SIEMENS

SIMATIC APT

Programming Reference (Tables) Manual

Order Number: PPX:APT-8102–10
Text Assembly Number: 2801048–0007
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.

MANUAL PUBLICATION HISTORY

SIMATIC APT Programming Reference (Tables) 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 (2801048–0001)
Fifth Edition	02/94	Fifth Edition (2801048–0002)
Sixth Edition	10/94	Sixth Edition (2801048–0003)
Seventh Edition	05/95	Seventh Edition (2801048–0004)
Eighth Edition	11/96	Eighth Edition (2801048–0005)
Ninth Edition	12/98	Ninth Edition (2801048–0006)
Tenth Edition	4/01	Tenth Edition (2801048–0007)

LIST OF EFFECTIVE PAGES

Pages	Description	Pages	Description
Cover/Copyright	Tenth Edition		
History/Effective Pages	Tenth Edition		
Trademarks/Copyrights	Tenth Edition		
iii — xxv	Tenth Edition		
1-1 — 1-26	Tenth Edition		
2-1 — 2-6	Tenth Edition		
3-1 — 3-13	Tenth Edition		
4-1 — 4-90	Tenth Edition		
5-1 — 5-29	Tenth Edition		
6-1 — 6-41	Tenth Edition		
7-1 — 7-106	Tenth Edition		
8-1 — 8-53	Tenth Edition		
9-1 — 9-15	Tenth Edition		
10-1 — 10-17	Tenth Edition		
A-1 — A-106	Tenth Edition		
Index -1 — Index-11	Tenth Edition		
Registration	Tenth Edition		

Trademarks and Copyrights

Contents

4.2 Standard Modules (continued)

4.3 Intelligent Modules 4-46

List of Figures

List of Figures (continued)

List of Figures (continued)

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 (Tables) Manual* (Volume 2) and *SIMATIC APT Programming Reference (Graphics/Math) Manual* (Volume 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 APT 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

APT Overview

1.1 Process Control Solutions

Modular Approach

APT is a program design tool that allows you to solve a process control problem by using a structured, top-down approach. While APT encourages and supports this modular technique, it does not require you to use this approach.

Modular design involves breaking a large problem into smaller, more manageable pieces. You can solve each subproblem independently, and then integrate these subsets to provide a total solution. This structured design approach can significantly reduce the time that is necessary to develop and debug a given solution.

APT Philosophy

A basic philosophy of APT is that the control solution should reflect the natural organization of the physical plant. In arranging the physical layout of the plant, the design engineer usually breaks the process into unit operations (mixers, reactors, etc.). While each unit is designed independently, the engineer also considers the overall requirements for the integrated process.

For example, Figure 1-1 shows a process that consists of a weighing system and two kettles, or cookers, along with the corresponding pipes, pumps, valves, etc. Typically, the design engineer determines the size of the equipment according to the general production requirements and then deals with the design specifics on an equipment-by-equipment basis.

The process control engineer can use a similar approach and divide the control solution along natural equipment boundaries. The overall control requirement is to move ingredients through a weigh system, into the cookers, and then on to the next processing phase. Each equipment area has its own requirements that can be determined independently; however, integration requirements make it necessary to consider the inter-connectivity of the total solution.

A successful modular design partitions the problem according to what is needed solely for unit operation and what is needed to connect the units. If this approach is used in the problem in Figure 1-1, for example, the solution to the control of Cooker 1 is also the solution to the control of Cooker 2. This approach significantly reduces the effort to solve the total problem.

NOTE: The *SIMATIC APT Applications Manual* gives a more complete discussion of the recommended approach to designing a process control program for an application.

Basic Tasks

In designing a control solution with APT, you must complete the following basic tasks.

- Identify the physical process, and partition it along natural boundaries that define the equipment and ingredients that are involved.
- Identify the control process, which defines the sequence of events in the process as well as the continuous control of equipment and ingredients.
- Identify exceptional conditions that require special processing or operator intervention.

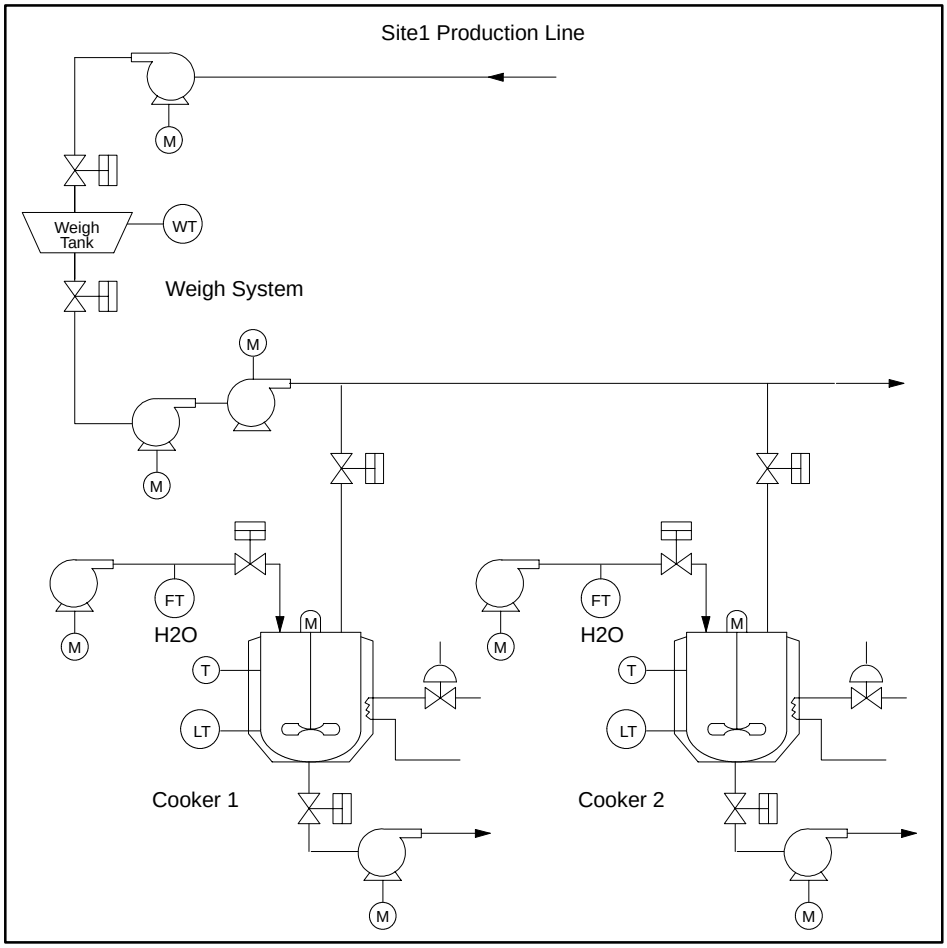


Figure 1-1 Production Process Line

1.2 Identifying the Physical Process

Equipment Areas

One of the basic tasks in designing the process control solution is to identify the physical process. Because of the modularity of APT, one of the primary considerations in identifying the physical process is to partition that process into manageable units that can be operated independently. This is the first step in breaking the large problem into subproblems.

One method of breaking the control solution of a production line into subproblems is by partitioning along lines of major processing equipment. For example, Figure 1-2 shows four equipment areas: a weigh system, two cookers, and a transportation system that moves ingredients from the weigh system into the cookers.

Equipment

A subtask of identifying the physical process is to identify the pieces of equipment that are used in each equipment area.

For example, the weigh system in Figure 1-2 consists of two valves and a weight indicator. Each cooker consists of a water valve, a dump valve, an agitator motor, a steam valve, a flow transmitter, and temperature and level indicators. The transportation system includes two motor-driven pumps to transport the ingredients.

Ingredients

Another subtask of identifying the physical process is to identify the ingredients that are used in the process. In the example in Figure 1-2, the weight of each ingredient is part of the weigh system, and cooking times that are based on ingredients are part of the cookers.

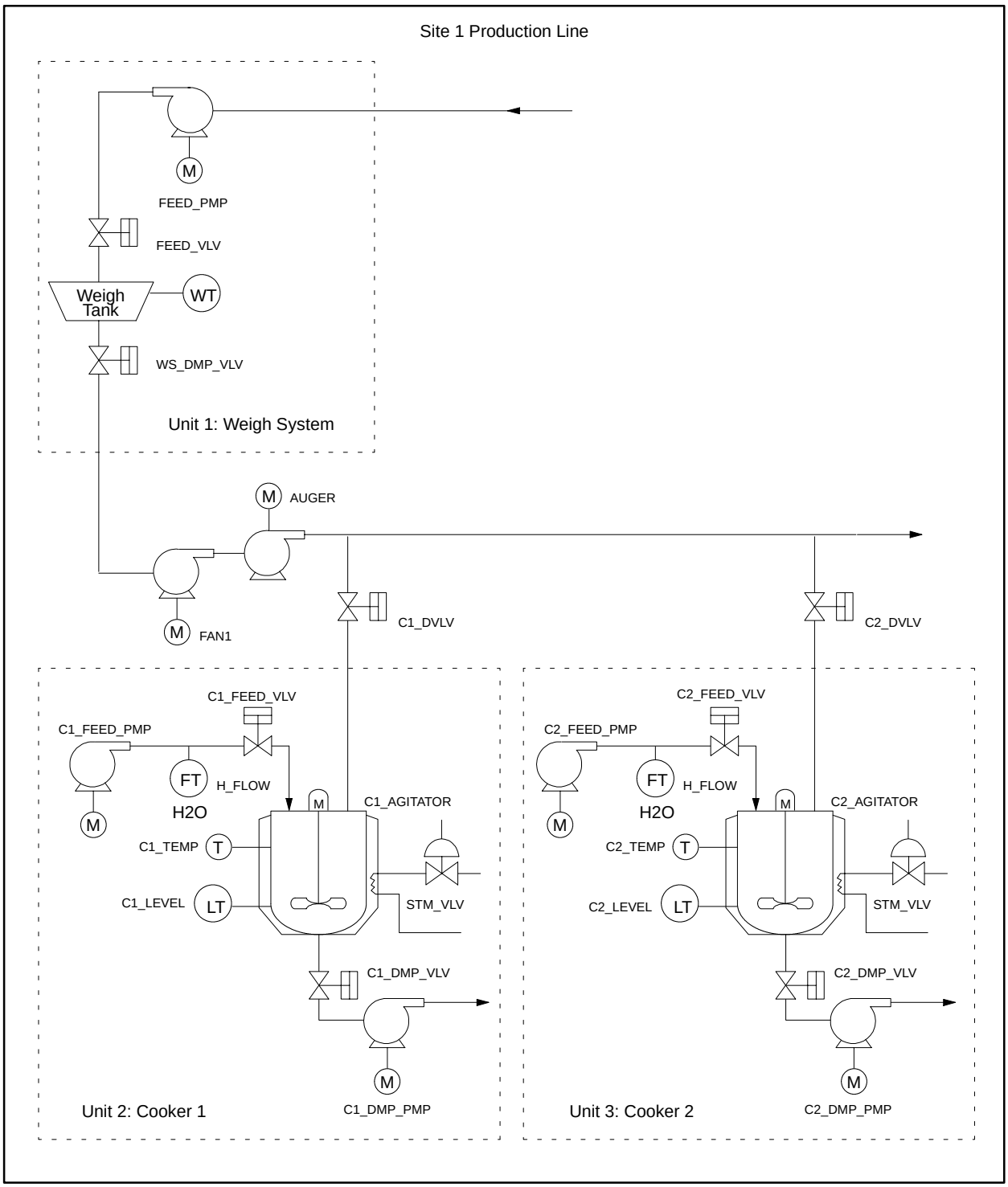


Figure 1-2 Units in Process Production Line

1.3 Identifying the Control Process

Sequence of Events

To define the sequence of events, examine each step, or state, that the process must follow in order to produce a product. If you break the physical process into manageable units, the definition of this sequence becomes much easier.

In the production line example, you can define a relatively simple process if you look only at the steps involved in one cooker. The sequence of events for one cooker include the following.

- Fill cooker.
- Cook ingredients.

  Turn on steam.

  Raise temperature to certain point.
- Mix ingredients.

  Start agitator.
- Cool ingredients.

  Reduce temperature to certain point.
- Dump cooker.

Although each of these steps can involve additional details, an upper-level definition is sufficient at this point. Later, each major step or procedure can be refined further to produce the total control solution.

Continuous Control

Continuous control is used to define the control of field equipment that must be controlled no matter what step is currently being executed in the process.

For example, in the cooking process, the steam valve must be controlled continuously to reach and maintain the specified temperature. The temperature transmitter must be monitored continuously to determine when the specified temperature is reached.

1.4 Identifying Safe-State Conditions

Emergency Conditions

In any physical process, situations can arise that require you to stop the process and make certain that all equipment is in a safe-state condition. For example, if a motor overheats, it may be necessary to use a special procedure to control the process during this emergency. In designing the control solution, be sure to identify all situations that require special procedures.

In the cooker, for example, emergency conditions might include the following.

- Failure of the steam valve to close.
- Failure of weigh system to deliver appropriate amount of ingredient.
- Failure of agitator motor to start.

Emergency Procedures

Like the control process for normal operation, the control process for handling special conditions involves not only a sequence of events but also continuous control and monitoring of equipment.

For example, if continuous monitoring of the temperature in the cooker indicates that the steam valve has failed to close at the appropriate time, you may need to dump the cooker in order to avoid overheating the ingredients. This procedure requires the dumping process to occur outside of the normal sequence of events.

1.5 APT Program Structure and the Physical Process

APT Hierarchy

The APT environment provides a hierarchical structure that assists you in breaking your large process control problem into subproblems. The hierarchy shown in Figure 1-3 consists of three levels.

- Program Directory Level
- Program Content Level
- Unit Content Level

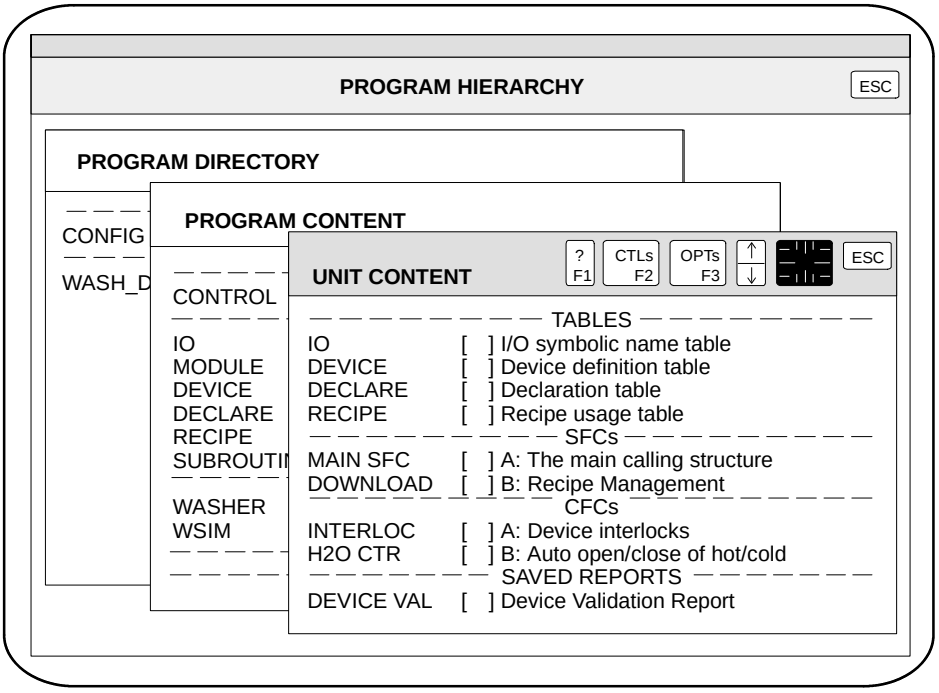


Figure 1-3 APT Hierarchy

Program Directory

The Program Directory Level contains a list of all programs, as shown in Figure 1-4. The physical process that is identified in solving the process control problem is the basis for the definition of a program.

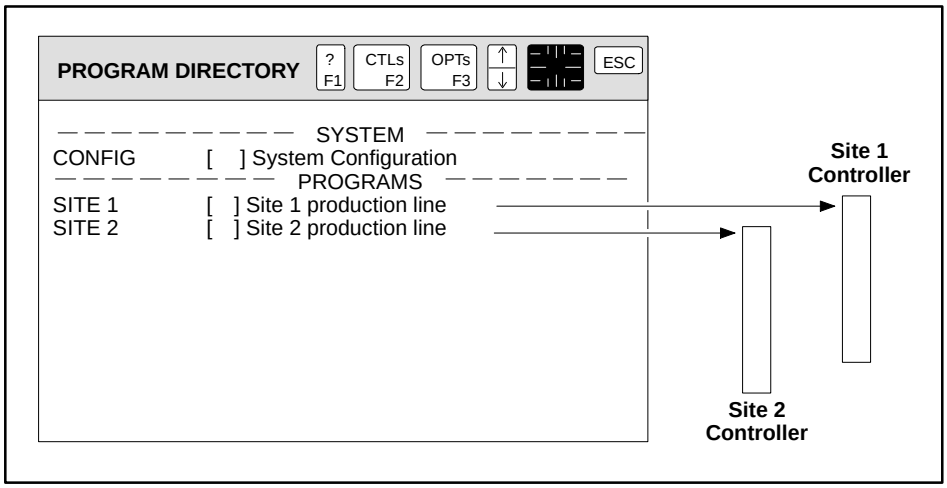


Figure 1-4 APT Program Directory

A program in APT is that portion of the process that can run on a single controller. The size of the program depends on controller memory size, safety considerations, and other characteristics of the process line.

Program Content

The Program Content Level, as shown in Figure 1-5, contains a list of all units that you identify as a part of one program. The equipment areas, or units, that are identified in the process line are the basis of the unit definitions at the Program Content Level.

This level of the hierarchy includes a means of identifying equipment and ingredients that must be accessed by more than one unit, or area, of equipment. User-designed and customized tasks are defined in the subroutine table, which is also at the program level.

For example, the transportation system in the Site 1 Production Line shown in Figure 1-5 needs to be accessed by all three units in the process. The equipment in the transportation system is defined in the Device Definition Table, and control can be handled by the program-level Sequential Function Charts (SFC)s and Continuous Function Charts (CFC)s. Ingredients for the product(s) are defined in the Recipe Template. The amounts of ingredients for each product are defined in the Recipe Usage Table.

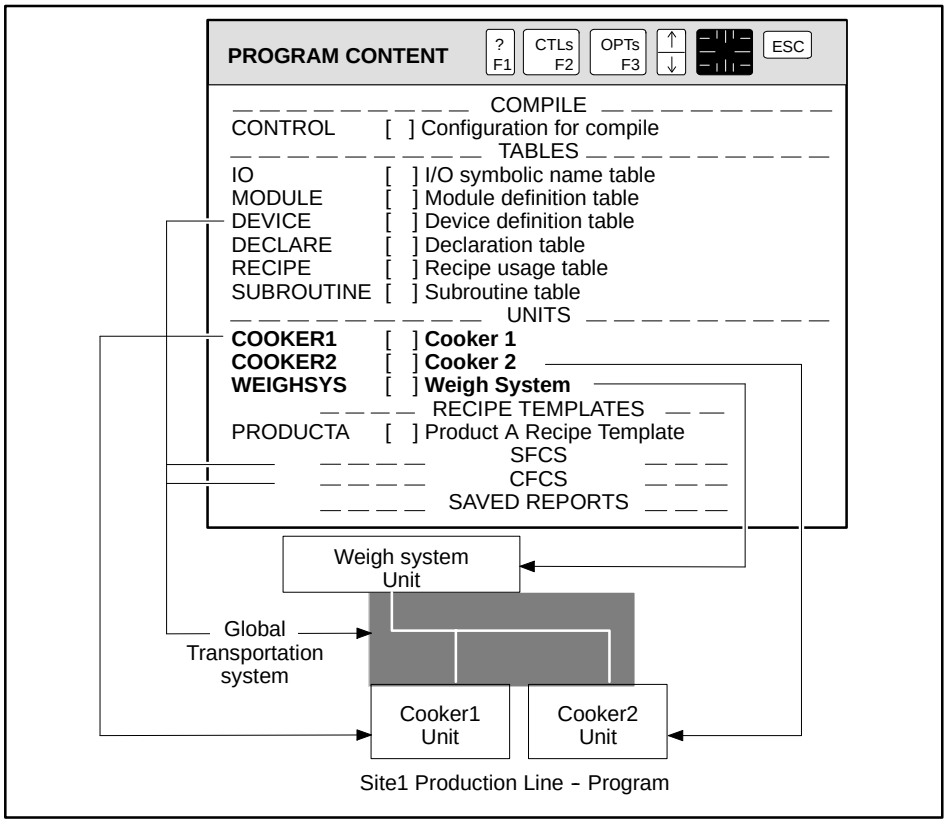


Figure 1-5 APT Program Content Directory

Unit Content

The Unit Content Level contains the definition of the actual control processes, including sequential and continuous procedures that are specific to the unit, as well as special procedures to handle emergency conditions. This level also includes a means of identifying equipment and ingredients that are local to each unit.

Sequential control of the process at the unit level is defined in one or more SFCs, which are listed at the Unit Content Level, as shown in Figure 1-6. Continuous control at the unit level is defined in one or more CFCs.

The equipment that is local to the unit is identified in the Device Definition Table at the Unit Level. Amounts of ingredients for the products made in the unit are stored in the Recipe Usage Table at the Unit Level.

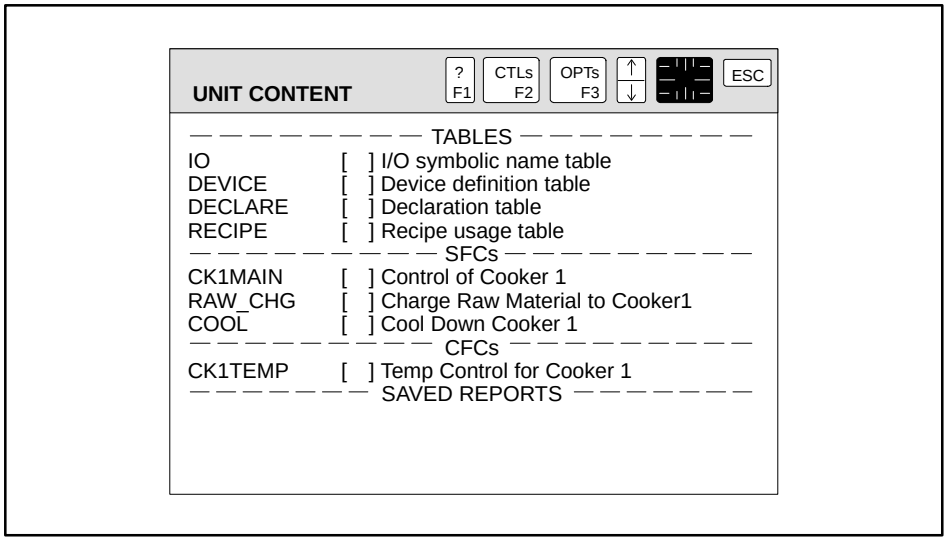


Figure 1-6 APT Unit Content Directory for COOKER1

1.6 APT Objects

APT Object Definition

An APT object is any portion of the control system that can be named and has attributes that can be configured. A unit is an object that includes a set of items that define the unit. These items that define the unit are also objects. These items include field devices (such as pumps, motors and valves), control strategies, and processes. SFCs and CFBs are also objects.

All of these objects combine to define the control of the unit. You open a valve to begin filling the cooker; you open a steam valve to start heating the jacket of the cooker; and you start the motor to start agitating the cooking ingredients.

Table 1-1 lists the basic APT objects, along with the maximum characters that can be used in each object name.

Table 1-1 APT Objects

Object	Naming Location	Maximum Characters
Program	Program Directory	8
Unit	Program Content	8
SFC	Program Content Unit Content	8
CFC	Program Content Unit Content	8
CFB	CFC	8
I/O Symbolic Name	I/O Symbolic Name Table Module Definition Table	12
Device	Device Definition Table	12
Constant	Declaration Table	12
Variable	Declaration Table	12
Recipe	Recipe Usage Table	12
Recipe Template	Program Content	8
Subroutine	Program Content	8

Naming Objects

Observe the following rules when you name an object.

- Valid characters that can be used in APT object names are alphanumeric characters and the underscore.
- APT object names must contain at least one letter. Object names are not case-sensitive; that is, you can enter a name in either upper-case or lower-case letters.

- APT object names must not be in the form of scientific notation, for example, 45E6 is not a valid object name.
- No words listed in Table 1-2 can be used as an object name.

Table 1-2 Key Words

ABS	COM4	INT_TO_REAL	ONN	SFC
AND	CON	LEAD_LAG	ONS	SIN
ARCCOS	COS	LEFTSHIFT	OR	SQRT
ARCSIN	EDGE	LIMIT	ORDER	SSABORT
ARCTAN	ELSE	LN	OUT	SSARM
ARRAY	ELSIF	LOAD_ARRAY	PBITS_TO_INT	SSDEFINE
ASM	END	LOG	PRAGMA	SSDISARM
AUX	ENDIF	LOGSTEP	PRINT	SSENTRY
BCDBIN	EXP	LOOKUP_TABLE	PRIORITY	SSRETURN
BEGIN	FALSE	LOOP	PRN	SSTRIGGER
BINBCD	FLAG	LPT1	PROUND	STATUS
BITCLEAR	FOR	LPT2	PTRUNC	SUBROUTINE
BITS_TO_INT	FRAC	LPT3	PUBLIC	TAN
BITSET	FRS	LPT4	RANGE	THEN
BITTEST	GLOBAL	MATH	REAL	TIMING
BLOCK	IF	MAX	RECORD	TRIGGER
BODY	IN	MIN	RETENTIVE	TRUE
BOOLEAN	IN_ASM	MINMAX	RETURN	TRUNC
BY	INHERIT	MOD	REVERSE	TYPE
CHAR	INIT	NIL	RIGHTSHIFT	UNSCALE
CLEAR	IN_OUT	NOT	ROUND	UNTIL
COM1	INTEGER	OF	SCALE	WHILE
COM2	INTERPOLATE	OFF	SCOPE	XOR
COM3	INT_TO_BITS	ON	SETSSI	

Configuring Objects

Each object that you name in APT must be configured. To configure a program, for example, you define units and global equipment and ingredients. APT provides a set of six definition tables at the Program Content Level that you use to define equipment and ingredients and user-defined subtasks. These tables, shown in Figure 1-7, allow you to define the I/O modules in your program as well as global I/O symbolic names, devices, subroutines, declared constants and variables, and recipes. Program-level sequence control is handled in the program-level SFCs. Program-level continuous control is handled in the program-level CFCs.

To configure a unit, you define local equipment, ingredients, and the control processes. Four definition tables are available at the Unit Content Level, as shown in Figure 1-7. Each unit contains a table for defining I/O symbolic names, devices, declared constants and variables, and recipes that are local to the unit. SFCs and CFCs are part of a unit and are configured at this level also.

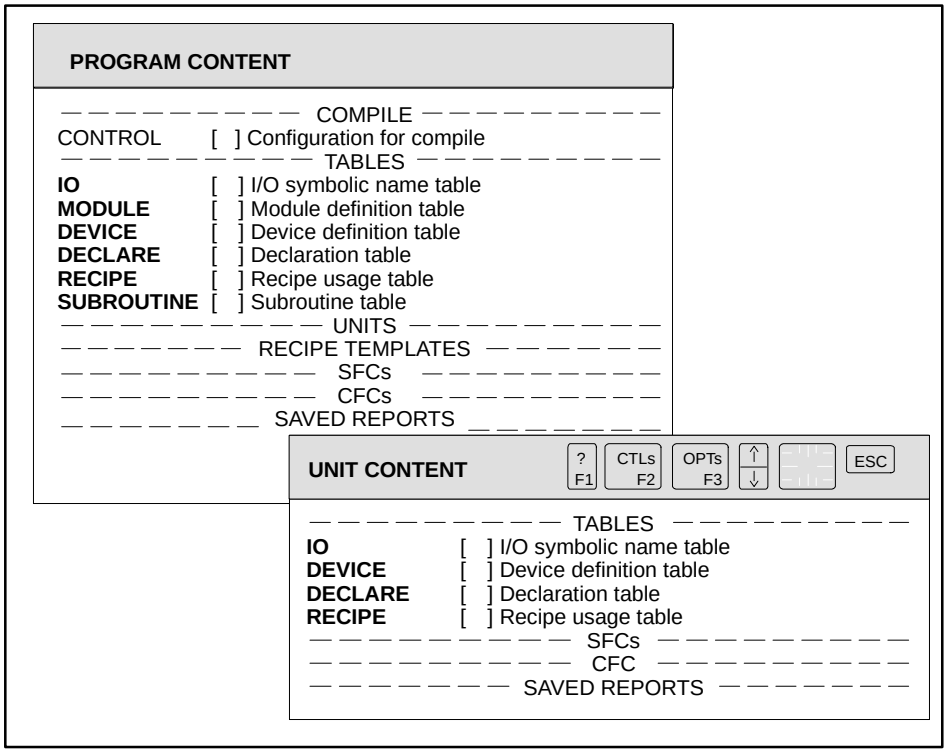


Figure 1-7 APT Tables

Scope of Objects

Because of the modular approach of APT, objects have a definite scope within the program. Objects that are defined at the Program Content Level are global and can be accessed from any point within the program.

Objects that are defined at the Unit Content Level are local to that unit and can be accessed only within that unit.

When you name objects in APT, you must consider the scope of that object and use names that are unique within that scope.

Dot Extensions

When you name an object, you can access various aspects of that object by appending a dot extension to the name that you created.

For example, if you identify *switch102* as the open limit switch for *valve102*, you can also refer to that switch as *valve102*.**OLS**. The dot extension **.OLS** always indicates an open limit switch for the named device. You can determine if *valve102* is open by checking the status of *valve102***.OPND**.

Each type of object has a corresponding list of dot extensions that provide access to the different attributes that you configured for that object.

Recipes

Recipes provide a structure that allows you to define the necessary ingredients to produce a product. Recipes can also serve as a structure for reports, control strategies, etc., and are considered objects that you configure. The Recipe Template at the Program Level allows you to create your own form for configuring the attributes of a recipe. The values of the ingredients in a recipe can be configured at either the Program Level or at the Unit Level.

For example, if Cooker 1 is designed to create three different products, you can create three recipes (one for each product) in the Recipe Usage Table at the Program Level. Since you can create only one product at a time in the unit, you can define a working recipe at the Unit Level and then move the appropriate recipe from the Program Level into the working recipe at the Unit Level.

1.7 APT Languages and the Control Process

Textual and Graphics-based Languages

APT provides four languages that you use to define the actual solution to your control process. Table 1-3 lists these languages according to type.

Table 1-3 APT Programming Languages

Language Type	Programming Language
Textual	The State Control Language The Math Language
Graphics-based	The Sequential Function Chart (SFC) Language The Continuous Function Chart (CFC) Language

- The State Control Language consists of commands and expressions that you use in SFCs to control and evaluate the states of objects.
- The Math Language allows you to provide additional custom control by defining mathematical calculations in SFCs, CFCs and subroutines.
- The Sequential Function Chart (SFC) Language uses boxes (⊏⊐) that represent steps, or states, to define the sequential control of your process. Transitions (†) control the flow from box to box. Lines that connect these symbols define the program flow from one step to another.
- The Continuous Function Chart (CFC) Language uses boxes that represent pre-defined algorithms that you configure to provide continuous control of your process. You can also design your own algorithms in the user-defined subroutines.

The State Control Language and the Math Language are called textual languages to differentiate them from the graphics-based languages, namely, the SFC Language and the CFC Language.

With the SFC and CFC Languages, you combine graphical elements on the screen and configure relationships and characteristics with graphics and text. With the text-based languages, you create a series of sequentially executed program statements.

State Control Language

The State Control Language is a textual language that includes both commands and conditional expressions. The commands are related specifically to the objects that they control and have a common format as shown in Figure 1-8.

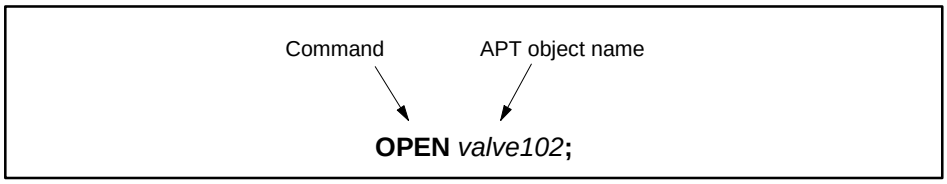


Figure 1-8 APT Step Commands

Conditional expressions in the State Control Language evaluate to true or false and monitor the state of an object as shown in Figure 1-9.

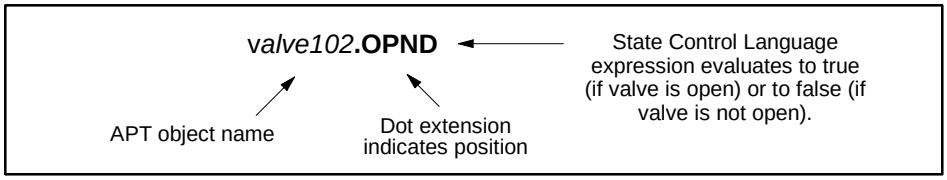


Figure 1-9 Conditional Expressions

Math Language

The Math Language is a text-based language that provides a means for you to perform arithmetic calculations and logical operations in both sequential and continuous control operations.

The Math Language allows you to make assignment statements, use WHILE loops, use IF..THEN..ELSE expressions, and perform various pre-defined math functions and procedures.

Sequential Function Chart Language

The SFC Language is a graphics-based language that allows you to define the sequence of events that control your physical process. SFCs control and monitor the normal operation of objects. The SFC can be thought of as the "supervisor" of all of the other objects within the unit.

For example, the Site 1 Production Line has a sequential order of processing in the cooker unit.

- Fill the cooker and then turn on the steam.
- Wait for the temperature to reach a certain point and then turn on the agitator.
- Wait for the temperature to cool down and then open the drain valve.

This sequential order of events is translated into the SFC Language, which consists of a sequence of steps and transitions. A step specifies the control action; a transition monitors the states of objects to determine when the process is ready to proceed to the next step.

An SFC step can contain one or more State Control Language statements. An SFC step can also include a math section consisting of one or more Math Language statements.

An SFC transition consists of one conditional expression, which determines when the system deactivates one step and activates the next step.

Figure 1-10 shows the initial portion of an SFC. In Step 1, the system requests an action that opens *valve102*. The system remains in that step, or state, until it receives information that the valve is opened (Transition 1). This transition controls the flow, or sequence, from Step 1 to Step 2.

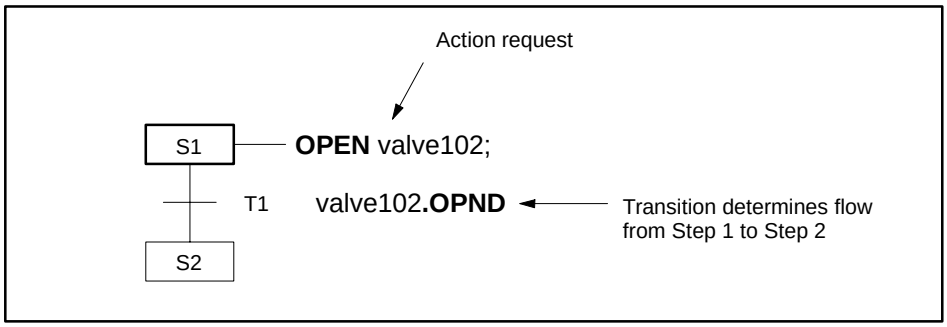


Figure 1-10 SFC Steps and Transitions

Like the physical process, the control process can be broken into small, logical pieces. The SFC Language allows you to nest SFCs; that is, one step can call another SFC and, therefore, make it possible to solve problems in parts.

For example, you can have a main SFC for Cooker 1. The first phase, or step, in the main SFC calls a subordinate SFC that prepares the devices for the process. The second step calls another subordinate SFC that fills the cooker and monitors the ingredients. In configuring the main SFC, you define the upper level of the process (procedures) without concern for details. In configuring a subordinate SFC you concentrate on details, but only on the details related to a smaller portion of the overall process control solution.

The SFC Language includes parallel branches that make it possible to do two or more things simultaneously. Selection branches provide a decision-making function that allows the system to follow different paths depending on specified circumstances.

Continuous Function Chart Language

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

A Continuous Function Chart consists of one or more Continuous Function Blocks (CFBs) that are represented as boxes. Each box represents a pre-defined or user-defined algorithm that you configure to control objects. A CFB can represent an analog alarm, a PID loop, a limiter, a selector, etc. A form allows you to configure all the attributes of the CFB, and you can embed additional calculations by providing associated Math Language statements.

In the Cooker example, various devices need to be controlled continuously to allow the cooker to reach and hold a specified temperature. In the example in Figure 1-11, the PID loop controls the steam valve and reads the temperature transmitter to be certain that the temperature is reached and maintained continuously.

After naming and configuring the PID block, the system controls the CFB like a device. The CFB can be enabled and disabled from an SFC, and the CFB has dot extensions that can be used to access status or other configured attributes.

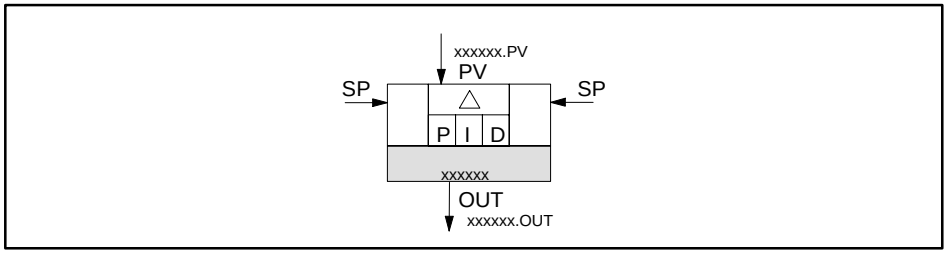


Figure 1-11 Continuous Function Blocks

1.8 APT and Safe-State Conditions

Monitoring Emergency Conditions

APT allows you to define special procedures to handle situations that make it necessary to stop the process for safety reasons. For example, you can configure a CFB to provide continuous monitoring of conditions that could become critical.

Safe-State SFCs

Safe-state SFCs allow you to define special sequential procedures for use when critical conditions are detected. Safe-state SFCs are separate operations for handling special processing conditions. In an emergency, therefore, the program can exit the normal sequence of events and go to another SFC to shut down equipment and return the process to safe operational conditions.

To configure a safe-state SFC, you must first define the conditions that trigger the execution of the special procedure. You can define a priority level that determines which safe-state SFC is executed if several emergency conditions are detected simultaneously. You can also define a safe-state SFC that is associated with only a single SFC and all its subordinate SFCs.

1.9 Controller Support

Controller Models Supported By APT

APT supports two controller families, the Series 505 and the S5. The Series 505 controllers that APT supports are the following models.

- SIMATIC 545
- SIMATIC 545L (545 Lite)
- SIMATIC 555
- SIMATIC 560
- SIMATIC 560T
- SIMATIC 565
- SIMATIC 565T
- SIMATIC 565P
- SIMATIC 575

The S5 controllers that APT supports are these models.

- S5-135U with 928B CPU
- S5-155U with 948 CPU
- S5-155H with 948R CPU

Table 1-4 shows the part numbers for the Series 505 controller/CPU configurations that are supported for this release.

Table 1-4 Series 505 Configurations in APT

Controller	CPU Part Number(s)
545	545-1101 Rel. 2.1
545	545-1102 Rel. 3.x
545L	545-1103 Rel. 4.x
545	545-1104 Rel. 4.x
545	545-1105 Rel. 5.x
545	545-1106 Rel. 5.x
555	555-110x Rel. 3.x, 4.x, and 5.x
560	560-2120 Rel. 3.x
560T	560-2820 Rel. 6.x
565	560-2120 Rel. 3.x and 565-2120 Rel. 3.x
565T	560-2820 Rel. 6.x and 565-2120 Rel. 3.x
565P	560-2820 Rel. 6.x and 565-2820 Rel. 2.x
575	575-210x Rel. 3.x, 4.x, and 5.x

Table 1-5 lists the minimum firmware support and SIMATIC STEP 5 support for controller models in the S5 family.

Table 1-5 APT Minimum Firmware Support and STEP 5 Support (S5)

Feature	CPU		
	928B	948R	948U
Minimum Firmware	Revision 2	Revision 1	Revision 1
STEP 5 Compatible	Stage 6, version 6.3	Stage 6, version 6.3	Stage 6, version 6.3

Table 1-6 lists the minimum firmware support and SIMATIC TISOFT support for controller models in the Series 505 family. For releases earlier than the minimums supported by APT, you must purchase an upgrade for your controller.

Table 1-6 APT Minimum Firmware Support and TISOFT Support (Series 505)

Feature	Controller								
	545	545L	555	560	560T	565	565T	565P	575
Minimum Firmware	2.1	4.0	3.0	3.0	6.0	3.0	See Note 1	See Note 2	3.0
TISOFT Compatible	Rel. 4.0	Rel. 6.0	Rel. 4.0	Rel. 4.0	Rel. 4.0	Rel 4.0	Rel. 4.0	Rel. 4.0	Rel. 4.0
Note 1: The 560-2820 card must be Release 6.0 or higher. The 565-2120 card must be Release 3.0 or higher. Note 2: The 560-2820 card must be Release 6.0 or higher. The 565-2820 card must be Release 2.0 or higher.									

Table 1-7 lists the APT tools that are supported for each controller family (Series 505 and S5).

Table 1-7 APT Tools Supported by Series 505/S5 Controllers

Tool	Series 505	S5	Tool	Series 505	S5
APT Compile Control	Yes	Yes[1]	Units	Yes	Yes
Module Editor	Yes	No	SFCs	Yes	Yes
I/O Editor	Yes	Yes	CFCs	Yes	Yes
Devices	Yes	Yes	Reports	Yes	Yes[3]
Declarations	Yes	Yes	Debug	Yes	Yes
Recipes	Yes	Yes	MAITT	Yes	Yes
Subroutines	Yes	Yes[2]	Industrial Ethernet	Yes	Yes

1 S5 users can incorporate FBs, PBs, and SBs that they have written or purchased.
2 S5 users can program OB1-OB39 in subroutines.
3 Series 505 and S5 users get a comprehensive report of all the I/O that they have configured in APT.

Table 1-8 lists the APT features supported by the different S5 CPUs.

Table 1-8 APT Features Supported by S5 Controller Models

Feature	CPU		
	928B	948U	948R
WHILE Loops	Yes	Yes	Yes
Array Indexing	Yes	Yes	Yes
Real/Integer Array Indexing with Expressions	Yes	Yes	Yes
Boolean Array Indexing with Expressions	No	No	No
Real Math	Yes	Yes	Yes
All CFBs Supported	Yes	Yes	Yes
SFPGM-only Instructions	No [1]	No [1]	No [1]
Analog I/O Modules Support	Yes	Yes	Yes
Immediate Read/Write	Yes	Yes	Yes
COPY_BYTES Procedure	No	No	No
Real Number PID Inputs	Yes	Yes	Yes
Integer PID Inputs	No	No	No

1 While SFPGM code is not used with S5 controllers, all SFPGM functionality has been incorporated into the STL code.

Controller Support (continued)

Table 1-9 lists the APT features supported by the Series 505 controllers.

Table 1-9 APT Features Supported by Series 505 Controller Models

Feature	Controller								
	545	545L	555	560	560T	565	565T	565P	575
WHILE Loops	Yes	Yes	Yes	No	No	Yes	Yes	Yes	Yes
Array Indexing	Yes	Yes	Yes	Note 1	Note 1	Yes	Yes	Yes	Yes
Real/Integer Array Indexing with Expressions in RLL	Note 5 Note 8	Yes Note 8	Yes Note 8	No	No	No	No	No	Yes Note 8
Boolean Array Indexing with Expressions in SFPGM	Yes	Yes	Yes	No	No	Yes	Yes	Yes	Yes
Real Math	Yes	Yes	Yes	Note 2	Note 2	Yes	Yes	Yes	Yes
All CFBs Supported	Yes	Yes	Yes	Note 3	Note 3	Yes	Yes	Yes	Yes
SFPGM-only Instructions	Yes	Yes	Yes	No	No	Yes	Yes	Yes	Yes
Analog I/O Modules Support	Yes	Yes	Yes	Note 4	Note 4	Yes	Yes	Yes	Yes
Use Local Cs on the CPU	Yes	No	Yes	No	Yes	No	Yes	Yes	Yes
Cyclic RLL task support	Note 5	Yes	Yes	No	No	No	No	No	Yes
Interrupt RLL task support	Note 6	Yes	Note 6	No	No	No	No	No	No
Immediate Read/Write	Note 5	Yes	Yes	No	No	No	No	No	Yes
Fast Bitpick operations	Note 7	Yes	Yes	No	Note 7	No	Note 7	Note 7	Yes
COPY_BYTES procedure	Note 5	Yes	Yes	No	No	No	No	No	Yes
Real Number PID Inputs	Yes	Yes	Yes	No	No	Yes	Yes	Yes	Yes
Integer Number PID Inputs	Yes	Yes	Yes	No	No	Yes	Yes	Yes	Yes

1 Arrays can be addressed with literal indexes only.
2 Real numbers can be used, but only with limited MATH capabilities.
3 Only Interlock and Math CFBs can be used.
4 Analog input points must have the .RAW extension. For example, AI point IVAR1 must be addressed as IVAR1.**RAW**. APT cannot add scaling or filtering to the input from these points.
5 545 firmware Rel. 3.0 and higher.
6 545 firmware Rel. 3.0 and higher. 555 firmware Rel. 3.0 and higher
7 545 firmware Rel. 3.0. 560T RLL card firmware Rel. 6.x.
8 Avoid indexing with expressions wherever possible because it produces less efficient code generation.

Chapter 2

Programs and Units

2.1 Understanding APT Program and Unit Extensions

Program Definition

A program in APT is that portion of the process that can run on a single controller. The actual size of the program depends on controller memory size, safety considerations, and other characteristics of the process line.

At the Program Content Level of an APT program you define the divisions of plant operation (APT units) for your process along lines of equipment and ingredient requirements. At this level, you also define the I/O modules, I/O symbolic names, devices, declarations, recipes, and user-defined subtasks (subroutines) that you want to access from all APT units in the program. Recipe templates are also defined at the Program Content Level. Global sequential and continuous control requirements can be handled at the Program Content Level through global SFCs and CFCs.

Unit Definition

The APT unit contains the control logic for a single division of plant operation. This includes sequential (unit-level SFCs) and continuous (unit-level CFCs) control and the safe-state procedures for the unit. At this level, you also define the I/O symbolic names, devices, declarations, and recipes that are local to this unit only.

APT Extensions

When you name a program, APT creates the extension variables that are listed in Table 2-1 and described in the paragraphs that follow.

Table 2-1 APT Program and Unit Extensions

Level	Extension	Definition
Program-Level Extensions	**.TSCAN**	RLL or STL scan time in milliseconds (read-only integer)
	.PWRFL	powerfail (read-only boolean)
	.TODS [1]	time of day in seconds (read-only real)
	.TOD [1]	time of day in hours (read-only real)
	.IHOUR [1]	current hour past midnight (read-only integer)
	.IMIN [1]	current minute of the hour (read-only integer)
	.ISEC [1]	number of seconds in the current minute (read-only integer)
	.IYEAR [1]	current calendar year (read-only integer)
	.IMONTH [1]	month of the year (read-only integer)
	.IDAY [1]	day of the month (read-only integer)
	.IDOW [1]	day of the week (read-only integer)
Program- and Unit-Level Extensions	**.DIS_SF1** [2,3]	Disable first of two control blocks (read/write boolean)
	.DIS_SF2 [2,3]	Disable second of two control blocks (read/write boolean)
	.ENABL	Enable (read/write boolean)
	.ABORT	Abort (read/write boolean)
	.ECODE1 [2,4]	Error code (read-only integer)
	.IID1 [2,3]	Error controller block (read-only integer)
	.SNUM1 [2,3]	Error line number (read-only integer)
	.ECODE2 [2,4]	Error code (read-only integer)
	.IID2 [2,3]	Error controller block (read-only integer)
	.SNUM2 [2,3]	Error line number (read-only integer)
	.OVRUN2 [2,3]	Indicates that the second control block is taking more than a half second to complete the calculations (read-only integer)

1 Not used by 560/560T controllers. Not created if **Do not convert TOD to integer fields** field is selected in Compiler Control File.
2 Not used by 560/560T controllers.
3 Not used by S5 controllers; these extensions pertain to SFPGM code.
4 For S5 controllers, program level **ECODE1** and **ECODE2** contain runtime error information; however, unit level **ECODE1** and **ECODE2** do not. For Series 505 controllers, **ECODE1** and **ECODE2** contain error information at both the program and the unit level.

Stopping and Restarting SFCs

The *unit_name*.**ENABL** extension can be used to restart the unit SFCs either during the debug operation or from an OSx (PCS) workstation. The *prog_name*.**ENABL** extension restarts the program SFCs.

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 **.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 **.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 *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.

Monitoring Analog Inputs

When you create a program for a Series 505 controller, APT defines two control blocks that are used to monitor analog inputs.

- The first control block is used to scale unfiltered analog inputs and most math CFBs (exceptions: Interlock and Math). This process is executed as quickly as possible.
- The second control block monitors filtered analog inputs. This process is executed as quickly as possible.

Both control blocks can be disabled by setting these extensions to true: *prog_name*.**DIS_SF1** and *prog_name*.**DIS_SF2**.

Availability

These control blocks require special function programming, and are supported for Series 505 controllers only. S5 controllers can scale and filter analog inputs, but do not use these control blocks or the extensions (**.DIS_SF1**, **.DIS_SF2**, **.ECODE**, **.IID**, and **.SNUM**) associated with them. The 560/560T controllers do not use these control blocks or their associated extensions, and do not scale or filter analog inputs.

The extensions listed below provide data about the two SFPGM control blocks that monitor the global analog inputs in all Series 505 controllers except for the 560/560T. In the descriptions below, the value of *x* would equal one to indicate the first control block, or two for the second control block.

- If an execution error occurs, the *prog_name*.**ECODE***x* extension contains a non-zero value.
- The *prog_name*.**IID***x* extension identifies the SFPGM associated with the block.
- The *prog_name*.**SNUM***x* extension contains the SFPGM line number that contains the error.
- The control blocks can be disabled by setting the *prog_name*.**DIS_SF***x* extension to true.

Table 2-2 lists the SFPGM error codes and the associated number that appear in the **.ECODE1** or **.ECODE2** extension.

Table 2-2 Program Extension 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)

Using Scan Time and Time-of-Day Variables

You can use the *program_name*.**TSCAN** extension variable to read the program scan time for all controller models. For Series 505 controllers, you can also read status word 10, using the format %STW10, to obtain the scan time.

The time-of-day extension variables (**.TODS**, **.IHOUR**, **.ISEC**, etc.) can be used anywhere in the program to read the current time from the real-time clock. These extensions roll over at midnight. You cannot set the clock from your APT program, although you can set the clock using the APT Debug option. You can also set the clock from TISOFT if you have a Series 505 controller.

If you have a Series 505 controller, you can access status words 141–144 to read the real-time clock. Use the format %STW## and read the date or time as a variable. See the your controller's programming reference manual more information.

Using the Powerfail Variable

The powerfail extension variable (*prog_name*.**PWRFL**) can be monitored and used to trigger a high-priority interlock or a safe-state SFC in the event of a power loss. The powerfail bit is set on return from powerfail and remains true for one controller scan.

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. To control the response to a power failure, use a retentive safe-state SFC. Place an SSARM command in the initial step of the main SFC, and use the *prog_name*.**PWRFL** extension as a trigger in the SSTRIGGER command.

Chapter 3

I/O and Module Definitions

3.1 Understanding Module and I/O Definitions

Referring to I/O

The APT system provides two methods for referring to an I/O point. Series 505 controllers and S5 controllers can both use these two methods, which are explained below.

- **Symbolic Name** The standard method of referring to an I/O point in a program is with an I/O symbolic name that you define in the I/O Symbolic Name Table as explained in Chapter 5.

  If you define a symbolic name, you can use that name anywhere in the program to refer to the actual, physical I/O point; and you can change the location of the I/O module without making changes to your program.

- **Direct Address** Another method of referring to an I/O point is to use the internal controller address, or direct address, that is associated with the field wiring to an I/O module.

  The internal controller address is not typically used in the program. If you use these direct addresses, such as %WX, %WY, etc., for Series 505, or %PW, %OW, etc., for S5, in your program, and later move a module across channels (Series 505) or change the module's address (S5), you have to change all program references to those addresses. See the Math Language Overview Chapter in the *SIMATIC APT Programming Reference (Graphics/Math) Manual* for more information about direct memory addressing.

Figure 3-1 shows how you configure the symbolic name and the controller address for your I/O if you have a Series 505 controller. Figure 3-2 shows how you configure the symbolic name and the controller address for your I/O if you have an S5 controller.

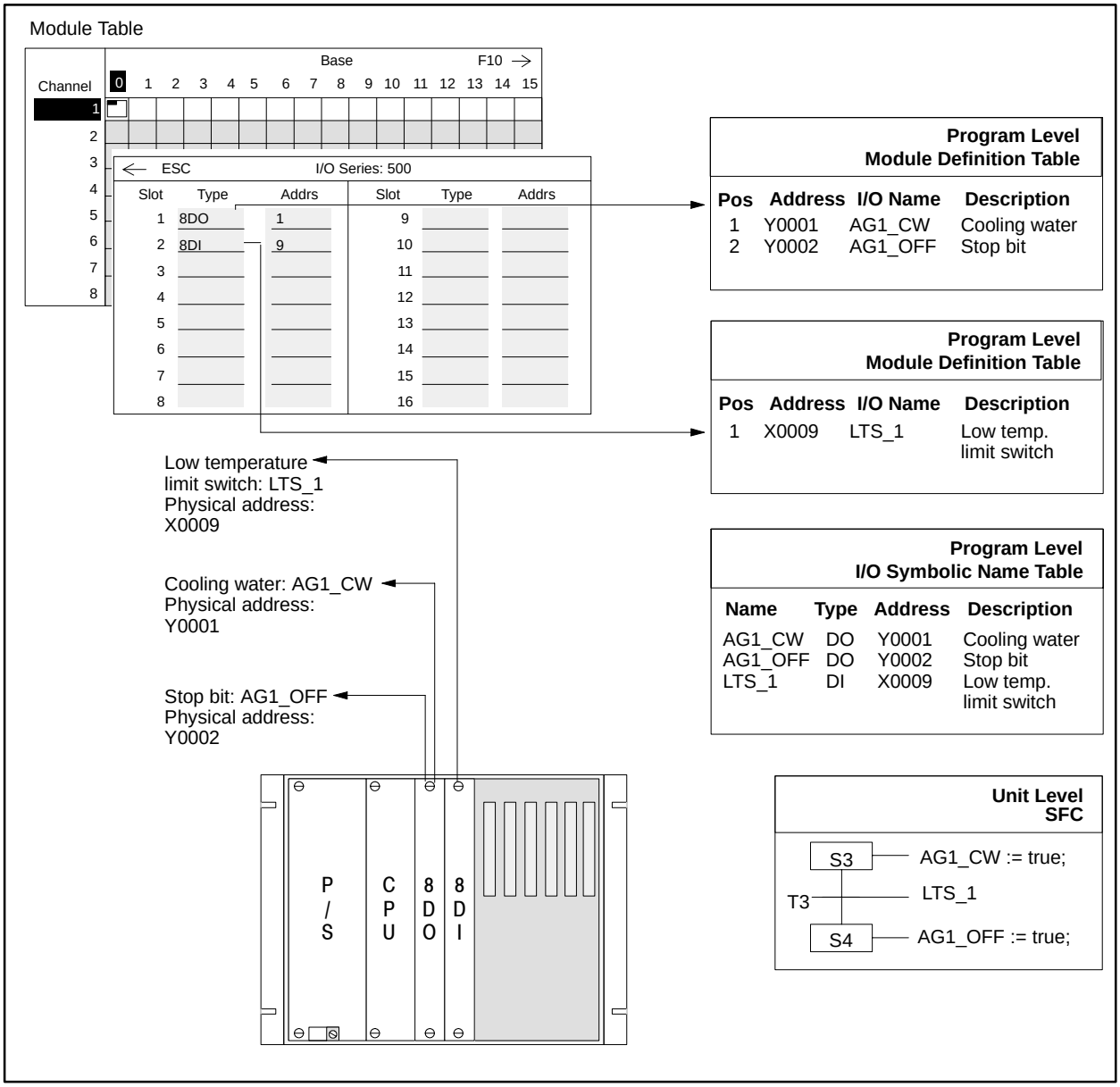


Figure 3-1 Physical Addresses and I/O Symbolic Names for Series 505 Controllers

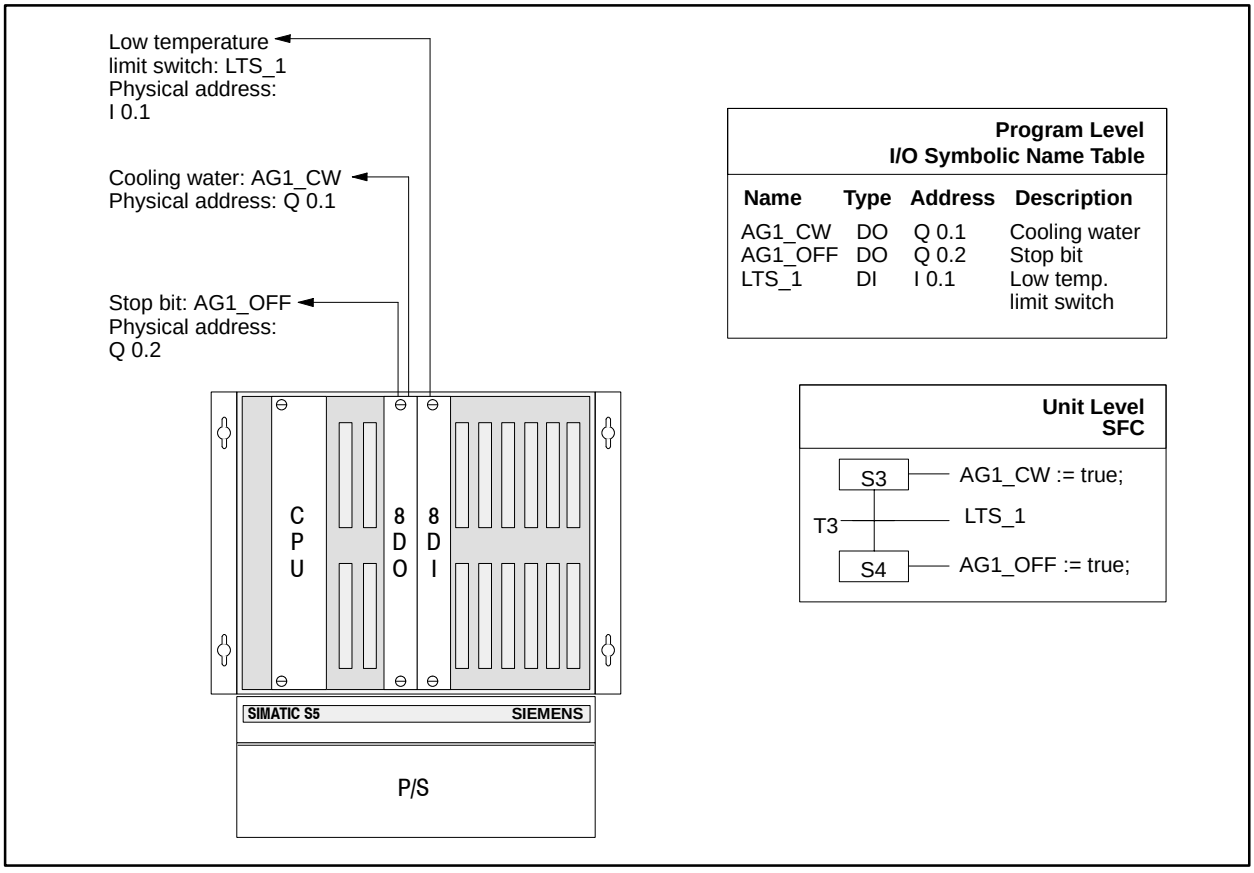


Figure 3-2 Physical Addresses and I/O Symbolic Names for S5 Controllers

I/O and Module Tables

An I/O Symbolic Name Table is available at both the Program Level and at the Unit Content Level. I/O names that are defined at the program level are considered global and can be used in any unit in that program. I/O names that are defined at the unit level can be used only within that unit.

The Module Definition Table is a utility which allows you to specify the location of a module in a channel, base, and slot. The Module Definition Table also associates a controller memory address with each I/O symbolic name that identifies a physical I/O point wired to a module.

For S5 controllers, you must define the internal controller address in the I/O Symbolic Name Table instead of using the Module Definition Table, because you set the address of the I/O with dipswitches on the I/O card. Use the I/O Symbol Table to associate a controller memory address with each I/O symbolic name that identifies a physical I/O point wired to a module.

Availability

I/O Symbolic Name Tables are supported for Series 500™ I/O, Series 505 I/O, and S5 I/O. The Module Definition Table is only supported for Series 500 I/O and Series 505 I/O.

3.2 Defining Modules and I/O

Overview

The procedure you follow to configure your I/O points and modules differs depending on your controller type.

For Series 505 controllers, APT allows you to define modules and I/O in either of the following orders:

- Configure Modules First. This is a one-step process: you define an I/O name in the Module Definition Table, and that name automatically appears in the I/O Symbolic Name Table as the default type. See Figure 3-3.

  Configuring modules first is especially useful for intelligent I/O modules, because APT automatically creates appropriate I/O names and identifies I/O points that have specific purposes.

- Define I/O First. This is a two-step process: you define the symbolic names and then configure the modules. The I/O names are not linked with the physical addresses until you define the I/O name in the Module Definition Table. See Figure 3-4.

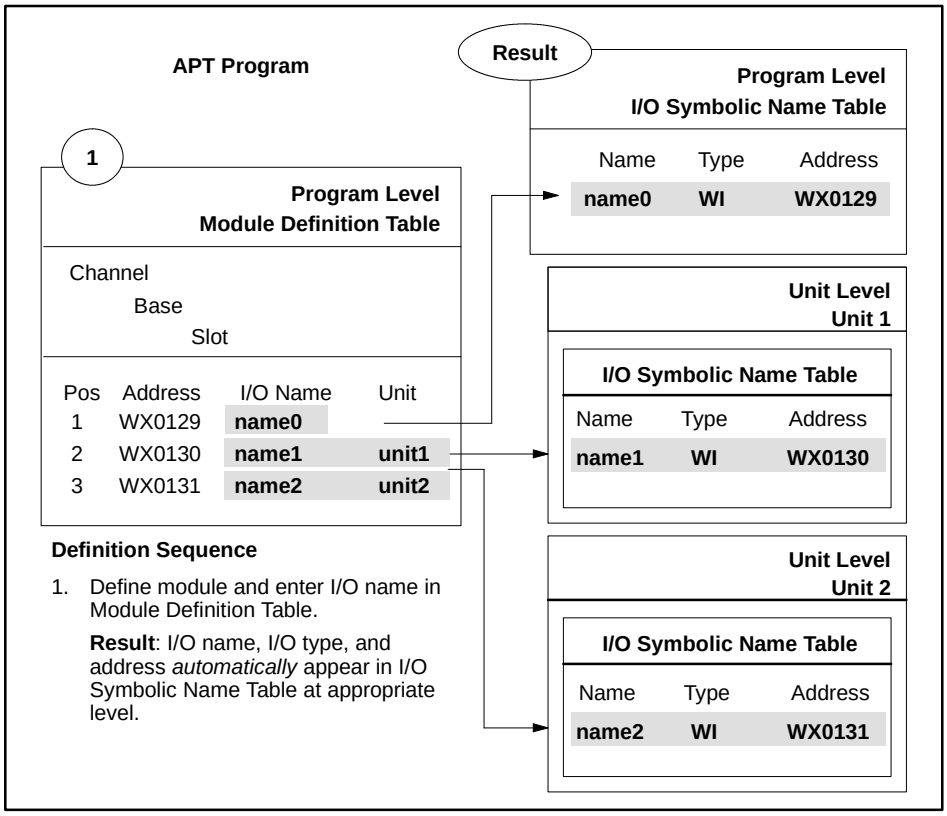


Figure 3-3 Defining Series 505 Modules Before I/O Symbolic Names

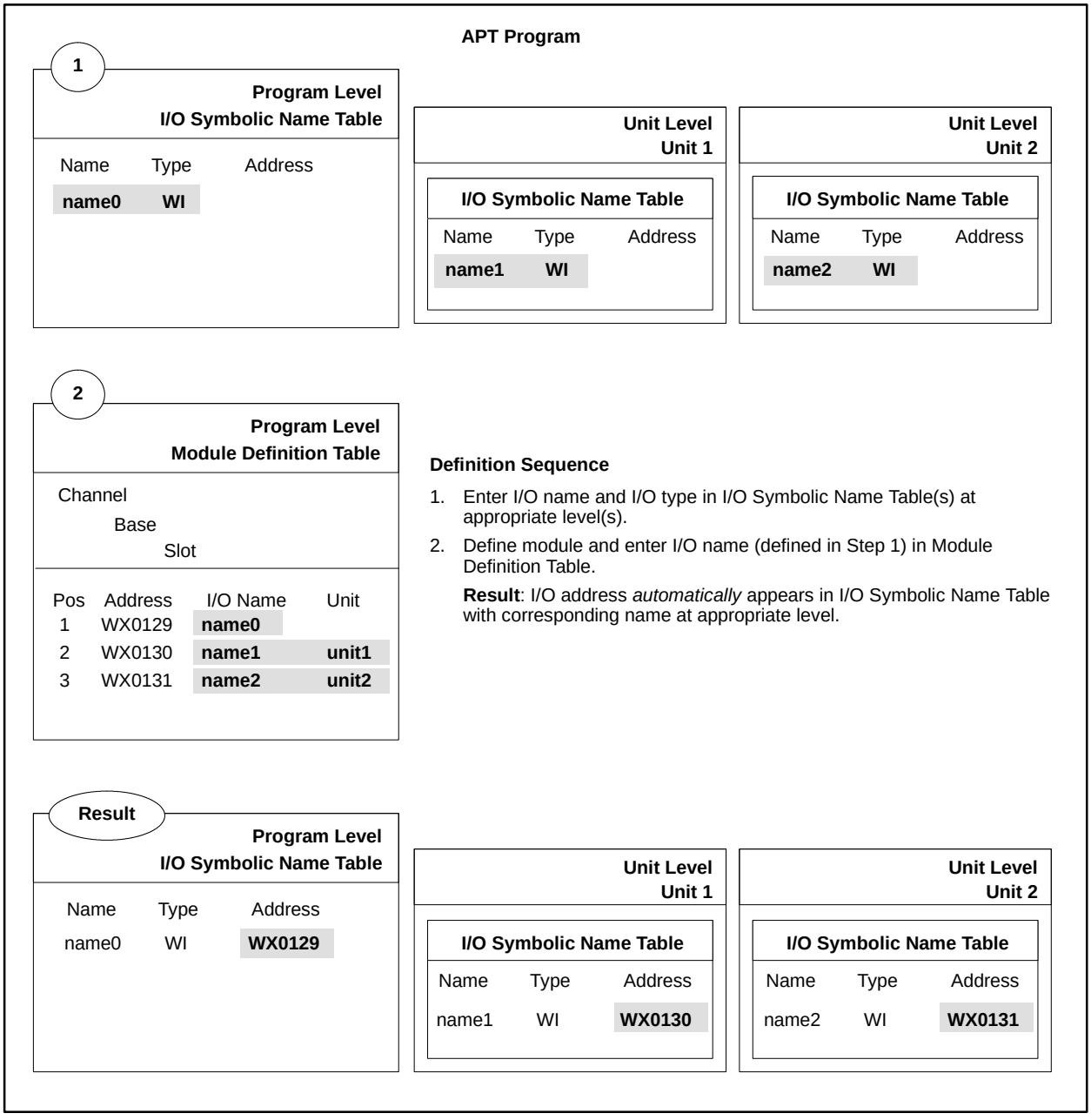


Figure 3-4 Defining I/O Symbolic Names Before Series 505 Modules

For S5 controllers, you use only the I/O Symbolic Name Table, and not the Module Definition Table, so it is a one-step process to link the symbolic name to the physical address of your I/O points. See Figure 3-5.

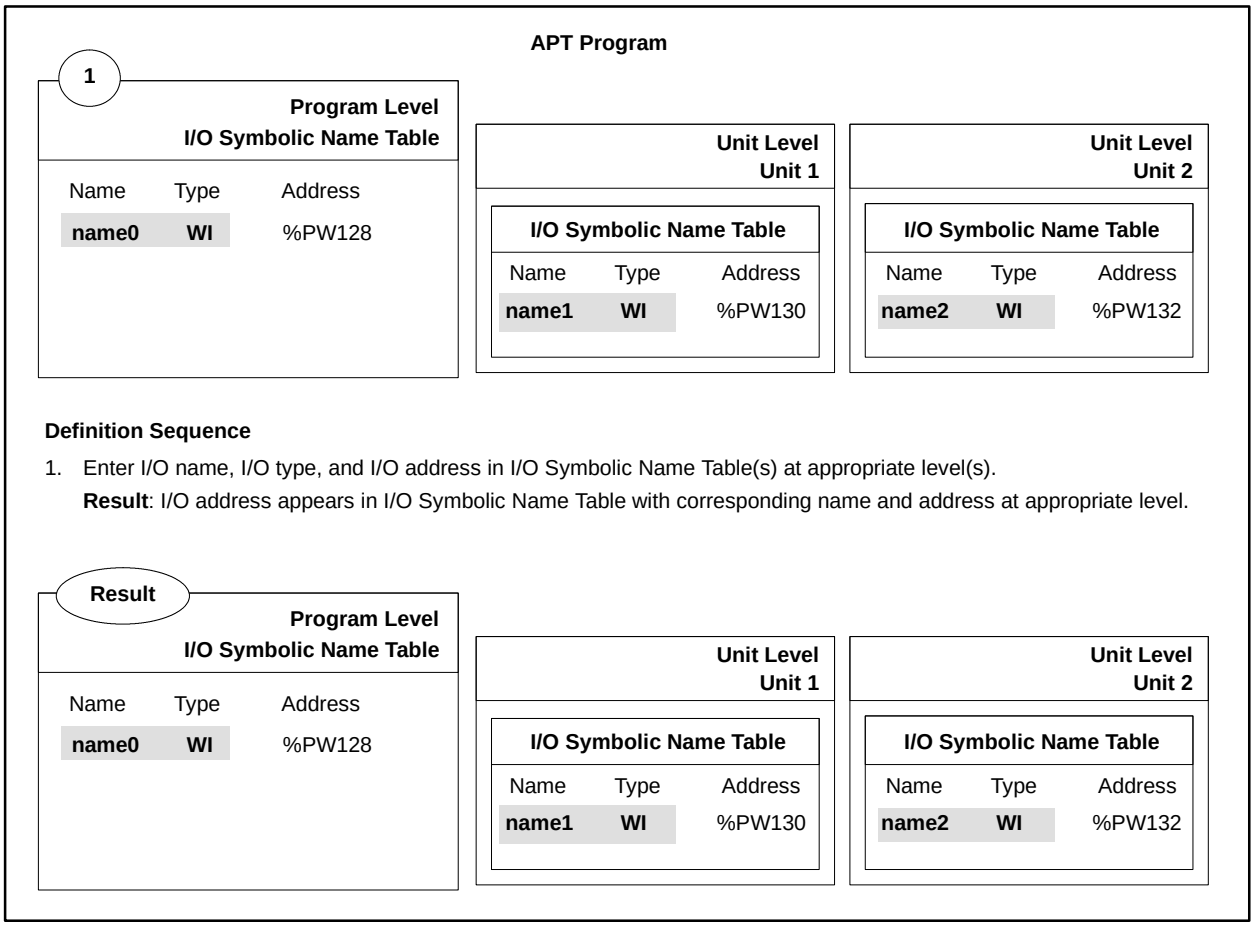


Figure 3-5 Defining I/O Symbolic Names for S5 Controllers

Defining I/O Symbolic Names

The I/O Symbolic Name Table identifies all I/O symbolic names in an APT program. These names may or may not be associated with an actual physical I/O point. For S5 controllers, you can test your program when your I/O is not physically connected by selecting **Image Register: (X)** in the I/O Symbolic Name Table. Series 505 controllers get the image register automatically.

Table 3-1 lists the I/O code for each I/O type available in APT for Series 505 controllers. Table 3-2 lists the I/O code for each I/O type available in APT for S5 controllers. Some types of I/O require additional information such as high and low ranges, or offer options such as filtering and square root. Each I/O type is described in detail in Chapter 5, which also shows the form that you use to define each type.

Table 3-1 I/O Symbolic Name Types for Series 505 Controllers

Type of I/O Point	Code	Value Type	Modules Supported	Page Reference
Analog Input	AI	Real [1]	4AI, 8AI, 4AI4O, 6AI2O, 8AI4O, 16AI, 2AI40	5-6
Analog Output	AO	Integer	2AO, 4AO, 8AO, 4AI4O, 6AI2O, 8AI4O, 2AI4O	5-11
Binary-coded Decimal Input	BI	Integer	8WI	5-20
Binary-coded Decimal Output	BO	Integer	8WO, 8WOSF	5-22
Digital Flag	DF	Boolean	8DO, 16DO, 32DO	5-14
Digital Input	DI	Boolean	8DI, 16DI, 32DI	5-16
Digital Output	DO	Boolean	8DO, 16DO, 32DO	5-17
Resistance Temperature Detector	RT	Real [1]	RT	5-24
Thermocouple	TC	Real [1]	TC	5-27
Word Input	WI	Integer	8WI, 16WI	5-18
Word Output	WO	Integer	8WO, 16WO, 8WOSF	5-19

1 These values appear in integer form for 560/560T controllers.

Table 3-2 lists the code for each I/O type available in APT for S5 controllers.

Table 3-2 I/O Symbolic Name Types for S5 Controllers

Type of I/O Point	Code	Value Type	Modules Supported [1]	Page Reference
Analog Input	AI	Real	460, 465, 466 (not 463)	5-6
Analog Output	AO	Integer	470	5-11
Binary-coded Decimal Input	BI	Integer	420, 430, 431, 432, 434, 435, 436	5-20
Binary-coded Decimal Output	BO	Integer	420, 430, 431, 432, 434, 435, 436	5-22
Digital Flag [2]	DF	Boolean	441, 451, 453, 454, 456, 457, 458, 482, 498	5-14
Digital Input	DI	Boolean	420, 430, 431, 432, 434, 435, 436	5-16
Digital Output	DO	Boolean	441, 451, 453, 454, 456, 457, 458, 482, 498	5-17
Word Input	WI	Integer	420, 430, 431, 432, 434, 435, 436	5-18
Word Output	WO	Integer	441, 451, 453, 454, 456, 457, 458, 482, 498	5-19

1 Listed are the part numbers for the modules supported. The prefix to all of the part numbers is 6ES5.
2 An APT digital flag includes several interrelated logical statements that control an external bit. It is different from an S5 flag.

Defining Modules for Series 505

For Series 505 controllers, you use the Module Definition Table to identify all I/O modules that you use in your control system. Table 3-3 lists the code for each module type along with the associated I/O types and other requirements. Each Series 505 module type is described in detail in Chapter 4, which also shows the form that you use in identifying the physical address for the I/O points in your system.

Table 3-3 Module Types for Series 505 Controllers

Module	I/O Series	Code	No. I/O Points	Memory Type	I/O Name Type	No. Slots Required†	Page Reference
Standard Modules							
2-Ch. Analog Output	500	2AO	2	WY	AO	–	4-9
	505	2AO	2	WY	AO	–	4-10
4-Ch. Analog Input	500	4AI	4	WX	AI	–	4-11
4-Ch. Analog Output	505	4AO	4	WY	AO	–	4-12
4-Ch. AI/4-Ch. AO	505	4AI4O	4 4	WY WY	AI AO	–	4-13
6-Ch. AI/2-Ch. AO	505	6AI20	6 2	WX WY	AI AO	–	4-14
8-Ch. AI/4-Ch. AO	505	8AI4O	12 4	WX WY	AI AO	–	4-15
8-Ch. Analog Input	500	8AI	8	WX	AI	–	4-16
	505	8AI	8	WX	AI	–	4-17
8-Ch. Analog Output	500	8AO	8	WY	AO	–	4-18
	505	8AO	8	WY	AO	–	4-19
Parallel Word Input	500	8WI	8	WX	WI	–	4-20
	505	8WI	8	WX	WI	–	4-21
	505	16WI	16	WX	WI	–	4-22
Parallel Word Output	500	8WO	8	WY	WO	–	4-23
	505	8WO	8	WY	WO	–	4-24
	505	16WO	16	WY	WO	–	4-25
Parallel Word Out (SF)	505	8WOSF	8	WY	WO	–	4-26
8-Ch. Discrete Input	500	8DI	8	X	DI	*	4-27
	505	8DI	8	X	DI	*	4-28
8-Ch. Discrete Output	500	8DO	8	Y	DO	*	4-29
	505	8DO	8	Y	DO	*	4-30
16-Ch. Analog Input	505	16AI	16	WX	AI	–	4-31
16-Ch. Discrete Input	505	16DI	16	X	DI	*	4-32
16-Ch. Discrete Output	505	16DO	16	Y	DO	*	4-33
20-Ch. AI/4-Ch. AO	505	2AI4O	20 4	WX WY	AI AO	–	4-34

† The slot width for all modules is user configurable.

* These modules require one or two slots, depending upon the voltage.

Table 3-3 Module Types for Series 505 Controllers (continued)

Module	I/O Series	Code	No. I/O Points	Memory Type	I/O Name Type	No. Slots Required†	Page Reference
Standard Modules							
32-Ch. Discrete Input	500	32DI	32	X	DI	*	4-35
	505	32DI	32	X	DI	*	4-36
32-Ch. Discrete Output	500	32DO	32	Y	DO	*	4-37
	505	32DO	32	Y	DO	*	4-38
Smartslice	505	SLICE	16 8	X Y	DI DO	-	4-39
User Defined Modules	500	USER	0-64 0-64	X,Y** WX, WY**	DI DO	-	4-40
User Defined Modules	505	USER	0-64 0-64	X,Y** WX,WY**	WI WO	-	4-42
Interrupt Module	505	INTRP	24 8	X Y	DI DO	-	4-44
Intelligent Modules							
ASCII Message Output	500	ASCII	7 1	WX WY	WI WO	-	4-46
Programmable BASIC	500	BASIC	4 4	WX WY	WI WO	-	4-48
	505	BASIC	4 4	WX WY	WI WO	-	4-49
Dual Comm. Port	500	DCOMM	7 1	WX WY	WI WO	-	4-50
High-Speed Pulse Input	500	HSPI	8 5 3	WY WX Y	WO WI DO	-	4-52
High-Speed Counter	505	HSC	3 5	WX WO	WI WO	-	4-54
TIWAY Interface	500	NIM	8	Y	DO	-	4-55
	505	NIM	8	Y	DO	-	4-56
Peerlink	500	PLINK	3 5	WX WY	WI WO	-	4-57
	505	PLINK	3 5	WX WY	WI WO	-	4-59
RTD	500	RTD	16	WX	RT	-	4-61
	505	8RTD	8	WX	RT	-	4-62
	505	16RTD	16	WX	RT	-	4-62
Servo Axis	500	SERVO	4 4	WX WY	WI WO	-	4-63
Thermocouple	500	TC	8	WX	TC	-	4-64
	505	TC	8	WX	TC	-	4-65

† The slot width for all modules is user configurable.
* These modules require one or two slots, depending upon the voltage.
** Total count must be less than or equal to 64 locations. The mix must be I/O symbolic names.

Table 3-3 Module Types for Series 505 Controllers (continued)

Module	I/O Series	Code	No. I/O Points	Memory Type	I/O Name Type	No. Slots Required†	Page Reference
Intelligent Modules (continued)							
386/ATM Coprocessor	505	ATM	4 4	WX WY	WI WO	-	4-66
Expert Solutions Processor	500	ESP	4 4	WX WY	WI WO	-	4-67
TurboMold	500	TURBO	4 4	WX WY	WI WO	-	4-68
	505	TURBO	4 4	WX WY	WI WO	-	4-69
High-Speed PID Ctrl.	500	HSPID	4 4	WX WY	WI WO	-	4-70
110 VAC Redun. Out.	500	110VO	3 5	X Y	DI DO	-	4-71
110 VAC Rapid Resp.	500	110VR	2 6	X Y	DI DO	-	4-72
120 VDC Rapid Resp.	500	120VR	2 4	X Y	DI DO	-	4-73
24 VDC Rapid Resp.	500	24VDC	2 6	X Y	DI DO	-	4-74
FIM (SIMOREG Mode)	505	SREG	10 10	WX WY	WI WO	-	4-76
FIM (SIMOREG Broadcast Mode)	505	SREGB	16 16 10	X Y WY	DI DO WO	-	4-78
FIM (SIMOVERT Mode)	505	SVRT	4 4	WX WY	WI WO	-	4-80
FIM (SIMOVERT Broadcast Mode)	505	SVRTB	16 16 4	X Y WY	DI DO WO	-	4-82
Communications Processor	505	H1	4 4	WX WY	WI WO	-	4-84
High Density Advanced Function	505	16AF	16 16 32 32	X Y WX WY	DI DO WI WO	-	4-85
Ethernet TCP/IP Adaptor	505	ENET	2 6	WX WY	WI WO	-	4-87
High Speed Counter Encoder	505	HSCE	18 14	WX WY	WI WO	-	4-88
Program Port Expander	505	PPEXP	2 6	WX WY	WI WO	-	4-90

† The slot width for all modules is user configurable.

Chapter 4

Series 500/505 Modules

4.1 Module Definition Procedure

Overview

The APT Module Definition Utility lets you configure the wiring of the controller hardware to I/O points by specifying a channel, base, slot, type, and starting address. It also lets you assign each controller address (physical address) on that module to an I/O symbolic name.

Availability

The APT Module Definition Utility is available for Series 500 I/O and Series 505 I/O only. It is not available for S5 I/O.

Each I/O point can be associated with only one controller address, but all points do not have to have a symbolic name. Once you have defined a symbolic name and assigned it an address, you can refer to that I/O point by either its symbolic name or by its associated controller address. However, if you use the physical address in your program and later move the module, you have to change those addresses in the program.

The process of defining a module (Figure 4-1) is a two-step procedure.

- Select a channel, base, and slot, and specify a module type and starting address.
- Assign an I/O symbolic name to each address, and specify a unit and description if necessary.

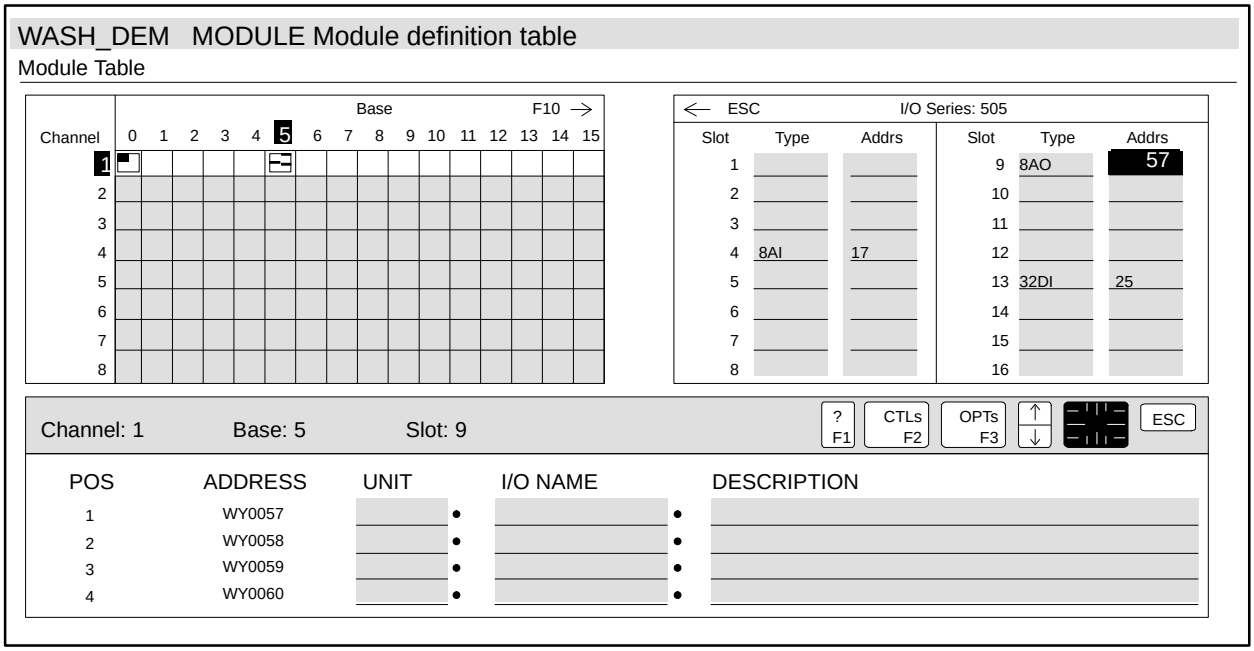


Figure 4-1 Defining Modules

Assigning Addresses

The first step in defining a module is to select the channel, base, and slot to specify a module type and starting address. The input windows for this step are illustrated in Figure 4-2. Follow the editing procedures described in the chapter on using table definition utilities in the *SIMATIC APT User Manual* to select a channel, base, and slot to specify the module type and the starting address.

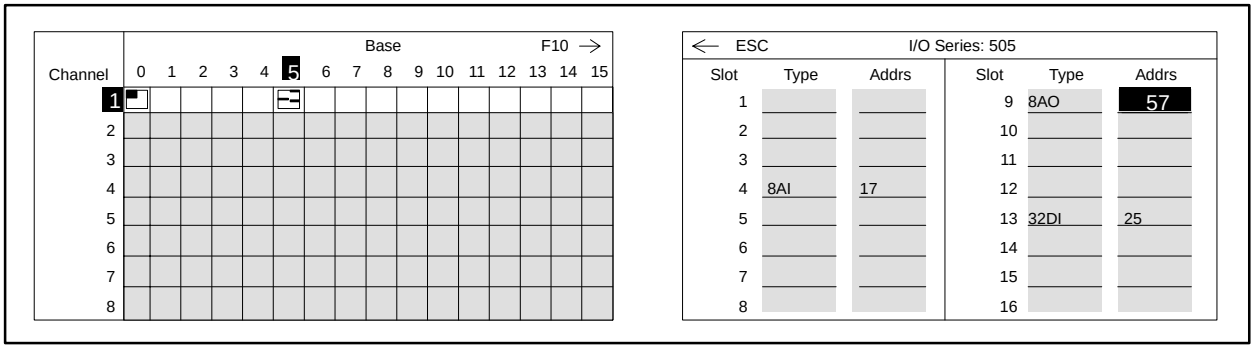


Figure 4-2 Assigning an Address to a Module

After you select a channel, base, and slot and specify the module type, APT shows the default starting address. The default is the next available address that has a sufficient number of addresses for that module. If you change the default, the address must be on an eight-point boundary (1, 9, 17, etc.) and in the range for that channel.

⚠ WARNING

APT assumes that the field I/O matches the configuration of the Module Table, including slot size. A mismatch between your field I/O and your Module Table configuration could cause unpredictable operation by the controller.

Unpredictable operation by the controller after you download an APT program could result in death or serious injury and/or equipment damage.

Verify that the field I/O matches the APT configuration before placing the controller in RUN mode. This verification must be done as part of any actions carried out prior to system commissioning. The Verify operation, accessed from the hierarchy, and the AUX functions, accessed from Debug, warn you of I/O mismatches. Refer to the "Compiling an APT Program" and the "Debugging an APT Program" chapters in the *SIMATIC APT User Manual* for more information about these operations.

The address ranges for each channel for the 560, 560T, 565, and 565P controllers are listed in Table 4-1.

Table 4-1 Channel Address Ranges: 560/560T/565/565P/565T

Channel	Address Range	Channel	Address Range
1	1 – 1024	5	4097 – 5120
2	1025 – 2048	6	5121 – 6144
3	2049 – 3072	7	6145 – 7168
4	3073 – 4096	8	7169 – 8192

Table 4-2 lists the address ranges for Channel 1 for the 545, 545L, 555, and 575.

Table 4-2 Channel Address Ranges: 545/545L/555/575

Controller	Channel	Address Range
545 (Rel. 2.1 and higher)	1	2048 I/O points: 2048 maximum; up to 1024 may be analog or word points, which must be numbered 1–1024.
545L	1	I/O points: 1024 maximum in any mix.
555	1	I/O points: 8192 maximum in any mix.
575 (Rel. 3.0 and higher)	1	I/O points: 8192 maximum in any mix.

After you indicate the starting address, APT automatically assigns the remaining points in the module to the next available sequential addresses.

You can change the I/O series from Series 500 to Series 505, and vice versa. APT reports any incompatible modules when you select this option and prompts you about deleting them. If you select **Yes,** the modules are removed from the table. If you select **No**, the I/O series change is aborted and no changes are made in the channel/base status.

You can change the slot width for a module if the APT default for the module is not correct.

Some modules are double-wide and require two I/O slots. You may be able to save a slot in a Series 500 base by placing a double-wide module in the last slot of the base and allowing it to extend beyond the base.

Assigning I/O Names

The Module Definition Sub-Editor allows you to assign I/O symbolic names to each controller address. The I/O names in a module must be unique to the program or unit. If you do not enter the name of the unit, APT assumes that the I/O point is global and exists at the Program Content Level.

For standard modules, a form appears like the one shown in Figure 4-3.

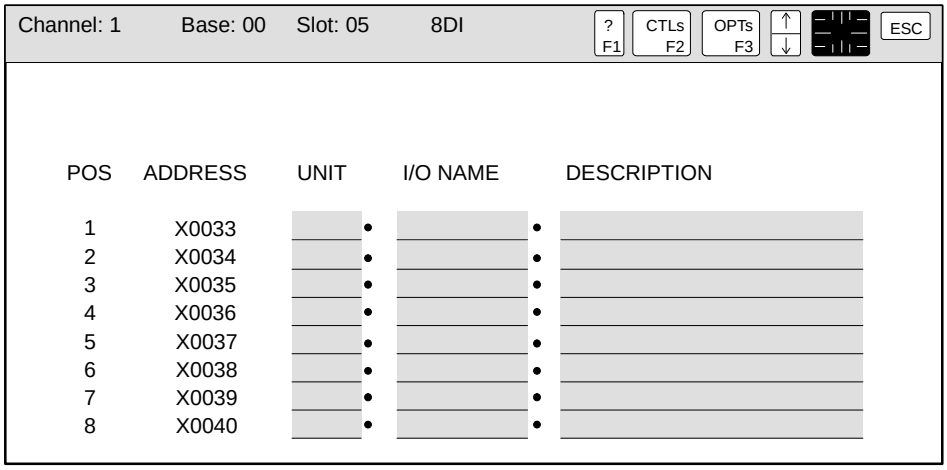


Figure 4-3 Assigning I/O Names

If you have previously defined the I/O point in the I/O Symbolic Name Definition Table, you only need to enter the I/O name and unit (if the point is not global). The completion aids (•) allow you to pick a unit and I/O point within the unit defined. When you exit from the Module Definition Sub-Editor, APT places the description from the I/O Symbolic Name Table into the Description field of the Module Definition Sub-Editor.

After you enter the I/O points in the Module Definition Sub-Editor, the I/O symbolic name and all related information are automatically inserted in the I/O Symbolic Name Table as shown in Figure 4-4.

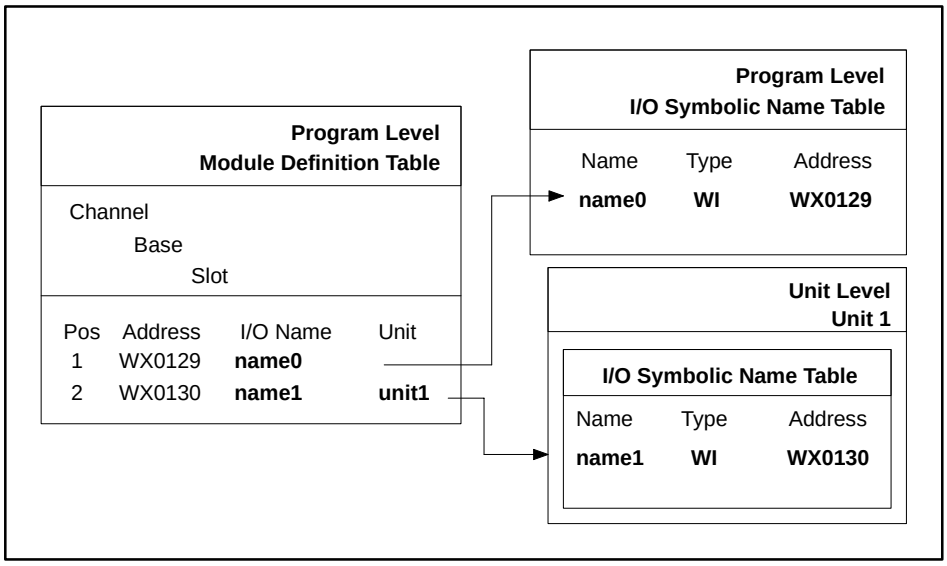


Figure 4-4 Getting I/O Names from Module Table

Most of the intelligent modules require two forms for configuration.

- The first form appears the first time you edit the module and allows you to enter a default prefix for the I/O names and the unit.
- If you enter the default prefix (maximum five characters), APT automatically creates I/O symbolic names that appear in the second form. These default names are also inserted automatically in the I/O Symbolic Name Table at the required unit.

4.2 Standard Modules

2-Channel Analog Output (Series 500: 2AO)

The Series 500 2-Channel Analog Output (2AO) Module is a differential, 12-bit digital-to-analog converter that has two user-selectable voltage ranges per channel. The ranges are bipolar (–10 to 10) and unipolar (0 to 10). Regardless of the voltage range you select, the current output from each channel is always 0 to 20 milliamps.

Figure 4-5 shows the form that you use to define a Series 500 2-Channel Analog Output Module.

Slots required:	1	(user-configurable)
Address type:	X	(6 APT-reserved locations)
	WY	(2 consecutive locations)
I/O symbolic name type:	AO	(analog output)

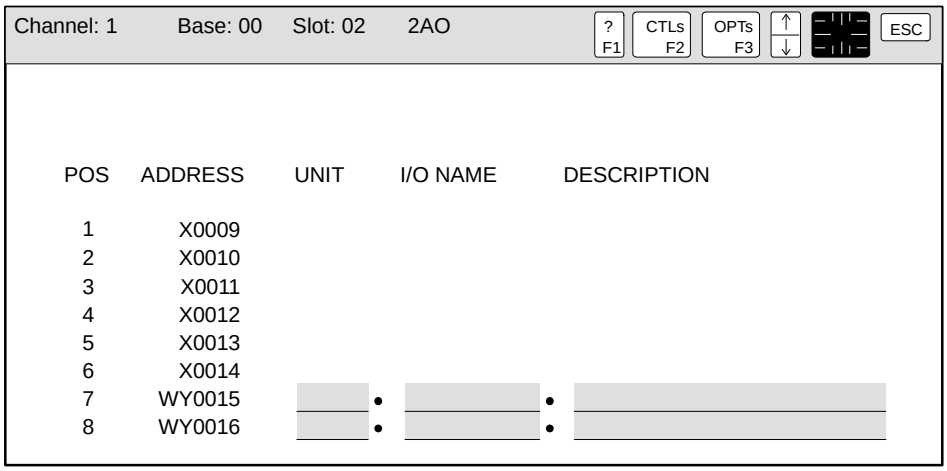


Figure 4-5 Series 500 2-Channel Analog Output Module

Standard Modules (continued)

2-Channel Analog Output (Series 505:2AO)

The Series 505 2-Channel Analog Output (2AO) Module is a differential, 12-bit digital-to-analog converter that has two user-selectable voltage ranges per channel. The ranges are bipolar (–10 to 10) and unipolar (0 to 10). Regardless of the voltage range you select, the current output from each channel is always 0 to 20 milliamps.

Figure 4-6 shows the form that you use to define a Series 505 2-Channel Analog Output Module.

Slots required:	2	(user-configurable)
Address type:	WY	(2 consecutive locations)
	WY	(6 APT-reserved locations)
I/O symbolic name type:	AO	(analog output)

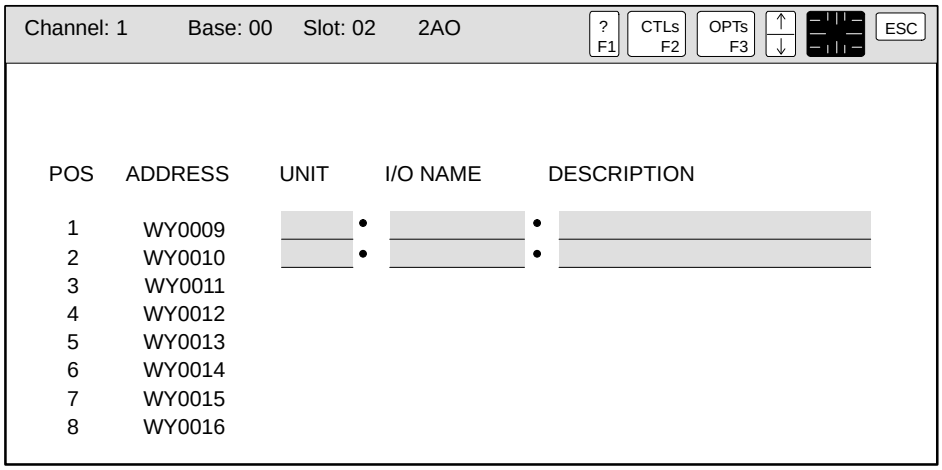


Figure 4-6 Series 505 2-Channel Analog Output Module

4-Channel Analog Input (Series 500: 4AI)

The Series 500 4-Channel Analog Input (4AI) Module is a differential, 12-bit analog-to-digital converter that accepts bipolar (–10 to 10 or –5 to 5 volts) or unipolar (0 to 10 or 0 to 5 volts) signals and converts them to the equivalent integer representation for the controller. The data sheet for the module contains details of the electrical characteristics of the module.

Figure 4-7 shows the form that you use to define a Series 500 4-Channel Analog Input Module.

Slots required:	1	(user-configurable)
Address type:	WX	(4 consecutive locations)
	Y	(4 APT-reserved locations)
I/O symbolic name type:	AI	(analog input)

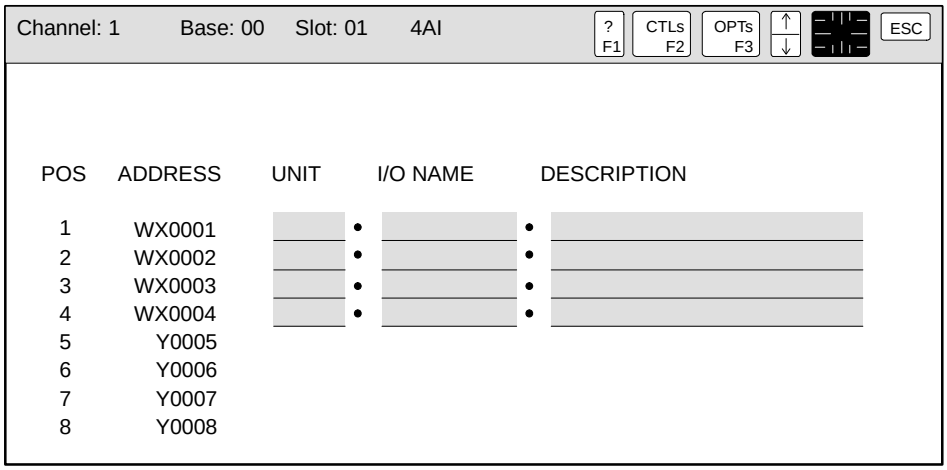


Figure 4-7 Series 500 4-Channel Analog Input Module

Standard Modules (continued)

4-Channel Analog Output (Series 505: 4AO)

The Series 505 4-Channel Analog Output (4AO) Module is a differential, 12-bit digital-to-analog converter that has four user-selectable voltage ranges per channel. The ranges are bipolar (–10 to 10) and unipolar (0 to 10). Regardless of the voltage range you select, the current output from each channel is always 0 to 20 milliamps.

Figure 4-8 shows the form that you use to define a Series 505 4-Channel Analog Output Module.

Slots required:	2	(user-configurable)
Address type:	WY	(4 consecutive locations)
	WY	(4 APT-reserved locations)
I/O symbolic name type:	AO	(analog output)

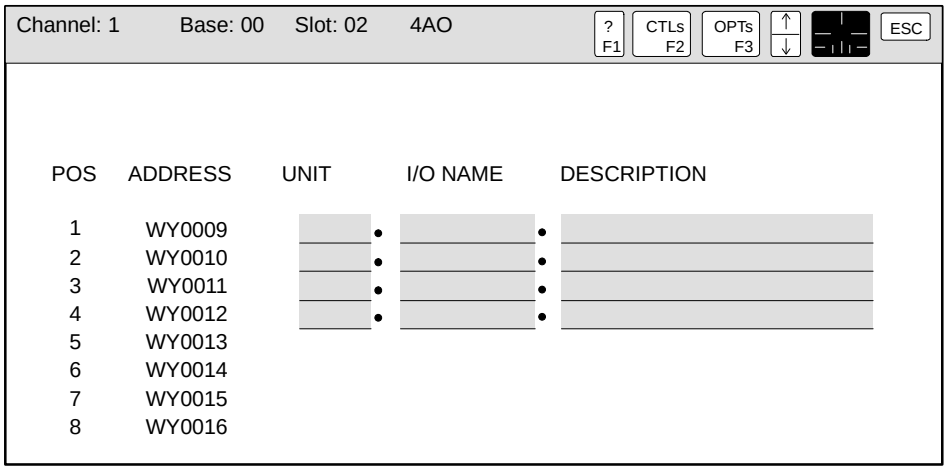


Figure 4-8 Series 505 4-Channel Analog Output Module

4-Channel Analog Input/4-Channel Analog Output (Series 505: 4AI4O)

The Series 505 4-Channel Analog Input/4-Channel Analog Output (4AI4O) Module combines 12-bit analog-to-digital converters and 12-bit digital-to-analog converters into one unit. Refer to the user manual for this module for details about the electrical characteristics of this module.

Figure 4-9 shows the form that you use to define a Series 505 4-Channel Analog Input/4-Channel Analog Output Module.

Slots required:	1	(user-configurable)
Address type:	WX	(4 consecutive locations)
	WY	(4 consecutive locations)
I/O symbolic name type:	AI	(analog input)
	AO	(analog output)

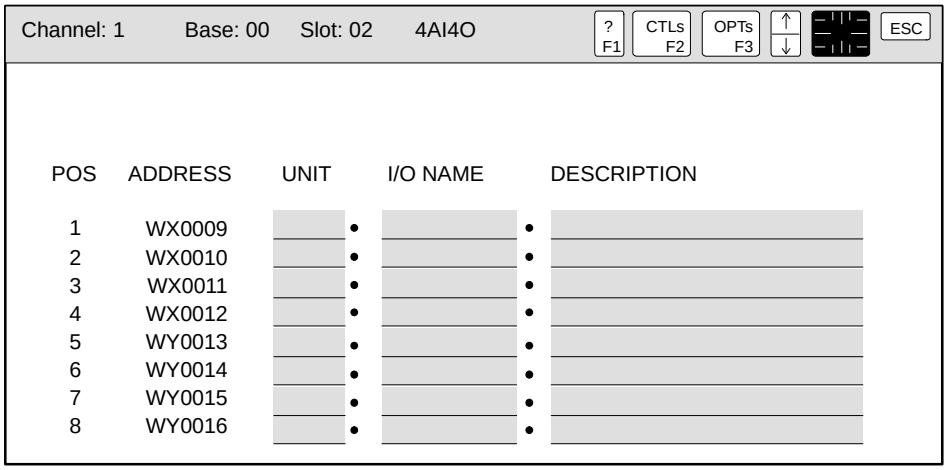


Figure 4-9 Series 505 4-Channel Analog Input/4-Channel Analog Output Module

Standard Modules (continued)

6-Channel Analog Input/2-Channel Analog Output (Series 505: 6AI2O)

The Series 505 6-Channel Analog Input/2-Channel Analog Output (6AI2O) Module combines 12-bit analog-to-digital converters and 12-bit digital-to-analog converters into one unit. Refer to the user manual for this module for details about the electrical characteristics of this module.

Figure 4-10 shows the form that you use to define a Series 505 6-Channel Analog Input/2-Channel Analog Output Module.

Slots required:	1	(user-configurable)
Address type:	WX	(6 consecutive locations)
	WY	(2 consecutive locations)
I/O symbolic name type:	AI	(analog input)
	AO	(analog output)

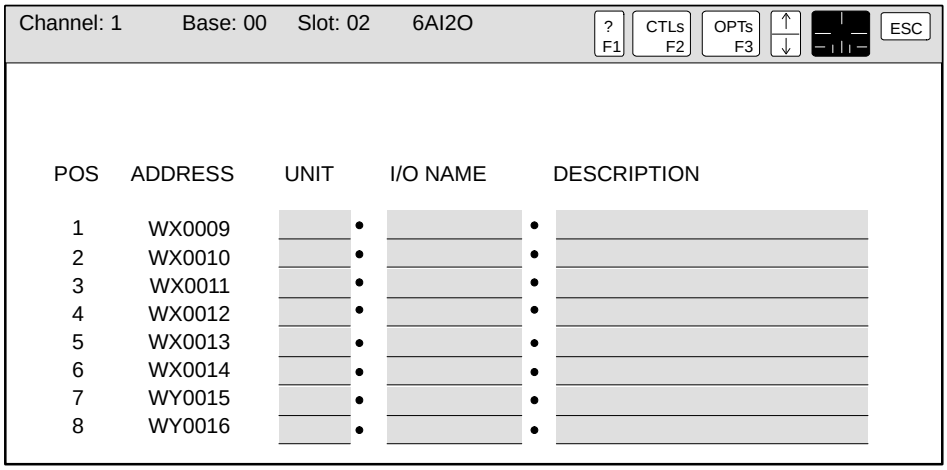


Figure 4-10 Series 505 6-Channel Analog Input/2-Channel Analog Output Module

8-Channel Analog Input/4-Channel Analog Output (Series 505: 8AI4O)

The Series 505 8-Channel Analog Input/4-Channel Analog Output (8AI4O) Module combines 12-bit analog-to-digital converters and 12-bit digital-to-analog converters into one unit. Refer to the user manual for this module for details about the electrical characteristics of this module.

Figure 4-11 shows the form that you use to define a Series 505 8-Channel Analog Input/4-Channel Analog Output Module.

Slots required:	1	(user-configurable)
Address type:	WX	(12 consecutive locations)
	WY	(4 consecutive locations)
I/O symbolic name type:	AI	(analog input)
	AO	(analog output)

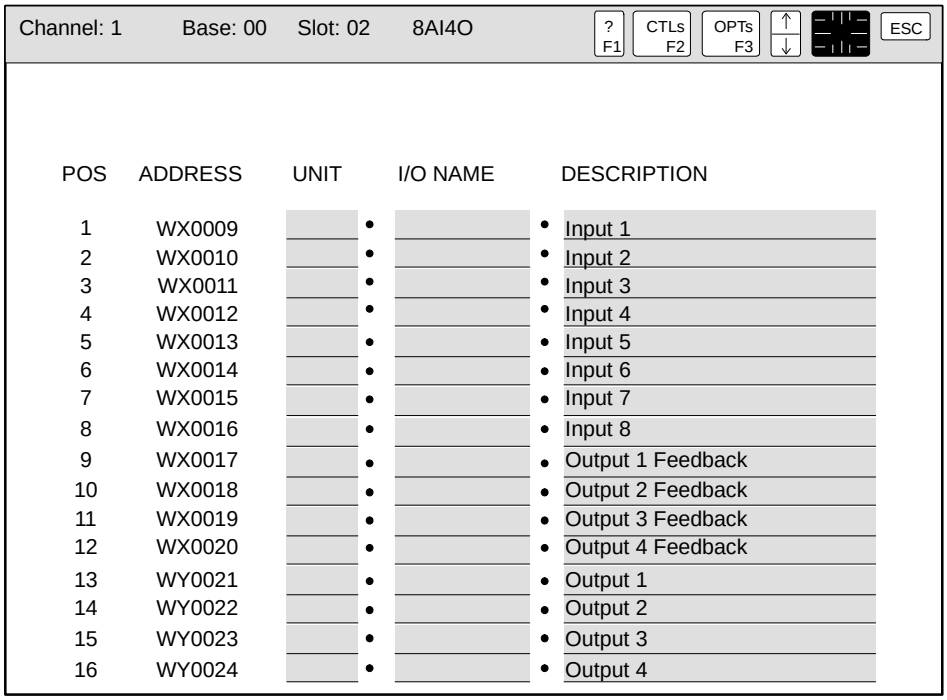


Figure 4-11 Series 505 4-Channel Analog Input/4-Channel Analog Output Module

Standard Modules (continued)

8-Channel Analog Input (Series 500: 8AI)

The Series 500 8-Channel Analog Input (8AI) Module is a single-ended, 12-bit analog-to-digital converter. The module accepts eight analog inputs. You can select current or voltage at the terminal strip on the module. The voltage input range is –5 to +5 volts; the current input range is 0 to 20 milliamps (across a 250-ohm resistor). The user manual for the module contains more details.

Figure 4-12 shows the form that you use to define a Series 500 8-Channel Analog Input Module.

Slots required:	1	(user-configurable)
Address type:	WX	(8 consecutive locations)
I/O symbolic name type:	AI	(analog input)

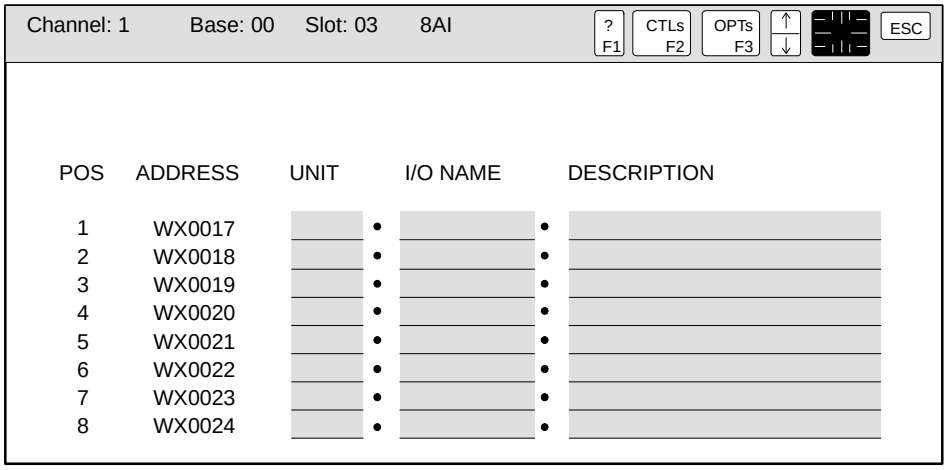


Figure 4-12 Series 500 8-Channel Analog Input Module

8-Channel Analog Input (Series 505: 8AI)

The Series 505 8-Channel Analog Input (8AI) Module is a single-ended, 12-bit analog-to-digital converter. The module accepts eight analog inputs. You can select current or voltage at the terminal strip on the module. The voltage input range is –5 to +5 volts; the current input range is 0 to 20 milliamps (across a 250-ohm resistor). The user manual for the module contains more details.

Figure 4-13 shows the form that you use to define a Series 505 8-Channel Analog Input Module.

Slots required:	2	(user-configurable)
Address type:	WX	(8 consecutive locations)
I/O symbolic name type:	AI	(analog input)

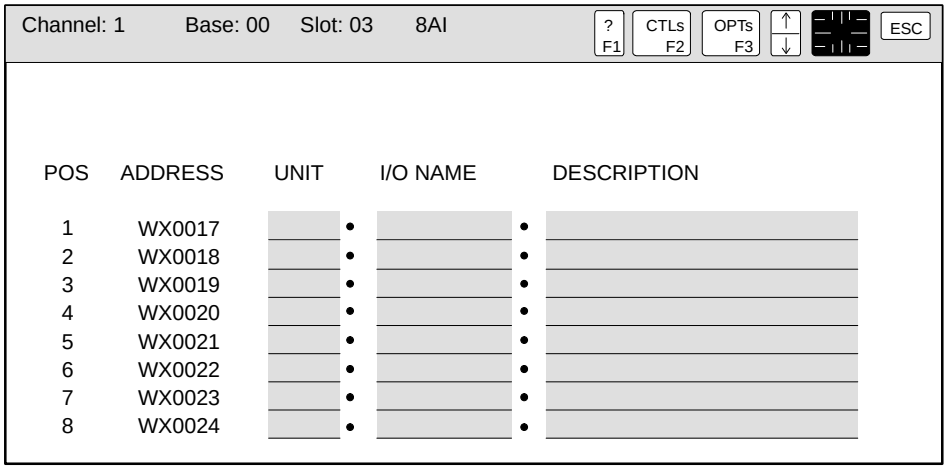


Figure 4-13 Series 505 8-Channel Analog Input Module

Standard Modules (continued)

8-Channel Analog Output (Series 500: 8AO)

The Series 500 8-Channel Analog Output (8AO) Module is a single-ended, 12-bit digital-to-analog converter. The output from the module is simultaneously available as a voltage (0 to 10 volts) and as a current (0 to 20 milliamps). Both outputs are on the terminal strip for each channel. The user manual for the module contains more details.

Figure 4-14 shows the form that you use to define a Series 500 8-Channel Analog Output Module.

Slots required:	1	(user-configurable)
Address type:	WY	(8 consecutive locations)
I/O symbolic name type:	AO	(analog output)

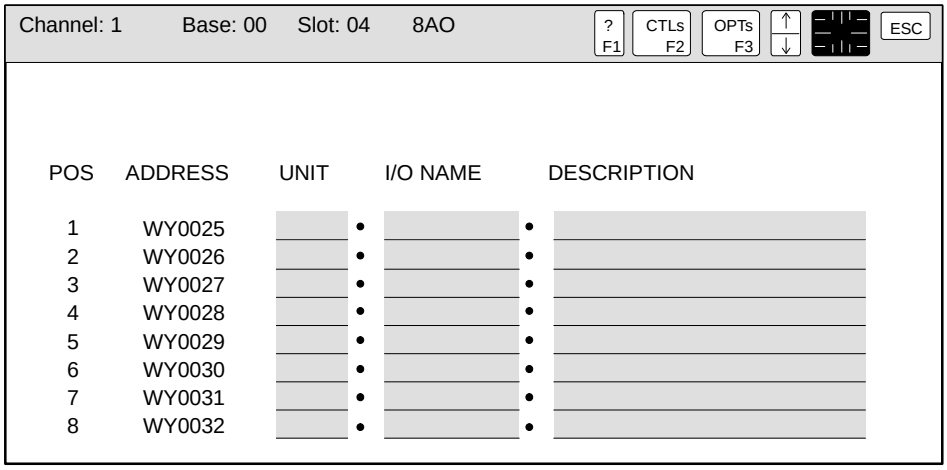


Figure 4-14 Series 500 8-Channel Analog Output Module

8-Channel Analog Output (Series 505: 8AO)

The Series 505 8-Channel Analog Output (8AO) Module is a single-ended, 12-bit digital-to-analog converter. The output from the module is simultaneously available as a voltage (0 to 10 volts) and as a current (0 to 20 milliamps). Both outputs are on the terminal strip for each channel. The user manual for the module contains more details.

Figure 4-15 shows the form that you use to define a Series 505 8-Channel Analog Output Module.

Slots required:	2	(user-configurable)
Address type:	WY	(8 consecutive locations)
I/O symbolic name type:	AO	(analog output)

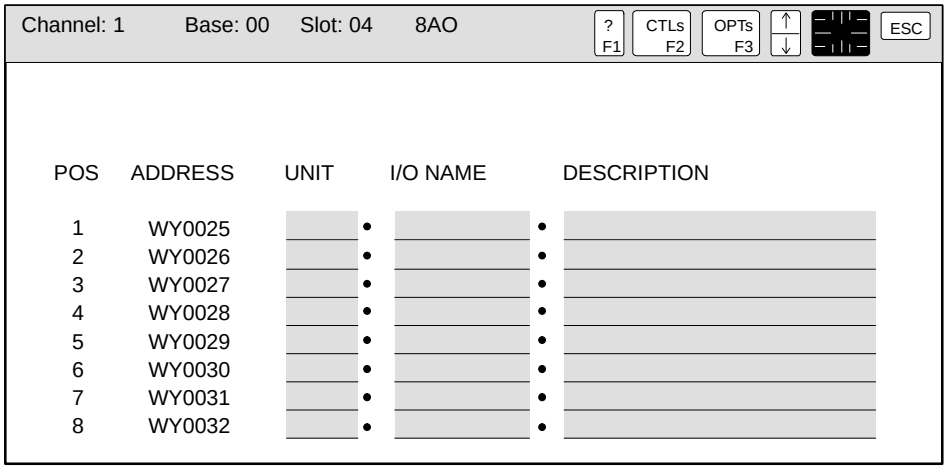


Figure 4-15 Series 505 8-Channel Analog Output Module

Standard Modules (continued)

Parallel Word Input (Series 500: 8WI)

The Series 500 Parallel Word Input (8WI) Module is an eight-channel (multiplexed) module that accepts the following types of input signals.

- TTL
- CMOS
- Up to 28 volts DC

Each channel provides a 16-bit data word that is available as a WX memory location. BCD (binary-coded decimal) thumbwheel switches are commonly connected to a Parallel Word Input Module.

Figure 4-16 shows the form that you use to define a Series 500 Parallel Word Input Module.

Slots required:	1	(user-configurable)
Address type:	WX	(8 consecutive locations)
I/O symbolic name type:	WI	(word input)*

* WI is the default I/O type, but you can change this to a binary-coded decimal input (BI) in the I/O Symbolic Name Table.

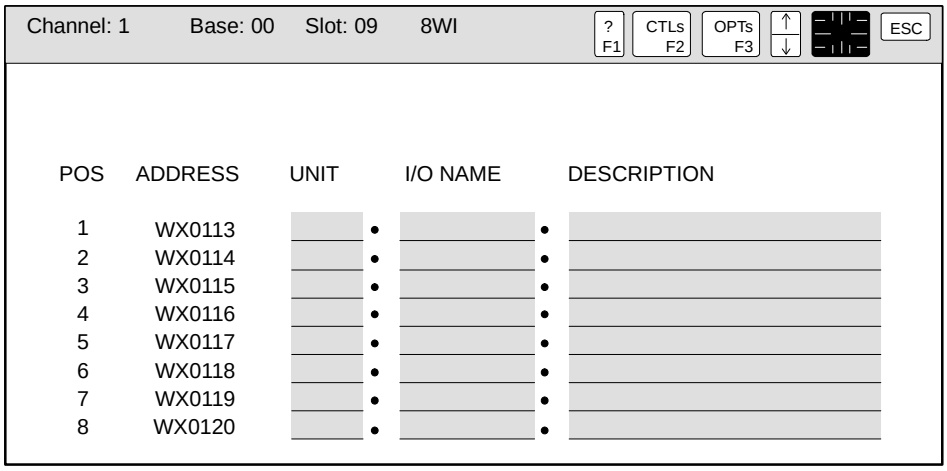


Figure 4-16 Series 500 8-Channel Parallel Word Input Module

Parallel Word Input (Series 505: 8WI)

The Series 505 Parallel Word Input (8WI) Module is an eight-channel (multiplexed) module that accepts the following types of input signals.

- TTL
- CMOS
- Up to 28 volts DC

Each channel provides a 16-bit data word that is available as a WX memory location. BCD (binary-coded decimal) thumbwheel switches are commonly connected to a Parallel Word Input Module.

Figure 4-17 shows the form that you use to define a Series 505 Parallel Word Input Module.

Slots required:	2	(user-configurable)
Address type:	WX	(8 consecutive locations)
I/O symbolic name type:	WI	(word input)*

* WI is the default I/O type, but you can change this to a binary-coded decimal input (BI) in the I/O Symbolic Name Table.

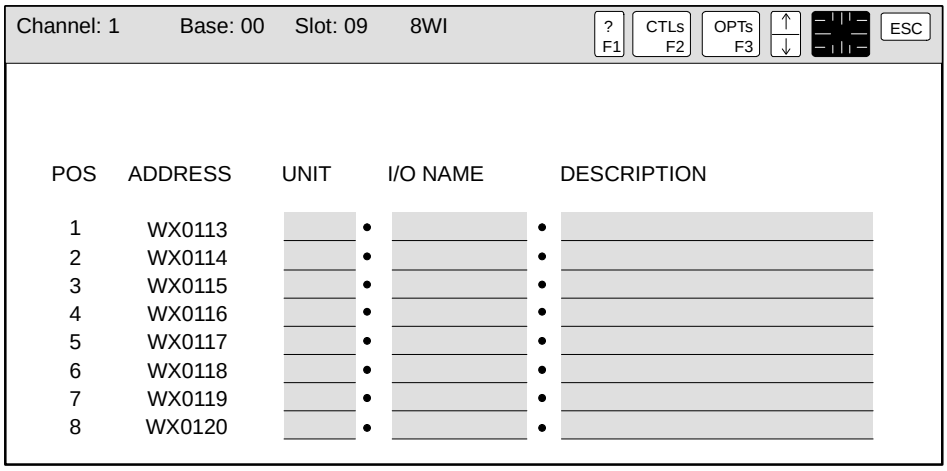


Figure 4-17 Series 505 8-Channel Parallel Word Input Module

Standard Modules (continued)

Parallel Word Input (Series 505: 16WI)

The Series 505-Compatible Parallel Word Input (16WI) Module is a 16-channel (multiplexed) input module. Each channel provides a 16-bit data word that is available as a WX memory location.

Figure 4-18 shows the form that you use to define a Series 505-compatible 16-point parallel word input module.

Slots required:	1	(user-configurable)
Address type:	WX	(16 consecutive locations)
I/O symbolic name type:	WI	(word input)*

* WI is the default I/O type, but you can change this to a binary-coded decimal input (BI) in the I/O Symbolic Name Table.

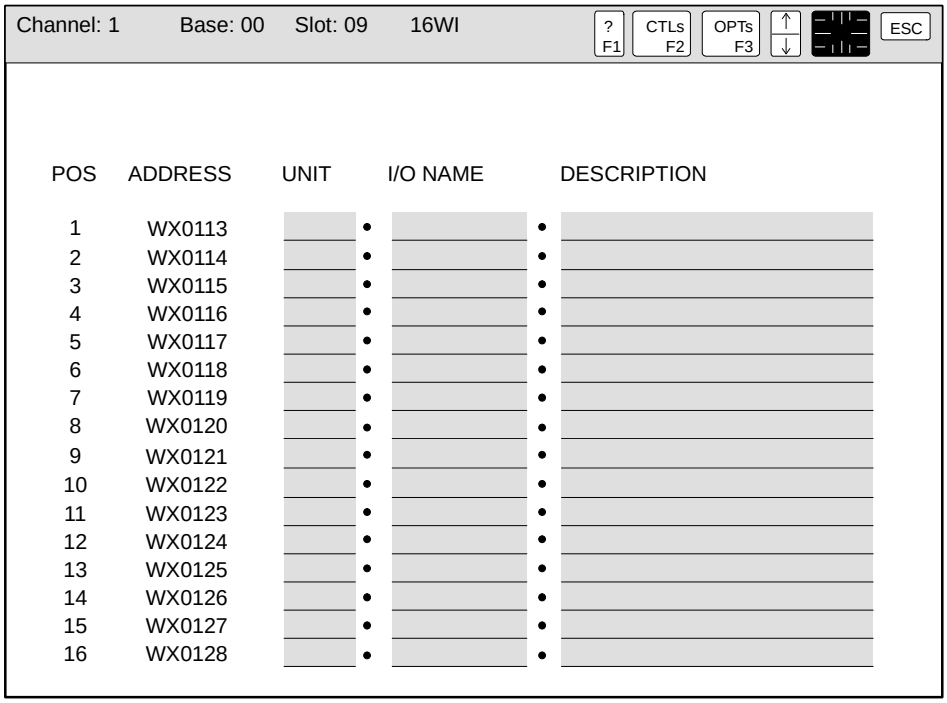


Figure 4-18 Series 505-Compatible 16-Channel Parallel Word Input Module

Parallel Word Output (Series 500: 8WO)

The Series 500 Parallel Word Output (8WO) Module is an eight-channel (multiplexed) module that outputs TTL or CMOS signals. Each channel provides a 16-bit data word that is available as a WY memory location. A BCD (binary-coded decimal) display is commonly connected to a Parallel Word Output Module.

