PLC**Logix**

User Guide

Table of Contents

Chapter 1: Welcome to PLCLogix

1-1 Introduction

PLCLogix is a Programmable Logic Controller (PLC) simulator that emulates the operation of a ControlLogix controller and RSLogix 5000 software. PLCLogix is an ideal tool for learning the fundamentals of ladder logic programming. It will allow you to practice and develop your programming skills using the industry-standard RSLogix 5000 PLC programming software. It provides users with the ability to write, edit and debug programs written using a tag-based format. RSLogix 5000 uses tags, which is a powerful method of programming PLCs but also more complex. PLCLogix provides an interactive approach to learning and understanding the operation of a sophisticated tag-based PLC in a realistic simulated manufacturing environment. Figure 1-1 shows an example of a PLCLogix circuit.

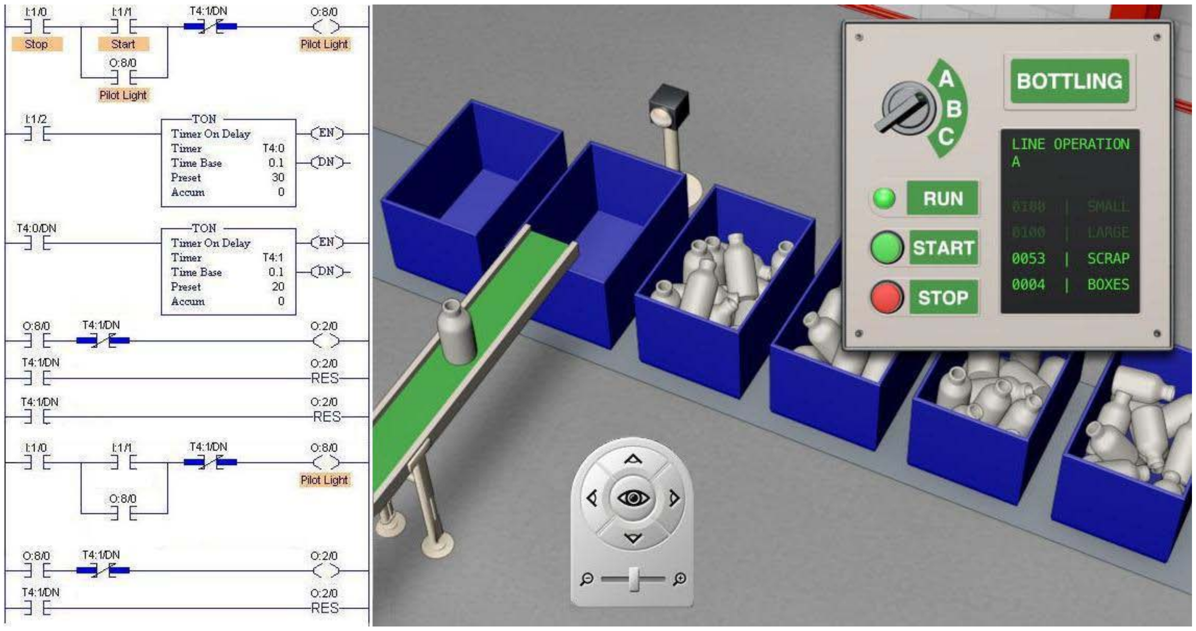


FIGURE 1-1 PLCLogix elements.

One of the main advantages of using PLCLogix is that it enables you to gain “hands on” experience in the operation of the Logix 5000 PLC. By using PLCLogix, you are able to gain much-needed programming practice by creating and running your own ladder logic programs using tag-based memory. The integration of the ladder programs with the 3D “worlds” provides a unique opportunity for programming in real-time and observing the operation of complex control devices and systems.

PLCLogix functionality includes a graphical controller organizer and a point-and-click method of configuring various I/O. The application organization is based on using tasks, programs, and routine structures. In addition, it features sophisticated data handling and incorporates both arrays and structures to provide maximum flexibility and emulation of real world control applications. PLCLogix also includes a free-form ladder editor that allows you to modify multiple rungs of logic at the same time. The point-and-click graphical interface provides a simple, intuitive method of entering and editing ladder logic programs.

The PLCLogix Graphic User Interface (GUI) displays the interactive animation as well as the ladder logic, controller organizer, I/O chassis, and a range of control panels. The ladder logic display is the same format as Logix 5000. The controller organizer also follows the same convention as Logix 5000 to provide a seamless transition from PLCLogix simulation to the real-world Logix 5000 control.

1-2 Graphic User Interface (GUI).

The Graphic User Interface for PLCLogix is designed to emulate RSLogix 5000, with the main difference being the addition of a virtual I/O chassis and a range of 3D simulation worlds. The purpose of the GUI is to provide a range of information displayed on a single screen. This information ranges from bit display to program code to status indicators. The Bit Status is represent by green horizontal bars on either side of the I/O device. The bar on the left is referred to as the rung-condition-in and the bar on the right is the rung-condition-out. When the green bars are illuminated, it indicates a high bit (1) is present in the I/O memory location. If the I/O point is not green, it means that a low bit (0) is present at that address.

Figure 1-2 shows the main components for the PLCLogix GUI. The Menu Bar provides access to a variety of Windows-based functions including Help, Search, and I/O Worlds. The Windows Commands Toolbar contains common Windows-based instructions such as New, Open, Save, Print, Cut, Copy, etc. The Ladder Instruction Toolbar provides category tabs which contain a wide range of instructions in various subsets, or categories. Instructions are inserted into ladder programs by clicking on instructions in the various categories.

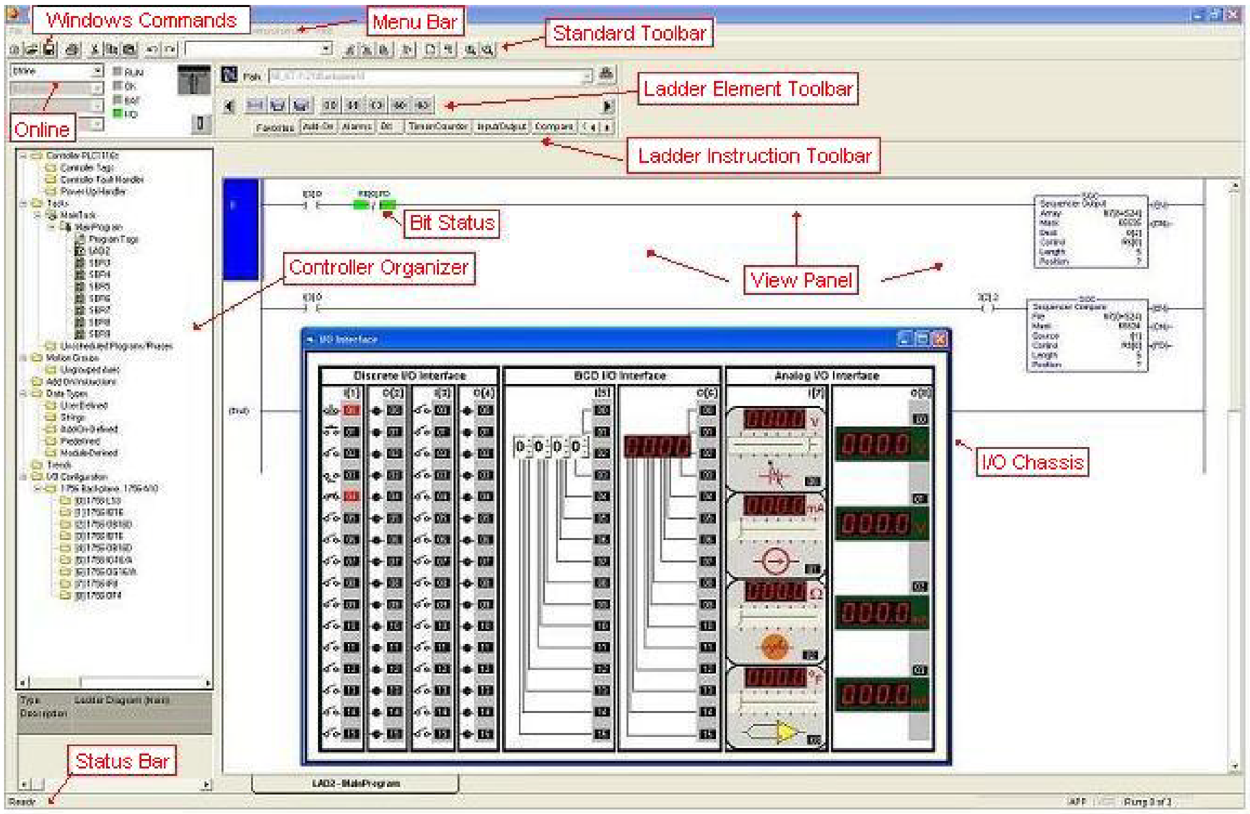


FIGURE 1-2 PLCLogix GUI.

