Micro800 Programmable Controllers General Instructions

Catalog Numbers 2080-L50E, 2080-L70E, 2080-LC10, 2080-LC20, 2080-LC30, 2080-LC50, 2080-LC70

Rockwell Automation Publication 2080-RM001K-EN-E - March 2022
Supersedes Publication 2080-RM001J-EN-E - March 2021

Reference Manual

Original Instructions

Important User Information

Read this document and the documents listed in the additional resources section about installation, configuration, and operation of this equipment before you install, configure, operate, or maintain this product. Users are required to familiarize themselves with installation and wiring instructions in addition to requirements of all applicable codes, laws, and standards.

Activities including installation, adjustments, putting into service, use, assembly, disassembly, and maintenance are required to be carried out by suitably trained personnel in accordance with applicable code of practice.

If this equipment is used in a manner not specified by the manufacturer, the protection provided by the equipment may be impaired.

In no event will Rockwell Automation, Inc. be responsible or liable for indirect or consequential damages resulting from the use or application of this equipment.

The examples and diagrams in this manual are included solely for illustrative purposes. Because of the many variables and requirements associated with any particular installation, Rockwell Automation, Inc. cannot assume responsibility or liability for actual use based on the examples and diagrams.

No patent liability is assumed by Rockwell Automation, Inc. with respect to use of information, circuits, equipment, or software described in this manual.

Reproduction of the contents of this manual, in whole or in part, without written permission of Rockwell Automation, Inc., is prohibited.

Throughout this manual, when necessary, we use notes to make you aware of safety considerations.

WARNING: Identifies information about practices or circumstances that can cause an explosion in a hazardous environment, which may lead to personal injury or death, property damage, or economic loss.

ATTENTION: Identifies information about practices or circumstances that can lead to personal injury or death, property damage, or economic loss. Attentions help you identify a hazard, avoid a hazard, and recognize the consequence.

IMPORTANT Identifies information that is critical for successful application and understanding of the product.

Labels may also be on or inside the equipment to provide specific precautions.

SHOCK HAZARD: Labels may be on or inside the equipment, for example, a drive or motor, to alert people that dangerous voltage may be present.

BURN HAZARD: Labels may be on or inside the equipment, for example, a drive or motor, to alert people that surfaces may reach dangerous temperatures.

ARC FLASH HAZARD: Labels may be on or inside the equipment, for example, a motor control center, to alert people to potential Arc Flash. Arc Flash will cause severe injury or death. Wear proper Personal Protective Equipment (PPE). Follow ALL Regulatory requirements for safe work practices and for Personal Protective Equipment (PPE).

Rockwell Automation recognizes that some of the terms that are currently used in our industry and in this publication are not in alignment with the movement toward inclusive language in technology. We are proactively collaborating with industry peers to find alternatives to such terms and making changes to our products and content. Please excuse the use of such terms in our content while we implement these changes.

Preface

Chapter 1

Find information about instructions and ladder elements

Chapter 2

Ladder Diagram (LD) language

Chapter 3

Ladder Diagram (LD) elements

Chapter 4
Alarm instruction

Chapter 5
Arithmetic instructions

Chapter 6
ASCII serial port instructions

Chapter 7

Binary instructions

Chapter 8

Boolean instructions

Chapter 9

Communication instructions

Chapter 10

Compare instructions

Chapter 11

Counter instructions

Chapter 12

Data conversion instructions

Chapter 13

Data manipulation instructions

Chapter 14

High-Speed Counter (HSC) instructions

Chapter 15

HSCE instructions

Chapter 16

Input/Output instructions

Chapter 17

Interrupt instructions

Motion control instructions

Chapter 18

Process control instructions

Chapter 19

Chapter 20

Program control instruction

Chapter 21

Proportional Integral Derivative (PID) instruction

Chapter 22

Real Time Clock (RTC) instructions

Chapter 23

Socket instructions

Chapter 24

String manipulation instructions

Chapter 25

Timer instructions

Preface

In this manual

This guide provides reference information about the instruction set available for developing programs for use in Micro800® control systems. The instruction set includes Structured Text (ST), Ladder Diagram (LD) Function Block Diagram (FBD) programming language support. Additionally, the ladder elements supported in Connected Components Workbench™ development environment are defined.

Supported controllers

Connected Components Workbench includes configuration support for the following Micro800 controllers.

- 2080-LC10-12AWA
- 2080-LC10-12DWD
- 2080-LC10-12QBB
- 2080-LC10-12QWB
- 2080-LC20-20AWB
- 2080-LC20-20QBB
- 2080-LC20-20QWB
- 2080-LC30-10QVB
- 2080-LC30-10QWB
- 2080-LC30-16AWB
- 2080-LC30-16QVB
- 2080-LC30-16QWB
- 2080-LC30-24QBB
- 2080-LC30-24QVB
- 2080-LC30-24QWB
- 2080-LC30-48AWB
- 2080-LC30-48QBB
- 2080-LC30-48QVB
- 2080-LC30-48QWB
- 2080-LC50-24AWB
- 2080-LC50-24QBB
- 2080-LC50-24QVB
- 2080-LC50-24QWB
- 2080-LC50-48AWB
- 2080-LC50-48QBB
- 2080-LC50-48QVB
- 2080-LC50-48QWB
- 2080-LC50-48QWB - SIM
- 2080-L50E-24AWB
- 2080-L50E-24QBB
- 2080-L50E-24QVB
- 2080-L50E-24QWB
- 2080-L50E-48AWB
- 2080-L50E-48QBB
- 2080-L50E-48QVB
- 2080-L50E-48QWB
- 2080-L70E-24AWB
- 2080-L70E-24QBB
- 2080-L70E-24QBBN
- 2080-L70E-24QWB
- 2080-L70E-24QWBN
- 2080-LC70-24AWB
- 2080-LC70-24QBB
- 2080-LC70-24QWB

Additional resources

These documents contain additional information concerning related Rockwell Automation® products.

Resource	Description

Resource	Description
Industrial Automation Wiring and Grounding Guidelines, publication 1770-4.1 available at http://literature.rockwellautomation.com/idc/groups/literature/documents/in/1770-in041_-en-p.pdf.	Provides general guidelines for installing a Rockwell Automation industrial system.
Product Certifications website, http://www.ab.com	Provides declarations of conformity, certificates, and other certification details.
MicroLogix Controllers to Micro800 Controllers Migration Guide, available at https://literature.rockwellautomation.com/idc/groups/literature/documents/rm/2080-rm002_-en-e.pdf	Shows how to use the software tools to select a suitable Micro800 controller, and also how to convert your MicroLogix programs to work with the Micro800 controller.
Micro800 Controllers Starter Pack Quick Start, available at https://literature.rockwellautomation.com/idc/groups/literature/documents/qs/2080-qs004_-en-e.pdf	Shows how to use a Micro800 controller with a PanelView 800 terminal.
Configuring Micro800 Controllers on FactoryTalk Gateway, available at https://literature.rockwellautomation.com/idc/groups/literature/documents/qs/2080-qs005_-en-e.pdf	Provides quick start instructions for configuring a Micro800 controller on FactoryTalk Linx Gateway.

You can view or download publications at http://www.rockwellautomation.com/literature. To order paper copies of technical documentation, contact your local Rockwell Automation distributor or sales representative.

Legal Notices

Rockwell Automation publishes legal notices, such as privacy policies, license agreements, trademark disclosures, and other terms and conditions on the Legal Notices page of the Rockwell Automation website.

End User License Agreement (EULA)

You can view the Rockwell Automation End User License Agreement (EULA) by opening the license.rtf file located in your product's install folder on your hard drive.

The default location of this file is:

C:\Program Files (x86)\Common Files\Rockwell\license.rtf.

Open Source Software Licenses

The software included in this product contains copyrighted software that is licensed under one or more open source licenses.

You can view a full list of all open source software used in this product and their corresponding licenses by opening the oss_license.txt file located in your product's OPENSOURCE folder on your hard drive. This file is divided into these sections:

- Components
  Includes the name of the open source component, its version number, and the type of license.
- Copyright Text
  Includes the name of the open source component, its version number, and the copyright declaration.
- Licenses
  Includes the name of the license, the list of open source components citing the license, and the terms of the license.

The default location of this file is:

> C:\Program Files (x86)\Common Files\Rockwell\Help\<*product name*>\Release Notes\OPENSOURCE\oss_licenses.txt.

You may obtain Corresponding Source code for open source packages included in this product from their respective project web site(s). Alternatively, you may obtain complete Corresponding Source code by contacting Rockwell Automation via the **Contact** form on the Rockwell Automation website: http://www.rockwellautomation.com/global/about-us/contact/contact.page. Please include "Open Source" as part of the request text.

Commercial Software Licenses

The following table lists the commercially licensed software components in Connected Components Workbench.

Component	Copyright
Actipro v17.1.0651	Copyright © 2002-2013 Actipro Software LLC
ag-grid v21.0.1	Copyright (c) 2015-2020 AG GRID LTD

Find information about instructions and ladder elements

Connected Components Workbench includes a comprehensive instruction set with structures and arrays, development environments for ladder logic, structured text, function block diagram, and user-defined function block programs.

Additionally, Connected Components Workbench includes user-interface configuration tools for Micro800 controllers, PowerFlex® drives, a Safety Relay device, PanelView™ Component graphic terminals, and serial and network connectivity options.

For information about a specific instruction, including a description, parameter details, and language examples, locate the instruction from the table of contents, or from the following reference topics.

- Instruction reference in alphabetical order on page 22

For a description of the ladder elements used in Connected Components Workbench, see the following section:

- Ladder Diagram (LD) language reference on page 29

Instruction blocks

The Connected Components Workbench instruction set includes IEC 61131-3 compliant instruction blocks for Micro800 controllers. Instruction blocks collectively include operators on page 19, functions on page 19 and function blocks on page 21.

Operators

The Connected Components Workbench instruction set includes IEC 61131-3 compliant instruction blocks for Micro800 controllers; operators are included in the Connected Components Workbench instruction set.

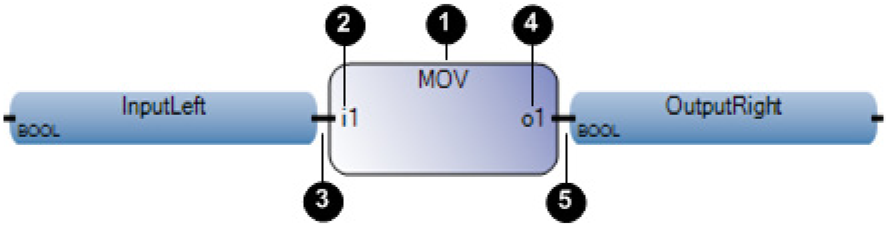
An operator is a basic logical operation such as arithmetic, boolean, comparator, or data conversion.

Functions

Functions have one or more input parameters and one output parameter.

Instruction block format

An instruction block is represented by a single rectangle, and has a fixed number of input connection points and output connection points. An elementary instruction block performs a single function.

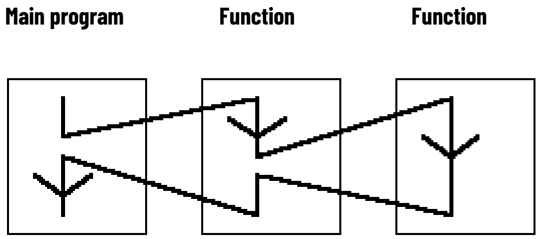


Item No.	Item	Description
1	Block name	The name of the function to be performed by the instruction block is written inside its rectangle shape (at the top).
2	Input	Each input of an instruction block is labeled and has a defined type.
3	Input connection	Inputs are connected on the left border.
4	Output	Each output of an instruction block is labeled and has a defined type.
5	Output connection	Outputs are connected on the right border.

Calling a function

Connected Components Workbench^{TM} does not support recursive function calls. When a function of the Functions section is called by itself or one of its called functions, a run-time error occurs. Furthermore, functions do not store the local values of their local variables. Since functions are not instantiated, they cannot call function blocks.

- A function can be called by a program, by a function, or by a function block.
- Any program of any section can call one or more functions. A function can have local variables.
- A function has no instance meaning local data is not stored and so is usually lost from one call to the other.
- Because the execution of a function is driven by its parent program, the execution of the parent program is suspended until the function ends.



Defining function and parameter names

The interface of a function must be explicitly defined with a type and a unique name for each of its calling (input) parameters or return (output) parameters.

A function has one return parameter. The value of a return parameter for a function block is different for each programming language (FBD, LD, ST).

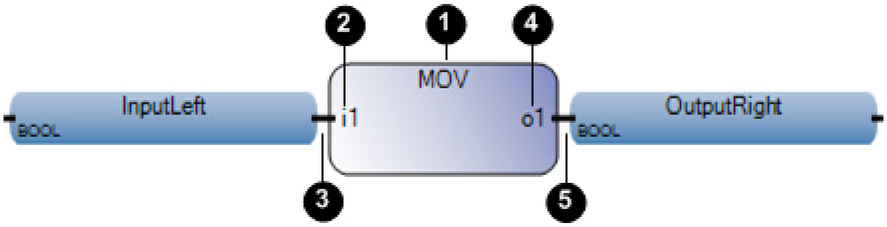
Function names and function parameter names can use up to 128 characters. Function parameter names can begin with a letter or an underscore followed by letters, numbers, and single underscores.

Function blocks

A function block is an instruction block that has input and output parameters and works on internal data (parameters). It can be written in Structured Text, Ladder Diagram, or Function Block Diagram languages.

Instruction block format

An instruction block is represented by a single rectangle, and has a fixed number of input connection points and output connection points. An elementary instruction block performs a single function.



Item No.	Item	Description
❶	Block name	The name of the function to be performed by the instruction block is written inside its rectangle shape (at the top).
❷	Input	Each input of an instruction block is labeled and has a defined type.
❸	Input connection	Inputs are connected on the left border.
❹	Output	Each output of an instruction block is labeled and has a defined type.
❺	Output connection	Outputs are connected on the right border.

Calling a function block

When a function block is called in a program, an instance of the block is actually called. The instance uses the same code, but the input and output parameters are instantiated, which means local variables are copied for each instance of the function block. The values of the variables of a function block instance are stored from one cycle to the other.

A function block can be called by a program, or by another function block. They cannot be called by functions because functions are not instantiated.

Defining function block and parameter names

The interface of a function block must be explicitly defined with a type and a unique name for each of its calling (input) parameters or return (output) parameters. Function blocks can have more than one output parameter. The value of a return parameter for a function block is different for each programming language (FBD, LD, ST).

Function block names and function block parameter names can use up to 128 characters. Function block parameter names can begin with a letter or an underscore followed by letters, numbers, and single underscores.

Instruction set in alphabetical order

The following table lists the Micro800 controller instructions available in Connected Components Workbench and their mapped instructions in Logix theme in alphabetical order.

Instruction	Mapped Instruction (Logix Theme)	Category	Type	Description
- on page 97	SUB	Arithmetic	Operator	Subtracts one Integer, Real or Time value from another Integer, Real or Time value.
* on page 86	MUL	Arithmetic	Operator	Multiplies two or more Integer or Real values.
/ on page 78	DIV	Arithmetic	Operator	Division of two Integer or Real values.
+ on page 68	ADD	Arithmetic	Operator	Adds two or more Integer, Real, Time, or String values.
< on page 225	LES	Compare	Operator	Compares Integer, Real, Time, Date, and String input values to determine whether the first is less than the second.
<= on page 227	LEQ	Compare	Operator	Compares Integer, Real, Time, Date, and String input values to determine whether the first is less than or equal to the second.
<> on page 228	NEQ	Compare	Operator	Compares Integer, Real, Time, Date, and String input values to determine whether the first is not equal to the second.
= on page 219	EQU	Compare	Operator	Tests whether one value is equal to another.
> on page 222	GRT	Compare	Operator	Compares Integer, Real, Time, Date, and String input values to determine whether the first is greater than the second.
>= on page 224	GEQ	Compare	Operator	Compares Integer, Real, Time, Date, and String input values to determine whether the first is greater than or equal to the second.
ABL on page 105	ABL	Communications	Function block	Counts the number of characters in the buffer up to and including end of line character.
ABS on page 63	ABS	Arithmetic	Function	Returns the absolute value of a Real value.
ACB on page 111	ACB	Communications	Function block	Counts the total number of characters in the buffer.
ACL on page 107	ACL	Communications	Function block	Clears the receive and transmit buffers.
ACOS on page 65	ACS	Arithmetic	Function	Calculates the arc-cosine of a Real value.
ACOS_LREAL on page 66	ACOS_LREAL	Arithmetic	Function	Calculates the arc-cosine of a Long Real value.
AFI on page 509	AFI	Program control	Function	Temporarily disables a rung when debugging.
AHL on page 109	AHL	Communications	Function block	Sets or resets modem handshake lines.
AND on page 150	AND	Boolean operations	Operator	Performs a boolean AND operation between two or more values.
AND_MASK on page 125	AND_MASK	Binary operations	Function	Performs a bit to bit AND between two Integer values.
ANY_TO_BOOL on page 239	ANY_TO_BOOL	Data conversion	Function	Converts a non-Boolean value to a Boolean.

Instruction	Mapped Instruction (Logix Theme)	Category	Type	Description
ANY_TO_BYTE on page 240	ANY_TO_BYTE	Data conversion	Function	Converts a value to a Byte.
ANY_TO_DATE on page 241	ANY_TO_DATE	Data conversion	Function	Converts a String, Integer, Real, or Time data type to Date data type.
ANY_TO_DINT on page 243	STOD	Data conversion	Function	Converts a value to a Double Integer.
ANY_TO_DWORD on page 244	ANY_TO_DWORD	Data conversion	Function	Converts a value to a Double Word value.
ANY_TO_INT on page 245	ACI	Data conversion	Function	Converts a value to an Integer.
ANY_TO_LINT on page 246	ANY_TO_LINT	Data conversion	Function	Converts a value to a Long Integer.
ANY_TO_LREAL on page 247	ANY_TO_LREAL	Data conversion	Function	Converts a value to a Long Real.
ANY_TO_LWORD on page 248	ANY_TO_LWORD	Data conversion	Function	Converts a value to a Long Word.
ANY_TO_REAL on page 249	STOR	Data conversion	Function	Converts a value to a Real.
ANY_TO_SINT on page 250	ANY_TO_SINT	Data conversion	Function	Converts a value to a Short Integer.
ANY_TO_STRING on page 250	DTOS	Data conversion	Function	Converts a value to a String.
ANY_TO_TIME on page 252	ANY_TO_TIME	Data conversion	Function	Converts a value to the Time data type.
ANY_TO_UDINT on page 253	ANY_TO_UDINT	Data conversion	Function	Converts a value to an Unsigned Double Integer.
ANY_TO_UINT on page 254	ANY_TO_UINT	Data conversion	Function	Converts a value to an Unsigned Integer.
ANY_TO_ULINT on page 255	ANY_TO_ULINT	Data conversion	Function	Converts a value to an Unsigned Long Integer.
ANY_TO_USINT on page 256	ANY_TO_USINT	Data conversion	Function	Converts a value to an Unsigned Short Integer.
ANY_TO_WORD on page 257	ANY_TO_WORD	Data conversion	Function	Converts a value to a Word.
ARD on page 113	ARD	Communications	Function block	Reads characters from the input buffer and places them into a string.
ARL on page 116	ARL	Communications	Function block	Reads one line of characters from the input buffer and places them into a string.
ASCII on page 603	ASCII	String manipulation	Function	Returns the ASCII code for characters in a string. Character -> ASCII code.
ASIN on page 69	ASN	Arithmetic	Function	Calculates the arcsine of a Real value.
ASIN_LREAL on page 71	ASN_LREAL	Arithmetic	Function	Calculates the arcsine of a Long Real value.
ATAN on page 72	ATN	Arithmetic	Function	Calculates the arctangent of a Real value.
ATAN_LREAL on page 74	ATAN_LREAL	Arithmetic	Function	Calculates the arctangent of a Long Real value.
AVERAGE on page 259	AVE	Data Manipulation	Function block	Calculates a running average over a number of a defined samples.
AWA on page 118	AWA	Communications	Function	Writes a string with two appended (user-configured) characters to an external device.
AWT on page 120	AWT	Communications	Function	Writes characters from a source string to an external device.
BSL on page 126	BSL	Binary operations	Function block	Shifts a bit in an array element to the left.
BSR on page 130	BSR	Binary operations	Function block	Shifts a bit in an array element to the right.
CHAR on page 605	CHAR	String manipulation	Function	Returns a one character string for an ASCII code. ASCII code -> character.
COM_IO_WDOG on page 163	COM_IO_WDOG	Communications	Function block	Monitors communication to the controller.
COP on page 260	COP	Data conversion	Function block	Copies the binary data in the source element to the destination element.
COS on page 75	COS	Arithmetic	Function	Calculates the cosine of a Real value.
COS_LREAL on page 77	COS_LREAL		Function	Calculates the cosine of a Long Real value.
CTD on page 231	CTD	Counter	Function	Counts integers from a given value down to 0, 1 by 1.
CTU on page 233	CTU	Counter	Function	Counts integers from 0 up to a given value, 1 by 1.
CTUD on page 235	CTUD	Counter	Function	Counts integers from 0 up to a given value, 1 by 1, or from a given value down to 0, 1 by 1.
DELETE on page 606	DELETE	String manipulation	Function	Deletes characters from a string.

Instruction	Mapped Instruction (Logix Theme)	Category	Type	Description
DERIVATE on page 463	DERIVATE	Process Control	Function block	Differentiation of a real value over a defined cycle time.
DLG on page 335	DLG	Input/Output	Function Block	Writes variable values from the run-time engine into a Data Logging File on an SD Card.
DOY on page 634	DOY	Time	Function	Turn on an output if the value of the real-time clock is in the range of the Year Time setting.
EXPT on page 80	EXPT	Arithmetic	Function	Calculates the Real value of a base number raised to the power of the Integer exponent.
F_TRIG on page 145	OSF	Boolean operations	Function block	Detects a falling edge of a Boolean variable.
FFL on page 473	FFL	Process Control	Function Block	Loads 8 bit, 16 bit, 32 bit, or 64 bit data into a user-created array called a FIFO stack.
FFU on page 473	FFU	Process Control	Function Block	Unloads 8 bit, 16 bit, 32 bit, or 64 bit data from a user-created array called a FIFO (first in first out) stack in the same order data was loaded using the FFL instruction.
FIND on page 608	FIND	String manipulation	Function	Locates and provides the position of sub-strings within strings.
HSC on page 272	HSC	Input/Output	Function block	HSC applies high presets, low presets and output source values to the high-speed counter.
HSC_SET_STS on page 287	HSC_SET_STS	Input/Output	Function block	HSC_SET_STS manually sets or resets the HSC counting status.
HSCE on page 309	HSCE	Input/Output	Function block	HSCE start, stop and read accumulator value.
HSCE_CFG on page 312	HSCE_CFG	Input/Output	Function block	HSCE_CFG is the high speed counter configuration.
HSCE_CFG_PLS on page 314	HSCE_CFG_PLS	Input/Output	Function block	HSCE_CFG_PLS is the high speed counter PLS configuration.
HSCE_READ_STS on page 317	HSCE_READ_STS	Input/Output	Function block	HSCE_READ_STS reads high speed counter status.
HSCE_SET_STS on page 318	HSCE_SET_STS	Input/Output	Function block	HSCE_SET_STS manually set/reset high speed counter status.
HYSTER on page 478	HYSTER	Process Control	Function block	Boolean hysteresis on difference of reals.
IIM on page 337	IIM	Input/Output	Function block	Updates inputs prior to normal output scan.
INSERT on page 610	INSERT	String manipulation	Function	Inserts sub-strings at user-defined positions within strings.
INTEGRAL on page 480	INTEGRAL	Process Control	Function block	Integrates a real value during the defined cycle time.
IOM on page 339	IOM	Input/Output	Function block	Updates outputs prior to normal output scan.
IPIDCONTROLLER on page 513	IPIDCONTROLLER	Process Control	Function block	Configure and control the inputs and outputs used for the Proportional Integral Derivative (PID) logic.
KEY_READ on page 341		Input/Output	Function block	Micro810 only. Reads the Key status on the optional LCD module when the user display is active.
KEY_READ_REM on page 343	KEY_READ_REM	Input/Output	Function block	Micro820 only. Reads the Key status on the optional Remote LCD module when the user display is active.
LCD on page 325		Input/Output	Function	Micro810 only. Displays a string or number on an LCD screen.
LCD_BKLT_REM on page 328	LCD_BKLT_REM	Input/Output	Function	Sets the Remote LCD backlight parameters in a user program.
LCD_REM on page 330	LCD_REM	Input/Output	Function	Displays user defined messages for the Remote LCD.
LEFT on page 612	LEFT	String manipulation	Function	Extracts characters from the left side of a string.
LFL(LIFO load) on page 485	LFL	Process Control	Function block	Loads 8 bit, 16 bit, 32 bit, or 64 bit data into a user-created array called a LIFO stack.
LFU(LIFO unload) on page 487	LFU	Process Control	Function block	Unloads 8 bit, 16 bit, 32 bit, or 64 bit data from a user-created array called a LIFO (last in first out) stack in the same order data was loaded using the LFL instruction.
LIM_ALRM on page 59	LIM	Alarm	Function block	An alarm with hysteresis on a Real value for high and low limits.

Instruction	Mapped Instruction (Logix Theme)	Category	Type	Description
LIMIT on page 505	LIMIT	Process Control	Function	Restricts integer values to a given interval.
LOG on page 82	LOG	Arithmetic	Function	Calculates the logarithm (base 10) of a Real value.
MAX on page 268	MAX	Data Manipulation	Function	Calculates the maximum of two integer values.
MC_AbortTrigger on page 399	MC_AbortTrigger	Motion	Function block	Aborts Motion function blocks that are connected to trigger events.
MC_Halt on page 402	MC_Halt	Motion	Function block	Commands a controlled motion stop under normal operating conditions.
MC_Home on page 405	MAH	Motion	Function block	Commands the axis to perform the <*search home*> sequence.
MC_MoveAbsolute on page 408	MAM	Motion	Function block	Commands a controlled motion to a specified absolute position.
MC_MoveRelative on page 412	MC_MoveRelative	Motion	Function block	Commands a controlled motion of a specified distance relative to the actual position at the time of the execution.
MC_MoveVelocity on page 416	MCD	Motion	Function block	Commands a never ending controlled motion at a specified velocity.
MC_Power on page 420	MSO	Motion	Function block	Control the power stage, ON or OFF.
MC_ReadActualPosition on page 428	MC_ReadActualPosition	Motion	Function block	Returns the actual position of the feedback axis.
MC_ReadActualVelocity on page 428	MC_ReadActualVelocity	Motion	Function block	Returns the actual velocity of the feedback axis.
MC_ReadAxisError on page 430	MC_ReadAxisError	Motion	Function block	Reads the axis errors not related to the Motion control instruction blocks.
MC_ReadBoolParameter on page 434	MC_ReadBoolParameter	Motion	Function block	Returns the value of a vendor specific parameter of type BOOL.
MC_ReadParameter on page 437	MC_ReadParameter	Motion	Function block	Returns the value of a vendor specific parameter of type Real.
MC_ReadStatus on page 439	MC_ReadStatus	Motion	Function block	Returns the status of the axis with respect to the motion currently in progress.
MC_Reset on page 445	MAFR	Motion	Function block	Transitions the axis state from ErrorStop to StandStill by resetting all internal axis-related errors.
MC_SetPosition on page 448	MRP	Motion	Function block	Shifts the coordinate system of an axis by manipulating the actual position.
MC_Stop on page 450	MAS	Motion	Function block	Commands a controlled motion stop and transfers the axis state to Stopping.
MC_TouchProbe on page 453	MC_TouchProbe	Motion	Function block	Records an axis position at a trigger event.
MC_WriteBoolParameter on page 457	MC_WriteBoolParameter	Motion	Function block	Modifies the value of a vendor specific parameter of type Bool.
MC_WriteParameter on page 459	MC_WriteParameter	Motion	Function block	Modifies the value of a vendor specific parameter of type Real.
MID on page 614	MID	String manipulation	Function	Extracts characters from the middle of a string.
MIN on page 266	MIN	Data Manipulation	Function	Calculates the minimum of two integer values.
MLEN on page 616	MLEN	String manipulation	Function	Calculates the length of a string.
MM_INFO on page 345	MM_INFO	Input/Output	Function block	Reads memory module header information.
MOD on page 83	MOD	Arithmetic	Function	Performs a Modulo calculation on Integer values.
MODULE_INFO on page 348	MODULE_INFO	Input/Output	Function block	Reads module information from a plug-in module or an expansion module.
MOV on page 85	MOV	Arithmetic	Operator	Assigns the input value to the output.
MSG_CIPGENERIC on page 166	MSG	Communications	Function	Sends a CIP generic explicit message.
MSG_CIPSYMBOLIC on page 173	MSG_CIPSYMBOLIC	Communications	Function	Sends a CIP symbolic explicit message.
MSG_MODBUS on page 177	MSG_MODBUS	Communications	Function	Sends a Modbus message.
MSG_MODBUS2 on page 182	MSG_MODBUS2	Communications	Function	Sends a MODBUS/TCP message over an Ethernet Channel.
MUX4B on page 160	MUX4B	Boolean	Function	Multiplexer between four BOOL inputs, outputs a BOOL value.
MUX8B on page 156	MUX8B	Boolean	Function	Multiplexer between eight BOOL inputs, outputs a BOOL value.

Instruction	Mapped Instruction (Logix Theme)	Category	Type	Description
Neg on page 88	NEG	Arithmetic	Operator	Converts a value to a negative.
NOP on page 509	NOP	Program Control	Function	Functions as a placeholder.
NOT on page 151	NOT	Boolean operations	Operator	Converts Boolean values to negated values.
NOT_MASK on page 133	NOT_MASK	Binary operations	Function	Integer bit-to-bit negation mask, inverts a parameter value.
OR on page 149	OR	Boolean operations	Operator	Boolean OR of two or more values.
OR_MASK on page 134	OR_MASK	Binary operations	Function	Integer OR bit-to-bit mask, turns bits on.
PID on page 536	PID	Process Control	Function block	An output instruction that controls physical properties such as temperature, pressure, liquid level, or flow rate using process loops.
PLUGIN_INFO on page 359	PLUGIN_INFO	Input/Output	Function block	Gets module information from a generic plug-in module (excluding Memory Module).
PLUGIN_READ on page 361	PLUGIN_READ	Input/Output	Function block	Reads data from a generic plug-in module (excluding Memory Module).
PLUGIN_RESET on page 363	PLUGIN_RESET	Input/Output	Function block	Resets a generic plug-in module, hardware reset (excluding Memory Module).
PLUGIN_WRITE on page 365	PLUGIN_WRITE	Input/Output	Function block	Writes data to a generic plug-in module (excluding Memory Module).
POW on page 89	XPY	Arithmetic	Function	Calculates the value of a Real number raised to a power of the Real exponent.
PWM on page 489	PWM	Process Control	Function block	Turns the PWM (Pulse Width Modulation) output for a configured PWM channel ON or OFF.
R_TRIG on page 147	OSR	Boolean operations	Function block	Detects a rising edge of a Boolean variable.
RAND on page 91	RAND	Arithmetic	Function	Calculates random integer values from a defined range.
RCP on page 367	RCP	Input/Output	Function block	Reads and writes recipe data to and from an SD memory card.
REPLACE on page 619	REPLACE	String manipulation	Function	Replaces parts of a string with new sets of characters.
RHC on page 332	RHC	Input/Output	Function	Reads high-speed clock.
RIGHT on page 617	RIGHT	String manipulation	Function	Extracts characters from the right side of a string.
ROL on page 136	ROL	Binary operations	Function	For 32-bit integers, rotates integer bits to the left.
ROR on page 137	ROR	Binary operations	Function	For 32-bit integers, rotates integer bits to the right.
RPC on page 334	RPC	Input/Output	Function	Reads user program checksum.
RS on page 148	RS	Boolean operations	Function block	Reset dominant bistable.
RTC_READ on page 369	RTC_READ	Input/Output	Function block	Reads the real-time clock (RTC) module information.
RTC_SET on page 371	RTC_SET	Input/Output	Function block	Sets RTC (real-time clock) data to the RTC module information.
RTO on page 632	RTO	Time	Function block	Retentive timing. Increases an internal timer when input is active, but does not reset the internal timer when input changes to inactive.
SCALER on page 492	SCP	Process Control	Function block	Scales the input value according to output range.
SCL on page 497	SCL	Process Control	Function block	Converts an unscaled input value to a floating point value in engineering units.
SHL on page 139	SHL	Binary operations	Function	For 32-bit integers, moves integers to the left and places 0 in the least significant bit.
SHR on page 141	SHR	Binary operations	Function	For 32-bit integers, moves integers to the right and places 0 in the most significant bit.
SIN on page 93	SIN	Arithmetic	Function	Calculates the sine of a Real value.
SIN_LREAL on page 94	SIN_LREAL	Arithmetic	Function	Calculates the sine of a Long Real value.
SOCKET_ACCEPT on page 554	SOCKET_ACCEPT	Communications	Function block	Accepts a TCP connection request from a remote destination and returns a socket instance used to send and receive data on the newly created connection.

Instruction	Mapped Instruction (Logix Theme)	Category	Type	Description
SOCKET_CREATE on page 557	SOCKET_CREATE	Communications	Function block	Creates an instance of the socket and returns an instance number that is used as an input in any follow-on socket operations.
SOCKET_DELETE on page 562	SOCKET_DELETE	Communications	Function block	Deletes a created socket instance.
SOCKET_DELETEALL on page 564	SOCKET_DELETEALL	Communications	Function block	Deletes all created socket instances.
SOCKET_INFO on page 566	SOCKET_INFO	Communications	Function block	Returns information for the socket such as error codes and execution status.
SOCKET_OPEN on page 572	SOCKET_OPEN	Communications	Function block	Opens the connection for the specified destination address for Transmission Control Protocol (TCP) connections. For User Datagram Protocol (UDP) connections, associates a destination IP address and port number with the specified socket.
SOCKET_READ on page 576	SOCKET_READ	Communications	Function block	Reads data on a socket.
SOCKET_WRITE on page 580	SOCKET_WRITE	Communications	Function block	Sends data on a socket.
SQRT on page 96	SQR	Arithmetic	Function	Calculates the square root of a Real value.
SR on page 152	SR	Boolean operations	Function block	Set dominant bistable.
STACKINT on page 494	STACKINT	Process Control	Function block	Manages stack of integers.
STIS on page 379	STS	Interrupt	Function	Starts the selected timed user interrupt (STI) timer from the control program rather than starting automatically.
SUS on page 509	SUS	Program Control	Function block	Suspends the execution of the <M800 controller>.
SYS_INFO on page 373	SYS_INFO	Input/Output	Function block	Reads the status data block for the Micro800 controller.
TAN on page 99	TAN	Arithmetic	Function	Calculates the tangent of a Real value.
TAN_LREAL on page 100	TAN_LREAL	Arithmetic	Function	Calculates the tangent of a Long Real value.
TDF on page 636	TDF	Time	Function	Computes the time difference between TimeA and TimeB.
TND on page 504	TND	Process Control	Function	Stops the current cycle of the user program scan.
TOF on page 623	TOF	Time	Function block	Off-delay timing. Increases an internal timer up to a given value.
TON on page 625	TON	Time	Function block	On-delay timing. Increases an internal timer up to a given value.
TONOFF on page 628	TONOFF	Time	Function block	Delay turning on an output on a true rung, and then delay turning off the output on the false rung.
TOW on page 638	TOW	Time	Function	Turns on an output if the value of the real-time clock is in the range of the Time of Week setting.
TP on page 630	TP	Time	Function block	Pulse timing. On a rising edge, increases an internal timer up to a given value.
TRIMPOT_READ on page 375	TRIMPOT_READ	Input/Output	Function block	Reads the trimpot value from a specific trimpot.
TRUNC on page 102	TRN	Arithmetic	Function	Truncates Real values, leaving just the Integer.
TTABLE on page 153	TTABLE	Boolean	Function	Provides the value of the output based on the combination of inputs.
UIC on page 381	UIC	Interrupt	Function	Clears the lost bit for the selected user interrupt.
UID on page 382	UID	Interrupt	Function	Disables a specific user interrupt.
UIE on page 384	UIE	Interrupt	Function	Enables a specific user input.
UIF on page 386	UIF	Interrupt	Function	Flushes or removes a pending user input.
XOR on page 151	XOR	Boolean operations	Operator	Boolean exclusive OR of two values.
XOR_MASK on page 142	XOR_MASK	Binary operations	Function	Integer exclusive OR bit-to-bit mask, returns inverted bit values.

Ladder Diagram (LD) language

A Ladder Diagram (LD) is a graphical representation of Boolean equations that combines contacts (input arguments) with coils (output results). Using graphic symbols in a program chart (organized like a relay ladder wiring diagram), the LD language describes the tests and modifications of Boolean data.

LD graphic symbols are organized within the chart as an electrical contact diagram. The term "ladder" comes from the concept of rungs connected to vertical power rails at both ends where each rung represents an individual circuit.

Connected Components Workbench provides an LD language editor and supports the elements on page 33 and instructions that are supplied with the Connected Components Workbench software only.

Ladder Diagram (LD) program

A Ladder Diagram (LD) is a graphical representation of Boolean equations that combines contacts (input arguments) with coils (output results). Using graphic symbols in a program chart (organized like a relay ladder wiring diagram), the LD language describes the tests and modifications of Boolean data.

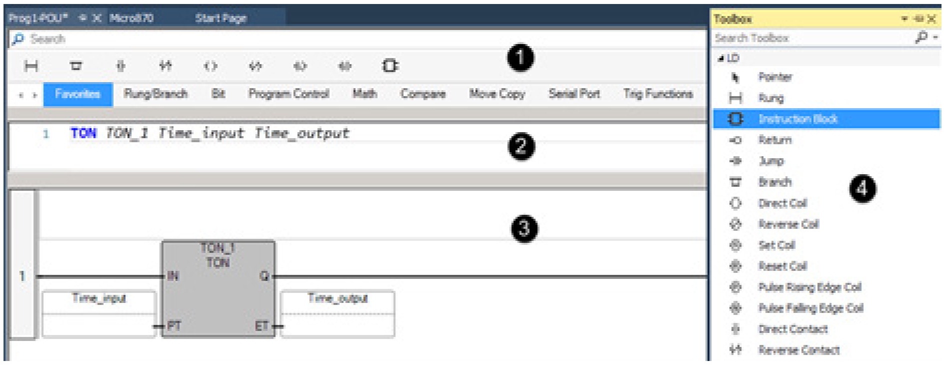
The LD language uses graphic symbols in a program chart, organized like a relay ladder wiring diagram, to describe the tests and modifications of Boolean data.

Connected Components Workbench provides an Ladder Diagram language editor on page 53 that supports the elements and instructions that are supplied with the Connected Components Workbench software only.

LD program development environment

The language editor for a Ladder Diagram (LD) program where you develop an LD Program Organizational Unit (POU).

The following picture shows the main areas of LD program development environment.

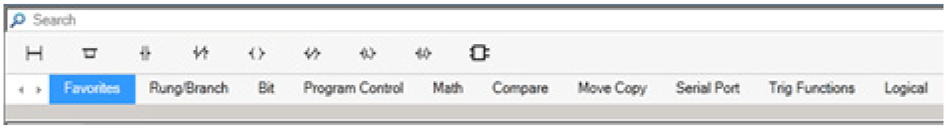


```
1  TON TON_1 Time_input Time_output
```

No.	Name	Description
1	Instruction toolbar on page 30	Quickly select an instruction element and place it in the LD graphical editor, or single click to add in the LD text editor.
2	LD text editor	Edits the logic using ASCII instruction mnemonics.
3	LD graphic editor	Edits the logic using graphical instruction elements.
4	LD toolbox	Adds elements to the LD graphical editor.

Instruction Toolbar (LD)

Instruction Toolbar is the colloquial name for the secondary pane in the language editor pane, which functions like a toolbar and is used to add language elements on page 30 such as instructions to the language editor workspace. It is complementary to the general workbench toolbox.



Item	Description
Tabs	Lists the instruction elements by category. Click on a category to see the instructions within that category. Arrow keys can also be used to select different categories. Click on the arrows to the left of the tabs to scroll through the categories. Click on the down arrow to the right of the tabs to see a list of all the categories.
Instructions	Lists the instructions that correspond to the Tabs category selected. A category may also have basic instruction elements pinned at the beginning of the list. Click on the arrows to the left of the instructions to scroll through the instructions.
Search	Filters the instructions by category immediately as names or description keyword is entered. Click on the category name to see the filtered instructions for that category.
Favorites	Customizes a list of instructions that can be quickly found and added to your program.

Add instruction elements from instruction toolbar

Use the Instruction toolbar on page 30 to search, navigate, and add instructions to Ladder Diagram (LD) program's language editor workspace.

To add an instruction element from Instruction Toolbar to the language editor:

1. Click on the category tab, which includes the instruction you want to add. You can navigate the tab using the Arrow Keys and reorder their position by a drag-and-drop operation.
2. Select the instruction by a drag-and-drop operation or clicking on it.
3. (optional) To quickly locate the instruction, click in the search field and then you can type to find the instruction element by name or keyword. To exit search and enable arrow key navigation, press the **Esc** key.
4. (optional) Right-click on the instruction and select **Add to Favorites** to add it to **Favorites** tab and select **Remove from Favorites** to remove it. The favorites setting will be automatically saved.

Ladder Diagram (LD) elements

Ladder Diagram (LD) elements are the components used to build a ladder diagram program. All the elements listed in the following table can be added to a ladder diagram program within Connected Components Workbench.

Element	Description
Rung on page 33	Represents a group of circuit elements that lead to the activation of a coil.
Instruction Block (LD) on page 39	Instructions include operators, functions, and function blocks including user-defined function blocks.
Branch on page 36	Two or more instructions in parallel.
Coil on page 39	Represents the assignment of outputs or internal variables. In an LD program, a coil represents an action.
Contact on page 45	Represents the value or function of an input or internal variable.
Return on page 50	Represents the conditional end of a diagram output.
Jump on page 51	Represents the conditional and unconditional logic in the LD program that control the control the execution of diagrams.

Rung

Rungs are graphic components of Ladder Diagram (LD) programs that represent a group of circuit elements that lead to the activation of a coil. Use labels to identify rungs within the diagram. Labels, along with jumps, control the execution of a diagram. Comments are free-format text you can add above the rung for documentation purposes.

Add a rung to a LD program

Rungs on page 33 are graphic components of a Ladder Diagram (LD) program that represent a group of circuit elements that lead to the activation of a coil.

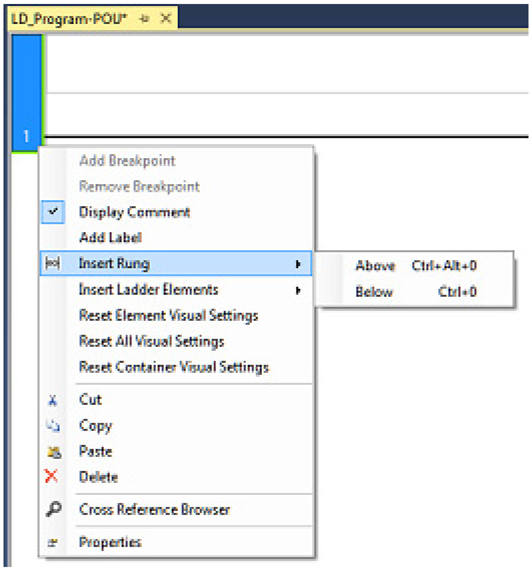
In Connected Components Workbench you can add a rung to a Ladder Diagram (LD) program from the:

- Language Diagram (LD) language editor
- Multi-language Editor located on the Tools menu
- LD Toolbox

To add a rung element to a Ladder Diagram program:

1. In the LD language editor, right-click an existing rung and then either:
   - Click **Copy**, and then click **Paste** to insert a copy of the rung into the language editor.
   - Click **Insert Rung**, and then either:
     - Click **Above** to add the rung above the selected rung.

- Click **Below** to add the rung below the selected rung.



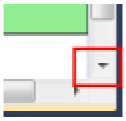
- Select a rung or an element in the LD language editor, and then press either:
  - **CTRL+ALT+0** - to add the rung above the selected rung.
  - **CTRL+0** - to add the rung below the selected rung.
- Select a rung or an element in the LD language editor, and then press either:
  - Click **Tools** > **Multi-language Editor** > **Insert Rung Below** to add the rung below the selected rung.
  - Click **Tools** > **Multi-language Editor** > **Insert Rung Above** to add the rung above the selected rung.

2. (optional) Open the LD Toolbox to show the rung element,
   - To insert a rung below an existing rung, select the rung in the LD language editor and then double-click **Rung** in the LD Toolbox.

     If an element is not selected prior to double-clicking **Rung** in the LD Toolbox, the rung is inserted below the last rung in the LD language editor.
   - Select **Rung** and then drag the element into the LD language editor.

     A plus sign (+) appears in the LD language editor to show a valid target. Release the mouse button to add the element.

Tip: If the ladder diagram contains more than 355 rungs, use the down triangle rather than the scroll bar to view additional rungs.



Rung labels

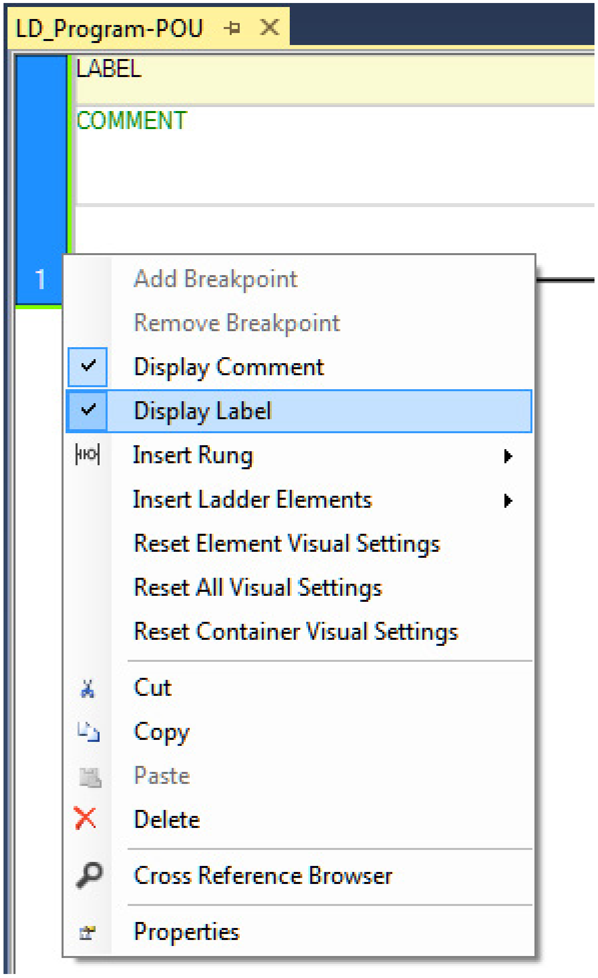
Labels are optional additions for every rung in the Ladder Diagram (LD) language editor.

Labels can be an unlimited number of characters, beginning with a letter or underscore character followed by letters, numbers, and underscore characters. Labels cannot have spaces or special characters (for example, '+', '-', or '\').

To add a label for a rung in the LD language editor:

1. To add a label to a rung, right-click the rung to open the LD language editor menu, and then select **Display Label**.

   The rung updates to include the label and the LD language editor menu shows a checkmark next to **Display Label**.



2. Select the **Label**, and type a description.
3. (optional) To remove the label, click **Display Label** from the LD language editor menu.

Rung comments

Comments are optional for every rung in the Ladder Diagram (LD) language editor. By default, a comment is included when you add a rung element in the LD language editor on page 33.

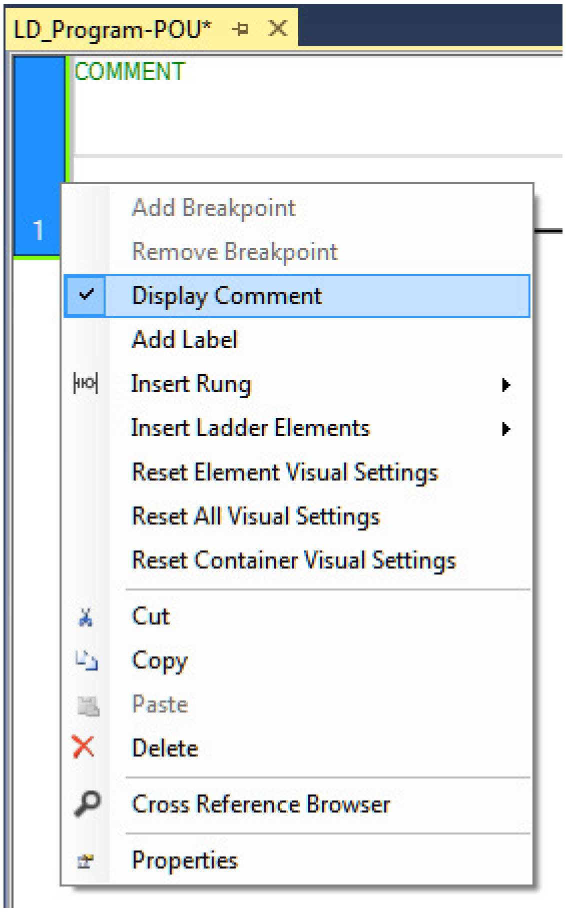
Comments are:

- Entered in the space above the rung.

- Saved in rich text format.
- Stored in the controller.

To add or remove a comment for a rung in the LD language editor:

1. To remove a comment, right-click the rung to open the LD language editor menu.

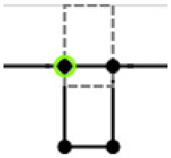


2. From the LD language editor menu, click **Display Comment**. The comment is removed from the rung and the check mark next to **Display Comment** on the LD language editor menu is removed.
3. (optional) To add a comment to a rung, click **Display Comment** from the LD language editor menu.

Branch

Branches create alternative routing for connections. You can add parallel branches to elements on a rung using the Ladder Diagram (LD) language editor on page 37.

Branch example



Add a branch to a LD program

Branches on page 36 are graphic components of Ladder Diagram (LD) programs that create alternative routing for connections and may include parallel branches.

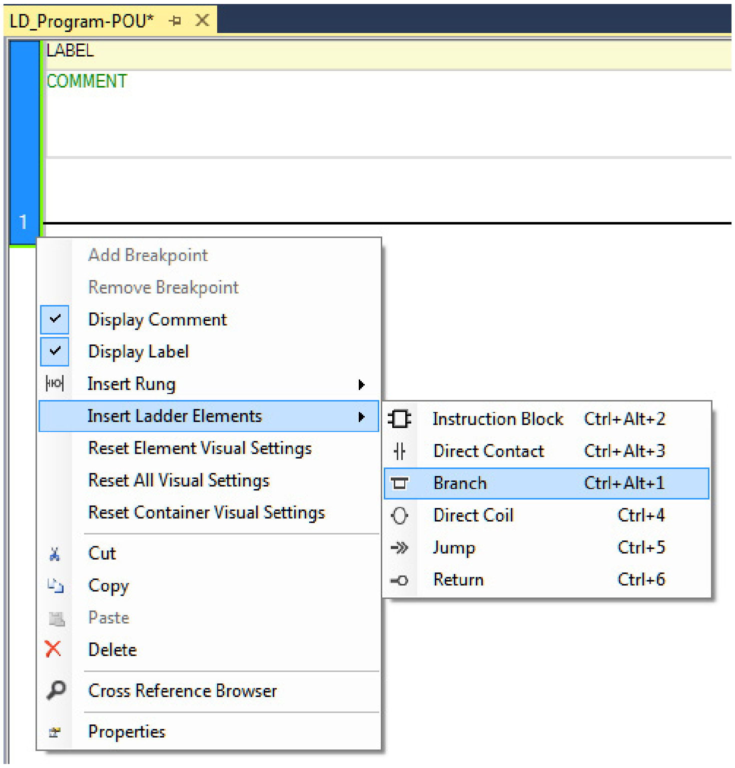
In Connected Components Workbench you can add a branch to a Ladder Diagram (LD) program from the:

- Ladder Diagram (LD) language editor
- Multi-language Editor located on the Tools menu
- LD Toolbox
- Instruction Toolbar (LD)

To add a branch to a Ladder Diagram program:

1. In the LD language editor, verify the LD program has a defined rung for the branch. Then do one of the following:
   - Right-click a rung or an element in the LD language editor, select **Insert Ladder Elements** and then click **Branch**.

     If the **Variable Selector** opens, select a variable or click **OK** without selecting a variable to add the **Branch** element.



- Select a rung or element in the LD language editor and then press either:

  CTRL+ALT+1 to add the branch to the left of the selected element or rung.
  CTRL+1 to add the branch to the right of the selected element.

- Select a rung or an element in the LD language editor and then either:

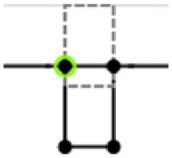
  Click **Tools** > **Multi-language Editor** > **Insert Branch Before** to add the branch to the left of the selected element.
  Click **Tools** > **Multi-language Editor** > **Insert Branch After** to add the branch to the right of the selected element.

- Open the LD Toolbox to show the branch element and then either:

  Double-click the branch element to add it to the LD language editor.
  Drag the branch element into the LD language editor and position it on the rung.
  A plus sign (+) appears in the LD language editor to show a valid target. Release the mouse button to add the element.

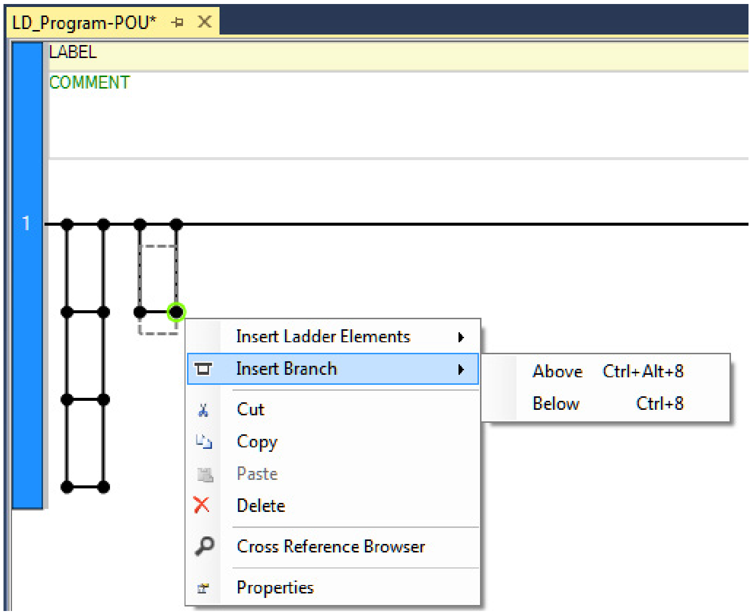
2. (optional) To insert a parallel branch:

   a. In the LD language editor, right-click the branch to open the LD language editor menu.



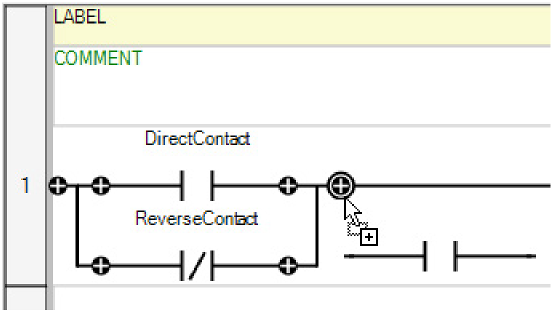
   b. From the LD language editor menu, select **Insert Branch** and then click either:

   Above to add a branch above the selected branch.
   Below to add the branch below the selected branch.



3. (optional) To move a branch element to another location in a LD program, select the element and drag the element to new a location in the LD program.

   A plus sign (+) appears in the LD language editor to show a valid target. Release the mouse button to insert the element in the target location.



Instruction Block (LD)

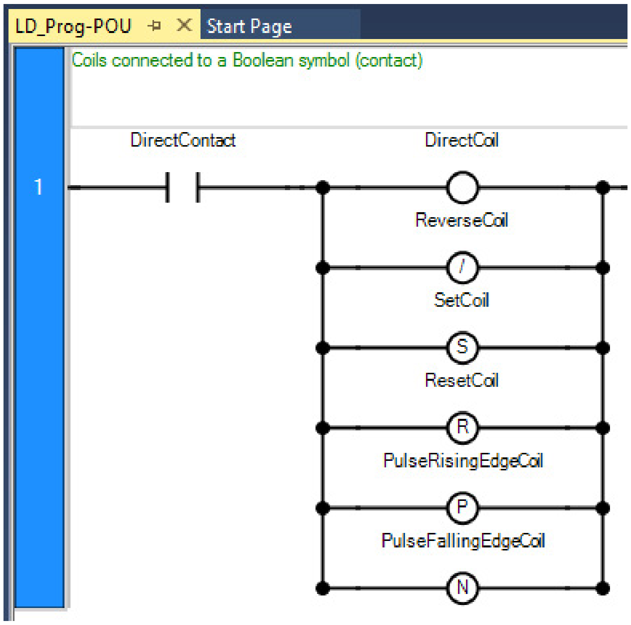
A Ladder Diagram (LD) Instruction Block element is a IEC 61131-3 compliant functional element in a LD program that can be a function block, a function, a user-defined function block, a user-defined function, or an operator.

Coil

Coils are graphic components of Ladder Diagram (LD) programs that represent the assignment of an output or of an internal variable. In LD programs, a coil represents an action. A coil must be connected on the left to a Boolean symbol, such as a contact, or to a Boolean output of an instruction block. Coils can only be added to a defined rung in the LD language editor. The coil definition can be modified after the coil is added to the rung.

The following example shows the coil element types available for Ladder Diagram programs.

Example: Coils



Add coil elements

Coils are graphic components of Ladder Diagram (LD) programs that represent an action taken an output or of an internal variable.

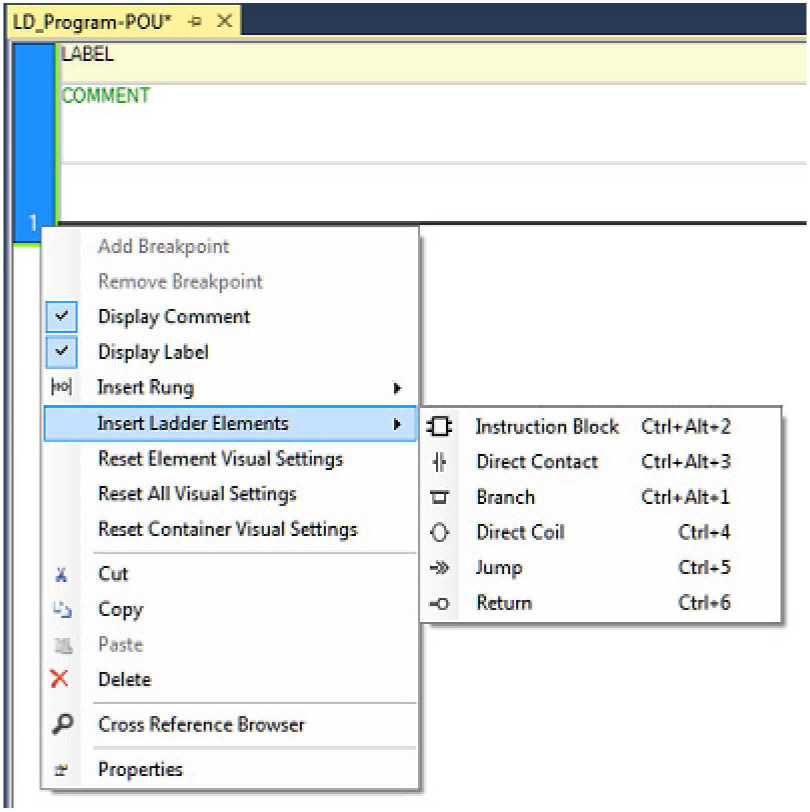
In Connected Components Workbench you can add a coil to a Ladder Diagram (LD) program from the:

- Ladder Diagram (LD) language editor
- Multi-language Editor located on the Tools menu
- LD Toolbox
- Instruction Toolbar (LD)

1. To add a coil element to a Ladder Diagram program:in the LD language editor, verify the LD program has a defined rung for the coil. Then do one of the following:

   - Right-click a rung or an element in the LD language editor, select **Insert Ladder Elements** and then click **Direct Coil**.

     If the **Variable Selector** opens, select a variable or click **OK** without selecting a variable to add the **Direct Coil** element.



- Select a rung or an element in the LD language editor, and then press **CTRL+4** to add the **Direct Coil** to the right side of the rung.
- Select a rung or an element in the LD language editor, and then click **Tools** > **Multi-language Editor** > **Insert Coil** to add the coil to the right side of the rung.
- Open the LD Toolbox to show the coil elements (**Direct Coil**, **Reverse Coil**, **Set Coil**, **Reset Coil, Pulse Rising Edge Coil**, **Pulse Falling Edge Coil**) and then either:

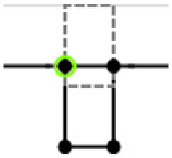
  Double-click the coil element to add it to the LD language editor.

Drag the coil element into the LD language editor and position it on the rung.

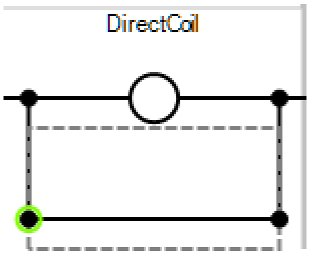
A plus sign (+) appears in the LD language editor to show a valid target. Release the mouse button to add the element.

2. (optional) To insert a parallel coil

a. In the LD language editor, verify the LD program has a defined branch and then right-click the top level of the branch on page 36 to open the LD language editor menu.

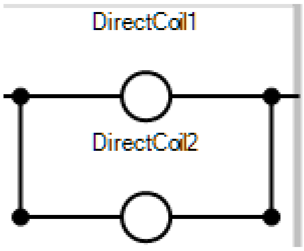


b. From the LD language editor menu, select **Insert Ladder Elements** and then click **Direct Coil**. The element is inserted on the top level of the branch.



c. Right-click the bottom level of the branch to open the LD language editor menu.

d. From the LD language editor menu, select **Insert Ladder Elements** and then click **Direct Coil**. The element is inserted on the bottom level of the branch.



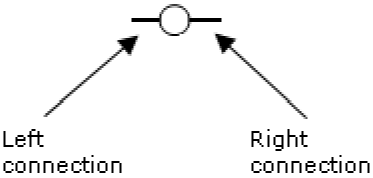
3. (optional) To change the type of coil, in the LD language editor select the coil, and then press the **space bar** until the coil type displays in the language editor.

Every time the space bar is pressed the coil type changes from direct, to reverse, to set, to reset, to pulse rising edge, to pulse falling edge.

Direct Coil

Coils on page 39 are graphic components of Ladder Diagram (LD) programs that represent the assignment of an output or of an internal variable.

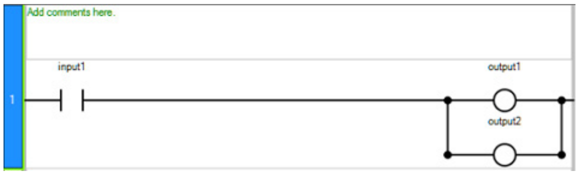
A direct coil supports a Boolean output of a connection line Boolean state.



The associated variable is assigned with the Boolean state of the left connection. The state of the left connection is propagated into the right connection. The right connection must be connected to the right vertical power rail (unless you have parallel coils, where only the upper coil must be connected to the right vertical power rail).

The associated Boolean variable must be an output or it must be user-defined.

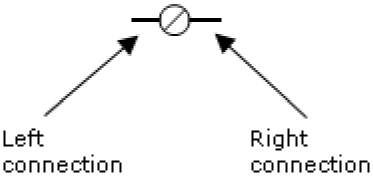
Direct coil example



Reverse Coil

Coils on page 39 are graphic components of Ladder Diagram (LD) programs that represent the assignment of an output or of an internal variable.

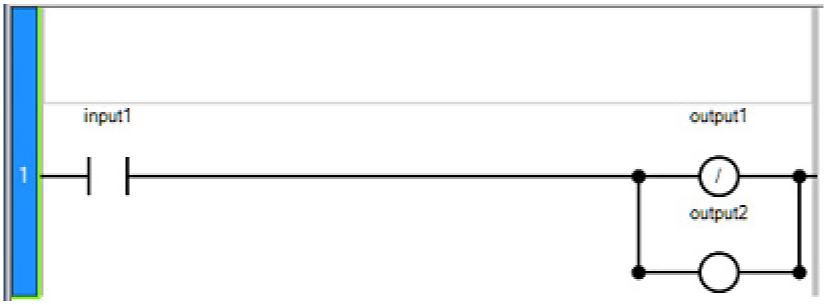
A reverse coil element supports a Boolean output according to the Boolean negation of a connection line state.



The associated variable is assigned with the Boolean negation of the state of the left connection. The state of the left connection is propagated into the right connection. The right connection must be connected to the right vertical power rail (unless you have parallel coils, where only the upper coil must be connected to the right vertical power rail).

The associated Boolean variable must be output or it must be user-defined.

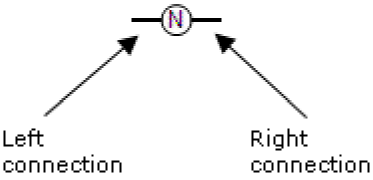
Reverse Coil example



Pulse Falling Edge Coil

Coils on page 39 are graphic components of Ladder Diagram (LD) programs that represent the assignment of an output or of an internal variable.

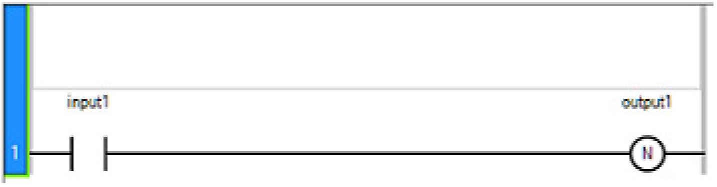
Pulse falling edge on page 43 (or negative) coils support a Boolean output of a connection line Boolean state.



The associated variable is set to TRUE when the Boolean state of the left connection falls from TRUE to FALSE. The output variable resets to FALSE in all other cases. The state of the left connection is propagated into the right connection. The right connection must be connected to the right vertical power rail (unless you have parallel coils, where only the upper coil must be connected to the right vertical power rail).

The associated Boolean variable must be output or it must be user-defined.

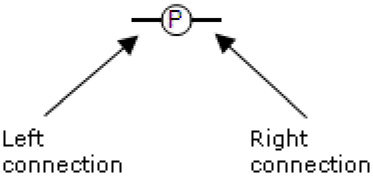
Pulse Falling Edge Coil example



Pulse Rising Edge Coil

Coils on page 39 are graphic components of Ladder Diagram (LD) programs that represent the assignment of an output or of an internal variable.

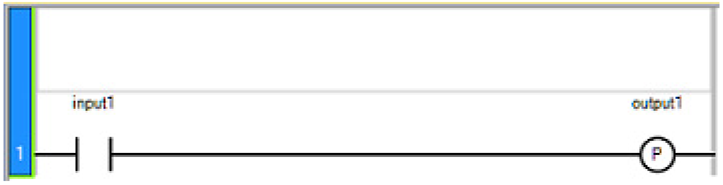
Pulse rising edge (or positive) coils support a Boolean output of a connection line Boolean state.



The associated variable is set to TRUE when the Boolean state of the left connection rises from FALSE to TRUE. The output variable resets to FALSE in all other cases. The state of the left connection is propagated into the right connection. The right connection must be connected to the right vertical power rail (unless you have parallel coils, where only the upper coil must be connected to the right vertical power rail).

The associated Boolean variable must be output or user-defined.

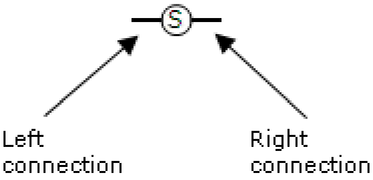
Pulse Rising Edge Coil example



Set Coil

Coils are graphic components of Ladder Diagram (LD) programs that represent the assignment of an output or of an internal variable. In LD programs, a coil represents an action.

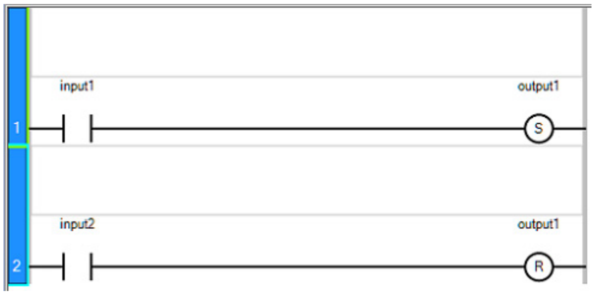
Set coils support a Boolean output of a connection line Boolean state.



The associated variable is set to TRUE when the Boolean state of the left connection becomes TRUE. The output variable keeps this value until an inverse order is made by a Reset coil. The state of the left connection is propagated into the right connection. The right connection must be connected to the right vertical power rail (unless you have parallel coils, where only the upper coil must be connected to the right vertical power rail).

The associated Boolean variable must be output or it must be user-defined.

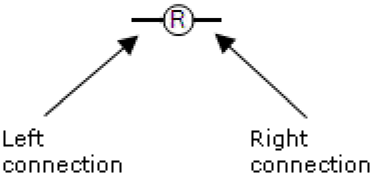
Set Coil example



Reset Coil

Coils on page 39 are graphic components of Ladder Diagram (LD) programs that represent the assignment of an output or of an internal variable.

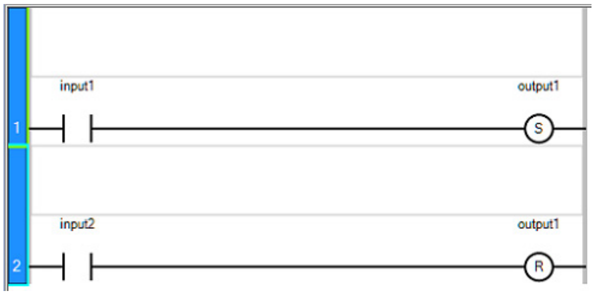
Reset coils support a Boolean output of a connection line Boolean state.



The associated variable is reset to FALSE when the Boolean state of the left connection becomes TRUE. The output variable keeps this value until an inverse order is made by a Set coil. The state of the left connection is propagated into the right connection. The right connection must be connected to the right vertical power rail (unless you have parallel coils, where only the upper coil must be connected to the right vertical power rail).

The associated Boolean variable must be output or user-defined.

Reset Coil example

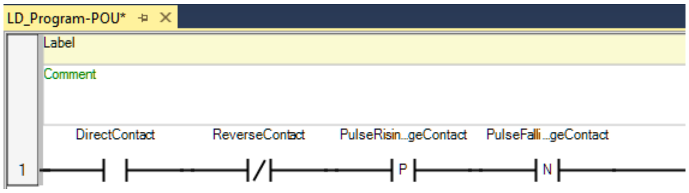


Contact

Contacts are graphic components of Ladder Diagram (LD) programs. Depending on the type, a contact represents the value or function of an input or of an internal variable. Contacts can only be added to a defined rung in the LD language editor. After a contact is added, its definition can be modified.

The following example shows the contact element types available for Ladder Diagram programs.

Example: Contacts



Add a contact to a LD program

Contacts are graphic components of a Ladder Diagram (LD) program. Depending on the type, a contact represents the value or function of an input or of an internal variable. Contacts can only be added to a defined rung in the LD language editor.

In Connected Components Workbench you can add a contact to a Ladder Diagram (LD) program from the:

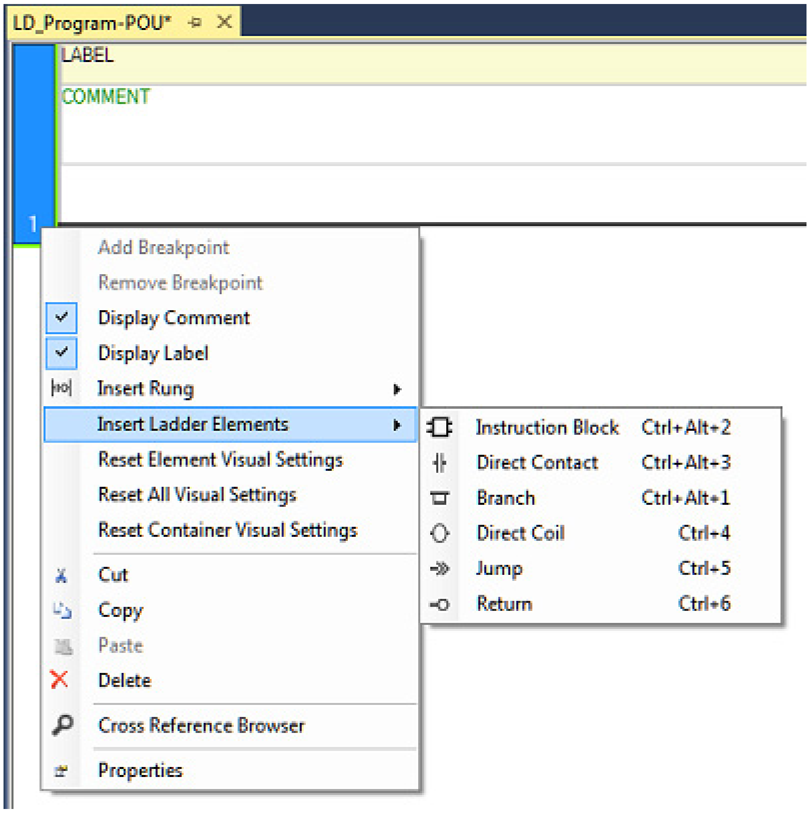
- Ladder Diagram (LD) language editor
- Multi-language Editor located on the Tools menu
- LD Toolbox
- Instruction Toolbar (LD)

To add a contact element to a Ladder Diagram program:

1. In the LD language editor, verify the LD program has a defined rung for the contact. Then do one of the following:

   - Right-click a rung or an element in the LD language editor, select **Insert Ladder Elements** and then click **Direct Contact**.

     If the **Variable Selector** opens, select a variable or click **OK** without selecting a variable to add the **Direct Contact** element.



   - Select a rung or an element in the LD language editor, and then press either:

     CTRL+ALT+3 to add the Direct Contact element to the left side of the selected element or the rung.

CTRL+3 to add the Direct Contact element to the right side of the selected element or the rung.

- Select a rung or an element in the LD language editor and then either:

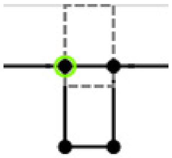
  Click **Tools** > **Multi-language Editor** > **Insert Contact Before** to add the contact to the left side of the selected element or rung.
  Click **Tools** > **Multi-language Editor** >**Insert Contact After** to add the contact to the right side of the selected element or rung.

- Open the LD Toolbox to show the contact elements (**Direct Contact**, **Reverse Contact**, **Pulse Rising Edge Contact**, **Pulse Falling Edge Contact**) and then either:

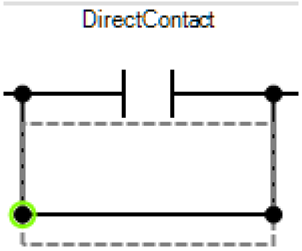
  Double-click the contact element to add it to the LD language editor.
  Drag the contact element into the LD language editor and position it on the rung.
  A plus sign (+) appears in the LD language editor to show a valid target. Release the mouse button to add the element.

2. (optional) To insert a parallel contact:

a. In the LD language editor, verify the LD program has a defined branch on page 36 and then right-click the top level of the branch to open the LD language editor menu.

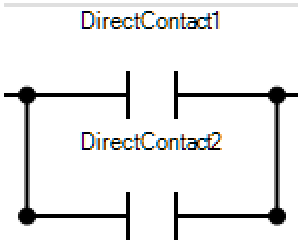


b. From the LD language editor menu, select **Insert Ladder Elements** and then click **Direct Contact**. The element is inserted on the top level of the branch.



c. Right-click the bottom level of the branch to open the LD language editor menu.

d. From the LD language editor menu, select **Insert Ladder Elements** and then click **Direct Contact**. The element is inserted on the bottom level of the branch.



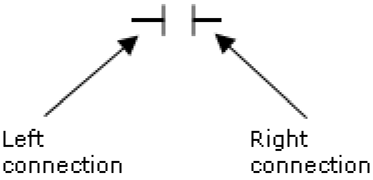
3. (optional) To change the type of contact, in the language editor select the contact, and then press the **space bar** until the the contact type displays in the language editor.

   Every time the space bar is pressed the contact type changes from direct, to reverse, to pulse rising edge, to pulse falling edge.

Direct Contact

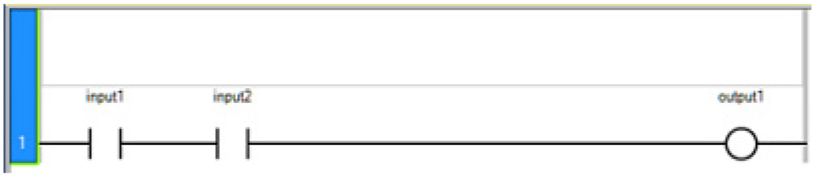
Contacts on page 45 are graphic components of Ladder Diagram (LD) programs.

Direct contacts support a Boolean operation between a connection line state and a Boolean variable.



The state of the connection line on the right of the contact is the logical AND between the state of the left connection line and the value of the variable associated with the contact.

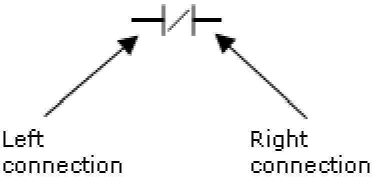
Direct Contact example



Reverse Contact

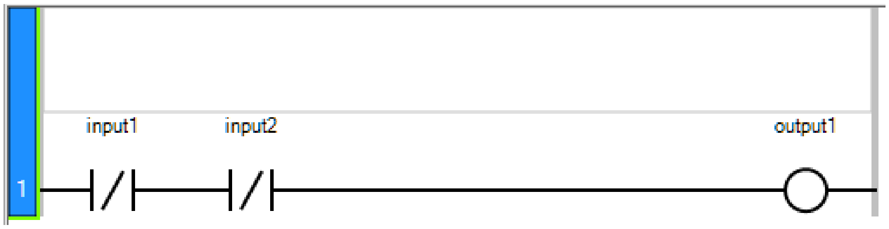
Contacts on page 45 are graphic components of Ladder Diagram (LD) programs.

Reverse contacts support a Boolean operation between a connection line state and the Boolean negation of a Boolean variable.



The state of the connection line on the right of the contact is the logical AND between the state of the left connection line and the Boolean negation of the value of the variable associated with the contact.

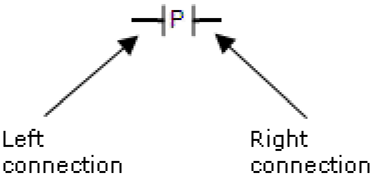
Reverse Contact example



Pulse Rising Edge Contact

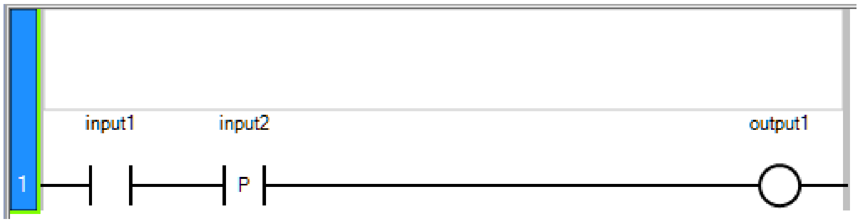
Contacts on page 45 are graphic components of Ladder Diagram (LD) programs.

Pulse rising edge (or positive) contacts support a Boolean operation between a connection line state and the rising edge of a Boolean variable.



The state of the connection line on the right of the contact is set to TRUE when the state of the connection line on the left is TRUE, and the state of the associated variable rises from FALSE to TRUE. The state is reset to FALSE in all other cases.

Pulse Rising Edge Contact example



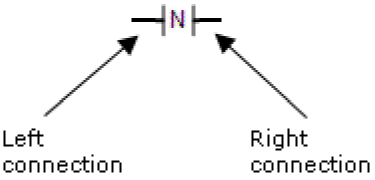
Recommendation: Restrict the use of output variables with edge contacts

We recommend not using outputs or variables with a Pulse rising edge contact on page 49 (positive) or a Pulse falling edge contact on page 50 (negative). These contacts are for physical inputs in a ladder diagram. To detect the edge of a variable or an output, we recommend using the R_TRIG/F_TRIG function block, which is supported and works in any language at any location in your program.

Pulse Falling Edge Contact

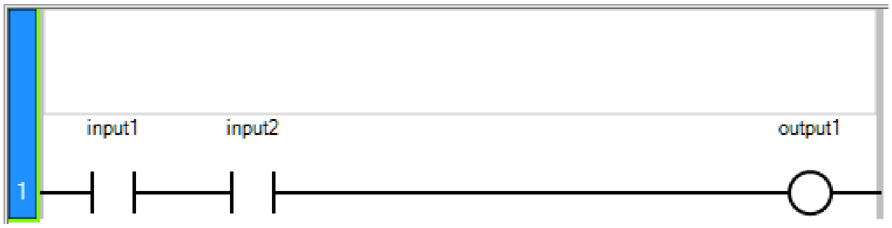
Contacts are graphic components of Ladder Diagram (LD) programs.

Pulse falling edge (or negative) contacts support a Boolean operation between a connection line state and the falling edge of a Boolean variable.



The state of the connection line on the right of the contact is set to TRUE when the state of the connection line on the left is TRUE, and the state of the associated variable falls from TRUE to FALSE. The state is reset to FALSE in all other cases.

Pulse Falling Edge Contact example



Recommendation: Restrict the use of output variables with edge contacts

We recommend not using outputs or variables with a Pulse rising edge contact on page 49 (positive) or a Pulse falling edge contact on page 50 (negative). These contacts are for physical inputs in a ladder diagram. To detect the edge of a variable or an output, we recommend using the R_TRIG/F_TRIG function block, which is supported and works in any language at any location in your program.

Return

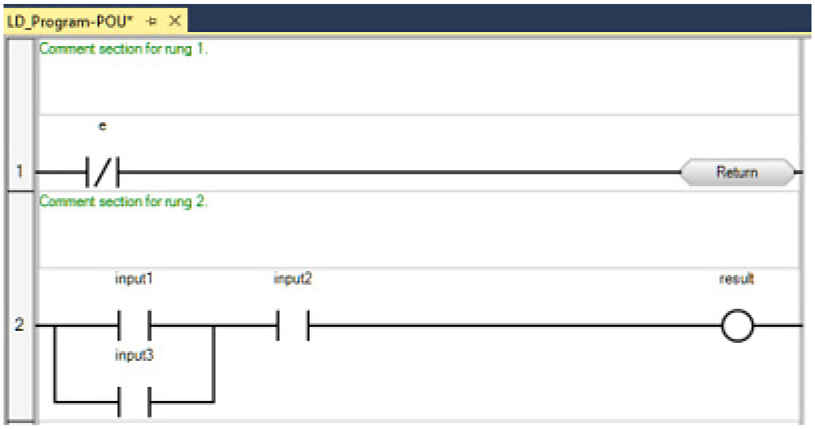
Returns are outputs that represent a conditional end of a Ladder Diagram (LD) program.

You cannot place connections to the right of a return element.

When the left connection line has the TRUE Boolean state, the diagram ends without executing the instructions located on the next lines of the diagram.

When the LD diagram is a function, its name is associated with an output coil to set the return value (returned to the calling diagram).

Return example



To insert a return in a Ladder Diagram program:

Do one of the following:

- Right-click a rung or an element in the LD language editor, select **Insert Ladder Elements** and then click **Return**.
- Select a rung or element in the LD language editor and then press **CTRL+6**.
- Select a rung or an element in the LD language editor and then click **Tools** > **Multi-language Editor** > **Insert Return**.
- Open the LD Toolbox to show the return element and then either:

  Double-click the return element to add it to the LD language editor.
  Drag the return element into the LD language editor and position it on the rung.
  A plus sign (+) appears in the LD language editor to show a valid target. Release the mouse button to add the element.

Jump

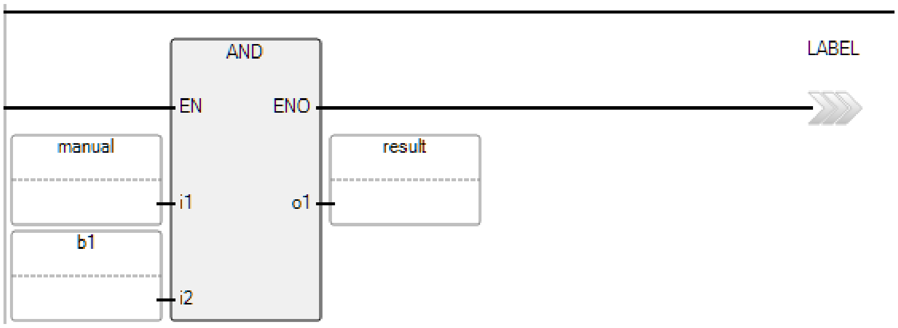
Jumps are conditional or unconditional elements that control the execution of Ladder Diagram (LD) programs.

Jump notation

The following notation indicates a jump to a label:

>>LABEL - Jump to a label where the label name is "LABEL"

Jump example



To insert a jump:

Do one of the following:

- Right-click a rung or an element in the LD language editor, select **Insert Ladder Elements** and then click **Jump**.
- Select a rung or element in the LD language editor and then press **CTRL+5**.
- Select a rung or an element in the LD language editor and then click **Tools** > **Multi-language Editor** > **Insert Jump**.
- Open the LD Toolbox to show the jump element and then either:

  Double-click the jump element to add it to the LD language editor.
  Drag the jump element into the LD language editor and position it on the rung.
  A plus sign (+) appears in the LD language editor to show a valid target. Release the mouse button to add the element.

Instruction blocks in LD programs

The Connected Components Workbench instruction set includes IEC 61131-3 compliant instruction blocks. Instruction blocks collectively include function blocks, functions and operators. You can connect instruction block inputs and outputs to variables, contacts, coils, or other instruction block inputs and outputs.

Instruction block conventions

The IEC61131-3 programming language specification addresses numerous aspects of programmable controllers including the operating system execution, data definitions, programming languages, and instruction sets. The IEC61131-3 specification provides a minimum set of functionalities that can be extended to meet end user applications.

Instruction block names

Functions and function blocks are represented by a box that displays the name of the instruction, and the short version of the parameter names. For function blocks, the instance name is displayed above the function block name.

Instruction block return parameters

- The return parameter of a function has the same name as the function. The return parameter is the only output.
- The return parameters of a function block can have any name. Multiple return parameters can provide multiple outputs.
- You can define the parameters of programs for multiple devices by navigating the tabs for individual devices displayed in the **Parameter** view.

Work in the LD language editor

When you add items to a rung in the Ladder Diagram (LD) program, they are added according to the following criteria.

- The first element on a rung is inserted at the position you select in the ladder diagram.
- Subsequent elements are inserted to the right of the selected item on the rung.
- You cannot insert an element to the right of a coil return or jump.

Different methods to add an element to Ladder Diagram program:

- **LD Ladder Editor**
  - Add elements, delete elements, and copy and paste elements.
  - Use LD keyboard shortcuts on page 55 to add elements.
- **LD Text Editor**
  - Add, modify and delete elements.
  - Copy and paste elements from/to RSLogix 500 and <RSLX5000>.
- **Multi-language Editor** located on the **Tools** menu
  - Add elements.
  - Export an image of the LD program.
  - Enable or disable the automatic opening of the **Variable Selector** and **Instruction Block Selector**.
- **LD Toolbox**
  - Add elements.
- **Instruction Toolbar**
  - Add elements.

You can replace an assigned variable directly from the language editor, or from the **Variable Selector**.

To modify a variable from the language editor:

1. In the language editor, click the variable name to display a drop-down list of global and local variables.
2. Do one of the following:
   - Enter a new variable name and double-click the variable to open **Variable Selector**. Press **Enter** to confirm the new variable.
   - Select a different variable name from the drop-down list.

To modify a variable from the Variable Selector:

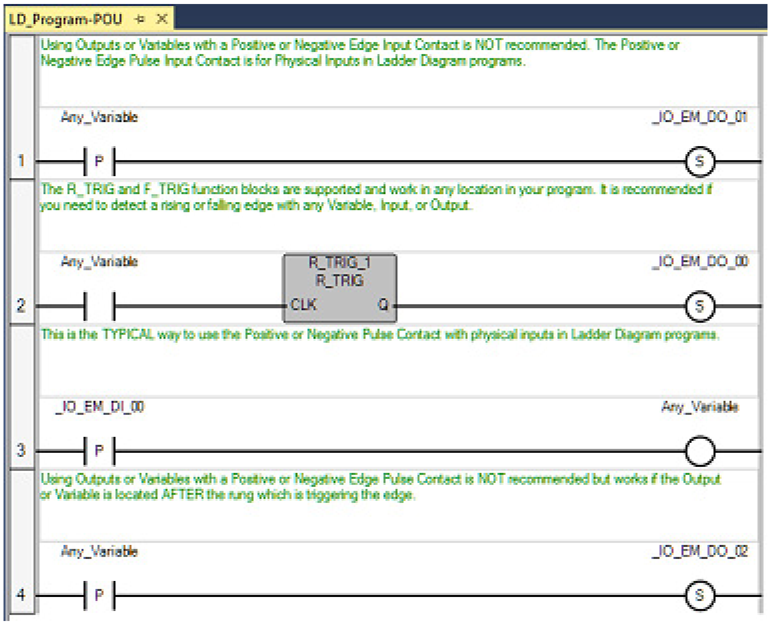
1. In the language editor, double-click the variable to open the **Variable Selector**.
2. Click the variable name, then select a different variable from the drop-down list of global and local variables.
3. Click an existing variable, then type constant values in the text box provided.

Ladder Diagram (LD) program examples

The following examples are Ladder Diagram (LD) programs.

Example: R_TRIG function block

The following example program shows the recommended usage of an R_TRIG function block used to detect an edge while connected to the controller.



Using Outputs or Variables with a Positive or Negative Edge Input Contact is NOT recommended. The Positive or Negative Edge Pulse Input Contact is for Physical Inputs in Ladder Diagram programs.

The R_TRIG and F_TRIG function blocks are supported and work in any location in your program. It is recommended if you need to detect a rising or falling edge with any Variable, Input, or Output.

This is the TYPICAL way to use the Positive or Negative Pulse Contact with physical inputs in Ladder Diagram programs.

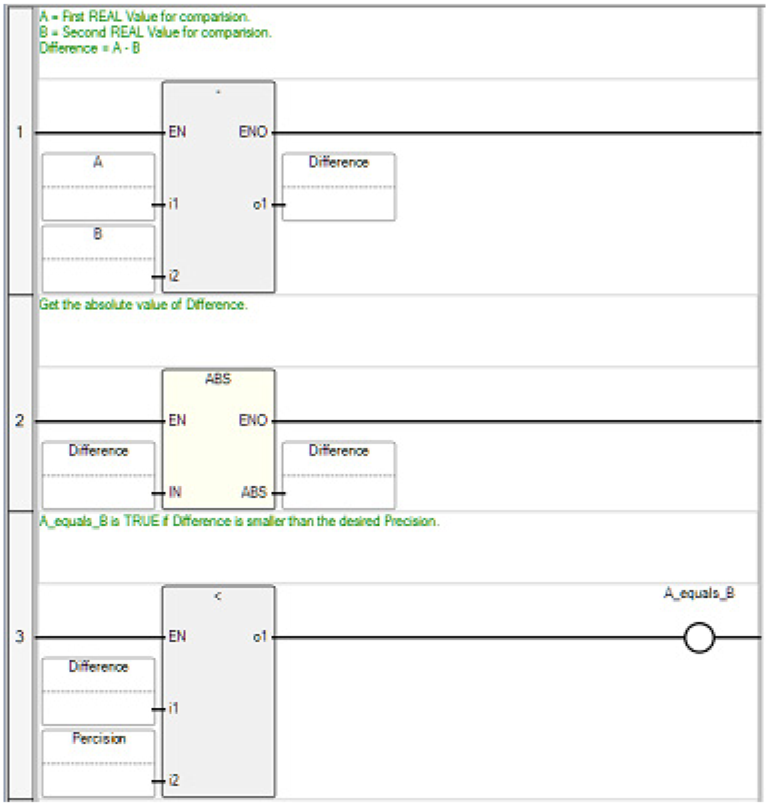
Using Outputs or Variables with a Positive or Negative Edge Pulse Contact is NOT recommended but works if the Output or Variable is located AFTER the rung which is triggering the edge.

Example: Comparing Real Values using Subtraction (-) ABS, and Less than (<)

The Real data type is not recommended when comparing values for equality because of differences in the way numbers are rounded. Two output values may appear equal in a Connected Components Workbench display, but will evaluate as false.

For example, 23.500001 compared to 23.499999 will both display as 23.5 in the variable input display, but will not be equal in the controller.

To test whether two Real data type values are equal, you can use a Subtraction instruction to get the difference between the values and then determine if the difference is Less Than an established precision value. See the following LD program example for comparing two Real data type values.



A = First REAL Value for comparision.
B = Second REAL Value for comparision.
Difference = A - B

Get the absolute value of Difference.

A_equals_B is TRUE if Difference is smaller than the desired Precision.

LD Keyboard shortcuts

The following keyboard shortcuts are available for use with the Ladder Diagram language on page 29.

Shortcut	Description
Ctrl+0	Inserts a rung after a selected rung.[1]
Ctrl+Alt+0	Inserts a rung before a selected rung.[1]
Ctrl+ 1	Inserts a branch after a selected element.
Ctrl+Alt+ 1	Inserts a branch before a selected element.
Ctrl+2	Inserts an instruction block after a selected element.[2]
Ctrl+Alt+2	Inserts an instruction block before a selected element.[2]
Ctrl+3	Inserts a contact after a selected element.[2]

Shortcut	Description
Ctrl+Alt+3	Inserts a contact before a selected element.[2]
Ctrl+4	Inserts a coil after a selected element.
Ctrl+5	Inserts a jump after a selected element.
Ctrl+Alt+5	Inserts a jump after a selected element.
Ctrl+6	Inserts a return after a selected element.
Ctrl+8	Inserts a branch above the selected branch.
Ctrl+Alt+8	Inserts a branch below the selected branch.
Delete	Removes a selected rung or element.
Enter	When a rung is selected, pressing the Enter key selects the first element of the rung. If there is no rung element, nothing happens.
Spacebar	When a coil or contact is selected, pressing the Spacebar changes the contact or coil type.
Shift+Enter	Inserts a line break.
Ctrl+Enter	Opens a line above the current line.
Ctrl+Shift+Enter	Opens a line below the current line.
Ctrl+Shift+L	Removes the current line.
Ctrl+Delete	Removes the next word in the current line.
Backspace	Removes the character on the left.
Ctrl+Backspace	Removes the previous word in the current line.
Ctrl+C	Copies the selected text to the clipboard.
Ctrl+Insert	Copies the selected text to the clipboard.
Ctrl+V	Pastes text saved on the clipboard to the insertion point.
Shift+Insert	Pastes text saved on the clipboard to the insertion point.
Ctrl+Z	Undoes the previous command.
Ctrl+Y	Redoes the previous command.
Ctrl+Shift+Z	Redoes the previous command.
Ctrl+Left	Moves to the previous statement or word.
Ctrl+Right	Moves to the next statement or word.
Home	Moves to the first element of the selected rung, if there is no rung element nothing happens.
End	Moves to the last element of the selected rung, if there is no rung element nothing happens.
Ctrl+Home	Moves to the first element of the first rung, if there is no rung element, the first rung is selected.
Ctrl+End	Moves to the last element of the last rung, if there is no rung element, the last rung is selected.
Page Up	Moves to the top of the visible code.
Page Down	Moves to the bottom of the visible code.
Ctrl+J	Moves to the matching bracket.
Ctrl+Down	Scrolls down.
Ctrl+Up	Scrolls up.
Shift+Down	Selects down.
Shift+Left Mouse Click	Selects multiple rungs. Click each rung individually.
Shift+Up	Selects up.
Shift+Left	Selects left.
Shift+Right	Selects right.

Shortcut	Description
Ctrl+Shift+Left	Selects to the previous statement or word.
Ctrl+Shift+Right	Selects to the next statement or word.
Shift+Home	Selects from the insertion point until the start of the line.
Shift+End	Selects from the insertion point until the end of the line.
Ctrl+Shift+Home	Selects from the insertion point until the start of the document.
Ctrl+Shift+End	Selects from the insertion point until the end of the document.
Shift+Page Up	Selects from the insertion point until the top of the visible code.
Shift+Page Down	Selects from the insertion point until the end of the visible code.
Ctrl+Shift+Page Up	Selects from the insertion point until the top of the visible code.
Ctrl+Shift+Page Down	Selects from the insertion point until the end of the visible code.
Ctrl+A	Selects the entire document.
Ctrl+D	When a rung or one element of the rung is selected, after pressing CTRL+D user can edit rung comment.
Ctrl+R	Enable or disable the Automatic Selector Invocation. By default, either the Instruction Block Selector or Variable Selector dialog opens when an element is added to a Ladder Diagram program.
Ctrl+Shift+W	Selects the next word.
Ctrl+Shift+J	Selects to the matching bracket.
Shift+Alt+Down	Selects the current and next lines.
Shift+Alt+Up	Selects the current and previous lines.
Shift+Alt+Left	Selects left on the current line.
Shift+Alt+Right	Selects right on the current line.
Ctrl+Shift+Alt+ Left	Selects available columns in lines of code from the left to right.
Ctrl+Shift+Alt+Right	Selects available columns in lines of code from the right to left.
Esc	Deselects the selected text.
Insert	Toggles between the overwrite/insert typing mode.

[1] When no rung is selected, a rung is added at the end of the rung list.

[2] When a branch is selected, an element is inserted at the end of the branch.

Alarm instruction

Use the alarm instruction to provide alerts when a configured high or low limit is reached.

Function block	Description
LIM_ALRM on page 59	Hysteresis on a real value for high and low limits.

LIM_ALRM (limit alarm)

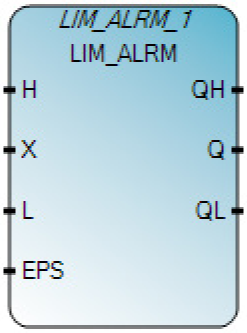
LIM_ALRM is an alarm with hysteresis on a Real value for high and low limits.

A hysteresis is applied on high and low limits. The hysteresis delta used for either high or low limit is one half of the EPS parameter.

A Process alarm is an alarm that occurs when a fault is received and processed by the controller. Process level alarms provide an alert when the module has exceeded the configured high or configured low limits for each channel.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.

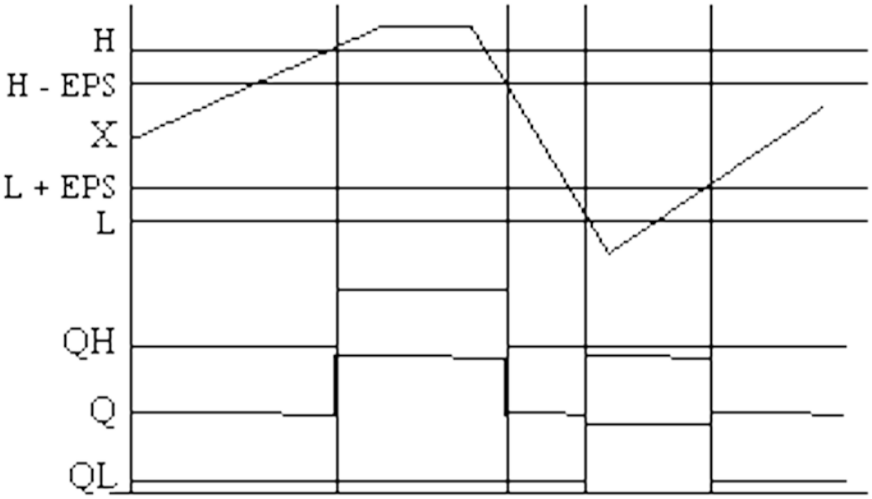


Use this table to help determine the parameter values for this instruction.

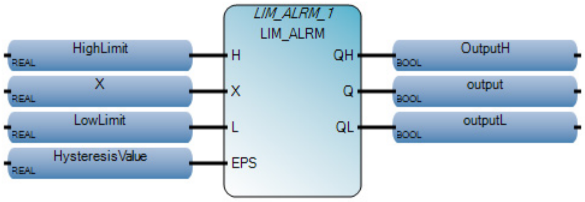
Parameter	Parameter type	Data type	Description
EN	Input	BOOL	When TRUE, enables the instruction block. TRUE - execute current LIM_ALRM computation. FALSE - there is no computation. Applies only to Ladder Diagram programs.
H	Input	REAL	High limit value.
X	Input	REAL	Input is any real value.
L	Input	REAL	Low limit value.
EPS	Input	REAL	Hysteresis value (must be greater than zero).
QH	Output	BOOL	High alarm: TRUE if X above high limit H.

Q	Output	BOOL	Alarm output: TRUE if X out of limits.
QL	Output	BOOL	Low alarm: TRUE if X below low limit L.
ENO	Output	BOOL	Enables outputs. Applies only to Ladder Diagram programs.