Figure 4-19 shows the form that you use to define a Series 500 Parallel Word Output Module, which has the following configuration requirements.

Slots required:	1	(user-configurable)
Address type:	WY	(8 consecutive locations)
I/O symbolic name type:	WO	(word output)*

* WO is the default I/O type, but you can change this to a binary-coded decimal output (BO) in the I/O Symbolic Name Table.

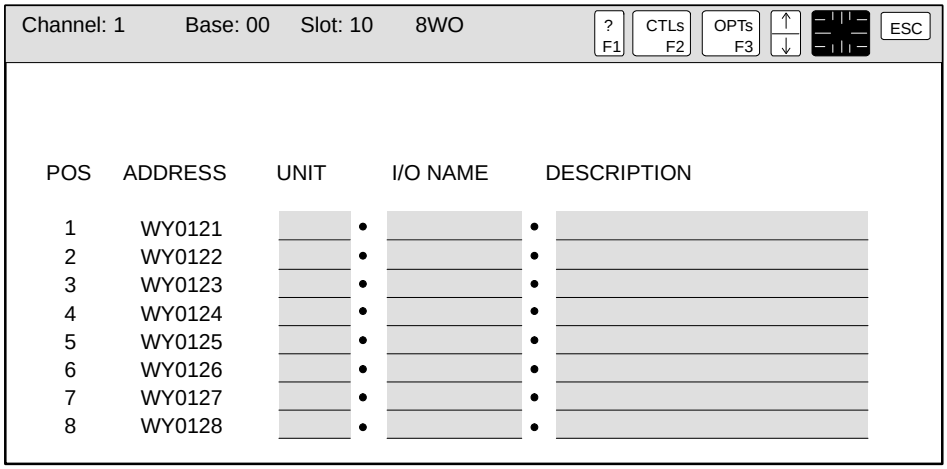


Figure 4-19 Series 500 8-Channel Parallel Word Output Module

Standard Modules (continued)

Parallel Word Output (Series 505: 8WO)

The Series 505 Parallel Word Output (8WO) Module is an eight-channel (multiplexed) module that outputs TTL or CMOS signals. Each channel provides a 16-bit data word that is available as a WY memory location. A BCD (binary-coded decimal) display is commonly connected to a Parallel Word Output Module.

Figure 4-20 shows the form that you use to define a Series 505 Parallel Word Output Module.

Slots required:	2	(user-configurable)
Address type:	WY	(8 consecutive locations)
I/O symbolic name type:	WO	(word output)*

* WO is the default I/O type, but you can change this to a binary-coded decimal output (BO) in the I/O Symbolic Name Table.

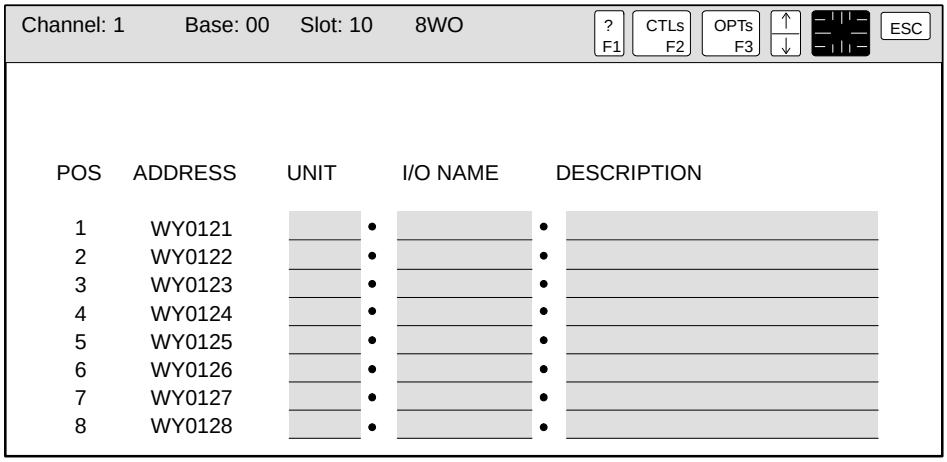


Figure 4-20 Series 505 8-Channel Parallel Word Output Module

Parallel Word Output (Series 505: 16WO)

The Series 505-Compatible Parallel Word Output (16WO) Module is a 16-channel (multiplexed) output module. Each channel provides a 16-bit data word that is available as a WY memory location.

Figure 4-21 shows the form that you use to define a Series 505-compatible parallel word output module.

Slots required:	1	(user-configurable)
Address type:	WY	(16 consecutive locations)
I/O symbolic name type:	WO	(word output)*

* WO is the default I/O type, but you can change this to a binary-coded decimal output (BO) in the I/O Symbolic Name Table.

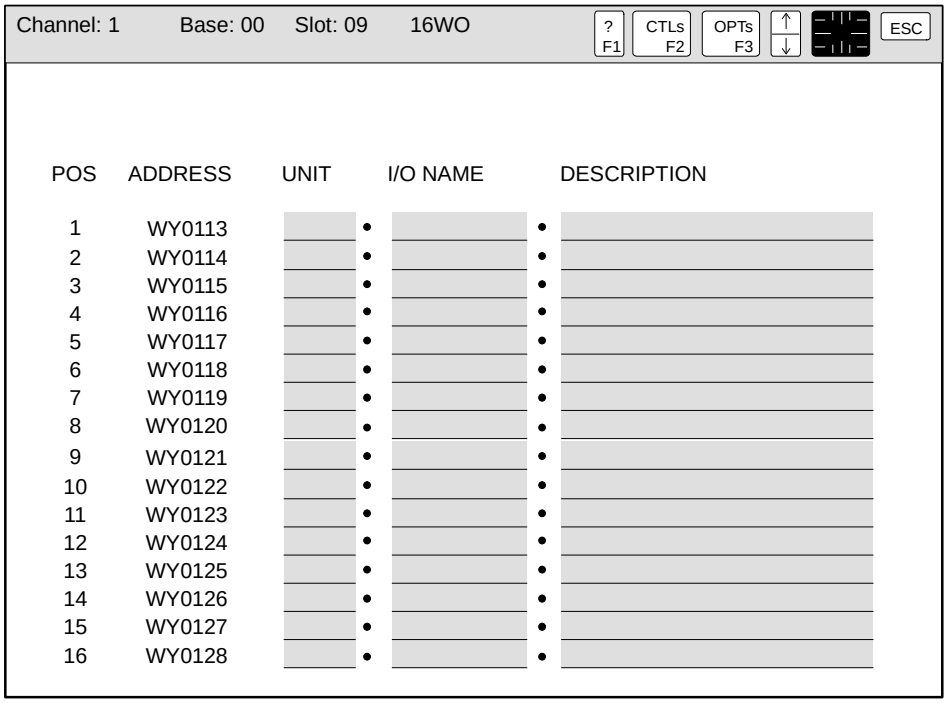


Figure 4-21 Series 505-Compatible 16-Channel Parallel Word Output Module

Standard Modules (continued)

Parallel Word Output SF (Series 505: 8WOSF)

The Series 505-Compatible Parallel Word Output (8WOSF) Special Function (SF) Module operates similarly to the Parallel Word Output (8WO) Module. However, the controller interacts with an 8WOSF module as if it were an intelligent (SF) module. The 8WOSF module is an eight-channel (multiplexed) module that outputs TTL or CMOS signals. Each channel provides a 16-bit data word that is available as a WY memory location. A BCD (binary-coded decimal) display is commonly connected to a Parallel Word Output SF Module.

Figure 4-22 shows the form that you use to define a Series 505-Compatible Parallel Word Output Module.

Slots required:	2	(user-configurable)
Address type:	WY	(8 consecutive locations)
I/O symbolic name type:	WO	(word output)*

* WO is the default I/O type, but you can change this to a binary-coded decimal output (BO) in the I/O Symbolic Name Table.

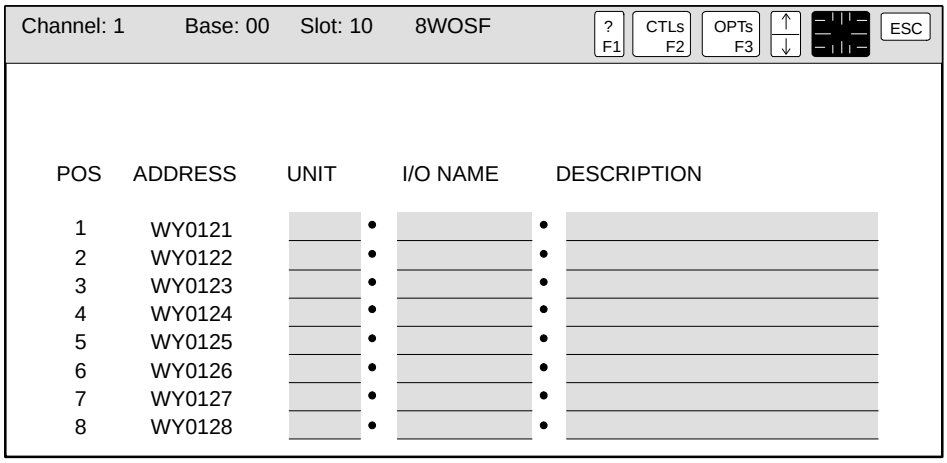


Figure 4-22 Series 505-Compatible 8-Channel Parallel Word Output SF Module

8-Channel Discrete Input (Series 500: 8DI)

The Series 500 8-Channel Discrete Input (8DI) Module accepts eight discrete inputs, which may come from field devices such as push buttons or limit switches.

Figure 4-23 shows the form that you use to define a Series 500 8-Channel Discrete Input Module.

Slots required:	1	(user-configurable)
Address type:	X	(8 consecutive locations)
I/O symbolic name type:	DI	(digital input)

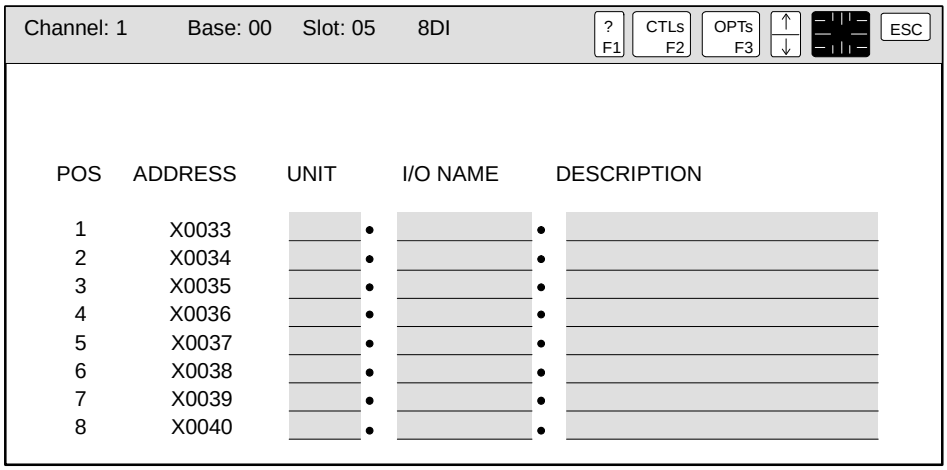


Figure 4-23 Series 500 8-Channel Discrete Input Module

Standard Modules (continued)

8-Channel Discrete Input (Series 505: 8DI)

The Series 505 8-Channel Discrete Input (8DI) Module accepts eight discrete inputs, which may come from field devices such as push buttons or limit switches.

Figure 4-24 shows the form that you use to define a Series 505 8-Channel Discrete Input Module, which has the following configuration requirements.

Slots required:	1	(user-configurable)
Address type:	X	(8 consecutive locations)
I/O symbolic name type:	DI	(digital input)

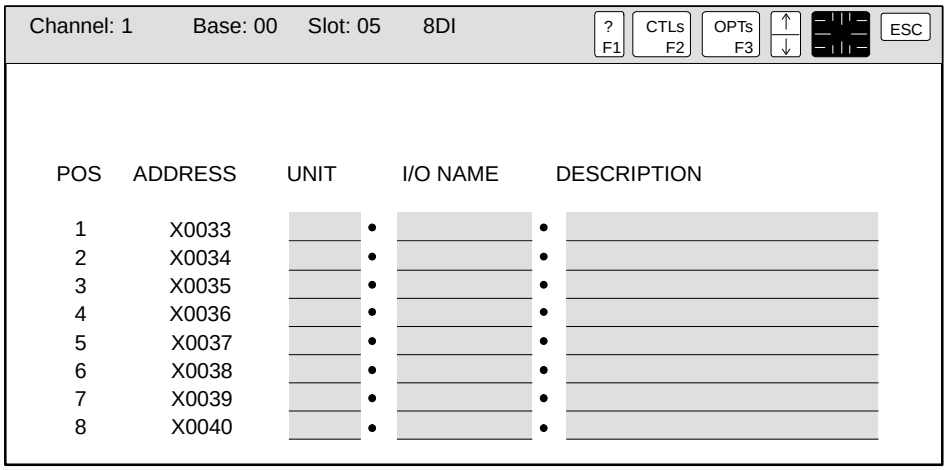


Figure 4-24 Series 505 8-Channel Discrete Input Module

8-Channel Discrete Output (Series 500: 8DO)

The Series 500 8-Channel Discrete Output (8DO) Module provides eight channels of discrete outputs that drive field devices such as pilot lamps, motor starters, or solenoid valves.

Figure 4-25 shows the form that you use to define a Series 500 8-Channel Discrete Output Module.

Slots required:	1	(user-configurable)
Address type:	Y	(8 consecutive locations)
I/O symbolic name type:	DO	(digital output)*

* DO is the default I/O type, but you can change this to a digital flag (DF) in the I/O Symbolic Name Table.

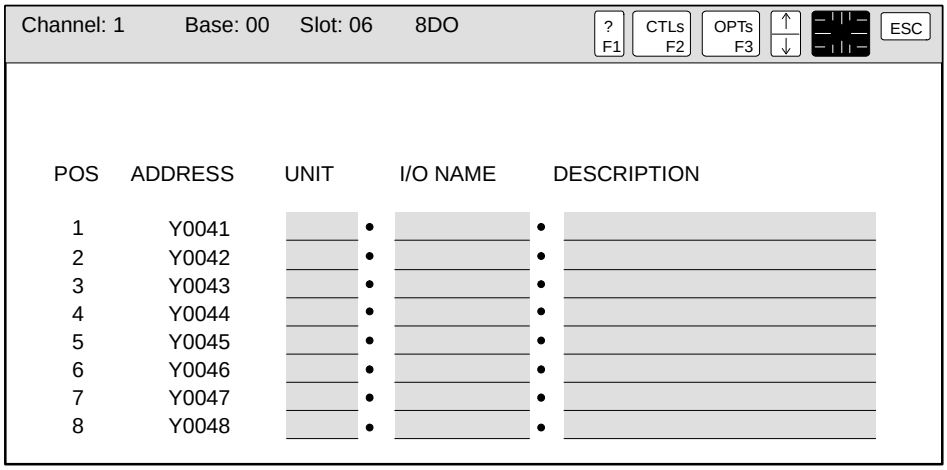


Figure 4-25 Series 500 8-Channel Discrete Output Module

Standard Modules (continued)

8-Channel Discrete Output (Series 505: 8DO)

The Series 505 8-Channel Discrete Output (8DO) Module provides eight channels of discrete outputs that drive field devices such as pilot lamps, motor starters, or solenoid valves.

Figure 4-26 shows the form that you use to define a Series 505 8-Channel Discrete Output Module.

Slots required:	1 or 2;	depends on the output voltage (user-configurable)
Address type:	Y	(8 consecutive locations)
I/O symbolic name type:	DO	(digital output)*

* DO is the default I/O type, but you can change this to a digital flag (DF) in the I/O Symbolic Name Table.

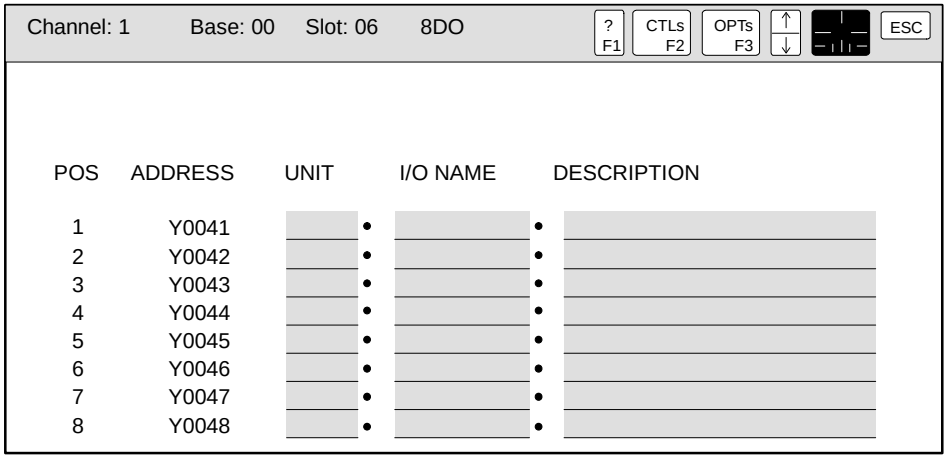


Figure 4-26 Series 505 8-Channel Discrete Output Module

16-Channel Analog Input (Series 505: 16AI)

The Series 505 16-Channel Analog Input (16AI) Module is a 16-channel analog-to-digital converter. In unipolar mode, the module's voltage input range is 0 to 5 VDC, or 0 to 10 VDC. Its current input range is 0 to 20 mA. In bipolar mode, the module's voltage input range is –5 to +5 VDC, or –10 to +10 VDC. Its current input range is –20 to +20 mA. Please refer to the module's user manual for more details concerning the operation of the module.

Figure 4-27 shows the form that you use to define a Series 505 16-Channel Analog Input Module.

Slots required:	1	(user-configurable)
Address type:	WX	(16 consecutive locations)
I/O symbolic name type:	AI	(analog input)

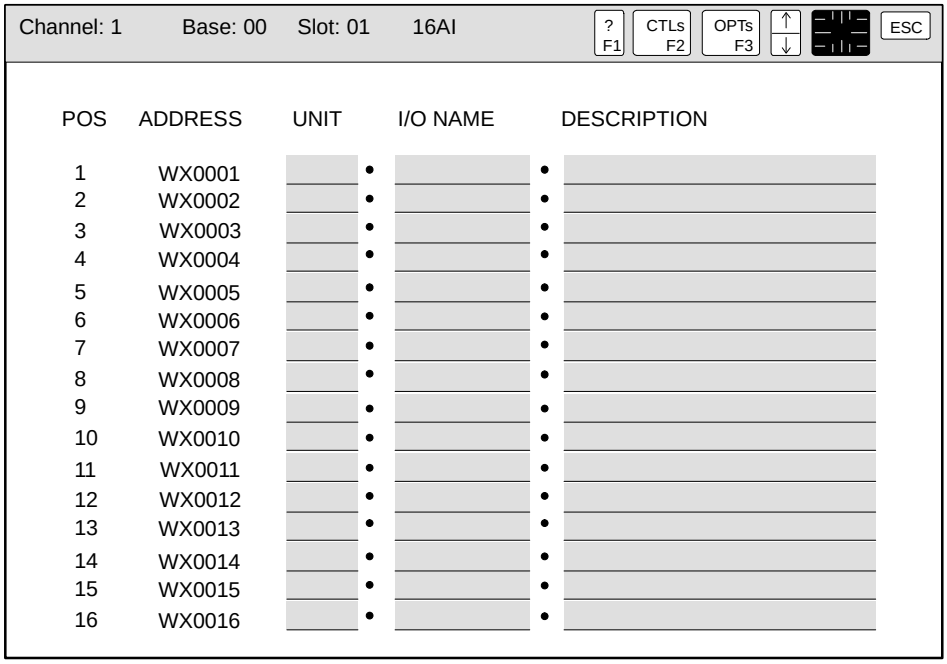


Figure 4-27 Series 505 High Density Analog Input Module

Standard Modules (continued)

16-Channel Discrete Input (Series 505: 16DI)

The Series 505 16-Channel Discrete Input (16DI) Module accepts sixteen discrete inputs, which may come from field devices such as push buttons or limit switches.

Figure 4-28 shows the form that you use to define a Series 505 16-Channel Discrete Input Module.

Slots required:	1	(user-configurable)
Address type:	X	(16 consecutive locations)
I/O symbolic name type:	DI	(digital input)

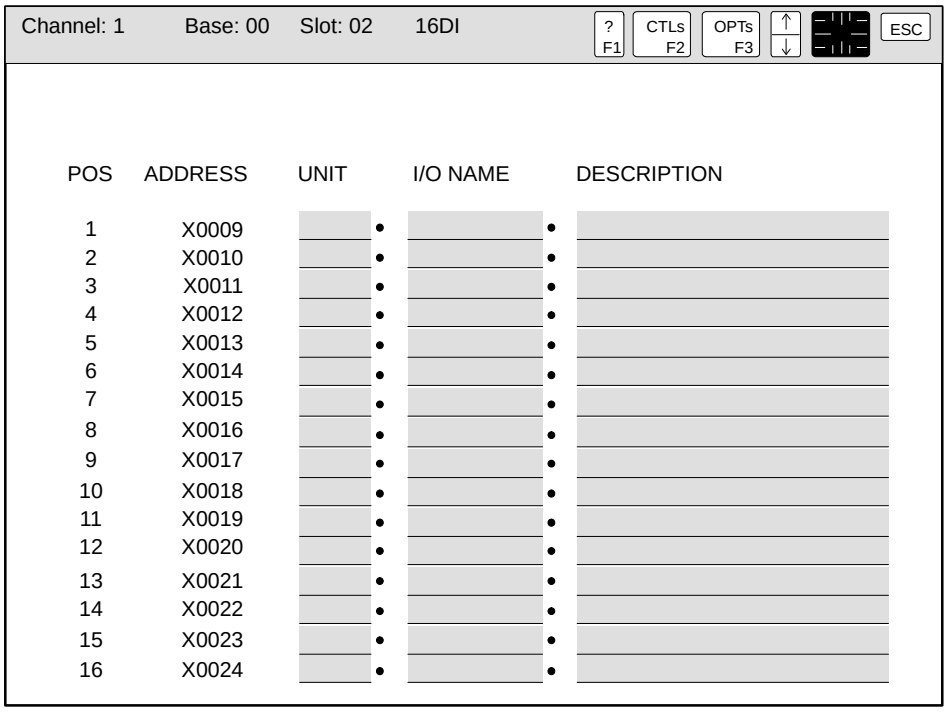


Figure 4-28 Series 505 16-Channel Discrete Input Module

16-Channel Discrete Output (Series 505: 16DO)

The Series 505 16-Channel Discrete Output (16DO) Module provides sixteen channels of discrete outputs that drive field devices such as pilot lamps, motor starters, or solenoid valves.

Figure 4-29 shows the form that you use to define a Series 505 16-Channel Discrete Output Module.

Slots required:	1 or 2;	depends on the output voltage (user-configurable)
Address type:	Y	(16 consecutive locations)
I/O symbolic name type:	DO	(digital output)

* DO is the default I/O type, but you can change this to a digital flag (DF) in the I/O Symbolic Name Table.

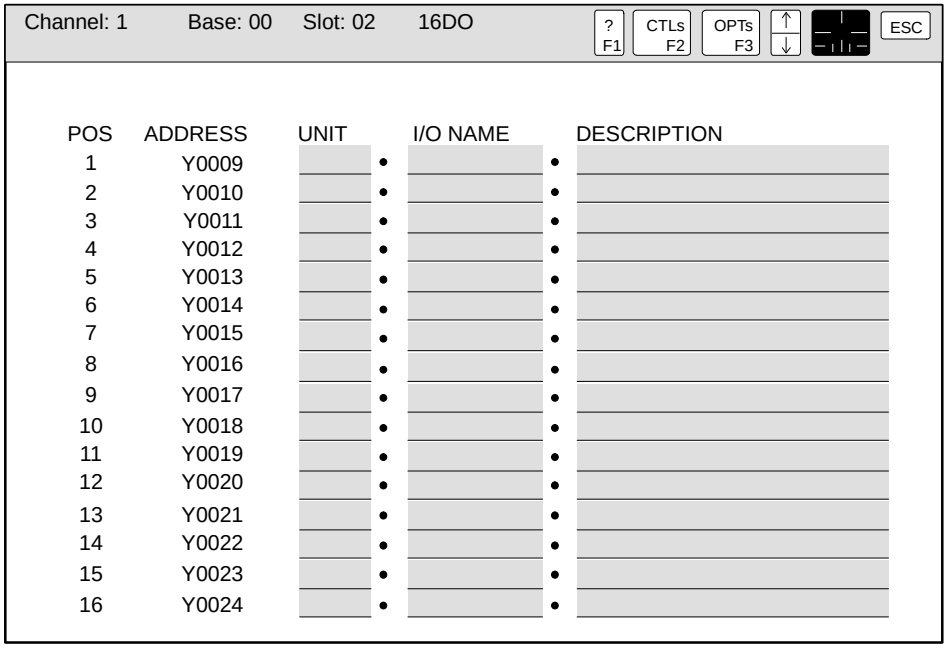


Figure 4-29 Series 505 16-Channel Discrete Output Module

Standard Modules (continued)

20-Channel Analog Input/ 4-Channel Analog Output (Series 505: 2AI4O)

The Series 505 20-Channel Analog Input/4-Channel Analog Output (2AI4O) Module combines 12-bit analog-to-digital converters and 12-bit digital-to-analog converters into one unit. Refer to the module's user manual for details on its electrical characteristics.

Figure 4-30 shows the form that you use to define a Series 505 20-Channel Analog Input/4-Channel Analog Output Module.

Slots required:	1	(user-configurable)
Address type:	WX	(20 consecutive locations)
	WY	(4 consecutive locations)
I/O symbolic name type:	AI	(analog input)
	AO	(analog output)

Channel: 1 Base: 00 Slot: 01 2AI4O ? F1 CTLs F2 OPTs F3 ESC

POS	ADDRESS	UNIT	I/O NAME	DESCRIPTION
1	WX0001			Input 1
2	WX0002			Input 2
3	WX0003			Input 3
4	WX0004			Input 4
5	WX0005			Input 5
6	WX0006			Input 6
7	WX0007			Input 7
8	WX0008			Input 8
9	WX0009			Input 1 Error Word
10	WX0010			Input 2 Error Word
11	WX0011			Input 3 Error Word
12	WX0012			Input 4 Error Word
13	WX0013			Input 5 Error Word
14	WX0014			Input 6 Error Word
15	WX0015			Input 7 Error Word
16	WX0016			Input 8 Error Word
17	WX0017			Output 1 Feedback
18	WX0018			Output 2 Feedback
19	WX0019			Output 3 Feedback
20	WX0020			Output 4 Feedback
21	WY0021			Output 1
22	WY0022			Output 2
23	WY0023			Output 3
24	WY0024			Output 4

Figure 4-30 Series 505 20-Channel Analog Input/4-Channel Analog Output Module

32-Channel Discrete Input (Series 500: 32DI)

The Series 500 32-Channel High Density Discrete Input (32DI) Module accepts 32 channels of discrete inputs from field devices such as push buttons or limit switches.

Figure 4-31 shows the form that you use to define a Series 500 32-Channel High Density Discrete Input Module.

Slots required:	1	(user-configurable)
Address type:	X	(32 consecutive locations)
I/O symbolic name type:	DI	(digital input)

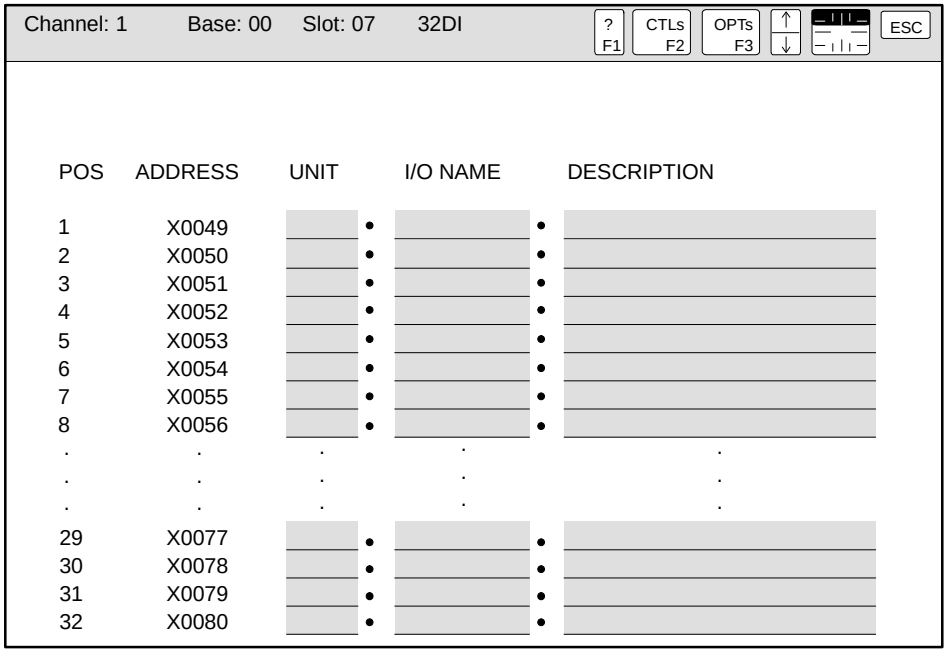


Figure 4-31 Series 500 32-Channel High Density Discrete Input Module

Standard Modules (continued)

32-Channel Discrete Input (Series 505: 32DI)

The Series 505 32-Channel High Density Discrete Input (32DI) Module accepts 32 channels of discrete inputs from field devices such as push buttons or limit switches.

Figure 4-32 shows the form that you use to define a Series 505 32-Channel High Density Discrete Input Module.

Slots required:	1 or 2;	depends on the output voltage (user-configurable)
Address type:	X	(32 consecutive locations)
I/O symbolic name type:	DI	(digital input)

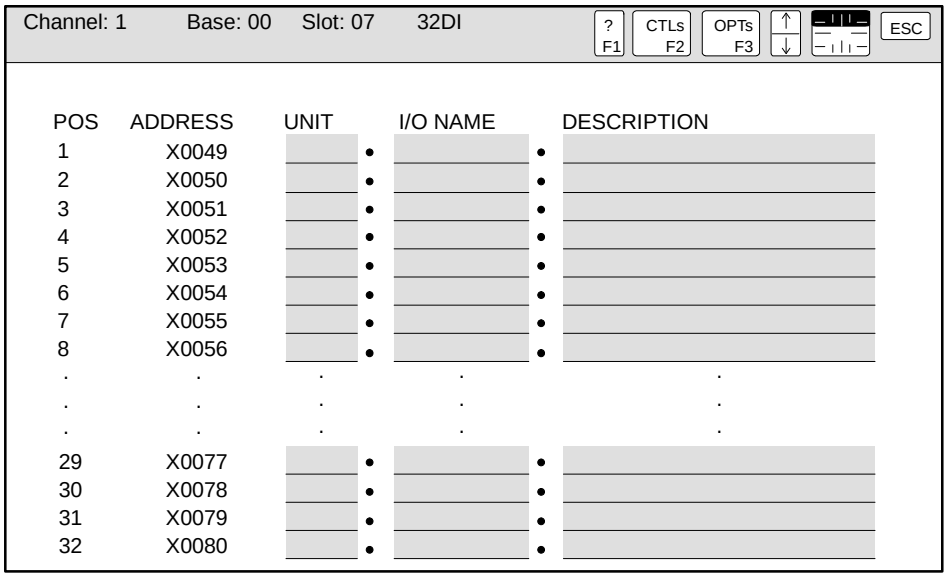


Figure 4-32 Series 505 32-Channel High Density Discrete Input Module

32-Channel Discrete Output (Series 500: 32DO)

The Series 500 32-Channel High Density Discrete Output (32DO) Module provides 32 channels of discrete outputs that control the operation of equipment such as solenoid valves or motor starters.

Figure 4-33 shows the form that you use to define a Series 500 32-Channel High Density Discrete Output Module.

Slots required:	1	(user-configurable)
Address type:	Y	(32 consecutive locations)
I/O symbolic name type:	DO	(digital output)*

* DO is the default I/O type, but you can change this to a digital flag (DF) in the I/O Symbolic Name Table.

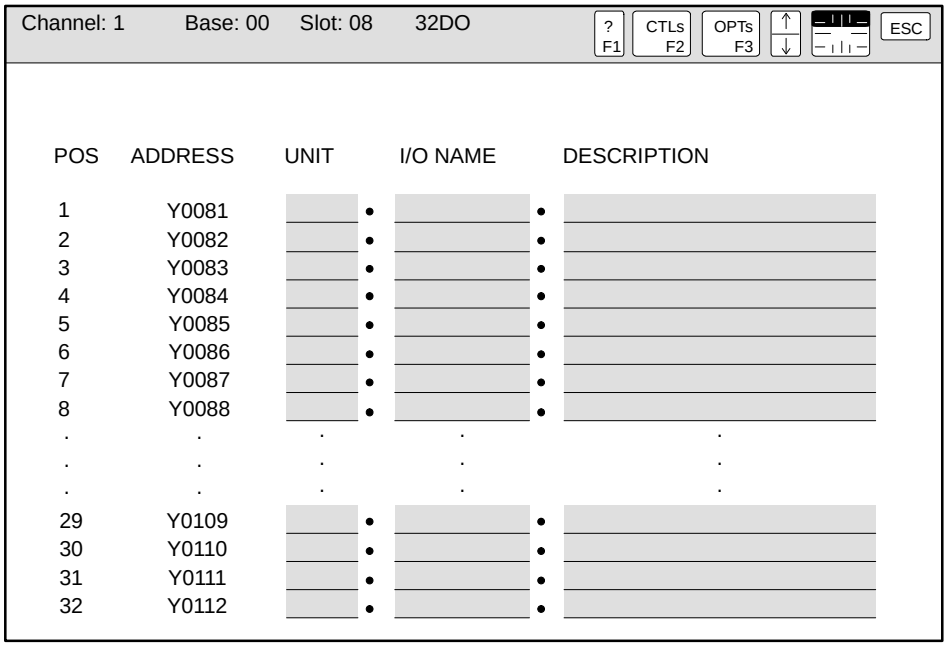


Figure 4-33 Series 500 32-Channel High Density Discrete Output Module

Standard Modules (continued)

32-Channel Discrete Output (Series 505: 32DO)

The Series 505 32-Channel High Density Discrete Output (32DO) Module provides 32 channels of discrete outputs that control the operation of equipment such as solenoid valves or motor starters.

Figure 4-34 shows the form that you use to define a Series 505 32-Channel High Density Discrete Output Module.

Slots required:	1 or 2;	depends on the output voltage (user-configurable)
Address type:	Y	(32 consecutive locations)
I/O symbolic name type:	DO	(digital output)*

* DO is the default I/O type, but you can change this to a digital flag (DF) in the I/O Symbolic Name Table.

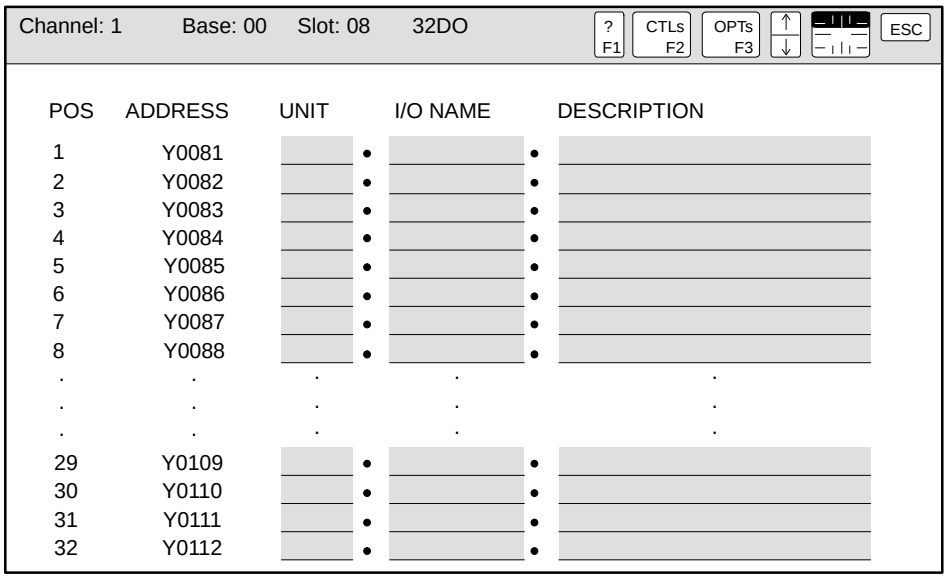


Figure 4-34 Series 505 32-Channel High Density Discrete Output Module

Smartslice (Series 505: SLICE)

The Series 505 Smartslice (SLICE) Module is a self-contained block of I/O that can be located at a remote distance from the base.

Figure 4-35 shows the form that you use to define a Series 505 Smartslice Module.

Slots required:	1	(user-configurable)
Address type:	X	(16 consecutive locations)
	Y	(8 consecutive locations)
I/O symbolic name type:	DI	(digital input)
	DO	(digital output)

Channel: 1 Base: 00 Slot: 02 SLICE ? F1 CTLs F2 OPTs F3 ESC

POS	ADDRESS	UNIT	I/O NAME	DESCRIPTION
1	X0009			
2	X0010			
3	X0011			
4	X0012			
5	X0013			
6	X0014			
7	X0015			
8	X0016			
9	X0017			
10	X0018			
11	X0019			(not used)
12	X0020			(not used)
13	X0021			(not used)
14	X0022			(not used)
15	X0023			(not used)
16	X0024			(not used)
17	Y0025			
18	Y0026			
19	Y0027			
20	Y0028			
21	Y0029			
22	Y0030			
23	Y0031			(not used)
24	Y0032			(not used)

Figure 4-35 Series 505 Smartslice Module

Standard Modules (continued)

User-Defined Module (Series 500: USER)

The Series 500 User (USER) Module is a user-definable module that is not associated with any specific hardware module. Use this module if your module does not fit any of the other modules supported by APT.

The User Module allows you to select the mix of X, Y, WX and WY locations for your specific module needs. If your module is a special function module, select the SF option in the form. The User Module cannot be reconfigured. In order to change the configuration, you must first delete the module and then add a new one. You will have to reconnect the associated I/O points.

Normal I/O and Special Function Modules have eight I/O points. Configurations of a single I/O type are allowed, but modules of that configuration are already supported by APT. The configuration can be in any combination of an input and an output type (X/Y, X/WY, WX/Y, WX/WY), except that the configuration of 5 X and 3 Y is not supported.

High Density Modules are those that have more than eight I/O points and are limited by APT to 64 I/O points. The configuration for the I/O points must follow these rules.

- Discrete I/O points (X and Y) are allowed on eight-point boundaries. The following values are supported: 0, 8, 16, 24, 32, and 64.
- Word I/O points (WX and WY) are allowed on boundaries that are multiples of two. The following values are supported: 0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 32, and 64.
- Unlike Normal I/O and Special Function Modules, High Density Modules allow any of the four I/O types to be combined.

Figure 4-36 shows the forms that you use to define the Series 500 User Module.

Slots required:	1	(user-configurable)
Address type:	X	(user-configurable)
	Y	(user-configurable)
	WX	(user-configurable)
	WY	(user-configurable)
I/O symbolic name type:	DI	(digital input)
	DO	(digital output)
	WI	(word input)
	WO	(word output)

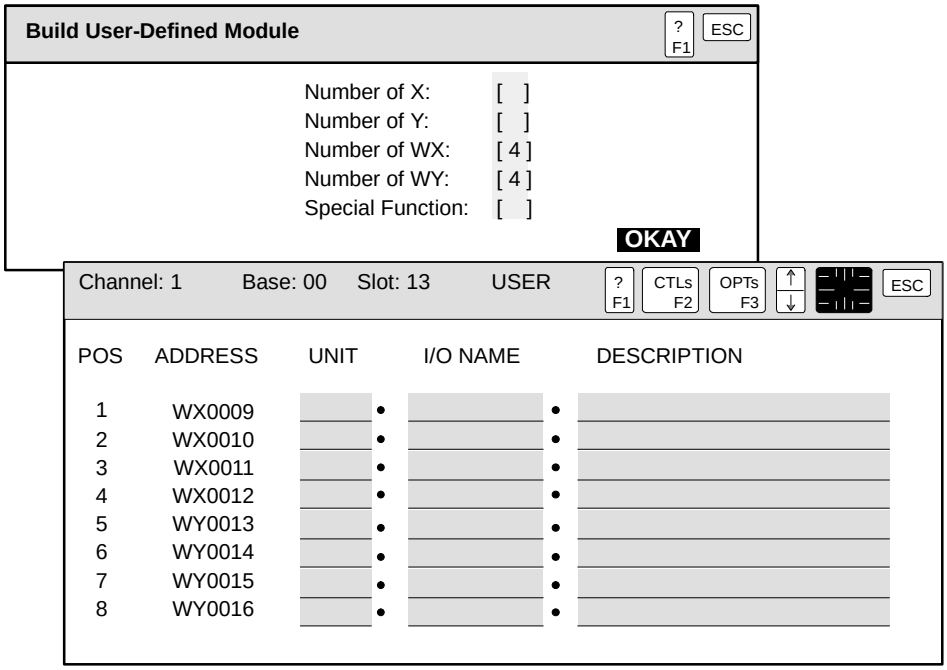


Figure 4-36 Series 500 User Module

Standard Modules (continued)

User-Defined Module (Series 505: USER)

The Series 505 User (USER) Module is a user-definable module that is not associated with any specific hardware module. Use this module if your module does not fit any of the APT supported modules.

The User Module allows you to select the mix of X, Y, WX and WY locations for your specific module needs. If your module is a special function module, select the SF option in the form. The User Module cannot be reconfigured. In order to change the configuration, first delete the module and then add a new one. You will have to reconnect the associated I/O points.

Normal I/O and Special Function Modules have eight I/O points. Configurations of a single I/O type are allowed, but modules of that configuration are already supported by APT. The configuration can be in any combination of an input and an output type (X/Y, X/WY, WX/Y, WX/WY), except that the configuration of 5 X and 3 Y is not supported.

High Density Modules are those that have more than eight I/O points and are limited by APT to 64 I/O points. The configuration for the I/O points must follow these rules.

- Discrete I/O points (X and Y) are allowed on eight-point boundaries. The following values are supported: 0, 8, 16, 24, 32, and 64.
- Word I/O points (WX and WY) are allowed on boundaries that are multiples of two. The following values are supported: 0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 32, and 64.
- Unlike Normal I/O and Special Function Modules, High Density Modules allow any of the four I/O types to be combined.

NOTE: If you are using the User Module to define a rack of I/O, for instance, the ET200, you do not have to use all the types of I/O available for the User Module. However, when you physically wire your I/O rack, you must follow the order expected by the User Module: that is, Xs, Ys, WXs, and finally WYs. So if you are only using WXs and WYs in your ET200 rack, the WXs must precede the WYs.

Figure 4-37 shows the forms that you use to define the Series 505 User Module.

Slots required:	1	(user-configurable)
Address type:	X	(user-configurable)
	Y	(user-configurable)
	WX	(user-configurable)
	WY	(user-configurable)
I/O symbolic name type:	DI	(digital input)
	DO	(digital output)
	WI	(word input)
	WO	(word output)

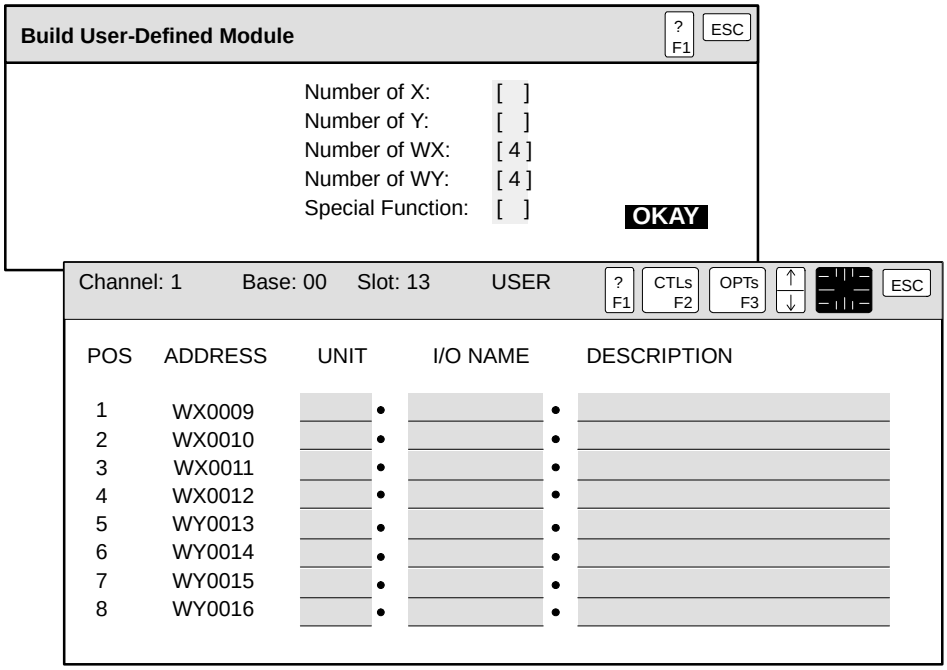


Figure 4-37 Series 505 User Module

Standard Modules (continued)

Isolated Interrupt Discrete Input Module (Series 505: INTRP)

The Series 505 Isolated Interrupt Discrete Input (INTRP) Module can be configured in two ways:

- Sixteen isolated discrete inputs, all non-interrupts.
- Configurable mix of interrupt inputs and non-interrupt inputs. In this mode, the module functions as a 32-point I/O module with 16 physical inputs and 16 internal points. Points 1–8 are the non-interrupt inputs, 9–16 are the interrupt inputs, 17–24 are the internal interrupt status inputs, and 25–32 are the internal interrupt enable bits.

In the interrupt input mode, the interrupt is first enabled (bits 25–32), and then the status is continuously checked (bits 17–24). When an interrupt occurs (bits 9–16), it triggers an immediate response without having to wait for the controller to complete its current program scan. The controller executes the interrupt code and continues with its normal execution (unless the interrupt code specifies that it remain halted).

When used for 16 discrete inputs, the interrupt module can be installed on a local base or on a remote base. When used for interrupts, the interrupt module must be installed on the local base. For this reason, you cannot use the INTRP module with 560, 565, and 575 controllers.

Figure 4-38 shows the forms that you use to define the Series 505 Interrupt Module.

Slots required:	1	
Address type:	X	24 consecutive locations
	Y	8 consecutive locations
I/O symbolic name type:	DI	(digital input)
	DO	(digital output)*

* DO is the default I/O type, but you can change this to a digital flag (DF) in the I/O Symbolic Name Table.

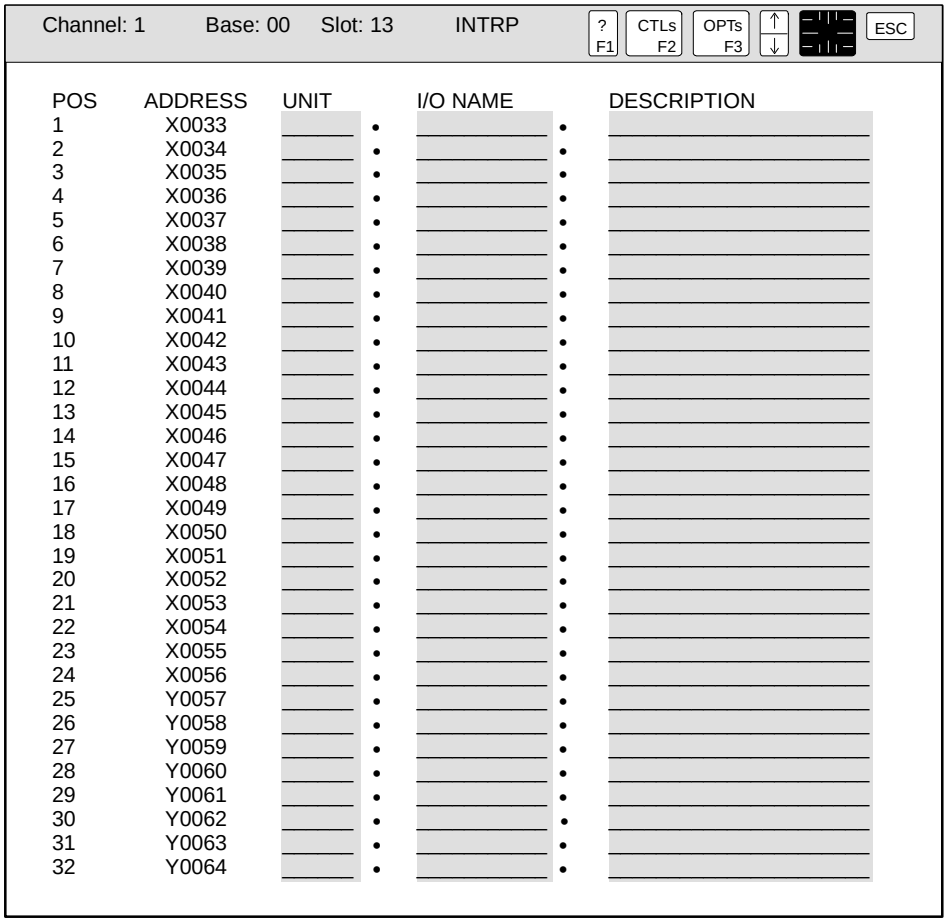


Figure 4-38 Series 505 Isolated Interrupt Discrete Input Module

4.3 Intelligent Modules

ASCII Message Output (Series 500: ASCII)

The Series 500 ASCII Message Output (ASCII) Module is an intelligent I/O module that lets you generate, store, and transmit messages in standard U.S. ASCII-character format. You program the module outside of the APT environment.

There are two ways to communicate with the ASCII module.

- Whenever you program an ASCII module, you define the control relay or digital output that initiates each function and the control relay that signals its completion.

  APT does not recognize these control relays (CRs) unless you reserve CR memory locations in the compile control file. (See the section on Compiling an APT Program (Chapter 6) in the *SIMATIC APT User Manual*.) If you use digital outputs in your ASCII module program, you must also configure these within APT.

- The module uses seven WX locations and one WY location. APT appends the default prefix to the status words (e.g., **xxxxx_ST1** or **xxxxx_ST2**) so that you can monitor the module from a program. For more information about the meaning of these variables, see the manual that comes with the ASCII module.

Figure 4-39 shows the extensions and forms that you use to define an ASCII Message Output Module.

Slots required:	2	double-wide (user-configurable)
Address type:	WX	(7 consecutive locations)
	WY	(1 location)
I/O symbolic name type:	WI	(word input)
	WO	(word output)

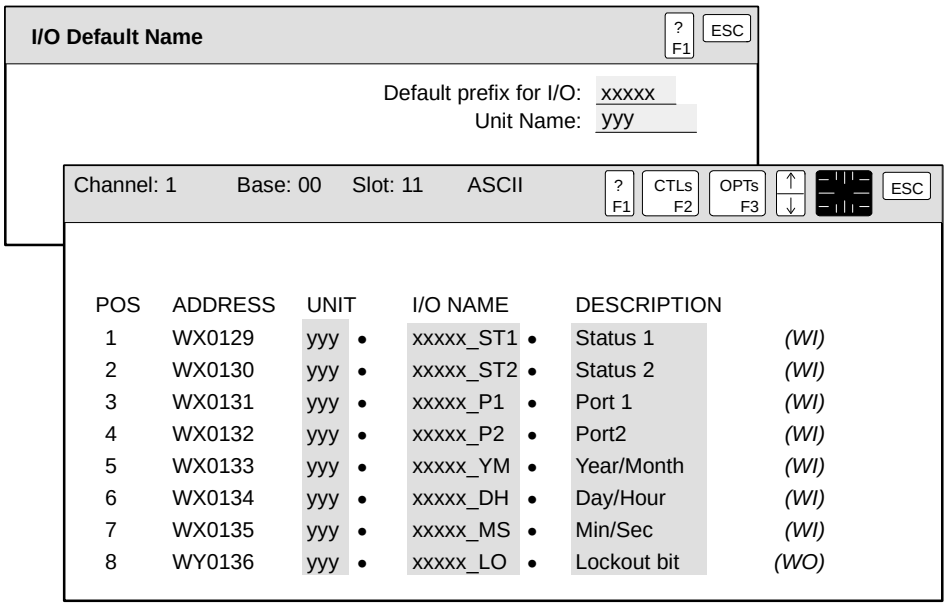


Figure 4-39 Series 500 ASCII Output Module

Intelligent Modules (continued)

Programmable BASIC (Series 500:BASIC)

The Series 500 Programmable BASIC (BASIC) Module is an intelligent module that has a BASIC operating system, on-board memory, and operator interfaces that allow information processing in parallel to the controller. You program the module outside of the APT environment.

APT appends the default prefix that you specify to the I/O words. For information about the meaning of these variables and how to program the module, see the manual that comes with the BASIC Module.

Figure 4-40 shows the forms that you use to define the Series 500 Programmable BASIC Module.

Slots required:	2	double-wide (user-configurable)
Address type:	WX	(4 consecutive locations)
	WY	(4 consecutive locations)
I/O symbolic name type:	WI	(word input)
	WO	(word output)

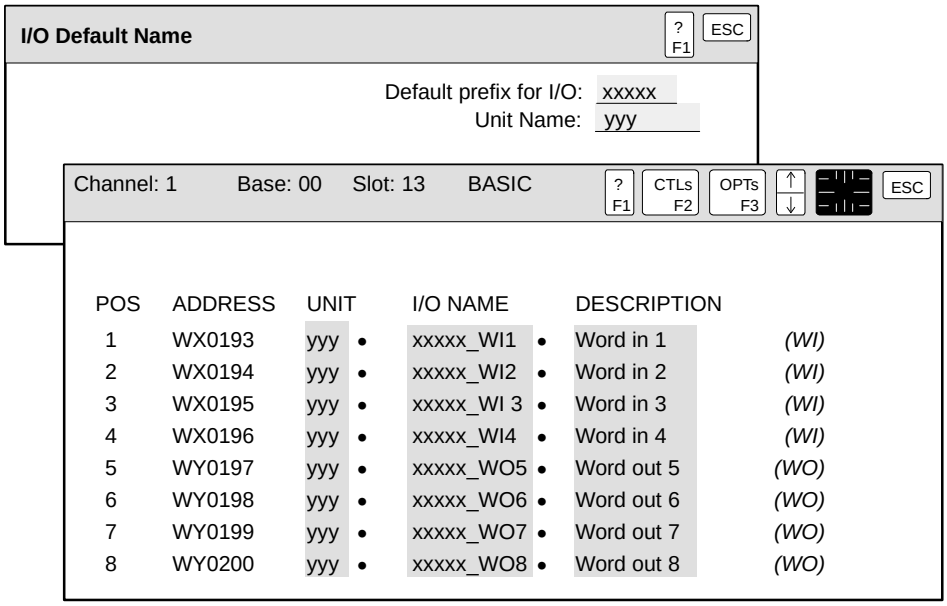


Figure 4-40 Series 500 BASIC Module

Programmable BASIC (Series 505: BASIC)

The Series 505 Programmable BASIC (BASIC) Module is an intelligent module that has a BASIC operating system, on-board memory, and operator interfaces that allow information processing in parallel to the controller. You program the module outside the APT environment.

APT appends the default prefix that you specify to the I/O words. For information about the meaning of these variables and how to program the module, see the manual that comes with the BASIC Module.

Figure 4-41 shows the forms that you use to define the Series 505 Programmable BASIC Module.

Slots required:	1	(user-configurable)
Address type:	WX	(4 consecutive locations)
	WY	(4 consecutive locations)
I/O symbolic name type:	WI	(word input)
	WO	(word output)

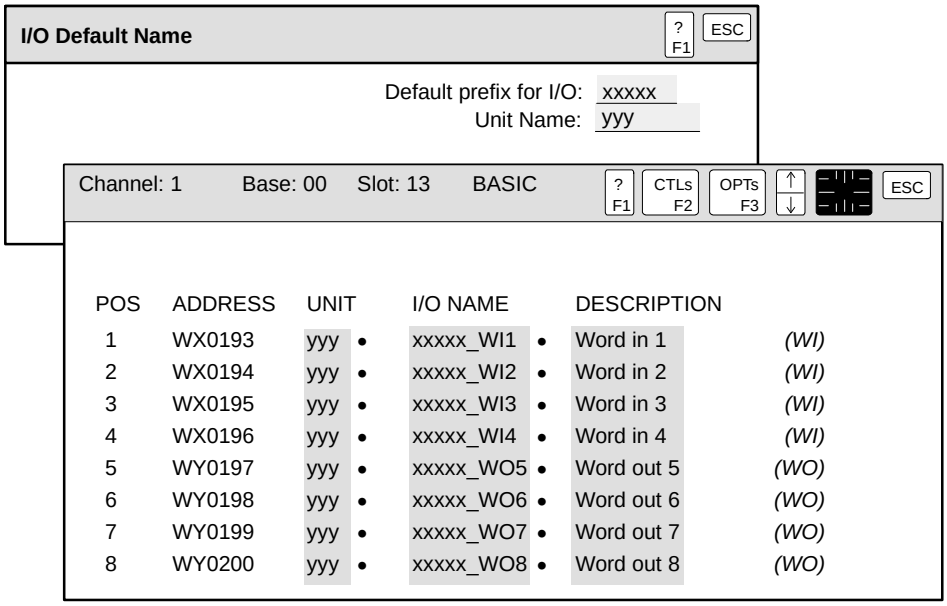


Figure 4-41 Series 505 BASIC Module

Intelligent Modules (continued)

Dual Comm Port (Series 500: DCOMM)

The Series 500 Dual Communications Port (DCOMM) Module provides two independent RS-232-C/423 communications ports to the controller. It can serve as a programming, monitoring, or troubleshooting tool from any remote I/O base.

APT appends the default prefix that you specify to the status words. For more information about the meaning of these variables, see the manual that comes with the Dual Communications Port Module.

Figure 4-42 shows the forms that you use to define the Series 500 Dual Communications Port Module.

Slots required:	2	double-wide (user-configurable)
Address type:	WX	(7 consecutive locations)
	WY	(1 location)
I/O symbolic name type:	WI	(word input)
	WO	(word output)

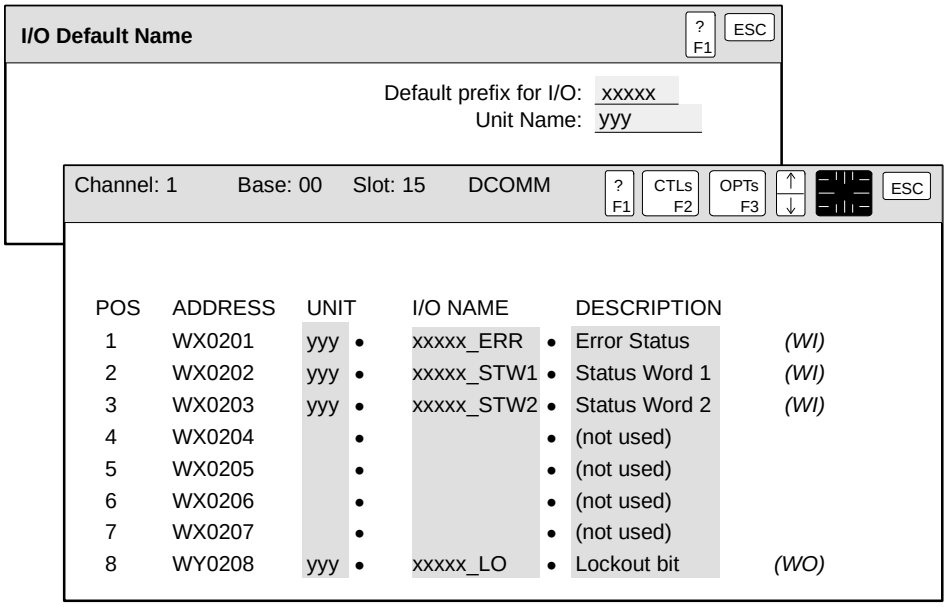


Figure 4-42 Series 500 Dual Comm Port Module

Intelligent Modules (continued)

High Speed Pulse Input (Series 500: HSPI)

The Series 500 High Speed Pulse Input (HSPI) Module is an intelligent module that contains its own microprocessor and memory. It can make decisions based on the total count values from a field device. Typically, this device counts pulse trains that a standard controller cannot accurately count during its scan time.

You program the module outside the APT environment.

When you edit the Series 500 HSPI module, you must specify a counter mode. The module supports six different modes.

- Modes 1–3 are bi-directional, up/down counters.
- Mode 4 provides rate calculations.
- Mode 5 provides ratio calculations.
- Mode 6 provides sum/difference calculations.

APT appends the default prefix that you specify to the status words. For more information about the meaning of these variables, see the manual that comes with the High Speed Pulse Input Module.

Figure 4-43 shows the form that you use to define Counter Mode 1 of the Series 500 High Speed Pulse Input Module.

Counter Modes 2 through 6 have forms similar to the one shown in Figure 4-43.

Slots required:	2	double-wide (user-configurable)
Address type:	WY	(8 consecutive locations)
	WX	(5 consecutive locations)
	Y	(3 consecutive locations
I/O symbolic name type:	WO	(word output)
	WI	(word input)
	DO	(discrete output)

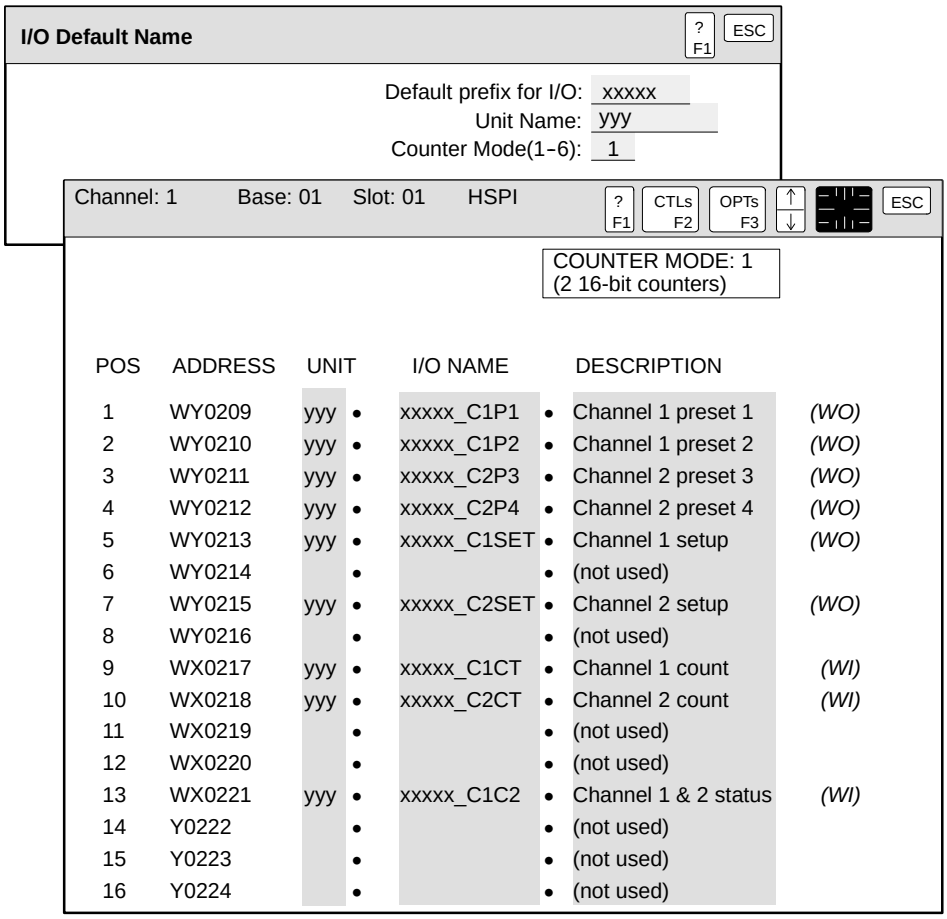


Figure 4-43 Series 500 High Speed Pulse Input Module

Intelligent Modules (continued)

High Speed Counter/Encoder (Series 505: HSC)

The Series 505 High Speed Counter/Encoder (HSC) Module is an intelligent module that contains its own microprocessor and memory. It can make decisions based on the total count values from a field device. Typically, this device counts pulse trains that a standard controller cannot accurately count during its scan time.

You program the module outside the APT environment.

APT appends the default prefix that you specify to the status words. For more information, see the user's manual for the High Speed Counter and Encoder Module.

Figure 4-44 shows the form that you use to define the Series 505 High Speed Counter/Encoder Module.

Slots required:	1	(user-configurable)
Address type:	WX	(3 consecutive locations)
	WY	(5 consecutive locations)
I/O symbolic name type:	WI	(word input)
	WO	(word output)

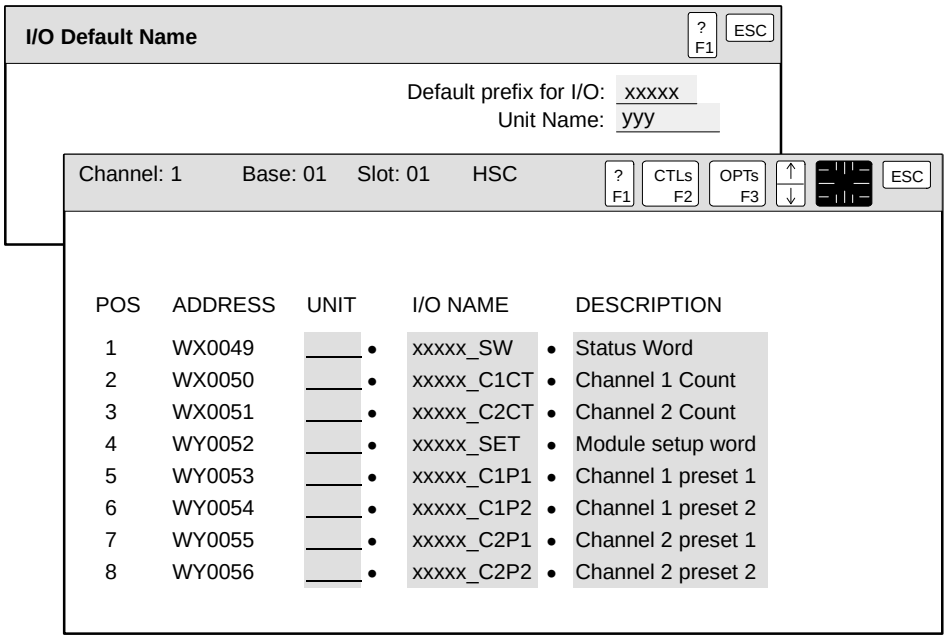


Figure 4-44 Series 505 High Speed Counter Module

Network Interface/ TIWAY (Series 500: NIM)

The Series 500 Network Interface/TIWAY (NIM) Module is the TIWAY I interface for Series 500 programmable controllers. The module provides two TIWAY I communication ports.

APT appends the default prefix that you specify to the status word. For more information about the meaning of this variable, see the manual that comes with the NIM Module.

Figure 4-45 shows the form that you use to define the Series 500 Network Interface Module.

Slots required:	2	double-wide (user-configurable)
Address type:	Y	(1 location)
	Y	(7 APT-reserved locations)
I/O symbolic name type:	DO	(digital output)

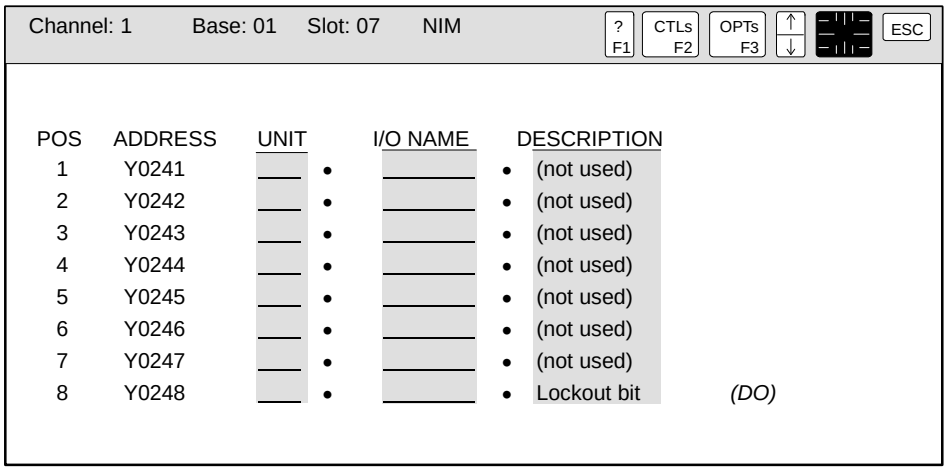


Figure 4-45 Series 500 NIM Module

Intelligent Modules (continued)

Network Interface/ TIWAY (Series 505: NIM)

The Series 505 Network Interface/TIWAY (NIM) Module is the TIWAY I interface for Series 505 programmable controllers. The module provides two TIWAY I communication ports.

APT appends the default prefix that you specify to the status word. For more information about the meaning of this variable, see the manual that comes with the NIM Module.

Figure 4-46 shows the form that you use to define the Series 505 Network Interface Module.

Slots required:	1	(user-configurable)
Address type:	Y	(1 location)
	Y	(7 APT-reserved locations)
I/O symbolic name type:	DO	(digital output)

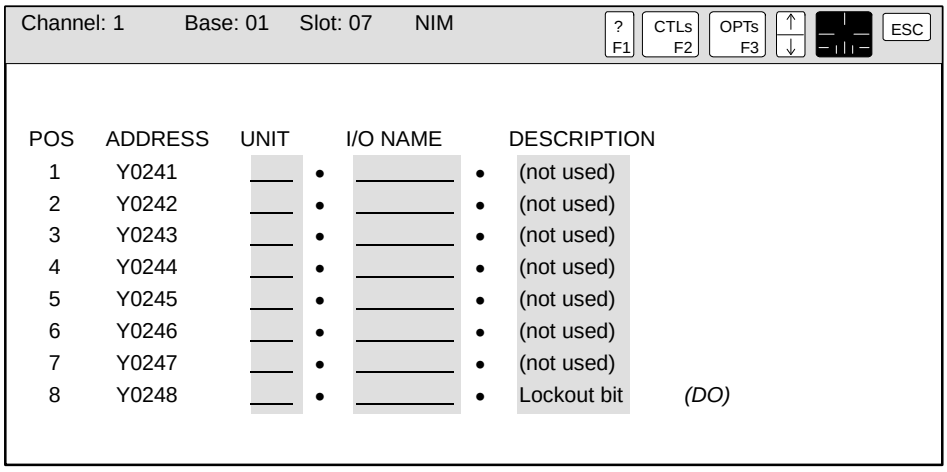


Figure 4-46 Series 505 NIM Module

Peerlink (Series 500: PLINK)

The Series 500 Peerlink (PLINK) Module is an intelligent module that allows each programmable controller on a peer-to-peer network to distribute a block of data to every other controller on the network.

For example, Figure 4-47 shows a network with three stations. Each controller has a block of 16 reserved words for each station. Each controller can write to the reserved words for its own station. The words that are reserved for other stations in that controller are read-only.

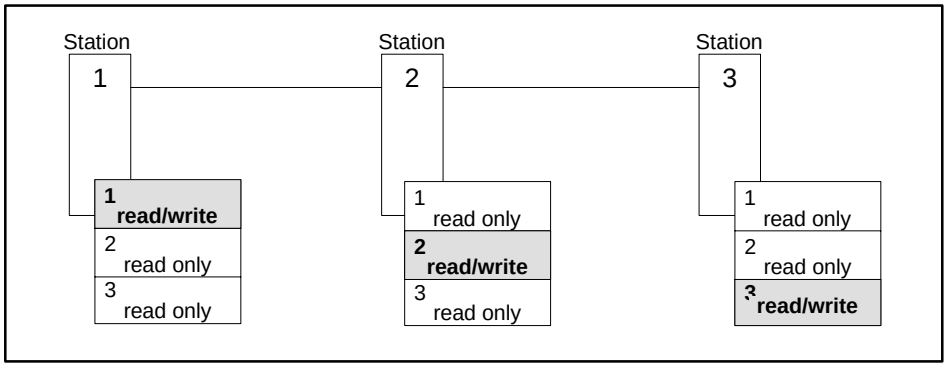


Figure 4-47 Series 500 Peer-to-Peer Network

APT appends the default prefix that you specify to the status words. For more information about the operation of Peerlink, see the manual that comes with the module.

Intelligent Modules (continued)

Figure 4-48 shows the forms that you use to define a Series 500 Peerlink Module.

Slots required:	2	double-wide (user-configurable)
Address type:	WX	(3 consecutive locations)
	WY	(5 consecutive locations)
I/O symbolic name type:	WI	(word input)
	WO	(word output)

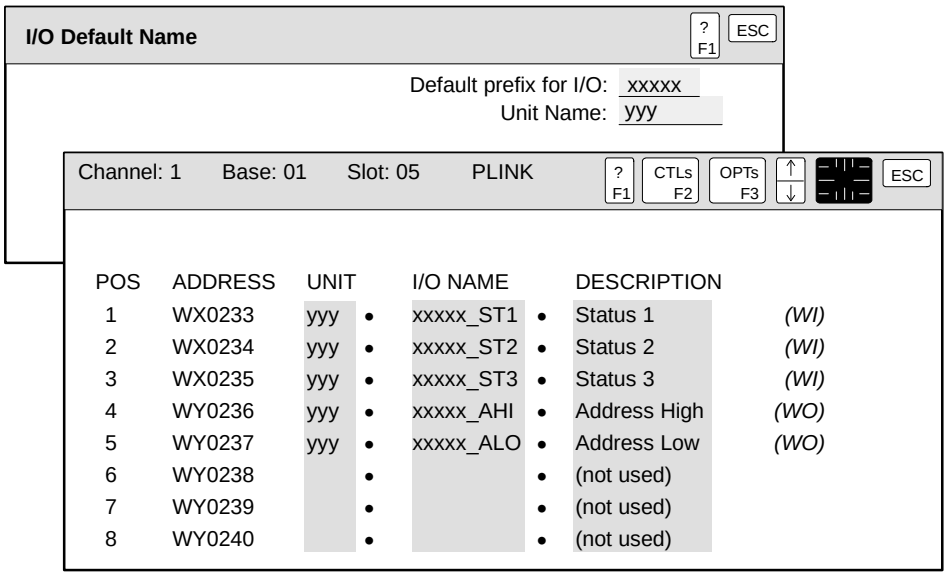


Figure 4-48 Series 500 Peerlink Module

Peerlink (Series 505: PLINK)

The Series 505 Peerlink (PLINK) Module is an intelligent module that allows each programmable controller on a peer-to-peer network distribute a block of data to every other controller on the network.

For example, Figure 4-49 shows a network with three stations. Each controller has a block of 16 reserved words for each station. Each controller can write to the reserved words for its own station. The words that are reserved for other stations in that controller are read-only.

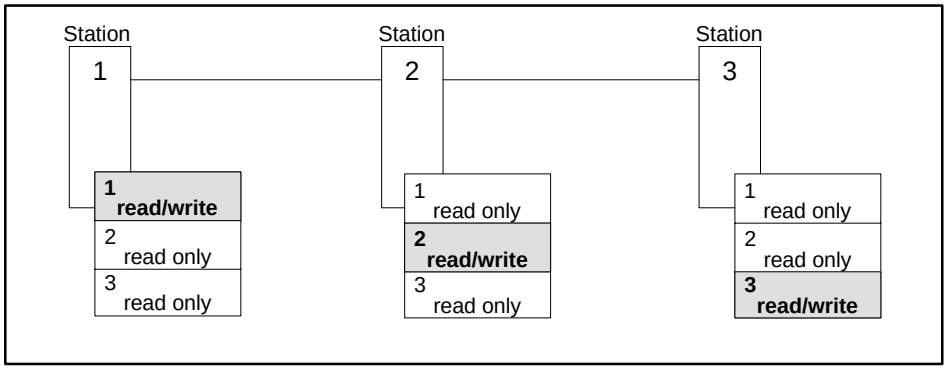


Figure 4-49 Series 505 Peer-to-Peer Network

APT appends the default prefix that you specify to the status words. For more information about the operation of Peerlink, see the manual that comes with the module.

Intelligent Modules (continued)

Figure 4-50 shows the forms that you use to define a Series 505 Peerlink Module.

Slots required:	1	(user-configurable)
Address type:	WX	(3 consecutive locations)
	WY	(5 consecutive locations)
I/O symbolic name type:	WI	(word input)
	WO	(word output)

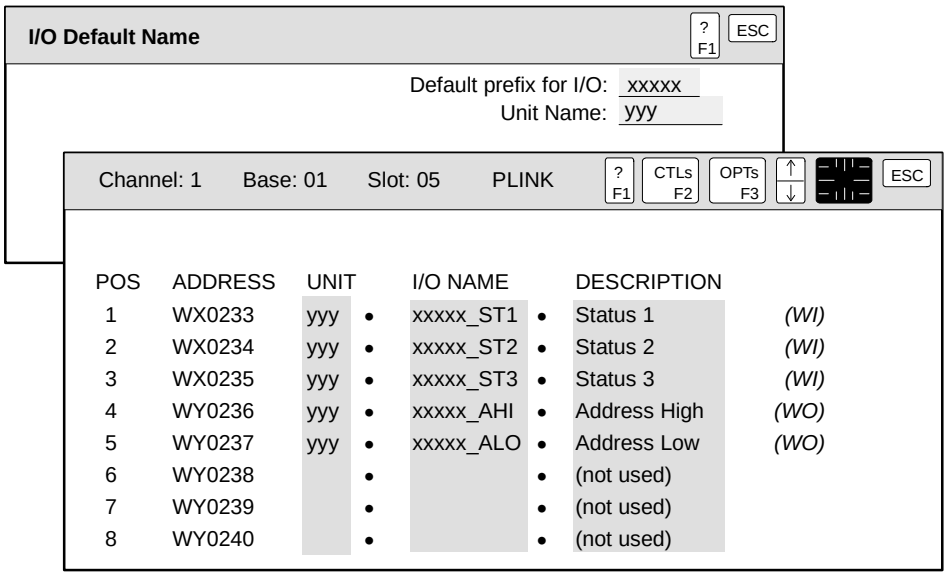


Figure 4-50 Series 505 Peerlink Module

Resistance Temperature Detector (Series 500: RTD)

The Series 500 Resistance Temperature Detector (RTD) Module is an intelligent module that measures an RTD resistance and converts the value to a temperature reading.

The RTD Module maintains a small, consistent current through its attached detector and monitors the voltage generated by the current passing through the detector. For more information, see the manual that comes with the RTD module. If the RTD input is the process variable for a loop or an analog alarm, the .IERR extension of the loop contains the error code if an error occurs.

Figure 4-51 shows the form that you use to define the Series 500 Resistance Temperature Detector Module.

Slots required:	2	double wide (user-configurable)
Address type:	WX	(16 consecutive locations)
I/O symbolic name type:	RT	(Resistance Temperature Detector)

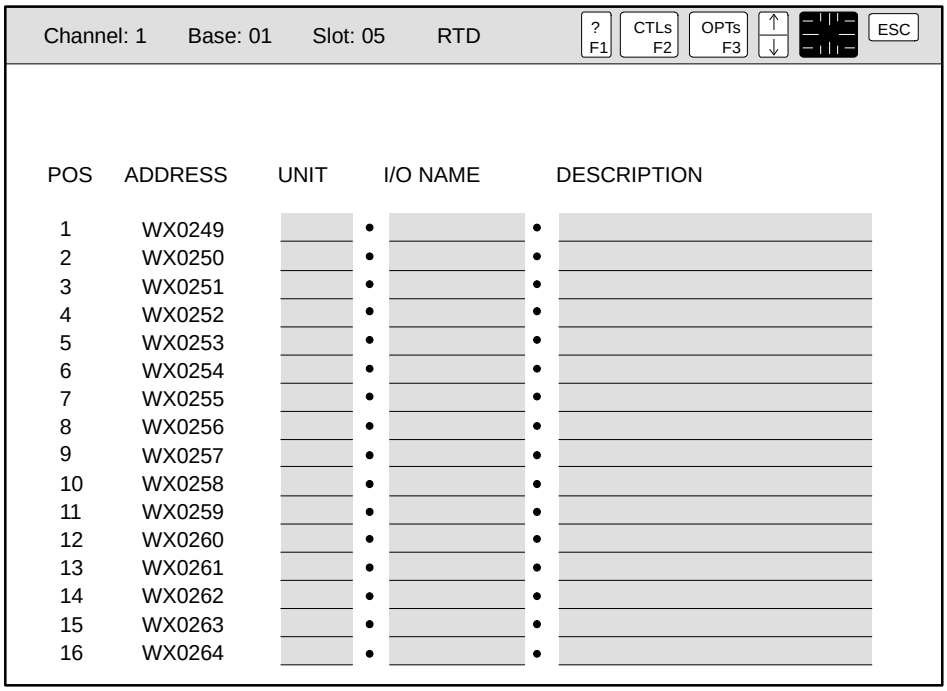


Figure 4-51 Series 500 RTD Module

Intelligent Modules (continued)

Resistance Temperature Detector (Series 505: 8RTD and 16RTD)

The Series 505 Resistance Temperature Detector (8RTD or 16RTD) Module is an intelligent module that measures an RTD resistance and converts it to a temperature value. The RTD Module maintains a small, consistent current through its attached detector and monitors the voltage generated by the current passing through the detector. For more information, see the manual that comes with the RTD module.

When you configure a Series 505 RTD module for 16 word inputs (16RTD), the errors for the first eight inputs are returned on the corresponding inputs 9 through 16. If you are using the inputs as process variables in loops or analog alarms, APT does not recognize the error code because it is not contained in the loop's or analog alarm's **.IERR** extension. If you want the error to be displayed in the **.IERR** extension, configure the module for eight word inputs (8RTD).

Figure 4-52 shows the form to use for defining the Series 505 xxRTD Module (xx=8 or 16).

Slots required:	1	(user-configurable)
Address type:	WX	(8 or 16 consecutive locations)
I/O symbolic name type:	RT	(Resistance Temperature Detector)

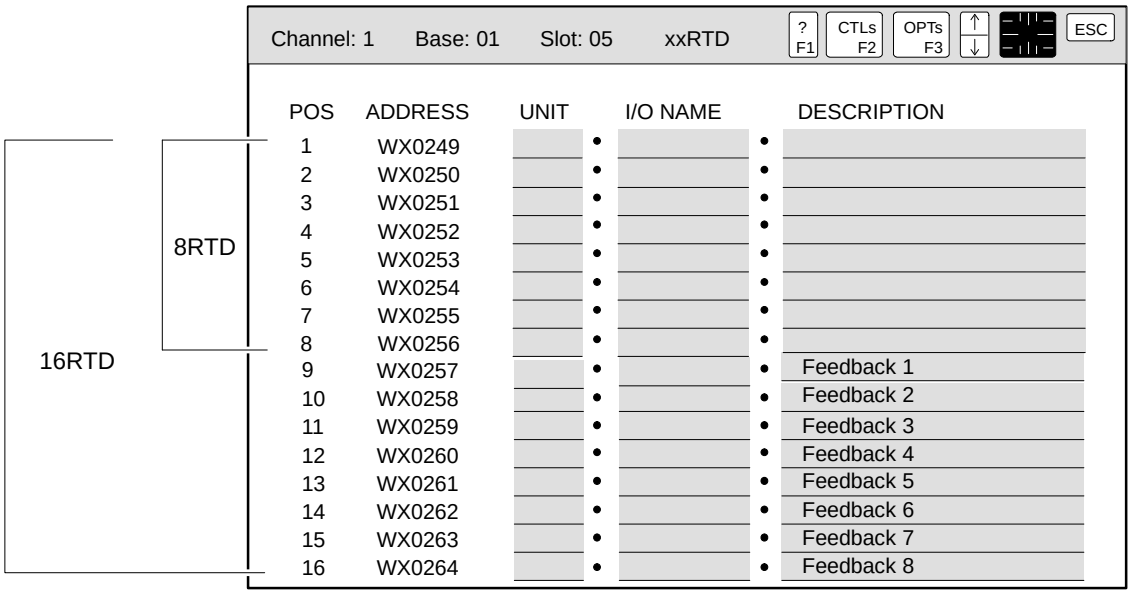


Figure 4-52 Series 505 xxRTD Module

Servo Axis (Series 500: SERVO)

The Series 500 Servo Axis (SERVO) Module controls a single axis of motion. You can synchronize a Servo Axis Module to serve in a multiple-axis system with other Servo Axis Modules.

APT appends the default prefix that you specify to the status words. For more information about the meaning of these variables, see the manual that comes with the SERVO Module.

The Servo Axis Module executes real-time control that is independent of the controller scan. It can select and execute any program, communicating with the module through the eight contiguous I/O words shown in Figure 4-53.

Figure 4-53 shows the forms that you use to define the Series 500 Servo Axis Module.

Slots required:	2	double-wide (user-configurable)
Address type:	WX	(4 consecutive locations)
	WY	(4 consecutive locations)
I/O symbolic name type:	WI	(word input)
	WO	(word output)

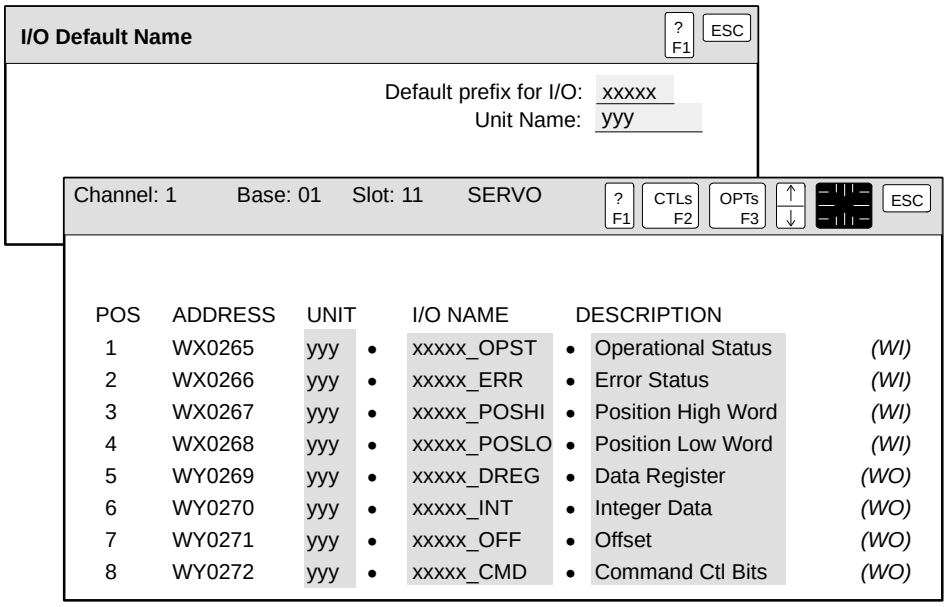


Figure 4-53 Series 500 Servo Axis Module

Intelligent Modules (continued)

Thermocouple (Series 500: TC)

The Series 500 Thermocouple (TC) Module is an intelligent module that measures a thermocouple voltage and converts the value to a temperature reading. The Thermocouple Module also accepts millivolt analog inputs. If the thermocouple input is the process variable for a loop or an analog alarm, the **.IERR** extension of the loop will contain the error code if an error occurs.

Figure 4-54 shows the form that you use to define the Series 500 Thermocouple Module.

Slots required:	2	double-wide (user-configurable)
Address type:	WX	(8 consecutive locations)
I/O symbolic name type:	TC	(Thermocouple)

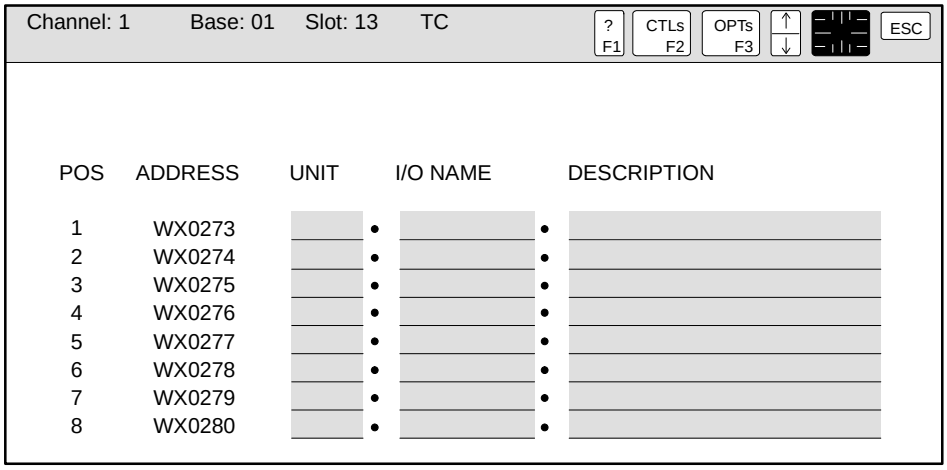


Figure 4-54 Series 500 Thermocouple Module

Thermocouple (Series 505: TC)

The Series 505 Thermocouple (TC) Module is an intelligent module that measures a thermocouple voltage and converts the value to a temperature reading. The Thermocouple Module also accepts millivolt analog inputs. If the thermocouple input is the process variable for a loop or an analog alarm, the **.IERR** extension of the loop will contain the error code if an error occurs.

Figure 4-55 shows the form that you use to define the Series 505 Thermocouple Module.

Slots required:	1	(user-configurable)
Address type:	WX	(8 consecutive locations)
I/O symbolic name type:	TC	(Thermocouple)

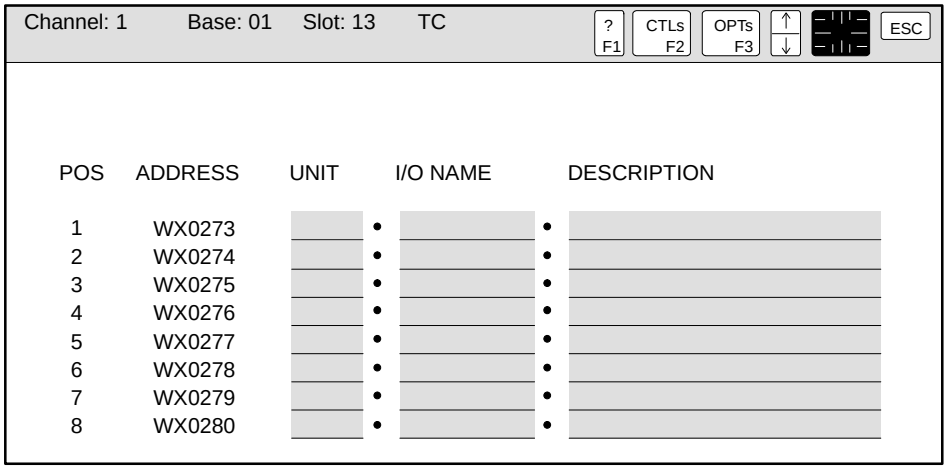


Figure 4-55 Series 505 Thermocouple Module

Intelligent Modules (continued)

386/ATM Coprocessor (Series 505: ATM)

The Series 505 386/ATM Coprocessor (ATM) Module is an intelligent module that is an IBM-compatible Intel 80386 computer with a diskette drive and a hard disk.

Figure 4-56 shows the form that you use to define the Series 505 386/ATM Coprocessor Module.

Slots required:	3	triple-wide (user-configurable)
Address type:	WX	(4 consecutive locations)
	WY	(4 consecutive locations)
I/O symbolic name type:	WI	(word input)
	WO	(word output)

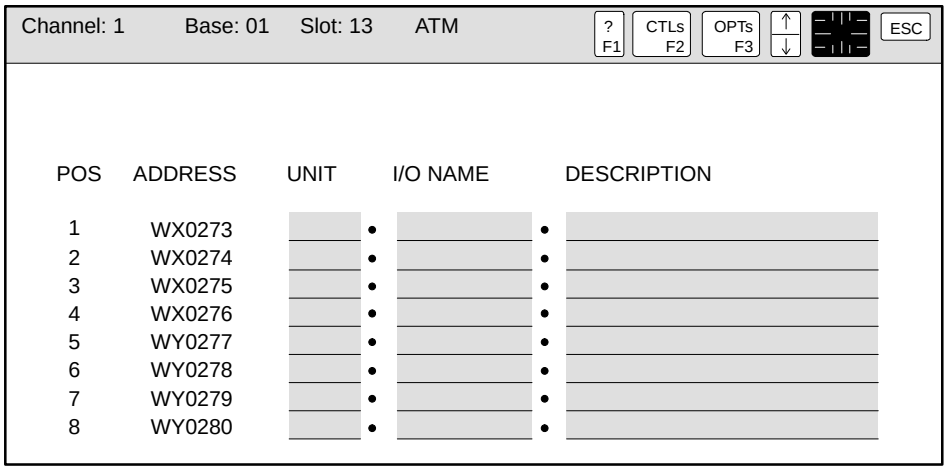


Figure 4-56 Series 505 386/ATM Coprocessor Module

Expert Solutions Processor (Series 500: ESP)

The Series 500 Expert Solutions Processor (ESP) Module is a high performance MS-DOS computer. APT supports only the 4WX/4WY I/O configuration.

APT appends the default prefix that you specify to the status words. For information about the meaning of these variables and how to program the module, see the manual that comes with the Expert Solutions Module.

Figure 4-57 shows the form that you use to define the Series 500 Expert Solutions Processor Module.

Slots required:	2	double-wide (user-configurable)
Address type:	WX	(4 consecutive locations)
	WY	(4 consecutive locations)
I/O symbolic name type:	WI	(word input)
	WO	(word output)

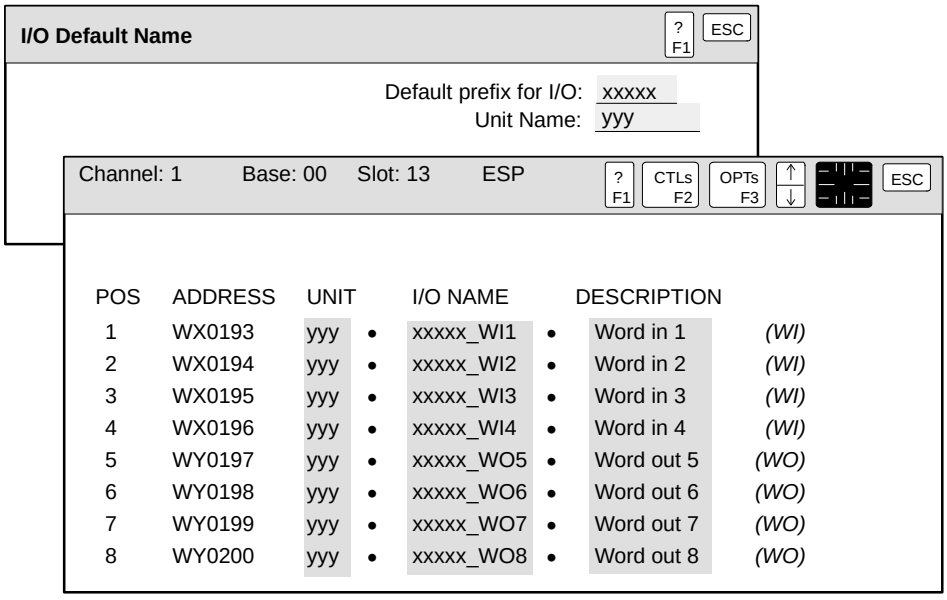


Figure 4-57 Series 500 Expert Solutions Processor Module

Intelligent Modules (continued)

TurboMold (Series 500: TURBO)

The Series 500 TurboMold (TURBO) Module is an intelligent module that has its own set of high-speed I/O for rubber injection molding machine control. The I/O consist of four analog inputs and four analog outputs. The TurboMold module provides these functions.

- Clamp opening and closing
- Injection
- Pack and hold
- Barrel refill/cure

APT appends the default prefix that you specify to the status words. For information about the meaning of these variables, see the manual that comes with the TurboMold Module.

Figure 4-58 shows the forms that you use to define the Series 500 TurboMold Module.

Slots required:	2	double-wide (user-configurable)
Address type:	WX	(4 consecutive locations)
	WY	(4 consecutive locations)
I/O symbolic name type:	WI	(word input)
	WO	(word output)

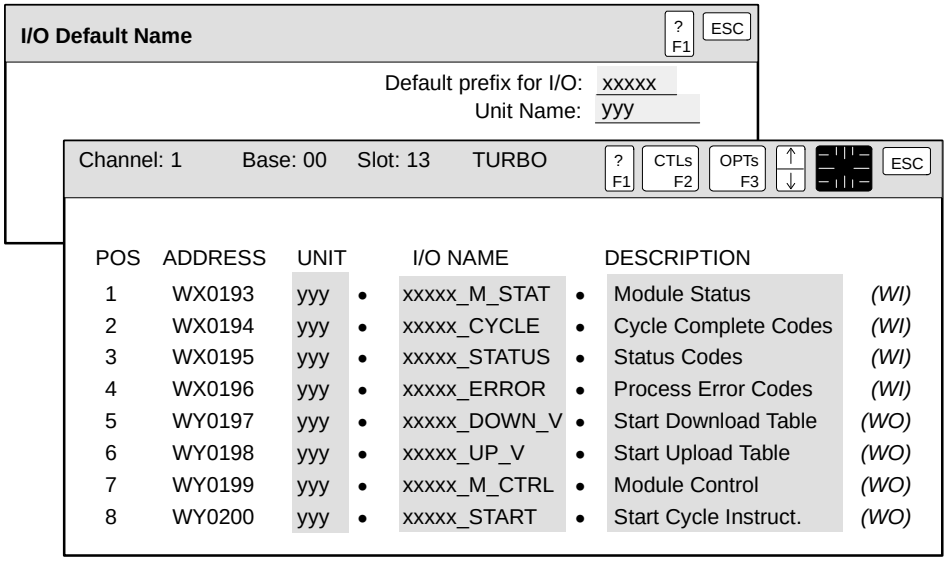


Figure 4-58 Series 500 TurboMold Module

TurboMold (Series 505: TURBO)

The Series 505 TurboMold (TURBO) Module is an intelligent module that has its own set of high-speed I/O for rubber injection molding machine control. The I/O consist of four analog inputs and four analog outputs. The TurboMold module provides these functions.

- Clamp opening and closing
- Injection
- Pack and hold
- Barrel refill/cure

APT appends the default prefix that you specify to the status words. For information about the meaning of these variables, see the manual that comes with the TurboMold Module.

Figure 4-59 shows the forms that you use to define the Series 505 TurboMold Module.

Slots required:	1	(user-configurable)
Address type:	WX	(4 consecutive locations)
	WY	(4 consecutive locations)
I/O symbolic name type:	WI	(word input)
	WO	(word output)

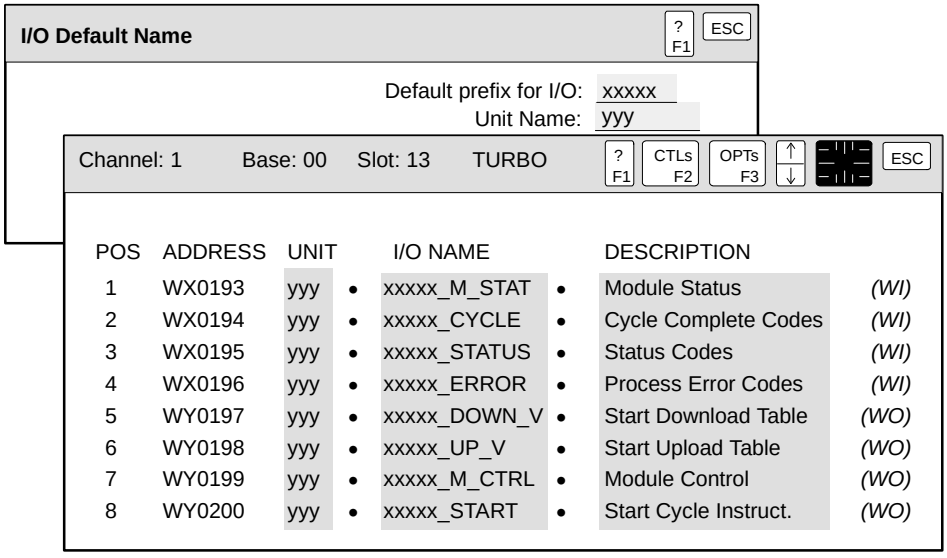


Figure 4-59 Series 505 TurboMold Module

Intelligent Modules (continued)

High Speed PID Controller (Series 500: HSPID)

The Series 500 High Speed PID Controller (HSPID) Module is an intelligent module that has its own set of high-speed I/O for PID loop control. The I/O consist of four analog inputs and four analog outputs.

APT appends the default prefix that you specify to the status words. For information about the meaning of these variables, see the manual that comes with the High Speed PID Controller Module.

Figure 4-60 shows the forms that you use to define the Series 500 High Speed PID Controller Module.

Slots required:	2	double-wide (user-configurable)
Address type:	WX	(4 consecutive locations)
	WY	(4 consecutive locations)
I/O symbolic name type:	WI	(word input)
	WO	(word output)

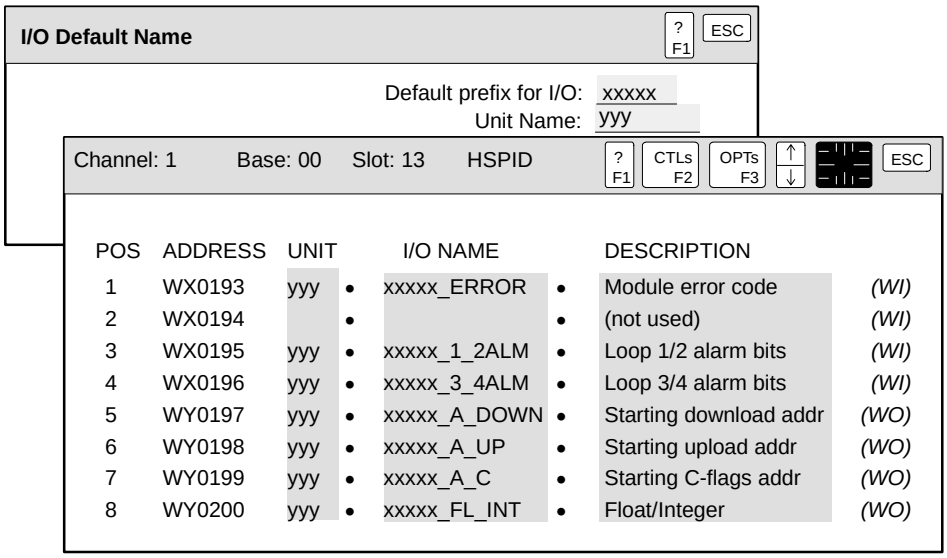


Figure 4-60 Series 500 High Speed PID Controller Module

110 VAC Redundant Output (Series 500: 110VO)

The Series 500 110 VAC Redundant Output (110VO) Module is an intelligent I/O module that senses an input and switches an output without requiring controller intervention. The 110 VAC Redundant Module accepts 110 VAC signals from field devices such as push buttons, limit switches, and optical sensors. The module provides two physically separate and isolated channels.

APT appends the default prefix that you specify to the status words. For more information about the meaning of these variables, see the manual that comes with the 110 VAC Redundant Output Module.

Figure 4-61 shows the forms that you use to define the Series 500 110 VAC Redundant Output Module.

Slots required:	1	(user-configurable)
Address type:	X	(3 consecutive locations)
	Y	(5 consecutive locations)
I/O symbolic name type:	DI	(digital input)
	DO	(digital output)

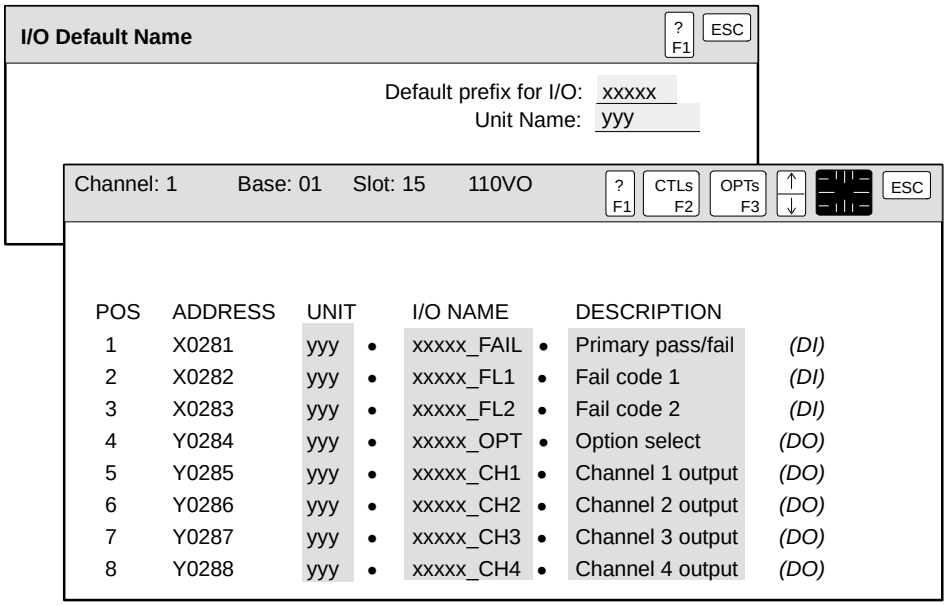


Figure 4-61 Series 500 110 VAC Redundant Output Module

Intelligent Modules (continued)

110 VAC Rapid Response (Series 500: 110VR)

The Series 500 110 VAC Rapid Response (110VR) Module is an intelligent I/O module that senses an input and switches an output without controller intervention. The 110 VAC Rapid Response Module accepts 110 VAC signals from field devices such as push buttons, limit switches, and optical sensors. The module provides two physically separate and isolated channels.

APT appends the default prefix that you specify to the status words. For more information about the meaning of these variables, see the manual that comes with the 110 VAC Rapid Response Module.

Figure 4-62 shows the forms that you use to define the Series 500 110 VAC Rapid Response Module.

Slots required:	1	(user-configurable)
Address type:	X	(2 consecutive locations)
	Y	(4 consecutive locations)
	Y	(2 APT-reserved locations)
I/O symbolic name type:	DI	(digital input)
	DO	(digital output)

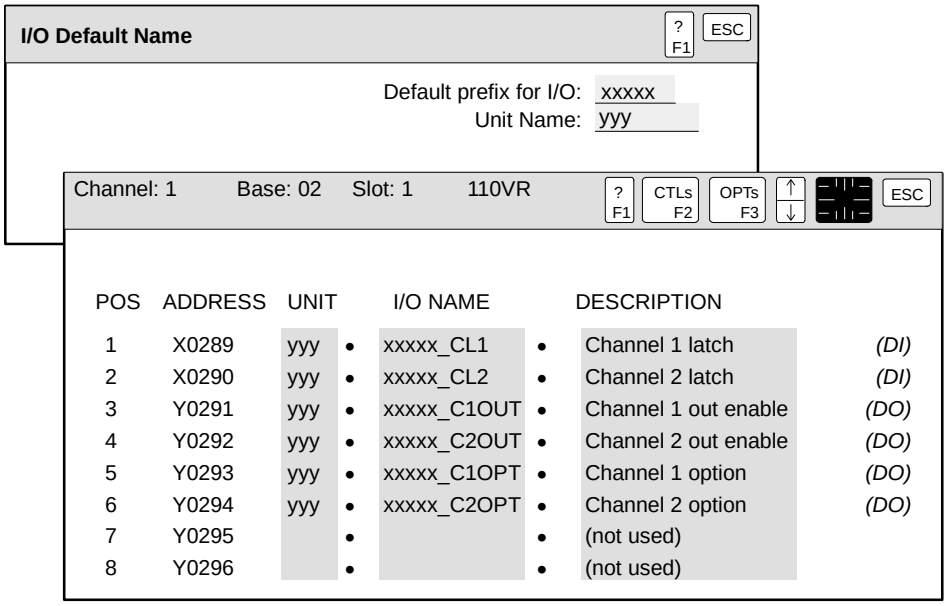


Figure 4-62 Series 500 110 VAC Rapid Response Module

120 VDC Rapid Response (Series 500: 120VR)

The Series 500 120 VDC Rapid Response (120VR) Module is an intelligent I/O module that senses an input and switches an output without controller intervention. The 120 VDC Rapid Response Module accepts 120 VDC signals from field devices such as push buttons, limit switches, and optical sensors. The module provides two physically separate and isolated channels.

APT appends the default prefix that you specify to the status words. For more information about the meaning of these variables, see the manual that comes with the 120 VDC Rapid Response Module.

Figure 4-63 shows the forms that you use to define the Series 500 120 VDC Rapid Response Module.

Slots required:	1	(user-configurable)
Address type:	X	(2 consecutive locations)
	Y	(4 consecutive locations)
	Y	(2 APT-reserved locations)
I/O symbolic name type:	DI	(digital input)
	DO	(digital output)

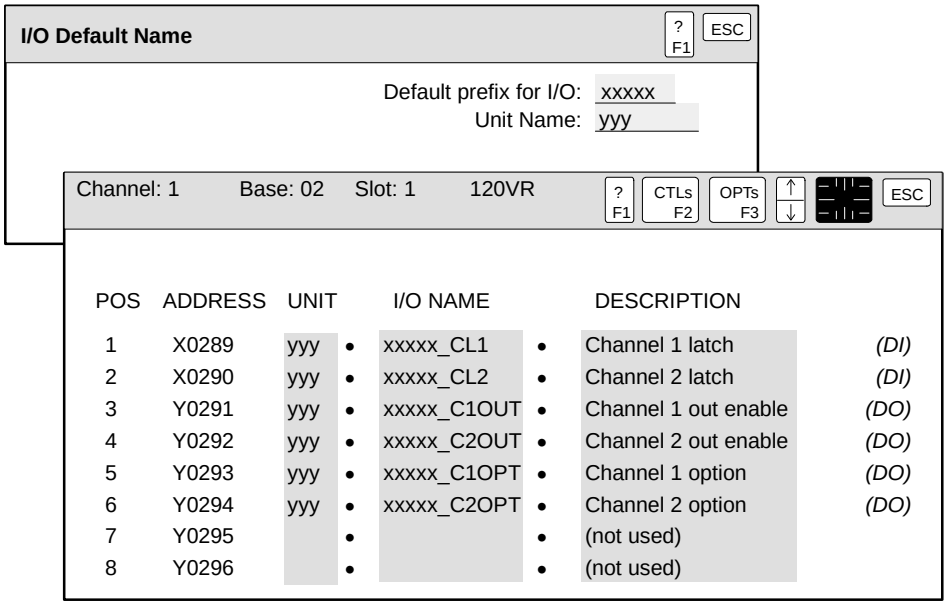


Figure 4-63 Series 500 120 VDC Rapid Response Module

Intelligent Modules (continued)

24 VDC Rapid Response (Series 500: 24VDC)

The Series 500 24 VDC Rapid Response (24VDC) Module is an intelligent I/O module that senses an input and switches an output without controller intervention. The 24 VDC Rapid Response Module accepts 24 VAC signals from field devices such as push buttons, limit switches, and optical sensors. It detects an input and responds with an output more quickly than a standard module that uses normal, discrete I/O. The module provides two physically separate and isolated channels.

APT appends the default prefix that you specify to the status words. For more information about the meaning of these variables, see the manual that comes with the 24 VDC Rapid Response Module.

Figure 4-64 shows the forms that you use to define the Series 500 24 VDC Rapid Response Module.

Slots required:	1	(user-configurable)
Address type:	X	(2 consecutive locations)
	Y	(4 consecutive locations)
	Y	(2 APT-reserved locations)
I/O symbolic name type:	DI	(digital input)
	DO	(digital output)

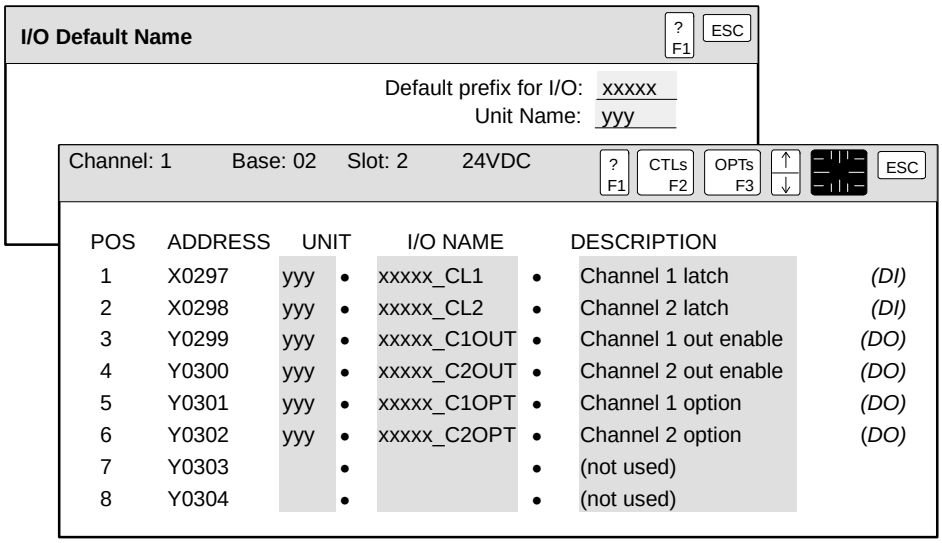


Figure 4-64 Series 500 24 VDC Rapid Response Module

Intelligent Modules (continued)

Field Interface SIMOREG Mode (Series 505:SREG)

The Series 505 Field Interface Module (FIM) with the SIMOREG (SREG) configuration is a field interface module that controls a SIMOREG DC drive. Although the module requires a configuration of 14 WYs, four are unused. APT downloads only 10 WYs to the controller. The controller then communicates to the DC drive; the 10 WXs contain the response from the DC drive. (See the manuals for the SIMATIC 505 Field Interface Module and the SIMOREG Digital DC Drive Serial Interface for more information concerning the contents of these WYs and WXs.)

Since the FIM module acts like a remote base controller (RBC) card and can handle up to 15 SIMOREG DC motors, you need to define the SIMOREG DC drives that are controlled by the FIM module in a separate I/O base. Configure the I/O base on the FIM module with switch S1. For example, if you set switch S1 to 2 on the FIM module, then you must configure your SREG modules in APT for I/O base 2.

When you set up an SREG in APT, each SREG appears as a slot in the I/O base. You can use slots 1 through 15 to define the SIMOREG DC motors that are controlled by the FIM module, and you must define a SIMOREG broadcast channel (SREGB) in slot 16. The SIMOREGs do not work unless you have an SREGB defined in slot 16 of whichever I/O base you have configured for the FIM module. You do not have to use the SREGB, but it must be configured. (See the *SIMATIC 505 Field Interface Module User Manual* for more information.)

Figure 4-65 shows the form that you use to define a FIM Module in this mode.

Slots required:	1	(user-configurable)
Address type:	WX	(10 consecutive locations)
	WY	(10 consecutive locations)
I/O symbolic name type:	WI	(word input)
	WO	(word output)

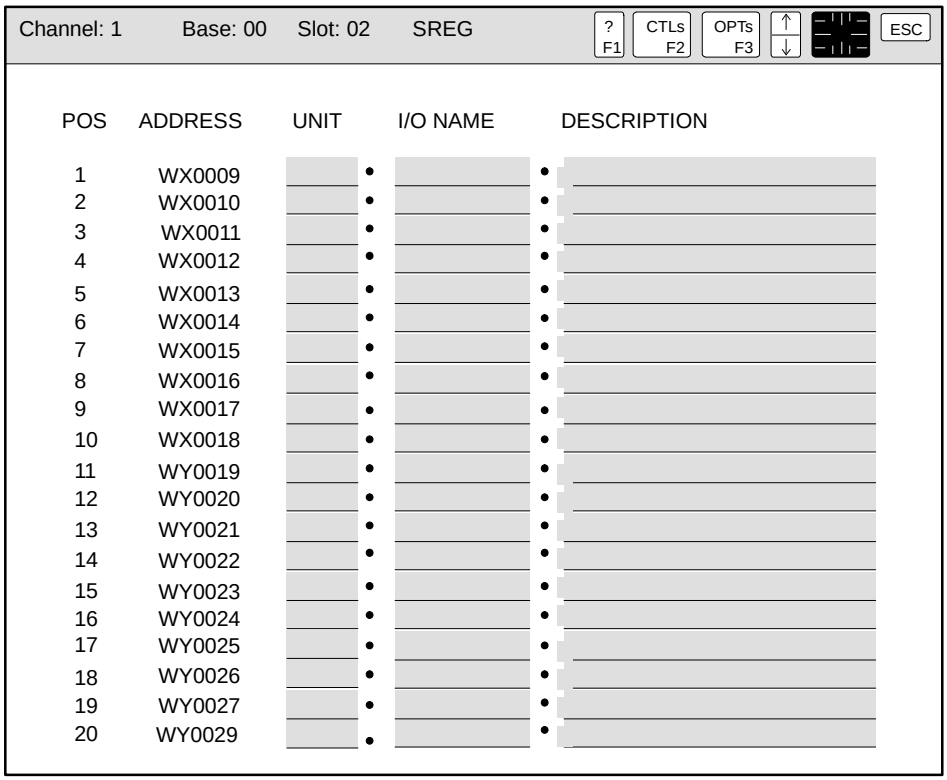


Figure 4-65 Series 505 FIM Module: SIMOREG Mode

Intelligent Modules (continued)

Field Interface SIMOREG Broadcast Mode (Series 505: SREGB)

The Series 505 Field Interface Module (FIM) with the SIMOREG broadcast (SREGB) configuration is a field interface module that controls Siemens SIMOREG DC drives in broadcast mode. You can send an immediate command to one or more DC motors, or to all 15 motors, with the SREGB. (See the *SIMATIC 505 Field Interface Module User Manual* for a description of the immediate broadcast mode.)

Although the module requires a configuration of 16 WYs, six are unused. Only 10 WYs are downloaded to the controller. You must declare the SREGB in slot 16 of whichever I/O base you have configured to contain the SREGs. Slot 16 is designated as the broadcast channel for the FIM module. The SREGs do not work unless there is an SREGB defined in slot 16.

Figure 4-66 shows the form that you use to define a FIM module in this mode.

Slots required:	1	(user-configurable)
Address type:	X	(16 consecutive locations)
	Y	(16 consecutive locations)
	WY	(10 consecutive locations)
I/O symbolic name type:	DI	(digital input)
	DO	(digital output)
	WO	(word output)

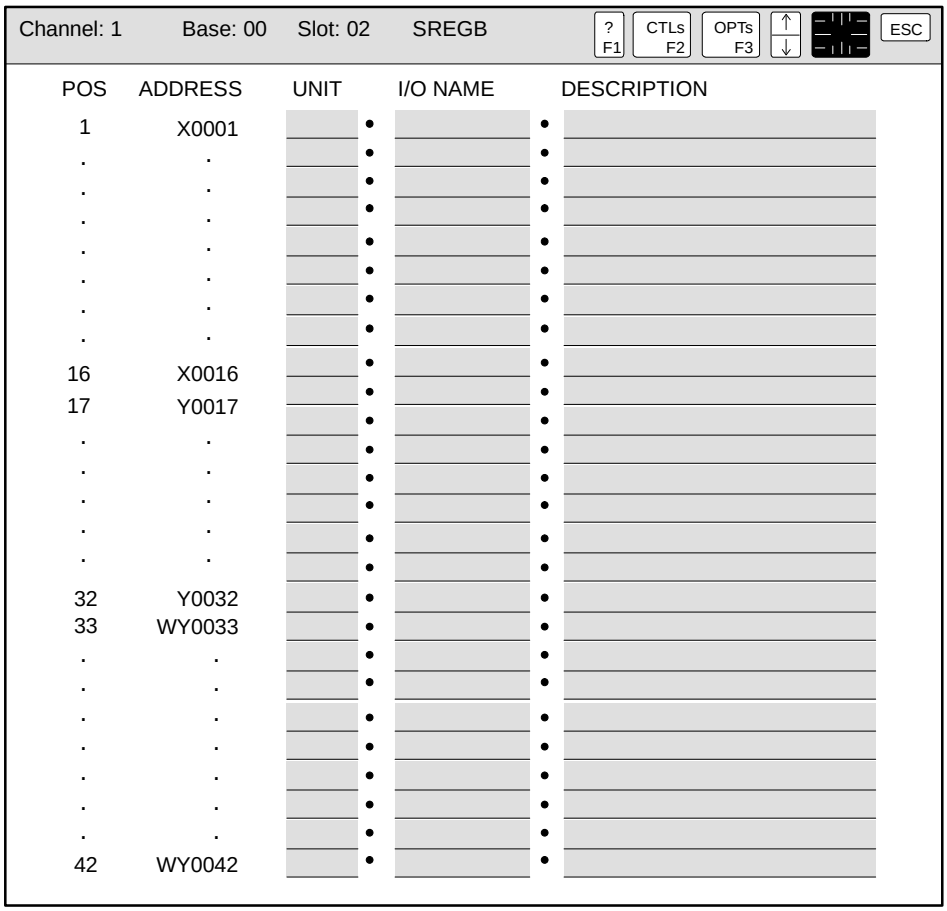


Figure 4-66 Series 505 FIM Module: SIMOREG Broadcast Mode

Intelligent Modules (continued)

Field Interface SIMOVERT Mode (Series 505:SVRT)

The Series 505 Field Interface Module (FIM) with the SIMOVERT (SVRT) configuration is a field interface module that controls a SIMOVERT AC drive. APT downloads four WYs to the controller. The controller then communicates to the AC drive; the four WXs contain the response from the AC drive. (See the manuals for the SIMATIC 505 Field Interface Module and the SIMOVERT AC Drive Serial Interface for more information concerning the contents of these WYs and WXs.)

Since the FIM module acts like a remote base controller (RBC) card and can handle up to 15 SIMOVERT AC motors, you need to define the SIMOVERT AC drives that are controlled by the FIM module in a separate I/O base. Configure the I/O base on the FIM module with switch S1. For example, if you set switch S1 to 2 on the FIM module, then you must configure your SVRT modules in APT for I/O base 2.

When you set up an SVRT in APT, each SVRT appears as a slot in the I/O base. You can use slots 1 through 15 to define the SIMOVERT AC motors that are controlled by the FIM module, and you must define a SIMOVERT broadcast channel (SVRTB) in slot 16. The SIMOVERTs do not work unless you have an SVRTB defined in slot 16 of whichever I/O base you have configured for the FIM module. You do not have to use the SVRTB, but it must be configured. (See the *SIMATIC 505 Field Interface Module User Manual* for more information.)

Figure 4-67 shows the form that you use to define a FIM module in this mode.

