

AutoMax®  
Ladder Logic  
Language

# Industrial

---

# CONTROLS

---

Instruction Manual J-3677-4



The information in this user's manual is subject to change without notice.

**DANGER**

**ONLY QUALIFIED ELECTRICAL PERSONNEL FAMILIAR WITH THE CONSTRUCTION AND OPERATION OF THIS EQUIPMENT AND THE HAZARDS INVOLVED SHOULD INSTALL, ADJUST, OPERATE, OR SERVICE THIS EQUIPMENT. READ AND UNDERSTAND THIS MANUAL AND OTHER APPLICABLE MANUALS IN THEIR ENTIRETY BEFORE PROCEEDING. FAILURE TO OBSERVE THIS PRECAUTION COULD RESULT IN SEVERE BODILY INJURY OR LOSS OF LIFE.**

**WARNING**

**PROGRAMS INSERTED INTO THE PRODUCT SHOULD BE REVIEWED BY QUALIFIED PERSONNEL WHO ARE FAMILIAR WITH THE CONSTRUCTION AND OPERATION OF THE SYSTEM AND THE POTENTIAL HAZARDS INVOLVED. FAILURE TO OBSERVE THIS PRECAUTION COULD RESULT IN BODILY INJURY.**

**WARNING**

**THE USER MUST PROVIDE AN EXTERNAL, HARDWIRED EMERGENCY STOP CIRCUIT OUTSIDE THE CONTROLLER CIRCUITRY. THIS CIRCUIT MUST DISABLE THE SYSTEM IN CASE OF IMPROPER OPERATION UNCONTROLLED MACHINE OPERATION MAY RESULT IF THIS PROCEDURE IS NOT FOLLOWED. FAILURE TO OBSERVE THIS PRECAUTION COULD RESULT IN BODILY INJURY.**

Norton® is a registered trademark of Peter Norton Computing, Inc.

MODBUS® is a registered trademark of Gould, Inc.

IBM™ is a trademark of International Business Machines.

Reliance®, AutoMax®, and AutoMate® are registered trademarks of Reliance Electric Company or its subsidiaries.

Shark™, R-Net™, and ReSource™ are trademarks of Reliance Electric Company or its subsidiaries.

# Table of Contents

<b>1.0</b>	<b>INTRODUCTION</b>	<b>1-1</b>
1.1	Compatibility with Earlier Versions	1-1
1.3	Related Hardware and Software	1-2
<b>2.0</b>	<b>PROGRAMMING FOR AutoMax SYSTEMS</b>	<b>2-1</b>
2.1	Configuration	2-1
2.2	Application Tasks	2-2
2.3	Variables and Constants	2-3
2.3.1	Variables	2-3
2.3.2	Subscripted Variables (Arrays)	2-4
2.3.3	Variable Control Types	2-6
2.3.4	Pre-defined Common Memory Variables	2-7
<b>3.0</b>	<b>LADDER LOGIC LANGUAGE DESCRIPTION</b>	<b>3-1</b>
3.1	Application Program Structures	3-1
3.2	Variable Names in Ladder Logic	3-3
3.3	Variable Types in Ladder Logic	3-3
<b>4.0</b>	<b>LADDER LOGIC LANGUAGE OPERATIONS</b>	<b>4-1</b>
4.1	Contacts	4-1
4.1.1	Normally Open Contact	4-1
4.1.2	Normally Closed Contact	4-1
4.1.3	Upward Transition Contact	4-2
4.1.4	Downward Transition Contact	4-2
4.2	Output (Coil) Operations	4-3
4.2.1	Coil	4-3
4.2.2	Timer	4-3
4.2.3	Counter	4-5
4.2.4	Shift Register	4-6
4.2.5	Event Coil	4-7
4.3	REMARK Sequence	4-8
<b>5.0</b>	<b>TASK EXECUTION</b>	<b>5-1</b>
5.1	Buffering	5-1
<b>6.0</b>	<b>STOPPING A TASK CONTAINING COILS WITH INTERNAL MEMORY</b>	<b>6-1</b>
6.1	Task Stopped with STOP Command	6-1
6.2	Task Stopped Through Power Loss or with STOP ALL Command	6-1
<b>7.0</b>	<b>EXECUTION TIME AND MEMORY USAGE ESTIMATES</b>	<b>7-1</b>

# Appendices

## Appendix A

Converting a DCS 5000 Ladder Logic Task to the Current Version of the AutoMax Executive Software .....	A-1
---	-----

# List of Figures

Figure 3.1	- Ladder Sequence Structure .....	3-2
Figure 3.2	- Sample Printout of Task Listing .....	3-3
Figure 3.3	- Arrays Used As Contact and Coil .....	3-4
Figure 4.1	- Normally Open Contact .....	4-1
Figure 4.2	- Normally Closed Contact .....	4-2
Figure 4.3	- Upward Transition Contact .....	4-2
Figure 4.4	- Downward Transition Contact .....	4-2
Figure 4.5	- Coil Operation .....	4-3
Figure 4.6	- Timer Output Blocks .....	4-4
Figure 4.7	- Operation of On-Delay Timer .....	4-4
Figure 4.8	- Operation of Off-Delay Timer .....	4-5
Figure 4.9	- Counter .....	4-6
Figure 4.10	- Shift Register .....	4-7
Figure 4.11	- Event Coil .....	4-7
Figure 4.12	- REMARK Sequence .....	4-8

# List of Tables

Execution Time and Memory Usage Estimates .....	7-1
---	-----



# 1.0 INTRODUCTION

The products described in this manual are manufactured or distributed by Reliance Electric Industrial Company.

The AutoMax Programming Executive software includes the software used to create Ladder Logic programs. Instruction manual J-3684 describes the AutoMax Programming Executive software Version 2.0. Instruction manual J-3750 describes the AutoMax Programming Executive software Version 3.0. Instruction manual J2-3045 describes the AutoMax Programming Executive software Version 3.3. Instruction manual J2-3066 describes the AutoMax Programming Executive software Version 3.4. This instruction manual describes AutoMax Ladder Logic language for version 2.0 and later AutoMax Programming Executive software.

Features that are either new or different from those in version 1.0 AutoMax Programming Executive software (M/N 57C304 through 57C307) are so noted.

This instruction manual is organized as follows:

- 1.0 Introduction
  - 2.0 General information about programming for AutoMax systems
  - 3.0 General information about programming in Ladder Logic/PC
  - 4.0 Ladder Logic operations
  - 5.0 Task execution
  - 6.0 Stopping tasks with internal memory coils
  - 7.0 Execution time and memory usage
- Appendix A Converting tasks created with previous versions of the Executive software to the current version
- Appendix B AutoMax Processor Compatibility with versions of the AutoMax Programming Executive

## 1.1 Compatibility with Earlier Versions

Version 2.0 of the AutoMax Programming Executive requires AutoMax Processor M/N 57C430A or 57C431; Version 3.0 and later require AutoMax Processor M/N 57C430A, 57C431, or 57C435. M/N 57C430 cannot co-exist in the same rack with M/N 57C430A, 57C431, or 57C435.

