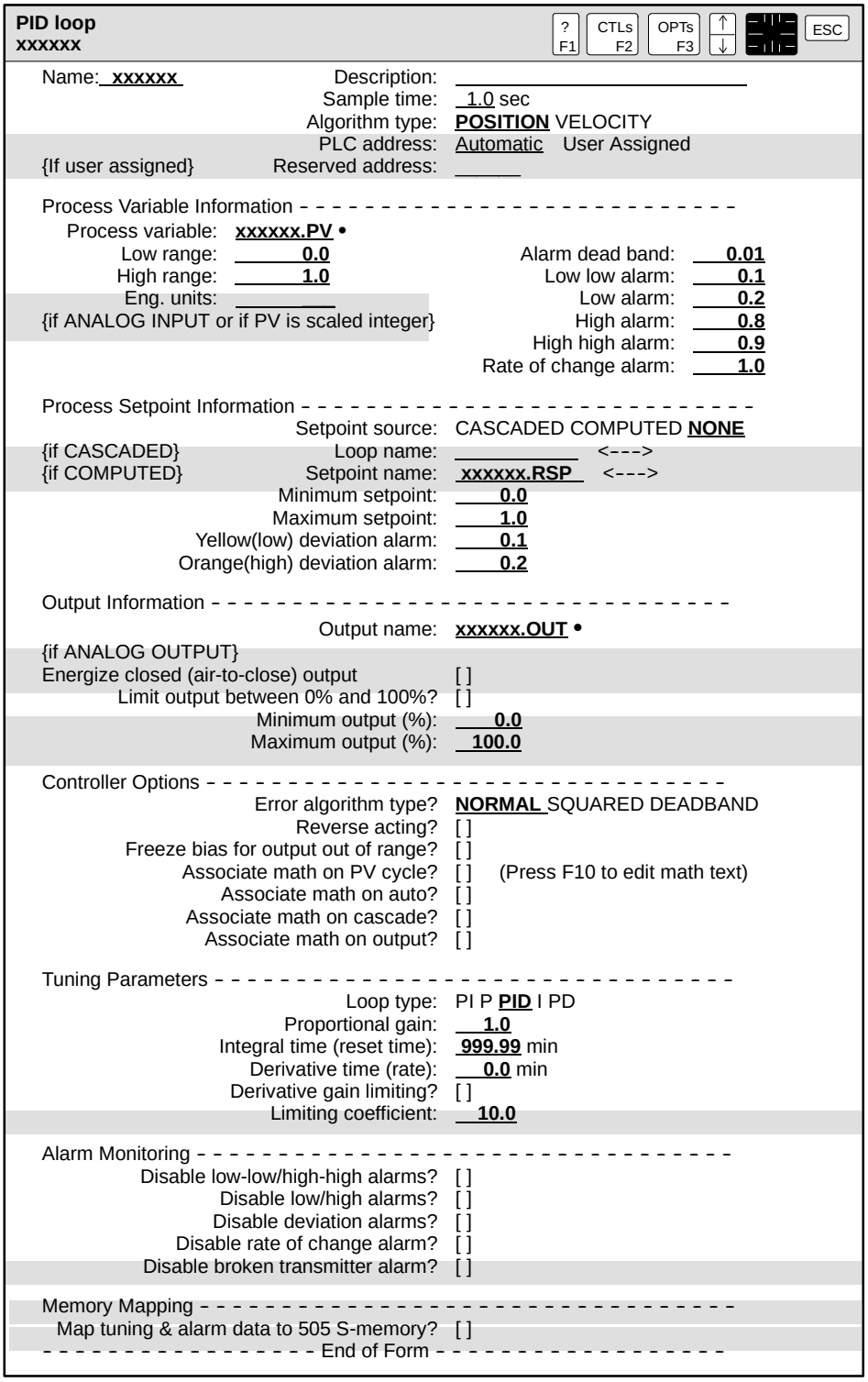


Figure 2-3 PID Form

Low range: Real number in engineering units that is the low range of the PV. Used for alarming; for Series 505, also used to scale an integer PV. If PV is an analog input, this value is taken from the values that you entered in the I/O Table. For Series 505, if PV is a scaled integer, this value is taken from the value entered in the Declaration Table.

High range: Real number in engineering units that is the high range of the PV. Used for alarming; for Series 505, also used to scale an integer PV. If PV is an analog input, this value is taken from the values that you entered in the I/O Table. For Series 505, if PV is a scaled integer, this value is taken from the value entered in the Declaration Table.

Eng. units: Optional documentation field to describe units of final, scaled value (eight characters maximum); appears only if AI or SI is selected as input.

Alarm deadband: Real number in engineering units; prevents the alarm from changing when the process variable is within the deadband.

Low low alarm: Real number in engineering units; must be less than or equal to low alarm value and greater than or equal to low range of PV.

Low alarm: Real number in engineering units; must be less than or equal to high alarm value of PV.

High alarm: Real number in engineering units; must be less than or equal to high high alarm value of PV.

High high alarm: Real number in engineering units; must be greater than or equal to high alarm value and less than or equal to high range of PV.

Rate of change alarm: Real number in engineering units/minute that is used to set rate-of-change alarm.

Setpoint source: CASCADED, COMPUTED, or NONE; default is NONE.

Loop name: Name of PID loop that holds setpoint (SP) input if setpoint source is cascaded.

Setpoint name: Name of real variable that indicates setpoint (SP) input if setpoint source is computed.

Minimum setpoint: Real number that indicates minimum value of setpoint; must be greater than or equal to the low range of the PV.

Maximum setpoint: Real number that indicates maximum value of setpoint; must be less than or equal to the high range of the PV.

Yellow deviation alarm: Value that indicates the maximum allowable error (SP – PV) for the yellow deviation zone is reached.

Orange deviation alarm: Value that indicates the maximum allowable error (SP – PV) for the orange deviation zone is reached.

Output name: Name of the output variable.

PID Blocks (continued)

Energize closed output: X indicates the analog output is air-to-close; a blank field indicates the analog output is air-to-open. If energized closed is selected, the output signal is inverted so that .LMN goes from one to zero as .OUT goes from zero to one; it does not invert the sign of the error term as does reverse acting.

Limit output between 0% and 100%: X indicates that you want to restrict output range to something less than 0.0 to 100.0 for position algorithm or -100.0 to 100.0 for velocity algorithm.

Minimum output: Real number that sets low limit; value must be between 0.0 to 100.0 for position algorithm or -100.0 to 100.0 for velocity algorithm.

Maximum output: Real number that sets high limit; value must be between 0.0 to 100.0 for position algorithm or -100.0 to 100.0 for velocity algorithm.

Error algorithm: NORMAL, SQUARED, or DEADBAND; default is NORMAL.

Reverse acting: X indicates that you want to use the reverse-acting control algorithm; a blank field indicates direct-acting.

Freeze bias for output out of range: X indicates that you want freeze-bias anti-reset windup algorithm; blank indicates that you want bias backtracking.

Associate math: X indicates that you want to include additional math on PV cycle, auto, cascade, or on output.

Loop type: P, PI, PD, or PID; default is PID. The type of loop you choose determines which tuning constants are available to you; the others are forced to non-operational values. See Figure 2-8.

Proportional gain: Integer between 0 and 100 that indicates proportional gain in % / %; default is 1.

Integral time (reset time): Real number between 0.0 and infinity that indicates reset time of integral mode in minutes; default is 999.99.

Derivative time (rate): Real number between 0 and infinity that indicates the derivative time (rate) in minutes; default is 0.0.

Derivative gain limiting: X indicates that you want to limit the derivative gain. You must then enter the limiting coefficient.

Disable alarm monitoring: X indicates that you do not want to monitor the indicated alarm.

Map tuning and alarm data to S-memory: (Series 505 only)
X indicates that APT does not map PID alarm and tuning parameters from Series 505 V-memory to S-memory, but rather uses the existing Series 505 S-memory alarm and tuning parameters.

Process Variable Information

The process variable to the PID block is automatically designated *cfb_name*.**PV**. If you have a Series 505 controller, you can change this to the name of any analog, integer, scaled integer, or real input that you define in the I/O Name or Declaration Tables, described in the *SIMATIC APT Programming Reference (Tables) Manual*. If you have an S5 controller, you can change the name of the PV to any scaled analog input or real input that you have defined. S5 controllers must have real PVs; integer PVs are not processed. If you want to use integers for PVs, see the *SIMATIC APT Applications Manual*, Chapter 2, for programming hints about incorporating integer inputs for an S5 PID. If the block is graphically connected to another block, the process variable name is inserted automatically.

PID Blocks (continued)

High and low limits for the process variable are used to set the high and low range for a real number input. For Series 505 controllers, they can also be used to scale an integer variable input.

- If the input is a real (S5 or Series 505) or integer (Series 505 only) variable, you specify the high and low values in the PID form.
- If the input is an analog input or scaled integer, the high and low ranges are automatically those that you specify when you define the analog input in the I/O Symbolic Name Table, or when you define the scaled integer in the Declaration Table.

Alarm definitions allow you to monitor the process variable for deviation outside a specified range. The standard control blocks provide low-low, low, high, and high-high alarms; see Figure 2-4. If the process variable is outside the limits of the alarms, you may use the alarms to generate warnings and shutdown procedures for the process equipment itself.

The **alarm deadband** prevents the alarm from changing when the process variable is within the specified deadband of each alarm zone.

A **rate-of-change alarm** allows you to monitor how fast the process variable is changing.

These alarms are monitored at the lesser frequency: every 2 seconds or the sample time. To change the alarm values, you write to the corresponding extension variables listed in Table 2-2 on page 2-23.

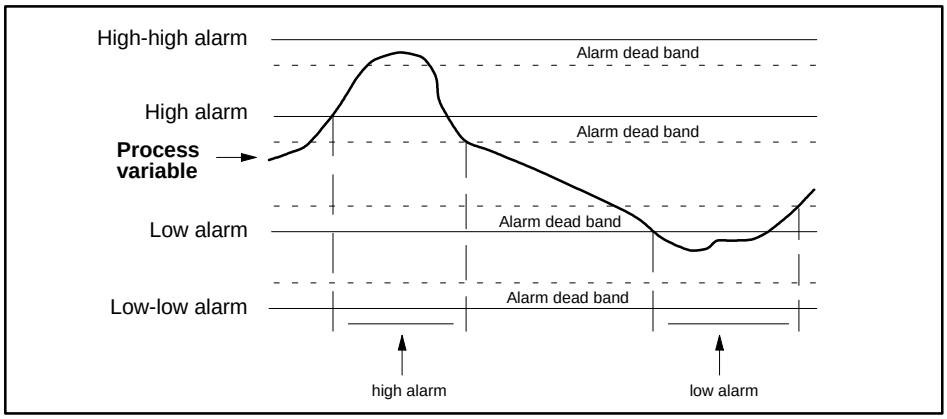


Figure 2-4 Process Variable Alarms

Process Setpoint Information

Three options are available for designating the setpoint, or target value, for a PID block:

- Cascaded setpoint: the setpoint is the output of another PID block.
- Computed setpoint: the setpoint is the name of any real variable that you define in the I/O Name or Declaration Tables or the output from any CFB other than a PID.
- None: the setpoint is assigned from the program by writing to the *cfb_name*.**SP** variable.

PID Blocks (continued)

Minimum and **maximum setpoint** values define the range of the setpoint value. These values must be within the range of the process variable.

Deviation alarms (yellow/low and orange/high) refer to the specified tolerance around the setpoint; that is, they check to see whether the process variable is within a desired range of the setpoint. The yellow and orange deviation zones move up and down with the setpoint as it changes (see Figure 2-5). The yellow/low deviation is closer to the setpoint, and the orange/high deviation is farther away from the setpoint. To change the deviation alarm values, write to the corresponding extension variables, listed in Table 2-2.

The alarm deadband that you specified for the process variable alarms operates around the deviation alarms also.

The deviation alarms can be used as indicators to show how well the loop is controlling, or they can be used as limits to activate some procedure.

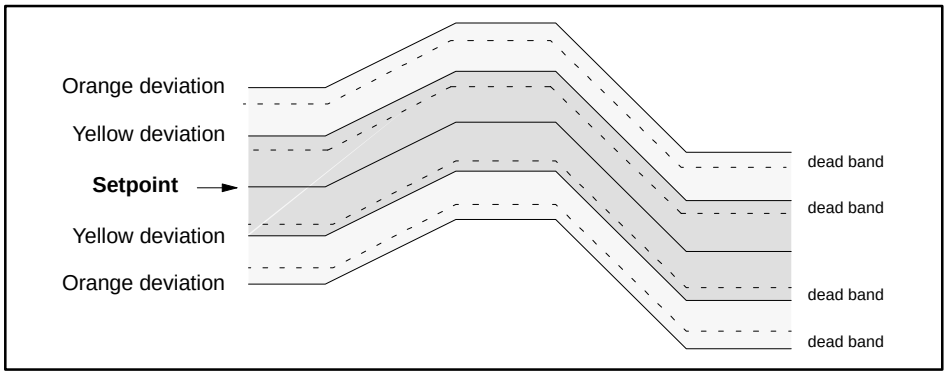


Figure 2-5 Setpoint Deviation Alarms

Output Information

The output from the PID block is automatically designated *cfb_name*.**OUT**. If you have a Series 505 controller, you can change this to the name of any real, integer, analog output variable, or *cfb_name*.**IOUT**. If you have an S5 controller, you can change this to the name of any real variable or analog output; the S5 PID does not process integers. If the output is an analog output (AO) variable, the PID automatically provides the appropriate scaling that you defined in the I/O Name Table.

The output from a PID is limited to a specific range.

- The range for a position algorithm is 0.0 to 1.0. For a position algorithm, the values of the **.HLIM** and **.LLIM** extensions should be between 0.0 and 1.0, where 0.0 = 0% and 1.0 = 100%.
- The range for a velocity algorithm is –1.0 to +1.0. For a velocity algorithm, the values of the **.HLIM** and **.LLIM** extensions should be between –1.0 and 1.0, where –1.0 = –100% and 1.0 = 100%.

If your analog output is air-to-close, you can specify that the output be energize-closed by placing X in the Energize Closed Output field. The output signal is inverted such that zero to one becomes one to zero. This option does not invert the sign of the error term as does a reverse acting loop.

You can specify other values as the minimum and maximum outputs by placing X in the Limit Output field. The values that you specify must be within the normal range for the corresponding algorithm. These limits only apply when the loop is in the AUTO or CASCADE modes.

Figure 2-6 shows the output parameters for a PID loop.

PID Blocks (continued)

Energize Closed (air-to-close) Output: X indicates that the output signal is inverted so that .LMN goes from one to zero as .OUT goes from zero to one; option appears when the output is an analog output variable. APT calculates the .LMN output as: **.LMN = (1 - .OUT**)

This calculation is done for all three loop modes: Manual, Auto, and Cascade.

NOTE: With Series 505 controllers, the air-to-close calculation cannot be made when you indicate X to select **Map tuning and alarm data to S-memory**. If you want to perform the air-to-close calculation, you need to have the Series 505 V-memory mapping to S-memory. S5 controllers do not need the remapping to perform the air-to-close calculation.

Limit Output: X indicates that you wish to change the minimum and maximum output values from their defaults. The values must be between 0 and 100 for the position algorithm and –100.0 to 100.0 for the velocity algorithm. Marking this field automatically gives you the Freeze Bias option, even though it is not marked.

```
Output Information - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                                    Output name:  xxxxxx.OUT
{if ANALOG OUTPUT}
                  Energize Closed (air-to-close) Output  [ ]
                  Limit output between 0% and 100%?  [X]
                                    Minimum output (%):    0.0
                                    Maximum output (%):    100.0
```

Figure 2-6 Output Parameters for PID Loop

You cannot edit the Limit Output field if you have already marked the PID block for OSx tag translation.

Controller Options

When you define the PID block, you have the following controller options: an error algorithm, direct or reverse control, freeze bias for an out-of-range output, and associated math.

Error Algorithm

The error algorithm can be normal, squared, or deadband:

- Normal error algorithm provides control based on the value of the error.
- Squared error algorithm provides control based on the value of the square of the error. This produces a control equation that is less responsive than normal error control.
- Deadband error algorithm provides control based on an error deadband that eliminates gain for small errors within the deadband. The deadband is defined by the yellow deviation alarm limit.

Direct/Reverse Acting Control

The control calculation in the PID CFB can be either direct- or reverse-acting:

- Direct-acting control means that when you increase the setpoint, the output increases, and when you decrease the setpoint, the output decreases. The control action is relative to the change in loop error. If the error increases, then the controller increases the output.

  Example: An example of direct-acting control is heating a vessel with a steam jacket. When heating, the setpoint is higher than the PV, resulting in a positive error. To achieve optimum control, the PID first assigns a large output value to the valve that controls the flow of steam, and then reduces the output value as the temperature in the vessel increases. The output to the steam valve continues to decrease as the temperature increases until a steady-state output is reached that maintains the temperature at the setpoint.

- Reverse-acting control decreases the output if the setpoint is increased, and increases the output if the setpoint is decreased. The direction of change in the setpoint and output is opposite. If the error increases (goes positive), then the controller decreases the output.

  Example: An example of reverse-acting control is cooling a vessel with coolant in the jacket. When cooling, the setpoint is lower than the PV, resulting in a negative error. The PID achieves optimum control by first assigning a large output value to the valve that controls the flow of coolant, and then reduces the output as the temperature in the vessel decreases. The output to the coolant valve continues to decrease as the temperature decreases until a steady-state output is reached that maintains the temperature at the setpoint.

PID Blocks (continued)

There is one difference between reverse-acting calculations and direct-acting calculations. In reverse-acting calculations, the tuning parameters have been negated to correct the control algorithm to compensate for the reverse action; this is not necessary in direct-acting calculations.

Freeze Bias

Reset windup occurs in a PID loop whenever the output of the loop is no longer able to change the process variable by adjusting the output. In this situation, the integral mode continues to increase or decrease the output: this action has no effect on the controlled variable.

The controller has built-in reset windup protection for the PID loop that monitors the output and detects an attempt to move the output beyond 0.0 or 100.0 percent.

Two types of reset windup protection are available, depending on whether you use the **Freeze bias for output** option or not.

- If you select the **Freeze bias** option, the anti-reset windup algorithm stops the integral mode from changing the bias whenever the output exceeds the high or low limit. This freeze bias is also available automatically if you select **Limit output between 0 and 100%**.
- If you do not select the **Freeze bias** option, the algorithm uses bias backtracking and computes what the bias needs to be in order to make the output equal to the high or low output limit.

If you need to limit the output further, you can provide additional anti-reset windup protection by writing directly to the **.AWS** extension (explained on page 2-28).

Associated Math

To expand the capability of the PID blocks, you can append Math computations to the operation. This is referred to as associated math in the form. For Series 505, this math compiles as SFPGM; for S5 controllers, associated math compiles as STL.

- **Associate math on PV cycle?** This math code executes in manual, automatic, and cascade modes before the PID calculation is performed. Preprocessing of the process variable, such as special filtering, occurs in this section.

  To modify the loop process variable from within an associated math block, manipulate the **.LPV** (loop process variable) extension, not the **.IN** (input) extension. APT loads the **.IN** value into the **.LPV** extension before executing the associated math and uses the **.LPV** extension in the computations.

  This section executes either every two seconds or at the sample time, whichever is quicker.

- **Associate math on auto?** This math code executes only when the controller is in automatic mode. Special computations on the setpoint occurs in this section.

  To modify the setpoint in the loop, manipulate the **.LSP** (internal loop setpoint) extension, not the **.SP** (setpoint) extension. APT loads the **.SP** value into the **.LSP** extension before executing the associated math and uses the **.LSP** extension in the computations.

- **Associate math on cascade?** Math code in this section executes only when the controller is in cascade mode. Special computations on the remote setpoint occurs in this section.

  To modify the setpoint in the loop, manipulate the **.LSP** extension, not the **.RSP** (remote setpoint) extension. APT loads the **.RSP** value into the **.LSP** extension before executing the associated math and uses the **.LSP** extension in the computations.

PID Blocks (continued)

For Series 505 controllers, the **.IPV** extension variable contains the scaled integer representation of the internal process variable. The **.ISP** variable contains the scaled integer representation of the internal setpoint. These variables can be used in associated math to convert the corresponding value to a fractional value (**.ISP – .IPV** / 100.0) without concern for engineering units. The **.IPV** and **.ISP** extension variables are not available for S5 controllers.

- **Associate math on output?** Math code in this section executes in AUTO or CASCADE modes after the loop calculation, but before the field value of the loop output is updated.

  If you want to modify the output after the loop computation, but before the field update, manipulate the **.LMN** (internal loop output) extension, not the **.OUT** extension. APT performs the loop calculation and then loads the **.LMN** extension into the **.OUT** extension. For air-to-close on analog output, **.LMN** goes from zero to one for the associated math on output and one to zero after the associated math on output is complete.

Use the internal loop variables, **.LPV**, **.LSP**, and **.LMN**, only within the associated math for the loop. In all other areas of APT, use the **.PV** (for S5 or Series 505) or **.IPV** (Series 505 only) extension to access the process variable for the loop and the **.OUT** (for S5 or Series 505) or **.IOUT** (Series 505 only) extension to access the output; to access the setpoint, use the **.SP** extension in automatic mode and the **.RSP** extension in cascade mode.

The full functionality of the Math language is available in associated math, except for the following restrictions.

- Associated math does not support INIT, BEGIN, and BODY statements. Enter your Math language statements without using INIT, BEGIN, or BODY.
- Associated math does not support declaration of local variables.
- Associated math does not support direct addressing for S5, except in an inline assembly math statement. (See Appendix C for a discussion of the inline assembly math statement.) Some direct addressing is permitted for Series 505.
- %T1 through %T9 are reserved for special purposes, but Series 505 controllers can use %T10 through %T15 as local variables.
- Associated math is SFPGM-only for Series 505; you cannot use any RLL-only functions in associated math if you have a Series 505 controller.

Tuning Parameters

Your selection of a loop type, such as P or PI, determines which values are forced into the loop tuning constants.

When the loop type is not PID, APT continuously supplies an appropriate value in the controller. When the loop type has no proportional (P) term, the extension **.KC** is zero. When the loop type has no integral (I) term, the extension **.TI** is 1.0×10^{18}. When the loop type has no derivative (D) term, the extension **.TD** is zero.

Figure 2-7 shows the tuning parameters for a PID Loop. Figure 2-8 shows tuning parameters for the other combinations of proportional, integral, and derivative control.

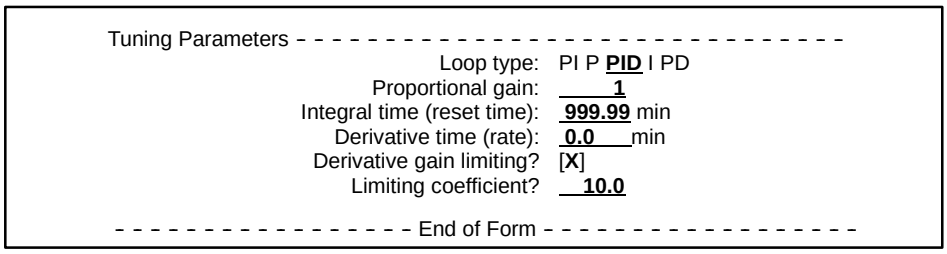


Figure 2-7 Tuning Parameters for PID Loop

Proportional gain: Integer between zero and infinity that indicates proportional gain in % / %; default is 1.

Integral time (reset time): Real number between 0.0 and infinity that indicates reset time of integral mode in minutes; default is 999.99.

Derivative time (rate): Real number between 0.0 and infinity that indicates the derivative time rate in minutes; default is 0.0.

Derivative gain limiting: X limits the derivative gain. See discussion below.

Limiting coefficient: real number between 5.0 and 30.0 that you enter if you select derivative gain limiting; default is 10.

In the standard PID algorithm, the algorithm responds excessively to process noise if the coefficient of the derivative term (Td/Ts) is significantly above the 10 to 20 range. This causes disturbances that lead to erratic behavior of the process.

To solve this problem, the controller allows you the option of selecting a derivative gain limiting coefficient (Kd). The use of this coefficient enables the process variable to be filtered with a time constant that is proportional to the derivative time (Td). The PID equations with the derivative gain limiting coefficient are shown on the following page (top box):

PID Blocks (continued)

Position Algorithm

$$Y_n = Y_{n-1} + \frac{Ts}{Ts + (Td/Kd)} \times \left(PV_n - Y_{n-1}\right)$$

$$\overline{Mx} = Ki \times e_n + Mx_{n-1}$$

$$\overline{M} = Kc \times e_n - Kr\ (Y_n - Y_{n-1}) + \overline{Mx}$$

Velocity Algorithm

$$Y_n = Y_{n-1} + \frac{Ts}{Ts + (Td/Kd)} \times \left(PV_n - Y_{n-1}\right)$$

$$\Delta M_n = Kc \times (e_n - e_{n-1}) + Ki \times e_n - Kr \times (Y_n - 2 \times Y_{n-1} + Y_{n-2})$$

Variable	Definition	Variable	Definition
M_n	Loop output	e_n	Error (SP - PV)
Mx_n	Bias (Mx is the initial valve position)	Td	Rate time
Kc	Proportional gain	Ti	Reset time
Kd	Derivative gain-limiting coefficient	Ts	Sample time
Ki	Integral gain, Kc (Ts/Ti)	Y_n	Filtered PV
Kr	Rate gain, Kc (Td/Ts)	PV_n	Process variable
$\overline{M}$	Calculated output	$\overline{Mx}$	Calculated bias

Tuning Parameters for I Loop

```
Tuning Parameters - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                                  Loop type:  PI P PID I PD
                  Integral time (reset time):  999.99 min

- - - - - - - - - - - - - - - - - End of Form - - - - - - - - - - - - - - - - -
```

Tuning Parameters for P Loop

```
Tuning Parameters - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                                  Loop type:  PI P PID I PD
                        Proportional gain:  1

- - - - - - - - - - - - - - - - - End of Form - - - - - - - - - - - - - - - - -
```

Tuning Parameters for PD Loop

```
Tuning Parameters - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                                  Loop type:  PI P PID I PD
                        Proportional gain:  1
                  Derivative time (rate):  0.0 min
                  Derivative gain limiting?  [ ]

- - - - - - - - - - - - - - - - - End of Form - - - - - - - - - - - - - - - - -
```

Tuning Parameters for PI Loop

```
Tuning Parameters - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                                  Loop type:  PI P PID I PD
                        Proportional gain:  1
                  Integral time (reset time):  999.99 min

- - - - - - - - - - - - - - - - - End of Form - - - - - - - - - - - - - - - - -
```

Figure 2-8 Tuning Parameters for I, P, PD, and PI Loops

PID Commands and Flags

Table 2-1 lists the commands, APT flags, and boolean extensions that you can use in the parallel section of an SFC step or a math section in SFC or a CFB to monitor and control a PID block.

Table 2-1 Controlling a PID Block

Command	APT Flag	Boolean Extension	Description
AUTO	.RATO	.GAUTO	Places loop in automatic mode.
MANUAL (MAN)	.RMAN	.GMAN	Places loop in manual mode; deactivates PID calculation.
CASCADE (CAS)	.RCAS	.GCASC	Places loop in cascade mode.

The commands manipulate the APT flags and the boolean extensions in order to control the loop. Use the commands, rather than the extensions, to control the mode of the loop, because APT can overwrite the boolean mode extensions. Use the APT flags and boolean extensions to monitor the status of the loop. If you use an APT flag extension, you must use the LATCH, ON, and CLEAR math procedures, described in Chapter 11, "Math Functions and Procedures."

When the **.NRDY** extension is set to true, no data is written to **.OUT** (or **.LMN** for a Series 505, when **Map tuning and alarm data to S-memory** field is selected), and you cannot place the loop into automatic or cascade mode. If either **.GAUTO** or **.GCASC** is held true in controller logic when **.NRDY** is also true, the loop cycles back and forth between manual mode and the alternate (automatic or cascade) mode.

PID Blocks (continued)

Loops in Manual Mode

In manual mode, the loop computations are not performed, and the control program or operator can manipulate the output directly by writing to the **.OUT** (output) extension. For Series 505 controllers, you can also write to the **.LMN** (loop output) extension when **Map tuning and alarm data to S-memory** is selected (marked with X).

Loops in Automatic Mode

In automatic mode, the loop takes the setpoint from the **.SP** (setpoint) extension. Loop calculations are then performed, and the controller output is updated.

Loops in Cascade Mode

In cascade mode, the setpoint is provided by an external source. This source can be the output of another CFB, a declared real variable, or the output from another PID block.

If you want to connect the output of another loop to the setpoint, select **CASCADED** in the Setpoint Source field of the form. In this case, for Series 505, the setpoint is taken from the scaled integer output (0 to 32000) from the PID source block. For S5, the setpoint is taken from the normalized output (0–1) of the PID source block and is scaled to the engineering units of the PV of the cascaded loop.

In order to put a loop in cascade mode, you must follow this sequence.

1. Set the source (master) loop's output to the desired value (the setpoint for the slave loop).
2. Put the cascade (slave) loop in cascade mode.
3. Put the source (master) loop in automatic mode.

The master loop then feeds its output to the slave loop's setpoint. If a slave loop is switched out of cascade, then all of its master loops are placed in manual. A request to place a master loop in auto or cascade is denied unless the slave loop is in cascade.

If the setpoint source is any other real output, select **COMPUTED** in the Setpoint Source field of the form. In this case, the setpoint is taken from the variable that you entered in the form.

If the setpoint source is not configured as **CASCADED** or **COMPUTED**, you cannot use the CASCADE command.

You cannot edit the Setpoint Source field if you have already marked the PID block for OSx tag translation.

PID Extensions

A full list of all the PID extension variables that you can use to monitor and/or control a PID block is contained in Table 2-2. More information is provided concerning some of these extensions on the pages that follow.

Availability

All PID extensions are available for Series 505 controllers. Since S5 controllers must have real inputs and real outputs, the following integer extensions are not provided for S5: **.IPV**, **.IIN**, **.ISP**, **.IMN**, **.IOUT**, and **.IBIAS**.

Table 2-2 PID Extensions

Read-only Boolean		Read-only Integer		Read-only Real	
.IMAN	in manual mode	.IOUT	integer/analog output [3, 10]	.OUT	real output [3]
.IAUTO	in auto mode	.ECODE	error code [10]	.ERR	loop error
.ICASC	in cascade mode	.IID	error controller block [10]		
.SERR	error sign (1 = –, 0 = +)	.SNUM	error line number [10]		
.INHHA	PV > high-high alarm	.SMODE	block status		
.INHA	PV > high alarm	.VFLAG	status word		
.INLA	PV < low alarm	.IERR	loop error [4, 10]		
.INLLA	PV < low-low alarm				
.INYDA	error > yellow deviation				
.INODA	error > orange deviation				
.INRCA	rate of change > .RCA				
.INBTA	broken transmitter [10]				
.OVRUN	loop not completing				
Read/Write Boolean		**Read/Write Integer**		**Read/Write Real**	
.NRDY	not ready	.IPV	internal PV [5, 10]	.PV	real input [5]
.GMAN	go to manual [1]	.IIN	integer input [5, 10]	.LPV	internal scaled PV input [7]
.GAUTO	go to auto [1]	.ISP	internal SP [10]	.LSP	internal scaled SP input [7]
.GCASC	go to cascade [1]	.IMN	internal output [10]	.LMN	internal output [7]
.RMAN	go to manual [2]	.AWS	anti-reset windup [6]	.HLIM	high limit for output
.RATO	go to auto [2]	.CFL	C-flag low	.LLIM	low limit for output
.RCAS	go to cascade [2]	.CFH	C-flag high	.IN	real input [5]
		.IBIAS	integer value of bias [10]	.RSP	setpoint source [8]
				.BIAS	integral of loop error

PID Blocks (continued)

Table 2-2 PID Extensions (continued)

	Read/Write Real (continued)	
	.ST	sample time
	.PVH	PV high range
	.PVL	PV low range
	.KC	proportional gain
	.TI	integral time
	.TD	derivative time
	.HHA	high-high alarm limit
	.HA	high alarm limit
	.LA	low alarm limit
	.LLA	low-low alarm limit
	.ODA	orange deviation limit
	.YDA	yellow deviation limit
	.RCA	rate-of-change limit
	.SP	setpoint [9]
	.SPH	setpoint high limit
	.SPL	setpoint low limit

1	These extensions are manipulated by the command(s) associated with the block.
2	These booleans are APT flags that are manipulated by the Math procedures Latch, Clear, and On.
3	**.IOUT** is only created when the integer is selected in the output field. **.IOUT** is always read-only. **.OUT** is read-only when the block is in auto or cascade.
4	The **.IERR** extension is generated only when a Thermocouple or RTD (8WX version) input is configured.
5	For each input, APT creates **.IPV** if input value is an integer or **.PV** if value is a real number. The value of the **.IN** extension is identical to that of the **.PV** extension (**.IIN** is identical to **.IPV**).
6	If a PID block is associated with an Anti-Reset Windup block, the Anti-Reset Windup CFB controls this value.
7	Read-only, unless it is used inside associated math (or, for Series 505 controllers, unless **Map tuning and alarm data to S-memory** is selected (**X**)), in which case the extension is read/write.
8	If setpoint source is computed, this **.RSP** or a real variable can be entered as setpoint source. If setpoint source is cascaded, **.RSP** is an alternate name for the source loop **.OUT** extension variable.
9	The **.SP** extension maps to Series 505 S-memory and is the same as **.LSP** when **Map tuning and alarm data to S-memory** is selected (**X**).
10	These extensions are not supported for S5 controllers.

PID V-Flag and C-Flag Extension Variables

The PID **.VFLAG** extension variable includes the boolean extension variables that are listed in Table 2-3. Figure 2-9 shows the **.VFLAG** bits.

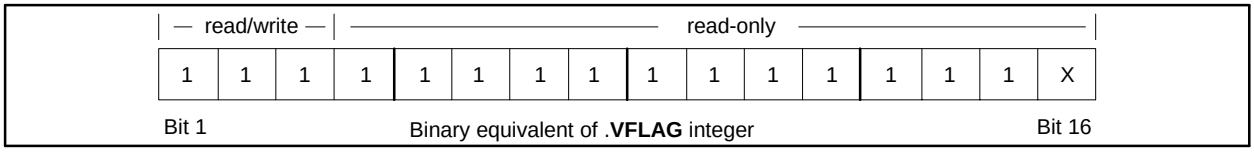


Figure 2-9 The .VFLAG Extension Bits

Table 2-3 Loop V-Flags

Bit Number		Function	Extension	
Read/Write	1	1=go to manual mode	**.GMAN**	boolean
	2	1=go to auto mode	**.GAUTO**	boolean
	3	1=go to cascade mode	**.GCASC**	boolean
Read Only	4	1=loop in auto mode	**.IAUTO**	boolean
	5	1=loop in cascade mode	**.ICASC**	boolean
	6	0=error is positive 1=error is negative	**.SERR**	boolean
	7	1=PV in high-high alarm	**.INHHA**	boolean
	8	1=PV in high alarm	**.INHA**	boolean
	9	1=PV in low alarm	**.INLA**	boolean
	10	1=PV in low-low alarm	**.INLLA**	boolean
	11	1=PV in yellow deviation alarm	**.INYDA**	boolean
	12	1=PV in orange deviation alarm	**.INODA**	boolean
	13	1=PV in rate-of-change alarm	**.INRCA**	boolean
	14	1=broken transmitter alarm	**.INBTA** *	boolean
	15	1=loop not within sample time	**.OVRUN**	boolean
	16	not used		

* The **.INBTA** extension is not provided for S5 controllers to use with the PID loop, but the broken transmitter is detected. For S5 controllers, a broken transmitter extension is provided as an extension on an analog input (*ai_name*.**BTA**).

The controller maintains a set of C-flag bits that contain additional loop operational data. While V-flag bits are held in one 16-bit extension variable (**.VFLAG**), it takes two (the **.CFL** and **.CFH** extensions) to contain the C-flag bits. See Figure 2-10.

The C-flag bits contained in the **.CFL** and **.CFH** extensions are initialized as shown in Table 2-4.

PID Blocks (continued)

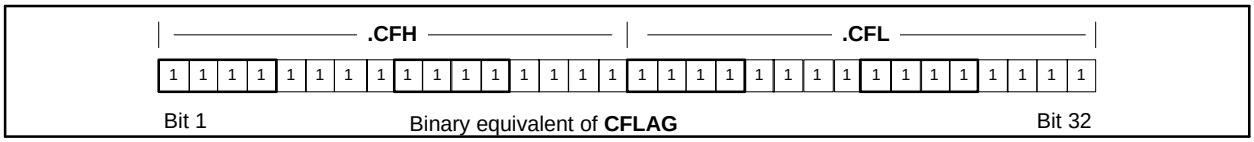


Figure 2-10 The .CFH and .CFL Extension Bits

Table 2-4 Loop C-Flags

Bit Number	Function	Source
1	0=PV scale 0% offset (unipolar only) 1=PV scale 20% offset (unipolar only)	I/O Table
2	1=take square root of PV	I/O Table
3	1=monitor high/low alarms	PID form
4	1=monitor high-high/low-low alarms	PID form
5	1=monitor yellow/orange deviations	PID form
6	1=monitor rate-of-change alarm	PID form
7	1=monitor broken transmitter alarm	PID form
8	0=position algorithm 1=velocity algorithm	PID form
9	0=direct acting 1=reverse acting	PID form
10	1=control based on error squared	PID form
11	1=control based on error deadband	PID form
12	1=auto-mode lock	set to 0
13	1=cascade-mode lock	set to 0
14	1=setpoint lock	set to 0
15	0=output scale 0% offset (unipolar only) 1=output scale 20% offset (unipolar only)	I/O Table
16	1=special function on process variable	set to 1
17	1=special function on setpoint	set to 1
18	1=freeze bias when output is out of range	PID form
19	1=ramp/soak on setpoint	set to 0
20	0=output is unipolar 1=output is bipolar	I/O Table
21	0=PV is unipolar 1=PV is bipolar	I/O Table
22	1= do derivative gain limiting	PID form
23-32	For Series 505, these are used to hold SFPGM number if bit 16 or bit 17 = 1	internal

Loop Status Extensions

The boolean extensions **.IMAN** (in manual), **.IAUTO** (in automatic), and **.ICASC** (in cascade) indicate the internal state of the loop. These bits do not, however, indicate whether the loop computations have completed initialization.

The **.SMODE** (status mode) extension provides information about the loop status. During loop operation, **.SMODE** can assume any one of the following seven integer values.

0: The program has just been downloaded into the controller, but the controller has not been switched to run mode.

1: The loop has just transitioned into manual mode from either automatic or cascade; or the loop has just transitioned into manual mode when the controller has just switched to run mode after a program download.

2: The loop is in manual mode and has been in this mode for at least one operation of the loop calculations, or for two seconds, whichever is less.

3: The loop has just transitioned into automatic mode from either manual or cascade, and the **.SP** (setpoint) and **.LSP** (loop setpoint) extensions have been initialized to the current value of the process variable. This makes the loop error equal to zero; the loop output does not change to provide a bumpless transfer.

4: The loop is in automatic mode and has been in this mode for at least one operation of the loop calculations. Normal control computations based on the setpoint and process variable are being performed.

5: The loop has just transitioned into cascade mode, and setpoint initialization is complete with a bumpless transfer to the **.RSP** extension.

6: The loop is in the remote or cascaded setpoint mode and has been in this mode for at least one operation of the loop calculations, and normal loop computations are being performed on the **.RSP** extension.

NOTE: Wait until **.SMODE** equals 3, 4, 5, or 6 before you change the loop setpoint from an SFC step, math, debug, or OSx; otherwise, the loop may overwrite the value with the process variable. To determine loop status, always check the value of the **.SMODE** extension, not the value of **.IMAN**, **.IAUTO**, or **.ICASC**. The loop always changes mode before APT finishes the calculation.

PID Blocks (continued)

Reset Windup Protection and the .AWS Extension

The **.AWS** extension defines the anti-reset windup status and can have any one of the following values during operation of the loop:

0: The output is within the constraints defined by **.HLIM** (high limit) and **.LLIM** (low limit) extensions.

1: The output is constrained at the low limit; if the loop tries to decrease the output by decreasing the bias, the output does not change. The output and bias may increase.

2: The output is constrained at the high limit; if the loop tries to increase the output by increasing the bias, the output does not change. The output and bias may decrease.

3: The block is no longer integrating, and the loop does not have a control path to the controlled variable. The output may still change if the process variable changes and gain > 0.

If you use external reset windup protection (See "Other Control Blocks" Chapter 9), or if you select **Limit output between 0% and 100%**, the **.AWS** extension indicates the status of the loop. However, you can write a 3 to the **.AWS** extension at any time to freeze the integral mode. Any other value in the **.AWS** extension resumes the integral mode.

Map Tuning and Alarm Data to 505 S-Memory

The S5 controllers store loop parameters in data blocks. However, for Series 505 controllers, all loop parameters are stored in S-memory. Table 2-5 lists the PID extensions and their associated variables in Series 505 S-memory. The **n** portion of the Series 505 S-memory variable name is the integer number of the loop. For example, LPV24 is the Series 505 S-memory variable that contains the process variable data for loop #24.

For Series 505 controllers, you can use more than one extension to read some S-memory variables, e.g., the LMXn variable. Whether you use an integer or a real extension depends on how the corresponding variable has been declared.

Table 2-5 PID S-Memory Extensions (Series 505 only)

Read/Write Integer		Read/Write Real		Read-only Real [1]	
Extension	**S-memory**	**Extension**	**S-memory**	**Extension**	**S-memory**
.CFH	LCFHn	.BIAS	LMXn.	.PIDL.KC	LKCn.
.CFL	LCFLn	.ERR	LERRn.	.PIDL.LHA	LHAn.
.IADB	LADBn	.LMN	LMNn.	.PIDL.LHHA	LHHAn.
.IBIAS	LMXn	.LPV	LPVn.	.PIDL.LLA	LLAn.
.IMN	LMNn	.LSP	LSPn.	.PIDL.LLLA	LLLAn.
.IPV	LPVn	.PIDL.ADB	LADBn.	.PIDL.LODA	LODAn.
.ISP	LSPn	.PIDL.BIAS	LMXn.	.PIDL.LRCA	LRCAn.
.PIDL.CFH	LCFHn	.PIDL.DGL	LKDn.	.PIDL.LYDA	LYDAn.
.PIDL.CFL	LCFLn	.PIDL.ERR	LERRn.	.PIDL.TD	LTDn.
.PIDL.IADB	LADBn	.PIDL.LPV	LPVn.	.PIDL.TI	LTIn.
.PIDL.IBIAS	LMXn	.PIDL.LSP	LSPn.		
.PIDL.IHA	LHAn	.PIDL.OUT [2]	LMNn.		
.PIDL.IHHA	LHHAn	.PIDL.PVH	LPVHn.		
.PIDL.ILA	LLAn	.PIDL.PVL	LPVLn.		
.PIDL.ILLA	LLLAn	.PIDL.SPH	LSPHn.		
.PIDL.IOUT [2]	LMNn	.PIDL.SPL	LSPLn.		
.PIDL.IPV	LPVn	.PIDL.ST	LTSn.		
.PIDL.ISP	LSPn	.PVH	LPVHn.		
.PIDL.VFLAG	LVFn	.PVL	LPVLn.		
.VFLAG	LVFn	.ST	LTSn.		

1 When the **Map tuning and alarm data to S-memory** option is not selected, APT writes over these Series 505 S-memory extensions with the contents of their associated Series 505 V-memory extensions as quickly as possible. Any user-entered data to the S-memory extensions is written over, effectively making these extensions read-only. Refer to Table 2-6.

2 Read-only in automatic and cascade modes; read/write in manual mode.

PID Blocks (continued)

When the **Map tuning and alarm data to S-memory** option is not selected for a Series 505 controller, APT maps the loop tuning parameters and alarm data to V-memory. APT also maps the **.SP** (setpoint) and **.OUT** (output). Table 2-6 lists the PID extensions in Series 505 V-memory that APT copies. APT writes over these Series 505 S-memory extensions with the contents of their associated V-memory extensions as quickly as possible. Any user-entered data to the Series 505 S-memory extensions is written over, effectively making S-memory extensions read-only.

When the **Map tuning and alarm data to S-memory** option is selected for a Series 505 controller, APT uses the loop tuning parameters and alarm data for that loop directly from S-memory. Using this option can save twelve V-memory locations per loop and the time required to move the data from V-memory to S-memory. This selection reduces CPU loading and allows non-OSx operator interfaces (such as CVU and the loop access module) to readily interact with the loop. You can use this feature by entering X in the **Map tuning and alarm data to S-memory** field of the PID form. This option makes the extensions, listed in Table 2-6, operate as read/write Series 505 S-memory locations, and causes the **.SP** extension to map to S-memory. Most of the time this option should be selected (X).

Table 2-6 PID V-Memory Extensions Copied to S-Memory Extensions (Series 505 only)

V-memory Extension	S-memory Extension	V-memory Extension	S-memory Extension
.KC	.PIDL.KC	.LA	.PIDL.LA
.TI	.PIDL.TI	.LLA	.PIDL.LLA
.TD	.PIDL.TD	.ODA	.PIDL.ODA
.HA	.PIDL.HA	.YDA	.PIDL.YDA
.HHA	.PIDL.HHA	.RCA	.PIDL.RCA
.SP	.LSP	.OUT	.PIDL.OUT

When you have an analog output that requires the energized closed (air-to-close) output option for a Series 505 controller, then the **Map tuning and alarm data to S-memory** option is not selected. In this manner, the **.OUT** can be manipulated before it is sent to the field because APT will use the **.OUT** extension in Series 505 V-memory to invert the signal. For air-to-close on analog output, **.LMN** goes from zero to one for the associated math on output and **.OUT** goes from one to zero after the associated math on output is complete.

You cannot edit the **Map tuning and alarm data to S-memory** field if you have already marked the PID block for OSx tag translation.

RTD and Thermocouple Module Extensions for Series 505

For Series 505 controllers, the status of Series 500/Series 505 RTD and Thermocouple modules is reported by the **.IERR** and **.INBTA** extensions. For S5 controllers, the status of RTD and Thermocouple is reported in the AI extension (**.BTA**) for the RTD or Thermocouple, not in the loop extensions.

For Series 505, the **.INBTA** extension becomes true when the associated module has an error, for example, a broken transmitter.

For Series 505, the **.IERR** extension normally equals the internally scaled PV of the the RTD or thermocouple input. The **.IERR** extension contains the module error code when:

- The module has an error and,
- You specified the RTD or Thermocouple name without an extension in the process variable field (all standard or advanced blocks except for the ratio station).

Refer to the Series 500/505 RTD or Thermocouple user manual for a definition of the error code. If you specified the RTD or Thermocouple name with the **.RAW** extension, then APT does not execute the **.IERR** logic.

2.3 On/Off Block

Overview

The on/off block shown in Figure 2-11 is essentially a high-gain proportional control block that contains a PID loop. If the absolute value of the loop error is greater than or equal to the value of the yellow deviation alarm, a discrete output (*cfb_name*.**DOUT** or a device) is manipulated.

The output from an on/off block can be any one of the following: digital output, boolean variable, or the name of a device (VSS, VMD, VSD, VSN, VDD, MDN, MSN, MSS).

If the output is the name of a device, the block expects the device to be in a locked state. (See the chapter on devices in the *SIMATIC APT Programming Reference (Tables) Manual* for information about locking devices.)

Availability

On/off blocks are available for Series 505 and S5 controllers. They are not supported for the 560/560T controllers.

Block Configuration

Figure 2-11 shows the graphic for an on/off CFB.

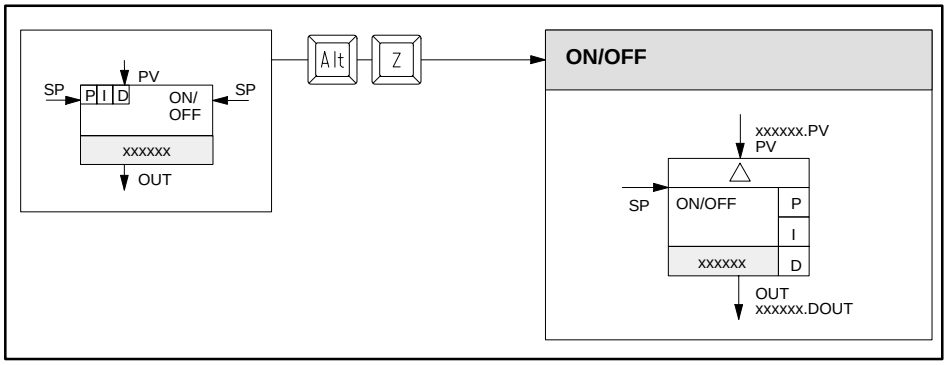


Figure 2-11 On/Off CFB

- If the controller is direct-acting, the output is set to true when the error is beyond the low yellow deviation. The output remains true until the PV passes through the positive yellow deviation as shown in Figure 2-12.
- If the controller is reverse-acting, the output is set to true when the error is beyond the high yellow deviation. The output remains true until the PV passes through the negative yellow deviation as shown in Figure 2-12.
- The **.IOUT** extension manipulates **.DOUT** and is read only when the block is in auto or cascade and read/write when the block is in manual. **.DOUT** is true when **.IOUT** = 1, and false when **.IOUT** = 0.

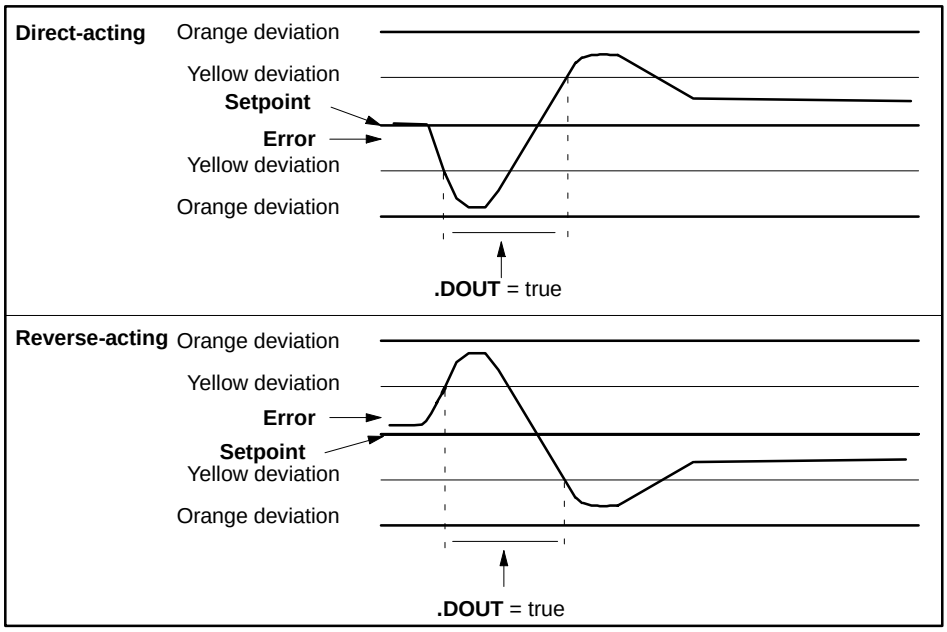


Figure 2-12 Direct and Reverse Acting On/Off Block

On/Off Block (continued)

Figure 2-13 shows the form that you use to define an on/off block. The shaded entries in the form depend on a previous selection. The information on the form is the same as for the PID block except that you cannot have an analog output or associated math on the output. See page 2-9 for process variable information, page 2-11 for process setpoint information, page 2-13 for output information, page 2-15 for controller options, and page 2-23 for PID extensions and for a description of the options that apply to the on/off block.

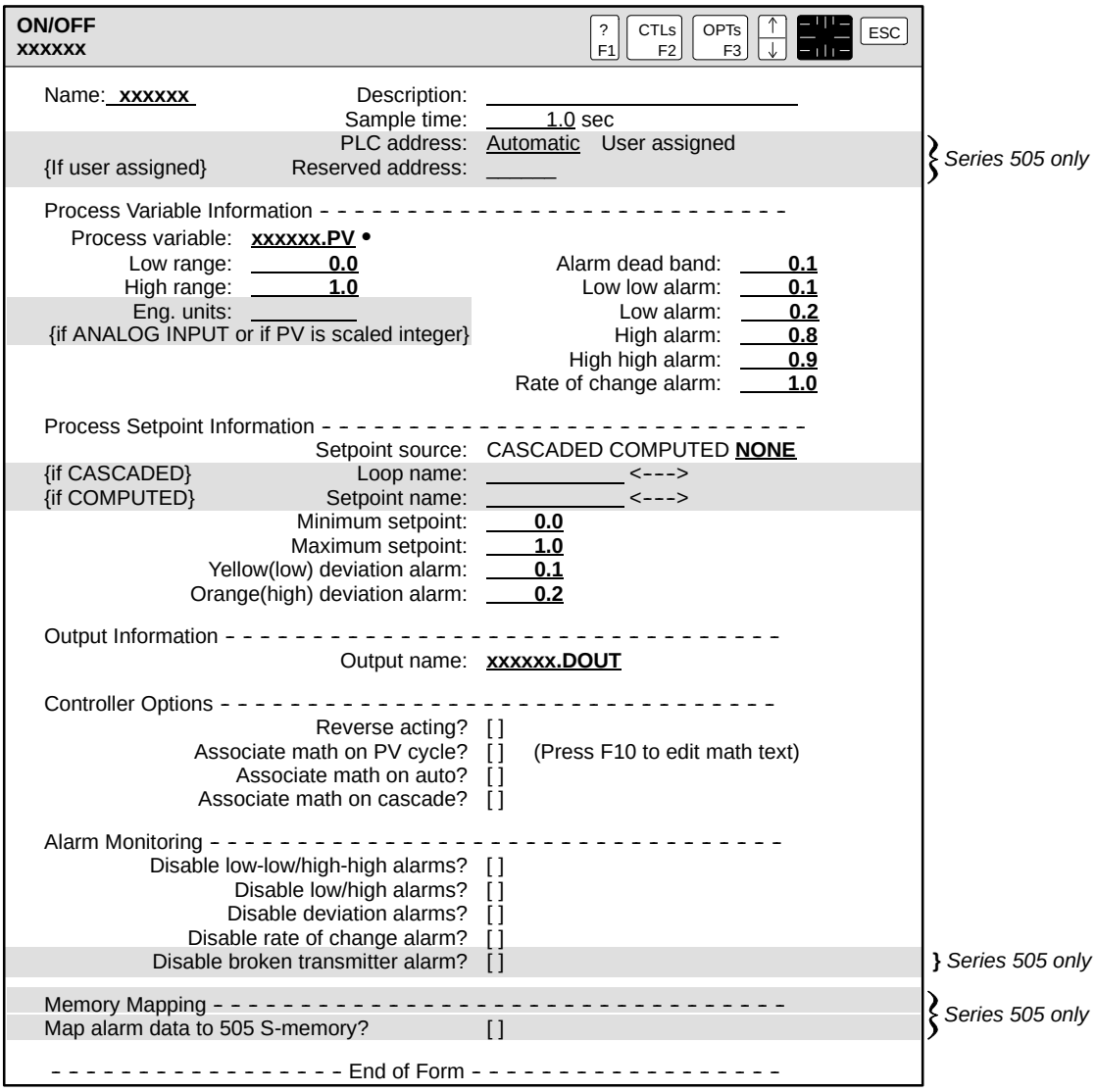


Figure 2-13 On/Off Form

On/Off Commands

Table 2-7 lists the commands that determine the mode of operation for the on/off block: MANUAL (MAN), AUTO, and CASCADE (CAS). These commands operate the same way that they do for the PID block. See page 2-21 for PID commands.

Table 2-7 On/Off Commands

Command	Description
AUTO	Places loop in automatic mode and gets setpoint from *cfb_name*.**SP**
MANUAL (MAN)	Places loop in manual mode; deactivates PID calculation; to manipulate output: modify **.IOUT** = 1 for **.DOUT** = true, and **.IOUT** = 0 for **.DOUT** = false
CASCADE (CAS)	Places loop into cascade mode; to change setpoint, modify the remote source (output from connected loop, block, or real variable).

On/Off Block (continued)

On/Off Extensions

Table 2-8 lists the extensions that you can use to monitor and control and on/off block. See page 2-23 for PID extensions and for details about these extensions. The V-flag and C-flag extension variables that are available for the on/off block operate the same way as those for the PID block; see Table 2-3 and Table 2-4.

Table 2-8 On/Off Extensions

Read-only Boolean		Read-only Integer		Read-only Real	
.IMAN	in manual mode	.ECODE	error code [10]	.ERR	alarm error
.IAUTO	in auto mode	.IID	error control block [10]		
.ICASC	in cascade mode	.SNUM	error line number [10]		
.SERR	error sign (1 = –, 0 = +)	.SMODE	block status		
.INHHA	PV > high-high alarm	.VFLAG	status word		
.INHA	PV > high alarm	.IERR	alarm error [1, 10]		
.INLA	PV < low alarm	.IOUT	manipulates .DOUT [2]		
.INLLA	PV < low-low alarm				
.INYDA	error > yellow deviation				
.INODA	error > orange deviation				
.INRCA	rate of change > .RCA				
.INBTA	broken transmitter [10]				
.OVRUN	loop not completing				
.DOUT	boolean output				

Table 2-8 On/Off Extensions (continued)

Read/Write Boolean		Read/Write Integer		Read/Write Real	
.NRDY	not ready	**.IPV**	internal PV [5, 10]	**.PV**	real input [5]
.GMAN	go to manual [3]	**.IIN**	integer input [5, 10]	**.LPV**	scaled PV input [6]
.GAUTO	go to auto [3]	**.ISP**	internal SP [10]	**.LSP**	scaled SP input [6]
.GCASC	go to cascade [3]	**.IMN**	internal output [10]	**.LMN**	internal output [7]
.RMAN	go to manual [4]	**.CFL**	C-flag low	**.RSP**	setpoint source [8]
.RATO	go to auto [4]	**.CFH**	C-flag high	**.ST**	sample time
.RCAS	go cascade [4]	**.IBIAS**	integer value of bias [10]	**.IN**	real input [5]
				.PVH	PV high range
				.PVL	PV low range
				.HHA	high-high alarm limit
				.HA	high alarm limit
				.LA	low alarm limit
				.LLA	low-low alarm limit
				.ODA	orange deviation limit
				.YDA	yellow deviation limit
				.RCA	rate-of-change limit
				.SP	setpoint [9]
				.SPH	setpoint high limit
				.SPL	setpoint low limit
				.BIAS	integral of loop error

1 The **.IERR** extension is generated only when a Series 505 Thermocouple or RTD (8WX version) input is configured.

2 The **.IOUT** extension manipulates **.DOUT** and is read only when the block is in auto or cascade and read/write when the block is in manual. **.DOUT** is true when **.IOUT** = 1, and false when **.IOUT** = 0.

3 These extensions are manipulated by the command(s) associated with the block.

4 These booleans are APT flags that are manipulated by the Math procedures Latch, Clear, and On.

5 For each input, APT creates **.IIN** if input value is an integer or **.IN** if value is a real number. The value of the **.PV** extension is identical to that of the **.IN** extension (**.IIN** is identical to **.IPV**).

6 Read-only, unless it is used inside associated math (or, for Series 505 controllers, unless **Map tuning and alarm data to S-memory** is selected (**X**)), in which case the extension is read/write.

7 **.LMN** has no effect on **.DOUT**

8 If setpoint source is computed, this **.RSP** or a real variable can be entered as setpoint source. If setpoint source is cascaded, **.RSP** is an alternate name for the source loop **.OUT** extension variable.

9 The **.SP** extension maps to Series 505 S-memory and is the same as **.LSP** when **Map tuning and alarm data to S-memory** is selected (**X**).

10 These extensions are not supported for S5 controllers.

2.4 Analog Alarm Block

Overview

The analog alarm block offers the same types of alarms as the other standard control blocks. You can use it to monitor a process variable, but not to control it. (The analog alarm block does not have an output nor does it have tuning parameters.) If you select an analog alarm, you use the high/low alarm capability of the loops and you are also able to set a setpoint and use the orange and yellow deviations from the setpoint as alarms.

You can select a setpoint source of computed, value, or internal.

- If you select a setpoint source of **computed**, the setpoint is provided by an external source. This source can be the output of another CFB, a declared real variable, or the output from another PID block.
- If you select a setpoint source of **value**, you specify the actual value of the setpoint when you define the block. This value can be changed only by editing the CFB form.
- If you select a setpoint source of **internal**, the value of the setpoint is determined by the value that you write to the *cfb_name*.**SP** extension.

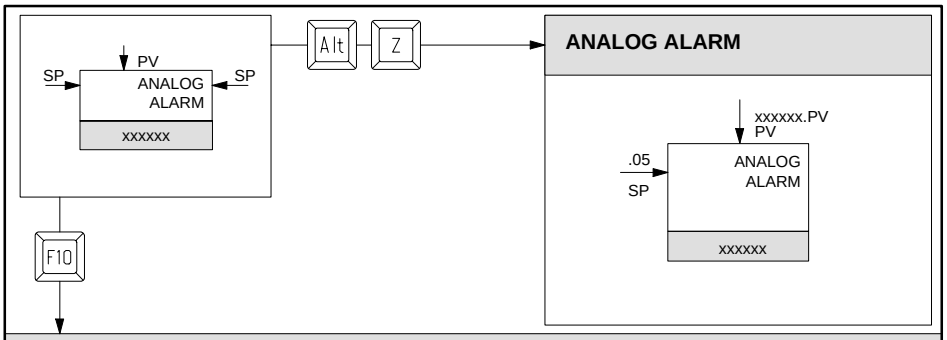
Figure 2-14 shows the form that you use to define an analog alarm block. The shaded entries in the form depend on a previous selection. The information on the form is the same as for the PID block, except that you have only one associated math section and it is always enabled. See page 2-9 for process variable information, and page 2-11 for process setpoint information and for a description of the options that apply to the analog alarm block.

The user-assigned address for an analog alarm is %AALMn.

Availability

Analog alarm blocks are available for both Series 505 and S5 controllers; however, for the Series 505, the analog alarm block is part of the firmware, while for the S5 the block has been implemented through code. Consequently, the S5 analog alarm block has several distinctive features:

- The Series 505 S-memory mapping feature is not supported for S5.
- The process variable for S5 controllers must be a real number, for instance, a scaled analog input.
- The PLC Reserved Address feature is not supported for S5.
- The broken transmitter signal is an AI extension and not an alarm extension for S5.



ANALOG ALARM
xxxxxx

? F1 | CTLs F2 | OPTs F3 | ↑ ↓ | ESC

Name: **xxxxxx** Description: _____
Sample time: 1.0 sec
PLC address: Automatic User Assigned
{If user assigned} Reserved address: _____ } *Series 505 only*

Process Variable Information –
Process variable: **xxxxxx.PV ●**
Low range: **0.0** Alarm dead band: **0.1**
High range: **1.0** Low low alarm: **0.1**
Eng. units: _____ Low alarm: **0.2**
{if ANALOG INPUT or if PV is scaled integer} High alarm: **0.8**
High high alarm: **0.9**
Rate of change alarm: **1.0**

Process Setpoint Information –
Setpoint source: COMPUTED VALUE **INTERNAL**
{if VALUE} Setpoint value: _____ <--->
{if COMPUTED} Setpoint name: _____ <--->
Minimum setpoint: **0.0**
Maximum setpoint: **1.0**
Yellow(low) deviation alarm: **0.1**
Orange(high) deviation alarm: **0.2**
Associate user math with alarm block? [ ] (Press F10 to edit math text)

Alarm Monitoring –
Disable low-low/high-high alarms? [ ]
Disable low/high alarms? [ ]
Disable deviation alarms? [ ]
Disable rate of change alarm? [ ]
Disable broken transmitter alarm? [ ] } *Series 505 only*

Memory Mapping –
Map alarm data to 505 S-memory? [ ] } *Series 505 only*

– – – – – – – – – – – – – – – – – End of Form – – – – – – – – – – – – – – – – –

Figure 2-14 Analog Alarm Form

Analog Alarm Block (continued)

Analog Alarm Commands

Table 2-9 lists the commands that determine the mode of operation for the analog alarm block: ENABLE (ENA) and DISABLE (DIS). Note that associated math is always executed.

Table 2-9 Analog Alarm Commands

Command	Description
ENABLE (ENA)	Turns on, or activates, the alarming.
DISABLE (DIS)	Turns off, or deactivates, the alarming.

Analog Alarm Extensions

Table 2-10 lists the extensions that you can use to monitor and control the analog alarm block.

The **.SMODE** extension defines the current status of the block and differs from the **.SMODE** extension for the PID.

0: The deviation alarms are not executing.

1: The deviation alarms are executing. Transitions can check for a 1 in the **.SMODE** variable to determine that the setpoint has been updated.

The internal representation of the scaled process variable is stored in the **.APV** extension variable. The setpoint is stored in the **.ASP** variable. If you want to manipulate either the process variable or the setpoint in associated math, you use these variables.

See page 2-23 for details about the other PID extensions.

Table 2-10 Analog Alarm Extensions

Read-only Boolean		Read-only Integer		Read-only Real	
.DSABL	disabled	**.ECODE**	error code [1]	**.ERR**	alarm error
.INHHA	PV > high-high alarm	**.IID**	error controller block [1]		
.INHA	PV > high alarm	**.SNUM**	error line number [1]		
.INLA	PV < low alarm	**.SMODE**	block status		
.INLLA	PV < low-low alarm	**.VFLAG**	status word		
.INYDA	error > yellow deviation	**.IERR**	alarm error [1, 2]		
.INODA	error > orange deviation				
.INRCA	rate of change > **.RCA**				
.INBTA	broken transmitter [1]				
.OVRUN	alarm not completing				

Table 2–10 Analog Alarm Extensions (continued)

Read/Write Boolean		Read/Write Integer		Read/Write Real	
.ENABL	enable [3]	**.IPV**	internal PV [1, 5]	**.PV**	real input [5]
.NRDY	not ready	**.IIN**	integer input [1, 5]	**.APV**	internal PV input [6]
.REN	request enable [4]	**.ISP**	internal SP [1]	**.ASP**	internal SP input [6]
.RDIS	request disable [4]	**.CFL**	C-flag low	**.RSP**	setpoint source [7]
		.CFH	C-flag high	**.ST**	sample time
				.IN	real input [5]
				.PVH	PV high range
				.PVL	PV low range
				.HHA	high-high alarm
				.HA	high alarm limit
				.LA	low alarm limit
				.LLA	low-low alarm limit
				.ADB	alarm deadband
				.ODA	orange deviation limit
				.YDA	yellow deviation limit
				.RCA	rate-of-change limit
				.SP	setpoint [8]
				.SPH	setpoint high limit
				.SPL	setpoint low limit

1 These extensions are not supported for S5 controllers.

2 The **.IERR** extension is generated only when a Series 505 Thermocouple or RTD input (8WX version) is configured.

3 This extension is manipulated by the command(s) associated with the block.

4 These booleans are APT flags that are manipulated by the Math procedures Latch, Clear, and On.

5 For each input, APT creates **.IPV** if input value is an integer, or **.PV** if value is a real number. The value of the extension is identical to that of the **.PV** extension (**.IIN** is identical to **.IPV**).

6 Read-only, unless used inside associated math or **Map tuning and alarm data to S-memory** selected (X).

7 If setpoint source is computed, this **.RSP** or a real variable can be entered as setpoint source.

8 The **.SP** extension maps to Series 505 S-memory and is the same as **.ASP** when **Map tuning and alarm data to S-memory** is selected (X).

Analog Alarm Block (continued)

Analog Alarm V-Flag and C-Flag Extension Variables

The analog alarm **.VFLAG** extension variable includes the boolean extension variables that are listed in Table 2-11. Figure 2-15 shows the **.VFLAG** bits. The **.VFLAG** value is updated when the block is enabled.

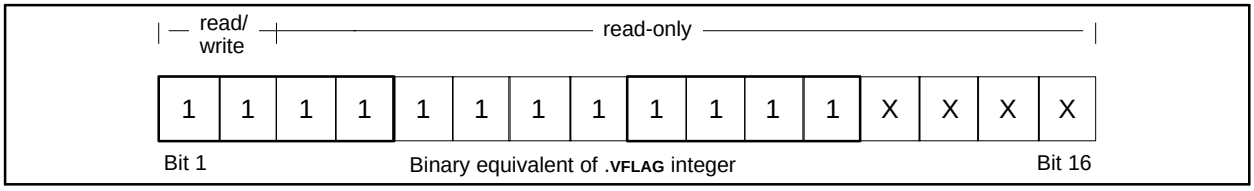


Figure 2-15 The .VFLAG Extension Bits

Table 2-11 Analog Alarm V-Flags

Bit Number		Function	Extension
Read/Write	1*	1=enable alarm	**.ENABL**
	2*	1=disable alarm	**.DSABL**
Read Only	3	1=PV in high-high alarm	**.INHHA**
	4	1=PV in high alarm	**.INHA**
	5	1=PV in low alarm	**.INLA**
	6	1=PV in low-low alarm	**.INLLA**
	7	1=PV in yellow deviation alarm	**.INYDA**
	8	1=PV in orange deviation alarm	**.INODA**
	9	1=PV in rate-of-change alarm	**.INRCA**
	10	1=broken transmitter alarm	**.INBTA** [1]
	11	1=analog alarm is overrunning	**.OVRUN**
	12	1=enabled	**.ENABL** [2]
	13–16	not used	

1 The **.INBTA** extension is for Series 505 controllers. If you have an S5, use the analog input **.BTA** extension.
2 Write a 1 to bits 1 and 2 to enable/disable the alarm. To determine the status of the alarm, read bit 12. If you attempt to read bits 1 or 2, status is not returned because these bits are always cleared.

The controller maintains a set of C-flag bits that contain additional operational data. While V-flag bits are held in one 16-bit extension variable (**.VFLAG**), it takes two (the **.CFL** and **.CFH** extensions) to contain the C-flag bits. See Figure 2-16.

The **.CFL** and **.CFH** extensions are initialized as shown in Table 2-12.

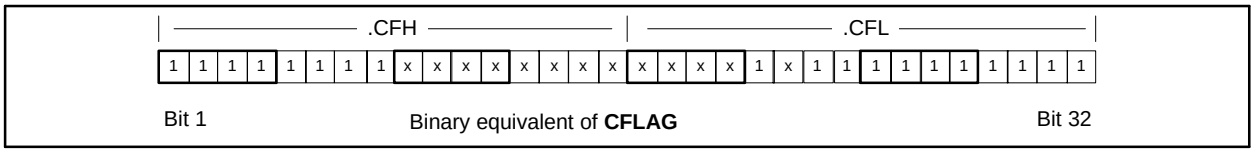


Figure 2-16 The .CFH and .CFL Extension Bits

Table 2-12 Analog Alarm C-Flags

Bit Number	Function	Source
1	0=PV scale 0% offset 1=PV scale 20% offset	I/O Table
2	1=take square root of PV	I/O Table
3	1=monitor high/low alarms	Analog form
4	1=monitor high-high/low-low alarms	Analog form
5	1=monitor yellow/orange deviations	Analog form
6	1=monitor rate-of-change alarm	Analog form
7	1=monitor broken transmitter alarm	Analog form
8	0=local setpoint 1=remote setpoint	Analog form
9–20	not used	
21	0=PV is unipolar 1=PV is bipolar	I/O Table
22	not used	
23–32	For Series 505, these are used to hold SFPGM number if bit 16 or bit 17 = 1	internal

Associated Math

To expand the capability of the analog alarm block, you can append Math computations. For Series 505, the math compiles as SFPGM; for S5, associated math compiles as STL.

You can use the associated math to enable the block using **Name.ENABL := true;** in the associated math section.

To modify the process variable from within an associated math block, manipulate the **.APV** extension. To modify the setpoint variable, manipulate the **.ASP** extension.

NOTE: The associated math for the analog alarm CFB always executes even when the block is not enabled.

Analog Alarm Block (continued)

S-Memory Variables (Series 505 only)

Table 2-13 lists the analog alarm extensions and their associated variables in Series 505 S-memory. The **n** portion of the S-memory variable is the integer number of the analog alarm. For example, APV55 is the S-memory variable that contains the process variable data for analog alarm #55.

You can use more than one extension to read some Series 505 S-memory variables, e.g., APVn.

Table 2-13 Analog Alarm Series 505 S-Memory Extensions

Read/Write Real		Read-only Real [1]	
Extension	**S-memory**	**Extension**	**S-memory**
.AL_RECORD.ADB	AADBn.	.AL_RECORD.AHA	AHAn.
.AL_RECORD.AERR	AERRn.	.AL_RECORD.AHHA	AHHAn.
.AL_RECORD.PV	APVn.	.AL_RECORD.ALA	ALAn.
.AL_RECORD.PVH	APVHn.	.AL_RECORD.ALLA	ALLAn.
.AL_RECORD.PVL	APVLn.	.AL_RECORD.AODA	AODAn.
.AL_RECORD.SP	ASPn.	.AL_RECORD.ARCA	ARCAn.
.AL_RECORD.SPH	ASPHn.	.AL_RECORD.AYDA	AYDAn.
.AL_RECORD.SPL	ASPLn.		
.AL_RECORD.ST	ATSn.		
.APV	APVn.		
.APVH	APVHn.		
.APVL	APVLn.		
.ASP	ASPn.		
.ERR	AERRn.		
.PVH	APVHn.		
.PVL	APVLn.		
.ST	ATSn.		

1 For Series 505, when the **Map alarm data to S-memory** option is not selected, APT writes over these Series 505 S-memory extensions with the contents of their associated V-memory extensions as quickly as possible. Any user-entered data to the S-memory extensions is written over, effectively making these extensions read-only. Refer to Table 2-14.

Table 2-13 Analog Alarm Series 505 S-Memory Extensions (continued)

Read/Write Integer	
Extension	**S-Memory**
.AL_RECORD.CFH	ACFHn.
.AL_RECORD.CFL	ACFLn.
.AL_RECORD.IHA	AHAn.
.AL_RECORD.IHHA	AHHAn.
.AL_RECORD.ILA	ALAn.
.AL_RECORD.ILLA	ALLAn.
.AL_RECORD.IPA	APVn.
.AL_RECORD.ISP	ASPNn.
.AL_RECORD.VFLAG	AVFn.
.CFH	ACFHn.
.CFL	ACFLn.
.IPV	APVn.
.ISP	ASPn.
.VFLAG	AVFn.

Table 2-14 lists the analog alarm extensions in Series 505 V-memory that APT maps to Series 505 read-only S-memory. APT writes over these Series 505 S-memory locations with the contents of their associated V-memory locations as quickly as possible. Any user-entered data to the Series 505 S-memory extensions is written over, effectively making these extensions read-only. You can disable this feature by entering X in the Map tuning and alarm data to S-memory field of the analog alarm form. This option makes these extensions (**.HA**, **.HHA**, **.LA**, etc.) read/write S-memory locations, and causes the **.SP** extension to map to Series 505 S-memory only, not to V-memory.

You cannot edit the **Map alarm data to S-memory** field if you have already marked the analog alarm block for OSx tag translation.

Table 2-14 Analog Alarm Series 505 V-Memory Extensions Copied to S-Memory Extensions

V-Memory Extension	S-Memory Extension
.HA	.AL_RECORD.AHA
.HHA	.AL_RECORD.AHHA
.LA	.AL_RECORD.ALA
.LLA	.AL_RECORD.ALLA
.ODA	.AL_RECORD.AODA
.YDA	.AL_RECORD.AYDA
.SP	.AL_RECORD.SP

Chapter 3

Dynamic Control Blocks

3.1 Understanding Dynamic Blocks

Overview

The APT dynamic blocks provide a means to create simulations and to build models for complex control strategies. APT provides three basic types of dynamic blocks:

- The simulation blocks allow you to add dynamic characteristics to the process and include first and second order lag blocks, first and second order lead lag blocks, and a dead time compensator.
- The derivative block simply computes the rate of change of the input. To determine this rate, the block computes the change in the input from the last sample time and then divides this change value by the sample time.
- The integrator block produces an output that is proportional to the integral of the input (or the sum of the inputs) with respect to the sample time.

Availability

Dynamic blocks are available for Series 505 and S5 controllers. They are not supported for the 560/560T controllers.

Enabling and Disabling Blocks

All dynamic CFBs must be activated with the ENABLE command or with an assignment statement that sets the .ENABL extension to true. See Figure 3-1. Dynamic blocks can also be enabled from an assignment statement in a math CFB.

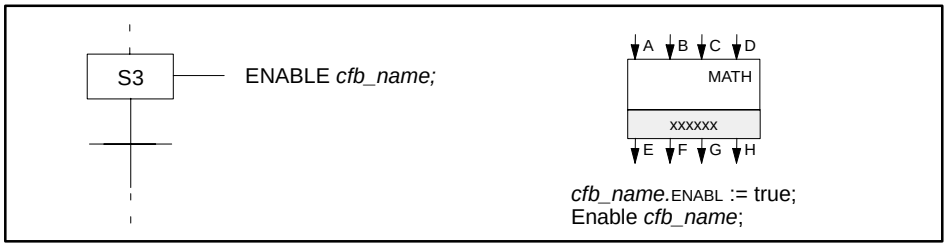


Figure 3-1 Enabling a Dynamic CFB

To deactivate the block, use the DISABLE command with the block name or set the .ENABL extension to false.

All dynamic blocks have a boolean extension (.NRDY) that you can use to interlock the execution of the block to some external conditions.

- If you set the .NRDY extension to true, the block is forced to the disabled state.
- To return the block to the enabled state you must not only set the .NRDY extension to false but also re-enable the block with a command or an assignment statement.

Initialization

Two types of initialization are available with the simulation blocks. Forward initialization allows straight simulations; backward initialization supports model base control where the actual process variable defines the initial value of the output. No initialization of the output or input is done for the derivative blocks.

The upper section of Figure 3-2 illustrates forward initialization, which operates as follows:

- The output is set to the value that you specify as the initial output (.OUTIC); this value serves as a bias that is added to an internal variable in the simulation block.
- The current input value is stored internally (.INIC) in the block as the initial input. Changes in the output are then computed relative to changes in the difference between the new, current input and the value stored as the initial input.

The bottom part of Figure 3-2 shows backward initialization, which operates as follows:

- The current value of the output from the block is stored as the initial output (.OUTIC).
- The expected value of the input is computed by dividing the output by the steady-state gain. The expected input is moved to the block input and is also stored internally as the initial input (.INIC). For the integrator block, the current value of the input is stored internally in the block as the initial input (.INIC).

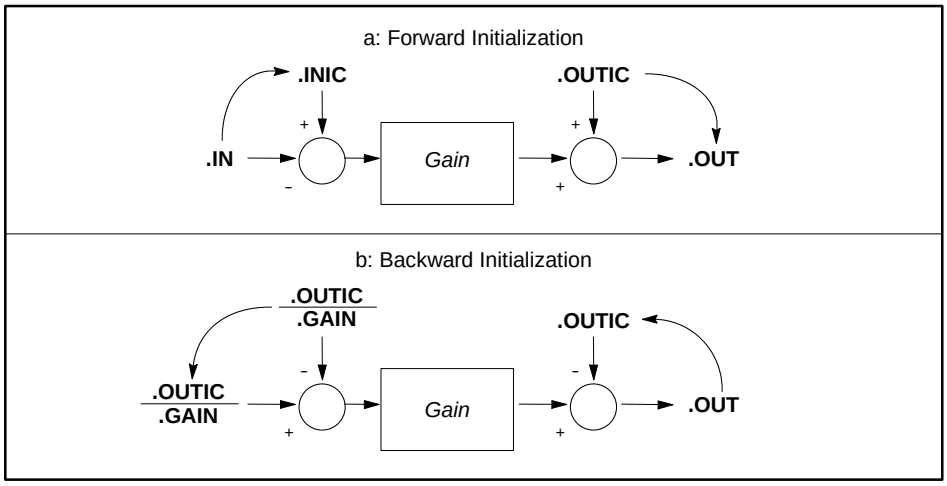


Figure 3-2 Forward and Backward Initialization

All simulation calculations are relative to the difference between the input and the .INIC extension variable. This means that after an initialization, either the input or the .INIC must change before the output can start to change.

Following either forward or backward initialization, a delay table is initialized so that the output does not respond to changes in the input for the time specified by the ratio of dead time to sample time.

NOTE: The dynamic block does not update the output until the second sample time following initialization.

Simulation Equations

Following initialization, the first and second order lags and first and second order lead lags function as follows.

- Each sample time after the initialization scan, the block updates the delay table using the following simulation formula:

$$\textit{Table entry} = B1 \times (\textit{input} - \textit{initial input}) + B2 \times (\textit{last input} - \textit{initial input}) - \left[A1 \times (\textit{table entry} - 1) + A2 \times (\textit{table entry} - 2)\right] + \textit{initial output}$$

  The values of B1, B2, A1, and A2 depend on the type of simulation block (first order lag, second order lag, etc.) and the steady-state gain that you specify in the form.

- The next output from the simulation block is determined by the table entry computed *n* samples ago where *n* is the ratio of delay time to sample time.

NOTE: The ratio of dead time to sample time should not exceed 62. The dead time table that simulates the delay has a length of 62 entries.

The delayed output of the block is always available as the output name that you enter in the form. The intermediate output without the delay is also available as *cfb_name*.DMOUT.

If you choose a first or second order lag, or a first or second order lead lag, APT creates three extensions that you can use to control the response to the change in input. These extensions are .TAU1, .TAU2, and .TLEAD.

- If you choose a first order lag, .TAU1 contains the value that you enter as the first time constant; the system initializes .TAU2, and .TLEAD to 0.

- If you choose a second order lag, .TAU1 and .TAU2 contain the values that you enter as the first and second time constants; the system initializes .TLEAD to 0.

- If you choose a first-order lead lag, .TAU1 and .TLEAD contain the values that you enter as the lag time and lead time; the system initializes .TAU2 to 0.

- If you choose a second-order lead lag, .TAU1, .TAU2, and .TLEAD contain the values that you enter as the first and second time constants and the lead time.

NOTE: Before you change the input or use the output of a dynamic block, check the status of the .ENABLD extension to be certain that the value is true.

3.2 Second Order Lead Lag

Overview

The second order lead lag block simulates a dynamic process in which specified constants determine the relationship between an immediate change in the input and a gradual, corresponding change in the output.

Figure 3-3 shows the formula that is used for a second order lag block.

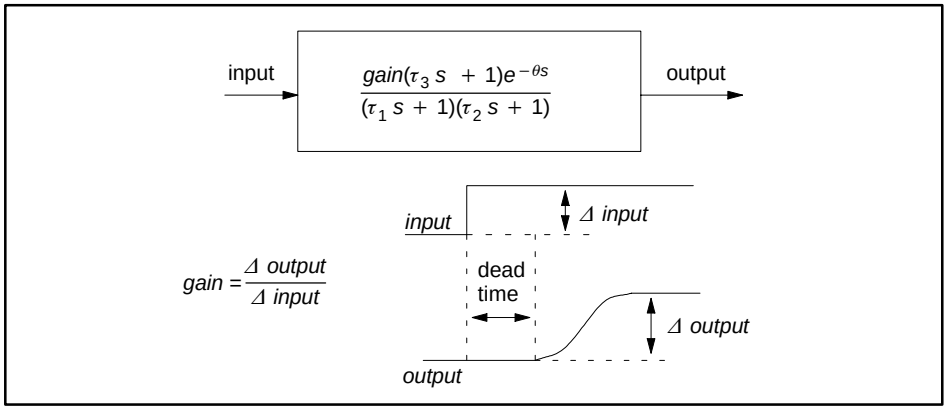


Figure 3-3 Second Order Lead Lag

- Gain specifies the ratio of the change in output to the change in input at a steady state.
- τ_1, τ_2, and τ_3 determine the characteristic "S" shape of the output response to a change in input. τ_1, τ_2, and τ_3 are the first order time constant, the second order time constant and the lead time constant, respectively.
- θ is the dead time in seconds.
- *s* is the LaPlace operator.

Availability

Second order lead lag blocks are available for Series 505 and S5 controllers. They are not supported for the 560/560T controllers.

Second Order Lead Lag (continued)

Block Configuration

Figure 3-4 shows the form that you use to define a second order lead lag block.

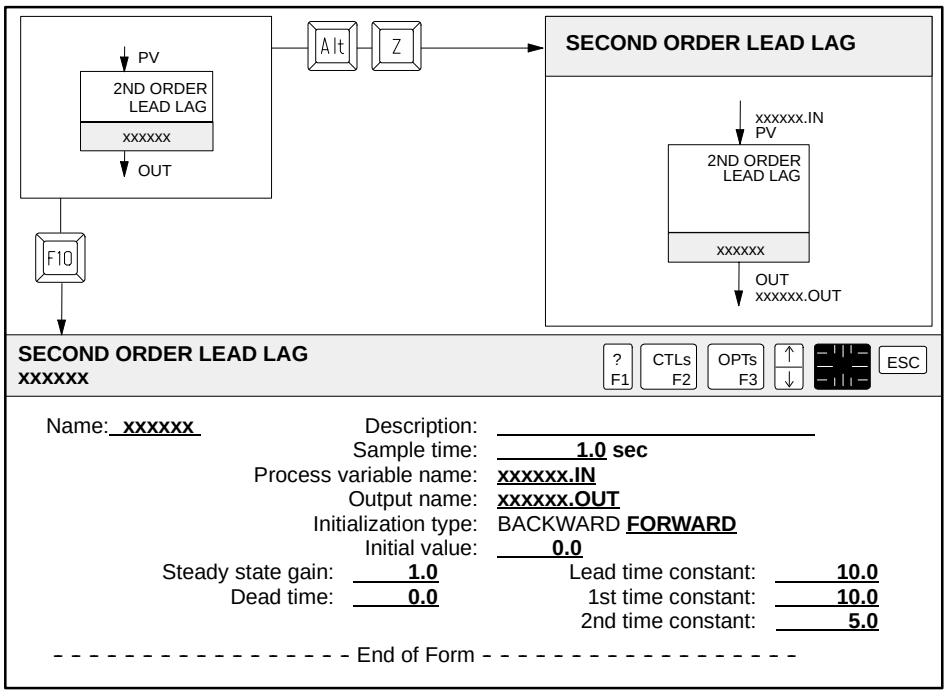


Figure 3-4 Second Order Lead Lag Form

Sample time: sample time in seconds; default is 1; value can be in half-second increments between 0.5 and 4095.5.

Process variable name: name of process variable input (PV); default is *cfb_name*.IN.

Output name: name of output (OUT); default is *cfb_name*.OUT.

Initialization type: FORWARD or BACKWARD; default is FORWARD.

Initial value: initial value; default is zero; appears only if you select FORWARD initialization.

Steady state gain: steady state gain (the change in output divided by the change in input); default is one.

Dead time: dead time in seconds; default is zero.

Lead time constant: lead time in seconds; default is ten.

First time constant: first lag time in seconds; default is ten.

Second time constant: second lag time in seconds; default is five.

Commands and Extensions

Table 3-1 lists the commands that determine the mode of operation for the second order lead lag block.

Table 3-1 Second Order Lead Lag Commands

Command	Description
ENABLE (ENA)	Turns on, or activates, the block.
DISABLE (DIS)	Turns off, or deactivates, the block.

Table 3-2 lists the extensions that you can use to monitor and control a second order lead lag block.

Table 3-2 Second Order Lead Lag Extensions

Read-only Boolean		Read-only Integer		Read-only Real	
.ENABLD	enabled	.IOUT	integer output [3]	.OUT	output [3]
		.ECODE	error code [4]	.DMOUT	output minus delay
		.IID	error controller block [4]		
		.SNUM	error line number [4]		
Read/Write Boolean		**Read/Write Integer**		**Read/Write Real**	
.ENABL	enable [1]			.IN	input
.NRDY	not ready			.INIC	initial input
.REN	request enable [2]			.OUTIC	initial output
.RDIS	request disable [2]			.GAIN	steady-state gain
				.TAU1	time constant one
				.TAU2	time constant two
				.TLEAD	lead time
				.DTIME	dead time

1 This extension is manipulated by the command(s) associated with the block.

2 These booleans are APT flags that are manipulated by the Math procedures Latch, Clear, and On.

3 For each output, APT creates .IOUT if output value is an integer or .OUT if value is a real number. The outputs are read-only variables if the block is enabled and read/write when the block is disabled.

4 These extensions are not supported for S5 controllers.

3.3 Second Order Lag

Overview

The second order lag block simulates a dynamic process in which specified time constants determine the relationship between an immediate change in the input and a gradual corresponding change in the output.

Figure 3-5 shows the formula that is used for a second order lag block.

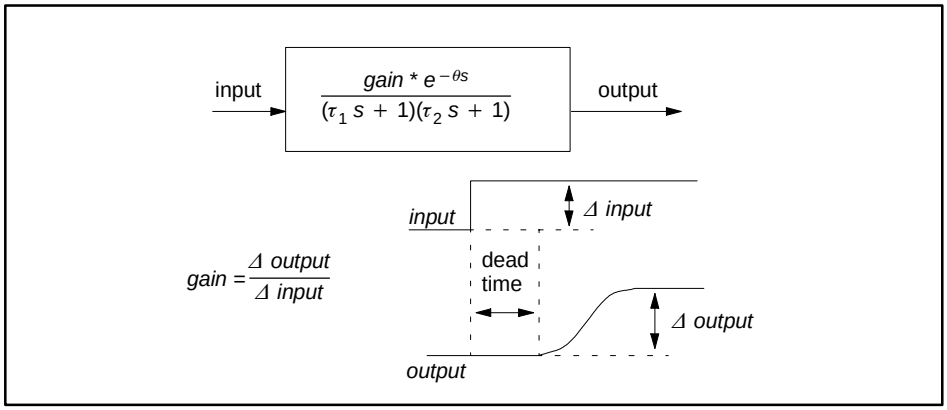


Figure 3-5 Second Order Lag

- Gain specifies the ratio of the change in output to the change in input at a steady state.
- τ_1 and τ_2 determine the characteristic "S" shape of the output response to a change in input. τ_1 and τ_2 are the first order time constant and the second order time constant, respectively.
- θ is the dead time in seconds.
- *s* is the LaPlace operator.

Availability

Second order lag blocks are available for Series 505 and S5 controllers. They are not supported for the 560/560T controllers.

Block Configuration

Figure 3-6 shows the form that you use to define a second order lag block.

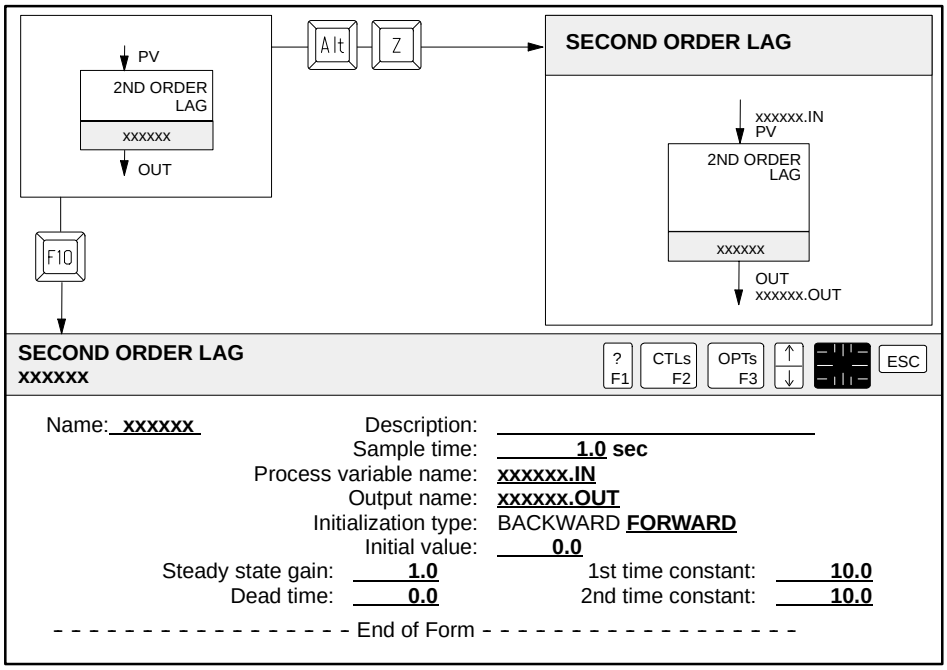


Figure 3-6 Second Order Lag Form

Sample time: sample time in seconds; default is 1; value can be in half-second increments between 0.5 and 4095.5.

Process variable name: name of process variable input (PV); default is *cfb_name*.IN.

Output name: name of output (OUT); default is *cfb_name*.OUT.

Initialization type: FORWARD or BACKWARD; default is FORWARD.

Initial value: initial value; default is zero; appears only if you select FORWARD initialization.

Steady state gain: steady state gain (the change in output divided by the change in input); default is one.

Dead time: dead time in seconds; default is zero.

First time constant: first lag time in seconds; default is ten.

Second time constant: second lag time in seconds; default is ten.

Second Order Lag (continued)

Commands and Extensions

Table 3-3 lists the commands that determine the mode of operation for the second order lag block.

Table 3-3 Second Order Lag Commands

Command	Description
ENABLE (ENA)	Turns on, or activates, the block.
DISABLE (DIS)	Turns off, or deactivates, the block.

Table 3-4 lists the extensions that you can use to monitor and control a Second Order Lag block.

Table 3-4 Second Order Lag Extensions

Read-only Boolean		Read-only Integer		Read-only Real	
.ENABLD	enabled	.IOUT	integer output [3]	.OUT	output [3]
		.ECODE	error code [4]	.DMOUT	output minus delay
		.IID	error controller block [4]		
		.SNUM	error line number [4]		
Read/Write Boolean		**Read/Write Integer**		**Read/Write Real**	
.ENABL	enable [1]			.IN	input
.NRDY	not ready			.INIC	initial input
.REN	request enable [2]			.OUTIC	initial output
.RDIS	request disable [2]			.GAIN	steady-state gain
				.TAU1	time constant one
				.TAU2	time constant two
				.DTIME	dead time

1 This extension is manipulated by the command(s) associated with the block.

2 These booleans are APT flags that are manipulated by the Math procedures Latch, Clear, and On.

3 For each output, APT creates .IOUT if output value is an integer or .OUT if value is a real number. The outputs are read-only variables if the block is enabled and read/write when the block is disabled.

