ICON Program Instruction List

Last updated: July 26, 2010

Introduction

Programming takes place on the “Program view” tab. To insert a new instruction, either press the insert key or right click on the program to display the menu and select "Insert".

A dialog box allows you to enter a caption (instruction description) for the module or instruction you are about to insert. Next, select which type of instruction you want from the drop-down list. Some of the instructions have another blank to fill in, e.g. the number of expressions desired for an "Expression" instruction. Your new instruction will be inserted after the line in the program that is highlighted.

NOTE: the program automatically inserts the type of instruction for example "Expression" or "Module" as the first word of the caption and is not editable. So if you type "Expression 1" in the description, you will see "Expression Expression 1" in your program. So try to be more descriptive in naming. To change the description after it has been inserted, use the **Recaption** function (ctrl R).

To configure an instruction, double click to bring up the General Parameters configuration window. A few instructions don't have general parameters and you will go directly to table parameters. The Module instruction has no configurable parameters. Many instructions also have a table of parameters to configure.

Select the "Table" button from the instruction's General Parameters window to access the table parameters. For tables, you may click the "Resize" button to change the length of the table.

There are three types of configuration fields. A variable field allows you to type in a text string for the variable name or right click for a list to choose from (see next section). Text/Number fields require you to type in a text string or number as appropriate. Drop-down list fields require you to select from a list. For specifics on how to configure each instruction, see the detailed descriptions later in this document.

Variables

Instructions take values from source or input variables, perform operations on these values (often under the control of control variables) and return the results into result, destination or status variables.

Variable values

In this system a variable stores a value in IEEE four byte floating point standard. However, in addition to float, some instructions (and math functions) convert the float value to an unsigned, 24-bit integer to perform boolean operations and then convert the result back to float. This allows bit wise boolean operators of AND, OR, XOR, compliment, shift right, shift left and integer modulo (division remainder) to be simulated on a floating point number. The number of bits is restricted to 24 because this is the largest integer that can be represent in 4 byte floating point notation without loss of precision.

Time is represented as an integer (in float notation) between 0 and 16,777,215 (0xffffff) seconds. It is displayed as DD:HH:MM:SS where DD is day (00-99), HH is hour (00-23), MM is minute (00-59) and SS is second (00-59). Date is represented as an integer (in float notation) between 0 and 49710 days (136 years). 0 represents January 1, 2000.
To provide better user display and input of these different data types, in addition to float, values may be displayed/entered in decimal integer (0-16,777,215), hexadecimal integer (000000-ffffff), time format (00:00:00:00-99:23:59:59) or date format (YY/MM/DD). The end result of the special display/data entry fields is simply a floating point number representing the integer between 0 and 16,777,215. 16,777,215 is the largest integer that can be represented in 32 bit floating point notation and hence is the upper limit for integers.

Variable names

To define a local variable, simply type a new variable name into any field that requires a variable. Or, to use an already defined variable, right click in any variable field to display a drop down list of all variables defined. A variable name must start with a letter. The rest of the variable name can contain any alpha-numeric character or the underscore (_). Names can be up to 16 characters long. Variables are local only to the current module. A variable by the same name in a different module is a different variable.
If you place a # in front of the variable name the variable is visible to all modules in the selected loop (loop global). If you place an ! in front of the variable name, the variable is visible to all modules in all loops (super global).

When an instruction is first inserted, all variables for that instruction are set to Null to indicate that they are undefined. You may also enter Null to undefine a variable. Some instructions will operate with default values if certain variables are left undefined. However, if a critical variable is left undefined the instruction is effectively disabled. When an instruction is first inserted, since all variables are set to Null, the instruction is disabled.

Variable subscripting

Every variable is actually a subscripted array. The array can be from 1 to 65536 elements long. Every declared variable is at least one element long and can be represented as var_name[0]. For this index only, the [0] is implied and is not displayed although you may enter this notation without error. Variables with additional elements are accessed with explicit subscripting. Therefore var_name [1] accesses the second element of an array, var_name [2] the third etc. The largest index would be var_name[65535] which accesses the last possible element in a given variable array. Normally, however, variables are only one or at most several hundred elements long.

Variable creation is automatic. If you create a variable with the name var_name[25] the variable is created with 26 elements. If the variable exists but is less than 26 elements long it is recreated with the new longer length. When entering variable names during instruction configuration, for instructions that require a subscripted array, the proper sized array will automatically be created.

!**System**

There is a special subscripted system variable called "!System" which contains the following system parameters:

!System[0]	The system date (day of century, 0=January 1, 2000)
!System[1]	The system time as second of day (0-86399)
!System[2]	The system millisecond of day in 10 millisecond steps. (0.00-86399.99)
!System[3]	Master alarm (must be user programmed with an Alarm instruction)
!System[4]	Set to 1 on first pass of loop 1, then 0 (used for startup initializations)
!System[5]	Loop 2 first pass
!System[6]	Loop 3 first pass
!System[7]	Loop 4 first pass
!System[8]	Loop 1 timing interval in milliseconds
!System[9]	Loop 2 timing interval in milliseconds
!System[10]	Loop 3 timing interval in milliseconds
!System[11]	Loop 4 timing interval in milliseconds
!System[12]	Time required to execute loop1
!System[13]	Loop 2 execution time
!System[14]	Loop 3 execution time
!System[15]	Loop 4 execution time
!System[16]	Loop 1 pass missed counter
!System[17]	Loop 2 pass missed counter
!System[18]	Loop 3 pass missed counter
!System[19]	Loop 4 pass missed counter
!System[20]	Serial port 1 (If master then value is number of mS required to execute all I/O instructions) (If slave, number of timeout errors)

!System[21]	Serial port 1 (If slave, number of CRC errors)
!System[22]	Serial port 1 (If slave, number of address or illegal command errors)
!System[23]	Serial port 1 (If slave, number of correctly processed commands)
!System[24]	Serial port 2 ((If master then value is number of mS required to execute all I/O instructions) (If slave, number of timeout errors)
!System[25]	Serial port 2 (If slave, number of CRC errors)
!System[26]	Serial port 2 (If slave, number of address or illegal command errors)
!System[27]	Serial port 2 (If slave, number of correctly processed commands)
!System[28]	Serial port 3 (If master then value is number of mS required to execute all I/O instructions) (If slave, number of timeout errors)
!System[29]	Serial port 3 (If slave, number of CRC errors)
!System[30]	Serial port 3 (If slave, number of address or illegal command errors)
!System[31]	Serial port 3 (If slave, number of correctly processed commands)
!System[32]	Serial port 4 (If master then value is number of mS required to execute all I/O instructions) (If slave, number of timeout errors)
!System[33]	Serial port 4 (If slave, number of CRC errors)
!System[34]	Serial port 4 (If slave, number of address or illegal command errors)
!System[35]	Serial port 4 (If slave, number of correctly processed commands)
!System[36] to !System [255]	Currently unused.
!System[256] to !System [511]	256 X10 code registers. The value is 0 if off and 1 if on.
!System[512] to !System[2559]	2048 Modbus slave registers for TCP.
!System[2560] to !System[3583]	1024 Modbus slave registers for COM1.
!System[3584] to !System[4607]	1024 Modbus slave registers for COM2.
!System[4608] to !System[5631]	1024 Modbus slave registers for COM3.
!System[5632] to !System[6655]	1024 Modbus slave registers for COM4.
!System[6656] to !System[7679]	1024 ASCII slave registers for TCP.
!System[7680] to !System[8703]	1024 ASCII slave registers for COM1.
!System[8704] to !System[9727]	1024 ASCII slave registers for COM2.
!System[9728] to	1024 ASCII slave registers for COM3.