Slots required:	1	(user-configurable)
Address type:	WX	(4 consecutive locations)
	WY	(4 consecutive locations)
I/O symbolic name type:	WI	(word input)
	WO	(word output)

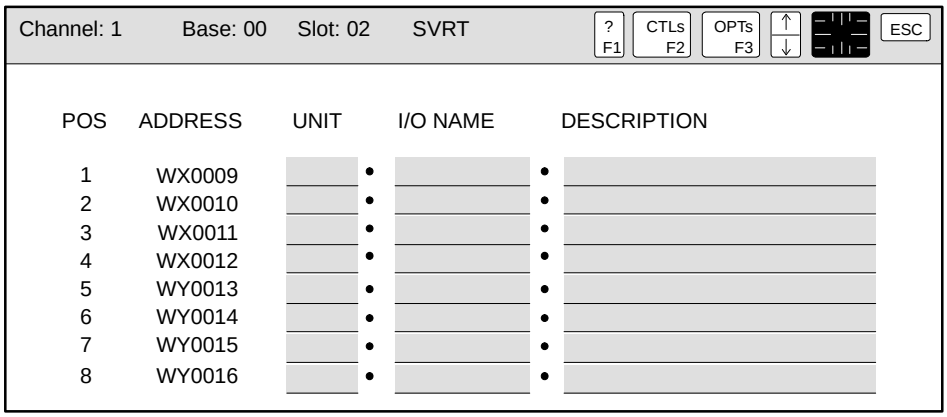


Figure 4-67 Series 505 FIM Module: SIMOVERT Mode

Intelligent Modules (continued)

Field Interface SIMOVERT Broadcast Mode (Series 505:SVRTB)

The Series 505 Field Interface Module (FIM) with the SIMOVERT broadcast (SVRTB) configuration is a field interface module that controls Siemens SIMOVERT AC drives in broadcast mode. You can send an immediate command to one or more AC motors, or to all 15 motors, with the SVRTB. (See the *SIMATIC 505 Field Interface Module User Manual* for a description of the immediate broadcast mode.)

Although the module requires a configuration of eight WYs, four are unused. Only four WYs are downloaded to the controller. You must declare the SRVTB in slot 16 of whichever I/O base you have configured to hold your SVRTs. Slot 16 is the broadcast channel on the FIM module. The SVRTs do not work unless there is an SVRTB defined in slot 16.

Figure 4-68 shows the form that you use to define a FIM module in this mode.

Slots required:	1	(user-configurable)
Address type:	X	(16 consecutive locations)
	Y	(16 consecutive locations)
	WY	(4 consecutive locations)
I/O symbolic name type:	DI	(digital input)
	DO	(digital output)
	WO	(word output)

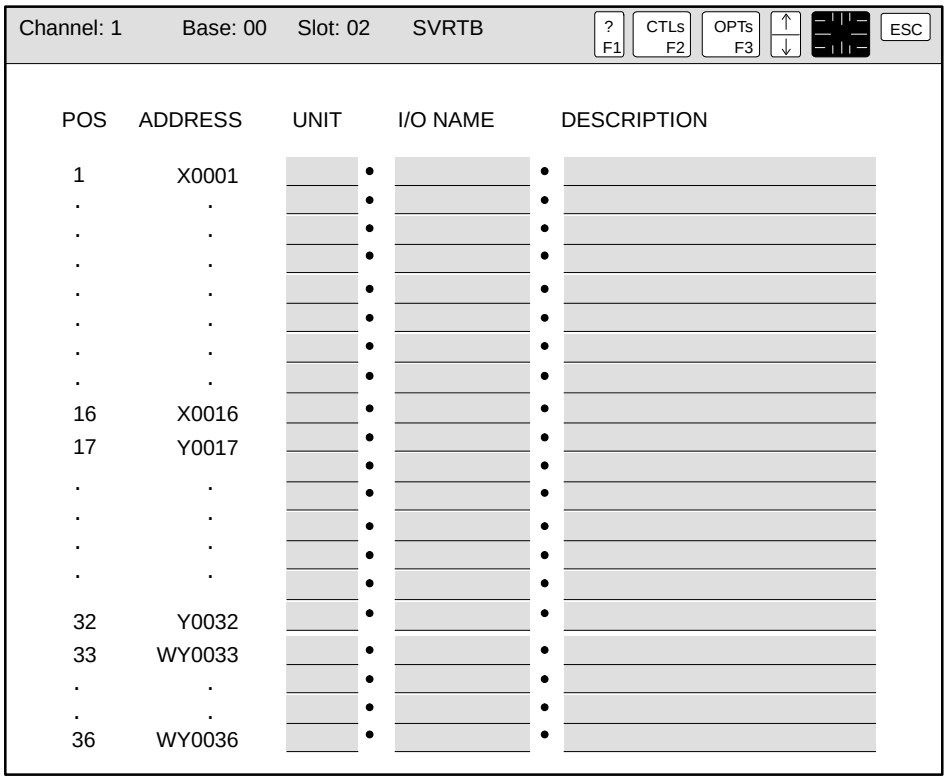


Figure 4-68 Series 505 FIM Module: SIMOVERT Broadcast Mode

Intelligent Modules (continued)

Communications Processor (Series 505: H1)

The Series 505 Communications Processor Module (CP1434TF) can be used for both controller-to-controller communication (peer) and supervisory host to controller communication (TF) on the Industrial Ethernet. The information transferred and the logic to perform the transfer have to be configured elsewhere in APT. The module editor defines the CP module. Figure 4-69 shows the form that you use to define the module.

The module uses four WXs and four WYs. They are 16-bit words, in which the bits are used for handshaking. The WYs contain the start bits for 40 jobs and the WXs contain the corresponding status bits for the 40 jobs. The bits have to be manipulated for communication to occur. The last WX and WY are not currently used.

APT appends the default prefix that you specify to the status words (xxxxx in Figure 4-69). For information about the meaning of these variables, see the manual that comes with the CP1434TF communications processor.

Slots required:	2	double-wide
Address type:	WX	(4 consecutive locations)
	WY	(4 consecutive locations)
I/O symbolic name type:	WI	(word input)
	WO	(word output)

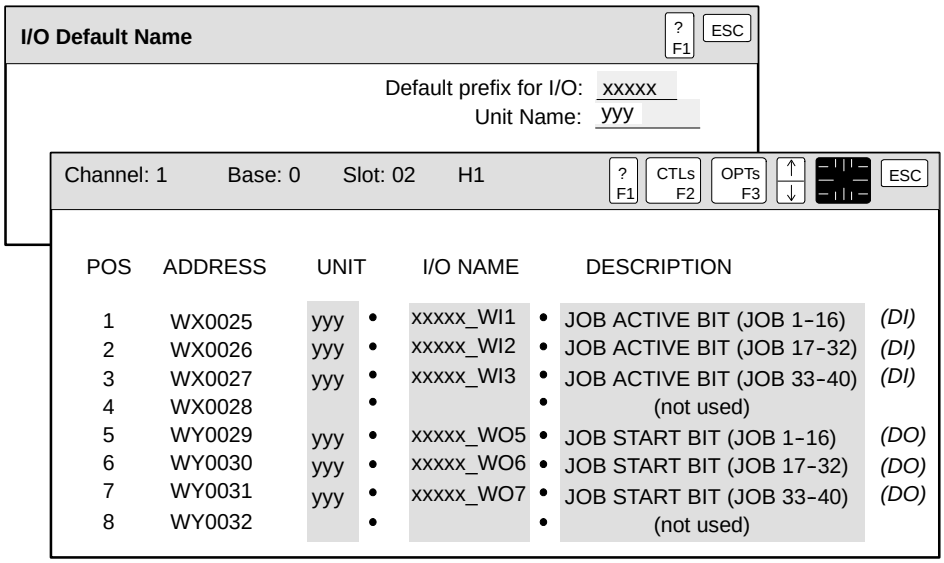


Figure 4-69 Series 505 Communication Processor

High Density Advanced Function (Series 505: 16AF)

The Series 505 High Density Advanced Function (16AF) Module provides 16 channels and performs processing on the analog signals before transferring that information to the programmable controller. This module supports scaling, alarming, peak/valley hold, averaging, and digital filtering for each of its 16 channels. Refer to the module's user manual for more details concerning the operation of the module.

Figure 4-70 shows the form that you use to define a Series 505 High Density Advanced Function (16AF) Module.

Slots required:	1	(user-configurable)
Address type:	X	(16 consecutive locations)
	Y	(16 consecutive locations)
	WX	(32 consecutive locations)
	WY	(32 consecutive locations)
I/O symbolic name type:	DI	(digital input)
	DO	(digital output)
	WI	(word input)
	WO	(word output)

Intelligent Modules (continued)

Channel: 1 Base: 00 Slot: 01 16AF

? F1 | CTLs F2 | OPTs F3 | ↑ ↓ | ESC

POS	ADDRESS	UNIT	I/O NAME	DESCRIPTION
16	X0016			Module ready flag
17	Y0017			Data Transfer Type Bit 1
18	Y0018			Data Transfer Type Bit 2
19	Y0019			Data Transfer Type Bit 3
20	Y0020			(not used)
.	.			(not used)
.	.			(not used)
.	.			(not used)
27	Y0027			Averaging reset (all channels)
28	Y0028			Valley hold reset (all channels)
29	Y0029			Peak hold reset (all channels)
30	Y0030			Read peak hold or valley hold
31	Y0031			Read flags or peak/valley hold
32	Y0032			Data ready flag
33	WX0033			Channel 1 conversion data
34	WX0034			Channel 2 conversion data
35	WX0035			Channel 3 conversion data
36	WX0036			Channel 4 conversion data
37	WX0037			Channel 5 conversion data
38	WX0038			Channel 6 conversion data
39	WX0039			Channel 7 conversion data
40	WX0040			Channel 8 conversion data
41	WX0041			Channel 9 conversion data
42	WX0042			Channel 10 conversion data
43	WX0043			Channel 11 conversion data
44	WX0044			Channel 12 conversion data
45	WX0045			Channel 13 conversion data
46	WX0046			Channel 14 conversion data
47	WX0047			Channel 15 conversion data
48	WX0048			Channel 16 conversion data
49	WX0049			
.	.			
.	.			
.	.			
64	WX0064			
65	WY0065			
.	.			
.	.			
.	.			
96	WY0096			

Figure 4-70 Series 505 High Density Advanced Function Module

Ethernet TCP/IP Adapter (Series 505: ENET)

The Series 505 Ethernet TCP/IP Adapter (ENET) Module provides communication capability using the Transmission Control Protocol/Internet Protocol (TCP/IP).

APT appends the default prefix that you specify to the status words. Refer to the module's user manual for more details concerning the operation of the module. APT does not communicate with the controller by means of this module. APT does not support program download, debug, MAITT, or any other APT communication functions with this module.

Figure 4-71 shows the form that you use to define a Series 505 Ethernet TCP/IP Adapter (ENET) Module.

Slots required:	1	(user-configurable)
Address type:	WX	(2 consecutive locations)
	WY	(6 consecutive locations)
I/O symbolic name type:	WI	(word input)
	WO	(word output)

Channel: 1 Base: 00 Slot: 01 ENET ? F1 CTLs F2 OPTs F3 ESC

POS	ADDRESS	UNIT	I/O NAME	DESCRIPTION
1	WX0001		XXXXX_M_STAT	Module Status Word
2	WX0002		XXXXX_C_STAT	Command Status Word
3	WY0003		XXXXX_M_CTRL	Module Control Word
4	WY0004		XXXXX_C_CTRL	Command Control Word
5	WY0005		XXXXX_CMDSL1	Command Slot 1
6	WY0006		XXXXX_CMDSL2	Command Slot 2
7	WY0007		XXXXX_CMDSL3	Command Slot 3
8	WY0008		XXXXX_CMDSL4	Command Slot 4

Figure 4-71 Series 505 Ethernet TCP/IP Adapter Module

Intelligent Modules (continued)

High Speed Counter Encoder (Series 505: HSCE)

The Series 505 High Speed Counter Encoder Module is an intelligent module that contains its own microprocessor and memory. It provides a total of six counters and eight outputs, divided into two channels.

APT appends the default prefix that you specify to the status words. Refer to the module's user manual for more details concerning the operation of the module.

Figure 4-72 shows the form that you use to define a Series 505 High Speed Counter Encoder (HSCE) Module.

Slots required:	1	(user-configurable)
Address type:	WX	(18 consecutive locations)
	WY	(14 consecutive locations)
I/O symbolic name type:	WI	(word input)
	WO	(word output)

Channel: 1 Base: 00 Slot: 01 HSCE

? F1 | CTLs F2 | OPTs F3 | ↑ ↓ | ESC

POS	ADDRESS	UNIT	I/O NAME	DESCRIPTION
1	WX0001		XXXXX_C1CCVH	Ctr 1 Current Count Value High
2	WX0002		XXXXX_C1CCVL	Ctr 1 Current Count Value Low
3	WX0003		XXXXX_C2CCV	Ctr 2 Current Count Value
4	WX0004		XXXXX_C3CCV	Ctr 3 Current Count Value
5	WX0005		XXXXX_C4CCVH	Ctr 4 Current Count Value High
6	WX0006		XXXXX_C4CCVL	Ctr 4 Current Count Value Low
7	WX0007		XXXXX_C5CCV	Ctr 5 Current Count Value
8	WX0008		XXXXX_C6CCV	Ctr 6 Current Count Value
9	WX0009		XXXXX_C1HR1H	Ctr 1 Holding Register 1 High
10	WX0010		XXXXX_C1HR1L	Ctr 1 Holding Register 1 Low
11	WX0011		XXXXX_C1HR2H	Ctr 1 Holding Register 2 High
12	WX0012		XXXXX_C1HR2L	Ctr 1 Holding Register 2 Low
13	WX0013		XXXXX_C4HR3H	Ctr 4 Holding Register 3 High
14	WX0014		XXXXX_C4HR3L	Ctr 4 Holding Register 3 Low
15	WX0015		XXXXX_C4HR4H	Ctr 4 Holding Register 4 High
16	WX0016		XXXXX_C4HR4L	Ctr 4 Holding Register 4 Low
17	WX0017		XXXXX_STATHI	Module Status, High Word
18	WX0018		XXXXX_STATLO	Module Status, Low Word
19	WY0019		XXXXX_SETHI	Setup Word, High Word
20	WY0020		XXXXX_SETLO	Setup Word, Low Word
21	WY0021		XXXXX_C1P1H	Ctr 1 Preset 1 High Word
22	WY0022		XXXXX_C1P1L	Ctr 1 Preset 1 Low Word
23	WY0023		XXXXX_C1P2H	Ctr 1 Preset 2 High Word
24	WY0024		XXXXX_C1P2L	Ctr 1 Preset 2 Low Word
25	WY0025		XXXXX_C2P3	Ctr 2 Preset 3 Word
26	WY0026		XXXXX_C3P4	Ctr 3 Preset 4 Word
27	WY0027		XXXXX_C4P5H	Ctr 4 Preset 5 High Word
28	WY0028		XXXXX_C4P5L	Ctr 4 Preset 5 Low Word
29	WY0029		XXXXX_C4P6H	Ctr 4 Preset 6 High Word
30	WY0030		XXXXX_C4P6L	Ctr 4 Preset 6 Low Word
31	WY0031		XXXXX_C5P7	Ctr 5 Preset 7 Word
32	WY0032		XXXXX_C6P8	Ctr 6 Preset 8 Word

Figure 4-72 Series 505 High Speed Counter Encoder

Intelligent Modules (continued)

Program Port Expander (Series 505: PPEXP)

The Series 505 Program Port Expander (PPEXP) Module provides additional RS-232C and RS-422 communications ports for the programmable controller to use. Refer to the module's user manual for more details concerning the operation of the module.

Figure 4-73 shows the form that you use to define a Series 505 Program Port Expander (PPEXP) Module.

Slots required:	1	(user-configurable)
Address type:	WX	(2 consecutive locations)
	WY	(6 consecutive locations)
I/O symbolic name type:	WI	(word input)
	WO	(word output)

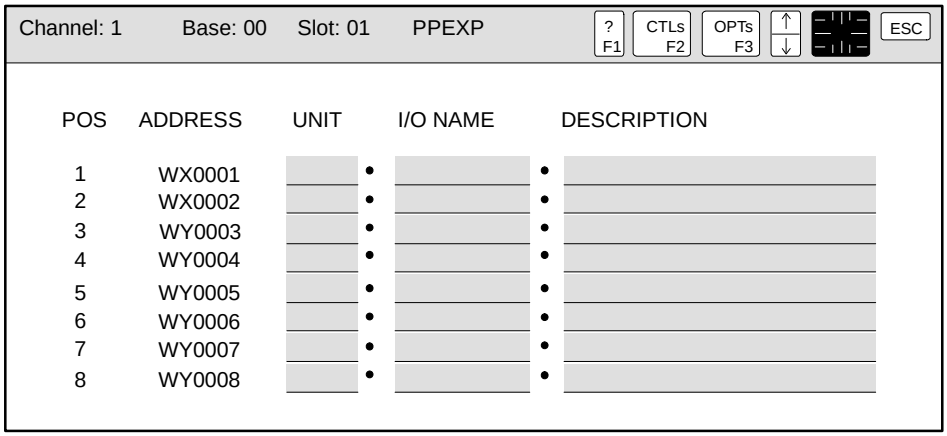


Figure 4-73 Series 505 Program Port Expander Module

Chapter 5

I/O Symbolic Names

5.1 Defining I/O Symbolic Names

Overview

The APT I/O Symbolic Name Definition Utility lets you define I/O symbolic names that you then use to refer to I/O points in your system.

The I/O Symbolic Name Definition Utility lets you identify various characteristics of I/O points, including the type of input or output, high and low input ranges, and the rates at which variables are filtered.

You can enter the I/O Symbolic Name Table at either the Program Content Level or the Unit Content Level. A symbolic name that is defined at the Program Content Level is considered global and can be used in any unit in that program. A symbolic name that is defined at the Unit Content Level can be used only within that unit.

Availability

The I/O Symbolic Name Definition Utility is supported for both Series 505 and S5 controllers. The Module Editor Utility is not supported for S5 controllers; use the I/O Symbolic Name Definition Utility to define your I/O symbolic names if you have an S5 controller.

Entering I/O Symbolic Names

When you define an I/O symbolic name, you enter information in the following fields.

Name can be from 1 to 12 characters long and can consist of any combination of letters, digits, and underscores. At least one of the characters must be a letter.

The name must be unique within the current program or unit, depending on where you are defining the I/O point. It cannot be the same as the name of any other I/O point, nor can it be the same as the name of any other object (declaration, device, etc.) within its scope. A symbolic name cannot be an APT keyword. See Chapter 1 for a list of key words.

Type is a two-letter code that represents the type of I/O point that you are defining.

Other Information that you need to enter appears on the screen after you specify the name and type. This information varies depending on the type of I/O point that you are defining.

You cannot edit the Name or Type fields if you have already marked the I/O name for OSx (PCS) tag translation. Additional fields for thermocouple (TC), resistance temperature detector (RT), and analog input (AI) I/O types cannot be edited after the name is marked for translation. These fields are listed under the descriptions for these I/O types.

5.2 I/O Types

Analog I/O

Analog values are 16-bit words that evaluate to one distinct signal. APT provides two types of analog I/O symbolic names: analog input (AI) and analog output (AO).

Availability

AI and AO are available for Series 500 I/O, Series 505 I/O, and S5 I/O. The symbol names and their use in APT are the same regardless of your controller type; however, the actual I/O addresses are different.

Analog I/O are typically used with CFBs, as shown in Figure 5-1, but they can also be monitored and changed in an SFC.

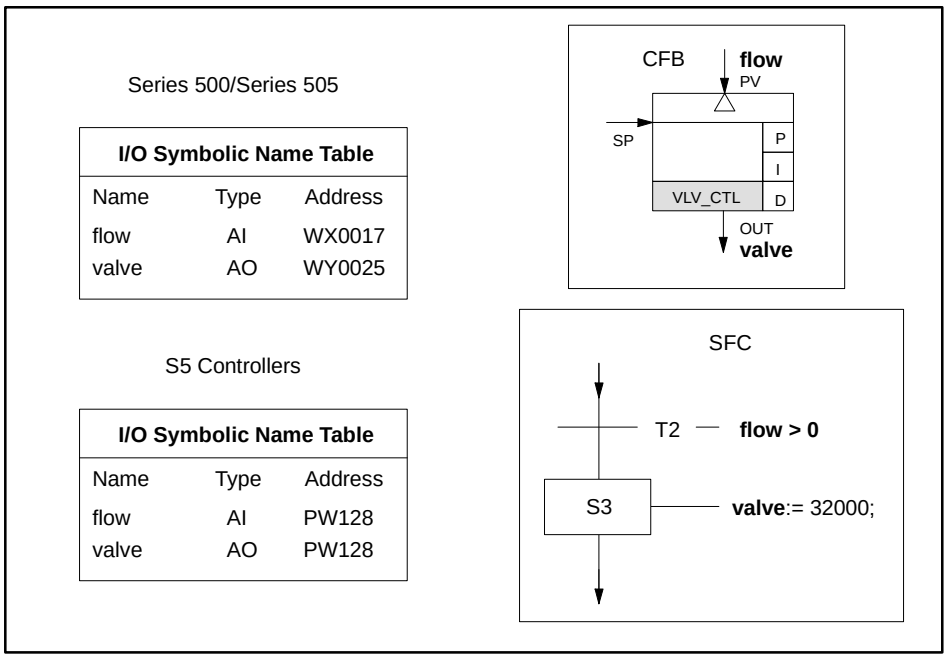


Figure 5-1 Using Analog I/O

I/O Types (continued)

Digital I/O

Digital I/O values are one-bit discrete values that evaluate to True or False. APT provides three types of digital I/O symbolic names: digital input (DI), digital output (DO), and digital flag (DF).

Availability

DI, DO, and DF are available for Series 500 I/O, Series 505 I/O, and S5 I/O. The symbol names and their use in APT are the same regardless of your controller type; however, the actual I/O addresses are different.

Digital I/O symbolic names can be used in SFC steps and transitions to detect and change the status of field equipment.

Figure 5-2 shows the symbolic name assignments used in an SFC that checks the status of a digital input, writes to a digital output, and turns on a digital flag. Figure 5-2 also shows how to use digital I/O in a CFB Math statement. (Digital flags cannot be used on the left side of an assignment statement.)

NOTE: A digital flag contains logic that controls an external bit, that is, the flag is a logical construct that manipulates a digital output. A digital flag requires the commands LATCH, CLEAR, and ON to set its logic state. This is different from an S5 flag, which controls a single internal bit and has no associated logic or commands.

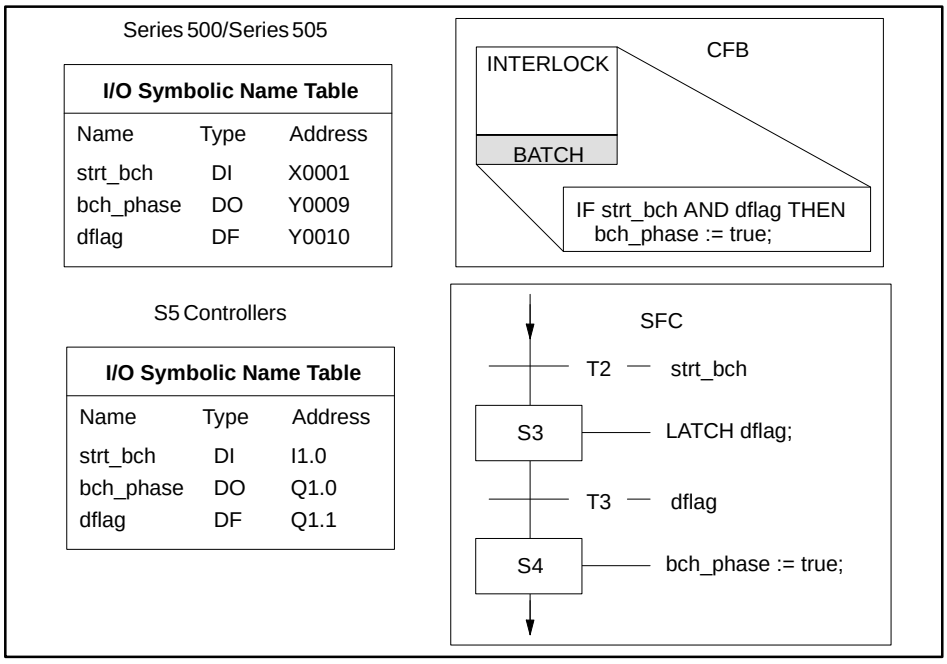


Figure 5-2 Using Digital I/O

Word I/O

Word I/O values are 16-bit values that contain 16 different bits of information. APT provides four types of word I/O symbolic names: word input (WI), word output (WO), binary-coded decimal input (BI), and binary-coded decimal output (BO).

Availability

WI, WO, BI, and BO are available for Series 500 I/O, Series 505 I/O, and S5 I/O. The symbol names and their use in APT are the same regardless of your controller type; however, the actual I/O addresses are different.

If you have a Series 505 controller, and you configure a parallel word module in the Module Definition Table before you define the I/O symbolic name, the type defaults to word input or word output. You can change the type to binary-coded decimal in the I/O Symbolic Name Table.

Intelligent I/O

For Series 500 I/O and Series 505 I/O, APT provides two special input types that are used with intelligent I/O modules, the Resistance Temperature Detector (RT) and the Thermocouple (TC).

Availability

RT and TC inputs are only available for Series 500 I/O and Series 505 I/O.

The Image Register Option

Series 505 controllers have an image register for digital and word I/O in their firmware. S5 controllers have an image register that handles digital I/O in their firmware (the PII and PIQ area); in addition, APT has provided the Image Register option in the I/O Table for S5 peripheral I/O.

If you have an S5 controller, you must select (**X**) the Image Register option when you want to do the following:

- Test your program when your peripheral I/O is not physically connected.
- Translate your peripheral I/O to OSx (PCS).
- Evaluate your peripheral I/O using Debug.

You do not have to deselect the Image Register option when you connect your I/O; actual physical I/O values automatically take precedence over the values contained by the image register.

NOTE: When your output module is not physically connected, and you wish to use the Image Register, you must program an error OB. Use a system-activated subroutine (described in Chapter 10) to program OB 23, 24, or 25. You can put any necessary code in the math section of the subroutine, or else simply enter a BEGIN statement. If you do not program an error OB, the controller goes into STOP mode. You do not have to program an error OB for input modules.

5.3 Analog Input

Using Analog Input (AI)

An analog input (AI) is a voltage or current signal from the field to the controller that varies continuously over a specified range of voltages or currents. Analog inputs normally come from measuring devices, such as flow meters and pressure transmitters, and are proportional to the measured parameter. The three scaling options that are available for analog inputs are described at the end of this section.

Availability

Both the Series 505 and S5 controllers can scale, calculate square roots, and filter an analog input. The 560/560T controllers, which do not support floating point calculations, cannot do these functions.

- The **No Scaling** option causes no scaling to be done, and the raw unprocessed input from the module is used. Neither filtering nor the square root functions can be done on the analog input.

  For S5 I/O, when you choose any scaling option (that is, when you do not select No Scaling), the analog input or output is bit-shifted by the SCALE or UNSCALE procedure. See the chapter called "Math Functions and Procedures" in the *SIMATIC APT Programming Reference (Graphics/Math) Manual* for more information. When an S5 analog input is used as the process variable for a PID loop, analog alarm block, or on/off block, the analog input must be scaled.

  For Series 500 and Series 505 I/O, if an analog input is used as the process variable for any of the standard or advanced control blocks (PID loop, analog alarm, dual mode, etc.) then the algorithms for these blocks can do a square root and an internal scaling based on the values of the process variable low and high ranges (PVL and PVH).

NOTE: For Series 500 and Series 505 I/O, do not select the **No Scaling** option for the uncontrolled (wild) process variable for a ratio station block. You can select **No Scaling** for the controlled process variable, however.

For Series 500 and Series 505 I/O, if you choose an analog input for the process variable for any standard or advanced control block and also select the **No Scaling** option, then you may still use the PID **.IN** or **.PV** extensions just like a scaled analog input.

If you do not select the **No Scaling** option: When you change the values of the process variable low or high ranges (PVL and PVH) for standard or advanced blocks during program execution, the range for the analog input (*loop_name*.**IN** or AI *name*) does not change correspondingly and therefore will no longer reflect the same value as the **.LPV**. If you want to change the high and low values you must change them on the AI form, recompile and download.

- The **square root** option normalizes the incoming signal to a value between 0 and 1, takes the square root, and then scales the value as shown below for bipolar values.

  0 to 32000 ⟶ 0 to 1 ⟶ $\sqrt{value}$ ⟶ scale

  -32000 to 0 ⟶ -1 to 0 ⟶ $-\sqrt{|value|}$ ⟶ scale

- The **filter** option applies a first-order exponential filter at a rate determined by a specified time constant. If the filter is used with the square root option, the filter is applied before the square root. The following formula is used to process the value every half second:

  $$name = (e^{\frac{-.5}{ftau}})name + (1 - e^{\frac{-.5}{ftau}})name.srv$$

You can use an AI symbolic name in an SFC step or Math statement or in a CFB that uses an analog input. For example, the process variable in a PID loop is normally an analog input. An AI symbolic name has the extensions listed in Table 5-1.

Table 5-1 AI Extensions

Extension	Definition
name[1]	scaled[2], processed (filtered) real value (read-only)
name.**BTA**[3]	broken transmitter flag (read-only)
name.**FLTR**[1]	filtered value (read-only)
name.**FTAU**[1]	time constant that indicates rate for filter (read/write)
name.**IO**	image register value (Image Register option[3])(read-only)
name.**OVR**[3]	overrange flag (read-only)
name.**RAW**	raw, integer input from module (read-only)
name.**SRV**[1]	raw, scaled[2] value before filter (read-only)

1 Used by Series 505 and S5 but not 560/560T controllers.
2 All but the 560/560T controllers can scale inputs.
3 Used by S5 only.

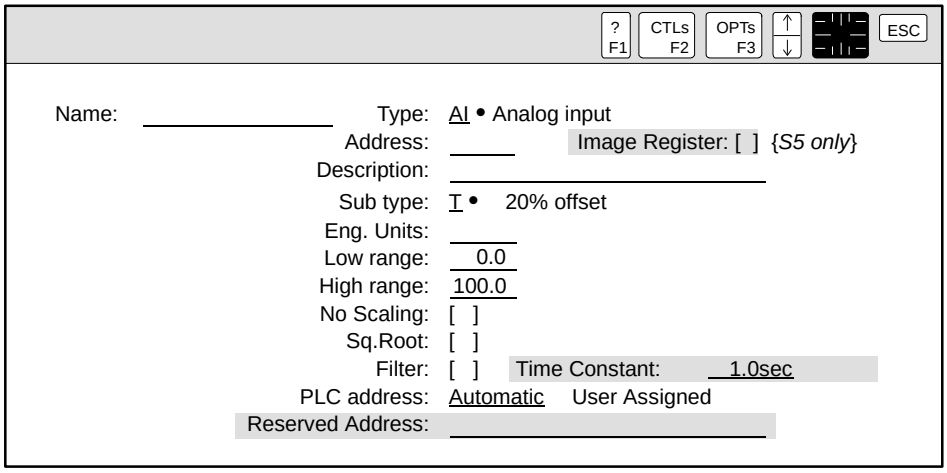
When you reference the symbolic name of an AI in your program, APT uses the scaled value (for all but the 560/560T). The scaling of the analog input is updated each time the associated math is executed. Any program attempts to modify this scaled value are overwritten by the system. The 560/560T controllers, which do not scale inputs, use the raw integer input, *name*.**RAW**, from an analog module.

When you mark the AI object name for OSx (PCS) tag translation in the Mark PCS Tags subeditor, you cannot edit the Name and Type fields. Furthermore, if you mark the AI object name for translate without using the .**RAW** extension, you also cannot edit the No Scaling field. The AI object name without the .**RAW** extension translates to an OSx **CALC** data type, whereas *object_name*.**RAW** translates to an OSx AI data type.

Analog Input (continued)

Defining AI

When you select Analog Input, this form appears:



Name: unique, symbolic name (12 characters maximum).

Type: 2-letter code for Analog Input.
Series 500/505 modules that use this I/O type include 4AI, 8AI, 4AI4O, 6AI2O, 8AI4O.
S5 modules that use this I/O type include the 460, 465, and 466.

Address: you can only edit this field if you have an S5 controller. For S5, valid addresses are %PW0-254, %OW0-254, and %IW0-126. The most commonly used area for analog inputs is %PW128-254. The % sign must be included in the address. For Series 500/505, addresses are not assigned on this form; they are assigned in the Module Definition Table.

Image Register: this field only appears for S5 controllers. When the image register is selected (X), you can test your program without connecting up your I/O. You can also translate your AI to OSx. If your I/O is connected, the image register is updated with the I/O values every program scan. You do not need the Image Register option in order to test programs with I/O disconnected for Series 500/505; the controller already has that ability.

Description: optional (30 characters maximum).

Sub type: code (B, T, or Z) that represents voltage or current range of input to the Series 500/505 or S5 module as listed below.

(Series 500/505)
20% offset (T) is the default.
- **B** (Bipolar): -10 to +10 V or -5 to +5 V
- **T** (20% offset): 1 to 5 V, 2 to 10 V, or 4 to 20 mA
- **Z** (Zero bias): 0 to 5 V, 0 to 10 V, or 0 to 20 mA

(S5)
20% offset (T) is the default.
- **B** (Bipolar): ±12 mV, ±50 mV, ±500 mV, ±1 V, ±5 V, ±10 V, ±20 mA
- **T** (20% offset): 4 to 20 mA
- **Z** (Zero bias): 0 to 12 mV, 0 to 50 mV, 0 to 500 mV, Pt100, 0 to 1 V, 0 to 5 V, 0 to 10 V, or 0 to 20 mA

Engineering Units: optional documentation field to describe units of final, scaled value (8 characters maximum).

Low Range (all but the 560/560T)**:** real number that indicates low range of input in engineering units. Value must be less than high range. A value must be entered for all controllers, but is used by all but the 560/560T.

High Range (all but the 560/560T)**:** real number that indicates high range of input in engineering units. Value must be greater than low range. A value must be entered for all controllers, but is used by all but the 560/560T.

No Scaling (all but the 560/560T)**:** X indicates no scaling is to be done on the input. Scaling is not possible for the 560/560T.

Square Root (all but the 560/560T)**:** X indicates square root of input is used. Select this option if the input for the process variable comes from a device (such as an orifice meter) that requires a square root calculation to scale its value.

Filter (all but the 560/560T)**:** X indicates the input is filtered.

Time Constant (all but the 560/560T)**:** real number that indicates rate in seconds for filter; field appears only if you select **Filter**.

PLC Address (all but the 560/560T)**:** Refers to controller address allocation for the scaled value as listed below. Automatic is the default.

> **Automatic:** APT automatically assigns the controller address
> **User Assigned**: Allows user to specify a controller address for the scaled value.

Reserved Address (all but the 560/560T)**:** starting controller address you specify for the scaled value of the analog input.

(Series 500/505)
The types of memory available for this declaration are %V, %G, and %Gx (where x is the application ID). See your controller manual for the types of memory your controller supports. Memory types %G and %Gx are only available for a 575 controller.

(S5)
The types of memory available for this declaration are %DB:DD or %DX:DD.

Analog Input (continued)

Scaling Analog Values

Series 505 controller analog values are scaled differently from S5 controller analog values. For Series 500/Series 505 I/O, three scaling options are available for analog input and output values: bipolar, 20% offset, and zero bias. See Table 5-2.

Table 5-2 Scaling Series 500/505 Analog I/O

Scale Option	Input Range	Low Range		High Range	
		Input Value	Integer Value	Input Value	Integer Value
Bipolar	-10 to +10 volts	-10	-32000	10	32000
	-5 to +5 volts	-5	-32000	5	32000
20% Offset	1 to 5 volts	1	6400	5	32000
	2 to 10 volts	2	6400	10	32000
	4 to 20 milliamps	4	6400	20	32000
Zero Bias	0 to 5 volts	0	0	5	32000
	0 to 10 volts	0	0	10	32000
	0 to 20 milliamps	0	0	20	32000

For S5, the scaling of the analog inputs are handled differently from the scaling of outputs. Three scaling options are available for analog input values: bipolar, 20% offset, and zero bias. Analog inputs are allowed up to 200% overranging. The following measuring ranges are supported: ±12 mV, ±50 mV, ±500 mV, Pt100, ±1 V, ±5 V, ±10 V, ±4-20 mA (with 4 and 2 wire transducers). Table 5-3 lists examples of the different ranges.

Table 5-3 Scaling S5 Analog Inputs

Scale Option	Input Range	Low Range			High Range		
		Input Value	Integer Value	APT .RAW Value[1]	Input Value	Integer Value	APT .RAW Value[1]
Bipolar	-10 to +10 volts	-10	-2048	-16384	10	2048	16384
	-5 to +5 volts	-5	-2048	-16384	5	2048	16384
20% Offset	4 to 20 milliamps	4	512	4096	20	2560	20480
Zero Bias	0 to 5 volts	0	0	0	5	2048	16384
	0 to 10 volts	0	0	0	10	2048	16384
	0 to 20 milliamps	0	0	0	20	2048	16384
	PT100 0 to 200 ohms	0	0	0	200	2048	16384

1 Analog inputs are 16-bit numbers with 12 bits of resolution. The three bits that are taken off contain error information on overranging and broken transmitter. For test purposes you must multiply your 0 to 2048 range by 8 to use as the analog_name.RAW.

5.4 Analog Output

Using Analog Output (AO)

Analog Output (AO) is a signal from the controller to a process control device that provides modulation information for field equipment. Analog outputs normally go to analog display meters and proportional control valves.

Availability

Analog outputs are available for Series 500 I/O, Series 505 I/O, and S5 I/O. The symbol names and their use in APT are the same regardless of your controller type; however, the actual I/O addresses are different.

The value of an AO is always an integer. When you configure an AO, you must specify the scaling option. Series 500 and Series 505 modules can choose between three options: zero offset, 20% offset, and bipolar. S5 modules can choose between two options: zero offset and bipolar. The scaling range of an option is different for the different controller families: for instance, a bipolar value of -10 volts would correspond to an AO value of -32000 for a Series 505 controller, whereas a bipolar value of -10 volts would correspond to an AO value of -1024 for an S5 controller.

The scaling options are used only when the AO is configured as the output of a loop. If you write directly to the AO from a math block, you must use the UNSCALE function to get a properly scaled output.

An AO symbolic name can be used in an SFC step or Math statement or can be configured in a CFB that uses an analog output. For example, the output from a PID loop is normally an analog output. An AO symbolic name has the extensions listed in Table 5-4.

Table 5-4 AO Extensions

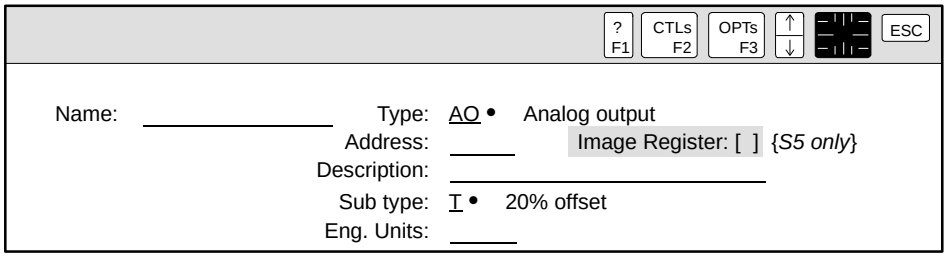
Extension	Definition
name	integer value (read-only)
name.**IO**	image register value (Image Register option[1]) (read-only)
name.**OVR**	overrange occurred during scaling[1] (read-only)

1 Used by S5 only.

Analog Output (continued)

Defining AO

When you select Analog Output, this form appears:



Name: unique, symbolic name (12 characters maximum).

Type: 2-letter code for Analog Output.
Series 500/505 modules that use this I/O type include 2AO, 4AO, 8AO, 4AI40, 6AI20, 8AI40.
The S5 module that uses this type is the 470.

Address: you can only edit this field if you have an S5 controller. For S5, valid addresses are %PW0-254, %OW0-254, and %QW0-126. The most commonly used area for analog outputs is %PW128-254. The % sign must be included in the address. For Series 500/505, addresses are not assigned on this form; they are assigned in the Module Definition Table.

Image Register: this field only appears for S5 controllers. When the image register is selected (X), you can test your program without connecting up your I/O. You can also translate your AO to OSx and evaluate your AO using Debug. However, if your AO module is not connected, then the error OB must be programmed, or else the controller will go into STOP mode. You do not need the Image Register option in order to test programs with I/O disconnected for Series 500/505; the controller already has that ability.

Description: optional (30 characters maximum).

Sub type: code (B, T, or Z) that represents type of output to the field as listed below.

(Series 500/505)
20% (T) is the default.
- **B** (Bipolar): -10 to +10 V or -5 to +5 V
- **T** (20% offset): 1 to 5 V, 2 to 10 V, or 4 to 20 mA
- **Z** (Zero bias): 0 to 5 V, 0 to 10 V, or 0 to 20 mA

(S5)
Zero bias (Z) is the default.
- **B** (Bipolar): -10 V to 10 V
- **T** (20% offset): NOT AVAILABLE
- **Z** (Zero bias): 0 to 20 mA, 0 to 10 V, 0 to 5 V, or 4 to 20 mA

NOTE: If your S5 analog output module has a built-in 20% offset, you must select zero bias. If your module does not have a 20% offset built in, you can choose between zero bias and bipolar. You cannot choose 20% offset.

Engineering Units: optional documentation field to describe units of final value (8 characters maximum).

For Series 500/Series 505 I/O, three scaling options are available for analog output values: bipolar, 20% offset, and zero bias. See Table 5-2.

For S5 analog outputs, two scaling options are available; bipolar and zero bias. A 20% offset module is entered as a zero bias module, because the offset is done in the modules themselves. Table 5-5 lists the scaling options available for S5 analog outputs.

Table 5-5 Scaling S5 Analog Outputs

Scale Option	Input Range	Low Range			High Range		
		Input Value	Integer Value	APT Value	Input Value	Integer Value	APT Value
Bipolar	-10 to +10 volts	-10	-1024	-16384	10	1024	16384
	-5 to +5 volts	-5	-1024	-16384	5	1024	16384
Zero Bias (Includes 20% Offset)	0 to 5 volts	0	0	0	5	1024	16384
	0 to 10 volts	0	0	0	10	1024	16384
	1 to 5 volts	1	0	0	5	1024	16384
	2 to 10 volts	2	0	0	10	1024	16384
	0 to 20 milliamps	0	0	0	20	1024	16384
	4 to 20 milliamps	4	0	0	20	1024	16384

1 Analog outputs are 16-bit numbers with 11 bits of resolution. Four bits are added to match the format of an analog input; they do not contain any information. For test purposes you must multiply your 0 to 1024 or +/- 1024 range by 16 to check the value that is placed in the analog output name.
2 If your S5 analog output module has a built-in 20% offset, you must select zero bias. If your module does not have a 20% offset built in, you can choose between zero bias and bipolar.

5.5 Digital Flag

Using Digital Flag (DF)

A Digital Flag (DF) is a read/write boolean value that can be used anywhere you might use a digital output and can be assigned to any digital output module.

Availability

DFs are available for Series 500 I/O, Series 505 I/O, and S5 I/O. The symbol names and their use in APT are the same regardless of your controller type; however, the actual I/O addresses are different.

NOTE: If you have Series 500 or Series 505 I/O and you define a DF in the I/O Symbolic Name Table before you define the DO module, the point remains a DF when you enter the name in the module. If you define the module first, the default is a digital output (DO), and you must change the type to DF in the I/O Symbolic Name Table.

A digital flag I/O symbolic name is used in the parallel section of an SFC step or a Math section with the commands listed in Table 5-6.

Table 5-6 DF Commands

Command	Definition
LATCH	Turns on output until CLEAR command turns it off.
ON	Turns on output as long as the SFC step or Math statement is active.
CLEAR	Turns off output; takes precedence over ON or LATCH.

Digital flags differ from boolean and DO (digital output) variables. All references to a digital flag are logically connected, and the flag state is set in one place in the compiled program. In contrast, an S5 flag controls a single internal bit and has no associated logic or commands. References to boolean or digital output (DO) variables can appear any number of places in the compiled code.

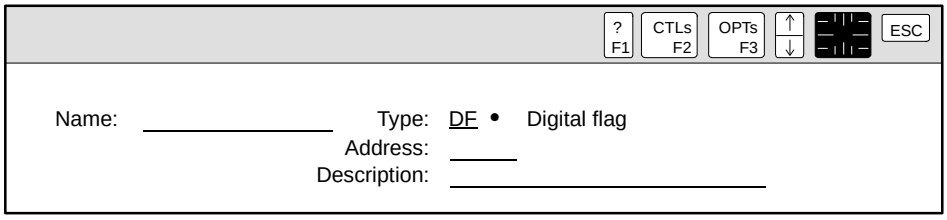
Use the following commands to turn on and off *s1_low,* a digital flag.

```
ON (s1_low);  LATCH (s1_low);  CLEAR (s1_low);
```

NOTE: Do not use a digital flag on the left side of an assignment statement.

Defining DF

When you select Digital Flag, this form appears:



Name: unique, symbolic name (12 characters maximum).

Type: 2-letter code for Digital Flag.
Series 500 and Series 505 modules that use this I/O type include 8DO, 16DO, 32DO, 110VO, 110VR, 120VR, 24VDC.
For S5, all digital output cards can use this type.

Address: you can only edit this field if you have an S5 controller. For S5, valid addresses are %Q0.0–127.7. The % sign must be included in the address. For Series 500/505, addresses are not assigned on this form; they are assigned in the Module Definition Table.

Description: optional (30 characters maximum).

NOTE: For S5 controllers, APT digital flags are stored in the PIQ area, so S5 controllers do not need the Image Register for DFs. If you want to test your S5 program without I/O connected, enter **No** to the "System stop if addressing error occurs" option in the Compiler Control file. After this selection is downloaded to the controller, you must do a COLDSTART.

5.6 Digital Input

Using Digital Input (DI)

Digital Input (DI) is a signal from the field to the controller that reflects the status of field equipment, such as the position of a valve or the input from a limit switch.

Availability

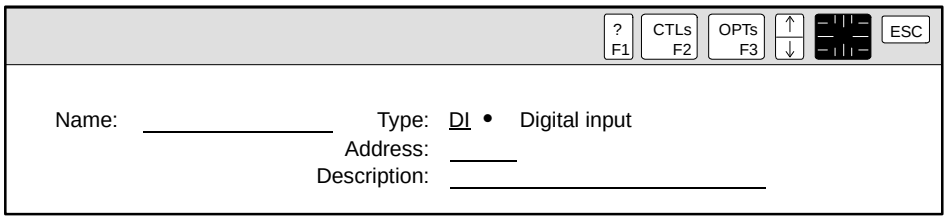
DIs are available for Series 500 I/O, Series 505 I/O, and S5 I/O. The symbol names and their use in APT are the same regardless of your controller type; however, the actual I/O addresses are different.

Digital Input is a read-only boolean value that is either true or false and represents a discrete value such as on/off, open/closed, or running/stopped.

A DI symbolic name can be used in an SFC transition to detect the status of equipment and can be configured as feedback for a field device. See Chapter 6 for information about inputs used as device feedback.

Defining DI

When you select Digital Input, this form appears:



Name: unique, symbolic name (12 characters maximum).

Type: 2-letter code for Digital Input.
Series 500 and Series 505 modules that use this I/O type include 8DI, 16DI, 32DI, 110VO, 110VR, 120VR, 24VDC, INTRP, SLICE, USER, SREGB, and SVRTB.
For S5, all digital input cards can use this type.

Address: you can only edit this field if you have an S5 controller. For S5, valid addresses are %I0.0–127.7. The % sign must be included in the address. For Series 500/505, addresses are not assigned on this form; they are assigned in the Module Definition Table.

Description: optional (30 characters maximum).

NOTE: For S5 controllers, digital inputs are stored in the PII area, so S5 controllers do not need the Image Register for DIs. If you want to test your S5 program without I/O connected, enter **No** to the "System stop if addressing error occurs" option in the Compiler Control file. After this selection is downloaded to the controller, you must do a COLDSTART.

5.7 Digital Output

Using Digital Output (DO)

Digital Output (DO) is a signal from the controller that changes the on/off state of field equipment.

Availability

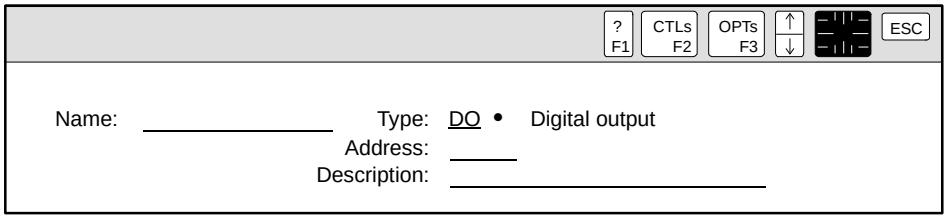
DOs are available for Series 500 I/O, Series 505 I/O, and S5 I/O. The symbol names and their use in APT are the same regardless of your controller type; however, the actual I/O addresses are different.

Digital Output is a read/write boolean value that is either true or false and represents discrete values such as on/off, open/shut, or start/stop.

A DO symbolic name can be used in an SFC step or Math statement to change the status of equipment. A DO can also be configured as the output for a field device. See Chapter 7 for information about device outputs.

Defining DO

When you select Digital Output, this form appears:



Name: unique, symbolic name (12 characters maximum).

Type: 2-letter code for Digital Output.
Series 500 and Series 505 modules that use this I/O type include 8DO, 16DO, 32DO, 110VO, 110VR, 120VR, 24VDC, NIM, HSPI, SLICE, USER, SREGB, and SVRTB.
For S5, all digital output cards can use this type.

Address: you can only edit this field if you have an S5 controller. For S5, valid addresses are %Q0.0–127.7. The % sign must be included in the address. For Series 500/505, addresses are not assigned on this form; they are assigned in the Module Definition Table.

Description: optional (30 characters maximum).

NOTE: For S5 controllers, digital outputs are stored in the PIQ area, so S5 controllers do not need the Image Register for DOs. If you want to test your S5 program without I/O connected, enter **No** to the "System stop if addressing error occurs" option in the Compiler Control file. After this selection is downloaded to the controller, you must do a COLDSTART.

5.8 Word Input

Using Word Input (WI)

Word Input (WI) is a signal from the field to the controller that is a read-only integer. Because it does not require special processing, Word Input does not include scaling or filtering information.

Availability

WIs are available for Series 500 I/O, Series 505 I/O, and S5 I/O. The symbol names and their use in APT are the same regardless of your controller type; however, the actual I/O addresses are different.

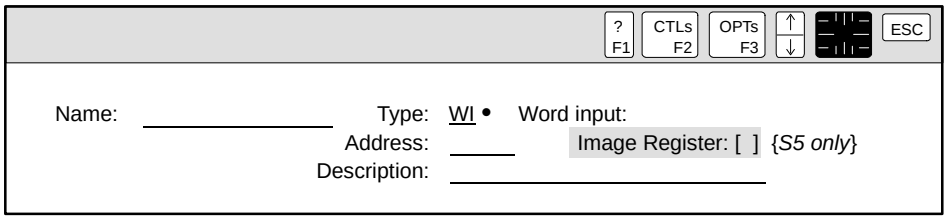
A WI symbolic name can be used in a CFB, in an SFC transition, or in a Math statement as the parameter to a function such as BITTEST. A WI symbolic name has the extensions listed in Table 5-7.

Table 5-7 WI Extensions

Extension	Definition
name	converted, integer value
name.IO	image register value (Image Register option[1])
1 Used by S5 only.	

Defining WI

When you select Word Input, this form appears:



Name: unique, symbolic name (12 characters maximum).

Type: 2-letter code for Word Input.
Series 500 and Series 505 modules that use this I/O type include 8WI, 16WI, ASCII, BASIC, DCOMM, HSPI, PLINK, SERVO, HSC, H1, USER, ATM, ESP, TURBO, HSPID, SREG, and SVRT.
For S5, all digital input cards can use this type.

Address: you can only edit this field if you have an S5 controller. For S5, valid addresses are %PW0-254, %OW0-254, and %IW0-126. The most commonly used area for word inputs is %IW0-126. The % sign must be included in the address. For Series 500/505, addresses are not assigned on this form; they are assigned in the Module Definition Table.

Image Register: this field only appears for S5 controllers. When the image register is selected (X), you can test your program without connecting up your I/O. You can also translate your WI to OSx. If your I/O is connected, the image register is updated with the I/O values every program scan. This option is not necessary in order to test programs with I/O disconnected for Series 500/505; the controller already has that ability.

Description: optional (30 characters maximum).

5.9 Word Output

Using Word Output (WO)

Word Output (WO) is a signal from the controller to a process-control device that is a read/write integer. Because it does not require special processing, Word Output does not include scaling or filtering information.

Availability

WOs are available for Series 500 I/O, Series 505 I/O, and S5 I/O. The symbol names and their use in APT are the same regardless of your controller type; however, the actual I/O addresses are different.

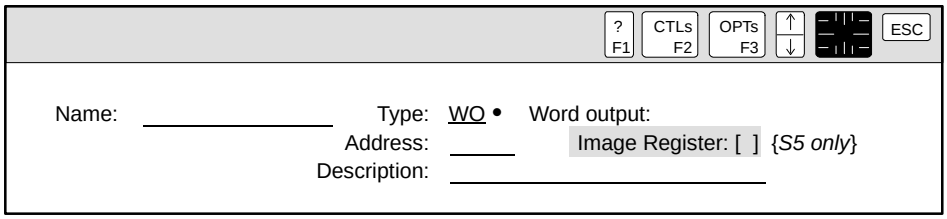
A WO symbolic name can be used in an SFC step, in a Math statement, or in a CFB. If you assign a value to a WO, the value must be an integer. A WO symbolic name has the extensions listed in Table 5-8.

Table 5-8 WO Extensions

Extension	Definition
name	converted, integer value
name.IO	image register value (Image Register option[1])
1 Used by S5 only.	

Defining WO

When you select Word Output, this form appears:



Name: unique, symbolic name (12 characters maximum).

Type: 2-letter code for Word Output.
Series 500 and Series 505 modules that use this I/O type include 8WO, 16WO, 8WOSF, ASCII, BASIC, DCOMM, HSPI, PLINK, SERVO, HSC, H1, USER, ATM, ESP, TURBO, HSPID, SREG, SREGB, SVRT, SVRTB.
For S5, all digital output cards can use this type.

Address: you can only edit this field if you have an S5 controller. For S5, valid addresses are %PW0-254, %OW0-254, and %QW0-126. The most commonly used area for word outputs is %QW0-126. The % sign must be included in the address. For Series 500/505, addresses are not assigned on this form; they are assigned in the Module Definition Table.

Image Register: this field only appears for S5 controllers. When the image register is selected (X), you can test your program without connecting up your I/O. You can also translate your WO to OSx and evaluate your WO using Debug. However, if your WO module is not connected, then the error OB must be programmed, or else the controller will go into STOP mode. This option is not necessary in order to test programs with I/O disconnected for Series 500/505; the controller already has that ability.

Description: optional (30 characters maximum).

5.10 Binary-coded Decimal Input

Using Binary-coded Decimal Input (BI)

Binary-coded Decimal Input (BI) is a signal from the field to the controller that represents a decimal number that has been coded into a binary representation.

Availability

BIs are available for Series 500 I/O, Series 505 I/O, and S5 I/O. The symbol names and their use in APT are the same regardless of your controller type; however, the actual I/O addresses are different.

Binary-coded decimal inputs are most commonly used to get thumbwheel information into the program. If you declare your BCD input as a BI I/O type, then APT converts the BCD value to an integer. The BI symbolic name contains this converted integer, which can be used in the math language of an SFC step or a CFB.

The BI symbolic name has the extensions listed in Table 5-9. Notice that the *name*.**RAW** contains the binary value, while the symbolic name contains the converted integer.

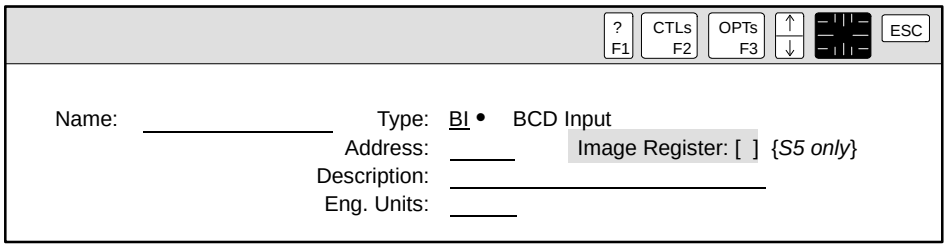
Table 5-9 BI Extensions

Extension	Definition
name	converted, integer value (read-only)
name.**IO**	image register value (Image Register option[1])(read-only)
name.**RAW**	raw, integer value of BCD input from module (read-only)

1 Used by S5 only.

Defining BI

When you select Binary-coded Decimal Input, this form appears, as shown below.



Name: unique, symbolic name (12 characters maximum).

Type: 2-letter code for Binary-coded Decimal Input.
The Series 500 and Series 505 module that uses this I/O type is 8WI.
For S5, this I/O type is used by digital input cards.

Address: you can only edit this field if you have an S5 controller. For S5, valid addresses are %PW0-254, %OW0-254, and %IW0-126. The most commonly used area for BCD inputs is %IW0-126. The % sign must be included in the address. For Series 500/505, addresses are not assigned on this form; they are assigned in the Module Definition Table.

Image Register: this field only appears for S5 controllers. When the image register is selected (X), you can test your program without connecting up your I/O. You can also translate your BI to OSx. If your I/O is connected, the image register is updated with the I/O values every program scan This option is not necessary in order to test programs with I/O disconnected for Series 500/505; the controller already has that ability.

Description: optional (30 characters maximum).

Engineering Units: optional documentation field to describe units of final value (eight characters maximum).

5.11 Binary-coded Decimal Output

Using Binary-coded Output (BO)

Binary-coded Decimal Output (BO) is a signal from the controller to a process-control device such as a BCD display. A Binary-coded Decimal Output is a binary number represented as a 16-bit integer and coded into a decimal representation.

Availability

BOs are available for Series 500 I/O, Series 505 I/O, and S5 I/O. The symbol names and their use in APT are the same regardless of your controller type; however, the actual I/O addresses are different.

A BO symbolic name is used as an output from a CFB or in an SFC step or Math statement. A BO symbolic name has the extensions listed in Table 5-10. Notice that the BO I/O type receives an integer output value and converts it to the BCD format before sending it to the BCD display.

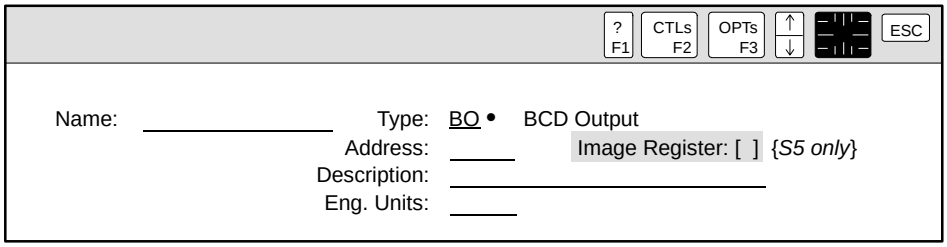
Table 5-10 BO Extensions

Extension	Definition
name	binary value of integer
name.IO	image register value (Image Register option[1])
name.OUT	integer that represents BCD value that is sent to the field (read only)

1 Used by S5 only.

Defining BO

When you select Binary-coded Decimal Output, this form appears:



Name: unique, symbolic name (12 characters maximum).

Type: 2-letter code for Binary-coded Decimal Output.
The Series 500 and Series 505 module that uses this I/O type is 8WO.
For S5, this I/O type is used by digital output cards.

Address: you can only edit this field if you have an S5 controller. For S5, valid addresses are %PW0-254, %OW0-254, and %QW0-126. The most commonly used area for BCD outputs is %OW0-126. The % sign must be included in the address. For Series 500/505, addresses are not assigned on this form; they are assigned in the Module Definition Table.

Image Register: this field only appears for S5 controllers. When the image register is selected (X), you can test your program without connecting up your I/O. You can also translate your BO to OSx and evaluate your BO using Debug. However, if your BO module is not connected, then the error OB must be programmed, or else the controller will go into STOP mode. This option is not necessary in order to test programs with I/O disconnected for Series 500/505; the controller already has that ability.

Description: optional (30 characters maximum).

Engineering Units: optional documentation field to describe units of final value (eight characters maximum).

5.12 Resistance Temperature Detector

Using Resistance Temperature Detector (RT)

Resistance Temperature Detector (RT) is a signal from the field to the controller that represents an analog-to-digital conversion of the input temperature. This is a symbolic name type that is used specifically for the inputs to an RTD module.

Availability

RTs are available for Series 500 and Series 505 I/O only.

An RT symbolic name can be used in an SFC step or Math statement or can be configured in a CFB. An RT symbolic name has the extensions listed in Table 5-11.

Table 5-11 RT Extensions

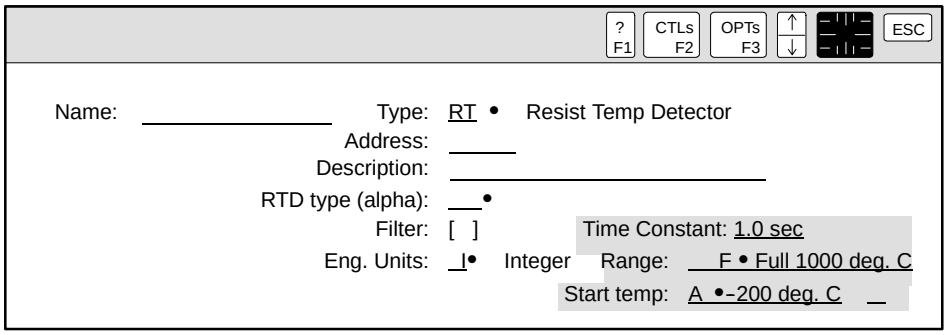
Extension	Definition
name[1]	scaled, processed (filtered) real value (read-only)
name.**FTAU** [1]	time constant that indicates rate for filter (read/write)
name.**RAW**	raw integer input from module (read-only)
name.**SRV** [1]	raw, scaled value before filter (read-only)
1 Not available for the 560/560T controllers.	

When you reference the symbolic name of an RT in your program, APT uses the scaled value for all but the 560/560T; the scaling is updated each time the associated SFPGM is executed. Any program attempts to modify this scaled value are overwritten by the system. The 560/560T controllers, which do not scale inputs, use the raw integer input, *name*.**RAW**, from an RTD module.

NOTE: Refer to the appropriate Series 500 or Series 505 RTD manual for configuration information.

Defining RT (Series 500)

When you select Resistance Temperature Detector (for the Series 500), this form appears:



Name: unique, symbolic name (12 characters maximum).

Type: 2-letter code for Resistance Temperature Detector.
The module that uses this I/O type is RTD.

Address: you cannot edit this field; address is assigned in the Module Definition Table.

Description: optional (30 characters maximum).

RTD type: code that represents mode of operation of RTD as listed below.

1	100 ohm Platinum,	0.003850 ohm/ohm/C
2	100 ohm Platinum,	0.003916 ohm/ohm/C
3	100 ohm Platinum,	0.003902 ohm/ohm/C
4	100 ohm Nickel,	0.006720 ohm/ohm/C
5	100 ohm Platinum,	0.003900 ohm/ohm/C

Filter: X indicates that you want to filter the input with a first-order exponential filter at a rate determined by a specified time constant.

Time Constant: real number that indicates rate in seconds at which you want to filter input; field appears only if you select **Filter**.

Engineering Units: code that represents the output format, as listed below; default is C.

C Degrees Centigrade
F Degrees Fahrenheit
O Ohms
I Integer

Integer Range: code that represents range of the temperature being monitored, as listed below; default is F; field appears if you select **Integer** for units.

F Full 1000 deg C.
H Half 500 deg C.
Q Quarter 250 deg C.
E Eighth 125 deg C.

Start Temp: code that represents lowest value in temperature range being monitored, as listed below; default is A; field appears if you select **Integer** for units.

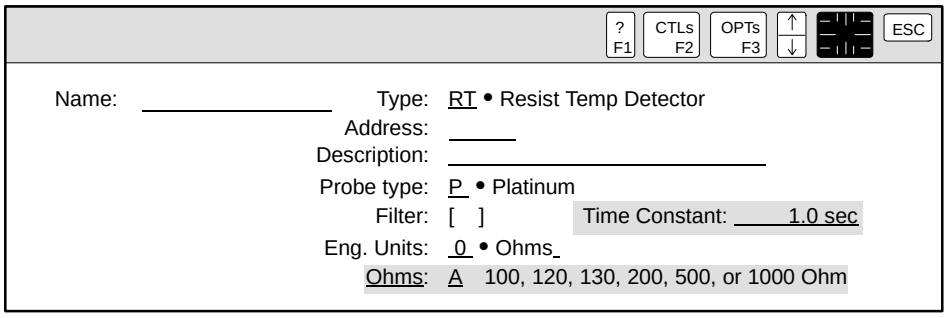
A -200 deg C.
E 300 deg C.
B -75 deg C.
F 425 deg C.
C 50 deg C.
G 550 deg C.
D 175 deg C.
H 675 deg C.

You cannot edit the Name or Type fields of an RT I/O tag you have marked for OSx (PCS) translation unless you first unmark the tag.

Resistance Temperature Detector (continued)

Defining RT (Series 505)

When you select Resistance Temperature Detector (for the Series 505), this form appears:



Name: unique, symbolic name (12 characters maximum).

Type: 2-letter code for Resistance Temperature Detector
The module that uses this I/O type is 8RTD or 16RTD.

Address: you cannot edit this field; address is assigned in the Module Definition Table.

Description: optional (30 characters maximum).

RTD type: code that represents material of which RT is composed:

P	Platinum
N	Nickel
C	Copper

Filter: X indicates that you want to filter the input with a first-order exponential filter at a rate determined by a specified time constant.

Time Constant: real number that indicates rate (in seconds) at which you want to filter input; field appears only if you select **Filter**.

Engineering Units: code that represents the output format, as listed below; default is C.

C Degrees Centigrade: different ranges for the different types of RTD.

F Degrees Fahrenheit: different ranges for the different types of RTD.

	Platinum	**Copper**	**Nickel**
°C	-200.0 to 850.0	-200.0 to 260.0	-80.0 to 275.0
°F	-328.0 to 1562.0	-328.0 to 50.0	-112.0 to 527.0

O Ohms: Resistance measurements

A	ohms x 100	Range	1.0 to 2000.0
B	ohms x 10	Range	1.0 to 320.0

I Integer: scaled integer Range 0.0 to 32000.0

You cannot edit the Name or Type fields of an RT I/O tag you have marked for PCS translation unless you first unmark the tag.

5.13 Thermocouple

Using Thermocouple (TC)

Thermocouple (TC) is a signal from the field to the controller that represents a thermocouple voltage. This a symbolic name type that is used specifically for the inputs to a thermocouple module.

Availability

TCs are available for Series 500 and Series 505 I/O only.

A TC symbolic name can be used in an SFC step or Math statement or can be configured in a CFB. A TC symbolic name has the extensions listed in Table 5-12.

Table 5-12 TC Extensions

Extension	Definition
name [1]	scaled, processed (filtered) real value (read-only)
name.**FTAU** [1]	time constant that indicates rate for filter (read/write)
name.**RAW**	raw integer input from module (read-only)
name.**SRV** [1]	raw, scaled value before filter (read-only)
1 Not available for the 560/560T controllers.	

When you reference the symbolic name of a TC in your program, APT uses the scaled value for all but the 560/560T; the scaling is updated each time the associated SFPGM is executed. Any program attempts to modify this scaled value are overwritten by the system. The 560/560T controllers, which do not scale inputs, use the raw integer input, *name*.**RAW**, from a thermocouple module.

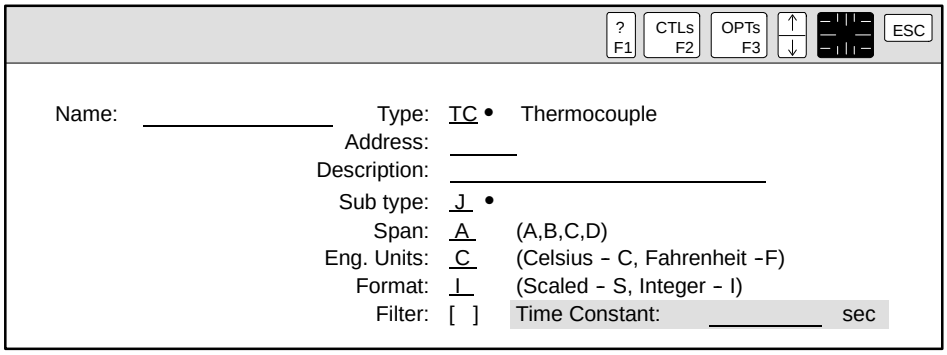
The TC behaves differently depending on whether it is attached to a Series 500 or a Series 505 Thermocouple Module. Until the I/O point is connected, the TC behaves as a Series 500 module point.

NOTE: Refer to the appropriate Series 500 or Series 505 thermocouple manual for configuration information.

Thermocouple (continued)

Defining TC (Series 500)

When you select Thermocouple (for the Series 500), this form appears:



Name: unique, symbolic name (12 characters maximum).

Type: 2-letter code for Thermocouple.
The module that uses this I/O type is TC.

Address: you cannot edit this field; address is assigned in the Module Definition Table.

Description: optional (30 characters maximum).

Sub Type: code that represents type of input to the module as listed below.

J Probe Type J
K Probe Type K
T Probe Type T
R Probe Type R
S Probe Type S
B Probe Type B
E Probe Type E
N Probe Type N

Span: code (A,B,C, or D) that represents a voltage span for millivolt inputs or a temperature span for thermocouple inputs; default is A. For more information about these spans, see the manual that comes with the thermocouple module.

Engineering Units: code that represents the temperature format as listed below.
C Degrees Centigrade
F Degrees Fahrenheit

Format: code that represents the output format as listed below.
S Scaled – Corresponds to the thermocouple "Degree" format
I Integer – Corresponds to the thermocouple "Integer" format

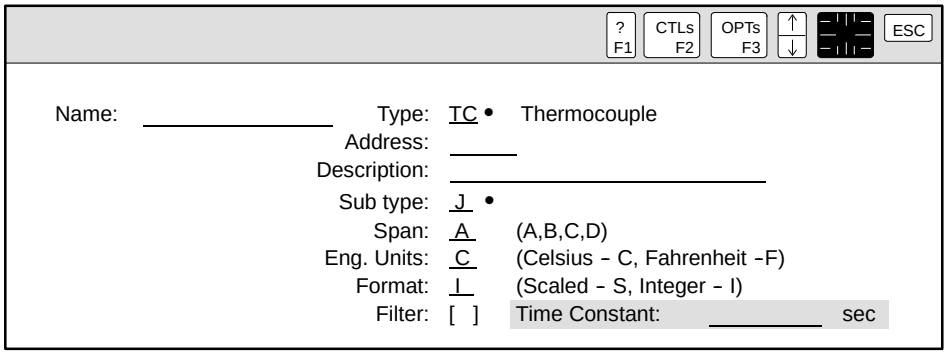
Filter: X indicates you want to filter the input with a first-order exponential filter at a rate determined by a specified time constant.

Time Constant: real number that indicates rate in seconds at which you want to filter input; field appears only if you select **Filter**.

You cannot edit the Name, Type, or Engineering Units fields of an TC I/O tag you have marked for OSx (PCS) translation unless you first unmark the tag.

Defining TC (Series 505)

When you select Thermocouple (for the Series 505), this form appears:



Name: unique, symbolic name (12 characters maximum).

Type: 2-letter code for Thermocouple.
The module that uses this I/O type is TC.

Address: you cannot edit this field; address is assigned in the Module Definition Table.

Description: optional (30 characters maximum).

Sub Type: code that represents type of input to the module as listed below.

J Probe Type J
K Probe Type K
T Probe Type T
R Probe Type R
S Probe Type S
E Probe Type E
N Probe Type N

Span: code that represents a temperature span for thermocouple inputs. The only span available for a Series 505 TC is A. For more information about this span, see the manual that comes with the thermocouple module.

Engineering Units: code that represents the temperature format as listed below.

C Degrees Centigrade
F Degrees Fahrenheit

Format: code that represents the output format as listed below.

S Scaled – Corresponds to the thermocouple "Engineering Units" format
I Integer – Corresponds to the thermocouple "Scaled Integer" format

Filter: X indicates you want to filter the input with a first-order exponential filter at a rate determined by a specified time constant.

Time Constant: real number that indicates rate in seconds at which you want to filter input; field appears only if you select **Filter**.

You cannot edit the Name, Type, or Engineering Units fields of an TC I/O tag you have marked for OSx (PCS) translation unless you first unmark the tag.

Chapter 6

Device Definitions

6.1 Basic Operation of Devices

Overview

An APT device is an object that uses a collection of I/O points to monitor and manipulate a field device, such as a valve, motor, cylinder, or press.

APT valves, motors, cylinders, and presses have two basic modes of operation.

- **Manual mode** indicates that a field device can be controlled by an operator.
- **Auto mode** indicates that the field device is under the control of the controller from a Sequential Function Chart, or from a Continuous Function Block.

The State Control Language and the Math Language include commands that you use to manipulate the field devices in auto mode. These commands have no effect in manual mode. APT also provides dot extensions that are appended to device names and used to monitor and control the field devices. In auto mode, commands take precedence over assignments made to extension variables.

The Device Definition Utility allows you to define and name APT devices in your process and to define the I/O points that control the corresponding field devices. Before you compile a program, the field I/O points associated with the APT device must also be defined in the Module Definition Table (for Series 505 controllers), or in the I/O Symbolic Name Table (for S5 controllers). See Figure 6-1 and Figure 6-2.

A Device Definition Table is available at both the Program Level and at the Unit Content Level. Devices defined at the Program Level can be referenced in any unit in that program. Devices defined at the Unit Level can be referenced only within that unit.

Each device type is described in detail in Chapter 7, which also shows the form that you use in configuring a device. You cannot edit the Name or Type fields in these forms if you have already marked the device for OSx (PCS) tag translation.

Availability

All APT devices (valves, motors, cylinders, presses, and stopwatch timers) are supported for both Series 505 and S5 controllers.

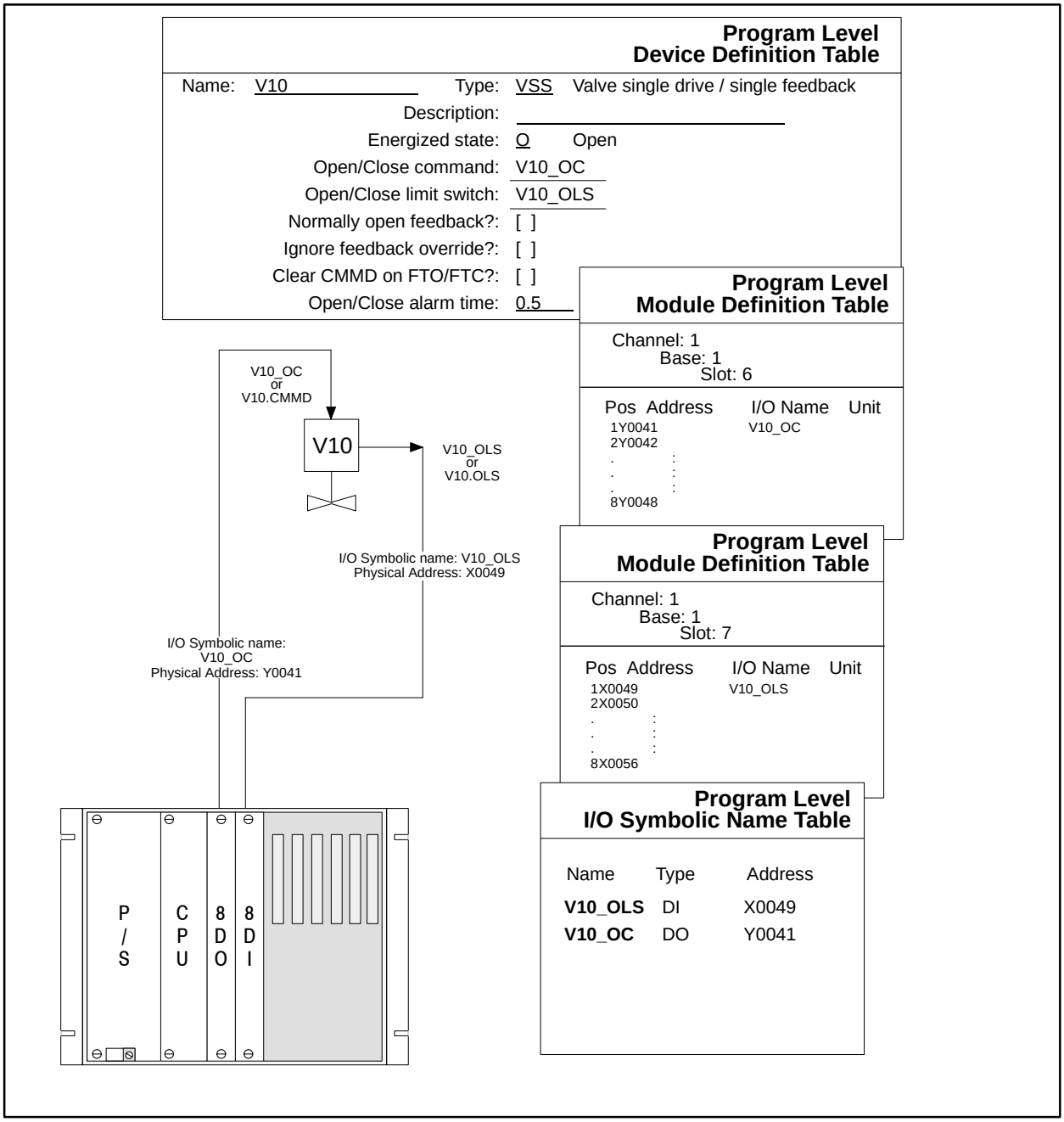
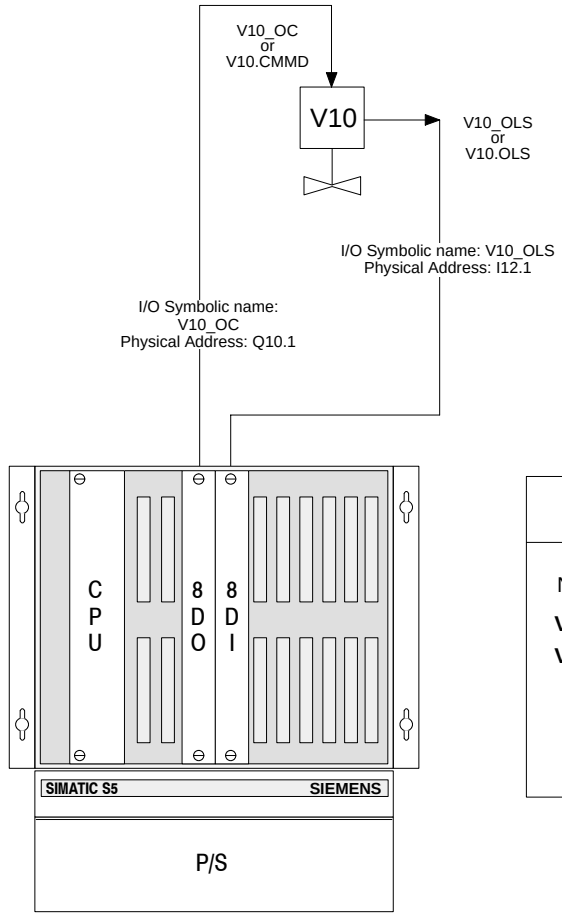


Figure 6-1 Devices, Modules, and I/O Symbolic Names For Series 505 Controllers

Program Level
Device Definition Table

Name:	V10	Type:	VSS Valve single drive / single feedback
		Description:	
		Energized state:	O Open
		Open/Close command:	V10_OC
		Open/Close limit switch:	V10_OLS
		Normally open feedback?:	[ ]
		Ignore feedback override?:	[ ]
		Clear CMMD on FTO/FTC?:	[ ]
		Open/Close alarm time:	0.5 seconds



Program Level
I/O Symbolic Name Table

Name	Type	Address
V10_OLS	DI	I12.1
V10_OC	DO	Q10.1

Figure 6-2 Devices and I/O Symbolic Names For S5 Controllers

Devices Types

Table 6-1 lists characteristics of the APT devices provided by the system. If none of these devices fit your application, you can control and monitor field devices with Math Language statements in a math or interlock CFB.

Table 6-1 APT Devices

Device Type	APT Code	Failed State
Valves		
Hand Valve/Dual-Feedback	VND	
Single-Drive/Null-Feedback (Energize Open)	VSN (Type O)	Closed
Single-Drive/Null-Feedback (Energize Closed)	VSN (Type C)	Open
Single-Drive/Single-Feedback (Energize Open)	VSS (Type O)	Closed
Single-Drive/Single-Feedback (Energize Closed)	VSS (Type C)	Open
Single-Drive/Dual-Feedback (Energize Open)	VSD (Type O)	Closed
Single-Drive/Dual-Feedback (Energize Closed)	VSD (Type C)	Open
Dual-Drive/Dual-Feedback	VDD	Closed
Motor-Drive/Dual-Feedback	VMD	Closed
Three-Position Low/High/Dual-Feedback	BV1	Closed
Three-Position Open/Position/Dual-Feedback	BV2	Closed
User-defined	VUD	User-defined
Motors		
Single-Drive/Null-Feedback	MSN	Stopped
Single-Drive/Single-Feedback	MSS	Stopped
Dual-Drive/Null-Feedback	MDN	Stopped
Dual-Drive/Single-Feedback	MDS	Stopped
Reversible Forward/Reverse/Dual-Feedback	RM1	Stopped
Reversible Drive/Direction/Dual-Feedback	RM2	Stopped
Two-Speed Low/High/Dual-Feedback	TS1	Stopped
Two-Speed Drive/Speed/Dual-Feedback	TS2	Stopped
User-defined	MUD	User-defined
Cylinder		
Single-Drive/Dual-Feedback (Energize Extend)	CSD (Type E)	Retracted
Single-Drive/Dual-Feedback (Energize Retract)	CSD (Type R)	Extended
User-defined	CUD	User-defined

Table 6-1 APT Devices (continued)

Device Type	APT Code	Failed State
Press		
Hand Press/Dual-Feedback	PND	
Single-Drive/Null-Feedback (Energize Raise)	PSN (Type R)	Down
Single-Drive/Null-Feedback (Energize Lower)	PSN (Type L)	Up
Single-Drive/Single-Feedback (Energize Raise)	PSS (Type R)	Down
Single-Drive/Single-Feedback (Energize Lower)	PSS (Type L)	Up
Single-Drive/Dual-Feedback (Energize Raise)	PSD (Type R)	Down
Single-Drive/Dual-Feedback (Energize Lower)	PSD (Type L)	Up
Dual-Drive/Dual-Feedback	PDD	Down
Motor-Drive/Dual-Feedback	PMD	Down
Three-Position Low/High/Dual-Feedback	PS1	Down
Three-Position Raise/Position/Dual-Feedback	PS2	Down
User-defined	PUD	User-defined
Stopwatch		
Timer	TMR	

Commands

All devices have associated commands in the State Control Language that you use to control the device when it is in auto mode. Commands have no effect in manual mode. You can use commands from the parallel section of an SFC step or a math block that generates RLL code. Table 6-2 lists the device commands with the corresponding APT device types.

Table 6-2 APT Device Commands

Command	Action	Devices						
Valve Commands								
LOCK	Place in auto mode	VSN	VSS	VSD	VDD	VMD	BV1/2	VUD
UNLOCK	Place in manual mode	VSN	VSS	VSD	VDD	VMD	BV1/2	VUD
OPEN	Open valve	VSN	VSS	VSD	VDD	VMD		VUD
OPENA	Open option A							VUD
OPENB	Open option B							VUD
OPENH	Open high, option H						BV1/2	VUD
OPENL	Open low, option L						BV1/2	VUD
CLOSE	Close valve	VSN	VSS	VSD	VDD	VMD	BV1/2	VUD
RESET	Clear feedback overrides, trigger alarm timer to reset		VSS	VSD	VDD	VMD	BV1/2	VUD
Motor Commands								
LOCK	Place in auto mode	MSN	MSS	MDN	MDS	RM1/2	TS1/2	MUD
UNLOCK	Place in manual mode	MSN	MSS	MDN	MDS	RM1/2	TS1/2	MUD
START	Start motor	MSN	MSS	MDN	MDS			MUD
STARTF	Start forward					RM1/2		
STARTR	Start reverse					RM1/2		
STARTH	Start high						TS1/2	
STARTL	Start low						TS1/2	
STOP	Stop motor	MSN	MSS	MDN	MDS	RM1/2	TS1/2	MUD
RESET	Clear feedback overrides, trigger alarm timer to reset		MSS		MDS	RM1/2	TS1/2	MUD
Cylinder Commands								
LOCK	Place in auto mode	CSD	CUD					
UNLOCK	Place in manual mode	CSD	CUD					
EXTEND	Extend piston	CSD	CUD					
RETRACT	Retract piston	CSD	CUD					
RESET	Clear feedback overrides, trigger alarm timer to reset	CSD	CUD					

Table 6-2 APT Device Commands (continued)

Command	Action	Devices						
Press Commands								
LOCK	Place in auto mode	PSN	PSS	PSD	PDD	PMD	PS1/2	PUD
UNLOCK	Place in manual mode	PSN	PSS	PSD	PDD	PMD	PS1/2	PUD
RAISE	Raise press	PSN	PSS	PSD	PDD	PMD		PUD
RAISEA	Raise option A							PUD
RAISEB	Raise option B							PUD
RAISEH	Raise high, option H						PS1/2	PUD
RAISEL	Raise low, option L						PS1/2	PUD
LOWER	Lower press	PSN	PSS	PSD	PDD	PMD	PS1/2	PUD
RESET	Clear feedback overrides, trigger alarm timer to reset		PSS	PSD	PDD	PMD	PS1/2	PUD
Stopwatch Commands								
START	Start stopwatch	TMR						
STOP	Stop stopwatch	TMR						
HOLD	Hold current count	TMR						
CONT	Continue count	TMR						
CONTINUE	Continue count	TMR						
RESET	Clear current count	TMR						

Extensions

All APT devices have associated extension variables that can be used to monitor and control the operation of a field device. APT creates these variables by appending a dot extension to the APT device name. These extensions create two types of bit variables.

- Status bits, which are read-only boolean values, allow you to monitor the condition of the device. These include output bits, position bits, feedback inputs, and fail bits.
- Command bits, which are read-write boolean values, change the condition of the field device. When you use the commands associated with the device, the commands manipulate the command-bit extensions.

All of the following methods can be used to change the state of a device; however, since commands overwrite APT flags and boolean extensions, do not try to control a device using a mixed-method approach.

Method	Example
Use a command in an SFC	**Lock** *valve_1*;
Use a command in MATH (RLL only)	**Lock** (*valve_1*);
Use an APT flag	**On** (*valve_1*.**RTL**);
Use a boolean extension	*valve_1*.**LOCKD := true;**

Use the commands, rather than the command-bit extensions, to control the mode of the device. Use the APT flags and any of the boolean extensions to monitor the status of the device. If you use an APT flag extension, you must use the LATCH, ON, and CLEAR math procedures, described in the chapter called “Math Functions and Procedures” in the *SIMATIC APT Programming Reference (Graphics/Math) Manual*.

The stopwatch device has other types of extensions that are explained in the “Stopwatch” section of Chapter 7 in this manual.

When you want to manipulate a field device, you can use commands that allow you to execute an action on an object from the parallel section of an SFC step or a math block that generates RLL code for Series 505 controllers, or STL code for S5 controllers. You can also use math procedures to control the APT flag extensions of an object. Refer to Figure 6-3. In this example, v10 opens when Step 1 becomes active. When the status bit *v10*.**OPND** becomes true, indicating that the valve is actually open, the program moves to Step 2.

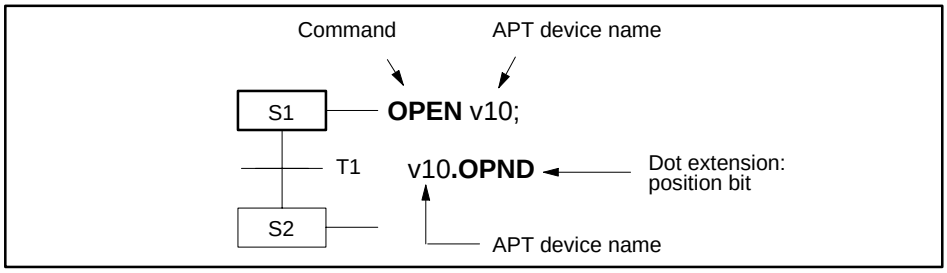


Figure 6-3 Using Device Commands and Extensions

Table 6-3 lists the valve extensions, their basic function, and the corresponding valve types. Table 6-4 lists the motor extensions; Table 6-5 lists the cylinder extensions; Table 6-6 lists the press extensions; and Table 6-7 lists the timer extensions.

⚠ WARNING

APT does not prevent you from writing to read-only variables. If you write to these variables, the controller overwrites them on the next scan with the system-defined values.

Writing to read-only variables may cause your application to execute in an unpredictable and unsafe manner that could result in death or serious injury and/or property damage.

Understand what you are doing when you write to read-only variables.

Table 6-3 Valve Extensions

Extension	Definition	Valve Types							
Output Bits (Read-only)									
.CMMD	open/close command		VSN	VSS	VSD	VDD	VMD		VUD
.OPENC	open command					VDD	VMD		VUD
.SHIGH	open high							BV1	
.SLOW	open low							BV1	
.DRV	open command							BV2	
.POS	open position							BV2	
.CLSC	close command					VDD	VMD		VUD
Position Bits (Read-only)									
.OPND	opened	VND	VSN	VSS	VSD	VDD	VMD	BV1/2	VUD
.CLSD	closed	VND	VSN	VSS	VSD	VDD	VMD	BV1/2	VUD
.OPNDA	opened option A								VUD
.OPNDB	opened option B								VUD
.OPNDH	opened high, option H							BV1/2	VUD
.OPNDL	opened low, option L							BV1/2	VUD
.TRVL	traveling		VSN	VSS	VSD	VDD	VMD	BV1/2	VUD
Feedback Inputs (Read-only) [1]									
.OLS	open feedback	VND		VSS [2]	VSD	VDD	VMD		VUD
.CLS	closed feedback	VND		VSS [3]	VSD	VDD	VMD		VUD
.HIO	open high feedback							BV1/2	
.LIO	open low feedback							BV1/2	
Fail Bits (Read-only)									
.FTO	fail to open			VSS	VSD	VDD	VMD	BV1/2	VUD
.FTC	fail to close			VSS	VSD	VDD	VMD	BV1/2	VUD
.FTOH	fail to open high							BV1/2	
.FTOL	fail to open low							BV1/2	
.FAILD	failed	VND			VSD	VDD	VMD	BV1/2	VUD
Operation Bits (Read/Write)									
.DSBLD	forced to manual mode		VSN	VSS	VSD	VDD	VMD	BV1/2	VUD
.LOCKD	auto mode		VSN	VSS	VSD	VDD	VMD	BV1/2	VUD
.NRDY	not ready		VSN	VSS	VSD	VDD	VMD	BV1/2	VUD
.MOPEN	manual open	VND	VSN	VSS	VSD	VDD	VMD	BV1/2	VUD
.MHIGH	manual high							BV1/2	
.RTS	stop the valve motor						VMD		

1 If the feedback inputs are not external discrete inputs (type DI), these bits are read/write.
2 Energize-open valve only
3 Energize-closed valve only

Table 6-3 Valve Extensions (continued)

Extension	Definition	Valve Types						
Override Bits (Read/Write)								
.OVRD	override feedback		VSS					
.OVRDO	override open feedback			VSD	VDD	VMD		VUD
.OVRDC	override closed feedback			VSD	VDD	VMD		VUD
.OVRDH	override high, option H feedback						BV1/2	
.OVRDL	override low, option L feedback						BV1/2	
.CRESET	close reset							VUD
.ORESET	open reset							VUD
Flags/Commands (Read/Write)								
.RTL/LOCK	place in auto mode	VSN	VSS	VSD	VDD	VMD	BV1/2	VUD
.RTU/ UNLOCK	place in manual mode	VSN	VSS	VSD	VDD	VMD	BV1/2	VUD
.RTO/OPEN	open valve	VSN	VSS	VSD	VDD	VMD		VUD
.RTOA/ OPENA	open option A							VUD
.RTOB/ OPENB	open option B							VUD
.RHIGH/ OPENH	open high						BV1/2	VUD
.RLOW/ OPENL	open low						BV1/2	VUD
.RTC/CLOSE	close valve	VSN	VSS	VSD	VDD	VMD	BV1/2	VUD
.RESET/ RESET	clear feedback overrides, trigger alarm timer to reset		VSS	VSD	VDD	VMD	BV1/2	VUD

Table 6-3 Valve Extensions (continued)

Extension	Definition	Valve Types						
Timer Status (Read-only)								
.CLSTO	close timeout				VDD	VMD	BV1/2	VUD
.OPNTO	open timeout				VDD	VMD	BV1/2	VUD
Timer Integer Current (Read-only)								
.CTCC	close timer counter current (1 count equals 1/10 second)	VSN	VSS	VSD	VDD	VMD	BV1/2	VUD
.OTCC	open timer counter current (1 count equals 1/10 second)	VSN	VSS	VSD	VDD	VMD	BV1/2	VUD
Timer Integer Preset (Read/Write)								
.CTCP	close timer counter preset (1 count equals 1/10 second)	VSN	VSS	VSD	VDD	VMD	BV1/2	VUD
.OTCP	open timer counter preset (1 count equals 1/10 second)	VSN	VSS	VSD	VDD	VMD	BV1/2	VUD

Table 6-4 Motor Extensions

Extension	Definition	Motor Types						
Output Bits (Read-only)								
.CMMD	start/stop output	MSN	MSS	MDN	MDS			MUD
.STRTC	start output			MDN	MDS			MUD
.STOPC	stop output			MDN	MDS			MUD
.SFWRD	start forward output					RM1		
.SREV	start reverse output					RM1		
.SHIGH	start high output						TS1	
.SLOW	start low output						TS1	
.DRV	start output					RM2	TS2	
.DIR	direction output					RM2		
.SPEED	speed output						TS2	
Position Bits (Read-only)								
.RUNNG	running	MSN	MSS	MDN	MDS	RM1/2	TS1/2	MUD
.RUNF	running forward					RM1/2		
.RUNR	running reverse					RM1/2		
.RUNH	running high						TS1/2	
.RUNL	running low						TS1/2	
.STPPD	stopped	MSN	MSS	MDN	MDS	RM1/2	TS1/2	MUD
.TRVL	traveling	MSN	MSS	MDN	MDS	RM1/2	TS1/2	MUD
Feedback Inputs (Read-only) [1]								
.RUNIO	run feedback		MSS		MDS			MUD
.FIO	forward feedback					RM1/2		
.RIO	reverse feedback					RM1/2		
.HIO	high feedback						TS1/2	
.LIO	low feedback						TS1/2	
Fail Bits (Read-only)								
.FTR	fail to run		MSS		MDS	RM1/2	TS1/2	MUD
.FTS	fail to stop		MSS		MDS	RM1/2	TS1/2	MUD
.FTRF	fail to run forward					RM1/2		
.FTRR	fail to run reverse					RM1/2		
.FTRH	fail to run high						TS1/2	
.FTRL	fail to run low						TS1/2	
.FAILD	failed					RM1/2	TS1/2	
Operation Bits (Read/Write)								
.DSBLD	forced to manual mode	MSN	MSS	MDN	MDS	RM1/2	TS1/2	MUD
.LOCKD	auto mode	MSN	MSS	MDN	MDS	RM1/2	TS1/2	MUD
.NRDY	not ready	MSN	MSS	MDN	MDS	RM1/2	TS1/2	MUD
.MSTRT	manual start	MSN	MSS	MDN	MDS	RM1/2	TS1/2	MUD
.MREV	manual reverse					RM1/2		
.MHIGH	manual high						TS1/2	

1 If the feedback inputs are not type DI, these bits are read/write.

Table 6-4 Motor Extensions (continued)

Extension	Definition	Motor Types						
Override Bits (Read/Write)								
.OVRD	override feedback		MSS		MDS			MUD
.OVRDF	override forward feedback					RM1/2		
.OVRDR	override reverse feedback					RM1/2		
.OVRDH	override high feedback						TS1/2	
.OVRDL	override low feedback						TS1/2	
.RRESET	run reset							MUD
.SRESET	stop reset							MUD
Flags/Commands (Read/Write)								
.RTL/LOCK	place in auto mode	MSN	MSS	MDN	MDS	RM1/2	TS1/2	MUD
.RTU/ UNLOCK	place in manual mode	MSN	MSS	MDN	MDS	RM1/2	TS1/2	MUD
.RTR/START	start motor	MSN	MSS	MDN	MDS			MUD
.RFWRD/ STARTF	start forward					RM1/2		
.RREV/ STARTR	start reverse					RM1/2		
.RHIGH/ STARTH	start high						TS1/2	
.RLOW/ STARTL	start low						TS1/2	
.RTS/STOP	stop motor	MSN	MSS	MDN	MDS			MUD
.RSTOP/STOP	stop motor					RM1/2	TS1/2	
.RESET/ RESET	clear feedback overrides, trigger alarm timer to reset		MSS		MDS	RM1/2	TS1/2	MUD
Timer Status (Read-only)								
.STPTO	stop timeout				MDS	RM1/2	TS1/2	MUD
.STRTO	start timeout				MDS	RM1/2	TS1/2	MUD
Timer Integer Current (Read-only)								
.RTCC	run timer counter current (1 count equals 1/10 second)	MSN	MSS	MDN	MDS	RM1/2	TS1/2	MUD
.STCC	stop timer counter current (1 count equals 1/10 second)	MSN	MSS	MDN	MDS	RM1/2	TS1/2	MUD
Timer Integer Preset (Read/Write)								
.RTCP	run timer counter preset (1 count equals 1/10 second)	MSN	MSS	MDN	MDS	RM1/2	TS1/2	MUD
.STCP	stop timer counter preset (1 count equals 1/10 second)	MSN	MSS	MDN	MDS	RM1/2	TS1/2	MUD

Table 6-5 Cylinder Extensions

Extension	Definition	Cylinder Types
Output Bits (Read-only)		
.CMMD	extend/retract command	CSD CUD
.EXTC	extend command	CUD
.RETC	retract command	CUD
Position Bits (Read-only)		
.EXTENDED	extended	CSD CUD
.RETRACTED	retracted	CSD CUD
.TRVL	traveling	CSD CUD
Feedback Inputs (Read-only) [1]		
.ELS	extended feedback	CSD CUD
.RLS	retracted feedback	CSD CUD
Fail Bits (Read-only)		
.FTE	fail to extend	CSD CUD
.FTR	fail to retract	CSD CUD
.FAILD	failed	CSD CUD
Operation Bits (Read/Write)		
.DSBLD	forced to manual mode	CSD CUD
.NRDY	not ready	CSD CUD
.LOCKD	auto mode	CSD CUD
.MEXTEND	manual extend	CSD CUD
Override Bits (Read/Write)		
.OVRDE	override extend feedback	CSD CUD
.OVRDR	override retract feedback	CSD CUD
.ERESET	extend reset	CUD
.RRESET	retract reset	CUD
Flags/Commands (Read/Write)		
.RTL/LOCK	place in auto mode	CSD CUD
.RTU/ UNLOCK	place in manual mode	CSD CUD
.RTE/EXTEND	extend cylinder	CSD CUD
.RTR/ RETRACT	retract cylinder	CSD CUD
.RESET/ RESET	clear feedback overrides, trigger alarm timer to reset	CSD CUD

1 If the feedback inputs are not external discrete inputs (type DI), these bits are read/write.

Table 6-5 Cylinder Extensions (continued)

Extension	Definition	Cylinder Types
Timer Status (Read-only)		
.EXTTO	extend timeout	CUD
.RETTO	retract timeout	CUD
Timer Integer Current (Read-only)		
.ETCC	extend timer counter current (1 count equals 1/10 second)	CSD CUD
.RTCC	retract timer counter current (1 count equals 1/10 second)	CSD CUD
Timer Integer Preset (Read/Write)		
.ETCP	extend timer counter preset (1 count equals 1/10 second)	CSD CUD
.RTCP	retract timer counter preset (1 count equals 1/10 second)	CSD CUD

1 If the feedback inputs are not external discrete inputs (type DI), these bits are read/write.

Table 6-6 Press Extensions

Extension	Definition	Press Types							
Output Bits (Read-only)									
.CMMD	raise/lower command		PSN	PSS	PSD	PDD	PMD		PUD
.UPC	raise command					PDD	PMD		PUD
.SHIGH	raise high							PS1	
.SLOW	raise low							PS1	
.DRV	raise command							PS2	
.POS	raise position							PS2	
.DOWNC	lower command					PDD	PMD		PUD
Position Bits (Read-only)									
.UP	up	PND	PSN	PSS	PSD	PDD	PMD	PS1/2	PUD
.DOWN	down	PND	PSN	PSS	PSD	PDD	PMD	PS1/2	PUD
.UPA	up option A								PUD
.UPB	up option B								PUD
.UPH	up high, option H							PS1/2	PUD
.UPL	up low, option L							PS1/2	PUD
.TRVL	traveling		PSN	PSS	PSD	PDD	PMD	PS1/2	PUD
Feedback Inputs (Read-only) [1]									
.ULS	up feedback	PND		PSS[2]	PSD	PDD	PMD		PUD
.DLS	down feedback	PND		PSS[3]	PSD	PDD	PMD		PUD
.HIO	open high feedback							PS1/2	
.LIO	open low feedback							PS1/2	
Fail Bits (Read-only)									
.FTR	fail to raise			PSS	PSD	PDD	PMD	PS1/2	PUD
.FTL	fail to lower			PSS	PSD	PDD	PMD	PS1/2	PUD
.FTRH	fail to raise high							PS1/2	
.FTRL	fail to raise low							PS 1/2	
.FAILD	failed	PND			PSD	PDD	PMD	PS1/2	PUD
Operation Bits (Read/Write)									
.DSBLD	forced to manual mode		PSN	PSS	PSD	PDD	PMD	PS1/2	PUD
.LOCKD	auto mode		PSN	PSS	PSD	PDD	PMD	PS1/2	PUD
.NRDY	not ready		PSN	PSS	PSD	PDD	PMD	PS1/2	PUD
.MRAISE	manual raise	PND	PSN	PSS	PSD	PDD	PMD	PS1/2	PUD
.MHIGH	manual high							PS1/2	
.RTS	stop the press motor						PMD		

1 If the feedback inputs are not external discrete inputs (type DI), these bits are read/write.
2 Energize-raise press only.
3 Energize-lower press only.

Table 6-6 Press Extensions (continued)

Override Bits (Read/Write)								
.OVRD	override feedback		PSS					
.OVRDD	override down feedback			PSD	PDD	PMD		PUD
.OVRDU	override up feedback			PSD	PDD	PMD		PUD
.OVRDH	override high option H feedback						PS1/2	
.OVRDL	override low, option L feedback						PS1/2	
.URESET	lower reset							PUD
.DRESET	raise reset							PUD
Flags/Command (Read/Write)								
.RTL / LOCK	place in auto mode	PSN	PSS	PSD	PDD	PMD	PS1/2	PUD
.RTU / UNLOCK	place in manual mode	PSN	PSS	PSD	PDD	PMD	PS1/2	PUD
.RTO/RAISE	raise press	PSN	PSS	PSD	PDD	PMD		PUD
.RTOA/ RAISEA	raise option A							PUD
.RTOB/ RAISEB	raise option B							PUD
.RHIGH/ RAISEH	raise high						PS1/2	PUD
.RLOW/ RAISEL	raise low						PS1/2	PUD
.RTC/LOWER	lower press	PSN	PSS	PSD	PDD	PMD	PS1/2	PUD
.RESET/ RESET	clear feedback overrides, trigger alarm timer to reset		PSS	PSD	PDD	PMD	PS1/2	PUD
Timer Status (Read-only)								
.UPTO	down timeout				PDD	PMD	PS1/2	PUD
.DOWNTO	up timeout				PDD	PMD	PS1/2	PUD
Timer Integer Current (Read-only)								
.DTCC	down timer counter current (1 count equals 1/10 second)	PSN	PSS	PSD	PDD	PMD	PS1/2	PUD
.UTCC	up timer counter current (1 count equals 1/10 second)	PSN	PSS	PSD	PDD	PMD	PS1/2	PUD
Timer Integer Preset (Read/Write)								
.DTCP	down timer counter preset (1 count equals 1/10 second)	PSN	PSS	PSD	PDD	PMD	PS1/2	PUD
.UTCP	up timer counter preset (1 count equals 1/10 second)	PSN	PSS	PSD	PDD	PMD	PS1/2	PUD

Table 6-7 Timer Extensions

Extension	Definition	Extension	Definition
Read-only			
.RUNNG	running	.GT	true when (**.CUR > .PSET**)
.HOLD	holding	.EQ	true when (**.CUR = .PSET**)
.MAXC	maximum value reached	.TOUT	time out (**.CUR ≥ .PSET**)
Read/Write			
.RATE	tick rate of timer	.CUR	current timer count
.PSET	preset		

6.2 Device Modes

Manual Mode

A field device is in manual mode whenever the **.LOCKD** bit for that device is false. When the program is first downloaded to the controller, all devices are in manual mode and set to the failed (de-energized) state, which is listed in Table 6-1.

After the program is in the controller, the following conditions place a field device in manual mode.

- The **.DSBLD** bit for that device is set to true.
- An UNLOCK command is issued for that device.
- The unit containing the device is disabled (unit_name**.ENABL** is false; this applies only to APT devices local to a unit, not to APT devices defined at the Program Level.)

A device remains in manual mode until a LOCK command places the device in auto mode, or the device is interlocked (**.NRDY** is true). However, any one of the above conditions takes precedence over a LOCK command and prevents the object from leaving manual mode.

In manual mode, an operator can control a device by toggling the operation bits (**.MOPEN** bit for valves, the **.MSTRT** bits for motors, the **.MEXTEND** bit for cylinders, and **.MRAISE** for presses). These bits can also be manipulated from the following types of CFBs: math, interlock, PID, or analog alarm.

- To open (start, extend, raise) the device, set the **.MOPEN** (**.MSTRT, .MEXTEND,** or **.MRAISE**) bit to true.
- To close (stop, retract, lower) the device, set the **.MOPEN** (**.MSTRT, .MEXTEND,** or **.MRAISE**) bit to false.

The OPEN/CLOSE (START/STOP, EXTEND/RETRACT, RAISE/LOWER) commands have no effect in manual mode.

Auto Mode

A field device is in auto, or locked, mode whenever the **.LOCKD** bit is true. A LOCK command places the device in auto mode if no manual-mode conditions exist (page 6-23) and the device is interlocked (**.NRDY** is true). After the device enters auto mode, it remains that way until any one of the conditions that places the device in manual mode becomes true.

In auto mode, the state of a valve is controlled by the OPEN/CLOSE commands. A motor is controlled by the START/STOP commands. A cylinder is controlled by the EXTEND/RETRACT commands. A press is controlled by the RAISE/LOWER commands.

Because the OPEN/CLOSE (START/STOP, EXTEND/RETRACT, and RAISE/LOWER) commands are latched, it is not necessary to issue a command continuously to keep a device in the desired state. If you issue OPEN and CLOSE (START and STOP, EXTEND and RETRACT, RAISE and LOWER) commands simultaneously, the command that takes the device to its failed state takes precedence over the command that moves it away from the failed state.

While a field device is in auto mode, the **.MOPEN** (**.MSTRT**, **.MEXTEND**, and **.MRAISE**) bit follows the desired state. If the desired state is open (running, extended, raised), **.MOPEN** (**.MSTRT**, **.MEXTEND**, **.MRAISE**) is forced to true; otherwise, **.MOPEN** (**.MSTRT**, **.MEXTEND**, **.MRAISE**) is forced to false. While in auto mode, the **.MOPEN** (**.MSTRT**, **.MEXTEND**, **.MRAISE**) bit cannot be changed manually.

Figure 6-4 shows the commands and dot extensions that determine the mode of a device.

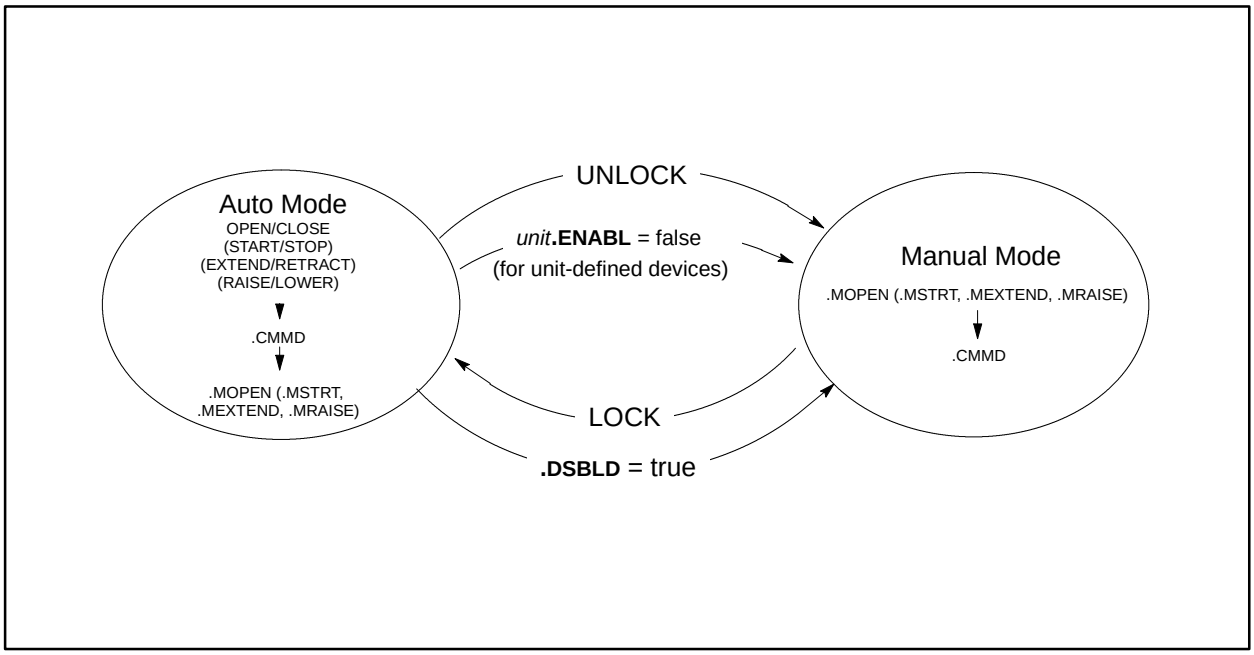


Figure 6-4 Manual and Auto Modes

Changing Modes

When a field device is switched from manual mode to auto mode, the device maintains the state that existed before the mode change. For example, if a valve is open, it remains open to provide a bumpless transfer. If the valve is in a traveling state, it remains in that state and continues toward the state that was requested before the mode change.

If a LOCK command is issued at the same time that the manual-control operation bit (**.MOPEN**/**.MSTRT**/**.MEXTEND/.MRAISE**) is toggled, the device enters auto mode before the change is noticed. As a result, the device does not change state; and the desired state is the same as it was before you requested the mode change.

The way that the device behaves when switching from auto mode to manual mode depends on how the **.MOPEN**/**.MSTRT**/**.MEXTEND/.MRAISE** bit is controlled.

- If the bit is controlled from an operator station that does not continuously write to the bit, switching from locked to manual mode is identical to switching from manual to locked mode. (The device maintains the state that existed before the mode change).
- If the bit is controlled directly by a switch or push button, the device may begin traveling to a new state if the state indicated by the **.MOPEN** (**.MSTRT**, **.MEXTEND/.MRAISE**) bit is not the same as the current state of the device.

If an UNLOCK and an OPEN (START/EXTEND/RAISE) or a CLOSE (STOP/RETRACT/LOWER) command are issued simultaneously, the device enters manual mode before the open or close request is processed; as a result, the device does not change state.

Changing States

There are two cases when a device can change state without operator action.

- If the **.NRDY** bit becomes true, the device is forced to the failed state and the **.MOPEN** (**.MSTRT**, **.MEXTEND, .MRAISE**) bit shows the corresponding failed-state value.

  Commands or assignment statements that attempt to change the **.MOPEN** (**.MSTRT**, **.MEXTEND, .MRAISE**) bit and move the device away from its failed state are overridden while the **.NRDY** bit is true.

  When **.NRDY** returns to false, the device remains in its failed state until it receives a request to change.

- The second special case is when the power to the controller fails. In this case, the device behaves exactly as if the **.NRDY** bit were true during the power loss and returns to false when the power returns.

6.3 Device Feedback

APT devices have feedback bits that indicate that the device has reached the required state or position. A device has either no feedback bits (null feedback, page 6-29), one feedback bit (single feedback, page 6-30), or two feedback bits (dual feedback, page 6-32).

Override Bits

Devices with feedback have override bits that allow you to ignore the feedback status.

- Single-feedback devices have one override bit (**.OVRD**).
- Dual-feedback valves have two override bits (**.OVRDO** and **.OVRDC**).
- Batch values have two override bits (**.OVRDH** and **.OVRDL**).
- Reversible and two-speed motors each have two override bits (**.OVRDF**/**.OVRDR** and **.OVRDL**/**.OVRDH**).
- Dual feedback cylinders have two override bits (**.OVRDE** and **.OVRDR**).
- Dual feedback presses have two override bits (**.OVRDU** and **.OVRDD**).
- Three-position presses have two overide bits (**.OVRDH** and **.OVRDL**).

If an override bit is true, the corresponding feedback bit is ignored. The **.TRVL** bit operates like the null feedback device when the override bit is true.

Reset Command

The RESET command allows you to set all override bits to false. When the SFC step or math section that contains this command is active, attempts to change the override bits to true are ignored.

The RESET command also triggers the appropriate alarm timer to reset. The .**TRVL** bit becomes true and the alarm time begins timing down. From this point, the behavior of the device depends on the state of the override bits.

Null Feedback

A null feedback device provides no indication that the device has actually reached the desired state or position, but assumes the desired state has been reached after the specified alarm time expires.

The desired position is determined by the **.MOPEN** (**.MSTRT, MEXTEND, .MRAISE**) bit in manual mode or by the OPEN/CLOSE (START/STOP, EXTEND/RETRACT, RAISE/LOWER) commands in auto mode.

Whenever the desired state of a null feedback device changes or the controller returns from a power loss, the **.TRVL** bit for that device becomes true and the alarm time begins timing down.

When the alarm time elapses, the **.TRVL** bit becomes false and the bit that indicates the desired position becomes true: either **.OPND** or **.CLSD** (correspondingly, either **.RUNNG** or **STPPD**, **.EXTENDED** or **.RETRACTED, .UP** or **.DOWN**). That bit then remains true until either the desired state changes or the controller loses power.

The logic for a null feedback device is shown in Appendix A.

Single Feedback

A single-feedback device has one input that indicates whether or not the device has actually reached the desired state, or position. Single-feedback devices use the following feedback bits:

Device	Feedback Bits
Energize-open Valve (O)	**.OLS**
Energize-close Valve (C)	**.CLS**
Motor	**.RUNIO**
Energize-raise Press (R)	**.ULS**
Energize-lower Press (L)	**.DLS**

A feedback is expected to be true when the device is in its energized state; otherwise, it is false.

Whenever the desired state of a single-feedback device changes or the controller returns from a power loss, the **.TRVL** bit becomes true and the alarm time begins timing down. From this point, the behavior of the device depends on the state of the override (**.OVRD**) bit.

If **.OVRD** is false, the following actions occur.

- If the feedback bit indicates that the device is in the desired state, the appropriate position or state bit listed below is true:

Device	Position or State Bit	
Valve	**.OPND**	**.CLSD**
Motor	**.RUNNG**	**.STPPD**
Press	**.UP**	**.DOWN**

- If the alarm time is expired and the feedback bit indicates that the device is not in the desired state, the appropriate fail bit listed below is true:

Device	Fail Bit	
Valve	**.FTO**	**.FTC**
Motor	**.FTR**	**.FTS**
Press	**.FTR**	**.FTL**

  If the alarm time is not expired and the feedback bit is false, the **.TRVL** bit is true.

If **.OVRD** is true, the feedback bit is ignored, and the device operates like a null-feedback device.

When the step or math section that contains the RESET command is active, the **.OVRD** bit is forced to false; attempts to change this bit to true are overwritten.

When the step or math section that contains the RESET command becomes inactive, the **.OVRD** bit remains false until changed by an operator station or by the program.

The logic for a single-feedback device is shown in Appendix A.

Dual Feedback

A dual-feedback device has two inputs that indicate whether or not the device has actually reached the desired state or position. Dual-feedback devices use the following feedback bits:

Device	Feedback Bits	
Valve	.OLS	.CLS
Batch Valve	.HIO	.LIO
Reversible Motor	.FIO	.RIO
Two-Speed Motor	.LIO	.HIO
Cylinder	.ELS	.RLS
Press	.ULS	.DLS
Three-Position Press	.HIO	.LIO

Whenever the desired state of a device changes or the controller returns from a power loss, the **.TRVL** bit becomes true, and the alarm time begins timing down. From this point, the behavior of the device depends on the state of the following override bits:

Device	Override Bits	
Valve	.OVRDO	.OVRDC
Batch Valve	.OVRDH	.OVRDL
Reversible Motor	.OVRDF	.OVRDR
Two-Speed Motor	.OVRDL	.OVRDH
Cylinder	.OVRDE	.OVRDR
Press	.OVRDU	.OVRDD
Three-Position Press	.OVRDH	.OVRDL

Both override bits false. If both override bits are false, the following actions occur.

- If the appropriate feedback input for the required position is true, the corresponding position or state bit listed below is true:

Device	Position or State Bit	
Valve	.OPND	.CLSD
Batch Valve	.OPNDH	.OPNDL
Reversible Motor	.RUNF	.RUNR
Two-Speed Motor	.RUNL	.RUNH
Cylinder	.EXTENDED	.RETRACTED
Press	.UP	.DOWN
Three-Position Press	.UPH	.UPL

- If the alarm time is expired and the appropriate feedback input for the desired position is false, the corresponding fail bit listed below is true:

Device	Fail Bit	
Valve	.FTO	.FTC
Batch Valve	.FTOH	.FTOL
Reversible Motor	.FTRF	.FTRR
Two-Speed Motor	.FTRL	.FTRH
Cylinder	.FTE	.FTR
Press	.FTR	.FTL
Three-Position Press	.FTRH	.FTRL

- If the alarm time is expired and both feedback bits are true, the **.FAILD** bit is true.

- If the alarm time is not expired and neither feedback input is in the desired state, the **.TRVL** bit is true.

Either override bit true. If either override bit is true, the state that is overridden acts like a null-feedback device.

Both override bits true. If both override bits are true, the feedback bits are ignored, and the device operates like a null-feedback device.

In switching the override bits between states, the position and fail bits assume whatever status is correct for the current state of the command and feedback bits.

When the SFC step or math section that contains the RESET command is active, the override bits are forced to false; attempts to change these bits to true are overwritten. When the SFC step or math section that contains the RESET command becomes inactive, the override bits remain false until changed by an operator station or by the program.

The logic for a dual-feedback device is shown in Appendix A.

6.4 Device Power Fail Recovery

Power-Fail Recovery Logic for Series 505

For Series 505 controllers, APT devices are based on non-retentive control relays. However, the controller saves only the state of the retentive control relays during a power failure. For this reason, APT includes power-fail recovery logic in the code for each device unless you select the Uninterruptible Power Supply (**UPS**) option in the Compiler Control file. The power-fail recovery logic is represented by the **.PFAIL** control relays shown in the figures of Appendix A, and provides a means of saving the status of a device during a power failure.

APT executes the power-fail recovery logic for all APT devices during every controller scan. For each device, the states of the device's status control relays are copied into a word in the Series 505 controller V-Memory. V-Memory is one of the memory areas which is backed up by a battery when there is no power to the controller. The APT designation for the V-Memory location that stores device status is the **.VFLAGS** extension to the device. After the recovery from a power failure, the device status data contained in V-Memory is copied back into the appropriate control relays. All devices then go to their safe-state positions.

NOTE: The device **.VFLAGS** extension, like the V-Flags used in the controller's analog alarm or loop code, is an integer word that stores status information; the device **.VFLAGS** integer extension and the V-Flags for a loop or analog alarm are not otherwise related.

APT does not generate power-fail recovery logic for user-defined devices. If your process requires power-fail recovery logic, you can write code based on the boolean extensions, status boolean array elements, or **.VFLAGS** integer extension for your user-defined devices; see Section 6.5.

If you do select the **UPS** option in the Compile Control File, the states of the device status control relays are not copied into V-Memory, and the **.VFLAGS** extension does not contain meaningful information. After recovery from power failure all devices enter the de-energized state and are unlocked. This action is not controlled.

NOTE: The battery backup on your Series 505 controller must be enabled in order for the controller to save data during a power failure.

Power-Fail Recovery Logic for S5

For S5 controllers, all memory used for APT devices is retentive. Consequently, the power-fail recovery logic that writes the status of a device to the device's **.VFLAGS** integer extension is not necessary. Also, APT does not generate power-fail recovery logic for your devices if you have an S5 controller.

If your process requires power-fail recovery logic, you can write code based on the boolean extensions or status boolean array elements of your devices; see Section 6.5. For S5 controllers, the device **.VFLAGS** integer extension does not exist.

After recovery from power failure, all S5 devices enter their de-energized states and are unlocked. The devices must be unlocked and returned to their desired position.

NOTE: The battery backup on your S5 controller must be enabled in order for the controller to save data during a power failure.

6.5 Device Status

You can monitor or change the status of a device by examining one of the following memory locations.

- Device boolean extensions (read/write)
- Device status boolean array (read/write)
- Device **.VFLAGS** integer extension (read only) (Series 505 only)

NOTE: For S5 controllers, the device status boolean array is numbered backwards from the Series 505. The examples in this section use the Series 505 numbering. For example the MSS device **.RUNNG** extension maps to **.STATUS [1]** for Series 505, and to **.STATUS [16]** for S5.

The boolean extension and the status boolean array represent two ways to access the same memory location. For example, you can achieve the same results by writing a value to a **.DSBLD** extension or by writing to the array **.STATUS [6]**.

You can do a bit test on the **.VFLAGS** extension of a device to examine the status of the device. Because the **.VFLAGS** extension is read only, you cannot change the status by writing to this extension.

The relationship of these different memory locations is shown for the APT devices for Series 505 controllers in Figure 6-5 through Figure 6-8.