The Online Bar places the PLC online (Run) or offline (Program). When the PLC is Offline, program edits can be made. When switched to Online, the program is downloaded, and the PLC is in the Run mode. The Standard Toolbar consists of typical edit functions, such as cut, copy, paste that are used repeatedly during the development and debugging of ladder logic programs. The Ladder Element Toolbar features both common editing functions as well as more specialized edit capabilities. The Controller Organizer is a graphical representation of the contents of your controller project. The Status Bar provides prompts during software operation as well as on-going status information updates. The View Panel is the window which contains the ladder logic diagram and various editors, such as Tag Editors.

1-3 Controller Organizer

The PLCLogix controller organizer is based on the same format as the RSLogix 5000 controller organizer, and displays tasks, programs, and structures in a tree-like format, similar to Windows Explorer. The controller organizer is a graphical representation of the contents of a project, and makes it easy to see all related information about programs, data and I/O configurations, and to navigate through programs, routines, and tasks. Figure 1-3 shows an example of the PLCLogix Controller Organizer. The main folders in the Controller Organizer tree are: Controller Project Name, Tasks, Motion Groups, Add-On Instructions, Data Types, Trends, and I/O Configuration.

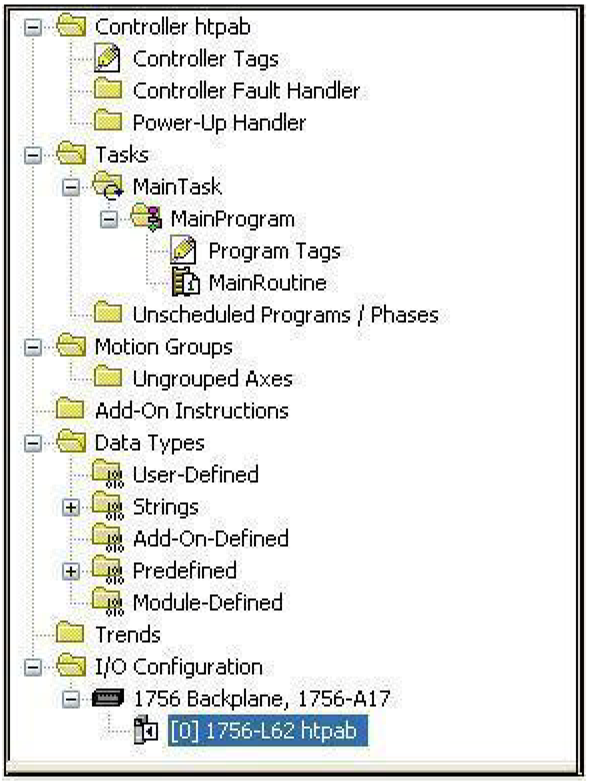
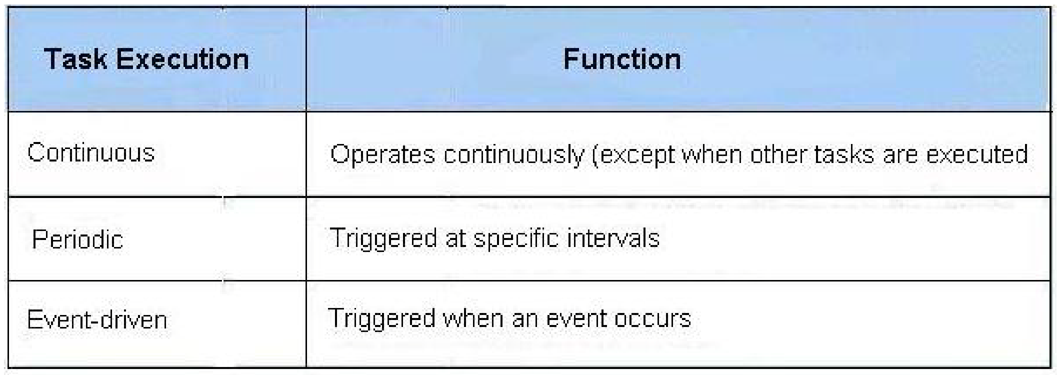


FIGURE 1-3 Controller Organizer

Earlier versions of PLC control software, such as the RSLogix 500, would have a main program and subprogram(s) that would control an application. RSLogix 5000 and PLCLogix have controller organizational models that allow for multiple applications, and each application is known as a *project*. A project holds all of the elements that are contained in an application, including tasks, programs, and routines.

Tasks are an important part of a controller organizer, and provide scheduling and priority information for one or more programs that are executed based on specific criteria. A task is basically a scheduling mechanism used for executing programs. A PLCLogix project can have multiple tasks, and each task can be triggered (executed) continuously, periodically, or when an event occurs. Figure 1-4 shows the various types of tasks that can be executed: Continuous, Periodic, and Event.



Task Execution	Function
Continuous	Operates continuously (except when other tasks are executed
Periodic	Triggered at specific intervals
Event-driven	Triggered when an event occurs

FIGURE 1-4 Task Execution Types

A Continuous Task is self-triggered and automatically repeats. It runs constantly in the background, and when it completes a full scan it immediately restarts. For most applications the continuous task will hold the PLC user-created program. Only one task can be executed at a time so continuous tasks will be executed whenever other tasks are not triggered. When a new project is initiated, a continuous task is created by default. A program does not require a continuous task, and there can be only one continuous task, regardless of number of tasks.

Periodic tasks operate at specific pre-determined intervals and contain program commands that need to be executed on a timed basis. A periodic task performs a function at a specific rate. The time period can be adjusted from 1 ms to 2000 s. Periodic tasks can be assigned a priority level with high priority tasks interrupting lower-priority tasks. Event-driven tasks will execute when a specified event takes place. Tasks in the Logix5000 controller are executed by priority. Continuous tasks have the lowest priority, which is fixed. Periodic and Event-driven tasks have adjustable priority levels. Event-driven tasks are generally used for Axis- and motion-control applications.

Tasks are divided into one or more programs, and each task can operate up to 100 of these programs. Once a task is executed, every program assigned to the task will be triggered in the order they are stored in the controller's memory. A program is basically a set of related tags and routines. Each program consists of tags, a main executable routine and other routines, such as a fault routine. A routine is a set of logic instructions written in a PLC language, such as ladder logic.

1-4 I/O Chassis

The PLCLogix I/O chassis shown in Figure 1-5 contains discrete input and output modules as well as BCD and Analog input and output devices. There are two 16-point discrete inputs and two 16-point discrete outputs. There are five different input switch types: NO switch, NC pusbutton, NO pushbutton, NO limit switch, and NC limit switch. The discrete output simulators use green LED lights to indicate when an output is ON or OFF. The BCD input and output modules are designed to simulate a range of decimal input and output values. The 4 BCD thumbwheels connected to the BCD input module allows for decimal values of 0000 to 9999 to be entered. The output value is displayed using four 7-segment displays to indicate a decimal range from 0000 to 9999.

