Proficy* Logic Developer - PLC

Function Block Diagram (FBD) Language

Proficy Machine Edition 8.0

2013

Notice

GE Intelligent Platforms, Inc. reserves the right to make improvements to the products described in this publication at any time and without notice.

We want to hear from you. If you have any comments, questions, or suggestions about our documentation, send them to the following email address:
doc@ge.com

Disclaimer of Warranties and Liability

The information contained in this manual is believed to be accurate and reliable. However, GE Intelligent Platforms, Inc. assumes no responsibilities for any errors, omissions or inaccuracies whatsoever. Without limiting the foregoing, GE Intelligent Platforms, Inc. disclaims any and all warranties, expressed or implied, including the warranty of merchantability and fitness for a particular purpose, with respect to the information contained in this manual and the equipment or software described herein. The entire risk as to the quality and performance of such information, equipment and software, is upon the buyer or user. GE Intelligent Platforms, Inc. shall not be liable for any damages, including special or consequential damages, arising out of the use of such information, equipment and software, even if GE Intelligent Platforms, Inc. has been advised in advance of the possibility of such damages. The use of the information contained in the manual and the software described herein is subject to GE Intelligent Platforms, Inc. standard license agreement, which must be executed by the buyer or user before the use of such information, equipment or software.

Table Of Contents

FBD Instructions

(For PACSystems firmware version 3.50 and later.)
FBD logic instructions are what FBD blocks are composed of. To create executable units of logic, you insert the instructions and their operands into an FBD block. Each instruction performs an operation on variables defined for the target the FBD logic is associated with.
You can also add one or more Comment Blocks (TEXT boxes) to your FBD. TEXT boxes are for documentation only. They have no inputs and no outputs.

Notes

- All available FBD instructions are contained in the **FBD Instructions** drawer of the Toolchest. You can drag these instructions to any open area in the FBD editor; that is, where there is no existing TEXT box, instruction, or wire.
- In FBD, you cannot write a value to array parameters.
- The maximum size (bytes) of a compiled FBD block is 128 KB.
- You can have a maximum of 13 FBD editor windows open at one time.

In the Toolchest, FBD logic instructions are grouped according to the type of operation performed. The instruction groups are:

- Advanced Math
- Bit Operations
- Comment Block
- Comparison
- Control
- Counters
- Data Move
- Math
- PACMotion
- Program Flow
- Timers
- Type Conversion

PACMotion instructions are described at length in both GFK-2448 and online help.

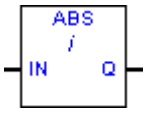
Communication instructions are described in the Addendum of this manual, after the index.

Advanced Math

(For PACSystems firmware version 3.50 and later.)
Advanced math instructions perform logarithmic, exponential, square root, trigonometric, and inverse trigonometric operations.

Instruction	*Mnemonic*	*Description*
Absolute value	ABS	Calculates the positive value of the IN operand
Exponential	EXP	Calculates the inverse natural logarithm of the IN operand ($\mathbf{e}^{IN}$)
	EXPT	Calculates IN1 to the IN2 power ($IN1^{IN2}$)
Inverse Trig	ACOS	Calculates the inverse cosine of the IN operand in radians
	ASIN	Calculates the inverse sine of the IN operand in radians
	ATAN	Calculates the inverse tangent of the IN operand in radians
Logarithmic	LN	Calculates the natural logarithm of the operand IN
	LOG	Calculates the base 10 logarithm of the operand IN
Square Root	SQRT	Calculates the square root of the operand IN
Trig	COS	Calculates the cosine of the operand IN in radians
	SIN	Calculates the sine of the operand IN in radians
	TAN	Calculates the tangent of the operand IN in radians

Absolute Value



Operation

When the FBD instruction receives data, it places the absolute (positive) value of input IN in output Q.

Operands

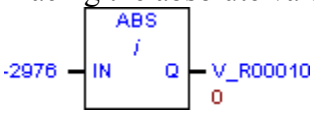
Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

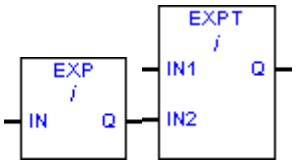
Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	INT, DINT, REAL, or LREAL variable or constant. Must be of the same data type as for Q.	data flow, I, Q, M, T, G, R, P, L, W, AI, AQ, symbolic, I/O variable	The value to process
Q	INT, DINT, REAL, or LREAL variable.		The positive value of IN

Example

Placing the absolute value of –2,976, which is 2,976, in %R00010:



Exponential



Operation

When an exponential instruction receives data, it performs the appropriate exponential operation on the REAL or LREAL input value(s) and places the result in output Q.

- For the inverse natural log (EXP) function, ***e*** is raised to the power specified by IN and the result is placed in Q.
- For the Power of X (EXPT) function, the value of input IN1 is raised to the power specified by the value IN2 and the result is placed in output Q.

The instruction is carried out unless at least one of the following invalid situations is encountered:

- There is overflow.
- For EXP, IN is negative infinity.
- For EXPT, IN1 is negative.

EXP Operands

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	REAL or LREAL variable or constant. Must be the same data type as for Q.	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The power to raise ***e*** to; the number to calculate the inverse natural log of. **Note:** When IN = -∞, EXP returns a value of 0, as expected, but does *not* complete the instruction.
Q	REAL or LREAL variable		e^{IN}

EXPT Operands

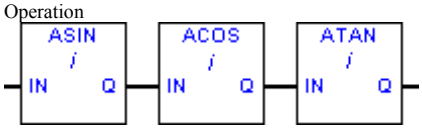
Operand	*Data Type*	*Memory Area*	*Description*

IN	REAL or LREAL variable or constant. All operands must be of the same data type.	R, P, L, AI, AQ, W, I, Q, M, T, G	The base value
IN2	REAL or LREAL variable or constant		The exponent
Q	REAL or LREAL variable		$IN1^{IN2}$

Inverse Trig

(PACSystems firmware version 3.50 or later; to process LREALs: 5.50 or later.)

Operation



Operation

When an Inverse Sine (ASIN), Inverse Cosine (ACOS), or Inverse Tangent (ATAN) instruction receives data, it computes respectively the inverse sine, inverse cosine or inverse tangent of IN and stores the result in radians in output Q. Both IN and Q are REAL or LREAL values; both IN and Q must be of the same data type.
The output Q is returned when the function is performed without overflow, unless an invalid operation occurs.

ASIN Operands

Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	REAL or LREAL variable or constant. Must be the same data type as for Q.	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	-1 ≤ IN ≤ 1. The value to process.
Q	REAL or LREAL variable		Inverse sine of IN. Expressed in radians. (-π/2) ≤ Q ≤ (À/2).

ACOS Operands

Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	REAL or LREAL variable or constant. Must be the same data type as for Q.	R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	-1 ≤ IN ≤ 1. The value to process.
Q	REAL or LREAL variable		Inverse cosine of IN. Expressed in radians. 0 ≤ Q ≤ À.

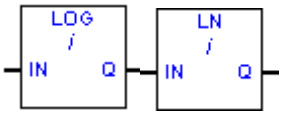
ATAN Operands

Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	REAL or LREAL variable or constant. Must be the same data type as for Q.	R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	-∞ ≤ IN ≤ +-. The value to process.
Q	REAL or LREAL variable		Inverse tangent of IN. Expressed in radians. (-π/2) ≤ Q ≤ (À/2).

Logarithmic



Operation

When a logarithmic instruction receives data, it performs the appropriate logarithmic operation on the REAL or LREAL value in input IN and places the result in output Q.

- For the Base 10 Logarithm (LOG) function, the base 10 logarithm of IN is placed in Q.
- For the Natural Logarithm (LN) function, the natural logarithm of IN is placed in Q.

The output Q is returned unless at least one of the following invalid situations is encountered:

- There is overflow.
- IN is negative.

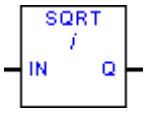
Operands

Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	REAL or LREAL variable or constant. Must be the same data type as for Q.	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The value to calculate the LOG or LN of.
Q	REAL or LREAL variable		LOG (IN) or LN (IN).

Square Root



Operation

When the Square Root instruction receives data, it finds the square root of IN and stores the result in Q. The output Q must be the same data type as IN.
The output Q is returned unless at least one of these invalid situations is encountered:

- There is overflow.
- IN is negative.

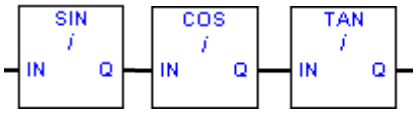
Operands

Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	LREAL, REAL, INT, or DINT variable or constant. Must be the same data type as for Q.	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The value to calculate the square root of. If IN < 0, the instruction will not execute.
Q	LREAL, REAL, INT, or DINT variable		The calculated square root.

Trig



Operation

The SIN, COS, and TAN functions are used to find the trigonometric sine, cosine, and tangent, respectively, of its input. When one of these functions receives an input, it computes the sine (or cosine or tangent) of IN and stores the result in output Q.
The output Q is returned when the function is performed without overflow, unless an invalid operation occurs.

Operands

Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

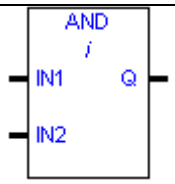
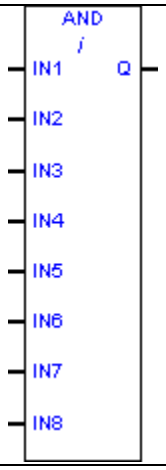
Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	REAL or LREAL variable or constant. Must be the same data type as for Q.	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	Number of radians. ($-9.22x10^{18} < IN < 9.22x10^{18}$)
Q	REAL or LREAL variable		Trigonometric value of IN

Bit Operations

(For PACSystems firmware version 3.50 and later.)
Bit operation instructions perform logical operations on bit strings, or bit rotations or bit shifts in bit strings.

Instruction	*Mnemonics*	*Description*
Logical AND	AND	Compares the bit strings IN1, IN2, IN3, ... , IN8 bit by bit. When two corresponding bits are 1, a 1 is placed in the corresponding location in output string Q; otherwise, a 0 is placed in the corresponding location in Q.
Logical NOT	NOT	Logical inverse. Sets the state of each bit in output bit string Q to the inverse state of the corresponding bit in bit string IN1.
Logical OR	OR	Compares the bit strings IN1, IN2, IN3, ... , IN8 bit by bit. When two corresponding bits are 0, a 0 is placed in the corresponding location in output string Q; otherwise, a 1 is placed in the corresponding location in Q.
Logical XOR	XOR	Compares the bit strings IN1, IN2, IN3, ... , IN8 bit by bit. When two corresponding bits are different, a 1 is placed in the corresponding location in the output bit string Q; when two corresponding bits are the same, a 0 is placed in Q.
Rotate Bits	ROL	Rotate Left. Rotates all bits in a string a specified number of places to the left.
	ROR	Rotate Right. Rotates all bits in a string a specified number of places to the right.
Shift Bits	SHIFTL	Shift Left. Shifts all bits in a word or string of words to the left by a specified number of places.
	SHIFTR	Shift Right. Shifts all bits in a word or string of words to the right by a specified number of places.

Logical AND

Minimum number of inputs: 2	Maximum number of inputs: 8
	

Operation

When the Logical AND instruction is executed, it examines each bit in bit string IN1 and the corresponding bit in bit string IN2, beginning with the least significant bit in each.
If the values of both bits are 1, AND places a 1 in the corresponding location in output string Q. If either bit is 0 or both bits are 0, AND places a 0 in string Q in that location.
If there are more inputs (IN3 and so on), AND examines each bit in bit string IN3 and the corresponding bit in bit string Q, beginning with the least significant bit in each.
If the values of both bits are 1, AND places a 1 in the corresponding location in output string Q. If the value of either bit is 0 or the values of both bits are 0, AND places a 0 in string Q in that location.
This process is repeated until all inputs (IN1 through IN8) have been examined.

Tips

- You can use Logical AND to build masks or screens, where only certain bits are passed (the bits opposite a 1 in the mask), and the values of all other bits are set to 0.
- You can also use Logical AND to clear an area of WORD or DWORD memory by ANDing the bits with another bit string known to contain all 0s. The IN1, IN2, IN3, ... , IN8 bit strings specified can overlap.

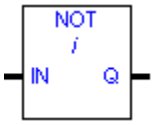
Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		

IN1, IN2, IN3, ... , IN8 (All inputs must be the same data type.)	BOOL, BYTE, or WORD variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The values to AND: Minimum: 2 inputs. Maximum: 8 inputs.
	DWORD variable or constant	data flow, R, P, L, AI, AQ, I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable	
Q (Must be the same data type as the inputs.)	BOOL, BYTE, or WORD variable	data flow, I, Q, M, T, SA, SB, SC, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The result of the AND
	DWORD variable	data flow, R, P, L, AI, AQ, I, Q, M, T, G, SA, SB, SC, W, symbolic, I/O variable	

Logical NOT



Operation

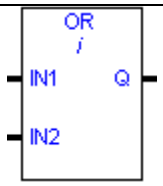
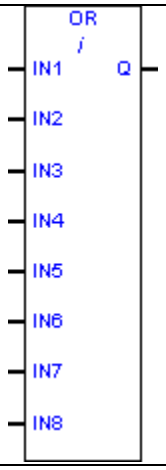
When the Logical Not or Logical Invert (NOT) instruction receives input, it sets the state of each bit in the output bit string Q to the opposite of the state of the corresponding bit in bit string IN.
All bits are altered each time the instruction is executed, making output string Q the logical complement of IN. Logical NOT passes data to the right whenever it is executed.

Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	BOOL, BYTE, or WORD variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The constant to NOT, or the reference for the first WORD or DWORD of the string to NOT
	DWORD variable or constant	data flow, R, P, L, AI, AQ, I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable	
Q (Must be the same data type as IN.)	BOOL, BYTE, or WORD variable	data flow, I, Q, M, T, SA, SB, SC, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The reference for the first WORD or DWORD of the NOT's result
	DWORD variable	data flow, R, P, L, AI, AQ, I, Q, M, T, G, SA, SB, SC, W, symbolic, I/O variable	

Logical OR

Minimum number of inputs: 2	Maximum number of inputs: 8
	

Operation

When it is executed, the Logical OR instruction examines each bit in bit string IN1 and the corresponding bit in bit string IN2, beginning with the least significant bit in each. If the value of at least one bit is 1, OR places a 1 in the corresponding location in output string Q. If the values of both bits are 0, OR places a 0 in string Q in that location. The instruction passes data to the right whenever it is executed.
If there are more inputs (IN3 and so on), OR examines each bit in bit string IN3 and the corresponding bit in bit string Q, beginning with the least significant bit in each. If the value of at least one bit is 1, OR places a 1 in the corresponding location in output string Q. If the values of both bits are 0, OR places a 0 in string Q in that location. The instruction passes data to the right whenever it is executed.
This process is repeated (IN4 and so on) until all inputs have been examined.

Tips

- You can use OR to combine strings or to control many outputs with one simple logical structure. OR is the equivalent of two relay contacts in parallel multiplied by the number of bits in the string.
- You can use OR to drive indicator lamps directly from input states, or to superimpose blinking conditions on status lights.

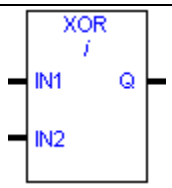
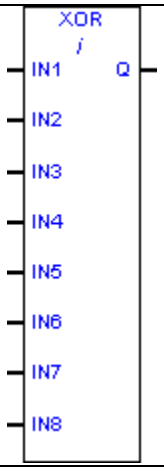
Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		

IN1, IN2, IN3, ... , IN8 (All inputs must be the same data type.)	BOOL, BYTE, or WORD variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The values to OR: Minimum: 2 inputs. Maximum: 8 inputs.
	DWORD variable or constant	data flow, R, P, L, AI, AQ, I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable	
Q (Must be the same data type as the inputs.)	BOOL, BYTE, or WORD variable	data flow, I, Q, M, T, SA, SB, SC, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The result of the OR
	DWORD variable	data flow, R, P, L, AI, AQ, I, Q, M, T, G, SA, SB, SC, W, symbolic, I/O variable	

Logical XOR

Minimum number of inputs: 2	Maximum number of inputs: 8
	

Operation

When the Exclusive OR (XOR) instruction receives data, it examines each bit in bit string IN1 with the corresponding bit in string IN2. If the bits are different, a 1 is placed in the corresponding position in the output bit string.
When it is executed, XOR examines each bit in string IN1 and the corresponding bit in string IN2, beginning with the least significant bit in each.
For each pair of bits examined, if the value of only one bit is 1, then XOR places a 1 in the corresponding location in bit string Q. XOR passes output to the right whenever it is executed.
If there are more inputs (IN3 and so on), XOR examines each bit in bit string IN3 and the corresponding bit in bit string Q, beginning with the least significant bit in each.
For each pair of bits examined, if the value of only one bit is 1, then XOR places a 1 in the corresponding location in bit string Q.
This process is repeated until all inputs (IN4 and so on) have been examined.

Tips

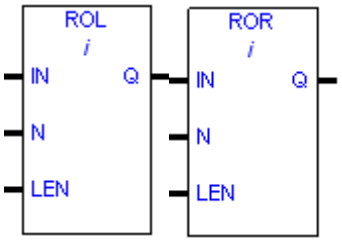
- If an input string IN2, IN3, IN4, ... , or IN8 and output string Q begin at the same reference, a 1 placed in the current input string IN1, IN2, IN3, ... , IN8 causes the corresponding bit in string Q to alternate between 0 and 1, changing state with each execution as long as input is received.
- You can program longer cycles by pulsing the input to the instruction at twice the desired rate of flashing. The execution time should be one scan long (oneshot type coil or selfresetting timer).
- You can use XOR to quickly compare two bit strings, or to blink a group of bits at the rate of one ON state per two instructions.
- XOR is useful for transparency masks.

Operands

Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN1, IN2, IN3, ... , IN8 (All inputs must be the same data type.)	BOOL, BYTE, or WORD variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The values to XOR: Minimum: 2 inputs. Maximum: 8 inputs.
	DWORD variable or constant	data flow,R, P, L, AI, AQ, I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable	
Q (Must be the same data type as the inputs.)	BOOL, BYTE, or WORD variable	data flow, I, Q, M, T, SA, SB, SC, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The result of the XOR
	DWORD variable	data flow, R, P, L, AI, AQ, I, Q, M, T, G, SA, SB, SC, W, symbolic, I/O variable	

Rotate Bits



Operation

When receiving data, the Rotate Bits Right (ROR_DWORD and ROR_WORD) and Rotate Bits Left (ROL_DWORD and ROL_WORD) functions rotate all the bits in a string of WORDs or DWORDs N positions respectively to the right or to the left. When rotation occurs, the specified number of bits is rotated out of the input string respectively to the right or to the left and back into the string on the other side. The result is placed in output string Q. If you want the input string to be rotated, the output parameter Q must use the same memory location as the input parameter IN. The entire rotated string is written each time the instruction is executed.
A string length of 1 to 256 words or double words can be selected for either function. The Rotate Bits functions passes data to the output Q when the number of bits to rotate is greater than or equal to zero, and is less than or equal to the total length of the string.

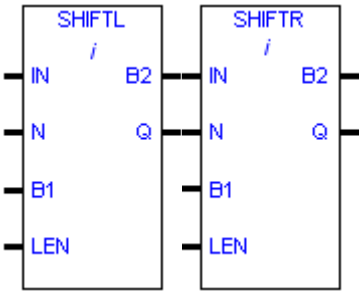
Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	WORD variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The first WORD or DWORD in the string to rotate
	DWORD variable or constant	data flow, R, P, L, AI, AQ, I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable	
N	INT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The number of positions to rotate. 0 ≤ N ≤ (number of bits in the string).
LEN	Constant		The number of WORDs or DWORDs in the string to rotate. 1 ≤ LEN ≤ 256.

Q	DWORD variable	data flow, R, P, L, AI, AQ, I, Q, M, T, G, SA, SB, SC, W, symbolic, I/O variable	The first WORD or DWORD of the rotated string
	WORD variable	data flow, I, Q, M, T, SA, SB, SC, G, R, P, L, AI, AQ, W, symbolic, I/O variable	

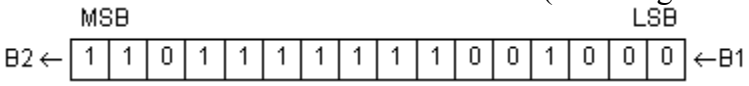
Shift Bits



Operation

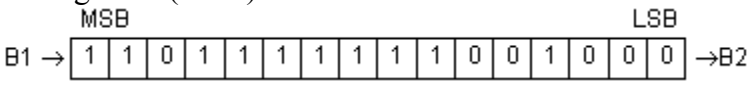
Shift Left

When the Shift Left (SHIFTL_WORD) instruction receives data, it shifts all the bits in a word or group of words to the left by a specified number of places. When the shift occurs, the specified number of bits is shifted out of the output string to the left. As bits are shifted out of the high end of the string (Most Significant Bit (MSB)), the same number of bits is shifted in at the low end (Least Significant Bit (LSB)).



Shift Right

When the Shift Right (SHIFTR_WORD) instruction receives data, it shifts all the bits in a word or group of words a specified number of places to the right. When the shift occurs, the specified number of bits is shifted out of the output string to the right. As bits are shifted out of the low end of the string (LSB), the same number of bits is shifted in at the high end (MSB).



Shift Left and Shift Right

A string length of 1 to 256 words can be selected for either function.
The number of places specified for the shift (N) must be more than zero and less than the number of bits in the string.
If N is out of range, no shift occurs and no output is generated.
The bits being shifted into the beginning of the string are specified via input parameter B1. If a length greater than 1 has been specified as the number of bits to shift, each bit is filled with the same value (0 or 1). This can be:

- The boolean output of another program function.
- All 1s. To do this, use the #ALW_ON (always on) system bit (in memory location %S7), as a permissive to input B1.

- All 0s. To do this, use the #ALW_OFF (always off) system bit (in memory location %S8), as a permissive to input B1.

The Shift Bits function passes data to the output Q, unless the number of bits specified to shift is zero or is greater than the array size.
Output Q is the shifted copy of the input string. If you want the input string to be shifted, the output parameter Q must use the same memory location as the input parameter IN. The entire shifted string is written time the instruction is executed. Output B2 is the last bit shifted out. For example, if four bits were shifted, B2 would be the fourth bit shifted out.

Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	WORD variable or constant.	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The first WORD or DWORD in a string of WORDs or DWORDs to shift
	DWORD variable or constant	data flow, R, P, L, AI, AQ, I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable	
N	INT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The number of places (bits) to shift the array. 0<N<(number of bits in the string=(16*??)). If N is out of range, the results may differ.
B1	BOOL Variable	I, Q, G, M, T, S, SA, SB, SC	The bit value to shift into the array
	Bit reference in a non-BOOL variable	data flow, I, Q, M, T, G, S, SA, SB, SC, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	
LEN	Constant		The number of WORDs or DWORDs in the string. 1 ≤ LEN ≤ 256.
B2	BOOL Variable	I, Q, G, M, T, SA, SB, SC	(Optional.) The bit value of the last bit shifted out of the array.
	Bit reference in a non-BOOL variable	data flow, I, Q, M, T, G, SA, SB, SC, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	

Q (Must be the same data type as IN.)	WORD variable	data flow, I, Q, M, T, G, SA, SB, SC, R, AI, AQ, W, symbolic, I/O variable	The first WORD or DWORD of the shifted array
	DWORD variable	data flow, R, P, L, AI, AQ, I, Q, M, T, G, SA, SB, SC, W, symbolic, I/O variable	

Comment Block

(For PACSystems firmware version 3.50 and later.)
You can add one or more text blocks to your FBD. Text blocks are for documentation only. They have no inputs and no outputs.

Instruction	*Mnemonics*	*Description*
Text	n/a	Enter any text for documentation.

Text

Operation

The content of the TEXT object will be automatically truncated to fit its size. To display more text, increase the size of the text object by selecting and dragging one of the handles.

Operands

There are no operands for the TEXT object.

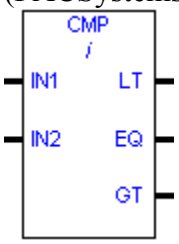
Comparison

(For PACSystems firmware version 3.50 and later.)
Comparison instructions compare two values of the same data type or determine whether a number lies within a specified range. Original values are unaffected.

Instruction	*Mnemonic*	*Description*
Compare	CMP	Compares two numbers of the same data type, IN1 and IN2. ▪ If IN1 < IN2, the LT output is set to ON. ▪ If IN1 = IN2, the EQ output is set to ON. ▪ If IN1 > IN2, the GT output is set to ON.
Equal	EQ	Tests two numbers, IN1 and IN2, for equality
Greater or Equal	GE	Tests whether one number, IN1, is greater than or equal to another number, IN2
Greater Than	GT	Tests whether one number, IN1, is greater than another number, IN2
Less or Equal	LE	Tests whether one number, IN1, is less than or equal to another number, IN2
Less Than	LT	Tests whether one number, IN1, is less than another number, IN2
Not Equal	NE	Tests two numbers, IN1 and IN2, for inequality
Range	RANGE	Tests whether one number, IN, is within the range defined by two other numbers, L1 and L2

Compare

(PACSystems firmware version 3.50 or later; to compare LREALs: 5.50 or later.)



Operation

When the Compare (CMP) instruction receives data, it compares the value IN1 to the value IN2.

- If IN1 < IN2, CMP passes TRUE to the LT (Less Than) output.
- If IN1 = IN2, CMP passes TRUE to the EQ (Equal) output.
- If IN1 > IN2, CMP passes TRUE to the GT (Greater Than) output.

IN1 and IN2 must be the same data type. CMP compares data of the following types: DINT, INT, LREAL, REAL, and UINT.

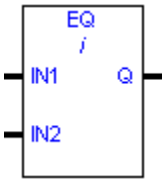
Tip: To compare values of different data types, first use conversion functions to make the types the same.

Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN, IN2	DINT, INT, LREAL, REAL, or UINT variable or constant	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The two values to compare.
LT	BOOL Variable	I, Q, G, M, T, SA, SB, SC	Output LT is TRUE when IN1 < IN2.
	Bit reference in a non-BOOL variable	I, Q, M, T, G, SA, SB, SC, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	
EQ	BOOL Variable	I, Q, G, M, T, SA, SB, SC	Output EQ is TRUE when IN1 = IN2.
	Bit reference in a non-BOOL variable	I, Q, M, T, G, SA, SB, SC, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	
GT	BOOL Variable	I, Q, G, M, T, SA, SB, SC	Output GT is TRUE when IN1 > IN2.
	Bit reference in a non-BOOL variable	I, Q, M, T, G, SA, SB, SC, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	

Equal



Operation

When the instruction receives data, it compares input IN1 to input IN2. These operands must be the same data type. If inputs IN1 and IN2 are equal, the instruction sets the output Q to TRUE.

Tip: To compare values of different data types, first use conversion functions to make the types the same. The EQ function requires data to be one of the following types: DINT, INT, UINT, LREAL, REAL, STRUCTURE, or enumerated data type.

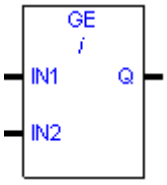
Operands

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN1, IN2	DINT, INT, UINT, LREAL, REAL, STRUCTURE, or enumerated variable or constant	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The values to be compared in the relational equation IN1 = IN2. IN1 and IN2 must be of the same data type.
Q	BOOL Variable	I, Q, G, M, T, SA, SB, SC	The output. If IN1 = IN2, then Q is TRUE.
	Bit reference in a non-BOOL variable	I, Q, M, T, G, SA, SB, SC, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	

Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- For STRUCTURES, all elements must be in either discrete or analog memory; a mix is not allowed. TON, TOF, TP, and custom structures are not supported by EQ.

Greater or Equal



Operation

When the instruction receives data, it compares input IN1 to input IN2. These operands must be the same data type. If IN1 ≥ IN2, the sets the output Q to TRUE.

Tip: To compare values of different data types, first use conversion functions to make the types the same. The GE function requires data to be one of the following types: DINT, INT, LREAL, REAL, or UINT.

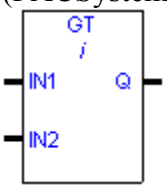
Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN, IN2	DINT, INT, LREAL, REAL, or UINT variable or constant	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The values to be compared in the relational equation IN1 ≥ IN2. IN1 and IN2 must be the same data type.
Q	BOOL Variable	I, Q, G, M, T, SA, SB, SC	The output. If IN1 ≥ IN2, then Q is TRUE.
	Bit reference in a non-BOOL variable	I, Q, M, T, G, SA, SB, SC, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	

Greater Than

(PACSystems firmware version 3.50 or later; to compare LREALs: 5.50 or later.)



Operation

When the instruction receives data, it compares input IN1 to input IN2. These operands must be the same data type. If IN1 > IN2, the instruction sets the output Q to TRUE.

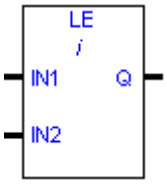
Tip: To compare values of different data types, first use conversion functions to make the types the same. The GT function requires data to be one of the following types: DINT, INT, LREAL, REAL, or UINT.

Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN, IN2	DINT, INT, LREAL, REAL, or UINT variable or constant	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The values to be compared in the relational equation IN1 > IN2. IN1 and IN2 must be the same data type.
Q	BOOL Variable	I, Q, G, M, T, SA, SB, SC	The output. If IN1 > IN2, then Q is TRUE.
	Bit reference in a non-BOOL variable	I, Q, M, T, G, SA, SB, SC, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	

Less or Equal



Operation

When the instruction receives data, it compares input IN1 to input IN2. These operands must be the same data type. If IN1 ≤ IN2, the instruction sets the output Q to TRUE.

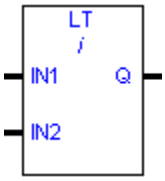
Tip: To compare values of different data types, first use conversion functions to make the types the same. The LE function requires data to be one of the following types: DINT, INT, LREAL, REAL, or UINT.

Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN1, IN2	DINT, INT, LREAL, REAL, or UINT variable or constant	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The values to be compared in the relational equation IN1 ≤ IN2. IN1 and IN2 must be the same data type.
Q	BOOL Variable	I, Q, G, M, T, SA, SB, SC	The output. If IN1 ≤ IN2, then Q is TRUE.
	Bit reference in a non-BOOL variable	I, Q, M, T, G, SA, SB, SC, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	

Less Than



Operation

When the instruction receives data, it compares input IN1 to input IN2. These operands must be the same data type. If IN1 < IN2, the instruction sets the output Q to TRUE.

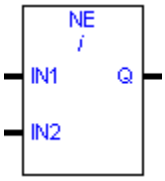
Tip: To compare values of different data types, first use conversion functions to make the types the same. The LT function requires data to be one of the following types: DINT, INT, LREAL, REAL, or UINT.

Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN1, IN2	DINT, INT, LREAL, REAL, or UINT variable or constant	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The values to be compared in the relational equation IN1 < IN2. IN1 and IN2 must be the same data type.
Q	BOOL Variable	I, Q, G, M, T, SA, SB, SC	The output. If IN1 < IN2, then Q is TRUE.
	Bit reference in a non-BOOL variable	I, Q, M, T, G, SA, SB, SC, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	

Not Equal



Operation

When the instruction receives data, it compares input IN1 to input IN2. These operands must be the same data type. If IN1 ≠ IN2, the instruction sets the output Q to TRUE.

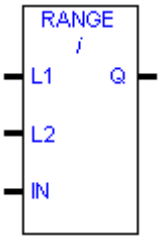
Tip: To compare values of different data types, first use conversion functions to make the types the same. The NE function requires data to be one of the following types: DINT, INT, LREAL, REAL, or UINT.

Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN1, IN2	DINT, INT, LREAL, REAL, or UINT variable or constant	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The values to be compared in the relational equation IN1 ≠ IN2. IN1 and IN2 must be of the same data type.
Q	BOOL Variable	I, Q, G, M, T, SA, SB, SC	The output. If IN1 ≠ IN2, then Q is TRUE.
	Bit reference in a non-BOOL variable	I, Q, M, T, G, SA, SB, SC, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	

Range



Operation

When the Range function is enabled, it compares the value of input IN against the range delimited by operands L1 and L2. Either L1 or L2 can be the high or low limit. When L1 ≤ IN ≤ L2 or L2 ≤ IN ≤ L1, output parameter Q is set to TRUE (1). Otherwise, Q is set FALSE (0).

Operands

Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	DINT variable or constant	data flow, R, P, L, AI, AQ, I, Q, M, T, G, W, symbolic, I/O variable	The value to compare against the range delimited by L1 and L2. Must be of the same data type as L1 and L2.
	DWORD variable or constant	data flow, R, P, L, AI, AQ, I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable	
	INT, UINT, or WORD variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	
L1	DINT variable or constant	data flow, R, P, L, AI, AQ, I, Q, M, T, G, W, symbolic, I/O variable	The start point of the range. May be the upper limit or the lower limit. Must be of the same data type as IN and L2.
	DWORD variable or constant	data flow, R, P, L, AI, AQ, I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable	

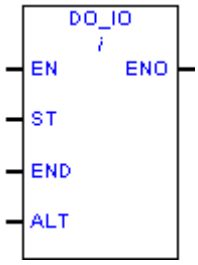
	INT, UINT, or WORD variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	
L2	DINT variable or constant	data flow, R, P, L, AI, AQ, I, Q, M, T, G, W, symbolic, I/O variable	The end point of the range. May be the lower or upper limit. Must be of the same data type as IN and L1.
	DWORD variable or constant	data flow, R, P, L, AI, AQ, I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable	
	INT, UINT, or WORD variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	
Q	BOOL Variable	I, Q, G, M, T, SA, SB, SC	if L1 ≤ IN ≤ L2 or L2 ≤ IN ≤ L1, then Q isTRUE.
	Bit reference in a non-BOOL variable	data flow, I, Q, M, T, G, SA, SB, SC, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	

Control

(For PACSystems firmware version 3.50 and later.)
Control instructions limit program execution and change the way the CPU executes the application program.

Instruction	*Mnemonic*	*Description*
Do I/O	DO_IO	For one scan, immediately services a specified range of inputs or outputs. All inputs or outputs on a module are serviced if any reference locations on that module are included in the DO I/O instruction. Partial I/O module updates are not performed. Optionally, a copy of the scanned I/O can be placed in internal memory, rather than at the real input points.
Edge Detectors	R_TRIG F_TRIG	The *Rising Edge Trigger* R_TRIG and *Falling Edge Trigger* F_TRIG detect the changing state of a Boolean signal. The outputs of both function blocks produce a single pulse when an edge is detected. Execution of the function block defines the pulse.
Mask I/O Interrupt	MASK_IO_INTR	Requests special Controller service 17. Use MASK_IO_INTR with I/O variables to mask or unmask an interrupt from an input board.
Proportional Integral Derivative Control	PID_ISA PID_IND	Provides two PID (Proportional/Integral/Derivative) closed-loop control algorithms: ▪ Standard ISA PID algorithm (PID_ISA) ▪ Independent term algorithm (PID_IND)
Service Request	SVC_REQ	Requests a special Controller service
Suspend IO	SUS_IO	Suspends for one sweep all normal I/O updates, except those specified by DO I/O instructions
Suspend I/O Interrupt	SUSP_IO_INTR	Requests special Controller service 32. Use SUSP_IO_INTR with I/O variables to suspend a set of I/O interrupts and to cause occurrences of these interrupts to be queued until these interrupts are resumed.

Do I/O



Operation

When the EN input operand is set to ON, the DO I/O (DO_IO) instruction updates inputs or outputs for one scan while the program is running. You can also use DO_IO to update selected I/O during the program in addition to the normal I/O scan. You can use DO_IO in conjunction with a Suspend IO (SUS_IO) instruction, which stops the normal I/O scan. If input references are specified, DO_IO allows the most recent values of inputs to be obtained for program logic. If output references are specified, DO I/O updates outputs based on the most current values stored in I/O memory. I/O is serviced in increments of entire I/O modules; the Controller adjusts the references, if necessary, while DO_IO executes. DO_IO does not scan I/O modules that are not configured.

DO_IO continues to execute until all inputs in the selected range have reported or all outputs have been serviced on the I/O modules. Program execution then returns to the function that follows the DO_IO.

If the range of references includes an option module such as a high speed counter, all the input data (%I and %AI) or all the output data (%Q and %AQ) for that module are scanned. The ALT parameter is ignored while scanning option modules.

DO_IO sets its ENO output parameter whenever it receives data unless:

- Not all references of the type specified are present within the selected range.
- The CPU is unable to properly handle the temporary list of I/O created by the function.
- The range specified includes I/O modules that are associated with a "Loss of I/O" fault.

Note: (PACSystems firmware version 2.0 and later, preemptive block scheduling.) A DO_IO instruction inside an interrupt block being executed may cause the block to be preempted when a new, incoming interrupt has the same priority.

Do I/O for Inputs

When the EN input is set to ON and input references are specified, the Controller scans input points from the starting reference (ST) to the ending reference (END). If a reference is specified for ALT, a copy of the new input values is placed in memory beginning at that reference, and the real input values are not updated.

ALT must be the same size as the reference type scanned. If used, ALT must always be a WORD data type.

If no reference is specified for ALT, the real input values are updated. This allows inputs to be scanned one or more times during the program execution portion of the CPU scan.

Do I/O for Outputs

When the EN input is set to ON and output references are specified, the Controller writes to the output points. If no value is specified in ALT, the range of outputs written to the output modules is specified by the starting reference (ST) and the ending reference (END). If outputs should be written to the output points from internal memory other than %Q or %AQ, the beginning reference is specified for ALT and the end reference is automatically calculated from the length of the END-ST range.

Operands

Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use a BOOL array of length 16 or more instead of a WORD variable. Restrictions apply.

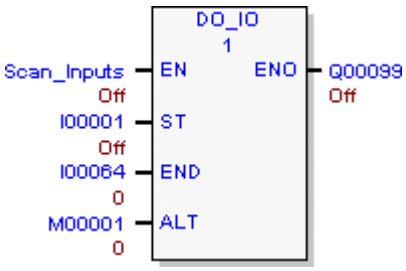
Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
EN	BOOL variable	data flow, I, Q, M, T, G, S, SA, SB, SC, discrete symbolic	Enable input. When set to ON, DO_IO executes. When set to OFF, DO_IO does not execute.
	Bit reference in non-BOOL variable	I, Q, M, T, G, S, SA, SB, SC, R, P, L, AI, AQ, W, non-discrete symbolic	
ST	WORD or BOOL variable	I, Q, AI, AQ, R, M, W, I/O variable	The starting address of the set of input or output points or words to be serviced. ST and END must be in the same memory area. **Notes** ▪ If ST and END are mapped to discrete memory, ST must be byte-aligned, that is, its reference address must start at (8n+1), for example, %I01, %Q09, %Q49. ▪ If ST and END are mapped to analog memory, they can have the same reference address. ▪ If an I/O variable is assigned to the ST parameter, then the same I/O variable must also be assigned to the END parameter, and the entire module is scanned.
END	WORD or	I, Q, AI, AQ, R, M,	The address of the end bit of input or output

	BOOL variable	W, I/O variable	points or words to be serviced. Must be in the same memory area as ST. **Notes** ▪ If ST and END are mapped to discrete memory, END's reference address must be 8n, for example, %I08, %Q16. ▪ If ST and END are mapped to analog memory, they can have the same reference address. ▪ If an I/O variable is assigned to the END parameter, then the same I/O variable must also be assigned to the ST parameter, and the entire module is scanned.
ALT	WORD variable	data flow, I, Q, M, T, G, R, AI, AQ, W, symbolic	(Optional.) For an input scan, ALT specifies the address to store scanned input point/word values. For an output scan, ALT specifies the address to get output point/word values from, to send to the I/O modules. **Note:** ALT must be the same data type as ST and END (discrete or analog), but ALT does not have to be in the same memory area.
ENO	BOOL variable	data flow, I, Q, M, T, G, SA, SB, SC, discrete symbolic	(Optional.) Set to ON whenever DO I/O receives data unless: ▪ Not all references of the type specified are present within the selected range. ▪ The CPU is not able to properly handle the temporary list of I/O created by the function. ▪ The range specified includes I/O modules that are associated with a "Loss of I/O" fault.
	Bit reference in non-BOOL variable	I, Q, M, T, G, SA, SB, SC, R, P, L, AI, AQ, W, non-discrete symbolic	

Examples

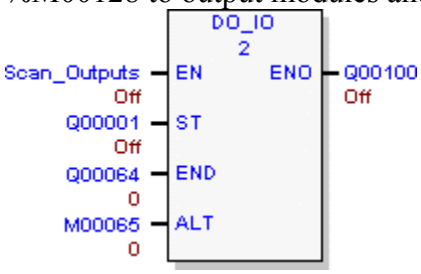
Do I/O for inputs

When Scan_Inputs is set to ON (1), the DO_IO executes, the PACSystems scans references %I00001 through %I00064, and %Q00099 is set to ON (1). A copy of the scanned inputs is placed in internal memory from %M00001 through %M00064.



Do I/O for outputs

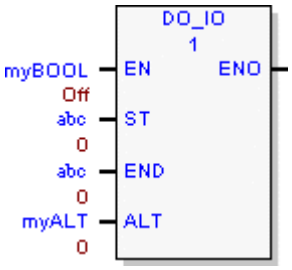
Because a reference is entered for ALT, the values at %Q00001 through %Q00064 are not written to output modules. When DO_IO executes; that is, when Scan_Outputs is set to ON (1), the PACSystems writes the values from references %M00065 through %M00128 to output modules and %Q00100 is set to ON (1).



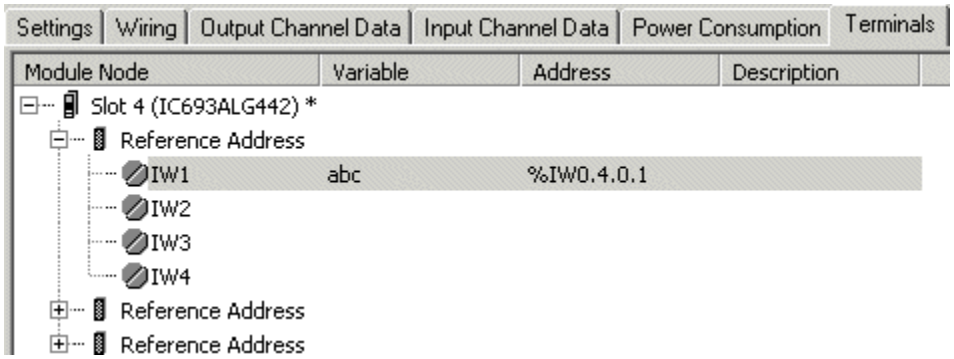
Do I/O and I/O variables

The following example shows the same I/O variable named abc assigned to the ST and END parameters.

Note: If an I/O variable is assigned to the ST parameter, as in this example, then the same I/O variable must also be assigned to the END parameter, and the entire module is scanned.



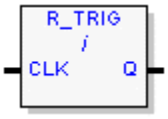
The I/O variable named abc is also assigned to the IW1 (Input WORD) terminal of an IC693ALG442 module.



Edge Detectors - R_TRIG and F_TRIG

Operation

The IEC 61131-3 standard defines two edge detector function blocks. The *Rising Edge Trigger* R_TRIG and *Falling Edge Trigger* F_TRIG detect the changing state of a Boolean signal. The outputs of both function blocks produce a single pulse when an edge is detected.

 **Rising Edge Trigger R_TRIG**	 **Falling Edge Trigger F_TRIG**
You can declare instance variables of type R_TRIG and F_TRIG.	
The structure elements CLK and STATE are not accessible in logic. If they are used in logic, the compiler reports an error.	
All structure elements can be viewed in the Data Watch window and Data Monitor.	
Elements of an edge detector structure variable cannot be published. Publishing the elements would allow other applications to write to the instance data.	
When power flow is false, all Boolean output flow arguments are set to FALSE (this is the standard behavior for a function or function block that is not Enabled). This means that if power flow or Q argument is Boolean flow, it is set to False; if it is a variable, no change is made, that is, the variable will retain its last set value.	
For R_TRIG, when the CLK input transitions from False to True, the output Q is set to True (Q = CLK and NOT STATE; STATE = CLK) for one function block instance execution. The output Q then remains False until a new rising edge is detected.	For F_TRIG, when the CLK input transitions from True to False, the output Q is set to True (Q = NOT CLK and NOT STATE; STATE = NOT CLK) for one function block instance execution. The output Q then remains False until a new falling edge is detected.
For R_TRIG, when the Controller transitions from stop to run mode and the CLK input is set to True, the output Q is set to True after the first execution of the function block instance. After the second execution, the output is set to False.	For F_TRIG, when the Controller transitions from stop to run mode and the CLK input is set to False, the output Q is set to True after the first execution of the function block instance. After the next execution, the output is set to False.

Operands

Operand	*Data Type*	*Memory Area*	*Description*

CLK	BOOL	I, Q, M, T, SA, SB, SC, G, symbolic	Edge detecting input
Q	Variable	I, Q, M, T, symbolic	Edge detecting output
STATE	BOOL		Internal Value

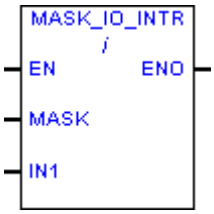
Notes

- C blocks cannot use edge detector function blocks.
- The Structure element Q is accessible in user logic. It is read-only.
- Instance variables reside in symbolic discrete memory, which is non-retentive.

CPU Support

R_TRIG and F_TRIG are supported for PACSystems CPUs with firmware version 5.00.

MASK_IO_INTR



Operation

Use MASK_IO_INTR to mask or unmask an interrupt from an input board when using I/O variables.

Tip: When not using I/O variables, you can use SVC_REQ 17.

When an interrupt is masked, the CPU does not execute the corresponding interrupt block when the input transitions and causes an interrupt.

There is successful execution unless one or more of the following occurs:

- The I/O board is not a supported input module.
- The reference address specified does not correspond to a valid interrupt trigger reference.
- The specified channel does not have its interrupt enabled in the configuration.

Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
EN	BOOL variable or bit reference in non-BOOL variable	data flow, I, Q, M, T, G, S, SA, SB, SC, discrete symbolic, I/O variable	Enable input. When set to ON, MASK_IO_INTR executes. When set to OFF, MASK_IO_INTR does not execute.
	Bit reference in non-BOOL variable	R, P, L, AI, AQ, W, I, Q M, T, G, non-discrete symbolic, I/O variable	
MASK	BOOL variable or bit reference in non-BOOL variable	data flow, I, Q, M, T, G, S, SA, SB, SC, R, P, L, AI, AQ, W, symbolic, I/O variable	If MASK is set to ON, this indicates mask. If MASK is set to OFF, this indicates unmask.
IN1	BOOL or WORD variable	I, Q M, T, G, R, P, L, AI, AQ, W, I/O variable	The value of IN1 indicates the interrupt trigger to be masked or unmasked.