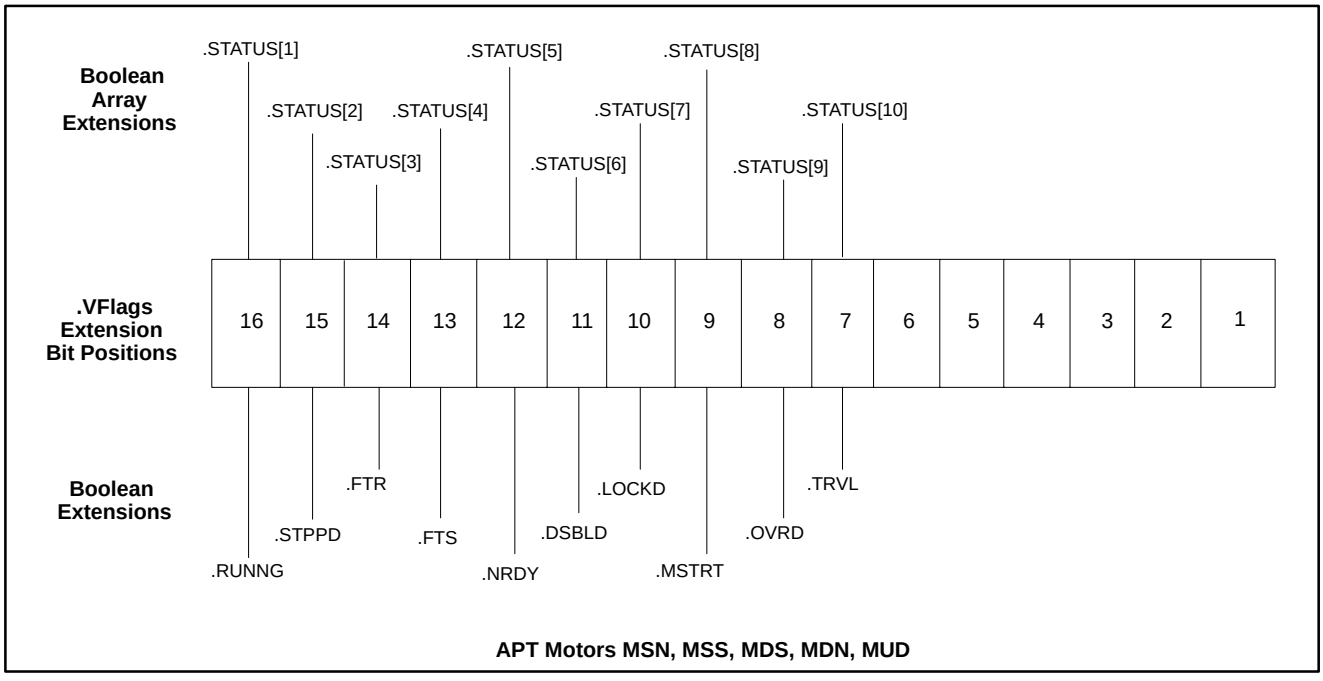
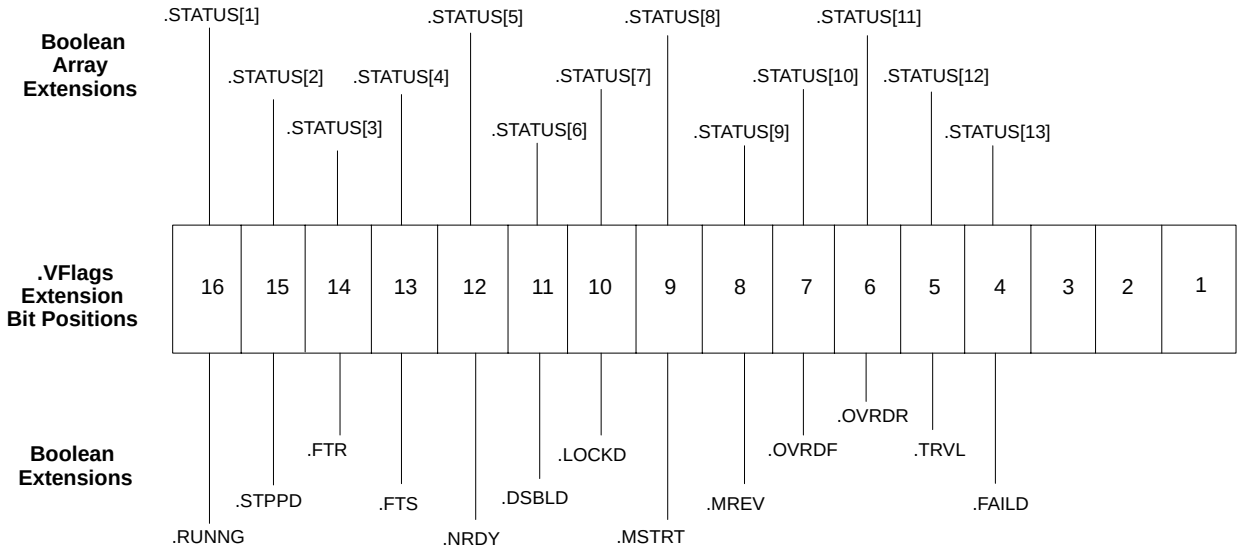
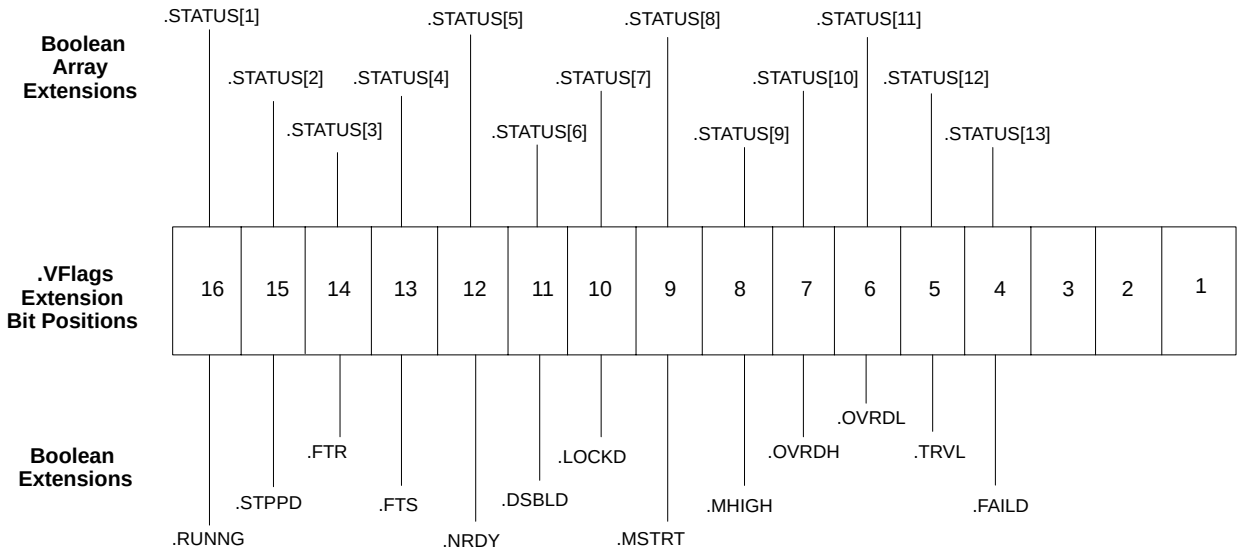


Figure 6-5 Memory Locations Showing Status for APT Motors



APT Motors RM1, RM2



APT Motors TS1, TS2

Figure 6-5 Memory Locations Showing Status for APT Motors (continued)

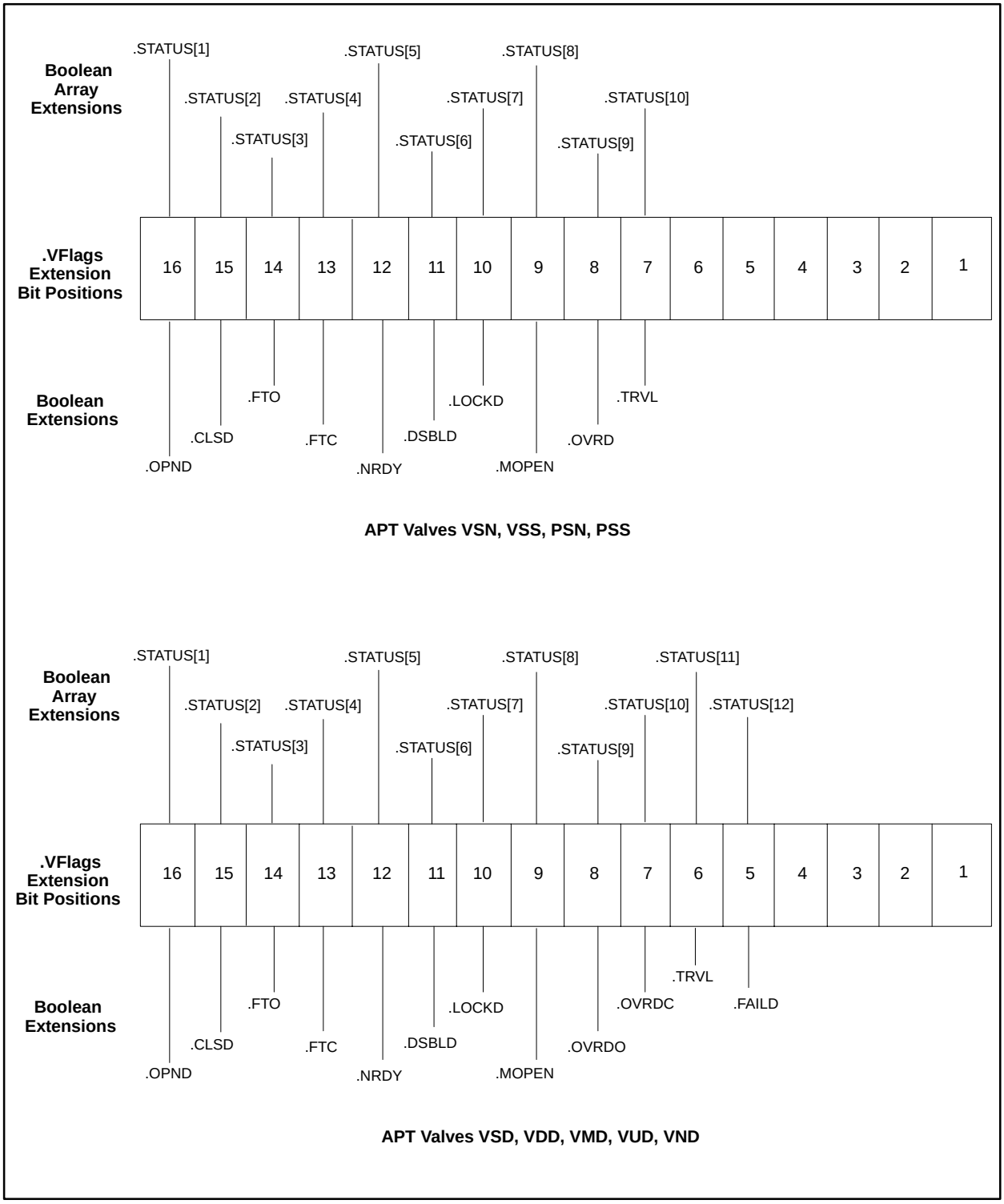


Figure 6-6 Memory Locations Showing Status for APT Valves

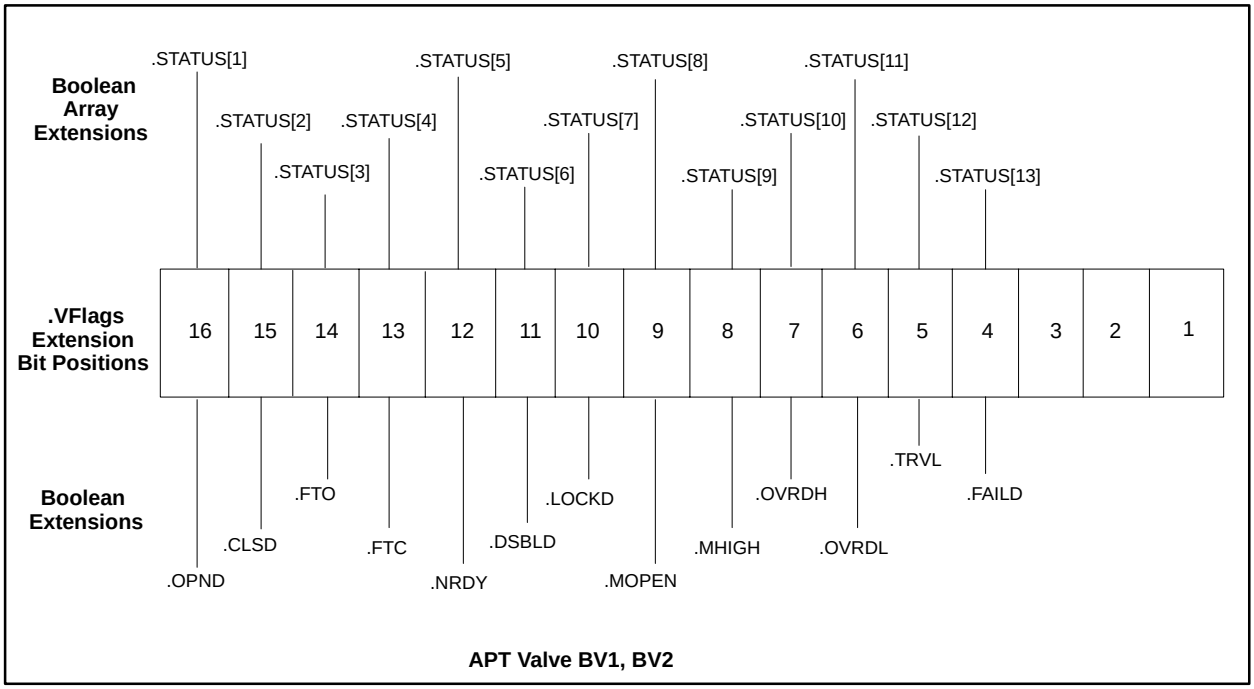


Figure 6-6 Memory Locations Showing Status for APT Valves (continued)

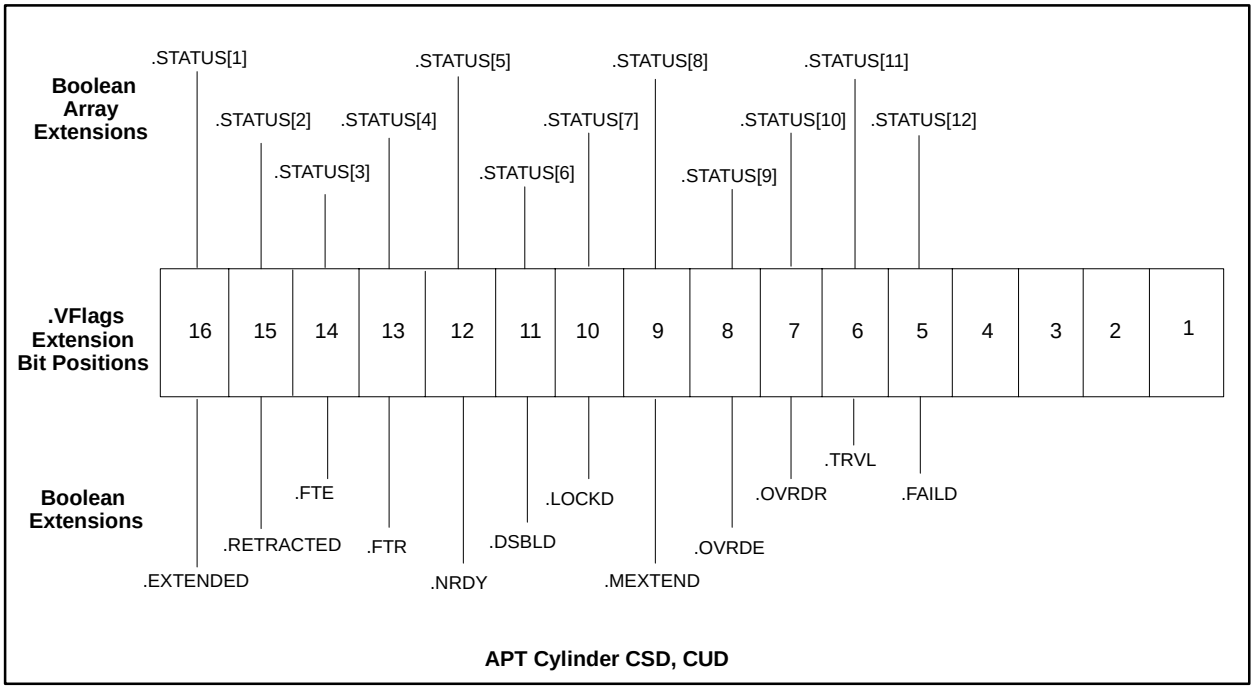


Figure 6-7 Memory Locations Showing Status for APT Cylinders

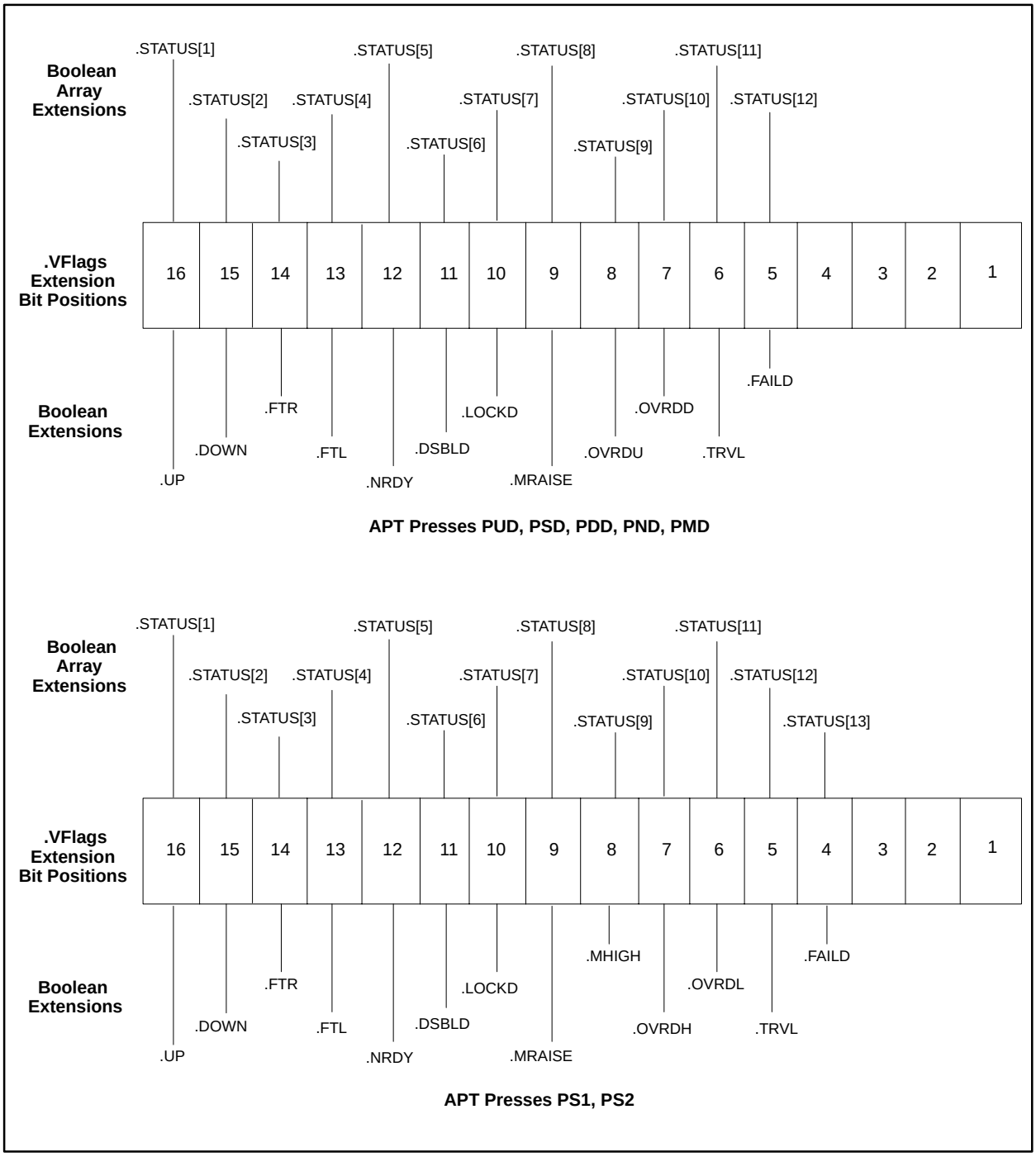


Figure 6-8 Memory Locations Showing Status for APT Presses

Chapter 7

Device Types

7.1 Valves

Hand-Operated/ Dual-Feedback Valve (VND)

The VND device has two positions (open and closed) and is controlled by two discrete feedback signals.

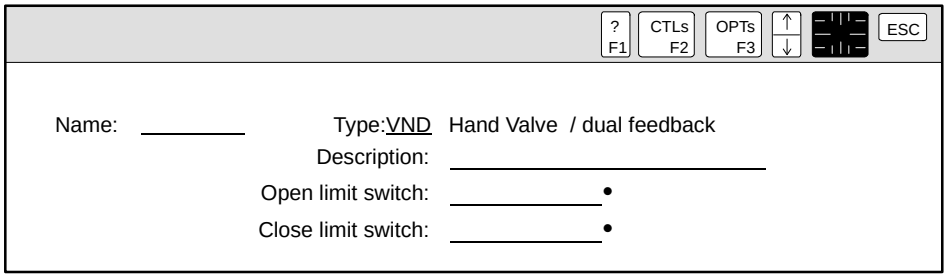
The two feedback signals consist of an open feedback signal (**.OLS**) and a closed feedback signal (**.CLS**).

- The **.OLS** bit should be true when the valve is open and false when it is closed.
- The **.CLS** bit should be true when the valve is closed and false when it is open.

The **.MOPEN** extension shows the state of the valve.

- If the valve is open (**.OLS**=true), the control signal (**.MOPEN**) is set to true.
- If the desired state is closed, (**.CLS**=true), the **.MOPEN** bit is set to false.

When you select VND in the Device Definition Table, this form appears:



Name: unique name that identifies valve (12 characters maximum)

Type: VND (3-letter code for device)

Description: 30 characters maximum (optional)

Open limit switch: symbolic name of discrete input, discrete output, digital flag, boolean or APT flag variable, or *device_name*.**OLS** that is open feedback signal.

Close limit switch: symbolic name of discrete input, discrete output, digital flag, boolean or APT flag variable, or *device_name*.**CLS** that is close feedback signal.

NOTE: The VND does not support the dual feedback logic. Refer to Appendix A for the logic of a VND.

Table 7-1 lists the extensions and commands used with a VND device.

Table 7-1 VND Extensions and Commands

Extension		Commands
Read-only Boolean		There are no commands.
.OPND	opened	
.CLSD	closed	
.OLS [1]	open feedback	
.CLS [2]	closed feed back	
.FAILD	failed (both feedback bits are true)	
.MOPEN	open/close status	
.STATUS	device status	

1 *device_name*.**OLS** is interchangeable with symbolic name configured as Open Limit Switch for energize-open valves. If feedback is not a DI, this is read/write.

2 *device_name*.**CLS** is interchangeable with symbolic name configured as Close Limit Switch for energize-closed valves. If feedback is not a DI, this is read/write.

Figure 7-1 shows how the input bits (on the left) affect the status bits (across the bottom) and/or the output bits (on the right).

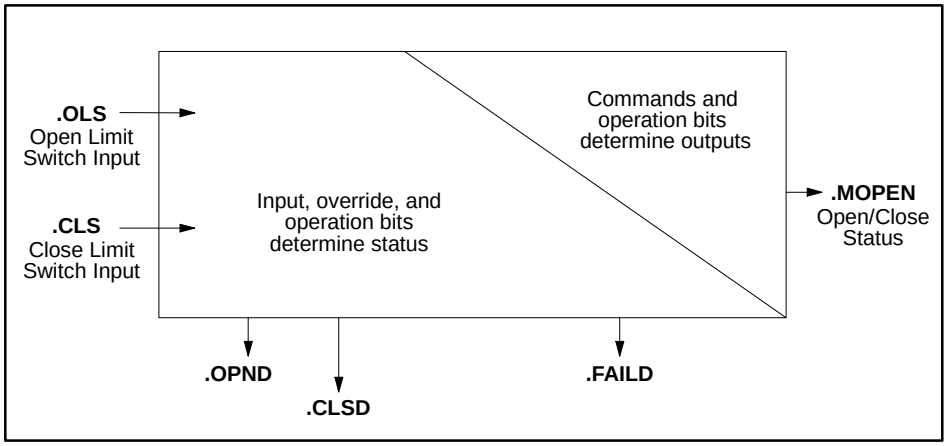


Figure 7-1 VND Extensions and Commands

Valves (continued)

Single-Drive/ Null-Feedback Valve (VSN)

The VSN device has two positions (open and closed) and is controlled by one discrete signal with no feedback. Two types of VSN devices are available: energize-open (Type O) and energize-closed (Type C).

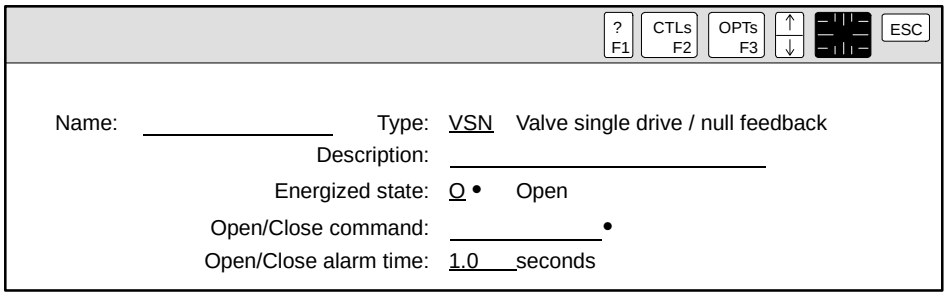
- If the desired state of an energize-open valve is open (**.MOPEN**=true), the field control signal (**.CMMD**) is set to true.

  If the desired state is closed (**.MOPEN**=false), the **.CMMD** bit is set to false.

- If the desired state of an energize-closed valve is open (**.MOPEN**=true), the control signal (**.CMMD**) is set to false.

  If the desired state is closed (**.MOPEN**=false), the **.CMMD** bit is set to true.

When you select VSN in the Device Definition Table, this form appears:



Name: unique name that identifies valve (12 characters maximum)

Type: VSN (3-letter code for device)

Description: 30 characters maximum (optional)

Energized state: O indicates energize open; C indicates energize closed; default is O.

Open/**Close command**: symbolic name of discrete output, boolean variable, or *device_name***.CMMD** that opens and closes valve.

Open/**Close alarm time**: real number between 0.1 and 3276.7 that indicates number of seconds to allow valve to change states.

Table 7-2 lists the extensions and commands used with a VSN device.

Table 7-2 VSN Extensions and Commands

Extension		Commands	
Read-only Boolean		LOCK	place in auto mode
.CMMD [1]	open/close command	UNLOCK	place in manual mode
.OPND	opened	OPEN	open valve
.CLSD	closed	CLOSE	close valve
.TRVL	traveling		
Read/Write Boolean			
.DSBLD	forced to manual mode		
.LOCKD	locked (auto mode)		
.NRDY	not ready		
.MOPEN	manual open		
.STATUS	device status		
Read-only Integer			
.VFLAGS	device status (505 only)		
Read/Write Integer			
.OTCP	open timer/counter preset		
.OTCC	open timer/counter current		
.CTCP	close timer/counter preset		
.CTCC	close timer/counter current		

1 *device_name*.**CMMD** is interchangeable with symbolic name configured as Open/Close command.

Figure 7-2 shows how the operation bits (across the top) affect the status bits (across the bottom) and/or the output bits (on the right).

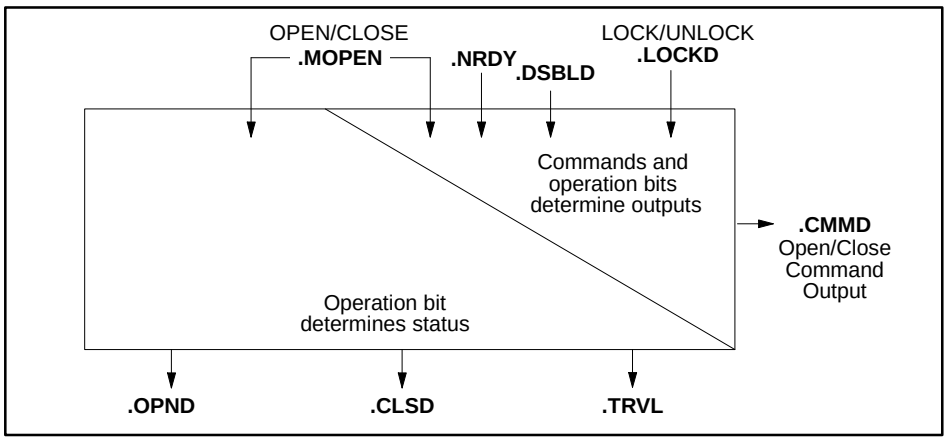


Figure 7-2 VSN Extensions and Commands

Valves (continued)

Single-Drive/ Single-Feedback Valve (VSS)

The VSS device has two positions (open and closed) and is controlled by a single discrete signal with one discrete feedback signal. Two types of VSS devices are available: energize-open (Type O) and energize-closed (Type C).

- If the desired state of an energize-open valve is open (**.MOPEN**=true), the control signal (**.CMMD)** is set to true. If the desired state is closed (**.MOPEN**=false), the **.CMMD** bit is set to false.

  The feedback signal for the energize-open valve (**.OLS**) should be true when the valve is open and false when the valve is closed.

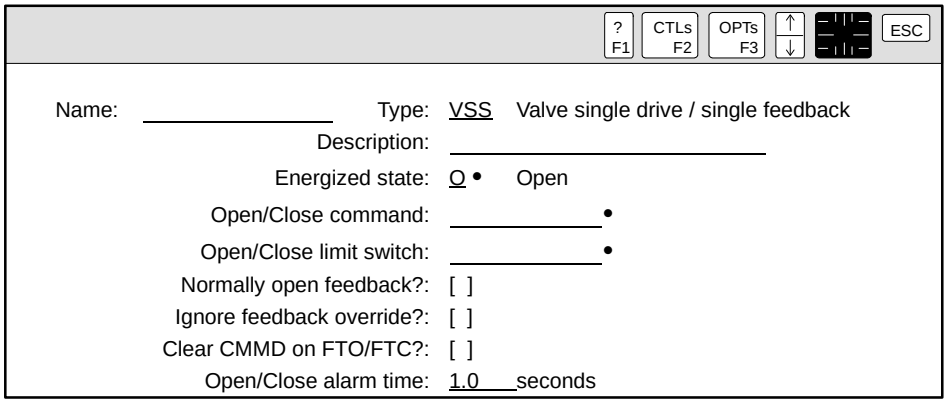
- If the desired state of an energize-close valve is open (**.MOPEN**=true), the control signal (**.CMMD)** is set to false. If the desired state is closed (**.MOPEN**=false), the **.CMMD** bit is set to true.

  The feedback signal for the energize-close valve (**.CLS**) should be false when the valve is open and true when the valve is closed.

- If the **Clear CMMD on FTO/FTC** option is selected, the **.CMMD** bit changes to false when the **.FTR** becomes true. The **.CMMD** bit remains false until a RESET command is issued.

- The RESET command issues an OPEN/CLOSE command that turns on the **.TRVL** bit. The OPEN/CLOSE alarm timer starts counting down when the RESET bit goes false.

When you select VSS in the Device Definition Table, this form appears:



Name: unique name that identifies valve (12 characters maximum)

Type: VSS (3-letter code for device)

Description: 30 characters maximum (optional)

Energized state: O indicates energize open; C indicates energize closed; default is O.

Open/Close command: symbolic name of discrete output, boolean variable, or *device_name*.**CMMD** that opens and closes valve.

Open/Close limit switch: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.**OLS** that is the feedback signal.

Normally open feedback: feedback response is inverted, e.g., open feedback is false when the valve is opened.

Ignore feedback override: Override feedback bit is ignored.

Clear CMMD on FTO/FTC: the **.CMMD** bit becomes false when the **.FTO** or **.FTC** bit becomes true.

Open/Close alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow valve to change states.

Valves (continued)

Table 7-3 lists the extensions and commands used with a VSS device.

Table 7-3 VSS Extensions and Commands

Extension		Commands	
Read-only Boolean		LOCK	place in auto mode
.CMMD [1]	open/close command	UNLOCK	place in manual mode
.OPND	opened	OPEN	open valve
.CLSD	closed	CLOSE	close valve
.TRVL	traveling	RESET	clear feedback override and/or issues open/close command after **.FTO** or **.FTC** is true.
.OLS [2]	open feedback (O)		
.CLS [3]	closed feedback (C)		
.FTO	fail to open		
.FTC	fail to close		
Read/Write Boolean			
.DSBLD	forced to manual mode		
.LOCKD	locked (auto mode)		
.NRDY	not ready		
.MOPEN	manual open		
.OVRD	override feedback		
.STATUS	device status		
Read-only Integer			
.VFLAGS	device status (505 only)		
Read/Write Integer			
.OTCP	open timer/counter preset		
.OTCC	open timer/counter current		
.CTCP	close timer/counter preset		
.CTCC	close timer/counter current		

1 *device_name*.**CMMD** is interchangeable with symbolic name configured as Open/Close command.
2 *device_name*.**OLS** is interchangeable with symbolic name configured as Open Limit Switch for energize-open valves. If feedback is not a DI, this is read/write.
3 *device_name*.**CLS** is interchangeable with symbolic name configured as Close Limit Switch for energize-closed valves. If feedback is not a DI, this is read/write.

Figure 7-3 shows how the input bits (on the left) and the operation bits (across the top) affect the status bits (across the bottom) and/or the output bits (on the right).

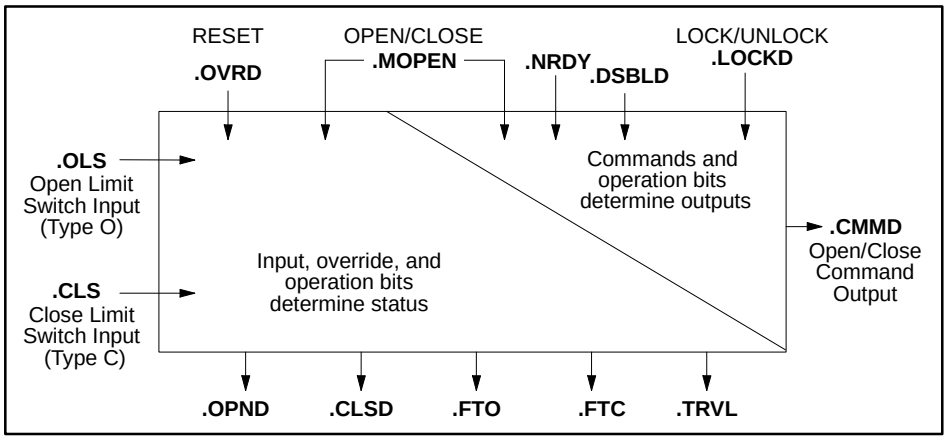


Figure 7-3 VSS Extensions and Commands

Valves (continued)

Single-Drive/ Dual-Feedback Valve (VSD)

The VSD device has two positions (open and closed) and is controlled by a single discrete signal with two discrete feedback signals. Two types of VSD devices are supported: energize-open (Type O) and energize-closed (Type C).

One control signal (**.CMMD**) determines the state of the valve.

- If the desired state of an energize-open valve is open (**.MOPEN**=true), the control signal (**.CMMD**) is set to true.

  If the desired state is closed, (**.MOPEN**=false), the **.CMMD** bit is set to false.

- If the desired state of an energize-closed valve is open (**.MOPEN**=true), the control signal (**.CMMD**) is set to false.

  If the desired state is closed, (**.MOPEN**=false), the **.CMMD** bit is set to true.

- If the **Clear CMMD on FTO/FTC** option is selected, the **.CMMD** bit changes to false when the **.FTR** becomes true. The **.CMMD** bit remains false until a RESET command is issued.

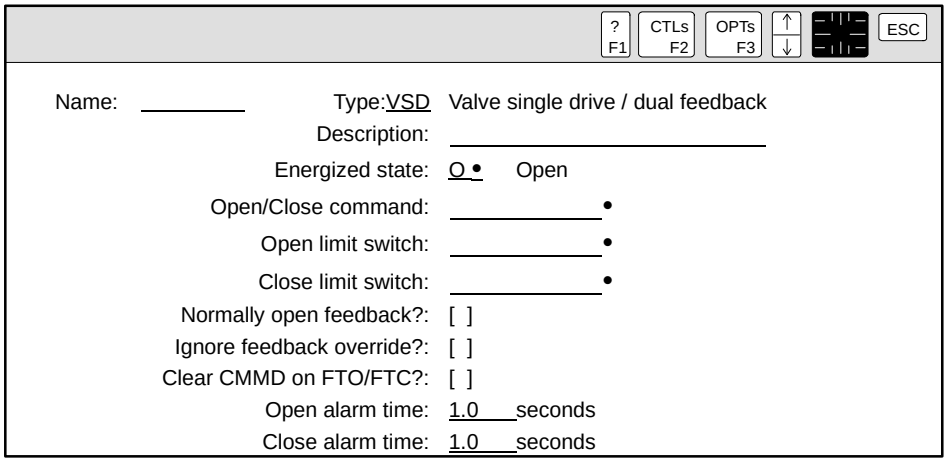
- The RESET command issues an OPEN/CLOSE command that turns on the **.TRVL** bit. The OPEN/CLOSE alarm timer starts counting down when the RESET bit goes false.

For both types of valves, the two feedback signals consist of an open feedback signal (**.OLS**) and a closed feedback signal (**.CLS**).

- The **.OLS** bit should be true when the valve is open and false when it is closed.

- The **.CLS** bit should be true when the valve is closed and false when it is open.

When you select VSD in the Device Definition Table, this form appears:



Name: unique name that identifies valve (12 characters maximum)

Type: VSD (3-letter code for device)

Description: 30 characters maximum (optional)

Energized state: O indicates energize open; C indicates energize closed; default is O.

Open/Close command: symbolic name of discrete output, boolean variable, or *device_name*.**CMMD** that opens and closes valve

Open limit switch: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.**OLS** that is open feedback signal.

Close limit switch: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.**CLS** that is close feedback signal.

Normally open feedback: feedback response is inverted, e.g, open feedback is false when the valve is opened.

Ignore feedback override: Override feedback bit is ignored.

Clear CMMD on FTO/FTC: the **.CMMD** bit becomes false when the **.FTO** or **.FTC** bit becomes true.

Open alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow valve to open.

Close alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow valve to close.

Valves (continued)

Table 7-4 lists the extensions and commands used with a VSD device.

Table 7-4 VSD Extensions and Commands

Extension		Commands	
Read-only Boolean		LOCK	place in auto mode
.CMMD [1]	open/close command	UNLOCK	place in manual mode
.OPND	opened	OPEN	open valve
.CLSD	closed	CLOSE	close valve
.TRVL	traveling	RESET	clear feedback override and/or issue open/close command after **.FTO** or **.FTC** is true.
.OLS [2]	open feedback (O)		
.CLS [3]	closed feedback (C)		
.FTO	fail to open		
.FTC	fail to close		
.FAILD	failed (both feedback bits are true)		
Read/Write Boolean			
.DSBLD	forced to manual mode		
.LOCKD	locked (auto mode)		
.NRDY	not ready		
.MOPEN	manual open		
.OVRDO	override open feedback		
.OVRDC	override closed feedback		
.STATUS	device status		
Read-only Integer			
.VFLAGS	device status (505 only)		
Read/Write Integer			
.OTCP	open timer/counter preset		
.OTCC	open timer/counter current		
.CTCP	close timer/counter preset		
.CTCC	close timer/counter current		

1 *device_name*.**CMMD** is interchangeable with symbolic name configured as Open/Close command.
2 *device_name*.**OLS** is interchangeable with symbolic name configured as Open Limit Switch for energize-open valves. If feedback is not a DI, this is read/write.
3 *device_name*.**CLS** is interchangeable with symbolic name configured as Close Limit Switch for energize-closed valves. If feedback is not a DI, this is read/write.

Figure 7-4 shows how the input bits (on the left) and the operation bits (across the top) affect the status bits (across the bottom) and/or the output bits (on the right).

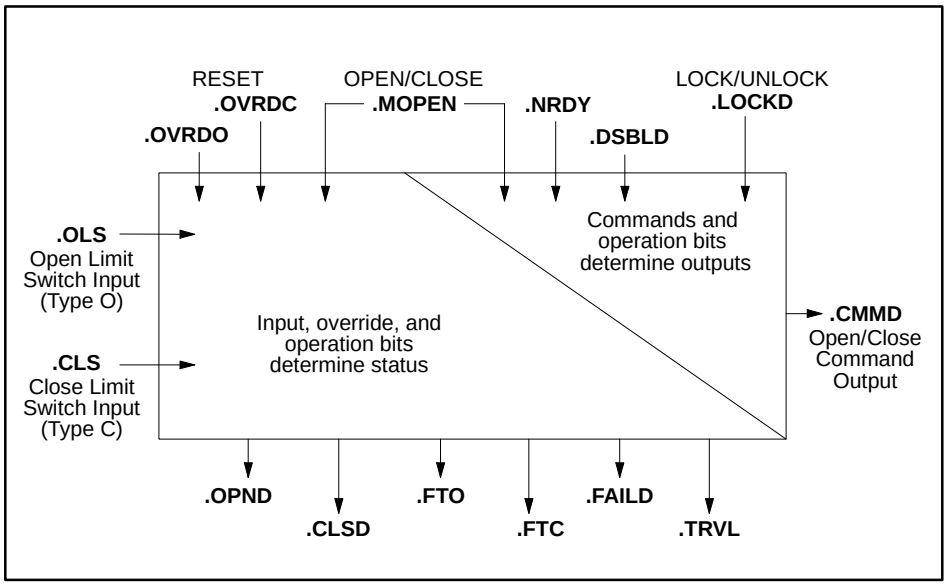


Figure 7-4 VSD Extensions and Commands

Valves (continued)

Dual-Drive/ Dual-Feedback Valve (VDD)

The VDD device has two positions (open and closed) and is controlled by two discrete signals with two discrete feedback signals.

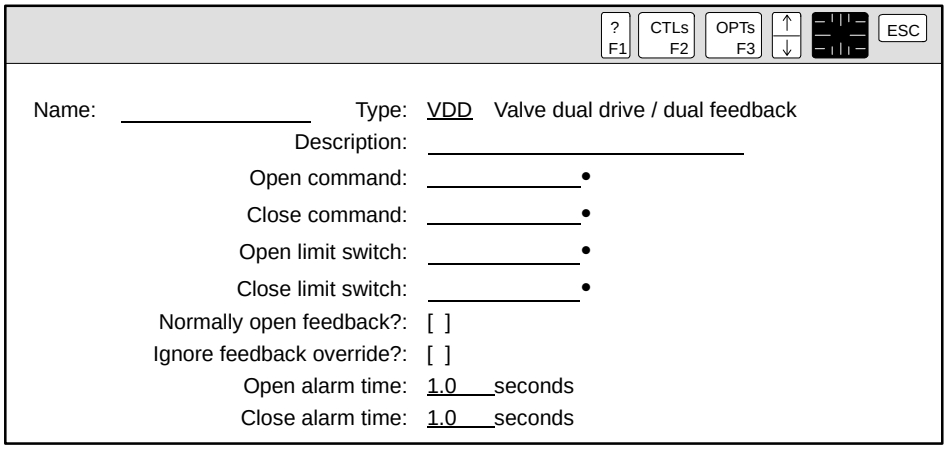
The two control signals consist of an open signal (**.OPENC**) and a close signal (**.CLSC**), which are both normally false.

- If the desired state is open (**.MOPEN**=true), the **.OPENC** bit is set to true. The **.OPENC** bit remains true until either the open feedback signal is true or the open alarm time expires; then **.OPENC** is set to false.
- If the desired state is closed, the **.CLSC** bit is set to true to close the valve. The **.CLSC** bit remains true until either the close feedback signal is true or the close alarm time expires; then **.CLSC** is set to false.
- The RESET command issues an OPEN/CLOSE command that turns on the **.TRVL** bit. The OPEN/CLOSE alarm timer starts counting down when the RESET bit goes false.

The two feedback signals consist of an open feedback signal (**.OLS**) and a closed feedback signal (**.CLS**).

- The **.OLS** bit should be true when the valve is open; otherwise, it should be false.
- The **.CLS** bit should be true when the valve is closed; otherwise, it should be false.

When you select VDD in the Device Definition Table, this form appears:



Name: unique name that identifies valve (12 characters maximum)

Type: VDD (3-letter code for device)

Description: 30 characters maximum (optional)

Open command: symbolic name of discrete output, boolean variable, or *device_name*.**OPENC** that opens valve.

Close command: symbolic name of discrete output, boolean variable, or *device_name*.**CLSC** that closes valve.

Open limit switch: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.**OLS** that is open feedback signal.

Close limit switch: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.**CLS** that is close feedback signal.

Normally open feedback: feedback response is inverted, e.g, open feedback is false when the valve is opened.

Ignore feedback override: Override feedback bit is ignored.

Open alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow valve to open.

Close alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow valve to close.

Valves (continued)

Table 7-5 lists the extensions and commands used with a VDD device.

Table 7-5 VDD Extensions and Commands

Extension		Commands	
Read-only Boolean		LOCK	place in auto mode
.CMMD	open/close command	UNLOCK	place in manual mode
.OPENC [1]	open command	OPEN	open valve
.CLSC [2]	close command	CLOSE	close valve
.OPND	opened	RESET	clear feedback override
.CLSD	closed		
.TRVL	traveling		
.OLS [3]	open feedback		
.CLS [4]	closed feedback		
.FTO	fail to open		
.FTC	fail to close		
.FAILD	failed (both feedback bits are true)		
.CLSTO	close timeout		
.OPNTO	open timeout		
Read/Write Boolean			
.DSBLD	forced to manual mode		
.LOCKD	locked (auto mode)		
.NRDY	not ready		
.MOPEN	manual open		
.OVRDO	override open feedback		
.OVRDC	override closed feedback		
.STATUS	device status		
Read-only Integer			
.VFLAGS	device status (505 only)		
Read/Write Integer			
.OTCP	open timer/counter preset		
.OTCC	open timer/counter current		
.CTCP	close timer/counter preset		
.CTCC	close timer/counter current		

1 *device_name*.**OPENC** is interchangeable with symbolic name configured as Open command.
2 *device_name*.**CLSC** is interchangeable with symbolic name configured as Close command.
3 *device_name*.**OLS** is interchangeable with symbolic name configured as Open Limit Switch. If feedback is not a DI, this is read/write.
4 *device_name*.**CLS** is interchangeable with symbolic name configured as Close Limit Switch. If feedback is not a DI, this is read/write.

Figure 7-5 shows how the input bits (on the left) and the operation bits (across the top) affect the status bits (across the bottom) and/or the output bits (on the right).

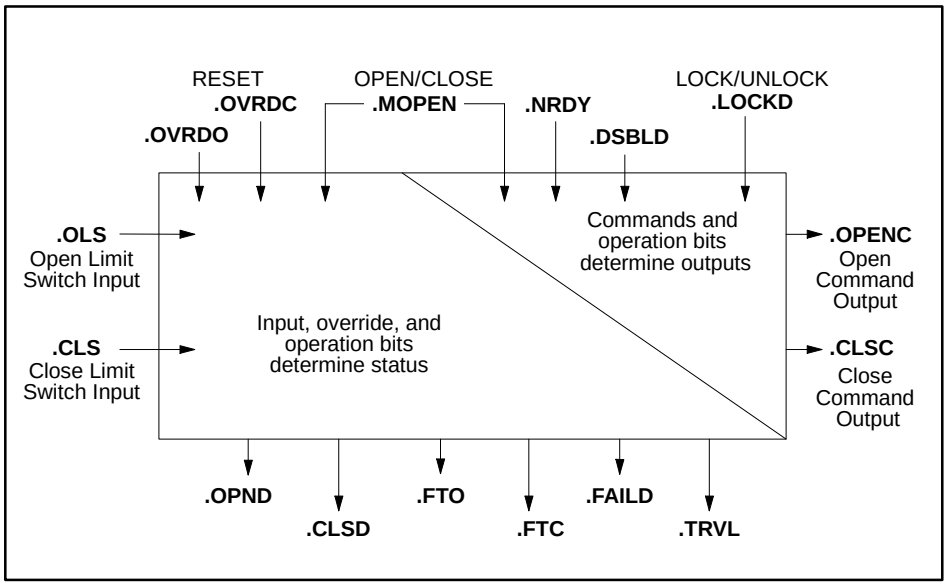


Figure 7-5 VDD Extensions and Commands

Valves (continued)

Motor-Drive/ Dual-Feedback Valve (VMD)

The VMD device has two positions (open and closed) and is controlled by two discrete signals with two discrete feedback signals.

The two control signals consist of a start signal (**.OPENC**) and a stop signal (**.CLSC**).

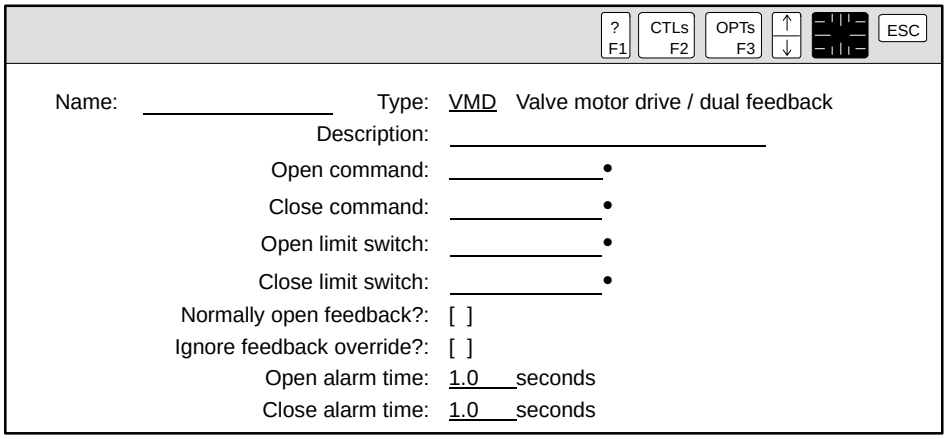
- If the desired state of the valve is open (**.MOPEN**=true), the **.OPENC** bit is set to true until the open feedback is true or until the alarm time expires; then **.OPENC** is set to false.
- If the desired state is closed (**.MOPEN**=false), the **.CLSC** bit is set to true until closed feedback is true or until the alarm time expires; then **.CLSC** is set to false.
- If the valve is stopped in mid-travel, the **.TRVL** bit remains true with the open and close alarm times reset.
- The RESET command issues an OPEN/CLOSE command that turns on the **.TRVL** bit. The OPEN/CLOSE alarm timer starts counting down when the RESET bit goes false.

The two feedback signals consist of an open feedback signal (**.OLS**) and a closed feedback signal (**.CLS**).

- The **.OLS** bit should be true when the valve is open; otherwise, it should be false.
- The **.CLS** bit should be true when the valve is closed; otherwise, it should be false.

The VMD valve can be stopped at any point of its travel by setting the **.RTS** extension to true.

When you select VMD in the Device Definition Table, this form appears:



Name: unique name that identifies motor-driven valve (12 characters maximum)

Type: VMD (3-letter code for device)

Description: 30 characters maximum (optional)

Open command: symbolic name of discrete output, boolean variable, or *device_name*.**OPENC** that opens motor valve.

Close command: symbolic name of discrete output, boolean variable, or *device_name*.**CLSC** that closes motor valve.

Open limit switch: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.**OLS** that is open feedback signal.

Close limit switch: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.**CLS** that is close feedback signal.

Normally open feedback: feedback response is inverted, e.g, open feedback is false when the valve is opened.

Ignore feedback override: Override feedback bit is ignored.

Open alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow valve to open; also indicates maximum time to keep open signal true.

Close alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow valve to close; also indicates maximum time to keep close signal true.

Valves (continued)

Table 7-6 lists the extensions and commands used with a VMD device.

Table 7-6 VMD Extensions and Commands

Extension		Commands	
	Read-only Boolean	LOCK	place in auto mode
.CMMD	open/close command	UNLOCK	place in manual mode
.OPENC [1]	open command	OPEN	open valve
.CLSC [2]	close command	CLOSE	close valve
.OPND	opened	RESET	clear feedback override
.CLSD	closed		
.TRVL	traveling		
.OLS [3]	open feedback		
.CLS [4]	closed feedback		
.FTO	fail to open		
.FTC	fail to close		
.FAILD	failed (both feedback bits are true)		
.CLSTO	close timeout		
.OPNTO	open timeout		
	Read/Write Boolean		
.DSBLD	forced to manual mode		
.LOCKD	locked (auto mode)		
.NRDY	not ready		
.MOPEN	manual open		
.OVRDO	override open feedback		
.OVRDC	override closed feedback		
.STATUS	device status		
.RTS	stop travel		
	Read-only Integer		
.VFLAGS	device status (505 only)		
	Read/Write Integer		
.OTCP	open timer/counter preset		
.OTCC	open timer/counter current		
.CTCP	close timer/counter preset		
.CTCC	close timer/counter current		

1 *device_name*.**OPENC** is interchangeable with symbolic name configured as Open command.
2 *device_name*.**CLSC** is interchangeable with symbolic name configured as Close command.
3 *device_name*.**OLS** is interchangeable with symbolic name configured as Open Limit Switch. If feedback is not a $\overline{\text{DI}}$, this is read/write.
4 *device_name*.**CLS** is interchangeable with symbolic name configured as Close Limit Switch. If feedback is not a $\overline{\text{DI}}$, this is read/write.

Figure 7-6 shows how the input bits (on the left) and the operation bits (across the top) affect the status bits (across the bottom) and/or the output bits (on the right).

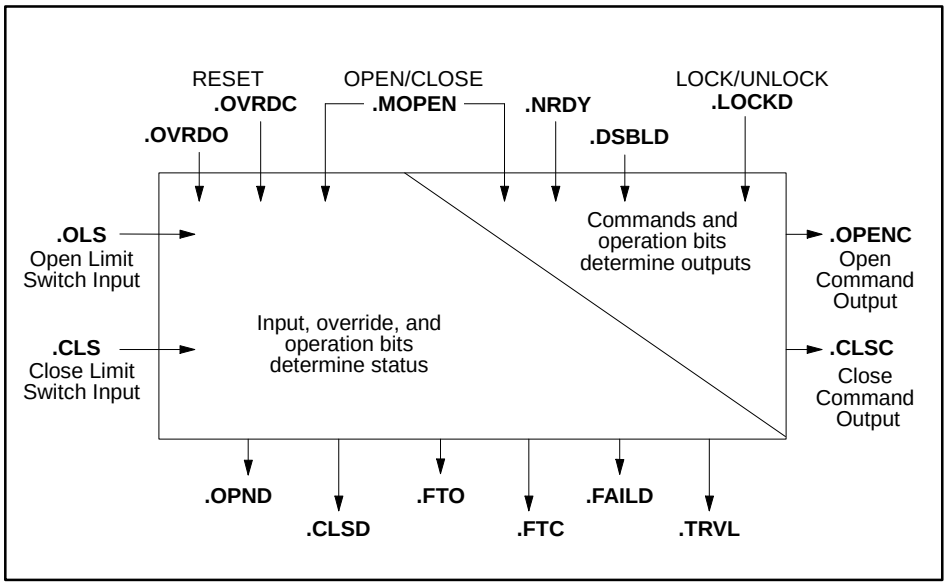


Figure 7-6 VMD Extensions and Commands

Valves (continued)

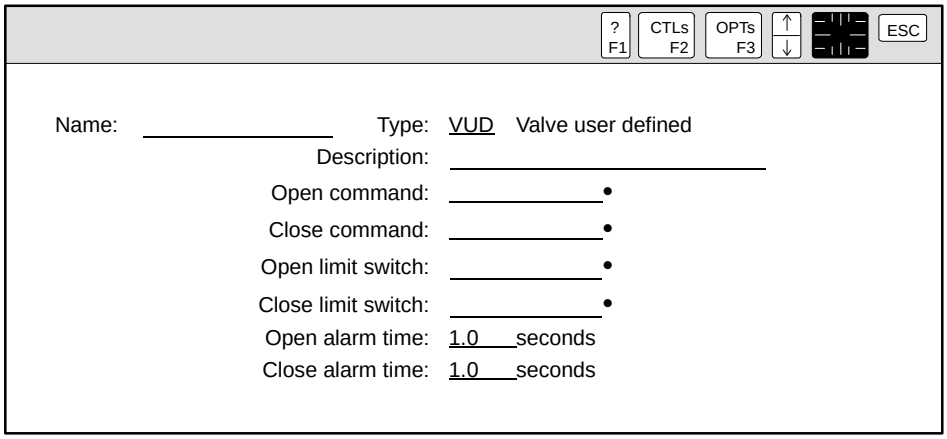
User-defined Valve (VUD)

The VUD device has two positions (open and closed) and is controlled by two discrete signals with two discrete feedback signals.

The VUD is essentially the shell of a VDD device. The code which defines the VDD has been reduced to a minimum, and extensions have been provided to allow you to create your own customized valve and translate it for OSx (PCS).

The VUD has two alarm timers. When the **.ORESET** (**.CRESET**) bit transitions from false to true, the alarm timer starts. When the timer times out, the **.OPNTO** (**.CLSTO**) bit becomes true, and remains true until the **.ORESET** (**.CRESET**) bit becomes false.

When you select VUD in the Device Definition Table, this form appears:



Name: unique name that identifies valve (12 characters maximum)

Type: VUD (3-letter code for device)

Description: 30 characters maximum (optional)

Open command: symbolic name of discrete output, boolean variable, or *device_name*.**OPENC** that opens valve.

Close command: symbolic name of discrete output, boolean variable, or *device_name*.**CLSC** that closes valve.

Open limit switch: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.**OLS** that is open feedback signal.

Close limit switch: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.**CLS** that is close feedback signal.

Open alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow valve to open.

Close alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow valve to close.

Table 7-7 lists the extensions and commands used with a VUD device.

Valves (continued)

Table 7-7 VUD Extensions and Commands

Extension		Commands	
	Read-only Boolean	LOCK	place in auto mode
.CMMD	open/close command	UNLOCK	place in manual mode
.OPENC [1]	open command	OPEN	open valve
.CLSC [2]	close command	CLOSE	close valve
.OPNDx [3]	opened	RESET	clear feedback override
.CLSD	closed	OPENA	open with option A
.TRVL	traveling	OPENB	open with option B
.OLS [4]	open feedback	OPENH	open high
.CLS [5]	closed feedback	OPENL	open low
.FTO	fail to open		
.FTC	fail to close		
.FAILD	failed (both feedback bits are true)		
.OPNTO	open alarm timer time out		
.CLSTO	close alarm timer time out		
	Read/Write Boolean		
.DSBLD	forced to manual mode		
.LOCKD	locked (auto mode)		
.NRDY	not ready		
.MOPEN	manual open		
.OVRDO	override open feedback		
.OVRDC	override closed feedback		
.ORESET	open alarm timer reset		
.CRESET	close alarm timer reset		
.STATUS	device status		
	Read-only Integer		
.VFLAGS	device status (505 only)		
	Read/Write Integer		
.OTCP	open timer/counter preset		
.OTCC	open timer/counter current		
.CTCP	close timer/counter preset		
.CTCC	close timer/counter current		

1 *device_name*.**OPENC** is interchangeable with symbolic name configured as Open command.
2 *device_name*.**CLSC** is interchangeable with symbolic name configured as Close command.
3 The extension is **.OPNDx** where **x**= **A**, **x** = **B**, **x** = **H**, or **x** = **L**.
4 *device_name*.**OLS** is interchangeable with symbolic name configured as Open Limit Switch. If feedback is not a $\overline{\text{DI}}$, this is read/write.
5 *device_name*.**CLS** is interchangeable with symbolic name configured as Close Limit Switch. If feedback is not a $\overline{\text{DI}}$, this is read/write.

Figure 7-7 shows how the input bits (on the left) and the operation bits (across the top) affect the status bits (across the bottom) and/or the output bits (on the right).

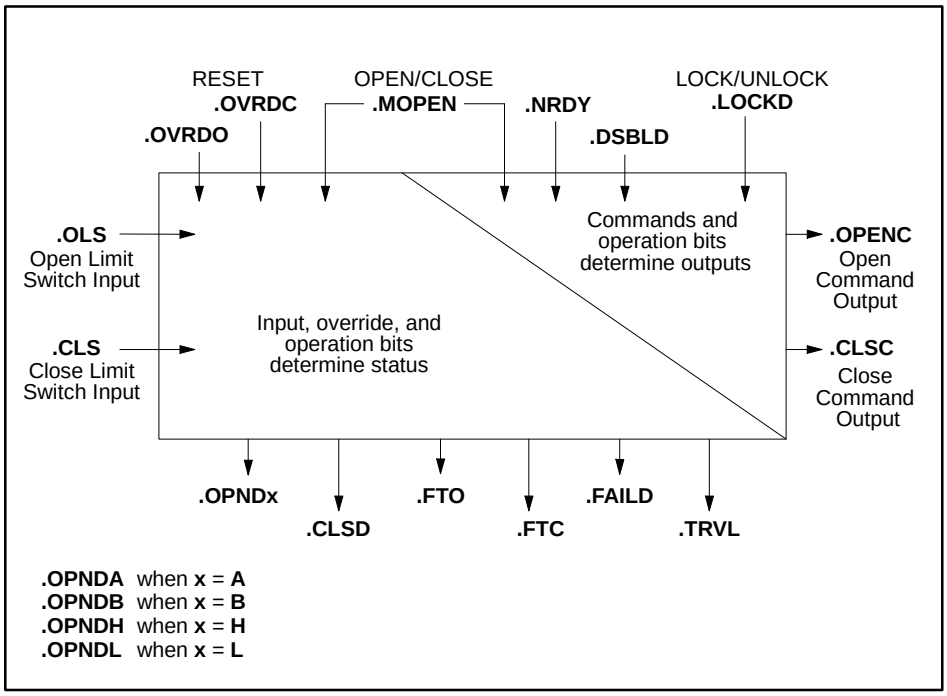


Figure 7-7 VUD Extensions and Commands

7.2 Three-Position Valves with Dual Feedback

Three-Position Valve/Type 1 (BV1)

The BV1 device is a three-position valve (high, low, and closed) and is controlled by two discrete signals with two discrete feedback signals.

The two discrete control signals consist of a low signal (**.SLOW**) and a high signal (**.SHIGH**). The two signals cannot both be true at the same time.

- If the desired state of the valve is open at high position (**.MOPEN**=true and **.MHIGH**=true), the **.SHIGH** bit is set to true (energized) to open the valve at high position.
- If the desired state of the valve is open at low position (**.MOPEN**=true and **.MHIGH**=false), the **.SLOW** bit is set to true (energized) to open the valve at low position.
- If the desired state is closed (**.MOPEN**=false), the currently energized signal (**.SHIGH** or **.SLOW**) is set to false.
- The valve, when receiving an open signal, starts opening and keeps opening until the currently energized signal becomes false.
- You can change the position of a three-position valve without closing it.
- The RESET command issues an OPEN/CLOSE command that turns on the **.TRVL** bit. The OPEN/CLOSE alarm timer starts counting down when the RESET bit goes false.

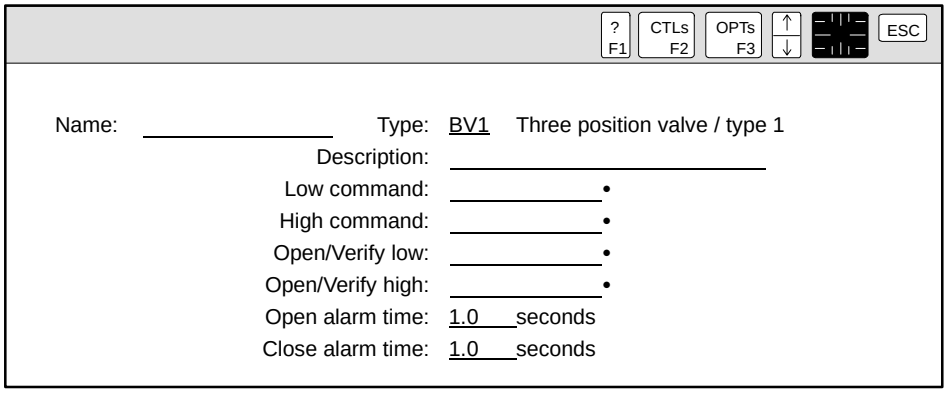
The two feedback signals consist of an open-low feedback signal (**.LIO**) and an open-high feedback signal (**.HIO**).

- The **.LIO** bit should be true when the valve is open at the low position; otherwise, it should be false.
- The **.HIO** bit should be true when the valve is open at the high position; otherwise, it should be false.

In manual mode, the outputs (**.SLOW**/**.SHIGH**) are set to the appropriate state based on the status of the **.MHIGH** and the **.MOPEN** bits, which you manipulate from an operator station or from the program.

In auto mode, the OPENL, OPENH, and CLOSE commands set the state of the output bits. In this mode, the **.MHIGH** and **.MOPEN** bits are set to reflect the last requested state. This is done to provide for a bumpless transfer if the device changes modes.

When you select BV1 in the Device Definition Table, the form appears as shown below.



Name: unique name that identifies valve (12 characters maximum)

Type: BV1 (3-letter code for device)

Description: 30 characters maximum (optional)

Low command: symbolic name of discrete output, boolean variable, or *device_name*.**SLOW** that opens valve at low position.

High command: symbolic name of discrete output, boolean variable, or *device_name*.**SHIGH** that opens valve a high position.

Open/Verify low: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.**LIO** that is low feedback signal.

Open/Verify high: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.**HIO** that is high feedback signal.

Open alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow valve to change from closed to open.

Close alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow valve to change from open to closed.

Table 7-8 lists the extensions and commands used with a BV1 device.

Three-Positioned Valves with Dual Feedback (continued)

Table 7-8 BV1 Extensions and Commands

Extension		Commands	
Read-only Boolean		LOCK	place in auto mode
.SLOW [1]	open low	UNLOCK	place in manual mode
.SHIGH [2]	open high	OPENL	open low
.OPND	opened	OPENH	open high
.OPNDL	opened low	CLOSE	close valve
.OPNDH	opened high	RESET	clear feedback overrides
.CLSD	closed		
.TRVL	traveling		
.LIO [3]	low feedback		
.HIO [4]	high feedback		
.FTOL	fail to open low		
.FTOH	fail to open high		
.FTO [5]	fail to open		
.FTC	fail to close		
.FAILD	failed (both feedback bits are true)		
.OPNTO	open timeout		
.CLSTO	close timeout		
Read/Write Boolean			
.DSBLD	forced to manual mode		
.LOCKD	locked (auto mode)		
.NRDY	not ready		
.MHIGH	manual high		
.MOPEN	manual open		
.OVRDL	override low feedback		
.OVRDH	override high feedback		
.STATUS	device status		
Read-only Integer			
.VFLAGS	device status (505 only)		
Read/Write Integer			
.OTCP	open timer/counter preset		
.OTCC	open timer/counter current		
.CTCP	close timer/counter preset		
.CTCC	close timer/counter current		

1 *device_name*.**SLOW** is interchangeable with symbolic name configured as low command.
2 *device_name*.**SHIGH** is interchangeable with symbolic name configured as high command.
3 *device_name*.**LIO** is interchangeable with symbolic name configured as Run/Verify low feedback signal. If feedback is not a DI, this is read/write.
4 *device_name*.**HIO** is interchangeable with symbolic name configured as Run/Verify high feedback signal. If feedback is not a DI, this is read/write.
5 The **.FTO** bit is set to true when either the **.FTOL** or **.FTOH** bit is true.

Figure 7-8 shows how the input bits (on the left) and the operation bits (across the top) affect the status bits (across the bottom) and/or the output bits (on the right).

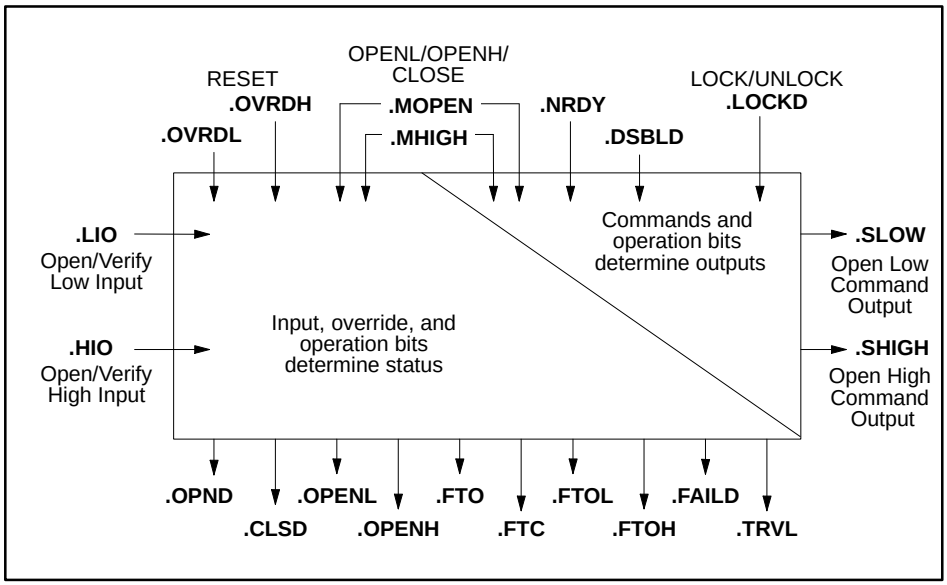


Figure 7-8 BV1 Extensions and Commands

Three-Position Valves with Dual Feedback (continued)

Three-Position Valve/Type 2 (BV2)

The BV2 device is a three-position valve (high, low, and closed) and is controlled by two discrete signals with two discrete feedback signals.

The two discrete control signals consist of an open/close signal (**.DRV**) and a position signal (**.POS**) that determines whether the valve opens at high or low position.

- If the desired state of the valve is open high (**.MOPEN**=true and **.MHIGH**=true), the **.DRV** and **.POS** bits are both set to true.
- If the desired state is open low (**.MOPEN**=true and **.MHIGH**=false), the **.DRV** bit is set to true and the **.POS** bit is set to false.
- If the desired state is closed (**.MOPEN**=false), the **.DRV** bit is set to false to close the valve.
- The valve, when receiving an open signal, starts opening and keeps opening until **.DRV** becomes false.
- You can change the position of a three-position valve without closing it.
- The RESET command issues an OPEN/CLOSE command that turns on the **.TRVL** bit. The OPEN/CLOSE alarm timer starts counting down when the RESET bit goes false.

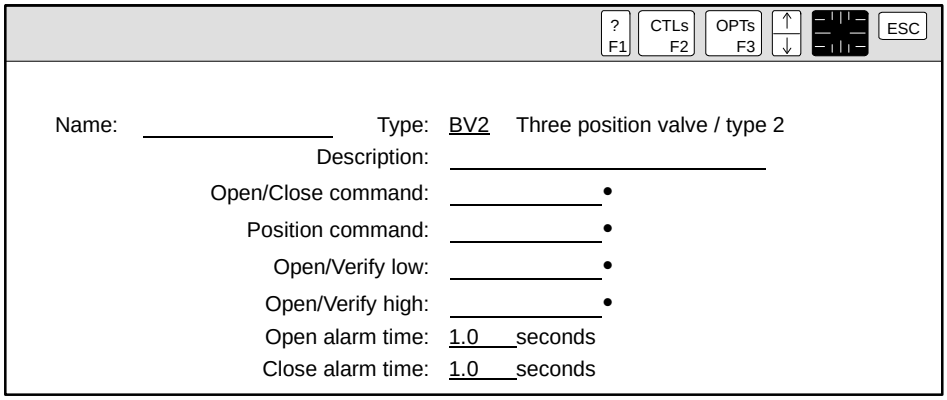
The two feedback signals consist of an open low feedback signal (**.LIO**) and an open high feedback signal (**.HIO**).

- The **.LIO** bit should be true when the valve is open at low position; otherwise, it should be false.
- The **.HIO** should be true when the valve is open at high position; otherwise it should be false.

In manual mode, the outputs (**.DRV**/**.POS**) are set to the appropriate state based on the status of the **.MHIGH** and the **.MOPEN** bits, which you manipulate from an operator station or from the program.

In auto mode, the OPENL, OPENH, and CLOSE commands set the state of the output bits. In this mode, the **.MHIGH** and **.MOPEN** bits are set to reflect the last requested state. This is done to provide for a bumpless transfer if the device changes modes.

When you select BV2 in the Device Definition Table, the form appears as shown below.



Name: unique name that identifies valve (12 characters maximum)

Type: BV2 (3-letter code for device)

Description: 30 characters maximum (optional)

Open/Close command: symbolic name of discrete output, boolean variable, or *device_name*.**DRV** that opens and closes valve.

Position command: symbolic name of discrete output, boolean variable, or *device_name*.**POS** that indicates the position signal.

Open/Verify low: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.**LIO** that is low feedback signal.

Open/Verify high: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.**HIO** that is high feedback signal.

Open alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow valve to change from closed to open.

Close alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow valve to change from open to closed.

Table 7-9 lists the extensions and commands used with a BV2 device.

Three-Positioned Valves with Dual Feedback (continued)

Table 7-9 BV2 Extensions and Commands

Extension		Commands	
Read-only Boolean		LOCK	place in auto mode
.DRV [1]	open/close command	UNLOCK	place in manual mode
.POS [2]	position	OPENL	open low
.OPND	opened	OPENH	open high
.OPNDL	opened low	CLOSE	close valve
.OPNDH	opened high	RESET	clear feedback overrides
.CLSD	closed		
.TRVL	traveling		
.LIO [3]	low feedback		
.HIO [4]	high feedback		
.FTOL	fail to open low		
.FTOH	fail to open high		
.FTO [5]	fail to open		
.FTC	fail to close		
.FAILD	failed (both feedback bits are true)		
.OPNTO	open timeout		
.CLSTO	close timeout		
Read/Write Boolean			
.DSBLD	forced to manual mode		
.LOCKD	locked (auto mode)		
.NRDY	not ready		
.MHIGH	manual high		
.MOPEN	manual open		
.OVRDL	override low feedback		
.OVRDH	override high feedback		
.STATUS	device status		
Read-only Integer			
.VFLAGS	device status (505 only)		
Read/Write Integer			
.OTCP	open timer/counter preset		
.OTCC	open timer/counter current		
.CTCP	close timer/counter preset		
.CTCC	close timer/counter current		

1 *device_name*.**DRV** is interchangeable with symbolic name configured as open/close command.
2 *device_name*.**POS** is interchangeable with symbolic name configured as position command.
3 *device_name*.**LIO** is interchangeable with symbolic name configured as Open/Verify low feedback signal.
4 *device_name*.**HIO** is interchangeable with symbolic name configured as Open/Verify high feedback signal.
5 The **.FTO** bit is set to true when either the **.FTOL** or **.FTOH** bit is true.

Figure 7-9 shows how the input bits (on the left) and the operation bits (across the top) affect the status bits (across the bottom) and/or the output bits (on the right).

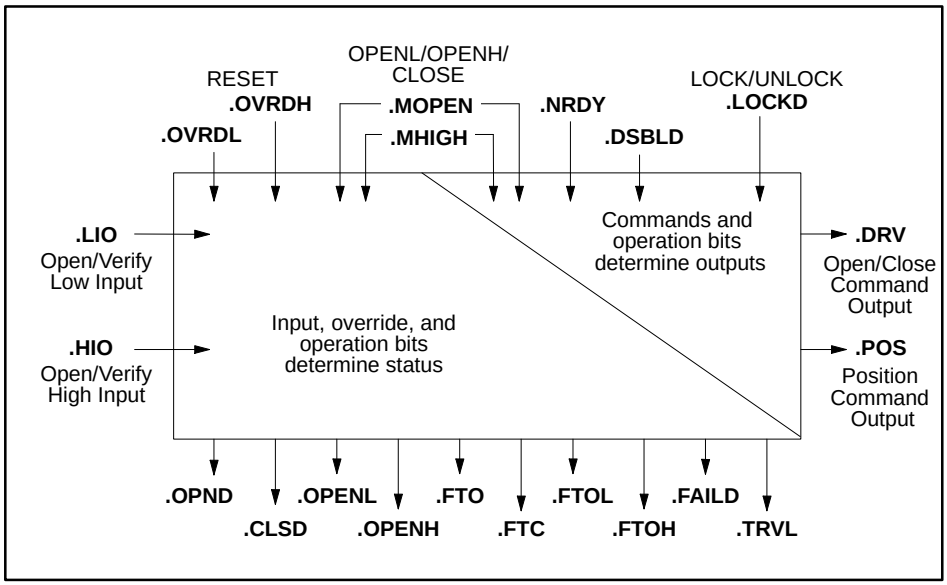


Figure 7-9 BV2 Extensions and Commands

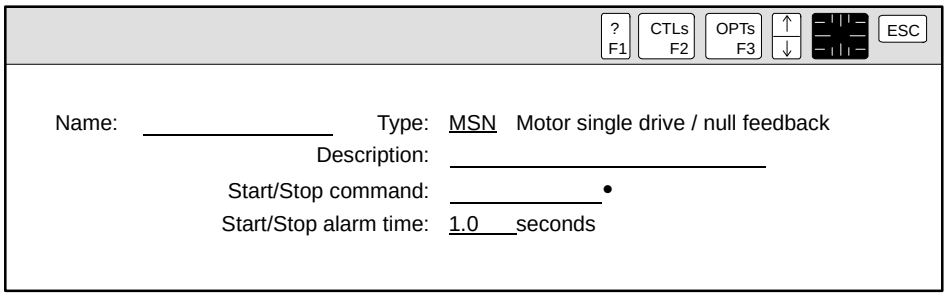
7.3 Motor

Single-Drive/ Null-Feedback Motor (MSN)

The MSN device is a two-state motor (running and stopped) and is controlled by a single discrete signal with no feedback.

- If the desired state of the motor is running (**.MSTRT**=true), the control signal (**.CMMD**) is set to true.
- If the desired state is stopped (**.MSTRT**=false), the **.CMMD** bit is set to false.

When you select MSN in the Device Definition Table, this form appears:



Name: unique name that identifies motor (12 characters maximum)

Type: MSN (3-letter code for device)

Description: 30 characters maximum (optional)

Start/Stop command: symbolic name of discrete output, boolean variable, or *device_name*.**CMMD** that starts and stops motor.

Start/Stop alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow motor to change states.

Table 7-10 lists the extensions and commands used with an MSN device.

Table 7-10 MSN Extensions and Commands

Extension		Commands	
Read-only Boolean		LOCK	place in auto mode
.CMMD [1]	start/stop command	UNLOCK	place in manual mode
.RUNNG	running	START	start motor
.STPPD	stopped	STOP	stop motor
.TRVL	traveling		
Read/Write Boolean			
.DSBLD	forced to manual mode		
.LOCKD	locked (auto mode)		
.NRDY	not ready		
.MSTRT	manual start		
.STATUS	device status		
Read-only Integer			
.VFLAGS	device status (505 only)		
Read/Write Integer			
.RTCP	run timer/counter preset		
.RTCC	run timer/counter current		
.STCP	stop timer/counter preset		
.STCC	stop timer/counter current		

1 *device_name*.**CMMD** is interchangeable with symbolic name configured as Start/Stop command.

Figure 7-10 shows the operation bits (across the top) affect the status bits (across the bottom) and/or the output bits (on the right).

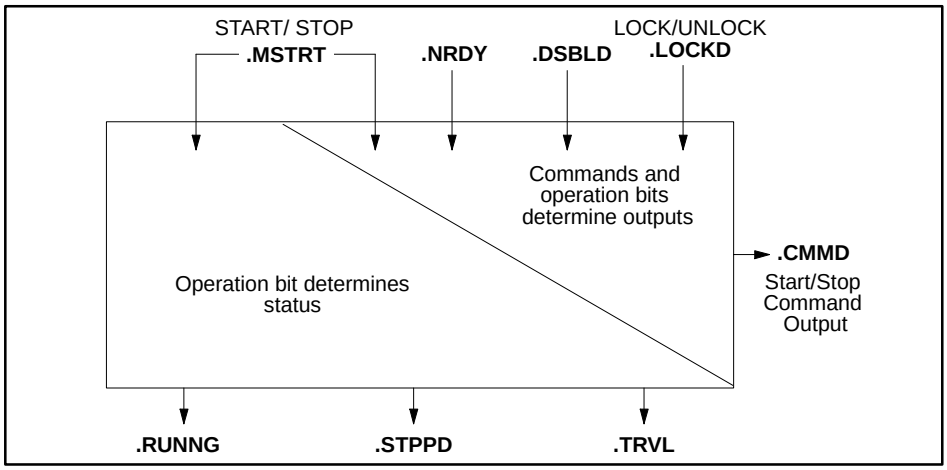


Figure 7-10 MSN Extensions and Commands

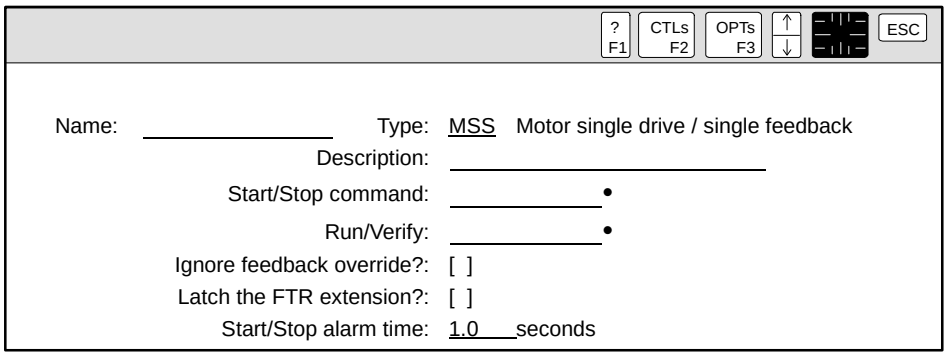
Motor (continued)

Single-Drive/ Single-Feedback Motor (MSS)

The MSS device is a two-state motor (running and stopped) and is controlled by a single discrete signal with a single discrete feedback signal.

- If the desired state of the motor is running (**.MSTRT**=true), the control signal (**.CMMD**) is set to true.
- If the desired state of the motor is stopped (**.MSTRT**=false), the control signal (**.CMMD**) is set to false.
- The feedback signal (**.RUNIO**) should be true when the motor is running and false when the motor is stopped.
- If the **Latch the FTR extension** option is selected, the **.CMMD** bit will change to false when the **.FTR** bit becomes true. The **.CMMD** bit remains false until a RESET command is issued.
- The RESET command issues a START/STOP command that turns on the **.TRVL** bit. The START/STOP alarm timer starts counting down when the RESET bit goes false.

When you select MSS in the Device Definition Table, this form appears:



Name: unique name that identifies motor (12 characters maximum)

Type: MSS (3-letter code for device)

Description: 30 characters maximum (optional)

Start/Stop command: symbolic name of discrete output, boolean variable, or *device_name***.CMMD** that starts and stops motor.

Run/Verify: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name***.RUNIO** that is feedback signal.

Ignore feedback override: Override feedback bit is ignored.

Latch the FTR extension: the **.CMMD** bit becomes false when the **.FTR** bit becomes true, and the **.RUNNG** bit has the **.FTR** bit latched.

Start/Stop alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow motor to change states.

Table 7-11 lists the extensions and commands used with an MSS device.

Table 7-11 MSS Extensions and Commands

Extension		Commands	
Read-only Boolean		LOCK	place in auto mode
.CMMD [1]	start/stop command	UNLOCK	place in manual mode
.RUNNG	running	START	start motor
.STPPD	stopped	STOP	stop motor
.TRVL	traveling	RESET	clear feedback override, and/or issue start/stop command after **.FTR** or **.FTS** is true.
.RUNIO [2]	run feedback		
.FTR	fail to run		
.FTS	fail to stop		
Read/Write Boolean			
.DSBLD	forced to manual mode		
.LOCKD	locked (auto mode)		
.NRDY	not ready		
.MSTRT	manual start		
.OVRD	override feedback		
.STATUS	device status		
Read-only Integer			
.VFLAGS	device status (505 only)		
Read/Write Integer			
.RTCP	run timer/counter preset		
.RTCC	run timer/counter current		
.STCP	stop timer/counter preset		
.STCC	stop timer/counter current		

1 *device_name*.**CMMD** is interchangeable with symbolic name configured as Start/Stop command.
2 *device_name*.**RUNIO** is interchangeable with symbolic name configured as Run/Verify feedback signal. If feedback is not a DI, this is read/write.

Figure 7-11 shows how the input bit (left) and the operation bits (top) affect the status bits (bottom) and/or the output bits (right).

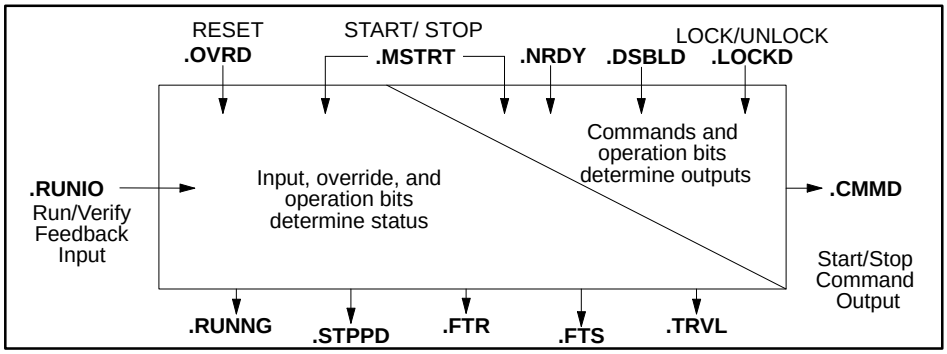


Figure 7-11 MSS Extensions and Commands

Motors (continued)

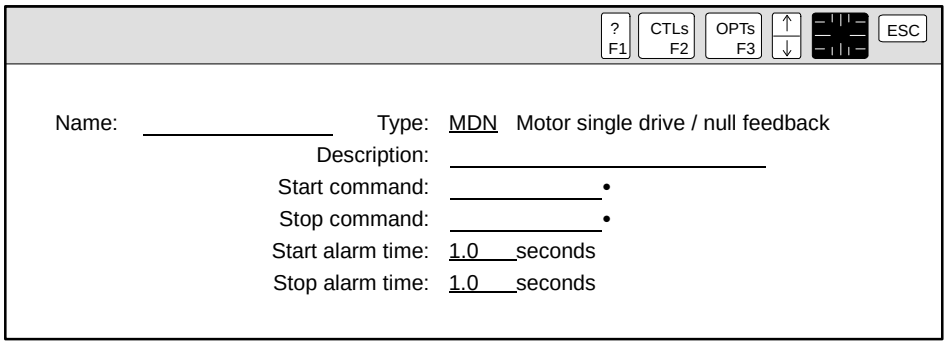
Dual-Drive/ Null-Feedback Motor (MDN)

The MDN device is a two-state motor (running and stopped) and is controlled by two discrete signals with no feedback.

The two discrete control signals consist of a start signal (**.STRTC**) and a stop signal (**.STOPC**).

- If the desired state of the motor is running (**.MSTRT**=true), the **.STRTC** bit is set to true to start the motor. The **.STRTC** bit remains true until the start alarm time expires; then **.STRTC** is set to false.
- If the desired state of the motor is stopped (**.MSTRT**=false), the **.STOPC** is set to true to stop the motor. The **.STOPC** bit remains true until the stop alarm time expires; then **.STOPC** is set to false.
- The RESET command issues a START/STOP command that turns on the **.TRVL** bit. The START/STOP alarm timer starts counting down when the RESET bit goes false.

When you select MDN in the Device Definition Table, this form appears:



Name: unique name that identifies motor (12 characters maximum)

Type: MDN (3-letter code for device)

Description: 30 characters maximum (optional)

Start command: symbolic name of discrete output, boolean variable, or *device_name*.**STRTC** that starts motor.

Stop command: symbolic name of discrete output, boolean variable, or *device_name*.**STOPC** that stops motor.

Start alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow motor to change from stopped to running.

Stop alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow motor to change from running to stopped.

Table 7-12 lists the extensions and commands used with an MDN device.

Table 7-12 MDN Extensions and Commands

Extension		Commands	
Read-only Boolean		LOCK	place in auto mode
.CMMD	start/stop command	UNLOCK	place in manual mode
.STRTC [1]	start command	START	start motor
.STOPC [2]	stop command	STOP	stop motor
.RUNNG	running		
.STPPD	stopped		
.TRVL	traveling		
Read/Write Boolean			
.DSBLD	forced to manual mode		
.LOCKD	locked (auto mode)		
.NRDY	not ready		
.MSTRT	manual start		
.STATUS	device status		
Read-only Integer			
.VFLAGS	device status (505 only)		
Read/Write Integer			
.RTCP	run timer/counter preset		
.RTCC	run timer/counter current		
.STCP	stop timer/counter preset		
.STCC	stop timer/counter current		

1 *device_name*.STRTC is interchangeable with symbolic name configured as Start command.
2 *device_name*.STOPC is interchangeable with symbolic name configured as Stop command.

Figure 7-12 shows how the operation bits (across the top) affect the status bits (across the bottom) and/or the output bits (on the right).

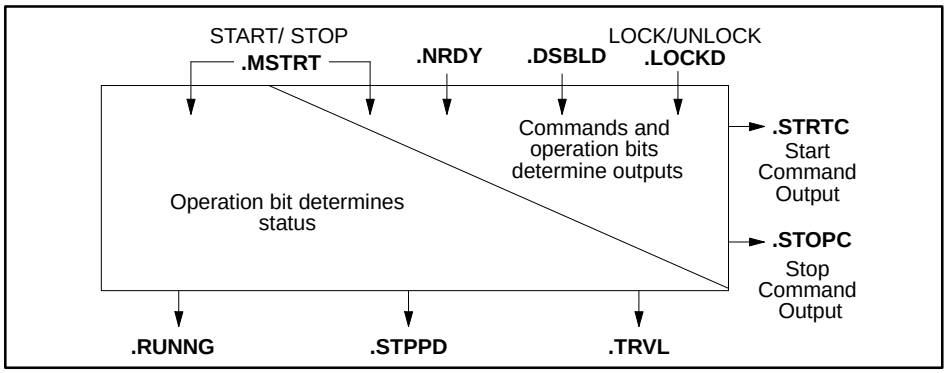


Figure 7-12 MDN Extensions and Commands

Motor (continued)

Dual-Drive/ Single-Feedback Motor (MDS)

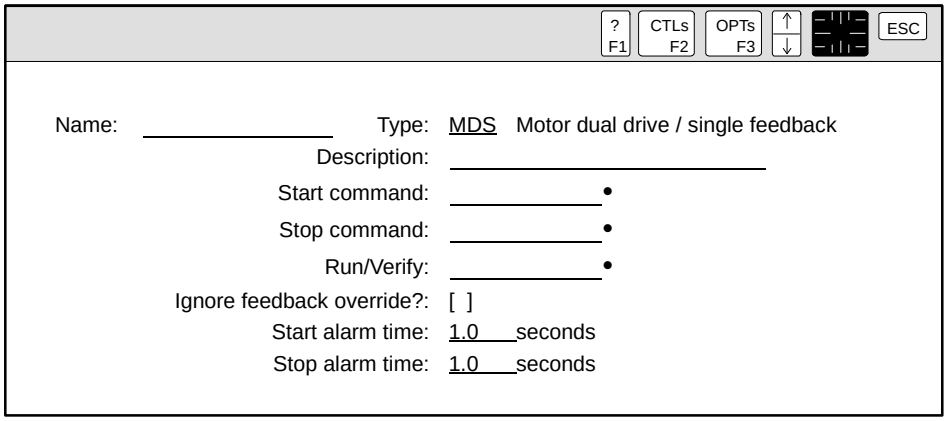
The MDS device is a two state motor (running and stopped) and is controlled by two discrete signals with a single discrete feedback signal.

The two discrete control signals consist of a start signal (.STRTC) and a stop signal (.STOPC).

- If the desired state of the motor is running (.MSTRT=true), the .STRTC bit is set to true to start the motor. The .STRTC bit remains true until either .RUNNG=true or the start alarm time expires; then .STRTC is set to false.
- If the desired state of the motor is stopped (.MSTRT=false), the .STOPC bit is set to true to stop the motor. The .STOPC bit remains true until either .STPPD=true or the stop alarm time expires; then .STOPC is set to false.
- It is not necessary to keep either of the control signals on once the motor is in the desired state.
- The RESET command issues a START/STOP command that turns on the .TRVL bit. The START/STOP alarm timer starts counting down when the RESET bit goes false.

The feedback signal (.RUNIO) should be true when the motor is running and false when the motor is stopped.

When you select MDS in the Device Definition Table, this form appears:



Name: unique name that identifies motor (12 characters maximum)

Type: MDS (3-letter code for device)

Description: 30 characters maximum (optional)

Start command: symbolic name of discrete output, boolean variable, or *device_name*.STRTC that starts motor.

Stop command: symbolic name of discrete output, boolean variable, or *device_name*.STOPC that stops motor.

Run/Verify: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.RUNIO that is feedback signal.

Ignore feedback override: Override feedback bit is ignored.

Start alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow motor to change from stopped to running; also indicates maximum time to keep start signal true.

Stop alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow motor to change from running to stopped; also indicates maximum time to keep stop signal true.

Motor (continued)

Table 7-13 lists the extensions and commands used with an MDS device.

Table 7-13 MDS Extensions and Commands

Extension		Commands	
Read-only Boolean		LOCK	place in auto mode
.CMMD	start/stop command	UNLOCK	place in manual mode
.STRTC [1]	start command	START	start motor
.STOPC [2]	stop command	STOP	stop motor
.RUNNG	running	RESET	clear feedback override
.STPPD	stopped		
.TRVL	traveling		
.RUNIO [3]	run feedback		
.FTR	fail to run		
.FTS	fail to stop		
.STPTO	stop timeout		
.STRTO	start timeout		
Read/Write Boolean			
.DSBLD	forced to manual mode		
.LOCKD	locked (auto mode)		
.NRDY	not ready		
.MSTRT	manual start		
.OVRD	override feedback		
.STATUS	device status (505 only)		
Read-only Integer			
.VFLAGS	device status		
Read/Write Integer			
.RTCP	run timer/counter preset		
.RTCC	run timer/counter current		
.STCP	stop timer/counter preset		
.STCC	stop timer/counter current		

1 *device_name*.STRTC is interchangeable with symbolic name configured as Start command.
2 *device_name*.STOPC is interchangeable with symbolic name configured as Stop command.
3 *device_name*.RUNIO is interchangeable with symbolic name configured as Run/Verify feedback signal. If feedback is not a DI, this is read/write.

Figure 7-13 shows how the input bit (on the left) and the operation bits (across the top) affect the status bits (across the bottom) and/or the output bits (on the right).

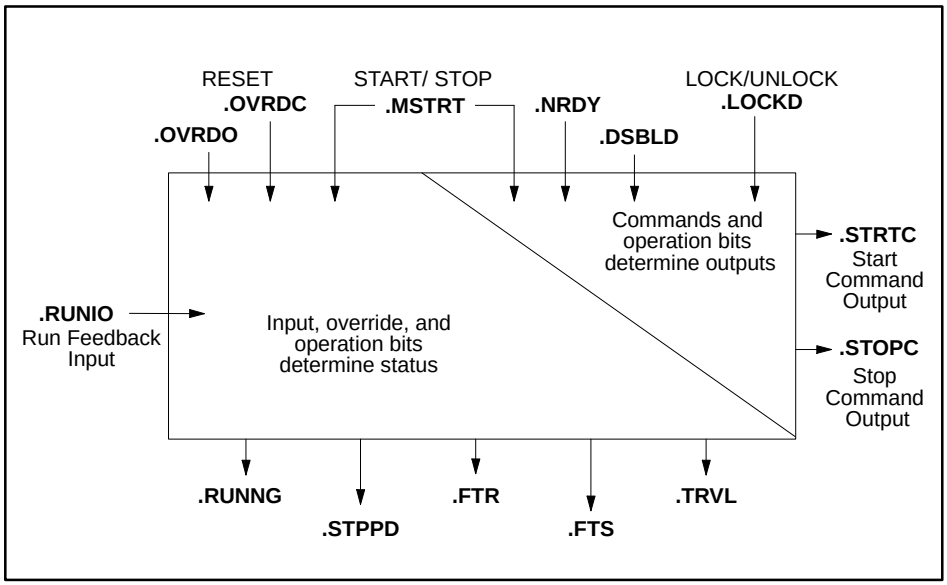


Figure 7-13 MDS Extensions and Commands

Motor (continued)

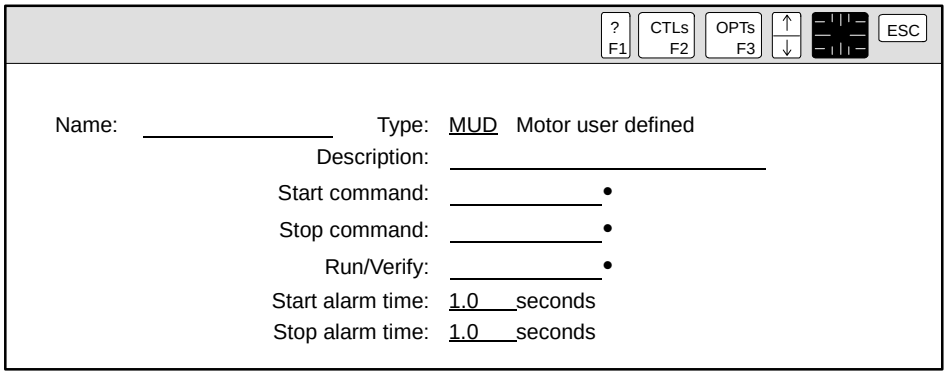
User-defined Motor (MUD)

The MUD device is a two-state motor (running and stopped) and is controlled by two discrete signals with a single discrete feedback signal.

The MUD is essentially the shell of a MDS device. The code which defines the MDS has been reduced to a minimum, and extensions have been provided to allow you to create your own customized motor and translate it for OSx (PCS).

The MUD has two alarm timers. When the .SRESET (.RRESET) bit transitions from false to true, the alarm timer starts. When the timer times out, the .STRTO (.STPTO) bit becomes true, and remains true until the .SRESET (.RRESET) bit becomes false.

When you select MUD in the Device Definition Table, this form appears:



Name: unique name that identifies motor (12 characters maximum)

Type: MUD (3-letter code for device)

Description: 30 characters maximum (optional)

Start command: symbolic name of discrete output, boolean variable, or *device_name*.STRTC that starts motor.

Stop command: symbolic name of discrete output, boolean variable, or *device_name*.STOPC that stops motor.

Run/Verify: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.RUNIO that is feedback signal.

Start alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow motor to change from stopped to running; also indicates maximum time to keep start signal true.

Stop alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow motor to change from running to stopped; also indicates maximum time to keep stop signal true.

Motors (continued)

Table 7-14 lists the extensions and commands used with an MUD device.

Table 7-14 MUD Extensions and Commands

Extension		Commands	
Read-only Boolean		LOCK	place in auto mode
.CMMD	start/stop command	UNLOCK	place in manual mode
.STRTC [1]	start command	START	start motor
.STOPC [2]	stop command	STOP	stop motor
.RUNNG	running	RESET	clear feedback override
.STPPD	stopped		
.TRVL	traveling		
.RUNIO [3]	run feedback		
.FTR	fail to run		
.FTS	fail to stop		
.STRTO	start alarm timer time out		
.STPTO	stop alarm timer time out		
Read/Write Boolean			
.DSBLD	forced to manual mode		
.LOCKD	locked (auto mode)		
.NRDY	not ready		
.MSTRT	manual start		
.OVRD	override feedback		
.RRESET	start alarm timer reset		
.SRESET	stop alarm timer reset		
.STATUS	device status		
Read-only Integer			
.VFLAGS	device status (505 only)		
Read/Write Integer			
.RTCP	run timer/counter preset		
.RTCC	run timer/counter current		
.STCP	stop timer/counter preset		
.STCC	stop timer/counter current		

1 *device_name*.STRTC is interchangeable with symbolic name configured as Start command.
2 *device_name*.STOPC is interchangeable with symbolic name configured as Stop command.
3 *device_name*.RUNIO is interchangeable with symbolic name configured as Run/Verify feedback signal. If feedback is not a DI, this is read/write.

Figure 7-14 shows how the input bit (on the left) and the operation bits (across the top) affect the status bits (across the bottom) and/or the output bits (on the right).

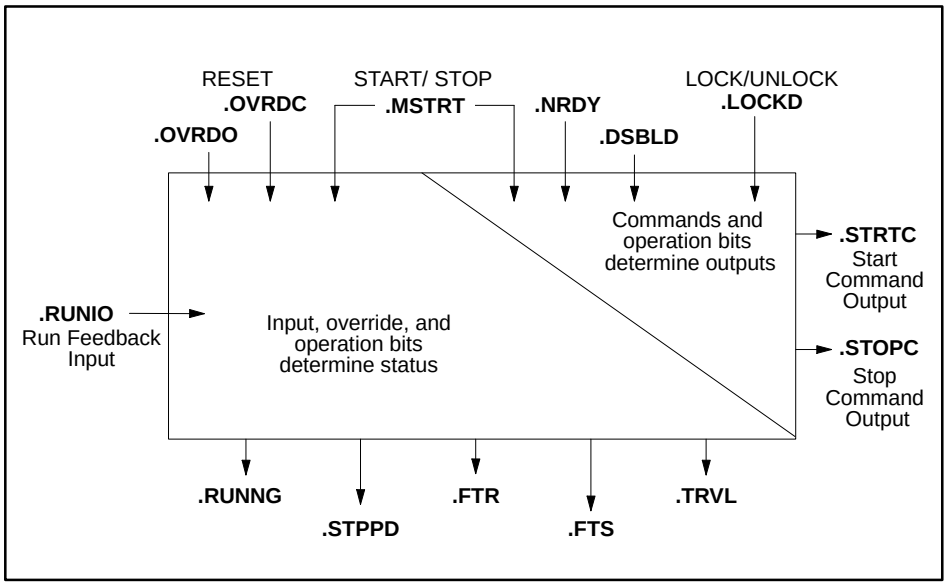


Figure 7-14 MUD Extensions and Commands

7.4 Reversible Motors with Dual Feedback

Reversible Motor/ Type 1 (RM1)

The RM1 device is a three-state motor (forward, reverse, and stopped) and is controlled by two discrete signals with two discrete feedback signals.

The two discrete control signals consist of a forward signal (.SFWRD) and a reverse signal (.SREV). The two signals cannot both be true at the same time.

- If the desired state of the motor is running forward (.MSTRT=true and .MREV=false), the .SFWRD bit is set to true (energized) to start the motor in a forward direction.
- If the desired state of the motor is running reverse (.MSTRT=true and .MREV=true), the .SREV bit is set to true (energized) to start the motor in a reverse direction.
- If the desired state of the motor is stopped (.MSTRT=false), the currently energized signal (.SFRWD or .SREV) is set to false.
- The motor, when receiving a start signal, starts running and keeps running until the currently energized signal becomes false.
- If you request a change in direction while the motor is running, the .DRV bit is set to false, and the motor must stop before the direction is changed. After the motor stops, the .DRV bit is set to true to start the motor in the new direction.
- The RESET command issues a START/STOP command that turns on the .TRVL bit. The START/STOP alarm timer starts counting down when the RESET bit goes false.

The two feedback signals consist of a running-forward feedback signal (.FIO) and a running-reverse feedback signal (.RIO).

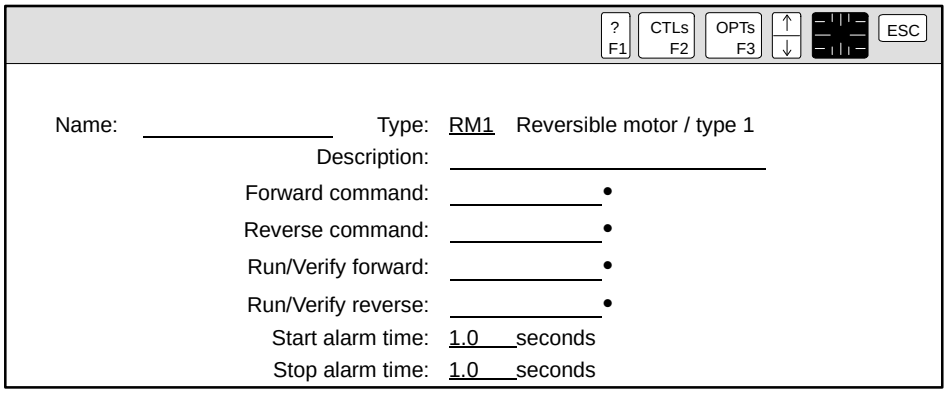
- The .FIO bit should be true when the motor is running in a forward direction; otherwise it should be false.
- The .RIO bit should be true when the motor is running in a reverse direction; otherwise it should be false.

In manual mode, the forward and reverse signals (.SFWRD/.SREV) are set to the appropriate state based on the status of the .MREV and the .MSTRT bits, which you manipulate from an operator station or from the program.

In auto mode, the STARTF, STARTR, and STOP commands set the state of the output bits. In this mode, the .MREV and .MSTRT bits are set to reflect the last requested state. This is done to provide for a bumpless transfer if the device changes modes.

The logic for the reversible motor is shown in Appendix A, "Device RLL Code."

When you select RM1 in the Device Definition Table, this form appears:



Name: unique name that identifies motor (12 characters maximum)

Type: RM1 (3-letter code for device)

Description: 30 characters maximum (optional)

Forward command: symbolic name of discrete output, boolean variable, or *device_name*.SFWRD that starts motor in forward direction.

Reverse command: symbolic name of discrete output, boolean variable, or *device_name*.SREV that starts motor in reverse direction.

Run/Verify forward: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.FIO that is forward feedback signal.

Run/Verify reverse: symbolic name of discrete input, discrete output, boolean variable, APT flag variable, or *device_name*.RIO that is reverse feedback signal.

Start alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow motor to change from stopped to running.

Stop alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow motor to change from running to stopped.

Table 7-15 lists the extensions and commands used with an RM1 device.

Reversible Motors with Dual Feedback (continued)

Table 7-15 RM1 Extensions and Commands

Extension		Commands	
Read-only Boolean		LOCK	place in auto mode
.DRV	forward/reverse command	UNLOCK	place in manual mode
.SFWRD [1]	start forward	STARTF	start forward
.SREV [2]	start reverse	STARTR	start reverse
.RUNNG	running	STOP	stop motor
.RUNF	running forward	RESET	clear feedback overrides
.RUNR	running reverse		
.STPPD	stopped		
.TRVL	traveling		
.FIO [3]	forward feedback		
.RIO [4]	reverse feedback		
.FTRF	fail to run forward		
.FTRR	fail to run reverse		
.FTR [5]	fail to run		
.FTS	fail to stop		
.FAILD	failed (both feedback bits are true)		
.STRTO	start timeout		
.STPTO	stop timeout		
Read/Write Boolean			
.DSBLD	forced to manual mode		
.LOCKD	locked (auto mode)		
.NRDY	not ready		
.MSTRT	manual start		
.MREV	manual reverse		
.OVRDF	override forward feedback		
.OVRDR	override reverse feedback		
.STATUS	device status		
Read-only Integer			
.VFLAGS	device status (505 only)		
Read/Write Integer			
.RTCP	run timer/counter preset		
.RTCC	run timer/counter current		
.STCP	stop timer/counter preset		
.STCC	stop timer/counter current		

1 *device_name*.SFWRD is interchangeable with symbolic name configured as Forward command.
2 *device_name*.SREV is interchangeable with symbolic name configured as Reverse command.
3 *device_name*.FIO is interchangeable with symbolic name configured as Run/Verify forward feedback signal. If feedback is not a DI, this is read/write.
4 *device_name*.RIO is interchangeable with symbolic name configured as Run/Verify reverse feedback signal. If feedback is not a DI, this is read/write.
5 The .FTR bit is set to true when either the .FTRF or .FTRR bit is true.

Figure 7-15 shows how the input bits (on the left) and the operation bits (across the top) affect the status bits (across the bottom) and/or the output bits (on the right).

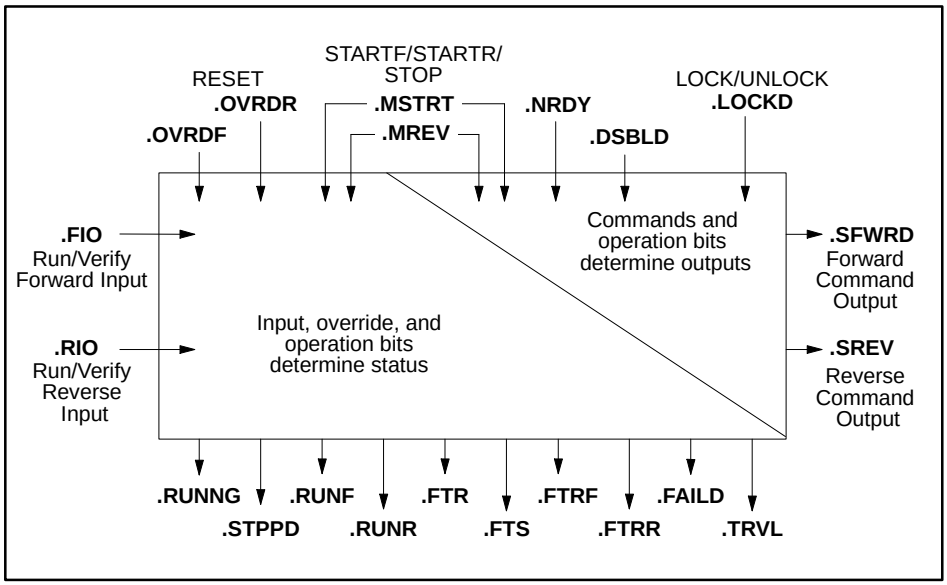


Figure 7-15 RM1 Extensions and Commands

Reversible Motors with Dual Feedback (continued)

Reversible Motor/ Type 2 (RM2)

The RM2 device is a three-state motor (forward, reverse, and stopped) and is controlled by two discrete signals with two discrete feedback signals.

The two control signals consist of a start/stop signal (.DRV) and a direction signal (.DIR) that determines whether the motor runs forward or reverse.

- If the desired state is running forward (.MSTRT=true and .MREV=false), the .DRV bit is set to true and the .DIR bit to false.
- If the desired state is running reverse (.MSTRT=true and .MREV=true), the .DRV and .DIR bits are both set to true.
- If the desired state is stopped (.MSTRT=false), the .DRV bit is set to false to stop the motor.
- The motor, when receiving a start signal, starts running and keeps running until .DRV becomes false or .DIR changes state.
- If you request a change in direction while the motor is running, the .DRV bit is set to false, and the motor must stop before the direction is changed. After the motor stops, .DRV is set to true to start the motor in the new direction.
- The RESET command issues a START/STOP command that turns on the .TRVL bit. The START/STOP alarm timer starts counting down when the RESET bit goes false.

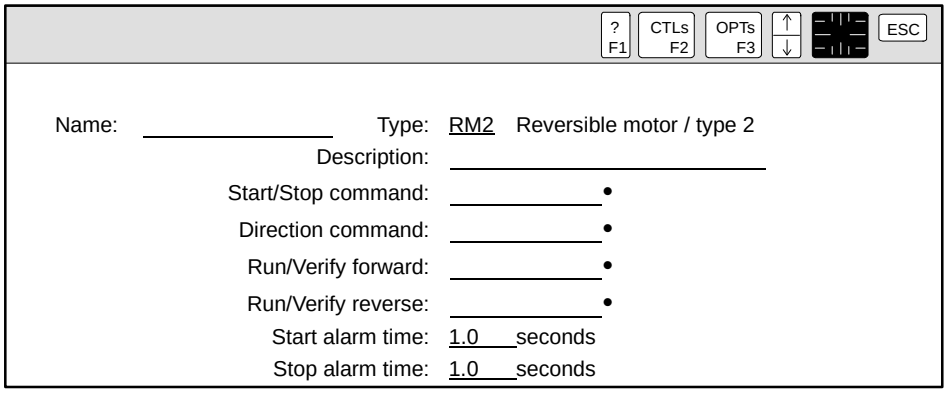
The two feedback signals consist of a running-forward feedback signal (.FIO) and a running-reverse feedback signal (.RIO).

- The .FIO bit should be true when the motor is running in a forward direction; otherwise, it should be false.
- The .RIO bit should be true when the motor is running in a reverse direction; otherwise, it should be false.

In manual mode, the control signals (.DRV/.DIR) are set to the appropriate state based on the status of the .MREV and the .MSTRT bits, which you manipulate from an operator station or from the program.

In auto mode, the STARTF, STARTR, and STOP commands set the state of the output bits. In this mode, the .MREV and .MSTRT bits are set to reflect the last requested state. This is done to provide for a bumpless transfer if the device changes modes.

When you select RM2 in the Device Definition Table, this form appears:



Name: unique name that identifies motor (12 characters maximum)

Type: RM2 (3-letter code for device)

Description: 30 characters maximum (optional)

Start/Stop command: symbolic name of discrete output, boolean variable, or *device_name*.DRV that starts and stops motor.

Direction command: symbolic name of discrete output, boolean variable, or *device_name*.DIR that is direction signal.

Run/Verify forward: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.FIO that is forward feedback signal.

Run/Verify reverse: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.RIO that is reverse feedback signal.

Start alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow motor to change from stopped to running.

Stop alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow motor to change from running to stopped.

Table 7-16 lists the extensions and commands used with an RM2 device.

Reversible Motors with Dual Feedback (continued)

Table 7-16 RM2 Extensions and Commands

Extension		Commands	
Read-only Boolean		LOCK	place in auto mode
.DRV [1]	start/stop command	UNLOCK	place in manual mode
.DIR [2]	direction	STARTF	start forward
.RUNNG	running	STARTR	start reverse
.RUNF	running forward	STOP	stop motor
.RUNR	running reverse	RESET	clear feedback overrides
.STPPD	stopped		
.TRVL	traveling		
.FIO [3]	forward feedback		
.RIO [4]	reverse feedback		
.FTRF	fail to run forward		
.FTRR	fail to run reverse		
.FTR [5]	fail to run		
.FTS	fail to stop		
.FAILD	failed (both feedback bits are true)		
.STRTO	start timeout		
.STPTO	stop timeout		
Read/Write Boolean			
.DSBLD	forced to manual mode		
.LOCKD	locked (auto mode)		
.NRDY	not ready		
.MSTRT	manual start		
.MREV	manual reverse		
.OVRDF	override forward feedback		
.OVRDR	override reverse feedback		
.STATUS	device status		
Read-only Integer			
.VFLAGS	device status (505 only)		
Read/Write Integer			
.RTCP	run timer/counter preset		
.RTCC	run timer/counter current		
.STCP	stop timer/counter preset		
.STCC	stop timer/counter current		

1 *device_name*.DRV is interchangeable with symbolic name configured as start/stop command.
2 *device_name*.DRV is interchangeable with symbolic name configured as direction command.
3 *device_name*.FIO is interchangeable with symbolic name configured as Run/Verify forward feedback signal. If feedback is not a DI, this is read/write.
4 *device_name*.RIO is interchangeable with symbolic name configured as Run/Verify reverse feedback signal. If feedback is not a DI, this is read/write.
5 The .FTR bit is set to true when either the .FTRF or .FTRR bit is true.

Figure 7-16 shows how the input bits (on the left) and the operation bits (across the top) affect the status bits (across the bottom) and/or the output bits (on the right).

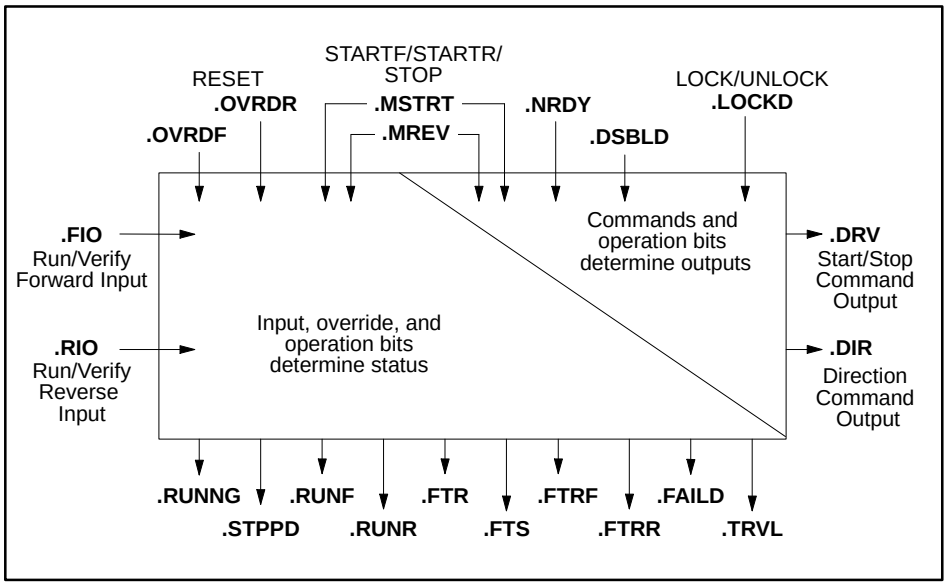


Figure 7-16 RM2 Extensions and Commands

7.5 Two-Speed Motors with Dual Feedback

Two-Speed Motor/ Type 1 (TS1)

The TS1 device is a three-state motor (high, low, and stopped) and is controlled by two discrete signals with two discrete feedback signals.

The two discrete control signals consist of a low signal (.SLOW) and a high signal (.SHIGH). The two signals cannot both be true at the same time.

- If the desired state of the motor is running at high speed (.MSTRT=true and .MHIGH=true), the .SHIGH bit is set to true (energized) to start the motor at high speed.
- If the desired state of the motor is running at low speed (.MSTRT=true and .MHIGH=false), the .SLOW bit is set to true (energized) to start the motor at slow speed.
- If the desired state is stopped (.MSTRT=false), the currently energized signal (.SHIGH or .SLOW) is set to false.
- The motor, when receiving a start signal, starts running and keeps running until the currently energized signal becomes false.
- You can change the speed of a two-speed motor without stopping it.
- The RESET command issues a START/STOP command that turns on the .TRVL bit. The START/STOP alarm timer starts counting down when the RESET bit goes false.

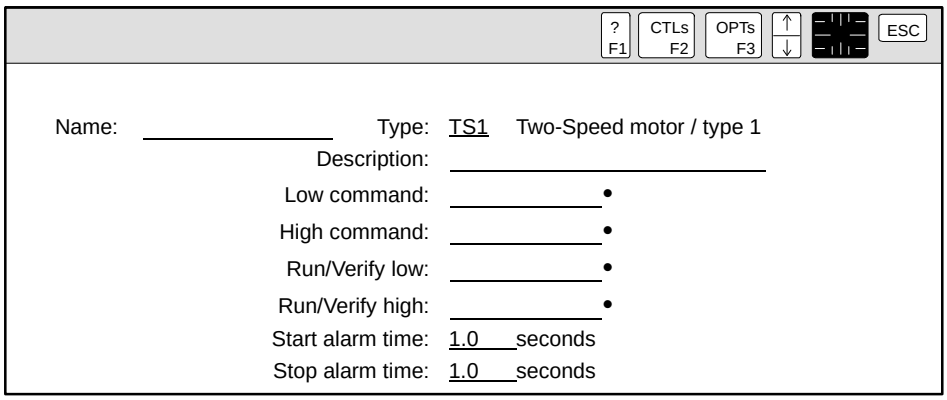
The two feedback signals consist of a running-low feedback signal (.LIO) and a running-high feedback signal (.HIO).

- The .LIO bit should be true when the motor is running at low speed; otherwise, it should be false.
- The .HIO bit should be true when the motor is running at high speed; otherwise, it should be false.

In manual mode, the outputs (.SLOW/.SHIGH) are set to the appropriate state based on the status of the .MHIGH and the .MSTRT bits, which you manipulate from an operator station or from the program.

In auto mode, the STARTL, STARTH, and STOP commands set the state of the output bits. In this mode, the .MHIGH and .MSTRT bits are set to reflect the last requested state. This is done to provide for a bumpless transfer if the device changes modes.

When you select TS1 in the Device Definition Table, this form appears:



Name: unique name that identifies motor (12 characters maximum)

Type: TS1 (3-letter code for device)

Description: 30 characters maximum (optional)

Low command: symbolic name of discrete output, boolean variable, or *device_name*.SLOW that starts motor at low speed.

High command: symbolic name of discrete output, boolean variable, or *device_name*.SLOW that starts motor at high speed.

Run/Verify low: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.LIO that is low feedback signal.

Run/Verify high: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.HIO that is high feedback signal.

Start alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow motor to change from stopped to running.

Stop alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow motor to change from running to stopped.

Table 7-17 lists the extensions and commands used with a TS1 device.

Two-Speed Motors with Dual Feedback (continued)

Table 7-17 TS1 Extensions and Commands

Extension		Commands	
Read-only Boolean		LOCK	place in auto mode
.SLOW [1]	start low	UNLOCK	place in manual mode
.SHIGH [2]	start high	STARTL	start low
.RUNNG	running	STARTH	start high
.RUNL	running low	STOP	stop motor
.RUNH	running high	RESET	clear feedback overrides
.STPPD	stopped		
.TRVL	traveling		
.LIO [3]	low feedback		
.HIO [4]	high feedback		
.FTRL	fail to run low		
.FTRH	fail to run high		
.FTR [5]	fail to run		
.FTS	fail to stop		
.FAILD	failed (both feedback bits are true)		
.STRTO	start timeout		
.STPTO	stop timeout		
Read/Write Boolean			
.DSBLD	forced to manual mode		
.LOCKD	locked (auto mode)		
.NRDY	not ready		
.MHIGH	manual high		
.MSTRT	manual start		
.OVRDL	override low feedback		
.OVRDH	override high feedback		
.STATUS	device status		
Read-only Integer			
.VFLAGS	device status (505 only)		
Read/Write Integer			
.RTCP	run timer/counter preset		
.RTCC	run timer/counter current		
.STCP	stop timer/counter preset		
.STCC	stop timer/counter current		

1 *device_name*.SLOW is interchangeable with symbolic name configured as low command.
2 *device_name*.SHIGH is interchangeable with symbolic name configured as high command.
3 *device_name*.LIO is interchangeable with symbolic name configured as Run/Verify low feedback signal. If feedback is not a DI, this is read/write.
4 *device_name*.HIO is interchangeable with symbolic name configured as Run/Verify high feedback signal. If feedback is not a DI, this is read/write.
5 The .FTR bit is set to true when either the .FTRL or .FTRH bit is true.

Figure 7-17 shows how the input bits (on the left) and the operation bits (across the top) affect the status bits (across the bottom) and/or the output bits (on the right).

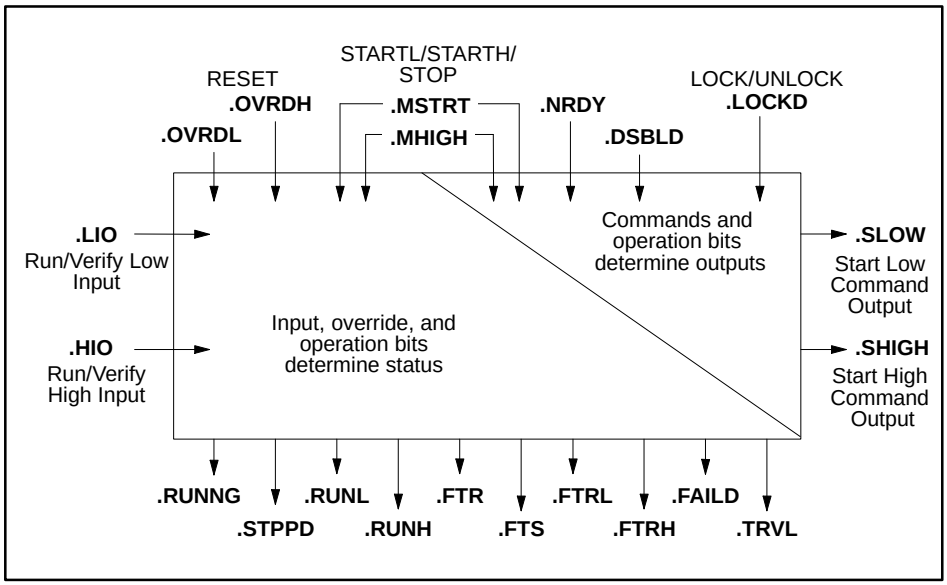


Figure 7-17 TS1 Extensions and Commands

Two-Speed Motors with Dual Feedback (continued)

Two-Speed Motor/ Type 2 (TS2)

The TS2 device is a three-state motor (high, low, and stopped) and is controlled by two discrete signals with two discrete feedback signals.