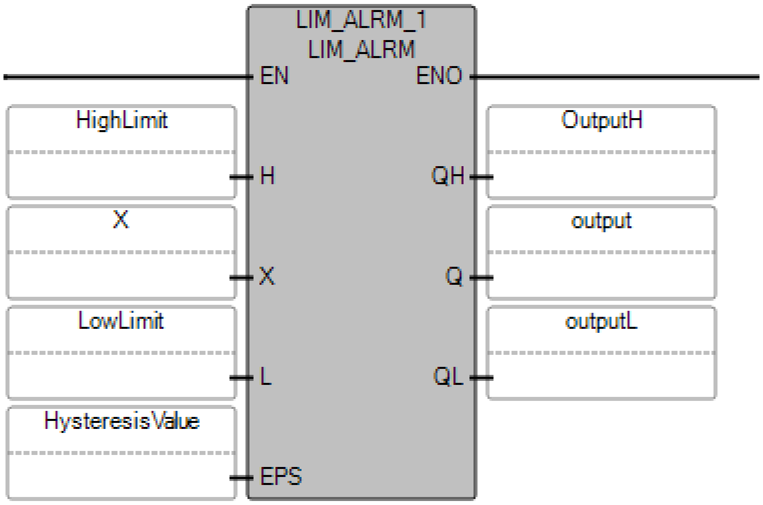
LIM_ALRM timing diagram example



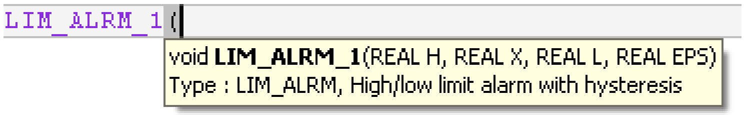
LIM_ALRM Function Block Diagram example



LIM_ALRM Ladder Diagram example

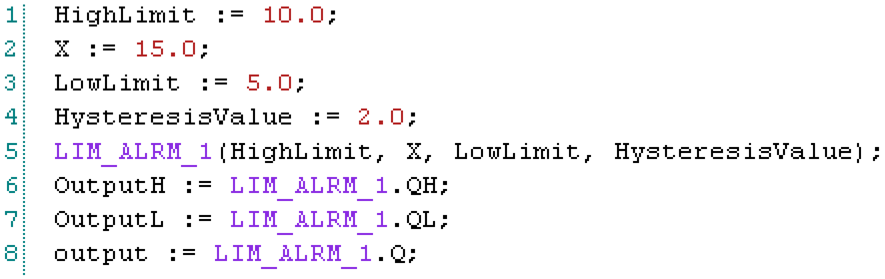


LIM_ALRM Structured Text example



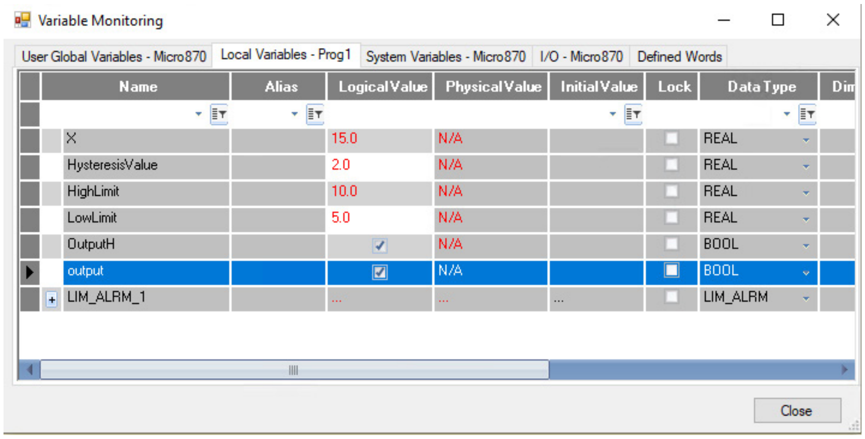
```
LIM_ALRM_1(
```

void **LIM_ALRM_1**(REAL H, REAL X, REAL L, REAL EPS)
Type : LIM_ALRM, High/low limit alarm with hysteresis



```
HighLimit := 10.0;
X := 15.0;
LowLimit := 5.0;
HysteresisValue := 2.0;
LIM_ALRM_1(HighLimit, X, LowLimit, HysteresisValue);
OutputH := LIM_ALRM_1.QH;
OutputL := LIM_ALRM_1.QL;
output := LIM_ALRM_1.Q;
```

Results



Name	Alias	Logical Value	Physical Value	Initial Value	Lock	Data Type	Dim
X		15.0	N/A		□	REAL	
HysteresisValue		2.0	N/A		□	REAL	
HighLimit		10.0	N/A		□	REAL	
LowLimit		5.0	N/A		□	REAL	
OutputH		☑	N/A		□	BOOL	
output		☑	N/A		□	BOOL	
+ LIM_ALRM_1		...	...	...	□	LIM_ALRM	

Arithmetic instructions

Use the arithmetic instructions to perform mathematical calculations.

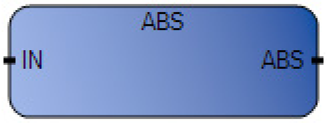
Function	Description
ABS on page 63	Returns the absolute value of a Real value.
ACOS on page 65	Calculates the arc-cosine of a Real value.
ACOS_LREAL on page 66	Calculates the arc-cosine of a Long Real value.
Addition on page 68	Adds two or more Integer, Real, Time, or String values.
ASIN on page 69	Calculates the arcsine of a Real value.
ASIN_LREAL on page 71	Calculates the arcsine of a Long Real value.
ATAN on page 72	Calculates the arctangent of a Real value.
ATAN_LREAL on page 74	Calculates the arctangent of a Long Real value.
COS on page 75	Calculates the cosine of a Real value.
COS_LREAL on page 77	Calculates the cosine of a Long Real value.
Division on page 78	Division of two Integer or Real values.
EXPT on page 80	Calculates the Real value of a base number raised to the power of the Integer exponent.
LOG on page 82	Calculates the logarithm (base 10) of a Real value.
MOD on page 83	Performs a Modulo calculation on Integer values.
MOV on page 85	Copies an input value to an output.
Multiplication on page 86	Multiplies two or more Integer or Real values.
Neg on page 88	Converts a value to a negative.
POW on page 89	Calculates the value of a Real number raised to a power of the Real exponent.
RAND on page 91	Calculates random integer values from a defined range.
SIN on page 93	Calculates the sine of a Real value.
SIN_LREAL on page 94	Calculates the sine of a Long Real value.
SQRT on page 96	Calculates the square root of a Real value.
Subtraction on page 97	Subtracts one Integer, Real or Time value from another Integer, Real or Time value.
TAN on page 99	Calculates the tangent of a Real value.
TAN_LREAL on page 100	Calculates the tangent of a Long Real value.
TRUNC on page 102	Truncates Real values, leaving just the Integer.

ABS (absolute value)

Returns the absolute (positive) value of a Real value.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

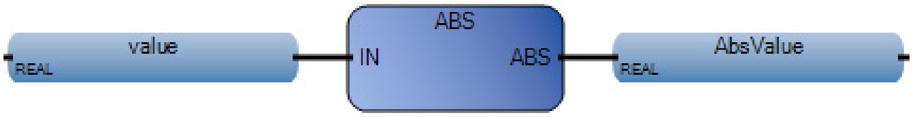
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



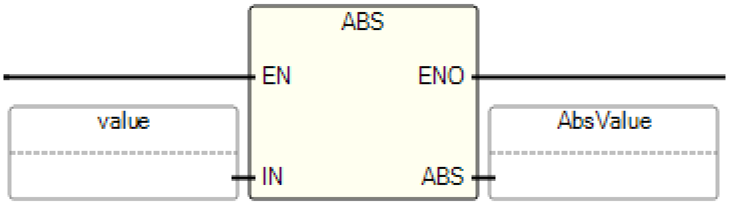
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute current absolute computation. FALSE - there is no computation. Applies to Ladder Diagram programs.
IN	Input	REAL	Any signed Real value.
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.
ABS	Output	REAL	Absolute value (always positive).

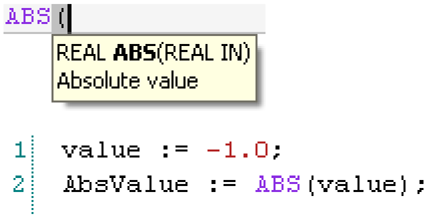
ABS Function Block Diagram example



ABS Ladder Diagram example



ABS Structured Text diagram example



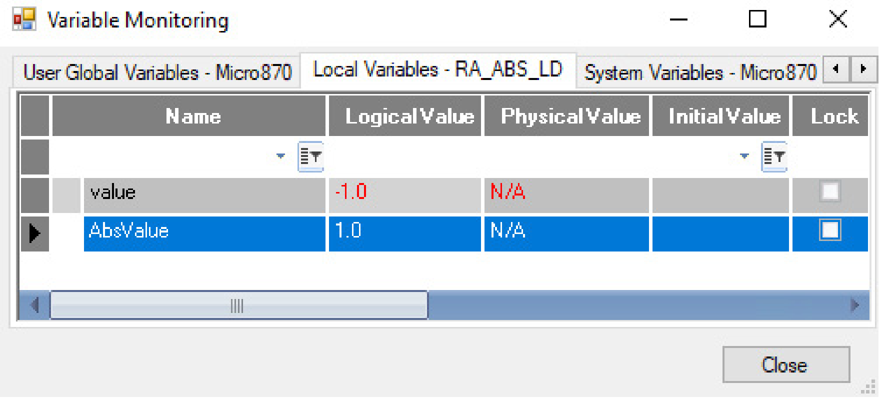
```
ABS(
REAL ABS(REAL IN)
Absolute value
```

```
value := -1.0;
AbsValue := ABS(value);
```

(* ST Equivalence: *)

over := (ABS (delta) > range);

Results



Variable Monitoring

User Global Variables - Micro870 | Local Variables - RA_ABS_LD | System Variables - Micro870

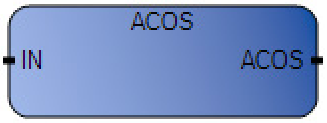
Name	Logical Value	Physical Value	Initial Value	Lock
value	-1.0	N/A		□
AbsValue	1.0	N/A		□

ACOS (arccosine of source)

Calculates the arc-cosine of a Real value. Input and output values are in radians.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

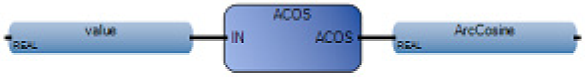
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



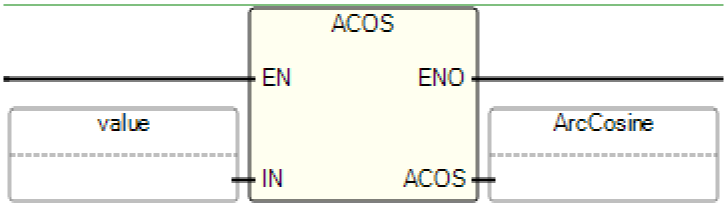
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute current arc-cosine computation. FALSE - there is no computation. Applies to Ladder Diagram programs.
IN	Input	REAL	Must be in set [-1.0 .. +1.0].
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.
ACOS	Output	REAL	Arc-cosine of the input value (in set [-p1/2..+p1/2])=0 for invalid input.

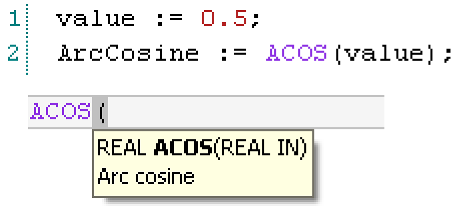
ACOS Function Block Diagram example



ACOS Ladder Diagram example



ACOS Structured Text example



```
value := 0.5;
ArcCosine := ACOS(value);
```

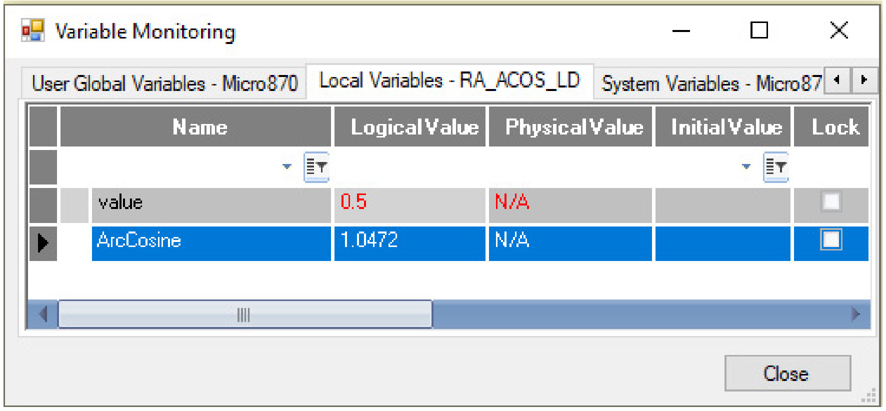
```
ACOS(
REAL ACOS(REAL IN)
Arc cosine
```

(* ST Equivalence: *)

cosine := COS (angle);

result := ACOS (cosine); (* result is equal to angle *)

Results


Variable Monitoring
User Global Variables - Micro870
Local Variables - RA_ACOS_LD
System Variables - Micro87
Close

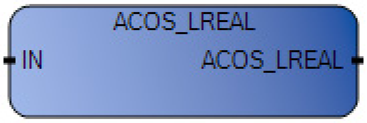
Name	Logical Value	Physical Value	Initial Value	Lock
value	0.5	N/A		□
ArcCosine	1.0472	N/A		□

ACOS_LREAL (arccosine Long Real)

Calculates the arc-cosine of a Long Real value.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

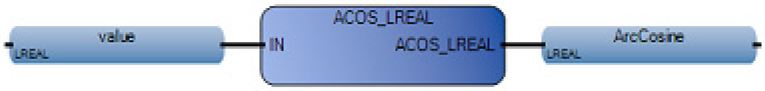
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



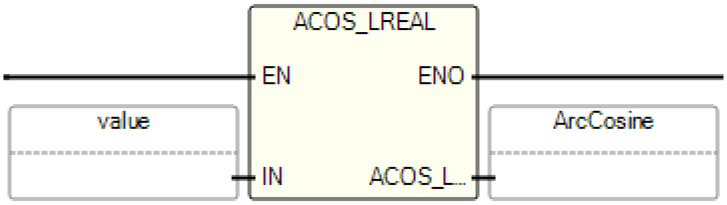
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute current computation. FALSE - there is no computation. Applies to Ladder Diagram programs.
IN	Input	LREAL	Must be in set [-1.0 .. +1.0].
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.
ACOS_LREAL	Output	LREAL	Arc-cosine of the input value (in set [0.0 .. PI]) = 0.0 for invalid input.

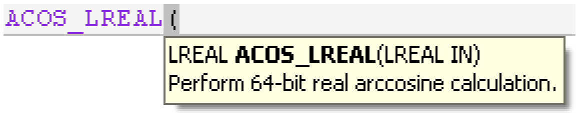
ACOS_LREAL Function Block Diagram example



ACOS_LREAL Ladder Diagram examples

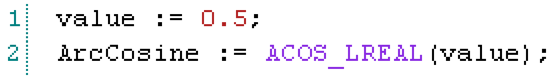


ACOS_LREAL Structured Text examples



```
ACOS_LREAL(
```

LREAL **ACOS_LREAL**(LREAL IN)
Perform 64-bit real arccosine calculation.



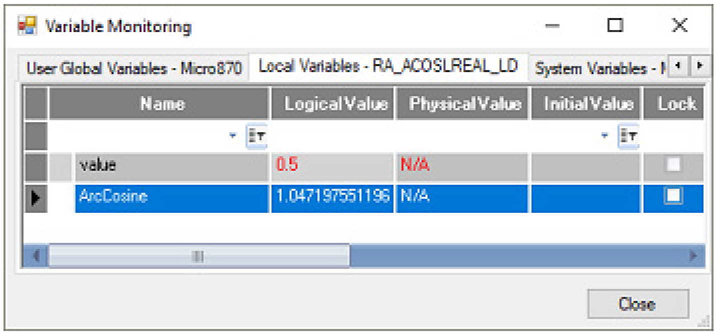
```
value := 0.5;
ArcCosine := ACOS_LREAL(value);
```

(* ST Equivalence: *)

cosine := COS_LREAL (angle);

result := ACOS_LREAL (cosine); (* result is equal to angle *)

Results



Variable Monitoring

User Global Variables - Micro870 | Local Variables - RA_ACOSLREAL_LD | System Variables - I

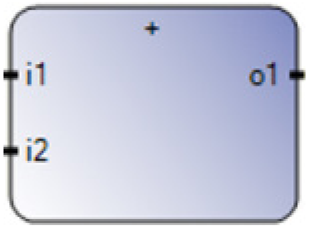
Name	Logical Value	Physical Value	Initial Value	Lock
value	0.5	N/A		□
ArcCosine	1.047197551196	N/A		□

Addition

Adds two or more Integer, Real, Time, or String values.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute current addition computation. FALSE - there is no computation. Applies only to Ladder Diagram programs.
i1	Input	SINT USINT BYTE INT UINT WORD DINT UDINT DWORD LINT ULINT LWORD REAL LREAL TIME STRING	Addend in Real, Time, or String data type. All inputs must be the same data type.

i2	Input	SINT USINT BYTE INT UINT WORD DINT UDINT DWORD LINT ULINT LWORD REAL LREAL TIME STRING	Addend in Real, Time, or String data type. All inputs must be the same data type.
o1	Output	SINT USINT BYTE INT UINT WORD DINT UDINT DWORD LINT ULINT LWORD REAL LREAL TIME STRING	Sum of the input values in Real, Time, or String format. Input and output must use the same data type.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.

Addition Structured Text example

(* ST equivalence: *)

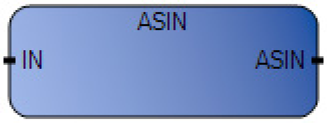
```
ao10 := ai101 + ai102;
ao5 := (ai51 + ai52) + ai53;
```

ASIN (arcsine)

Calculates the arcsine of a Real value. Input and output values are in radians.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

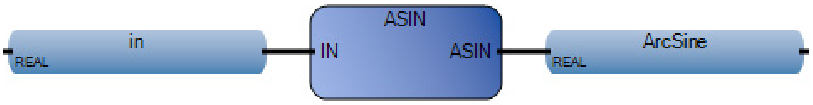
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



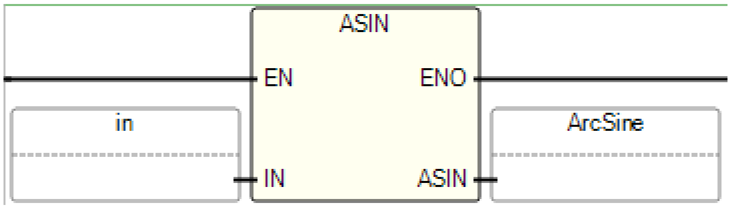
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute current arcsine computation. FALSE - there is no computation. Applies to Ladder Diagram programs.
IN	Input	REAL	Must be in set [-1.0 .. +1.0].
ASIN	Output	REAL	Arcsine of the input value(in set [-p1/2..+p1/2])=0 for invalid input.
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

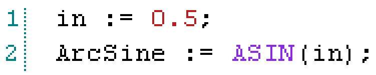
ASIN Function Block Diagram example



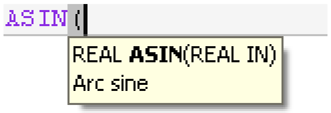
ASIN Ladder Diagram example



ASIN Structured Text example



```
in := 0.5;
ArcSine := ASIN(in);
```



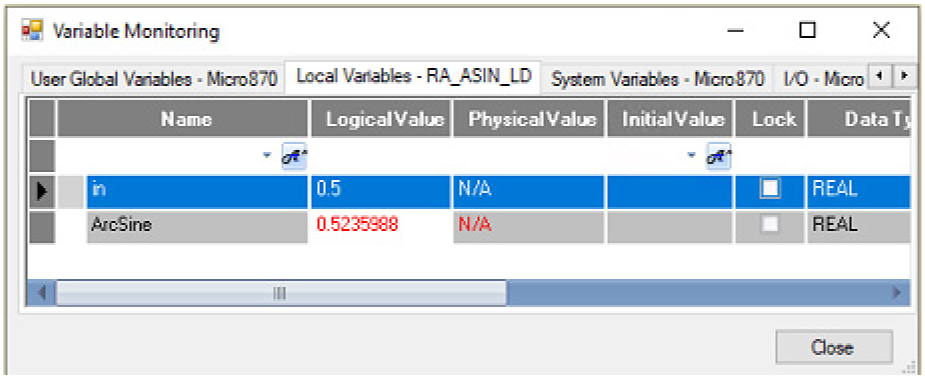
```
ASIN(
REAL ASIN(REAL IN)
Arc sine
```

(* ST Equivalence: *)

sine := SIN (angle);

result := ASIN (sine); (* result is equal to angle *)

Results


Variable Monitoring
User Global Variables - Micro870
Local Variables - RA_ASIN_LD
System Variables - Micro870
I/O - Micro
Close

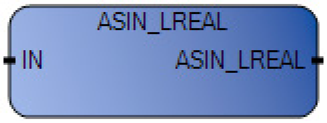
Name	Logical Value	Physical Value	Initial Value	Lock	Data Ty
in	0.5	N/A		☐	REAL
ArcSine	0.5235988	N/A		☐	REAL

ASIN_LREAL (arcsine Long Real)

Calculates the arcsine of a Long Real value.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

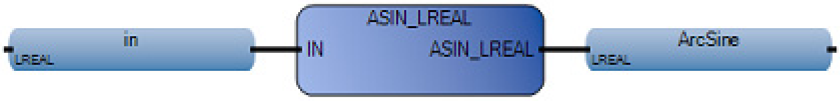
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



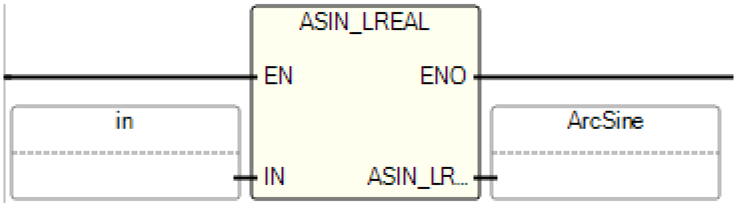
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute current computation. FALSE - there is no computation. Applies to Ladder Diagram programs.
IN	Input	LREAL	Must be in set [-1.0 .. +1.0].
ASIN_LREAL	Output	LREAL	Arcsine of the input value (in set [-PI/2 .. +PI/2]) = 0.0 for invalid input.
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

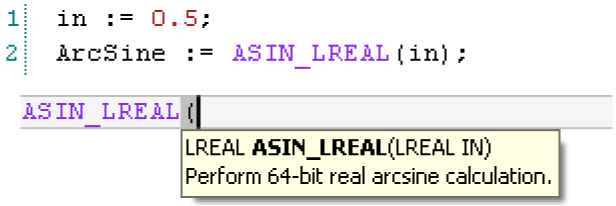
ASIN_LREAL Function Block Diagram example



ASIN_LREAL Ladder Diagram example



ASIN_LREAL Structured Text example



```
in := 0.5;
ArcSine := ASIN_LREAL(in);
```

```
ASIN_LREAL(
```

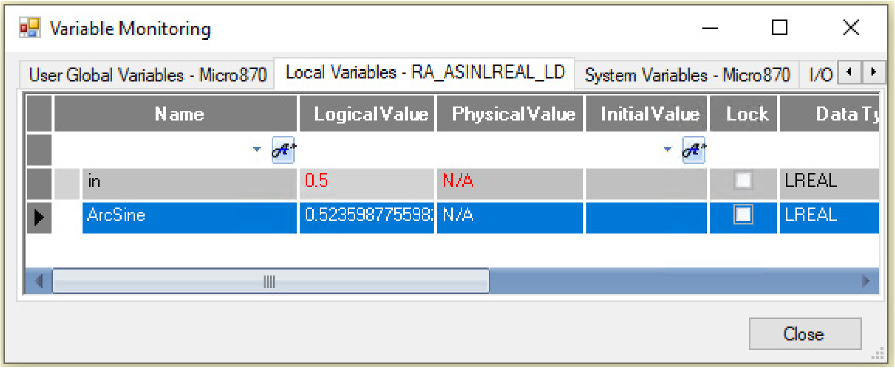
LREAL **ASIN_LREAL**(LREAL IN)
Perform 64-bit real arcsine calculation.

(* ST Equivalence: *)

sine := SIN_LREAL (angle);

result := ASIN_LREAL (sine); (* result is equal to angle *)

Results


Variable Monitoring
User Global Variables - Micro870
Local Variables - RA_ASINLREAL_LD
System Variables - Micro870
I/O
Close

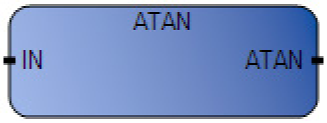
Name	Logical Value	Physical Value	Initial Value	Lock	Data Ty
in	0.5	N/A		□	LREAL
ArcSine	0.523598775598	N/A		□	LREAL

ATAN (arctangent)

Calculates the arctangent of a Real value.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

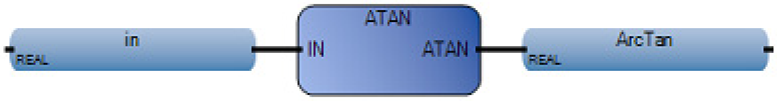
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



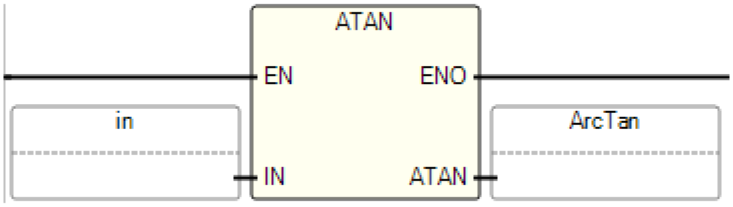
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute current arctangent computation. FALSE - there is no computation. Applies to Ladder Diagram programs.
IN	Input	REAL	Any Real value.
ATAN	Output	REAL	Arctangent of the input value (in set [-PI/2 .. +PI/2]) = 0.0 for invalid input.
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

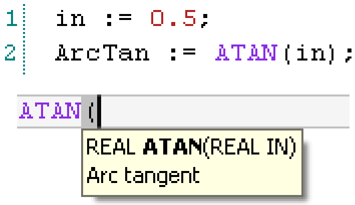
ATAN Function Block Diagram example



ATAN Ladder Diagram example



ATAN Structured Text example



```
in := 0.5;
ArcTan := ATAN(in);
```

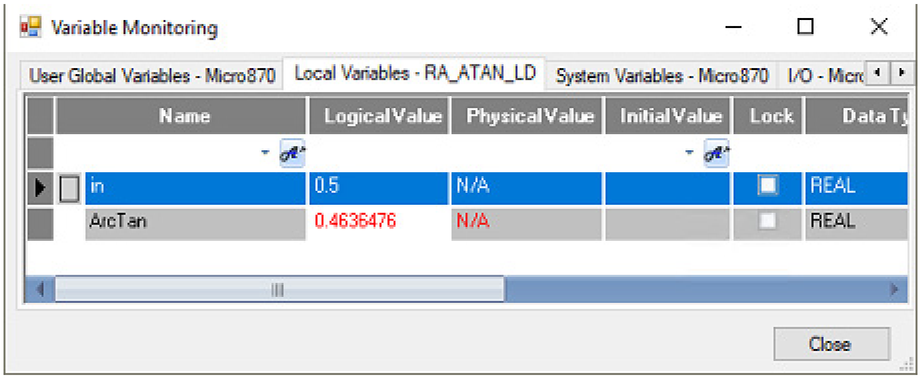
```
ATAN(
REAL ATAN(REAL IN)
Arc tangent
```

(* ST Equivalence: *)

tangent := TAN (angle);

result := ATAN (tangent); (* result is equal to angle*)

Results


Variable Monitoring
User Global Variables - Micro870
Local Variables - RA_ATAN_LD
System Variables - Micro870
I/O - Micro
Close

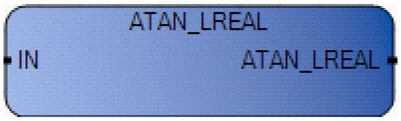
Name	Logical Value	Physical Value	Initial Value	Lock	Data Ty
in	0.5	N/A		□	REAL
ArcTan	0.4636476	N/A		□	REAL

ATAN_LREAL (arctangent Long Real)

Calculates the arctangent of a Long Real value.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

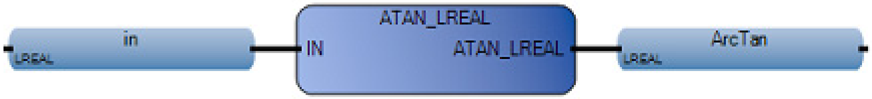
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



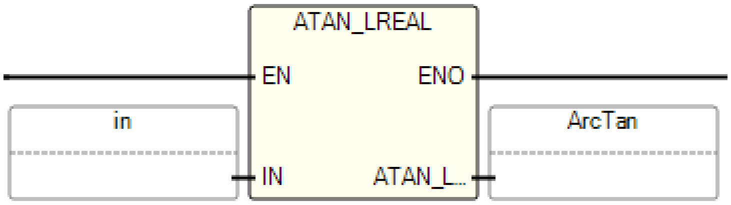
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute current computation. FALSE - there is no computation. Applies to Ladder Diagram programs.
IN	Input	LREAL	Any Long Real value.
ATAN_LREAL	Output	LREAL	Arctangent of the input value (in set [-PI/2 .. +PI/2]) = 0.0 for invalid input.
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

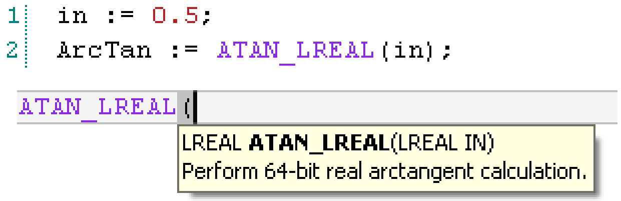
ATAN_LREAL Function Bock Diagram example



ATAN_LREAL Ladder Diagram example



ATAN_LREAL Structured Text example



```
in := 0.5;
ArcTan := ATAN_LREAL(in);
```

```
ATAN_LREAL(
```

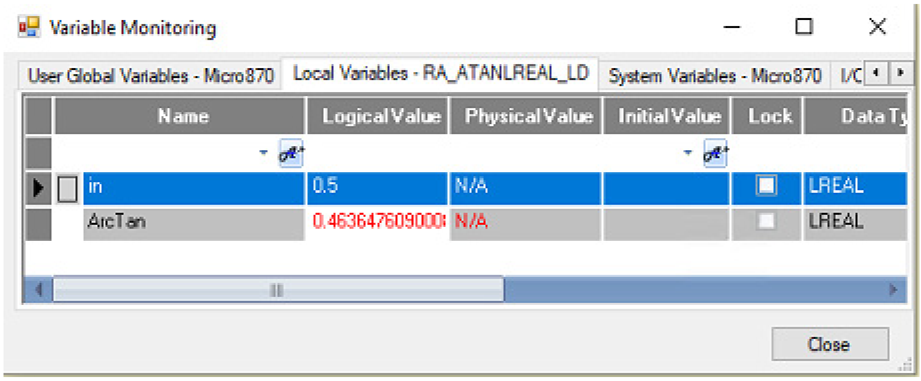
LREAL **ATAN_LREAL**(LREAL IN)
Perform 64-bit real arctangent calculation.

(* ST Equivalence: *)

tangent := TAN_LREAL (angle);

result := ATAN_LREAL (tangent); (* result is equal to angle*)

Results


Variable Monitoring
User Global Variables - Micro870
Local Variables - RA_ATANLREAL_LD
System Variables - Micro870
I/C
Close

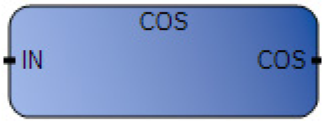
Name	Logical Value	Physical Value	Initial Value	Lock	Data Ty
in	0.5	N/A		□	LREAL
ArcTan	0.463647609000	N/A		□	LREAL

COS (cosine)

Calculates the cosine of a Real value.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

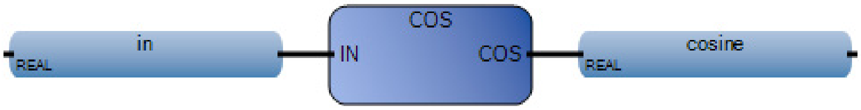
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



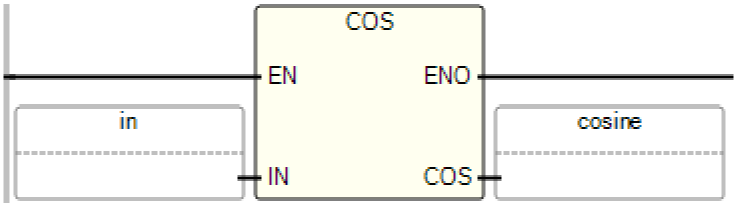
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute current cosine computation. FALSE - there is no computation. Applies to Ladder Diagram programs.
IN	Input	REAL	Any Real value.
COS	Output	REAL	Cosine of the input value (in set [-1.0 .. +1.0]).
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

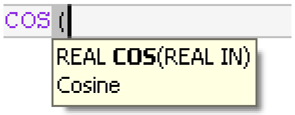
COS Function Block Diagram example



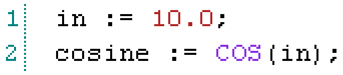
COS Ladder Diagram example



COS Structured Text example



```
COS (
REAL COS(REAL IN)
Cosine
```



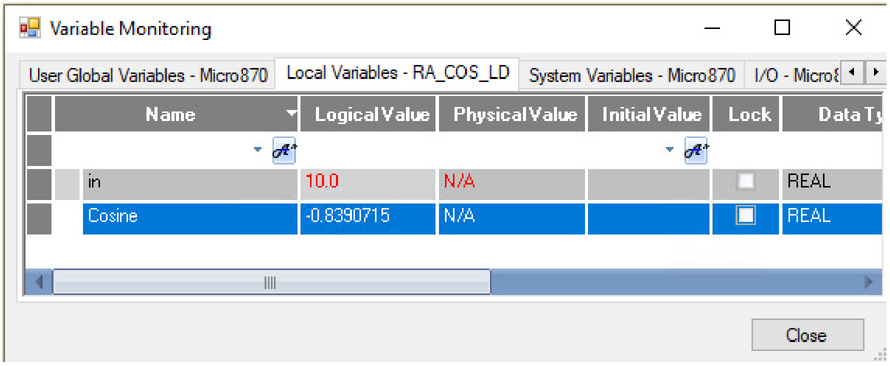
```
in := 10.0;
cosine := COS(in);
```

(* ST Equivalence: *)

cosine := COS (angle);

result := ACOS (cosine); (* result is equal to angle *)

Results



Variable Monitoring

User Global Variables - Micro870 | Local Variables - RA_COS_LD | System Variables - Micro870 | I/O - Micro8

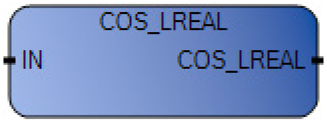
Name	Logical Value	Physical Value	Initial Value	Lock	Data Ty
in	10.0	N/A		□	REAL
Cosine	-0.8390715	N/A		□	REAL

COS_LREAL (cosine Long Real)

Calculates the cosine of a Long Real value.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

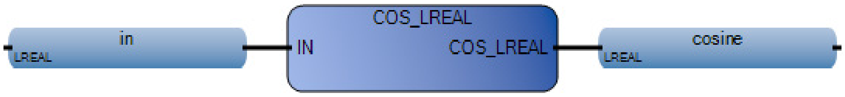
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



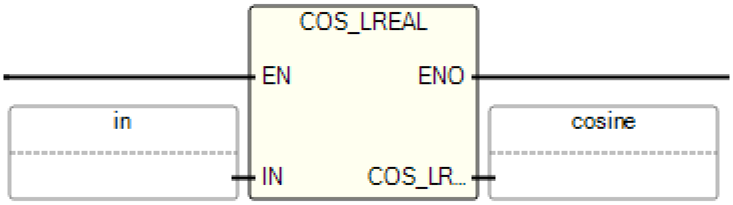
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute current cosine computation. FALSE - there is no computation. Applies to Ladder Diagram programs.
IN	Input	LREAL	Any Long Real value.
COS_LREAL	Output	LREAL	Cosine of the input value (in set [-1.0 .. +1.0]).
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

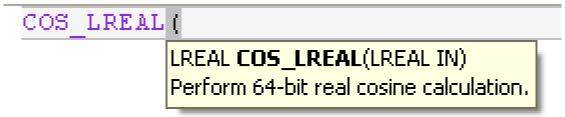
COS_LREAL Function Block Diagram example



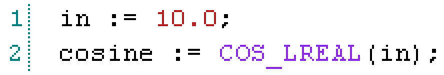
COS_LREAL Ladder Diagram example



COS_LREAL Structured Text example



```
COS_LREAL(
LREAL COS_LREAL(LREAL IN)
Perform 64-bit real cosine calculation.
```



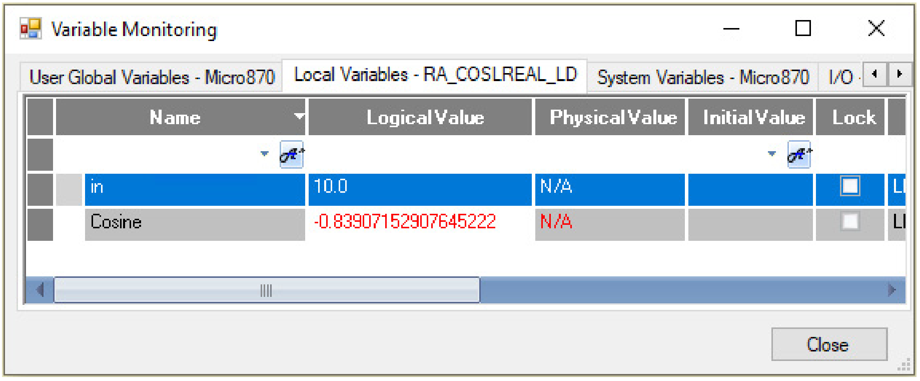
```
in := 10.0;
cosine := COS_LREAL(in);
```

(* ST Equivalence: *)

cosine := COS_LREAL (angle);

result := ACOS_LREAL (cosine); (* result is equal to angle *)

Results



Variable Monitoring

User Global Variables - Micro870 | Local Variables - RA_COSLREAL_LD | System Variables - Micro870 | I/O

Name	Logical Value	Physical Value	Initial Value	Lock	
in	10.0	N/A		□	LI
Cosine	-0.83907152907645222	N/A		□	LI

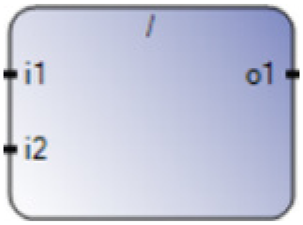
Close

Division

Divides the first Integer or Real input value by the second Integer or Real input value.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute current division computation. FALSE - there is no computation. Applies only to Ladder Diagram programs.
i1	Input	SINT USINT BYTE INT UINT WORD DINT UDINT DWORD LINT ULINT LWORD REAL LREAL	Dividend in non-zero Integer or Real data type. All inputs must be the same data type.
i2	Input	SINT USINT BYTE INT UINT WORD DINT UDINT DWORD LINT ULINT LWORD REAL LREAL	Divisor in non-zero Integer or Real data type. All inputs must be the same data type.

o1	Output	SINT USINT BYTE INT UINT WORD DINT UDINT DWORD LINT ULINT LWORD REAL LREAL	Quotient of the inputs in non-zero Integer or Real data type. Input and output must use the same data type.
ENO	Output	BOOL	Enable out. Applies only to Ladder Diagram programs.

Division Structured Text example

(* ST Equivalence: *)

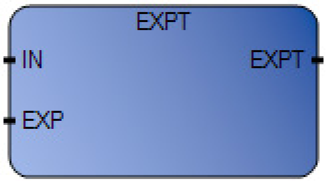
```
ao10 := ai101 / ai102;
ao5 := (ai5 / 2) / ai53;
```

EXPT (exponent)

Raises the value of IN (base) to the power of EXP (exponent) and outputs the Real result of the operation.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

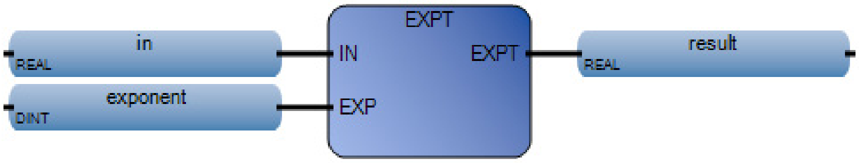
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



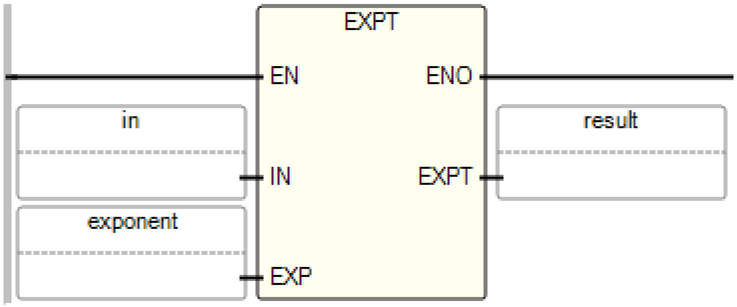
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute current exponent computation. FALSE - there is no computation.
IN	Input	REAL	Any signed Real value.
EXP	Input	DINT	Integer exponent.
EXPT	Output	REAL	The Real value of IN to the power of EXP.
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

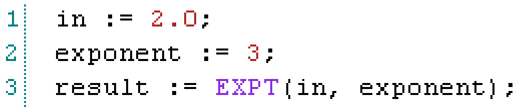
EXPT Function Block Diagram example



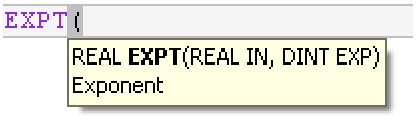
EXPT Ladder Diagram example



EXPT Structured Text example



```
in := 2.0;
exponent := 3;
result := EXPT(in, exponent);
```



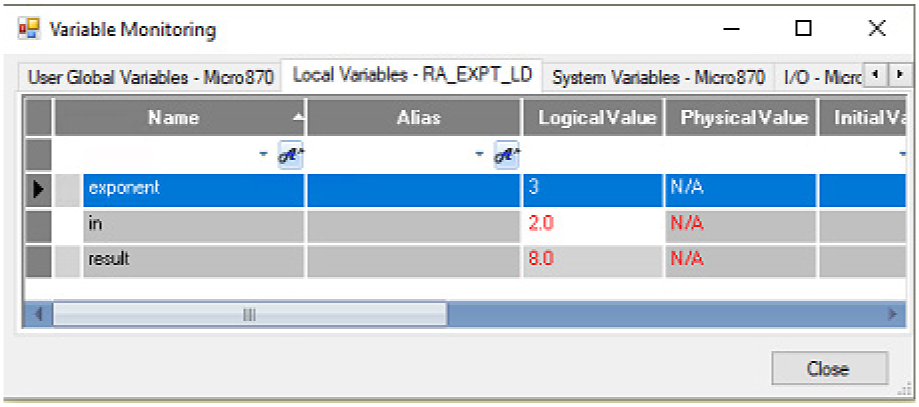
```
EXPT(
```

REAL **EXPT**(REAL IN, DINT EXP)
Exponent

(* ST Equivalence: *)

tb_size := ANY_TO_DINT (EXPT (2.0, range));

Results


Variable Monitoring
User Global Variables - Micro870
Local Variables - RA_EXPT_LD
System Variables - Micro870
I/O - Micro
Close

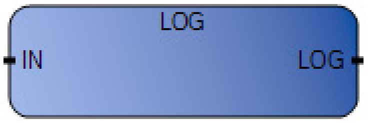
Name	Alias	Logical Value	Physical Value	Initial Va
exponent		3	N/A	
in		2.0	N/A	
result		8.0	N/A	

LOG (log base 10)

Calculates the logarithm (base 10) of a Real value.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

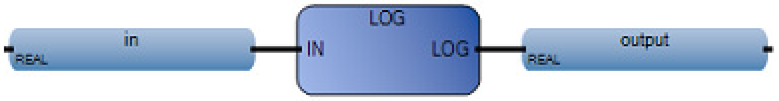
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



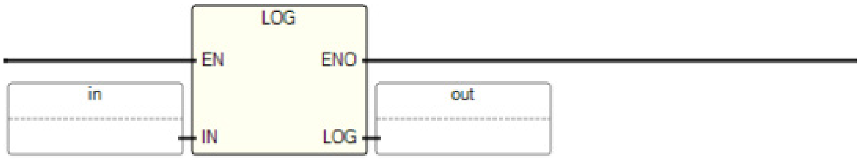
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute current logarithm computation. FALSE - there is no computation.
IN	Input	REAL	Must be greater than zero.
LOG	Output	REAL	Logarithm (base 10) of the input value. The returned result is -3.4E+38 for a zero IN value and negative IN value.
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

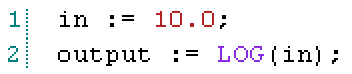
LOG Function Block Diagram example



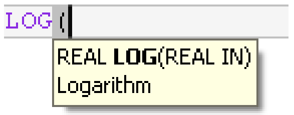
LOG Ladder Diagram example



LOG Structured Text example



```
in := 10.0;
output := LOG(in);
```



LOG(

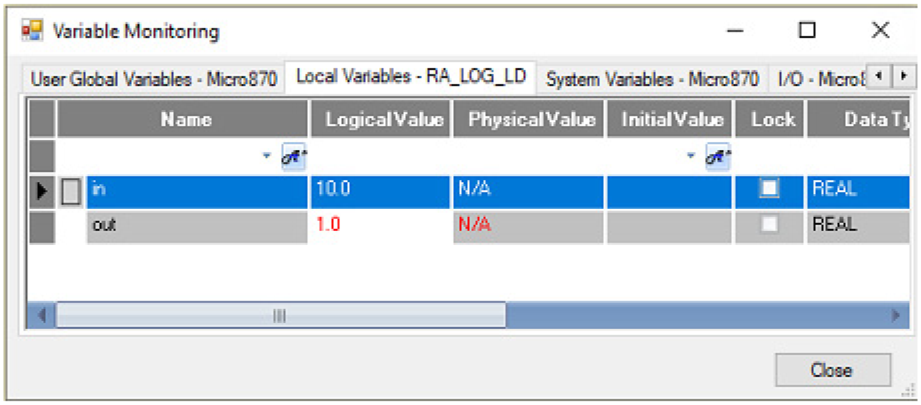
REAL **LOG**(REAL IN)
Logarithm

(* ST Equivalence: *)

xpos := ABS (xval);

xlog := LOG (xpos);

Results


Variable Monitoring
User Global Variables - Micro870
Local Variables - RA_LOG_LD
System Variables - Micro870
I/O - Micro8
Close

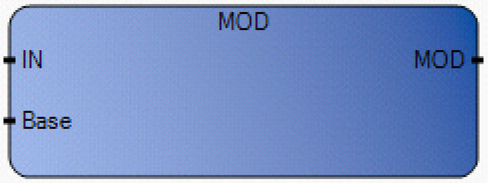
Name	Logical Value	Physical Value	Initial Value	Lock	Data Ty
in	10.0	N/A		☐	REAL
out	1.0	N/A		☐	REAL

MOD (modulo)

Divide the IN input by the Base input and place the remainder in the MOD output.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

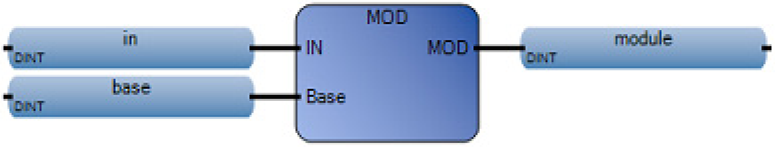
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



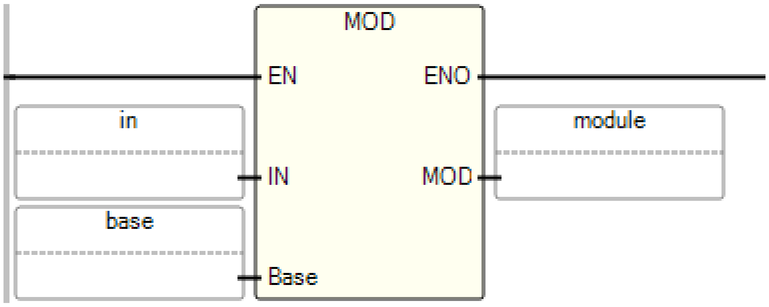
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute the module computation. FALSE - there is no computation. Applies to Ladder Diagram programs.
IN	Input	DINT	Any signed integer value.
Base	Input	DINT	Must be greater than zero.
MOD	Output	DINT	Modulo calculation (input MOD base) / returns -1 if Base <= 0.
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

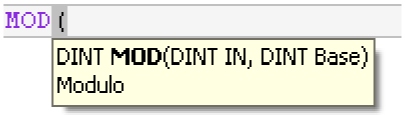
MOD Function Block Diagram example



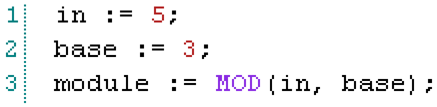
MOD Ladder Diagram example



MOD Structured Text example



```
MOD (
DINT MOD(DINT IN, DINT Base)
Modulo
```



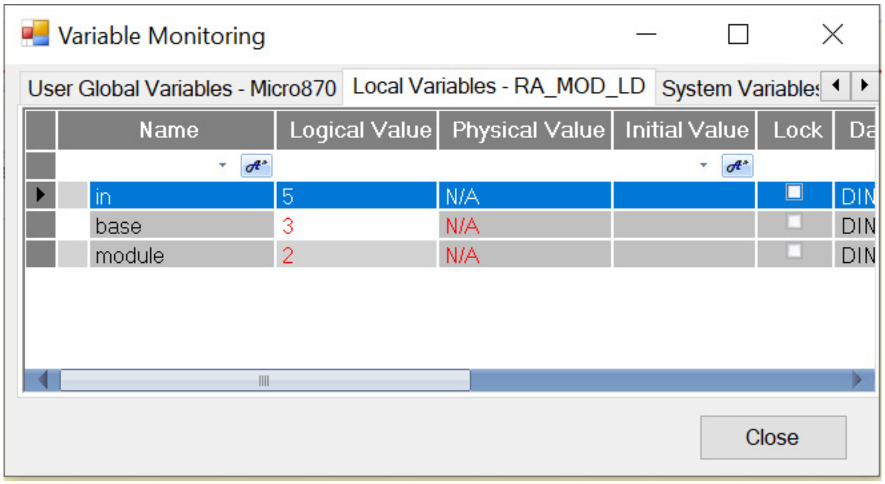
```
in := 5;
base := 3;
module := MOD(in, base);
```

(* ST Equivalence: *)

division_result := (value / divider); (* integer division *)

rest_of_division := MOD (value, divider); (* rest of the division *)

Results



Variable Monitoring

User Global Variables - Micro870 | Local Variables - RA_MOD_LD | System Variable

Name	Logical Value	Physical Value	Initial Value	Lock	Da
in	5	N/A		☐	DIN
base	3	N/A		☐	DIN
module	2	N/A		☐	DIN

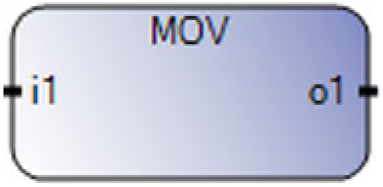
MOV (move)

Assigns the input (i1) value to the output (o1).

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

For Structured Text programs, use the Equal (=) operator instead of MOV.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute the direct link to an output computation. FALSE - there is no computation. Applies only to Ladder Diagram programs.

i1	Input	BOOL DINT REAL TIME STRING SINT USINT INT UINT UDINT LINT ULINT DATE LREAL BYTE WORD DWORD LWORD	Input and output must use the same data type.
o1	Output	BOOL DINT REAL TIME STRING SINT USINT INT UINT UDINT LINT ULINT DATE LREAL BYTE WORD DWORD LWORD	Input and output must use the same data type.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.

Structured Text example

(* ST equivalence: *)

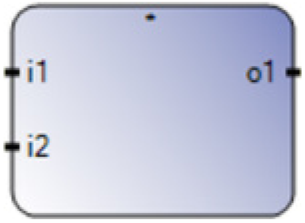
```
    ao23 := ai10;
```

Multiplication

Multiplies two or more Integer or Real values.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute current multiplication computation. FALSE - there is no computation. Applies only to Ladder Diagram programs.
i1	Input	SINT USINT BYTE INT UINT WORD DINT UDINT DWORD LINT ULINT LWORD REAL LREAL	Factor in Integer or Real data type. All inputs must be the same data type.
i2	Input	SINT USINT BYTE INT UINT WORD DINT UDINT DWORD LINT LINT LWORD REAL LREAL	Factor in Integer or Real data type. All inputs must be the same data type.

o1	Output	SINT USINT BYTE INT UINT WORD DINT UDINT DWORD LINT ULINT LWORD REAL LREAL	Product of the inputs in Integer or Real data type. Input and output must use the same data type.
ENO	Output	BOOL	Enable out. Applies only to Ladder Diagram programs.

Multiplication Structured Text example

(* ST equivalence *)

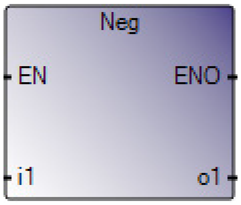
```
ao10 := ai101 * ai102;
ao5 := (ai51 * ai52) * ai53;
```

Neg (negation)

Converts a value to a negated value.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute current convert to negative computation. FALSE - there is no computation. Applies only to Ladder Diagram programs.
i1	Input	SINT INT DINT LINT REAL LREAL	Input and output must be the same data type.

o1	Output	SINT INT DINT LINT REAL LREAL	Input and output must be the same data type.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.

Neg Structured Text example

(* ST equivalence: *)

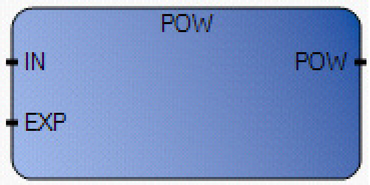
```
ao23 := - (ai10);
ro100 := - (ri1 + ri2);
```

POW (raise power)

When the first argument is 'base' and the second argument is 'exponent', calculate the Real result of (base exponent).

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

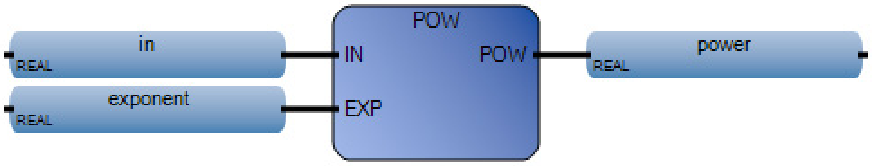
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



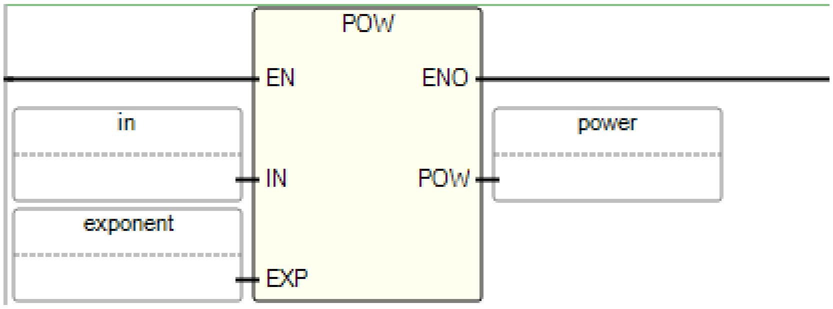
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute current exponent computation. FALSE - there is no computation. Applies to Ladder Diagram programs.
IN	Input	REAL	Real number to be raised.
EXP	Input	REAL	Power (exponent).
POW	Output	REAL	(IN EXP) 1.0 if IN is not 0.0 and EXP is 0.0 0.0 if IN is 0.0 and EXP is negative 0.0 if both IN and EXP are 0.0 0.0 if IN is negative and EXP does not correspond to an integer.
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

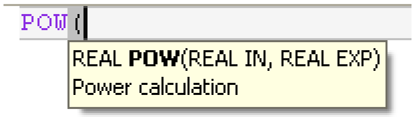
POW Function Block Diagram example



POW Ladder Diagram example

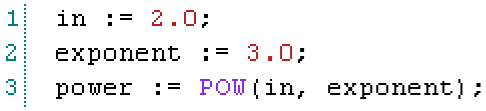


POW Structured Text example



```
POW(
```

REAL **POW**(REAL IN, REAL EXP)
Power calculation

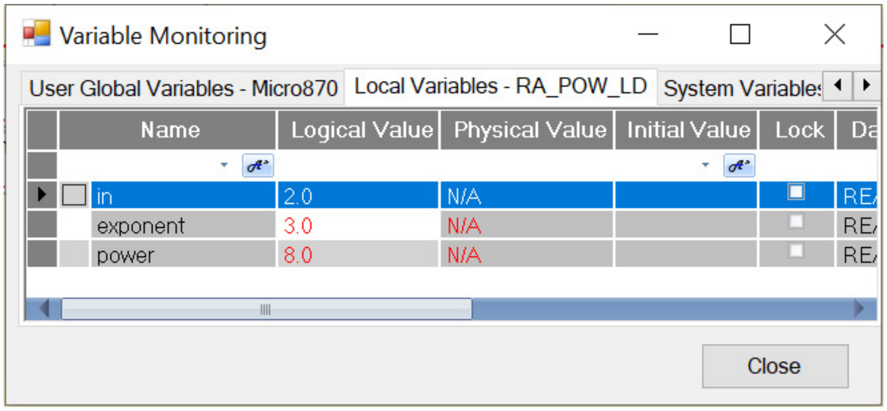


```
in := 2.0;
exponent := 3.0;
power := POW(in, exponent);
```

(* ST Equivalence: *)

result := POW (xval, power);

Results



Variable Monitoring

User Global Variables - Micro870 | Local Variables - RA_POW_LD | System Variables

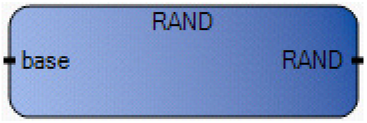
	Name	Logical Value	Physical Value	Initial Value	Lock	Da
▶ □	in	2.0	N/A		□	RE
	exponent	3.0	N/A		□	RE
	power	8.0	N/A		□	RE

RAND (random value)

Calculates random integer values from a defined range.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

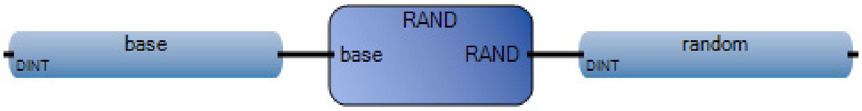
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



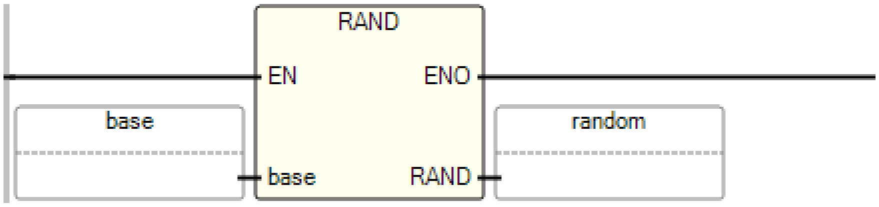
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute the random integer value computation. FALSE - there is no computation. Applies to Ladder Diagram programs.
base	Input	DINT	Defines the supported set of numbers.
RAND	Output	DINT	Random value in set [0..base-1].
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

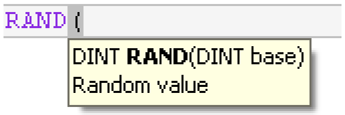
RAND Function Block Diagram example



RAND Ladder Diagram example

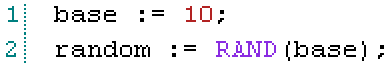


RAND Structured Text example



```
RAND (
```

DINT **RAND**(DINT base)
Random value



```
base := 10;
random := RAND(base);
```

(* ST Equivalence: *)

selected := MUX4 (RAND (4), 1, 4, 8, 16);

(*

random selection of 1 of 4 pre-defined values

the value issued of RAND call is in set [0..3],

so 'selected' issued from MUX4, will get 'randomly' the value

1 if 0 is issued from RAND,

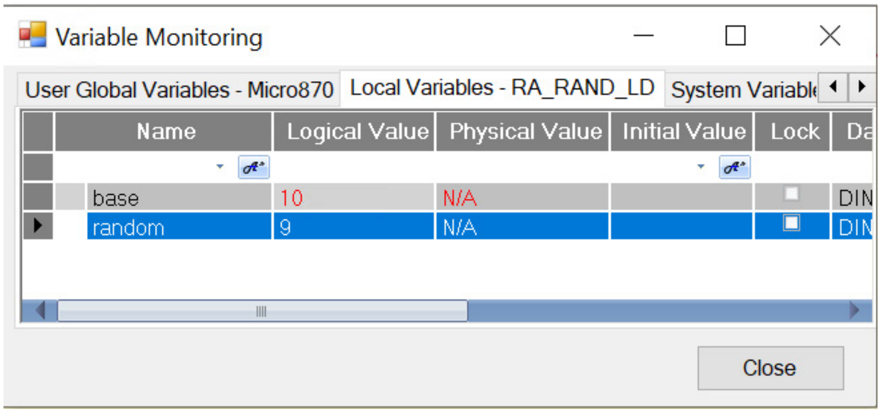
or 4 if 1 is issued from RAND,

or 8 if 2 is issued from RAND,

or 16 if 3 is issued from RAND,

*)

Results



Variable Monitoring

User Global Variables - Micro870 | Local Variables - RA_RAND_LD | System Variabl

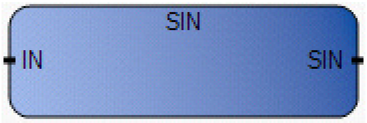
Name	Logical Value	Physical Value	Initial Value	Lock	Da
base	10	N/A		□	DIN
random	9	N/A		□	DIN

SIN (sine)

Calculates the sine of a Real value.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

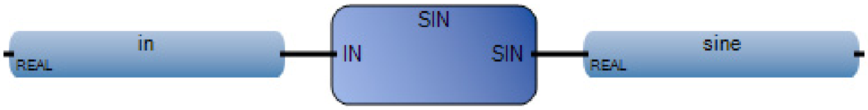
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



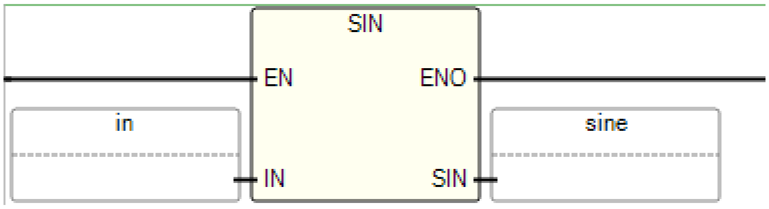
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute current sine computation. FALSE - there is no computation. Applies to Ladder Diagram programs.
IN	Input	REAL	Any Real value.
SIN	Output	REAL	Sine of the input value (in set [-1.0 .. +1.0]).
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

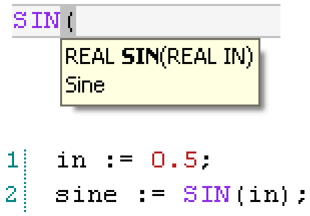
SIN Function Block Diagram example



SIN Ladder Diagram example



SIN Structured Text example



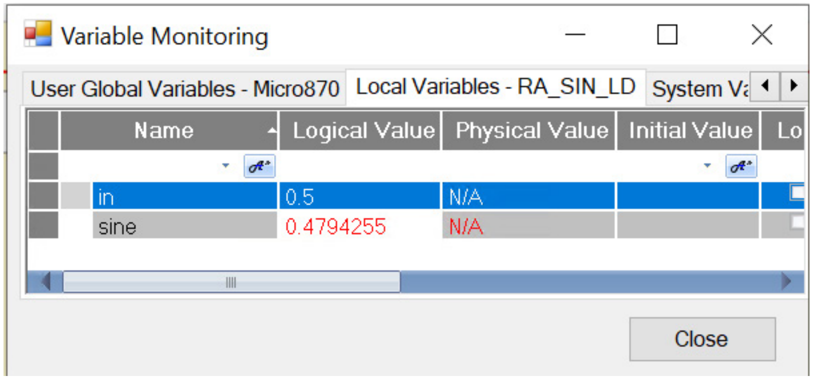
```
in := 0.5;
sine := SIN(in);
```

(* ST Equivalence: *)

sine := SIN (angle);

result := ASIN (sine); (* result is equal to angle *)

Results



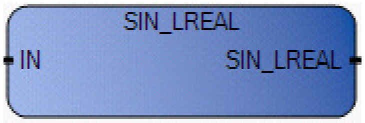
Name	Logical Value	Physical Value	Initial Value	Lo
in	0.5	N/A		
sine	0.4794255	N/A		

SIN_LREAL (sine Long Real)

Calculates the sine of a Long Real value.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

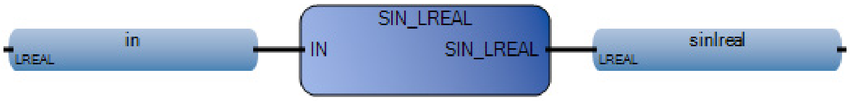
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



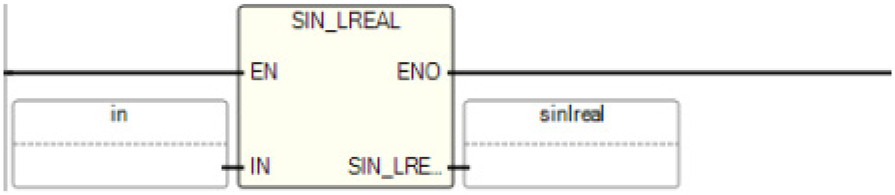
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute current computation. FALSE - there is no computation. Applies to Ladder Diagram programs.
IN	Input	LREAL	Any Long Real value.
SIN_LREAL	Output	LREAL	Sine of the input value (in set [-1.0 .. +1.0]).
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

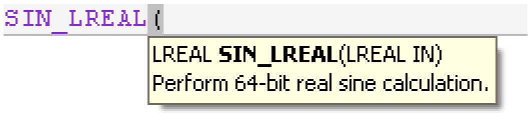
SIN_LREAL Function Block Diagram example



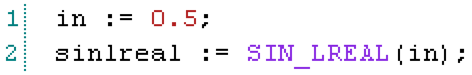
SIN_LREAL Ladder Diagram example



SIN_LREAL Structured Text example



```
SIN_LREAL(
LREAL SIN_LREAL(LREAL IN)
Perform 64-bit real sine calculation.
```

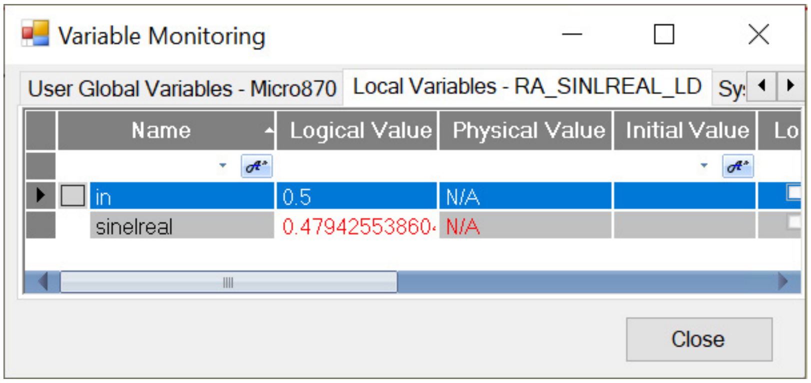


```
in := 0.5;
sinlreal := SIN_LREAL(in);
```

(* ST Equivalence: *)

TESTOUTPUT1 := SIN_LREAL(TESTINPUT1) ;

Results



Variable Monitoring

User Global Variables - Micro870 | Local Variables - RA_SINLREAL_LD | Sy:

Name	Logical Value	Physical Value	Initial Value	Lo
in	0.5	N/A		
sinelreal	0.47942553860‹	N/A		

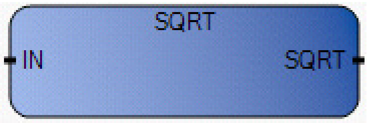
Close

SQRT (square root)

Calculates the square root of a Real value.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

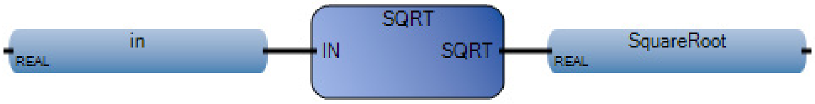
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



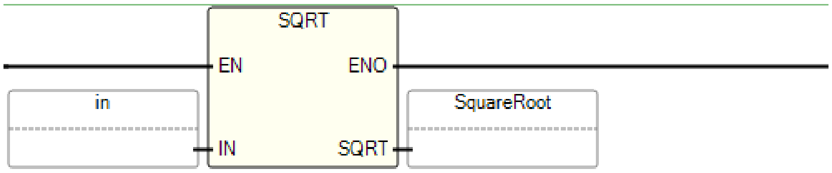
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute current square root computation. FALSE - there is no computation. Applies to Ladder Diagram programs.
IN	Input	REAL	Must be greater than or equal to zero.
SQRT	Output	REAL	Square root of the input value. The returned result is 0 for a negative IN value.
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

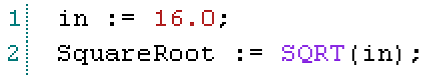
SQRT Function Block Diagram example



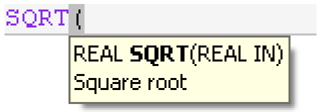
SQRT Ladder diagram example



SQRT Structured Text example



```
in := 16.0;
SquareRoot := SQRT(in);
```



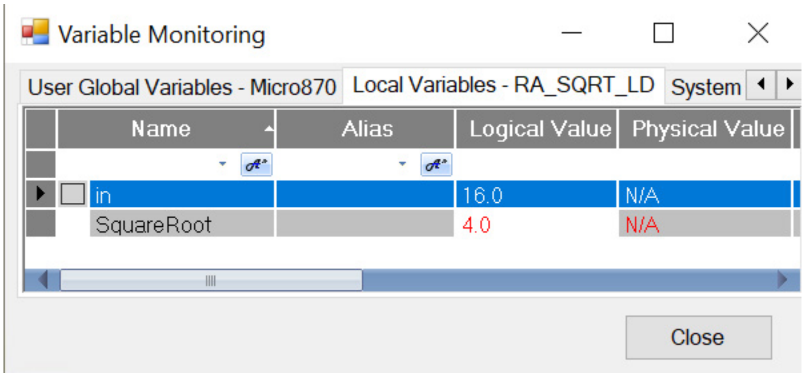
```
SQRT(
REAL SQRT(REAL IN)
Square root
```

(* ST Equivalence: *)

xpos := ABS (xval);

xroot := SQRT (xpos);

Results



Variable Monitoring

User Global Variables - Micro870 | Local Variables - RA_SQRT_LD | System

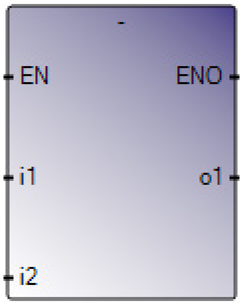
Name	Alias	Logical Value	Physical Value
in		16.0	N/A
SquareRoot		4.0	N/A

Subtraction

Subtracts one Integer, Real, or Time value from another Integer, Real or Time value.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description

EN	Input	BOOL	Instruction enable. TRUE - execute current addition computation. FALSE - there is no computation. Applies only to Ladder Diagram programs.
i1	Input	SINT USINT BYTE INT UINT WORD DINT UDINT DWORD LINT ULINT LWORD REAL LREAL TIME	Minuend in any Integer, Real or Time data type. All inputs must be the same data type.
i2	Input	SINT USINT BYTE INT UINT WORD DINT UDINT DWORD LINT ULINT LWORD REAL LREAL TIME	Subtrahend in any Integer, Real or Time data type. All inputs must be the same data type.
o1	Output	SINT USINT BYTE INT UINT WORD DINT UDINT DWORD LINT ULINT LWORD REAL LREAL TIME	Difference of the minuend and the subtrahend in any Integer, Real or Time data type. Output must be the same data type as inputs.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.

Subtraction Structured Text example

(* ST equivalence: *)

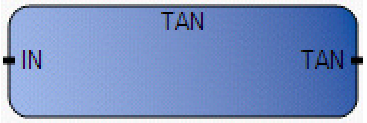
```
ao10 := ai101 - ai102;
ao5 := (ai51 - 1) - ai53;
```

TAN (tangent)

Calculates the tangent of a Real value.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

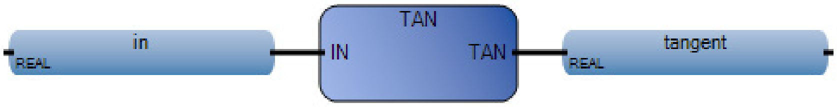
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



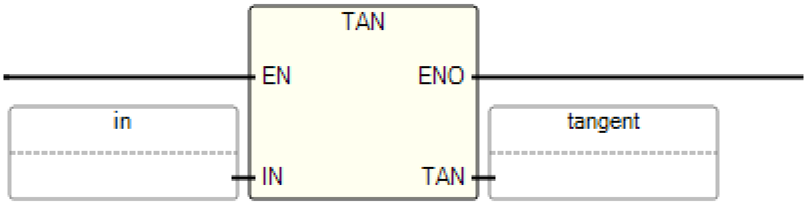
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - perform current tangent computation. FALSE - there is no computation. Applies to Ladder Diagram programs.
IN	Input	REAL	Cannot be equal to PI/2 modulo PI.
TAN	Output	REAL	Tangent of the input value = 1E+38 for invalid input.
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

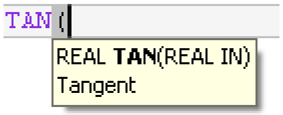
TAN Function Block Diagram example



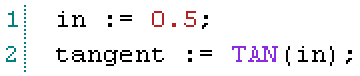
TAN Ladder Diagram example



TAN Structured Text example



```
TAN(
REAL TAN(REAL IN)
Tangent
```



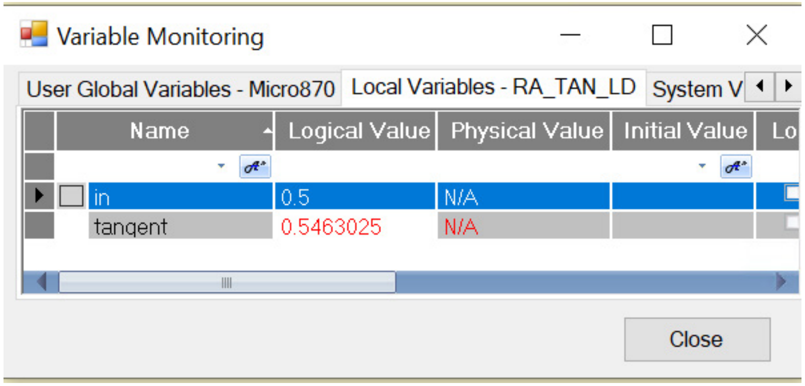
```
in := 0.5;
tangent := TAN(in);
```

(* ST Equivalence: *)

tangent := TAN (angle);

result := ATAN (tangent); (* result is equal to angle*)

Results



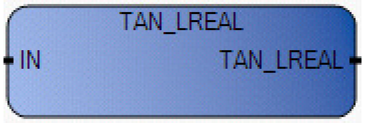
Name	Logical Value	Physical Value	Initial Value	Lo
in	0.5	N/A		
tangent	0.5463025	N/A		

TAN_LREAL (tangent Long Real)

Calculates the tangent of a Long Real value.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.

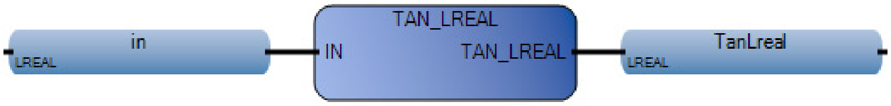


Use this table to help determine the parameter values for this instruction.

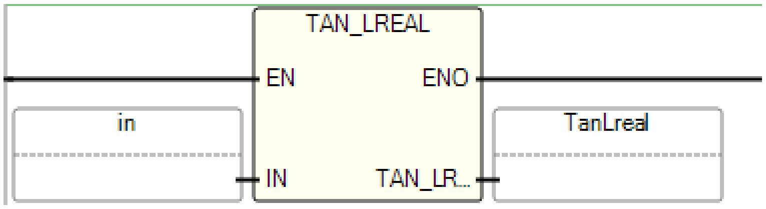
Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - perform current computation. FALSE - there is no computation. Applies to Ladder Diagram programs.
IN	Input	LREAL	Cannot be equal to PI/2 modulo PI.
TAN_LREAL	Output	LREAL	Tangent of the input value = 1E+38 for invalid input.

ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

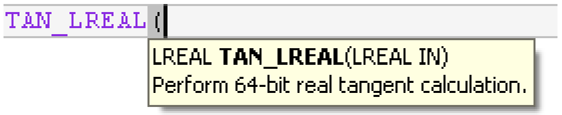
TAN_LREAL Function Block Diagram example



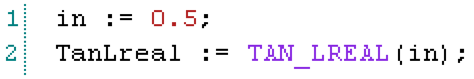
TAN_LREAL Ladder Diagram example



TAN_LREAL Structured Text example



```
TAN_LREAL (
LREAL TAN_LREAL(LREAL IN)
Perform 64-bit real tangent calculation.
```



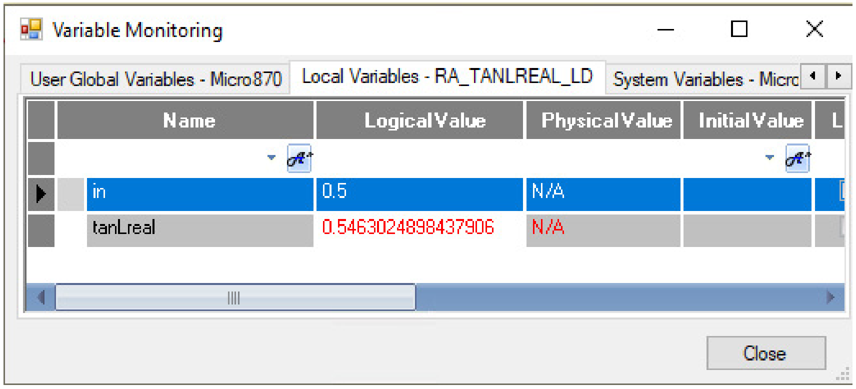
```
in := 0.5;
TanLreal := TAN_LREAL(in);
```

(* ST Equivalence: *)

tangent := TAN_LREAL (angle);

result := ATAN_LREAL (tangent); (* result is equal to angle*)

Results


Variable Monitoring
User Global Variables - Micro870
Local Variables - RA_TANLREAL_LD
System Variables - Micro
Close

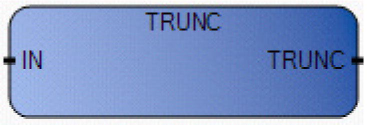
Name	Logical Value	Physical Value	Initial Value	L
in	0.5	N/A		
tanLreal	0.5463024898437906	N/A		

TRUNC (truncate)

Truncates Real values, leaving just the Integer.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

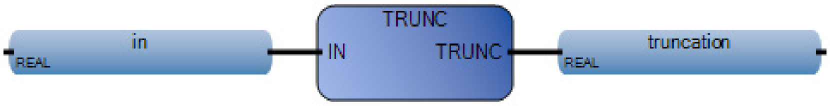
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



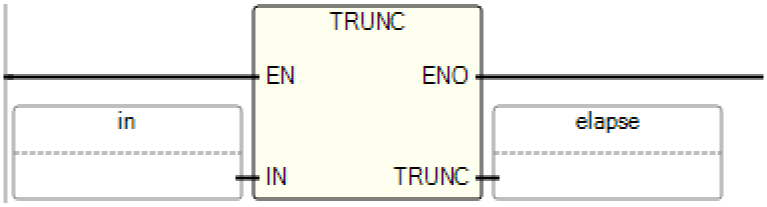
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - perform the truncation of Real value computation. FALSE - there is no computation. Applies to Ladder Diagram programs.
IN	Input	REAL	Any Real value.
TRUNC	Output	REAL	If IN>0, biggest integer less or equal to the input. If IN<0, least integer greater or equal to the input.
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

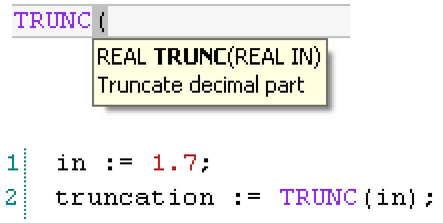
TRUNC Function Block Diagram example



TRUNC Ladder Diagram example



TRUNC Structured Text example



```
TRUNC (
```

REAL **TRUNC**(REAL IN)
Truncate decimal part

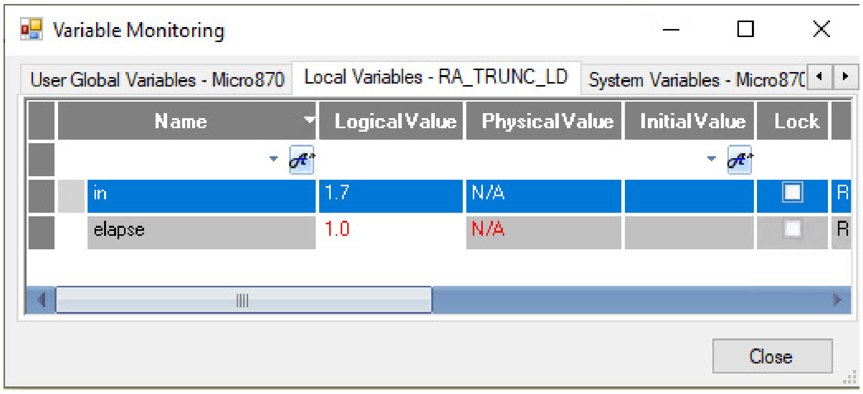
```
in := 1.7;
truncation := TRUNC(in);
```

(* ST Equivalence: *)

result := TRUNC (+2.67) + TRUNC (-2.0891);

(* means: result := 2.0 + (-2.0) := 0.0; *)

Results



Variable Monitoring
User Global Variables - Micro870
Local Variables - RA_TRUNC_LD
System Variables - Micro870
Close

Name	Logical Value	Physical Value	Initial Value	Lock	
in	1.7	N/A		☐	R
elapse	1.0	N/A		☐	R

ASCII serial port instructions

Use the ASCII serial port instructions to use or alter the communication channel for receiving or transmitting data.

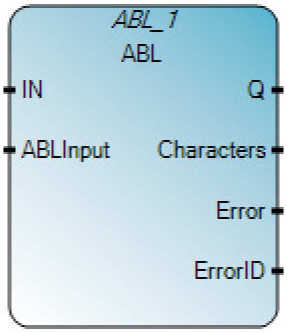
Function block	Description
ABL on page 105	Counts the number of characters in the buffer up to and including end of line character.
ACB on page 111	Counts the total number of characters in the buffer.
ACL on page 107	Clears the receive and transmit buffers.
AHL on page 109	Sets or resets modem handshake lines.
ARD on page 113	Reads characters from the input buffer and places them into a string.
ARL on page 116	Reads one line of characters from the input buffer and places them into a string.
AWA on page 118	Writes a string with two appended (user-configured) characters to an external device.
AWT on page 120	Writes characters from a source string to an external device.

ABL (ASCII test for buffer line)

Counts the number of ASCII characters in the input buffer up to and including the end-of-line termination character.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro830, Micro850, and Micro870 controllers.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
IN	Input	BOOL	Rung input state. TRUE - When Rising Edge is detected, start the function block with the precondition that the last operation is complete. FALSE - The instruction block is idle.

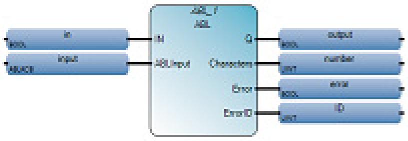
ABLInput	Input	ABLACB	The channel to be operated. Use the ABLACB data type on page 122 to define the Channel, TriggerType, and Cancel parameters for ABLInput.
Q	Output	BOOL	Indicates when the character count is ongoing or complete. The outputs update asynchronously from the program scan. Output Q cannot be used to re-trigger the instruction since IN is edge triggered. TRUE - The function block is complete. FALSE - The function block is not complete.
Characters	Output	UINT	The number of characters in the buffer. The buffer limit is 82 characters.
Error	Output	BOOL	Indicates the existence of an error condition. TRUE - An error is detected. FALSE - No error.
ErrorID	Output	UINT	A unique numeric that identifies the error. The errors for this instruction are defined in ABL error codes.

ABL error codes

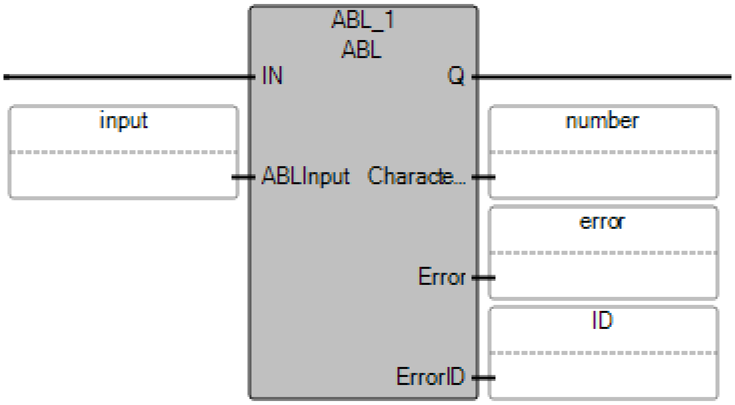
Use this table to determine the ABL error codes and descriptions.

Error code	Error description
03	Transmission cannot be completed because the Clear-to-Send signal was lost.
06	Illegal parameter was detected.
07	Cannot complete ASCII send or receive because channel configuration has been shut down using the channel configuration dialog box.
08	Cannot complete ASCII Write due to an ASCII transmission already in progress.
09	ASCII communication requested is not supported by current channel configuration.
10	The Cancel was set, stopping instruction execution. No action required.
11	The requested length for the string is either invalid, a negative number, greater than 82, or 0. Applies to ARD and ARL function blocks.
13	The requested (.LEN) in the control block is a negative number or a value greater than the string size stored with the source string. Applies to AWA and AWT function blocks.
14	The ACL function block was cancelled.
16	Serial port is not supporting RTS or CTS control lines.

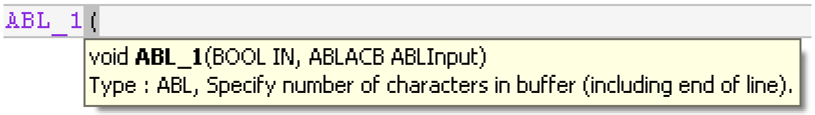
ABL Function Block Diagram example



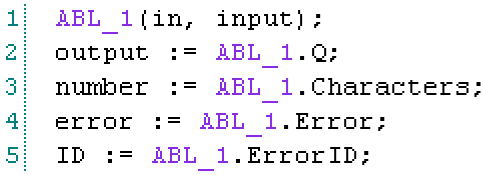
ABL Ladder Diagram example



ABL Structured Text example



```
ABL_1(
void ABL_1(BOOL IN, ABLACB ABLInput)
Type : ABL, Specify number of characters in buffer (including end of line).
```



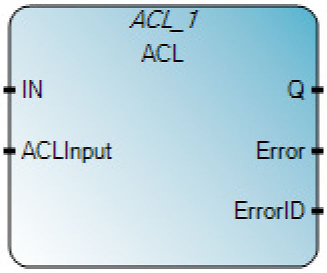
```
ABL_1(in, input);
output := ABL_1.Q;
number := ABL_1.Characters;
error := ABL_1.Error;
ID := ABL_1.ErrorID;
```

ACL (ASCII clear buffer)

Clears the receive and transmit buffers, and removes instructions from the ASCII queue.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro830, Micro850, and Micro870 controllers.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
IN	Input	BOOL	Rung input state. TRUE - When Rising Edge is detected, start the function block with the precondition that the last operation is complete. FALSE - The instruction block is idle.

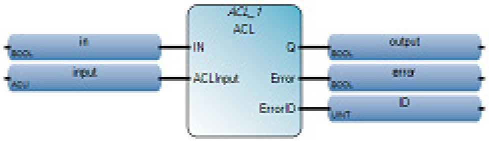
ACLInput	Input	ACLI	The channel to be operated, and the state of the transmit and receive buffers. For RXBuffer, clears the receive buffer and removes the receive ASCII function blocks (ARL and ARD) from the ASCII queue. For TXBuffer, clears the transmit buffer and removes the transmit ASCII function blocks (AWA and AWT) from the ASCII queue. Use the ACLI data type on page 122 to define the Channel, RXBuffer, and TXBuffer parameters for ACLInput.
Q	Output	BOOL	Indicates when the ASCII queue clearing process is ongoing or complete. TRUE - The function block is complete. FALSE - The function block is not complete.
Error	Output	BOOL	Indicates the existence of an error condition. TRUE - An error is detected. FALSE - No error.
ErrorID	Output	UINT	A unique numeric that identifies the error. The errors for this instruction are defined in ABL error codes.

ABL error codes

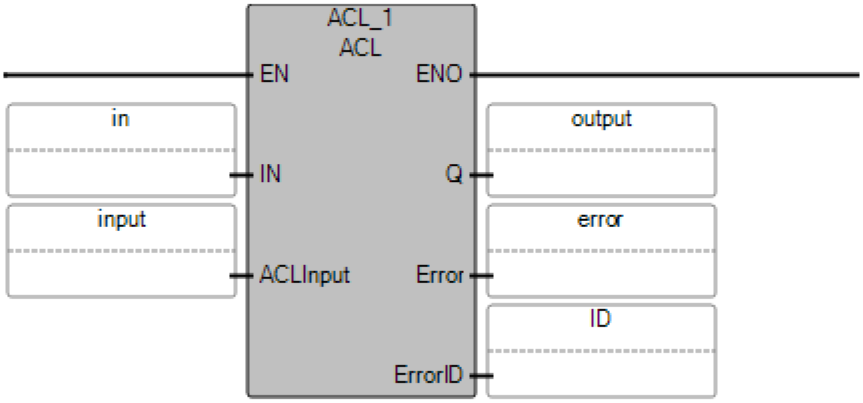
Use this table to determine the ABL error codes and descriptions.

Error code	Error description
03	Transmission cannot be completed because the Clear-to-Send signal was lost.
06	Illegal parameter was detected.
07	Cannot complete ASCII send or receive because channel configuration has been shut down using the channel configuration dialog box.
08	Cannot complete ASCII Write due to an ASCII transmission already in progress.
09	ASCII communication requested is not supported by current channel configuration.
10	The Cancel was set, stopping instruction execution. No action required.
11	The requested length for the string is either invalid, a negative number, greater than 82, or 0. Applies to ARD and ARL function blocks.
13	The requested (.LEN) in the control block is a negative number or a value greater than the string size stored with the source string. Applies to AWA and AWT function blocks.
14	The ACL function block was cancelled.
16	Serial port is not supporting RTS or CTS control lines.

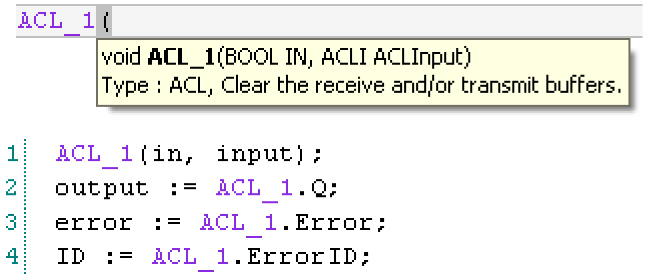
ACL Function Block Diagram example



ACL Ladder Diagram example



ACL Structured Text example



```
ACL_1(
```

void **ACL_1**(BOOL IN, ACLI ACLInput)
Type : ACL, Clear the receive and/or transmit buffers.

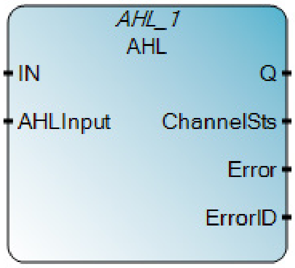
```
ACL_1(in, input);
output := ACL_1.Q;
error := ACL_1.Error;
ID := ACL_1.ErrorID;
```

AHL (ASCII handshake lines)

Sets or resets the RS-232 Request to Send (RTS) handshake control lines for a modem.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro830, Micro850, and Micro870 controllers.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
IN	Input	BOOL	Rung input state. TRUE - When Rising Edge is detected, start the instruction block with the precondition that the last operation is complete. FALSE - The instruction block is idle.

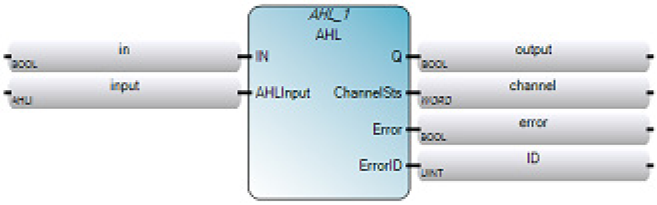
AHLInput	Input	AHLI	The channel to be operated, and the set or reset of the RTS control line for the modem. Use the AHLI data type on page 123 to define the Channel, SetRts, ClrRts, and Cancel parameters for AHLInput.
Q	Output	BOOL	Indicates when the set or reset is complete. Outputs of this instruction are updated asynchronously from the program scan. Output Q cannot be used to re-trigger the instruction since IN is edge triggered. TRUE - The function block is complete. FALSE - The function block is not complete.
ChannelSts	Output	WORD	Displays the current status (0000 to 001F) of the handshake lines for the specified channel.
Error	Output	BOOL	Indicates the existence of an error condition. TRUE - An error is detected. FALSE - No error.
ErrorID	Output	UINT	A unique numeric that identifies the error. The errors for this instruction are defined in ABL error codes.

ABL error codes

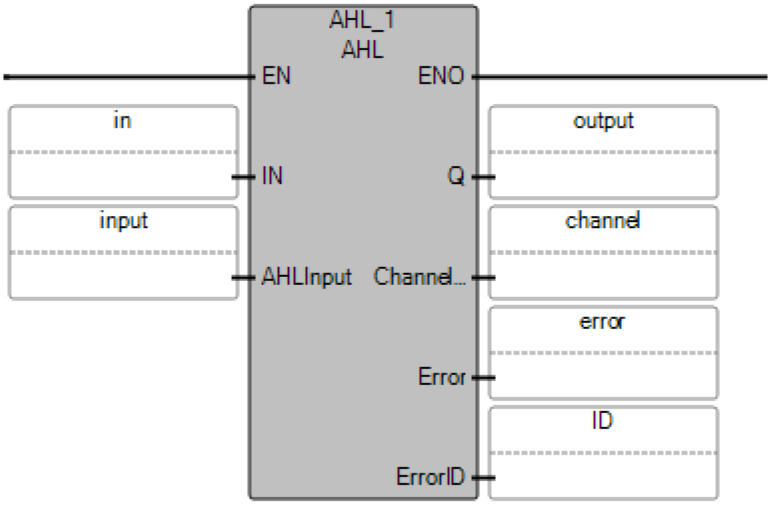
Use this table to determine the ABL error codes and descriptions.

Error code	Error description
03	Transmission cannot be completed because the Clear-to-Send signal was lost.
06	Illegal parameter was detected.
07	Cannot complete ASCII send or receive because channel configuration has been shut down using the channel configuration dialog box.
08	Cannot complete ASCII Write due to an ASCII transmission already in progress.
09	ASCII communication requested is not supported by current channel configuration.
10	The Cancel was set, stopping instruction execution. No action required.
11	The requested length for the string is either invalid, a negative number, greater than 82, or 0. Applies to ARD and ARL function blocks.
13	The requested (.LEN) in the control block is a negative number or a value greater than the string size stored with the source string. Applies to AWA and AWT function blocks.
14	The ACL function block was cancelled.
16	Serial port is not supporting RTS or CTS control lines.

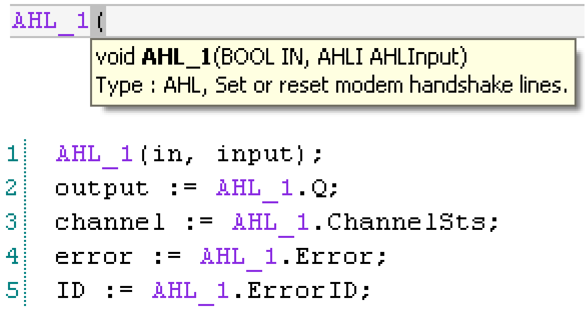
AHL Function Block Diagram example



AHL Ladder Diagram example



AHL Structured Text example



```
AHL_1(
void AHL_1(BOOL IN, AHLI AHLInput)
Type : AHL, Set or reset modem handshake lines.
```

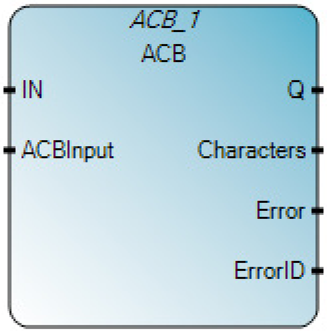
```
AHL_1(in, input);
output := AHL_1.Q;
channel := AHL_1.ChannelSts;
error := AHL_1.Error;
ID := AHL_1.ErrorID;
```

ACB (ASCII characters in buffer)

Counts the total number of ASCII characters in the buffer including end of line.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro830, Micro850, and Micro870 controllers.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description

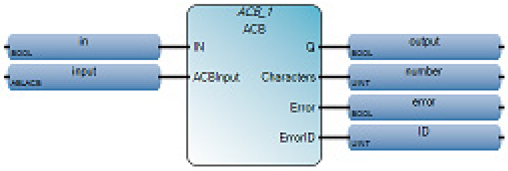
IN	Input	BOOL	Rung input state. TRUE - When Rising Edge is detected, start the instruction block with the precondition that the last operation is complete. FALSE - The instruction block is idle.
ACBInput	Input	ABLACB on page 122	The channel to be operated. Use the ABLACB data type to define the Channel, TriggerType, and Cancel parameters for ACBInput.
Q	Output	BOOL	Indicates whether the character count is ongoing or complete. Outputs of this function block are updated asynchronously from the program scan. Output Q cannot be used to re-trigger the instruction since IN is edge triggered. TRUE - The counting is complete. FALSE - The counting is ongoing.
Characters	Output	UINT	The number of characters in the buffer.
Error	Output	BOOL	Indicates the existence of an error condition. FALSE - No error. TRUE - An error is detected.
ErrorID	Output	UINT	A unique numeric that identifies the error. The errors for this instruction are defined in ABL error codes.

ACB error codes

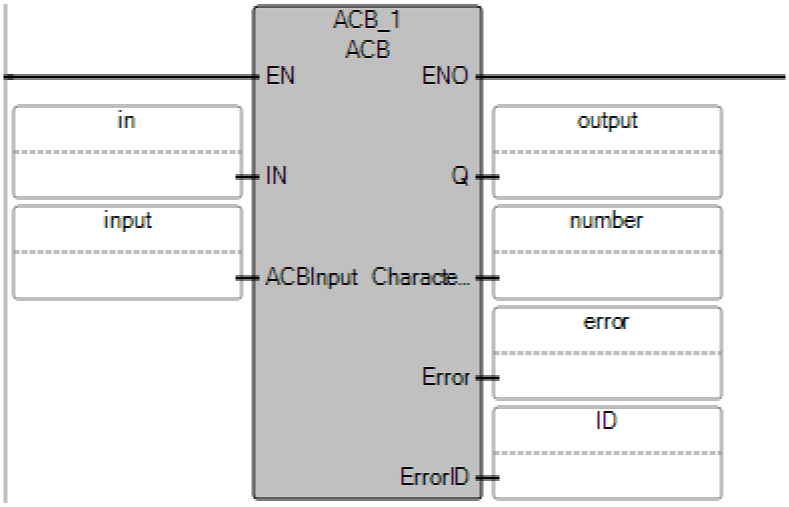
Use this table to determine the ABL error codes and descriptions.

Error code	Error description
03	Transmission cannot be completed because the Clear-to-Send signal was lost.
06	Illegal parameter was detected.
07	Cannot complete ASCII send or receive because channel configuration has been shut down using the channel configuration dialog box.
08	Cannot complete ASCII Write due to an ASCII transmission already in progress.
09	ASCII communication requested is not supported by current channel configuration.
10	The Cancel was set, stopping instruction execution. No action required.
11	The requested length for the string is either invalid, a negative number, greater than 82, or 0. Applies to ARD and ARL function blocks.
13	The requested (.LEN) in the control block is a negative number or a value greater than the string size stored with the source string. Applies to AWA and AWT function blocks.
14	The ACL function block was cancelled.
16	Serial port is not supporting RTS or CTS control lines.

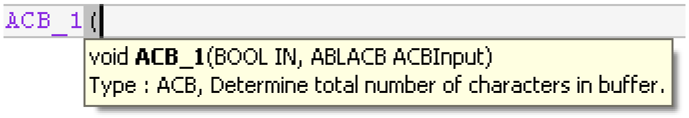
ACB Function Block Diagram example



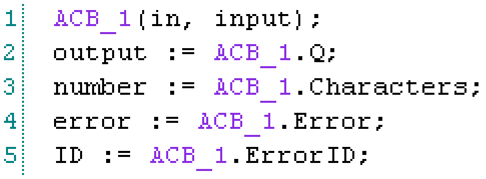
ACB Ladder Diagram example



ACB Structured Text example



```
ACB_1(
void ACB_1(BOOL IN, ABLACB ACBInput)
Type : ACB, Determine total number of characters in buffer.
```



```
ACB_1(in, input);
output := ACB_1.Q;
number := ACB_1.Characters;
error := ACB_1.Error;
ID := ACB_1.ErrorID;
```

ARD (ASCII read)

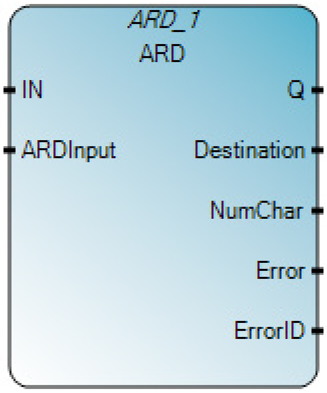
Reads ASCII characters from the input buffer and stores them into a string.

Operation details:

- The ARD instruction runs until all of the characters in the ASCII buffer are received. If another ASCII instruction on page 105 is executed, it is queued until ARD is finished.
- To cancel the ARD instruction, execute an ACL instruction.
- To prevent the ARD instruction delaying the ASCII queue while it waits for the required number of characters, use the results of an ACB instruction to trigger the ARD instruction.
- Status of the instruction can be extracted from the control bit of the instruction instance (for example, ARD_1.controlbit). This shows if the instruction is blocking the ASCII instruction queue waiting for more characters:
  - 7th bit = Instruction is enabled.
  - 6th bit = Instruction is in the queue.
  - 5th bit = Instruction is done.
  - 3rd bit = Instruction has an error.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro830, Micro850, and Micro870 controllers.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
IN	Input	BOOL	Rung input state. TRUE - When Rising Edge is detected, start the instruction block with the precondition that the last operation is complete. FALSE - The instruction block is idle.
ARDInput	Input	ARDARL on page 123	Read characters from the buffer. The maximum is 82. Use the ARDARL data type to define the Channel, Length, and Cancel parameters for the ARDInput.
Q	Output	BOOL	Indicates when the buffer read is ongoing or complete. Outputs of this instruction are updated asynchronously from the program scan. Output Q cannot be used to re-trigger the instruction since IN is edge triggered. TRUE - The function block is complete. FALSE - The function block is not complete.
Destination	Output	ASCIILOCADDR	The string element where the characters are stored.
NumChar	Output	UINT	The number of characters.
Error	Output	BOOL	Indicates the existence of an error condition. TRUE - An error is detected. FALSE - No error.
ErrorID	Output	UINT	A unique numeric that identifies the error. The errors for this instruction are defined in ABL error codes.

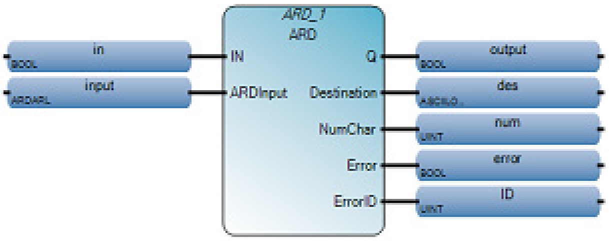
ABL error codes

Use this table to determine the ABL error codes and descriptions.