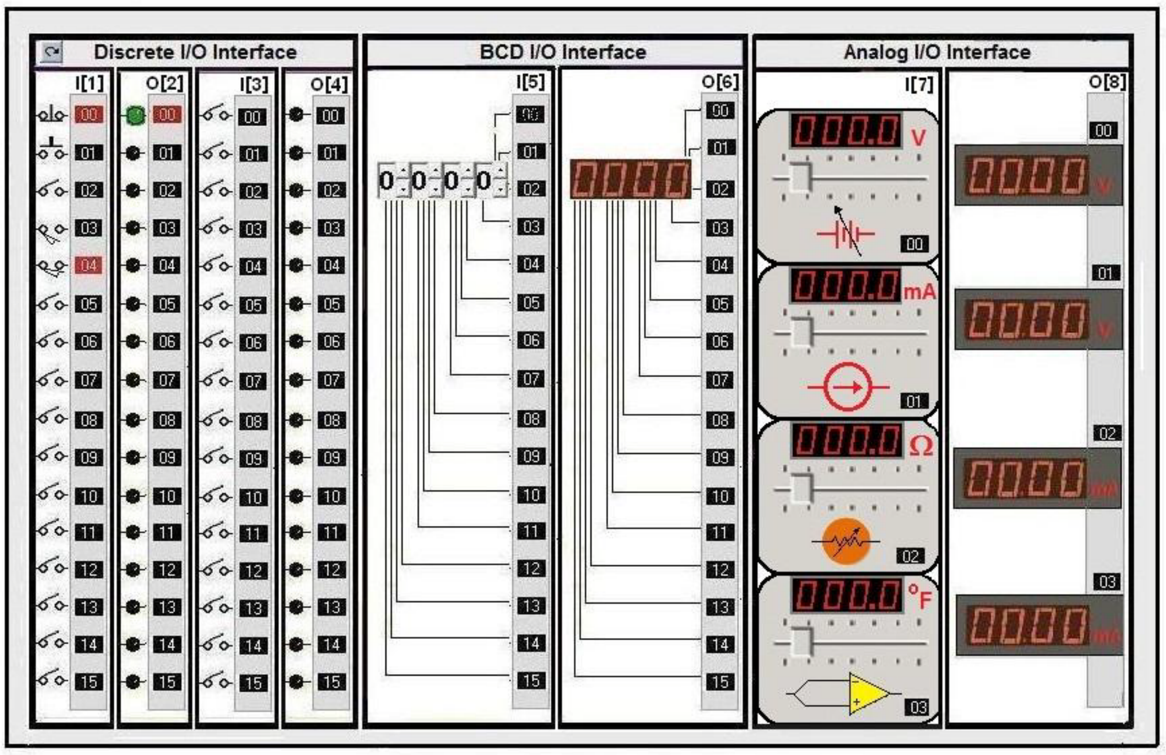
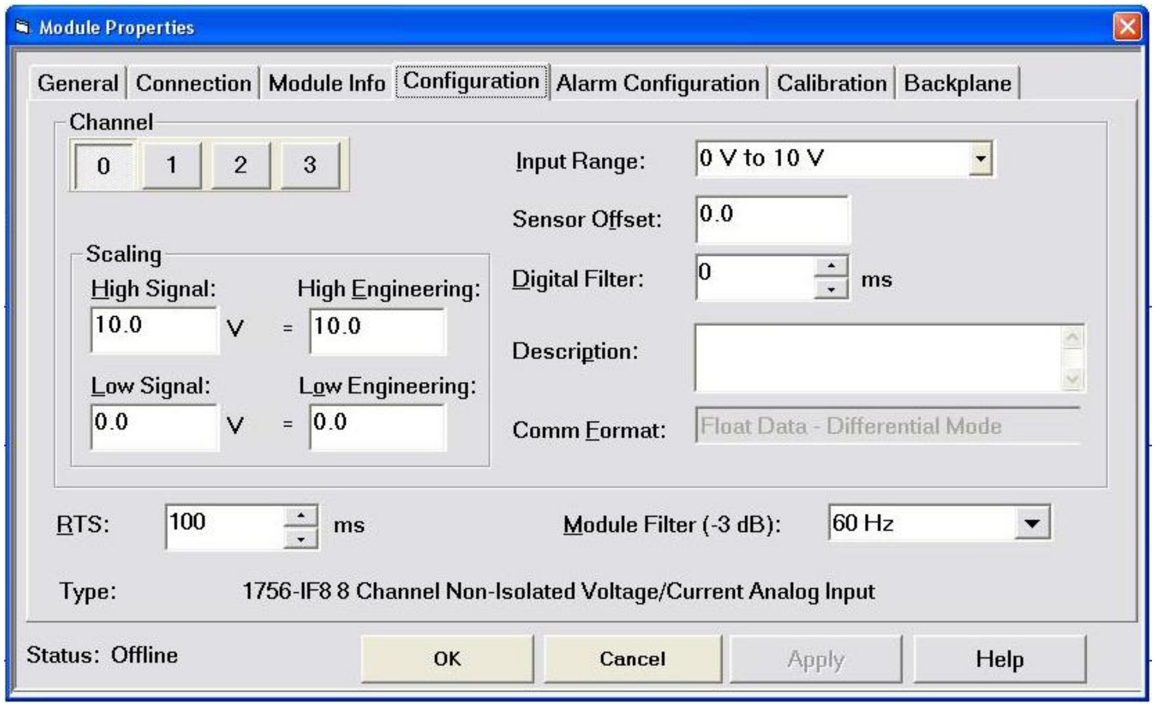


FIGURE 1-5 PLCLogix I/O Chassis

Each of the simulated discrete input devices are selectable among five different types of devices by right-clicking on the device with your mouse. The five settings available are normally closed (NC) pushbutton, normally open (NO) pushbutton, NO switch, NO limit switch, and NC limit switch.

The Analog I/O interface in the PLCLogix chassis provides a four-channel analog input and a four channel analog output. The analog input signals simulated are voltage, current, resistance, and temperature. Each of the four channels has a default setting of one of these signals. The channels can be further customized by changing the values in the parameter settings window. The analog input and output displays also include a decimal point and linear adjustment for setting precise analog input values ranging from 000.0 to 999.9. Figure 1-6 shows the various parameter settings for the analog I/O modules in the PLCLogix I/O Chassis.



Scaling
High Signal: 10.0 V = High Engineering: 10.0
Low Signal: 0.0 V = Low Engineering: 0.0

Input Range: 0 V to 10 V
Sensor Offset: 0.0
Digital Filter: 0 ms
Description:
Comm Format: Float Data - Differential Mode

RTS: 100 ms
Module Filter (-3 dB): 60 Hz

Type: 1756-IF8 8 Channel Non-Isolated Voltage/Current Analog Input

Status: Offline

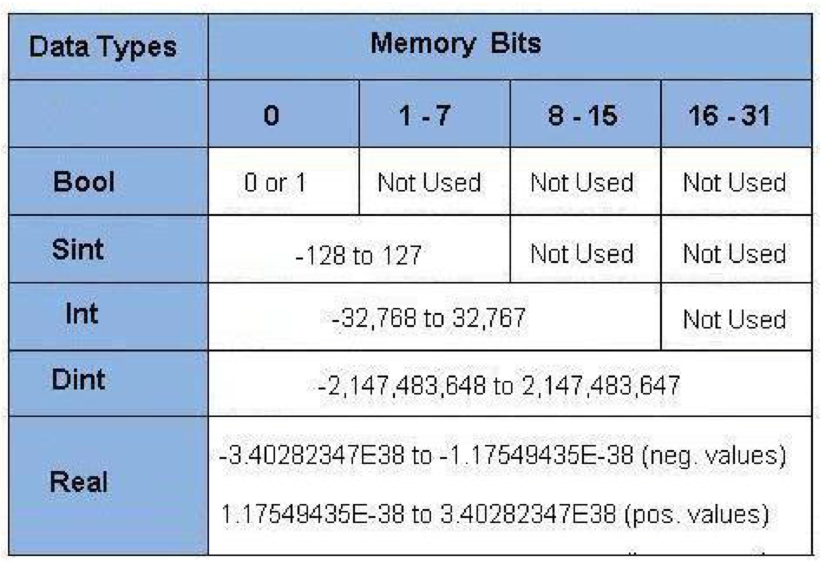
FIGURE 1-6 Analog I/O Module Properties.

Chapter 2: Tags and Aliases

2-1 Introduction to Tags

Logix 5000 and PLCLogix controllers define memory by using variable names, also known as *tags* and aliases. Tag-based memory structures are the newest type of memory addressing used by PLCs. A tag is simply another name for a memory location with an assigned data type. For example, Start_PB1 could be a tag name assigned to a start button, instead of something less user-friendly, such as I:1/05.