The two discrete control signals consist of a start/stop signal (.DRV) and a speed signal (.SPEED) that determines whether the motor runs at high or low speed.

- If the desired state of the motor is running high (.MSTRT=true and .MHIGH=true), the .DRV and .SPEED bits are both set to true.
- If the desired state is running low (.MSTRT=true and .MHIGH=false), the .DRV bit is set to true and the .SPEED bit is set to false.
- If the desired state is stopped (.MSTRT=false), the .DRV bit is set to false to stop the motor.
- The motor, when receiving a start signal, starts running and keeps running until .DRV becomes false.
- You can change the speed of a two-speed motor without stopping it.
- The RESET command issues a START/STOP command that turns on the .TRVL bit. The START/STOP alarm timer starts counting down when the RESET bit goes false.

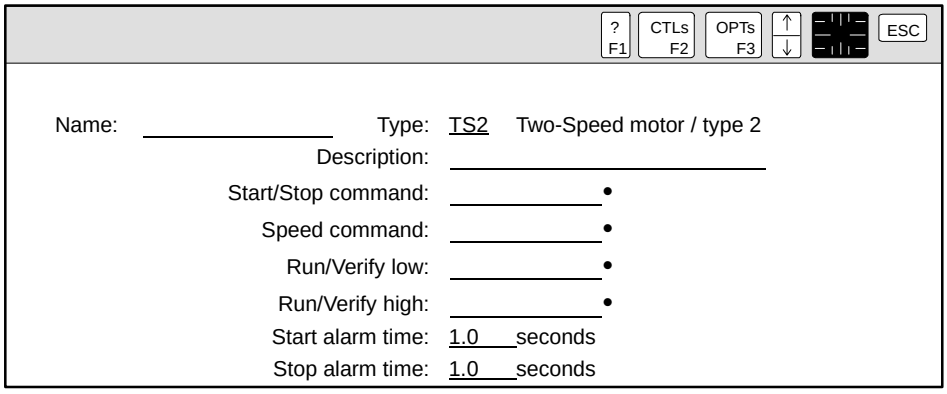
The two feedback signals consist of a running-low feedback signal (.LIO) and a running-high feedback signal (.HIO).

- The .LIO bit should be true when the motor is running at low speed; otherwise, it should be false.
- The .HIO should be true when the motor is running at high speed; otherwise it should be false.

In manual mode, the outputs (.DRV/.SPEED) are set to the appropriate state based on the status of the .MHIGH and the .MSTRT bits, which you manipulate from an operator station or from the program.

In auto mode, the STARTL, STARTH, and STOP commands set the state of the output bits. In this mode, the .MHIGH and .MSTRT bits are set to reflect the last requested state. This is done to provide for a bumpless transfer if the device changes modes.

When you select TS2 in the Device Definition Table, this form appears:



Name: unique name that identifies motor (12 characters maximum)

Type: TS2 (3-letter code for device)

Description: 30 characters maximum (optional)

Start/Stop command: symbolic name of discrete output, boolean variable, or *device_name*.DRV that starts and stops motor.

Speed command: symbolic name of discrete output, boolean variable, or *device_name*.SPEED that indicates the speed signal.

Run/Verify low: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.LIO that is low feedback signal.

Run/Verify high: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.HIO that is high feedback signal.

Start alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow motor to change from stopped to running.

Stop alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow motor to change from running to stopped.

Table 7-18 lists the extensions and commands used with a TS2 device.

Two-Speed Motors with Dual Feedback (continued)

Table 7-18 TS2 Extensions and Commands

Extension		Commands	
Read-only Boolean		LOCK	place in auto mode
.DRV [1]	start/stop command	UNLOCK	place in manual mode
.SPEED [2]	speed	STARTL	start low
.RUNNG	running	STARTH	start high
.RUNL	running low	STOP	stop motor
.RUNH	running high	RESET	clear feedback overrides
.STPPD	stopped		
.TRVL	traveling		
.LIO [3]	low feedback		
.HIO [4]	high feedback		
.FTRL	fail to run low		
.FTRH	fail to run high		
.FTR [5]	fail to run		
.FTS	fail to stop		
.FAILD	failed (both feedback bits are true)		
.STRTO	start timeout		
.STPTO	stop timeout		
Read/Write Boolean			
.DSBLD	forced to manual mode		
.LOCKD	locked (auto mode)		
.NRDY	not ready		
.MHIGH	manual high		
.MSTRT	manual start		
.OVRDL	override low feedback		
.OVRDH	override high feedback		
.STATUS	device status		
Read-only Integer			
.VFLAGS	device status (505 only)		
Read/Write Integer			
.RTCP	run timer/counter preset		
.RTCC	run timer/counter current		
.STCP	stop timer/counter preset		
.STCC	stop timer/counter current		

1 *device_name*.DRV is interchangeable with symbolic name configured as start/stop command.
2 *device_name*.SPEED is interchangeable with symbolic name configured as speed command.
3 *device_name*.LIO is interchangeable with symbolic name configured as Run/Verify low feedback signal.
4 *device_name*.HIO is interchangeable with symbolic name configured as Run/Verify high feedback signal.
5 The .FTR bit is set to true when either the .FTRL or .FTRH bit is true.

Figure 7-18 shows how the input bits (on the left) and the operation bits (across the top) affect the status bits (across the bottom) and/or the output bits (on the right).

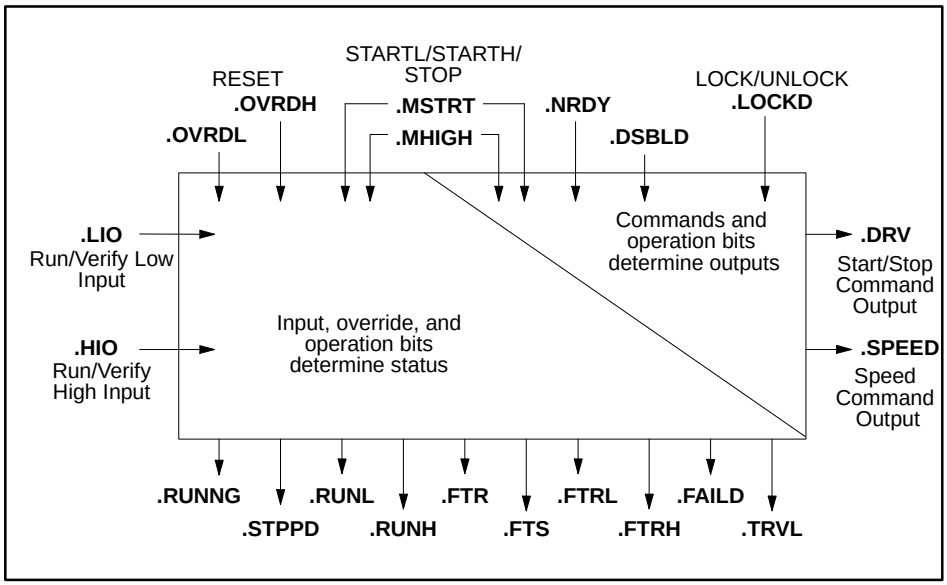


Figure 7-18 TS2 Extensions and Commands

7.6 Cylinder

Single-Drive/ Dual-Feedback Cylinder (CSD)

The CSD device has two positions (extended and retracted) and is controlled by a single discrete signal with two discrete feedback signals. Two types of CSD devices are supported: energize-extend (Type E) and energize-retract (Type R).

One control signal (.CMMD) determines the state of the cylinder.

- If the desired state of an energize-extend cylinder is extended (.MEXTEND=true), the control signal (.CMMD) is set to true.

  If the desired state is retracted, (.MEXTEND=false), the .CMMD bit is set to false.

- If the desired state of an energize-retract cylinder is extended (.MEXTEND=true), the control signal (.CMMD) is set to false.

  If the desired state is retracted, (.MEXTEND=false), the .CMMD bit is set to true.

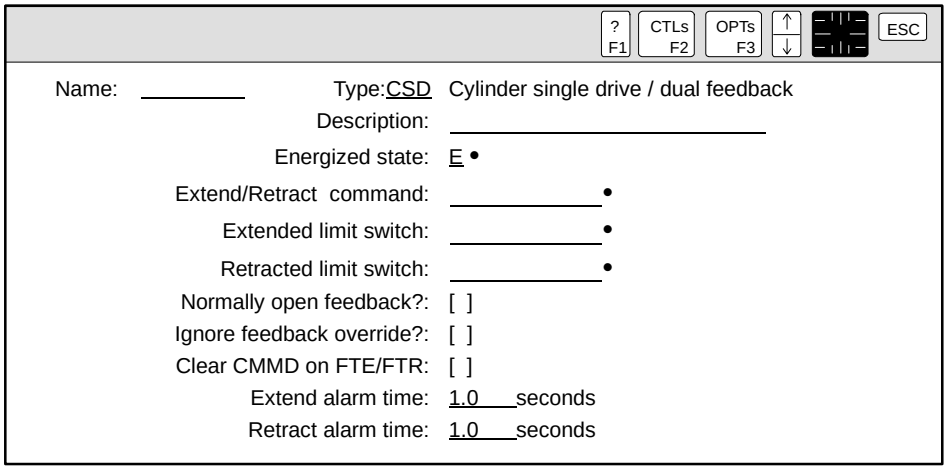
- The RESET command issues an EXTEND/RETRACT command that turns on the .TRVL bit. The EXTEND/RETRACT alarm timer starts counting down when the RESET bit goes false.

For both types of cylinders, the two feedback signals consist of an extended feedback signal (.ELS) and a retracted feedback signal (.RLS).

- The .ELS bit should be true when the piston is extended, and false when it is retracted.

- The .RLS bit should be true when the piston is retracted and false when it is extended.

When you select CSD in the Device Definition Table, this form appears:



Name: unique name that identifies cylinder (12 characters maximum)

Type: CSD (3-letter code for device)

Description: 30 characters maximum (optional)

Energized state: E indicates energize extend; R indicates energize retract; default is E.

Extend/Retract command: symbolic name of discrete output, boolean variable, or *device_name*.CMMD that extends and retracts the piston.

Extended limit switch: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.ELS that is extend feedback signal..

Retracted limit switch: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.RLS that is retract feedback signal.

Normally open feedback: feedback response is inverted, e.g, extend feedback is false when the piston is extended.

Ignore feedback override: Override feedback bit is ignored.

Clear CMMD on FTE/FTR: the CMMD bit becomes false when the FTE or the FTR bit becomes true.

Extend alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow cylinder to extend.

Retract alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow cylinder to retract.

Cylinder (continued)

Table 7-19 lists the extensions and commands used with a CSD device.

Table 7-19 CSD Extensions and Commands

Extension		Commands	
Read-only Boolean		LOCK	place in auto mode
.CMMD [1]	extend/retract command	UNLOCK	place in manual mode
.EXTENDED	extended	EXTEND	extend piston
.RETRACTED	retracted	RETRACT	retract piston
.TRVL	traveling	RESET	clear feedback override
.ELS [2]	extended feedback (E)		
.RLS [3]	retracted feedback (R)		
.FTE	fail to extend		
.FTR	fail to retract		
.FAILD	failed (both feedback bits are true)		
Read/Write Boolean			
.DSBLD	forced to manual mode		
.LOCKD	locked (auto mode)		
.NRDY	not ready		
.MEXTEND	manual extend		
.OVRDE	override extend feedback		
.OVRDR	override retract feedback		
.STATUS	device status		
Read-only Integer			
.VFLAGS	device status (505 only)		
Read/Write Integer			
.ETCP	extend timer/counter preset		
.ETCC	extend timer/counter current		
.RTCP	retract timer/counter preset		
.RTCC	retract timer/counter current		

1 *device_name*.CMMD is interchangeable with symbolic name configured as Extend/Retract command.
2 *device_name*.ELS is interchangeable with symbolic name configured as Extended Limit Switch for energize-extend cylinders. If feedback is not a DI, this is read/write.
3 *device_name*.RLS is interchangeable with symbolic name configured as Retracted Limit Switch for energize-retract cylinders. If feedback is not a DI, this is read/write.

Figure 7-19 shows how the input bits (on the left) and the operation bits (across the top) affect the status bits (across the bottom) and/or the output bits (on the right).

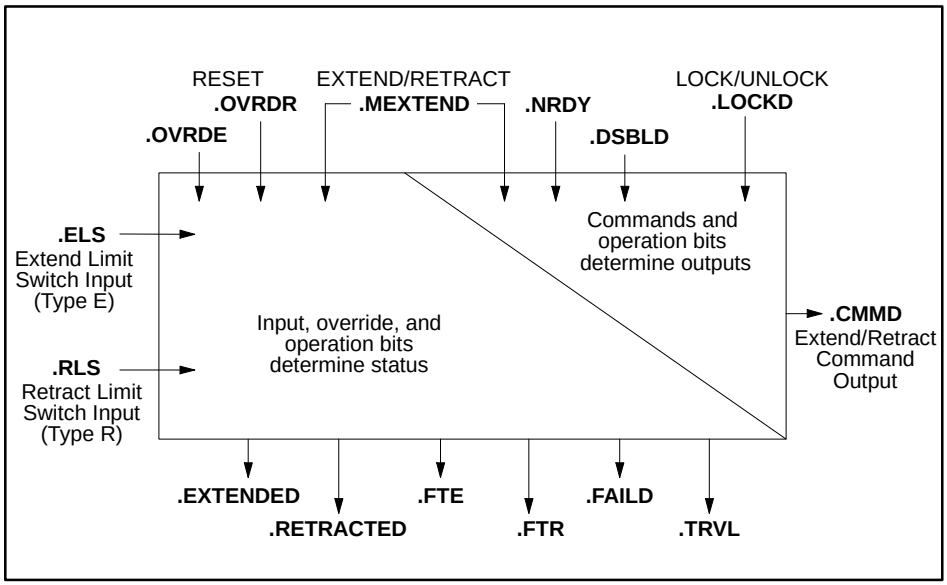


Figure 7-19 CSD Extensions and Commands

Cylinder (continued)

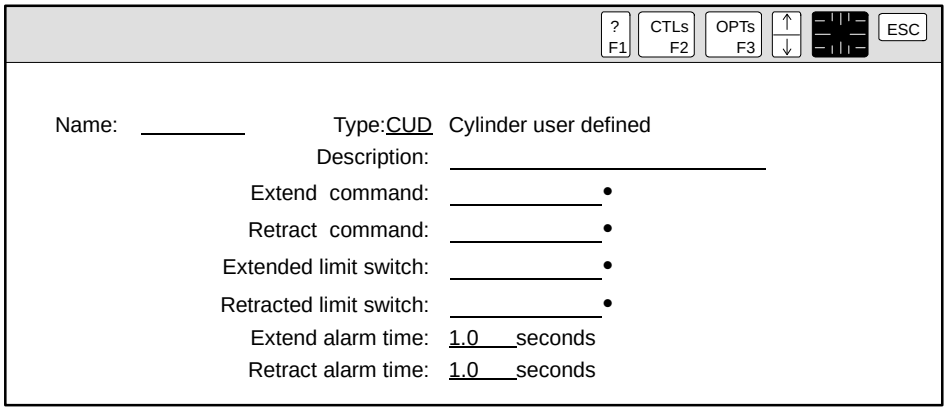
User-defined Cylinder (CUD)

The CUD device has two positions (open and closed) and is controlled by two discrete signals with two discrete feedback signals.

The CUD is essentially the shell of a CSD device. The code that defines the CSD has been reduced to a minimum, and extensions are provided to allow you to create your own customized cylinder and translate it for OSx (PCS).

The CUD has two alarm timers. When the .ERESET (.RRESET) bit transitions from false to true, the alarm timer starts. When the timer times out, the .EXTTO (.RETTO) bit becomes true, and remains true until the .ERESET (.RRESET) bit becomes false.

When you select CUD in the Device Definition Table, this form appears:



Name: unique name that identifies cylinder (12 characters maximum)

Type: CUD (3-letter code for device)

Description: 30 characters maximum (optional)

Extend command: symbolic name of discrete output, boolean variable, or *device_name*.EXTC that extends the piston.

Retract command: symbolic name of discrete output, boolean variable, or *device_name*.RETC that retracts the piston.

Extended limit switch: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.ELS that is extend feedback signal..

Retracted limit switch: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.RLS that is retract feedback signal.

Extend alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow the piston to be extended.

Retract alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow the piston to be retracted.

Table 7-20 lists the extensions and commands used with a CUD device.

Cylinder (continued)

Table 7-20 CUD Extensions and Commands

Extension		Commands	
Read-only Boolean		LOCK	place in auto mode
.CMMD	extend/retract command	UNLOCK	place in manual mode
.EXTC [1]	extend command	EXTEND	extend piston
.RETC [2]	retract command	RETRACT	retract piston
.EXTENDED	extended	RESET	clear feedback override
.RETRACTED	retracted		
.TRVL	traveling		
.ELS [3]	extended feedback		
.RLS [4]	retracted feedback		
.FTE	fail to extend		
.FTR	fail to retract		
.FAILD	failed (both feedback bits are true)		
.EXTTO	extend alarm timer time out		
.RETTO	retract alarm timer time out		
Read/Write Boolean			
.DSBLD	forced to manual mode		
.LOCKD	locked (auto mode)		
.NRDY	not ready		
.MEXTEND	manual extend		
.OVRDE	override extend feedback		
.OVRDR	override retract feedback		
.ERESET	extend alarm timer reset		
.RRESET	retract alarm timer reset		
.STATUS	device status		
Read-only Integer			
.VFLAGS	device status (505 only)		
Read/Write Integer			
.ETCP	extend timer/counter preset		
.ETCC	extend timer/counter current		
.RTCP	retract timer/counter preset		
.RTCC	retract timer/counter current		

1 *device_name*.EXTC is interchangeable with symbolic name configured as Extend command.
2 *device_name*.RETC is interchangeable with symbolic name configured as Retract command.
3 *device_name*.ELS is interchangeable with symbolic name configured as Open Limit Switch for energize-open valves. If feedback is not DI, this is read/write.
4 *device_name*.RLS is interchangeable with symbolic name configured as Close Limit Switch for energize-closed valves. If feedback is not DI, this is read/write.

Figure 7-20 shows how the input bits (on the left) and the operation bits (across the top) affect the status bits (across the bottom) and/or the output bits (on the right).

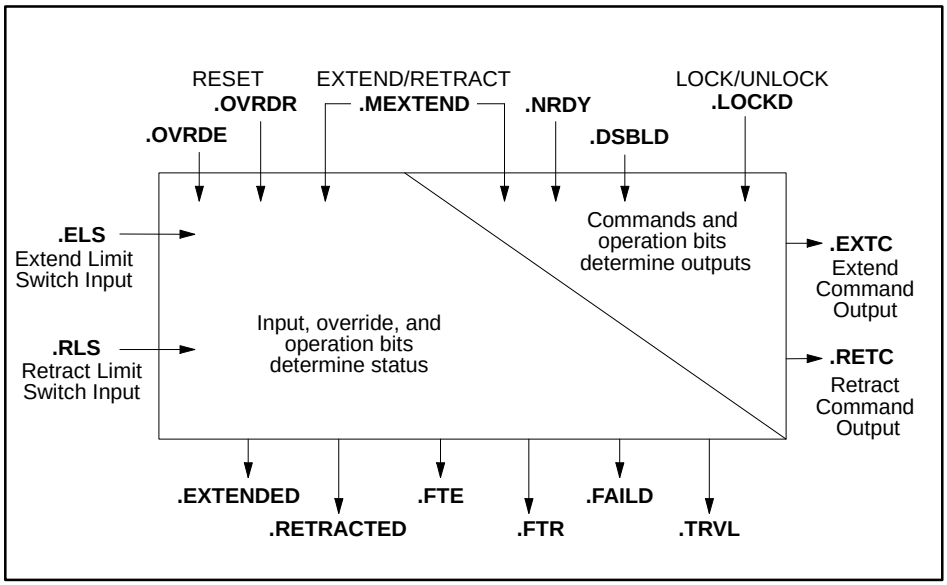


Figure 7-20 CUD Extensions and Commands

7.7 Presses

Hand-Operated/ Dual-Feedback Press (PND)

The PND device has two positions (open and closed) and is controlled by two discrete feedback signals.

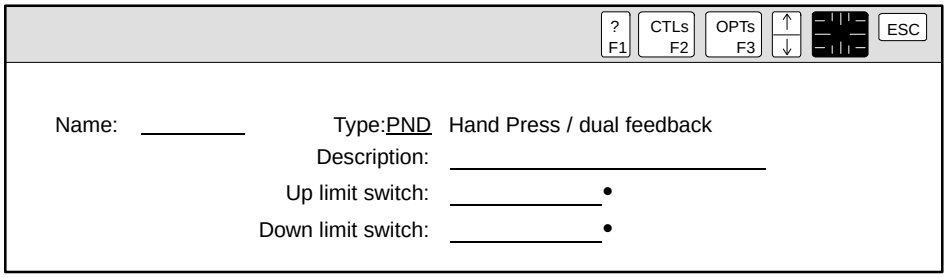
The two feedback signals consist of an up feedback signal (.ULS) and a down feedback signal (.DLS).

- The .ULS bit should be true when the press is up and false when it is down.
- The .DLS bit should be true when the press is down and false when it is up.

The .MRAISE extension shows the state of the press.

- If the press is up (.ULS=true), the control signal (.MRAISE) is set to true.
- If the desired state is down, (.DLS=true), the .MRAISE bit is set to false.

When you select PND in the Device Definition Table, this form appears:



Name: unique name that identifies valve (12 characters maximum)

Type: PND (3-letter code for device)

Description: 30 characters maximum (optional)

Up limit switch: symbolic name of discrete input, discrete output, digital flag, boolean or APT flag variable, or *device_name*.ULS that is up feedback signal.

Down limit switch: symbolic name of discrete input, discrete output, digital flag, boolean or APT flag variable, or *device_name*.DLS that is down feedback signal.

NOTE: The PND does not support the dual feedback logic. Refer to Appendix A for the logic of a PND.

Table 7-21 lists the extensions and commands used with a PND device.

Table 7-21 PND Extensions and Commands

Extension		Commands
Read-only Boolean		There are no commands.
.UP	press is up	
.DOWN	press is down	
.ULS [1]	up feedback	
.DLS [2]	down feedback	
.FAILD	failed (both feedback bits are true)	
.MRAISE	up/down status	
.STATUS	device status	

1 *device_name*.ULS is interchangeable with symbolic name configured as Up Limit Switch for energize-open presses. If feedback is not a DI, this is read/write.

2 *device_name*.DLS is interchangeable with symbolic name configured as Down Limit Switch for energize-closed presses. If feedback is not a DI, this is read/write.

Figure 7-21 shows how the input bits (on the left) affect the status bits (across the bottom) and/or the output bits (on the right).

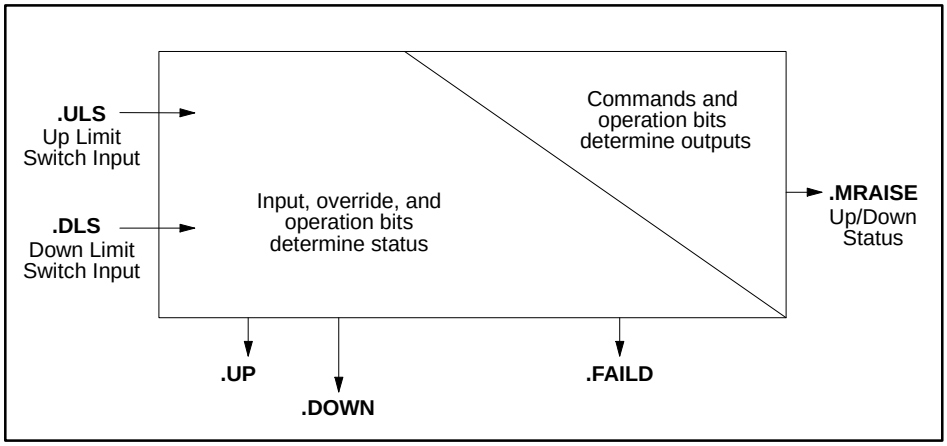


Figure 7-21 PND Extensions and Commands

Presses (continued)

Single-Drive/ Null-Feedback Press (PSN)

The PSN device has two positions (up and down) and is controlled by one discrete signal with no feedback. Two types of PSN devices are available: energize-raise (Type R) and energize-lower (Type L).

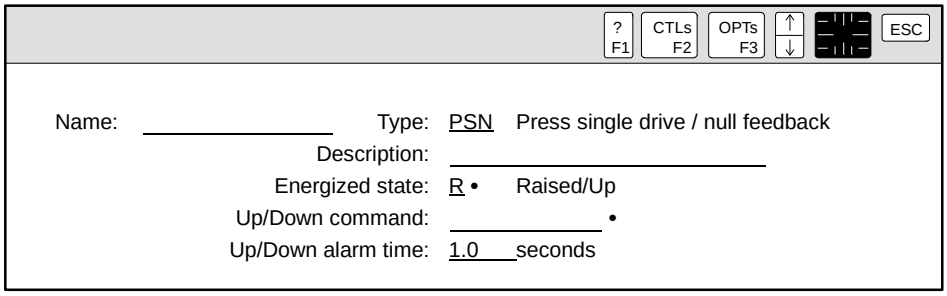
- If the desired state of an energize-raise press is up (.MRAISE=true), the field control signal (.CMMD) is set to true.

  If the desired state is down (.MRAISE=false), the .CMMD bit is set to false.

- If the desired state of an energize-lower press is up (.MRAISE=true), the control signal (.CMMD) is set to false.

  If the desired state is down (.MRAISE=false), the .CMMD bit is set to true.

When you select PSN in the Device Definition Table, the form below appears.



Name: unique name that identifies press (12 characters maximum)

Type: PSN (3-letter code for device)

Description: 30 characters maximum (optional)

Energized state: R indicates energize raise; L indicates energize lower; default is R.

Up/Down command: symbolic name of discrete output, boolean variable, or *device_name*.CMMD that raises and lowers the press.

Up/Down alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow press to change states.

Table 7-22 lists the extensions and commands used with a PSN device.

Table 7-22 PSN Extensions and Commands

Extension		Commands	
Read-only Boolean		LOCK	place in auto mode
.CMMD [1]	raise/lower command	UNLOCK	place in manual mode
.UP	up	RAISE	raise press
.DOWN	down	LOWER	lower press
.TRVL	traveling		
Read/Write Boolean			
.DSBLD	forced to manual mode		
.LOCKD	locked (auto mode)		
.NRDY	not ready		
.MRAISE	manual raise		
.STATUS	device status		
Read-only Integer			
.VFLAGS	device status (505 only)		
Read/Write Integer			
.UTCP	up timer/counter preset		
.UTCC	up timer/counter current		
.DTCP	down timer/counter preset		
.DTCC	down timer/counter current		

1 *device_name*.CMMD is interchangeable with symbolic name configured as Raise/Lower command.

Figure 7-22 shows how the operation bits (across the top) affect the status bits (across the bottom and/or the output bits (on the right).

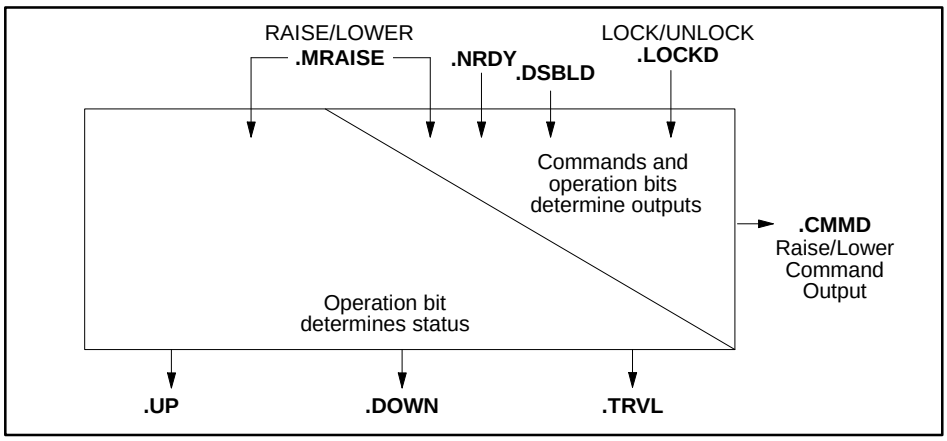


Figure 7-22 PSN Extensions and Commands

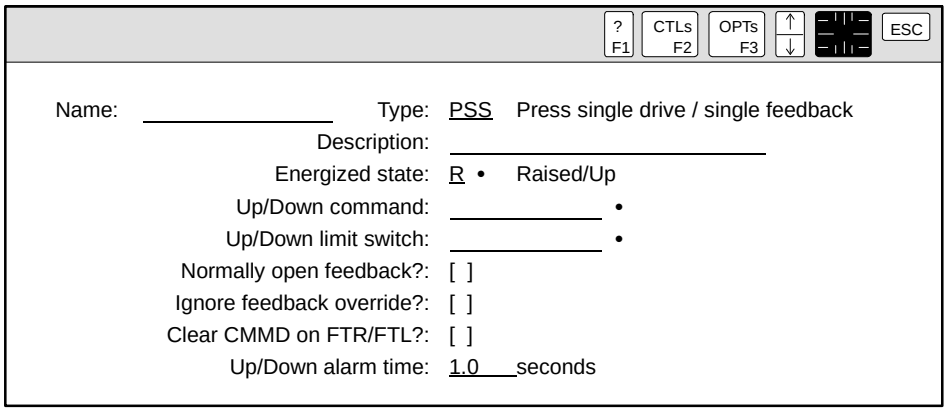
Presses (continued)

Single-Drive/ Single-Feedback Press (PSS)

The PSS device has two positions (up and down) and is controlled by a single discrete signal with one discrete feedback signal. Two types of PSS devices are available: energize-raise (Type R) and energize-lower (Type L).

- If the desired state of an energize-raise press is up (.MRAISE=true), the field control signal (.CMMD) is set to true. If the desired state is down (.MRAISE=false), the .CMMD bit is set to false.
- The feedback signal for the energize-raise press (.ULS) should be true when the press is up and false when the press is down.
- If the desired state of an energize-lower press is up (.MRAISE=true), the control signal (.CMMD) is set to false. If the desired state is down (.MRAISE=false), the .CMMD bit is set to true.
- The feedback signal for the energize-lower press (.DLS) should be false when the press is up and true when the press is down.
- If the Clear CMMD on FTR/FTL option is selected, the .CMMD bit will change to false when the .FTR bit becomes true. The .CMMD bit will remain false until a RESET command is issued.
- The RESET command issues a RAISE/LOWER command that turns on the .TRVL bit. The UP/DOWN alarm timer will start counting down when the RESET bit goes false.

When you select PSS in the Device Definition Table, the form below appears.



Name: unique name that identifies press (12 characters maximum)

Type: PSS (3-letter code for device)

Description: 30 characters maximum (optional)

Energized state: R indicates energize raise; L indicates energize lower; default is R.

Up/Down command: symbolic name of discrete output, boolean variable, or *device_name*.CMMD that raises and lowers press.

Up/Down limit switch: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.ULS that is the feedback signal.

Normally open feedback: feedback response is inverted, e.g, up feedback is false when the press is up.

Ignore feedback override: Override feedback bit is ignored.

Clear CMMD on FTR/FTL: the .CMMD bit becomes false when the .FTR or .FTL bit becomes true.

Up/Down alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow press to change states.

Presses (continued)

Table 7-23 lists the extensions and commands used with a PSS device.

Table 7-23 PSS Extensions and Commands

Extension		Commands	
Read-only Boolean		LOCK	place in auto mode
.CMMD [1]	raise/lower command	UNLOCK	place in manual mode
.UP	up	RAISE	raise press
.DOWN	down	LOWER	lower press
.TRVL	traveling	RESET	clear feedback override and/or issues raise/lower command after .FTR or .FTL is true.
.ULS [2]	up feedback (R)		
.DLS [3]	down feedback (L)		
.FTR	fail to raise		
.FTL	fail to lower		
Read/Write Boolean			
.DSBLD	forced to manual mode		
.LOCKD	locked (auto mode)		
.NRDY	not ready		
.MRAISE	manual raise		
.OVRD	override feedback		
.STATUS	device status		
Read-only Integer			
.VFLAGS	device status (505 only)		
Read/Write Integer			
.UTCP	up timer/counter preset		
.UTCC	up timer/counter current		
.DTCP	down timer/counter preset		
.DTCC	down timer/counter current		

1 *device_name*.CMMD is interchangeable with symbolic name configured as Raise/Lower command.
2 *device_name*.ULS is interchangeable with symbolic name configured as Up Limit Switch for energize-raise presses. If feedback is not a DI, this is read/write.
3 *device_name*.DLS is interchangeable with symbolic name configured as Down Limit Switch for energize-lower presses. If feedback is not a DI, this is read/write.

Figure 7-23 shows how the input bits (on the left) and the operation bits (across the top) affect the status bits (across the bottom) and/or the output bits (on the right).

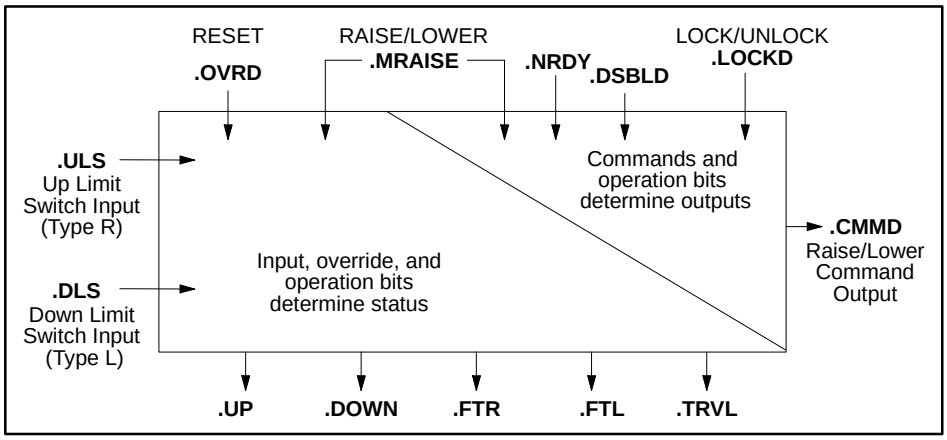


Figure 7-23 PSS Extensions and Commands

Presses (continued)

Single-Drive/ Dual-Feedback Press (PSD)

The PSD device has two positions (up and down) and is controlled by a single discrete signal with two discrete feedback signals. Two types of PSD devices are supported: energize-raise (Type R) and energize-lower (Type L).

One control signal (.CMMD) determines the state of the press.

- If the desired state of an energize-raise press is up (.MRAISE=true), the control signal (.CMMD) is set to true.

  If the desired state is down (.MRAISE=false), the .CMMD bit is set to false.

- If the desired state of an energize-lower press is up (.MRAISE=true), the control signal (.CMMD) is set to false.

  If the desired state is down (.MRAISE=false), the .CMMD bit is set to true.

- If the Clear CMMD on FTR/FTL option is selected, the .CMMD bit will change to false when the .FTR bit becomes true. The .CMMD bit will remain false until a RESET command is issued.

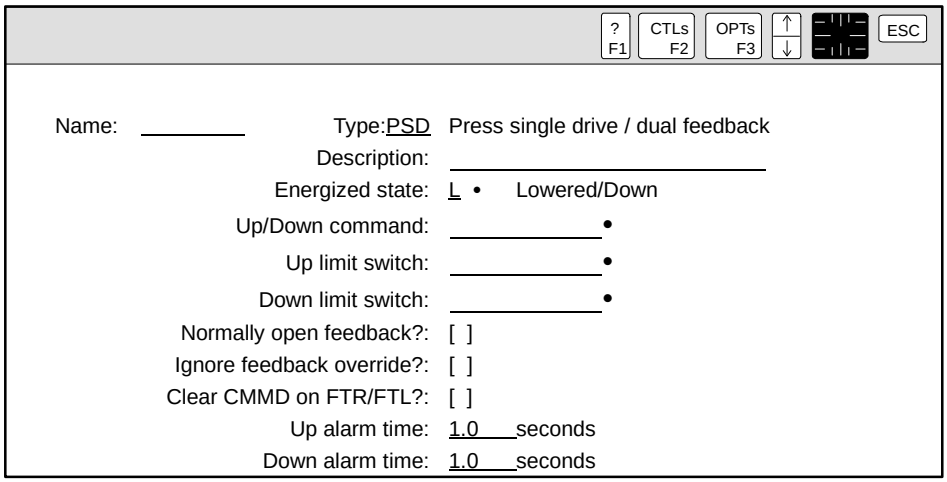
- The RESET command issues a RAISE/LOWER command that turns on the .TRVL bit. The UP/DOWN alarm timer will start counting down when the RESET bit goes false.

For both types of presses, the two feedback signals consist of an up feedback signal (.ULS) and a down feedback signal (.DLS).

- The .ULS bit should be true when the press is up and false when it is down.

- The .DLS bit should be true when the press is down and false when it is up.

When you select PSD in the Device Definition Table, the form below appears.



Name: unique name that identifies press (12 characters maximum)

Type: PSD (3-letter code for device)

Description: 30 characters maximum (optional)

Energized state: R indicates energize raise; L indicates energize lower; default is R.

Up/Down command: symbolic name of discrete output, boolean variable, or *device_name*.CMMD that raises and lowers press.

Up limit switch: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.ULS that is up feedback signal.

Down limit switch: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.DLS that is down feedback signal.

Normally open feedback: feedback response is inverted, e.g, up feedback is false when the press is up.

Ignore feedback override: Override feedback bit is ignored.

Clear CMMD on FTR/FTL: the .CMMD bit becomes false when the .FTR or .FTL bit becomes true.

Up alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow press to raise.

Down alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow press to lower.

Presses (continued)

Table 7-24 lists the extensions and commands used with a PSD device.

Table 7-24 PSD Extensions and Commands

Extension		Commands	
Read-only Boolean		LOCK	place in auto mode
.CMMD [1]	up/down command	UNLOCK	place in manual mode
.UP	up	RAISE	raise press
.DOWN	down	LOWER	lower press
.TRVL	traveling	RESET	clear feedback override and/or issues raise/lower command after .FTR or .FTL is true.
.ULS [2]	up feedback (R)		
.DLS [3]	down feedback (L)		
.FTR	fail to raise		
.FTL	fail to lower		
.FAILD	failed (both feedback bits are true)		
Read/Write Boolean			
.DSBLD	forced to manual mode		
.LOCKD	locked (auto mode)		
.NRDY	not ready		
.MRAISE	manual raise		
.OVRDU	override up feedback		
.OVRDD	override down feedback		
.STATUS	device status		
Read-only Integer			
.VFLAGS	device status (505 only)		
Read/Write Integer			
.UTCP	up timer/counter preset		
.UTCC	up timer/counter current		
.DTCP	down timer/counter preset		
.DTCC	down timer/counter current		

1 *device_name*.CMMD is interchangeable with symbolic name configured as Raise/Lower command.
2 *device_name*.ULS is interchangeable with symbolic name configured as Up Limit Switch for energize-raise presses. If feedback is not a DI, this is read/write.
3 *device_name*.DLS is interchangeable with symbolic name configured as Down Limit Switch for energize-lower presses. If feedback is not a DI, this is read/write.

Figure 7-24 shows how the input bits (on the left) and the operation bits (across the top) affect the status bits (across the bottom) and/or the output bits (on the right).

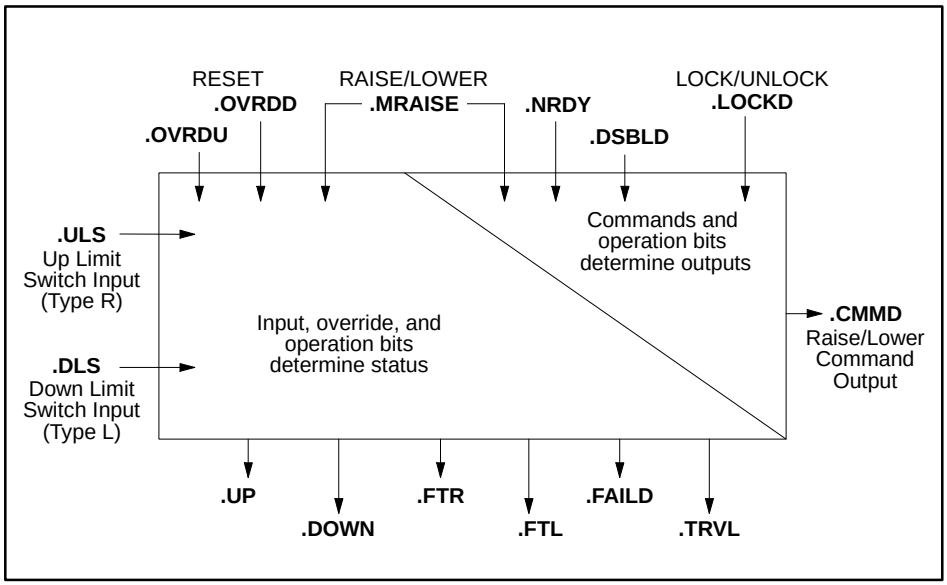


Figure 7-24 PSD Extensions and Commands

Presses (continued)

Dual-Drive/ Dual-Feedback Press (PDD)

The PDD device has two positions (up and down) and is controlled by two discrete signals with two discrete feedback signals.

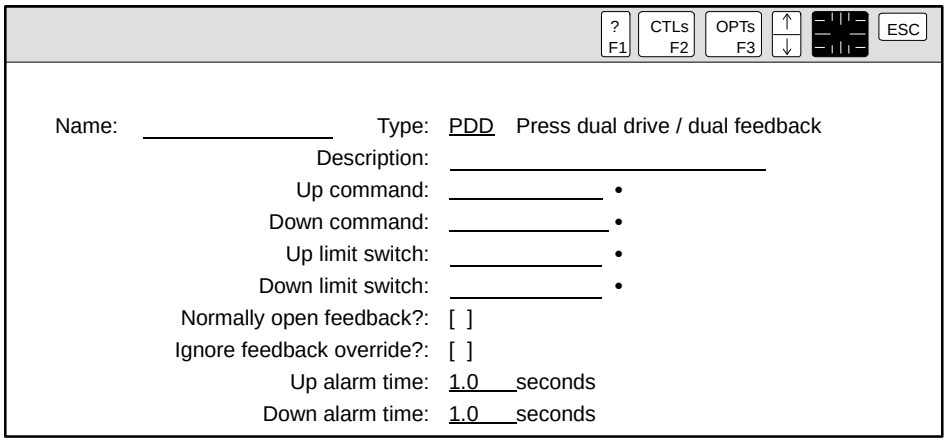
The two control signals consist of an up signal (.UPC) and a down signal (.DOWNC), which are both normally false.

- If the desired state is up (.MRAISE=true), the .UPC bit is set to true. The .UPC bit remains true until either the up feedback signal is true or the up alarm time expires; then .UPC is set to false.
- If the desired state is down, the .DOWNC bit is set to true to lower the press. The .DOWNC bit remains true until either the down feedback signal is true or the down alarm time expires; then .DOWNC is set to false.
- The RESET command issues a RAISE/LOWER command that turns on the .TRVL bit. The UP/DOWN alarm timer will start counting down when the RESET bit goes false.

The two feedback signals consist of an up feedback signal (.ULS) and a down feedback signal (.DLS).

- The .ULS bit should be true when the press is up; otherwise, it should be false.
- The .DLS bit should be true when the press is down; otherwise, it should be false.

When you select PDD in the Device Definition Table, the form below appears.



Name: unique name that identifies press (12 characters maximum)

Type: PDD (3-letter code for device)

Description: 30 characters maximum (optional)

Up command: symbolic name of discrete output, boolean variable, or *device_name*.UPC that raises press.

Down command: symbolic name of discrete output, boolean variable, or *device_name*.DOWNC that lowers press.

Up limit switch: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.ULS that is up feedback signal.

Down limit switch: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.DLS that is down feedback signal.

Normally open feedback: feedback response is inverted, e.g, up feedback is false when the press is up.

Ignore feedback override: Override feedback bit is ignored.

Up alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow press to raise.

Down alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow press to lower.

Presses (continued)

Table 7-25 lists the extensions and commands used with a PDD device.

Table 7-25 PDD Extensions and Commands

Extension		Commands	
Read-only Boolean		LOCK	place in auto mode
.CMMD	raise/lower command	UNLOCK	place in manual mode
.UPC [1]	up command	RAISE	raise press
.DOWNC [2]	lower command	LOWER	lower press
.UP	up	RESET	clear feedback override
.DOWN	down		
.TRVL	traveling		
.ULS [3]	up feedback		
.DLS [4]	down feedback		
.FTR	fail to raise		
.FTL	fail to lower		
.FAILD	failed (both feedback bits are true)		
.UPTO	up timeout		
.DOWNTO	down timeout		
Read/Write Boolean			
.DSBLD	forced to manual mode		
.LOCKD	locked (auto mode)		
.NRDY	not ready		
.MRAISE	manual raise		
.OVRDU	override up feedback		
.OVRDD	override down feedback		
.STATUS	device status		
Read-only Integer			
.VFLAGS	device status (505 only)		
Read/Write Integer			
.UTCP	up timer/counter preset		
.UTCC	up timer/counter current		
.DTCP	down timer/counter preset		
.DTCC	down timer/counter current		

1 *device_name*.UPC is interchangeable with symbolic name configured as Raise command.
2 *device_name*.DOWNC is interchangeable with symbolic name configured as Lower command.
3 *device_name*.ULS is interchangeable with symbolic name configured as Up Limit Switch. If feedback is not a DI, this is read/write.
4 *device_name*.DLS is interchangeable with symbolic name configured as Down Limit Switch. If feedback is not a DI, this is read/write.

Figure 7-25 shows how the input bits (on the left) and the operation bits (across the top) affect the status bits (across the bottom) and/or the output bits (on the right).

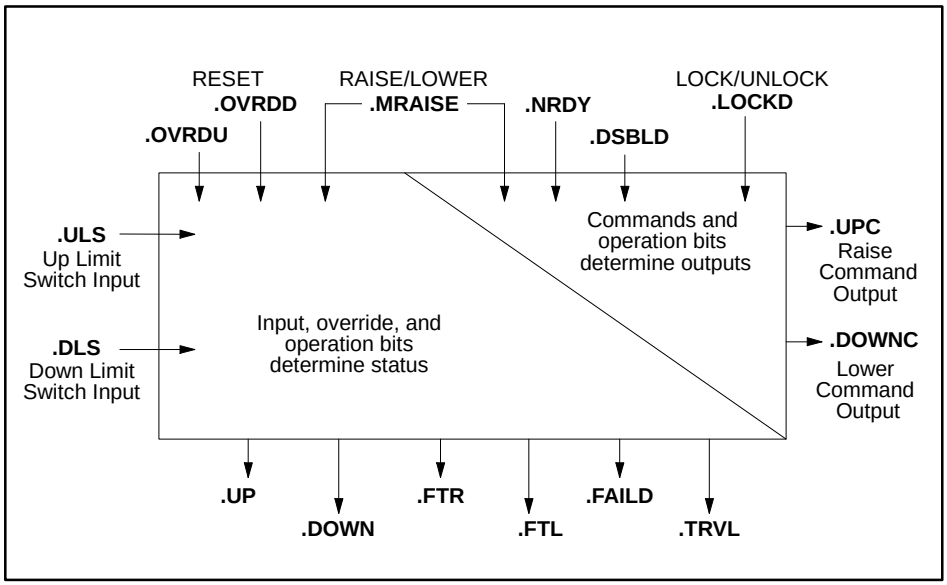


Figure 7-25 PDD Extensions and Commands

Presses (continued)

Motor-Drive/ Dual-Feedback Press (PMD)

The PMD device has two positions (up and down) and is controlled by two discrete signals with two discrete feedback signals.

The two control signals consist of a raise signal (.UPC) and a lower signal (.DOWNC).

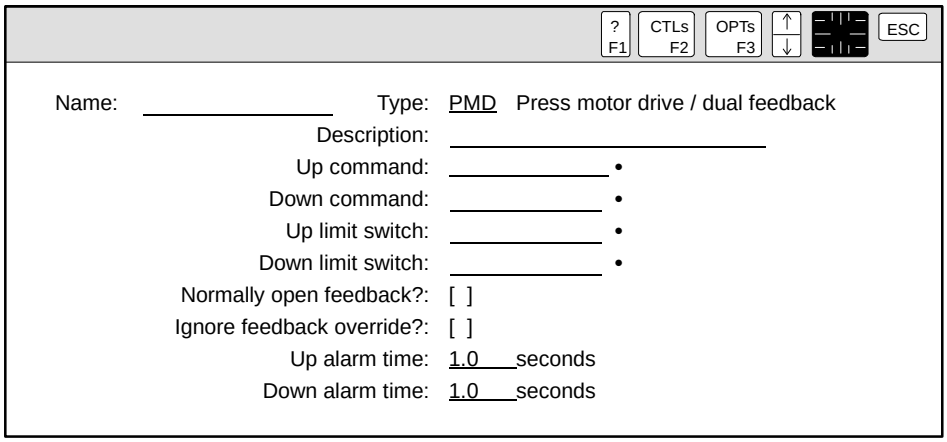
- If the desired state of the press is up (.MRAISE=true), the .UPC bit is set to true until the up feedback is true or until the alarm time expires; then .UPC is set to false.
- If the desired state is down (.MRAISE=false), the .DOWNC bit is set to true until down feedback is true or until the alarm time expires; then .DOWNC is set to false.
- If the press is stopped in mid-travel, the .TRVL bit remains true with the up and down alarm times reset.
- The RESET command issues a RAISE/LOWER command that turns on the .TRVL bit. The UP/DOWN alarm timer will start counting down when the RESET bit goes false.

The two feedback signals consist of an up feedback signal (.ULS) and a down feedback signal (.DLS).

- The .ULS bit should be true when the press is up; otherwise, it should be false.
- The .DLS bit should be true when the press is down; otherwise, it should be false.

The PMD press can be stopped at any point of its travel by setting the .RTS extension to true.

When you select PMD in the Device Definition Table, the form appears as shown below.



Name: unique name that identifies motor-driven press (12 characters maximum)

Type: PMD (3-letter code for device)

Description: 30 characters maximum (optional)

Up command: symbolic name of discrete output, boolean variable, or *device_name*.UPC that raises motor press.

Down command: symbolic name of discrete output, boolean variable, or *device_name*.DOWNC that lowers motor press.

Up limit switch: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.ULS that is up feedback signal.

Down limit switch: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.DLS that is down feedback signal.

Normally open feedback: feedback response is inverted, e.g, up feedback is false when the press is up.

Ignore feedback override: Override feedback bit is ignored.

Up alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow press to raise; also indicates maximum time to keep up signal true.

Down alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow press to lower; also indicates maximum time to keep down signal true.

Table 7-26 lists the extensions and commands used with a PMD device.

Presses (continued)

Table 7-26 PMD Extensions and Commands

Extension		Commands	
Read-only Boolean		LOCK	place in auto mode
.CMMD	raise/lower command	UNLOCK	place in manual mode
.UPC [1]	up command	RAISE	raise press
.DOWNC [2]	lower command	LOWER	lower press
.UP	up	RESET	clear feedback override
.DOWN	down		
.TRVL	traveling		
.ULS [3]	up feedback		
.DLS [4]	down feedback		
.FTR	fail to raise		
.FTL	fail to lower		
.FAILD	failed (both feedback bits are true)		
.UPTO	up timeout		
.DOWNTO	down timeout		
Read/Write Boolean			
.DSBLD	forced to manual mode		
.LOCKD	locked (auto mode)		
.NRDY	not ready		
.MRAISE	manual raise		
.OVRDU	override up feedback		
.OVRDD	override down feedback		
.STATUS	device status		
.RTS	stop travel		
Read-only Integer			
.VFLAGS	device status (505 only)		
Read/Write Integer			
.UTCP	up timer/counter preset		
.UTCC	up timer/counter current		
.DTCP	down timer/counter preset		
.DTCC	down timer/counter current		

1 *device_name*.UPC is interchangeable with symbolic name configured as Raise command.
2 *device_name*.DOWNC is interchangeable with symbolic name configured as Lower command.
3 *device_name*.ULS is interchangeable with symbolic name configured as Up Limit Switch. If feedback is not a DI, this is read/write.
4 *device_name*.DLS is interchangeable with symbolic name configured as Down Limit Switch. If feedback is not a DI, this is read/write.

Figure 7-26 shows how the input bits (on the left) and the operation bits (across the top) affect the status bits (across the bottom) and/or the output bits (on the right).

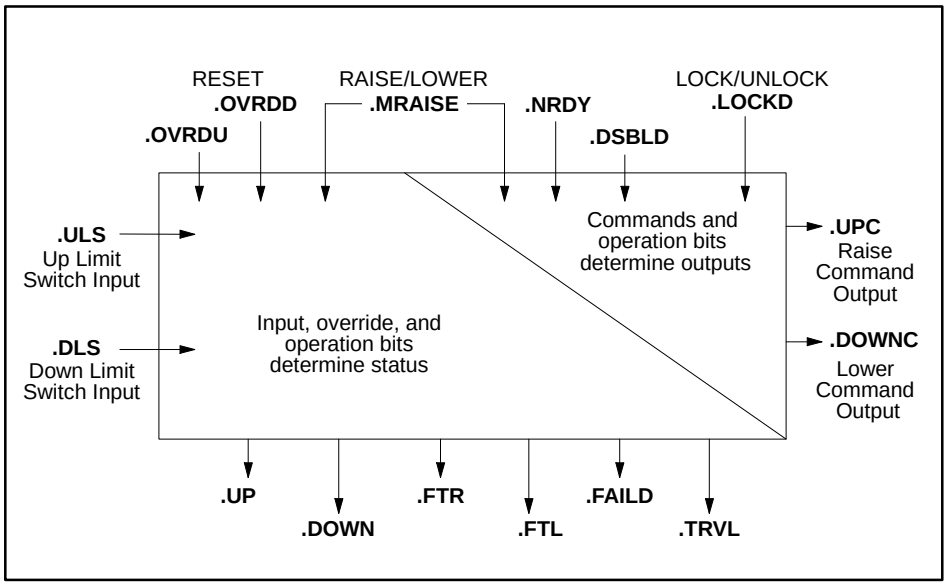


Figure 7-26 PMD Extensions and Commands

Presses (continued)

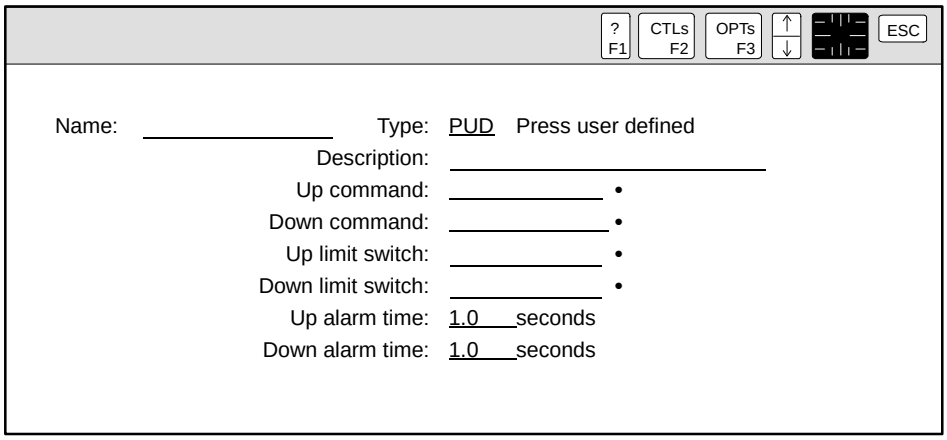
User-defined Press (PUD)

The PUD device has two positions (up and down) and is controlled by two discrete signals with two discrete feedback signals.

The PUD is essentially the shell of a PDD device. The code which defines the PDD has been reduced to a minimum, and extensions have been provided to allow you to create your own customized press and translate it for OSx (PCS).

The PUD has two alarm timers. When the .URESET (.DRESET) bit transitions from false to true, the alarm timer starts. When the timer times out, the .UPTO (.DOWNTO) bit becomes true, and remains true until the .URESET (.DRESET) bit becomes false.

When you select PUD in the Device Definition Table, the form below appears.



Name: unique name that identifies press (12 characters maximum)

Type: PUD (3-letter code for device)

Description: 30 characters maximum (optional)

Up command: symbolic name of discrete output, boolean variable, or *device_name*.UPx (where x = A, B, or C) that raises press.

Down command: symbolic name of discrete output, boolean variable, or *device_name*.DOWNC that lowers press.

Up limit switch: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.ULS that is up feedback signal.

Down limit switch: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.DLS that is down feedback signal.

Up alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow press to raise.

Down alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow press to lower.

Table 7-27 lists the extensions and commands used with a PUD device.

Presses (continued)

Table 7-27 PUD Extensions and Commands

Extension		Commands	
Read-only Boolean		LOCK	place in auto mode
.CMMD	raise/lower command	UNLOCK	place in manual mode
.UPx [1]	raise command	RAISE	raise press
.DOWNC [2]	lower command	LOWER	lower press
.UP	up	RESET	clear feedback override
.DOWN	down	RAISEA	raise with option A
.TRVL	traveling	RAISEB	raise with option B
.ULS [3]	up feedback	RAISEH	raise high
.DLS [4]	down feedback	RAISEL	raise low
.FTR	fail to raise		
.FTL	fail to lower		
.FAILD	failed (both feedback bits are true)		
.UPTO	up alarm timer time out		
.DOWNTO	down alarm timer time out		
Read/Write Boolean			
.DSBLD	forced to manual mode		
.LOCKD	locked (auto mode)		
.NRDY	not ready		
.MRAISE	manual raise		
.OVRDU	override up feedback		
.OVRDD	override down feedback		
.URESET	up alarm timer reset		
.DRESET	down alarm timer reset		
.STATUS	device status		
Read-only Integer			
.VFLAGS	device status (505 only)		
Read/Write Integer			
.UTCP	up timer/counter preset		
.UTCC	up timer/counter current		
.DTCP	down timer/counter preset		
.DTCC	down timer/counter current		

1 *device_name*.UPx is interchangeable with symbolic name configured as Raise command. The extension is .UPA when x = A, .UPB when x = B, etc.
2 *device_name*.DOWNC is interchangeable with symbolic name configured as Lower command.
3 *device_name*.ULS is interchangeable with symbolic name configured as Up Limit Switch. If feedback is not a DI, this is read/write.
4 *device_name*.DLS is interchangeable with symbolic name configured as Down Limit Switch. If feedback is not a DI, this is read/write.

Figure 7-27 shows how the input bits (on the left) and the operation bits (across the top) affect the status bits (across the bottom) and/or the output bits (on the right).

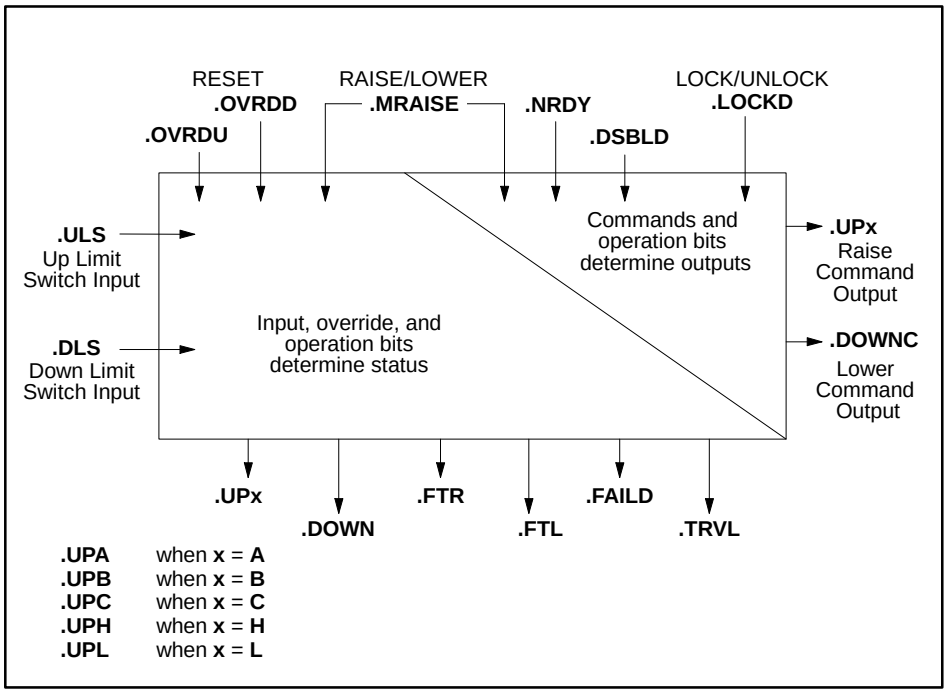


Figure 7-27 PUD Extensions and Commands

7.8 Three-Position Press with Dual Feedback

Three-Position Press/Type 1 (PS1)

The PS1 device is a three-position press (high, low, and down) and is controlled by two discrete signals with two discrete feedback signals.

The two discrete control signals consist of a low signal (.SLOW) and a high signal (.SHIGH). The two signals cannot both be true at the same time.

- If the desired state of the press is up at high position (.MRAISE=true and .MHIGH=true), the .SHIGH bit is set to true (energized) to raise the press at high position.
- If the desired state of the press is up at low position (.MRAISE=true and .MHIGH=false), the .SLOW bit is set to true (energized) to raise the press at low position.
- If the desired state is down (.MRAISE=false), the currently energized signal (.SHIGH or .SLOW) is set to false.
- The press, when receiving an up signal, starts raising and keeps raising until the currently energized signal becomes false.
- The RESET command issues a RAISE/LOWER command which turns on the .TRVL bit. The RAISE/LOWER alarm timer will start counting down when the RESET bit goes false.

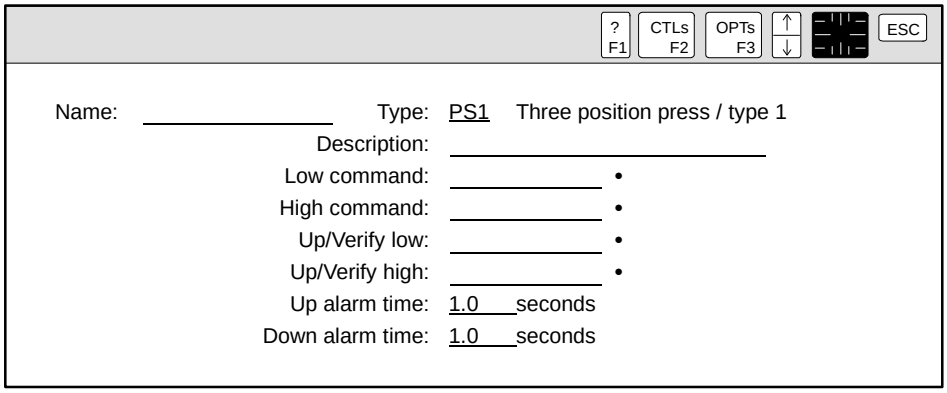
The two feedback signals consist of an raise-low feedback signal (.LIO) and an raise-high feedback signal (.HIO).

- The .LIO bit should be true when the press is up at the low position; otherwise, it should be false.
- The .HIO bit should be true when the press is up at the high position; otherwise, it should be false.

In manual mode, the outputs (.SLOW/.SHIGH) are set to the appropriate state based on the status of the .MHIGH and the .MRAISE bits, which you manipulate from an operator station or from the program.

In auto mode, the RAISEL, RAISEH, and LOWER commands set the state of the output bits. In this mode, the .MHIGH and .MRAISE are set to reflect the last requested state. This is done to provide for a bumpless transfer if the device changes modes.

When you select PS1 in the Device Definition Table, the form below appears.



Name: unique name that identifies press (12 characters maximum)

Type: PS1 (3-letter code for device)

Description: 30 characters maximum (optional)

Low command: symbolic name of discrete output, boolean variable, or *device_name*.SLOW that raises press at low position.

High command: symbolic name of discrete output, boolean variable, or *device_name*.SHIGH that raises press at high position.

Up/Verify low: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.LIO that is low feedback signal.

Up/Verify high: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.HIO that is high feedback signal.

Up alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow press to change from down to up.

Down alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow press to change from up to down.

Table 7-28 lists the extensions and commands used with a PS1 device.

Three-Position Press with Dual Feedback (continued)

Table 7-28 PS1 Extensions and Commands

Extension		Commands	
Read-only Boolean		LOCK	place in auto mode
.SLOW [1]	raise low	UNLOCK	place in manual mode
.SHIGH [2]	raise high	RAISEL	raise low
.UP	up	RAISEH	raise high
.UPL	up low	LOWER	lower press
.UPH	up high	RESET	clear feedback overrides
.DOWN	down		
.TRVL	traveling		
.LIO [3]	low feedback		
.HIO [4]	high feedback		
.FTRL	fail to raise low		
.FTRH	fail to raise high		
.FTR [5]	fail to raise		
.FTL	fail to lower		
.FAILD	failed (both feedback bits are true)		
.UPTO	up timeout		
.DOWNTO	down timeout		
Read/Write Boolean			
.DSBLD	forced to manual mode		
.LOCKD	locked (auto mode)		
.NRDY	not ready		
.MHIGH	manual high		
.MRAISE	manual raise		
.OVRDL	override low feedback		
.OVRDH	override high feedback		
.STATUS	device status		
Read-only Integer			
.VFLAGS	device status (505 only)		
Read/Write Integer			
.UTCP	up timer/counter preset		
.UTCC	up timer/counter current		
.DTCP	down timer/counter preset		
.DTCC	down timer/counter current		

1 *device_name*.SLOW is interchangeable with symbolic name configured as low command.
2 *device_name*.SHIGH is interchangeable with symbolic name configured as high command.
3 *device_name*.LIO is interchangeable with symbolic name configured as Raised/Verify low feedback signal. If feedback is not a DI, this is read/write.
4 *device_name*.HIO is interchangeable with symbolic name configured as Raised/Verify high feedback signal. If feedback is not a DI, this is read/write.
5 The .FTR bit is set to true when either the .FTRL or .FTRH bit is true.

Figure 7-28 shows how the input bits (on the left) and the operation bits (across the top) affect the status bits (across the bottom) and/or the output bits (on the right).

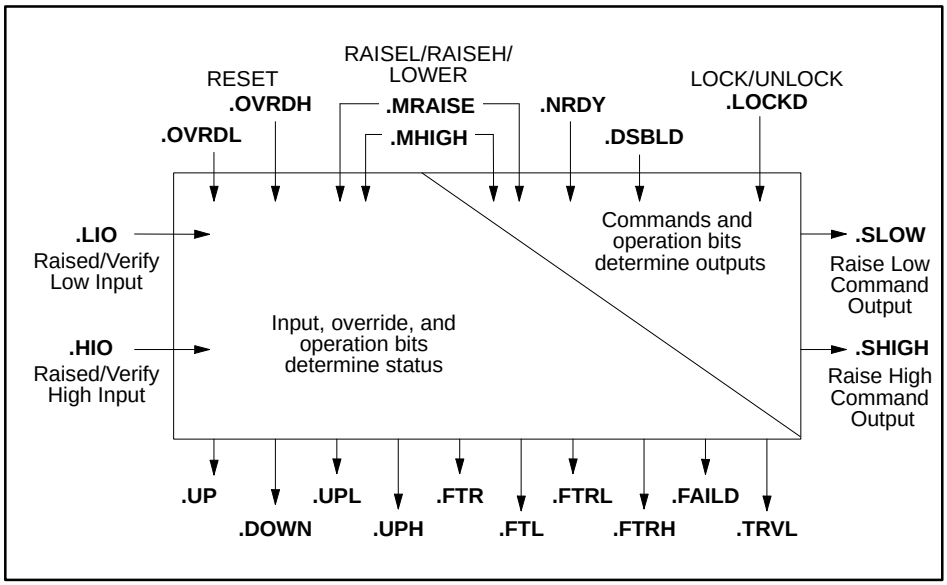


Figure 7-28 PS1 Extensions and Commands

Three-Position Press with Dual Feedback (continued)

Three-Position Press/Type 2 (PS2)

The PS2 device is a three-position press (high, low, and down) and is controlled by two discrete signals with two discrete feedback signals.

The two discrete control signals consist of a raise/lower signal (.DRV) and a position signal (.POS) that determines whether the press raises at high or low position.

- If the desired state of the press is raise high (.MRAISE=true and .MHIGH=true), the .DRV and .POS bits are both set to true.
- If the desired state is raise low (.MRAISE=true and .MHIGH=false), the .DRV bit is set to true and the .POS bit is set to false.
- If the desired state is down (.MRAISE=false), the .DRV bit is set to false to lower the press.
- The press, when receiving a raise signal, starts raising and keeps raising until .DRV becomes false.
- The RESET command issues a RAISE/LOWER command that turns on the .TRVL bit. The RAISE/LOWER alarm timer will start counting down when the RESET bit goes false.

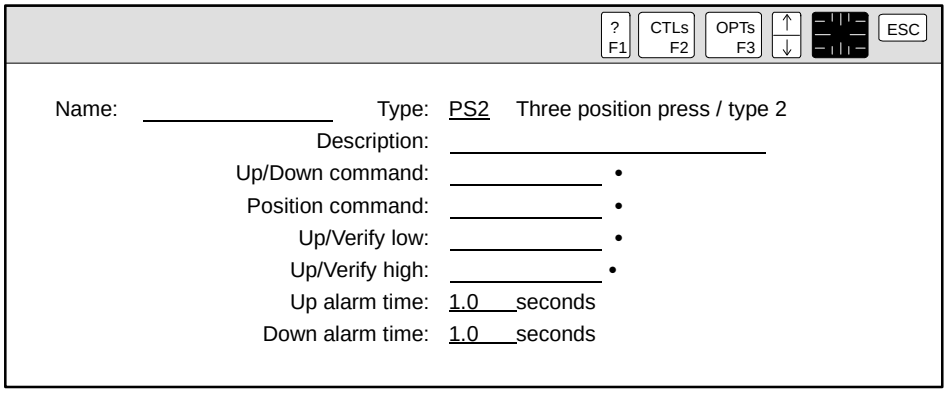
The two feedback signals consist of an raise-low feedback signal (.LIO) and a raise-high feedback signal (.HIO).

- The .LIO bit should be true when the press is up at low position; otherwise, it should be false.
- The .HIO should be true when the press is up at high position; otherwise it should be false.

In manual mode, the outputs (.DRV/.POS) are set to the appropriate state based on the status of the .MHIGH and the .MRAISE bits, which you manipulate from an operator station or from the program.

In auto mode, the RAISEL, RAISEH, and LOWER commands set the state of the output bits. In this mode, the .MHIGH and .MRAISE bits are set to reflect the last requested state. This is done to provide for a bumpless transfer if the device changes modes.

When you select PS2 in the Device Definition Table, the form below appears.



Name: unique name that identifies press (12 characters maximum)

Type: PS2 (3-letter code for device)

Description: 30 characters maximum (optional)

Up/Down command: symbolic name of discrete output, boolean variable, or *device_name*.DRV that raises and lowers press.

Position command: symbolic name of discrete output, boolean variable, or *device_name*.POS that indicates the position signal.

Up/Verify low: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.LIO that is low feedback signal.

Up/Verify high: symbolic name of discrete input, discrete output, digital flag, boolean variable, APT flag variable, or *device_name*.HIO that is high feedback signal.

Up alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow press to change from down to up.

Down alarm time: real number between 0.1 and 3276.7 that indicates number of seconds to allow press to change from up to down

Table 7-29 lists the extensions and commands used with a PS2 device.

Three-Position Press with Dual Feedback (continued)

Table 7-29 PS2 Extensions and Commands

Extension		Commands	
Read-only Boolean		LOCK	place in auto mode
.DRV [1]	raise/lower command	UNLOCK	place in manual mode
.POS [2]	position	RAISEL	raise low
.UP	up	RAISEH	raise high
.UPL	raised low	LOWER	lower press
.UPH	raised high	RESET	clear feedback overrides
.DOWN	down		
.TRVL	traveling		
.LIO [3]	low feedback		
.HIO [4]	high feedback		
.FTRL	fail to raise low		
.FTRH	fail to raise high		
.FTR [5]	fail to raise		
.FTL	fail to lower		
.FAILD	failed (both feedback bits are true)		
.UPTO	up timeout		
.DOWNTO	down timeout		
Read/Write Boolean			
.DSBLD	forced to manual mode		
.LOCKD	locked (auto mode)		
.NRDY	not ready		
.MHIGH	manual high		
.MRAISE	manual raise		
.OVRDL	override low feedback		
.OVRDH	override high feedback		
.STATUS	device status		
Read-only Integer			
.VFLAGS	device status (505 only)		
Read/Write Integer			
.UTCP	up timer/counter preset		
.UTCC	up timer/counter current		
.DTCP	down timer/counter preset		
.DTCC	down timer/counter current		

1 *device_name*.DRV is interchangeable with symbolic name configured as up/down command.
2 *device_name*.POS is interchangeable with symbolic name configured as position command.
3 *device_name*.LIO is interchangeable with symbolic name configured as Raised/Verify low feedback signal.
4 *device_name*.HIO is interchangeable with symbolic name configured as Raised/Verify high feedback signal.
5 The .FTR bit is set to true when either the .FTRL or .FTRH bit is true.

Figure 7-29 shows how the input bits (on the left) and the operation bits (across the top) affect the status bits (across the bottom) and/or the output bits (on the right).

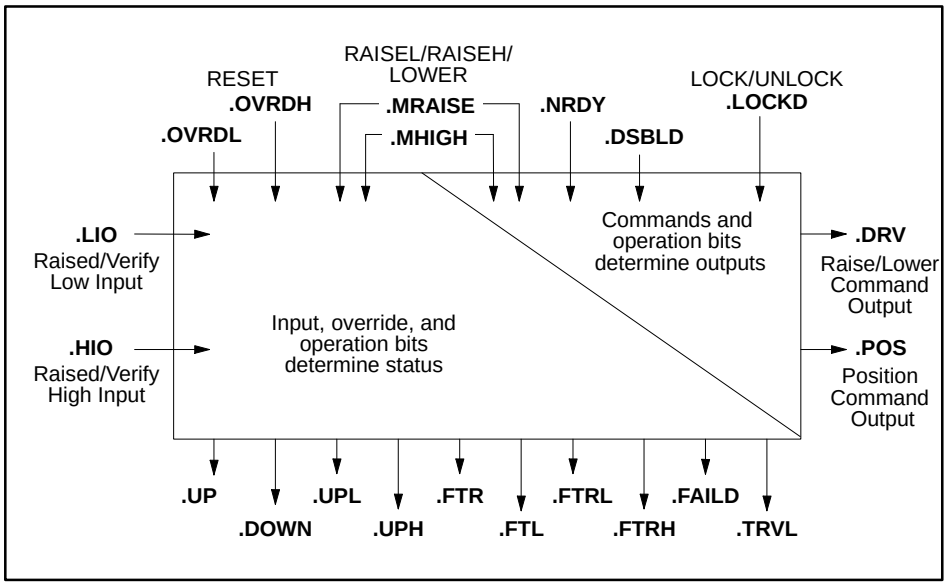


Figure 7-29 PS2 Extensions and Commands

7.9 Stopwatch

Using the Device Timer

The TMR device is a three-state device (running, stopped, and hold). The timer can be controlled by commands in an SFC step or math statement or by assigning values in math statements.

Five commands allow you to manipulate the timer device from an SFC step or math block that generates RLL code.

- The START command starts the timer by setting the current value (.CUR) to 0 and the .RUNNG bit to true.

  Each time the tick-rate value elapses, the .CUR value is increased by 1.

- The STOP command stops the timer by setting the .RUNNG bit to false. The .TOUT extension also becomes false. The STOP command has no effect on the current value.

- The HOLD command sets the .HOLD bit to true to stop the timer temporarily. This command freezes the current value and the current tick-rate position. The .HOLD bit is automatically set to true if there is a controller power failure, and the Uninterruptible Power Supply option has not been selected in the compile control file.

- The CONTINUE command sets the .HOLD bit to false and continues timing from the point at which it was frozen by the HOLD command.

- The RESET command sets the .CUR value to 0. If the timer is frozen with a HOLD command, the RESET command unfreezes it by setting the .HOLD bit to false.

The .CUR value is normally a read-only integer; however, it is possible to write to the .CUR variable in a math statement. The .PSET value is an integer variable that contains the preset value of the timer.

NOTE: Because the TMR device stores data temporarily by using internal memory locations, avoid using timers in user-defined subroutines.

NOTE: If you write to the current value (.CUR) while the timer is running, the timer continues to count from the new current value.

Some timer commands take precedence over the other commands:

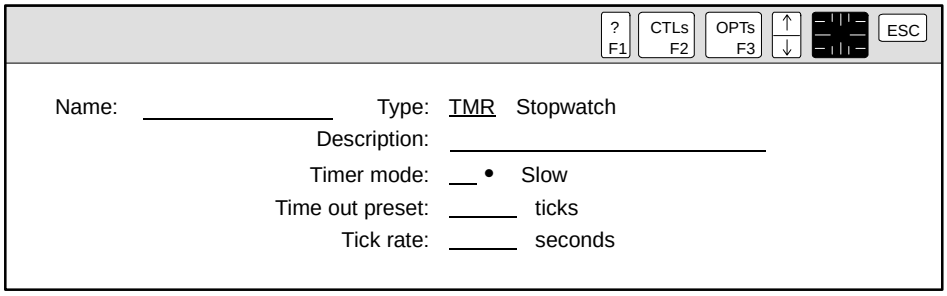
- RESET works at any time.
- STOP takes precedence over START, HOLD, and CONTINUE.
- CONTINUE and RESET take precedence over HOLD.

The tick rate of the timer can be changed by writing to the integer variable, *device_name*.RATE.

- If the timer is a slow timer (Type S), the value should be the number of tenths of seconds in each tick.
- If the timer is a fast timer (Type F), the value should be the number of milliseconds in each tick.
- If you change the tick rate while the timer is running, the change does not take effect until the current tick is completed.

Timer (TMR)

When you select TMR in the Device Definition Table, this form appears:



Name: unique name that identifies timer (12 characters maximum)

Type: TMR (3-letter code for device)

Description: 30 characters maximum (optional)

Timer mode: F indicates fast timer (minimum rate of .001 second);
S indicates slow timer (minimum rate of .1 second). For S5 controllers, only the slow timer is available.

Time out preset: real number between 0.0 and 32767.0 that indicates number of ticks between starting and stopping.

Tick rate: real number between 0.0 and 3276.7 seconds in 0.1 second increments for a slow timer; the range for a fast timer is 0.0 to 32.767 seconds in 0.001 second increments.

Stopwatch (continued)

Table 7-30 lists the extensions and commands used with a TMR device.

Table 7-30 TMR Extensions and Commands

Extension		Commands	
Read-only Boolean		START	start timer
.TOUT	time out (.CUR ≥ .PSET)	STOP	stop timer
.GT	time when .CUR > .PSET	HOLD	hold current time
.EQ [1]	value when .CUR = .PSET	CONT	continue counting
.MAXC	maximum value (32767) reached	RESET	clear current count
.HOLD	holding		
Read/Write Boolean			
.RUNNG	running		
Read/Write Integer			
.PSET	preset		
.RATE [2]	tick rate of timer		
.CUR	current timer count		

1 *device_name*.EQ may never come true for timers with very small tick rates (less than 100 milliseconds). The program cannot access the timer fast enough to catch the current value equal to the preset; the current value may appear to jump from less than .PSET to greater than .PSET.

2 For a slow timer, *device_name*.RATE = 10 × tick rate that you enter in the form. For a fast timer, *device_name*.RATE = 1000 × tick rate that you enter in the form.

Chapter 8

APT Declarations

8.1 Declaring Constants and Variables

Overview

To use constants and variables in your APT program, you must specify the names for these values in the Declaration Table. The Declaration Table Editor allows you to define constants and variables that can be used throughout the program. Constants have an unchanging assigned value. Variables can be assigned an initial value that can change during the execution of the program.

You can declare these constants and variables in the Declaration Definition Table at either the Program Content Level or the Unit Content Level. A constant or variable defined at the Program Content Level can be used in that program and in any unit within that program; a constant or variable defined at the Unit Content Level can be used only from in that unit.

Availability

The Declaration Table is available for both Series 505 and S5 controllers.

Declaration Types

APT provides seventeen types of declarations. Figure 8-1 shows the completion aid (•) for these types with the appropriate letter code for each. These declaration types fall into three general categories: single values, arrays, and timers.

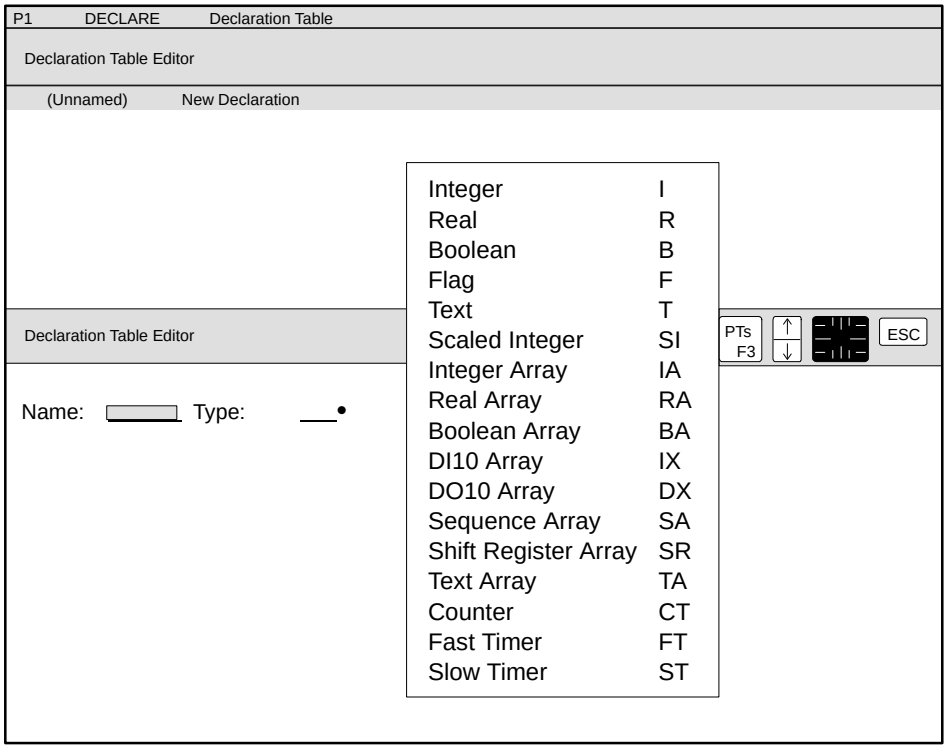


Figure 8-1 Declaration Types

Entering Constants and Variables

When you declare constants and variables, you enter information in the following fields, which can vary for declaration types.

Name: The name of the variable or constant can be from 1 to 12 characters long and can consist of any combination of letters, digits, and/or underscores. At least one of the characters must be a letter. The name must be unique within the current program or unit. It cannot be the same as the name of any other object within its scope.

Type: The code represents the type of value that you are declaring.

Description: The description is optional and cannot exceed 30 characters in length.

Constant: Put an X between the brackets to indicate that the declaration is a constant value. Leave the field blank if the declaration is a variable.

Value: The numeric value must agree with the specified type. The default is zero. If you are defining a constant, the entry in this field represents an unchanging value and is a read-only value. If you are defining a variable, the entry in this field is the initial value, which can be overwritten in an assignment statement.

You can enter the name of a constant declaration for the initial value of a variable that can be overwritten in an assignment statement.

Engineering Units: Optional documentation field to describe the measurement units, such as kg, liter, or psi (8 characters maximum).

Create Unique Status: Put an X between the brackets to indicate that a separate integer will be translated for an OSx (PCS) status attribute. You must also write code in a Math block that changes values of the status by writing to the **.STATUS** extension.

Create Status Word: Put an X between the brackets to indicate that a status word will be created for OSx (PCS). Refer to “Create Unique Status” for more information.

PLC Address: There are three controller addressing choices:

- Automatic — APT automatically assigns the controller address. Automatic addressing is the default.
- User Assigned — Allows the user to specify a controller address. Selecting this option brings up the Reserved Address field.
- None — No controller address is assigned. This is primarily intended for support of OSx (PCS) non-networked tags.

Reserved Address: Enter the controller address in this field if you choose to assign the declaration address yourself. For Series 505 controllers, specify the address in standard direct controller addressing, using this format: **%Annnn**. The **%** indicates that this is a direct address, **A** is a memory type (**C** for boolean, **V** for real or integer numbers), and **nnnn** is the address location. For S5 controllers, use the format **%Annn.n** for booleans, **%AWnnn** for integers, and **%ADnnn** for real numbers. The **%** indicates that this is a direct address, **A** is a memory type (**F** for flag memory, **DBnn:D** for data memory), and **nnn.n** or **nnn** is the address location. Do not use a " **.** " character for real numbers. For an array, define only the first element in the array.

If you use this option, the address must be in the reserved memory area, which you specify in the Compiler Control File. For Series 505 controllers, G, VMM, and VME memory types cannot be reserved. For more information about using reserved memory, see the appendix on Direct Memory Addressing in the *SIMATIC APT Programming Reference (Graphics/Math) Manual.*

You cannot edit certain fields if you have already marked a declaration for OSx (PCS) tag translation. These fields are listed in Table 8-1 on page 8-6. When you mark objects for translation for S5 controllers, you cannot assign the S5 memory types FW, FD, S, SW, SD, or RS to objects that must be translated.

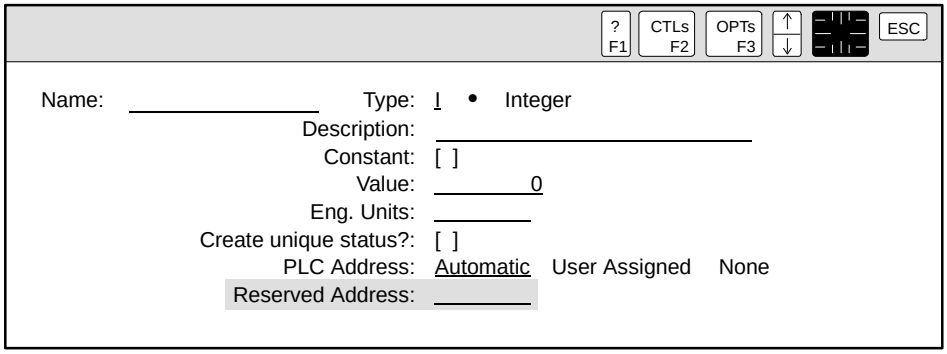
Table 8-1 OSx (PCS) Translation and Declaration Fields That Do Not Change

Declaration Type	Declaration Fields (Cannot be edited after marked for translation)
Integer (I)	Name, Type, Constant, Unique Status, PLC Address
Real (R)	Name, Type, Constant, Unique Status, PLC Address
Boolean (B)	Name, Type, Retentive, Unique Status, PLC Address
Flag (F)	Name, Type, Retentive, PLC Address
Text (T)	Name, Type, Number of Text Fields, Constant, PLC Address
Scaled Integer (SI)	Name, Type, Low Range, High Range, Unique Status, PLC Address
Integer Array (IA)	Name, Type, Constant, Number in Array, PLC Address
Real Array (RA)	Name, Type, Constant, Number in Array, PLC Address
Boolean Array (BA)	Name, Type, Retentive, Number in Array, PLC Address
DI10 (IX)	Name, Type, Retentive, PLC Address
DO10 Array (DX)	Name, Type, Retentive, PLC Address
Sequence Array (SA)	Name, Type, Number in Array, PLC Address
Shift Register Array(SR)	Name, Type, Number in Array, PLC Address
Text Array (TA)	Name, Type, Constant, Number in Array, PLC Address
Counter (CT)	Name, Type, PLC Address
Fast Timer (FT)	Name, Type, PLC Address
Slow Timer (ST)	Name, Type, PLC Address

8.2 Single-Value Declaration Types

Integer (I)

An integer value is a positive or negative number and cannot contain a decimal point. When you select I as the declaration type, this form appears:



Name: unique, symbolic name (12 characters maximum).

Type: I (code for Integer).

Description: optional (30 characters maximum).

Constant: X indicates value is a constant; blank field indicates value is a variable.

Value: integer between –32768 and +32767; specifies value of constant or initial value of variable.

Engineering Units: optional documentation field to describe the measurement units, such as kg, liter, or psi (8 characters maximum).

Create unique status: X indicates that a separate integer will be translated for an OSx (PCS) status attribute.

PLC Address: there are three controller addressing choices: Automatic, User Assigned, and None. (See descriptions below.) Automatic addressing is the default.

> **Automatic:** APT automatically assigns the controller address.
> **User Assigned**: Allows the user to specify a controller address.
> **None:** No controller address is assigned. (This is primarily intended for support of OSx non-networked tags.)

Reserved address: controller address you specify for the declaration, e.g., %V19 or %K19 (if constant) for a Series 505 controller, or %FW19 or %DBn:DW19 for S5. The types of Series 505 memory available for variable declarations are %V, %VMM, %VMS, %G, %Gx (where x is the application ID), %G; for constant declarations: %K, and %STW. The types of S5 memory available for variable declarations are %FW, %SW, %DBn:DW, and %DXn:DW; for constant declarations: %FW, %SW, %DBn:DW, %DXn:DW (where n is the block number), and %RSW.

Single-Value Declaration Types (continued)

Table 8-2 lists the extensions and commands for an integer declaration. The commands can be used in the parallel section of an SFC step or a Math block that generates RLL code for Series 505 controllers, or STL code for S5 controllers. Integer declarations and extensions can be used in a transition, in the Math section of an SFC step, or in Math Language statements in a CFB.

Table 8-2 Integer Extensions and Commands

Extensions		Command	
.STATUS	Integer value that will be translated to an OSx (PCS) status attribute. You must also write code in a Math block that changes the value of this extension.	INCREMENT (INC)*	Adds 1 to the current value of the variable.
		DECREMENT (DEC)*	Subtracts 1 from the current value of the variable.

* These integer commands execute only once each time that the program enters the step. These commands work only on variables defined as an integer in the Declaration Table. For Series 505 controllers, note that the INCREMENT and DECREMENT commands use less memory and execute faster than a math statement (i=i+1 or i=i–1). For S5 controllers, execution time and memory used are the same. However, for Series 505, the math step or CFB must be RLL code only.

Figure 8-2 illustrates the use of variables (int_1, int_2, int_3) in an SFC step and transition.

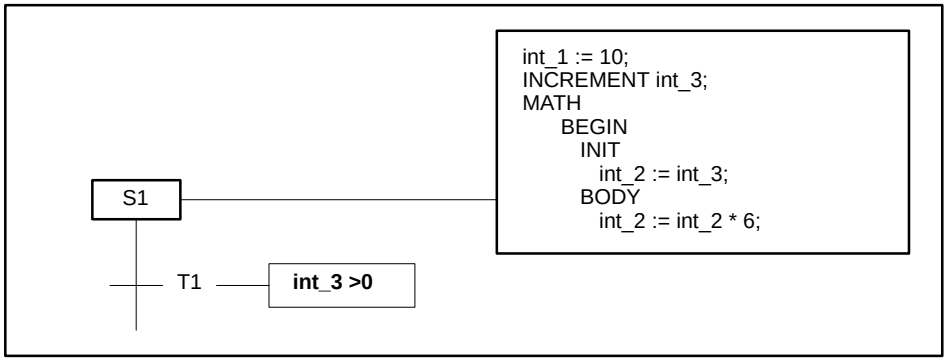
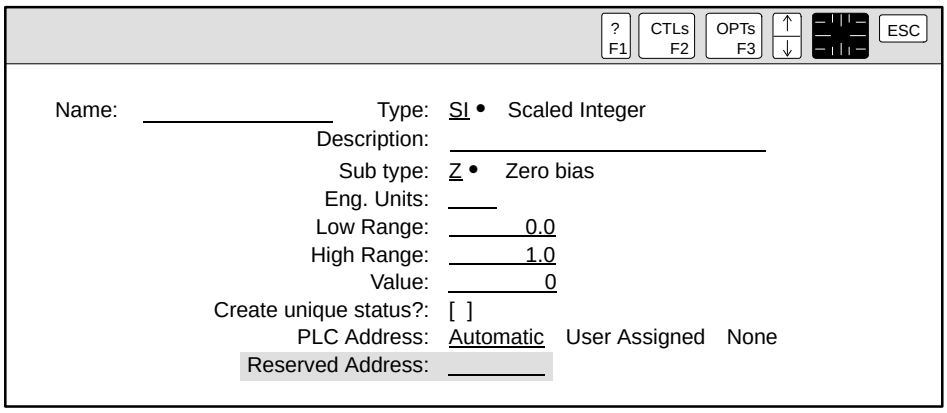


Figure 8-2 Using Integer Declarations in SFC Steps

Scaled Integer (SI)

A scaled integer value has the same properties as a normal integer with one exception: when marked for translation, a scaled integer is translated to an OSx (PCS) analog output. The high and low range, and engineering units are sent to OSx. In OSx, the value is converted to a real number. In APT, it stays an integer.



Name: unique, symbolic name (12 characters maximum).

Type: SI (code for Scaled Integer).

Description: optional (30 characters maximum).

Sub type: code (B, T, or Z) that represents the type of output to the field as listed below. Zero bias (Z) is the default.

(Series 505)

- **B** (Bipolar): –32000 to +32000
- **T** (Twenty-percent offset): +6400 to +32000
- **Z** (Zero bias): 0 to +32000

(S5)

- **B** (Bipolar): –1024 to +1024
- **Z** (Zero bias): 0 to 1024

Engineering Units: optional documentation field to describe the measurement units, such as kg, liter, or psi (8 characters maximum).

Single-Value Declaration Types (continued)

Low Range: real number that indicates low range of output in engineering units. Value must be less than high range.

High Range: real number that indicates high range of output in engineering units. Value must be greater than low range.

Value: integer between –32768 and +32767; specifies initial value of variable.

Create unique status: X indicates that a separate integer will be translated for an OSx (PCS) status attribute.

PLC Address: there are three controller addressing choices: Automatic, User Assigned, and None. Automatic addressing is the default.

> **Automatic:** APT automatically assigns the controller address.
> **User Assigned**: Allows the user to specify a controller address.
> **None:** No controller address is assigned. (This is primarily intended for support of OSx non-networked tags.)

Reserved address: controller address you specify for the declaration, e.g., %V19 or %K19 (if constant) for a Series 505 controller, and %FW19 or %DBn:DW19 for S5. The types of Series 505 memory available for this declaration are %V, %VMM, %VMS, %G, or %Gx (where x is the application ID). The types of S5 memory available for this declaration are %FW, %SW, %DBn:DW, and %DXn:DW (where n is the block number).

Table 8-3 lists the extensions and commands for the scaled integer. The commands can be used in the parallel section of an SFC step or a Math block that generates RLL code for Series 505 controllers, or STL code for S5 controllers. The scaled integer and extensions can be used in a transition, in the Math section of an SFC step, or in Math Language statements in a CFB.

Table 8-3 Scaled Integer Extensions and Commands

Extensions		Command	
.STATUS	Integer value that will be translated to an OSx (PCS) status attribute. You must also write code in a Math block that changes the value of this extension.	INCREMENT (INC)*	Adds 1 to the current value of the variable.
		DECREMENT (DEC)*	Subtracts 1 from the current value of the variable.
* These integer commands execute only once each time that the program enters the step. These commands work only on variables defined as an integer in the Declaration Table. Note that the INCREMENT and DECREMENT commands use less memory and execute faster than a math statement (i=i+1 or i=i-1). However, for Series 505, the math step or CFB must be RLL code only.			

Figure 8-3 illustrates the use of variables (int_1, int_2, int_3) in an SFC step and transition.

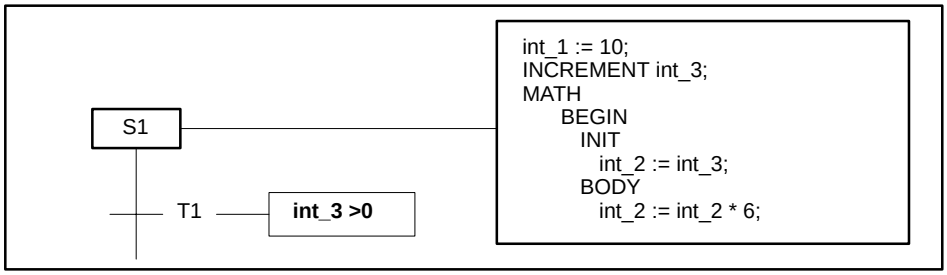
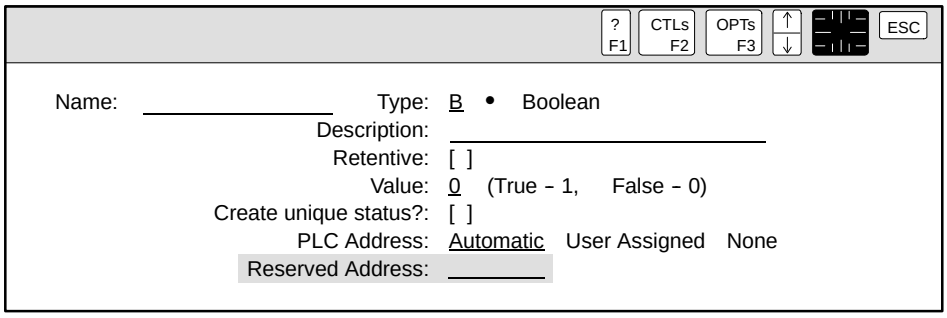


Figure 8-3 Using Scaled Integer Declarations in SFC Steps

Single-Value Declaration Types (continued)

Boolean (B)

A boolean value is either true (1) or false (0). When you select B as the declaration type, this form appears:



Name: unique, symbolic name (12 characters maximum).

Type: B (code for boolean).

Description: optional (30 characters maximum).

Retentive: For Series 505 controllers, X indicates that variable retains previous value in event of power failure; blank field indicates value is forced to false on return from a power failure. For S5 controllers, leave this field blank. All S5 boolean memory is retentive.

Value: 1 (true) or 0 (false); specifies initial value of variable.

Create unique status: X indicates that a separate integer will be translated for an OSx (PCS) status attribute.

PLC Address: there are three controller addressing choices: Automatic, User Assigned, and None. (See descriptions below.) Automatic addressing is the default.

> **Automatic:** APT automatically assigns the controller address.
> **User Assigned:** Allows the user to specify a controller address.
> **None:** No controller address is assigned. (This is primarily intended for support of OSx non-networked tags.)

Reserved address: controller address you specify for the declaration, e.g., %C100 for a Series 505 controller, and %F100.1 or %DBn:D100.1 for S5. The type of Series 505 memory available for this declaration is %C. The types of S5 memory available for this declaration are %F, %S, %DBn:D, and %DXn:D (where n is the block number).

For Series 505 controllers, the retentive range is in the upper quarter of each 1024 bits of C-Memory. All S5 boolean memory is retentive.

Table 8-4 lists the extension for boolean variables. There are no commands associated with boolean variables. A boolean variable and its extension can be used in a transition, in the Math section of an SFC step, or in Math Language statements in a CFB.

Table 8-4 Boolean Extensions

Extensions		Command
.STATUS	Integer value that will be translated to an OSx (PCS) status attribute. You must also write code in a Math block that changes the value of this extension.	There are no commands.

Figure 8-4 illustrates the use of variables (bool_1, bool_2, and bool_3) in an SFC step and transition.

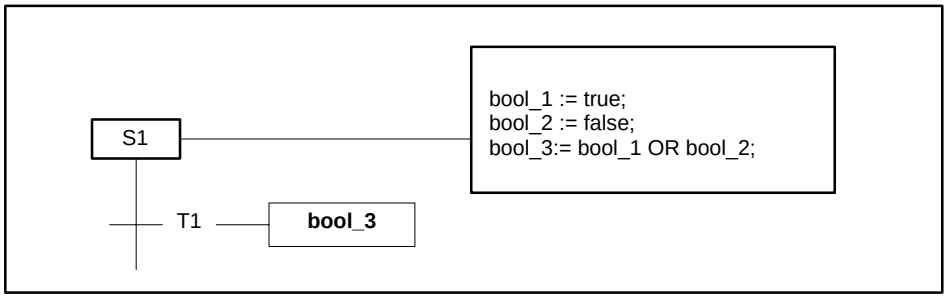
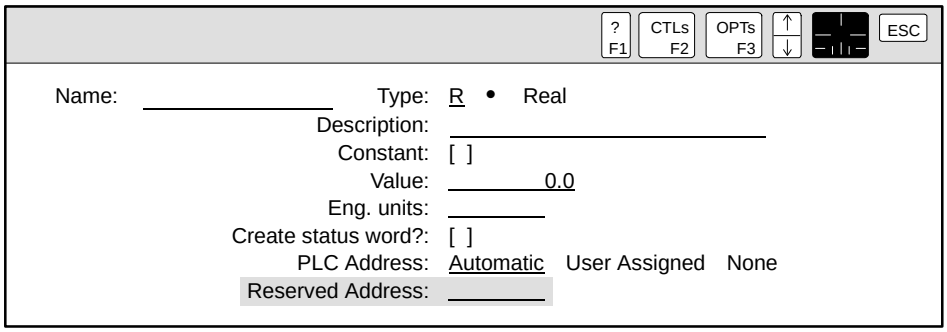


Figure 8-4 Using Boolean Declarations in SFC Steps

Single-Value Declaration Types (continued)

Real (R)

A real value is a floating point number that can contain a decimal point (3.0) or that can be expressed in scientific notation (1E12 or 1.0E12). When you select R as the declaration type, this form appears:



Name: unique, symbolic name (12 characters maximum).

Type: R (code for Real).

Description: optional (30 characters maximum).

Constant: X indicates that the value is a constant; blank field indicates that value is a variable.

Value: real number that specifies constant or initial value of variable; valid range for a real number for Series 505 controllers 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. Valid range for a real number for S5 controllers is $-1.701412\ E^{+38}$ to $+1.701412\ E^{+38}$; any value except 0.0 that falls between $-1.469368\ E^{-39}$ and $+1.469368\ E^{-39}$ generates a controller error.

Engineering Units: optional documentation field to describe the measurement units, such as kg, liter, or psi (8 characters maximum).

Create status word: X indicates that a status word will be created for OSx (PCS).

PLC Address: there are three controller addressing choices: Automatic, User Assigned, and None. (See descriptions below.) Automatic addressing is the default.

> **Automatic:** APT automatically assigns the controller address.
> **User Assigned:** Allows the user to specify a controller address.
> **None:** No controller address is assigned. (This is primarily intended for support of OSx non-networked tags.)

Reserved address: controller address you specify for the declaration, e.g., %V19 or %K19 (if constant) for a Series 505 controller, and %FW19 or %DBn:DW19 for S5. The types of Series 505 memory available for this declaration are %V, %VMM, %VMS, %G, and %Gx (where x is the application ID), and %K. The types of S5 memory available for this declaration are %FD, %SD, %DBn:DD, and %DXn:DD (where n is the block number).

NOTE: Do not enter the "." after the %V19 for Series 505 controllers, even though real numbers are expressed that way in the controller. %V19 automatically reserves %V19 and %V20. Be sure to reserve two V-Memory locations in the Compiler Control File for each real number.

Do not enter the "." after the %FD19 for S5 controllers since the "D" indicates that the number is real. If you are using flag memory, reserve four flag bytes for each real number. If you are using data word memory, reserve two data words for each real number.

Table 8-5 lists the extensions for real declarations. There are no commands for a real declaration. Real declarations and extensions can be used in a transition, in the Math section of an SFC step, or in Math Language statements in a CFB.

Table 8-5 Real Extensions

Extensions		Command
.STATUS	Integer value that will be translated to a OSx (PCS) status attribute. You must also write code in a Math block that changes the value of this extension.	There are no commands.

Figure 8-5 illustrates the use of variables (real_1, real_2, and real_3) in an SFC step.

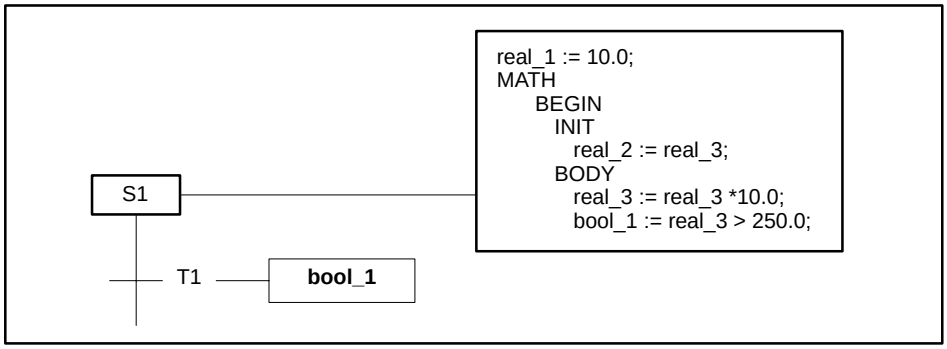


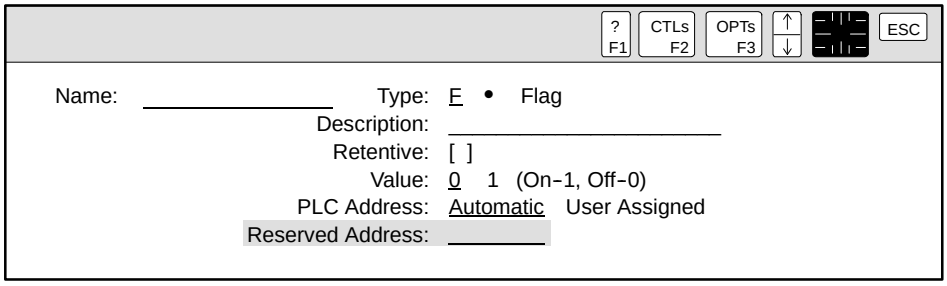
Figure 8-5 Using Real Declarations in SFC Steps

Single-Value Declaration Types (continued)

APT Flag (F)

An APT flag value is either on (1 or true) or off (0 or false). When you select F as the declaration type, the form below appears.

APT flags differ from boolean variables. All references to an APT flag are logically connected, and the APT flag state is set in one place in the compiled program. References to a boolean variable can appear any number of places in the compiled code.



Name: unique, symbolic name (12 characters maximum).

Type: F (code for APT Flag).

Description: optional (30 characters maximum).

Retentive: For Series 505 controllers, X indicates that variable retains previous value in event of power failure; blank field indicates value is set to false on return from a power failure. All APT flags for S5 controllers are retentive; leave this field blank.

Value: 1 (on) or 0 (off); specifies initial value of variable.

PLC Address: there are two controller addressing choices: Automatic and User Assigned. (See descriptions below.) Automatic addressing is the default.

> **Automatic:** APT automatically assigns the controller address.
> **User Assigned**: Allows the user to specify a controller address. (There is no None option for APT flags.)

Reserved address: controller address you specify for the declaration, e.g., %C100 for a Series 505 controller, and %F100.1 or %DBn:D100.0 for S5. The type of Series 505 memory available for this declaration is %C. The types of S5 memory available for this declaration are %F, %S, %DBn:D, and %DXn:D (where n is the block number).

For Series 505 controllers, the retentive range is in the upper quarter of each 1024 bits of C-Memory. All S5 boolean memory is retentive.

Table 8-6 lists the commands for the APT declaration flags. These commands can be used in the parallel section of an SFC step or a Math block that generates RLL or STL code. APT flags can be used in a transition, in the Math section of an SFC step, or in Math Language statements in a CFB. When used in an assignment statement, APT flags can only be used on the right side of the assignment.

Table 8-6 Flag Commands

Command	Action
LATCH	Sets APT flag value to on (true) when the SFC step is active or the CFB is enabled. Flag remains on, even if the step or CFB is disabled, until a CLEAR sets it to off (false).
ON	Sets APT flag value to on (true) only as long as the SFC step is active or the CFB is enabled; flag value is off (false) when the step or CFB is disabled. ON should not be used in Sampled or Event CFBs.
CLEAR	Sets APT flag value to off (false) when the SFC step or CFB is enabled. Flag remains off, even if the step or CFB is disabled, until a LATCH or ON sets it to on. CLEAR takes precedence over ON or LATCH.

The APT flag commands are identical to the Math procedures LATCH, ON, and CLEAR.

For Series 505 controllers, APT flag procedures are RLL-only and cannot be used in the associated math text for CFBs such as the PID or Analog Alarm. S5 controllers can use APT flag procedures in associated math.

- Do not assign an initial value to an APT flag when you intend to use the APT flag only with an ON procedure. The value of the APT flag is always zero outside the step or CFB in which the ON procedure is located.
- Do not use the ON procedure in sampled or event Math CFBs because the APT flag is on for only one controller scan.

An example of how APT flags are used in a program is shown in Figure 8-6.

Single-Value Declaration Types (continued)

Figure 8-6 shows the values and use of APT flag variables (flag_1 and flag_2) in SFC steps and transitions.

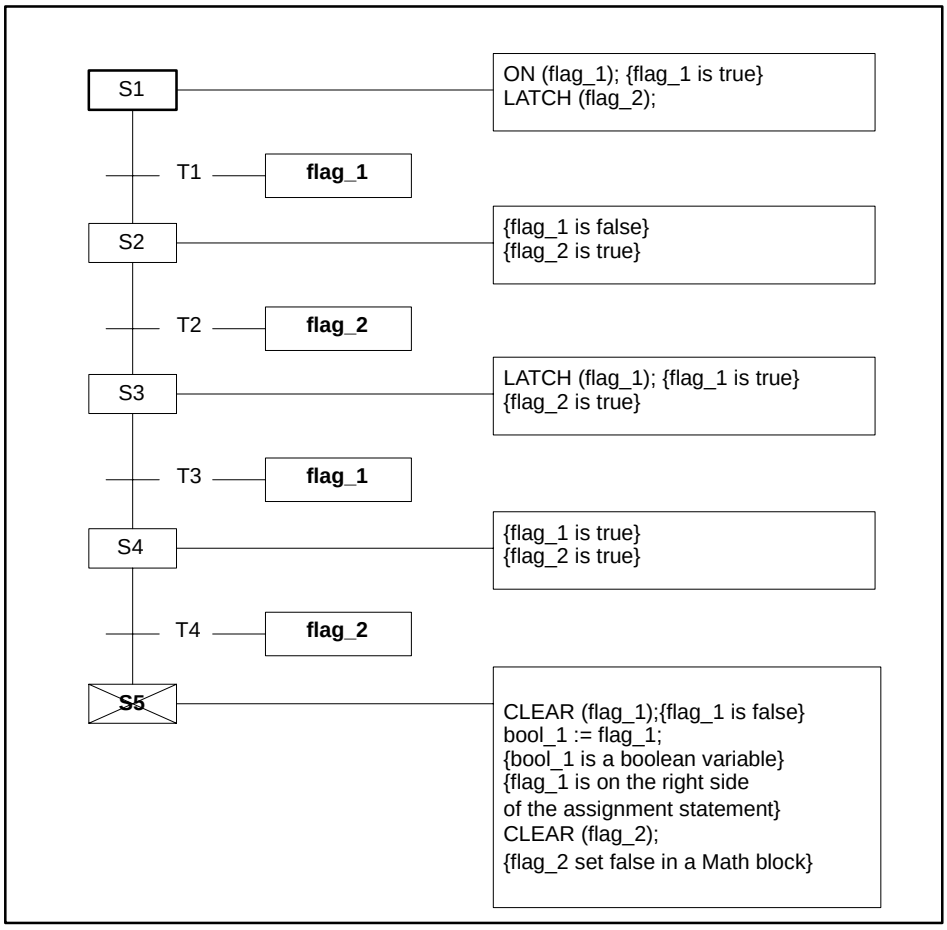
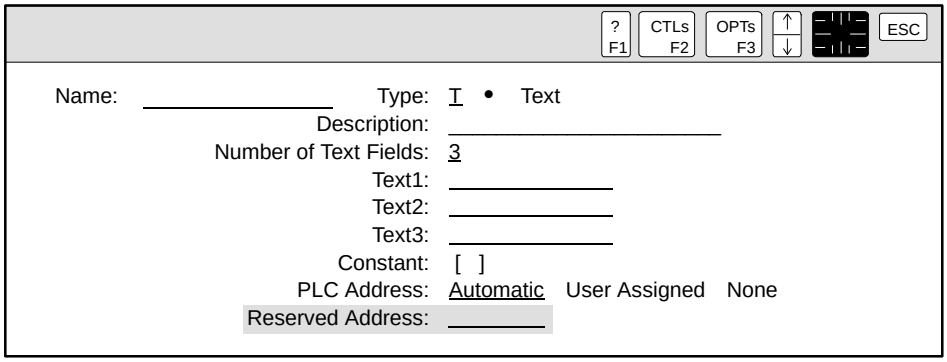


Figure 8-6 Using APT Flag Declarations in SFC Steps

Text (T)

A text variable is a string variable that can contain characters. The text variable can be 1, 2, or 3 fields long. Each field can handle up to 30 characters: the longest text variable can contain 90 characters. Each field requires 15 memory locations.

The text variable exists primarily for OSx (PCS), so that APT can have a match with the OSx text tags. When you select text as the declaration type, the following form appears:



Name: unique, symbolic name (12 characters maximum).

Type: T (code for Text).

Description: optional (30 characters maximum).

Number of Text Fields: Indicates number of 30-character fields (1, 2, or 3 fields; default is 1).

Text1, Text2, Text3: Actual text is entered here. Any printable character is allowed (30 characters maximum per field).

Constant: X indicates that the text is constant; a blank field indicates that the text is variable.

PLC Address: there are three controller addressing choices: Automatic, User Assigned, and None. (See descriptions below.) Automatic addressing is the default.

> **Automatic:** APT automatically assigns the controller address.
> **User Assigned:** Allows the user to specify a controller address.
> **None:** No controller address is assigned. (This is primarily intended for support of OSx non-networked tags.)

Single-Value Declaration Types (continued)

Reserved address: controller address you specify for the declaration, e.g., %V19 or %K19 (if constant) for a Series 505 controller, or %DBn:DW19 for S5. For each text field, you must allocate 15 memory locations in the Compiler Control File if Series 505 V- or K-Memory is used. You must allocate 30 memory locations for each text field if S5 flag memory is used. The types of Series 505 memory available for this declaration are %V, %VMM, %VMS, %G, %Gx (where x is the application ID), and %K. The types of S5 memory available for this declarations are %DBn:DW, and %DXn:DW(where n is the block number).

Table 8-7 lists the extensions available for the text declaration. The text declaration and its extensions can be used in the Math section of an SFC step or in Math language statements in a CFB, but not in a transition.

Table 8-7 Text Extensions

Extensions		Command
.TEXT1	30 characters of text	There are no commands.
.TEXT2	30 characters of text	
.TEXT3	30 characters of text	
Note: The .status extension is not available for text tags.		

The text variable exists in APT so that you can configure an OSx (PCS) text tag. Only simple assignments can be done with text variables.

- You can assign character strings to text variables only by using the declaration table.

NOTE: You cannot assign a text variable to a string that is enclosed in quotes. A statement such as **alarm_1 := "High alarm";** is not allowed. You must assign the string either in the declaration table or in an assignment statement to another text variable that was created in the declaration table.

- You can reassign one text variable to another text variable if they have the same number of text fields. A statement such as **alarm_1 := alarm_2**; is permitted, as long as alarm_1 and alarm_2 have the same number of text fields assigned to them.
- You can use the value 1, 2, or 3 in brackets to specify a particular text field, or you can specify the text field by using the proper extension:

  alarm_1[1] := alarm_2[2];

  or

  alarm_1.text1 := alarm_2.text2;

NOTE: You cannot use expressions as indices for text fields. To specify a text field, use either the appropriate integer value, enclosed in brackets, or else the correct extension, as shown above.

Figure 8-7 illustrates the use of text variables (alarm_1, alarm_2, alarm_3) in an SFC step. Notice that you can either assign the entire text variable to another text variable, or else assign individual fields selectively.

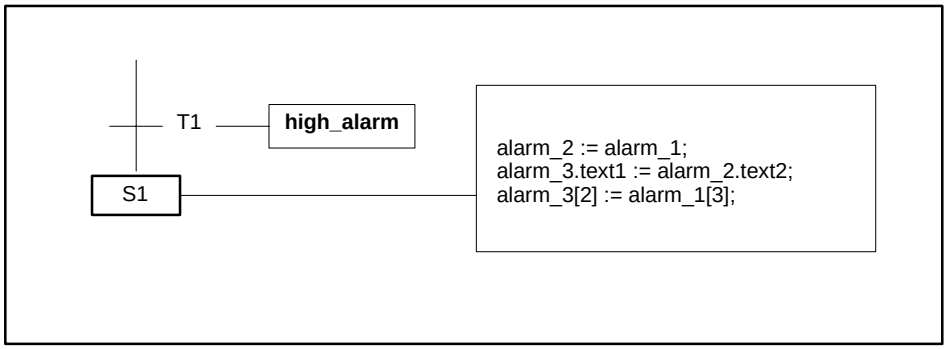


Figure 8-7 Using Text Declarations in SFC Steps

8.3 Arrays

Using Arrays

An array is an indexed collection of values that you can reference as a whole or as individual elements.

To assign the value of all elements in an array to the corresponding elements of another array, use the array names in an assignment statement: **array_1 := array_2;**. The two arrays must be the same size.

Use an integer in brackets following the array name to specify an element in the array. To assign values to a single element, use an assignment statement such as **array_1[1] := 10;**

For Series 505 controllers, the index in a boolean array can be either a literal value or an integer variable. For S5 controllers, the index in a boolean array can only be a literal value. Some Series 505 controllers support indexing arrays with expressions. S5 controllers do not support indexing boolean arrays with expressions. For Series 505 controllers, you can index a boolean element with an expression or an integer variable, such as [I42], only in SFPGM and only if the controller release supports this SFPGM operation. Refer to Table 1-9, in Chapter 1, for features specific to the controller release.

For both Series 505 and S5 controllers, the index in a real or integer array can be a literal value, an expression or an integer variable. For Series 505 controllers, if you use an expression or integer variable for the index of a real or integer array, then calculations on the array are done in SFPGM or RLL, based on the firmware of the controller. Refer to Table 1-9, in Chapter 1, for features specific to the controller release.

The index in a text array must be a literal value (1,2,3, etc.). A text array cannot use an integer variable or an expression as an index.

Arrays can be used in assignment statements in the parallel section of an SFC step or in the Math section of an SFC step or CFB. Figure 8-8 illustrates the use of integer arrays (iarr_1 and iarr_2) in an SFC step and transition.

When you create user-defined subroutines, do not use an expression for the index of an array as a parameter; use a variable instead. For Series 505 controllers, you can store the intermediate results of an expression in a controller T-Memory location before calling the subroutine ($\%T_x$, where x = 10–16). For S5 controllers, you can store the intermediate result of an expression in a temporary variable before calling the subroutine.

Another alternative to using an expression for the index of an array is to use a variable instead. Refer to Table 1-9, in Chapter 1, for APT features supported by controller models.

Figure 8-8 illustrates the use of an array (iarr_1) in an SFC step and transition.

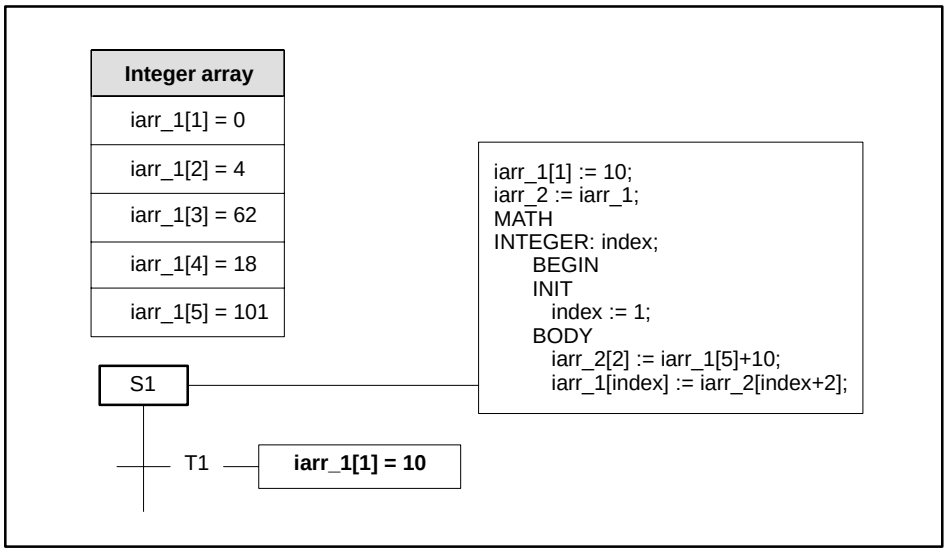


Figure 8-8 Using Arrays in SFC Steps

⚠ WARNING

Programs generated by APT do not check the upper/lower limits of an array. It is possible to index an array out of its configured boundaries.

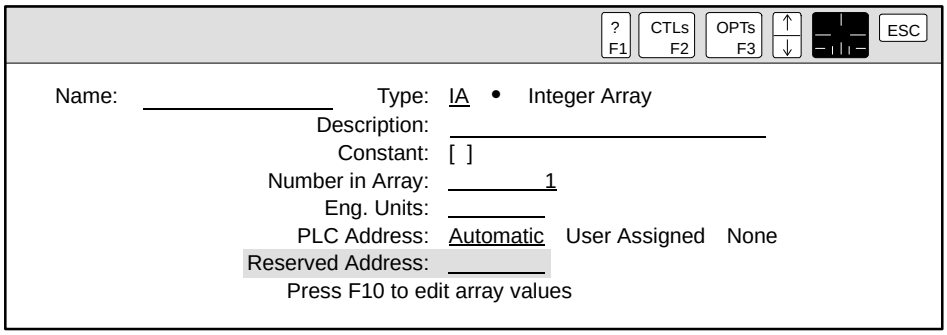
Indexing outside the boundaries of an array may cause your application to execute in an unpredictable manner that could result in death or serious injury and/or property damage.

Take care to design your APT program in a manner that makes it impossible to index outside of array boundaries.

Arrays (continued)

Integer Array (IA)

An integer array is an indexed collection of integers. When you select IA as the declaration type, the form below appears:



Name: unique, symbolic name (12 characters maximum).

Type: IA (code for Integer Array).

Description: optional (30 characters maximum).

Constant: X indicates that each value is a constant; blank field indicates that each value is a variable.

Number in Array: indicates number of elements in the array; default is 1. For Series 505 controllers: integer arrays can have up to 32767 elements. For S5 controllers: integer arrays can have up to 256 elements.

