

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. Control Block language is typically used for programming process control loops. It consists of BASIC statements and Control Block function calls. Refer to J-3675, J-3677, and J2-3094 for more information about Enhanced BASIC and Ladder Logic/PC 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 define system-wide variables within the AutoMax Programming Executive. This eliminates the requirement to write a configuration task for the rack. See the AutoMax Programming Executive (J-3750) for information about configuring variables.

The information that follows is applicable 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 AutoMax Application Tasks

AutoMax Processors allow real-time concurrent operation of multiple programs, or application tasks, on the same Processor. The tasks are executed on a priority basis and share all system data.

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 Universal Drive Control Application Tasks

Universal Drive Control (UDC) Control Block tasks are used exclusively for drive control applications. Each UDC module in an AutoMax rack can run up to 2 independent Control Block tasks (one for Drive A and another for Drive B) that provide speed loop control. UDC Control Block tasks share all system data. These tasks are not assigned a priority.

Each task operates on its own variables. The same variable names may be used in different tasks, but each variable is recognized only 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.

UDC Control Block tasks can use most, but not all, of the Control Block function calls in the AutoMax Control Block language. See Appendix C for a list of the Control Block functions that are allowed in UDC Control Block tasks. Also, note that the descriptions of Control Block functions in this manual indicate whether a particular function can be used in UDC Control Block tasks.

UDC Control Block tasks can be up to 20 Kbytes in size. These tasks are run at a fixed tick rate of .5 milliseconds. The maximum scan time allowed (set by using the SCAN LOOP function) is 20 ticks (10 milliseconds). Note that if a UDC module will be running two tasks, both must be assigned the same scan time. See section 53.0 for information regarding how to estimate execution time for a UDC Control Block task.

2.4 AutoMax Programming Conventions

This section describes programming conventions that apply to all Configuration, BASIC, Control Block, and Ladder Logic/PC tasks.

2.4.1 Naming

All task names are limited to 8 characters. The initial character must always be a letter. Only letters (A-Z), underscores (_), and numbers (0-9) are permitted. Spaces and other characters are not permitted in task names. The file extension is used to identify the task. Extension .CNF identifies configuration tasks (used only in Programming Executive Version 2.1 and earlier). .BAS is used for BASIC tasks. AutoMax Control Block tasks use extension .BLK. UDC Control Block tasks also use extension .BLK. PC/Ladder Logic tasks have a .PC extension.

Variable names in BASIC, AutoMax Control Block, and UDC Control Block tasks are limited to 16 characters. Variable names in Ladder Logic/PC tasks are limited to 14 characters (16 characters in V4.0 and later). The initial character of variable names must always be a letter or an underscore. Only letters (A-Z), numbers (0-9) and the

underscore (`_`) are permitted within the variable name. Spaces and other characters are not permitted in variable names. Terminating characters used to specify the variable type (see section 2.4.3) are not included in the size limits.

2.4.2 Constants

A constant, also known as a literal, is a fixed value that is not associated with a variable name. Listed below are the five types of constants that can be used in AutoMax, along with their size limitations:

1. Integer Constants (Integers and Double Integers)

Single integer value range is +32767 to -32768 with no fractional part. Double integer value range is +2147483647 to -2147483648 with no fractional part.

2. Hexadecimal Constants

Hexadecimal constants are integers in base 16, or "hex" format. The allowable range of hexadecimal constants is 0 to 0FFFFFFFFH. Hexadecimal constants are not sign-extended when they are stored. Leading zeroes are used to fill in any of the hex digits not specified. This means that numbers must be entered as 2's complement signed numbers. For example, if you enter 0F371H, it will be stored as 0000F371H, not as 0FFFFFF371H.

A hexadecimal number has three parts:

`{0}NNNNNNNNH`

where: 0 = required only when the first digit of the hexadecimal number is an alpha character (A-F) NNNNNNNN = the hexadecimal number H = specifies that the number is in hexadecimal format; always required

3. Real Constants

Real constants are decimal values. The value can be in the following ranges:

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

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

`{sign}{digits}{.}{E}{sign}{digits}`
For example: -1234.5678E+11

Use scientific notation to enter large numbers. Use double asterisks to indicate exponentiation.

4. String Constants

String constants are sequences of alphanumeric and other printable characters. Line terminators (<CR>) are not allowed. String constants must be enclosed either in single or double quotes. If one type of quotes is used in the sequence itself, the other type must be used to enclose the sequence. String constants may be up to 132 characters long.

5. Boolean Constants

There are four boolean constants: TRUE, ON, FALSE, and OFF.

2.4.3 Variables

A variable is a named location that represents a value or a physical I/O location. A variable may be either a simple variable or a subscripted (array) element. Depending upon the operations specified in the application task, the value of a variable may change from line to line. BASIC tasks use the most recently assigned value of a variable when performing calculations. Control Block and Ladder Logic/PC tasks latch the value of common double integer, integer, and boolean variables at the beginning of the task scan to ensure that external inputs will not change state during a scan.

The data type of a variable determines the type of information stored in that variable. For variables that contain numeric information, the data type also determines the range of values that may be stored in the variable. Values that are not in the allowable range will cause an error when the task is compiled or when it is put into run, depending upon the type of error. The data type of a variable is specified with a terminating character for four of the five types of variables. The fifth type, real variables, do not have a terminating character.

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:

9.2233717×10^{18} > positive value >
 5.4210107×10^{18}
 $(-20) - 9.2233717 \times 10^{18}$ > negative value >
 $-2.7105054 \times 10^{18}$ (-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}{.}{E}{sign}{digits}
For example: -1234.5678E+11

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@. Note that inverted booleans cannot be assigned to outputs from control blocks.

5. String Variables

Used to store any alphanumeric sequence of printable characters, including spaces, tabs, and special characters. The terminating character is \$.

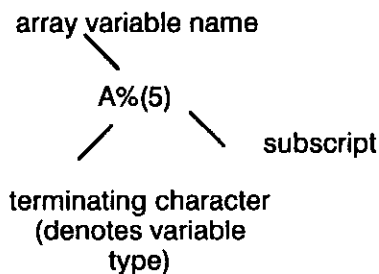
The sequence in a string variable cannot include a line terminator (<CR>). When defined, the sequence must be enclosed either in single or double quotes. If one type of quotes is used in the sequence itself, the other type must be used to enclose the sequence.

Version 1.0 Executive software allowed a fixed maximum length of 31 characters for string variables. Version 2.0 and later allow string variables of variable length, from 1 to 255 characters. To specify the maximum size of a string variable, add a colon and a number (1-255) immediately after the \$ character. For example, defining A\$:50 as a local variable in an application task will reserve space for 50 characters. Note that if no length is specified, the default length is 31.

2.4.4 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 65535. 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 (or bracket), the subscript(s), and a right parenthesis (or bracket). 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.

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.

2.4.5 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 not be changed by application tasks. 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 is 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 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.

Common memory variables can be of any data type. They may be read to or written from. Common I/O variables are long integer, integer, or boolean variables that represent actual physical I/O locations. Common I/O variables that represent inputs may be read but not written to. I/O variables that represent outputs may be read or written to.