!System[10751]	
!System[10752] to !System[11775]	1024 ASCII slave registers for COM4.

The value of !System[4] is set to 1 on the first pass of loop 1 after power on. This can be used to execute a sequence of instructions in an If statement block to do any required variable initializations. The value of !System[4] is 0 for all subsequent passes. Use !System[5] to !System[7] for loops 2-4 initialization.

!System[8] shows the loop 1 execution interval that you have set. !System[12] shows the actual loop 1 execution time. If loop 1 takes longer to execute than the loop 1 interval a loop pass will be missed and the !System[16] variable is incremented. There are corresponding variables for loops 2 to 4.

During the execution of a loop various input and output (I/O) instructions will be executed to get data from instruments and send control signals to relays and other controls. These operations are buffered so the instructions execute without waiting for hardware operations. A separate thread executes the actual I/O operations on the buffered data. The length of time required to execute one pass of all I/O instructions on a particular serial COM port is recorded to !System[20] for COM1 in milliseconds, !System[24] for COM2, !System[28] for COM3 and !System[32] for COM4.

You should not write to any of these variables except the loop counters (!System[16] - !System[23]) to reset to 0.

!Exp_con and !Exp_tmp
The special variable !Exp_con contains all the constants used in all your expressions. Every time you enter a new constant in an expression the compiler first checks to see if that value has already been entered. If it has then this original value is used. If the value is new then a new subscripted entry is added to !Exp_con. **Do not** write to these variables or the values used as constants will change and you will get wrong results from your expressions.
The !Exp_tmp variable holds all the temporary values during an expression evaluation. Again, do not write to these variables or you will get unexpected behavior.

You may view the values of these variables if you wish, however they are used for behind the scenes operation and are of most interest to the programmer of the ICON firmware to verify correct constant generation, and expression evaluation operations.

Loops
The basic structure for the program includes four loops marked by Module Loop_1, Module Loop_2, Module Loop_3 and Module Loop_4. The end of the program is marked with Module End. These five module instructions cannot be deleted from a program. Under system general parameters you may select the loop execution interval. If loop 1 is configured to execute every second, then at the start of each second, all the instructions in loop 1 are executed. The loop is not executed again until the start of the next second.

You may disable a loop altogether by setting its loop execution time to 00:00:00:00. A higher numbered loop may be interrupted by a lower number loop if execution of the higher numbered loop is not completed in time. A lower numbered loop completes execution of all instructions in that loop before returning to complete execution of instructions in higher numbered loops.

All variable values are maintained between loop execution passes and results from the previous pass become inputs for the next pass.

Instructions

The instructions can be summarized as follows:

Instruction	**Category**	**Description**
Asciimaster	I/O (Input/Output)	Command a remote I/O device that talks serial or TCP ASCII to exchange data.
Asciislave	I/O	Respond to a device that talks serial or TCP ASCII.
Modmaster	I/O	Operate a Modbus RTU slave over serial or TCP. This is how most inputs and outputs external to the ICON are accessed.
Modslave	I/O	ICON responds to an external Modbus master RTU device over serial or TCP. This can be used for ICON to ICON communication.
X10	I/O	Only used in the very special case where the ICON is controlling X10 devices in a home automation application.
OWconfig	I/O	Used to configure the ICONIO Modbus to 1-wire I/O board.
Onoff	Control	Control on/off digital controls.
Pid	Control	Execute analog Proportional, Integral and Differential (PID) control loops.
Pwm	Control	Pulse a digital control on and off in a Pulse Width Modulation (PWM) manner in proportion to an analog control variable value.
Timepos	Control	Position a controller by controlling an open and close relay. Used for vent and valve position.
Circbuf	Statistics	Calculates a rolling average, standard deviation, maximum, minimum, total, difference, circular average (wind direction) and circular standard deviation (wind direction sigma theta) on a circular buffer of data.
Statistics	Statistics	Calculates average, standard deviation, maximum, minimum, total, difference, circular average (wind

		direction) and circular standard deviation (wind direction sigma theta) over a user specified time period.
Histevt	Historical data	Save the occurrence of an event or alarm in one of four historical event files.
Histper	Historical data	Save the results of statistical calculations on a periodic time base in one of four historical periodic files.
Limit	Alarms or control	Test the value of a variable against high and/or low limits to generate values that can be used in alarms.
Alarm	Alarms	Check an array of alarms and set the output variable true when any alarm changes from a no alarm to alarm condition.
Bounds	Process input	Clamp an input between an upper and a lower bound.
Convert	Process input	Convert an input to engineering units using an interpolating polynomial.
Correct	Process input	Correct an instrument reading for zero and gain drift.
Counter	Process input	Create an event counter or measure frequency
Fuzin	Process input	Fuzify an analog input to a fuzzy variable range (0-1).
Lookup	Process input	Convert an input to engineering units using a lookup table.
Translate	Process input	Translate a counting sequence of values to a random list of values or vice versa.
Fan	Block move	Move a block of variable values from one variable to an indexed position in a second block or perform the reverse.
Map	Block move	Move values from a random set of variables to a single indexed variable list.
If/Elseif/Endif	Program structure	Execute blocks of instructions conditionally depending on which If or Elseif expression evaluates to a true (non-zero) value.
Module	Program structure	Mark the beginning of a program block called a module. All local variables within a module are different from variables in a second module even if the variables are of the same name. Also, each module can have one HMI screen.
Timer	Timing functions	Create up and down timers.
Constant	Constants and persistent storage	Introduce constant values into a program. Can be used for program startup initialization. Can be used to capture values during operation to flash memory for initialization after startup after power failure.

Expression	Evaluate mathematical expressions	Can evaluate mathematical expressions. Includes trigonometric, log, date and time functions along with boolean and fuzzy logic operators.

Module

A Module is a special type of instruction. It has no function itself other than to help organize your program into sections called modules and keep local variables separated from each other. Certain functions may only be performed on modules. Modules contain a group of related instructions. Each module can have one HMI screen. Any variable name not starting with an ! or # is unique to the module.

Alarm

This instruction sets a master alarm variable to a value of 1 if any alarms in an array of alarm variables change from 0 to non-zero. The master alarm may then be set to 0 from an HMI screen and the master alarm variable will not change to a 1 again until another 0 to non-zero transition of any alarm variable in the alarm array. If the master alarm is set to 1 and all alarms clear to 0 then the master alarm clears to zero also.

If the master alarm variable is set to !System[3], then the HMI screen for module "End" (the last module in the ICON program) is selected when !System[3] is a 1.

General Parameters

Master:	Enter your master alarm variable name. Use !System[3] to bring up the master alarm HMI screen.
Alarm:	The start variable in an array of alarm variables of length "Number of alarms".
Number of alarms:	The number of alarm variables in the alarm array.

Asciimaster

This instruction provides a way to prompt an ASCII device for data and convert the returned ASCII string to numeric values for use within the ICON program.

General Parameters

Enable	If Null the instruction is always enabled. Otherwise the variable value must be 1 for the instruction to execute.
Status	If Null, a return status is not available to your program but the instruction will still be executed. Otherwise the value is set as follows: 1 Receiving valid responses. 2 Timeout because no characters received. 3 Received at least one character in response string 4 Still waiting for more characters in response string
TX data	The optional variable containing the value or array of values inserted into the TX string when prompting the ASCII device for data.