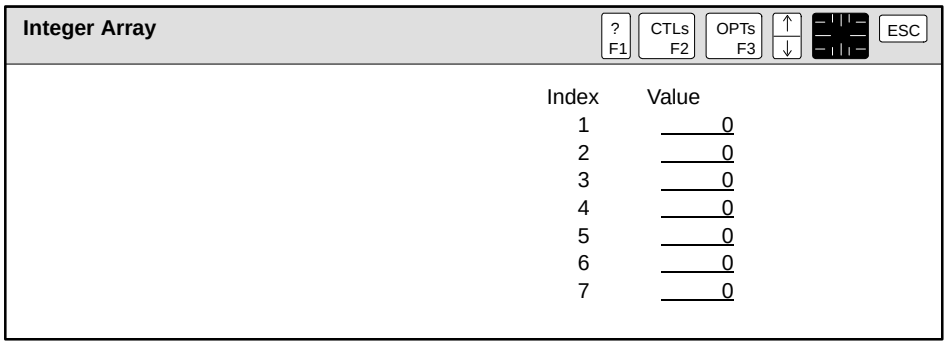
Engineering Units: optional documentation field to describe the measurement units, such as kg, liter, or psi (8 characters maximum).

PLC Address: there are three controller addressing choices: Automatic, User Assigned, and None. (See descriptions below.) Automatic addressing is the default.

> **Automatic:** APT automatically assigns the controller address.
> **User Assigned:** Allows the user to specify a controller address.
> **None:** No controller address is assigned. (This is primarily intended for support of OSx non-networked tags.)

Reserved address: starting controller address you specify for the declaration, e.g., %V19 or %K19 (if constant) for a Series 505 controller, or %DBn:DW19 for S5. You must reserve sufficient memory, based on the array size, in the Compiler Control File. The types of Series 505 memory available for this declaration are %V, %VMM, %VMS, %G, %Gx (where x is the application ID), and %K. The types of S5 memory available for this declaration are %DBn:DW and %DXn:DW (where n is the block number).

To specify values in the array, press F10 . If you specify 7 as the number of elements in the array, the dialogue window below appears:



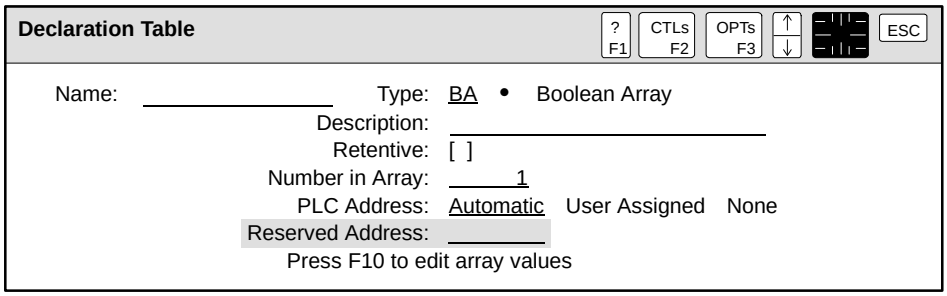
Index: location of value in array

Value: integer between –32768 and 32767 for both Series 505 and S5 controllers; specifies constant value or initial value of array element; default is 0.

Arrays (continued)

Boolean Array (BA)

A boolean array is an indexed collection of boolean values that are either true or false. When you select BA as the declaration type, the form below appears:



Name: unique, symbolic name (12 characters maximum).

Type: BA (code for Boolean Array).

Description: optional (30 characters maximum).

Retentive: For Series 505 controllers, X indicates that each element retains its previous value in the event of power failure; a blank field indicates each value is forced to false on return from a power failure. All booleans are retentive in S5 controllers; leave this field blank.

Number in Array: indicates number of elements in the array; default is 1. For Series 505 controllers, retentive boolean arrays can have 256 elements; non-retentive arrays can have 1024 elements. For S5 controllers, boolean arrays can have 4096 elements.

PLC Address: there are three controller addressing choices: Automatic, User Assigned, and None. (See descriptions below.) Automatic addressing is the default.

> **Automatic:** APT automatically assigns the controller address.
> **User Assigned:** Allows the user to specify a controller address.
> **None:** No controller address is assigned. (This is primarily intended for support of OSx non-networked tags.)

Reserved address: starting controller address you specify for the declaration, e.g., %C100 for a Series 505 controller, or %F100.1 for S5. You must reserve enough memory for the size of the array in the Compiler Control File. For S5 controllers, recall that D memory holds 16 booleans per word, and F and S memory hold 8 booleans per word. The type of Series 505 memory available for this declaration is %C memory. The types of S5 memory available for this declaration are %F, %DBn:D, %DXn:D (where n is the block number), and %S.

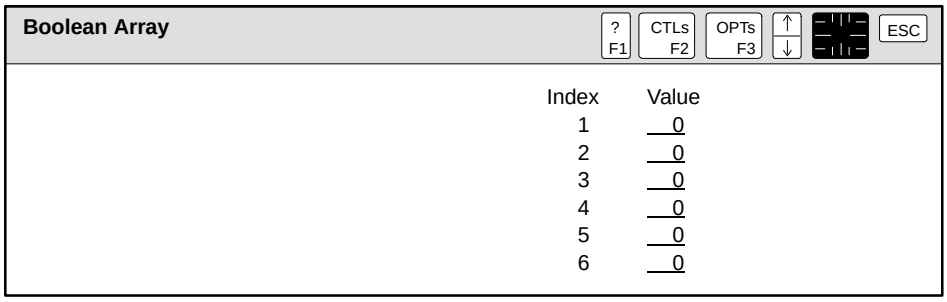
Reserved address, continued: For Series 505 controllers, if you have a non-retentive boolean array larger than 768 elements on the 560T, 565P, 545, 555, or 575, then you must use the extended local C-Memory that starts at memory location %C10241. In order to use extended (local) C-Memory, you must reserve all the lower non-retentive memory plus the size of the array in the Compiler Control File.

For instance, if you have a boolean array of 1024 elements, you need to reserve (768*8) + 1024 = 7168 for the 545, 555, and the 575. For the 560T and the 565P, you need to reserve (768*(number of RCC cards*2)) + 1024. In this case, your reserved C-Memory address would be %C10241.

Refer to the Control Relays Permitted table in the *SIMATIC TI505 Programming Reference User Manual* for C-Memory ranges.

On the 560 and 565, non-retentive boolean arrays larger than 768 cannot be given a reserved address, but an automatic address can be assigned, which will be given an overflow C-Memory address.

To specify values in the array, press F10 . If you specified 6 as the number of elements in the array, the dialogue window below would appear:



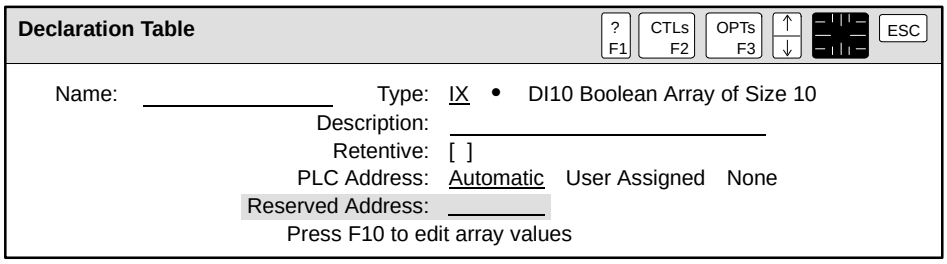
Index: location of value in array.

Value: 0 (false) or 1 (true); specifies value of element in the array; default is 0.

Arrays (continued)

DI10 Array (IX)

A DI10 array is a boolean array with a length of ten that is translated to OSx (PCS) as a DI10 tag type.



Name: unique, symbolic name (12 characters maximum).

Type: IX (code for DI10 Array).

Description: optional (30 characters maximum).

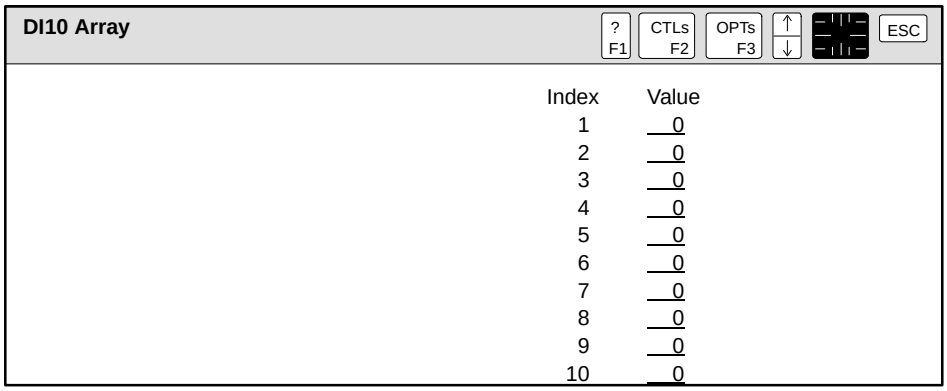
Retentive: For Series 505 controllers, X indicates that each element retains its previous value in the event of power failure; a blank field indicates each value is forced to false on return from a power failure. All booleans are retentive in S5 controllers; leave this field blank.

PLC Address: there are three controller addressing choices: Automatic, User Assigned, and None. (See descriptions below.) Automatic addressing is the default.

> **Automatic:** APT automatically assigns the controller address.
> **User Assigned:** Allows the user to specify a controller address.
> **None:** No controller address is assigned. (This is primarily intended for support of OSx non-networked tags.)

Reserved address: starting controller address you specify for the declaration, e.g., %C100 for a Series 505 controller, or %F100.1 for S5. You must reserve 10 C locations in the Compiler Control File for Series 505 controllers, or 2 flag bytes for S5 controllers. The type of Series 505 memory available for this declaration is %C memory. The types of S5 memory available for this declaration are %F, %DBn:D, %DXn:D (where n is the block number), and %S.

To specify values in the array, press F10 and the dialogue window below appears:



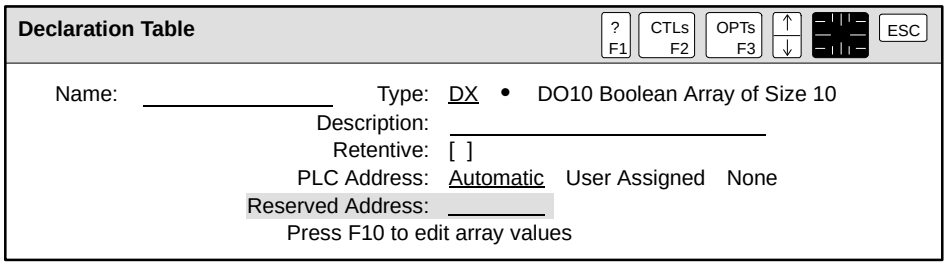
Index: location of value in array

Value: 0 (false) or 1 (true); specifies value of element in the array; default is 0.

Arrays (continued)

DO10 Array (DX)

A DO10 array is a boolean array with a length of ten that is translated to OSx (PCS) as a DO10 tag type.



Name: unique, symbolic name (12 characters maximum).

Type: DX (code for DO10 Array).

Description: optional (30 characters maximum).

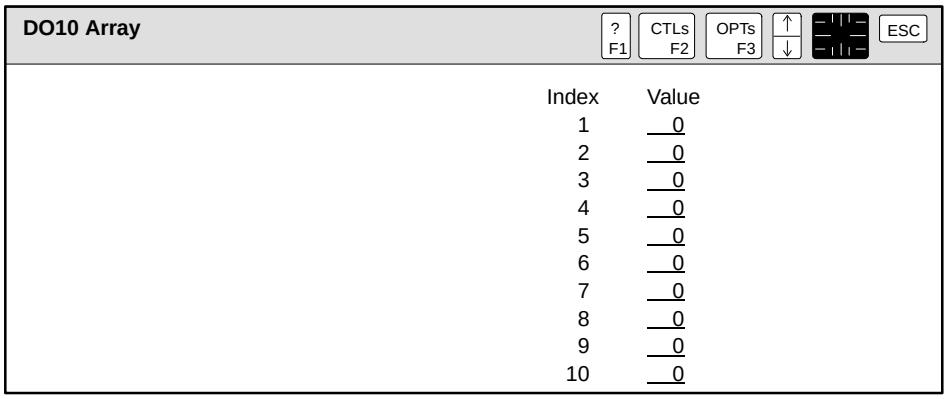
Retentive: For Series 505, X indicates that each element retains its previous value in the event of power failure; a blank field indicates each value is forced to false on return from a power failure. All booleans are retentive in S5 controllers.

PLC Address: there are three controller addressing choices: Automatic, User Assigned, and None. (See descriptions below.) Automatic addressing is the default.

> **Automatic:** APT automatically assigns the controller address.
> **User Assigned:** Allows the user to specify a controller address.
> **None:** No controller address is assigned. (This is primarily intended for support of OSx non-networked tags.)

Reserved address: starting controller address you specify for the declaration, e.g., %C100 for a Series 505 controller, or %F100.1 for S5. You must reserve 10 C locations in the Compiler Control File for Series 505 controllers, or 2 flag bytes for S5 controllers. The type of Series 505 memory available for this declaration is %C memory. The types of S5 memory available for this declaration are %F, %DBn:D, %DXn:D, and %S (where n is the block number).

To specify values in the array, press F10 and the dialogue window below appears:



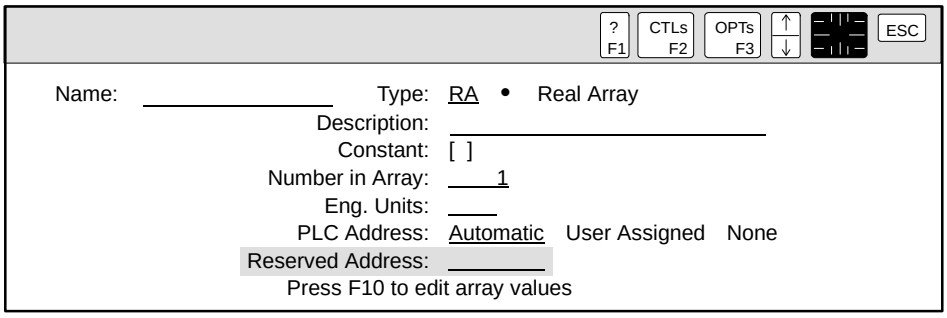
Index: location of value in array

Value: 0 (false) or 1 (true); specifies value of element in the array; default is 0.

Arrays (continued)

Real Array (RA)

A real array is an indexed collection of real, or floating point, values. When you select RA as the declaration type, the form below appears:



Name: unique, symbolic name (12 characters maximum).

Type: RA (code for Real Array).

Description: optional (30 characters maximum).

Constant: X indicates that each value is a constant; blank field indicates that each value is a variable.

Number in Array: indicates number of elements in the array; default is 1. For Series 505 controllers: real arrays can have up to 16382 elements. For S5 controllers: real arrays can have up to 128 elements.

Engineering Units: optional documentation field to describe the measurement units, such as kg, liter, or psi (8 characters maximum).

PLC Address: there are three controller addressing choices: Automatic, User Assigned, and None. (See descriptions below.) Automatic addressing is the default.

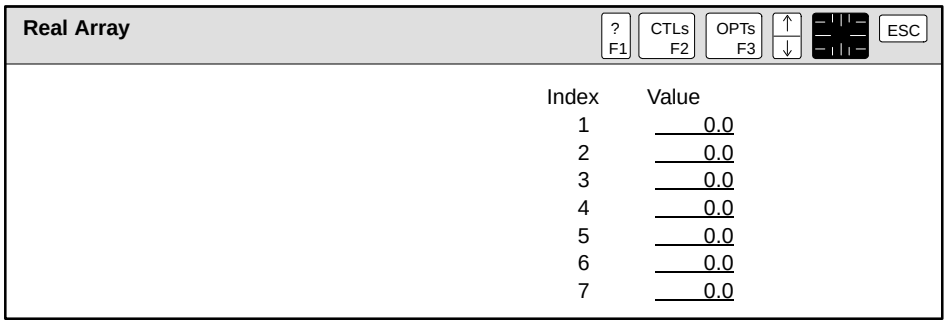
> **Automatic:** APT automatically assigns the controller address.
> **User Assigned:** Allows the user to specify a controller address.
> **None:** No controller address is assigned. (This is primarily intended for support of OSx non-networked tags.)

Reserved address: starting controller address you specify for the declaration, e.g., %V19 (if variable) or %K19 (if constant) for a Series 505 controller, or %DBn:DD19 for S5. You must reserve enough memory in the Compiler Control File for the size of the array. The types of Series 505 memory available for this declaration are %V, %VMM, %VMS, %G, %Gx (where x is the application ID), and %K. The types of S5 memory available for this declaration are %DBn:DD, and %DXn:DD (where n is the block number).

NOTE: For Series 505 controllers, do not enter the "." after the %V19, even though real numbers are expressed that way in the controller. %V19 will reserve %V19 and %V20. Be sure to reserve two V locations in the Compiler Control File for each real number.

Do not enter the "." after the %DD19 (although MAITT and Debug require it when you use these utilities). %DBn:DD19 reserves %DW19 and %DW20. Be sure to reserve two data words in the Compiler Control File for each real number.

To specify values in the array, press F10 . If you specify 7 as the number of elements in the array, the dialogue window below appears:



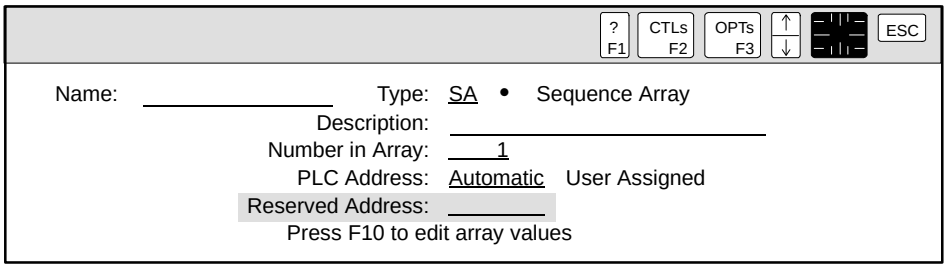
Index: location of value in array.

Value: real number that specifies constant or initial value of variable. For Series 505 controllers, 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. For S5 controllers, the valid range for a real number is $-1.701412\ E^{+38}$ to $1.701412\ E^{+38}$. Any value except 0.0 that falls between $-1.469368\ E^{-39}$ and $+1.469368\ E^{-39}$ generates a controller error.

Arrays (continued)

Sequence Array (SA)

A sequence array is a special array of integers that you can use for sequence control. When you select SA as the declaration type, the form below appears:



Name: unique, symbolic name (12 characters maximum).

Type: SA (code for Sequence Array).

Description: optional (30 characters maximum).

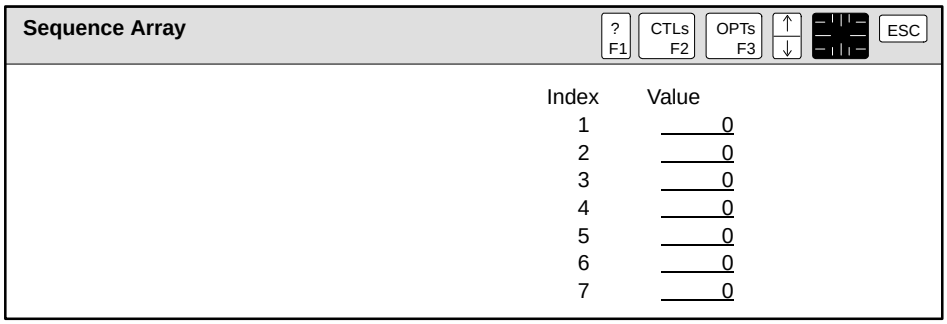
Number in Array: indicates number of elements in the array; default is 1. Both Series 505 and S5 controllers can have up to 256 elements.

PLC Address: there are two controller addressing choices: Automatic and User Assigned. (See descriptions below.) Automatic addressing is the default.

> **Automatic:** APT automatically assigns the controller address.
> **User Assigned:** Allows the user to specify a controller address. (There is no None option for sequence arrays.)

Reserved address: starting controller address you specify for the declaration, e.g., %V19 for a Series 505 controller, or %DBn:DW19 for S5. You must reserve enough memory for the size of the array in the Compiler Control File. The type of Series 505 memory available for this declaration is %V memory. The types of S5 memory available for this declaration are %DBn:DW and %DXn:DW (where n is the block number).

To specify values in the array, press F10 . If you specify 7 as the number of elements in the array, the dialogue window below appears:



Index: location of value in array

Value: integer between –32768 and 32767 for both Series 505 and S5 controllers; specifies initial value of array element; default is 0.

NOTE: For Series 505 controllers, ensure that the end of your sequence array does not exceed memory location V32767, whether the address is automatic or user-assigned. The controller generates an error with the RLL MWFT instruction when you use V-memory locations greater than 32767. If you are using automatic addressing, switch to user-assigned addressing and reserve enough V-memory to store the sequence array. If you are using user-assigned addressing, put your sequence array in lower V-memory.

Arrays (continued)

Table 8-8 lists the extensions and commands for the sequence array. The commands can be used in the parallel section of an SFC step or a Math block that generates RLL or STL code. Elements can be used in a transition, in the Math section of an SFC step, or in Math Language statements in a CFB.

Table 8-8 SA Extensions and Commands

Extensions		Command	
.PTR	Integer value that indicates current position in the array.	RESET[1]	Sets pointer to zero.
.PNTR	Integer value that indicates one greater than the current position in the array.	ADVANCE[1]	Moves pointer to next element and puts value in **.IOUT**.
.IOUT	Integer value advanced from array.		
.EMPTY	Boolean that indicates that the pointer has reached last element in the array.		

1 These sequence array commands execute only once each time that the program enters the step. These commands work only on variables defined as a sequence array in the Declaration Table.

After a reset, the **.PNTR** extension contains the value of the starting address of the array. After each advance of the array, this variable is incremented by one. You can change the value of the **.PNTR** extension as long as the value remains inside the correct range of addresses allocated for the array. The value of the **.PNTR** extension is one greater than the value of the **.PTR** extension.

An example of how to use a sequence array in an SFC is shown in Figure 8-9.

Figure 8-9 illustrates the use of a sequence array (sarr_1) in SFC steps and transitions. Only one ADVANCE command should be placed in any one step, and ADVANCE commands should not appear in consecutive steps. In order to advance in subsequent steps, you must put steps between each advance step.

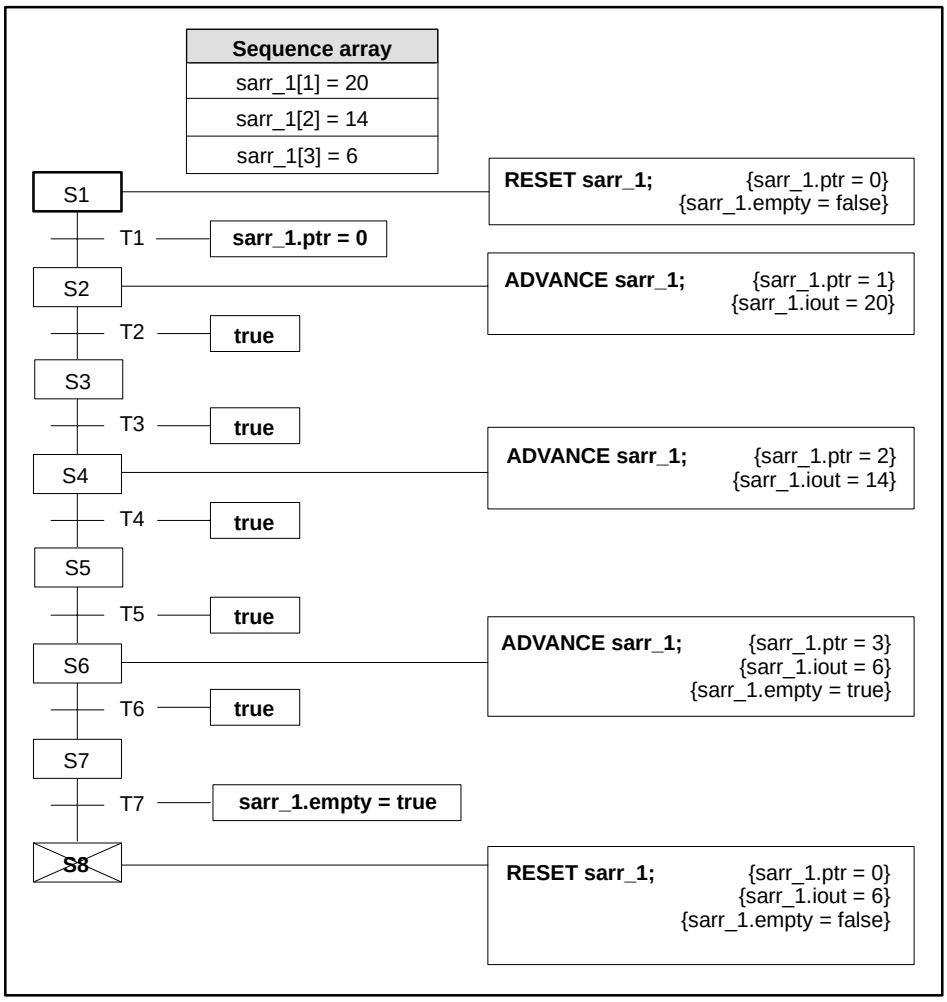
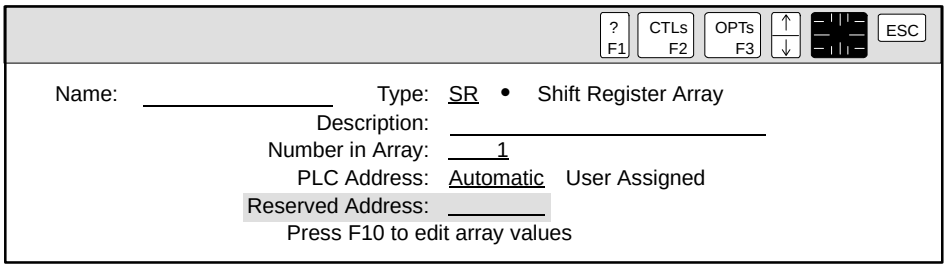


Figure 8-9 Using Sequence Arrays in SFC Steps

Arrays (continued)

Shift Register Array (SR)

A shift register array is a special array of integers that you can use to shift the values in the array down 1 element and insert a value into the first element. When you select SR as the declaration type, the form below appears.



Name: unique, symbolic name (12 characters maximum).

Type: SR (code for Shift Register Array).

Description: optional (30 characters maximum).

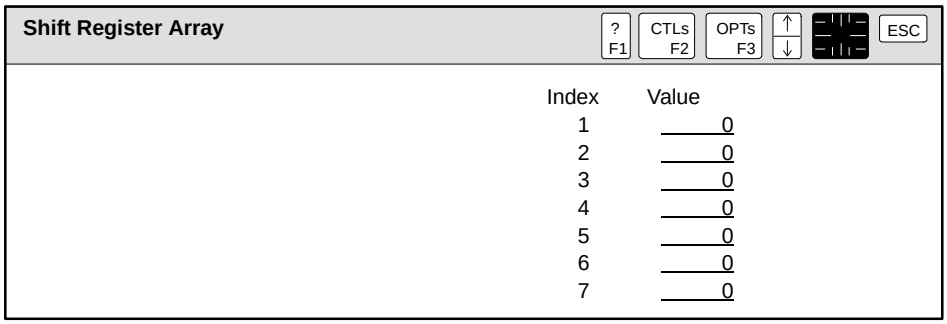
Number in Array: indicates number of elements in the array; default is 1. For Series 505 controllers: shift register arrays can have up to 1023 elements. For S5 controllers: shift register arrays can have up to 256 elements.

PLC Address: there are two controller addressing choices: Automatic and User Assigned. (See descriptions below.) Automatic addressing is the default.

> **Automatic:** APT automatically assigns the controller address.
> **User Assigned:** Allows the user to specify a controller address. (There is no None option for shift register arrays.)

Reserved address: starting controller address you specify for the declaration, e.g., %V19 for a Series 505 controller, or %DBn:DW19 for S5. You must reserve enough memory in the Compiler Control File for the size of the array. The type of Series 505 memory available for this declaration is %V. The types of S5 memory available for this declaration are %DBn:DW and %DXn:DW (where n is the block number).

To specify values in the array, press F10 . If you specify 7 as the number of elements in the array, the dialogue window below appears.



Index: location of value in array.

Value: integer between –32768 and 32767 for both Series 505 and S5 controllers; specifies initial value of array element; default is 0.

Arrays (continued)

Table 8-9 lists the extensions and commands for the shift register array. The commands can be used in the parallel section of an SFC step or a Math block that generates RLL or STL code. Elements can be used in a transition, in the Math section of an SFC step, or in Math Language statements in a CFB. An example of how to use a shift register array in an SFC is shown in Figure 8-10.

Table 8-9 SR Extensions and Commands

Extensions		Command	
.PV	Integer value that gets inserted in the first element in the array after a SHIFT (read-only integer).	SHIFT[1]	Shift the values in the array down 1 element and insert the **.PV** value into the first element.
.RESET [2]	Becomes false to clear the shift register data. The **.RESET** extension becomes true to indicate that the shift register array can be activated if the **.ENABL** extension is set to true. (read/write boolean).	SET_RESET[1]	Ready the shift register.
		CLEAR_RESET[1]	Clear the shift register data.
.ENABL	Indicates the shift register array has been activated (read/write boolean).	ENABLE[1]	Enable the shift register.
		DISABLE[1]	Disable the shift register.

1 These shift register array commands execute only once each time that the program enters the step. These commands work only on variables defined as a shift register array in the Declaration Table.

2 The shift register needs to be set and enabled before it can shift data, i.e., **.RESET** extension and **.ENABL** extension must be set to true with the SET_RESET and ENABLE commands.

Figure 8-10 illustrates the use of a shift register array (sarr_1) in SFC steps and transitions. Only one SHIFT command should be placed in any one step, and SHIFT commands should not appear in consecutive steps.

When you are finished with the shift register, you can clear it, using the **CLEAR_RESET** *shift_register_name* command. This sets all values in the shift register array to zero.

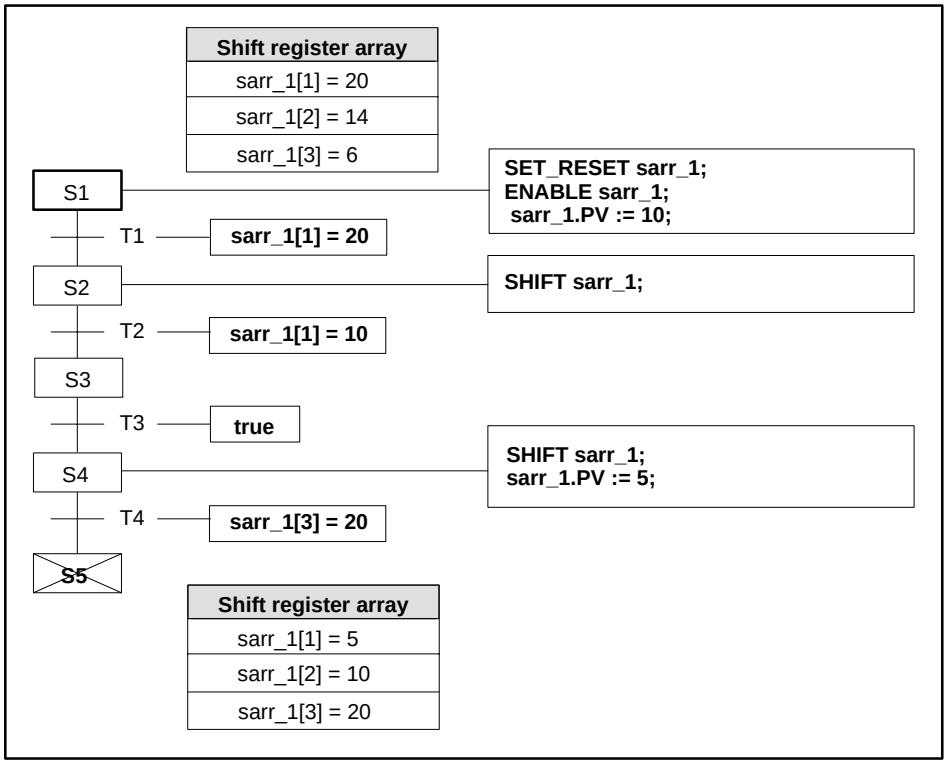


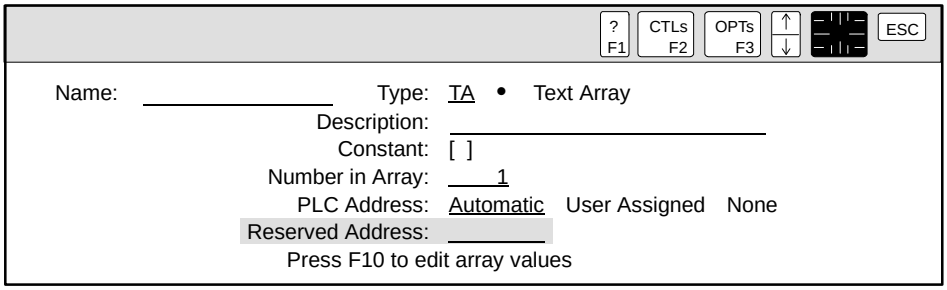
Figure 8-10 Using Shift Register Arrays in SFC Steps

Arrays (continued)

Text Array (TA)

A text array is an array of text variables. Text variables contain up to 30 characters. Each element in the text array is 30 characters long and requires 15 integer memory locations.

Text arrays are primarily for OSx (PCS) tag translate. When you select TA as the declaration type, the form below appears.



Name: unique, symbolic name (12 characters maximum).

Type: TA (code for Text Array).

Description: optional (30 characters maximum).

Constant: X indicates that the text is constant; blank field indicates that the text is variable.

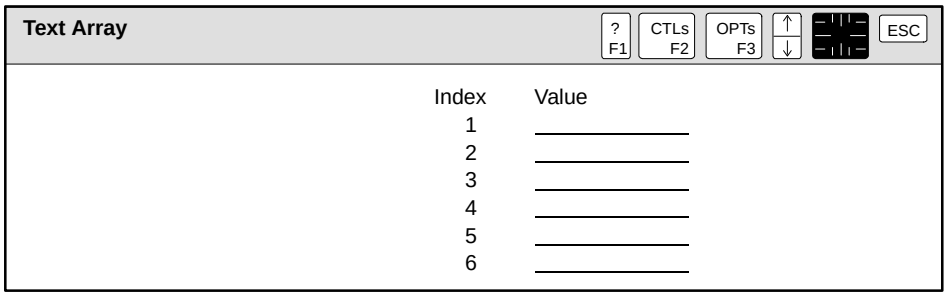
Number in Array: indicates number of elements in the array; default is 1. For Series 505 controllers: text arrays can have up to 2184 elements. For S5 controllers: text arrays can have up to 17 elements.

PLC Address: there are three controller addressing choices: Automatic, User Assigned, and None. (See descriptions below.) Automatic addressing is the default.

> **Automatic:** APT automatically assigns the controller address.
> **User Assigned:** Allows the user to specify a controller address.
> **None:** No controller address is assigned. (This is primarily intended for support of OSx non-networked tags.)

Reserved address: starting controller address you specify for the declaration, e.g., %V19 or %K19 (if constant) for a Series 505 controller, or %DBn:DW19 for S5. You must reserve enough memory in the Compiler Control File for the size of the array. For example, to reserve V locations in a Series 505 controller, the number of V locations = 15 x (number of elements in the array). For S5, the reserved number of data words = 15 x (number of elements in the array). The type of Series 505 memory available for this declaration is %V. The types of S5 memory available for this declaration are %DBn:DW, and %DXn:DW (where n is the block number).

To specify values in the array, press F10 . If you specify 6 as the number of elements in the array, the dialogue window below appears.



Index: location of value in array.

Value: actual text for text array is entered here. Each field can be up to 30 characters long. Any printable character is allowed. The default is empty.

The text array is primarily used to configure OSx (PCS) text tags. Several rules apply when you use text arrays in APT:

- You can assign the character string to the text variable either on the declaration form or through an assignment statement in a Math section.

NOTE: You cannot assign an element in the text array to a string that is enclosed in quotes, as in **alarm [1] := "High Alarm"**. You must assign the string either in the declaration table or in an assignment statement to another text array.

- You can reassign a text array to another text array if they have the same number of array elements. A statement such as **alarm_1:=alarm_2;** is allowed as long as **alarm_1** and **alarm_2** have the same number of array elements.
- You can only index an element in a text array by using a literal value, such as:

```
alarm_1[1]   :=   alarm_2[4];
```

NOTE: You cannot use an integer variable or an expression to index an element in a text array. You must use the appropriate integer value.

8.4 Counters

Using Counters (CT)

You can use a counter inside an SFC step or in the Math section of an SFC step or CFB. The counter counts up from 0 to a preset value every time the input boolean goes from false to true. The preset value can be assigned from the program or when you define the counter in the Declaration Table.

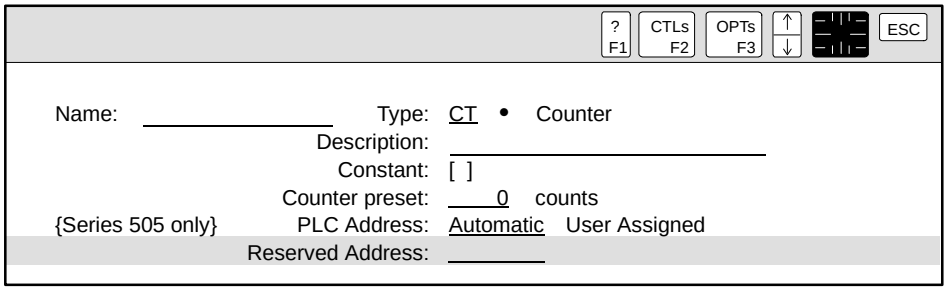
A counter can be controlled with the **.INPUT** and **.ENABL** extensions from Math assignment statements in an SFC step or in a Math or interlock CFB.

- To start the counter counting, the **.ENABL** bit must be true.
- When the **.INPUT** bit transitions from false to true, the current counter count (**.TCC**) is incremented by one.
- When the **.TCC** count equals the **.TCP** count, the **.COUT** becomes true.
- To reset the counter, set the **.ENABL** bit to false. The **.COUT** becomes false, and the **.TCC** resets to zero.

Because a declared counter in Series 505 controllers stores data temporarily in internal memory locations, you must not use counters in user-defined subroutines. You can use counters in user-defined subroutines in S5 controllers.

NOTE: APT counter operation is different from that of S5 counters. APT counters count up from 0 to a preset value. When the count equals the preset, the **.COUT** becomes true. When you use the APT Declaration Table to declare a counter for your S5 controller, the counter that is generated operates as an APT counter.

When you select CT as the declaration type, the form below appears.



Name: unique, symbolic name (12 characters maximum).

Type: CT (code for Counter).

Constant: (Series 505 only) X indicates protected counter; blank field indicates preset can be changed. For S5 controllers, leave this field blank.

Description: optional (30 characters maximum).

Counter preset: integer between 0 and 32767; specifies preset value of counter.

PLC Address: Automatic addressing is the default. If you have a Series 505 controller, you also get the User Assigned option.

> **Automatic:** APT automatically assigns the controller address.
> **User Assigned:** allows the user to specify a Series 505 controller address.
> (There is no None option for counters.)

Reserved address: (Series 505 only) starting controller address you specify for the declaration, e.g., %TC2. The type of Series 505 memory available for this declaration is %TC.

Counters (continued)

Table 8-10 lists the extensions for the counter. Extensions can be used in a transition, in the Math section of an SFC step, or in Math Language statements in a CFB.

Table 8-10 CT Extensions

Extensions		Command
.TCC	Current counter count (read-only integer).	There are no commands.
.TCP	Preset counter count (read/write integer).	
.ENABL	Indicates counter has been activated (read/write boolean).	
.INPUT	Variable that increments the current counter count every time the value goes from false to true (read/write boolean).	
.COUT	Becomes true when the current count equals the preset. The **.COUT** extension is false when the current count does not equal the preset (read-only boolean).	

Figure 8-11 shows examples that illustrate how to use counter commands and extensions.

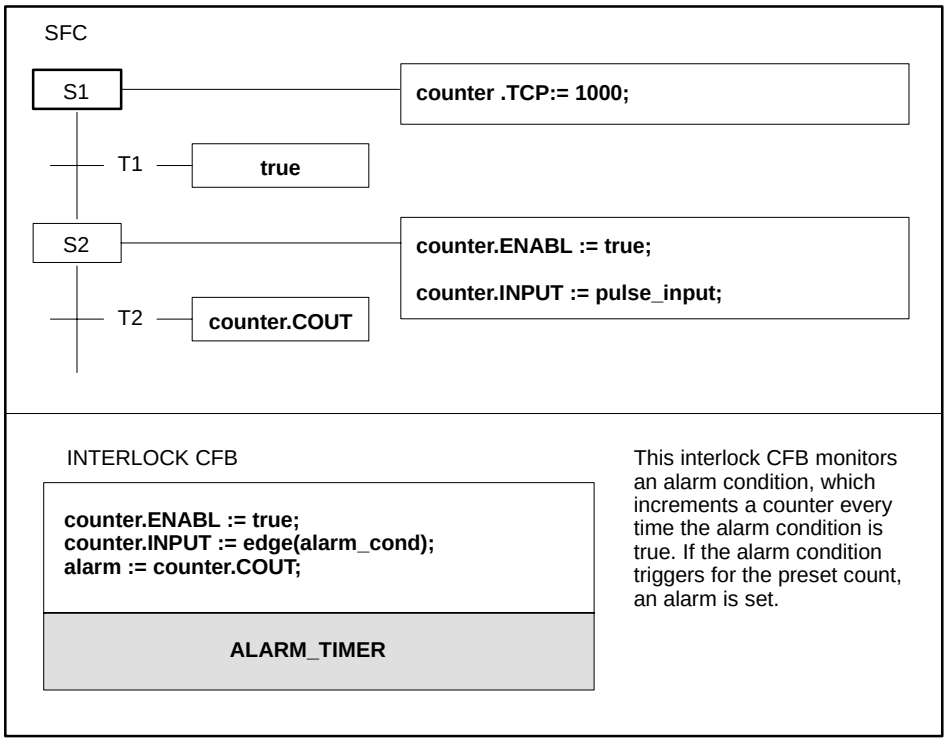


Figure 8-11 Using Counters in SFCs or CFBs

8.5 Timers

Using Timers

The fast and slow timers allow you to set up a delay time inside an SFC step or in the Math section of an SFC step or CFB. The timers count down to 0 from a preset value. The preset value can be assigned from the program or when you define the timer in the Declaration Table.

A timer can be started with the DELAY command.

- The DELAY command causes the timer to begin counting down as soon as the SFC step becomes active. The timer automatically resets the current value to the preset value and sets **.TOUT** to false when the program exits that step, or the Math section stops executing.
- DELAY commands for the same timer should not be placed in consecutive SFC steps because there is not enough time for the timer to be reset.
- The DELAY command can also be used in a CFB that is RLL only.

A timer can also be controlled with the **.RESET** and **.ENABL** extensions from Math assignment statements in an SFC step or in a Math or interlock CFB

- To start or restart the timer, set both the **.RESET** and **.ENABL** bits to true.
- To pause the timer, keep the **.RESET** bit true and change the **.ENABL** bit to false. To resume timing, set the **.ENABL** bit back to true.
- To reset the timer, set the **.RESET** bit to false. It does not matter which state the **.ENABL** bit is in.

Because a declared timer in Series 505 controllers store data temporarily in internal memory locations, you must not use timers in user-defined subroutines. You can use timers in user-defined subroutines in S5 controllers.

NOTE: For S5 controllers, APT uses the S5 timers T0–T255. They operate as On Delay timers.

Figure 8-12 shows examples that illustrate how to use timer commands and extensions.

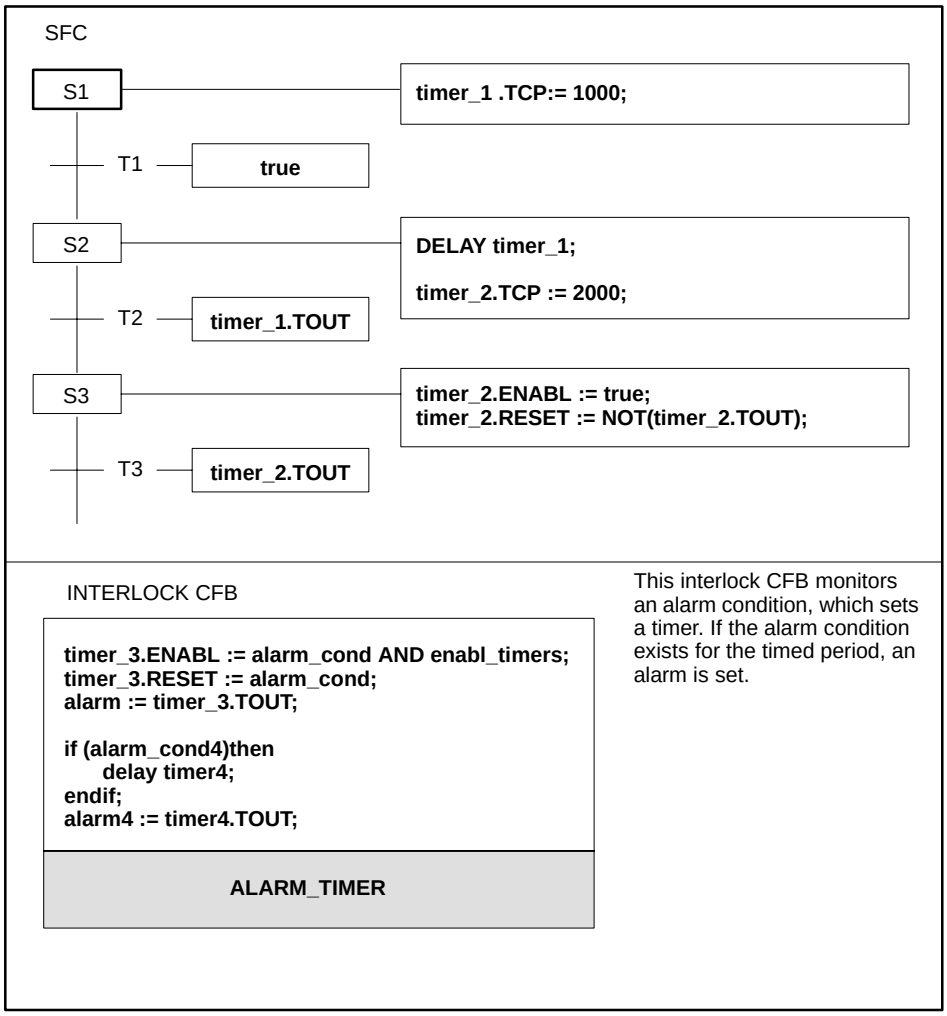


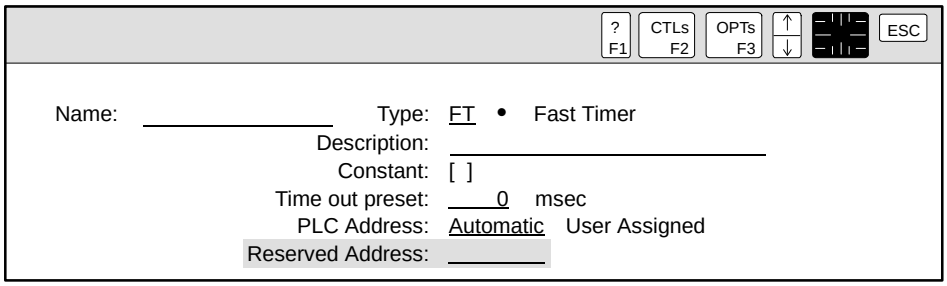
Figure 8-12 Using Timers in SFCs or CFBs

NOTE: Be consistent when you access a timer; that is, do not use the DELAY command and write to **.ENABL** and **.RESET** for the same timer.

Timers (continued)

Fast Timer (FT)

The fast timer counts down from the preset to 0 at a rate of .001 seconds. When you select FT as the declaration type, the form below appears:



Name: unique, symbolic name (12 characters maximum).

Type: FT (code for Fast Timer).

Description: optional (30 characters maximum).

Constant: (Series 505 only) X indicates protected timer; blank field indicates preset can be changed.

Time out preset: integer between 0 and 32767; specifies preset value of timer multiplied by .001 seconds.

PLC Address: there are two controller addressing choices: Automatic and User Assigned. Automatic addressing is the default.

> **Automatic:** APT automatically assigns the controller address.
> **User Assigned:** Allows the user to specify a controller address. (There is no None option for fast timers.)

Reserved address: starting controller address you specify for the declaration, e.g., %TC2 for a Series 505 controller, or %TMR2 for S5. The type of Series 505 memory available for this declaration is %TC. The type of S5 memory available for this declaration is %TMR.

NOTE: For S5 controllers, the APT fast timer uses a 10-millisecond ON DELAY timer. Consequently, while the APT preset and current values are expressed in milliseconds, the accuracy for S5 timers is increments of 10 milliseconds.

Table 8-11 lists the extensions and commands for the fast timer. The commands can be used in the parallel section of an SFC step or a Math block that generates RLL or STL code. Extensions can be used in a transition, in the Math section of an SFC step, or in Math Language statements in a CFB.

Table 8-11 FT Extensions and Commands

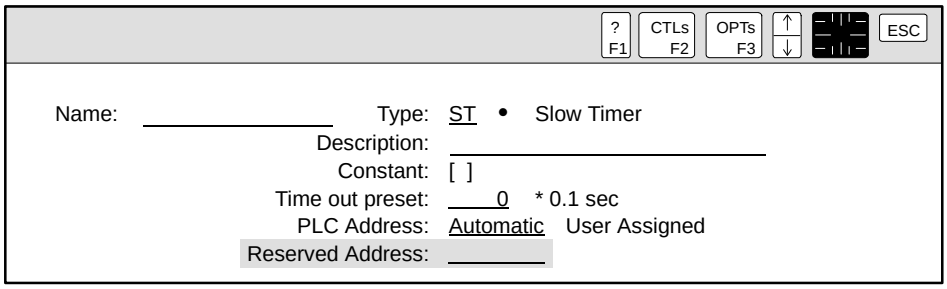
Extensions		Command	
.TCC	Current timer count (read-only integer).	DELAY*	Starts timer.
.TCP	Preset counter time: 0.001 × value (read/write integer).		
.ENABL	Indicates timer has been activated (read/write boolean).		
.RESET	Becomes false to indicate that the current count has been reset to the preset value. The .RESET extension becomes true to indicate that the timer can be activated if the .ENABL extension is set to true (read/write boolean).		
.TOUT	Becomes true when the current count equals zero. The .TOUT extension is false when the current count does not equal zero (read-only boolean).		

* This timer command executes only once each time that the program enters the step. This command works only on variables defined as a timer in the Declaration Table.

Timers (continued)

Slow Timer (ST)

The slow timer counts down from the preset to 0 at a rate of .1 seconds. When you select ST as the declaration type, the form below appears:



Name: unique, symbolic name (12 characters maximum).

Type: ST (code for Slow Timer).

Description: optional (30 characters maximum).

Constant: (Series 505 only) X indicates protected timer; blank field indicates preset can be changed.

Time out preset: integer between 0 and 32767; specifies preset value of timer multiplied by .1 seconds.

PLC Address: there are two controller addressing choices: Automatic and User Assigned. Automatic addressing is the default.

> **Automatic:** APT automatically assigns the controller address.
> **User Assigned:** Allows the user to specify a controller address. (There is no None option for slow timers.)

Reserved address: starting controller address you specify for the declaration, e.g., %TC2 for a Series 505 controller, or %TMR2 for S5. The type of Series 505 memory available for this declaration is %TC. The type of S5 memory available for this declaration is %TMR.

Table 8-12 lists the extensions and commands for the slow timer. The commands can be used in the parallel section of an SFC step or a Math block that generates RLL or STL code. Extensions can be used in a transition, in the Math section of an SFC step, or in Math Language statements in a CFB.

Table 8-12 ST Extensions and Commands

Extensions		Command	
.TCC	Current timer count (read-only integer).	DELAY*	Starts timer.
.TCP	Preset counter time: 0.1 × value (read/write integer).		
.ENABL	Indicates timer has been activated (read/write boolean).		
.RESET	Becomes false to indicate that the current count has been reset to the preset value. The **.RESET** extension becomes true to indicate that the timer can be activated if the **.ENABL** extension is set to true. (read/write boolean).		
.TOUT	Becomes true when the current count equals zero. The **.TOUT** extension is false when the current count does not equal zero (read-only boolean).		

* This timer command executes only once each time that the program enters the step. This command works only on variables defined as a timer in the Declaration Table.

Chapter 9
Recipes

9.1 Understanding Recipes

Overview

An APT recipe serves as a storage place for a set of related values that have different data types. You develop an APT recipe in two basic steps:

- The first step is to add a Recipe Template at the Program Content Level. Each template contains a list of elements and corresponding data types that make up a recipe.
- The second step is to create a recipe and specify the associated template in the Recipe Usage Table. This can be done at either the Program Content Level or at the Unit Content Level. You can use this Recipe Usage Table to assign initial values (or quantities) to the recipe elements.

Recipe Template

The recipe template contains the master list of all ingredients and control information that you need to make a set of similar products. Because a Recipe Template is a master list, you can include all of the ingredients for making variations of a similar product in one template.

For example, in the *WASH_DEM* program, all the parameters for controlling the wash cycle are included in one template, as shown in Figure 9-1. Another template could be created that contains the parameters for controlling a drying cycle.

Recipe Usage Table

A recipe that you name in the Recipe Usage Table automatically contains the same elements as the template that you specify. The default values of these elements is zero. You can edit the recipe and assign initial values to each element; elements can be assigned a value of zero.

For example, the recipe for a normal wash cycle shown in Figure 9-1 assigns a value of warm for temperature setpoint, 1.5 minutes for wash time, 1 minute for rinse time, etc. The recipe for the P_Press wash cycle assigns a value of cold for temperature setpoint, 0.75 minutes for wash time, and 1.0 minutes for rinse time.

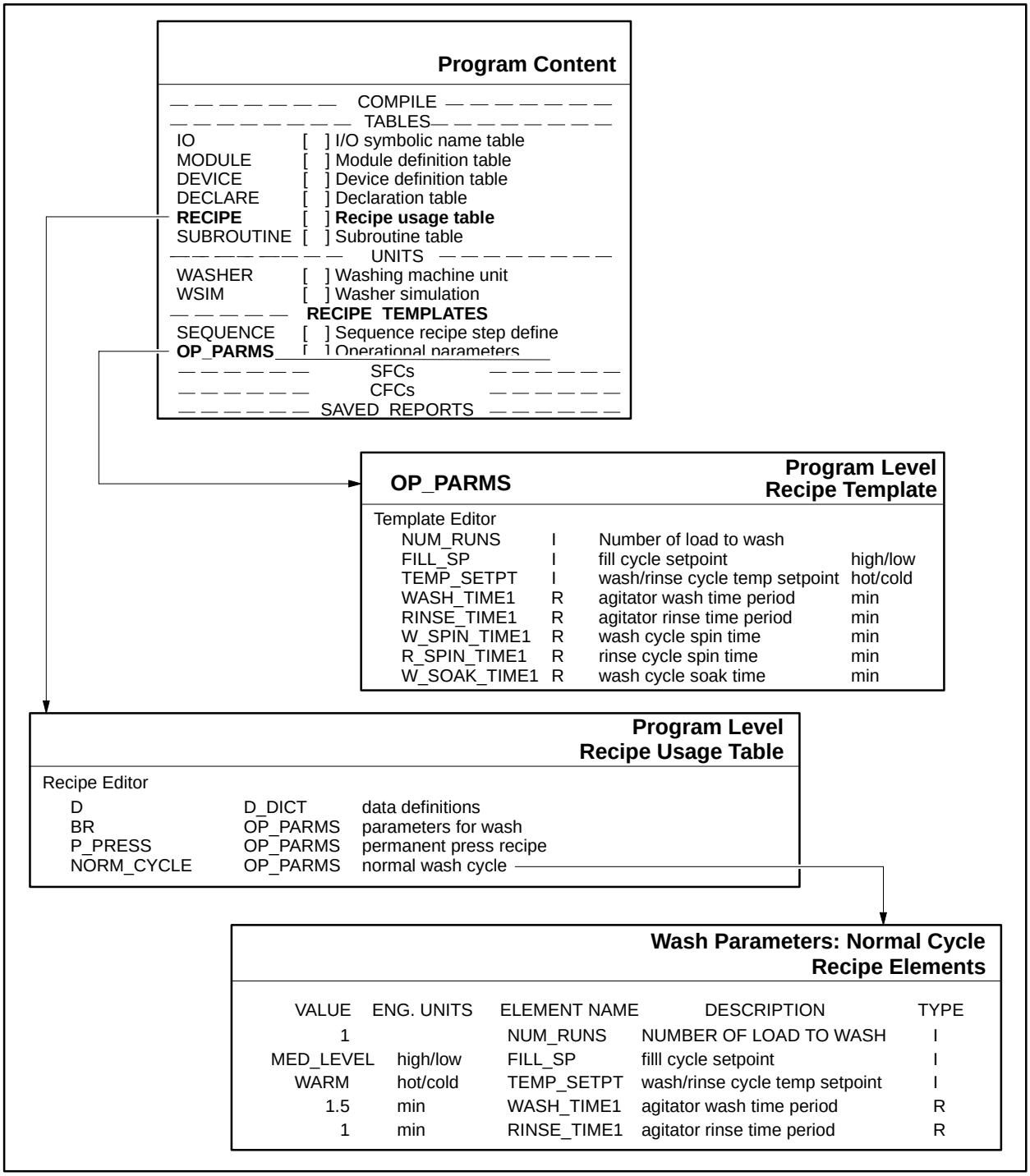


Figure 9-1 Recipe Usage Table

9.2 Defining Recipes

Overview

APT provides two methods for using recipes in a program:

- You can define recipe elements at the Program Level, reserve a space for the recipe at the Unit Level, and then use a command in an SFC step to move the values defined at the Program Level into the unit recipe.
- You can edit a recipe at the Unit Level and access the recipe elements with no additional steps. This allows you to store a set of values for use in a process that does not change.

Downloading from OSx (PCS)

The UNLOCK or CLEAR command makes the recipe available for data. The SELECT command moves the data into the recipe. Figure 9-2 shows how to use the handshaking bits in APT. First, use the CLEAR command to clear the recipe; this sets the extensions to zero. Then, use the SELECT command to select a recipe and download it to the controller. When the download is complete, the .INUSE extension automatically goes to true, indicating that the recipe is in the controller and is ready to process.

APT provides the following extensions that are specifically designed to work with the OSx (PCS) recipe package to enable the handshaking and allow recipes to be downloaded from OSx to APT.

- The **.INUSE** extension indicates when the recipe is available. OSx can only download a recipe when the recipe is not in use, and **.INUSE** = false.
- The **.DSTBL** extension indicates that a recipe is in the process of being downloaded. While OSx is downloading a recipe, **.DSTBL** = false.
- The recipe data is ready when the **.DRDY** extension is true. The **.INUSE** extension must also equal true, indicating that the recipe is in use, and **.DSTBL** must be true, indicating that the data is stable. See Figure 9-2.

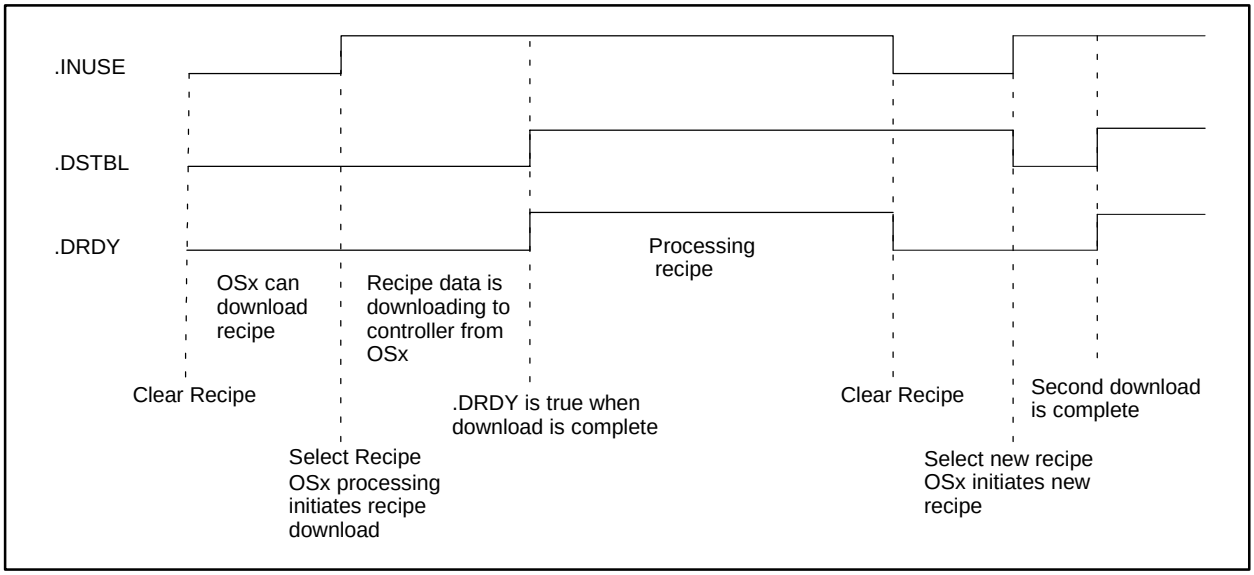


Figure 9-2 OSx (PCS) Handshaking with APT

APT also provides the ability to scale recipes from OSx (PCS), select a specific recipe from OSx, and to put a recipe on hold indefinitely from OSx.

APT recipes can be used as recipes or data structures. APT recipes can handle different data types with a common name. Data structures can be marked for tag translate, but they do not get the recipe extensions for OSx handshaking.

Selecting a Recipe

APT allows you to move recipes from one area of the hierarchy to another. For example, you can copy a Program Level recipe to the Unit Level, from one Program Level recipe to another, or from a Unit Level recipe to the Program Level.

To select a recipe, use **SELECT** *destination_recipe source_recipe*; in the parallel section of an SFC step or a math block that generates RLL code.

The *destination_recipe* is the name of the recipe that you want to use in your program. The *source_recipe* is the name of the recipe that you want to copy.

To copy recipes in a Math block, use the syntax shown below.

```
recipe1 := recipe2;
```

Using Recipe Elements

You can reference the elements of the recipe in an assignment statement in either the parallel or math section of an SFC step, or in the math section of a CFB. You can also use a recipe element as the input to a CFB.

To access values of elements in the recipe, you use the recipe name with the element as an extension. For example, if the unit recipe name is *WR* (for Working Recipe) and the elements are *Num_runs*, *Fill_sp*, and *Wash_time1*, you reference these elements by: **WR.num_runs**, **WR.fill_sp**, and **WR.wash_time1**.

You can assign values to recipe elements, or assign these elements to other variables, or to extensions of other objects that you defined in the Declaration Table:

```
WR.num_runs := 5;
Timer_Set := WR.wash_time1;
```

NOTE: For more efficient code assignments, use integers or real numbers instead of boolean values in recipes.

Recipe Extensions and Commands

Table 9-1 shows the extensions and commands used with recipes. Remember that data structures do not use extensions or commands, and that constant recipes do not use the commands.

Table 9-1 Recipe Extensions and Commands

Extensions (Read/Write Boolean)		Command	
.RTU	Request to unlock. (F)[1]	UNLOCK/CLEAR	Makes the recipe available for data.
.INUSE	Data is in use.	SELECT	Makes data from one recipe available to another.
.DSTBL	Data is stable.		
.DRDY	Data is ready for use.		
.STATUS	Recipe status.[2]		

1 This boolean is an APT flag that is manipulated by the Math Procedures Latch, Clear, and On.

2 The boolean extension **.INUSE** and element 1 of the status boolean array **.STATUS[1]** are equivalent. The boolean extension **.DSTBL** and element 2 of the status boolean array **.STATUS[2]** are also equivalent. For example, you achieve the same results by changing the state of the **.INUSE** extension, or by writing to **.STATUS[1].**

Example

Figure 9-3 illustrates how recipes can be handled within an SFC. In this example, you are waiting for the master recipe to receive the recipe data from the operator station. When the recipe data is received, it sets the master recipe .DRDY to true. APT then moves the master recipe into the slave recipe and clears the master recipe (which sets master recipe .DRDY to false, as is shown in Figure 9-2).

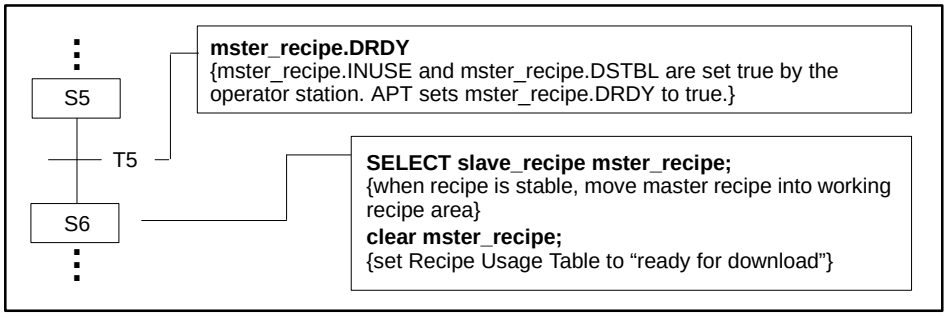


Figure 9-3 Using Recipes

9.3 Recipe Templates

Creating a Recipe Template

Recipe Templates reside at the Program Content level of the hierarchy. Each APT program can have more than one Recipe Template, as is shown in Figure 9-4.

A Recipe Template is the source for all Program Content and Unit Content recipes because it contains the master list of all elements necessary for making a specific product. The list can include the names of ingredients as well as other processing information such as time, temperature, etc.

The first step in defining a template is to add a template to the hierarchy at the Program Content Level.

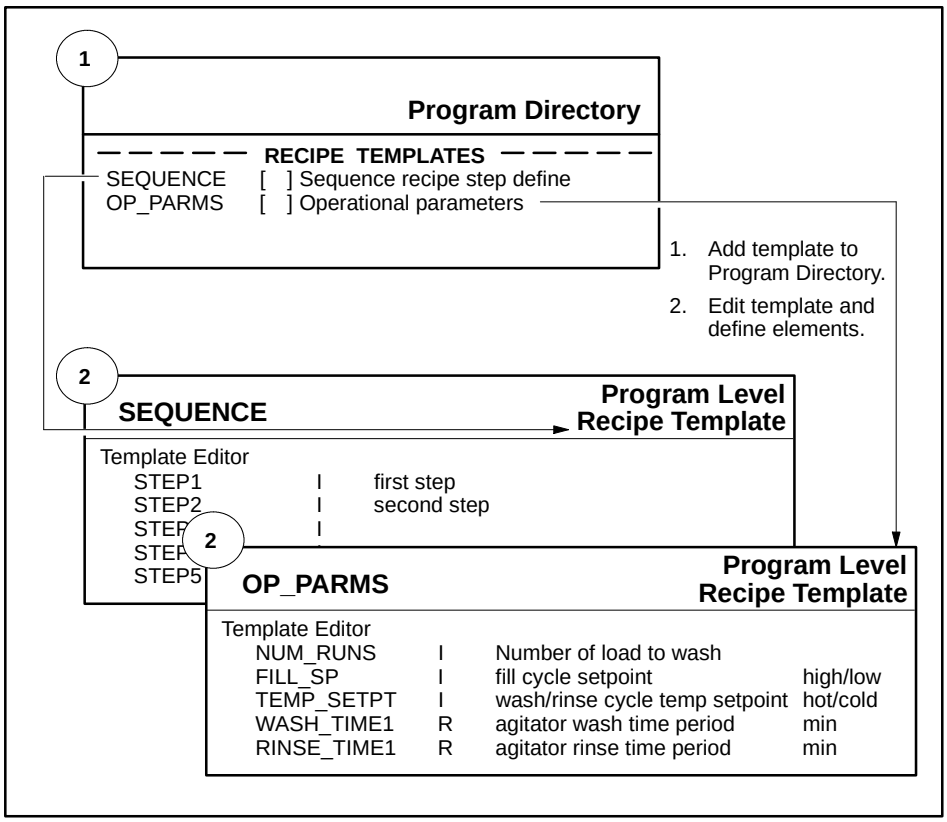
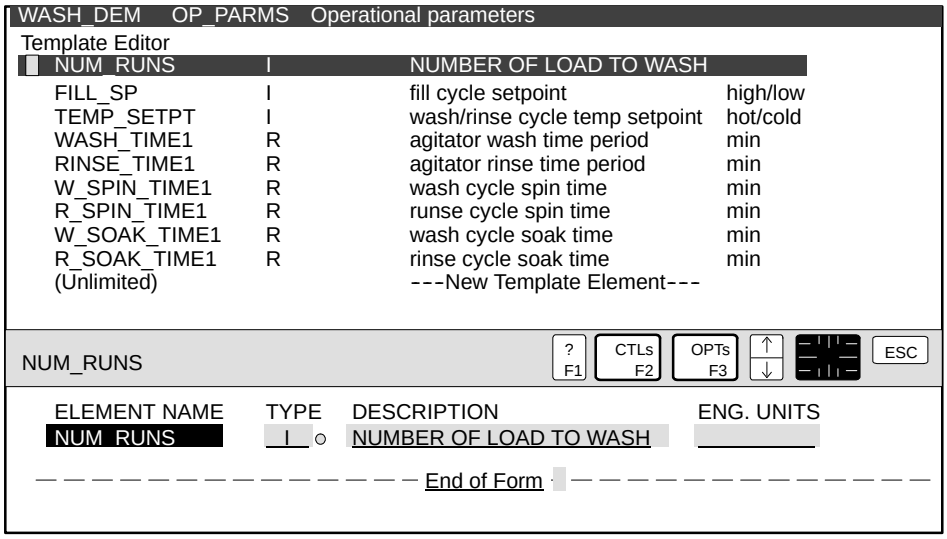


Figure 9-4 Creating a Template

Editing a Template

The next step is to edit the template and define the ingredients and/or process for the recipe. When you edit a Recipe Template, this form appears:



Element Name: symbolic name that identifies the element (12 characters maximum). Do not use the following extension names, which are created by APT for recipe control: **.INUSE**, **.STATUS**, **.DRDY**, **.RTU**, and **.DSTBL.**

Type: code that represents one of the following: B (boolean), R (real), I (integer), BA (boolean array), IA (integer array), or RA (real array).

Description: optional description of the template element (30 characters maximum).

Engineering units: unit of measurement associated with the element (optional, for description only in APT).

NOTE: The maximum number of elements in a recipe for a Series 505 controller is determined by the amount of V-memory you have available. The maximum number of elements in a recipe for an S5 controller is 256 words.

9.4 Program and Unit Recipe Usage Tables

Creating a Program or Unit Recipe

A Recipe Usage Table is available at both the Program Content Level and the Unit Content Level of the hierarchy. The Recipe Usage Tables contain names of recipes and values for the elements defined in the template.

Recipes in the Recipe Usage Table can have the same or different template names, e.g., OP_PARMS, shown in Figure 9-5, and OP_PARMS and SEQUENCE in Figure 9-6. Each recipe in the Unit Content Recipe Usage Table can have either the same or a different recipe name, e.g., WR and SR, shown in Figure 9-6. The same names are possible because they are in different units.

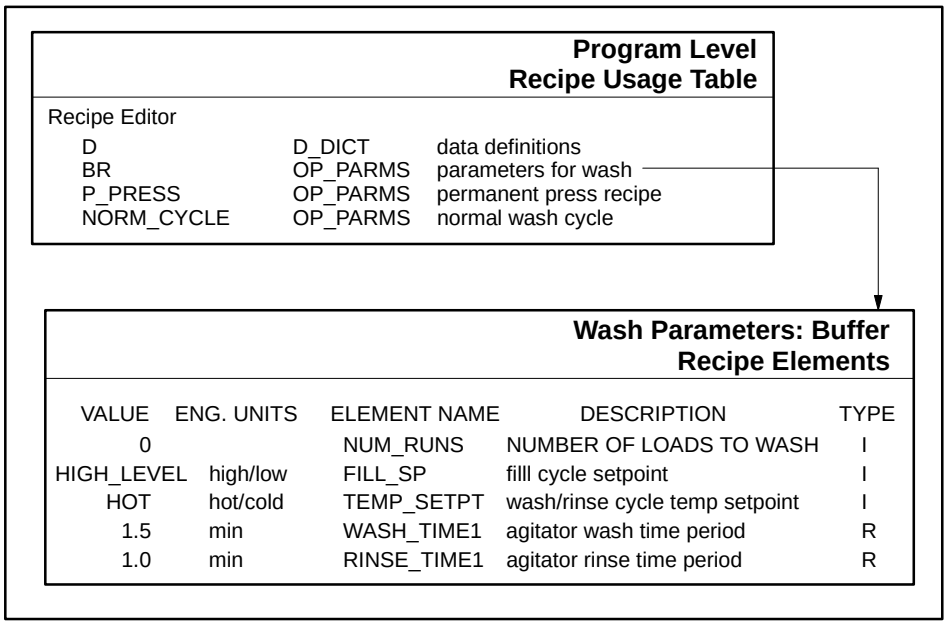


Figure 9-5 Program Recipe Usage Table

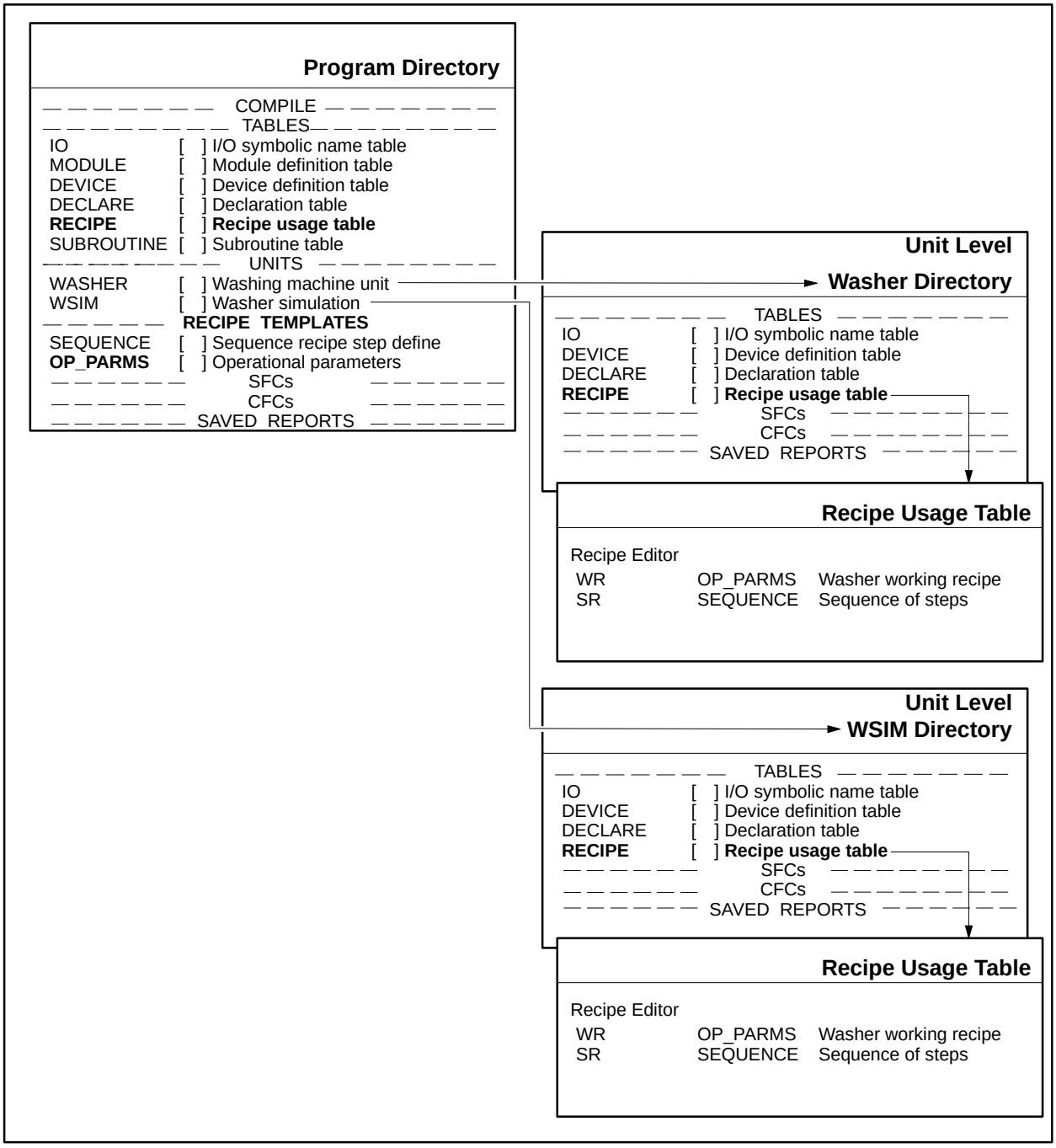


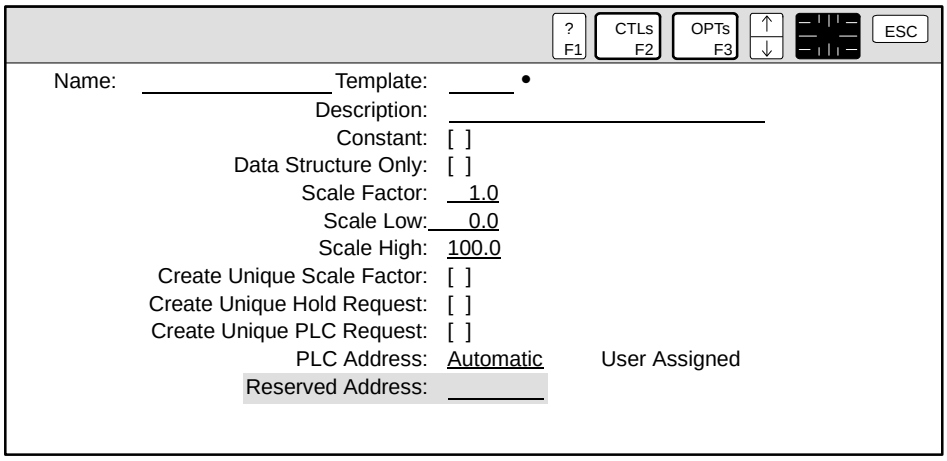
Figure 9-6 Unit Recipe Usage Table

Editing Recipe Elements

You must define a Recipe Template before you define a recipe in the Recipe Usage Table. When you edit the Recipe Usage Table, specify the name of a recipe and the name of a template, and choose whether the recipe is constant or a data structure only. The scaling parameters, scale factor, hold request, and the controller request are primarily for OSx (PCS) use.

Then, edit the recipe elements in order to assign initial values in the Recipe Usage Table. You can also assign values to recipe elements with math blocks, as is described on page 9-6.

When you select or create a recipe, this form appears:



Name: unique, symbolic name (12 characters maximum).

Template: Valid template name.

Description: 30 characters maximum (optional).

Constant: X indicates recipe is constant. A constant recipe can have boolean elements in it, but you cannot hold Series 505 C-memory locations or S5 flag and data memory constant. A constant recipe does not use the commands for APT recipes, but it can use the handshaking bits (extensions).

Data Structure Only: X means that the recipe is only a data structure and that it does not use the commands and extensions for APT recipes.

Scale Factor: A positive scale factor can be entered to scale the recipe that is downloaded from OSx (PCS). You can enter fractions or whole numbers between 0.0 – 999.0. Default is 1.0.

Scale Low: The lowest scale factor allowed. It must be a positive number between 0.0 and 999.0.

Scale High: The highest scale factor allowed. It must be a positive number between 1.0 – 999.0. Default is 100.0.

Create Unique Scale Factor: X allocates a real variable called *recipe_name*.**SCALE** so that you can dynamically change the scale factor. The *recipe_name*.**SCALE** variable must be a positive number between 0.0 – 999.0.

Create Unique Hold Request: X allocates a bit called *recipe-name*.**HOLDREQ** so that you can dynamically put a recipe on hold indefinitely. When downloading a recipe from OSx (PCS), you can transfer a recipe a block at a time. The **HOLDREQ** bit can stop OSx from downloading the next block of the recipe. The recipe is on hold when the bit is set true and downloading resumes when the bit is set to false. Hold request eliminates the OSx maximum time to wait for the next step.

Create Unique PLC Request: X allocates an integer variable called *recipe-name*.**RECREQ** so that a recipe can be requested by number. If your recipe in OSx were named R32767, you would download R32767 into *recipe-name* by setting *recipe-name*.**RECREQ** equal to 32767.

.**RECREQ** must always contain a value.

–1 cancels the download that is in progress from OSx.
–2 illustrates waiting for the next recipe selection (this is the default).
–3 requests the last selected recipe on OSx.
0 to 32767 are valid integer recipe numbers.

PLC Address: Refers to controller address allocation for the declaration as listed below.

Automatic addressing is the default: APT automatically assigns the controller address.
User Assigned: Allows user to specify a controller address.

Reserved address: controller starting address specified for the recipe, e.g., %V19 and %K19 (if constant) for Series 505 controllers or %DBn:DW19 for S5 controllers. The types of Series 505 memory available for a recipe are V and K memory. The types of S5 memory available for a recipe are %DBn:DW and %DXn:DW (where n is the block number). A recipe with a user-assigned starting address cannot have any boolean elements because of data type mix (V-memory and C-memory for Series 505). You cannot have an all-boolean recipe with user-assigned addressing. You must carefully calculate the amount of memory to reserve because of the mix of reals and integers.

Refer to the *SIMATIC PCS 7 OSx Recipe Manual* for more information.

When you edit the recipe elements, the table appears as shown in Figure 9-7. All the information that you specified in the template is automatically placed in the recipe, and you can specify the appropriate values for each element. Note that you can only change the initial value in the Value field if you have already marked the recipe for OSx (PCS) tag translation.

The value that you assign must correspond to the engineering units, value type, and range for each element. If the value is a boolean value, you must specify a 1 for true or a 0 for false.

The value can be a variable name, although it must be declared a CONSTANT variable. APT does not allow the recipe elements to be indirectly changed. You can change the recipe elements in a math statement, as shown in "Using Recipe Elements" on page 9-6, by setting a numeric value equal to the *recipe_name*.***recipe_element***. You cannot, however, set the value equal to a variable, then change that variable in APT, and have the recipe updated. The variable changes, but the *recipe_name*.***recipe_element*** keeps the same initial value. You cannot enter a setpoint, a process variable, or an output into the value field, although you can set a recipe element equal to any of these variables in a math statement.

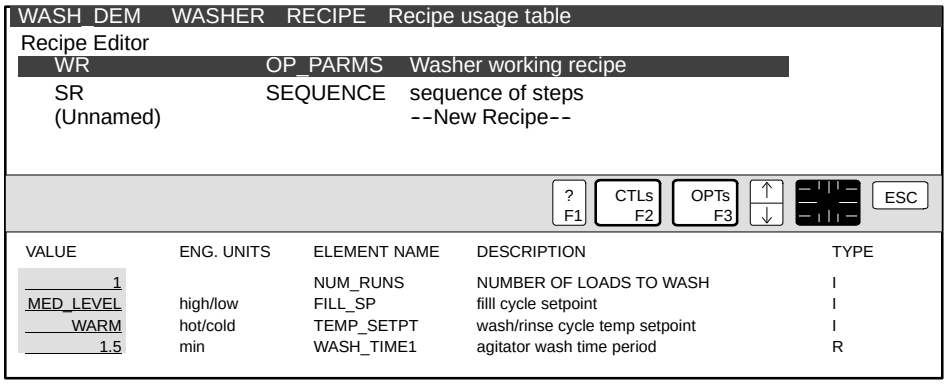


Figure 9-7 Editing Recipe Elements

9.5 Implementing Recipes in APT

Recipes are first loaded with their values, and then implemented in APT to produce the desired product.

Values can be loaded into the recipe by either downloading the recipes from an operator station, such as OSx (PCS), or by loading the values directly from APT. Loading the values directly from APT can be done in math blocks in CFCs or SFCs, or by filling in the values in the recipe usage table. Downloading from OSx is accomplished by manipulating the handshaking bits, as is shown in Figure 9-2.

Once the recipes are in place, they are implemented in the controller by using either SFCs or sequence arrays. To use an SFC or a sequence array, you invoke the recipe step-by-step. First, you load the ***recipe_elements*** into the appropriate places (that is, in setpoints, timer and counter presets, etc.) using math statements. Then, you wait for a condition to come true before you move to the next step. You continue to sequence through the steps until your process is complete. These techniques are discussed in the *SIMATIC APT Applications Manual*.

Chapter 10

User Subroutines

10.1 User-defined and System-activated Subroutines

Overview

The *SIMATIC APT Programming Reference (Graphics/Math) Manual* introduces the concept of math procedures. The procedure, a pre-defined operation that does a single mathematical task, is used as a single statement in a math block. You can expand the list of APT-provided procedures by custom-designing your own procedure as a subroutine. You can use subroutines for code that is repeated where only the inputs and outputs change. When the subroutine needs to be executed, use the subroutine name in a math statement, just as you do for any APT procedure.

APT provides two types of subroutines.

- **User-defined subroutines** Subroutines written in the Math language that can pass up to five parameters. See Section 10.2.
- **System-activated subroutines** Subroutines used to program the interrupt organization blocks (OB1 through OB39) for S5 controllers. See Section 10.3.

Availability

The user-defined subroutines are available for both controller families; the system-activated subroutines are only available for S5 controllers.

Prerequisites

Before you attempt to write a subroutine, you must be familiar with the APT Math language rules. If you have a Series 505 controller, you must know the differences between RLL- and SF program-generated code. If you have an S5 controller, read the STL syntax information provided in the appendix on "Inline Assembly Code for S5," located in the *SIMATIC APT Programming Reference (Graphics/Math) Manual*.

User-defined Subroutine

You create a user-defined subroutine from the Subroutine Table Editor, using the Math language. You can pass a maximum of five parameters to the subroutine when it is called for execution. These parameters can be single values or arrays. You can call a user-defined subroutine from SFC math, CFB math, or from the associated math of a PID loop or an analog alarm.

The parameters of a user-defined subroutine are both inputs and outputs. That is, you can modify a parameter within the subroutine, and pass the new value of the parameter back to the calling function when the subroutine completes its execution.

System-activated Subroutine

If you have an S5 controller, you can create a subroutine, using the Subroutine Table Editor, that programs an OB between OB1–OB39. You can only program one OB per subroutine, and the subroutine must contain the complete code for the OB; you cannot access one OB from several subroutines. To write the code for your interrupt OB, you can use either the Math language or STL. If you use STL, you are only permitted to issue a jump command to an FB that is written in STL and downloaded with STEP 5.

System-activated subroutines do not pass parameters. They are not called from within APT; instead, they are executed when the OB interrupt or event occurs.

10.2 Creating a User-defined Subroutine

Using the Subroutine Form

When you create a user-defined subroutine, use the form shown in Figure 10-1. The parameters for a user-defined subroutine are optional. You can pass parameters as input-only, output-only, or input/output.

NOTE: APT treats all subroutine parameters as input/output. You must ensure that your subroutine does not write to parameters that are intended to be input-only.

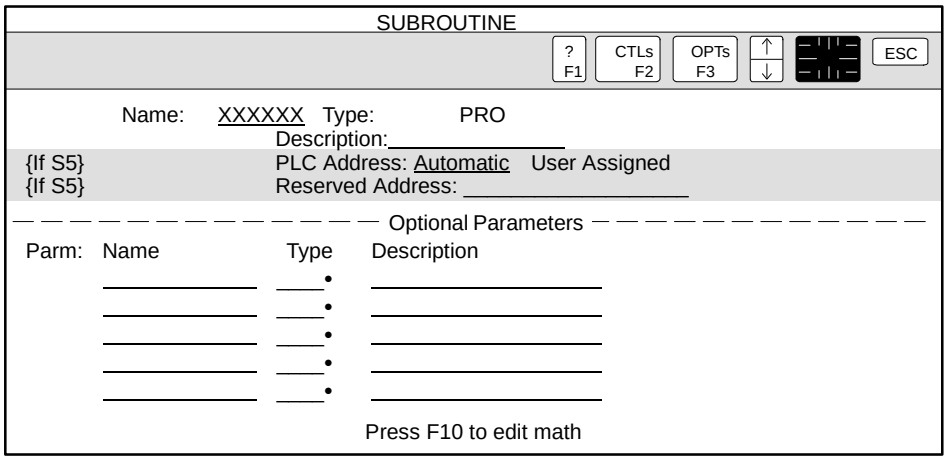


Figure 10-1 Subroutine Form for User-defined Subroutines

Name: Unique name that identifies the subroutine (8 characters, maximum).

Type: PRO (procedure).

Description: Description of the subroutine (30 characters, maximum).

PLC Address: There are two controller addressing choices: Automatic and User Assigned. Automatic addressing is the default.

> **Automatic:** Use Automatic to create a user-defined subroutine.
> **User Assigned:** (S5 only) Use the User Assigned field to create a system-activated subroutine. The system-activated subroutine is described in Section 10.3.

Reserved Address: (S5 only) Used when creating a system-activated subroutine.

Parameter Name: Name of the parameter referenced in the math section; must be unique within the subroutine (12 characters, maximum).

Parameter Type: Type of parameter: Integer (I), Real (R), Boolean (B), APT Flag (F), Integer Array (IA), Real Array (RA), Boolean Array (BA).

Parameter Description: Optional description of the parameter (30 characters, maximum).

Using Parameters in a User-defined Subroutine

The user-defined subroutine supports up to five parameters. The parameter names are local and can only be referenced within the subroutine. You can use the same names elsewhere in the program. If you use a parameter name that is the same as an object name (e.g., Global Declaration Table variable), the parameter is referenced, not the object. Parameter names are governed by the same rules that apply to local declarations for Math language.

If you use an array as a parameter type in a user-defined subroutine, the entire array is copied, both when the inputs are copied to the subroutine and when the outputs are copied from the subroutine. You can pass one element (for example, ARRAY1[1]) in an array as a parameter in the subroutine.

NOTE: (Series 505 only) If your user-defined subroutine is SFPGM-only, you cannot create a local variable or reference a global entity from within the subroutine. You must use the parameter list to pass the variables to and from APT.

Math in a User-defined Subroutine

After entering the optional parameters, you are ready to write the math for your user-defined subroutine. The math text is comprised of three sections: the optional code specifier (PRAGMA); declarations; and Math statements. Figure 10-2 shows these three sections.

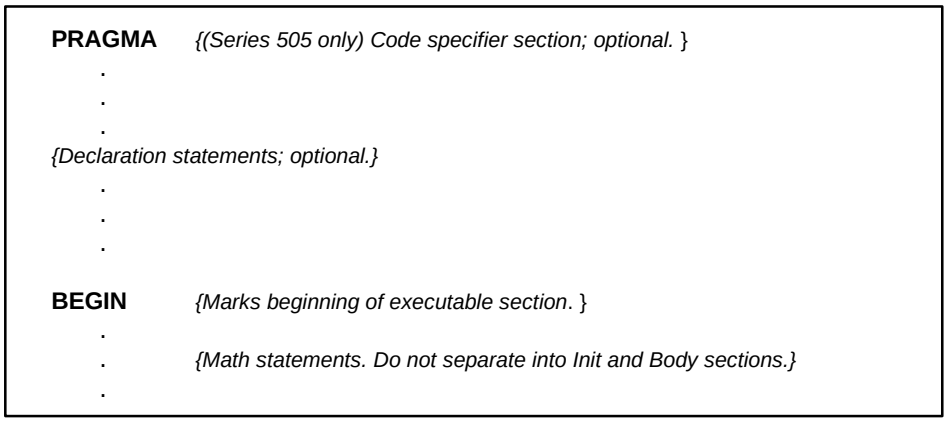


Figure 10-2 Math Structure for User-defined Subroutine

The structure of the user-defined subroutine is the same as for any math block, except that there can be no Init or Body sections. PRAGMA, the optional code specifier section, is only available for Series 505 controllers.

Design Guidelines

Follow these rules as you design your user-defined subroutine.

- The rules for Math language apply to subroutine math. However, the subroutine cannot have an Init or Body section.
- Do not use any of the APT flag procedures (Latch, On, or Clear). Use APT flags only on the right of an assignment statement or within an IF expression.
- The local declarations for your subroutine follow the same rules as local declarations in the Math language.
- You cannot use INCREMENT or DECREMENT in subroutines.
- You can nest subroutines up to four levels.
- A subroutine is not re-entrant and cannot call itself.

Your controller type also influences the way you must design a user-defined subroutine. See the following pages for some Series 505-specific design guidelines.

For Series 505 controllers, the code that you enter in your user-defined subroutine determines whether the subroutine can be called from RLL or SFPGM math blocks. Code not specific to RLL or SF program blocks can be called from both types of math blocks. For a discussion of RLL-specific and SF program-specific math blocks, refer to the Math Language Overview chapter in the *SIMATIC APT Programming Reference (Graphics/Math) Manual*. If you have an S5 controller, the considerations discussed below do not apply, because APT compiles all of your code in STL.

RLL-only Subroutine (Series 505 only)

APT generates an RLL-only subroutine if you do any of the following in your code:

- Use RLL-only functions or procedures, and include the optional code specifier PRAGMA ("RLL").
- Base your code on integer math operations.
- Create local declarations.
- Reference global entities (variables, declarations, recipes, CFB extensions, etc.) in the subroutine math.

Note that APT generates more overhead code for a subroutine that is RLL specific. To handle the input/output parameters in an RLL subroutine, APT generates code to do the following tasks:

- Copy input values to the subroutine parameters.
- Call the subroutine for execution.
- Copy the parameter values to the outputs.

SFPGM-only Subroutine (Series 505 only)

A user-defined subroutine can be generated in either RLL or SFPGM. However, the SFPGM subroutines do not allow local declarations or references to global entities.

APT generates an SF program-only subroutine if the code:

- Uses SF program-functions or procedures, and includes the optional code specifier PRAGMA ("SF").
- Bases your code on real math operations.
- Uses indirect addressing in arrays, unless your controller performs indirect addressing in RLL (e.g., 545 Rel. 2.x or greater, 555, or 575). See Table 1-9 in Chapter 1.

Avoiding Temporary Variables in SFPGM-only Subroutines

As a normal part of its operation, a Series 505 controller can temporarily suspend execution of a user-defined subroutine that has been compiled into SFPGM code, in order to enable execution of another subroutine. When this happens, the controller allocates locations in V-Memory to be used as internal, temporary compiler variables for the subroutine that has been suspended. It is possible for data to be lost under these circumstances, since the temporary variables of the first subroutine may be overwritten during execution of the subsequent subroutine.

To avoid the possibility of creating temporary variables that could be overwritten when your first subroutine is interrupted, you need to observe the following guidelines for math statements that you use in an SFPGM-only user-defined subroutine.

- If a function or procedure can accept either a variable or an expression as a parameter, only use variables. An expression causes temporary variables to be allocated. If you need to use a value generated by an expression, you can store the intermediate results of the expression in a Series 505 controller T-Memory location before calling the subroutine (%Tx, where x = 11–16).
- When you use the index of an array as a parameter, do not use an expression for the index; use a variable instead. You can store the intermediate results of an expression in a Series 505 controller T-Memory location before calling the subroutine (%Tx, where x = 11–16).
- Use the PTRUNC procedure instead of the TRUNC function.
- Use the PROUND procedure instead of the ROUND function.
- Avoid using the EDGE function, or device or declaration timers, since they all use internal memory locations for the temporary storage of data.

Example of a User-defined Subroutine

The following example uses a user-defined subroutine to calculate the level of liquid in a tank. If you had several tanks whose levels were being determined in a similar manner, then this subroutine would also allow you to calculate the level of the liquid in each tank.

This example uses two pressure transmitters to calculate the level and the density of the fluid in the tank.

The name of the example subroutine is TANK_LVL; it passes these five parameters: LT_TANK_A, LT_TANK_C, CONSTNTS, SPEC_GRAV, and LVL_TANK.

The first three parameters are inputs: LT_TANK_A and LT_TANK_C are the two pressures that reflect the level of the fluid in the tank, and CONSTNTS is an array of three. SPEC_GRAV and LVL_TANK are the outputs from the subroutine.

Pressure transmitter 1 is represented by LT_TANK_A, which shows the pressure at point A, located one inch above the bottom of the tank.

Pressure transmitter 2 is represented by LT_TANK_C, and is 50 inches higher than the first transmitter. The total tank height is 100 inches.

The values 50, 1 and 100 are sent in the array called CNSTNTS, because they are the constants for this tank.

The math of the user-defined subroutine example is shown in Figure 10-3. For Series 505, this math is compiled in SFPGM, because the subroutine uses the LIMIT math function. The PRAGMA statement shown in the example is only for a Series 505 controller; an S5 does not need it.

```
PRAGMA (“SF”);                {Use this statement if using a Series 505 controller}
BEGIN

{calculate specific gravity}
{subtract pressure at bottom of tank at point A from pressure at middle of
  tank at point C and divide by the distance between the two probes}

SPEC_GRAV := (LT_TANK_A – LT_TANK_C) / CONSTNTS[1];

{calculate tank level in inches}
{divide pressure at the bottom of tank by specific gravity and add
  the height of the bottom transmitter}

LVL_TANK := 0.0;
IF (SPEC_GRAV > 0.0) THEN
   LVL_TANK := (LT_TANK_A / SPEC_GRAV) + CONSTNTS[2];
ENDIF;

{convert level to percentage}
{divide tank level by total height of tank}

LVL_TANK := LVL_TANK / CONSTNTS[3] * 100.0 {%};
LIMIT (0.0,100.0,LVL_TANK,LVL_TANK);
```

Figure 10-3 Math Example for User-defined Subroutine

All of the variables are declared at the global level. A math block calls the subroutine when it is enabled. The math block is as follows:

```
begin
TANK_LVL(LT_TANK_A, LT_TANK_C,CONSTNTS, SPEC_GRAV, LVL_TANK);
```

You can use the subroutine to calculate the density and level in several tanks, instead of writing CFBs to do the same calculation.

10.3 Creating a System-activated Subroutine

Subroutine Form

System-activated subroutines are only available for S5 controllers. The system-activated subroutine allows you to program the code for OB1–OB39. You can use either the APT Math language or STL. The code executes when the OB is called; in other words, when the interrupt or event occurs.

You can program only one OB in each subroutine, and the subroutine must contain all the code for the given OB. For example, if you have several tasks for OB13 (100msec interrupt) to perform, you must put all the tasks in the same subroutine.

Figure 10-4 shows the form you use to program the system-activated subroutine. To indicate that you wish the subroutine to be system-activated, rather than user-defined, you select the User Assigned option. You also need to place the number of the OB that your subroutine accesses in the Reserved Address field. Since it is a reserved address, be sure to enter the percent (%) sign before the OB number.

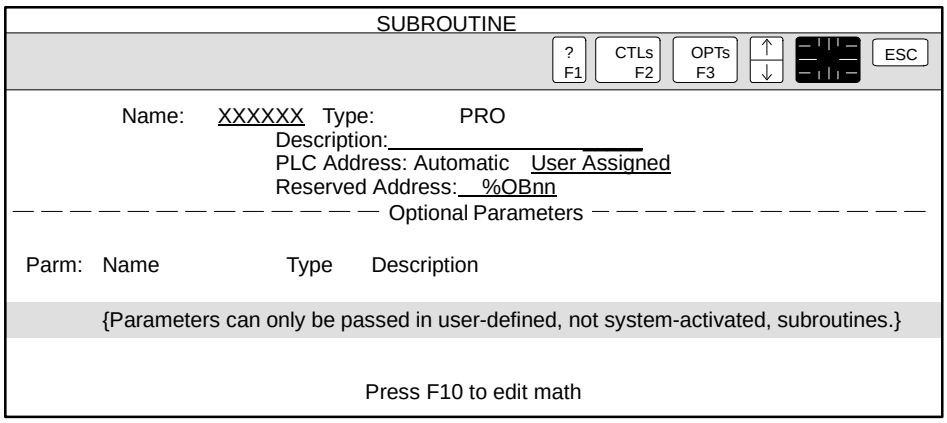


Figure 10-4 Subroutine Form for System-activated Subroutines

Name: Unique name that identifies the subroutine (8 characters, maximum).

Type: PRO (procedure).

Description: Description of the subroutine (30 characters, maximum).

PLC Address: There are two controller addressing choices: Automatic and User Assigned. Automatic addressing is the default.

> **Automatic:** Used to create a user-defined subroutine.
> **User Assigned:** Use the User-Assigned field to create a system-activated subroutine. You must enter the address of the OB that is accessed by your subroutine in the Reserved Address field.

Reserved Address: Enter the address of the OB that is accessed by your subroutine. The syntax is %OBn (where n is 1 through 39).

Programming in a System-activated Subroutine

The math text of the system-activated subroutine is comprised of two sections: the declaration section, and the Math statement section. Figure 10-5 shows these two sections. The structure of the system-activated subroutine is the same as for any math block, except that there can be no Init or Body sections.

```
{Declaration statements; optional.}
    .
    .
    .

BEGIN       {Marks beginning of executable section. }
    .
    .       {Math statements. Do not separate into Init and Body sections.}
    .       {You can use the IN_ASM Math statement in this section.}
    .
```

Figure 10-5 Math Structure for System-activated Subroutine

You can program the OB that is used by your subroutine entirely in the APT Math language; or, if you like, you can jump from the OB to an FB that you have programmed from STEP 5 using STL code.

Design Guidelines

Follow these rules as you design your system-activated subroutine.

- The rules for Math language apply to subroutine math. However, the subroutine cannot have an Init or Body section.
- Do not use any of the APT flag procedures (Latch, On, or Clear). Use APT flags only on the right of an assignment statement or within an IF expression.
- 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: *Subroutine_name*.**NNL.***Declaration_name*
- All of the Math language statements, functions, and procedures are available to you, except for the following procedures: SCALE, UNSCALE, LOOKUP_TABLE, LEAD_LAG, or INTERPOLATE.

NOTE: APT generates an error if you try to use SCALE, UNSCALE, LOOKUP_TABLE, LEAD_LAG, or INTERPOLATE in a system-activated subroutine.

- You cannot call a user-defined subroutine from a system-activated subroutine.
- You cannot call a system-activated subroutine from anywhere within APT, including from a user-defined or system-activated subroutine. System-activated subroutines are invoked when an interrupt event occurs.

⚠ WARNING

System-activated subroutines execute automatically when the interrupt condition or event occurs.

Trying to call a system-activated subroutine from within your APT program could cause the controller to behave unpredictably, which could result in death or serious injury to personnel, and/or damage to equipment.

Do not call system-activated subroutines from within APT.

When you program a system-activated subroutine, APT generates much more efficient code if you interrupt only on Block boundaries and not on Operations boundaries. Select the Interrupt on Block Boundaries option from the Compiler Control File.

When the interrupt occurs for the following OBs, the CPU executes the interrupt and returns to the point in the code where it was before the interrupt occurred.

CPU928B	CPU948
OBs 2 through 19	OBs 2 through 19
OBs 23 through 27	OBs 23 through 34
OBs 29 through 35	OB 37

Any data that you manipulate from the OB program of your system-activated subroutine during an interrupt takes on the new value your OB assigns. For instance, you might have a global variable that counts the number of cartons being manufactured in your factory, and design an OB to reset the counter to zero. Each time there is an interrupt, when the process resumes, the value of the counter variable is zero, regardless of what it was prior to the interrupt. In this manner, you can purposefully manipulate variables with a system-activated subroutine.

Accessing an FB from a System-activated Subroutine

If you want to access an FB in your system-activated subroutine, so that you can program using STL, you must understand how to integrate external blocks into APT, using the IN_ASM statement. See the appendix on "Inline Assembly Code for S5," located in the *SIMATIC APT Programming Reference (Graphics/Math) Manual*.

To bring the FB into your system-activated subroutine, follow these steps:

1. Program the FB from STEP 5.
2. Download the FB to the controller, using STEP 5.
3. Use an IN_ASM statement in the code of your system-activated APT subroutine to invoke the FB.

Using Math Statements in a System-activated Subroutine

When you program your OB exclusively in the APT Math language, the code is more readable, and the code is also automatically checked by the compiler. You can program using all of the Math language statements, functions, and procedures with the exception of the procedures noted earlier in this section, under "Design Guidelines": INTERPOLATE, LOOKUP_TABLE, LEAD_LAG, SCALE, and UNSCALE.

Programming your OB with APT Math language, and invoking an FB that has been coded from STEP 5 in STL, are not mutually exclusive options; you can do both in a single subroutine. Figure 10-6 shows an example of a system-activated subroutine that performs operations in the APT Math language and also calls out from the OB to an FB that has been programmed with STL.

Figure 10-6 shows an example of a system-activated subroutine called Sub1.

```
{Local declaration section}

INTEGER: INT5;

BEGIN

{Section for Math statements, including IN_ASM}

INT5  :=  100;

{Math statements}
                    .
                    .
                    .
IN_ASM  [*
                    {STL statements (L, JU, C, and A are permitted in IN_ASM)}
                    {This is where your OB invokes the FB you have programmed in STL}
                                        .
                                        .
                                        .
                    {This STL statement references INT5, the local variable of the subroutine}
                    L  SUB1.NNL.INT5
*];
{More Math statements, including more IN_ASM statements}
                    .
                    .
                    .
```

Figure 10-6 Math Example for System-activated Subroutine

Notice that the local declaration, INT1, is referenced within an IN_ASM statement using the syntax that is outlined earlier in this section, under "Design Guidelines."

Appendix A

Device RLL Code for Series 505

A.1 Device Extensions and Table Options

Availability

If you have a Series 505 controller, your APT devices generate RLL code. If you have an S5 controller, your APT devices generate STL code. The RLL code presented in this appendix is for Series 505 only.

Section A.1 lists the various device extensions in tabular form, according to the options that can be chosen in the device table. For each device option, reference is made to the figure in Section A.2 that illustrates the appropriate RLL code.

Table A-1 lists the options used in the tables of this section. Notice that the definition is the same as the field label used in the device table. For example, the code N.O. FDBK [X] means that an **X** was placed in the brackets adjacent to the **Normally open feedback** field for a device.

Table A-1 Device Options Used in the RLL Code

Device	Option	Definition
Valves	EO	Energized state: open
	EC	Energized state: closed
	Clear FTx [X]	Clear CMMD on FTO/FTC?: [X] (Selected)
Motors	Latch FTR [X]	Latch the FTR extension?: [X] (Selected)
Cylinders	EE	Energized state: extend
	ER	Energized state: retract
Presses	ER	Energized state: raised
	EL	Energized state: lowered
	Clear FTx [X]	Clear CMMD on FTR/FTL?: [X] (Selected)
All Devices	No UPS	Uninterruptible power supply not selected
	IGN FDBK OVRD [ ]	Ignore feedback override?: [ ] (Not selected)
	IGN FDBK OVRD [X]	Ignore feedback override?: [X] (Selected)
	N.O. FDBK [X]	Normally open feedback?: [X] (Selected)

For each example, the Uninterruptible Power Supply (UPS) option was not selected in the compile control table. When the UPS option is selected, the **.PFAIL** contact does not appear. The **.PFAIL** contact is triggered by the *pgm_name*.**PWRFL** extension.

If the **Ignore Feedback Override** option is selected, the **.OVRDx** logic is not generated.

VND

The extensions and options for the VND device are listed in Table A-2 and illustrated in the figures specified in the table.

Table A-2 VND Extensions and Options

Extension	Option	Figure
OPND		Figure A-1
CLSD		Figure A-2
FAILD		Figure A-3
MOPEN		Figure A-4

VSN

The extensions and options for the VSN device are listed in Table A-3 and illustrated in the figures specified in the table. See Table A-1 for definitions of the options.

Table A-3 VSN Extensions and Options

Extension	Option	Figure
Move image from V-Memory	NO UPS	Figure A-5
LOCKD		Figure A-6
CMMD	EO	Figure A-7
	EC	Figure A-8
MOPEN	EO	Figure A-9
	EC	Figure A-10
OPND	EO	Figure A-11
	EC	Figure A-12
CLSD	EO	Figure A-13
	EC	Figure A-14
TRVL		Figure A-15
Move image to V-Memory	NO UPS	Figure A-16

Device Extensions and Table Options (continued)

VSS

The extensions and options for the VSS device are listed in Table A-4 and illustrated in the figures specified in the table. See Table A-1 for definitions of the options.

Table A-4 VSS Extensions and Options

Extension	Option	Figure
Move image from V-Memory	NO UPS	Figure A-5
LOCKD		Figure A-6
OVRD	IGN FDBK OVRD [ ]	Figure A-17
OPENC	EO	Figure A-18
	EC	Figure A-19
MOPEN	EO	Figure A-20
	EC	Figure A-21
Open Time Expired	EO	Figure A-22
	EC	Figure A-23
Closed Time Expired	EO	Figure A-24
	EC	Figure A-25
OPND	EO	Figure A-26
	EC	Figure A-27
	EO, N.O. FDBK [X]	Figure A-28
	EC, N.O. FDBK [X]	Figure A-29
	EO, IGN FDBK OVRD [X]	Figure A-30
	EC, IGN FDBK OVRD [X]	Figure A-31
	EO, Clear FTx [X]	Figure A-32
	EO, N.O. FDBK [X], IGN FDBK OVRD [X]	Figure A-33
	EO, N.O. FDBK [X], IGN FDBK OVRD [X], Clear FTx [X]	Figure A-34
	EC, N.O. FDBK [X],IGN FDBK OVRD [X]	Figure A-35
CLSD	EO	Figure A-36
	EC	Figure A-37
	EO, N.O. FDBK [X]	Figure A-38
	EC, N.O. FDBK [X]	Figure A-39
	EO, IGN FDBK OVRD [X]	Figure A-40
	EC, IGN FDBK OVRD [X]	Figure A-41
	EC, Clear FTx [X]	Figure A-42
	EO, N.O. FDBK [X], IGN FDBK OVRD [X]	Figure A-43
	EC, N.O. FDBK [X], IGN FDBK OVRD [X]	Figure A-44
	EC, N.O. FDBK [X], IGN FDBK OVRD [X], Clear FTx [X]	Figure A-45

Table A-4 VSS Extensions and Options (continued)

Extension	Option	Figure
TRVL	EO	Figure A-46
	EC	Figure A-47
FTO		Figure A-48
FTC		Figure A-49
CMMD	EO/EC	Figure A-50
	EO, Clear FTx [X]	Figure A-51
	EC, Clear FTx [X]	Figure A-52
Move image to V-Memory	NO UPS	Figure A-16

Device Extensions and Table Options (continued)

VSD

The extensions and options for the VSD device are listed in Table A-5 and illustrated in the figures specified in the table. See Table A-1 for definitions of the options.

Table A-5 VSD Extensions and Options

Extension	Option	Figure
Move image from V-Memory	NO UPS	Figure A-5
LOCKD		Figure A-6
OVRDO	IGN FDBK OVRD [ ]	Figure A-53
OVRDC	IGN FDBK OVRD [ ]	Figure A-54
OPENC	EO	Figure A-18
	EC	Figure A-19
MOPEN	EO	Figure A-20
	EC	Figure A-21
Open Time Expired	EO	Figure A-22
	EC	Figure A-23
Closed Time Expired	EO	Figure A-24
	EC	Figure A-25
OPND	EO	Figure A-55
	EC	Figure A-56
	EO, N.O. FDBK [X]	Figure A-57
	EC, N.O. FDBK [X]	Figure A-58
	EO, IGN FDBK OVRD [X]	Figure A-59
	EC, IGN FDBK OVRD [X]	Figure A-60
	EO, Clear FTx [X]	Figure A-61
	EO, N.O. FDBK [X], IGN FDBK OVRD [X]	Figure A-62
	EO, N.O. FDBK [X], IGN FDBK OVRD [X], Clear FTx [X]	Figure A-63
	EC, N.O. FDBK [X], IGN FDBK OVRD [X]	Figure A-64

Table A-5 VSD Extensions and Options (continued)

Extension	Option	Figure
CLSD	EO	Figure A-65
	EC	Figure A-66
	EO, N.O. FDBK [X]	Figure A-67
	EC, N.O. FDBK [X]	Figure A-68
	EO, IGN FDBK OVRD [X]	Figure A-69
	EC, IGN FDBK OVRD [X]	Figure A-70
	EC, Clear FTx [X]	Figure A-71
	EO, N.O. FDBK [X], IGN FDBK OVRD [X]	Figure A-72
	EC, N.O. FDBK [X], IGN FDBK OVRD [X]	Figure A-73
	EC, N.O. FDBK [X], IGN FDBK OVRD [X], Clear FTx [X]	Figure A-74
TRVL	EO	Figure A-46
	EC	Figure A-47
FAILD		Figure A-75
FTO		Figure A-76
	IGN FDBK OVRD [X]	Figure A-77
FTC		Figure A-78
	IGN FDBK OVRD [X]	Figure A-79
CMMD	EO/EC	Figure A-80
	EO, Clear FTx [X]	Figure A-81
	EC,Clear FTx [X]	Figure A-82
Move image to V-Memory	NO UPS	Figure A-16

Device Extensions and Table Options (continued)

VDD

The extensions and options for the VDD device are listed in Table A-6 and illustrated in the figures specified in the table. See Table A-1 for definitions of the options.

Table A-6 VDD Extensions and Options

Extension	Option	Figure
Move image from V-Memory	NO UPS	Figure A-5
LOCKD		Figure A-6
OVRDO	IGN FDBK OVRD [ ]	Figure A-53
OVRDC	IGN FDBK OVRD [ ]	Figure A-54
CMMD		Figure A-83
OPENC		Figure A-84
CLSC		Figure A-85
MOPEN		Figure A-9
OPNTO		Figure A-86
CLSTO		Figure A-87
OPND		Figure A-88
	N.O. FDBK [X]	Figure A-89
	IGN FDBK OVRD [X]	Figure A-90
	N.O. FDBK [X], IGN FDBK OVRD [X]	Figure A-91
CLSD		Figure A-92
	N.O. FDBK [X]	Figure A-93
	IGN FDBK OVRD [X]	Figure A-94
	N.O. FDBK [X], IGN FDBK OVRD [X]	Figure A-95
TRVL		Figure A-96
FTO		Figure A-97
FTC		Figure A-98
FAILD		Figure A-99
Move image to V-Memory	NO UPS	Figure A-16

VMD

The extensions and options for the VMD device are listed in Table A-7 and illustrated in the figures specified in the table. See Table A-1 for definitions of the options.

Table A-7 VMD Extensions and Options

Extension	Option	Figure
Move image from V-Memory	NO UPS	Figure A-5
LOCKD		Figure A-6
OVRDO	IGN FDBK OVRD [ ]	Figure A-53
OVRDC	IGN FDBK OVRD [ ]	Figure A-54
CMMD		Figure A-83
STOP		Figure A-100
OPENC		Figure A-101
CLSC		Figure A-102
MOPEN		Figure A-9
OPNTO		Figure A-103
CLSTO		Figure A-104
OPND		Figure A-88
	N.O. FDBK [X]	Figure A-89
	IGN FDBK OVRD [X]	Figure A-90
	N.O. FDBK [X], IGN FDBK OVRD [X]	Figure A-91
CLSD		Figure A-92
	N.O. FDBK [X]	Figure A-93
	IGN OVRD [X]	Figure A-94
	N.O. FDBK [X], IGN FDBK OVRD [X]	Figure A-95
TRVL		Figure A-96
FTO		Figure A-97
FTC		Figure A-98
FAILD		Figure A-99
Move image to V-Memory	NO UPS	Figure A-16

Device Extensions and Table Options (continued)

BV1

The extensions and options for the BV1 device are listed in Table A-8 and illustrated in the figures specified in the table. See Table A-1 for definitions of the options.

Table A-8 BV1 Extensions and Options

Extension	Option	Figure
Move image from V-Memory	NO UPS	Figure A-5
LOCKD		Figure A-6
OVRDL	IGN FDBK OVRD [ ]	Figure A-105
OVRDH	IGN FDBK OVRD [ ]	Figure A-106
SLOW		Figure A-107
SHIGH		Figure A-108
MOPEN		Figure A-109
MHIGH		Figure A-110
OPNTO		Figure A-111
CLSTO		Figure A-112
OPNDL		Figure A-113
OPNDH		Figure A-114
OPND		Figure A-115
CLSD		Figure A-116
TRVL		Figure A-117
FTOL		Figure A-118
FTOH		Figure A-119
FTO		Figure A-120
FTC		Figure A-121
FAILD		Figure A-99
Move image to V-Memory	NO UPS	Figure A-16

BV2

The extensions and options for the BV2 device are listed in Table A-9 and illustrated in the figures specified in the table. See Table A-1 for definitions of the options.

Table A-9 BV2 Extensions and Options

Extension	Option	Figure
Move image from V-Memory	NO UPS	Figure A-5
LOCKD		Figure A-6
OVRDL	IGN FDBK OVRD [ ]	Figure A-105
OVRDH	IGN FDBK OVRD [ ]	Figure A-106
DRV		Figure A-122
POS		Figure A-123
MOPEN		Figure A-124
MHIGH		Figure A-110
OPNTO		Figure A-125
CLSTO		Figure A-126
OPNDL		Figure A-113
OPNDH		Figure A-114
OPND		Figure A-115
CLSD		Figure A-116
TRVL		Figure A-127
FTOL		Figure A-118
FTOH		Figure A-119
FTO		Figure A-120
FTC		Figure A-121
FAILD		Figure A-99
Move image to V-Memory	NO UPS	Figure A-16

Device Extensions and Table Options (continued)

VUD

The extensions and options for the VUD device are listed in Table A-10 and illustrated in the figures specified in the table.

Table A-10 VUD Extensions and Options

Extension	Option	Figure
OPNTO		Figure A-128
CLSTO		Figure A-129

MSN

The extensions and options for the MSN device are listed in Table A-11 and illustrated in the figures specified in the table. See Table A-1 for definitions of the options.

Table A-11 MSN Extensions and Options

Extension	Option	Figure
Move image from V-Memory	NO UPS	Figure A-130
LOCKD		Figure A-6
CMMD		Figure A-131
MSTRT		Figure A-132
RUNNG		Figure A-133
STPPD		Figure A-134
TRVL		Figure A-135
Move image to V-Memory	NO UPS	Figure A-136

MSS

The extensions and options for the MSS device are listed in Table A-12 and illustrated in the figures specified in the table. See Table A-1 for definitions of the options.

Table A-12 MSS Extensions and Options

Extension	Option	Figure
Move image from V-Memory	NO UPS	Figure A-130
LOCKD		Figure A-6
OVRD	IGN FDBK OVRD [ ]	Figure A-17
STRTC		Figure A-137
MSTRT		Figure A-138
Running Time Expired		Figure A-139
Stopped Time Expired		Figure A-140
RUNNG		Figure A-141
	IGN FDBK OVRD [X]	Figure A-142
	LATCH FTR [X]	Figure A-143
	IGN FDBK OVRD [X], LATCH FTR [X]	Figure A-144
STPPD		Figure A-145
	IGN FDBK OVRD [X]	Figure A-146
TRVL		Figure A-147
FTR		Figure A-148
FTS		Figure A-149
CMMD		Figure A-150
	LATCH FTR [X]	Figure A-151
Move image to V-Memory	NO UPS	Figure A-136

Device Extensions and Table Options (continued)

MDN

The extensions and options for the MDN device are listed in Table A-13 and illustrated in the figures specified in the table. See Table A-1 for definitions of the options.

Table A-13 MDN Extensions and Options

Extension	Option	Figure
Move image from V-Memory	NO UPS	Figure A-130
LOCKD		Figure A-6
CMMD		Figure A-152
STRTC		Figure A-153
STOPC		Figure A-154
MSTRT		Figure A-132
RUNNG		Figure A-133
STPPD		Figure A-134
TRVL		Figure A-135
Move image to V-Memory	NO UPS	Figure A-136

MDS

The extensions and options for the MDS device are listed in Table A-14 and illustrated in the figures specified in the table. See Table A-1 for definitions of the options.

Table A-14 MDS Extensions and Options

Extension	Option	Figure
Move image from V-Memory	NO UPS	Figure A-130
LOCKD		Figure A-6
OVRD	IGN FDBK OVRD [ ]	Figure A-17
CMMD		Figure A-152
STRTC		Figure A-155
STOPC		Figure A-156
MSTRT		Figure A-132
STRTO		Figure A-157
STPTO		Figure A-158
RUNNG		Figure A-159
	IGN FDBK OVRD [X]	Figure A-160
STPPD		Figure A-161
	IGN FDBK OVRD [X]	Figure A-162
TRVL		Figure A-163
FTR		Figure A-164
FTS		Figure A-165
Move image to V-Memory	NO UPS	Figure A-136

MUD

The extensions and options for the MUD device are listed in Table A-15 and illustrated in the figures specified in the table.

Table A-15 MUD Extensions and Options

Extension	Option	Figure
STRTO		Figure A-166
STPTO		Figure A-167

Device Extensions and Table Options (continued)

RM1

The extensions and options for the RM1 device are listed in Table A-16 and illustrated in the figures specified in the table. See Table A-1 for definitions of the options.

Table A-16 RM1 Extensions and Options

Extension	Option	Figure
Move image from V-Memory	NO UPS	Figure A-130
LOCKD		Figure A-6
OVRDF	IGN FDBK OVRD [ ]	Figure A-168
OVRDR	IGN FDBK OVRD [ ]	Figure A-169
FWRD		Figure A-170
SFWRD		Figure A-171
REV		Figure A-172
SREV		Figure A-173
MSTRT		Figure A-174
MREV		Figure A-175
STRTO		Figure A-176
STPTO		Figure A-177
RUNF		Figure A-178
RUNR		Figure A-179
RUNNG		Figure A-180
STPPD		Figure A-181
TRVL		Figure A-182
FTRF		Figure A-183
FTRR		Figure A-184
FTR		Figure A-185
FTS		Figure A-186
FAILD		Figure A-187
Move image to V-Memory	NO UPS	Figure A-136

RM2

The extensions and options for the RM2 device are listed in Table A-17 and illustrated in the figures specified in the table. See Table A-1 for definitions of the options.

Table A-17 RM2 Extensions and Options

Extension	Option	Figure
Move image from V-Memory	NO UPS	Figure A-130
LOCKD		Figure A-6
OVRDF	IGN FDBK OVRD [ ]	Figure A-168
OVRDR	IGN FDBK OVRD [ ]	Figure A-169
FWRD		Figure A-170
SFWRD		Figure A-171
DRV		Figure A-188
DIR		Figure A-189
MSTRT		Figure A-190
MREV		Figure A-191
STRTO		Figure A-176
STPTO		Figure A-177
RUNF		Figure A-178
RUNR		Figure A-179
RUNNG		Figure A-180
STPPD		Figure A-181
TRVL		Figure A-182
FTRF		Figure A-183
FTRR		Figure A-184
FTR		Figure A-185
FTS		Figure A-186
FAILD		Figure A-187
Move image to V-Memory	NO UPS	Figure A-136

Device Extensions and Table Options (continued)

TS1

The extensions and options for the TS1 device are listed in Table A-18 and illustrated in the figures specified in the table. See Table A-1 for definitions of the options.

Table A-18 TS1 Extensions and Options

Extension	Option	Figure
Move image from V-Memory	NO UPS	Figure A-130
LOCKD		Figure A-6
OVRDL	IGN FDBK OVRD [ ]	Figure A-192
OVRDH	IGN FDBK OVRD [ ]	Figure A-193
SLOW		Figure A-194
SHIGH		Figure A-195
MSTRT		Figure A-196
MHIGH		Figure A-197
STRTO		Figure A-198
STPTO		Figure A-199
RUNL		Figure A-200
RUNH		Figure A-201
RUNNG		Figure A-202
STPPD		Figure A-203
TRVL		Figure A-204
FTRL		Figure A-205
FTRH		Figure A-206
FTR		Figure A-207
FTS		Figure A-208
FAILD		Figure A-187
Move image to V-Memory	NO UPS	Figure A-136

TS2

The extensions and options for the TS2 device are listed in Table A-19 and illustrated in the figures specified in the table. See Table A-1 for definitions of the options.