ENO	BOOL variable	data flow, I, Q, M, T, G, SA, SB, SC, discrete symbolic, I/O variable	(Optional.) Set to ON whenever MASK_IO_INTR receives data unless: ▪ Not all references of the type specified are present within the selected range. ▪ The CPU is not able to properly handle the temporary list of I/O created by the function. ▪ The range specified includes I/O modules that are associated with a "Loss of I/O" fault.

Masking / Unmasking Module Interrupts

During module configuration, interrupts from a module can be enabled or disabled. If a module's interrupt is disabled, it cannot be used to trigger logic execution in the application program and it cannot be unmasked. However, if an interrupt is enabled in the configuration, it can be dynamically masked or unmasked by the application program during system operation.

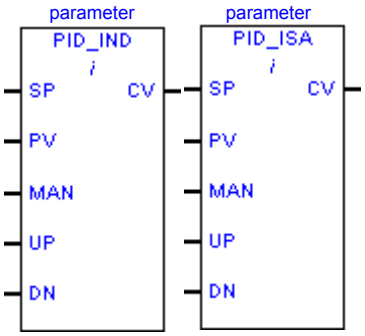
For the application program to mask and unmask interrupts that are enabled, use MASK_IO_INTR when using I/O variables.

When the interrupt is not masked, the CPU processes the interrupt and schedules the associated program logic for execution. When the interrupt is masked, the CPU processes the interrupt but does not schedule the associated program logic for execution.

When the CPU transitions from Stop to Run, the interrupt is unmasked.

For additional information on configuring and using VME module interrupts in a PACSystems control system, refer to *PACSystems RX7i User's Guide to Integration of VME Modules* (GFK-2235).

Proportional Integral Derivative (PID)



Overview

The Proportional Integral Derivative (PID) control built-in function block is a general purpose algorithm used for closed-loop process control. The PID built-in function block compares a Process Variable (PV) feedback with a desired process Set Point (SP) and updates a Control Variable (CV) output based on the error.
The built-in function block uses PID loop gains and other parameters stored in the Reference Array, a 40-WORD array, to solve the PID algorithm at the desired time interval.

Time interval for the PID built-in function block

The PID built-in function block does not execute more often than once every 10 milliseconds. This could change your results if you set it up to execute every scan and the scan is less than 10 milliseconds. In such a case, PID does not run until enough scans have occurred to accumulate an elapsed time of 10 milliseconds. For example, if the scan time is 9 milliseconds, PID executes every other scan with an elapsed time of 18 milliseconds for every time it executes.
The longest possible interval between executions is 10.9 minutes. The PID built-in function block compensates for the actual time elapsed since the last execution within 100 microseconds.
The PID algorithm is solved only if the current CPU elapsed time clock is at or later than the last PID solution time plus the sample period. If the sample period is set to 0, the built-in function block executes each time it is enabled; however, it is restricted to a minimum of 10 milliseconds as noted above.

Scaling input and output

All parameters are 16-bit WORDs for compatibility with 16-bit analog process variables. This enables %AI memory to be used for input Process Variables and %AQ to be used for output Control Variables.
As scaled 16-bit integer numbers, some parameters must be defined in either PV counts or units, or CV counts or units. The SP input must be scaled over the same range as PV because the PID built-in function block calculates error from the difference of these two inputs. The PV and CV counts do not have to use the same scaling. Either can range from

-32768 or 0 to +32767, matching analog scaling, or from 0 to 10000, to display variables as 0.00% to 100.00%. If the PV and CV variable do not use the same scaling, scale factors are included in the PID gains.

Operands

Operand	*Data Type*	*Memory Area*	*Description*
parameter	one-dimensional WORD array of 40 words	R, P, L, W, symbolic	Control parameter. This 40-word array, the Reference Array, is the PID control block information, which contains both user and internal parameters. The 40 consecutive words must not be shared. **Note:** Undo is available when you edit the control parameter.
i	The solve order for the built-in function block.		
SP	INT variable or constant; BOOL array of length 16 or more (restrictions apply)	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The control loop or process Set Point. Comparing the PV Counts to SP, PID adjusts the output CV so that PV matches SP (zero error).
PV	INT variable; BOOL array of length 16 or more (restrictions apply)	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The control loop Process Variable input. Must be scaled over the same range as SP. Often a %AI input.
MAN	BOOL variable	data flow, I, Q, M, T, G, S, SA, SB, SC, discrete symbolic, I/O variable	Manual mode indicator. When SET to ON (1, TRUE), the PID built-in function block is in Manual mode. If MAN is set to OFF (0, FALSE), the PID is in Automatic mode.
	Bit reference in non-BOOL variable	I, Q, M, T, G, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	
UP	BOOL variable	data flow, I, Q, M, T, G, S, SA, SB, SC, discrete symbolic, I/O variable	If both MAN and UP are set to ON (1, TRUE), UP increases the CV by 1 CV per solution, that is, 1 CV per access to the PID built-in function block.

	Bit reference in non-BOOL variable	I, Q, M, T, G, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	
DN	BOOL variable	data flow, I, Q, M, T, G, S, SA, SB, SC, discrete symbolic, I/O variable	If both MAN and DN are set to ON (1, TRUE), DN decreases the CV by 1 CV per solution, that is, 1 CV per access to the PID built-in function block.
	Bit reference in non-BOOL variable	I, Q, M, T, G, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	
CV	INT variable; BOOL array of length 16 or more (restrictions apply)	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The Control Variable output to the process. Often a %AQ analog output.

Reference Array Parameters

Besides the two input words and the three Manual control operands, the PID built-in function block requires 13 user-defined parameters in the Reference Array. These parameters must be set before executing the built-in function block. The other 27 parameters are used by the Controller and are non-configurable. The %Ref shown in the table below is the beginning address of the Reference Array (the A operand). The number after the plus sign is the offset in the array. For example, if the Reference Array starts at %R100, %R113 contains the Manual Command (%Ref+0013) used to set the Control Variable and the integrator in Manual mode.

Notes

- The Reference Array array must be %R, %P, %L, %W registers, non-discrete symbolic variable, or an I/O variable. Every PID built-in function block in logic must use a different 40-word array even if all 13 user parameters are the same, because other words in the array are used for internal PID data storage. Ensure that there are at least 40 %R, %P, %L, or %W registers between the starting reference address and the highest configurable %R, %P, %L, or %W register.
- Undo is available when you edit the control parameter.

Register and Parameter	*Low Bit Units and Range of Values*	*Description*
%REF+0000 Loop Number	UINT. 0 to 255	Optional. Loop number; number of the PID built-in function block. For user display only. Provides a common identification in the Controller with the loop number defined

		by an operator interface device. **Note:** The loop number is displayed under the block address when logic is monitored from the FBD editor.
%REF+0001 Algorithm	UINT. Non-configurable. Set and maintained by the Controller.	1 = ISA algorithm (PID_ISA) 2 = Independent algorithm (PID_IND)
%REF+0002 Sample Period	Low bit set at 1 represents 10ms. UINT. Range: 0 (every scan) to 65,535 (10.9 min).	The shortest time in 10 ms increments, between solutions of the PID algorithm. For example, use a 10 for a 100 ms sample period. If Sample Period is 0, the algorithm is solved every time PID executes, unless the scan is under 10 ms. (For information on PID scheduling, see Sample Period and PID Scheduling). The PID algorithm is solved only if the current Controller elapsed time clock is at or later than the last PID solution time plus this Sample Period. PID compensates for the actual time elapsed since the last execution, within 100 microseconds.
%REF+0003 Dead Band +	PV Counts. 0 to 32767 (never negative)	Integer (INT) values defining the upper (+) and lower (-) Dead Band limits in PV Counts. If no Dead Band is required, these values must be 0. If the PID Error (SP - PV) or (PV - SP) is above the (-) value and below the (+) value, the PID calculations are solved with an Error of 0. If non-zero, the (+) value must be greater than 0 and the (-) value less than 0 or the PID built-in function block will not operate. **Tip:** Leave these at 0 until the PID loop gains are set up or tuned. After that, a Dead Band might be added to avoid small CV output changes due to variations in error, perhaps to reduce mechanical wear. **Note:** The Deadband Action bit determines how PID uses the deadband limits.
%REF+0004 Dead Band .	PV Counts. -32768 to 0 (never positive)	
%REF+0005 ▪ In the PID_IND algorithm,	Low bit set at 1 represents 1% of (CV%)/(PV%). Range of represented values: 0	The change in the CV in CV Counts for a 100 PV Count change in the Error Term. It is displayed as 0.00 %/% with an implied decimal point of 2. A Kp or Kc entered as

this word stores the Proportional Gain, Kp ▪ In the PID_ISA algorithm, this word stores the Controller gain, Kc	to 327.67%	450 is displayed as 4.50 and results in a (Kp*Error/100) or (Kc*Error/100) = (450*Error/100) contribution to the PID Output. **Tip:** When the PID_IND algorithm is used, Kp is generally the first gain set to adjust a PID loop, a PD loop (Proportional Derivative) or a PI loop (Proportional Integral, which is rarely used). Using Kp by itself may be sufficient to control certain types of processes.
%REF+0006 Derivative Gain.Kd	Low bit set at 1 represents 0.01 seconds. Integer. Range of represented values: 0 to 327.67 sec.	The change in the CV in CV Counts if the Error or PV changes 1 PV Count every 10 ms. Entered as a time with the low bit indicating 10 ms, it is displayed as 0.00 seconds with an implied decimal point of 2. For example, a Kd entered as 120 is displayed as 1.20 Sec and results in a [Kd * (delta Error)/(delta time)] = (120*4/3) contribution to the PID Output if Error was changing by 4 PV Counts every 30 ms. **Tip:** When the PID_IND algorithm is used, Kd can be used to speed up a slow loop response, but is very sensitive to PV input noise. (You can alleviate noise sensitivity by using the derivative filter, which is enabled by setting bit 5 of the Config Word.) In some processes, you can omit Kd and control the process by using Kp by itself or Kp in combination with Ki.
%REF+0007 Integral Rate.Ki	Low bit set at 1 represents 1 repeat/1000 seconds. Integer values, 0 to 32,767, representing the values 0 to 32.767 repeats/sec	The change in the CV in CV Counts if the Error is a constant 1 PV Count. Displayed as 0.000 Repeats/Sec with an implied decimal point of 3. For example, a Ki entered as 1400 is displayed as 1.400 Repeats/Sec and results in a (Ki * Error * dt) = (1400*20*50/1000) contribution to PID Output for an Error of 20 PV Counts and a 50 ms Controller scan time (Sample Period of 0). **Tip:** When the PID_IND algorithm is used, Ki is usually the second gain set in a PID loop, Kp being the first gain set. Ki can be omitted entirely in a PD (Proportional Derivative) loop or P (Proportional only) loop. Ki provides inertia to the control system, that is, it acts as an automatic bias.
%REF+0008	CV Counts. Integer, -	Number of CV Counts added to the PID

CV Bias/Output Offset	32768 to +32767 (add to integrator output)	Output before the rate and amplitude clamps. Can be used to set non-zero CV values if only Kp Proportional gains are used, or for feed forward control of this PID loop output from another control loop. For example, if you want to let the PID built-in function block regulate error around the output midpoint, set the Bias to the midpoint of the range. For example, for a full range 0 through +32,767, +16,383 is the midpoint. **Note:** In the PID_IND algorithm, Ki acts as an automatic bias.
%REF+0009 Upper Clamp	CV Counts. Integer, .32768 to +32767. (Must be greater than %Ref+10.) Output limit.	Required values. Number of CV Counts that define the highest and lowest value for CV. The Upper Clamp must have a more positive value than the Lower Clamp; otherwise, the PID built-in function block will not work. These are usually used to define limits based on physical limits for a CV output. They are also used to scale the Bar Graph display for CV (visible when tuning the PID in the PID Project Values dialog box). The block has antireset windup to modify the integrator value when a CV clamp is reached.
%REF+0010 Lower Clamp	CV Counts. Integer, .32768 to +32767. (Must be less than %Ref+09.) Output limit.	
%Ref+0011 Minimum Slew Time	UINT. Number of seconds/Full Travel. Range: 0 (none) to +32767 sec to move +32767 CV	Minimum number of seconds for the CV output to move from 0 to full travel of 100% or +32767 CV Counts. It is an inverse rate limit on how fast the CV output can be changed. If positive, CV cannot change more than (+32767 CV Counts) * (Delta Time (in seconds)) / (Minimum Slew Time). For example, if the Sample Period is 2.5 seconds and the Minimum Slew Time is 500 seconds, CV cannot change more than +32767*2.5/500 = 163 CV Counts per PID solution. An antiwindup feature adjusts the integrator value if the CV rate limit is exceeded. If Minimum Slew Time is 0, there is no CV rate limit. **Note:** Set Minimum Slew Time to 0 while tuning or adjusting PID loop gains.
%Ref+0012	Low 6 bits used.	The low 6 bits of this word are used to

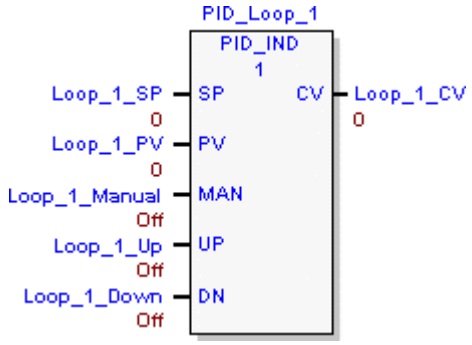
Config Word	BOOLEAN.	modify five standard PID settings. The other bits should be set to 0. **Bit 0:** Error Term. When this bit is 0, the error term is the normal (SP-PV). When this bit is 1, the error term is (PV-SP), which reverses the sign of the feedback term: this is for Reverse Acting controls where the CV must go down when the PV goes up. **Bit 1:** Output Polarity. When this bit is 0, the CV output represents the output of the PID calculation. When it is set to 1, the CV output represents the negative of output of the PID calculation rather than the normal positive value. **Bit 2:** Derivative action on PV. When this bit is 0, the derivative action is applied to the Error Term. When it is set to 1, the derivative action is applied to PV. **Bit 3:** Deadband action. When this bit is set to 0: ▪ If the error is within the deadband limits, then the error is forced to be zero. ▪ Otherwise, the error is not affected by the deadband limits. When the Deadband action bit is set to 1: ▪ If the error is within the deadband limits, then the error is forced to be zero. ▪ If the error is outside the deadband limits, then the error is reduced by the deadband limit (error error - deadband limit). **Bit 4:** Antireset windup action. When this bit is 0, the antireset windup action uses a reset back calculation. When the output is clamped, this replaces the accumulated Y remainder value with whatever value is necessary to produce the clamped output exactly. When the bit is 1, this replaces accumulated Y term with the value of the Y term at the start of the calculation. In this way, the preclamp Y value is held as long as the output is clamped. **Bit 5:** Enable derivative filtering. When this

		bit is set to 0, no filtering is applied to the derivative term. When set to 1, a first order filter is applied. This limits the effects of higher frequency process disturbances on the derivative term. **Bits 6 and 7:** Unused. Should be set to 0. **Note:** The bits are set in powers of 2. For example, to set Config Word to 0 for default PID configuration, you would add 1 to change the Error Term from (SP-PV) to (PV-SP), or add 2 to change the Output Polarity from (CV = PID Output) to (CV = - PID Output), or add 4 to change Derivative Action from Error rate of change to PV rate of change.
%Ref+0013 Manual Command	CV Counts. Integer. Tracks CV in Auto or Sets CV in Manual	Value set to the current CV output while the PID built-in function block is in Automatic mode. When the built-in function block is switched to Manual mode, this value is used to set the CV output and the internal value of the integrator within the Upper and Lower Clamp and Slew Time limits.
%Ref+0014 Control Word	Low 5 bits used. BOOLEAN. Maintained by the Controller, ***unless*** Bit 0 is set.	A discrete data structure with the first five bit positions in the following format: **Bit 0.** Word value: 1. Override. Internal parameter normally left at 0. If this low bit is set to 1, this word and the internal SP, PV and CV parameters must be used for remote operation of this PID built-in function block. This enables remote operator interface devices, such as a computer, to take control away from the Controller program. **Note:** If you do not want to enable remote operation of the PID built-in function block, ensure the Control Word is set to 0. If the low bit is 0, the next 4 bits can be read to track the status of the PID input contacts as long as the PID Enable contact is set to ON. **Bit 1.** Word value: 2. Manual/Auto. If set to 1, the PID built-in function block is in Manual mode; if set to 0, it is in Automatic mode. **Bit 2.** Word value: 4. Enable. Should normally be set to 1; otherwise the PID built-in function block is never called. **Bit 3.** Word value: 8. UP/Raise. If set to 1 and Manual (Bit 1) is set to 1, CV is incremented every solution **Bit 4.** Word value: 16. DN/Lower. If set to 1

		and Manual (Bit 1) is set to 1, CV is decremented every solution.
%Ref+0015 Internal SP	N/A. Set and maintained by the Controller. Non-configurable.	Tracks SP in. Must be set externally if Override bit = 1.
%Ref+0016 Internal CV	N/A. Set and maintained by the Controller. Non-configurable.	Tracks CV out.
%Ref+0017 Internal PV	N/A. Set and maintained by the Controller. Non-configurable.	Tracks PV in. Must be set externally if Override bit = 1.
%Ref+0018 Output	N/A. Set and maintained by the Controller. Non-configurable.	Signed word value representing the output of the built-in function block before the application of the optional inversion. If no output inversion is configured and the output polarity bit in the Config Word is set to 0, this value equals the CV output. If inversion is selected and the output polarity bit is set to 1, this value equals the negative of the CV output.
%Ref+0019 Diff Term Storage	N/A. Set and maintained by the Controller. Non-configurable.	Used internally for storage of intermediate values. *Do not write to this location.*
%Ref+0020 and %Ref+0021 Int Term Storage	N/A. Set and maintained by the Controller. Non-configurable.	Used internally for storage of intermediate values. *Do not write to these locations.*
%Ref+0022 Slew Term Storage	N/A. Set and maintained by the Controller. Non-configurable.	Used internally for storage of intermediate values. *Do not write to this location.*
%Ref+0023, 0024, 0025 Clock	N/A. Set and maintained by the Controller. Non-configurable.	Internal elapsed time storage (time last PID executed). *Normally, do not write to these locations. Occasionally, circumstances may justify writing to these locations.*
%Ref+0026 Y Remainder Storage	N/A. Set and maintained by the Controller. Non-configurable.	Holds remainder for integrator division scaling for 0 steady state error. Also used for antireset windup action.

%Ref+0027 Lower Range for SP, PV	PV Counts. Integer, -32768 to +32767 (Must be less than %Ref+28) for display	Optional integer values in PV Counts that define the highest and lowest display value for the SP and PV.
%Ref+0028 Upper Range for SP, PV	PV Counts. Integer, -32768 to +32767 (Must be greater than %Ref+27) for display	
%Ref+0029 to %Ref+0034 Reserved for internal use	N/A. Non-configurable.	Do not use these references. %Ref+0030 and %Ref+0031 are used when calculating the derivative filter.
%Ref+0035 to Ref+0039 Reserved for external use	N/A. Non-configurable.	

Example



Note: *PID_Loop_1* is an array. Its Array Dimension 1 property is set to 40, and its Data Type property is set to WORD.

Automatic operation

The PID built-in function block executes in automatic operation every scan if you set the MAN input operand to OFF (0, FALSE). The built-in function block compares the current Controller elapsed time clock with the last PID solution time stored in the internal Reference Array. If the time difference is greater than the sample period defined in the third word (%Ref+2) of the Reference Array, the PID algorithm is solved by using the time difference and both the last solution time and Control Variable output are updated. In Automatic mode, the output Control Variable is placed in the Manual Command parameter (%Ref+13).

Manual operation

If you set the MAN input operand to ON (1, TRUE), the PID built-in function block is placed in Manual mode. The output Control Variable (CV) is set from the Manual Command parameter %Ref+13. If either the UP input or DN input parameter is set to ON (1, TRUE), the Manual Command word is respectively incremented or decremented by one CV count every PID solution. For faster manual changes of the output CV, you can also add or subtract any CV count value directly to/from the Manual Command word.
The PID built-in function block uses the CV Upper and CV Lower Clamp parameters to limit the CV output. If a positive Minimum Slew Time is defined, it is used to limit the rate of change of the CV output.

If either the CV amplitude or rate limit is exceeded, the value stored in the integrator is adjusted so that CV is at the limit. This antireset windup feature means that even if the error tried to drive CV above (or below) the clamps for a long period of time, the CV output will move off the clamp as soon as the error term changes sign.
This operation, with the Manual Command tracking CV in Automatic mode and setting CV in Manual mode, provides a bumpless transfer between Automatic and Manual modes. The CV Upper and Lower Clamps and the Minimum Slew Time still apply to the CV output in Manual mode and the internal value stored in the integrator is updated. This means that if you step the Manual Command in Manual mode, the CV output does not change any faster that the Minimum Slew Time (Inverse) rate limit and does not go above or below the CV Upper or CV Lower Clamp limits.

Note: A specific PID built-in function block should not be called more than once per scan.

Internal Parameters in Reference Array

The PID built-in function block reads 13 user parameters and uses the rest of the 40-word Reference Array for internal PID storage. Normally you would not change any of these values. If you execute the PID built-in function block in Automatic mode after a long delay, you might want to use SVC_REQ 16 or SVC_REQ 51 to load the current Controller elapsed time clock into %Ref+23 to update the last PID solution time to avoid a step change on the integrator. If you have set the Override low bit of the Control Word (%Ref+14) to 1, the next four bits of the Control Word must be set to control the BOOL input parameters of PID built-in function block, and the Internal SP and PV must be set because you have taken control of the PID built-in function block away from FBD logic.

PID Algorithm Selection (PID_IND or PID_ISA) and Gains

The PID built-in function block can be programmed selecting either the Independent (PID_IND) term or standard ISA (PID_ISA) versions of the PID algorithm. The only differences in the algorithms are:

- How the Integral and Derivative gains are defined.
- What word 6 of the Reference Array is used for.

Algorithm	*Value*	*Definition*	*Condition*
Both	*Error=*	(SP – PV) Reverse Acting	if the low bit of Config Word is set to 0
		(PV - SP) Direct Acting	if the low bit of Config Word is set to 1

Note: Direct Acting is sometimes referred to as Forward Acting.

Both PID types calculate the Error term as SP – PV, Reverse Acting, which can be changed to the Direct Acting mode, PV – SP, by setting the Error Term to 1. The Error Term is the low bit (0) in the Config Word (%Ref+12).

- In a **direct** acting proportional (P) loop, an increase in the process variable (PV) causes an increase in the output (CV).
- In a **reverse** acting P loop, an increase in PV causes a decrease in CV.

Introducing the integral term (I) changes the behavior:

- In a **direct** acting PI loop, the output (CV) increases when the process variable (PV) is greater than the setpoint (SP).
- In a **reverse** acting PI loop, the output (CV) decreases when the process variable (PV) is greater than the setpoint (SP).

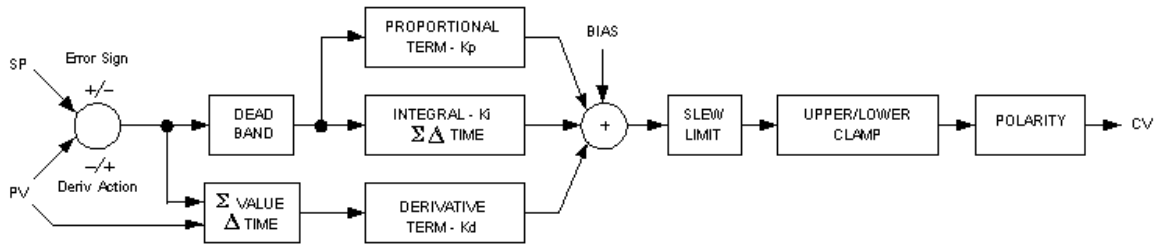
Algorithm	*Value*	*Definition*	*Condition*

Both	*Derivative=*	(Error – previous Error)/dt	if the 3rd bit of Config Word is set to 0
		(PV – previous PV)/dt	if the 3rd bit of Config Word is set to 1
Both	*dt=*	(Current Controller Elapsed Time clock) – (Controller Elapsed Time Clock at Last PID solution)	

The Derivative is normally based on the change of the Error term since the last PID solution, which may cause a large change in the output if the SP value is changed. If this is not desired, the third bit of the Config Word can be set to 1 to calculate the Derivative based on the change of the PV. The dt (or Delta Time) is determined by subtracting the last PID solution clock time for this built-in function block from the current Controller elapsed time clock.

Algorithm	*Value*	*Definition*
PID_IND	PID output	Kp * Error + Ki * Error * dt + Kd * Derivative + CV Bias
PID_ISA	PID output	Kc * (Error + Error * dt/Ti + Td * Derivative) + CV Bias

where Kc is the Controller gain, and Ti is the Integral time and Td is the Derivative time. The advantage of PID_ISA is that adjusting the Kc changes the contribution for the integral and derivative terms as well as the proportional one, which may make loop tuning easier. If you have PID gains in terms or Ti and Td, use Kp = Kc, Ki = Kc/Ti, and Kd = Kc*Td
to convert them to use as PID User Parameter inputs.
The CV Bias term is an additive term separate from the PID components. It may be required if you are using only Proportional Kp gain and you want the CV to be a non-zero value when the PV equals the SP and the Error is 0. In this case, set the CV Bias to the desired CV when the PV is at the SP. CV Bias can also be used for feed forward control where another PID loop or control algorithm is used to adjust the CV output of this PID loop.
If an Integral Ki gain is used, the CV Bias would normally be 0 as the integrator acts as an automatic bias. Just start up in Manual mode and use the Manual Command word (%Ref+13) to set the integrator to the desired CV, then switch to Automatic mode. This also works if Ki is 0, except the integrator will not be adjusted based on the Error after going into Automatic mode.
PID_IND. The following diagram shows how the PID_IND algorithm works:



PID_ISA. The ISA Algorithm (PID_ISA) is similar except that the Kc gain is factored out of Ti and Td so that the integral gain is (Kc/Ti) and the derivative gain is (Kc * Td). The Error sign, DerivAction and Polarity are set by bits in the Config Word user parameter.

CV Amplitude and Rate Limits

The built-in function block does not send the calculated PID Output directly to CV. Both PID algorithms can impose amplitude and rate of change limits on the output Control Variable. The maximum rate of change is determined by dividing the maximum 100% CV value (32767) by the Minimum Slew Time, if specified as greater than 0. For example, if the Minimum Slew Time is 100 seconds, the rate limit will be 327 CV counts per second. If the dt solution time was 50 milliseconds, the new CV output cannot change more than 327*50/1000 or 16 CV counts from the previous CV output.

The CV output is then compared to the CV Upper and CV Lower Clamp values. If either limit is exceeded, the CV output is set to the clamped value. If either the rate or the amplitude limit is exceeded modifying CV, the internal integrator value is adjusted to match the limited value to avoid reset windup.
Finally, the block checks the Output Polarity (2nd bit of the Config Word (%Ref+12)) and changes the sign of the output if the bit is 1.

Value	*Definition*	*Condition*
CV=	Clamped PID output	if the Output Polarity bit is set to 0
	(-Clamped PID output)	if the Output Polarity bit is set to 1

If the block is in Automatic mode, the final CV is placed in the Manual Command (%Ref+13). If the block is in Manual mode, the PID equation is skipped as CV is set by the Manual Command, but all the rate and amplitude limits are still checked. That means the Manual Command cannot change the output above the CV Upper Clamp or below the CV Lower Clamp and the output cannot change faster than the Minimum Slew Time allowed.

Sample Period and PID Scheduling

The PID built-in function block is a digital implementation of an analog control function, so the dt sample time in the PID Output equation is not the infinitesimally small sample time available with analog controls. The majority of processes being controlled can be approximated as a gain with a first or second order lag, possibly with a pure time delay. The PID built-in function block sets a CV output to the process and uses the process feedback PV to determine an Error to adjust the next CV output. A key process parameter is the total time constant, which is how fast the PV responds when the CV is changed. The total time constant, Tp+Tc, for a first order system is the time required for PV to reach 63% of its final value when CV is stepped. (For further information, see PV Unit Reaction Curve Input from Process.) The PID built-in function block cannot control a process unless its Sample Period is well under half the total time constant. Larger Sample Periods will make it unstable.
The Sample Period should be no bigger than the total time constant divided by 10 (or down to 5 worst case). For example, if PV seems to reach about 2/3 of its final value in 2 seconds, the Sample Period should be less than 0.2 seconds, or 0.4 seconds worst case. On the other hand, the Sample Period should not be too small, such as less than the total time constant divided by 1000, or the (Ki * Error * dt) term for the PID integrator will round down to 0. For example, a very slow process that takes 10 hours or 36000 seconds to reach the 63% level should have a Sample Period of 40 seconds or longer.
Unless the process is very fast, it is not usually necessary to use a Sample Period of 0 to solve the PID algorithm every PID scan. If many PID loops are used with a Sample Period greater than the scan time, there may be wide variations in Controller scan time if many loops end up solving the algorithm at the same time. The simple solution is to sequence one or more 1 bits through an array of bits set to 0 that is being used to enable data flow to individual PID built-in function blocks.

Setting User Parameters and Tuning Loop Gains for Simple Processes

Because all PID built-in function block parameters are totally dependent on the process being controlled, there are no predetermined values that will work; however, it is usually a simple, iterative procedure to find acceptable loop gain for simple processes.

Note: The following is only a ***possible*** sequence of steps.

To set user parameters and tune loop gains for *simple* processes:

1. Set all the built-in function block parameters to 0, then set the CV Upper and CV Lower Clamps to the highest and lowest CV expected. Set the Sample Period to a value in the range [(estimated process time constant)/10] to [(estimated process time constant)/100].
2. Put the PID built-in function block in Manual mode and set Manual Command (%Ref+13) to different values to check if CV can be moved to Upper and Lower Clamp. Record PV value at some CV point and load it into SP.

3. Set a small gain, such as (100 * Maximum CV/Maximum PV), into Kp and turn off Manual mode. Step SP by 2 to 10% of the Maximum PV range and observe PV response. Increase Kp if PV step response is too slow or reduce Kp if PV overshoots and oscillates without reaching a steady value.
4. Once a Kp is found, start increasing Ki to get overshooting that dampens out to a steady value in 2 to 3 oscillations. This may require reducing Kp. Also try different step sizes and CV operating points.
5. After suitable Kp and Ki gains are found, try adding Kd to get quicker responses to input changes providing it does not cause oscillations.
   Kd is often not needed and does not work with noisy PV.
6. Check gains over different SP operating points and add Dead Band and Minimum Slew Time if needed. Some Reverse Acting processes may need setting Config Word Error Sign or Polarity bits.

Setting User Parameters and Tuning Loop Gains for Not-So-Simple Processes

To set user parameters and tune loop gains for *not-so-simple* processes:

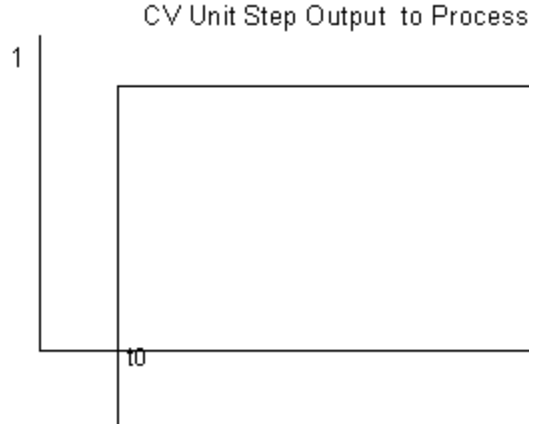
1. Determine the K, Tp, and Tc process characteristics.
2. Use the K, Tp, and Tc parameters to estimate initial Kp, Ki, and Kd parameters.
3. Convert the estimated Kp, Ki, and Kd parameters into the units required by the PID built-in function block.
4. Set the converted Kp, Ki, and Kd parameters, and other user parameters in the Reference Array.
5. Fine tune until you get the desired result.

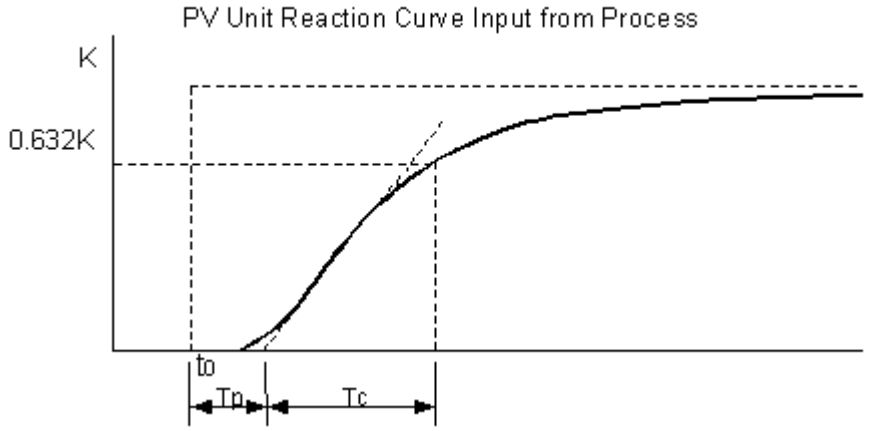
Determining the K, Tp, and Tc Process Characteristics

The PID loop gains, Kp, Ki and Kd, are determined by the characteristics of the process being controlled. Two key questions when setting up a PID loop are:

- How big is the change in PV when CV changes by a fixed amount, or what is the open loop gain?
- How fast does the system respond, or how quickly does PV change after the CV output is stepped?

Many processes can be approximated by a process gain, first or second order lag and a pure time delay. In the frequency domain, the transfer function for a first order lag system with a pure time delay is:
PV(s)/CV(s) = G(s) = K * e **(–Tp s)/(1 + Tc s)
Plotting a step response at time t0 in the time domain provides an open loop unit reaction curve:





The tangent at the maximum slope of the PV unit reaction curve enables you to determine the following process model parameters:

Parameter	*Description*
K	Process open loop gain = final change in PV/change in CV at time t0 **Note:** No subscript on K.
Tp	Process or pipeline time delay or dead time after t0 before the process output PV starts moving
Tc	First order Process time constant, time required after Tp for PV to reach 63.2% of the final PV

The following procedure is usually the quickest way to measure the K, Tp, and Tc parameters:

To determine the K, Tp, and Tc process characteristics

1. Put the PID built-in function block in Manual mode.
2. Make a small step in CV output, by changing the Manual Command (%Ref+13), and plot the PV response.
3. Repeat step 2 over time.

   For slow processes, this can be done manually, but for faster processes a chart recorder or computer graphic data logging package will help.

The CV step size should be large enough to cause an observable change in PV, but not so large that it disrupts the process being measured. A good size may be from 2 to 10% of the difference between the CV Upper and CV Lower Clamp values.

Setting Loop Gains for Not-So-Simple Processes

Once you have determined the three process model parameters, K, Tp and Tc, you can use them to estimate initial PID loop gains. Two approaches are suggested:

- The approach developed by Ziegler and Nichols in the 1940s provides good response to system disturbances with gains producing an amplitude ratio of 1/4. The amplitude ratio is the ratio of the second peak over the first peak in the closed loop response.
- The "Ideal Tuning" procedure provides the best response to SP changes, delayed only by the Tp process delay or dead time.

To determine initial loop gains using the Ziegler and Nichols approach:

1. Calculate the Reaction rate:
   R = K/Tc
2. For Proportional control only, calculate Kp as
   Kp = 1/(R * Tp) = Tc/(K * Tp)
   For Proportional and Integral control, use
   Kp = 0.9/(R * Tp) = 0.9 * Tc/(K * Tp)
   Ki = 0.3 * Kp/Tp
   For Proportional, Integral and Derivative control, use
   Kp = G/(R * Tp), where G is from 1.2 to 2.0
   Ki = 0.5 * Kp/Tp
   Kd = 0.5 * Kp * Tp
3. Check that the Sample Period is in the range (Tp + Tc)/10 to (Tp + Tc)/1000

To determine initial loop gains using the "Ideal Tuning" procedure:

1. Calculate Kp = 2 * Tc/(3 * K * Tp)
2. Calculate Ki = Tc
3. If Derivative term is used, calculate Kd = Ki/4

To convert the estimated Kp, Ki, and Kd parameters into the units required by the PID built-in function block.

1. After determining initial gains, convert them to integer User Parameters. To avoid scaling problems, you should calculate the Process gain, K, as a change in input PV Counts divided by the output step change in CV Counts and not in process PV or CV engineering units. Specify all times in seconds.
2. After determining Kp, Ki and Kd, you can multiply Kp and Kd by 100 and enter them as integer while you can multiply Ki by 1000 and enter it into the User Parameter Reference Array.

Tuning the PID in the PID Project Values dialog box

To tune the PID in the PID Project Values dialog box, that is, to set user-defined parameters in the Reference Array:

1. In the FBD editor, right-click a PID built-in function block and select **Tuning**.
   The PID Project Values dialog box appears.
2. Enter values and select options as required.
3. To save your entries, click **Update Project** and when the confirmation popup appears, click **Yes**.
   This will update the PID's initial values in the project on your workstation, but it will not update the Controller.

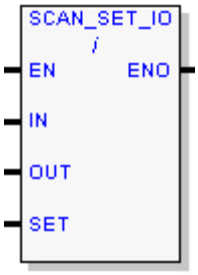
Tip: In the PID - Project Values dialog box, click the Help button to view reference information about the various tuning parameters. The Help topics will automatically scroll into view as you hover over or select controls on the dialog box.

Notes

- You can use 0 for most default values, except the CV Upper Clamp, which must be greater than the CV Lower Clamp for the PID built-in function block to operate.

- Once suitable PID values have been chosen, they should be defined as constants in some MOV instructions so that they can be used to reload default PID user parameters if needed.

SCAN SET I/O



Mnemonic:
SCAN_SET_IO

Operation

A Scan Set I/O (SCAN_SET_IO) instruction scans the I/O of a specified scan set number. You can specify whether the inputs and outputs of the associated scan are scanned.

To assign less frequent scans:

1. In the Project tab of the Navigator, expand the Hardware Configuration node.
2. Expand the main rack.
3. Double-click the slot that contains the CPU.
   The Parameter editor displays the CPUs parameters.
4. In the Scan Sets tab, set the Number of Sweeps to a value greater than 1.

Notes

- Modules can be assigned to scan sets in hardware configuration.
- Execution of the SCAN_SET_IO instruction will not impact the normal scanning process of the corresponding scan set. For example, if the corresponding scan set is configured for non-default Number of Sweeps and/or Output delay settings, these are still in effect regardless of how many executions of the SCAN_SET_IO instruction occur in a sweep.
- The SCAN_SET_IO instruction will not scan any modules in the specified scan set if the modules do not support DO_IO scanning.

Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

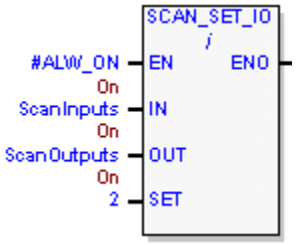
Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction		
EN	BOOL variable, bit reference in non-BOOL	Data Flow	Enable input. When set to ON, SCAN_SET_IO executes. When set to OFF, SCAN_SET_IO does not execute.

ENO	non-BOOL variable		Optional. Activated when all arguments to the instruction are valid and there are no errors in scanning.
IN			If set to True, the inputs are scanned.
OUT			If set to True, the outputs are scanned.
SET	UINT variable	All except %S memory types	Number of the scan set to be scanned. Scan Sets are specified in the CPU hardware configuration and assigned to modules in the module hardware configuration. **Note:** Valid range for PACSystems is 1 through 32.

Examples

By using the SCAN_SET_IO instruction in an interrupt block, you can create a custom I/O scan. For example, two SCAN_SET_IO instructions can be used in an interrupt block to scan the inputs of a Scan Set at the beginning of the block and the outputs of the same Scan Set at the end of the block.
See the following example.

- When ScanInputs is set to ON, input data for all I/O modules assigned to Scan Set 2 is updated.
- When ScanOutputs is set to ON, output data for all I/O modules assigned to Scan Set 2 is updated.

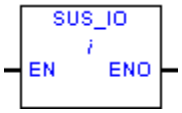


Unsupported Modules

The following PACSystems modules do not support DO_IO scanning:

IC693BEM331 90-30 Genius Bus Controller
IC694BEM331 RX3i Genius Bus Controller
IC693BEM341 90-30 2.5 GHz FIP Bus Controller
IC693DNM200 90-30 Device Net Master
IC695PBM300 RX3i ProfiBus Master
IC695PBS301 RX3i ProfiBus Slave
IC687BEM731 90-70 Genius Bus Controller
IC697BEM731 90-70 Standard Width Genius Bus Controller

Suspend I/O



Operation

The Suspend I/O (SUS_IO) function stops normal I/O scans from occurring for one CPU scan. During the next output scan, all outputs are held at their current states. During the next input scan, the input references are not updated with data from inputs. However, during the input scan portion of the scan, the CPU verifies that Genius bus Controllers have completed their previous output updates.

Note: SUS_IO function suspends all I/O, both analog and discrete, whether integrated I/O, Genius I/O, or Ethernet Global Data. For details, refer to Chapter 4 in the *TCP/IP Ethernet Communications for the Series 90 PLC User's Manual*, GFK-1541.

When the EN input parameter is set to ON (1, TRUE), all I/O servicing stops except that provided by DO_IO functions.

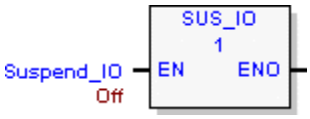
Warning: If the EN parameter is forced ON, no regular I/O scan can be performed until EN is set to OFF. (See Forces.)

Whenever SUS_IO executes, it sets its ENO output parameter to ON.

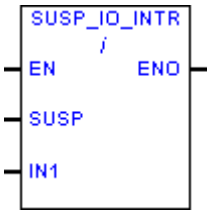
Operands

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
EN	BOOL variable	data flow, I, Q, M, T, G, S, SA, SB, SC, discrete symbolic, I/O variable	Enable input. When set to ON, SUS_IO executes. **Warning:** If the EN parameter is forced ON, no regular I/O scan can be performed until EN is set to OFF. (See Forces.)
	Bit reference in non-BOOL variable	I, Q, M, T, G, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	
ENO	BOOL variable	data flow, I, Q, M, T, G, discrete symbolic, I/O variable	(Optional.) Output. When SUS_IO executes, ENO is always set to ON.
	Bit reference in non-BOOL variable	I, Q, M, T, G, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	

Example



SUSP_IO_INTR



Operation

Use SUSP_IO_INTR to suspend or resume an interrupt from an input board when using I/O variables.

Tip: When not using I/O variables, you can use SVC_REQ 32.

When an interrupt is suspended, the CPU does not execute the corresponding interrupt block when the input transitions and causes an interrupt.
There is successful execution unless one or more of the following occurs:

- The I/O board is not a supported input module.
- The reference address specified does not correspond to a valid interrupt trigger reference.
- The specified channel does not have its interrupt enabled in the configuration.

Operands

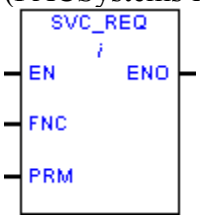
Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
EN	BOOL variable or bit reference in non-BOOL variable	data flow, I, Q, M, T, G, S, SA, SB, SC, discrete symbolic, I/O variable	Enable input. When set to ON, SUSP_IO_INTR executes. When set to OFF, SUSP_IO_INTR does not execute.
	Bit reference in non-BOOL variable	R, P, L, AI, AQ, W, I, Q M, T, G, non-discrete symbolic, I/O variable	
SUSP	BOOL variable or bit reference in non-BOOL variable	data flow, I, Q, M, T, G, S, SA, SB, SC, R, P, L, AI, AQ, W, symbolic, I/O variable	If state of SUSP is set to ON, this indicates suspend. If state of SUSP is set to OFF, this indicates resume.
IN1	BOOL or WORD variable	I, Q M, T, G, R, P, L, AI, AQ, W, I/O variable	The value of IN1 indicates the interrupt trigger to be suspended or resumed.

ENO	BOOL variable	I, Q M, T, G, SA, SB, SC, discrete symbolic, I/O variable	(Optional.) Set to ON whenever SUSP_IO_INTR receives data unless: ▪ Not all references of the type specified are present within the selected range. ▪ The CPU is not able to properly handle the temporary list of I/O created by the function. ▪ The range specified includes I/O modules that are associated with a "Loss of I/O" fault.

Service Request

(PACSystems firmware version 3.50 and later.)



Operation

When the EN input operand of SVC_REQ is set to ON (1, TRUE), SVC_REQ requests the Controller to perform the special Controller service identified by the FNC operand. Parameters for SVC_REQ are located in the parameter block, which begins at the reference identified by the PRM operand. The number of 16-bit memory locations required depends on the type of special Controller service being requested. The parameter block is used to store both the function's inputs and outputs.
SVC_REQ sets the ENO operand to 1 (ON, TRUE) **unless** an incorrect function number, incorrect parameters, or out-of-range references are specified. Various specific SVC_REQ functions have additional causes for failure.

Note: (PACSystems firmware version 2.0 and later, preemptive block scheduling.) Most of the SVC_REQ functions, when used inside an interrupt block being executed, may cause the block to be preempted when a new, incoming interrupt has the same priority.