Refer to Appendix B for a listing of the AutoMax Processor modules that are compatible with Version 2 and later of the AutoMax Programming Executive software.

The thick black bar shown at the right-hand margin of this page will be used throughout this instruction manual to signify new or revised text or figures.

## 1.2 Additional Information

You should be familiar with the instruction manuals which describe your system configuration. This may include, but is not limited to, the following:

- J-3618 NORTON EDITOR REFERENCE MANUAL
- J-3649 AutoMax CONFIGURATION TASK INSTRUCTION MANUAL
- J-3650 AutoMax PROCESSOR INSTRUCTION MANUAL
- J-3675 AutoMax ENHANCED BASIC INSTRUCTION MANUAL
- J-3676 AutoMax CONTROL BLOCK LANGUAGE INSTRUCTION MANUAL
- J2-3018 AutoMax REMOTE I/O SHARK INTERFACE INSTRUCTION MANUAL
- Your ReSource AutoMax PROGRAMMING EXECUTIVE INSTRUCTION MANUAL
- Your personal computer and DOS instruction manuals
- IEEE 518 GUIDE FOR THE INSTALLATION OF ELECTRICAL EQUIPMENT TO MINIMIZE ELECTRICAL NOISE INPUTS TO CONTROLLERS

## 1.3 Related Hardware and Software

M/N 57C390 contains the AutoMax Programming Executive software Version 2.0 on 5<sup>1</sup>/<sub>4</sub>" floppy disks and a set of hardware and software reference manuals. M/N 57C391 contains the AutoMax Programming Executive software Version 2.0 on 3<sup>1</sup>/<sub>2</sub>" floppy disks and a set of hardware and software reference manuals. M/N 57C395 contains the AutoMax Programming Executive software Version 3.X on both 5<sup>1</sup>/<sub>4</sub>" and 3<sup>1</sup>/<sub>2</sub>" floppy disks and a set of hardware and software reference manuals. The two model numbers are sold separately and are identical except for the size of the floppy disks. The Programming Executive software includes the tools required for programming in Enhanced BASIC, Control Block, and Ladder Logic/PC languages.

M/N 57C392 and 57C393 are the AutoMax Programming Executive Version 2.0 software updates. The two model numbers are sold separately and are identical except for the disk size. M/N 57C397 is the AutoMax Programming Executive Software Version 3.X update. These updates effectively upgrade your existing software to the most current version.



The AutoMax Programming Executive software is used with the following hardware, which is sold separately.

1. M/N 57C430A, 57C401, or 57C435 AutoMax Processor.
2. IBM-compatible 80386-based personal computer running DOS Version 3.1 or later.
3. M/N 61C127 RS-232C ReSource Interface Cable. This cable is used to connect the personal computer to the Processor module.
4. M/N 57C404A (and later) Network Communications module. This module is used to connect racks together as a network and supports communication with all racks on the network that contain 57C404A modules through a single Processor module. M/N 57C404 can be used to connect racks on a network; however, you cannot communicate over the network to the racks that contain M/N 57C404 Network modules. You must instead connect directly to the Processors in those racks.
5. M/N 57C413 or 57C423 Common Memory module. This module is used when there is more than one Processor module in the rack.
6. M/N 57C492 Battery Back-Up. This unit is used when there is a M/N 57C413 Common Memory module in the rack.
7. M/N 57C384 Battery Back-Up Cable. This cable is used with the Battery Back-Up unit.
8. M/N 57C554 AutoMax Remote I/O Shark Interface Module. This module is used to connect a Shark remote rack to the AutoMax Remote I/O network.
9. M/N 57552 Universal Drive Controller module. This module is used for drive control applications.



## 2.0 PROGRAMMING FOR AutoMax SYSTEMS

In AutoMax systems, application programs, also referred to as tasks, can be written in Ladder Logic/PC language, Control Block language, and Enhanced BASIC language. Refer to J-3675 and J-3676 for more information about BASIC and Control Block programming.

In addition to multi-processing, AutoMax systems incorporate multi-tasking. This means that each AutoMax Processor (up to four) in a rack allows real-time concurrent operation of multiple application tasks.

Multi-tasking features allow the programmer's overall control scheme to be separated into individual tasks, each written in the programming language best suited to the task. This simplifies writing, check-out, and maintenance of programs; reduces overall execution time; and provides faster execution for critical tasks.

Programming in AutoMax systems consists of configuration, or defining the hardware, system-wide variables, and application tasks in that system, as well as application programming.

### 2.1 Configuration

#### Version 3.0 and Later Systems

If you are using AutoMax Version 3.0 or later, you configure the system within the AutoMax Programming Executive. See the AutoMax Programming Executive instruction manual for information about configuration if you are using Version 3.0 or later. The information on configuration that follows is applicable only if you are using AutoMax Version 2.1 or earlier. If you are using AutoMax 3.0 or later, you can skip over the remainder of this section and continue with 2.2.

#### Version 2.1 and Earlier Systems

AutoMax Version 2.1 and earlier requires a configuration task in order to define the following:

1. All tasks that will reside on the Processors in a rack.
2. All variables that equate to physical I/O in the system.
3. All other variables that must be accessible to all Processors in the rack.

One configuration task is required for each rack that contains at least one Processor. The configuration task must be loaded onto the Processor(s) in the rack before any application task can be executed because it contains information about the physical organization of the entire system.

The configuration task does not actually execute or run; it serves as a central storage location for system-wide information. Note that local variables, those variables that do not need to be accessible to more than one task, do not need to be defined in the configuration task. Refer to J-3649 for more information about configuration tasks.

## 2.2 Application Tasks

AutoMax Processors allow real-time concurrent operation of multiple programs, or application tasks, on the same Processor module. The tasks are executed on a priority basis and share all system data. Application tasks on different Processor modules in the rack are run asynchronously.

Each task operates on its own variables. The same variable names may be used in different tasks, but each variable is only recognized within the confines of its task unless it is specifically designated a COMMON variable. Changing local variable ABC% (designated LOCAL) in one task has no effect on variable ABC% in any other task.

Multi-tasking in a control application can be compared to driving a car. The programmer can think of the different functions required as separate tasks, each with its own priority.

In driving a car, the operator must monitor the speedometer, constantly adjust the pressure of his foot on the gas pedal, check the rearview mirror for other traffic, stay within the boundaries of his lane, etc., all while maintaining a true course to his destination. All of these functions have an importance or priority attached to them, with keeping the car on the road being the highest priority. Some tasks, like monitoring the gasoline gauge, require attention at infrequent intervals. Other tasks require constant monitoring and immediate action, such as avoiding obstacles on the road.

In a control application the Processor needs to be able to perform calculations necessary for executing a control scan loop, monitor an operator's console, log error messages to the console screen, etc. Of these tasks, executing the main control loop is obviously the most important, while logging error messages is the least important. Multi-tasking allows the control application to be broken down into such tasks, with their execution being dependent upon specified "events," such as an interrupt, operator input, or the expiration of a time interval.