4 These extensions are not supported for S5 controllers.

3.4 First Order Lead Lag

Overview

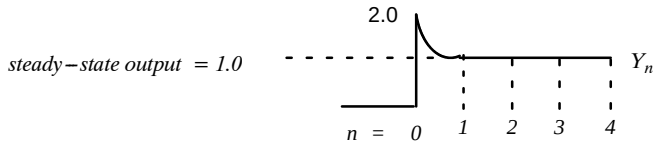
The first order lead lag block simulates a dynamic process in which specified time constants determine both lead and lag times.

The output depends on the ratio of lead to lag as explained below. Assume the following values in each example:

$$\Delta\ input\ =\ 1.0 \qquad gain\ =\ 1.0$$

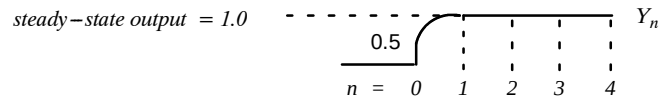
- If $T_{Lead}\ /\ T_{Lag}$ is greater than 1.0, then the initial response overshoots the steady-state output value.

$$Initial\ output\ =\ \Delta input\ *\ Gain\left(\frac{T_{Lead}}{T_{Lag}}\right)\ =\ 1.0\ *\ 1.0\left(\frac{2.0}{1.0}\right)\ =\ 2.0$$



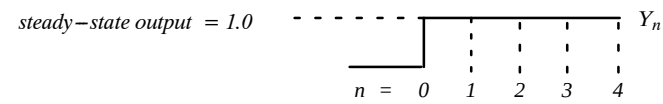
- If $T_{Lead}\ /\ T_{Lag}$ is less than 1.0, then the initial response undershoots the steady-state output value.

$$Initial\ output\ =\ \Delta input\ *\ Gain\left(\frac{T_{Lead}}{T_{Lag}}\right)\ =\ 1.0\ *\ 1.0\left(\frac{1.0}{2.0}\right)\ =\ 0.5$$



- If $T_{Lead}\ /\ T_{Lag}$ is equal to 1.0, then the initial response instantaneously reaches the steady-state output value.

$$Initial\ output\ =\ \Delta input\ *\ Gain\left(\frac{T_{Lead}}{T_{Lag}}\right)\ =\ 1.0\ *\ 1.0\left(\frac{1.0}{1.0}\right)\ =\ 1.0$$



First Order Lead Lag (continued)

Figure 3-7 shows the formula that is used for a first order lead lag block.

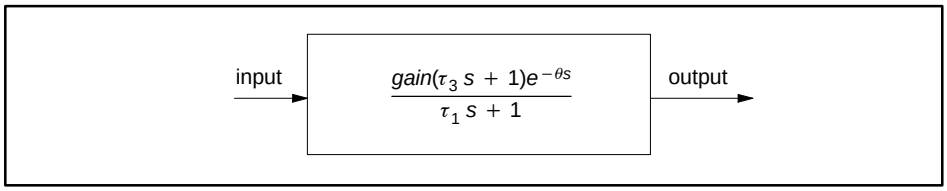


Figure 3-7 First Order Lead Lag

- Gain specifies the ratio of the change in output to the change in input at a steady state.
- τ_1 specifies a time in seconds that is associated with the response to change in the input that causes 63.2% of the change in output. It is called the lag time constant.
- τ_3 specifies a time in seconds that is associated with the lead response after a change in the input. It is called the lead time constant.
- θ is the dead time in seconds.
- s is the LaPlace operator.

Availability

First order lead lag blocks are available for Series 505 and S5 controllers. They are not supported for the 560/560T controllers.

Block Configuration

Figure 3-8 shows the form that you use to define a first order lead lag block.

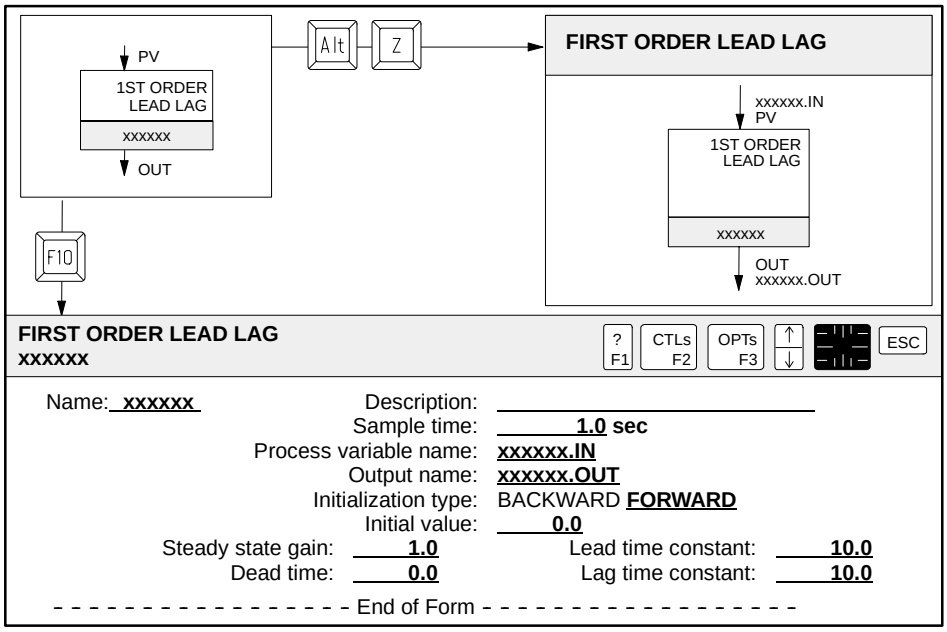


Figure 3-8 First Order Lead Lag Form

Sample time: sample time in seconds; default is 1; value can be in half-second increments between 0.5 and 4095.5.

Process variable name: name of process variable input (PV); default is *cfb_name*.IN.

Output name: name of output (OUT); default is *cfb_name*.OUT.

Initialization type: FORWARD or BACKWARD; default is FORWARD.

Initial value: initial value; default is zero; appears only if you select FORWARD initialization.

Steady state gain: steady state gain (the change in output divided by the change in input); default is one.

Dead time: dead time in seconds; default is zero.

Lead time constant: lead time in seconds; default is ten.

Lag time constant: lag time in seconds; default is ten.

First Order Lead Lag (continued)

Commands and Extensions

Table 3-5 lists the commands that determine the mode of operation for the first order lead lag block.

Table 3-5 First Order Lead Lag Commands

Command	Description
ENABLE (ENA)	Turns on, or activates, the block.
DISABLE (DIS)	Turns off, or deactivates, the block.

Table 3-6 lists the extensions that you can use to monitor and control a first order lead lag block.

Table 3-6 First Order Lead Lag Extensions

Read-only Boolean		Read-only Integer		Read-only Real	
.ENABLD	enabled	.IOUT	integer output [3]	.OUT	output [3]
		.ECODE	error code [4]	.DMOUT	output minus delay
		.IID	error controller block [4]		
		.SNUM	error line number [4]		
Read/Write Boolean		**Read/Write Integer**		**Read/Write Real**	
.ENABL	enable [1]			.IN	input
.NRDY	not ready			.INIC	initial input
.REN	request enable [2]			.OUTIC	initial output
.RDIS	request disable [2]			.GAIN	steady-state gain
				.TAU1	time constant one
				.TLEAD	lead time
				.DTIME	dead time

1 This extension is manipulated by the command(s) associated with the block.

2 These booleans are APT flags that are manipulated by the Math procedures Latch, Clear, and On.

3 For each output, APT creates .IOUT if output value is an integer or .OUT if value is a real number. The outputs are read-only variables if the block is enabled and read/write when the block is disabled.

4 These extensions are not supported for S5 controllers.

3.5 First Order Lag

Overview

The first order lag block simulates a dynamic process in which a specified first order time constant determines the relationship between an immediate change in the input and a gradual corresponding change in the output.

Figure 3-9 shows the formula that is used for a first order lag block.

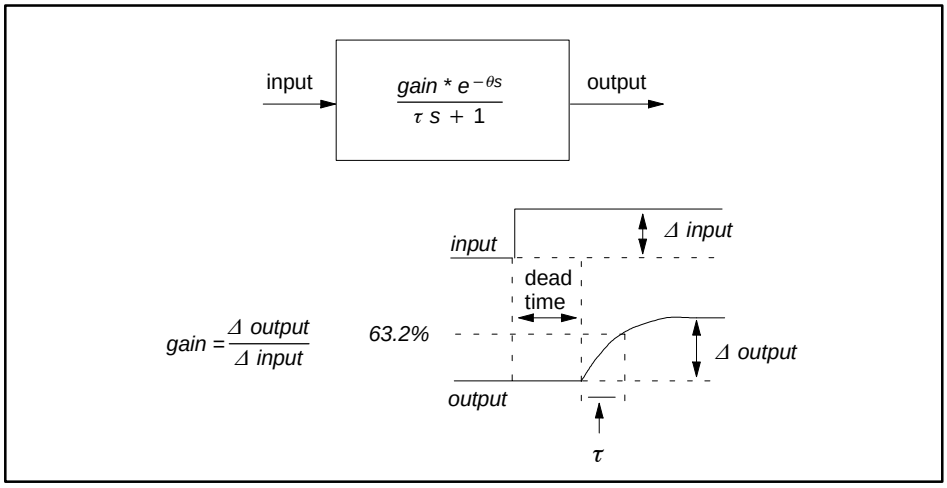


Figure 3-9 First Order Lag

- Gain specifies the ratio of the change in output to the change in input at a steady state.
- τ specifies the time in seconds required to reach 63.2 per cent of the final output after a change in the input. It is the time constant.
- θ is the dead time in seconds.
- *s* is the LaPlace operator.

Availability

First order lag blocks are available for Series 505 and S5 controllers. They are not supported for the 560/560T controllers.

First Order Lag (continued)

Block Configuration

Figure 3-10 shows the form that you use to define a first order lag block.

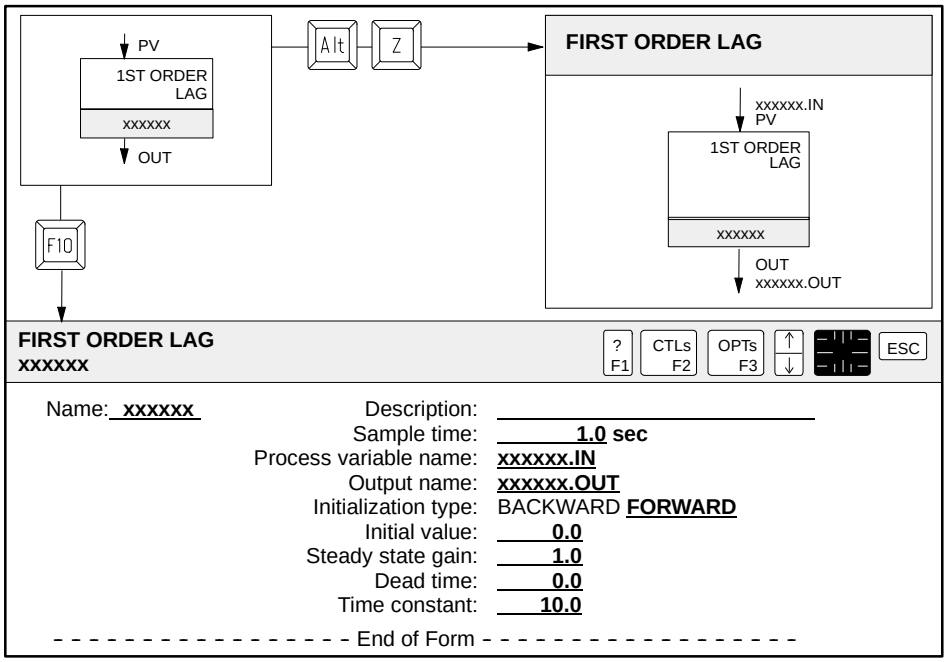


Figure 3-10 First Order Lag Form

Sample time: sample time in seconds; default is 1; value can be in half-second increments between 0.5 and 4095.5.

Process variable name: name of process variable input (PV); default is *cfb_name*.IN.

Output name: name of output (OUT); default is *cfb_name*.OUT.

Initialization type: FORWARD or BACKWARD; default is FORWARD.

Initial value: initial value; default is zero; appears only if you select FORWARD initialization.

Steady state gain: steady state gain (the change in output divided by the change in input); default is one.

Dead time: dead time in seconds; default is zero.

Time constant: time in seconds for output to reach 63.2% of final steady state value when responding to a step change in input; default is ten.

Commands and Extensions

Table 3-7 lists the commands that determine the mode of operation for the first order lag block.

Table 3-7 First Order Lag Commands

Command	Description
ENABLE (ENA)	Turns on, or activates, the block.
DISABLE (DIS)	Turns off, or deactivates, the block.

Table 3-8 lists the extensions that you can use to monitor and control a first order lag block.

Table 3-8 First Order Lag Extensions

Read-only Boolean		Read-only Integer		Read-only Real	
.ENABLD	enabled	.IOUT	integer output [3]	.OUT	real output [3]
		.ECODE	error code [4]	.DMOUT	output minus delay
		.IID	error controller block [4]		
		.SNUM	error line number [4]		
Read/Write Boolean		**Read/Write Integer**		**Read/Write Real**	
.ENABL	enable [1]			.IN	input
.NRDY	not ready			.INIC	initial input
.REN	request enable [2]			.OUTIC	initial output
.RDIS	request disable [2]			.GAIN	steady-state gain
				.TAU1	time constant one
				.DTIME	dead time

1 This extension is manipulated by the command(s) associated with the block.

2 These booleans are APT flags that are manipulated by the Math Procedures Latch, Clear, and On.

3 For each output, APT creates .IOUT if output value is an integer or .OUT if value is a real number. The outputs are read-only variables if the block is enabled and read/write when the block is disabled.

4 These extensions are not supported for S5 controllers.

3.6 Derivative

Overview

The derivative block computes the rate of change in the input by using the following formula:

$$Rate\ of\ change = \frac{\Delta\ input}{sample\ time}$$

Figure 3-11 shows the formula that you use to define a derivative block. The result of this calculation is placed in the output field of the derivative block.

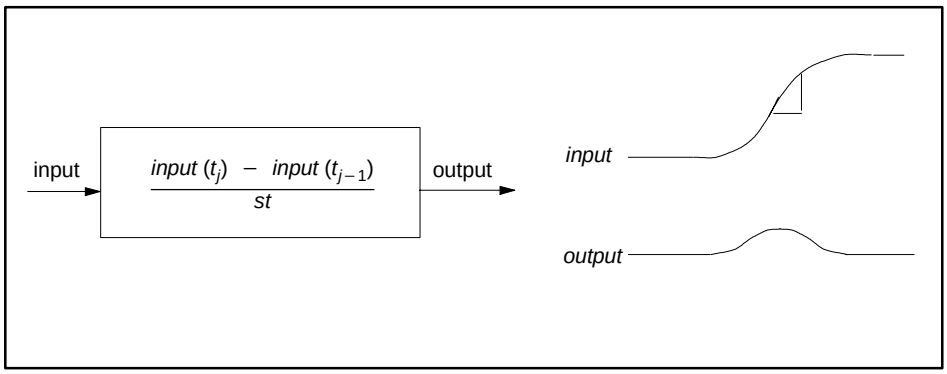


Figure 3-11 Derivative

You can use the derivative block to calculate the rate of change of a variable. You can set a flag if the rate of change exceeds a predetermined limit.

To monitor the rate of change of a variable, follow these steps:

1. Enable the block and wait for it to initialize:

   cfb_name.ENABLD = true;

2. Monitor the rate of change of a certain variable as the output of the derivative block.

3. If the rate of change becomes too great, set a flag that identifies the condition.

Availability

Derivative blocks are available for Series 505 and S5 controllers. They are not supported for the 560/560T controllers.

Block Configuration

Figure 3-12 shows the form that you use to define a derivative block.

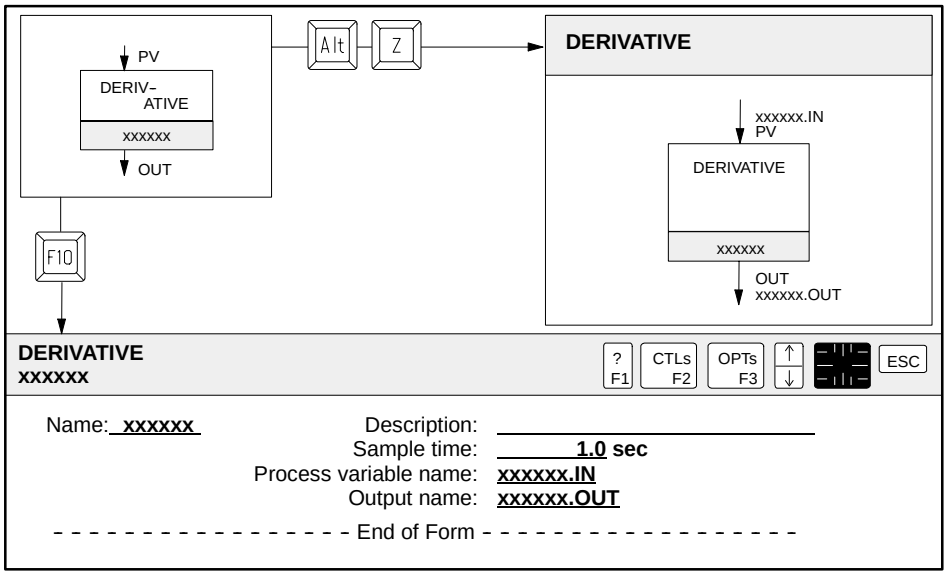


Figure 3-12 Derivative Form

Sample time: sample time in seconds; default is 1; value can be in half-second increments between 0.5 and 4095.5.

Process variable name: name of process variable input (PV); default is *cfb_name*.IN.

Output name: name of output (OUT); default is *cfb_name*.OUT.

Derivative (continued)

Commands and Extensions

Table 3-9 lists the commands that determine the mode of operation for the derivative block.

Table 3-9 Derivative Commands

Command	Description
ENABLE (ENA)	Turns on, or activates, the block.
DISABLE (DIS)	Turns off, or deactivates, the block.

Table 3-10 lists the extensions that you can use to monitor and control a derivative block.

Table 3-10 Derivative Extensions

Read-only Boolean		Read-only Integer		Read-only Real	
.ENABLD	enabled	.IOUT	integer output [3]	.OUT	real output [3]
		.ECODE	error code [4]		
		.IID	error controller block [4]		
		.SNUM	error line number [4]		
Read/Write Boolean		**Read/Write Integer**		**Read/Write Real**	
.ENABL	enable [1]			.IN	input
.NRDY	not ready				
.REN	request enable [2]				
.RDIS	request disable [2]				

1 This extension is manipulated by the command(s) associated with the block.

2 These booleans are APT flags that are manipulated by the Math procedures Latch, Clear, and On.

3 For each output, APT creates .IOUT if output value is an integer or .OUT if value is a real number. The outputs are read-only variables if the block is enabled and read/write when the block is disabled.

4 These extensions are not supported for S5 controllers.

3.7 Dead Time Delay

Overview

The dead time delay block simulates a dynamic process with a specified time delay between a change in the input and a corresponding change in the output. This block operates like the other dynamic blocks except for the update equation:

*table_entry = input * gain*

Figure 3-13 shows the formula that is used for a Dead Time Delay block.

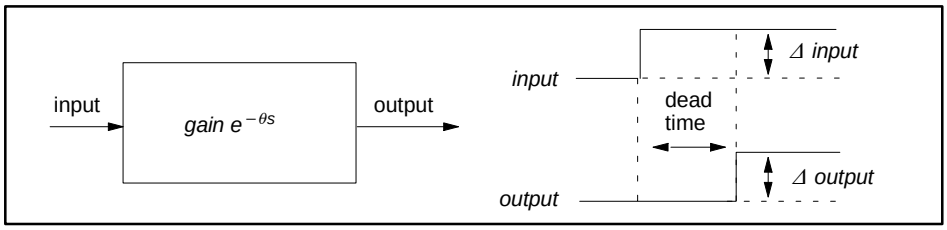


Figure 3-13 Dead Time Delay Form

- Gain specifies the ratio of the change in output to the change in input at a steady state.
- Dead time is the interval between the input change and the start of the response.
- θ is the dead time in seconds.
- *s* is the LaPlace operator.

Availability

Dead time delay blocks are available for Series 505 and S5 controllers. They are not supported for the 560/560T controllers.

Dead Time Delay (continued)

Block Configuration

Figure 3-14 shows the form that you use to define a dead time delay block.

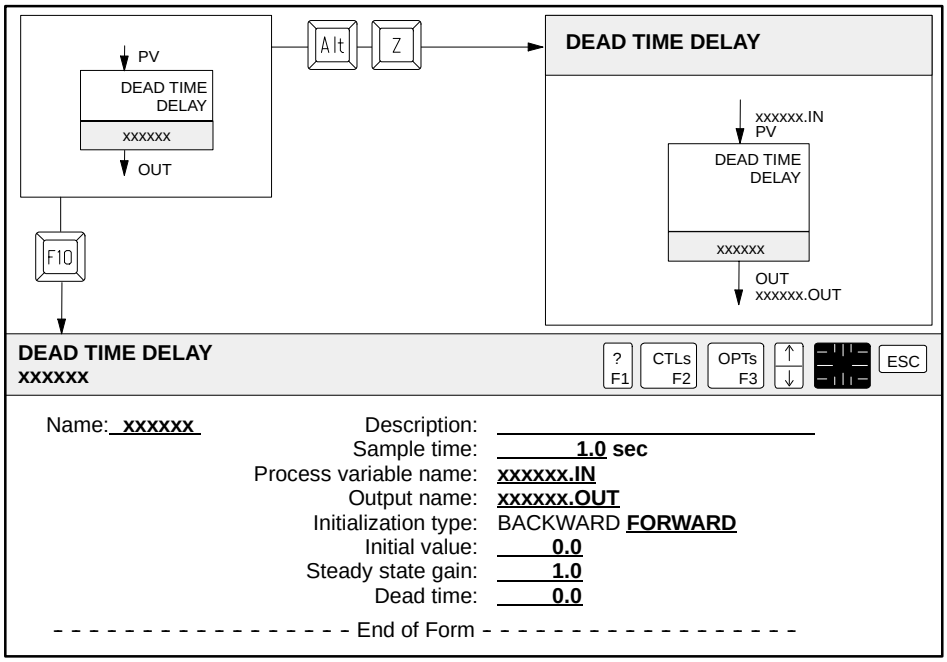


Figure 3-14 Dead Time Delay Form

Sample time: sample time in seconds; default is 1; value can be in half-second increments between 0.5 and 4095.5.

Process variable name: name of process variable input (PV); default is *cfb_name*.IN.

Output name: name of output (OUT); default is *cfb_name*.OUT.

Initialization type: FORWARD or BACKWARD; default is FORWARD.

Initial value: initial value; default is zero; appears only if you select FORWARD initialization.

Steady state gain: steady state gain (the change in output divided by the change in input); default is one.

Dead time: dead time in seconds; default is zero.

Commands and Extensions

Table 3-11 lists the commands that determine the mode of operation for the dead time delay block.

Table 3-11 Dead Time Delay Commands

Command	Description
ENABLE (ENA)	Turns on, or activates, the block.
DISABLE (DIS)	Turns off, or deactivates, the block.

Table 3-12 lists the extensions that you can use to monitor and control a dead time delay block.

Table 3-12 Dead Time Delay Extensions

Read-only Boolean		Read-only Integer		Read-only Real	
.ENABLD	enabled	.IOUT	integer output [3]	.OUT	real output [3]
		.ECODE	error code [4]	.DMOUT	output minus delay
		.IID	error controller block [4]		
		.SNUM	error line number [4]		
Read/Write Boolean		**Read/Write Integer**		**Read/Write Real**	
.ENABL	enable [1]			.IN	input
.NRDY	not ready			.INIC	initial input
.REN	request enable [2]			.OUTIC	initial output
.RDIS	request disable [2]			.GAIN	steady-state gain
				.DTIME	dead time

1 This extension is manipulated by the command(s) associated with the block.

2 These booleans are APT flags that are manipulated by the Math procedures Latch, Clear, and On.

3 For each output, APT creates .IOUT if output value is an integer or .OUT if value is a real number. The outputs are read-only variables if the block is enabled and read/write when the block is disabled.

4 These extensions are not supported for S5 controllers.

3.8 Integrator

Overview

The integrator block produces an output that is proportional to the integral of the input (or the sum of the inputs) with respect to the sample time.

The integrator block uses the following formula to update the delay table:

$$table\ entry = (table\ entry_1) + \frac{1.0}{steady-state\ gain} * (input - initial\ input) * sample\ time$$

The steady-state gain (.TAU1) controls the rate of change of the output. A large gain produces a slow integrator; a small gain produces a fast integrator.

Figure 3-15 shows the formula that is used for an integrator block.

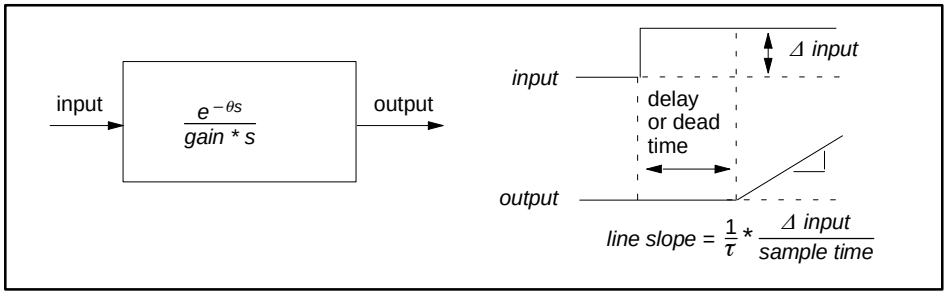


Figure 3-15 Integrator

The integrator block can be used to perform a totalization function if the steady-state gain is set to 1.0. To totalize an input, follow these steps:

1. Enable the block and wait for it to initialize: *cfb_name*.ENABLD = true
2. Open the valve that controls the flow.

   The integrator then changes the output based on the change of the current input from the initial input.

To reset the integrator block, follow these steps:

1. Disable the block and check for: *cfb_name*.ENABLD = false
2. Enable the block and check for: *cfb_name*.ENABLD = true
3. Set *cfb_name*.INIC = 0

Availability

Integrator blocks are available for Series 505 and S5 controllers. They are not supported for the 560/560T controllers.

Block Configuration

Figure 3-16 shows the form that you use to define an integrator block.

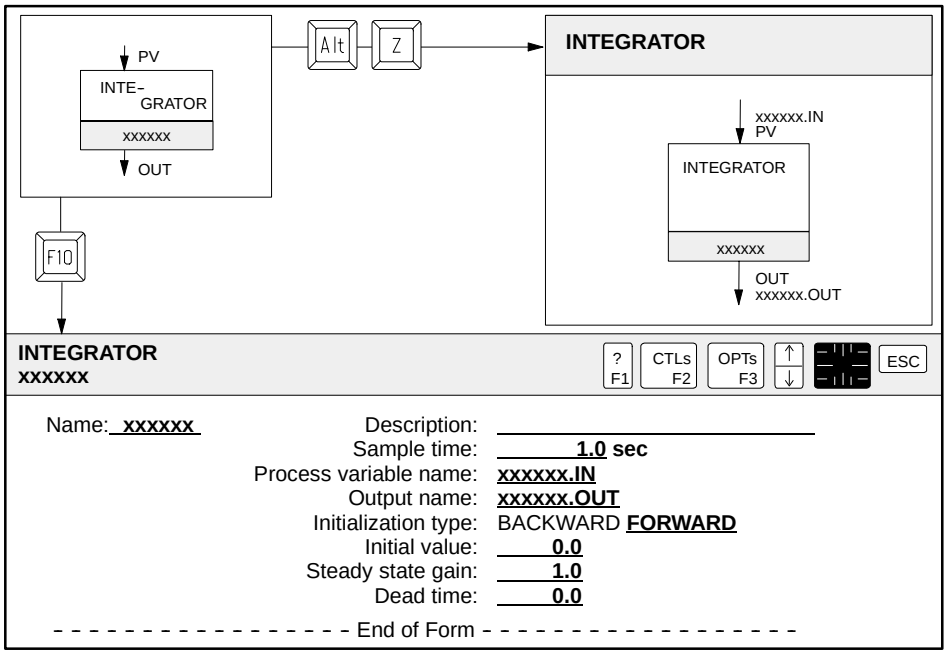


Figure 3-16 Integrator Form

Sample time: sample time in seconds; default is 1; value can be in half-second increments between 0.5 and 4095.5.

Process variable name: name of process variable input (PV); default is *cfb_name*.IN.

Output name: name of output (OUT); default is *cfb_name*.OUT.

Initialization type: FORWARD or BACKWARD; default is FORWARD.

Initial value: initial value; default is zero; appears only if you select FORWARD initialization.

Steady state gain: 1 / (steady state gain) is the slope of the integrator output with a constant input.

Dead time: dead time in seconds; default is zero.

Integrator (continued)

Commands and Extensions

Table 3-13 lists the commands that determine the mode of operation for the integrator block.

Table 3-13 Integrator Commands

Command	Description
ENABLE (ENA)	Turns on, or activates, the block.
DISABLE (DIS)	Turns off, or deactivates, the block.

Table 3-14 lists the extensions that you can use to monitor and control an integrator block.

Table 3-14 Integrator Extensions

Read-only Boolean		Read-only Integer		Read-only Real	
.ENABLD	enabled	.IOUT	integer output [3]	.OUT	real output [3]
		.ECODE	error code [4]	.DMOUT	output minus delay
		.IID	error controller block [4]		
		.SNUM	error line number [4]		
Read/Write Boolean		**Read/Write Integer**		**Read/Write Real**	
.ENABL	enable [1]			.IN	input
.NRDY	not ready			.INIC	initial input
.REN	request enable [2]			.OUTIC	initial output
.RDIS	request disable [2]			.GAIN	steady-state gain
				.TAU1	time constant one
				.DTIME	dead time

1 This extension is manipulated by the command(s) associated with the block.

2 These booleans are APT flags that are manipulated by the Math procedures Latch, Clear, and On.

3 For each output, APT creates .IOUT if output value is an integer or .OUT if value is a real number. The outputs are read-only variables if the block is enabled and read/write when the block is disabled.

4 These extensions are not supported for S5 controllers.

Chapter 4

Advanced Control Blocks

4.1 Understanding Advanced Control Blocks

Overview

The advanced control blocks provide a combination of PID and dynamic control.

- The dead time compensator removes the effect of dead time from the setpoint response of a single input/output process.
- The dual mode block combines PID control with on/off control.
- Two blocks provide feedforward control: feedforward output adjust, and feedforward setpoint adjust.
- The ratio station block keeps a constant ratio between two variables.

Availability

Advanced control blocks are available for Series 505 and S5 controllers. They are not supported for the 560/560T controllers.

PID Functions

The advanced control blocks have the basic functionality of the PID loops. See the appropriate section of Chapter 2, “Standard Control Blocks,” for an explanation of the operation of the PID block. The external anti-reset windup protection can also be used with dynamic blocks. See Chapter 9, “Other Control Blocks,” for information about these control blocks.

Dynamic Functions

Some of the advanced control blocks incorporate the lead and lead-lag functions of the dynamic blocks. Refer to Chapter 3 for details on dynamic control blocks to determine the parameters for first order lag, first order lead-lag, second order lag, and second order lead-lag.

4.2 Dead Time Compensator

Overview

The dead time compensator block removes the effect of dead time from the setpoint response of a single input, single output process.

The dead time compensator includes a PID loop and has the same basic options and operating characteristics as those listed in Chapter 2 on standard control blocks. However, you cannot include any associated math and you must configure the dynamic model.

The dead time compensator block is basically the traditional Smith Predictor control algorithm. You have the option of choosing either a first or second order lag plus a dead time dynamic model.

The model is driven by the output of the PID that also controls the input variable. The output is sent through a model with a dead time block and is subtracted from the process input to generate an error. The error and the model output are added together and sent to the PID loop as the process signal.

Two extension variables are used to compute the process signal: **.MOUT** (model output including the delay) and **.DMOUT** (model output without the delay). The signal is based on the following formula.

cfb_name.LPV = cvb_name.DMOUT + (cfb_name.LPV − cfb_name.MOUT)

This value can also be monitored with the **.SPV** (“seen” process variable) extension.

Availability

Dead time compensator blocks are available for Series 505 and S5 controllers. They are not supported for the 560/560T controllers.

Dead Time Compensator (continued)

Block Configuration

Figure 4-1 shows the dead time compensator CFB graphic.

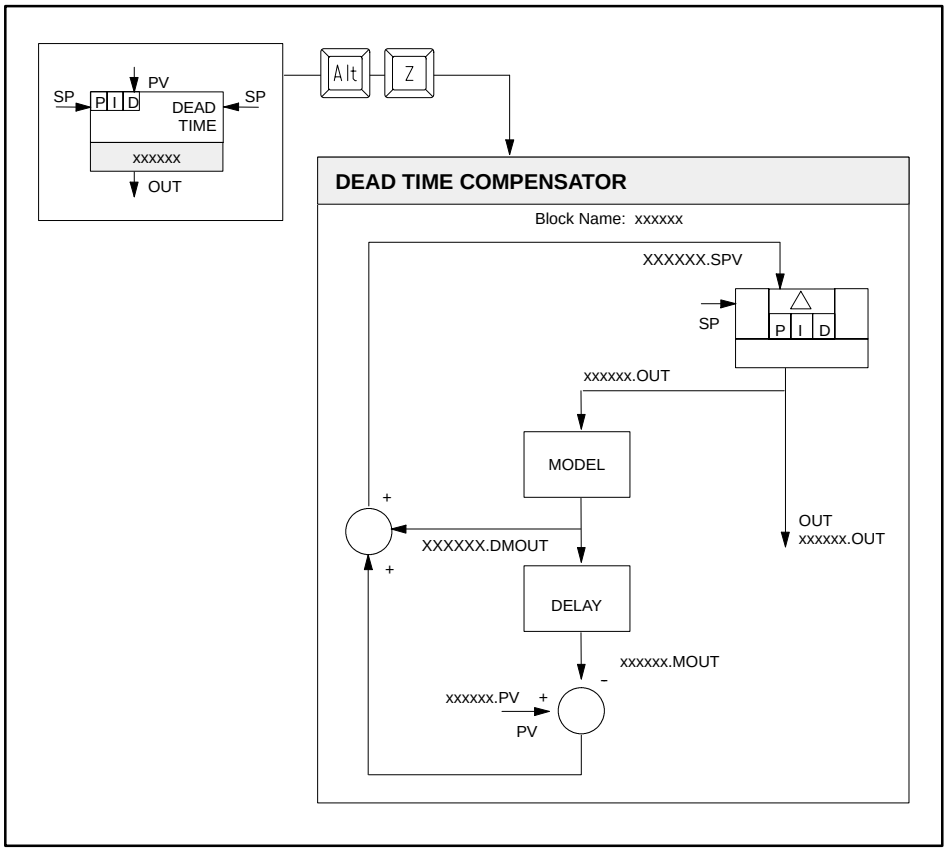


Figure 4-1 Dead Time Compensator CFB

Figure 4-2 shows the form that you use to define the dead time compensator block. The shaded entries in the form depend on the previous selection. The information on the form is the same as the PID block except that you cannot have associated math. See page 2-9 for process variable information, page 2-11 for process setpoint information, page 2-13 for output information, page 2-15 for controller options, and pages 2-21 to 2-23 for PID commands and extensions and for a description of the PID options that apply to the dead time compensator block. Also see Section 3.2, "Second Order Lead Lag," Section 3.3, "Second Order Lag," Section 3.4, "First Order Lead Lag," and Section 3.5, "First Order Lag," for a description of the dynamic options that apply to the dead time compensator block.

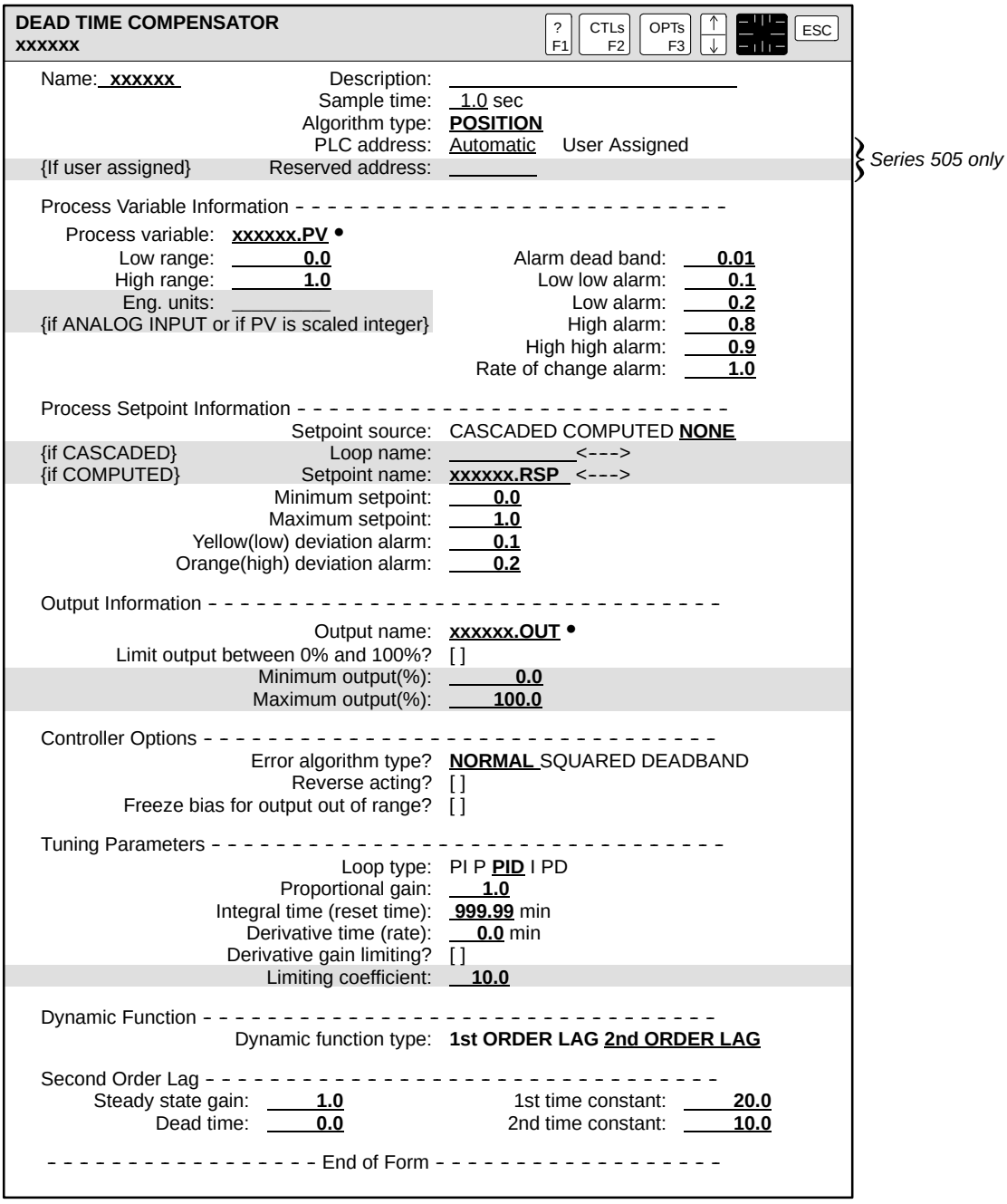


Figure 4-2 Dead Time Compensator Form

Dead Time Compensator (continued)

Commands and Extensions

Table 4-1 lists the commands that determine the mode of operation for the dead time compensator block. These commands operate the same way that they do for the PID block. See page 2-21 for PID commands.

Table 4-1 Dead Time Compensator Commands

Command	Description
AUTO	Places loop in automatic mode and gets setpoint from *cfb_name*.SP.
MANUAL (MAN)	Places loop in manual mode; deactivates PID calculation.
CASCADE (CAS)	Places loop into cascade mode and gets setpoint from remote source.

Table 4-2 lists the extensions that you can use to monitor and control a dead time compensator block. See page 2-23 for PID extensions.

Table 4-2 Dead Time Compensator Extensions

Read-only Boolean		Read-only Integer		Read-only Real	
.IMAN	in manual mode	.IOUT	integer/analog output [2,8]	.OUT	real output [2]
.IAUTO	in auto mode	.ECODE	error code [8]	.ERR	alarm error
.ICASC	in cascade mode	.IID	error controller block [8]	.MOUT	model output with delay
.SERR	error sign (1 = –, 0 = +)	.SNUM	error line number [8]	.DMOUT	model output without delay
.INHHA	PV > high-high alarm	.SMODE	block status	.SPV	“seen” process input
.INHA	PV > high alarm	.VFLAG	status word		
.INLA	PV < low alarm	.IERR	alarm error [3]		
.INLLA	PV < low-low alarm				
.INYDA	error > yellow deviation				
.INODA	error > orange deviation				
.INRCA	rate of change > .RCA				
.INBTA	broken transmitter				
.OVRUN	loop not completing				
Read/Write Boolean		**Read/Write Integer**		**Read/Write Real**	
.NRDY	not ready	.IIN	integer input [4,8]	.IN	real input [4]
.GMAN	go to manual [1]	.IPV	internal PV [4,8]	.LPV	scaled PV input
.GAUTO	go to auto [1]	.ISP	internal SP [8]	.LSP	scaled SP input
.GCASC	go to cascade [1]	.IMN	internal output [8]	.LMN	internal output

Table 4-2 Dead Time Compensator Extensions (continued)

Read/Write Boolean (continued)		Read/Write Integer (continued)		Read/Write Real (continued)	
.RMAN	go to manual [5]	.AWS	anti-reset windup [6]	.HLIM	high limit for output
.RATO	go to auto [5]	.CFL	C-flag low	.LLIM	low limit for output
.RCAS	go cascade [5]	.CFH	C-flag high	.PV	real input [4]
		.IBIAS	integer value of bias [8]	.RSP	setpoint source [7]
				.ST	sample time
				.PVH	PV high range
				.PVL	PV low range
				.KC	proportional gain
				.TI	integral time
				.TD	derivative time
				.HHA	high-high alarm limit
				.HA	high alarm limit
				.LA	low alarm limit
				.LLA	low-low alarm limit
				.ODA	orange deviation limit
				.YDA	yellow deviation limit
				.RCA	rate-of-change limit
				.SP	setpoint
				.SPH	setpoint high limit
				.SPL	setpoint low limit
				.GAIN	steady-state gain
				.TAU1	time constant one
				.TAU2	time constant two
				.DTIME	dead time
				.BIAS	integral of loop error

1 These extensions are manipulated by the command(s) associated with the block.

2 For each output, APT creates **.IOUT** if output value is an integer or **.OUT** if value is a real number. The outputs are read-only variables if block is in auto or cascade and read/write when block is manual.

3 The **.IERR** extension is generated only when a Series 505 Thermocouple or RTD input (8WX version) is configured.

4 For each input, APT creates **.IIN** if input value is an integer or **.IN** if value is a real number. The value of the **.PV** extension is identical to that of the **.IN** extension (**.IIN** is identical to **.IPV**).

5 These booleans are APT flags that are manipulated by the Math procedures Latch, Clear, and On.

6 If a PID block is associated with an Anti-Reset Windup block, the Anti-Reset Windup CFB controls this value.

7 If setpoint source is computed, this **.RSP** or a real variable can be entered as setpoint source. If setpoint source is cascaded, **.RSP** is an alternate name for the source loop **.OUT** extension variable.

8 These extensions are not supported for S5 controllers.

4.3 Dual Mode

Overview

The dual mode block provides a mechanism to start up a process variable in a minimum amount of time and with a minimum amount of overshoot.

The dual mode block combines PID control with on/off control. The PID loop must be in auto or cascade mode before you enable the dual mode block. The PID loop has the same options and operating characteristics as those listed in Chapter 2, "Standard Control Blocks." You cannot include associated math on the output because the dual mode block operates on the output of the PID loop.

To properly control the dual mode block, include these steps in an SFC:

1. Place the PID loop in auto or cascade.
2. After the loop is in the appropriate mode, enable the dual mode block.
3. Change the setpoint to the desired value to move the process variable into orange deviation.

When the dual mode block is enabled, the **.DMODE** extension is set to 1 and retains that value until the PID loop takes control. When the process deviation crosses the orange deviation line, the following steps occur (Figure 4-3):

1. The output is set to 100% until the process re-crosses the orange deviation line.
2. The output is set to 0% for the first time interval that you specify (Delay time 1).
3. The output is set to the preset value for the second time interval that you specify (Delay time 2).
4. The bias is set to the preset value, and control is returned to the PID loop.
5. The dual mode portion is deactivated until it is reset with an ENABLE command.

The orange deviation and the delay times are the critical tuning constants for proper operation of the dual mode block. The block assumes that the process variable starts at a point below the orange deviation limit. Initial setpoint change must be large enough to push the error into orange deviation.

After the loop gets control of the process, you can change the deviation limits by writing to the appropriate extension variables.

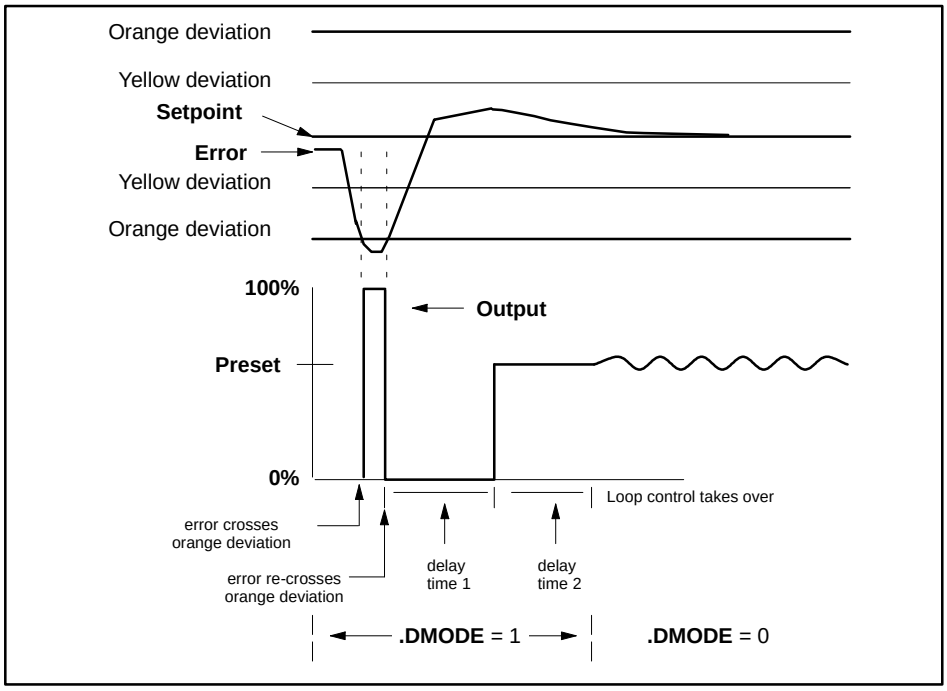


Figure 4-3 Dual Mode Operation

Availability

Dual mode blocks are available for Series 505 and S5 controllers. They are not supported for the 560/560T controllers.

Enabling and Disabling Blocks

The dual mode CFB must be activated with the ENABLE command or with an assignment statement that sets the **.ENABL** extension to true. See Figure 4-4. The dual mode block can also be enabled from an assignment statement wherever you can use the Math language, for example, SFC steps, CFBs, etc.

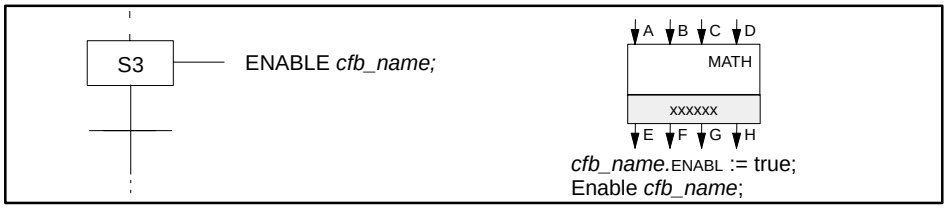


Figure 4-4 Enabling a Dual Mode CFB

Dual Mode (continued)

To deactivate the dual mode CFB, use the DISABLE command with the block name or set the **.ENABL** extension to false.

The dual mode block has a boolean extension (**.NRDY**) that you can use to interlock the execution of the block to some external conditions:

- If you set the **.NRDY** extension to true, the block is forced to the disabled state.
- To return the block to the enabled state you must not only set the **.NRDY** extension to false, but you must also re-enable the block with a command or an assignment statement.

Block Configuration

Figure 4-5 shows the dual mode CFB graphic.

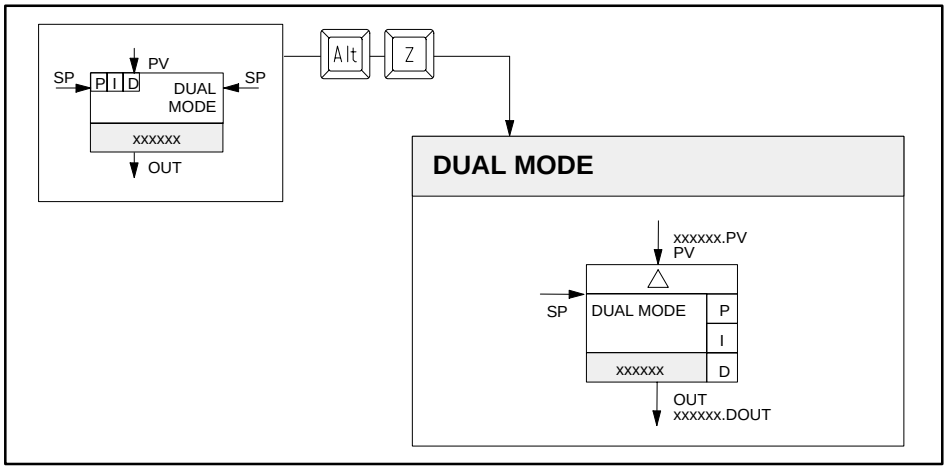


Figure 4-5 Dual Mode Graphic

Figure 4-6 shows the form that you use to define the dual mode block. The shaded entries in the form depend on the previous selection. The information on the form is the same as the PID block except that you cannot have associated math on the output and you must enter the dual mode parameters. See page 2-9 for process variable information, page 2-11 for process setpoint information, page 2-13 for output information, page 2-15 for controller options, and pages 2-21 to 2-23 for PID commands and extensions and for a description of the PID options that apply to the dual mode block.

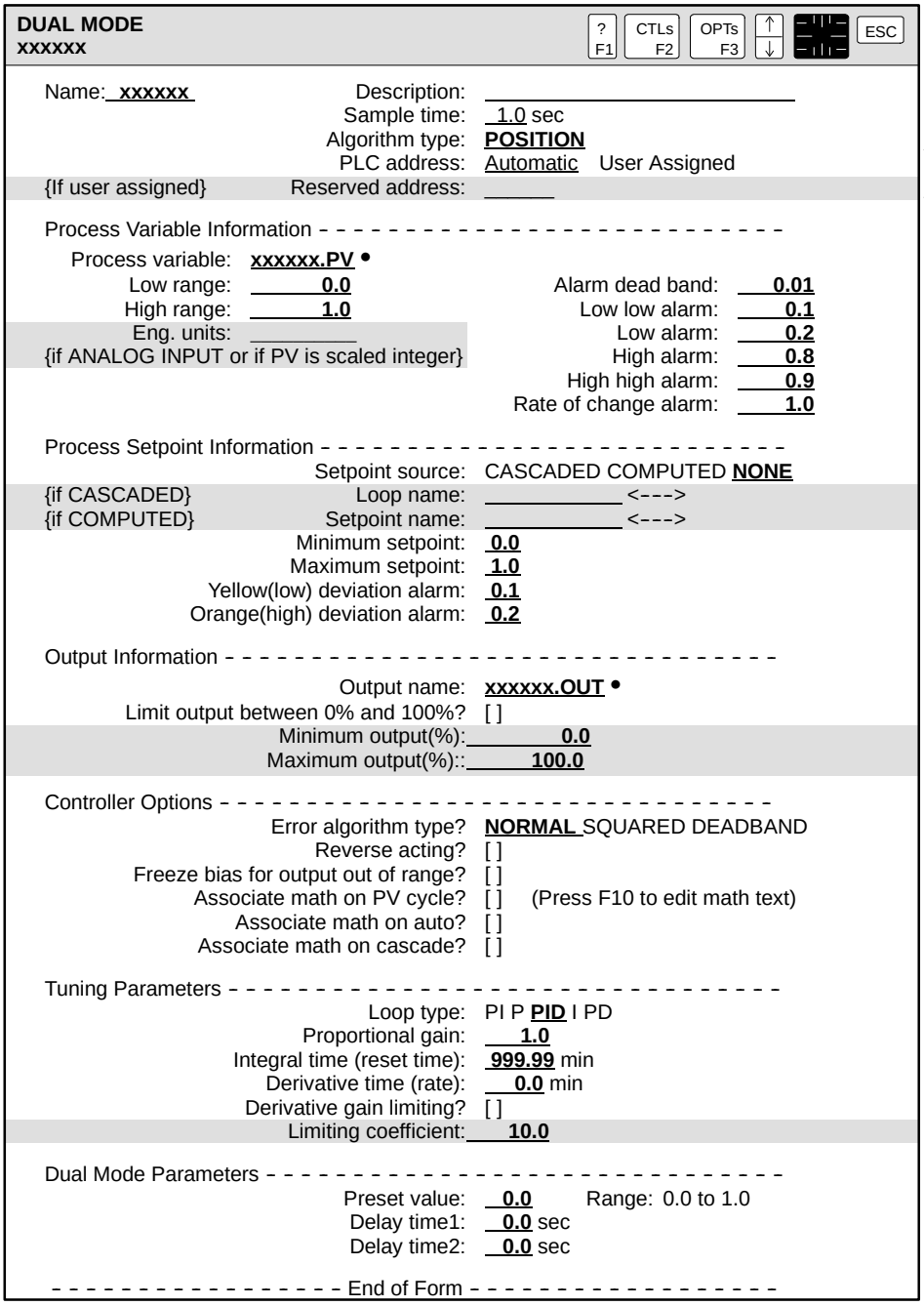


Figure 4-6 Dual Mode Form

Preset value: output value between 0 and 100% (0 – 1). The output will be held at the preset value for the second time interval.

Delay time 1: Delay time, in seconds, for the first time interval. During the first time interval, the output is held at 100%. Default for delay time 1 is 0.0.

Delay time 2: Delay time, in seconds, for the second time interval. During the second time interval, the output is equal to the preset value. The default is 0.0.

Dual Mode (continued)

Commands and Extensions

Table 4-3 lists the commands that determine the mode of operation for the dual mode block. These commands operate the same way that they do for the PID block. See page 2-21 for PID commands.

Table 4-3 Dual Mode Commands

Command	Description
AUTO	Places loop in automatic mode and gets setpoint from *cfb_name*.SP.
MANUAL (MAN)	Places loop in manual mode; deactivates PID calculation.
CASCADE (CAS)	Places loop into cascade mode and gets setpoint from remote source.
ENABLE (ENA)	Turns on, or activates, the Dual Mode function.
DISABLE (DIS)	Turns off, or deactivates, the Dual Mode function.

Table 4-4 lists the extensions for the dual mode block. See page 2-23 for PID extensions.

Table 4-4 Dual Mode Extensions

Read-only Boolean		Read-only Integer		Read-only Real	
.IMAN	in manual mode	.IOUT	integer/analog output [1,3]	.OUT	real output [1]
.IAUTO	in auto mode	.ECODE	error code [3]	.ERR	alarm error
.ICASC	in cascade mode	.IID	error controller block [3]		
.SERR	error sign (1 = –, 0 = +)	.SNUM	error line number [3]		
.INHHA	PV > high-high alarm	.SMODE	block status		
.INHA	PV > high alarm	.VFLAG	status word		
.INLA	PV < low alarm	.IERR	alarm error [2]		
.INLLA	PV < low-low alarm				
.INYDA	error > yellow deviation				
.INODA	error > orange deviation				
.INRCA	rate of change > .RCA				
.INBTA	broken transmitter				
.OVRUN	loop not completing				

1 For each output, APT creates .IOUT if output value is an integer or .OUT if value is a real number. The outputs are read-only variables if the block is in auto or cascade and read/write when the block is manual.

2 The .IERR extension is generated only when a Series 505 Thermocouple or RTD input (8WX version) is configured.

3 These extensions are not supported for S5 controllers.

Table 4-4 Dual Mode Extensions (continued)

Read/Write Boolean		Read/Write Integer		Read/Write Real	
.NRDY	not ready	**.IIN**	integer input [3,6]	**.IN**	real input [6]
.GMAN	go to manual [4]	**.IPV**	internal PV [3,6]	**.LPV**	scaled PV input
.GAUTO	go to auto [4]	**.ISP**	internal SP [3]	**.LSP**	scaled SP input
.GCASC	go to cascade [4]	**.IMN**	internal output [3]	**.LMN**	internal output
.RMAN	go to manual [5]	**.AWS**	anti-reset windup [7]	**.HLIM**	high limit for output
.RATO	go to auto [5]	**.CFL**	C-flag low	**.LLIM**	low limit for output
.RCAS	go cascade [5]	**.CFH**	C-flag high	**.PV**	real input [6]
.REN	request enable [5]	**.DMODE**	dual mode status	**.RSP**	setpoint source [8]
.RDIS	request disable [5]	**.IBIAS**	integer value of bias [3]	**.ST**	sample time
				.PVH	PV high range
				.PVL	PV low range
				.KC	proportional gain
				.TI	integral time
				.TD	derivative time
				.HHA	high-high alarm limit
				.HA	high alarm limit
				.LA	low alarm limit
				.LLA	low-low alarm limit
				.ODA	orange deviation limit
				.YDA	yellow deviation limit
				.RCA	rate-of-change limit
				.SP	setpoint
				.SPH	setpoint high limit
				.SPL	setpoint low limit
				.PRSET	preset (0.0 to 1.0)
				.DLY1	delay time 1, in seconds
				.DLY2	delay time 2, in seconds
				.BIAS	integral of loop error

3 These extensions are not supported for S5 controllers.

4 These extensions are manipulated by the command(s) associated with the block.

5 These booleans are APT flags that are manipulated by the Math procedures Latch, Clear, and On.

6 For each input, APT creates **.IIN** if input value is an integer or **.IN** if value is a real number. The value of the **.PV** extension is identical to that of the **.IN** extension (**.IIN** is identical to **.IPV**).

7 If a PID block is associated with an Anti-Reset Windup block, the Anti-Reset Windup CFB controls this value.

8 If setpoint source is computed, this **.RSP** or a real variable can be entered as setpoint source. If setpoint source is cascaded **.RSP** is an alternate name for the source loop **.OUT** extension variable.

4.4 Feedforward Output Adjust

Overview

The feedforward output adjust block uses two independent control blocks, a PID loop and a dynamic simulation block, to drive a single output. The function of the feedforward output adjust block is to incorporate the features of a dynamic block and a PID block into a single block so that the output is always updated correctly. The feedforward output adjust uses the velocity algorithm where the output spans –100.0 to 100.0.

- On each update, the block adds the contribution of the dynamic block. If the output has exceeded a limit, the block sets the output to the limit value.
- The PID output is then used for further adjustment. If the PID is pushing the block beyond a limit, the output is set to the limit value, and the **.AWS** (anti-reset windup status) extension is set as explained in Chapter 2, "Standard Blocks."
- If the dynamic block is pushing the output beyond a limit while the loop is moving it away, the loop has final control; and the output is moved away from the limit.

The PID block has control of the output which is updated each loop scan or every two seconds, whichever is faster.

NOTE: Because of the dynamic portion of the block, the dynamic sample time should be between 0.5 and 4095.5 seconds in half-second increments. The sample time of the PID loop should match the dynamic sample time.

The ENABLE and DISABLE commands control the dynamic block. (See page 4-9.) The PID block responds to the standard PID commands. You can monitor **.FMODE** (feedforward mode) to determine the status of the block: 1 indicates that the block is executing; 0 indicates that the block is disabled; 3 indicates that the block is initializing.

A typical application of the feedforward output adjust block is to reject the effect of a known, measurable disturbance on the controlled variable. The disturbance variable has no default name in the form that you use to define the block.

Availability

Feedforward output adjust blocks are available for Series 505 and S5 controllers. They are not supported for the 560/560T controllers.

Block Configuration

Figure 4-7 shows the feedforward output adjust CFB graphic.

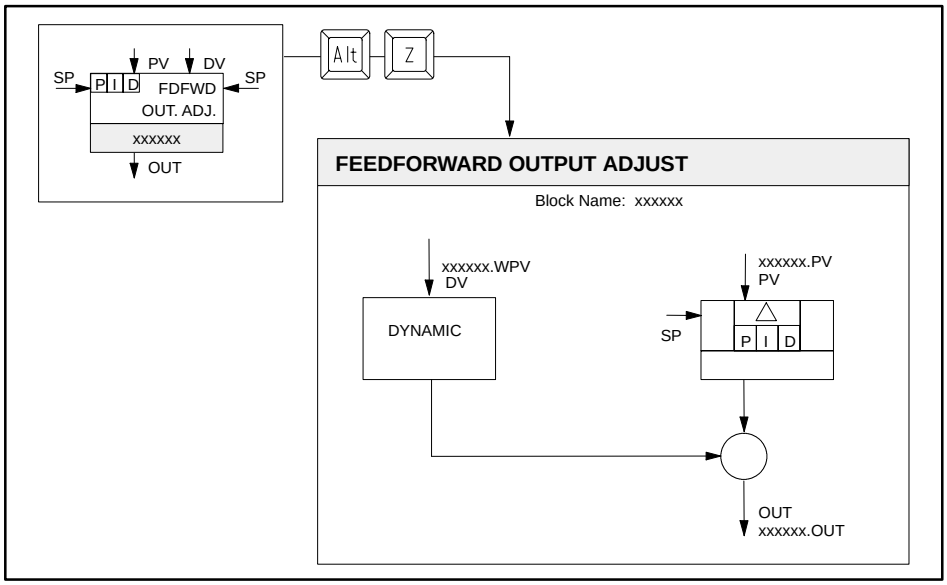


Figure 4-7 Feedforward Output Adjust CFB Graphic

Figure 4-8 shows the form that you use to define the feedforward output adjust block. The shaded entries in the form depend on the previous selection. The information on the form is the same as the PID block except that you cannot have associated math. See page 2-9 for process variable information, page 2-11 for process setpoint information, page 2-13 for output information, page 2-15 for controller options, and pages 2-21 to 2-23 for PID commands and extensions and for a description of the PID options that apply to the feedforward output adjust block. Also see Section 3.2, “Second Order Lead Lag,” Section 3.3, “Second Order Lag,” Section 3.4, “First Order Lead Lag,” and Section 3.5, “First Order Lag,” for a description of the dynamic options that apply to the feedforward output adjust block.

Feedforward Output Adjust (continued)

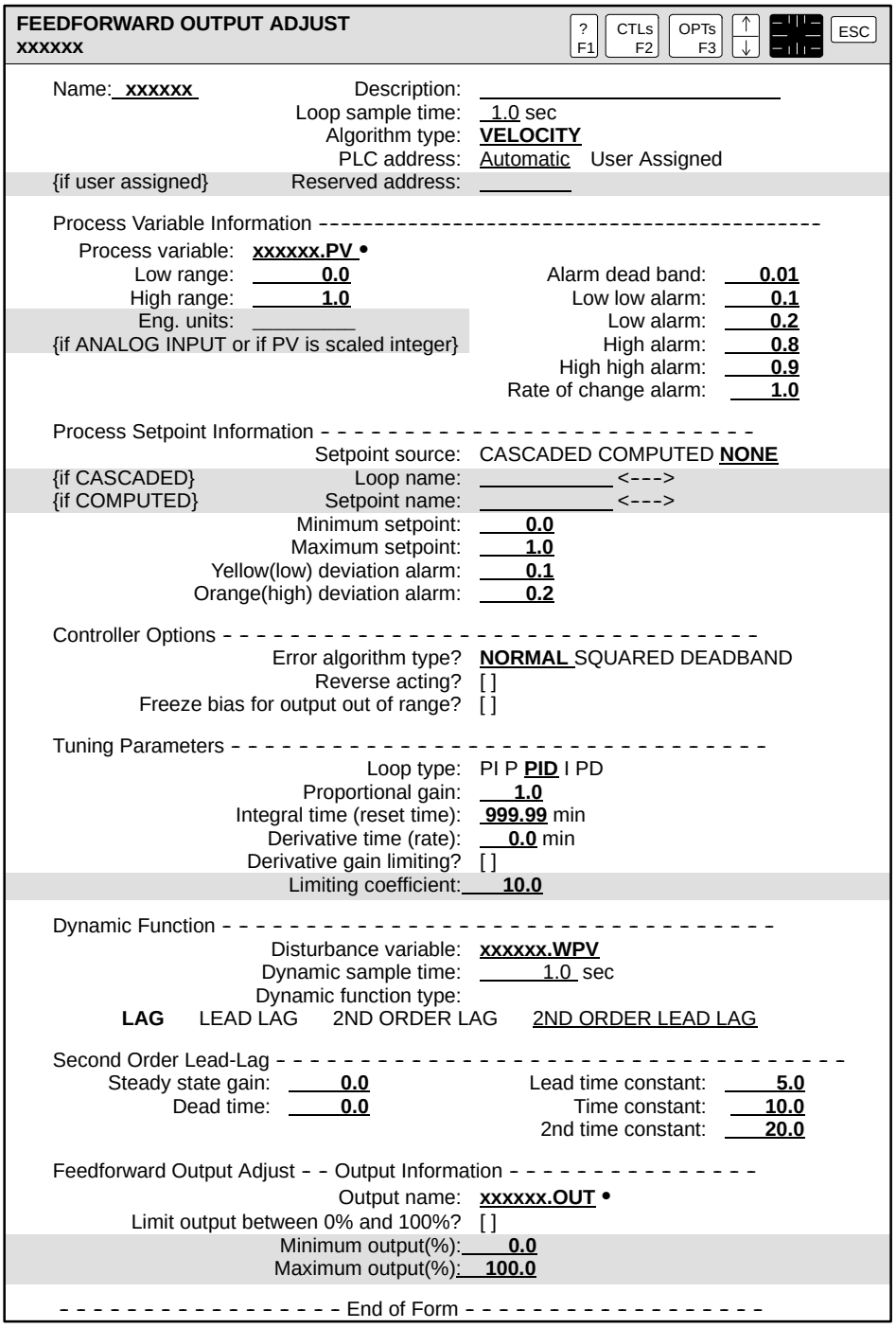


Figure 4-8 Feedforward Output Adjust Form

Commands and Extensions

Table 4-5 lists the commands that determine the mode of operation for the feedforward output adjust block. These commands operate the same way that they do for the PID block. See page 2-21 for PID commands.

Table 4-5 Feedforward Output Adjust Commands

Command	Description
AUTO	Places loop in automatic mode and gets setpoint from *cfb_name*.**SP**.
MANUAL (MAN)	Places loop in manual mode; deactivates PID calculation.
CASCADE (CAS)	Places loop into cascade mode and gets setpoint from remote source.
ENABLE (ENA)	Turns on, or activates, the dynamic function.
DISABLE (DIS)	Turns off, or deactivates, the dynamic function.

Feedforward Output Adjust (continued)

Table 4-6 lists the extensions that you can use to monitor and control and feedforward output adjust block. See page 2-23 for PID extensions.

Table 4-6 Feedforward Output Adjust Extensions

Read-only Boolean		Read-only Integer		Read-only Real	
.ENABL	enabled [3]	.IOUT	integer/analog output [1,8]	.OUT	real output [1]
.IMAN	in manual mode	.ECODE	error code [8]	.ERR	alarm error
.IAUTO	in auto mode	.IID	error controller block [8]	.FFO	dynamic output
.ICASC	in cascade mode	.SNUM	error line number [8]	.SOUT	PID output
.SERR	error sign (1 = –, 0 = +)	.SMODE	block status		
.INHHA	PV > high-high alarm	.VFLAG	status word		
.INHA	PV > high alarm	.IERR	alarm error [2]		
.INLA	PV < low alarm				
.INLLA	PV < low-low alarm				
.INYDA	error > yellow deviation				
.INODA	error > orange deviation				
.INRCA	rate of change > .RCA				
.INBTA	broken transmitter				
.OVRUN	loop not completing				
Read/Write Boolean		**Read/Write Integer**		**Read/Write Real**	
.NRDY	not ready	.IIN	integer input [5]	.IN	real input [5]
.GMAN	go to manual [3]	.IPV	internal PV [5]	.LPV	scaled PV input
.GAUTO	go to auto [3]	.ISP	internal SP	.LSP	scaled SP input
.GCASC	go to cascade [3]	.IMN	internal output	.LMN	internal output
.RMAN	go to manual [4]	.AWS	anti-reset windup [6]	.HLIM	high limit for output
.RATO	go to auto [4]	.CFL	C-flag low	.LLIM	low limit for output
.RCAS	go cascade [4]	.CFH	C-flag high	.PV	real input [5]
.REN	request enable [4]	.FMODE	feedforward block status	.RSP	setpoint source [7]
.RDIS	request disable [4]	.IBIAS	integer value of bias	.ST	sample time

Table 4-6 Feedforward Output Adjust Extensions (continued)

	Read/Write Real (continued)	
	.PVH	PV high range
	.PVL	PV low range
	.KC	proportional gain
	.TI	integral time
	.TD	derivative time
	.HHA	high-high alarm limit
	.HA	high alarm limit
	.LA	low alarm limit
	.LLA	low-low alarm limit
	.ODA	orange deviation limit
	.YDA	yellow deviation limit
	.RCA	rate-of-change limit
	.SP	setpoint
	.SPH	setpoint high limit
	.SPL	setpoint low limit
	.GAIN	steady-state gain
	.TAU1	time constant one
	.TAU2	time constant two
	.TLEAD	lead-time constant
	.DTIME	dead time
	.BIAS	integral of loop error

1	For each output, APT creates **.IOUT** if output value is an integer or **.OUT** if value is a real number. The outputs are read-only variables if the block is in auto or cascade and read/write when the block is manual.
2	The **.IERR** extension is generated only when a Series 505 Thermocouple or RTD input (8WX version) is configured.
3	These extensions are manipulated by the command(s) associated with the block.
4	These booleans are APT flags that are manipulated by the Math procedures Latch, Clear, and On.
5	For each input, APT creates **.IIN** if input value is an integer or **.IN** if value is a real number. The value of the **.PV** extension is identical to that of the **.IN** extension (**.IIN** is identical to **.IPV**).
6	If a PID block is associated with an Anti-Reset Windup block, the Anti-Reset Windup CFB controls this value.
7	If setpoint source is computed, this **.RSP** or a real variable can be entered as setpoint source. If setpoint source is cascaded **.RSP** is an alternate name for the source loop **.OUT** extension variable.
8	These extensions are not supported for S5 controllers.

4.5 Feedforward Setpoint Adjust

Overview

The feedforward setpoint adjust block allows you to insert a dynamic block between the output of a segment of math and the setpoint of a PID loop. The dynamic block is not required; however, without it, this block has the same effect as a PID block with associated math in cascade mode. Associated math is also optional, and you must select it on the form if your application requires a math section.

Because the feedforward setpoint adjust CFB operates on the output of the loop, you cannot associate math on the output.

With the dynamic block, the feedforward setpoint adjust block has the following operation.

- When the PID goes into cascade mode (**.SMODE** = 5), the dynamic block is initialized using the backward initialization algorithm. (The current output becomes the initial simulated output condition, and the initial simulated input condition is computed as the output divided by the gain.)
- After initialization (**.SMODE** = 6), the block executes the math and sends the output through the dynamic block. The output of the dynamic block is used by the PID as the setpoint.

Availability

Feedforward setpoint adjust blocks are available for Series 505 and S5 controllers. They are not supported for the 560/560T controllers.

Block Configuration

Figure 4-9 shows the feedforward setpoint adjust CFB graphic.

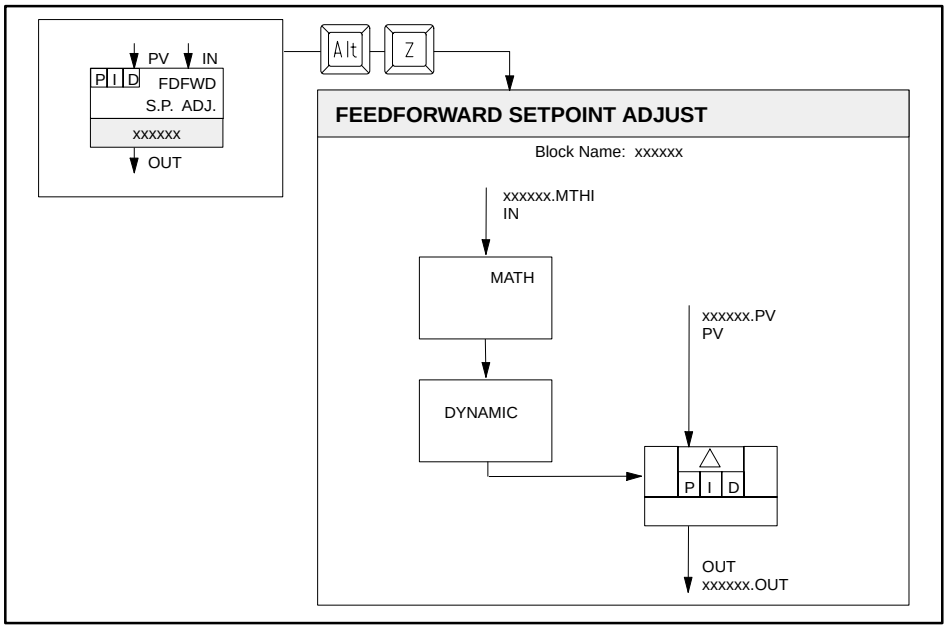
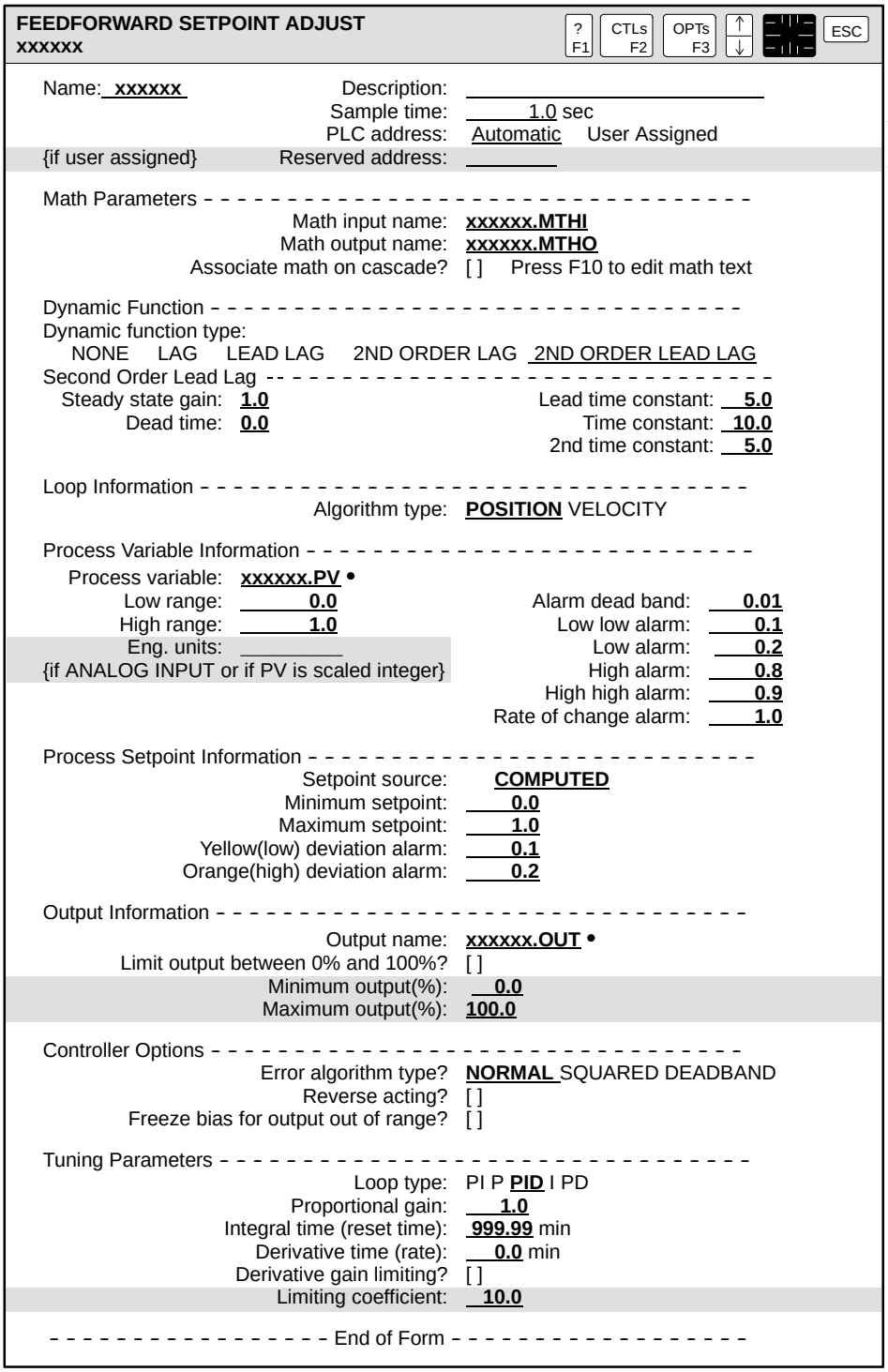


Figure 4-9 Feedforward Setpoint Adjust CFB Graphic

Figure 4-10 shows the form that you use to define the feedforward setpoint adjust block. The shaded entries in the form depend on the previous selection. The information on the form is the same as the PID block except that you cannot have associated math. See page 2-9 for process variable information, page 2-11 for process setpoint information, page 2-13 for output information, page 2-15 for controller options, and pages 2-21 to 2-23 for PID commands and extensions and for a description of the PID options that apply to the feedforward setpoint adjust block. Also see Section 3.2, "Second Order Lead Lag," Section 3.3, "Second Order Lag," Section 3.4, "First Order Lead Lag," and Section 3.5, "First Order Lag," for a description of the dynamic options that apply to the feedforward setpoint adjust block.

Feedforward Setpoint Adjust (continued)



FEEDFORWARD SETPOINT ADJUST
xxxxxx

? F1 | CTLs F2 | OPTs F3 | ↑ ↓ | ESC

Name: **xxxxxx** Description: _____
Sample time: 1.0 sec
PLC address: Automatic User Assigned
{if user assigned} Reserved address: _____

Series 505 only

Math Parameters –
Math input name: **xxxxxx.MTHI**
Math output name: **xxxxxx.MTHO**
Associate math on cascade? [ ] Press F10 to edit math text

Dynamic Function –
Dynamic function type:
NONE LAG LEAD LAG 2ND ORDER LAG 2ND ORDER LEAD LAG
Second Order Lead Lag –
Steady state gain: **1.0** Lead time constant: **5.0**
Dead time: **0.0** Time constant: **10.0**
2nd time constant: **5.0**

Loop Information –
Algorithm type: **POSITION** VELOCITY

Process Variable Information –
Process variable: **xxxxxx.PV** •
Low range: **0.0** Alarm dead band: **0.01**
High range: **1.0** Low low alarm: **0.1**
Eng. units: _____ Low alarm: **0.2**
{if ANALOG INPUT or if PV is scaled integer} High alarm: **0.8**
High high alarm: **0.9**
Rate of change alarm: **1.0**

Process Setpoint Information –
Setpoint source: **COMPUTED**
Minimum setpoint: **0.0**
Maximum setpoint: **1.0**
Yellow(low) deviation alarm: **0.1**
Orange(high) deviation alarm: **0.2**

Output Information –
Output name: **xxxxxx.OUT** •
Limit output between 0% and 100%? [ ]
Minimum output(%): **0.0**
Maximum output(%): **100.0**

Controller Options –
Error algorithm type? **NORMAL** SQUARED DEADBAND
Reverse acting? [ ]
Freeze bias for output out of range? [ ]

Tuning Parameters –
Loop type: PI P **PID** I PD
Proportional gain: **1.0**
Integral time (reset time): **999.99** min
Derivative time (rate): **0.0** min
Derivative gain limiting? [ ]
Limiting coefficient: **10.0**

– – – – – – – – – – – – – – – – – End of Form – – – – – – – – – – – – – – – – –

Figure 4-10 Feedforward Setpoint Adjust Form

Commands and Extensions

Table 4-7 lists the commands that determine the mode of operation for the feedforward setpoint adjust block. These commands operate the same way that they do for the PID block. See page 2-21 for PID commands.

Table 4-7 Feedforward Setpoint Adjust Commands

Command	Description
AUTO	Places loop in automatic mode and gets setpoint from *cfb_name*.SP.
MANUAL (MAN)	Places loop in manual mode; deactivates PID calculation.
CASCADE (CAS)	Places loop into cascade mode.

Table 4-8 lists the extensions that you can use to monitor and control a feedforward setpoint adjust block. See page 2-23 for PID extensions.

Table 4-8 Feedforward Setpoint Adjust Extensions

Read-only Boolean		Read-only Integer		Read-only Real	
.IMAN	in manual mode	.IOUT	integer/analog output [1,8]	.OUT	real output [1]
.IAUTO	in auto mode	.ECODE	error code [8]	.ERR	alarm error
.ICASC	in cascade mode	.IID	error controller block [8]		
.SERR	error sign (1 = –, 0 = +)	.SNUM	error line number [8]		
.INHHA	PV > high-high alarm	.SMODE	block status		
.INHA	PV > high alarm	.VFLAG	status word		
.INLA	PV < low alarm	.IERR	alarm error [2]		
.INLLA	PV < low-low alarm				
.INYDA	error > yellow deviation				
.INODA	error > orange deviation				
.INRCA	rate of change > .RCA				
.INBTA	broken transmitter [8]				
.OVRUN	loop not completing				
Read/Write Boolean		**Read/Write Integer**		**Read/Write Real**	
.NRDY	not ready	.IIN	integer input [5,8]	.IN	real input [5]
.GMAN	go to manual [3]	.IPV	internal PV [5,8]	.MTHI	math block input
.GAUTO	go to auto [3]	.ISP	internal SP [8]	.MTHO	math block output
.GCASC	go to cascade [3]	.IMN	internal output [8]	.LPV	scaled PV input
.RMAN	go to manual [4]	.AWS	anti-reset windup [6]	.LSP	scaled SP input
.RATO	go to auto [4]	.CFL	C-flag low	.LMN	internal output
.RCAS	go cascade [4]	.CFH	C-flag high	.HLIM	high limit for output
		.IBIAS	integer value of bias [8]	.LLIM	low limit for output

Feedforward Setpoint Adjust (continued)

Table 4-8 Feedforward Setpoint Adjust Extensions (continued)

	Read/Write Real (continued)	
	.RSP	setpoint source [7]
	.ST	sample time
	.PV	real input [5]
	.PVH	PV high range
	.PVL	PV low range
	.KC	proportional time
	.TI	integral time
	.TD	derivative time
	.HHA	high-high alarm limit
	.HA	high alarm limit
	.LA	low alarm limit
	.LLA	low-low alarm limit
	.ODA	orange deviation limit
	.YDA	yellow deviation limit
	.RCA	rate-of-change limit
	.SP	setpoint
	.SPH	setpoint high limit
	.SPL	setpoint low limit
	.GAIN	steady-state gain
	.TAU1	time constant one
	.TAU2	time constant two
	.TLEAD	lead-time constant
	.DTIME	dead time
	.BIAS	integral of loop error

1 For each output, APT creates **.IOUT** if output value is an integer or **.OUT** if value is a real number. The outputs are read-only variables if the block is in auto or cascade and read/write when the block is manual.

2 The **.IERR** extension is generated only when a Series 505 Thermocouple or RTD input (8WX version) is configured.

3 These extensions are manipulated by the command(s) associated with the block.

5 For each input, APT creates **.IIN** if input value is an integer or **.IN** if value is a real number. The value of the **.PV** extension is identical to that of the **.IN** extension (**.IIN** is identical to **.IPV**).

6 If a PID block is associated with an Anti-Reset Windup block, the Anti-Reset Windup CFB controls this value.

7 If setpoint source is computed, this **.RSP** or a real variable can be entered as setpoint source. If setpoint source is cascaded **.RSP** is an alternate name for the source loop **.OUT** extension variable.

8 These extensions are not supported for S5 controllers.

4.6 Ratio Station

Overview

The ratio station block is used to keep a constant ratio between two process variables. The setpoint to the block is the ratio that you want to maintain between a controlled variable and an uncontrolled (wild) process variable.

To ensure a bumpless transfer, the algorithm keeps the setpoint and the output unchanged during the transition from manual or automatic to cascade mode. Thereafter, the loop uses the following formula to compute the loop setpoint and bring the process variable to the appropriate value:

$$setpoint = (ratio\ setpoint \times wild\ variable) + offset$$

When you configure the analog input for the ratio station block, do not select the **No Scaling** option for the uncontrolled process variable for a ratio station block. You can, however, select **No Scaling** for the controlled process variable.

Availability

Ratio station blocks are available for Series 505 and S5 controllers. They are not supported for the 560/560T controllers.

Block Configuration

Figure 4-11 shows the graphic for a ratio station CFB.

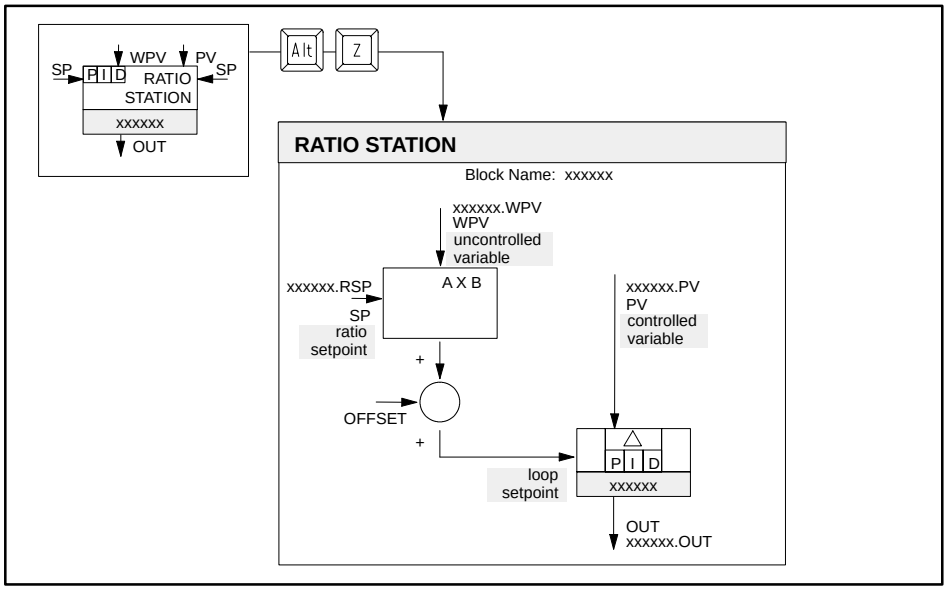


Figure 4-11 Ratio Station Graphic

Figure 4-12 shows the form that you use to define the ratio station block. The shaded entries in the form depend on the previous selection. The information on the form is the same as the PID block except that you cannot have associated math.

Ratio Station (continued)

See page 2-9 for process variable information, page 2-11 for process setpoint information, page 2-13 for output information, page 2-15 for controller options, and pages 2-21 to 2-23 for PID commands and extensions and for a description of the PID options that apply to the the ratio station block.

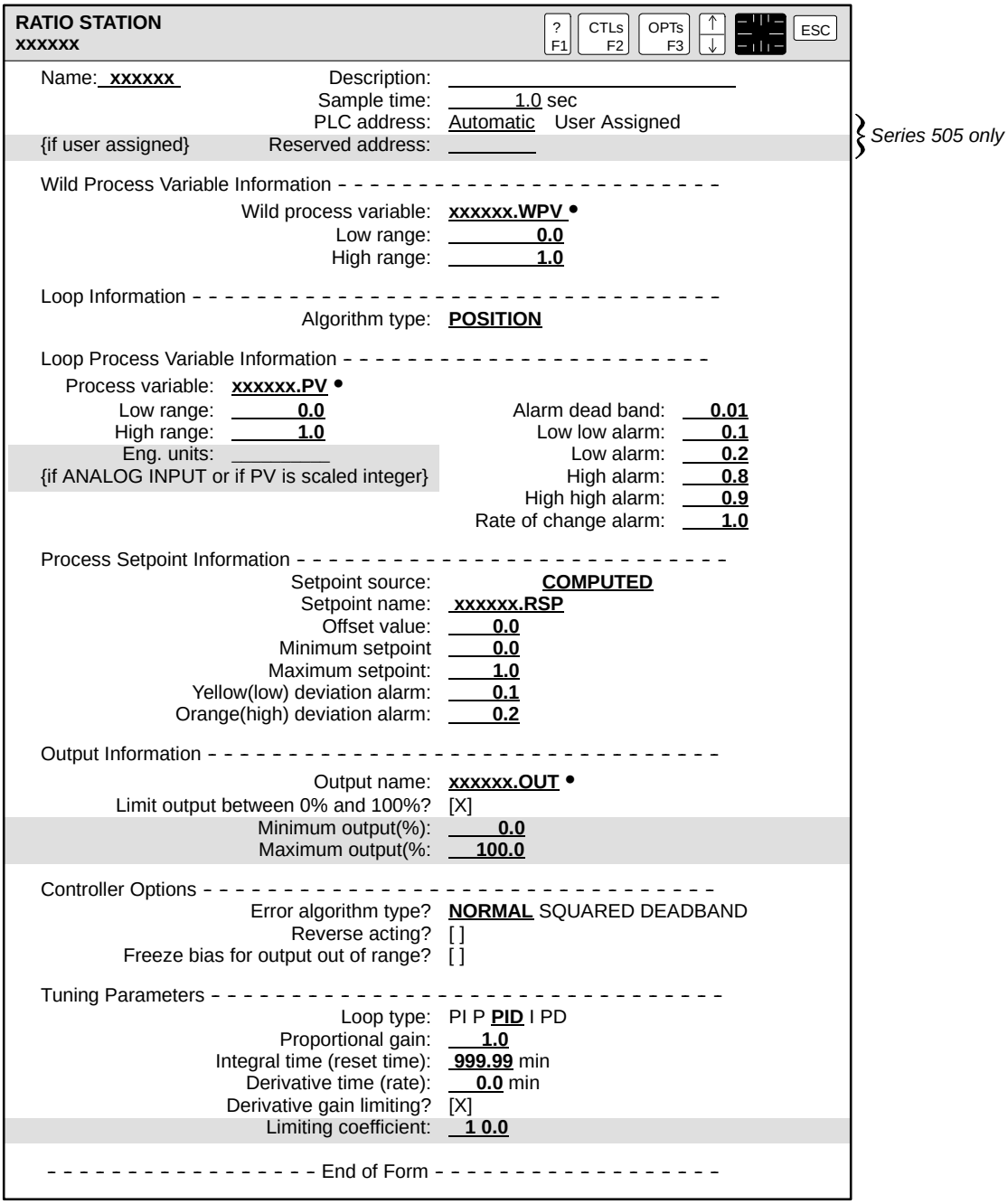


Figure 4-12 Ratio Station Form

Commands and Extensions

Table 4-9 lists the commands that determine the mode of operation for the ratio station block. These commands operate in the same way that they do for the PID block. See page 2-21 for PID commands.

Table 4-9 Ratio Station Commands

Command	Description
AUTO	Places loop in automatic mode and gets setpoint from *cfb_name*.**SP**.
MANUAL (MAN)	Places loop in manual mode; deactivates PID calculation.
CASCADE (CAS)	Places loop into cascade mode and gets setpoint from remote source.

Table 4-10 lists the extensions that you can use to monitor and control and ratio station block. See page 2-23 for PID extensions.

Table 4-10 Ratio Station Extensions

Read-only Boolean		Read-only Integer		Read-only Real	
.IMAN	in manual mode	**.IOUT**	integer/analog output [1,3]	**.OUT**	real output [1]
.IAUTO	in auto mode	**.ECODE**	error code [3]	**.ERR**	alarm error
.ICASC	in cascade mode	**.IID**	error controller block [3]	**.RATIO**	ratio of inputs
.SERR	error sign (1 = –, 0 = +)	**.SNUM**	error line number [3]		
.INHHA	PV > high-high alarm	**.SMODE**	block status		
.INHA	PV > high alarm	**.VFLAG**	status word		
.INLA	PV < low alarm	**.IERR**	alarm error [2]		
.INLLA	PV < low-low alarm				
.INYDA	error > yellow deviation				
.INODA	error > orange deviation				
.INRCA	rate of change > **.RCA**				
.INBTA	broken transmitter [3]				
.OVRUN	loop not completing				

1 For each output, APT creates **.IOUT** if output value is an integer or **.OUT** if value is a real number. The outputs are read-only variables if the block is in auto or cascade and read/write when the block is manual.