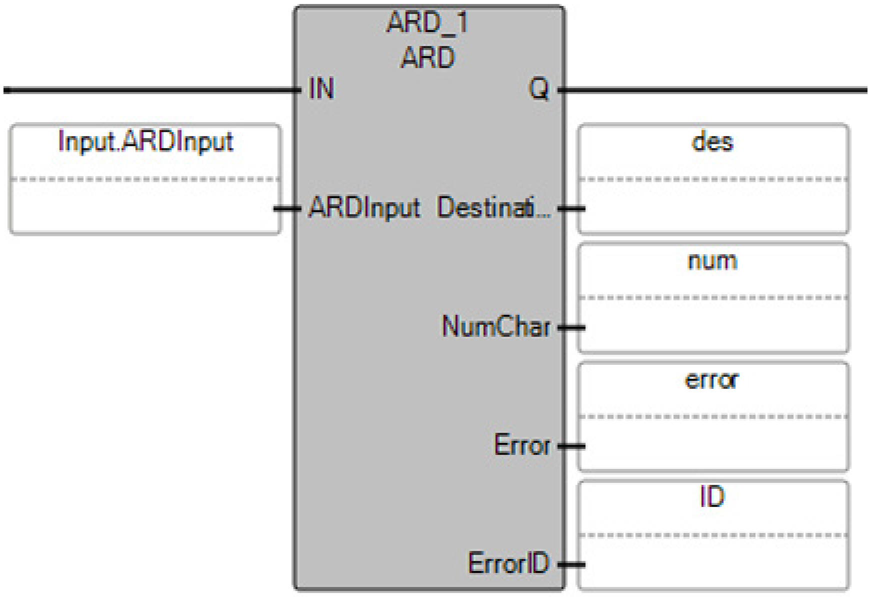
Error code	Error description
03	Transmission cannot be completed because the Clear-to-Send signal was lost.
06	Illegal parameter was detected.
07	Cannot complete ASCII send or receive because channel configuration has been shut down using the channel configuration dialog box.
08	Cannot complete ASCII Write due to an ASCII transmission already in progress.
09	ASCII communication requested is not supported by current channel configuration.
10	The Cancel was set, stopping instruction execution. No action required.

11	The requested length for the string is either invalid, a negative number, greater than 82, or 0. Applies to ARD and ARL function blocks.
13	The requested (.LEN) in the control block is a negative number or a value greater than the string size stored with the source string. Applies to AWA and AWT function blocks.
14	The ACL function block was cancelled.
16	Serial port is not supporting RTS or CTS control lines.

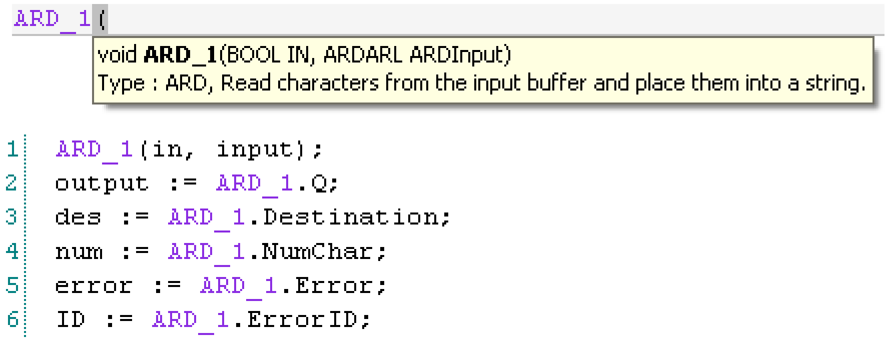
ARD Function Block Diagram example



ARD Ladder Diagram example



ARD Structured Text example



```
ARD_1(
```

void **ARD_1**(BOOL IN, ARDARL ARDInput)
Type : ARD, Read characters from the input buffer and place them into a string.

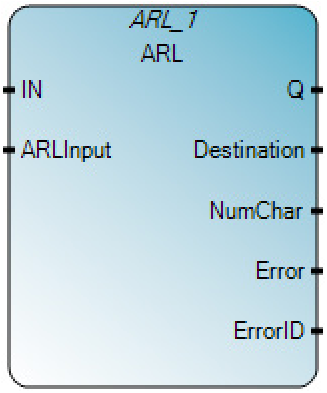
```
ARD_1(in, input);
output := ARD_1.Q;
des := ARD_1.Destination;
num := ARD_1.NumChar;
error := ARD_1.Error;
ID := ARD_1.ErrorID;
```

ARL (ASCII read line)

Reads a line of ASCII on page 105 characters from the buffer, up to and including the termination characters, and stores them in a string.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro830, Micro850, and Micro870 controllers.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
IN	Input	BOOL	Rung input state. TRUE - When Rising Edge is detected, start the instruction block with the precondition that the last operation is complete. FALSE - The instruction block is idle.
ARLInput	Input	ARDARL on page 123	Read a line of ASCII characters from the buffer. The maximum is 82. Use the ARDARL data type to define the Channel, Length, and Cancel parameters for the ARDLInput.
Q	Output	BOOL	Indicates when the read line from the input buffer is ongoing or complete. Outputs of this instruction are updated asynchronously from the program scan. Output Q cannot be used to re-trigger the instruction since IN is edge triggered. TRUE - The function block is complete. FALSE - The function block is not complete.
Destination	Output	ASCIILOCADDR	The string element where the characters are stored.
NumChar	Output	UINT	The number of characters in the line, including the termination character.
Error	Output	BOOL	Indicates the existence of an error condition. TRUE - An error is detected. FALSE - No error.
ErrorID	Output	UINT	A unique numeric that identifies the error. The errors for this instruction are defined in ABL error codes.

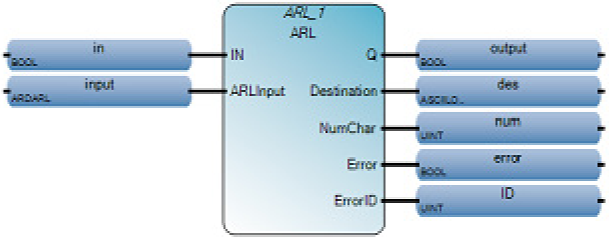
ABL error codes

Use this table to determine the ABL error codes and descriptions.

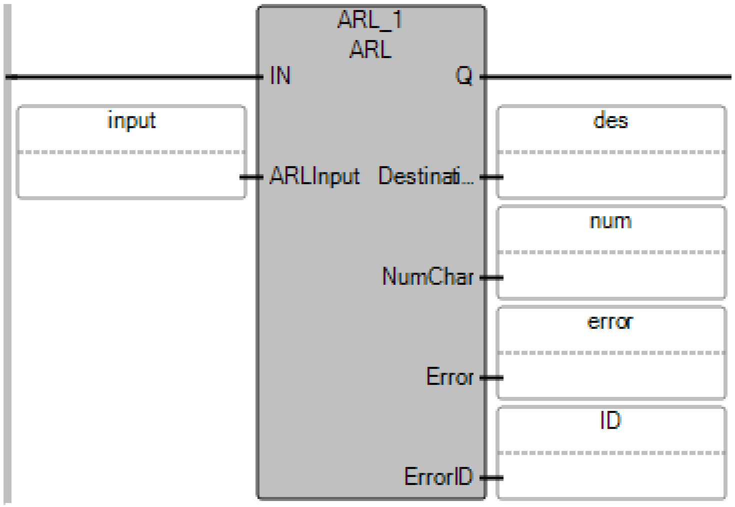
Error code	Error description
03	Transmission cannot be completed because the Clear-to-Send signal was lost.
06	Illegal parameter was detected.
07	Cannot complete ASCII send or receive because channel configuration has been shut down using the channel configuration dialog box.

08	Cannot complete ASCII Write due to an ASCII transmission already in progress.
09	ASCII communication requested is not supported by current channel configuration.
10	The Cancel was set, stopping instruction execution. No action required.
11	The requested length for the string is either invalid, a negative number, greater than 82, or 0. Applies to ARD and ARL function blocks.
13	The requested (.LEN) in the control block is a negative number or a value greater than the string size stored with the source string. Applies to AWA and AWT function blocks.
14	The ACL function block was cancelled.
16	Serial port is not supporting RTS or CTS control lines.

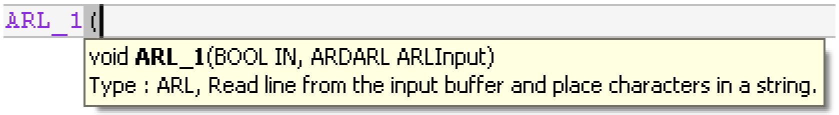
ARL Function Block Diagram example



ARL Ladder Diagram example

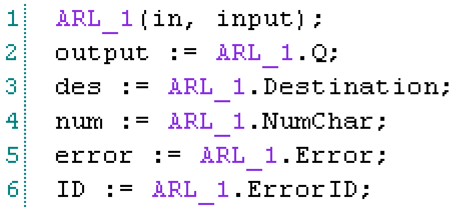


ARL Structured Text example



ARL_1(

void **ARL_1**(BOOL IN, ARDARL ARLInput)
Type : ARL, Read line from the input buffer and place characters in a string.



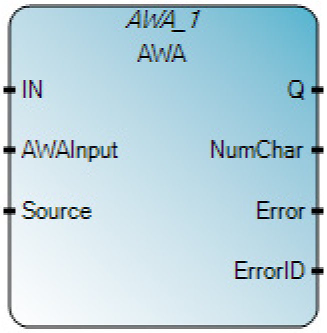
```
ARL_1(in, input);
output := ARL_1.Q;
des := ARL_1.Destination;
num := ARL_1.NumChar;
error := ARL_1.Error;
ID := ARL_1.ErrorID;
```

AWA (ASCII write append)

Writes a string with two appended (user-configured) characters to an external device.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro830, Micro850, and Micro870 controllers.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
IN	Input	BOOL	Rung input state. TRUE - Rising Edge is detected, start the instruction block with the precondition that the last operation is complete. FALSE - The instruction block is idle.
AWAInput	Input	AWAAWT	The channel and number (Length) of the characters to write to the buffer. Maximum is 82. Use the AWAAWT data type on page 123 to define Channel, Length, and Cancel parameters for AWAInput.
Source	Input	ASCIILOCADDR	The source string that was output as a character array by either the ARD or ARL instruction.
Q	Output	BOOL	Indicates when the write is ongoing or complete. Outputs of this instruction are updated asynchronously from the program scan. Output Q cannot be used to re-trigger the instruction since IN is edge triggered. TRUE - The function block is complete. FALSE - The function block is not complete.
NumChar	Output	UINT	The number of characters. NumChar may be less than the Length requested to be transmitted if the length of the Source String is shorter than the requested Length. Updates when the transmission is complete and Q is TRUE.
Error	Output	BOOL	Indicates the existence of an error condition. TRUE - An error is detected. FALSE - No error.
ErrorID	Output	UINT	A unique numeric that identifies the error. The errors for this instruction are defined in ABL error codes.

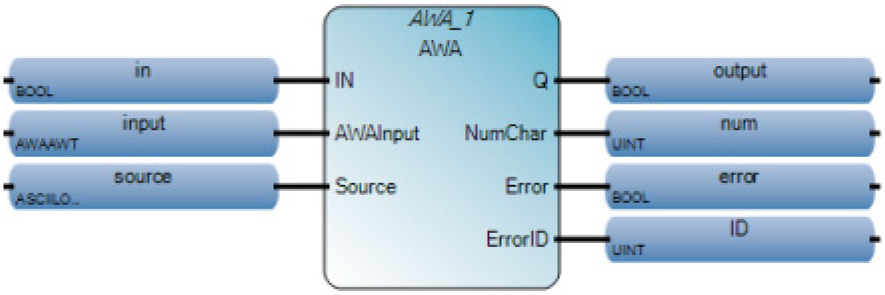
ABL error codes

Use this table to determine the ABL error codes and descriptions.

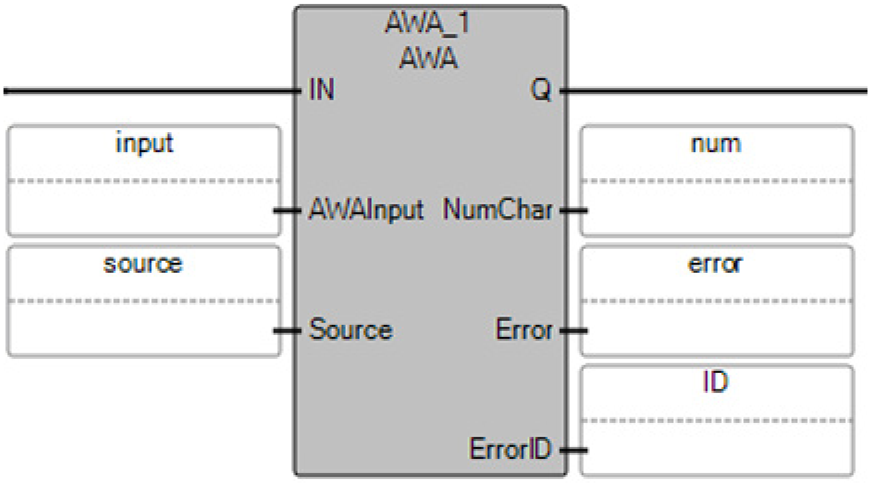
Error code	Error description
03	Transmission cannot be completed because the Clear-to-Send signal was lost.
06	Illegal parameter was detected.
07	Cannot complete ASCII send or receive because channel configuration has been shut down using the channel configuration dialog box.

08	Cannot complete ASCII Write due to an ASCII transmission already in progress.
09	ASCII communication requested is not supported by current channel configuration.
10	The Cancel was set, stopping instruction execution. No action required.
11	The requested length for the string is either invalid, a negative number, greater than 82, or 0. Applies to ARD and ARL function blocks.
13	The requested (.LEN) in the control block is a negative number or a value greater than the string size stored with the source string. Applies to AWA and AWT function blocks.
14	The ACL function block was cancelled.
16	Serial port is not supporting RTS or CTS control lines.

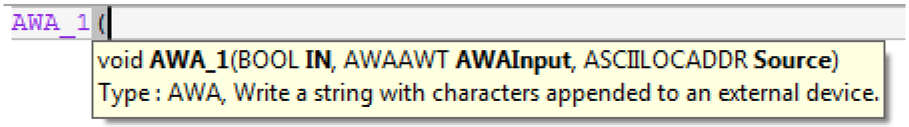
AWA Function Block Diagram example



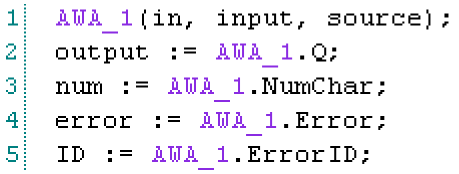
AWA Ladder Diagram example



AWA Structured Text example



```
AWA_1(
void AWA_1(BOOL IN, AWAAWT AWAInput, ASCIILOCADDR Source)
Type : AWA, Write a string with characters appended to an external device.
```



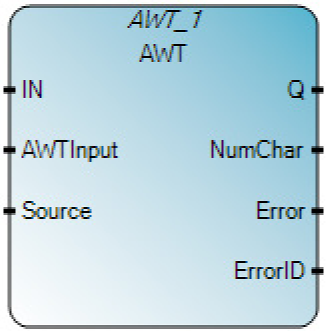
```
AWA_1(in, input, source);
output := AWA_1.Q;
num := AWA_1.NumChar;
error := AWA_1.Error;
ID := AWA_1.ErrorID;
```

AWT (ASCII write)

Writes ASCII characters from a source string to an external device.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro830, Micro850, and Micro870 controllers.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
IN	Input	BOOL	Rung input state. TRUE - When Rising Edge is detected, start the instruction block with the precondition that the last operation is complete. FALSE - The instruction block is idle.
AWTInput	Input	AWAAWT	The channel and number (Length) of the characters to write to the buffer. Maximum is 82. Use the AWAAWT data type on page 123 to define Channel, Length, and Cancel parameters for AWTInput.
Source	Input	ASCIILOCADDR	The source string that was output as a character array by either the ARD or ARL instruction.
Q	Output	BOOL	Indicates when the write is ongoing or complete. Outputs of this instruction are updated asynchronously from the program scan. Output Q cannot be used to re-trigger the instruction since IN is edge triggered. TRUE - The function block is complete. FALSE - The function block is not complete.
NumChar	Output	UINT	The number of characters. NumChar may be less than the Length requested to be transmitted if the length of the Source String is shorter than the requested Length. Updates when the transmission is complete and Q is TRUE.
Error	Output	BOOL	Indicates the existence of an error condition. TRUE - An error is detected. FALSE - No error.
ErrorID	Output	UINT	A unique numeric that identifies the error. The errors for this instruction are defined in ABL error codes.

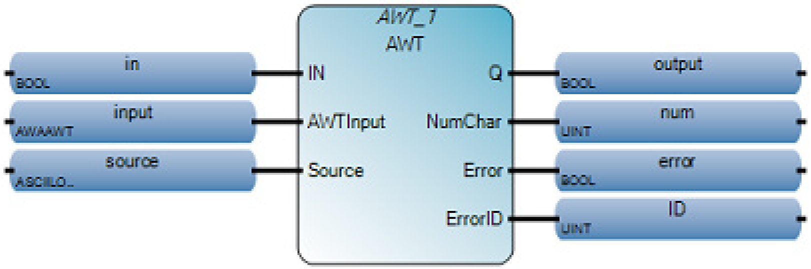
ABL error codes

Use this table to determine the ABL error codes and descriptions.

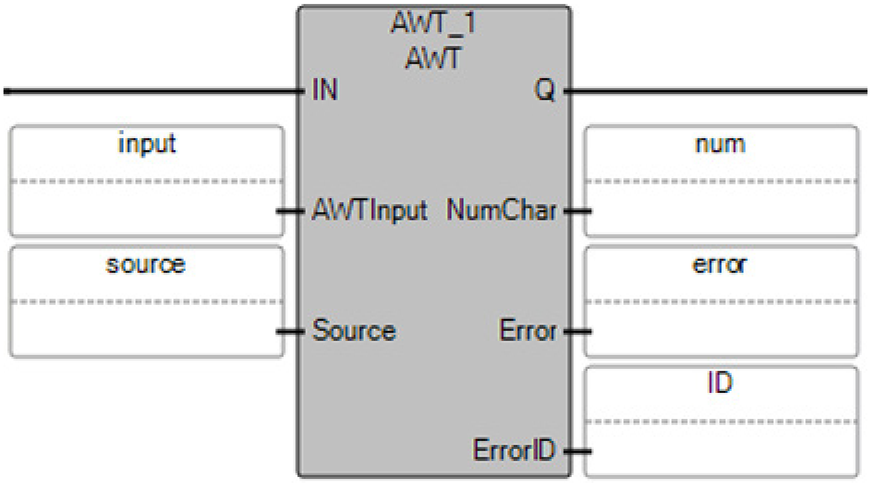
Error code	Error description
03	Transmission cannot be completed because the Clear-to-Send signal was lost.
06	Illegal parameter was detected.
07	Cannot complete ASCII send or receive because channel configuration has been shut down using the channel configuration dialog box.

08	Cannot complete ASCII Write due to an ASCII transmission already in progress.
09	ASCII communication requested is not supported by current channel configuration.
10	The Cancel was set, stopping instruction execution. No action required.
11	The requested length for the string is either invalid, a negative number, greater than 82, or 0. Applies to ARD and ARL function blocks.
13	The requested (.LEN) in the control block is a negative number or a value greater than the string size stored with the source string. Applies to AWA and AWT function blocks.
14	The ACL function block was cancelled.
16	Serial port is not supporting RTS or CTS control lines.

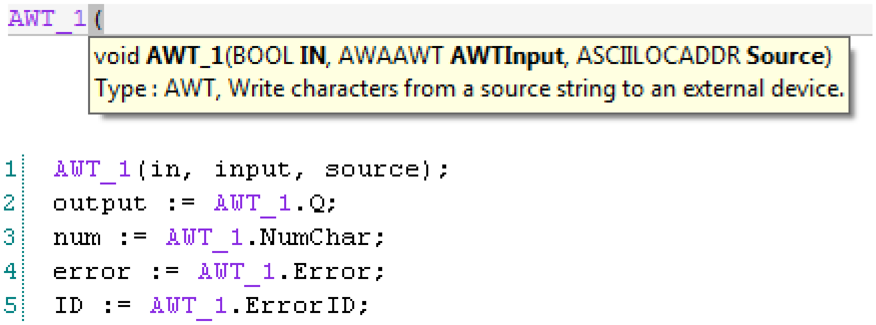
AWT Function Block Diagram example



AWT Ladder Diagram example



AWT Structured Text example



AWT_1(

void **AWT_1**(BOOL **IN**, AWAAWT **AWTInput**, ASCIILOCADDR **Source**)
Type : AWT, Write characters from a source string to an external device.

```
AWT_1(in, input, source);
output := AWT_1.Q;
num := AWT_1.NumChar;
error := AWT_1.Error;
ID := AWT_1.ErrorID;
```

ASCII parameter details

The following topics provide additional details for ASCII parameters and structured data types.

- ABLACB data type
- ACL data type
- AHL ChannelSts data type
- AHLI data type
- ARDARL data type
- AWAAWT data type

ABLACB data type

Use this table to help determine the parameter values for the ABLACB data type.

Parameter	Data type	Description
Channel	UINT	Serial port number: • 2 for the embedded serial port, or • 5-9 for serial port plug-ins installed in slots 1 through 5: • 5 for slot 1 • 6 for slot 2 • 7 for slot 3 • 8 for slot 4 • 9 for slot 5
TriggerType	USINT	Represents one of the following: • 0: Msg Triggered Once (when IN goes from False to True) • 1: Msg triggered continuously when IN is True • Other value: Reserved
Cancel	BOOL	When this input is set to TRUE, this function block does not execute.

ACL data type

Use this table to help determine the parameter values for the ABL data type.

Parameter	Data type	Description
Channel	UINT	Serial port number: • 2 for the embedded serial port, or • 5-9 for serial port plug-ins installed in slots 1 through 5: • 5 for slot 1 • 6 for slot 2 • 7 for slot 3 • 8 for slot 4 • 9 for slot 5
RXBuffer	BOOL	When TRUE, clears the receive buffer and removes the receive ASCII function blocks (ARL and ARD) from the ASCII queue.
TXBuffer	BOOL	When TRUE, clears the transmit buffer and removes the transmit ASCII function blocks (AWA and AWT) from the ASCII queue.

AHL ChannelSts data type

Use this table to help determine the parameter values for the AHL ChannelSts data type.

Parameter	Data type	Description
DTRstatus	UINT	Used for the DTR signal (reserved)

Parameter	Data type	Description
DCDstatus	UINT	Used for the DCD signal (bit 3 of word) 1 indicates active
DSRstatus	UINT	Used for the DSR signal (reserved)
RTSstatus	UINT	Used for the RTS signal (bit 1 of word) 1 indicates active
CTSstatus	UINT	Used for the CTS signal (bit 0 of word) 1 indicates active

AHLI data type

Use this table to help determine the parameter values for the AHL data type.

Parameter	Data type	Description
Channel	UINT	Serial port number: • 2 for the embedded serial port, or • 5-9 for serial port plug-ins installed in slots 1 through 5: • 5 for slot 1 • 6 for slot 2 • 7 for slot 3 • 8 for slot 4 • 9 for slot 5
ClrRts	BOOL	Used to reset the RTS control line.
SetRts	BOOL	Used to set the RTS control line.
Cancel	BOOL	When this input is set to TRUE, this function block does not execute.

ARDARL data type

Use this table to help determine the parameter values for the ARDARL data type.

Parameter	Data type	Description
Channel	UINT	Serial port number: • 2 for the embedded serial port, or • 5-9 for serial port plug-ins installed in slots 1 through 5: • 5 for slot 1 • 6 for slot 2 • 7 for slot 3 • 8 for slot 4 • 9 for slot 5
Length	UINT	The number of characters that you want to read from the buffer (maximum is 82).
Cancel	BOOL	When this input is set to TRUE, this function block does not execute. If already executing, operation ceases.

AWAAWT data type

Use this table to help determine the parameter values for the AWAAWT data type.

Parameter	Data type	Description

Parameter	Data type	Description
Channel	UINT	Serial port number: • 2 for the embedded serial port, or • 5-9 for serial port plug-ins installed in slots 1 through 5: • 5 for slot 1 • 6 for slot 2 • 7 for slot 3 • 8 for slot 4 • 9 for slot 5
Length	UINT	Defines the number of characters to write to the buffer (maximum is 82). When Length is set to 0, AWA sends 0 bytes of user data and 2 bytes of appended characters to the buffer.
Cancel	BOOL	When TRUE, this function block does not execute. If already executing, operation ceases.

Binary instructions

Use Binary instructions to perform mathematical operations.

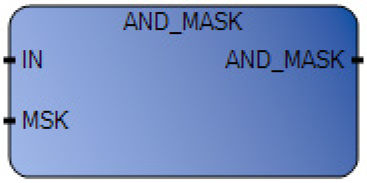
Operator	Description
AND_MASK on page 125	Performs a bit-to-bit AND between two Integer values.
NOT_MASK on page 133	Integer bit-to-bit negation mask, inverts a parameter value.
BSL on page 126	Shifts a bit in an array element to the left.
BSR on page 130	Shifts a bit in an array element to the right.
OR_MASK on page 134	Integer OR bit-to-bit mask, turns bits on.
ROL on page 136	For 32-bit integers, rotates integer bits to the left.
ROR on page 137	For 32-bit integers, rotates integer bits to the left.
SHL on page 139	For 32-bit integers, moves integers to the left and places 0 in the least significant bit.
SHR on page 141	For 32-bit integers, moves integers to the right and places 0 in the most significant bit.
XOR_MASK on page 142	Integer exclusive OR bit-to-bit mask, returns inverted bit values.

AND_MASK (AND mask)

Performs a bit to bit AND between two integer values.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

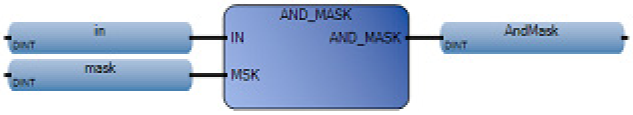
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



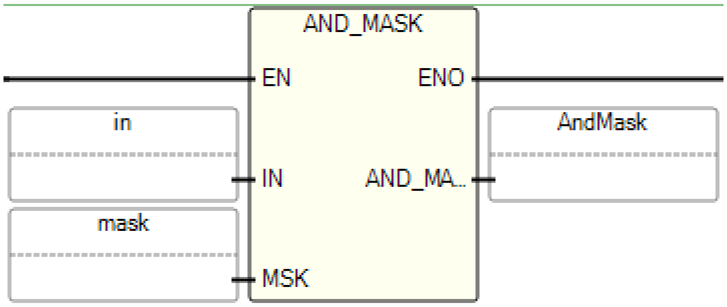
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute the Integer AND bit-to-bit mask computation. FALSE - there is no computation. Applies to Ladder Diagram programs.
IN	Input	DINT	Must have integer format.
MSK	Input	DINT	Must have integer format.
AND_MASK	Output	DINT	Bit-to-bit logical AND between IN and MSK.
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

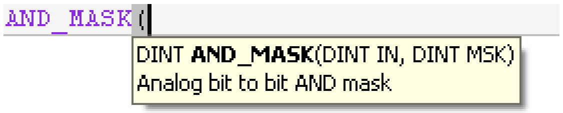
AND_MASK Function Block Diagram example



AND_MASK Ladder Diagram example

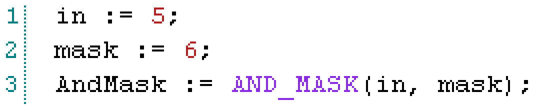


AND_MASK Structured Text example



```
AND_MASK(
```

DINT **AND_MASK**(DINT IN, DINT MSK)
Analog bit to bit AND mask



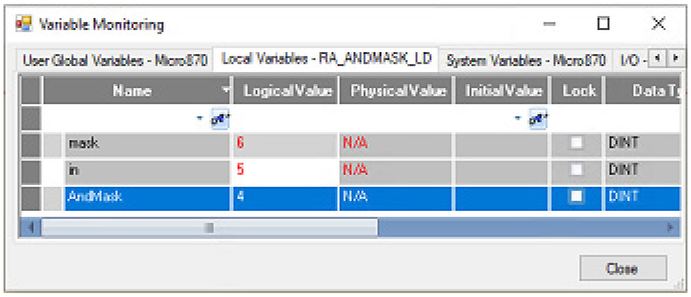
```
in := 5;
mask := 6;
AndMask := AND_MASK(in, mask);
```

(* ST Equivalence: *)

parity := AND_MASK (xvalue, 1); (* 1 if xvalue is odd *)

result := AND_MASK (16#abc, 16#f0f); (* equals 16#a0c *)

Results


Variable Monitoring
User Global Variables - Micro870
Local Variables - RA_ANDMASK_LD
System Variables - Micro870
I/O -
Close

Name	LogicalValue	PhysicalValue	InitialValue	Lock	Data T
mask	6	N/A		□	DINT
in	5	N/A		□	DINT
AndMask	4	N/A		□	DINT

BSL (bit shift left)

Shifts a bit in an array element to the left.

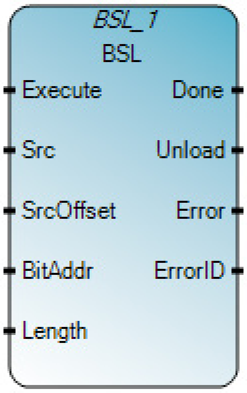
Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.

Operation details:

The BSL instruction is an immediate process on false-to-true rung transition and updates output synchronously. When Execute is TRUE, the leftmost bit (Src + SrcOffset and Length) is copied into the Unload bit and all bits in the array or non-array are shifted left by one bit. Length and 16 bit boundary are considered except for BOOL data types. The external bit is then moved to bit 0 (Src + SrcOffset) of the first element.

For wraparound operations, set the position of the BitAddr to the last bit position or to the Unload bit. Possible usage of the BSL instruction, track bottles through a bottling line where each bit represents a bottle.



Use this table to help determine the parameter values for this instruction.

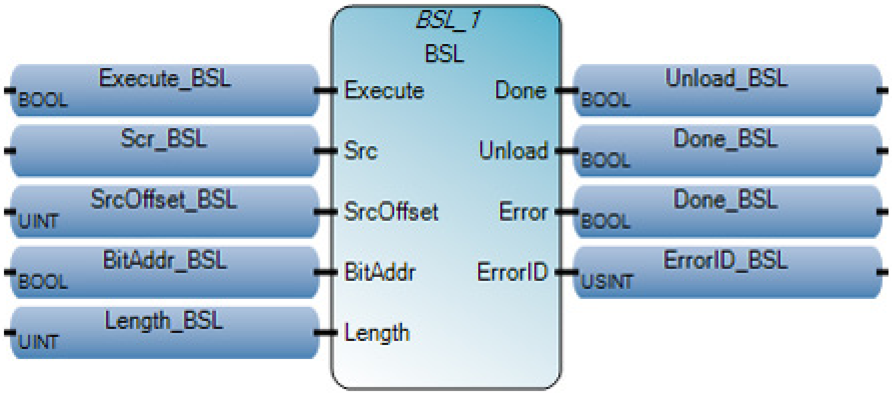
Parameter	Parameter Type	Data Type	Description
Execute	Input	BOOL	Instruction enable. TRUE - Rising Edge detected, shifts bit to the left one position. • Verify fault conditions first. • If Length = 0, the external bit is moved into the Unload bit. No bit shift is done on Scr. Error and ErrorID bits are reset. Done bit is set. • If Length > 0 and Length ≤ 2048, the Error and ErrorID bits are reset. After the bit shift completes, the Done bit is set. • If Length > 0 and Length ≤ 2048, the leftmost bit (addressed by Src + SrcOffset and Length) is copied into the Unload bit and all bits that are part of the array or non-array are shifted left by one bit (up to the bit Length and 16-bit boundary except for BOOL). External bit is moved to bit 0 (Src + SrcOffset) of the first element. FALSE - Rising Edge not detected, do not enable BSL operation.
Scr	Input	ANY_ELEMENTARY	The address of the Src (bit) to be shifted. Supported data types: BOOL, DWORD, INT, UINT, WORD, DINT and UDINT. • Arrays: Set Scr to a variable based address such as: Source1, Source1[0], or Source1[1] . • Non-arrays: Set Scr to a variable address such as Source1.
SrcOffset	Input	UINT	If SrcOffset is 0, start from the first element. • Arrays: Set SrcOffset to 0. If set to Source1[0] or Source1[1] an error occurs: 'Source offset exceeds the size of the array.' • Non-arrays: Set SrcOffset to 0 or an error occurs: 'Source offset exceeds the size of the array.'
BitAddr	Input	BOOL	Location of the bit shifted into Src.

Length	Input	UINT	Length contains the number of bits in the Src to be shifted. Supports shifting across array elements. • For BOOL data type, number of Booleans in the array to be shifted. • For 16- and 32-bit data types, bits are shifted in multiples of 16 (such as 16, 32, and 64). If Length is not an even multiple of 16, the number of shifted bits is sent to the next 16-bit boundary. • Length is based on the size of the data type. If Length exceeds the range, an error occurs, 'Source offset exceeds the size of the array.' Length values: • BOOL: 1 • 16-bit word: 1-16 • 32-bit word: 1-32 • 64-bit word: 1-64
Done	Output	BOOL	When TRUE, operation completed successfully. When FALSE, operation encountered an error condition.
Unload	Output	BOOL	Bit shifted out from Src address.
Error	Output	BOOL	When a fault occurs, Error is set to true.
ErrorID	Output	USINT	When a fault occurs, ErrorID contains the error code.

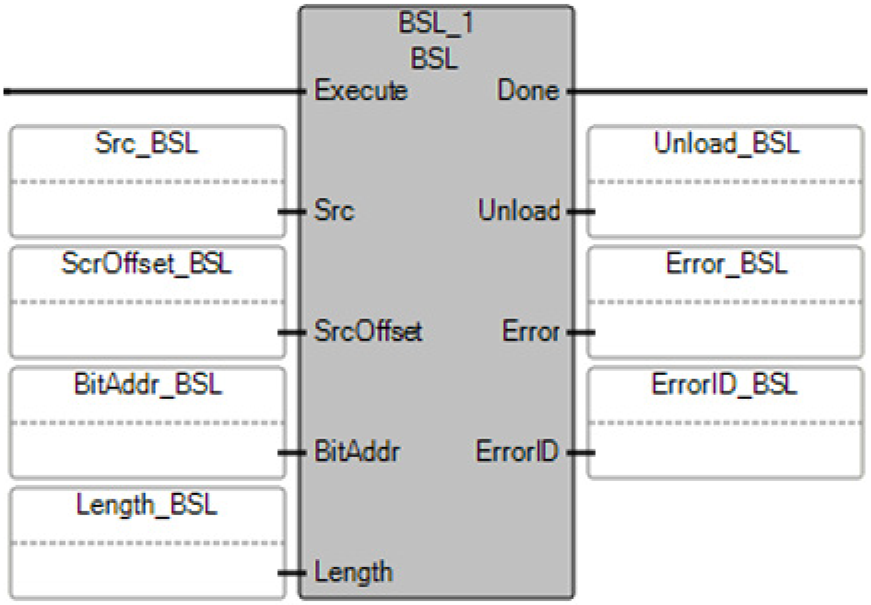
BSL error codes

Error code	Error description
01	Not supported dimension.
02	Data type not supported.
03	Length of bits exceeds 2048.
04	Source offset exceeds the size of the array.
05	Length of bits exceeds the size of the array.
07	Invalid parameters.

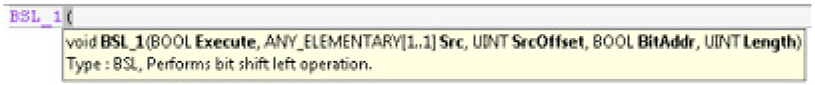
BSL Function Block Diagram example



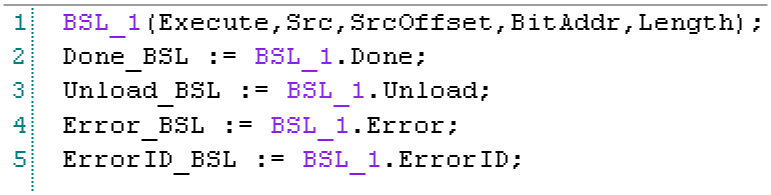
BSL Ladder Diagram example



BSL Structured Text example

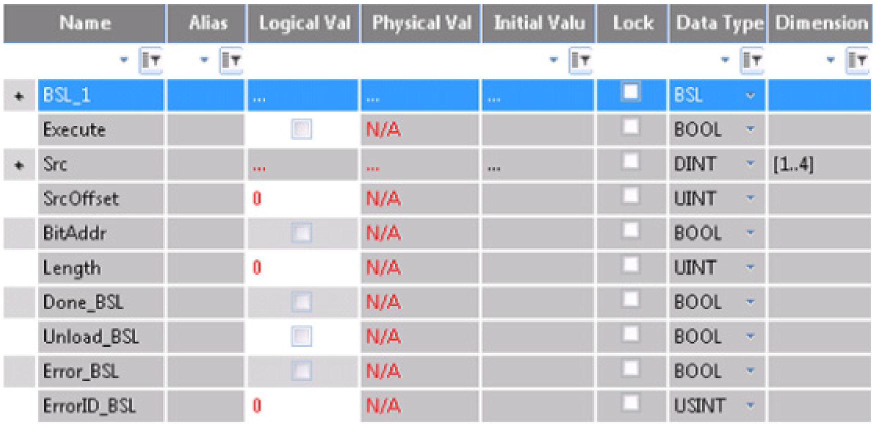


```
BSL_1(
void BSL_1(BOOL Execute, ANY_ELEMENTARY[1..1] Src, UINT SrcOffset, BOOL BitAddr, UINT Length)
Type : BSL, Performs bit shift left operation.
```



```
BSL_1(Execute,Src,SrcOffset,BitAddr,Length);
Done_BSL := BSL_1.Done;
Unload_BSL := BSL_1.Unload;
Error_BSL := BSL_1.Error;
ErrorID_BSL := BSL_1.ErrorID;
```

Results



	Name	Alias	Logical Val	Physical Val	Initial Valu	Lock	Data Type	Dimension
+	BSL_1		...	...	...	□	BSL	
	Execute		□	N/A		□	BOOL	
+	Src		...	...	...	□	DINT	[1..4]
	SrcOffset		0	N/A		□	UINT	
	BitAddr		□	N/A		□	BOOL	
	Length		0	N/A		□	UINT	
	Done_BSL		□	N/A		□	BOOL	
	Unload_BSL		□	N/A		□	BOOL	
	Error_BSL		□	N/A		□	BOOL	
	ErrorID_BSL		0	N/A		□	USINT	

BSR (bit shift right)

Shifts a bit in an array element to the right.

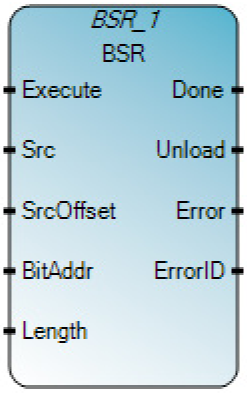
Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.

Operation details:

The BSR instruction is an immediate process on false-to-true rung transition and updates output synchronously. When Execute is TRUE, the right most bit (bit 0 of the element addressed by Src + SrcOffset) is copied into the Unload bit and all bits in the array or non-array are shifted right by one bit. Length and 16 bit boundary are considered except for BOOL data types. The external bit is then moved to bit 0 (Src + SrcOffset) of the first element.

For wraparound operations, set the position of the BitAddr to the last bit position or to the Unload bit. Possible usage of the BSL instruction, track bottles through a bottling line where each bit represents a bottle.



Use this table to help determine the parameter values for this instruction.

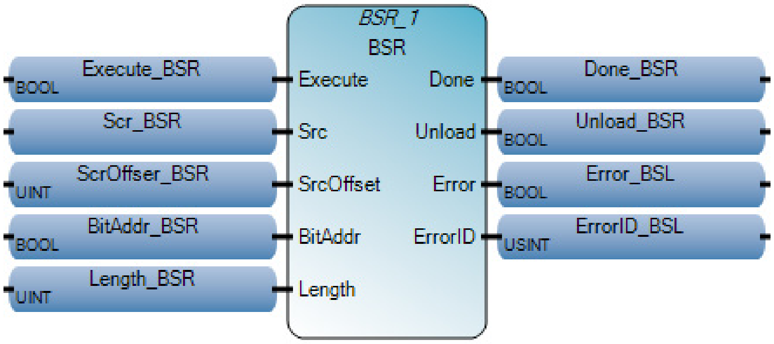
Parameter	Parameter Type	Data Type	Description

Execute	Input	BOOL	Instruction enable. TRUE - Rising Edge detected, shifts bit to the right one position. FALSE - Rising Edge not detected, do not enable BSR operation.
Scr	Input	ANY_ELEMENTARY	The address of the Src (bit) to be shifted. Supported data types: BOOL, DWORD, INT, UINT, WORD, DINT and UDINT. • Arrays: Set Scr to a variable based address such as: Source1, Source1[0], or Source1[1]. • Non-arrays: Set Scr to a variable address such as Source1.
SrcOffset	Input	UINT	If SrcOffset is 0, start from the first element. • Arrays: Set SrcOffset to 0. If set to Source1[0] or Source1[1] an error occurs: 'Source offset exceeds the size of the array.' • Non-arrays: Set SrcOffset to 0 or an error occurs: 'Source offset exceeds the size of the array.'
BitAddr	Input	BOOL	Location of the bit shifted into Src.
Length	Input	UINT	Length contains the number of bits in the Src to be shifted. Supports shifting across array elements. • For BOOL data type, number of Booleans in the array to be shifted. • For 16- and 32-bit data types, bits are shifted in multiples of 16 (such as 16, 32, and 64). If Length is not an even multiple of 16, the number of shifted bits is to the next 16-bit boundary. • Length is based on the size of the data type. If Length exceeds the range, an error occurs, 'Source offset exceeds the size of the array.' Length values: • BOOL: 1 • 16-bit word: 1-16 • 32-bit word: 1-32 • 64-bit word: 1-64
Done	Output	BOOL	When TRUE, operation completed successfully. When FALSE, operation encountered an error condition.
Unload	Output	BOOL	Bit shifted out from Src address.
Error	Output	BOOL	When a fault occurs, Error is set to true.
ErrorID	Output	USINT	When a fault occurs, ErrorID contains error code.

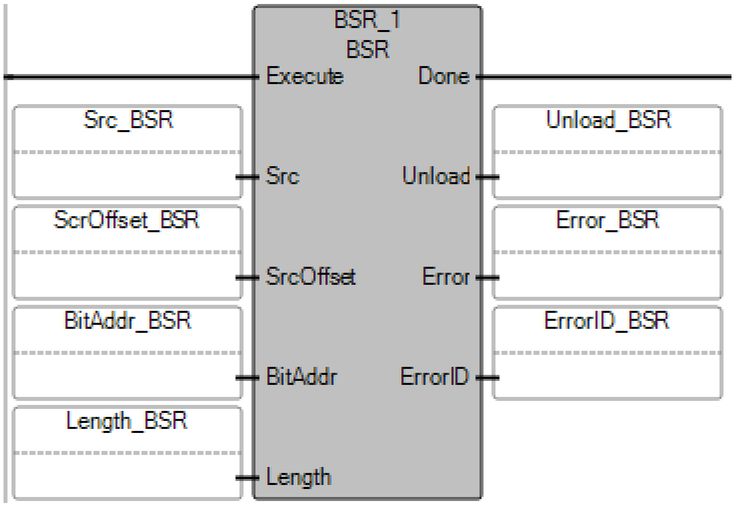
BSR error codes

Error code	**Error description**
01	Not supported dimension.
02	Data type not supported.
03	Length of bits exceeds 2048.
04	Source offset exceeds the size of the array.
05	Length of bits exceeds the size of the array.
07	Invalid parameters.

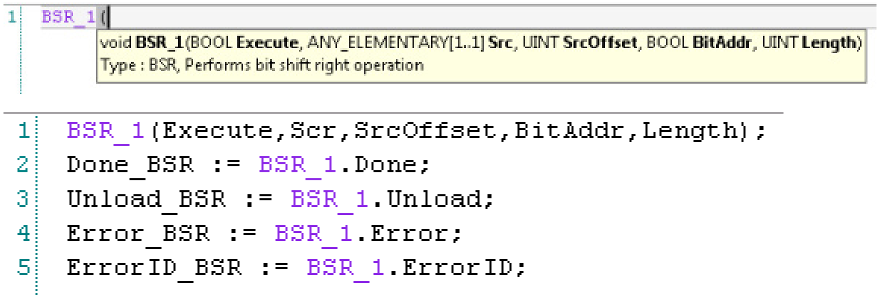
BSR Function Block Diagram example



BSR Ladder Diagram example



BSR Structured Text example

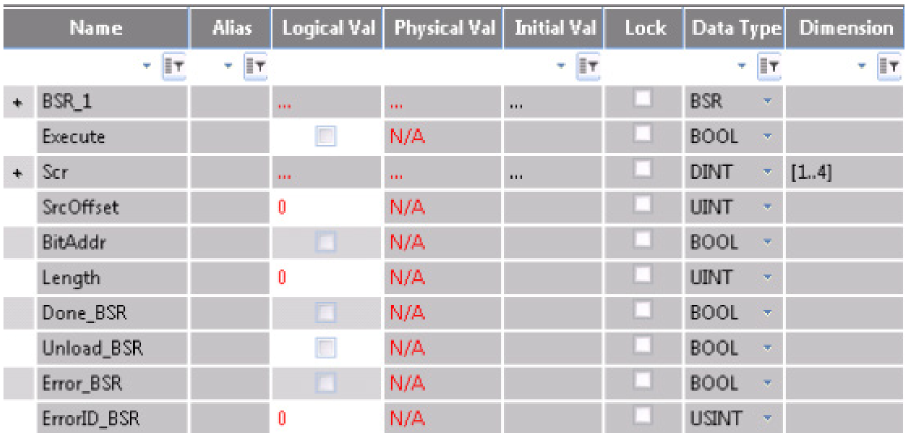


```
BSR_1(
```

void **BSR_1**(BOOL **Execute**, ANY_ELEMENTARY[1..1] **Src**, UINT **SrcOffset**, BOOL **BitAddr**, UINT **Length**)
Type : BSR, Performs bit shift right operation

```
BSR_1(Execute,Scr,SrcOffset,BitAddr,Length);
Done_BSR := BSR_1.Done;
Unload_BSR := BSR_1.Unload;
Error_BSR := BSR_1.Error;
ErrorID_BSR := BSR_1.ErrorID;
```

Results



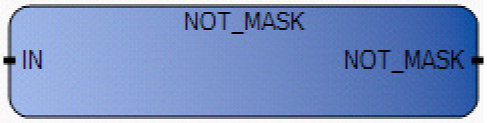
	Name	Alias	Logical Val	Physical Val	Initial Val	Lock	Data Type	Dimension
+	BSR_1		...	...	...	□	BSR	
	Execute		□	N/A		□	BOOL	
+	Scr		...	...	...	□	DINT	[1..4]
	SrcOffset		0	N/A		□	UINT	
	BitAddr		□	N/A		□	BOOL	
	Length		0	N/A		□	UINT	
	Done_BSR		□	N/A		□	BOOL	
	Unload_BSR		□	N/A		□	BOOL	
	Error_BSR		□	N/A		□	BOOL	
	ErrorID_BSR		0	N/A		□	USINT	

NOT_MASK (bit to bit NOT mask)

Integer bit-to-bit negation mask, inverts a parameter value.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

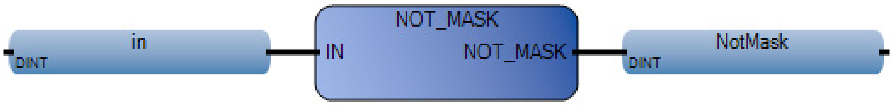
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



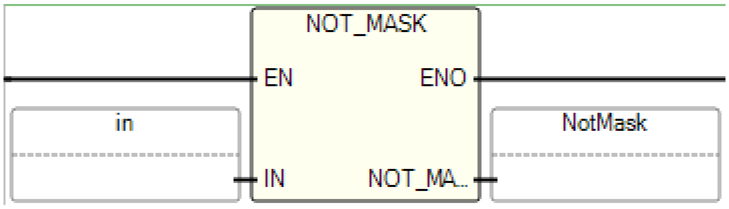
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute the bit-to-bit negation mask computation. FALSE - there is no computation. Applies to Ladder Diagram programs.
IN	Input	DINT	Must have integer format.
NOT_MASK	Output	DINT	Bit-to-bit negation on 32 bits of IN.
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

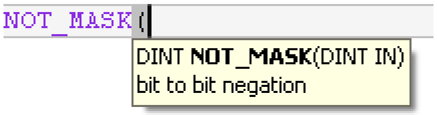
NOT_MASK Function Block Diagram example



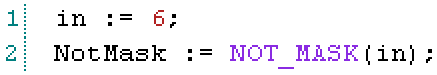
NOT_MASK Ladder Diagram example



NOT_MASK Structured Text example



```
NOT_MASK(
DINT NOT_MASK(DINT IN)
bit to bit negation
```



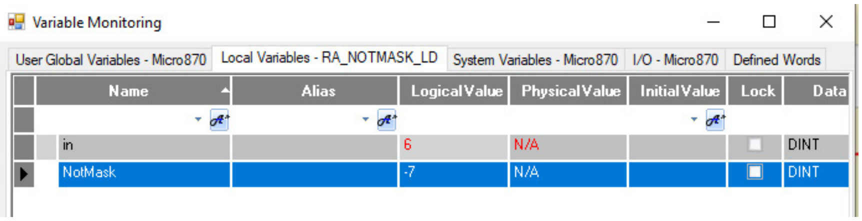
```
in := 6;
NotMask := NOT_MASK(in);
```

(*ST equivalence: *)

result := NOT_MASK (16#1234);

(* result is 16#FFFF_EDCB *)

Results


Variable Monitoring
User Global Variables - Micro870
Local Variables - RA_NOTMASK_LD
System Variables - Micro870
I/O - Micro870
Defined Words

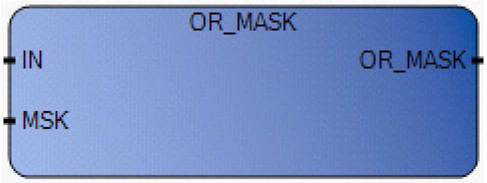
Name	Alias	Logical Value	Physical Value	Initial Value	Lock	Data
in		6	N/A		□	DINT
NotMask		-7	N/A		□	DINT

OR_MASK (bit to bit OR mask)

Integer OR bit-to-bit mask, turns bits on.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.

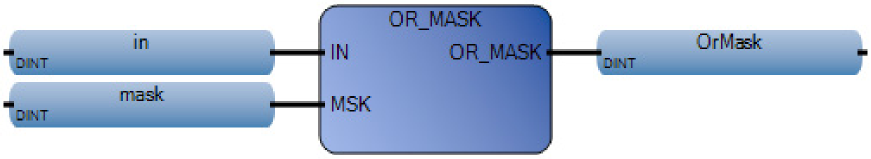


Use this table to help determine the parameter values for this instruction.

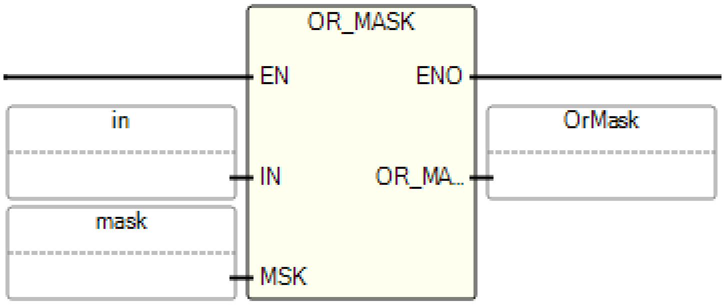
Parameter	Parameter Type	Data Type	Description

EN	Input	BOOL	Instruction enable. TRUE - execute the Integer OR bit-to-bit mask computation. FALSE - there is no computation. Applies to Ladder Diagram programs.
IN	Input	DINT	Must have integer format.
MSK	Input	DINT	Must have integer format.
OR_MASK	Output	DINT	Bit-to-bit logical OR between IN and MSK.
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

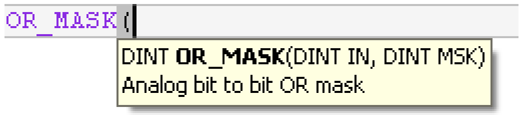
OR_MASK Function Block Diagram example



OR_MASK Ladder Diagram example

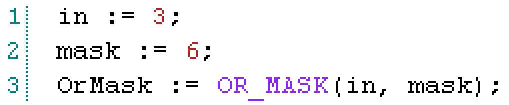


OR_MASK Structured Text examples



```
OR_MASK(
```

DINT **OR_MASK**(DINT IN, DINT MSK)
Analog bit to bit OR mask



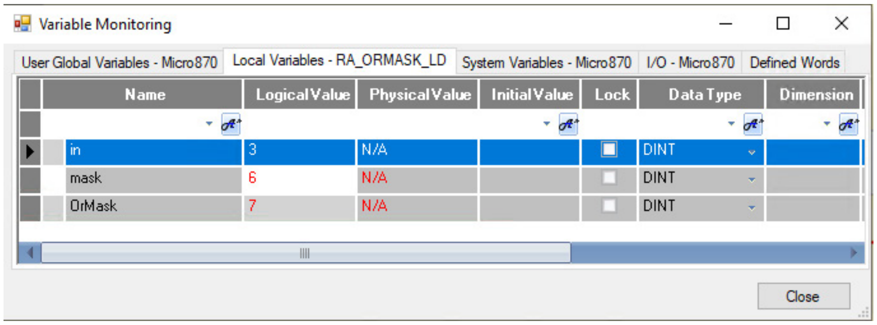
```
 in := 3;
mask := 6;
 OrMask := OR_MASK(in, mask);
```

(* ST Equivalence: *)

parity := OR_MASK (xvalue, 1); (* makes value always odd *)

result := OR_MASK (16#abc, 16#f0f); (* equals 16#fbf *)

Results



Variable Monitoring

User Global Variables - Micro870 | Local Variables - RA_ORMASK_LD | System Variables - Micro870 | I/O - Micro870 | Defined Words

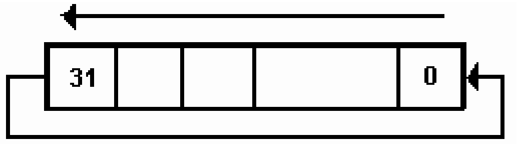
Name	Logical Value	Physical Value	Initial Value	Lock	Data Type	Dimension
in	3	N/A		☐	DINT	
mask	6	N/A		☐	DINT	
OrMask	7	N/A		☐	DINT	

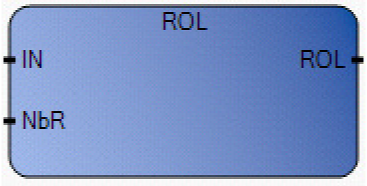
ROL (rotate left)

Rotates the DINT type input by NbR bits to the left in a circular form and fills the bits on the right with the bits that are rotated.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.

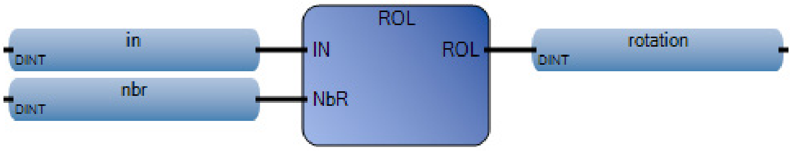




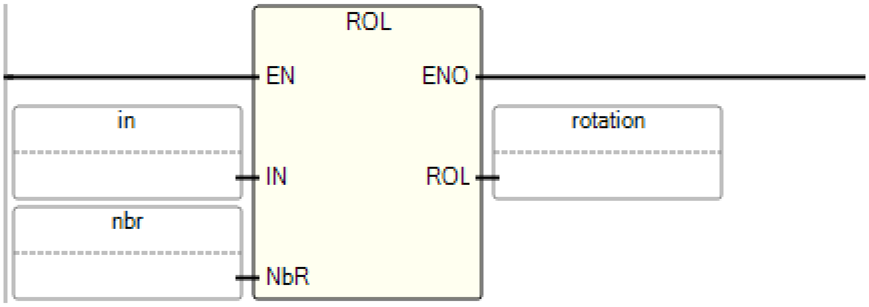
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute the rotate bits left integer value computation. FALSE - there is no computation. Applies to Ladder Diagram programs.
IN	Input	DINT	Integer value.
NbR	Input	DINT	Number of 1-bit rotations (in set [1..31]).
ROL	Output	DINT	Left rotated value. When NbR <= 0, no change occurs.
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

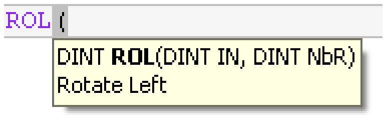
ROL Function Block Diagram example



ROL Ladder Diagram example



ROL Structured Text example


ROL (

DINT **ROL**(DINT IN, DINT NbR)
Rotate Left

```
in := 123;
nbr := 2;
rotation := ROL(in, nbr);
```

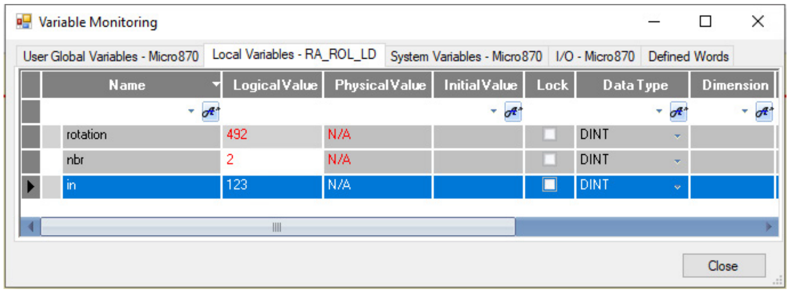
(* ST Equivalence: *)

result := ROL (register, 1);

(* register = 2#0100_1101_0011_0101*)

(* result = 2#1001_1010_0110_1010*)

Results


Variable Monitoring
User Global Variables - Micro870
Local Variables - RA_ROL_LD
System Variables - Micro870
I/O - Micro870
Defined Words
Close

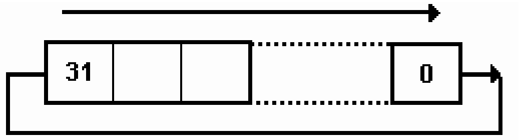
Name	Logical Value	Physical Value	Initial Value	Lock	Data Type	Dimension
rotation	492	N/A		□	DINT	
nbr	2	N/A		□	DINT	
in	123	N/A		□	DINT	

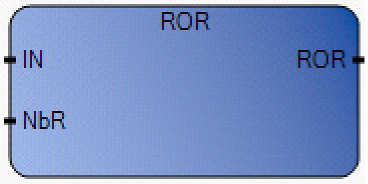
ROR (rotate right)

Rotate the DINT type input by NbR bits to right in a circular form and fills the bits on the left with the bits that are rorated.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.

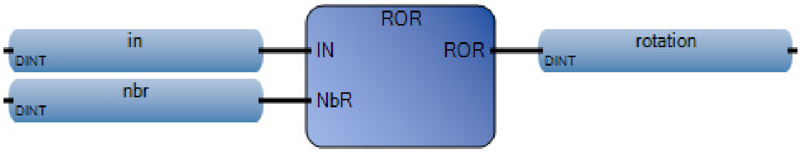




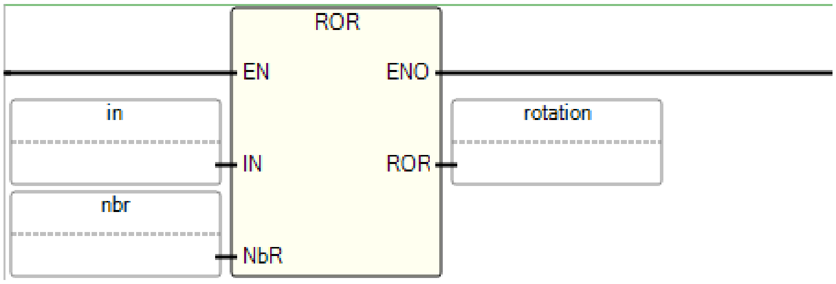
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute the rotate bits right integer value computation. FALSE - there is no computation. Applies to Ladder Diagram programs.
IN	Input	DINT	Any integer value.
NbR	Input	DINT	Number of 1-bit rotations (in set [1..31]).
ROR	Output	DINT	Right rotated value. There is no effect if NbR <= 0.
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

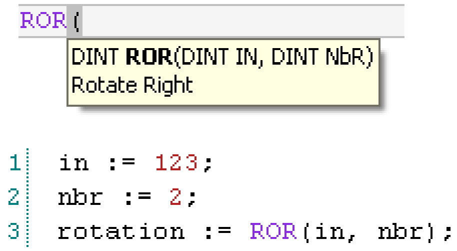
ROR Function Block Diagram example



ROR Ladder Diagram example



ROR Structured Text example



```
ROR(
DINT ROR(DINT IN, DINT NbR)
Rotate Right

in := 123;
nbr := 2;
rotation := ROR(in, nbr);
```

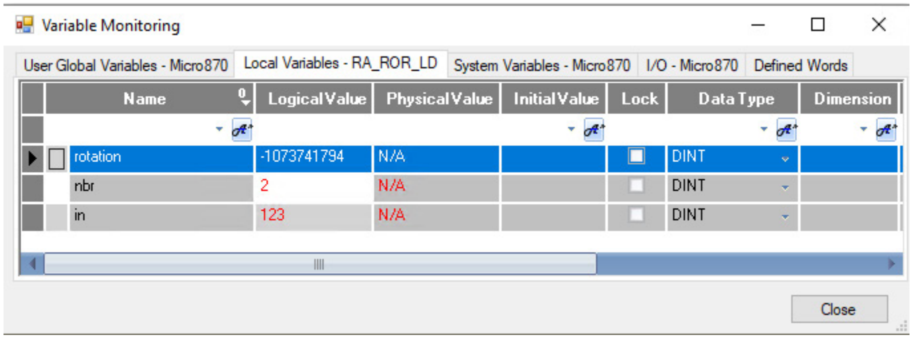
(* ST Equivalence: *)

result := ROR (register, 1);

(* register = 2#0100_1101_0011_0101 *)

(* result = 2#1010_0110_1001_1010 *)

Results



Variable Monitoring
User Global Variables - Micro870
Local Variables - RA_ROR_LD
System Variables - Micro870
I/O - Micro870
Defined Words
Close

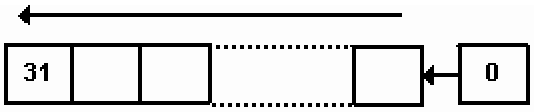
Name	Logical Value	Physical Value	Initial Value	Lock	Data Type	Dimension
rotation	-1073741794	N/A		□	DINT	
nbr	2	N/A		□	DINT	
in	123	N/A		□	DINT	

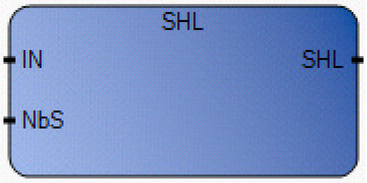
SHL (shift left)

For 32-bit integers, moves integers to the left and places 0 in the least significant bit.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



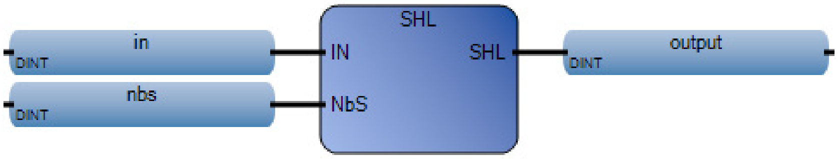


Use this table to help determine the parameter values for this instruction.

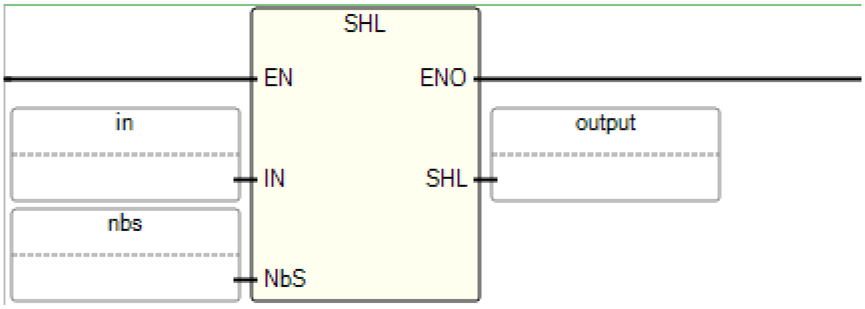
Parameter	Parameter Type	Data Type	Description

EN	Input	BOOL	Instruction enable. TRUE - move integers to the left. FALSE - there is no integer movement. Applies to Ladder Diagram programs.
IN	Input	DINT	Any integer value.
NbS	Input	DINT	Number of 1 bit shifts (in set [1..31]).
SHL	Output	DINT	Left shifted value. There is no effect if NbS <= 0. If a value of 0, replaces the least significant bit.
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

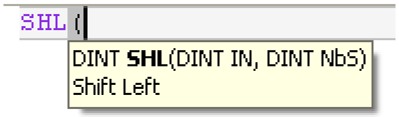
SHL Function Block Diagram example



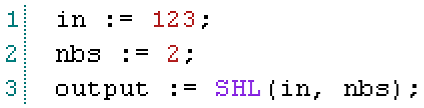
SHL Ladder Diagram example



SHL Structured Text example



```
SHL (
DINT SHL(DINT IN, DINT NbS)
Shift Left
```



```
in := 123;
nbs := 2;
output := SHL(in, nbs);
```

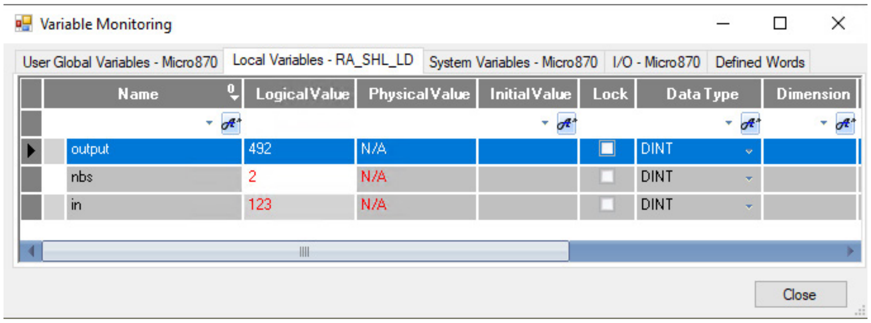
(* ST Equivalence: *)

result := SHL (register,1);

(* register = 2#0100_1101_0011_0101 *)

(* result = 2#1001_1010_0110_1010 *)

Results


Variable Monitoring
User Global Variables - Micro870
Local Variables - RA_SHL_LD
System Variables - Micro870
I/O - Micro870
Defined Words
Close

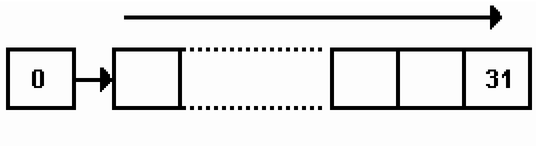
Name	Logical Value	Physical Value	Initial Value	Lock	Data Type	Dimension
output	492	N/A		□	DINT	
nbs	2	N/A		□	DINT	
in	123	N/A		□	DINT	

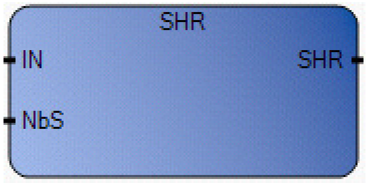
SHR (shift right)

Shifts the 32 bits of an integer to the right and replicates the leftmost bit (significant bit) to fill the vacant bits.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.

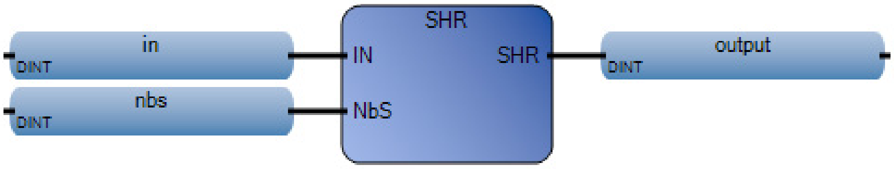




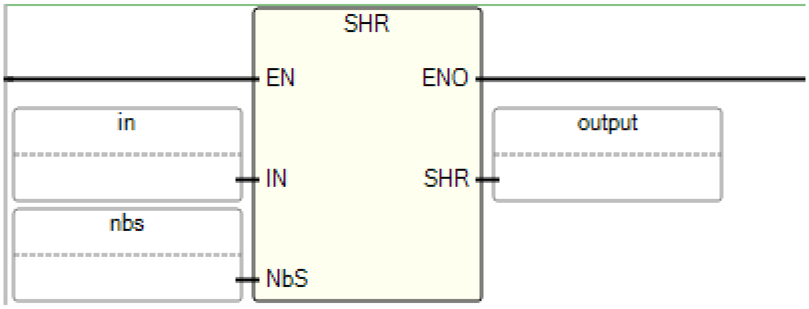
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - move integers to the right. FALSE - there is no integer movement. Applies to Ladder Diagram programs.
IN	Input	DINT	Any integer value.
NbS	Input	DINT	Number of 1 bit shifts (in set [1..31]).
SHR	Output	DINT	Right shifted value. There is no effect if NbS <= 0. If a value of 0, replaces the most significant bit.
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

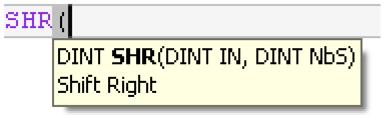
SHR Function Block Diagram example



SHR Ladder Diagram example

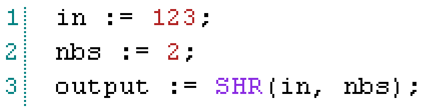


SHR Structured Text example



SHR (

DINT **SHR**(DINT IN, DINT NbS)
Shift Right



```
in := 123;
nbs := 2;
output := SHR(in, nbs);
```

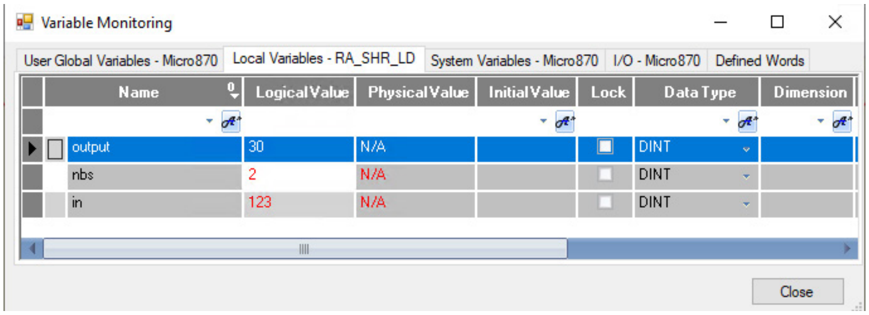
(* ST Equivalence: *)

result := SHR (register,1);

(* register = 2#1100_1101_0011_0101 *)

(* result = 2#0110_0110_1001_1010 *)

Results


Variable Monitoring
User Global Variables - Micro870
Local Variables - RA_SHR_LD
System Variables - Micro870
I/O - Micro870
Defined Words
Close

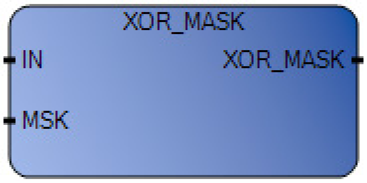
		Name	Logical Value	Physical Value	Initial Value	Lock	Data Type	Dimension
▶	☐	output	30	N/A		☐	DINT	
		nbs	2	N/A		☐	DINT	
		in	123	N/A		☐	DINT	

XOR_MASK (exclusive OR mask)

Integer exclusive OR bit-to-bit mask, returns inverted bit values.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

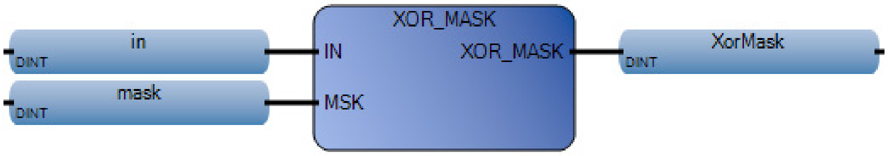
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



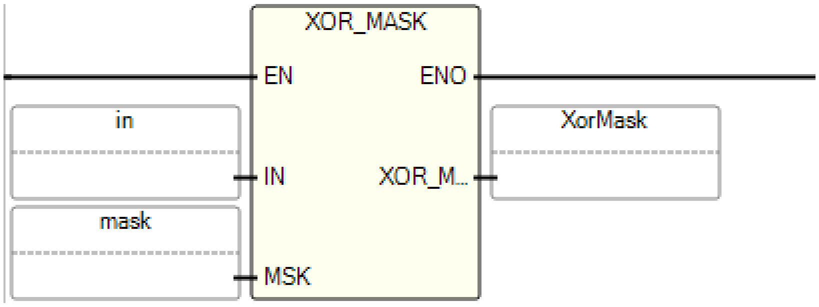
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - perform the exclusive OR bit-to-bit mask computation. FALSE - there is no computation. Applies to Ladder Diagram programs.
IN	Input	DINT	Must have integer format.
MSK	Input	DINT	Must have integer format.
XOR_MASK	Output	DINT	Bit-to-bit logical Exclusive OR between IN and MSK.
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

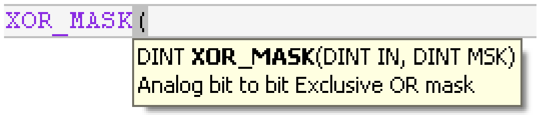
XOR_MASK Function Block Diagram example



XOR_MASK Ladder Diagram example

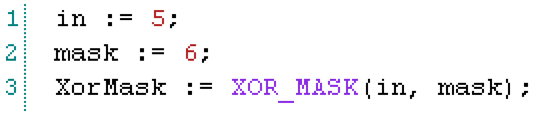


XOR_MASK Structured Text example



```
XOR_MASK(
```

DINT **XOR_MASK**(DINT IN, DINT MSK)
Analog bit to bit Exclusive OR mask



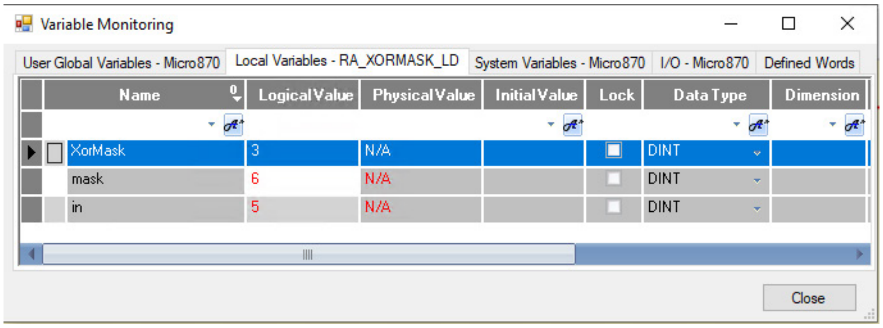
```
in := 5;
mask := 6;
XorMask := XOR_MASK(in, mask);
```

(* ST Equivalence: *)

crc32 := XOR_MASK (prevcrc, nextc);

result := XOR_MASK (16#012, 16#011); (* equals 16#003 *)

Results



Variable Monitoring
User Global Variables - Micro870
Local Variables - RA_XORMASK_LD
System Variables - Micro870
I/O - Micro870
Defined Words
Close

Name	Logical Value	Physical Value	Initial Value	Lock	Data Type	Dimension
XorMask	3	N/A		☐	DINT	
mask	6	N/A		☐	DINT	
in	5	N/A		☐	DINT	

Boolean instructions

Use Boolean instructions to determine an output value based on a logical calculation from inputs. The module outputs can be directly controlled from the program or independently controlled by the module using the Boolean instructions.

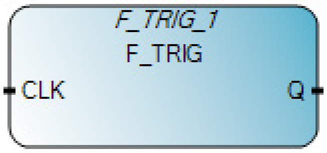
Function	**Description**
MUX4B on page 160	Multiplexer between four BOOL inputs, outputs a BOOL value.
MUX8B on page 156	Multiplexer between eight BOOL inputs, outputs a BOOL value.
TTABLE on page 153	Provides the value of the output based on the combination of inputs.
Function block	**Description**
F_TRIG on page 145	Detects a falling edge of a Boolean variable.
RS on page 148	Reset dominant (highest priority when determining instruction behavior) bistable.
R_TRIG on page 147	Detects a rising edge of a Boolean variable.
SR on page 152	Set dominant bistable.
Operator	**Description**
AND on page 150	Performs a boolean AND operation between two or more values.
NOT on page 151	Converts Boolean values to negated values.
XOR on page 151	Boolean exclusive OR of two values.
OR on page 149	Boolean OR of two or more values.

F_TRIG (falling edge detection)

Detects a falling edge of a Boolean variable. The F_TRIG block sets output Q for one cycle when input CLK toggles from set to cleared (i.e., a falling edge is detected at input CLK).

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.

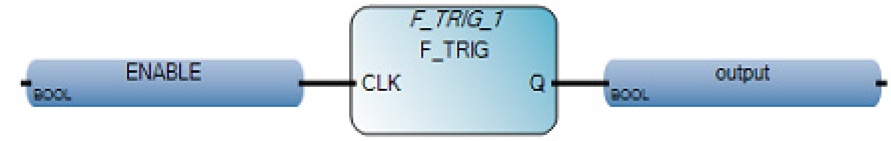


Use this table to help determine the parameter values for this instruction.

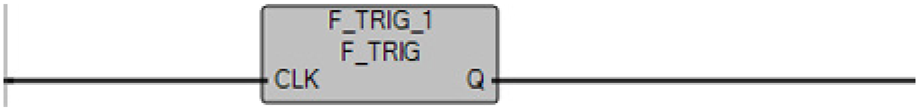
Parameter	**Parameter Type**	**Data Type**	**Description**

CLK	Input	BOOL	Checks the input for a falling edge. Any Boolean variable. TRUE = No falling edge detected. FALSE = Falling edge detected on input CLK, set output Q to TRUE.
Q	Output	BOOL	Indicates status for Q output. TRUE = Falling edge detected, sets output Q for one more cycle. FALSE = No change to output Q.

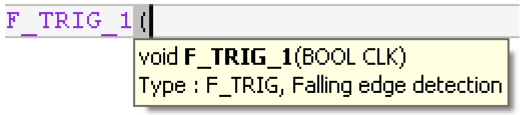
F_TRIG Function Block Diagram example



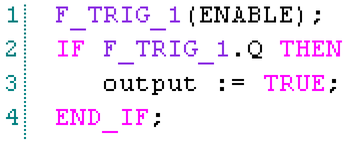
F_TRIG Ladder Diagram example



F_TRIG Structured Text example



```
F_TRIG_1(
void F_TRIG_1(BOOL CLK)
Type : F_TRIG, Falling edge detection
```

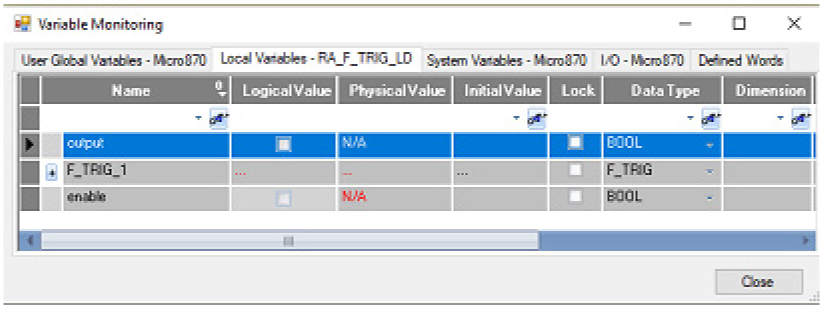


```
F_TRIG_1(ENABLE);
IF F_TRIG_1.Q THEN
    output := TRUE;
END_IF;
```

(* ST Equivalence: F_TRIG1 is an instance of a F_TRIG block *)

```
F_TRIG1(cmd);
nb_edge := ANY_TO_DINT(F_TRIG1.Q) + nb_edge;
```

Results


Variable Monitoring
User Global Variables - Micro870
Local Variables - RA_F_TRIG_LD
System Variables - Micro870
I/O - Micro870
Defined Words
Close

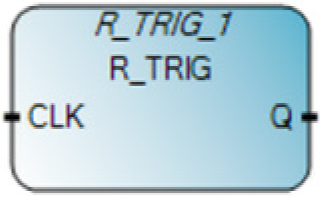
Name	Logical Value	Physical Value	Initial Value	Lock	Data Type	Dimension
output	□	N/A		□	BOOL	
F_TRIG_1	...	...	...	□	F_TRIG	
enable	□	N/A		□	BOOL	

R_TRIG (rising edge detector)

Detects a rising edge of a Boolean variable. The R_TRIG block sets output Q for one cycle when input CLK toggles from cleared to set (i.e., a rising edge is detected at input CLK).

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

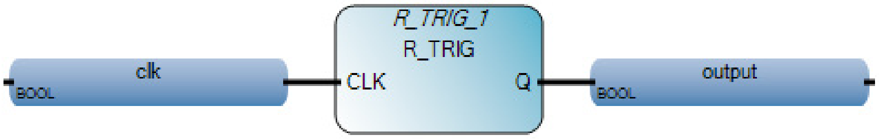
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



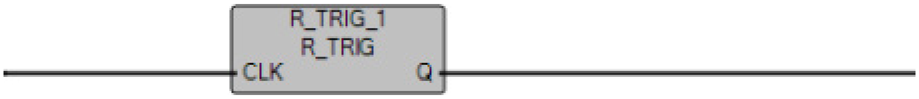
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
CLK	Input	BOOL	Any Boolean variable. TRUE - Rising Edge detected, set Q to TRUE. FALSE - no Rising Edge detected, set Q to FALSE.
Q	Output	BOOL	TRUE - when CLK is TRUE. FALSE - in all other cases.

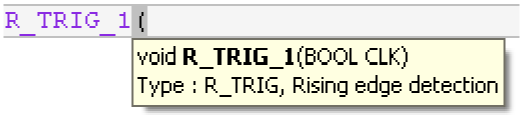
R_TRIG Function Block Diagram example



R_TRIG Ladder Diagram example



R_TRIG Structured Text example

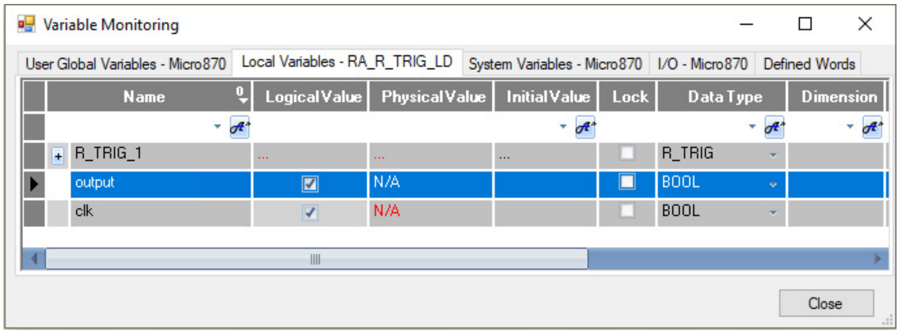


```
R_TRIG_1(
void R_TRIG_1(BOOL CLK)
Type : R_TRIG, Rising edge detection
```

(* ST Equivalence: R_TRIG1 is an instance of a R_TRIG block *)

```
R_TRIG1(cmd);
nb_edge := ANY_TO_DINT(R_TRIG1.Q) + nb_edge;
```

Results



Variable Monitoring

User Global Variables - Micro870 | Local Variables - RA_R_TRIG_LD | System Variables - Micro870 | I/O - Micro870 | Defined Words

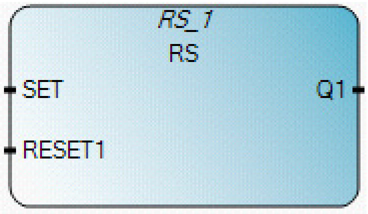
Name	Logical Value	Physical Value	Initial Value	Lock	Data Type	Dimension
R_TRIG_1	...	...	...	□	R_TRIG	
output	☑	N/A		□	BOOL	
clk	✓	N/A		□	BOOL	

RS (reset/set)

Resets or sets the dominant (highest priority when determining instruction behavior) bistable.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

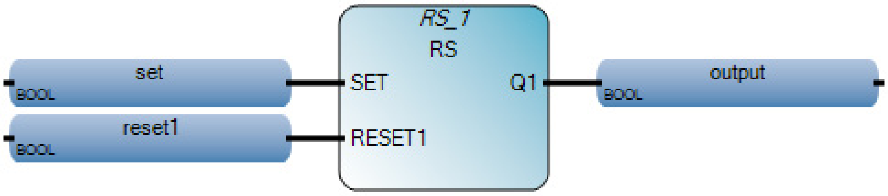
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



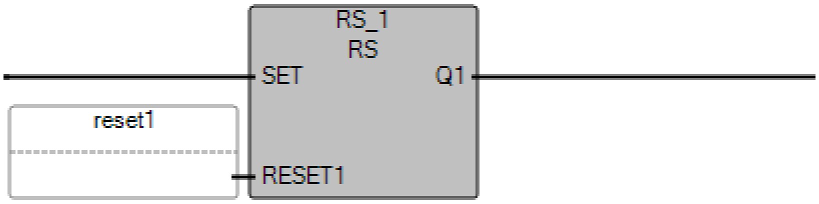
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
SET	Input	BOOL	TRUE - sets Q1 to TRUE.
RESET1	Input	BOOL	TRUE - resets Q1 to FALSE (dominant).
Q1	Output	BOOL	Boolean memory state.

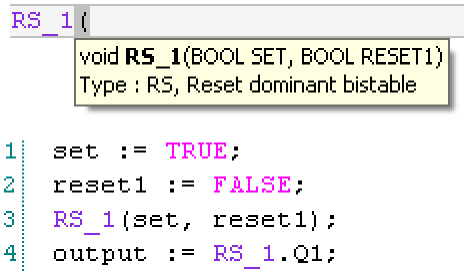
RS Function Block Diagram example



RS Ladder Diagram example



RS Structured Text example



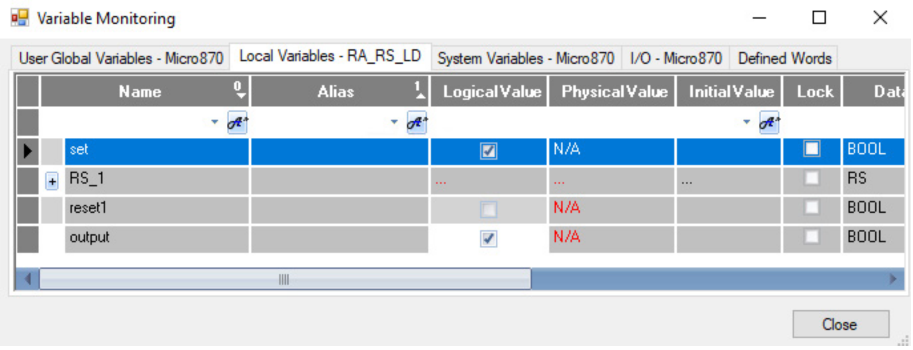
```
RS_1(
void RS_1(BOOL SET, BOOL RESET1)
Type : RS, Reset dominant bistable
```

```
set := TRUE;
reset1 := FALSE;
RS_1(set, reset1);
output := RS_1.Q1;
```

(* ST Equivalence: RS1 is an instance of a RS block *)

```
RS1(start_cmd, (stop_cmd OR alarm));
command := RS1.Q1;
```

Results


Variable Monitoring
User Global Variables - Micro870
Local Variables - RA_RS_LD
System Variables - Micro870
I/O - Micro870
Defined Words
Close

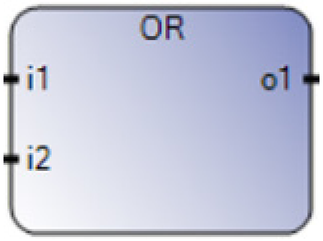
Name	Alias	Logical Value	Physical Value	Initial Value	Lock	Dat
set		☑	N/A		☐	BOOL
RS_1		...	...	...	☐	RS
reset1		☐	N/A		☐	BOOL
output		☑	N/A		☐	BOOL

OR

Performs a logical OR operation of two or more Boolean values and returns the Boolean value true if either input is true, otherwise returns false.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
i1	Input	BOOL	
i2	Input	BOOL	
o1	Output	BOOL	Boolean OR of the input terms. TRUE - When one or more of the inputs are TRUE. FALSE - When inputs are FALSE.

OR Structured Text example

(* ST equivalence: *)

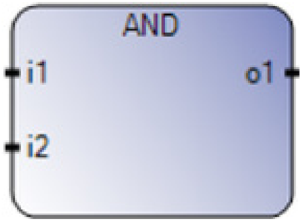
```
bo10 := bi101 OR NOT (bi102);
bo5 := (bi51 OR bi52) OR bi53;
```

AND

Performs a boolean AND operation between two or more values.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
i1	Input	BOOL	Value in Boolean data type.
i2	Input	BOOL	Value in Boolean data type.
o1	Output	BOOL	Result of the Boolean AND operation of the input values.

AND Structured Text example

(* ST equivalence: *)

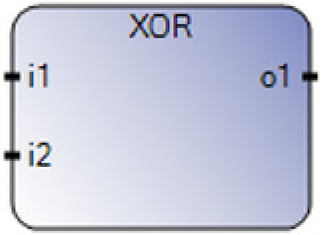
```
bo10 := bi101 AND NOT (bi102);
bo5 := (bi51 AND bi52) AND bi53;
```

XOR (exclusive OR)

Performs an exclusive OR operation of two Boolean values.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
i1	Input	BOOL	
i2	Input	BOOL	
o1	Output	BOOL	Boolean exclusive OR of the two input terms. TRUE - When either input is TRUE. FALSE - When both inputs are FALSE or both inputs are TRUE.

XOR Structured Text example

(* ST equivalence: *)

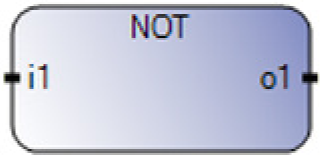
```
bo10 := bi101 XOR NOT (bi102);
bo5 := (bi51 XOR bi52) XOR bi53;
```

NOT

Converts Boolean values to negated values.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
i1	Input	BOOL	Any Boolean value or complex expression.
o1	Output	BOOL	TRUE when IN is FALSE. FALSE when IN is TRUE.

NOT Structured Text example

(* ST equivalence: *)

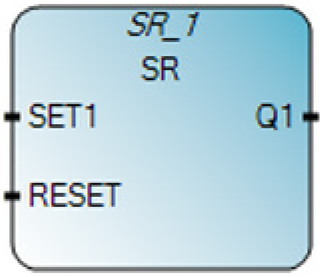
```
    bo10 := NOT (bi101);
```

SR (set/reset)

Set a dominant bistable.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



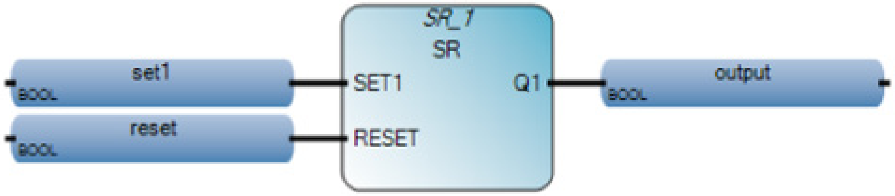
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
SET1	Input	BOOL	TRUE - sets Q1 to TRUE (dominant: highest priority when determining instruction behavior).
RESET	Input	BOOL	TRUE - resets Q1 to FALSE.
Q1	Output	BOOL	Boolean memory state. TRUE - when SET1 is TRUE. FALSE - when RESET is TRUE.

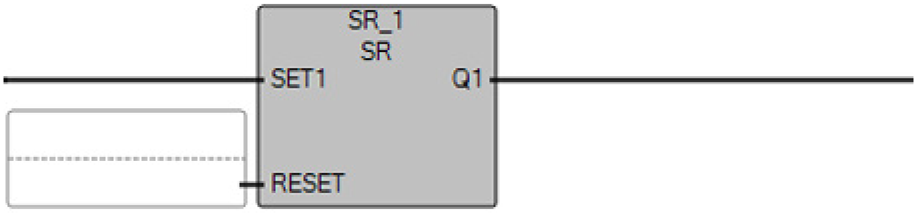
Dominant bistable example

Set1	Reset	Q1	Result Q1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

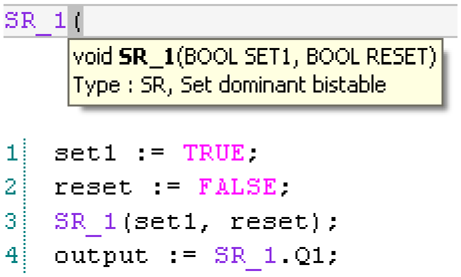
SR Function Block Diagram example



SR Ladder Diagram example



SR Structured Text example



```
SR_1(
```

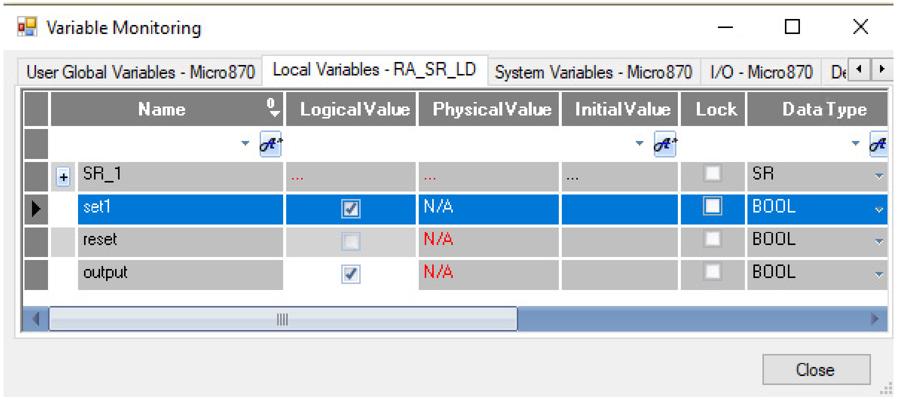
void **SR_1**(BOOL SET1, BOOL RESET)
Type : SR, Set dominant bistable

```
set1 := TRUE;
reset := FALSE;
SR_1(set1, reset);
output := SR_1.Q1;
```

(* ST Equivalence: SR1 is an instance of a SR block *)

```
SR1((auto_mode & start_cmd), stop_cmd);
command := SR1.Q1;
```

Results


Variable Monitoring
User Global Variables - Micro870
Local Variables - RA_SR_LD
System Variables - Micro870
I/O - Micro870
De
Close

Name	Logical Value	Physical Value	Initial Value	Lock	Data Type
SR_1	...	...	...	□	SR
set1	☑	N/A		□	BOOL
reset	□	N/A		□	BOOL
output	☑	N/A		□	BOOL

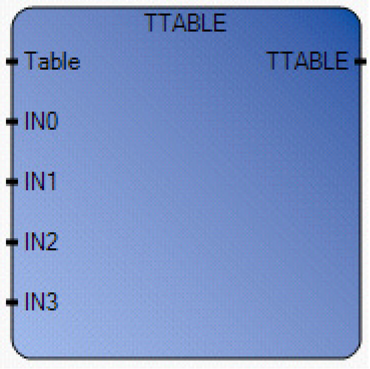
TTABLE (truth table)

Provides the value of the output based on the combination of inputs.

If the value is 0xABCD and In3 through In0 corresponds to the number 7, then TTABLE is the value of bit 7 in the table (which is 1). The least significant bit in the table is bit 0.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
Table	Input	UINT	Truth table of BOOLEAN function.
IN0	Input	BOOL	Any BOOL input value.
IN1	Input	BOOL	Any BOOL input value.
IN2	Input	BOOL	Any BOOL input value.
IN3	Input	BOOL	Any BOOL input value.
TTABLE	Output	BOOL	The value of the output according to the combination of inputs.

TTABLE input combinations

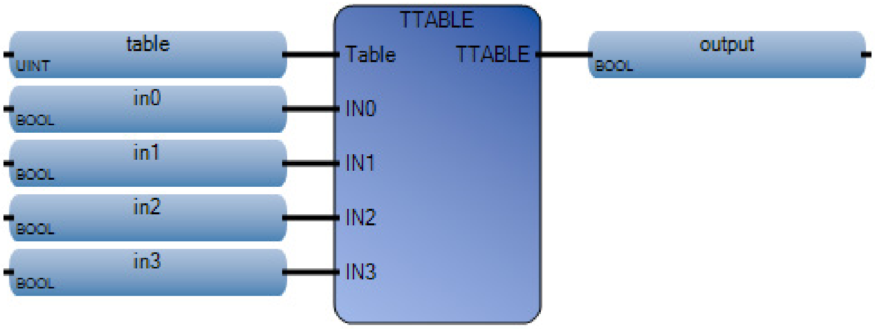
The TTABLE instruction has four inputs, and therefore 16 combinations. These combinations are found in a truth table; for each combination, the output value can be adjusted. The number of configurable combinations depends on the number of inputs connected to the function.

Truth Table combination example.

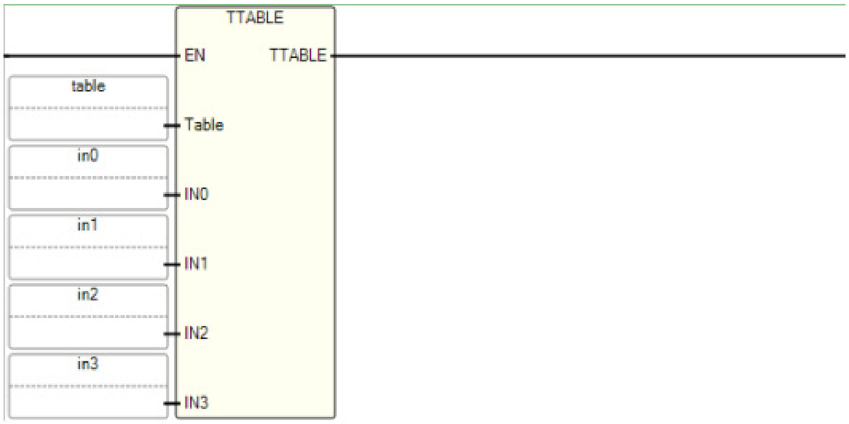
Number	In3	In2	In1	In0
1	0	0	0	0
2	0	0	0	1
3	0	0	1	0
4	0	1	1	1
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

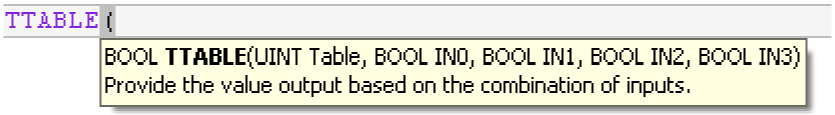
TTABLE Function Block Diagram diagram



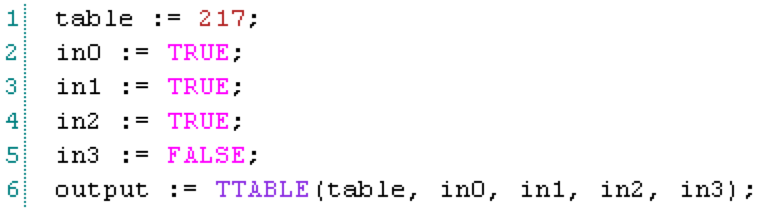
TTABLE Ladder Diagram example



TTABLE Structured Text example

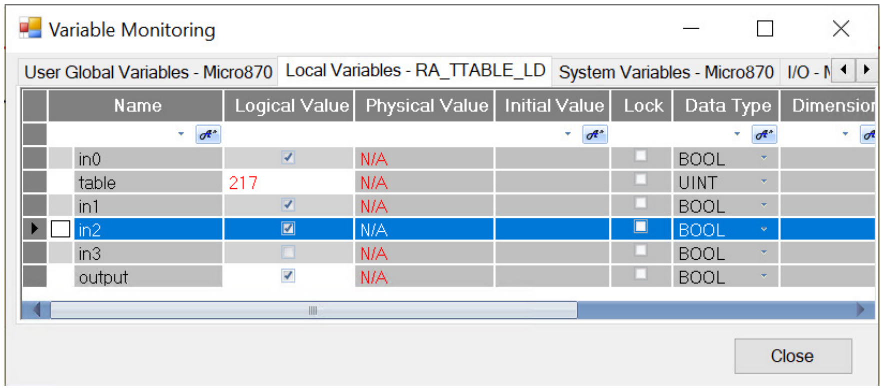


```
TTABLE(
BOOL TTABLE(UINT Table, BOOL IN0, BOOL IN1, BOOL IN2, BOOL IN3)
Provide the value output based on the combination of inputs.
```



```
table := 217;
in0 := TRUE;
in1 := TRUE;
in2 := TRUE;
in3 := FALSE;
output := TTABLE(table, in0, in1, in2, in3);
```

Results



Variable Monitoring
User Global Variables - Micro870
Local Variables - RA_TTABLE_LD
System Variables - Micro870
I/O - M
Close

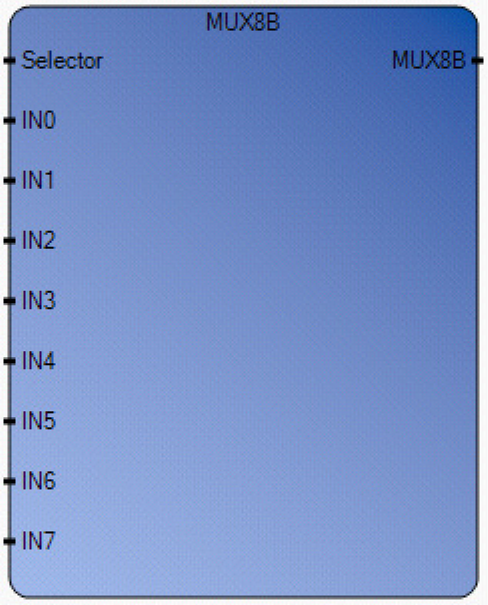
Name	Logical Value	Physical Value	Initial Value	Lock	Data Type	Dimension
in0	✓	N/A		□	BOOL	
table	217	N/A		□	UINT	
in1	✓	N/A		□	BOOL	
in2	☑	N/A		□	BOOL	
in3	□	N/A		□	BOOL	
output	✓	N/A		□	BOOL	

MUX8B (multiplexer of 8 BOOL inputs)

Multiplexer between eight BOOL inputs, outputs a BOOL value.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.

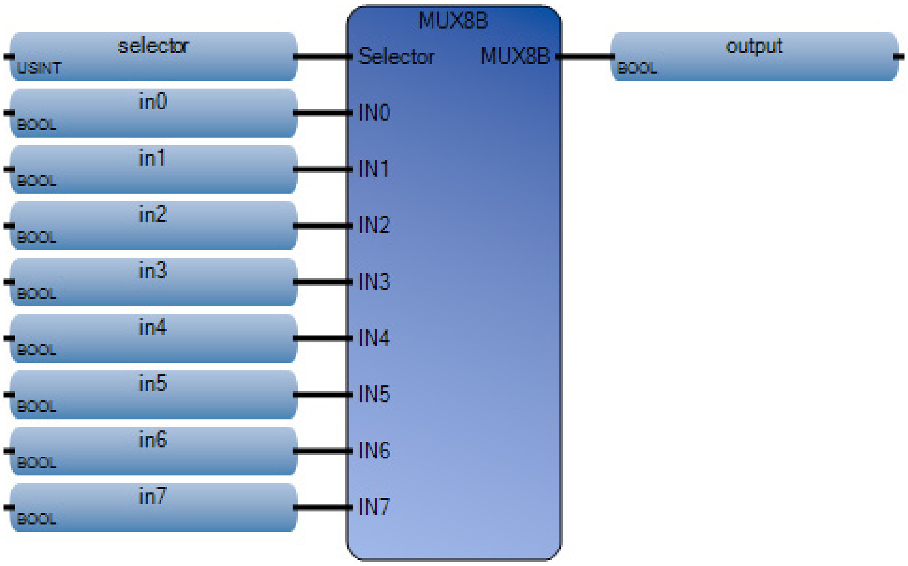


Use this table to help determine the parameter values for this instruction.