The following table is a representation of typical tasks found in a control application and the kind of event that might trigger each.

<u>Task</u>	<u>Triggering Event</u>
Execute main control loop	Expiration of a hardware timer that indicates the interval at which to begin a new scan
Respond to external I/O input	Generation of a hardware interrupt by an input module
Read operator data	Input to an operator panel
Log information	Expiration of a software timer

Each of these tasks would be assigned a priority level (either in the specific configuration task for the rack, or in later versions of the Programming Executive software, through the configuration option). The priority determines which task should run at any particular instant. The more important the task, the higher the task priority.

## 2.3 Variables and Constants

All operations performed in AutoMax tasks use constants or variables. Constants, also known as literals, are quantities with fixed value represented in numeric format. Variables are names that represent stored values or physical I/O. These values may change during program execution. BASIC language tasks always use the current value of a common (i.e., system-wide) variable in performing calculations.

Control Block and PC/Ladder Logic tasks capture (latch) the values of all common simple double integer, integer, and boolean variables at the beginning of the task scan. Strings, reals, and array variables of any type are not latched. This means that control Block and PC/Ladder Logic tasks do not see the most current state of common simple double integer, integer, and boolean variables; instead, they see the state of these variables at the beginning of the scan. Any changes made to these variable values by Control Block or PC/Ladder Logic tasks are written to the variable locations at the end of the scan of this particular task. See section 2.3.3 for more information about common variables.

### 2.3.1 Variables

The following section describes the variable types in AutoMax tasks. When constants are used in tasks, they must fall into the ranges specified. For example, an integer constant must be in the range specified for integers, +32767 to -32768. Note carefully that all variable types cannot be used in all programming languages.

#### 1. Long Integer or Double Integer Variables

Used to store 32 bits. The value can be in the range +2147483647 to -2147483648 with no fractional part. The terminating character is !. If you assign a real number (see #3 below) to an integer variable, the fractional part will be truncated.

#### 2. Integer or Single Integer Variables

Used to store 16 bits. The value can be in the range +32767 to -32768 with no fractional part. The terminating character is %. If you assign a real number (see #3 below) to the variable, the fractional part will be truncated. Note that all internal integer calculations are in double precision, or 32 bits.

#### 3. Real Variables

Used to store a decimal value. The value can be in the following ranges:

$9.2233717 \times 10^{**18} >$  positive value >  
 $5.4210107 \times 10^{**(-20)}$   
 $-9.2233717 \times 10^{**18} >$  negative value >  
 $-2.7105054 \times 10^{**(-20)}$

There is no terminating character for real variables. Use scientific notation to enter large numbers. Use double asterisks to indicate exponentiation.

Only eight digits of significance are accepted. The format for entering real constants is as follows:

{sign}{digits}{.} {digits} {E}{sign}{digits}

For example: -1234.5678E+11

Real variables cannot be used in Ladder Logic tasks.

#### 4. Boolean Variables

Used to store the status of 1 bit. The value can be either TRUE or FALSE or ON or OFF. The terminating character is @. Note that in BASIC tasks, the inverted sense (negative of the current state) of a boolean variable is indicated with NOT. In Control Block language only, the inverted sense of a boolean variable can be indicated by entering a minus sign in front of the name when referencing it in the application task. For example, STATUS@ indicates the normal sense of variable STATUS@, while -STATUS@ would indicate the inverted sense of variable STATUS@.

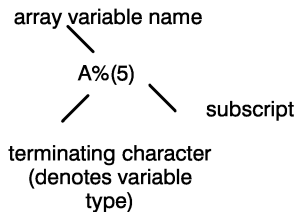
#### 5. String Variables

String variables are used to store any alphanumeric sequence of printable characters, including spaces, tabs, and special characters. The terminating character is \$. String variables cannot be used in Ladder Logic tasks.

### 2.3.2 Subscripted Variables (Arrays)

Array variables are used to store a collection of data all of the same data type. Arrays are permitted for all data types. Arrays are limited to four dimensions, or subscripts. The number of elements in each dimension is limited to 99999. This size is further limited by available memory. The term array is used to denote the entire collection of data. Each item in the array is known as an element.

Array variables are specified by adding a subscript(s) after the variable name which includes the appropriate terminating character to denote the type of data stored in the array. The terminating character is followed by a left parenthesis, the subscript(s), and a right parenthesis. Multiple subscripts are separated by commas. Note that subscripts can be integer constants as well as arithmetic expressions that result in integer values.



An array with one dimension, i.e., one subscript, is said to be one-dimensional. An array with two subscripts is said to be two-dimensional, etc. The first element in each dimension of the array is always element 0. Therefore, the total number of elements in each dimension of the array is always one more than the largest subscript. For example, array A%(10) is a one-dimensional array containing eleven integer values.

### Example 1 - One-dimensional array

A%	0	1	2	3	4	5
	185	2	53	79	99	122

|  
value of A

### Example 2 - Two-dimensional array

B% (6, 3)

	0	1	2	3	4	5	6	
B%	0	185	2	53	79	99	122	40
	1	70	36	46	31	34	85	6
	2	77	73	21	365	476	51	47
	3	18	23	53	342	39	224	107

In the case of string arrays, version 1.0 Executive software always allocated the maximum amount of memory for each element in the array, regardless of whether the string stored in that element was of the maximum length, 31 characters. Version 2.0 (and later) Executive software allows the programmer to specify the maximum size of elements in the array, from 1 to 255 characters.

To specify the maximum size of string variables in an array, add a colon and a number (1-255) immediately after the \$ character when declaring the variable in an application task or defining it during configuration. For example, defining A\$:10(20) as a local variable in an application task allocates space for 21 string values of 10 characters each. Note that if no length is specified in the initial array reference, the default maximum is 31.

To define an array that will be common, i.e., accessible to all tasks in the rack, you need to first define the variable. If you are using AutoMax Version 2.1 or earlier, this is done with a MEMDEF or NVMEMDEF statement in the configuration task for the rack. If you are using AutoMax Version 3.0 or later, common variables are defined within the Programming Executive. For example, ARRAY1@(10) will allocate space for 11 boolean variables. Then, in an application task for the rack, you declare the array a COMMON variable as follows:

```
COMMON ARRAY1@(10).
```

Each element of the array that will be used in the task can be defined with LET statements as follows:

```
LET ARRAY1@(0) = TRUE
```

(boolean values can only be TRUE/FALSE or ON/OFF). Other application tasks in the rack can access the value in variable ARRAY1@(0) simply by declaring it a COMMON variable.

## 2.3.3 Variable Control Types

The control type of a variable refers to the way the variable is declared or defined in the rack configuration and application tasks. There are two control variable types in AutoMax systems, local and common.

### 1. Local

Local variables are variables that are not defined in the configuration for the rack and are therefore accessible only to the application task in which they are defined. BASIC and Control Block tasks must define the variables with a BASIC LOCAL statement. For Ladder Logic/PC tasks, the editor prompts for whether the variable is local or common when the task is being created.