Operands

Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
EN	BOOL variable	data flow, I, Q, M, T, G, S, SA, SB, SC, discrete symbolic, I/O variable	Enable input. When set to ON, the SVC_REQ executes.
	Bit reference in non-BOOL variable	R, P, L, AI, AQ, W, I, Q M, T, G, non-discrete symbolic, I/O variable	
FNC	INT variable or constant	data flow, I, Q M, T, G, R, P, L, AI,	Function number; Service Request number. The constant or reference that identifies the

		AQ, W, non-discrete symbolic, I/O variable	requested service.
PRM	INT variable	I, Q M, T, G, R, P, L, AI, AQ, W, data flow, non-discrete symbolic, I/O variable	The first WORD in the parameter block for the requested service. Successive 16-bit locations store additional parameters as required. The number of WORDs required for SVC_REQ depends on the value of the FNC operand. **Notes** ▪ You can use data flow only if the parameter block requires only one WORD. ▪ If you use a symbolic variable or an I/O variable, ensure that its Array Dimension 1 property is set to a value large enough to contain the entire parameter block.
ENO	BOOL variable	data flow, I, Q, M, T, G, discrete symbolic, I/O variable	(Optional.) Output. Set to ON (1, TRUE) **unless** an incorrect function number, incorrect parameters, or out-of-range reference is specified. Various specific SVC_REQ functions have additional causes for failure.
	Bit reference in non-BOOL variable	R, P, L, AI, AQ, W, I, Q M, T, G, non-discrete symbolic, I/O variable	

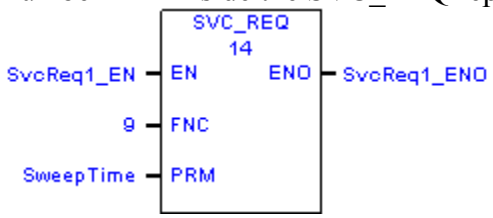
Available SVC_REQ functions

SVC_REQ 1 Change/Read Constant Sweep Timer
SVC_REQ 2 Read Window Modes and Times Values
SVC_REQ 3 Change Programmer Communications Window Mode and Timer Value
SVC_REQ 4 Change System Communications Window Mode and Timer Value
SVC_REQ 5 Change Background Task Window Mode and Timer Value
SVC_REQ 6 Change/Read Number of Words to Checksum
SVC_REQ 7 Read or Change the Time-of-Day Clock
SVC_REQ 8 Reset Watchdog Timer
SVC_REQ 9 Read Sweep Time from Beginning of Sweep
SVC_REQ 10 Read Folder Name
SVC_REQ 11 Read Controller ID
SVC_REQ 12 Read Controller Run State
SVC_REQ 13 Shut Down (Stop) Controller
SVC_REQ 14 Clear Fault Tables
SVC_REQ 15 Read Last-Logged Fault Table Entry
SVC_REQ 16 Read Elapsed Time Clock
SVC_REQ 17 Mask/Unmask I/O Interrupt
SVC_REQ 18 Read I/O Override Status

SVC_REQ 19 Set Run Enable/Disable
SVC_REQ 20 Read Fault Tables
SVC_REQ 21 User-Defined Fault Logging
SVC_REQ 22 Mask/Unmask Timed Interrupts
SVC_REQ 23 Read Master Checksum
SVC_REQ 24 Reset Smart Module or Daughterboard
SVC_REQ 25 Disable/Enable EXE Block and Standalone C Program Checksums
SVC_REQ 26 Role Switch
SVC_REQ 27 Write to Reverse Transfer Area
SVC_REQ 28 Read from Reverse Transfer Area
SVC_REQ 29 Read Elapsed Power Down Time
SVC_REQ 32 Suspend/Resume I/O Interrupt
SVC_REQ 43 Disable Data Transfer Copy in Backup
SVC_REQ 45 Skip Next Output and Input Scan (Suspend I/O)
SVC_REQ 50 Read Elapsed Time Clock (Two DWORDs)
SVC_REQ 51 Read Sweep Time from Beginning of Sweep (DWORD)
SVC_REQ 55 Set Application Redundancy Mode
SVC_REQ 56 (PACSystems) Logic Driven Read of Nonvolatile Storage
SVC_REQ 57 (PACSystems) Logic Driven Write to Nonvolatile Storage

Example

When the enabling input SvcReq1_EN is set to ON, SVC_REQ function number 9 is called, with the parameter block that occupies one WORD represented by the variable SweepTime. If the operation succeeds, the output SvcReq1_ENO is set to ON. The number "14" inside the SVC_REQ represents the FBD solve order.



SVC_REQ 1: Change/Read Constant Sweep Timer

(PACSystems firmware version 3.50 and later.)

Operation

Use SVC_REQ function 1 to:

- Disable Constant Sweep mode
- Enable Constant Sweep mode and use the old Constant Sweep timer value
- Enable Constant Sweep mode and use a new Constant Sweep timer value
- Set a new Constant Sweep timer value only
- Read Constant Sweep mode state and timer value.

The parameter block has a length of two words used for both input and output.

SVC_REQ executes successfully unless:

- A number other than 0, 1, 2, or 3 is entered as the requested operation:
- The scan time value is greater than 2550 ms.
- Constant sweep time is enabled with no timer value programmed or with an old value of 0 for the timer.

To disable Constant Sweep mode:

Enter SVC_REQ 1 with this parameter block:

Address	0
Address + 1	Ignored

To enable Constant Sweep mode and use the old timer value:

Enter SVC_REQ 1 with this parameter block:

Address	1
Address + 1	0

If the timer value does not already exist, entering 0 causes the function to set the ENO output to OFF.

To enable Constant Sweep mode and use a new timer value:

Enter SVC_REQ 1 with this parameter block:

Address	1
Address + 1	New timer value **Note:** If the timer value does not already exist, entering 0 causes the function to set the ENO output to OFF.

To change the timer value without changing the selection for sweep mode state:

Enter SVC_REQ 1 with this parameter block:

Address	2
Address + 1	New timer value

To read the current timer state and value without changing either:

Enter SVC_REQ 1 with this parameter block:

Address	3
Address + 1	ignored

Output

SVC_REQ 1 returns the timer state and value in the same parameter block references:

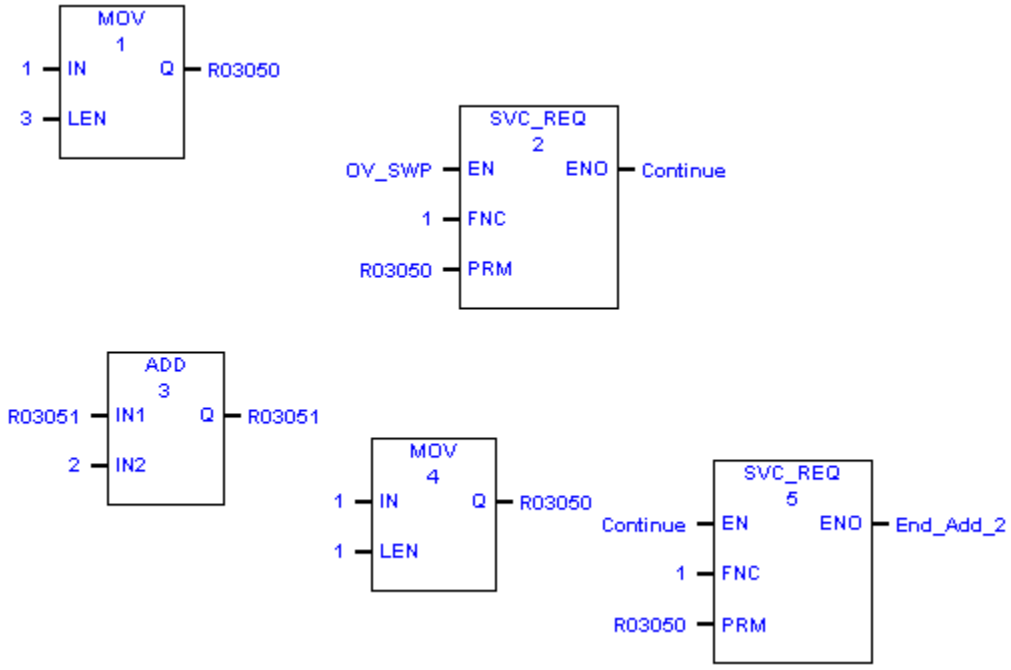
Address	0 = Normal Sweep 1 = Constant Sweep
Address + 1	Current timer value

If the word address + 1 contains the hexadecimal value FFFF, no timer value has ever been programmed.

Example

The two-WORD parameter block for SVC_REQ 1 is at locations %R03050 and %R03051, respectively represented by variables R03050 and R03051.
The value 3 is moved to %R03050 to set the value of the PRM parameter. When OV_SWP is set to ON, SVC_REQ 1 reads the current timer state and value into %R03050 and %R03051 without changing either. The ADD instruction increases the current timer value, stored in %R03051, by two milliseconds. The MOV instruction with a solve order of 4 sets the value of the PRM parameter to 1. The new timer value, stored in %R03051, is sent to the Controller.
On any scan in which OV_SWP is set to OFF, the Continue variable is set to OFF and the second SVC_REQ 1 instruction, with a solve order of 5, is not executed: the timer value is unaffected during this scan.

Note: The numbers inside the instructions refer to the FBD solve order.



SVC_REQ 2: Read Window Modes and Time Values

(PACSystems firmware version 3.50 and later.)

Operation

Use SVC_REQ 2 to obtain the current window mode and time values for the programmer communications window and the system communications.
The parameter block has a length of three words. All parameters are output parameters. It is not necessary to enter values in the parameter block to program this instruction.

Output

Address	*Window*	*High Byte*	*Low Byte*
address	Programmer Window	Mode	Value in ms
address+1	System Communications Window	Mode	Value in ms
address+2	Background Window	Mode	Value in ms

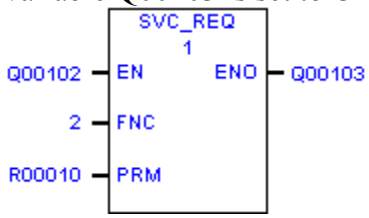
Note: A window is disabled when the time value is zero.

Mode Values

Mode Name	*Value*	*Description*
Limited Mode	0	The execution time of the window is limited to its respective default value or to a value defined using SVC_REQ 3 for the programmer communications window or SVC_REQ 4 for the systems communications window. The window will terminate when it has no more tasks to complete.
Constant Mode	1	Each window will operate in a Run to Completion mode, and the CPU will alternate among the three windows for a time equal to the sum of each window's respective time value. If one window is placed in Constant mode, the remaining two windows are automatically placed in Constant mode. If the CPU is operating in Constant Window mode and a particular window's execution time is not defined using the associated SVC_REQ instruction, the default time for that window is used in the constant window time calculation.
Run to Completion Mode	2	Regardless of the window time associated with a particular window, whether default or defined using a service request instruction, the window will run until all tasks within that window are completed.

Example

When enabling input Q00102 is set to ON, the SVC_REQ 2 executes. The CPU places the current time values of the windows in the parameter block starting at location %R00010, that the R00010 variable is mapped to. If the operation is successful, the variable Q00103 is set to ON.



Note: The 1 inside the instruction refers to the FBD solve order.

SVC_REQ 3: Change Controller Communications Window Mode

(PACSystems firmware version 3.50 and later.)

Operation

Use SVC_REQ 3 to change the Controller communications window mode and timer value. The change takes place during the next CPU sweep after the instruction is called. SVC_REQ 3 sets ENO to 1 (ON, TRUE) unless a mode other than 0 (Limited) or 2 (Run-to-Completion) is selected.
The parameter block has a length of one word.

To disable the Controller communications window:

Enter SVC_REQ 3 with this parameter block:

Address	High Byte	*Low Byte*
Address	0	0

To reenable or change the Controller communications window mode:

Enter SVC_REQ 3 with this parameter block:

Address	*High Byte*	*Low Byte*
Address	Mode	1ms ≤ value ≤ 255ms

SVC_REQ 4: Change Backplane Communications Window Mode and Timer Value

Operation

Use SVC_REQ 4 to change the Backplane Communications window mode and timer value. The change takes place during the next CPU sweep after the instruction is called. SVC_REQ 4 sets the ENO operand to 1 (ON, TRUE) unless a mode other than 0 (Limited), 1 (Constant), or 2 (Run-to-Completion) is selected.
The parameter block has a length of one word.

To disable the Backplane Communications window:

Enter SVC_REQ 4 with this parameter block:

Address	*High Byte*	*Low Byte*
Address	0	0

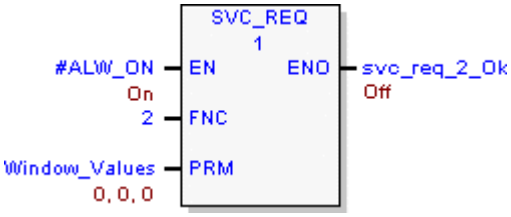
To enable the Backplane Communications window mode:

Enter SVC_REQ 4 with this parameter block:

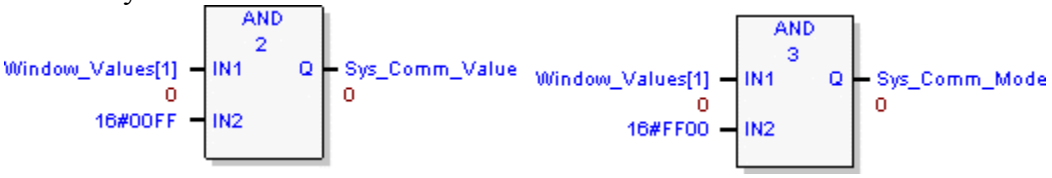
Address	*High Byte*	*Low Byte*
Address	Mode	1ms ≤ value ≤ 255ms

Example

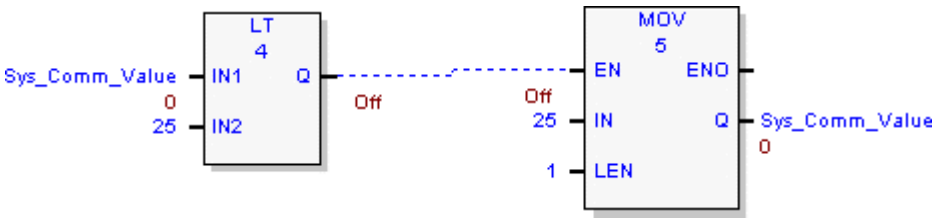
Read the Window Times from the Controller.



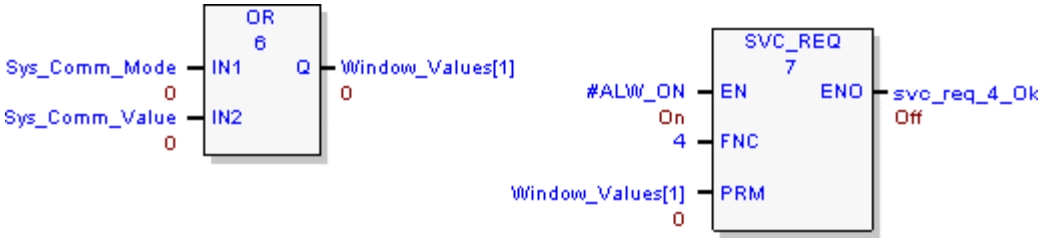
Get the System Communication Window Value and Mode.



If the System Communication Window Value is less than 25 then set it to 25.



Determine the new values and then write to the Controller.



SVC_REQ 5: Change Background Task Window Mode and Timer Value

(PACSystems firmware version 3.50 and later.)

Operation

Use SVC_REQ 5 to change the Background Task Window mode and timer value. The change takes place during the next CPU sweep after the instruction is called.
SVC_REQ 5 sets the ENO output parameter to 1 (ON, TRUE) unless a mode other than 0 (Limited) or 2 (Run-to-Completion) is selected.
The parameter block has a length of one word.

To disable the Background Task window:

Enter SVC_REQ 5 with this parameter block:

Address	*High Byte*	*Low Byte*
Address	0	0

To enable the Background Task window mode:

Enter SVC_REQ 5 with this parameter block:

Address	*High Byte*	*Low Byte*
Address	Mode	1ms ≤ value ≤ 255ms

SVC_REQ 6: Change/Read Number of Words to Checksum

(PACSystems firmware version 3.50 and later.)

Operation

Use SVC_REQ 6 to read the current word count in the program to be checksummed or set a new word count. By default, 16 words are checked. The instruction is successful unless some number other than 0 or 1 is entered as the requested operation.
The parameter block has a length of 2 words.

To read the word count:

Enter a zero in the first word of the parameter block.

Address	0
Address + 1	Ignored

The instruction returns the current checksum (word count) in the second word of the parameter block. No range is specified for the read function; the value returned is the number of words currently being checksummed.

Address	0
Address + 1	Current word count

To set a new word count:

Enter a one in the first word of the parameter block and the new word count in the second word.

Address	1
Address + 1	New word count

The CPU changes the number of words to be checksummed to the value given in the second word of the parameter block, rounded up to the next multiple of 8. To disable checksumming, set the new word count to 0.

SVC_REQ 7: Read or Change the Time-of-Day Clock

(PACSystems firmware version 3.50 and later.)

Operation

Use SVC_REQ 7 to read or change the time of day clock in the CPU. The instruction is successful unless:

- An invalid number is entered for the requested operation.
- An invalid data format is specified.
- Data is provided in an unexpected format.

Parameter Block Formats

In the first two words of the parameter block, you specify whether to read or set the time and date, and which format to use.

Address	*2-Digit Year Format*	*4-Digit Year Format*
Address (word 1)	0 = read time and date	0 = read time and date
	1 = set time and date	1 = set time and date
Address+1 (word 2)	0 = numeric data format	80h = numeric data format
	1 = BCD format	81h = BCD format
	2 = unpacked BCD format	82h = unpacked BCD format
	3 = packed ASCII format (with embedded spaces and colons)	83h = packed ASCII format
	4 = POSIX format	n/a
Address+2 to the end (word 3)	Data	Data

Words 3 to the end of the parameter block contain output data returned by a read function, or new data being supplied by a change function. In both cases, format of these data words is the same. When reading the date and time, words (address + 2) to the end of the parameter block are ignored on input.

The format and length of the parameter block depends on the data format and number of digits required for the year:

Data Format and N-digit Year	*Length of parameter block (number of words)*
BCD, 2-digit year	6
BCD, 4-digit year	6
POSIX format	6
Unpacked BCD 2	9

Unpacked BCD 4	10
Numeric (2 and 4 digit year)	9
Packed ASCII, 2-digit year	12
Packed ASCII, 4-digit year	13

In any format,

- Hours are stored in 24-hour format.
- Day of the week is a numeric value ranging from 1 (Sunday) to 7 (Saturday).

Value	*Day of the Week*
1	Sunday
2	Monday
3	Tuesday
4	Wednesday
5	Thursday
6	Friday
7	Saturday

BCD, 2-Digit Year

In BCD format, each time and date item occupies one byte, so the parameter block has six words. The last byte of the sixth word is not used. When setting the date and time, this byte is ignored; when reading date and time, the instruction returns a null character (00).

Parameter Block Format	*Address*	*Example (Sun., July 3, 2005, at 2:45:30 p.m. = 14:45:30 in 24-hour format)*
1 = change or 0 = read	address	0 (read)
1 (BCD format)	address+1	1 (BCD format)

High Byte	*Low Byte*		*High Byte*	*Low Byte*
month	year	address+2	07 (July)	05 (year)
hours	day of month	address+3	14 (hours)	03 (day)
seconds	minutes	address+4	30 (seconds)	45 (minutes)
(null)	day of week	address+5	00	01 (Sunday)

BCD, 4-Digit Year

In this format, each time and date item occupies one byte, so the parameter block has six words. All bytes are used.

Parameter Block Format	*Address*	*Example (Sun., July 3, 2005, at 2:45:30 p.m. = 14:45:30 in 24-hour format)*
1 = change or 0 = read	address	00 (read)

81h (BCD format, 4-digit)	address+1		81h (BCD, 4-digit)	
High Byte	*Low Byte*		*High Byte*	*Low Byte*
year	year	address+2	20 (year)	05 (year)
day of month	month	address+3	03 (day)	07 (July)
minutes	hours	address+4	45 (minutes)	14 (hours)
day of week	seconds	address+5	01 (Sunday)	30 (seconds)

POSIX

The POSIX format of the Time-of-Day clock uses two signed 32-bit integers (two DINTs) to represent the number of seconds and nanoseconds since midnight January 1, 1970. Reading the clock in POSIX format might make it easier for your application to calculate time differences. This format can also be useful if your application communicates to other devices using the POSIX time format. To read and/or change the date and time using POSIX format, enter SVC_REQ 7 with this parameter block:

Parameter Block Format	*Address*	*Example* December 1st, 2000 at 12 noon
1 = change or 0 = read	address	0
4 (POSIX format)	address+1	4
Seconds (LSW)	address+2	975,672,000
(MSW)	address+3	
Nanoseconds (LSW)	address+4	0
(MSW)	address+5	

The resolution of the Controllers POSIX clock is 100 microseconds. The overall accuracy of the POSIX clock is +/- 0.01%. The accuracy of an individual sample of the POSIX clock is approximately 105 microseconds; however, the SVC_REQ instruction is not protected against operating system and user interrupts. The timing and length of these interrupts are unpredictable. As a result, the clock sample returned by SVC_REQ 7 can sometimes be much more than 105 microseconds old by the time execution is returned to the FBD logic. Interrupts can also delay when a new clock time is installed using SVC_REQ 7.

The CPU's maximum POSIX clock value is 7FFFFFFF (hexadecimal) seconds and 999,999,999 (decimal) nanoseconds, which corresponds to January 19th, 2038 at 3:14 am. This is the maximum POSIX value that SVC_REQ 7 can accept for changing the clock. This is also the maximum POSIX value SVC_REQ 7 can return after the Time-Of-Day clock passes this date.

If SVC_REQ 7 receives an invalid POSIX time to write to the clock, it does not change the Time-Of-Day clock and disables its data flow output.

Note: When reading the CPU clock in POSIX format, the data returned is not easy for someone to interpret. If desired, it is up to the application logic to convert the POSIX time into year, month, day of month, hour, and seconds.

Unpacked BCD (2-Digit Year)

In Unpacked BCD format, each digit of the time and date items occupies the low-order four bits of a byte. The upper four bits of each byte are always zero. This format requires nine words. Values are hexadecimal.

Parameter Block Format	*Address*	*Example (Thurs., Dec. 8, 2002, at 9:34:57 a.m.*
1 = change or 0 = read	address	0h
2 (Unpacked BCD format)	address+1	2h

High Byte	*Low Byte*		*High Byte*	*Low Byte*
	year	address+2	00h	02h
	month	address+3	01h	02h
	day of month	address+4	00h	08h
	hours	address+5	00h	09h
	minutes	address+6	03h	04h
	seconds	address+7	05h	07h
	day of week	address+8	00h	05h

Unpacked BCD (4-Digit Year)

In Unpacked BCD format, each digit of the time and date items occupies the low-order four bits of a byte. The upper four bits of each byte are always zero. This format requires nine words. Values are hexadecimal.

Parameter Block Format	*Address*	*Example (Thurs., Dec. 8, 2002, at 9:34:57 a.m.*
1 = change or 0 = read	address	0h
2 (Unpacked 4-digit BCD format)	address+1	2h

High Byte	*Low Byte*		*High Byte*	*Low Byte*
	year	address+2	20h	02h
	month	address+3	01h	02h
	day of month	address+4	00h	08h
	hours	address+5	00h	09h
	minutes	address+6	03h	04h
	seconds	address+7	05h	07h
	day of week	address+8	00h	05h

Numeric, 2-Digit Year

In numeric format, the year, month, day of month, hours, minutes, seconds and day of week each occupy one unsigned integer. To read and/or change the date and time using the numeric format, enter SVC_REQ 7 with this parameter block:

Parameter Block Format	*Address*	*Example Wed., June 15, 2005, at 12:15:30 a.m.*
1 = change or 0 = read	address	0
0 (Numeric format)	address+1	0

High Byte	*Low Byte*		*Value*
	year	address+2	05
	month	address+3	06
	day of month	address+4	15
	hours	address+5	12
	minutes	address+6	15
	seconds	address+7	30
	day of week	address+8	04

Numeric, 4-Digit Year

In numeric format, the year, month, day of month, hours, minutes, seconds and day of week each occupy one unsigned integer. To read and/or change the date and time using the numeric format, enter SVC_REQ 7 with this parameter block:

Parameter Block Format	*Address*	*Example Wed., June 15, 2005, at 12:15:30 a.m.*
1 = change or 0 = read	address	0
80h (Numeric format, 4-digit year)	address+1	80h

High Byte	*Low Byte*		*Value*
	year	address+2	2005
	month	address+3	06
	day of month	address+4	15
	hours	address+5	12
	minutes	address+6	15
	seconds	address+7	30
	day of week	address+8	04

Packed ASCII, 2-Digit Year

In Packed ASCII format, each digit of the time and date items is an ASCII formatted byte. Spaces and colons are embedded into the data to format it for printing or display. ASCII format for a 2-digit year requires 12 words in the parameter block.

Parameter Block Format	*Address*	*Example (Mon., Oct. 5, 2005, at 11:13:25 p.m. = 23:13:25 in 24-hour format)*
1 = change or 0 = read	address	0h (read)
3 (ASCII format)	address+1	3h (ASCII format)

High Byte	*Low Byte*		*High Byte*	*Low Byte*
year	year	address+2	35h (5)	30h (0)
month	(space)	address+3	31h (1)	20h (space)
(space)	month	address+4	20h (space)	30h (0)
day of month	day of month	address+5	35h (5)	30h (leading 0)
hours	(space)	address+6	32h (2)	20h (space)
: (colon)	hours	address+7	3Ah (:)	33h (3)
minutes	minutes	address+8	33h (3)	31h (1)
seconds	: (colon)	address+9	32h (2)	3Ah (:)
(space)	seconds	address+10	20h (space)	35h (5)
day of week	day of week	address+11	32h (2 = Mon.)	30h (leading 0)

Packed ASCII, 4-Digit Year

In Packed ASCII format, each digit of the time and date items is an ASCII formatted byte. Spaces and colons are embedded into the data to format it for printing or display. ASCII format for a 4-digit year requires 13 words in the parameter block.

Parameter Block Format	*Address*	*Example (Mon., Oct. 5, 2005, at 11:13:25 p.m. = 23:13:25 in 24-hour format)*
1 = change or 0 = read	address	0 (read)
83h (ASCII 4 digit)	address+1	83h (ASCII 4 digit)

High Byte	*Low Byte*		*High Byte*	*Low Byte*
year (hundreds)	year (thousands)	address+2	30 (0)	32 (2)
year (ones)	year (tens)	address+3	35 (5)	30 (0)
month (tens)	(space)	address+4	31 (1)	20 (space)
(space)	month (ones)	address+5	20 (space)	30 (0)
day of month (ones)	day of month (tens)	address+6	35 (5)	30 (leading 0)
hours (tens)	(space)	address+7	32 (2)	20 (space)
: (colon)	hours (ones)	address+8	3A (:)	33 (3)
minutes (ones)	minutes (tens)	address+9	33 (3)	31 (1)
seconds (tens)	: (colon)	address+10	32 (2)	3A (:)
(space)	seconds (ones)	address+11	20 (space)	35 (5)

day of week (ones)	day of week (tens)	address+12	32 (2 = Mon.)	30 (leading 0)

SVC_REQ 8: Reset Watchdog Timer

(PACSystems firmware version 3.50 and later.)

Operation

Use SVC_REQ 8 to reset the watchdog timer during the scan.
Ordinarily, when the watchdog timer expires, the CPU shuts down without warning. SVC_REQ 8 allows the timer to keep going during a time-consuming task (for example, while waiting for a response from a communications line).

Warning: Ensure that resetting the watchdog timer does not adversely affect the controlled process.

SVC_REQ 8 has no associated parameter block; however, you must still specify a dummy parameter, which SVC_REQ 8 will not use.

SVC_REQ 9: Read Sweep Time from Beginning of Sweep

(PACSystems firmware version 3.50 and later.)

Operation

Use SVC_REQ 9 to read the time in milliseconds since the start of the scan. The data is unsigned 16-bit integer.

Note: For a higher resolution, in nanoseconds, use SVC_REQ 51.

Output

The parameter block is an output parameter block only; it has a length of one word.

address	time since start of scan

SVC_REQ 10: Read Folder Name

(PACSystems firmware version 3.50 and later.)

Operation

Use SVC_REQ 10 to read the name of the currently executing project.

Output

The output parameter block has a length of four words. It returns eight ASCII characters: the program name (from one to seven characters) followed by null characters (00h). The last character is always a null character. If the program name has fewer than seven characters, null characters are appended to the end.

Address	*Low Byte*	*High Byte*
address	character 1	character 2
address+1	character 3	character 4
address+2	character 5	character 6
address+3	character 7	00

SVC_REQ 11: Read Controller ID

(PACSystems firmware version 3.50 and later.)

Operation

Use SVC_REQ 11 to read the name of the Controller executing the program.

Output

The output parameter block has a length of four words. It returns eight ASCII characters: the Controller ID (from one to seven characters) followed by null characters (00h). The last character is always a null character. If the Controller ID has fewer than seven characters, null characters are appended to the end.

Address	*Low Byte*	*High Byte*
address	character 1	character 2
address+1	character 3	character 4
address+2	character 5	character 6
address+3	character 7	00

SVC_REQ 12: Read Controller Run State

(PACSystems firmware version 3.50 and later.)

Operation

Use SVC_REQ 12 to read the current RUN state of the CPU.

Output

The parameter block is an output parameter block only; it has a length of one word.

address	1 = run/disabled
	2 = run/enabled

SVC_REQ 13: Shut Down (Stop) CPU

(PACSystems firmware version 3.50 and later.)

Operation

Use SVC_REQ 13 to stop the CPU after the specified number of scans has been performed. All outputs go to their designated default states at the start of the next CPU scan. An informational "Shut Down Controller" fault is placed in the Controller Fault table. The I/O scan continues as configured.

Input

The parameter block has a length of 1 WORD.

Address	0 = the PACSystems CPU transitions to Stop mode at the start of next scan.
	1 through 5 = the PACSystems finishes executing this scan, then executes this number of scans, and then transitions to Stop mode at the start of the following scan.
	-1 = the PACSystems CPU finishes executing this scan, then executes the number of scans configured in the Number of Last Scans parameter in the CPU's Scan tab, and then transition to Stop mode at the start of the following scan.

The CPU may execute fewer scans than specified in this parameter, for example, if a fatal error or another SVC_REQ 13 with a smaller value is encountered.

SVC_REQ 14: Clear Controller or I/O Fault Table

(PACSystems firmware version 3.50 and later.)

Operation

Use SVC_REQ 14 to clear either the Controller Fault table or the I/O fault table. The SVC_REQ output is set to ON unless some number other than 0 or 1 is entered as the requested operation.

The parameter block has a length of 1 WORD. It is an input parameter block only. There is no output parameter block.

Address	0 = clear Controller Fault table
	1 = clear I/O fault table

SVC_REQ 15: Read Last-Logged Fault Table Entry

(PACSystems firmware version 3.50 and later.)

Operation

Use SVC_REQ 15 to read the last entry logged in either the Controller fault table or the I/O fault table. The SVC_REQ output parameter ENO is set to ON unless some invalid number is entered as the requested operation or the fault table is empty.
The parameter block has a length of 22 WORDs. The first WORD is used only for input, and the other WORDs only for output.
The input parameter block is as follows:

Address	*Format*
Address	0 = Read Controller Fault table
	1 = Read I/O fault table
	80h = Read extended Controller Fault table
	81h = Read extended I/O fault table

Output

The format of the output parameter block depends on whether SVC_REQ 15 reads the Controller Fault table or the I/O fault table, or the extended Controller fault table or the extended I/O fault table.
The first word of the parameter block is used for input. The following words are used for output.

Controller Fault table Output Format		*Address*	*I/O Fault Table Output Format*	
High Byte	*Low Byte*		*High Byte*	*Low Byte*
spare	long/short	address+1	memory type	long/short
spare	spare	address+2		offset
slot	rack	address+3	slot	rack
	task	address+4	block	bus
fault action	fault group	address+5		point
	error code	address+6	fault action	fault category
	fault specific data	address+7	fault type	fault category
		address+8 to address+18	fault specific data	fault description
minutes	seconds	address+19	minutes	seconds
day of month	hour	address+20	day of month	hour

year	month	address+21	year	month
milliseconds (extended format only)		address+22	milliseconds (extended format only)	
reserved (extended format only)		address+23	reserved (extended format only)	

Long/Short Value

The first byte (low byte) of word *address +1* contains a number that indicates the length of the fault-specific data in the fault entry. These possible values are:

Controller Fault table	00 = 8 bytes (short)
Extended Controller Fault table	01 = 24 bytes (long)
I/O fault table	02 = 5 bytes (short)
Extended I/O fault table	03 = 21 bytes (long)

Note: The short I/O fault table is not available. The value returned for the I/O fault table is always 03 = 21 bytes (long).

SVC_REQ 16: Read Elapsed Time Clock

(PACSystems firmware version 3.50 and later.)

Operation

Use SVC_REQ 16 to read the system's elapsed time clock. The elapsed time clock measures the time in seconds since the CPU was powered on.
The parameter block has a length of three words used for output only.
The resolution of the Controller's elapsed time clock is 100 microseconds. The overall accuracy of the elapsed time clock is +/- 0.01%. The accuracy of an individual sample of the elapsed time clock is approximately 105 microseconds.

Warning: The SVC_REQ instruction is not protected against operating system and user interrupts. The timing and length of these interrupts are unpredictable. The clock sample returned by SVC_REQ 16 can sometimes be much more than 105 microseconds old by the time execution is returned to the FBD logic.

Tip: (PACSystems only.) For higher resolution, in nanoseconds, use SVC_REQ 50.

Output

address	Seconds from power on (low order)
address+1	Seconds from power on (high order)
address+2	100 microsecond ticks

The first two words are the elapsed time in seconds. The last word is the number of 100 microsecond ticks in the current second.

Warning: The SVC_REQ instruction is not protected against operating system and user interrupts. The timing and length of these interrupts are unpredictable. The clock sample returned by SVC_REQ 16 can sometimes be much more than 105 microseconds old by the time execution is returned to the FBD logic.

SVC_REQ 17: Mask/Unmask I/O Interrupt

(PACSystems firmware version 3.50 and later.)

Operation

Use SVC_REQ 17 to mask or unmask an interrupt from an input board. When an interrupt is masked, the CPU does not execute the corresponding interrupt block when the input transitions and causes an interrupt.
The parameter block is an input parameter block only; it has a length of three words.

address	0 = unmask input 1 = mask input
address+1	memory type
address+2	reference (offset)

"Memory type" is a decimal number that resides in the low byte of word *address + 1*. It corresponds to the memory type of the input:

70	%I memory in bit mode
10	%AI memory

Successful execution occurs unless:

- Some number other than 0 or 1 is entered as the requested operation.
- The memory type of the input to be masked or unmasked is not %I or %AI memory.
- The I/O board is not a supported input module.
- The reference address specified does not correspond to a valid interrupt trigger reference.
- The specified channel does not have its interrupt enabled in the configuration.

Masking / Unmasking Module Interrupts

During module configuration, interrupts from a module can be enabled or disabled. If a module's interrupt is disabled, it cannot be used to trigger logic execution in the application program and it cannot be unmasked. However, if an interrupt is enabled in the configuration, it can be dynamically masked or unmasked by the application program during system operation.
For the application program to mask and unmask interrupts that are enabled, use SVC_REQ 17. To mask or unmask an interrupt from an open VME module, the application logic should pass VME_INT_ID (17 decimal, 11H) as the memory type and the VME interrupt id as the offset to SVC_REQ 17.
When the interrupt is not masked, the CPU processes the interrupt and schedules the associated program logic for execution. When the interrupt is masked, the CPU processes the interrupt but does not schedule the associated program logic for execution.
When the CPU transitions from Stop to Run, the interrupt is unmasked.

For additional information on configuring and using VME module interrupts in a PACSystems RX7i control system, refer to *PACSystems RX7i User's Guide to Integration of VME Modules* (GFK-2235).

SVC_REQ 18: Read I/O Forced Status

(PACSystems firmware version 3.50 and later.)

Operation

Use SVC_REQ 18 to read the current status of forced values in the CPU's %I and %Q memory areas.

Note: SVC_REQ 18 does not detect forced values in %G or %M memory types. Use %S0011 (#OVR_PRE) to detect forced values in %I, %Q, %G, and/or %M memory types.

The parameter block has a length of one word used for output only.

Output

address	0 = No forced values are set
	1 = Forced values are set

SVC_REQ 19: Set Run Enable/Disable

(PACSystems firmware version 3.50 and later.)

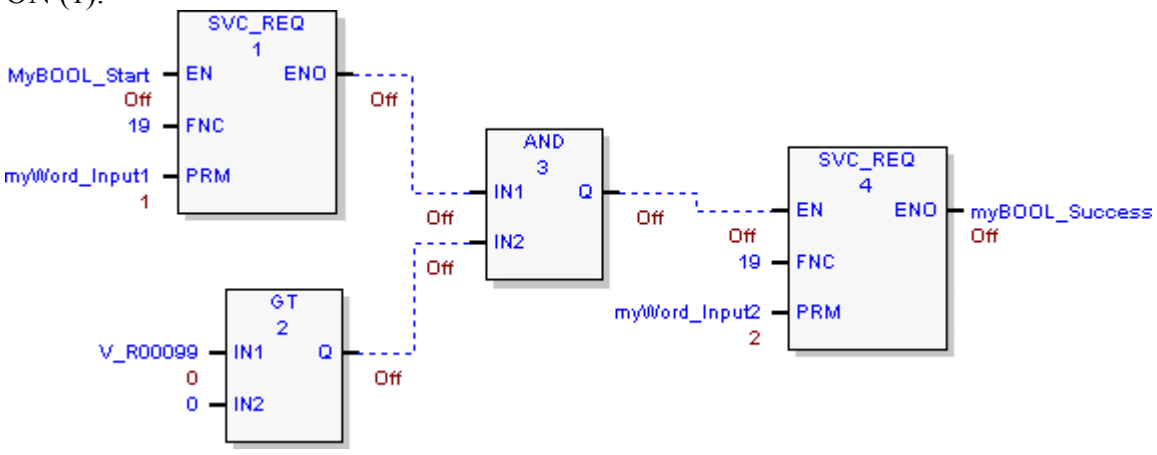
Operation

Use SVC_REQ 19 to enable FBD logic to control the RUN mode of the CPU.
The parameter passed indicates which function to perform. The ENO output parameter is set to ON if the instruction executes successfully. It is set to OFF if the requested operation is not SET RUN DISABLE mode (1) or SET RUN ENABLE mode (2).
The parameter block is an input parameter block only with this format:

address	1 = SET RUN DISABLE mode 2 = SET RUN ENABLE mode

Example

When MyBOOL_Start is set to ON (1), the RUN DISABLE mode is set, because myWord_Input1 is set to 1.
When the first SVC_REQ instruction has completed normally, and register %R00099 is greater than zero, the mode changes to RUN ENABLE, because myWord_Input2 is set to 2.
When the second SVC_REQ instruction completes normally, myBOOL_Success is set to ON (1).



SVC_REQ 20: Read Fault Tables

Operation

Use SVC_REQ 20 to retrieve the entire Controller or I/O fault table and return it to designated registers, from where they are accessible to FBD logic.
The first input parameter designates which table is to be read. A second input parameter (always zero for the standard Read Fault Tables) is used by the extended format to read a designated fault entry or to read a range of fault entries. The fault table data is placed in the parameter block following the input parameters.
The ENO output parameter is set to ON if the instruction executes successfully. It is set to OFF if the requested operation is not Read Controller Fault table (00h), Read I/O Fault Table (01h), Read Extended Controller Fault table (80h), or Read Extended I/O Fault Table (81h), or if there is not enough of the specified memory reference to hold the fault data. If the specified fault table is empty, the function sets the ENO output to ON, but returns only the fault table header information.
The parameter block is an input and output parameter block. The parameter block comes in two formats:

- Non-Extended (Read Controller Fault Table (00h), Read I/O Fault Table (01h))
- Extended (Read Extended Controller Fault Table (80h), or Read Extended I/O Fault Table (81h))

Non-Extended Formats

For non-extended formats (Read Controller Fault table (00h), Read I/O Fault Table (01h)), SVC_REQ 20 requires 693 registers available.

Format of the input parameter block

Address	Value
address	00h = Read Controller Fault table 01h = Read I/O fault table
address+1	Always zero (0)

Format of the output parameter block

Address	High Byte	Low Byte
address	Unused	0 = Controller Fault table 1 = I/O fault table
address+1	Unused	Always zero (0)
address+2 through address+14	Unused	Unused
address+15 through	Time since last clear, in BCD format:	

address+17		
address+15	Minutes	Seconds
address+16	Day of Month	Hour
address+17	Year	Month
address+18	Number of faults since last clear	
address+19	Number of faults in queue	
address+20	Number of faults read	
address+21	Start of fault data	

For the non-extended formats, each fault table entry is 21 words long (42 bytes). This request returns 16 Controller Fault table entries or 32 I/O fault table entries, or the actual number of faults if it is fewer. If the fault table read is empty, no data is returned.
The following table shows the return format of both a Controller Fault table entry and an I/O fault table entry.

Format of Returned Data for Fault Table Entries

Address	**Controller Fault table**		**I/O Fault Table**	
	High Byte	**Low Byte**	**High Byte**	**Low Byte**
address+21	Unused	Long/Short: number of bytes of fault extra data in the fault entry: ▪ 00: 8 bytes ▪ 01: 24 bytes	Memory type	Long/Short: number of bytes of fault extra data in the fault entry: ▪ 02: 5 bytes ▪ 03: 21 bytes
address+22	Unused	Spare	Offset	
address+23	Slot	Rack	Slot	Rack
address+24	Task		Bus address	I/O Bus Number (block)
address+25	Fault action	Fault group	Point	
address+26	Error code		Fault action	Fault group
address+27	Fault extra data		Fault type	Fault category
address+28			Fault extra data	Fault description
address+29 through address+38			Fault extra data	
address+39 through address+41	Time stamp in BCD format:		Time stamp in BCD format:	

address+39	Minutes	Seconds	Minutes	Seconds
address+40	Day of month	Hour	Day of month	Hour
address+41	Year	Month	Year	Month
address+42	Start of next fault output parameter block			

Extended Formats

Each extended format request can read a maximum of 64 faults, or the size of the fault table if it contains less than 64 faults.
For extended formats (Read Extended Controller Fault table (80h), or Read Extended I/O Fault Table (81h)), the Controller calculates the number of entries being read. You must ensure that enough registers are available to receive the amount of fault entries requested. If the amount of data requested exceeds the registers available, the CPU returns a fault indicating that reference memory is out of range.
The total size of the fault table for the extended fault format is
Header Size + ((# fault entries) * (size of fault entry))

Format of the input parameter block

Address	**Value**
address	80h = Read extended Controller Fault table 81h = Read extended I/O fault table
address+1	Starting index of faults to be read
address+2	Number of faults to be read

Format of the output parameter block

Address	**High Byte**	**Low Byte**
address	Unused	80h = Extended Controller Fault table 81h = Extended I/O fault table
address+1	Starting index of faults to be read	
address+2	Number of faults to be read	
address+3 through address+14	Unused	
address+15 through address+17	Time since last clear, in BCD format	
address+15	Minutes	Seconds
address+16	Day of month	Hour
address+17	Year	Month
address+18	Number of faults since last clear	
address+19	Number of faults in queue	

address+20	Number of faults read
address+21 through address+36	Unused
address+37	Start of fault data

For Read Extended Controller Fault table (80h) and Read Extended I/O Fault Table (81h), each extended fault table entry is 23 words long (46 bytes).

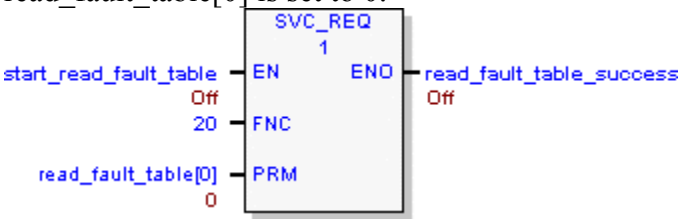
Format of Returned Data for Fault Table Entries

Address	**Controller Fault table**		**I/O Fault Table**	
	High Byte	**Low Byte**	**High Byte**	**Low Byte**
address+37	Unused	Long/Short: number of bytes of fault extra data in the fault entry: ▪ 00: 8 bytes ▪ 01: 24 bytes	Reference address memory type	Long/Short: number of bytes of fault extra data in the fault entry: ▪ 02: 5 bytes ▪ 03: 21 bytes
address+38	Unused		Reference address offset	
address+39	Slot	Rack	Slot	Rack
address+40	Task		Bus address	I/O Bus Number (block)
address+41	Fault action	Fault group	Point	
address+42	Error code		Fault action	Fault group
address+43	Fault extra data		Fault type	Fault category
address+44			Fault extra data	Fault description
address+45 through address+54			Fault extra data	
address+55 through address+58	Time stamp in BCD format:		Time stamp in BCD format:	
address+55	Minutes	Seconds	Minutes	Seconds
address+56	Day of month	Hour	Day of month	Hour
address+57	Year	Month	Year	Month
address+58	Milliseconds		Milliseconds	
address+59	Unused		Unused	
address+60	Start of next fault output parameter block			

Examples

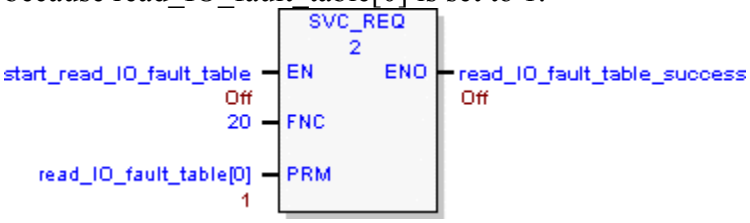
Example 1: Non-Extended Format

When start_read_fault_table is set to ON (1), the Controller Fault table is read, because read_fault_table[0] is set to 0.



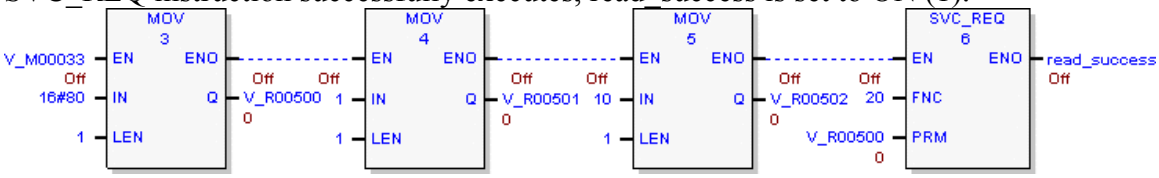
Example 2: Non_Extended Format

When start_read_IO_fault_table is set to ON (1), the Controller I/O fault table is read, because read_IO_fault_table[0] is set to 1.



Example 3: Extended Format

When input %M00033 is set to ON (1), the extended Controller Fault table is read. %R00500 contains the fault table type (Controller Extended); %Q00501 contains the starting index of faults to read; %R00502 contains the number of faults to read. When the SVC_REQ instruction successfully executes, read_success is set to ON (1).



SVC_REQ 21: User-Defined Fault Logging

(PACSystems firmware version 3.50 and later.)

Operation

Use SVC_REQ 21 to define a fault that can be displayed in the Controller fault table. The fault contains binary information or an ASCII message. The user-defined fault codes start at 0 hex.

The error code information for the fault must be within the range 0 to 2047 for an "Application Msg:" to be displayed. If the error code is in the range 81 to 112 decimal, the CPU sets a fault bit of the same number in %SA system memory. This allows up to 32 bits to be individually set.

Error Code	*Status Bit*
Errors 0–80	No bit set
Errors 81–112	Sets %SA
Errors 113–2047	No bit set
Errors 2048–32,767	Reserved

When the EN input parameter is set to ON (1, TRUE), the fault data array referenced by IN is logged as a fault to the Controller Fault table. If EN is set to OFF (0, FALSE), the ENO output parameter is set to OFF. If the error code is out of range, ENO is set to OFF and the fault is not logged as requested.