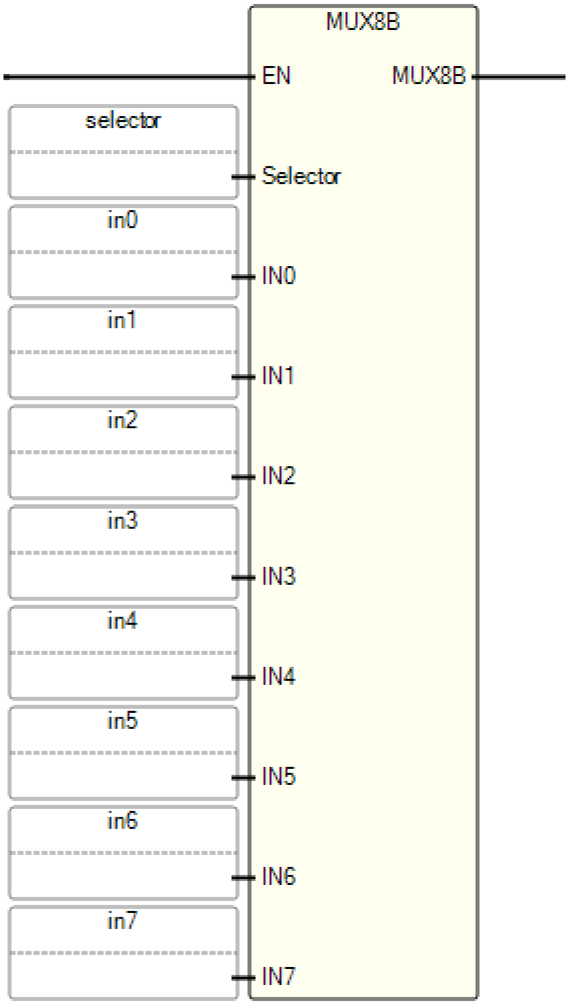
Parameter	Parameter Type	Data Type	Description
Selector	Input	USINT	Selector integer value, must be in set [0...7].
IN0	Input	BOOL	Any BOOL input value. TRUE - when Selector is 0. FALSE - when Selector is not 0.
IN1	Input	BOOL	Any BOOL input value. TRUE - when Selector is 1. FALSE - when Selector is not 1.
IN2	Input	BOOL	Any BOOL input value. TRUE - when Selector is 2. FALSE - when Selector is not 2.
IN3	Input	BOOL	Any BOOL input value. TRUE - when Selector is 3. FALSE - when Selector is not 3.
IN4	Input	BOOL	Any BOOL input value. TRUE - when Selector is 4. FALSE - when Selector is not 4.
IN5	Input	BOOL	Any BOOL input value. TRUE - when Selector is 5. FALSE - when Selector is not 5.
IN6	Input	BOOL	Any BOOL input value. TRUE - when Selector is 6. FALSE - when Selector is not 6.
IN7	Input	BOOL	Any BOOL input value. TRUE - when Selector is 7. FALSE - when Selector is not 7.

MUX8B	Output	BOOL	TRUE - when: • In0 if Selector = 0 • In1 if Selector = 1 • In2 if Selector = 2 • In3 if Selector = 3 • In4 if Selector = 4 • In5 if Selector = 5 • In6 if Selector = 6 • In7 if Selector = 7 FALSE - for all other values of the Selector.

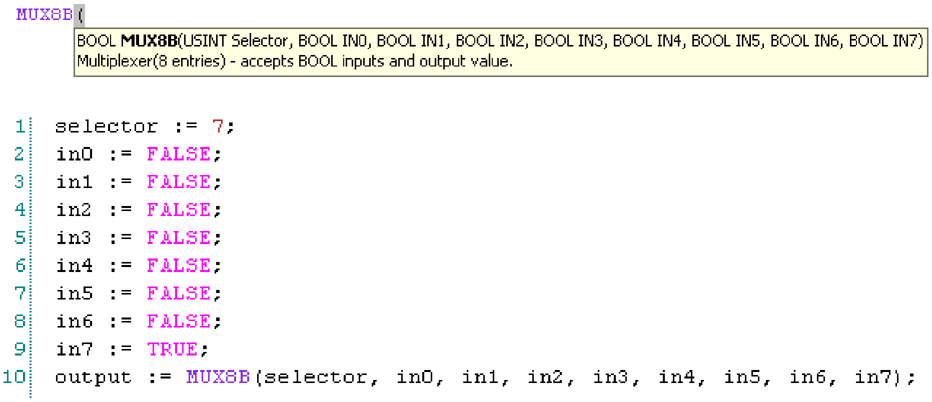
MUX8B Function Block Diagram example



MUX8B Ladder Diagram example



MUX8B Structured Text example



```
MUX8B(
```

BOOL **MUX8B**(USINT Selector, BOOL IN0, BOOL IN1, BOOL IN2, BOOL IN3, BOOL IN4, BOOL IN5, BOOL IN6, BOOL IN7)
Multiplexer(8 entries) - accepts BOOL inputs and output value.

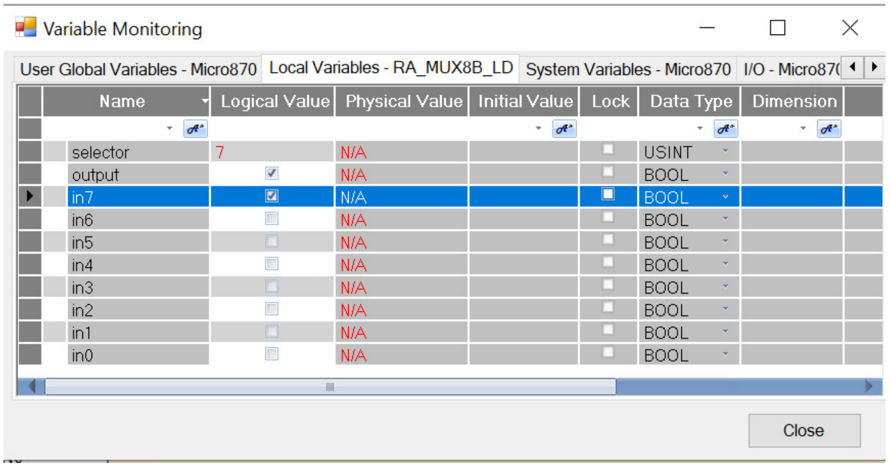
```
selector := 7;
in0  := FALSE;
in1  := FALSE;
in2  := FALSE;
in3  := FALSE;
in4  := FALSE;
in5  := FALSE;
in6  := FALSE;
in7  := TRUE;
output := MUX8B(selector, in0, in1, in2, in3, in4, in5, in6, in7);
```

(* ST Equivalence: *)

range := MUX8 (choice, 1, 5, 10, 50, 100, 500, 1000, 5000);

(* select from 8 predefined ranges, for example, if choice is 3, range will be 50 *)

Results


Variable Monitoring
User Global Variables - Micro870
Local Variables - RA_MUX8B_LD
System Variables - Micro870
I/O - Micro87(
Close

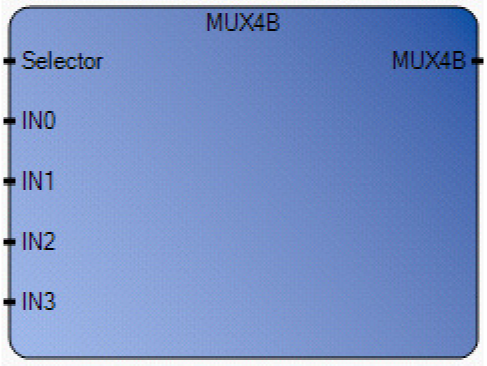
Name	Logical Value	Physical Value	Initial Value	Lock	Data Type	Dimension
selector	7	N/A		□	USINT	
output	☑	N/A		□	BOOL	
in7	☑	N/A		□	BOOL	
in6	□	N/A		□	BOOL	
in5	□	N/A		□	BOOL	
in4	□	N/A		□	BOOL	
in3	□	N/A		□	BOOL	
in2	□	N/A		□	BOOL	
in1	□	N/A		□	BOOL	
in0	□	N/A		□	BOOL	

MUX4B (multiplexer of 4 BOOL inputs)

Multiplexer between four BOOL inputs, outputs a BOOL value.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.

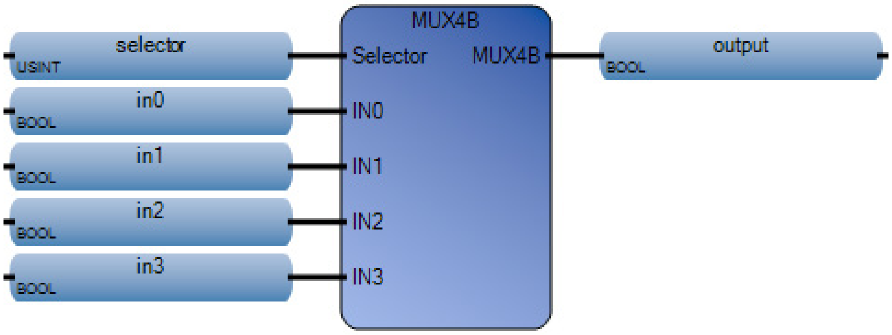


Use this table to help determine the parameter values for this instruction.

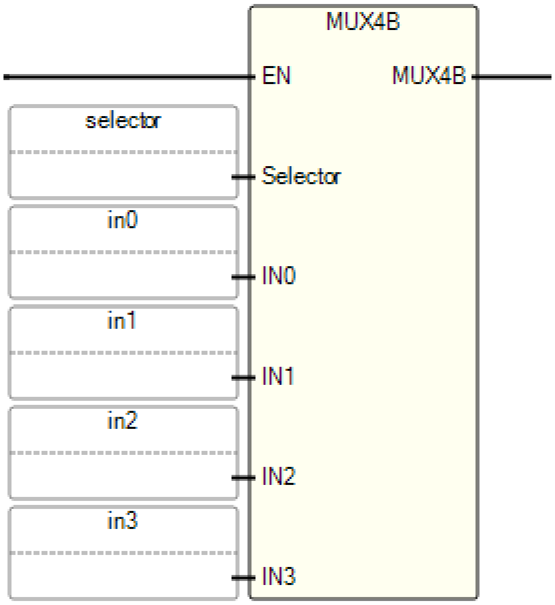
Parameter	Parameter Type	Data Type	Description
Selector	Input	USINT	Selector integer value, must be in set [0...3].
IN0	Input	BOOL	Any BOOL input value. TRUE - when Selector is 0. FALSE - when Selector is not 0.
IN1	Input	BOOL	Any BOOL input value. TRUE - when Selector is 1. FALSE - when Selector is not 1.

IN2	Input	BOOL	Any BOOL input value. TRUE - when Selector is 2. FALSE - when Selector is not 2.
IN3	Input	BOOL	Any BOOL input value. TRUE - when Selector is 3. FALSE - when Selector is not 3.
MUX4B	Output	BOOL	TRUE - when: ● In0 if Selector = 0 ● In1 if Selector = 1 ● In2 if Selector = 2 ● In3 if Selector = 3 FALSE - for all other values of the Selector.

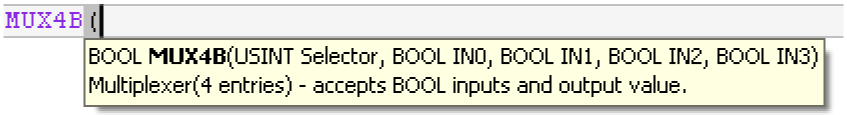
MUX4B Function Block Diagram example



MUX4B Ladder Diagram example

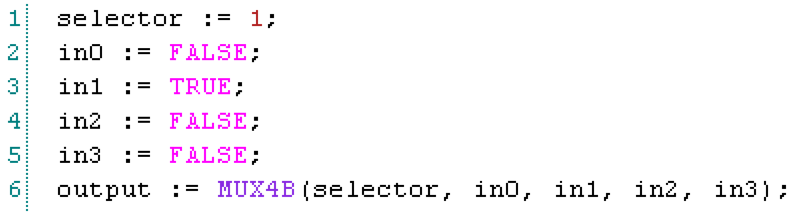


MUX4B Structured Text example



```
MUX4B(
```

BOOL **MUX4B**(USINT Selector, BOOL IN0, BOOL IN1, BOOL IN2, BOOL IN3)
Multiplexer(4 entries) - accepts BOOL inputs and output value.



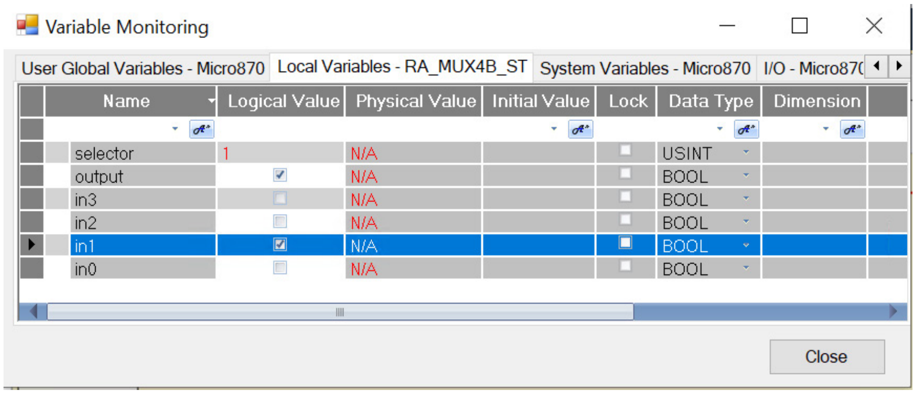
```
selector := 1;
in0 := FALSE;
in1 := TRUE;
in2 := FALSE;
in3 := FALSE;
output := MUX4B(selector, in0, in1, in2, in3);
```

(* ST Equivalence: *)

range := MUX4 (choice, 1, 10, 100, 1000);

(* select from 4 predefined ranges, for example, if choice is 1, range will be 10 *)

Results


Variable Monitoring
User Global Variables - Micro870
Local Variables - RA_MUX4B_ST
System Variables - Micro870
I/O - Micro870
Close

Name	Logical Value	Physical Value	Initial Value	Lock	Data Type	Dimension
selector	1	N/A		□	USINT	
output	☑	N/A		□	BOOL	
in3	□	N/A		□	BOOL	
in2	□	N/A		□	BOOL	
in1	☑	N/A		□	BOOL	
in0	□	N/A		□	BOOL	

Communication instructions

Use Communication instructions to read, write, compare, and convert communication strings.

Function block	Description
COM_IO_WDOG on page 163	monitors communications to the controller.
MSG_CIPGENERIC on page 166	Sends a CIP generic explicit message.
MSG_CIPSYMBOLIC on page 173	Sends a CIP symbolic explicit message.
MSG_MODBUS on page 177	Sends a Modbus message.
MSG_MODBUS2 on page 182	Sends a MODBUS/TCP message over an Ethernet Channel.

COM_IO_WDOG

Monitors external messaging to controller inputs and outputs. For example, if CIP write command to variable _IO_EM_DO_00 is not received over EtherNet/IP within the configured timeout, the watch dog timer will expire and all controller outputs are reset.

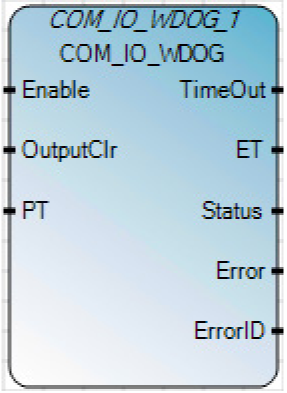
EtherNet/IP, Modbus TCP, and Modbus RTU protocols are supported.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator. It is only supported with firmware version 12.00 and later.

Note:

- Although multiple instances of this instruction can exist, only one instance can be enabled. Otherwise, an error will occur with ErrorID set to 2.
- Only controller embedded, plug-in, and expansion digital I/O are supported.



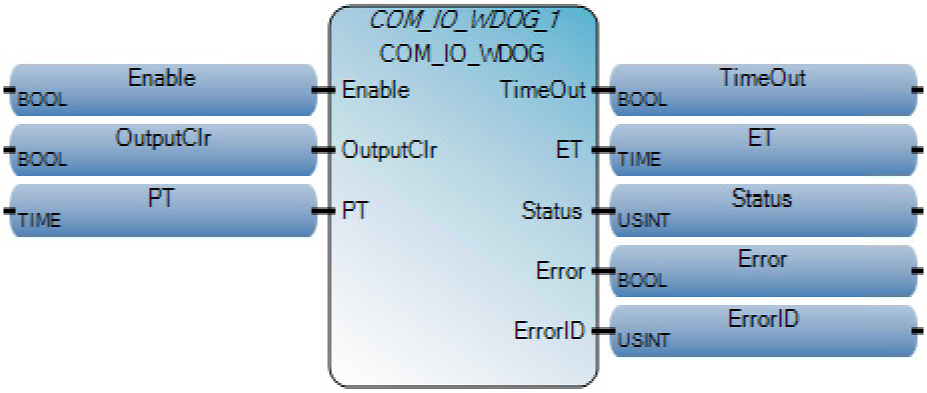
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
Enable	Input	BOOL	TRUE - These are the three cases of the TRUE condition. 1. On rising edge of TRUE, the function block starts to execute. Status bit is set to 0 and the rest of the bits are cleared. Then verify the below error conditions. If no errors occur, proceeds to the next step. • If PresetValue is less than one second, Error is set to TRUE and ErrorID is set to 1, Status bit 4 is set, TimeOut and ElapsedTime are cleared. • If the timer is already acquired by the other instance, Error is set to TRUE and ErrorID is set to 2, Status bit 4 is set, TimeOut and ElapsedTime are cleared. 2. PresetTime > ElapseTime. When Enable is set to TRUE and the function block starts to execute. Check if any commands are received. If any commands are received, reset the timer immediately. Set ElapsedTime to 0 and set Status bit 2. The rest of the bits are cleared. If no commands are received, set Status bit 1 and the rest of the bits are cleared. 3. PresetTime = ElapsedTime. When Enable is set to TRUE and the function block starts to execute. Check if any commands are received. If received, reset the timer immediately. Set ElapsedTime to 0 and set the Status bit 2. The rest of the bits are cleared. Else set the TimeOut bit, set the Status bit 3 and the rest of the bits are cleared. Set the internal variable to clear the digital outputs at the end of the scan if the configured action is to clear all the digital outputs at the end of the scan. FALSE - the instruction does not execute and function block outputs are cleared.
OutputClr	Input	BOOL	0 - Do nothing if timeout occurs. 1 - Clear all the digital outputs at the end of the scan (Embedded, EXIO and UPM) if timeout occurs.
PT	Input	TIME	Duration to wait before timeout. The value for a timeout cannot be less than one second, or an error occurs. The maximum value for PresetTime can be the maximum value within TIME data type.
TimeOut	Output	BOOL	TRUE - ElapsedTime equals to PresetTime. FALSE - Enable is set to FALSE; the timer is not elapsed or an error occurred.
ET	Output	TIME	The current elapsed time. The possible values range is from 0 ms to 1193h2m47s294ms.
Status	Output	USINT	Status of the function block. Bit 0 - Enable Bit 1 - Timer is running, no output or input has been received. Bit 2 - The output or input command has been received. Bit 3 - Timeout occurred. No output or input command has been received. Bit 4 - Error occurred. Other bits are reserved.
Error	Output	BOOL	Indicates an error occurred.
ErrorID	Output	USINT	When an error occurs, ErrorID contains the error code.

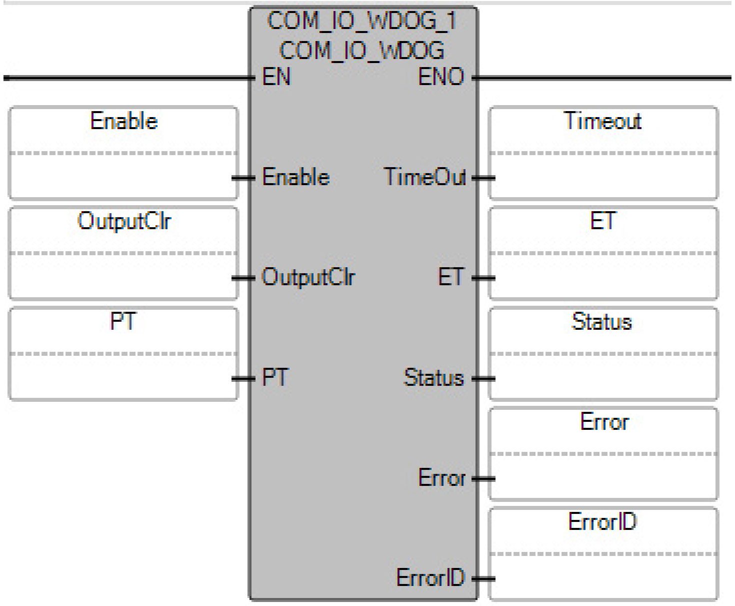
COM_IO_WDOG error code

ErrorID Code	Error description
1	The PresetTime is less than one second.
2	Another COM_IO_WDOG function block instance is already executing.

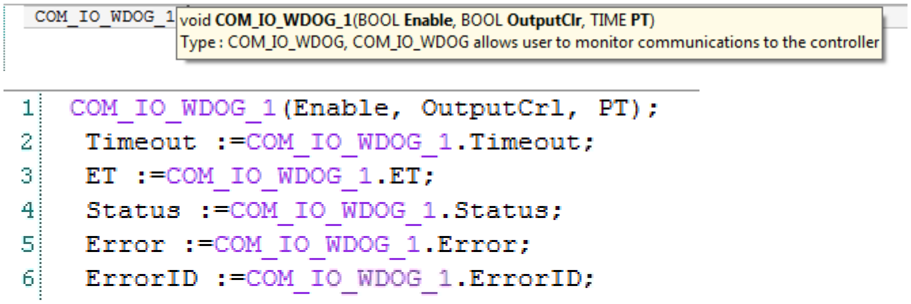
COM_IO_WDOG Function Block Diagram example



COM_IO_WDOG Ladder Diagram example



COM_IO_WDOG Structured Text example


COM_IO_WDOG_1

void **COM_IO_WDOG_1**(BOOL **Enable**, BOOL **OutputClr**, TIME **PT**)
Type : COM_IO_WDOG, COM_IO_WDOG allows user to monitor communications to the controller

```
COM_IO_WDOG_1(Enable, OutputCrl, PT);
 Timeout :=COM_IO_WDOG_1.Timeout;
 ET :=COM_IO_WDOG_1.ET;
 Status :=COM_IO_WDOG_1.Status;
 Error :=COM_IO_WDOG_1.Error;
 ErrorID :=COM_IO_WDOG_1.ErrorID;
```

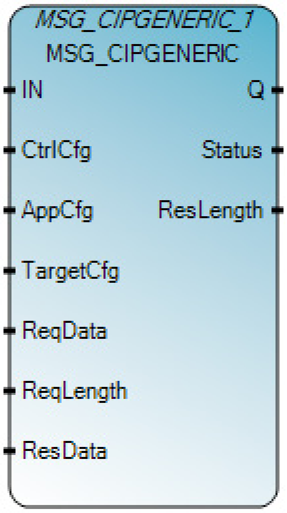
MSG_CIPGENERIC (common industrial protocol generic message)

Sends a common industrial protocol (CIP) explicit message over an Ethernet channel or a serial port.

A maximum of four message requests per channel can be processed in one scan. For Ladder Diagram programs, message requests are executed at the end of a ladder scan.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.

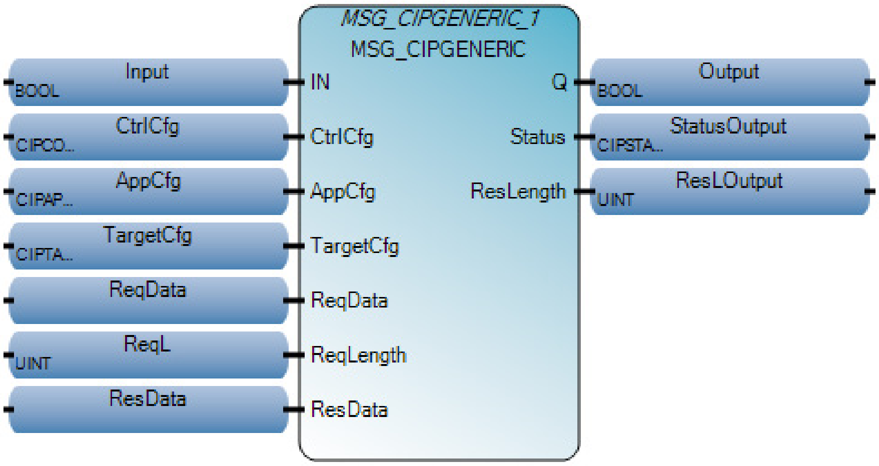


Use this table to help determine the parameter values for this instruction.

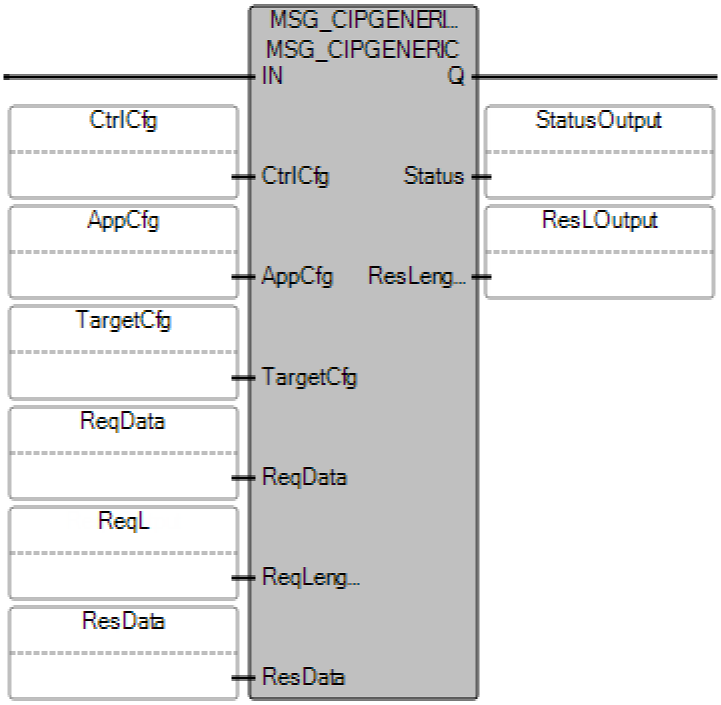
Parameter	Parameter Type	Data Type	Description
IN	Input	BOOL	Rung input state. TRUE - Rising Edge detected, start the instruction block with the precondition that the last operation has been completed. FALSE - Rising Edge not detected, idle.
CtrlCfg	Input	CIPCONTROLCFG	The instruction block execution control configuration. Use the CIPCONTROLCFG data type on page 168 parameters to define CtrlCfg.
AppCfg	Input	CIPAPPCFG	CIP service and application path (EPATH) configuration. Use the CIPAPPCFG data type on page 168 parameters to define AppCfg.
TargetCfg	Input	CIPTARGETCFG	Target device configuration. Use the CIPTARGETCFG data type on page 171 parameters to define TargetCfg.
ReqData	Input	USINT[1..1]	CIP message request data. The array size should be greater than the ReqLength size.
ReqLength	Input	UINT	CIP message request data length: • 0 - 490
ResData	Input	USINT[1..1]	CIP message response data. The array size should be greater than the ReqLength size. When a MSG is triggered or re-triggered, data in the ResData array is cleared.
Q	Output	BOOL	Outputs of this instruction are updated asynchronously from the program scan. Output Q cannot be used to re-trigger the instruction since IN is edge triggered. TRUE - MSG instruction finished successfully. FALSE - MSG instruction is not finished.
Status	Output	CIPSTATUS on page 169	The instruction block status. When a MSG is triggered, or re-triggered, all elements inside Status are reset. The Status output is defined in CIPSTATUS data type.

ResLength	Output	UINT	CIP message response data length: • 0 - 490 When a MSG is triggered, or re-triggered, ResLength is reset to 0.

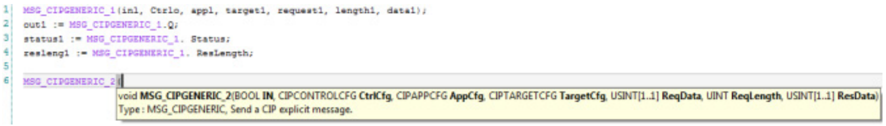
MSG_CIPGENERIC Function Block Diagram example



MSG_CIPGENERIC Ladder Diagram example



MSG_CIPGENERIC Structured Text example



```
MSG_CIPGENERIC_1(inl, Ctrlo, appl, targetl, requestl, lengthl, datal);
outl := MSG_CIPGENERIC_1.Q;
statusl := MSG_CIPGENERIC_1. Status;
reslengl := MSG_CIPGENERIC_1. ResLength;

MSG_CIPGENERIC_2(
```

void MSG_CIPGENERIC_2(BOOL IN, CIPCONTROLCFG CtrlCfg, CIPAPPCFG AppCfg, CIPTARGETCFG TargetCfg, USINT[1..1] ReqData, UINT ReqLength, USINT[1..1] ResData)
Type : MSG_CIPGENERIC, Send a CIP explicit message.

CIPAPPCFG data type

Use this table to help define the parameters for the CIPAPPCFG data type.

Parameter	Data type	Description
Service	USINT	Service code: 1 – 127
Class	UINT	Logical segment's Class ID value: 1 – 65535
Instance	UDINT	Logical segment's Instance ID value: 0 – 4294967295
Attribute	UINT	Logical segment's Attribute ID value: 1 - 65535, 0 - No Attribute ID used
MemberCnt	USINT	Members ID count. Maximum Member ID values used: 1 - 3, 0 - No Member ID used
MemberId	UINT[3]	Member ID values: 0 - 65535

CIPCONTROLCFG data type

Use this table to help determine the parameter values for the CIPCONTROLCFG data type.

Parameter	Data Type	Description
Cancel	BOOL	TRUE - Cancel the execution of the function block. Bit is cleared when the message is enabled. If the Cancel parameter is set, and the message is enabled (EN bit is set) and not done (DN bit is not set), then the message execution is cancelled and the ER bit is set.
TriggerType	USINT	Represents one of the following: • 0: Msg Triggered Once (when IN goes from False to True) • 1 to 65535: Cyclic trigger value in milliseconds. Msg is triggered periodically when IN is True. Set the value to 1 to trigger the MSG as quickly as possible.
StrMode	USINT	Reserved for future use.

CIP message triggering

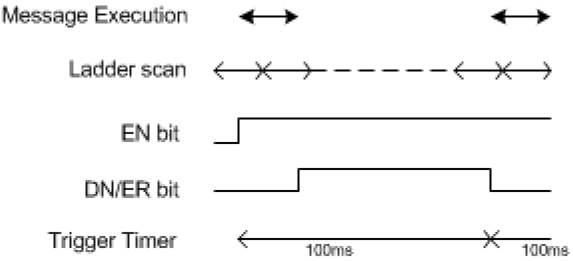
A CIP message can be triggered periodically by setting a non-zero value to the TriggerType parameter.

Use this table to help define the actions for the TriggerType parameter.

Action	Results
Message is enabled	Trigger timer starts
Trigger timer expires before the message completes	Message is immediately triggered in the next ladder scan cycle.
Message completes before the trigger time expires	Message is triggered when the trigger time expires.

Example: message triggering

In the following example, the TriggerType value is set to 100.



CIPSTATUS data type

Use this table to help determine the parameter values for the CIPSTATUS data type.

Parameter	Data type	Description
Error	BOOL	This bit is set to TRUE when the function block execution encounters an error condition.
ErrorID	UINT	Error code value. ErrorIDs are defined in CIPSTATUS error codes on page 170.
SubErrorID	UINT	Sub Error code value. SubErrorIDs defined in CIPSTATUS error codes.
ExtErrorID	UINT	CIP extended status error code value.
StatusBits	UINT	This parameter can be used to verify control bits: • Bit 0: EN – Enable • Bit 1: EW – Enable Wait • Bit 2: ST – Start • Bit 3: ER – Error • Bit 4: DN – Done • Bit 5: CIPCONN - CIP Connection Closure • Bit 6: EIPSESS - EIP Session Closure • Other bits are reserved StatusBits are defined for CIPSTATUS status bits on page 169.

CIPSTATUS status bits

The CIPSTATUS status bits are set based on the status of the message execution, the communication buffers, and the rung conditions.

-	-	-	-	-	-	-	-	-	-	-	4	3	2	1	0

Bit	Name	Description	Behavior
0	EN	Enable	Set when the rung goes true and remains set until either the DN bit or the ER bit is set and the rung goes false.
1	EW	Enable Waiting	Set when the communication buffer is allocated for the message request. Cleared when the ST bit is set.
2	ST	Start	Set when the message has been transmitted and is waiting for a reply. Cleared when the DN bit is set.
3	ER	Error	Set when message transmission fails. An error code is written to ErrorID. The ER bit and error code values are cleared the next time the rung goes from false to true.
4	DN	Done	Set when the message is transmitted successfully. The DN bit is cleared the next time the rung goes from false to true. When the Done bit is set, all other bits are cleared to indicate the MSG completed successfully. When an error is detected and the Error bit is set, the other status bits (EN/EW/ST) are not cleared.

Bit	Name	Description	Behavior
5	CIPCONN	Done	Set when the CIP Connection for the Communication is closed. The CIPCONN bit is applicable when ConnClose is True, for other cases the CIPCONN bit is False. The CIPCONN bit is also used for Serial, Ethernet and USB.
6	EIPSESS	Done	Set when the Encapsulation CIP Session for the Communication is closed. The EIPSESS bit is applicable when ConnClose is True, for other cases the EIPSESS bit is False. This bit is used for Ethernet only.

CIPSTATUS error codes

Use this table to help determine the parameter values for the ErrorID and SubErrorID fields of the CIPSTATUS parameter when the ER bit is set.

ErrorID code	SubErrorID	Error code description
33	Parameter configuration related errors	
	32	Bad Channel number.
	36	Unsupported CIP connection type.
	40	Unsupported CIP symbolic data type.
	41	Invalid CIP symbol name.
	43	Unsupported CIP Class value or MemberID count.
	48	The instruction block’s input data array size is not sufficient.
	49	Invalid target path.
	50	Bad service code.
	51	The instruction block’s transmit data array size is too big for CIP communication. The maximum length for the user data to be transmitted varies for different message configurations. If the total CIP message payload (including user data and CIP message overload) is beyond 504 bytes, an error 0x21 (subError 0x33) is reported.
	52	Bad Segment type value.
	53	Bad UCCM timeout value. If the encapsulation timeout value is less than the UCCM timeout or the difference between encapsulation timeout and UCCM time out is less than or equal to one second, an error 0x21 (subError 0x35) is reported.
	54	Bad connected timeout value. If the encapsulation timeout value is less than the CONNECTED message timeout or the difference between the encapsulation timeout and the CONNECTED message time out is less than or equal to one second, an error 0x21 (subError 0x36) is reported.
55	Timeout related errors	
	112	Message timed out while waiting in the message wait queue.
	113	Message timed out while waiting for the connection to the link layer to be established.
	114	Message timed out while waiting to transmit to the link layer.
	115	Message timed out while waiting for a response from the link layer.
69	Server response format related error codes	
	65	Message reply does not match request.
	68	Message reply data type not valid/supported. (MSG_CIPSYMBOLIC).
208	No IP address configured for the network.	
209	Maximum number of connections used – no connections available.	
210	Invalid internet address or node address.	
217	Message execution was canceled by user. (Cancel parameter was set to TRUE).	
218	No network buffer space available.	
222	Reserved.	
223	The Link address is not available. A TCP/IP or Ethernet configuration change is in progress.	

ErrorID code	SubErrorID	Error code description
224	CIP response error code. SubErrorID specifies the CIP status and ExtErrorID specifies the CIP extended status value. Refer to the CIP specification for possible error code values.	
255	Channel is shutdown or reconfiguration is in progress. Error code occurs immediately after power on until a connection is established, and is normal behavior. It may also occur in one of the following situations: • An Ethernet cable is disconnected • An IP address cannot be detected • A serial port plug-in is present but not configured	

CIPTARGETCFG data type

Use this table to help determine the parameter values for the CIPTARGETCFG data type.

Parameter	Data type	Description
Path	STRING[80]	Path for the target. A maximum of two hops can be specified. The path syntax is: • {"<port>,<node/slot address>"}2
CipConnMode	USINT	CIP Connection type. • 0 - Unconnected (default) • 1 - Class3 connection
UcmmTimeout	UDINT	Unconnected message timeout (in milliseconds). The amount of time to wait for a reply for unconnected messages, including connection establishment for connected message. • Valid values: 250-10,000 ms. • Set to 0 to use the default value of 3000 ms (3 seconds). • A value set to less than 250 ms will be set to 250 ms (minimum). • A value set to greater than 10,000 ms will be set to 10,000 ms (maximum).
ConnMsgTimeout	UDINT	Class3 Connection timeout (in milliseconds). The amount of time to wait for a reply for connected messages. The connection closes when the timeout expires. • Valid values: 800-10,000 ms. • Set to 0 to use the default value of 10,000 ms (10 seconds). • A value set to less than 800 will be set to 800 ms (minimum). • A value set to greater than 10,000 ms will be set to 10,000 ms (maximum).
ConnClose	BOOL	Connection closing behavior: • TRUE - Close the connection when the message completes. • FALSE - Do not close the connection when the message completed (default).

Target path for CIP messaging

The target path for CIP messaging contains parameters which determine the path and destination of the of the CIP message.

The target path string parameter uses the following syntax:

- "<local port>, <1st target's address>, [<1st target's local port>, <2nd target's address>]"

The 1st hop must be present; the 2nd hop is optional.

String element	Description

Local port	Local port used to send out the message. The port should be an active EtherNet/IP or CIP Serial port - USB ports are not supported.
1st Target address	Target address of the 1st hop. • For EIP, specify the target's IP address. The IP address should be a unicast address and should not be 0, multicast, broadcast, local address or a loop back (127.x.x.x) address. • For CIP Serial, specify the target's node address. The supported value is 1.
Local port of the 1st Target	Local port used to send out the message.
2nd Target address	Target address of the 2nd hop.

Target path example

The following table lists example values used in a target path string and describes the results for each string.

String element	Description
"0,0"	The target device is the local device.
"6,1"	Through Port 6 (Micro830 UPM Serial port) reach the Node at 1.
"4,192.168.1.100"	Through Port 4 (Micro850 embedded Ethernet port) reach the Node at 192.168.1.100.
"4,192.168.1.100,1,0"	Through Port 4 (Micro850 embedded Ethernet port) reach the Node at 192.168.0.100 (Logix ENET module). From ENET module, through the Backplane port (Port 1) reach the Logix controller at Slot 0.

CIP/EIP message connections

A maximum of 16 CIP (class 3) and 16 EIP connections are supported for client message execution. The following table describes the CIP/EIP connection behavior.

Scenario	Results
Message request is enabled and CipConnMode=1.	If a connection to the target does not exist, a CIP connection is established. If a connection to the target already exists, the existing CIP connection is used.
Message request is enabled, CipConnMode=1, and the message's local port is Ethernet.	If an EIP connection to the target does not exist, an EIP connection is established prior to establishing a CIP connection.
Message request is enabled, CipConnMode=0, and the message's local port is Ethernet.	If an EIP connection to the target does not exist, an EIP connection is established.
Message execution is completed, and ConnClose is set to True.	If there is only one connection to the target, the connection is closed. If there is more than one connection to the target, the connection is closed when the last message execution is completed. When a CIP connection is closed, any associated EIP connection is also closed. If more than one CIP connection uses the same EIP connection, the EIP connection will be closed after all associated CIP connections are closed.
When ConnClose is true, the message connection and EIP Session are closed upon completion of the message execution.	If more than one message shares the same connection then the connection is closed upon completion of the last message.
A CIP or EIP connection that is not associated with any active message is closed if it is idle for x seconds. Where x is a configurable Encapsulation Inactivity Timeout value that can be set using the CIP Set Service.	See CIP Specification Volume II TCP/IP objects for details regarding the CIP Set Service.

Scenario	Results
Message execution is completed, and ConnClose is set to False.	The connection is not closed.
Connection is not associated with an active message and remains idle for the amount of time specified in ConnTimeOut parameter.	The connection is closed.
Controller transitions from an executing mode (Run, Remote Run, Remote Test Single Scan and Remote Single Rung) to a non-executing mode.	All active connections are forcibly closed.

CIP message timeout timers

The following table describes how timers for CIPTARGETCFG timeout parameters (UcmmTimeout and ConnMsgTimeout) behave based on message requests and status.

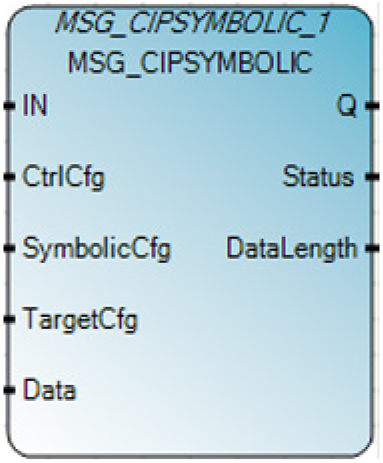
Action	Results
Message is enabled	UcmmTimeout timer is activated
Connection is requested	ConnMsgTimeout timer is activated
ConnMsgTimeout timer is active	UcmmTimeout timer is disabled
Connection request is completed	UcmmTimeout timer is reactivated

MSG_CIPSYMBOLIC (common industrial protocol symbolic message)

Sends a common industrial protocol (CIP) symbolic message over an Ethernet channel or a serial port.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



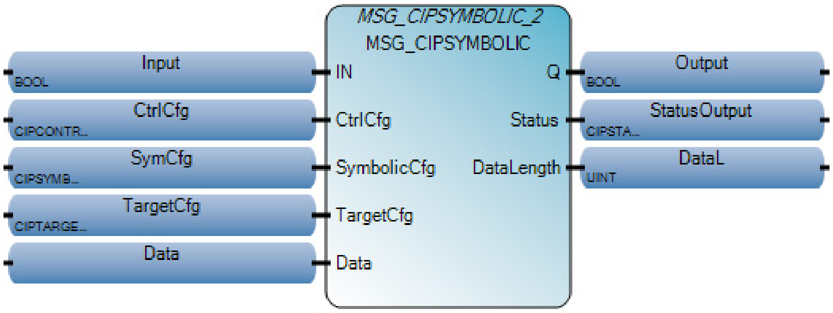
MSG_CIPSYMBOLIC operation

When the function block is enabled, the receive buffers for the Read operations are cleared on the rising edge of Enable.

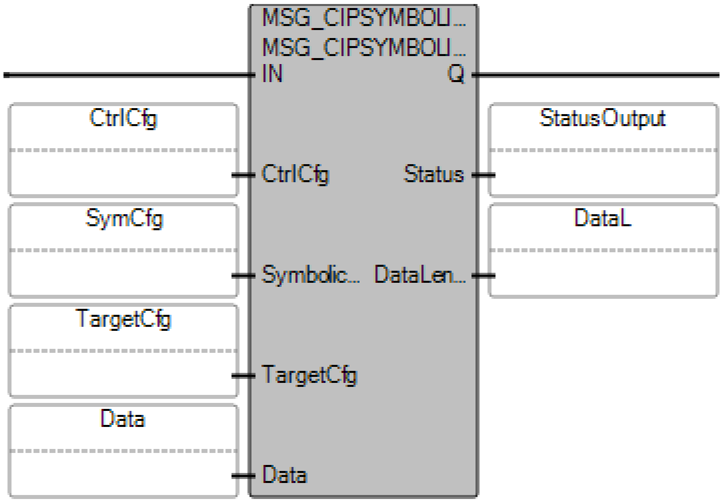
Arguments

Parameter	Parameter Type	Data Type	Description
IN	Input	BOOL	Rung input state. TRUE - Rising Edge detected, start the instruction block with the precondition that the last operation has been completed. FALSE - Rising Edge not detected, idle.
CtrlCfg	Input	CIPCONTROLCFG	The instruction block execution control configuration. Use the CIPCONTROLCFG data type on page 168 parameters to define CtrlCfg.
SymbolicCfg	Input	CIPSYMBOLICCFG on page 175	Information for the symbol for Read and Write.
TargetCfg	Input	CIPTARGETCFG	Target device configuration. Use the CIPTARGETCFG data type on page 171 parameters to define TargetCfg.
Data	Input	USINT[490]	Read command stores the data returned from the server. Write command buffers the data to be sent to the server. When an MSG is triggered or re-triggered, Data is cleared for the MSG Read command.
Q	Output	BOOL	Outputs of this instruction are updated asynchronously from the program scan. Output Q cannot be used to re-trigger the instruction since IN is edge triggered. TRUE - MSG instruction finished successfully. FALSE - MSG instruction is not finished.
Status	Output	CIPSTATUS	Function block execution status When an MSG is triggered, or re-triggered, all elements inside Status are reset. The Status output is defined in CIPSTATUS data type.
DataLength	Output	UDINT	Number of data bytes for Read service. For Write service, it's 0. When an MSG is triggered or re-triggered, DataLength is reset to 0 for MSG Read command.

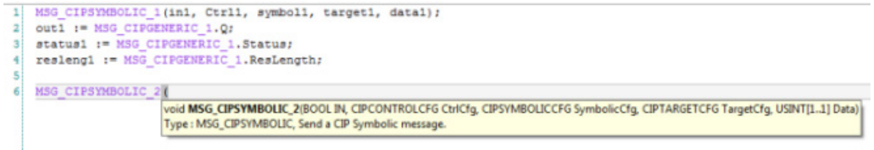
MSG_CIPSYMBOLIC Function Block Diagram example



MSG_CIPSYMBOLIC Ladder Diagram example



MSG_CIPSYMBOLIC Structured Text example



```
MSG_CIPSYMBOLIC_1(in1, Ctrl1, symbol1, target1, data1);
out1 := MSG_CIPGENERIC_1.Q;
status1 := MSG_CIPGENERIC_1.Status;
resleng1 := MSG_CIPGENERIC_1.ResLength;

MSG_CIPSYMBOLIC_2(
```

void **MSG_CIPSYMBOLIC_2**(BOOL IN, CIPCONTROLCFG CtrlCfg, CIPSYMBOLICCFG SymbolicCfg, CIPTARGETCFG TargetCfg, USINT[1..1] Data)
Type : MSG_CIPSYMBOLIC, Send a CIP Symbolic message.

CIPSYMBOLICCFG data type

Use this table to help determine the parameter values for the CIPSYMBOLICCFG data type.

Parameter	Data type	Description
Service	USINT	Service code: • 0 - Read (default) • 1 - Write
Symbol	STRING	Name of the variable to Read/Write. • Maximum of 80 characters. • Field cannot be empty. Symbol syntax defined in Symbolic Read/Write syntax on page 176.
Count	UINT	Number of variable elements to Read/Write: • Valid values: 1 - 490 • 1 is used if the value is set to 0.
Type	User-defined	User-defined data type for the target variable. Type defined in Symbolic data type support.
Offset	USINT	Reserved for future use. A byte offset of Read/Write variable used to Read/Write a large size variable that cannot be processed in one message. • 0 – 0xFF
Reserved for future use.		

Symbolic data type support

Use this table to help determine the data types MSG_CIPSYMBOLIC supports.

Data type	Data type value (hexadecimal)	Description
BOOL	193 (0xC1)	Logical Boolean with values TRUE (1) and FALSE (0)
SINT	194 (0xC2)	Signed 8-bit integer value
INT	195 (0xC3)	Signed 16-bit integer value
DINT	196 (0xC4)	Signed 32-bit integer value
LINT	197 (0xC5)	Signed 64-bit integer value
USINT	198 (0xC6)	Unsigned 8-bit integer value
UINT	199 (0xC7)	Unsigned 16-bit integer value
UDINT	200 (0xC8)	Unsigned 32-bit integer value
ULINT	201 (0xC9)	Unsigned 64-bit integer value
REAL	202 (0xCA)	32-bit floating point value
LREAL	203 (0xCB)	64-bit floating point value
STRING	218 (0xDA)	Character string

Symbolic Read/Write syntax

Syntax defines the combinations of symbols of a valid read/write instruction block.

Valid symbol names

To be valid, each symbol name must meet the following requirements.

- Begin with a letter or underscore character followed by a letter, digit, or single underscore character.
- Be 40 characters or less.
- Not contain two consecutive underscore characters.
- Use special characters [ ] . , as separators.

Symbol syntax

Use this table to help define the valid syntax for symbols. Only global variables are supported.

Symbol	Syntax	Example
Variable	PROGRAM:<program name>,<symbol name>	PROGRAM:POU1.MyTag
Array	<symbol name>[dim3, dim2, dim1] (Maximum supported dimension is 3.	MyTag1[0] MyTag2[3,6] MyTag3[1,0,4]
Structure	<symbol name>.<symbol name of struct field>	MyTag4.time.year MyTag5.local.time[1].year

Supported Data Packet Size for CIP Serial Function

For Micro830, Micro850 and Micro870 controllers, both embedded serial port and plug-in serial ports can support CIP serial communication. CIP serial communication data packet includes user data and CIP packet header.

When working as a CIP serial client, Micro830/Micro850 serial ports can support a maximum of 490 bytes of read/write user data. This maximum specification applies to CIP serial data packets with a minimum packet header size. When the size of a packet header is bigger than the minimum packet header size, the maximum size of user data that the CIP client can support is less than 490 bytes. If data packet size is greater than the maximum data size supported by the CIP client, the function block reports an error (0x21) and a sub-error (0x33).

When working as a CIP serial server, Micro830, Micro850 and Micro870 serial ports can support a minimum of 255 bytes of read/write user data. This minimum user data size specification applies to CIP serial data packets with maximum packet header size. When the size of CIP packet header is less than the maximum packet header size, the CIP client can support data packet size that is greater than the minimum specification (that is, greater than 255 bytes). However, if user data size is greater than the maximum data size supported by the CIP server function, the CIP data packet could be dropped, and the client will time out.

IMPORTANT For CIP serial server function, it is recommended not to read/write more than 255 bytes of user data in a single CIP message.

MSG_MODBUS (modbus message)

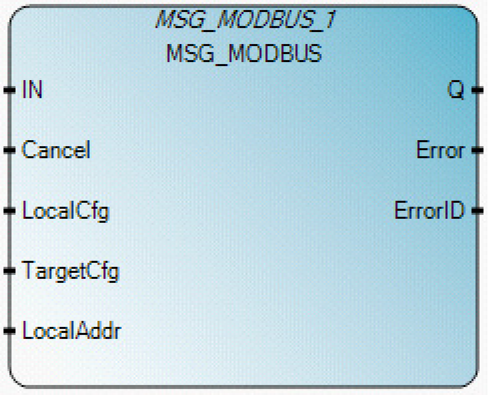
Sends a Modbus message over a serial port.

Operation details:

- A maximum of four message requests per channel can be processed in one scan. For Ladder Diagram programs, message requests are executed at the end of a ladder scan.
- If a trigger is set to continuous, error codes are also continuously cleared. To view error codes, add a rung before the MSG_MODBUS instruction.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro830, Micro850, and Micro870 controllers.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
IN	Input	BOOL	Rung input state. TRUE - Rising Edge detected, start the instruction block with the precondition that the last operation has been completed. FALSE - Rising Edge not detected, not started.
Cancel	Input	BOOL	TRUE - Cancel the execution of the instruction block. FALSE - when IN is TRUE. Cancel input is dominant.
LocalCfg	Input	MODBUSLOCPARA	Define structure input (local device). Define the input structure for the local device using the MODBUSLOCPARA data type on page 179.
TargetCfg	Input	MODBUSTARPARA	Define structure input (target device). Define the input structure for the target device using the MODBUSTARPARA data type on page 182.
LocalAddr	Input	MODBUSLOCADDR	MODBUSLOCADDR is a 125 Word array that is used by Read commands to store the data (1-125 words) returned by the Modbus slave and by Write commands to buffer the data (1-125 words) to be sent to the Modbus slave.
Q	Output	BOOL	Outputs of this instruction are updated asynchronously from the program scan. Output Q cannot be used to re-trigger the instruction since IN is edge triggered. TRUE - MSG instruction finished successfully. FALSE - MSG instruction is not finished.
Error	Output	BOOL	Indicates an error occurred. TRUE - An error is detected. FALSE - No error.
ErrorID	Output	UINT	A unique numeric that identifies the error. The errors for this instruction are defined in MSG_MODBUS error codes.

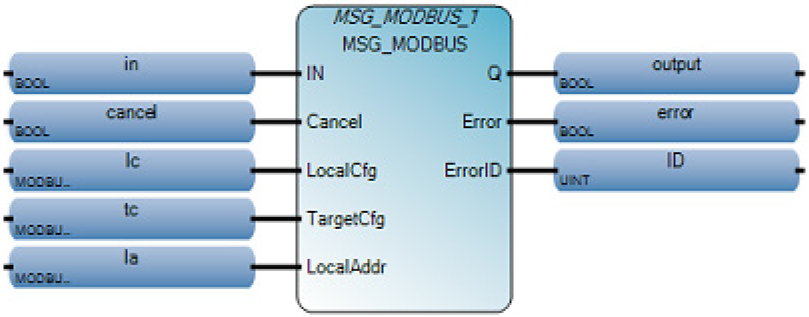
MSG_MODBUS error codes

This table describes error codes for MSG_MODBUS.

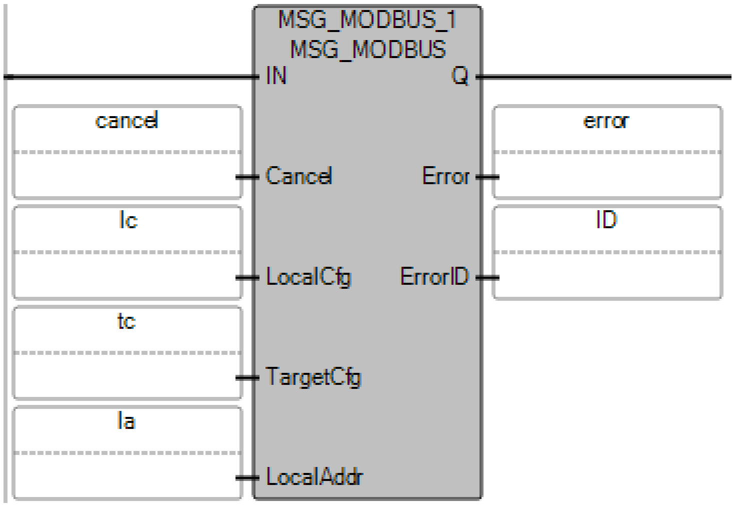
Error code	Error description
3	The value of the TriggerType has been changed from 2 - 255.
20	The local communication driver is incompatible with the MSG instruction.
21	A local channel configuration parameter error exists.
22	The Target or Local Bridge address is higher than the maximum node address.
33	A bad MSG file parameter exists.
54	A lost modem.
55	The message timed out in the local processor. A link layer timeout.
217	The user cancelled the message.
129	An illegal function.
130	An illegal data address.
131	An illegal data value.
132	A slave device failure.
133	Acknowledge.
134	The slave device is busy.
135	Negative acknowledge.

Error code	Error description
136	A memory parity error.
137	A non-standard reply.
255	The channel has been shut down.

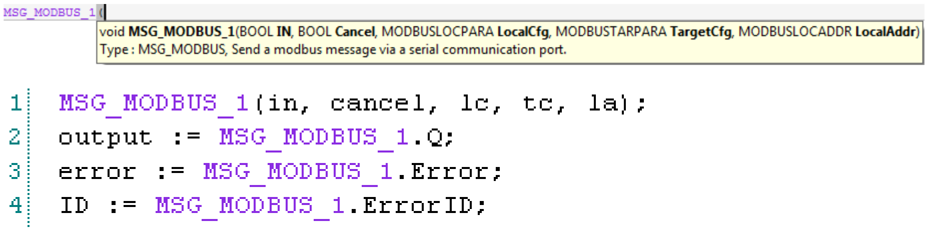
MSG_MODBUS Function Block Diagram example



MSG_MODBUS Ladder Diagram example



MSG_MODBUS Structured Text example



MSG_MODBUS_1(

void **MSG_MODBUS_1**(BOOL **IN**, BOOL **Cancel**, MODBUSLOCPARA **LocalCfg**, MODBUSTARPARA **TargetCfg**, MODBUSLOCADDR **LocalAddr**)
Type : MSG_MODBUS, Send a modbus message via a serial communication port.

```
MSG_MODBUS_1(in, cancel, lc, tc, la);
output := MSG_MODBUS_1.Q;
error := MSG_MODBUS_1.Error;
ID := MSG_MODBUS_1.ErrorID;
```

MODBUSLOCPARA data type

Use this table to help determine the parameter values for the MODBUSLOCPARA data type.

Parameter	Data type	Description
Channel	UINT	Micro800 PLC serial port number: • 2 for the embedded serial port, or • 5-9 for serial port plug-ins installed in slots 1 through • 5 for slot 1 • 6 for slot 2 • 7 for slot 3 • 8 for slot 4 • 9 for slot 5
TriggerType	USINT	Represents one of the following: • 0: Msg Triggered Once (when IN goes from False to True) • 1: Msg triggered continuously when IN is True • Other value: Reserved
Cmd	USINT	Represents one of the following: • 01: Read Coil Status (0xxxx) • 02: Read Input Status (1xxxx) • 03: Read Holding Registers (4xxxx) • 04: Read Input Registers (3xxxx) • 05: Write Single Coil (0xxxx) • 06: Write Single Register (4xxxx) • 15: Write Multiple Coils (0xxxx) • 16: Write Multiple Registers (4xxxx) • Others: Custom command support. MODBUSLOCPARA custom command support: Custom Commands in the range of 0-255 that are not already assigned to a Modbus command are also supported. If a custom command is used then the LocalCfg:ElementCnt contains the number of bytes received. The response is received into the Local Address Data and overwrites the request data. • Example for CMD=0x2B • Local Address Data 1:0x0E, READ_DEVICE_ID_MEI • Local Address Data 2:0x01, READ_DEV_ID_BASIC • Local Address Data 3:0x00, Read Vendor Object
ElementCnt	UINT	Limits • For Read Coil/Discrete inputs: 2000 bits • For Read Register: 125 words • For Write Coil: 1968 bits • For Write Register: 123 words

MSG_MODBUS message triggering

A Modbus message can be triggered periodically by setting a non-zero value to the TriggerType parameter.

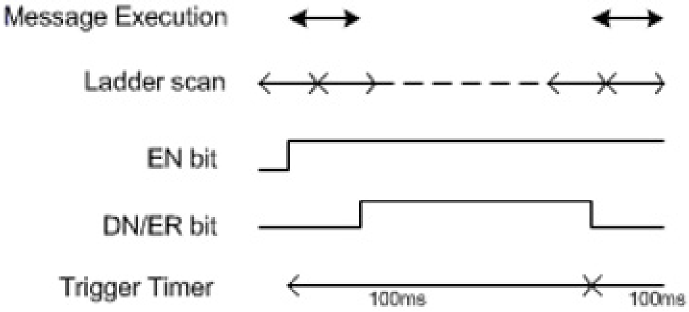
This table describes the TriggerType parameter behavior when used with the MSG_MODBUS on page 177 function block.

Action	Results
Message is enabled	Trigger timer starts

Action	Results
Trigger timer expires before the message completes	Message is immediately triggered in the next ladder scan cycle.
Message completes before the trigger time expires	Message is triggered when the trigger time expires.

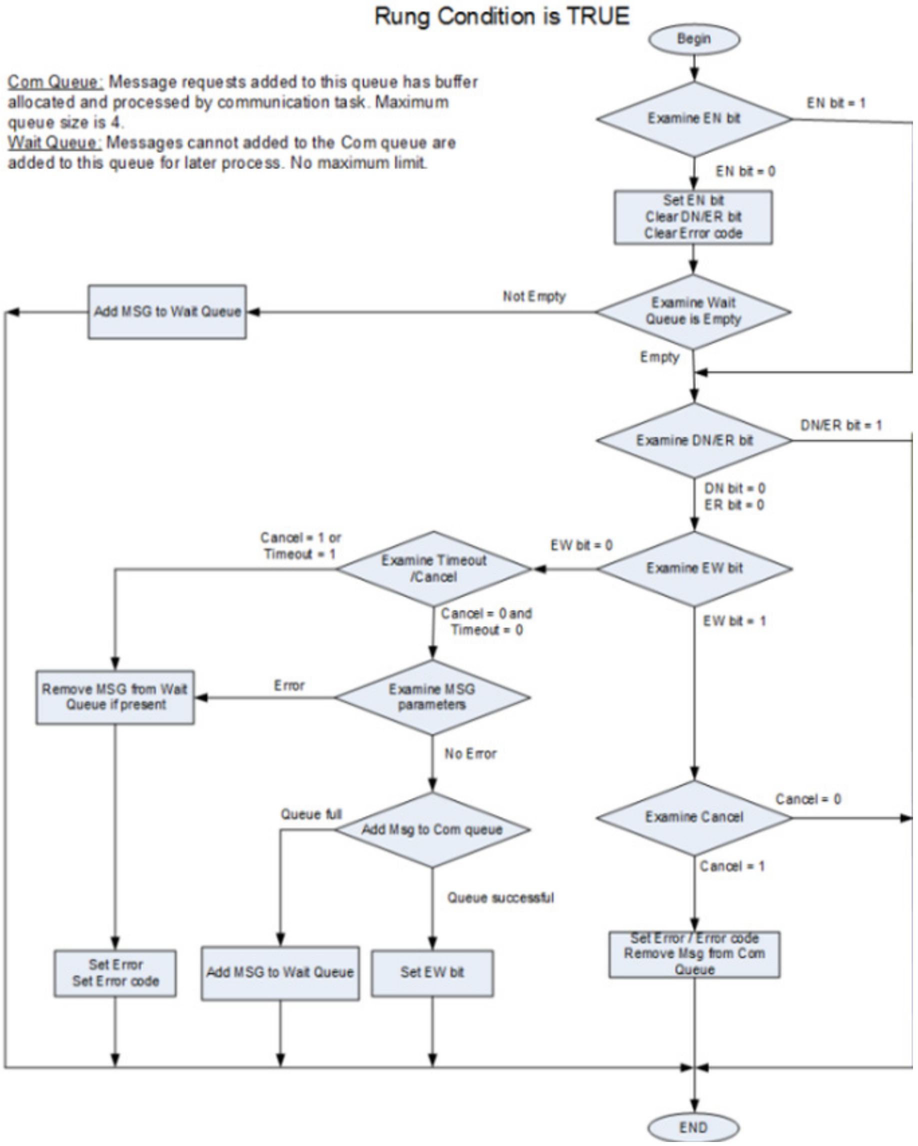
Example: message triggering

In the following example, the TriggerType value is set to 100.



Message execution process (Rung = TRUE)

The following process diagram describes the message instruction events that occur when the Rung condition is True.



Rung Condition is TRUE

Com Queue: Message requests added to this queue has buffer allocated and processed by communication task. Maximum queue size is 4.
Wait Queue: Messages cannot added to the Com queue are added to this queue for later process. No maximum limit.

Com queue: Message requests added to the Com queue have a buffer allocated and processed by the communication task. The maximum queue size limit is 4.

Wait queue: Messages that cannot be added to the Com queue are added to the Wait queue to be processed at a later time. The Wait queue does not have a maximum size limit.

MODBUSTARPARA data type

The following table describes the MODBUSTARPARA data type.

Parameter	Data type	Description
Addr	UDINT	Target data address (1 - 65536). Decreases by one when sending.
Node	USINT	The default slave node address is 1. The range is 1- 247. Zero is the Modbus broadcast address and is only valid for Modbus write commands (for example, 5, 6, 15 and 16).

MSG_MODBUS2 (MODBUS/TCP message)

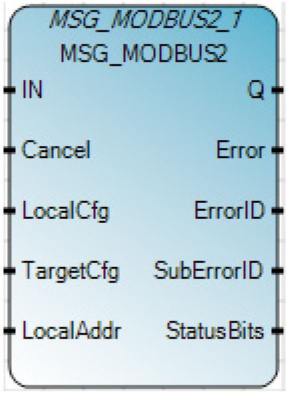
Sends a MODBUS/TCP message over an Ethernet Channel.

Operation details:

- A maximum of four message requests per channel can be processed in one scan. For Ladder Diagram programs, message requests are executed at the end of a ladder scan.
- When MSG_MODBUS2 is enabled, the receive buffers for Read operations are cleared on the rising edge of Enable.
- Canceling the execution of the MSG_MODBUS2 instruction does not guarantee the message request going out is Cancelled, but does guarantee the response is not processed.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro850, and Micro870 controllers. For the Micro800 Simulator, this instruction can be added to a program but is only a placeholder to prevent the instruction from being deleted during controller change.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
IN	Input	BOOL	Rung input state. TRUE - Rising Edge detected, start the instruction block with the precondition that the last operation has been completed. FALSE - Rising Edge not detected, idle.
Cancel	Input	BOOL	TRUE - Cancel the execution of the instruction block. Canceling the execution of the MSG_MODBUS2 instruction does not guarantee the message request going out is Cancelled, but does guarantee the response is not processed. FALSE - when IN is TRUE. Cancel input is dominant.
LocalCfg	Input	MODBUS2LOCPARA	Defines structure input (local device). Define the input structure for the local device using the MODBUS2LOCPARA data type on page 186.
TargetCfg	Input	MODBUS2TARPARA	Defines structure input (target device). Define the input structure for the target device using the MODBUS2TARPARA data type on page 187.
LocalAddr	Input	MODBUSLOCADDR	MODBUSLOCADDR data type is a 125 Word array. LocalAddr usage: • For Read commands, store the data (1-125 words) returned by the Modbus slave. • For Write commands, buffer the data (1-125 words) to be sent to the Modbus slave.

Q	Output	BOOL	Outputs of this instruction are updated asynchronously from the program scan. Output Q cannot be used to re-trigger the instruction since IN is edge triggered. TRUE - MSG instruction finished successfully. FALSE - MSG instruction is not finished.
Error	Output	BOOL	Indicates an error detected. TRUE - An error occurred. FALSE - No error.
ErrorID	Output	UINT	A unique numeric that identifies the error. The errors for this instruction are defined in Modbus2 error codes.
SuberrorID	Output	UINT	Used to verify status bits: ● Bit 0: EN – Enable ● Bit 1: EW – Enable Wait ● Bit 2: ST – Start ● Bit 3: ER – Error ● Bit 4: DN – Done Other bits are reserved.
StatusBits	Output	UINT	SubError code value when Error is TRUE. When a MSG is triggered, or re-triggered, a previously set SubErrorID is cleared.

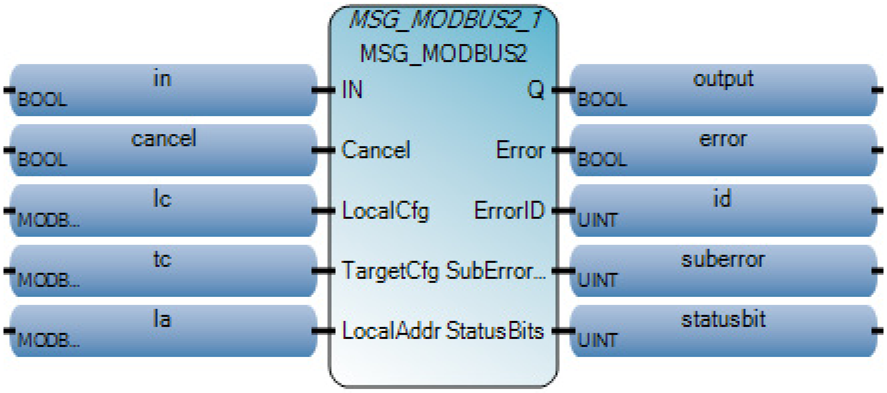
MSG_MODBUS2 error and sub-error codes

When the ER bit is set, the ErrorID and SubErrorID fields display the following error codes.

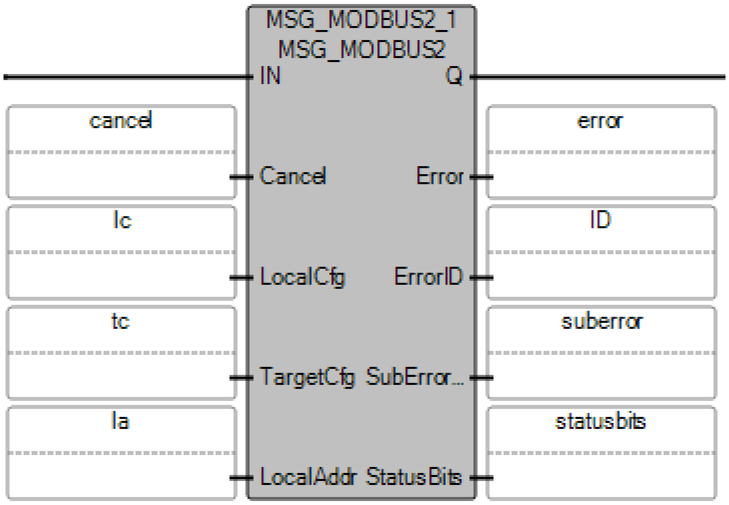
Error ID	SubErrorID	Description
33	Parameter configuration related errors	
	32	Bad Channel number.
	37	Bad Element count.
	38	Bad Data Address.
55	Timeout related errors	
	112	Message timed out while waiting in the message wait queue.
	113	Message timed out while waiting for the a connection to the link layer to be established.
	114	Message timed out while waiting to transmit to the link layer.
	115	Message timed out while waiting for a response from the link layer.
69	Server Response format related error codes.	
208	No IP address configured for the network.	
209	Maximum number of connections used – no connections available.	
210	Invalid internet address or node address.	
217	Message execution was canceled by user. (Cancel parameter was set to TRUE).	
222	Network connection fail to establish before timeout.	
255	Channel is shutdown or reconfiguration is in progress. Error code occurs immediately after power on until a connection is established, and is normal behavior. It may also occur if an Ethernet cable is disconnected or an IP address cannot be detected.	
	Slave response error codes	
129	Illegal Function Code	
130	Illegal Data Address	
131	Illegal Data Value	

132	Server Failure
133	Acknowledge
134	Negative Acknowledge
136	Memory parity Error
137	Non-standard reply error code. Actual error code can be found in the SubErrorID.

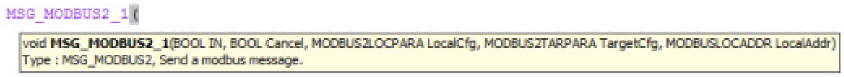
MSG_MODBUS2 Function Block Diagram example



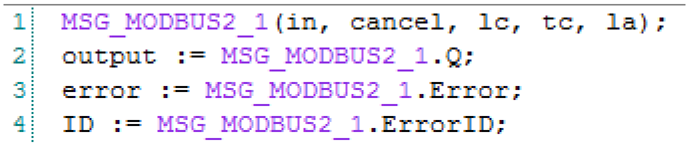
MS_MODBUS2 Ladder Diagram example



MSG_MODBUS2 Structured Text example



```
MSG_MODBUS2_1(
void MSG_MODBUS2_1(BOOL IN, BOOL Cancel, MODBUS2LOCPARA LocalCfg, MODBUS2TARPARA TargetCfg, MODBUSLOCADDR LocalAddr)
Type : MSG_MODBUS2, Send a modbus message.
```



```
MSG_MODBUS2_1(in, cancel, lc, tc, la);
output := MSG_MODBUS2_1.Q;
error := MSG_MODBUS2_1.Error;
ID := MSG_MODBUS2_1.ErrorID;
```

MODBUS2LOCPARA data type

Use this table to help determine the parameter values for the MODBUS2LOCPARA data type.

Parameter	Data Type	Description
Channel	UINT	Local Ethernet port number: • 4 for Micro850 & Micro820 embedded Ethernet port.
TriggerType	UDINT	Message trigger type: • 0: Msg Triggered Once (when IN goes from False to True) • 1 to 65535 - Cyclic trigger value in milliseconds. Message triggered periodically when IN is true and the previous message execution completes. • Set the value to 1 to trigger messages as quickly as possible. See below MSG_MODBUS2 message triggering.
Cmd	USINT	Modbus command: • 01: Read Coil Status (0xxxx) • 02: Read Input Status (1xxxx) • 03: Read Holding Registers (4xxxx) • 04: Read Input Registers (3xxxx) • 05: Write Single Coil (0xxxx) • 06: Write Single Register (4xxxx) • 15: Write Multiple Coils (0xxxx) • 16: Write Multiple Registers (4xxxx) • Others: Custom command support MODBUS2LOCPARA custom command support: Custom Commands in the range of 0-255 not already assigned to a Modbus command are also supported. If a custom command is used then LocalCfg:ElementCnt contains the number of bytes received. The response is received in Local Address Data and overwrites the request data. Example for CMD=0x2B: • Local Address Data 1:0x0E, READ_DEVICE_ID_MEI • Local Address Data 2:0x01, READ_DEV_ID_BASIC • Local Address Data 3:0x00, Read Vendor Object
ElementCnt	UINT	Limits • For Read Coil/Discrete inputs: 2000 bits • For Read Register: 125 words • For Write Coil: 1968 bits • For Write Register: 123 words

MSG_MODBUS2 message triggering

A Modbus message can be triggered periodically by setting a non-zero value to the TriggerType parameter.

This table describes what happens when the TriggerType parameter is used with the MSG_MODBUS2 on page 182 function block.

Action	Results
Message is enabled.	Trigger timer starts.
Trigger timer expires before the message completes.	Message is immediately triggered in the next ladder scan cycle.
Message completes before the trigger time expires.	Message is triggered when the trigger time expires.

MODBUS2TARPARA data type

Use this table to help determine the parameter values for the MODBUS2TARPARA data type.

Parameter	Data type	Description
Addr	UDINT	Target device's Modbus data address: • 1 - 65536. • Decreases by one when sending. • Firmware uses low-word of address if the address value is greater than 65536.
NodeAddress[4]	USINT	Target device's IP address. The IP address should be a valid unicast address and cannot be 0, multicast, broadcast, local address or loop back address (127.x.x.x). For example, to specify 192.168.2.100: • NodeAddress[0]=192 • NodeAddress[1]=168 • NodeAddress[2]=2 • NodeAddress[3]=100
Port	UINT	Target TCP port number. Standard Modbus/TCP port is 502. 1 - 65535 Set to 0 to use the default value 502
UnitId	USINT	Unit Identifier. Used to communicate with slave devices through a Modbus bridge. Refer Modbus specification for more details. Note that Micro800 shall not attempt to validate this value. 0 - 255 Set to 255 if Target device is not a bridge.
MsgTimeOut	UDINT	Message timeout (in milliseconds). Amount of time to wait for a reply for an initiated command. • 250-10,000 • Set to 0 to use the default value 3,000. • A value less than 250 (minimum) is set to 250. • A value greater than 10,000 (maximum) is set to 10,000. See Modbus/TCP message timeout timers.
ConnTimeOut	UDINT	TCP Connection establishment timeout (in milliseconds). Amount of time to wait for establishing successful TCP connection to the Target device. • 250-10,000 • Set to 0 to use the default value 5,000. • A value less than 250 (minimum) is set to 250. • A value greater than 10,000 (maximum) is set to 10,000. See Modbus/TCP message timeout timers.
ConnClose	BOOL	TCP connection closing behavior. • True - Close the TCP connection upon message completion. • False - Do not close the TCP connection upon message completion [Default]. See Modbus/TCP message connections.

Modbus/TCP message timeout timers

This table describes the behavior for MsgTimeOut and ConnTimeOut based on message requests and status.

Action	Results

Action	Results
Message is enabled.	Activates the MsgTimeOut timer.
TCP connection is requested.	Activates the ConnTimeOut timer.
ConnTimeOut timer is active.	Disables the MsgTimeOut timer.
Connection request is complete.	Reactivates the MsgTimeOut timer.

Modbus/TCP message connections

Modbus/TCP client supports a maximum of 16 connections. This table describes Modbus/TCP connection behavior.

Scenario	Results
Message request is enabled, and a connection to the target does not exist.	If a connection to the target does not exist, a new connection is established. If a connection to the target already exists, the existing connection is used.
Message execution is completed, and ConnClose is set to True.	If there is only one connection to the target, the connection is closed. If there is more than one connection to the target, the connection is closed when the last message execution is completed.
Message execution is completed, and ConnClose is set to False.	The connection is not closed.
Connection is not associated with an active message, and remains idle for the amount of time specified in ConnTimeOut parameter.	The connection is closed.
Controller transitions from an executing mode (Run, Remote Run, Remote Test Single Scan and Remote Single Rung) to a non-executing mode.	All active connections are forcibly closed.

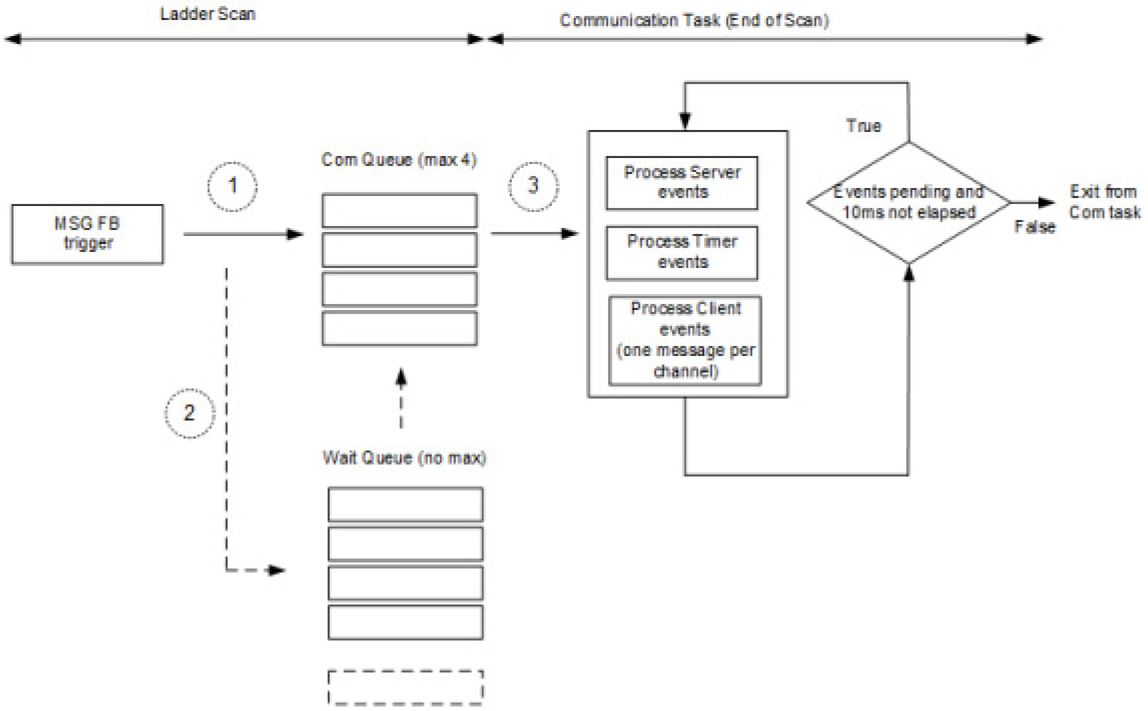
Message execution processes and timing diagrams

The following topics describe how and when MSG_CIPGENERIC on page 166, MSG_CIPSYMBOLIC on page 173 and MSG_MODBUS2 on page 182 message instructions execute based on their bit and rung conditions.

- Message execution process (general)
- Message execution process (Rung = TRUE)
- Message execution timing diagram (Rung = True)
- Message execution process (Rung = FALSE)
- Message execution timing diagram (Rung = FALSE)
- Message execution process (Error)
- Message execution timing diagram (Error)

Message execution process (general)

The following diagram shows how and when messages execute based on the status of the Com queue.

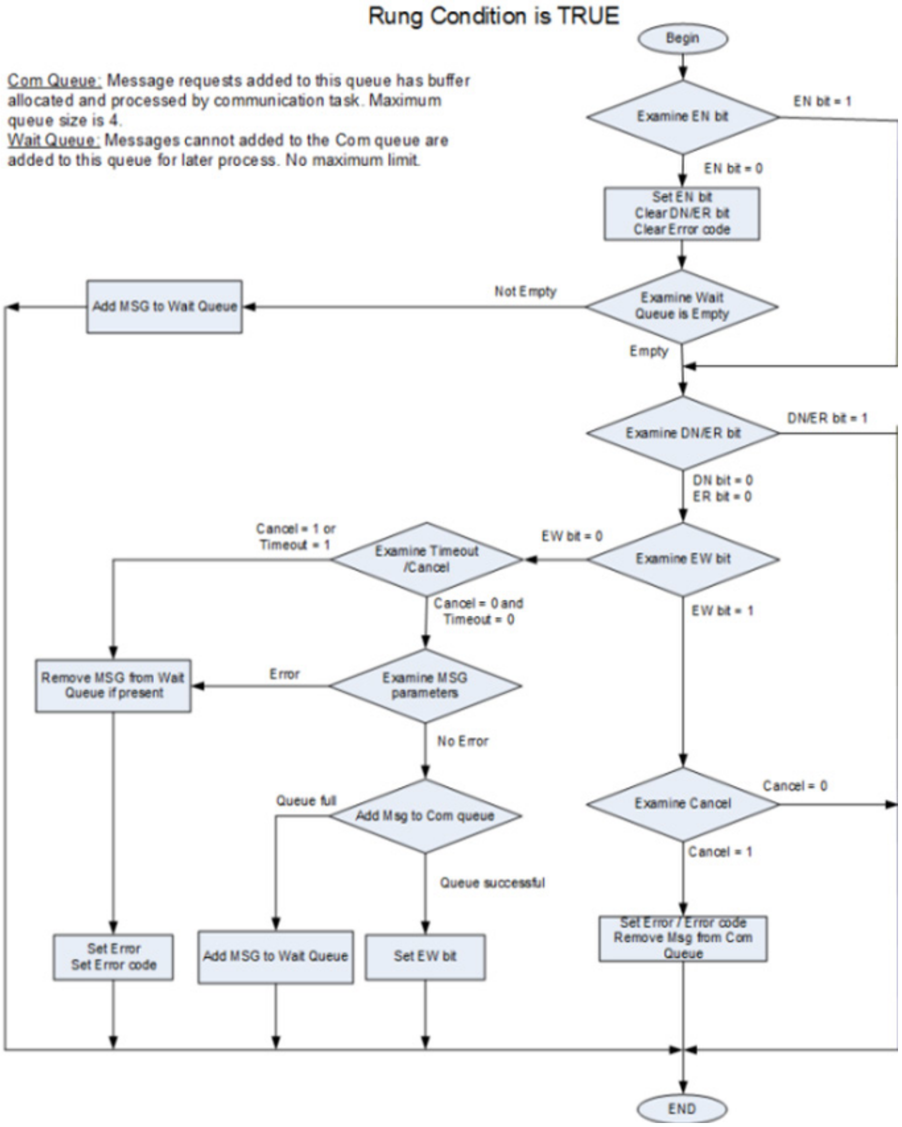


The following table describes the sequence of events identified in the preceding diagram.

No.	Description of events
1	The message is enabled. If the Com queue is empty, the buffer is allocated for the message and the message is added to the Com queue for transmission. The Com queue size is 4 and each channel has a separate queue.
2	If the Com queue is full, the message is added to the Wait Queue. When the Com queue is empty, the message in the Wait queue is added to the Com queue. There is no size limit for the Wait Queue and each channel has a separate queue.
3	The communication task executes the messages in the Com queue on every End-of-Scan for transmission. Each channel's queue is processed one by one in a round robin fashion. One message from each channel is executed, and the process continues until all messages are executed or the communication schedule (10ms) expires. The channel next to the last processed channel is scheduled to appear first in the next End-of-Scan.

Message execution process (Rung = TRUE)

The following process diagram describes the message instruction events that occur when the Rung condition is True.



Com queue: Message requests added to the Com queue have a buffer allocated and processed by the communication task. The maximum queue size limit is 4.

Wait queue: Messages that cannot be added to the Com queue are added to the Wait queue to be processed at a later time. The Wait queue does not have a maximum size limit.

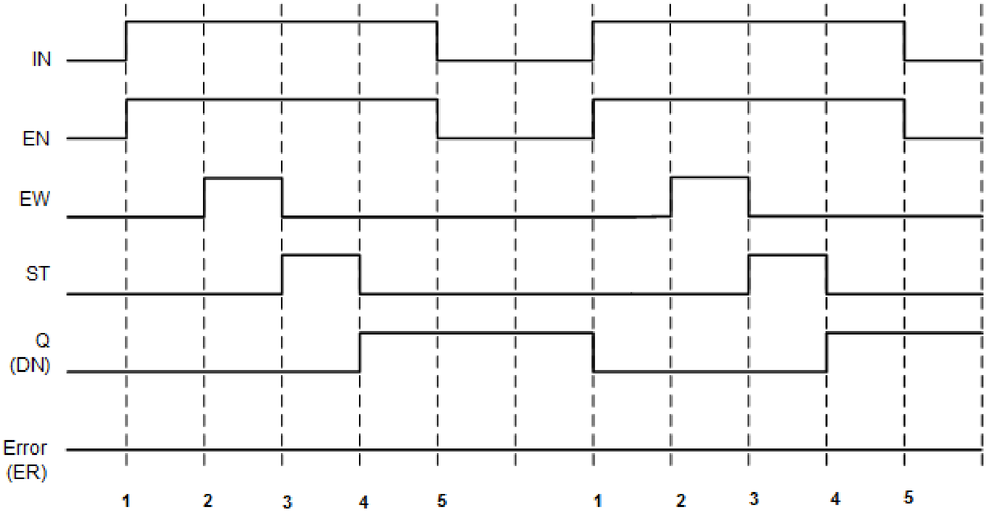
Message execution timing diagram (Rung = TRUE)

The following table describes the message conditions and bit status for each execution step identified in the timing diagram while the rung condition remains true.

Step	Message description	Bit status
1	Rung condition goes TRUE. Message execution is enabled.	EN bit is set. All other bits are cleared.

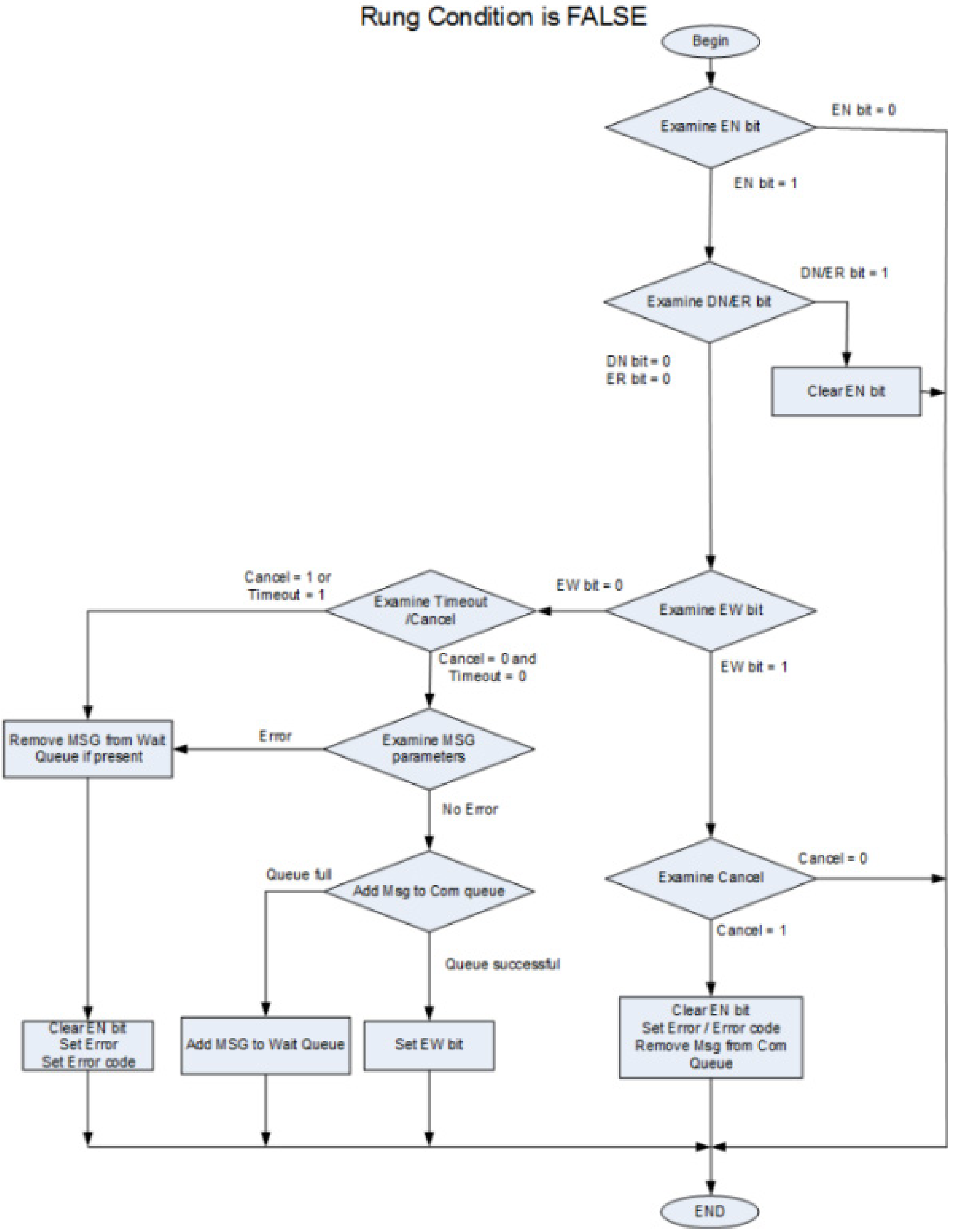
Step	Message description	Bit status
2	Message control buffer is acquired. At this time, input data (that is, the "data" parameter for write messages) is copied for transmission. Subsequent changes to the input data will not be reflected in the transmitted message.	EW bit is set.
3	Message transmission starts.	EW bit is cleared. ST bit is set.
4	Message response is received.	ST bit is cleared. DN bit is set.
5	Rung condition goes FALSE.	EN bit is cleared.

Timing diagram for (Rung = TRUE)



Message execution process (Rung = FALSE)

The following process diagram describes the message instruction events that occur when the Rung condition is False.



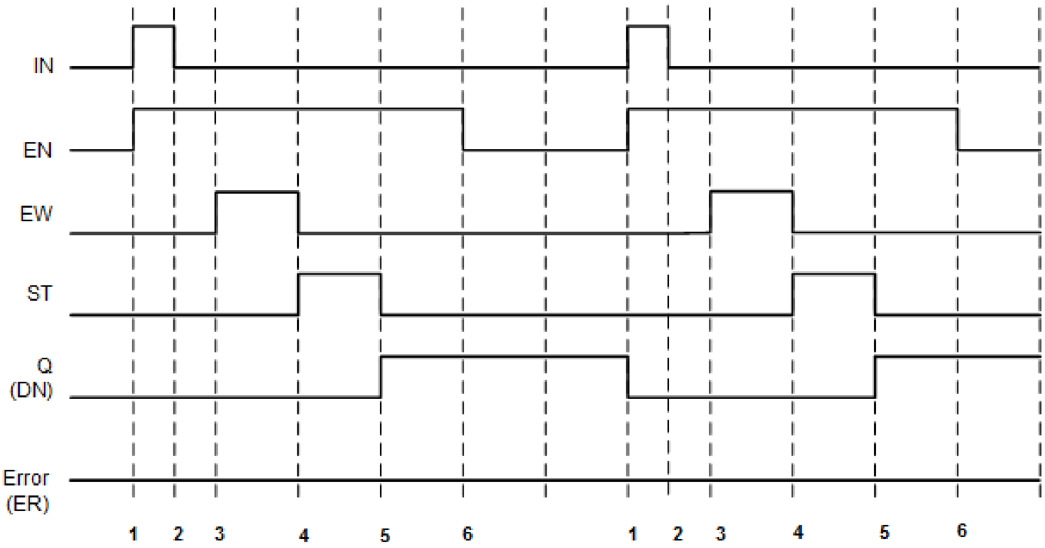
Message execution timing diagram (Rung = FALSE)

The following table describes the message conditions and bit status for each execution step identified in the timing diagram when the rung goes to FALSE during execution.

Step	Message description	Bit status

Step	Message description	Bit status
1	Rung condition goes TRUE. Message execution is enabled.	EN bit is set. All other bits are cleared.
2	Rung condition goes FALSE. Message execution continues.	
3	Message buffer is acquired.At this time, input data (that is, the "data" parameter for write messages) is copied for transmission. Subsequent changes to the input data will not be reflected in the transmitted message.	EW bit is set.
4	Message transmission starts.	EW bit is cleared. ST bit is set.
5	Message response is received.	ST bit is cleared. DN bit is set.
6	Message is scanned again after step 5.	EN bit is cleared.

Timing diagram for (Rung = FALSE)



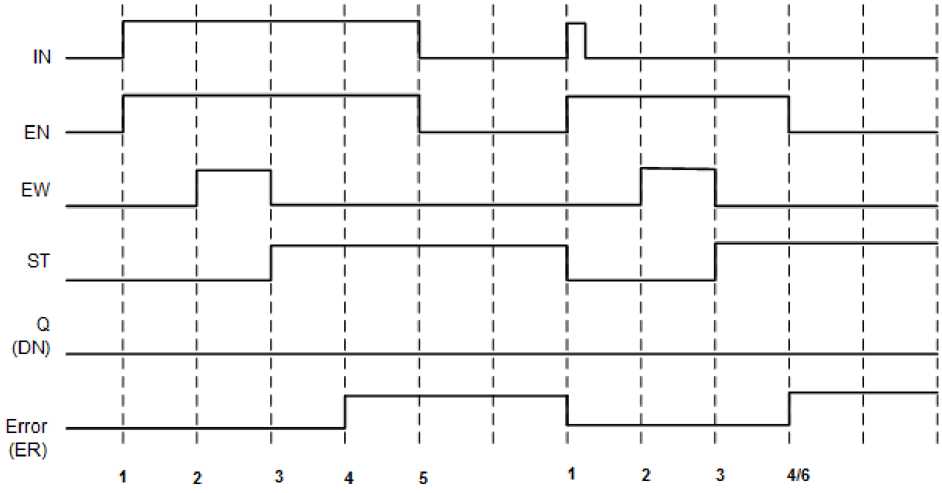
Message execution process (Error)

The following table describes the message conditions and bit status for each execution step identified in the timing diagram when an error occurs during execution.

Step	Message description	Bit status
1	Rung condition goes TRUE. Message execution is enabled.	EN bit is set. All other bits are cleared.
2	Message buffer is acquired.	EW bit is set.
3	Message transmission starts.	EW bit is cleared. ST bit is set.
4	Message transmission times out.	EW and ST bits do not change.
4-6	As rung condition goes FALSE.	EN bit is cleared. ER bit is set.

Message execution timing diagram (Error)

The following timing diagram shows a typical pattern when an error occurs during execution.



Use the communication (message) function blocks

This section provides specific details and examples for using communication instructions on page 163 in logic programs. See the following topics for details of and examples for using the MSG_CIPGENERIC on page 196 and MSG_CIPSYMBOLIC function blocks to create programs on page 204.

Configure object data values for explicit messaging (MSG_CIPGENERIC)

To use the MSG_CIPGENERIC on page 166 function block for explicit messaging, configure the AppCfg parameter with the correct values.

For additional information about message communication

There are several sources of information covering the implementation and use of message communication, including Connected Components Workbench Help, user manuals and the Rockwell Automation Literature Library.

The following table lists additional sources of information relevant to message communication.

Information source	Description	How to find the information
User manual for your specific communication device	Contains important information about messaging and specific information for configuring message function blocks.	Connected Components Workbench Help menu
EtherNet/IP Adapter 22-COMM-E FRN 1.xxx, Appendix C	Provides information about the EtherNet/IP objects that can be accessed using Explicit Messages.	Connected Components Workbench Help menu
EtherNet/IP specification	Defines the objects to be included in every CIP device: Identity object, Message Router object and the Network object.	ODVA web site (http://www.odva.org)

Micro800 Programmable Controllers: Getting Started with CIP Client Messaging	Provides quickstart instructions for using CIP GENERIC and CIP Symbolic Messaging in Micro830 and Micro850 programmable logic controllers (PLC).	Rockwell Automation Literature Library

To access user manuals and quickstart guides:

1. To access quickstart guides, on the **Help** menu, click **View Help**.
2. Double-click on **Connected Components Workbench**.
3. Double-click on **Getting Started with Connected Components Workbench**.
4. To access drive manuals, on the **Help** menu, click **User Manuals** to display the Manuals dialog box.
5. Click the plus (+) sign next to Drives to expand the category, and then expand the class until the manual is located.
6. Double-click the manual name to open the .pdf file.
7. To access the EtherNet/IP manual, on the **Help** menu, click **User Manuals** to display the Manuals dialog box.
8. Click the plus (+) sign next to **Drives** to expand the category, and then expand the PowerFlex 4-class Peripherals class.
9. Double-click the 22-COMM-E EtherNet/IP Adapter User Manual to open the .pdf file.

To access manuals from the Rockwell Automation Literature Library:

1. Go to http://literature.rockwellautomation.com.

2. To access non-English language versions of user manuals, select the language from the Publication Language drop-down box (right corner).
3. Enter the full or partial device catalog number in the **Search** box. For example, enter 2080-LC30 to view Micro830 user manuals.
4. In the **Search** box, type the full or partial device catalog number. For example, enter 2080-LC30 to view Micro830 user manuals.

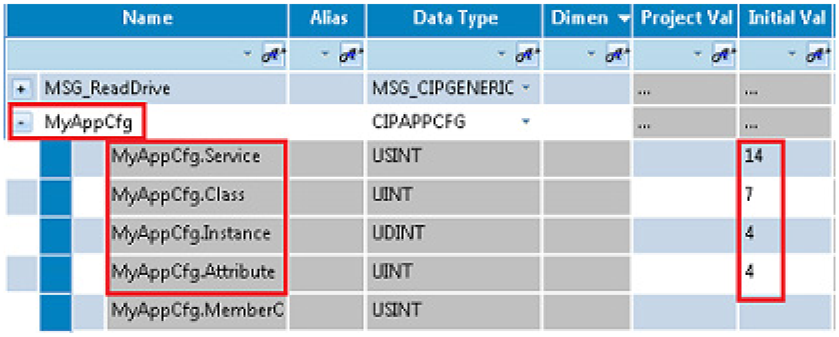
CIP Register object data

MSG_CIPGENERIC function blocks use the CIP Register object data in the AppCfg parameter. The object data includes the following:

- Class Code
- Instance
- Instance Attribute
- Service

Values for the MSG_CIPGENERIC AppCfg parameter

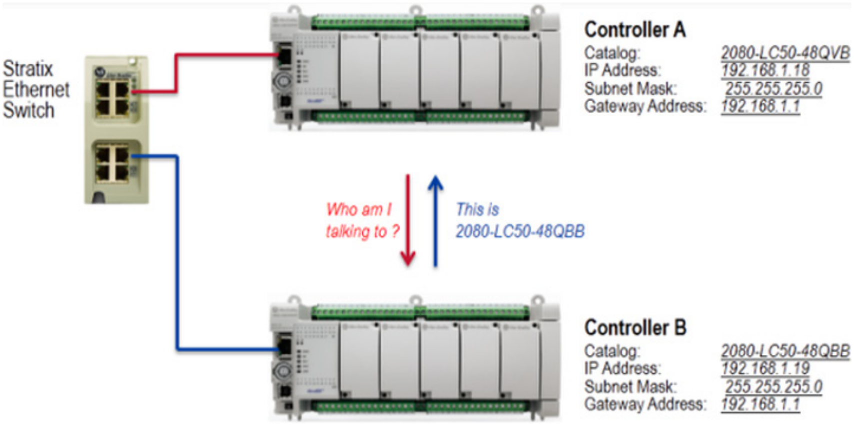
Use the values from the CIP register object in your input variables to configure the MSG_CIPGENERIC function block parameters. The following image shows the CIP register object data values used in the MSG_CIPGENERIC function block parameters.



Name	Alias	Data Type	Dimen	Project Val	Initial Val
+ MSG_ReadDrive		MSG_CIPGENERIC		...	...
- MyAppCfg		CIPAPPCFG		...	...
MyAppCfg.Service		USINT			14
MyAppCfg.Class		UINT			7
MyAppCfg.Instance		UDINT			4
MyAppCfg.Attribute		UINT			4
MyAppCfg.MemberC		USINT			

Example: How to create an MSG_CIPGENERIC messaging program to read data from a controller

This example shows how to create a message program that retrieves Controller B catalog information from Controller A using a MSG_CIPGENERIC function block and a COP function block.



Perform the following tasks to create a MSG_CIPGENERIC messaging program that reads information from a different controller.

No	Task
1	Identify initial values for the input variables (MSG_CIPGENERIC) on page 196
2	Add a MSG_CIPGENERIC function block and variables on page 197
3	Configure initial values for variables on page 198
4	Add a contact and a coil on page 201
5	Add a COP function block, variables and contact (MSG_CIPGENERIC) on page 202
6	Verify correct IP configuration on Controller B on page 203

Identify initial values for the input variables, MSG_CIPGENERIC

Follow these general steps to add input variables and initial values, and obtain the Identity Object values to configure the AppCfg parameter initial values.

To add input variables and initial values:

1. From the **Help** menu, click **User Manuals**.
2. Expand the Drives selection and locate the user manual for the type of communication adapter you are using (EtherNet/IP Adapter User Manual).
3. Double-click the manual to open it.
4. Review the Appendix headings to locate the section that provides information about the EtherNet/IP objects that can be accessed using Explicit Messages (Appendix C).
5. Go to the Appendix section and identify the object type related to your explicit message (Identity object).
6. Identify the initial values for the AppCfg parameters based on the type information being retrieved.

Ethernet/IP object data and AppCfg parameters example

The following table identifies the specific Ethernet/IP object data used to read catalog information from a controller.

Input variable example	AppCfg parameter	Ethernet/IP object data option	Description	Initial value
MyAppCfg.Service	Service	Service code	Implement for class = Yes Implement for Instance = Yes Get attribute single	14 (0x0E in hexadecimal)
MyAppCfg.Class	Class	Class code	EtherNet/IP object class = Identity object	01
MyAppCfg.Instance	Instance	Instances	22-COMM-E	01
MyAppCfg.Attribute	Attribute	Instance attribute	Get product name and rating as SHORT STRING	07

Add a MSG_CIPGENERIC function block and variables

To add a MSG_CIPGENERIC function block to a ladder diagram program and then add input variables to the function block, perform the following steps.

To add a MSG_CIPGENERIC function block:

1. Add a controller:
   - Expand the **Controllers** folder and the Micro850 folder to view all Micro850 controllers.
   - Double-click a controller (2080-LC50-48QVB) to add it to **Project Organizer**.
2. Add a ladder diagram program:
   - In **Project Organizer**, right-click **Programs**, click **Add**, and then click **New LD: Ladder Diagram**.
   - Right-click the ladder diagram icon in **Project Organizer**, click **Rename** and type CIPExplicitMessage.