In BASIC and Control Block tasks, local variables can be defined as tunable. Tunables are variables whose value can be tuned, i.e., changed within limits, by the operator through the On-Line menu of the Executive software. The value of tunable variables can also be changed by application tasks by using the BASIC language WRITE\_TUNE function. BASIC and Control Block tasks must define tunable variables with a variation of the BASIC LOCAL statement that includes the tuning parameters. Ladder Logic/PC tasks cannot use tunable variables.

The value of local variables at the time of initial task installation is always 0. The effect of a Stop All or a power failure on variable values in the rack depends on the variable type. Local tunable variable values in both AutoMax and UDC application tasks are always retained. Local variable values are retained for AutoMax tasks, but not for UDC tasks.

AutoMax Processors will retain the last values of all local variables. UDC modules will retain the variable values for the following: parameter configuration data, UDC test switch information, and D/A setup configuration. The variable values of the following input data will also be retained: feedback registers, UDC-PMI communication status registers, and UDC task error log information. UDC modules will NOT retain local variable values and data found in the following registers, which are considered outputs: command registers, application registers, the ISCR (interrupt status and control register), scans per interrupt register, and scans per interrupt counter register. See the AutoMax Programming Executive instruction manual for more information on the STOP ALL and system re-initialization conditions.

### 2. Common

Common variables are variables that are defined in the configuration for the rack and are therefore accessible to all application tasks in the rack. There are two types of common variables, those that refer to memory locations, and those that refer to actual physical I/O locations. The two types are defined differently in the configuration task for the rack.







## 3.0 LADDER LOGIC LANGUAGE DESCRIPTION

Ladder Logic language, also called PC language, permits the AutoMax distributed control system to perform sequential logic operations in real time using industry standard ladder diagrams. The ladder diagram structure is used to define the logic process and establish the sequences and types of operations to be performed.

Along with the typical normally open and closed contact and coil operations, AutoMax ladder logic also provides functions for software timers, counters, shift registers, event coils, and remark sequences, as well as functions for detecting transitions (rising or falling edge) on bits.

The multi-tasking capability of the AutoMax distributed control system allows it to run multiple ladder logic tasks, each with its own scan period. In cases where all sequences do not require evaluation at the same rate, you can create two or more tasks that execute at different scan intervals. Multi-tasking conserves processing capacity, since all ladder sequences need not be evaluated at a high rate to satisfy the high scan rate requirements of a few key ladder sequences.

An important feature of AutoMax ladder logic is its use of symbolic, or variable, names in place of physical addresses. This allows you to assign unique names up to 14 characters long to all internal points, input and output points, timers, counters, and shift registers. In addition, each element in a sequence can have a 40-character description. The real-time display allows you to view the description of any element in a sequence.

AutoMax Ladder Logic does not support arithmetic operations. These functions are available in either the BASIC or the Control Block languages. For additional information, see the AutoMax Enhanced BASIC Language Instruction Manual J-3675 and AutoMax Control Block Language Instruction Manual J-3676.

Note that Ladder Logic tasks can run on AutoMax Processors only. If you are using Universal Drive Controller modules for drive control, you cannot run Ladder Logic tasks on a UDC module. See J-3676 for more information.

Note that this instruction manual describes the ladder logic language only. For more information about the the editor used to create and edit ladder logic tasks in the AutoMax Executive Software, refer to the ReSource AutoMax Programming Software instruction manual.

### 3.1 Application Program Structures

A ladder logic task is a symbolic representation of a specific logic process. It consists of one or more ladder sequences. The total number of sequences allowed is limited only by the amount of memory available and the complexity of the sequences. Ladder logic operations are described in chapter 4.

A standard ladder logic sequence consists of a 6-row by 9-column matrix of elements, plus a coil in the tenth column. Software timers, counters, and shift registers consist of a 6-by-6 matrix of elements with the coil in the tenth column. Sequences are numbered between 1 and 32767.

As in relay logic, series operations represent logical AND operations and parallel branches represent logical OR operations. Figure 3.1 shows the basic sequence structure as it would appear on the screen.

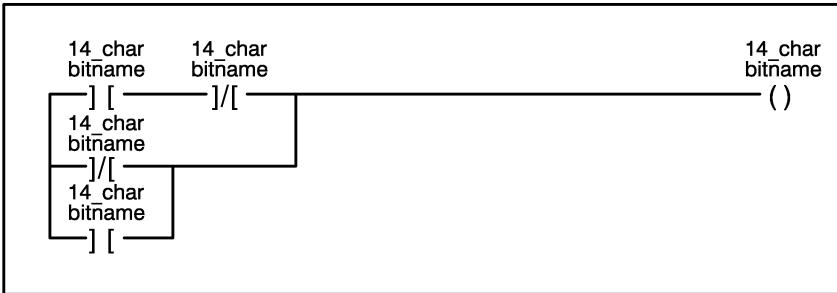


Figure 3.1 - Ladder Sequence Structure

The AutoMax Executive software will include the following in the printout of a ladder program listing. Note that you can also print out a cross reference of the program. See J-3684 for more information about cross reference.

1. All sequences as entered
2. The 40-character description for each element (as four lines of 10 characters)
3. Cross-reference data for each coil beneath the coil symbol and description
4. The type of variable (common in uppercase type and local in lowercase type) used for each contact
5. The number of the sequence that defines each variable.

See Figure 3.2 for a sample printout of one sequence.

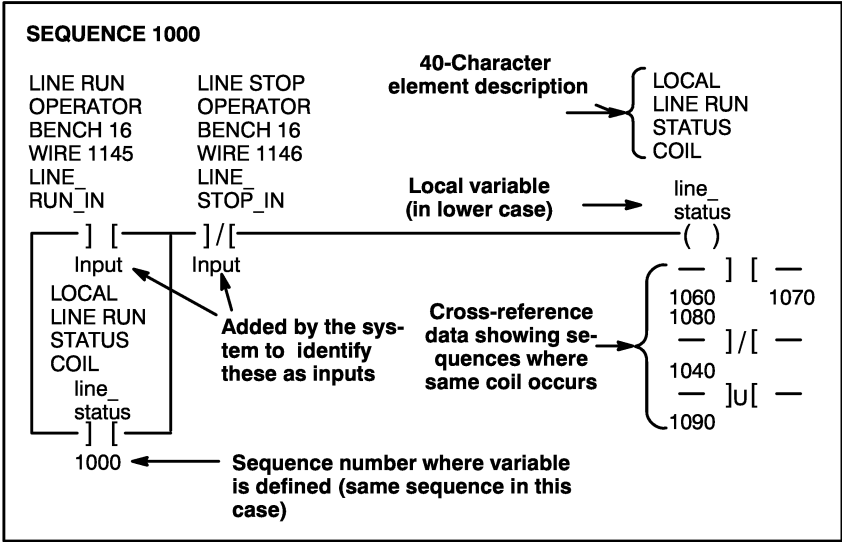


Figure 3.2 - Sample Printout of Task Listing

### 3.2 Variable Names in Ladder Logic

All ladder logic operations must be assigned a symbolic, or variable, name. The type of variable used depends on the type of operation selected. Contacts and coils must be boolean (bit) variables, and all other operations must be represented by integer variables. Names must begin with a letter and can contain up to 14 alphanumeric characters which are entered as two lines of seven characters. Typing the underscore “\_” on the first line inserts an underscore and moves the cursor to the second line, allowing you to divide the variable name into recognizable segments. The parentheses, “(” and “)”, are used to specify arrays. See the examples below for valid variable names:

OUTPUT_	OUTPUT_
ARRAY	(10)

### 3.3 Variable Types in Ladder Logic

Each symbolic, or variable, name must be defined as one of two types: local to the task or common to all Processors in a rack. The AutoMax editor will prompt you for this information.