2 The **.IERR** extension is generated only when a Series 505 Thermocouple or RTD input (8WX version) is configured.

3 These extensions are not supported for S5 controllers.

Ratio Station (continued)

Table 4-10 Ratio Station Extensions (continued)

Read/Write Boolean		Read/Write Integer		Read/Write Real	
.NRDY	not ready	.IIN	integer input [3,6]	.IN	real input [6]
.GMAN	go to manual [4]	.IWPV	wild process variable [7]	.WPV	wild process variable [7]
.GAUTO	go to auto [4]	.IPV	internal PV [3,6]	.LPV	scaled PV input
.GCASC	go to cascade [4]	.ISP	internal SP [3]	.LSP	scaled SP input
.RMAN	go to manual [5]	.IMN	internal output [3]	.LMN	internal output
.RATO	go to auto [5]	.AWS	anti-reset windup [8]	.HLIM	high limit for output
.RCAS	go cascade [5]	.CFL	C-flag low	.LLIM	low limit for output
		.CFH	C-flag high	.RSP	setpoint source [9]
		.IBIAS	integer value of bias [3]	.ST	sample time
				.PV	real input [6]
				.PVH	PV high range
				.PVL	PV low range
				.KC	proportional gain
				.TI	integral time
				.TD	derivative time
				.HHA	high-high alarm limit
				.HA	high alarm limit
				.LA	low alarm limit
				.LLA	low-low alarm limit
				.ODA	orange deviation limit
				.YDA	yellow deviation limit
				.RCA	rate-of-change limit
				.SP	setpoint
				.SPH	setpoint high limit
				.SPL	setpoint low limit
				.OFFST	ratio offset
				.BIAS	integral of loop error

3 These extensions are not supported for S5 controllers.

4 These extensions are manipulated by the command(s) associated with the block.

5 These booleans are APT flags that are manipulated by the Math procedures Latch, Clear, and On.

6 For each input, APT creates **.IIN** if input value is an integer or **.IN** if value is a real number. The value of the **.PV** extension is identical to that of the **.IN** extension (**.IIN** is identical to **.IPV**).

7 For each input, APT creates **.IWPV** if the input is an integer or **.WPV** if value is a real number.

8 If a PID block is associated with an Anti-Reset Windup block, the Anti-Reset Windup CFB controls this value.

9 If setpoint source is computed, this **.RSP** or a real variable can be entered as setpoint source. If setpoint source is cascaded **.RSP** is an alternate name for the source loop **.OUT** extension variable.

Chapter 5

Limiter Blocks

5.1 Understanding Limiter Blocks

Overview

The APT limiter blocks provide a means to limit the value of an output that is based on an input. Two basic types of limiter blocks are available:

- The output-limiter block checks for an out-of-range input and keeps the output of the block within the high and low limits that you specify.
- The rate-limiter block can be used as a ramp function to change the output at a specified rate until it is equal to the input.

Availability

Limiter blocks are available for Series 505 and S5 controllers. They are not supported for the 560/560T controllers.

Inputs and Outputs

The input to a limiter block can be a real number, an integer, or an analog input.

The output responds to the input. If the output is an analog value, APT assumes that the input is a real number limited to the range of 0.0 to 1.0, and automatically scales the output to the correct format (zero offset, 20% bias, or bipolar). If an input is outside this range, the output is set to the maximum or minimum value of the subtype.

Enabling and Disabling Blocks

All limiter CFBs must be activated with the ENABLE command or with an assignment statement that sets the **.ENABL** extension to true. See Figure 5-1. Limiter blocks can also be enabled from an assignment statement in a math CFB.

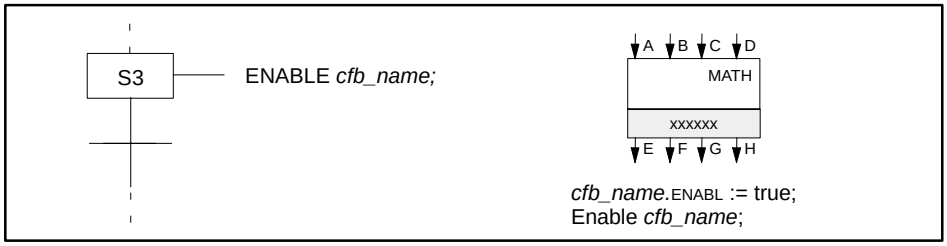


Figure 5-1 Enabling a Limiter CFB

To deactivate the block, use the DISABLE command with the block name or set the **.ENABL** extension to false.

All limiter blocks have a boolean extension (**.NRDY**) that you can use to interlock the execution of the block to some external conditions.

- If you set the **.NRDY** extension to true, the block is forced to the disabled state.
- To return the block to the enabled state you must not only set the **.NRDY** extension to false but also re-enable the block with a command or an assignment statement.

5.2 Output Limiter

Overview

The output limiter block limits the value of a variable to within a specified range.

- If the value of the input is less than the value of the low limit, the output limiter block assigns the value of the low limit to the output.
- If the value of the input is greater than the value of the high limit, the output limiter block assigns the value of the high limit to the output.

Availability

Output limiter blocks are available for Series 505 and S5 controllers. They are not supported for the 560/560T controllers.

Block Configuration

When you define an output limiter CFB with the form that is shown in Figure 5-2, you have the following options.

- Use the default names that appear in the form.
- Add new names: declared variables or dot extensions from other blocks.

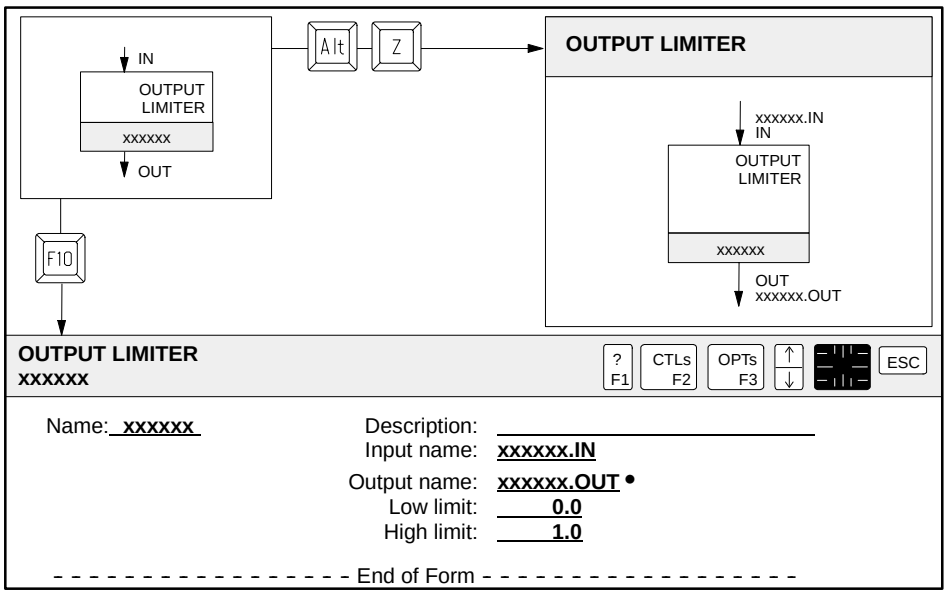


Figure 5-2 Output Limiter Form

Input name (IN): name of analog, real, or integer input; default is *cfb_name*.**IN**.

Output name (OUT): name of analog, real, or integer output that represents limited value of the input; default is *cfb_name*.**OUT**.

Low limit: value of the low limit; default is zero. The value of the output is never less than the value of the low limit.

High limit: value of the high limit; default is one. The value of the output is never greater than the value of the high limit.

Output Limiter (continued)

Commands and Extensions

Table 5-1 lists the commands that determine the mode of operation for the output limiter block.

Table 5-1 Output Limiter Commands

Command	Description
ENABLE (ENA)	Turns on, or activates, the block.
DISABLE (DIS)	Turns off, or deactivates, the block.

Table 5-2 lists the extensions that you can use to monitor and control a output limiter block.

Table 5-2 Output Limiter Extensions

Read-only Boolean		Read-only Integer		Read-only Real	
.ENABLD	enabled	.IOUT	integer output [3]	.OUT	output [3]
		.ECODE	error code [4]		
		.IID	error controller block [4]		
		.SNUM	error line number [4]		
Read/Write Boolean		**Read/Write Integer**		**Read/Write Real**	
.ENABL	enable [1]	.IIN	integer input [5]	.IN	input [5]
.NRDY	not ready			.HLIM	high limit value
.REN	request enable [2]			.LLIM	low limit value
.RDIS	request disable [2]				

1 This extension is manipulated by the command(s) associated with the block.

2 These booleans are APT flags that are manipulated by the Math procedures Latch, Clear, and On.

3 For each output, APT creates **.IOUT** if output value is an integer or **.OUT** if value is a real number. The outputs are read-only variables if the block is enabled and read/write when the block is disabled.

4 These extensions are Series 505 error codes only and do not contain the error codes for S5 controllers.

5 For each input, APT creates **.IIN** if input value is an integer or **.IN** if value is a real number.

5.3 Rate Limiter

Overview

The rate limiter block can be used as a ramp function to restrict the rate of change of an input.

The rate limiter block executes at the sample rate defined in the form. On each execution, the block compares the input to the output. If the values of the input and output are not equal, the rate limiter block lowers or raises the value of the output to decrease the difference between the two. The rate at which the value of the output changes is determined by the rate limit that you specify.

For example, if the initial value of the output is 140.0, the rate limit is 1.0 units/sample, the sample time is 1.0 seconds, and the input is 150.0, the maximum change each second that the block executes is 1. Therefore, it takes 10 executions, or 10 seconds, for the output to reach the steady-state value of the input at 150.0.

If the output is advancing from 140 to 145 in increments of two, the output sequence is

140 142 144 145

To ramp an output variable from the current value to another, change the input value to the desired output. The output then changes to the input at the rate that you specify.

When the two values are equal, the block sets the **.EQ** extensions to true.

NOTE: To ensure a bumpless transfer, assign the value of the input to the output before you enable the block.

Availability

Rate limiter blocks are available for Series 505 and S5 controllers. They are not supported for the 560/560T controllers.

Rate Limiter (continued)

Block Configuration

When you define a rate limiter CFB with the form that is shown in Figure 5-3, you have the following options.

- You can use the default names that appear in the form.
- You can add new names: declared variables or dot extensions from other blocks.

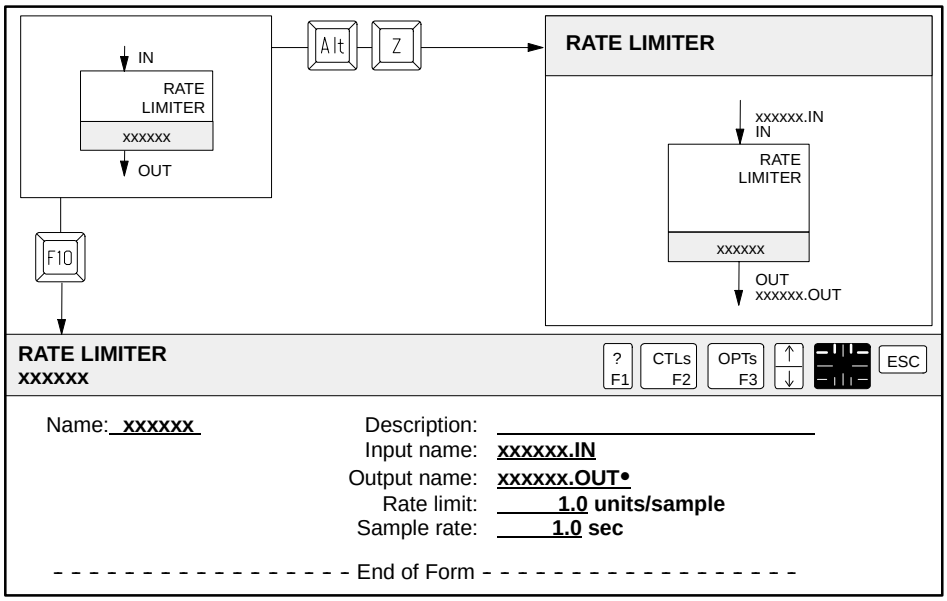


Figure 5-3 Rate Limiter Form

Input name (IN): name of analog, real, or integer input; default is *cfb_name*.**IN**.

Output name (OUT): name of analog, real, or integer output; default is *cfb_name*.**OUT**.

Rate limit: rate of change, in units per sample rate; contains maximum change that is allowed in the output per execution; default is 1.

Sample rate: specifies, in seconds, how often the block is executed; default is 1. The range is 0.5 to 4095.0 seconds.

Commands and Extensions

Table 5-3 lists the commands that determine the mode of operation for the rate limiter block.

Table 5-3 Rate Limiter Commands

Command	Description
ENABLE (ENA)	Turns on, or activates, the block.
DISABLE (DIS)	Turns off, or deactivates, the block.

Table 5-4 lists the extensions that you can use to monitor and control a rate limiter block.

Table 5-4 Rate Limiter Extensions

Read-only Boolean		Read-only Integer		Read-only Real	
.ENABLD	enabled	.IOUT	integer output [3]	.OUT	output [3]
.EQ	input = output	.ECODE	error code [4]		
		.IID	error controller block [4]		
		.SNUM	error line number [4]		
Read/Write Boolean		**Read/Write Integer**		**Read/Write Real**	
.ENABL	enable [1]	.IIN	integer input [5]	.IN	input [5]
.NRDY	not ready			.RLIM	rate limit value
.REN	request enable [2]				
.RDIS	request disable [2]				

1 This extension is manipulated by the command(s) associated with the block.

2 These booleans are APT flags that are manipulated by the Math procedures Latch, Clear, and On.

3 For each output, APT creates **.IOUT** if output value is an integer or **.OUT** if value is a real number. The outputs are read-only variables if the block is enabled and read/write when the block is disabled.

4 These extensions do not contain error codes for S5 controllers. (Series 505-only error codes)

5 For each input, APT creates **.IIN** if input value is an integer or **.IN** if value is a real number.

Chapter 6

Selector Blocks

6.1 Understanding Selector Blocks

Overview

The APT selector blocks provide a means to select one signal from a set of up to four signals. Four basic types of selector blocks are available:

- High, low, and inswitch selectors allow you to choose from four inputs and send the selected signal to an output.
- Median and average selectors allow you to choose from three inputs and send the median or average value to an output.
- The outswitch selector allows you to send an input signal to one of four outputs.
- The threshold selector allows you to determine whether or not the input value has gone above or below a limit. If the value is outside the limit, a boolean variable or discrete output is set to true or 1.

Availability

Selector blocks are available for Series 505 and S5 controllers. They are not supported for the 560/560T controllers.

Inputs and Outputs

The inputs to selector blocks can be any combination of real numbers, integers, analog inputs or empty fields. The output responds as follows:

- If the inputs are integers, APT sends the appropriate value to the output.
- If any inputs are a combination of real and integer values and the output is an analog value, the selected input is sent directly to the output with no scaling.
- If any input is an empty field, APT creates an extension variable that is a real number between 0.0 and 1.0.
- For the high, low, inswitch, and outswitch selector blocks, if the inputs are all real numbers and the output is an analog value, APT assumes that the inputs are limited to the range of 0.0 to 1.0 and automatically scales the output to the correct format (zero offset, 20% offset, or bipolar). If an input is outside this range, the output is set to the maximum or minimum value of the subtype.
- The average selector and median selector blocks provide no scaling on analog outputs. They send out a raw integer.
- The threshold selector block sends out only digital output.

Enabling and Disabling Blocks

All selector CFBs must be activated with the ENABLE command or with an assignment statement that sets the **.ENABL** extension to true. See Figure 6-1. Selector blocks also can be enabled from an assignment statement in a math CFB.

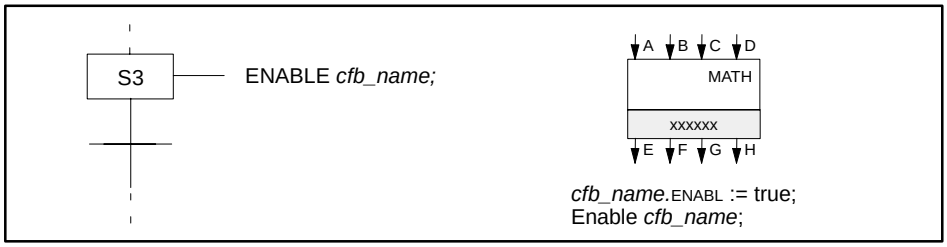


Figure 6-1 Enabling a Selector CFB

To deactivate the block, use the DISABLE command with the block name or set the **.ENABL** extension to false.

All selector blocks have a boolean extension (**.NRDY**) that you can use to interlock the execution of the block to some external conditions.

- If you set the **.NRDY** extension to true, the block is forced to the disabled state.
- To return the block to the enabled state you must not only set the **.NRDY** extension to false, but you must also re-enable the block with a command or an assignment statement.

6.2 Average Selector

Overview

The average selector block computes the numerical average of three inputs.

Before the block performs its calculation, it adds and subtracts from the median of the three inputs the value that you specify as the maximum deviation. If an input is outside of this range, that input is not used in the calculation (the average is computed from the remaining values).

For example, assume three inputs as shown in Figure 6-2: 155, 150, and 160 with a maximum deviation of 2. The median value is 155; the other two values are outside the range of 155 +/- 2; therefore, only one value is used and the output is 155. If, however, the inputs are 155, 162, and 160, the output would be 161: the average of 160 and 162.

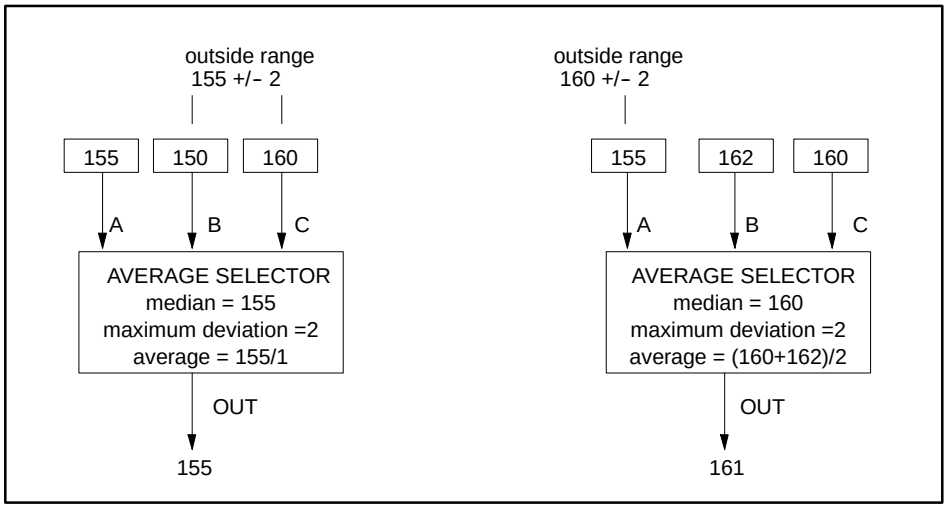


Figure 6-2 Selecting an Average

Availability

Average selector blocks are available for Series 505 and S5 controllers. They are not supported for the 560/560T controllers.

Average Selector (continued)

Block Configuration

When you define an average selector CFB with the form that is shown in Figure 6-3, you have these options:

- You can use the default names that appear in the form. The default extensions must correspond to the correct slot; i.e., **.IN1** or **.IIN1** must be input A, etc.
- You can add new names: declared variables or dot extensions from other blocks.

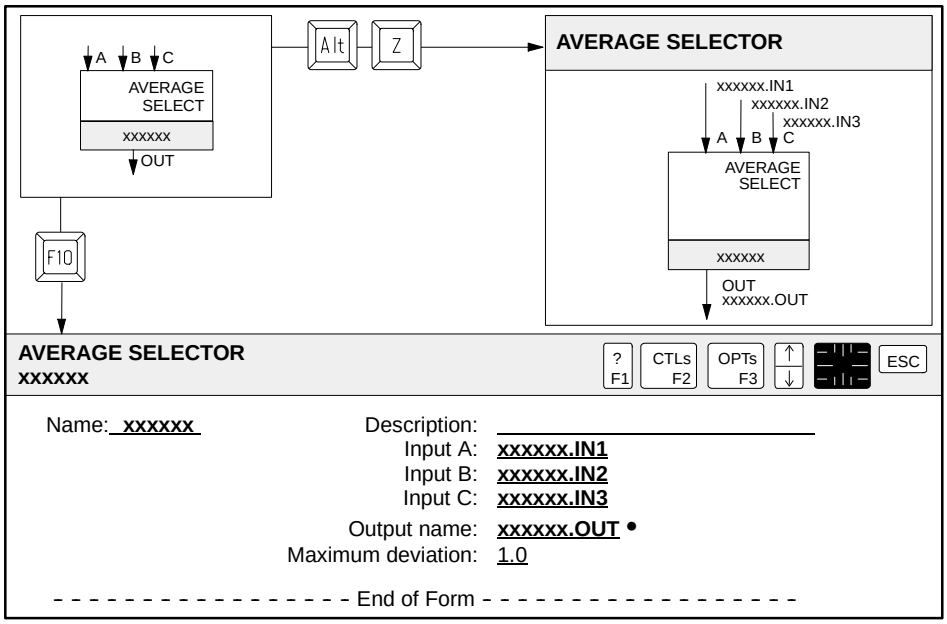


Figure 6-3 Average Selector Form

Input A: name of analog, real, or integer that is first input (A); default is *cfb_name*.**IN1**.

Input B: name of analog, real, or integer that is second input (B); default is *cfb_name*.**IN2**.

Input C: name of analog, real, or integer that is third input (C); default is *cfb_name*.**IN3**.

Output name: name of analog, real, or integer output (OUT) that represents average of values of inputs that are within maximum deviation of median input; default is *cfb_name*.**OUT**.

Maximum deviation: value of maximum deviation. If the absolute value of the difference between the median input and any other input is greater than the value of the maximum deviation, that input is not used in the average calculation.

Commands and Extensions

Table 6-1 lists the commands that determine the mode of operation for the average selector block.

Table 6-1 Average Selector Commands

Command	Description
ENABLE (ENA)	Turns on, or activates, the block.
DISABLE (DIS)	Turns off, or deactivates, the block.

Table 6-2 lists the extensions that you can use to monitor and control a average selector block.

Table 6-2 Average Selector Extensions

Read-only Boolean		Read-only Integer		Read-only Real	
.ENABLD	enabled	.IOUT	integer output [3]	.OUT	real output [3]
.IN1OK	input A within deviation range	.POS	position of median		
.IN2OK	input B within deviation range	.ECODE	error code [4]		
.IN3OK	input C within deviation range	.IID	error controller block [4]		
		.SNUM	error line number [4]		
		.STAT	# of inputs used		
		.ISTATUS	maps boolean status to an integer [5]		
Read/Write Boolean		**Read/Write Integer**		**Read/Write Real**	
.ENABL	enable [1]	.IIN1	integer input A [6]	.IN1	real input A [5]
.NRDY	not ready	.IIN2	integer input B [6]	.IN2	real input B [5]
.REN	request enable [2]	.IIN3	integer input C [6]	.IN3	real input C [5]
.RDIS	request disable [2]			.TVAL	maximum deviation

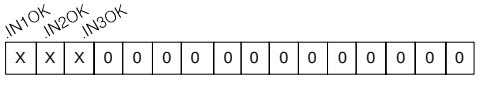
1 This extension is manipulated by the command(s) associated with the block.

2 These booleans are APT flags that are manipulated by the Math procedures Latch, Clear, and On.

3 For each output, APT creates **.IOUT** if output value is an integer or **.OUT** if value is a real number. The outputs are read-only variables if the block is enabled and read/write when the block is disabled.

4 These extensions do not contain error codes for S5 controllers. (Series 505-only error codes)

5 The extension **.IN1OK** maps to bit 1 of a 16 bit integer, **.IN2OK** maps to bit 2, **.IN3OK** maps to bit 3, as shown:



For example, when **.ISTATUS** has a decimal value of 8192, **.IN1OK** and **.IN2OK** = 0, and **.IN3OK** =1.

6 For each input, APT creates **.IIN** if input value is an integer or **.IN** if value is a real number.

6.3 High Selector

Overview

The high selector block selects the highest of up to four enabled inputs and moves it to the output.

The block automatically enables only those inputs for which you either enter names or draw graphic connections. Inputs that are neither named nor graphically connected are automatically disabled. You can also enable an input by setting the corresponding **.SW** extension to 1, or you can disable it by setting the **.SW** extension to 0.

Availability

High selector blocks are available for Series 505 and S5 controllers. They are not supported for the 560/560T controllers.

Block Configuration

When you define a high selector CFB with the form shown in Figure 6-4, you have the following options:

- You can use the default names that appear in the form. The default extensions must correspond to the correct slot; i.e., **.IN1** or **.IIN1** must be input A, etc.
- You can add new names: declared variables or dot extensions from other blocks.
- You can leave fields blank.

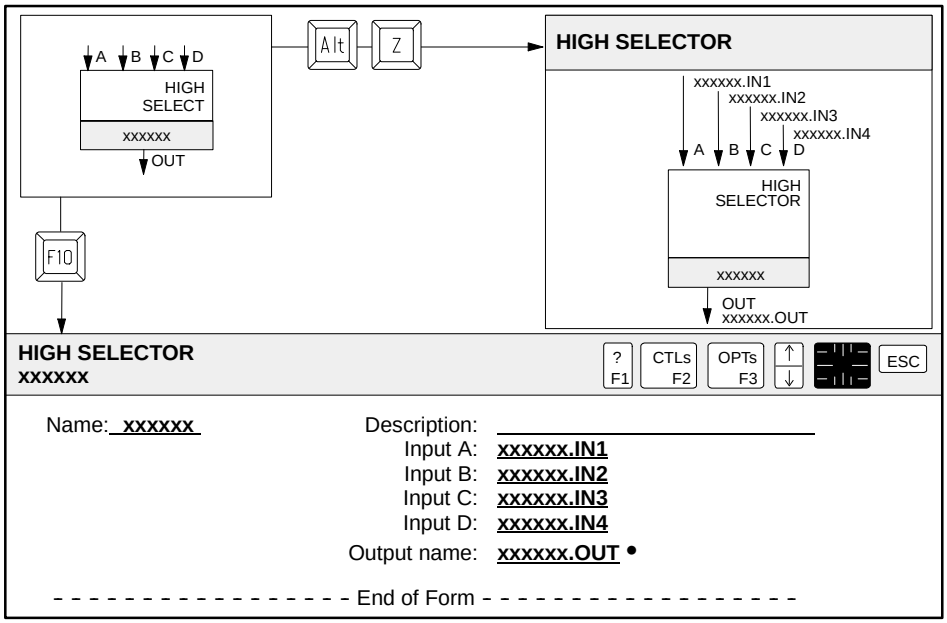


Figure 6-4 High Selector Form

Input A: name of analog, real, or integer that is first input (A); default is *cfb_name*.**IN1**; field can be empty.

Input B: name of analog, real, or integer that is second input (B); default is *cfb_name*.**IN2**; field can be empty.

Input C: name of analog, real, or integer that is third input (C); default is *cfb_name*.**IN3**; field can be empty.

Input D: name of analog, real, or integer that is fourth input (D); default is *cfb_name*.**IN4**; field can be empty.

Output name: name of analog, real, or integer output (OUT) that represents value of highest of enabled inputs; default is *cfb_name*.**OUT**.

High Selector (continued)

Commands and Extensions

Table 6-3 lists the commands that determine the mode of operation for the high selector block.

Table 6-3 High Selector Commands

Command	Description
ENABLE (ENA)	Turns on, or activates, the block.
DISABLE (DIS)	Turns off, or deactivates, the block.

Table 6-4 lists the extensions that you can use to monitor and control a high selector block.

Table 6-4 High Selector Extensions

Read-only Boolean		Read-only Integer		Read-only Real	
.ENABLD	enabled	.IOUT	integer output [3]	.OUT	real output [3]
		.POS	current high input		
		.ECODE	error code [4]		
		.IID	error controller block [4]		
		.SNUM	error line number [4]		
Read/Write Boolean		**Read/Write Integer**		**Read/Write Real**	
.ENABL	enable [1]	.IIN1	integer input A [5]	.IN1	real input A [5]
.NRDY	not ready	.IIN2	integer input B [5]	.IN2	real input B [5]
.REN	request enable [2]	.IIN3	integer input C [5]	.IN3	real input C [5]
.RDIS	request disable [2]	.IIN4	integer input D [5]	.IN4	real input D [5]
		.SW1	input A enable		
		.SW2	input B enable		
		.SW3	input C enable		
		.SW4	input D enable		

1 This extension is manipulated by the command(s) associated with the block.

2 These booleans are APT flags that are manipulated by the Math procedures Latch, Clear, and On.

3 For each output, APT creates **.IOUT** if output value is an integer or analog or **.OUT** if value is a real number. The outputs are read-only variables if the block is enabled and read/write when the block is disabled.

4 These extensions do not contain error codes for S5 controllers. (Series 505-only error codes)

5 For each input, APT creates **.IIN** if input value is an integer or **.IN** if value is a real number.

6.4 Inswitch Selector

Overview

The inswitch selector block allows you to select from among four inputs.

You select the input by setting the value of the **.POS** (position) extension. You can set the value of the **.POS** extension either directly or by using the SWITCH command in an SFC step.

The value of **.POS** must be 1, 2, 3, or 4; if you use SWITCH, this corresponds to settings A, B, C, or D.

For example, both of these instructions produce the same result, namely, they send the input signal C to the output.

```
block_name .POS := 3;
SWITCH block_name C;
```

Availability

Inswitch selector blocks are available for Series 505 and S5 controllers. They are not supported for the 560/560T controllers.

Inswitch Selector (continued)

Block Configuration

When you define an inswitch selector CFB with the form in Figure 6-5, you have the following options:

- You can use the default names that appear in the form. The default extensions must correspond to the correct slot; i.e., **.IN1** or **.IIN1** must be input A, etc.
- You can add new names: declared variables or dot extensions from other blocks.
- You can leave fields blank.

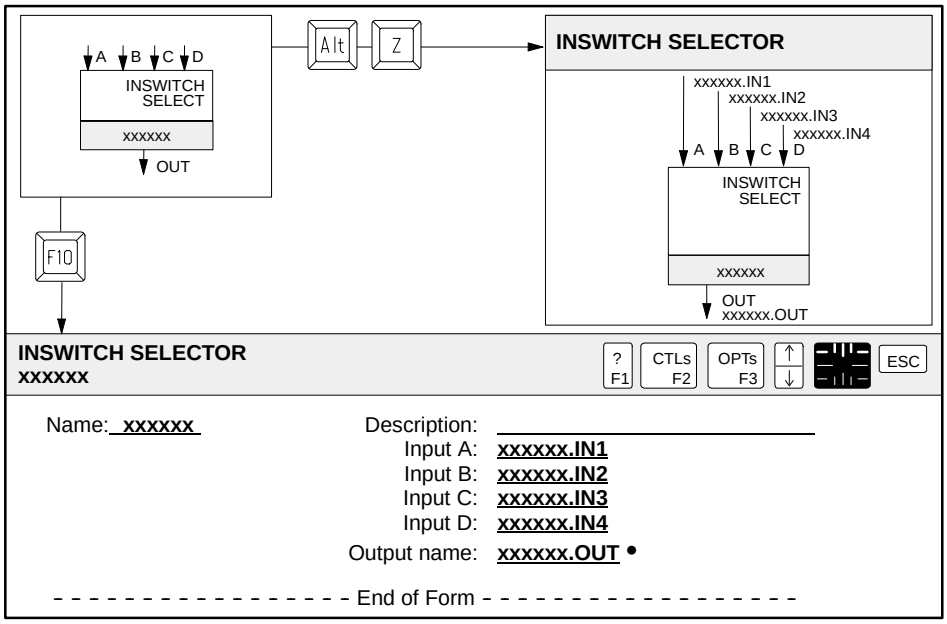


Figure 6-5 Inswitch Selector Form

Input A: name of analog, real, or integer that is first input (A); default is *cfb_name*.**IN1**; field can be empty.

Input B: name of analog, real, or integer that is second input (B); default is *cfb_name*.**IN2**; field can be empty.

Input C: name of analog, real, or integer that is third input (C); default is *cfb_name*.**IN3**; field can be empty.

Input D: name of analog, real, or integer that is fourth input (D); default is *cfb_name*.**IN4**; field can be empty.

Output name: name of analog, real, or integer output (OUT) that represents value of selected input; default is *cfb_name*.**OUT**.

Commands and Extensions

Table 6-5 lists the commands that determine the mode of operation for the inswitch selector block.

Table 6-5 Inswitch Selector Commands

Command	Description
ENABLE (ENA)	Turns on, or activates, the block.
DISABLE (DIS)	Turns off, or deactivates, the block.
SWITCH	Selects input A, B, C, or D and sends to output; for example **SWITCH block_name A**; sends input A to output.

Table 6-6 lists the extensions that you can use to monitor and control a inswitch selector block.

Table 6-6 Inswitch Selector Extensions

Read-only Boolean		Read-only Integer		Read-only Real	
.ENABLD	enabled	.IOUT	integer out [3]	.OUT	real output [3]
		.ECODE	error code [4]		
		.IID	error controller block [4]		
		.SNUM	error line number [4]		
Read/Write Boolean		**Read/Write Integer**		**Read/Write Real**	
.ENABL	enable [1]	.IIN1	integer input A [5]	.IN1	real input A [5]
.NRDY	not ready	.IIN2	integer input B [5]	.IN2	real input B [5]
.REN	request enable [2]	.IIN3	integer input C [5]	.IN3	real input C [5]
.RDIS	request disable [2]	.IIN4	integer input D [5]	.IN4	real input D [5]
		.POS	current input position		

1 This extension is manipulated by the command(s) associated with the block.

2 These booleans are APT flags that are manipulated by the Math procedures Latch, Clear, and On.

3 For each output, APT creates **.IOUT** if output value is an integer or analog, or **.OUT** if value is a real number.
The outputs are read-only variables if the block is enabled and read/write when the block is disabled.

4 These extensions do not contain error codes for S5 controllers. (Series 505-only error codes)

5 For each input, APT creates **.IIN** if input value is an integer or **.IN** if value is a real number.

6.5 Low Selector

Overview

The low selector block selects the lowest of up to four enabled inputs and moves it to the output.

The block automatically enables only those inputs for which you either enter names or draw graphic connections. Inputs that are neither named nor graphically connected are automatically disabled. You can also enable an input by setting the corresponding **.SW** extension to 1, or you can disable it by setting the **.SW** extension to 0.

Availability

Low selector blocks are available for Series 505 and S5 controllers. They are not supported for the 560/560T controllers.

Block Configuration

When you define a low selector CFB with the form that is shown in Figure 6-6, you have the following options.

- You can use the default names that appear in the form. The default extensions must correspond to the correct slot; i.e., **.IN1** or **.IIN1** must be input A, etc.
- You can add new names: declared variables or dot extensions from other blocks.
- You can leave fields blank.

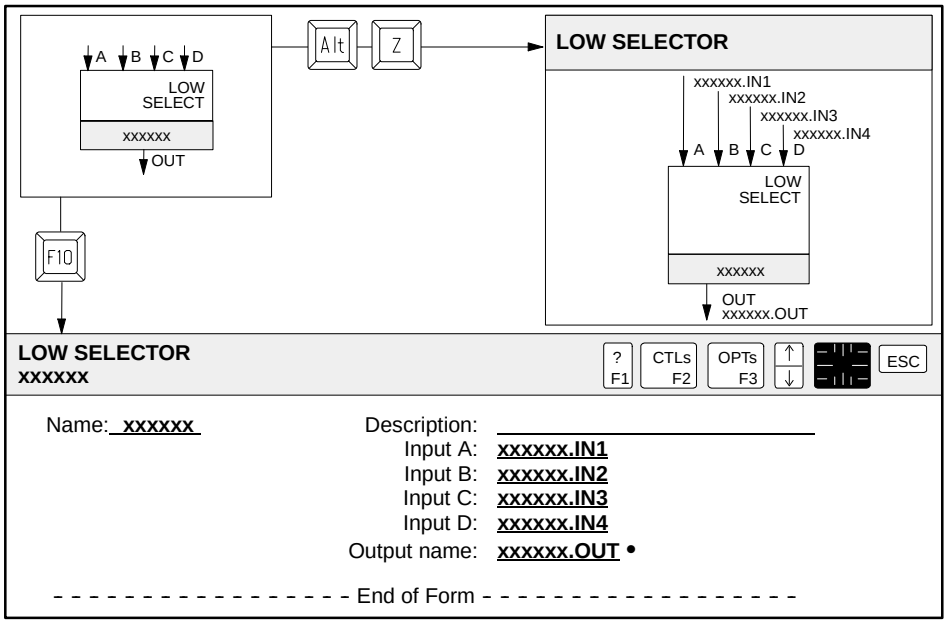


Figure 6-6 Low Selector Form

Input A: name of analog, real, or integer that is first input (A); default is *cfb_name*.**IN1**; field can be empty.

Input B: name of analog, real, or integer that is second input (B); default is *cfb_name*.**IN2**; field can be empty.

Input C: name of analog, real, or integer that is third input (C); default is *cfb_name*.**IN3**; field can be empty.

Input D: name of analog, real, or integer that is fourth input (D); default is *cfb_name*.**IN4**; field can be empty.

Output name: name of analog, real, or integer output (OUT) that represents value of lowest of enabled inputs; default is *cfb_name*.**OUT**.

Low Selector (continued)

Commands and Extensions

Table 6-7 lists the commands that determine the mode of operation for the low selector block.

Table 6-7 Low Selector Commands

Command	Description
ENABLE (ENA)	Turns on, or activates, the block.
DISABLE (DIS)	Turns off, or deactivates, the block.

Table 6-8 lists the extensions that you can use to monitor and control a low selector block.

Table 6-8 Low Selector Extensions

Read-only Boolean		Read-only Integer		Read-only Real	
.ENABLD	enabled	.IOUT	integer out [3]	.OUT	real output [3]
		.ECODE	error code [4]		
		.IID	error controller block [4]		
		.SNUM	error line number [4]		
		.POS	current low input		
Read/Write Boolean		**Read/Write Integer**		**Read/Write Real**	
.ENABL	enable [1]	.IIN1	integer input A [5]	.IN1	real input A [5]
.NRDY	not ready	.IIN2	integer input B [5]	.IN2	real input B [5]
.REN	request enable [2]	.IIN3	integer input C [5]	.IN3	real input C [5]
.RDIS	request disable [2]	.IIN4	integer input D [5]	.IN4	real input D [5]
		.SW1	input A enable		
		.SW2	input B enable		
		.SW3	input C enable		
		.SW4	input D enable		

1 This extension is manipulated by the command(s) associated with the block.

2 These booleans are APT flags that are manipulated by the Math procedures Latch, Clear, and On.

3 For each output, APT creates **.IOUT** if output value is an integer or analog, or **.OUT** if value is a real number.
The outputs are read-only variables if the block is enabled and read/write when the block is disabled.

4 These extensions do not contain error codes for S5 controllers. (Series 505-only error codes)

5 For each input, APT creates **.IIN** if input value is an integer or **.IN** if value is a real number.

6.6 Median Selector

Overview

The median selector block selects the numerical median of three inputs.

Availability

Median selector blocks are available for Series 505 and S5 controllers. They are not supported for the 560/560T controllers.

Block Configuration

When you define a median selector CFB with the form in Figure 6-7, you have the following options:

- You can use the default names that appear in the form. The default extensions must correspond to the correct slot; i.e., **.IN1** or **.IIN1** must be input A, etc.
- You can add new names: declared variables or dot extensions from other blocks.

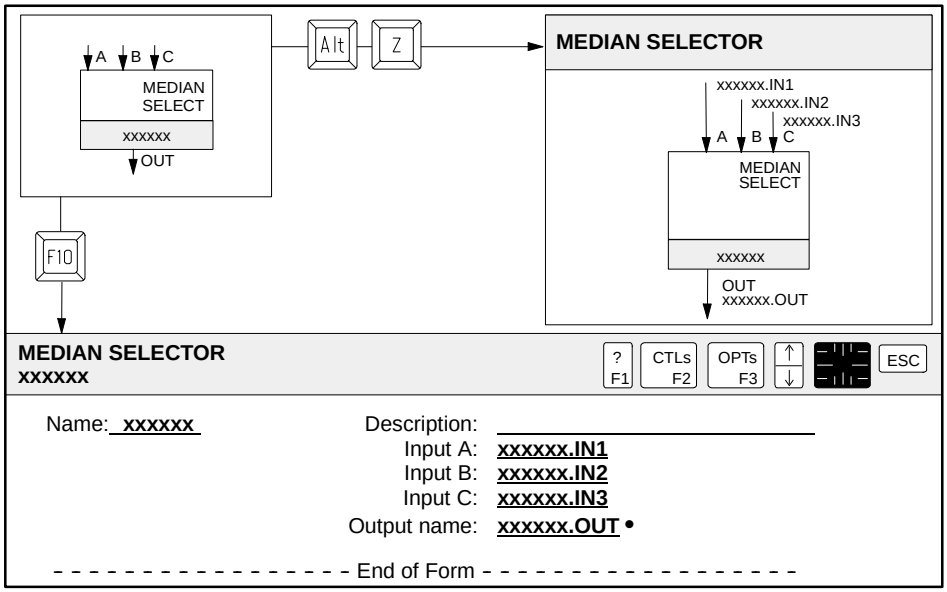


Figure 6-7 Median Selector Form

Input A: name of analog, real, or integer that is first input (A); default is *cfb_name*.**IN1**.

Input B: name of analog, real, or integer that is second input (B); default is *cfb_name*.**IN2**.

Input C: name of analog, real, or integer that is third input (C); default is *cfb_name*.**IN3**.

Output name: name of analog, real, or integer output (OUT) that represents value of median of enabled inputs; default is *cfb_name*.**OUT**.

Median Selector (continued)

Commands and Extensions

Table 6-9 lists the commands that determine the mode of operation for the median selector block.

Table 6-9 Median Selector Commands

Command	Description
ENABLE (ENA)	Turns on, or activates, the block.
DISABLE (DIS)	Turns off, or deactivates, the block.

Table 6-10 lists the extensions that you can use to monitor and control a median selector block.

Table 6-10 Median Selector Extensions

Read-only Boolean		Read-only Integer		Read-only Real	
.ENABLD	enabled	.IOUT	integer out [3]	.OUT	real output [3]
		.ECODE	error code [4]		
		.IID	error controller block [4]		
		.SNUM	error line number [4]		
		.POS	position of median		
Read/Write Boolean		**Read/Write Integer**		**Read/Write Real**	
.ENABL	enable [1]	.IIN1	integer input A [5]	.IN1	real input A [5]
.NRDY	not ready	.IIN2	integer input B [5]	.IN2	real input B [5]
.REN	request enable [2]	.IIN3	integer input C [5]	.IN3	real input C [5]
.RDIS	request disable [3]				

1 This extension is manipulated by the command(s) associated with the block.

2 These booleans are APT flags that are manipulated by the Math procedures Latch, Clear, and On.

3 For each output, APT creates **.IOUT** if output value is an integer or analog, or **.OUT** if value is a real number.
The outputs are read-only variables if the block is enabled and read/write when the block is disabled.

4 These extensions do not contain error codes for S5 controllers. (Series 505-only error codes)

5 For each input, APT creates **.IIN** if input value is an integer or **.IN** if value is a real number.

6.7 Outswitch Selector

Overview

The outswitch selector block allows you to send an input signal to any one of four outputs. You select the output by setting the value of the **.POS** extension. You can set the value of the **.POS** extension directly or you can use the SWITCH command in an SFC step.

For example, both of these instructions produce the same result, namely, they send the input signal to output C.

```
block_name .POS := 3;
SWITCH block_name C;
```

Availability

Outswitch selector blocks are available for Series 505 and S5 controllers. They are not supported for the 560/560T controllers.

Block Configuration

When you define an outswitch selector CFB with the form in Figure 6-8, you have the following options:

- You can use the default names that appear in the form. The default extensions must correspond to the correct slot; i.e., **.OUT1** or **.IOUT1** must be output A, etc.
- You can add new names: declared variables or dot extensions from other blocks.
- You can leave fields blank.

APT, also allows you to write directly to the unselected output; e.g., if the switch is set to A, you can write to the B, C, and D **.PV$_x$** extensions.

Outswitch Selector (continued)

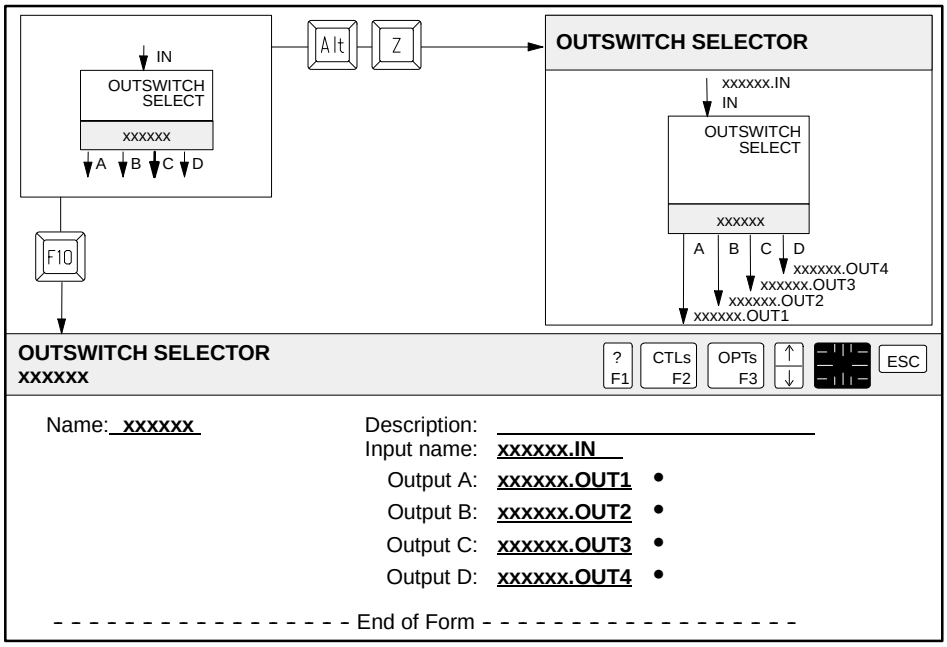


Figure 6-8 Outswitch Selector Form

Input name (IN): name of analog, real, or integer input; default is *cfb_name*.**IN**.

Output A (A): name of analog, real, or integer that is first output; default is *cfb_name*.**OUT1**; field can be empty.

Output B (B): name of analog, real, or integer that is second output; default is *cfb_name*.**OUT2**; field can be empty.

Output C (C): name of analog, real, or integer that is third output; default is *cfb_name*.**OUT3**; field can be empty.

Output D (D): name of analog, real, or integer that is fourth output; default is *cfb_name*.**OUT4**; field can be empty.

Commands and Extensions

Table 6-11 lists the commands that determine the mode of operation for the outswitch selector block.

Table 6-11 Outswitch Selector Commands

Command	Description
ENABLE (ENA)	Turns on, or activates, the block.
DISABLE (DIS)	Turns off, or deactivates, the block.
SWITCH	Selects input A, B, C, or D and sends to output; for example **SWITCH block_name A**; sends input to output A.

Table 6-12 lists the extensions that you can use to monitor and control a outswitch selector block.

Table 6-12 Outswitch Selector Extensions

Read-only Boolean		Read-only Integer		Read-only Real	
.ENABLD	enabled	.IOUT1	integer out A [3]	.OUT1	real output A [3]
		.IOUT2	integer out B [3]	.OUT2	real output B [3]
		.IOUT3	integer out C [3]	.OUT3	real output C [3]
		.IOUT4	integer out D [3]	.OUT4	real output D [3]
		.ECODE	error code [4]		
		.IID	error controller block [4]		
		.SNUM	error line number [4]		
Read/Write Boolean		**Read/Write Integer**		**Read/Write Real**	
.ENABL	enable [1]	.IIN	integer input A [5]	.IN	real input [5]
.NRDY	not ready	.POS	current output position	.PV1	alternate out A
.REN	request enable [2]			.PV2	alternate out B
.RDIS	request disable [2]			.PV3	alternate out C
				.PV4	alternate out D

1 This extension is manipulated by the command(s) associated with the block.

2 These booleans are APT flags that are manipulated by the Math procedures Latch, Clear, and On.

3 For each output, APT creates **.IOUT** if output value is an integer or analog, or **.OUT** if value is a real number.
The outputs are read-only variables if the block is enabled and read/write when the block is disabled.

4 These extensions do not contain error codes for S5 controllers. (Series 505-only error codes)

5 For each input, APT creates **.IIN** if input value is an integer or **.IN** if value is a real number.

6.8 Threshold Selector

Overview

The threshold selector block monitors a variable to determine whether or not its value falls below or rises above a specified limit. The initial value of the limit is taken from the form definition but can be changed from the program.

You can select either a high threshold or a low threshold:

- If you select a high threshold, a boolean output becomes true if the input value is greater than or equal the limit value.
- If you select a low threshold, a boolean output becomes true if the input value is less than or equal the limit value.

Availability

Threshold selector blocks are available for Series 505 and S5 controllers. They are not supported for the 560/560T controllers.

Block Configuration

When you define a threshold selector CFB with the form in Figure 6-9, you have the following options:

- You can use the default names that appear in the form.
- You can add new names: declared variables or dot extensions from other blocks.

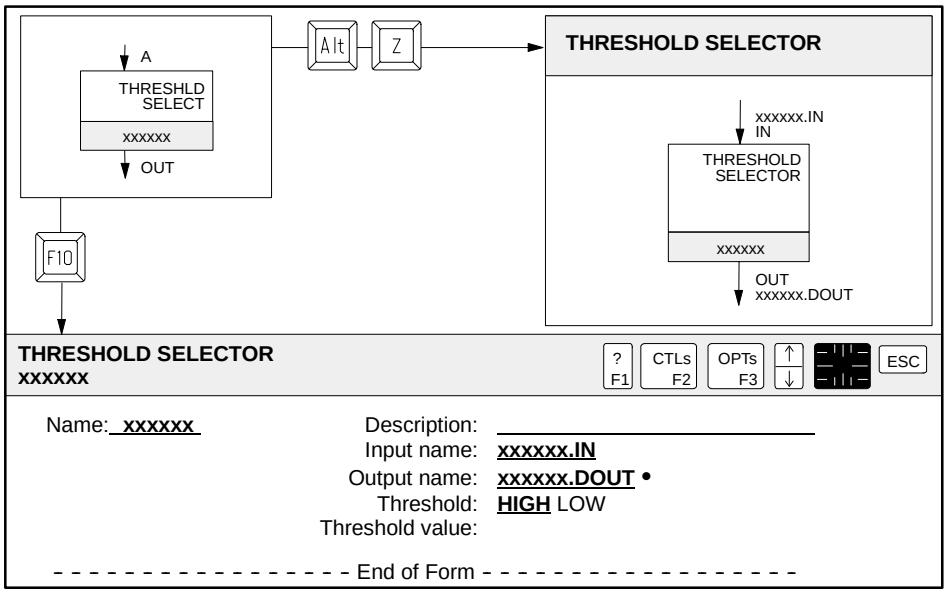


Figure 6-9 Threshold Selector Form

Input name: name of analog, real or integer input (A); default is *cfb_name*.**IN**.

Output name: name of digital output or boolean variable that is output (OUT); default is *cfb_name*.**DOUT**.

Threshold: HIGH or LOW; the default is HIGH.

Threshold value: integer or real value of limit; value must be same type (integer or real) as input.

Threshold Selector (continued)

Commands and Extensions

Table 6-13 lists the commands that determine the mode of operation for the threshold selector block.

Table 6-13 Threshold Selector Commands

Command	Description
ENABLE (ENA)	Turns on, or activates, the block.
DISABLE (DIS)	Turns off, or deactivates, the block.

Table 6-14 lists the extensions that you can use to monitor and control a threshold selector block.

Table 6-14 Threshold Selector Extensions

Read-only Boolean		Read-only Integer		Read-only Real	
.ENABLD	enabled	.SNUM	error line number [4]		
.DOUT	boolean output [1]	.ECODE	error code [4]		
		.IID	error controller block [4]		
Read/Write Boolean		**Read/Write Integer**		**Read/Write Real**	
.ENABL	enable [2]	.IIN	integer input A [5]	.IN	input [5]
.NRDY	not ready	.LIMIT	limit if input is integer [5]	.LIMIT	limit if input is real [5]
.REN	request enable [3]				
.RDIS	request disable [3]				

1 The output is a read-only variable when the block is enabled and read/write when the block is disabled.

2 This extension is manipulated by the command(s) associated with the block.

3 These booleans are APT flags that are manipulated by the Math procedures Latch, Clear, and On.

4 These extensions do not contain error codes for S5 controllers. (Series 505-only error codes)

5 If the input is an integer, APT creates **.IIN**; and **.LIMIT** is an integer. If the input is a real value, APT creates **.IN** and **.LIMIT** is a real value.

Chapter 7

Valve Control Blocks

7.1 Understanding Valve Control Blocks

Overview

The APT valve blocks provide additional valve control:

- The motor position control block allows you to set a motor driven valve in an intermediate position, neither fully opened or fully closed.
- The proportional time control block allows you to make a discrete device perform like a continuous device.
- The split range block allows you to use analog outputs for valve control.
- The valve sequencer allows you to control two valves with one output.

Availability

All valve blocks are available for Series 505 and S5 controllers. They are not supported for the 560/560T controllers.

Enabling and Disabling Blocks

All valve CFBs must be activated with the ENABLE command or with an assignment statement that sets the **.ENABL** extension to true. See Figure 7-1. Valve blocks can also be enabled from an assignment statement in a math CFB.

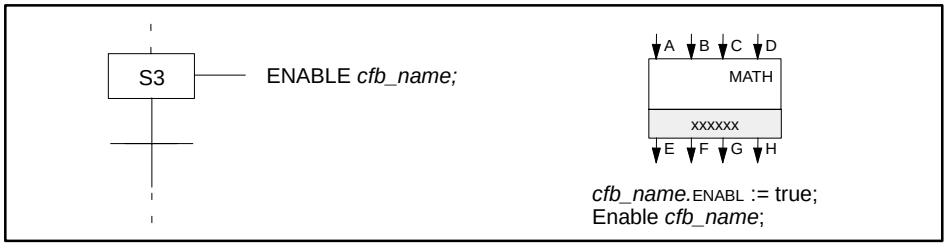


Figure 7-1 Enabling a Valve CFB

To deactivate the block, use the DISABLE command with the block name, or set the **.ENABL** extension to false.

All valve blocks have a boolean extension (**.NRDY**) that you can use to interlock the execution of the block to some external conditions.

- If you set the **.NRDY** extension to true, the block is forced to the disabled state.
- To return the block to the enabled state you must not only set the **.NRDY** extension to false, but you must also re-enable the block with an enable command or an assignment statement.

7.2 Motor Position Control

Overview

The motor position control block is used to drive a motor-driven valve to an intermediate position between 0 and 100%. The motor-driven valve is a device (VMD) that you define in the Device Definition Table.

The input to the block is a feedback signal that indicates the position of the valve. The setpoint should be a real value between 0.0 and 1.0. The algorithm compares the input to the setpoint.

- If the difference between the input and the setpoint is outside the range specified by the dead zone, the signals that open and close the valve are manipulated.
- If the difference between the input and the setpoint is within the dead zone, no control action is taken.

The dead zone is entered in the form as a percentage (0 to 100). If you want to change the dead zone from the program, write a value that is between 0.0 and 1.0 to the **.DBAND** (dead band) extension.

The motor position control block expects the valve to be in a locked state. (See the "Devices" chapter in the *APT Programming Reference (Tables) Manual* for information about locking a motor-driven valve.) If the block is enabled and the valve is not locked, the block does not execute until the valve is locked. The block does not check any other status bits that are associated with the valve. Every time the block is enabled, the value of the input is copied to the setpoint, effecting a "bumpless" transfer.

Availability

The motor position control block is supported for Series 505 and S5 controllers, but not the 560/560T controllers.

Motor Position Control (continued)

Block Configuration

When you define a motor position control CFB with the form in Figure 7-2, you have the following options:

- You can use the default names that appear in the form.
- You can add new names: declared variables or dot extensions from other blocks.

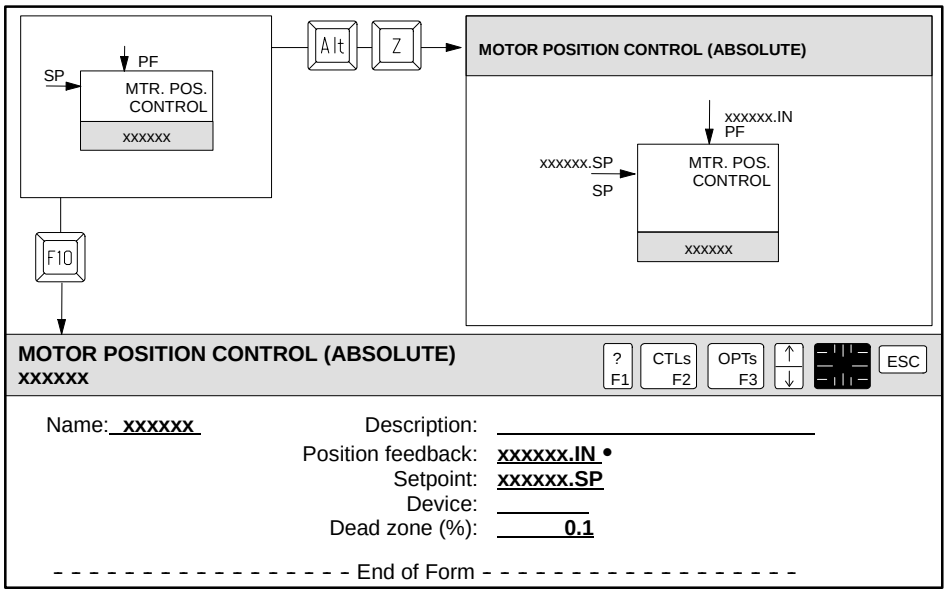


Figure 7-2 Motor Position Control Form

Position feedback: name of analog or real input (PV) that represents position feedback of valve as a scaled value between 0 and 1; default is *cfb_name*.**IN**.

Setpoint: name of real variable (SP) that represents setpoint (desired position of valve) as a scaled value between 0 and 1; default is *cfb_name*.**SP**.

Device: name of motor-driven valve (type VMD) that is to be controlled by this block; device is configured in Device Definition Table.

Dead zone(%): size of dead zone as a percentage (0 to 100); default is 0.1.

Commands and Extensions

Table 7-1 lists the commands that determine the mode of operation for the motor position control block.

Table 7-1 Motor Position Control Commands

Command	Description
ENABLE (ENA)	Turns on, or activates, the block.
DISABLE (DIS)	Turns off, or deactivates, the block.

Table 7-2 lists the extensions that you can use to monitor and control a motor position control block.

Table 7-2 Motor Position Control Extensions

Read-only Boolean		Read-only Integer		Read-only Real	
.ACTV	device locked	.SNUM	error line number [4]	.PV	real input [3]
		.ECODE	error code [4]		
		.IID	error controller block [4]		
Read/Write Boolean		**Read/Write Integer**		**Read/Write Real**	
.ENABL	enable [1]			.IN	real input [3]
.NRDY	not ready			.SP	setpoint
.REN	request enable [2]			.DBAND	deadband
.RDIS	request disable [2]				

1 This extension is manipulated by the command(s) associated with the block.

2 These booleans are APT flags that are manipulated by the Math procedures Latch, Clear, and On.

3 The PV is a read-only variable when the block is enabled and is read/write when the block is disabled.

4 These extensions do not contain the error codes for the S5 controllers. (Series 505-only error codes)

7.3 Proportional Time Control

Overview

The proportional time control block is used to make an on/off device act like a continuous device. This block allows you to set an output to true for a specified fraction of its duty cycle and to false the rest of the time.

Proportional time control is commonly applied to extruders. For example, you might control the temperature of a zone in an extruder by switching the heater associated with that zone on and off.

By pulsing an output according to the load on the system, the oscillation of the controlled variable can be reduced; and the amount of energy added to the system is more in line with what is needed. For example, a fully loaded system has full heating all of the time; a system with no load has heating none of the time.

The input to the block specifies the fraction of the time that the control output should be true (open or running for devices). The duty cycle specifies how often the algorithm should re-evaluate the on/off time of the output. For example, if the duty cycle is 10 seconds and the input variable is 0.5, the output is true for five seconds and false for five seconds. At the end of each ten-second period, the input is re-evaluated to determine the on/off time for the output.

Any device except a two-speed or reversible motor can be used with the proportional time control block. See the “Devices” chapter in the *APT Programming Reference (Tables) Manual* for information about these devices.

Availability

The proportional time control block is supported for both Series 505 and S5 controllers, but not for the 560/560T controllers.

Block Configuration

When you define a proportional time control CFB with the form that is shown in Figure 7-3, you have the following options:

- You can use the default names that appear in the form.
- You can add new names: declared variables or dot extensions from other blocks.

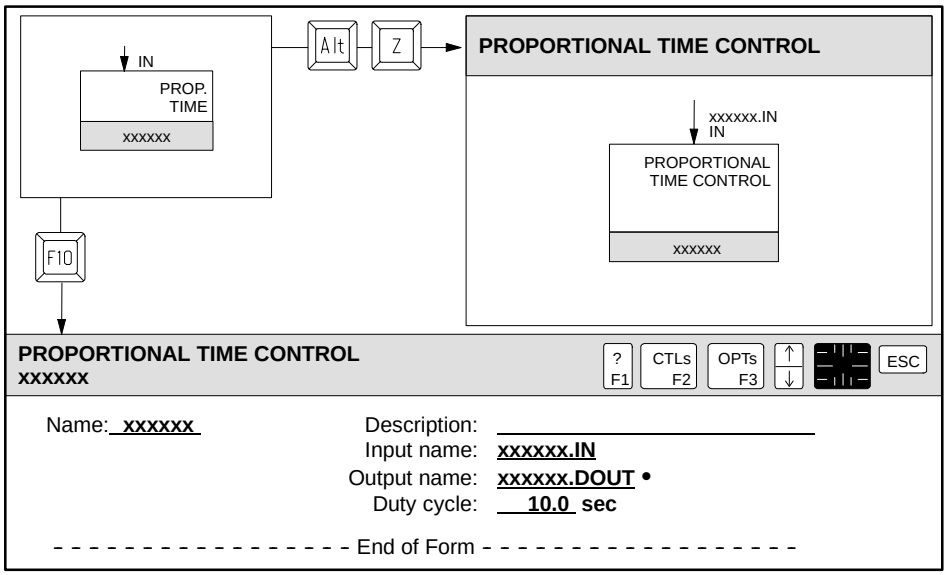


Figure 7-3 Proportional Time Control Form

Input name: name of real input (IN) that represents fraction of duty cycle during which the output is true; value must be between 0.0 and 1.0; default is *cfb_name*.**IN**.

Output name: name of output (OUT); name can be boolean or discrete variable or name of device (VSS, VMD, VSD, VSN, VDD, MDS, MDN, MSN, MSS). The specified output becomes true for the fraction of the duty cycle specified by input; it has a value of false for the remainder of the duty cycle; default is *cfb_name*.**DOUT**.

Duty cycle: duration of duty cycle, in seconds; value depends on the timer compiled. An initial value ≤ 32 seconds generates a fast timer. A fast timer has a range of 0.001 to 32.767 seconds in 0.001 second increments. An initial value > 32 seconds generates a slow timer. A slow timer has a range of 0.1 to 3276.7 seconds in 0.1 second increments. If you have an S5 controller, APT automatically assigns you the slow timer.

During operation, the range for the duty cycle dot extension value is limited to the range of the timer generated. Although you can enter a specific duty cycle, the execution time of the CFB is determined in part by how fast the associated SFPGM for Series 505 can be executed, or how fast the scan time is for S5 controllers.

Proportional Time Control (continued)

> **⚠ WARNING**
>
> **APT allows the duty cycle of the proportional time control block to be changed while the block is enabled. The new duty cycle value does not take effect immediately.**
>
> **Changing the value could cause unpredictable and/or unsafe process operation that could result in death or serious injury to personnel, and/or equipment damage.**
>
> **Always disable the proportional time control block before changing the value of the duty cycle.**

Commands and Extensions

Table 7-3 lists the commands that determine the mode of operation for the proportional time control block.

Table 7-3 Proportional Time Control Commands

Command	Description
ENABLE (ENA)	Turns on, or activates, the block.
DISABLE (DIS)	Turns off, or deactivates, the block.

Table 7-4 lists the extensions that you can use to monitor and control a proportional time control block.

Table 7-4 Proportional Time Control Extensions

Read-only Boolean		Read-only Integer		Read-only Real	
.DOUT	output to device [1]	.SNUM	error line number [4]		
		.ECODE	error code [4]		
		.IID	error controller block [4]		
Read/Write Boolean		**Read/Write Integer**		**Read/Write Real**	
.ENABL	enable [2]			.IN	fraction of time output is true
.NRDY	not ready			.DUTY	duty cycle
.REN	request enable [3]				
.RDIS	request disable [3]				

1 The output is a read-only variable when the block is enabled and read/write when the block is disabled.

2 This extension is manipulated by the command(s) associated with the block.

3 These booleans are APT flags that are manipulated by the Math procedures Latch, Clear, and On.

4 These extensions do not contain the error codes for the S5 controllers. (Series 505-only error codes)

7.4 Split Range

Overview

The split range block allows you to split an input signal into two output signals. The input must be a real number between 0 and 1.

As the input signal changes from 0.0 to 1.0, the two outputs change. When you define the block, you enter the input range that specifies the input at which each output first reaches the 0.0% state.

- When the input is 0.0, the first output (A) is 100% (1.0); the second output (B) is 0.0% (0.0). If the reverse scaling option is selected, both outputs (A and B) are 0.0% (0.0). See Figure 7-4.
- When the input is 1.0, the first output (A) is 0.0% (0.0); the second output (B) is 100.0% (1.0). If the reverse scaling option is selected, both outputs (A and B) are 100% (1.0). See Figure 7-4.

Breakpoints can overlap so that at some time both outputs are non-zero over some range of input values. Also, the breakpoint ranges can be set so that there is a band of input ranges over which both outputs are 0.0.

If either output is an analog output, the split range block scales the outputs to the appropriate ranges that you specify for the analog outputs. An extension is also created (**.OUT**) that contains the scaled real value (0.0 to 1.0) of the output.

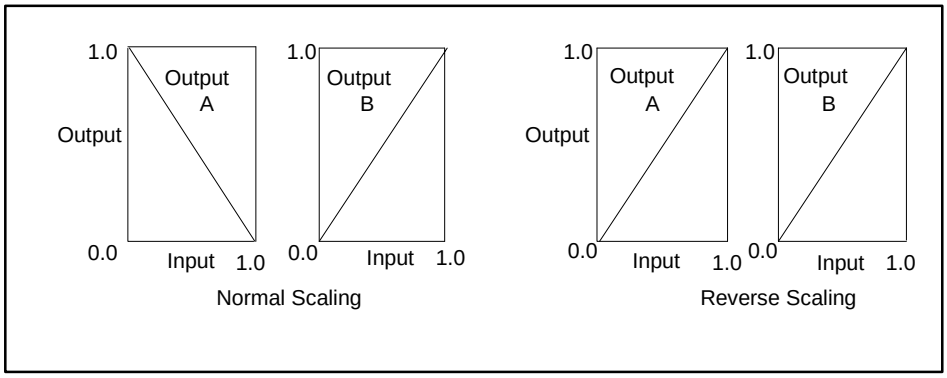


Figure 7-4 Split Range Reverse and Normal Scaling

Split Range (continued)

When the split range block is not active, the outputs are not updated by the split range block.

- If you want to modify the outputs when the block is disabled, you must write the correct values to the analog outputs. For example, if the first output is a 20% offset signal, then you must write the appropriate value to the analog variable to force the signal to its zero value. For Series 505 controllers the value is 6400; for S5 controllers, the value is 512.
- The preferred method of manipulating the outputs is to write a value to the input that sets the outputs to the desired values, and then activate the block.

For Output 1, to use the split range with a reverse-acting activator (i.e., air to close), place an **X** in the Reverse Scaling field, shown in the dialogue window below.

```
                    As the input changes from      0% to      49.5%
                    output 1 changes from        100% to        0%

                    Is Output 1 Air-to-close (reverse scaling)? [X]

- - - - - - - - - - - - - - - - - End of Form - - - - - - - - - - - - - - - - - -
```

If you want Output 2 to do reverse-scaling, connect the real output of the split-range block to a scale block.

- In the scale block, specify that “as the input changes from 1.0 to 0.0, the output changes from 0.0 to 1.0.”
- Define the output of the scale block as the desired analog output variable.

Availability

The split range block is supported for both Series 505 and S5 controllers, but not for the 560/560T controllers.

Block Configuration

When you define a split range CFB with the form that is shown in Figure 7-5, you have the following options:

- You can use the default names that appear in the form.
- You can add new names: declared variables or dot extensions from other blocks.

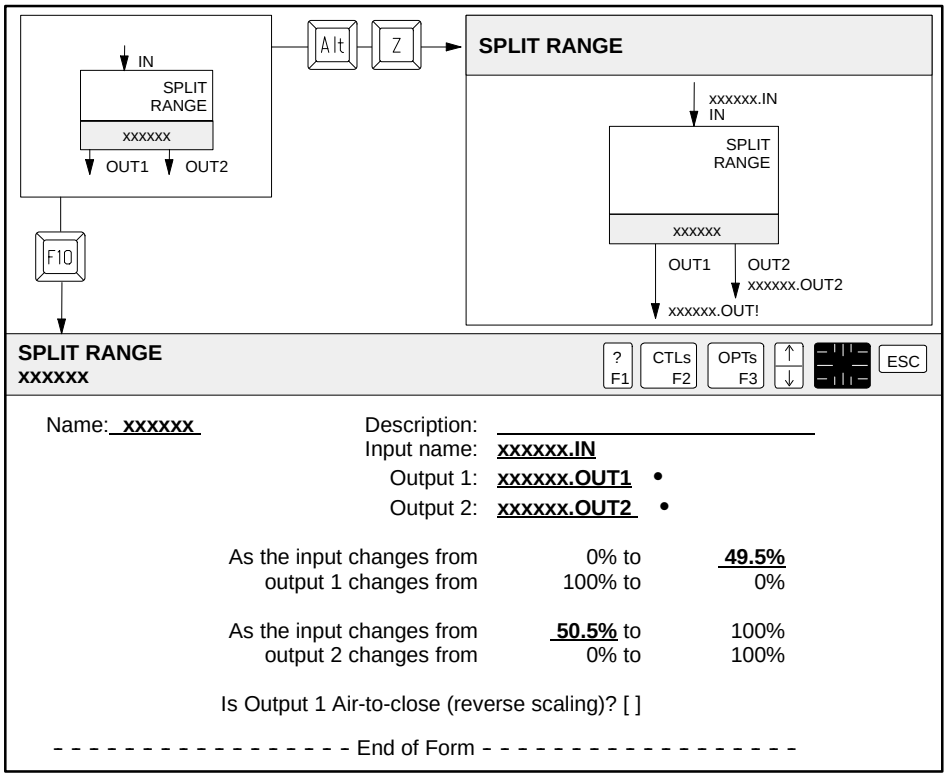


Figure 7-5 Split Range Form

Input name: name of real input (IN); value must be between zero and one; default is *cfb_name*.**IN**.

Output 1: name of analog, integer or real that is first output (OUT1); default is *cfb_name*.**OUT1**.

Output 2: name of analog, integer or real that is second output (OUT2); default is *cfb_name*.**OUT2**.

As the input changes from 0% to _____ output 1 changes from 100% to 0%: breakpoint of the first range; that is, the percentage of input at which the value of the first output is zero; default is 49.5. If the reverse scaling option is selected, the first output (A) is 100% (1.0); the second output (B) is 0% (0.0). If you want Output 2 to do reverse-scaling, connect the real output of the split-range block to a scale block.

As the input changes from _____ to 100% output 2 changes from 0% to 100%: breakpoint of the second range; that is, the percentage of input at which the value of the second output is zero; default is 50.5.

Is Output 1 Air-to-close (reverse scaling): output 1 changes from 0% to 100% as input changes from 0% to_____%.

Split Range (continued)

Commands and Extensions

Table 7-5 lists the commands that determine the mode of operation for the split range block.

Table 7-5 Split Range Commands

Command	Description
ENABLE (ENA)	Turns on, or activates, the block.
DISABLE (DIS)	Turns off, or deactivates, the block.

Table 7-6 lists the extensions that you can use to monitor and control a split range block.

Table 7-6 Split Range Extensions

Read-only Boolean		Read-only Integer		Read-only Real	
.ENABLD	enabled	.SNUM	error line number [3]	.OUT1	real output A [4]
		.ECODE	error code [3]	.OUT2	integer output B [4]
		.IID	error controller block[3]		
		.IOUT1	integer output A [4]		
		.IOUT2	integer output B [4]		
Read/Write Boolean		**Read/Write Integer**		**Read/Write Real**	
.ENABL	enable[1]			.IN	real input
.NRDY	not ready			.BRKP1	1st breakpoint
.REN	request enable [2]			.BRKP2	2nd breakpoint
.RDIS	request disable [2]				

1 This extension is manipulated by the command(s) associated with the block.

2 These booleans are APT flags that are manipulated by the Math procedures Latch, Clear, and On.

3 These extensions do not contain the error codes for the S5 controllers. (Series 505-only error codes)

4 For each output, APT creates **.IOUT** if output is an integer or analog, or **.OUT** if value is a real number. The outputs are read-only variables if the block is enabled, and read/write when the block is disabled.

7.5 Valve Sequencer

Overview

The valve sequencer block manipulates two continuous control valves with a single output. A control I/O point determines which valve responds to the output signal.

The output of the valve sequencer block controls two valves: one must be an “energize open” (EO) valve, and the other must be an “energize close” (EC) valve.

One valve is associated with the false value of the control I/O point and the other with the true value. When the output signal reaches the increasing switch point, the “false” valve is fully open. When the output signal reaches the decreasing switch point, the “true” valve is at a fully closed position.

You can select valve action of either direct or reverse if the output is an integer or an analog output.

- Direct action increases the output as the input increases.
- Reverse action decreases the output as the input increases.

You can select a characterization type of none, equal %, or interpolation.

- If you select a characterization type of none, there is a linear relationship between the input and the output and they are always equal.
- If you select a characterization type of equal %, the relationship between the input and output is determined by the following equation:

$$output = \frac{input}{Z + ((1 - Z) \times input)}$$

$$where\ Z = \sqrt{\frac{1}{rangeability}}$$

- If you select a characterization type of interpolation, the output is determined by input and output arrays that each contain 11 elements. (See the explanation of Correlated Lookup Table in Section 9.4 for more information about these arrays.)

Availability

The valve sequencer block is supported for Series 505 and S5 controllers, but not the 560/560T controllers.

Valve Sequencer (continued)

Block Configuration

When you define a valve sequencer with the form that is shown in Figure 7-6, you have the following options:

- You can use the default names that appear in the form.
- You can add new names: declared variables or dot extensions from other blocks.

The shaded entries in the form depend on a previous selection.

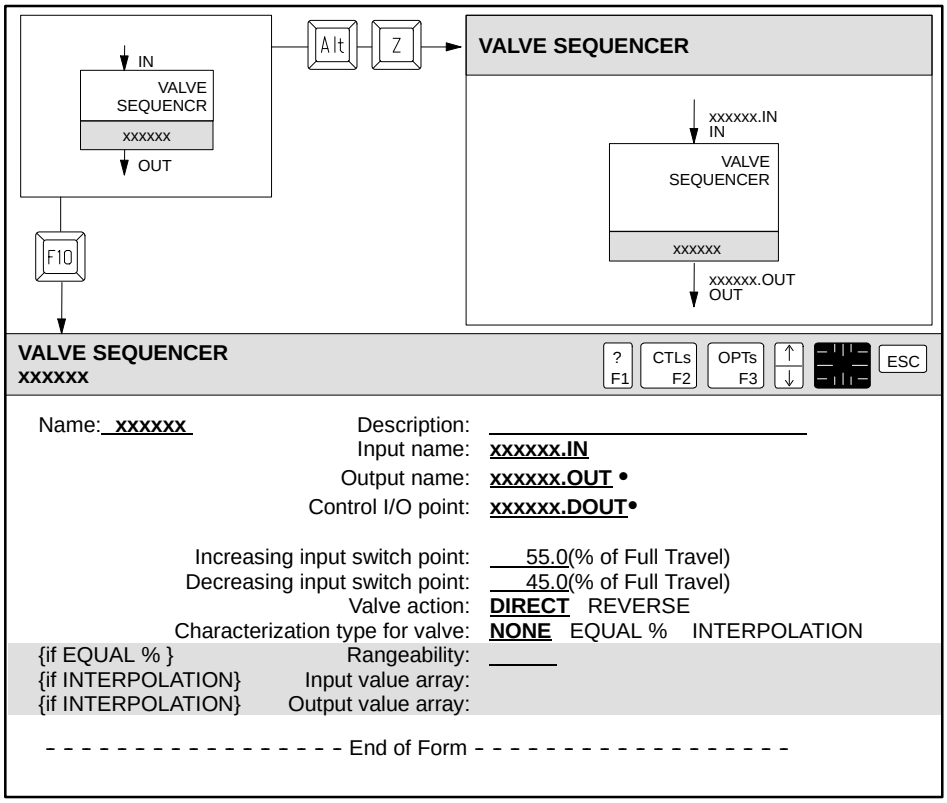


Figure 7-6 Valve Sequencer Form

Input name: name of real input (IN); default is *cfb_name*.**IN**.

Output name: name of analog, real, or integer output (OUT); default is *cfb_name*.**OUT**.

Control I/O point: name of control I/O point, which can be a discrete or boolean variable; default is *cfb_name*.**DOUT**.

Increasing input switch point: real number between 0.0 and 100.0 that indicates value at which an increase in input causes the control I/O point to become true.

Decreasing input switch point: real number between 0.0 and 100.0 that indicates value at which a decrease in input causes the control I/O point to become false.

Valve action: default is DIRECT.

Characterization type for valve: EQUAL %, INTERPOLATION, or NONE; default is NONE.

Rangeability: displayed only if you select a characterization type of EQUAL %; a real number between 0.0 and 100.0.

Input value array: displayed only if you select INTERPOLATION; name of input array, which must contain 11 elements; values in array must be real values in ascending order.

Output value array: displayed only if you select INTERPOLATION; name of output array, which must contain 11 elements; values in array must be real values in ascending order.

Valve Sequencer (continued)

Commands and Extensions

Table 7-7 lists the commands that determine the mode of operation for the valve sequencer block.

Table 7-7 Valve Sequencer Commands

Command	Description
ENABLE (ENA)	Turns on, or activates, the block.
DISABLE (DIS)	Turns off, or deactivates, the block.

Table 7-8 lists the extensions that you can use to monitor and control a valve sequencer block.

Table 7-8 Valve Sequencer Extensions

Read-only Boolean		Read-only Integer		Read-only Real	
.ENABLD	enabled	.SNUM	error line number [4]	.OUT	real output [3]
		.ECODE	error code [4]		
		.IID	error controller block [4]		
		.IOUT	integer output [3]		
Read/Write Boolean		**Read/Write Integer**		**Read/Write Real**	
.ENABL	enable [1]			.IN	real input
.NRDY	not ready			.BRKPH	true breakpoint
.DOUT	output			.BRKPL	false breakpoint
.REN	request enable [2]			.RGN	rangeability value
.RDIS	request disable [2]				

1 This extension is manipulated by the command(s) associated with the block.

2 These booleans are APT flags that are manipulated by the Math procedures Latch, Clear, and On.

3 For each output, APT creates **.IOUT** if output is an integer or analog, or **.OUT** if value is a real number. The outputs are read-only variables if the block is enabled, and read/write when the block is disabled.

4 These extensions do not contain the error codes for the S5 controllers. (Series 505-only error codes)

Chapter 8

Math Control Blocks

8.1 Understanding Math Control Blocks

Overview

The APT math control blocks provide continuous math operations. Three basic types of math control blocks are available:

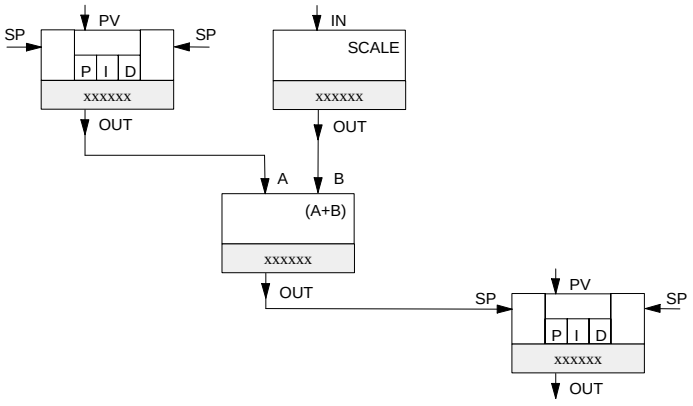
- The arithmetic blocks include absolute value, divider, multiplier, subtractor, square, square root, and summer. These can be used individually or with inputs from other blocks. For an example of how arithmetic blocks are used, see Figure 8-1.

  Arithmetic blocks are enabled for continuous execution and execute as often as the controller allows. For Series 505, they are grouped together in an SF program, separate from all other blocks (including math and interlock). The order of execution does not follow data flow connections. For APT, the algorithm that controls object download affects the execution flow of CFBs.

- The interlock block provides execution of Math Language statements that control safety interlocking.

- The math block is similar to the interlock block except that it can be enabled and disabled. If you want to solve an involved equation, use this math block instead of combining arithmetic blocks.

Figure 8-1 shows how arithmetic blocks can be used.



To ensure that math is executed synchronously with a PID, make sure that you include math for a PID within the associated math section for that PID.

The scale and summer CFBs in this example can be replaced by math statements in the associated math section of the PID CFB in the top left corner.

Figure 8-1 Using Math Control Blocks

Availability

The arithmetic, interlock, and math blocks are supported for Series 505 and S5 controllers. For 560/560T controllers, math blocks and interlock blocks are supported; arithmetic blocks are not.

For S5 controllers, APT automatically compiles all math operations into STL code, whereas the type of code used for Series 505 controllers (RLL or SFPGM) depends on the math operation being performed. In cases where an operation can be performed using either RLL or SFPGM, you have the option to specify what type of code APT uses for compile.

See Chapter 10 for more information about how APT compiles math blocks into STL, RLL, and SFPGM code.

Inputs and Outputs

The inputs to math control blocks can be analog inputs, integers, or real numbers.

- The interlock block has no inputs or outputs.
- Inputs to the math block can be any combination of integers, real numbers, or empty fields.
- For S5 controllers, inputs to the absolute value, divider, multiplier, subtractor, square, square root, and summer blocks must all be the same type, i.e., all integer or all float.

  If the **.OUT** extension variable is an integer and the inputs are not integers, **.OUT** is assigned the appropriate value rounded to the nearest integer.

Math control blocks provide no scaling on analog outputs.

8.2 Absolute Value

Overview

The absolute value block computes the absolute value of a variable.

The absolute value block is an active block and cannot be disabled.

When you define an absolute value CFB with the form in Figure 8-2, you have the following options:

- You can use the default names that appear in the form.
- You can add new names: declared variables or dot extensions from other blocks.

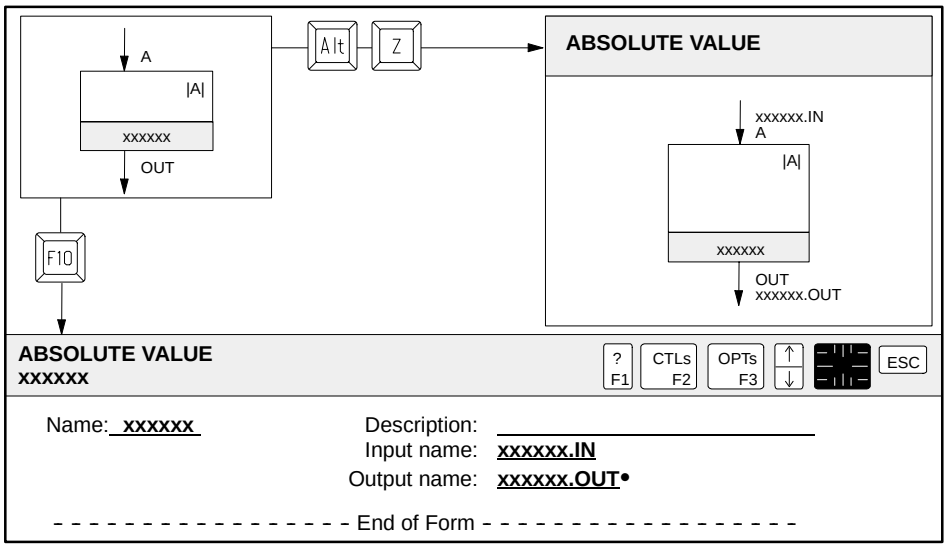


Figure 8-2 Absolute Value Form

Input name: name of analog, integer, or real input (A); default is *cfb_name*.**IN**.

Output name: name of analog, integer, or real output (OUT), which represents absolute value of the input; default is *cfb_name*.**OUT**.

Availability

The absolute value block is supported for Series 505 and S5 controllers, but not for the 560/560T controllers.

Absolute Value (continued)

Extensions

Table 8-1 lists the extensions that you can use to monitor and control an absolute value block.

Table 8-1 Absolute Value Extensions

Read-only Integer		Read-only Real	
.IOUT	integer output [1]	.OUT	real output [1]
Read/Write Integer		**Read/Write Real**	
.IIN	integer input [2]	.IN	real input [2]
1 For each output, APT creates **.IOUT** if output is an integer or analog, or **.OUT** if value is a real number. The outputs are read-only variables if the block is enabled, and read/write when the block is disabled.			
2 For each input, APT creates **.IIN** if input value is an integer, or **.IN** if value is a real number.			

8.3 Divider

Overview

The divider block divides one input variable by another input variable.

The divider block is an active block and cannot be disabled.

When you define a divider CFB with the form that is shown in Figure 8-3, you have the following options:

- You can use the default names that appear in the form.
- You can add new names: declared variables or dot extensions from other blocks.

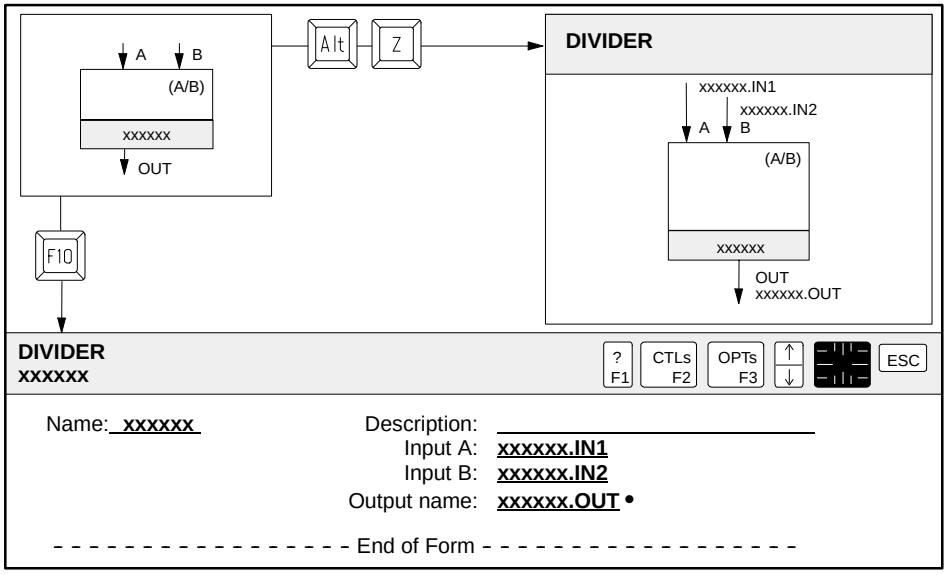


Figure 8-3 Divider Form

Input A: name of analog, integer, or real input (A) that is dividend; default is *cfb_name*.**IN1**.

Input B: name of analog, integer, or real input (B), which represents divisor; default is *cfb_name*.**IN2**.

Output name: name of analog, integer, or real output (OUT), which represents quotient of inputs; default is *cfb_name*.**OUT**.

Availability

The divider block is supported for Series 505 and S5 controllers, but not for the 560/560T controllers.

Divider (continued)

Extensions

Table 8-2 lists the extensions that you can use to monitor and control a divider block.

Table 8-2 Divider Extensions

Read-only Integer		Read-only Real	
.IOUT	integer output [1]	.OUT	real output [1]
Read/Write Integer		**Read/Write Real**	
.IIN1	integer input A [2]	.IN1	real input A [2]
.IIN2	integer input B [2]	.IN2	real input B [2]

1 For each output, APT creates **.IOUT** if output is an integer or analog, or **.OUT** if value is a real number. The outputs are read-only variables if the block is enabled, and read/write when the block is disabled.

2 For each input, APT creates **.IIN** if input value is an integer, or **.IN** if value is a real number.

8.4 Interlock

Overview

The interlock block allows you to use the Math Language to create a CFB that executes continuously. The interlock block is a continuous block that has no inputs or outputs and that cannot be disabled.

NOTE: If you place an INIT section in an interlock block, the INIT section executes only on the first scan following a transition to RUN mode after download (one time only).

The interlock block allows you to write device-override control, or to define special devices. An interlock is typically used to force a device to a known state without regard to any step commands or assignments from other math blocks. An interlock block begins executing as soon as the controller changes to RUN mode.

High-priority interlocks are executed before SFCs, other CFBs, and devices. Low-priority interlocks are executed after devices. Refer to Chapter 2 in the *SIMATIC APT Applications Manual* for more details.

Figure 8-4 shows the form that you use to define an interlock block.

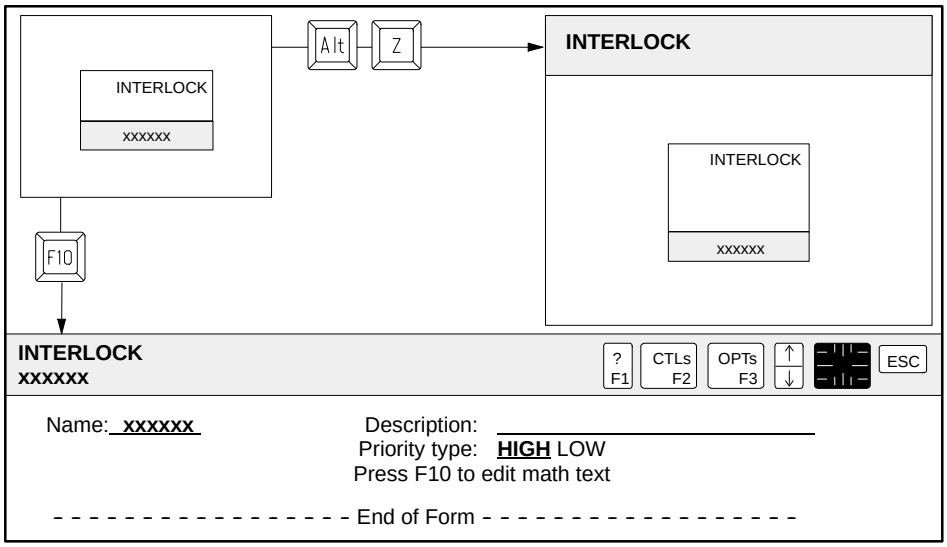


Figure 8-4 Interlock Form

Priority type: default is HIGH.

Availability

The interlock block is supported for all controllers.

Interlock (continued)

Priorities

You can select a priority type of either high or low.

- High-priority interlock blocks execute before any step or device logic. Because high-priority blocks are executed before any devices are enabled, they are commonly used for internal validation checks. For example, you might use this block to ensure that a valve is enabled before a motor is started.

  The following math code insures that the valve is always closed unless the motor is running.

  *valve1***.NRDY** := *motor1***.STPPD**

  The **.NRDY** bit of the valve depends on the state of the motor (running or stopped).

- Low-priority interlock blocks execute in the device section of RLL and execute after all SFC step commands. They have the same priority level as devices. They may be used, for example, to create your own devices to supplement the pre-programmed devices that are supplied with APT.

NOTE: For Series 505 controllers, a high or low priority affects an interlock block only if that block is composed entirely of RLL code. If a block contains any SFPGM code, the selection of high or low priority has no effect on the execution of that block. (See page 10-4 in Chapter 10, "Math Language Overview," for more information on RLL and SFPGM code.)

8.5 Math

Overview

The math block allows you to use the full power and flexibility of the Math Language to write your own controller code. See Chapter 10, "Math Language Overview," and Chapter 11, "Math Functions and Procedures," for an explanation of the Math Language.

Availability

The math block is supported for all controllers.

Controlling Speed of Series 505 Math Execution

A math block executes only when enabled. The speed of execution is determined both by the controller and by the type of code generated. All programming is done in STL for S5 controllers. However, if you have a Series 505 controller, your execution speed depends on the type of code used to perform your math operations. RLL executes more quickly than SFPGM.

- For a Series 505 controller, RLL is generated for the block if all operations are boolean, integer, or real number assignments, as shown in the following examples:

```
real_variable := 5.0 ;
real_variable1 := real_variable2;
array1[1] := 5;
```

- For a Series 505 controller, SFPGM code is generated when you use any real-number operations, or when you use any SF address types, such as T-memory direct addresses, as shown in the following examples:

```
real_variable := 5.0 * real_variable;
array2[int_variable] := 5;
 %T10 := 1;
```

If you have a Series 505 controller, balance your code between RLL and SFPGM in order to maximize controller utilization.

Math (continued)

Figure 8-5 shows the form that you use to define a math block. You can select from the following types: continuous, event, sampled, or active.