RX data	The variable (or variable array) used to receive the value(s) from the ASCII device's response string
Com type	Select the desired serial port or ethernet for TCP operations.
IP address	If ethernet is selected enter the IP address of the ASCII device.
Port number	If ethernet is selected enter the port number of the ASCII device.
Timeout (10 mS steps)	Enter the maximum length of time to wait for a response from the ASCII device before aborting.
Retries	Enter the number of times to attempt to get the data before setting a bad status value in the status variable.
Number to send	Enter the number of values to be inserted into the TX string to be sent to the ASCII device. These variable values obtained from the TX data variable array are the only items that can change in the TX string.
Number to receive	Enter the number of expected values to be parsed from the ASCII device's response string.
TX string	Enter the ASCII string to be sent to the ASCII device. Use a "#" to indicate the insertion of the next value from the TX data variable if needed. For example, w,1,#,# would send the string w,1,value of tx[0],value of tx[1] if the TX data variable was called tx.
RX parse control	Enter a string to define parsing. Use the "#" to indicate parsing a number. Separate the #s by the parsing character. For example, #,#;# would parse the string for a number up to delimiter , then parse another number up to delimiter ; and then parse a third number. The results would be placed in rx[0], rx[1] and rx[2] if the RX data variable name is rx.

Asciislave

This instruction transfers Ascii register values between the ICON Ascii slave registers and ICON variables. Asciislave operation is configured under "System Parameter Config". Configure under "HMI edit/Serial ports" section if doing serial Ascii and under "Internet connections" if doing TCP Ascii.

General Parameters

Enable	If Null the instruction is always enabled. Otherwise the variable value must be 1 for the instruction to execute.
I/O transfer	Transfer data between the ICON Ascii slave registers and the ICON variable specified here.
Com type	Select between Ethernet, Serial 1, Serial 2, Serial 3 or Serial 4.
Register address	Select the Ascii slave starting register address. This can be 1-1024 for either ethernet or the serial ports.
Block length	Select the number of variables to transfer. You can't transfer beyond the last register. So the maximum length is 1024 if you start with register 1.
Direction	You may select between "Read from Ascii registers" or "Write to Ascii

	registers".

The Ascii slave software expects an ASCII device to send an ASCII string in the following format to write data to the ICON's Ascii registers.

w,start address,data 1, data 2, ...

Start address must be in the range of 1 to 1024. For example the string "w,1,1,2,3,4,5,6,7,8" would write the values 1-8 to registers 1 to 8.

The Ascii slave software expects an ASCII device to send an ASCII string in the following format to read data from the ICON's Ascii registers.

r,start address,length

Start address must be in the range of 1 to 1024 and the length must be such that the range does not go beyond 1024. For example the string "r,1,8 would cause the following ASCII string to be sent "1,2,3,4,5,6,7,8" assuming those were the values in the first 8 Ascii registers.

For Ascii slave strings, the delimiter is always a comma (,).

Bounds

This instruction sets the result value to the value of the input. If the input value is less than Low bound, the result is clamped to the Low bound value. If the input value is greater than High bound, the result is clamped to the High bound value. The Alarm variable value is set to 0. If the input value is less than Low bound alarm, the Alarm is set to the Low alarm value. If the input value is greater than High bound alarm, the Alarm is set to the High alarm value.

General Parameters

Input	Bounds check this value.
Result	Contains the bounds checked result.
Alarm	Contains the state of the alarm or 0 if no low or high alarm.
High alarm	The high alarm value.
Low alarm	The low alarm value.
High bound alarm	If the Input is greater than this value the High alarm is written to the Alarm variable.
High bound:	If the Input is greater than this value the Result is clamped to this value.
Low bound:	If the Input is less than this value the result is clamped to this value.
Low bound alarm:	If the Input is less than this value the Low alarm is written to the Alarm variable.

Circbuf

This instruction creates a circular buffer with the specified number of entries N. During each pass of the loop a new value and status pair are entered into the circular buffer. When the end of the buffer is reached the program begins writing over the oldest data. Once full, the buffer always contains the last N data values with corresponding status values.

Many different statistical calculations may be made to this data. For each calculation, the number of values, minimum number valid and calculation type may be configured. To make a calculation the program steps back through the specified number of data entries calculating the specified statistic (ex. average). Before each value is used its status is checked against "Max valid status" If the status is less than or equal to the "Max valid status" the value is used in the calculation and the "OR" of all "valid" status bits is made to a "valid" status temporary variable.

If the status is greater than "Max valid status" the value is not used in the calculation and the "OR" of all "invalid" status bits is made to an "invalid" status temporary variable. The result of the calculation is posted to the results[even] index and the status to the results[odd] index. If there are at least "Minimum valid" number of data points the "valid" status value is set in results[odd]. If there are less than "Minimum valid" number of data points the "invalid" status value is set in results[odd].

General Parameters

Input:	Points to a variable with two values. The first value is the actual data and the second value is interpreted as status.
Results:	The results (in pairs) of the circular buffer calculations. The calculation result is in the lower (even) numbered index and the status is in the upper (odd) index. If there are no valid readings in the circular buffer the result is not updated (but of coarse the status is).
Control:	If Null always enter data and calculate. Otherwise a value of 0 does nothing, 1 enters input value into the circular buffer and calculates results, 2 enters input value into buffer but does not update the calculation results and 3 enters input value into buffer, calculates results and then sets all status elements in circular buffer to 0xffffff to mark as invalid, effectively clearing the buffer. A value of 4 performs the same function as 1 but does not increment the circular buffer data position. Hence data is always saved to the same circular buffer location until a different control value is sent.
Max valid status:	Enter the highest value for a status that is considered valid. For instance, if any status bits 0, 1, 2 or 3 when set still indicate valid status enter a value of 15 (1111 binary).
Circular buffer length:	Set the maximum length (number of elements) in the circular buffer before overwriting oldest data.

Table Parameters

Calc type:	This selection set allows you to pick the type of calculation to perform on the buffer of data for this result. Choices include average, standard deviation, maximum, minimum, total, difference, circular average (wind direction) and

	circular standard deviation (wind direction sigma theta).
Number in calc:	Enter the number of elements to use in the calculation. For instance, if the buffer is 60 elements long but for this calculation you only want to use the last 10, you may enter 10.
Minimum valid:	Enter the minimum number of data points that must be valid in the circular buffer for the result of the calculation to be valid.

Constant

This instruction loads one or more constant values into a variable array. Note, this instruction provides a special "Capture" mode which provides the only mechanism for saving variable values through a power failure (persistent storage).

General Parameters

Variable:	The constant value is saved in this variable. If the table is longer than 1 the values continue as a subscripted array.
Data type:	Values are always stored as floats. But if you select Date, Time, Dec or Hex integer you will be given an alternate input field.
Mode:	For "Normal" the constant values defined in the table are always moved to the variable array every time the instruction is executed. For "Startup" they are only moved on the first pass. For "Capture" the constants are moved to the variables at startup and the constant values in the table are updated (and saved on disk) if the variable values change to differ from the values in the constant table. "Disable" disables the instruction from doing anything when executed.

Table Parameters

Value:	The constant values to be moved to the variable array.
Description:	A description string.

Counter

This instruction takes as "Input" a count value that counts from 0 to 65535 (16 bit) and then wraps back to 0. This count can be interpreted as event counts or frequency. There are three modes of operation:

1. Incremental counter. In this mode, the previous count value during the last loop scan is subtracted from the current input count and the difference is placed in the Result.
2. Absolute counter. In this mode the Result contains the value of an absolute counter that is incremented by the result of the incremental counter on each scan. A base value may be set which is subtracted from the absolute counter value to allow a 0 or preset result to be set at any count value. This mode is used for the 1-wire rain fall measurement.