You should define a variable as local if it does not represent a physical I/O point and it is not referenced by any other application task. Local variables are typically used as temporary storage for coils or for TIMER or COUNTER preset names which are not referenced outside of the task.

You should define a variable as common if it is a physical I/O point or it is referenced by another application task within the system. A contact name which has not also been used as a coil name is assumed to be common. All variables which are defined as common

must also be defined in the configuration for the rack in which the ladder logic task will run. If you are using AutoMax Version 2.1 or earlier, see the Configuration Task Instruction Manual (J-3649) for more information on the configuration task.

By specifying a subscript with an array name, you can use boolean array elements as contacts and coils. Arrays can be one-dimensional only. The variable name is still limited to 14 characters, including the parentheses for enclosing the subscript value. When an array name is specified, it is assumed to be common, and it must therefore be defined in the configuration task for the rack in which the ladder logic task will run. See the Enhanced BASIC Language Instruction Manual, J-3675, for more information on arrays. See figure 3.3.

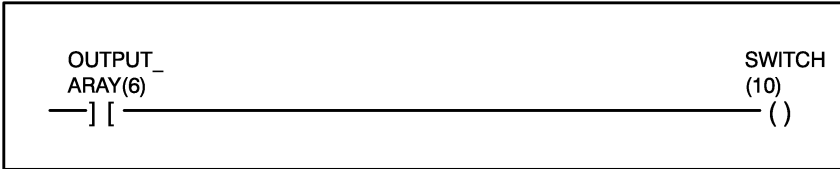


Figure 3.3 - Arrays Used As Contact and Coil

# 4.0 LADDER LOGIC LANGUAGE OPERATIONS

You will probably recognize some operations, like the normally open contact and coil, as standard ladder diagram operations commonly used in programmable controller applications. Other operations supported by AutoMax ladder logic, such as the upward transition, are typically handled with multiple operations in one or more additional sequences in most programmable controllers.

## 4.1 Contacts

AutoMax ladder logic supports four input operations specifically designed to operate on bits of data in the ladder logic data buffer. Each bit is associated with one of the following: digital input/output; common memory; local memory; or the output of a counter, timer, shift register, or event coil.

### 4.1.1 Normally Open Contact

The normally open contact operation reads the state of the specified bit variable and, depending on the bit value, generates a logically true or logically false condition.

If the bit is ON (1), the logical sense of the operation is TRUE. If the bit is OFF (0), the logical sense is FALSE.

This operation can be located anywhere within the ladder diagram except in the tenth element position (the right-most in the display). This position is reserved for output operations. See figure 4.1.

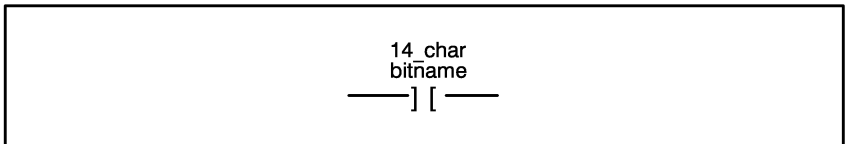


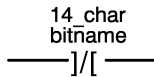
Figure 4.1 - Normally Open Contact

### 4.1.2 Normally Closed Contact

The normally closed contact operation reads the state of the specified bit variable and, depending on the bit value, generates a logically true or logically false condition.

If the bit is OFF (0), the logical sense of the operation is TRUE. If the bit is ON (1), the logical sense of the operation is FALSE.

This operation can be located anywhere within the ladder diagram except in the tenth element position, which is reserved for output operations. See figure 4.2.



14\_char  
bitname  
———] / [———

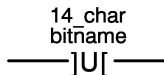
Figure 4.2 - Normally Closed Contact

### 4.1.3 Upward Transition Contact

The upward transition contact operation causes the ladder logic to determine if the bit has changed from the OFF to the ON state during the last scan. Depending upon whether a transition occurs, the ladder logic generates a logically true or logically false condition.

If the bit was turned ON (a 0 to 1 transition), the logical sense of the operation is TRUE. If the bit is OFF (0) or still ON (1), i.e., there has been no transition since the last scan, the logical sense of the operation is FALSE.

This operation can be located anywhere within the ladder diagram except in the tenth element position, which is reserved for output operations.



14\_char  
bitname  
———] U [———

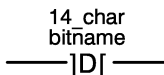
Figure 4.3 - Upward Transition Contact

### 4.1.4 Downward Transition Contact

The downward transition contact operation causes the ladder logic to determine if the bit has changed from the ON to OFF state during the last scan. Depending upon whether a transition occurs, the ladder logic generates a logically true or logically false condition.

If the bit was turned OFF (a 1 to 0 transition), the logical sense of the operation is TRUE. If the bit is ON (1) or still OFF (0), i.e., there has been no transition since the last scan, the logical sense of the operation is FALSE.

This operation can be located anywhere within the ladder diagram except in the tenth element position, which is reserved for output operations. See figure 4.4.



14\_char  
bitname  
———] D [———

Figure 4.4 - Downward Transition Contact



## 4.2 Output (Coil) Operations

Ladder logic language supports the following six output, or coil, operations: COIL, ON-DELAY TIMER, OFF-DELAY TIMER, COUNTER, SHIFT REGISTER, and SET EVENT. A variable name can only be used once as a coil in a ladder logic task. See chapter 6 for details concerning the initial memory states for these functions.

### 4.2.1 Coil

The coil operation writes the state of the rung to the specified bit variable. If the rung is logically true (ON), a 1 is stored at the specified bit. If the rung is logically false (OFF), a 0 is stored at the specified bit.

This operation must be located in the tenth element position, which is reserved for output operations. Only one coil per sequence is permitted. See figure 4.5.

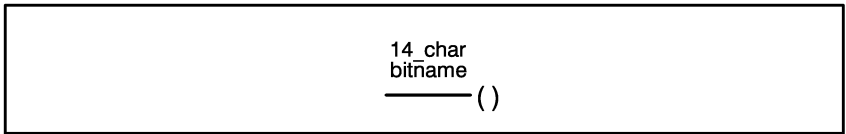


Figure 4.5 - Coil Operation

### 4.2.2 Timer

Ladder logic supports two types of software timers: On-Delay and Off-Delay. Each timer consists of enabling logic in any combination of contacts, lines, and spaces in a 6 row by 6 column matrix with the timer coil in column 10. Only one coil per sequence is permitted.