When a tag is created, it must be given a data type. A data type is a definition of the size and layout of memory allocated for the created tag. Data types define how many bits, bytes, or words of data that will be used by a tag. There are two main data types: basic and structured. Figure 2-1 shows the data types that are considered to be basic, or atomic. These include Boolean (BOOL), Short Integer (SINT), Integer (INT), Double Integer (DINT) and real numbers (REAL). RSLogix 5000 data structures include timers, counters, arrays, messages, and PID. PLCLogix has 32-bit memory locations, which means that a tag will always reserve 32 bits of memory, regardless of whether the data is Boolean or integer. The memory bits shown in Figure 2-1 indicate 32-bits for each memory type, even though some use less (e.g. Bool only uses one bit) only use one bit.

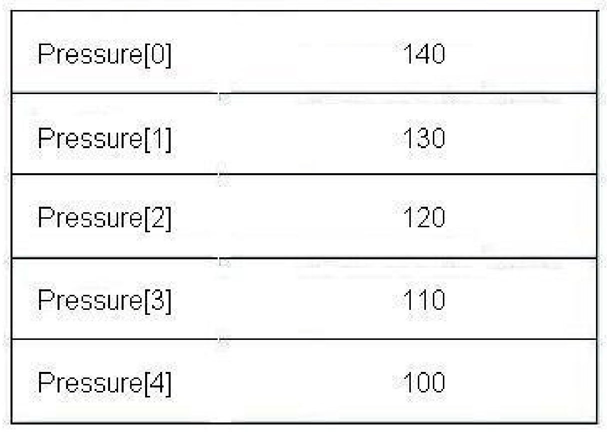


Data Types	Memory Bits			
	0	1 - 7	8 - 15	16 - 31
Bool	0 or 1	Not Used	Not Used	Not Used
Sint	-128 to 127		Not Used	Not Used
Int	-32,768 to 32,767			Not Used
Dint	-2,147,483,648 to 2,147,483,647			
Real	-3.40282347E38 to -1.17549435E-38 (neg. values) 1.17549435E-38 to 3.40282347E38 (pos. values)			

Figure 2-1 Data Types

A tag can also be defined as a compound set of the data types such as a structure or an *array***.** Unlike other PLCs, Logix 5000 and PLCLogix processors do not use indexed or direct addressing. Instead, they use arrays. An array is a type of tag that contains a block of data, and is similar to a data table. Arrays are numerically sequenced tags of the same data type that occupy a contiguous memory location.

An array is basically a table of tags, and is capable of holding the values of multiple tags. It is, essentially, a type of tag that consists of a block of multiple pieces of data. Each individual piece of data in the array is called an *element*. Each element in an array must be of the same type. An array tag holds each element in its assigned order in a contiguous block of the controller's memory. Arrays are useful for indexing applications, when the elements are required to be stepped through (indexed). Arrays can be created in 1, 2, or 3 dimensions. Figure 2-2 shows an example of a 1-dimensional array which holds 5 different values of pressure ranging from 100 to 140.



Pressure[0]	140
Pressure[1]	130
Pressure[2]	120
Pressure[3]	110
Pressure[4]	100

FIGURE 2-2 One-dimensional array.

PLCLogix also has a User Defined Data Table (UDT), or structure, which allows users to setup their own specific data structure for customized applications. Structures are capable of holding multiple types of data and include a description of each member. Structures enable you to assign both a name and description for each member within a user defined data type. The member name is used to access the associated data, and the member description helps to define the purpose of the member. By adding a description to a tag, it is possible to store another 120 characters of information.

Earlier Allen Bradley PLCs programmed with RSLogix 5 and RSLogix 500 software had memory locations where I/O and other internal values were stored, and these different data files could only hold one data type. These PLCs require very specific addressing to indicate I/O addresses, timers, counters, bits, variables, etc. Logix 5000 programming software has eliminated the use of data files and in its place is the tag database. The tag database organizes memory locations in one area, and each tag is assigned its own data type and radix.

2-2 Alias Tags

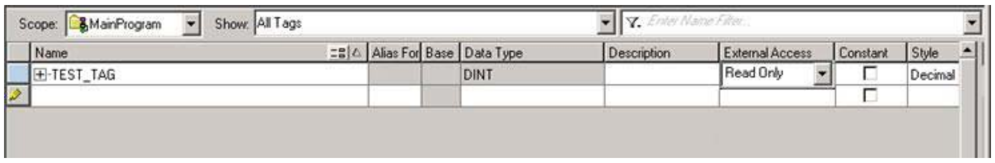
Another type of tag in Logix5000 controllers is an alias tag, which is an identifier for all or part of another tag in the application. Alias tags mirror the base tag to which they refer. In other words, an alias tag is a tag that represents another tag. When the base tag value changes, so does the alias tag. Alias tags allow a user to write a program with tag names assigned to physical I/O points. So, once the system design is complete all that is needed is the mapping of the appropriate tags. Unlike a base tag, an alias tag does not have a defined data-type because it simply assumes the data-type of the tag that it refers to. Aliases are commonly used to assign a descriptive name to an I/O device, or to simplify a name of a complex tag. During the download process, PLCLogix converts program references for alias tags to the physical memory used by the data that the aliases point to. The main benefit of the alias tag is that it allows you to create a new name for a piece of data.

A summary of the tags used in a given program is stored in a tag list. PLCLogix can access the tag list, online in the controller or offline in the software file, and make the tag list available to other software packages (e.g. Excel). With RSLogix 5000, once the software has been configured for the chosen controller, input and output variables are defined in the tag list to establish the link between hardware and software. In PLCLogix, the software is preconfigured and the controller link is already established. The data editors in both systems allow for the creation of a tag by assigning a tag name and defining the data type. The minimum memory allocation for a tag is 4 bytes, or 32 bits. An additional 40 bytes are required for each tag name. Since PLCLogix and ControlLogix are 32-bit controllers, a tag always reserves 32 bits of memory even if it is a Boolean or integer data type.

Each tag is stored individually in the processor, which allows for new tags to be created while the controller is on-line and in the Run mode. Because tags are independent of PLC I/O points in ControlLogix and PLCLogix, it is possible to write a program before any PLC I/O points have even been decided or configured. This allows the PLC programmer to create data within the controller that is structured to suit the needs of the application. Unlike older-model PLCs, where the user has access to a fixed set of either registers or bits, the ControlLogix and PLCLogix processors treat memory much more dynamically. Its versatility allows the user to choose their own names, data types, complex structures, and even arrays - much like how computer programmers might expect to treat memory.

<u>2-3 Creating Tags</u>

There are several methods available for creating tags – they can be created one at a time as you write the program, or they can be created in the tag editor shown in Figure -3. The tag editor contains a spreadsheet-like view of the tags where the tags can be created and edited. When an instruction is first used a “?” will indicated the need for a tag. There are three simple ways to create a tag using the “?” symbol. One method is to either right click or double click on the “?” and select an existing tag from the drop down box. Another method is to double click on the “?” and type in a tag name. If the name does not yet exist, right click on the tag name and select “Create New Tag Name”. A tag can also be created by clicking on the tag name and drag and drop an existing tag to a new instruction. Using any of these methods, PLCLogix will automatically assign the correct data type according to the instruction selected.

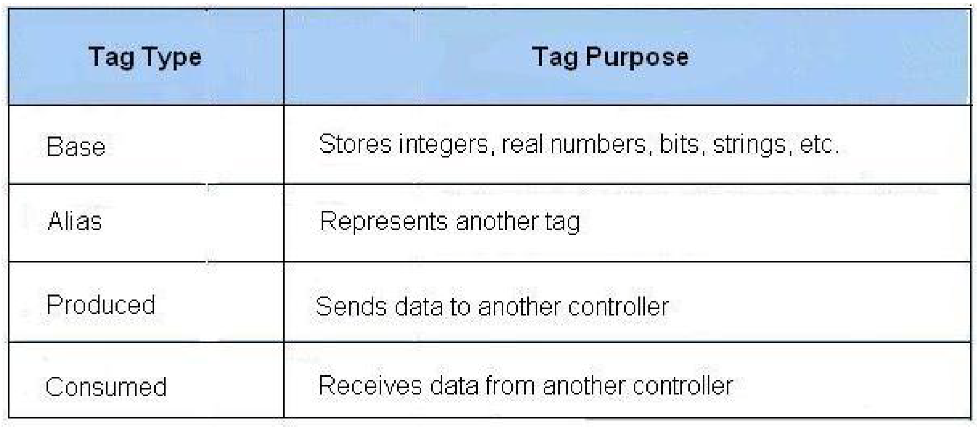


Scope: MainProgram Show: All Tags Enter Name Filter...

