

Appendix A

Using Variables

All operations performed in AutoMax programs use symbolic names (variables) to represent storage locations for inputs, outputs, and other data. You use these variables to reference ladder instruction input and output parameters.

With the Editor you name variables just as you do in the Variable Configurator. Variable names can be a maximum of 16 characters. The valid characters are A–Z, a–z, 0–9, and _ (underscore). Names must begin with a letter.

The type of variable you choose depends upon the type of operation to be performed. Variables are either:

- simple stores a single piece of data
The data can be Boolean, integer, or double integer.

 - array stores a collection of data of the same data type
The data can be Boolean, integer, or double integer.

 - data structure stores both Boolean and double integer data
Data structures are used for Timer and Counter data types.
You can use pre-defined keywords to help you specify the individual timer and counter elements.
-

Variables also have these properties:

- Variables store data according to the data type they are defined as. The Editor assigns a default data type to variables you enter and provides a way for you to change the type, so you need not enter the variable type indicators @, %, and !. For more information, see section A.1.
- Variables have a scope, either global or local. The case of the first letter of the variable name indicates the scope of the variable. A variable name beginning with an upper-case letter is interpreted by the Editor as a global variable, while a variable name beginning with a lower-case letter is interpreted by the Editor as a local variable. For more information, see section A.3.

You cannot have two variables of different data types or scopes with the same name.

To configure global variables, you must define them in the Variable Configurator.

- Variables must have a name and can have a description.

An additional characteristic of simple and array variables is the option for you to access (index into) the data within these variables down to the bit level. For example, within a simple integer variable you can access each bit with a variable or with a constant. For more information, see section A.2.

A.1 Data Types

This section describes Boolean, integer, double integer, timer, and counter variables in more detail.

A.1.1 Boolean Variables

A Boolean variable stores the status (either 1 or 0) of one bit. When a bit has a value of 1, it is said to be on and logically true. When a bit has a value of 0, it is said to be off and logically false.

You can reference an individual bit using any of these methods:

Method	Example
simple Boolean	switch1
bit-indexed integer or double integer	vat.15 vat.31 vat.fill
Boolean array element	panel[20]
bit-indexed integer or double integer array element	panel[rack2].15
timer and counter status bits	timer_name.Q counter_name.QD

See section A.2 for more information.

This table presents data about the number of bits for some configurations:

A rack with:	Contains a maximum of:
one 7010 Processor	16, 384 bits
Common Memory module and 6011 processor(s)	14,496 bits
one 6011 Processor	11,464 bits

These bits are used by ladder programs for simple Boolean variables and Boolean array variables. These bits can be used by a single program or split among multiple programs.

You define the size of an array by entering a value in Maximum Array Index located within the Variable Properties dialog box. For more information about arrays, see section A.4.

A.1.2 Integer and Double Integer Variables

Integer and double integers have the following data and value ranges:

	Data Range	Value Range
integer	16 bits	-32768 to 32767
double integer	32 bits	-2147483648 to 2147483647

You can reference an integer or double integer variable the following ways:

Method	Example
simple	vat
element of an array	panel[rack2] panel[10]
timer or counter	timer_name.Elapsed counter_name.Current

You can also use decimal and hexadecimal constants. See section A.2 for more information.

■ A.1.3 Timer Variables

A timer variable is a data structure that combines pre-defined elements into a single symbol.

This element:	Is a:
TPreset	double integer element that stores the value at which the timer expires (where 1 = 10ms)
Elapsed	double integer element that stores the amount of time that has elapsed (where 1 = 10 ms) Elapsed is initialized to 0 when a program first begins to run unless the timer is a global and declared non-volatile in the Variable Configurator.
Q	Boolean output that indicates that the timer timed out
T	Boolean output that indicates that the timer is timing

To reference any element of a timer variable in a ladder logic program

- Assign a name to the timer variable and then add the reserved element name as an index.

For example, to reference a timer's elapsed value, type this: *name*.Elapsed. And to reference the Q status bit, type this: *name*.Q.

Where *name* is the name of the timer data structure.

IMPORTANT:

- Timer elements cannot be forced.
- You must enter global timers into the Variable Configurator as five-element, double integer, non-volatile arrays. Example: TIMER1!(4).

■ A.1.4 Counter Variables

A counter variable is a data structure that combines pre-defined elements into a single symbol.