3. Frequency measurement. Calculates frequency from the number of counts received during the configured frequency period.
4.

General Parameters

Input:	The input count from a 16 bit counter (generally a 1-wire OWCTR2 or Modbus device).
Result:	The event counts or frequency is placed here.
Enable/Set:	For incremental counter or frequency mode, if Null then instruction is always enabled. Otherwise the Enable value must be set to 1 for the instruction to count or measure frequency. Any other value for Enable disables counting or frequency measurement functions. For absolute counter mode, setting this Set value to a non-zero value causes the Base variable to set to the value "absolute counter value - Set value". Use a value of 65536 to set the Base value to the current absolute counter value. After the Base value is set, the Set variable value is reset to 0. For this mode, the Result is calculated as "absolute counter value-Base value".
Update/Base:	Set to "Null" for incremental counter function. Set to a variable whose value sets the "Base" for absolute counter functions. The Result value is calculated as "absolute counter value-Base value". Set to the I/O input register (first value in the Modbus 1-wire data input array) that increments when a new count value is available on "Input" for frequency measurement. This insures that a false frequency of 0 is not returned if the frequency measurement period is faster than the 1-wire scan time.
Frequency period:	For frequency measurements only, enter a non-zero time period over which to make the frequency measurement (Often 10 seconds). Frequency measurement only works if in a loop with a scan time of once per second or slower.

Convert

Converts values using an interpolating polynomial. Most common use is to convert raw inputs (from sensors) to engineering units. The instruction allows you to enter a table of input (raw) and converted values. The program will perform a least squares fit of your data to generate conversion coefficients A-F for the conversion polynomial. This has the form Y=A+B*X +C*X^2+D*X^3+E*X^4+F*X^5+G*X^6+F*X^7 where Y is the converted value and X is the raw input value. A-F are the conversion coefficients that are labled 1-8 in table parameters.
The information to calculate the coefficients can be entered in one of two ways.

1. Configure the "Select function" and "Poly degree" in the general parameters and fill in the "Input value" and "Converted value" pairs in the "Table parameters" section. Set the table length to the number of input/converted value pairs. Leave the five variables

"Raw data" to "Control" in the General parameters section null.
2. Configure the five variables "Raw data" to "Control" as described in General parameters.
3.

General parameters

Input:	This variable value is converted using the interpolating polynomial.
Result:	The converted value is returned here.
Select function:	"Log" takes the log of the input before applying the conversion.
Poly degree:	Selects the degree of the polynomial (1-7). You must have at least one more input/convert pair in the table below than the degree of the polynomial.
Raw Data:	An array of 20 variables that contain the "Raw" readings. You must fill in a minimum of the first 2 values. You must have at least one more raw-convert pair than the degree of the polynomial.
Converted Data:	An array of 20 variables that contain the "Converted" readings. You must fill in a minimum of the first 2 values. You must have at least one more raw-convert pair than the degree of the polynomial.
Calculated Coefficients:	The calculated conversion coefficients which are calculated from the raw data and converted data arrays above under control of the control array below.
Error Data:	The error data array contains the error of the calculated converted value (calculated from the raw value converted with the polynomial) - the expected converted value.
Control:	Set control[0]=1, control[1]=degree of polynomial (1-7), control[2]=number of pairs (2-20) and control[3]=0 (Normal) or 1 ("Log" takes the log of the input before applying the conversion). After the coefficients are calculated control[0] is automatically set to 0 and control[4] is set to the absolute value of the largest error in the error data array above.

Table parameters

Conversion coefficient:	The conversion coefficients used in the equation Y=A+B*X +C*X^2+D*X^3+E*X^4+F*X^5+G*X^6+H*X^7 where Y is the converted value and X is the raw input value. A-F are the conversion coefficients that are labeled 1-8.
Input value:	The table of raw input values that are used by the program to calculate the best least squares fit to this input/converted data set.
Converted value:	The table of converted results that are used by the program to calculate the best least squares fit to this input/converted data set.

Correct

This instruction is designed to be part of an instrument auto calibration routine. Uncorrected instrument data is corrected based on the results of a two-point calibration.

General Parameters

Uncorrected input:	Raw uncorrected instrument data is input from this variable. It is used to calculate the Corrected result and is also placed in the averaging circular buffer so that averaged instrument responses can be calculated.
Corrected result:	The corrected instrument data result is placed here.
Expected cal:	Points to a variable with two values. The value at the lower index is the expected low calibration response of the instrument. The value at the higher index is the expected high calibration response.
Captured cal:	Points to a variable with four values. The value at index 0 is the actual instrument response to the expected low value. The value at index 1 is the actual instrument response to the expected high value. The value at index 2 is the correction offset. The value at index 3 is the correction gain.
Control:	The value on this variable controls the capture of the instrument expected low and high responses.
Correction type:	You may select None, Zero or Zero/span. If None is selected, the Corrected result is set the same as the Uncorrected input. If Zero is selected the output is corrected for Zero or low offset error only. If Zero/span is selected a two point linearization is applied to the input data to correct for both offset and gain error.
Low capture match:	When the Control value equals this value the low response is captured.
High capture match:	When the Control value equals this value the high response is captured.
Number average elements:	The response values are calculated by using this number of elements. A result is actually captured or saved only when the Control value equals a Low or High capture match.

Expression

This instruction evaluates one or more math expressions and puts the results in the specified variables. You may enter constants in your expressions. Prefix a value with 0x to indicate hexadecimal. Otherwise the value is assumed to be in decimal. (There is no octal or binary format support.) The expressions are similar to ‘C’ with the following exceptions:

- All operators are executed from left to right. Precedence can be changed with parentheses groupings.
- The "Or" function is ‘:’ (colon) instead of ‘|’ (pipe).
- There are no pointers or unary operators.
- The built-in functions have only one parameter and the unary operators are now functions.
- Two parameter functions have been changed to operators.
- A number of new operators and functions have been added.

You might ask, what is the difference between having one Expression instruction with multiple expressions and multiple Expression instructions, each with just one expression? From an operational point of view, a higher priority loop cannot interrupt operation of a lower priority loop except between instructions. Therefore, an Expression instruction with multiple expressions, cannot be interrupted until all the expressions within that instruction are executed. From a logical grouping point of view, it is often desirable to group related expressions in the same instruction.

Operators:

+	Add
-	Subtract
*	Multiply
/	Divide
%	Integer modulo (remainder of division)
:	Integer bitwise OR
&	Integer bitwise AND
^	Integer bitwise XOR
:>	Fuzzy logic OR (Maximum of operands)
&>	Fuzzy logic AND (Minimum of operands)
^>	Fuzzy logic XOR (Absolute value of difference of operands)
>>	Integer shift right
<<	Integer shift left
::	Logical OR (1 if either or both operands are non-zero, 0 if not)
&&	Logical AND (1 if both operands non-zero, 0 if not)
==	Logical equals (1 if operands are equal, 0 if not)
!=	Logical not equals (1 if operands are not equal, 0 if not)
<	Logical less than (1 if left operand is < right operand, 0 if not)
<=	Logical less than or equal (1 if left operand is <= right operand, 0 if not)
>	Logical greater than (1 if left operand is > right operand, 0 if not)
>=	Logical greater than or equal (1 if left operand is >= right operand, 0 if not)
'if'	If the expression to the right of the 'if' is non-zero, the result is the value of the expression to the left of the if. Otherwise the result value is not changed
‘orider’	Result is left operand if right operand>=1E38 else result is right operand
‘oridel’	Result is left operand if right operand>1 or <0 else result is right operand
‘min’	Result is the minimum value of the left or right operand
‘max’	Result is the maximum value of the left or right operand
'pow’	Result is set to the left operand raised to the power of the right operand
‘atn2’	Same as ‘C’ function atn2(left operand,right operand)