Software timers require three additional parameters: Preset Name, Preset Value, and Current Value Name. The Preset Name (PN) is the name of the variable that contains the preset value. This variable can be LOCAL to the task or COMMON to the system. If PN was defined as COMMON, it can later be modified under software control by a BASIC or Control Block program.

The Preset Value (PV) is the number of 0.1 second increments of time that is initially stored in PN. The PV is limited to 32767 intervals of 0.1 second. In the event of a STOP ALL or power cycle, PV will be loaded and Current Value Name will be set to 0 unless Current Value Name has been defined as a non-volatile COMMON memory variable.

The Current Value Name (CN) is the name of the variable that contains the current value of the timer. This value is the elapsed value, which is determined by the enable logic state (on or off). The variable can be LOCAL to the task or COMMON to the system. If Current Value Name is defined as a non-volatile COMMON memory variable, its value will be retained through a STOP\_ALL or power cycle. Refer to Figure 4.6.

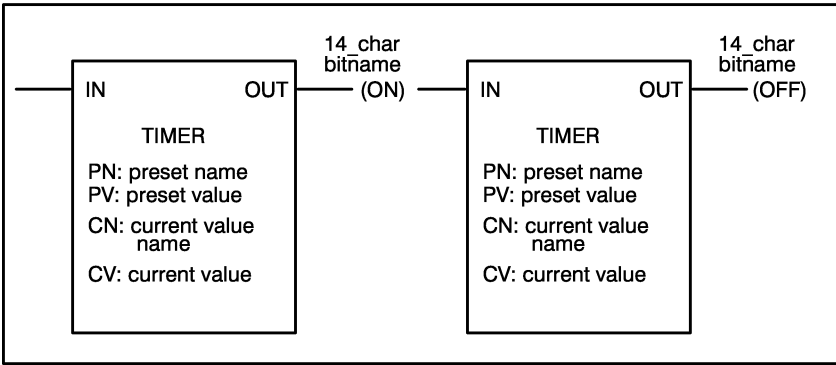


Figure 4.6 - Timer Output Blocks

The On-Delay timer provides a logically true output whenever the input is true and the preset time interval has elapsed. Whenever the input is false, the elapsed count is initialized to zero. When the input is true, the elapsed time is incremented at 0.1 second intervals. When the elapsed count reaches the preset value and the input is true, the output is also true. See figure 4.7.

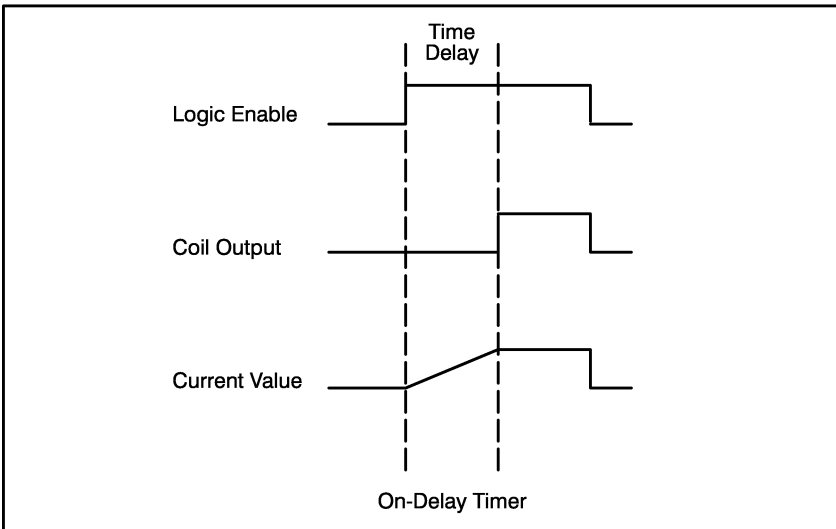


Figure 4.7 - Operation of On-Delay Timer

The Off-Delay timer provides a logically false output whenever the input is false and the preset time interval has elapsed. Whenever the input is true, the elapsed count is initialized with the preset count. When the input is false, the elapsed count is decremented towards zero at 0.1 second intervals. Whenever the elapsed count is zero and the input is false, the output is also false. See figure 4.8.

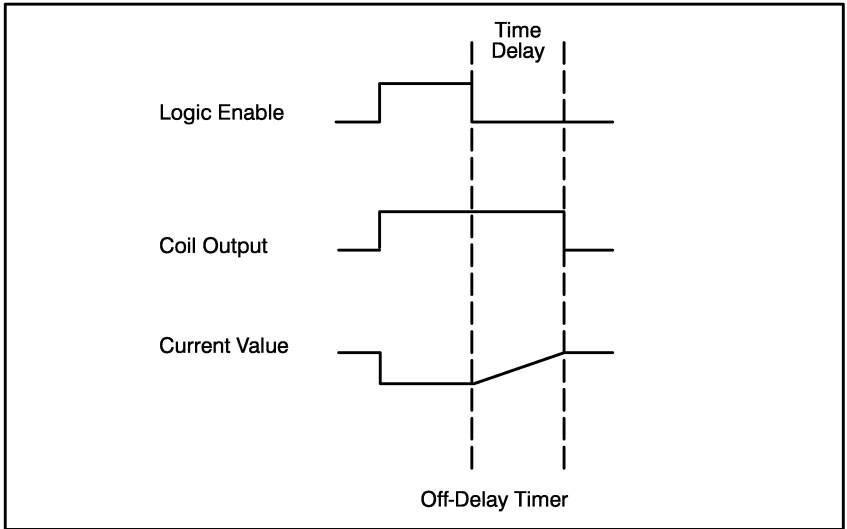


Figure 4.8 - Operation of Off-Delay Timer

### 4.2.3 Counter

The software counter is an up-down counter capable of counting over the range of +/- 32767. The counter consists of a counter block and the logic that drives it. The logic can be any combination of contacts, lines, and spaces in a 6-row by 6-column matrix. The counter block contains three inputs (UP, DOWN, and RESET) that must always terminate in the first, third, and fifth rungs in the ladder diagram.

Counters require three additional parameters: Preset Name, Preset Value, and Current Value Name. The Preset Name (PN) is the name of the variable that contains the desired count value. This variable can be local to the task or common to the system. If PN was defined as common, it can be modified under software control by the BASIC or control block program.

The Preset Value (PV) is the count value that is initially stored in PN. In the event of a STOP ALL or power cycle, PV will be loaded and Current Value Name will be set to 0 unless Current Value Name has been defined as a non-volatile COMMON memory variable.

The Current Value Name (CN) is the name of the variable that contains the current elapsed value of the counter. This variable can be local to the task or common to the system. If Current Value Name is defined as a non-volatile COMMON memory variable, its value is retained through a STOP ALL or power cycle. The elapsed value is determined by the state (on or off) of the UP, DOWN, and RESET logic state. See Figure 4.9.