This element:	Is a:
Current	double integer element that contains the current count Current is initialized to 0 when a program first begins to run unless the counter is a global and declared non-volatile in the Variable Configurator.
CPreset	double integer element that contains the value that the counter should count to or from
QU	Boolean output that is true when the value in Current is greater than or equal to the preset value (CPreset)
QD	Boolean output that is true when the value in Current is less than or equal to zero

To reference any element within a counter variable in a ladder logic program

- Assign a name to the counter variable and then add the reserved element name as an index

For example, to reference a counter's Current element, type this: *name*.Current.

where *name* is the name of the counter data structure.

IMPORTANT

- Counter elements cannot be forced.
- You must enter global counters into the Variable Configurator as five-element, double integer, non-volatile arrays.
Example: COUNTER1!(4).

A.1.5 Labels

Labels are used only in JMP and LBL instructions. When you enter a new name for a JMP or LBL instruction, the Editor automatically defines it as a Label.

A.2 Accessing Data Within Variables Via Bit-Indexing and Element-Indexing

For the ladder instruction parameters, you can specify bits within integer and double integer variables and elements and bits within an array variable. You can refer to bits or elements by using constants or variables. For example, you could access bit 17 in the double integer variable *pump* by using either *pump.17*, or any bit within *pump* using *pump.fill*. Notice a period (.) was used to separate *pump* from the bit. The period (.) is the bit delimiter.

To specify an element name or number, enclose it in square brackets ([]). For example, if *pump* was an array and you want element number 17 within the array, you can access it by: *pump[17]*. To access any element within the array *pump*, you can use *pump[fill]*. The square brackets ([]) are the element delimiter.

This methodology is called indexing. Using bit-indexed and element-indexed variables can help reduce the number of unique names required for your application and help you manage data collection and manipulation. Bit-indexed variables can also help make referring to I/O points easier. Just assign a name for the register corresponding to the I/O module and reference each I/O point by appending the appropriate bit number. For example, if one I/O module is controlling all the switches for a conveyor belt, you can assign the name “conveyor_switch” to the I/O module and reference each I/O point by appending a bit’s name or constant, such as “conveyor_switch.2.”

However, use bit-indexing with caution because the program’s execution time increases with the complexity of the variable.

Only variables assigned to I/O modules can have names assigned to the whole integer and to each bit.

You can use bit-indexed variables on relay instructions. For specific information about the supported variable types for an instruction, see the section pertaining to that instruction.

Element and bit variable names can each be a maximum of 16 characters (A–Z, 0–9, and `_`), not including the delimiter (`.` or `[]`). Valid ranges for constants used to reference bits are 0–15 for integers and 0–31 for double integers.

IMPORTANT

You can force only simple variables. You cannot force element-indexed or bit-indexed variables. For example, you cannot force variables like: `vat.13`, `array_var[11]`, `array_var[index_name]`, `array_var[11].12` or `array_var[index_name].bit_name`.

A.3 Global and Local Variables (Scope)

A property of a variable is its scope. A variable's scope can be either local or global.

A.3.1 Local Variables

Local variables are those that can only be used in the program in which they are defined. No other programs can reference them. If you type in the first letter of a variable name using lower case, the default scope will be local. The names of local variables appear in lower case.

A.3.2 Global Variables

Global variables can be referenced by ladder, Control Block, or BASIC programs in a rack. These variables can refer to memory locations, physical I/O locations, or network locations. Global memory variables can be of any data type supported by the Editor. If you type in the first letter of a variable name using upper case, the default scope will be global.

Global variables used for physical I/O must be simple variables. They can be either Boolean, integer, or double integer.

Global I/O variables representing this:	Can be:
inputs	read but not written to
outputs	read or written to

Within BASIC or Control Block programs, use the statement COMMON to access global variables.

A.4 Arrays

Arrays in ladder programs can have only one dimension. An item within an array is called an element.

This type of array:	Can have a maximum of:
integer and double integer	65,536 elements
Boolean	16,000 elements

The maximum size of a local array or global array when no Common Memory module is present is limited by the amount of available application memory. If a Common Memory module is present, a global array is limited by the amount of memory available on the module.

The first element within the array is always 0. Therefore, if an array had 65,536 elements, they would be numbered 0–65,535. You define the size of an array by entering a value in Maximum Array Index located within the Variable Properties tab.