'vpd'	Result is vapor pressure deficit in inches of water with temperature in degrees C (left) and Relative Humidity in % (right)
'dp'	Result is dew point in degrees C with temperature in degrees C (left) and Relative Humidity in % (right)
'outb'	Outputs a data byte (right) to the I/O port address (left). Example: 0x340 ‘outb’ 0x55. In this example the data 55 hex will be sent to the I/O port at address 340 hex. The address and data can themselves be expressions. The expression returns a result of 0 for proper operation and 3E38 if the address is illegal.
'out2b'	Outputs two bytes (value on right 0-65535) to the I/O port address (left) and I/O port address+1. Example: 0x340 ‘out2b’ 0x0A55. In this example 55 hex is sent to I/O port 340 hex and 0A hex is sent to I/O port 341 hex. If this represents a Digital to Analog (D-A) converter the analog output would be updated to the value represented by 0A55 hex.
‘loopz’	Loop while masked I/O port value is zero. The left operand selects the I/O port for the status byte and the right operand defines a mask. The loop instruction will read the status and bit wise "and" with the mask. Example: 0x340 ‘loopnz’ 1. In this example the expression will wait until the low bit of the byte read from I/O address 340 hex sets to a 1. If the loop ends within 10 mS a 0 is returned. Otherwise 3E38 is returned to indicate a timeout.
'loopnz’	Loop while masked I/O port value is non-zero which is the reverse logic from the above 'loopz' instruction.

Note: For "Integer" (binary) operations the float value is converted to an unsigned, 24 bit integer. If the float value is greater than 16,777,215 the value is clamped to 16,777,215 and if the value is less than 0 it is clamped to 0 before conversion to an integer.

For the fuzzy logic OR, AND and XOR functions the operand values are tested for valid fuzzy values (0 to 1) before executing the operation. If an operand is less than 0 its value is set to 0. If an operand is greater than 1 it’s value is set to 1.

Functions:

logical(X)	Result is 0 if X=0, result is 1 if X!=0
bool(X)	Result is 0 if X<0.5, result is 1 if X>=0.5
fuzzy(X)	Result is 0 if X<=0, result is 1 if X>=1, otherwise result=X
compl(X)	Result is 0 if X is non-zero, result is 1 if X=0
compb(X)	X is converted to an integer and a bitwise compliment is performed
compf(X)	Result is 1-X
abs(X)	Result is the absolute value of X
neg(X)	Result is the negative of X
inv(X)	Result is 1/X
ceil(X)	Result is the first integer >= to X
floor(X)	Result is the first integer <=X

sqrt(X)	Result is the square root of X
exp(X)	Result is e raised to the X power
log(X)	Result is the natural log of X
log10(X)	Result is log base 10 of X
sin(X)	Result is sine of X
cos(X)	Result is cosine of X
tan(X)	Result is tangent of X
asin(X)	Result is arc sine of X
acos(X)	Result is arc cosine of X
atan(X)	Result is arc tangent of X
year(X)	Result is the year of date X
month(X)	Result is the month of date X
day(X)	Result is the day of date X
dow(X)	Result is the day of week of date X (0=Monday, 6=Sunday)
hour(X)	Result is the hour of time X
minute(X)	Result is the minute of time X
second(X)	Result is the second of time X
hbit(X)	Result is count of position of highest bit set in X
lbit(X)	Result is count of position of lowest bit set in X
inb(X)	Result is I/O byte read from address X. Example: inb(0x340). The result is the value of the byte (0 to 255) at I/O address 340 hex. The address can also be a variable (but not a complete expression). If the address is illegal a value of 3E38 is returned.
in2b(X)	Result is I/O byte read from address X + 256*I/O byte from address X+1. Example: in2b(0x340). The result is the value of the byte at I/O address 340 hex plus the value of the byte at address 341 hex multiplied by 256 (same as shifted left 8 positions). If these two I/O locations represent the result register for an Analog to Digital (A-D) converter then the result is the last A-D reading. Again, an illegal I/O address returns a result of 3E38.

Note: The operand of the function can only be a constant or a variable.

For hbit and lbit, 0 indicates no bit set otherwise a value of 1-32 indicates the bit position from right to left. 00000000000000000000000000000010 would return a value of 2. 10000000000000000000000000000001 would return a value of 32 for hbit and 1 for lbit.

If you must send a byte out to an I/O port to select an A-D channel and start an A-D conversion, then loop while waiting on a status bit from another I/O port to determine when the A-D conversion is complete and finally read the result of the A-D conversion. To facilitate the loop function the loopz and loopnz operators are provided. A clever expression can be formed to read an A-D converter as follows:

(ad_start_adr ‘outb’ ad_start_val)+(ad_stat_adr ‘loopnz’ ad_stat_mask)+in2b(ad_result)

The expression in the first () starts the A-D converter, The expression in the second () waits for the conversion to complete and the in2b function reads the result. The first 2 expressions return 0 if executed properly or 3E38 if not. Therefore, if the result is between 0 and 65535 the conversion was proper. If the result is greater than 65535 the conversion is bad!

Table Parameters

Variable:	The result of the calculation is placed in this variable.
Status:	After an expression is entered, click the "Update" button to compile your expression. Then click "Refresh". If "Status" is 0 the expression compiled correctly. Otherwise the number indicates the column in which a syntax error was found.
Expression:	Enter your math expression here. Double click to create an expanded field for expression entry if you need a wider field.
Comment:	A comment describing the operation of the expression.

Fan

This instruction copies one or a block of several values from a subscripted source variable to a subscripted destination variable. It is designed to provide three functions:

1. Move a subscripted block of variable values from one variable to another. This is useful to copy an array of local values to an array of global values for module value export. (Set "Count index", "Time index", Time def" and "Last period" to Null.)
2. Select one value (or one block) from a position within the source variable array based on a selection index and write to the destination variable. (Set Direction to "Multiple sources to one destination".)
3. Select the source value (or block) and write to a position within the destination array based on a selection index. (Set Direction to "One source to multiple destinations".)

Notice a fan can move just one value at a time if the block size is 1 but can move a larger block if the block size is greater than 1. For instance, if direction is "Multiple sources to one destination", the "Block size" is 4 and the effective index value is 0, then source[0] to source[3] are moved to destination[0] to destination[3]. If the effective index is 1 then source[4] to source[7] are moved to destination[0] to destination[3].

You can turn the operation around by selecting the direction as "One source to multiple destinations". Now if the "Block size" is 4 and the effective index value is 0 then source[0] to source[3] are moved to destination[0] to destination[3]. If the index is 1 then source[0] to source[3] are moved to destination[4] to destination[7].

The effective index value can be created two ways.

1. If a variable is defined for "Count index" then this value creates the effective index. If an index is greater than the value set in "Number of indexes"-1 then the effective index

is set to "Number of indexes"-1.
2. If a variable is defined for "Time index" then this value is compared against the values in the array variable "Time def" to create an index as described below".

For this example assume that the "Time index" variable is called time and increments from 0. There is also a "Time def" variable called time_def which is configured to have the following values:

time_def[0]=10
time_def[1]=10
time_def[2]=30

Now if time<=time_def[0] then an index of 0 is created. So in this example, when time<=10 then an index of 0 is created. If time>time_def[0] and time<=time_def[0]+time_def[1] then index 1 is generated. So in this example, when time>10 and time<=20 then index 1 is generated. If time>time_def[0]+time_def[1] and time<=time_def[0]+time_def[1]+time_def[2] then index 2 is generated. So in this example, if time>20 and time<=50 then index 2 is generated. If time>time_def[0]+time_def[1]+time_def[2] then index 3 is generated. So in this example, if time>50 then index 3 is generated.