Name	Alias For	Base	Data Type	Description	External Access	Constant	Style
⊞-TEST_TAG			DINT		Read Only	☐	Decimal
						☐	

FIGURE 2-3 Tag editor.

The tag data type indicates the data format used by the tag, such as an integer, Bool, REAL, Timer, Counter, control, etc. Figure 2-4 shows the tag types used in PLCLogix and RSLogix Controllers. A Base tag is generally selected to hold data from logic-based operations using bits, integer numbers and real (floating point) numbers. Produced tags and Consumed tags are mainly used in the transfer of data between two or more controllers.



Tag Type	Tag Purpose
Base	Stores integers, real numbers, bits, strings, etc.
Alias	Represents another tag
Produced	Sends data to another controller
Consumed	Receives data from another controller

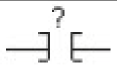
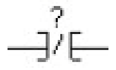
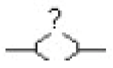
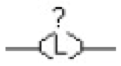
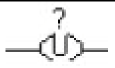
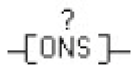
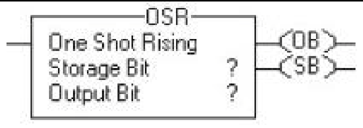
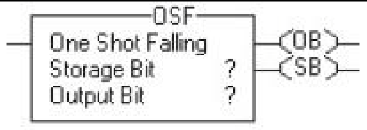
FIGURE 2-4 Tag Types.

Chapter 3: Instruction Set

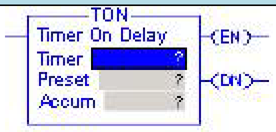
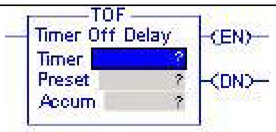
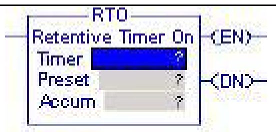
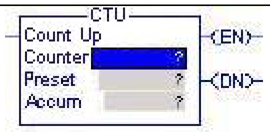
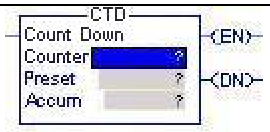
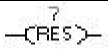
3-1 Introduction

PLCLogix features an extensive instruction set of over 70 commands. These instructions encompass all of the main ladder logic programming commands associated with Logix 5000. The PLCLogix Instruction Set consists of the following groups of commands: Bit Instructions, Timer and Counter Instructions, Program Control, Compare, Communications, Math, and Data Handling/Transfer instructions.

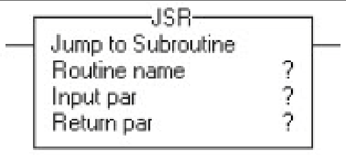
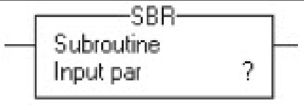
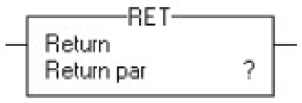
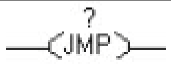
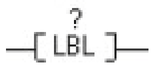
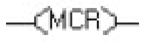
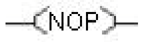
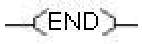
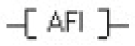
3-2 Bit Instructions

Instruction Mnemonic	Instruction Name	Symbol	Description
XIC	Examine If Closed		Examines a bit for an On (set, high) condition.
XIO	Examine If Open		Examines a bit for an Off (cleared, low) condition.
OTE	Output Energize		When rung conditions are true, the OTE will either set or clear the data bit.
OTL	Output Latch		When enabled, the instruction signals to the controller to turn on the addressed bit. The bit remains on, regardless of the rung condition.
OTU	Output Unlatch		When enabled, it clears (unlatches) the data bit. The bit remains Off, regardless of rung condition.
ONS	One Shot		Enable/disable outputs for one scan. Storage bit status determines whether this instruction enables or disables the rest of the rung.
OSR	One Shot Rising		A retentive input instruction that triggers an event to occur once. It either sets or clears the output bit, depending on the storage bit status.
OSF	One Shot Falling		This instruction either sets or clears the output bit, depending on the storage bit's status.

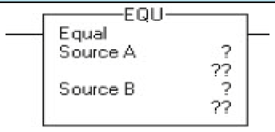
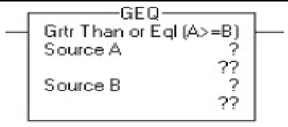
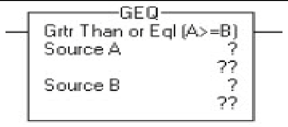
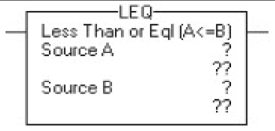
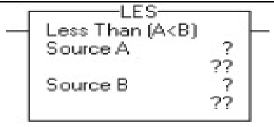
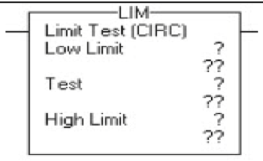
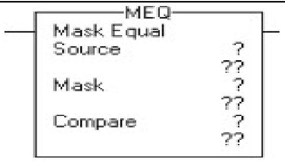
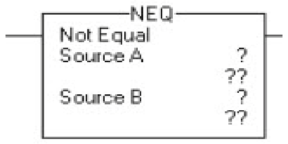
3-3 Timer and Counter Instructions

Instruction Mnemonic	Instruction Name	Symbol	Description
TON	Timer ON Delay	 TON Timer On Delay Timer ? Preset ? Accum ? EN DN	A non-retentive timer that accumulates time when the instruction is enabled. The accumulated value is reset when rung conditions go false.
TOF	Timer Off Delay	 TOF Timer Off Delay Timer ? Preset ? Accum ? EN DN	A non-retentive timer that accumulates time when the rung makes a true-to-false transition.
RTO	Retentive Timer On	 RTO Retentive Timer On Timer ? Preset ? Accum ? EN DN	A retentive timer that accumulates time when the instruction is enabled. Retains its accumulated value when rung conditions become false.
CTU	Count Up	 CTU Count Up Counter ? Preset ? Accum ? EN DN	An instruction that counts false-to-true rung transitions. It counts upward and the accumulated value is incremented by one count on each of these transitions.
CTD	Count Down	 CTD Count Down Counter ? Preset ? Accum ? EN DN	This instruction counts downward on each false-to-true rung transition. The accumulated value is decremented by one count on each of these transitions.
RES	Reset		This instruction is used to reset a timer, counter or control structure. The accumulated value of these instructions are cleared when the RES instruction is enabled.

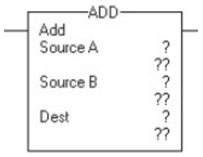
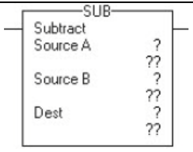
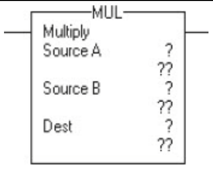
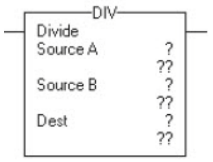
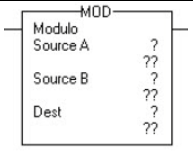
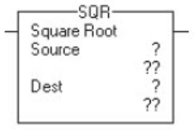
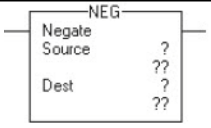
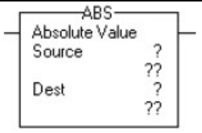
3-4 Program Control Instructions

Instruction Mnemonic	Instruction Name	Symbol	Description
JSR	Jump to Subroutine	 JSR Jump to Subroutine Routine name ? Input par ? Return par ?	This instruction jumps execution to a specific routine and initiates the execution of this routine, called a subroutine.
SBR	Subroutine		Stores recurring sections of program logic.
RET	Return		Used to return to the instruction following the a JSR operation.
JMP	Jump to Label		Skips sections of ladder logic.
LBL	Label		Target of the JMP instruction with the same label name.
MCR	Master Cont. Res.		Used in pairs to create a program zone that can disable all rungs between the MCR instructions.
NOP	No Operation		This instruction functions as a placeholder.
END	End		End rung in ladder logic circuit.
AFI	Always False Instruction		Sets the rung condition to False.