The parameter block is an input parameter block only with this format:

Parameter	*MSB*	*LSB*
address	Error code	
address+1	Text2	Text1
address+2	Text4	Text3
address+3	Text6	Text5
address+4	Text8	Text7
address+5	Text10	Text9
address+6	Text12	Text11
address+7	Text14	Text13
address+8	Text16	Text15
address+9	Text18	Text17
address+10	Text20	Text19
address+11	Text22	Text21
address+12	Text24	Text23

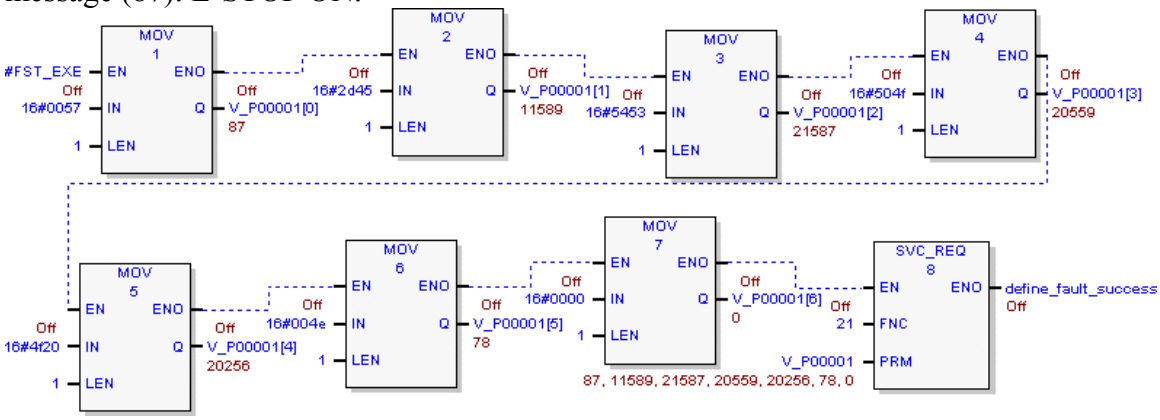
In the input parameter data, you can select an error code in the range 0 to 2047 and text information that will be placed in the fault extra data portion of a long Controller fault. The Controller fault address, fault group, and fault action are filled in by SVC_REQ. The fault text bytes 1 through 24 can be used to pass binary or ASCII data with the fault. If the first byte of the fault text data is non-zero, the data will be an ASCII message string. This message will then be displayed in the fault description area of the fault table. If the message is less than 24 characters, the ASCII string must be NULL byte-terminated. The programmer displays "Application Msg:" and the ASCII data is displayed as a message immediately following "Application Msg:". If the error code is between 1 and 2047, the error code number is displayed immediately after "Msg" in the "Application Msg:" string. If the error code is greater than 2047, SVC_REQ 21 is ignored and sets its ENO output parameter to 0 (OFF, FALSE).
If the first byte of text is zero, then only "Application Msg:" is displayed in the fault description. The next 1-23 bytes will be considered binary data for user data logging. This data can be displayed.

Note: When a user-defined fault is displayed in the Controller Fault table, a value of -32,768 (8000 hex) is added to the error code. For example, the error code 5 is displayed as -32,763.

For more information, refer to chapter 5, "PLC Control and Status," in the *Programming Software User's Manual*, GFK-0263.

Example

The first MOV instruction moves our new fault error code 87 as hex 0057 into V_P00001[0]. Fault error code 87 is part of our new fault message. The values of the next inputs give the ASCII codes for the text of the error message. For the second MOV instruction, the input is 2D45. The low byte, 45, decodes to the letter E; the high byte, 2D, decodes to -. The string continues with S T O P O and N. The final character, 00, is the null character that terminates the string. In summary, the decoding yields the string message (87): E-STOP ON.



SVC_REQ 22: Mask/Unmask Timed Interrupts

(PACSystems firmware version 3.50 and later.)

Operation

Use SVC_REQ 22 to mask or unmask timed interrupts and to read the current mask. When the interrupts are masked, the CPU does not execute any timed interrupt block timed program that is associated with a timed interrupt. Timed interrupts are masked/unmasked as a group. They cannot be individually masked or unmasked. Successful execution occurs unless some number other than 0 or 1 is entered as the requested operation or mask value.
The parameter block is an input and output parameter block.
To determine the current mask, use this format:

address	0 = Read interrupt mask

The Controller returns this format:

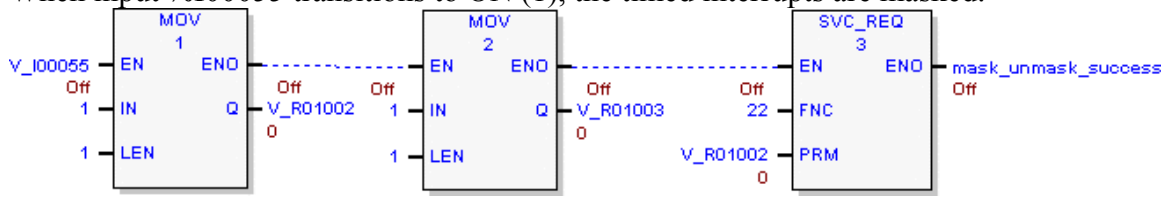
address	0 = Read interrupt mask
address+1	0 = Timed interrupts are unmasked 1 = Timed interrupts are masked

To change the current mask, use this format:

address	1 = Mask/unmask interrupts
address+1	0 = Unmask timed interrupts 1 = Mask timed interrupts

Example

When input %I00055 transitions to ON (1), the timed interrupts are masked.



SVC_REQ 23: Read Master Checksum

Operation

Use SVC_REQ 23 to read master checksums for the set of user program(s) and the configuration, and to read the checksum for the block from which the service request is made.
There is no input parameter block for this service request. The output parameter block requires 15 WORDs of memory.

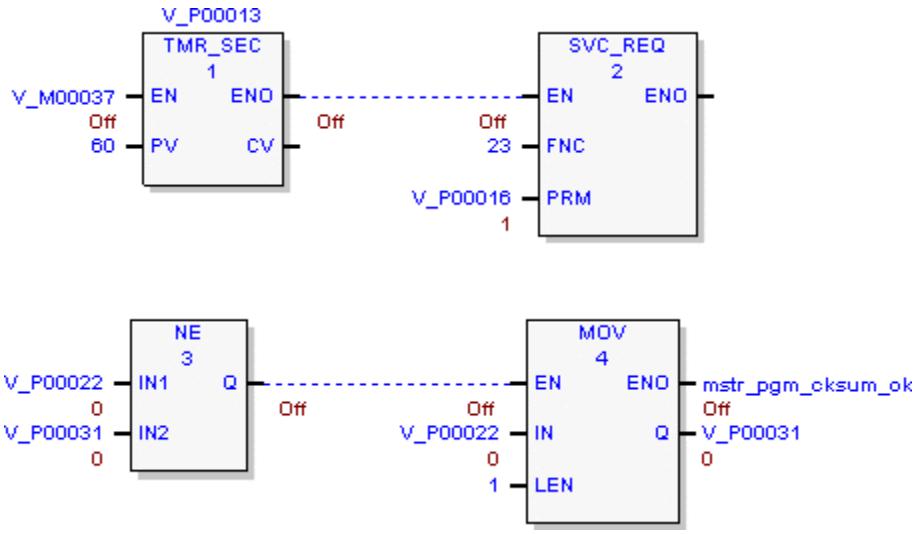
Output

When a Run Mode Store is active, the program checksums may not be valid until the store (download) is complete. To determine when checksums are valid, three flags (one each for Program Block Checksum, Master Program Checksum, and Master Configuration Checksum) are provided at the beginning of the output parameter block.

Address	*Description*
Address	Program Checksum Valid (0 = not valid, 1 = valid)
Address + 1	Master Program Checksum Valid (0 = not valid, 1 = valid)
Address + 2	Master Configuration Checksum Valid (0 = not valid, 1 = valid)
Address + 3	Number of LD/SFC Blocks (including _MAIN)
Address + 4	Size of User Program in Bytes (DWORD data type)
Address + 6	Program Set Additive Checksum
Address + 7	Program CRC Checksum (DWORD data type)
Address + 9	Size of Configuration Data in Bytes
Address + 10	Configuration Additive Checksum
Address + 11	Configuration CRC Checksum (DWORD data type)
Address + 13	high byte: always zero low byte: Currently Executing Block's Additive Checksum
Address + 14	Currently Executing Block's CRC Checksum

Example

When the timer using registers %P00013 through %P00015 expires, the checksum read is performed. The checksum data returns in registers %P00016 through %P00030. The master program checksum in registers %P00022 and %P00023 (the program checksum is a DWORD data type and occupies two adjacent registers) is compared with the last saved master program checksum. If these are different, the current master program checksum is saved in registers %P00031 and %P00032.


V_P00013
TMR_SEC
1
V_M00037
EN
ENO
Off
60
PV
CV
Off
Off
SVC_REQ
2
EN
ENO
23
FNC
V_P00016
PRM
1
NE
3
V_P00022
IN1
Q
0
V_P00031
IN2
0
Off
Off
MOV
4
EN
ENO
mstr_pgm_cksum_ok
Off
V_P00022
IN
Q
V_P00031
0
LEN
1
0

SVC_REQ 24: Reset Smart Module

(PACSystems firmware version 3.50 and later.)

Operation

Use SVC_REQ 24 to reset a daughterboard or smart module. The SVC_REQ output parameter ENO is set to ON (1, TRUE) unless one of the following conditions exists:

- An invalid number for rack and/or slot is entered.
- There is no module at the specified location.
- The module at the specified location does not support a runtime reset.
- The CPU was unable to reset the module at the specified location.

For this function, the parameter block has a length of 1 WORD. It is an input parameter block only.

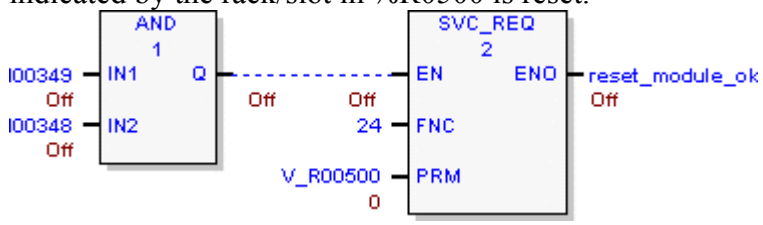
address	Module Slot (low byte)
	Module Rack (high byte)

Notes

- Rack 0, Slot 1 indicates that a reset is to be sent to the daughterboard.
- It is important to invoke SVC_REQ 24 for a given module for only one sweep at a time. Each time SVC_REQ 24 executes, the target module is reset again, regardless of whether it has finished starting up from a previous reset.
- PACSystems RX3i redundancy CPUs support SVC_REQ 24 to reset an IC695RMX128 module only if the Redundancy Link parameter on the Settings tab of the RMX128 is set to Disabled.

Example

The parameter block (set up elsewhere in logic) containing the module's rack and slot for this service request, is located at %R00500.
When input %I00349 is set to ON (1) and input %I00348 is set to ON (1), the module indicated by the rack/slot in %R0500 is reset.



SVC_REQ 25: Disable/Enable EXE Block and Standalone C Program Checksums

(PACSystems firmware version 3.50 and later.)

Operation

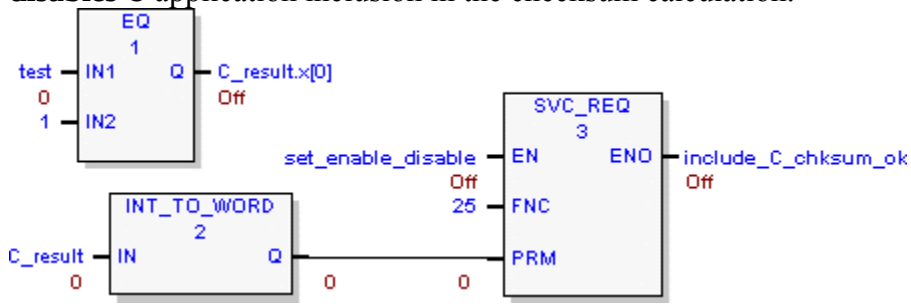
Use SVC_REQ 25 to enable or disable the inclusion of C blocks in the background checksum calculation. The default is to include the checksums.
This service request uses only an input parameter block.

address	0 = Disable C block inclusion in checksum calculation
	1 = Enable C block inclusion in checksum calculation

The parameter block is unchanged after execution of the service request.

Example

When the value of the INT variable 'test' is 1 (set elsewhere in logic), C_result.x[0] is set to ON (1), the data type of C_result is converted from INT to WORD, and svc_req 25 **enables** C application inclusion in the checksum calculation.
When the value of the INT variable 'test' is 0 (set elsewhere in logic), C_result.x[0] is set to OFF (0), the data type of C_result is converted from INT to WORD, and svc_req 25 **disables** C application inclusion in the checksum calculation.



SVC_REQ 26: Role Switch

(PACSystems RX7i redundancy CPUs with firmware version 3.50 or later; PACSystems RX3i redundancy CPUs with firmware version 5.70 or later.)

Operation

Use SVC_REQ 26 to cause the CPUs to switch roles on the next scan (active to backup and backup to active) if the CPUs are synchronized and the timing requirements of the role switch request are met. A manual role switch cannot occur within 10 seconds of a previous manual role switch. Role switches due to failures or resynchronization are always allowed (the 10-second limitation does not apply).

Note: The ENO output parameter from SVC_REQ 26 is set to 1 (ON, TRUE) to indicate that a role switch will be attempted on the next scan. The ON setting does not indicate that a role switch has occurred or that a role switch will occur on the next scan.

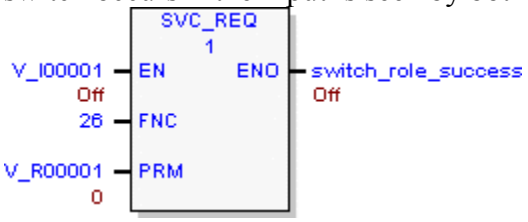
SVC_REQ 26 has no associated parameter block; however, the programming software requires that an entry be made for the PRM input parameter. Enter any appropriate reference here; it will not be used.

For more information about Hot Standby CPU Redundancy, refer to the *Series 90-70 Hot Standby CPU Redundancy User's Guide*, GFK-0827. For more information about Enhanced Hot Standby CPU Redundancy, refer to the *Series 90-70 Enhanced Hot Standby CPU Redundancy User's Guide*, GFK-1527.

Example

When %I00001 is set to ON(1), SVC_REQ 26 is activated, causing a role switch between CPUs.

The 10 second limitation allows this SVC_REQ to be in both CPUs, so that only a single switch occurs if the input is seen by both CPUs.



SVC_REQ 27: Write to Reverse Transfer Area

(PACSystems CRE models with firmware version 3.50 and later and all CRU models.) For all information about using this service request, refer to "Programming a Data Transfer from Backup Unit to Active Unit" in the *Series 90-70 Enhanced Hot Standby CPU Redundancy User's Guide* (GFK-1527).

SVC_REQ 28: Read from Reverse Transfer Area

(PACSystems CRE models with firmware version 3.50 and later and all CRU models.) For all information about using this service request, refer to "Programming a Data Transfer from Backup Unit to Active Unit" in the *Series 90-70 Enhanced Hot Standby CPU Redundancy User's Guide* (GFK-1527).

SVC_REQ 29: Read Elapsed Power Down Time

(PACSystems firmware version 3.50 and later.)

Operation

Use SVC_REQ 29 to read the amount of time elapsed between the last power-down and the most recent powerup. If the watchdog timer expired before power-down, the CPU cannot calculate the power down elapsed time, so the time is set to 0.
SVC_REQ 29 cannot be accessed from a C block.
SVC_REQ 29 always sets the output parameter ENO to ON (1, TRUE).
The parameter block has a length of three WORDs used for output only.

Output

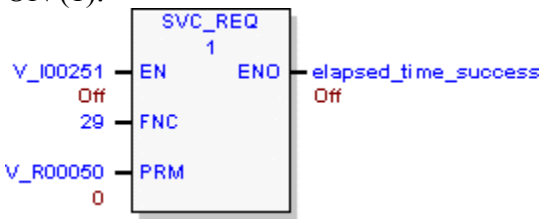
Address	*Description*
Address	Power-Down Elapsed Seconds (low order)
Address+1	Power-Down Elapsed Seconds (high order)
Address+2	100 microsecond ticks

If the watchdog timer expired before power-down, the CPU cannot calculate the power down elapsed time, so the time is set to 0.

Note: Although this request responds with a resolution of 100 microseconds, the actual accuracy is 1 second. The battery-backed clock used when the Controller is powered down is accurate to within 1 second.

Example

When input %I00251 is set to ON (1), the elapsed power-down time is placed into the parameter block that starts at %R00050. The variable elapsed_time_success is then set to ON (1).



SVC_REQ 32: Suspend/Resume I/O Interrupt

(PACSystems firmware version 3.50 and later.)

Operation

Use SVC_REQ 32 to suspend a set of I/O interrupts and cause occurrences of these interrupts to be queued until these interrupts are resumed. The number of I/O interrupts that can be queued depends on the I/O module's capabilities. The CPU informs the I/O module that its interrupts are to be suspended or resumed. The I/O module's default is resumed. The Suspend applies to all I/O interrupts associated with the I/O module. Interrupts are suspended and resumed within a single scan.
SVC_REQ 32 uses only an input parameter block. Its length is three WORDs.

Address	0 = resume interrupt 1 = suspend interrupt
Address + 1	memory type. Must be set to 70 (%I memory).
Address + 2	reference (offset)

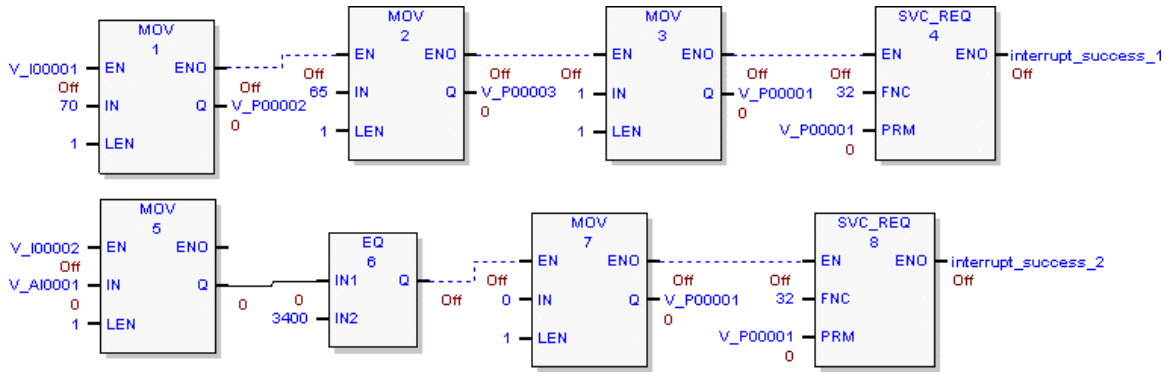
Successful execution occurs only if all of the following conditions are true:

- A 0 or 1 is passed as the first parameter.
- The memory type parameter is 70 (%I memory).
- The I/O module associated with the specified address is an appropriate module for this operation.
- For a Series 90-70 High Speed Counter, the reference address specified is the first %I reference set for the High Speed Counter.
- There is communication between the CPU and this I/O module; that is, the board is present, and the board has no fatal faults.

Example

Note: I/O interrupts, unless suspended or masked, can interrupt the execution of a function block. The most often used application of this Service Request is to prevent the effects of the interrupts for diagnostic or other purposes.

Interrupts from the high speed counter (HSC) module whose starting point reference address is %I00065 is suspended while the CPU solves the logic of the SVC_REQ with a solve order of 4.
Without the suspend, an interrupt from the HSC can occur during execution of the MOV instruction with a solve order of 5. If the output of this MOV instruction has a value of 3,400, then a 0 is moved into %P00001, and the logic of the SVC_REQ with a solve order of 8 is executed.
%AI0001 is the first non-discrete input reference for the High Speed Counter.


MOV
1
V_I00001
Off
70
1
EN
ENO
IN
Q
LEN
Off
V_P00002
0
MOV
2
Off
65
1
EN
ENO
IN
Q
LEN
Off
V_P00003
0
MOV
3
Off
1
1
EN
ENO
IN
Q
LEN
Off
V_P00001
0
SVC_REQ
4
Off
32
V_P00001
0
EN
ENO
FNC
PRM
interrupt_success_1
Off
MOV
5
V_I00002
Off
V_AI0001
0
1
EN
ENO
IN
Q
LEN
0
EQ
6
0
3400
IN1
IN2
Q
Off
MOV
7
Off
0
1
EN
ENO
IN
Q
LEN
Off
V_P00001
0
SVC_REQ
8
Off
32
V_P00001
0
EN
ENO
FNC
PRM
interrupt_success_2
Off

SVC_REQ 43: Disabling Data Transfer Copy in Backup Unit

(PACSystems CRE models with firmware version 3.50 or later and all CRU models.)

Operation

Use SVC_REQ 43 on the backup unit of a CPU redundancy system or CPU over Genius redundancy system to allow the backup unit to bypass the copy of the shared I/O data from the active unit. This operation can help determine if the active and backup CPUs are arriving at the same results.

SVC_REQ 43 is ignored if issued when the units are not synchronized, or if issued in the active unit.

SVC_REQ 43 disables the copy of data for one sweep beginning with the output data transfer and ending with the input data transfer of the next sweep. The copy can be disabled for multiple sweeps by invoking SVC_REQ 43 once each sweep for the appropriate number of sweeps.

The special resynchronization data transfer always occurs, even if SVC_REQ 43 is invoked in the first sweep after synchronization (this data transfer includes all shared inputs, all shared outputs, and the internal data that must be exchanged) since the resynchronization data transfer occurs before the start of logic execution.

SVC_REQ 43 can be set up to disable the copies for all transfers or just the output transfers. If just the output copy is disabled, the two units can still use the same set of inputs on each unit. This enables you to test the ability of the two units to derive the same results from the same inputs.

In all cases, the configured data transfers are still transferred over the link every sweep and the rendezvous points are still met. The effect of SVC_REQ 43 is to disable the copy of the data from the transfer to the actual reference memories configured.

Warning: When SVC_REQ 43 is in effect, the backup unit still takes control of the redundancy system in event of a failure or role switch. Switches to the backup unit may cause a momentary interruption of data of the outputs since the two units may not be generating the exact same results.

Consider disabling outputs on the backup unit while SVC_REQ 43 is in effect. Disabling outputs on the backup unit eliminates the risk of an unsynchronized switch of control (which can cause a momentary interruption of data in the outputs) if the active unit fails or loses power while the input/output copies are disabled. However, if the active unit does fail or lose power while outputs are disabled on the backup unit, the system's outputs will go to their default settings. A secondary effect of disabling outputs on the backup unit is that the unsynchronized fault action table is used by the active unit to determine which faults are fatal.

Note: If the CPU is already in RUN/ENABLED mode, a command to disable its outputs will not take effect until one sweep after the command is received. Therefore, disable the outputs at least one sweep before you enable SVC_REQ 43.

SVC_REQ 43 cannot be used to disable output data transfer on the Primary unit when outputs are enabled on the Primary Unit. If that is attempted, SVC_REQ 43 is rejected.

A fault is logged the first time SVC_REQ 43 is used as a warning that the Controllers are not completely synchronized.
The reverse data transfer, if any, is unaffected by SVC_REQ 43.
Enabling logic should be used with SVC_REQ 43. A contact with a non-transferred reference should be part of this enabling logic. That will allow SVC_REQ 43 to be activated or deactivated directly without being overwritten with the value from the active unit.
If SVC_REQ 43 is invoked multiple times in a single sweep, the last call is the one that determines the action taken.
SVC_REQ 43 has an input block and no output block.
Successful execution occurs unless:

- The values in the input block are out of range.
- SVC_REQ 43 was invoked when the redundancy units in a CPU redundancy system or CPU over Genius redundancy system were not synchronized.
- SVC_REQ 43 was issued to the active unit.
- The CPU does not support SVC_REQ 43.

If SVC_REQ 43 is unsuccessful, the ENO output parameter is set to OFF (0, FALSE).

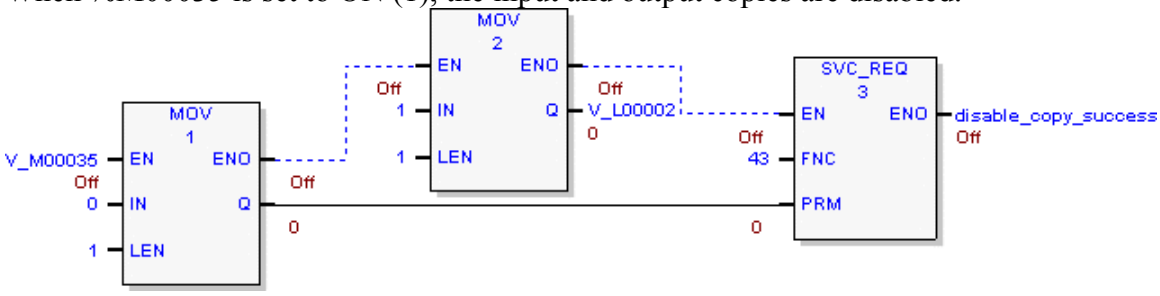
Input

The parameter block has a length of 2 WORDs.

Address	0
Address+1	1 = disable input and output copies 2 = disable output copy only

Example

When %M00035 is set to ON (1), the input and output copies are disabled.



Validating the backup unit

You can use SVC_REQ 43 to validate the input scan, that is, to determine if the backup Controller unit is collecting inputs properly. You can also use it to validate the logic solution, that is, to determine whether the backup Controller unit is calculating outputs and internal variables properly.

To validate the backup unit's input scan:

1. Activate SVC_REQ 43 on the backup unit, passing the values 0 and 1 to disable the input and output data transfer copies.

2. Observe the backup unit's %I and %AI reference tables. The values in these tables correspond to the inputs that the backup is currently collecting.
3. Visually compare the backup unit's %I and %AI reference tables with the active unit's tables. Pay special attention to the %I and %AI references that are configured to be shared between the two units.
4. When you are satisfied that the backup unit is collecting inputs properly, disable the rung that calls SVC_REQ 43.

To validate the backup unit's logic solution:

1. Activate SVC_REQ 43 on the backup unit, passing the values 0 and 2 to disable the output data transfer copy.
2. Observe the backup unit's %Q, %AQ, %M, and %R reference tables. The values in these tables correspond to the outputs that the backup is currently calculating.
3. Visually compare the backup unit's %Q, %AQ, %M, and %R reference tables with the active unit's tables. Pay special attention to the %Q, %AQ, %M, and %R references that are configured to be shared between the two units.
4. When you are satisfied that the backup unit is calculating outputs and internal variables properly, disable the rung that calls SVC_REQ 43.

SVC_REQ 45: Skip Next Output and Input Scan (Suspend I/O)

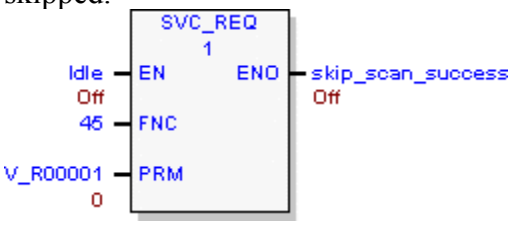
(PACSystems RX3i firmware version 3.50 and later; only for use with the DSM324i and Motion Mate DSM314. This service request is supported by PACSystems RX3i for easy conversion of Series 90-30 applications. The Suspend I/O (SUS_IO) instruction, which is supported by all PACSystems firmware versions, should be used in new applications.)

Operation

Use SVC_REQ 45 to skip the next output and input scans. Any changes to the output reference tables during the scan in which the SVC_REQ 45 was executed is not reflected on the physical outputs of the corresponding modules. Any change to the physical input data on the modules is not reflected in the corresponding input references during the scan after the one in which the SVC_REQ 45 was executed.
SVC_REQ 45 has no parameter block.
The DO_IO instruction is not affected by the use of SVC_REQ 45. It still updates the I/O when used in the same logic program as SVC_REQ 45.

Example

When the variable named "Idle" is set to ON (1), the next output and input scans are skipped.



SVC_REQ 50: Read Elapsed Time Clock (Two DWORDs)

(PACSystems firmware version 3.50 and later.)

Operation

Use SVC_REQ 50 to read the system's elapsed time clock. The elapsed time clock measures the time since the CPU was powered on and SVC_REQ 50 expresses it in seconds and nanoseconds.
The parameter block has a length of two DWORDs (four words) used for output only. The resolution of the Controller's elapsed time clock may vary depending on the hardware platform. The overall accuracy of the elapsed time clock is +/- 0.01%. The accuracy of an individual sample of the elapsed time clock may be approximately 105 microseconds.

Warning: The SVC_REQ instruction is not protected against operating system and user interrupts. The timing and length of these interrupts are unpredictable. The clock sample returned by SVC_REQ 50 can sometimes be much more than 105 microseconds old by the time execution is returned to the LD logic.

Output

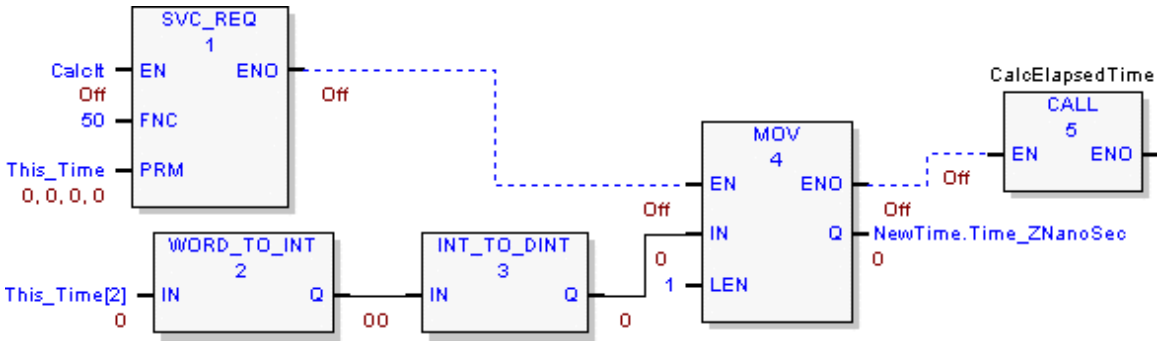
address	Seconds from power on (low order)
address+1	Seconds from power on (high order)
address+2	Approximate nanosecond ticks (low order)
address+3	Approximate nanosecond ticks (high order)

The first two words contain the elapsed time in seconds. The last two words contain the approximate number of nanoseconds in the current second. (The hardware does not count nanoseconds. It may count 100 microsecond ticks that are expressed as nanoseconds.)

Warning: The SVC_REQ instruction is not protected against operating system and user interrupts. The timing and length of these interrupts are unpredictable. The clock sample returned by SVC_REQ 50 can sometimes be much more than 105 microseconds old by the time execution is returned to the LD logic.

Example

Service Request 50 Read Elapsed Time Clock reads time since power on. Seconds is a DINT in the first two words. Nanoseconds is a DINT in the next two words. It differs from SVC_REQ 16 in that this SVC_REQ returns nanoseconds, while SVC_REQ 16 returns 100 microseconds tics.
This code snippit calculates the time between one call to the service request and the next. It runs when the variable named CalcIt is set to TRUE (1).



The ST program named CalcElapsedTime is called from FBD to calculate elapsed time. ST is a better language for calculations than either RLD or FBD.

```
Time_Dif_Nanosec := NewTime.Time_ZNanoSec - OldTime.Time_ZNanoSec;

IF Time_Dif_Nanosec < 0 THEN 'we rolled over the seconds since last scan
   Time_Dif_Nanosec := Time_Dif_Nanosec + 1000000000;
END_IF;

'Save NewTime in OldTime
OldTime.Time_Sec := NewTime.Time_Sec;
OldTime.Time_ZNanoSec := NewTime.Time_ZNanoSec;
```

SVC_REQ 51: Read Sweep Time from Beginning of Sweep (DWORD)

(PACSystems firmware version 3.50 and later.)

Operation

Use SVC_REQ 51 to read the time (expressed in nanoseconds) since the start of the scan. The data is DWORD (two words, that is, four bytes).

Output

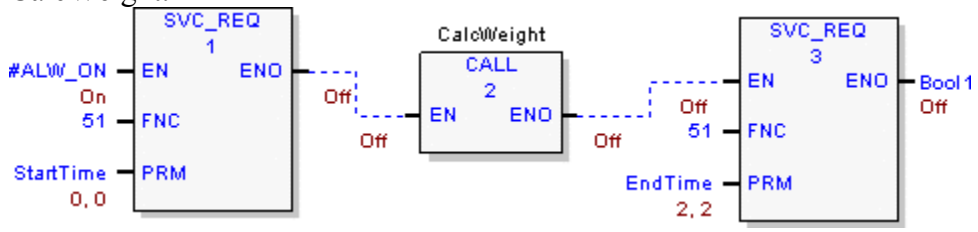
The parameter block is an output parameter block only; it has a length of two words (one DWORD). The time is expressed in nanoseconds, but the actual resolution is closer to 100 microsecond ticks and may vary depending on the hardware platform.

address	time since start of scan (low order)
address + 1	time since start of scan (high order)

If the time does not fit into a DWORD, the value is clamped at 0xFFFFFFFF and the ENO output parameter is set to OFF (0, FALSE).

Example

Use SVC_REQ 51 to determine the time required to execute the subroutine named CalcWeight.



SVC_REQ 55: Set Application Redundancy Mode

CPU Support

SVC_REQ 55 is supported for PACSystems non-HSB CPUs with firmware version 5.00 or later.

Operation

Use SVC_REQ 55 with a non-HSB CPU to request the CPU to send a redundancy role switch command to all Ethernet interfaces in the Controller that are configured for redundant IP operation.

Your application must monitor the LAN Interface Status (LIS) word for each Ethernet interface to determine whether the Redundant IP address is active at that interface.

SVC_REQ 55 is recognized in non-HSB CPUs only. SVC_REQ 55 has no effect on Ethernet interfaces that are not configured for redundant IP operation.

Input

The input parameter block has a length of one word (two bytes).

address	0 = Backup redundancy role 1 = Active redundancy role

SVC_REQ 56: Read from Nonvolatile Storage

CPU Support

SVC_REQ 56 is supported for PACSystems with firmware version 6.00 or later.

Operation

PACSystems Controllers support a 64KB nonvolatile flash memory area, which can be accessed by the logic-driven read/write service requests. Values are stored in the nonvolatile storage area by means of SVC_REQ 57. These values are applied to the Controller user memory on powerup.

If you want only to write to nonvolatile storage and have the values restored on a power cycle, you may not need to use SVC_REQ 56. However, a logic driven read from nonvolatile storage can be commanded as needed. For example, you can use #FST_SCN with SVC_REQ 56 calls to force a reload on each Stop to Run transition.

SVC_REQ 56 specifies a read operation from nonvolatile storage when the PACSystems is running. You can specify which reference address range to read and optionally a different destination memory location in CPU memory in which to place the read data. Using different memory locations enables you to set up a comparison between existing values in CPU memory with values in nonvolatile storage.

SVC_REQ 56 execution time varies depending on the number of values stored in nonvolatile storage, because it finds the most recent value for the requested reference address range.

Note: You can also read flash memory on a stopped Controller.

Data length

You can read up to 32 words (64 bytes) inclusively per invocation of SVC_REQ 56.

Discrete memory

Discrete memory can be read as individual bits or as bytes. For more information, see address.

If a discrete memory destination is forced, the forced value remains intact in CPU memory even though the count in word 10 (address + 10) indicates that all the data was read and transferred.

Transitions are affected. If a memory location has an associated transition bit and SVC_REQ 56 causes a transition on that value, the transition bit is set.

Maximum of one active instruction

When SVC_REQ 56 is active, it does not support an interrupt that attempts to activate SVC_REQ 57 or a second instance of SVC_REQ 56. If an attempt fails, an error indicating that another instance is active will be returned.

Storage disabled conditions

By default, the following write operations disable SVC_REQ 56 until logic is written to nonvolatile storage:

- Run Mode Store (RMS), even if a second RMS reverts everything to the original state
- Test-Edit session, even when you cancel your edits
- Word-for-word change
- Downloading to RAM only of a stopped PACSystems CPU, even if the downloaded contents are equal to the contents already on the nonvolatile storage

Setting input word 8 (address + 7) to a value of 1 enables SVC_REQ 56 despite the above conditions.

ENO and power flow to the right

If the status is Success or Partial Read (see address+9), then on the SVC_REQ instruction, ENO is set to True in FBD and ST, and power flow passes to the right in LD.

Input and Status

The parameter block has a length of 11 words (22 bytes). The first nine words contain the input you supply for the read operation. The last two words contain the status of the read operation.

<table>
<tr><td>address</td><td>Source memory, that is, the memory area to read from nonvolatile storage.
<table>
<tr><th>For this memory type:</th><th>Enter this decimal value:</th></tr>
<tr><td>%R</td><td>8</td></tr>
<tr><td>%AI</td><td>10</td></tr>
<tr><td>%AQ</td><td>12</td></tr>
<tr><td>%I (byte mode)</td><td>16</td></tr>
<tr><td>%Q (byte mode)</td><td>18</td></tr>
<tr><td>%T (byte mode)</td><td>20</td></tr>
<tr><td>%M (byte mode)</td><td>22</td></tr>
<tr><td>%G (byte mode)</td><td>56</td></tr>
<tr><td>%I (bit mode)</td><td>70</td></tr>
<tr><td>%Q (bit mode)</td><td>72</td></tr>
<tr><td>%T (bit mode)</td><td>74</td></tr>
<tr><td>%M (bit mode)</td><td>76</td></tr>
<tr><td>%G (bit mode)</td><td>86</td></tr>
<tr><td>%W</td><td>196</td></tr>
</table></td></tr>
<tr><td>address + 1 and address + 2</td><td>The zero-based offset N to read from nonvolatile storage. Address + 1, the least significant word, contains the complete offset for any memory area except %W, which also requires the use of address + 2 for offsets greater than 65,535.
▪ For %I, %Q, %M, %T, and %G memory in byte mode, N = (Ra - 1) / 8, where Ra = one-based reference address. For example, to read</td></tr>
</table>

<table>
<tr><td></td><td>from the one-based bit reference address %T33, enter the byte offset 4: (33 - 1) / 8 = 4.
▪ For %W, %R, %AI, and %AQ memory, and for %I, %Q, %M, %T, and %G memory in bit mode, N = Ra - 1. For example, to read from the one-based reference address %R200, enter the zero-based reference offset 199; to read from %I73 in bit mode, enter offset 72. For memory in bit mode, the offset must be set on a byte boundary, that is, a number exactly divisible by 8: 0, 8, 16, 24, and so on.</td></tr>
<tr><td>address + 3</td><td>Length. The number of items to read from nonvolatile storage beginning at the reference address calculated from the offset defined at [address + 1 and address + 2]. The length can be one of the following:
<table>
<tr><th>Description</th><th>Valid range</th></tr>
<tr><td>The number of words (16-bit registers) to read from %W, %R, %AI, or %AQ nonvolatile storage</td><td>1 through 32 words</td></tr>
<tr><td>The number of bytes to read from %I, %Q, %M, %T, or %G in byte mode nonvolatile storage</td><td>1 through 64 bytes</td></tr>
<tr><td>The number of bits to read from %I, %Q, %M, %T, or %G in bit mode nonvolatile storage</td><td>1 through 512 bits in increments of 8 bits</td></tr>
</table>
The value must reside in the low byte of address + 3. The high byte must be set to zero.</td></tr>
<tr><td>address + 4</td><td>Destination memory. The CPU memory area to write the read data to. This does not need to be the same memory area as specified at [address]. Writing to a different memory area enables you to compare the values that were already in the CPU with the values read from nonvolatile storage.
<table>
<tr><th>For this memory type:</th><th>Enter this decimal value:</th></tr>
<tr><td>%R</td><td>8</td></tr>
<tr><td>%AI</td><td>10</td></tr>
<tr><td>%AQ</td><td>12</td></tr>
<tr><td>%I (byte mode)</td><td>16</td></tr>
<tr><td>%Q (byte mode)</td><td>18</td></tr>
<tr><td>%T (byte mode)</td><td>20</td></tr>
<tr><td>%M (byte mode)</td><td>22</td></tr>
<tr><td>%G (byte mode)</td><td>56</td></tr>
<tr><td>%I (bit mode)</td><td>70</td></tr>
<tr><td>%Q (bit mode)</td><td>72</td></tr>
<tr><td>%T (bit mode)</td><td>74</td></tr>
</table></td></tr>
</table>

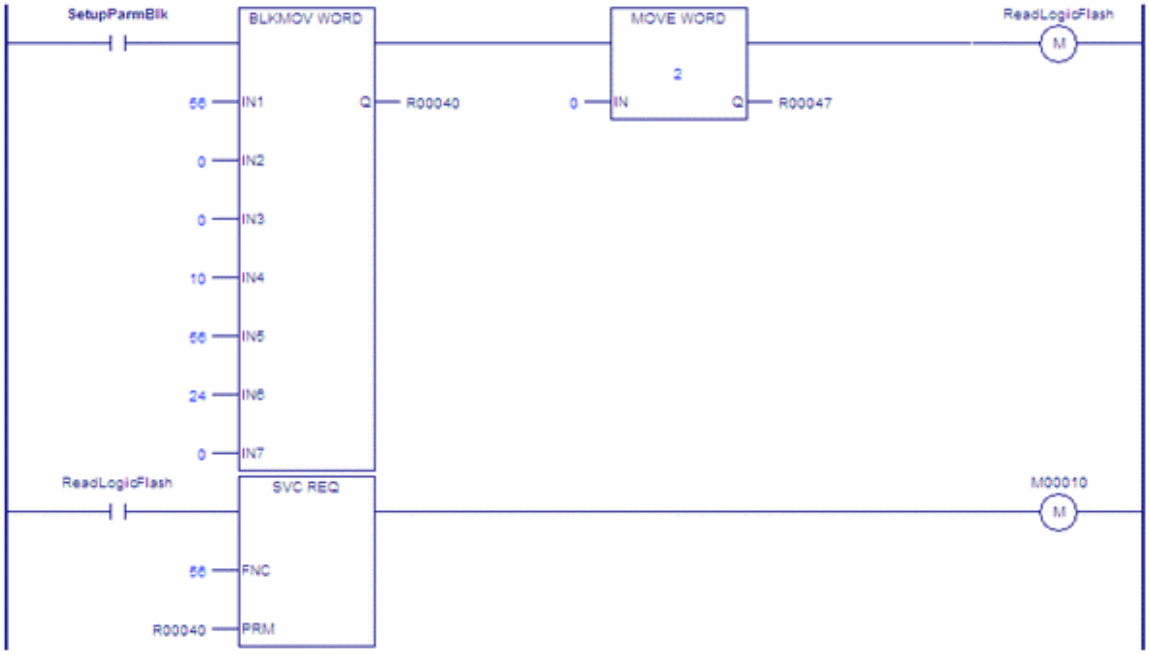
<table>
<tr><td></td><td>
<table>
<tr><td>%T (bit mode)</td><td>74</td></tr>
<tr><td>%M (bit mode)</td><td>76</td></tr>
<tr><td>%G (bit mode)</td><td>86</td></tr>
<tr><td>%W</td><td>196</td></tr>
</table>
</td></tr>
<tr><td>address + 5 and address + 6</td><td>The zero-based offset N in CPU memory to start writing the read data to. Address + 5, the least significant word, contains the complete offset for any memory area except %W, which also requires the use of address + 6 for offsets greater than 65,535.
For %I, %Q, %M, %T, and %G memory in byte mode, N = (Ra - 1) / 8, where Ra = one-based reference address. For example, to write to the one-based bit reference address %T33, enter the byte offset 4: (33 - 1) / 8 = 4.
For %W, %R, %AI, and %AQ memory, and for %I, %Q, %M, %T, and %G memory in bit mode, N = Ra - 1. For example, to write to the one-based reference address %R200, enter the zero-based reference offset 199; to write to %I73 in bit mode, enter offset 72.</td></tr>
<tr><td>address + 7</td><td>When bit 0 is set to 1, storage disabled conditions are ignored. A read is allowed even if the logic in RAM has changed since nonvolatile storage was read or written.
Bits 1 through 15 must be set to zero; otherwise, the read fails.</td></tr>
<tr><td>address + 8</td><td>Reserved. Must be set to zero; otherwise, the read fails.</td></tr>
<tr><td>address + 9</td><td>Response status. The status read from nonvolatile storage.
The low byte contains the major error code.
The high byte contains the minor error code.
<table>
<tr><th>Minor</th><th>Major</th><th>Description</th></tr>
<tr><td>00</td><td>01</td><td>Success. All values requested were found and copied.</td></tr>
<tr><td>01</td><td>01</td><td>Partial Read. All values found were copied, but some or all values were not in storage.</td></tr>
<tr><td>01</td><td>02</td><td>Insufficient Destination Memory. The Destination memory location is not large enough to store the requested values.</td></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td>02</td><td>02</td><td>Invalid Length. The length requested is larger than 64 bytes or less than 1 byte or the number of bits is not an exact multiple of 8.</td></tr>
</table>
</td></tr>
</table>

	01	**03**	Storage Busy. A SVC_REQ 57 or another SVC_REQ 56 instruction is active. For example, an interrupt block is attempting to execute SVC_REQ 56 when the block it interrupted was executing SVC_REQ 56.
	01	**04**	Storage Disabled. The logic in RAM differs from the logic in nonvolatile storage. See Storage disabled conditions.
	02	**04**	Storage Closed. Either the storage has not been created or a previous corruption error or unexpected read/write failure closed the storage.
	01	**05**	Unexpected Read Failure. A command to the storage hardware failed unexpectedly.
	02	**05**	Corrupted storage. A corrupted checksum or storage header caused a read to fail.
address + 10	Response Count. The number of words, bytes, or bits copied.		

Examples

Example 1

The following LD logic reads ten continuous bytes written to nonvolatile storage, ranging from %G1 through %G80. The read values are written to the range %G193 thrugh %G273. The value applied to IN1, 56, determines byte mode.
The parameter block starts at %R00040. The response words are returned to %R00049 and %R00050.