When creating indexes by time, a special feature can be enabled by creating a variable for "Last period". This variable will contain the value 0 except during the last loop scan at the end of each period when it will contain 1. So in the above example the value will be 1 when time=10, time=20 and time=50.

General Parameters

Source:	The data source.
Destination:	The data destination.
Count index:	The index determines which block of data is selected. If the index is greater than the maximum index allowed, the last block is moved. (Set to "Null" to use indexing by time or just a basic block move as described in 1. above.)
Time index:	A variable containing seconds since start of sequence. This value is usually generated from a Timer instruction or variable !System[1] (seconds of day). (Set to Null to use indexing by count or just a basic block move as described in 1. above.)
Time def:	Points to a variable array with time (second) values. Each value is the duration for the index. If the value of the Time index is less than or equal to the first duration time, the first block is moved. If the value of the time index is greater than the first time interval but less than or equal to the sum of the first and second time intervals, the second block is moved. This indexing continues until the end of the Time index definition table is reached. If the time is greater than all the time intervals, the last block is moved. (Set to Null to use indexing by count.)
Last period:	If this variable is non Null, a value of 1 is written to this variable during the last

	loop scan of the program for each index period. By using this variable, the end of each time period can be indicated. A 0 is written to this variable during all other times.
Block size:	Indicates how many variable indexes to move during a fan operation.
Number of indexes:	Indicates the maximum number of index blocks. If the value is 10 then only effective indexes of 0 to 9 are allowed.
Direction:	Select "Multiple sources to one destination" or "One source to multiple destinations".

Fuzin
This instruction converts an input value to a fuzzy variable (0-1).

General Parameters

Input:	The variable value to test.
Result:	The result of the fuzzy conversion.
Activate:	If the input is less than activate the result is 0.
Full activation:	As the input varies from Activate to full activation, Result varies from 0 to 1.
De-activate:	The Result is 1 if the input value is between Full activation and de-activate.
Full de-activation:	As the input varies from De-activate to full de-activation the Result varies from 1 to 0. The result is 0 for all input values greater than full de-activation.

Histevt
General Parameters

Data:	The event data value to be recorded to the event file.
Status:	The event status to control when an event record is recorded to the event file. The status is also recorded to the file.
Event file:	You may select the event file in which to record the event (1-4) or 0 to disable the instruction. The actual file configuration takes place under System Parameter Config.
Event number:	The event number within the file. Up to 1024 different events can be defined per event file.
Force midnight record	If the value is selected as "Yes", then an event is recorded at midnight regardless of the state of the status variable. This feature can be used to establish values for graphing. With this feature enabled, the graphing section of the HMI reporting package can be guaranteed to know the value of an event at the start of a day.

When the Status value is odd (low bit set) the current date and time, Data value, Status value and Event number are written to the selected Event file. The event file can be used to record the setting and clearing of alarms and the results of calibrations.

Histper

General Parameters

Data:	This is the variable array of data to write to the file.
Status:	The value is set to 1 during the pass data is written to the periodic file on flash disk. It is 0 for all other passes.
Periodic file:	You may select the periodic file in which to record the array of Data (1-4) or 0 to disable the instruction. The actual file configuration takes place under "System Parameter Config". Data is actually transferred from the array to the disk file on the interval specified under "System Parameter Config".

If, Elseif and Endif

These three instructions allow you to create blocks of code that only execute if conditions are true. To start a block choose the "If" statement. Next enter your conditional instructions and end the block with "Endif". An "If" statement is configured identically to the "Expression" instruction. The only difference is that if the last expression in the "If" instruction evaluates to non-zero, then all the instructions between the "If" and the "Endif" are executed. If the last expression in the "If" evaluates to 0, the instructions between the "If" and the "Endif" are skipped.

You may add sub blocks within the "If" block with the "Elseif" statement. (The "Elseif" statement is configured identically to the "If" statement). In this case, if the first "If" statement evaluates to non-zero, then all instructions from the "If" to the first "Elseif" are executed. Then all instructions are skipped until the "Endif" is reached. If the first "If" evaluates to 0, then all instructions are skipped until the first "Elseif" instruction. If this instruction evaluates to non-zero then all instructions from this "Elseif" to the next "Elseif" are executed. Then all instructions are skipped from the second "Elseif" to the "Endif". The first block of instructions who's "If" or "Elseif" instruction evaluates to non-zero is executed. All other blocks are skipped. You may nest "If" – "Endif" blocks within "If" - "Elseif" or "If" - "Endif" blocks.

Limit

This instruction tests an input value against limits and returns a result depending on the comparison.

General Parameters

Input	The variable value to test.
Result	The result of the comparison.
Limit	If non null then this subscripted array variable contains the limit values.
High	Result is set to this value when the input is greater than high limit and has not gone below High clear.
Went high	Result is set to this value the first time the input is greater than high limit.
Window	Result is set to this value when the input is between high and low limits.
Went window	Result is set to this value the first time the input is between high clear and low clear limits after being in either the high or low states.

Low	Result is set to this value when the input is less than low limit and has not come above Low clear.
Went low	Result is set to this value the first time the input is less than low limit.
High limit	Sets the high limit.
High clear	High does not clear until the input goes below this value.
Low clear	Low does not clear until the input goes above this value.
Low limit	Sets low limit.

Lookup

This instruction converts a raw input value to a converted value using a lookup table. The table can be up to 128 pairs long. If a match is not found in the table the result is not updated. This way more than one lookup table can be cascaded if there are more than 128 entries required.

The table consists of from two to 128 raw-convert pairs. First the input value is checked to see if it is >= to the first raw value and <= to the second raw value. If it is then the result is the interpolated value between the first converted value and the second.

If the value does not match then the input value is checked to see if it is >= to the second raw value and <= to the third raw value. This checking continues until a match range is found. The result is then the interpolated value between the converted pairs that correspond to the raw range selected.

General Parameters

Input:	The raw input data value.
Result:	The resultant data value.

Table Parameters

Raw:	The raw value match.
Converted:	The converted result.

Map

This instruction can map values from a random collection of variables to an ordered set in the Array. This is useful for creating the input array to the Histper instruction. Or the instruction can disperse values from the Array to a random collection of variables. This is useful for taking Modbus input values from the input array and sending them to individual variables.

General Parameters

Array:	The ordered array.
Direction:	"To array" takes the Individual values and moves the values to the ordered array. "From array" takes the values from the ordered Array and moves the values to the Individual variables.

Table Parameters

Individual:	The table of individual variables.
Description:	A description string

Modmaster

This instruction executes Modus master RTU instructions.

General Parameters

Enable:	If Null the instruction is always enabled. Otherwise the variable value must be 1 for the instruction to execute.	
Status:	If Null, a return status is not available to your program but the instruction will still be executed. Otherwise the value is set as follows:	
	1	indicates successful operation.
	2	Timeout (caused by no connection to Modbus device, bad Modbus device, bad serial link, wrong unit address, wrong baud rate or wrong parity).
	3	At least 1 character was received (wrong baud rate).
	4	Good unit ID but no body or checksum (bad serial link).
	5	Bad checksum (bad serial link).
	6	Command rejected (illegal command).
	7	Unknown internal error.
I/O transfer:	The data to be written to Modbus registers for output. Data is written from Modbus registers to here for input.	
Com type:	You may select etherne or serial ports 1-4.	
IP address:	If using ethernet, select the Modbus unit's IP address.	
Port:	If using ethernet, select the Modbus unit's port number (usually 502).	
Unit address:	Select the slave address of the Modbus device.	
Register address:	Select the Modbus starting register address (starts at 1).	
Command:	Select the Modbus command. You may select Read coils (1), Read input discretes (2), Read registers (3), Read input registers (4), Set single coil (5), Set single register (6), Set multiple coils (15) and Set multiple registers (16).	