Table A-19 TS2 Extensions and Options

Extension	Option	Figure
Move image from V-Memory	NO UPS	Figure A-130
LOCKD		Figure A-6
OVRDL	IGN FDBK OVRD [ ]	Figure A-192
OVRDH	IGN FDBK OVRD [ ]	Figure A-193
DRV		Figure A-209
SPEED		Figure A-210
MSTRT		Figure A-190
MHIGH		Figure A-197
STRTO		Figure A-211
STPTO		Figure A-212
RUNL		Figure A-200
RUNH		Figure A-201
RUNNG		Figure A-202
STPPD		Figure A-203
TRVL		Figure A-213
FTRL		Figure A-205
FTRH		Figure A-206
FTR		Figure A-207
FTS		Figure A-208
FAILD		Figure A-187
Move image to V-Memory	NO UPS	Figure A-136

Device Extensions and Table Options (continued)

CSD

The extensions and options for the CSD device are listed in Table A-20 and illustrated in the figures specified in the table. See Table A-1 for definitions of the options.

Table A-20 CSD Extensions and Options

Extension	Option	Figure
Move image from V-Memory	NO UPS	Figure A-214
LOCKD		Figure A-6
OVRDE	IGN FDBK OVRD [ ]	Figure A-215
OVRDR	IGN FDBK OVRD [ ]	Figure A-216
CMMD	EE	Figure A-217
	ER	Figure A-218
MEXTEND	EE	Figure A-219
	ER	Figure A-220
Extended Time Expired	EE	Figure A-221
	ER	Figure A-222
Retracted Time Expired	EE	Figure A-223
	ER	Figure A-224
EXTENDED	EE	Figure A-225
	ER	Figure A-226
	EE, N.O. FDBK [X]	Figure A-227
	ER, N.O. FDBK [X]	Figure A-228
	EE, IGN FDBK OVRD [X]	Figure A-229
	ER, IGN FDBK OVRD [X]	Figure A-230
	EE, N.O. FDBK [X], IGN FDBK OVRD [X]	Figure A-231
	ER, N.O. FDBK [X], IGN FDBK OVRD [X]	Figure A-232

Table A-20 CSD Extensions and Options (continued)

Extension	Option	Figure
RETRACTED	EE	Figure A-233
	ER	Figure A-234
	EE, N.O. FDBK [X]	Figure A-235
	ER, N.O. FDBK [X]	Figure A-236
	EE, IGN OVRD [X]	Figure A-237
	ER, IGN OVRD [X]	Figure A-238
	EE, N.O. FDBK [X], IGN FDBK OVRD [X]	Figure A-239
	ER, N.O. FDBK [X], IGN FDBK OVRD [X]	Figure A-240
TRVL	EE	Figure A-241
	ER	Figure A-242
FTE		Figure A-243
FTE	IGN FDBK OVRD [X]	Figure A-244
FTR		Figure A-245
FTR	IGN FDBK OVRD [X]	Figure A-246
FAILD		Figure A-247
Move image to V-Memory	NO UPS	Figure A-248

Device Extensions and Table Options (continued)

CUD

The extensions and options for the CUD device are listed in Table A-21 and illustrated in the figures specified in the table.

Table A-21 CUD Extensions and Options

Extension	Option	Figure
EXTTO		Figure A-249
RETTO		Figure A-250

Press

The extensions and options for the Press devices are exactly like the Valve devices with different extensions. Refer to the Valve section for examples on how the RLL looks for Presses.

A.2 RLL for Devices

Figure A-1 VND: OPND

Figure A-2 VND: CLSD

Figure A-3 VND: FAILD

Figure A-4 VND: MOPEN

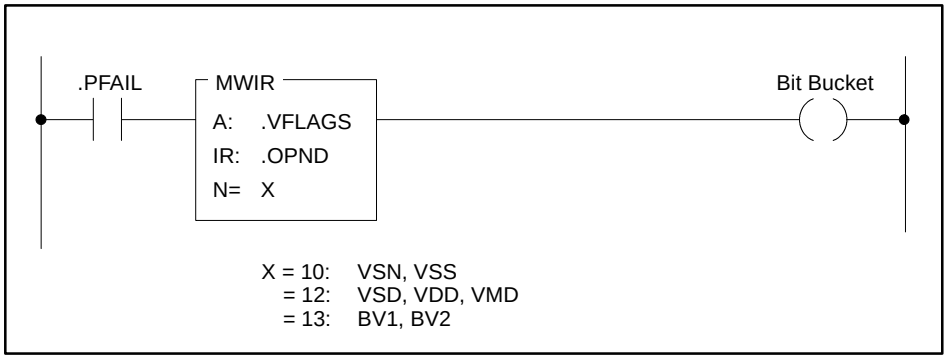


Figure A-5 VALVES: Move Image from V

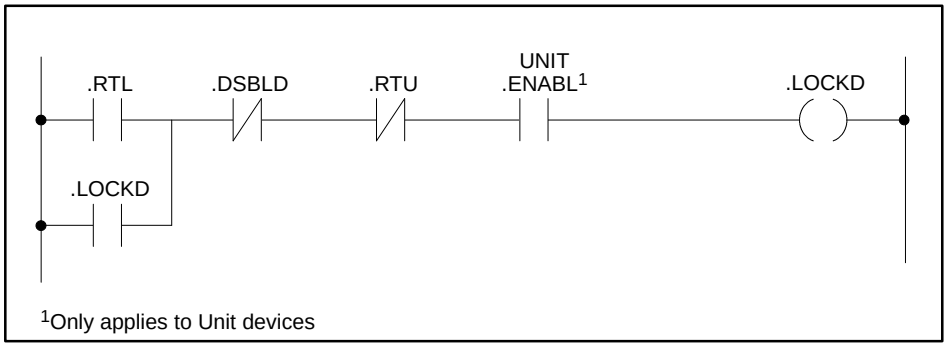


Figure A-6 ALL: LOCKD

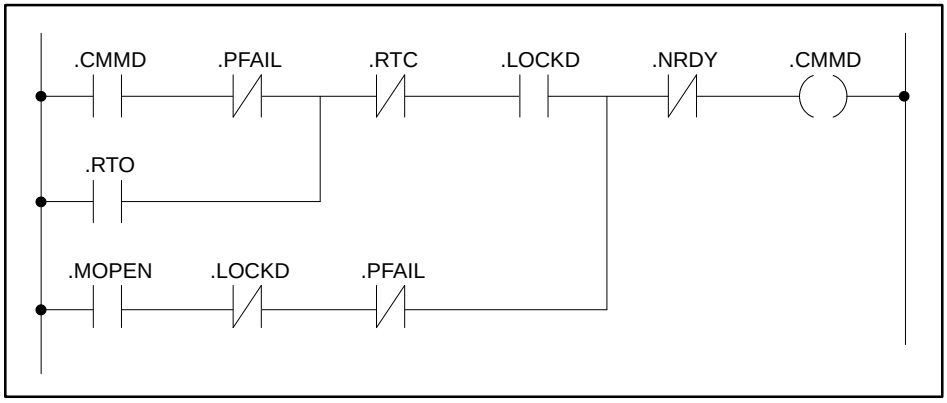


Figure A-7 VSN: CMMD (EO)

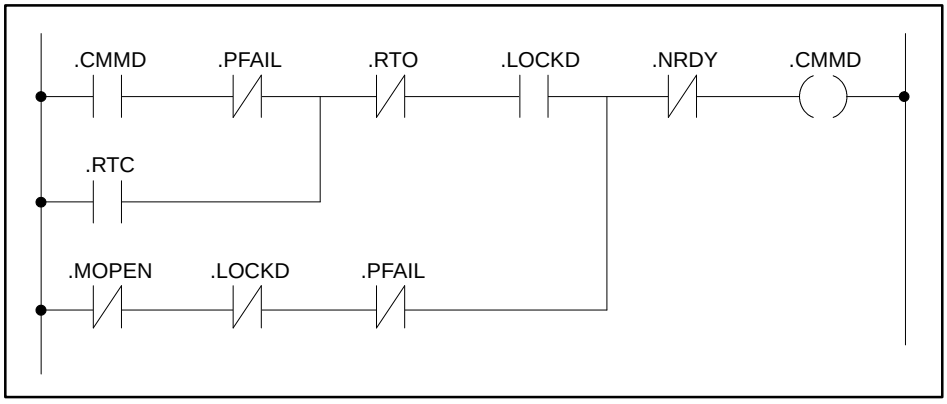


Figure A-8 VSN: CMMD (EC)

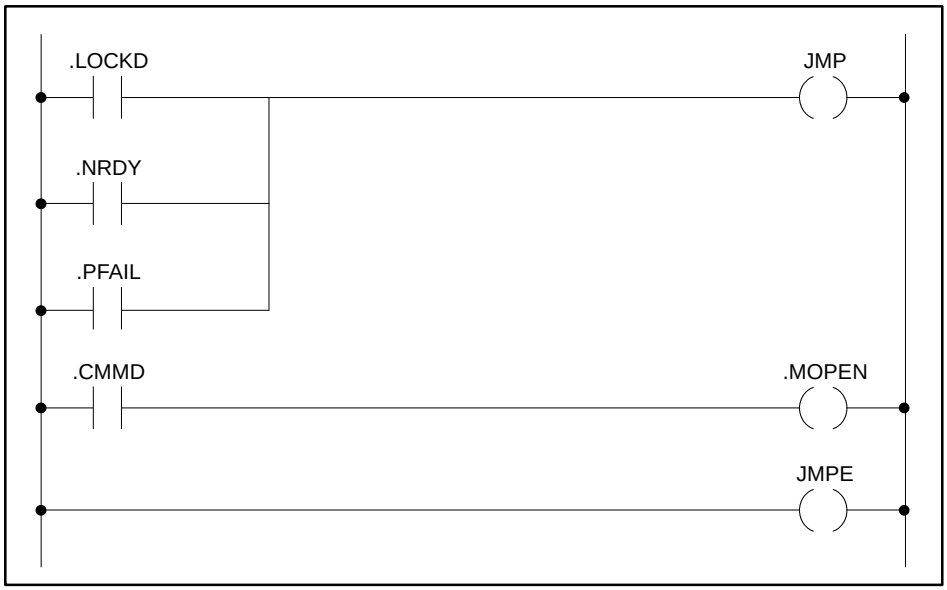


Figure A-9 VSN: MOPEN (EO)

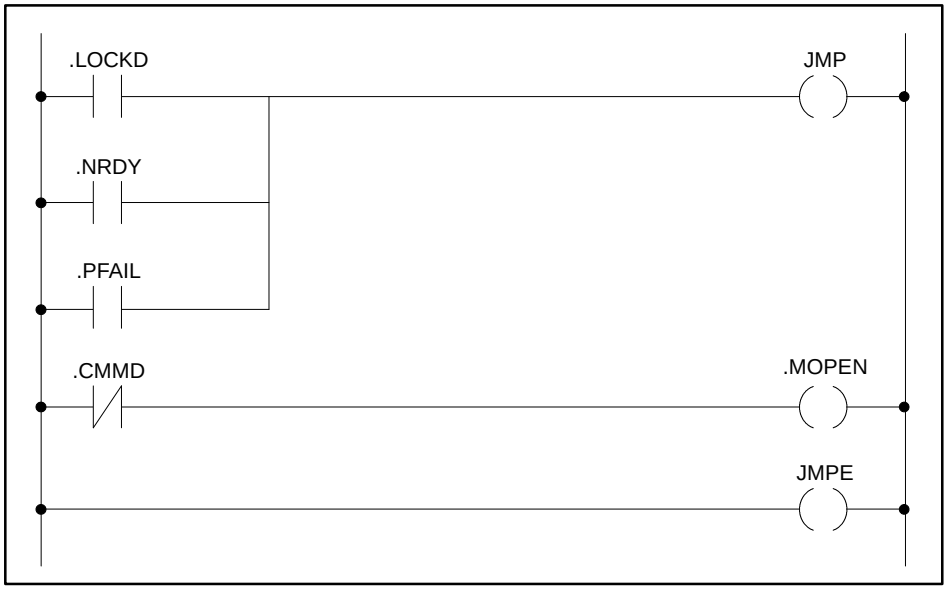


Figure A-10 VSN: MOPEN (EC)

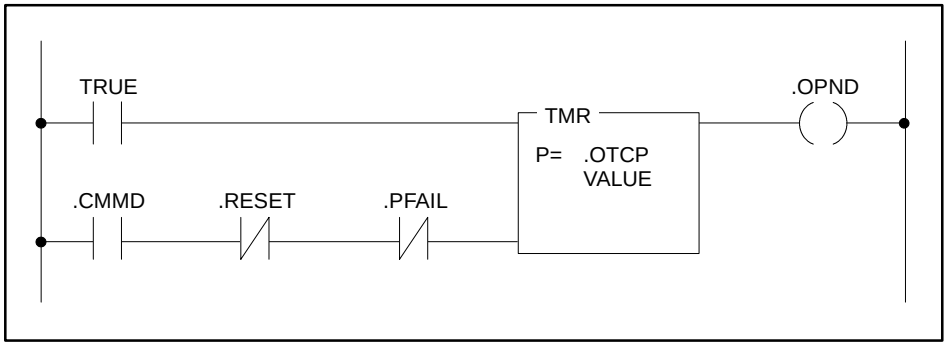


Figure A-11 VSN: OPND (EO)

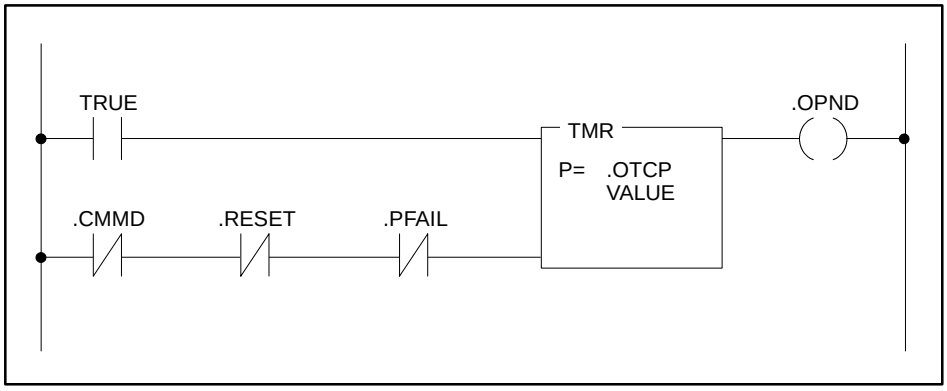


Figure A-12 VSN: OPND (EC)

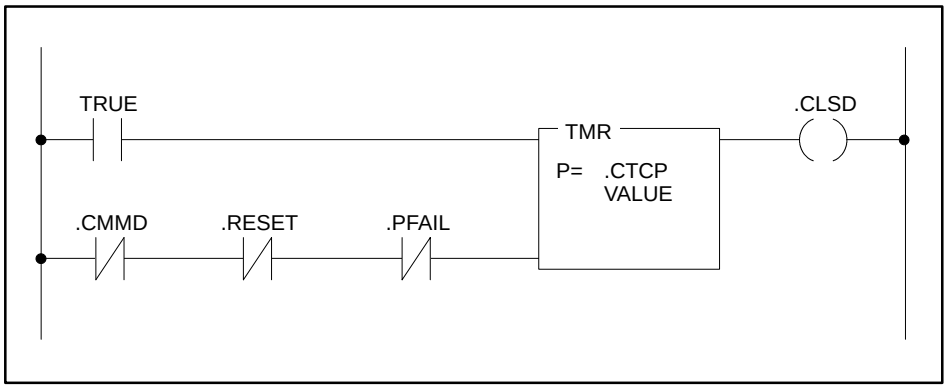


Figure A-13 VSN: CLSD (EO)

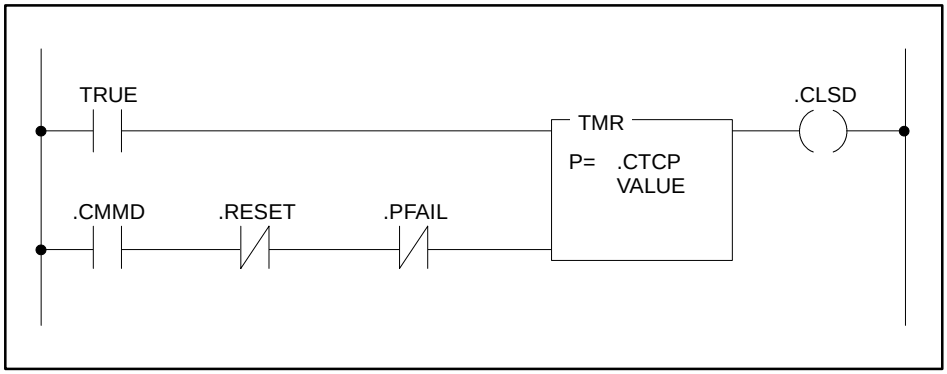


Figure A-14 VSN: CLSD (EC)

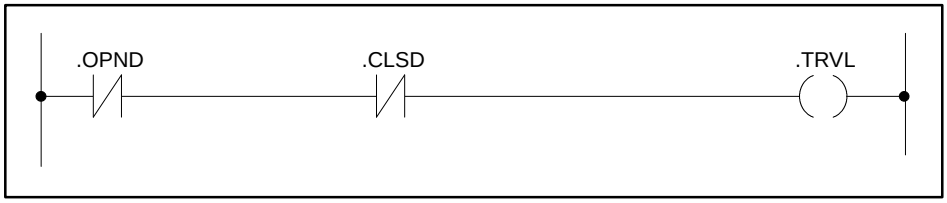


Figure A-15 VSN: TRVL

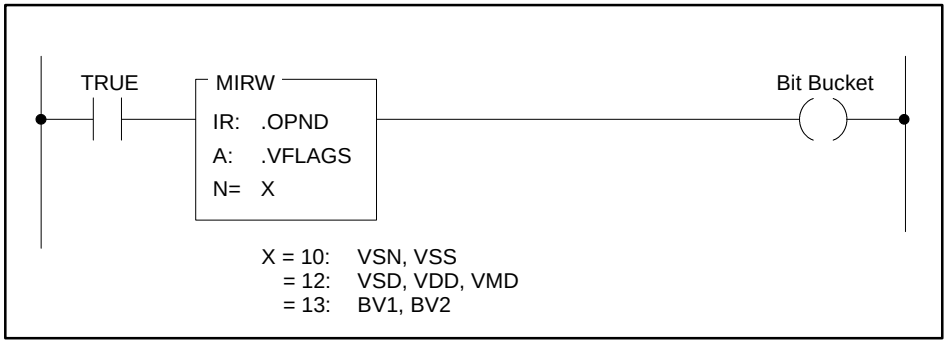


Figure A-16 VALVES: Move Image to V

Figure A-17 VSS: OVRD

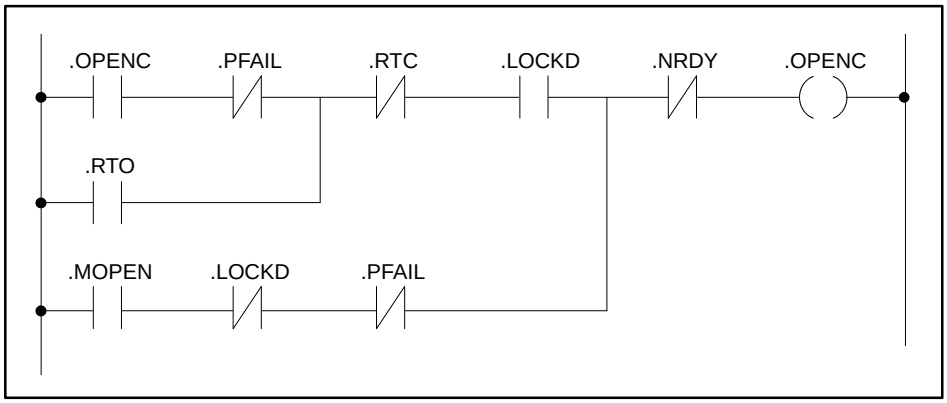


Figure A-18 VSS: OPENC (EO)

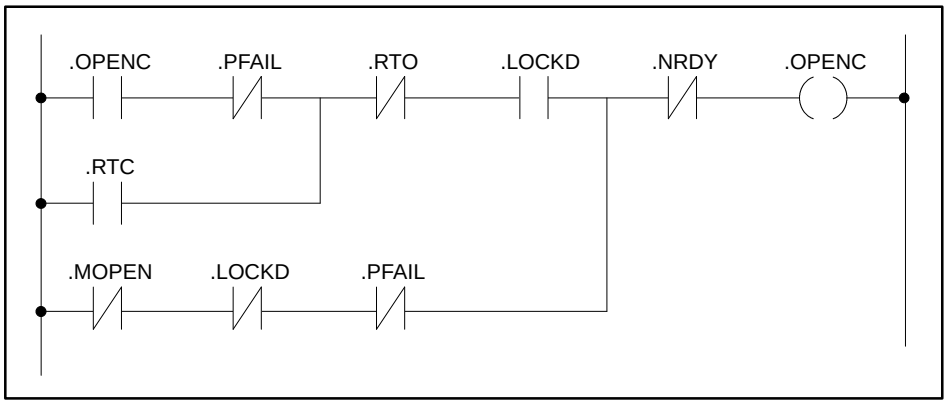


Figure A-19 VSS: OPENC (EC)

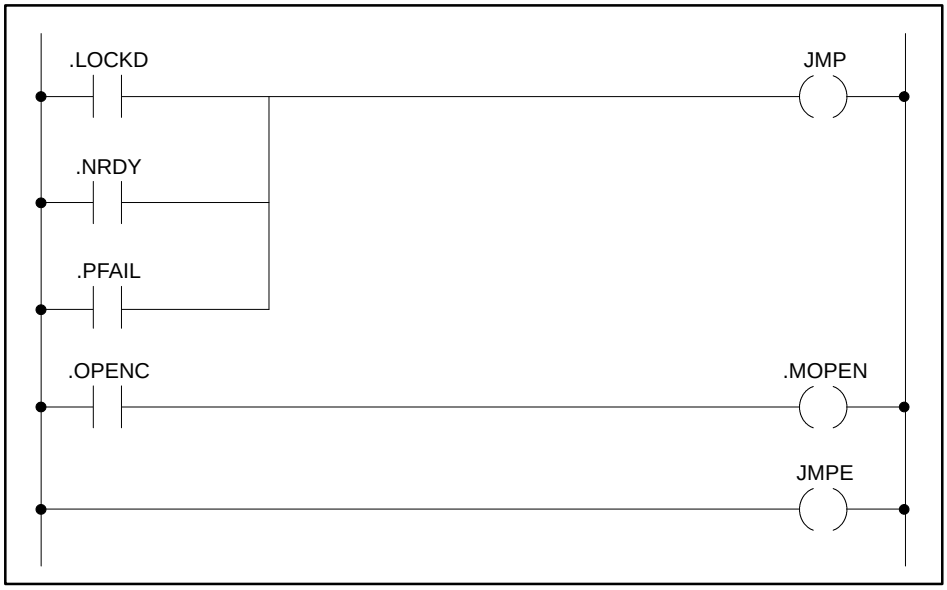


Figure A-20 VSS: MOPEN (EO)

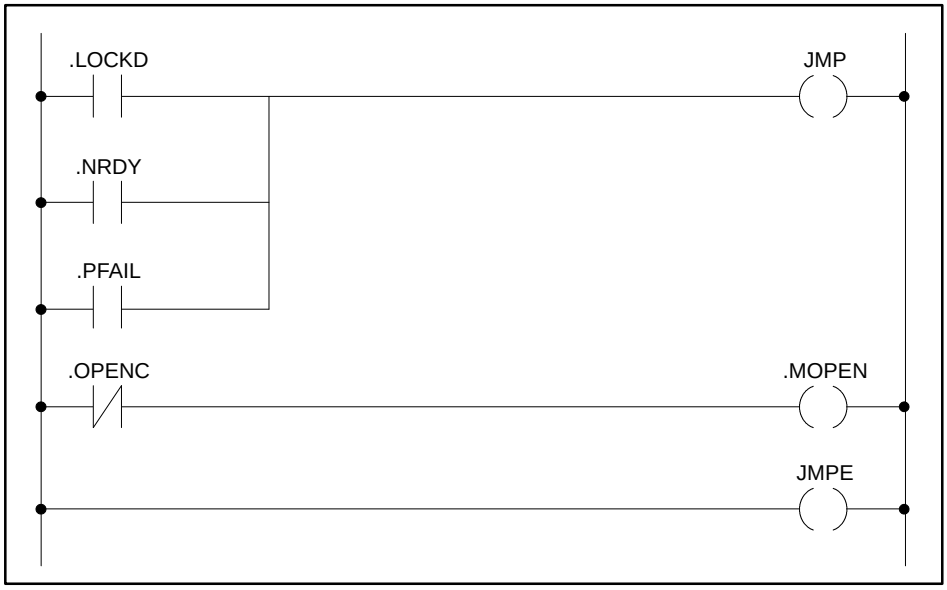


Figure A-21 VSS: MOPEN (EC)

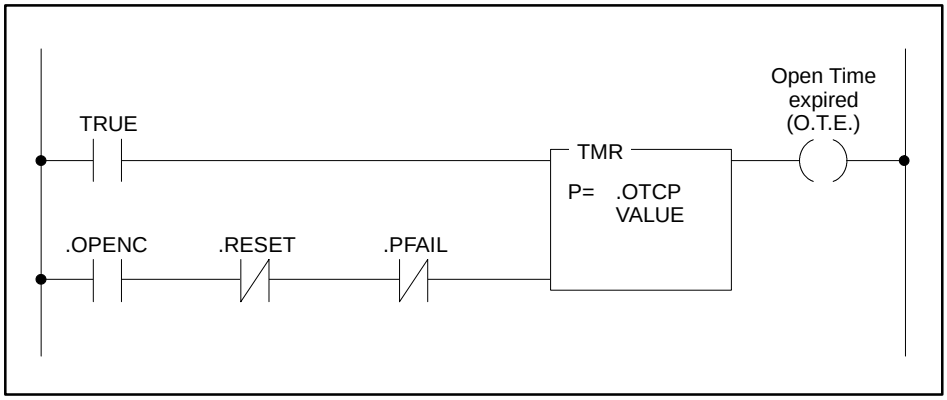


Figure A-22 VSS: OPEN TIME EXPIRED [O.T.E.] (EO)

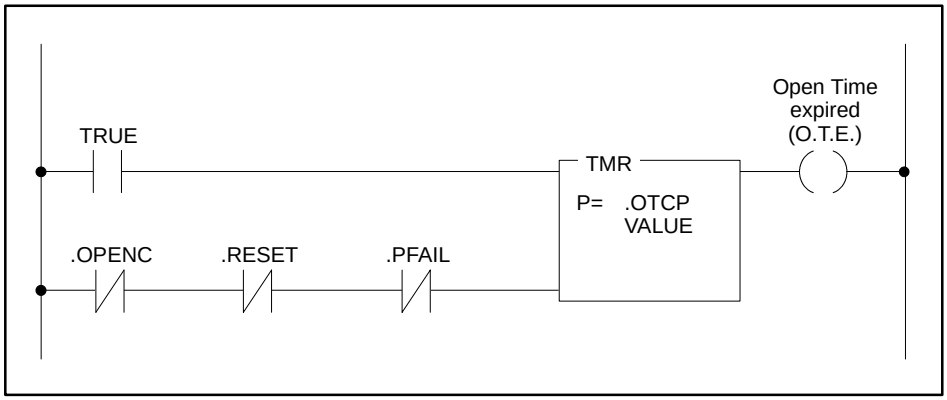


Figure A-23 VSS: OPEN TIME EXPIRED [O.T.E.] (EC)

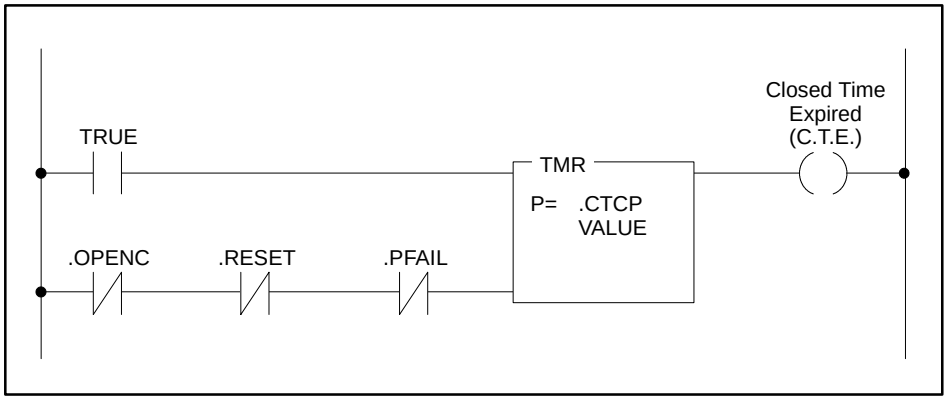


Figure A-24 VSS: CLOSED TIME EXPIRED [C.T.E.] (EO)

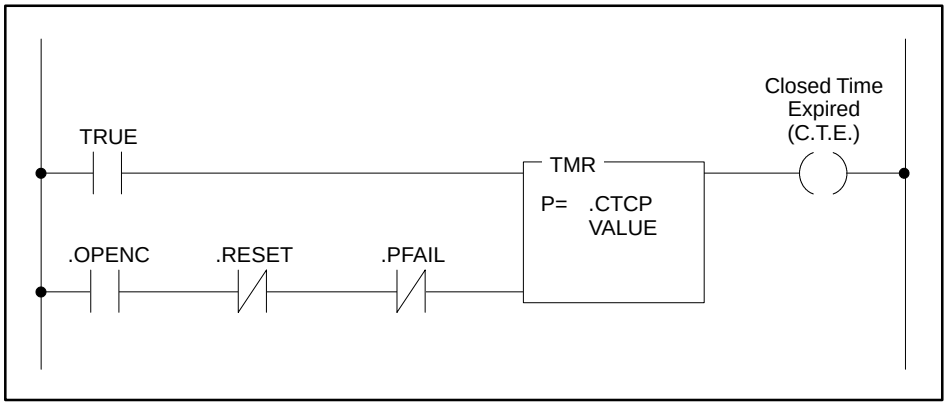


Figure A-25 VSS: CLOSED TIME EXPIRED [C.T.E.] (EC)

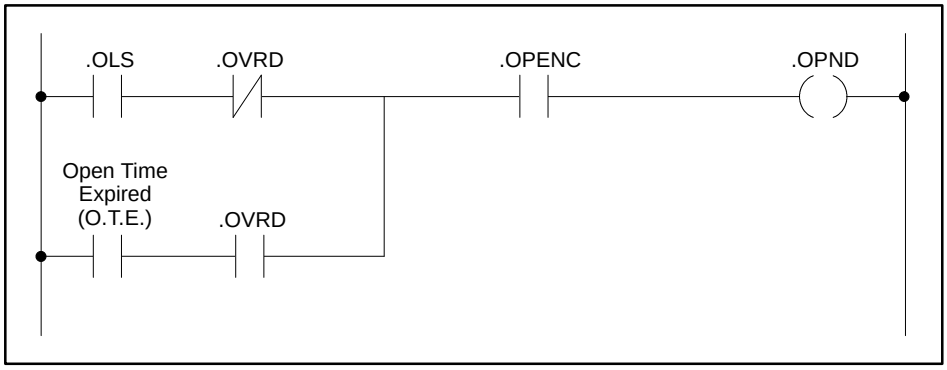


Figure A-26 VSS: OPND (EO)

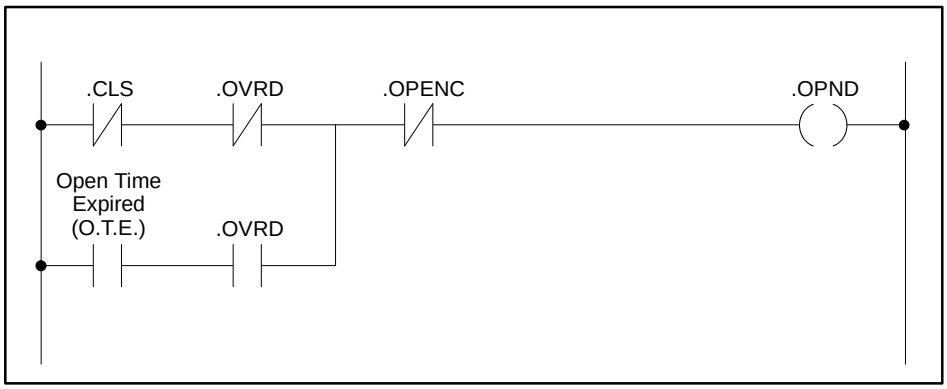


Figure A-27 VSS: OPND (EC)

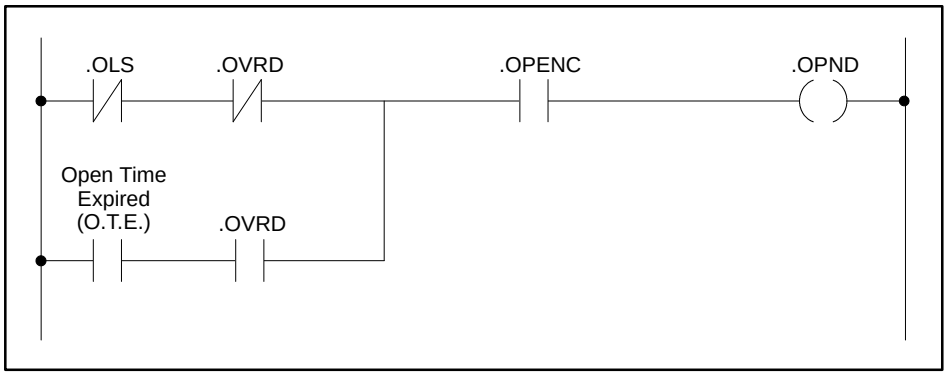


Figure A-28 VSS: OPND (EO) (N.O. FDBK)

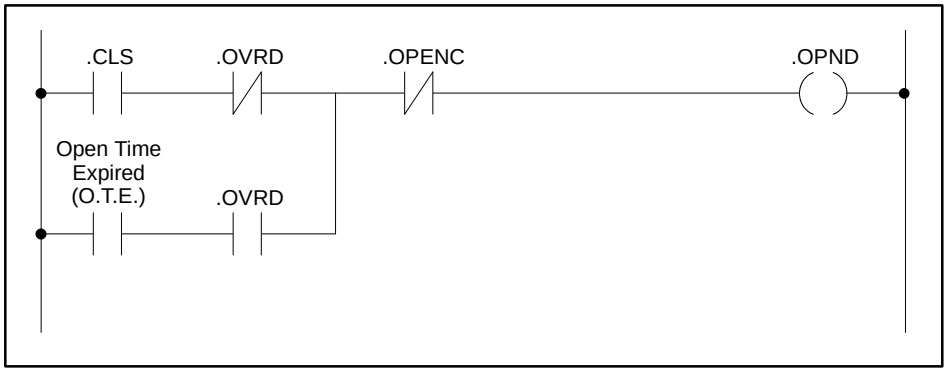


Figure A-29 VSS: OPND (EC) (N.O. FDBK)

Figure A-30 VSS: OPND (EO) (IGNORE FDBK OVRD)

Figure A-31 VSS: OPND (EC) (IGNORE FDBK OVRD)

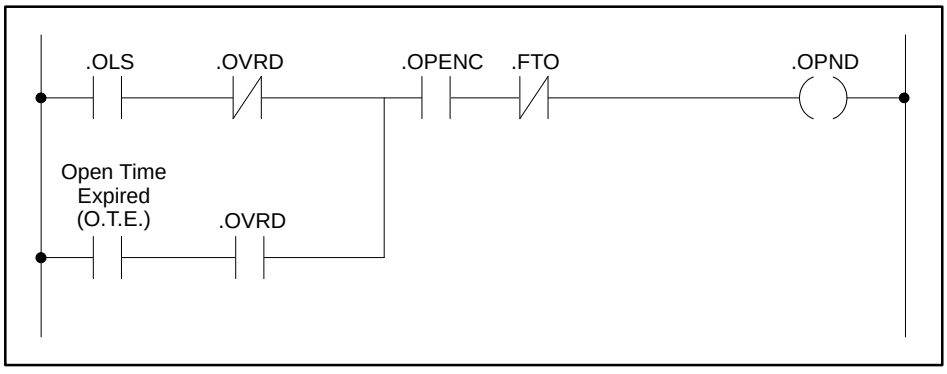


Figure A-32 VSS: OPND (EO) (CLEAR CMMD ON FTO/FTC)

Figure A-33 VSS: OPND (EO) (N.O. FDBK) (IGNORE FDBK OVRD)

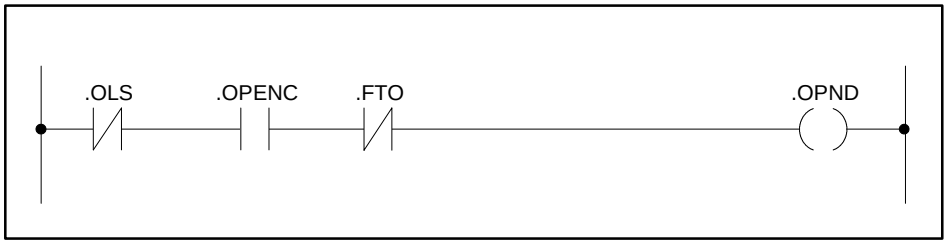


Figure A-34 VSS: OPND (EO) (N.O. FDBK) (IGNORE FDBK OVRD) (CLEAR CMMD ON FTO/FTC)

Figure A-35 VSS: OPND (EC) (N.O. FDBK) (IGNORE FDBK OVRD)

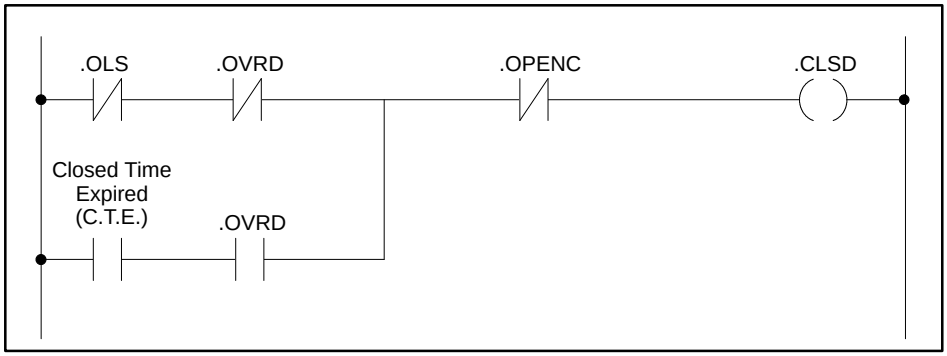


Figure A-36 VSS: CLSD (EO)

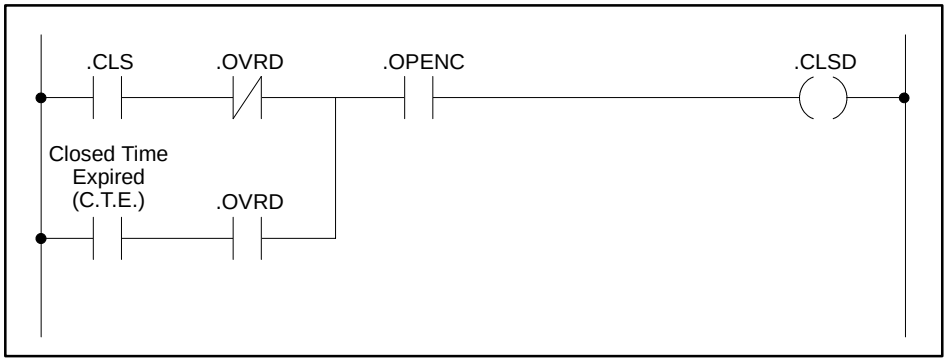


Figure A-37 VSS: CLSD (EC)

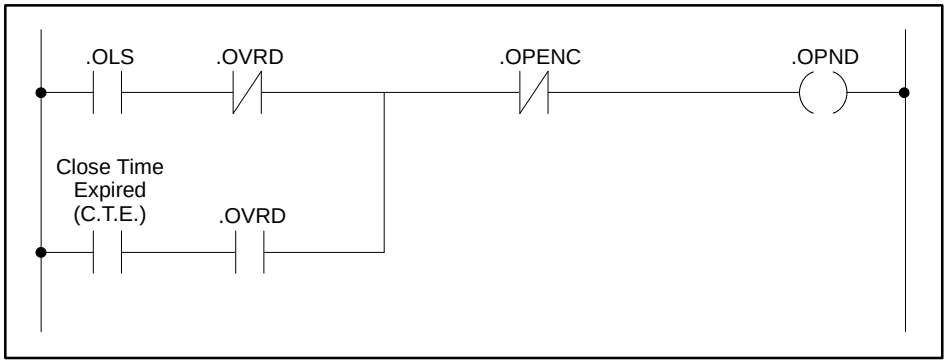


Figure A-38 VSS: CLSD (EO) (N.O. FDBK)

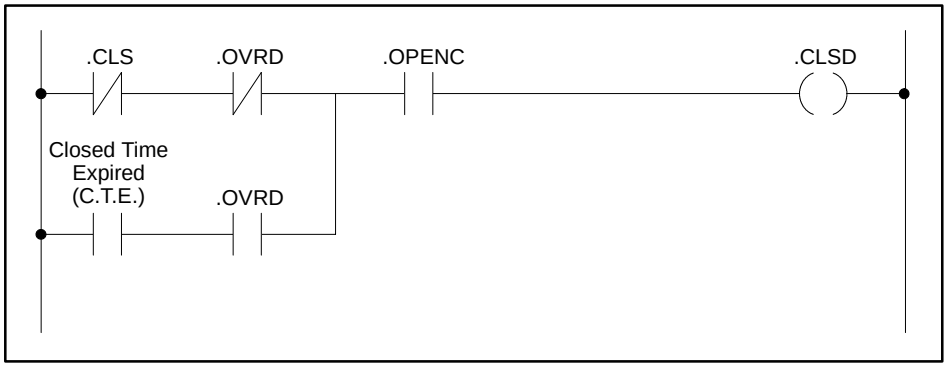


Figure A-39 VSS: CLSD (EC) (N.O. FDBK)

Figure A-40 VSS: CLSD (EO) (IGNORE FDBK OVRD)

Figure A-41 VSS: CLSD (EC) (IGNORE FDBK OVRD)

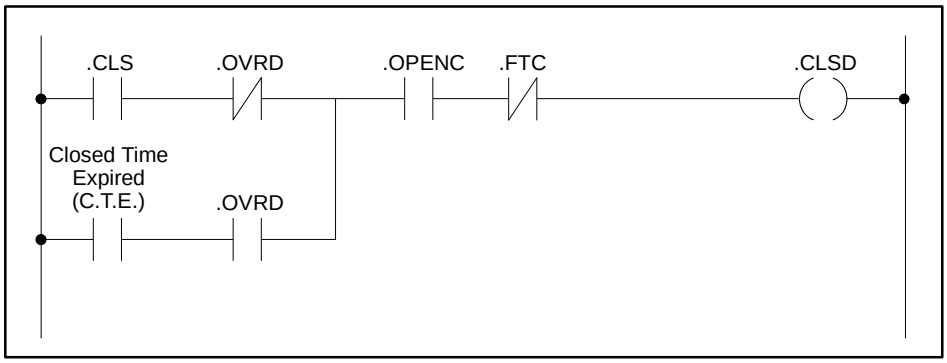


Figure A-42 VSS: CLSD (EC) (CLEAR CMMD ON FTO/FTC)

Figure A-43 VSS: CLSD (EO) (N.O. FDBK) (IGNORE FDBK OVRD)

Figure A-44 VSS: CLSD (EC) (N.O. FDBK) (IGNORE FDBK OVRD)

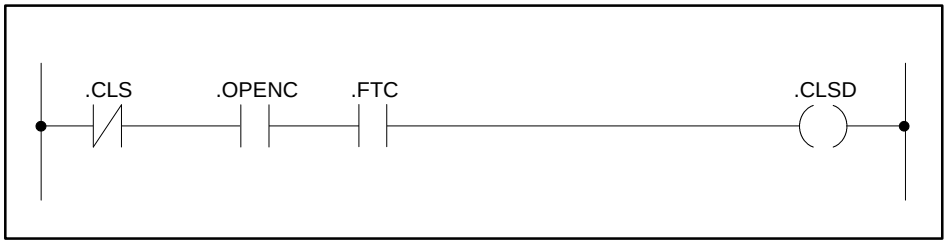


Figure A-45 VSS: CLSD (EC) (N.O. FDBK) (IGNORE FDBK OVRD) (CLEAR CMMD OF FTO/FTC)

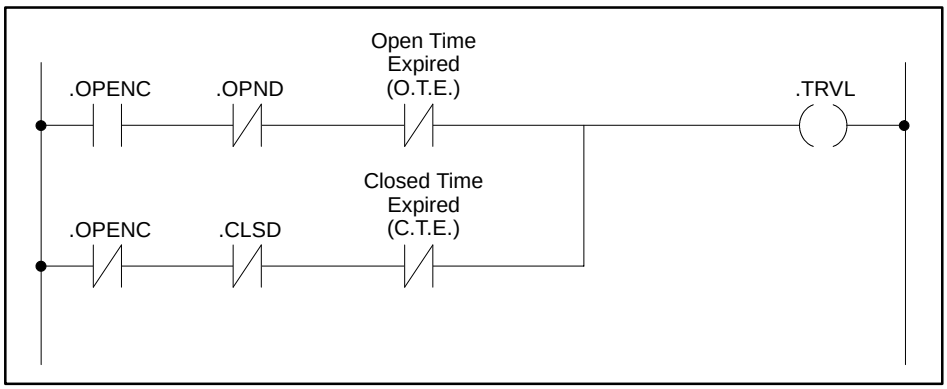


Figure A-46 VSS: TRVL (EO)

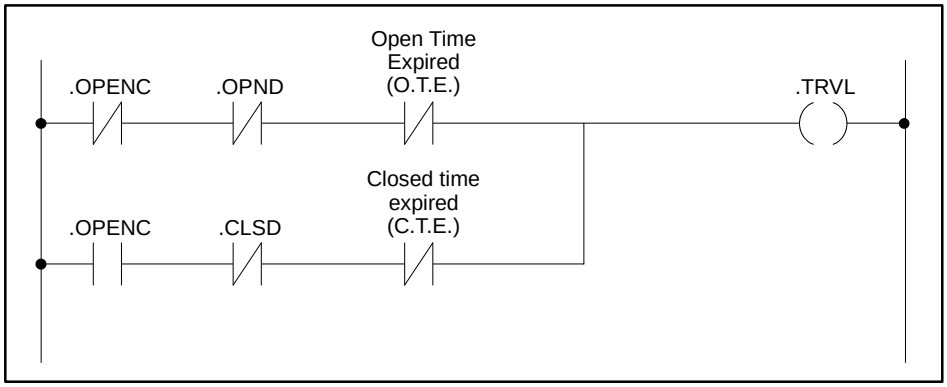


Figure A-47 VSS: TRVL (EC)

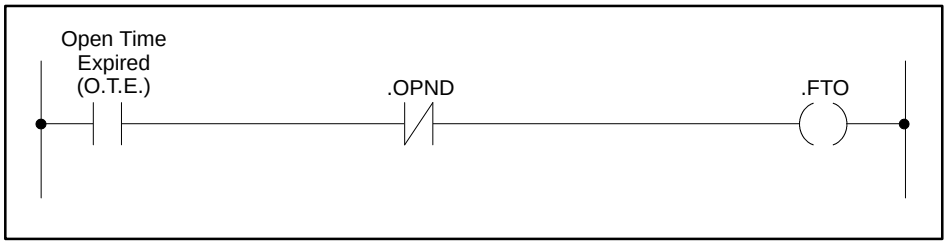


Figure A-48 VSS: FTO

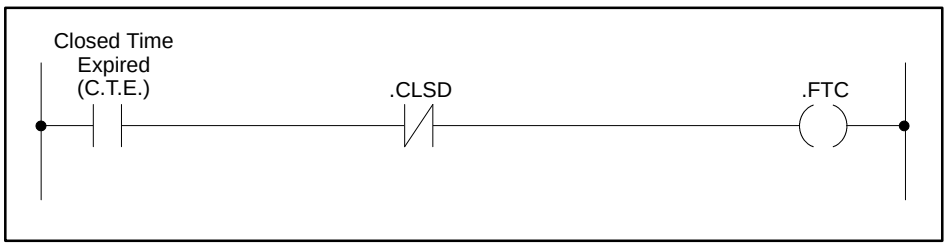


Figure A-49 VSS: FTC

Figure A-50 VSS: CMMD (EO/EC)

Figure A-51 VSS: CMMD (EO) (CLEAR CMMD ON FTO/FTC)

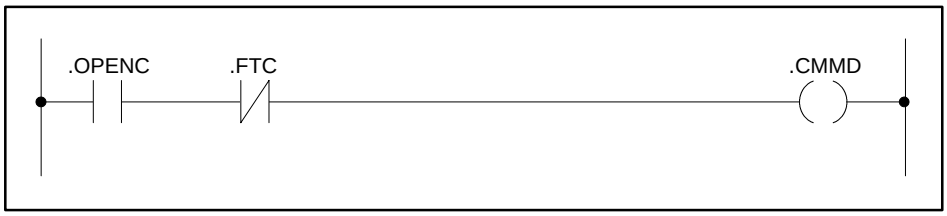


Figure A-52 VSS: CMMD (EC) (CLEAR CMMD ON FTO/FTC)

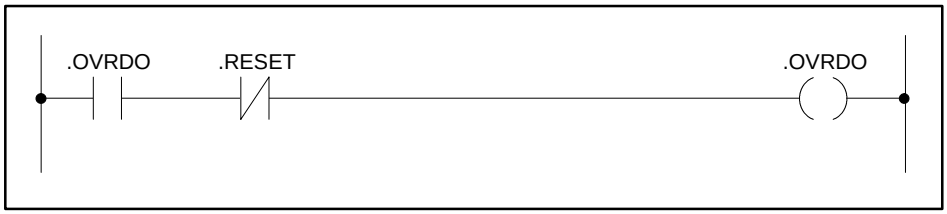


Figure A-53 VSD: OVRDO

Figure A-54 VSD: OVRDC

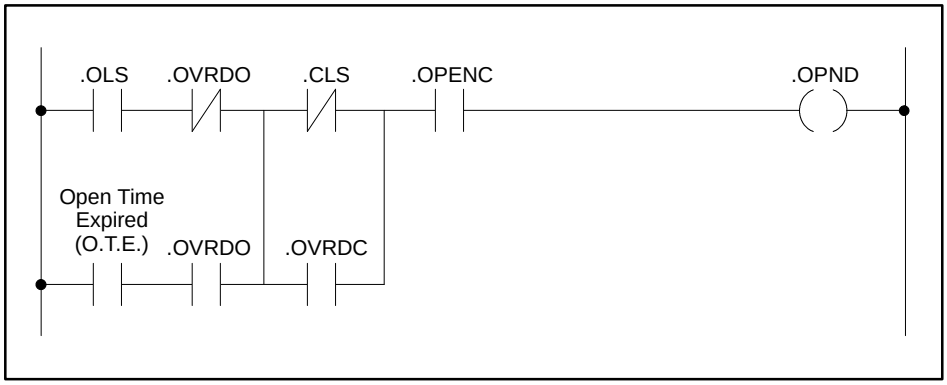


Figure A-55 VSD: OPND (EO)

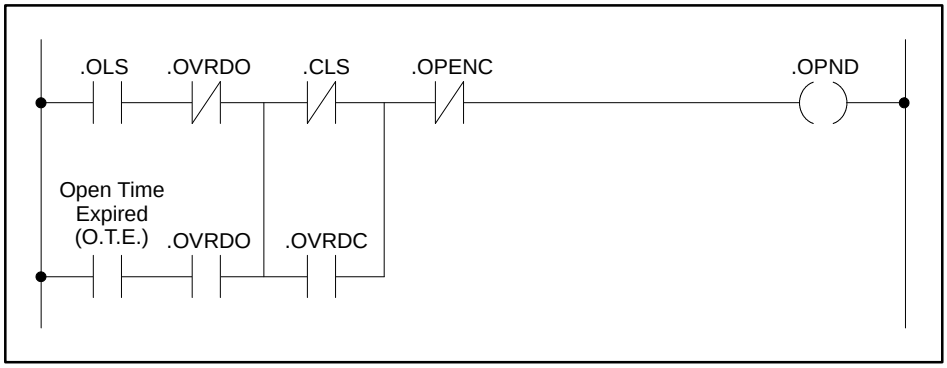


Figure A-56 VSD: OPND (EC)

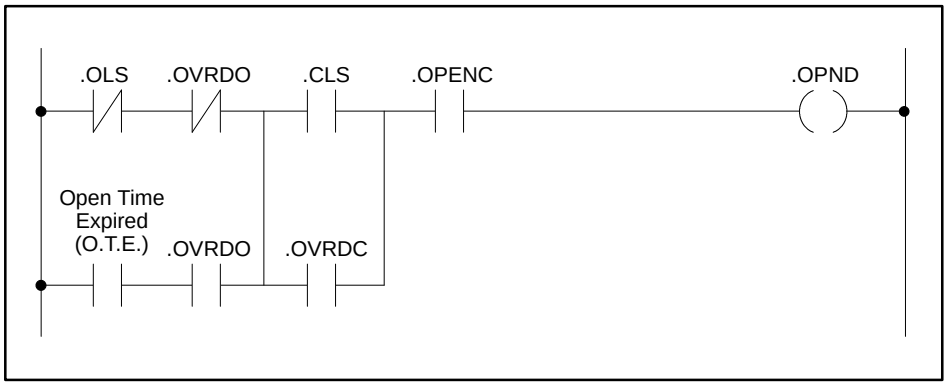


Figure A-57 VSD: OPND (EO) (N.O. FDBK)

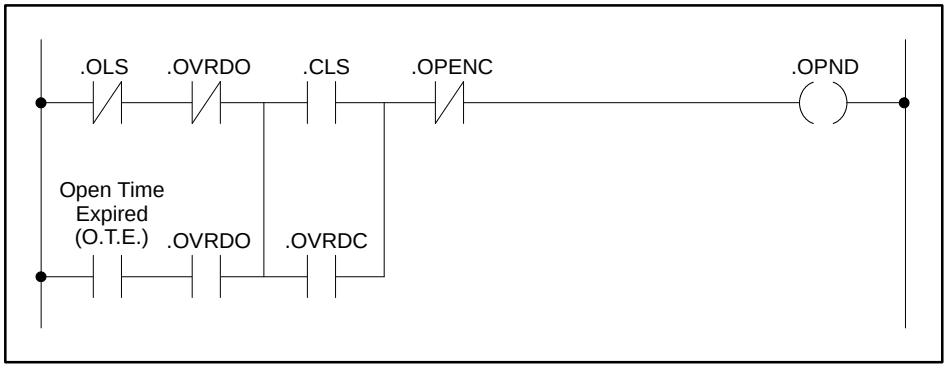


Figure A-58 VSD: OPND (EC) (N.O. FDBK)

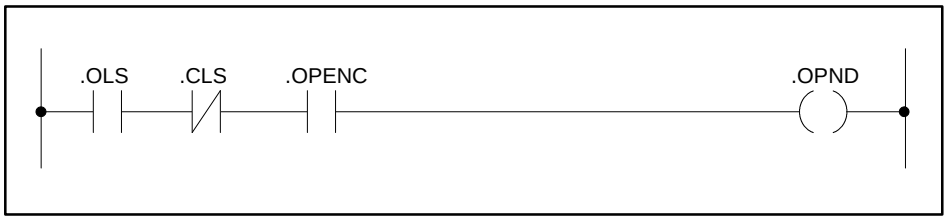


Figure A-59 VSD: OPENC (EO) (IGNORE FDBK OVRD)

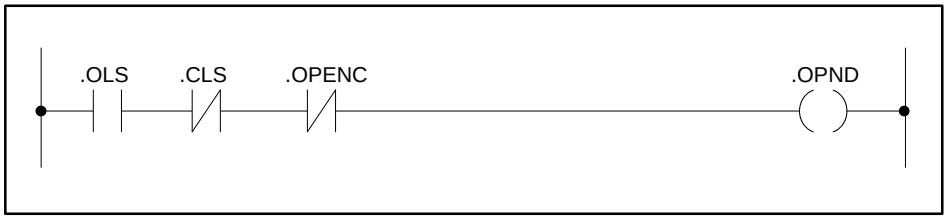


Figure A-60 VSD: OPND (EC) (IGNORE FDBK OVRD)

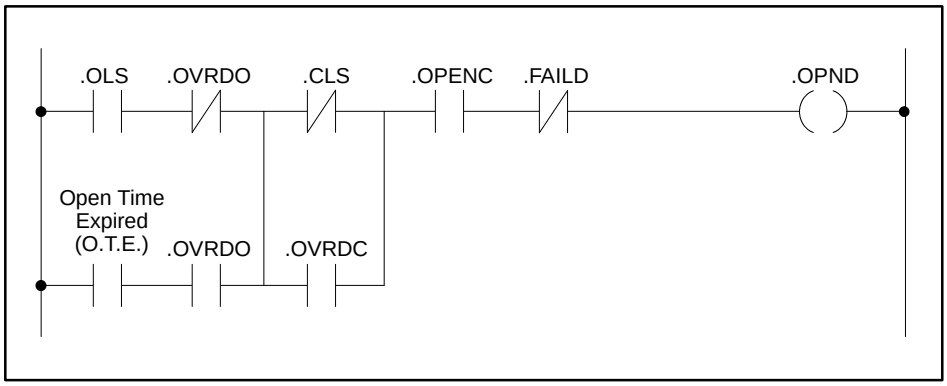


Figure A-61 VSD: OPND (EO) (CLEAR CMMD ON FTO/FTC)

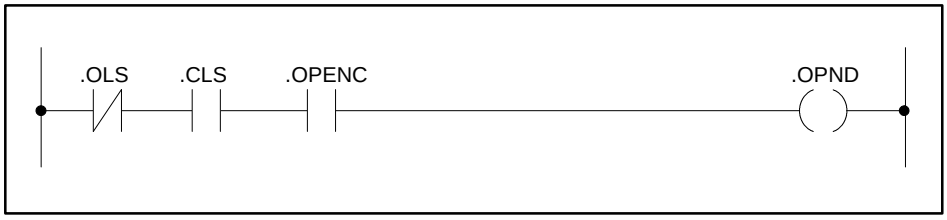


Figure A-62 VSD: OPND (EO) (N.O. FDBK) (IGNORE FDBK OVRD)

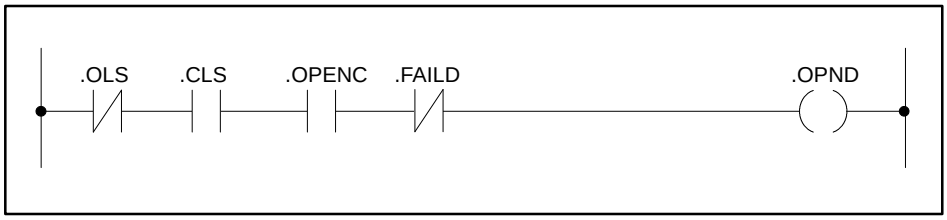


Figure A-63 VSD: OPND (EO) (N.O. FDBK) (IGNORE FDBK OVRD) (CLEAR CMMD ON FTO/FTC)

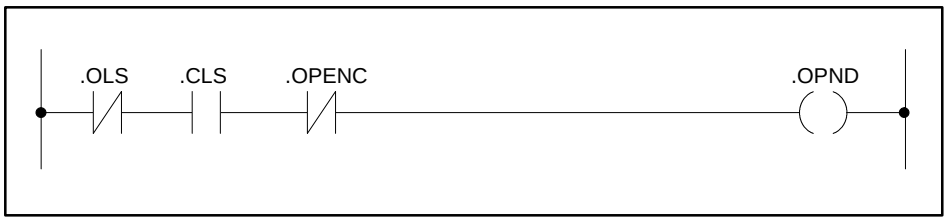


Figure A-64 VSD: OPND (EC) (N.O. FDBK) (IGNORE FDBK OVRD)

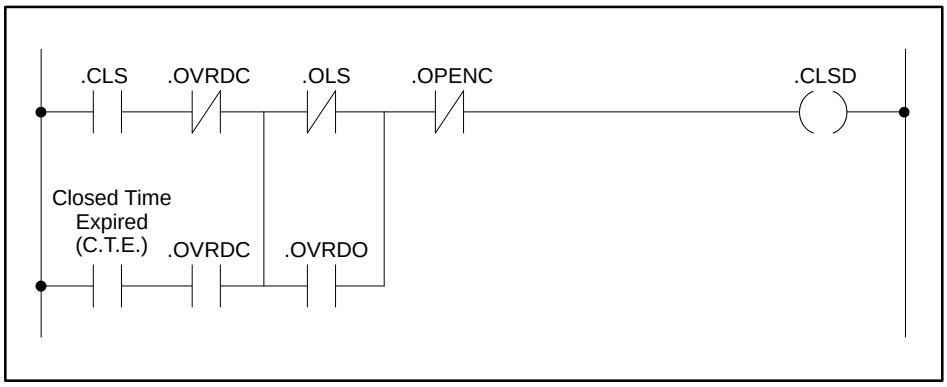


Figure A-65 VSD: CLSD (EO)

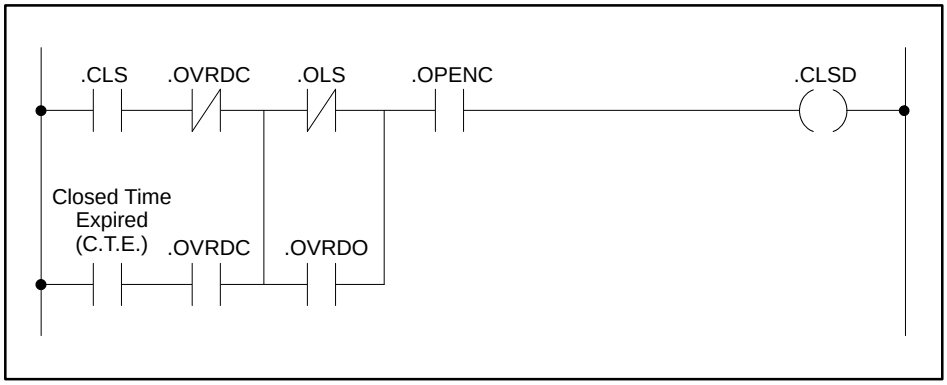


Figure A-66 VSD: CLSD (EC)

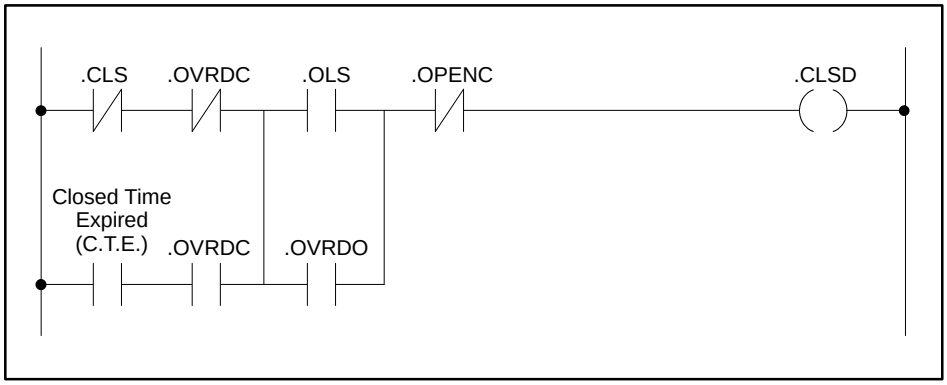


Figure A-67 VSD: CLSD (EO) (N.O. FDBK)

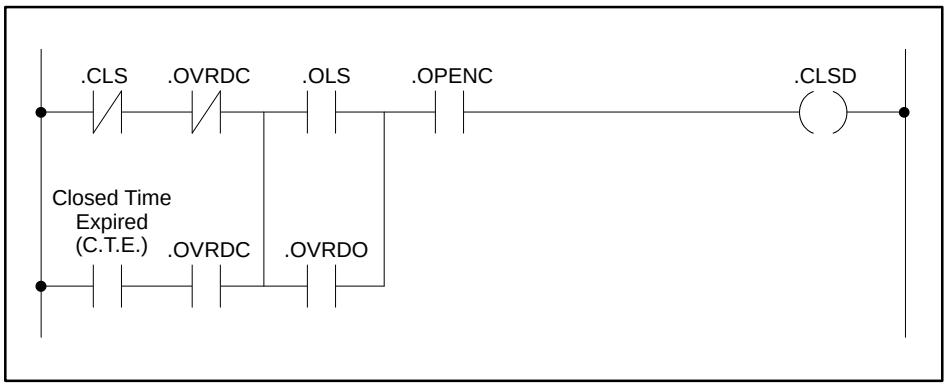


Figure A-68 VSD: CLSD (EC) (N.O. FDBK)

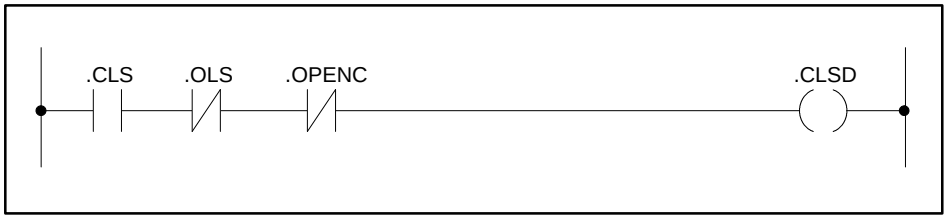


Figure A-69 VSD: CLSD (EO) (IGNORE FDBK OVRD)

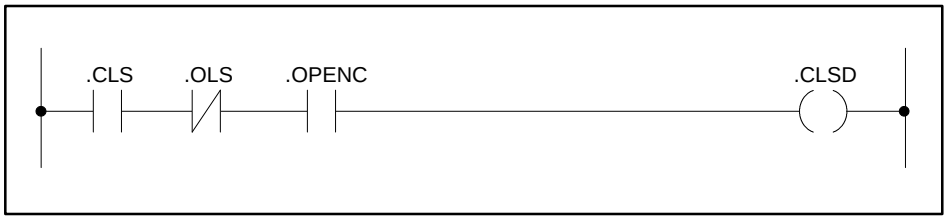


Figure A-70 VSD: CLSD (EC) (IGNORE FDBK OVRD)

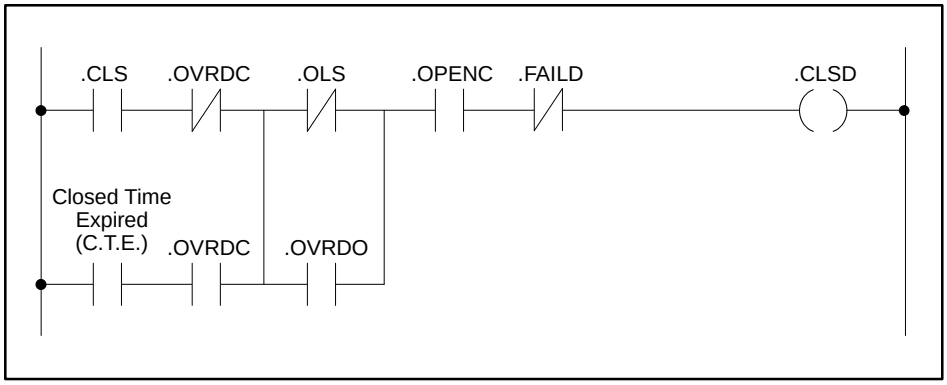


Figure A-71 VSD: CLSD (EC) (CLEAR CMMD ON FTO/FTC)

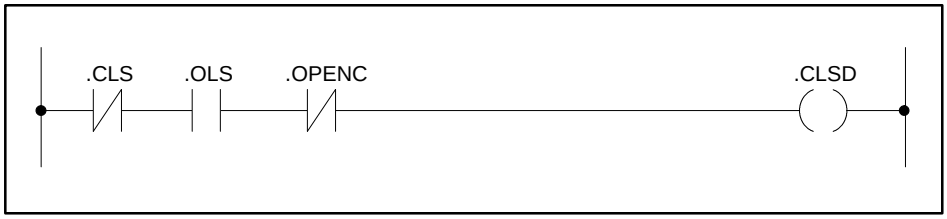


Figure A-72 VSD: CLSD (EO) (N.O. FDBK) (IGNORE FDBK OVRD)

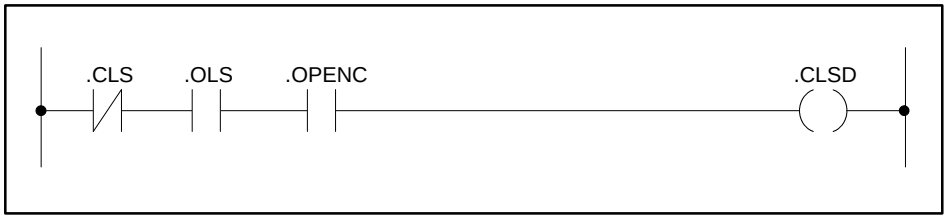


Figure A-73 VSD: CLSD (EC) (N.O. FDBK) (IGNORE FDBK OVRD)

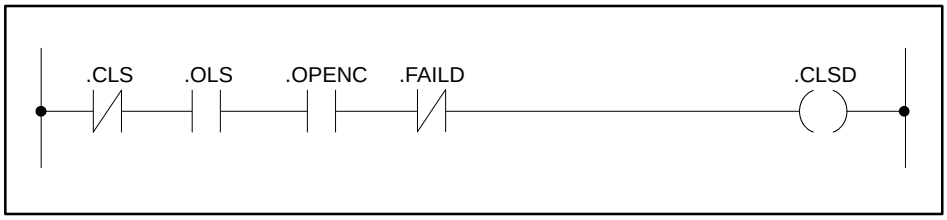


Figure A-74 VSD: CLSD (EC) (N.O. FDBK) (IGNORE FDBK OVRD) (CLEAR CMMD ON FTO/FTC)

Figure A-75 VSD: FAILD

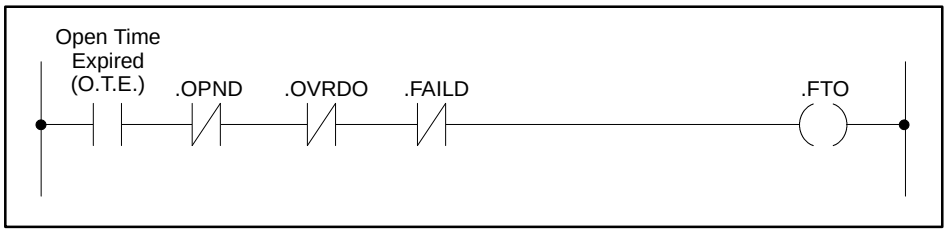


Figure A-76 VSD: FTO

Figure A-77 VSD: FTO (IGNORE FDBK OVRD)

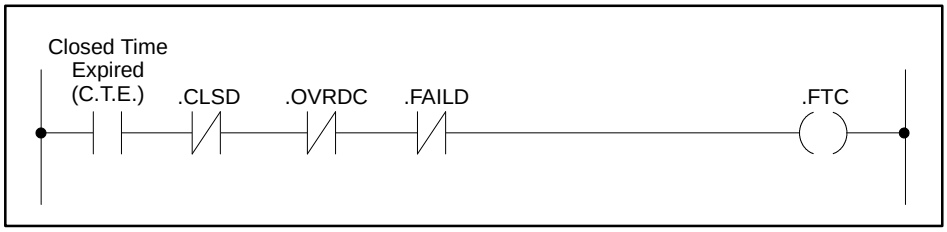


Figure A-78 VSD: FTC

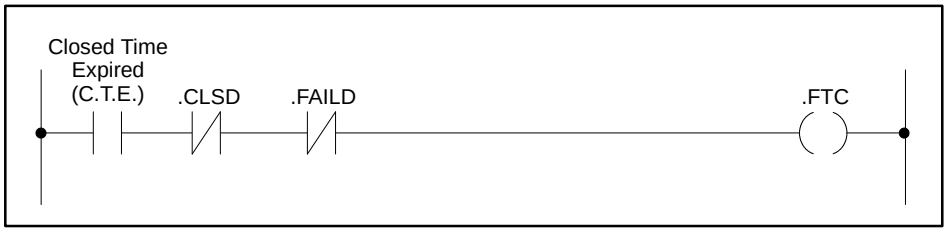


Figure A-79 VSD: FTC (IGNORE FDBK OVRD)

Figure A-80 VSD: CMMD (EO)

Figure A-81 VSD: CMMD (EO) (CLEAR CMMD ON FTO/FTC)

Figure A-82 VSD: CMMD (EC) (CLEAR CMMD ON FTO/FTC)

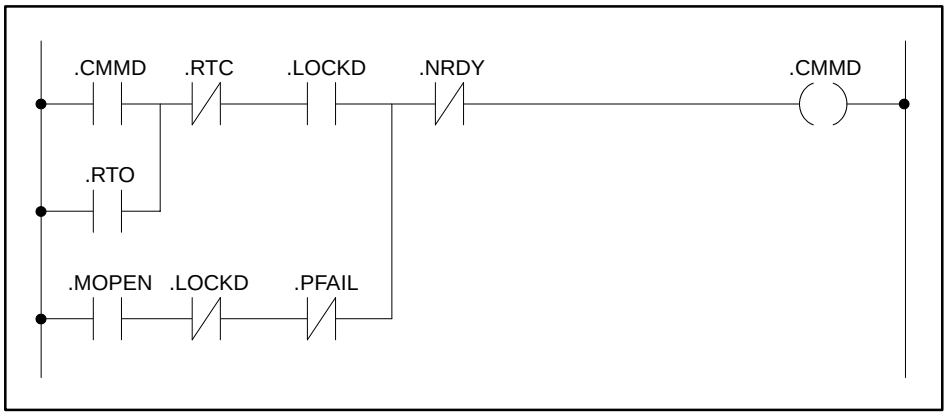


Figure A-83 VDD: CMMD

Figure A-84 VDD: OPENC

Figure A-85 VDD: CLSC

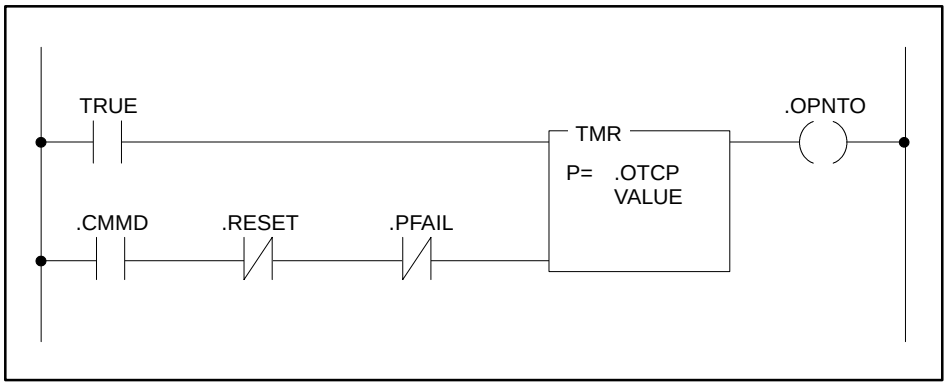


Figure A-86 VDD: OPNTO

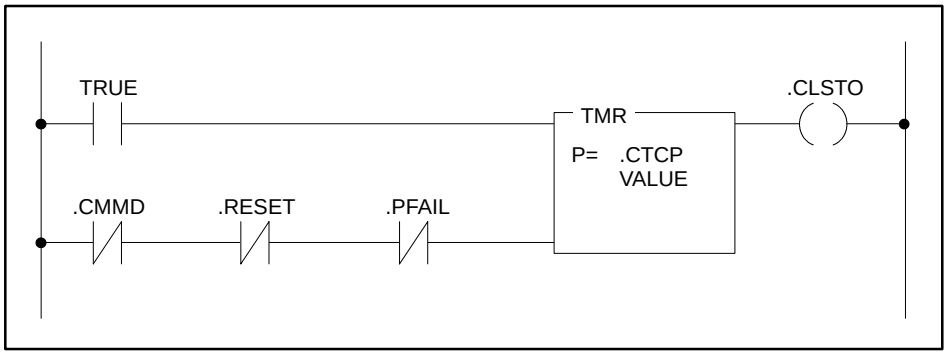


Figure A-87 VDD: CLSTO

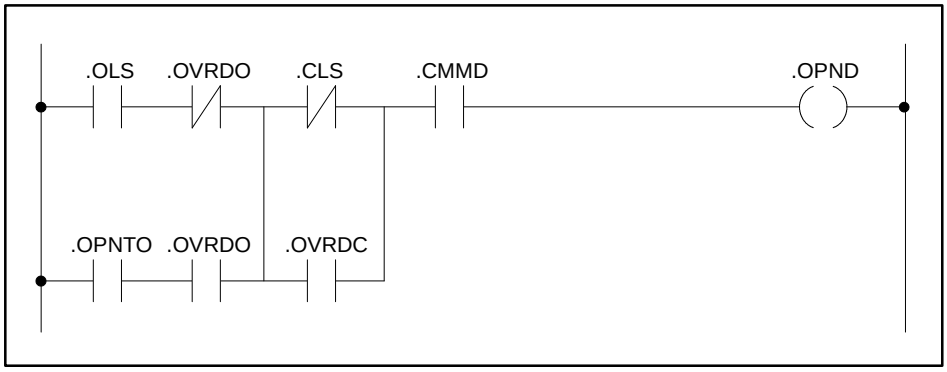


Figure A-88 VDD: OPND

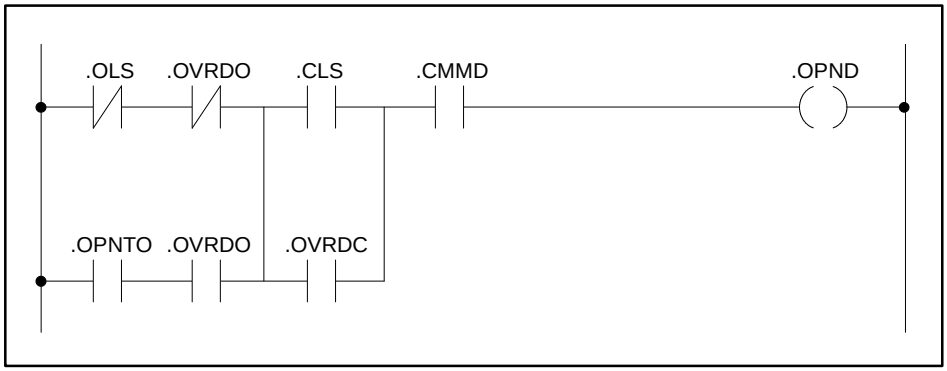


Figure A-89 VDD: OPND (N.O. FDBK)

Figure A-90 VDD: OPND (IGNORE FDBK OVRD)

Figure A-91 VDD: OPND (N.O. FDBK) (IGNORE FDBK OVRD)

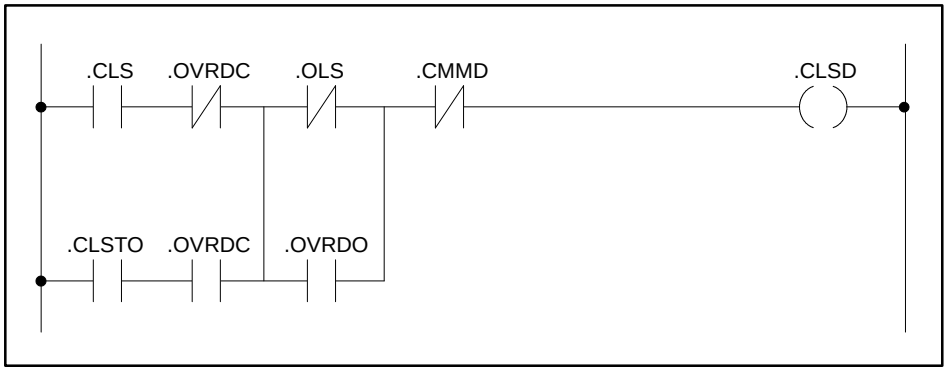


Figure A-92 VDD: CLSD

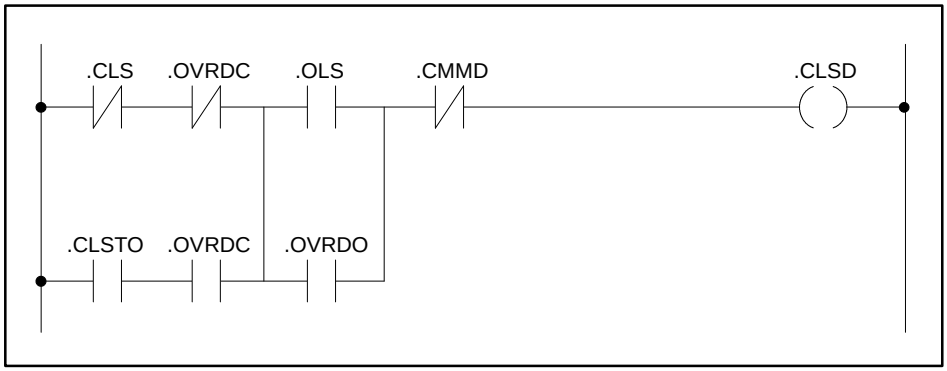


Figure A-93 VDD: CLSD (N.O. FDBK)

Figure A-94 VDD: CLSD (IGNORE FDBK OVRD)

Figure A-95 VDD: CLSD (N.O. FDBK) (IGNORE FDBK OVRD)

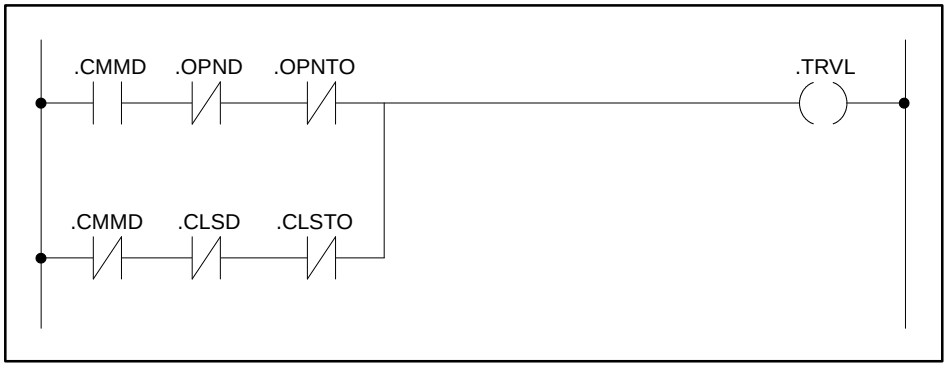


Figure A-96 VDD: TRVL

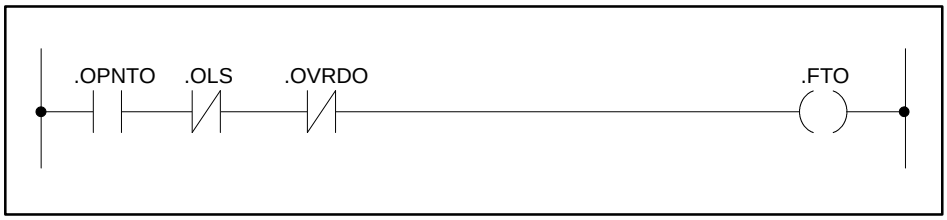


Figure A-97 VDD: FTO

Figure A-98 VSD: FTC

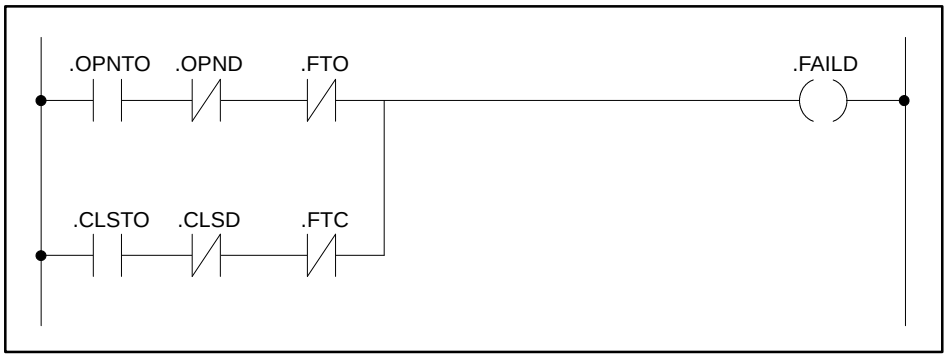


Figure A-99 VDD: FAILD

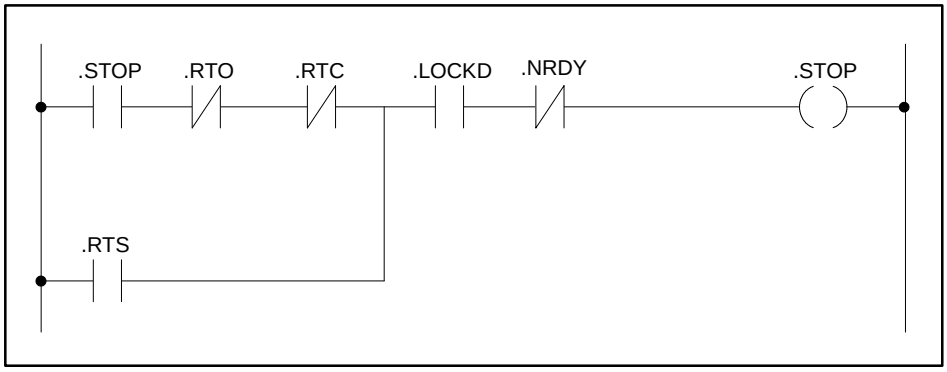


Figure A-100 VMD: STOP

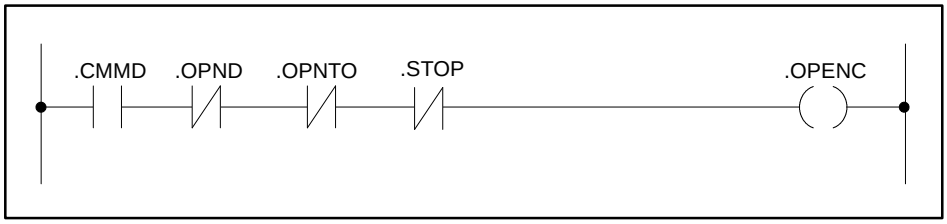


Figure A-101 VMD: OPENC

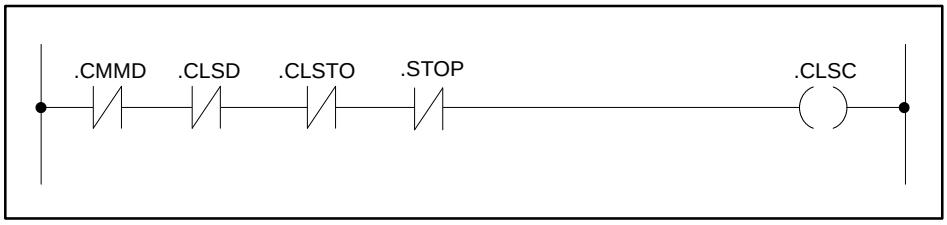


Figure A-102 VMD: CLSC

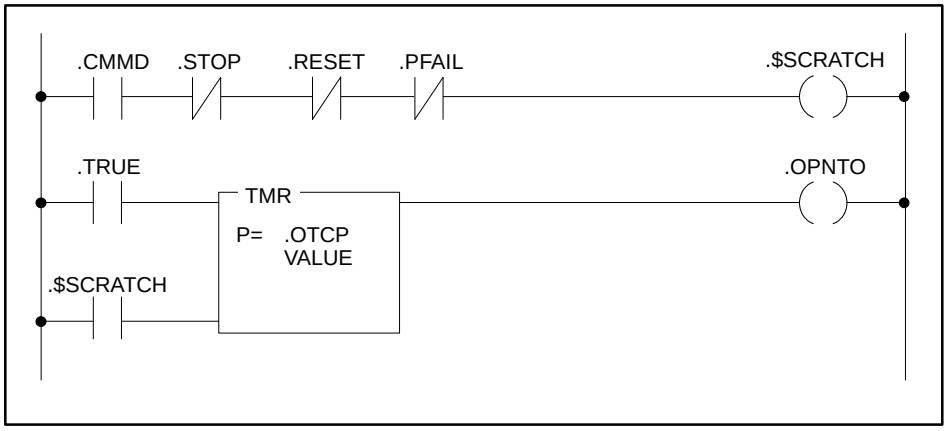


Figure A-103 VMD: OPNTO

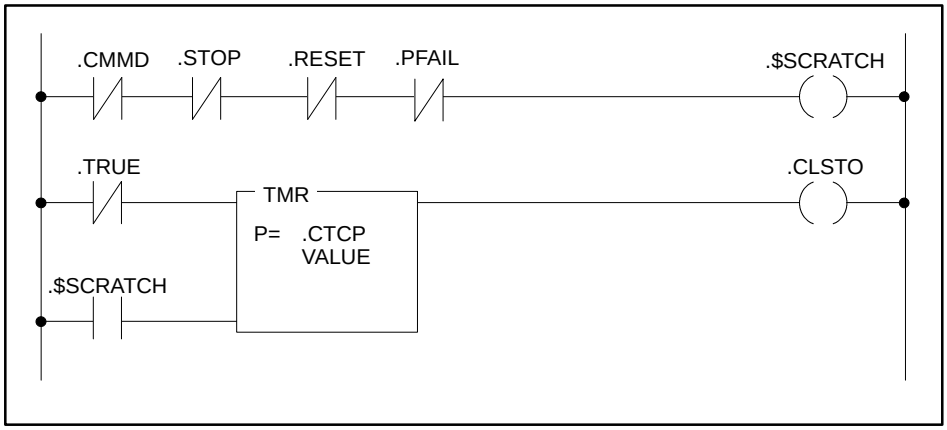


Figure A-104 VMD: CLSTO

Figure A-105 BV1: OVRDL

Figure A-106 BV1: OVRDH

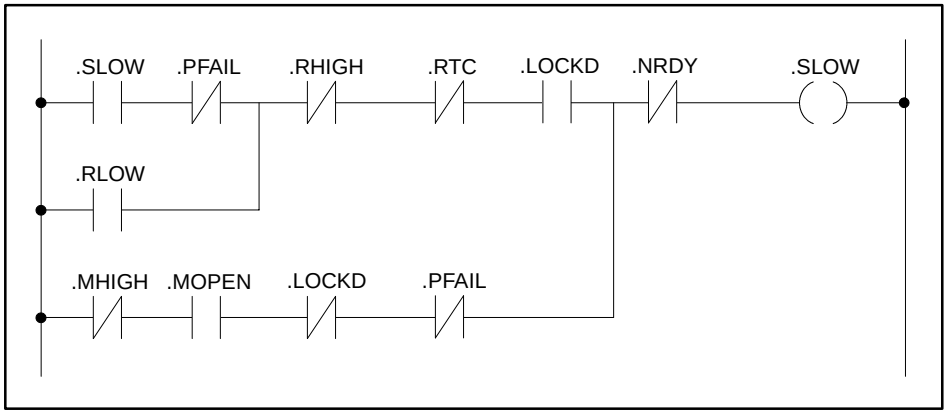


Figure A-107 BV1: SLOW

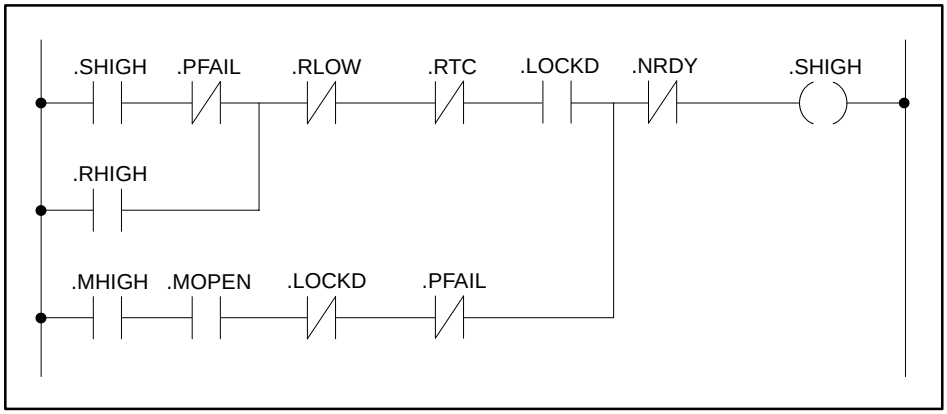


Figure A-108 BV1: SHIGH

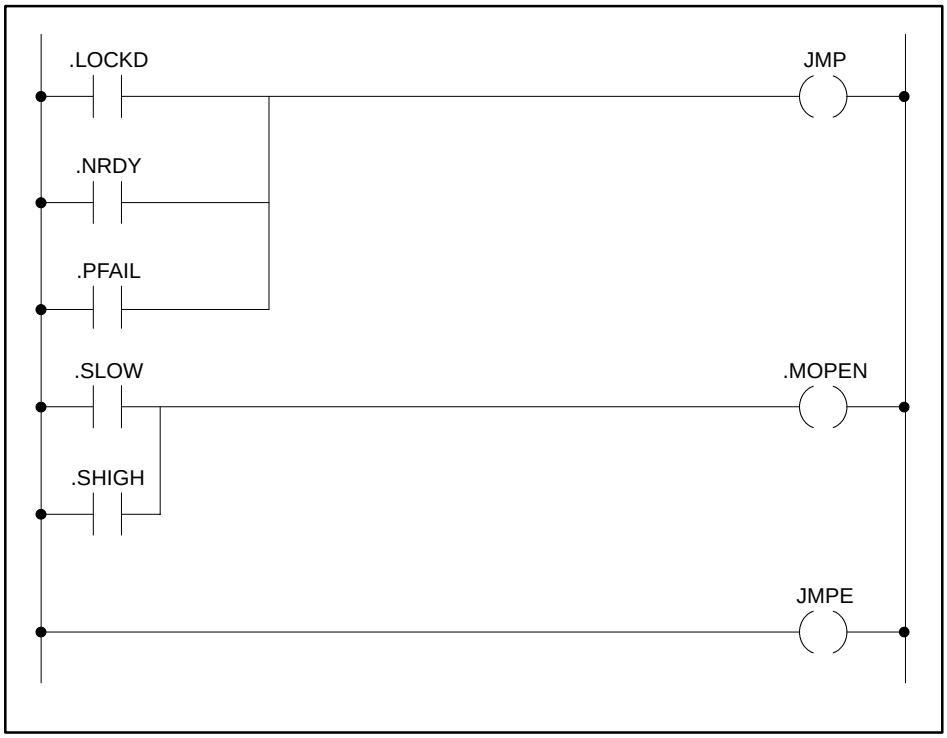


Figure A-109 BV1: MOPEN

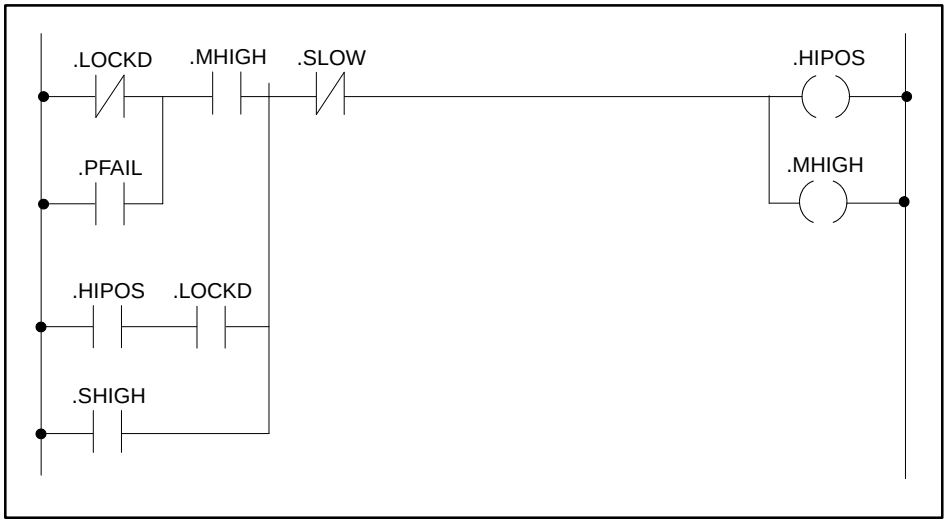


Figure A-110 BV1: MHIGH

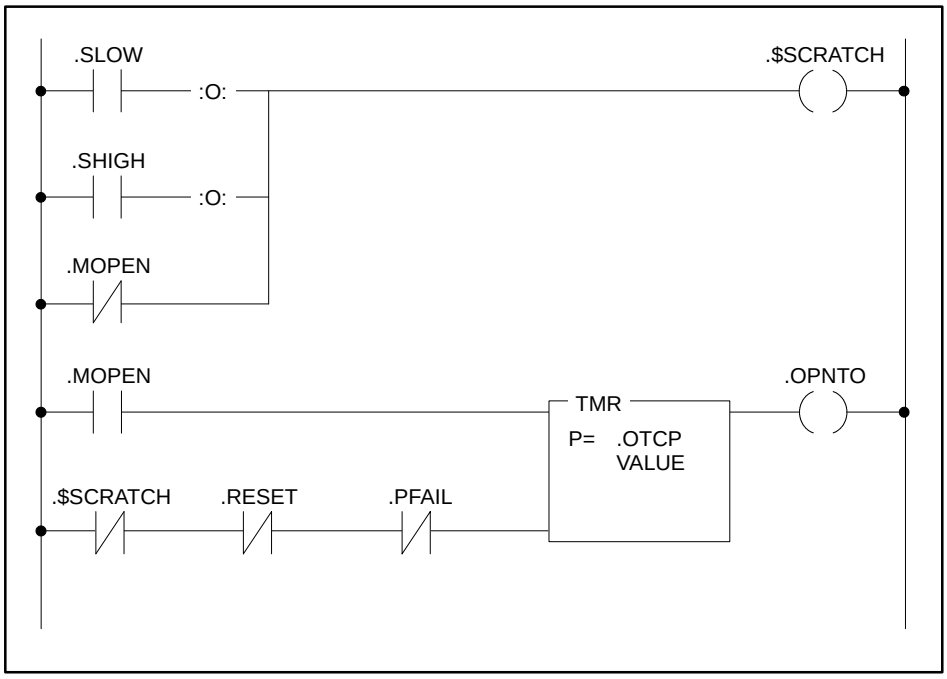


Figure A-111 BV1: OPNTO

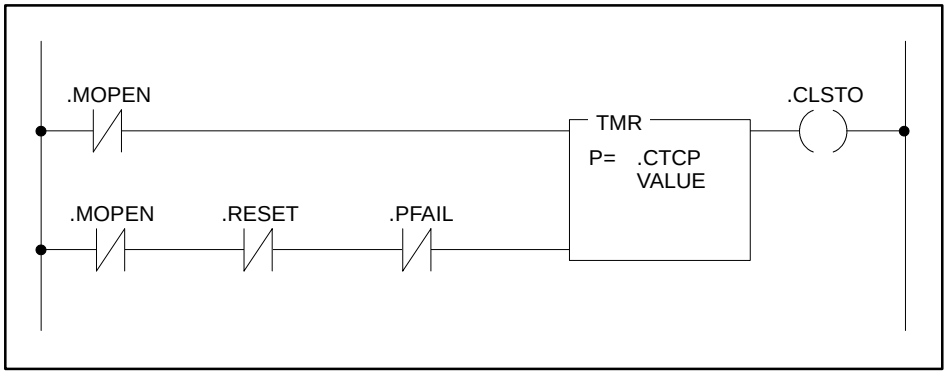


Figure A-112 BV1: CLSTO

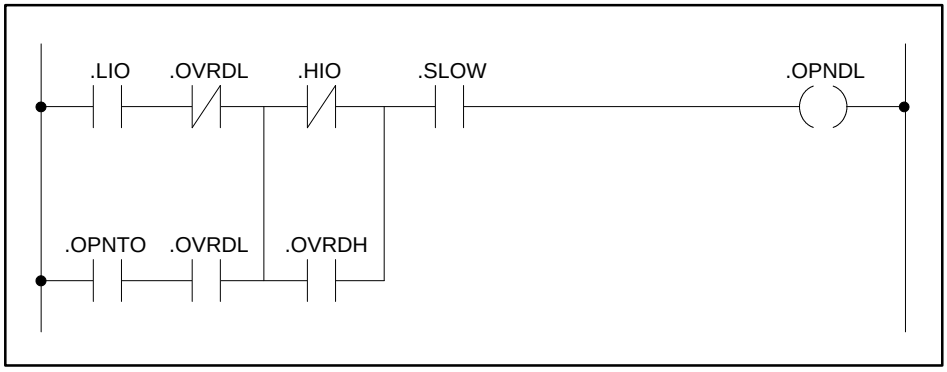


Figure A-113 BV1: OPNDL

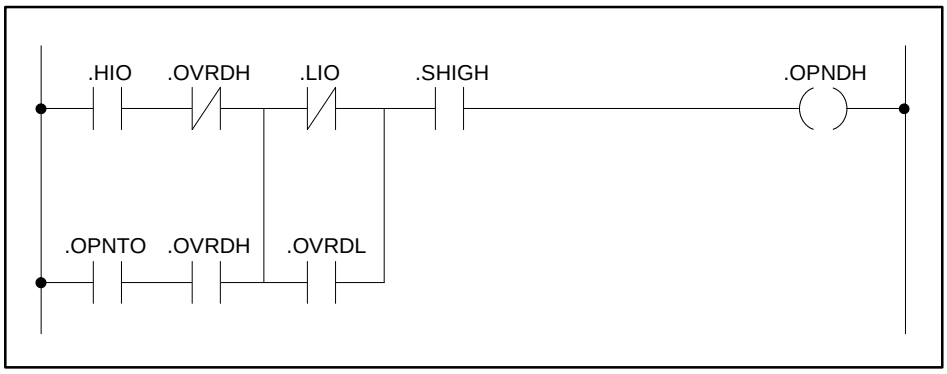


Figure A-114 BV1: OPNDH

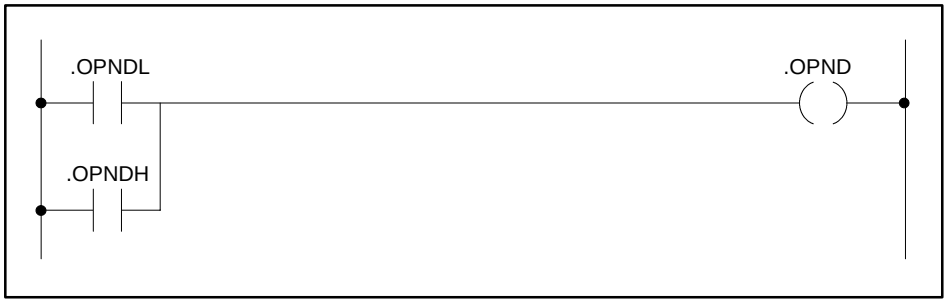


Figure A-115 BV1: OPND

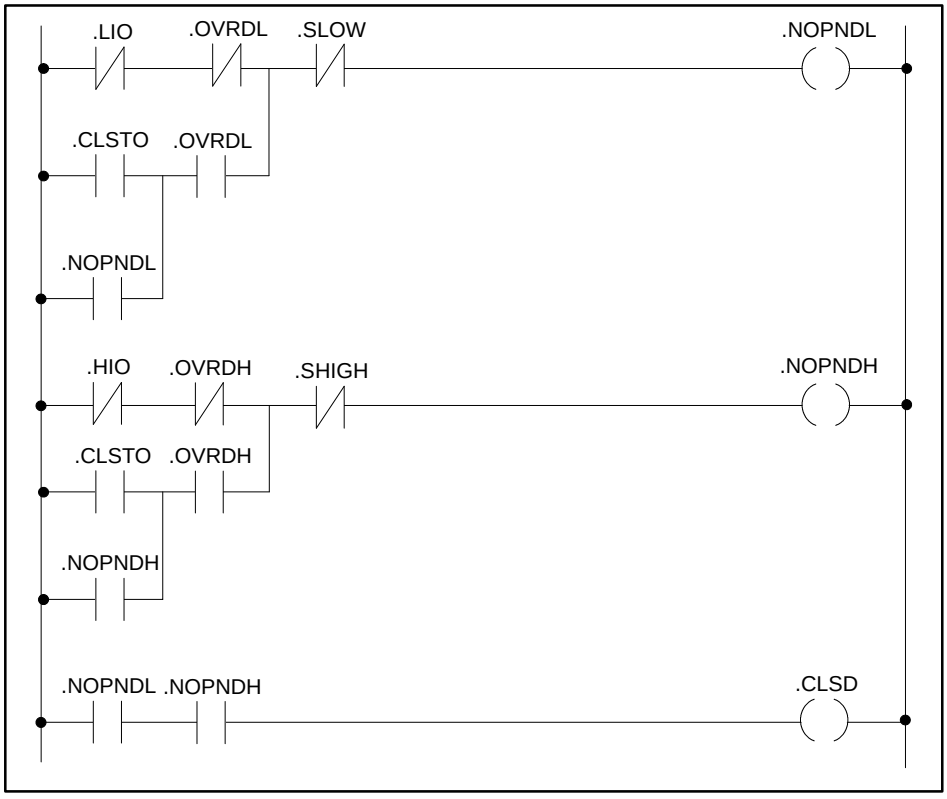


Figure A-116 BV1: CLSD

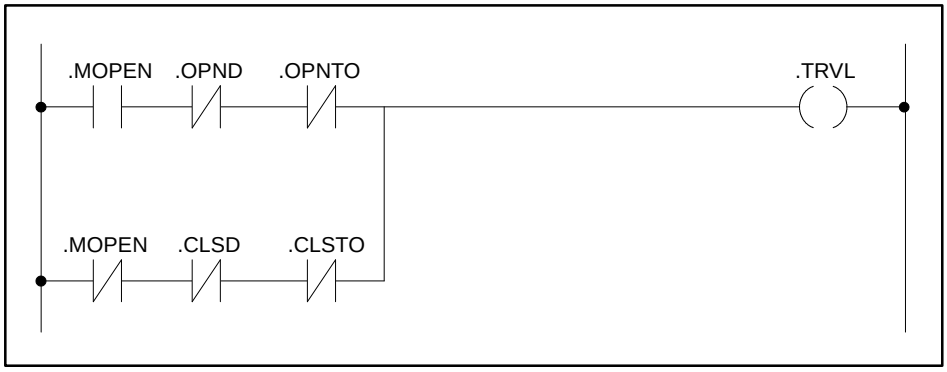


Figure A-117 BV1: TRVL

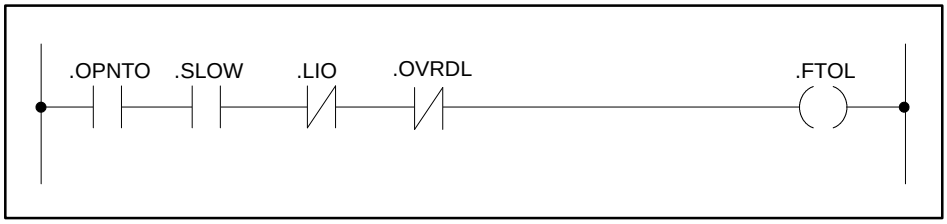


Figure A-118 BV1: FTOL

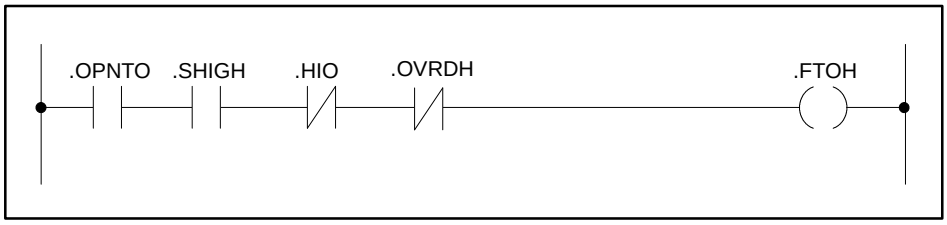


Figure A-119 BV1: FTOH

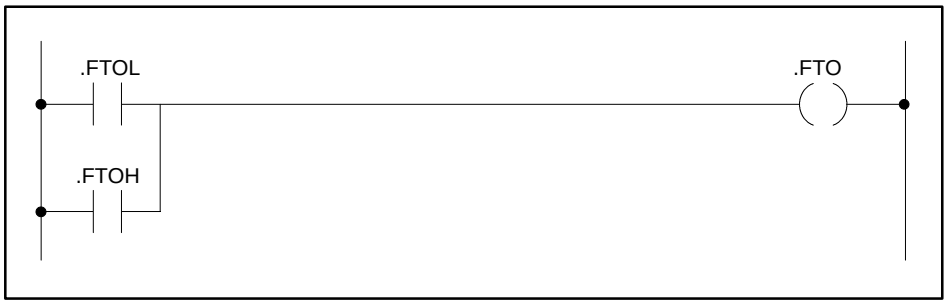


Figure A-120 BV1: FTO

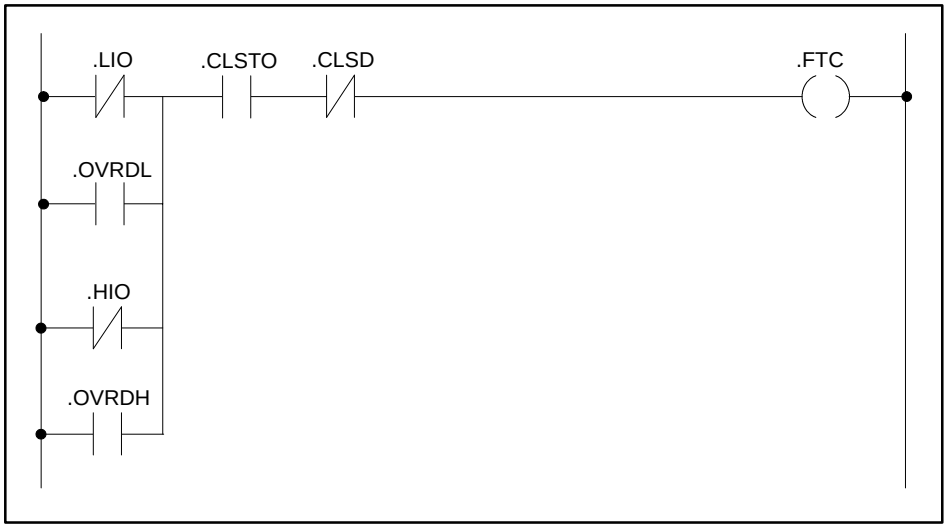


Figure A-121 BV1: FTC

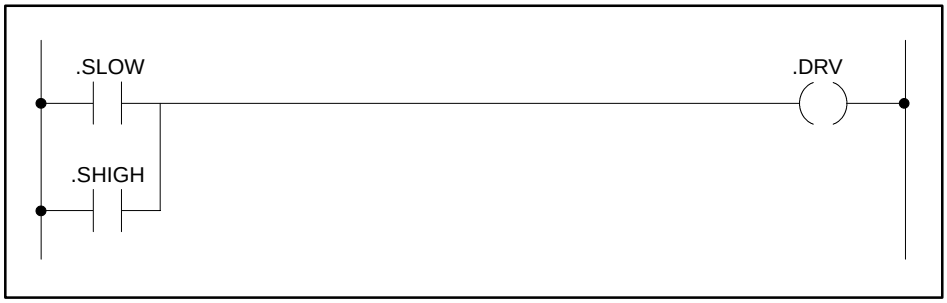


Figure A-122 BV2: DRV

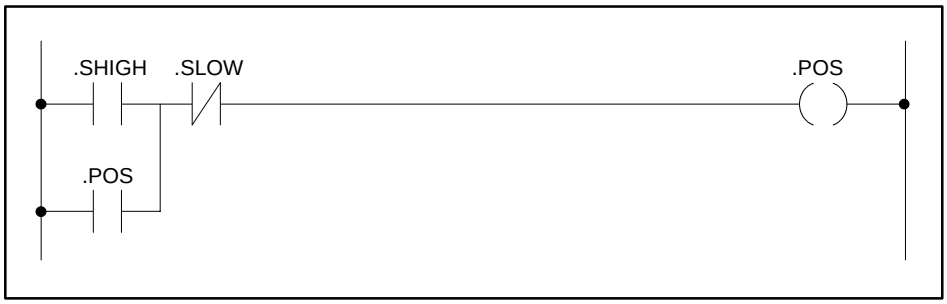


Figure A-123 BV2: POS

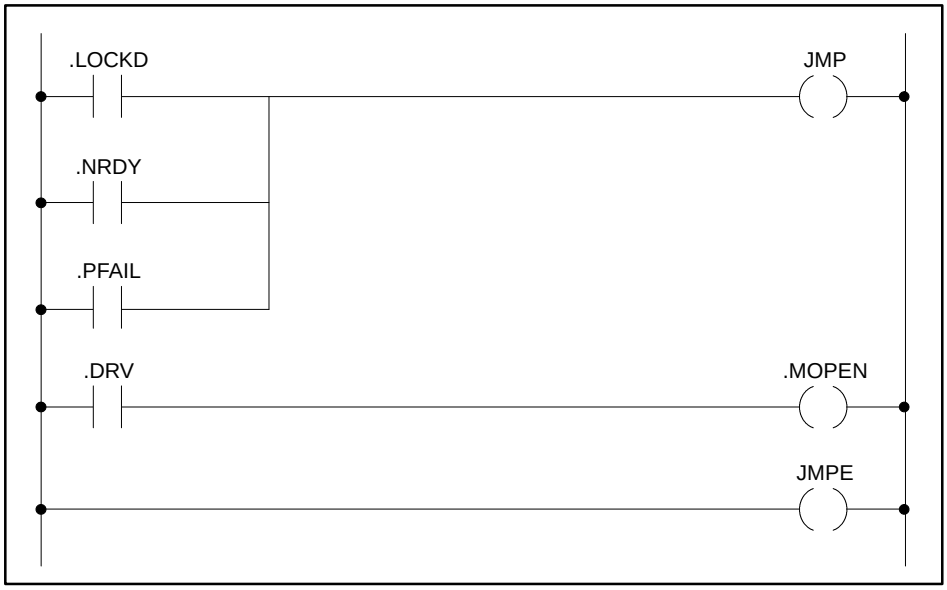


Figure A-124 BV2: MOPEN

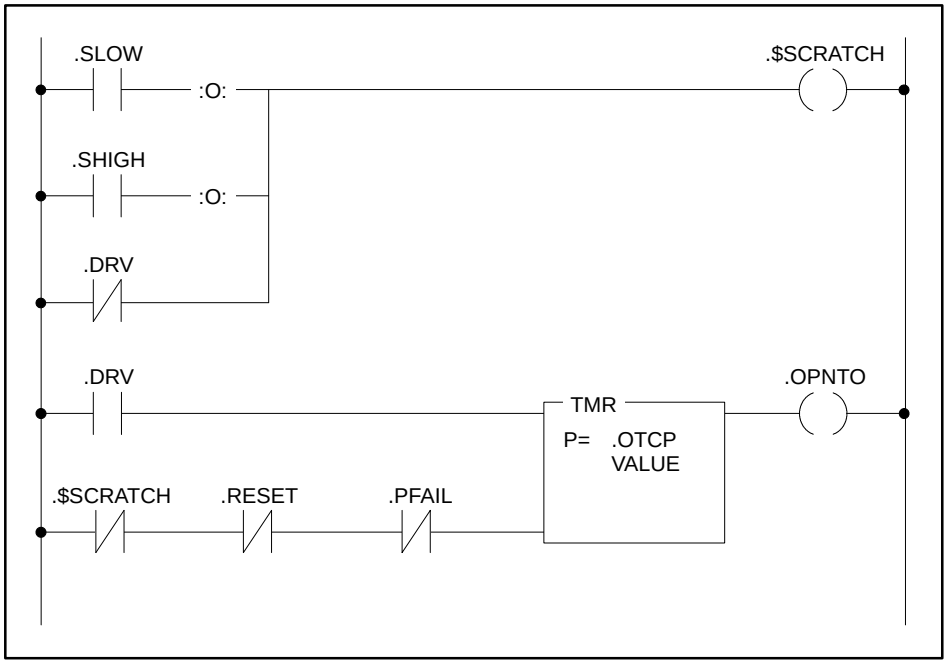


Figure A-125 BV2: OPNTO

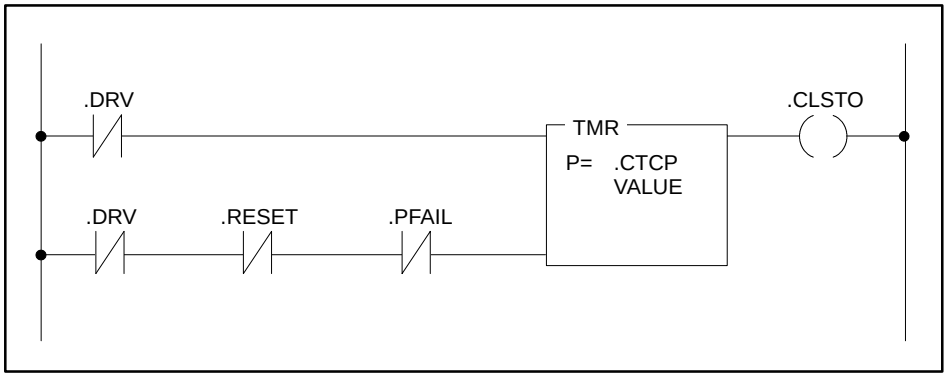


Figure A-126 BV2: CLSTO

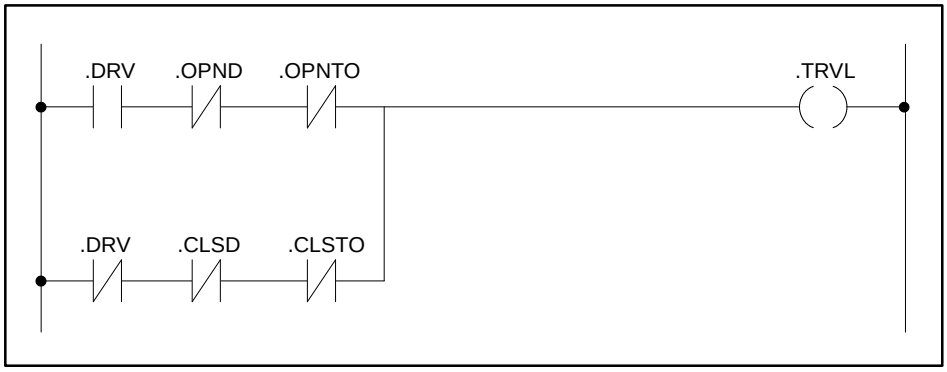


Figure A-127 BV2: TRVL

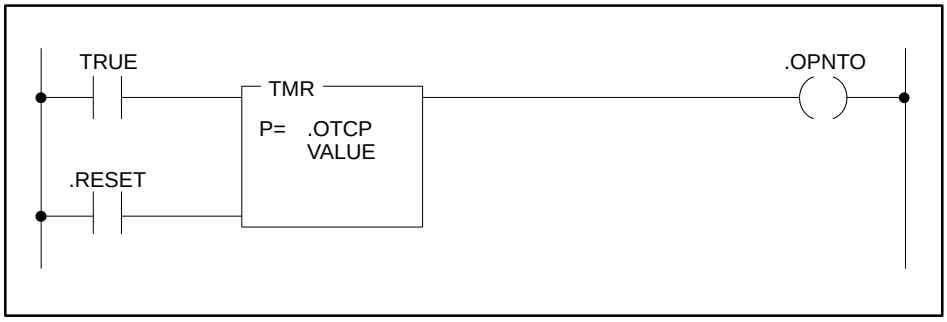


Figure A-128 VUD: OPNTO

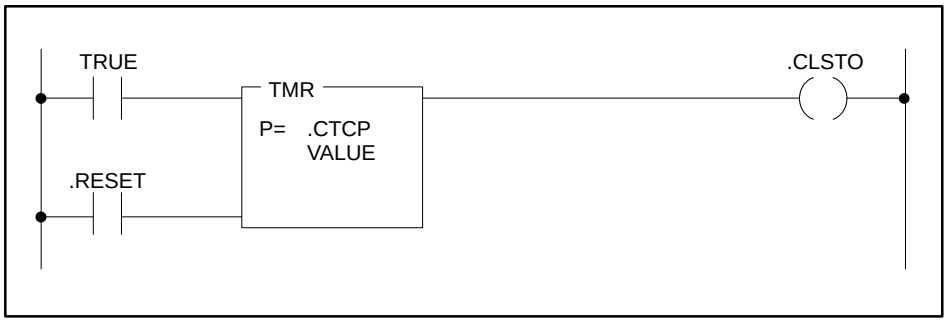


Figure A-129 VUD: CLSTO

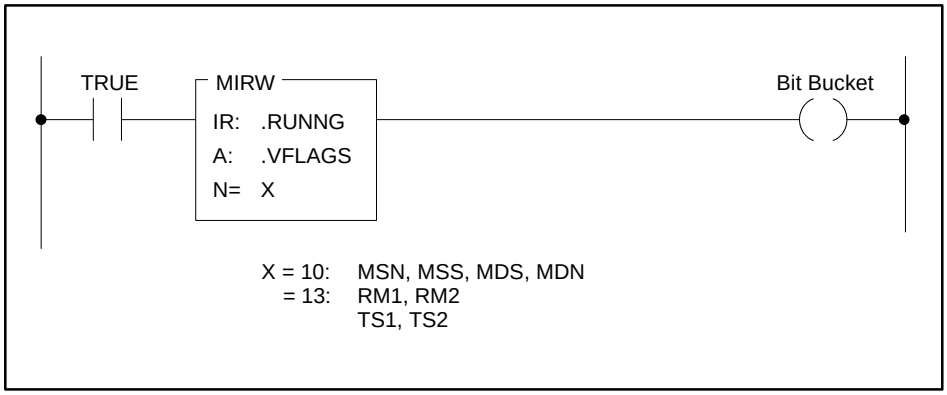


Figure A-130 MOTORS: Move Image To V

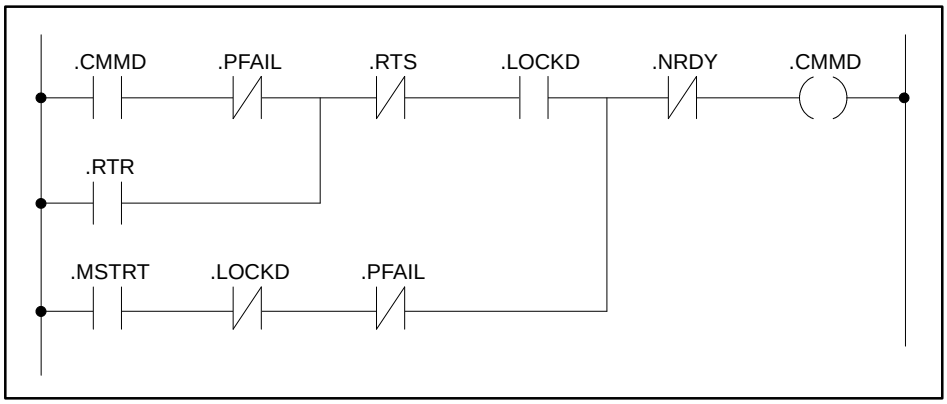


Figure A-131 MSN: CMMD

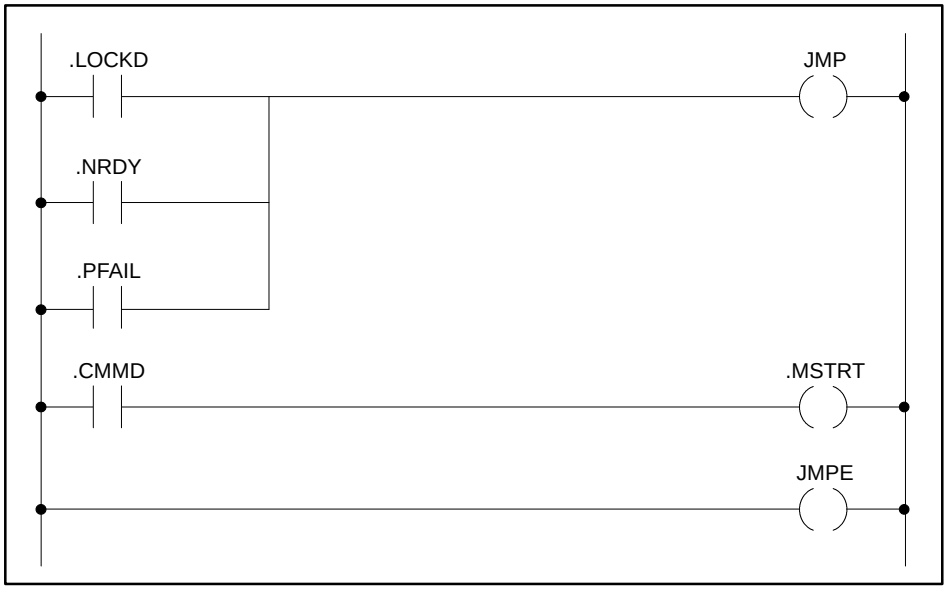


Figure A-132 MSN: MSTRT

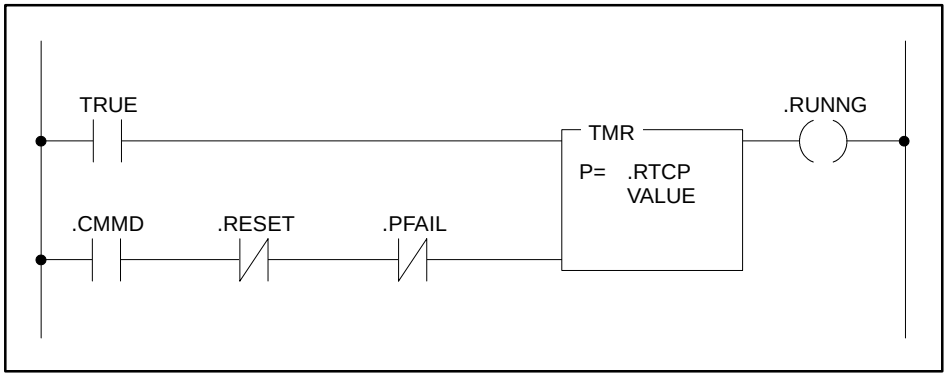


Figure A-133 MSN: RUNNG

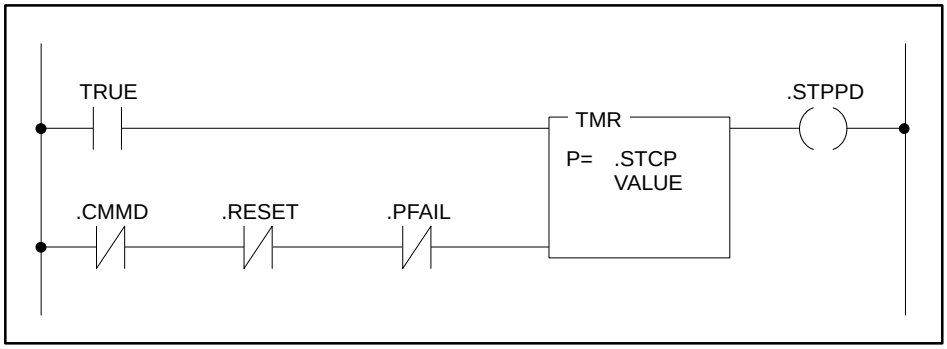


Figure A-134 MSN: STPPD

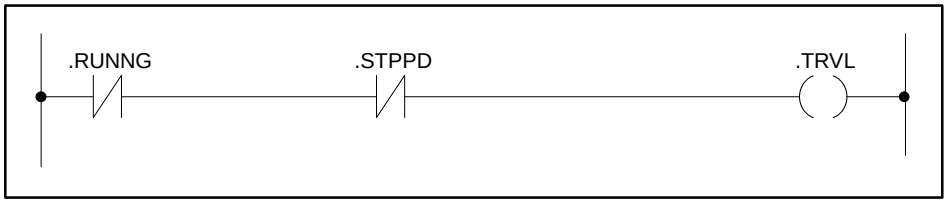


Figure A-135 MSN: TRVL

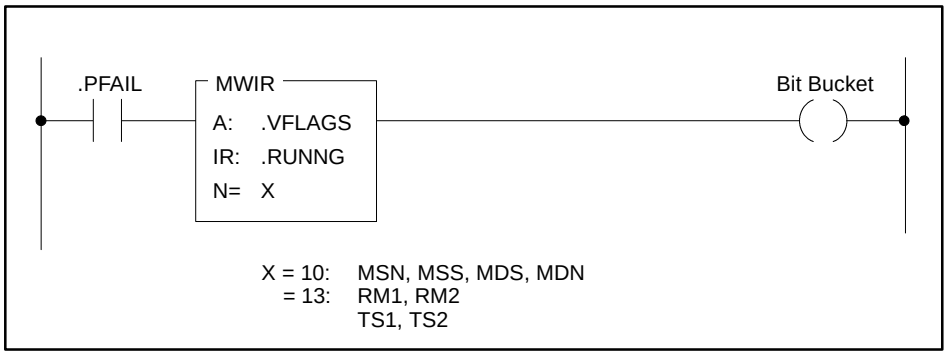


Figure A-136 MOTORS: Move Image from V

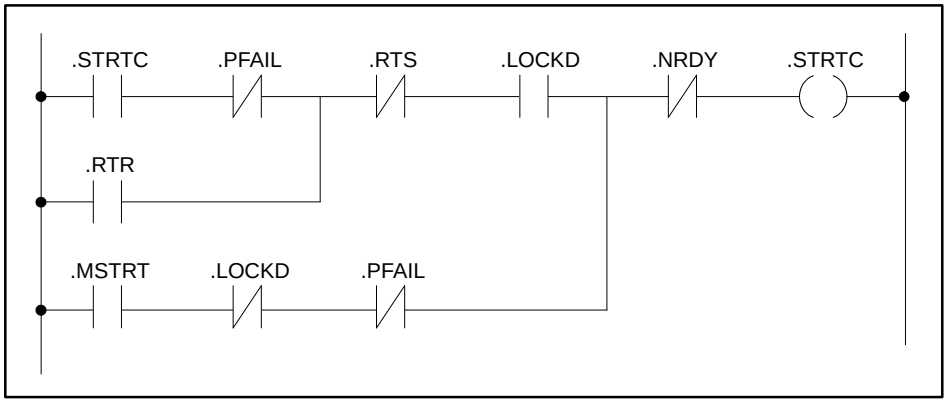


Figure A-137 MSS: STRTC

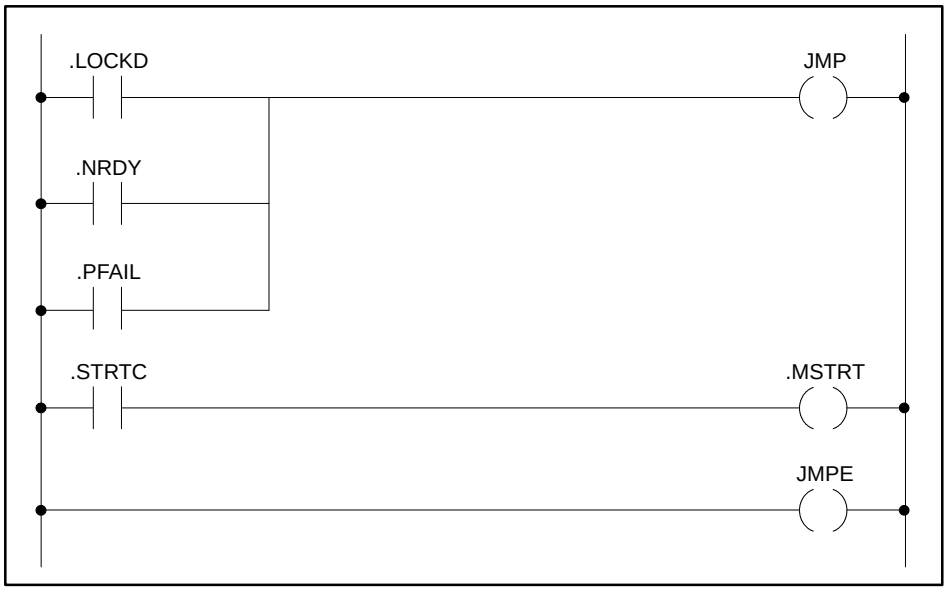


Figure A-138 MSS: MSTRT

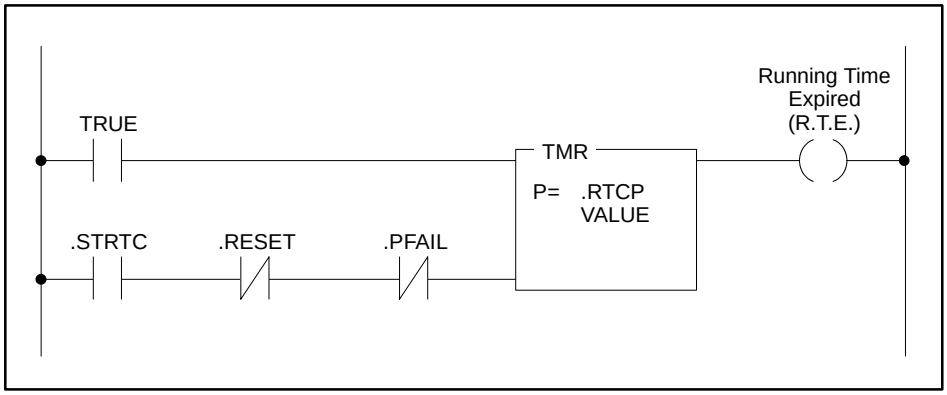


Figure A-139 MSS: Running Time Expired (R.T.E.)