Parameter block for Example 1:

Address + offset	Address	Input Value	Definition
address + 0	%R00040	56	Data type = %G (byte mode)
address + 1	%R00041	0	Address written from, to be read, low word
address + 2	%R00042	0	Address written from, to be read, high word
address + 3	%R00043	10	Length = 10 bytes
address + 4	%R00044	56	Data type to write to = %G (byte mode)
address + 5	%R00045	24	Address to write to, low word
address + 6	%R00046	0	Address to write to, high word
address + 7	%R00047	0	Storage disabled conditions are enforced.
address + 8	%R00048	0	Reserved. Must be set to 0.
address + 9	%R00049	n/a	Response status
address + 10	%R00050	n/a	Response count

Example 2

Overwriting data in the same memory area. See SVC_REQ 57, Example 2.

SVC_REQ 57: Write to Nonvolatile Storage

CPU Support

SVC_REQ 57 is supported for PACSystems with firmware version 6.00 or later.

Operation

PACSystems Controllers support a 64KB nonvolatile flash memory area, which can be accessed by the logic-driven read/write service requests. Values are stored in the nonvolatile storage area by using SVC_REQ 57. These values are applied to the Controller user memory on power up.
SVC_REQ 57 specifies a range of reference addresses to read from a running PACSystems CPU and write to nonvolatile storage. This functionality is intended to retain a limited set of values, such as set points or tuning parameters, that need to change when the PACSystems is running. To write data that is static when the PACSystems is stopped, see write flash memory on a stopped Controller.

Note: Nonvolatile storage is intended for storing values that do not change frequently. Once the nonvolatile storage area fills up, a power cycle or stop mode store is required to store more values. The logic -driven write is not a replacement for battery backed RAM for values that change frequently or during every sweep.

SVC_REQ 57 scans the nonvolatile storage to find the most recent values stored for the specified range. If it finds no values for the range or the most recent stored values are different, then the new values are written to nonvolatile storage.

Data length

You can write up to 32 words (64 bytes) inclusively per invocation of SVC_REQ 57. Each invocation requires 4 words of command data (8 bytes). A 1-byte write requires 9 bytes whereas a 64-byte write requires 72 bytes. You can generally make the most efficient use of nonvolatile storage by transferring data in 64-byte increments. See, however, Fragmentation.

Erase Cycles

The flash component on the PACSystems CPU is rated for 100K erase cycles. Erase cycles occur under the following conditions:

- Write to flash is commanded from the programmer
- Clear flash operation
- Flash compaction after a power cycle when flash memory allotted for SVC_REQ 57 has become full

Discrete memory

Discrete memory can be written to as individual bits or as bytes. For more information, see address.
Forced and transition information is not written to nonvolatile storage.

Retentiveness

Writing values to nonvolatile storage for non-retentive memory such as %T does not make the memory retentive. For example, all values stored to %T memory are set to zero on power-up or a stop to run transition. You can, however, read such values from storage after power-up or stop to run transition by using SVC_REQ 56.

Maximum of one active instruction

When SVC_REQ 57 is active, it does not support an interrupt that attempts to activate SVC_REQ 56 or a second instance of SVC_REQ 57.

Storage disabled conditions

By default, the following write operations disable SVC_REQ 57 until logic is written to nonvolatile storage:

- Run Mode Store (RMS), even if a second RMS reverts everything to the original state
- Test-Edit session, even when you cancel your edits
- Word-for-word change
- Downloading to RAM only of a stopped PACSystems CPU, even if the downloaded contents are equal to the contents already on the nonvolatile storage

Setting bit 0 of input word 4 (address + 4) to a value of 1 enables SVC_REQ 57 despite the above conditions.

Error checking

When writing to nonvolatile storage, error checking is provided to ensure that logic and the Hardware Configuration (HWC) in nonvolatile memory match the logic and HWC in PACSystems RAM.

Fragmentation

Due to the nature of the media in PACSystems CPUs, writes may produce fragmentation, which causes the loss of more space on a write than is actually required for the write. For instance, if you write 64 bytes, but there are fewer than 64 bytes remaining in the current memory sector, the data is written into a new sector. This occurs because data records are not allowed to span sectors, which means that there may be unused bytes at the end of full sectors. The response data (address + 8 and address + 9) to the write request will show that the amount of available memory is reduced by the amount of data lost in the old sector plus the 64 bytes of data plus the 8 bytes of command data.

When nonvolatile storage is full

When logic driven user nonvolatile storage is full, a fault is logged. Before you can use SVC_REQ 57 to write again, use one of the following solutions:

To retain the most up-to-date data and continue writing with SVC_REQ 57 to nonvolatile storage:

1. Stop the PACSystems.
2. Power cycle the PACSystems.

A power cycle *when nonvolatile storage is full* triggers a compaction of existing data. During compaction, multiple writes of the same reference memory address are removed, which leaves only the most recent data, and contiguous reference memory addresses are combined into the fewest number of records necessary.
If compaction cannot take place, a second fault is logged and you need to use one of the following two solutions.

To retain specific data from nonvolatile storage, clear nonvolatile storage, and then return the data to nonvolatile storage:

1. While the PACSystems is still running, use SVC_REQ 56 to read the desired values into PACSystems memory.
2. Upload the current values from PACSystems memory as initial values to your project.
3. Stop the PACSystems.
4. Do one of the following:
   - Clear the flash memory
     - or -
   - Write to flash. The flash is erased prior to writing, which frees up some space.
5. Download the initial values to the PACSystems.
6. Start the PACSystems.
7. Use SVC_REQ 57 to write the desired values from PACSystems memory to nonvolatile storage.

To write to flash to erase everything:

1. Stop the PACSystems.
2. Write to flash.
   The flash is erased prior to writing, which frees up some space.

Equality

Because data in nonvolatile storage is not considered part of the project, writing to nonvolatile storage does not impact equality between the CPU and Logic Developer - PLC.

Redundancy

Redundancy systems can benefit from the use of logic driven user nonvolatile storage as long as all of the references saved to nonvolatile storage are included in the transfer lists. Each redundancy CPU maintains its own separate logic driven user nonvolatile storage by means of SVC_REQ 57 during its logic scan. If the values of reference addresses to be stored to user nonvolatile storage are synchronized, the logic driven user nonvolatile storage data in each CPU is identical. If the values to be stored are not synchronized, then each CPU's user nonvolatile storage may be different.

ENO and power flow to the right

If the status is Success or Partial Read (see address + 6), then on the SVC_REQ instruction, ENO is set to True in FBD and ST, and power flow passes to the right in LD.

Input and Output

The parameter block has a length of 12 words (24 bytes). The first six words contain the input you supply for the write operation. The last six words contain the status of the write operation and storage numbers.

<table>
<tr><td>address</td><td>The number representing the memory area that contains the data to write to nonvolatile storage.
<table>
<tr><th>For this memory type:</th><th>Enter this decimal value:</th></tr>
<tr><td>%R</td><td>8</td></tr>
<tr><td>%AI</td><td>10</td></tr>
<tr><td>%AQ</td><td>12</td></tr>
<tr><td>%I (byte mode)</td><td>16</td></tr>
<tr><td>%Q (byte mode)</td><td>18</td></tr>
<tr><td>%T (byte mode)</td><td>20</td></tr>
<tr><td>%M (byte mode)</td><td>22</td></tr>
<tr><td>%G (byte mode)</td><td>56</td></tr>
<tr><td>%I (bit mode)</td><td>70</td></tr>
<tr><td>%Q (bit mode)</td><td>72</td></tr>
<tr><td>%T (bit mode)</td><td>74</td></tr>
<tr><td>%M (bit mode)</td><td>76</td></tr>
<tr><td>%G (bit mode)</td><td>86</td></tr>
<tr><td>%W</td><td>196</td></tr>
</table></td></tr>
<tr><td>address + 1 and address + 2</td><td>The zero-based offset N in the CPU memory area that contains the data to write to nonvolatile storage memory. Address + 1, the least significant word, contains the complete offset for any memory area except %W, which also requires the use of address + 2 for offsets greater than 65,535.
For %I, %Q, %M, %T, and %G memory in byte mode, N = (Ra - 1) / 8, where Ra = one-based reference address. For example, to write the data starting at the one-based bit reference address %T33, enter the byte offset 4: (33 - 1) / 8 = 4.
For %W, %R, %AI, and %AQ memory, and for %I, %Q, %M, %T, and %G memory in bit mode, N = Ra - 1. For example, to write to nonvolatile storage the data starting at the one-based reference address %R200 in CPU memory, enter the zero-based reference offset 199; to write to nonvolatile storage the data starting at CPU memory address %I73 in bit mode, enter offset 72. For memory in bit mode, the offset must be set on a byte boundary, that is, a number exactly divisible by 8: 0, 8, 16, 24, and so on.</td></tr>
</table>

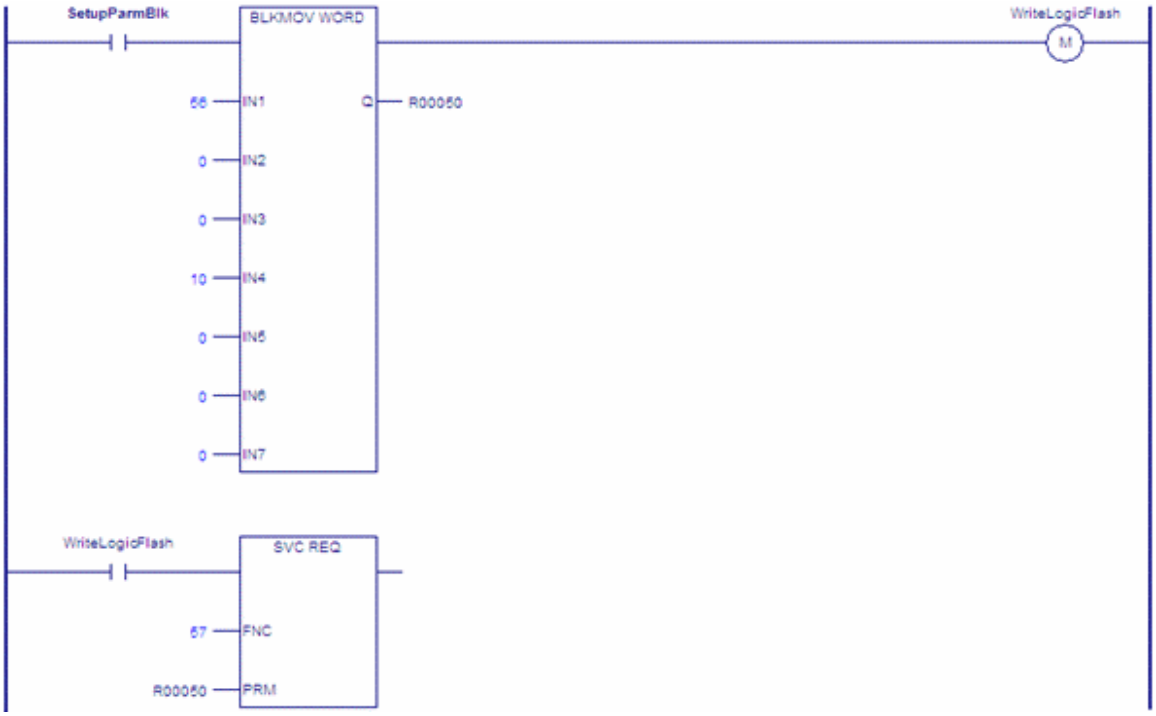
<table>
<tr><td>address + 3</td><td>Length. The number of items to write to nonvolatile storage beginning at the reference address calculated from the offset defined at [address + 1 and address + 2]. The length can be one of the following:
<table>
<tr><th>Description</th><th>Valid range</th></tr>
<tr><td>The number of words (16-bit registers) to write to %W, %R, %AI, or %AQ nonvolatile storage</td><td>1 through 32 words</td></tr>
<tr><td>The number of bytes to write to %I, %Q, %M, %T, or %G in byte mode nonvolatile storage</td><td>1 through 64 bytes</td></tr>
<tr><td>The number of bits to write to %I, %Q, %M, %T, or %G in bit mode nonvolatile storage</td><td>1 through 512 bits in increments of 8 bits</td></tr>
</table>
The value must reside in the low byte of address + 3. The high byte must be set to zero.</td></tr>
<tr><td>address + 4</td><td>▪ When bit 0 is set to 1, storage disabled conditions are ignored. A write is allowed even if the logic in RAM has changed since nonvolatile storage was read or written.
▪ Bits 1 through 15 must be set to zero; otherwise, the write fails.</td></tr>
<tr><td>address + 5</td><td>Reserved. Value must be set to zero.</td></tr>
<tr><td>address + 6</td><td>Response status.
▪ The low byte contains the major error code.
▪ The high byte contains the minor error code.
For example, 0001 means minor error code 00, major error code 01.
<table>
<tr><th>Minor</th><th>Major</th><th>Description</th></tr>
<tr><td>00</td><td>01</td><td>Success. All values requested were written.</td></tr>
<tr><td>01</td><td>01</td><td>Existing values found. All values requested are in storage, but one or more values were already stored.</td></tr>
<tr><td>01</td><td>02</td><td>Insufficient source memory. Counting from the offset, not enough reference addresses are left in the specified memory area.</td></tr>
<tr><td>02</td><td>02</td><td>Invalid length. The length requested was larger than 64 bytes or less than 1 byte or the number of bits is not divisible by 8.</td></tr>
<tr><td>03</td><td>02</td><td>Invalid source reference address. The memory area specified is not supported, the starting or ending offset is out of range, or the offset is not byte-aligned for discrete memory areas.</td></tr>
<tr><td>04</td><td>02</td><td>Invalid request. Spare bits or spare words in the parameter block are not set to zero.</td></tr>
</table></td></tr>
</table>

	01	**03**	Storage busy. A SVC_REQ 56 or another SVC_REQ 57 instruction is active. For example, an interrupt block is attempting to execute SVC_REQ 57 when the block it interrupted was executing SVC_REQ 57.
	01	**04**	Storage disabled. The logic in RAM differs from the logic stored in nonvolatile storage. See Storage disabled conditions.
	02	**04**	Storage closed. Either the storage has not been created or a previous corruption error or unexpected read/write failure closed the storage.
	01	**05**	Unexpected write failure. The command to the storage hardware failed unexpectedly.
	02	**05**	Corrupted storage. The write failed due to a bad checksum or corrupted storage header information.
	01	**06**	Write failed. Storage is full.
address + 7	Count of items written. Words, bytes, or bits.		
address + 8 and address + 9	The number of bytes available in nonvolatile storage.		
address + 10 and address + 11	Reserved.		

Examples

Example 1

The following LD logic writes ten continuous bytes to nonvolatile storage, ranging from %G1 through %G80. The value applied to IN1, 56, determines byte mode.



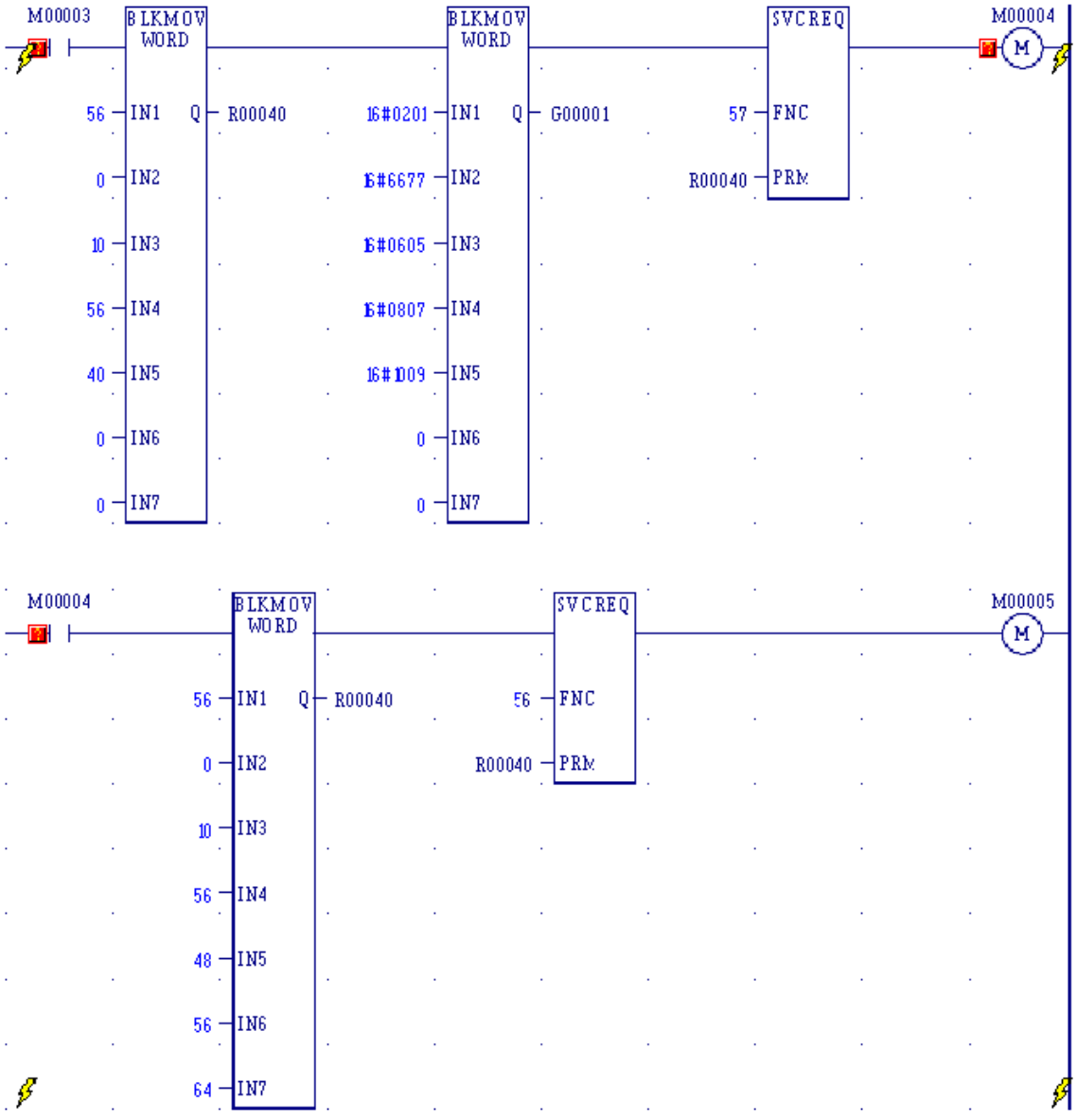
Parameter block for Example 1:

Address + offset	Address	Input Value	Definition
address + 0	%R00050	56	Data type = %G (byte mode)
address + 1	%R00051	0	Beginning address of data to write, low word
address + 2	%R00052	0	Beginning address of data to write, high word
address + 3	%R00053	10	Length = 10 bytes
address + 4	%R00054	0	Storage disabled conditions are enforced.
address + 5	%R00055	0	Reserved. Must be set to 0.
address + 6	%R00056	n/a	Response status. The low byte contains the major error code; the high byte contains the minor error code.
address + 7	%R00057	n/a	Count of bytes written.
address + 8	%R00058	n/a	The number of bytes available in nonvolatile storage.
address + 9	%R00059	n/a	
address + 10	%R00060	n/a	Reserved

address + 11	%R00061	n/a	Reserved

Example 2: Overwriting data in the same memory area

The following example illustrates the use of SVC_REQ 56 and SVC_REQ 57.
To overwrite some of the data in the same memory area, we can use the same LD logic with different values. Look at the status words returned for the below SVC_REQ execution starting from %G321.



Counters

(For PACSystems firmware version 3.50 and later.)

Data required for Counters

The data associated with FBD counter built-in function blocks is retentive through power cycles.

Instruction	*Mnemonic*	*Description*
Down Counter	DNCTR	Counts down from a preset value. The output is set to ON whenever the Current Value is ≤ 0.
Up Counter	UPCTR	Counts up to a designated value. The output is set to ON whenever the Current Value is ≥ the Preset Value.

Data Required for Counters

Each counter uses a WORD array[3] variable of %R, %AI, %AQ, %P, %L, %W, symbolic, or I/O variable memory area to store the following:

Current Value (CV) Word 1

Preset Value (PV) Word 2

Control word Word 3

When you enter a counter, you must enter a beginning address for the three-word array (block of registers), as the ???? operand.

Warning: Do not use two consecutive registers as the starting addresses of two counters. Logic Developer - PLC does not check or warn you if register blocks overlap. Counters do not work if you place the current value of a second counter on top of the preset value for the previous counter.

Word 1: Current value (CV)

Warning: Be very careful if you write to word 1 (CV), as there is a risk that the built-in function block may not work.

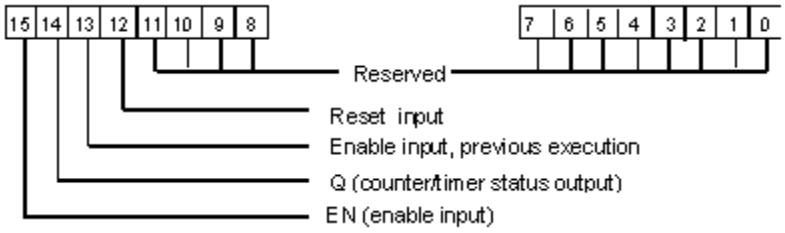
Word 2: Preset value (PV)

When the Preset Value (PV) operand is a variable, it is normally set to a different location than word 2 of the Counter built-in function block.

- If you use a different address and you change word 2 directly, your change has no effect, as PV overwrites word 2.
- If you use the same address for the PV operand and word 2, you can change the Preset Value in word 2 while the counter is running and the change is effective.

Word 3: Control word

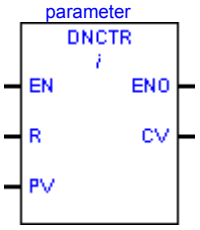
The control word stores the state of the boolean inputs and outputs of its associated counter, as shown below:



Note: Bits 0 through 11 are not used for counters.

Warning: The third word (Control) can be read but should not be written to; otherwise, the built-in function block does not work.

Down Counter



Operation

The Down Counter (DNCTR) built-in function block counts down from a preset value. The minimum Preset Value (PV) is zero; the maximum PV is +32,767 counts. When the built-in function block is executed, if R is ON, the Current Value (CV) is reset to PV. Otherwise, the Current Value (CV) is decremented by 1 to a minimum value of -32,768. The output (Q) is set to ON whenever CV ≤ 0.
The output state (Q) of DNCTR is retentive on power failure; no automatic initialization occurs at power-up.

Notes

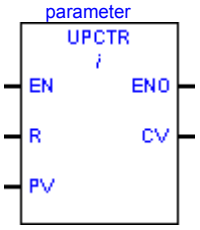
- You must perform an initial reset to enable the DNCTR built-in function block to function properly. If the DNCTR is not initially reset, CV decrements from zero, and the output of the DNCTR is immediately set to ON.
- The value of CV is both written to the CV operand and stored in Word 1 of the main DNCTR operand.

Operands

Operand	*Data Type*	*Memory Area*	*Description*
parameter	one-dimensional WORD array of 3 words	R, P, L, W, symbolic	The DNCTR control parameter. Word 1: Current Value (CV) Word 2: Preset Value (PV) Word 3: Control word **Cautions** ▪ Do not write to word 3 by any means. ▪ Overlapping reference addresses may cause erratic counter operation. **Note:** Undo is available when you edit the DNCTR control parameter.
i	The solve order for the built-in function block.		

EN	BOOL variable	data flow, I, Q, M, T, G, S, SA, SB, SC, discrete symbolic, I/O variable	If EN is ON when the built-in function block is executed, the counter built-in function block operates normally. If EN is OFF, execution continues with the next instruction and the counter is ignored.
	Bit reference in a non-BOOL variable	data flow, I, Q, M, T, G, S, SA, SB, SC, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	
R	BOOL variable	I, Q, M, T, G, S, SA, SB, SC, discrete symbolic, I/O variable	If R is ON (and EN is ON) when the built-in function block is executed, CV is reset to PV.
	Bit reference in a non-BOOL variable	I, Q, M, T, G, S, SA, SB, SC, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	
PV	INT variable or constant; BOOL array of length 16 or more (restrictions apply)	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	(Optional.) Preset Value to copy into word 2 of the counter's address when the counter built-in function block is enabled or reset. 0 ≤ PV ≤ 32,767. If PV is out of range, word 2 cannot be reset. **Note:** Instead of using the PV parameter, you can write directly to word 2 by using a MOVE instruction or an operator interface.
ENO	BOOL variable	data flow, I, Q, M, T, G, discrete symbolic, I/O variable	(Optional.) The state of the counter built-in function block. Set to ON when CV ≤ 0; Set to OFF otherwise.
	Bit reference in a non-BOOL variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	
CV	INT variable; BOOL array of length 16 or more (restrictions apply)	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	(Optional.) The current value of the counter built-in function block. This value is also stored in Word 1 in the DNCTR control operand.

Up Counter



Operation

The Up Counter (UPCTR) built-in function block counts up to the Preset Value (PV). The range is 0 to +32,767 counts. When the Current Value (CV) of the counter reaches 32,767, it remains there until reset. When the built-in function block counter is executed, if the reset parameter (R) is ON, CV resets to 0. Otherwise, CV increments by 1. CV can be incremented past the Preset Value (PV). The output (Q) is set to ON if the resulting CV ≥ PV.

Tip: The value of CV is both written to the CV operand and stored in Word 1 of the main UPCTR operand.

The state of the counter built-in function block is retentive on power failure; no automatic initialization occurs at powerup.

Operands

Operand	*Data Type*	*Memory Area*	*Description*
parameter	one-dimensional WORD array of 3 words	R, P, L, W, symbolic	The UPCTR control parameter is a three-element WORD array: Word 1: Current Value (CV) Word 2: Preset Value (PV) Word 3: Control word **Cautions** ▪ Do not write to word 3 by any means. ▪ Overlapping reference addresses may cause erratic counter operation. **Note:** Undo is available when you edit the UPCTR control parameter.
i	The solve order for the built-in function block.		

EN	BOOL variable	data flow, I, Q, M, T, G, S, SA, SB, SC, discrete symbolic, I/O variable	If EN is ON when the counter built-in function block is executed, the counter operates normally. If EN is OFF, execution continues with the next instruction and the counter is ignored.
	Bit reference in a non-BOOL variable	data flow, I, Q, M, T, G, S, SA, SB, SC, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	
R	BOOL variable	I, Q, M, T, G, S, SA, SB, SC, discrete symbolic, I/O variable	If R is ON (and EN is ON) when the built-in function block is executed, CV is reset to 0.
	Bit reference in a non-BOOL variable	I, Q, M, T, G, S, SA, SB, SC, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	
PV	INT variable or constant; BOOL array of length 16 or more (restrictions apply)	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	Preset Value to copy into word 2 of the counter's address when the counter is enabled or reset. 0 ≤ PV ≤ 32,767. If PV is out of range, it does not affect word 2. **Note:** If the PV parameter is a variable with the same address as Word 2 of the control parameter, you can write a new PV value directly to Word 2 by using a MOVE instruction or an operator interface.
ENO	BOOL variable	data flow, I, Q, M, T, G, discrete symbolic, I/O variable	(Optional.) The state of the counter. Set to ON when CV ≥ PV; Set to OFF otherwise.
	Bit reference in a non-BOOL variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	
CV	INT variable; BOOL array of length 16 or more (restrictions apply)	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The current value of the counter built-in function block. This value is also stored in Word 1 of the DNCTR control operand.

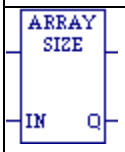
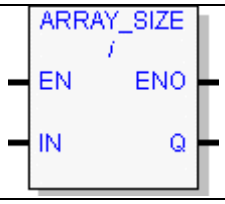
Data Move

(For PACSystems firmware version 3.50 and later.)
Data Move instructions provide basic move capabilities.

Instruction	*Mnemonics*	*Description*
Array Size	ARRAY_SIZE	Counts the number of elements in an array
Array Size Dimension 1	ARRAY_SIZE_DIM1	Returns the value of the Array Dimension 1 property of an array
Array Size Dimension 2	ARRAY_SIZE_DIM2	Returns the value of the Array Dimension 2 property of a two-dimensional array
BUS Read	BUS_RD	Reads data from the backplane of a bus.
BUS Read Modify Write	BUS_RMW_BYTE BUS_RMW_DWORD BUS_RMW_WORD	Updates a data element using the read/modify/write cycle on a bus.
BUS Test and Set	BUS_TS_BYTE BUS_TS_WORD	Handles semaphores on a bus.
BUS Write	BUS_WRT	Writes data to the backplane of a bus.
Communication Request	COMM_REQ	Enables the program to communicate with an intelligent module, such as a Genius Communications Module or a Programmable Coprocessor Module.
Fanout	FANOUT	Copies the input value to multiple outputs. You can configure the instruction to have two through eight outputs.
Move	MOV	Copies data as individual bits, so the new location does not have to be the same data type. Data can be moved into a different data type without prior conversion.
Move Data Ex	MOVE_DATA_EX	Copies data with the same data types without prior conversion and with optional data coherency
Move from Flat	MOVE_FROM_FLAT	This instruction is not currently available in FBD and ST. Use LD for the MOVE_FROM_FLAT operation.
Move to Flat	MOVE_TO_FLAT	Copies elements of symbolic User-defined Data Types (UDTs) to flat non-symbolic memory across mismatching data types with optional data coherency
Size Of	SIZE_OF	Counts the number of bits used by a variable of any data type except a BYTE array in non-discrete memory or a double-segment

		structure variable

Array Size

LD	FBD	ST
		Formal convention: Q := ARRAY_SIZE(In := [input]);

Operation

ARRAY_SIZE counts the number of elements in the array assigned to input IN.

Output

- In LD and FBD, ARRAY_SIZE writes the number to output Q.
- In ST, the value is assigned to the variable to the left of the assignment operator, represented by variable Q in the ST code above.
- If a non-array variable is assigned to input IN, the value of Q is 1.
- In an array of n structure variables, the value n is written to Q: the elements in each structure variable are not counted.

Tip: If the array assigned to input IN of ARRAY_SIZE is passed to a parameterized C block for processing, also pass the value of output Q to the block. In the C block logic, use the value of output Q to ensure you process all the array elements without exceeding the end of the array. For a two-dimensional array, this method works only if all elements are treated identically, for example, all are initialized to the same value.

Operands

Input Operands

Operand	*Data Type*	*Memory Area*	*Description*
i	(FBD only.) The solve order for the instruction.		
power flow (LD only)			When set to On, ARRAY_SIZE executes. When set to Off, ARRAY_SIZE does not execute.
EN (FBD only)	BOOL variable	data flow, I, Q, M, T, S, SA, SB, SC, G, discrete symbolic, I/O variable	Enable input. When set to On, ARRAY_SIZE solves. When set to Off, ARRAY_SIZE does not solve.
	Bit reference in non-BOOL variable	R, P, L, AI, AQ, W, I, Q, M, T, G, non-discrete symbolic, I/O variable	
IN	Array of any data type	data flow, I, Q, M, T, S, SA, SB, SC, G, R, P, L, AI, AQ, W, symbolic,	Array whose elements are counted

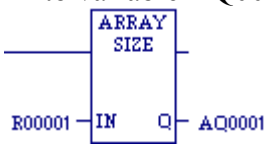
		I/O variable	

Output Operands

Operand	*Data Type*	*Memory Area*	*Description*
power flow (LD only; optional)			Set to On when ARRAY_SIZE has executed successfully.
ENO (FBD only; optional)	BOOL variable	data flow, I, Q, M, T, S, SA, SB, SC, G, discrete symbolic, I/O variable	ENO is set to On if EN is set to On.
	Bit reference in non-BOOL variable	R, P, L, AI, AQ, W, I, Q, M, T, G, non-discrete symbolic, I/O variable	
Q	DINT or DWORD variable. ST also supports INT and WORD variables.	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	Number of elements in the array assigned to input IN. In ST, the value of Q is assigned to the variable on the left side of the assignment operator.

Example

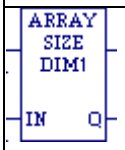
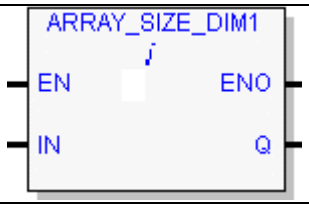
The two-dimensional array R00001 has its Array Dimension 1 property set to 4 and its Array Dimension 2 property set to 3. ARRAY_SIZE calculates 4 * 3 and writes the value 12 to variable AQ0001.



CPU Support

ARRAY_SIZE is supported for all PACSystems CPUs.

Array Size Dimension 1

LD	FBD	ST
		Formal convention: Q := ARRAY_SIZE_DIM1(In := [input]);

Operation

ARRAY_SIZE_DIM1 returns the value of the Array Dimension 1 property of an array.
Output

- In LD and FBD, ARRAY_SIZE_DIM1 writes the value to output Q.
- In ST, the value is assigned to the variable to the left of the assignment operator, represented by variable Q in the ST code above.
- If a non-array variable is assigned to input IN, the value of Q is 0.

You can use the value to ensure that a loop using a variable index to access array elements does not exceed the array's first dimension.

Operands

Input Operands

Operand	*Data Type*	*Memory Area*	*Description*
i	(FBD only.) The solve order for the instruction.		
power flow (LD only)			When set to On, ARRAY_SIZE_DIM1 executes. When set to Off, ARRAY_SIZE_DIM1 does not execute.
EN (FBD only)	BOOL variable	data flow, I, Q, M, T, S, SA, SB, SC, G, discrete symbolic, I/O variable	Enable input. When set to On, ARRAY_SIZE_DIM1 solves. When set to Off, ARRAY_SIZE_DIM1 does not solve.
	Bit reference in non-BOOL variable	R, P, L, AI, AQ, W, I, Q, M, T, G, non-discrete symbolic, I/O variable	
IN	Array of any data type	data flow, I, Q, M, T, S, SA, SB, SC, G, R, P, L, AI, AQ, W, symbolic, I/O	The array whose Array Dimension 1 property value is returned in the Q output (LD, FBD) or assigned to the variable on the left side of the assignment statement

		variable	(ST)

Output Operands

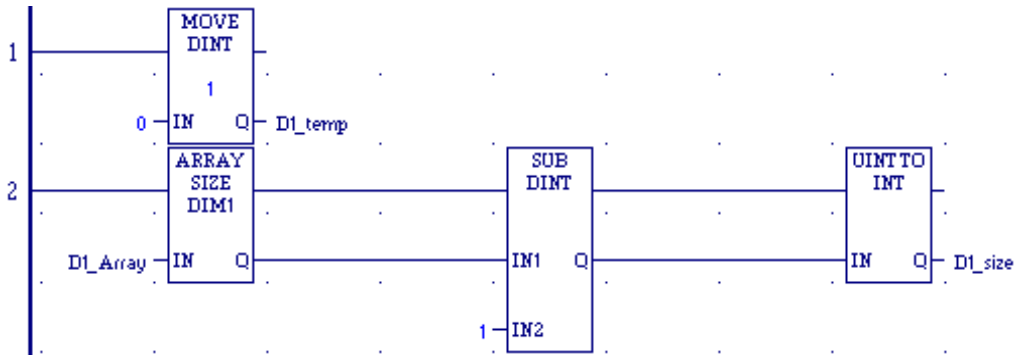
Operand	*Data Type*	*Memory Area*	*Description*
power flow (LD only; optional)			Set to On when ARRAY_SIZE_DIM1 has executed successfully.
ENO (FBD only; optional)	BOOL variable	data flow, I, Q, M, T, S, SA, SB, SC, G, discrete symbolic, I/O variable	ENO is set to On if EN is set to On.
	Bit reference in non-BOOL variable	R, P, L, AI, AQ, W, I, Q, M, T, G, non-discrete symbolic, I/O variable	
Q	DINT or DWORD variable. ST also supports INT and WORD variables.	data flow, I, Q, M, T, S, SA, SB, SC, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The value of the Array Dimension 1 property of the array assigned to input IN. The value is set to 0 if a non-array is assigned to IN. In ST, the value is assigned to the variable on the left side of the assignment operator. **Note:** Because the index of the first element of an array is zero, the index of the last element is one less than the value assigned to Q.

Examples

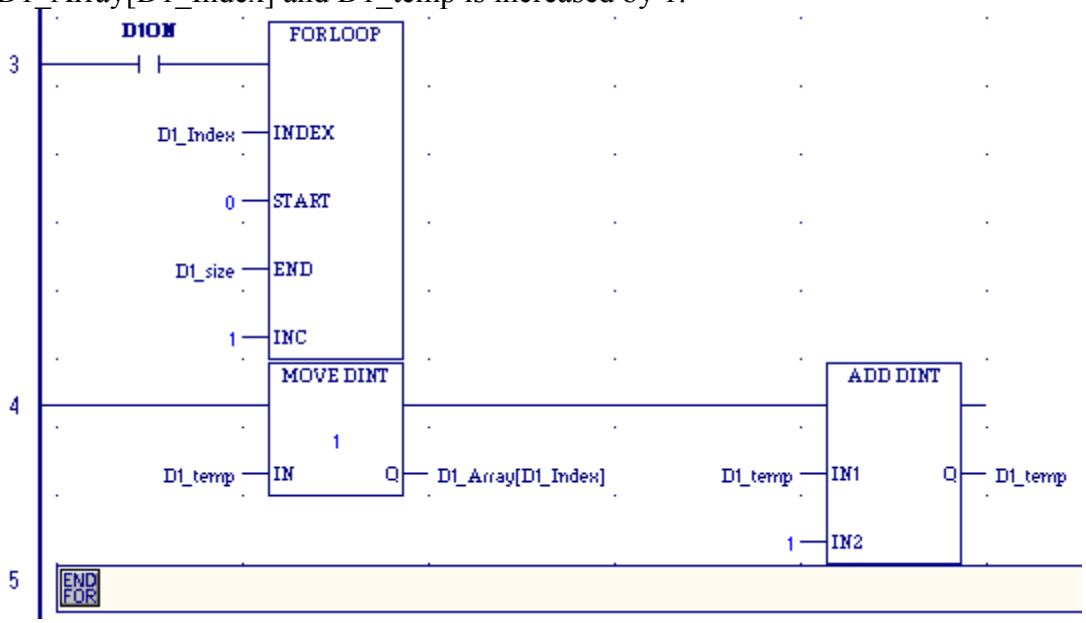
FOR_LOOP in LD logic

If you set up a FOR_LOOP that accesses array elements by means of a variable index, you must ensure that the FOR_LOOP does not iterate beyond the last element of the array.

In the following logic, MOVE_DINT initializes the variable D1_temp to 0. ARRAY_SIZE_DIM1 counts the number of elements of a one-dimensional array named D1_Array and outputs the result to output Q. Because the index of the first element of an array is zero, the loop must iterate (Q - 1) times. SUB_DINT performs the subtraction and the result is converted to an INT value and assigned to variable D1_size.



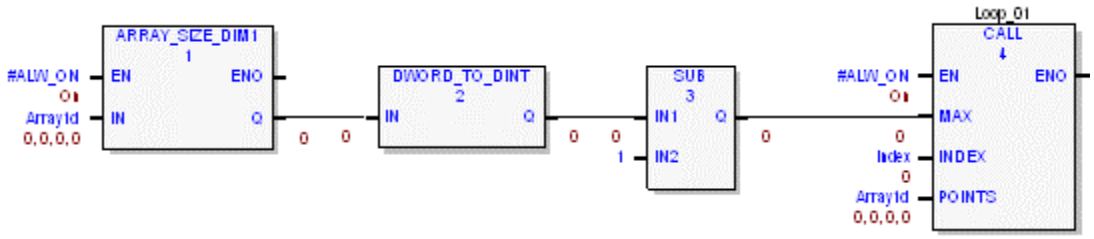
In the diagram below, the FOR_LOOP executes when D1ON is set to On. The variable index (D1_Index, see INDEX input) increments by 1 (see INC input) from 0 (see START input) through D1_size, the value calculated by ARRAY_SIZE_DIM1 and SUB_DINT (see END input). In each loop, the value of D1_temp is assigned to the element D1_Array[D1_Index] and D1_temp is increased by 1.



Loop initialization in FBD logic

FBD is not designed for loops that are completed in one scan. You can, however, use FBD to set up the parameters required for a loop and pass them to an LD or ST block that performs the loop.

In the following example, ARRAY_SIZE_DIM1 counts the number of elements of a one-dimensional array named Array1d and the value is converted from DWORD to DINT. Because an array element index is zero-based, the loop must iterate 1 less time than the number of elements. SUB performs the subtraction and passes the result to the MAX input of an LD or ST parameterized block named Loop_01. A value of 0 (zero) is assigned to the INDEX input and the array named Array1d is passed to the POINTS parameter.



In the non-displayed logic of Loop_01, a loop iterates by means of a variable index through all the elements of the POINTS formal parameter. The latter is set up as an 8-DWORD array, whereas the Array1d array passed to POINTS is a 4-DWORD array. Because the value assigned to MAX is 3, the loop iterates only four times, from 0 through 3, thus not exceeding the number of elements of the Array1d array used to populate the POINTS parameter. The Loop_01 block can thus be used to process the points of a 4-point or 8-point analog module.

Loop in ST logic

The following ST logic uses a For loop to initialize all elements of a one-dimensional REAL array, Array1d, to 0.0.
Dim1Size and i are INT variables. Array_Size_Dim1 is used to count the number of elements of Array1d. Because the lowest possible index of an array element is zero, the for loop iterates from 0 through a value 1 less than the number of elements.

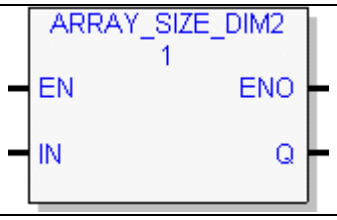
```
Dim1Size := Array_Size_Dim1(Array1d);
for i := 0 to (Dim1Size - 1) do
     Array1d[i] := 0.0
end_for;
```

If the above logic is used in an ST parameterized block or user-defined function block, any one-dimensional REAL array can be passed to it and have its elements initialized to 0.0.

CPU Support

ARRAY_SIZE_DIM1 is supported for all PACSystems CPUs.

Array Size Dimension 2

LD	FBD	ST
		Formal convention: Q := ARRAY_SIZE_DIM2(In := [input]);

Operation

ARRAY_SIZE_DIM2 returns the value of the Array Dimension 2 property of a two-dimensional array.

Output

- In LD and FBD, ARRAY_SIZE_DIM2 writes the value to output Q.
- In ST, the value is assigned to the variable to the left of the assignment operator, represented by variable Q in the ST code above.
- If a non-array variable is assigned to input IN, the value of Q is 0.

Using output Q as the upper limit of variable index loops in LD and ST

In an LD or ST block that is not a parameterized block or a User Defined Function Block (UDFB), you can use the output Q value to ensure that a loop using a variable index to access array elements does not exceed the array's second dimension.
Examples:

- LD logic: FOR_LOOP that iterates through an array's second dimension
- LD logic: FOR_LOOP that iterates through both dimensions of an array
- ST logic: FOR_LOOP that iterates through both dimensions of an array

Workaround to process two-dimensional arrays in parameterized blocks and UDFBs

Parameterized blocks and UDFBs do not support two-dimensional array parameters. All two-dimensional arrays passed into a parameterized block or UDFB are converted to flat, one-dimensional arrays. The total number of elements in the two-dimensional array input should equal the number of elements in the one-dimensional array parameter. If a two-dimensional array is passed into a parameterized block or UDFB, the ARRAY_SIZE_DIM1 and ARRAY_SIZE_DIM2 values should also be passed to determine which array element is being modified. The following example shows how a two-dimensional array is represented in one-dimensional format:

Two-dimensional array [3,2]	One-dimensional array [6]
Array[0,0]	Array[0]
Array[0,1]	Array[1]
Array[1,0]	Array[2]

Array[1,1]	Array[3]
Array[2,0]	Array[4]
Array[2,1]	Array[5]

The code to determine the one-dimensional parameter array index from the array indexes and array dimension 2 value of the two-dimensional array is

```
Flat_Index = (Dim1 * Array_Dim2) + Dim2
```

where

- Flat_Index is the index of the one-dimensional array element
- Dim1 is the one-dimensional index of the two-dimensional array element
- Array_Dim2 is the value of the Array Dimension 2 property of the two-dimensional array
- Dim2 is the two-dimensional index of the two-dimensional array element

Example: Passing a two-dimensional array from LD logic to an ST UDFB

No one-scan loop support in FBD

FBD is not designed for loops that are completed in one scan. You can use FBD to set up the loop parameters and pass them to an LD or ST block that performs the loop.

First dimension

To ensure that a variable index does not exceed the first dimension, ARRAY_SIZE_DIM1 is required.

Operands

Input Operands

Operand	*Data Type*	*Memory Area*	*Description*
i	(FBD only.) The solve order for the instruction.		
power flow (LD only)			When set to On, ARRAY_SIZE_DIM2 executes. When set to Off, ARRAY_SIZE_DIM2 does not execute.
EN (FBD only)	BOOL variable	data flow, I, Q, M, T, S, SA, SB, SC, G, discrete symbolic, I/O variable	Enable input. When set to On, ARRAY_SIZE_DIM2 solves. When set to Off, ARRAY_SIZE_DIM2 does not solve.
	Bit reference in non-BOOL variable	R, P, L, AI, AQ, W, I, Q, M, T, G, non-discrete symbolic, I/O variable	
IN	Array of any data type	data flow, I, Q, M, T, S, SA, SB, SC, G, R, P, L, AI, AQ,	The array whose Array Dimension 2 property value is returned in the Q output (LD, FBD) or assigned to the variable on

		W, symbolic, I/O variable	the left side of the assignment statement (ST)

Output Operands

Operand	*Data Type*	*Memory Area*	*Description*
power flow (LD only; optional)			Set to On when ARRAY_SIZE_DIM2 has executed successfully.
ENO (FBD only; optional)	BOOL variable	data flow, I, Q, M, T, S, SA, SB, SC, G, discrete symbolic, I/O variable	ENO is set to On if EN is set to On.
	Bit reference in non-BOOL variable	R, P, L, AI, AQ, W, I, Q, M, T, G, non-discrete symbolic, I/O variable	
Q	DINT or DWORD variable. ST also supports INT and WORD variables.	data flow, I, Q, M, T, S, SA, SB, SC, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The value of the Array Dimension 2 property of the array assigned to the IN input. The value is set to 0 if a non-array is assigned to IN. In ST, the value is assigned to the variable on the left side of the assignment operator. **Note:** Because the index of the first element of an array is zero, the index of the last element is one less than the value assigned to Q.