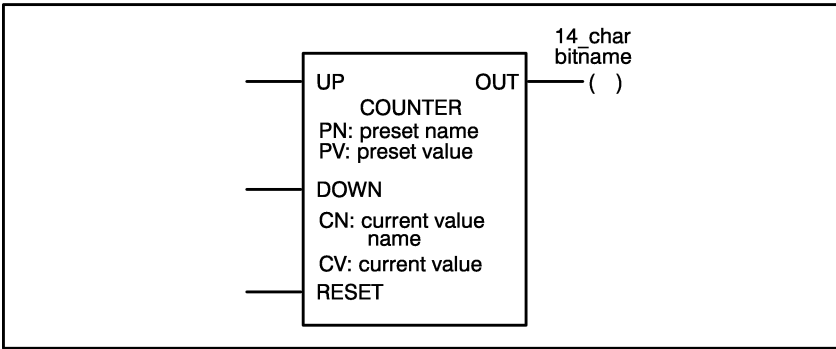


Figure 4.9 - Counter

The counter provides a logically true output whenever the elapsed counts are greater than or equal to the preset count. Whenever a positive transition (input changes from 0 to 1) is received at the UP or DOWN input, the internal elapsed count will be incremented or decremented accordingly. If a transition is received at both inputs during the same scan, no change will take place.

Whenever the elapsed count is greater than or equal to the preset count, the output coil will be true. The elapsed count will change in response to inputs up to a maximum of +/- 32767, regardless of the value of the preset count. Whenever RESET is high, the elapsed count will be reset to zero. The RESET input overrides all other inputs.

#### 4.2.4 Shift Register

The software shift register has a variable length up to 16 bits and shifts to the right. The shift register consists of a shift register block and the logic that drives it. The logic can be any combination of contacts, lines, and spaces in a 6-row by 6-column matrix. The shift register block contains three inputs (DATA, SHIFT, and RESET) that must always terminate in the first, third, and fifth rungs in the ladder diagram.

Shift registers require two additional parameters: Shift Register Length and Current Value Name.

The Shift Register Length (LEN) is the length ( $\leq 16$  bits) of the shift register. This parameter must be entered as an integer constant rather than a variable name.

The Current Value Name (CN) contains the name of the variable that contains the current elapsed value of the shift register. In the event of a STOP ALL or power cycle, Current Value Name will be set to 0 unless it has been defined as a non-volatile COMMON memory variable. In this case, it will retain its current value. This variable can be local to the task or common to the system. The elapsed value is determined by the state (on or off) of the DATA, SHIFT, and RESET logic. See Figure 4.10.

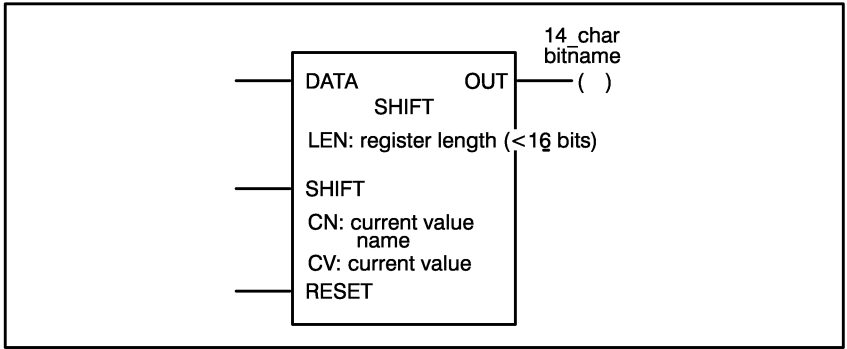


Figure 4.10 - Shift Register

Whenever a positive transition occurs (change from 0 to 1) at the SHIFT input, the data present on the DATA input will be shifted into the left-most bit in the shift register. In addition, all of the bits in the shift register will shift one position to the right. The state of the output coil is always equal to the state of the right-most bit in the shift register. The RESET input will reset the entire shift register. The RESET input overrides all other inputs.

#### 4.2.5 Event Coil

A software Event Coil provides synchronization with BASIC and Control Block tasks in the same rack. The software Event consists of a coil and the logic that drives it. The logic that drives it can be any combination of contacts, lines and spaces in a 6-row by 6-column matrix. The Event Coil is valid only in column 10, and only one Event Coil per statement is permitted.

The software Event requires one additional parameter: the Event Name. The Event Name (EN) is the name of the software event. The Event is set by an upward transition of the SET input. This results in a logically TRUE output. See figure 4.11.

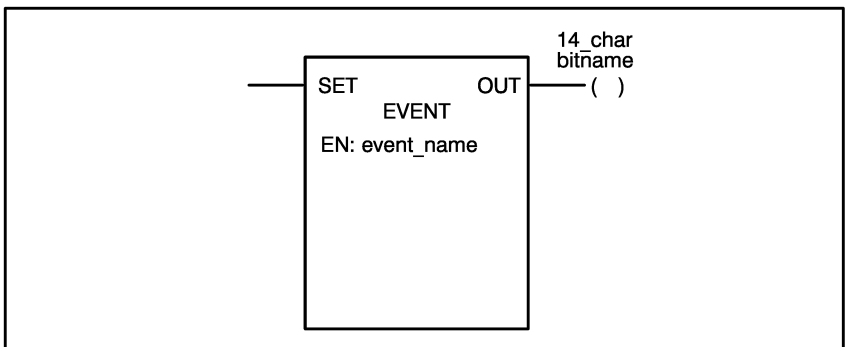



Figure 4.11 - Event Coil

When using the software event you must insure that another task responds to any EVENTS when they are set. Otherwise, once the system counts the maximum of 32767 events, a STOP ALL/CLEAR will occur.

### 4.3 REMARK Sequence

Ladder Logic supports a REMARK sequence to provide additional documentation within the ladder program. Up to 16 lines of text may be entered for each REMARK sequence.

A REMARK sequence consists of a sequence number and a REMARK keyname. The keyname may contain up to 14 characters and must be unique within the task. Refer to Figure 4.12.



REMARK: Keyname

Figure 4.12 - REMARK Sequence

The text associated with the keyname is not contained in the ladder logic program itself but resides in a separate file that you create with the AutoMax text editor. Remarks are stored in the file as follows:

! keyname

They can be followed by 0-16 lines of ASCII text. Refer to the ReSource AutoMax Programming Instruction Manual (J-3684 or J-3750) for more information on creating REMARK sequences.

## 5.0 TASK EXECUTION

When a ladder logic task is turned ON, the initial state of all volatile coil variables will be false (OFF or 0) and no transitions are detected as a result of turning the task ON. Common coil names, which are defined in the configuration task for the chassis as non-volatile, or NVMEMDEF variables, are left unchanged.

In order to be detected, a transition must occur while the task is running. This also applies to transitions which might result from a variable being forced. Consequently, if a variable is forced to a particular state while the task is turned off, a transition will not be detected when the task is then turned on. If, on the other hand, a variable is forced while the task is running and the forced state causes a transition, this transition will be detected by the task.

Transition contacts that reference external inputs will detect the transition in the next scan that runs after the transition occurs. Transition contacts that reference a coil in the task will detect the transition in the same scan in which the coil changed state if they are located in sequence after the coil. Otherwise they will detect the transition in the next scan.