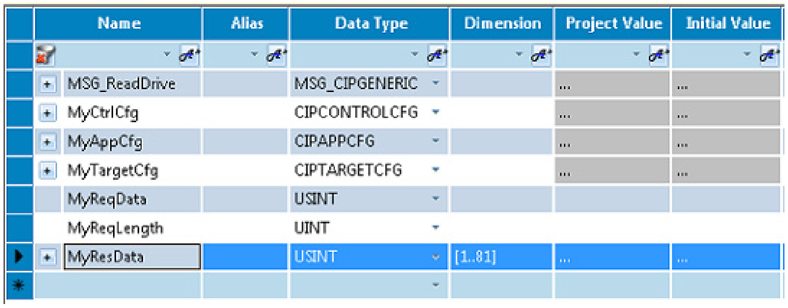
- Double-click the ladder diagram program in **Project Organizer** to display the LD POU in the language editor.

3. Add the MSG_CIPGENERIC function block:
   - In the **Toolbox**, select **Instruction Block** and drag and drop it onto the ladder rung to display the **Instruction Block Selector**.
   - In Search, type **MSG** to display the message function blocks.
   - Type **MSG_ReadDrive** in the **Instance** field.
   - Double-click **MSG_CIPGENERIC** to add an instance of the function block to the ladder diagram.
4. Add MSG_CIPGENERIC local input variables:
   - In **Project Organizer**, double-click **Local Variables** to display the **Local Variables** page.
   - In the **Variables** page, add the variables and data types listed in the table.

Parameter	Variable Name	Data Type
CtrlCfg	MyCtrlCfg	CIPCONTROLCFG
AppCfg	MyAppCfg	CIPAPPCFG
TargetCfg	MyTargetCfg	CIPTARGETCFG
ReqData	MyReqData	USINT
ReqLength	MyReqLength	UINT
ResData	MyResData	USINT (array)

5. For the MyResData variable, double click **Dimension** and change the array size to [1..81].

   The **Variables** page should look similar to the following image.



Name	Alias	Data Type	Dimension	Project Value	Initial Value
MSG_ReadDrive		MSG_CIPGENERIC		...	...
MyCtrlCfg		CIPCONTROLCFG		...	...
MyAppCfg		CIPAPPCFG		...	...
MyTargetCfg		CIPTARGETCFG		...	...
MyReqData		USINT			
MyReqLength		UINT			
MyResData		USINT	[1..81]	...	...

Configure initial values for variables

Follow these steps to add initial values to the input variables you previously created and then assign the variables to the correct MSG_CIPGENERIC function block input parameter.

To assign variables to MSG_CIPGENERIC:

1. To configure initial values for the MyCtrlCfg input variable:
   - From the **Local Variables** page, expand MyCtrlCfg to view its parameters.
   - Enter the following values in the **Initial Value** column for each parameter.

Parameter	Initial value	Comments
MyCtrlCfg.Cancel	Leave blank	Not needed.
MyCtrlCfg.TriggerType	0	We only need to retrieve the catalog number once.
MyCtrlcfg.StrMode	Leave blank	Not needed.

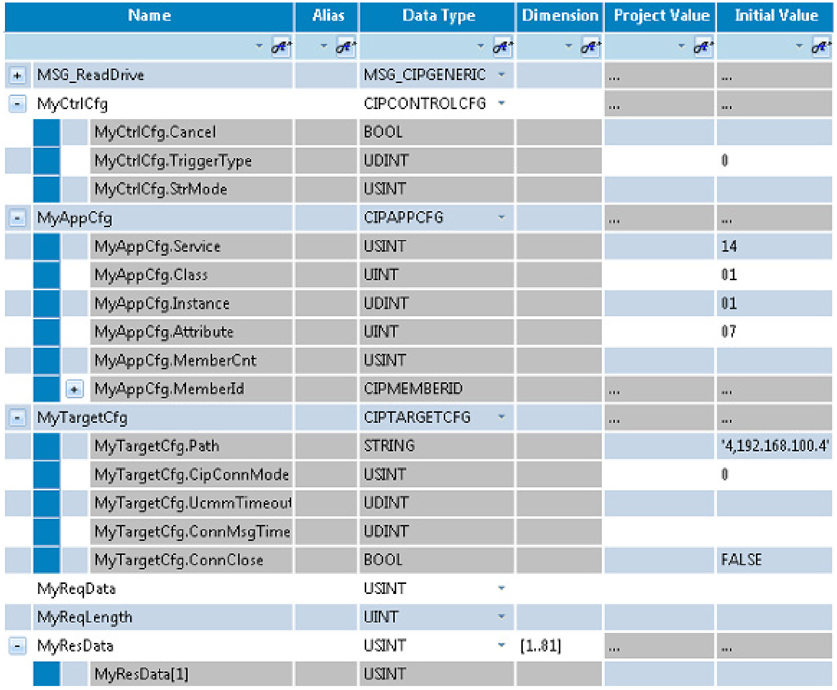
2. To configure initial values for the MyAppCfg input variable
   - From the **Local Variables** page, expand MyAppCfg to view its parameters.
   - Enter the following values in the **Initial Value** column for each parameter.

Parameter	Initial value
MyAppCfg.Service	14
MyAppCfg.Class	01
MyAppCfg.Instance	01
MyAppCfg.Attribute	07

3. To configure initial values for the MyTargetCfg input variable
   - From the **Local Variables** page, expand MyTargetCfg to view its parameters.
   - Enter the following values in the **Initial Value** column for each parameter.

Parameter	Initial Value	Comments
MyTargetCfg.Path	'4,192.168.100.4'	The first '4' indicates the message is being sent out of the embedded Ethernet port. 192.168.100.4 is the IP address of the drive Ethernet interface.
MyTargetCfg.CipConnMode	0	Unconnected is preferred for CIP messages.
MyTargetCfg.UcmmTimeout	blank	Unconnected messages have a timeout default of 3000 milliseconds if their Initial Values are blank.
MyTargetCfg.ConnMsgTimeout	blank	Connected messages have a timeout default of 3000 milliseconds if their Initial Values are blank.
MyTargetCfg.ConnClose	FALSE	For Connected messaging, the CIP connection could be closed immediately after completion of the message instruction by setting the Initial Value to TRUE.

The parameters in the **Variables** page should look similar to the following image.

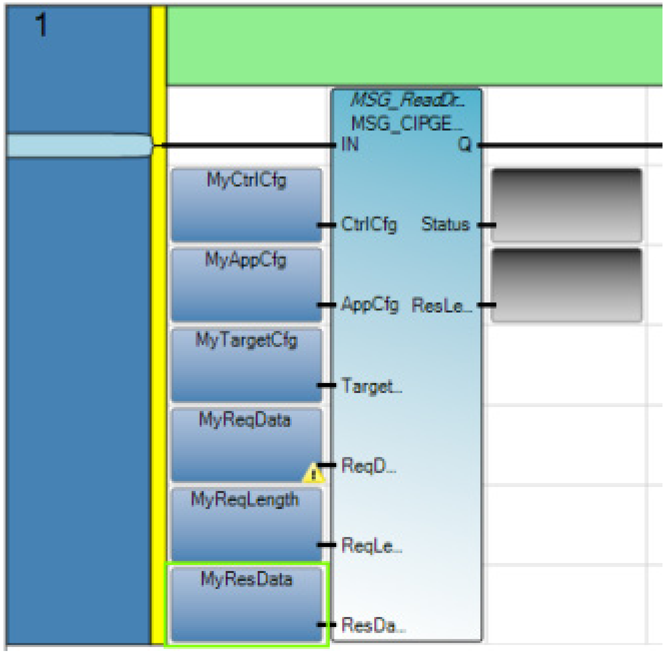


Name	Alias	Data Type	Dimension	Project Value	Initial Value
MSG_ReadDrive		MSG_CIPGENERIC		...	...
MyCtrlCfg		CIPCONTROLCFG		...	...
MyCtrlCfg.Cancel		BOOL			
MyCtrlCfg.TriggerType		UDINT			0
MyCtrlCfg.StrMode		USINT			
MyAppCfg		CIPAPPCFG		...	...
MyAppCfg.Service		USINT			14
MyAppCfg.Class		UINT			01
MyAppCfg.Instance		UDINT			01
MyAppCfg.Attribute		UINT			07
MyAppCfg.MemberCnt		USINT			
MyAppCfg.MemberId		CIPMEMBERID		...	...
MyTargetCfg		CIPTARGETCFG		...	...
MyTargetCfg.Path		STRING			'4,192.168.100.4'
MyTargetCfg.CipConnMode		USINT			0
MyTargetCfg.UcmmTimeout		UDINT			
MyTargetCfg.ConnMsgTime		UDINT			
MyTargetCfg.ConnClose		BOOL			FALSE
MyReqData		USINT			
MyReqLength		UINT			
MyResData		USINT	[1..81]	...	...
MyResData[1]		USINT			

4. To assign the variables to the parameters
   - In the ladder diagram POU, click the top portion of the variable input block to display the variable drop-down list.
   - From the list, assign each input parameter to its correct input variable as identified in the following table.

Parameter	Input variable	Comments
CtrlCfg	MyCtrlCfg	The catalog number must only be retrieved one time so the MyCtrlCfg.TriggerType initial value is set to 0.
AppCfg	MyAppCfg	The initial values were determined by looking up the object data values for Service, Class, Instance and Attribute.
Target	MyTargetCfg	The initial values are for target device configuration.
ReqData	MyReqData	Because this is a Read message, there is no request data so the ReqData parameters is not used.
ReqLength	MyReqLength	Because this is a Read message, there is no request data so the ReqLength parameters is not used.
ResData	MyResData	The catalog number string is stored in the array in the ODVA short string format. The first array element defines the strength length and the rest store the string character's hexadecimal value. The maximum number of characters is 80, plus the length element so MyResData is defined as a 1 dimension array with 81 elements.

The instance of the MSG_CIPGENERIC function block should look similar to the following image.



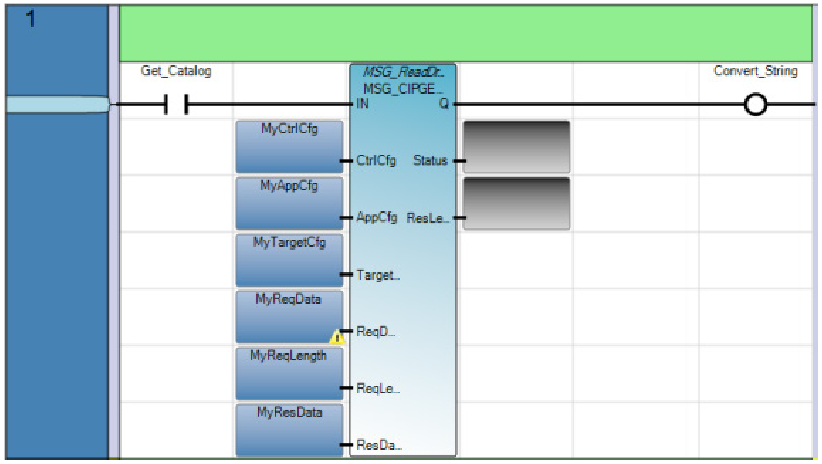
Add a contact and a coil

Use the following steps to add a coil and a contact to the MSG_CIPGENERIC instruction that converts the catalog information to a human readable string.

To add a coil to MSG_CIPGENERIC:

1. In the **Toolbox**, select **Direct Contact** and drag and drop it to the left of the MSG_CIPGENERIC function block input on the first ladder rung.
2. In the **Variable Selector**, type **Get_Catalog** in the Name field for the contact.
3. In the **Toolbox**, select **Direct Coil** and drag and drop it to the right of the MSG_CIPGENERIC function block output on the first ladder rung.
4. In the **Variable Selector**, type **Convert_String** in the Name field for the coil.

The first rung of your ladder diagram program for MSG_CIPGENERIC messaging should look similar to the following image.



Add a COP function block, variables and contact (MSG_CIPGENERIC)

Use the following steps to add a COP function block, variables and a contact. The COP instruction is used to convert data from the source data type (for example, DINT or REAL) to the destination data type. In this example, the catalog information is converted to a human readable string.

To add a COP function block:

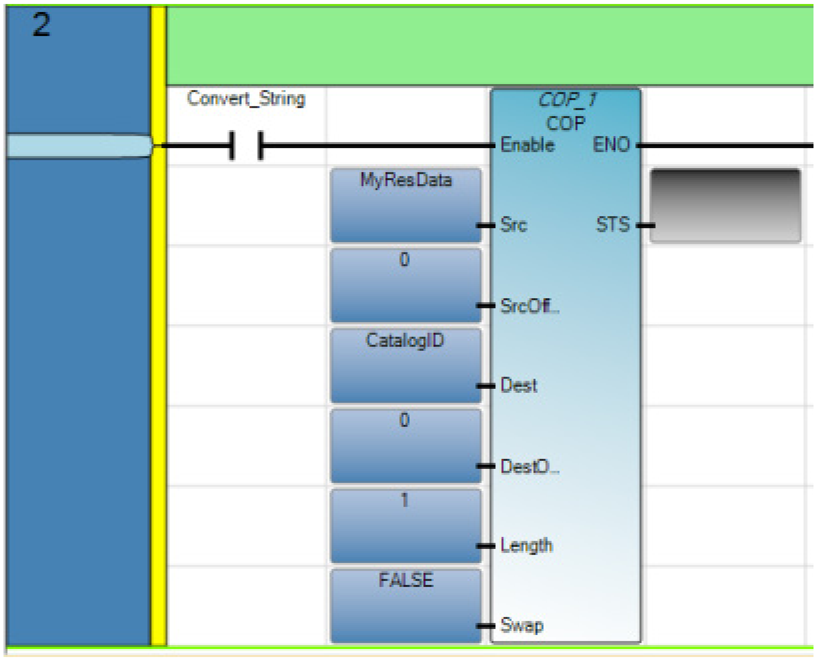
1. In the **Toolbox**, select **Rung** and drag and drop it directly under the first ladder rung to add a second rung.
2. Add the COP function block:
   - In the **Toolbox**, select **Block** and drag and drop it onto the second ladder rung to open the **Instruction Block Selector**.
   - Double-click **COP** to add an instance of the function block to the ladder diagram.
3. Add local input variables for COP:
   - In **Project Organizer**, double-click **Local Variables** to display the **Local Variables** page.
   - In the **Local Variables** page, add the variables and data types listed in the following table.

Parameter	Variable name	Data type
Src	MyResData	Array USINT
SrcOffset	0	UINT
Dest	CatalogID	Array STRING
DestOffset	0	UINT
Length	1	UINT
Swap	FALSE	BOOLEAN

4. For the CatalogID variable, double click **Dimension** and change the array size to [1..1]
5. Add a contact:
   - In the **Toolbox**, select **Direct Contact** and drag and drop it to the left of the COP function block input on the second ladder rung.
   - In the **Variable Selector**, select the **Convert_String** variable for the contact.

Result

The second rung of your ladder diagram program for MSG_CIPGENERIC messaging should look similar to the following image.



Verify correct IP configuration on Controller B

Follow these steps to verify the IP address settings are correct on Controller B.

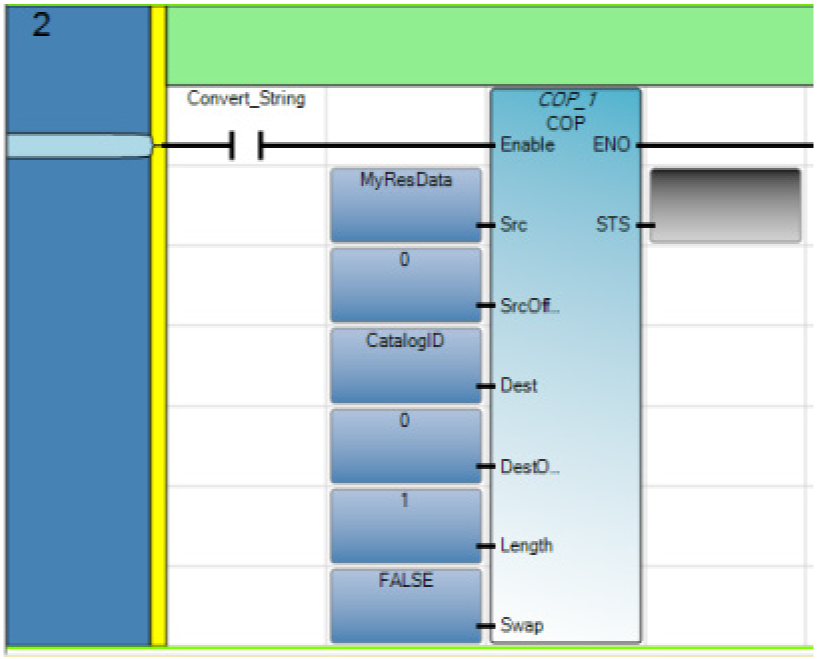
To verify the IP address:

1. Open the application workspace for the controller:
2. In **Project Organizer**, double-click the controller to open the controller workspace.
3. In the controller workspace, expand **Ethernet** in the **Controller** tree and then click Internet Protocol to display the controller configuration page.
4. Verify the IP address settings are correct as identified in the following table.

IP configuration option	Value
IP address	192.168.1.19
Subnet Mask	255.255.255.0
Gateway address	192.168.1.1

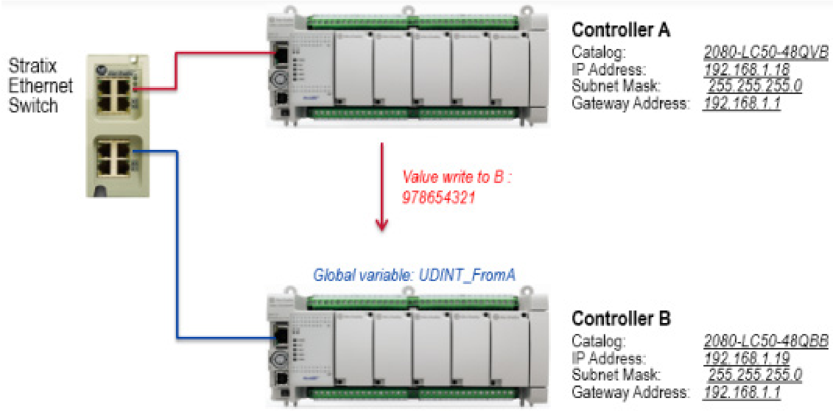
Results

The Internet Protocol options in your controller configuration page should look similar to the following image.



Example: How to create an MSG_CIPSYMBOLIC messaging program to write a value to a variable

This example shows how to create a message program to write a value to a Controller B global variable from Controller A.



Controller A
Catalog: 2080-LC50-48QVB
IP Address: 192.168.1.18
Subnet Mask: 255.255.255.0
Gateway Address: 192.168.1.1

Controller B
Catalog: 2080-LC50-48QBB
IP Address: 192.168.1.19
Subnet Mask: 255.255.255.0
Gateway Address: 192.168.1.1

To create a MSG_CIPSYMBOLIC messaging program used to write a value to a global variable on another controller, perform the following tasks.

No	Task
1	Add a COP function block, variables, and a contact (MSG_CIPSYMBOLIC) on page 205
2	Add an Equal operator and a coil on page 206
3	Add a MSG_CIPSYMBOLIC function block, variables and a contact on page 207
4	Verify correct IP configuration on Controller B on page 203
5	Create global variable for Controller B on page 210
6	Review the complete program results on page 210

Add a COP function block, variables and a contact (MSG_CIPSYMBOLIC)

The COP instruction is used to convert the data you enter to the destination data type so the data is compatible with the controller variable.

To add a COP function block:

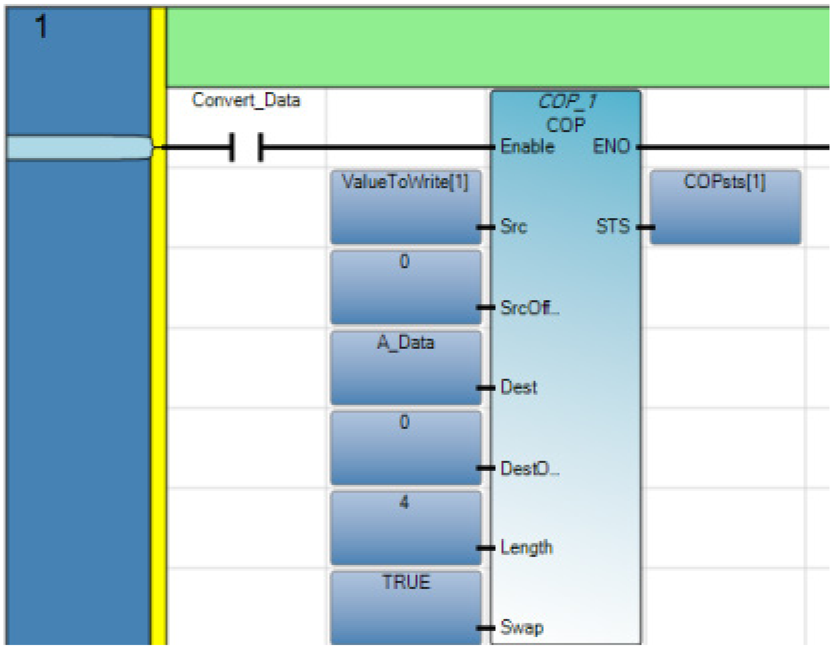
1. Add a controller:
   - Expand the **Controllers** folder and the Micro850 folder to view all Micro850 controllers.
   - Double-click a controller (2080-LC50-48QVB) to add it to **Project Organizer**.
2. Add a ladder diagram program:
   - In **Project Organizer**, right-click **Programs**, click **Add**, and then click **New LD: Ladder Diagram**.
   - Right-click the ladder diagram icon in **Project Organizer**, click **Rename** and type **CIPSymbolicMessage**.
   - Double-click the ladder diagram program in **Project Organizer** to display the LD POU in the language editor.
3. Add a COP function block:
   - In the **Toolbox**, select **Instruction Block** and drag and drop it onto the first ladder rung to open the **Instruction Block Selector**.
   - In Search, type **COP**, and double-click **COP** to add an instance of the function block to the ladder diagram.
4. Add COP variables and initial values:
   - In the ladder diagram POU, double-click **Local Variables** to open the **Local Variables** page.
   - In the **Variables** page, add the variables and data types listed in the table below.
5. Create Arrays:
   - For ValueToWrite, double-click **Dimension** and change the array size to [1..1].
   - For A_Data, double-click **Dimension** and change the array size to [1..4].
6. Enter the data from the **Value** column of the table below into the **Initial Value** field for each variable.
7. Add a contact:
   - In the **Toolbox**, select **Direct Contact** and drag and drop it to the left of the COP function block input on the first ladder rung.
   - In the **Variable Selector**, assign a variable named **Convert_Data** to contact.

Use the variables defined in the table for the COP function block.

Parameter	Variable name	Data type

Parameter	Variable name	Data type
Src	ValueToWrite	Array UDINT Initial value: 987654321
SrcOffset	0	UINT
Dest	A_Data	Array USINT
DestOffset	0	UINT
Length	4	UINT
Swap	TRUE	BOOLEAN
STS	COPsts	Array UINT

The first rung of your ladder diagram program for MSG_CIPSYMBOLIC messaging should look similar to the following image.



Add an Equal operator and a coil

The Equal instruction is used to trigger writing a value if the data type conversion was successful. To add an Equal (=) operator, variables and a coil, perform the following steps.

To add an Equal operator:

1. In the **Toolbox**, select **Rung** and drag and drop it directly under the first ladder rung to add a second rung.
2. Add an Equal operator:
   - In the **Toolbox**, select **Instruction Block** and drag and drop it onto the second ladder rung to open the **Instruction Block Selector**.
   - In Search, type the '=' sign and double-click **'='** to add an instance of the operator to the ladder diagram.
3. To add Equal variables:
   - In the ladder diagram POU, double-click a variable to display the **Variable Selector**.

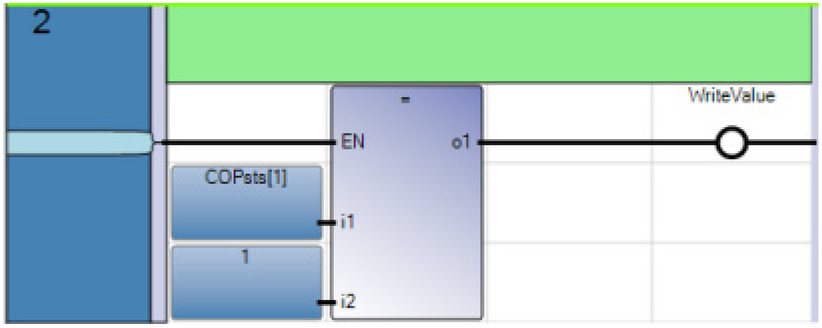
- In the **Variable Selector**, assign variable names as listed in the following table.

Parameter	Variable name
i1	COPsts
i2	1

4. To add a coil to the Equal operator:

   In the **Toolbox**, select **Direct Coil** and drag and drop it to the right of the Equal operator output on the second ladder rung.
   In the **Variable Selector**, type **WriteValue** in the Name field for the coil.

The second rung of your ladder diagram program for MSG_CIPGENERIC messaging should look similar to the following image.



Add a MSG_CIPSYMBOLIC function block, variables and a contact

To add a MSG_CIPSYMBOLIC function block, input variables and a contact to a ladder diagram, perform the following steps.

To add function block and variables:

1. In the **Toolbox**, select **Rung** and drag and drop it directly under the second ladder rung to add a third rung.
2. Add the MSG_CIPSYMBOLIC function block:
   - In the **Toolbox**, select **Instruction Block** and drag and drop it onto the ladder rung to display the **Instruction Block Selector**.
   - In Search, type **MSG** to display the message function blocks.
   - Type **MSG_WriteVariable** in the **Instance** field.
   - Double-click **MSG_CIPSYMBOLIC** to add an instance of the function block called MSG_WriteVariable to the ladder diagram.
3. Assign variable names:
   - In the ladder diagram POU, double-click a variable to display the **Variable Selector**.
   - In the **Variable Selector**, assign variable names as listed in the following table.
4. For the Data variable, double click **Dimension** and change the array size to [1...4].

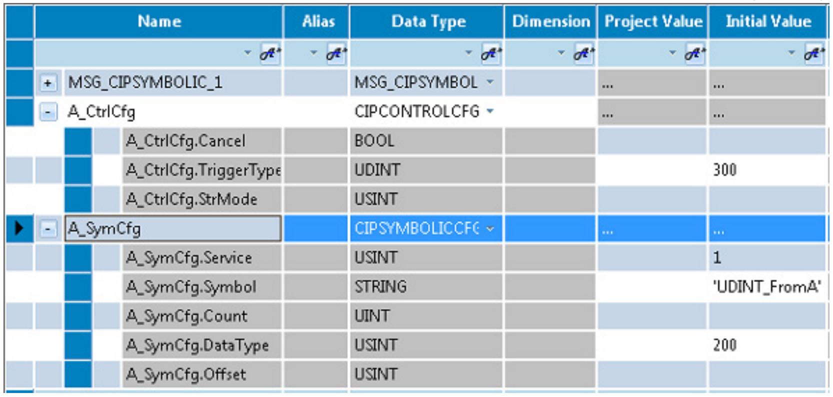
5. Configure initial values for the local variables, add CtrlCfg initial values:
   - From the **Local Variables** page, expand the CtrlCfg parameter to view additional parameters.
   - Enter the following values in the Initial Value column for each parameter.

Parameter	Initial value
A_CtrlCfg.Cancel	Leave blank
A_CtrlCfg.TriggerType	300
A_Ctrlcfg.StrMode	Leave blank

6. Add SymCfg initial values:
   - From the **Local Variables** page, expand the SymCfg parameter to view additional parameters.
   - Enter the following values in the Initial Value column for each parameter.

Parameter	Initial value
A_SymCfg.Service	1
A_SymCfg.Symbol	'UDINT_FromA'
A_SymCfg.Count	Leave blank
A_SymCfg.DataType	200
A_SymCfg.Offset	Leave blank

The Local Variables selector should look similar to the following image.

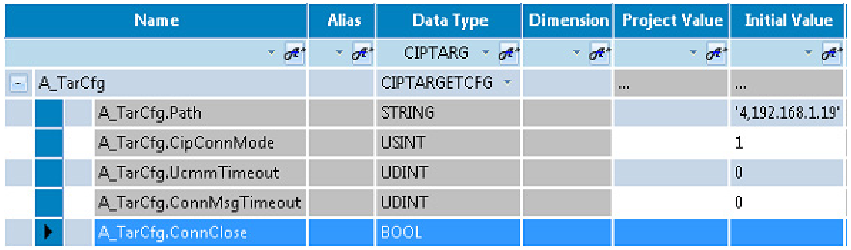


Name	Alias	Data Type	Dimension	Project Value	Initial Value
+ MSG_CIPSYMBOLIC_1		MSG_CIPSYMBOL		...	...
- A_CtrlCfg		CIPCONTROLCFG		...	...
A_CtrlCfg.Cancel		BOOL			
A_CtrlCfg.TriggerType		UDINT			300
A_CtrlCfg.StrMode		USINT			
- A_SymCfg		CIPSYMBOLICCF(		...	...
A_SymCfg.Service		USINT			1
A_SymCfg.Symbol		STRING			'UDINT_FromA'
A_SymCfg.Count		UINT			
A_SymCfg.DataType		USINT			200
A_SymCfg.Offset		USINT			

7. Configure initial values for TargetCfg
   - From the ladder diagram POU, double-click the A_TarCfg variable to open the global variables selector.
   - Expand the TargetCfg parameter to view additional parameters.
   - Enter the following values in the Initial Value column for each parameter.

Parameter	Initial value
A_TarCfg.Path	'4,192.168.1.19'
A_TarCfg.CipConnMode	1
A_TarCfg.UcmmTimeout	0
A_TarCfg.ConnMsgTimeout	0
A_TarCfg.ConnClose	Leave blank

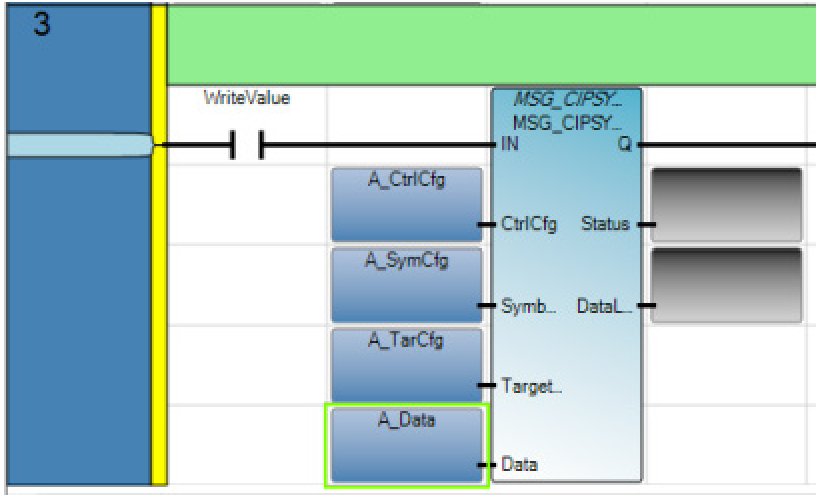
The User Global Variables selector should similar to the following image.



Name	Alias	Data Type	Dimension	Project Value	Initial Value
		CIPTARG			
A_TarCfg		CIPTARGETCFG		...	...
A_TarCfg.Path		STRING			'4,192.168.1.19'
A_TarCfg.CipConnMode		USINT			1
A_TarCfg.UcmmTimeout		UDINT			0
A_TarCfg.ConnMsgTimeout		UDINT			0
A_TarCfg.ConnClose		BOOL			

8. Add a contact:
   - In the **Toolbox**, select **Direct Contact** and drag and drop it to the left of the MSG_CIPSYMBOLIC function block input on the third ladder rung.
   - In the **Variable Selector**, assign the **WriteValue** variable to the contact.

The third rung of your ladder diagram program for MSG_CIPSYMBOLIC messaging should look similar to the following image.



Verify correct IP configuration on Controller B

Follow these steps to verify the IP address settings are correct on Controller B.

To verify the IP address:

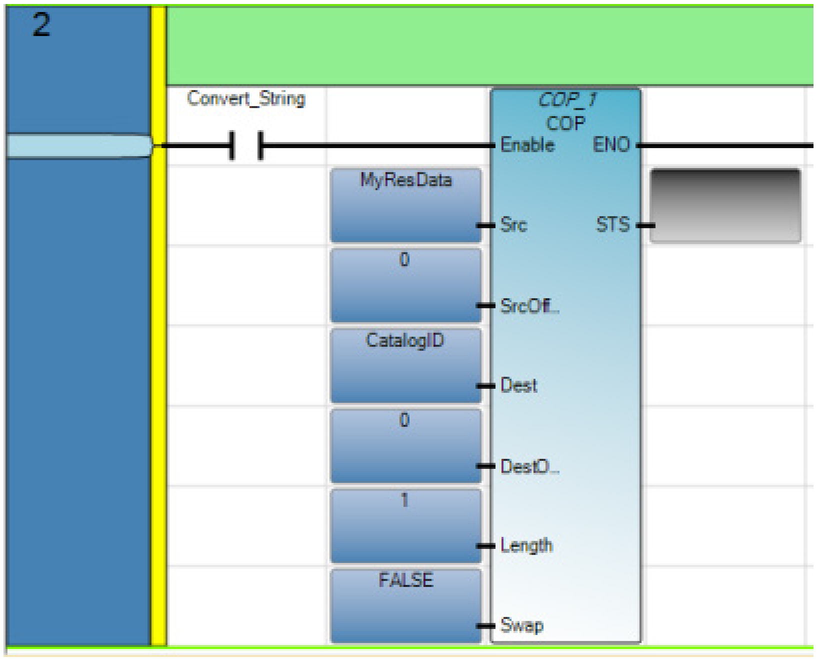
1. Open the application workspace for the controller:
2. In **Project Organizer**, double-click the controller to open the controller workspace.
3. In the controller workspace, expand **Ethernet** in the **Controller** tree and then click Internet Protocol to display the controller configuration page.
4. Verify the IP address settings are correct as identified in the following table.

IP configuration option	Value

IP address	192.168.1.19
Subnet Mask	255.255.255.0
Gateway address	192.168.1.1

Results

The Internet Protocol options in your controller configuration page should look similar to the following image.



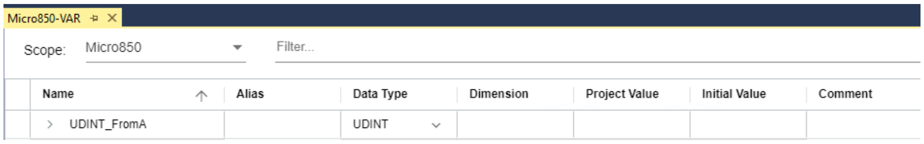
Create global variable for Controller B

Follow these steps to create a Global variable for controller B.

To create a Global variable:

1. In **Project Organizer**, double-click **Global Variables** to display the **Global Variables** selector.
2. Type UDINT_FromA in the **Name** column.
3. Configure the remaining parameters:
   - Verify the data type is UDINT.
   - Verify the **Read/Write** attribute is selected.

The **Global Variables** selector should look similar to the following image.

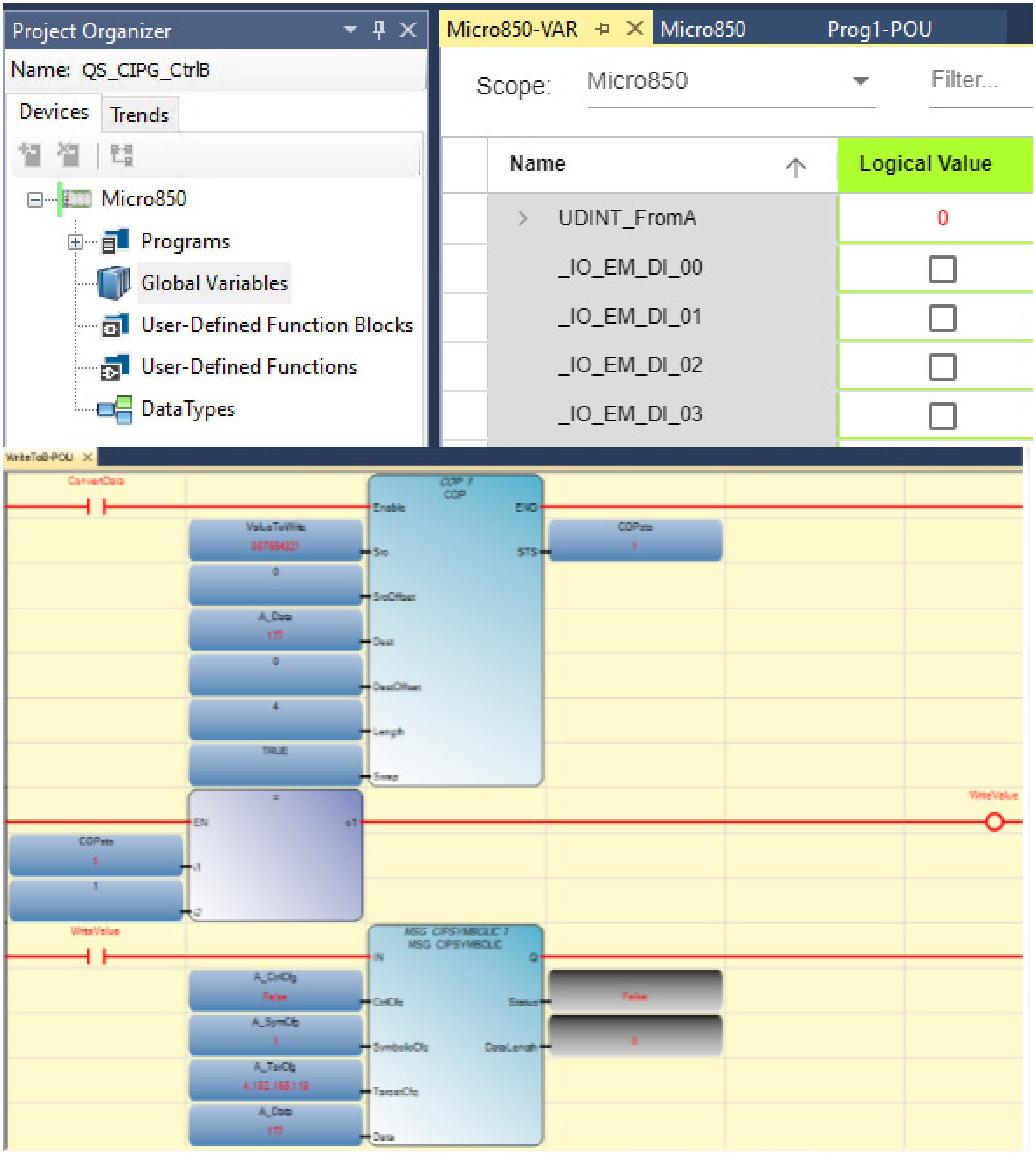


Scope: Micro850

Name	Alias	Data Type	Dimension	Project Value	Initial Value	Comment
UDINT_FromA		UDINT				

Review the complete program results

The following example shows the complete MSG_CIPSYMBOLIC messaging program after it has executed.



Scope: Micro850

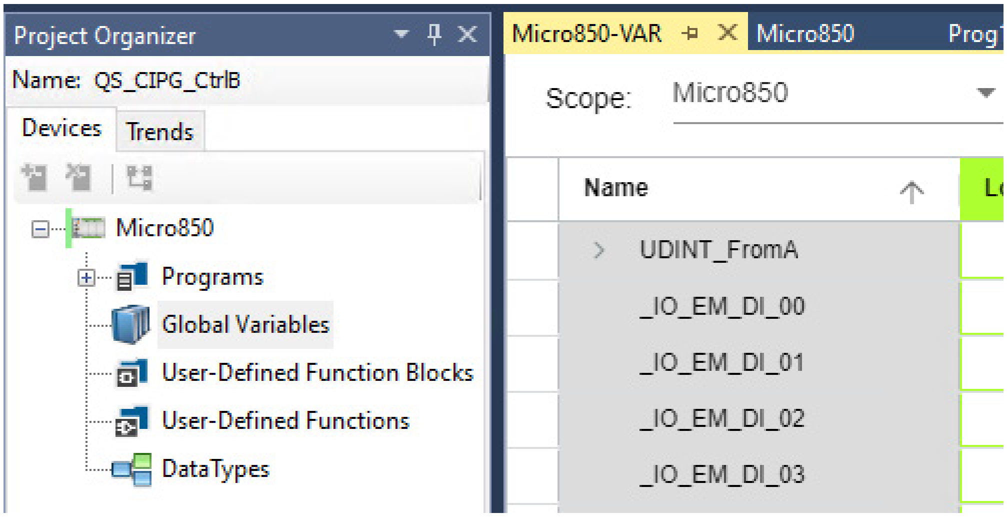
Filter...

Name	Logical Value
UDINT_FromA	0
_IO_EM_DI_00	□
_IO_EM_DI_01	□
_IO_EM_DI_02	□
_IO_EM_DI_03	□

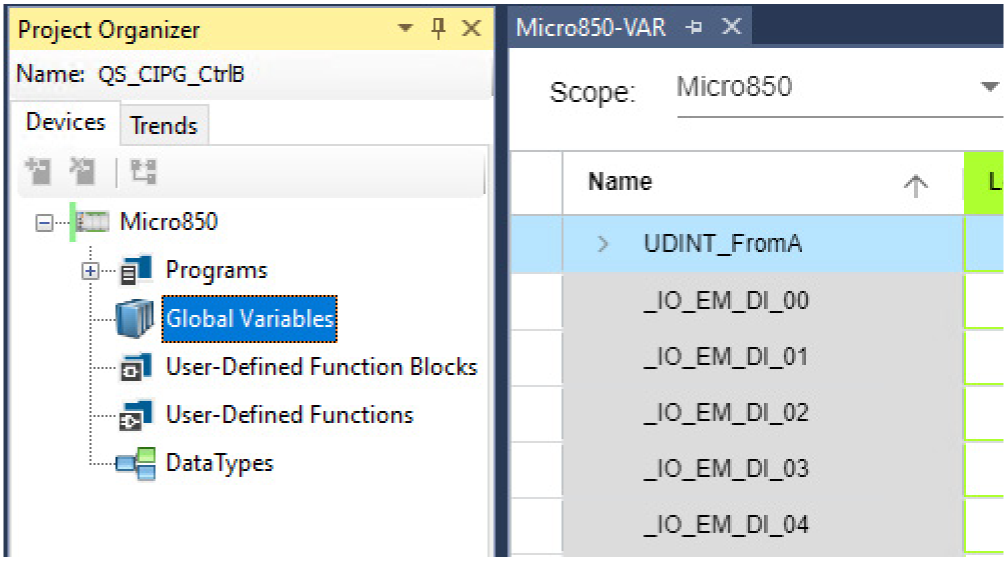
Controller B results

The following two images show the values for Controller B before and after the messaging program is executed.

Before the program executes



After the program executes

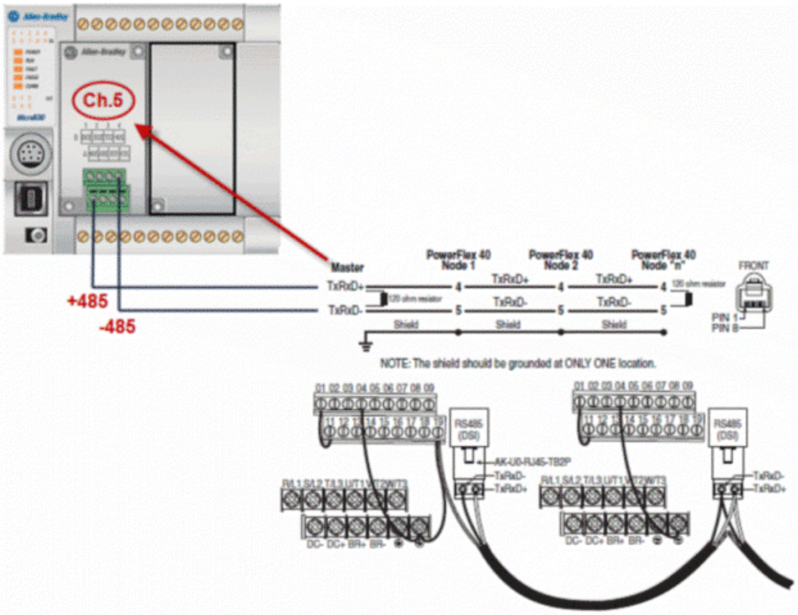


Example: How to configure Modbus communication to read from and write to a drive

These examples describe how to configure Modbus communication to read status data from and write control data to a PowerFlex 4 drive using the MSG_MODBUS on page 177 instruction.

Micro830 wiring

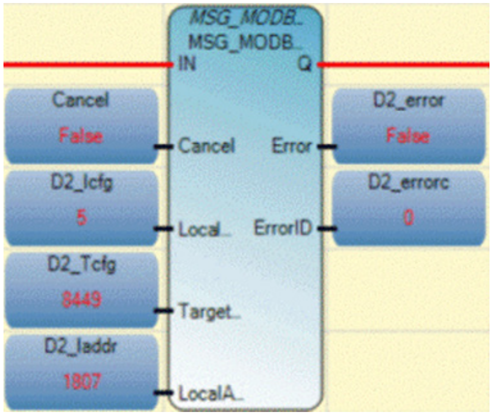
This example uses a Micro830 controller with a SERIALISOL module plugged into the first slot (Channel 5). A single PowerFlex 40 is connected, but the diagram below shows how to wire for multi-drop. Refer to the user manual for additional wiring information.



AK-U0-RJ45-TB2P is an RJ45 connector with 2 terminal blocks for RS485 communications

Modbus Read example

The following MSG_MODBUS instruction can be used to read status data from the PowerFlex 40 drive.


MSG_MODB...

MSG_MODB...

IN Q

Cancel: Cancel False

Local...: D2_lcfg 5

Target...: D2_Tcfg $449

LocalA...: D2_laddr 1807

Error: D2_error False

ErrorID: D2_errorc 0

Drive status

An "1807" indicates the drive is

- Ready (bit 0 ON),
- Active (bit 1 ON)
- Commanded Forward (bit 2 ON)
- Rotating Forward (bit 3 ON)

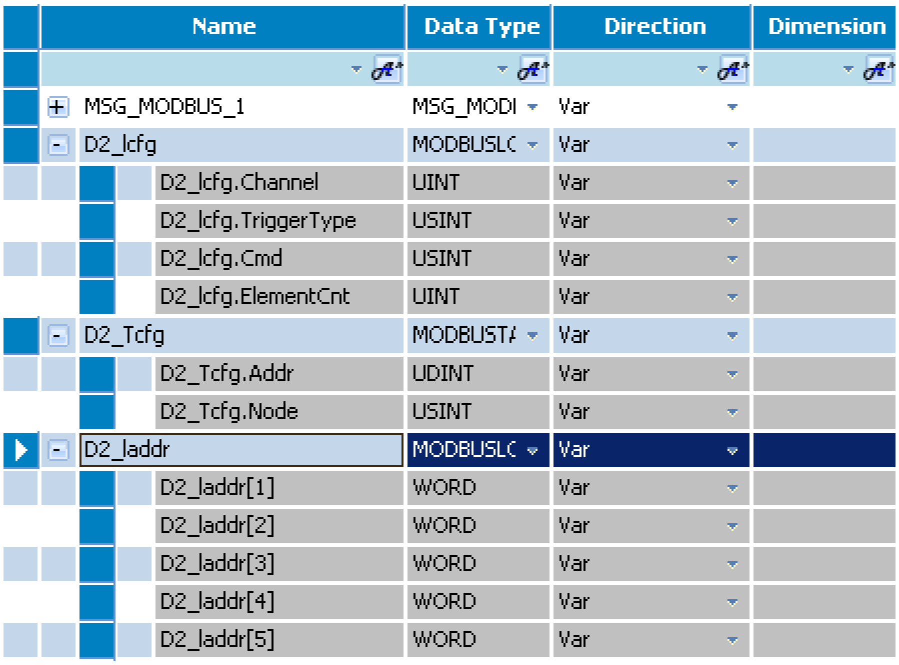
- Status of some digital inputs on the drive

A "278" indicates 27.8Hz.

Refer to the PowerFlex user manual for additional information about Logic Status word bits, error code descriptions, commanded and actual speeds, and other status codes.

MSG_MODBUS Read configuration

The following image shows the variable options for the MSG_MODBUS instruction block used to read status data from a PowerFlex 40 drive.



Name	Data Type	Direction	Dimension
MSG_MODBUS_1	MSG_MODI	Var	
D2_lcfg	MODBUSLC	Var	
D2_lcfg.Channel	UINT	Var	
D2_lcfg.TriggerType	USINT	Var	
D2_lcfg.Cmd	USINT	Var	
D2_lcfg.ElementCnt	UINT	Var	
D2_Tcfg	MODBUST/	Var	
D2_Tcfg.Addr	UDINT	Var	
D2_Tcfg.Node	USINT	Var	
D2_laddr	MODBUSLC	Var	
D2_laddr[1]	WORD	Var	
D2_laddr[2]	WORD	Var	
D2_laddr[3]	WORD	Var	
D2_laddr[4]	WORD	Var	
D2_laddr[5]	WORD	Var	

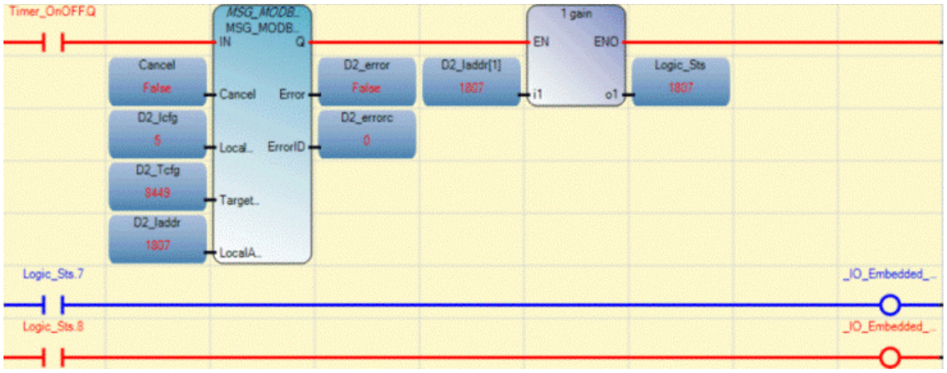
MSG_MODBUS Read variables

The following table identifies the variables and the values used to configure the MSG_MODBUS instruction to read status data from a PowerFlex 4 drive.

Variable	Value	Description
*.Channel	5	Channel 5 - location of SERIALISOL module
*.TriggerType	0	Trigger on False-to-True transition
*.Cmd	3	Modbus Function Code "03" - Read Holding Registers
*.ElementCnt	4	Length
*.Addr	8449	PowerFlex Logic Status word address + 1
*.Node	2	PowerFlex Node address
*_laddr[1]	{data}	PowerFlex Logic Status word
*_laddr[2]	{data}	PowerFlex Error Code
*_laddr[3]	{data}	PowerFlex Commanded Speed (Speed Reference)
*_laddr[4]	{data}	PowerFlex Speed Feedback (Actual Speed)

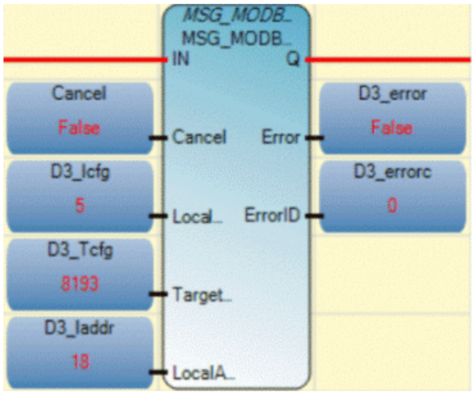
MOV instruction example

The following images shows an example of using the MOV instruction to move the *_l[1] array value to a Word, which allows you to directly access the individual bits.



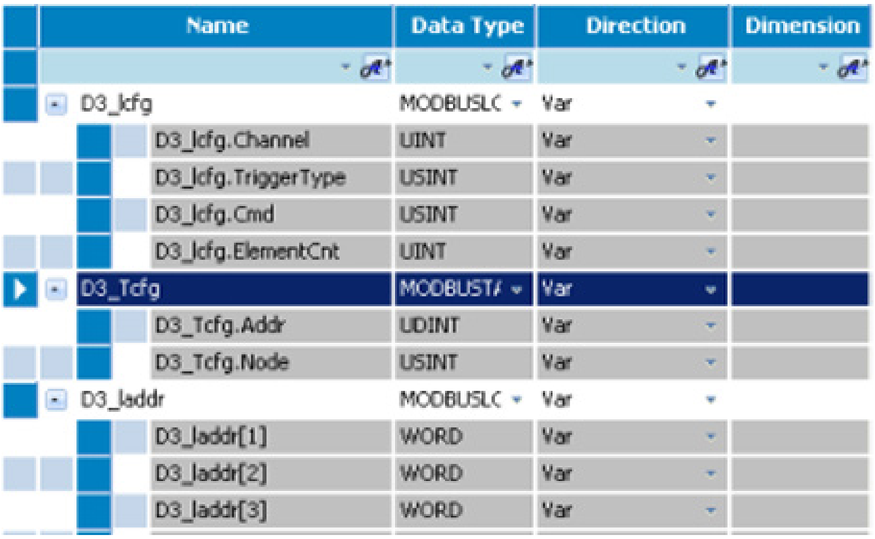
Modbus Write example

The following MSG_MODBUS instruction is used to write control data to a PowerFlex 40 drive.



MSG_MODBUS Write configuration

The following image shows the variables and the values used to configure the MSG_MODBUS instruction to write control data to a PowerFlex 4 drive.



Name	Data Type	Direction	Dimension
D3_lcfg	MODBUSLC	Var	
D3_lcfg.Channel	UINT	Var	
D3_lcfg.TriggerType	USINT	Var	
D3_lcfg.Cmd	USINT	Var	
D3_lcfg.ElementCnt	UINT	Var	
D3_Tcfg	MODBUST/	Var	
D3_Tcfg.Addr	UDINT	Var	
D3_Tcfg.Node	USINT	Var	
D3_laddr	MODBUSLC	Var	
D3_laddr[1]	WORD	Var	
D3_laddr[2]	WORD	Var	
D3_laddr[3]	WORD	Var	

MSG_MODBUS Write variables

The following table lists the variables, values and describes the purpose of each variable.

Variable	Value	Description
*.Channel	5	Channel 5 - location of SERIALISOL module
*.TriggerType	0	Trigger on False-to-True transition
*.Cmd	16	Modbus Function Code "16" - Write Holding Registers
*.ElementCnt	2	Length
*.Addr	8193	PowerFlex Logic Status word address + 1
*.Node	2	PowerFlex Node address
*_laddr[1]	{data}	PowerFlex Logic Command word
*_laddr[2]	{data}	PowerFlex Speed Reference word

Communication protocol support

The MSG_CIP function blocks support different communication protocols as described in this section.

Function block support for message communication protocols.

Messaging Protocol	Communication media	Use this function block
Modbus/RTU client and server	Through a Serial port configured as Modbus RTU.	MSG_MODBUS on page 177
Modbus/TCP client and server	Over the Ethernet instead of through a serial port.	MSG_MODBUS2 on page 182
Ethernet IP client and server	Through an embedded Ethernet channel.	MSG_CIPSYMBOLIC on page 173 MSG_CIPGENERIC on page 166
CIP Serial client and server	Ethernet cable or Serial cable.	MSG_CIPSYMBOLIC on page 173

ASCII	Through an RS-232 port configured with an ASCII driver.	See ASCII serial port instructions on page 105.

Modbus RTU

Modbus is a half-duplex, master-slave communications protocol that allows a single master to communicate with a maximum of 247 slave devices. The Modbus network master reads and writes bits and registers. Modbus RTU is supported by configuring the Serial port as Modbus RTU.

For more information about the Modbus protocol, refer to the Modbus Protocol Specifications (available from http://www.modbus.org).

Modbus/TCP

The Modbus/TCP Server communication protocol uses the same Modbus mapping features as Modbus RTU, but it is supported over the Ethernet instead of through a Serial port.

The Micro850 controller supports up to 16 simultaneous Modbus TCP Server connections. No protocol configuration is required other than configuring the Modbus mapping table.

EtherNet/IP

Micro820 and Micro850 controllers support up to 16 simultaneous EtherNet/IP server connections through an embedded Ethernet communication channel on page 218. The channel can be used to connect a Micro850 controller to various devices through a local area network using a 10 Mbps/100 Mbps transfer rate.

Common Industrial Protocol (CIP)

The CIP protocol is a transport and application layer protocol used for messaging over EtherNet/IP, ControlNet, and DeviceNet networks that allows HMIs to easily connect to a Micro830 or a Micro850 controller.

CIP explicit messaging

The CIP protocol is used for explicit messaging. Explicit Messaging (configuration, data collection, and diagnostics) is used to transfer data that does not require continuous updates. Explicit messaging is supported using CIP Generic client messaging through the MSG_CIPGENERIC function block and using CIP Symbolic client messaging through the MSG_CIPSYMBOLIC function block.

CIP Serial

CIP serial uses DF1 Full Duplex protocol, and provides point-to-point connection between two devices. It combines data transparency (American National Standards Institute ANSI - X3.28-1976 specification subcategory D1) and 2-way simultaneous transmission with embedded responses (subcategory F1)

Micro830 and Micro850 controllers support CIP Serial through an RS-232 connection to external devices, such as computers running RSLinx Classic software, PanelView Component terminals (firmware revisions 1.70 and above), or other controllers that support CIP Serial over DF1 Full-Duplex, such as ControlLogix and CompactLogix controllers that have embedded serial ports.

The Serial Port driver can be configured as CIP Serial, Modbus RTU, ASCII or Shutdown through the device configuration tree.

Embedded communication channels

The Micro820, Micro830, and Micro850 controllers have the following additional embedded communication channels:

- A non-isolated RS-232/485 combo port
- A non-isolated USB programming port (Micro830 and Micro850 only)
- An RJ-45 ethernet port (Micro820, and Micro850 only)

Compare instructions

Use Compare instructions to compare Integer, Real, Time, Date, and String values using an expression or a specific compare instruction.

Instruction	Description
(=) Equal on page 219	Compares the first input to the second input to determine equality. For Integer, Real, Time, Date, and String data types.
(>) Greater Than on page 222	Compares input values to determine whether the first is greater than the second.
(>=) Greater Than or Equal on page 224	Compares input values to determine whether the first is greater than or equal to the second.
(<) Less Than on page 225	Compares input values to determine whether the first is less than the second.
(<=) Less Than or Equal on page 227	Compare input values to determine whether the first is less than or equal to the second.
(<>) Not Equal on page 228	Compares input values to determine whether the first is not equal to the second.

Equal

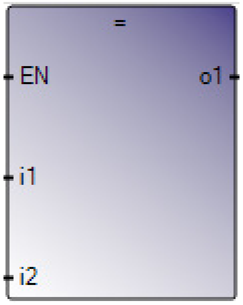
Performs an operation that compares the first input to the second input to determine equality for Integer, Real, Time, Date, and String data types.

Operation details:

- Equality testing of Time values is not recommended for TON, TP, and TOF instruction blocks.
- The Real data type is not recommended when comparing values for equality because numbers in the math operation are rounded differently than those that appear in the variable output display. Consequently, two output values may appear equal in the display, though evaluate as false. For example, 23.500001 compared to 23.499999, both display as 23.5 in the variable input display, but are not equal in the controller.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Function enable. TRUE - execute the equality comparison. FALSE - there is no comparison. Applies only to Ladder Diagram programs.
i1	Input	BOOL SINT USINT BYTE INT UINT WORD DINT UDINT DWORD LINT ULINT LWORD REAL LREAL TIME DATE STRING	All inputs must be the same data type. The Time input applies to the Structured Text, Ladder Diagram and Function Block Diagram languages. The Real data type is not recommended.

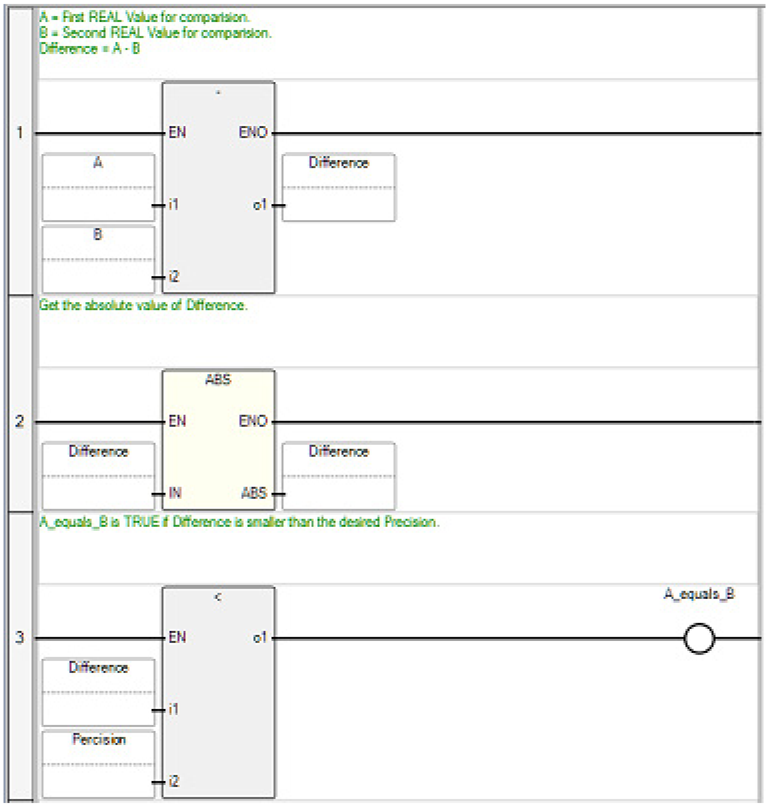
Parameter	Parameter Type	Data Type	Description
i2	Input	BOOL SINT USINT BYTE INT UINT WORD DINT UDINT DWORD LINT ULINT LWORD REAL LREAL TIME DATE STRING	
o1	Output	BOOL	TRUE if i1 = i2.

Compare Real Values using Subtraction (-) ABS, and Less than (<) example

The Real data type is not recommended when comparing values for equality because of differences in the way numbers are rounded. Two output values may appear equal in a Connected Components Workbench display, but will evaluate as false.

For example, 23.500001 compared to 23.499999 will both display as 23.5 in the variable input display, but will not be equal in the controller.

To test whether two Real data type values are equal, you can use a Subtraction instruction to get the difference between the values and then determine if the difference is Less Than an established precision value. See the following LD program example for comparing two Real data type values.



A = First REAL Value for comparison.
B = Second REAL Value for comparison.
Difference = A - B

Get the absolute value of Difference.

A_equals_B is TRUE if Difference is smaller than the desired Precision.

Equal (=) Structured Text example

(* ST Equivalence: *)

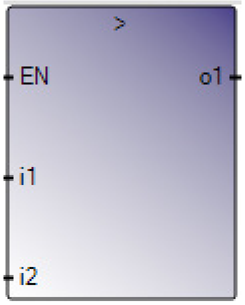
```
aresult := (10 = 25); (* aresult is FALSE *)
mresult := ('ab' = 'ab'); (* mresult is TRUE *)
```

Greater than

Compares Integer, Real, Time, Date, and String input values to determine whether the first is greater than the second.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute the input comparison. FALSE -there is no comparison. Applies only to Ladder Diagram programs.
i1	Input	SINT USINT BYTE INT UINT WORD DINT UDINT DWORD LINT ULINT LWORD REAL LREAL TIME DATE STRING	All inputs must be the same data type.

i2	Input	SINT USINT - BYTE INT UINT WORD DINT UDINT DWORD LINT ULINT LWORD REAL LREAL TIME DATE STRING	
o1	Output	BOOL	TRUE if i1 > i2.

Greater than (>) Structured Text example

(* ST Equivalence: *)

```
aresult := (10 > 25); (* aresult is FALSE *)
mresult := ('ab' > 'a'); (* mresult is TRUE *)
```

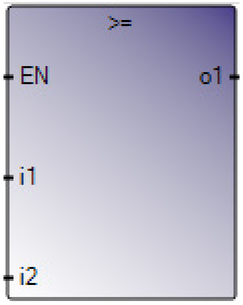
Greater than or equal

Compares Integer, Real, Time, Date, and String input values to determine whether the first is greater than or equal to the second.

For TON, TP, and TOF, equality testing of Time values is not recommended.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description

EN	Input	BOOL	Instruction enable. TRUE - execute the input comparison. FALSE - there is no comparison. Applies only to Ladder Diagram programs.
i1	Input	SINT USINT BYTE INT UINT WORD DINT UDINT DWORD LINT ULINT LWORD REAL LREAL TIME DATE STRING	All inputs must be the same data type. The Time input applies to the Structured Text, Ladder Diagram and Function Block Diagram languages.
i2	Input	SINT USINT BYTE INT UINT WORD DINT UDINT DWORD LINT ULINT LWORD REAL LREAL TIME DATE STRING	
o1	Output	BOOL	TRUE if i1 >= i2.

Greater than or equal (>=) Structured Text example

(* ST Equivalence: *)

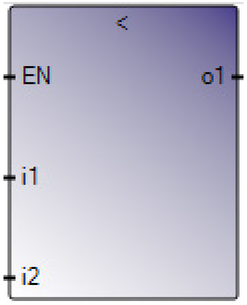
```
aresult := (10 >= 25); (* aresult is FALSE *)
mresult := ('ab' >= 'ab'); (* mresult is TRUE *)
```

Less than

Compares Integer, Real, Time, Date, and String input values to determine whether the first is less than the second.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute the input comparison. FALSE - there is no comparison. Applies only to Ladder Diagram programs.
i1	Input	SINT USINT BYTE INT UINT WORD DINT UDINT DWORD LINT ULINT LWORD REAL LREAL TIME DATE STRING	All inputs must be the same data type.

i2	Input	SINT USINT BYTE INT UINT WORD DINT UDINT DWORD LINT ULINT LWORD REAL LREAL TIME DATE STRING	
o1	Output	BOOL	TRUE if i1 < i2.

Less than (<) Structured Text example

(* ST Equivalence: *)

```
aresult := (10 < 25); (* aresult is TRUE *)
mresult := ('z' < 'B'); (* mresult is FALSE *)
```

(* IL equivalence: *)

LD	10
LT	25
ST	aresult
LD	'z'
LT	'B'
ST	mresult

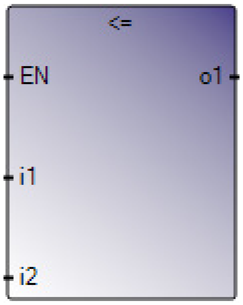
Less than or equal

Compares Integer, Real, Time, Date, and String input values to determine whether the first is less than or equal to the second.

For TON, TP, and TOF, equality testing of Time values is not recommended.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute the input comparison. FALSE - there is no comparison. Applies only to Ladder Diagram programs.
i1	Input	SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME - DATE - STRING	All inputs must be the same data type. The Time input applies to the Structured Text, Ladder Diagram and Function Block Diagram languages.
i2	Input	SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME - DATE - STRING	
o1	Output	BOOL	TRUE if i1 <= i2.

Less than or equal (<=) Structured Text example

(* ST Equivalence: *)

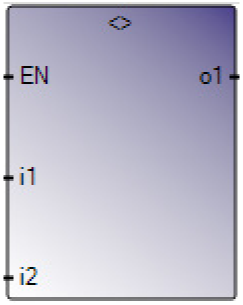
```
aresult := (10 <= 25); (* aresult is TRUE *)
mresult := ('ab' <= 'ab'); (* mresult is TRUE *)
```

Not equal

Compares Integer, Real, Time, Date, and String input values to determine whether the first is not equal to the second.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute current compare computation. FALSE - there is no computation. Applies only to Ladder Diagram programs.
i1	Input	BOOL SINT USINT BYTE INT UINT WORD DINT UDINT DWORD LINT ULINT LWORD REAL LREAL TIME DATE STRING	All inputs must be the same data type.

i2	Input	BOOL SINT USINT BYTE INT UINT WORD DINT UDINT DWORD LINT ULINT LWORD REAL LREAL TIME DATE STRING	
o1	Output	BOOL	TRUE - if first <> second.

Not equal (<>) Structured Text example

(* ST Equivalence: *)

```
aresult := (10 <> 25); (* aresult is TRUE *)
mresult := ('ab' <> 'ab'); (* mresult is FALSE *)
```

Counter instructions

Use Counter instructions to control operations based on the number of events.

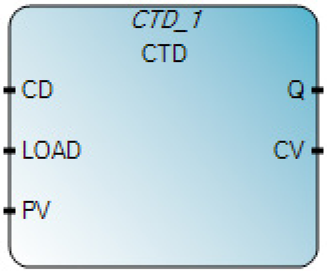
Instruction	Description
CTD on page 231	Counts integers from a given value down to 0, 1 by 1.
CTU on page 233	Counts integers from 0 up to a given value, 1 by 1.
CTUD on page 235	Counts integers from 0 up to a given value, 1 by 1, or from a given value down to 0, 1 by 1.

CTD (count down)

Counts integers from a given value down to 0, 1 by 1.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

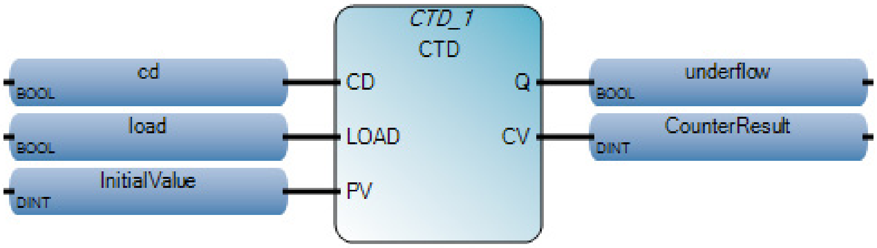
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



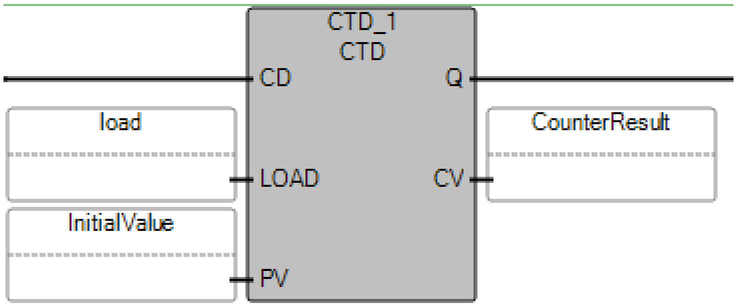
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
CD	Input	BOOL	Counts down. TRUE - Rising edge detected, count down in increments of one. FALSE - Falling edge detected, hold the counter value at the same value.
LOAD	Input	BOOL	Load verifies the PV value against the count down value. TRUE - set CV=PV. FALSE - Continue incrementing count down by one.
PV	Input	DINT	Programmed maximum value of the counter.
Q	Output	BOOL	Indicates whether the count down instruction has resulted in a number less than or equal to the maximum value of the counter. TRUE - Counter result <=0 (Underflow condition). FALSE - Counter result >0.
CV	Output	DINT	Current counter value.

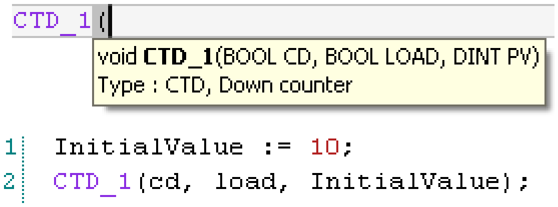
CTD Function Block Diagram example



CTD Ladder Diagram example



CTD Structured Text example



```
CTD_1(
void CTD_1(BOOL CD, BOOL LOAD, DINT PV)
Type : CTD, Down counter
```

```
InitialValue := 10;
CTD_1(cd, load, InitialValue);
```

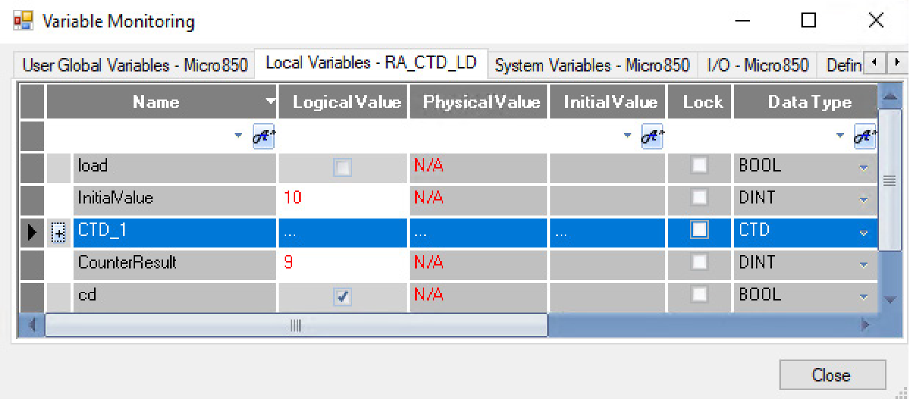
(*ST Equivalence: CTD1 is an instance of block *)

CTD1(trigger,load_cmd,100);

underflow := CTD1.Q;

result := CTD1.CV;

Results



Variable Monitoring

User Global Variables - Micro850 | Local Variables - RA_CTD_LD | System Variables - Micro850 | I/O - Micro850 | Defin

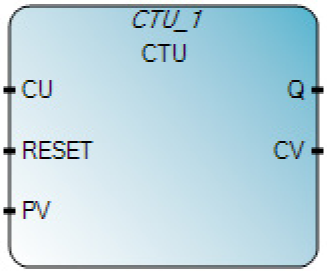
Name	Logical Value	Physical Value	Initial Value	Lock	Data Type
load	□	N/A		□	BOOL
InitialValue	10	N/A		□	DINT
CTD_1	...	...	...	□	CTD
CounterResult	9	N/A		□	DINT
cd	☑	N/A		□	BOOL

CTU (count up)

CTU counts (integers) from 0 up to a given value, 1 by 1.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

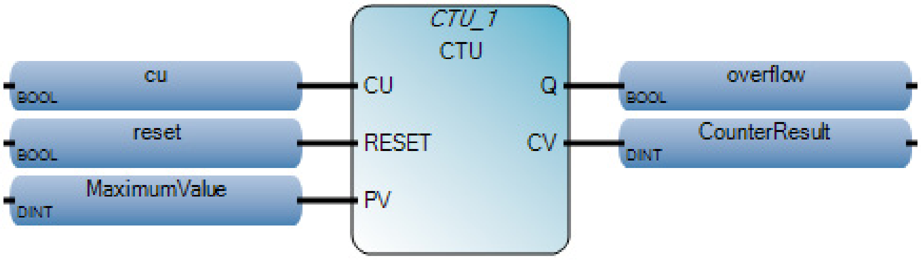
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



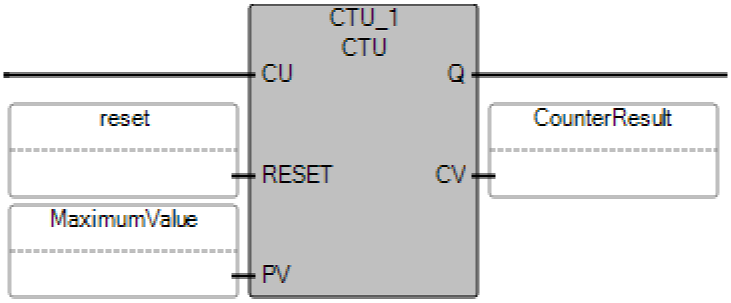
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
CU	Input	BOOL	Counts upward. TRUE - Rising edge detected, count upward in increments of one. FALSE - Falling edge detected, hold the counter value at the same value.
RESET	Input	BOOL	Reset verifies the PV value against the count upward value. TRUE - set the CV value to zero. FALSE - Continue incrementing count upward by one.
PV	Input	DINT	Programmed maximum value of the counter.
Q	Output	BOOL	Indicates whether the count up instruction has resulted in a number greater than or equal to the maximum value of the counter. TRUE - Counter result =>PV (Overflow condition). FALSE - Counter result < PV
CV	Output	DINT	Current counter result.

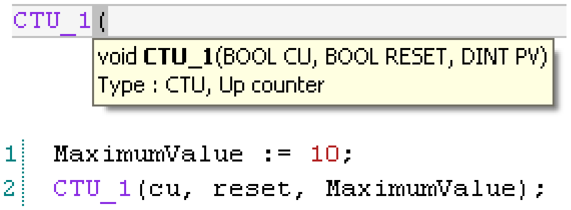
CTU Function Block Diagram example



CTU Ladder Diagram example



CTU Structured Text example



```
CTU_1(
```

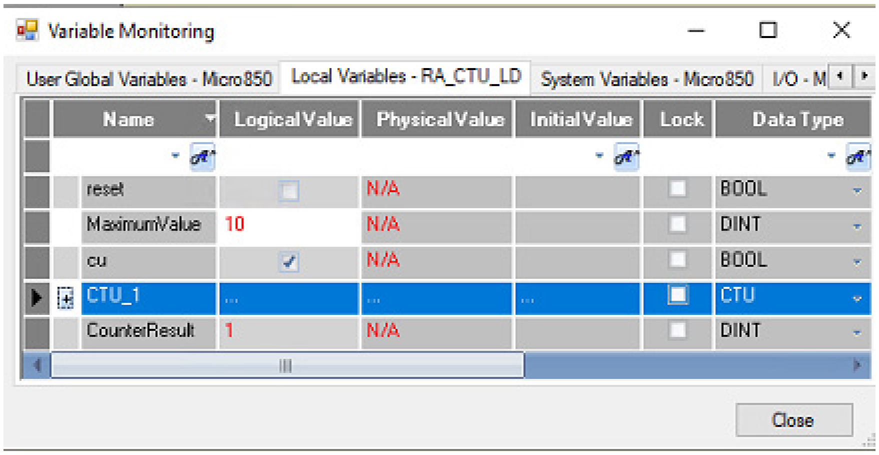
void **CTU_1**(BOOL CU, BOOL RESET, DINT PV)
Type : CTU, Up counter

```
MaximumValue := 10;
CTU_1(cu, reset, MaximumValue);
```

(* ST Equivalence: CTU1 is an instance of CTU block*)

```
CTU1(trigger,NOT(auto_mode),100);
overflow := CTU1.Q;
result := CTU1.CV;
```

Results



Variable Monitoring

User Global Variables - Micro850 | Local Variables - RA_CTU_LD | System Variables - Micro850 | I/O - M

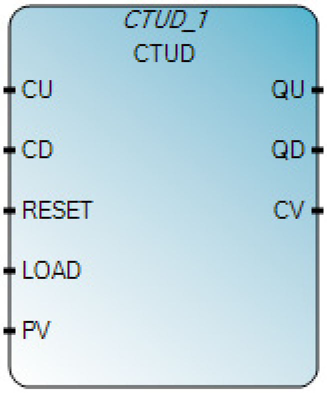
Name	Logical Value	Physical Value	Initial Value	Lock	Data Type
reset	□	N/A		□	BOOL
MaximumValue	10	N/A		□	DINT
cu	☑	N/A		□	BOOL
CTU_1	...	...	...	□	CTU
CounterResult	1	N/A		□	DINT

CTUD (count up down)

Counts integers from 0 up to a given value, 1 by 1, or from a given value down to 0, 1 by 1.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

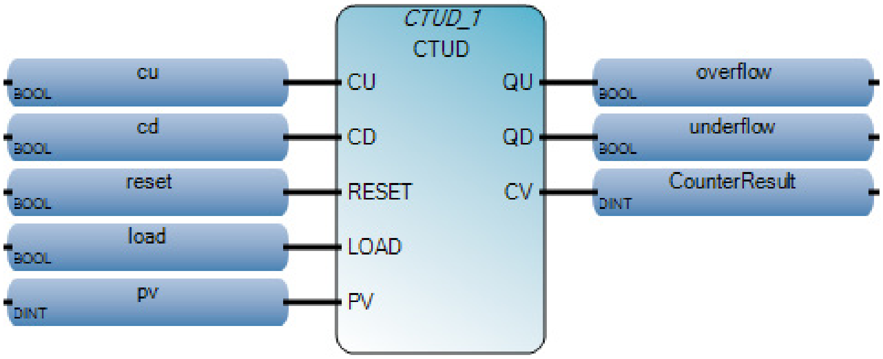
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



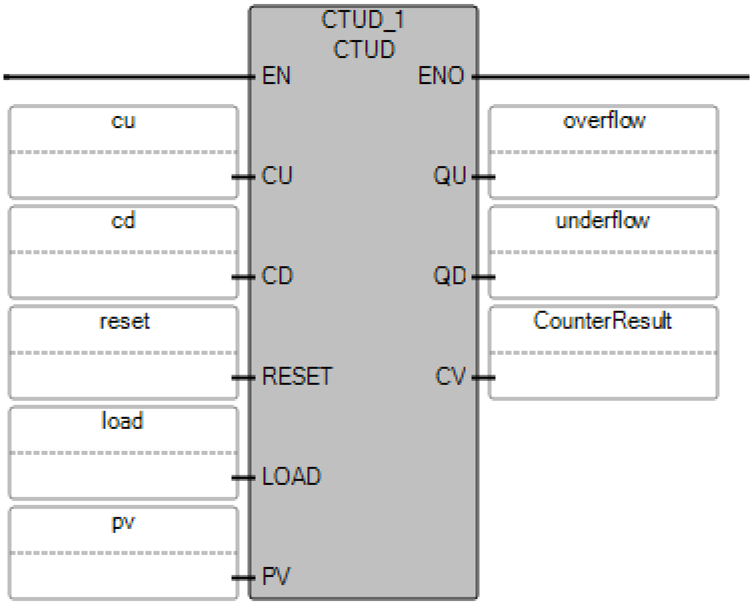
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
CU	Input	BOOL	TRUE - Rising Edge detected, count up.
CD	Input	BOOL	TRUE - Rising Edge detected, count down.
RESET	Input	BOOL	Reset dominant (highest priority when determining instruction behavior) command. (CV = 0 when RESET is TRUE).
LOAD	Input	BOOL	Load command. TRUE - set CV = PV.
PV	Input	DINT	Programmed maximum value.
QU	Output	BOOL	Overflow. TRUE - when CV >= PV.
QD	Output	BOOL	Underflow. TRUE - when CV <= 0.
CV	Output	DINT	Counter result.

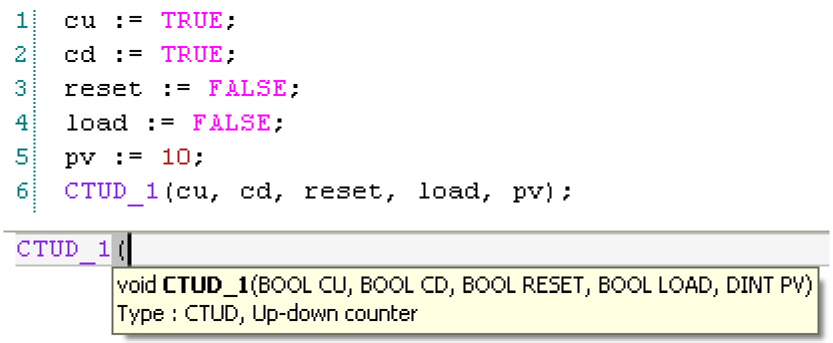
CTUD Function Block Diagram example



CTUD Ladder Diagram example



CTUD Structured Text example



```
cu := TRUE;
cd := TRUE;
reset := FALSE;
load := FALSE;
pv := 10;
CTUD_1(cu, cd, reset, load, pv);
```

```
CTUD_1(
void CTUD_1(BOOL CU, BOOL CD, BOOL RESET, BOOL LOAD, DINT PV)
Type : CTUD, Up-down counter
```

(* ST Equivalence: We suppose CTUD1 is an instance of block*)

```
CTUD1(trigger1, trigger2, reset_cmd, load_cmd,100);
full := CTUD1.QU;
empty := CTUD1.QD;
nb_elt := CTUD1.CV;
```

Data conversion instructions

Use Data conversion instructions to convert the data type of a variable to a different data type.

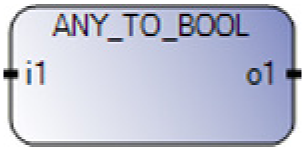
Instruction	Description
ANY_TO_BOOL on page 239	Converts a non-Boolean value to a Boolean.
ANY_TO_BYTE on page 240	Converts a value to a Byte.
ANY_TO_DATE on page 241	Converts a String, Integer, Real, or Time data type to Date data type.
ANY_TO_DINT on page 243	Converts a value to a Double Integer.
ANY_TO_DWORD on page 244	Converts a value to a Double Word value.
ANY_TO_INT on page 245	Converts a value to an Integer.
ANY_TO_LINT on page 246	Converts a value to a Long Integer.
ANY_TO_LREAL on page 247	Converts a value to a Long Real.
ANY_TO_LWORD on page 248	Converts a value to a Long Word.
ANY_TO_REAL on page 249	Converts a value to a Real.
ANY_TO_SINT on page 250	Converts a value to a Short Integer.
ANY_TO_STRING on page 250	Converts a value to a String.
ANY_TO_TIME on page 252	Converts a value to the Time data type.
ANY_TO_UDINT on page 253	Converts a value to an Unsigned Double Integer.
ANY_TO_UINT on page 254	Converts a value to an Unsigned Integer.
ANY_TO_ULINT on page 255	Converts a value to an Unsigned Long Integer.
ANY_TO_USINT on page 256	Converts a value to an Unsigned Short Integer.
ANY_TO_WORD on page 257	Converts a value to a Word.

ANY_TO_BOOL

Converts a non-Boolean value to a Boolean value.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description

EN	Input	BOOL	Instruction enable. TRUE - execute the conversion to Boolean computation. FALSE - there is no computation. Applies to Ladder Diagram programs.
i1	Input	SINT USINT BYTE INT UINT WORD DINT UDINT DWORD LINT ULINT LWORD REAL LREAL TIME DATE STRING	Any non-Boolean value.
o1	Output	BOOL	Boolean value.

ANY_TO_BOOL Structured Text example

(* ST Equivalence: *)

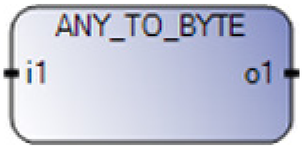
```
ares := ANY_TO_BOOL (10);               (* ares is TRUE *)
tres := ANY_TO_BOOL (t#0s);             (* tres is FALSE *)
mres := ANY_TO_BOOL ('FALSE');          (* mres is FALSE *)
```

ANY_TO_BYTE

Converts a value to an 8-bit Byte value.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute the conversion to Byte computation. FALSE - there is no computation. Applies to Ladder Diagram programs.

i1	Input	BOOL SINT USINT INT UINT WORD DINT UDINT DWORD LINT ULINT LWORD REAL LREAL TIME DATE STRING	Any non-Byte value.
o1	Output	BYTE	An 8-bit Byte value.
ENO	Output	BOOL	Enable out. Applies only to Ladder Diagram programs.

ANY_TO_BYTE Structured Text example

```
(* ST Equivalence: *)

bres := ANY_TO_BYTE (true);                (* bres is 1 *)
tres := ANY_TO_BYTE (t#0s46ms);            (* tres is 46 *)
mres := ANY_TO_BYTE ('0198');              (* mres is 198 *)
```

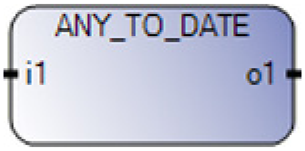
ANY_TO_DATE

Converts a String, Integer, Real, or Time data type to Date data type.

Boolean, Byte, and Word date types are supported but are not recommended.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



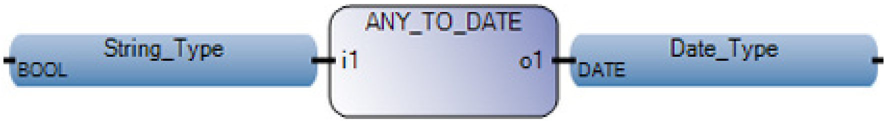
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute the conversion to the Date computation. FALSE - there is no computation. Applies only to Ladder Diagram programs.

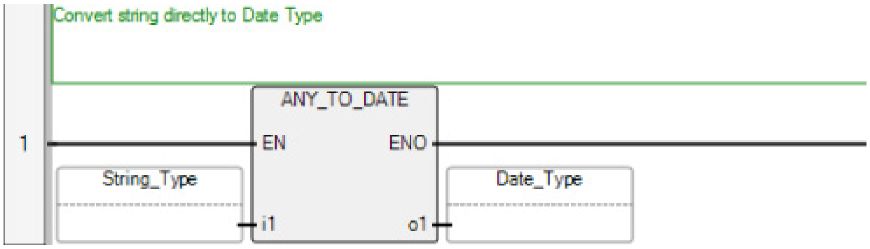
i1	Input	BOOL SINT USINT BYTE INT UINT WORD DINT UDINT DWORD LINT ULINT LWORD REAL LREAL TIME STRING	• Strings are directly converted to DATE data type and must be in the format of YYYY-MM-DD, DATE#YYYY-MM-DD, or D#YYYY-MM-DD. • Integers and Real, which must be positive, are in units of seconds and added to the date 1970-01-01. • Time is added to the date 1970-01-01.
o1	Output	DATE	Converted date value. Errors during conversion (such as a negative date) results in an INVALID result.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.

ANY_TO_DATE Function Block Diagram example

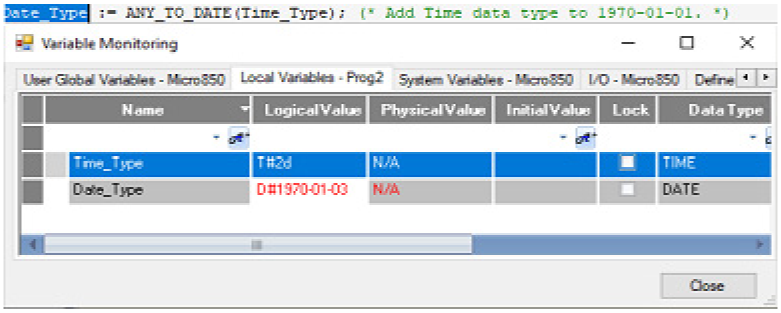
ANY_TO_DATE



ANY_TO_DATE Ladder Diagram example


Convert string directly to Date Type

ANY_TO_DATE Structured Text example



```
Date_Type := ANY_TO_DATE(Time_Type); (* Add Time data type to 1970-01-01. *)
```

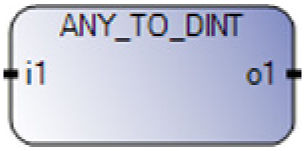
Name	LogicalValue	PhysicalValue	InitialValue	Lock	Data Type
Time_Type	T#2d	N/A		□	TIME
Date_Type	D#1970-01-03	N/A		□	DATE

ANY_TO_DINT

Converts a value to 32-bit Double Integer value.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute the conversion to the 32-bit Double Integer computation. FALSE - there is no computation. Applies only to Ladder Diagram programs.
i1	Input	BOOL SINT USINT BYTE INT UINT WORD UDINT DWORD LINT ULINT LWORD REAL LREAL TIME DATE STRING	Any value other than a Double Integer.
o1	Output	DINT	A 32-bit Double Integer value.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.

ANY_TO_DINT Structured Text example

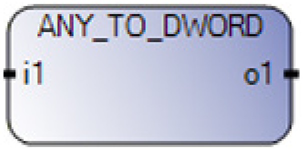
```
(* ST Equivalence: *)

bres := ANY_TO_DINT (true);          (* bres is 1 *)
tres := ANY_TO_DINT (t#1s46ms);      (* tres is 1046 *)
mres := ANY_TO_DINT ('0198');        (* mres is 198 *)
```

ANY_TO_DWORD

Converts a value to a 32-bit double Word value.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute the conversion to the 32-bit double Word computation. FALSE - there is no computation. Applies only to Ladder Diagram programs.
i1	Input	BOOL SINT USINT BYTE INT UINT WORD DINT UDINT LINT ULINT LWORD REAL LREAL TIME DATE STRING	Any value other than a double word.
o1	Output	DWORD	A 32-bit double Word value.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.

ANY_TO_DWORD Structured Text example

(* ST Equivalence: *)

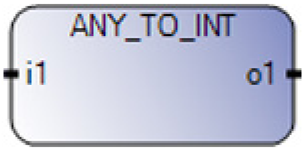
```
bres := ANY_TO_DWORD (true);              (* bres is 1 *)
tres := ANY_TO_DWORD (t#1s46ms);          (* tres is 1046 *)
mres := ANY_TO_DWORD ('0198');            (* mres is 198 *)
```

ANY_TO_INT

Converts a value to a 16-bit Integer value.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute the conversion to the 16-bit Integer computation. FALSE - there is no computation. Applies only to Ladder Diagram programs.
i1	Input	BOOL SINT USINT BYTE UINT WORD DINT UDINT DWORD LINT ULINT LWORD REAL LREAL TIME DATE STRING	Any non-16-bit Integer value.
o1	Output	INT	A 16-bit Integer value.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.

ANY_TO_INT Structured Text example

(* ST Equivalence: *)

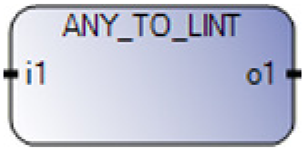
```
bres := ANY_TO_INT (true);                  (* bres is 1 *)
tres := ANY_TO_INT (t#0s46ms);              (* tres is 46 *)
mres := ANY_TO_INT ('0198');                (* mres is 198 *)
```

ANY_TO_LINT

Converts a value to a 64-bit Long Integer value.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute the conversion to the 64-bit Long Integer computation. FALSE - there is no computation. Applies only to Ladder Diagram programs.
i1	Input	BOOL SINT USINT BYTE INT UINT WORD DINT UDINT DWORD LINT ULINT LWORD REAL LREAL TIME DATE STRING	Any value other than a Long Integer.
o1	Output	LINT	A 64-bit Long Integer value.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.

ANY_TO_LINT Structured Text example

(* ST Equivalence: *)

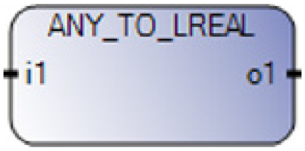
```
bres := ANY_TO_LINT (true);                    (* bres is 1 *)
tres := ANY_TO_LINT (t#0s46ms);                (* tres is 46 *)
mres := ANY_TO_LINT ('0198');                  (* mres is 198 *)
```

ANY_TO_LREAL

Converts any value to a Long Real value.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute the conversion to the long Real computation. FALSE - there is no computation. Applies only to Ladder Diagram programs.
i1	Input	BOOL SINT USINT BYTE INT UINT WORD DINT UDINT DWORD LINT ULINT LWORD REAL TIME DATE STRING	Any value other than a long Real.
o1	Output	LREAL	A long real value.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.

ANY_TO_LREAL Structured Text example

(* ST Equivalence: *)

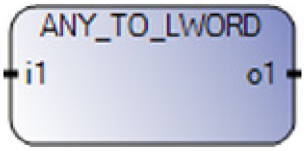
```
bres := ANY_TO_LREAL (true);                    (* bres is 1.0 *)
tres := ANY_TO_LREAL (t#1s46ms);                (* tres is 1046.0 *)
ares := ANY_TO_LREAL (198);                     (* ares is 198.0 *)
```

ANY_TO_LWORD

Converts a value to a 64-bit Long Word value.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute the conversion to the 64-bit Long Word computation. FALSE - there is no computation. Applies only to Ladder Diagram programs.
i1	Input	BOOL SINT USINT BYTE INT UINT WORD DINT UDINT DWORD LINT ULINT REAL LREAL IME DATE STRING	Any value other than a Long Word.
o1	Output	LWORD	A 64-bit Long Word value.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.

ANY_TO_LWORD Structured Text example

(* ST Equivalence: *)

```
bres := ANY_TO_LWORD (true);                    (* bres is 1 *)
tres := ANY_TO_LWORD (t#0s46ms);                (* tres is 46 *)
```

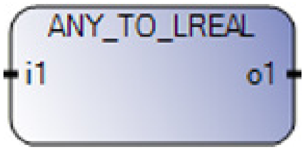
```
mres := ANY_TO_LWORD ('0198');                    (* mres is 198 *)
```

ANY_TO_REAL

Converts a value to a Real value.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute the conversion to the Real computation. FALSE - there is no computation. Applies only to Ladder Diagram programs.
i1	Input	BOOL SINT USINT BYTE INT UINT WORD DINT UDINT DWORD LINT ULINT LWORD LREAL TIME DATE STRING	Any value other than Real.
o1	Output	REAL	A real value.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.

ANY_TO_REAL Structured Text example

(* ST Equivalence: *)

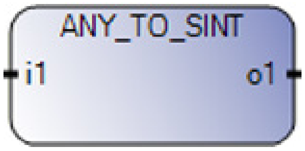
```
bres := ANY_TO_REAL (true);               (* bres is 1.0 *)
tres := ANY_TO_REAL (t#1s46ms);           (* tres is 1046.0 *)
ares := ANY_TO_REAL (198);                (* ares is 198.0 *)
```

ANY_TO_SINT

Converts a value to a Short Integer value.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute the conversion to the 8-bit Short Integer computation. FALSE - there is no computation. Applies only to Ladder Diagram programs.
i1	Input	BOOL USINT BYTE INT UINT WORD DINT UDINT DWORD LINT ULINT LWORD REAL LREAL TIME DATE STRING	Any non-Short Integer value.
o1	Output	SINT	A Short Integer value.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.

ANY_TO_SINT Structured Text example

(* ST Equivalence: *)

```
bres := ANY_TO_SINT (true);            (* bres is 1 *)
tres := ANY_TO_SINT (t#0s46ms);        (* tres is 46 *)
mres := ANY_TO_SINT ('0198');          (* mres is 198 *)
```

ANY_TO_STRING

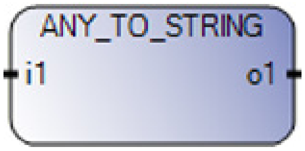
Converts a value to a String value.

Operation details:

- When converting a REAL data type to a String the ANY_TO_STRING instruction uses the IEEE 754 format.
  - ANY_TO_STRING converts 125.0 to 1.25000E+02
- When the target string length is 5 chars:
  - ANY_TO_STRING converts 125.0 to 1.25000E+02 and outputs 1.250 to the target string.
  - Consider creating a user-defined function block to convert from Exponential notation to number.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute the conversion to String computation. FALSE - there is no computation. Applies only to Ladder Diagram programs.
i1	Input	BOOL SINT USINT BYTE INT UINT WORD DINT UDINT DWORD LINT ULINT LWORD REAL LREAL TIME DATE	Any value other than String.

o1	Output	STRING	If IN is a Boolean, 'FALSE' or 'TRUE'.
			If IN is an Integer or a real, a decimal representation.
			If IN is a TIME, can be:
			TIME time1
			STRING s1
			time1 :=13 ms;
			s1:=ANY_TO_STRING(time1);
			(* s1 = '0s13' *).

ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.

ANY_TO_STRING Structured Text example

(* ST Equivalence: *)

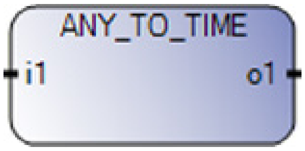
```
bres := ANY_TO_STRING (TRUE);                     (* bres is 'TRUE' *)
ares := ANY_TO_STRING (125);                      (* ares is '125' *)
```

ANY_TO_TIME

Converts a non-Time or non-Date value to a Time value.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute the conversion to the Time computation. FALSE - there is no computation. Applies only to Ladder Diagram programs.

i1	Input	BOOL SINT USINT BYTE INT UINT WORD DINT UDINT DWORD LINT ULINT LWORD REAL LREAL STRING	Any positive value other than a Time or Date data type. IN (or integer part of IN if it is real) is the number of milliseconds. STRING (number of milliseconds, for example, a value of 300032 represents 5 minutes and 32 milliseconds).
o1	Output	TIME	Time value represented by IN. A value of 1193h2m47s295ms indicates an invalid time.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.

ANY_TO_TIME Structured Text example

(* ST Equivalence: *)

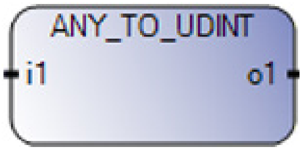
```
ares := ANY_TO_TIME (1256);                    (* ares := t#1s256ms *)
rres := ANY_TO_TIME (1256.3);                  (* rres := t#1s256ms *)
```

ANY_TO_UDINT

Converts a value to a 32-bit Unsigned Double Integer value.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute the conversion to the 32-bit Double Integer computation. FALSE - there is no computation. Applies only to Ladder Diagram programs.

i1	Input	BOOL SINT USINT BYTE INT UINT WORD DINT DWORD LINT ULINT LWORD REAL LREAL TIME DATE STRING	Any value other than an Unsigned Double Integer.
o1	Output	UDINT	A 32-bit Unsigned Double Integer value.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.

ANY_TO_UDINT Structured Text example

(* ST Equivalence: *)

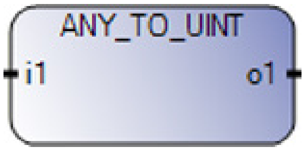
```
bres := ANY_TO_UDINT (true);            (* bres is 1 *)
tres := ANY_TO_UDINT (t#1s46ms);        (* tres is 1046 *)
mres := ANY_TO_UDINT ('0198');          (* mres is 198 *)
```

ANY_TO_UINT

Converts a value to an Unsigned Integer value.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute the conversion to the 16-bit Unsigned Integer computation. FALSE - there is no computation. Applies only to Ladder Diagram programs.

i1	Input	BOOL SINT USINT BYTE INT WORD DINT UDINT DWORD LINT ULINT LWORD REAL LREAL TIME DATE STRING	Any non-Unsigned Integer value.
o1	Output	UINT	An Unsigned Integer value.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.

ANY_TO_UINT Structured Text example

(* ST Equivalence: *)

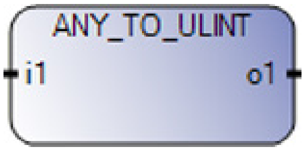
```
bres := ANY_TO_UINT (true);                (* bres is 1 *)
tres := ANY_TO_UINT (t#0s46ms);            (* tres is 46 *)
mres := ANY_TO_UINT ('0198');              (* mres is 198 *)
```

ANY_TO_ULINT

Converts a value to a 64-bit Unsigned Long Integer value.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute the conversion to the 64-bit Unsigned Long Integer computation. FALSE - there is no computation. Applies only to Ladder Diagram programs.

i1	Input	BOOL SINT USINT BYTE INT UINT WORD DINT UDINT DWORD LINT LWORD REAL LREAL TIME DATE STRING	Any value other than an Unsigned Long Integer.
o1	Output	ULINT	A 64-bit Unsigned Long Integer value.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.

ANY_TO_ULINT Structured Text example

(* ST Equivalence: *)

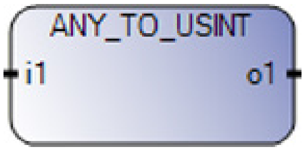
```
bres := ANY_TO_ULINT (true);                    (* bres is 1 *)
tres := ANY_TO_ULINT (t#0s46ms);                (* tres is 46 *)
mres := ANY_TO_ULINT ('0198');                  (* mres is 198 *)
```

ANY_TO_USINT

Converts a value to an Unsigned Short Integer value.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute the conversion to the 8-bit Unsigned Short Integer computation. FALSE - there is no computation. Applies only to Ladder Diagram programs.

i1	Input	BOOL SINT BYTE INT UINT WORD DINT UDINT DWORD LINT ULINT LWORD REAL LREAL TIME DATE STRING	Any non-Short Integer value.
o1	Output	USINT	An Unsigned Short Integer value.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.