- A continuous block executes repeatedly, as often as possible.
- An event block executes only once. To repeat the execution of an event block, you must disable and then enable the block again. You might use an event block to generate a report or to total the ingredients at the end of a batch. For Series 505 controllers, this type of math block should not contain an INIT section, because only the INIT section will execute.
- A sampled block executes at intervals that you specify. The sample time is a compiled variable that cannot be changed at run time. The sample time is a read-only real value that is stored at address "%T4" for a Series 505 controller performing an SFPGM math operation, or at address *CFBname*.**NNL.PRE** for an S5 controller or a Series 505 controller performing an RLL math operation. For S5 controllers, the sample time value is in tenths of seconds.
- An active block is continuously active and cannot be turned off. An active block is similar to an interlock block of low priority.

⚠ WARNING

An event math block will be queued for execution if the enable for the block was set when the controller transitions to the RUN mode.

Control devices can operate in an unpredictable manner and fail in an unsafe condition that could result in death or serious injury to personnel, and/or damage to equipment.

The enable for the event math block needs to be reset to false immediately after the block finishes execution. Design your code to handle enabling and disabling of the math block enable after a transition to RUN mode.

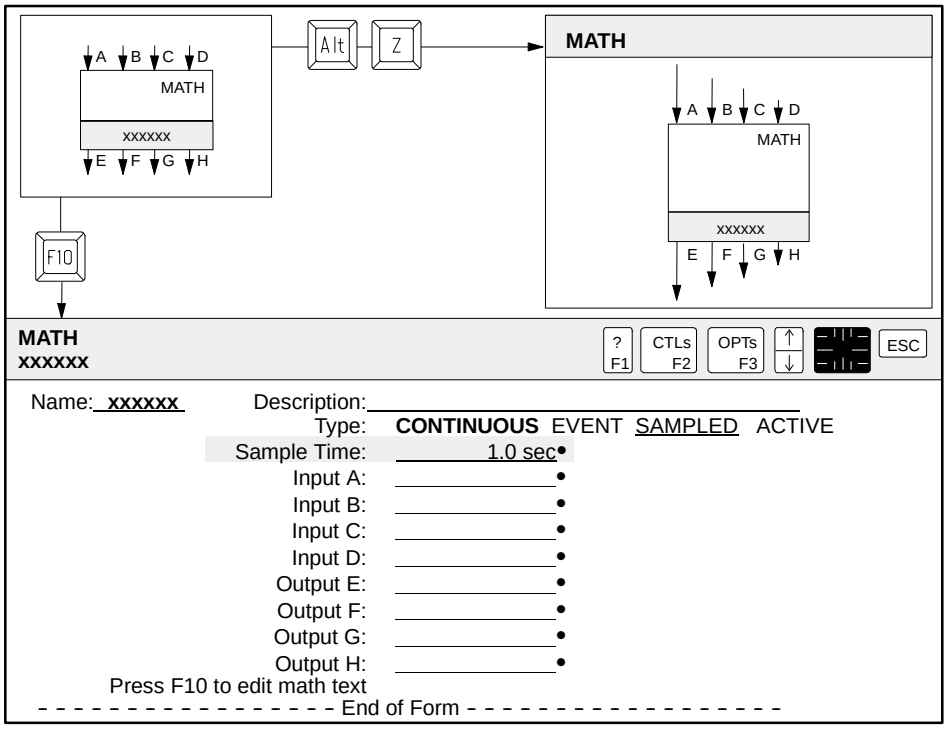


Figure 8-5 Math Form

Type: CONTINUOUS, EVENT, SAMPLED, ACTIVE; default: CONTINUOUS.
Sample time: sample time in seconds (only if you select SAMPLED type); default is 1 second. For Series 505 RLL-only math and for S5 STL math, valid values range from 0.1 to 3276.7 in 0.1 second increments. For Series 505 SFPGM-only math, valid values range from 0.5 to 4095.5 in 0.5 second increments.

Input A: name of real, integer, boolean, or APT flag variable that is input (A).
Input B: name of real, integer, boolean, or APT flag variable that is input (B).
Input C: name of real, integer, boolean, or APT flag variable that is input (C).
Input D: name of real, integer, boolean, or APT flag variable that is input (D).
Output E: name of real, integer, boolean, or APT flag variable that is output (E).
Output F: name of real, integer, boolean, or APT flag variable that is output (F).
Output G: name of real, integer, boolean, or APT flag variable that is output (G).
Output H: name of real, integer, boolean, or APT flag variable that is output (H).

If a field is left blank, no corresponding extension is created.

NOTE: You can enter extension variables in the fields on the form and avoid defining these variables in the Declaration Table. For example, enter *cfb_name*.**IN1** for Input A, *cfb_name*.**IN2** for Input B, etc.

Math (continued)

Enabling and Disabling Blocks

All math CFBs, except for active types, must be activated with the ENABLE command or with an assignment statement that sets the **.ENABL** extension to true. See Figure 8-6. Math blocks can also be enabled from an assignment statement in a math CFB.

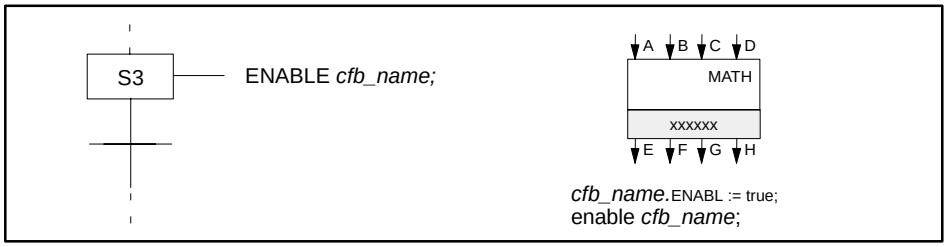


Figure 8-6 Enabling a Math CFB

To deactivate the block, use the DISABLE command with the block name or set the **.ENABL** extension to false. Active-type math CFBs cannot be deactivated.

All math blocks have a boolean extension (**.NRDY**) that you can use to interlock the execution of the block to some external conditions.

- If you set the **.NRDY** extension to true, the block is forced to the disabled state.
- To return the block to the enabled state you must not only set the **.NRDY** extension to false, but you must also re-enable the block with a command or an assignment statement.

Commands and Extensions

Table 8-3 lists the commands that determine the mode of operation for the math block.

Table 8-3 Math Commands

Command	Description
ENABLE (ENA)	Turns on, or activates, the block.
DISABLE (DIS)	Turns off, or deactivates, the block.

Table 8-4 lists the extensions that you can use to monitor and control a math block.

Table 8-4 Math Extensions

Read-only Boolean		Read-only Integer [4]		Read-only Real [4]	
.ENABLD	enabled [2]	.IOUTE	integer output E [5]	.OUTE	real output E [5]
.DOUTE	boolean output E	.IOUTF	integer output F [5]	.OUTF	real output F [5]
.DOUTF	boolean output F	.IOUTG	integer output G [5]	.OUTG	real output G [5]
.DOUTG	boolean output G	.IOUTH	integer output H [5]	.OUTH	real output H [5]
.DOUTH	boolean output H	.SNUM	error line number [7]		
		.ECODE	error code [7]		
		.IID	error controller block [7]		
Read/Write Boolean		**Read/Write Integer [4]**		**Read/Write Real [4]**	
.ENABL	enable [1,2]	.IINA	integer input A [6]	.INA	real input A [6]
.NRDY	not ready [2]	.IINB	integer input B [6]	.INB	real input B [6]
.REN	request enable [3]	.IINC	integer input C [6]	.INC	real input C [6]
.RDIS	request disable [3]	.IIND	integer input D [6]	.IND	real input D [6]
.DINA	boolean input A				
.DINB	boolean input B				
.DINC	boolean input C				
.DIND	boolean input D				

1 This extension is manipulated by the command(s) associated with the block.

2 These extensions do not exist for active-type math CFBs.

3 These booleans are APT flags that are manipulated by the Math procedures Latch, Clear, and On.

4 The extensions with the numbered inputs or outputs supported in earlier APT software releases (**.IOUT1**, **.IOUT2**, etc., **.OUT1**, **.OUT2**, etc., **.IIN1**, **.IIN2**, etc., **.IN1**, **.IN2**, etc.) are still supported.

5 For each output, APT creates **.IOUT** if output is an integer or analog, or **.OUT** if value is a real number. The outputs are read-only variables if the block is enabled, and read/write when the block is disabled.

6 For each input, APT creates **.IIN** if input value is an integer, or **.IN** if value is a real number.

7 These extensions do not contain error codes for S5 controllers. (Series 505-only error codes)

8.6 Multiplier

Overview

The multiplier block multiplies one input variable by another input variable.

The multiplier block is an active block and cannot be disabled.

When you define a multiplier CFB with the form in Figure 8-7, you have the following options:

- You can use the default names that appear in the form.
- You can add new names: declared variables or dot extensions from other blocks.

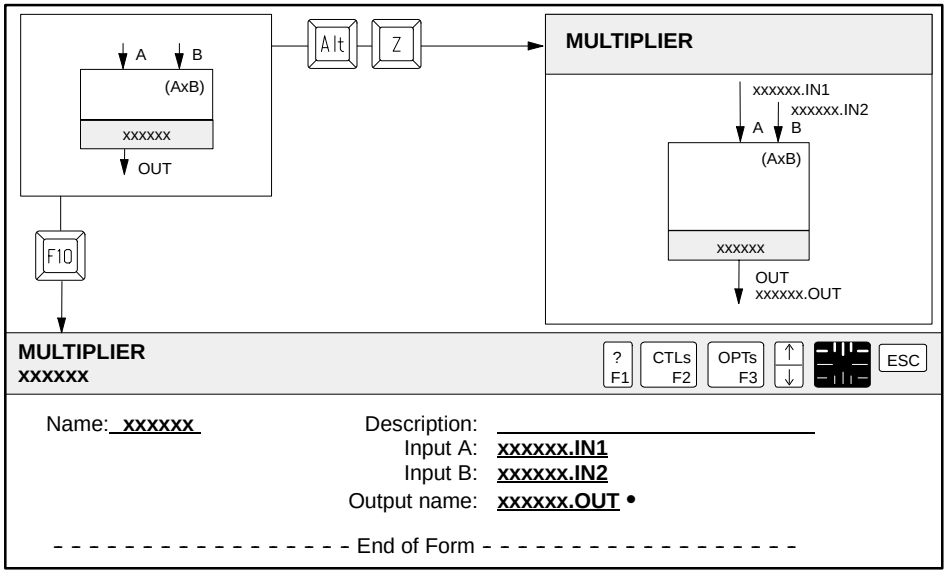


Figure 8-7 Multiplier Form

Input A: name of analog, integer, or real that is first input (A), which represents multiplicand; default is *cfb_name*.**IN1**.

Input B: name of analog, integer, or real that is second input (B), which represents multiplier; default is *cfb_name*.**IN2**.

Output name: name of analog, integer, or real output (OUT), which represents product of inputs; default is *cfb_name*.**OUT**.

Availability

The multiplier block is supported for Series 505 and S5 controllers, but not for the 560/560T controllers.

Extensions

Table 8-5 lists the extensions that you can use to monitor and control a multiplier block.

Table 8-5 Multiplier Extensions

Read-only Integer		Read-only Real	
.IOUT	integer output [1]	**.OUT**	real output [1]
Read/Write Integer		**Read/Write Real**	
.IIN1	integer input A [2]	**.IN1**	real input A [2]
.IIN2	integer input B [2]	**.IN2**	real input B [2]

1 For each output, APT creates **.IOUT** if output value is an integer or analog, or **.OUT** if value is a real number. The outputs are read-only variables if the block is enabled, and read/write when the block is disabled.

2 For each input, APT creates **.IIN** if input value is an integer, or **.IN** if value is a real number.

8.7 Square

Overview

The square block computes the square of a variable.

The square block is an active block and cannot be disabled.

When you define a square CFB with the form in Figure 8-8, you have the following options:

- You can use the default names that appear in the form.
- You can add new names: declared variables or dot extensions from other blocks.

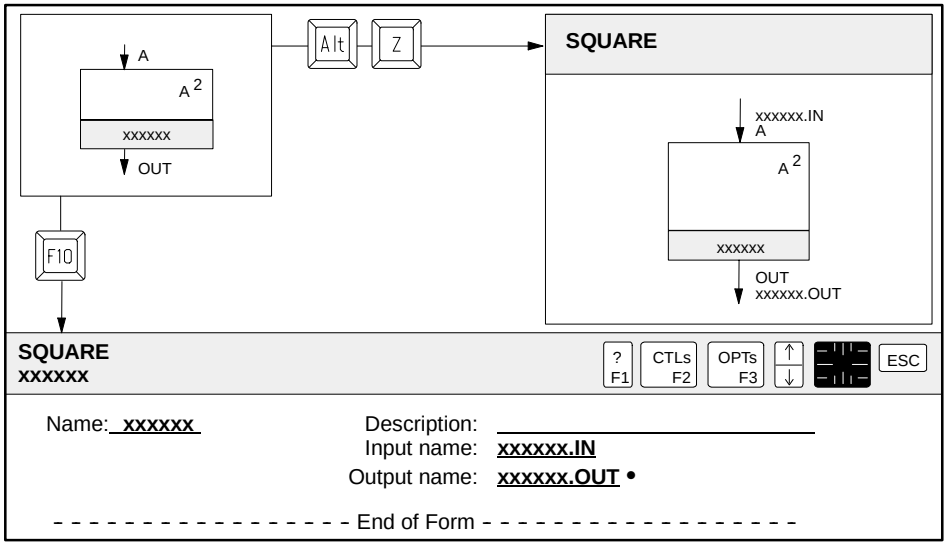


Figure 8-8 Square Form

Input name: name of analog, integer, or real input (A); default is *cfb_name*.**IN**.

Output name: name of analog, integer, or real output (OUT) that represents square of input; default is *cfb_name*.**OUT**.

Availability

The square block is supported for Series 505 and S5 controllers, but not for the 560/560T controllers.

Extensions

Table 8-6 lists the extensions that you can use to monitor and control a square block.

Table 8-6 Square Extensions

Read-only Integer		Read-only Real	
.IOUT	integer output [1]	.OUT	output [1]
Read/Write Integer		**Read/Write Real**	
.IIN	integer input [2]	.IN	real input [2]
1 For each output, APT creates **.IOUT** if output value is an integer or analog, or **.OUT** if value is a real number. The outputs are read-only variables if the block is enabled, and read/write when the block is disabled.			
2 For each input, APT creates **.IIN** if input value is an integer, or **.IN** if value is a real number.			

8.8 Square Root

Overview

The square root block computes the square root of a variable. The input must be a positive real number; otherwise,the output is invalid.

The square root block is an active block and cannot be disabled.

When you define a square root CFB with the form in Figure 8-9, you have the following options:

- You can use the default names that appear in the form.
- You can add new names: declared variables or dot extensions from other blocks.

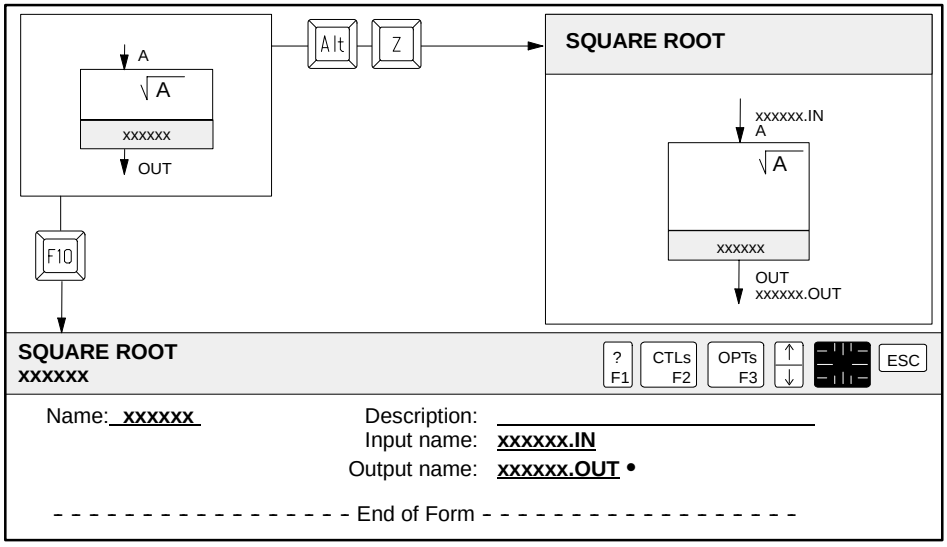


Figure 8-9 Square Root Form

Input name: name of analog, integer, or real input (A); default is *cfb_name*.**IN**.

Output name: name of analog, integer, or real output (OUT), which represents square root of the input; default is *cfb_name*.**OUT**.

Availability

The square root block is supported for Series 505 and S5 controllers, but not for the 560/560T controllers.

Extensions

Table 8-7 lists the extensions that you can use to monitor and control a square root block.

Table 8-7 Square Root Extensions

Read-only Integer		Read-only Real	
.IOUT	integer output [1]	.OUT	output [1]
Read/Write Integer		**Read/Write Real**	
.IIN	integer input [2]	.IN	real input [2]

1 For each output, APT creates **.IOUT** if output value is an integer or analog, or **.OUT** if value is a real number. The outputs are read-only variables if the block is enabled, and read/write when the block is disabled.

2 For each input, APT creates **.IIN** if input value is an integer, or **.IN** if value is a real number.

8.9 Subtractor

Overview

The subtractor block subtracts one input variable from another input variable.

The subtractor block is an active block and cannot be disabled.

When you define a subtractor CFB with the form in Figure 8-10, you have the following options:

- You can use the default names that appear in the form.
- You can add new names: declared variables or dot extensions from other blocks.

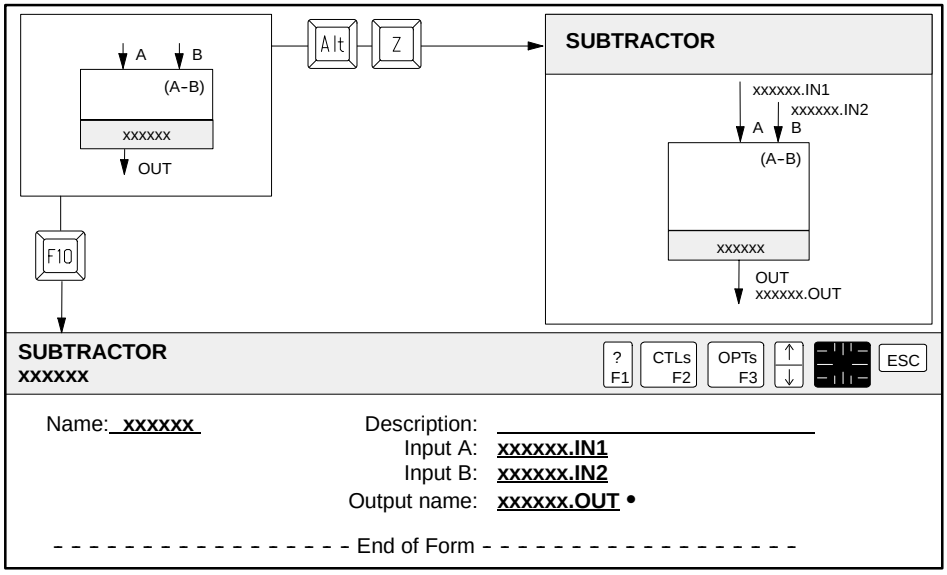


Figure 8-10 Subtractor Form

Input A: name of analog, integer, or real that is first input (A), which represents minuend; default is *cfb_name*.**IN1**.

Input B: name of analog, integer, or real that is second input (B), which represents subtrahend; default is *cfb_name*.**IN2**.

Output name: name of analog, integer, or real output (OUT), which represents difference of inputs; default is *cfb_name*.**OUT**.

Availability

The subtractor block is supported for Series 505 and S5 controllers, but not for the 560/560T controllers.

Extensions

Table 8-8 lists the extensions that you can use to monitor and control a subtractor block.

Table 8-8 Subtractor Extensions

Read-only Integer		Read-only Real	
.IOUT	integer output [1]	.OUT	output [1]
Read/Write Integer		**Read/Write Real**	
.IIN1	integer input A [2]	.IN1	real input A [2]
.IIN2	integer input B [2]	.IN2	real input B [2]

1 For each output, APT creates **.IOUT** if output value is an integer or analog, or **.OUT** if value is a real number. The outputs are read-only variables if the block is enabled, and read/write when the block is disabled.

2 For each input, APT creates **.IIN** if input value is an integer, or **.IN** if value is a real number.

8.10 Summer

Overview

The summer block adds one input variable to another input variable.

The summer block is an active block and cannot be disabled.

When you define a summer CFB with the form in Figure 8-11, you have the following options:

- You can use the default names that appear in the form.
- You can add new names: declared variables or dot extensions from other blocks.

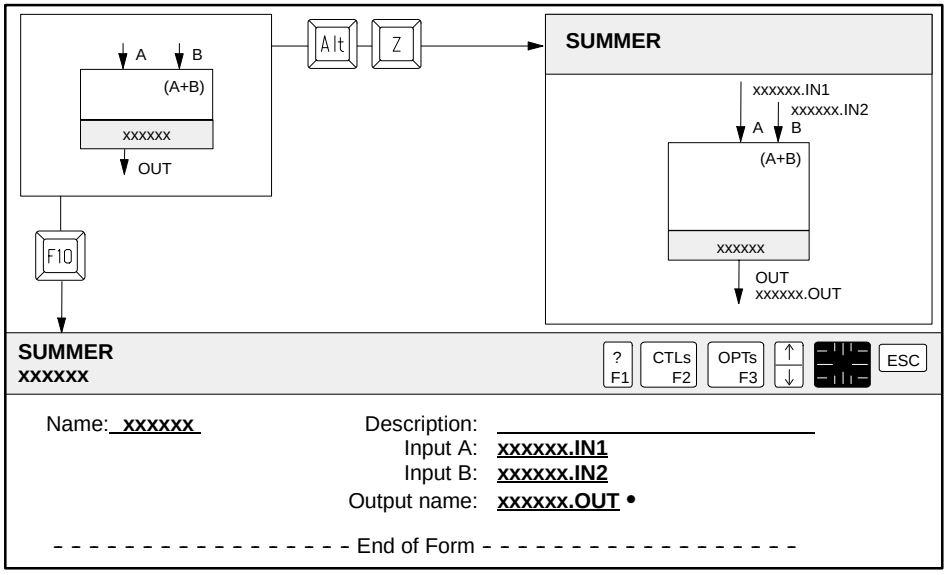


Figure 8-11 Summer Form

Input A: name of analog, integer, or real that is first input (A), which represents the first addend; default is *cfb_name*.**IN1**.

Input B: name of analog, integer, or real second input (B), which represents the second addend; default is *cfb_name*.**IN2**.

Output name: name of analog, integer, or real output (OUT), which represents the sum of the inputs; default is *cfb_name*.**OUT**.

Availability

The summer block is supported for Series 505 and S5 controllers, but not for the 560/560T controllers.

Extensions

Table 8-9 lists the extensions that you can use to monitor and control a summer block.

Table 8-9 Summer Extensions

Read-only Integer		Read-only Real	
.IOUT	integer output [1]	.OUT	output [1]
Read/Write Integer		**Read/Write Real**	
.IIN1	integer input A [2]	.IN1	real input A [2]
.IIN2	integer input B [2]	.IN2	real input B [2]

1 For each output, APT creates **.IOUT** if output value is an integer or analog, or **.OUT** if value is a real number. The outputs are read-only variables if the block is enabled, and read/write when the block is disabled.

2 For each input, APT creates **.IIN** if input value is an integer, or **.IN** if value is a real number.

Chapter 9

Other Control Blocks

9.1 Understanding Other Control Blocks

Overview

APT provides several other functions that are available as continuous function blocks.

- Anti-reset windup protection blocks allow you to monitor a variable and provide reset windup protection in addition to that provided by the PID block.
- The correlated lookup table determines an output value based on the relationship between an input and two 11-element arrays.
- The scale block converts an input value to an output that is within the specified range.

Availability

All of the functions above are supported for Series 505 and S5 controllers. None of the functions listed above are supported for the 560/560T controllers.

Enabling and Disabling Blocks

These CFBs must be activated with the ENABLE command, or with an assignment statement that sets the **.ENABL** extension to true. See Figure 9-1. These blocks can also be enabled from an assignment statement in a math CFB.

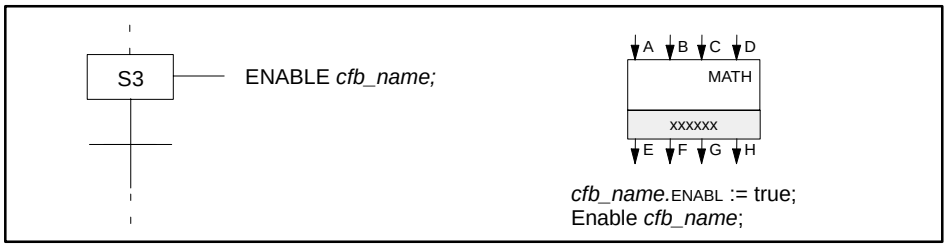


Figure 9-1 Enabling a CFB

To deactivate the block, use the DISABLE command with the block name, or set the **.ENABL** extension to false.

These control blocks have a boolean extension (**.NRDY**) that you can use to interlock the execution of the block to some external conditions.

- If you set the **.NRDY** extension to true, the block is forced to the disabled state.
- To return the block to the enabled state you must not only set the **.NRDY** extension to false, but you must also re-enable the block with a command or an assignment statement.

9.2 Anti-Reset Windup Protection/Constraint Type

Overview

The anti-reset windup protection/constraint type block protects the integral mode of a PID loop from windup by ensuring that a monitored variable remains within specified limits.

This block can be used when constraints within the control loop can be reached without being detected by the anti-reset windup protection that is internal to the PID block. This block is useful when you have two cascaded loops and the output of the source loop is greater than the output of the second loop.

The input to the block is the analog or real value that you want to monitor.

When the protected PID is in automatic or cascade mode and the constraint block is enabled, the block compares the monitored variable (input) with the high and low limits that you specify when you define the constraint block.

The block freezes the bias (the integral term) of the protected PID as follows:

- If the monitored variable is within the limits, the current value of the **.BIAS** (bias) extension is stored internally; and the **.AWS** (anti-windup status) extension is set to 0 to indicate normal loop operation.
- If the monitored variable goes below the value of the **.LLIM** (low limit) extension while the loop is still trying to decrease the output, the constraint block freezes the bias at the last recorded value; and the **.AWS** extension is set to 1.
- If the monitored variable exceeds the value of the **.HLIM** (high limit) extension while the loop is still trying to increase the output, the constraint block freezes the bias at the last recorded value before the constraint was exceeded; and the **.AWS** extension is set to 2.
- To stop the integral mode from operating, assign 3 to the **.AWS** extension.

NOTE: If the PID loop executes faster than the anti-reset windup protection/constraint CFB, then the CFB cannot protect the loop from windup. In this case, you must write code in the associated math section of the PID to provide the windup protection.

Availability

The anti-reset windup protection/constraint type block is supported for Series 505 and S5 controllers, but not for the 560/560T controllers.

Anti-Reset Windup Protection/Constraint (continued)

Block Configuration

Figure 9-2 shows the form that you use to define the constraint type of anti-reset windup protection.

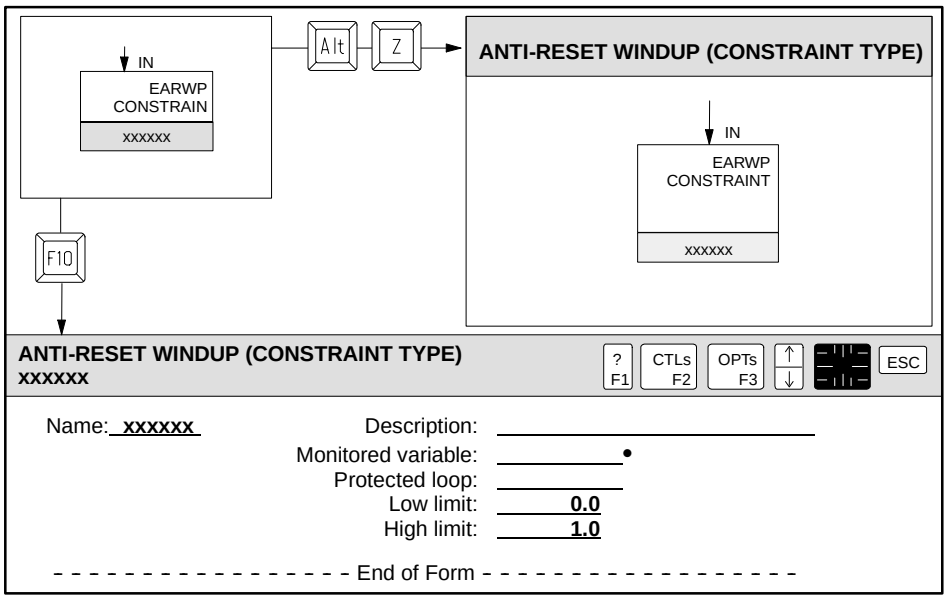


Figure 9-2 Anti-Reset Windup (Constraint Type) Form

Name: unique name that identifies block (eight characters maximum); default is name that you entered when you created the CFB graphic.

Description: 30 characters maximum (optional)

Monitored variable: name of analog or real input that is monitored variable (**.IN**).

Protected loop: name of PID loop that is protected by this block.

Low limit: real number below which the monitored variable must not fall.

High limit: real number above which the monitored variable must not rise.

Commands and Extensions

Table 9-1 lists the commands that determine the mode of operation for this anti-reset windup block.

Table 9-1 Anti-Reset Windup (Constraint Type) Commands

Command	Description
ENABLE (ENA)	Turns on, or activates, the block.
DISABLE (DIS)	Turns off, or deactivates, the block.

Table 9-2 lists the extensions that you can use to monitor and control this anti-reset windup block.

Table 9-2 Anti-Reset Windup (Constraint Type) Extensions

Read-only Boolean		Read-only Integer		Read-only Real	
		.SNUM	error line number [3]		
		.ECODE	error code [3]		
		.IID	error controller block [3]		
Read/Write Boolean		**Read/Write Integer**		**Read/Write Real**	
.ENABL	enable [1]			.IN	real input
.NRDY	not ready			.HLIM	high limit
.REN	request enable [2]			.LLIM	low limit
.RDIS	request disable [2]				

1 This extension is manipulated by the command(s) associated with the block.

2 These booleans are APT flags that are manipulated by the Math procedures Latch, Clear, and On.

3 These extensions do not contain the error codes for the S5 controllers. (Series 505-only error codes.)

9.3 Anti-Reset Windup Protection/Select Type

Overview

The anti-reset windup protection/select type block protects the integral mode of a PID loop from windup by ensuring that the output of the PID loop remains equal to the value of a monitored variable.

This block can be used when a PID loop does not have a direct path to the controlled variable. For example, if you have a selector block monitoring the output of two PID blocks, you can use two select type anti-reset windup blocks to compare the output of the selector block with each PID output.

The input to the block is the analog or real value that you want to monitor.

When the protected PID loop is in automatic or cascade mode and the select block is enabled, the block compares the monitored variable (input) to the output of the loop that you specify.

The block freezes the bias (the integral term) of the protected PID as follows:

- If the monitored variable and output do agree, the current value of the **.BIAS** (bias) extension is stored internally; and the **.AWS** (anti-windup status) extension is set to 0 to indicate normal loop operation.
- If the monitored variable and output do not agree, the select block sets the bias equal to the monitored variable; and the **.AWS** extension is set to 3.

NOTE: If the PID loop executes faster than the anti-reset windup protection/select CFB, then the CFB cannot protect the loop from windup. In this case, you must write code in the associated math section of the PID to provide the windup protection.

Availability

The anti-reset windup protection/select type block is supported for Series 505 and S5 controllers, but not for the 560/560T controllers.

Block Configuration

Figure 9-3 shows the form that you use to define the select type of anti-reset windup protection.

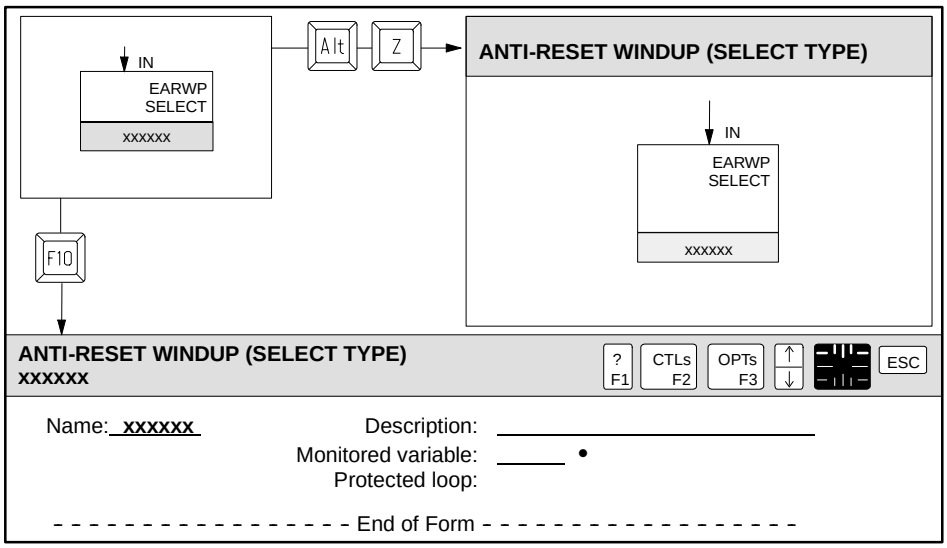


Figure 9-3 Anti-Reset Windup (Select Type) Form

Name: unique name that identifies block (eight characters maximum); default is name that you entered when you created the CFB graphic.

Description: 30 characters maximum (optional)

Monitored variable: name of analog, integer, or real input that is monitored input (IN).

Protected loop: name of PID loop that is protected by this block.

Anti-Reset Windup Protection/Select Type (continued)

Commands and Extensions

Table 9-3 lists the commands that determine the mode of operation for this anti-reset windup block.

Table 9-3 Anti-Reset Windup (Select Type) Commands

Command	Description
ENABLE (ENA)	Turns on, or activates, the block.
DISABLE (DIS)	Turns off, or deactivates, the block.

Table 9-4 lists the extensions that you can use to monitor and control this anti-reset windup block.

Table 9-4 Anti-Reset Windup (Select Type) Extensions

Read-only Boolean		Read-only Integer		Read-only Real	
		.SNUM	error line number [3]		
		.ECODE	error code [3]		
		.IID	error controller block [3]		
Read/Write Boolean		**Read/Write Integer**		**Read/Write Real**	
.ENABL	enable [1]			.IN	real input
.NRDY	not ready				
.REN	request enable [2]				
.RDIS	request disable [2]				

1 This extension is manipulated by the command(s) associated with the block.

2 These booleans are APT flags that are manipulated by the Math procedures Latch, Clear, and On.

3 These extensions do not contain the error codes for the S5 controllers. (Series 505-only error codes.)

9.4 Correlated Lookup Table

Overview

The correlated lookup table block determines an output value based on the relationship between an input value and a specified range of values.

- If the input is an integer, APT sends the appropriate value to the output.
- If the input is a real number and the output is an analog value, APT assumes that the input is limited to the range of 0.0 to 1.0 and automatically scales the output to the correct format (zero offset, 20% bias, or bipolar). If an input is outside this range, the output is set to the maximum or minimum value of the subtype.
- The input and output arrays must contain 11 elements. The values in the input array must be arranged in ascending order.

Availability

The correlated lookup table type block is supported for Series 505 and S5 controllers, but not for the 560/560T controllers.

Block Configuration

You can select a type of either lookup or interpolation.

If you select **Lookup**, the block compares the input value to the values in the input array, in sequence. The block determines the value of the output with one of the following techniques:

- If the input value is equal to the value of an element of the input array, the output is assigned the value of the same element number in the output array.
- If the input value is not equal to the value of an element of the input array, the block uses the value of the highest element of the input array that is not greater than the input value. The output is assigned the value of the same element number in the output array.
- If the input value is less than the value of the first element of the input array, the output is assigned the value of the first element of the output array.
- If the input value is greater than the value of any element of the input array, the output is assigned the value of the last element of the output array.

Correlated Lookup Table (continued)

If you select **Interpolation**, the block first compares the input value to the values in the input array. The block determines the value of the output with one of the following techniques:

- If the input value is equal to the value of an element of the input array, the output is assigned the value of the same element number in the output array.
- If the input value falls between the values of two elements of the input array, the output is assigned a value between the values of the same two element numbers in the output array. The value assigned to the output is interpolated between the values of the elements of the output array using the linear relationship calculated between the input value and the values of the elements of the input array.
- If the input value is less than the value of the first element of the input array, the output is assigned a value less than the value of the first element of the output array. The value assigned to the output is extrapolated from the values of the first two elements of the output array using the linear relationship calculated between the input value and the values of the first two elements of the input array.
- If the input value is greater than the value of the last element of the input array, the output is assigned a value greater than the value of the last element of the output array. The value assigned to the output is extrapolated from the values of the last two elements of the output array using the linear relationship calculated between the input value and the values of the last two elements of the input array.

When you define a correlated lookup table CFB with the form that is shown in Figure 9-4, you have the following options:

- You can use the default names that appear in the form.
- You can add new names: declared variables or dot extensions from other blocks.

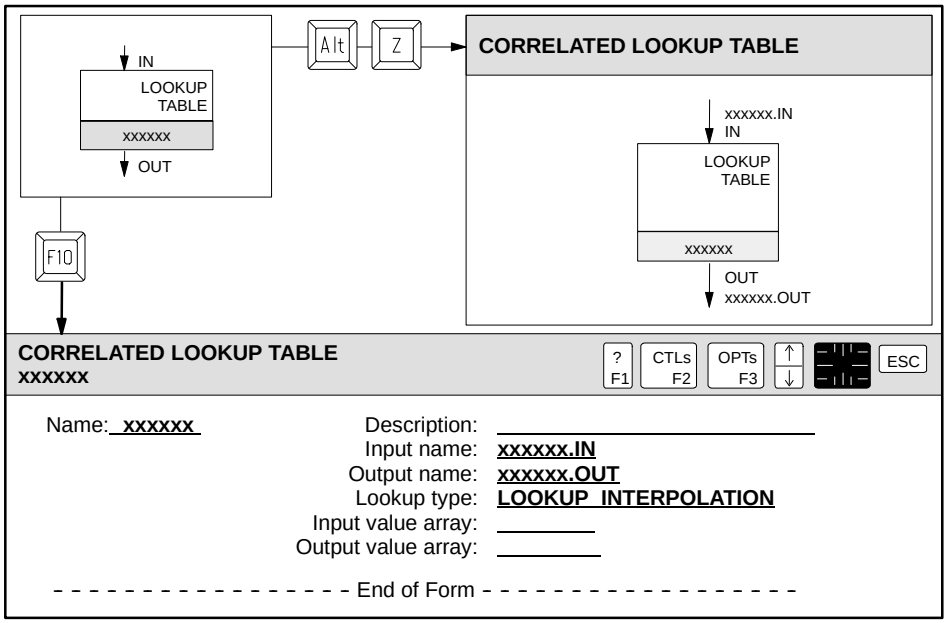


Figure 9-4 Correlated Lookup Table Form

Input name: name of real or integer input (IN); default is *cfb_name*.**IN**.

Output name: name of analog, real, or integer output (OUT); default is *cfb_name*.**OUT**.

Lookup type: LOOKUP or INTERPOLATION; default is LOOKUP.

Input value array: name of input value array, which must have a type of real. The array must contain 11 elements.

Output value array: name of output value array, which must have a type of real. The array must contain 11 elements.

Correlated Lookup Table (continued)

Commands and Extensions

Table 9-5 lists the commands that determine the mode of operation for this correlated lookup table block.

Table 9-5 Correlated Lookup Table Commands

Command	Description
ENABLE (ENA)	Turns on, or activates, the block.
DISABLE (DIS)	Turns off, or deactivates, the block.

Table 9-6 lists the extensions that you can use to monitor and control this correlated lookup table block.

Table 9-6 Correlated Lookup Table Extensions

Read-only Boolean		Read-only Integer		Read-only Real	
.ENABLD	enabled	.IOUT	integer output [3]	.OUT	real output [3]
		.ECODE	error code [5]		
		.IID	error controller block [5]		
		.SNUM	error line number [5]		
		.POS	position where input ≥ array element & ≤ next array element		
Read/Write Boolean		**Read/Write Integer**		**Read/Write Real**	
.ENABL	enable [1]	.IIN	integer input [4]	.IN	real input [4]
.NRDY	not ready				
.REN	request enable [2]				
.RDIS	request disable [2]				

1 This extension is manipulated by the command(s) associated with the block.

2 These booleans are APT flags that are manipulated by the Math procedures Latch, Clear, and On.

3 For each output, APT creates **.IOUT** if output is an integer or **.OUT** if value is a real number.

4 For each input APT creates **.IIN** if input is an integer or **.IN** if value is a real number.

5 These extensions do not contain the error codes for the S5 controllers. (Series 505-only error codes.)

9.5 Scale

Overview

The scale block transforms an input value to an output value by scaling the value between specified input and output ranges.

- If the input is an integer, APT sends the appropriate value to the output.
- If the input is a real number and the output is an analog value, APT assumes that the input is limited to the input range and automatically scales the output to the correct format (zero offset, 20% bias, or bipolar). If an input is outside this range, the output is set to the maximum or minimum value of the subtype.

The scale block allows you to define an input range and an output range. The block converts an input value to an output value by using the formula:

$$output = \frac{(input - input\ low\ range)}{(input\ high\ range - input\ low\ range)} * X + output\ low\ range$$

$$where\ X = output\ high\ range - output\ low\ range$$

The initial values of the high and low ranges are defined in the form and stored in extension variables that can be changed from the program.

A scale block can be used to change a 0.0 to 1.0 range signal to a 1.0 to 0.0 signal. For example, if you want to use the output from a block to drive a reverse-acting control valve, you can use a scale block to provide the correct inversion. For reverse-acting block execution, set the values in the scale form for the input to range from 1 to 0, and the values for the output to range from 0 to 1.

For Series 505 controllers, the analog output always ranges from 0 to 32000 (zero offset), from 6400 to 32000 (20% bias), or from –32000 to +32000 (bipolar). For S5 controllers, the analog output ranges are from 0 to 1024 (zero offset), or from –1024 to 1024 (bipolar).

Availability

The scale block is supported for Series 505 and S5 controllers, but not for the 560/560T controllers.

Block Configuration

When you define a scale CFB with the form that is shown in Figure 9-5, you have the following options:

- You can use the default names that appear in the form.
- You can add new names: declared variables or dot extensions from other blocks.

Scale (continued)

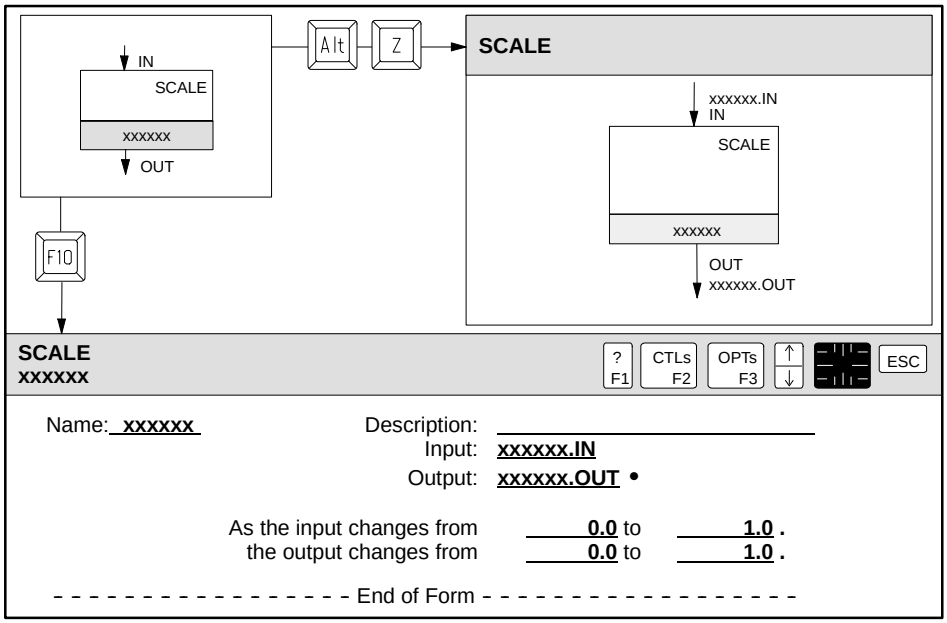


Figure 9-5 Scale Form

Input: name of real or integer input (IN); default is *cfb_name*.**IN**.

Output: name of analog, real, or integer output (OUT); default is *cfb_name*.**OUT**).

As the input changes from _____ to _____. Enter the low and high input ranges; default of low input range is zero; default of high input range is one.

the output changes from _____ to _____. Enter the low and high output ranges; default of low output range is zero; default of high output range is one.

Commands and Extensions

Table 9-7 lists the commands that determine the mode of operation for this scale block.

Table 9-7 Scale Commands

Command	Description
ENABLE (ENA)	Turns on, or activates, the block.
DISABLE (DIS)	Turns off, or deactivates, the block.

Table 9-8 lists the extensions that you can use to monitor and control this scale block.

Table 9-8 Scale Extensions

Read-only Boolean		Read-only Integer		Read-only Real	
.ENABLD	enabled	.IOUT	integer output [3]	.OUT	real output [3]
		.ECODE	error code [4]		
		.IID	error controller block [4]		
		.SNUM	error line number [4]		
Read/Write Boolean		**Read/Write Integer**		**Read/Write Real**	
.ENABL	enable [1]	.IIN	integer input [5]	.IN	real input [5]
.NRDY	not ready			.HRIN	input high range
.REN	request enable [2]			.LRIN	input low range
.RDIS	request disable [2]			.HROUT	output high range
				.LROUT	output low range

1 This extension is manipulated by the command(s) associated with the block.

2 These booleans are APT flags that are manipulated by the Math procedures Latch, Clear, and On.

3 For each output, APT creates **.IOUT** if output is an integer or **.OUT** if value is a real number.

4 These extensions do not contain the error codes for the S5 controllers. (Series 505-only error codes)

5 For each input, APT creates **.IIN** if input is an integer or **.IN** if value is a real number.

Chapter 10

Math Language Overview

10.1 Understanding the Math Language

Math Blocks

The Math Language is a text-based programming language that you use to perform arithmetic calculations and logical operations. A sequence of Math Language statements is called a “math block.” You can use math blocks in two different areas of APT:

- A math block can appear as the math section of a step in a Sequential Function Chart (SFC). See Figure 10-1.
- A math block can appear in a math or interlock CFB in a Continuous Function Chart (CFC). See Figure 10-2.

A math block in a step of an SFC executes repeatedly while the step is active. However, you can use the INIT and BODY commands, described in Section 10.5, to control the execution of portions of the math block.

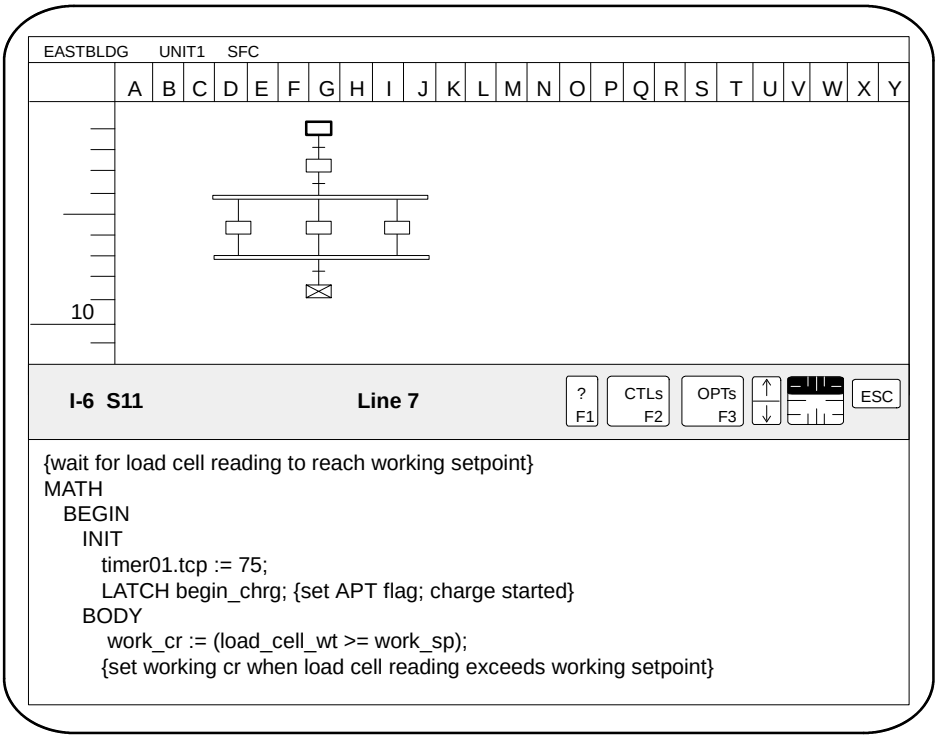


Figure 10-1 Math Section in an SFC

Figure 10-2 shows a math block in a Continuous Function Chart. Since this is a math CFB, you do not have to use the MATH statement to introduce the section of math code. You can still use the INIT and BODY statements, if appropriate, to control the execution of portions of the math block.

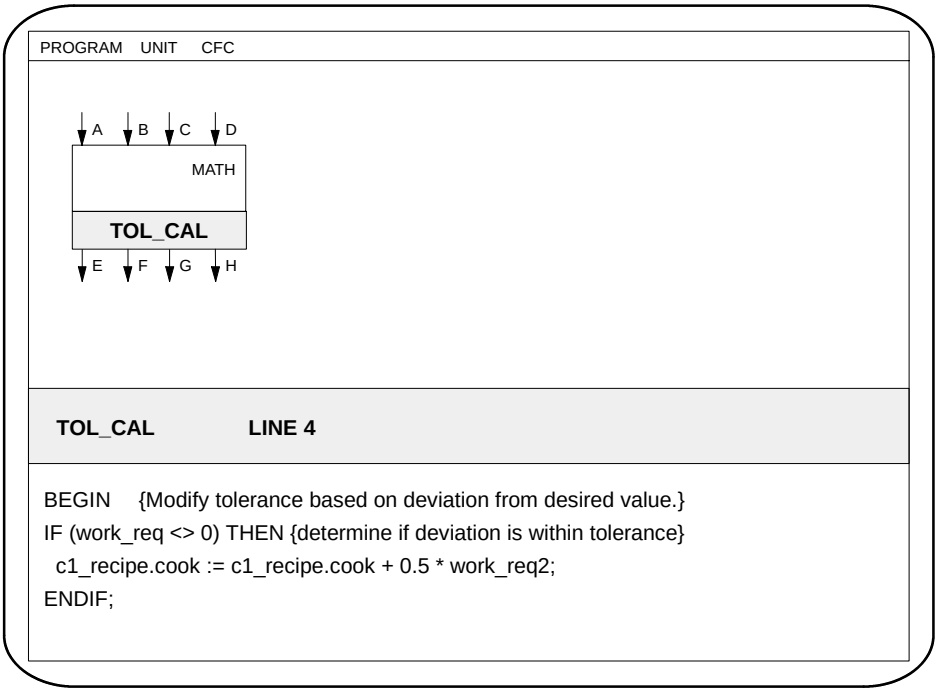


Figure 10-2 Math Block in a CFC

Types of Math Block Code

For S5 controllers, APT compiles all math blocks into STL (Statement List) code. However, for Series 505 controllers, math blocks can be compiled into either RLL (Relay Ladder Logic) code or SFPGM (Special Function Program) code. Certain Math Language statements can be compiled only into RLL code; other statements can be compiled only into SFPGM code; and some are non-exclusive and can be compiled into either type of code.

For Series 505 controllers, the Special Functions processor executes floating-point math calculations using SFPGM code, while RLL is executed separately. The 560/560T controllers do not have SFPGM code and consequently cannot perform SFPGM-type operations.

Although S5 controllers do not use SFPGM code, they do have SFPGM functionality; they perform both floating-point math and RLL calculations in STL in the cyclic scan.

If you have a Series 505 controller, bear in mind that you cannot mix RLL-only statements and SFPGM-only statements in the same math block. For example, a math block that contains an RLL-only statement cannot also contain an SFPGM-only statement. You can include non-exclusive statements in any math block. Use the PRAGMA statement, described later in this chapter, when you intend a math block to be only RLL or only SFPGM code. Table 11-1 and Table 11-2 in Chapter 11 list the math procedures and functions and identify whether they are for RLL only, SFPGM only, or both.

Guidelines for Series 505 Controllers

RLL code executes faster than SFPGM code; therefore, if you need to ensure the fastest possible execution, follow these conventions in writing math blocks for a Series 505 controller:

- Avoid functions and procedures that are SFPGM only.
- Avoid variables or expressions in integer or real arrays unless the controller release supports this in RLL: see the table on "APT Features Supported by Series 505 Controller Models" in the "APT Overview" chapter of the *SIMATIC APT Programming Reference (Tables) Manual*. The following are examples of each.

  ```
  array1[pointer]
  array7[ptr + 1]
  ```

- Use boolean values and integers in expressions. The following generate RLL code.

  ```
  bool1 OR bool2;
  int1 := 10 + int2;
  ```

- Avoid arithmetic operations and expressions with real values. The following generate SFPGM code and should be avoided.

  ```
  real1 > 2.0 * real2;
  real1 := real2 * 2.0;
  ```

- Use real values only in relational operations and in simple assignments. The following generate RLL code.

  ```
  real1 > 2.0;
  real1 := real2;
  real1 := 0.0;
  ```

- Avoid the use of temporary variables, which are explained in Section 10.3.

Key Words Used by APT

The Math Language described in this chapter includes key words APT defines for one and only one purpose.

Data types are declared in a math block by using key words such as BOOLEAN, INTEGER, REAL.

Delimiters indicate sections of a math block with key words, such as BEGIN, INIT, BODY. The key word MATH marks the beginning of the math section in an SFC step.

Conditional statements use the following types of key words: IF, THEN, ELSE, ELSIF, ENDIF.

Math Language functions perform a wide range of arithmetic (real and integer), trigonometric, and boolean operations. The key words defined by APT to identify these functions are listed in Table 11-2.

Math Language procedures perform specific tasks as defined by APT. The key words defined by APT to identify these procedures are listed in Table 11-1.

In the Math Language examples, key words appear in uppercase letters.

10.2 Identifiers

Overview

APT provides three basic types of identifiers.

User-defined identifiers that you create in one of the Definition Tables or inside a math block.

System-defined extensions that APT uses to access specific memory locations.

User-defined extensions that you create in a recipe.

You can enter identifiers in any combination of uppercase and lowercase letters. APT recognizes these as the same word or variable regardless of the case.

User-defined Identifiers

When you name a program, unit, SFC, or CFC, you are creating a user-defined identifier. Identifiers that you use in a math block are created in one of the following ways:

- You can use the Declaration Editor to declare APT flags, constants, variables, arrays, timers, and counters that you want to have available at the program or unit level.
- You can use the I/O Symbolic Name Table, Device Definition Table, Recipe Usage Table, and Continuous Function Chart (CFC) Editor to name objects in APT.
- You can use the Math Language declaration commands (BOOLEAN, REAL, INTEGER, TIMER, and ARRAY) to declare constants and variables that can be accessed only within the math block.

In the Math language examples in this manual, user-defined identifiers appear in lowercase letters.

NOTE: Although it is not necessary to declare a variable or a constant before you use it in a math block, you must declare all user-defined identifiers before you compile a program.

System-defined Extensions

APT defines extensions that are associated with APT objects: programs, units, devices, I/O points, and CFBs. You can use the names of these objects with an extension as identifiers in the Math Language. For a list of extensions associated with objects see the chapter containing information about the object.

In the Math Language examples, system-defined extensions appear in uppercase letters preceded by a period (.). See Figure 10-3.

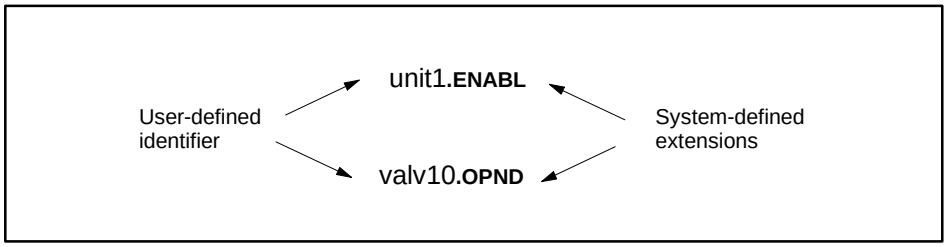


Figure 10-3 Identifiers and Extensions

User-defined Extensions

APT allows you to create user-defined extensions in a recipe. You can use these extensions with recipe variables in the Math Language. See the "Recipes" Chapter in the *SIMATIC APT Programming Reference (Tables) Manual* for information about recipes and user-defined extensions.

10.3 Direct Memory Addressing

Overview

APT allows you to access memory addresses directly by using a percent sign (%) prefix. The types of direct memory addressing available depend on the kind of controller you have.

Reserved memory (S5 and Series 505) Allows you to use memory locations that you reserve before compiling the program.

Status word memory (Series 505 only) Allows you to read status words that contain operational information about your Series 505 controller.

System Data Area (RS) memory (S5 only) Allows you to read the internal system parameters that contain operational information about your S5 controller.

Temporary (SFPGM) memory variables (Series 505 only) Allow you to use memory space only while a math block is executing.

Series 505 controllers can use direct memory addresses in the Math language. S5 controllers cannot, except in the IN_ASM math statements, which are described in Appendix C, "Inline Assembly Code." See Appendix B of this manual for a discussion of how direct memory addressing works in APT for Series 505 and S5 controllers.

Instead of referencing a direct address, consider making a declaration ("APT Declarations" chapter, *SIMATIC APT Programming Reference (Tables) Manual*) and marking the **User Assigned Address** option.

10.4 Data Types and Arithmetic Operations

Overview

The Math Language is a “strongly typed” language; that is, you must declare the type of each constant and variable in the program. The following are valid numeric data types in the Math Language: INTEGER, REAL (floating point), and BOOLEAN. See the “APT Declarations” chapter in the *SIMATIC APT Programming Reference (Tables) Manual* for an explanation of the naming conventions for constants and variables.

The procedures and functions described in this chapter accept only certain types of values as parameters; specifying a value with an invalid type causes an error.

Expressions

An expression is a valid combination of terms (constants or variables) and operators (arithmetic, relational, or logical) that can be evaluated to produce a result.

You can use parentheses to group operations within an expression for greater readability, and to specify a desired order of operation.

Expressions that include parentheses are evaluated beginning with the operation enclosed in the innermost parentheses and working left to right and outward. The power operator, the unary minus and the unary plus are evaluated from right to left. The order of operation is determined by the precedence level of the operator as listed in Table 10-1. Operators of equal precedence are evaluated from left to right.

Table 10-1 Precedence Level of Operators

Precedence Level	Operator(s)
1	( )
2	unary –, unary +, NOT
3	** (power)
4	*, /, MOD,
5	+, -
6	<, <=, >, >=, =, <>
7	AND
8	XOR
9	OR
10	:= (assignment)

Integers

An integer is composed of one or more digits (0–9). An integer cannot contain a decimal point. The valid range for an integer is –32768 to 32767. Integers can be expressed in any of the following ways:

- An integer value with no decimal point: 29994
- A binary (base 2) value preceded by the base number enclosed in pound signs (#): #2#101 (101_2=5)
- A hexadecimal (base 16) value preceded by the base number enclosed in pound signs (#): #16#752A ($752A_{16}$) = 29994

Arithmetic expressions using integers produce a numeric result that is also an integer. All values in the expression must be integers.

The integer operators available in APT are listed in Table 10-2.

Table 10-2 Integer Operators

Operator	Operation	Precedence Level	Example	Numeric Result
*	Multiplication	4	4 * 6	24
/	Division [1]	4	3 / 2	1
MOD	Modulo	4	27 MOD 12	3
+	Addition	5	4 + 6	10
–	Subtraction	5	4 – 6	–2
AND	Logical AND	7	#2#1100 AND #2#0101	0100_2
XOR	Exclusive OR	8	#2#1100 XOR #2#0101	1001_2
OR	Logical OR	9	#2#1100 OR #2#0101	1101_2

1 Division with integers results in a truncated value.

NOTE: All integer math uses 16-bit words with 16-bit word intermediate results, regardless of whether you have a Series 505 or an S5 controller.

Real Numbers

A real (floating point) number is a numeric value that includes a decimal point and is raised to a specified power of 10.

The valid range for a real number is dependent on controller type. For a Series 505, the range is -9.223372 E^{+18} to +9.223372 E^{+18}, and any value except 0.0 that falls between -2.710501 E^{-20} and +5.421011 E^{-20} generates a controller error. For an S5 controller, the valid range is -1.701412E^{+38} to +1.701412E^{+38}, and any value except 0.0 that falls between -1.469368E^{-39} to +1.469368E^{-39} generates a controller error.

A real number can be expressed in any of four ways but must always include a decimal point; it does not need to include the fractional portion. If the sign is omitted, it is assumed to be a positive sign (+).

- A value with a decimal point and the fractional portion followed by the letter E and an exponent: 12.0E-3 (12.0×10^{-3})
- A value with a decimal point followed by the letter E and an exponent: 12.E-3 (12.0×10^{-3})
- A value with a decimal point and the fractional portion: 12.0
- A value with a decimal point: 12.

Arithmetic expressions using real numbers produce a numeric result that is also real. All values in the expression must be real numbers.

Table 10-3 lists the operators available in APT to be used with real numbers; they are supported by all controllers except the 560/560T controllers.

Table 10-3 Real Operators

Operator	Operation	Precedence Level	Example	Numeric Result
**	Power	3	6.5 ** 2.0	42.25
*	Multiplication	4	4. * 6.	24.
/	Division	4	3.0 / 2.0	1.5
+	Addition	5	6.5 + 8.	14.5
-	Subtraction	5	6.5 - 8.0	-1.5

Boolean Values

There are only two boolean values.

- TRUE is equivalent to 1 or on.
- FALSE is equivalent to 0 or off.

A boolean expression produces a result that is either true or false. All values in a boolean expression must be of the same type.

Boolean expressions can include both relational and logical operators. The relational operators in Table 10-4 all have equal precedence.

Table 10-4 Relational Operators

Operator	Operation	Precedence Level	Example
<	Less than	6	int1 < 13
<=	Less than or equal to	6	real1 <= 14.5
>	Greater than	6	real2 > 11.44
>=	Greater than or equal to	6	int2 >= 15
=	Equal to	6	int3 = 14
< >	Not equal to	6	real3 < > 14.0

The logical operators are listed in Table 10-5. Expressions used with the logical operators must be enclosed in parentheses as shown in the examples.

Table 10-5 Logical Operators

Operator	Precedence Level	Example	Evaluates to True if
NOT	2	NOT mcv_1.OPND NOT (X=2)	term is False term is False
AND	4	mcv_1.CLSD AND mcv_2.CLSD	all terms are True
OR	5	mcv_3.OPND OR (pt101 < 14.9)	any term is True

10.5 Math Block Structure

Overview

A math block consists of a code specifier section, a declaration section, and an executable section. The code specifier and declaration sections are optional.

- The word MATH marks the beginning of the math section in an SFC step. The key word MATH is not used in a CFB math block or a subroutine.
- For Series 505 controllers, you can use the optional code specifier section to indicate certain controller-dependent information about how your math block compiles. The word PRAGMA indicates the beginning of this section.
- If you have a declaration section, each statement begins with one of the following terms: BOOLEAN, INTEGER, REAL, TIMER, or ARRAY.
- The word BEGIN marks the end of the declaration section and the beginning of the executable section in the math block. The only statements that can precede a BEGIN statement are declaration statements and the code specifier section.
- The executable section of a math block can contain an initialization portion and a body portion:

  When a math block has both an initialization section and a body portion, use the INIT and BODY statements to delineate the two sections of code.

  When a math block has an initialization portion, but no body portion, do not use the BODY statement.

  When a math block has a body portion, but no initialization portion, do not use either the INIT or the BODY statements.

Figure 10-4 shows the structure of a math block.

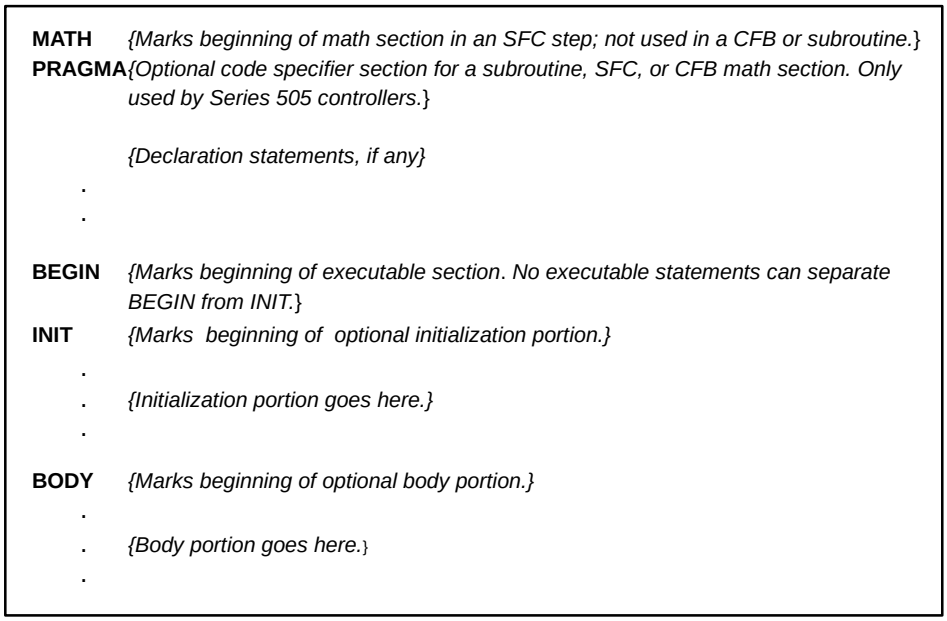


Figure 10-4 The Structure of a Math Block

Code Specifier Section (Series 505 only)

The code specifier section of a math block is an optional section that begins with the word PRAGMA. S5 controllers do not use the PRAGMA statement.

For a Series 505 controller, you can use PRAGMA to specify whether you want the math to be compiled exclusively in either RLL, RLL cyclic, RLL interrupt, or SFPGM code. The use of the PRAGMA statement does not force the type of compile; you must still follow the conventions described on page 10-4 to determine how the code is actually compiled. However, to alert you, PRAGMA causes a compile error to be generated if you inadvertently use SFPGM code in math that you intend to be RLL only, or vice versa.

If you use PRAGMA for a Series 505 controller, the code specifier section must precede the declaration section for subroutines, SFC or CFB math sections (Figure 10-4).

Use the following format for the PRAGMA statement:

PRAGMA <*type*> ;

The type can be **RLL**, **SF**, **RLL_CYCLIC**, or **RLL_INTERRUPT**. The types are described in Table 10-6.

Table 10-6 Controller-Dependent Code Specifications

Controller	Type	Meaning
Series 505	RLL	Code goes in main RLL section.
Series 505	SF	Code goes in SFPGM section.
Series 505	RLL_CYCLIC *	Code goes in cyclic RLL section; the Compiler Control File has a field for cyclic scan rate.
Series 505	RLL_INTERRUPT *	Code goes in interrupt RLL section; special modules can generate that interrupt.
* Neither RLL cyclic nor RLL interrupt math sections support APT flag procedures (ON, LATCH, CLEAR) or object commands (Open, Start . . .).		

Declaration Section

In the declaration section of a math block you can specify five types of variables: boolean, integer, real, array, and timer. These variables are local and can be referenced only within the math block. Local math block variables retain their values from execution to execution.

If present, the declaration section must precede the BEGIN statement as shown in Figure 10-5.

NOTE: APT flags must be declared with the Declaration Editor; you cannot declare APT flags as local variables in a math block.

Figure 10-5 gives an example of a declaration section for a math block.

```
{Declaration section}

{Statements each declare one variable as the specified type.}
BOOLEAN: bool1;
INTEGER: int1;
REAL: r1;
BOOLEAN RETENTIVE: boolr1;
TIMER FAST: tmr1;
ARRAY (1..19) OF REAL: array1;

{Statements each declare a list of variables as the specified type.}
BOOLEAN: bool2, bool3;
INTEGER: int2, int3, int4;
REAL: length, width, area;
BOOLEAN RETENTIVE: boolr2, boolr3;
TIMER FAST: tmr1, tmr2;
ARRAY (1..27) OF BOOLEAN: array2, array3;

{Statements each declare variable(s) as the  specified type and assign an initial value to each
variable in the list.}
BOOLEAN: bool4 := TRUE;
INTEGER: int5, int6 := 5;
REAL: r2,r3,r4 := 6.0;
BOOLEAN RETENTIVE: boolr4, boolr5:= FALSE;
TIMER SLOW: tmr2, tmr3:=0,10,true,false ;
ARRAY (1..11) OF INTEGER: array4, array5:= 75;

BEGIN       {Reserved word marks end of Declaration section and beginning of
            executable section.}
```

Figure 10-5 Declaration Section of a Math Block

Using a Declaration Statement

Declaration statements obey the following rules.

- Each declaration statement consists of a reserved word that indicates the type: **boolean**, **integer**, **real**, **timer**, or **array**.
- A colon (:) separates the type from the variable name(s).
- The maximum number of characters in a variable name is 12. Commas separate the variable names if you are assigning more than one variable to the same type.
- The optional portion of the declaration statement allows you to assign an initial value to the variable(s). The assigned value(s) must be in the form of a literal value, e.g., 5, 6.6, true. This value(s) is assigned to each variable in the declaration line. If variables require different initial values, use a different declaration line for each variable. If you need to initialize a timer you must supply values for all four constants.

Formatting a Declaration Statement

The following examples illustrate how to format a declaration statement for a variable, timer variable, or array.

To declare boolean, integer, and real variables, follow this format. Substitute boolean, integer, or real for ***type.***

<*type*> : variable, variable,... := constant ;

All booleans are retentive for S5 controllers. You do not need to specify "retentive." However, for Series 505 controllers, booleans can be either retentive or non-retentive, and so if you do not specify that a boolean is retentive, it will be treated as non-retentive. To indicate a retentive boolean for Series 505, use this format.

boolean retentive : variable, variable,... := constant ;

For a timer variable, use this format. For Series 505 controllers, substitute fast or slow for ***type***. For S5 controllers, the only type available is slow.

timer <*type*>: variable, variable,...:= constant1, constant2, constant3, constant4 ;

Constant1 is an integer and contains the current value of the timer.
Constant2 is an integer and contains the preset value of the timer.
Constant3 is a boolean that enables the timer.
Constant4 is a boolean that resets the timer.

The math timer variable is similar to the declaration timer, described in the "APT Declarations" chapter of the *APT Programming Reference Manual (Tables)*. However, the math timer variable does not support the DELAY command. Five extensions are associated with the math timer variable: **.TCC**, **.TCP**, **.ENABL**, **.RESET**, and **.TOUT**.

NOTE: If you have an S5 controller, you cannot locally declare a fast timer in the declaration section of a block. However, you can declare both fast and slow timers for S5 when you use the Declaration Table.

For an array, use this format. The array type can be real, integer, or boolean.

array (1..*n*) of <*type*>: variable, variable,...:= constant;

To specify array size, substitute a positive integer for **n**.
For Series 505 controllers, **n** can be any value in the range 1–32767.
For S5 controllers, the maximum number of elements depends on the array type. An integer array can have size 1–256; a real array can have size 1–128; and a boolean array can have size 1–4096.

If you assign a constant to the declaration, all elements in the array are set to the same value.

NOTE: For Series 505 controllers, the boolean type indicates a non-retentive array; use the array type boolean retentive for a retentive array.

Using a Locally-Declared Variable

Variables can be declared in two ways. They can be declared in the program or unit declaration table, or they can be declared within the declaration section of a math block. Variables that are declared from within a math block are considered locally-declared variables.

In order to monitor a locally-declared variable while you are executing Debug or MAITT, you need to use the following formats to designate the variable.

For locally-declared variables in a math CFB, use this format.

CFB_name.NNL.Local_Variable <.*timer_extension*>

For example, use the name FILL.NNL.TNK_LEVEL to view a locally-declared variable named TNK_LEVEL in a CFB named FILL.

For locally-declared variables in an SFC step, use this format. Substitute the appropriate step number for **x**.

$SFC_name.SMATHx.Local_Variable <.*timer_extension*>

For example, use the name $CLN.SMATH3.BRU1 to view a locally-declared variable named BRU1 in Step #3 of an SFC named CLN.

⚠ WARNING

Locally-declared variables in an SFC step contain the step number as part of the variable name. Resequencing the SFC may change the step number of the locally declared variable.

Resequencing an SFC that contains locally-declared variables could cause your application to operate in an unpredictable manner, or fail in an unsafe condition, resulting in death or serious injury to personnel, and/or damage to equipment.

If you use locally declared variables, do not resequence the SFC.

Follow these guidelines for using commands with locally-declared variables.

- Do not use the INC and DEC commands with locally-declared integers.
- Do not use the DELAY command with locally-declared timers.

Executable Section

BEGIN marks the beginning of the executable section in a math block as shown in Figure 10-6. The BEGIN statement does not end with a semicolon.

Every math block must contain one and only one BEGIN statement. The only statements in a math block that can precede the BEGIN statement are PRAGMA and the declaration statements (BOOLEAN, INTEGER, REAL, etc.).

```
{Optional code specifier  section (Series 505 controllers only)}
PRAGMA ("SF");

{Declaration section}
REAL: length, width, area;

BEGIN {Executable section}
length := 10.5;
width := 6.0;
area := length * width;
```

Figure 10-6 Executable Section of a Math Block

INIT marks the beginning of the initialization portion of a math block. This portion executes only the first time that the math block executes. INIT executes every time the SFC step is entered.

BODY marks the beginning of the body portion of a math block. This portion does not execute the first time that the block executes; however, the body portion executes each subsequent time that the block executes and continues to execute as long as the block remains active.

Figure 10-7 shows a math section with both an INIT and a BODY portion.

```
REAL : r1;              {Declaration section}
BEGIN                   {Marks beginning of executable section}
INIT                    {Marks the beginning of the initialization portion}
    r1 := 0.;           {Executes first time only}
BODY                    {Marks beginning of body portion}
    r1 := r1 + 1.;      {Executes every time except first}
```

Figure 10-7 The Structure of a Math Block

10.6 Statements

Overview

The executable section of a math block consists of one or more statements that can be any of the following: a comment, an assignment statement, a procedure statement, a print statement, or a conditional statement.

Comments

Text that is delimited by braces or by a parenthesis/asterisk combinations is a comment. A comment helps to document a program but does not affect its execution. For example:

{ This is a comment. }

(* This is also a comment. *)

A comment that begins with a left brace must end with a right brace; a comment that begins with a parenthesis/asterisk must end with an asterisk/parenthesis.

A statement can include a comment, or a comment can be on a separate line by itself.

You can nest one comment within another comment if you use different delimiters. For example, the following comments are valid:

{This comment (*contains a*) nested comment.}
(* So {does this} one *)

The following comment is not valid because both comments use braces as delimiters:

{This comment {is not} valid.}

Assignment Statement

The assignment operator (:=) assigns the value of the expression on the right-hand side of the statement to the identifier on the left-hand side of the statement and has the following format.

identifier := **expression**;

The identifier can be the name of a variable, an array, or an array element. For Series 505 only, the identifier can also be a direct controller address. The identifier cannot be an APT flag variable or read-only variable (a constant or field input such as AI, DI, WI, etc.).

The expression can be a single value or a complex expression that is evaluated to produce a result. In either case, the value of the expression on the left-hand side must evaluate to the same type of value as the identifier. See the examples in Figure 10-8.

Each assignment statement must end with a semicolon (;).

```
{Declaration section}
BOOLEAN: bool1, bool2, bool3, bool4;
INTEGER: int1, int2, int3;
REAL: length, width, area;

BEGIN {Executable section}

{Boolean assignment statements}
bool1 := TRUE;
bool2 := (pt101 >= 101.5);

{Assigns integer value to element one of array c.}
c[1] := (int1 + (int2 - int3)) * 8;

{Assign real values to variables, length and width.}
length := 10.5;
width := 6.0;

{Multiplies length by width and places result in a variable named area.}
area := length * width;

{Moves array b into array c.}
c := b;
```

Figure 10-8 Using Assignment Statements

Statements (continued)

Procedure Statement

Procedure statements consist of a predefined word from the Math Language, followed by the appropriate parameters enclosed in parentheses. Use the following format for a procedure statement. Each procedure statement must end with a semicolon (;).

keyword (parameter list);

The keyword in a procedure statement is either a predefined word that identifies the APT procedure, or a user-defined subroutine procedure. The parameter list depends on the procedure. Table 11-1 lists the APT procedure statements and the types of parameter that can be used with each statement.

Function Statement

Function statements consist of a variable, an assignment operator, and a predefined word from the Math Language, followed by the appropriate parameters in parentheses. Use the following format for a function statement. Each function statement must end with a semicolon (;).

variable := keyword (parameter list);

The keyword in a function statement is a predefined word that identifies the APT function. The type of values that are allowed as parameters depends on the function. Table 11-2 lists the APT function statements, the types of values that can be used as parameters, and the types of math code in which the function can be used.

Command Statement

Command statements allow you to execute an action on an object either from the parallel section of an SFC or from a math block that generates RLL or STL code. The commands that can be used with individual APT objects are listed with the explanation of each object. For example, see the "Device Types" chapter in the *APT Programming Reference (Tables) Manual* for an explanation of device commands. Use the following format for a command statement. Each function statement must end with a semicolon (;).

command object;

Print Statement (Series 505 only)

The PRINT statement is an SFPGM procedure that allows you to send text to Port 1 on the 545, 545L (545 Lite), 555, or 575, or one or both of the output ports on the 565T/565P card. The PRINT statement is not available for 560/560T or S5 controllers.

Use the following format for a PRINT statement. Each function statement must end with a semicolon (;).

PRINT(port-number,text);

- The port-number is an integer from 1 to 3 that specifies the print destination as follows:

  1 Print text to port 1 only.

  2 Print text to port 2 only.

  3 Print text to both port 1 and port 2.

- The text can be one or more character strings that specify the information to be printed. If the text includes more than one character string, the strings are separated by commas.

  The text must be enclosed in double quotes and can be any combination of uppercase and lowercase letters. For example, "This is a text string." However, some operator interfaces require the use of all uppercase letters.

  The text may include one or more of the special codes listed in Table 10-7. These special codes can be in either upper or lower case.

Table 10-7 PRINT Statement Codes

Code	Character	Code	Character
\b	Backspace	\t	Tab
\f	Form Feed	\v	Vertical Tab
\n	Carriage Return/Line Feed	\\	Backslash
\r	Carriage Return	\"	Quotation Mark

Figure 10-9 shows examples of print statements that all produce the same text including a tab (\t) and a carriage return and line feed (\n).

```
{Prints to Port 1}
PRINT(1,  "PRESSURE= ",pt101,"\t TEMPERATURE= ",tt101,"\n");


{Prints to Port 2}
PRINT(2,  "PRESSURE= ",pt101,
                "\t  TEMPERATURE= ",tt101,"\n");


{Prints to Ports 1 and 2}
PRINT(3,  "PRESSURE= ",pt101,"\t");
PRINT(3,  "TEMPERATURE= ",tt101,"\n");
```

Figure 10-9 Using Print Statements

IF Statement

The IF statement allows you to specify conditional execution of Math Language statements.

Use the following format for an IF statement:

IF (*boolean expression***) THEN**

Math Language statements **;**

ELSIF *(boolean expression)* **THEN**

Math Language statements ;

ELSE

Math Language statements ;

ENDIF ;

- The boolean expressions should be enclosed in parentheses.
- If the boolean expression in the IF clause is true, the Math Language statements in that clause are executed; otherwise, those statements are not executed.
- The optional ELSIF clause is executed only if the boolean expression in the IF clause is false. If the boolean expression in the ELSIF clause is true, the Math Language statements in that clause are executed; otherwise, statements in the ELSIF clause are not executed.
- The optional ELSE clause is executed only if the boolean expression in the IF clause and the boolean expression in the ELSIF clause are false.
- In any IF, ELSIF, or ELSE clause, the Math Language statements consist of one or more statements in the Math Language. Each Math Language statement must end with a semicolon.
- IF/THEN/ELSE statements can be nested up to six levels.

NOTE: The SFPGM math language of the Series 505 controller specifically supports the IF/THEN/ELSE construct. The RLL language, for Series 505, and STL, for S5, do not. APT must generate code to skip over the clause(s) in the IF/THEN/ELSE construct that should not be executed. This can lead to inefficient code. Avoid using ELSE and ELSIF in RLL or STL code.

Figure 10-10 shows several examples of the IF statement. In the fifth example section, the direct addresses used are for Series 505 controllers. For an S5 controller, direct addresses in the Math language are not allowed.

```
IF (chrg_path = 1) THEN
   chrg_sp := wpw1.c_setpoint + tare_wt;
ENDIF;


IF (chrg_path = 1) THEN
   chrg_sp := wpw1.c_setpoint + tare_wt;
ELSE
   chrg_sp :=wpw1.d_setpoint + tare_wt;
ENDIF;

IF (chrg_path = 1) THEN
   chrg_sp := wpw1.c_setpoint + tare_wt;
ELSIF (chrg_path = 2) THEN
   chrg_sp := wpw1.d_setpoint + tare_wt;
ELSE
   chrg_sp :=wpw1.e_setpoint + tare_wt;
ENDIF;

IF (chrg_path = 1) THEN
   chrg_sp := wpw1.c_setpoint + tare_wt;
ELSIF (chrg_path = 2) THEN
   chrg_sp := wpw1.d_setpoint + tare_wt;
ELSIF (chrg_path = 3) THEN
   chrg_sp := wpw1.e_setpoint + tare_wt;
ELSE
   chrg_sp :=wpw1.f_setpoint + tare_wt;
ENDIF;

IF ((%V8. <= %V10.) OR (%V12. > 0.0)) THEN      {Only Series 505 controllers}
   charg_path := 1;                             {can use direct addresses}
ELSE                                            {in Math language.}
   charg_path := 0;
ENDIF;


IF (B1) THEN
   work_req := 2;
ELSE
   work_req := 1;
ENDIF;
```

Figure 10-10 Using IF Statements

WHILE Statement

The WHILE statement allows you to program the multiple execution of math statements according to a specific condition. The WHILE statement can be used with S5 controllers and with the following Series 505 controllers: the 545, 545L, 555, 565/565T/565P, and 575 controllers. For Series 505 controllers, the WHILE statement is an SFPGM-only statement.

Use the following format for the WHILE statement. Enclose the boolean expression in parentheses.

WHILE (boolean expression) **LOOP**

Math Language statements ;

END LOOP ;

When the step containing the WHILE is active, or the math block containing the WHILE is enabled, the WHILE statement acts as follows.

- When the boolean expression in the WHILE clause is true, all the math language statements execute in order.
- The math language statements continue to execute until one of the following occurs:

  The boolean expression becomes false.

  The step is exited (for a WHILE statement in an SFC step).

  The math CFB is disabled (for a WHILE statement in a math CFB).

An example of the WHILE statement is provided in Figure 10-11. The code shown causes the first four locations of an integer array named FILL_ARRAY to be loaded with the value of 100.

```
CHK := 1;
WHILE (CHK <= 4) LOOP
   FILL_ARRAY[CHK] := 100;
   CHK := CHK + 1;
END LOOP;
```

Figure 10-11 Using the While Statement

Safety Guidelines for WHILE Loops

Use these safety guidelines to create code with WHILE statements:

- Do not create a program in which a WHILE statement can run endlessly (that is, an endless loop). Plan the boolean condition associated with the WHILE carefully. In the event that an endless loop condition does occur, deactivate the step or CFB in which the loop is located, and exit the WHILE statement.
- Initialize all variables associated with the WHILE for every execution of the loop.
- Do not program a WHILE statement that runs for more than 0.05 to 0.10 seconds, in order to avoid delaying the execution of other program tasks, such as analog alarms, SF programs, or other loops.
- Do not allow a WHILE condition that takes too long to occur.
- Do not allow a WHILE time that is too long.
- Do not allow many WHILE statements or iterations that require too much time to execute.

⚠ WARNING

Use of the WHILE loop in APT programs can either stop program execution or degrade controller performance.

The WHILE statement is powerful, and misuse of it could cause unpredictable operations that could result in death or serious injury to personnel, and/or damage to equipment.

Be sure to use the WHILE statement correctly. Follow the above Safety Guidelines for WHILE Loops to ensure that controller performance is not adversely affected.

Special SFC Considerations

When you use a WHILE statement in a step, the WHILE statement is executed to completion before any statements below the END statement are executed.

You can place a WHILE statement in any part of the math section of a step, including the INIT section. A WHILE that is located in the INIT section of a step runs to completion the first time the step is executed.

Special CFB Considerations

When you use a WHILE statement in a CFB, the WHILE statement is executed to completion before any statements below the END statement are executed.

The CFB **.ENABLD** bit does not turn on until all the statements in the CFB have been executed at least once. Therefore, an extensive WHILE can cause a delay between the time a CFB begins executing and the setting of the CFB **.ENABLD** bit.

When the math CFB containing the WHILE is sampled or is event-driven, a WHILE that has begun execution runs only as long as its associated boolean condition is true.

Chapter 11

Math Functions and Procedures

11.1 Overview

Understanding Procedures

A procedure is a pre-defined operation that performs a single task. In APT, a procedure appears as a single statement in a math block. Table 11-1 lists the procedures available in APT.

Understanding Functions

APT provides 20 functions that you can use in a program. A function appears on the right-hand side of an assignment statement, or in a boolean expression. Table 11-2 lists the functions available in APT.

Availability

All math procedures are supported for the Series 505 controllers, using either RLL, or SFPGM, or both. All math procedures except for SETSSI and COPY_BYTES are supported for S5 controllers, using STL. The SFPGM code-type procedures are not supported for the 560/560T controllers.

All math functions are supported for Series 505 controllers, using either RLL, or SFPGM, or both. All math functions are supported for S5 controllers, using STL. SFPGM code-type functions are not supported for the 560/560T controllers.

Table 11-1 Math Language Procedures

Procedure Type	Procedure (Parameters)	Code Type	Page
Boolean	FRS(boolean expression, boolean);	STL RLL [1]	11-27
	SETSSI(boolean);	RLL [1]	11-62
	UNPACK_BITS(boolean array, variable);	STL RLL [1]	11-68
Boolean to Integer	INT_TO_BITS(boolean array, variable);	STL RLL/SFPGM	11-32
	PACK_BITS(boolean array, variable);	STL RLL [1]	11-52
	PBITS_TO_INT(variable, integer);	STL SFPGM	11-53
APT Flag	CLEAR(flag_variable);	STL RLL [1]	11-19
	LATCH(flag_variable);	STL RLL [1]	11-37
	ON(flag_variable);	STL RLL [1]	11-51
Integer	BCDBIN(variable, variable);	STL RLL/SFPGM	11-11
	BINBCD(variable, variable);	STL RLL/SFPGM	11-12
	BIT_ASSIGN(variable, integer, expression);	STL RLL [1]	11-13
	BITCLEAR(variable, integer);	STL RLL/SFPGM	11-14
	BITSET(variable, integer);	STL RLL/SFPGM	11-15
	IREAD(variable);	STL RLL [1]	11-35
	IWRITE(variable);	STL RLL [1]	11-36

1 For Series 505: RLL-only procedures cannot be used in the same math block as SFPGM procedures.

Table 11-1 Math Language Procedures (continued)

Procedure Type	Procedure (Parameters)	Code Type	Page
Real	INTERPOLATE(in1, out1, array1, array2);	STL SFPGM	11-30
	LEAD_LAG(in1, lastin, out2, lead, lag, gain);	STL SFPGM [2]	11-38
	LOOKUP_TABLE(in1, out1, array1, array2);	STL SFPGM	11-44
Real or Integer (all values of same type)	COPY_BYTES(integer source, source offset, integer destination, destination offset, # of bytes);	RLL [1]	11-20
	COPY_DIRECT(destination, source, integer);	STL RLL/SFPGM	11-22
	LIMIT(in1, out1, low,high);	STL SFPGM	11-41
	LOAD_ARRAY(in1, array);	STL RLL/SFPGM	11-42
	MAX(in1, max);	STL SFPGM	11-48
	MIN(in1, min);	STL SFPGM	11-49
	MINMAX(in1, max, min);	STL SFPGM	11-50
Integer to Real	SCALE(in1, out1, type, low, high);	STL SFPGM [3]	11-58
Real to Integer	PROUND(integer, real);	STL SFPGM	11-54
	PTRUNC(integer, real);	STL SFPGM	11-55
	UNSCALE(in1, out1, type, low, high);	STL SFPGM [3]	11-70

1 For Series 505: RLL-only procedures cannot be used in the same math block as SFPGM procedures.
2 Gain, lead, and lag must be real variables or real numbers; input and output variables (in1, lastin, and out2) can be integer or real, but all three must be of same type.
3 Low and high values must be real numbers.

Table 11-2 Math Language Functions

Type of Function	Function (Parameters)	Value Type	Code Type	Page
Arithmetic Functions	ABS (expression)	Integer or Real	STL RLL/SFPGM	11-7
	EXP (expression)	Real	STL SFPGM	11-26
	FRAC (expression)	Real	STL SFPGM	11-28
	INT_TO_REAL (expression)	Real	STL SFPGM	11-34
	LN (expression)	Real	STL SFPGM	11-46
	LOG (expression)	Real	STL SFPGM	11-47
	ROUND (expression)	Integer	STL SFPGM	11-57
	SQRT (expression)	Real	STL SFPGM	11-65
	TRUNC (expression)	Integer	STL SFPGM	11-67
Trigonometric Functions	ARCCOS (expression)	Real	STL SFPGM	11-8
	ARCSIN (expression)	Real	STL SFPGM	11-9
	ARCTAN (expression)	Real	STL SFPGM	11-10
	COS (expression)	Real	STL SFPGM	11-24
	SIN (expression)	Real	STL SFPGM	11-64
	TAN (expression)	Real	STL SFPGM	11-66
Bit Functions	BITS_TO_INT (array variable)	Integer	STL SFPGM	11-16
	BITTEST (variable, integer)	Boolean	STL RLL [1]	11-18
	EDGE (expression)	Boolean	STL RLL [1]	11-25
	LEFTSHIFT (expression, expression)	Integer	STL SFPGM	11-40
	RIGHTSHIFT (expression, expression)	Integer	STL SFPGM	11-56

1 RLL-only functions cannot be used in the same math block as SFPGM functions.

Using Functions and Procedures in Subroutines (Series 505)

As a normal part of its operation, a Series 505 controller handles a subroutine that has been compiled into SFPGM code in the following ways:

- In some situations, APT allocates locations in Series 505 V-Memory to be used as compiler internal temporary variables by the subroutine.
- The Series 505 controller temporarily suspends execution of the subroutine in order to enable another subroutine to begin execution.

The combination of these two events can cause a loss of data: the temporary variables of the first subroutine may be overwritten during execution of the subsequent subroutine.

In consequence, if you have a Series 505 controller, you need to observe the following guidelines for math statements used in a subroutine, in order to avoid the possibility of creating temporary variables that risk being overwritten during the subroutine's interruption.

- For functions and procedures that accept either a variable or an expression as a parameter, use only a variable. An expression causes temporary variables to be allocated. You can store the intermediate results of an expression in a Series 505 T-Memory location before calling the subroutine ($\%T_x$, where x = 11–16).
- Use the PTRUNC procedure, not the TRUNC function, in user-defined subroutines.
- Use the PROUND procedure, not the ROUND function, in user-defined subroutines.
- Use the PBITS_TO_INT procedure, not the BITS_TO_INT function, in user-defined subroutines.

NOTE: The EDGE function also stores data temporarily by using internal memory locations. Therefore, if you have a Series 505 controller, avoid using EDGE in user-defined subroutines.

11.2 Function and Procedure Definitions

ABS Function

The ABS function is a real or integer function that is available in STL for S5 controllers and is available in both RLL and SFPGM for Series 505; ABS can be used with all controllers. ABS returns the positive value of the expression in parentheses.

To enter the ABS function, use:

ABS(expression)

- Expressions, values, or functions can go in the parenthesis as input.
- STL and SFPGM can use real or integer functions, but the expression must match the returned type. See Figure 11-1.
- Series 505 RLL can use only integer; refer to Figure 11-2.

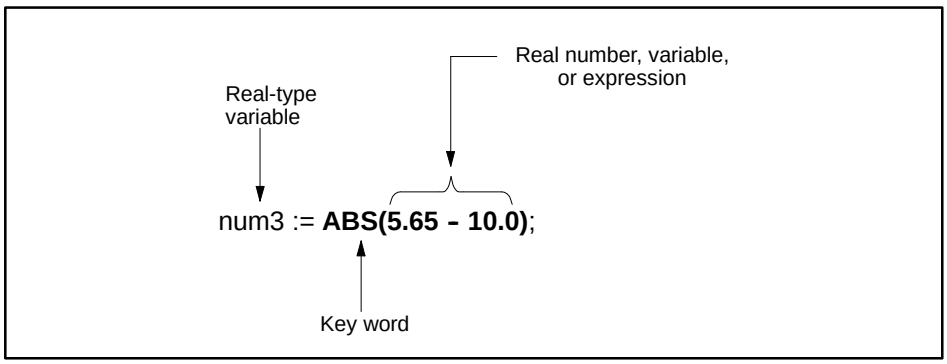


Figure 11-1 ABS Real Function

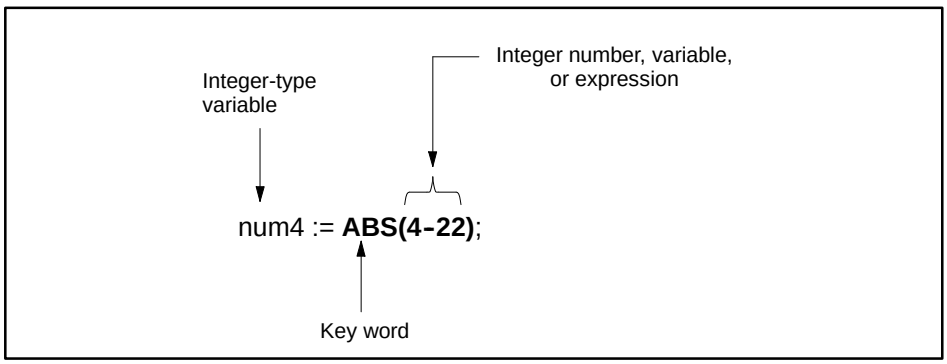


Figure 11-2 ABS Integer Function

Function and Procedure Definitions (continued)

ARCCOS Function

The ARCCOS function is a real function that is available in STL for S5 controllers and is available only in SFPGM for Series 505. ARCCOS can be used with the S5 controller family and the 545, 545L, 555, 565, 565T/565P, and 575 controllers. ARCCOS returns the inverse cosine of the expression in parentheses. The ARCCOS function is the inverse of the COS function.