### **WARNING**

**DEPENDING ON THE APPLICATION, TURNING ON A TASK MAY CAUSE OUTPUTS TO CHANGE STATE. THIS MAY CAUSE UNEXPECTED MACHINE MOVEMENT. FAILURE TO OBSERVE THIS PRECAUTION COULD RESULT IN BODILY INJURY.**

### 5.1 Buffering

Each ladder logic task will read all of its COMMON variables and save them in a local buffer before each scan begins. This insures that inputs will remain in a single state for the duration of the scan. After each sequence is evaluated, its coil is updated in the local buffer. Subsequent references to that coil will obtain the current state. At the end of each scan, the outputs are written from the local buffer to their actual physical location.





# 6.0 STOPPING A TASK CONTAINING COILS WITH INTERNAL MEMORY

Timers, counters, and shift registers all have internal memory associated with them. The memory will be in one of two possible states after a ladder logic task has been stopped.

## 6.1 Task Stopped with STOP Command

If the task was stopped by the operator through a STOP command, the internal memory for timers, counters, or shift registers is left unchanged from its state at the time the task was stopped. Therefore, if a timer was timing at the time the task was stopped, it will continue from where it left off when the task is restarted. Likewise, the value of any counter at the time the task was stopped is retained and counting continues from that value. The value of any shift register is also retained, and shifting continues with that accumulated data. Note that the Past state for inputs that are transition detected is left in the state the inputs were in when the task was stopped.

## 6.2 Task Stopped Through Power Loss or with STOP ALL Command

If all tasks were stopped either through power loss or with a STOP ALL command, the internal memory for timers, counters, or shift registers is usually set to the same state it was in when the task was initially downloaded to the Processor module. See below for specific details on each operation.

### ON-DELAY TIMER

Current value is set to the RESET state, i.e., the timer is ready to begin timing when the INPUT becomes TRUE, unless Current Value Name has been defined as a non-volatile COMMON memory variable. In this case, the current value is retained.

### OFF-DELAY TIMER

Current value is set to the “timed out” state, i.e., the timer has already timed and expired unless the Current Value Name has been defined as a non-volatile COMMON memory variable. In this case, the current value is retained. A TRUE state on the INPUT is required to RESET the timer, after which a FALSE state on the INPUT will cause the timer to begin timing.

### COUNTER

Current value is set to zero unless the Current Value Name has been defined as a non-volatile COMMON memory variable. In this case, the current value is retained. The past states of the UP and DOWN inputs are set FALSE.



## **SHIFT REGISTER**

Current value is set to zero unless the Current Value Name has been defined as a non-volatile COMMON memory variable. In this case, the current value is retained. The past state of the SHIFT input is set FALSE.

## **EVENT COIL**

The past state of the input is set FALSE to recognize the first rising edge.

# 7.0 EXECUTION TIME AND MEMORY USAGE ESTIMATES

The execution time for a ladder logic task can be estimated using the following timing estimates. After the total execution time is computed, divide by the scan time to obtain an estimate of the CPU usage for the task. A maximum of 2047 different symbols may be used in a single task.

Execution Time and Memory Usage Estimates

Ladder Logic Operation	6010 Execution Time in $\mu$ sec.	7010 Execution Time in $\mu$ sec.	Memory Usage in Bytes
Normally Open Contact	1.5	0.5	6
Normally Closed Contact	1.5	0.5	6
Transition Contacts			
AND	3.0	1.0	10
OR	3.5	1.2	12
Coils	26.0	7.1	8
Timers	50.0	14.7	24
Counters	50.0	14.0	24
Shift Registers	41.0	11.9	24
Events			
Event Set	272.0	152.0	
Event Not Set	18.0	9.6	24
Remark	11.0	3.5	24
Variables			8 + # of characters in name
System Overhead - fixed	314	100	3000
System Overhead - variable	$18c + 9f$	$6c + 3f$	$c/n * 10 + 10$ per sequence
<p><b>Key:</b></p> <p>c = number of common variables  f = number of forced variables  n = packing density, a function of how the common booleans are stored in memory. The value can range from 1 (worst case) to 16 (ideal case). Use a value of 8 as an estimate.</p>			



# Appendix A

## Converting a DCS 5000 Ladder Logic Task to the Current Version of the AutoMax Executive Software

You can easily convert any version 4 DCS 5000 Ladder Logic application task to run in an AutoMax processor. The only substantial difference between DCS 5000 and AutoMax Ladder Logic that is visible to the user is the Current Value Name (CN) parameter added in On-Delay and Off-Delay Timers, the Counter, and the Shift Register.

Programs that have been converted to AutoMax cannot be converted back to DCS 5000 format unless they are re-entered in their entirety.

The Ladder Logic/PC Editor in AutoMax will automatically convert your DCS 5000 task to AutoMax when you use the Editor to open the task, and then save the task again. You will see the message "Converting DCS 5000 to AutoMax" flash briefly on the screen when you open such a task. The Editor will simply add the Current Value parameter to the On-Delay, Off-Delay, Counter, and Shift Register operations in the task. It will assign variable names to Current Value Names in one of two methods. Method 1 is used when converting a DCS 5000 Ladder Logic task to AutoMax Version 1. Method 2 is used when converting a DCS 5000 Ladder Logic task to AutoMax Version 2.0 or AutoMax Version 3.0.

### Method 1

1. The Editor will search the task for any of the following variable names (asterisks denote any character):

```
CVNAME*  
CV*NAME*  
CR*VAL*  
**CVN**  
***CVN*
```

2. The first variable not found in the task will be used as the base for all Current Value variable names the Editor adds.
3. The Editor will append a number, beginning with 1 and increasing by 1 each time, after the underscore found at the end of the variable (note that all variables listed end with an underscore).

When you open the task again, the Current Value Names will have been added. If you want any Current Value Name to represent a common (as opposed to a local), you must revise your configuration task for the rack in which the task will run to include these variables.

### Method 2

AutoMax Version 1.0 B and later uses the following method to assign variable names to Current Value Names.

1. The Editor will create the Current Value Name by prefixing the Coil name for the element with "CV\_".











## **For additional information**

1 Allen-Bradley Drive

Mayfield Heights, Ohio 44124 USA

Tel: (800) 241-2886 or (440) 646-3599

[www.rockwellautomation.com](http://www.rockwellautomation.com)

---

### **Power, Control and Information Solutions Headquarters**

Americas: Rockwell Automation, 1201 South Second Street, Milwaukee, WI 53204-2496 USA, Tel: (1) 414.382.2000, Fax: (1) 414.382.4444

Europe/Middle East/Africa: Rockwell Automation, Vorstlaan/Boulevard du Souverain 36, 1170 Brussels, Belgium, Tel: (32) 2 663 0600, Fax: (32) 2 663 0640

Asia Pacific: Rockwell Automation, Level 14, Core F, Cyberport 3, 100 Cyberport Road, Hong Kong, Tel: (852) 2887 4788, Fax: (852) 2508 1846