ANY_TO_USINT Structured Text example

(* ST Equivalence: *)

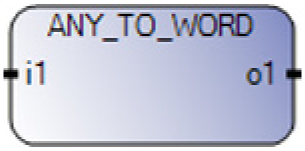
```
bres := ANY_TO_USINT (true);                    (* bres is 1 *)
tres := ANY_TO_USINT (t#0s46ms);                (* tres is 46 *)
mres := ANY_TO_USINT ('0198');                  (* mres is 198 *)
```

ANY_TO_WORD

Converts a value to a 16-bit Word value.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute the conversion to the 16-bit Word computation. FALSE - there is no computation. Applies only to Ladder Diagram programs.

i1	Input	BOOL SINT USINT BYTE INT DINT UDINT DWORD LINT ULINT LWORD REAL LREAL TIME DATE STRING	Any value other than a WORD value.
o1	Output	WORD	A Word value.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.

ANY_TO_WORD Structured Text example

(* ST Equivalence: *)

```
bres := ANY_TO_WORD (true);                    (* bres is 1 *)
tres := ANY_TO_WORD (t#0s46ms);                (* tres is 46 *)
mres := ANY_TO_WORD ('0198');                  (* mres is 198 *)
```

Data manipulation instructions

Use Data manipulation instructions to alter the output data to change the status without altering the program.

Instruction	Description
AVERAGE on page 259	Calculates a running average over a number of a defined samples.
COP on page 260	Copies the binary data in the source element to the destination element.
MAX on page 268	Calculates the maximum of two integer values.
MIN on page 266	Calculates the minimum of two integer values.

AVERAGE

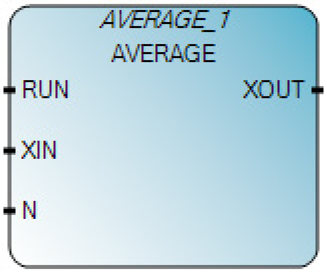
Calculates a running average over a number of a defined samples and stores the value at each cycle.

Operation details:

- The defined number of samples (N) cannot exceed 127.
- When setting or changing the value for N, set RUN to FALSE, then set it back to TRUE.
- If the RUN command is FALSE (reset mode), the output value is equal to the input value.
- When the maximum number of stored values is reached, the first stored value is erased by the last one.
- Using floating-point data types could result in inaccurate calculations due to the rounding limitations inherent in floating-point mathematics.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



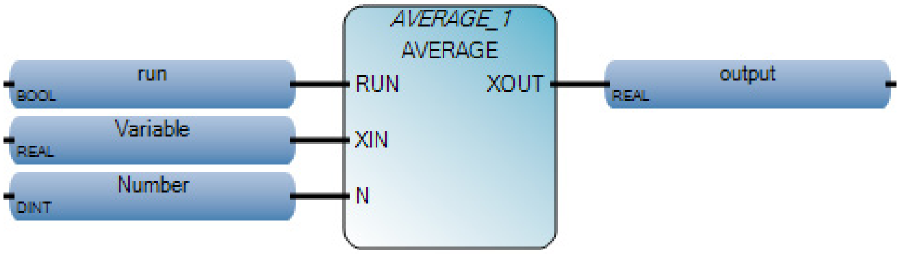
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description

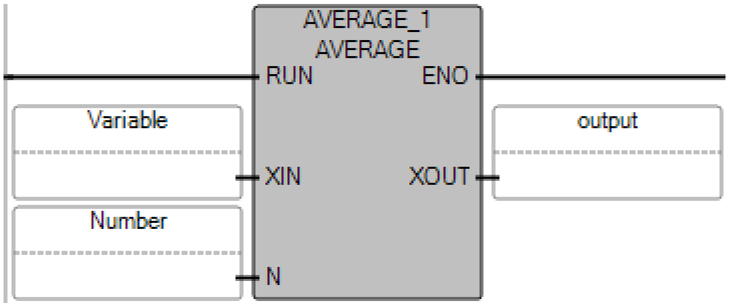
RUN	Input	BOOL	TRUE = run
			FALSE = reset

XIN	Input	REAL	Any real variable.
N	Input	DINT	Application defined number of samples.
XOUT	Output	REAL	Running average of XIN value.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.

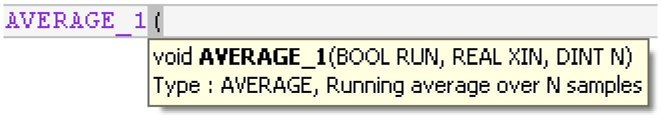
AVERAGE Function Block Diagram example



AVERAGE Ladder Diagram example

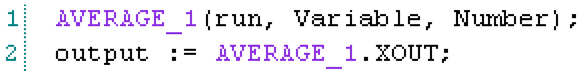


AVERAGE Structured Text example



```
AVERAGE_1(
```

void **AVERAGE_1**(BOOL RUN, REAL XIN, DINT N)
Type : AVERAGE, Running average over N samples



```
AVERAGE_1(run, Variable, Number);
output := AVERAGE_1.XOUT;
```

(* ST Equivalence: AVERAGE1 an instance of an AVERAGE block *)

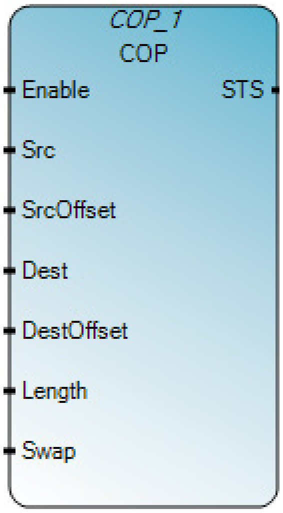
```
AVERAGE1((auto_mode & store_cmd), sensor_value, 100);
ave_value := AVERAGE1.XOUT;
```

COP

Copies the binary data in the source element to the destination element. The source element remains unchanged.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type		Description
Enable	Input	BOOL		Instruction block enable. COP iis level triggered. TRUE - perform copy. FALSE - the function block is idle.
Src	Input	BOOL SINT USINT BYTE INT UINT WORD DINT UDINT	DWORD REAL TIME DATE STRING LWORD ULINT LINT LREAL	Initial element to copy. If the source is a STRING data type the destination must be a either a STRING data type or a USINT (UCHAR and BYTE) data type. If it is not, a data type mismatch is reported.
SrcOffset	Input	UINT		The source element offset is used with array data types to identify the position in the source array to copy the data from. Set the offset to 0 if: • If it is not an array data type, or • To copy from the first element for an array data type.
Dest	Input	BOOL SINT USINT BYTE INT UINT WORD DINT UDINT	DWORD REAL TIME DATE STRING LWORD ULINT LINT	Initial element to be overwritten by the source. If the destination is a STRING data type the source must be either a STRING data type or a USINT (UCHAR and BYTE) data type. If it is not, a data type mismatch is reported.

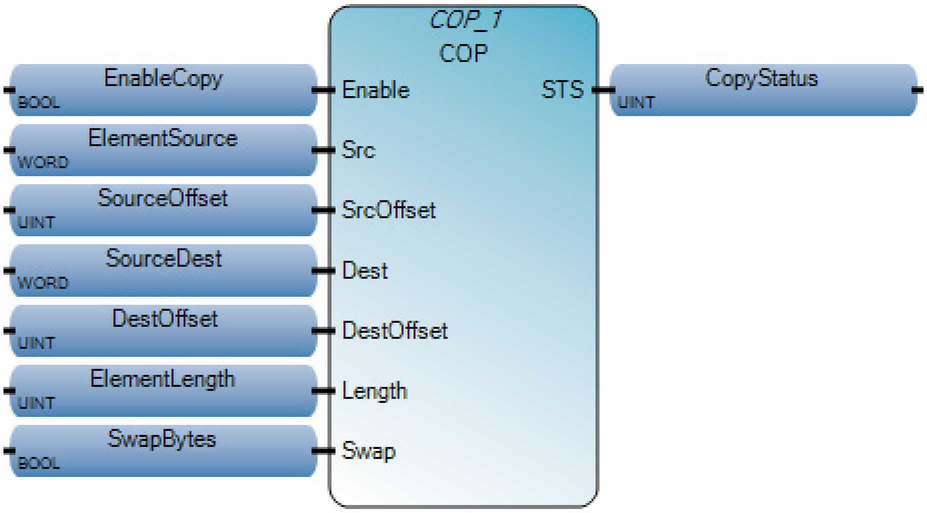
DestOffset	Input	UINT	The destination element offset is used with array data types to identify the position in the destination array to copy the data to. Set the offset to 0 if: • If it is not an array data type, or • To copy from the first element for an array data type.
Length	Input	UINT	Number of destination elements to copy. When the destination is a STRING data type, it indicates the number of strings to be copied.
Swap	Input	BOOL	Used to exchange the data from the source and destination elements, so that the destination data replaces the source data and the source data replaces the destination data. TRUE - Swap bytes according to the data type. A swap operation does not occur if: • The source data type or the destination data type is a STRING, or • If both the source and the destination are 1-byte length data.
Sts	Output	UINT	Status of the copy operation. The definitions for the Sts parameter are defined in COP status codes.
ENO	Output	BOOL	Enables output. Applies only to Ladder Diagram programs.

COP status codes (Sts)

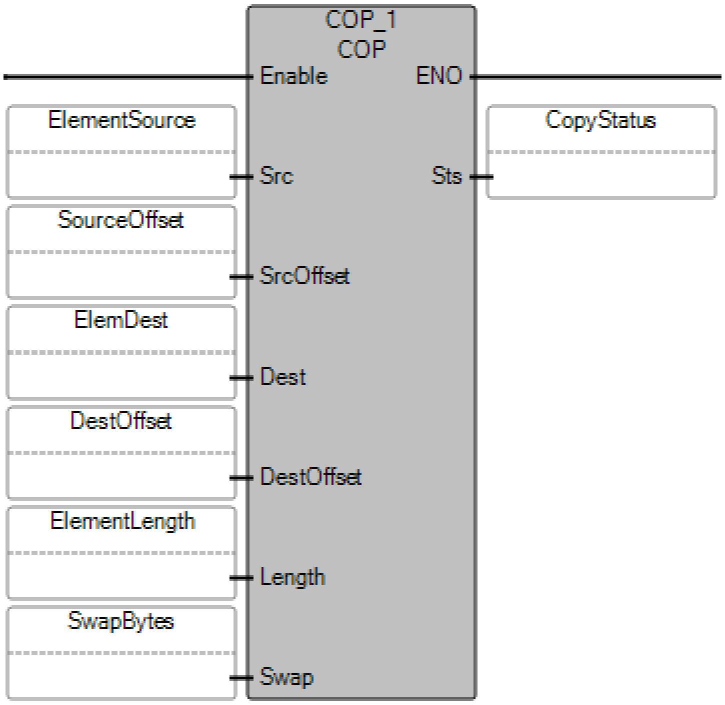
The following table describes the COP status codes.

COP Status code	Status description
0x00	No action taken (not enabled).
0x01	COP function block success.
0x02	Destination has spare bytes when copying from String.
0x03	Source data are truncated.
0x04	Copy length is invalid.
0x05	Data type mismatch when there is String Data type as either source or destination.
0x06	Source data size is too small for copy.
0x07	Destination data size is too small for copy.
0x08	Source Data offset is invalid.
0x09	Destination Data offset is invalid.
0x0A	Data is invalid in either source or destination.

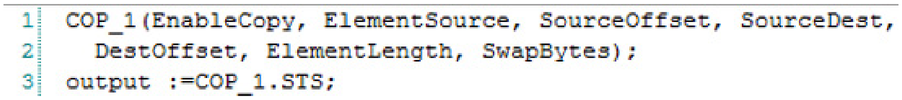
COP Function Block Diagram example



COP Ladder Diagram example



COP Structured Text example



```
COP_1(EnableCopy, ElementSource, SourceOffset, SourceDest,
  DestOffset, ElementLength, SwapBytes);
output :=COP_1.STS;
```

Copy to a different data type

When a copy to or from a String data type is performed, the ODVA short String format is used for data in the USINT array.

When COP on page 260 is used between any other pair of data types, the copy operation is valid, even if the data type in the source is not the same as the data type in the destination, and even when they are not in a valid format. The logic must be validated at the application level.

From a USINT array to a String array

To copy a USINT array to a String array, the data in the USINT array must be in this format:

- Byte1: Length of first String
- Byte2: First Byte Character
- Byte3: Second Byte Character
- Byte n: Last Byte Character
- Byte (n+1): Length of second String
- Byte (n+2): First Byte Character for second String

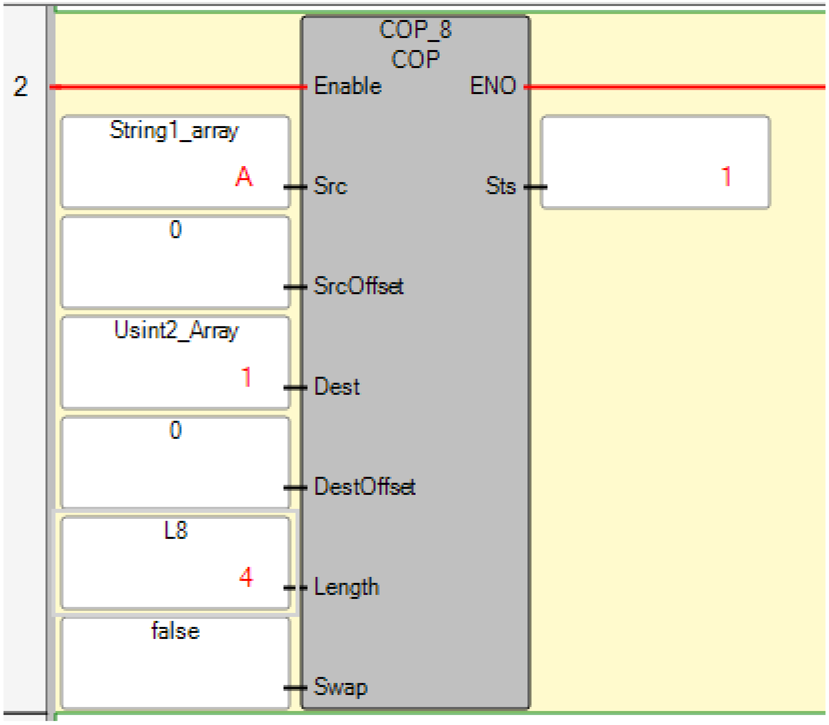
COP string array example

The following example shows a COP on page 260 instruction copying a string array into Usint array. The COP instruction skips all the elements in the source array with Zero length.

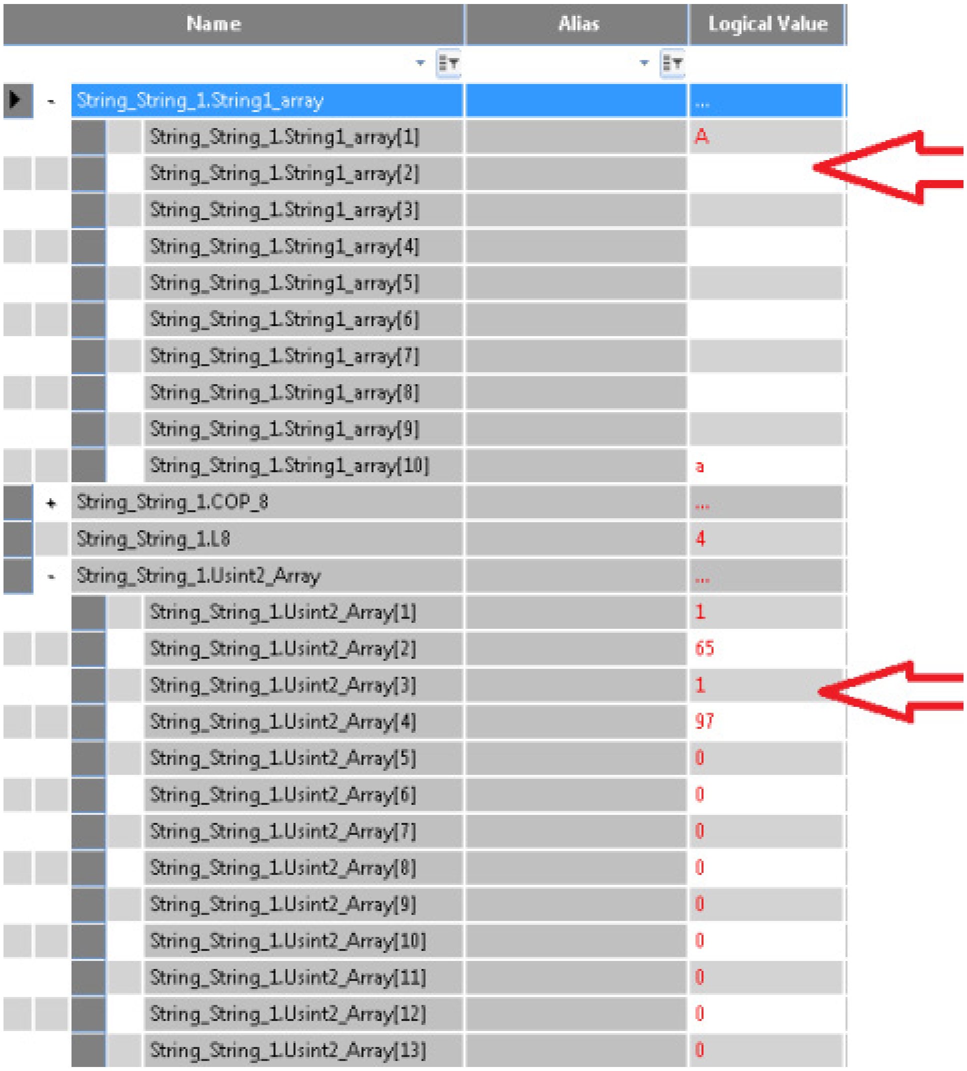
For this example:

- Length is specified as 4 bytes.
- Number of Destination Elements to copy is 4 bytes.
- All the array elements with Zero length (blanks) are skipped.
- The COP instruction finds a non zero length element in the string array[1], which is copied into the designation Usint array[1] with 1. One is the length of the string in String array[1]) and Usint array[2] is 65, which is the ASCII code of "A".
- The COP instruction finds a non zero length element in String array[10] which is "a", which is copied into destination Usint array[3] with 1. One is the length of string in String array[10]) and Usint array[4] is 97 which is ASCII code of "a".

Ladder Diagram


2
COP_8
COP
Enable
ENO
String1_array
A
Src
Sts
1
0
SrcOffset
Usint2_Array
1
Dest
0
DestOffset
L8
4
Length
false
Swap

Array elements with Logical Values



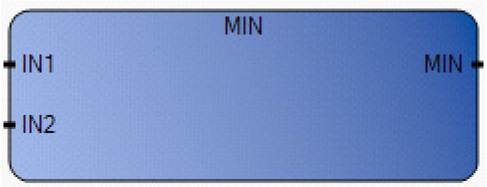
Name	Alias	Logical Value
String_String_1.String1_array		...
String_String_1.String1_array[1]		A
String_String_1.String1_array[2]		
String_String_1.String1_array[3]		
String_String_1.String1_array[4]		
String_String_1.String1_array[5]		
String_String_1.String1_array[6]		
String_String_1.String1_array[7]		
String_String_1.String1_array[8]		
String_String_1.String1_array[9]		
String_String_1.String1_array[10]		a
String_String_1.COP_8		...
String_String_1.L8		4
String_String_1.Usint2_Array		...
String_String_1.Usint2_Array[1]		1
String_String_1.Usint2_Array[2]		65
String_String_1.Usint2_Array[3]		1
String_String_1.Usint2_Array[4]		97
String_String_1.Usint2_Array[5]		0
String_String_1.Usint2_Array[6]		0
String_String_1.Usint2_Array[7]		0
String_String_1.Usint2_Array[8]		0
String_String_1.Usint2_Array[9]		0
String_String_1.Usint2_Array[10]		0
String_String_1.Usint2_Array[11]		0
String_String_1.Usint2_Array[12]		0
String_String_1.Usint2_Array[13]		0

MIN (minimum)

Calculates the minimum of two integer values.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.

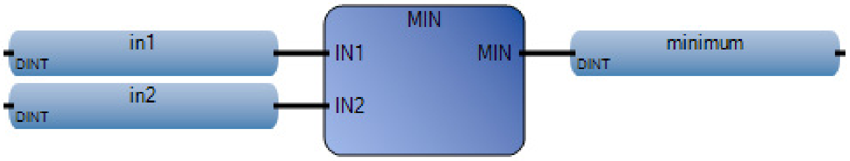


Use this table to help determine the parameter values for this instruction.

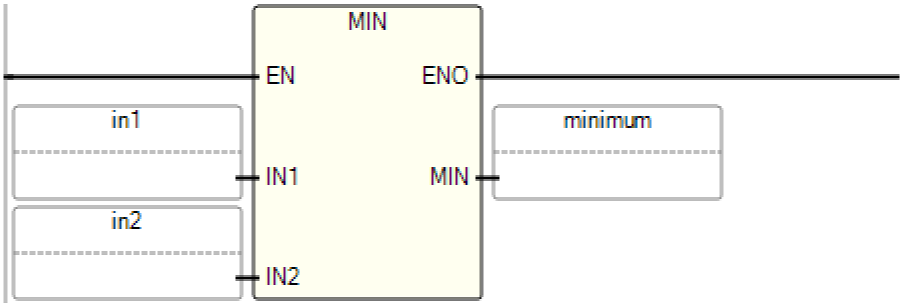
Parameter	Parameter Type	Data Type	Description

EN	Input	BOOL	Instruction enable. TRUE - execute the minimum integer value computation FALSE - there is no computation. Applies to Ladder Diagram programs.
IN1	Input	DINT	Any signed integer value.
IN2	Input	DINT	Cannot be Real.
MIN	Output	DINT	Minimum of both input values.
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

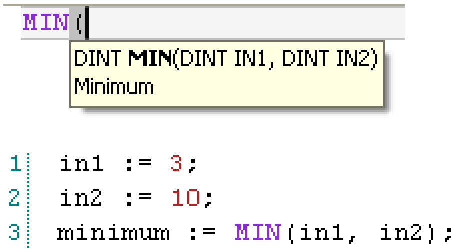
MIN Function Block Diagram example



MIN Ladder Diagram example



MIN Structured Text example



```
MIN(
DINT MIN(DINT IN1, DINT IN2)
Minimum
```

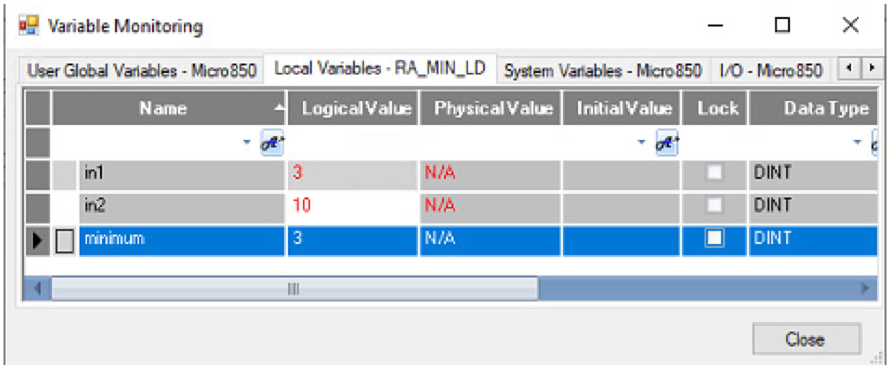
```
in1 := 3;
in2 := 10;
minimum := MIN(in1, in2);
```

(* ST Equivalence: *)

new_value := MAX (MIN (max_value, value), min_value);

(* bounds the value to the [min_value..max_value] set *)

Results


Variable Monitoring
User Global Variables - Micro850
Local Variables - RA_MIN_LD
System Variables - Micro850
I/O - Micro850
Close

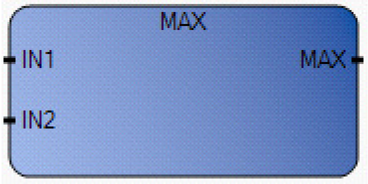
Name	Logical Value	Physical Value	Initial Value	Lock	Data Type
in1	3	N/A		□	DINT
in2	10	N/A		□	DINT
minimum	3	N/A		□	DINT

MAX (maximum)

Calculates the maximum of two integer values.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

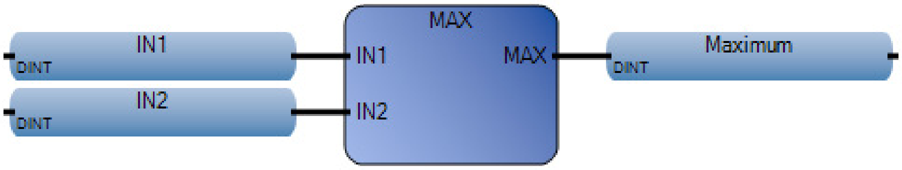
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



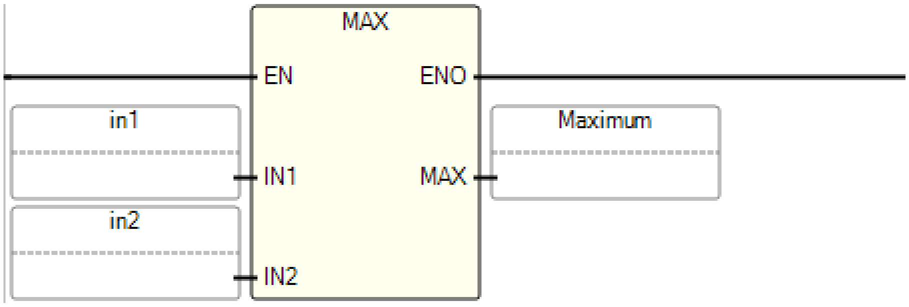
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - execute maximum integer value computation. FALSE - there is no computation. Applies to Ladder Diagram programs.
IN1	Input	DINT	Any signed integer value.
IN2	Input	DINT	Cannot be Real.
MAX	Output	DINT	Maximum of both input values.
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

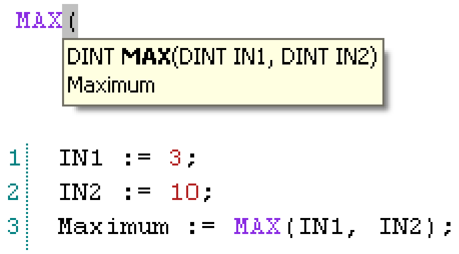
MAX Function Block Diagram example



MAX Ladder Diagram example



MAX Structured Text example



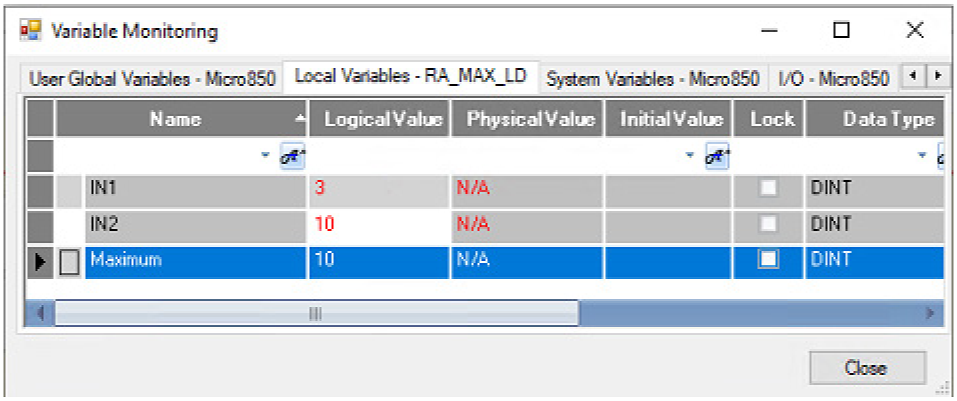
```
IN1 := 3;
IN2 := 10;
Maximum := MAX(IN1, IN2);
```

(* ST Equivalence: *)

new_value := MAX (MIN (max_value, value), min_value);

(* bounds the value to the [min_value..max_value] set *)

Results


Variable Monitoring
User Global Variables - Micro850
Local Variables - RA_MAX_LD
System Variables - Micro850
I/O - Micro850
Close

Name	Logical Value	Physical Value	Initial Value	Lock	Data Type
IN1	3	N/A		□	DINT
IN2	10	N/A		□	DINT
Maximum	10	N/A		□	DINT

High-Speed Counter (HSC) instructions

Use High-speed counter instructions to monitor and control the high-speed counter.

Instruction	Description
HSC on page 272	HSC applies high presets, low presets and output source values to the high-speed counter.
HSC_SET_STS on page 287	HSC_SET_STS manually sets or resets the HSC counting status.

What is a High-Speed Counter?

A high-speed counter detects and counts narrow (fast) pulses and then issues specialized instructions to initiate control operations when the detected counts reach their preset values. Control operations include the automatic and immediate execution of the high-speed counter interrupt routine and the immediate update of outputs based on the configured source and mask pattern.

High-speed counter capabilities

Because HSC instructions have high-performance requirements, their operation is performed by custom circuitry that runs in parallel with the main system processor. Enhanced capabilities of High-Speed Counters (HSC) include:

- 100 kHz operation high-speed direct control of outputs
- 32-bit signed integer data (count range of ± 2,147,483,647)
- Programmable high and low presets
- Overflow and underflow setpoint
- Automatic interrupt processing based on accumulated count
- Run-time editable parameters (from the user control program) HSC instruction operation

Micro800 controller support for HSC

All Micro830, Micro850 and Micro870 controllers, except for 2080-LCxx-AWB, support up to six HSC inputs. HSC functionality is implemented in Micro800 controllers using high-speed counter hardware (embedded inputs in the controller), and the HSC instruction in the application. The HSC instruction on page 271 configures the high-speed counter hardware and updates the image accumulator.

IMPORTANT The HSC function can only be used with the controller's embedded I/O. It cannot be used with expansion I/O modules.

HSC (high-speed counter)

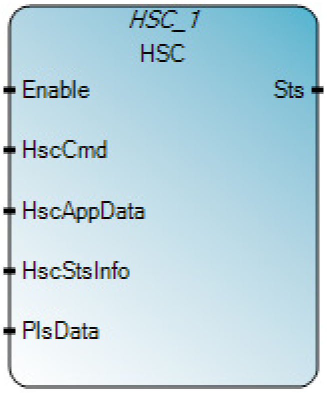
HSC applies high presets, low presets and output source values to the high-speed counter.

Operation details:

- Programmable Limit Switch (PLS) is enabled by setting the HSCAppData.PLSEnable parameter to True.
- The PLSPosition parameter is reset after a full cycle completes and the HSCSTS.HP value is reached.

This instructions applies to the Micro830, Micro850, and Micro870 controllers.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

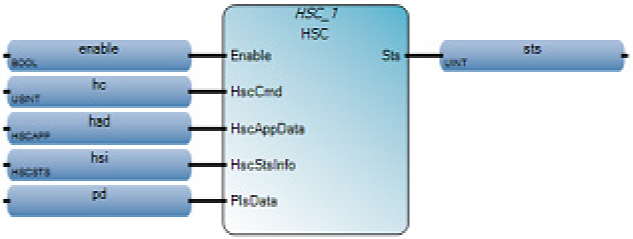


Use this table to help determine the parameter values for this instruction.

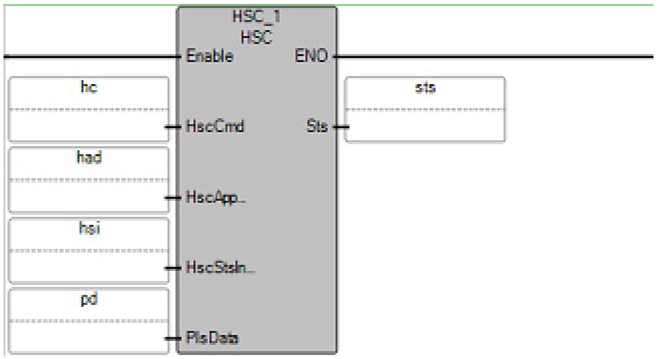
Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction block rung state. TRUE - the timer starts incrementing. FALSE - the function block is idle. The recommendation is not to use the EN parameter with the HSC function block because when EN is set to FALSE the timer continues to increment. Applies only to Ladder Diagram programs.
Enable	Input	BOOL	Enable instruction block. TRUE - execute the HSC operation specified in the HSC command parameter. FALSE - no HSC commands are issued.
HscCmd	Input	USINT	Issues commands to the HSC.
HSCAppData	Input	HSCAPP on page 274	HSC application configuration, which is usually only needed once.
HSCStsInfo	Input	HSCSTS on page 278	HSC dynamic status, which is continuously updated during HSC counting.

PlsData	Input	DINT UDINT	Programmable Limit Switch (PLS) data structure.
Sts	Output	UINT	HSC execution status. HSC status codes: • 0x00 - No action taken (not enabled). • 0x01 - HSC execution successful. • 0x02 - HSC command invalid. • 0x03 - HSC ID out of range. • 0x04 - HSC configure error.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.

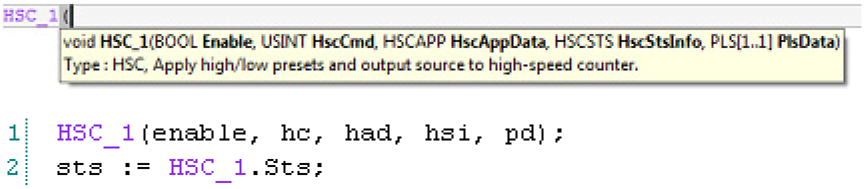
HSC Function Block Diagram example



HSC Ladder Diagram example



HSC Structured Text example



HSC_1(

void HSC_1(BOOL Enable, USINT HscCmd, HSCAPP HscAppData, HSCSTS HscStsInfo, PLS[1..1] PlsData)
Type : HSC, Apply high/low presets and output source to high-speed counter.

```
HSC_1(enable, hc, had, hsi, pd);
sts := HSC_1.Sts;
```

HSCCmd values

The following table describes the HSC commands for each HSC command value.

HSC command	Command description

0x01	HSC RUN • Start HSC (if HSC is in Idle mode and the rung is enabled). • Update HSC Status Information only (if HSC in Run mode and the rung is enabled).
0x02	HSC Stop: Stop a HSC counting (if HSC is in Run mode and the rung is enabled).
0x03	HSC Load/Set: reload the HSC Application Data (if rung is enabled) for 6 input elements: HPSetting, LPSetting, HPOutput, LPOutput, OFSetting, and UFSetting. **Note:** This command does not re-load the following input element: HSC accumulator.
0x04	HSC Accumulator Reset (if rung is enabled).

HSC command results

Command value	Result	Conditions
HscCmd =1	Starts the HSC mechanism, and the HSC transitions to running mode.	Setting the Enable input parameter to False does not stop counting while in running mode. HscCmd =2 must be issued to stop counting.
	The HSC mechanism automatically updates values.	HSC AppData.Accumalator is updated with HSC Sts.Accumulator
HscCmd =4 (reset)	Sets the HSC Acc value to the HSC AppData.Accumalator value.	HscCmd =4 does not stop HSC counting. If HSC is counting when HscCmd =4 is issued, some counting may be lost To set a specific value to HSC Acc while counting, write the value to HSC AppData.Accumalator immediately before HscCmd =4 is issued.

HSCAPP data type

Use the HSCAPP data type to define the HSCAppData parameter in HSC instruction. The HSCApp data type parameters are used to define HSC configuration data.

Use this table to help determine the parameter values for the HSCAPP data type.

Parameter	Data Type	Data format	User program access	Description
PLSEnable	BOOL	bit	read/write	Enable or disable the High-Speed Counter Programmable Limit Switch (PLS).
HSCID	UINT	word	read/write	Defines the HSC.
HSCMode	UINT	word	read/write	Defines the HSC mode.
Accumulator	DINT	long word	read/write	Initial accumulator value. HSCApp.Accumulator sets the initial accumulator value when the High-Speed Counter starts. When the HSC is in Counting mode, the Accumulator is automatically updated by the HSC sub-system to reflect the actual HSC accumulator value.
HPSetting	DINT	long word	read/write	High preset setting. The HSCApp.HPSetting parameter sets the upper setpoint (in counts) that defines when the HSC sub-system generates an interrupt. The data loaded into the high preset must be less than or equal to the data resident in the overflow (HSCAPP.OFSetting) parameter or an HSC error is generated.

LPSetting	DINT	long word	read/write	Low preset setting. HSCApp.LPSetting sets the lower setpoint (in counts) that defines when the HSC sub-system generates an interrupt. The data loaded into the low preset must be greater than or equal to the data resident in the underflow (HSCAPP.UFSetting) parameter or an HSC error is generated. If the underflow and low preset values are negative numbers, the low preset must be a number with an absolute value smaller than the underflow.
OFSetting	DINT	long word	read/write	Overflow setting. The HSCApp.OFSetting overflow setting defines the upper count limit for the counter. If the counter's accumulated value increments above the value specified in OFSetting, an overflow interrupt is generated. When the overflow interrupt is generated, the HSC sub-system resets the accumulator value to the underflow value and the counter continues counting from the underflow value (counts are not lost in this transition). OFSetting values must be: • Between -2,147,483,648 and 2,147,483,647. • Greater than the underflow value. • Greater than or equal to the data resident in the high preset (HSCAPP.HPSetting) or an HSC error is generated.
UFSetting	DINT	long word	read/write	Underflow setting. The HSCApp.UFSetting underflow setting that defines the lower count limit for the counter. If the counter's accumulated value decrements below the value specified in UFSetting, an underflow interrupt is generated. When the underflow interrupt is generated, the HSC sub-system resets the accumulated value to the overflow value and the counter starts counting from the overflow value (counts are not lost in the transition). UFSetting values must be: • Between -2,147,483,648 and 2,147,483,647. • Less than the overflow value. • Less than or equal to the data resident in the low preset (HSCAPP.LPSetting) or an HSC error is generated.
OutputMask	UDINT	word	read/write	Out mask for output. The HSCApp.OutputMask defines the embedded outputs on the controller that the High-Speed Counter can directly control. The HSC sub-system can, without control program interaction, turn outputs ON or OFF based on the High or Low presets of the HSC accumulator. The bit pattern stored in HSCApp.OutputMask defines which outputs are controlled by the HSC and which outputs are not controlled by the HSC. The HSCAPP.OutputMask bit pattern corresponds to the output bits on the controller and can only be configured during initial setup. Bits that are set (1) are enabled and can be turned on or off by the HSC sub-system. Bits that are set (0) cannot be turned on or off by the HSC sub-system. For example, to use the HSC to control outputs 0, 1, 3, assign: • HscAppData.OutputMask = 2#1011, or • HscAppData.OutputMask = 11

HPOutput	UDINT	long word	read/write	32-bit output setting for High preset reaching. HSCApp.HPOutput defines the state (1 = ON or 0 = OFF) of the outputs on the controller when the high preset is reached. For more information on how to directly turn outputs on or off based on the high preset. Configure the high output bit pattern during initial setup, or you can use the HSC function block to load the new parameters while the controller is operating.
LPOutput	UDINT	long word	read/write	32-bit output setting for Low preset reaching. HSCApp.LPOutput defines the state (1 = "on", 0 = "off ") of the outputs on the controller when the low preset is reached. For more information on how to directly turn outputs on or off based on the low preset. Configure the low output bit pattern during initial setup, or you can use the HSC function block to load the new parameters while the controller is operating.

HSCApp settings versus PLSData settings

When the PLS function is enabled, relevant HSCApp settings are superseded by the corresponding PLSData settings as shown in the following table.

HSCApp setting	PLSData setting
HSCAPP.HpSetting	HSCHP
HSCAPP.LpSetting	HSCLP
HSCAPP.HPOutput	HSCHPOutput
HSCAPP.LPOutput	HSCLPOutput

HSCApp.HSCID

The HSCApp.HSCID parameter identifies the High-Speed Counter.

The following table lists the values for the HSCID:

Output Selection	Bit	Description
First word of HSC Function Data	15-13	Module type of HSC: • 0x00 - Embedded. • 0x01 - Expansion. • 0x02 - Plug-in Port.
	12-8	Slot ID of the module: • 0x00 - Embedded. • 0x01-0x1F - ID of Expansion Module. • 0x01-0x05 - ID of Plug-in Port.
	7-0	HSC ID inside the module: • 0x00-0x0F - Embedded. • 0x00-0x07 - ID of HSC for Expansion. • 0x00-0x07 - ID of HSC for Plug-in Port. For the initial version of Connected Components Workbench, only IDs 0x00-0x05 are supported.

HSCApp.HSCMode

The HSCApp.HSCMode parameter sets the High-Speed Counter to one of 10 types of counting modes. The mode value is configured through the programming device and is accessible in the control program.

For additional information on HSC operating modes and input assignments, see HSC Inputs and Wiring Mapping in the *Micro830 and Micro850 Programmable Controllers User Manual*.

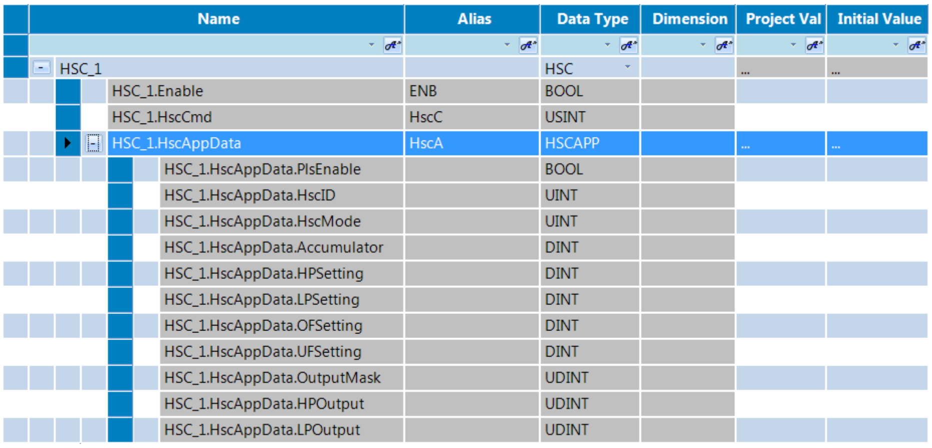
HSC operating modes, the main HSC and sub HSC support different modes.

- The main high-speed counters support 10 types of operation modes.
- Sub high-speed counters support 5 types of operation modes (mode 0, 2, 4, 6, 8).
- If the main high-speed counter is set to mode 1, 3, 5, 7 or 9, then the sub high-speed counter is disabled.

HSCMode	Counting mode
0	Up counter. The accumulator is immediately cleared (0) when it reaches the high preset. A low preset cannot be defined in this mode.
1	Up counter with external reset and hold. The accumulator is immediately cleared (0) when it reaches the high preset. A low preset cannot be defined in this mode.
2	Counter with external direction.
3	Counter with external direction, reset and hold.
4	Two input counter (up and down).
5	Two input counter (up and down) with external reset and hold.
6	Quadrature counter (phased inputs A and B).
7	Quadrature counter (phased inputs A and B) with external reset and hold.
8	Quadrature X4 counter (phased inputs A and B).
9	Quadrature X4 counter (phased inputs A and B) with external reset and hold.

HSCAppData parameters example

The following image shows the HSCAppData parameters in the **Variable Selector**.



Name	Alias	Data Type	Dimension	Project Val	Initial Value
HSC_1		HSC		...	...
HSC_1.Enable	ENB	BOOL			
HSC_1.HscCmd	HscC	USINT			
HSC_1.HscAppData	HscA	HSCAPP		...	...
HSC_1.HscAppData.PlsEnable		BOOL			
HSC_1.HscAppData.HscID		UINT			
HSC_1.HscAppData.HscMode		UINT			
HSC_1.HscAppData.Accumulator		DINT			
HSC_1.HscAppData.HPSetting		DINT			
HSC_1.HscAppData.LPSetting		DINT			
HSC_1.HscAppData.OFSetting		DINT			
HSC_1.HscAppData.UFSetting		DINT			
HSC_1.HscAppData.OutputMask		UDINT			
HSC_1.HscAppData.HPOutput		UDINT			
HSC_1.HscAppData.LPOutput		UDINT			

HSCSTS data type

HSCSTSInfo (data type HSCSTS) displays the status of the High-Speed Counter.

HSCSTSInfo status actions

During HSC counting, the following HSC status actions occur.

- If the HSC function block is counting with command 0x01, the HSC status is continuously updated.
- If an error occurs, the Error_Detected flag is turned on and an error code is set.

HSCSTSInfo parameters

Parameter	Data type	HSC mode	User program access	Description
CountEnable	BOOL	0...9	read only	Counting enabled.
ErrorDetected	BOOL	0...9	read/write	Non-zero means error detected.
CountUpFlag	BOOL	0...9	read only	Count up flag.
CountDwnFlag	BOOL	2...9	read only	Count down flag.
Mode1Done	BOOL	0 or 1	read/write	HSC is Mode 1A or Mode 1B; accumulator counts up to the HP value.
OVF	BOOL	0...9	read/write	Overflow is detected.
UNF	BOOL	0...9	read/write	Underflow is detected.
CountDir	BOOL	0...9	read only	1: count up; 0: count down.
HPReached	BOOL	2...9	read/write	High preset reached.
LPReached	BOOL	2...9	read/write	Low preset reached.
OFCauseInter	BOOL	0...9	read/write	Overflow caused a HSC interrupt.
UFCauseInter	BOOL	2...9	read/write	Underflow caused a HSC. interrupt.
HPCauseInter	BOOL	0...9	read/write	High preset reached, causing a HSC interrupt.
LPCauseInter	BOOL	2...9	read/write	Low Preset reached, causing a HSC interrupt.
PlsPosition	UINT	0...9	read only	Position of the Programmable Limit Switch (PLS). The PLSPosition parameter is reset after completing a full cycle and reaching the HP value.
ErrorCode	UINT	0...9	read/write	Displays the error codes detected by the HSC sub-system.
Accumulator	DINT		read/write	Actual accumulator reading.
HP	DINT		read only	Last high preset setting.
LP	DINT		read only	Last low preset setting.
HPOutput	UDINT		read/write	Last high preset output setting.
LPOutput	UDINT		read/write	Last low preset output setting.

HSCSTSInfo parameter details

HSCSTSInfo (data type HSCSTS) parameters are used to determine the status of the High-Speed Counter.

CountEnable

Parameter	Data type	HSC mode	User program access
HSCSTS.CountEnable	BOOL	0...9	read only

Indicates the status of the High-Speed Counter, whether counting is enabled (1) or disabled (0, default).

ErrorDetected

Parameter	Data type	HSC mode	User program access
HSCSTS.ErrorDetected	BOOL	0...9	read/write

Detects if an error is present in the HSC sub-system. Configuration errors are the most common types of error represented by the ErrorDetectedr. When the bit is set (1), look at the specific error code in parameter HSCSTS.ErrorCode, which is maintained by the controller. You can clear the ErrorDetected bit when necessary.

CountUpFlag

Parameter	Data type	HSC mode	User program access
HSCSTS.CountUpFlag	BOOL	0...9	read only

Used with all of the HSCs (modes 0...9). If the HSCSTS.CountEnable bit is set, the Count Up bit is set (1). If the HSCSTS.CountEnable is cleared, the Count Up bit is cleared (0).

CountDownFlag

Parameter	Data type	HSC mode	User program access
HSCSTS.CountDownFlag	BOOL	2...9	read only

Used with the bidirectional counters (modes 2...9). If the HSCSTS.CountEnable bit is set, the Count Down bit is set (1). If the HSCSTS.CountEnable bit is clear, the Count Down bit is cleared (0).

Mode1Done

Parameter	Data type	HSC mode	User program access
HSCSTS.Mode1Done	BOOL	0 or 1	read/write

The HSC sub-system sets the HSCSTS.Mode1Done status flag to (1) when the HSC is configured for Mode 0 or Mode 1 behavior, and the accumulator counts up to the High Preset value.

OVF

Parameter	Data type	HSC mode	User program access
HSCSTS.OVF	BOOL	0...9	read/write

The HSC sub-system sets the HSCSTS.OVF status flag to (1) whenever the accumulated value (HSCSTS.Accumulator) has counted through the overflow variable (HSCAPP.OFSetting). This bit is transitional and is set by the HSC sub-system. It is up to the control program to use, track, and clear (0) the overflow condition.

Overflow conditions do not generate a controller fault.

UNF

Parameter	Data type	HSC mode	User program access
HSCSTS.UNF	BOOL	0...9	read/write

The HSC sub-system sets the HSCSTS.UNF status flag to (1) whenever the accumulated value (HSCSTS.Accumulator) has counted through the underflow variable (HSCAPP.UFSetting). This bit is transitional and is set by the HSC sub-system. It is up to the control program to use, track, and clear (0) the underflow condition.

Underflow conditions do not generate a controller fault.

CountDir

Parameter	Data type	HSC mode	User program access
HSCSTS.CountDir	BOOL	0...9	read only

The HSC sub-system controls the HSCSTS.CountDir status flag. When the HSC accumulator counts up, the direction flag is set to (1). Whenever the HSC accumulator counts down, the direction flag is cleared (0).

If the accumulated value stops, the direction bit retains its value. The only time the direction flag changes is when the accumulated count reverses.

This bit is updated continuously by the HSC sub-system whenever the controller is in a run mode.

HPReached

Parameter	Data type	HSC mode	User program access
HSCSTS.HPReached	BOOL	2...9	read/write

The HSC sub-system sets the HSCSTS.HPReached status flag to (1) whenever the accumulated value (HSCSTS.Accumulator) is greater than or equal to the high preset variable (HSCAPP.HPSetting).

This bit is updated continuously by the HSC sub-system whenever the controller is in an executing mode. Writing to this element is not recommended.

LPReached

Parameter	Data type	HSC mode	User program access
HSCSTS.LPReached	BOOL	2...9	read only

The HSC sub-system sets the HSCSTS.LPReached status flag to (1) whenever the accumulated value (HSCSTS.Accumulator) is less than or equal to the low preset variable (HSCAPP.LPSetting).

This bit is updated continuously by the HSC sub-system whenever the controller is in an executing mode. Writing to this element is not recommended.

OFCauseInter

Parameter	Data type	HSC mode	User program access
HSCSTS.OFCauseInter	BOOL	0...9	read/write

The Overflow Interrupt status bit sets (1) when the HSC accumulator counts through the overflow value and the HSC interrupt is triggered. This bit can be used in the control program to identify that the overflow variable caused the HSC interrupt. If the control program needs to perform any specific control action based on the overflow, this bit is used as conditional logic.

This bit can be cleared (0) by the control program and is also cleared by the HSC sub-system whenever these conditions are detected:

- Low Preset Interrupt executes
- High Preset Interrupt executes
- Underflow Interrupt executes

UFCauseInter

Parameter	Data type	HSC mode	User program access
HSCSTS.UFCauseInter	BOOL	2...9	read/write

The Underflow Interrupt status bit sets (1) when the HSC accumulator counts through the underflow value and the HSC interrupt is triggered. This bit can be used in the control program to identify that the underflow condition caused the HSC interrupt. If the control program needs to perform any specific control action based on the underflow, this bit is used as conditional logic.

This bit can be cleared (0) by the control program and is also cleared by the HSC sub-system whenever these conditions are detected:

- Low Preset Interrupt occurs
- High Preset Interrupt occurs
- Overflow Interrupt occurs

HPCauseInter

Parameter	Data type	HSC mode	User program access
HSCSTS.HPCauseInter	BOOL	0...9	read/write

The High Preset Interrupt status bit sets (1) when the HSC accumulator reaches the high preset value and the HSC interrupt is triggered. This bit can be used in the control program to identify that the high preset condition caused the HSC interrupt. If the control program needs to perform any specific control action based on the high preset, this bit is used as conditional logic.

This bit can be cleared (0) by the control program and is also cleared by the HSC sub-system whenever these conditions are detected:

- Low Preset Interrupt occurs
- Underflow Interrupt occurs
- Overflow Interrupt occurs

LPCauseInter

Parameter	Data type	HSC mode	User program access
HSCSTS.LPCauseInter	BOOL	2...9	read/write

The Low Preset Interrupt status bit sets (1) when the HSC accumulator reaches the low preset value and the HSC interrupt is triggered. This bit can be used in the control program to identify that the low preset condition caused the HSC interrupt. If the control program needs to perform any specific control action based on the low preset, this bit is used as conditional logic.

This bit can be cleared (0) by the control program and is also cleared by the HSC sub-system whenever these conditions are detected:

- High Preset Interrupt occurs
- Underflow Interrupt occurs
- Overflow Interrupt occurs

PlsPosition

Parameter	Data type	HSC mode	User program access
HSCSTS.PLSPosition	UINT	0...9	read only

When the HSC is in Counting mode, and PLS is enabled, this parameter indicates which PLS element is used for the current HSC configuration.

ErrorCode

Parameter	Data type	HSC mode	User program access
HSCSTS.ErrorCode	BOOL	0...9	read only

Displays the error codes detected by the HSC sub-system.

Error code sub-element	HSC counting error code	User program access
Bit 15-8 (high byte)	0-255	The non-zero value for the high byte indicates that the HSC error is due to the PLS data setting. The value of the high byte indicates which element of the PLS data triggers the error.
Bit 7-0 (low byte)	0x00	No error occurring.
	0x01	Invalid HSC counting mode.
	0x02	Invalid high preset.
	0x03	Invalid overflow.
	0x04	Invalid underflow.
	0x05	No PLS data.

Accumulator

Parameter	Data type	User program access
HSCApp.Accumulator	DINT	read/write

Sets the initial accumulator value when the High-Speed Counter starts. When the HSC is in Counting mode, the Accumulator is automatically updated by the HSC sub-system to reflect the actual HSC accumulator value.

HP

Parameter	Data type	User program access
HSCSTS.HP	DINT	read only

The HSCSTS.HP is the upper setpoint (in counts) that defines when the HSC sub-system generates an interrupt.

The data loaded into the high preset must be less than or equal to the data resident in the overflow (HSCAPP.OFSetting) parameter or an HSC error is generated.

This is the latest high preset setting, which may be updated by PLS function from the PLS data block.

LP

Parameter	Data type	HSC mode	User program access
HSCSTS.LP	DINT		read only

The HSCSTS.LP is the lower setpoint (in counts) that defines when the HSC sub-system generates an interrupt.

The data loaded into the low preset must be greater than or equal to the data resident in the underflow (HSCAPP.UFSetting) parameter or an HSC error is generated. If the underflow and low preset values are negative numbers, the low preset must be a number with a smaller absolute value.

This is the latest low preset setting, which may be updated by PLS function from the PLS data block.

HPOutput

Parameter	Data type	User program access
HSCApp.HPOutput	UDINT	read/write

Defines the state (1 = ON or 0 = OFF) of the outputs on the controller when the high preset is reached. For more information on how to directly turn outputs on or off based on the high preset.

You can configure the high output bit pattern during initial setup, or you can use the HSC function block to load the new parameters while the controller is operating.

LPOutput

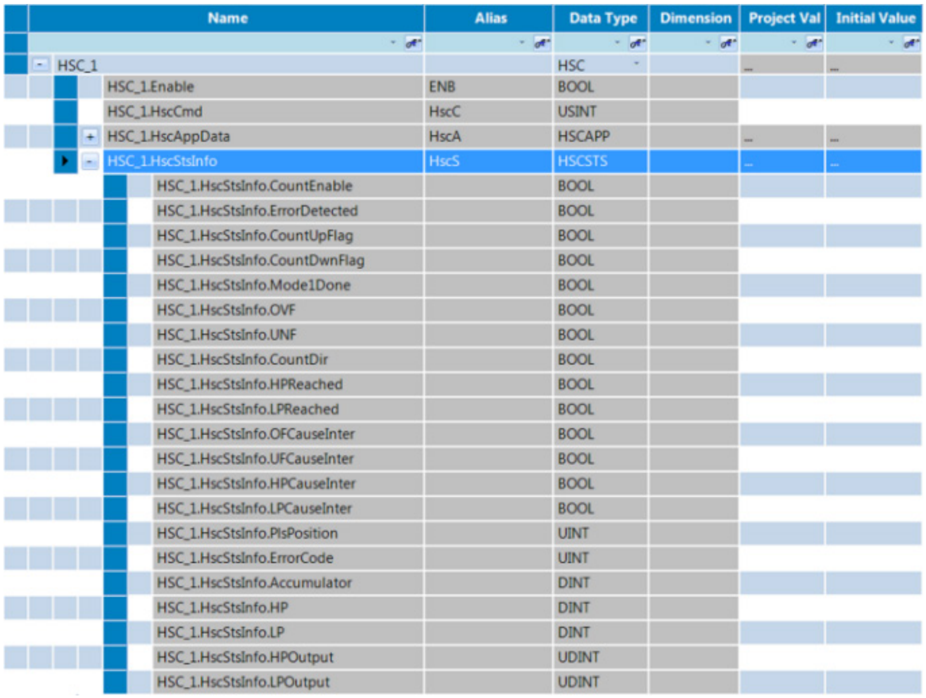
Parameter	Data type	User program access
HSCApp.LPOutput	UDINT	read/write

LPOutput (HSCApp.LPOutput) defines the state (1 = "on", 0 = "off ") of the outputs on the controller when the low preset is reached. For more information on how to directly turn outputs on or off based on the low preset.

You can configure the low output bit pattern during initial setup, or you can use the HSC function block to load the new parameters while the controller is operating.

HSCSTSInfo parameters example

The following image shows the HSCStsInfo parameters in the **Variable Selector**.



Name	Alias	Data Type	Dimension	Project Val	Initial Value
HSC_1		HSC		...	...
HSC_1.Enable	ENB	BOOL			
HSC_1.HscCmd	HscC	USINT			
HSC_1.HscAppData	HscA	HSCAPP		...	...
HSC_1.HscStsInfo	HscS	HSCSTS		...	...
HSC_1.HscStsInfo.CountEnable		BOOL			
HSC_1.HscStsInfo.ErrorDetected		BOOL			
HSC_1.HscStsInfo.CountUpFlag		BOOL			
HSC_1.HscStsInfo.CountDwnFlag		BOOL			
HSC_1.HscStsInfo.Mode1Done		BOOL			
HSC_1.HscStsInfo.OVF		BOOL			
HSC_1.HscStsInfo.UNF		BOOL			
HSC_1.HscStsInfo.CountDir		BOOL			
HSC_1.HscStsInfo.HPReached		BOOL			
HSC_1.HscStsInfo.LPReached		BOOL			
HSC_1.HscStsInfo.OFCauseInter		BOOL			
HSC_1.HscStsInfo.UFCauseInter		BOOL			
HSC_1.HscStsInfo.HPCauseInter		BOOL			
HSC_1.HscStsInfo.LPCauseInter		BOOL			
HSC_1.HscStsInfo.PlsPosition		UINT			
HSC_1.HscStsInfo.ErrorCode		UINT			
HSC_1.HscStsInfo.Accumulator		DINT			
HSC_1.HscStsInfo.HP		DINT			
HSC_1.HscStsInfo.LP		DINT			
HSC_1.HscStsInfo.HPOutput		UDINT			
HSC_1.HscStsInfo.LPOutput		UDINT			

PLS data type

PLSData (data type PLS) is used to configure the programmable limit switch on page 294.

PLSData structure elements

The PLS data structure is a flexible array with the following elements.

Element	Element order	Data Type	Element description
HSCHP	Word 0...1	DINT	High preset
HSCLP	Word 2...3	DINT	Low preset
HSCHPOutput	Word 4...5	UDINT	Output high data
HSCLPOutput	Word 6...7	UDINT	Output low data

The total number of elements for one PLS data structure should not exceed 255.

PLSData parameters

The following table lists the PLSData parameter details.

Element	Data Type	Data Format	HSC mode	User program access	Description
HSCHP	DINT	32-bit signed integer	0	read/writer	High preset
HSCLP	DINT	32-bit signed integer	0	read/writer	Low preset
HSCHPOutput	UDINT	32-bit binary	0	read/writer	Output high data
HSCLPOutput	UDINT	32-bit binary	0	read/writer	Output low data

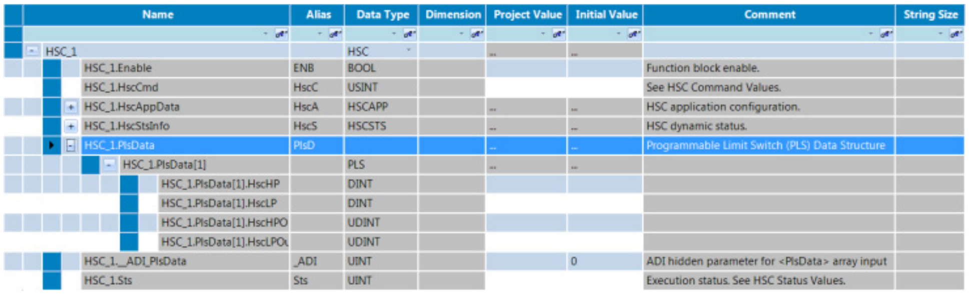
HSCApp settings versus PLSData settings

When the PLS function is enabled, relevant HSCApp settings are superseded by the corresponding PLSData settings as shown in the following table.

HSCApp setting	PLSData setting
HSCAPP.HpSetting	HSCHP
HSCAPP.LpSetting	HSCLP
HSCAPP.HPOutput	HSCHPOutput
HSCAPP.LPOutput	HSCLPOutput

PLSData parameters example

The following figure shows the PLSData parameters in the **Variable Selector**.



Name	Alias	Data Type	Dimension	Project Value	Initial Value	Comment	String Size
HSC_1		HSC		...	...		
HSC_1.Enable	ENB	BOOL				Function block enable.	
HSC_1.HscCmd	HscC	USINT				See HSC Command Values.	
HSC_1.HscAppData	HscA	HSCAPP		...	...	HSC application configuration.	
HSC_1.HscStsInfo	HscS	HSCSTS		...	...	HSC dynamic status.	
HSC_1.PlsData	PlsD			...	...	Programmable Limit Switch (PLS) Data Structure	
HSC_1.PlsData[1]		PLS		...	...		
HSC_1.PlsData[1].HscHP		DINT					
HSC_1.PlsData[1].HscLP		DINT					
HSC_1.PlsData[1].HscHPO		UDINT					
HSC_1.PlsData[1].HscLPOu		UDINT					
HSC_1._ADI_PlsData	_ADI	UINT			0	ADI hidden parameter for <PlsData> array input	
HSC_1.Sts	Sts	UINT				Execution status. See HSC Status Values.	

HSCE_CHANNEL data type

This describes the HSCE_CHANNEL data type:

```
@typedef struct struct_HSCE_channel
{

USINT    ModuleType;
USINT    SlotID;
USINT    HSCID;
} HSCE_CHANNEL;
```

Below is the description:

Byte	Description
Module Type	0x00: Embedded 0x01: Expansion (Sliced) 0x02: Universal Port

Byte	Description
Slot ID	0x00: Embedded 0x01-0x1F: ID of Expansion (Sliced) Module 0x01-0x05: ID of Universal Port
HSCID	0x00-0x0F: Embedded 0x00-0x07: ID of HSC on page 272 for Expansion 0x00-0x01: ID of HSC for Universal-Port

HSCE_STS data type

Below is the description of HSCE_STS data type:

Parameter	Data type	HSC mode	Description
CountEnable	BOOL	0...13	Counting enabled.
ErrorDetected	BOOL	0...13	Non-zero means error detected.
CountUpFlag	BOOL	0...13	Count up flag.
CountDwnFlag	BOOL	2...13	Count down flag.
Mode1Done	BOOL	0 or 1	HSC is Mode 1A or Mode 1B; accumulator counts up to the HP value.
OVF	BOOL	0...13	Overflow is detected.
UNF	BOOL	0...13	Underflow is detected.
CountDir	BOOL	0...13	1: count up; 0: count down.
HPReached	BOOL	2...13	High preset reached.
LPReached	BOOL	2...13	Low preset reached.
OFCauseInter	BOOL	0...13	Overflow caused a HSC on page 272 interrupt.
UFCauseInter	BOOL	2...13	Underflow caused a HSC interrupt.
HPCauseInter	BOOL	0...13	High preset reached, causing a HSC interrupt.
LPCauseInter	BOOL	2...13	Low Preset reached, causing a HSC interrupt.
StateInfo	USINT		HSCE counter state machine information
PlsPosition	UINT	0...9	Position of the Programmable Limit Switch (PLS). The PLSPosition parameter is reset after completing a full cycle and reaching the HP value.
ErrorCode	UINT	0...13	Displays the error codes detected by the HSC sub-system.
Accumulator	DINT		Actual accumulator reading.
HP	DINT		Last high preset setting.
LP	DINT		Last low preset setting.
HPOutput	UDINT		Last high preset output setting.
LPOutput	UDINT		Last low preset output setting.

PLS_HSCE data type

The Programmable Limit Switch function is an additional set of operating modes for the High Speed Counter. When operating in these modes, the preset and output data values are updated using user supplied data each time one of the presets is reached. These modes are programmed by providing a PLS file that contains the data sets to be used. PLS_HSCE data structure is a flexible array with each element defined as following:

Element	Data Type	Element description
HighPreset	LINT	High preset setting
LowPreset	LINT	Low preset setting
HiPresetOutput	UDINT	High preset output data
LoPresetOutput	UDINT	Low preset output data

```
// structure for PLS element
typedef struct PLS_HSCE_EleStruct
{
   LINT HighPreset;      // HSC high preset value
   LINT LowPreset;       // HSC low preset value
   UDINT HiPresetOutput; // HSC high preset output
   UDINT LoPresetOutput; // HSC low preset output
} PLS_HSCE_EleStruct;
```

The total number of elements for one PLS_HSCE data shall be not bigger than 24 for HSC plug-in.

HSC_SET_STS (high-speed counter set status)

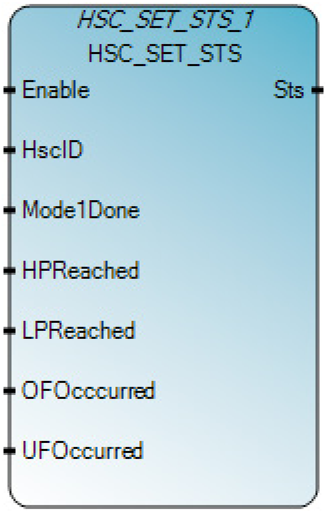
HSC_SET_STS manually sets or resets the HSC counting status.

Operation details:

- The HSC function block must be stopped (not counting) for the HSC_SET_STC function block to set or reset the HTS status. If HSC function is not stopped, the input parameters continue to update and changes made using HSC_SET_STS are ignored.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instructions applies to the Micro830, Micro850, and Micro870 controllers.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
Enable	Input	BOOL	Instruction block enable. TRUE - set/reset the HSC status. FALSE - there is no HSC status change.
HscID	Input	UINT	Manually sets ore resets the HSC status.
Mode1Done	Input	BOOL	Mode 1A or 1B counting is done. This bit can be set or reset when HSC is not counting.
HPReached	Input	BOOL	High preset reached. This bit can be set or reset when HSC is not counting.

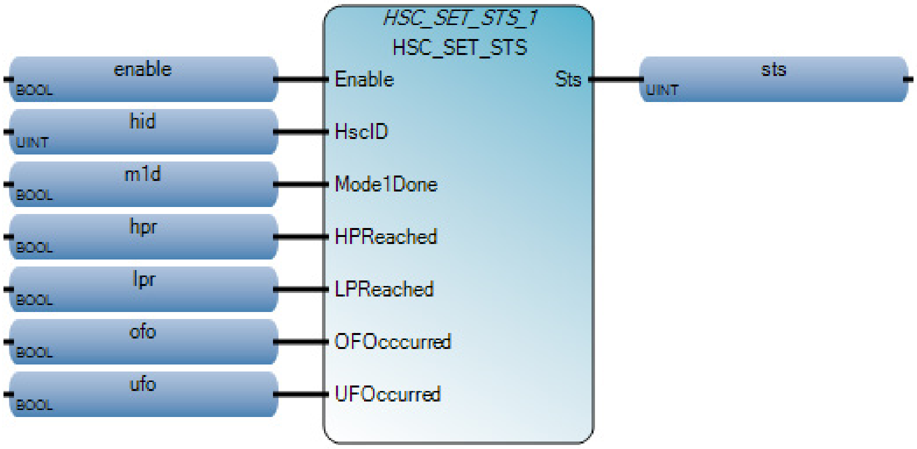
LPReached	Input	BOOL	Low preset reached. This bit can be set or reset when HSC is not counting.
OFOccurred	Input	BOOL	Overflow occurred. This bit can be set or reset when HSC is not counting.
UFOccurred	Input	BOOL	Underflow occurred. This bit can be set or reset when HSC is not counting.
Sts	Output	UINT	Status codes are defined in HSC status codes (Sts).
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.

HSC status codes (Sts)

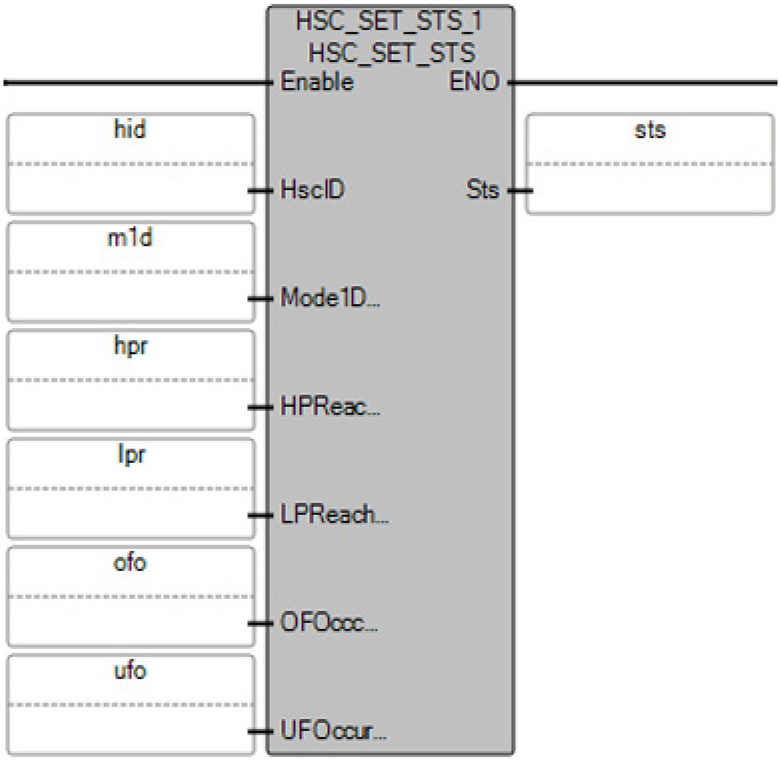
The following table describes the status codes for the HSC function block.

Status code	Status description
0x00	No action taken (not enabled).
0x01	HSC execution successful.
0x02	HSC command invalid.
0x03	HSC ID out of range.
0x04	HSC configure error.

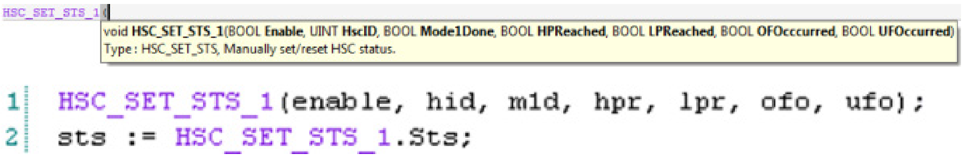
HSC_SET_STS Function Block Diagram example



HSC_SET_STS Ladder Diagram example



HSC_SET_STS Structured Text example



```
HSC_SET_STS_1(
```

void HSC_SET_STS_1(BOOL Enable, UINT HscID, BOOL Mode1Done, BOOL HPReached, BOOL LPReached, BOOL OFOcccurred, BOOL UFOccurred)
Type : HSC_SET_STS, Manually set/reset HSC status.

```
HSC_SET_STS_1(enable, hid, m1d, hpr, lpr, ofo, ufo);
sts := HSC_SET_STS_1.Sts;
```

Use the High-Speed Counter instructions

This section provides specific details and examples for using high-speed counter instructions in logic programs, including the following:

Update HSC application data on page 289

Configure a High-Speed Counter (HSC) user interrupt on page 290

Configure a Programmable Limit Switch (PLS) on page 294

Example: How to create a High-Speed Counter(HSC) on page 295

Example: How to add a Programmable Limit Switch (PLS) function on page 305

Example: Programmable Limit Switch (PLS) enabled on page 306

Update HSC application data

HSC configuration is defined in the HSC application data, and is usually only configured once before programming the HSC instruction. Changes made to the HSC application data (HSCAppData parameter) are ignored while the HSC is counting.

To update the HSC configuration

1. Update HSCAppData.
2. Call the HSC instruction on page 272 with command 0x03 (set/reload).

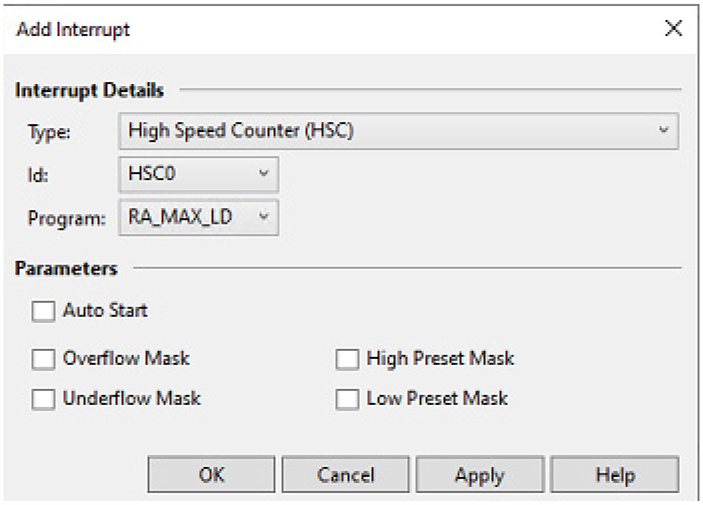
High-Speed Counter (HSC) User Interrupt dialog box

How do I open the High-Speed Counter (HSC) User Interrupt dialog box?

In Interrupt Type, select **High-Speed Counter (HSC) User Interrupt**.

Use the HSC interrupt dialog box to:

- Configure the interrupt properties on page 290, such as ID and the program to use it in.
- Configure the interrupt parameters.



Type: High Speed Counter (HSC)

Id: HSC0

Program: RA_MAX_LD

☐ Auto Start

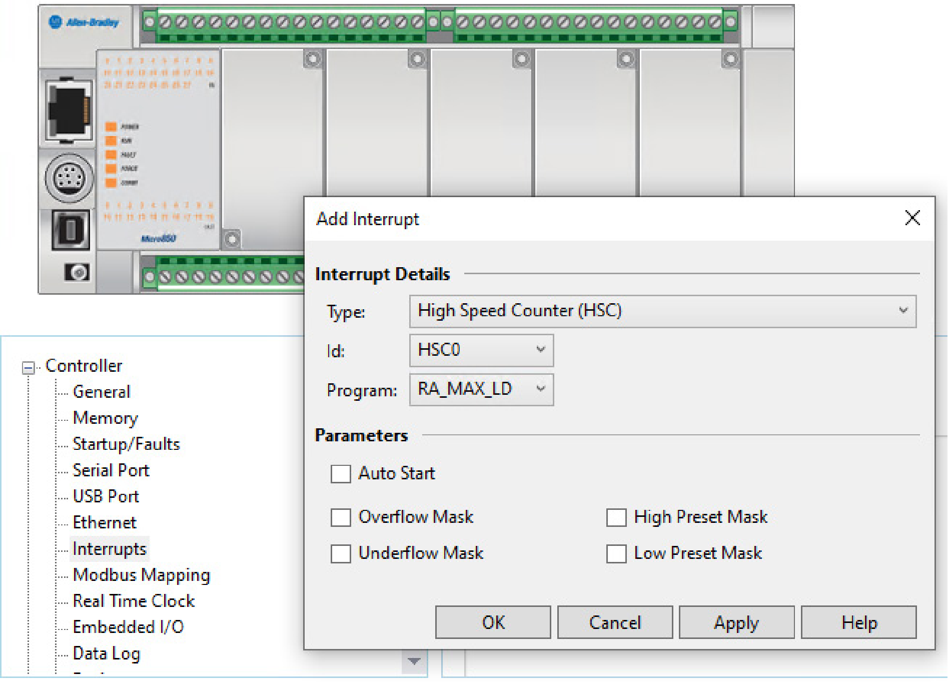
☐ Overflow Mask ☐ High Preset Mask

☐ Underflow Mask ☐ Low Preset Mask

Configure a High-Speed Counter (HSC) User Interrupt

A user interrupt causes the controller to suspend the task it is currently performing, perform a different task, and then return to the suspended task at the point where the task was suspended.

Micro830, Micro850 and Micro870 controllers support up to six HSC User Interrupts that can be used to execute selected user logic at a pre-configured event.



Add Interrupt

Interrupt Details

Type: High Speed Counter (HSC)

Id: HSC0

Program: RA_MAX_LD

Parameters

□ Auto Start

□ Overflow Mask □ High Preset Mask

□ Underflow Mask □ Low Preset Mask

Add and configure a High-Speed Counter (HSC) User Interrupt

To add and configure a HSC interrupt on page 290 from the controller's configuration workspace, perform the following steps.

To add an HSC interrupt:

1. In **Project Organizer**, double-click the controller to open the controller workspace.
2. In the **Controller** tree, click **Interrupts** to display the **Interrupt** configuration page.
3. Right-click an empty row, and click **Add** to open the **Interrupt** properties dialog box.
4. To configure an HSC interrupt:
   - In Interrupt Type, select **High-Speed Counter (HSC) User Interrupt**.
   - Select the HSC Interrupt properties on page 291.
   - Select the HSC Interrupt parameters on page 292.
5. Close the **Interrupt** properties dialog box.

HSC Interrupt properties

The HSC Interrupt properties status bits indicate the enabled/disabled status, the execution status, and whether or not the interrupt condition is lost.

User Interrupt Enable (HSC0.Enabled)

Parameter	Data format	HSC modes	User program access

HSC0.Enabled	bit	0…9	read only

Enabled bit is used to indicate HSC interrupt enable or disable status.

User Interrupt Executing (HSC0.EX)

Parameter	Data format	HSC modes	User program access
HSC0.Ex	bit	0…9	read only

The EX (User Interrupt Executing) bit is set (1) whenever the HSC sub-system begins processing the HSC subroutine due to any of the following conditions:

- Low preset reached
- High preset reached
- Overflow condition - count up through the overflow value
- Underflow condition - count down through the underflow value

The HSC EX bit can be used in the control program as conditional logic to detect if an HSC interrupt is executing.

The HSC sub-system will clear (0) the EX bit when the controller completes its processing of the HSC subroutine.

User Interrupt Pending (HSC0.PE)

Parameter	Data format	HSC modes	User program access
HSC0.PE	bit	0…9	read only

The PE (User Interrupt Pending) status flag indicates an interrupt is pending. The PE status bit can be monitored or used for logic purposes in the control program if you need to determine when a subroutine cannot be immediately executed. The PE bit is maintained by the controller and is set and cleared automatically.

User Interrupt Lost (HSC0.LS)

Parameter	Data format	HSC modes	User program access
HSC0.LS	bit	0…9	read only

The LS (User Interrupt Lost) is a status flag that indicates an interrupt has been lost. The controller can process 1 active user interrupt condition and maintain 1 pending user interrupt condition before it sets the lost bit.

The LS bit is set by the controller. It is up to the control program to use and monitor a lost condition.

HSC Interrupt parameters

The HSC interrupt parameters are used to configure the start and mask options.

Auto Start (HSC0.AS)

Parameter	Data format	HSC modes	User program access
HSC0.AS	bit	0…9	read only

Auto Start is configured with the programming device and stored as part of the user program. The auto start bit defines if the HSC interrupt function automatically starts whenever the controller enters any run or test mode.

Overflow Mask (HSC0.MV)

The **MV** (Overflow Mask) control bit is used to enable (allow) or disable (not allow) an overflow interrupt from occurring. If the bit is clear (0), and an **Overflow Reached** condition is detected by the **HSC**, the **HSC** user interrupt is not executed.

The **MV** bit is controlled by the user program and retains its value through a power cycle. The user program must set and clear the **MV** bit.

Parameter	Data format	HSC modes	User program access
HSC0.MV	bit	0…9	read only

Underflow Mask (HSC0.MN)

Parameter	Data format	HSC modes	User program access
HSC0.MN	bit	2…9	read only

The MN (Underflow Mask) control bit is used to enable (allow) or disable (not allow) an underflow interrupt from occurring. If the bit is clear (0), and an Underflow Reached condition is detected by the HSC, the HSC user interrupt is not executed.

The MN bit is controlled by the user program and retains its value through a power cycle. The user program must set and clear the MN bit.

High Preset Mask (HSC0.MH)

Parameter	Data format	HSC modes	User program access
HSC0.MH	bit	0…9	read only

The MH (High Preset Mask) control bit is used to enable (allow) or disable (not allow) a high preset interrupt from occurring. If this bit is clear (0), and a High Preset Reached condition is detected by the HSC, the HSC user interrupt is not executed.

The MH bit is controlled by the user program and retains its value through a power cycle. The user program must set and clear the MH bit.

Low Preset Mask (HSC0.ML)

Parameter	Data format	HSC modes	User program access
HSC0.ML	bit	2...9	read only

The ML (Low Preset Mask) control bit is used to enable (allow) or disable (not allow) a low preset interrupt from occurring. If this bit is clear (0), and a Low Preset Reached condition is detected by the HSC, the HSC user interrupt is not executed.

The ML bit is controlled by the user program and retains its value through a power cycle. The user program must set and clear the ML bit.

Configure a Programmable Limit Switch (PLS)

The high-speed counter on page 272 has additional operating modes for implementing a Programmable Limit Switch (PLS). The PLS function is used to configure the High-Speed Counter to operate as a PLS or as a rotary cam switch. The PLS function supports up to 255 pairs of high and low presets, and can be used when you need more than one pair of high and low presets.

Enabling PLS in the HSC

The PLS mode only operates in tandem with the HSC of the Micro800 controller, and must be enabled in the HSC instruction by setting the HSCAppData.PLSEnable parameter to True.

The PLSPosition parameter is reset after completing a full cycle and reaching the HSCSTS.HP value. Resetting the HSC instruction or moving 0 to the PLSPositon parameter does not reset the PLSPosition.

HSC operation when PLS is enabled

The PLS function can operate with all other HSC capabilities, including the ability to select which HSC events generate a user interrupt.

When the PLS function is enabled on page 306, and the controller is in run mode, the HSC counts incoming pulses, and the following events occur.

- When the count reaches the first preset (HSCHP or HSCLP) defined in the PLS data, the output source data (HSCHPOutput or HSCLPOutput) is written through the HSC mask (HSCAPP.OutputMask).
- At that point, the next presets (HSCHP and HSCLP) defined in the PLS data become active.
- When the HSC counts to the new preset, the new output data is written through the HSC mask.
- This process continues until the last element within the PLS data block is loaded.
- At that point the active element within the PLS data block is reset to zero.

- This behavior is referred to as circular operation.

The PLS preset difference between embedded HSC and plug-in HSC module

The PLS HSCHP and HSCLP preset behaviors are different between embedded HSC and plug-in HSC module. The embedded HSC High Preset bit will be set only when the last PLS is executed, while the plug-in HSC module High Preset bit will be set when first PLS is executed. For example,

- Embedded HSC PLS0-PLS23:

  High Preset will be set when PLS23HP=Accumulator value
  Low Preset will be set when PLS23LP=Accumulator value
- Plug-in HSC PLS0-PLS23:

  High Preset will be set when PLS0HP=Accumulator value
  Low Preset will be set when PLS0LP=Accumulator value

Example: How to create a High-Speed Counter (HSC) program

This example shows how to create a High-Speed Counter (HSC) on page 272 program that uses a quadrature encoder and includes a Programmable Limit Switch (PLS) function.

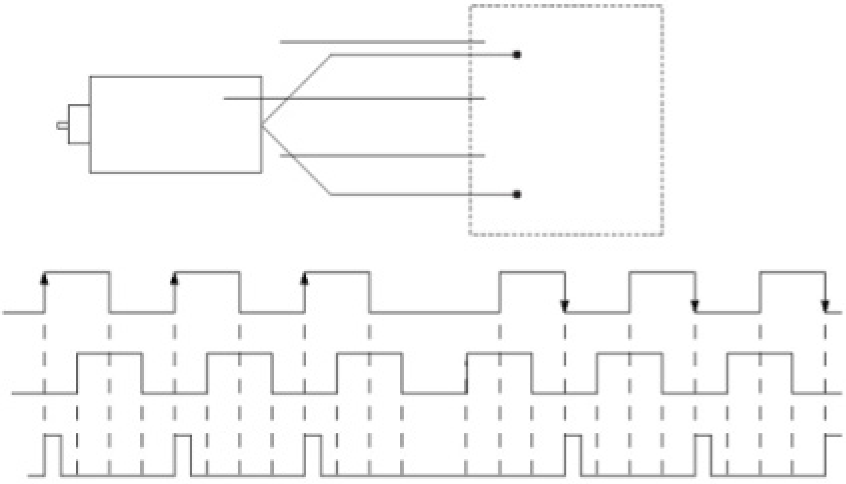
Quadrature encoder used in the example

The High Speed Counter program example uses an HSC function block and a quadrature counter with phased inputs A and B. The quadrature encoder determines the direction of rotation and the position for rotating equipment, such as a lathe. The Bidirectional Counter counts the rotation of the quadrature encoder.

The following quadrature encoder is connected to inputs 0 and 1. The count direction is determined by the phase angle between A and B:

- If A leads B, the counter increments.

- If B leads A, the counter decrements.



Create a High-Speed Counter (HSC) program

Perform the following tasks for to create, build, and test the HSC program, and then add a PLS function.

Table Heading	Table Heading
1	Create a ladder diagram and add variables on page 296
2	Assign values to the HSC variables on page 299
3	Assign variables and build the program on page 300
4	Test the program and run the High-Speed Counter on page 301
5	Add a Programmable Limit Switch (PLS) function on page 305

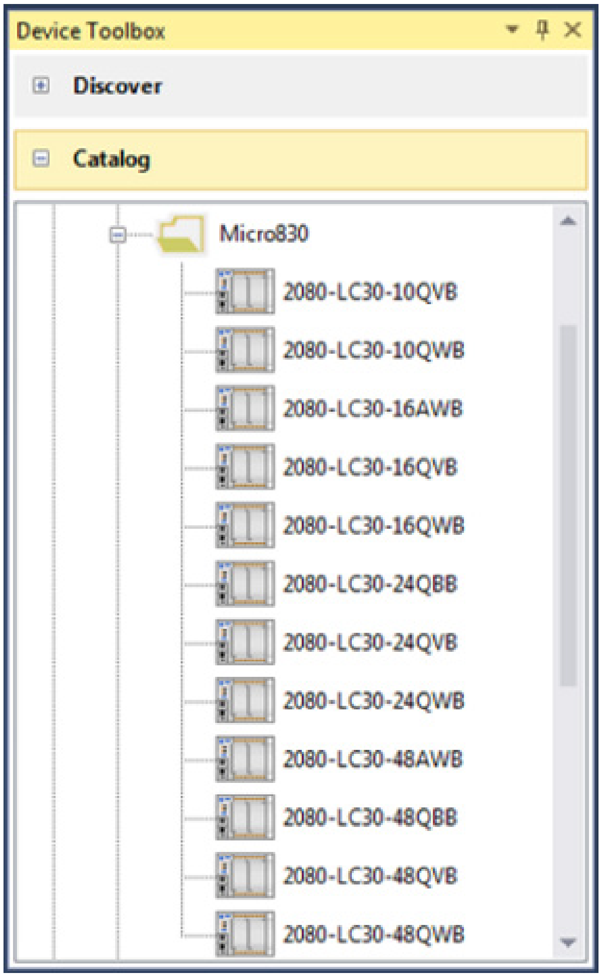
Create a ladder diagram and add variables

Create a ladder diagram and then add local variables to the rung. This sample program uses a 2080-LC50-24QVB controller. The HSC on page 272 is supported on all Micro830 and Micro850 controllers except 2080-LCxx-xxAWB controller types.

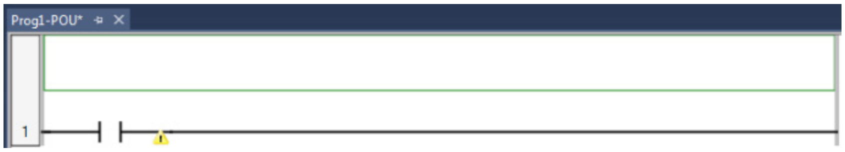
To create a ladder diagram and add variables:

1. In the **Device Toolbox**, expand the **Catalog** tab to view the device folders.

2. Expand the Controllers folder and the Micro830 folder to view all Micro830 controllers. Double-click a controller (2080-LC50-24QVB) to add it to the **Project Organizer**.

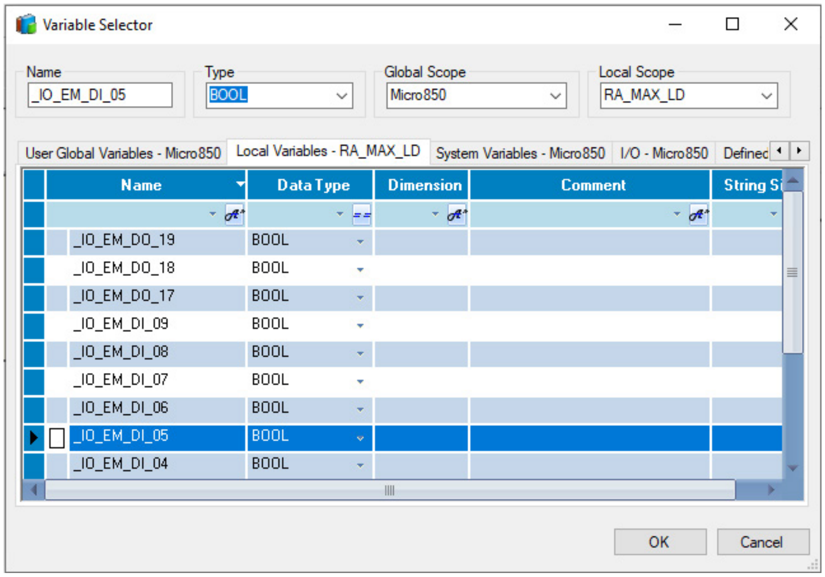


3. In the **Project Organizer**, right-click **Programs**, click **Add,** and then click **New LD: Ladder Diagram** to add a new ladder logic program.
4. Right-click **UntitledLD** and select **Open**.
5. In the **Toolbox** dialog box.
   - Double-click **Direct Contact** to add it to the rung, or
   - Drag and drop a **Direct Contact** onto the rung.



6. Assign a variable to the direct contact:
   - Double-click on the direct contact to display the **Variable Selector**, and then click the **I/O - Micro830** tab.

- Click **_IO_EM_DI_05**, and then click **OK** to assign the direct contact to input 5.



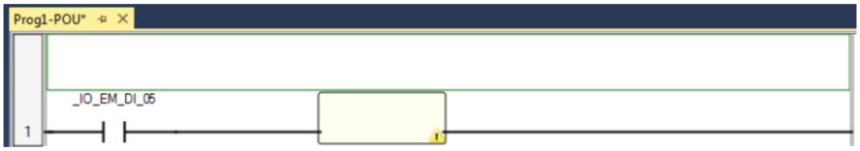
Variable Selector

Name: _IO_EM_DI_05
Type: BOOL
Global Scope: Micro850
Local Scope: RA_MAX_LD

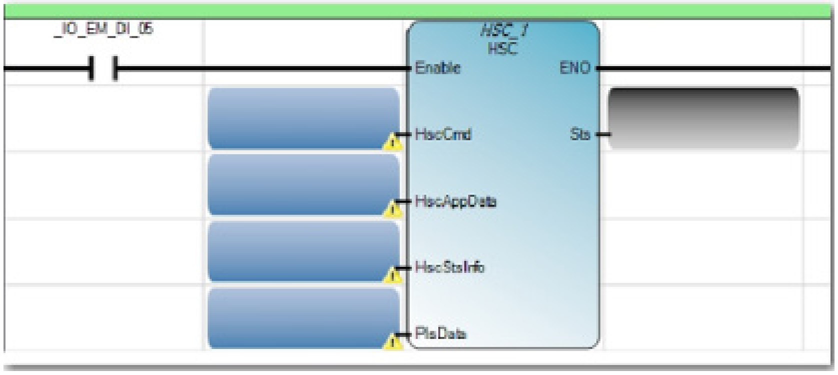
User Global Variables - Micro850
Local Variables - RA_MAX_LD
System Variables - Micro850
I/O - Micro850
Definec

Name	Data Type	Dimension	Comment	String Si
_IO_EM_DO_19	BOOL			
_IO_EM_DO_18	BOOL			
_IO_EM_DO_17	BOOL			
_IO_EM_DI_09	BOOL			
_IO_EM_DI_08	BOOL			
_IO_EM_DI_07	BOOL			
_IO_EM_DI_06	BOOL			
_IO_EM_DI_05	BOOL			
_IO_EM_DI_04	BOOL			

7. In the **Toolbox** dialog box, select a function block and drag it to the right of the direct contact as shown in the following image.



8. Double-click the function block to open the **Instruction Block Selector**.
9. In the **Instruction Block Selector**, select **HSC** and click **OK**.
10. Verify the ladder rung looks similar to the following figure.

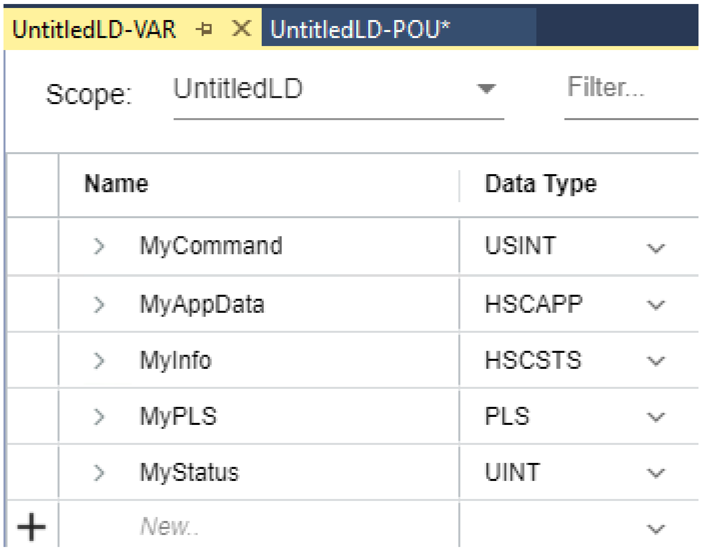


11. In the **Project Organizer**, double-click **Local Variables** to display the **Variables** page.
12. In the **Variables** page, add the following variables and data types.

Variable Name	Data Type
MyCommand	USINT
MyAppData	HSCAPP
MyInfo	HSCSTS
MyPLS	PLS
MyStatus	UINT

Result

The **Variables** page should look like the image below:



Scope: UntitledLD

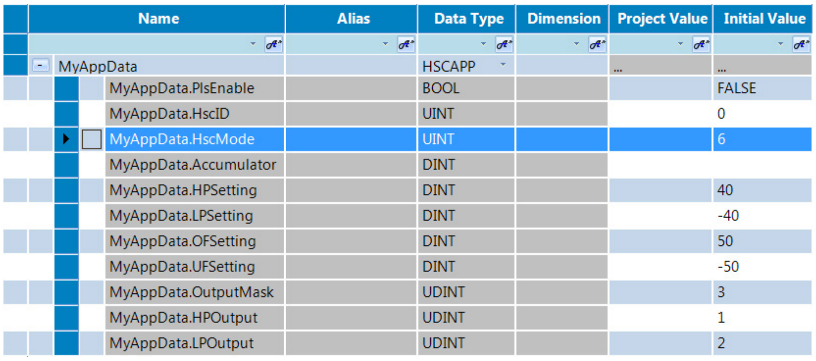
Name	Data Type
MyCommand	USINT
MyAppData	HSCAPP
MyInfo	HSCSTS
MyPLS	PLS
MyStatus	UINT

Assign values to the HSC variables

After you add variables, follow these steps to add values to the variables using the Initial Value column in the **Variable Selector**. A standard program usually uses a routine to assign values to the variables.

To assign values to the HSC variables:

1. Expand **MyAppData** to view all variables.
2. Assign the HSC mode value:
   - In the **Initial Value** field for the MyAppData.HSCMode variable, type 6.
   - See HSCMode in HSCAPP data type on page 274 for more information on the description for each value.
3. Assign the rest of the values to the MyAppData variables as shown in the following figure.
   - In the **Initial Value** field, enter the value.
   - See HSCAPP data on page 274 type for more information on the description for each value.



Name	Alias	Data Type	Dimension	Project Value	Initial Value
MyAppData		HSCAPP		...	...
MyAppData.PlsEnable		BOOL			FALSE
MyAppData.HscID		UINT			0
MyAppData.HscMode		UINT			6
MyAppData.Accumulator		DINT			
MyAppData.HPSetting		DINT			40
MyAppData.LPSetting		DINT			-40
MyAppData.OFSetting		DINT			50
MyAppData.UFSetting		DINT			-50
MyAppData.OutputMask		UDINT			3
MyAppData.HPOutput		UDINT			1
MyAppData.LPOutput		UDINT			2

4. Assign the HSC command value:

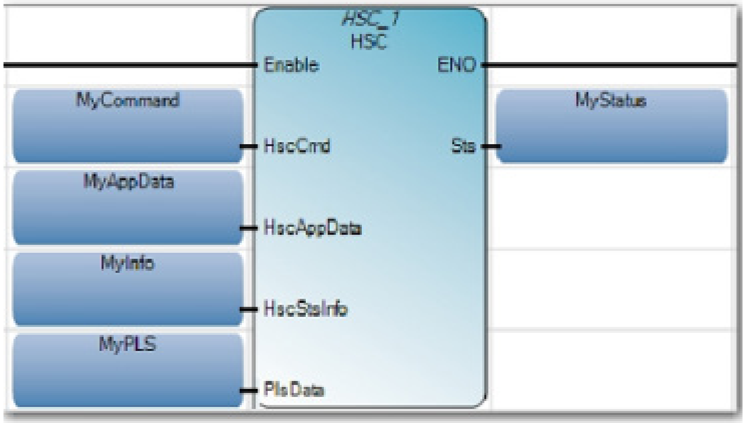
- In the **Initial Value** field for the MyCommand variable, type 1.
- See HSCCmd values on page 273 for more information on command values.

Assign variables and build the program

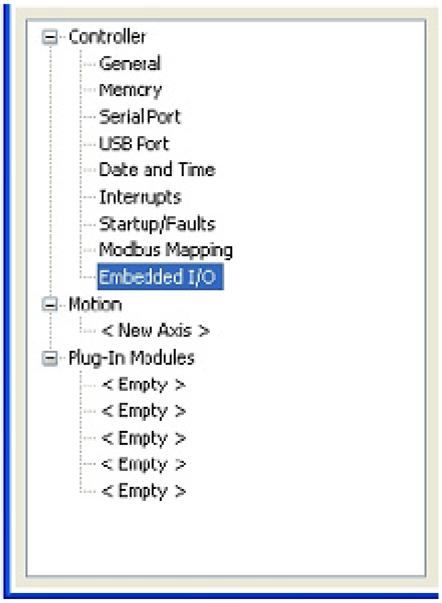
After you enter values in the HSC variables, follow these steps to assign the variables to the function block, and build the program.

To assign variables and build the program:

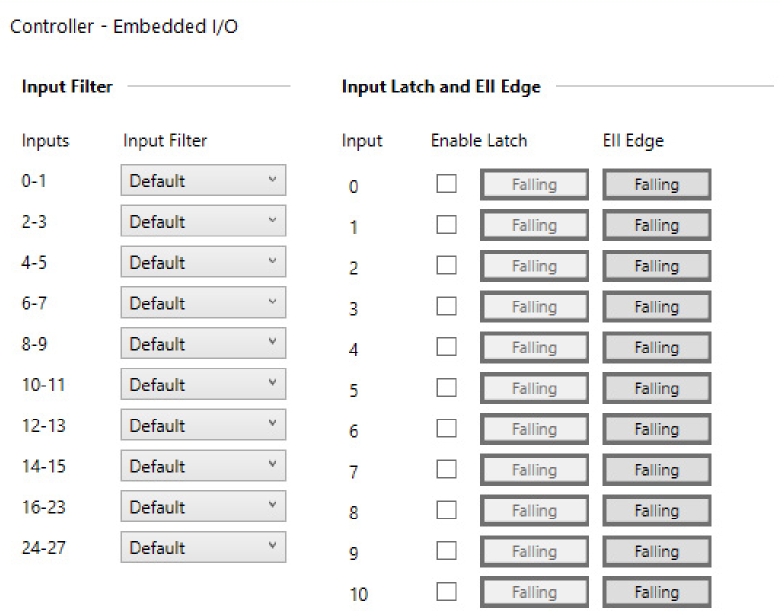
1. From the Ladder Diagram editor, assign each variable to the HSC function block element as shown.



2. From **Project Organizer**, click the controller to display the **Controller** tree.



3. From the **Controller** tree, click **Embedded I/O**, and select input filters for the encoder.



Controller - Embedded I/O

Input Filter

Inputs	Input Filter
0-1	Default
2-3	Default
4-5	Default
6-7	Default
8-9	Default
10-11	Default
12-13	Default
14-15	Default
16-23	Default
24-27	Default

Input Latch and EII Edge

Input	Enable Latch		EII Edge
0	□	Falling	Falling
1	□	Falling	Falling
2	□	Falling	Falling
3	□	Falling	Falling
4	□	Falling	Falling
5	□	Falling	Falling
6	□	Falling	Falling
7	□	Falling	Falling
8	□	Falling	Falling
9	□	Falling	Falling
10	□	Falling	Falling

4. Verify the encoder is connected to the Micro830 controller.
5. Start the Micro830 controller and connect the controller to your computer.
6. Build the program and then download the program to the controller.

Test the program and run the High-Speed Counter

After you download the HSC program to the controller, you can test it and then run the High-Speed Counter.

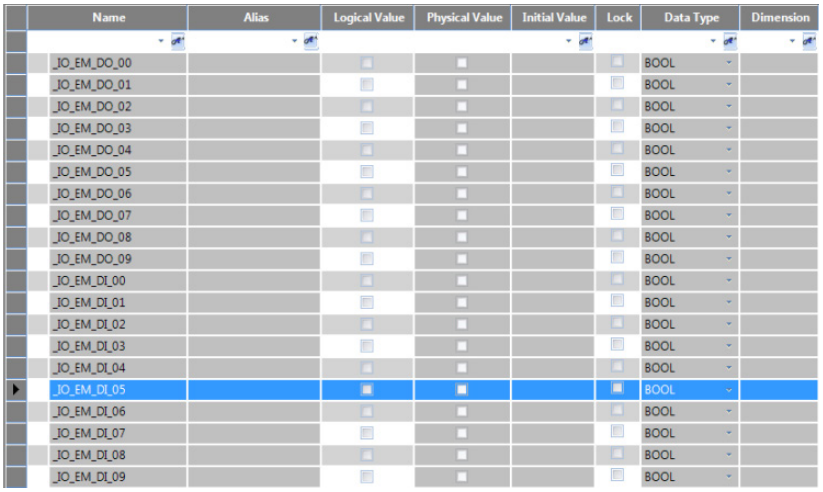
To test the program:

1. Connect to the Controller.
2. From the **Project Organizer**, double-click the HSC program, then double-click **Local Variables**.