Examples

- LD logic: FOR_LOOP that iterates through an array's second dimension
- LD logic: FOR_LOOP that iterates through both dimensions of an array
- ST logic: FOR_LOOP that iterates through both dimensions of an array
- Passing a two-dimensional array from LD logic to an ST UDFB

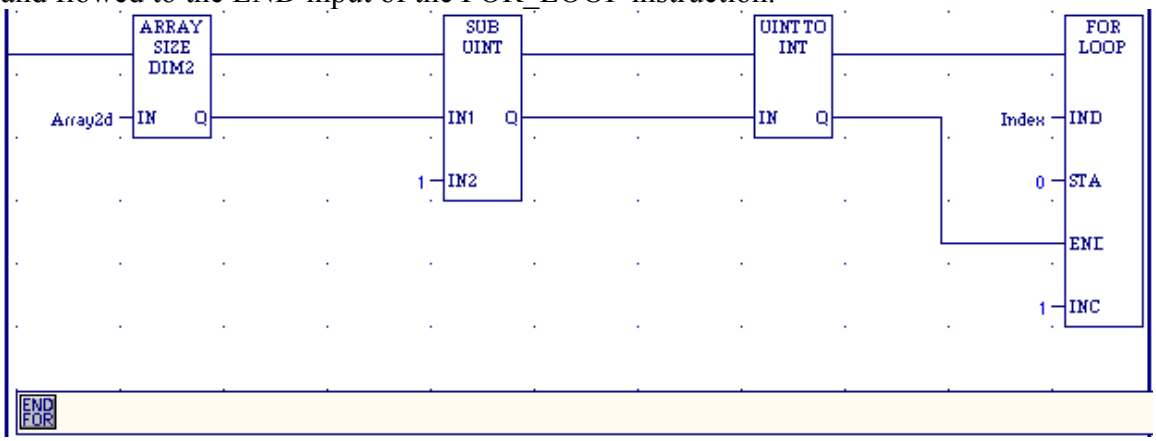
LD logic: FOR_LOOP that iterates through an array's second dimension

In an LD block that is not a parameterized block or a User Defined Function Block (UDFB), if you set up a FOR_LOOP that accesses elements in the second dimension of an array by means of a variable index, you must ensure that the FOR_LOOP does not exceed the array's second dimension.

Note: Parameterized blocks and User Defined Function Blocks (UDFBs) do not support two-dimensional array parameters. See Workaround to process two-dimensional arrays in parameterized blocks and UDFBs.

In the following logic, ARRAY_SIZE_DIM2 counts the number of elements of a two-dimensional array named Array2d and outputs the result to output Q. Because the index

of the first element of any dimension of an array is zero, the loop must iterate (Q - 1) times. SUB_UINT performs the subtraction and the result is converted to an INT value and flowed to the END input of the FOR_LOOP instruction.



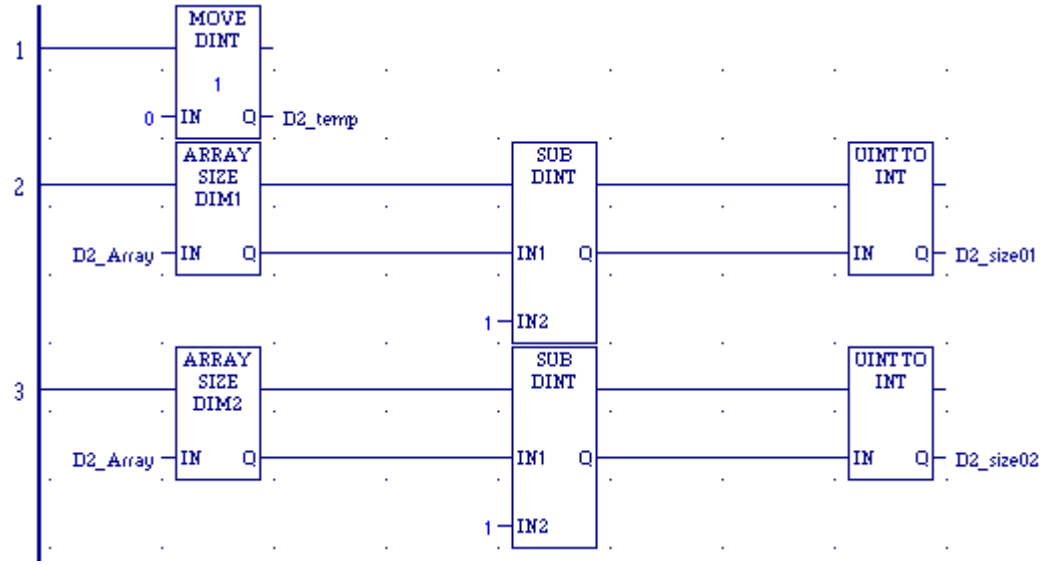
In the logic appearing between the FOR_LOOP and END_FOR instructions, various operations can take place with, for example, Array2d(1,Index), where Index increments by 1 (see INC input) from 0 (see START input) through the value calculated by ARRAY_SIZE_DIM2 and SUB_DINT (see END input).

LD logic: FOR_LOOP that iterates through both dimensions of an array

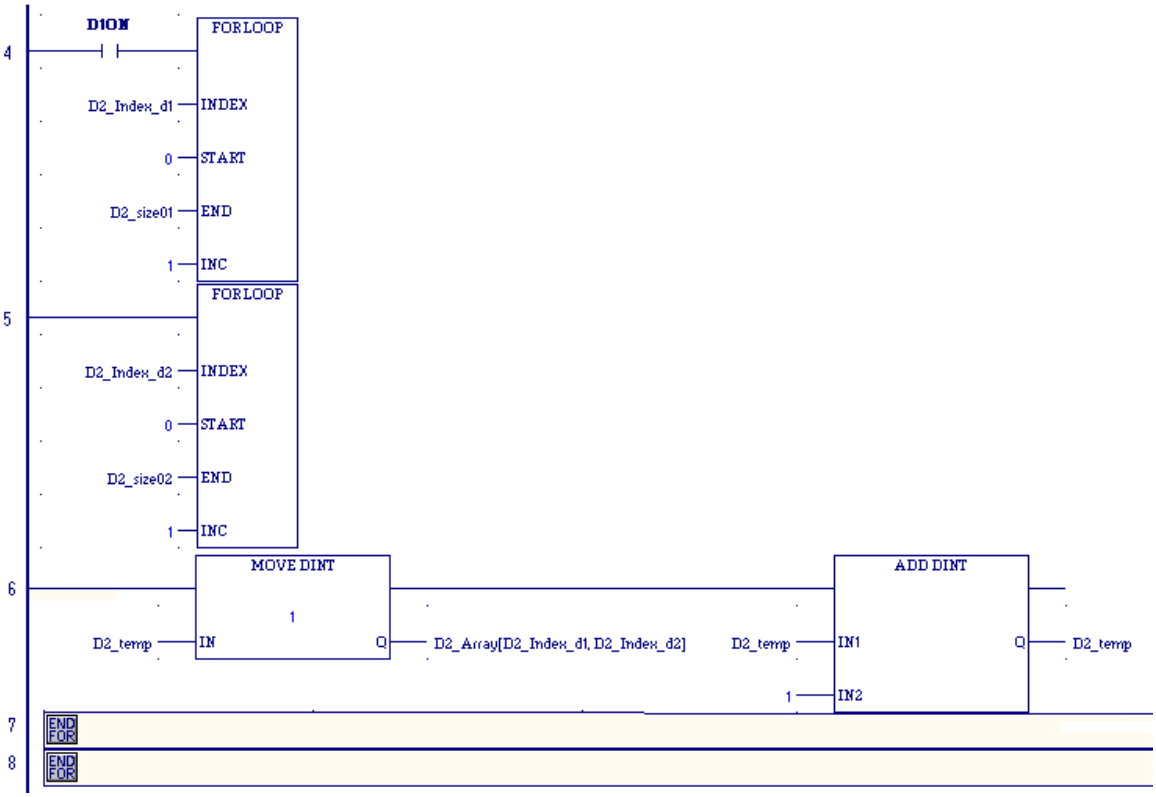
In an LD or ST block that is not a parameterized block or a User Defined Function Block (UDFB), you can use ARRAY_SIZE_DIM1, ARRAY_SIZE_DIM2, and nested FOR_LOOPs to ensure that operations on elements using two variable indexes each do not exceed either array dimension.

Note: Parameterized blocks and User Defined Function Blocks (UDFBs) do not support two-dimensional array parameters. See Workaround to process two-dimensional arrays in parameterized blocks and UDFBs.

In the following logic, MOVE_DINT initializes the variable D2_temp to 0.
ARRAY_SIZE_DIM1 counts the number of elements in the first dimension of a two-dimensional array named D2_Array. The value is placed in output Q. Because the index of the first element of an array is zero, the loop must iterate (Q - 1) times. SUB_DINT performs the subtraction and the result is converted to an INT value and assigned to variable D2_size01.
Likewise, ARRAY_SIZE_DIM2 counts the number of elements in the second dimension. After the subtraction and conversion, the value is assigned to variable D2_size02.



In the diagram below, the first FOR_LOOP executes when D1ON is set to ON. The variable index (D2_Index_d1, see INDEX input) increments by 1 (see INC input) from 0 (see START input) through D2_size01, the value calculated by ARRAY_SIZE_DIM1 and SUB_DINT (see END input). Within the first loop, the second FOR_LOOP executes. The variable index is D2_Index_d2, which increments by 1 from 0 through D2_size02, the value calculated by ARRAY_SIZE_DIM2 and SUB_DINT. Each time the second FOR_LOOP executes, the value of D2_temp is assigned to the element D2_Array[D2_Index_d1,D2_Index_d2] and D2_temp is incremented by 1.



ST logic: FOR_LOOP that iterates through both dimensions of an array

In an ST block that is not a parameterized block or a User Defined Function Block (UDFB), the following ST logic uses nested for loops to initialize all the elements of a two-dimensional REAL array, Array2d, to 0.0.

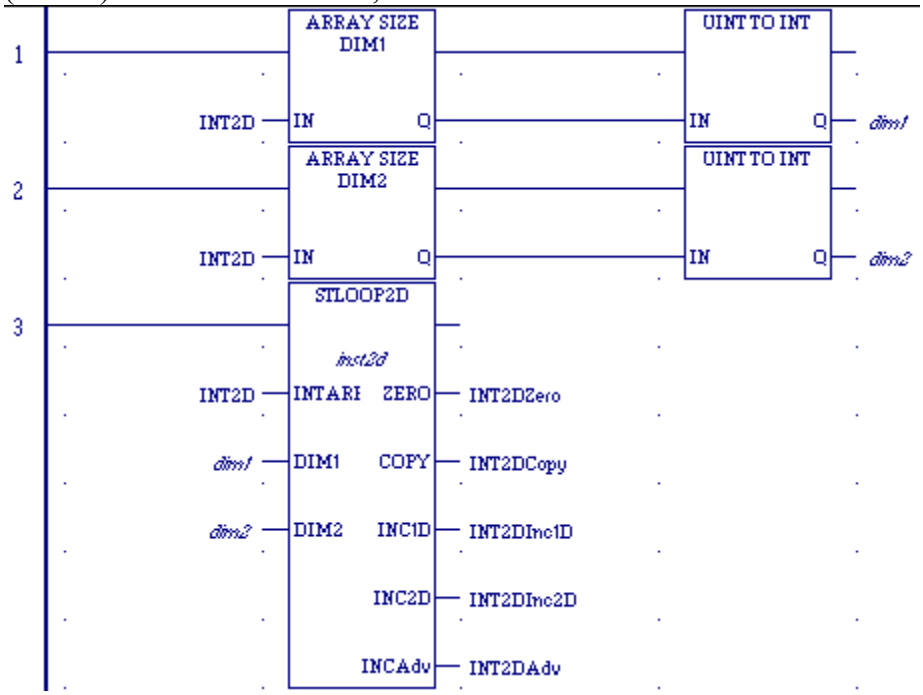
Note: Parameterized blocks and User Defined Function Blocks (UDFBs) do not support two-dimensional array parameters. See Workaround to process two-dimensional arrays in parameterized blocks and UDFBs.

Dim1Size, Dim2Size, i, and j are INT variables. Array_Size_Dim1 and Array_Size_Dim2 count the number of elements respectively in the first and second dimensions of Array2d. Because the lowest possible index of an array element is zero, the for loops iterate from 0 through a value 1 less than the number of elements in the respective dimension.

```
Dim1Size := Array_Size_Dim1(Array2d);
Dim2Size := Array_Size_Dim2(Array2d);
for i := 0 to (Dim1Size - 1) do
     for j := 0 to (Dim2Size - 1) do
          Array2d[i,j] := 0.0;
     end_for;
end_for;
```

Passing a two-dimensional array from LD logic to an ST UDFB

In the following LD logic, the two-dimensional array INT2D and its Array Dimension 1 and Array Dimension 2 values are passed to the ST User Defined Function Block (UDFB) named STLOOP2D, whose instance variable is inst2d.



INT2D is passed to the parameter INTARR, which is a one-dimensional array. One of the output parameters of STLOOP2D, INC1D, is a one-dimensional array; the other four are two-dimensional arrays. All five outputs are assigned to two-dimensional arrays for the calling LD logic to use.

In the following ST logic of UDFB STLOOP2D, five different operations on one-dimensional arrays are performed in such a way that the one-dimensional elements are

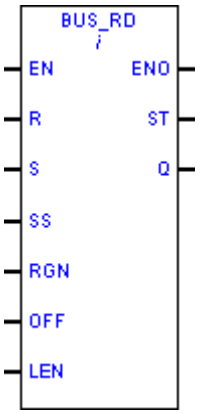
correctly mapped to two-dimensional elements of the two-dimensional arrays assigned to the LD Call to STLOOP2D.

```
'Zero a 2D array
for i := 0 to (DIM1 - 1) do
    for j := 0 to (DIM2 - 1) do
        loopcount := (i*(DIM2) + j);
        ZERO[loopcount] := 0;
    end_for;
end_for;

'Copy the 2D array
for i := 0 to (DIM1 - 1) do
    for j := 0 to (DIM2 - 1) do
        loopcount := (i*(DIM2) + j);
        COPY[loopcount] := INTARR[loopcount];
    end_for;
end_for;

'Incrementing a 1D array
for i := 0 to (DIM1 - 1) do
    for j := 0 to (DIM2 - 1) do
        loopcount := (i*(DIM2) + j);
        INC1D[loopcount] := loopcount;
    end_for;
end_for;

'Incrementing a 2D array
for i := 0 to (DIM1 - 1) do
    for j := 0 to (DIM2 - 1) do
        loopcount := (i*(DIM2) + j);
        INC2D[loopcount] := j;
    end_for;
end_for;

'2D array value = DIM1 * 1000 + DIM2
'For example, 0, 1, 2, ... j, 1000, 1001, 1002, ...
for i := 0 to (DIM1 - 1) do
    for j := 0 to (DIM2 - 1) do
        loopcount := (i*(DIM2) + j);
        INCAdv[loopcount] := i*1000 + j;
    end_for;
end_for;
```

CPU Support

ARRAY_SIZE_DIM2 is supported for all PACSystems CPUs.

Bus Read



Operation

The Bus Read (BUS_RD) function reads data from the bus.

Note: Bus functions (BUS_RD, BUS_WRT, BUS_RMW, or BUS_TS) require additional board-specific information to properly address the board. This information may be obtained from one of two sources:

- The board vendor may issue application notes on the correct use of the board.
- For VME boards, you can refer to the *Guidelines for the Selection of Third-Party VME Modules*, GFK-0448.

When BUS_RD is executed, it accesses memory from the module on the bus. It uses the R, S, SS, RGN, and OFF operands to determine the address of memory from which to read. BUS_RD then copies data with the length LEN to Controller locations beginning at output Q. Execution passes to the next instruction when the operation completes or when the operation is determined to be unsuccessful.

Note: When using preemptive block scheduling, a BUS_RD instruction inside an interrupt block that is being executed may cause the block to be preempted if a new, incoming interrupt has the same priority.

If EN is ON, the BUS_RD instruction stores a status value in the ST output, indicating whether or not the operation was successful and, if not, why it was unsuccessful.

ST Value	*Description*
0	Operation successful.
1	Bus error.
2	Module does not exist at rack/slot location.
3	Module at rack/slot location is an invalid type.
4	Start address outside the configured range.
5	End address outside the configured address range.
6	Absolute address is even but the interface is configured as odd byte only.
7	*Not used.*

8	Region not enabled.
9	*Not used.*
10	Function parameter invalid.

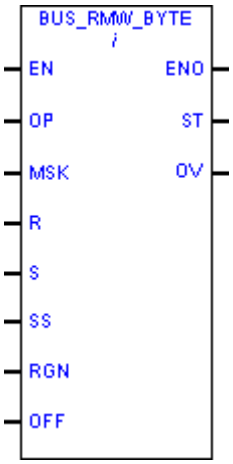
Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
EN	BOOL variable	data flow, I, Q, M, T, G, S, SA, SB, SC, discrete symbolic, I/O variable	If EN is ON when the instruction is executed, the instruction operates normally. If EN is OFF, execution continues with the next instruction and the instruction is ignored.
	Bit reference in a non-BOOL variable	data flow, I, Q, M, T, G, S, SA, SB, SC, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	
R	UINT variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The number of the rack that contains the module.
S	UINT variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The number of the slot that contains the module.
SS	UINT variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	(Optional.) The number of the subslot that contains the module. Default: 0.
RGN	WORD variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	(Optional.) The memory region number, as defined on the module's Memory tab. Default: 1.
OFF	DWORD variable or constant	data flow, I, Q, M, T, G, SA, SB, SC, R, P, L, AI, AQ, W, symbolic, I/O variable	The offset in the memory region. The exact memory location is calculated by adding the offset to the memory region's base address, which is defined on the module's Memory tab.
LEN	constant		Length. The number of BYTE, DWORD, or WORD values to read from the bus module. 1 ≤ ?? ≤ 32,767.
ST	WORD	data flow, I, Q, M,	The status of the operation. For details,

	variable	T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	see the ST table in Operation above.
ENO	BOOL variable	data flow, I, Q, M, T, G, discrete symbolic, I/O variable	(Optional.) Whether or not the operation was successfully initiated. If ENO is ON, the operation was successfully initiated. The results of the operation will be indicated in the ST output. If ENO is OFF, the operation was not performed. If EN was on, the ST output will indicate an error condition. If EN was OFF, the ST output is left unchanged.
	Bit reference in a non-BOOL variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	
Q	BYTE, DWORD, or WORD variable	data flow, I, Q, M, T, G, SA, SB, SC, R, P, L, AI, AQ, W, symbolic, I/O variable	The data read from the module. The data type of the variable determines the data units that are read. For example, if Q is a WORD variable named MyWORD, the instruction will move (LEN) WORD variables, or LENx2 bytes of data.

Bus Read Modify Write



Mnemonics:
BUS_RMW_BYTE
BUS_RMW_WORD
BUS_RMW_DWORD

Operation

The Bus Read/Modify/Write (BUS_RMW) function updates a data element on the bus.

Notes

- When using BUS_RMW_WORD, the absolute address must be a multiple of 2. The absolute address is equal to the base address + offset.
- When using BUS_RMW_DWORD, the absolute address must be a multiple of 4.
- Bus functions (BUS_RD, BUS_WRT, BUS_RMW, or BUS_TS) require additional information to properly address a board. This information may be obtained from one of two sources:
  - The board vendor may issue application notes on the correct use of the board.
  - For VME boards, you can also refer to the *Guidelines for the Selection of Third-Party VME Modules*, GFK-0448.

The BUS_RMW instruction uses the R, S, SS, RGN, and OFF operands to determine the address of the module memory that is to be accessed. Execution of the BUS_RMW instruction takes place in three phases:

- **Read phase:** BUS_RMW reads the byte, word, or dword of data from the bus at the address.
- **Modify phase:** The target byte, word, or dword of data is combined (AND/OR) with the data mask MSK. Selection of the AND or OR operation is made with the OP input. MSK is a word value. If byte data is operated on, only the lower eight bits of MSK are used. If word data is operated on, only the lower 16 bits of MSK are used.
- **Write phase:** The result is written back to the same bus address from which it was read.

The BUS_RMW instruction stores a status value in the ST output, indicating the success or failure of the operation. The possible values for ST are as follows:

ST Value	*Description*
0	Operation successful.
1	Bus error.
2	Module does not exist at rack/slot location.
3	Module at rack/slot location is an invalid type.
4	Start address outside the configured range.
5	End address outside the configured address range.
6	Absolute address is even but the interface is configured as odd byte only.
7	For BUS_RMW_WORD, the absolute address is not a multiple of 2. For BUS_RMW_DWORD, the absolute address is not a multiple of 4.
8	Region not enabled.
9	Function type too large for configured access type.
10	Function parameter invalid.

Operands

Notes

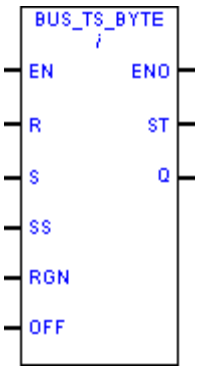
- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
EN	BOOL variable	data flow, I, Q, M, T, G, S, SA, SB, SC, discrete symbolic, I/O variable	If EN is ON when the instruction is executed, the instruction operates normally. If EN is OFF, execution continues with the next instruction and the instruction is ignored.
	Bit reference in a non-BOOL variable	data flow, I, Q, M, T, G, S, SA, SB, SC, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	
OP	Constant		Specifies how to use the MSK data to modify the target data: ▪ A value of 0 performs a bitwise AND operation on the data and the MSK data. ▪ A value of 1 ORs the data with the MSK data.
MSK	DWORD variable or constant	data flow, I, Q, M, T, G, S, R, P, L, AI, AQ, W,	The data mask. **Note:** For BUS_RMW_BYTE, only the lower 8 bits are used; for

		symbolic, I/O variable	BUS_RMW_WORD, only the lower 16 bits are used.
R	UINT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The number of the rack that contains the module
S	UINT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The number of the slot that contains the module
SS	UINT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	(Optional.) The number of the subslot that contains the module. Default: 0.
RGN	WORD variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	(Optional.) The memory region number, as defined on the module's Memory tab. Default: 1.
OFF	DWORD variable or constant	data flow, I, Q, M, T, G, SA, SB, SC, R, P, L, AI, AQ, W, symbolic, I/O variable	The offset in the memory region. The exact memory location is found by adding the offset to the memory region's base address, which is defined on the module's Memory tab.
ENO	BOOL variable	data flow, I, Q, M, T, G, discrete symbolic, I/O variable	(Optional.) Whether or not the operation was successfully initiated. If ENO is ON, the operation was successfully initiated. The results of the operation will be indicated in the ST output. If ENO is OFF, the operation was not performed. If EN was on, the ST output will indicate an error condition. If EN was OFF, the ST output is left unchanged.
	Bit reference in a non-BOOL variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	
ST	WORD variable	data flow, R, P, L, AI, AQ, W, symbolic, I/O variable	(Optional.) The status of the operation. For details, see the ST table in Operation above. **Note:** If BUS_RMW_WORD does not contain an even absolute address, or, if BUS_RMW_DWORD does not contain an absolute address that is a multiple of 4, the function block will return a status code of 7 and set ENO to OFF.
OV	BYTE, DWORD, or WORD	data flow, I, Q, M, T, G, SA, SB, SC,	(Optional.) The original value that was read from the module.

variable, depending on the mnemonic you are using	R, P, L, AI, AQ, W, symbolic, I/O variable

Bus Test and Set



Mnemonics:
BUS_TS_BYTE
BUS_TS_WORD

Operation

The Bus Test and Set (BUS_TS) function handles semaphores on the bus. BUS_TS exchanges a Boolean ON (1) for the value currently at the semaphore location. If that value was already ON, then BUS_TS does not obtain the semaphore. If the existing value was OFF, then the semaphore is reset and BUS_TS function has the semaphore and the use of the memory area it controls. The semaphore is cleared using the BUS_WRT function to write a 0 to the semaphore location.

Notes

- BUS_TS_WORD works only for even absolute addresses (absolute address = base address + offset).
- Using a bus function (BUS_RD, BUS_WRT, BUS_RMW, or BUS_TS) requires additional information on the correct way to address the board. This information may be obtained from one of two sources:
  - The board vendor may issue application notes on the correct use of the board.
  - For VME boards, you can also refer to the *Guidelines for the Selection of Third-Party VME Modules*, GFK-0448.

When BUS_TS executes, it accesses the module on the bus. It uses the R, S, SS, RGN, and OFF operands to determine the address of the memory to read from. BUS_TS then exchanges a Boolean ON with the data at the address. BUS_TS sets the Q output to ON if the semaphore was available (OFF) and was acquired. Execution passes to the next instruction when the operation completes or when the operation is known to be unsuccessful.
The BUS_TS instruction stores a status value in the ST output, indicating the success or failure of the operation. The possible values for ST are as follows:

ST Value	*Description*
0	Operation successful.
1	Bus error.

2	Module does not exist at rack/slot location.
3	Module at rack/slot location is an invalid type.
4	Start address outside the configured range.
5	End address outside the configured address range.
6	Absolute address is even but the interface is configured as odd byte only.
7	(BUS_TS_WORD only.) The absolute address is not a multiple of 2.
8	Region not enabled.
9	*Not used.*
10	Function parameter invalid.

Operands

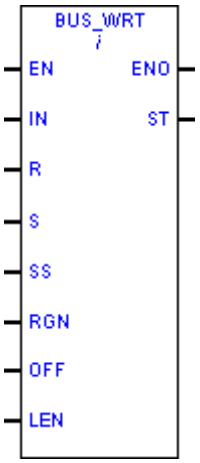
Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	***Data Type***	***Memory Area***	***Description***
EN	BOOL variable	data flow, I, Q, M, T, G, S, SA, SB, SC, discrete symbolic, I/O variable	If EN is ON when the instruction is executed, the instruction operates normally. If EN is OFF, execution continues with the next instruction and the instruction is ignored.
	Bit reference in a non-BOOL variable	data flow, I, Q, M, T, G, S, SA, SB, SC, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	
R	UINT variable or constant	data flow, R, P, L, AI, AQ, W, symbolic, I/O variable	The number of the rack that contains the module
S	UINT variable or constant	data flow, R, P, L, AI, AQ, W, symbolic, I/O variable	The number of the slot that contains the module
SS	UINT variable or constant	data flow, R, P, L, AI, AQ, W, symbolic, I/O variable	(Optional.) The number of the subslot that contains the module. Default: 0.
RGN	WORD variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	(Optional.) The memory region number, as defined on the module's Memory tab. Default: 1.
OFF	DWORD variable or constant	data flow, I, Q, M, T, G, SA, SB, SC, R, P, L, AI, AQ, W,	The offset within the memory region. The exact memory location is calculated by adding the offset to the

		symbolic, I/O variable	memory region's base address, which is defined on the module's Memory tab.
ENO	BOOL variable	data flow, I, Q, M, T, G, discrete symbolic, I/O variable	(Optional.) Whether or not the operation was successfully initiated. If ENO is ON, the operation was successfully initiated. The results of the operation will be indicated in the ST output. If ENO is OFF, the operation was not performed. If EN was on, the ST output will indicate an error condition. If EN was OFF, the ST output is left unchanged.
	Bit reference in a non-BOOL variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	
ST	WORD variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The status of the operation. **Note:** If BUS_TS_WORD does not contain an even absolute address, the function block will not attempt to acquire the semaphore, returning a status code of 7. The absolute address is the base address plus the offset (OFF).
Q	BOOL variable	data flow, I, Q, M, T, G, symbolic, I/O variable	Set to ON if the OFF (offset) semaphore was available. Otherwise, Q is set to OFF.
	Bit reference in a non-BOOL variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	

Bus Write



Operation

The Bus Write (BUS_WRT) function writes data to the bus.

Note: Using a Bus function (BUS_RD, BUS_WRT, BUS_RMW, or BUS_TS) requires additional information on the correct way to address the bus board. This information may be obtained from one of two sources:

- The board vendor may issue application notes on the correct use of the board.
- For VME boards, refer to the *Guidelines for the Selection of Third-Party VME Modules*, GFK-0448.

When BUS_WRT is set to ON, it accesses the bus module. It uses the R, S, SS, RGN, and OFF operands to determine the address of the memory to read from. BUS_WRT then copies the data from the input parameter IN to the bus module at the address. Execution passes to the next instruction when the operation completes or when the operation is known to be unsuccessful.

Note: (PACSystems firmware version 2.00 and later, preemptive block scheduling.) A BUS_WRT instruction inside an interrupt block being executed may cause the block to be preempted when a new, incoming interrupt has the same priority.

If EN is ON, the BUS_WRT instruction stores a status value in the ST output, indicating whether or not the operation was successful and, if not, why it was unsuccessful.

ST Value	*Description*
0	Operation successful.
1	Bus error.
2	Module does not exist at rack/slot location.
3	Module at rack/slot location is an invalid type.
4	Start address outside the configured range.
5	End address outside the configured address range.
6	Absolute address is even but the interface is configured as odd byte only.

7	*Not used.*
8	Region not enabled.
9	*Not used.*
10	Function parameter invalid.

Operands

Notes

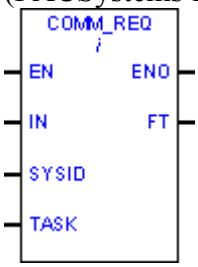
- For each mnemonic, use the corresponding data type for the IN operand. For example, BUS_WRT_BYTE requires IN to be a BYTE variable.
- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
EN	BOOL variable	data flow, I, Q, M, T, G, S, SA, SB, SC, discrete symbolic, I/O variable	If EN is ON when the instruction is executed, the instruction operates normally. If EN is OFF, execution continues with the next instruction and the instruction is ignored.
	Bit reference in a non-BOOL variable	data flow, I, Q, M, T, G, S, SA, SB, SC, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	
IN	BYTE or WORD variable	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The data to write to the module. If a variable is used, the data type of the variable determines the data units that are written. For example, if IN is a WORD variable named MyWORD, the instruction will move (LEN) WORD variables, or LENx2 bytes of data. If a constant is used, the data type for the move operation is assumed to be DWORD.
	DWORD variable	data flow, I, Q, M, T, G, S, SA, SB, SC, R, P, L, AI, AQ, W, symbolic, I/O variable	
	constant		
R	UINT variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The number of the rack that contains the module
S	UINT variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O	The number of the slot that contains the module

		variable	
SS	UINT variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	(Optional.) The number of the subslot that contains the module. Default: 0.
RGN	WORD variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	(Optional.) The memory region number, as defined on the module's Memory tab. Default: 1.
OFF	DWORD variable or constant	data flow, I, Q, M, T, G, SA, SB, SC, R, P, L, AI, AQ, W, symbolic, I/O variable	The offset in the memory region. The exact memory location is found by adding the offset to the memory region's base address, which is defined on the module's Memory tab.
LEN	constant		Length. The number of BYTEs, DWORDs, or WORDs to write to the bus module. 1 ≤ ?? ≤ 32,767.
ENO	BOOL variable	data flow, I, Q, M, T, G, discrete symbolic, I/O variable	(Optional.) Whether or not the operation was successfully initiated. If ENO is ON, the operation was successfully initiated. The results of the operation will be indicated in the ST output. If ENO is OFF, the operation was not performed. If EN was on, the ST output will indicate an error condition. If EN was OFF, the ST output is left unchanged.
	Bit reference in a non-BOOL variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	
ST	WORD variable	data flow, I, Q, M, T, G, SA, SB, SC, R, P, L, AI, AQ, W, symbolic, I/O variable	The status of the operation. For details, see Operation above.

Communication Request

(PACSystems firmware version 3.50 and later.)



Operation

The Communication Request (COMM_REQ) function communicates with a GE intelligent module, such as a Genius Communications Module, a Programmable Coprocessor Module, or a LAN Interface Module.

Notes

- The information presented here shows only the basic format of the COMM_REQ function. Many types of COMM_REQ functions have been defined. You will need additional information to program the COMM_REQ for each type of device. Programming requirements for each module that uses the COMM_REQ function are described in the specialty module's user documentation.
- If you are using Serial Communications, refer to the *Series 90 PLC Serial Communications User's Manual* (GFK-0582). If you are using MMS-Ethernet Communications, refer to the *MMS-Ethernet Communications for the Series 90-70 PLC User's Manual* (GFK-0686).

When COMM_REQ is executed, it sends the command block of data specified by the IN operand to the communications TASK in the intelligent or specialty module, at the rack/slot location specified by the SYSID operand. After sending the command block, COMM_REQ can either suspend execution and wait for a reply (for a maximum waiting period specified in the command), or immediately resume execution of the program.

- If the command block specifies that the program will not wait for a reply, the command block contents are sent to the receiving device and program execution immediately resumes. (The timeout value is ignored.) This is referred to as NOWAIT mode.
- If the command block specifies that the program will wait for a reply, the command block contents are sent to the receiving device and the CPU waits for a reply. The maximum length of time the Controller will wait for the device to respond is specified in the command block. If the device does not respond within that time, program execution resumes. This is referred to as WAIT mode.

COMM_REQ passes data, unless the timeout period is exceeded, or if a 0 timeout period has been specified. The Function Faulted (FT) output may be set ON if:

- The specified target module is not present or is faulted.
- The specified task is not valid for the device.
- The data length is 0.

The Function Faulted output may have these states:

Enable	*Error?*	*Function Faulted Output*
active	no	OFF
active	yes	ON
not active	no execution	OFF

Note: When using preemptive block scheduling, a COMM_REQ instruction inside an interrupt block being executed may cause the block to be preempted when a new, incoming interrupt has the same priority.

Command Block

The command block provides information to the intelligent module on the command to be performed. The command block starts at the reference specified by the operand IN. This address may be in any word-oriented area of memory (%R, %P, %L, %W, %AI, or %AQ). The length of the command block depends on the amount of data sent to the device.

The Command Block contains the data to be communicated to the other device, plus information related to the execution of the COMM_REQ. The command block has the following structure:

Address	*Length (in words)*
Address + 1	Wait/No Wait Flag
Address + 2	Status Pointer Memory
Address + 3	Status Pointer Offset
Address + 4	Idle Timeout Value
Address + 5	Maximum Communication Time
Address + 6 to Address + 133	Data Block

Information required for the command block can be placed in the designated memory area using an appropriate programming function, such as MOV.

When entering information for the command block, refer to these definitions:

Data Block Length

The number of data words starting with the data at address+6 to the end of the command block, inclusive. The data block length ranges from 1 to 128 words. Each COMM_REQ command has its own data block length. When entering the data block length, you must ensure that the command block fits within the register limits

Wait/No Wait Flag

This selects whether or not the program should wait for CCM communications to be completed.

For	*Enter*
No wait	0
Wait for reply	1

The flag bit is stored in the least significant bit (LSB) at address + 1. The rest of the word should be filled with zeros.

Status Pointer Memory Type

The two status pointer words specify a Controller memory location where the status word returned by the device will be written when the COMM_REQ completes.

Status Pointer Memory Type	address + 2
Status Pointer Offset	address + 3

Status pointer memory type contains a numeric code that specifies the user reference memory type for the status word. The table below shows the code for each reference type:

For this memory type		*Enter this decimal value*
%I	Discrete input table (BIT mode)	70
%Q	Discrete output table (BIT mode)	72
%I	Discrete input table (BYTE mode)	16
%Q	Discrete output table (BYTE mode)	18
%R	Register memory	8
%AI	Analog input table	10
%AQ	Analog output table	12

Notes

- The value entered determines the mode. For example, if you enter the %I bit mode is 70, then the offset will be viewed as that bit. On the other hand, if the %I value is 16, then the offset will be viewed as that byte.
- The high byte at address + 2 should contain zero.

Status Pointer Offset

The word at address + 3 contains the offset for the status word within the selected memory type.

Note: The status pointer offset is a zero-based value. For example, %R00001is at offset zero in the register table.

Idle Timeout Value

The idle timeout value is the maximum time the Controller CPU waits for the device to acknowledge receipt of the COMM_REQ. This value is ignored in NOWAIT mode. If

WAIT mode is selected, address + 4 specifies the idle timeout period in 100-microsecond increments.

Maximum Communication Time

The value at address +5 specifies the maximum time the Controller CPU waits for the device to complete the COMM_REQ. This time is also specified in 100-microsecond increments and is ignored in NOWAIT mode.

Data Block

The data block contains the command's parameters. The data block begins with a command number in address + 6, which identifies the type of communications function to be performed. Refer to the specific device manual (that is, PCM, GBC, Communications) for specific COMM_REQ command formats.

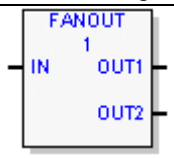
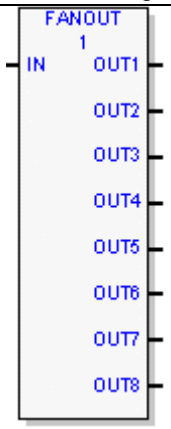
Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
EN	BOOL variable	I, Q, M, T, G, S, SA, SB, SC, discrete symbolic, I/O variable	Indicates whether the COMM_REQ function is enabled: ▪ If ON, communication is attempted with the module specified in the other parameters. The results are indicated in the FT output. ▪ If OFF, communication is not attempted with the module. The ENO output is automatically set to OFF and execution continues with the next instruction immediately. The FT output is left unchanged.
	Bit reference in a non-BOOL variable	I, Q, M, T, G, S, SA, SB, SC, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	
IN	WORD variable	R, P, L, AI, AQ, W, symbolic	The reference of the first WORD of the command block.
SYSID	WORD variable or constant; BOOL array of length 16 or more.	I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The rack number (most significant byte) and slot number (least significant byte) of the target device (intelligent module). **Note:** For systems that do not have expansion racks, SYSID must be zero for the main rack.
TASK	DWORD variable or constant; BOOL array of length 32 or more.	I, Q, M, T, G, SA, SB, SC, R, P, L, AI, AQ, W, symbolic, I/O variable	The task ID of the process on the target device

ENO	BOOL variable	data flow, I, Q, M, T, G, discrete symbolic, I/O variable	(Optional.) Whether or not the operation completed successfully. If ENO is ON, the operation was successfully initiated. The results of the operation will be indicated in the FT output. If ENO is OFF, the operation was not performed. If EN was on, the FT output will be set to OFF. If EN was OFF, the FT output is left unchanged.
	Bit reference in a non-BOOL variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	
FT	BOOL variable	I, Q, M, T, G, discrete symbolic, I/O variable	Function Faulted output. FT is set to ON if an error is detected while processing the instruction. Typically, this is due to one of the following: ▪ The COMM_REQ instruction is configured to work in WAIT mode, but the CPU does not support it. ▪ The specified target address (SYSID operand) is not present. ▪ The specified task (TASK operand) is not valid for the device. ▪ The data length is 0. ▪ The device's status pointer address (part of the command block) does not exist. This may be due to an incorrect memory type selection, or an address within that memory type that is out of range.
	Bit reference in a non-BOOL variable	I, Q, M, T, G, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	

Fanout

Minimum number of outputs: 2	Maximum number of outputs: 8
	

Operation

When it receives data, the FANOUT instruction copies the input value to multiple outputs of the same data type as the input. You can configure the instruction to have from two through eight outputs.

To configure the number of outputs:

1. In the FBD editor, right-click the FANOUT instruction and choose **Properties**. The Inspector displays the FANOUT instruction properties.
2. Set the value of the Outputs property.

Possible use for the FANOUT instruction

If you want to use the output of an instruction as the input for multiple instructions, wire the output of that instruction to the input of a FANOUT instruction and wire the outputs of FANOUT to the inputs of the multiple instructions.

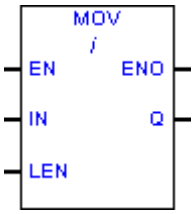
Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	BOOL, DINT, DWORD, INT, LREAL, REAL, UINT, or WORD variable or constant **Note:** Arrays are not supported.	I, Q, M, T, G, R, P, L, W, AI, AQ, symbolic, I/O variable	The input to copy to the outputs.

OUT1, OUT2, OUT3, ... , OUT8	Variables of the same data type as the IN operand	Minimum: 2 outputs. Maximum: 8 outputs.

Move



Operation

When the input operand, EN, is set to ON, the MOV instruction copies data as bits from one location in Controller memory to another. Because the data is copied as bits, the new location does not need to use the same type of memory area as the source. For example, you can copy data from an analog memory area into a discrete memory area, or vice versa.

MOV sets its output, ENO, whenever it receives data unless one of the following occurs:

- When the input, EN, is set to OFF, then the output, ENO, is set to OFF.
- When the input, EN is set to ON, and the input, IN, contains an indirect reference, and the memory of IN is out of range, then the output, ENO, is set to OFF.

The value to store at the destination Q is acquired from the IN parameter. If IN is a variable, the value to store in Q is the value stored at the IN address. If IN is a constant, the value to store in Q is that constant.

The result of the MOVE depends on whether the data type for the Q operand is a bit reference or a non-bit reference:

- If Q is a non-bit reference, LEN (the length) indicates the number of memory locations in which the IN value should be repeated, starting at the location specified by Q.
- If Q is a bit reference, IN is treated as an array of bits. LEN therefore indicates the number of bits to acquire from the IN parameter to make up the stored value. If IN is a constant, bits are counted from the least significant bit. If IN is a variable, LEN indicates the number of bits to acquire starting at the IN location. Regardless, only LEN bits are stored starting at address Q.

For example, if IN was the constant value 29 and LEN is 4, then the results of a MOV operation are as follows:

- ***Q is a WORD reference:*** The value 29 is repeatedly stored in locations Q, Q+1, Q+2, and Q+3.
- ***Q is a BOOL reference:*** The binary representation of 29 is 11101. Since LEN is 4, only the four least significant bits are used (1101). This value is stored at location Q in the same order, so 1 is stored in Q, 1 is stored in Q+1, 0 is stored in Q+2, and 1 is stored in Q+3.

If data is moved from one location in discrete memory to another, such as from %I memory to %T memory, the transition information associated with the discrete memory elements is updated to indicate whether or not the MOVE operation caused any discrete memory elements to change state.

Note: If an array of BOOL-type data specified in the Q operand does not include all the bits in a byte, the transition bits associated with that byte (which are not in the array) are cleared when the Move instruction receives data.

Data at the IN operand does not change unless there is an overlap in the source and destination—a situation that is to be avoided.

Operands

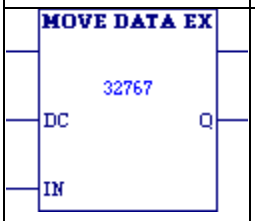
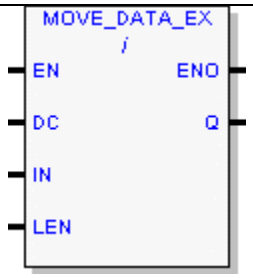
Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
EN	BOOL variable	data flow, I, Q, M, T, G, S, SA, SB, SC, discrete symbolic, I/O variable	Enable input. See Operation.
	Bit reference in non-BOOL variable	R, P, L, AI, AQ, W, I, Q, M, T, G, non-discrete symbolic, I/O variable	
IN	DINT, DWORD, INT, LREAL, REAL, UINT, WORD, STRUCTURE, enumerated variable, or a BOOL variable (or bit reference in non-BOOL variable), or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable BOOL, WORD, and DWORD variables can also use S, SA, SB, SC	The source of the data to copy into the output Q. This can be either a constant or a variable whose reference address is the location of the first source data item to move. If IN is not a constant, it must have the same data type as the variable in the Q parameter. The length of the data unit depends on the data type of the IN or Q variable. If IN is a constant, it must have a decimal fraction if Q is of the REAL or LREAL data type, for example 7.1 or 7.0. It must have no decimal fraction if Q is of any other data type, for example 7, not 7.0. If IN is a BOOL variable or a bit reference, an %I, %Q, %M, or %T reference address need not be byte-aligned, but 16 bits beginning with the reference address specified are displayed online.
LEN	constant		The number of data units to copy.

			The data unit depends on the data type of the IN and/or Q parameters. The valid range for LEN is as follows: ▪ If IN is a constant and Q is a BOOL variable, 1 ≤ LEN ≤ 16. ▪ If IN is a constant and Q is not a BOOL variable, 1 ≤ LEN ≤ 256. ▪ For all other cases, 1 ≤ LEN ≤ 32,767. LEN is also interpreted differently depending on the data type of the Q location. For more details, see Operation above.
ENO	BOOL variable	data flow, I, Q, M, T, G, discrete symbolic, I/O variable	(Optional.) Whether or not the move operation completed successfully. Output. See Operation.
	Bit reference in non-BOOL variable	R, P, L, AI, AQ, W, I, Q, M, T, G, non-discrete symbolic, I/O variable	
Q	DINT, DWORD, INT, LREAL, REAL, UINT, WORD, STRUCTURE, enumerated variable, or a BOOL variable (or bit reference in non-BOOL variable)	data flow, I, Q, M, T, SA, SB, SC, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The variable whose reference address is the location of the first destination data item. If IN is not a constant, Q must have the same data type as the variable in the IN parameter. If IN is a constant with a decimal fraction (like 7.1 or 7.0), Q must be of the REAL or LREAL data type. If IN has no decimal fraction (like 7), Q must not be of the REAL or LREAL data type. If Q is a BOOL variable or a bit reference, an %I, %Q, %M, or %T reference address does not need to be byte-aligned, but a full 16 bits beginning with the specified reference address are displayed online.

Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- For STRUCTURES, all elements must be in either discrete or analog memory; a mix is not allowed. TON, TOF, TP, and custom structures are not supported by MOV.

Move Data Explicit

LD	FBD	ST
		Formal convention: MOVE_DATA_EX(DC := , IN := , Length := , Q =>); - or - MOVE_DATA_EX(IN := , Length := , Q =>); **Tip:** Drag instruction from the Toolchest's LD Instructions or FBD Instructions drawer to your ST logic. The instruction with its input and output operand names are inserted in ST logic. Now assign a variable to the inputs and outputs, and a variable to the left of the assignment operator.

Operation

MOVE_DATA_EX provides optional data coherency by locking symbolic memory being written to during the copy operation. This enables a large number of bytes to be copied at one time.

Notes

- The input DC should be used only when using interrupt blocks and is required only when the same memory is used in more than one interrupt block, or in the main program and an interrupt block.
- If DC is True, an interrupt block cannot preempt the copy operation.
- If DC is False or not present, then interrupts can preempt the copy.
- Using DC can impact interrupt latency if the amount of data copied is large.

Operands

Notes

- All operands are required except for input DC (Data Coherency) and output Q.
- The constant value 0 is valid for DC.
- Input LEN (Length) must be a constant value.

Input Operands

Operand	*Data Type*	*Memory Area*	*Description*
i	(FBD only.) Solve order for the instruction.		