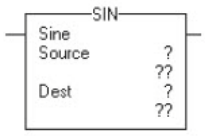
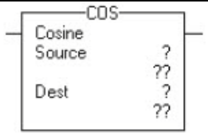
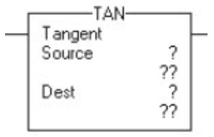
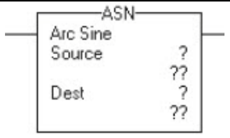
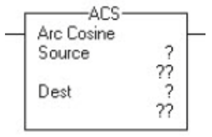
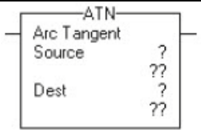
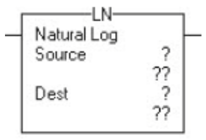
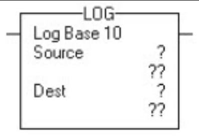
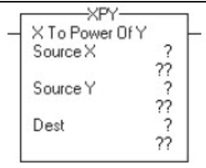
3-5 Compare Instructions

Instruction Mnemonic	Instruction Name	Symbol	Description
EQU	Equal		This instruction is used to test whether two values are equal. If Source A is equal to Source B, the instruction is logically true.
GEQ	Greater Than or Equal To	 GEQ Grtr Than or Eql (A>=B) Source A ? ?? Source B ? ??	Determines whether source A is greater than or equal to Source B. If the value at Source A is greater than or equal to the value at Source B, then the instruction is true.
GRT	Greater Than	 GEQ Grtr Than or Eql (A>=B) Source A ? ?? Source B ? ??	This instruction is used to test whether one value (Source A) is greater than another value (Source B).
LEQ	Less Than or Equal To	 LEQ Less Than or Eql (A<=B) Source A ? ?? Source B ? ??	Determines whether one value (Source A) is less than or equal to another (Source B).
LES	Less Than		This instruction determines whether Source A is less than Source B.
LIM	Limit	 LIM Limit Test (CIRC) Low Limit ? ?? Test ? ?? High Limit ? ??	This instruction is used to test for values within the range of the Low Limit to the High Limit.
MEQ	Mask Equal To	 MEQ Mask Equal Source ? ?? Mask ? ?? Compare ? ??	Passes the Source and Compare values through a Mask and compares the results.
NEQ	Not Equal To		This instruction tests whether Source A is not equal to Source B.

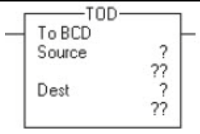
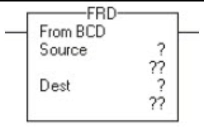
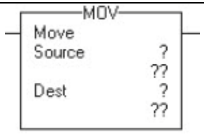
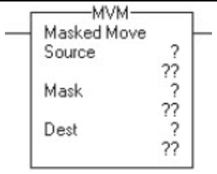
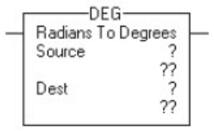
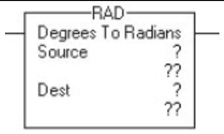
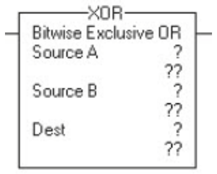
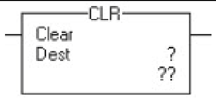
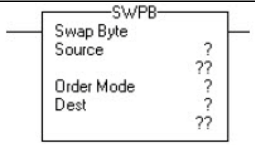
3-6 Math Instructions

Instruction Mnemonic	Instruction Name	Symbol	Description
ADD	Add	 ADD Add Source A ? ?? Source B ? ?? Dest ? ??	Adds Source A to Source B and stores the result in the Destination.
SUB	Subtract	 SUB Subtract Source A ? ?? Source B ? ?? Dest ? ??	Subtracts Source B from Source A and places the result in the Destination.
MUL	Multiply	 MUL Multiply Source A ? ?? Source B ? ?? Dest ? ??	Multiplies Source A by Source B and stores the result in the destination.
DIV	Divide	 DIV Divide Source A ? ?? Source B ? ?? Dest ? ??	Divides Source A by Source B and places the result in the Destination.
MOD	Modulo	 MOD Modulo Source A ? ?? Source B ? ?? Dest ? ??	Divides Source A by Source B and stores the remainder in the Destination.
SQR	Square Root		Calculates the square root of the source and places the float result in the Destination.
NEG	Negate		Changes the sign (+, -) of the Source and stores the result in the Destination.
ABS	Absolute Value		Takes the absolute value of the Source and places the result in the Destination.

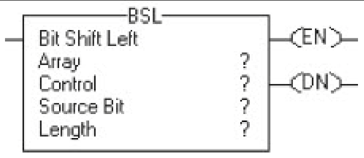
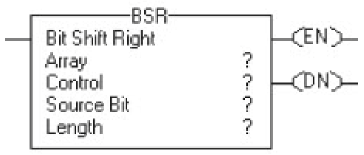
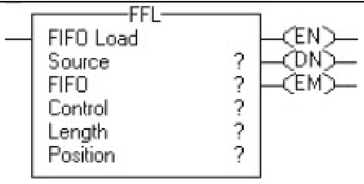
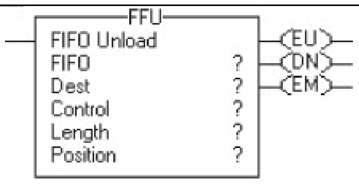
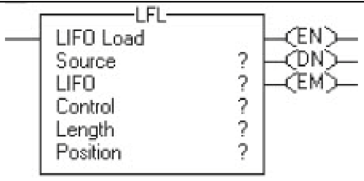
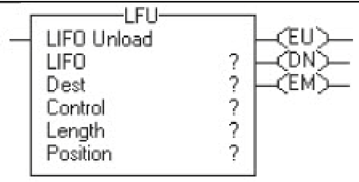
3-7 Advanced Math Instructions

Instruction Mnemonic	Instruction Name	Symbol	Description
SIN	Sine		Takes the sine of the Source value (in radians} and places the result in the Destination.
COS	Cosine		Takes the cosine of the Source value (in radians) and places the result in the Destination.
TAN	Tangent		Takes the tangent of the Source value (in radians) and stores the result in the Destination.
ASN	Arc Sine		Takes the arc sine of the Source value and places the result in the Destination (in radians).
ACS	Arc Cosine		Takes the arc cosine of the Source value and stores the result in the Destination (in radians).
ATN	Arc Tangent		Takes the arc tangent of the Source value and stores the result in the Destination (in radians).
LN	Natural Log		Takes the natural log of the Source value and stores the result in the Destination.
LOG	Log to the Base 10		Takes the log base 10 of the Source value and stores the result in the Destination.
XPY	X to the power of Y	 XPY X To Power Of Y Source X ? ?? Source Y ? ?? Dest ? ??	Takes Source A (X) to the power of Source B (Y) and stores the result in the Destination.

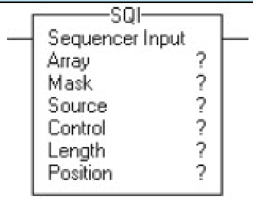
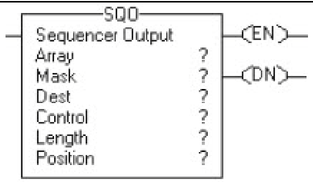
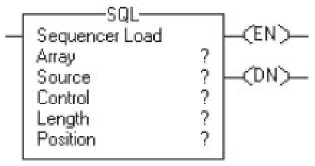
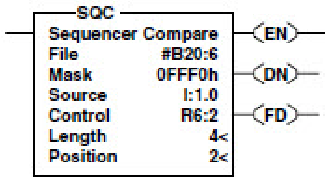
3-8 Data Handling/Transfer Instructions

Instruction Mnemonic	Instruction Name	Symbol	Description
TOD	Convert to BCD		This instruction converts a decimal value to a BCD value and stores the result in the Destination.
FRD	Convert to Integer		Converts a BCD value (Source) to a decimal value and stores the result in the Destination.
MOV	Move		Copies the Source (which remains unchanged) to the Destination.
MVM	Masked Move		Copies the Source to a Destination and allows segments of the data to be masked.
DEG	Degrees		Converts the Source (in radians) to degrees and places the result in the Destination.
RAD	Radians		Converts the Source (in degrees) to radians and stores the result in the Destination.
XOR	Bitwise Exclusive OR	 XOR Bitwise Exclusive OR Source A ? ?? Source B ? ?? Dest ? ??	Performs a bitwise XOR operation using the bits in Source A and Source B and stores the result in the Destination.
CLR	Clear		Clears all the bits of the Destination
SWPB	Swap Byte	 SWPB Swap Byte Source ? ?? Order Mode ? Dest ? ??	Rearranges the bytes stored in a tag.