To enter the ARCCOS function, use:

ARCCOS(expression)

- The expression in parentheses must evaluate to a real number between –1 and 1.
- The returned value is a real number that represents the measure of an angle in radians. For example, in Figure 11-3, the value of the variable, **angle1**, is 1.05.

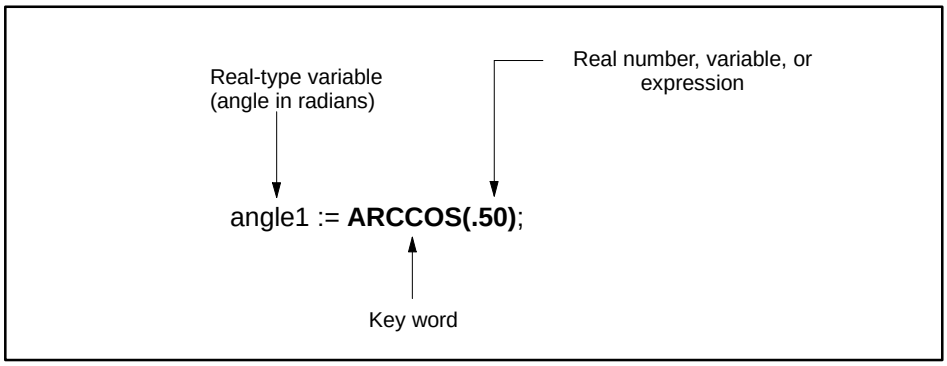


Figure 11-3 ARCCOS Function

ARCSIN Function

The ARCSIN function is a real function that is available in STL for S5 controllers and is available only in SFPGM for Series 505 controllers. ARCSIN can be used with the S5 controller family and the 545, 545L, 555, 565, 565T/565P, and 575 controllers. ARCSIN returns the inverse sine of the expression in parentheses. The ARCSIN function is the inverse of the SIN function.

To enter the ARCSIN function, use:

ARCSIN(expression)

- The expression in parentheses must evaluate to a real number between –1 and 1.
- The returned value is a real number that represents the measure of an angle in radians. For example, in Figure 11-4, the value of the variable, **angle2**, is 0.52.

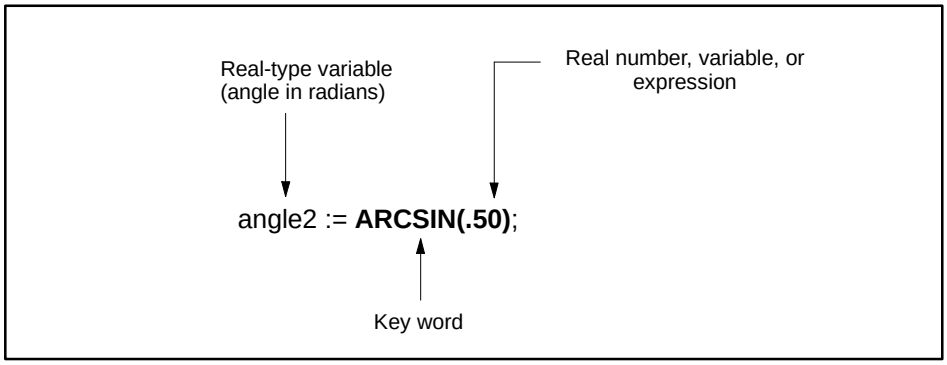


Figure 11-4 ARCSIN Function

Function and Procedure Definitions (continued)

ARCTAN Function

The ARCTAN function is a real function that is available in STL for S5 controllers and is available only in SFPGM for Series 505 controllers. ARCTAN can be used with the S5 controller family and the 545, 545L, 555, 565, 565T/565P, and 575 controllers. ARCTAN returns the inverse tangent of the expression in parentheses. The ARCTAN function is the inverse of the TAN function.

To enter the ARCTAN function, use:

ARCTAN(expression)

- The expression in parentheses must evaluate to a real number.
- The returned value is a real number that represents the measure of an angle in radians. For example, in Figure 11-5, the value of the variable, angle3, is 0.785.

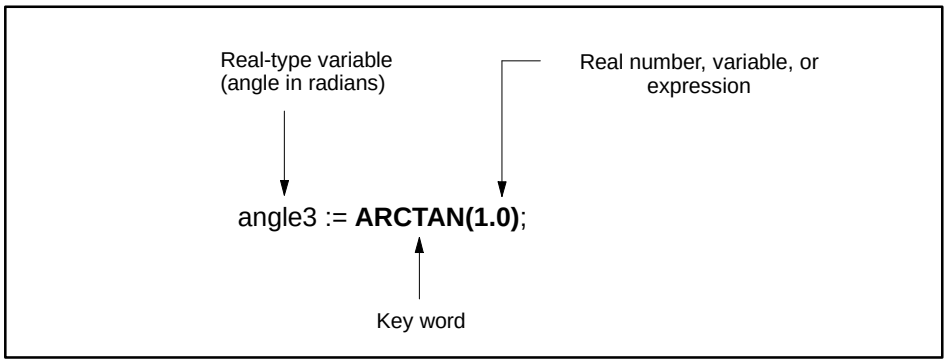


Figure 11-5 ARCTAN Function

BCDBIN Procedure

The BCDBIN procedure is available in STL for S5 controllers and is available in both RLL and SFPGM for Series 505 controllers. BCDBIN can be used with all controllers. BCDBIN converts a BCD (binary-coded decimal) value to an integer. After APT converts the BCD value to an integer, you can use the integer in math calculations, whereas you could not use the original BCD value.

To enter the BCDBIN procedure, use:

BCDBIN(variable,variable);

- The first variable in the parentheses contains the four-digit BCD value that is to be converted. Notice that the BCD value is in hex format, and the digits range from 0 to 9 (no A to F).
- The second variable in the parentheses is an integer variable that stores the converted result. See the example in Figure 11-6.

NOTE: To avoid using the BCDBIN procedure to bring in a value from an I/O point, consider declaring the I/O as a BI (instead of a WI). This way, the conversion is handled automatically for you.

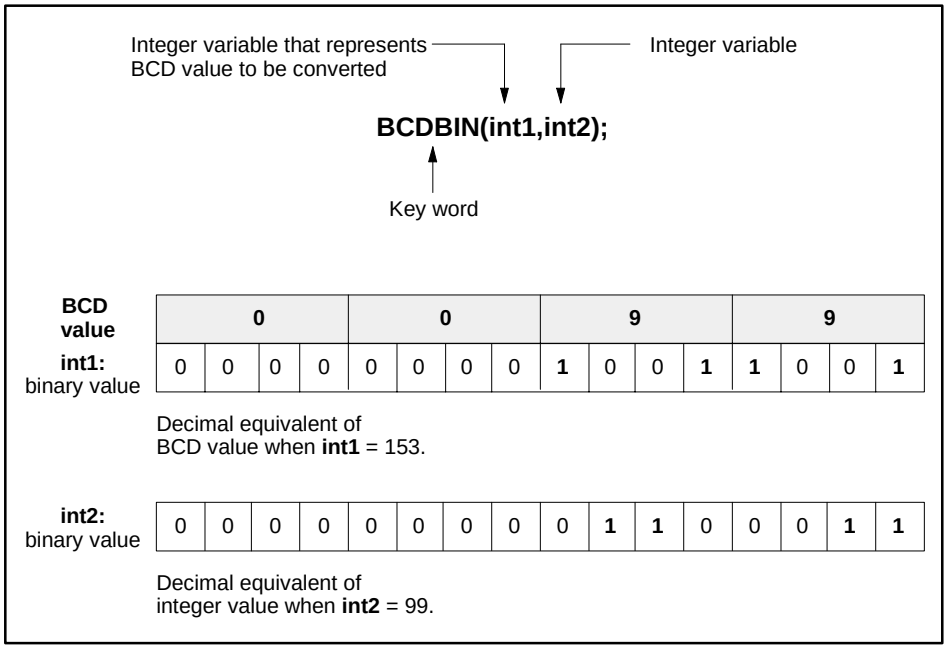


Figure 11-6 BCDBIN Procedure

Function and Procedure Definitions (continued)

BINBCD Procedure

The BINBCD procedure is available in STL for S5 controllers and is available in both RLL and SFPGM for Series 505 controllers. BINBCD can be used with all controllers. BINBCD converts an integer value to a BCD (binary-coded decimal) value. Notice that the BCD value is in hex format, and the digits range from 0 to 9 (no A to F).

To enter the BINBCD procedure, use:

BINBCD (variable,variable);

- The first variable in the parentheses contains the integer to be converted to a BCD value.
- The second variable in the parentheses is an integer variable that stores the converted value (that is, the BCD value). See the example in Figure 11-7.

NOTE: To avoid using the BINBCD procedure to send a value out to an I/O point, consider declaring the I/O as a BO (instead of a WO). This way, the conversion is handled automatically for you.

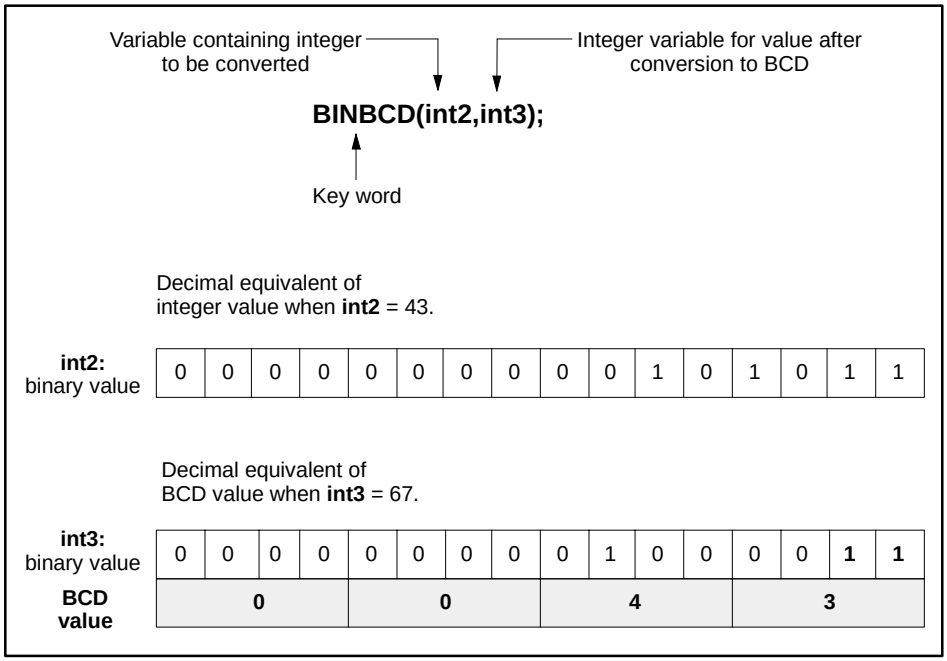


Figure 11-7 BINBCD Procedure

BIT_ASSIGN Procedure

The BIT_ASSIGN procedure is available in STL for S5 controllers and is available only in RLL for Series 505 controllers. BIT_ASSIGN can be used with all controllers. BIT_ASSIGN sets an individual bit position of an integer based on the result of a boolean expression.

Use the following format to enter the BIT_ASSIGN procedure.

BIT_ASSIGN(variable, integer, expression)

- The variable in the parentheses is the integer that contains the bit you want to set based on the boolean expression. This integer is treated as a binary value with bits numbered 1 (the most significant bit) to 16 (the least significant bit).
- The integer specifies which bit in the binary equivalent of the integer variable is the one that you want to set. The value must be a number between 1 and 16.
- The expression must be a boolean expression that evaluates to true or false.

Figure 11-8 shows the format used to enter the BIT_ASSIGN procedure.

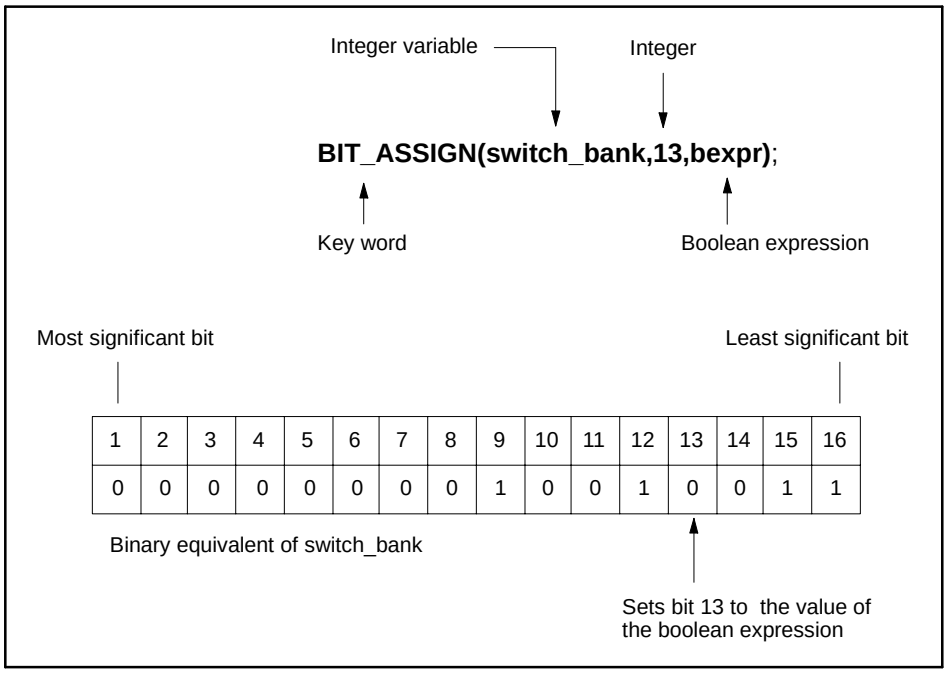


Figure 11-8 BIT_ASSIGN Procedure

Function and Procedure Definitions (continued)

BITCLEAR Procedure

The BITCLEAR procedure is available in STL for S5 controllers and is available in both RLL and SFPGM for Series 505 controllers. BITCLEAR can be used with all controllers. BITCLEAR resets a specified bit in an integer variable to false or off (0).

To enter the BITCLEAR procedure, use:

BITCLEAR (variable,integer);

- The variable in the parentheses is the integer that contains the bit you want to reset to 0. This integer is treated as a binary value with bits numbered 1 (the most significant bit) to 16 (the least significant bit).
- The integer specifies which bit in the binary equivalent of the integer variable is the one that you want to clear. The value must be a number between 1 and 16. See the example in Figure 11-9.

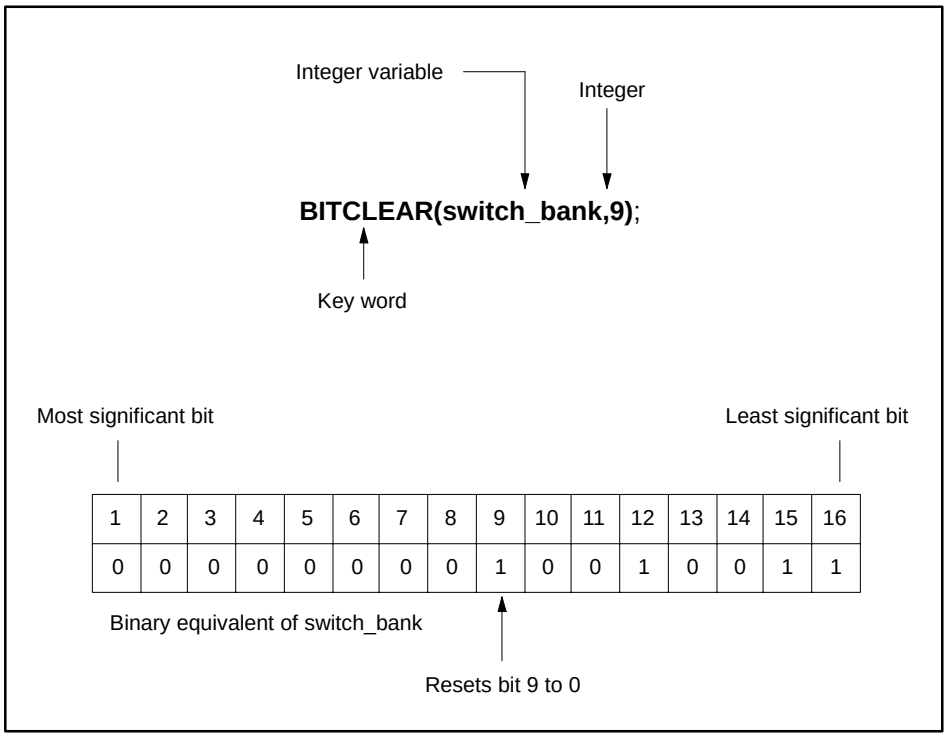


Figure 11-9 BITCLEAR Procedure

BITSET Procedure

The BITSET procedure is available in STL for S5 controllers and is available in both RLL and SFPGM for Series 505 controllers. BITSET can be used with all controllers. BITSET sets a specified bit in an integer variable to true or on (1).

To enter the BITSET procedure, use:

BITSET (variable,integer);

- The variable in the parentheses is the integer that contains the bit you want to set to 1. This integer is treated as a binary value with bits numbered 1 (the most significant bit) to 16 (the least significant bit).
- The integer specifies which bit in the binary equivalent of the integer variable is the one that you want to set. The value must be a number between 1 and 16. See the example in Figure 11-10.

Figure 11-10 shows how the BITSET procedure works.

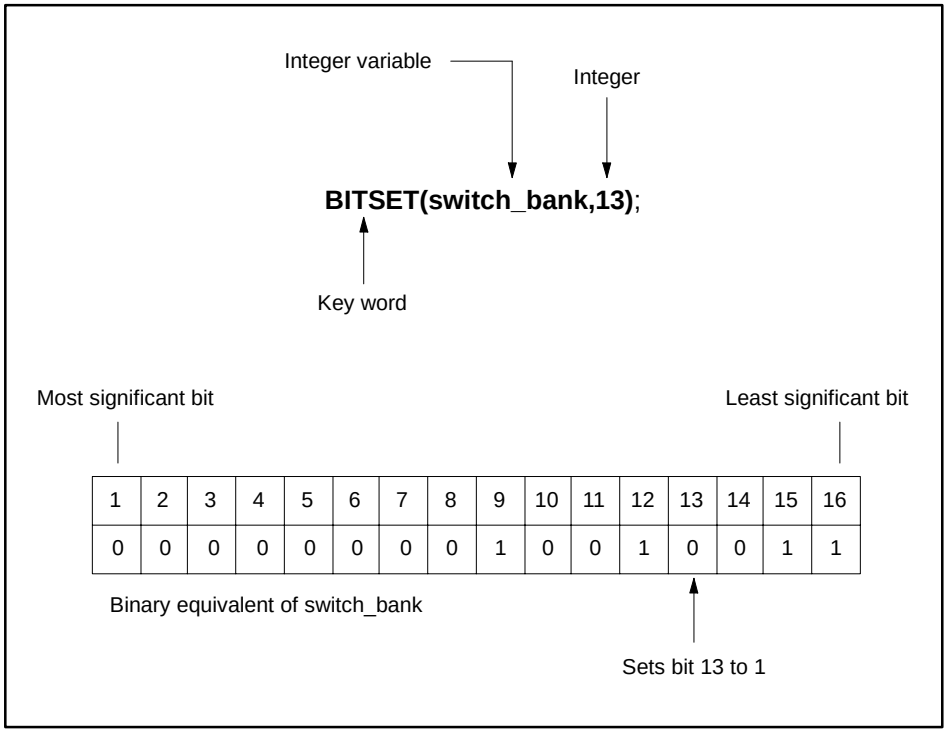


Figure 11-10 BITSET Procedure

Function and Procedure Definitions (continued)

BITS_TO_INT Function

The BITS_TO_INT function is an integer function that is available in STL for S5 controllers and is available in both RLL and SFPGM for Series 505 controllers. BITS_TO_INT can be used with the S5 controller family and the 545, 545L, 555, 565, 565T/565P, and 575 controllers. BITS_TO_INT moves an array of 16 boolean values into an integer variable.

If you have a Series 505 controller, consider using the PACK_BITS procedure; it generates faster, more efficient code than BITS_TO_INT in a Series 505 environment.

To enter the BITS_TO_INT function, use:

BITS_TO_INT(variable)

- The variable in parentheses must be a boolean array. (A boolean value of false is a 0; true is a 1.) Element 1 of the array is the most significant bit; element 16 is the least significant bit.
- The returned integer value is the decimal equivalent of the binary value constructed from the bit values of the boolean array. For example, in Figure 11-11, the value of the variable **binvalue** is 147.

The BITS_TO_INT procedure moves the first element of the array into the most significant bit of the integer (see Figure 11-11). The PACK_BITS procedure moves the first element of an array into the least significant bit of the integer.

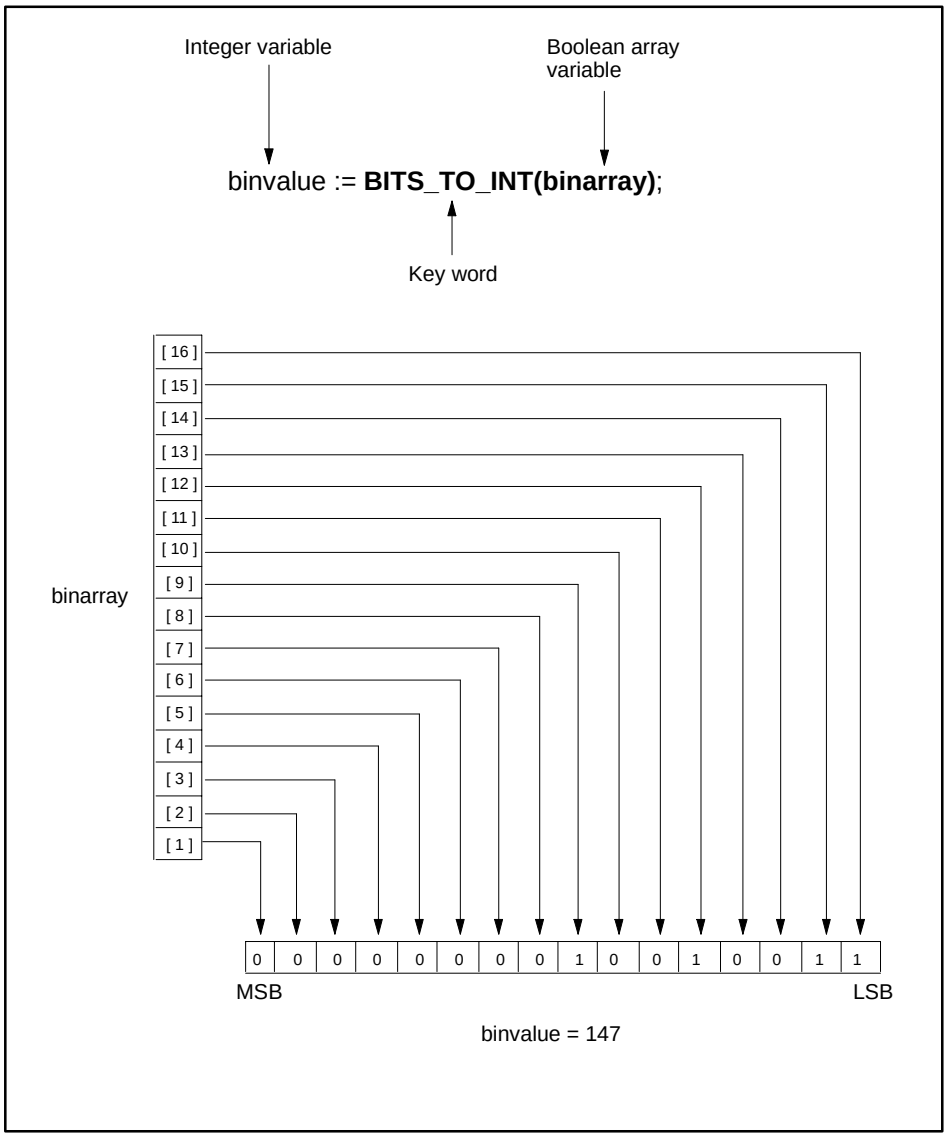


Figure 11-11 Example of BITS_TO_INT Operation

Function and Procedure Definitions (continued)

BITTEST Function

The BITTEST function is a boolean function that is available in STL for S5 controllers and is available only in RLL for Series 505 controllers. BITTEST can be used with all controllers. BITTEST checks the status of a specified bit. BITTEST can be used in an SFC transition.

To enter the BITTEST function, use:

BITTEST(variable,integer)

- The variable in parentheses must be the integer that contains the bit that you want to check. This integer is treated as a binary value with bits numbered 1 (the most significant bit) to 16 (the least significant bit).
- The integer specifies which bit in the binary equivalent of the integer variable is the one that you want to test. The value must be a number between 1 and 16.
- If the tested bit is 1, the returned value is true. For example, in Figure 11-12, the value of the variable, **bit12**, is true. If the tested bit is 0, the returned value is false.

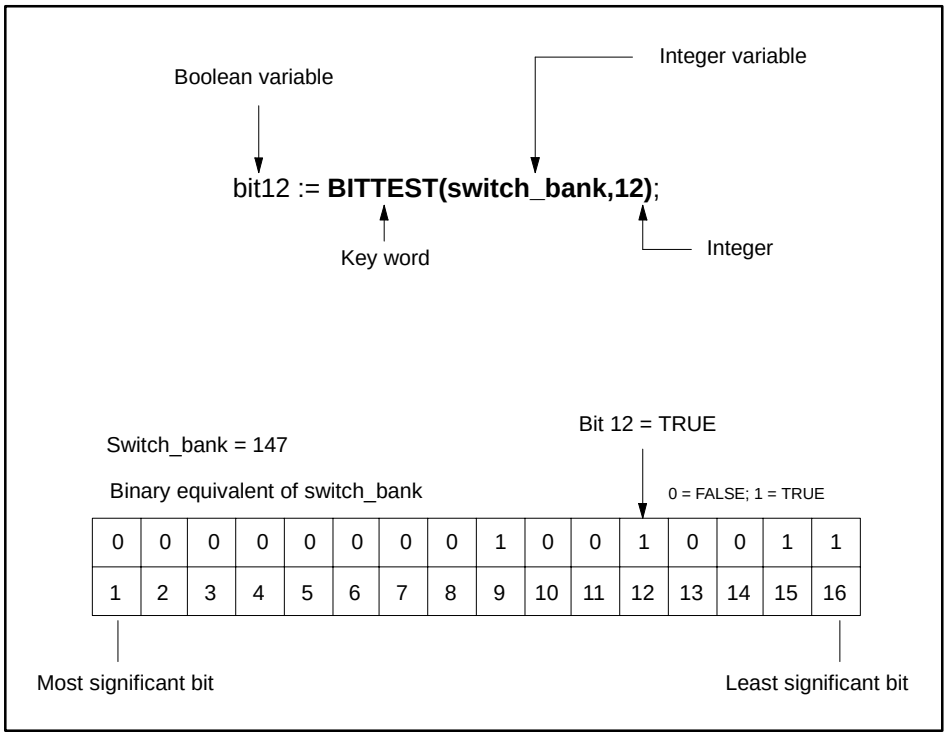


Figure 11-12 BITTEST Function

CLEAR Procedure

The CLEAR procedure is available in STL for S5 controllers and is available only in RLL for Series 505. It can be used with all controllers. CLEAR sets the APT flag value to off (false) after the procedure is executed. The APT flag remains off, even if the block is disabled, until a LATCH or ON sets it to on. CLEAR takes precedence over ON or LATCH.

To enter the CLEAR procedure, use:

CLEAR (flag_variable);

- The variable in the parentheses is the APT flag that is turned off (false). See the example in Figure 11-13.

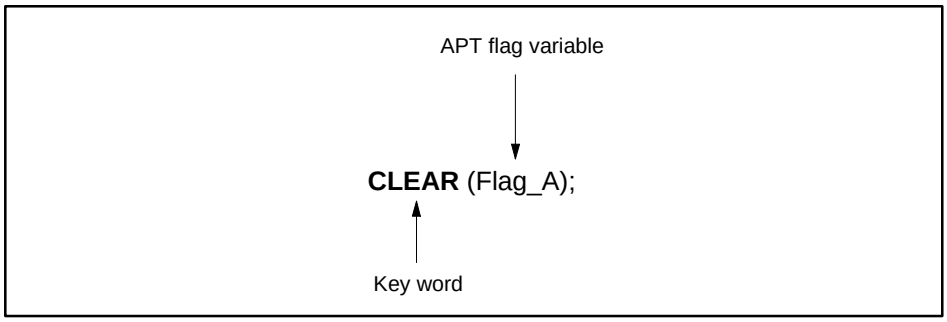


Figure 11-13 CLEAR Procedure

Function and Procedure Definitions (continued)

COPY_BYTES Procedure

The COPY_BYTES procedure is not available for S5 controllers; it is available only in RLL for Series 505 controllers. It can only be used with supported releases of the 545, 545L, 555, and 575 controllers.
COPY_BYTES copies one or more bytes from one memory region to another. Refer to Chapter 1 in the *SIMATIC APT Programming Reference (Tables) Manual* for controller release-specific features.

Use the following format to enter the COPY_BYTES procedure.

COPY_BYTES(source, source offset, destination, destination offset, # of bytes)

Source Identifies the starting integer address for the source. It can be either a direct address or a declared integer variable. For Series 505 controllers, you can use V, G, VMS, or VMM word memory.

Source Offset The integer value may either be 0, for no offset, or 1, for an offset of one byte.

Destination Identifies the starting integer address for the destination. It can be either a direct address or a declared integer variable. For Series 505 controllers, you can use V, G, VMS, or VMM word memory.

Destination Offset The integer value may either be 0, for no offset, or 1, for an offset of one byte.

Number of Bytes An integer representing the number of bytes to copy from the source to the destination.

⚠ WARNING

The COPY_BYTES procedure allows you to copy large amount of data to specified memory locations within the controller. APT does not check to see if these memory locations are currently in use by other sections of the APT program.

Overwriting controller memory could cause unpredictable operation, which could result in death or serious injury to personnel and/or equipment.

Be sure to reserve the appropriate amount of memory in the Compiler Control File and use only those memory locations in this procedure.

In Figure 11-14, 32 bytes are copied from the Series 505 address, %VMM0F000, and there is no offset to the VMM address of the declared variable, TABLE. This example uses a Series 505 direct address.

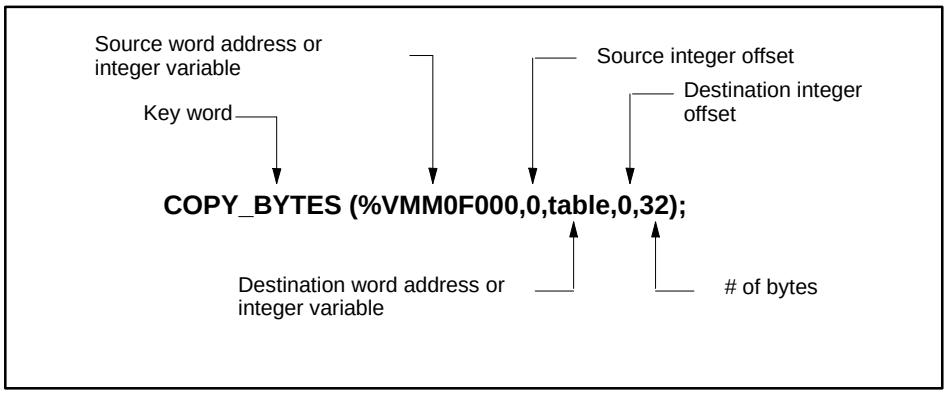


Figure 11-14 COPY_BYTES Procedure

Function and Procedure Definitions (continued)

COPY_DIRECT Procedure

The COPY_DIRECT procedure is available in STL for S5 controllers and is available in both RLL and SFPGM for Series 505 controllers. COPY_DIRECT can be used with all controllers. COPY_DIRECT moves a block of data from one location to another.

To enter the COPY_DIRECT procedure, use:

```
COPY_DIRECT(destination,source,integer);
```

Destination Identifies the starting address of the destination in controller memory. For S5 controllers, must be a declared array variable. For Series 505 controllers, can be either a declared array variable or else a direct address that is associated with a block of reserved memory.

Source For S5 controllers, must be a declared array variable. For Series 505 controllers, can be either a declared array variable or else a direct address that is associated with a block of reserved memory. If the values in the array are real, each value requires two locations in Series 505 reserved memory or in S5 data-word memory; four locations are required if you use S5 flag-word memory. See the examples in Figure 11-15.

Integer Indicates the number of elements that you want to copy from the beginning of the array into the specified controller memory locations.

An integer array that you want to use in a COPY_DIRECT procedure can contain as many as 32,767 elements for a Series 505 controller, or 256 for an S5 controller. A real array can contain 16,383 elements, if you have a Series 505 controller, or 128 if you have an S5.

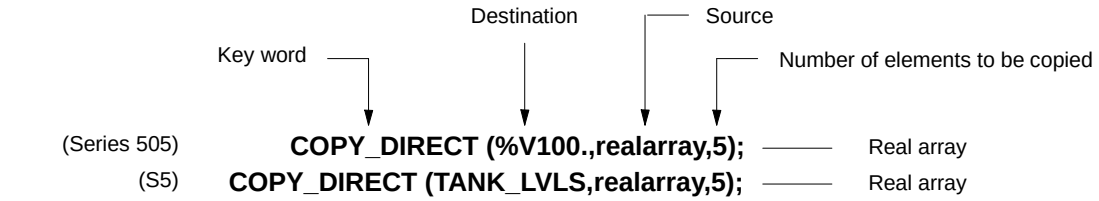
⚠ WARNING

The COPY_DIRECT procedure allows you to copy large amount of data to specified memory locations within the controller. APT does not check to see if these memory locations are currently in use by other sections of the APT program.

Overwriting controller memory could cause unpredictable operation, which could result in death or serious injury to personnel and/or equipment.

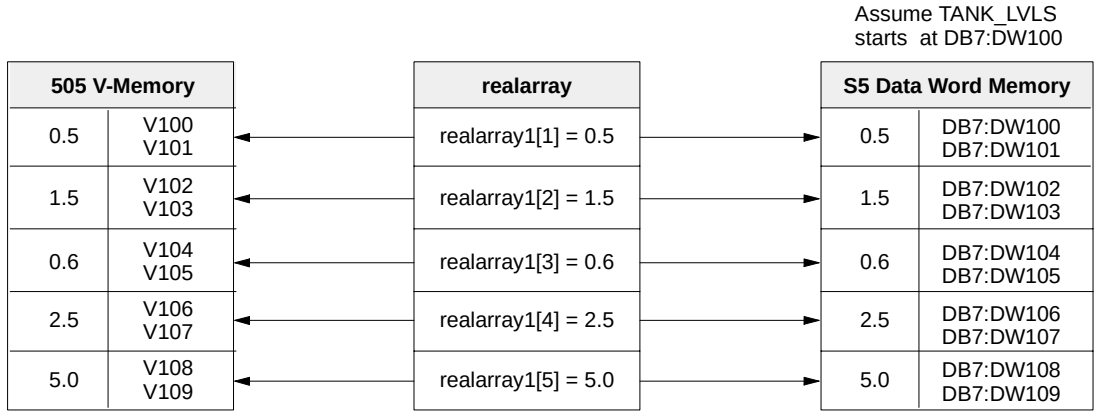
Be sure to reserve the appropriate amount of memory in the Compiler Control File and use only those memory locations in this procedure.

Figure 11-15 shows how the COPY_DIRECT procedure works.

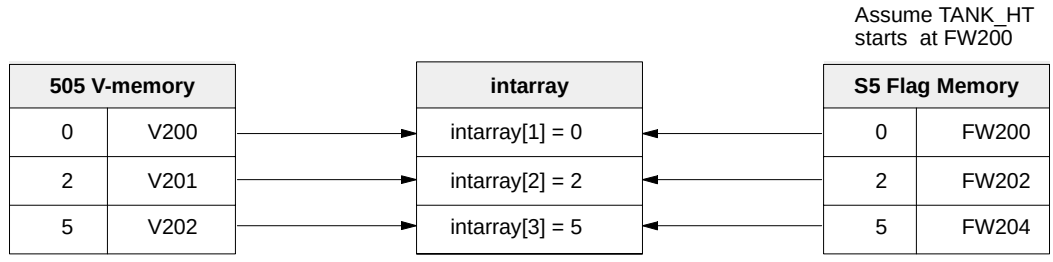


The real array in the examples above is used as the source; it supplies data to a destination.
For the Series 505 example, %V100 is used as the destination. For the S5 example, TANK_LVLS is the destination.

The integer array in the examples above is used as the destination; it receives data from a source location.
For the Series 505 example, %V200 is used as the source. For the S5 example, TANK_HT is the source.



Data is moved from the real array (source) to the Series 505 or S5 controller memory locations.



Data is moved from the Series 505 or S5 controller memory locations to the integer array (destination).

Figure 11-15 COPY_DIRECT Procedure

Function and Procedure Definitions (continued)

COS Function

The COS function is a real function that is available in STL for S5 controllers and is available only in SFPGM for Series 505 controllers. COS can be used with the S5 controller family and the 545, 545L, 555, 565, 565T/565P, and 575 controllers. COS returns the cosine of the expression in parentheses. The COS function is the inverse of the ARCCOS function.

To enter the COS function, use:

COS(expression)

- The expression in parentheses must evaluate to a real number that represents the measure of an angle in radians.
- For S5 controllers, the expression cannot evaluate to a negative number, and it must represent radians in the range 0–2π.
- The returned value is a real number between –1 and 1. For example, in Figure 11-16, the value of the variable, **angle1**, is 0.50.

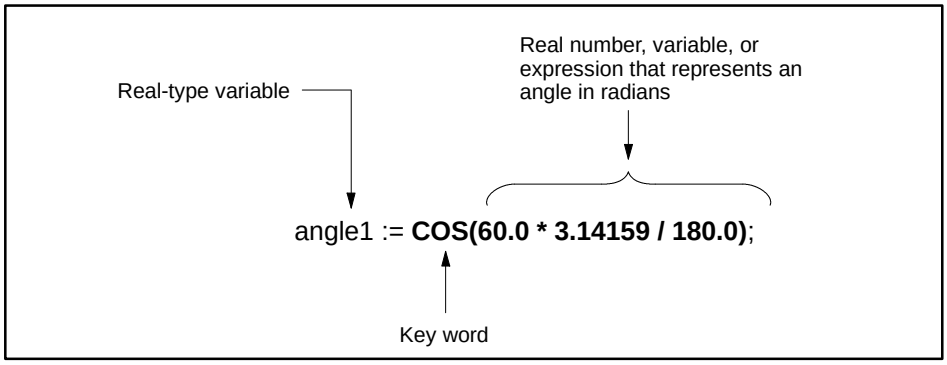


Figure 11-16 COS Function

EDGE Function

The EDGE function is a boolean function that is available in STL for S5 controllers and is available only in RLL for Series 505. It can be used with all controllers. EDGE detects a change from false to true in the value of a boolean expression. EDGE can be used in an SFC transition.

NOTE: For Series 505 controllers, avoid using EDGE in user-defined subroutines, since the EDGE function stores data temporarily by using internal memory locations.

To enter the EDGE function, use:

EDGE(expression)

- The variable in parentheses is a boolean value that is monitored to detect the first time the expression changes from false to true.
- When that expression changes from false to true, the returned value of the function becomes true and remains true for one scan of the controller.
- The returned value is a boolean variable.

In Figure 11-17, assume that bool1 and bool2 are declared as boolean values. In the first example, bool2 detects a false to true transition in bool1. In the second example, bool2 detects a true to false transition in bool1.

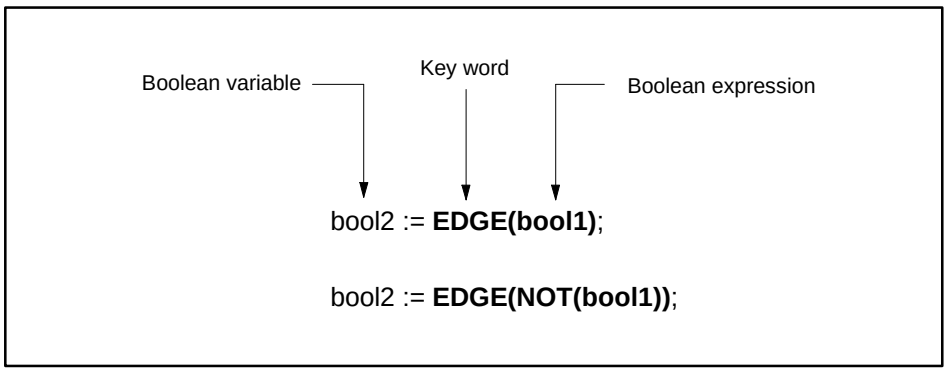


Figure 11-17 EDGE Function

Function and Procedure Definitions (continued)

EXP Function

The EXP function is a real function that is available in STL for S5 controllers and is available only in SFPGM for Series 505 controllers. EXP can be used with the S5 controller family and the 545, 545L, 555, 565, 565T/565P, and 575 controllers. EXP returns the value of e (2.71828) raised to the power represented by the expression in parentheses. (The EXP function is the inverse of the LN function.)

To enter the EXP function, use:

EXP(expression)

- The expression in parentheses must evaluate to a real number.
- The returned value is also a real number. For example, in Figure 11-18, the value of the variable, **real1**, is 2.7.

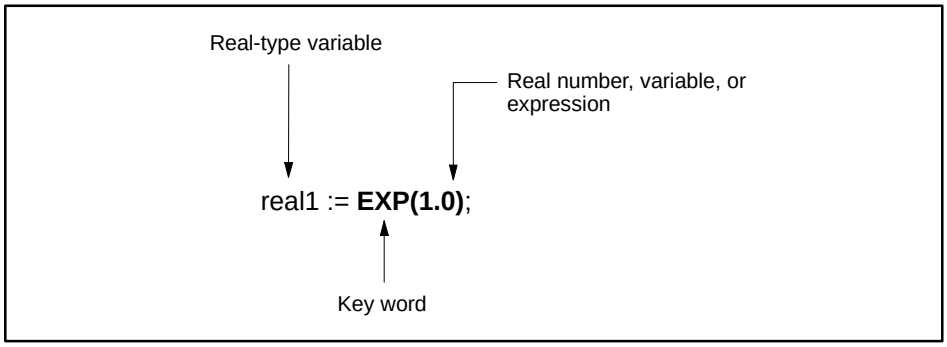


Figure 11-18 EXP Function

FRS Procedure

The Forced Role Swap (FRS) procedure is available in STL for S5 controllers, and is available only in RLL for Series 505 controllers. FRS can be used with the S5 948R and the SIMATIC 560, 560T, 565, 565T, and 565P controllers. This procedure allows you to change the roles of the controllers so that the current standby unit becomes active and the current active unit becomes the standby.

To enter the FRS procedure, use:

FRS (bexpr,out_bit);

- BEXPR — a boolean expression that triggers the forced role swap.
- OUT_BIT — a boolean value that is turned on after the role swap occurs. See Figure 11-19.

If an active controller with a standby detects an off-to-on transition on the input (BEXPR), it queues a role swap to occur at the beginning of the next controller scan. OUT_BIT is turned on when the forced role swap is executed.

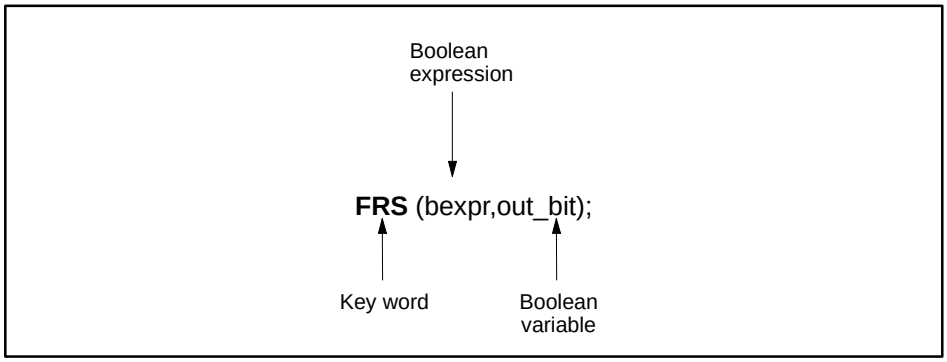


Figure 11-19 FRS Procedure

NOTE: For S5 controllers, APT uses FW0 (FY0 and FY1) as the H-flag word for redundant control.

Function and Procedure Definitions (continued)

FRAC Function

The FRAC function is a real function that is available in STL for S5 controllers and is available only in SFPGM for Series 505 controllers. FRAC can be used with S5 controllers and with the 545, 545L, 555, 565, 565T/565P, and 575 controllers. FRAC returns the fractional portion of the expression in parentheses.

To enter the FRAC function, use:

FRAC(expression)

- The expression in parentheses must evaluate to a real number.
- The returned value is also a real number. For example, in Figure 11-20, the value of the variable, **real2**, is 0.25.
- If the expression in parentheses is negative, the result of the FRAC function is also negative.

Figure 11-20 shows how the FRAC function works.

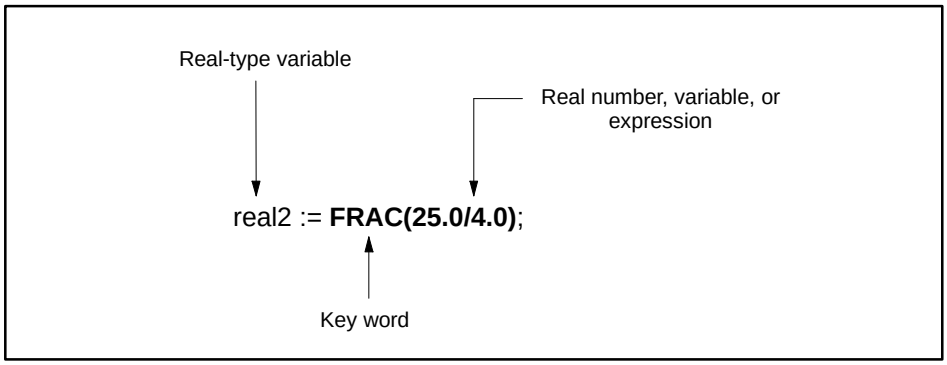


Figure 11-20 FRAC Function

Function and Procedure Definitions (continued)

INTERPOLATE Procedure

The INTERPOLATE procedure is available in STL for S5 controllers and is available only in SFPGM for Series 505 controllers. INTERPOLATE can be used with the S5 controller family and the 545, 545L, 555, 565, 565T/565P, and 575 controllers. This procedure calculates a value between two known values.

INTERPOLATE first locates an input value an input array and then uses one of the following techniques to determine the output.

- If the input is equal to an element of the input array, the output is assigned the value of the same element in the output array.
- If the input falls between two elements in the array, INTERPOLATE calculates the relative position between the two elements and uses this calculation in determining the output.
- If the input is less than the value of the first element in the input array, the output is assigned a value less than the value of the first element in the output array. This value is based on the relationship calculated between the input value and the values of the first two elements in the input array.
- If the input is greater than the value of the last element of the input array, the output is assigned a value greater than the value of the last element of the output array. This value is based on the relationship calculated between the input value and the values of the last two elements in the input array.

When you declare the arrays that you want to use in an INTERPOLATE procedure, you must specify 11 as the number of elements in each array.

To enter the INTERPOLATE procedure, use:

INTERPOLATE(input variable,output variable,input array,output array);

- The input variable must be a real variable.
- The output variable is a real variable that stores the result of the INTERPOLATE procedure. See the example in Figure 11-21.
- The input array is an 11-element array of real numbers. The values in the array must be arranged in ascending order.
- The output array is an 11-element array of real numbers. The values in the array must be arranged in the same order as the input array.

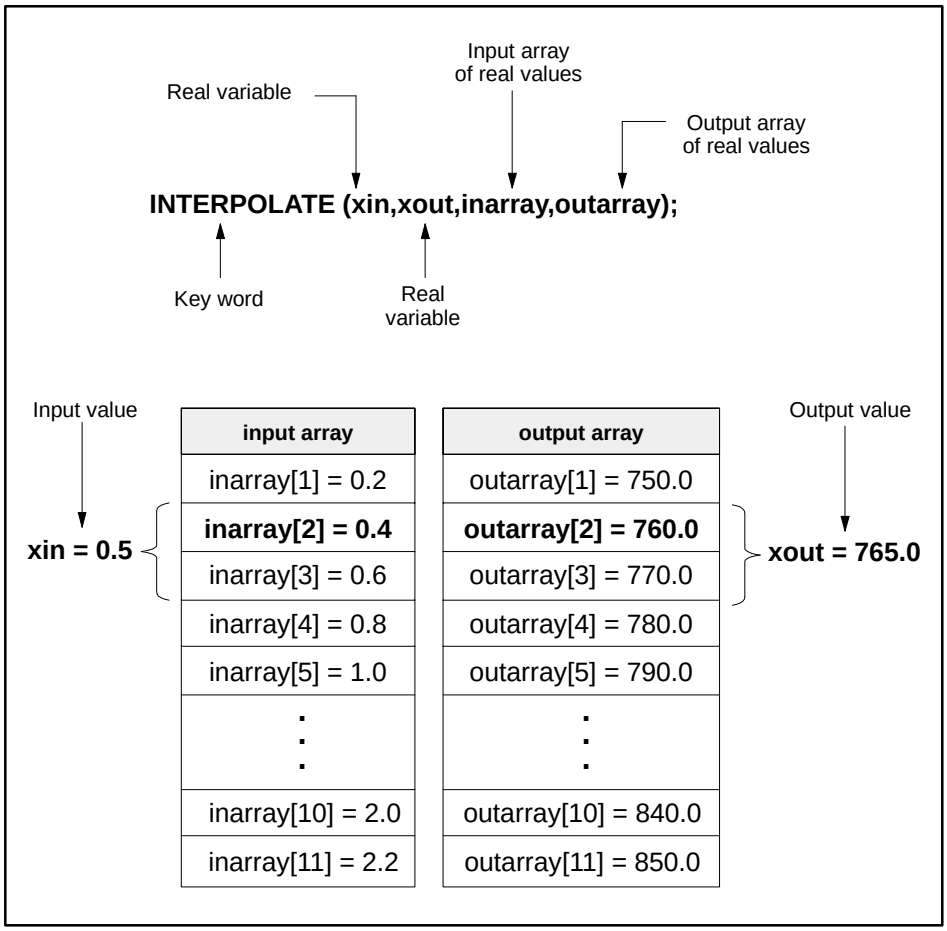


Figure 11-21 INTERPOLATE Procedure

Function and Procedure Definitions (continued)

INT_TO_BITS Procedure

The INT_TO_BITS procedure is available in STL for S5 controllers and is available in both RLL and SFPGM for Series 505. It can be used with all controllers. INT_TO_BITS converts an integer value to an array of 16 boolean values.

Consider using the UNPACK_BITS procedure; UNPACK_BITS generates faster, more efficient code than INT_TO_BITS.

To enter the INT_TO_BITS procedure, use:

INT_TO_BITS(variable,variable);

- The first variable in parentheses must be a 16-bit boolean array. A 0 bit equals a boolean value of false; 1 equals true. The most significant bit becomes element 1 of the array; the least significant bit becomes element 16.
- The second variable contains the integer that is to be converted. See the example in Figure 11-22.

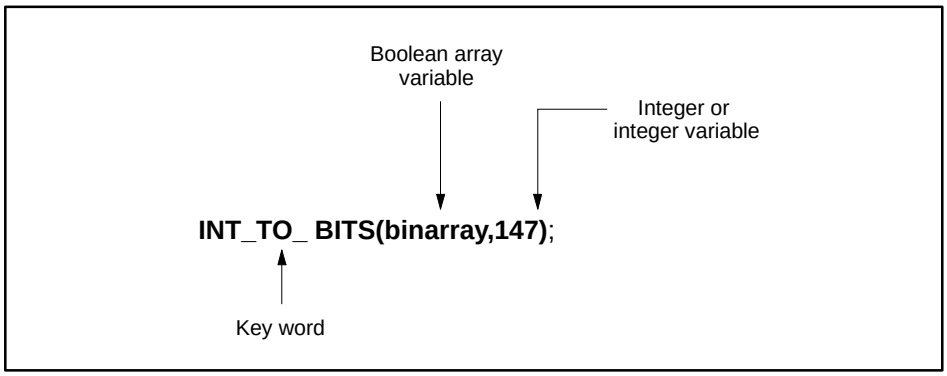


Figure 11-22 INT_TO_BITS Procedure

The INT_TO_BITS procedure moves the most significant bit of the integer into the first element of the array, as shown in Figure 11-23. The UNPACK_BITS procedure moves the least significant bit of the integer into the first element of an array.

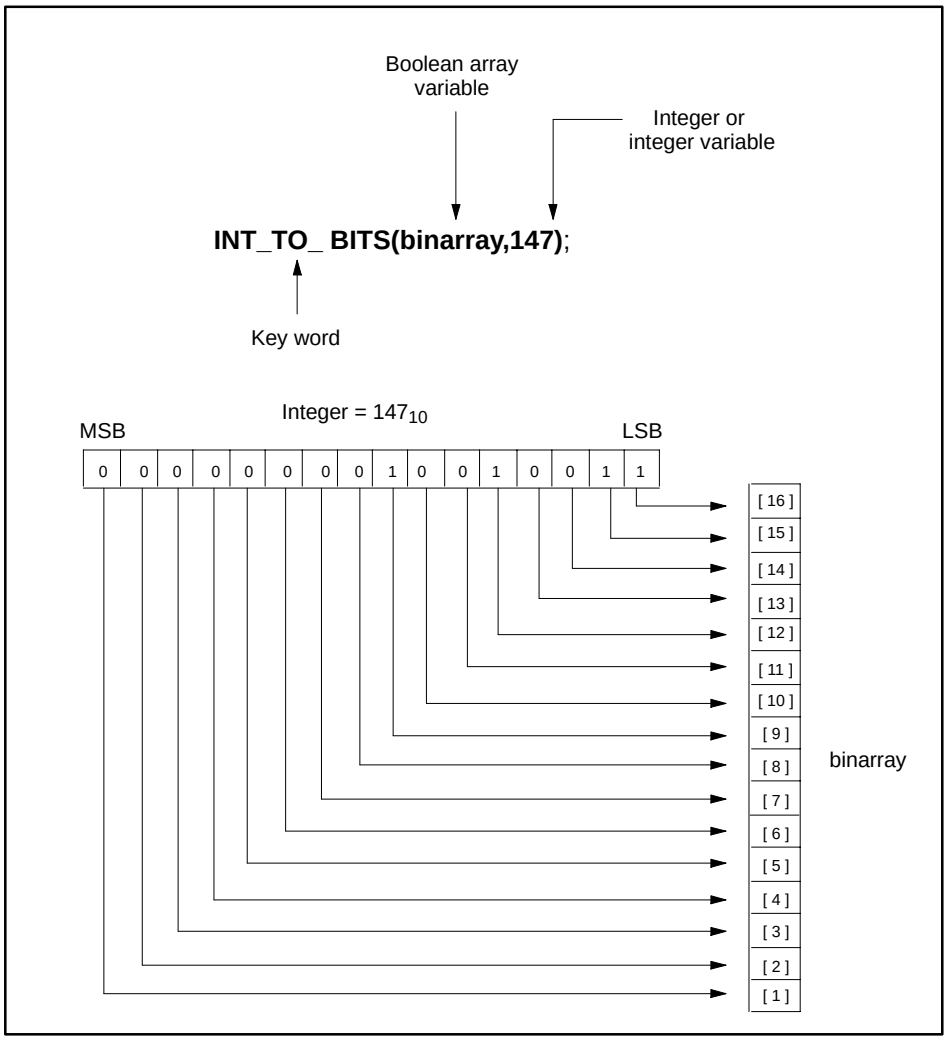


Figure 11-23 Example of INT_TO_BITS Operation

Function and Procedure Definitions (continued)

INT_TO_REAL Function

The INT_TO_REAL function is a real function that is available in STL for S5 controllers and is available only in SFPGM for Series 505 controllers. INT_TO_REAL can be used with the S5 controller family and the 545, 545L, 555, 565, 565T/565P, and 575 controllers. INT_TO_REAL converts an integer to a real number.

Converting a value from an integer to a real number does not change its value; it does, however, allow you to use that value in a function that requires real input values.

To enter the INT_TO_REAL function, use:

INT_TO_REAL(expression)

- The expression in parentheses must evaluate to an integer.
- The returned value is a real number. For example, in Figure 11-24, the value of the variable, **real2**, is 25.0.

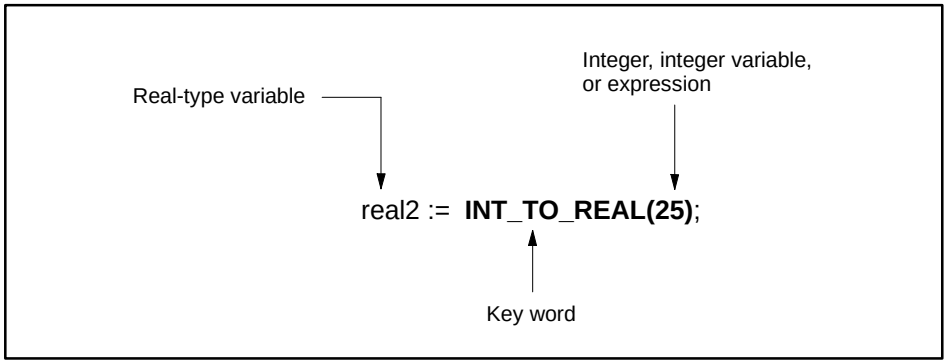


Figure 11-24 INT_TO_REAL Function

IREAD Procedure

The IREAD procedure is available in STL for S5 controllers and is available only in RLL for Series 505 controllers. It can be used with the S5 controller family and the 545, 545L, 555, and 575 controllers. IREAD performs an immediate read of a digital or word input from a module to the image register. Any reference to the associated I/O point gets the updated field value. Refer to the *SIMATIC TI505 Programming Reference Manual* for a list of Series 505 modules that support this procedure. All digital input modules for S5 support the IREAD procedure.

Use the following format to enter the IREAD procedure.

IREAD(Variable)

- The variable in parentheses must be a digital or word input from a supported module in base 0 for Series 505 controllers; the module can be located in any base for an S5 controller. See the example in Figure 11-25.

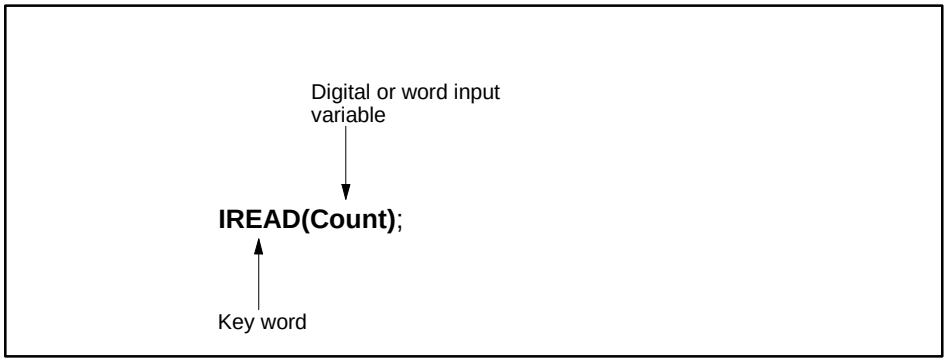


Figure 11-25 IREAD Procedure

⚠ WARNING

For Series 505 controllers, the IREAD procedure is only available for input points of a supported module in base 0 (zero). APT cannot determine whether the input points meet these requirements.

Failure to use the IREAD procedure correctly can cause your Series 505 controller to operate in an unpredictable manner, or to fail in an unsafe condition that could result in death or serious injury and/or equipment damage.

Do not use the IREAD procedure unless you are certain that the input points meet the stated requirements.

Function and Procedure Definitions (continued)

IWRITE Procedure

The IWRITE procedure is available in STL for S5 controllers and is available only in RLL for Series 505 controllers. It can be used with the S5 controller family and with the 545, 545L, 555, and 575 controllers. IWRITE performs an immediate write of a digital or word output to a module from the image register. Any manipulations to the associated I/O point immediately update the field value. Refer to the *SIMATIC TI505 Programming Reference Manual* for a list of Series 505 modules that support this procedure. All digital output modules for S5 support the IWRITE procedure.

Use the following format to enter the IWRITE procedure.

IWRITE(Variable)

- The variable in parentheses must be a digital or word output from a supported module in base 0 for Series 505 controllers; the module can be located in any base for an S5 controller. See the example in Figure 11-26.

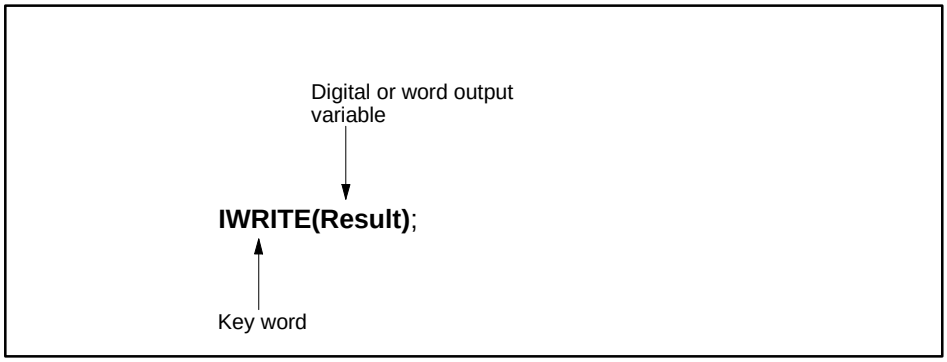


Figure 11-26 IWRITE Procedure

⚠ WARNING

For Series 505 controllers, the IWRITE procedure is only available for output points of a supported module in base 0 (zero). APT cannot determine whether the output points meet these requirements.

Failure to use the IWRITE procedure correctly can cause your Series 505 controller to operate in an unpredictable manner, or to fail in an unsafe condition that could result in death or serious injury and/or equipment damage.

Do not use the IWRITE procedure unless you are certain that the output points meet the stated requirements.

LATCH Procedure

The LATCH procedure is available in STL for S5 controllers and is available only in RLL for Series 505 controllers. It can be used with all controllers. LATCH sets the APT flag value to on (true) after the procedure is executed in the math block. The APT flag remains on, even if the block is disabled, until a CLEAR sets it to off (false).

To enter the LATCH procedure, use:

LATCH (flag_variable);

- The variable in the parentheses is the APT flag that is turned on (true). See the example in Figure 11-27.

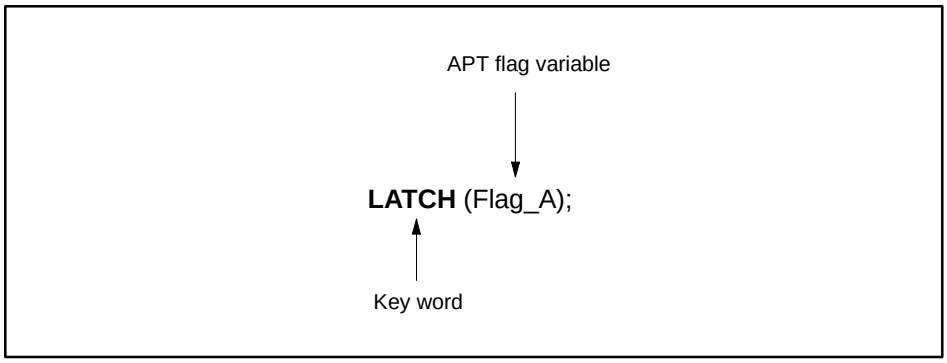


Figure 11-27 LATCH Procedure

Function and Procedure Definitions (continued)

LEAD_LAG Procedure

The LEAD_LAG procedure is available in STL for S5 controllers and is available only in SFPGM for Series 505 controllers. LEAD_LAG can be used with the S5 controller family and the 545, 545L, 555, 565, 565T/565P, and 575 controllers. This procedure calculates an output based on an input and the specified gain, lead, and lag values. Use the procedure only in a continuous function block that includes a sample time ("sample" math block).

To enter the LEAD_LAG procedure shown in Figure 11-28, use:

LEAD_LAG(input variable,previous input,output variable,lead,lag,gain);

- The first three variables in parentheses can be real or integer values, but all three must be the same type. Lead, lag, and gain must be real variables or real numbers.
- The input variable is the input value.
- Previous input stores the value of the last input.
- The output variable is the variable that stores the result of the LEAD_LAG procedure.
- Lead (T_{Lead}) specifies a time in minutes that is associated with the response immediately after a change in the input.
- Lag (T_{Lag}) specifies the time in minutes required to reach 63.2 percent of the final output after a change in the input.
- Gain specifies the ratio of the change in output to the change in input at a steady state, as shown in the following equation.

$$Gain = \frac{\Delta\ output}{\Delta\ input}$$

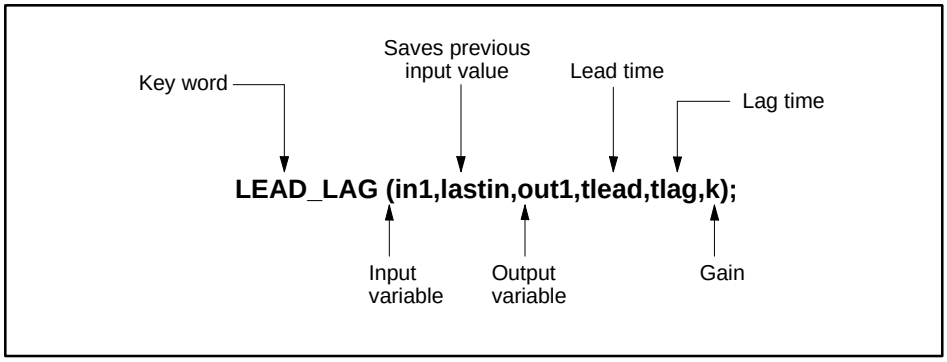


Figure 11-28 LEAD_LAG Procedure

The LEAD_LAG algorithm uses the following equation.

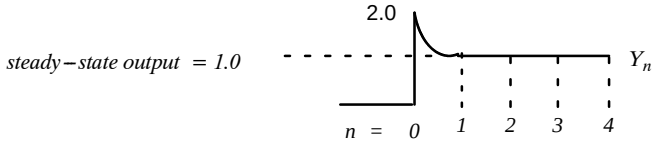
$$Y_n = \left(\frac{T_{Lag}}{T_{Lag} + T_s}\right) Y_{n-1} + Gain\left(\frac{T_{Lead} + T_s}{T_{Lag} + T_s}\right) X_n - Gain\left(\frac{T_{Lead}}{T_{Lag} + T_s}\right) X_{n-1}$$

where Y_n = *present output,* Y_{n-1} = *previous output,*
X_n = *present input,* X_{n-1} = *previous input, and*
T_s = *sample time in minutes.*

The output depends on the ratio of lead to lag as explained below. Assume the following values in each example: Δ *input and gain* = *1.0*

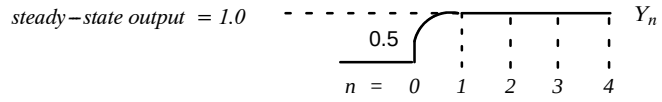
If T_{Lead} / T_{Lag} is greater than 1.0, then the initial response overshoots the steady-state output value.

$$Initial\ output = \Delta input * Gain\left(\frac{T_{Lead}}{T_{Lag}}\right) = 1.0 * 1.0\left(\frac{2.0}{1.0}\right) = 2.0$$



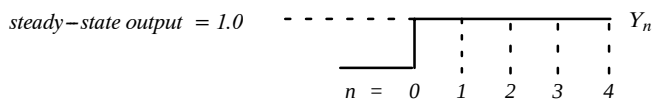
If T_{Lead} / T_{Lag} is less than 1.0, then the initial response undershoots the steady-state output value.

$$Initial\ output = \Delta input * Gain\left(\frac{T_{Lead}}{T_{Lag}}\right) = 1.0 * 1.0\left(\frac{1.0}{2.0}\right) = 0.5$$



If T_{Lead} / T_{Lag} is equal to 1.0, then the initial response instantaneously reaches the steady-state output value.

$$Initial\ output = \Delta input * Gain\left(\frac{T_{Lead}}{T_{Lag}}\right) = 1.0 * 1.0\left(\frac{1.0}{1.0}\right) = 1.0$$



Function and Procedure Definitions (continued)

LEFTSHIFT Function

The LEFTSHIFT function is an integer function that is available in STL for S5 controllers and is available only in SFPGM for Series 505 controllers. LEFTSHIFT can be used with the S5 controller family and the 545, 545L, 555, 565, 565T/565P, and 575 controllers. LEFTSHIFT moves the bits of an integer to the left (from the least significant bit toward the most significant bit).

To enter the LEFTSHIFT function, use:

LEFTSHIFT(expression,expression)

- The first expression is the integer that contains the bits that you want to shift to the left.
- The second expression must evaluate to an integer that specifies the number of positions to the left that you want to shift the bits. For S5 controllers, this must be a literal number, not an expression.
- The returned integer value is the decimal equivalent of the binary value that results from the shifting of the bits. For example, in Figure 11-29, the value of the variable, **int3**, is 1176.
- The vacated bits on the right are set to 0.

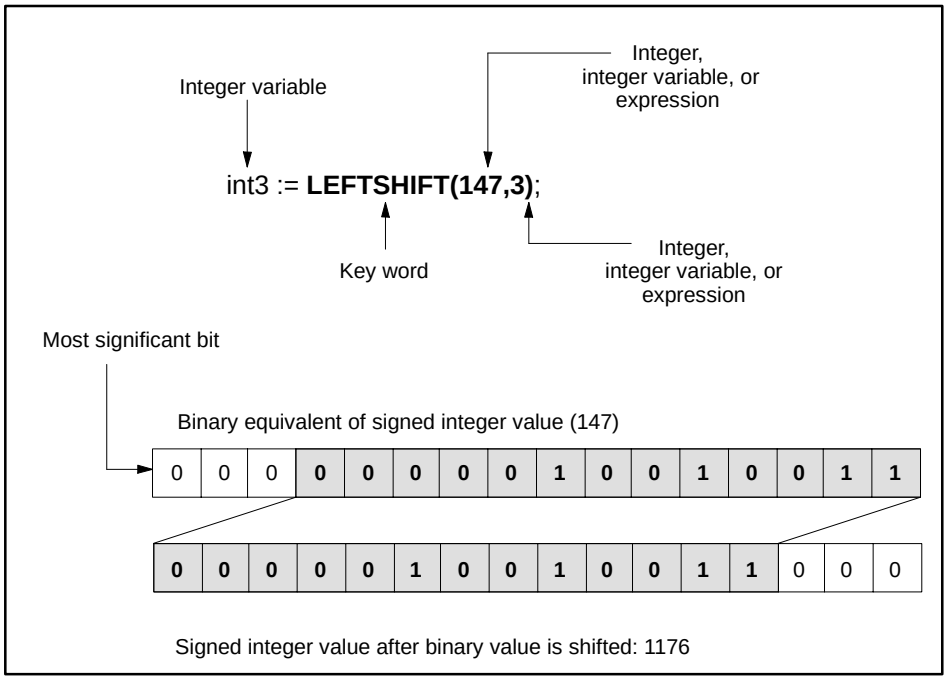


Figure 11-29 LEFTSHIFT Function

LIMIT Procedure

The LIMIT procedure is available in STL for S5 controllers and is available only in SFPGM for Series 505 controllers. LIMIT can be used with the S5 controller family and the 545, 545L, 555, 565, 565T/565P, and 575 controllers. The LIMIT procedure restricts the value of a variable. LIMIT copies the input value to the output variable and ensures that the output is between the specified high and low limits.

To enter the LIMIT procedure, use:

LIMIT(low-limit variable,high-limit variable,input variable,output variable);

- The input and output variables and the limit variables can be either real or integer values; but all of the values in the parentheses must be the same type. For example, if the input is a real value, the output variable and limit variables must also be real.
- If the input is greater than or equal to the low limit and less than or equal to the high limit, the output is assigned the input value. For example, in Figure 11-30, if you assume the values under each variable in the first procedure, **test1** is 4.2.
- If the input is greater than the high limit, the output is assigned the high-limit value.
- If the input is less than the low limit, the output is assigned the low-limit value. For example, in Figure 11-30, if you assume the values under each variable in the second procedure, **test2** is –3.

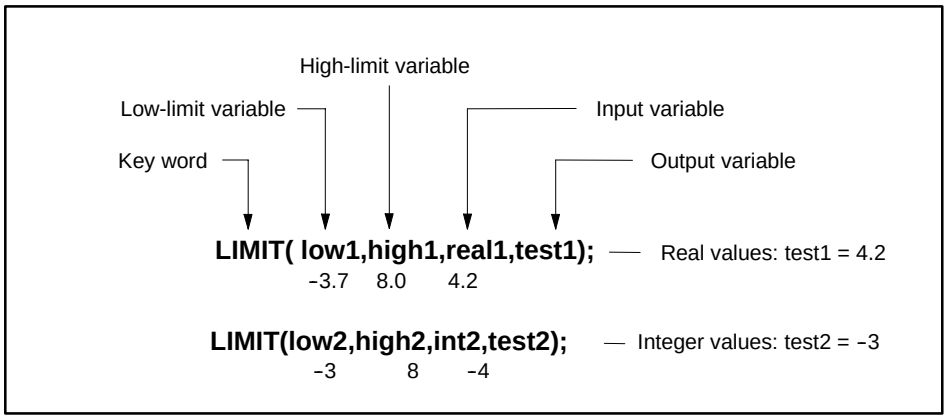


Figure 11-30 LIMIT Procedure

Function and Procedure Definitions (continued)

LOAD_ARRAY Procedure

The LOAD_ARRAY procedure is available in STL for S5 controllers and is available in both RLL and SFPGM for Series 505 controllers. LOAD_ARRAY can be used with all controllers. LOAD_ARRAY assigns a value to the elements of an array.

To enter the LOAD_ARRAY procedure, use:

LOAD_ARRAY(input variable,array variable);

- The variables in parentheses can be either real or integer, but both must be the same type. For example, if the input variable is a real value, the array variable must also be real.
- The LOAD_ARRAY procedure assigns the value of the input variable to each element in the array. See the examples in Figure 11-31.

For Series 505 controllers, the LOAD_ARRAY procedure can accommodate integer arrays containing as many as 32,767 elements, and real arrays containing up to 16,383 elements. For S5 controllers, you can use integer arrays of up to 256 elements, and real arrays of up to 128 elements.

Figure 11-31 shows how the LOAD_ARRAY procedure works.

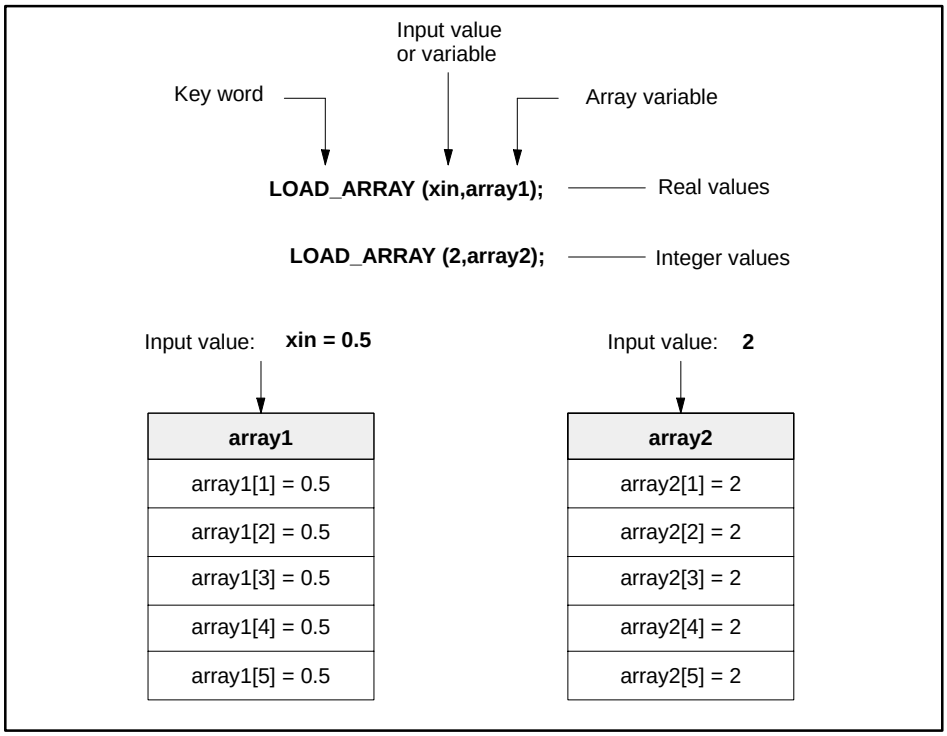


Figure 11-31 LOAD_ARRAY Procedure

Function and Procedure Definitions (continued)

LOOKUP_TABLE Procedure

The LOOKUP_TABLE procedure is available in STL for S5 controllers and is available only in SFPGM for Series 505 controllers. LOOKUP_TABLE can be used with the S5 controller family and the 545, 545L, 555, 565, 565T/565P, and 575 controllers. LOOKUP_TABLE determines an output value based on the relative position of an input value in an array of values.

The LOOKUP_TABLE procedure compares the input to the values in the input array and uses one of the following techniques to determine the output.

- If the input is equal to an element of the input array, the output is assigned the value of the same element in the output array.
- If the input is not equal to an element in the input array, LOOKUP_TABLE uses the value of the highest element that is not greater than the input. The output is assigned the value of the same element in the output array.
- If the input is less than the first element in the input array, the output is assigned the value of the first element in the output array.
- If the input is greater than any element in the input array, the output is assigned the value of the last element in the output array.

When you declare the arrays that you want to use in a LOOKUP_TABLE procedure, you must specify 11 as the number of elements in each array.

To enter the LOOKUP_TABLE procedure, use:

LOOKUP_TABLE(input variable,output variable,input array,output array);

- The input variable must be a real variable.
- The output variable is a real variable that stores the result of the LOOKUP_TABLE procedure. See the example in Figure 11-32.
- The input array must be an 11-element array variable and must contain real numbers. Values in the array must be arranged in ascending order.
- The output array must be an 11-element array variable and must contain real numbers. Values in the array must be arranged in the same order as the input array.

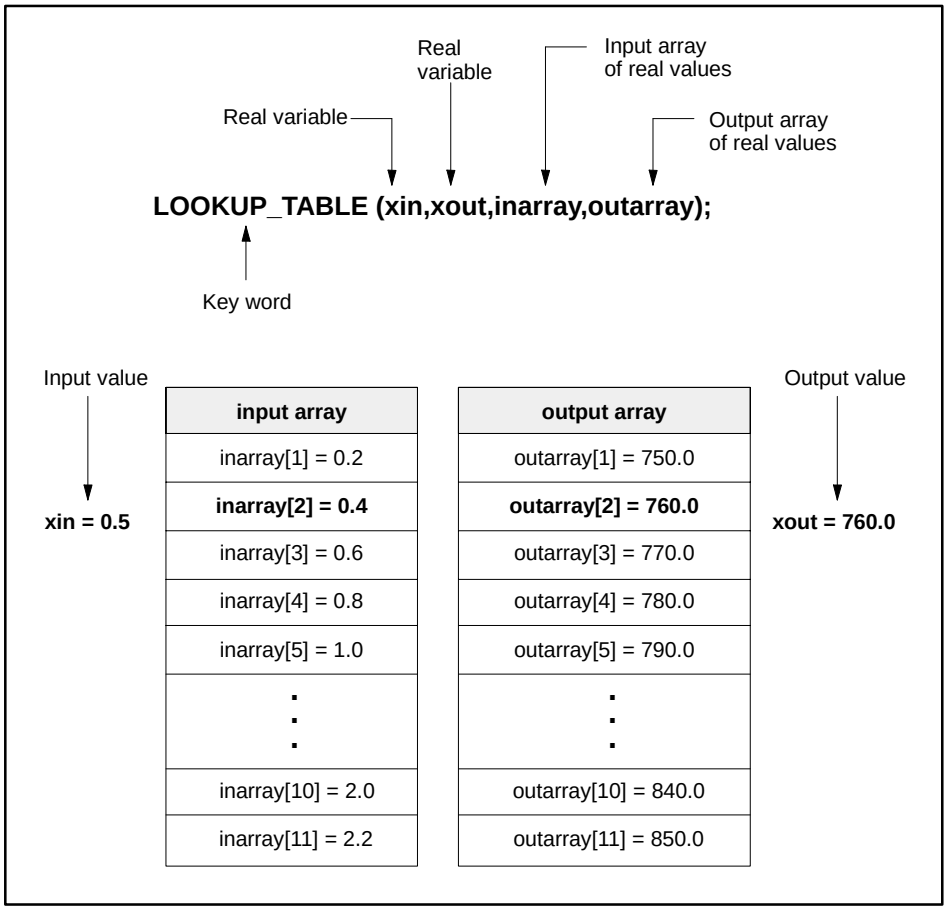


Figure 11-32 LOOKUP_TABLE Procedure

Function and Procedure Definitions (continued)

LN Function

The LN function is a real function that is available in STL for S5 controllers and is available only in SFPGM for Series 505 controllers. LN can be used with the S5 controller family and the 545, 545L, 555, 565, 565T/565P, and 575 controllers. LN returns the natural logarithm (base e) of the expression in parentheses. The LN function is the inverse of the EXP function.

To enter the LN function, use:

LN(expression)

- The expression in parentheses must evaluate to a real number.
- The returned value is also a real number. For example, in Figure 11-33, the value of the variable, **real2**, is 1.2.

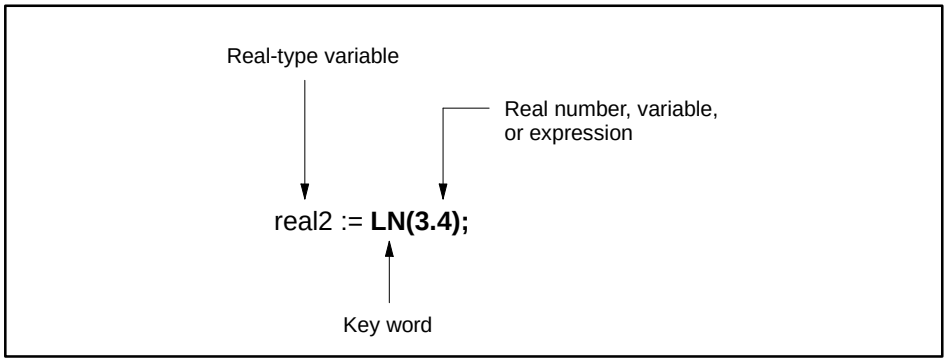


Figure 11-33 LN Function

LOG Function

The LOG function is a real function that is available in STL for S5 controllers and is available only in SFPGM for Series 505 controllers. LOG can be used with the S5 controller family and the 545, 545L, 555, 565, 565T/565P, and 575 controllers. LOG returns the common logarithm (base 10) of the expression in parentheses.

To enter the LOG function, use:

LOG(expression)

- The expression in parentheses must evaluate to a real number.
- The returned value is also a real number. For example, in Figure 11-34, the value of the variable, **real3**, is 2.0.

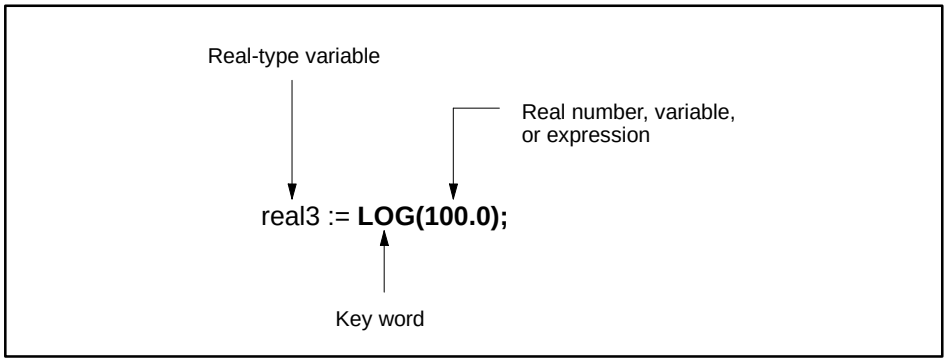


Figure 11-34 LOG Function

Function and Procedure Definitions (continued)

MAX Procedure

The MAX procedure is available in STL for S5 controllers and is available only in SFPGM for Series 505 controllers. It can be used with the S5 controller family and the 545, 545L, 555, 565, 565T/565P, and 575 controllers. MAX is used to track the maximum value of a variable over time. When the MAX procedure executes, the input value is assigned to the maximum value. As a result, when the value of the input changes, the maximum value can also change. See the example in Figure 11-35.

To enter the MAX procedure, use:

MAX(input variable,maximum variable);

- The variables in parentheses can be either real or integer, but both values in the parentheses must be the same type. For example, if the input is a real value, the maximum value must also be real.
- If the new input value is greater than the current maximum, this new input is assigned to the maximum value.

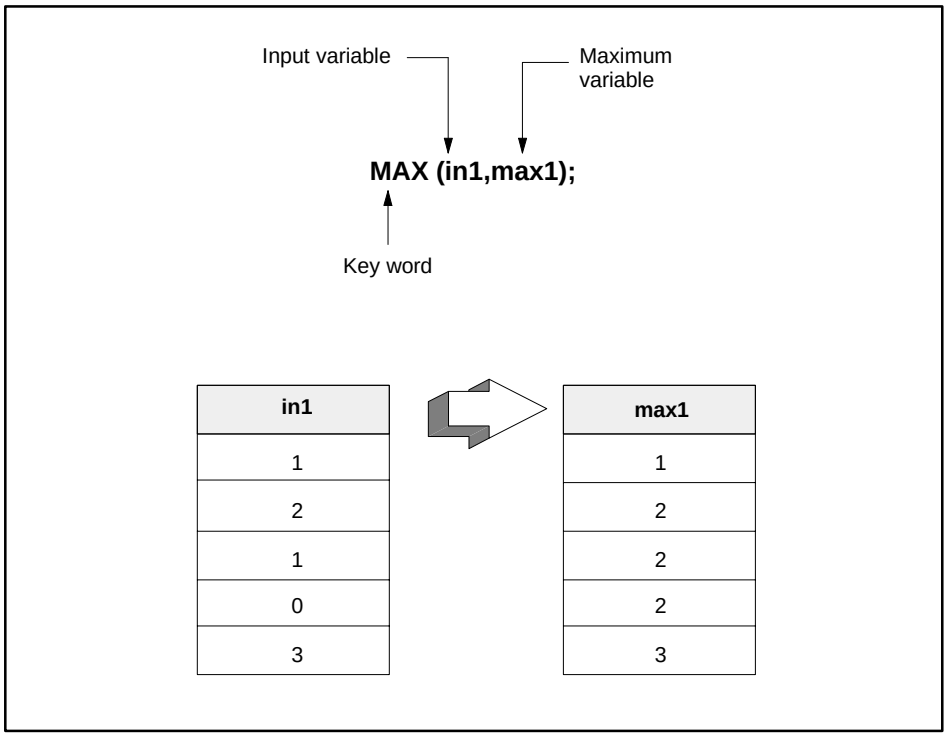


Figure 11-35 MAX Procedure

MIN Procedure

The MIN procedure is available in STL for S5 controllers and is available only in SFPGM for Series 505 controllers. It can be used with the S5 controller family and the 545, 545L, 555, 565, 565T/565P, and 575 controllers. MIN is used to track the minimum value of a variable over time. When the MIN procedure executes, the input value is assigned to the minimum value. As a result, when the value of the input changes, the minimum value can also change. See the example in Figure 11-36.

To enter the MIN procedure, use:

MIN(input variable,minimum variable);

- The variables in parentheses can be either real or integer; but both values in the parentheses must be the same type. For example, if the input is a real value, the minimum value must also be real.
- If the new input value is less than the current minimum, this new input is assigned to the minimum value.

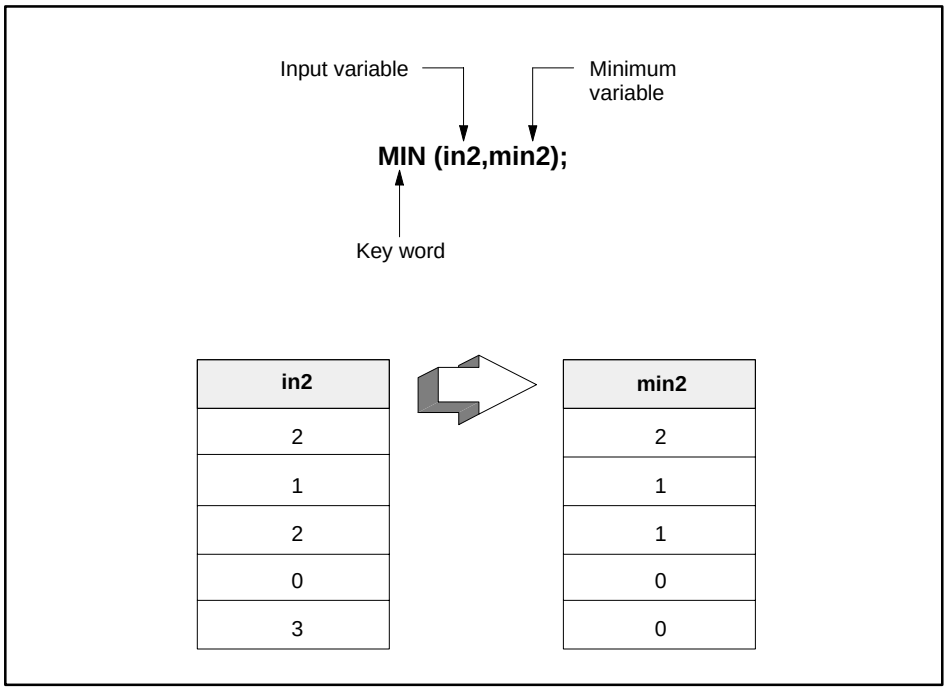


Figure 11-36 MIN Procedure

Function and Procedure Definitions (continued)

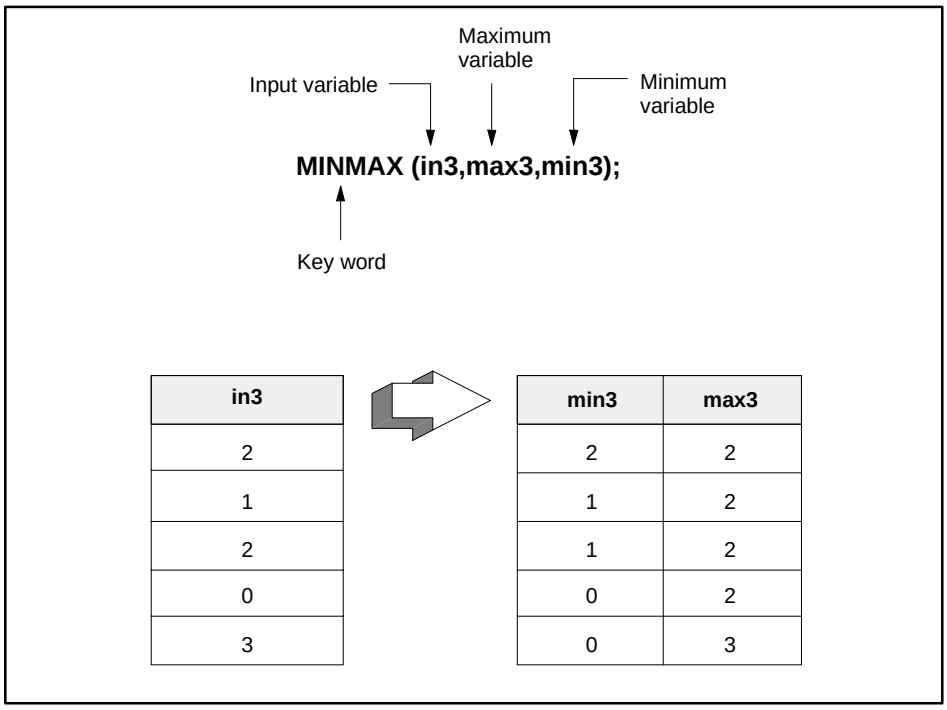
MINMAX Procedure

The MINMAX procedure is available in STL for S5 controllers and is available only in SFPGM for Series 505 controllers. It can be used with the S5 controller family and the 545, 545L, 555, 565, 565T/565P, and 575 controllers. MINMAX is used to track the minimum and maximum values of a variable. When the MINMAX procedure executes, the input value is assigned to the minimum and maximum values. As a result, when the input value changes, the minimum and maximum values can also change. See the example in Figure 11-37.