Power flow (LD)			When On, MOVE_DATA_EX executes. When Off, MOVE_DATA_EX does not execute.
EN (FBD)	BOOL variable or BOOL system variable	I, Q, M, T, S, SA, SB, SC, G, discrete symbolic, discrete I/O variable	Enable input. When On, MOVE_DATA_EX solves. When Off, MOVE_DATA_EX does not solve.
	Bit reference in non-BOOL variable	R, P, L, AI, AQ, W, I, Q, M, T, G, non-discrete symbolic, non-discrete I/O variable	
DC (Optional.)	▪ LD: data flow ▪ FBD: BOOL variable or BOOL system variable ▪ ST: BOOL variable, BOOL system variable, or BOOL constant	data flow, I, Q, M, T, G, symbolic, I/O variable	Data Coherency. ▪ If True, symbolic memory being written to is locked, enabling a coherent copying of symbolic data from one Controller memory area to another. ▪ If False (default), symbolic data is copied normally from one Controller memory area to another *without* data coherency. **Notes** ▪ DC should be used only when using interrupt blocks and is required only when the same memory is used in more than one interrupt block, or in the main program and an interrupt block. ▪ If DC is True, an interrupt block cannot preempt the copy operation. ▪ If DC is False or not present, then interrupts

			can preempt the copy.
IN	▪ Enumerated variable, array of enumerated variables, structure variable, or array of structure variables. ▪ (LD only.) Constant value 0.	I, Q, M, T, S, SA, SB, SC, G, R, P, L, AI, AQ, W, symbolic. data flow only if IN is connected to output Q of another MOVE_DATA_EX, MOVE_FROM_FLAT, or MOVE_TO_FLAT instruction.	Symbolic data to copy to the variable assigned to output Q as determined by the constant value assigned to LEN (Length). **Notes** ▪ The variable assigned to IN must be of the same data type as the variable assigned to output Q, except when the constant 0 (LD only) is assigned to IN. ▪ (LD only.) If the constant 0 is assigned to IN, then LEN (Length) is assigned the constant 1 and the variable or structure assigned to Q is set to its default (original) value.
▪ LEN (FBD) ▪ Length (ST)	INT constant in all languages	N/A	Length of IN; number of IN elements to copy. Valid range: 1 through 32,767. Default: 1. **Tip:** In the LD editor, double-click MOVE_DATA_EX to edit the LEN (Length) constant value.

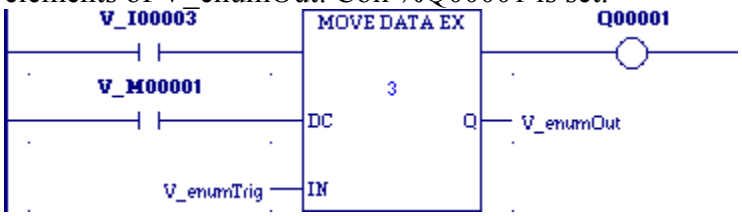
Output Operands

Operand	*Data Type*	*Memory Area*	*Description*
Power flow (LD; optional.)			On when MOVE_DATA_EX executed successfully
ENO (FBD; optional.)	BOOL variable	data flow, I, Q, M, T, S, SA, SB, SC, G, discrete symbolic, discrete I/O variable	ENO is set to On if EN is On.
	Bit reference in non-BOOL variable	R, P, L, AI, AQ, W, I, Q, M, T, G, non-discrete symbolic, non-discrete I/O variable	
Q	Enumerated variable, array of enumerated variables, structure variable, or array of structure variables	I, Q, M, T, S, SA, SB, SC, G, R, P, L, AI, AQ, W, symbolic data flow only if Q is connected to input IN of another MOVE_DATA_EX, MOVE_FROM_FLAT, or MOVE_TO_FLAT instruction	Variable or array to which IN is copied. **Note:** The variable assigned to Q must be of the same data type as the variable assigned to input IN, except when the constant 0 (LD only) is assigned to IN.

Examples

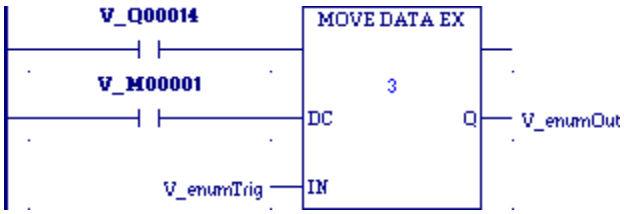
Example 1

When %I00003 is set, the first three elements of V_enumTrig are copied to the first three elements of V_enumOut. Coil %Q00001 is set.



Example 2

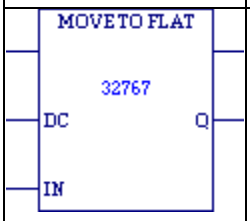
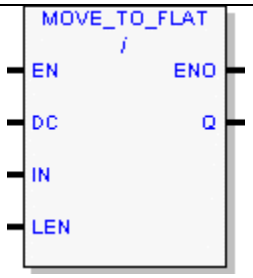
V_enumTrig and V_enumOut are both enumerated arrays of three elements in each array. Because Controllers do not recognize arrays, input Length should be 3, for the total number of enumerated elements to be copied. When the enabling input V_Q0014 is On, MOVE_DATA_EX copies three elements from memory location V_enumTrig to memory location V_enumOut.



CPU Support

MOVE_DATA_EX is supported for PACSystems with firmware version 6.00 or later.

MOVE_TO_FLAT

LD	FBD	ST
		Formal convention: MOVE_TO_FLAT(DC := , IN := , Length := , Q =>); - or - MOVE_TO_FLAT(IN := , Length := , Q =>); **Tip:** Drag instruction from Toolchest's LD Instructions or FBD Instructions drawer to your ST logic. The instruction with its input and output operand names are inserted in ST logic. Now assign a variable to the inputs and outputs, and a variable to the left of the assignment operator.

Operation

MOVE_TO_FLAT instruction copies data from symbolic or I/O variable memory to reference memory. MOVE_TO_FLAT copies across mismatching data types for an operation such as a Modbus transfer. MOVE_TO_FLAT provides optional data coherency by locking the reference memory being written to during the copy operation. This enables a large number of bytes to be copied at one time.

Notes

- The input DC should be used only when using interrupt blocks and is required only when the same memory is used in more than one interrupt block, or in the main program and an interrupt block.
- If DC is True, an interrupt block cannot preempt the copy operation.
- If DC is False or not present, then interrupts can preempt the copy.
- Using DC can impact interrupt latency if the amount of data copied is large.

Copying arrays and array elements

(LD, FBD.) The constant value assigned to input LEN (Length) determines the number of UDT array elements to be copied to the reference memory of the variable assigned to output Q.

Example: If the constant value 6 is assigned to LEN (length), then there should be a UDT array of at least six elements assigned to input IN. When logic executes, *n* bytes of data are copied from the UDT array elements to the reference memory of the variable assigned to output Q, where *n* is the length of the UDT array element (in bytes) times six.

Bounds check

If the number of bytes in the reference memory of the variable assigned to output Q is greater than the memory limit configured in Controller memory, an error appears in the  Feedback Zone and prevents downloading to the Controller.

Note: The constant value assigned to input LEN (Length) is not considered in this check.

Example: If the reference memory of the variable assigned to output Q is %R1 and the length (in bytes) of the UDT assigned to input IN is 6, then the highest used reference address is %R3. If the %R memory limit in the Controller is less than 3, then downloading to the Controller is prevented and an error appears in the Feedback Zone: Error 8038: %R memory usage exceeds limits in <block name>.
When MOVE_TO_FLAT receives power flow in LD, or executes in FBD or ST, the following occur:

- Data is copied from one location in Controller memory to another as determined by the constant value assigned to input LEN (Length).
- (LD.) The number inside MOVE_TO_FLAT is the number of UDT elements to be copied from. This number is the same as the input LEN value in FBD or ST. Valid range: 1 through 32,767.
- (Input Data Coherency optional.) If input DC is True, the reference memory of the variable assigned to output Q is locked, enabling a coherent copy of data from the UDT of input IN.
- If input DC is False (default), data is copied normally from the UDT to the reference memory of the variable assigned to Q without data coherency; that is, without locking the reference memory being copied to.

Operands

Input Operands

Operand	*Data Type*	*Memory Area*	*Description*
i	(FBD only.) Solve order for the instruction.		
Power flow (LD)			When On, MOVE_TO_FLAT executes. When Off, MOVE_TO_FLAT does not execute.
EN (FBD)	BOOL variable or BOOL system variable	I, Q, M, T, S, SA, SB, SC, G, discrete symbolic, discrete I/O variable	Enable input. When On, MOVE_TO_FLAT solves. When Off, MOVE_TO_FLAT does not solve.

	Bit reference in non-BOOL variable	R, P, L, AI, AQ, W, I, Q, M, T, G, non-discrete symbolic, non-discrete I/O variable	
DC (Optional.)	▪ LD: data flow ▪ FBD: BOOL variable or BOOL system variable ▪ ST: BOOL variable, BOOL system variable, or BOOL constant	data flow, I, Q, M, T, G, S, R, P, L, AI, AQ, W, symbolic, I/O variable	Data Coherency. ▪ If True, the reference memory being copied to is locked. This enables a coherent copy of a UDT to reference memory. ▪ If False (default), the UDT is copied normally to reference memory without locking the reference memory area being copied to; that is, *without* data coherency. **Notes** ▪ DC should be used only when using interrupt blocks and is required only when the same memory is used in more than one interrupt block, or in the main program and an interrupt block. ▪ If DC is True, an interrupt block cannot preempt the copy operation. ▪ If DC is False or not present, then interrupts can preempt the copy.
IN	User-defined Data Type (UDT) variable or UDT array	Discrete or non-discrete symbolic, discrete or non-discrete I/O variable	The data copied to the reference memory that the variable assigned to Q is mapped to. **Notes** ▪ Input LEN (Length) determines how many UDT variable elements to copy to Q. ▪ If an array head is assigned to input IN, the constant value assigned to LEN (Length) determines how many UDT array elements assigned to IN are copied to reference memory.

▪ LEN (FBD) ▪ Length (ST)	INT constant in all languages	N/A	Length of IN; number of UDT elements to copy to reference memory in output Q. Valid range: 1 through 32,767. Default: 1. **Tip:** In the LD editor, double-click MOVE_TO_FLAT to edit the LEN (Length) constant value.

Output Operands

Operand	*Data Type*	*Memory Area*	*Description*
Power flow (LD; optional.)			On when MOVE_TO_FLAT executed successfully
ENO (FBD; optional.)	BOOL variable	data flow, I, Q, M, T, S, SA, SB, SC, G, discrete symbolic, discrete I/O variable	ENO is set to On if EN is On
	Bit reference in non-BOOL variable	R, P, L, AI, AQ, W, I, Q, M, T, G, non-discrete symbolic, non-discrete I/O variable	
Q	BYTE, WORD, or array of BYTEs or WORDs	All memory areas except [%S, discrete symbolic, discrete I/O variable]	The variable, mapped to reference memory, that IN is copied to. The amount of data copied is determined by the constant value assigned to input LEN (Length). **Notes** ▪ Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ). ▪ BYTE arrays must be packed; that is, they must be in discrete memory.

Example

Consider the following:

- A UDT variable or UDT array is assigned to input IN.
- The constant value 8 is assigned to input LEN (Length).
- A DWORD variable mapped to %R1 is assigned to output Q.

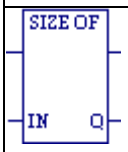
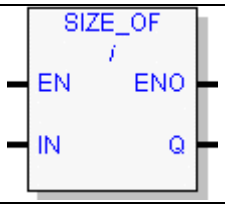
If the constant value 8 assigned is assigned to LEN (length), there should be a UDT array of at least eight elements assigned to IN. When MOVE_TO_FLAT executes, *n* bytes of

data are copied from the UDT variable or array to %R memory, starting at %R1 in the example, where *n* is the length of a UDT array element (in bytes) times eight.

CPU Support

MOVE_TO_FLAT is supported for PACSystems with firmware version 6.00 or later.

Size Of

LD	FBD	ST
		Formal convention: Q := SIZE_OF(In := [input]);

Operation

SIZE_OF counts the number of bits used by the variable assigned to input IN.

Output

- In LD and FBD, SIZE_OF writes the number of bits to output Q.
- In ST, the value is assigned to the variable to the left of the assignment operator, represented by variable Q in the ST code above.

A validation error occurs if you assign one of the following to input IN:

- A BYTE array in non-discrete memory
- A double-segment structure variable

Tip: If the variable assigned to input IN of SIZE_OF is passed to a parameterized C block for processing, also pass the value of output Q to the block. In C block logic, use the value of output Q to ensure that you process all the bits used by the variable without exceeding the end of the variable.

Operands

Input Operands

Operand	*Data Type*	*Memory Area*	*Description*
i	(FBD only.) The solve order for the instruction.		
power flow (LD only)			When set to On, SIZE_OF executes. When set to Off, SIZE_OF does not execute.
EN (FBD only)	BOOL variable	data flow, I, Q, M, T, S, SA, SB, SC, G, discrete symbolic, I/O variable	Enable input. When set to On, SIZE_OF solves. When set to Off, SIZE_OF does not solve.
	Bit reference in non-BOOL variable	R, P, L, AI, AQ, W, I, Q, M, T, G, non-discrete symbolic, I/O variable	
IN	Variable of any data type except BYTE arrays in non-	data flow, I, Q, M, T, S, SA, SB, SC, G, R, P, L,	The variable whose size in bits is

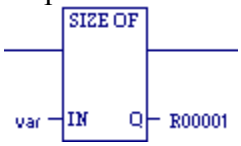
	discrete memory and double-segment structures	AI, AQ, W, symbolic, I/O variable	calculated

Output Operands

Operand	*Data Type*	*Memory Area*	*Description*
power flow (LD only; optional)			Set to On when SIZE_OF has executed successfully
ENO (FBD only; optional)	BOOL variable	data flow, I, Q, M, T, S, SA, SB, SC, G, discrete symbolic, I/O variable	ENO is set to On if EN is set to On.
	Bit reference in non-BOOL variable	R, P, L, AI, AQ, W, I, Q, M, T, G, non-discrete symbolic, I/O variable	
Q	DINT or DWORD variable. ST also supports INT and WORD variables.	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The number of bits used by the variable assigned to input IN. In ST, the value is assigned to the variable on the left side of the assignment operator.

Example

The single-segment structure named var assigned to input IN contains eight BOOL elements (8 * 1 = 8 bits) and twelve WORD elements (12 * 16 = 192 bits). SIZE_OF outputs the value 8 + 192 = 200 to the variable R00001 assigned to output Q.



CPU Support

SIZE_OF is supported for all PACSystems CPUs.

Math

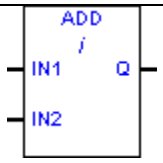
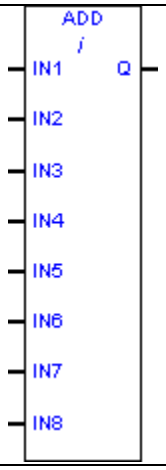
(For PACSystems firmware version 3.50 and later.)
FBD logic may need to include logic to convert data to a different data type before using a math or numerical instruction. The description of each instruction includes information about appropriate data types. The section data type conversion instructions explains how to convert data to a different data type.

Instruction	*Mnemonic*	*Description*
Add	ADD	Adds two through eight numbers.
Divide	DIV	Divides one number by another and outputs the quotient. **Note:** Avoid overflow conditions when performing divisions.
Modulus	MOD	Divides one number by another and outputs the remainder.
Multiply	MUL	Multiplies two through eight numbers. **Note:** Avoid overflow conditions when performing multiplications.
Negate	NEG	Changes the sign and places the result in an output location.
Scale	SCALE	Scales an input parameter and places the result in an output location.
Subtract	SUB	Subtracts one through seven numbers.

Avoiding Overflows

To convert INT to DINT values, the CPU uses standard 2's complement with the sign extended to the highest bit. You must check the sign of the low 16 bits and extend it into the second 16 bits. If the value of the most significant bit in an INT is 0 (positive), move a 0 to all 16 high bits. If the value of the most significant bit in an INT is 1 (negative), move a -1 or hex 0FFFFh to the 16 high bits.
Converting from DINT to INT data is easier, because the low 16 bits (first register) is the integer portion of a DINT (32-bit). The values of the upper 16 bits are either 0 (positive) or -1 (negative), otherwise the DINT number is too large to convert to 16 bits.

Add

Minimum number of inputs: 2	Maximum number of inputs: 8
	

Operation

When the FBD ADD instruction is executed, it adds the operands IN1, IN2, IN3, ... , IN8, and stores the sum in the output variable assigned to Q.
All of the inputs and Q must be of the same data type. The output is calculated when ADD is performed without overflow, unless an invalid operation occurs.

Operands

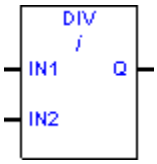
Notes

- If the result of an FBD ADD operation is greater than the maximum or less than the minimum, then the result is platform dependent.
- For FBD instructions that are similar to other supported languages, for example, the FBD ADD instruction and the LD ADD_INT instruction, an underflow or an overflow result may not be the same.
- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN1, IN2, IN3, ... , IN8	INT, DINT, LREAL, REAL, UINT variable or constant. All operands must be of the same data type.	I, Q, M, T, G, R, P, L, W, AI, AQ, symbolic, I/O variable	The inputs to add: Minimum: 2 inputs. Maximum: 8 inputs.

Q	INT, DINT, LREAL, REAL, UINT variable. Must be of the same data type as the inputs.	The sum of the inputs.

Divide



Operation

When the FBD DIV instruction is executed, it divides the operand IN1 by the operand IN2 and stores the quotient in the output variable assigned to Q.

Note: DIV rounds down; it does not round to the closest integer. For example, 24 DIV 5 = 4.

IN1, IN2, and Q must be of the same data type. The output is calculated when DIV is performed without overflow, unless an invalid operation occurs.

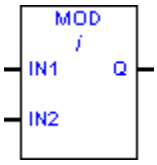
Operands

Notes

- For FBD instructions that are similar to other supported languages, for example, the FBD DIV instruction and the LD DIV_INT instruction, an underflow or an overflow result may not be the same.
- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN1, IN2	INT, DINT, LREAL, REAL, UINT variable or constant. All operands must be of the same data type.	I, Q, M, T, G, R, P, L, W, AI, AQ, symbolic, I/O variable	The values to process
Q	INT, DINT, LREAL, REAL, UINT variable. Must be the same data type as for the inputs.		The quotient of IN1 / IN2

Modulus



Operation

When the FBD Modulo Division (MOD) instruction is executed, it divides the operand IN1 by the operand IN2 of the same and stores the remainder of the division in the output variable assigned to Q.
IN1, IN2, and Q must be of the same data type. The sign of the result is always the same as the sign of input parameter IN1. Output Q is calculated using the formula:

```
Q = IN1-((IN1 DIV IN2) * IN2)
```

where DIV produces an integer number.
The output is always set to ON when the instruction is executed, unless there is an attempt to divide by zero. In that case, the output is set to OFF.

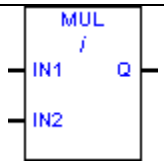
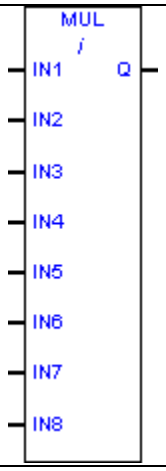
Operands

Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN1, IN2	INT, DINT, UINT variable or constant. All operands must be of the same data type.	I, Q, M, T, G, R, P, L, W, AI, AQ, symbolic, I/O variable	The values to process
Q	INT, DINT, UINT variable. Must be the same data type as for the inputs.		The remainder of IN1/IN2

Multiply

Minimum number of inputs: 2	Maximum number of inputs: 8
	

Operation

When the FBD MUL instruction is executed, it multiplies the operands IN1, IN2, IN3, ... , IN8, and stores the result in the output variable assigned to Q.
All of the inputs and Q must be of the same data type. The output is calculated when the instruction is performed without overflow, unless an invalid operation occurs.

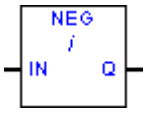
Operands

Notes

- If the result of an FBD MUL operation is greater than the maximum or less than the minimum, then the result is platform dependent.
- For FBD instructions that are similar to other supported languages, for example, the FBD MUL instruction and the LD MUL_INT instruction, an underflow or an overflow result may not be the same.
- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN1, IN2, IN3, ... , IN8	INT, DINT, LREAL, REAL, UINT variable or constant. All operands must be of the same data type.	I, Q, M, T, G, R, P, L, W, AI, AQ, symbolic, I/O variable	The values to multiply: Minimum: 2 inputs. Maximum: 8 inputs.
Q	INT, DINT, LREAL, REAL, UINT variable. Must be the same data type as for the inputs.		The product of the inputs

Negate



Operation

When the FBD NEG instruction is executed, it multiplies the operand IN by -1 and stores the result in the output variable assigned to Q.
IN and Q must be of the same data type. The sign of the result is always the opposite of the sign of the input parameter IN.

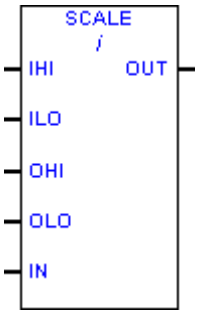
Operands

Notes

- For FBD instructions that are similar to other supported languages, for example, the FBD NEG instruction and the ST Negation operator, an underflow or an overflow result may not be the same.
- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN	INT, DINT, REAL, or LREAL variable or constant. Must be of the same data type as for Q.	I, Q, M, T, G, R, P, L, W, AI, AQ, symbolic, I/O variable	The value to process
Q	INT, DINT, REAL, or LREAL variable		The negated value of IN

Scale



Operation

IHI, ILO, OHI, OLO, IN, OUT, and Q must be of the same data type. When the FBD SCALE instruction is executed, it scales the input operand IN and places the result in the output variable assigned to the output operand OUT.
The output is calculated when the instruction is performed without overflow, unless an invalid operation occurs.

Operands

All operands can map to the following memory areas: I, Q, M, T, G, R, P, L, W, AI, AQ, symbolic, and an I/O variable.

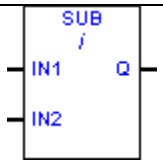
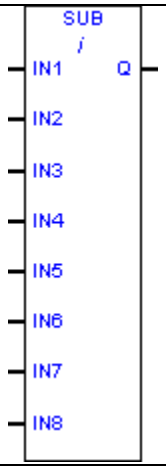
Notes

- If an overflow occurs, the result is the largest possible value with the proper sign.
- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Description*
IHI	DINT, INT, UINT, WORD variable or constant. All operands must be of the same data type.	Input HIgh; maximum input value (module-related). The upper limit of the unscaled data. IHI is used with ILO, OHI and OLO to calculate the scaling factor applied to the input value IN.
ILO	DINT, INT, UINT, or WORD variable or constant	Input LOw; minimum input value (module-related). The lower limit of the unscaled data.
OHI	DINT, INT, UINT, or WORD	Output HIgh; maximum output value. The upper limit of the scaled data. When the IN input is at

	variable or constant	the IHI value, the OUT value is the same as the OHI value.
OLO	DINT, INT, UINT, or WORD variable or constant	Output LOw; minimum output value. The lower limit of the scaled data. When the IN input is at the ILO value, the OUT value is the same as the OLO value.
IN	DINT, INT, UINT, or WORD variable or constant	INput value. The value to be scaled.
OUT	DINT, INT, UINT, or WORD variable or constant	OUTput value. The scaled equivalent of the input value.

Subtract

Minimum number of inputs: 2	Maximum number of inputs: 8
	

Operation

When the FBD SUB instruction is executed, it subtracts the input operand IN2 from the input operand IN1 and stores the result in the output variable assigned to Q.
If there are more inputs (IN3 and so on), the FBD SUB instruction subtracts the input operand IN3 from the intermediate result and stores the result in the output variable assigned to Q.
This process is repeated until all inputs have been subtracted from the intermediate result. All of the inputs and Q must be of the same data type. The output is calculated when SUB is performed without overflow, unless an invalid operation occurs.

Operands

Notes

- If the result of an FBD SUB operation is greater than the maximum or less than the minimum, then the result is platform dependent.
- For FBD instructions that are similar to other supported languages, for example, the FBD SUB instruction and the LD SUB_INT instruction, an underflow or an overflow result may not be the same.
- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		

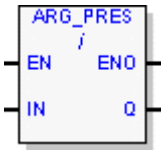
IN1, IN2, IN3, ... , IN8	INT, DINT, LREAL, REAL, UINT variable or constant. All operands must be of the same data type.	I, Q, M, T, G, R, P, L, W, AI, AQ, symbolic, I/O variable	The values to subtract: Minimum: 2 inputs. Maximum: 8 inputs.
Q	INT, DINT, LREAL, REAL, UINT variable. Must be the same data type as for the inputs.		The difference of IN1 - all other inputs

Program Flow

The Program Flow instruction limits program execution or changes the way the CPU executes the application program.

Instruction	*Mnemonic*	*Description*
Argument Present	ARG_PRES	Determines if an input or output parameter value existed when the function block instance of the parameter was invoked.
Call	CALL	Causes program execution to go to a specified subroutine block.

Argument Present



Operation

An argument present (ARG_PRES) instruction determines if an input or output parameter value existed when the function block instance of the parameter was invoked. For example, a parameter may be optional (pass by value).

Note: This instruction must be called from a function block instance or from a parameterized block.

Operands

Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use a BOOL array of length 16 or more instead of a WORD variable. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
EN	BOOL variable	data flow, I, Q, M, T, G, S, SA, SB, SC, discrete symbolic	Enable input. When set to ON, ARG_PRES executes. When set to OFF, ARG_PRES does not execute.
	Bit reference in non-BOOL variable	I, Q, M, T, G, S, SA, SB, SC, R, P, L, AI, AQ, W, non-discrete symbolic	
IN	▪ BOOL, DINT, DWORD, INT, REAL, UINT, WORD variable ▪ Variable array head name or variable array head name element [000]	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable BOOL, WORD, and DWORD variables can also use S, SA, SB, SC	Input parameter value. **Notes** ▪ IN cannot be a constant. ▪ For an array variable named *myArray*, specify *myArray* or *myArray [000]*.

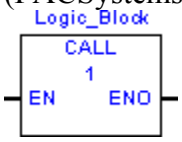
ENO	BOOL variable	I, Q, AI, AQ, R, M, W, I/O variable	(Optional.) Set to ON whenever ARG_PRES receives data unless: ▪ Not all references of the type specified are present within the selected range. ▪ The CPU is not able to properly handle the temporary list of I/O created by the function. ▪ The range specified includes I/O modules that are associated with a "Loss of I/O" fault.
	Bit reference in non-BOOL variable	I, Q, M, T, G, SA, SB, SC, R, P, L, AI, AQ, W, non-discrete symbolic	
Q	BOOL variable	data flow, I, Q, M, T, S, G, symbolic, I/O variable	▪ True: Input or output parameter value exists. ▪ False: (Default.) Input or output parameter value does not exist. **Note:** Q is required.

Example

See LD and ST examples.

Call

(PACSystems firmware version 3.50 and later.)



The FBD CALL instruction is a control instruction, which enables you to call and transfer control to other Logic Developer - PLC logic blocks within this Machine Edition target. When the called Controller logic block (this can be an LD, ST, , FBD, or a parameterized block) has been solved, control is transferred to the next FBD instruction in the solve order.

Note: The Call instruction cannot transfer control to a logic block outside of this target.

Parameter Data Types

Valid parameter data types for the CALL instruction are:

EN must be:	**ENO must be:**
BOOL	BOOL

If EN is set to **ON (1)**, the FBD CALL instruction executes; that is, the Controller logic block represented by the logic block name above the CALL instruction is executed.
If the input parameter EN is set to **OFF (0)**, the FBD CALL instruction doesn't execute; that is, the Controller logic block represented by the name above the CALL instruction is not called. The next FBD instruction in the solve order is then solved.

Notes

- You can use the CALL instruction in any FBD, including the _MAIN block, a parameterized block, or a function block.
- You cannot call the _MAIN block.
- The called block must exist in the target before you make the call.
- The maximum number of calls *to* other blocks is limited only by available memory.
- When you view an FBD block in the FBD editor and you select a CALL instruction, the following takes place:
  - If the  Inspector is open, it displays the properties of the block being called. (To open the Inspector, press SHIFT+F7.)
  - If the  Feedback Zone is open, its References tab displays the location of all calls to the block called by the CALL instruction. This applies regardless of the language of the called block. (To open the Feedback Zone, press SHIFT+F6.)
- You can set up recursive subroutines by having a block call itself. A target's default stack size allows at least eight nested calls before a stack overflow fault is logged and the Controller transitions to STOP/FAULT mode. To enable a deeper nesting of calls, increase the program stack size.
- When a parameterized block's Y0 parameter is set to ON, the ENO parameter of the CALL instruction is set to ON; when it returns OFF, the ENO parameter is set to OFF.
- Undo is not available when you change the name of the block being called.

Operands

Note: You can use symbolic or I/O variables for any operand and data type.

Operand	*Description*
Logic_block	Block name; the name of the block to transfer to. ▪ You cannot call the _MAIN block. ▪ You cannot call a block before actually creating the block. If the block name is invalid, an error message displays and the name is ignored. **Note:** Undo is not available when you change the name of *Logic_block*; that is, the block being called.
The parameter names are user-defined.	**Notes for parameterized C blocks** ▪ The valid data type, value range, and memory area for each parameter are stated in the C block's written documentation. ▪ You must set the C block's parameters. ▪ The names you have assigned to the parameters appear on the CALL

	instruction. **Notes for parameterized LD blocks** ▪ You must set the parameterized block's parameters. ▪ A parameterized block is not required to have the same number of inputs and outputs. ▪ The names you have assigned to the parameters appear on the CALL instruction. ▪ Valid operands on the CALL instruction include variables and indirect references. Input operands can also be constants. ▪ If a formal parameter is an array of BOOL type and has a length evenly divisible by 16, then a variable or array residing in %R memory can be passed to the parameterized block as an operand. For example, if a parameterized block has a formal parameter Y1 of data type BIT and length 48, you can pass a WORD array of length 3 to Y1. ▪ You do not need to define the BOOL parameter Y0 to use it in the parameterized block's logic. When the parameterized block stops executing and Y0 is set to ON, the ENO parameter of the CALL instruction is set to ON. If Y0 is OFF, the ENO parameter of the CALL instruction is set to OFF. ▪ If Y0 is not used, its state defaults to ON.

Timers

(OFDT, ONDTR, TMR timer built-in function blocks for PACSystems firmware version 3.50 and later.)

Instruction	*Mnemonic*	*Description*
Off Delay Timer	OFDT_HUNDS OFDT_SEC OFDT_TENTHS OFDT_THOUS	The built-in function block timer's Current Value (CV) resets to zero when the EN input parameter is set to ON. CV increments while EN is set to OFF. When CV=PV (Preset Value), the ENO output parameter is set to OFF until EN is set to ON again.
On Delay Stopwatch Timer	ONDTR_HUNDS ONDTR_SEC ONDTR_TENTHS ONDTR_THOUS	Retentive on delay built-in function block timer. Increments while the EN input parameter is set to ON and holds its value when the ENO output parameter is set to OFF.
On Delay Timer	TMR_HUNDS TMR_SEC TMR_TENTHS TMR_THOUS	Simple on delay built-in function block timer. Increments while the EN input parameter is set to ON and resets to zero when the ENO output parameter is set to OFF.

Standard function block	*Mnemonic*	*Description*
Timer Off Delay	TOF	The timer off delay standard function block delays setting the output, Q, to OFF, for the preset time, PT, when you set the input, IN, to OFF.
Timer On Delay	TON	The timer on delay standard function block delays setting the output, Q, to ON, when you set the input, IN, to ON, for the preset time, PT.
Timer Pulse	TP	The timer pulse standard function block outputs a pulse of a duration determined by the preset time, PT, when you set the input, IN, to ON.

Note: The data associated with timers is retentive through power cycles.

Data Required for the OFDT, ONDTR, and TMR Timers

Each OFDT, ONDTR, and TMR built-in function block timer uses a three-WORD array of %R, %W, %P, %L, symbolic, or I/O variable memory area to store the following information:

Current value (CV)	Word 1
Preset value (PV)	Word 2
Control word	Word 3

When you enter a timer, you must enter a beginning address for the three-word array (three-word block of registers).

Warning: Do not use two consecutive words (registers) as the starting addresses of two timers. Logic Developer - PLC does not check or warn you if register blocks overlap. Timers do not work if you place the current value of a second timer on top of the preset value for the previous timer.

Word 1: Current value (CV)

Warning: The first word (CV) can be read but should not be written to, or the built-in function block may not work.

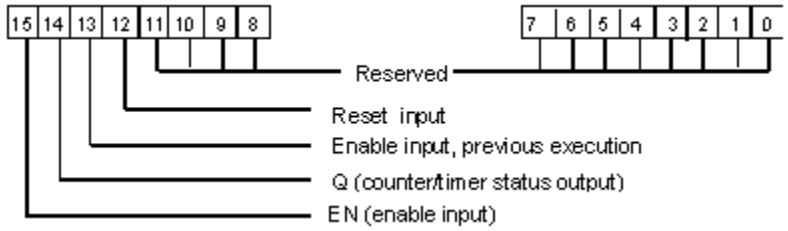
Word 2: Preset value (PV)

When the Preset Value (PV) operand is a variable, it is normally set to a different location than word 2 in the built-in function block timer.

- If you use a different address and you change word 2 directly, your change has no effect, because PV overwrites word 2.
- If you use the same address for the PV operand and word 2, you can change the Preset Value in word 2 while the timer is running and the change is effective.

Word 3: Control word

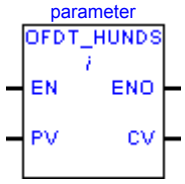
The control word stores the state of the boolean inputs and outputs of its associated timer, as shown in the following diagram:



Warning: The third word (Control) can be read but should not be written to; otherwise, the built-in function block does not work.

Note: Bits 0 through 13 are used for timer accuracy.

Off Delay Timer



Mnemonics:
OFDT_SEC
OFDT_TENTHS
OFDT_HUNDS
OFDT_THOUS

Operation

The Off-Delay Timer (OFDT) built-in function block increments while the EN input parameter is set to OFF, and the timer's Current Value (CV) resets to zero when EN is set to ON. OFDT passes data until the specified interval PV (Preset Value) has elapsed. Because there is no automatic initialization to the ENO output parameter at power-up, the value of ENO is retentive across power failure.
Time can be counted in various increments:

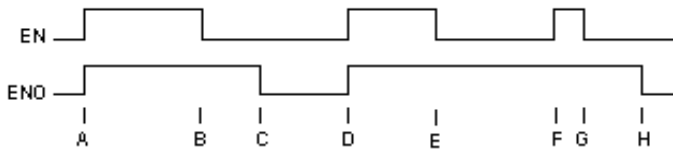
- **Seconds**
- **Tenths** (0.1) of a second
- **Hundredths** (0.01) of a second
- **Thousandths** (0.001) of a second

The range for PV is 0 through +32,767 time units. If PV is out of range, it has no effect on the timer's word 2.
When EN is first set to ON, CV is set to zero and ENO is set to ON.
ENO remains set to ON as long as EN is set to ON.
Each time OFDT is invoked with EN set to OFF, CV is updated to reflect the elapsed time since the timer was reset. ENO remains set to ON until CV equals or exceeds PV. When this happens, ENO is set to OFF and OFDT stops accumulating time. If PV equals 0 or is negative, ENO is set to OFF the first time it is invoked with EN set to OFF.
When EN is set to ON again, CV resets to zero.

Notes

- The best way to use an OFDT is to invoke it with a particular reference address exactly one time each scan. Do not invoke an OFDT with the same reference address more than once per scan (inappropriate accumulation of time would result). When an OFDT appears in an FBD block, it accumulates time once per scan. Subsequent calls to that same FBD block within the same scan has no effect on its OFDT.
- Do not program an OFDT with the same reference address in two different blocks.
- If you use recursion, that is, you have a block call itself either directly or indirectly, program the FBD block so that it invokes the timer before it makes any recursive calls to itself.
- See Using OFDT, ONDTR, and TMR Timers in PACSystems Parameterized FBD Blocks.
- See OFDT, ONDTR, and TMR Timer Limitations in User-defined Function Blocks (UDFBs).
- An OFDT sets its ENO parameter to OFF the first scan that EN is set to OFF if the previous scan time was greater than PV.
- When OFDT is used in a program block that is not called every scan, the timer accumulates time between calls to the program block unless it is reset. This means that OFDT operates like a timer operating in a program with a much slower scan than the timer in the main program block. For program blocks that are inactive for a long time, OFDT should be programmed to allow for this catch-up feature. For example, if a timer in a program block is reset and the program block is not called (is inactive) for four minutes, when the program block is called, four minutes of time will already have accumulated. If EN is set to OFF, these four minutes are applied to the timer, that is, CV is set to four minutes.

Timing diagram



A. EN and ENO are both set to ON; timer is reset (CV = 0).
B. EN is set to OFF; timer starts accumulating time.
C. CV reaches PV; ENO is set to OFF and timer stops accumulating time.
D. EN is set to ON; timer is reset (CV = 0).
E. EN is set to OFF; timer starts accumulating time.
F. EN is set to ON; timer is reset (CV = 0) before CV had a chance to reach PV. (The diagram is not to scale.)
G. EN is set to OFF; timer begins accumulating time.
H. CV reaches PV; ENO is set to OFF and timer stops accumulating time.

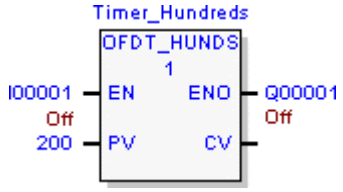
Operands

Operand	*Data Type*	*Memory Area*	*Description*
parameter	one-dimensional WORD array of 3 words	R, P, L, W, symbolic	Control parameter. This is a three-word WORD array: Word 1: Current value (CV) Word 2: Preset value (PV)

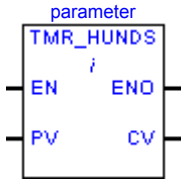
			Word 3: Control word **Cautions** ▪ Do not write to word 3 by any means. ▪ Overlapping reference addresses may cause erratic timer operation. **Note:** Undo is available when you edit the control parameter.
i	The solve order for the built-in function block.		
EN	BOOL variable	data flow, I, Q, M, T, G, S, SA, SB, SC, discrete symbolic, I/O variable	Enable input. See Operation.
	Bit reference in non-BOOL variable	R, P, L, AI, AQ, W, I, Q, M, T, G, non-discrete symbolic, I/O variable	
PV	INT variable or constant; BOOL array of length 16 or more (restrictions apply)	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	(Optional.) The Preset Value, used when the timer is enabled or reset. 0 ≤ PV ≤ +32,767. If PV is out of range, it has no effect on Word 2. **Note:** Instead of using the PV parameter, you can write directly to word 2 by using a MOV instruction or an operator interface.
ENO	BOOL variable	data flow, I, Q, M, T, G, discrete symbolic, I/O variable	(Optional.) Output. See Operation.
	Bit reference in non-BOOL variable	R, P, L, AI, AQ, W, I, Q, M, T, G, non-discrete symbolic, I/O variable	
CV	INT variable;BOOL array of length 16 or more (restrictions apply)	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	(Optional.) The current value of the timer; the same value as Word 1 in the control parameter.

Example

When %I00001 transitions from ON (1) to OFF (0), the Timer starts to time. When the Timer reaches a count of 200 (2 seconds), then %Q00001 is set to OFF (0).



On Delay Timer



Mnemonics:
TMR_SEC
TMR_TENTHS
TMR_HUNDS
TMR_THOUS

Operation

The On-Delay Timer (TMR) built-in function block increments while the EN input parameter is set to ON and resets to zero when EN is set to OFF. The ENO output parameter is set to ON after the specified interval PV (Preset Value) has elapsed, as long as EN is set to ON.
Time can be counted in various increments:

- **Seconds**
- **Tenths** (0.1) of a second
- **Hundredths** (0.01) of a second
- **Thousandths** (0.001) of a second

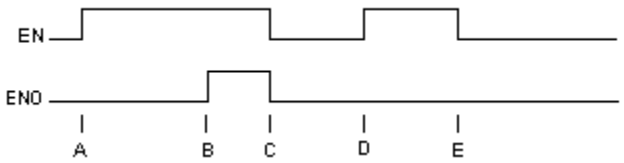
The range is 0 through +32,767 time units. The state of this timer is retentive on power failure; no automatic initialization occurs at power-up.
When TMR is invoked with EN set to OFF, then its Current Value (CV) is reset to zero, and ENO is set to OFF. Each time the TMR is invoked with EN set to ON, CV is updated to reflect the elapsed time since the timer was reset. When CV reaches PV, ENO is set to ON.

Notes

- The best way to use a TMR built-in function block is to invoke it with a particular reference address exactly one time each scan. Do not invoke a TMR with the same reference address more than once per scan (inappropriate accumulation of time would result). When a TMR appears in an FBD block, it will only accumulate time once per scan. Subsequent calls to that same FBD block within the same scan will have no effect on its TMRs.
- Do not program a TMR with the same reference address in two different blocks.
- If you use recursion (that is, having a block call itself either directly or indirectly), program the FBD block so that it invokes the timer before it makes any recursive calls to itself.
- See Using OFDT, ONDTR, and TMR Timers in PACSystems Parameterized FBD Blocks.
- See OFDT, ONDTR, and TMR Timer Limitations in User-defined Function Blocks (UDFBs).
- A TMR timer expires, that is, ENO is set to ON, the first scan that EN is set to ON if the previous scan time was greater than PV.
- When TMR is used in a program block that is not called every scan, TMR accumulates time between calls to the program block unless it is reset. This means that it operates like a timer operating in a program with a much slower sweep than the timer in the main program block. For program blocks that are inactive for a long time, TMR should be programmed to allow for this

catch-up feature. For example, if a timer in a program block is reset and the program block is not called (is inactive) for 4 minutes, when the program block is called, four minutes of time will already have accumulated. If the enable input is ON, these four minutes are applied to the timer, that is, CV is set to 4 minutes.

Timing Diagram



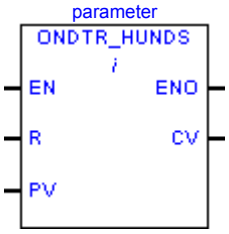
A. EN is set to ON; timer begins accumulating time.
B. CV reaches PV; ENO is set to ON and timer continues accumulating time.
C. EN is set to OFF; ENO is set to OFF; timer stops accumulating time and CV is cleared.
D. EN is set to ON; timer starts accumulating time.
E. EN is set to OFF before current value reaches PV; ENO remains set to OFF; timer stops accumulating time and is cleared to zero (CV=0).

Operands

Operand	*Data Type*	*Memory Area*	*Description*
parameter	one-dimensional WORD array of 3 words	R, P, L, W, symbolic	Control parameter. This is a three-word WORD array: Word 1: Current value (CV) Word 2: Preset value (PV) Word 3: Control word **Cautions** ▪ Do not write to word 3 by any means. ▪ Overlapping reference addresses may cause erratic timer operation. **Note:** Undo is available when you edit the control parameter.
i	The solve order for the built-in function block.		
EN	BOOL variable	data flow, I, Q, M, T, G, S, SA, SB, SC, discrete symbolic, I/O variable	Enable input. See Operation.

	Bit reference in non-BOOL variable	I, Q, M, T, G, S, SA, SB, SC, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	
PV	INT variable or constant; BOOL array of length 16 or more (restrictions apply)	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	(Optional.) The Preset Value, used when the timer is enabled or reset. 0 ≤ PV ≤ +32,767. If PV is out of range, it has no effect on Word 2. **Note:** Instead of using the PV parameter, you can write directly to word 2 by using a MOV instruction or an operator interface.
ENO	BOOL variable	data flow, I, Q, M, T, G, S, SA, SB, SC, discrete symbolic, I/O variable	(Optional.) Output. See Operation.
	Bit reference in non-BOOL variable	I, Q, M, T, G, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	
CV	INT variable; BOOL array of length 16 or more (restrictions apply)	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	(Optional.) The current value of the timer built-in function block; the same value as Word 1 in the control parameter.

On Delay Stopwatch Timer



Mnemonics:
ONDTR_SEC
ONDTR_TENTHS
ONDTR_HUNDS
ONDTR_THOUS

Operation

The retentive On-Delay Stopwatch Timer (ONDTR) built-in function block increments while the EN input parameter is set to ON, and it holds its value when EN is set to OFF. Time can be counted in various increments:

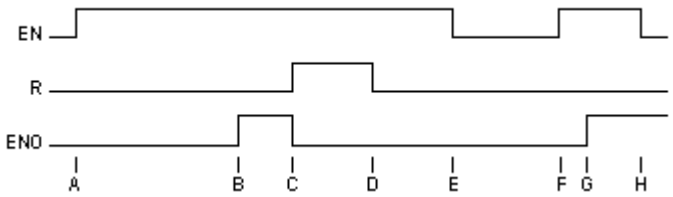
- **Seconds**
- **Tenths** (0.1) of a second
- **Hundredths** (0.01) of a second
- **Thousandths** (0.001) of a second

The range is 0 through +32,767 time units. Because no automatic initialization to ENO occurs at power-up, ENO is retentive across power failure.
When EN is first set to ON, ONDTR starts accumulating time (Current Value (CV)). As long as EN is set to ON, when this timer is encountered in FBD logic, its CV is updated. ONDTR continues accumulating until CV equals or exceeds Preset Value (PV). When this happens, the value of CV is retained and ENO remains set to ON regardless of the state of EN.
When EN is set to OFF, CV stops incrementing and is retained. ENO, if set to ON, remains set to ON. When EN is set to ON again, CV again increments, beginning at the retained value.
When the Reset parameter, R, is set to ON and PV is not equal to zero, then CV is set back to zero and ENO is set to OFF.
If PV equals zero and EN is set to ON, ENO is set to ON. Subsequently setting EN to OFF or setting R to ON has no effect on the timer output; it remains enabled.

Notes

- The best way to use an ONDTR built-in function block is to invoke it with a particular reference address exactly one time each scan. Do not invoke an ONDTR with the same reference address more than once per scan (inappropriate accumulation of time would result). When an ONDTR appears in an FBD block, it will only accumulate time once per scan. Subsequent calls to that same FBD block within the same scan will have no effect on its ONDTRs.
- Do not program an ONDTR with the same reference address in two different blocks.
- If you use recursion (that is, having a block call itself either directly or indirectly), program the FBD block so that it invokes the timer before it makes any recursive calls to itself.
- See Using OFDT, ONDTR, and TMR Timers in PACSystems Parameterized FBD Blocks.
- See OFDT, ONDTR, and TMR Timer Limitations in User-defined Function Blocks (UDFBs).
- An ONDTR expires, that is, ENO is set to ON, the first scan that EN is set to ON and R is set to OFF if the previous scan time was greater than PV.
- When ONDTR is used in a program block that is not called every scan, it accumulates time between calls to the program block unless it is reset. This means that ONDTR operates like a timer operating in a program with a much slower scan than the timer in the main program block. For program blocks that are inactive for a long time, ONDTR should be programmed to allow for this catch-up feature. For example, if a timer in a program block is reset and the program block is not called (is inactive) for four minutes, when the program block is called, four minutes of time will already have accumulated. If EN is set to ON and R is set to OFF, these four minutes are applied to the timer, that is, CV is set to four minutes.