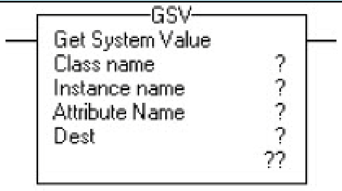
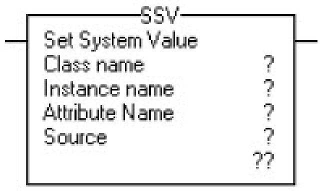
3-9 Array/Shift Instructions

Instruction Mnemonic	Instruction Name	Symbol	Description
BSL	Bit Shift Left	 BSL Bit Shift Left Array ? Control ? Source Bit ? Length ? EN DN	Shifts the specified bits within the Array (DINT) one position left.
BSR	Bit Shift Right	 BSR Bit Shift Right Array ? Control ? Source Bit ? Length ? EN DN	Shifts the specified bits within the Array one position right.
FFL	FIFO Load	 FFL FIFO Load Source ? FIFO ? Control ? Length ? Position ? EN DN EM	Copies the Source Value into a FIFO queue on successive false-to-true transitions.
FFU	FIFO Unload	 FFU FIFO Unload FIFO ? Dest ? Control ? Length ? Position ? EU DN EM	Unloads the Source value from the first position of the FIFO and stores that value in the Destination..
LFL	LIFO Load	 LFL LIFO Load Source ? LIFO ? Control ? Length ? Position ? EN DN EM	Copies the Source value to the LIFO.
LFU	LIFO Unload	 LFU LIFO Unload LIFO ? Dest ? Control ? Length ? Position ? EU DN EM	Unloads the value at .POS of the LIFO and stores 0 in that location.

3-10 Sequencer Instructions

Instruction Mnemonic	Instruction Name	Symbol	Description
SQI	Sequencer Input	 SQI Sequencer Input Array ? Mask ? Source ? Control ? Length ? Position ?	Detects when a step is complete in a sequence pair of SQO/SQI instructions.
SQO	Sequencer Output	 SQO Sequencer Output Array ? Mask ? Dest ? Control ? Length ? Position ? EN DN	Sets output conditions for the next step of sequence pair of SQO/SQI instructions.
SQL	Sequencer Load	 SQL Sequencer Load Array ? Source ? Control ? Length ? Position ? EN DN	Loads reference conditions into a sequencer array.
SQC	Seq. Compare	 SQC Sequencer Compare File #B20:6 Mask 0FFF0h Source I:1.0 Control R6:2 Length 4< Position 2< EN DN FD	RSLogix 500 instruction. Supported by PLCLogix.

3-11 Communication Instructions

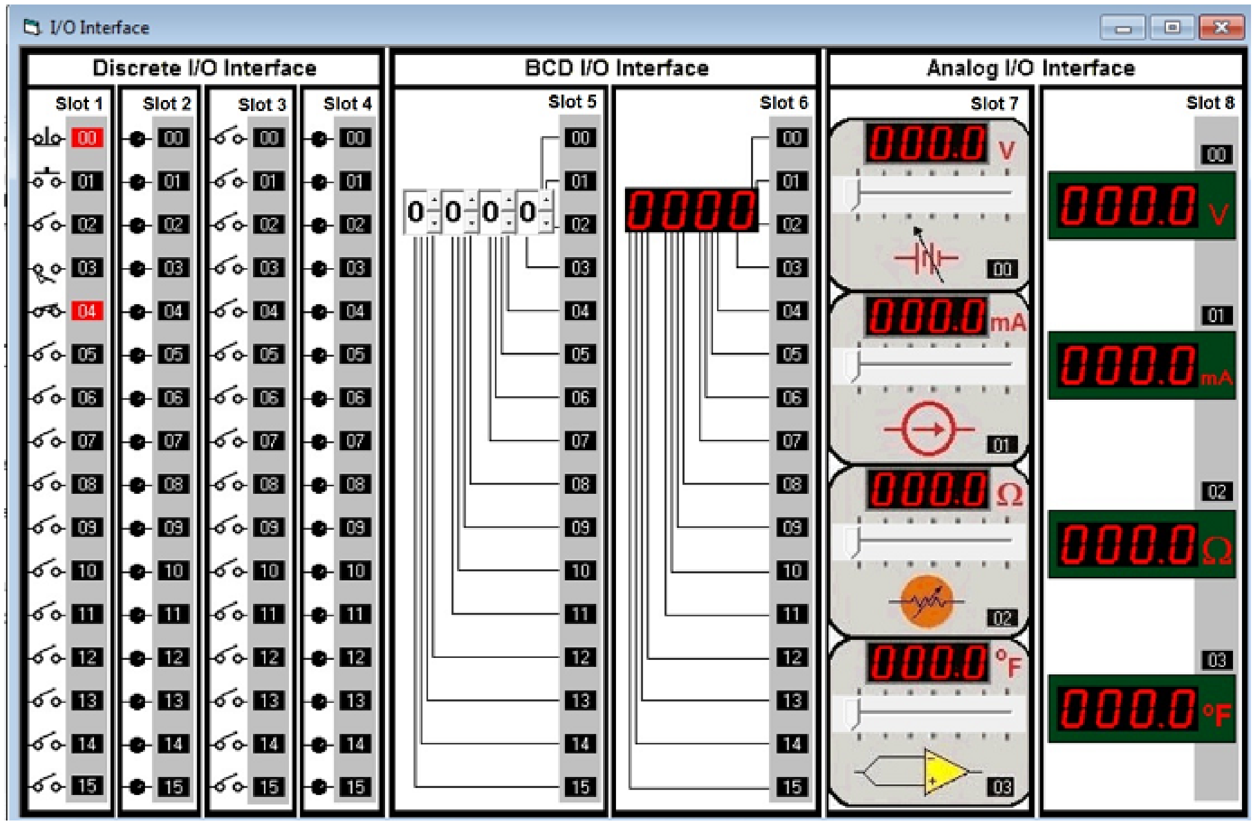
Instruction Mnemonic	Instruction Name	Symbol	Description
GSV	Get System Data	 GSV Get System Value Class name ? Instance name ? Attribute Name ? Dest ? ??	Gets controller system data that is stored in objects.
SSV	Set System Data	 SSV Set System Value Class name ? Instance name ? Attribute Name ? Source ? ??	Sets controller system data that is stored in objects.

Chapter 4: Tutorials

4-1 PLCLogix I/O Chassis

These tutorial exercises are designed to familiarize you with the operation of PLCLogix PLC simulation software and to step you through the process of creating, editing and testing simple PLC programs utilizing the Ladder Logic Instructions supported by PLCLogix.

From the Main Toolbar at the top of the screen, Click on the I/O Chassis tab and select I/O Rack. The simulator screen shown below should now be in view. For this exercise we will be using the Discrete I/O Interface section, which consists of 32 switches and 32 lights. Two groups of 16 toggle switches are shown connected to 2 Input cards of our simulated PLC. Likewise two groups of 16 Lights are connected to two output cards of our PLC. The two input cards are addressed as “Slot 1” and “Slot 3” while the output cards are addressed “Slot 2” and “Slot 4”.