To enter the MINMAX procedure, use:

MINMAX(input variable,maximum variable,minimum variable);

- The variables in parentheses can be either real or integer, but all of the values in the parentheses must be the same type. For example, if the input is a real value, the minimum and maximum values must also be real.
- If the new input value is less than the current minimum, this new input is assigned to the minimum value.
- If the new input value is greater than the current maximum, this new input is assigned to the maximum value.



in3
2
1
2
0
3

min3	max3
2	2
1	2
1	2
0	2
0	3

Figure 11-37 MINMAX Procedure

ON Procedure

The ON procedure is available in STL for S5 controllers and is available only in RLL for Series 505 controllers. It can be used with all controllers. ON sets the APT flag value to on (true) only as long as the procedure is executed. The APT flag value is off (false) when the procedure is not executed. Do not use ON in Sampled or Event math CFBs.

To enter the ON procedure, use:

ON (flag_variable);

The variable in the parentheses is the APT flag that is turned on (true). See the example in Figure 11-38.

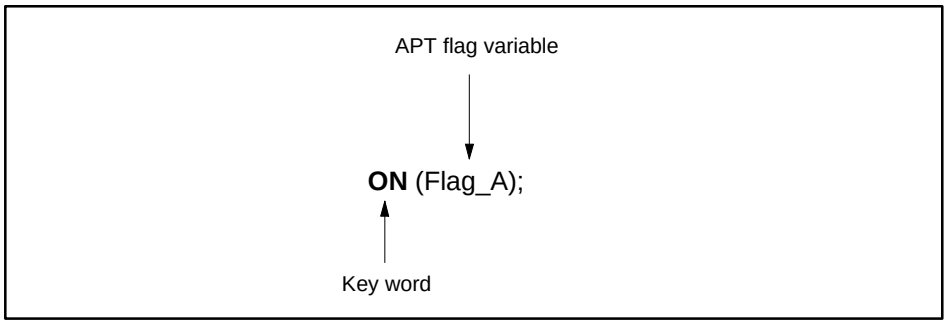


Figure 11-38 ON Procedure

Do not assign an initial value when you declare an APT flag that you intend only to use in an ON procedure. The value of the APT flag is always zero outside the step or CFB in which the ON procedure is located.

Function and Procedure Definitions (continued)

PACK_BITS Procedure

The PACK_BITS procedure is available in STL for S5 controllers and is available only in RLL for Series 505 controllers. It can be used with all controllers. PACK_BITS shifts the values in a boolean array into an integer variable.

When you define the boolean array for this procedure in the Declaration Table, the size of the array that you specify determines the number of bits moved into the integer variable.

To enter the PACK_BITS procedure, use:

PACK_BITS(variable,variable);

- The first variable in parentheses is a boolean array that can contain up to 16 elements. The first element of the array becomes the least significant bit of the integer variable. See the example in Figure 11-39.
- The second variable is the integer that receives the elements of the boolean array. All bits in the variable are set to 0 before the values of the array are shifted into the integer.

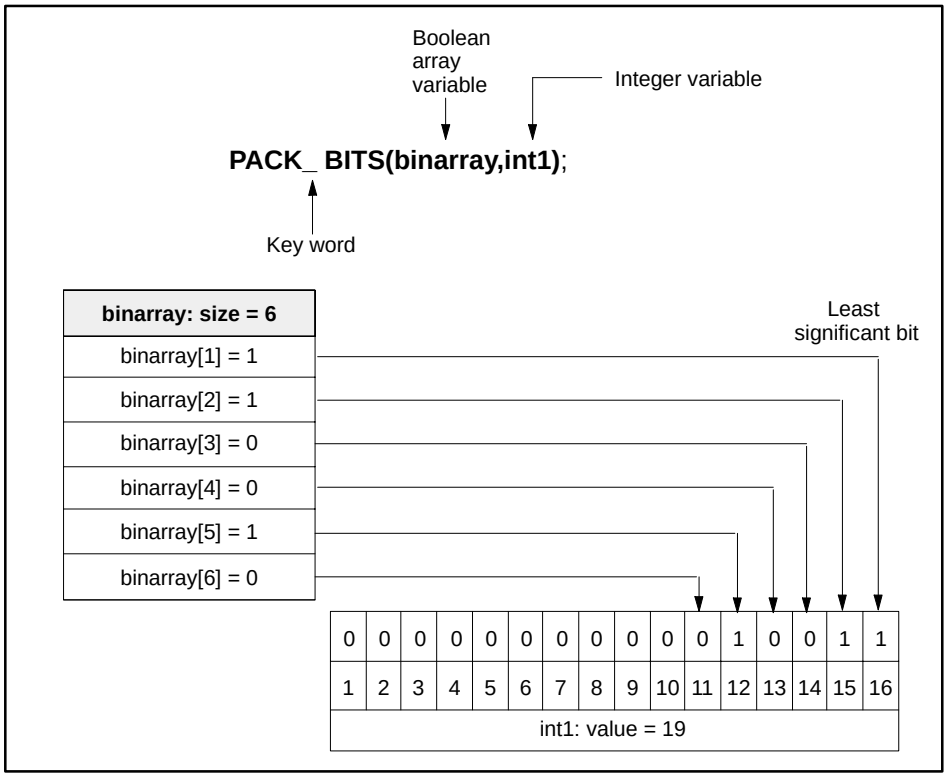


Figure 11-39 PACK_BITS Procedure

PBITS_TO_INT Procedure

The PBITS_TO_INT procedure is available in STL for S5 controllers and is available only in SFPGM and subroutines for Series 505 controllers. PBITS_TO_INT can be used with the S5 controller family and the 545, 545L, 555, 565, 565T/565P, and 575 controllers. PBITS_TO_INT moves an array of 16 boolean values into an integer variable.

To enter the PBITS_TO_INT procedure, use:

PBITS_TO_INT(variable, integer)

- The variable in parentheses must be a boolean array of length 16. (A boolean value of false is a 0; true is a 1.) Element 1 of the array is the most significant bit; element 16 is the least significant bit.
- The integer value is the decimal equivalent of the binary value constructed from the bit values of the boolean array. For example, in Figure 11-40 the value of the integer **binvalue** is 147.

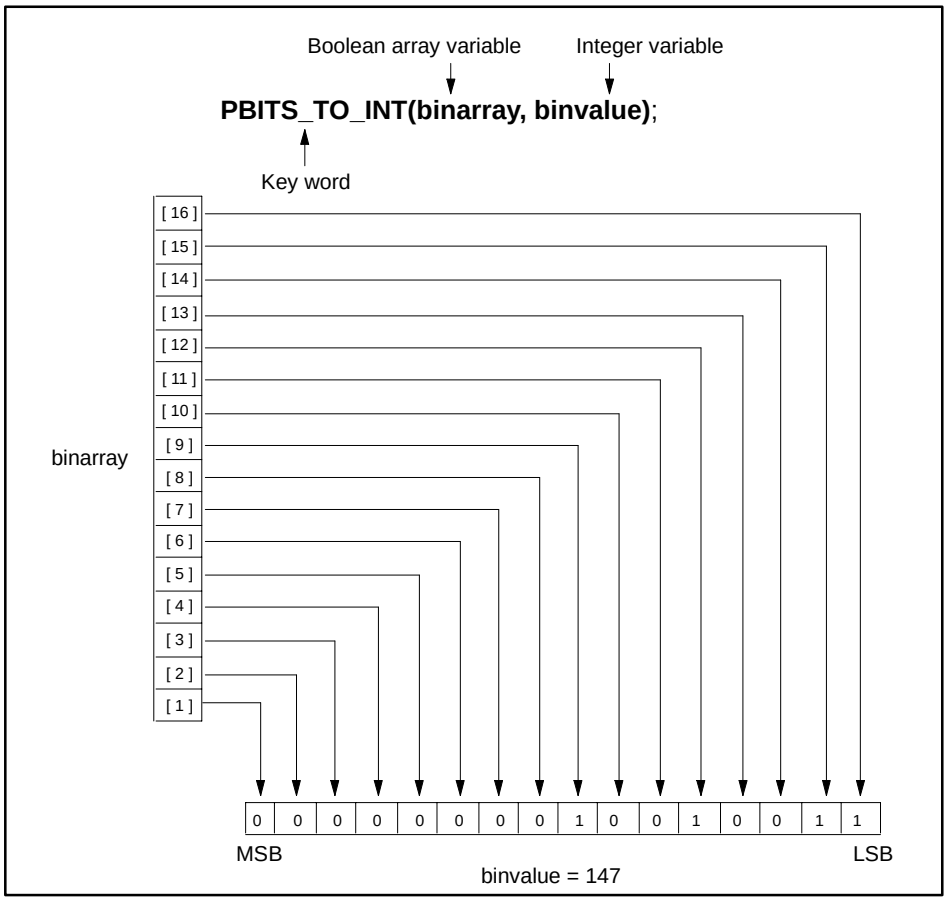


Figure 11-40 Example of PBITS_TO_INT Operation

Function and Procedure Definitions (continued)

PROUND Procedure

The PROUND procedure is available in STL for S5 controllers and is available only in SFPGM for Series 505 controllers. PROUND can be used with the S5 controller family and the 545, 545L, 555, 565, 565T/565P, and 575 controllers. PROUND changes a real number to the nearest integer. Values with the decimal portion equal to or greater than .50 are rounded to the higher integer; values less than .50 are rounded to the lower integer.

For Series 505 controllers, the PROUND procedure can be used in user-defined subroutines, whereas the ROUND function cannot. S5 controllers can use either PROUND or ROUND in user-defined subroutines.

To enter the PROUND procedure, use:

PROUND(integer, real variable)

- The real variable in parentheses must be a real number.
- The PROUND arguments can be variables, but not expressions.
- The integer value is the rounded result of the real variable. For example, in Figure 11-41, the value of the variable, **int2**, is 4 when the variable **temp_1** is 3.5.

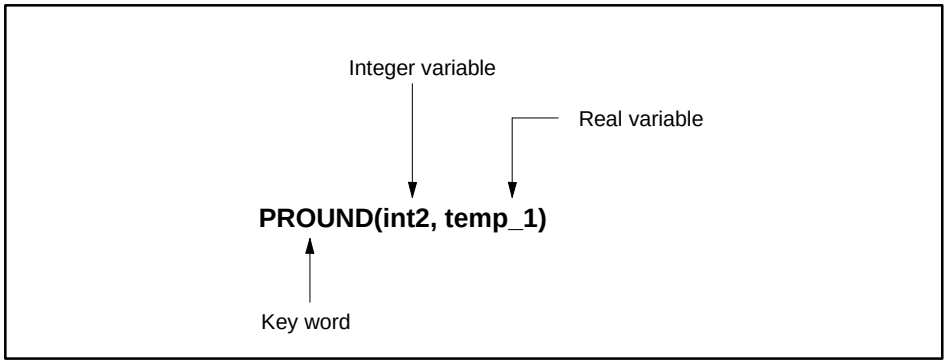


Figure 11-41 PROUND Procedure

PTRUNC Procedure

The PTRUNC procedure is available in STL for S5 controllers and is available only in SFPGM for Series 505 controllers. PTRUNC can be used with the S5 controller family and the 545, 545L, 555, 565, 565T/565P, and 575 controllers. PTRUNC returns the integer portion of the variable in parentheses; that is, it truncates a real number by eliminating its fractional portion and returning only its integer portion.

For Series 505 controllers, the PTRUNC procedure can be used in user-defined subroutines, whereas the TRUNC function cannot. S5 controllers can use either PTRUNC or TRUNC in user-defined subroutines.

To enter the PTRUNC procedure, use:

PTRUNC(integer, real variable)

- The real variable in parentheses must be a real number.
- The PTRUNC arguments can be variables, but not expressions.
- The returned value is the truncated result of the real variable. For example, in Figure 11-42, the value of the variable, **int2**, is 2 when the variable **temp_1** is 2.718.

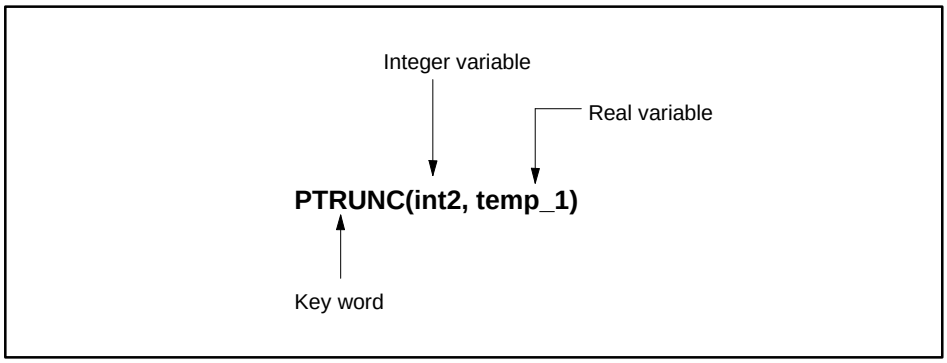


Figure 11-42 PTRUNC Procedure

Function and Procedure Definitions (continued)

RIGHTSHIFT Function

The RIGHTSHIFT function is an integer function that is available in STL for S5 controllers and is available only in SFPGM for Series 505 controllers. RIGHTSHIFT can be used with the S5 controller family and the 545, 545L, 555, 565, 565T/565P, and 575 controllers. RIGHTSHIFT moves the bits of an integer to the right (from the most significant bit toward the least significant bit).

To enter the RIGHTSHIFT function, use:

RIGHTSHIFT(expression,expression)

- The first expression is the integer that contains the bits that you want to shift to the right.
- The second expression must evaluate to an integer that specifies the number of positions to the right that you want to shift the bits. For S5 controllers, the second expression must be a literal number.
- The returned integer value is the decimal equivalent of the binary value that results from the shifting of the bits. For example, in Figure 11-43, the value of the variable, **int3**, is -4078.
- The vacated bits on the left are set to the value of the most significant (sign) bit.

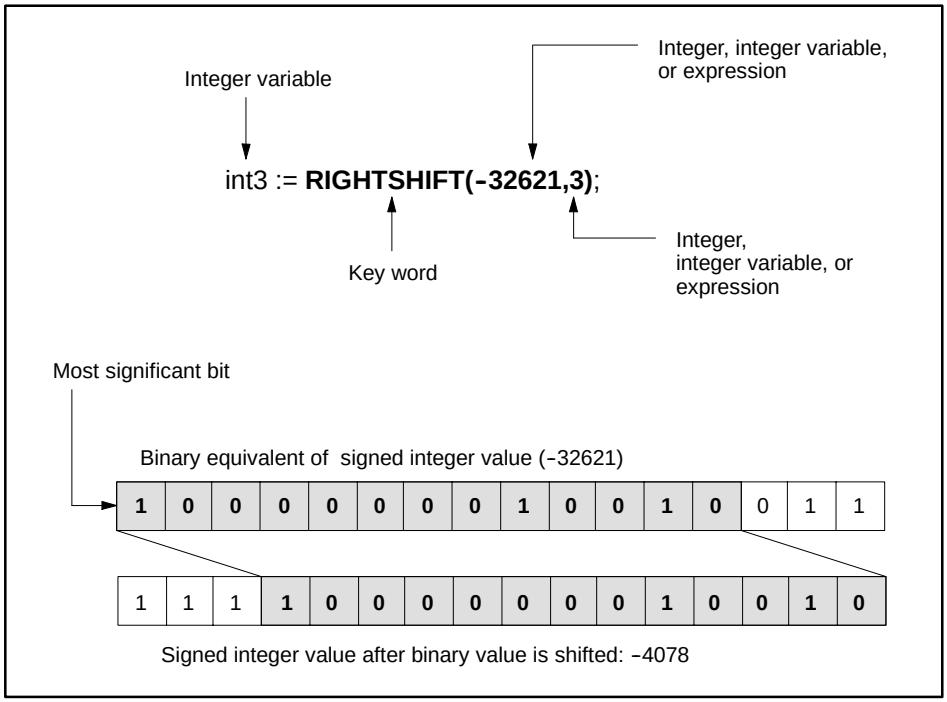


Figure 11-43 RIGHTSHIFT Function

ROUND Function

The ROUND function is an integer function that is available in STL for S5 controllers and is available only in SFPGM for Series 505 controllers. ROUND can be used with the S5 controller family and the 545, 545L, 555, 565, 565T/565P, and 575 controllers. ROUND changes a real number to the nearest integer. Values with the decimal portion equal to or greater than .50 are rounded to the higher integer; values less than .50 are rounded to the lower integer.

NOTE: For Series 505 controllers, the ROUND function cannot be used in user-defined subroutines. Use the PROUND procedure instead. S5 controllers can use ROUND function in user-defined subroutines.

To enter the ROUND function, use:

ROUND(expression)

- The expression in parentheses must evaluate to a real number.
- The returned value is an integer. For example, in Figure 11-44, the value of the variable, **int2**, is 4.

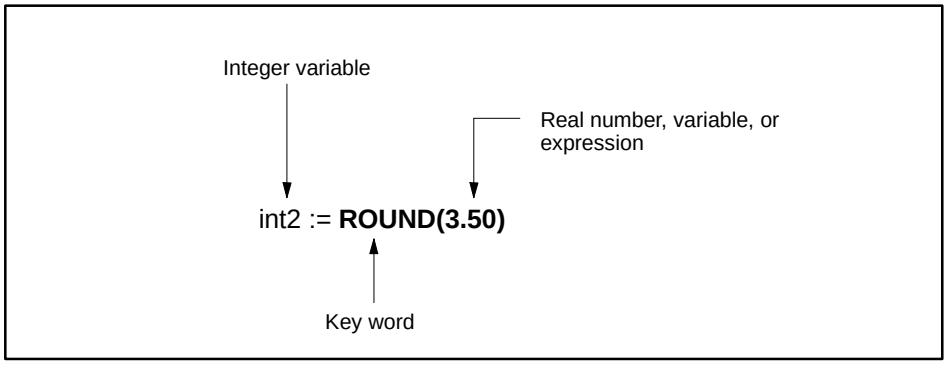


Figure 11-44 ROUND Function

Function and Procedure Definitions (continued)

SCALE Procedure

The SCALE procedure is available in STL for S5 controllers and is available only in SFPGM for Series 505 controllers. SCALE can be used with the S5 controller family and the 545, 545L, 555, 565, 565T/565P, and 575 controllers. SCALE converts an integer to a real value between the low and high limits that you specify.

The SCALE procedure anticipates an input that has the same format as an analog input. If you are not scaling an analog input, consider using the scale CFB.

The integer input can be any one of the following types.

Bipolar The valid range for a Series 505 integer input is –32,000 to 32,000. The valid range for an S5 integer input is –2048 to 2048.

Twenty-percent offset The valid range for a Series 505 integer input is 6,400 to 32,000. The valid range for an S5 integer input is 512 to 2560.

Zero bias The valid range for a Series 505 integer input is 0 to 32,000. The valid range for an S5 integer input is 0 to 2048.

Use the following format to enter the SCALE procedure.

SCALE(input variable,output variable,type,low limit,high limit);

- The input variable is an integer within the range specified by the type.
- The output variable is the real variable that stores the result of the SCALE procedure. See the examples in Figure 11-45 and Figure 11-47.
- The type is a character: B (Bipolar), T (Twenty-percent offset), or Z (Zero bias).
- The low limit is a real number that specifies the lower boundary of the scaling range.
- The high limit is a real number that specifies the upper boundary of the scaling range.

Function and Procedure Definitions (continued)

Using SCALE for a Series 505 Controller

The operation of the SCALE procedure depends on the type of controller. For a Series 505 controller, SCALE simply takes the input value and converts it from an integer to a real value. See Figure 11-45.

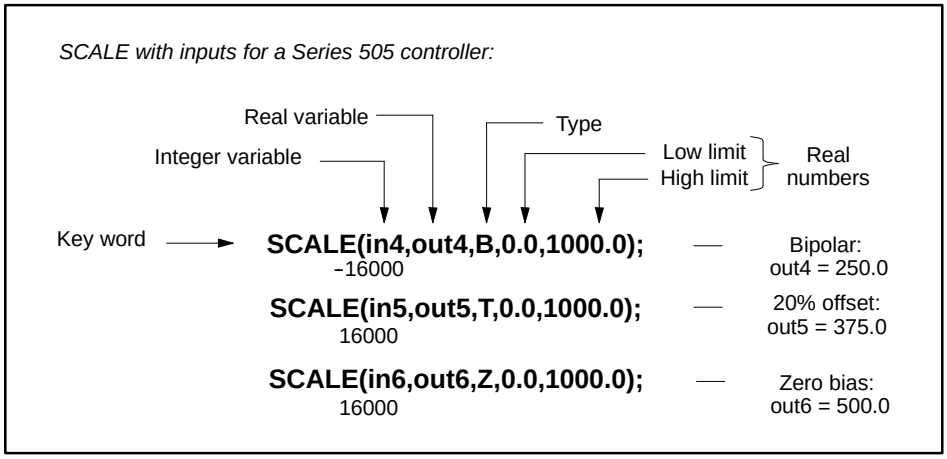


Figure 11-45 SCALE Procedure for Series 505

Using SCALE for an S5 Controller

If you have an S5 controller, the SCALE procedure automatically shifts your input value to the right by 3 bits before scaling it from an integer to a real. This shifting occurs because S5 analog inputs are 16-bit numbers, with 12 bits of resolution, and the 3 least significant bits contain error information. Figure 11-46 shows the bit shift that is performed before your value is scaled.

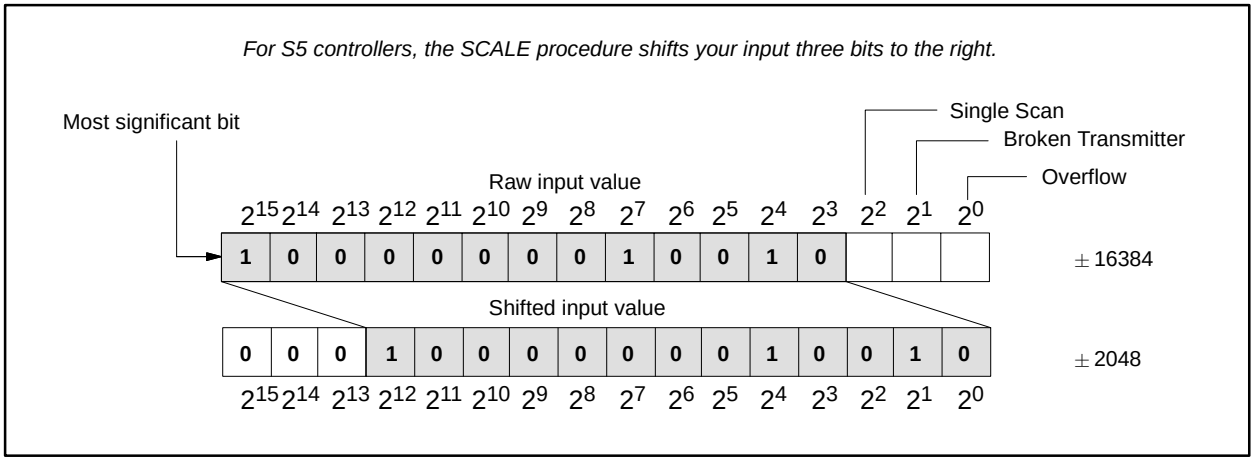
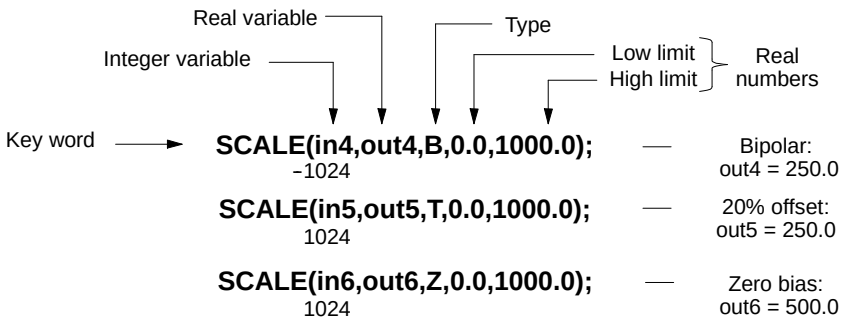


Figure 11-46 S5 Analog I/O Bit Shift

Compensating for the Bit Shift (S5 only)

If you want to scale something other than an analog input, you must compensate for the bit shift that is automatically performed when you use the SCALE procedure. Either use the scale CFB instead, since the CFB does not perform a bit shift, or else multiply your input value by 8 before you use the SCALE procedure. Do not use the LEFTSHIFT procedure; you lose the sign bit Figure 11-47 shows how to use the SCALE procedure for different types of input values if you have an S5 controller.

Scaling an analog input value for an S5 controller.



Scaling an input value that is not analog.
Multiply input by 8 before performing SCALE.

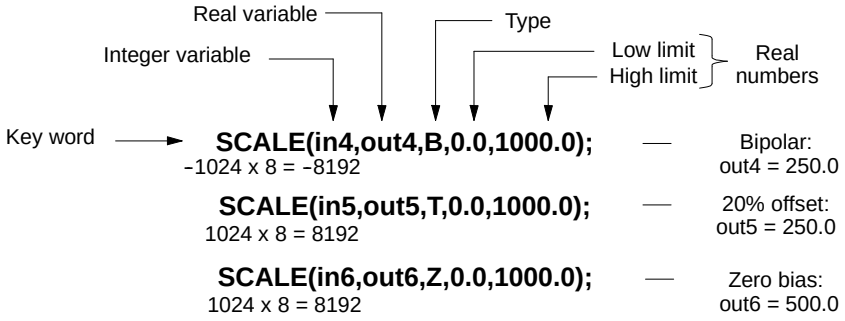


Figure 11-47 SCALE Procedure for S5

NOTE: If you test the SCALE procedure using AI.RAW, you must multiply your input value by 8 before you send it to SCALE. However, if you send AI.RAW to SCALE from an S5 analog input card, the input does not need to be multiplied in order to be read correctly by SCALE.

Function and Procedure Definitions (continued)

SETSSI Procedure

The SETSSI procedure is not supported for S5 controllers; it is available only in RLL for Series 505 controllers. It can be used only with the 560/560T/565/565T/565P controllers. SETSSI sets or resets the 560 scan-inhibit bit in Status Word 1, shown in Figure 11-48. This procedure allows you to control the synchronization of an active controller with a standby controller. See the *SIMATIC TI505 Programming Reference Manual* for additional information about this procedure.

NOTE: The SETSSI procedure is not available on all controllers.

To enter the SETSSI procedure, use:

SETSSI(boolean expression);

- If the boolean expression is true, the scan-inhibit bit is set to 1 and inhibits synchronization.
- If the boolean expression is false, the scan-inhibit bit is reset to 0 and allows synchronization.

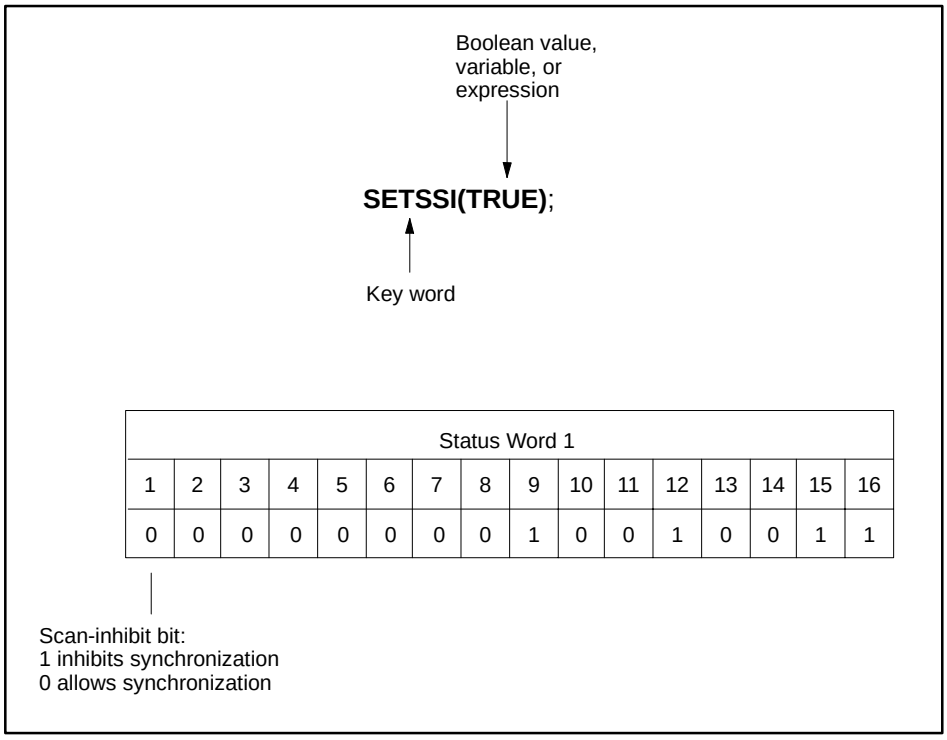


Figure 11-48 SETSSI Procedure

⚠ WARNING

The SETSSI procedure, when active, will not allow the standby controller to synchronize with an active controller in a hot backup configuration. If the active controller fails, the standby unit cannot assume control of the process.

With neither controller online, the process could execute in an unexpected manner that could result in death or serious injury and/or damage to equipment.

Be judicious in your use of the SETSSI procedure, and be sure that your program logic is correct. Minimize the amount of time that the SETSSI procedure is active within your program.

Function and Procedure Definitions (continued)

SIN Function

The SIN function is a real function that is available in STL for S5 controllers and is available only in SFPGM for Series 505 controllers. SIN can be used with the S5 controller family and the 545, 545L, 555, 565, 565T/565P, and 575 controllers. SIN returns the sine of the expression in parentheses.

To enter the SIN function, use:

SIN(expression)

- The expression in parentheses must evaluate to a real number that represents the measure of an angle in radians.
- For S5 controllers, the expression cannot evaluate to a negative number, and it must represent radians in the range 0–2π.
- The returned value is also a real number between –1 and 1. For example, in Figure 11-49, the value of the variable, **angle1**, is 0.50.

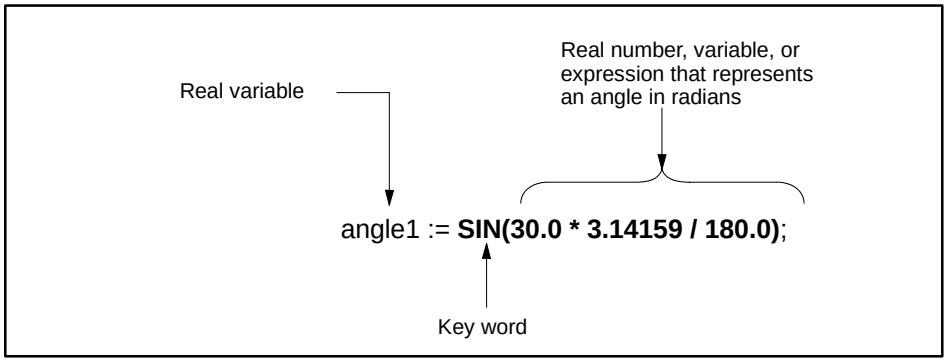


Figure 11-49 SIN Function

SQRT Function

The SQRT function is a real function that is available in STL for S5 controllers and is available only in SFPGM for Series 505 controllers. SQRT can be used with the S5 controller family and the 545, 545L, 555, 565, 565T/565P, and 575 controllers. SQRT returns the square root of the expression in parentheses.

To enter the SQRT function, use:

SQRT(expression)

- The expression in parentheses must evaluate to a positive real number.
- The returned value is also a real number. For example, in Figure 11-50, the value of the variable, **real1**, is 12.0.

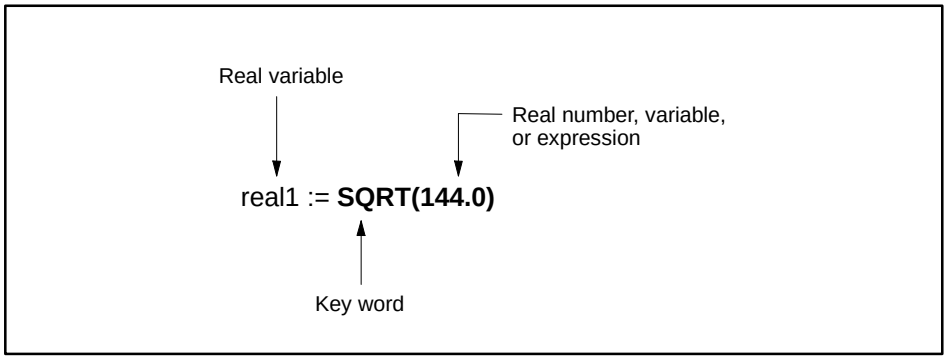


Figure 11-50 SQRT Function

Function and Procedure Definitions (continued)

TAN Function

The TAN function is a real function that is available in STL for S5 controllers and is available only in SFPGM for Series 505 controllers. TAN can be used with the S5 controller family and the 545, 545L, 555, 565, 565T/565P, and 575 controllers. TAN returns the tangent of the expression in parentheses.

To enter the TAN function, use:

TAN(expression)

- The expression in parentheses must evaluate to a real number that represents the measure of an angle in radians.
- For S5 controllers, the expression cannot evaluate to a negative number, and it must represent radians in the range 0–2π.
- The returned value is also a real number. For example, in Figure 11-51, the value of the variable, **angle1**, is 1.0.

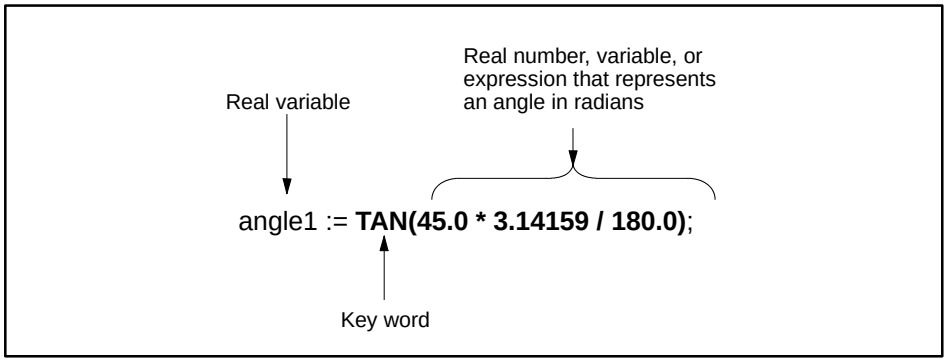


Figure 11-51 TAN Function

TRUNC Function

The TRUNC function is an integer function that is available in STL for S5 controllers and is available only in SFPGM for Series 505 controllers. TRUNC can be used with the S5 controller family and the 545, 545L, 555, 565, 565T/565P, and 575 controllers. TRUNC returns the integer portion of the expression in parentheses; that is, it truncates a real number by eliminating its fractional portion and returning only its integer portion.

NOTE: For Series 505 controllers, the TRUNC function cannot be used in user-defined subroutines. Use the PTRUNC procedure instead. S5 controllers can use TRUNC function in user-defined subroutines.

To enter the TRUNC function, use:

TRUNC(expression)

- The expression in parentheses must evaluate to a real number.
- The returned value is an integer. For example, in Figure 11-52, the value of the variable, **int2**, is 2.

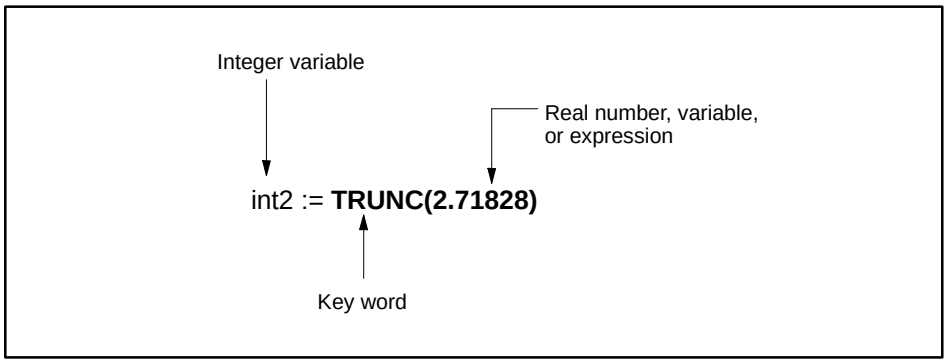


Figure 11-52 TRUNC Function

Function and Procedure Definitions (continued)

UNPACK_BITS Procedure

The UNPACK_BITS procedure is available in STL for S5 controllers and is available only in RLL for Series 505 controllers. It can be used with all controllers.

When you define the boolean array for this procedure in the Declaration Table, the size of the array that you specify determines the number of bits that are moved from the integer variable.

To enter the UNPACK_BITS procedure, use:

```
UNPACK_BITS(variable,variable);
```

- The first variable is a boolean array that can contain up to 16 elements. The least significant bit becomes element 1 of the array. See the example in Figure 11-53.
- The second variable is the integer that contains the bits to be moved.

Figure 11-53 shows how UNPACK_BITS shifts a specified number of bits from an integer into a boolean array.

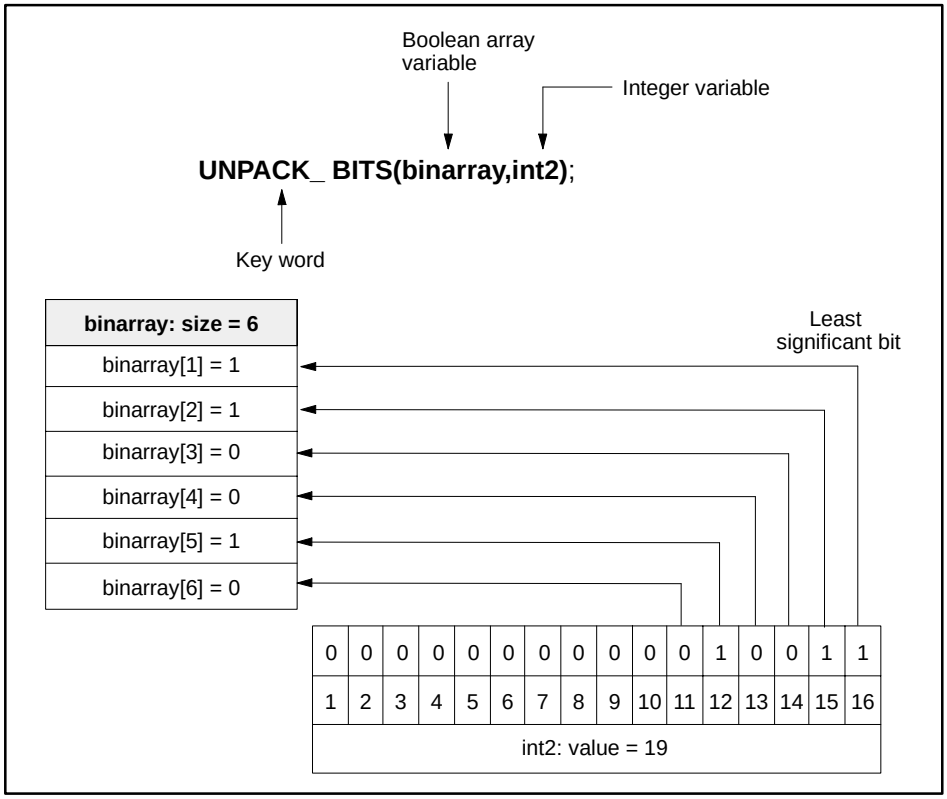


Figure 11-53 UNPACK_BITS Procedure

Function and Procedure Definitions (continued)

UNSCALE Procedure

The UNSCALE procedure is available in STL for S5 controllers and is available only in SFPGM for Series 505 controllers. UNSCALE can be used with the S5 controller family and the 545, 545L, 555, 565, 565T/565P, and 575 controllers. UNSCALE converts a real value, one that falls within specified low and high limits, to a scaled integer. The scaled integer output can be any one of the following types.

Bipolar For a Series 505 controller, bipolar unscales the real value to an integer between –32,000 and 32,000. For an S5 controller, bipolar unscales the real value to an integer between –1024 and 1024.

Twenty-percent offset For a Series 505 controller, twenty-percent offset unscales the real value to an integer between 6,400 and 32,000. This type is not available for S5 controllers.

Zero bias For a Series 505 controller, zero bias unscales the real value to an integer between 0 and 32,000. For an S5 controller, zero bias unscales the real value to an integer between 0 and 1024.

NOTE: If your S5 analog output module has a built-in twenty-percent offset, you must select zero bias for UNSCALE. If your S5 analog output module does not have a twenty-percent offset built in, you can choose between zero bias and bipolar; you still cannot use twenty-percent offset with UNSCALE.

To enter the UNSCALE procedure, use:

```
UNSCALE(input variable,output variable,type,low limit,high limit);
```

- The input variable is a real value within the range specified by the low and high limits.
- The output variable is the integer variable that stores the result of the UNSCALE procedure.
- The type is a character: B (Bipolar), T (Twenty-percent offset), or Z (Zero bias).
- The low limit is a real number that specifies the lower boundary of the scaling range.
- The high limit is a real number that specifies the upper boundary of the scaling range.

Using UNSCALE for a Series 505 Controller

The UNSCALE procedure generates an output that has the same format as an analog output. The operation of the UNSCALE procedure depends on the type of controller that you have. For a Series 505 controller, UNSCALE simply takes the input value and converts it from a real value to a scaled integer. See Figure 11-54.

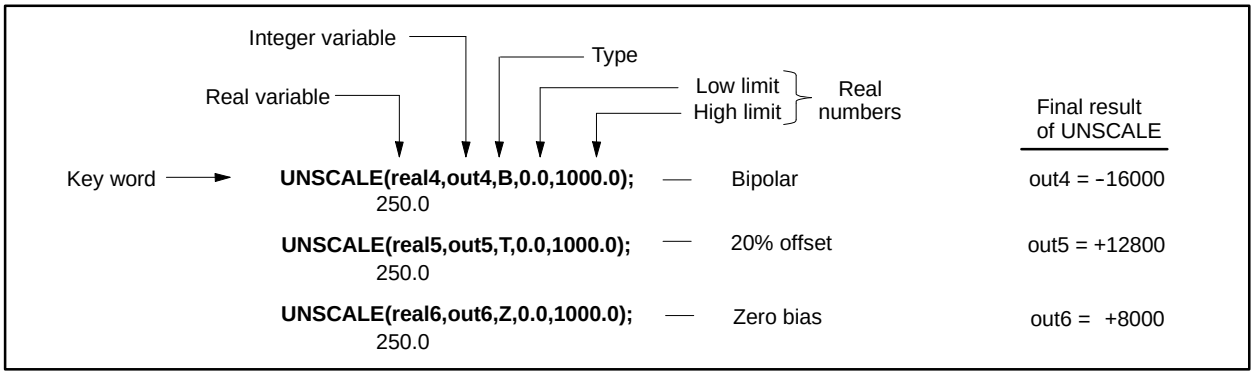


Figure 11-54 UNSCALE Procedure for Series 505

Using UNSCALE for an S5 Controller

If you have an S5 controller, the UNSCALE procedure converts a real value to a scaled integer and then shifts the integer by 4 bits to the left. This shifting occurs because S5 analog outputs are 16-bit numbers, with 11 bits of resolution, and the 4 least significant bits are irrelevant. Figure 11-55 shows the bit shift that UNSCALE performs after converting your real value to a scaled integer, in order to put the output variable in a format that can be properly interpreted by an S5 analog output module.

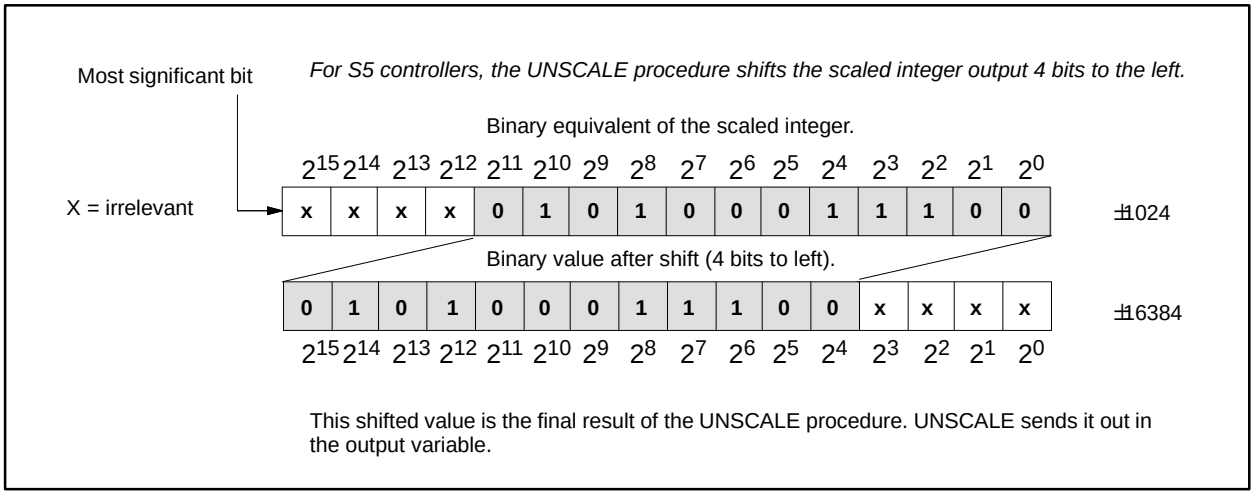


Figure 11-55 S5 Analog I/O Bit Shift

Function and Procedure Definitions (continued)

Viewing UNSCALE Results in Debug (S5 only)

If you have an S5 controller and use Debug to view the result of UNSCALE, the output variable falls in the range of 0–16384 (zero bias) or ±16384 (bipolar). This is because the value has been shifted by four bits, which is equivalent to multiplying it by a factor of 16. An S5 analog module reads these values as falling within the range 0–1024 (zero bias) or ±1024 (bipolar). Take this into account when you analyze what you are shown in Debug.

Figure 11-56 shows the result of using UNSCALE for an S5 controller. Notice that the final result of the procedure has been bit-shifted, according to the process shown in Figure 11-55.

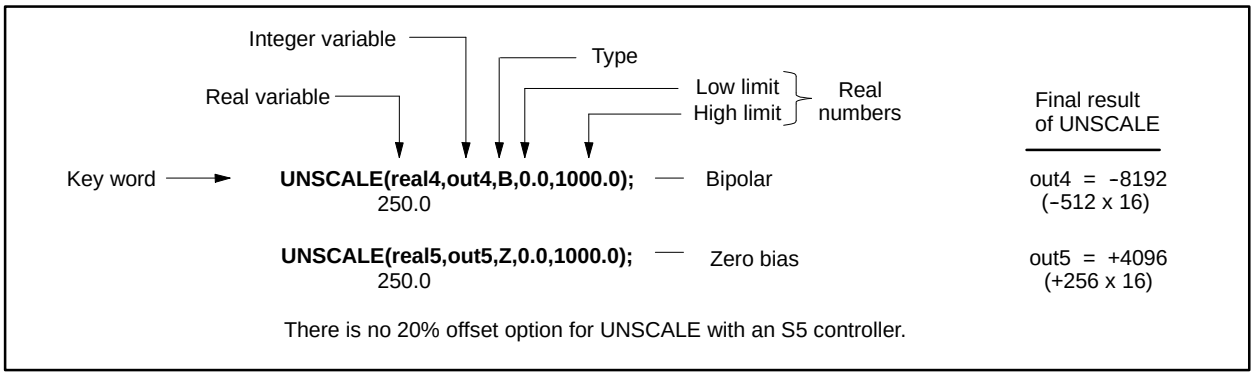


Figure 11-56 UNSCALE Procedure for S5

Compensating for the Bit Shift (S5 only)

If you want to use the output from the UNSCALE procedure for something other than an S5 analog output module, you must compensate for the bit shift that is automatically performed when you use the UNSCALE procedure. Either divide the value in the output variable by 16 after you perform UNSCALE, or else, if the value is positive, send the output variable to the RIGHTSHIFT procedure to shift it back (4 bits to the right). If your output variable value is negative, the RIGHTSHIFT procedure does not give you the correct result; RIGHTSHIFT takes the sign bit and assigns that value to the vacated four bits.

Chapter 12

Sequential Function Charts

12.1 Understanding Sequential Function Charts

Overview

As part of the process of developing an APT program, you create a structure that specifies the sequence of events in your control process. This structure is called a Sequential Function Chart (SFC).

An SFC is a graphical representation of a state-oriented (or step-oriented) control process. The SFC Editor provides a collection of graphics tools, text editors, and programming languages that you use to represent a control process as an SFC.

APT recognizes three types of SFCs.

- The main SFC starts the execution of program logic for a unit. Each unit has only one main SFC. You can designate any SFC as the main SFC by using the Promote option.
- A subordinate SFC executes its processing only when "called" by another SFC. You can nest any number of subordinate SFCs in a unit.
- A safe-state SFC executes when triggered by a specified event, such as an equipment failure. The safe-state SFC stops the processing of either all other SFCs in the currently active unit, or just a single SFC. When the safe-state SFC completes execution, control is transferred to another SFC, returning the program to some point in the normal process.

In general terms, an SFC consists of steps and transitions that can be arranged in a simple sequence or in branches.

- Each step contains program statements that represent a stage in the solution of a problem. Each step must be separated from the next step by one and only one transition.
- Each transition contains one conditional expression that determines when program control passes from one step to the next.
- Parallel branches allow you to follow a single transition with more than one step.
- Selection branches allow you to follow a single step with more than one transition condition.

Steps and transitions can contain comments to document your sequence of control. See Section 10.6 in Chapter 10 for information about using comments in your program.

Program Flow

Program control within an SFC flows from the top of the chart toward the bottom. Figure 12-1 illustrates how the step/transition structure controls program flow. When program control enters Step 1, that step is considered active. The statements in Step 1 execute at least once and continue executing until the expression in Transition 1 becomes true. At that time, Step 1 becomes inactive; and the statements in that step stop executing. Program control then passes through the transition to activate Step 2.

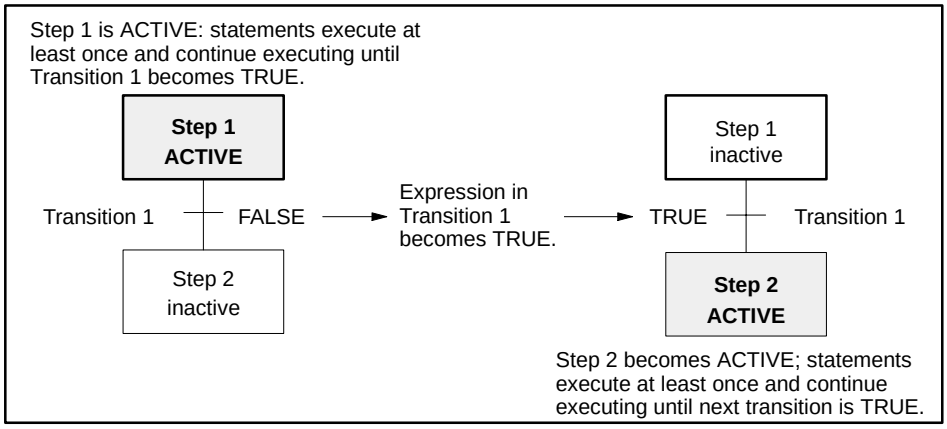


Figure 12-1 Program Flow through Steps and Transitions

When a program is first downloaded to the controller, all **.ENABL** extension variables are set to true. The first step in each main SFC is automatically activated when the controller goes into run mode.

You can control the operation of any SFC by toggling the **.ENABL** extension of the unit or program. If the *prog_name***.ENABL** bit is true and you set it to false, all SFCs are turned off and all devices are set to an unlocked state (manual mode). When you set the *prog_name***.ENABL** bit back to true, the first step of the main SFC is activated. All devices can then be placed in the locked state (automatic mode) with the appropriate command or assignment statement. The *unit_name***.ENABL** extension works similarly for unit SFCs and devices.

The *prog_name***.ABORT** extension variable allows you to stop program SFC execution by setting it to true. To restart the program, set the **.ABORT** extension back to false and then toggle the **.ENABL** extension as explained above. The *unit_name***.ABORT** extension works similarly for unit SFCs.

If the controller loses power, all units become inactive and remain inactive until the power returns. When power returns, each unit and/or program starts up in the initial step of the main SFC. One exception to this case is safe-state SFCs. If a retentive safe-state SFC was armed before the power failure and its trigger becomes true after the power returns, the safe-state activates and re-enables the unit. Chapter 13 of this manual contains the details about safe-state SFCs.

⚠ WARNING

If you have an S5 controller, some or all of your program variables, step transition expressions, and safe-state triggers can retain their value upon a return from power failure. This could cause unexpected program behavior, because SFC steps or safe-states could become active.

Unexpected program behavior can cause death or injury to personnel, and/or damage to equipment.

Carefully consider what happens to your program variables, step transition expressions, and safe-state triggers during power failure. The way you handle your operating system defaults determines whether some memory is retentive or not. Do not leave a variable set to true if that value could jeopardize program operation upon return from a power failure. See the chapter called "Compiling an APT Program" in the SIMATIC APT User Manual more information about setting operating system defaults.

SFC Scope

Like CFCs, SFCs are available at both the Unit Content Level and at the Program Content Level. The availability of SFCs at the Unit Content Level allows you to establish and maintain independent control of each unit. SFCs at the Program Content Level allow high-level sequence control across unit boundaries.

You can create any number of SFCs at both the Program Content and Unit Content levels. However, if you have a Series 505 controller and want to use the Step Logger option in the Debug Utility, you cannot have more than 64 SFCs in a unit. (Step Logger is not currently supported for S5.) One SFC can call a subordinate SFC only if both SFCs are in the same unit.

SFCs in the same unit can also communicate with each other through variables and objects residing at the Program Content Level or at the Unit Content Level for that unit. SFCs residing in different units can communicate with each other only through variables and objects residing at the Program Content Level. An SFC in a unit cannot access a subordinate SFC or safe-state SFC in another unit or at the global level. Global SFCs cannot access subordinate or safe-state SFCs that are defined in units.

12.2 Steps

Overview

Each step in an SFC can contain one or more commands that represent a stage in the solution of a problem. A step is entered from the top and exited from the bottom. An SFC can contain from 2 to 500 steps. APT automatically numbers each step with a unique label from S1 to S500.

Types

Figure 12-2 shows the three types of steps: initial, end, and regular.

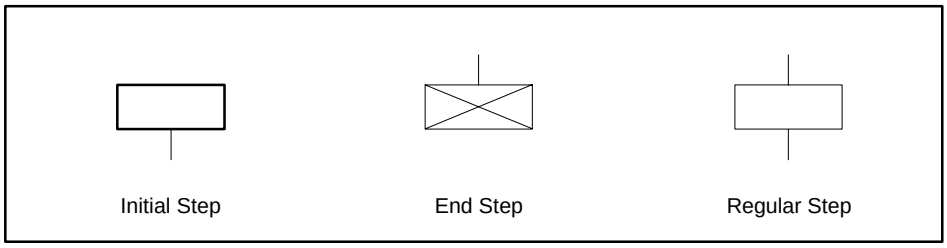


Figure 12-2 Types of Steps

The **initial step** is the step that is activated when an SFC is enabled. Each SFC must contain one and only one initial step.

The **end step** specifies the actions that occur when the execution of an SFC is complete. There can be no steps or transitions following an end step. An active end step remains active until the SFC is disabled or aborted. An SFC can have more than one end step.

A **regular step** is any step that is neither an initial step nor an end step.

Sections

The statements within a step can be divided into two sections: the parallel section and the math section. A step can include a parallel section, a math section, both, or neither. (You can create a step that does not include any statements). If a step includes both a parallel section and a math section, the parallel section must precede the math section.

The parallel section of a step contains statements that execute simultaneously ("in parallel"), regardless of the order in which they appear. The continuous execution of these statements continues as long as the step is active.

- Statements in the parallel section can be commands with the format:

  command object;

  The commands used with individual APT objects are listed with the explanation of each object. For example, see the chapter on devices for an explanation of device commands.

- Statements in the parallel section can be simple assignment statements, or boolean expressions that do not contain arithmetic operations or functions. The assignment statement has the format:

  identifier := identifier;

  Both identifiers are the same type of value (real, integer, boolean, array, etc.). These identifiers can be extension variables that are created by the system, names that were defined in the "APT Declarations" chapter in the *SIMATIC APT Programming Reference (Tables) Manual*, or numerical values.

  The boolean expression has the following format:

  boolean identifier: = boolean expression;

  The following is an example of a boolean expression:

  Bool: = Bool1 or Bool2 and Real1≥Real2;

NOTE: The objects referenced by the commands are located elsewhere in the STL or RLL code. Objects do not process the command until the object code executes.

The math section of a step consists of Math Language statements that execute sequentially, in the order in which they appear. The sequential execution of these statements continues as long as the step is active.

- The word MATH marks the end of the parallel section and indicates that the remainder of the step is a math section.
- The statements that follow a MATH statement constitute the math section of a step. The Math Language statements are described in Chapter 10, "Math Language Overview," and Chapter 11, "Math Language Functions and Procedures."
- Math statements can be commands that have the format:

  command object;

  The commands that can be used with individual APT objects are listed with the explanation of each object. For example, see the chapter on devices for an explanation of device commands.

Figure 12-3 illustrates a step that includes both a parallel section and a math section.

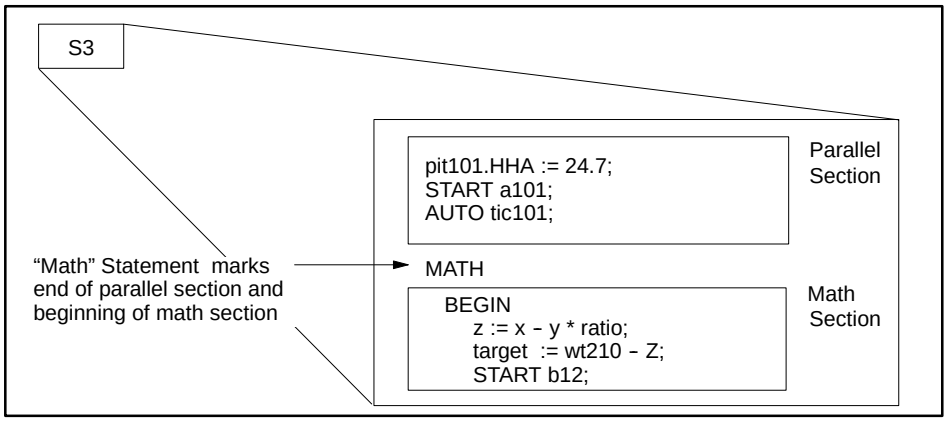


Figure 12-3 Sections of a Step

12.3 Transitions

Overview

An SFC can contain from 1 to 500 transitions. APT automatically numbers each transition in an SFC with a unique label from T1 to T500.

A transition statement contains one and only one boolean expression that determines when program control within an SFC passes from one step to another.

- A transition can be the boolean value, true; or it can be an expression that includes the relational and/or logical operations that are listed in Table 12-1.
- A transition expression cannot contain functions, procedures, or arithmetic operations (except **EDGE** and **BITTEST** functions).
- For Series 505 controllers, a transition expression cannot contain any S-memory variables, e.g., LOOP_NAME.BIAS, LOOP_NAME.LPV, %LPV45, %AODA60, etc.
- For S5 controllers, a transition expression cannot contain references to any direct address, e.g., %F12.2, %DB5:D12.2, etc.

Table 12-1 SFC Relational and Logical Operations

	Operator	Example
Relational	=	pt201 = 14
	< >	pt101 < > 14.0
	<	pt201 < 13
	< =	pt101 < = 14.5
	>	pt101 > 14.9
	> =	pt201 > = 15
Logical	NOT	NOT mcv103.OPND
	AND	mcv201.CLSD AND mcv202.CLSD
	OR	mcv103.OPND OR (pt101 < 14.9)

The boolean expression in a transition produces a result that is either true or false and must follow these rules.

- The values on both sides of each operator in an expression must be the same type. See Section 10.4 of the "Math Language Overview" chapter for more information about types of values and boolean expressions.
- Real values in a transition slow down program execution; therefore, if you must use relationships with real values, place these in a math section of the preceding SFC step as shown in Figure 12-4.

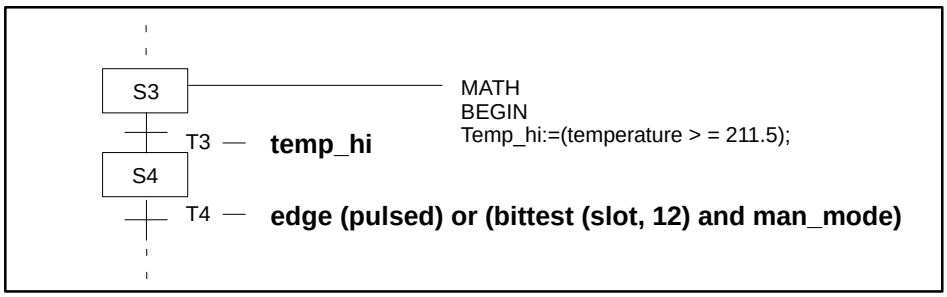


Figure 12-4 Using Transitions

Evaluation

A transition is entered from the top and exited from the bottom. Transitions are evaluated according to the following rules:

- A transition following an active step is evaluated after the statements in the parallel section of that step are executed.

  Because a transition is evaluated after the statements have been executed, all statements (command, assignment, and math) in an active step are always executed at least once.

- If the transition condition is false, the step remains active and the statements are executed again before the transition is re-evaluated.

- If the transition is true, the step becomes inactive; execution passes through the transition to the next step, which becomes active.

NOTE: If the step is an exit step in a subordinate SFC, it executes at least twice before returning to the calling SFC.

The control flow in Figure 12-5 follows:

- The statement in Step 3 is a command that instructs the controller to open valve *mcv201*.

- After the controller transmits the request to open valve *mcv201*, Transition T3 checks to determine if the valve is open. (If the valve is already open when Step S3 becomes active, the statement in Step S3 is executed once before the Transition T3 is evaluated.)

- If T3 is true, Step S3 becomes inactive and Step S4 becomes active. If T3 is false, Step S3 continues to instruct the controller to open the specified valve, and T3 checks the condition after each execution of the command.

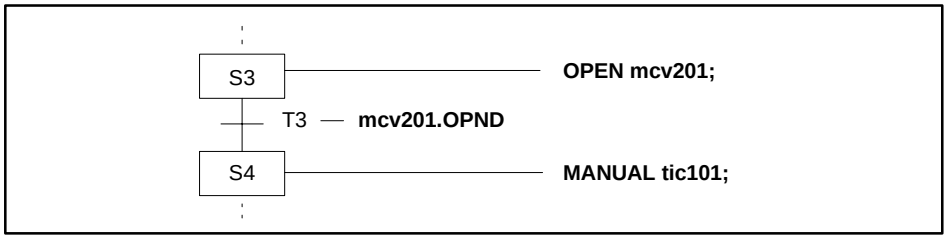


Figure 12-5 Flow Control in Transitions

⚠ CAUTION

Because transitions are evaluated after execution of the parallel section of a step, a transition to the following step can become true before the math section is complete.

If a step performs operations on the data created by a previous step before calculations for the previous step have been completed, erroneous results can occur. Erroneous results can cause loss of data and/or damage to equipment.

To prevent this potential error, create an APT flag to indicate when the math section is complete. Set the flag in the last line of the math section and use it in the transitions that follow.

As illustrated in Figure 12-6, one method of avoiding these difficulties is to include an integer variable in each math section to "signal" when execution of the math block is completed. If you test that variable as part of the condition of a transition, you can ensure that the math section is completed at least once.

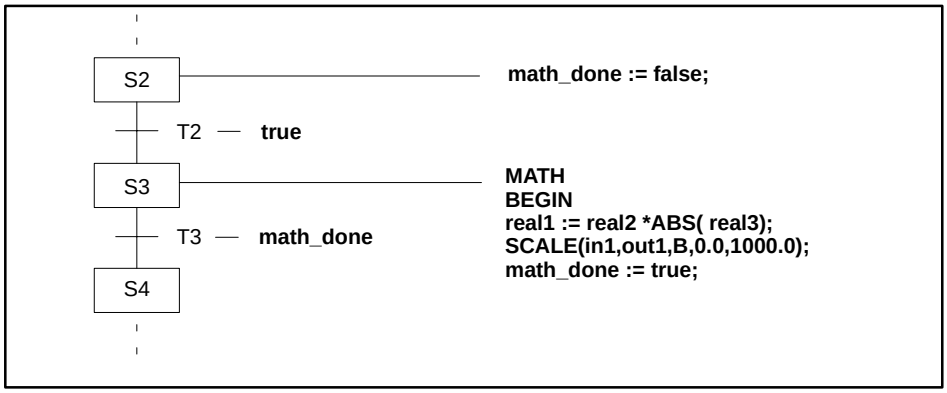


Figure 12-6 Testing for Completed Execution of a Step

12.4 Step/Transition Rules

Initial and End Steps

The following rules for steps and transitions are illustrated in Figure 12-7.

- Each SFC must contain one and only one initial step.
- The initial step does not necessarily have to be placed at the top of the chart. Regardless of its physical location, the execution of an SFC always begins with its initial step.
- No steps or transitions can follow an end step.
- Because an SFC can contain loops, it is possible to construct an SFC that does not include an end step.
- Because an SFC can include branches, an SFC can contain more than one end step.

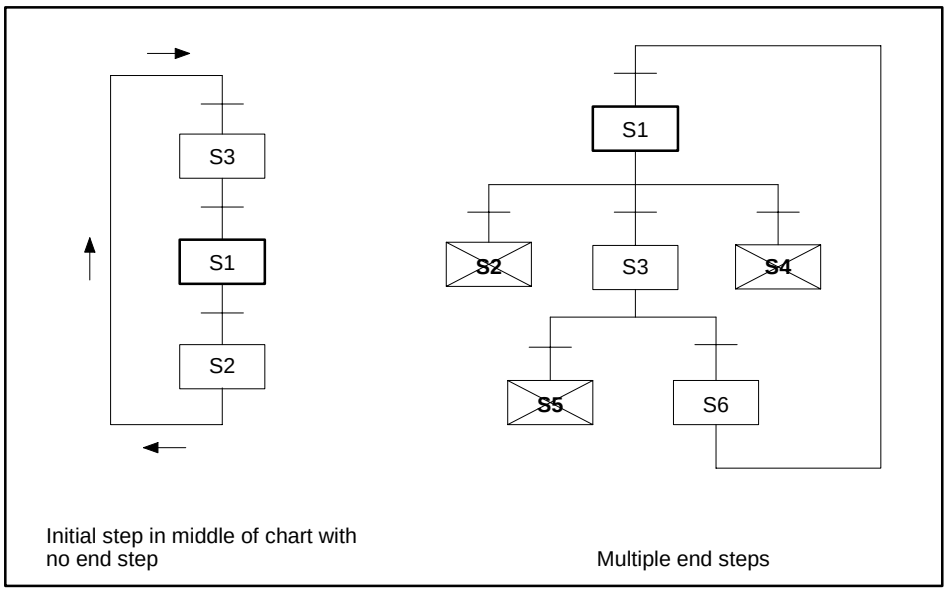


Figure 12-7 Initial and End Steps

Steps and Transitions

The following rules for steps and transitions are illustrated in Figure 12-8.

- Each initial and regular step must be followed by a transition.
- Each regular and end step must be preceded by a transition.
- The same step cannot both follow and precede the same transition; that is, the path leading from the step cannot loop directly back to the transition that preceded that step.
- The same transition cannot both follow and precede the same step; that is, the path leading from a transition cannot loop directly back to the step that preceded the transition.

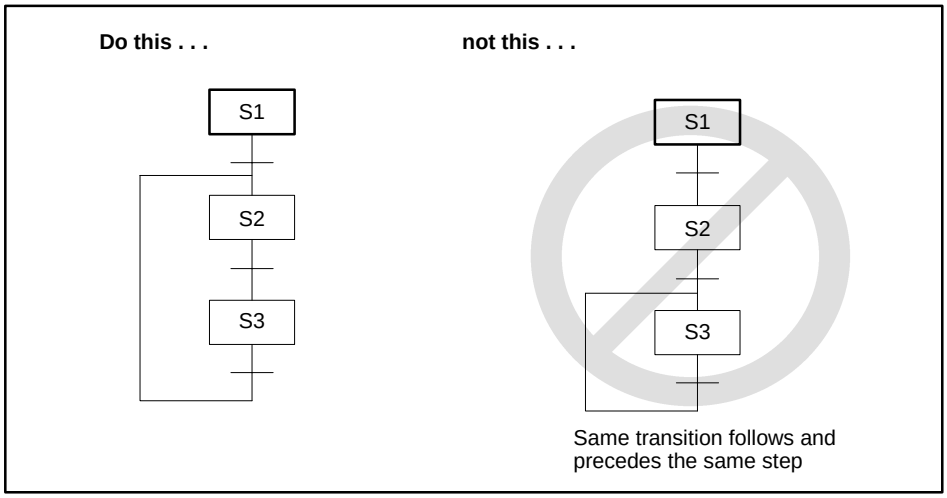


Figure 12-8 Step/Transition Rules

12.5 Lines, Arrows, and Graphic Connections

Lines

All symbols must be connected properly to ensure that the flow of control is continuous. When you use the drawing icons in the SFC editor (especially the macros), you can accidentally leave unconnected areas between the various structures that you create. Use the line-draw icon to connect the unconnected symbols.

The illustration on the left side of Figure 12-9 depicts an unconnected structure that might result if you use the parallel macro after you use the step-transition macro. The SFC on the right side of Figure 12-9 shows how the structure appears after you correct it with the line-draw icon.

Extraneous lines are normally ignored if they terminate on cell boundaries; however, an extraneous line that terminates in the interior of a cell causes an error when you compile the program.

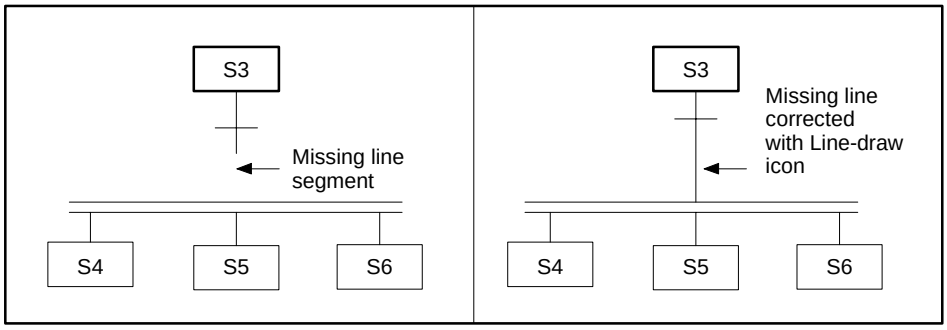


Figure 12-9 Using Line-draw Icon

NOTE: Lines in an SFC cannot cross each other. See page 12-16 for a method to avoid this problem.

Arrows

In some SFCs, determining the direction of control flow just by looking at the chart may be difficult. As illustrated in Figure 12-10, you can use the arrow icon to draw arrows beside lines in an SFC to clarify the direction of control flow.

You cannot use arrows to change the direction of flow, only to document the direction of flow in the program. Arrows are checked for accuracy when you compile the program, and all arrows must point in the correct direction. For example, all of the arrows in the SFC on the right side of Figure 12-10 point in incorrect directions and are errors that prevent the program from compiling.

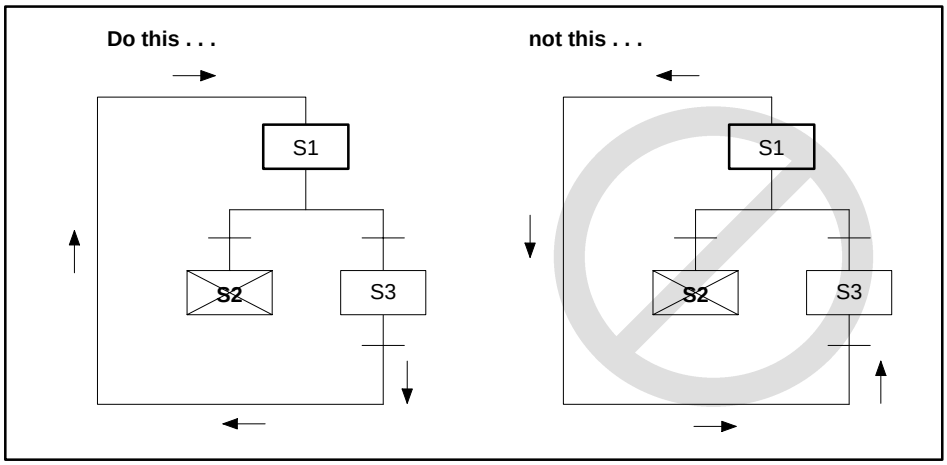


Figure 12-10 Using Arrows to Document Program Flow

Graphic Connections

SFCs that include many lines sometimes become cluttered and difficult to interpret, especially when they involve looping structures.

The graphic connection icon allows you to transfer control to a specified step without drawing a connecting line. This makes it possible to avoid intersecting lines in an SFC. For example, the graphic connection in Figure 12-11 transfers control to Step S7 without crossing any other lines.

Functionally, a graphic connection takes the place of a line that connects a transition to a step. It must follow a transition, and it must contain the label of the step to which control is to be transferred. A graphic connection can transfer control only within an SFC, not between SFCs.

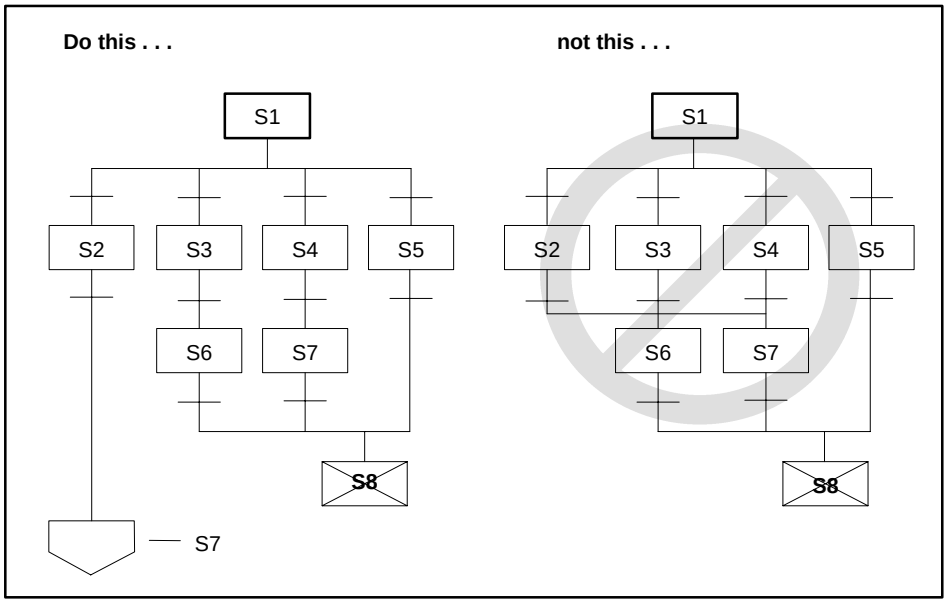


Figure 12-11 Graphic Connections

12.6 Parallel Branches

Parallel Branching

Parallel branching allows you to designate steps that execute at the same time as other steps. A parallel branch begins with a single transition, always contains a top crossbar, and can end with a bottom crossbar.

- The **top crossbar** follows a single transition and leads to two or more steps.
- A **bottom crossbar** follows two or more steps and leads to a single transition.

The left side of Figure 12-12 shows the top crossbar leading to two parallel steps. When Transition T2 becomes true, Steps S3 and S4 both become active; each step remains active until its corresponding transition becomes true.

The right side of Figure 12-12 shows two parallel steps leading to a bottom crossbar that returns to a single transition. Both Steps S3 and S4 (after they become active) remain active until Transition T3 becomes true, at which time End Step S5 becomes active.

If a step that immediately precedes the bottom crossbar of a parallel branch contains a call to a subordinate SFC, the transition that follows the crossbar is not evaluated until the end step of the subordinate SFC is active. See page 12-28 for information about subordinate SFCs.

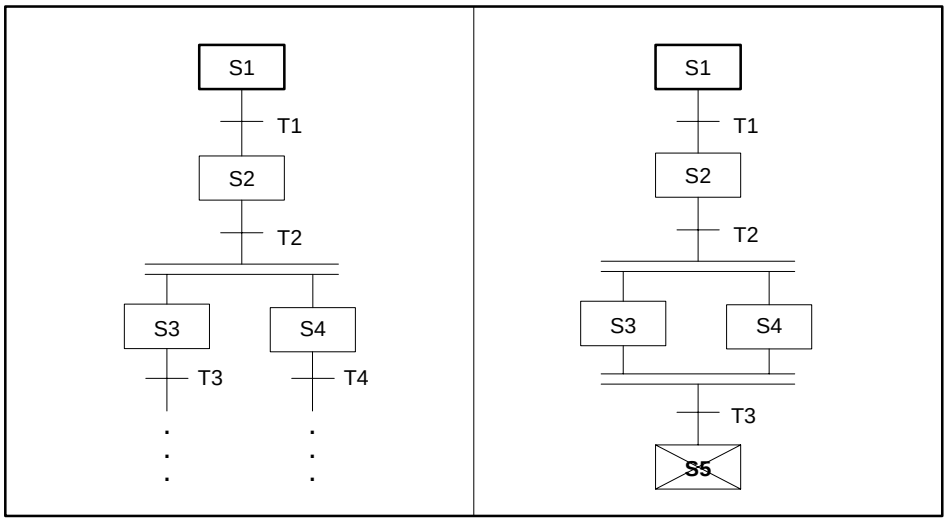


Figure 12-12 Simple Parallel Structures

The control flow in Figure 12-13 follows:

- Step S1 is active.
- When Transition T1 becomes true, Step S1 becomes inactive; and steps S2, S3, S4, and S5 become active.
- Steps S2 and S3 remain active until Transition T3 becomes true. Steps S2 and S3 then become inactive, and Step S7 becomes active.
- Step S4 remains active until Transition T2 becomes true.
  Step 4 then becomes inactive and End Step S6 becomes active.
- Steps S5 and S7 remain active until Transition T4 becomes true. Steps S5 and S7 then become inactive and End Step S8 becomes active.
- Transition T4 is not evaluated until both Steps S5 and S7 become active. The transition that follows the bottom crossbar of a parallel branch is not evaluated until each step that immediately precedes the crossbar is active.

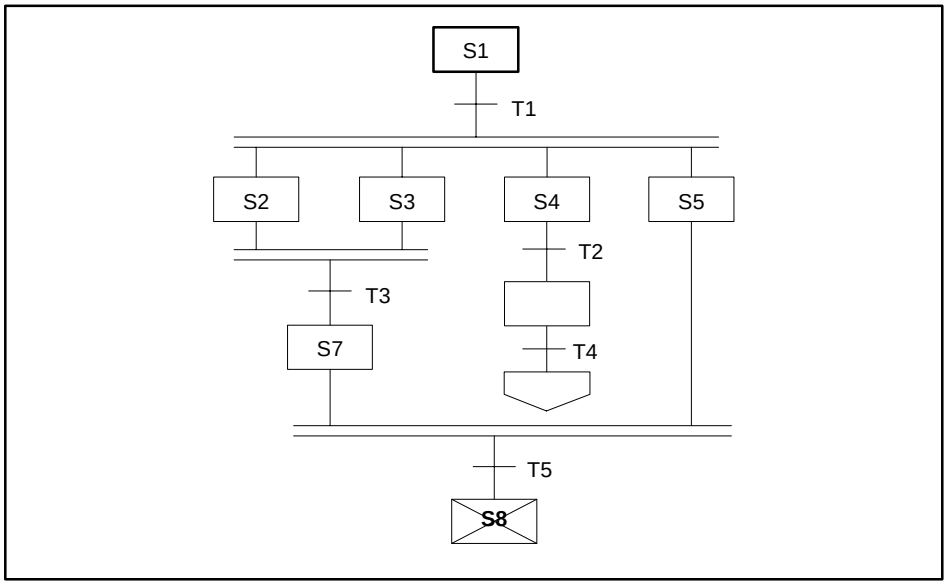


Figure 12-13 A More Complex Parallel Structure

The more complex structure of the irregular parallel branches in Figure 12-14 illustrates the following:

- A parallel structure does not have to be symmetrical.
- Parallel branches do not have to converge.

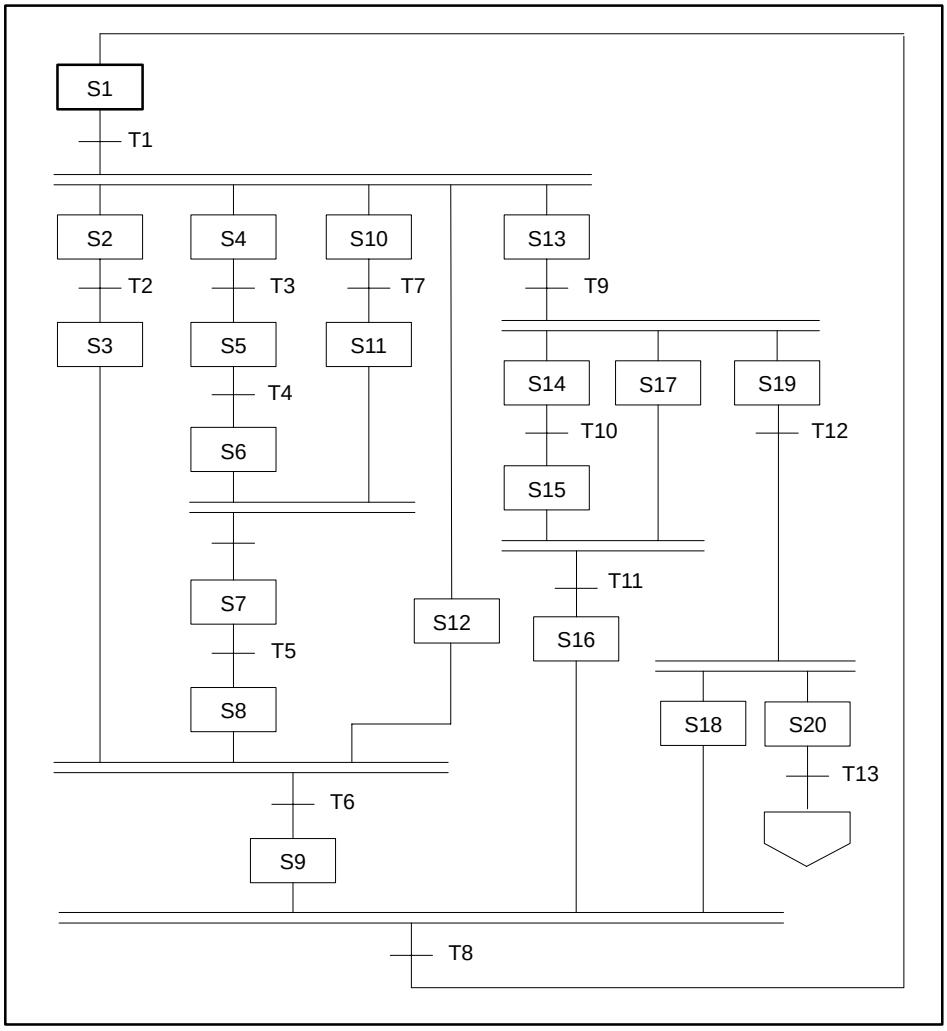


Figure 12-14 Irregular Parallel Branches

Parallel Branching Rules

Although the SFC Language allows you a great deal of flexibility, there are a few rules that you must observe when you implement a parallel branch:

- The top crossbar of a parallel branch must immediately follow a single transition.
- The top crossbar of a parallel branch can have only one entrance path, which must enter from above the crossbar.
- The top crossbar of a parallel branch must have at least two exit paths, which must exit downward from the crossbar.
- Each path exiting the top crossbar of a parallel branch must begin with a single step or a graphic connection.
- The bottom crossbar of a parallel branch must have at least two entrance paths, which must enter from above the crossbar.
- Each path entering the bottom crossbar of a parallel branch must end with a single step.
- The bottom crossbar of a parallel branch can have only one exit path, which must exit downward from the crossbar.
- The bottom crossbar of a parallel branch must be followed immediately by a single transition.

These rules are illustrated in Figure 12-15.

⚠ WARNING

APT allows the use of parallel end steps (that is, end steps in a parallel branch structure). When any end step is encountered in a sub-SFC, control returns to the parent SFC. If any other steps in the parallel branch are currently active when an end step is reached, they remain active.

Having multiple active SFC steps could cause unpredictable operation of the controller, which can result in death or serious injury to personnel, and/or damage to equipment.

Avoid the use of parallel end steps in any but the Main SFC.

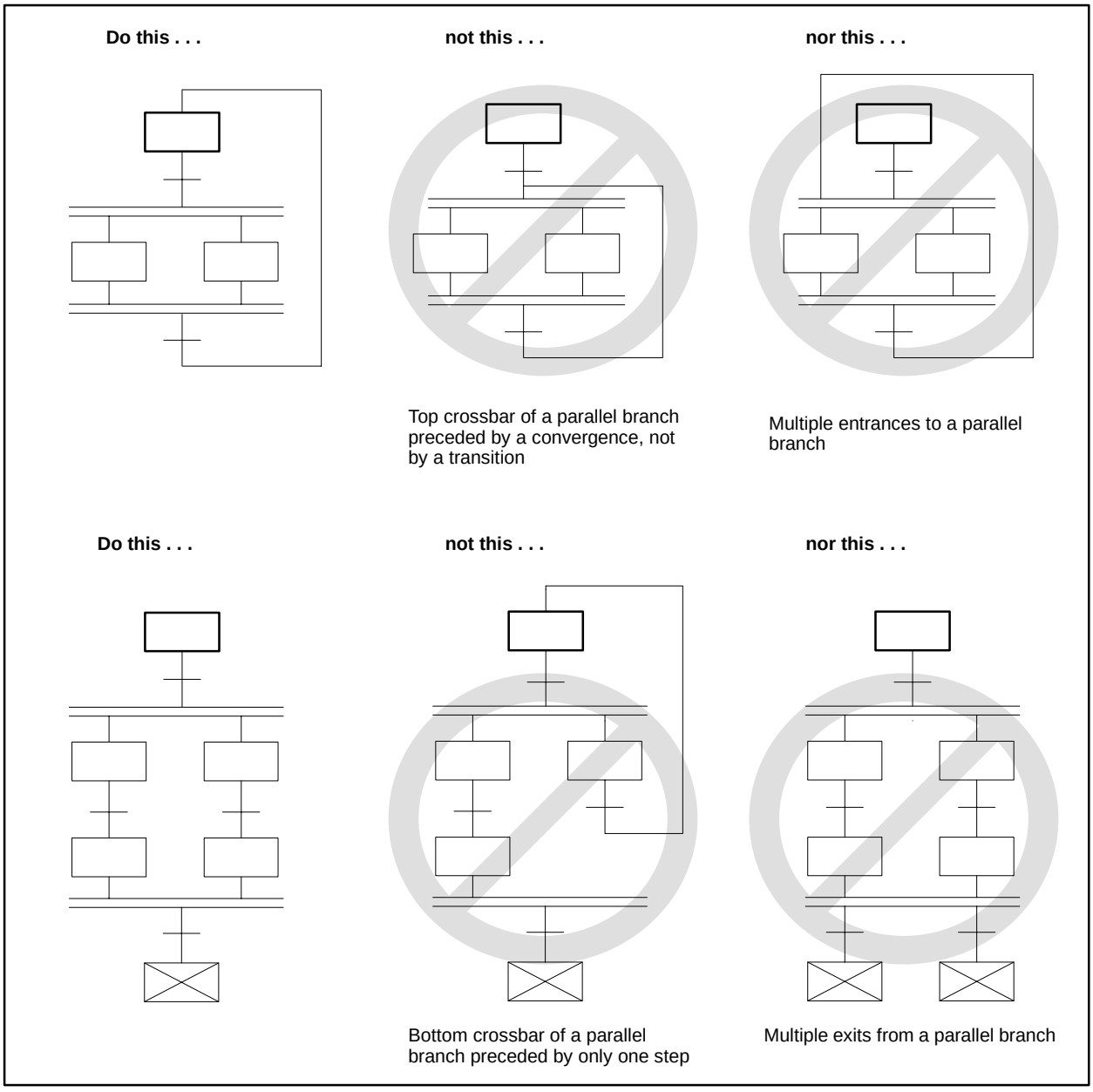


Figure 12-15 Parallel Branching Rules

12.7 Selection Branches

Selection Branching

Selection branching allows you to "divert" SFC execution to one of several paths depending on the value of transition conditions. The paths may or may not subsequently converge.

A selection branch begins with a single step, contains a divergence line, and can end with a convergence line.

The **divergence line** follows a single step and leads to two or more transitions.

The **convergence line** follows two or more transitions and leads to a single step.

The left side of Figure 12-16 shows a divergence line leading to two transitions. Step S2 (after it becomes active) remains active until either T2 or T3 becomes true. SFC execution then proceeds to the step following that transition. If more than one transition from a divergence line becomes true simultaneously, SFC execution continues through the path of the leftmost true transition.

The right side of Figure 12-16 shows a convergence line that leads to End Step S5 when either Transition T4 or T5 becomes true.

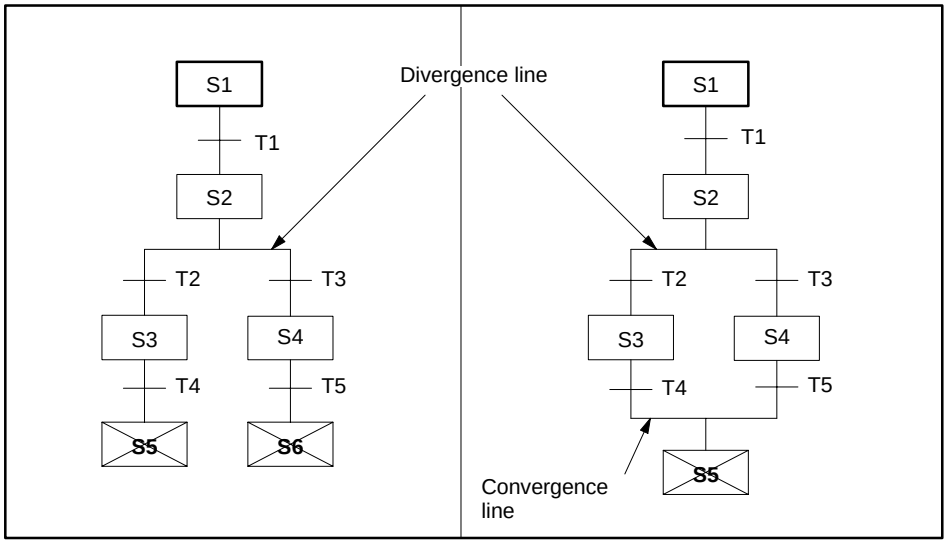


Figure 12-16 A Simple Selection Branch

The control flow in Figure 12-17 follows:

- After Step S2 becomes active, it remains active until one of the following occurs.

  Step S3 becomes active if Transition T2 becomes true.

  Step S4 becomes active if Transition T3 becomes true before T2.

  End Step S5 becomes active if Transition T4 becomes true before T2 or T3 becomes true.

- When Transition T2, T3, or T4 becomes true, Step S2 becomes inactive.

- Step S6 becomes active when one of the following occurs.

  Transition T5 becomes true while Step S3 is active (Step S3 becomes inactive), or

  Transition T6 becomes true while Step S4 is active (Step S4 becomes inactive).

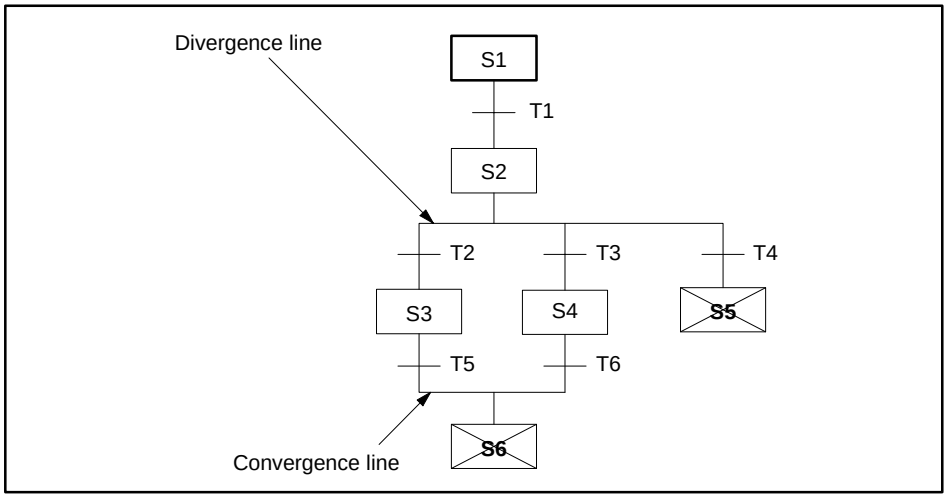


Figure 12-17 A More Complex Selection Branch

Figure 12-18 shows an SFC with two convergences and loops. Figure 12-19 shows an SFC with both selection branches and parallel branches. Try to avoid drawing very complex or irregular SFCs connections.