The maximum array index is the number that represents the last element within an array variable. When you enter a new array variable in an instruction parameter, the maximum array index is automatically defined as the element number you entered as part of the variable. Should you need a larger array, change the Maximum Array Index field of the variable's property sheet. Remember array elements are numbered starting at 0.

To specify the maximum array index for an array variable using the Variable Properties

- Step 1. Select the instruction that uses the array variable.
- Step 2. Access the variable properties by doing either of the following:
 - From the File menu, choose Properties and then the Variables tab.
 - Press the right mouse button, and choose Properties from the pop-up menu and then the Variables tab.
- Step 3. Select the array variable name from the variable list.
- Step 4. In the Maximum Array Index field, enter the number of the last element. The largest number you can enter is 65,535 (65,535 = space reserved for 65,536 elements).
- Step 5. Click OK.

Tip

To increase the maximum array index, you can also enter the array variable with a greater index. For example, the variable $A[100]$ has 101 elements. If you need a larger index, you can enter $A[200]$. Variable A now has 100 more elements than before.

A.5 Constants

Use constants to specify:

- unchanging values within ladder instruction input parameters
- an element of an array (array[2])
- a bit within an integer or double integer variable (pump.15)
- a bit within an element of an integer or double integer array (array[2].31)

You can include the plus (+) and minus (-) signs for constants entered in the ladder instruction parameters.

IMPORTANT

To enter a hexadecimal value that begins with a letter as a ladder instruction input parameter, you must enter the value using a leading 0; end the value with an "h or H." For example, enter 0A5A5H not A5A5H.

A.6 About Initializing Variables

You can choose the value that a variable contains when a program is downloaded to the Processor or while the program is running. This is useful for loading a pre-defined value into a variable for calculations, modifying parameters, or changing a timer's or counter's preset value. You can change the value of a variable and/or its initial value by:

- setting or forcing the variable
- changing a preset value for a timer or counter via inline editing
- changing the initial value via the Variable Properties dialog box

You can choose from the following initialization methods for a variable from its Variable Properties tab:

- No Initialization (No Init.)
- User Specified Value
- Retained Value

A.6.1 About the No Initialization (No Init.) Method

When a variable is configured to use no initialization, the value it contains is determined by the buffering of inputs and outputs and by program execution. For global and local variables, the default initialization is No Initialization. Local and global variables defined as using no initialization are set to 0 when the program is downloaded to the Processor. Global outputs are not cleared at the start of the first scan.

This table explains the value of variables under specific conditions:

Ladder Program Condition	Variable Type		
	Non-volatile global	Volatile global	Local
Power fail	values retained, unless battery backup fails	values lost and set to zero	values retained
Stop-All			
Program stopped	values unchanged by operation; programs that are still running may cause values to change		

Ladder Program Condition	Variable Type		
	Non-volatile global	Volatile global	Local
Program stopped and individually restarted	values unchanged by operation; programs that are still running may cause values to change		values retained, unless the logic within the program writes to variables
Program stopped and the configuration individually reloaded	values are lost; values are zero when the configuration is installed		values retained
Program stopped and configuration and programs individually reloaded	values are lost; values are zero when the configuration is installed		values are lost; values are zero when the program is installed
Stop-All occurred and configuration and programs reloaded	values are lost; values are zero when the configuration is installed		

A.6.2 About the User Specified Initialization Method

When a variable is configured to use a User Specified value, the value defined as the initial value is written to the variable at the start of the first scan. If the variable's initial value is changed via inline editing, forcing, setting, or a program's execution, this new value is not retained when the program is stopped and rerun. The value written to the variable each time the program is placed into run is always that specified in the Initial Value field of the Variable's property sheet. Choose User Specified initialization when you want a variable to use the initial value you defined every time the program is placed into run.

IMPORTANT

Do not use user-specified initialization for a global I/O variable being used as an input. If you do, a bus error (error 31) occurs when you try to run the task.

A.6.3 About the Retained Value Initialization Method

When a variable is configured to use a Retained Value, the value defined as the initial value is written to the variable at the start of the first scan. If the variable's initial value is changed via inline editing, forcing, setting, or through the Variable Properties dialog box, the value the variable contains when program execution stops is the one that is written to the variable when the program is rerun. Retained Value initialization is the default initialization type for timers and counters. An initial value of a variable configured to use Retained Value initialization can be changed by any ladder program. For example, you can change a timer or counter preset value by using ladder logic. Initial values changed online can be permanently installed only by saving the program from the Processor. Choose Retained Value initialization when you want a variable to be initialized with the value it had at the time the program was stopped. Make sure that you specify global variables using Retained Value initialization as non-volatile variables.