   You can see the values of the two HSC outputs: STS (MyStatus) and HSCSTS (MyInfo).

3. Double-click the _IO_EM_DI_05 direct contact to display the **Variable Selector** window.
4. Click the **I/O Micro830** tab, and then click the **_IO_EM_DI_05** row.

5. Select **Lock** and **Logical Value** to force the input to the **ON** position.



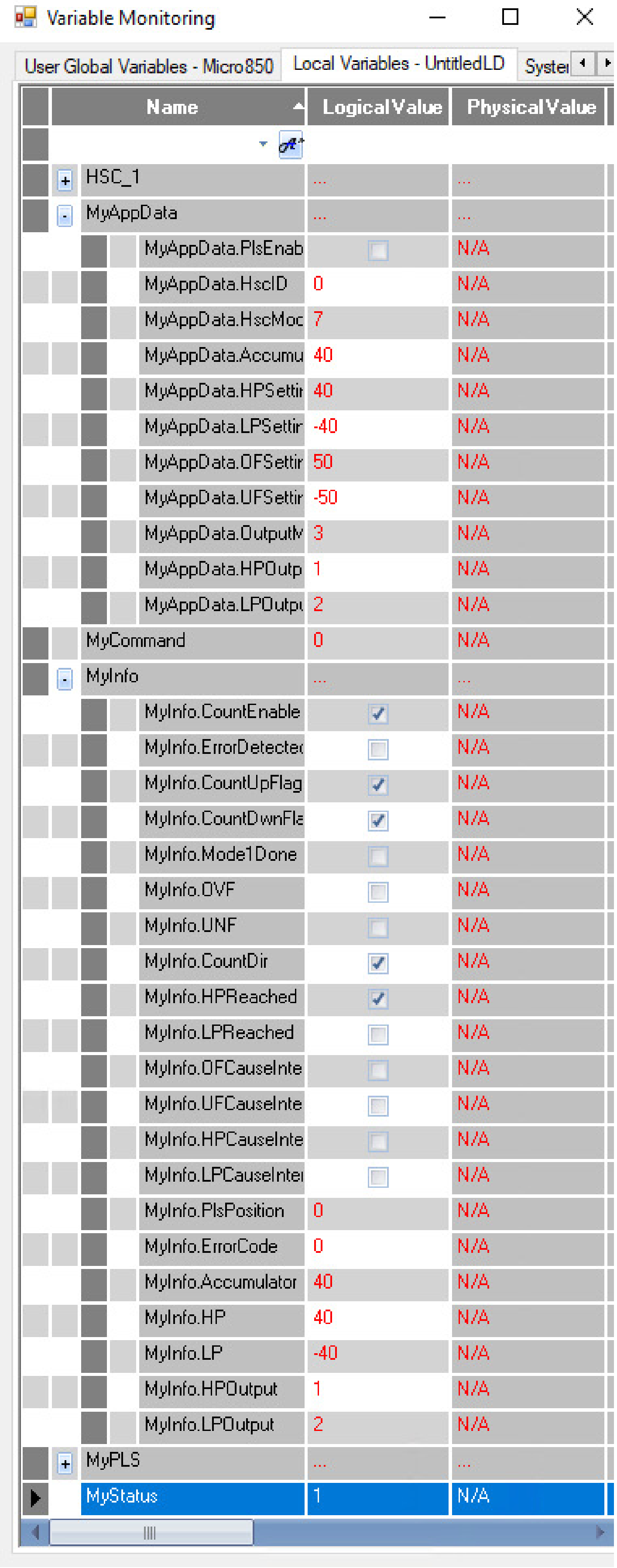
Name	Alias	Logical Value	Physical Value	Initial Value	Lock	Data Type	Dimension
_IO_EM_DO_00		□	□		□	BOOL	
_IO_EM_DO_01		□	□		□	BOOL	
_IO_EM_DO_02		□	□		□	BOOL	
_IO_EM_DO_03		□	□		□	BOOL	
_IO_EM_DO_04		□	□		□	BOOL	
_IO_EM_DO_05		□	□		□	BOOL	
_IO_EM_DO_06		□	□		□	BOOL	
_IO_EM_DO_07		□	□		□	BOOL	
_IO_EM_DO_08		□	□		□	BOOL	
_IO_EM_DO_09		□	□		□	BOOL	
_IO_EM_DI_00		□	□		□	BOOL	
_IO_EM_DI_01		□	□		□	BOOL	
_IO_EM_DI_02		□	□		□	BOOL	
_IO_EM_DI_03		□	□		□	BOOL	
_IO_EM_DI_04		□	□		□	BOOL	
_IO_EM_DI_05		□	□		□	BOOL	
_IO_EM_DI_06		□	□		□	BOOL	
_IO_EM_DI_07		□	□		□	BOOL	
_IO_EM_DI_08		□	□		□	BOOL	
_IO_EM_DI_09		□	□		□	BOOL	

6. To view results, click the **Local Variables** tab to view variable changes.
7. Expand **MyAppData** and **MyInfo** variable list.
8. Turn on the encoder to see the counter count up/down. For example, if the encoder is attached to a motor shaft, then turn on the motor to trigger the HSC count.
9. Verify the Logical Value of in the MyStatus variable is 1, which indicates the HSC is running.
10. View the counter value in MyInfo.Accumulator.

    See HSC (high-speed counter) on page 272 for the complete list of status codes.

Results

In this example, once MyInfo.Accumulator reaches a High Preset value of 40, output 0 turns on and the HPReached flag turns on. If MyInfo.Accumulator reaches a Low Preset value of -40, output 1 turns on and the LPReached flag turns on.


Variable Monitoring
User Global Variables - Micro850
Local Variables - UntitledLD
Syster

Name	Logical Value	Physical Value
+ HSC_1	...	...
- MyAppData	...	...
MyAppData.PlsEnab	□	N/A
MyAppData.HscID	0	N/A
MyAppData.HscMoc	7	N/A
MyAppData.Accumu	40	N/A
MyAppData.HPSettir	40	N/A
MyAppData.LPSettir	-40	N/A
MyAppData.OFSettir	50	N/A
MyAppData.UFSettir	-50	N/A
MyAppData.OutputM	3	N/A
MyAppData.HPOutp	1	N/A
MyAppData.LPOutpu	2	N/A
MyCommand	0	N/A
- MyInfo	...	...
MyInfo.CountEnable	☑	N/A
MyInfo.ErrorDetectec	□	N/A
MyInfo.CountUpFlag	☑	N/A
MyInfo.CountDwnFla	☑	N/A
MyInfo.Mode1Done	□	N/A
MyInfo.OVF	□	N/A
MyInfo.UNF	□	N/A
MyInfo.CountDir	☑	N/A
MyInfo.HPReached	☑	N/A
MyInfo.LPReached	□	N/A
MyInfo.OFCauseInte	□	N/A
MyInfo.UFCauseInte	□	N/A
MyInfo.HPCauseInte	□	N/A
MyInfo.LPCauseInter	□	N/A
MyInfo.PlsPosition	0	N/A
MyInfo.ErrorCode	0	N/A
MyInfo.Accumulator	40	N/A
MyInfo.HP	40	N/A
MyInfo.LP	-40	N/A
MyInfo.HPOutput	1	N/A
MyInfo.LPOutput	2	N/A
+ MyPLS	...	...
MyStatus	1	N/A

Add a Programmable Limit Switch (PLS) function

This example explains how to add a Programmable Limit Switch (PLS) function to the HSC program.

Variable values for the counter settings:

- **MyAppData.PlsEnable** is used to enable or disable the PLS settings. It should be set to FALSE (disabled) if the MyAppData variable is used.
- **MyAppData.HscID** is used to specify which embedded inputs will be used based on the mode and application type. See HSC Inputs and Wiring Mapping to know the different IDs that can be used as well as the embedded inputs and its characteristics.
- If ID 0 is used, ID 1 cannot be used on the same controller because the inputs are used by Reset and Hold.
- **MyAppData.HscMode** is used to specify the type of operation the HSC uses to count. See HSC Mode (HSCAPP.HSCMode) on page 274.

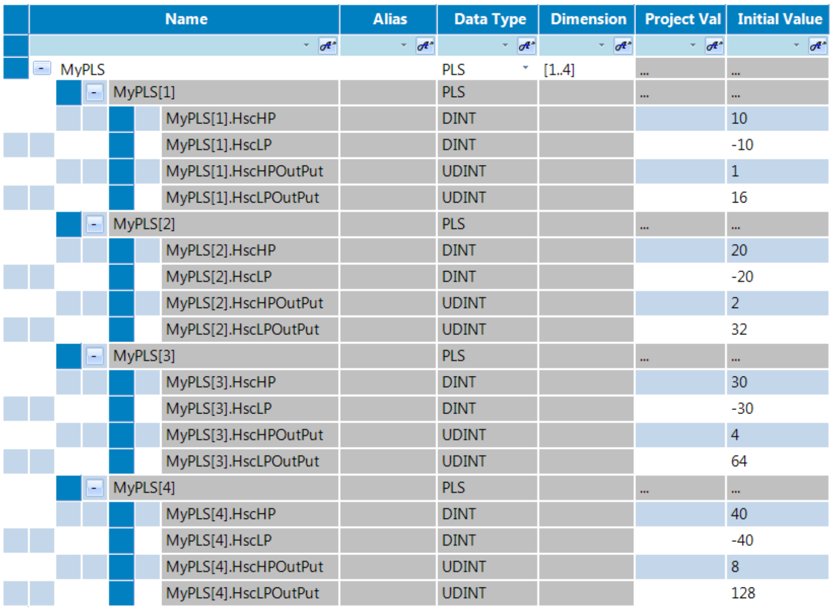
To enable PLS:

1. In **Project Organizer**, double-click **Local Variables** to display the **Variables** page.
2. Enable the PLS function:
   - In the Initial Value field for the MyAppData.PlsEnable variable, select TRUE.
3. Configure the underflow and overflow settings:
   - In the Initial Value field for MyAppData.OFSetting, type 50.
   - In the Initial Value field for MyAppData.UFSetting, type -50.
4. (optional) Configure the output mask if an output.

   Results for this example:

   - The PLS variable has a dimension of [1..4]. This means that HSC can have four pairs of High and Low Presets.
   - High Preset values should be lower than the OFSetting and the Low Preset should be greater than the UFSetting.

- The HscHPOutPut and HscLPOutPut values determine which outputs are turned on when a High Preset or Low Preset is reached.



Name	Alias	Data Type	Dimension	Project Val	Initial Value
MyPLS		PLS	[1..4]	...	...
MyPLS[1]		PLS		...	...
MyPLS[1].HscHP		DINT			10
MyPLS[1].HscLP		DINT			-10
MyPLS[1].HscHPOutPut		UDINT			1
MyPLS[1].HscLPOutPut		UDINT			16
MyPLS[2]		PLS		...	...
MyPLS[2].HscHP		DINT			20
MyPLS[2].HscLP		DINT			-20
MyPLS[2].HscHPOutPut		UDINT			2
MyPLS[2].HscLPOutPut		UDINT			32
MyPLS[3]		PLS		...	...
MyPLS[3].HscHP		DINT			30
MyPLS[3].HscLP		DINT			-30
MyPLS[3].HscHPOutPut		UDINT			4
MyPLS[3].HscLPOutPut		UDINT			64
MyPLS[4]		PLS		...	...
MyPLS[4].HscHP		DINT			40
MyPLS[4].HscLP		DINT			-40
MyPLS[4].HscHPOutPut		UDINT			8
MyPLS[4].HscLPOutPut		UDINT			128

Example: Programmable Limit Switch (PLS) enabled

This example describes the results when PLS is enabled using specific HSC on page 272 and PLSData parameter values.

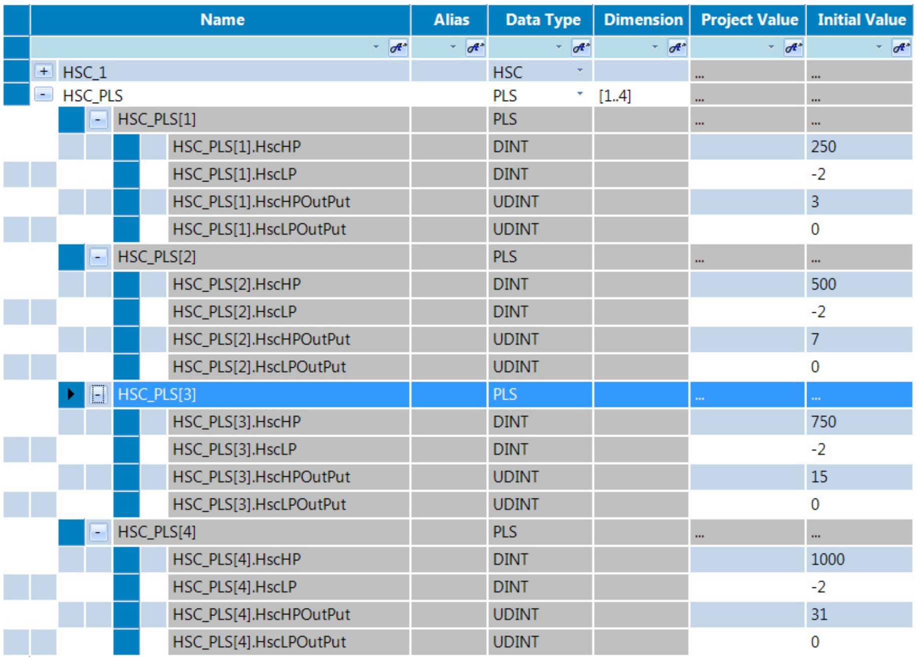
HSC parameter values

This example assumes the HSC parameters are set to the following values:

- HSCApp.OutputMask = 31
- HSCApp.HSCMode = 0
- HSC controls Embedded Output 0...4 only

PLSData parameter values

This example assumes the PLSData parameters for the variable (HSC_PLS) are configured as follows.



Name	Alias	Data Type	Dimension	Project Value	Initial Value
HSC_1		HSC		...	...
HSC_PLS		PLS	[1..4]	...	...
HSC_PLS[1]		PLS		...	...
HSC_PLS[1].HscHP		DINT			250
HSC_PLS[1].HscLP		DINT			-2
HSC_PLS[1].HscHPOutPut		UDINT			3
HSC_PLS[1].HscLPOutPut		UDINT			0
HSC_PLS[2]		PLS		...	...
HSC_PLS[2].HscHP		DINT			500
HSC_PLS[2].HscLP		DINT			-2
HSC_PLS[2].HscHPOutPut		UDINT			7
HSC_PLS[2].HscLPOutPut		UDINT			0
HSC_PLS[3]		PLS		...	...
HSC_PLS[3].HscHP		DINT			750
HSC_PLS[3].HscLP		DINT			-2
HSC_PLS[3].HscHPOutPut		UDINT			15
HSC_PLS[3].HscLPOutPut		UDINT			0
HSC_PLS[4]		PLS		...	...
HSC_PLS[4].HscHP		DINT			1000
HSC_PLS[4].HscLP		DINT			-2
HSC_PLS[4].HscHPOutPut		UDINT			31
HSC_PLS[4].HscLPOutPut		UDINT			0

PLS enabled results

For this example, the following events occur.

- When the ladder logic first runs: HSCSTS.Accumulator = 1, which means all outputs are turned off.
- When HSCSTS.Accumulator = 250, HSC_PLS[1].HSCHPOutput is sent through the HSCAPP.OutputMask, and energizes outputs 0 and 1.
- Sending the high preset output through the output mask repeats as the HSCSTS.Accumulator reaches 500, 750, and 1000, and the controller energizes outputs 0...2, 0...3, and 0...4 respectively.
- After the operation completes, the cycle resets and repeats from HSCSTS.HP = 250.
- When the full cycle completes and the HSCSTS.HP value is reached the PLSPositon parameter is reset.

HSCE instructions

Use HSCE instructions to monitor and control the high-speed counter.

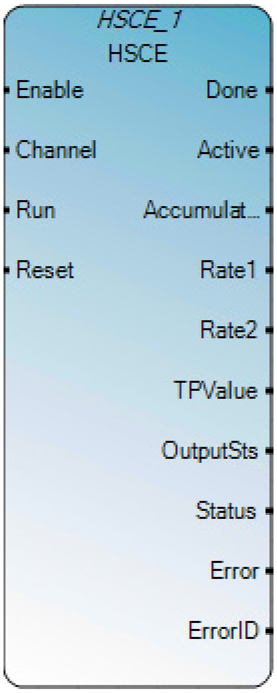
Instruction	Description
HSCE on page 309	HSCE start, stop and read accumulator value.
HSCE_CFG on page 312	HSCE_CFG is the high speed counter configuration.
HSCE_CFG_PLS on page 314	HSCE_CFG_PLS is the high speed counter PLS configuration.
HSCE_READ_STS on page 317	HSCE_READ_STS reads high speed counter status.
HSCE_SET_STS on page 309	HSCE_SET_STS manually set/reset high speed counter status.

HSCE

HSCE is used for controlling and reading the HSC counter.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro830, Micro850, and Micro870 controllers.

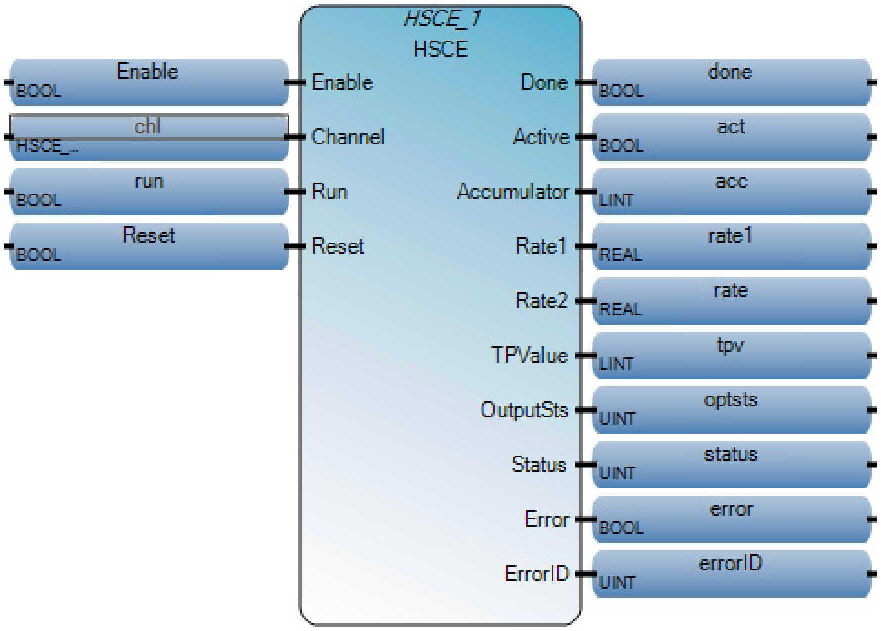


Use this table to help determine the parameter values for this instruction.

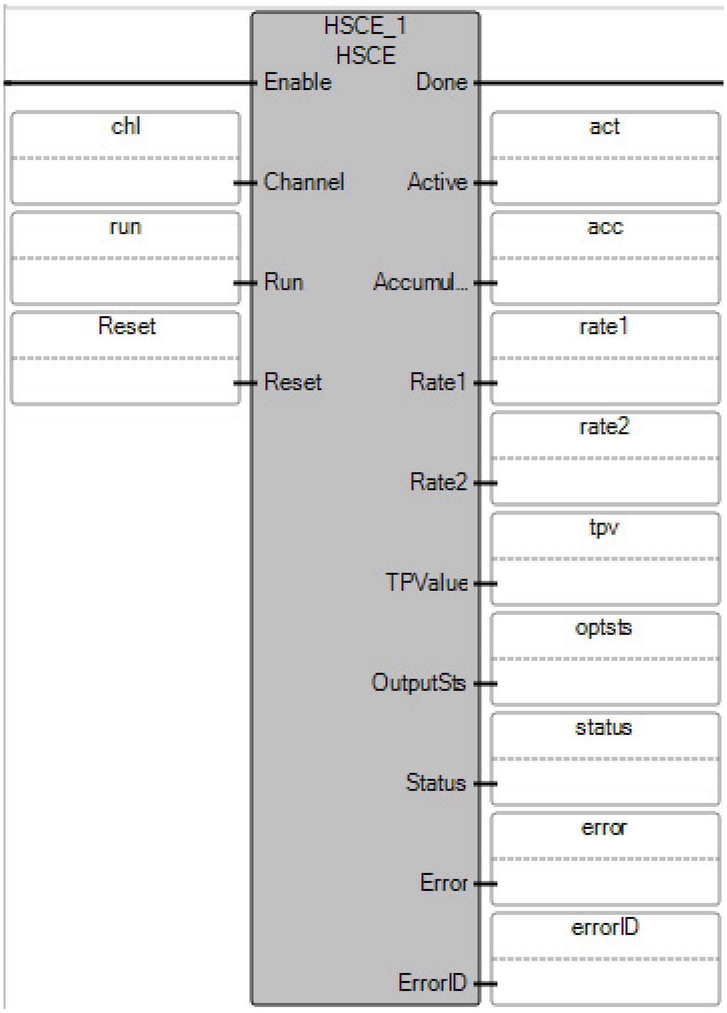
Parameter	Parameter Type	Data Type	Description

Enable	Input	BOOL	TRUE - the HSCE initiates the function block and the plug-in HSC module. Accumulator = InitAcc. Status of HSCE will be reset. Rate1 = 0, Rate2 = 0. FALSE - the Accumulator is updated by reading from the plug-in module. Done, Active, Error = FALSE while ErrorID = 0, Rate1, Rate2 = 0.
Channel	Input	HSCE_CHANNEL	The HSCE channel.
Run	Input	BOOL	For HSCE to count the operational state. TRUE - HSCE counts the pulses. FALSE - HSCE stops the counting.
Reset	Input	BOOL	True - all outputs are cleared and accumulator. Rate1 and Rate2 are also cleared to 0. Plug-in HSC module status clear. The priority of Reset input is higher than the input of Run
Done	Output	BOOL	True - when HSCE Enable is True and no error is detected False - when HSCE Enable is True but Run is False.
Active	Output	BOOL	True - when HSCE Enable is True, Done is True and Run is False. False - when HSCE Enable is True but Run is False.
Rate1	Output	REAL	Current Pulse Rate in user unit per second (Per Pulse method).
Rate2	Output	REAL	Current Pulse Rate in user unit per second (Cyclic method).
TPValue	Output	LINT	Accumulator Value capture when touch probe is triggered.
OutputSts	Output	UINT	HSC plug-in Physical & Virtual Output status and is only applicable for HSC counter 0. Bit 0: Output 0 (Physical Output Status) Bit 1 to 15: Output 1 to Output 15 (Virtual Output Status)
Status	Output	UINT	HSC status information.
Error	Output	BOOL	Indicates an error occurred.
ErrorID	Output	UINT	When an error occurs, ErrorID contains the error code.

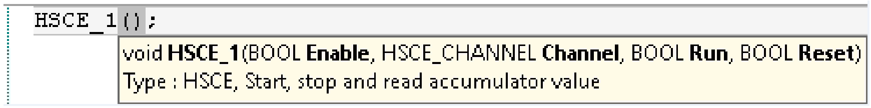
HSCE Function Block Diagram example



HSCE Ladder Diagram example

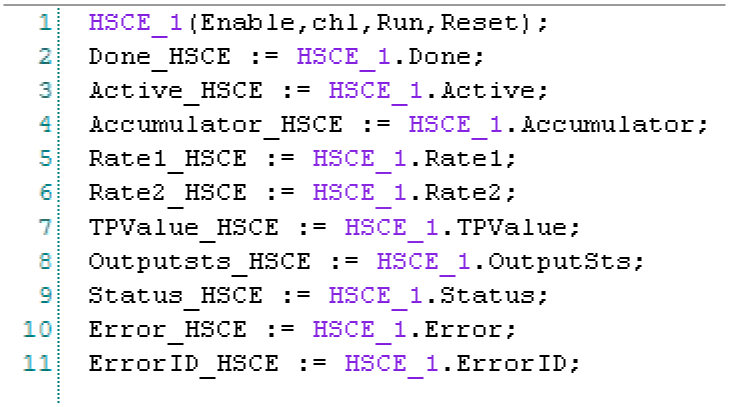


HSCE Text Structure example



```
HSCE_1();
```

void **HSCE_1**(BOOL **Enable**, HSCE_CHANNEL **Channel**, BOOL **Run**, BOOL **Reset**)
Type : HSCE, Start, stop and read accumulator value



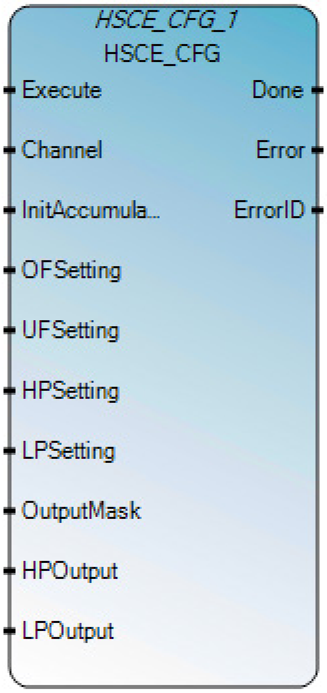
```
HSCE_1(Enable,chl,Run,Reset);
Done_HSCE := HSCE_1.Done;
Active_HSCE := HSCE_1.Active;
Accumulator_HSCE := HSCE_1.Accumulator;
Rate1_HSCE := HSCE_1.Rate1;
Rate2_HSCE := HSCE_1.Rate2;
TPValue_HSCE := HSCE_1.TPValue;
Outputsts_HSCE := HSCE_1.OutputSts;
Status_HSCE := HSCE_1.Status;
Error_HSCE := HSCE_1.Error;
ErrorID_HSCE := HSCE_1.ErrorID;
```

HSCE_CFG

HSCE_CFG is used to configure high speed counter.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

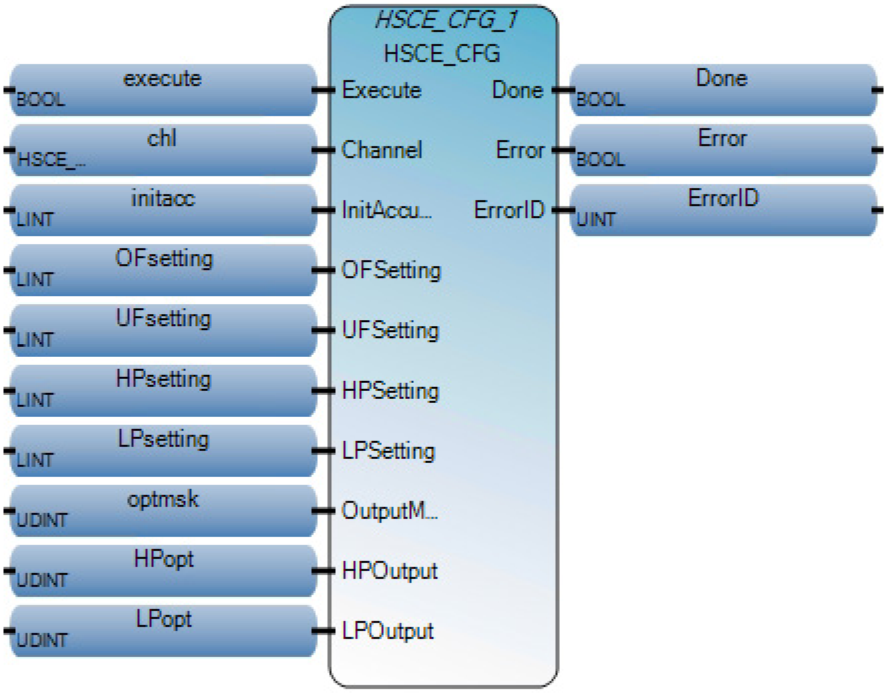
This instruction applies to the Micro820, Micro830, Micro850, and Micro870 controllers.



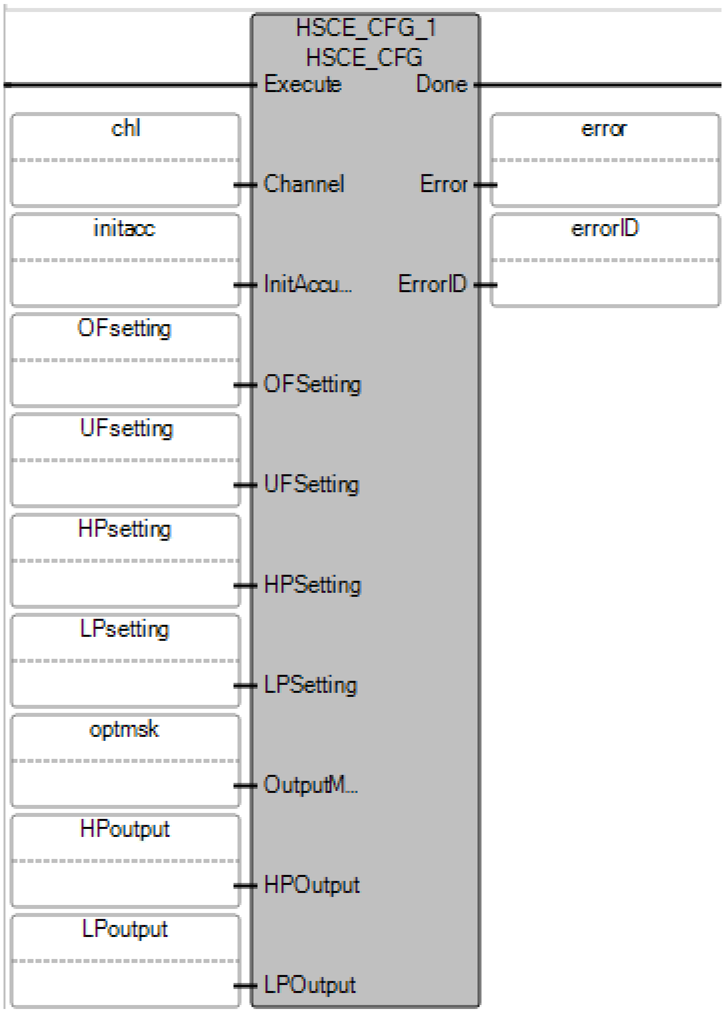
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
Execute	Input	BOOL	Rising Edge initiates the HSC configuration. (HSCE Enable should be equal to FALSE) Falling edge will clear all the output value.
Channel	Input	HSCE_CHANNEL	The HSCE channel.
InitAccumulator	Input	LINT	Accumulator initial value.
OFSetting	Input	LINT	Counter overflow limit value.
UFSetting	Input	LINT	Counter underflow limit value.
HPSetting	Input	LINT	High Preset (HP) Value of HSCE.
LPSetting	Input	LINT	Low Preset (LP) Value of HSCE.
PLS_Offset	Input	USINT	Offset to start with in the PLS data array.
OutputMask	Input	USINT	Output mask for PLS functionality.
HPOutput	Input	UDINT	High preset outputs state.
LPOutput	Input	UDINT	Low preset outputs state.
Done	Output	BOOL	HSC configuration action(initiated by this instruction) succeeds.
Error	Output	BOOL	Indicates an error occurred.
ErrorID	Output	UINT	When an error occurs, ErrorID contains the error code.

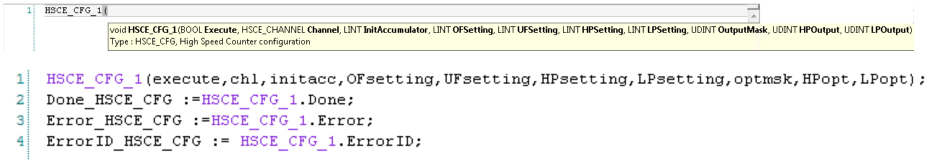
HSCE_CGF Function Block Diagram example



HSCE_CGF Ladder Diagram example



HSCE_CGF Text Structure example



```
HSCE_CFG_1(
```

void **HSCE_CFG_1**(BOOL **Execute**, HSCE_CHANNEL **Channel**, LINT **InitAccumulator**, LINT **OFSetting**, LINT **UFSetting**, LINT **HPSetting**, LINT **LPSetting**, UDINT **OutputMask**, UDINT **HPOutput**, UDINT **LPOutput**)
Type : HSCE_CFG, High Speed Counter configuration

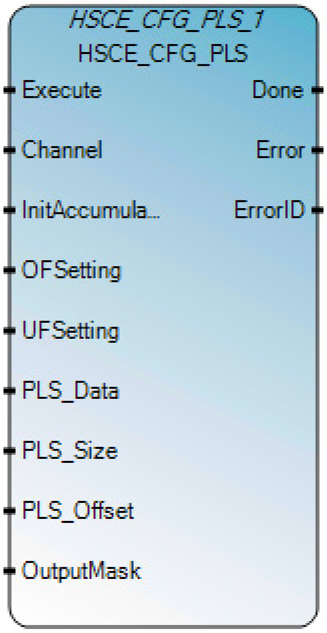
```
HSCE_CFG_1(execute,chl,initacc,OFsetting,UFsetting,HPsetting,LPsetting,optmsk,HPopt,LPopt);
Done_HSCE_CFG :=HSCE_CFG_1.Done;
Error_HSCE_CFG :=HSCE_CFG_1.Error;
ErrorID_HSCE_CFG := HSCE_CFG_1.ErrorID;
```

HSCE_CFG_PLS

This instruction is used for HSC configuration with Programmable Limit Switch (PLS). This function is an additional set of operating modes for the High Speed Counter. When operating in these modes, the preset and output data values are updated using user supplied data each time one of the presets is reached. These modes are programmed by providing a PLS file that contains the data sets to be used.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

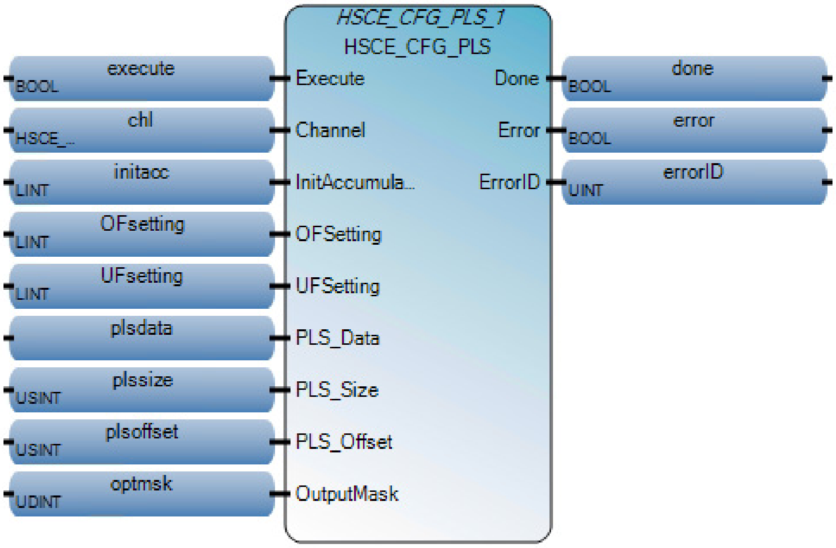
This instruction applies to the Micro820, Micro830, Micro850, and Micro870 controllers.



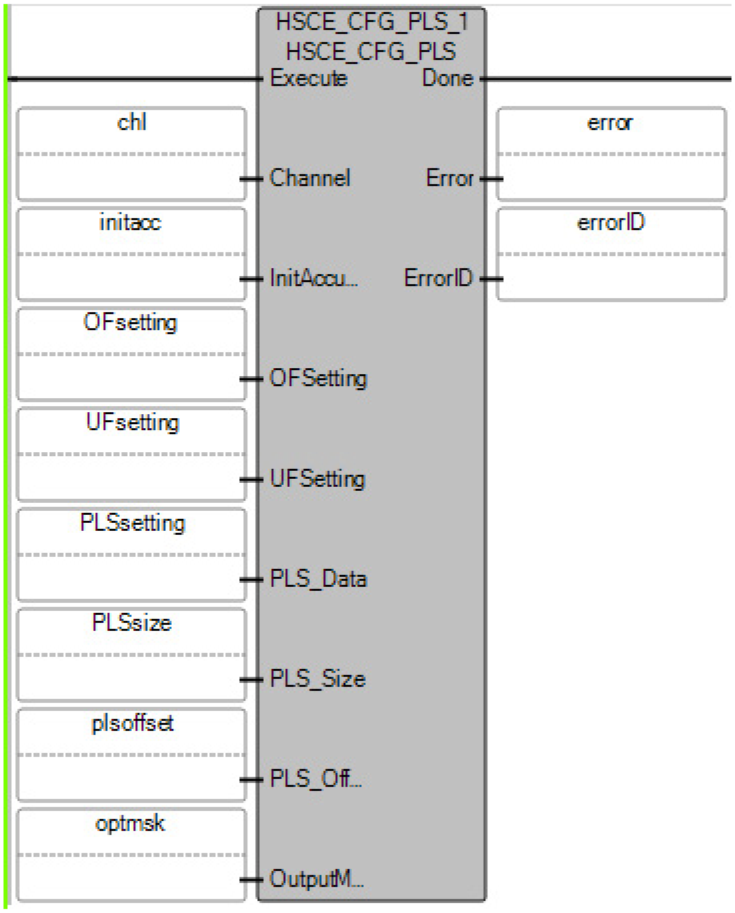
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
Execute	Input	BOOL	Rising Edge initiates the HSC configuration. (HSCE Enable should be equal to FALSE) Falling edge will clear all the output value.
Channel	Input	HSCE_CHANNEL	The HSCE channel.
InitAccumulator	Input	LINT	Accumulator initial value.
OFSetting	Input	LINT	Counter overflow limit value.
UFSetting	Input	LINT	Counter underflow limit value.
PLS_Data	Input	PLS2	Array of PLS (PLS_64)
PLS_Size	Input	USINT	PLS data size, and the maximum value is 24 for plug-in.
PLS_Offset	Input	USINT	Offset to start with in the PLS data array.
OutputMask	Input	USINT	Output mask for PLS functionality.
Done	Output	BOOL	HSC configuration action(initiated by this instruction) succeeds.
Error	Output	BOOL	Indicates an error occurred.
ErrorID	Output	UINT	When an error occurs, ErrorID contains the error code.

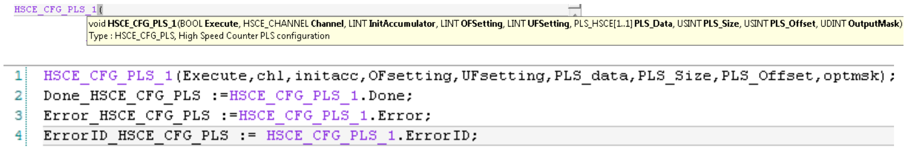
HSCE_CGF_PLS Function Block Diagram example



HSCE_CGF_PLS Ladder Diagram example



HSCE_CGF_PLS Text Structure example



```
HSCE_CFG_PLS_1(
void HSCE_CFG_PLS_1(BOOL Execute, HSCE_CHANNEL Channel, LINT InitAccumulator, LINT OFSetting, LINT UFSetting, PLS_HSCE[1..1] PLS_Data, USINT PLS_Size, USINT PLS_Offset, UDINT OutputMask)
Type : HSCE_CFG_PLS, High Speed Counter PLS configuration
```

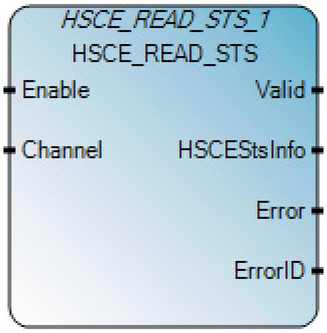
```
HSCE_CFG_PLS_1(Execute,chl,initacc,OFsetting,UFsetting,PLS_data,PLS_Size,PLS_Offset,optmsk);
Done_HSCE_CFG_PLS :=HSCE_CFG_PLS_1.Done;
Error_HSCE_CFG_PLS :=HSCE_CFG_PLS_1.Error;
ErrorID_HSCE_CFG_PLS := HSCE_CFG_PLS_1.ErrorID;
```

HSCE_READ_STS

The instruction is used to read current HSC status.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

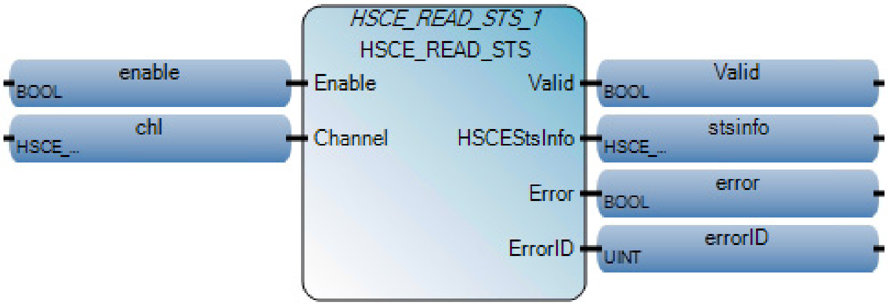
This instruction applies to the Micro820, Micro830, Micro850, and Micro870 controllers.



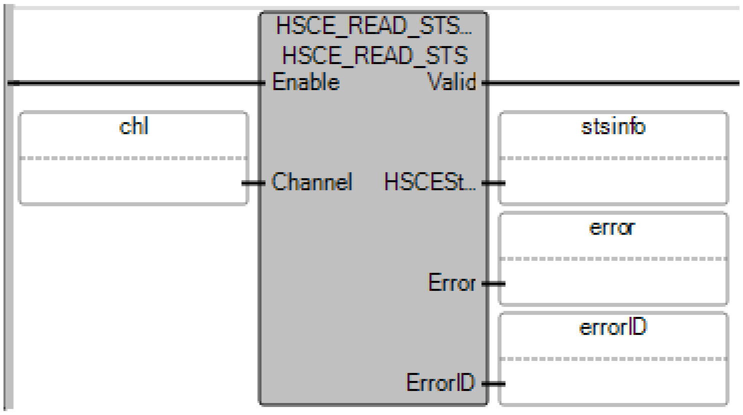
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
Enable	Input	BOOL	If Enable is True then HSC2StsInfo is updated.
Channel	Input	HSCE_CHANNEL	The HSCE channel.
Valid	Output	BOOL	HSC2StsInfo is Valid if TRUE.
HSCEStsInfo	Input	LINT	Counter overflow limit value.
Error	Output	BOOL	Indicates an error occurred.
ErrorID	Output	UINT	When an error occurs, ErrorID contains the error code.

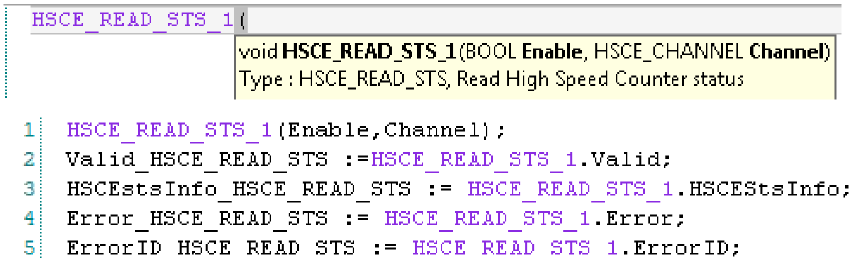
HSCE_READ_STS Function Block Diagram example



HSCE_READ_STS Ladder Diagram example



HSCE_READ_STS Text Structure example



```
HSCE_READ_STS_1(
void HSCE_READ_STS_1(BOOL Enable, HSCE_CHANNEL Channel)
Type : HSCE_READ_STS, Read High Speed Counter status
```

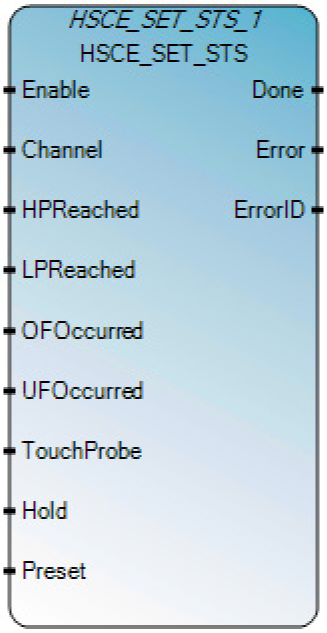
```
HSCE_READ_STS_1(Enable,Channel);
Valid_HSCE_READ_STS :=HSCE_READ_STS_1.Valid;
HSCEstsInfo_HSCE_READ_STS := HSCE_READ_STS_1.HSCEStsInfo;
Error_HSCE_READ_STS := HSCE_READ_STS_1.Error;
ErrorID HSCE READ STS := HSCE READ STS 1.ErrorID;
```

HSCE_SET_STS

The instruction manually sets and resets the HSC counting status flags. The HSC function block must be stopped for the HSCE_SET_STC function block to set or reset its HTS status. If the HSC function block is not stopped or HSC channel is not configured, HSC2_SET_STS function block will throw error.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

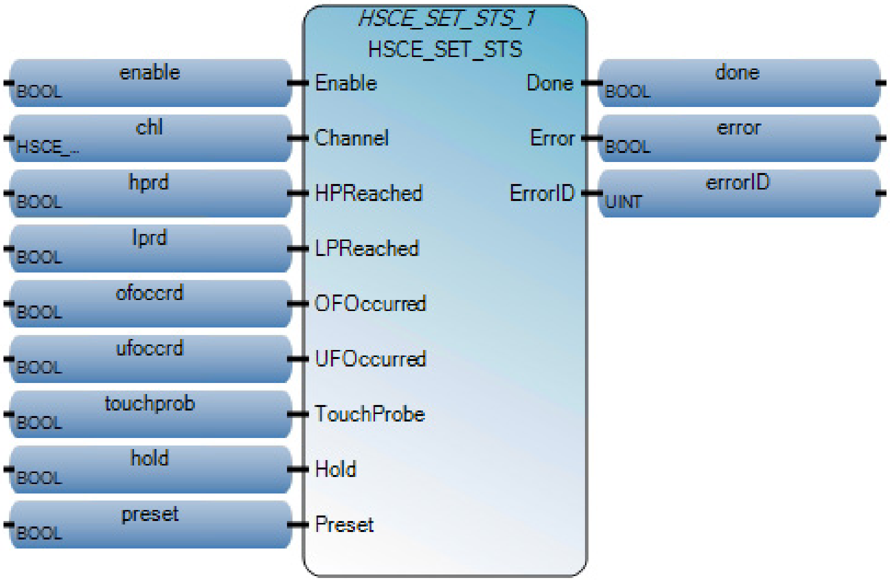
This instruction applies to the Micro820, Micro830, Micro850, and Micro870 controllers.



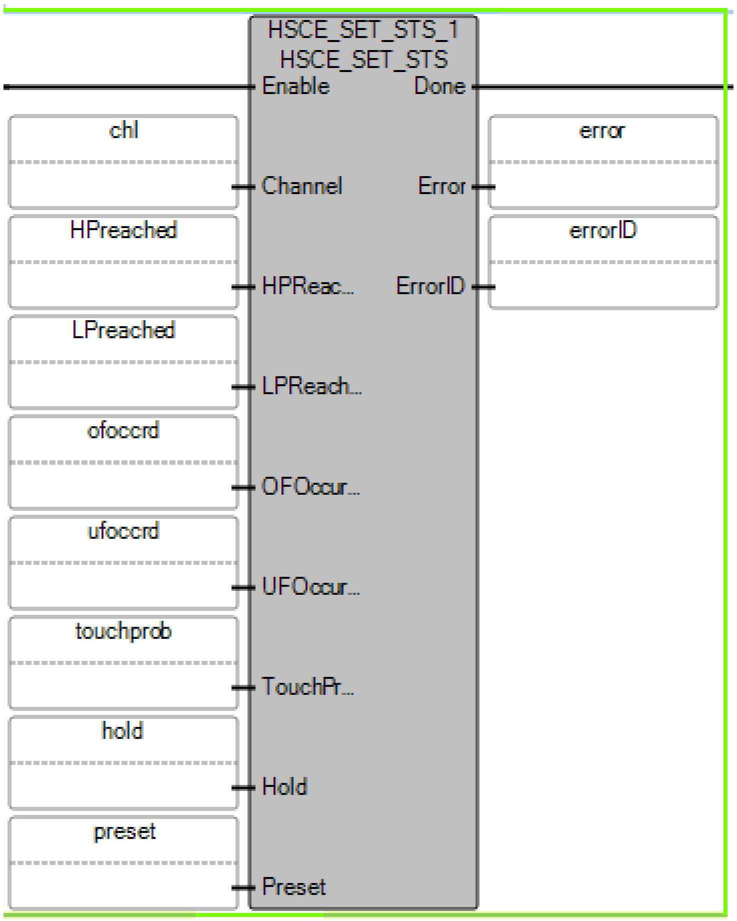
Use this table to help determine the parameter values for this instruction.

Parameter	**Parameter Type**	**Data Type**	**Description**
Enable	Input	BOOL	Function block enable. TRUE - set/reset the HSC status. FALSE - there is no HSC status change.
Channel	Input	HSCE_CHANNEL	The HSCE channel.
HPReached	Output	BOOL	When High preset value reached, this bit will be set to TRUE by plug-in module. Set or reset this bit by HSCE_SET_STS function block.
LPReached	Input	BOOL	When low preset value is reached, this bit will be set to be TRUE by plug-in module. Set or reset this bit by HSCE_SET_STS function block.
OFOccurred	Input	BOOL	When overflow occurred, this bit will be set to be TRUE by plug-in module.Set or reset this bit by HSCE_SET_STS function block.
UFOccurred	Input	BOOL	When underflow occurred, this bit will be set to be TRUE by plug-in module. Set or reset this bit by HSCE_SET_STS function block.
TouchProbe	Input	BOOL	When touch probe is triggered, this bit will be set to be TRUE by plug-in module.
Hold	Input	BOOL	When HSC hold is triggered this bit will be set to be TRUE by plugin module.Set or reset this bit by HSCE_SET_STS function block. This input parameter is only effective for counter 0. For counter 1 is always reset
Preset	Input	BOOL	When Z ACC Reset is triggered, this bit will be set to be TRUE by plug-in module. Set or reset this bit by HSCE_SET_STS function block. This input parameter is only effective for counter 0. For counter 1 is always reset.
Done	Output	BOOL	HSC configuration action(initiated by this instruction) succeeds.
Error	Output	BOOL	Indicates an error occurred.
ErrorID	Output	UINT	When an error occurs, ErrorID contains the error code.

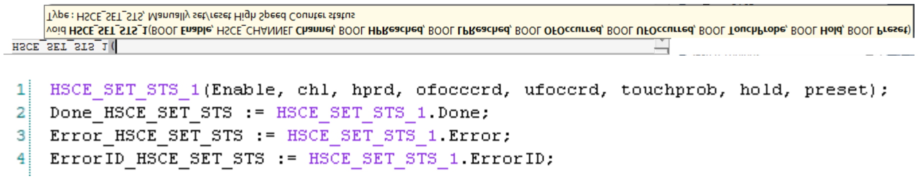
HSCE_SET_STS Function Block Diagram example



HSCE_SET_STS Ladder Diagram example



HSCE_SET_STS Text Structure example



```
Type : HSCE_SET_STS, Manually set/reset High Speed Counter status
void HSCE_SET_STS_1(BOOL Enable, HSCE_CHANNEL Channel, BOOL HPReached, BOOL LPReached, BOOL OFOccurred, BOOL UFOccurred, BOOL TouchProbe, BOOL Hold, BOOL Preset)
HSCE_SET_STS_1(
```

```
HSCE_SET_STS_1(Enable, chl, hprd, ofocccrd, ufoccrd, touchprob, hold, preset);
Done_HSCE_SET_STS := HSCE_SET_STS_1.Done;
Error_HSCE_SET_STS := HSCE_SET_STS_1.Error;
ErrorID_HSCE_SET_STS := HSCE_SET_STS_1.ErrorID;
```

HSCE error codes

The following table describes the status error codes for HSCE instructions on page 309:

ErrorID Code	Error description	Corrective action
0	The instruction successfully completed operation.	
1	Invalid HSC configuration files.	Contact the Rockwell Automation technical support representative. For contact information, see: http://www.rockwellautomation.com/support
2	Invalid HSC module type.	Correct the module type. For example, select module type as plug-in.
3	Invalid HSC Slot ID.	Correct the Slot ID in the Channel input of the function block.
4	Invalid HSC ID.	Correct the HSC ID. For example, set 0 for counter 0.
5	Invalid mode for the Channel.	Contact Rockwell Automation technical support representative.
6	Invalid PLS Size.	PLS size should be ≤24 and size of the PLS data array ≥PLS size.
7	Invalid PLS offset.	PLS offset + PLS size should be within the size of the PLS data array.
8	Invalid InitAccumulator value.	Correct the InitAccumulator value. Maybe it is over the boundary or out range of LP or HP limit (LP ≤ InitACC ≤ HP).
9	Invalid LP.	Correct the LP value. It may be over the boundary.
10	Invalid OF.	Correct the OF value.
11	Invalid UF.	Correct the UF value.
12	Invalid HP.	Correct the HP value.
13	There is no configuration for HSCE.	Contact the Rockwell Automation technical support representative.
14	Invalid HSCE state.	Check the HSCE related function block to confirm the state of this function block.
15	Invalid plug-in module	Check the plug-in module to confirm it is HSC module.
16	HSCE is running.	When HSC is counting, no configuration and setting status are allowed. HSCE_CFG on page 312 and HSCE_CFG_PLS on page 314 are executed while HSCE is running.
17	Stop HSC plug-in failure.	Contact the Rockwell Automation technical support representative.
18	Update rate write failure.	Contact the Rockwell Automation technical support representative.
19	Write the number of plus failure.	Contact the Rockwell Automation technical support representative.
20	Write Apply ACC failure.	Contact the Rockwell Automation technical support representative.
21	Write ACC failure.	Contact the Rockwell Automation technical support representative.
22	Write Apply failure.	Contact the Rockwell Automation technical support representative.
23	Write number of PLS failure.	Contact the Rockwell Automation technical support representative.
24	Write under flow failure.	Contact the Rockwell Automation technical support representative.
25	Write over flow failure.	Contact the Rockwell Automation technical support representative.
26	Write low preset failure.	Contact the Rockwell Automation technical support representative.
27	Write high preset failure.	Contact the Rockwell Automation technical support representative.
28	Write low preset out failure.	Contact the Rockwell Automation technical support representative.

ErrorID Code	Error description	Corrective action
29	Write high preset out failure.	Contact the Rockwell Automation technical support representative.
30	Write out mask failure.	Contact the Rockwell Automation technical support representative.
31	Write PLS low preset out failure.	Contact the Rockwell Automation technical support representative.
32	Write PLS high preset out failure.	Contact the Rockwell Automation technical support representative.
33	Write PLS low preset failure.	Contact the Rockwell Automation technical support representative.
34	Write PLS high preset out failure.	Contact the Rockwell Automation technical support representative.
35	Write PLS offset failure.	Contact the Rockwell Automation technical support representative.
36	Write PLS number failure.	Contact the Rockwell Automation technical support representative.
37	Read status failure.	Contact the Rockwell Automation technical support representative.
38	Read high preset failure.	Contact the Rockwell Automation technical support representative.
39	Read low preset failure.	Contact the Rockwell Automation technical support representative.
40	Read ACC failure.	Contact the Rockwell Automation technical support representative.
41	Read number of pulse width failure.	Contact the Rockwell Automation technical support representative.
42	Read pulse width failure.	Contact the Rockwell Automation technical support representative.
43	Read number of pulse failure.	Contact the Rockwell Automation technical support representative.
44	Read update rate failure.	Contact the Rockwell Automation technical support representative.
45	Write status failure.	Contact the Rockwell Automation technical support representative.
46	Read low preset out failure.	Contact the Rockwell Automation technical support representative.
47	Read high preset out failure.	Contact the Rockwell Automation technical support representative.
48	Read PLS number failure.	Contact the Rockwell Automation technical support representative.
49	N.A.	Contact the Rockwell Automation technical support representative.
50	Write apply status failure.	Contact the Rockwell Automation technical support representative.
51	Invalid channel input.	Contact the Rockwell Automation technical support representative.
52	Read touch probe failure.	Contact the Rockwell Automation technical support representative.
53	Write reset ACC failure.	Contact the Rockwell Automation technical support representative.
54	Start HSC failure.	Contact the Rockwell Automation technical support representative.
55	Counter is disabled.	The HSC counter is disabled. Check the Channel configuration to verify if the HSC counter is enabled.
56	Invalid output mask value.	Check if the output mask is within the valid range. For plug-in HSC module, the range is 0-65535.
57	Invalid high preset output.	Check the if the HP output is within the valid range. For plug-in HSC module, the range is 0-65535.
58	Invalid low preset output.	Check if the LP output is within the valid range. For plug-in HSC module, the range is 0-65535.
59	Not a supported UPM revision.	Check the revision configuration for the HSC plug-in module.
60	Not a HSC module is added.	Check if the actual plug-in module is HSC module.
61	HSCE plug-in module is not configured.	Contact the Rockwell Automation technical support representative.
62	A UPM plug-in Write error occurs while changing HSC plug-in Mode to Non-run.	Contact the Rockwell Automation technical support representative.
63	A UPM plug-in Write error occurs while changing HSC plug-in Mode to run mode.	Contact the Rockwell Automation technical support representative.
64	A UPM plug-in Read error occurs while reading Output Status.	Contact the Rockwell Automation technical support representative.

Input/Output instructions

Use Input/Output instructions to read or write data to or from a controller or module using signals sent to a device that is physically connected to a programmable logic controller. Input relays transfer signals to the internal relays, and output relays transfer signals to external output devices.

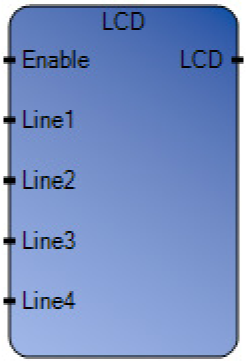
Instruction	Description
LCD on page 325	Micro810 only. Displays a string or number on an LCD screen.
LCD_BKLT_REM on page 328	Sets the remote LCD backlight parameters in a user program.
LCD_REM on page 330	Displays user defined messages for the remote LCD.
RHC on page 332	Reads the high speed clock value in the Micro800 controller.
RPC on page 334	Reads the user program checksum, either from the controller or memory module.
DLG on page 335	Writes variable values from the run-time engine into a Data Logging File on an SD Card.
IIM on page 337	Updates inputs prior to normal output scan.
IOM on page 339	Updates outputs prior to normal output scan.
KEY_READ on page 341	Micro810 only. Reads the Key status on the optional LCD module when the user display is active.
KEY_READ_REM on page 343	Micro820 only. Reads the Key status on the optional remote LCD module when the user display is active.
MM_INFO on page 345	Reads memory module header information.
MODULE_INFO on page 348	Reads module information from a plug-in or expansion module excluding the 2080-MEMBAK-RTC memory module.
PLUGIN_INFO on page 359	Reads module information from a generic plug-in or expansion module excluding the 2080-MEMBAK-RTC memory module.
PLUGIN_READ on page 361	Reads data from a generic plug-in module excluding the 2080-MEMBAK-RTC memory module.
PLUGIN_RESET on page 363	Resets the hardware for a generic plug-in module excluding the 2080-MEMBAK-RTC memory module.
PLUGIN_WRITE on page 365	Writes data to a generic plug-in module excluding the 2080-MEMBAK-RTC memory module.
RCP on page 367	Reads and writes recipe data to and from an SD memory card.
RTC_READ on page 369	Reads the real-time clock (RTC) module information.
RTC_SET on page 371	Sets RTC data to the RTC module information.
SYS_INFO on page 373	Reads the status data block for the Micro800 controller.
TRIMPOT_READ on page 375	Reads the trimpot value from a specific trimpot.

LCD

Displays a string or a number on the optional LCD screen.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

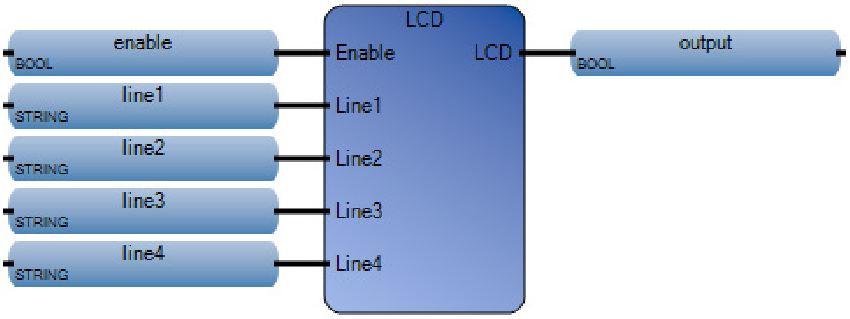
This instruction applies to the Micro810 controllers.



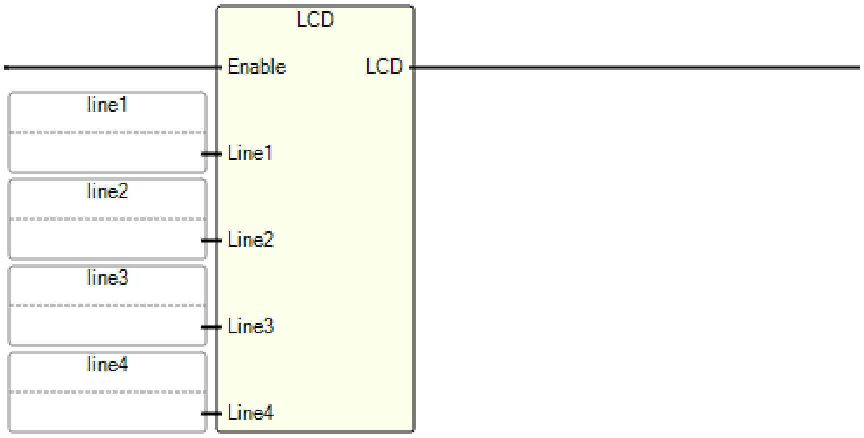
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
Enable	Input	BOOL	Instruction enable. TRUE - the LCD switches to the user-defined screen (strings display on the LCD screen) instead of the I/O status screen. FALSE - the LCD displays the contents of the I/O status screen.
Line1	Input	STRING	String displayed on line 1 of the LCD.
Line2	Input	STRING	String displayed on line 2 of the LCD.
Line3	Input	STRING	String displayed on line 3 of the LCD.
Line4	Input	STRING	String displayed on line 4 of the LCD.
LCD	Output	BOOL	TRUE - function is enabled.

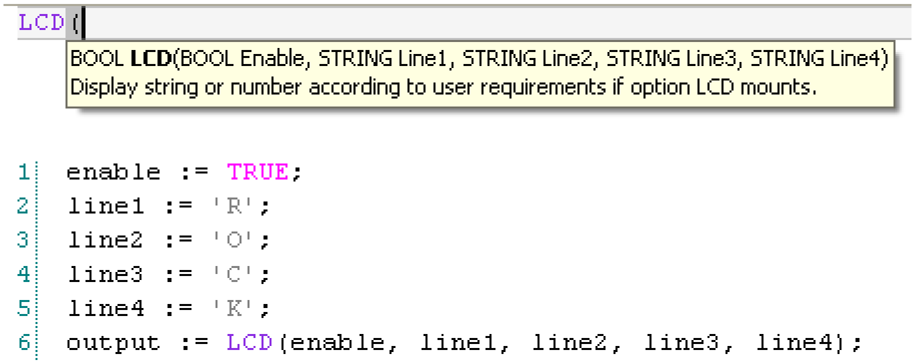
LCD Function Block Diagram example



LCD Ladder Diagram example



LCD Structured Text example



```
LCD (
```

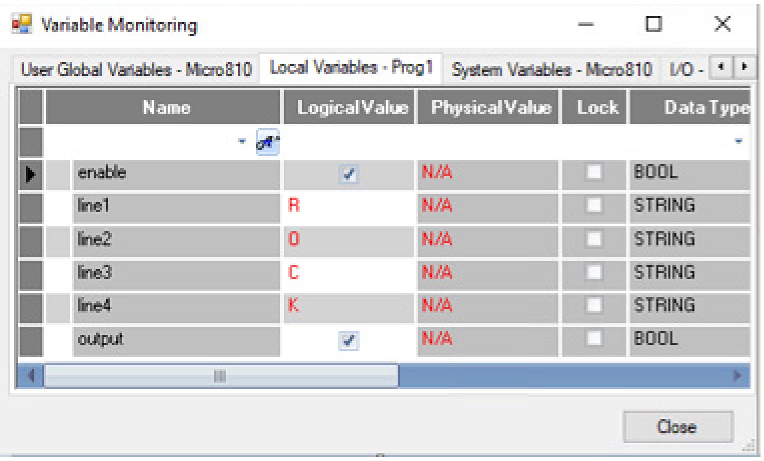
BOOL **LCD**(BOOL Enable, STRING Line1, STRING Line2, STRING Line3, STRING Line4)
Display string or number according to user requirements if option LCD mounts.

```
enable := TRUE;
line1 := 'R';
line2 := 'O';
line3 := 'C';
line4 := 'K';
output := LCD(enable, line1, line2, line3, line4);
```

(* ST Equivalence: *)

TESTOUTPUT := LCD(LCDENABLE, LINE1, LINE2, LINE3, LINE4) ;

Results



Name	Logical Value	Physical Value	Lock	Data Type
enable	☑	N/A	□	BOOL
line1	R	N/A	□	STRING
line2	O	N/A	□	STRING
line3	C	N/A	□	STRING
line4	K	N/A	□	STRING
output	☑	N/A	□	BOOL

LCD_BKLT_REM (remote LCD backlight)

Sets the Remote LCD backlight parameters in a user program.

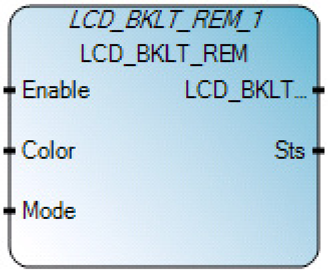
Operation details:

The backlight settings defined in LCD_BKLT_REM are used when the Remote LCD display is:

- a user defined screen defined using LCD_REM.
- the default IO Status screen.
- For all other screens, the backlight settings used are those defined using the menu on the Remote LCD.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820 controllers.

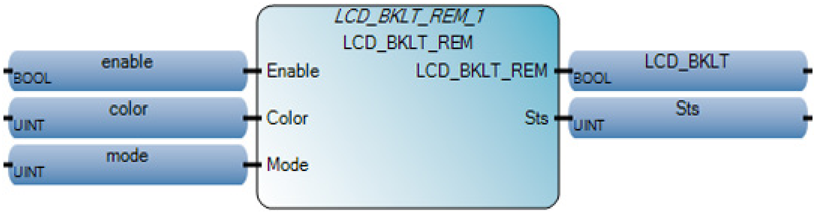


Use this table to help determine the parameter values for this instruction.

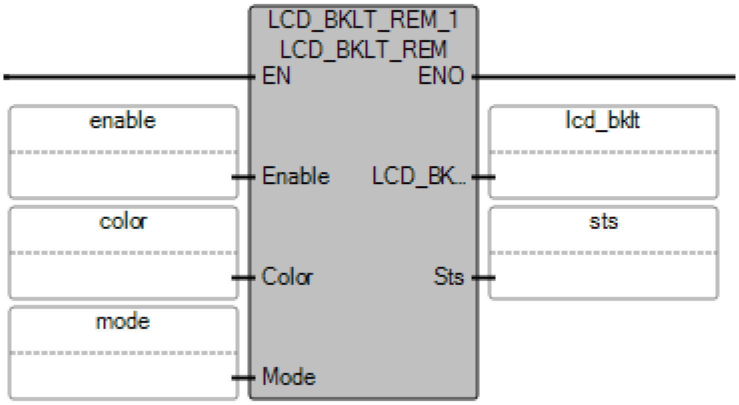
Parameter	Parameter Type	Data Type	Description
Enable	Input	BOOL	Instruction enable. TRUE - execute REM_LCD_BKLT, overwrite any current backlight settings. FALSE - REM_LCD_BKLT is disabled and the Remote LCD menu settings take effect.
Color	Input	UINT	Backlight Color Code • 0: White • 1: Blue • 2: Red • 3: Green • 4-65535: Reserved
Mode	Input	UINT	• 0 : Permanently OFF • 1: Permanently ON • 2: Flash (1 sec interval) • 3-65535: Reserved
LCD_BKLT_REM	Output	BOOL	TRUE - Instruction executed successfully. FALSE - Error occurred during instruction execution.

Sts	Output	UINT	Status of the remote LCD operation. LCD_BKLT_REM status codes: • 0 - Enable input is false. • 1 - Success. • 2 - Remote LCD not detected. May occur when: • Remote LCD is not physically connected to the controller or the wiring is incorrect. • Serial port settings are other than what is required for the Remote LCD. • 3 - Connection error. May occur when there is an internal state machine error such as an incompatibility between Controller FW version and RLCD FW version. • 4 - Invalid color code. • 5 - Invalid mode. • 6-65535 - Reserved.

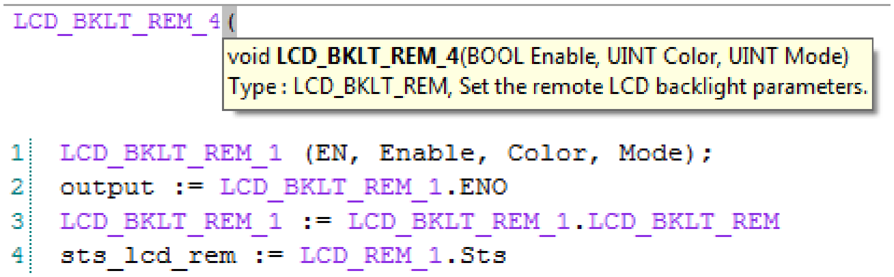
LCD_BKLT_REM Function Block Diagram examples



LCD_BKLT_REM Ladder Diagram example



LCD_BKLT_REM Structured Text example



```
LCD_BKLT_REM_4(
```

void **LCD_BKLT_REM_4**(BOOL Enable, UINT Color, UINT Mode)
Type : LCD_BKLT_REM, Set the remote LCD backlight parameters.

```
LCD_BKLT_REM_1 (EN, Enable, Color, Mode);
output := LCD_BKLT_REM_1.ENO
LCD_BKLT_REM_1 := LCD_BKLT_REM_1.LCD_BKLT_REM
sts_lcd_rem := LCD_REM_1.Sts
```

LCD_REM (remote LCD)

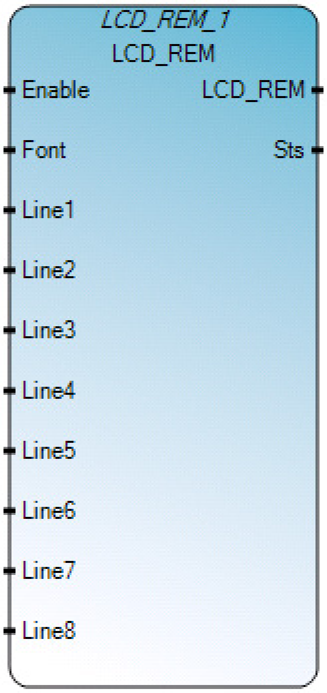
Displays user defined messages for the Remote LCD.

Operation details:

- For Line1 through Line8 the maximum string length is 24 characters.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820 controllers.

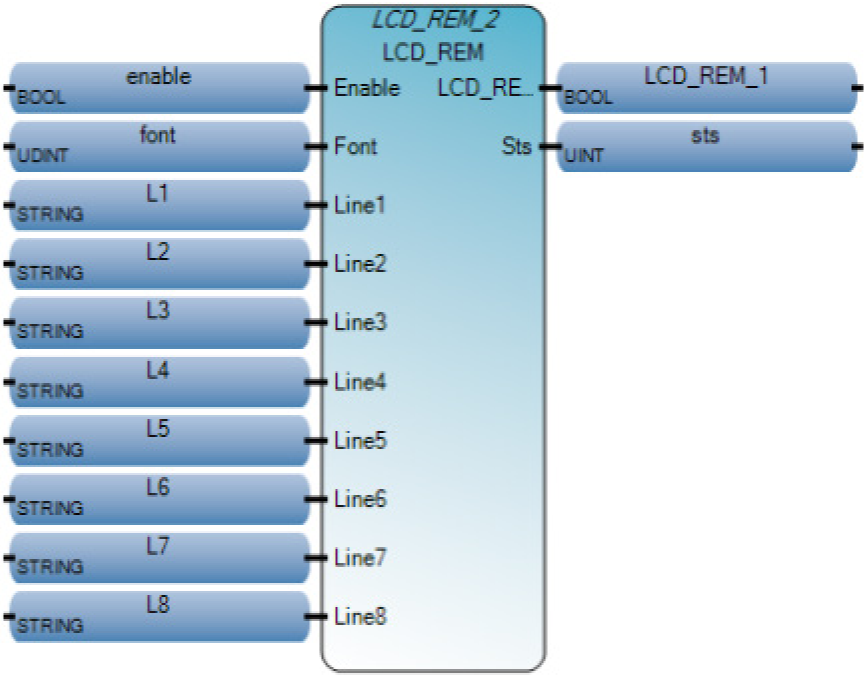


Use this table to help determine the parameter values for this instruction.

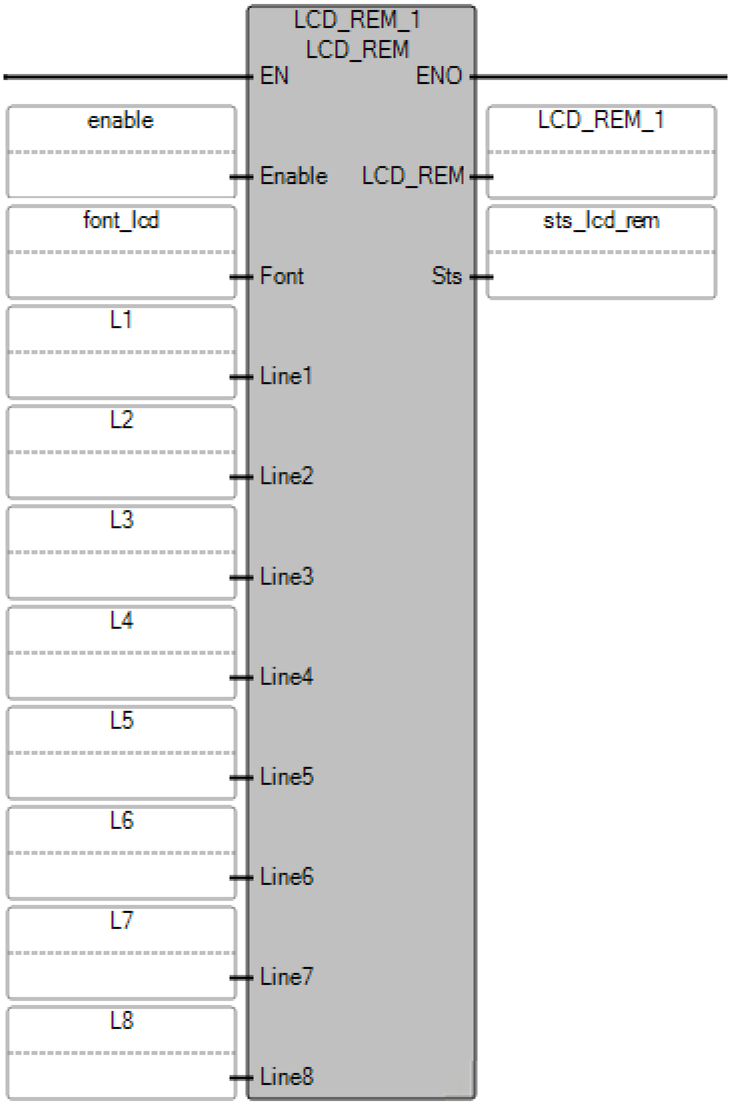
Parameter	Parameter Type	Data Type	Description
Enable	Input	BOOL	Enable the instruction block. TRUE - remote LCD switches to user-defined screen from I/O status screen. FALSE - remote LCD switches back to I/O status screen.
Font	Input	UDINT	Font size for startup message: • 0: Default (Large – 8x16) • 1: Small (8x8) • 2: Large (8x16) • 3: Extra Large (16x16) • 4 onward: Reserved The Remote LCD size is 192x64 pixels. When the font size for the start up message is small font, the Remote LCD displays: • strings in Line1 to Line8. • a maximum of 24 characters per line. When the font size for the start up message is large or extra large, the Remote LCD displays: • strings in Line1 to Line4. • ignores strings in Line5 to Line8. • a maximum of 12 characters per line.

Line1	Input	String	String to be displayed on line 1 of the LCD. For Line1 through Line8 the maximum string length is 24 characters.
Line2	Input	String	String displayed on line 2 of the LCD.
Line3	Input	String	String displayed on line 3 of the LCD.
Line4	Input	String	String displayed on line 4 of the LCD.
Line5	Input	String	String displayed on line 5 of the LCD.
Line6	Input	String	String displayed on line 6 of the LCD.
Line7	Input	String	String displayed on line 7 of the LCD.
Line8	Input	String	String displayed on line 8 of the LCD.
LCD_REM	Output	BOOL	Function block enable. When Enable = TRUE, user display is active. When Enable = FALSE, IO Status/Menu display is active.
Sts	Output	UINT	Status of the remote LCD operation. LCD_REM status codes: • 0 - Enable input is false. • 1 - User message displayed successfully. • 2 - Remote LCD not detected. • 3 - Connection error. May occur when: • Remote LCD is not physically connected to the controller (or the wiring is incorrect). • Serial port settings are other than what is required for the Remote LCD. • 4 - Invalid font code. • 5 - 5-65535 - Reserved.

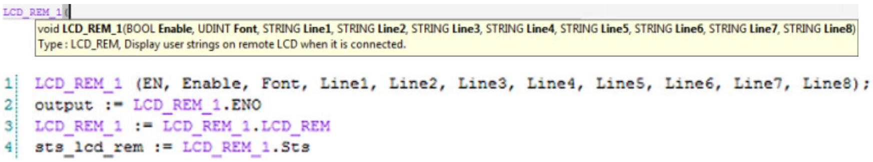
LCD_REM Function Block Diagram example



LCD_REM Ladder Diagram example



LCD_REM Structured Text example



```
LCD_REM_1(
void LCD_REM_1(BOOL Enable, UDINT Font, STRING Line1, STRING Line2, STRING Line3, STRING Line4, STRING Line5, STRING Line6, STRING Line7, STRING Line8)
Type : LCD_REM, Display user strings on remote LCD when it is connected.
```

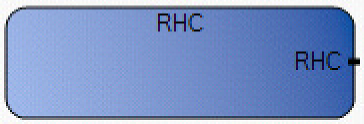
```
LCD_REM_1 (EN, Enable, Font, Line1, Line2, Line3, Line4, Line5, Line6, Line7, Line8);
output := LCD_REM_1.ENO
LCD_REM_1 := LCD_REM_1.LCD_REM
sts_lcd_rem := LCD_REM_1.Sts
```

RHC (read high speed clock)

Reads the high speed clock value in the Micro800 controller.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



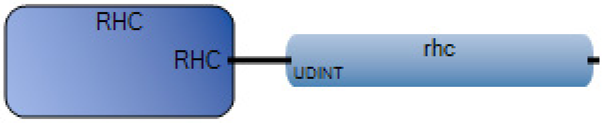
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - read high speed clock. FALSE - no operation. Applies to Ladder Diagram programs.
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.
RHC	Output	UDINT	The value of the high speed clock.

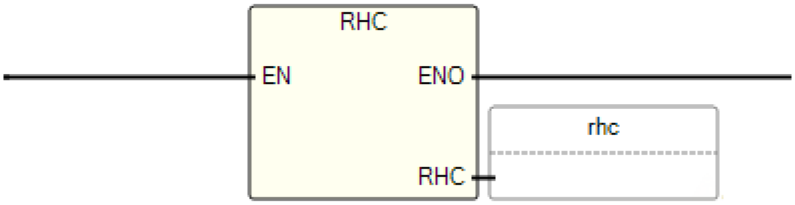
High-speed clock resolution

Controller Type	Increments	Timebase	Resolution
Micro810	4 every 40 microseconds	10 microseconds	40 microseconds
Micro820 Micro830 Micro850	1 every 10 microseconds	10 microseconds	10 microseconds

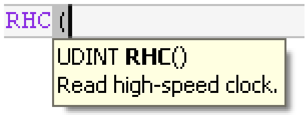
RHS Function Block Diagram example



RHS Ladder Diagram example

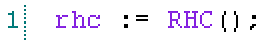


RHS Structured Text example



```
RHC (
```

UDINT **RHC()**
Read high-speed clock.



```
rhc := RHC();
```

(* ST Equivalence: *)

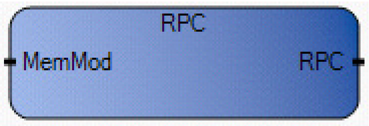
TESTOUTPUT2 := RHC() ;

RPC (read program checksum)

Reads the user program checksum, either from the controller or memory module.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

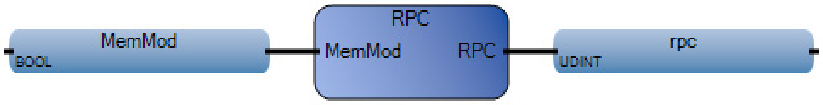
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



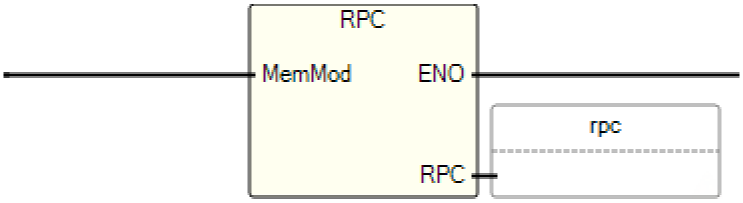
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
MemMod	Input	BOOL	TRUE - the value is taken from the memory module. FALSE - the value is taken from the Micro800 controller.
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.
RPC	Output	UDINT	The checksum value of the specified user program.

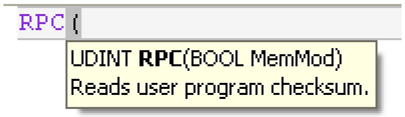
RPC Function Block Diagram example



RPC Ladder Diagram example

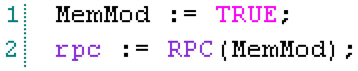


RPC Structured Text example



```
RPC (
```

UDINT **RPC**(BOOL MemMod)
Reads user program checksum.



```
MemMod := TRUE;
rpc := RPC(MemMod);
```

(* ST Equivalence: *)

TESTOUTPUT2 := RPC(TESTINPUT) ;

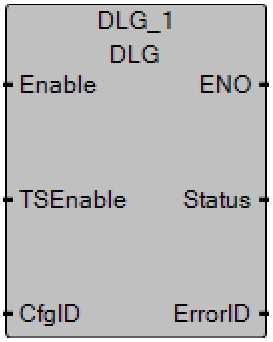
DLG (data log)

Writes variable values from the run-time engine into a Data Logging File on an SD Card.

When writing to a data log a maximum of 50 group folders are allowed per day. Each group folder has a maximum of 50 files with a file size of 4k-8k.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro830, Micro850, and Micro870 controllers.



Use this table to help determine the parameter values for this instruction.

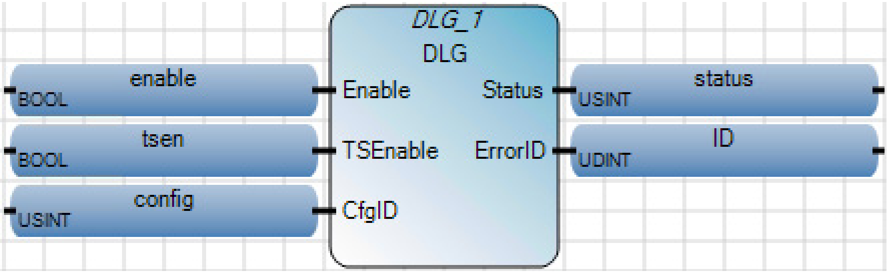
Parameter	Parameter Type	Data Type	Description
Enable	Input	BOOL	Data logging write enable. TRUE - Rising Edge Enable detected, start data logging operation when previous instruction operations are complete. FALSE - Rising Edge not detected.
TSEnable	Input	BOOL	TRUE - Date and time stamp logging enable flag.
CfgID	Input	USINT	Data logging configuration VA ID number from 1-10.
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.
Status	Output	USINT	Current status of the instruction. Data logging Status codes: • 0 - Idle • 1 - Doing • 2 - Succeed, indicates data logging is complete. • 3 - Error, indicates data logging completed with error.

ErrorID	Output	UDINT	A unique numeric error code for DLG.

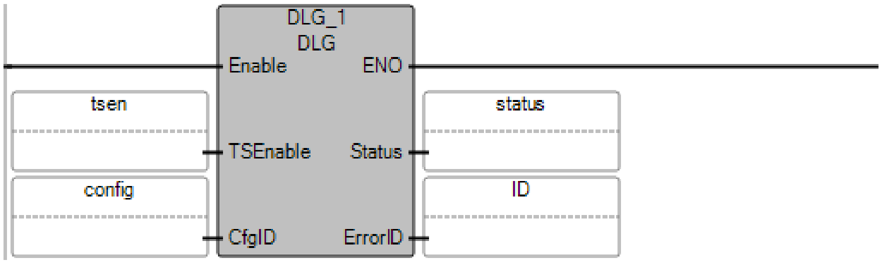
DLG error codes

Error code	Error Name	Comments
0	DLG_ERR_NONE	No error.
1	DLG_ERR_NO_SDCARD	SD card is absent.
2	DLG_ERR_RESERVED	Reserved.
3	DLG_ERR_DATAFILE_ACCESS	Access Data logging file error.
4	DLG_ERR_CFG_ABSENT	Data logging configuration file is absent.
5	DLG_ERR_CFG_ID	Configure ID is absent in data logging configuration file
6	DLG_ERR_RESOURCE_BUSY	The Data logging operation linked to this Data logging ID is used by another FB operation.
7	DLG_ERR_CFG_FORMAT	Data logging configuration file format is invalid.
8	DLG_ERR_RTC	Real time clock is invalid.
9	DLG_ERR_UNKNOWN	Unspecified error has occurred.

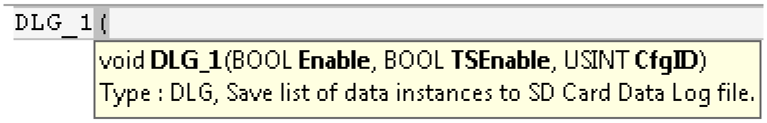
DLG Function Block Diagram example



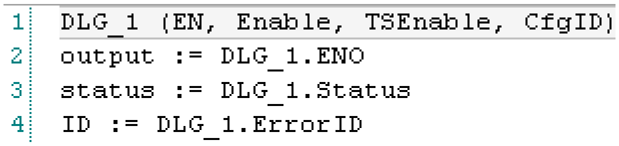
DLG Ladder Diagram example



DLG Structured Text example



```
DLG_1(
void DLG_1(BOOL Enable, BOOL TSEnable, USINT CfgID)
Type : DLG, Save list of data instances to SD Card Data Log file.
```



```
DLG_1 (EN, Enable, TSEnable, CfgID)
output := DLG_1.ENO
status := DLG_1.Status
ID := DLG_1.ErrorID
```

IIM (immediate input)

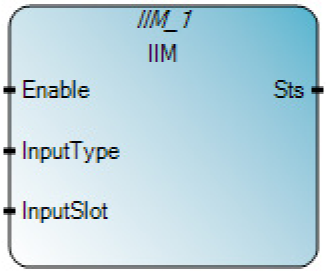
Update inputs prior to normal output scan.

Operation details:

- Typically used at the beginning of an interrupt program to select or mask inputs that are immediately scanned to get the current inputs.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instructions applies to the Micro830, Micro850, and Micro870 controllers.

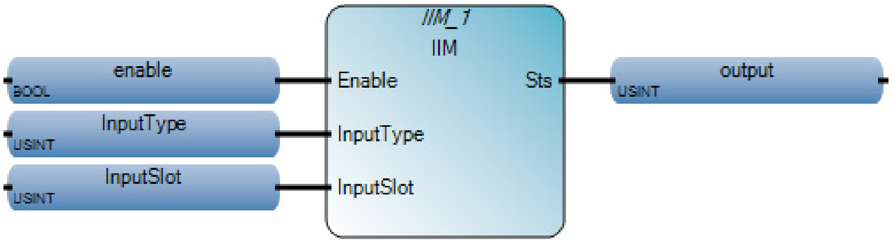


Use this table to help determine the parameter values for this instruction.

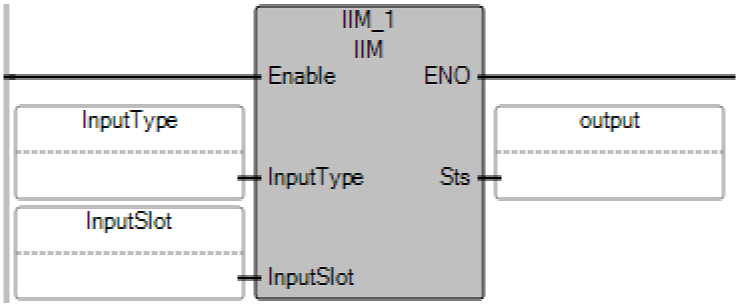
Parameter	Parameter Type	Data Type	Description
Enable	Input	BOOL	Instruction block enable. TRUE - execute instruction block. FALSE - do not execute.
InputType	Input	USINT	Identifies the type of input: 0 - Embedded input. 1 - Plug-in input.
InputSlot	Input	USINT	Identifies the input slot. 0 - Embedded input 1,2,3,4,5 - Plug-in slot number. (Slots are numbered from left to right, starting with number 1.) For embedded input, always 0. For Plug-in input, input slot is 1,2,3,4,5 (Plug-in slot number, starting with left-most slot = 1).
Sts	Output	USINT	Immediate input scan status. IIM status (Sts) codes: • 0x00 - Not enabled (no action taken). • 0x01 - Input/output scan success. • 0x02 - Input/output type invalid. • 0x03 - Input/output slot invalid.

ENO	Output	BOOL	Enable output. TRUE - Input updated. FALSE - Input not updated. Applies only to Ladder Diagram programs.

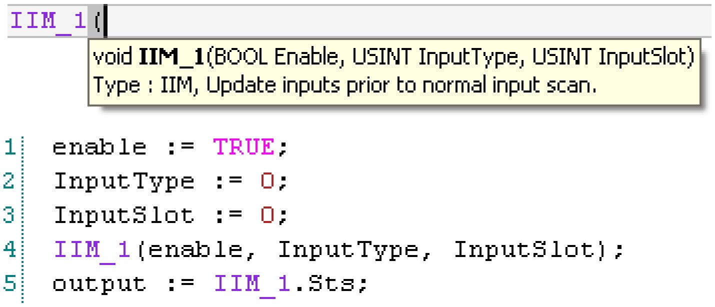
IIM Function Block Diagram example



IIM Ladder Diagram example



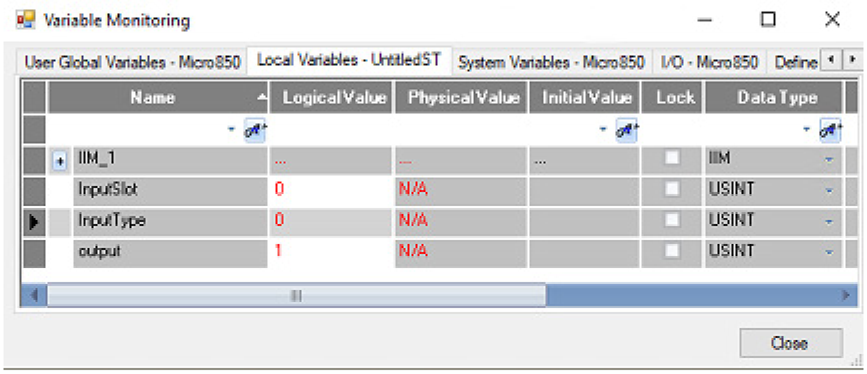
IIM Structured Text example



```
IIM_1(
void IIM_1(BOOL Enable, USINT InputType, USINT InputSlot)
Type : IIM, Update inputs prior to normal input scan.
```

```
enable := TRUE;
InputType := 0;
InputSlot := 0;
IIM_1(enable, InputType, InputSlot);
output := IIM_1.Sts;
```

Results



Variable Monitoring

User Global Variables - Micro850 | Local Variables - UntitledST | System Variables - Micro850 | I/O - Micro850 | Define

Name	Logical Value	Physical Value	Initial Value	Lock	Data Type
IIM_1	...	...	...	□	IIM
InputSlot	0	N/A		□	USINT
InputType	0	N/A		□	USINT
output	1	N/A		□	USINT

IOM (immediate output)

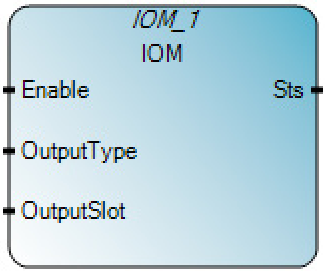
Update outputs prior to normal output scan.

Operation details:

- Typically used at the end of an interrupt program to select or mask which outputs are immediately scanned and updated.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instructions applies to the Micro830, Micro850, and Micro870 controllers.

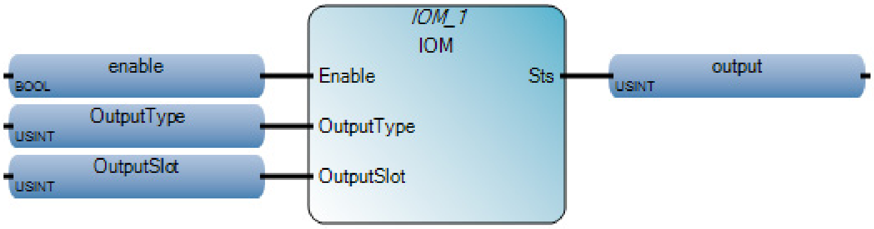


Use this table to help determine the parameter values for this instruction.

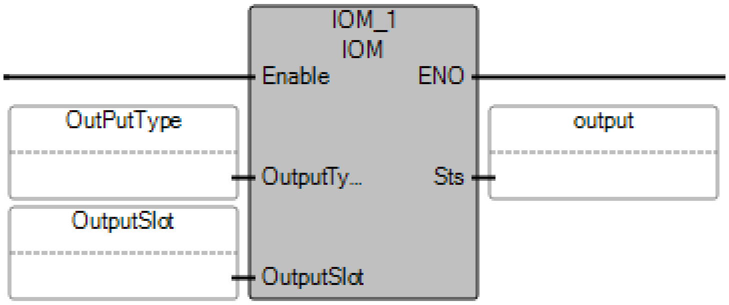
Parameter	Parameter Type	Data Type	Description
Enable	Input	BOOL	Instruction block enable. TRUE - execute instruction. FALSE - do not execute.
OutputType	Input	USINT	Identifies the type of output: 0 - Embedded output. 1 - Plug-in output.
OutputSlot	Input	USINT	Identifies the output slot: 0 - Embedded output 1,2,3,4,5 - Plug-in slot number. (Slots are numbered from left to right, starting with number 1.) For embedded output, always 0. For Plug-in output, output slot is 1,2,3,4,5 (Plug-in slot number, starting with left-most slot = 1).

Sts	Output	USINT	Immediate output scan status. IOM (Sts) status codes: • 0x00 - Not enabled (no action taken). • 0x01 - Input/output scan success. • 0x02 - Input/output type invalid. • 0x03 - Input/output slot invalid.
ENO	Output	BOOL	Enable output. TRUE - Output updated. FALSE - Output not updated. Applies only to Ladder Diagram programs.

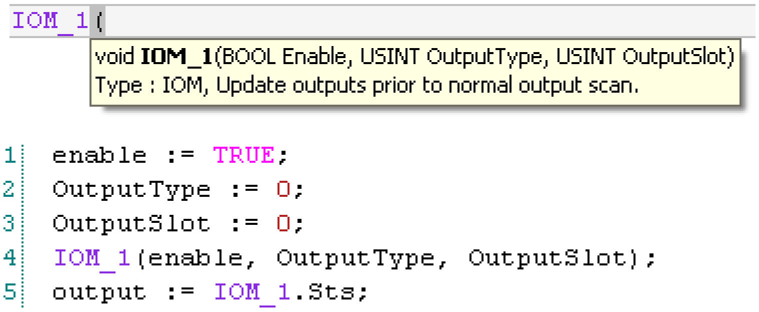
IOM Function Block Diagram example



IOM Ladder Diagram example



IOM Structured Text example

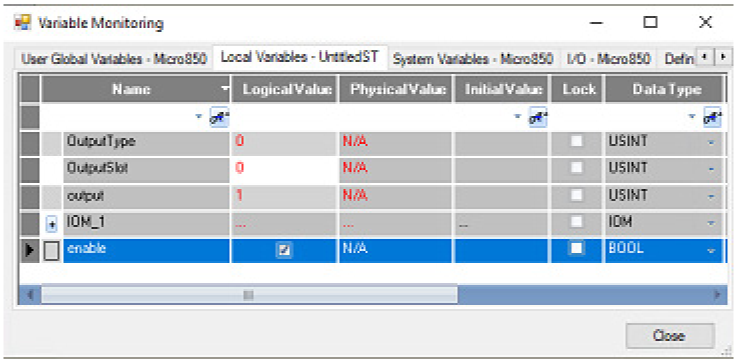


```
IOM_1(
```

void **IOM_1**(BOOL Enable, USINT OutputType, USINT OutputSlot)
Type : IOM, Update outputs prior to normal output scan.

```
enable := TRUE;
OutputType := 0;
OutputSlot := 0;
IOM_1(enable, OutputType, OutputSlot);
output := IOM_1.Sts;
```

Results


Variable Monitoring
User Global Variables - Micro850
Local Variables - UntitledST
System Variables - Micro850
I/O - Micro850
Defin
Close

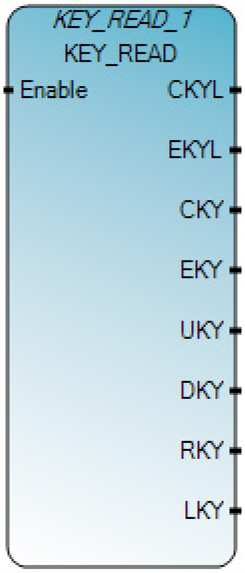
Name	LogicalValue	PhysicalValue	InitialValue	Lock	Data Type
OutputType	0	N/A		□	USINT
OutputSlot	0	N/A		□	USINT
output	1	N/A		□	USINT
IOM_1	...	...	...	□	IOM
enable	☑	N/A		□	BOOL

KEY_READ (read keys on LCD)

Reads the Key status on the optional LCD module when the user display is active.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

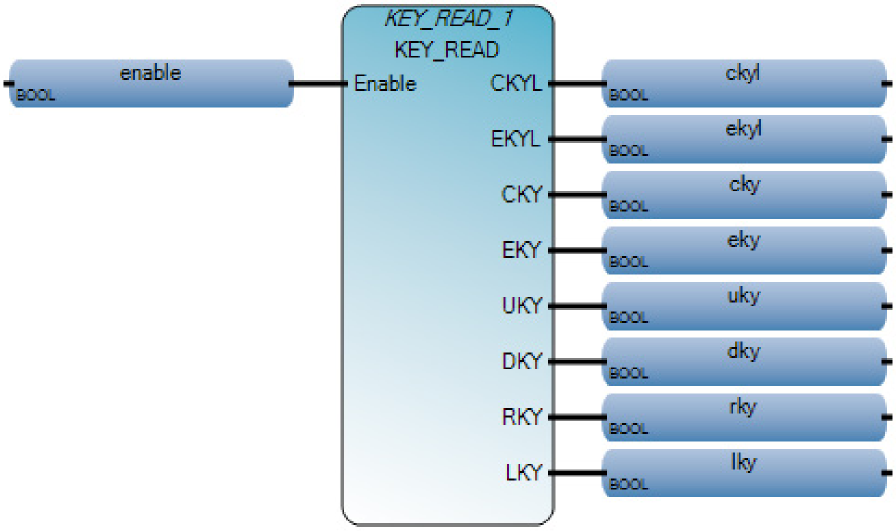
This instruction applies to the Micro810 controller.



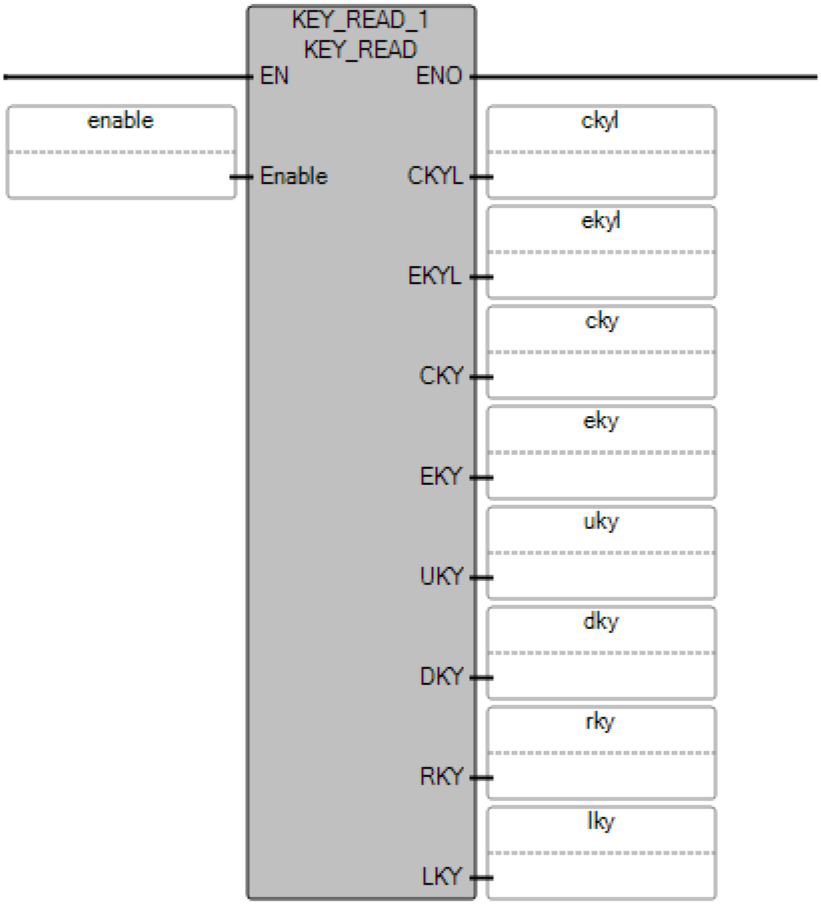
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
Enable	Input	BOOL	Enables the instruction block. TRUE - enable Read Keys on the Remote LCD keypad. FALSE - disable Read Keys on the Remote LCD keypad.
CKYL	Output	BOOL	TRUE: ESC key pressed for more than 2 seconds.
EKYL	Output	BOOL	TRUE: OK key pressed for more than 2 seconds.
CKY	Output	BOOL	TRUE: ESC key pressed.
EKY	Output	BOOL	TRUE: OK key pressed.
UKY	Output	BOOL	TRUE: Up key pressed.
DKY	Output	BOOL	TRUE: Down key pressed.
LKY	Output	BOOL	TRUE: Left key pressed.
RKY	Output	BOOL	TRUE: Right key pressed.