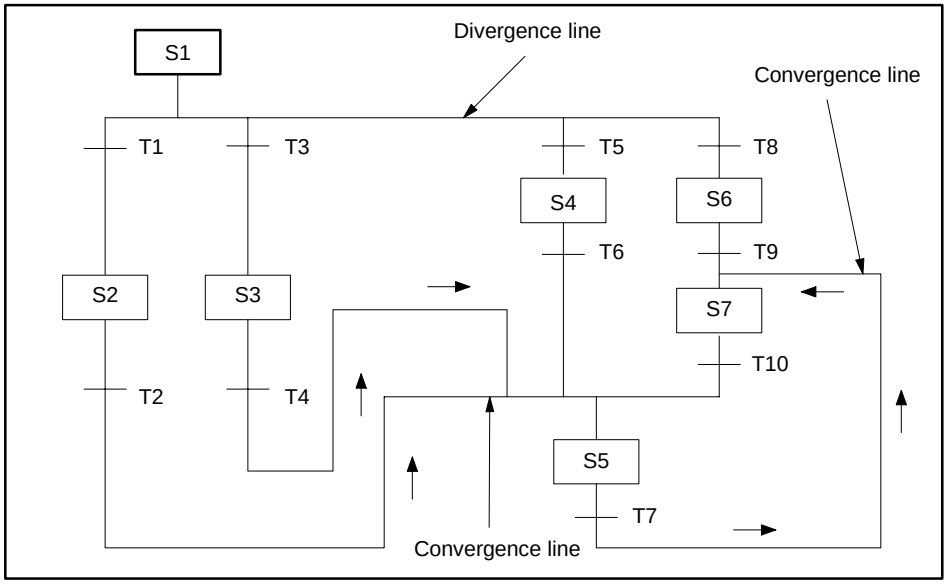


Figure 12-18 Two Convergences

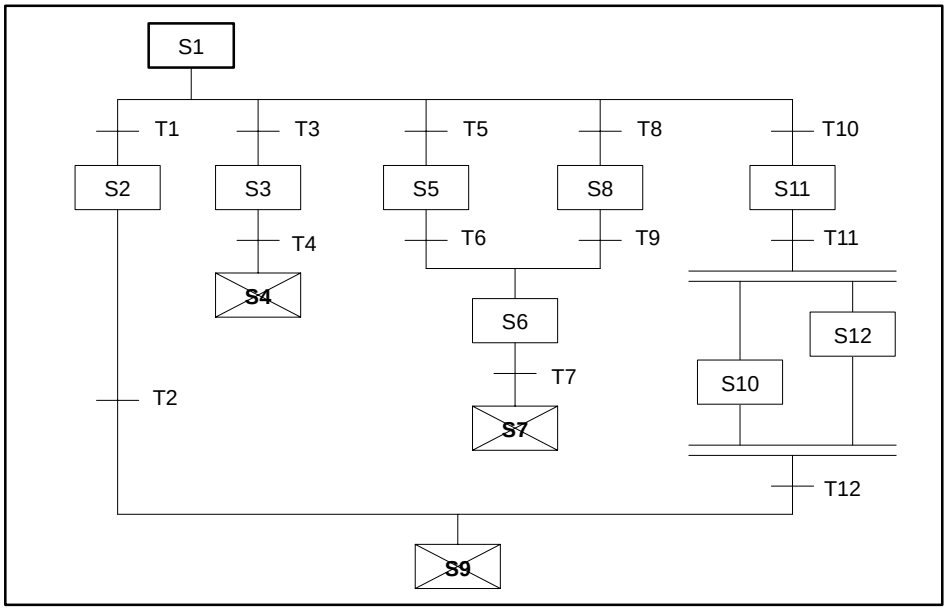


Figure 12-19 Selection and Parallel Branches

Selection Branching Rules

The rules that govern the implementation of a selection branch can be divided into rules of divergence and rules of convergence.

The rules that you must observe when you implement the divergence of a selection branch follow:

- A divergence is represented by a horizontal line.
- A divergence must immediately follow a single step.
- A divergence can have only one entrance path, which must enter from above the divergence line.
- A divergence must have at least two exit paths, each of which must begin at a different point on the divergence line.
- Each path exiting a divergence must begin with a single transition.
- A path must exit from each endpoint of a divergence line.
- A path exiting from an endpoint of a divergence line may exit upward or downward; a path exiting between the endpoints of a divergence line must exit downward.
- A divergence does not require a subsequent convergence. (Selection branches do not have to converge.)

These rules are illustrated in Figure 12-20.

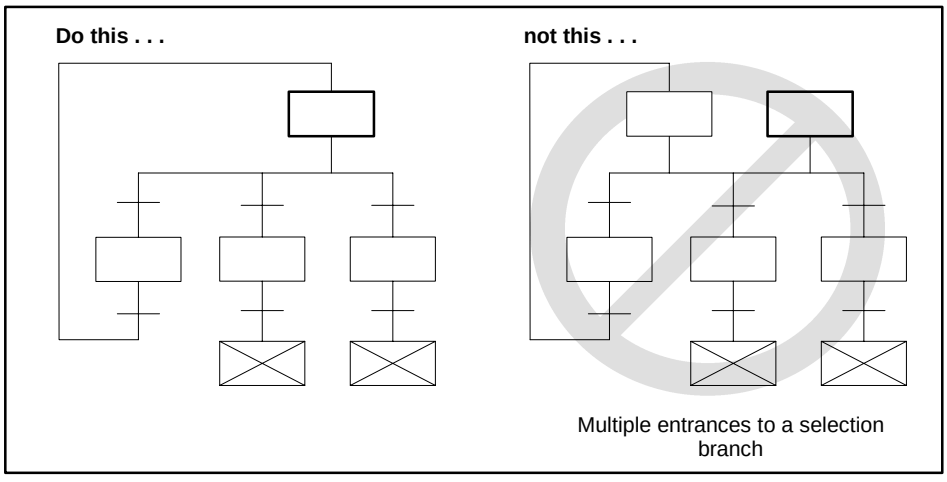


Figure 12-20 Divergence of a Selection Branch

The rules that you must observe when you implement the convergence of selection branches follow:

- A convergence is represented by a horizontal line.
- A convergence must have at least two entrance paths, each of which must end at a different point on the convergence line.
- Each path entering a convergence must end with a single transition.
- A path must enter each endpoint of a convergence line.
- A path entering an endpoint of a convergence line may enter from above or below; a path entering between the endpoints of a convergence line must enter from above.
- A convergence can have only one exit path, which must exit downward from the convergence line.
- A convergence must be followed immediately by a single step or graphic connection.
- A convergence does not require a prior divergence. (You can use a convergence to merge separate paths into one common path.)

These rules are illustrated in Figure 12-21.

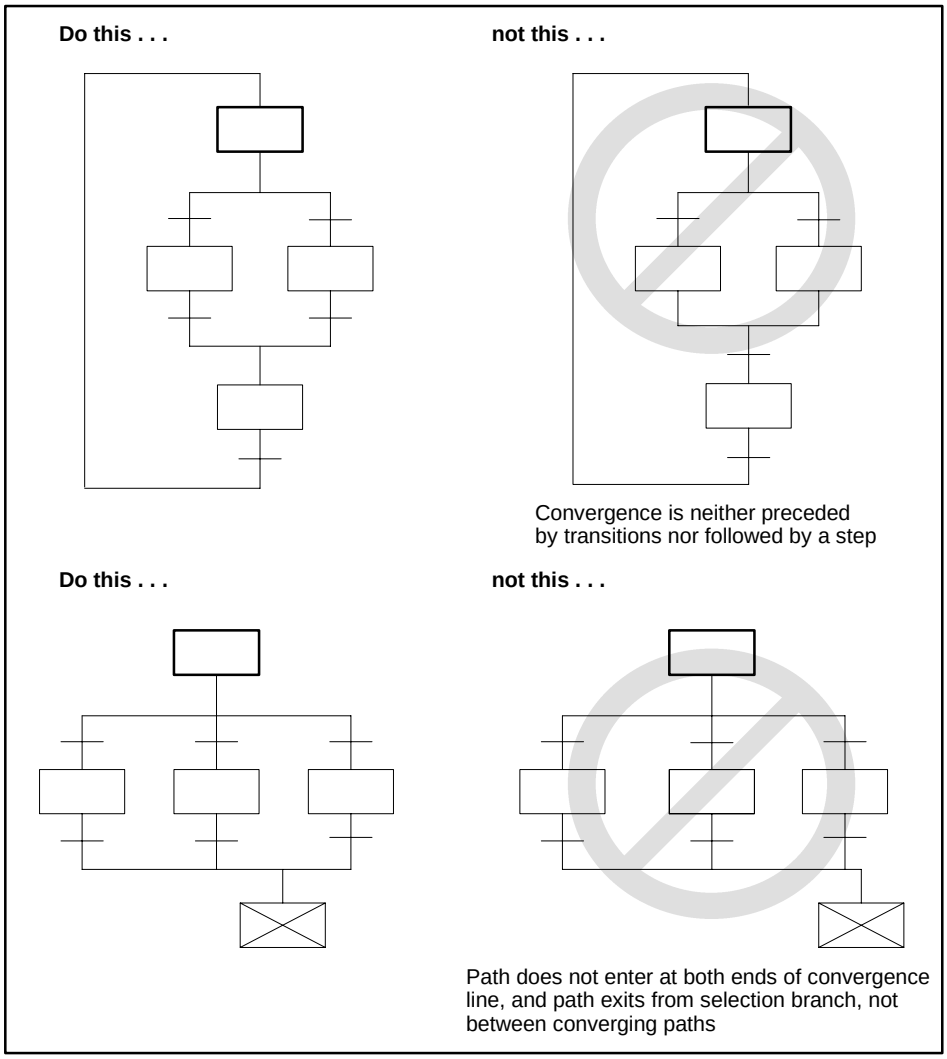


Figure 12-21 Selection Branches

12.8 Main and Subordinate SFCs

Main SFC

The main SFC starts the execution of program logic for a unit. Before you compile a unit, you must use the **Promote** option to designate one (and only one) SFC as the main SFC for that unit.

APT automatically promotes the first SFC that you enter. If you enter additional SFCs and determine that one of these should be the main SFC, promote that SFC using the **Promote** option.

When a unit is enabled (*unit_name*.**ENABL** is set equal to true), the main SFC for that unit begins to execute (its initial step becomes active). The main SFC is the only one in that unit that begins to execute automatically when the unit is enabled.

If the unit is disabled (*unit_name*.**ENABL** = false), all currently active steps in the SFC become inactive.

If the *unit_name*.**ABORT** variable is set to true, all SFC steps are forced inactive. SFC execution does not resume when *unit_name*.**ABORT** returns to false.

- To restart the SFC in the initial step of the main SFC, you must set the *unit_name*.**ENABL** bit to false and then to true.
- The one exception to this is the case of a safe-state SFC that was triggered before *unit_name*.**ABORT** was set to true. In this situation the safe-state SFC does start up when *unit_name*.**ABORT** goes to false. (See Chapter 13 for information on safe-state SFCs.)

Subordinate SFCs

When you design a process control system, you often begin by dividing the process into a series of broadly defined basic steps; then you develop the details of the control sequences that comprise each of these basic steps. In APT, you can implement this kind of control strategy through the use of subordinate SFCs.

A subordinate SFC is one that is "called" from another SFC. The calling SFC and the subordinate SFC must both reside in the same unit; that is, one SFC cannot call a subordinate SFC in a different unit.

You can "nest" subordinate SFCs; that is, one subordinate SFC can call another subordinate SFC. There is no limit to the number of levels to which you can nest subordinate SFCs.

A subordinate SFC can be called from only one step in only one SFC. In other words, two different SFCs cannot call the same subordinate SFC, nor can the same subordinate SFC be called from two different steps within an SFC.

To call a subordinate SFC, use:

SFC sfc_name;

- The *sfc_name* must be the name of a subordinate SFC residing in the same unit as the calling SFC.
- A subordinate "normal" (non-safe-state) SFC can be called only from a normal SFC; a subordinate safe-state SFC can be called only from a safe-state SFC.
- A subordinate SFC can be called from only one step in only one SFC.
- You can use the SFC command only in initial steps and regular steps, not in end steps.
- The SFC statement must be the only statement in the step. That is, a step that contains an SFC statement cannot contain any other statements except comments.

When you call an SFC, the following events occur:

- The initial step of the subordinate SFC becomes active, and the calling step in the calling SFC remains active.
- The subordinate SFC executes until it reaches an end step.
- The end step of the subordinate SFC and the calling step of the calling SFC remain active until the transition following the calling step becomes true.
- When that transition becomes true, the end step of the subordinate SFC and the calling step of the calling SFC simultaneously become inactive; the step following the transition becomes active.

You can think of a subordinate SFC as the “expansion” of a step in the calling SFC. For example, Figure 12-22 shows a step of an SFC that calls a three-step subordinate SFC.

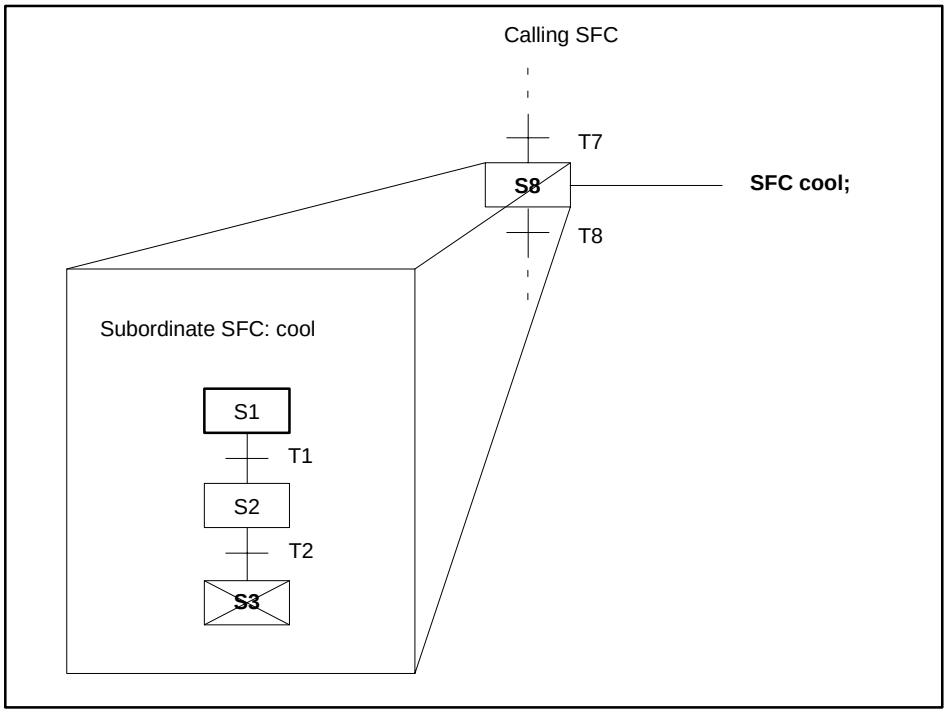


Figure 12-22 An “Expanded” Calling Step

Figure 12-23 illustrates the sequence of events that occurs when an SFC named *r01_cntl* calls a subordinate SFC named *cool*. The control flow follows:

- Step S8 of SFC, *r01_cntl*, calls the subordinate SFC *cool*. The execution of the calling SFC, *r01_cntl*, pauses at Step S8.
- The subordinate SFC *cool*, begins to execute from its initial step, S1, and continues to execute through its end step, S4.
- Step S4 remains active until Transition T8 in the calling SFC, *r01_cntl*, becomes true.
- The execution of the calling SFC, *r01_cntl*, resumes at Step S9.

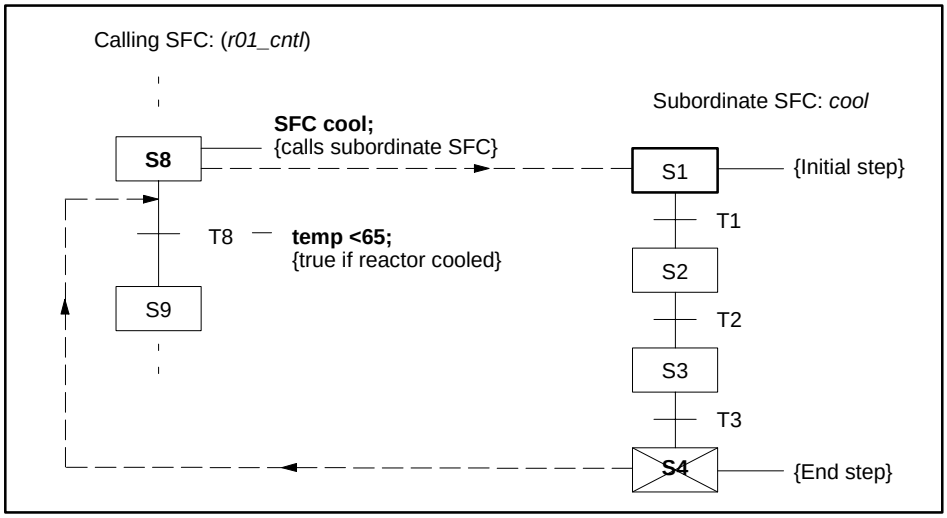


Figure 12-23 Subordinate SFC Execution

Chapter 13

Safe-State SFCs

13.1 Overview

Types

A safe-state SFC is a special SFC that is designed to interrupt the execution of other SFCs. A safe-state SFC can execute an emergency procedure, or it can perform special processing out of the normal flow of control.

There are three types of safe-state SFCs:

A **level safe-state SFC** is designed to interrupt the processing of a main or subordinate SFC. A level safe-state SFC can also interrupt another safe-state SFC with a lower priority.

A **local safe-state SFC** is designed to interrupt the processing of a single SFC and any of its subordinate SFCs.

A **subordinate safe-state SFC** is called by another safe-state SFC (main or subordinate) in the same manner that a "normal" (non-safe-state) subordinate SFC is called by a normal SFC. A subordinate safe-state SFC must be defined in the same manner as the calling safe-state SFC.

Safe-State Priority

Each main and subordinate level safe-state SFC must be assigned one of five priority levels, numbered 1 through 5, or a local priority. Level 1 safe-state SFCs have the highest priority and override all other safe-state SFCs. Local safe-state SFCs have the lowest priority.

- A safe-state SFC with any priority can interrupt the processing of normal SFCs. A safe-state SFC can also interrupt the processing of another safe-state SFC that has a lower priority.
- A safe-state SFC cannot interrupt the processing of another safe-state SFC that has a higher or equal priority. When a safe-state SFC interrupts the processing of another safe-state SFC, execution of the interrupted safe-state SFC is terminated.
- A safe-state SFC can call a subordinate safe-state SFC only if both SFCs have the same priority. A safe-state SFC cannot call a normal SFC, and a normal SFC cannot call a safe-state SFC.

Only one local safe-state of any given SFC can be active at one time. However, multiple local safe-state SFCs can be active if they belong to different SFCs. It is possible for two local SFCs in the same subordinate path to become active at the same time if the lowest subordinate SFC becomes active first.

Safe-State SFC Operation

When a level safe-state begins executing, all active steps in all other SFCs in the unit become inactive. When a local safe-state SFC begins executing, the associated normal SFC and its subordinate SFCs become inactive.

To set up a safe-state SFC, you need to identify the following:

- Digital outputs, boolean variables, or APT flags that identify, or trigger, the events that require an interruption.
- Sections of the program that can and cannot be interrupted. Sections that can be interrupted are “armed.” Sections that cannot be interrupted are “disarmed.”
- Options for execution at the end of the safe-state SFC determine if execution continues or if the program stops when the safe-state SFC is completed.

⚠ WARNING

Do not use the safe-state SFC as a replacement for a hardwired mechanical emergency stop function.

Doing so could cause unpredictable operations that could result in death or serious injury to personnel, and/or damage to equipment.

Any safety-critical operations must always be wired independently of any solid state computer-controlled operation. It is acceptable to use the safe-state SFC for supplemental protective purposes if, and only if, hard-wired backup is also used in the event of any computer malfunction.

13.2 Safe-State Commands

Overview

Seven APT commands are related to safe-state SFCs. Four of the commands can be used only in a safe-state SFC, as is explained below.

The initial step of a safe-state SFC must contain the following two commands:

- The SSDEFINE command sets the priority (Level 1–5 or local) of the safe-state SFC.
- The SSTRIGGER command specifies the condition that triggers the action of the safe-state SFC. The SSTRIGGER command is used only in a main safe-state SFC.

Each end step of the main SFC in level safe-state SFC network must contain one of the following commands:

- The SSABORT command stops all SFCs.
- The SSRETURN command specifies a step in a non-safe-state SFC to restart the normal processing when the safe-state SFC is completed.

Each end step of the main SFC in a local safe-state SFC network must contain only the SSRETURN command, and it must return to the SFC named in the SSDEFINE statement. The SSABORT command cannot be used in a local safe-state SFC.

The other three commands are generally used outside of the safe-state SFC:

- The SSARM command indicates that you want to begin monitoring the trigger condition so that you can interrupt the process if necessary.
- The SSDISARM command indicates that you want to stop monitoring the trigger condition so that the process will no longer be subject to an interruption.
- The SSENTRY command specifies a step where you want to restart the process when the safe-state SFC is completed. The SSENTRY command is used only in a non-safe-state SFC.

Table 13-1 summarizes the restrictions concerning the safe-state commands.

Table 13-1 Safe-State Command Restrictions

Command	Restrictions
SSABORT	Use only in the end step of a main-level safe-state SFC; must be the only statement in the step.
SSARM	(no restrictions)
SSDEFINE	Use only in the initial step of a safe-state SFC; must be the first statement in the step.
SSDISARM	(no restrictions)
SSENTRY	Use only in a "normal" (non-safe-state) SFC; must be the first statement in the step or immediately following another SSENTRY; maximum of 10 per step.
SSRETURN	Use only in the end step of a main safe-state SFC. A main safe-state SFC can have more than one SSRETURN but these must be the only statements in the step.
SSTRIGGER	Use only in the initial step of a main safe-state SFC; must immediately follow an SSDEFINE or another SSTRIGGER statement.

Table 13-2 summarizes the steps in safe-state SFCs that have requirements concerning the step commands that they must contain.

Table 13-2 Safe-State SFC Step Requirements

Safe-State SFC	Step	Requirements
Main/Subordinate	Initial	Must begin with an SSDEFINE statement.
Main	Initial	SSDEFINE statement must be followed by at least one SSTRIGGER statement.
Main	End	Must include an SSABORT or SSRETURN statement; cannot include any other type of statement.

SSENTRY Command

The SSENTRY command defines a label that you use to indicate where the program returns when a safe-state issues the appropriate SSRETURN command.

For the SSENTRY command, use the format:

SSENTRY label;

- Use the SSENTRY command only in a “normal” (non-safe-state) SFC.
- The label can consist of any combination of letters, digits, and underscores (maximum 12 characters). At least one character must be a letter. An SSENTRY label cannot be used more than once in a unit.
- The SSENTRY command identifies the step in the normal SFC that you want to use to restart the processing after a safe-state has completed execution.
- The SSENTRY command does not need a corresponding SSRETURN but can be referenced by more than one SSRETURN command.
- The SFC containing the step identified by the SSENTRY label must reside in the same unit as the SFC containing the SSRETURN statement that uses that label.
- You can include as many as 10 SSENTRY statements in one step. The SSENTRY statements must precede any other statements in the step.
- The step must consist of RLL statements only.

NOTE: If an SSENTRY command is in a subordinate SFC and a safe-state returns to that step, that step becomes active as well as the step that calls the subordinate SFC. If this is a nested subordinate SFC, all calling steps become active up to the main SFC calling step.

Safe-State Commands (continued)

SSRETURN Command

The SSRETURN command transfers program execution to the step identified by the corresponding SSENTRY label.

For the SSRETURN command, use the format:

SSRETURN label;

- You can use the SSRETURN only in the end step of a main safe-state SFC.
- The label can consist of any combination of letters, digits, and underscores (maximum 12 characters). At least one character must be a letter.
- The label must be defined by an SSENTRY statement in a “normal” (non-safe-state) SFC. An SSRETURN command terminates the execution of a safe-state SFC.
- When an SSRETURN statement executes, the step containing that statement is deactivated; and the step that contains the SSENTRY command with the same label becomes active.
- The SFC containing the step identified by the SSENTRY label must reside in the same unit as the SFC containing the SSRETURN statement that uses that label.
- The SSRETURN command of a local safe-state SFC must reference the SFC with which the local safe-state SFC is associated. The SSDEFINE command defines the SFC to which the local safe-state SFC is local.
- You can include more than one SSRETURN statement in a step of a main safe-state SFC, but these can be the only types of statements in the step.

NOTE: After the SSRETURN command transfers control back to the non-safe-state SFC step that is identified by the SSENTRY command, the statements in that step execute at least one time before another safe-state interruption can occur.

⚠ WARNING

APT allows you to specify any step as a return point for a safe state. Do not program a safe-state condition to return to a single step within a parallel structure.

If your return point is a single step within a parallel structure, this step is the only one that is active when you return, and your program is unable to advance past the parallel structure. The consequence is unpredictable operations that could result in death or serious injury to personnel, and/or damage to equipment.

Figure 13-1 illustrates the type of safe-state return condition that you must avoid.

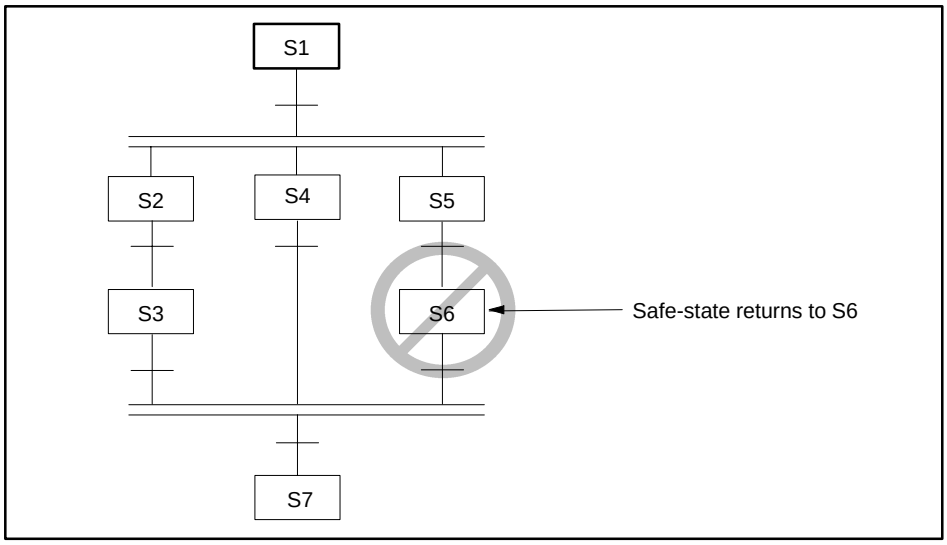


Figure 13-1 Avoid Safe-State Return to Parallel SFC Steps

Safe-State Commands (continued)

SSDEFINE Command

The SSDEFINE command defines an SFC as a safe-state SFC and specifies the priority of a safe-state SFC.

For the SSDEFINE command, use the format:

SSDEFINE LEVEL level;

SSDEFINE LEVEL level RETENTIVE;

SSDEFINE LOCAL TO sfc_name;

SSDEFINE LOCAL TO sfc_name RETENTIVE;

- Use the SSDEFINE command only in the initial step of a safe-state SFC.
- The initial step of every safe-state SFC must contain one and only one SSDEFINE statement, which must be the first statement in that step.
- The LEVEL must be an integer from 1 to 5. Level 1 has the highest priority; Level 5 has the lowest.
- The SFC name is the normal SFC with which the local safe-state SFC is associated.
- The priority determines what occurs when a safe-state SFC is triggered during the execution of another safe-state SFC.

  One safe-state SFC can interrupt the execution of another safe-state SFC only if the priority of the triggered safe-state SFC is higher than the priority of the currently executing safe-state SFC.

  If the priority of the triggered safe-state SFC is lower than or equal to the priority of the currently executing safe-state SFC, the execution of the currently executing SFC continues uninterrupted.

  Only one local safe-state of any given SFC can be active at one time. However, multiple local safe-state SFCs can be active if they belong to different SFCs. It is possible for two local safe-state SFCs in the same subordinate path to become active at the same time if the lowest subordinate SFC becomes active first.

- A subordinate safe-state SFC must have the same priority level or the same normal SFC name (local safe-state SFCs) as the SFC that calls it.
- The RETENTIVE option specifies that the safe-state SFC remains armed if it is armed when a power failure occurs. If you do not include the RETENTIVE option, a power failure effectively disarms a safe-state SFC.

  If you define a safe-state SFC with the RETENTIVE option, you do not have to enable the unit to restart the process.
- For Series 505 controllers, the step must consist of RLL statements only; this restriction does not apply to S5 controllers, into which only STL is loaded.

Safe-State Commands (continued)

SSTRIGGER Command

The SSTRIGGER command specifies a condition that triggers the execution of a main safe-state SFC.

For the SSTRIGGER command, use the format:

SSTRIGGER identifier;

- Use the SSTRIGGER command only in the initial step of a main safe-state SFC.
- The identifier is the name of a digital I/O point or a boolean or APT flag variable that triggers the execution of the safe-state SFC when it becomes true.
- When a safe-state SFC is armed and its trigger condition becomes true, it interrupts the execution of the currently executing SFC. A safe-state SFC can interrupt a “normal” (non-safe-state) SFC or a safe-state SFC with a lower priority.
- A triggered safe-state SFC does not interrupt the processing of a safe-state SFC with a higher or equal priority.
- If the trigger condition becomes true when the safe-state SFC is disarmed, the condition is ignored.
- The initial step of every main safe-state SFC must contain at least one SSTRIGGER, which must appear immediately following the SSDEFINE.

- You can use any number of SSTRIGGER statements in the initial step of a main safe-state SFC, but all statements must immediately follow the SSDEFINE statement and precede any other types of statement.

- To trigger a safe-state SFC after recovery from a power failure, specify the power failure detection bit as the trigger condition (SSTRIGGER *program_name*.**PWRFL**;). Program execution begins with the initial step of that safe-state SFC, assuming that the SFC was armed at the time of the power failure, and that the SFC was defined as retentive.

  The SFC cannot be a local safe-state SFC. A local safe-state SFC cannot become active until its associated SFC is active. The associated SFC requires several scans by the controller before becoming active; but the **.PWRFL** extension remains true for only one scan after return from a power failure.

  A retentive safe-state SFC with a **.PWRFL** extension as a trigger must be a Level 1 safe-state SFC and must be the only safe-state SFC at Level 1. If another retentive safe-state SFC at the same or higher level is active when the power fails, the safe-state SFC with the **.PWRFL** extension as a trigger does not execute.

Safe-State Commands (continued)

SSARM Command

The SSARM command "arms" a safe-state SFC and, thus, makes it possible for the safe-state SFC to interrupt the processing if the trigger becomes true.

For the SSARM command, use the format:

SSARM sfc_name;

- The *sfc_name* must be the name of a main safe-state SFC residing in the same unit as the currently executing SFC.
- A safe-state SFC can interrupt the processing of another SFC only if you have armed the safe-state SFC by using the SSARM command.
- When you use the SSARM command to arm a safe-state SFC, the program starts to monitor the conditions that are specified in the SSTRIGGER statement.
- A safe-state SFC interrupts the currently executing SFC and begins execution only when all of the following conditions are true:

  The safe-state SFC is armed. (The program constantly checks trigger conditions but ignores them until the safe-state is armed.)

  The currently executing SFC is either a "normal" (non-safe-state) SFC, *or* a safe-state SFC with a priority that is lower than the priority of the triggered safe-state SFC.

  A trigger condition becomes true.
- The SSARM remains in effect until an SSDISARM command cancels it. If the safe-state SFC is not defined as RETENTIVE, it is also disarmed by a controller power failure.

SSDISARM Command

The SSDISARM command "disarms" a safe-state SFC and, thus, does not allow a safe-state SFC to interrupt SFC processing.

For the SSDISARM command, use the format:

SSDISARM sfc_name;

- The *sfc_name* must be the name of a main safe-state SFC residing in the same unit as the currently executing SFC.
- When you use the SSDISARM command to disarm a safe-state SFC, the safe-state SFC can no longer interrupt the processing of another SFC.
- The SSDISARM command takes precedence over an SSARM command.

NOTE: When a unit is first enabled, all safe-state SFCs are initially disarmed.

Safe-State Commands (continued)

SSABORT Command

The SSABORT command suspends SFC execution in the current unit.

For the SSABORT command, use the format:

SSABORT;

- Use the SSABORT command only in the end step of a main-level safe-state SFC. You cannot use the SSABORT command with a local safe-state SFC.
- The SSABORT statement must be the only statement in the step; that is, a step that contains an SSABORT statement cannot contain any other statements.

When you use the SSABORT command to suspend unit execution, all SFCs in the current unit become inactive. The SFCs in the unit remain inactive until you re-enable the unit.

- The SSABORT command does not actually disable the unit but deactivates the current step and leaves no steps active in any SFCs in the unit.
- The SSABORT command does not disarm any armed safe-state SFCs; therefore, it is possible for an aborted unit to become active again if the trigger of an armed safe-state becomes true.
- The SSABORT command does not affect CFBs, devices, etc. in the unit.

To re-enable the unit and reactivate the main SFC after an SSABORT has executed, you must set the *unit_name*.**ENABL** bit to false and then set it back to true. You can do this with any one of the following methods.

- Use the modify option in the APT Debug Utility.
- Configuring the reset through an operator interface.
- Use an interlock CFB with an assignment statement such as *unit_name*.**ENABL** := X1, where X1 is a Digital Input (DI) or a boolean variable that you can use to write to the *unit_name*.**ENABL** bit to false and then back to true.

⚠ WARNING

If you shut down an SFC using the SSABORT command, the safe-state SFC remains armed. It can be reactivated and cause other portions of your SFC to begin again through the SSRETURN command.

If your safe-state SFC is still armed when your SFC reactivates, it could cause unexpected operation of your application, resulting in death or serious injury to personnel, and/or damage to equipment.

When you use SSABORT, always disarm your safe-state SFC.

13.3 Safe-State SFC Examples

Emergency Procedures

A main SFC named *chrg_raw* controls the charging of raw material into a reactor. The pressure in the reactor increases during this procedure. If the pressure becomes too high, an emergency procedure must be performed; the safe-state SFC named *hiprs_rw* controls this emergency procedure.

Figure 13-2 illustrates the sequence of events that occurs when *chrg_raw* is interrupted by *hiprs_rw*. The control flow follows:

- Step S1 of *chrg_raw* arms the safe-state SFC *hiprs_rw*. The program checks the initial step (S1) of *hiprs_rw* to determine the trigger condition (*pit101*.**INHHA**).

  Since the safe-state SFC is now armed, the program begins to monitor the status of the trigger condition.

- While *chrg_raw* is still in Step S3, the specified trigger condition of safe-state SFC *hiprs_rw* (*pit101*.**INHHA**) becomes true, indicating that the pressure is too high.

  At this point, APT checks the priority of *hiprs_rw* (2) against the levels of any other currently executing safe-state SFCs. If no Level 1 or Level 2 safe-state SFCs are executing, *hiprs_rw* can interrupt the process.

- Step S3 of *chrg_raw* deactivates, and the initial step of *hiprs_rw* is activated; *hiprs_rw* executes normally, and all steps in *chrg_raw* remain inactive.

- When *hiprs_rw* reaches its end step (S8), it immediately transfers execution to the step containing the statement, SSENTRY *hi_pres_lvl;* and *chrg_raw* begins to execute normally from that step. (Step S2 of *chrg_raw* is activated, and Step S8 of *hipres_rw* becomes inactive.)

  Because *hiprs_rw* is still armed and is no longer executing, the status of its trigger condition (*pit101*.**INHHA**) continues to be monitored. If the pressure becomes too high, the process is interrupted again.

- When *chrg_raw* reaches its end step (S4), *hiprs_rw* is disarmed. The status of its trigger condition is no longer monitored, so *hiprs_rw* can no longer interrupt normal processing.

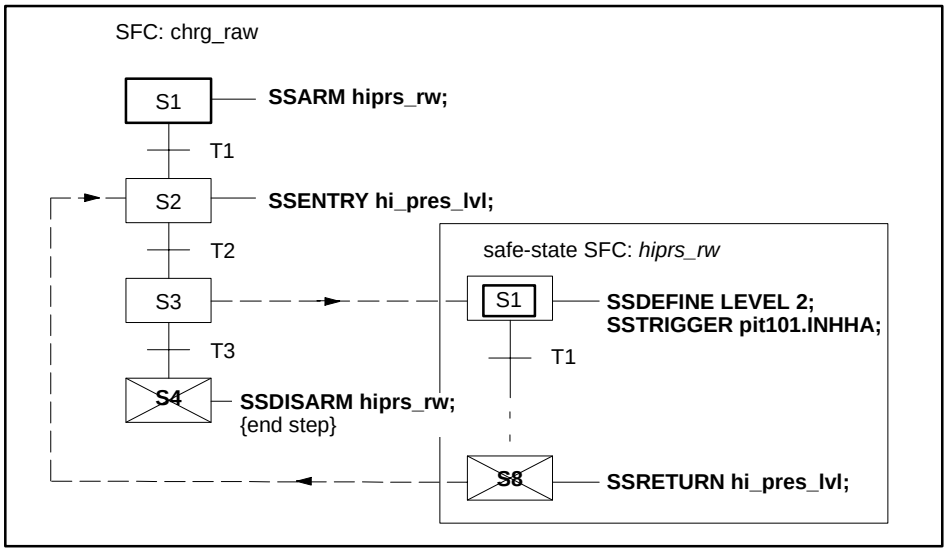


Figure 13-2 Safe-State SFC Execution

NOTE: If the safe-state *hiprs_rw* cannot lower the *pit101*.**INHHA** alarm during the execution of the safe-state, *hiprs_rw* executes repeatedly. If your intention is to execute a safe-state action only once, you can place the following declaration in a high priority interlock block: *pit101_htrig* := edge(*pit101*.**INHHA**). Then use *pit101_htrig* as the trigger for *hiprs_rw*.

Alternately, you may disarm the safe-state in the initial step of the safe-state.

Multiple Safe-State SFCs

Figure 13-3 illustrates how more than one safe-state SFC can interrupt a main SFC (a *main* SFC arms safe-state SFCs *ss2*, *ss1*, and *ss4)*:

- ss2 is a level 2 safe-state SFC that triggers when the temperature is too high.
- *ss1* is a level 1 safe-state SFC that triggers when the pressure is too high.
- *ss4* is a level 4 safe-state SFC that triggers when the pressure is too low.

Each of the safe-state SFCs is designed to return control to the initial step of *main*.

The following sequence of events is illustrated in Figure 13-3:

- When the temperature becomes too high, *ss2* interrupts *main* (Figure 13-3a).
- While *ss2* is executing, the pressure becomes too high. As a result, *ss1* (a Level 1 safe-state SFC) interrupts *ss2* (a Level 2 safe-state SFC). After *ss1* completes execution and returns control to *main* (Figure 13-3b).
- Because the temperature remains too high, *ss2* interrupts *main* as soon as the return step in the main SFC completes one execution. While *ss2* is executing, the pressure becomes too low. However, *ss4* (a Level 4 safe-state SFC), cannot interrupt *ss2* (a Level 2 safe-state SFC). When *ss2* completes execution, control returns to *main* (Figure 13-3c).
- Because the pressure remains too low, *ss4* interrupts *main* as soon as the return step in the main SFC completes one execution (Figure 13-3d).

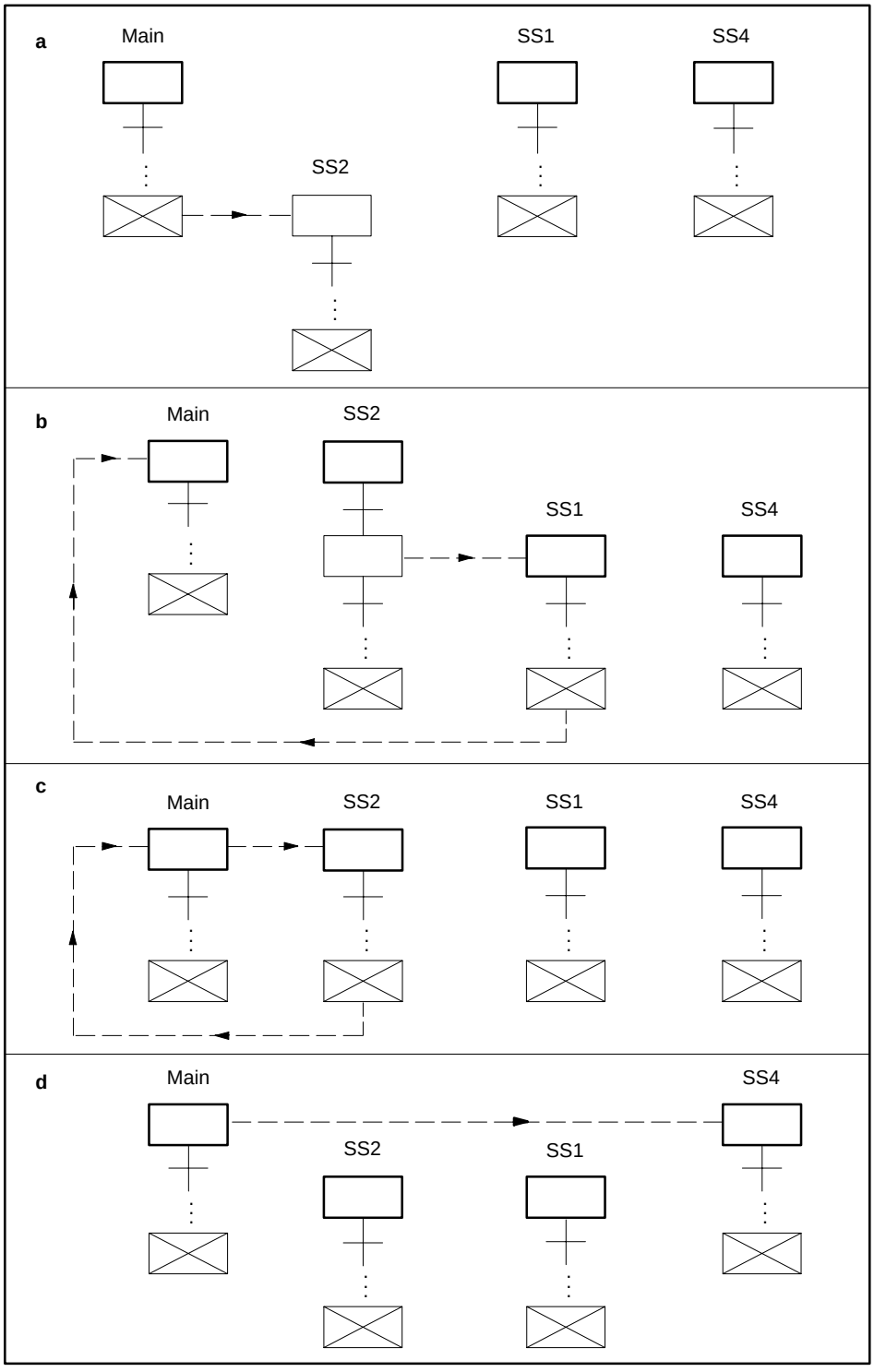


Figure 13-3 Safe-State Examples

Appendix A

Error Messages

A.1 Understanding the Compile Operation

Overview

After you compile a program, APT displays one of the following messages:

- **Program compiled SUCCESSFULLY!**
- **Program compiled successfully - with WARNINGS.**
- **Program compilation cancelled prior to completion!**
- **Compile not required. Object code is up to date.**
- **Program compile FAILED.**

The *COMPILE.RPT* file contains specific details about the compile.

Compiling with Debug

If you compile a Debug version of your program, the messages are not affected. The Debug Version does, however, require more controller memory. When you select the Debug version, APT creates additional code to the object so that the Debug utility options are available.

A.2 Successful Compile

Overview

When the Compile operation completes successfully, you can receive one the following messages:

- **Program compiled SUCCESSFULLY!**
- **Program compiled successfully - with WARNINGS.**

With a successful Compile, an Object File is created; and the program is ready to be downloaded to the controller. The Object File is built according to the specifications in the Compiler Control File.

You cannot use the APT Debug utility until you have a successful compile.

With a successful compile, the *COMPILE.RPT* file includes specific information about the compile and the amount and types of memory that are required for the program.

When a compile completes successfully with warnings, it means that although an object has been created, APT has found one or more items within the program that indicate a problem.

⚠ WARNING

APT provides compiler warnings whenever something is detected that might cause problems. These warnings are not compiler errors and do not stop you from downloading your program to the controller.

If you download your program to the controller without taking compiler warnings into consideration, your process could be adversely affected. Unpredictable operation of the controller can result in death or serious injury to personnel, and/or damage to equipment.

Read each compiler warning carefully and take appropriate action to correct warning conditions before you download the compile object file into your controller. A list of compiler errors and warnings, in numerical order, is provided in Table A-1.

A.3 Compile Report Messages

Overview

The compile operation may not complete for several reasons:

- **Program compilation cancelled prior to completion!**

  This message indicates that you have aborted your compile by pressing ESC. No object file has been created.

- **Compile not required. Object code is up to date.**

  This message indicates that you have not changed your APT program since your last successful compile. If you want to recompile, select **Force Compile** in the Compiler Control File.

NOTE: Be certain that you keep track of any warnings that exist in your program. A recompile that results in this message creates a new *COMPILE.RPT* without listing the warnings.

- **Program compile FAILED.**

  This message indicates that your program contains errors and a program Object File could not be created. You must correct all errors before you can create an OBJECT file for downloading.

Correcting Errors

If your compile fails, follow these steps:

1. Review the *COMPILE.RPT* report for information on the errors identified in the compile. The error messages identify the type of error and its location in your program. These messages are explained in Table A-1.

2. Correct all of the errors that you know how to fix, and re-compile your program.

   Some mistakes within your program can cause more than one error in the *COMPILE.RPT*. There are errors that, when identified during the compile, can cause a compile phase to be discontinued. Thus, if you fix an error of this type, the next compile continues further into your program but identifies other errors that were not yet encountered.

NOTE: The APT **Validate** option can be run from various levels in the hierarchy to check individual areas of your program for syntax-related errors. This can save you time.

The APT **Validate** option cannot find all problems that relate to using functions/options not supported by a specific controller model. This is identified in a later phase of the compile operation.

Repeat Steps 1 and 2 until you complete a successful compile. If you cannot fix all your problems following these steps, compile your program with the **Force Compile** selected in your Compiler Control File.

If you have followed the above steps and you still receive an Internal Error, call the Siemens Energy & Automation, Inc., Technical Services Group in the U.S.A. at (800) 333-7421. Outside the U.S.A., call 49-911-895-7000.

Table A-1 contains a list of errors that can appear in the *COMPILE.RPT* file. Many of these errors can be detected if you use the Validate option while you are building your program.

The line numbers for errors that occur in SFC Steps, Transitions and Math CFBs are calculated as accurately as possible but can be off by one or two lines.

Several errors can be flagged due to a single mistake in your program. If multiple errors occur within the same area of your program, fix the obvious error, typically listed first.

General errors and graphical errors are included in the table.

When a range error occurs, the following formats are used:

- **x .. y** means that the value must be greater than or equal to x and less than or equal to y.
- **x >. y** means that the value must be greater than x and less than or equal to y.
- **x .< y** means that the value must be greater than or equal to x and less than y.
- **x >< y** means that the value must be greater than x and less than y.

Table A-1 Program Warnings and Errors

Number/Error	Explanation
Corrupt cross reference files. Force compile occurring to recover data.	The cross reference files were corrupted. A force compile is required. You will not be able to incrementally compile or download and you may have to retranslate marked tags.
Force compile occurring due to controller record being updated.	You made changes in the Compiler Control File requiring a force compile. You will not be able to incrementally compile or download and you may have to retranslate marked tags.
Force compile occurring due to no previous object information existing.	The APT program was never compiled, the object code was deleted, or the archive that this program was restored from was archived with the source only. You will not be able to incrementally compile or download and you may have to retranslate marked tags.
Unit was unmarked or deleted since previous compilation.	A unit was unmarked in the Compiler Control File or a unit was deleted from the hierarchy.
New information was marked and translate flag is set "YES".	Additional objects were marked for translate and the Build Translate table option in the Compiler Control File was set to YES.
Change in the module editor requires reallocation of overflow CRs.	For Series 505, additional I/O points were added to addresses previously used as overflow CRs. The overflow CRs must start after the last configured I/O address for each channel.
Translate build flag changed to YES from NO or APPEND states.	The Build Translate table option in the Compiler Control File was changed to YES from NO or APPEND.
Assembly force compile is required.	Changes in APT require a force compile for phase 6 only. You will not be able to incrementally compile or download and you may have to retranslate marked tags.

Table A-1 Program Warnings and Errors (continued)

Number/Error		Explanation
2003	**Program compiled with no units marked.**	The program has units, but they are not marked for compile. The information in all units will not be compiled and will not exist in the controller.
2007	**Unit ... contains no main SFC.**	An SFC needs to be promoted in this unit. Each unit within your program must have one and only one main (promoted) SFC, which is used as the starting point of all sequential control for that unit.
2013	**Undefined controller record.**	This message is caused by an error in the Compiler Control File. A field has been left empty or contains an invalid entry.
2014	**No controller version number given.**	This message is caused by an error in the Compiler Control File. The controller version field has been left empty or contains an invalid entry.
2015	**No TISOFT version number given.**	(Series 505 only) This message is caused by an error in the Compiler Control File. The TISOFT version field has been left empty or contains an invalid entry.
2016	**Cannot find APT connector in parallel port.**	This error does not apply to APT Release 1.9 and later.
2019	**APT SYSTEM ERROR: Program file not found....**	APT attempted to execute one of the compile phases, e.g., SFC, NNL, assembler, etc., and was unable to find an executable program. Make sure that the DOS path includes the \apt subdirectory and that all files with .exe extensions are in this subdirectory.
2020	**APT SYSTEM ERROR: DOS error executing... Return code...**	An error occurred while executing one of the compile phases. Exit from APT and run the DOS CHKDSK /F command; and then reboot. The return code is a specific DOS error number. Refer to the DOS user manual for an explanation.
2023	**Program compile interrupted: Swap file load failed ....**	The compile was interrupted by a Ctrl-Break from the keyboard, or an error occurred while loading an executable swap file. Try to compile the program again. If this fails, exit from APT and run the DOS CHKDSK /F command; and then reboot.
2024	**No main global SFC.**	An SFC is not promoted at the global level. Promote one SFC as the main SFC.

Table A-1 Program Warnings and Errors (continued)

Number/Error		Explanation
3003	**No address assigned** **... must be addressable** **cannot use non-addressable objects**	No controller address was assigned, but the variable is being used by the program. Check that the variable is not declared with the NONE attribute under the controller address.
	...Cannot be constant and must be addressable	Trying to change a variable that was declared a constant or that accesses a variable that has no controller address. Check to see whether the variable is declared 'constant' or the NONE attribute under the controller address was selected.
	...Cannot be a WI, BI, or constant and must be addressable	Trying to change a variable that was declared a WI, BI, or a constant or that accesses a variable that has no controller address. Check to see whether the variable is a WI, BI, or constant, or whether the NONE attribute under the controller address was selected.
	...must be an array of size 11 and must be addressable	When using a correlated look-up table or value sequencer, the array size for the input value array and the output value array is eleven elements. Also, the array must contain controller addresses.
	Cannot copy a constant recipe **Cannot move a constant recipe** **Cannot select a constant recipe** **Cannot unblock a constant recipe** **Cannot clear a constant recipe**	A constant recipe cannot be copied, moved, selected, cleared, or unblocked because it cannot be written to. A constant recipe does not use any of the recipe extensions. Check that the recipe was not declared constant.
	Parameters must have same recipe template	Cannot copy to a recipe that does not use the same recipe template. The destination recipe and source recipe must be made from the same recipe template. Check that they are using the same recipe template.
	Address type must be WX	For AI, RT, WI, BI, and TC input modules, the address must be WX for Series 505 controllers.
	Address type must be WY	For an AO, BO, and WO output modules, the address must be WY for Series 505 controllers.
	Address type must be X	For a DI, the address must be X for Series 505 controllers.
	Address type must be Y	For a DF or DO the address must be Y for Series 505 controllers.
	Cannot filter with no scale range selected	Cannot select the No Scale option with the Filter option. Check attributes of your analog input and see if both are selected and choose only one.
	Only discrete modules are valid over addresses 1024 for this controller	The 545 release 2.x and greater can have 2048 discrete points. Check that only discrete I/O modules are used in the second 1024.
	Controller type and block type are incompatible	The 560T does not have the capability of handling floating point numbers (reals). The CFB you are using uses floating point numbers in its calculations. This error also occurs with an S5 controller when you try to use an unsupported CFB.
	Scan time must be 0 for variable scan type	If you select the scan time for the controller to be VARIABLE (which is the default), the total scan time must be set to 0 msec in order for the variable scan time to work properly. Check total scan time on the Compiler Control File to see that it is set to 0.

Table A-1 Program Warnings and Errors (continued)

Number/Error		Explanation
3003	**Cannot give both U-memory file name and a U-memory value**	If you want to use U-memory you must either allocate U-memory or give a U-memory filename; you cannot do both. Check the Compiler Control File to see if you have entered both, and use only one. Refer to the appropriate Series 505 programming reference manual in the section on External Subroutine Development.
	U-memory file does not exist	You entered a U-memory file name on the Compiler Control File, but you do not have a U-memory file in the **APT \PROGRAM*program_name*\PRR** subdirectory. The U-memory file is in S-record format for the Motorola CPUs. Check to see that it is in the **APT\PROGRAM*program_name*\PRR** subdirectory. Refer to the appropriate Series 505 programming reference manual, External Subroutine Development for details on the U-memory file.
	Value must be less than high range **Value must be greater than low range**	Check that the value entered is within the range of the variable. It must be less than the high range and greater than the low range.
	Value must be between ... and ...	Check the value entered is not greater than the upper limit or less than the lower limit.
	Controller type and I/O type are incompatible	I/O types: RT and TC are not supported for S5 controllers.
	PW addr. range is 0 ... 254; **OW addr. range is 0 ... 254;** **IW addr. range is 0 ... 126;**	You have entered an I/O input that is outside the supported address range for S5. Check your address.
	PW addr. range is 0 ... 254; **OW addr. range is 0 ... 254;** **QW addr. range is 0 ... 126;**	You have entered an I/O output that is outside the supported address range for S5. Check your address.
	I addr. range is 0 ... 127;	Digital inputs for S5 controllers must be in the 0–127 range. Check your address.
	Q addr. range is 0 ... 127;	Digital outputs for S5 controllers must be in the 0–127 range. Check your address.
3004	**... In ..., near line ...**	The listed error occurred near the specified line of a file.
3005	**... not defined** **... not defined type should be ...** **... not defined type should be ... In ..., near line ...**	The specified object has not been defined. It is referencing itself indirectly, or is at the wrong scoping level. An object's scoping level is determined by where it is defined in the hierarchy. Objects defined at the unit level can reference other objects defined within the unit and objects defined globally, but not objects defined within other units. Objects defined globally cannot reference objects defined at the unit level.

Table A-1 Program Warnings and Errors (continued)

Number/Error		Explanation
3006	**... type should be ... ... not defined in object ... , field ...** **... type should be ...** **in object ... , field ...** **... type should be ... ... was ... in object ... , field ...** **... type should be ... No value entered, no default value to substitute in object ..., field ...** **... type should be ... ... was ...** **... type should be ... ... was ... In ... , near line** **... type should be ...** **... type should be ... No value entered, substituted default value ... in object ..., field ...**	The specified object or verb parameter was of the wrong type. It must be one listed in the error message. If the object is defined, its type is listed. It is possible for the type to be one of the listed types and still fail validation. When this is the case, the error message lists the reasons. (This message is displayed only if you are executing an action on an object using a verb.) (This message is displayed only if you are executing an action on an object using a verb.) (This message is displayed only if you are executing an action on an object using a verb. The message lists the reason for the error.)
3008	**... value ... not in range ... in object ... , field ...** **In ... , near line ...value ... not in range ... in object ... , field ...** **value ... not in range ...** **value ... not in range ... in ... , near line ...**	(The message lists the reason for the error.) The value is not in the correct range. The specified value is for the object's field or verb parameter must be within the correct range.
3009	**Controller type not defined.**	The correct controller model was not entered in the Compiler Control File.
3011	**Object ... does not exist.**	The specified object was referenced and is not defined or does not exist at the correct scoping level. Scoping level is defined under Error 3005.
3012	**Missing parameter for verb or function ...** **Missing parameter for verb or function ... In ... , near line ...**	The specified verb or function was expecting a parameter but none was specified.
3013	**Extra parameter ... for verb or function ...** **Extra parameter ... for verb or function ... In ... , near line ...**	An extra parameter was specified for the specified verb or function.

Table A-1 Program Warnings and Errors (continued)

Number/Error		Explanation
3014	**Object missing for verb or function ...**	The object of the verb was not specified.
3015	**Invalid verb ... for object ...** **Invalid verb ... for object ... In ..., near line ...**	The specified verb is not valid for the specified object.
3018	**Value ... cannot be a constant boolean.**	Value in specified field cannot be a constant boolean.
3019	**Value ... must be literal or declared constant in object ... , field ...**	Value in specified field must be a literal or declared constant. Check value and its scoping level. Scoping level is defined under Error 3005.
3020	**Declaration for ... not found.**	A declared variable was referenced and was not defined or was not at the correct scoping level. Scoping level is defined under Error 3005.
3021	**String ... truncated.**	An input line is too long.
3022	**Cannot open package body library ...**	The library that expands the control code for an object could not be opened. Therefore, the objects of the specified type will not contain any control code. This may be caused by the deletion or corruption of the library. Call (800) 333-7421 in the U.S.A., or 49-911-895-7000 outside the U.S.A., for assistance.
4007	**Not enough memory to load ... package**	The package that defines information for an APT object could not be loaded because there is not enough memory. Use memory management techniques to get more conventional memory. For instance, try to load as much as you can in extended memory using an extended memory manager. You need at least 510K in order to run APT. See the appendix called “APT Computer Memory,” in the *SIMATIC APT User Manual*.
4008	**Cannot open input library ...**	The library that defines information for an object could not be opened. Therefore, all the standard object types shipped with APT cannot be used correctly. This may be caused by the deletion or corruption of the library. Call (800) 333-7421 in the U.S.A., or 49-911-895-7000 outside the U.S.A., for assistance.

Table A-1 Program Warnings and Errors (continued)

Number/Error		Explanation
5018	**field was <empty>, type should be ... in object ..., field ...**	The field of the specified object is empty. It must contain one of the types listed in the message.
5019	**... type should be ... literal in object ... , field ...**	The field of the specified object must contain a literal of the type listed in the message.
5020	**Real addresses require ... memory locations**	For Series 505, integers require one memory location and real numbers require two memory locations. For example, V1. refers to both V1 and V2. For S5 controllers, integers require two flag memory locations and real numbers require four flag memory locations. For example, FD1 refers to FY1, FY2, FY3, and FY4. For data memory in DB/DXs in S5 controllers, integers require one memory location and real numbers require two memory locations. For example, DB3:D1. refers to both D1 and D2.
5021	**I/O name ... has no address assigned.**	The specified I/O object was not assigned an address in the module table.
5022	**No value entered, substituted default value ...**	The problem usually occurs on a restored program from a previous release. This goes away after the first compile.
5023	**No value entered, no default value to substitute.**	The problem usually occurs on a restored program from a previous release. The field must be initialized with a value.
5024	**Math was selected but not defined in object ... , field ...**	User-defined math was selected but no math statements were entered. If this message is displayed as a warning, the compile will not fail. If this message is displayed as an error, you must enter the math statements before the program will compile.
5026	**Constant value ... assigned to recipe element ... in recipe ... has not been declared.** **Constant value ... assigned to recipe element ... in recipe ... is not of the correct type.**	A problem was encountered assigning a constant value to the specified recipe element. It was not declared, not the correct type, or does not exist at the correct scoping level. Scoping level is defined under Error 3005.
5030	**Address format ... is invalid. (Start with "%", no " . " allowed).**	The address format is incorrect. It must be preceded with a % sign, contain an address type (e.g., V, C, K, etc., for Series 505 and FW, DW, SW, etc., for S5), and be followed by the address offset. It cannot contain any trailing characters, including a trailing period.
5031	**... is an invalid address in object ..., field ...**	The address type was not valid (i.e., VMX instead of VMM), the format was incorrect (i.e. %VMMFF00 instead of %VMM0FF00), and/or the address was not an odd VMM or VMS offset.
5032	**... requiring ... locations is out of range (... reserved) in object ..., field ...** **... requiring ... locations is out of range (... reserved) value ... not in range ... returned address cross retentive/non-retentive boundary in object ..., field ...**	All elements of an array with a user defined address must be within the reserved address range. The specified range in the message is the closest valid range based on the starting address and array length. For Series 505 boolean arrays, requested array range may cross retentive/non-retentive boundary.

Table A-1 Program Warnings and Errors (continued)

Number/Error		Explanation
5058	**Converted value ... in float field to ...**	The object field requires a float value. The integer was automatically converted to a float. This goes away after the first compile.
5059	**Cannot specify an address for recipe ... if it contains boolean elements.**	You cannot assign a user address to a recipe if any of the elements are boolean.
5060	**...requiring ... locations is out of the reserved range**	All elements with a user-defined address must be within the reserved address range. Check to see that enough memory was reserved in the Compiler Control File.
5061	**Address format for ... requires block specifier (i.e., %DB3:DW4)**	When using data word memory in the S5, you must specify the data block that the data word is in by using the following format: %DBx:DWy, where the x refers to the block number and the y refers to which data word.
5062	**I/O address ... is assigned to more than 1 point.**	In checking the I/O database, two I/O points are assigned to the same address. Check your addresses to see where the overlap occurs. Generate an I/O report by entering an "R" on the I/O table.
5063	**I/O address ... has incorrect bit value.**	When using a digital point in S5, the bits range from 0–7. Therefore your address to the right of the "." must also range from 0–7. For example, addresses I1.0 to I1.7 are correct, but I1.8 is not permitted.
5064	**I/O address ... has incorrect format, should be %[Memory type] [address].**	For S5 controllers, the addresses have the following format: %[Memory type] [address], e.g., %IW10 or %PW128. See the chapter called "I/O Symbolic Names," in the *SIMATIC APT Programming Reference (Tables) Manual*.
5066	**Invalid address ... crosses 256 word data block boundary.**	A recipe or array that has a user-defined address is starting at an address that requires some of the array or recipe to cross data blocks. For instance, if you have a recipe that requires 250 words, and you assign a starting address of DB10:DW20, then the recipe requires memory past DB10:DW255. You must change the starting address so that the recipe or array will fit into 256 words.

Table A-1 Program Warnings and Errors (continued)

Number/Error		Explanation
7000	**The Sub-SFC ... that is called from ... doesn't exist.**	The SFC references a Sub-SFC that does not exist. Correct the spelling or create the named SFC.
7001	**SFC ... is called from more than one SFC (... and ...).**	An SFC can only be referenced from one SFC.
7002	**SFC ... is not called.**	An SFC must be a main SFC or referenced by another SFC.
7003	**SFC ... and Sub-SFC ... are different level safe-state SFCs.**	All safe-state sub-SFCs must be the same level as the calling SFC.
7004	**SFC ... is attempting to call the main SFC ... Cannot call the main SFC.**	The main SFC cannot be called from another SFC.
7005	**SFC ... is attempting to call safe-state SFC ... No calls allowed to safe-state SFCs.**	A normal SFC cannot call a safe-state SFC.
7006	**The trigger ... appears at more than one level.**	An SSTRIGGER statement appears in two safe-state SFCs with the same priority.
7007	**The trigger ... is used more than once at the same level.**	The same trigger has been used for two safe-state SFCs that are defined at the same level. Change the trigger or the level for either SFC.
7009	**SFC ... cannot disarm or arm ..., it is not a SFC.**	The SFC tries to disarm or arm a safe-state SFC that does not exist. Correct the spelling or create the named SFC.
7010	**SFC ... cannot disarm or arm SFC ..., it is not a main safe-state SFC.**	An SFC can only disarm or arm a main level safe-state.
7011	**Duplicate safe-state entry point ... in SFCs ... and ...**	An SSENTRY point is defined more than one time. Remove duplicate SSENTRY points.
7012	**Safe-state entry point ... referenced in SFC ... does not exist.**	An SSRETURN statement has a label that does not exist or cannot be found. Check for typographical errors or a label that is more than 12 characters; verify that a valid SFC contains the proper SSENTRY statement.
7013	**Maximum SFCs of 64 exceeded in unit ...**	You can have no more than 64 SFCs in any one unit.
7014	**The local SFC ... identified in safe-state ... does not exist.**	A local safe-state must reference a normal SFC. Correct the spelling or create the named SFC.
7015	**The ssreturn ... in SFC ... does not return to the local-to SFC ...**	An SSRETURN statement of a local safe-state SFC must return to the SFC to which it is local. Change the SSRETURN statement so that it points to an entry point inside the LOCAL-TO SFC.
7016	**SFC ... is not local to the same SFC as its parent ...**	All safe-state sub-SFCs must be the same level as the calling SFC.

Table A-1 Program Warnings and Errors (continued)

Number/Error		Explanation
8000	**[Graphical Error] Destination step S... of goto at tile ... not defined.**	An SFC graphical GOTO is not properly defined. A graphical GOTO must have a destination step.
8001	**[Graphical Error] Invalid symbol at location ....**	An unrecognized or invalid symbol is found.
8002	**[Graphical Error] Line leads off top of SFC.**	A line leads off the top of Row 1. This can occur if you delete the top row of the SFC.
8003	**[Graphical Error] Line leads off bottom of SFC.**	A line leading off the bottom of Row 100.
8004	**[Graphical Error] Line leads off left of SFC.**	A line leads off the left of Column A.
8005	**[Graphical Error] Line leads off right of SFC.**	A line leads off the right of Column Y.
8006	**[Graphical Error] Illegal parallel construct at location ..., invalid starting path.**	A path in the parallel construction is not valid. Be certain that a single path leads from a transition to the top parallel bar. At least two paths must lead down from the top bar, and each path must lead to a step, not a transition. A path cannot pass through two steps or two transitions in a row.
8007	**[Graphical Error] Illegal parallel construct at location ..., multiple transitions beginning.**	Two or more transitions are leading into a beginning parallel bar. Be certain that a single path leads from a transition to the top parallel bar.
8008	**[Graphical Error] Illegal parallel construct at location ..., multiple transitions ending.**	Two or more transitions are leading out of an ending parallel bar. Be certain that a single path leads down from the bottom parallel bar to a transition.
8009	**[Graphical Error] Illegal parallel construct on row ..., must begin at least 2 steps.**	A parallel SFC structure must have at least 2 paths leading from the beginning parallel bars. Each path must lead to a step, not a transition.
8010	**[Graphical Error] Illegal parallel construct on row ..., must end at least 2 steps.**	A parallel SFC structure must have at least 2 paths leading into the ending parallel bars. Each path must lead from a step, not a transition.
8011	**[Graphical Error] Illegal select construct on row ..., malformed select.**	An invalid divergence or convergence line is found in the SFC. Be certain that only a single path leads into a divergence line or leads down from a convergence line. A path cannot pass through two steps or two transitions in a row.
8012	**[Graphical Error] Illegal select construct on row ..., too many paths exiting select branch.**	A single vertical path must lead down from a convergence line.
8013	**[Graphical Error] Illegal select construct on row ..., too many paths into select branch.**	A divergence line must be entered by a single path leading from a step. Be certain that the path is vertical and in a downward direction. Lines cannot converge into a path that connects a step with a divergence line. A path cannot pass through two steps or two transitions in a row.
8014	**[Graphical Error] No transitions found before step S... at tile ....**	A transition is expected. A path can never be traced from one step to another without going through a transition.
8015	**[Graphical Error] No transitions found after step S... at tile ....**	A transition is expected. A path can never be traced from one step to another without going through a transition.
8016	**[Graphical Error] No transitions found before exit step S... at tile ....**	A transition is expected immediately prior to the exit step. A path can never be traced from one step to another without going through a transition.

Table A-1 Program Warnings and Errors (continued)

Number/Error		Explanation
8017	**[Graphical Error] No initial step for SFC.**	Each SFC must have one and only one Initial Step.
8018	**[Graphical Error] Too many initial steps specified. Only 1 is allowed.**	Every SFC must have one and only one Initial Step.
8019	**[Graphical Error] Incomplete graphics tile in input at location ....**	An incomplete structure is found. Be certain that all lines are connected.
8020	**[Graphical Error] Step S... at location ... has the transition T... both before and after.**	In an SFC, the path from a transition cannot lead to the step immediately preceding that transition. Check for a graphical GOTO or a line that loops back to the step it exits.
8021	**[Graphical Error] Illegal connection at location ....**	Two SFC steps are incorrectly connected. The following situations cause this error: illegal divergence lines, illegal loop backs, lines not connected to transitions or steps, a graphical GOTO not following a transition.
8022	**[Graphical Error] A transition follows transition at location ....**	This error is caused by an invalid SFC construction. A path traced from one step to another must pass through a single transition.
8023	**[Graphical Error] A transition precedes transition at location ....**	This error is caused by an invalid SFC construction. A path traced from one step to another must pass through a single transition.
8024	**[Graphical Error] Maximum step / transition label exceeded at location ...**	Only 500 steps / transitions are allowed in a single SFC. This is usually resolved by resequencing the SFC.
8050	**Duplicate step S... at location ...**	A duplicated step is found. Use Resequence option to correct.
8051	**Duplicate transition T... at location ...**	A duplicated transition is found. Use Resequence option to correct.
8052	**Left parenthesis missing.**	A left parenthesis is expected in the described location.
8053	**Right parenthesis missing.**	A left parenthesis appears without a right parenthesis.
8054	**Parenthesis mismatch.**	An odd number of parentheses occurs within a verb operation.
8055	**Right bracket missing.**	An array reference appears without a right bracket.
8056	**Illegal indexing of array.**	A non-integer value is used to index an array, or the index is outside the declared range of the array.
8057	**Illegal character ... detected.**	An unexpected character is encountered. This may indicate an attempt to use a real operator with integer or boolean variables. This often causes other errors to follow. This message is displayed when Math statements are placed in the parallel portion of a step. To correct this error, be sure to use the MATH and BEGIN statements to separate the math and the parallel portions within a step.
8058	**'=' character expected.**	A relational operator is expected.
8059	**Expression not valid outside of Math block**	A math statement appears in the parallel section of an SFC step.
8060	**Invalid identifier ....**	Identifier in the math statement is invalid. Be certain that the variable is declared with the appropriate type (integer, boolean, or real) and at the appropriate level (Unit or Program.)

Table A-1 Program Warnings and Errors (continued)

Number/Error		Explanation
8061	**Invalid direct controller identifier ....**	A direct controller address has been found that is invalid. Be certain that the syntax of the identifier and the controller size are correct. An example of an invalid address is %WW97. (WW is not a valid controller address type.)
8062	**Comment not closed.**	The closing delimiter for a comment is not found. Comments should be enclosed with one of the following delimiters: {...} or (*...*).
8063	**Invalid based integer ....**	Base other than 2 (binary), 10 (decimal), or 16 (hexadecimal) appears in the program.
8064	**Safe-state level expected.**	An SSDEFINE statement appears without a valid safe-state priority. Valid priority levels are 1, 2, 3, 4, and 5.
8065	**Semicolon expected before ...**	End of a statement is expected. APT statements must end with semicolons.
8066	**Integer value expected.**	An integer is expected. Array references and assignments with integers and some functions and procedure require integer variables.
8067	**Identifier expected.**	The statement requires an identifier. Check for an incomplete expression or statement.
8068	**Invalid hexadecimal integer ....**	An invalid hexadecimal integer is found. Check references in the Declaration Table.
8069	**Invalid binary integer ....**	An invalid binary integer is found. Check references in the Declaration Table.
8070	**Invalid decimal integer ....**	An invalid decimal integer has been encountered. Check references in the Declaration Table.
8071	**Based integers not allowed in expression.**	A decimal integer is required. The expression does not support bases of 2 or 16. If this occurs the binary or hexadecimal value must be replaced with its decimal equivalent. An SSDEFINE statement requires a literal decimal value to define the safe-state priority.
8072	**Invalid safe state level.**	A safe-state priority other than 1,2,3,4, or 5 is found.
8073	**Undeclared identifier ...**	Identifier is not declared in I/O Symbolic Table, Declaration Table, Recipe Templates, or in list of object extensions. This can also occur because of a syntax error in an APT command or an illegal math statement in the parallel section of an SFC step. For the 560 and 560T controllers, this error can occur when an analog input module is used. If the variable in the error is of type AI and you are using one of these controllers, be sure to use the **.RAW** extension of the AI point. This is because APT cannot scale or filter analog alarms with these controllers.
8074	**Invalid symbol type in expression.**	The symbolic name type of an identifier does not match the required type (real, integer, boolean, APT flag, and array).
8075	**Array index not valid.**	A non-integer value is used to index an array, or the index is outside the declared range of the array. Some instructions use only the array name (without an index). This error also occurs when you index a non-array variable.
8076	**Array index required.**	An array variable appears without an index.

Table A-1 Program Warnings and Errors (continued)

Number/Error		Explanation
8077	**Invalid real number ....**	An invalid real number or real number expression is found.
8078	**No SSABORT or SSRETURN allowed in step.**	An SSABORT or SSRETURN statement appears in a non-safe-state SFC end step.
8079	**Identifier expected.**	The statement requires an identifier. Check for an incomplete expression or statement.
8080	**Only SSABORT or SSRETURN allowed in step.**	End step of a safe-state SFC contains something other than an SSABORT or SSRETURN statement.
8081	**An SFC statement must be the only statement in a step.**	The call to a subordinate SFC, can be the only statement in the calling step.
8082	**Too many SSENTRY statements declared for step.**	The step contains more than the allowed number of safe-state entry point statements. There can be up to 10 SSENTRY statements in a step.
8083	**':=' operator expected.**	Assignment operator (:=) is expected in assignment statement.
8084	**Right side of assignment statement missing.**	Assignment statement not complete. Check for missing information on the right side of a assignment (:=) operator.
8085	**Unexpected input '...'.**	Element is not expected in the specified location. This message can occur if math expressions are placed in a transition.
8086	**Flag variable required for command.**	Command (ON, LATCH, CLEAR) requires an APT flag variable. Check Declaration Table or I/O Symbolic Table.
8087	**SFC file ... either empty or corrupted.**	Compiler cannot compile the specified SFC. Be certain that the SFC is valid. If the problem persists it may indicate a database or operating system problem.
8089	**Illegal mixing of types in assignment or boolean statement.**	A value or expression is assigned to a variable of a different type. For example, INT1 := true where INT1 is an integer. This error also occurs when a step or transition statement should evaluate to a boolean but does not. This message is also displayed when an assignment is made such as Recipe1:= Recipe2; and Recipe1 and Recipe 2 have different templates.
8090	**Boolean expression expected.**	A boolean expression is expected. Transitions must contain a single boolean expression. A blank transition can also cause this error.
8091	**Arrays are not of the same size.**	Certain APT statements require that the included arrays have the same declared number of elements.
8092	**Identifier ... is read-only.**	Variable is a read-only (a declared constant or an input such as a WI).
8093	**SSENTRY statement must be first statement in step.**	An SSENTRY statement appears some place other than the first line of an SFC step. Multiple SSENTRY statements are allowed as long as they appear before any other executable statements.
8095	**Real value . . . is not in range [-9.223372E18 . . -2.710501E-20, 0.0, 5.421011E-20 . . +9.223372E18]**	A real number was entered outside of its usable range.

Table A-1 Program Warnings and Errors (continued)

Number/Error		Explanation
8097	**Embedded comment detected.**	You can use braces { } or parentheses with asterisks (* *) as delimiters for comment statements. However, you cannot use the same opening delimiter twice in succession. For example, { {This comment is illegal} }. { (*This comment is permitted.*) }.
8098	**Label ... is too long: 12 characters is maximum allowed.**	The maximum number of characters allowed for a safe-state return label is twelve.
8099	**Integer ... is not in range [0..65535].**	The value of the unsigned integer indicated must be within the range shown.
8100	**TO keyword missing following keyword LOCAL.**	Error is caused by defining a safe-state SFC as local but giving the syntax incorrectly. Make sure the keyword TO follows the word LOCAL in the initial step of the safe-state SFC.
8101	**SFC name missing following LOCAL TO command.**	Error is caused by defining a safe-state SFC as local but giving the syntax incorrectly. Make sure an SFC name follows the key words LOCAL TO in the initial step of the safe-state SFC.
8102	**SSABORT is not valid from a local safe-state SFC.**	An attempt was made to execute an SSABORT from a local safe-state SFC. An SSABORT can only be executed from a level safe-state SFC. Use SSRETURN in local safe-state SFCs.
8103	**Comma expected before '...'.**	Commas are required in various places as parameter delimiters. Check the syntax for the indicated statement.
8104	**Integer identifier expected.**	An integer identifier is expected. BITTEST function can only be performed on integer variables.
8105	**SFC name ... is too long: ... characters is maximum allowed.**	SFC names cannot be more than 8 characters.
8106	**Identifier ... must have an address to be used in an SFC.**	Identifiers, for which the controller Address Field was set to NONE, cannot be used in either the parallel section of an SFC step or in SFC transitions.
8107	**Cannot reference direct addresses in SFC for current controller.**	For S5 controllers, direct addresses cannot be referenced in SFC steps or transitions in programs where the controller type is set to CPU928B, CPU948, or CPU948R.
8108	**Real value . . . is not in range [-0.1701412E39 . . -0.14698368E-38, 0.0, +0.14698368E-38 . . 0.1701412E39]**	A real number was entered outside of its usable range.

Table A-1 Program Warnings and Errors (continued)

Number/Error		Explanation
9000	**Illegal/Duplicate declaration.**	A variable has been declared twice. This occurs typically in an SFC step declaration or after copying an external unit into your program. This error also occurs if you use an APT keyword in your declaration.
9001	**Declaration type expected.**	A variable has not been properly declared. Check the syntax for the expression.
9002	**Colon character expected.**	Statement does not end with a colon.
9003	**Error at location ..., step ..., near line ....**	A general problem exists in the indicated SFC step.
9004	**"BEGIN" expected to start math statements.**	In an interlock or math CFB or in the math portion of an SFC step, a BEGIN is expected before any math statements. Be certain that no invalid declarations are in the step. Only valid declaration statements (integer:..., real:..., or boolean:...) are allowed before the BEGIN statement in a math or Interlock CFB or between the MATH and BEGIN in an SFC step.
9007	**End of math language statements expected.**	Typically this occurs when another error occurs in the specified SFC step.
9008	**Invalid initial value for boolean variable.**	TRUE and FALSE are the only valid initial values for boolean variables.
9009	**Invalid initial value for integer variable.**	A value less than -32768 or greater than +32767 appears as the initial value for an integer.
9010	**Invalid initial value for real variable.**	Real number does not fall within the valid range. The valid range for a real number is $-9.223372\ E^{+18}$ to $+9.223372\ E^{+18}$; any value except 0.0 that falls between $-2.710501\ E^{-20}$ and $+5.421011\ E^{-20}$ generates a controller error.
9011	**Comma expected.**	Commas are required in various places as parameter delimiters. Check the syntax for the indicated statement.
9012	**Left parenthesis expected.**	A left parenthesis is expected in the described location.
9013	**Incorrectly specified quoted string.**	A string includes an invalid escape character or does not end with a quote.
9014	**Real (floating point) variable expected.**	A real (i.e., floating point) variable is expected in the identified location.
9015	**Argument cannot be read-only or constant.**	A read-only or constant has been used illegally. Parameters to certain procedures cannot be literal numeric values but must be variables. This error can also occur when you attempt to use a constant or an input on the left side of an assignment statement.
9016	**Quoted string expected.**	A string value delimited by quotes is expected. Check syntax of the statement.
9017	**Array name expected.**	Certain commands require arrays as parameters. Check syntax of the statement.
9018	**Integer or real variable expected.**	An integer or real variable is expected. Check syntax of the statement.
9019	**Invalid argument in PRINT statement.**	Check syntax of the PRINT statement.
9020	**Invalid local variable name.**	Check syntax of the local variable. Check the keyword list and naming conventions.
9021	**Integer constant expected.**	A declared integer constant is required.

Table A-1 Program Warnings and Errors (continued)

Number/Error		Explanation
9022	**Possible nested comment.**	Use braces { } or parentheses with asterisks (* *) as delimiters for comment statements. You cannot use an opening delimiter twice in succession. For example, { {This comment is illegal} }. { (*This comment is permitted.*) }.
9023	**Beginning of poorly formed comment.**	Typically, this message is displayed when a comment is opened but not closed.
9024	**Illegal mixing of types near expression....**	One or more items in the expression are of the wrong type or need to be of the same type.
9026	**Bad argument for function or procedure call.**	A Math function or procedure is incorrectly programmed. Typically, a typographical error exists or an invalid variable type has been used.
9027	**Undefined/illegal function or procedure call.**	A Math function or procedure is incorrectly programmed. Typically, a typographical error exists or an invalid variable type has been used.
9028	**Array not proper size.**	The statement requires the array used to be of a specific size.
9029	**Poorly formed loop construct.**	Typically, this occurs when a WHILE loop is not properly closed with an END LOOP statement.
9030	**Boolean Array name expected.**	An array of a certain type is expected. Check the syntax of the statements indicated.
9031	**Real (float) Array name expected.**	An array of a certain type is expected. Check the syntax of the statements indicated.
9032	**Integer Array name expected.**	An array of a certain type is expected. Check the syntax of the statements indicated.
9033	**Integer or Real (float) Array name expected.**	An array of a certain type is expected. Check the syntax of the statements indicated.
9034	**Integer or Real (float) Address expected.**	The type of a statement address is incorrect. Check the syntax of the statements indicated.
9035	**Real (floating point) value expected.**	The type of value expected is incorrect. Check the syntax of the statements indicated.
9036	**Integer value expected.**	The type of value expected is incorrect. Check the syntax of the statements indicated.
9037	**Parameter type mismatch.**	One or more variables on the line are of the wrong type. Check the syntax of the statements indicated.
9038	**Illegal type mixing.**	One or more variables on the line are of the wrong type. Check the syntax of the statements indicated.
9039	**Component of a structure or recipe is not an assignable location.**	An attempt was made to assign a value to a recipe element that does not have an address.

Table A-1 Program Warnings and Errors (continued)

Number/Error		Explanation
9040	**Invalid initial value for timer variable.**	When a locally-declared timer is initialized, values for all four elements of the timer must be specified, and they must be assigned in the following order: **timer <*type*>: variable,** variable,...:= constant1, constant2, constant3, constant4; The timer type can be **fast** or **slow.** Constant1 is an integer and contains the current value of the timer. Constant2 is an integer and contains the preset value of the timer. Constant3 is a boolean that resets the timer. Constant4 is a boolean that enables the timer.
9041	**Array index range must start at 1, and its length must be >= 1.**	When an array is declared locally, the starting index range value must be 1.
9042	**Timer initial values out of range.**	The initial value for the Timer/Counter Current (TCC) or Timer/Counter Preset (TCP) is out of range. The valid range for these elements is 0–32767.
9043	**Identifier is APT keyword.**	Assign a different name to the identifier.
9044	**Parameter number ... not valid.**	Ensure the function or procedure has the required number of parameters and the parameter types are correct.
9045	**Parameter number ... has an invalid type.**	The parameter passed to the user-defined subroutine is a different type from the parameter declared in the declaration of the subroutine.
9046	**Cannot setup CHR currency.**	Recompile with the force option set.
9047	**Index of array reference is out of range.**	The index of the array reference is out of the size range specified in the array declaration.
9048	**Invalid array declaration.**	The format of the locally declared array is invalid. The correct format is: **array (1 ... n) of <type>: variable,** variable, ... := constant; The array type can be **real, integer, boolean,** or **boolean retentive.** The maximum size of the array **n** can range from 1–32767.
9049	**TRUNC/ROUND are not allowed in a User Defined Subroutine. Use PTRUNC/PROUND.**	The functions TRUNC and ROUND are not allowed in the body of a user-defined subroutine. Use the procedures PTRUNC or PROUND instead.
9050	**BITS_TO_INT is not allowed in a User Defined Subroutine. Use PBITS_TO_INT.**	The procedure BITS_TO_INT is not allowed in the body of a user-defined subroutine. Use the procedure PBITS_TO_INT instead.
9051	**Local declaration timers is not allowed in a User Defined Subroutine.**	Local declaration timers are not allowed in the body of a user-defined subroutine.
9056	**Array index must be an integer.**	An array variable appears without an index.
9057	**ENDIF keyword missing.**	The IF ... THEN construction (possibly containing one or more ELSE or ELSIF statements) does not end with an ENDIF.
9058	**THEN keyword missing.**	THEN does not appear following an IF.
9061	**Unexpected semicolon.**	Ensure the statement is composed correctly.

Table A-1 Program Warnings and Errors (continued)

Number/Error		Explanation
9063	**Endif, elsif or else not contained within if statement at location ..., step ..., near line ... in object ..., near line ...**	The IF/THEN/ELSIF/ELSE construct is incorrect. Refer to Chapter 10 of this manual for the correct syntax.
9066	**Extra comma found in parameter list.**	Ensure the function or procedure has the required number of parameters and the parameter types are correct.
9067	**Too many parameters found in call to subroutine.**	Ensure the function or procedure has the required number of parameters and the parameter types are correct.
9068	**Not enough parameters found in call to subroutine.**	Ensure the function or procedure has the required number of parameters and the parameter types are correct.
9072	**Integer or boolean variable expected.**	Ensure the function or procedure has the required number of parameters and the parameter types are correct.
9073	**Pragma only allowed as the first statement of the math block.**	The PRAGMA statement must be the first statement of a math block. Refer to Chapter 10 of this manual for the correct syntax.
9074	**Undeclared identifier ...**	Ensure the identifier is declared correctly.
9075	**Parameter cannot be CONSTANT type.**	User-defined subroutine parameters must be read/write variables.
9078	**INCREMENT/DECREMENT are not allowed in a user defined subroutine.**	Increment and decrement verbs are not allowed with a user subroutine.
10001	**: line ... structure too large for assignment on target machine.**	Step, transition, or Math CFB includes statement that cannot be compiled. This error also occurs when using an integer array size greater than 32767 or a real array size greater than 16383.
10002	**Blockname.name should not be accessed in parser**	Recompile with the force option set.
10003	**: line... no closing brace... on comment starting near line...**	Closing delimiter for a comment is not found. Comments should be enclosed with one of the following delimiter types: {...} or (*...*).
10006	**Identifier error.**	Recompile with the force option set.
10007	**INCOMPLETE production : direct address.**	Recompile with the force option set.
10008	**: line ... base out of range ....**	Base other than 2 (binary), 10 (decimal) or 16 (hexadecimal) appears in the program.

Table A-1 Program Warnings and Errors (continued)

Number/Error	Explanation
10010 **Digit ... not valid for base.**	Recompile with the force option set.
10014 **Invalid block/subroutine definition.**	Recompile with the force option set.
10015 **Subroutine declaration error.**	Recompile with the force option set.
10016 **User defined types not supported.**	Recompile with the force option set.
10017 **Unary minus not supported.**	Recompile with the force option set.
10018 **Reserved range for C memory locations above ... not checked.**	Phase 4 cannot check if the retentive or non-retentive C-memory locations are reserved correctly in the Compiler Control File. You are responsible.
10020 **Duplicate declaration of block ...**	Recompile with the force option set.
10021 **Duplicate declaration of subroutine ...**	Recompile with the force option set.
10022 **Public declaration hidden by local declaration ...**	Recompile with the force option set.
10023 **Multiply declared identifier ...**	Recompile with the force option set.
10024 **Multiply declared public identifier ...**	Recompile with the force option set.
10025 **Block ... not declared.**	Recompile with the force option set.
10026 **Range not checked on retentive C memory locations.**	Phase 4 cannot check if the retentive C-memory locations are reserved correctly in the Compiler Control File. You are responsible.
10027 **Emit rll compare error.**	Recompile with the force option set.
10028 **Range not checked.**	Recompile with the force option set.
10030 **Unit ... not declared.**	Recompile with the force option set.
10031 **Object of assign.array not accessible.**	Recompile with the force option set.
10032 **Boolean xor not supported.**	Recompile with the force option set.
10033 **Variable.variable not accessible.**	Recompile with the force option set.
10034 **Object of assign.id not accessible.**	Recompile with the force option set.
10042 **Function can not return complex type.**	Recompile with the force option set.
10043 **Records not supported as parameters to subroutines.**	Recompile with the force option set.

Table A-1 Program Warnings and Errors (continued)

Number/Error	Explanation
10044 **Initial value not supported for ... subtype.**	Recompile with the force option set.
10046 **Symbol table interface error.**	Recompile with the force option set.
10047 **: line ... symbol table error : symbol ... name too long**	The maximum number of characters permitted is 38. This message applies to all symbol names. For example, this number includes the SFC name (preceded by $), the step name, the user-declared variable name, and periods separating them.
10048 **Cannot add table ...**	Recompile with the force option set.
10050 **Could not open unit and global tables.**	Recompile with the force option set.
10051 **... ms table not found.**	Recompile with the force option set.
10052 **No block name.**	Recompile with the force option set.
10053 **Must be declaration section.**	Recompile with the force option set.
10054 **: line ... FATAL error out of memory.**	The compiler has run out of memory (i.e., an operating system limitation). This occurs when a step, transition, or Math CFB includes compound statement that cannot be converted into the target control during a compile. If you receive this error, divide the identified statement into several smaller statements without changing the intended logic.
10056 **Internal symbol table error.**	Recompile with the force option set.
10057 **... must be a block name.**	Recompile with the force option set.
10058 **... does not exist.**	Recompile with the force option set.
10060 **: line ... cannot check status of ...**	Variable on the indicated line has not been declared or generated, and, therefore, its status cannot be checked. Check the source or the variable (Declaration Tables, I/O, object extensions, etc.).
10061 **Variable or array extension not accessible.**	Recompile with the force option set.
10062 **Line ... arrays have different dimensions**	Arrays used in the statement on the indicated line must have identical dimensions.
10063 **Line ... index out of range for array ....**	A non-integer value is used to index an array, or the index is outside the declared range of the array.
10064 **Array index not an integer or integer expression.**	Recompile with the force option set.
10066 **Array length must be positive.**	Recompile with the force option set.
10067 **Array index must be one.**	Recompile with the force option set.
10068 **Formal actual parameter mismatch.**	Recompile with the force option set.
10070 **: line ... illegal array index : boolean array indexed by expression**	Boolean arrays can be indexed only by literal values such as 5,7, 9, 16, etc.
10071 **Complex types not supported for arrays.**	Recompile with the force option set.
10072 **Operand type conflict.**	Recompile with the force option set.

Table A-1 Program Warnings and Errors (continued)

Number/Error	Explanation
10073 **Semantic error.**	Recompile with the force option set.
10074 **INCOMPLETE nnl production : records not supported.**	Recompile with the force option set.
10076 **Multi-dimensional arrays are not supported.**	Recompile with the force option set.
10077 **: line ... out of RLL JMP labels.**	Compiler has run out of a controller-specific RLL construct in the 545, 545L, 555, 560/560T, 565/565T/565P, or 575 object. Typically this error indicates that you have overflowed the nesting limit of an IF/THEN/ELSE/ELSIF statement. If you receive this error, divide the identified statement into several smaller statements without changing the logic.
10078 **Object of flag statement not of type flag.**	Recompile with the force option set.
10080 **Line ... object of assignment statement not an assignable location.**	A read-only variable, e.g., a WI-type, a DI-type, is placed on the right side of an assignment statement.
10081 **Line ... operator operand type clash near ....**	A statement contains one or more parameters of the wrong type. For example, in the assignment statement VAR1:= VAR2 + VAR3, this message is displayed when VAR2 is a real number and VAR3 is an integer.
10083 **Incompatible assignment ...**	Recompile with the force option set.
10084 **Invalid order ...**	Recompile with the force option set.
10086 **Arrays with index expressions are not supported as actual parameters.**	Recompile with the force option set.
10087 **Invalid parameter.**	Recompile with the force option set.
10088 **Interface error ms routines sub mod.**	Recompile with the force option set.
10090 **Illegal attempt to assign structure subcomponents.**	An invalid recipe assignment is found. Recipes in the SELECT statement must have the same template.
10091 **Unable to index direct address.**	V-memory is the only memory allowed to be indexed.
10092 **Subscripted variable/direct address not indexable ...**	Only declared arrays or V-memory direct memory addresses are allowed to be indexed.
10093 **Line ... type conflict**	A statement contains one or more parameters of the wrong type. For example, in the assignment statement VAR1:= VAR2 + VAR3, this message is displayed when VAR2 is a real number and VAR3 is an integer.
10094 **Invalid type for not expression.**	Recompile with the force option set.
10096 **Ucoil already exists.**	Recompile with the force option set.
10098 **Undefined identifier ...**	Recompile with the force option set.
10100 **Cannot locate reserved location section of work list.**	Recompile with the force option set.
10101 **Cannot locate NLL section of work list.**	Recompile with the force option set.
10102 **Cannot open file ...**	Recompile with the force option set.

Table A-1 Program Warnings and Errors (continued)

Number/Error	Explanation
10103 **Cannot open assembly.apd file ...**	Recompile with the force option set.
10104 **Internal error : unable to open system file nnl.dat.**	Recompile with the force option set.
10106 **Cannot open file ...**	Recompile with the force option set.
10107 **Cannot open work list.**	Recompile with the force option set.
10108 **Path error ...**	Recompile with the force option set.
10110 **Unexpected end of file encountered in work list.**	Recompile with the force option set.
10111 **Parameter type clash.**	Recompile with the force option set.
10112 **Number of actual/formal parameters not the same.**	Recompile with the force option set.
10114 **Line ... value or converted value of based number ... is out of range.**	The value of the indicated based number cannot be represented on the target controller.
10116 **NNL too many errors, compilation terminated!**	Recompile with the force option set.
10117 **Address not accessible in current context.**	RLL-only statements appear in a step with SFPGM-only statements or direct references to 545, 545L, 555, 565/565T/565P, or 575 Loop, Analog Alarm, SFPGM, or SFSUB variables (%LSP1, etc.).
10118 **Line ... Statement not valid in current context (illegal mixing of RLL and SFPGM operations/code types).**	RLL-only and SFPGM-only operations appear in the same SFC step or a Math CFB. Chapter 11 lists the math functions and procedures with the code type for each. This error also occurs if you use the LEAD_LAG procedure in anything other than a sampled Math CFB. This error can also indicate that you have attempted to use an APT programming option that is not supported by the target controller. For example, SFPGMs are only supported by the 545, 545L, 555, 565/565T/565P, and the 575 controllers, and any APT operation requiring an SFPGM can only be used with these controllers. The "APT Overview" chapter in *SIMATIC APT Programming Reference (Tables) Manual* lists which APT programming options are available on the supported controllers.
10119 **Not supported as comment.**	Unrecognized character appears where a comment is expected. Comments should be enclosed with one of the following delimiters: {...} or (*...*).
10120 **No closing ... on comment starting near line ...**	Closing delimiter for a comment is not found. Comments should be enclosed with one of the following delimiters: {...} or (*...*).
10121 **Line ... illegal array index: boolean array indexed by expression.**	Boolean arrays can be indexed only with literal values (e.g. 1,2,3...). Other array types allow expressions.
10122 **... Error: line ... based number ... out of range for 16 bit representation.**	A boolean value cannot be represented in 16-bit boolean format.
10123 **Real number ... out of range for 32 bit representation.**	Recompile with the force option set. Check your code.