Timing diagram



A. EN is set to ON; timer built-in function block starts accumulating.
B. Current value (CV) reaches preset value (PV); ENO is set to ON. Timer continues to accumulate time until EN is set to OFF, R is set to ON, or current value becomes equal to the maximum time.
C. R is set to ON; ENO is set to OFF, accumulated time is reset (CV=0).
D. R is set to OFF; timer then starts accumulating again, because EN is set to ON.
E. EN is set to OFF; timer stops accumulating. Accumulated time stays the same.
F. EN is set to ON again; timer continues accumulating time.
G. CV becomes equal to PV; ENO is set to ON. Timer continues to accumulate time until EN is set to OFF, R is set to ON, or CV becomes equal to the maximum time.
H. EN is set to OFF; timer stops accumulating time.

Operands

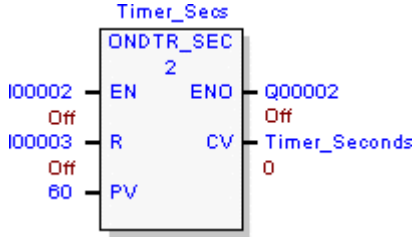
Operand	*Data Type*	*Memory Area*	*Description*
parameter	one-dimensional	R, P, L, W,	Control parameter. This is a three-

	WORD array of 3 words	symbolic	word WORD array: Word 1: Current value (CV) Word 2: Preset value (PV) Word 3: Control word **Cautions** ▪ Do not write to word 3 by any means. ▪ Overlapping reference addresses may cause erratic timer operation. **Note:** Undo is available when you edit the control parameter.
i	The solve order for the built-in function block.		
EN	BOOL variable	data flow, I, Q, M, T, G, S, SA, SB, SC, discrete symbolic, I/O variable	Enable input. See Operation.
	Bit reference in non-BOOL variable	I, Q, M, T, G, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	
R	BOOL variable	data flow, I, Q, M, T, G, S, SA, SB, SC, discrete symbolic, I/O variable	When R is ON, it resets the Current Value (Word 1) to zero.
	Bit reference in non-BOOL variable	I, Q, M, T, G, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	
PV	INT variable or constant; BOOL array of length 16 or more (restrictions apply)	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	(Optional.) The Preset Value, used when the timer built-in function block is enabled or reset. 0 ≤ PV ≤ +32,767. If PV is out of range, it has no effect on Word 2. **Note:** Instead of using the PV parameter, you can write directly to word 2 by using a MOV instruction or an operator interface.

ENO	BOOL variable	data flow, I, Q, M, T, G, discrete symbolic, I/O variable	(Optional.) Output. See Operation.
	Bit reference in non-BOOL variable	I, Q, M, T, G, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	
CV	INT variable; BOOL array of length 16 or more (restrictions apply)	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	(Optional.) Current Value of the timer; the same value as Word 1 in the control parameter.

Example

When %I00002 transitions from OFF (0) to ON (1), the Timer starts to time. When the Timer reaches a count of 60 (6 seconds), then %Q00002 is set to ON (1). %I00003 can be used to reset the Timer.



Using OFDT, ONDTR, and TMR Timers in PACSystems Parameterized FBD Blocks

Finding the Source Block

A parameterized FBD block is a particular type of FBD block that supports parameters. Special care must taken when programming OFDT, ONDTR, and TMR timers in PACSystems parameterized FBD blocks. OFDT, ONDTR, and TMR timers in parameterized FBD blocks can be programmed to track true real time as long as the guidelines and rules below are followed. If the guidelines and rules described here are not followed, then the operation of the OFDT, ONDTR, and TMR timers in parameterized FBD blocks is undefined.

Note: These rules are not enforced by Machine Edition. It is your responsibility to ensure these rules are followed.

The best use of an OFDT, ONDTR, or TMR timer is to invoke it with a particular reference address exactly one time each scan. With parameterized FBD blocks, it is important to use the appropriate reference address with the OFDT, ONDTR, or TMR timer and to call the parameterized FBD block an appropriate number of times.

Note: The OFDT, ONDTR, and TMR timers also have limitations in function blocks, that is, blocks with a type of Function Block. See OFDT, ONDTR, and TMR Timer Limitations in Function Blocks.

Finding the Source Block

The source block is either the _MAIN block or the lowest logic block of type Block that appears above the parameterized FBD block in the call tree. To determine the source block for a given parameterized FBD block, determine which block invoked that parameterized FBD block. If the calling block is _MAIN or of type Block, it is the source block. If the calling block is any other type (Parameterized Block or Function Block), apply the same test to the calling block. Continue back up the call tree until the _MAIN block or a block of type Block is found. This is the source block for the parameterized FBD block.

Programming OFDT, ONDTR, and TMR Timers in Parameterized FBD blocks

Different guidelines and rules apply depending on whether you want to use the parameterized FBD block in more than one place in your program logic.

Parameterized FBD block called from only one block

If your parameterized FBD block that contains an OFDT, ONDTR, or TMR timer will be called from only one logic block, follow these rules:

1. Call the parameterized FBD block exactly one time per execution of its source block.
2. Choose, for the OFDT, ONDTR, or TMR timer, a reference address that is not manipulated elsewhere. The reference address can be %R, %P, %L, %W,

symbolic, I/O variable, or a parameterized FBD block formal parameter (word array passed by reference).

Notes

- %L memory is the same %L memory available to a source block of block type Block. %L memory corresponds to %P memory when the source block is _MAIN.
- Undo is not available when you change the name of the parameterized FBD block being called.

Parameterized FBD block called from multiple blocks

When calling the parameterized FBD block from multiple blocks, it is imperative to separate the OFDT, ONDTR, or TMR timer reference address used by each call to the parameterized FBD block. Even so, the timer may increment incorrectly, but to minimize the risk, follow these rules:

1. Call the parameterized FBD block exactly one time per execution of each source block that it appears in.
2. Choose a %L reference address or parameterized FBD block formal parameter for the OFDT, ONDTR, or TMR timer reference address. Do not use a %R, %P, or %W reference address. Do not use a symbolic or I/O variable.

Notes

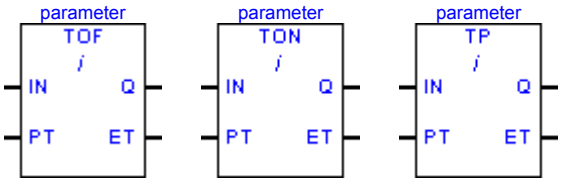
- We strongly recommended a %L location, which is inherited from the parameterized FBD block's source block. Each source block has its own reserved %L memory area, except the _MAIN block, which has a %P memory area instead. When the _MAIN block calls another block, the %P mappings from the _MAIN block are accessed by the called block as %L mappings.
- If you use a parameterized FBD block formal parameter (word array passed by reference), then the actual parameter that corresponds to this formal parameter must be a %L, %R, %P, %W reference address, symbolic, or I/O variable. If the actual parameter is a %R, %P, %W reference address, symbolic, or I/O variable, then a unique reference address must be used by each source block.
- Undo is not available when you change the name of the parameterized FBD block being called.

Recursion

If you use recursion, that is, if you have a block call itself either directly or indirectly, and your parameterized FBD block contains an OFDT, ONDTR, or TMR timer, then you must follow two additional rules:

1. Program the source block so that it invokes the parameterized FBD block before making any recursive calls to itself.
2. Do not program the parameterized FBD block to call itself directly.

TOF, TON, TP Timer Standard Function Blocks



Operation

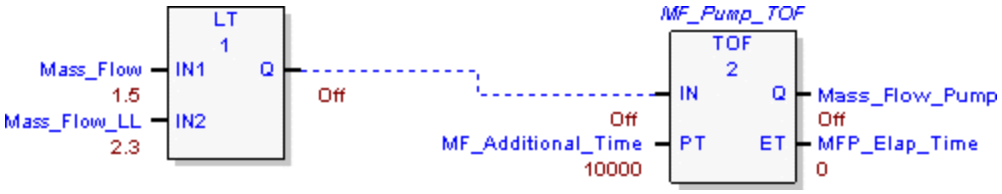
The FBD timer off delay (TOF) standard function block sets the output, Q, to ON, when the input, IN, is set to ON. TOF delays setting Q to ON for a specified duration, PT, after IN is set to OFF.
The FBD timer on delay (TON) standard function block delays setting the output, Q, to ON, after the input, IN, is set to ON, for a specified duration, PT.
The FBD timer pulse (TP) standard function block outputs a pulse of a specified duration, PT, once the input, IN, is set to ON.

Notes

- Forcing or writing values to the IN, PT, Q, ET, ENO, or TI timer standard function block instance data elements may cause indeterminate results.
- TOF requires instance data of data type TOF; TON requires instance data of data type TON; TP requires instance data of data type TP.
- Instance data for a timer standard function block can be a parameter of the current user-defined function block (UDFB) or parameterized block, or it can be a variable.
- When you assign a structure variable as instance data to an FBD instruction, its name appears immediately above the instruction. Undo is available.

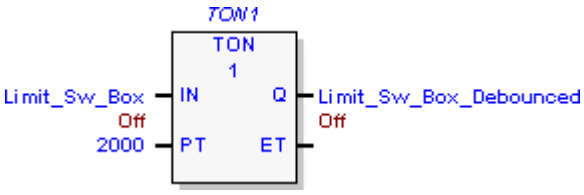
TOF Example

When the Mass Flow drops below the Mass Flow Lower Limit, start the Mass Flow Pump. When the Mass Flow rises above the Lower Limit, continue running the pump for the MF Additional Time.

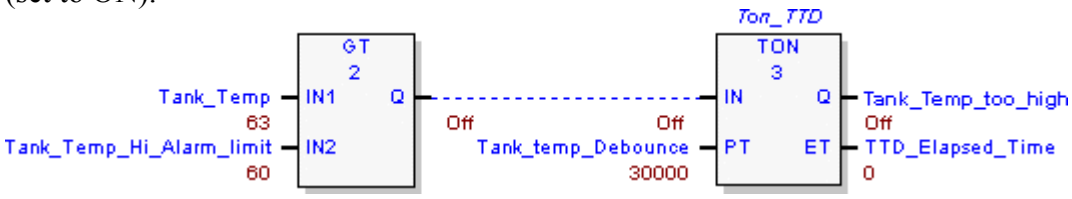


TON Examples

When the Box Limit Switch is set to ON, the on-delay timer must be set to ON for 2 seconds before the "debounced" input is activated.

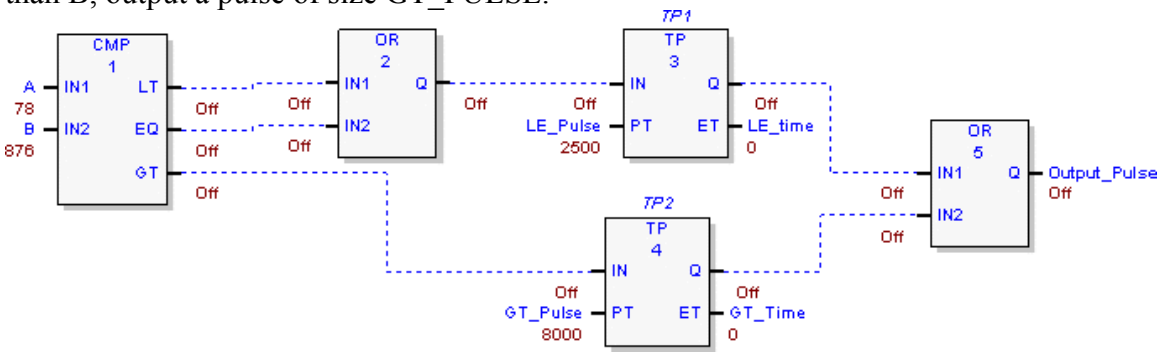


When the tank temperature exceeds the high alarm limit for more than the tank temperature debounce time (in milliseconds), the tank temp too high variable is activated (set to ON).



TP Example

When A is less than or equal to B, output a pulse of size LE_PULSE. When A is greater than B, output a pulse of size GT_PULSE.



Type Conversion

(For PACSystems firmware version 3.50 and later.)
FBD data type conversion instructions change a data item from one value format (data type) to another.

Note: Many programming instructions, such as math instructions, must be used with data of one data type. As a result, a data conversion is often required before using those instructions.

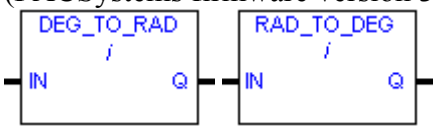
Instruction	*Mnemonic*	*Description*
Convert angles		
Convert Angles	DEG_TO_RAD	Converts degrees to radians
	RAD_TO_DEG	Converts radians to degrees
Convert BCD4		
Convert BCD4 to INT	BCD4_TO_INT	Converts BCD4 (4-digit Binary-Coded-Decimal) to INT (16-bit signed integer)
Convert BCD4 to REAL	BCD4_TO_REAL	Converts BCD4 to REAL (32-bit signed floating-point)
Convert BCD4 to UINT	BCD4_TO_UINT	Converts BCD4 to UINT (16-bit unsigned integer)
Convert BCD8		
Convert BCD8 to DINT	BCD8_TO_DINT	Converts BCD8 (8-digit Binary-Coded-Decimal) to DINT (32-bit signed integer)
Convert BCD8 to REAL	BCD8_TO_REAL	Converts BCD8 to REAL (32-bit signed floating-point)
Convert DINT		
Convert DINT to BCD8	DINT_TO_BCD8	Converts DINT (32-bit signed integer) to 8-digit Binary-Coded-Decimal (BCD8)
Convert DINT to DWORD	DINT_TO_DWORD	Converts DINT to DWORD (32-bit bit string)
Convert DINT to INT	DINT_TO_INT	Converts DINT to INT (16-bit signed integer)
Convert DINT to LREAL	DINT_TO_LREAL	Converts DINT to LREAL (64-bit signed floating-point)
Convert DINT to REAL	DINT_TO_REAL	Converts DINT to REAL (32-bit signed floating-point)

Convert DINT to UINT	DINT_TO_UINT	Converts DINT to UINT (16-bit unsigned integer)
Convert DWORD		
Convert DWORD to DINT	DWORD_TO_DINT	Converts DWORD (32-bit bit string) to DINT (32-bit signed integer)
Convert INT		
Convert INT to BCD4	INT_TO_BCD4	Converts INT (16-bit signed integer) to 4-digit Binary-Coded-Decimal (BCD4)
Convert INT to DINT	INT_TO_DINT	Converts INT to DINT (32-bit signed integer)
Convert INT to REAL	INT_TO_REAL	Converts INT to REAL (32-bit signed floating-point)
Convert INT to UINT	INT_TO_UINT	Converts INT to UINT (16-bit unsigned integer)
Convert INT to WORD	INT_TO_WORD	Converts INT to WORD (16-bit bit string)
Convert LREAL		
Convert LREAL to DINT	LREAL_TO_DINT	Converts LREAL (64-bit signed floating-point) to DINT (32-bit signed integer)
Convert LREAL to REAL	LREAL_TO_REAL	Converts LREAL to REAL (32-bit signed floating-point)
Convert REAL		
Convert REAL to DINT	REAL_TO_DINT	Converts REAL (32-bit signed floating-point) to DINT (32-bit signed integer)
Convert REAL to INT	REAL_TO_INT	Converts REAL to INT (16-bit signed integer)
Convert REAL to LREAL	REAL_TO_LREAL	Converts REAL to LREAL (64-bit signed floating-point)
Convert REAL to UINT	REAL_TO_UINT	Converts REAL to UINT (16-bit unsigned integer)
Convert UINT		
Convert UINT to BCD4	UINT_TO_BCD4	Converts UINT (16-bit unsigned integer) to 4-digit Binary-Coded-Decimal (BCD4)

Convert UINT to DINT	UINT_TO_DINT	Converts UINT to DINT (32-bit signed integer)
Convert UINT to INT	UINT_TO_INT	Converts UINT to INT (16-bit signed integer)
Convert UINT to REAL	UINT_TO_REAL	Converts UINT to REAL (32-bit signed floating-point)
Convert UINT to WORD	UINT_TO_WORD	Converts UINT to WORD (16-bit bit string)
Convert WORD		
Convert WORD to INT	WORD_TO_INT	Converts WORD (16-bit bit string) to INT (16-bit signed integer)
Convert WORD to UINT	WORD_TO_UINT	Converts WORD to UINT (16-bit unsigned integer)
Other		
Truncate	TRUNC_DINT	Truncate. Rounds a REAL (32-bit signed floating-point) value down to a DINT (32-bit signed integer)
	TRUNC_INT	Truncate. Rounds a REAL (32-bit signed floating-point) value down to an INT (16-bit signed integer)

Convert Angles

(PACSystems firmware version 3.50 or later; to process LREALs: 5.50 or later.)



Operation

When the Degrees to Radians (DEG_TO_RAD) or the Radians to Degrees (RAD_TO_DEG) instruction receives data, it performs the appropriate angle conversion on the REAL or LREAL value in input IN and places the result in output Q. DEG_TO_RAD and RAD_TO_DEG pass the converted data to the output Q when they perform without overflow.

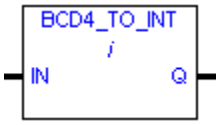
Operands

Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	REAL or LREAL variable or constant. Must be of the same data type as Q.	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The value to convert.
Q	REAL or LREAL variable		The converted value.

Convert BCD4 to INT



Operation

When BCD4_TO_INT receives data, it converts the input 4-digit Binary-Coded-Decimal (BCD4) data into the equivalent single-precision signed integer (INT) value, which it outputs to Q. BCD4_TO_INT does not change the original BCD4 data.

Note: The output data can be used directly as input for another program function.

The instruction places the converted data in output Q, unless the data is out of range.

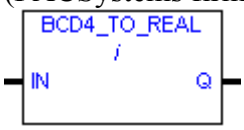
Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	WORD variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The BCD4 value to convert to INT.
Q	INT variable		The INT equivalent value of the original BCD4 value in IN.

Convert BCD4 to REAL

(PACSystems firmware version 3.50 and later.)



Operation

When BCD4_TO_REAL receives data, it converts the 4-digit Binary-Coded-Decimal (BCD4) data into the equivalent floating-point (REAL) value, which it outputs to Q. BCD4_TO_REAL does not change the original BCD4 data.

Note: The output data can be used directly as input for another program function.

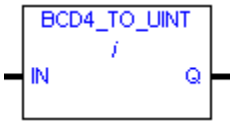
The instruction places the converted data in output Q, unless the data is out of range.

Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	WORD variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The BCD4 value to convert to REAL.
Q	REAL variable		The REAL equivalent value of the original BCD4 value in IN.

Convert BCD4 to UINT



Operation

When BCD4_TO_UINT receives data, it converts the input 4-digit Binary-Coded-Decimal (BCD4) data into the equivalent single-precision unsigned integer (UINT) value, which it outputs to Q. BCD4_TO_UINT does not change the original BCD4 data.

Note: The output data can be used directly as input for another program function.

The instruction passes the converted data to the output Q, unless the resulting data is outside the range 0 to +65,535.

Tip: One use of BCD4_TO_UINT is to convert BCD data from the I/O structure into integer data and store it in memory. This can provide an interface to BCD thumbwheels or external BCD electronics, such as high-speed counters and position encoders.

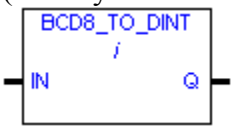
Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	WORD variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The BCD4 value to convert to UINT.
Q	UINT variable		The UINT equivalent value of the original BCD4 value in IN.

Convert BCD8 to DINT

(PACSystems firmware version 3.50 and later.)



Operation

When BCD8_TO_DINT receives data, it converts the input 8-digit Binary-Coded-Decimal (BCD8) data into the equivalent double-precision signed integer (DINT) value, which it outputs to Q. BCD8_TO_DINT does not change the original BCD8 data.

Note: The output data can be used directly as input for another program function.

The instruction passes the converted data to the output Q, unless the data is out of range.

Operands

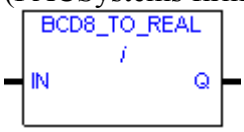
Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types.

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	DWORD variable or constant	data flow, R, P, L, AI, AQ,I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable	The BCD8 value to convert to DINT.
Q	DINT variable	data flow, R, P, L, AI, AQ, I, Q, M, T, G, W, symbolic, I/O variable	The DINT equivalent value of the original BCD8 value in IN.

Convert BCD8 to REAL

(PACSystems firmware version 3.50 and later.)



Operation

When BCD8_TO_REAL receives data, it converts the input 8-digit Binary-Coded-Decimal (BCD8) data into the equivalent floating-point (REAL) value, which it outputs to Q. BCD8_TO_REAL does not change the original BCD8 data.

Note: The output data can be used directly as input for another program function.

The instruction passes the converted data to the output Q, unless the conversion would result in a value that is out of range.

Warning: Converting from BCD8 to REAL may result in the loss of significant digits. This is because a BCD8 is stored in a DWORD, which uses 32 bits to store a value, whereas a REAL (32-bit IEEE floating point number) uses 8 bits to store the exponent and the sign and only 24 bits to store the mantissa.

Operands

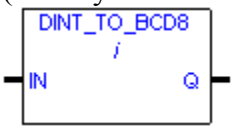
Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types.

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	DWORD variable or constant	data flow, R, P, L, AI, AQ, I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable	The BCD8 value to convert to REAL.
Q	REAL variable	data flow, R, P, L, AI, AQ, I, Q, M, T, G, W, symbolic, I/O variable	The REAL equivalent value of the original BCD8 value in IN.

Convert DINT to BCD8

(PACSystems firmware version 3.50 and later.)



Operation

When DINT_TO_BCD8 receives data, it converts the input signed double-precision integer (DINT) data into the equivalent 8-digit Binary-Coded-Decimal (BCD) values, which it outputs to Q. DINT_TO_BCD8 does not change the original DINT data.

Note: The output data can be used directly as input for another program function.

The instruction passes the converted data to the output Q, unless the conversion would result in a value that is outside the range 0 to 99,999,999.

Operands

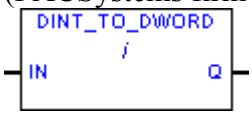
Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types.

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	DINT variable or constant	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The DINT value to convert to BCD8.
Q	DWORD variable	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, SA, SB, SC, symbolic, I/O variable	The BCD8 equivalent value of the original DINT value in IN.

Convert DINT to DWORD

(PACSystems firmware version 3.50 and later.)



Operation

When DINT_TO_DWORD receives data, it converts the input double-precision signed integer (DINT) data into the equivalent DWORD (32-bit bit string) value, which it outputs to Q. DINT_TO_DWORD does not change the original DINT data.

Note: The output data can be used directly as input for another FBD function.

The instruction passes the converted data to the output Q, unless the data is out of range.

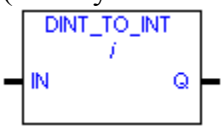
Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	DINT variable or constant	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The DINT value to convert to DWORD.
Q	DWORD variable	data flow, R, P, L, AI, AQ, I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable	The DWORD equivalent value of the original DINT value in IN.

Convert DINT to INT

(PACSystems firmware version 3.50 and later.)



Operation

When DINT_TO_INT receives data, if the input signed double-precision integer (DINT) data is within the range of the single-precision signed integer (INT) data type, DINT_TO_INT converts the input DINT value to the equivalent INT value and outputs it to Q.

If the input value overflows or underflows the INT range, respectively the maximum or minimum value of the INT range is output to Q. The only way in FBD logic to know whether a maximum or minimum value is genuine or resulted from an out-of-range condition is to set up bounds checking logic before using DINT_TO_INT.

DINT_TO_INT does not change the original DINT data.

The output data can be used directly as input for another program function.

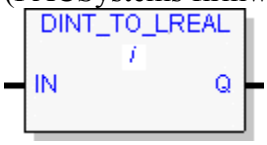
Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	DINT variable or constant	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The DINT value to convert to INT.
Q	INT variable		The INT equivalent value of the original DINT value in IN.

Convert DINT to LREAL

(PACSystems firmware version 5.50 and later.)



Operation

When DINT_TO_LREAL receives data, it converts the input signed double-precision integer (DINT) data into the equivalent double-precision floating-point (LREAL) value, which it outputs to Q. DINT_TO_LREAL does not change the original DINT data.

Note: The output data can be used directly as input for another program function.

The instruction passes the converted data to the output Q.

Operands

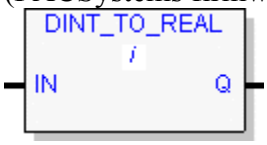
Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	DINT variable or constant	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The DINT value to convert to LREAL.
Q	LREAL variable		The LREAL equivalent value of the original DINT value in IN.

Convert DINT to REAL

(PACSystems firmware version 3.50 and later.)



Operation

When DINT_TO_REAL receives data, it converts the input signed double-precision integer (DINT) data into the equivalent floating-point (REAL) value, which it outputs to Q. DINT_TO_REAL does not change the original DINT data.

Note: The output data can be used directly as input for another program function.

The instruction passes the converted data to the output Q, unless the conversion would result in a value that is out of range.

Warning: Converting from DINT to REAL may result in the loss of significant digits for numbers with more than 7 significant base-10 digits. This is because a DINT uses 32 bits to store a value, which is the equivalent of up to 10 significant base-10 digits, whereas a REAL (32-bit IEEE floating point number) uses 8 bits to store the exponent and the sign and only 24 bits to store the mantissa, which is the equivalent of 7 or 8 significant base-10 digits. When the REAL result is displayed as a base-10 number, it may have up to 10 digits, but these are converted from the rounded 24-bit mantissa, so that the last 2 or 3 digits may be inaccurate.

Operands

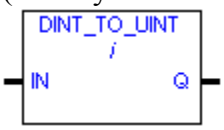
Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types.

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	DINT variable or constant	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The DINT value to convert to REAL.
Q	REAL variable		The REAL equivalent value of the original DINT value in IN.

Convert DINT to UINT

(PACSystems firmware version 3.50 and later.)



Operation

When DINT_TO_UINT receives data, if the input signed double-precision integer (DINT) data is within the range of the single-precision unsigned integer (UINT) data type, DINT_TO_UINT converts the input DINT value to the equivalent UINT value and outputs it to Q.

If the input value overflows or underflows the UINT range, respectively the maximum or minimum value of the UINT range is output to Q. The only way in FBD logic to know whether a maximum or minimum value is genuine or resulted from an out-of-range condition is to set up bounds checking logic before using DINT_TO_UINT.

DINT_TO_UINT does not change the original DINT data.

The output data can be used directly as input for another program function.

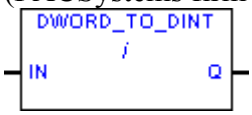
Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	DINT variable or constant	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The DINT value to convert to UINT.
Q	UINT variable		The UINT equivalent value of the original DINT value in IN.

Convert DWORD to DINT

(PACSystems firmware version 3.50 and later.)



Operation

When DWORD_TO_DINT receives data, it converts the input 32-bit bit string (DWORD) data into the equivalent double-precision signed integer (DINT) value, which it outputs to Q. DWORD_TO_DINT does not change the original DWORD data.

Note: The output data can be used directly as input for another FBD function.

The instruction passes the converted data to the output Q, unless the data is out of range.

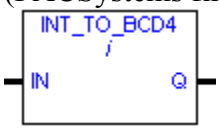
Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	DWORD variable or constant	data flow, R, P, L, AI, AQ, I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable	The DWORD value to convert to DINT.
Q	DINT variable	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The DINT equivalent value of the original DWORD value in IN.

Convert INT to BCD4

(PACSystems firmware version 3.50 and later.)



Operation

When INT_TO_BCD4 receives data, it converts the input signed single-precision integer (INT) data into the equivalent 4-digit Binary-Coded-Decimal (BCD) values, which it outputs to Q. INT_TO_BCD4 does not change the original INT data.

Note: The output data can be used directly as input for another program function.

The instruction passes the converted data to the output Q, unless the conversion would result in a value that is outside the range 0 through 9,999.

Tip: Data can be converted to BCD format to drive BCD-encoded LED displays or presets to external devices such as high-speed counters.

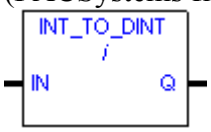
Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	INT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The INT value to convert to BCD4.
Q	BCD4 variable		The BCD4 equivalent value of the original INT value in IN.

Convert INT to DINT

(PACSystems firmware version 3.50 and later.)



Operation

When INT_TO_DINT receives data, it converts the input single-precision signed integer (INT) data into the equivalent double-precision signed integer (DINT) value, which it outputs to Q. INT_TO_DINT does not change the original INT data.

Note: The output data can be used directly as input for another program function.

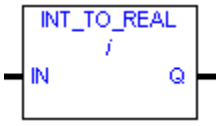
The instruction returns the converted data to the output Q, unless the data is out of range.

Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	INT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The INT value to convert to DINT.
Q	DINT variable	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The DINT equivalent value of the original INT value in IN.

Convert INT to REAL



Operation

When INT_TO_REAL receives data, it converts the input single-precision integer (INT) data into the equivalent floating-point (REAL) value, which it outputs to Q. INT_TO_REAL does not change the original INT data.

Note: The output data can be used directly as input for another program function.

The instruction passes the converted data to the output Q, unless the conversion would result in a value that is out of range.

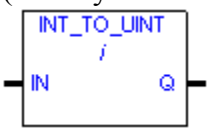
Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	INT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The INT value to convert to REAL.
Q	REAL variable		The REAL equivalent value of the original INT value in IN.

Convert INT to UINT

(PACSystems firmware version 3.50 and later.)



Operation

When INT_TO_UINT receives data, if the input signed single-precision integer (INT]) data is within the range of the unsigned single-precision integer (UINT) data type, INT_TO_UINT converts the input INT value to the equivalent UINT value and outputs it to Q.

If the input value underflows the UINT range, the minimum value of the UINT range is output to Q. The only way in FBD logic to know whether a minimum value is genuine or resulted from an out-of-range condition is to set up bounds checking logic before using INT_TO_UINT.

INT_TO_UINT does not change the original INT data.

The output data can be used directly as input for another program function.

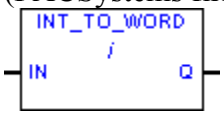
Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	INT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The INT value to convert to UINT.
Q	UINT variable		The UINT equivalent value of the original DINT value in IN.

Convert INT to WORD

(PACSystems firmware version 3.50 and later.)



Operation

When INT_TO_WORD receives data, it converts the input signed single-precision integer (INT) data into the equivalent 16-bit bit string (WORD) value, which it outputs to Q. INT_TO_WORD does not change the original INT data.

Note: The output data can be used directly as input for another FBD function.

The instruction passes the converted data to the output Q, unless the data is out of range.

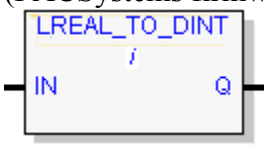
Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	INT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The INT value to convert to WORD.
Q	WORD variable	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The WORD equivalent value of the original INT value in IN.

Convert LREAL to DINT

(PACSystems firmware version 5.50 and later.)



Operation

When LREAL_TO_DINT receives data, if the input double-precision signed floating-point (LREAL) data is within the range of the double-precision signed integer (DINT) data type, LREAL_TO_DINT rounds the input LREAL value to the nearest DINT value and outputs it to Q.
If the input value overflows or underflows the DINT range, respectively the maximum or minimum value of the DINT range is output to Q. The only way in FBD logic to know whether a maximum or minimum value is genuine or resulted from an out-of-range condition is to set up bounds checking logic before using LREAL_TO_DINT.
LREAL_TO_DINT does not change the original LREAL data.
The output data can be used directly as input for another program function.

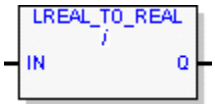
Operands

Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types.

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	LREAL variable or constant	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The LREAL value to convert to DINT.
Q	DINT variable		The DINT equivalent value of the original LREAL value in IN.

Convert LREAL to REAL



Operation

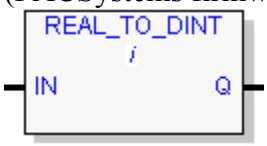
When LREAL_TO_REAL receives data, if the input double-precision signed floating-point (LREAL) data is within the range of the single-precision signed floating-point (REAL) data type, LREAL_TO_REAL converts the input LREAL value to the equivalent REAL value and outputs it to Q.
If the input value overflows or underflows the REAL range, respectively the maximum or minimum value of the REAL range is output to Q. The only way in FBD logic to know whether a maximum or minimum value is genuine or resulted from an out-of-range condition is to set up bounds checking logic before using LREAL_TO_REAL.
LREAL_TO_REAL does not change the original LREAL data.
The output data can be used directly as input for another program function.

Operands

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	LREAL	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The LREAL value to convert to REAL.
Q	REAL variable		The REAL equivalent value of the original LREAL value in IN.

Convert REAL to DINT

(PACSystems firmware version 3.50 and later.)



Operation

When REAL_TO_DINT receives data, if the input single-precision signed floating-point (REAL) data is within the range of the double-precision signed integer (DINT) data type, REAL_TO_DINT rounds the input REAL value to the nearest DINT value and outputs it to Q.

If the input value overflows or underflows the DINT range, respectively the maximum or minimum value of the DINT range is output to Q. The only way in FBD logic to know whether a maximum or minimum value is genuine or resulted from an out-of-range condition is to set up bounds checking logic before using REAL_TO_DINT.

REAL_TO_DINT does not change the original REAL data.

The output data can be used directly as input for another program function.

Tip: To truncate a REAL value and express the result as a DINT, that is, to remove the fractional part of the REAL number and express the remaining integer value as a DINT, use TRUNC_DINT.

Operands

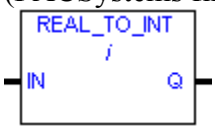
Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types.

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	REAL variable or constant	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The REAL value to convert to DINT.
Q	DINT variable		The DINT equivalent value of the original REAL value in IN.

Convert REAL to INT

(PACSystems firmware version 3.50 and later.)



Operation

When REAL_TO_INT receives data, if the input single-precision signed floating-point (REAL) data is within the range of the single-precision signed integer (INT) data type, REAL_TO_INT rounds the input REAL value to the nearest INT value and outputs it to Q.

If the input value overflows or underflows the INT range, respectively the maximum or minimum value of the INT range is output to Q. The only way in FBD logic to know whether a maximum or minimum value is genuine or resulted from an out-of-range condition is to set up bounds checking logic before using REAL_TO_INT.

REAL_TO_INT does not change the original REAL data.

The output data can be used directly as input for another program function.

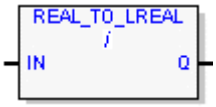
Tip: To truncate a REAL value and express the result as an INT, that is, to remove the fractional part of the REAL number and express the remaining integer value as an INT, use TRUNC_INT.

Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	REAL variable or constant	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The REAL value to convert to INT.
Q	INT variable		The INT equivalent value of the original REAL value in IN.

Convert REAL to LREAL



Operation

When REAL_TO_LREAL receives data, it converts the input single-precision floating-point (REAL) data to a double-precision floating-point (LREAL) value, which it outputs to Q. REAL_TO_LREAL does not change the original REAL data.

Note: The output data can be used directly as input for another program function.

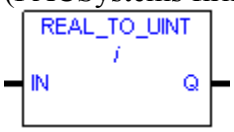
The instruction passes the converted data to the output Q, unless the data is NaN (Not a Number).

Operands

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	REAL variable or constant	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The REAL value to convert to LREAL.
Q	LREAL variable		The LREAL equivalent value of the original REAL value in IN.

Convert REAL to UINT

(PACSystems firmware version 3.50 and later.)



Operation

When REAL_TO_UINT receives data, if the input single-precision signed floating-point (REAL) data is within the range of the single-precision unsigned integer (UINT) data type, REAL_TO_UINT rounds the input REAL value to the nearest UINT value and outputs it to Q.

If the input value overflows or underflows the UINT range, respectively the maximum or minimum value of the UINT range is output to Q. The only way in FBD logic to know whether a maximum or minimum value is genuine or resulted from an out-of-range condition is to set up bounds checking logic before using REAL_TO_UINT.

REAL_TO_UINT does not change the original REAL data.

The output data can be used directly as input for another program function.

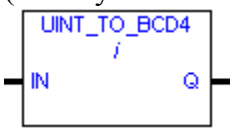
Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	REAL variable or constant	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The REAL value to convert to UINT.
Q	UINT variable		The UINT equivalent value of the original REAL value in IN.

Convert UINT to BCD4

(PACSystems firmware version 3.50 and later.)



Operation

When UINT_TO_BCD4 receives data, it converts the input unsigned single-precision integer (UINT) data into the equivalent 4-digit Binary-Coded-Decimal (BCD) values, which it outputs to Q. UINT_TO_BCD4 does not change the original UINT data.

Note: The output data can be used directly as input for another program function.

The instruction passes the converted data to the output Q, unless the conversion would result in a value that is outside the range 0 to 9,999.

Tip: Data can be converted to BCD format to drive BCD-encoded LED displays or presets to external devices such as high-speed counters.

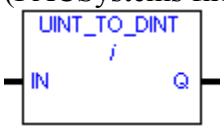
Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	UINT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The UINT value to convert to BCD4
Q	BCD4 variable		The BCD4 equivalent value of the original UINT value in IN

Convert UINT to DINT

(PACSystems firmware version 3.50 and later.)



Operation

When UINT_TO_DINT receives data, it converts the input unsigned single-precision integer (UINT) data into the equivalent signed double-precision integer (DINT) value, which it outputs to Q. UINT_TO_DINT does not change the original UINT data.

Note: The output data can be used directly as input for another program function.

The instruction passes the converted data to the output Q, unless the data is out of range.

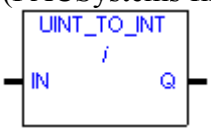
Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	UINT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The UINT value to convert to DINT.
Q	DINT variable		The DINT equivalent value of the original UINT value in IN.

Convert UINT to INT

(PACSystems firmware version 3.50 and later.)



Operation

When UINT_TO_INT receives data, if the input unsigned single-precision integer (UINT) data is within the range of the single-precision signed integer (INT) data type, UINT_TO_INT converts the input UINT value to the equivalent INT value and outputs it to Q.

If the input value overflows the INT range, the maximum value of the INT range is output to Q. The only way in FBD logic to know whether a maximum value is genuine or resulted from an out-of-range condition is to set up bounds checking logic before using UINT_TO_INT.

UINT_TO_INT does not change the original UINT data.

The output data can be used directly as input for another program function.

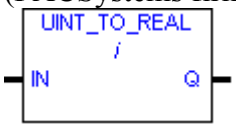
Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	UINT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The UINT value to convert to INT
Q	INT variable		The INT equivalent value of the original UINT value in IN

Convert UINT to REAL

(PACSystems firmware version 3.50 and later.)



Operation

When UINT_TO_REAL receives data, it converts the input unsigned single-precision integer (UINT) data into the equivalent floating-point (REAL) value, which it outputs to Q. UINT_TO_REAL does not change the original UINT data.

Note: The output data can be used directly as input for another program function.

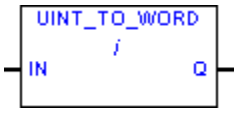
The instruction passes the converted data to the output Q, unless the conversion would result in a value that is out of range.

Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	UINT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The UINT value to convert to REAL
Q	REAL variable		The REAL equivalent value of the original UINT value in IN

Convert UINT to WORD



Operation

When UINT_TO_WORD receives data, it converts the input unsigned single-precision integer (UINT) data into the equivalent 16-bit bit string (WORD) value, which it outputs to Q. UINT_TO_WORD does not change the original UINT data.

Note: The output data can be used directly as input for another FBD function.

The instruction passes the converted data to the output Q, unless the resulting data is outside the range.

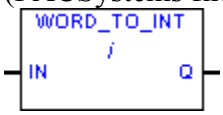
Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	UINT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The UINT value to convert to WORD.
Q	WORD variable	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The WORD equivalent value of the original UINT value in IN.

Convert WORD to INT

(PACSystems firmware version 3.50 and later.)



Operation

When WORD_TO_INT receives data, it converts the input 16-bit bit string (WORD) data into the equivalent single-precision signed integer (INT) value, which it outputs to Q. WORD_TO_INT does not change the original WORD data.

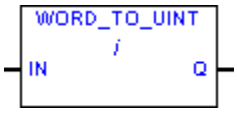
Note: The output data can be used directly as input for another FBD function.

Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	WORD variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The WORD value to convert to INT.
Q	INT variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The INT equivalent value of the original WORD value in IN.

Convert WORD to UINT



Operation

When WORD_TO_UINT receives data, it converts the input 16-bit bit string (WORD) data into the equivalent single-precision unsigned integer (UINT) value, which it outputs to Q. WORD_TO_UINT does not change the original WORD data.

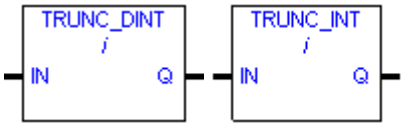
Note: The output data can be used directly as input for another FBD function.

Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	WORD variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The WORD value to convert to UINT
Q	UINT variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The UINT equivalent value of the original WORD value in IN

Truncate



Operation

When data is received, the Truncate functions TRUNC_DINT and TRUNC_INT round a floating-point (REAL) value down respectively to the nearest signed double-precision signed integer (DINT) or signed single-precision integer (INT) value. TRUNC_DINT and TRUNC_INT output the converted value to Q. The original data is not changed.

Note: The output data can be used directly as input for another program function.

TRUNC_DINT and TRUNC_INT pass the converted data to the output Q, unless the specified conversion would result in a value that is out of range.

Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
i	The solve order for the instruction.		
IN	REAL variable or constant	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The REAL value whose copy is to be converted and truncated. The original is left intact.
Q	DINT variable	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The truncated value of the original REAL value in IN.
	INT variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	

Index

Addendum

Communication

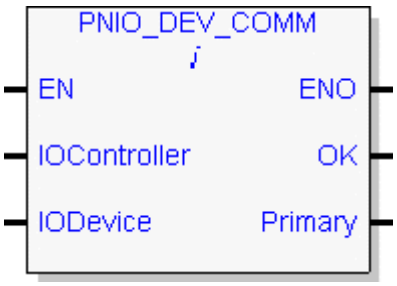
(For PACSystems firmware version 7.00 and later.)

Instruction	*Mnemonics*	*Description*
PNIO_DEV_COMM	PNIO_DEV_COMM	Verifies whether a Profinet Controller (PNC) is communicating with a Profinet IO-device and, in Hot Standby redundancy systems, whether the PNC is the device's primary IO-Controller.

PNIO_DEV_COMM

LD

FBD



ST

Formal convention:

```
PNIO_DEV_COMM(IOController := [input], IODevice := [input], OK => [output], Primary => [output]);
```

Tip: To spare yourself some typing, drag the instruction from the Toolchest's LD Instructions or FBD Instructions drawer to your ST logic. All the input and output names are inserted into your ST logic. You need only to supply the variable names or constants and remove the optional inputs or outputs you don't want to use.

Operation

PNIO_DEV_COMM verifies whether the Profinet Controller (PNC) identified by input IOController and the Profinet IO-device identified by input IODevice are communicating with one another. If so, output OK is set to On; otherwise, it is set to Off.

When PNIO_DEV_COMM is used in a Hot Standby (HSB) redundancy system, if OK is set to On, PNIO_DEV_COMM verifies whether the PNC is the device's Primary IO-Controller. If so, the Primary output is set to On; otherwise, the output is set to Off.

When PNIO_DEV_COMM is not used in an HSB redundancy system, Primary is set to the same value as OK.

Operands

Input Operands

Operand	*Data Type*	*Memory Area*	*Description*

i	(FBD only.) Solve order for the instruction.		
Power flow (LD)			When On, PNIO_DEV_COMM executes. When Off, PNIO_DEV_COMM does not execute.
EN (FBD)	BOOL variable or BOOL system variable	I, Q, M, T, S, SA, SB, SC, G, discrete symbolic, discrete I/O variable	Enable input. When On, PNIO_DEV_COMM solves. When Off, PNIO_DEV_COMM does not solve.
	Bit reference in non-BOOL variable	R, P, L, AI, AQ, W, I, Q, M, T, G, non-discrete symbolic, non-discrete I/O variable	
IOController	PNIO_CONTROLLER_REF variable	symbolic, I/O variable. LD and FBD also support data flow.	Variables used to identify respectively the Profinet Controller and Profinet device whose intercommunication status is to be reported
IODevice	PNIO_DEVICE_REF variable	symbolic, I/O variable. LD and FBD also support data flow.	

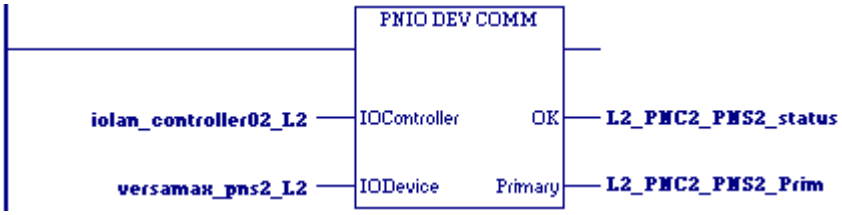
Output Operands

Operand	*Data Type*	*Memory Area*	*Description*
Power flow (LD; optional.)			There is power flow when PNIO_DEV_COMM executes successfully.
ENO (FBD; optional.)	BOOL variable	data flow, I, Q, M, T, S, SA, SB, SC, G, discrete symbolic, discrete I/O variable	ENO is set to On if EN is On.
	Bit reference in non-BOOL variable	R, P, L, AI, AQ, W, I, Q, M, T, G, non-discrete symbolic, non-discrete I/O variable	
OK	▪ LD: Power flow		(Optional.) Set to On if the specified IO Controller and IO device are currently communicating.

	▪ FBD and ST: BOOL variable		
Primary	▪ LD: Power flow ▪ FBD and ST: BOOL variable		(Optional.) Set to On if the PNC is the Primary IO Controller of the device in an HSB redundancy system. Requires communications with the device (OK is set to On). In a simplex system, Primary is set to On if OK is set to On.

Example

PNIO_DEV_COMM is used in an HSB redundancy system. It verifies whether the Profinet Controller (PNC) identified by the variable iolan_controller02_L2 and the Profinet device identified by the variable versamax_pns2_L2 are currently communicating with one another. If so, the BOOL variable L2_PNC2_PNS2_Prim is set to On. PNIO_DEV_COMM also verifies whether the identified PNC is the primary IO-Controller of the device; if so, the BOOL variable L2_PNC2_PNS2_status is set to On.



CPU Support

PNIO_DEV_COMM is supported for PACSystems CPUs with firmware version 7.00 or later.