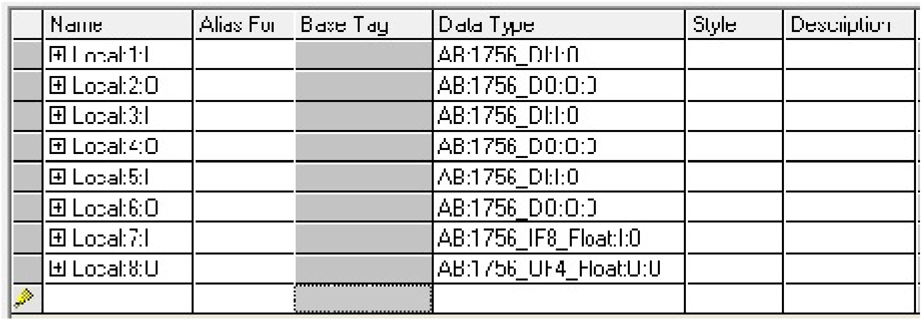
Use your mouse and right click on the various switches and note that the type of switch being used can be changed with each click. There are five possible switches that can be used as a discrete input device: normally open pushbutton, single pole switch, normally open limit switch, normally closed pushbutton, and normally closed limit switch. The color of the light connected to the discrete output device module can be changed by right clicking with your mouse on one of the lights and selecting from one of three colors" red, green, or yellow. Close the I/O rack by clicking on the red X in the upper right corner of the I/O rack window.

4-2 Tags and Aliases Tutorial

Create a new file by selecting File, New from the Main Toolbar. A window should now appear titled "New Controller". Enter a name for the new controller (i.e. Tutorial 2) and click OK. the Controller Organizer should now appear on the left-side of the screen.

To enter tags and aliases, follow the steps below:

1. Right click on the Controller Tags folder in the Controller Organizer and select Edit Tags. The Tag Editor appears, as shown below. The Editor displays the I/O modules (Data Type) and the descriptions for the tag. These descriptions are currently blank since no tag information has been assigned.

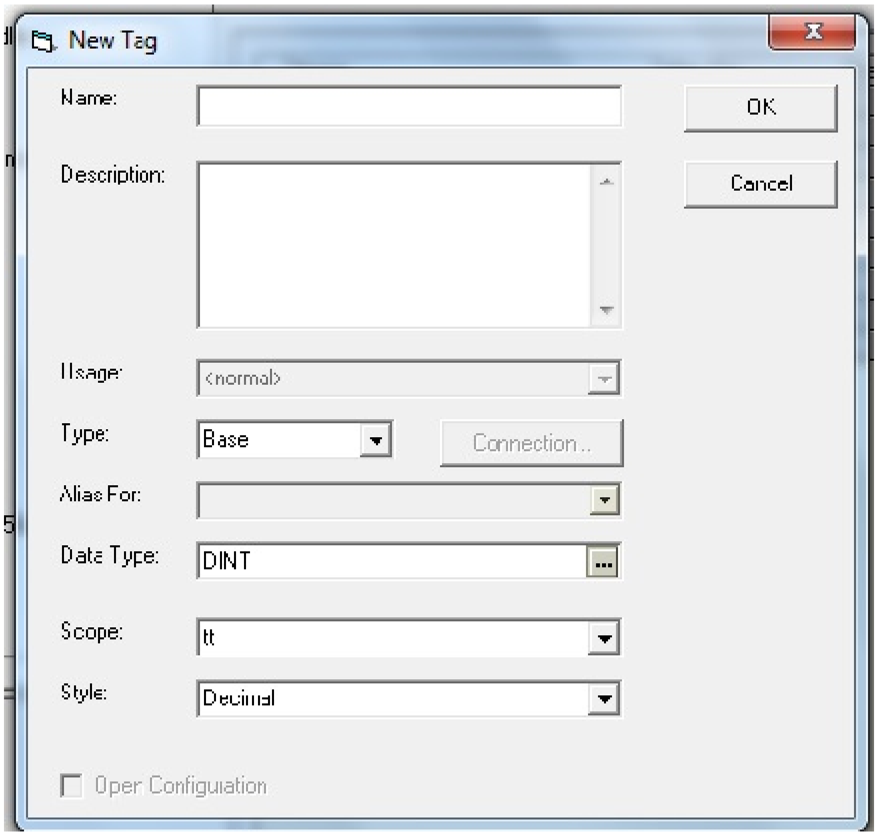


	Name	Alias For	Base Tag	Data Type	Style	Description
	⊞ Local:1:I			AB:1756_DI:I:0		
	⊞ Local:2:O			AB:1756_DO:O:0		
	⊞ Local:3:I			AB:1756_DI:I:0		
	⊞ Local:4:O			AB:1756_DO:O:0		
	⊞ Local:5:I			AB:1756_DI:I:0		
	⊞ Local:6:O			AB:1756_DO:O:0		
	⊞ Local:7:I			AB:1756_IF8_Float:I:0		
	⊞ Local:8:O			AB:1756_OF4_Float:O:0		
✎						

2. To create a Base Tag, enter the tag data type.

3. To create an Alias Tag, enter the tag which the new tag refers to.

Another method to create a new tag is to use the New Tag dialog function, as shown below. The New Tag dialog is accessed from the File tab on the main toolbar and by selecting New Component - Tag. Using this method, the Data Type selected will automatically select a default Style. You can also manually choose the Style in which you want to display the value of the tag (Hex, Binary, or Octal).



Name:
Description:
Usage: <normal>
Type: Base
Alias For:
Data Type: DINT
Scope: tt
Style: Decimal

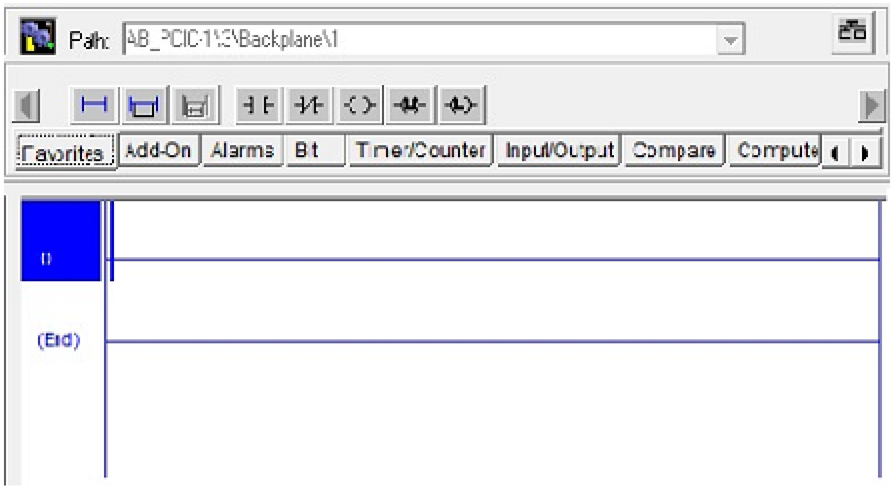
4-3 Ladder Logic Tutorial

Follow these steps to enter the ladder logic you will use to define your programs and routines.

1. Create a new File (File, New) assign a name and click OK.

2. Left click on the "+" in the box beside Main Program in the Controller Organizer.

3. Left double-click on the "Main Routine" displayed in the Controller Organizer. The ladder logic window should now be displayed as shown below.



4. From the Ladder Instruction toolbar, click on the tab corresponding to the instruction group from which you want to add an instruction.

5. Select the desired instruction and the instruction is added to the rung or branch on which you chose to put it.

6. Modify the instruction as necessary. Use the Tag Browser to choose a tag.

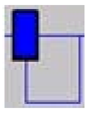
7. Use the Ladder Instruction toolbar to add additional rungs, branches, branch levels, or instructions as required by your routine.

8. Make the necessary modifications to your rung.

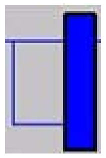
9. Right mouse click and choose “Accept Pending Rung Edit”.

4-4 Branching Tutorial

Add a branch. This icon on the instruction toolbar is used to insert a branch in a ladder logic program. If the cursor is on an instruction, the branch is placed immediately to the right of the instruction. If the cursor is on the rung number, the branch is placed first on the rung.



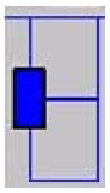
Move a branch level. In order to move a branch level to another location, click on the upper left corner of the branch.



Expand a branch. Click the right leg of the branch, and then drag the leg to the right or left.

Nest a branch. To place another branch structure within the original branch structure, place the cursor at the upper left corner of a branch leg and click on the Add Branch button.

Parallel branch. Place the cursor at the bottom left corner of a branch leg and click on the Add Branch Level button.



Copy branch level. Select the branch level you want to copy by clicking on the left edge and then click Copy Branch Level from the right mouse click menu. Then, click on a rung or instruction in the ladder logic program and click Paste from the right mouse click menu.