Register type:	Register types of float are 32 bits long and a maximum of 64 items can be transfered in one instruction. Register types of int are 16 bits long and a maximum of 128 items can be transferred in one instruction. The register types are defined as follows: • Float. • Unsigned int: Unsigned integer (0 to 65535) • 2s comp int: Two's compliment signed integer (-32768 to 32767) • Int and sign: 15 bits used for value. 16th bit indicates sign (0 positive, 1 negative) • BCD int: Unsigned Binary Coded Decimal (BCD) integer (0-9999) • Float (swapped).
Timeout:	Enter the timeout parameter in 10 millisecond steps. 100 would set 1000 milliseconds (1 second).
Retries:	Specify the number of retries before the status variable is updated with a failure.

Table Parameters

The table length indicates the number of items to transfer. The A and B coefficients convert the data according to an Ax + B conversion. Set A to 1 and B to 0 to perform no data conversion. Set A to -1 and B to 1 to invert logic.

Modslave

This instruction transfers Modus register values between the ICON Modbus slave registers and ICON variables. Modslave operation is configured under "System Parameter Config". Configure under "HMI edit/Serial ports" section if doing serial Modbus and under "Internet connections" if doing TCP Modbus.

General Parameters

Enable:	If Null the instruction is always enabled. Otherwise the variable value must be 1 for the instruction to execute.
I/O transfer:	Transfer data between the ICON Modbus slave registers and the ICON variable specified here.
Com type:	Select between Ethernet, Serial 1, Serial 2, Serial 3 or Serial 4.
Register address:	Select the Modbus slave starting register address. This can be 1-2048 for ethernet and 1-1024 for the serial ports.

Register type:	IRegister types of float are 32 bits long and a maximum of 64 items can be transfered in one instruction. Register types of int are 16 bits long and a maximum of 128 items can be transferred in one instruction. The register types are defined as follows: • Float. • Unsigned int: Unsigned integer (0 to 65535) • 2s comp int: Two's compliment signed integer (-32768 to 32767) • Int and sign: 15 bits used for value. 16th bit indicates sign (0 positive, 1 negative) • BCD int: Unsigned Binary Coded Decimal (BCD) integer (0-9999) Float (swapped).
Block length:	Select the number of variables to transfer. You can't transfer beyond the last register. So the maximum length for ethernet, if you start with register 1, is 2048. And the maximum length for a serial port, if you start with register 1, is 1024.
Direction:	You may select between "Read from Modbus registers" or "Write to Modbus registers".

Onoff

This instruction sets the result to 0 if the input is zero and 1 if the input is non-zero. Minimum delay times as described below must be honored. Enter times in number of passes. So if off time is set to 10, the output control must be off a minimum of 10 loop passes before it can change state. If the instruction is in a loop that executes every second, the times will represent seconds.

General Parameters

Input:	The value on this variable controls the result.
Result:	0 if input is 0 else 1.
Off time:	When the result is 0 it will remain 0 for this minimum time (number of loop passes) even if the input desires otherwise.
On time:	When the result is 1 it will remain 1 for this minimum time (number of loop passes) even if the input desires otherwise.
Off to on delay:	When the input desires to change result to 1, the result transition will delay for this time (number of loop passes).
On to off delay:	When the input desires to change result to 0, the result transition will delay for this time (number of loop passes).

OWconfig

This instruction sets an array of variables for configuring the ICON Modbus to 1-wire interface I/O system. Each ICON board can have a maximum of 64 1-wire Inputs/Outputs. Each ICON board can have a maximum of 16 1-wire devices set per Modbus transaction. Therefore you must have one OWconfig and one Modbus instruction for each 16 1-wire devices configured.

General Parameters

Config	The configuration variable array. There will be four variables in the array created for each 1-wire device. Therefore the array will be four to 64 variables long for a table length of one to 16. In other words, all five table parameters on one row are encoded into four variables.

Table Parameters

Description	Enter a descriptions of this device.
Device type	Enter the 1-wire device code that is attached to your 1-wire I/O module.
ID high	Enter the 1-wire device ID high value that is attached to your 1-wire I/O module.
ID low	Enter the 1-wire device ID low value that is attached to your 1-wire I/O module.
Bus	Use the selection set to select bus 1-4. Older ICON boards have four separate 1-wire busses and new ICON boards have only one bus.
Mode	If you have connected a separate +5V power supply to your 1-wire device select "Separate power". If you have only connected the device to the 1-wire signal and common, select "1-wire powered". The 1-wire devices can be scanned much quicker if you use separate power but you must also run additional wires. Some 1-wire devices such as the OWDO can only function with separate power.
Read delay uS	Select your desired delay. Usually use 15. On longer lines a larger value may need to be selected to eliminate errors.
Retries	Select them number of retries before an error is returned in the status variable.

Pid

This instruction executes the Proportional, Integral, Differential (PID) process control algorithm. An Error input is calculated as the difference between the setpoint (the desired value) and the actual instrument reading of the parameter to be controlled (feedback element). For example, if you are controlling the temperature of a greenhouse and the desired temperature is 75.0 and the actual temperature is 69.5 the Error is 6.5. This must be calculated in an expression instruction prior to executing the PID instruction. A "Control result" is generated according to the equation:

Control_result=Constant+Control_result[1]+Control_result[2]+Control_result[3] //The result of the PID
Control_result[1]=Error*Proportional //The result of the proportional term
Control_result[2]=(Error-Error_last_pass)*Differential //The result of the differential term
Control_result[3]=Control_result[3]+(Error*Integral) //The result of the integral calculation

Notice that this instruction returns the entire result of the PID in Control_result but the proportional only, differential only and integral only results are also returned in higher elements of the Control_result subscripted array. This can be handy for debugging and HMI display.

General Parameters

Error input:	The error value is obtained from this variable.
Control result:	The PID control result is updated here. Four values are calculated as described above.
Enable:	If Null, the control is always executed. If Enable=0 the PID is not executed, if Enable=1 the PID is executed and if Enable=2 the integral term is reset to 0 and the PID is executed. Note, you may also explicitly set the integral term to 0 (or any other value for that matter) by changing the value of Control_result[3]. This can be done with an expression instruction or from the HMI.
Tuning:	Not implemented at present. In the future one of several auto tuning algorithms may be selected.
Constant:	The control offset constant.
Proportional:	The control proportional constant.
Differential:	The control differential constant.
Integral:	The control integral constant.
Control interval:	Set to the time interval desired for execution. For instance, if set to 00:00:00:10 (10 seconds) the PID will execute every 10 seconds even if in a 1 second loop. Set to 0 to execute every loop scan. If the instruction is in a loop that scans faster than once per second the PID is updated every scan and this setting is ignored.

Pulse

Pulse maintains the Idle state value until the control changes value. When the Control changes value, the Pulse takes on the value of Control for the Duration time (number of loop passes) and then switches back to Idle state.

This instruction can be used to generate a pulse that starts on the changing edge of control. It can also be used in conjunction with write variable of an HMI object to generate a pulse. For this case Control and Pulse can point to the same variable. The HMI can write one value to this variable by the user at runtime. Duration time later the variable is set back to it’s idle state value. In this manner a pulse can be generated by one user HMI operation.