Table A-1 Program Warnings and Errors (continued)

Number/Error		Explanation
10124	**Line ... port number must be 1,2, or 3.**	A PRINT statement has an illegal port value. The number should be 1 (for Port 1 of the 565/565T/565P Loop CPU card or the 545, 545L, 555, or 575 controllers), or 2 (for Port 2 of the 565/565T/565P Loop CPU card), or 3 (for both Ports 1 and 2 of the 565/565T/565P Loop CPU card).
10125	**Function ... not defined/supported.**	Recompile with the force option set.
10126	**Actual parameter mode does not correspond to formal parameter mode.**	Recompile with the force option set.
10127	**No memory allocated for array.**	Recompile with the force option set.
10128	**Work list error.**	Recompile with the force option set.
10130	**Only init section will execute for event math block ...**	An event block executes only once. To repeat the execution of an event block, you must disable and then enable the block again. You might use an event block to generate a report or to total the ingredients at the end of a batch. This type of Math block should not contain an INIT section, because the INIT section is the only one that would execute.
10131	**Target ... ... not directly supported.**	APT only supports controllers available in the Compiler Control Editor.
10132	**Controller Release ... not directly supported.**	APT only supports controllers available in the Compiler Control Editor.
10133	**Programming Unit/TISOFT version ... not directly supported, generating code for TISOFT Rel 4.0.**	APT only supports TISOFT 4.0, or greater, compatible code.
10134	**Programming Unit Release/TISOFT version ... not directly supported, generating code for TISOFT Rel 4.0.**	APT only supports TISOFT 4.0, or greater, compatible code.
10141	**Line ... address not valid....**	The controller address referenced by this statement is not valid. Typically, this occurs when direct controller addresses are used. Check the address types and their valid ranges on the target controller.
10142	**Line ... looping construct not supported on current target.**	Math looping constructs are not valid on all controllers. For example, WHILE statements can only be used with the 545, 545L, 555, 565/565T/565P, the 575 and with S5 controllers.
10144	**Pragma ... being ignored.**	If the PRAGMA statement is not supported on a controller release, it will be ignored.
10145	**Public modifier ignored.**	Recompile with the force option set.
10147	**Line ... invalid direct address name ...**	The controller address name referenced by this statement is not valid. For example, %WW is not a controller address type. Check the address types and their valid ranges on the target controller.
10148	**Line ... port number must be 1 for 545 target**	The 545 and 545L support only one port.
10150	**Unit name... is reserved word.**	The unit name is a reserved word. Rename the unit.
10151	**Possible divide/mod by zero near ...**	The denominator of the expression cannot be zero.

Table A-1 Program Warnings and Errors (continued)

Number/Error	Explanation
10153 **Possible array size constraint violation.**	The array size for certain firmware procedure parameters must not exceed a specified size.
10154 **Undefined subroutine**	Procedure was called that does not exist. Check to make sure all your subroutines exist in APT.
10155 **Initial value must resolve to a constant or a literal**	Bad or invalid value specified as an initial value for a variable.
10156 **... record component not accessible.**	Recompile with the force option set.
10157 **Statement too long.**	Break up expression into smaller statements.
10158 **Function return must have a value**	Value returned by a user-defined subroutine is invalid.
10160 **Array length too large.**	Recompile with the force option set.
10162 **Reserved location(s) too large.**	Reduce the amount of memory reserved in the Compiler Control File.
10163 **Truncating Reserved memory address checking to 32000.**	Reduce the amount of memory reserved in the Compiler Control File.
10164 **Recursive calls to user-defined subroutines are not allowed.**	A user-defined subroutine is attempting to make a call to itself. This can also occur when subroutines are nested. That is, Sub1 calls Sub2, which in turn calls Sub1. APT does not allow recursive subroutine calls.
10165 **Call to library handler returned ERROR.**	Recompile with the force option set.
10167 **Unpredictable results can occur if TRUNC is used inside of a User Defined Subroutine used PTRUNC instead.**	For Series 505 controllers, use **PTRUNC** instead of **TRUNC** in subroutines.
10168 **Unpredictable results can occur if ROUND is used inside of a User Defined Subroutine used PROUND instead.**	For Series 505 controllers, use **PROUND** instead of **ROUND** in subroutines.
10169 **...: Programming structure used will not work with reals in a user defined subroutine. Near expression ...**	For Series 505 controllers, the subroutine has generated an SFPGM PACK instruction. The message lists the line of code in question. When the code is downloaded to the controller, it will not execute correctly. You can rewrite your math code and not use these types of math constructions indicated, or rewrite your math code so that it maps to RLL.
10170 **Lead_Lag should only be called from an analog_alarm, loop, or sampled math block.**	The **LEAD_LAG** procedure requires a CFB with a sampled time such as a sampled MATH CFB, loop, or Analog Alarm.
10171 **Unpredictable results can occur if BITS_TO_INT is used inside of a User Defined Subroutine used PBITS_TO_INT instead.**	For Series 505 controllers, use **PBITS_TO_INT** instead of **BITS_TO_INT** in subroutines.
10172 **User-defined subroutines are being nested too deep.**	User-defined subroutines can be nested to five levels. That is, SUB1 can call SUB2, which can call SUB3, etc. If the limit of five levels is exceeded, an error occurs. You need to reduce the nesting level.
10173 **Edge is not allowed in a user defined subroutine.**	Do not use the **EDGE** procedure in subroutines.
10177 **(...) Invalid type ... for ...**	Recompile with the force option set.

Table A-1 Program Warnings and Errors (continued)

Number/Error	Explanation
10181 **This instruction is not allowed in the controller release specified.**	Refer to Chapter 1 of the *SIMATIC APT Programming Reference (Tables) Manual* for a table of controller-supported features.
10182 **Parameter must have an I/O address type.**	The procedure or function requires a valid I/O address as a parameter.
10183 **Subroutine must be declared with an RLL_CYCLIC pragma.**	A math block declared with pragma RLL_CYCLIC must call subroutines that are also declared pragma RLL_CYCLIC.
10184 **Subroutine is declared with an RLL_CYCLIC pragma. The block must also be declared with an RLL_CYCLIC pragma.**	A subroutine declared with pragma RLL_CYCLIC must be called from a math block that is also declared pragma RLL_CYCLIC.
10185 **Subroutine must be declared with an RLL_INTERRUPT pragma.**	A math block declared with pragma RLL_INTERRUPT must call subroutines that are also declared pragma RLL_INTERRUPT.
10186 **Subroutine is declared with an RLL_INTERRUPT pragma. The block must also be declared with an RLL_INTERRUPT pragma.**	A subroutine declared with pragma RLL_INTERRUPT must be called from a math block that is also declared pragma RLL_INTERRUPT.
10187 **Direct Address type ... not allowed on controller specified.**	The controller specified in the Compiler Control Files does not support the address specified.
10188 **Unable to read opcode table.**	Recompile with the force option set.
10189 **Literal number cannot be accessed by byte addressing, index is assumed to be zero.**	A literal value cannot be accessed by byte addressing. The index will be replaced with a 0.
10190 **Expressions as parameters to subroutines that generate sfpgm code may generate unpredictable behavior in a heavily loaded controller.**	APT creates an internal variable to hold the result of the expression in the parameter. The variable is then passed as a parameter. The math block may be interrupted by a higher priority math block before completing the subroutine call. If the higher priority math block changes variables in the expression, unpredictable behavior may result.
10191 **Recipe of type CONSTANT cannot be on the left hand side of an assignment.**	Assignments to constants are not allowed.
10192 **Parameter must have an address type of X or WX.**	The procedure or function requires a valid X or WX I/O address as a parameter.
10193 **Parameter must have an address type of Y or WY.**	The procedure or function requires a valid Y or WY I/O address as a parameter.
10194 **Flags are reference-only variables in CYCLIC and INTERRUPT math blocks.**	Because APT flags are always evaluated in the normal RLL processing task, APT flags can only appear on the right side of an expression in a RLL_CYCLIC or RLL_INTERRUPT math block.
10195 **The source and destination byte offsets must be either 0 or 1.**	The offset must be the literal value 0 or 1.
10196 **The number of bytes to move must be greater than 0.**	The function or procedure must move at least 1 byte.
10197 **The index on an array element must be a literal number.**	The array element index must be a literal value.
10199 **Procedure call is not allowed within an interrupt organization block.**	You cannot call a user-defined subroutine, or the INTERPOLATE, LOOKUP_TABLE, SCALE, UNSCALE and LEAD_LAG procedures, in a system-activated subroutine.

Table A-1 Program Warnings and Errors (continued)

Number/Error	Explanation
10200 **. . has no image register, operation has no effect.**	If you do not have the image register selected, you cannot do an IREAD or an IWRITE to the I/O point.
10201 **Parameter . . . must be literal.**	For S5 controllers, specified parameter cannot be an expression or variable.
10202 **Parameter is out of range.**	One or more of the parameters has exceeded the allowable range for the instruction.
10203 **Fast timer cannot be locally declared for specified controller type.**	You cannot locally declare a fast timer for S5.
11012 **Reserved memory, controller type, or number of RCC cards have changed resulting in change to translated addresses.**	Tags marked for translation to PCS will be allocated new addresses due to changes since the last time the Build Translate Table option was selected. Changes were made either in the controller type, in the number of RCC cards, or in these reserved memory locations: K-Memory, V-Memory, number of loops, number of analog alarms, number of timer/counters, number of retentive CRs, or number of nonretentive CRs.
11013 **Reserved memory and/or controller type have changed resulting in change to translated address**	Tags marked for translation to PCS will be allocated new addresses due to changes since the last time the Build Translate Table option was selected. Changes were made either in the controller type or in these reserved memory locations: DB, DX, Flag, Timer and Counter.
11015 **A previously translated unit has been marked for compile again resulting in translated addresses to be changed.**	Tags marked for translation to PCS will be allocated new addresses. This occurs when a unit that was at one time included in a compile with the Build Translate Table option was at a later time not included in a compile with the Build Translate Table option, but has now been re-included in the current compile.

Table A-1 Program Warnings and Errors (continued)

Number/Error	Explanation
12000 **Number of RCCs listed in ....**	An invalid number appears for the number of 560/560T 565/565T/565P RCCs. 1, 2, 3, or 4 RCCs can be used; each RCC has two I/O channels.
12045 **Out of retentive CRs. Symbol = ... in ...**	The APT program requires more retentive CRs than your controller supports. If you receive this error for a 560/560T/565/565T/565P controller, be certain that the number of RCCs in your Compiler Control File indicates the correct number of RCCs in your controller. If you receive this error for any controller, you must do one of the following: • Reduce the number of boolean variables (including arrays) that are declared as retentive in your program. • Reduce the amount of retentive CR memory reserved in the Compiler Control File. • For a 560/560T/565/565T/565P, you can increase the number of RCCs in your system (maximum=4).
12046 **Out of nonretentive CRs and unused I/O locations. Symbol = ... in ....**	The APT program requires more non-retentive CRs than your controller supports. If the compiler runs out of non-retentive CRs, it uses Ys that are unused after the last configured I/O point on each channel. If you receive this error for a 560/560T/565/565T/565P controller, be certain that the number of RCCs in your Compiler Control File indicates the correct number of RCCs in your controller. If you receive this error for any controller, you must do one of the following: • Reduce the number of boolean variables (including arrays) that are declared as non-retentive in your program. • Reduce the amount of non-retentive CR memory reserved in the Compiler Control File. • Reduce the value of the last I/O module starting point on each I/O channel (i.e., remove any gaps containing unused I/O addresses). This can be done in the I/O Module Editor and does not require the physical relocation of I/O modules. • For a 560/560T/565/565T/565P, you can increase the number of RCCs in your system (maximum=4).
12058 **Cannot re-allocate address for structure element x.**	In certain instances when you do a force compile, or when an assembly force compile occurs, APT cannot fix the addresses of Boolean elements in recipes that contain Boolean arrays. When this occurs, the assembler generates error 12058 during program compilation. You must then recompile your program with Translate set to **Yes** and re-install your tags to the operator station.

Table A-1 Program Warnings and Errors (continued)

Number/Error	Explanation
12080 **Program requires... words of L-Memory. Max = ...** 12081 **Program requires... S-Memory. Max = ...** 12082 **Program requires... V-Memory. Max = ...** 12083 **Program requires... words of K-Memory. Max = ...** 12084 **Program requires... one shots. Max = ...** 12085 **Program requires... tmr/ctr/dcat boxes. Max = ...** 12086 **Program requires... table move boxes. Max = ...** 12087 **Program requires... shift boxes. Max = ...** 12088 **Program requires... drums. Max = ...** 12090 **Program using ... of invalid memory type ... for current controller type.** 12114 **Program requires... analog alarms. Max = ...** 12115 **Program requires... loops. Max = ...** 12116 **Program requires... RLL subroutines. Max = ...** 12117 **Program requires... special function programs. Max = ...** 12118 **Program requires... special function subroutines. Max = ...** 12119 **Program requirements exceed available controller memory:**	APT compiler cannot allocate the required amount of memory. Either there is an insufficient amount of memory in your controller or you have attempted to use an APT operation not supported by your controller. For example, for Series 505 controllers, SFPGMs are only supported by the 545, 545L, 555, 565/565T/565P, and the 575, and any APT operation requiring an SFPGM can only be used with these controllers. APT attempts to compile code for the controller indicated in the Compiler Control File. Check to make sure this file correctly indicates the controller that you intend to use. The Controller Support section of the "APT Overview" Chapter in the *SIMATIC APT Programming Reference (Tables) Manual* lists which APT programming options are available on the supported controllers. If you have not attempted to use an unsupported option for your controller, be sure that the memory size in Compiler Control File matches the memory size of controller. If this is the only error, deselect Force Compile in the Compiler Control File before recompiling. If you still receive one of these errors, do one of the following: • Reduce the general memory usage of your present program. • Reduce the amount of reserved memory in the Compiler Control File (as possible). • Increase the amount of memory in your controller. For the 545-1101 and the 560/560T/565/565T/565P models, and the S5 928B, you can add additional memory cards. • Do not compile a Debug version. Compiling a Debug version uses more memory.

Table A-1 Program Warnings and Errors (continued)

Number/Error	Explanation
PROFIBUS-DP Errors/Messages	
12180 **Non-numeric string " ... " not within comment.**	Check the indicated file and line for a word outside of a comment. Words can only be within comments.
12181 **Encountered end of file before comment termination.**	Check the indicated file and line for a comment that was not closed by an ending symbol "}" or "*)". This may occur on a line previous to the one reported.
12182 **PROFIBUS-DP slave ... is invalid and must be in range [n_1 . . n_2].**	Check the indicated file and line for a slave number that is not within the required range.
12183 **Channel ... is invalid and must be in range [n_1 . . n_2].**	Check the indicated file and line for a channel number that is not within the required range.
12184 **Base ... is invalid and must be in range [n_1 . . n_2].**	Check the indicated file and line for a base number that is not within the required range.
12185 **Slot ... is invalid and must be in range [n_1 . . n_2].**	Check the indicated file and line for a slot number that is not within the required range.
12186 **PROFIBUS-DP module ... is invalid and must be in range [n_1 . . n_2].**	Check the indicated file and line for a module number that is not within the required range.
12187 **Required module, channel and base missing.**	Check the indicated file and line for a missing module number, channel number, and base number.
12188 **Required channel and base missing.**	Check the indicated file and line for a missing channel number and base number.
12189 **Required base missing.**	Check the indicated file and line for a missing base number.
12190 **... slots have already been specified. Slot ... is extraneous.**	Check the indicated file and line for extra slot numbers that are not needed.
12191 **... slots have already been specified, but only ... slots are available.**	Check the indicated file and line. There are too many slots requested and not enough available.
12192 **PROFIBUS-DP slave ... , module ... has already been used.**	Check the indicated file and line. The slave module specified was previously used in this file.
12194 **Invalid PROFIBUS starting address for ... .**	The indicated slave module was not given a starting address in the module editor. Please verify that this slave and module are associated with a channel, base, and slot and given a valid starting address in the module editor.

Table A-1 Program Warnings and Errors (continued)

Error		Explanation
PROFIBUS-DP Errors/Messages		
12195	**File or path name not found**	**EXPORT.2BF** did not exist in the **/PRR** directory.
	Disk full	Disk containing APT was full.
	Slave ... in APT but not in COM PROFIBUS	The indicated slave was in the PROFIBUS.CFG file, but not in the COM PROFIBUS export file.
	Slave ... in COM PROFIBUS but not in APT	The indicated slave was in the COM PROFIBUS export file, but not in the PROFIBUS.CFG file.
	Slave ..., Module ... in APT but not in COM PROFIBUS	The indicated slave module was in the PROFIBUS.CFG file, but not in the COM PROFIBUS export file.
	Slave ... , Module ... in COM PROFIBUS but not in APT	The indicated slave module was in the COM PROFIBUS export file, but not in the PROFIBUS.CFG file.
	Slave ... , Module ... APT: ,nX nY nWX nWY, COM PB: nX nY nWX nWY	There is a mismatch between PROFIBUS.CFG and the COM PROFIBUS data for this slave module.
12198	**... is not an integer in range [0 . . 32767]**	Check the indicated file and line for an integer that is outside the required range.

Table A-1 Program Warnings and Errors (continued)

Number/Error		Explanation
13000 codes are for Series 505 controllers only.		
13000	**505 ASSEMBLY BLOCK Invalid application ID ..., must be within A to Z for instruction 'DEPENDENCY', parameter 1 near line ... of object $575**	An invalid application ID was entered in the Compiler Control Editor. Correct the ID and recompile.
	505 ASSEMBLY BLOCK Invalid record ... in U memory file ...	The U-memory file specified contains an unknown record. Correct the U-memory file and recompile.
	... address has been assigned multiple times.	A cross-reference can be generated to determine where the loop or analog address was used multiple times. Correct the error and recompile.
	RLL statement is too complex to compile near line ... of object ... in ...	Simplify the indicated RLL statement and recompile the program.
	Statement is too complex to compile near line ... of object ... in ...	Simplify the indicated SF or RLL statement and recompile the program.
	Statement is too complex to execute at run time near line ... of object ... in ...	The statement is too complex for controller to execute at run time. Simplify the indicated statement and recompile the program.
	Reserving U memory (...) is not permitted when a U memory file name (...) is declared.	Make a choice between reserving U-memory or entering a U-memory file name.
	... out of range for instruction "PACK_AALM", parameter ... near line ... of object ...	More than 128 analog alarms have been created or reserved.
	... out of range for instruction "PACK_LOOP", parameter ... near line ... of object ...	More than 64 loops have been created or reserved.
	... out of range for instruction "MWFT", parameter ... near line ... of object ...	A sequence array is occupying a V-memory location greater than V32767. You must put the sequence array in lower V-memory. Follow these tips. • If the sequence array is given a reserved memory address, move the array to a lower V-memory location. • Reduce any reserved V-memory, if used. • Use a force compile to remove holes in memory. • Select YES in Translate Build Option to remove any holes due to fixed marked tag address.

Table A-1 Program Warnings and Errors (continued)

Number/Error	Explanation
14006 **Potential PCS/controller mismatch. Reinstall translator tags to correct.**	This is a warning. It means you have changed your marked tags, but not installed them on PCS.
14008 **Tag...in...cannot be translated. (If I/O point, try marking 'IMAGE REGISTER' field) Memory Type = . . .**	This tag cannot be translated to PCS. PCS can only translate I, Q, F, D, DW, DD, TMR, and CTR. If the tag is an I/O point, then mark the image register field on the I/O Form.
14012 **DO NOT ATTEMPT TO INSTALL TAGS. Tag ... in ... has an address out of range, ..., which cannot be translated to TISTAR Release 1.x or TISTAR 2.x.**	For Series 505 controllers, TISTAR Releases 1.x and 2.x did not support V- and K-memory addresses above 65536. The following are possible solutions: — Reduce the amount of any reserved V- and K-memory. — Make sure any user-assigned addresses are below this value if the object is marked for tag translate. — Recompile with a YES Translate Build to move all translated tags to lower memory.
14025 **Secondary name was not given.**	No secondary name was given in the Compiler Control File. Enter a name and recompile.
14028 **PCS does not support range span of 0 for ...**	Change the ranges for the given object so that the range span is not 0.
14031 **Possible overflow of RBE memory. RBE Memory Reserved = ... Kbytes. Estimated RBE Memory Usage = ... Kbytes.**	An estimate of RBE memory usage has been calculated and is greater than the amount reserved. Refer to the chapter on "APT and the PCS Operator Interface" in the *SIMATIC APT Applications Manual* and check the usage again.
14032 **RBE memory ... Kbytes is out of range. Range = 0 to ....**	Reduce the amount of RBE memory reserved in the Compiler Control File.
14047 **The OUT attribute for the AO tag ... had an initial value of ... which is not within the high/low range of ... The initial value being translated to PCS was adjusted to ...**	The OUT attribute for an AO tag had an invalid initial value. If you do not want the initial value assigned by APT, you must unmark the tag and either change the high and low ranges or else change the initial value. You can then remark the tag and recompile.

Table A-1 Program Warnings and Errors (continued)

Number/Error	Explanation
15000 codes are for S5 controllers only	
15007 **S5 ASSEMBLY BLOCK Syntax error near token "A(" near line ... of object ... in ...** **S5 ASSEMBLY BLOCK Syntax error near token "AN(" near line ... of object ... in ...** **S5 ASSEMBLY BLOCK Syntax error near token "O(" near line ... of object ... in ...** **S5 ASSEMBLY BLOCK Syntax error near token "ON(" near line ... of object ... in ...**	Either a subroutine is named A or the assembly code is incorrect. Either a subroutine is named AN or the assembly code is incorrect. Either a subroutine is named O or the assembly code is incorrect. Either a subroutine is named ON or the assembly code is incorrect. If you receive any of these errors, do one of the following: • If a subroutine exists with the name A, AN, O, or ON, rename the subroutine and recompile. • If in-line assembly code is being used, check for the correct syntax and recompile. • Use the Extract utility as described in the DOS Utilities appendix section of the *SIMATIC APT User Manual* to determine if the generated code is correct for the given object.
16025 **Direct address % . . . is not in the reserved memory range near line . . . of object . . . in . . .**	The direct address that is called out is not in the reserved memory in the Compiler Control File. Check the items that are reserved in the Compiler Control File.
16035 **S5 ASSEMBLY BLOCK . . . block is out of range. Range = . . . to . . . near line . . . of object . . . in . . .**	DRs or DXs are out of range. The range is 0 to 255.
16038 **FB/FX . . . is too large (. . . words) for psect . . . near line . . . of object . . . in . . .**	The FB or FX is too large to fit in the given object. Reduce your code for that given object.
16045 **Statement is too complex to execute at run time, near line ... of object ... in ...**	The statement is too complex for the controller to execute at run time. Simplify the indicated statement and recompile the program.
16060 **Symbol . . . not retrieved in . . .**	Make sure your symbol is declared.
16071 **Invalid index . . . for . . . . Range is . . . to . . .** **Invalid reserved memory . . . for . . .**	The index given is out of range for that type of memory. Keep your indexes within the defined range.
16075 **Out of . . . for object . . .**	The type of memory has been exceeded. In order to get more memory, you can reduce reserved memory for the given memory type, perform a force compile to compress any deleted memory, or reduce your code for the given object.
16078 **No more memory locations exist for. . . in . . .**	Reduce any reserved memory in which the given declaration could be allocated. Perform a force compile to compress any holes that might exist.

A.4 Translate/Download/Debug Error Messages

Overview

You may receive one of the following error messages during translation or program download. If you cannot correct the problem by following the suggestions in Table A-2, or if you receive a message not on this list, call (800) 333–7421 in the U.S.A., or 49–911–895–7000 outside the U.S.A., for assistance.

Table A-2 Translate, Download, and Debug Errors

Translate Errors		
Message	**Cause**	**Corrective Action**
Translation failed with tag validation.	The tag data conflicts with the PCS configuration.	Examine the *TRANSLAT.RPT* file in the **\APT\PROGRAM*program_name*\PRR** subdirectory for error messages.
PCS communications error.	Communications between the remote computer and the PCS system are not available or are broken.	Check cabling between the remote computer and PCS system; check the communications card installation; be certain that you followed the setup and translate procedures described in this manual.
PCS does not respond.	Communications between the remote computer and the PCS system are not available or are broken.	Check cabling between the remote computer and PCS system; check the communications card installation; be certain that you followed the setup and translate procedures described in this manual.
PCS APT server does not respond.	APT cannot communicate with the APT server task running on the PCS node.	Be certain that you followed the setup and translate procedures described in this manual.
Insufficient PCS user privilege.	The user does not have the PCS privilege to translate tags.	Use User ID with appropriate security privilege.
PCS secondary name not found.	The secondary name does not exist in PCS.	Use correct secondary name, or change PCS configuration.
Cannot open file on OSU.	The remote computer cannot access the file *INSTALL.TAG* or cannot write to the file *TRANSLAT.RPT*.	Be certain that the subdirectory **\APT\PROGRAM*program_name*\PRR** exists on the remote computer and that the file *INSTALL.TAG* is located in the subdirectory **\APT\PROGRAM*program_name***. Be certain that the file *TRANSLAT.RPT* is not write-protected.
Cannot open file on PCS.	Cannot open a file in the **usr/tistar/data/tmp** directory on the PCS system.	This subdirectory is created when you install PCS software. If this subdirectory has been removed, you can create it again. Be certain that it contains no write-protected files.
Cannot translate. PCS must be in offline state.	PCS is not in the **Offline** state.	Set PCS to the **Offline** state.
OSU disk access error.	Unable to read the hard disk on the remote computer.	Check the hard disk on the remote computer for errors, or to see if it is full.
Fatal Error—APT file has incorrect or missing information.	TISTAR Model 20 TRANS does not know to interpret some information in the INSTALL.TAG file.	Check for UNIT tags, TEXT tags, or other tag types not supported in TISTAR Rel. 1.3.x.

Table A-2 Translate, Download, and Debug Errors (continued)

Translate Errors		
Message	**Cause**	**Corrective Action**
PCS error occurred installing tags.	An error occurred during the install phase of tag translation.	Check the file *TRANSLAT.RPT* for messages.
PCS tag configuration currently active.	The tag configurator or tag translator is already running.	Be certain that the tag configurator software is not running on the PCS system, i.e., no one is configuring PCS tags.

Download Errors		
Message	**Cause**	**Corrective Action**
Taskcode error detected.	The controller returned the error that follows.	Refer to your controller user manual for details.
NIM error detected.	The NIM returned the error that follows.	Refer to your NIM user manual for details.
PCS communications error.	Communications between the APT node and the PCS node are not available or are broken.	Check cabling between the APT node and PCS node; check the communications card installation; be certain that you followed the setup and translate procedures described in this manual.
PCS is not responding.	Communications between the APT node and the PCS node are not available or are broken.	Check cabling between the APT node and PCS node; check the communications card installation; be certain that you followed the setup and translate procedures described in this manual.
Unable to initiate PCS APT server.	APT cannot communicate with the APT server task running on the PCS node.	Be certain that you followed the setup and translate procedures described in this manual.
TISTAR-TIWAY communications error.	Communications between the DEU and the NIM are not available or are broken.	Check TIWAY cabling.
PC-controller communications error.	Direct link communications between the remote computer and the controller are not available or are broken.	Check cabling. Be certain that the baud rate and serial communications port (specified when you set up APT) are correct.
PCS must be in operate state.	PCS is not in the Operate state.	Set PCS to the Operate state.
Cannot set baud rate.	The specified port is invalid for communications.	Be certain that the baud rate and serial communications port (specified when you set up APT) are correct.
User is not privileged.	The user does not have the PCS privilege to download a program to the controller.	Use a User ID with appropriate security privilege.
Secondary name not found.	The secondary name does not exist in the PCS database.	Use the correct secondary name, or change PCS configuration.

Table A-2 Translate, Download, and Debug Errors (continued)

Download Errors		
Message	**Cause**	**Corrective Action**
Secondary is not connected.	The secondary is not physically connected to the PCS system.	Connect the secondary to the PCS system.
Secondary is of wrong type.	The APT program does not match the connected secondary.	Recompile APT for the correct secondary, or connect the correct secondary to the PCS system.
Last compile did not complete.	The remote computer cannot access the file *RETURN* or compile did not complete.	Be certain that the subdirectory **\APT\PROGRAM*program_name*\MAKE** exists on the remote computer and that it contains the file *RETURN*. Also check that you have a successfully compiled program.
Cannot access APT file.	The remote computer hard disk may be bad, or the download files are corrupted.	Use the DOS command **CHKDSK** to check the hard disk and files. Be certain that the subdirectory **\APT\PROGRAM*program_name*\MAKE** exists on the remote computer and that it contains the file *RETURN*. Be certain that the subdirectory **\APT\PROGRAM*program_name*** exists on the remote computer and that it contains the *OBJECT.DWN* and *VERSION.DWN* files for each unit that is compiled.
Current download aborted.	The user aborted the download.	Note that the controller may contain an invalid program when you abort the download.
Program not found.	The program name that you specified for downloading was not found.	Use the correct program name.
Download file not found.	The file *OBJECT.DWN* is missing.	Be certain that the file *OBJECT.DWN* exists in the subdirectory **\APT\PROGRAM*program_name***.
No loop card on controller.	The controller does not process loop, analog alarm, or Special Function Program information.	Install a 545, 545L, 555, 565/565T/565P, or a 575; or compile for another type of controller.
Program too large.	The controller memory cannot hold the program.	Reduce the size of the program, or increase controller memory.
Not enough RCCs on controller.	More RCCs were specified in the APT compile file than are actually installed (560/560T/ 565/565T/565P only).	Specify fewer RCCs in the APT compile file, or install additional RCCs.
APT locked from writing to controller.	Certain download functions have been limited.	See the last page of the Release Notes, “Remove Debug Functionality.”

Debug Errors		
Message	**Cause**	**Corrective Action**
Debug out of variable storage space.	Too many incremental compiles.	Force compile.

A.5 Archive/Restore Error Messages

Overview

You may receive one of the following error messages while archiving or restoring a program. If you cannot correct the problem by following the suggestions in Table A-3, and need assistance, call (800) 333-7421 in the U.S.A., or 49-911-895-7000 outside the U.S.A.

The Archive and Restore processes depend on the environment variables APT_COMMON, APT_DB, and APT_LOCAL. If these variables are not set, or if they are set but do not specify a disk drive, then APT, APTARCH, and APTREST must be executed from the drive that contains the APT information provided in the standard installation directories **\APT\TEXT**, **\APT\DATABASE**, and **\APT\PROGRAM**.

Table A-3 Archive and Restore Errors

Archive Error	Restore Error	Error Message	Explanation/Comments
	✓	**Archive record sequence error**	The data records in the specified archive file are out of sequence. The archive files are either out of order or corrupted. For archives on diskette, be sure that the disks are inserted in the proper order.
✓	✓	**Cancelled by user**	The Archive or Restore process was aborted by the user.
	✓	**Cannot delete directory**	An attempt to delete the program information from the APT program directory has failed. Be sure that no files in the program directory have the read-only attribute set. Verify that APT is installed correctly and that all appropriate environment variables have been set. Refer to the "Overview" information that precedes this table.
	✓	**Cannot make program directory**	An attempt to create a program directory has failed. Verify that APT is properly installed and that all APT environment variables being used, especially APT_LOCAL, are properly set. Refer to the "Overview" information that precedes this table. Be sure that the disk that holds the APT program directory is not full. See the "Disk is full" error description. There may be insufficient memory for this operation to be completed properly. See "Insufficient memory" error description.
✓	✓	**Cannot open/read *ARCCRC.DAT* file**	The specified file cannot be found. Verify that APT is properly installed and that all APT environment variables being used, especially APT_COMMON, are properly set. Refer to the "Overview" information that precedes this table.
	✓	**Cannot open/read archive file**	An attempt to open or read from the <***program_name***>.<***number***> file has failed. The specified archive file does not exist, is damaged, or is not a valid program archive file. Be sure that the path and file (program) name are valid. For archives on diskette, be sure to insert the diskettes in the proper order.

Table A-3 Archive and Restore Errors (continued)

Archive Error	Restore Error	Error Message	Explanation/Comments
✓		**Cannot open/read *ARCLIST.DAT* file**	The specified file cannot be found. Verify that APT is properly installed and that all APT environment variables being used, especially APT_COMMON, are properly set. Refer to the "Overview" information that precedes this table.
	✓	**Cannot open/read next archive file**	An attempt to open or read from the next ***<program_name>.<number>*** file has failed. The specified archive file does not exist, is damaged, or is not a valid program archive file. Be sure that the path and file (program) name are valid. For archives on diskette, be sure that the disks are inserted in the proper order.
✓		**Cannot open/write archive file**	An attempt to open or to write to the ***<program_name>.<number>*** file has failed. If the specified file was created by a previous archive, be sure that the file's read-only attribute is not set. If the target is a diskette, be sure that it is not write-protected. Be sure that the target disk is not full. See the "Disk is full" error description.
	✓	**Cannot open/write program file**	An attempt to open (create) or write to a program file has failed. Verify that APT is properly installed and that all APT environment variables being used, especially APT_LOCAL, are properly set. Refer to the "Overview" information that precedes this table. Be sure that the disk that holds the APT program directory is not full. See the "Disk is full" error description. There may be insufficient memory for this operation to be completed properly. See "Insufficient memory" error description.
✓		**Cannot write record count to archive file**	An attempt to write this archive's total record count to the ***<program_name>.1*** file has failed. If target is a diskette, be sure that it is not write-protected. Be sure that the target disk is not full. See "Disk is full" error description.
	✓	**CRC error during file expansion**	A data record in the specified archive file contains an invalid CRC value. The archive file is either incomplete or corrupted; successful restoration is unlikely.
	✓	**Database conversion error**	An error occurred during conversion of the restored program to the current database format. If restored from the APT hierarchy, examine the *TMSTDOUT.LOG* file for more information.

Table A-3 Archive and Restore Errors (continued)

Archive Error	Restore Error	Error Message	Explanation/Comments
	✓	**Database duplicate key**	An attempt to insert a data record with a non-unique key into the database has failed. Either the archive or the database is corrupted. Verify that APT is properly installed and that all APT environment variables being used, especially APT_DB, are properly set. Refer to the "Overview" information that precedes this table. Archive all programs in the database, reinitialize the database with the **NEWDB** utility, then try to restore the program again.
	✓	**Database record not legal member of set**	An attempt to insert a data record into a database set has failed. Either the archive or the database is corrupted. Verify that APT is properly installed and that all APT environment variables being used, especially APT_DB, are properly set. Refer to the "Overview" information that precedes this table. Archive all programs in the database, reinitialize the database with the **NEWDB** utility, then try to restore the program again.
✓	✓	**Disk is full**	Archiving: the disk drive in the specified archive path is full. Delete all unnecessary files on the disk then restart the archive, *or* restart the archive and specify a disk with adequate free space *or* restart the archive and specify a diskette drive. Restoring: the disk drive containing the APT database or the program directory is full. Delete all unnecessary files on the full disk, then restore the program, *or* archive all programs in the database, reinitialize the database with the **NEWDB** utility, and then restore the program. For more information: for an archive or restore from the APT hierarchy, see the *TMSTDOUT.LOG* file; for the APTARCH and APTREST utilities, repeat the process with the -t2 switch set.
✓	✓	**DOS or unknown error**	An unanticipated error was encountered. Execute the APTARCH or APTREST utilities with the -t2 switch to get more information. (Only applicable to the **APTARCH** and **APTREST** utilities.)
	✓	**File and program name mismatch**	The file name and APT program name stored in the file are different. Verify that the specified program name is correct. If the archive file name has been changed, change it back to the original name using the <***program_name***>.<***number***> format.

Table A-3 Archive and Restore Errors (continued)

Archive Error	Restore Error	Error Message	Explanation/Comments
	✓	**File not found**	The specified archive file does not exist. Be sure that the path and program name are valid.
✓	✓	**General Archive failure** *or* **General Restore failure**	An unanticipated error was encountered. For more information: for an archive or restore from the APT hierarchy, see the *TMSTDOUT.LOG* file; for the APTARCH and APTREST utilities, repeat the process with the -t2 switch set.
✓	✓	**Insufficient memory**	Insufficient memory to execute the process. Execute the DOS command **MEM** or **CHKDSK** to check available memory. Unload all unnecessary TSRs, programs, drivers, etc. and try again.
	✓	**Invalid command in archive file**	The specified archive file contains an invalid archive command. The archive file is either incomplete or corrupted; successful restoration is unlikely.
	✓	**Invalid data in archive file**	The specified archive file contains invalid data. The archive file is either incomplete or corrupted; successful restoration is unlikely.
✓		**Invalid database**	An APT database file cannot be found or the database is corrupted. Verify that APT is properly installed and that all APT environment variables being used, especially APT_DB, are properly set. Refer to the "Overview" information that precedes this table. If all else fails, reinitialize the database with the **NEWDB** utility, then restore your program archive files from either the APT hierarchy, or using the APTREST utility.
✓		**Invalid database record**	An APT database record could not be found. The database record may be corrupted. Verify that APT is properly installed and that all appropriate APT environment variables, especially APT_DB, are properly set. Refer to the "Overview" information that precedes this table. If all else fails, reinitialize the database with the **NEWDB** utility, then restore your program archive files from either the APT hierarchy, or using the APTREST utility.
✓	✓	**Invalid disk drive**	An invalid disk drive name was specified for the APT database or program directory. Verify that APT is properly installed and that all appropriate APT environment variables, especially APT_DB, are properly set. Refer to the "Overview" information that precedes this table. For more information: for an archive or restore from the APT hierarchy, see the *TMSTDOUT.LOG* file; for the APTARCH and APTREST utilities, repeat the process with the -t2 switch set.

Table A-3 Archive and Restore Errors (continued)

Archive Error	Restore Error	Error Message	Explanation/Comments
	✓	**Old Archive ... cannot be converted to current database**	An APT program archived with APT Rel. 1.3 or earlier cannot be restored to the current APT release. You must first restore the program to either APT Rel. 1.3a or 1.4a and then re-archive the program. The Rel. 1.3a or 1.4a archive may then be restored to the current APT release.
	✓	**Package conversion error**	An error occurred during package conversion of an archive record. For more information for a restore error from the archive dialog box, see the *TM.STDOUT.LOG* in the temporary files directory. For more information for a restore error from the **APTREST** executable, repeat the restore process with the -t2 or -t3 switch set.
✓		**Program does not exist**	The specified APT program does not exist. Be sure that the program name is valid. If you use more than one APT database, be sure that the current database is the correct one. Verify that all appropriate APT environment variables, especially APT_DB, are properly set. Refer to the "Overview" information that precedes this table.
✓	✓	**Unable to locate database file**	An APT database file cannot be found or the database is corrupted. Verify that APT is correctly installed and that all appropriate APT environment variables, especially APT_DB, are properly set. Refer to the "Overview" information that precedes this table. If all else fails, reinitialize the database with the **NEWDB** utility, then restore your program archive files from the APT hierarchy, or using the APTREST utility.

A.6 DOS Operating System Error Messages

The DOS operating system may generate an error code when you validate or compile an APT program. A DOS error code is listed in a compile or validate report as the term **Errno = nn**. The **nn** is defined in Table A-4. Refer to the documentation for your DOS operating system for corrective measures. If you cannot correct the problem, or if you receive a message not on this list, call (800) 333-7421 in the U.S.A., or 49-911-895-7000 outside the U.S.A., for assistance.

The two error codes most commonly reported are #02 and #24.

- **Errno = 02** usually means that a file or subdirectory has been deleted.
- **Errno = 24** usually means that insufficient files have been specified in the *CONFIG.SYS* file.

Table A-4 DOS Error Messages

Code	Meaning	Code	Meaning
-1	Operating system error	18	Cross-device link
01	User is not owner	19	No such device
02	No such file or directory	20	Is not a directory
03	No such process	21	Is a directory
04	Interrupted system call	22	Invalid argument
05	I/O error	23	No more files (system)
06	No such device or address	24	No more files (process)
07	Arg list is too long	25	Not a terminal
08	Exec format error	26	Text file is busy
09	Bad file number	27	File is too large
10	No child process	28	No space left
11	No more processes allowed	29	Seek issued to pipe
12	No memory available	30	Read-only file system
13	Access denied	31	Too many links
14	Bad address	32	Broken pipe
15	Bulk device required	33	Math function argument error
16	Resource is busy	34	Math function result is out of range
17	File already exists		

Appendix B

Direct Memory Addressing

B.1 Types of Direct Memory Addressing

Overview

APT allows you to access memory addresses directly by using a percent sign (%) prefix. The types of direct memory addressing available depend on the kind of controller you have.

Status Word memory Allows you to read status words that contain operational information about your Series 505 controller.

System Data Area (RS) memory Allows you to read the internal system parameters that contain operational information about your S5 controller.

Temporary (SFPGM) memory variables Allow you to use memory space only while a math block is executing.

Reserved memory Allows you to use memory locations that you reserve before compiling the program.

Availability

Status word memory and temporary (SFPGM) memory variables are supported for Series 505 controllers only. System Data Area (RS) memory is supported for S5 controllers only. Reserved memory is supported for both Series 505 and S5 controllers.

Status Words (Series 505 only)

If you have a Series 505 controller, APT allows you to read Status Words by using the %STW address as a variable. For example, %STW1 allows you to directly access the information in Status Word 1. For information about the meaning of the status words, see the manual that comes with your Series 505 controller.

System Data Area (RS) (S5 only)

If you have an S5 controller, APT allows you to read RS words by using the %RSW address as a variable. For example, %RSW14 on a 928B allows you to directly access the information in RS14, which is the base address of the flag area. For a 948U/R, %RS9.0 allows you to directly access the bit information in RS9.0, which is the QVZ test bit. For more information about the RS memory area, see the manual that comes with your S5 controller.

Temporary Variables (Series 505 only)

If you have a Series 505 controller, APT allows you to access 16 controller memory locations that are available for the temporary storage of values. These temporary variables are available only within an SFPGM math block, which is supported by the 545, 555, 565, 565T, 565P, and 575 controllers. The values remain valid only while the math block is executing.

- When you assign an integer value to temporary memory, the value occupies the indicated location: %T13 := 5 assigns the value of 5 to memory location T13 (Figure B-1).

NOTE: Assign values only to T-Memory locations T11–T16. APT and the controller use locations T1–T10.

- When you assign a real value to temporary memory, place a period at the end of the variable name. A real value occupies the indicated location and the next consecutive memory location: %T14. := 6.0 assigns the value of 6.0 to memory locations T14 and T15 as illustrated in Figure B-1.

NOTE: Do not assign a real value to %T16 because only one memory location is available at that address.

%T1	*	%T5	*	%T9	*	%T13	5
%T2	*	%T6	*	%T10	*	%T14	6.0
%T3	*	%T7	*	%T11		%T15	
%T4	*	%T8	*	%T12		%T16	

%T13 accesses integer value stored in memory location **T13**

%T14. accesses real value stored in memory locations **T14** and **T15**

* Reserved for APT and controller functions.

Figure B-1 Using Temporary Memory (Series 505 only)

Reserved Memory

For both controller families, APT permits you to reserve controller memory. See Section B.2 for a discussion of reserved memory and direct addressing for Series 505 controllers. See Section B.3 for a discussion of reserved memory and direct addressing for S5 controllers.

B.2 Using Direct Addresses (Series 505 Controllers)

Reserved Memory for Series 505

When you set up the Compiler Control File, APT allows you to reserve memory locations for your use. Figure B-2 shows the portion of the compile control screen for a 565/565T where you specify these reserved memory locations.

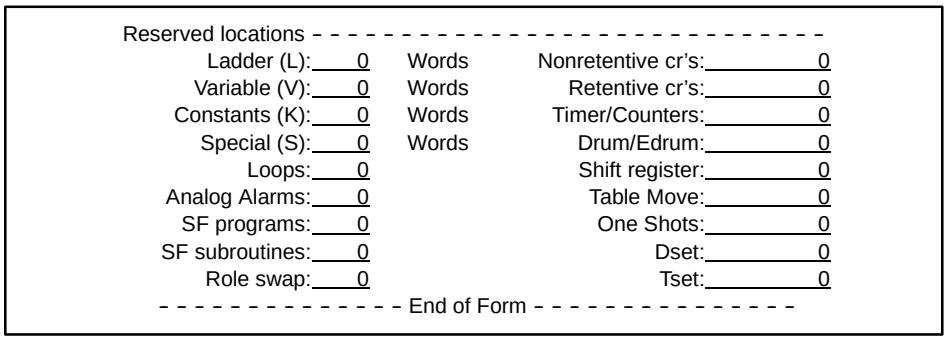


Figure B-2 Control File Reserved Memory (Series 505)

NOTE: Most APT applications do not use the **Reserve Memory** option, but rather let APT determine the memory allocation. In the event that you have an existing ladder logic program, your ladder logic program can coexist with APT. You must reserve all memory associated with that program in order for your program to run properly. You can edit your ladder logic program in TISOFT before you download APT's program.

Ladder (L) Memory is memory used for ladder logic code (RLL). Reserve the number of L words you need to store your own ladder logic program which you will use in conjunction with APT. Refer to the *SIMATIC 505 Programming Reference Manual* for a detailed list of the number of L words required by each instruction.

Variable (V) Memory is memory used for storing read/write integer and real values. Integer values require one memory location and real values require two memory locations. If you are reserving memory to store arrays you must reserve enough memory for the entire array. APT accesses reserved V locations by %V1, %V2, etc., for integers and %V1., %V3., etc., for reals.

Constant (K) Memory is memory used for storing read only integer and real values. Integer values require one memory location and real values require two memory locations. If you are reserving memory to store arrays you must reserve enough memory for the entire array. APT accesses reserved K locations by %K1, %K2, etc., for integers and %K1., %K3., etc., for reals.

Special (S) Memory is memory used for storing loops, analog alarms and special function programs. Enter the number of S-memory words you need to reserve. You can use TISOFT to see how much S-memory your loops, analog alarms, or special function programs require.

Loops enter the number of loops you want to reserve. In APT, access these reserved loops on the CFB loop definition form by %loop1, %loop2, etc.

Analog Alarms enter the number of analog alarms you want to reserve. In APT, access these reserved analog alarms on the CFB analog alarm form by %aalm1, %aalm2, etc.

SF Programs is memory used for Special Function programs (SFPGM). Enter the number of SF programs you want to reserve. They will be called SFPGM1, SFPGM2, etc. and must be created and called by a ladder logic program in TISOFT.

SF Subroutines subroutines that are SFPGM only. Enter the number of SF subroutines you want to reserve. They will be called SFSUB1, SFSUB2, etc. and must be created and called by a ladder logic program in TISOFT.

Role Swap The 565 controller has the redundant controller option where two controllers are running and one controller is a hot back-up of the other controller. Enter the number of Force Role Swaps you need to reserve. They will be called FRS1, FRS2, etc. and the role swap will have to be programmed in ladder logic in TISOFT.

Non-retentive CRs is memory used to store internal digital information. Non-retentive CRs lose their value in the event of a power failure. Enter the number of non-retentive CRs you need to reserve. If you are using arrays, you must reserve enough memory for the whole array. APT accesses these reserved CRs by %C1, %C2, etc. See the *SIMATIC APT Programming Reference (Tables) Manual,* Boolean Array Declaration.

NOTE: When you reserve retentive and non-retentive CRs, you are not reserving actual addresses; you are only reserving an amount of space. For instance, if you specify 780 non-retentive CRs, you get %C1–%C768 and %C1025–%C1036. If you specify 10 retentive CRs, you get %C769–%C778.

Retentive CRs is memory used to store internal digital information. Retentive CRs keep their value in the event of a power failure. Enter the number of retentive CRs you need to reserve. If you are using arrays, you must reserve enough memory for the whole array. APT accesses these reserved CRs by %C769, %C770, etc. See the Boolean Array Declaration section in the *SIMATIC APT Programming Reference (Tables) Manual*.

Timers/Counters is memory used for storing the current and preset values for timers and counters. Enter the number of timers and counters you need to reserve. In APT, access these timers and counters on the timer and counter declaration forms by entering %TC1, %TC2, etc.

Drum/Edrum is memory that stores the Drum Step preset and current and the Drum Count preset and current. Enter the number of drums you need to reserve. Drums are not used by APT; they must be created and called by a ladder logic program in TISOFT.

Shift Register is a memory type that consists of one byte per shift register. Enter the number of bytes you wish to reserve. The shift registers must be created and called by a ladder logic program in TISOFT.

Table Move is a block of memory within the controller reserved for the operation of table moves. Enter the number of words you would like to reserve to maintain the current count for each table move. The MWTT and MWFT will have to be created and called by a ladder logic program in TISOFT.

One Shots is a memory type that consists of one byte per configured one shot instruction. Enter the number of bytes you would like to reserve to monitor one shots. The one shots will have to be programmed in a ladder logic program in TISOFT.

Dset Enter the number of date set box instructions you want to reserve. They must be programmed and called by a ladder logic program in TISOFT.

Tset Enter the number of time set box instructions you want to reserve. They must be programmed and called by a ladder logic program in TISOFT.

Other Reserved Locations

Two other reserved locations exist for the 545, 555, and 575. They are Report by Exception and User Memory.

Report by Exception (RBE) (Only available for controller Release 3.0 and higher and OSx/PCS Release 3.x and higher.) To use Report by Exception with PCS or OSx tag translate, memory must be allocated. To calculate the amount of memory needed, refer to the *SIMATIC APT Applications Manual* in the chapter on "APT and the OSx Operator Interface." The default allocation of 20K of memory is the minimum allocation. Any entry less than 20K results in a 20K allocation. If you require more than 20K, you must enter all of the memory required for RBE.

User Memory Externally developed subroutines written in C, PASCAL or assembly languages, can be used by APT, if you reserve user memory for them. There are two ways to allocate user memory. Either reserve the number of U-memory words you need, or else enter the U-memory filename. If you are using a U-memory filename, APT determines the amount of U-memory needed. Only use one method. The U-memory file contains the externally-developed subroutines, and it must be located in the **APT\PROGRAM\<*your program*>\PRR** subdirectory. See the *SIMATIC 505 Programming Reference Manual* for more details about this type of memory.

Table B-1 lists the direct addresses that are supported by APT for Series 505 controllers.

Table B-1 Direct Addresses Supported by APT for Series 505 Controllers

Address	Description	Debug	MAITT	APT Objects	Math Language
%AACK	Analog alarm acknowledge	Yes	Yes		Yes
%AADB	Analog alarm deadband	Yes	Yes		Yes
%AALM	Analog alarm			Yes	
%ACFH	Analog alarm C-flags high	Yes	Yes		Yes
%ACFL	Analog alarm C-flags low	Yes	Yes		Yes
%AERR	Analog alarm error	Yes	Yes		Yes
%AHA	Analog alarm high alarm	Yes	Yes		Yes
%AHHA	Analog alarm high-high alarm	Yes	Yes		Yes
%ALA	Analog alarm low alarm	Yes	Yes		Yes
%ALLA	Analog alarm low-low alarm	Yes	Yes		Yes
%AODA	Analog alarm orange deviation alarm	Yes	Yes		Yes
%APET	Analog alarm process execution time	Yes	Yes		Yes
%APV	Analog alarm process variable	Yes	Yes		Yes
%APVH	Analog alarm process variable high	Yes	Yes		Yes
%APVL	Analog alarm process variable low	Yes	Yes		Yes
%ARCA	Analog alarm rate-of-change alarm	Yes	Yes		Yes
%ASP	Analog alarm setpoint	Yes	Yes		Yes
%ASPH	Analog alarm setpoint high	Yes	Yes		Yes
%ASPL	Analog alarm setpoint low	Yes	Yes		Yes
%ATS	Analog alarm sample time	Yes	Yes		Yes
%AVF	Analog alarm V-flags	Yes	Yes		Yes
%AYDA	Analog alarm yellow deviation alarm	Yes	Yes		Yes
%C	Control relay	Yes	Yes	Yes	Yes
%DCP	Drum current preset	Yes	Yes		Yes
%DSC	Drum step current	Yes	Yes		Yes
%DSP	Drum step preset	Yes	Yes		Yes
%G	Global memory *	Yes	Yes	Yes	Yes
%Gx	Global application memory *	Yes	Yes	Yes	Yes
%K	Constant *	Yes	Yes	Yes	Yes
%LACK	Loop acknowledge	Yes	Yes		Yes
%LADB	Loop deadband	Yes	Yes		Yes
%LCFH	Loop C-flags high	Yes	Yes		Yes
%LCFL	Loop C-flags low	Yes	Yes		Yes
%LERR	Loop error	Yes	Yes		Yes
%LHA	Loop high alarm	Yes	Yes		Yes

Table B-1 Direct Addresses Supported by APT for Series 505 Controllers (continued)

Address	Description	Debug	MAITT	APT Objects	Math Language
%LHHA	Loop high-high alarm	Yes	Yes		Yes
%LKC	Loop proportional gain	Yes	Yes		Yes
%LKD	Loop derivative gain limiting coefficient	Yes	Yes		Yes
%LLA	Loop low alarm	Yes	Yes		Yes
%LLLA	Loop low-low alarm	Yes	Yes		Yes
%LMN	Loop output	Yes	Yes		Yes
%LMX	Loop bias	Yes	Yes		Yes
%LODA	Loop orange deviation alarm	Yes	Yes		Yes
%LOOP	Loop			Yes	
%LPET	Loop process execution time	Yes	Yes		Yes
%LPV	Loop process variable	Yes	Yes		Yes
%LPVH	Loop process variable high	Yes	Yes		Yes
%LPVL	Loop process variable low	Yes	Yes		Yes
%LRCA	Loop rate-of-change alarm	Yes	Yes		Yes
%LRSF	Loop ramp soak flags	Yes	Yes		Yes
%LSP	Loop setpoint	Yes	Yes		Yes
%LSPH	Loop setpoint high	Yes	Yes		Yes
%LSPL	Loop setpoint low	Yes	Yes		Yes
%LTD	Loop derivative time	Yes	Yes		Yes
%LTI	Loop integral time	Yes	Yes		Yes
%LTS	Loop sample time	Yes	Yes		Yes
%LVF	Loop V-flags	Yes	Yes		Yes
%LYDA	Loop yellow deviation alarms	Yes	Yes		Yes
%PPET	SFSUB process execution time	Yes	Yes		Yes
%SPET	SFPGM process execution time	Yes	Yes		Yes
%STW	Controller status word *	Yes	Yes	Yes	Yes
%TCC	Timer/counter current	Yes	Yes		Yes
%TCP	Timer/counter preset	Yes	Yes		Yes
%TPET	Task process execution time	Yes	Yes		Yes
%VMM, %VMS	VME memory *	Yes	Yes	Yes	Yes
%V	Variable memory *	Yes	Yes	Yes	Yes
%WX	Word input *	Yes	Yes		Yes
%WY	Word output *	Yes	Yes		Yes
%X	Discrete input *	Yes	Yes		Yes
%Y	Discrete output *	Yes	Yes		Yes

* See the following section, "Direct Addressing Guidelines for Series 505 Controller."

Direct Addressing Guidelines for Series 505 Controllers

For Series 505 controllers, integer values require one memory location. Real values require two memory locations, and are referenced with a period (.) following the address, except in the Declaration Table. For example, %V100. accesses the real value stored in memory location V100 and V101.

For Series 505 S-memory, real values for loops and analog alarms only require one memory location; however, they still have to be referenced with a period (.). Refer to Chapter 2, Table 2-5, for loop memory, and Table 2-13 for analog alarm memory.

The first available direct memory address for a Series 505 controller is 1, not 0. Do not use a direct memory address to access Series 505 control memory addresses X0, Y0, V0, K0, STW0, WX0, or WY0, e.g., %X0, %Y0, etc. These addresses do not exist, and an "illegal direct address" error occurs when the program is compiled.

The following types of memory are available only on the 575 controller: G, VMM, and VMS.

There are two types of Series 505 VME memory: VMS, which corresponds to VME address (16-bit address), and VMM, which corresponds to VME address (24-bit address). Both of these memory types use byte-oriented addressing. For example, writing an integer (V-memory) to %VMM0F000 writes to both %VMM0F000 and %VMM0F001. Consequently, you cannot reference odd-numbered VMM- or VMS-memory addresses.

Both VMS and VMM memory must be referenced with hex addresses. Also, the address must begin with a digit. Since hex addresses can sometimes begin with a letter (A through F), you may have to insert a "0" (zero) in front of the address.

575 G-memory types can be referenced as %G1, or as %GA1, where A is the application ID (A through Z are valid application IDs). If you do not specify an application, the default is the current application. Initial values will only be downloaded for the declarations defined without the application ID.

B.3 Using Direct Addresses (S5 Controllers)

Reserved Memory for S5 Controllers

When you fill out the Compiler Control File, APT allows you to reserve memory locations for your use. Figure B-3 shows the portion of the compiler control screen for an S5 928B or 948U/948R CPU where you specify these reserved memory locations.

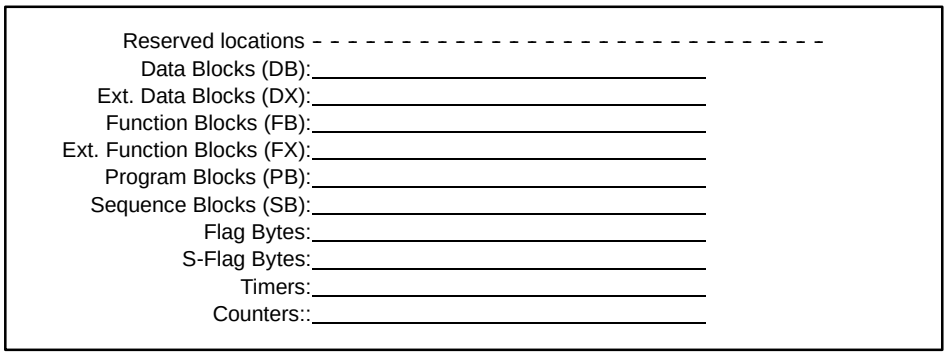


Figure B-3 Control File Reserved Memory (S5)

Enter the actual numbers that you want to reserve. For instance, if you place a 10 after Data Blocks on the form, then you reserve DB10. Separate numbers by commas, and show a range of numbers by placing a colon (**:**) between the starting number and the ending number. For example, if you enter these numbers after Function Blocks: 40, 41, 50:59, 255, then you are reserving FB40, FB41, FB50 through FB59, and FB255. Because you have reserved these function blocks, APT will not use them, and you can program them in STL and download them to the controller using STEP 5.

NOTE: If your function blocks, program blocks, or sequence blocks use any data blocks, flag words, etc., you must reserve these, too.

Data Blocks (DB) Enter the data blocks that your S5 program uses. Valid entries are 5–255 for the 948U/R CPUs, and 3–255 for the 928B. The controller uses DB0–DB2 for a 928B CPU, or DB0–DB4 for the 948U/R. When you reserve a data block, you get the entire data block. APT assumes the length of the DB is 256 words.

Extended Data Blocks (DX) Enter the extended data blocks that your S5 program uses. Valid entries are 2–255. When you reserve an extended data block, you get the entire data block. APT assumes the length of the DX is 256 words.

Function Blocks (FB) Enter the function blocks that you want to reserve. Valid entries are 0–255. These function blocks can be either ones that you purchased or ones you have written. You must download the function blocks to the controller after you download your APT program. Do not put the controller in RUN mode until you have downloaded all reserved function blocks to the controller. Every time you clear the controller you must download the function blocks again. If your function blocks use any data or flags, reserve the appropriate memory. Maximum FB length permitted by APT is 4K.

Ext. Function Blocks (FX) Enter the extended function blocks that you want to reserve. Valid entries are 0–255. You must download the extended function blocks to the controller after you download APT. Every time you clear the controller you must download the extended function blocks again. Maximum FX length permitted by APT is 4K.

Program Blocks (PB) Enter the program blocks that you want to reserve. Valid entries are 0–255. Maximum PB length permitted by APT is 4K.

Sequence Blocks (SB) Enter the sequence blocks that you want to reserve. Valid entries are 0–255. Maximum SB length permitted by APT is 4K.

Flag Bytes Enter the flag bytes that you want to reserve. Valid entries are 2–199. APT uses FW0 (FY0 and FY1) as the H-flag word for redundant control. You cannot reserve flag bytes 200–255 because APT uses them as a scratch flag area. Your function block, program block, or sequence block can share the scratch flag area with APT, but you cannot reserve these bytes.

S-Flag Bytes Enter the extended flag bytes (S-flags) that you want to reserve. Valid entries are 0–1023 for a 928B CPU, or 0–4096 for the 948U/R CPU.

NOTE: When you reserve a flag byte or an extended flag byte, you get all eight bits of the flag. If you are using flags for integers, remember to reserve two flags for each integer. If you are using real numbers, remember to reserve four flags for each real number.

Timers Enter the number of S5 timers that you want to reserve. Valid entries are 0–255, which reserve timers TMR0 to TMR255.

Counters Enter the number of S5 counters that you want to reserve. Valid entries are 0–255, which reserve counters CTR0 to CTR255.

Table B-2 lists the direct addresses that are supported by APT for S5 controllers. See the description for each reserved memory type to determine the range that can be reserved.

Table B-2 Direct Addresses Supported by APT for S5 Controllers

Memory Types	Range	Size in Bits	Description	Debug	MAITT	Declarations and I/O Symbols	Inline Assembly*
%CT	0.0 to 255.15	1	Counters (bits)	Yes	Yes		Yes **
%CTR	0 to 255	16	Counters (words)	Yes	Yes	Yes	Yes
%DB	0 to 255		Data Block				Yes
%DB#:D	0.0 to 255.15	1	Data Bit	Yes	Yes	Yes	Yes
%DB#:DW	0 to 255	16	Data Word	Yes	Yes	Yes	Yes
%DB#:DD	0 to 254	32	Double Data Word	Yes	Yes	Yes	Yes
%DB#:DL	0 to 255	8	Left-Hand Byte of DW	Yes	Yes		Yes
%DB#:DR	0 to 255	8	Right-Hand Byte of DW	Yes	Yes		Yes
%DX	0 to 255		Extended Data Blocks				Yes
%DX#:D	0.0 to 255.15	1	Data Bit	Yes	Yes	Yes	Yes
%DX#:DW	0 to 255	16	Data Word	Yes	Yes	Yes	Yes
%DX#:DD	0 to 254	32	Double Data Word	Yes	Yes	Yes	Yes
%DX#:DL	0 to 255	8	Left-Hand Byte of DW	Yes	Yes		Yes
%DX#:DR	0 to 255	8	Right-Hand Byte of DW	Yes	Yes		Yes
%F	0.0 to 255.7	1	Flag Bit	Yes	Yes	Yes	Yes
%FB	0 to 255		Function Block				Yes
%FY	0 to 255	8	Flag Byte	Yes	Yes		Yes
%FW	0 to 254	16	Flag Word	Yes	Yes	Yes	Yes
%FD	0 to 252	32	Flag Double Word	Yes	Yes	Yes	Yes
%I	0 to 127.7	1	Digital Input	Yes	Yes	Yes	Yes
%IB	0 to 127	8	Digital Input Byte	Yes	Yes		Yes
%IW	0 to 126	16	Digital Input Word	Yes	Yes	Yes	Yes
%ID	0 to 124	32	Double Word Input	Yes	Yes	Yes	Yes

* Cannot use direct addresses in the Math language except in the Inline Assembly Math Statement.
** Only available for 948 CPUs.

Table B-2 Direct Addresses Supported by APT for S5 Controllers (continued)

Memory Types	Range	Size in Bits	Description	Debug	MAITT	Declarations and I/O Symbols	Inline Assembly*
%KB	0 to 255	8	Constant (1 byte)				Yes
%KC	0 to 999	16	Constant (count)				Yes
%KF	-32768 to 32767	16	Constant (fixed point)				Yes
%KG	-1.7E38 to 1.7E38	32	Constant (floating point)				Yes
%KH	0 to FFFF	16	Constant (hexadecimal)				Yes
%KM	0s and 1s only	16	Constant (2 byte-bit pattern)				Yes
%KS	ASCII Characters	16	Constant (2 characters)				Yes
%KT	0.0 to 999.3	16	Constant (time)				Yes
%KY	0 to 255	16	Constant (2 bytes)				Yes
%OW	0 to 254	16	Extended Peripheral Word	Yes	Yes	Yes	Yes
%OY	0 to 255	8	Extended Peripheral Byte	Yes	Yes		Yes
%PB	0 to 255		Program Blocks				Yes
%PW	0 to 254	16	Peripheral Word	Yes	Yes	Yes	Yes
%PY	0 to 255	8	Peripheral Byte	Yes	Yes		Yes
%Q	0.0 to 127.7	1	Digital Output	Yes	Yes	Yes	Yes
%QB	0 to 127	8	Digital Output Byte	Yes	Yes		Yes
%QW	0 to 126	16	Digital Output Word	Yes	Yes	Yes	Yes
%QD	0 to 124	32	Double Output Word	Yes	Yes	Yes	Yes
%RI	0.0 to 255.15	1	Interface Data Area (bits)	Yes	Yes		Yes **
%RIW	0 to 255	16	Interface Data Area (words)	Yes	Yes		Yes

* Cannot use direct addresses in the Math Language except in the Inline Assembly Math Statement.
** Only available for 948 CPUs.

Table B-2 Direct Addresses Supported by APT for S5 Controllers (continued)

Memory Types	Range	Size in Bits	Description	Debug	MAITT	Declarations and I/O Symbols	Inline Assembly*
%RJ	0.0 to 255.15	1	Extended Interface Data Area (bits)	Yes	Yes		Yes **
%RJW	0 to 255	16	Extended Interface Data Area (words)	Yes	Yes		Yes
%RS	0.0 to 255.15	1	System Data Area (bits)	Yes	Yes		Yes **
%RSW	0 to 255	16	System Data Area (words)	Yes	Yes		Yes
%RT	0.0 to 255.15	1	Extended System Data Area (bits)	Yes	Yes		Yes **
%RTW	0 to 255	16	Extended System Data Area (words)	Yes	Yes		Yes
%S	0.0 to 1023.7 (928B) 0.0 to 4095.7 (948U/R)	1	Extended Flag Bit	Yes	Yes	Yes	Yes
%SB	0 to 255		Sequence Blocks				Yes
%SY	0.0 to 1023 (928B) 0.0 to 4095 (948U/R)	8	Extended Flag Byte	Yes	Yes		Yes
%SW	0 to 1022 (928B) 0 to 4094 (948U/R)	16	Extended Flag Word	Yes	Yes	Yes	Yes
%SD	0 to 1020 (928B) 0 to 4092 (948U/R)	32	Extended Double Flag Word	Yes	Yes	Yes	Yes
%TM	0.0 to 255.15	1	Timers (bits)	Yes	Yes		Yes **
%TMR	0 to 255	16	Timers (words)	Yes	Yes	Yes	Yes

* Cannot use direct addresses in the Math language except in the Inline Assembly Math Statement.
** Only available for 948 CPUs.

Direct Addressing Guidelines for S5 Controllers

For S5 controllers, integer values require two flag-byte memory locations or one DB/DX data-word memory location. Real values require four flag-byte memory locations or two DB/DX data-word memory locations. Real numbers are referenced with a period (.) following the address, except in the Declaration Table. For example, %FD100. accesses the real value stored in memory locations FY100, FY101, FY102, and FY103.

Use the APT Declaration Table to assign symbolic names to reserved flag and DB/DX memory so that you can reference flag and DB/DX memory in APT by their symbolic names instead of their direct addresses.

Use an inline assembly math statement to call OBs greater than OB39 and reserved FBs, FXs, PBs, or SBs. Refer to Appendix C of this manual, “Inline Assembly Code for S5,” for information on the inline assembly math statement.

You cannot reserve the scratch flag area (F200 to F255), but you can use it in your own FBs, FXs, PBs, or SBs when you call them with the inline assembly math statement. APT uses this area on an ongoing basis and continually overwrites its contents. If your FB, FX, PB, or SB uses scratch flags, they are local to the block, and they are overwritten after the FB, PB, or SB executes.

If you want to use external FBs, FXs, PBs, or SBs, you must reserve them in the Compiler Control File. Download your APT program before you download your FBs, FXs, PBs, or SBs to the controller. Any time you clear the controller memory, you must download the blocks again. If you put your controller in RUN mode and the blocks have not been downloaded, you get an error and the controller goes to STOP mode. To avoid having to re-download your blocks when you use a Force Compile, do not select the Clear Memory option; this way your blocks remain in the controller memory, and you can set the controller to RUN mode.

When you reserve flag or DB/DX memory, you must download them into the controller before you use them. If you perform a COLDSTART afterwards, the flag words are set to zero and must be downloaded again; however, the DB/DX memory retains the same value as before the COLDSTART. If you perform an overall reset, which clears all memory, you must download both flag and DB/DX memory again.

Appendix C

Inline Assembly Code for S5

C.1 The IN_ASM Statement

Overview

This appendix describes how to integrate external FBs, FXs, PBs, SBs, OBs, DBs, and DXs into APT and how to use the inline assembly math statement that is provided by APT for S5 controllers. The purpose of the inline assembly math statement is to permit you to call external blocks that have been written using STL into APT. You can use inline assembly math statements with the APT Math language.

If you have an S5 controller, you can incorporate the following types of blocks into APT:

- Purchased or previously-written function blocks (FBs) and extended function blocks (FXs), program blocks (PBs), or sequence blocks (SBs).
- Interrupt and non-interrupt organization blocks (OBs).
- Data blocks (DBs) or extended data blocks (DXs) that you need to open.

You can call existing FBs, FXs, PBs, SBs, and OBs in STL, perform load and transfer operations, or open DBs and DXs using an inline assembly math statement.

Table C-1 lists those operations and operands which are supported for inline assembly use. The operations and operands listed here are the only ones supported. Additional operations may be used internally by APT, but these operations are purposely undocumented, with no commitment as to future support or compatiblity.

Availability

Inline assembly math statements are supported only for S5 controllers.

Table C-1 Supported Operations for APT S5 Inline Assembly

	Operations	Operands
Boolean Operations	A, AN, O, ON	%Ix.x, %Qx.x, %Dx.x, %Fx.x, %Sx.x, %TMRx.x, %CTRx.x
	A, AN, O, ON	APT boolean
	=	%Ix.x, %Qx.x, %Dx.x, %Fx.x, %Sx.x
	=	APT boolean
	O	
	A(	
	O(	
	)	
Datablock Currency Operations	C	%DBx
	CX	%DXx
External Block Calls	JC, JU	%FBx, %PBx, %SBx, %OBx
	DOU, DOC	%FXx

Table C-1 Supported Operations for APT S5 Inline Assembly (continued)

	Operations	Operands
Load Operations	L	%IBx, %IWx, %IDx
	L	%QBx, %QWx, %QDx
	L	%FYx, %FWx, %FDx
	L	%DLx, %DRx, %DWx, %DDx
	L	%KBx, %KCx, %KFx, %KGx, %KHx, %KMx, %KSx, %KTx, %KYx
	L	%PYx, %PWx
	L	%OYx, %OWx
	L	%RIWx, %RJWx, %RSWx, %RTWx
	L	%SYx, %SWx, %SDx
	L	%TMRx, %CTRx
	L	APT integer
	L	APT real
	LC	%TMRx, %CTRx
Transfer Operations	T	%IBx, %IWx, %IDx
	T	%QBx, %QWx, %QDx
	T	%FYx, %FWx, %FDx
	T	%DLx, %DRx, %DWx, %DDx
	T	%PYx, %PWx
	T	%OYx, %OWx
	T	%RIWx, %RJWx, %RSWx, %RTWx
	T	%SYx, %SWx, %SDx
	T	APT integer
	T	APT real
Address Error Handling	IAE	
	RAE	

Formatting Inline Assembly Math Statements

Inline assembly math statements can be used in APT anywhere that you can use the Math language: in an SFC step, in a math CFB, in associated math, or in a subroutine. All FB, FX, PB, SB, and OB calls can be made with an inline assembly math statement.

NOTE: You must enter IN_ASM statements entirely in uppercase letters. Do not use mixed case in an IN_ASM statement.

Figure C-1 shows an example of a math block that includes an IN_ASM statement. The math block is called Math 1.

```
{Local declaration section}

INTEGER:  INT5;

BEGIN

{Section for Math statements, including IN_ASM}

INT5  :=  100;

{More math statements}
      .
      .
      .
IN_ASM  [*              {STL statements (L, JU, C, and A are permitted in IN_ASM)}
L MATH1.NNL.INT5   {References INT5, the local variable of the subroutine}
JU %FB101
*];

{More Math statements, including more IN_ASM statements}
      .
      .
      .
```

Figure C-1 Math Example Using an IN_ASM Math Statement

The local declarations for your subroutine follow the same rules as local declarations in the Math language. However, if you want to use a local declaration in an IN_ASM statement, you must reference it according to the following syntax: *Block_name*.**NNL.***Declaration_name*. In Figure C-1, the local declaration, INT1, is referenced within an IN_ASM statement using this syntax.

C.2 Parameters Available for IN_ASM Statements

Overview

When you call an FB, FX, PB, SB, or OB that has parameters, the parameters are either loaded into the accumulators or, for FBs and FXs only, loaded into a parameter list. This section discusses the different types of parameters that you can use in an inline assembly math statement.

Direct Addresses

You can use direct addresses as parameters in an inline assembly math statement. Refer to Appendix B for all the direct addresses allowed. If you use a direct address, you must precede it with the percent sign (%). Do not enter a space after the % sign. All direct addresses supported for S5 controllers are permitted in inline assembly math statements.

```
L %FW10     {loads a direct address}
```

Flags and Other Bits

To enter an S5 bit as a parameter, you must use the AND command. For example, to enter the S5 flag F1.1 as a parameter, use the following syntax:

```
A %F1.1
```

To incorporate the input bit I1.0 as a parameter, use the following syntax:

```
A %I1.0
```

Words and Double Words

To enter an S5 word or double word as a parameter, you must use the LOAD command. For example, to enter the analog input PW128 as a parameter, use the following syntax:

```
L %PW128
```

To incorporate the input double word ID120 as a parameter, use the following syntax:

```
L %ID120
```

APT Symbolic Names

You can use APT symbolic names as parameters in an inline assembly math statement. How you handle the parameter depends on whether you are sending it to the accumulators (for an FB, FX, PB, SB, or OB) or to a parameter list (FB, FX only).

NOTE: When you refer to APT symbolic names in any inline assembly math statement, regardless of where you are sending the parameter, write the symbolic name entirely in uppercase letters. The Table editors for APT use uppercase. If you refer to an APT symbolic name in mixed case when you make inline assembly math statements, the assembler cannot match it to the correct declaration.

If you are using an APT symbolic name as a parameter that is sent to an accumulator, follow these steps.

1. Use the Declaration Table, I/O Table, Recipe Table, or Device Table to declare the APT symbolic name(s). You can use automatic addressing.

2. Issue a load statement according to the following syntax.

```
L THETA     {loads the symbolic name Theta}
            {notice that the symbolic name is uppercased}
```

NOTE: When you use the symbolic name, the call to the DB/DX occurs when loading parameters to the accumulators, but not when loading parameters in a parameter list.

If you are using APT symbolic names as parameters that are sent to a parameter list, follow these steps.

1. Use the Compiler Control File to reserve a DB/DX.
2. Use the Declaration Table, Recipe Table, or Device Table to declare the APT symbolic name(s) and assign them to addresses in the DB/DX. All the symbolic names you want to send to the parameter list must be stored in the same DB/DX.
3. Use an inline assembly math statement to call the DB/DX that contains your APT symbolic names. You can then jump to your FB or FX, and load the desired APT symbolic names into the parameter list.

   Follow the syntax that is shown below for your inline assembly math statement. For this example, DB50 has been reserved in the Compiler Control File, and the APT symbols ANGLE1 and ANGLE2 are real numbers stored in the DB. You want to load ANGLE1 and ANGLE2 into FB101.

```
IN_ASM[*    {identifies beginning of inline assembly statement}
C %DB50     {open DB50}
JU %FB101   {jump to FB101}
JU 3        {reason for this jump is explained in Section C.3}
L ANGLE1    {loads the symbolic name ANGLE1}
L ANGLE2    {loads the symbolic name ANGLE2}
            {notice that the symbolic names are uppercased}
*];         {identifies the end of inline assembly statement}
```

Constants

You can also use the S5 constants as parameters in inline assembly math statements. These constants cannot be used anywhere else in APT. When you use S5 constants in an inline assembly math statement, you must enter them according to the formatting requirements of APT, which are listed in Table C-2. For examples of how to load the different S5 constants, see Table C-3.

Table C-2 S5 Constants Used in Inline Assembly Math Statements

Constant	Type	Range	APT Formatting Requirements
KB[1]	1 byte	0 to 255	Value must be only 1 byte.
KY	2 bytes	0 to 255 for each byte	Byte values must be separated by commas.
KC	count	0 to 999	Value is the counter number.
KT	time	0.0 to 999.3	Value identifies 3-digit multiplier and timebase.
KF	fixed point	-32768 to +32767	Plus sign is optional.
KG	floating point	1.7E38 to -1.7E38	Can enter with a decimal point or in the E format.
KH [2]	hexadecimal	0 to FFFF	Value must be entered in hexadecimal form.
KM	bit pattern (16 bits)	0s and 1s only	Value must be entered in binary form.
KS [3]	2 characters	ASCII characters	Characters must be entered in single quotation marks.
DH [1,2]	32-bit hexadecimal	0000 0000 to FFFF FFFF	Value must be entered in hexadecimal form.

1 KB and DH can only be used to load parameters into the accumulators, not in a parameter list.
2 If your hexidecimal number begins with a letter, you must precede the letter with a 0 (zero).
3 You must enter 2 characters; blanks are allowed.

Table C-3 provides examples of load statements for the different S5 constants. The load statement requires a space between the L command and the constant. Do not put a space between the constant and the value that follows the constant.

Table C-3 Loading Constants for Inline Assembly Math

Load Statement	Comment
L %KB50	Loads a byte with a value of 50.
L %KY1,4	Loads two bytes with values of 1 and 4.
L %KC10	Sets the counter to a maximum count of 10.
L %KT256.1	Identifies the timebase (0.1 sec) and the multiplier (256), resulting in a timeout of 25.6 seconds.
L %KF+ 2	Loads an integer with a value of 2.
L %KG100.0	Loads a floating point number with a value of 100.0.
L %KG1.0E2	Loads a floating point number with a value of 100.0.
L %KH01A0	Loads a hexadecimal number with a value of 01A0.
L %KM00010001 00010001	Loads a bit pattern with a value of 0001 0001 0001 0001.
L %KS'ST'	Loads the ASCII values for the letters "S" and "T".
L %DH0000FFFF	Loads a hexadecimal number with a value of 0000FFFF.

C.3 How to Incorporate Your Block

Overview

To incorporate an FB, FX, PB, SB, DB, or DX into APT, you must first reserve the block in the Compiler Control File. For information about how to reserve blocks in the Compiler Control File, see the appendix on "Direct Memory Addressing" in this manual. You do not have to reserve OBs in the Compiler Control File, because OBs are resident in the S5 controller.

You can call external FBs, FXs, PBs, OBs, SBs, DBs, and DXs in an inline assembly math statement from any one of these four locations: from within a math CFB, from associated math, from an SFC step with Math language, or from a subroutine. For information about how to edit a math CFB, see Chapter 8 of this manual. To learn about the Math language, see Chapter 10 of this manual. To edit a subroutine, see Chapter 10, "User Subroutines," in the *SIMATIC APT Programming Reference (Tables) Manual*.

You must download the external FBs, FXs, PBs, SBs, DBs, and DXs to the controller using STEP 5. The inline assembly math statements are downloaded with APT. When you download your program to the controller initially, be sure to download APT before you download the FBs, FXs, PBs, SBs, DBs, and DXs. If you clear the memory in the controller later for any reason, you must re-download all external FBs, FXs, PBs, SBs, DBs, and DXs.

Only selected OBs can be executed using the inline assembly math statement. If you have a 928B or 948U CPU, you can execute OB40–OB255. If you have a 948R CPU, you can execute OB121–OB255; OB40–OB120 are used for redundancy.

If you want to execute OB1–OB39, you need to use an APT system-activated subroutine, described in the "User Subroutines" chapter of the *SIMATIC APT Programming Reference (Tables) Manual*.

NOTE: You cannot modify the code in OBs using STEP 5. If you try to edit OB1–OB39 using STEP 5, your modifications are overwritten by APT.

Incorporating Blocks with No Parameters

Many FBs, FXs, PBs, SBs, and OBs do not pass parameters through the accumulators or in a parameter list. If you want to incorporate an FB, FX, PB, or SB without parameters into APT, you must first reserve the block in the Compiler Control File. You can then edit your math CFB, SFC step, associated math, or subroutine.

Use the following format to create inline assembly math statements:

1. Enter the inline assembly identifier that indicates the beginning of the statement.

   ```
   IN_ASM[*
   ```

2. Enter the jump (that is, the call) to the FB, FX, PB, SB, or OB.

   ```
   JU %FBx
   JU %FXx
   JU %PBx
   JU %SBx
   JU %OBx
   ```

   where `x` is the number that identifies the FB, FX, PB, SB, or OB.

3. Enter the inline assembly identifier that indicates the end of the statement.

   ```
   *];
   ```

   An inline assembly statement is a Math statement and must end in a semicolon.

Example

The following example shows an inline assembly math statement that incorporates a function block, FB101, into APT. Assume FB101 is a user-written subroutine that identifies the end of a telegram. FB101 uses an already existing DB, DB50. Both FB101 and DB50 have been reserved in the Compiler Control File. No parameters are required for FB101, and nothing is sent to the accumulators upon completion. This FB is used to send a telegram to the CP524 card. The FB determines the end of the telegram by calculating the error-checking LRC. FB101 then puts the LRC at the end of the telegram.

```
BEGIN
IN_ASM[*   {identifies beginning of inline assembly statement}
C %DB50    {open DB50}
JU %FB101  {jump to FB101}
*];        {identifies the end of inline assembly statement}
```

Loading a Parameter List

The only type of block that uses a parameter list is a function block/extended function block (FB/FX). If you want to incorporate an external FB or FX with parameters into APT, you must first reserve the FB or FX in the Compiler Control File. You can then edit your math CFB, SFC step, associated math, or subroutine.

Follow this format to enter inline assembly math statements:

1. Enter the inline assembly identifier that indicates the beginning of the statement.

   ```
   IN_ASM[*
   ```

2. Enter the jump (that is, the call) to the FB.

   ```
   JU %FBx
   ```

   where x is the number that identifies the FB.

3. Enter the jump statement to jump past the parameters.

   ```
   JU y
   ```

   where y is the length of parameters in words, plus 1.

   To calculate the length of your parameters, see Table C-4.

4. Load the parameters.

   See Section C.2 for examples of load statements for the various types of parameter (direct addresses, symbolic names, bits, constants) that are permitted.

5. Enter the inline assembly identifier that indicates the end of the statement.

   ```
   *];
   ```

   An inline assembly statement is a Math statement and must end in a semicolon.

NOTE: An S5 FB/FX can accept up to 40 parameters.

Use Table C-4 to calculate the length of the parameters you are loading into your parameter list. Table C-4 shows the length in words for the address types of the various parameters that can be used with LOAD (**L**) statements and with AND (**A**) statements.

Table C-4 Size of Various Address Types

Address Type	Description	Parameter Format	Number of Words
CTR	Counter	L %CTRn	1
DB	Data blocks	C %DBn	1
DD	Data double word	L %DDn	1
DL	Left (that is, high-order) data byte	L %DJn	1
DR	Right (that is, low-order) data byte	L %DRn	1
DW	Data word	L %DWn	1
F	Flag	A %Fn.m	1
FB	Function block without parameter list	JU %FBn	1
FD	Flag double word	L %FDn	1
FW	Flag word	L %FWn	1
FY	Flag byte	L %FYn	1
I	Input bit	A %In.m	1
IB	Input byte	L %IBn	1
ID	Input double word	L %IDn	1
IW	Input word	L %IWn	1
KC	Count constant (BCD) in range 0...999	L %KCn	1
KF	Fixed-point number in the range -32768 to +32767	L %KFn	1
KG	Floating-point number in the range -1.7E38 to +1.7E38	L %KGn	2
KH	Hexadecimal constant (maximum 4 digits)	L %KHn	1
KM	Binary constant (16 digits)	L %KMn	1
KS	Character constant (2 ASCII characters)	L %KS	1
KT	Time constant with time base 1.0...999.3	L %KTn.m	1
KY	Two one-byte absolute values separated by a comma	L %KYn.m	1
OW	Extended peripheral word	L %OWn	1
OY	Extended peripheral byte	L %OYn	1
PB	Program block	JU %PBn	1
PW	Peripheral word	L %PWn	1
PY	Peripheral byte	L %PYn	1
In all parameter format examples, **n** represents the byte, and **m** represents the bit.			

Table C-4 Size of Various Address Types (continued)

Address Type	Description	Parameter Format	Number of Words
Q	Output bit	A %Qn.m	1
QB	Output byte	L %QBn	1
QW	Output word	L %QWn	1
QD	Output double word	L %QDn	1
RIW	System transfer data word	L %RIWn	1
RJW	Extended system transfer data word	L %RJWn	1
RSW	System data word	L %RSWn	1
RTW	Extended system data word	L %RTWn	1
SB	Sequence block	JU %SBn	1
TMR	Timer	L %TMRn	1
In all parameter format examples, **n** represents the byte and **m** represents the bit.			

NOTE: You cannot load KB, DH, SY, SW, or SD data types as a parameter in a parameter list.

⚠ WARNING

The parameter list must match what the FB/FX requires.

If you change, add, or delete parameters, the FB/FX code in the controller changes in an unpredictable manner. Unpredictable controller operations could result in death and/or damage to equipment.

Double check your parameters list against the parameters required by the FB/FX. If you do change, add, or delete parameters, double check your FB after you download the parameters to ensure correctness.

See the following page for an example of an inline assembly math statement that calls a function block and loads a parameter list.

Example

The following example shows an inline assembly math statement that calls a function block and loads a parameter list. The size of the parameter list was calculated using the information from Table C-4, as shown below.

```
L %KF+128       {Value of KF, from LOAD table, is 1 word}
L %KS' P'       {Value of KS, from LOAD table, is 1 word}
L %KY1,4        {Value of KY, from LOAD table, is 1 word}
L %KG100.0      {Value of KG, from LOAD table, is 2 words}
L %KG0.0        {Value of KG, from LOAD table, is 2 words}
A %I1.0         {Value of I, from AND table, is 1 word}
L OUT           {Value of FD or DD, from LOAD table, is 1 word}
A %Q1.0         {Value of Q, from AND table, is 1 word}
A %Q1.1         {Value of Q, from AND table, is 1 word}
A %Q1.2         {Value of Q, from AND table, is 1 word}
```

The block in this example is FB40, an analog input scale FB. The analog input is %PW130, which is a zero-bias input with a range of 0.0 to 100.0. The result of the analog scaling is placed in the variable named OUT1. This variable, OUT1, was declared in the declaration section of APT and given a user-assigned address in DB6, in accordance with the procedure described in Section C.2. Since the size of the parameter list, as calculated above, is 12, the jump to go past the parameters is 13.

```
BEGIN
IN_ASM[*        {identifies beginning of inline assembly statement}
C %DB6          {calls the DB that stores the APT symbol named OUT1}
JU %FB40        {jump to FB40}
JU 13           {jump the length of parameters (12 words) plus 1}
L %KF+128       {first parameter, first I/O point on analog card}
L %KS' P'       {analog card is in the PW area}
L %KY1,4        {the second I/O point on analog card; it is zero bias}
L %KG100.0      {the upper limit of the scaled value}
L %KG0.0        {the lower limit of the scaled value}
A %I1.0         {bit that requests a single scan}
L OUT1          {the result from FB40 goes into the APT symbol OUT1}
A %Q1.0         {the broken transmitter bit}
A %Q1.1         {the overrange bit}
A %Q1.2         {bit that indicates whether single scan has occurred}
*];             {identifies the end of inline assembly statement}
```

Using the Accumulators

You can use the accumulators to pass the parameters of an external FB, FX, PB, SB, or OB. You must reserve FBs, FXs, PBs, and SBs in the Compiler Control File before you incorporate them into APT. You do not have to reserve OBs in the Compiler Control File.

Four accumulators are available for storage of the parameters of your blocks; do not load more than four accumulators. Each time you load a parameter into Accumulator 1 it moves the other parameters over to the next higher accumulator. If you want to load all four accumulators, follow these steps.

1. Load the first two parameters into Accumulator 1 and enter the **ENT** command.
2. Load the third parameter into Accumulator 1 and enter the **ENT** command.
3. Load the fourth parameter into Accumulator 1.

   The first parameter moves to Accumulator 4, the second parameter moves to Accumulator 3, the third parameter moves to Accumulator 2, and the last parameter stays in Accumulator 1.

⚠ WARNING

You can load a maximum of four accumulators before you begin to overwrite data. Loading more than four accumulators could result in unpredictable operation of the controller.

Unpredictable operation of the controller could result in death or serious injury to personnel, and/or damage to equipment.

Do not load more than four parameters, total, to the accumulators.

Loading the Accumulators

Follow this format to use an inline assembly math statement to load the accumulators:

1. Enter the inline assembly identifier that indicates the beginning of the statement.

   ```
   IN_ASM[*
   ```

2. Load the accumulators.

   See Section C.2 for examples of load statements for the various types of parameter (direct addresses, symbolic names, bits, constants) that are permitted.

3. Enter the jump to the FB, FX, PB, SB, or OB.

   ```
   JU %FBx
   JU %FXx
   JU %PBx
   JU %SBx
   JU %OBx
   ```

   where x is the number that identifies the FB, FX, PB, SB, or OB.

4. Enter the inline assembly identifier that indicates the end of the statement.

   ```
   *];
   ```

   An inline assembly statement is a Math statement and must end in a semicolon.

Example

This example shows an inline assembly math statement that loads parameters for a cosine function block at FB102 into the accumulators. Both the inputs and the results of the cosine function block are stored in the accumulators. In this example, THETA is an angle that has already been declared in the declaration section of APT. COSTHETA is a variable that has been declared in APT; it holds the result of the FB. Notice that THETA and COSTHETA do not have to have a user-assigned address, since they are passed through the accumulators instead of through a parameter list.

```
BEGIN
IN_ASM[*        {identifies beginning of inline assembly statement}
L THETA         {load THETA into Accumulator 1}
JU %FB102       {jump to the cosine FB}
= %F250.2       {error flag}
T COSTHETA      {transfer result from Accumulator 1 to COSTHETA}
*];             {identifies the end of inline assembly statement}
```

This code loads THETA into Accumulator 1 and then jumps to FB102. The cosine of THETA is placed into Accumulator 1 and then transferred to the variable COSTHETA. If the calculation is successful, FB102 sets RLO.

Opening a Data Block

If your direct address is a data word, you can call the data block (DB or DX) in the inline assembly math statement before you use the data word, as long as you have already reserved the DB or DX in the Compiler Control File.

If you declare a data word in the declaration section of APT, and assign it to a reserved DB or DX, you can use the left-hand byte and the right-hand byte. To specify the left-hand byte, you can either use the memory type DL. To specify the right-hand byte, use memory type DR.

The following example calls a data block, DB3, and loads the left-hand and right-hand bytes of data word 12.

```
BEGIN
C %DB3            {calls DB3}
L %DB3:DL12       {loads left-hand byte of DW12}
L %DB3:DR12       {loads right-hand byte of DW12}
```

You must issue the call to the DB/DX before the jump to an FB/FX if you are using the DB/DX data as inputs to a parameter list.

Index

Symbols

A

B

C

D

E

F

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X

Customer Response

We would like to know what you think about our user manuals so that we can serve you better. How would you rate the quality of our manuals?

	Excellent	Good	Fair	Poor
Accuracy	_____	_____	_____	_____
Organization	_____	_____	_____	_____
Clarity	_____	_____	_____	_____
Completeness	_____	_____	_____	_____
Graphics	_____	_____	_____	_____
Examples	_____	_____	_____	_____
Overall design	_____	_____	_____	_____
Size	_____	_____	_____	_____
Index	_____	_____	_____	_____

Would you be interested in giving us more detailed comments about our manuals?

☐ **Yes!** Please send me a questionnaire.

☐ **No.** Thanks anyway.

Your Name: _____

Title: _____

Telephone Number: (____) _____

Company Name: _____

Company Address: _____

_____

_____

Manual Name: APT Programming Reference (Graphics/Math) Manual **Edition:** Tenth
Manual Assembly Number: 2801047–0007 **Date:** 4/01
Order Number: PPX:APT–8102–10

FOLD

SIEMENS ENERGY & AUTOMATION INC
3000 BILL GARLAND ROAD
PO BOX 1255
JOHNSON CITY TN 37605-1255

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO.3 JOHNSON CITY, TN

POSTAGE WILL BE PAID BY ADDRESSEE

ATTN TECHNICAL COMMUNICATIONS M/S 1255
SIEMENS ENERGY & AUTOMATION INC
PO BOX 1255
JOHNSON CITY TN 37605-1255

FOLD