IMPORTANT

Do not use retained-value initialization for global I/O variables. Doing so causes an error when the program is verified.

A.6.4 About Initializing Timer and Counter Variables

You can define the initialization mode for timer and counter preset variables. Choose between a User Specified Value or Retained Value. The Retained Value is the default initialization mode. You cannot define an initialization method for the Elapsed timer element or the Current counter element. These elements are usually reset to 0 at the start of program scan, unless the timer or counter data structure is defined as global and non-volatile. If you reference a timer or counter element in a program whose data structure is not used in a timer or counter instruction within the same program, the variable uses the No Initialization method. You cannot change it to User Specified Value or Retained Value. However, if you later add that timer or counter data structure on a newly added timer or counter instruction, the initialization type of the timer or counter elements becomes Retained Value. You can then choose either a Retained Value or User Specified Value initialization. The value contained in the Initial Value field is the timer's or counter's preset value.

A.6.5 About Initializing Arrays

You can define an initial value for individual elements of an array. The list of available elements and their current values appears in the Array Index list box.

A.6.6 Defining the Type of Initialization To Use for a Variable

You can define how a variable is initialized when a program is downloaded to the Processor and put into run. This is useful for loading a pre-defined value into a variable for calculations, modifying values for parameters, or changing a timer's or counter's preset value. Use the Variable Properties dialog box for the variable whose initialization method you want to change.

To define the type of initialization to use for a variable

- Step 1. In an offline program, select the instruction containing the variable whose initialization method you want to change.
- Step 2. From the File menu, choose Properties. The Instruction Properties dialog box is displayed.
- Step 3. Choose the Variable Properties tab, and select the variable whose initialization method you want to change.
- Step 4. Choose the variable initialization method you want to use. Choose from No Init. (No Initialization), User Specified Value, or Retained Value.
- Step 5. If you have chosen User Specified Value or Retained Value, define an initial value.
- Step 6. Accept the change by clicking OK or Apply.
- Step 7. Save the program, and reload it to the Processor.

Tip

You can change the initial value while editing the program offline or online.

A.6.7 Defining the Initial Value of a Variable

You can define the value that a variable contains when a program is downloaded to the Processor or placed into run. You can also define an initial value for individual elements of an array. The list of available elements and their current values appears in the variable's Variable Properties tab.

The initial value applies to User Specified Value or Retained Value initialization.

To define the initial value of a variable

- Step 1. In an offline or online program, select the instruction containing the variable whose initial value you want to change.
- Step 2. From the File menu, choose Properties. The Instruction Properties dialog box is displayed.
- Step 3. Choose the Variable Properties tab, and select the variable whose initial value you want to change.
- Step 4. Make sure that the selected initialization method is the one you want to use.

Step 5. Use the following table to determine your next step:

If the variable is:	Do the following:
not an element of an array	<ul style="list-style-type: none"><li data-bbox="641 215 992 309">• In the Initial Value field, enter a value that you want to use as the initial value. A value of 0 is the default value. <p data-bbox="641 317 992 434">In the case of a timer or counter data structure, the value in the Initial Value field is the timer's or counter's preset value. A preset value cannot be a negative number.</p>
an element of an array	<p data-bbox="641 447 992 591">Step A. In the Array Index list box, select the element whose initial value you want to change. The element's current value is displayed next to the array element.</p> <p data-bbox="641 599 992 695">Step B. In the Initial Value field, enter a value that you want to use as the initial value. Zero is the default value.</p>

Step 6. To accept the change, click OK or Apply.

When changing the initial value of a variable using Retained Value initialization during an online editing session, a dialog box prompts you to decide if the changed value should be sent to the rack. Choosing Yes immediately sends the change to the rack. The variable's value is immediately changed. When you save the program from the Processor, the newly changed initial value replaces the original value. Choosing No cancels the change.

Any change made to the value of a variable using User Specified Value initialization during an online editing session does not take effect until the program is stopped and re-run.

Tip

You can enter initial values in hexadecimal. To enter a hexadecimal value that begins with a letter, you must enter the value using a leading 0; end the value with an “ h or H.” For example, enter 0A5A5H, not A5A5H.