General Parameters

Control:	Starts a pulse.
Pulse:	The pulse value.
Duration:	The pulse duration in number of loop passes.
Idle state:	Pulse idle value.

Pwm

This instruction pulse width modulates the control result. If the frame time is 60 and the input is 0.5, the control result will be 0 for 30 seconds and 1 for 30 seconds if in a loop that executes every second.

General Parameters

Control input:	This variable value is expected to vary from 0 to 1.
Control result:	This variable value will pulse between 0 and 1. As the input approaches 1 the result is 1 for longer periods.
Frame time:	The pulse width modulation period in number of loop passes.
Minimum time:	This is the minimum time the result is in a 0 or 1 state in number of loop passes. (This is used to prevent a control from coming on for a very short period of time.)

Statistics

During each pass of the loop a new value from the "Input" variable is used for statistical calculations. Before each value is used in the statistical calculation the input status "Input[1]" is checked against "Max valid status" If the status is less than or equal to the "Max valid status" the value is used in the calculation and the "OR" of all "valid" status bits is made to a "valid" status temporary variable.

If the status is greater than "Max valid status" the value is not used in the calculation and the "OR" of all "invalid" status bits is made to an "invalid" status temporary variable. The result of the calculation is posted to the "Result[even]" index and the status to the "Result[odd]" index. If there at least "Minimum valid" number of data points the "valid" status value is set in "Result[odd]". If there are less than "Minimum valid" number of data points the "invalid" status value is set in "Result[odd]".

General Parameters

Input:	Points to a variable with two values. The first value is the actual data and the second value is interpreted as status.
Results:	The results (in pairs) of the circular buffer calculations. The calculation result is in the lower (even) numbered index and the status is in the upper (odd) index. If there are no valid readings in the circular buffer the result is not updated (but of coarse the status is).
Control:	If value is 0 do nothing, 1 enables the update of the statistical calculation with the results posted, 2 enables the update of the statistical calculation but results are not posted on the result variable and 3 performs the same functions as 1 and also resets all statistical calculations to the initialization state.
Max valid status:	Enter the highest value for a status that is considered valid. For instance, if any status bits 0, 1, 2 or 3 when set still indicate valid status enter a value of 15 (1111 binary).

Minimum valid:	Minimum number of valid readings required before posting a new result value.

Table Parameters

Calc type:	This selection set allows you to pick the type of statistical calculation. Choices include average, standard deviation, maximum, minimum, total, difference, circular average (wind direction) and circular standard deviation (wind direction sigma theta).

Timepos

This instruction controls an open and a close relay to position a vent, valve or similar control between full closed and full open by pulsing the appropriate relay for the appropriate time.

General Parameters

Control input:	Expects a variable whose value varies from 0 to 1 to control a position between closed and full open.
Control result:	Two values are placed here. The value at the lower index controls the close relay. The value at the higher index controls the open relay. Each value can be 0 (relay open) or 1 (relay closed which causes the control to move). When the Control input is 0 the close relay is always activated to ensure the control is closed and remains closed. At all other times the open and close relays are open (deactivated) except for short periods of close (activate) to reposition the control.
Open time:	Enter the time it takes for the control to go from full closed to full open in loop passes. If it takes 1 minute for a vent to go from closed to open enter 60 if in a loop that executes once a second.
Error time:	Enter the minimum relay pulse time in loop passes. This time can be set to prevent a control relay from activating for less than a desired minimum time. This minimizes relay chatter and constant position hunting. If 3 is entered the open or close relay will not activate unless it will stay activated for a minimum of 3 seconds (if executing in a loop with a 1 second scan time).

Timer

This instruction can be configured as an up or a down counter to count loop passes. If counting up it will stop at 16,777,215 and if counting down it will stop at 0.

General Parameters

Control:	If this variable is set to "Null" the counter is always enabled to count. Otherwise, if the value is 0 the "Time" is set to 0. If the value is 1 "Time" counts. If the value is 2 "Time" is set to 16,777,215 and if the value is 3 the "Time" value is frozen at its current state.
Time:	The timer value in loop passes. If in a loop that executes once per second

	this value increments or decrements by seconds. You may write a value to this variable from another instruction in your program and/or the HMI if desired.
Time offset:	Adjust the reset time by this offset. (Can only used in loops of 1 second or slower.)
Time interval:	If interval is non-zero the timer is reset whenever the ((Second_of Century - Time_offset)/Time_interval) has no remainder. If set as an up counter the timer is reset to 0. If set as a down counter the timer is reset to the "Time interval" value-1. (Can only used in loops of 1 second or slower.)
Type:	Select as an up or down counter.

Translate

This instruction allows the conversion between a counting number 0, 1, 2 … and re-maps to a random sequence. Or the opposite can take place.

General Parameters

Input:	The input value to perform the conversion on.
Result:	The resultant converted value.
Direction:	If "Count to random" is selected, the value of "Input" is used as an index to select the corresponding table value. This value is then returned in "Result". If "Random to count" is selected, the value of "Input" is used to locate a match in the table. The index for this match value is then returned in "Result".

Table Parameters

Random match:	The list of table values

For example, if you would like the input counting sequence 0, 1, 2, 3 and 4 to generate the result random pattern of 0, 10h, 30h, 50h and 70h, set the input variable to the variable with the counting sequence, the result variable to the variable to accept the random lookup and set direction to "Count to random". Place the random sequence of values into the table.

But if you would like the input random sequence of 0, 10h, 30h, 50h and 70h to generate the output counting sequence of 0, 1, 2, 3 and 4, set the input variable to the variable with the random sequence, the result variable to the variable to accept the counting sequence and select the direction to "Random to count". As before, place the random sequence of values into the table.

If the direction is set to "Count to random" and the input index is greater than the number of table rows, the value 0 is placed in result. If the direction is set to "Random to count" and a match is not found, a count (index) equal to the number of rows is placed in result.

X10

This instruction works with the PowerLinc X10 interface. Order part numbers 1132 and 1132S from www.smarthome.com. With this interface plugged into a 110 VAC outlet you can control

X10 lamp, appliance and relay modules. For more information on X10 visit www.smarthome.com or www.X10.com. X10 has become the defacto standard for controlling lighting in homes.

The X10 PowerLinc is bi-directional. The X10 command cause an X10 command to be transmitted over the powerline to control a device. Other X10 controllers such as switches, manual controllers, wireless remotes and motion detectors can also generate X10 commands. The ICON will monitor these external signals and set the state of 256 variables (!System[24] to !System[279]) to either 0 or 1 in response to the X10 off and on commands on the 256 different house/unit code combinations.

General Parameters

Enable:	If null then the command is executed on every loop pass. Otherwise a value of 0 disables, 1 causes the instruction to execute on the "Repeat time" interval and 2 causes execution every pass.	
Command:	The value of this variable defines the X10 command as defined below:	
	Command	Description
	0	Off
	1	On
	2	Dim
	3	Bright
	4	All lights off
	5	All lights on
	6	All units off
Com port:	Use X10 on serial port 1 only at 9600 baud.	
House code:	Select the house code A-P.	
Unit code:	Select the unit code 1-16.	
Repeat time:	The repeat time sets how often to re-issue the command if the value of "Enable" is 1. Because X10 can be so unreliable it is often useful to keep re-issuing the command periodically. This insures that the device finally gets the command and goes to the proper state. Sometimes noise on the power line causes a device to go to a wrong state. By re-issuing the command periodically the device can be put back in the correct state. There is no support for 2-way X10 devices since the command can be re-issued.	