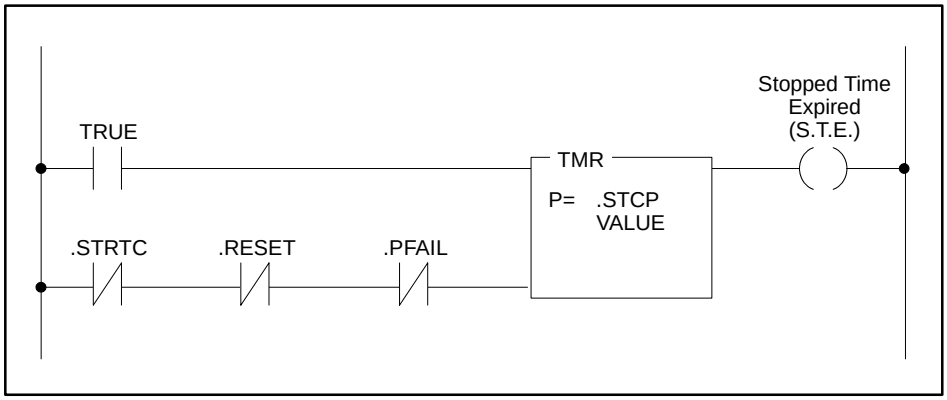


Figure A-140 MSS: Stopped Time Expired (S.T.E)

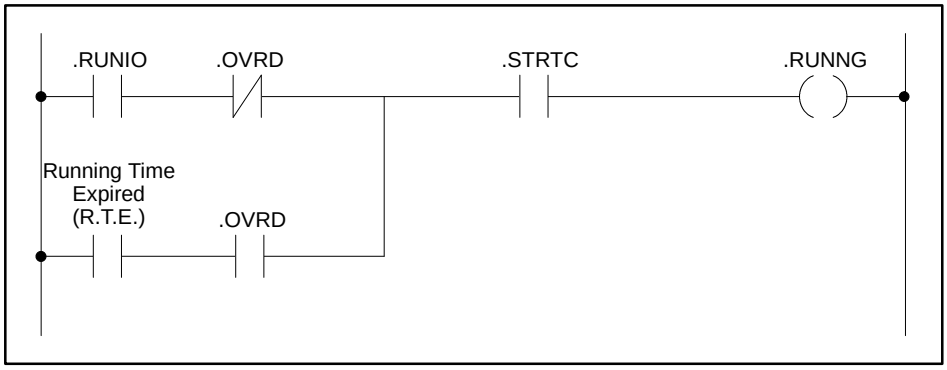


Figure A-141 MSS: RUNNG

Figure A-142 MSS: RUNNG (IGNORE FDBK OVRD)

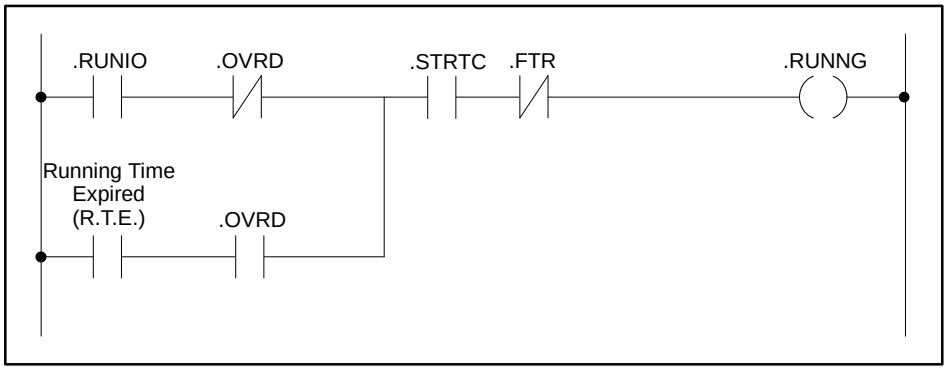


Figure A-143 MSS: RUNNG (LATCH FTR)

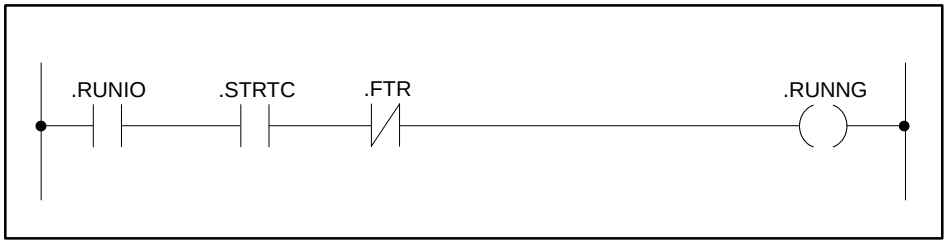


Figure A-144 MSS: RUNNG (IGNORE FDBK OVRD) (LATCH FTR)

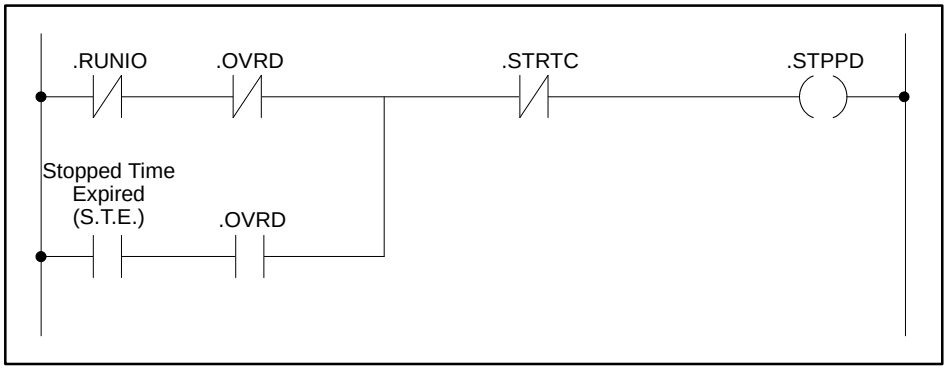


Figure A-145 MSS: STPPD

Figure A-146 MSS: STPPD (IGNORE FDBK OVRD)

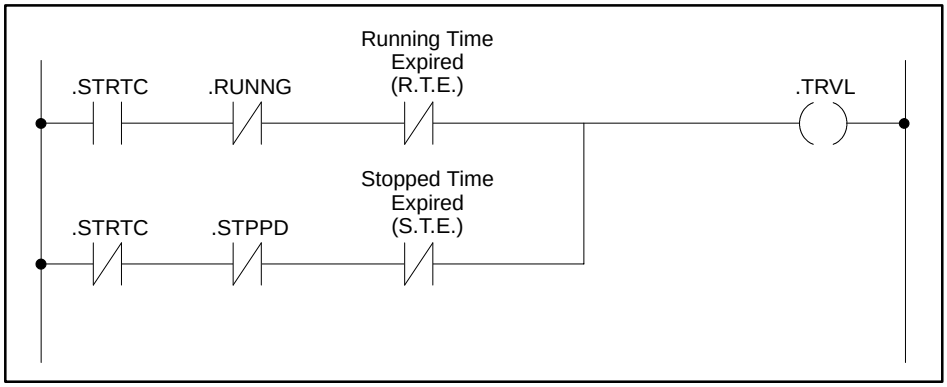


Figure A-147 MSS: TRVL

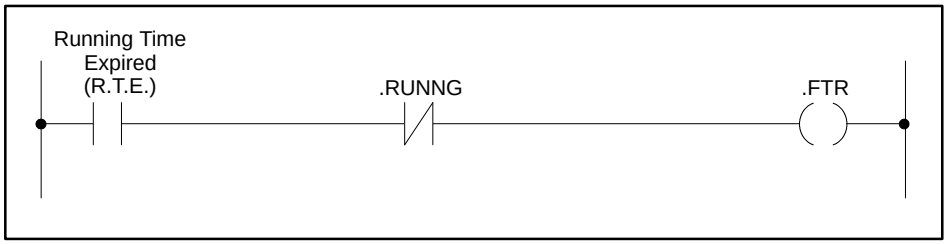


Figure A-148 MSS: FTR

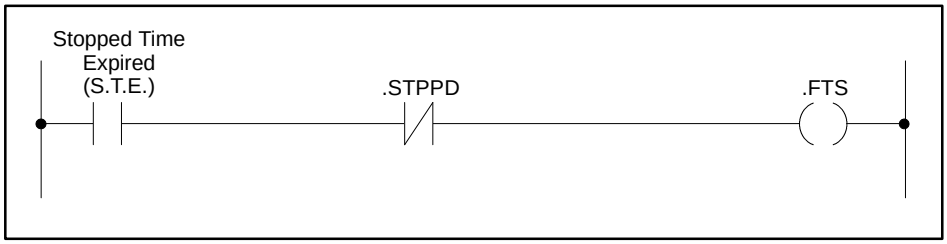


Figure A-149 MSS: FTS

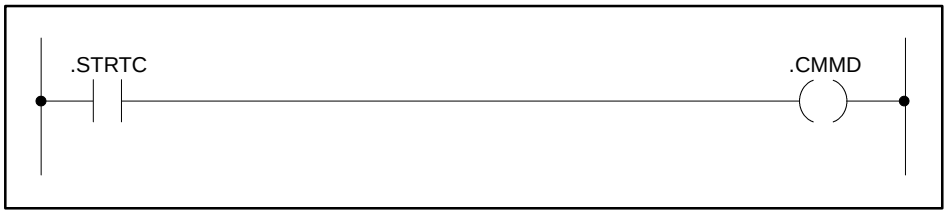


Figure A-150 MSS: CMMD

Figure A-151 MSS: CMMD (LATCH FTR)

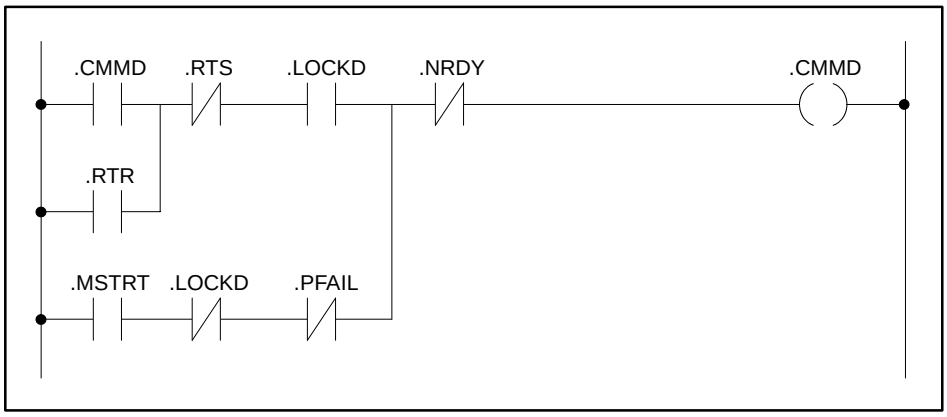


Figure A-152 MDN: CMMD

Figure A-153 MDN: STRTC

Figure A-154 MDN: STOPC

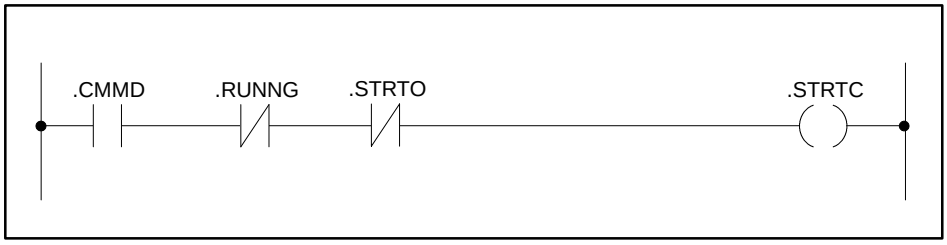


Figure A-155 MDS: STRTC

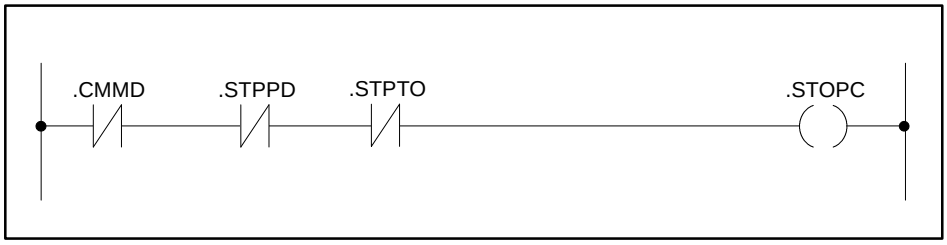


Figure A-156 MDS: STOPC

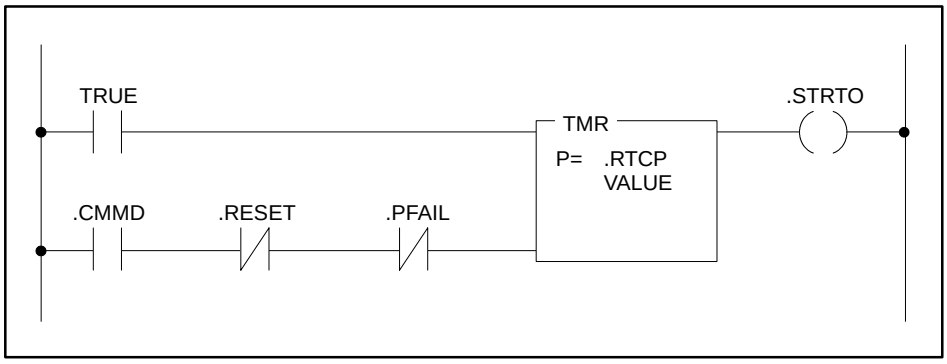


Figure A-157 MDS: STRTO

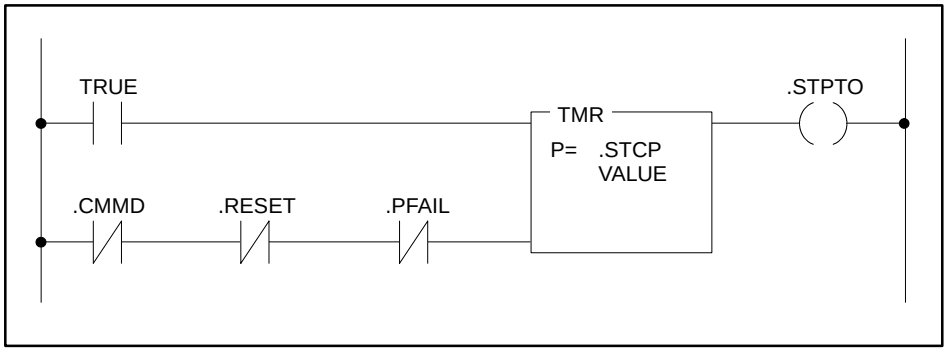


Figure A-158 MDS: STPTO

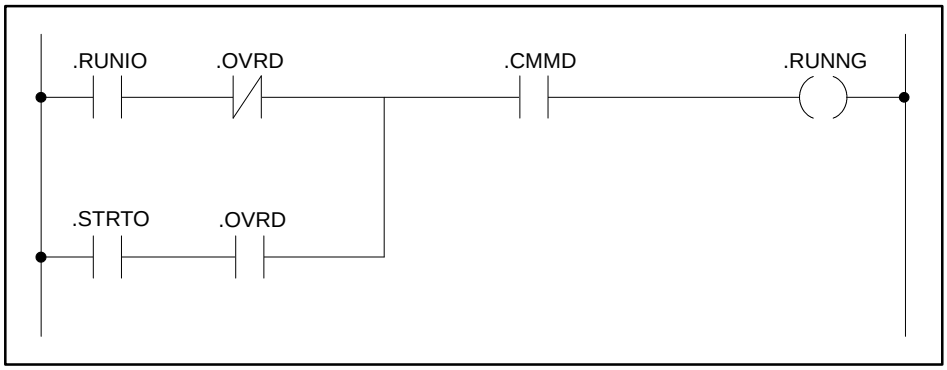


Figure A-159 MDS: RUNNG

Figure A-160 MDS: RUNNG (IGNORE FDBK OVRD)

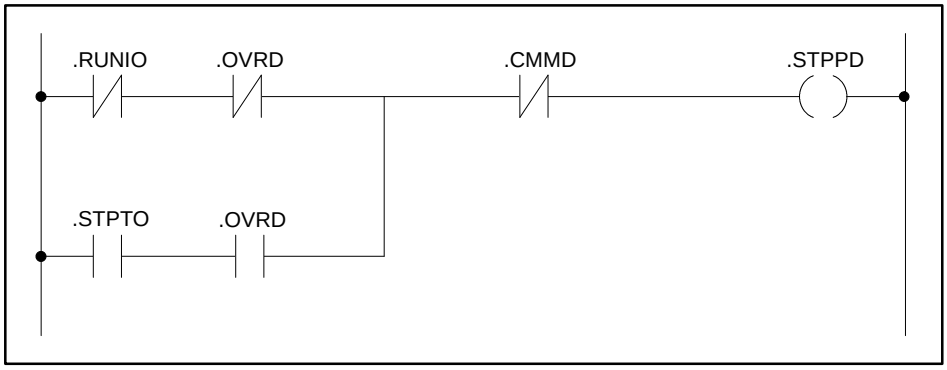


Figure A-161 MDS: STPPD

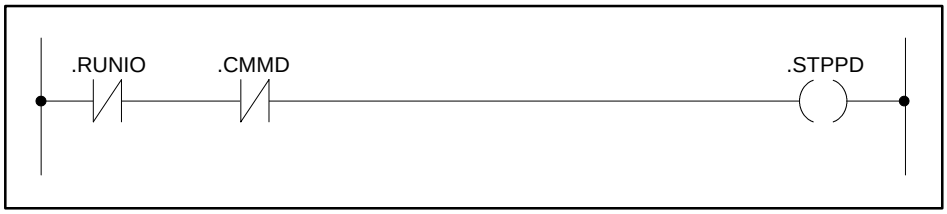


Figure A-162 MDS: STPPD (IGNORE FDBK OVRD)

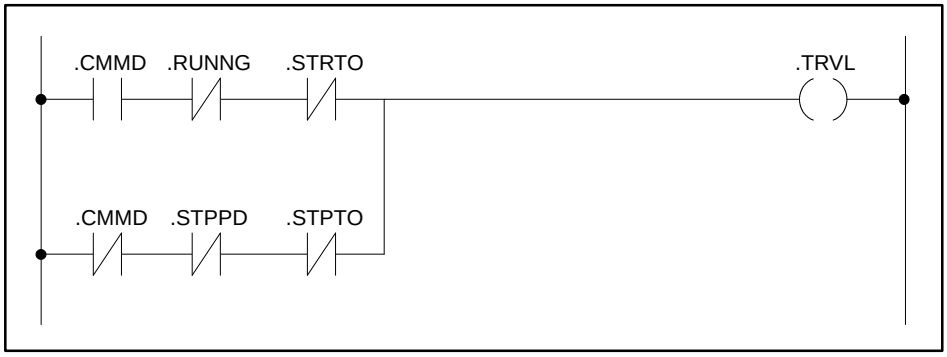


Figure A-163 MDS: TRVL

Figure A-164 MDS: FTR

Figure A-165 MDS: FTS

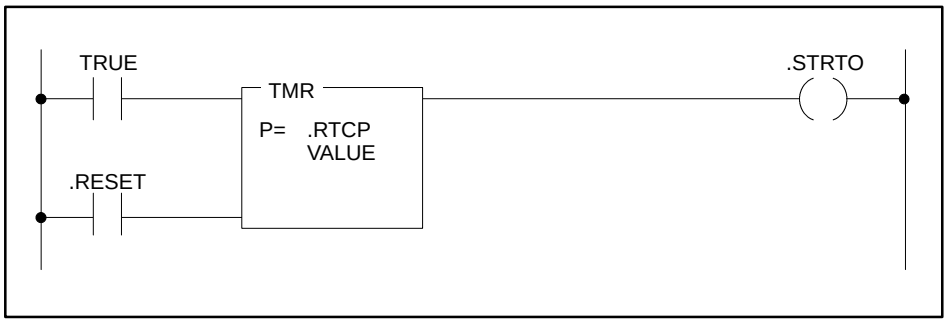


Figure A-166 MUD: STRTO

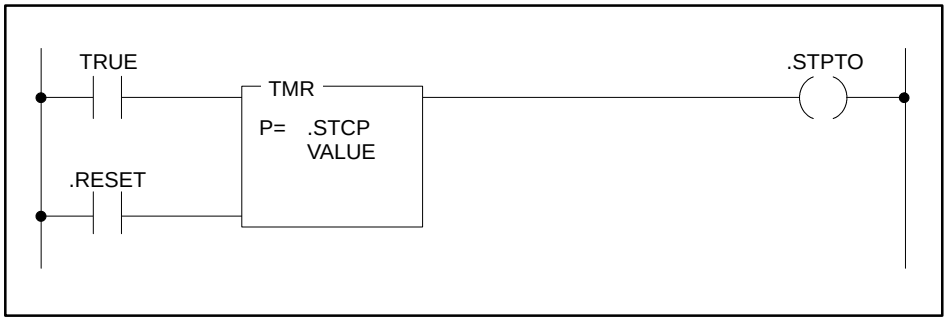


Figure A-167 MUD: STPTO

Figure A-168 RM1: OVRDF

Figure A-169 RM1: OVRDR

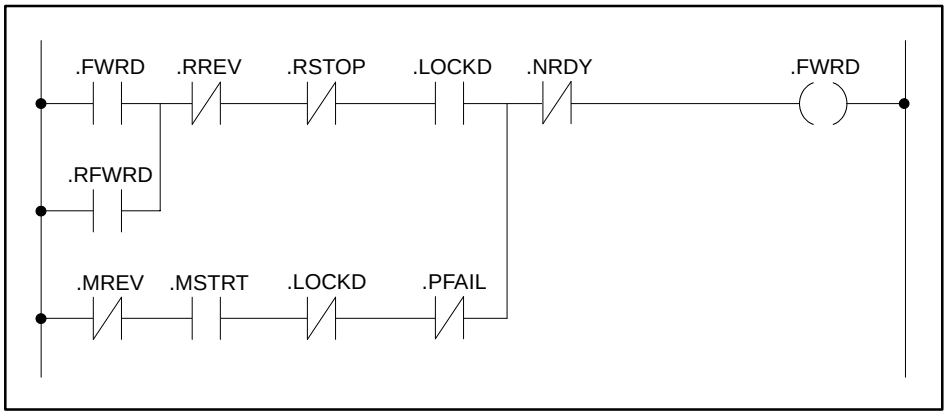


Figure A-170 RM1: FWRD

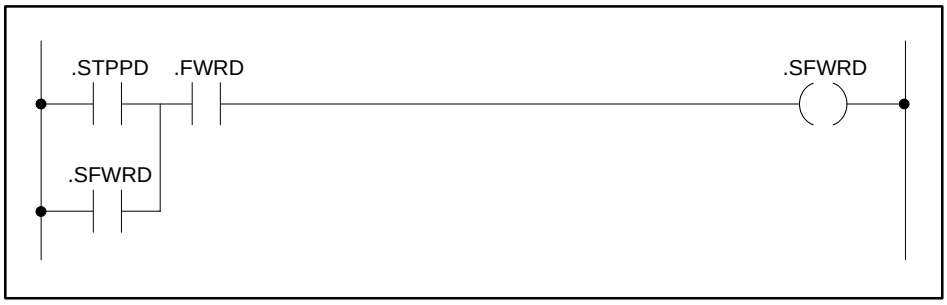


Figure A-171 RM1: SFWRD

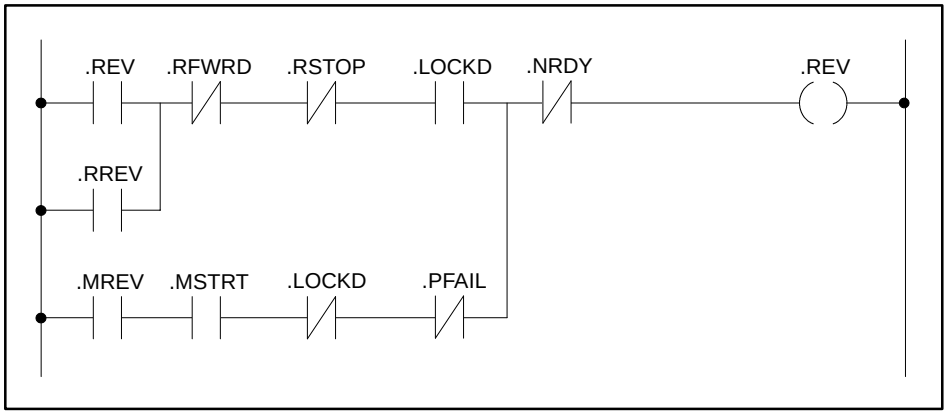


Figure A-172 RM1: REV

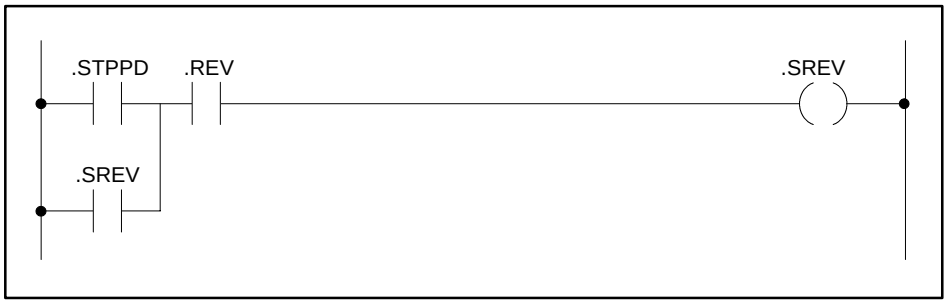


Figure A-173 RM1: SREV

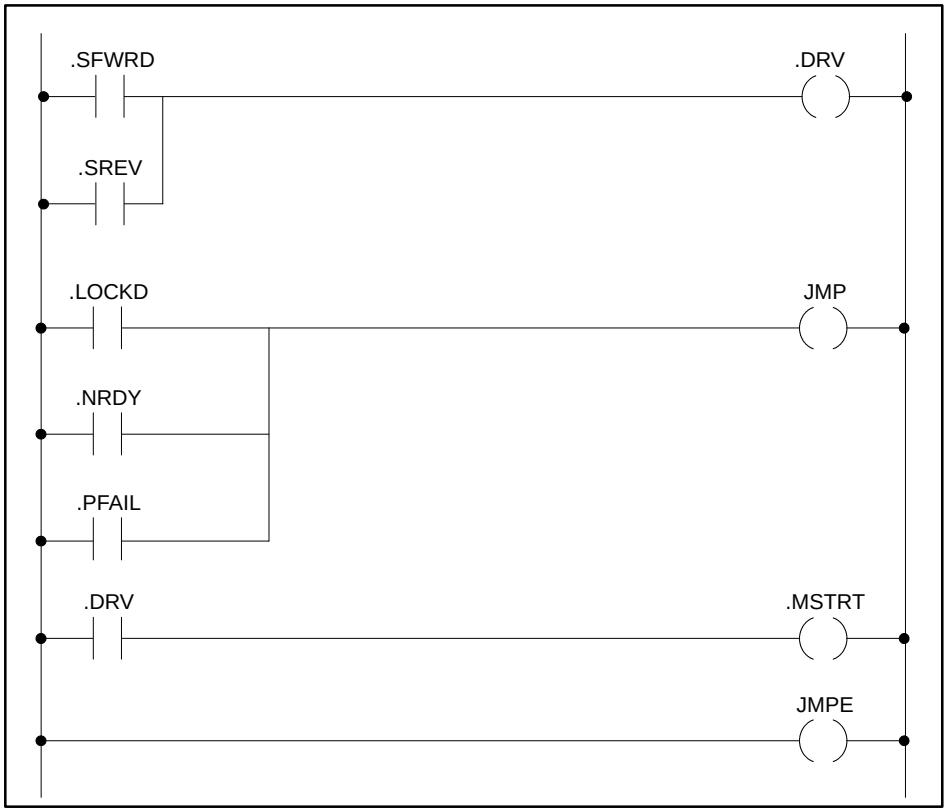


Figure A-174 RM1: MSTRT

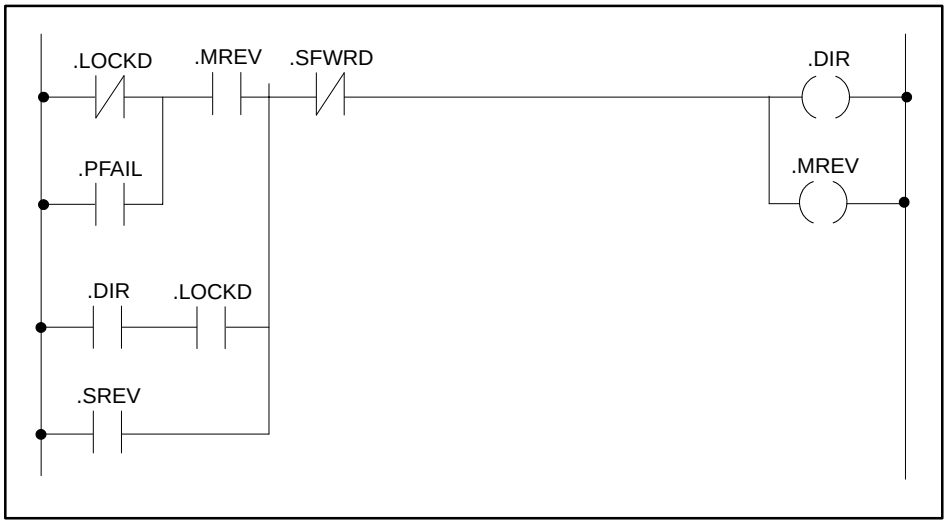


Figure A-175 RM1: MREV

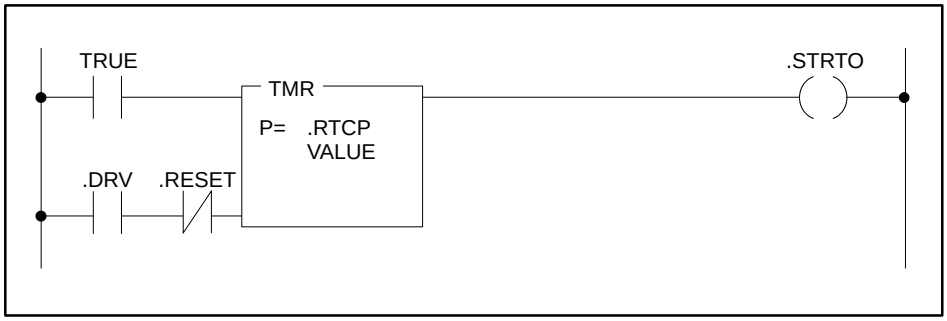


Figure A-176 RM1: STRTO

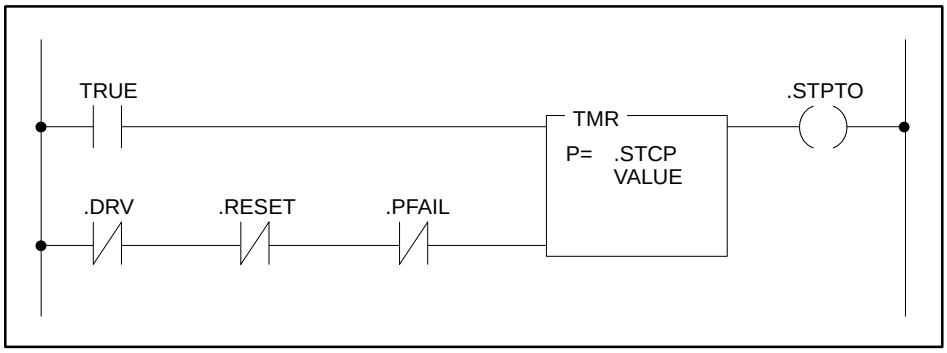


Figure A-177 RM1: STPTO

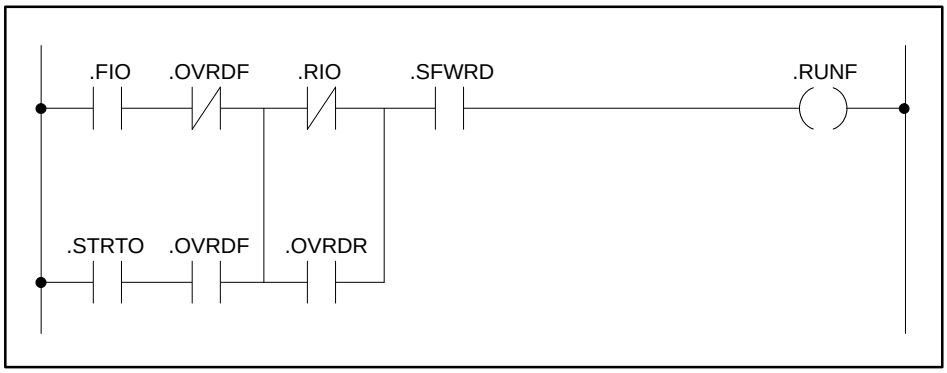


Figure A-178 RM1: RUNF

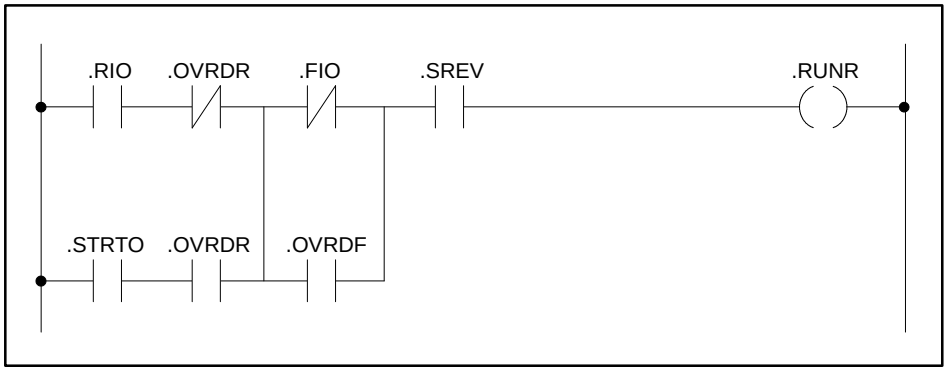


Figure A-179 RM1: RUNR

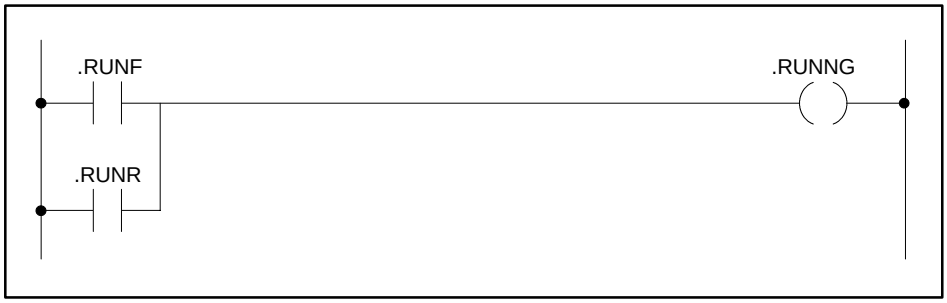


Figure A-180 RM1: RUNNG

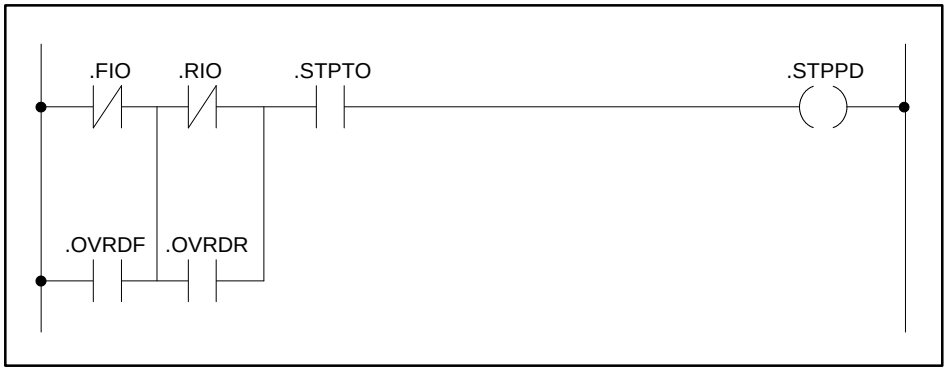


Figure A-181 RM1: STPPD

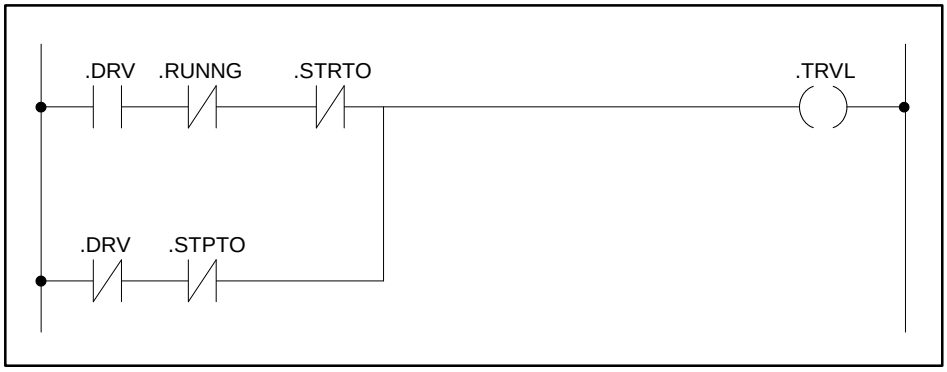


Figure A-182 RM1: TRVL

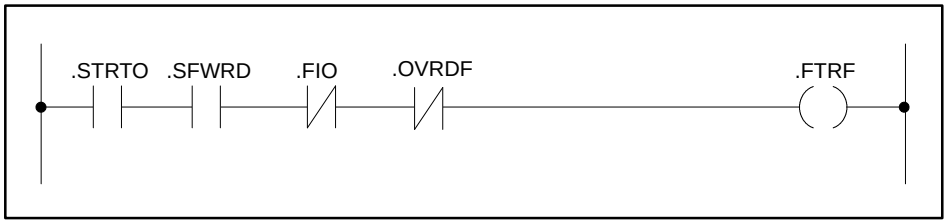


Figure A-183 RM1: FTRF

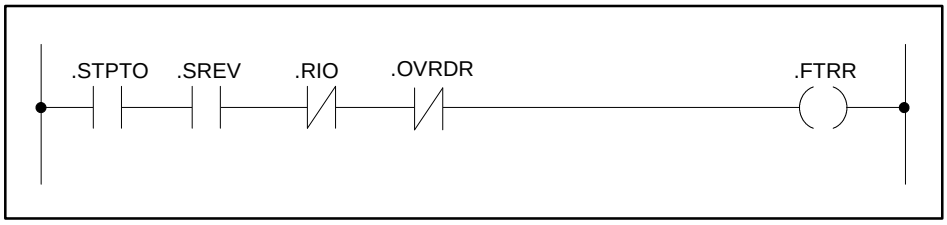


Figure A-184 RM1: FTRR

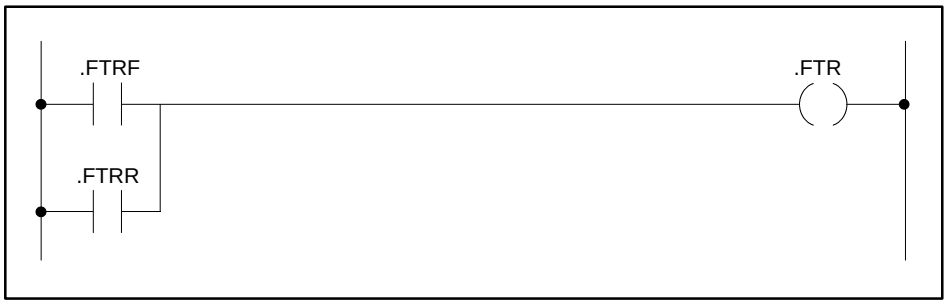


Figure A-185 RM1: FTR

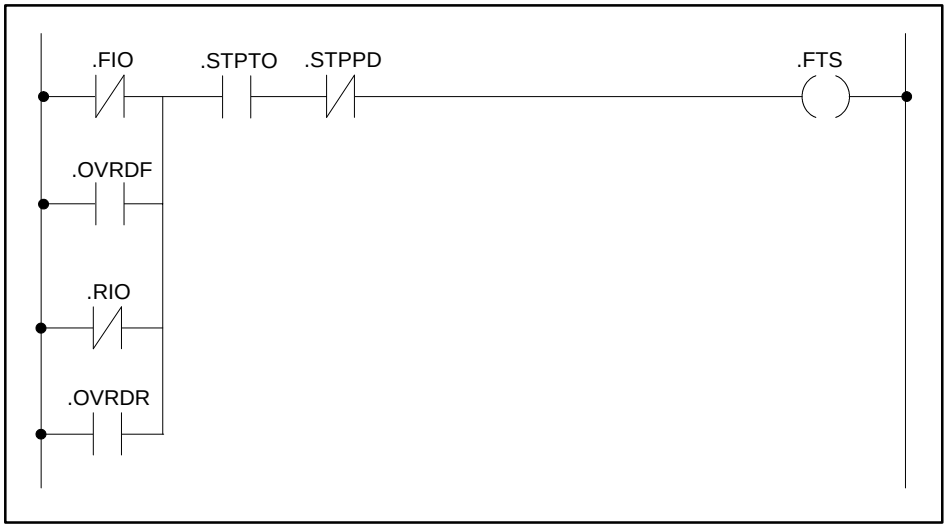


Figure A-186 RM1: FTS

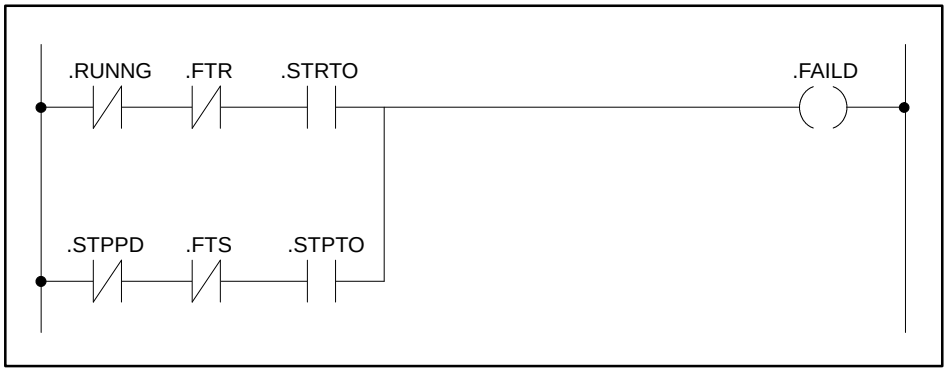


Figure A-187 RM1: FAILD

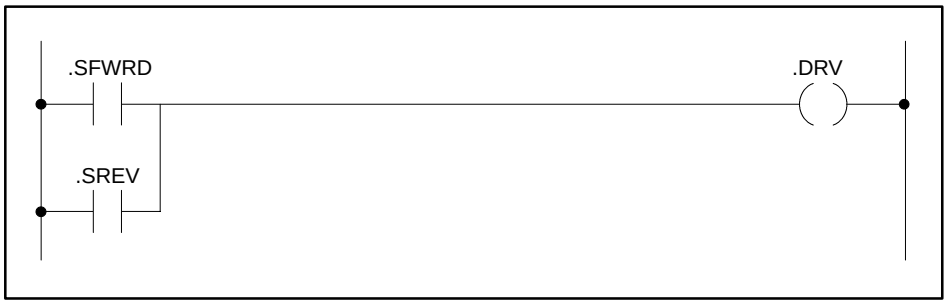


Figure A-188 RM2: DRV

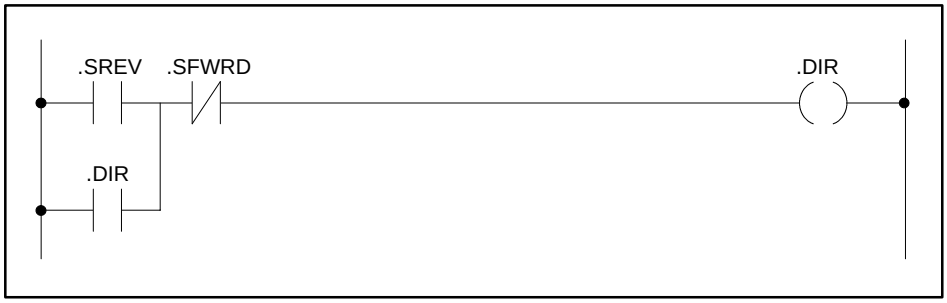


Figure A-189 RM2: DIR

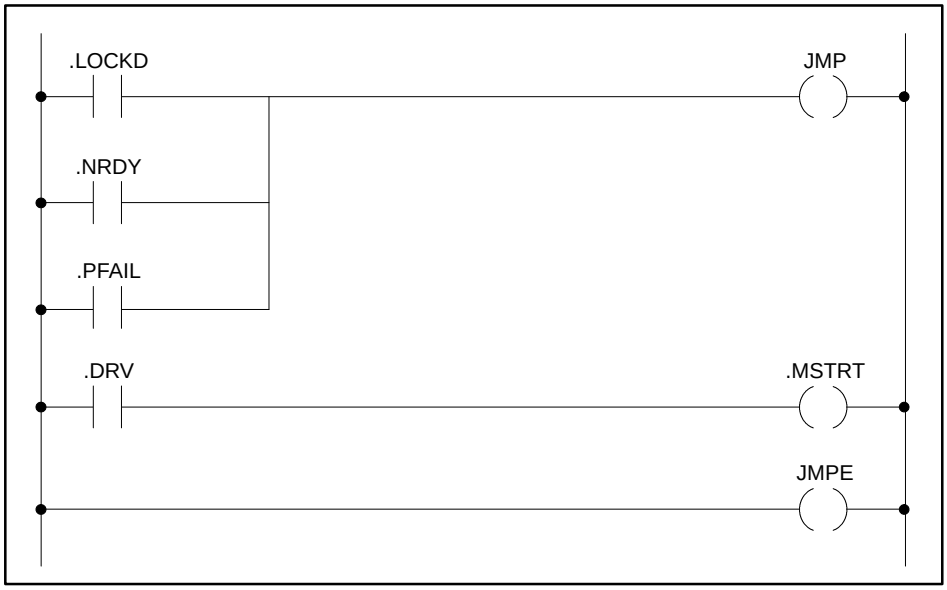


Figure A-190 RM2: MSTRT

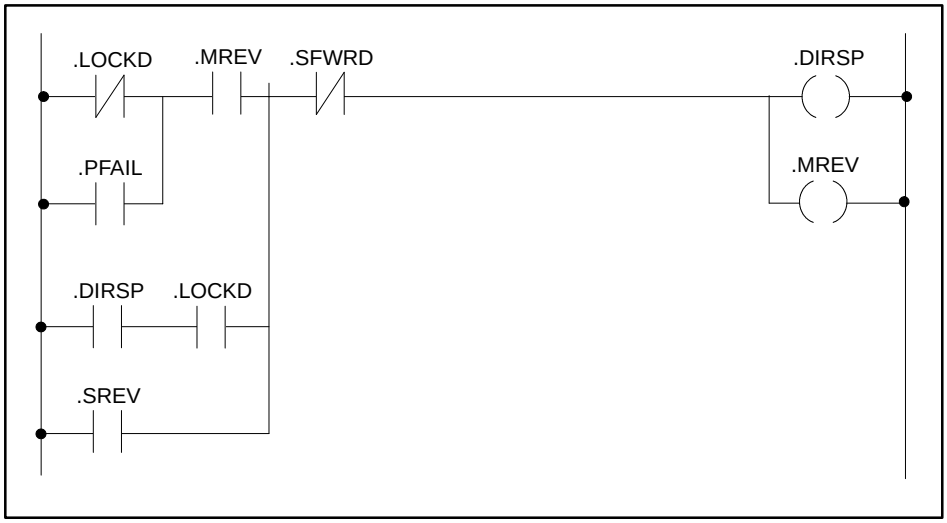


Figure A-191 RM2: MREV

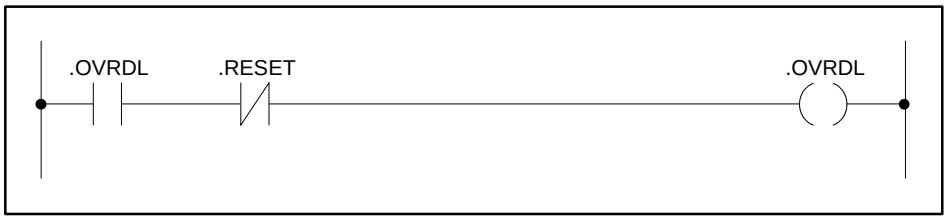


Figure A-192 TS1: OVRDL

Figure A-193 TS1: OVRDH

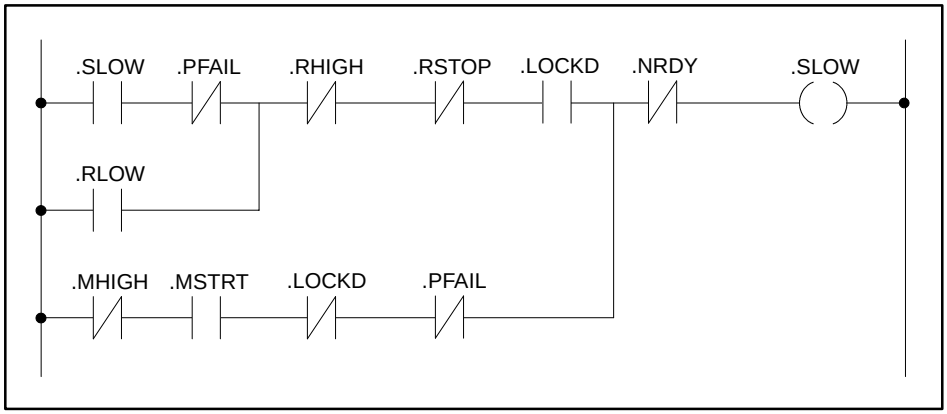


Figure A-194 TS1: SLOW

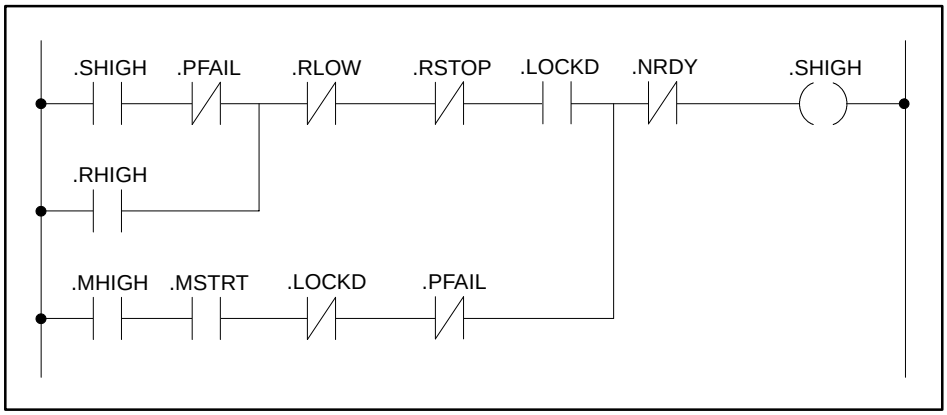


Figure A-195 TS1: SHIGH

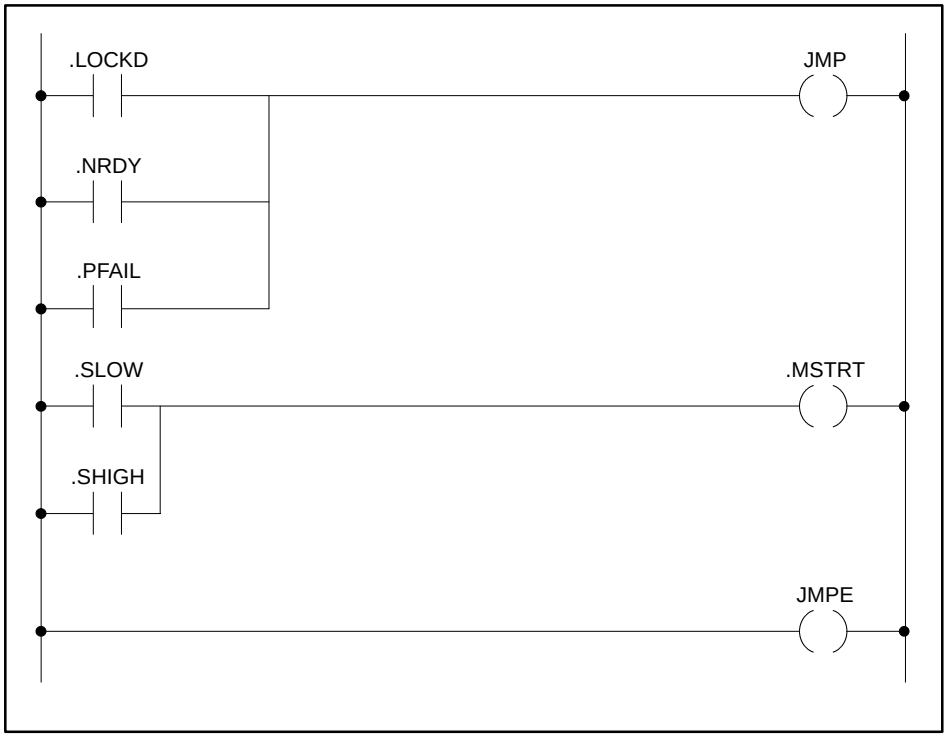


Figure A-196 TS1: MSTRT

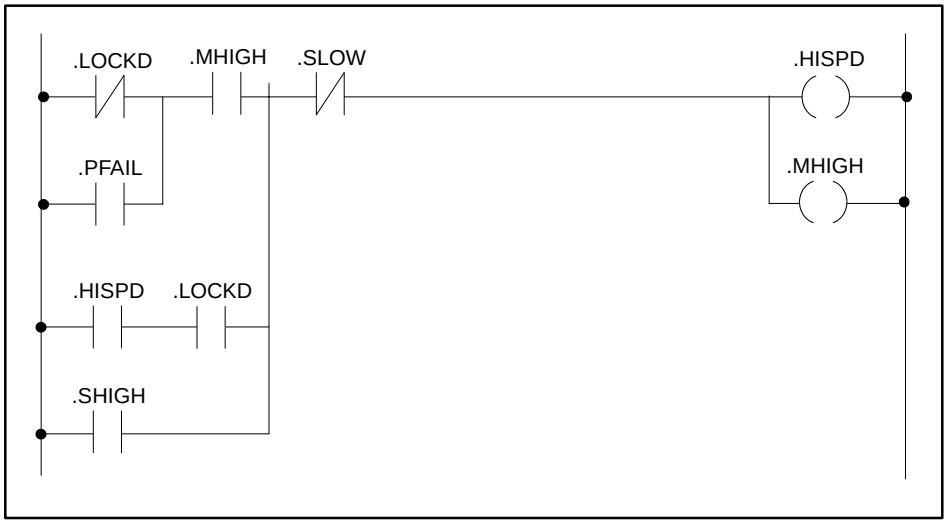


Figure A-197 TS1: MHIGH

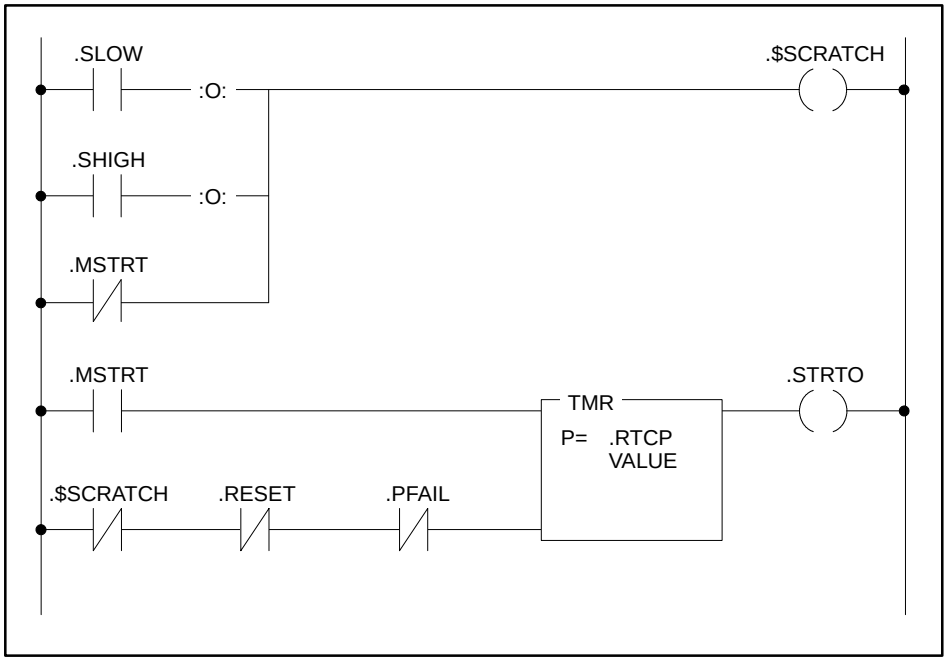


Figure A-198 TS1: STRTO

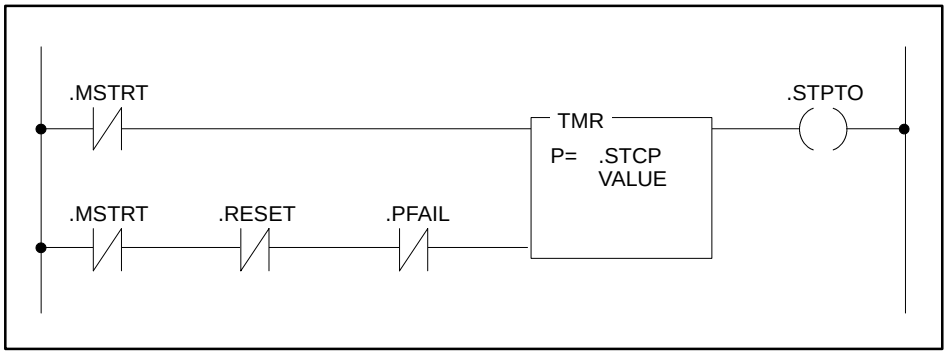


Figure A-199 TS1: STPTO

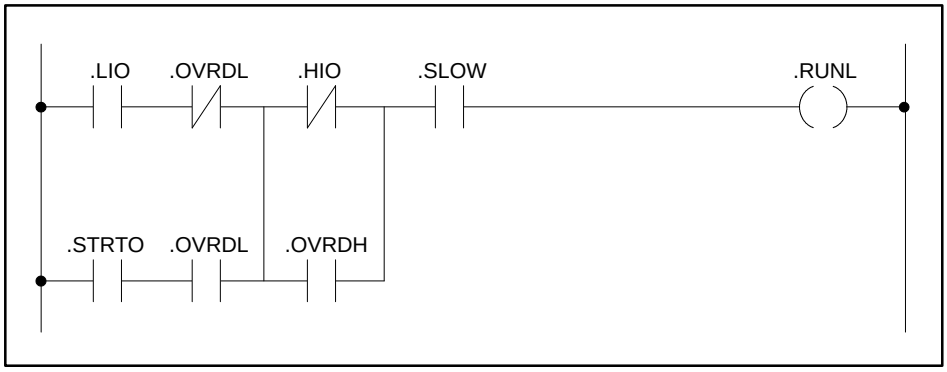


Figure A-200 TS1: RUNL

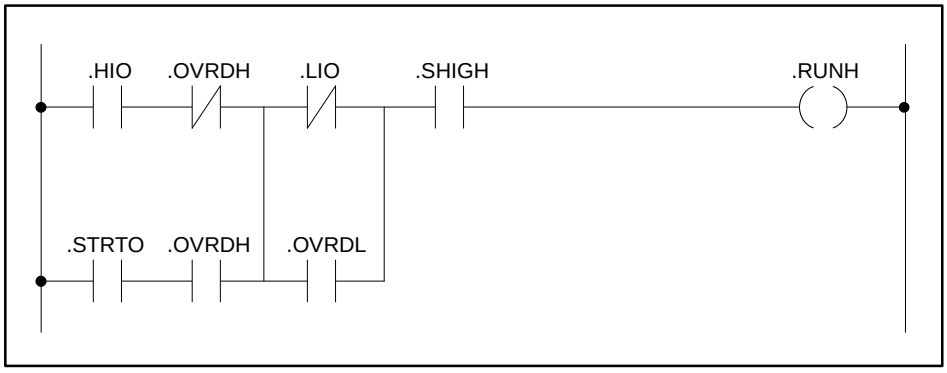


Figure A-201 TS1: RUNH

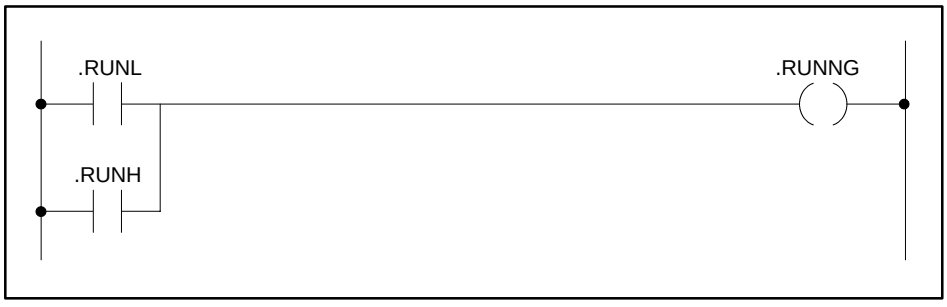


Figure A-202 TS1: RUNNG

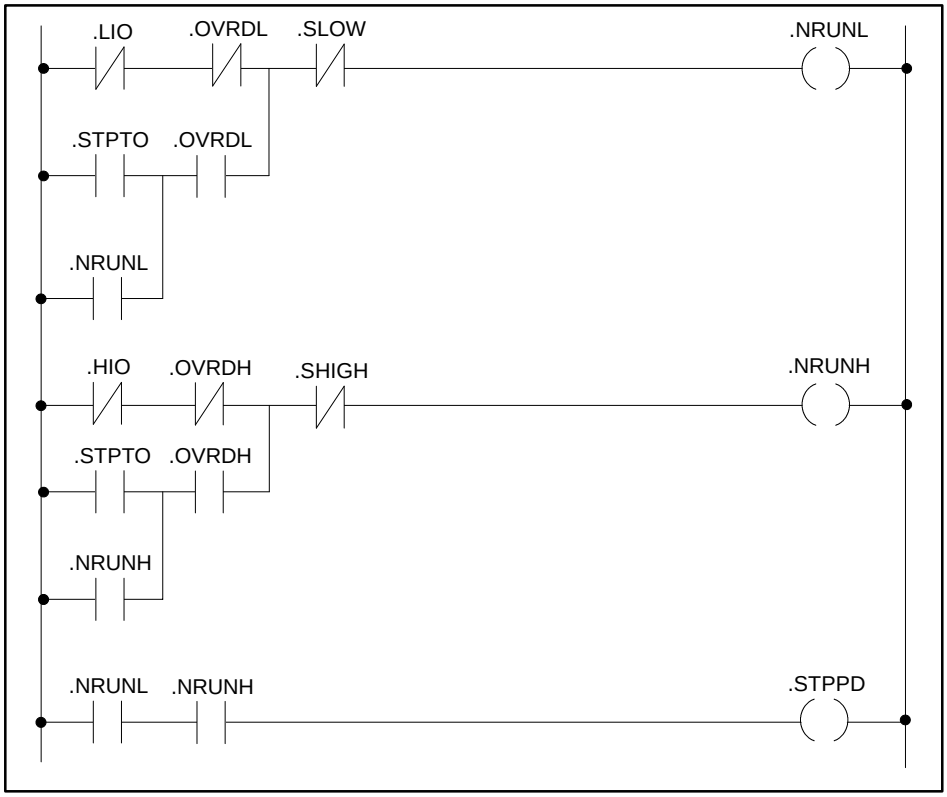


Figure A-203 TS1: STPPD

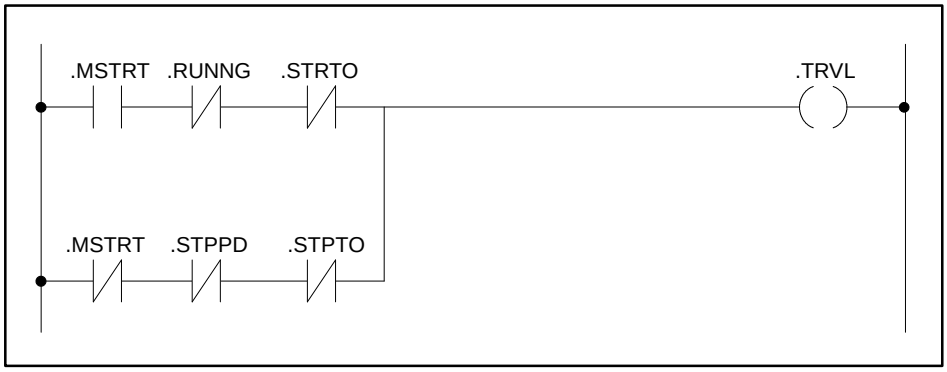


Figure A-204 TS1: TRVL

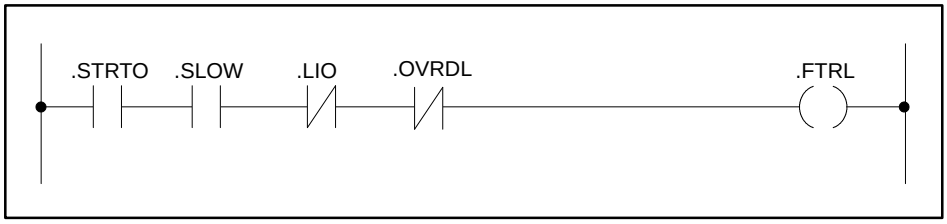


Figure A-205 TS1: FTRL

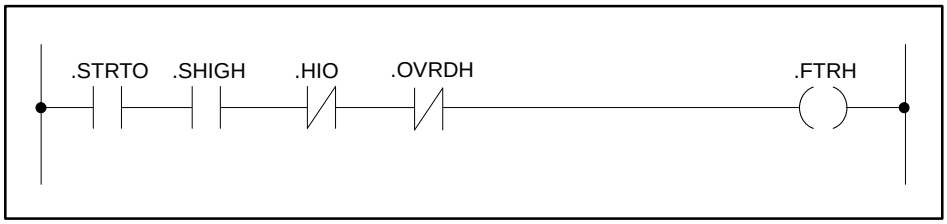


Figure A-206 TS1: FTRH

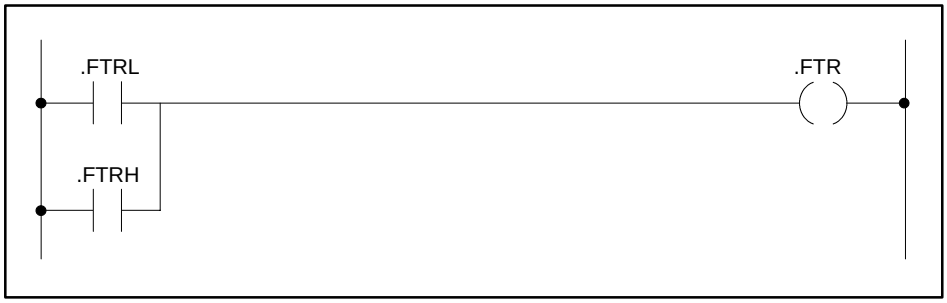


Figure A-207 TS1: FTR

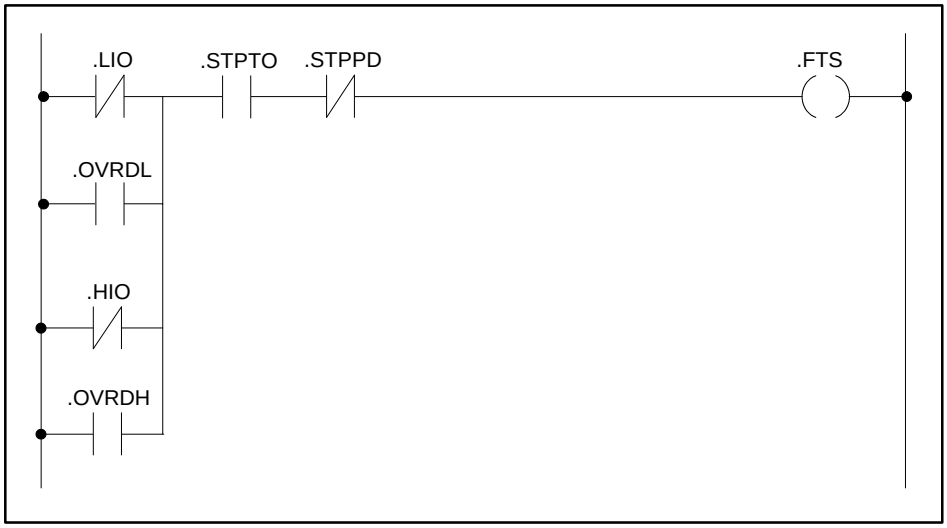


Figure A-208 TS1: FTS

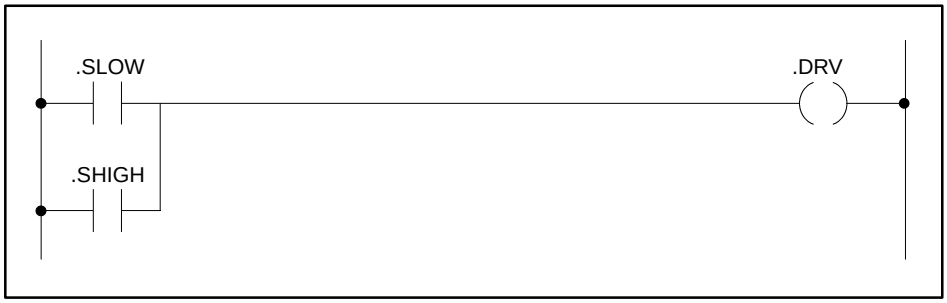


Figure A-209 TS2: DRV

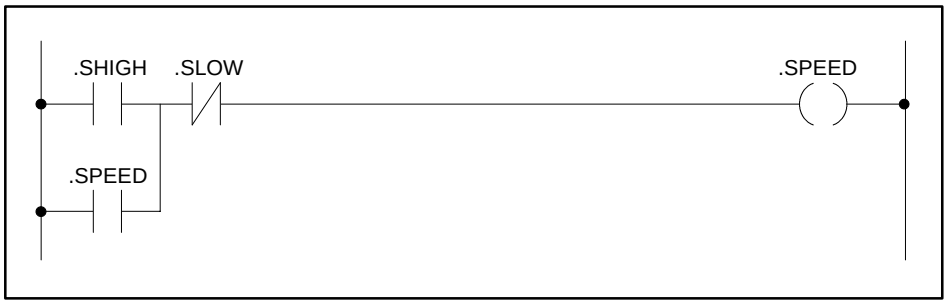


Figure A-210 TS2: SPEED

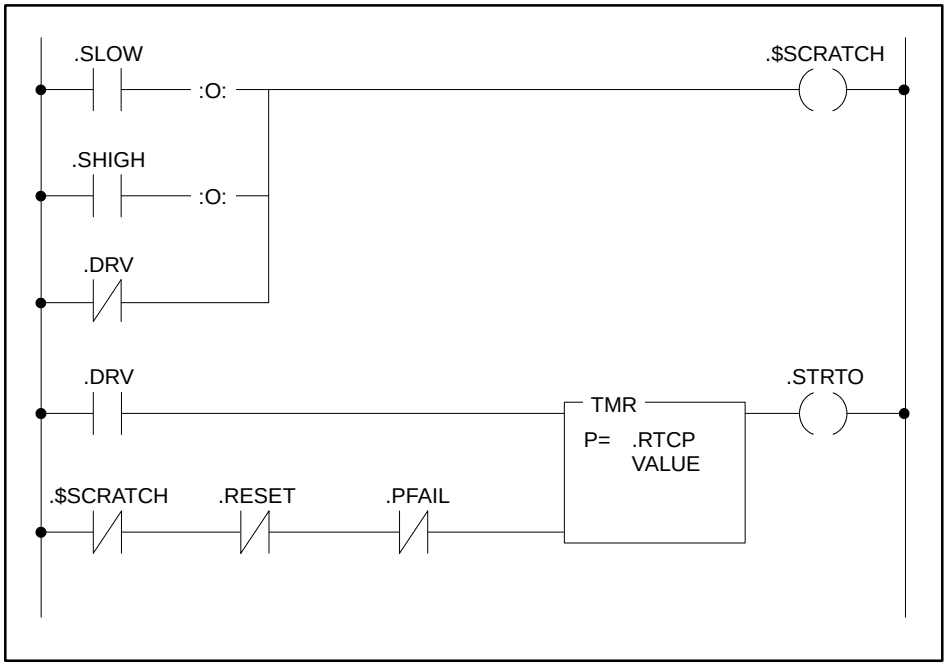


Figure A-211 TS2: STRTO

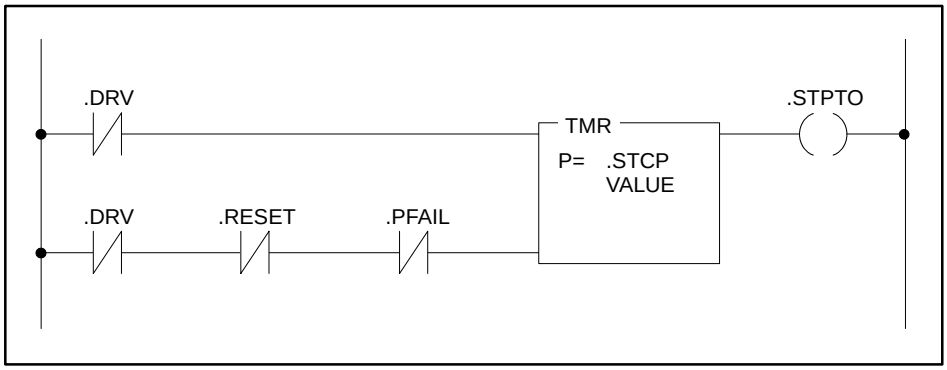


Figure A-212 TS2: STPTO

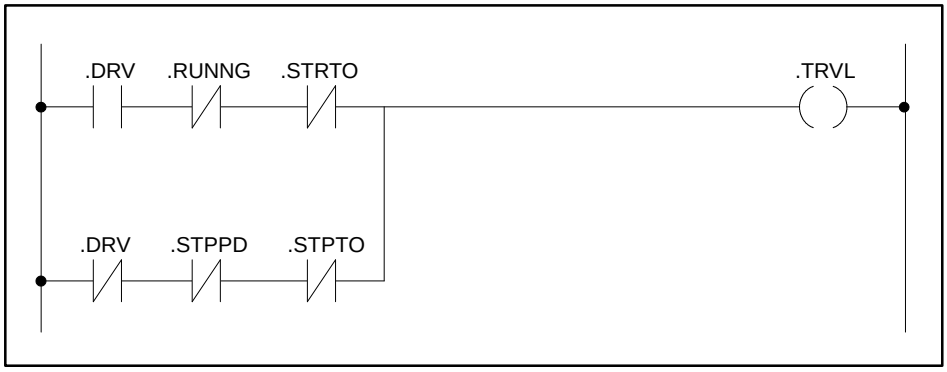


Figure A-213 TS2: TRVL

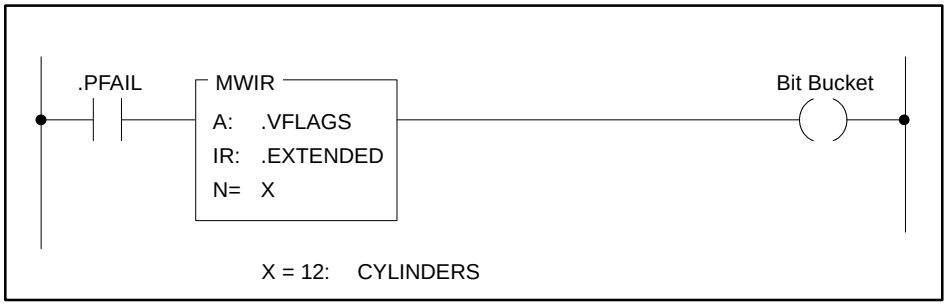


Figure A-214 CYLINDERS: Move Image from V

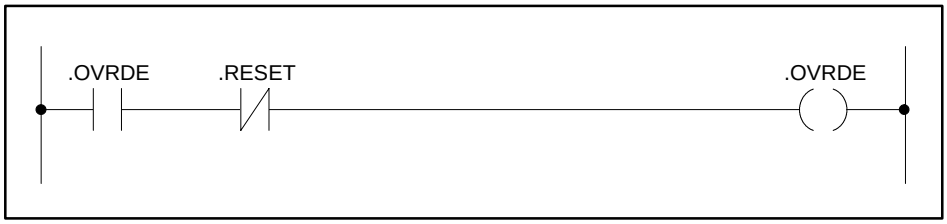


Figure A-215 CSD: OVRDE

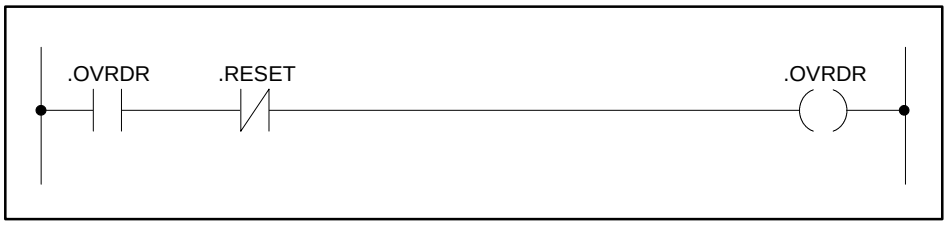


Figure A-216 CSD: OVRDR

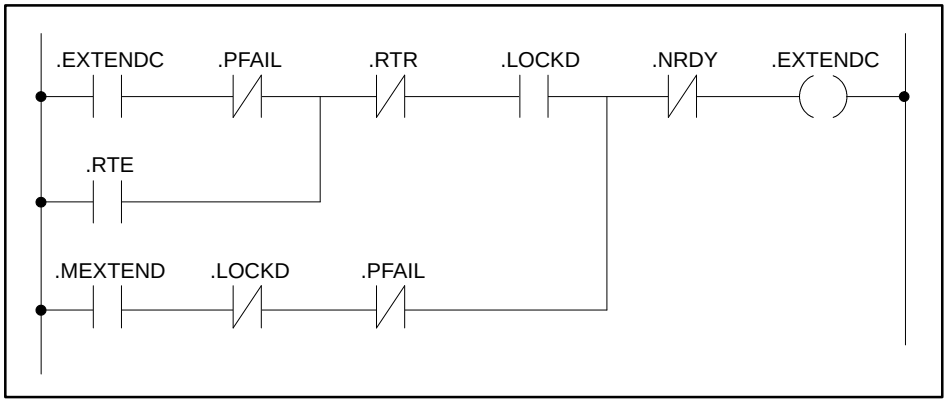


Figure A-217 CSD: CMMD (EE)

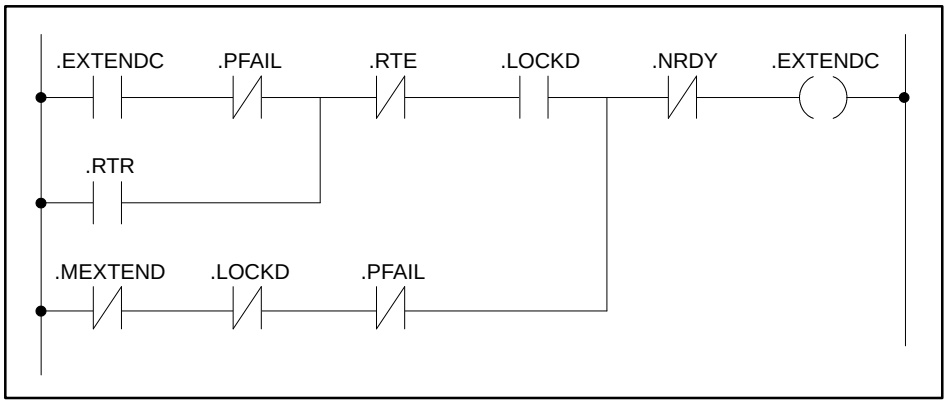


Figure A-218 CSD: CMMD (ER)

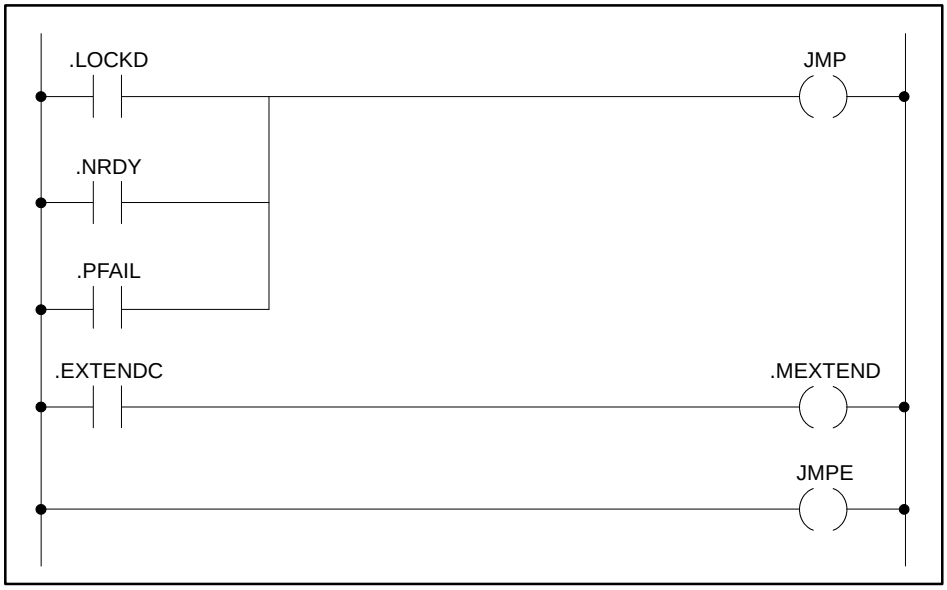


Figure A-219 CSD: MEXTEND (EE)

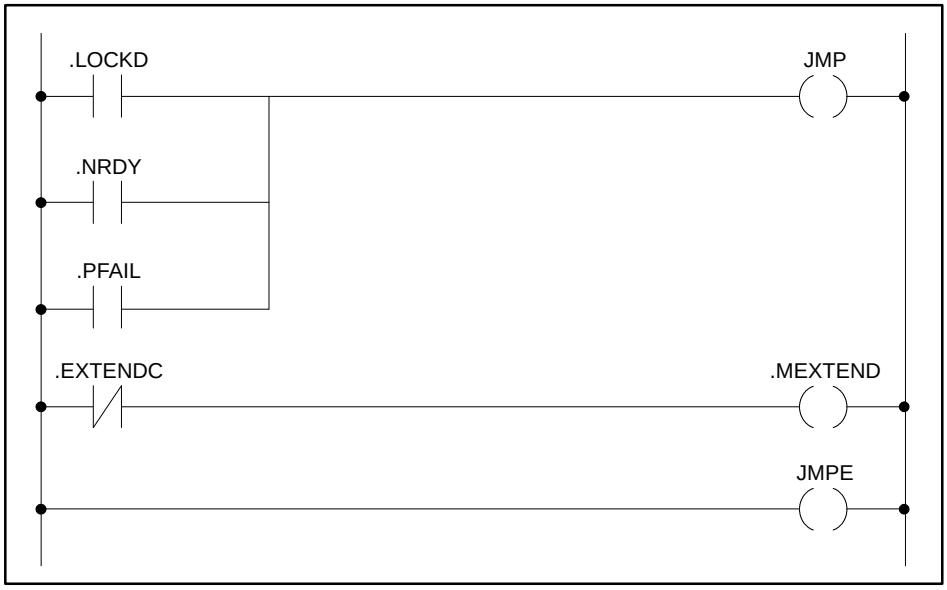


Figure A-220 CSD: MEXTEND (ER)

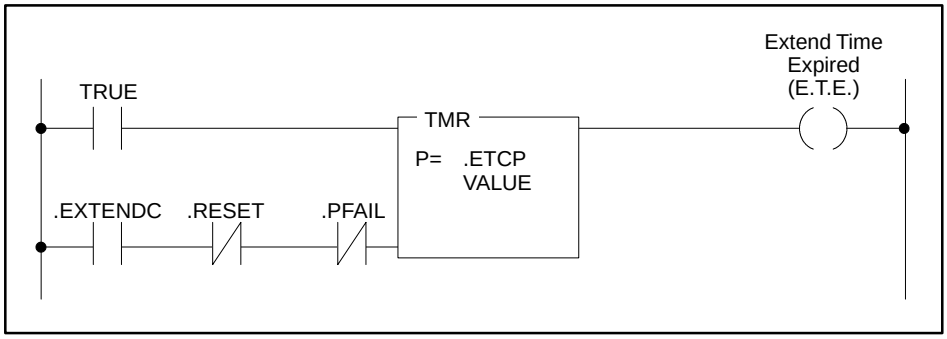


Figure A-221 CSD: EXTEND TIME EXPIRED [E.T.E] (EE)

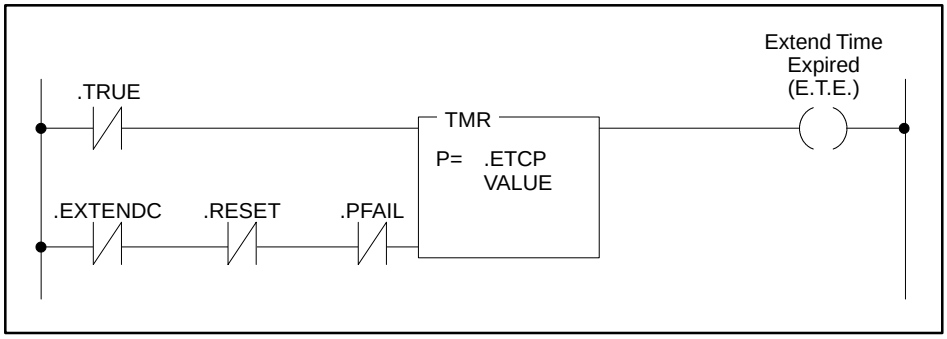


Figure A-222 CSD: EXTEND TIME EXPIRED [E.T.E] (ER)

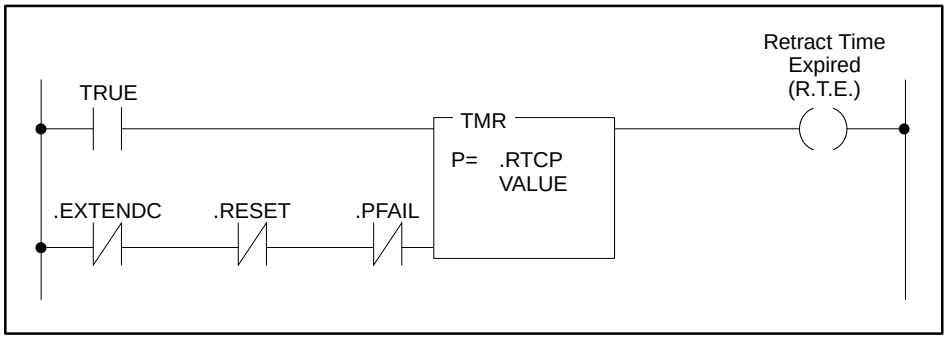


Figure A-223 CSD: RETRACT TIME EXPIRED [R.T.E] (EE)

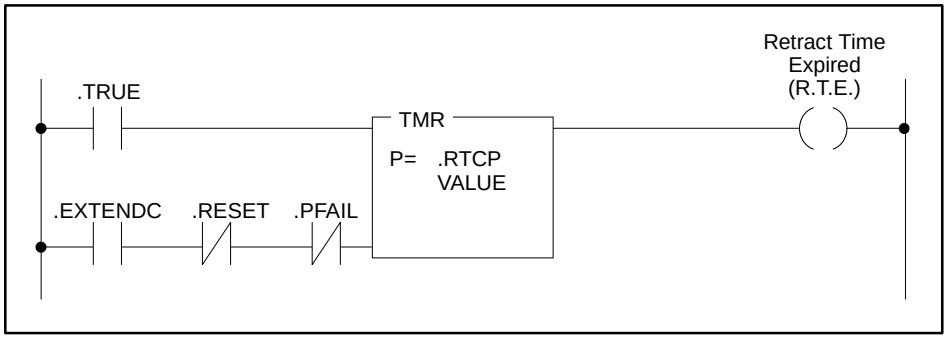


Figure A-224 CSD: RETRACT TIME EXPIRED [R.T.E] (ER)

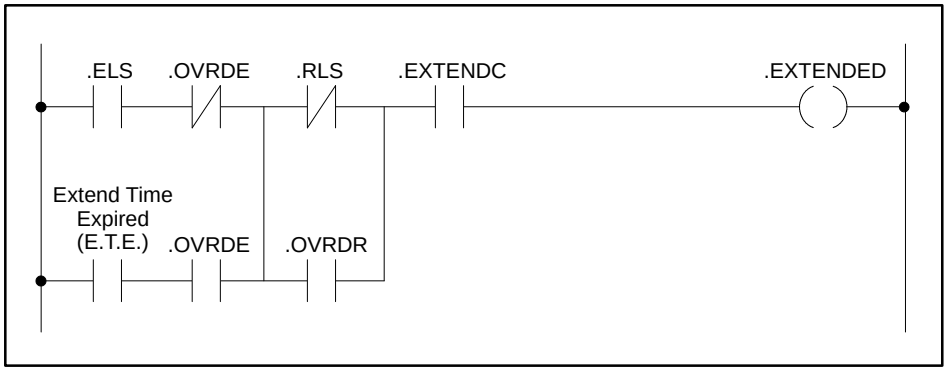


Figure A-225 CSD: EXTENDED (EE)

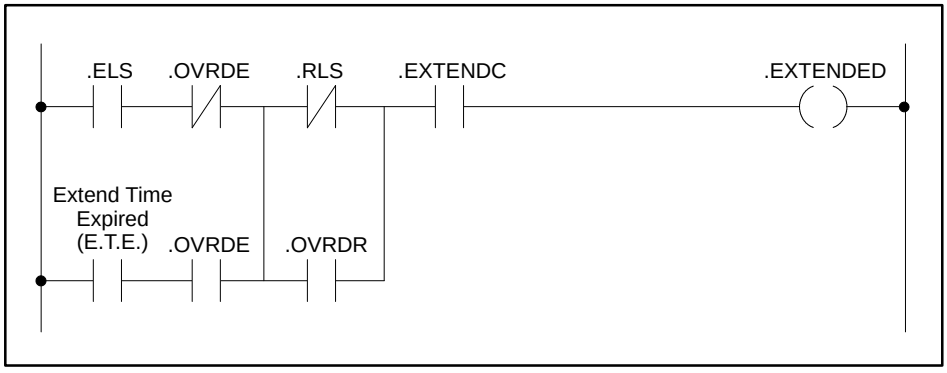


Figure A-226 CSD: EXTENDED (ER)

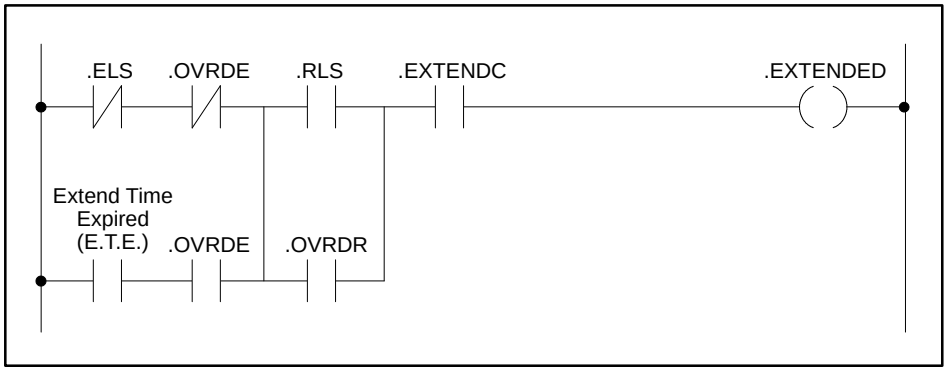


Figure A-227 CSD: EXTENDED (EE) (N.O. FDBK)

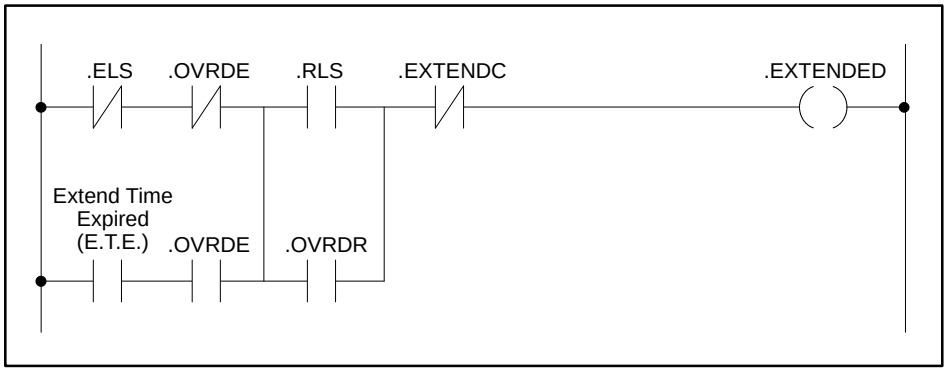


Figure A-228 CSD: EXTENDED (ER) (N.O. FDBK)

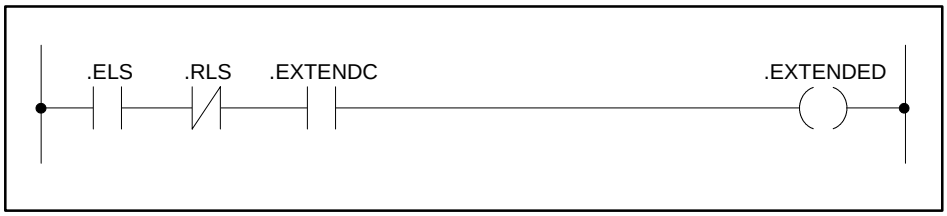


Figure A-229 CSD: EXTENDED (EE) (IGNORE FDBK OVRD)

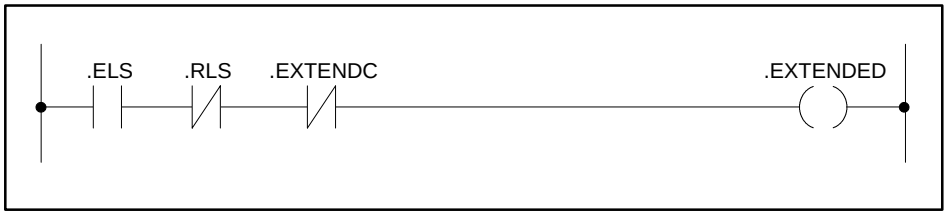


Figure A-230 CSD: EXTENDED (ER) (IGNORE FDBK OVRD)

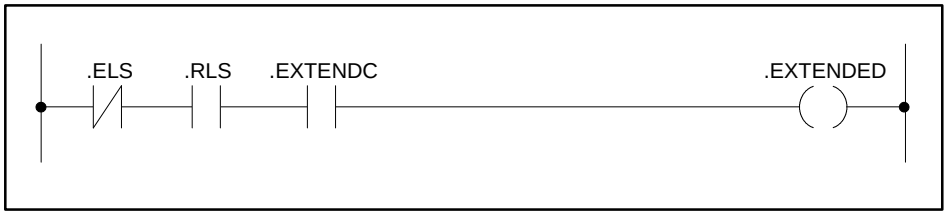


Figure A-231 CSD: EXTENDED (EE) (N.O. FDBK) (IGNORE FDBK OVRD)

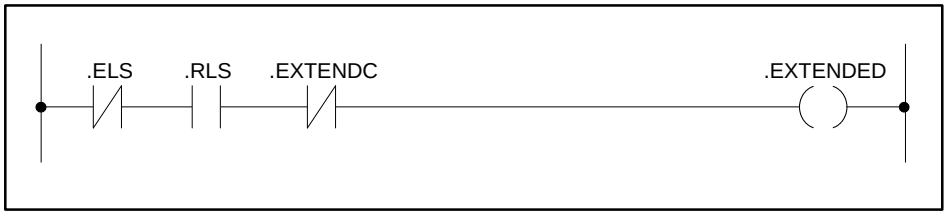


Figure A-232 CSD: EXTENDED (ER) (N.O. FDBK) (IGNORE FDBK OVRD)

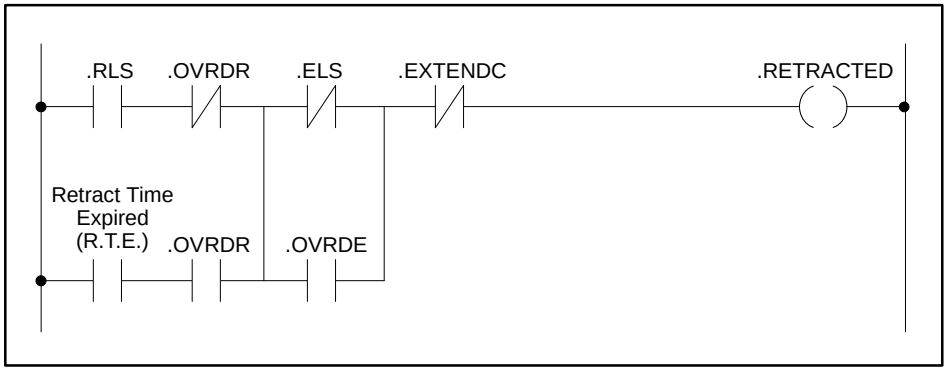


Figure A-233 CDS: RETRACTED (EE)

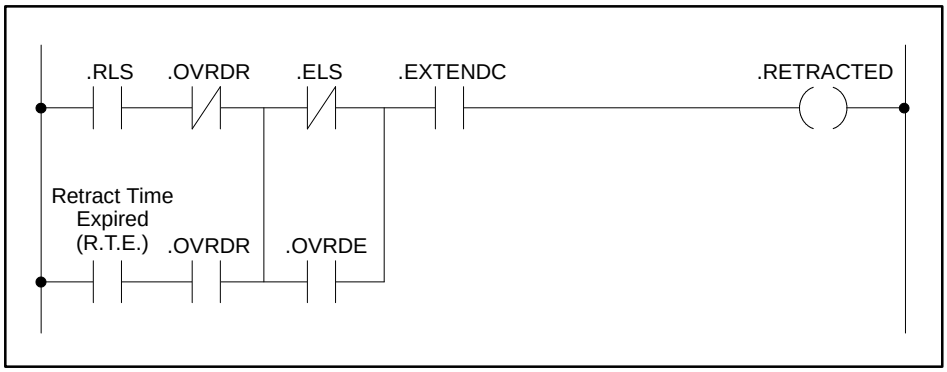


Figure A-234 CSD: RETRACTED (ER)

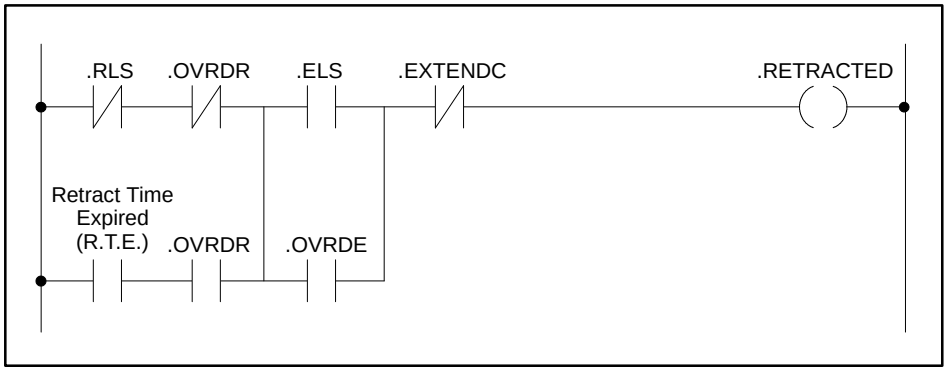


Figure A-235 CSD: RETRACTED (EE) (N.O. FDBK)

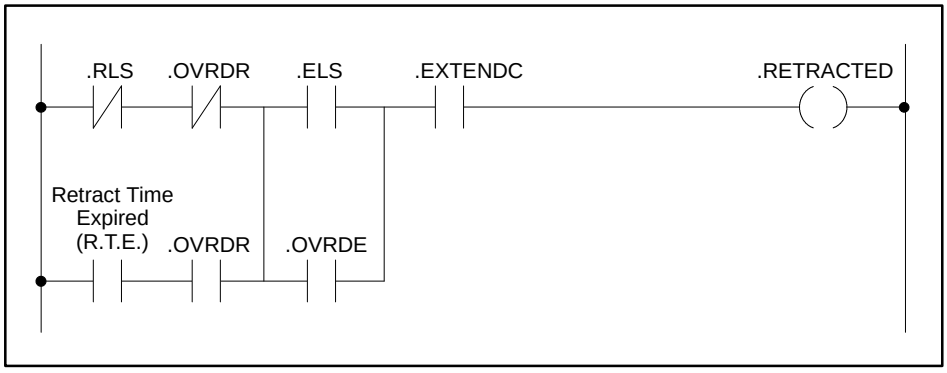


Figure A-236 CSD: RETRACTED (ER) (N.O. FDBK)

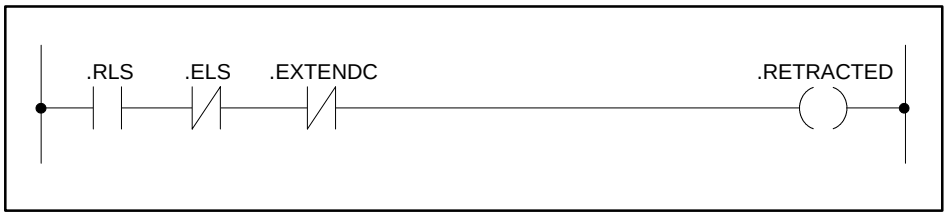


Figure A-237 CSD: RETRACTED (EE) (IGNORE FDBK OVRD)

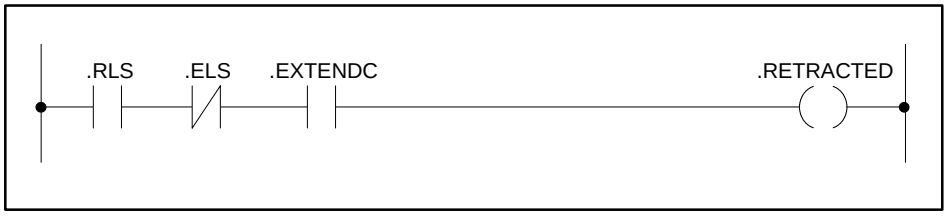


Figure A-238 CSD: RETRACTED (ER) (IGNORE FDBK OVRD)

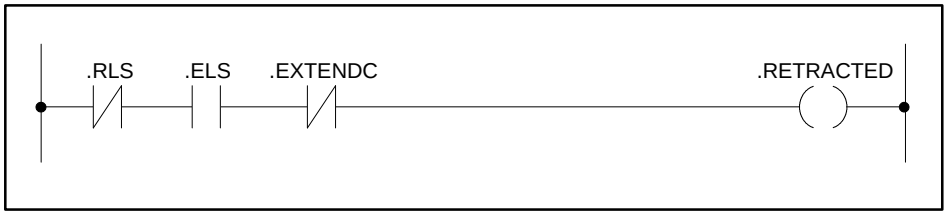


Figure A-239 CSD: RETRACTED (EE) (N.O. FDBK) (IGNORE FDBK OVRD)

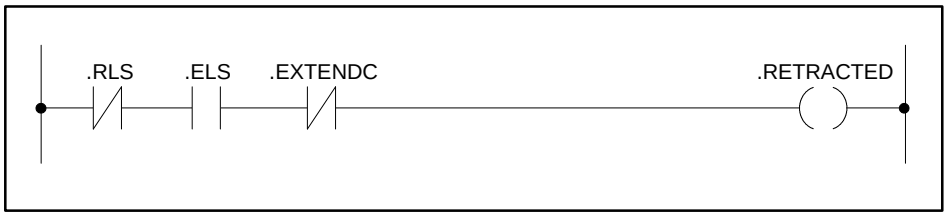


Figure A-240 CSD: RETRACTED (ER) (N.O. FDBK) (IGNORE FDBK OVRD)

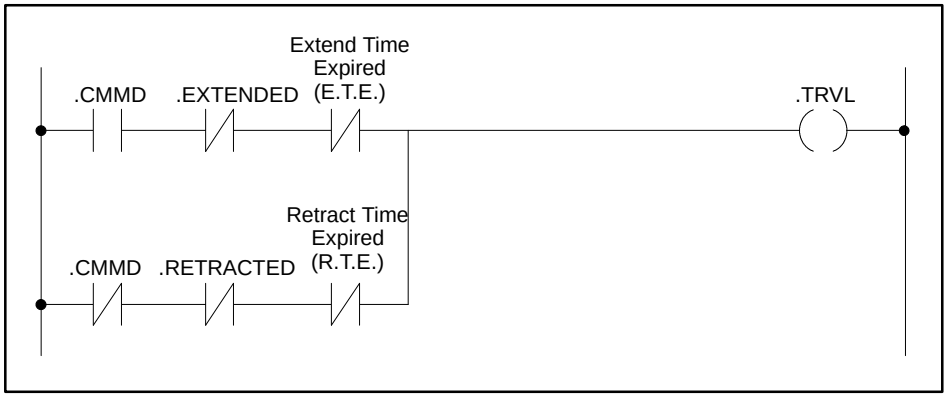


Figure A-241 CSD: TRVL (EE)

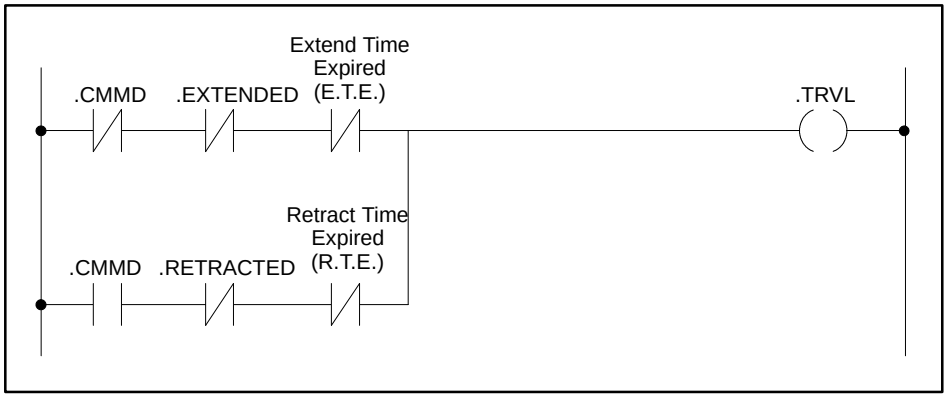


Figure A-242 CSD: TRVL (ER)

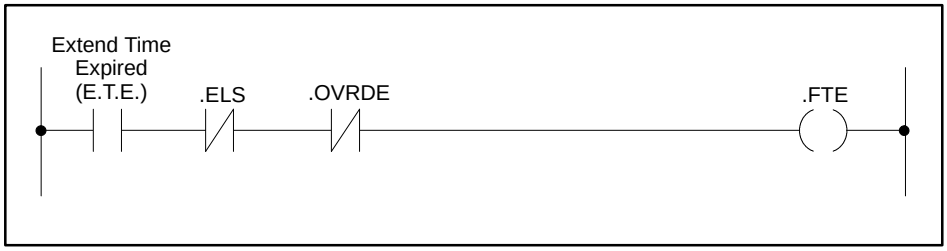


Figure A-243 CSD: FTE

Figure A-244 CSD: FTE (IGNORE FDBK OVRD)

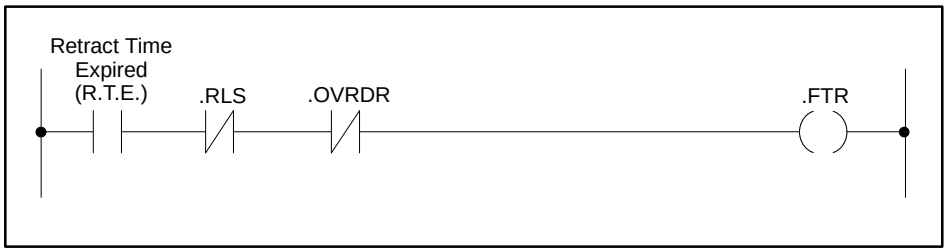


Figure A-245 CSD: FTR

Figure A-246 CSD: FTR (IGNORE FDBK OVRD)

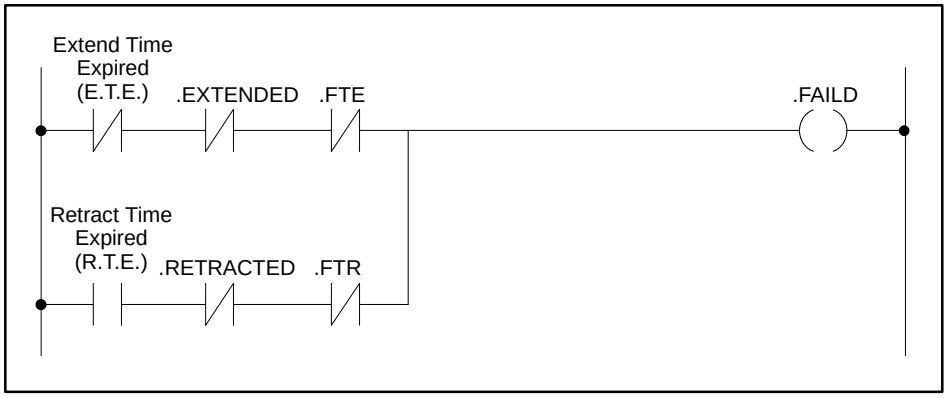


Figure A-247 CSD: FAILD

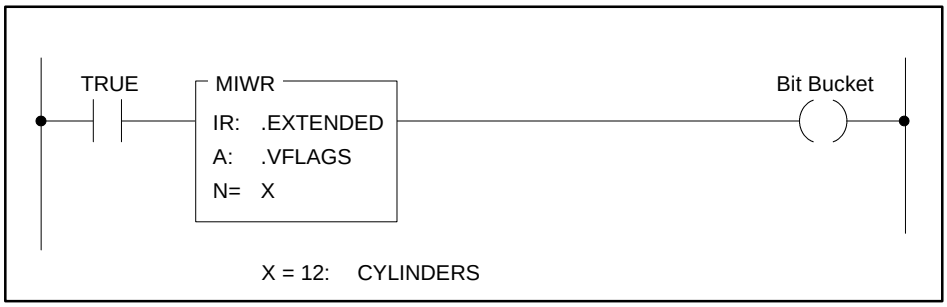


Figure A-248 CYLINDERS: Move image to V

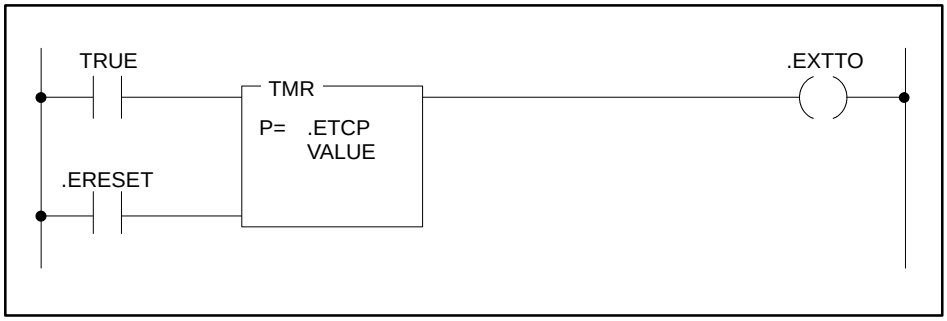


Figure A-249 CUD: EXTTO

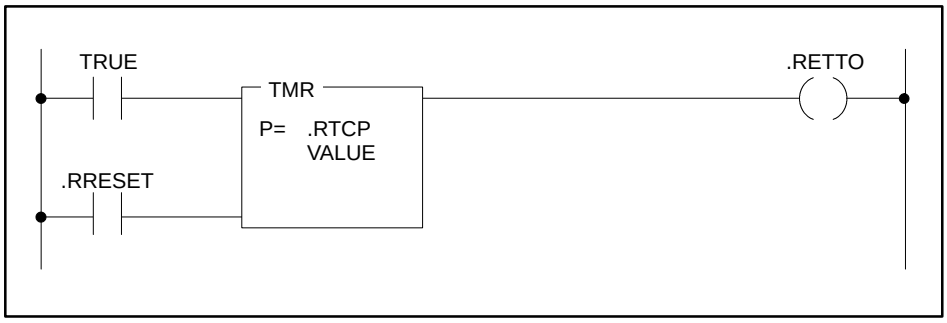


Figure A-250 CUD: RETTO

Index

Numbers

A

B

C

D

E

F

H

I

K

L

M

N

O

P

R

S

T

U

V

W

Z

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: SIMATIC APT Programming Reference (Tables) Manual **Edition:** Tenth
Manual Assembly Number: 2801048-0007 **Date:** 4/01
Order Number: PPX:APT-8102-10

FOLD

SIEMENS ENERGY & AUTOMATION INC
3000 BILL GARLAND RD
P O 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
P O BOX 1255
JOHNSON CITY TN 37605-1255

FOLD