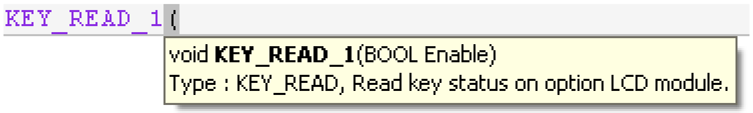
KEY_READ Function Block Diagram example



KEY_READ Ladder Diagram example

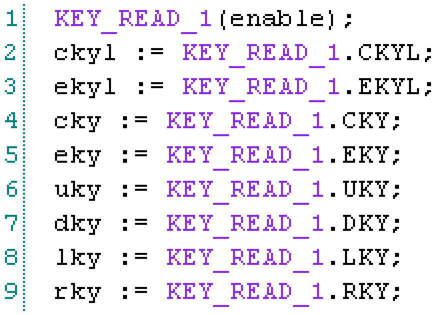


KEY_READ Structured Text example



```
KEY_READ_1(
```

```
void KEY_READ_1(BOOL Enable)
Type : KEY_READ, Read key status on option LCD module.
```



```
KEY_READ_1(enable);
ckyl := KEY_READ_1.CKYL;
ekyl := KEY_READ_1.EKYL;
cky := KEY_READ_1.CKY;
eky := KEY_READ_1.EKY;
uky := KEY_READ_1.UKY;
dky := KEY_READ_1.DKY;
lky := KEY_READ_1.LKY;
rky := KEY_READ_1.RKY;
```

(* ST Equivalence: *)

```
KEY_READ_1(KEYENABLE) ;
KEY_EKYL := KEY_READ_1.EKYL ;
KEY_CKY := KEY_READ_1.CKY ;
KEY_EKY := KEY_READ_1.EKY ;
KEY_UKY := KEY_READ_1.UKY ;
KEY_DKY := KEY_READ_1.DKY ;
KEY_RKY := KEY_READ_1.RKY ;
KEY_LKY := KEY_READ_1.LKY ;
```

KEY_READ_REM (read keys for remote LCD)

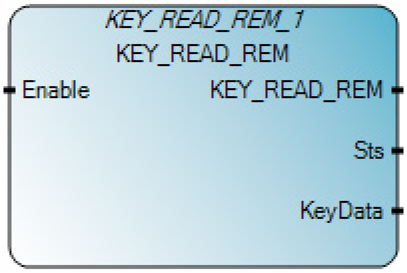
Reads the Key status on the optional Remote LCD module when the user display is active.

Operation details:

- Use the LCD_REM instruction to activate the user display on the Remote LCD module. If the user display is not active, an error occurs during KEY_READ_REM execution.
- P-BUTTON property in LCD Function File activates; otherwise all key status is FALSE.
- Only single key presses are supported for the KEY_READ_REM instruction; two-key press combinations are not supported.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820 controllers.



Use this table to help determine the parameter values for this instruction.

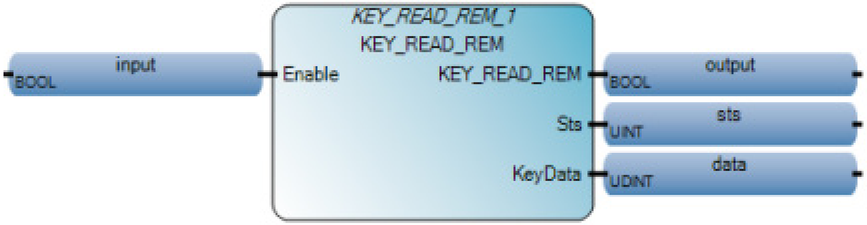
Parameter	Parameter Type	Data Type	Description
Enable	Input	BOOL	Instruction block enable. TRUE - Enable FALSE - Disable
KEY_READ_REM	Output	BOOL	TRUE - Remote LCD Key data is read successfully. FALSE - Enable is false, there is an error reading. Remote LCD Key Data or User Display is not active.
Sts	Output	UINT	Status of the KEY_READ_REM operation. KEY_READ_REM status codes: • 0 - Enable Input is False. • 1 - Key data read successfully. • 2 - Remote LCD not detected. May occur when: • Remote LCD is not physically connected to the controller (or the wiring is incorrect). • Serial port settings are other than what is required for the Remote LCD. • 3 - Connection Error. May occur when there is an internal state machine error. Possible cause, an incompatibility between Controller FW version and RLCD FW version. • 4 - User Display is not active. • 5-65535 - Reserved.
KeyData	Output	UDINT	Remote LCD KeyPad Data. KeyData definitions are defined in KeyData bitfields table.

KeyData bitfields table

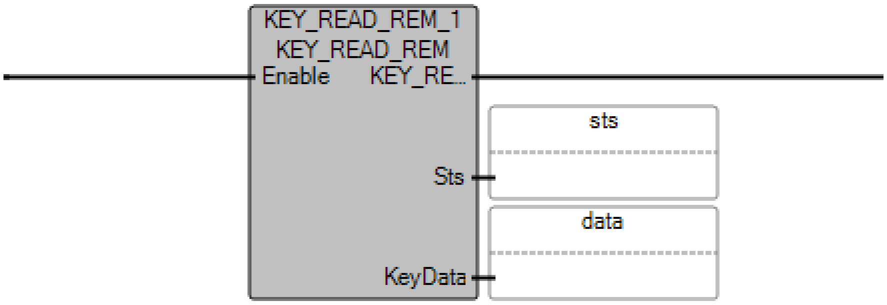
Use this table to help determine the KeyData bitfields for KEY_READ_REM.

Bit No. in KeyData	Name	Parameter Description
0	UKY	TRUE = Up key pressed.
1	DKY	TRUE = Down key pressed.
2	LKY	TRUE = Left key pressed.
3	RKY	TRUE = Right key pressed.
4	F1KY	TRUE = F1 key pressed.
5	F2KY	TRUE = F2 key pressed.
6	F3KY	TRUE = F3 key pressed.
7	F4KY	TRUE = F4 key pressed.
8	F5KY	TRUE = F5 key pressed.
9	F6KY	TRUE = F6 key pressed.
10	EKY	TRUE = Enter key pressed.
11	CKY	TRUE = Cancel key pressed.
12	EKYL	TRUE = Enter key pressed for more than 2 seconds.
13	CKYL	TRUE = Cancel key pressed for more than 2 seconds.
14-31	--	Reserved.

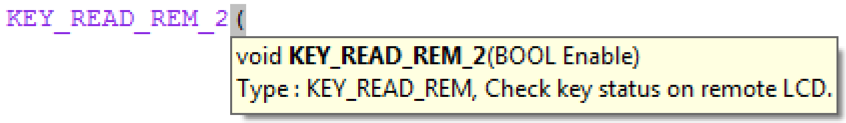
KEY_READ_REM Function Block Diagram example



KEY_READ_REM Ladder Diagram example

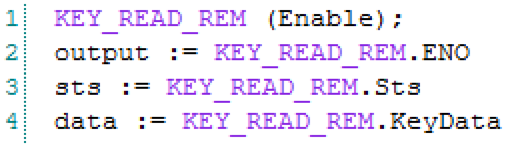


KEY_READ_REM Structured Text example



```
KEY_READ_REM_2(
```

void **KEY_READ_REM_2**(BOOL Enable)
Type : KEY_READ_REM, Check key status on remote LCD.



```
KEY_READ_REM (Enable);
output := KEY_READ_REM.ENO
sts := KEY_READ_REM.Sts
data := KEY_READ_REM.KeyData
```

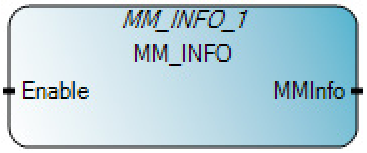
MM_INFO (memory module information)

Reads Memory Module header information. When a Memory Module is not present, all values return zero (0).

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.

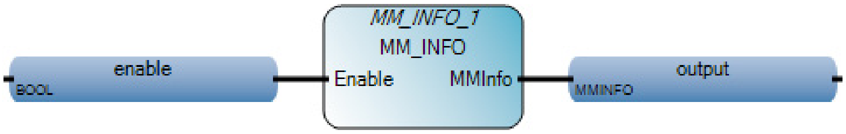
The outputs are always reset when the instruction is applied to the simulated controller (2080-LC50-48QWB-SIM).



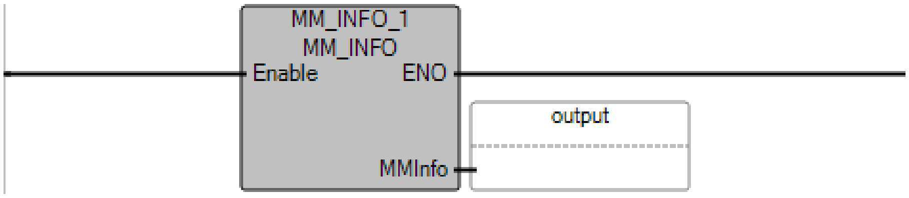
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
Enable	Input	BOOL	Instruction block enable. TRUE - read Memory Module header information. FALSE - there is no read operation, and the output Memory Module information is invalid.
MMInfo	Output	MMINFO	Memory Module Information is defined in the MMINFO data type on page 347.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.

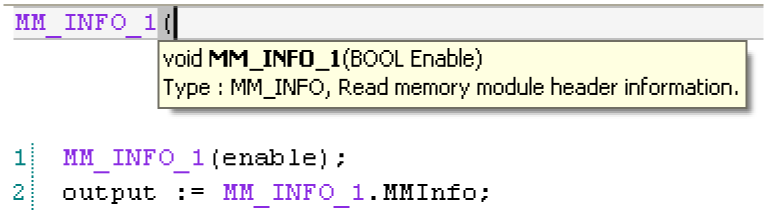
MM_INFO Function Block Diagram example



MM_INFO Ladder Diagram example



MM_INFO Structured Text example



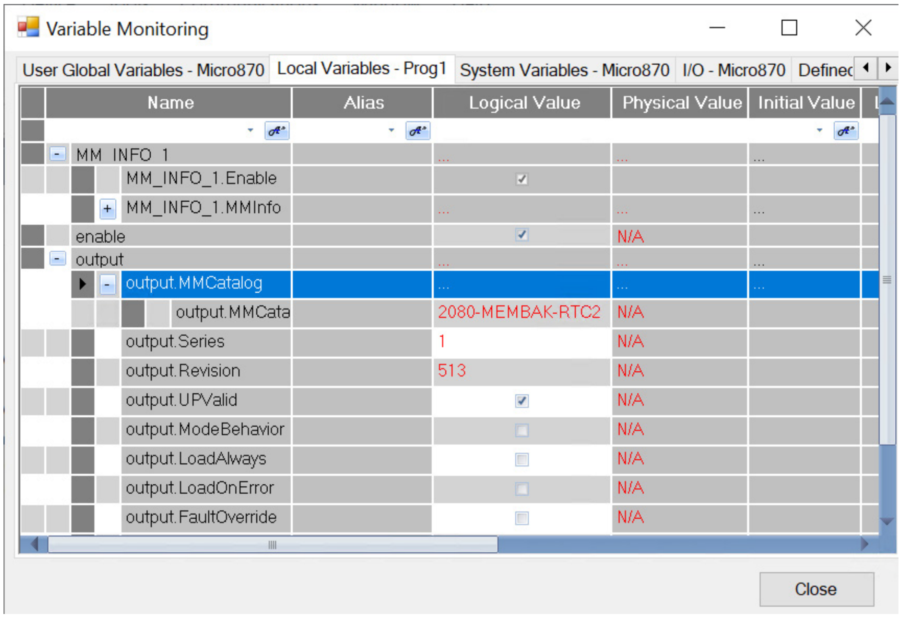
```
MM_INFO_1(
```

void **MM_INFO_1**(BOOL Enable)
Type : MM_INFO, Read memory module header information.

```
MM_INFO_1(enable);
output := MM_INFO_1.MMInfo;
```

Results

For controllers using 2080-MEMBAK-RTC:

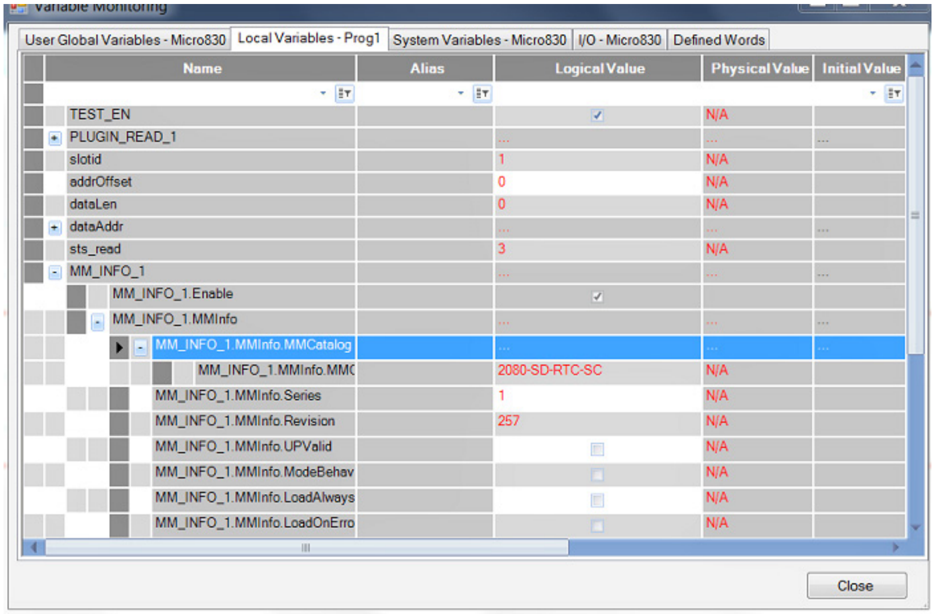


Variable Monitoring

User Global Variables - Micro870 | Local Variables - Prog1 | System Variables - Micro870 | I/O - Micro870 | Definec

Name	Alias	Logical Value	Physical Value	Initial Value
MM_INFO_1		...	...	...
MM_INFO_1.Enable		☑		
MM_INFO_1.MMInfo		...	...	...
enable		☑	N/A	
output		...	...	...
output.MMCatalog		...	...	...
output.MMCata		2080-MEMBAK-RTC2	N/A	
output.Series		1	N/A	
output.Revision		513	N/A	
output.UPValid		☑	N/A	
output.ModeBehavior		☐	N/A	
output.LoadAlways		☐	N/A	
output.LoadOnError		☐	N/A	
output.FaultOverride		☐	N/A	

For controllers using an SD card:



Variable Monitoring

User Global Variables - Micro830 | Local Variables - Prog1 | System Variables - Micro830 | I/O - Micro830 | Defined Words

Name	Alias	Logical Value	Physical Value	Initial Value
TEST_EN		☑	N/A	
PLUGIN_READ_1		...	...	...
slotid		1	N/A	
addrOffset		0	N/A	
dataLen		0	N/A	
dataAddr		...	...	...
sts_read		3	N/A	
MM_INFO_1		...	...	...
MM_INFO_1.Enable		☑		
MM_INFO_1.MMInfo		...	...	...
MM_INFO_1.MMInfo.MMCatalog		...	...	...
MM_INFO_1.MMInfo.MMC		2080-SD-RTC-SC	N/A	
MM_INFO_1.MMInfo.Series		1	N/A	
MM_INFO_1.MMInfo.Revision		257	N/A	
MM_INFO_1.MMInfo.UPValid		☐	N/A	
MM_INFO_1.MMInfo.ModeBehav		☐	N/A	
MM_INFO_1.MMInfo.LoadAlways		☐	N/A	
MM_INFO_1.MMInfo.LoadOnErro		☐	N/A	

MMINFO data type

The following table describes the MMINFO data type parameters.

Parameter	Data type	Description
MMCatalog	MMCATNUM	The catalog number of the Memory Module. When using the MM_INFO on page 345 instruction on controllers with an SD card, the MMCatalog is "SD CARD".
Series	UINT	The series of the Memory Module. When using the MM_INFO instruction on controllers with an SD card, the series is 1.
Revision	UINT	The revision of the Memory Module. When using the MM_INFO on controllers with an SD card, the revision is 257.

Parameter	Data type	Description
UPValid	BOOL	User program is present (TRUE: possibly valid project is found). Note: Even if TRUE, there is still a possibility that the project will be detected during download or restore as invalid if individual files are missing or corrupt.
ModeBehavior	BOOL	Mode behavior (TRUE: Go to RUN on power up).
LoadAlways	BOOL	Memory Module restore to controller always on power up.
LoadOnError	BOOL	Memory Module restore to controller if power up with error.
FaultOverride	BOOL	Override fault on power up.
MMPresent	BOOL	Memory Module is present.

MODULE_INFO

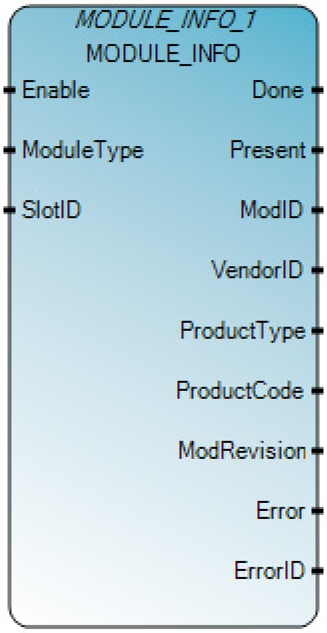
Reads module information from a plug-in module or an expansion module.

Operation details:

- Plug-in module information is read during RUN time.
- The 2080-MEMBAK-RTC memory plug-in module is not supported.
- Expansion module information is read when the module is powered on.
- When a plug-in or expansion module is not defined with a ModuleID, ProductType, or ProductCode, the MODULE_INFO operation returns 0 for the respective output parameter.
- The plug-in and expansion module identification information is defined by Allen-Bradley and is provided below as part of the MODULE_INFO description.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro830, Micro850, and Micro870 controllers. Expansion modules are only supported on the Micro850 and Micro870 controllers.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
Enable	Input	BOOL	Instruction block enable. TRUE - executes MODULE_INFO read operation. FALSE - does not execute the read operation. All output data values are reset to 0.
ModuleType	Input	USINT	Identifies the module type: • 1 - 2085 Expansion Module. • 2 - 2080 Plug-in Module.
SlotID	Input	USINT	The slot number where the plug-in or expansion module is located. Slot IDs are: 1, 2, 3, 4, 5 Slot 1 is on the far left.
Done	Output	BOOL	TRUE - Operation completed successfully. FALSE - Operation is not executing or an error condition occurred.
Present	Output	BOOL	Detects the plug-in or expansion module in the controller slot. TRUE - Module is physically present. FALSE - Module is not physically present.
ModID	Output	UINT	The identification for the module in the controller slot. • Plug-in modules are defined with a unique module identifier. • Expansion modules are not defined with a unique module identifier, ModID returns 0.
VendorID	Output	UINT	The plug-in or expansion module vendor ID. For Allen-Bradley products, the vendor ID is 1.
ProductType	Output	UINT	The plug-in or expansion module product type.
ProductCode	Output	UINT	The plug-in or expansion module product code.
ModRevision	Output	UINT	The plug-in or expansion module revision information.
Error	Output	BOOL	Indicates the existence of an error condition. TRUE - An error is detected. FALSE - No error.
ErrorID	Output	USINT	A unique numeric that identifies the error. The errors are defined in the MODULE_INFO error codes.

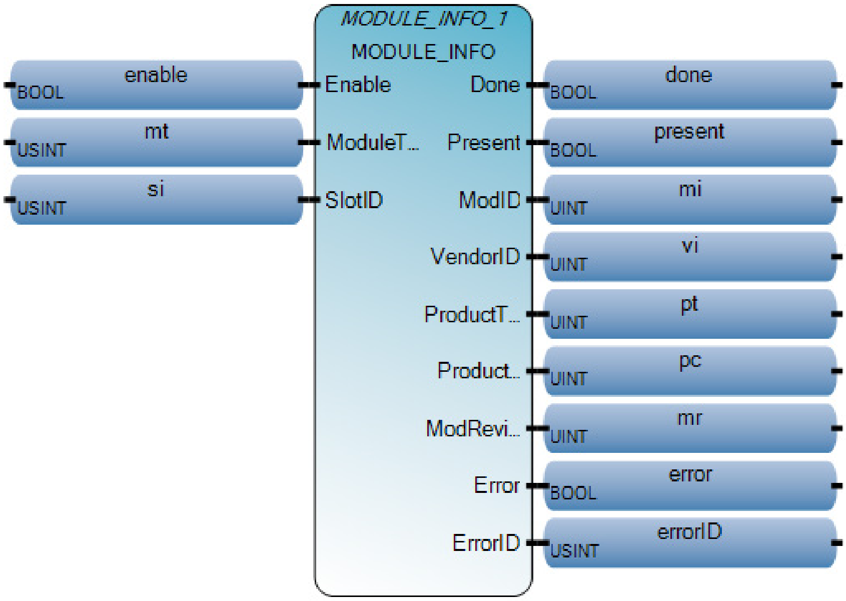
MODULE_INFO error codes

Use this table to determine the MODULE_INFO error codes and descriptions.

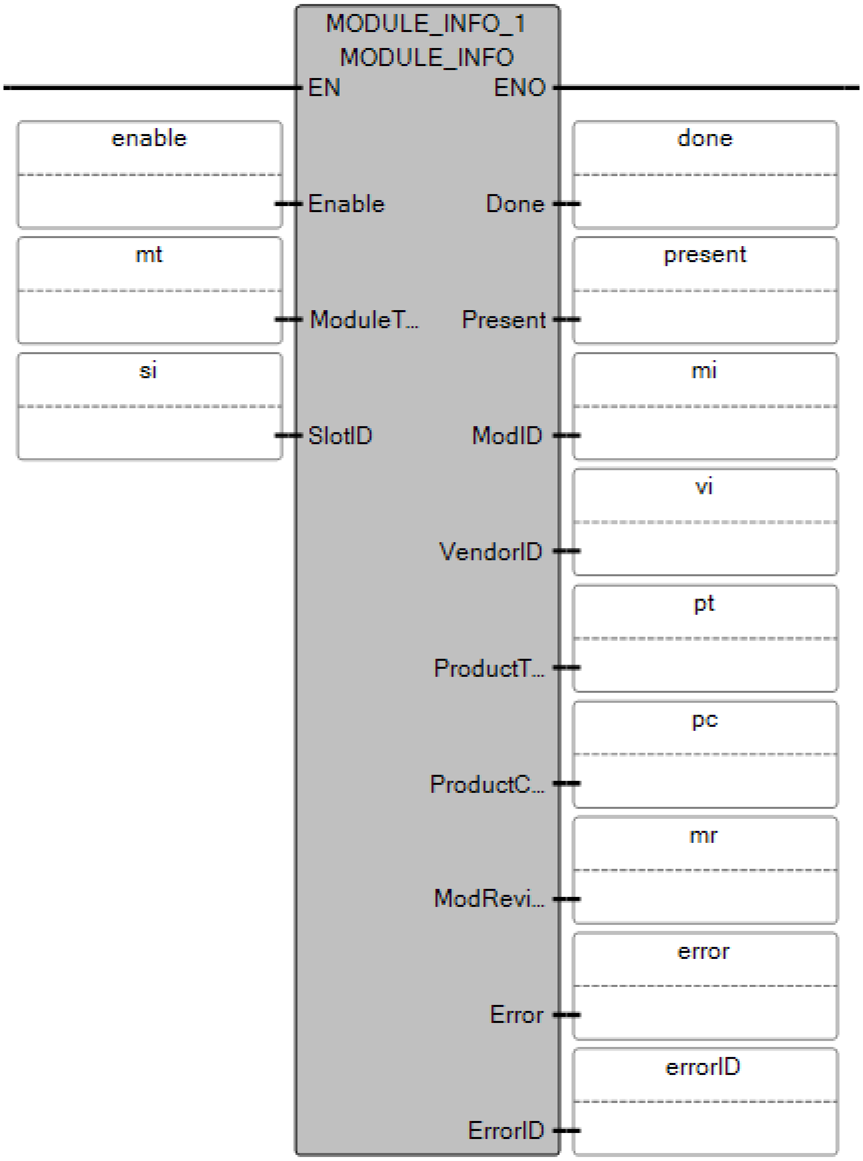
Error code	Error description
1	Invalid module type. Change to a valid module type. Valid module types are: • 1 - 2085 Expansion Module. Only supported on Micro850 controllers. • 2 - 2080 Plug-in Module. Supported on Micro820, Micro830, and Micro850 controllers.
2	Invalid slot number. Change to a valid slot number.
3	Invalid expansion module type.
4	Expansion module fatal error.
5	Plug-in module Read Info is not supported.
6	Plug-in module read error occurred while reading the Module ID.
7	Plug-in module read error occurred while reading the Vendor ID.
8	Plug-in module read error while reading the Product Type.

Error code	Error description
1	Invalid module type. Change to a valid module type. Valid module types are: • 1 - 2085 Expansion Module. Only supported on Micro850 controllers. • 2 - 2080 Plug-in Module. Supported on Micro820, Micro830, and Micro850 controllers.
2	Invalid slot number. Change to a valid slot number.
3	Invalid expansion module type.
4	Expansion module fatal error.
5	Plug-in module Read Info is not supported.
9	Plug-in module read error occurred while reading the Product Code.
10	Plug-in module read error occurred while reading the Module Revision.

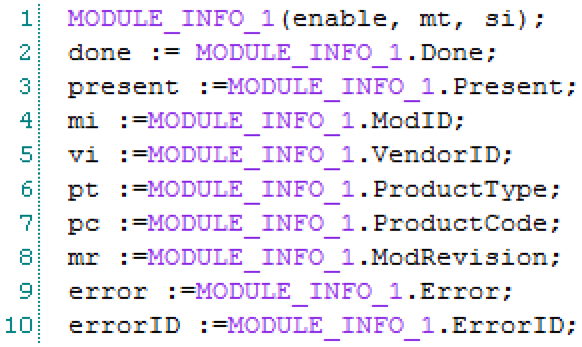
MODULE_INFO Function Block Diagram example



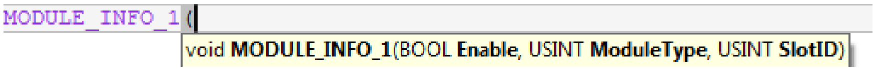
MODULE_INFO Ladder Diagram example



MODULE_INFO Structured Text example

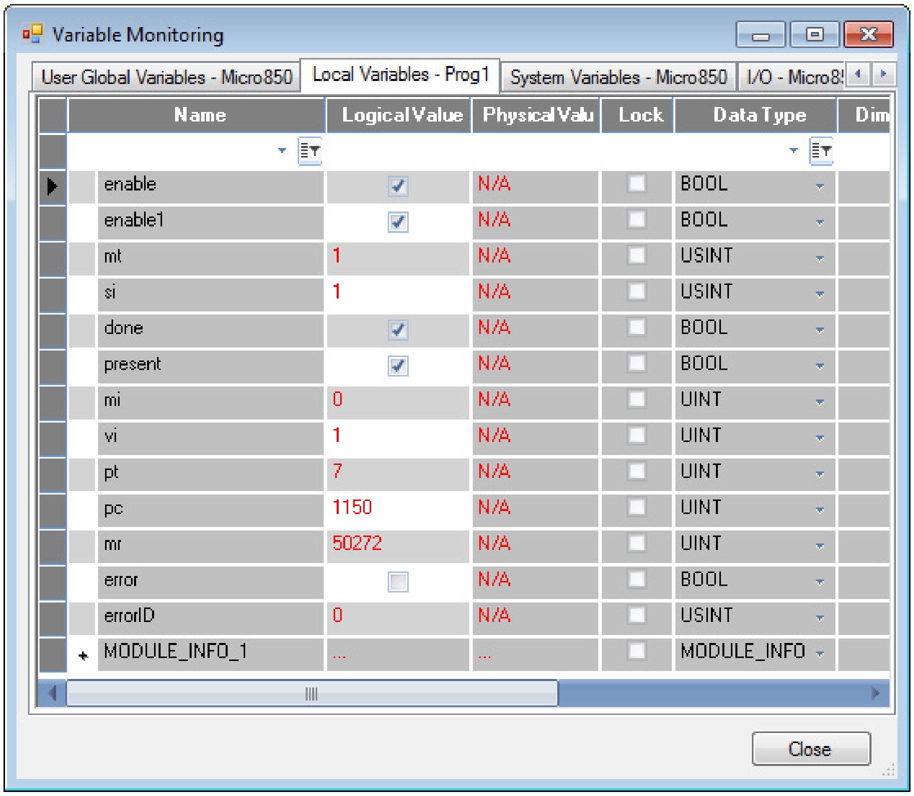


```
MODULE_INFO_1(enable, mt, si);
done := MODULE_INFO_1.Done;
present :=MODULE_INFO_1.Present;
mi :=MODULE_INFO_1.ModID;
vi :=MODULE_INFO_1.VendorID;
pt :=MODULE_INFO_1.ProductType;
pc :=MODULE_INFO_1.ProductCode;
mr :=MODULE_INFO_1.ModRevision;
error :=MODULE_INFO_1.Error;
errorID :=MODULE_INFO_1.ErrorID;
```



```
MODULE_INFO_1(
void MODULE_INFO_1(BOOL Enable, USINT ModuleType, USINT SlotID)
```

Results



Name	Logical Value	Physical Valu	Lock	Data Type	Dim
enable	☑	N/A	□	BOOL	
enable1	☑	N/A	□	BOOL	
mt	1	N/A	□	USINT	
si	1	N/A	□	USINT	
done	☑	N/A	□	BOOL	
present	☑	N/A	□	BOOL	
mi	0	N/A	□	UINT	
vi	1	N/A	□	UINT	
pt	7	N/A	□	UINT	
pc	1150	N/A	□	UINT	
mr	50272	N/A	□	UINT	
error	□	N/A	□	BOOL	
errorID	0	N/A	□	USINT	
+ MODULE_INFO_1	...	...	□	MODULE_INFO	

MODULE_INFO - plug-in and expansion module information

The following information provides the plug-in and expansion module Type, Module ID, Vendor ID, Product Codes and the Revision Word descriptions for expansion modules as defined by Allen-Bradley.

Plug-in module information

Use this table to determine the plug-in module information defined by Allen-Bradley.

Plug-in Module	Plug-in Type	Module ID	Vendor ID	Product Type	Product Code
2080-IF2	Analog	96	1	10	32
2080-IF4	Analog	98	1	10	33
2080-OF2	Analog	100	1	10	34
2080-TC2	Analog	102	1	10	35
2080-RTD2	Analog	104	1	10	36
2080-DNET20	Communication	34	1	12	249
2080-SERIALISOL	Communication	32	1	-	-
2080-IQ4	Digital	192	1	7	192
2080-OB4	Digital	193	1	7	193
2080-OV4	Digital	194	1	7	194
2080-IQ4OB4	Digital	195	1	7	195
2080-IQ4OV4	Digital	196	1	7	196
2080-OW4I	Digital	197	1	7	197
2080-MOT-HSC	Specialty	48	1	43	48

2080-TRIMPOT6	Specialty	72	1	-	-

Expansion module information

Use this table to determine the expansion module information defined by Allen-Bradley.

Model number	Expansion Type	Module ID	Vendor ID	Product Type	Product Code
2085-IF4	Analog	-	1	10	208
2085-IF8	Analog	-	1	10	206
2085-IRT4	Analog	-	1	10	213
2085-OF4	Analog	-	1	10	214
2085-IA8	Digital	-	1	7	1148
2085-IM8	Digital	-	1	7	1152
2085-IQ16	Digital	-	1	7	1144
2085-IQ32T	Digital	-	1	7	1145
2085-OA8	Digital	-	1	7	1149
2085-OB16	Digital	-	1	7	1146
2085-OV16	Digital	-	1	7	1147
2085-OW16	Digital	-	1	7	1151
2085-OW8	Digital	-	1	7	1150

Expansion module Revision Word descriptions

Use this table to determine the Revision Word information for Allen-Bradley expansion modules.

Bit	Name	Description
15 14	Max Baud Rate (1:0)	These bits identify the maximum frequency for Max Baud Rate (1:0). • 00 (bin) is 2 Mbps • 01 (bin) is 4 Mbps • 10 (bin) is 8 Mbps • 11 (bin) is 16 Mbps
13 12 11 10	Minor Revision (3:0)	Minor Revision (3:0) is the product minor revision designation. This field indicates the minor revision of the Catalog number designated by the Vendor ID, Product Type, Product Code, Series and Major Revision. The Minor Revision ranges from 0 to 15.
9 8 7 6 5	Major Revision (4:0)	Major Revision (4:0) is the product major revision designation. This field indicates the major revision of the Catalog number designated by the Vendor ID, Product Type, Product Code, and Series. The Major ranges from 0 to 31.

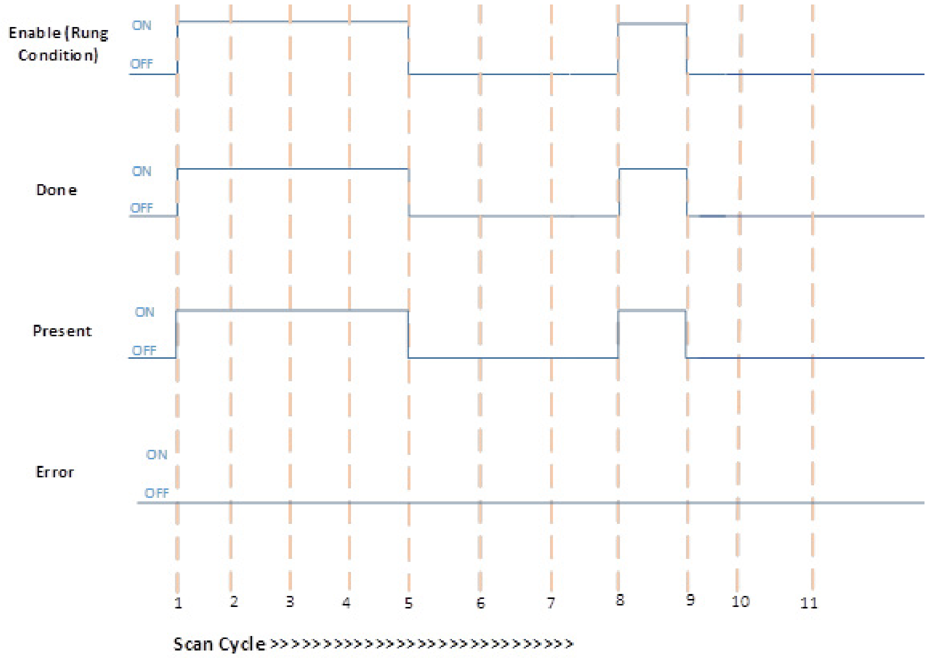
4	Series (4:0)	SERIES (4:0) is the product Series designation. This field indicates the Series letter of the Catalog number designated by the Vendor ID, Product Type, and Product Code.

Series (4:0)	Series	Series (4:0)	Series	Series (4:0)	Series	Series (4:0)	Series
0	A	8	I	16	Q	24	Y
1	B	9	J	17	R	25	Z
2	C	10	K	18	S	26	AA
3	D	11	L	19	T	27	AB
4	E	12	M	20	U	28	AC
5	F	13	N	21	V	29	AD
6	G	14	O	22	W	30	AE
7	H	15	P	23	X	31	>AE

MODULE_INFO instruction timing diagrams

The following timing diagram examples describe execution scenarios for the MODULE_INFO instruction on page 348.

Successful execution when a module is physically present

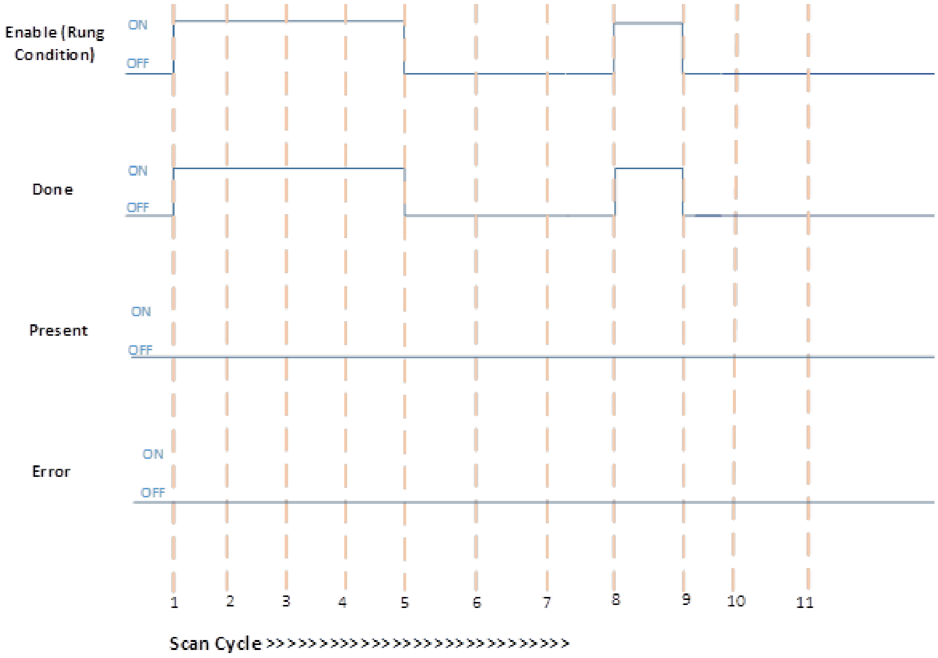


Use this table to help determine the MODULE_INFO parameter values for each scan cycle.

Scan Cycle	Description

Scan Cycle	Description
1, 8	Rung condition becomes TRUE when: • Enable input bit is TRUE. • ModuleType and SlotID are valid. A physical module is present. • Done and Present output bits are TRUE. • Error output bit is FALSE. • Update module related information for Module ID, Vendor ID, Product Type, Product Code, and Module Revision accordingly.
2,3,4	No change in rung condition.
5. 9	Rung condition becomes FALSE when Enable bit is FALSE. All output parameters are cleared.
6, 7, 10, 11	No change in rung condition. Enable bit is FALSE. All output parameters are cleared.

Successful execution when Module is not physically present

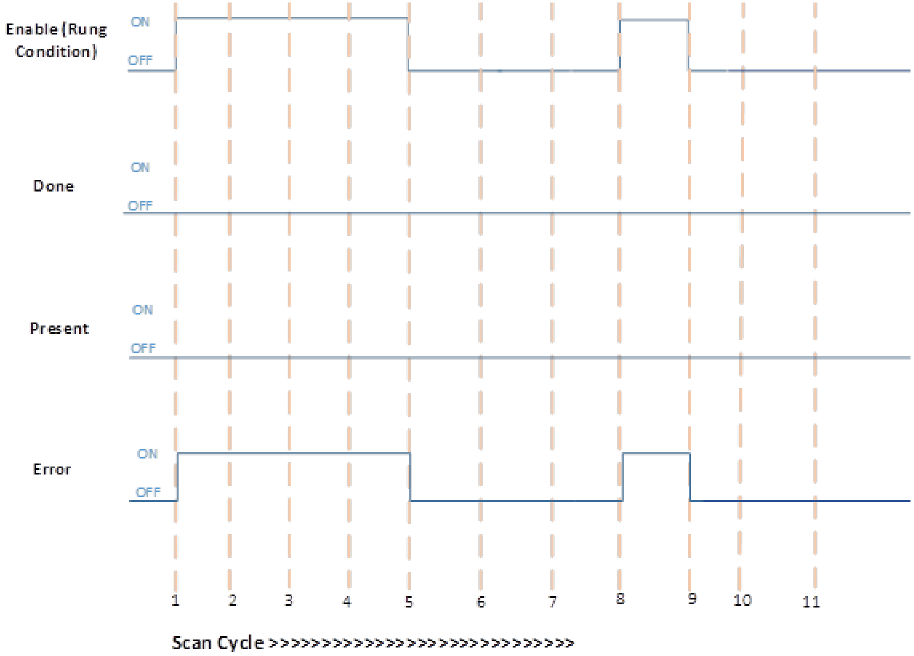


Use this table to help determine the MODULE_INFO parameter values for each scan cycle.

Scan Cycle	Description
1, 8	Rung condition becomes TRUE when: • Enable input bit is TRUE. • ModuleType and SlotID are valid. A physical module is not present. • Done output bit is TRUE. • Error and Present output bits are FALSE. • Update module related information for Module ID, Vendor ID, Product Type, Product Code, and Module Revision accordingly.
2,3,4	No change in rung condition. • Enable input bit is TRUE. • Input parameters are valid and a physical module is not present. • Update output parameters accordingly.

Scan Cycle	Description
5. 9	Rung condition becomes FALSE when: • Enable bit is FALSE. • All output parameters are cleared.
6, 7, 10, 11	No change in rung condition. Enable bit is FALSE. All output parameters are cleared.

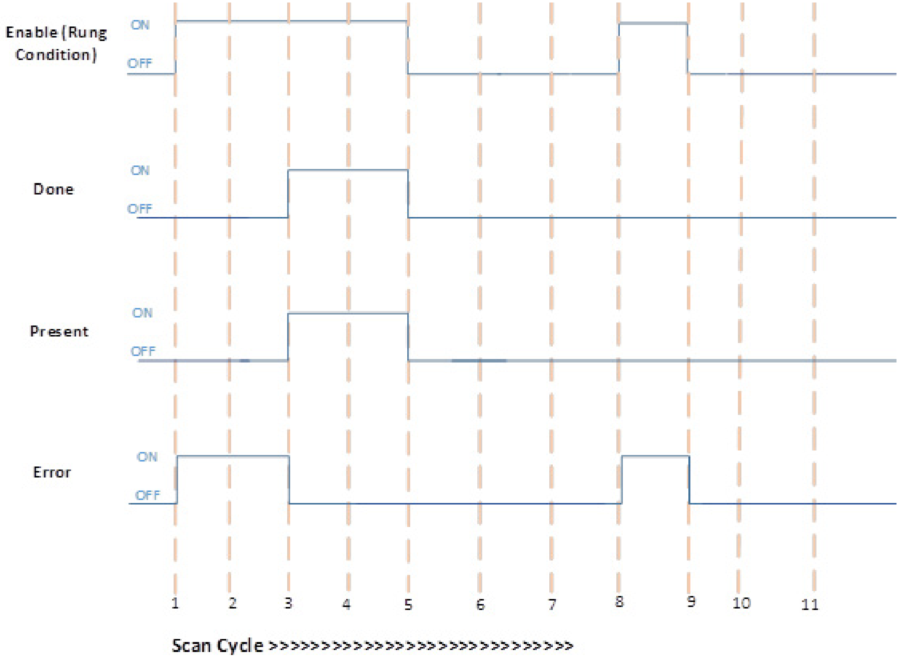
MODULE_INFO execution with Error



Use this table to help determine the MODULE_INFO parameter values for each scan cycle.

Scan Cycle	Description
1, 8	Rung condition becomes TRUE when: • Enable input bit is TRUE. • ModuleType and SlotID are valid. A physical module is not present. • Done and Present output bits are TRUE. • Error output bit is TRUE. • Module related information for Module ID, Vendor ID, Product Type, Product Code, and Module Revision is cleared.
2,3,4	No change in rung condition. • Enable input bit is TRUE. • ModuleType, SlotID, or both are invalid. • Update output parameters accordingly.
5. 9	Rung condition becomes FALSE when: • Enable bit is FALSE. • All output parameters are cleared.
6, 7, 10, 11	No change in rung condition. Enable bit is FALSE. All output parameters are cleared.

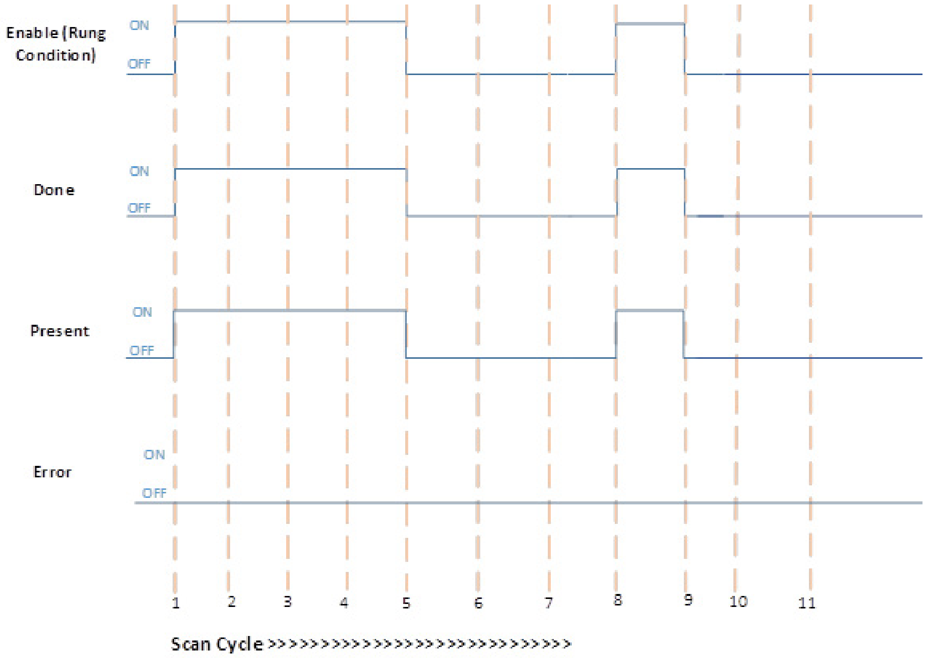
MODULE_INFO successful execution with error - no physical module



Use this table to help determine the MODULE_INFO parameter values for each scan cycle.

Scan Cycle	Description
1, 8	Rung condition becomes TRUE when: • Enable input bit is TRUE. • ModuleType or SlotID or both are invalid. A physical Module is not present. • Done and Present output bits are FALSE. • Error output bit is TRUE. • Module related information for Module ID, Vendor ID, Product Type, Product Code, and Module Revision is cleared.
2	No change in rung condition. • Enable input bit is TRUE. • ModuleType or SlotID or both are invalid. • Update output parameters accordingly.
3, 4	No change in rung condition. • Enable input bit is TRUE. • ModuleType and SlotID are both are valid. Module is physically present. • Done and Present output bits are TRUE. • Error output bit is FALSE. • Update module information for Module ID, Vendor ID, Product Type, Product Code, and Module Revision accordingly.
5. 9	Rung condition becomes FALSE when: • Enable bit is FALSE. • All output parameters are cleared.
6, 7, 10, 11	No change in rung condition. Enable bit is FALSE. All output parameters are cleared.

MODULE_INFO successful execution with error when physical module is present



Use this table to help determine the MODULE_INFO parameter values for each scan cycle.

Scan Cycle	Description
1, 8	Rung condition becomes TRUE when: • Enable input bit is TRUE. • ModuleType or SlotID or both are valid. Module is physically present. • Done and Present output bits are TRUE. • Error output bit is FALSE • Update module related information for Module ID, Vendor ID, Product Type, Product Code, and Module Revision accordingly.
2	No change in rung condition. • Enable input bit is TRUE. • Module input parameters are valid and module is physically present. • Update output parameters accordingly.
3, 4	No change in rung condition. • Enable input bit is TRUE. • ModuleType and SlotID are both are invalid. Module is physically present. • Done and Present output bits are FALSE. • Error output bit is TRUE and cleared. • Update module information for Module ID, Vendor ID, Product Type, Product Code, and Module Revision accordingly.
5. 9	Rung condition becomes FALSE when: • Enable bit is FALSE. • All output parameters are cleared.
6, 7, 10, 11	No change in rung condition. Enable bit is FALSE. All output parameters are cleared.

PLUGIN_INFO (plug-in information)

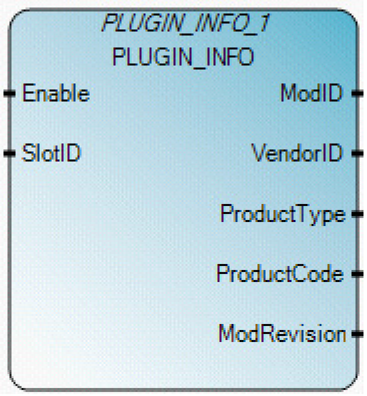
Read module information from a generic Plug-in or Expansion module.

Operation details:

- In Connected Components Workbench 10 or higher, the PLUGIN_INFO instruction can read any generic Plug-in or Expansion Module information, except the 2080-MEMBAK-RTC module.
- When a generic Plug-in or Expansion Module is not present, all values return to zero (0).

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro830, Micro850, and Micro870 controllers.

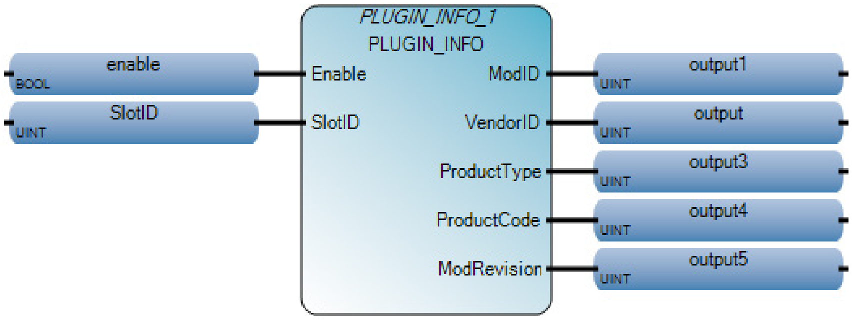


Use this table to help determine the parameter values for this instruction.

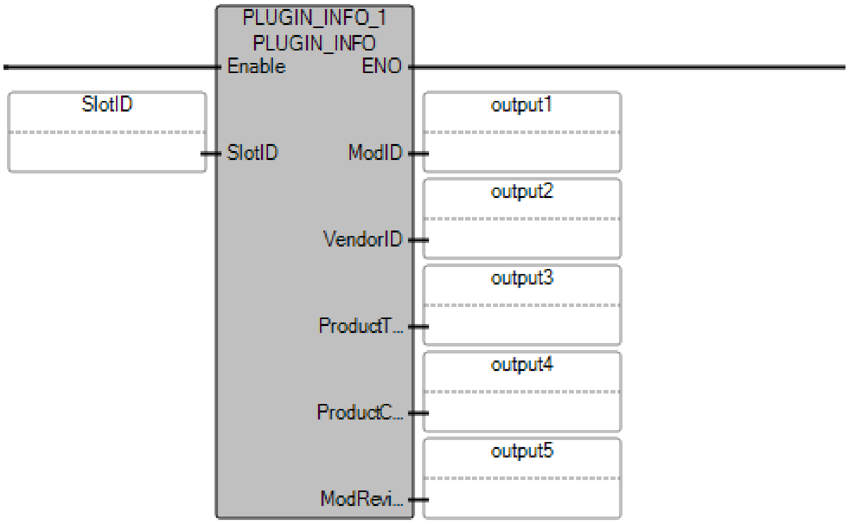
Parameter	Parameter Type	Data Type	Description
Enable	Input	BOOL	Instruction block enable. TRUE - execute Plug-in or Expansion module information read. FALSE - the instruction block is not executed. All output data values are reset to 0.
SlotID	Input	UINT	Plug-in Slot number: Slot ID = 1,2,3,4,5 (Starting with the first Slot from left = 1). Expansion Slot number: Slot ID = 101, 102, 103, 104 (Starting with the first Slot from left = 101).
ModID	Output	UINT	Plug-in Generic Module physical ID. • If the Expansion Module is not supported ModID = 0xFFF • If a Plug-in or Expansion Module is not present ModID = 0x0000
VendorID	Output	UINT	The Plug-in or Expansion Generic Module vendor ID. For Allen Bradley products, the vendor ID = 1. If a Plug-in or Expansion Module is not present VendorID = 0x0000
ProductType	Output	UINT	Plug-in or Expansion Generic Module product type. If a Plug-in or Expansion Module is not present ProductType = 0x0000

ProductCode	Output	UINT	Plug-in or Expansion Generic Module product code. If a Plug-in or Expansion Module is not present ProductCode = 0x0000
ModRevision	Output	UINT	Plug-in or Expansion Generic Module revision information. If a Plug-in or Expansion Module is not present ModRevision = 0x0000
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs. Ladder Diagram adds the ENO output automatically.

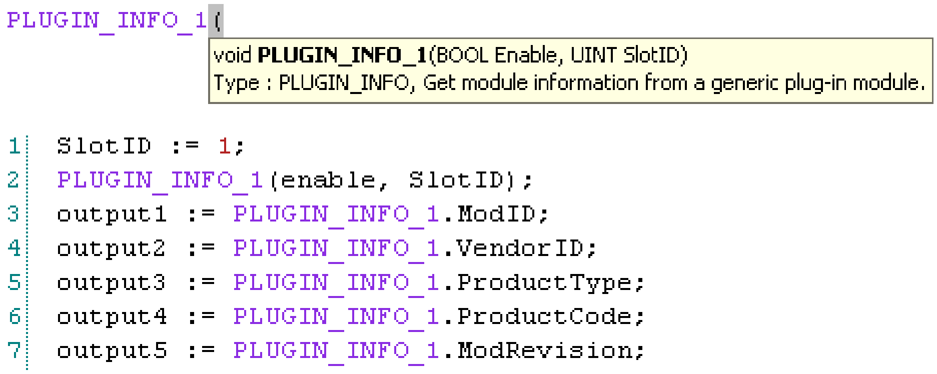
PLUGIN_INFO Function Block Diagram example



PLUGIN_INFO Ladder Diagram example



PLUGIN_INFO Structured Text example

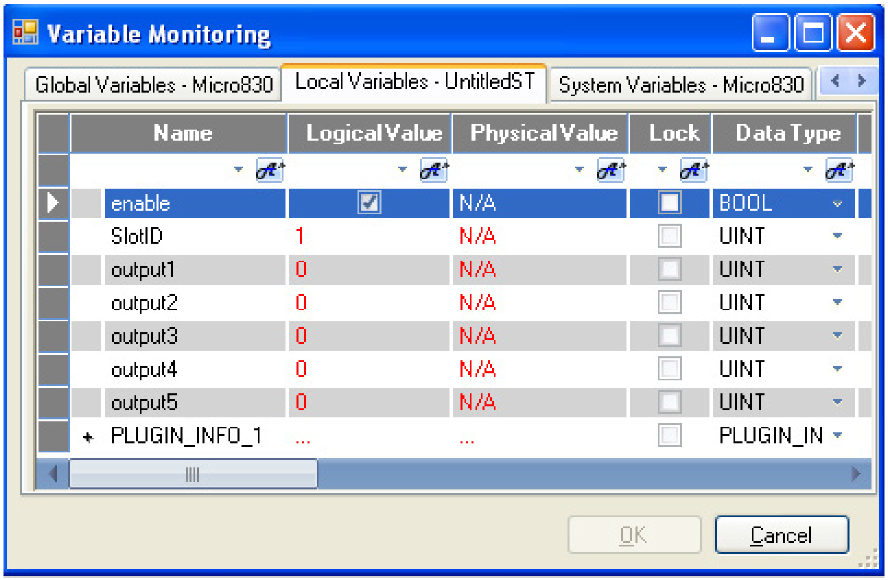


```
PLUGIN_INFO_1(
```

void **PLUGIN_INFO_1**(BOOL Enable, UINT SlotID)
Type : PLUGIN_INFO, Get module information from a generic plug-in module.

```
SlotID := 1;
PLUGIN_INFO_1(enable, SlotID);
output1 := PLUGIN_INFO_1.ModID;
output2 := PLUGIN_INFO_1.VendorID;
output3 := PLUGIN_INFO_1.ProductType;
output4 := PLUGIN_INFO_1.ProductCode;
output5 := PLUGIN_INFO_1.ModRevision;
```

Results



Variable Monitoring
Global Variables - Micro830
Local Variables - UntitledST
System Variables - Micro830
OK
Cancel

Name	Logical Value	Physical Value	Lock	Data Type
enable	☑	N/A	□	BOOL
SlotID	1	N/A	□	UINT
output1	0	N/A	□	UINT
output2	0	N/A	□	UINT
output3	0	N/A	□	UINT
output4	0	N/A	□	UINT
output5	0	N/A	□	UINT
+ PLUGIN_INFO_1	...	...	□	PLUGIN_IN

PLUGIN_READ (read plugin)

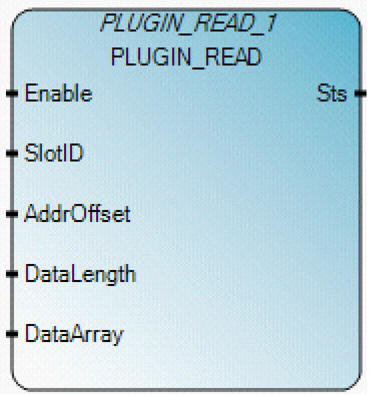
Reads data from a generic plug-in module.

Operation details:

- Any plug-in module except for 2080-MEMBAK-RTC modules.
- When a Plug-in Generic Module is not present, all values return to zero (0).

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

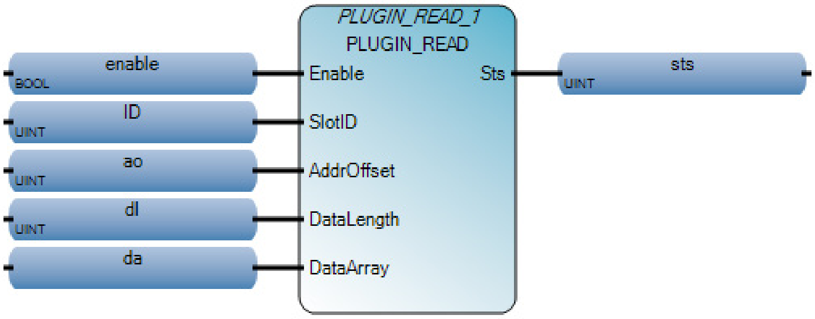
This instruction applies to the Micro820, Micro830, Micro850, and Micro870 controllers.



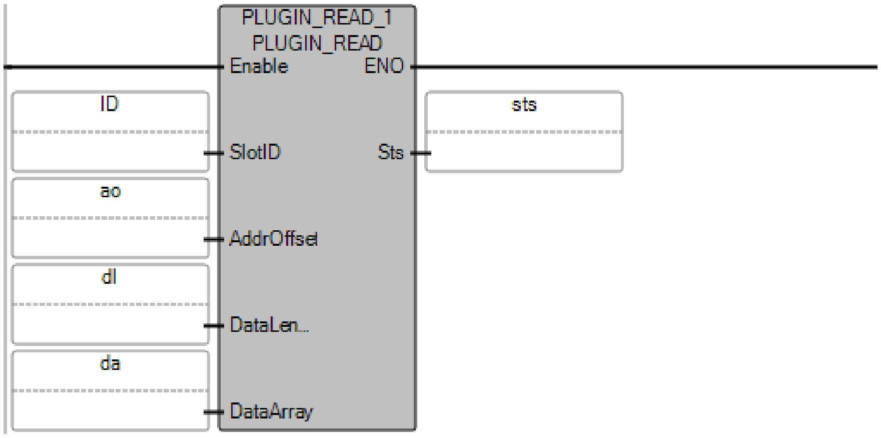
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
Enable	Input	BOOL	Instruction block enable. TRUE - execute UPM read. FALSE - there is no read operation and the data inside the data array is invalid.
SlotID	Input	UINT	Plug-in slot number. Slot ID = 1,2,3,4,5 (starting with the far left slot = 1).
Offset	Input	UINT	Address offset of the first data to be read, calculating from the first byte of the Plug-in Generic Module.
DataLength	Input	UINT	The number of bytes to be read.
DataArray	Input	USINT	An array used to store the data read from the Plug-in Generic Module.
Sts	Output	UINT	Status codes for PLUGIN_READ. Status (Sts) codes: • 0x00 - Function block not enabled (no operation). • 0x01 - Plug-in operation success. • 0x02 - Plug-in operation fails due to an invalid Slot ID. • 0x03 - Plug-in operation fails since it is not a valid Plug-in Generic module. • 0x04 - Plug-in operation fails due to data operated out of range. • 0x05 - Plug-in operation fails due to a data access parity error.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.

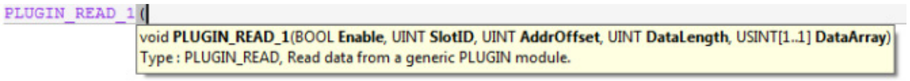
PLUGIN_READ Function Block Diagram example



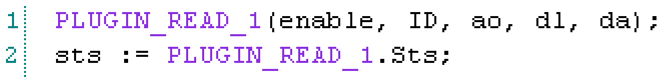
PLUGIN_READ Ladder Diagram example



PLUGIN_READ Structured Text example



```
PLUGIN_READ_1(
void PLUGIN_READ_1(BOOL Enable, UINT SlotID, UINT AddrOffset, UINT DataLength, USINT[1..1] DataArray)
Type : PLUGIN_READ, Read data from a generic PLUGIN module.
```



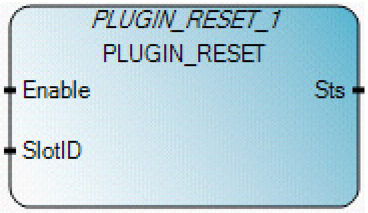
```
PLUGIN_READ_1(enable, ID, ao, dl, da);
sts := PLUGIN_READ_1.Sts;
```

PLUGIN_RESET (reset plugin)

Resets any Plug-in Generic Module hardware except 2080-MEMBAK-RTC modules. After the hardware reset, the Plug-in Generic Module is ready for configuration and operation.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro830, Micro850, and Micro870 controllers.

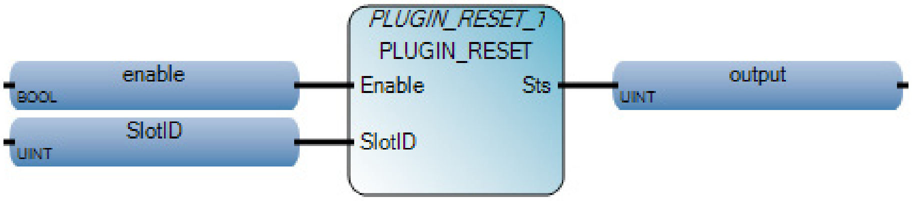


Use this table to help determine the parameter values for this instruction.

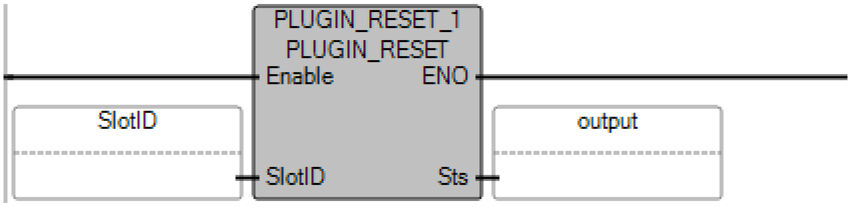
Parameter	Parameter Type	Data Type	Description
Enable	Input	BOOL	Instruction block enable. TRUE - execute Plug-in reset. FALSE - there is no reset operation.
SlotID	Input	UINT	Plug-in slot number. Slot ID = 1,2,3,4,5 (starting with the far left slot = 1).

Sts	Output	UINT	PLUGIN_RESET status codes. Status (Sts) codes: • 0x00 - Function block not enabled (no operation). • 0x01 - Plug-in operation success. • 0x02 - Plug-in operation fails due to an invalid Slot ID. • 0x03 - Plug-in operation fails since it is not a valid Plug-in Generic module. -2080-MOT-HSC module configuration is in High Speed Counter Instruction mode. • 0x04 - Plug-in operation fails due to data operated out of range. • 0x05 - Plug-in operation fails due to a data access parity error.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.

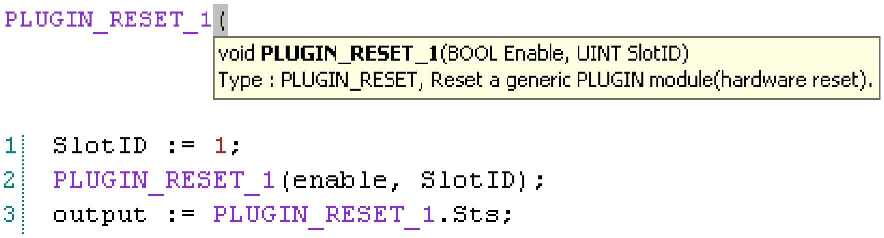
PLUGIN_RESET Function Block Diagram example



PLUGIN_RESET Ladder Diagram example



PLUGIN_RESET Structured Text example

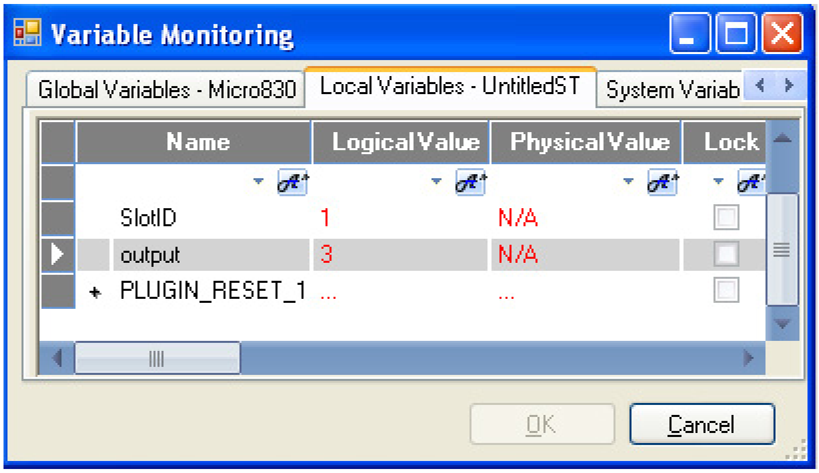


```
PLUGIN_RESET_1(
```

void **PLUGIN_RESET_1**(BOOL Enable, UINT SlotID)
Type : PLUGIN_RESET, Reset a generic PLUGIN module(hardware reset).

```
SlotID := 1;
PLUGIN_RESET_1(enable, SlotID);
output := PLUGIN_RESET_1.Sts;
```

Results



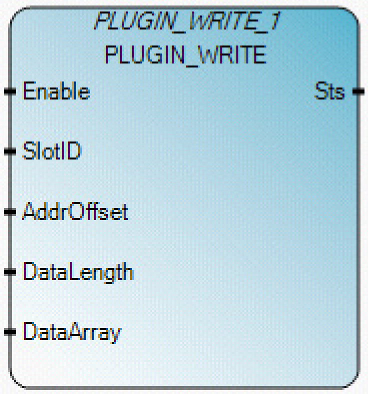
	Name	Logical Value	Physical Value	Lock
	SlotID	1	N/A	□
▶	output	3	N/A	□
	+ PLUGIN_RESET_1	...	...	□

PLUGIN_WRITE (write plugin)

Writes a block of data to any Plug-in Generic Module hardware except 2080-MEMBAK-RTC modules.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro830, Micro850, and Micro870 controllers.

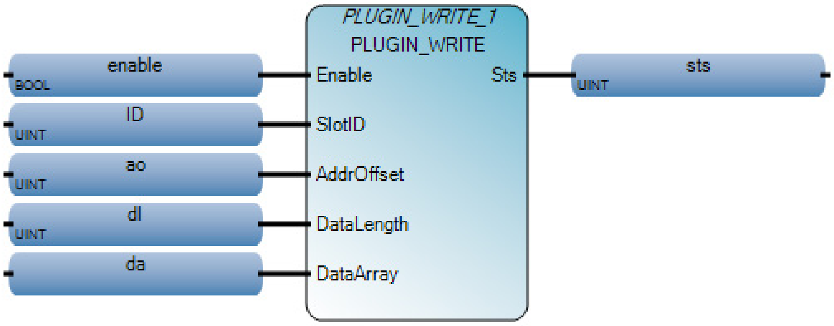


Use this table to help determine the parameter values for this instruction.

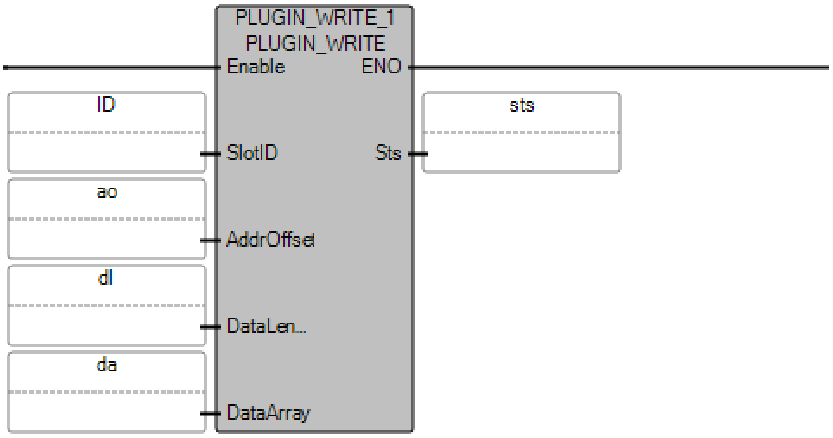
Parameter	Parameter Type	Data Type	Description
Enable	Input	BOOL	Instruction block enable. TRUE - execute Plug-in write. FALSE - there is no data write operation.
SlotID	Input	UINT	Plug-in slot number. Slot ID = 1,2,3,4,5 (starting with the far left slot = 1).
AddrOffset	Input	UINT	Address offset of the first data to be written, calculating from the first byte of the Plug-in Generic Module.
DataLength	Input	UINT	The number of bytes to be written.
DataArray	Input	USINT	Data to be written to the Plug-in Generic Module.

Sts	Output	UINT	PLUGIN_WRITE status codes. Status (Sts) codes: • 0x00 - Function block not enabled (no operation). • 0x01 - Plug-in operation success. • 0x02 - Plug-in operation fails due to an invalid Slot ID. • 0x03 - Plug-in operation fails since it is not a valid Plug-in Generic module. • 0x04 - Plug-in operation fails due to data operated out of range. • 0x05 - Plug-in operation fails due to a data access parity error.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.

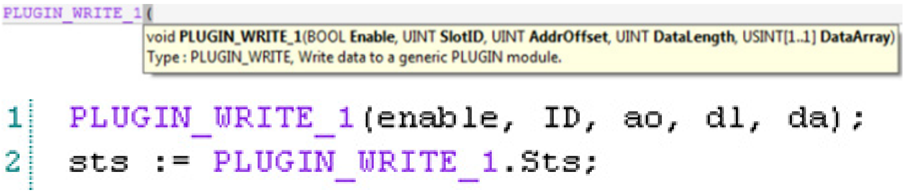
PLUGIN_WRITE Function Block Diagram example



PLUGIN_WRITE Ladder Diagram example



PLUGIN_WRITE Structured Text example



```
PLUGIN_WRITE_1(
void PLUGIN_WRITE_1(BOOL Enable, UINT SlotID, UINT AddrOffset, UINT DataLength, USINT[1..1] DataArray)
Type : PLUGIN_WRITE, Write data to a generic PLUGIN module.
```

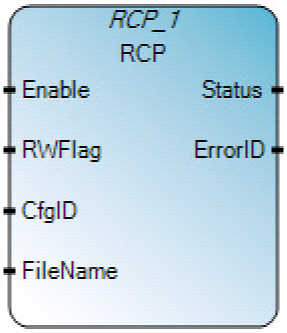
```
PLUGIN_WRITE_1(enable, ID, ao, dl, da);
sts := PLUGIN_WRITE_1.Sts;
```

RCP (recipe)

Reads the data value of a variable from the recipe data file that resides in the recipe data file folder in the SD card and updates the value to the run-time engine. Writes the variable value with the run-time engine to the recipe data file in the SD card.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro830, Micro850, and Micro870 controllers.



Use this table to help determine the parameter values for this instruction.

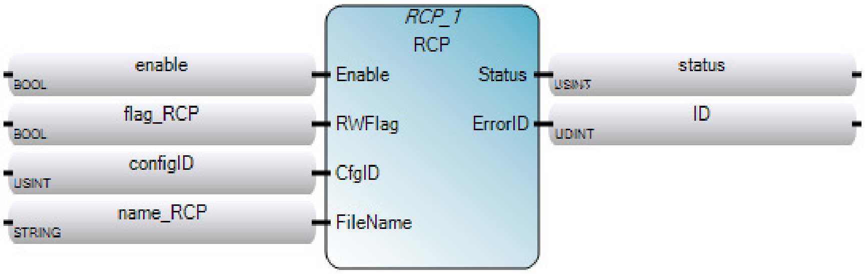
Parameter	Parameter Type	Data Type	Description
Enable	Input	BOOL	Enable recipe read/write instruction block. TRUE - Rising Edge detected, execute recipe instruction if the previous operation is completed. FALSE - Rising Edge not detected, do not execute recipe instruction.
RWFlag	Input	BOOL	TRUE - RWFlag (Write operation). Recipe writes the the variable's values with the run-time engine into a recipe data file on the SD card. FALSE - RWFlag (Read operation). The recipe reads the variable's values from the SD card and updates the corresponding variable's value to the runtime engine.
CfgID	Input	USINT	Recipe configuration VA ID number 1-10.
FileName	Input	STRING	Recipe data file name (maximum 30 characters length).
Status	Output	USINT	Recipe instruction block current status. RCP status codes: • 0 Idle • 1 Doing • 2 Succeed, complete without error. • 3 Error, complete with error.
ErrorID	Output	UDINT	The numeric RCP error code. The definitions are defined in RCP error codes.

RCP error codes

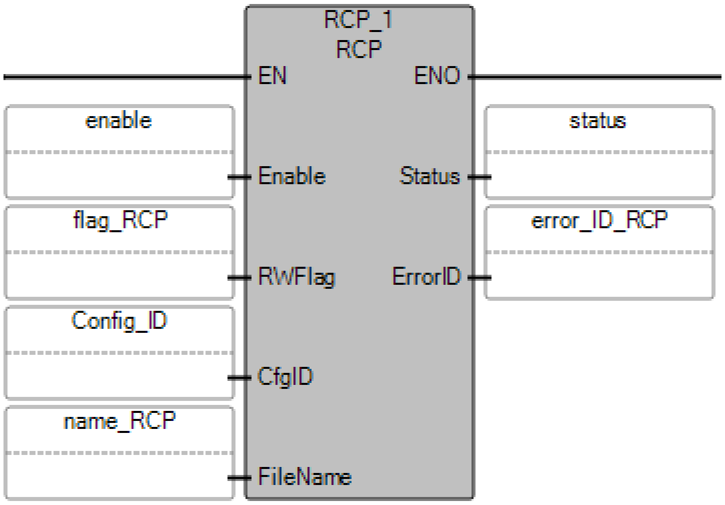
Error code	Error Name
0	RCP_ERR_NONE
1	RCP_ERR_NO_SDCARD
2	RCP_ERR_DATAFILE_FULL

Error code	Error Name
3	RCP_ERR_DATAFILE_ACCESS SD card are identified as: • broken. • full. • read only.
4	RCP_ERR_CFG_ABSENT
5	RCP_ERR_CFG_ID
6	RCP_ERR_RESOURCE_BUSY
7	RCP_ERR_CFG_FORMAT
8	RCP_ERR_RESERVED Reserved for future possible expansion.
9	RCP_ERR_UNKNOWN
10	RCP_ERR_DATAFILE_NAME
11	RCP_ERR_DATAFOLDER_INVALID
12	RCP_ERR_DATAFILE_ABSENT
13	RCP_ERR_DATAFILE_FORMAT
14	RCP_ERR_DATAFILE_SIZE Recipe data file size is too large (>4kb).

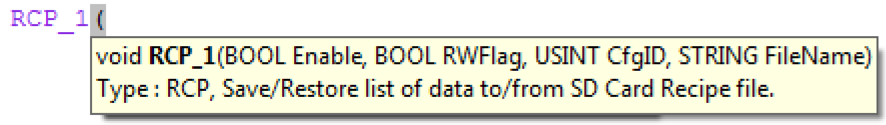
RCP Function Block Diagram example



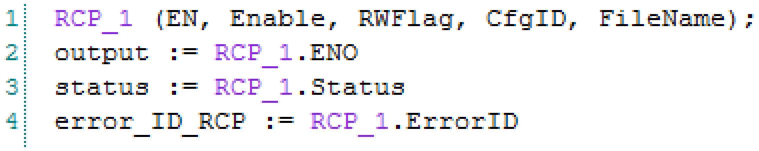
RCP Ladder Diagram example



RCP Structured Text example



```
RCP_1(
void RCP_1(BOOL Enable, BOOL RWFlag, USINT CfgID, STRING FileName)
Type : RCP, Save/Restore list of data to/from SD Card Recipe file.
```



```
RCP_1 (EN, Enable, RWFlag, CfgID, FileName);
output := RCP_1.ENO
status := RCP_1.Status
error_ID_RCP := RCP_1.ErrorID
```

RTC_READ (read real-time clock)

Reads the real-time clock (RTC) module information.

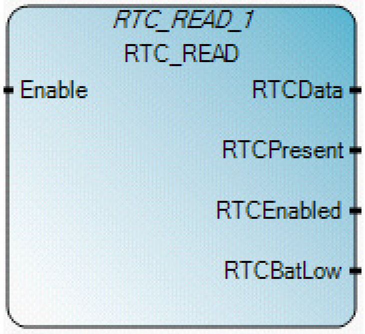
Operation details:

- Micro810 or Micro820 controller with embedded RTC:
  - RTCBatLow is always set to zero (0).
  - RTCEnabled is always set to one (1).
- When the embedded RTC has lost its charge/memory due to loss of power:
  - RTCData is set to 2000/1/1/0/0/0.
  - RTCEnabled is set to one (1).

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.

The outputs are always reset when the instruction is applied to the simulated controller (2080-LC50-48QWB-SIM).



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
Enable	Input	BOOL	Instruction block enable. TRUE - execute RTC information read. FALSE - there is no read operation and output RTC data is invalid.
RTCData	Output	RTC	RTC data information: yy/mm/dd, hh/mm/ss, week. RTCData output is defined using the RTC data type.
RTCPresent	Output	BOOL	TRUE - Free Running clock is utilized, or RTC hardware is plugged in. FALSE - Free Running clock is not utilized, or RTC hardware is not plugged in.

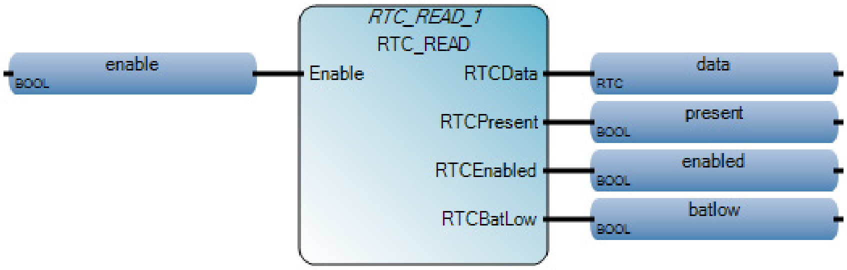
RTCEnabled	Output	BOOL	TRUE - Free Running clock is utilized, or RTC hardware is enabled (timing). FALSE - Free Running clock is not utilized, RTC hardware is disabled (not timing).
RTCBatLow	Output	BOOL	TRUE - RTC battery is low. FALSE - RTC battery is not low.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.

RTC data type

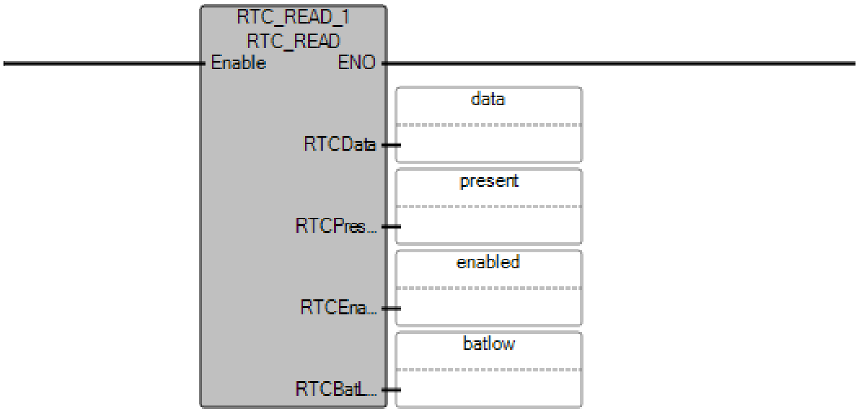
Use this table to help determine the parameter values for the RTC data type.

Parameter	Data type	Description
Year	UINT	The year setting for the RTC. 16-bit value, and the valid range is from 2000 (Jan 01, 00:00:00) to 2098 (Dec. 31, 23:59:59)
Month	UINT	The month setting for the RTC.
Day	UINT	The day setting for the RTC.
Hour	UINT	The hour setting for the RTC.
Minute	UINT	The minute setting for the RTC.
Second	UINT	The second setting for the RTC.
DayOfWeek	UINT	The day of the week setting for the RTC. This parameter is ignored for RTC_SET.

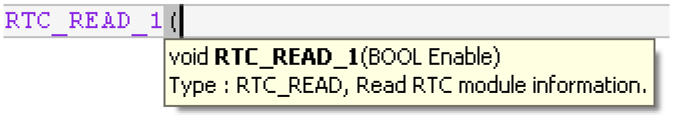
RTC_READ Function Block Diagram example



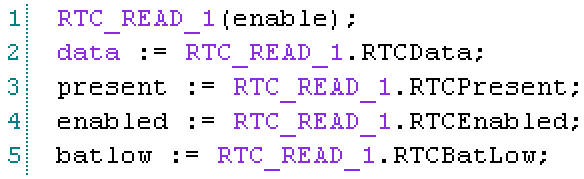
RTC_READ Ladder Diagram example



RTC_READ Structured Text example



```
RTC_READ_1(
void RTC_READ_1(BOOL Enable)
Type : RTC_READ, Read RTC module information.
```



```
RTC_READ_1(enable);
data := RTC_READ_1.RTCData;
present := RTC_READ_1.RTCPresent;
enabled := RTC_READ_1.RTCEnabled;
batlow := RTC_READ_1.RTCBatLow;
```

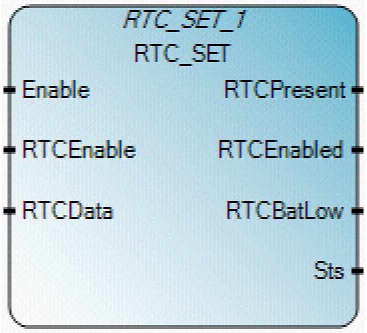
RTC_SET (set real-time clock)

Set RTC (real-time clock) data to the RTC module information.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.

The outputs are always reset when the instruction is applied to the simulated controller (2080-LC50-48QWB-SIM).



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
Enable	Input	BOOL	Instruction block enable. TRUE - execute RTC_SET with the RTC information from input. Typically, only execute for 1 program scan when updating the RTC. FALSE - do not execute RTC_SET. Set to FALSE to operate RTC normally.
RTCEnable	Input	BOOL	TRUE - To enable RTC with the RTC data specified. FALSE - To disable RTC.
RTCData	Input	RTC	RTC data information: yy/mm/dd, hh/mm/ss, week as defined in the RTC data type. RTCData is ignored when RTCEnable = 0.
RTCPresent	Output	BOOL	TRUE - Free Running clock is utilized, or RTC hardware is plugged in. FALSE - Free Running clock is not utilized, or RTC hardware is not plugged in.
RTCEnabled	Output	BOOL	TRUE - Free Running clock is utilized, or RTC hardware is enabled (timing). FALSE - Free Running clock is not utilized, or RTC hardware is disabled (not timing).
RTCBatLow	Output	BOOL	TRUE - RTC battery is low. FALSE - RTC battery is not low.

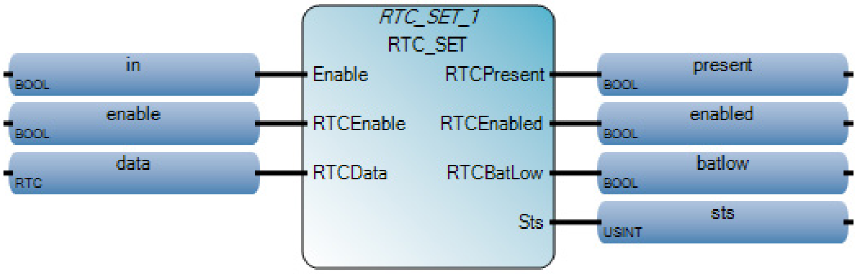
Sts	Output	USINT	The read operation status. RTC_Set status (Sts) values: • 0x00 - Function block not enabled (no operation). • 0x01 - RTC set operation success. • 0x02 - RTC set operation fails.

RTC data type

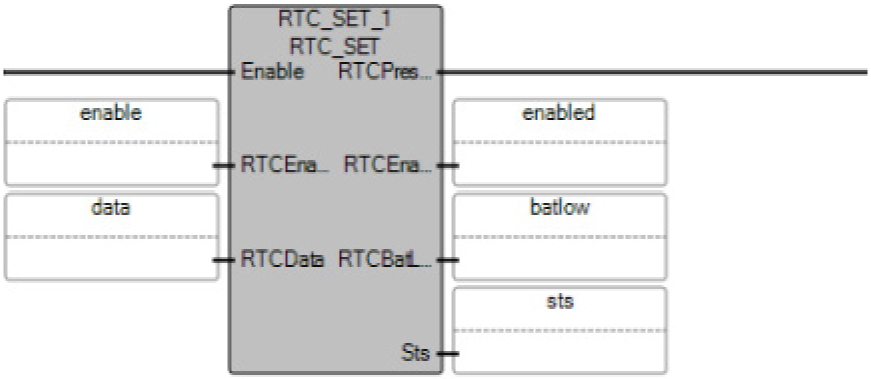
Use this table to help determine the parameter values for the RTC data type.

Parameter	Data type	Description
Year	UINT	The year setting for the RTC. 16-bit value, and the valid range is from 2000 (Jan 01, 00:00:00) to 2098 (Dec. 31, 23:59:59)
Month	UINT	The month setting for the RTC.
Day	UINT	The day setting for the RTC.
Hour	UINT	The hour setting for the RTC.
Minute	UINT	The minute setting for the RTC.
Second	UINT	The second setting for the RTC.
DayOfWeek	UINT	The day of the week setting for the RTC. This parameter is ignored for RTC_SET.

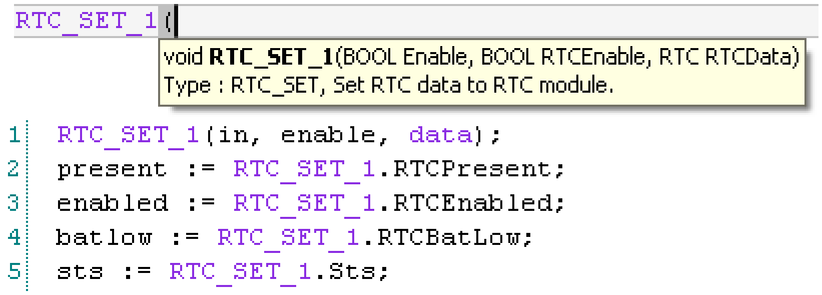
RTC_SET Function Block Diagram example



RTC_SET Ladder Diagram example



RTC_SET Structured Text example



```
RTC_SET_1(
```

void **RTC_SET_1**(BOOL Enable, BOOL RTCEnable, RTC RTCData)
Type : RTC_SET, Set RTC data to RTC module.

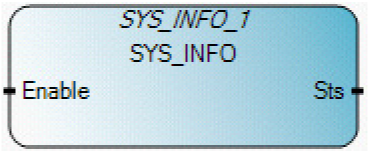
```
RTC_SET_1(in, enable, data);
present := RTC_SET_1.RTCPresent;
enabled := RTC_SET_1.RTCEnabled;
batlow := RTC_SET_1.RTCBatLow;
sts := RTC_SET_1.Sts;
```

SYS_INFO

Reads the status data block for the Micro800 controller.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

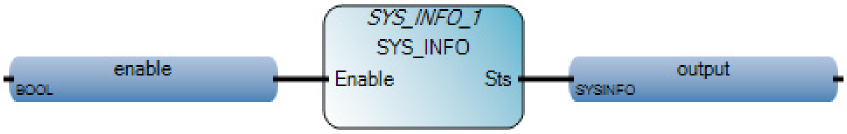
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



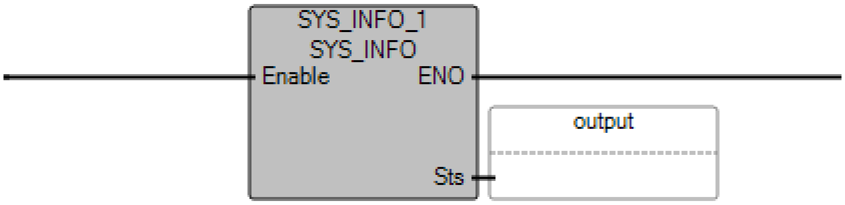
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
Enable	Input	BOOL	Instruction block enable. TRUE - execute read operation. FALSE - do not execute function.
Sts	Output	SYSINFO	System status data block. The Sts output is defined in the SYS_INFO data type on page 374.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.

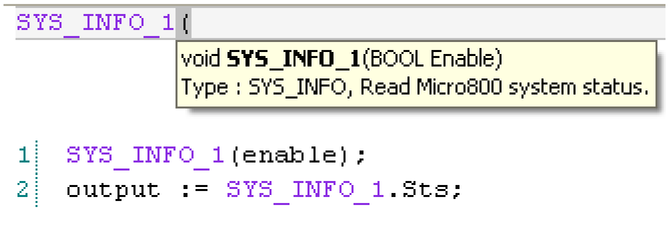
SYS_INFO Function Block Diagram example



SYS_INFO Ladder Diagram example



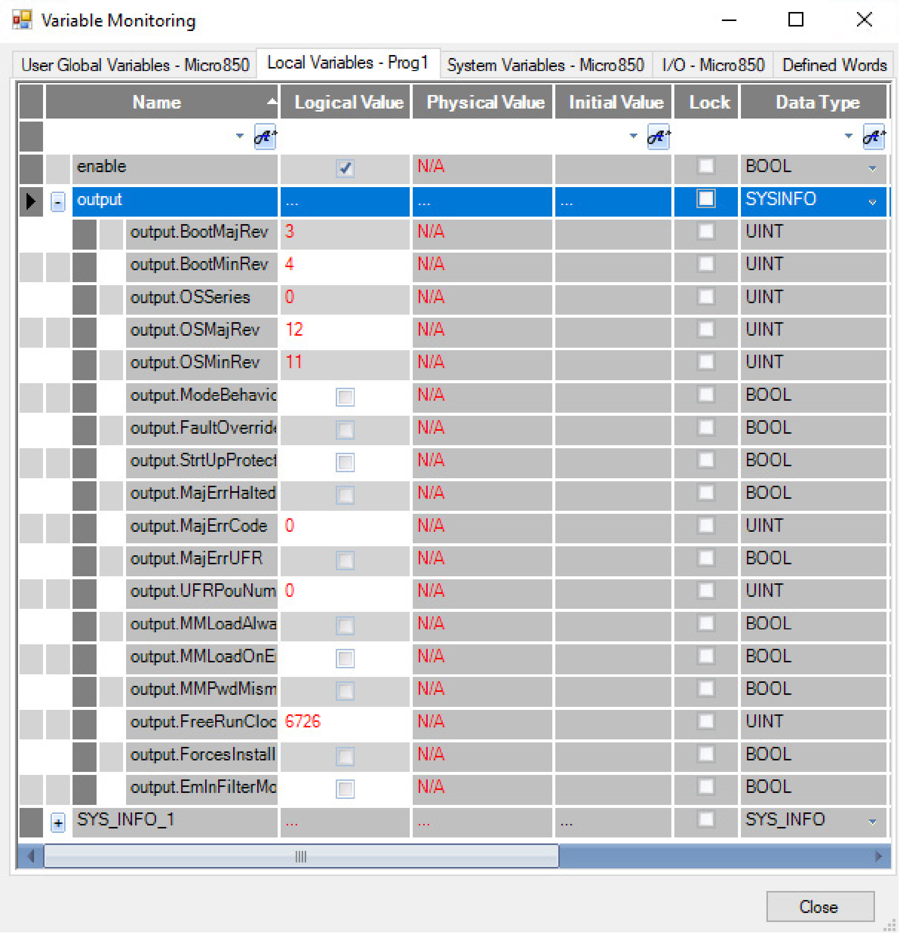
SYS_INFO Structured Text example



```
SYS_INFO_1(
void SYS_INFO_1(BOOL Enable)
Type : SYS_INFO, Read Micro800 system status.
```

```
SYS_INFO_1(enable);
output := SYS_INFO_1.Sts;
```

Results



Name	Logical Value	Physical Value	Initial Value	Lock	Data Type
enable	☑	N/A		☐	BOOL
output	...	...	...	☐	SYSINFO
output.BootMajRev	3	N/A		☐	UINT
output.BootMinRev	4	N/A		☐	UINT
output.OSSeries	0	N/A		☐	UINT
output.OSMajRev	12	N/A		☐	UINT
output.OSMinRev	11	N/A		☐	UINT
output.ModeBehavic	☐	N/A		☐	BOOL
output.FaultOverrid	☐	N/A		☐	BOOL
output.StrtUpProtect	☐	N/A		☐	BOOL
output.MajErrHalted	☐	N/A		☐	BOOL
output.MajErrCode	0	N/A		☐	UINT
output.MajErrUFR	☐	N/A		☐	BOOL
output.UFRPouNum	0	N/A		☐	UINT
output.MMLoadAlwa	☐	N/A		☐	BOOL
output.MMLoadOnE	☐	N/A		☐	BOOL
output.MMPwdMism	☐	N/A		☐	BOOL
output.FreeRunCloc	6726	N/A		☐	UINT
output.ForcesInstall	☐	N/A		☐	BOOL
output.EmInFilterMc	☐	N/A		☐	BOOL
SYS_INFO_1	...	...	...	☐	SYS_INFO

SYS_INFO data type

The following table describes the SYSINFO data type.

Parameter	Data type	Description
BootMajRev	UINT	Boot Major Revision.
BootMinRev	UINT	Boot Minor Revision.
Operating System Series	UINT	Operating System Series: 0 indicates a series A device 1 indicates a series B device
OSMajRev	UINT	OS Major Revision.
OSMinRev	UINT	OS Minor Revision.
ModeBehaviour	BOOL	Mode Behavior (TRUE: Go to RUN on power up).

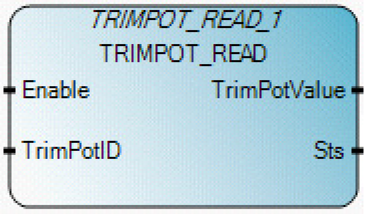
Parameter	Data type	Description
FaultOverride	BOOL	Fault Override (TRUE: Override error on power up).
StrtUpProtect	BOOL	Startup Protection (TRUE: Run startup protection program on power up). For future release.
MajErrHalted	BOOL	Major error halted (TRUE: Major error halted).
MajErrCode	UINT	Major error code.
MajErrUFR	BOOL	Major error during user fault routine. For future release.
UFRPouNum	UINT	User fault routine program number.
MMLoadAlways	BOOL	Memory Module restore to controller always on power up (TRUE: Restore).
MMLoadOnError	BOOL	Memory Module restore to controller if power up with error (TRUE: Restore).
MMPwdMismatch	BOOL	Memory Module password mismatch (TRUE: Controller and Memory Module password mismatch).
FreeRunClock	UINT	Free running clock that increments every 100 microseconds from 0 to 65535 and then returns to 0. You can use the Clock, which is globally accessible, if you need more resolution than the standard 1 millisecond timer. Only supported for Micro830 and Micro850 controllers. Value for Micro810 controllers remains 0.
ForcesInstall	BOOL	Forces enabled (TRUE: Enabled).
EMINFilterMod	BOOL	Embedded filter modified (TRUE: Modified).

TRIMPOT_READ (read trimpot)

Reads the trimpot value from a specific trimpot.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro830, Micro850, and Micro870 controllers.

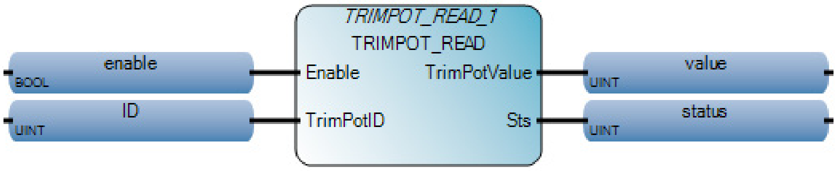


Use this table to help determine the parameter values for this instruction.

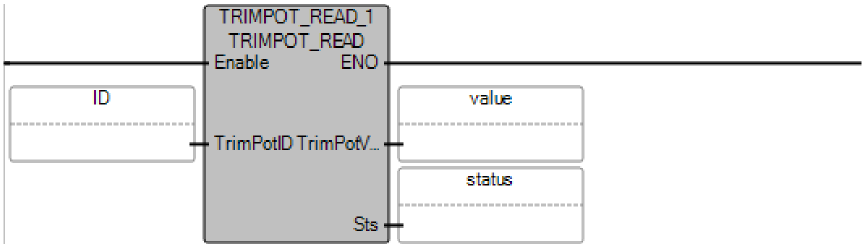
Parameter	Parameter Type	Data Type	Description
Enable	Input	BOOL	Instruction block enable. TRUE - execute Trimpot read. FALSE - there is no read operation and output Trimpot value is invalid.
TrimPotID	Input	UINT	The ID of the Trimpot to be read. TrimPotID is defined in Trimpot ID definitions on page 376.
TrimPotValue	Output	UINT	Current trimpot value.

Sts	Output	UINT	The Trimpot read operation status. TRIMPOT status (Sts) codes: ● 0x00 - Function block not enabled (no read/write operation). ● 0x01- Read/write operation success. ● 0x02 - Read/write operation fails due to an invalid Trimpot ID. ● 0x03 - Write operation fails due to an out of range value.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.

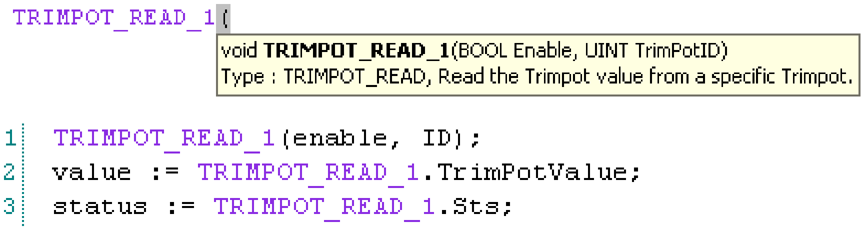
TRIMPOT Function Block Diagram example



TRIMPOT Ladder Diagram example



TRIMPOT Structured Text example



```
TRIMPOT_READ_1(
```

void **TRIMPOT_READ_1**(BOOL Enable, UINT TrimPotID)
Type : TRIMPOT_READ, Read the Trimpot value from a specific Trimpot.

```
TRIMPOT_READ_1(enable, ID);
value := TRIMPOT_READ_1.TrimPotValue;
status := TRIMPOT_READ_1.Sts;
```

Trimpot ID definition

The following table describes the Trimpot ID definition used in the TRIMPOT_read instruction on page 375.

Output selection	Bit	Description
Trimpot ID definition	15 - 13	Module type of trimpot: ● 0x00 - Embedded. ● 0x01 - Expansion. ● 0x02 - Plug-in Port.

Output selection	Bit	Description
	12 - 8	Slot ID of the module: • 0x00 - Embedded. • 0x01-0x1F - ID of Expansion Module. • 0x01-0x05 - ID of Plug-in Port.
	7 - 4	Trimpot type: • 0x00 - Reserved. • 0x01 - Digital Trimpot Type 1 (LCD Module 1). • 0x02 - Mechanical Trimpot Module 1.
	3 - 0	Trimpot ID inside the module: • 0x00-0x0F - Embedded. • 0x00-0x07 - ID of Trimpot for Expansion. • 0x00-0x07 - ID of Trimpot for Plug-in Port. The trimpot ID starts from 0.

Interrupt instructions

Use Interrupt instructions to signal the processor that an event needs attention. Usually the interrupt signal is used for high-priority conditions that require interruption of the current code the processor is executing.

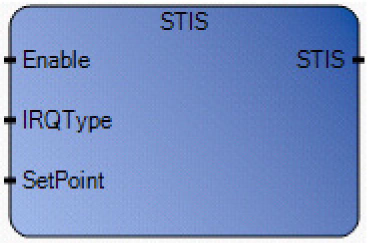
Function	Description
STIS on page 379	Starts the selected timed user interrupt (STI) timer from the control program rather than starting automatically.
UIC on page 381	Clears the lost bit for the selected user interrupt.
UID on page 382	Disables a specific user interrupt.
UIE on page 384	Enables a specific user input.
UIF on page 386	Flushes or removes a pending user input.

STIS (select timed start)

Starts the selectable timed user interrupt (STI) timer from the control program rather than starting automatically.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.

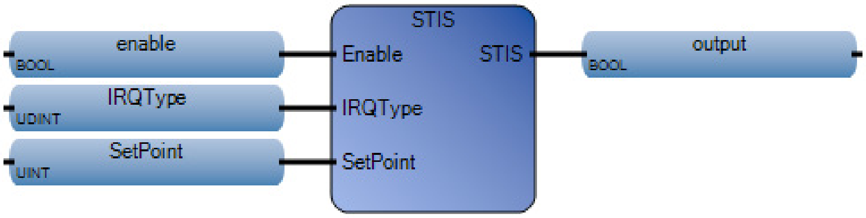


Use this table to help determine the parameter values for this instruction.

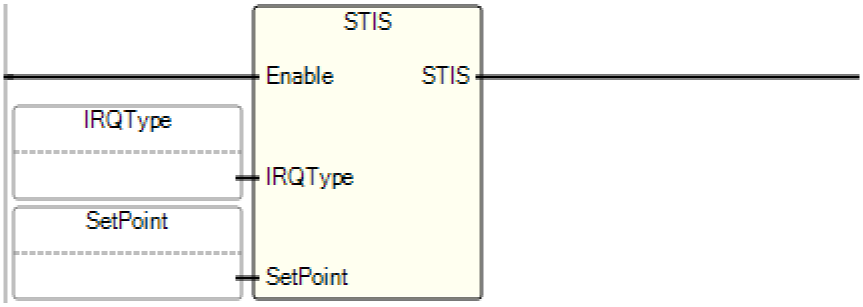
Parameter	Parameter Type	Data Type	Description
Enable	Input	BOOL	Instruction enable. TRUE - start the STI timer from the control program. FALSE - do not perform function.
IRQType	Input	UDINT	Use the STI defined words. - IRQ_STI0 - IRQ_STI1 - IRQ_STI2 - IRQ_STI3

SetPoint	Input	UINT	The amount of time (in ms) which must expire prior to executing the selectable timed interrupt. A value of 0 disables the STIS function. A value between 1 and 65535 enables the STIS function.
STIS	Output	BOOL	Rung status (same as Enable).

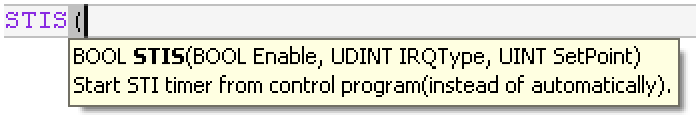
STIS Function Block Diagram example



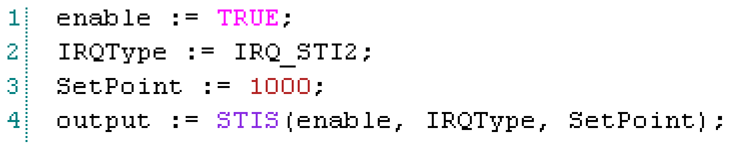
STIS Ladder Diagram example



STIS Structured Text example



```
STIS (
BOOL STIS(BOOL Enable, UDINT IRQType, UINT SetPoint)
Start STI timer from control program(instead of automatically).
```

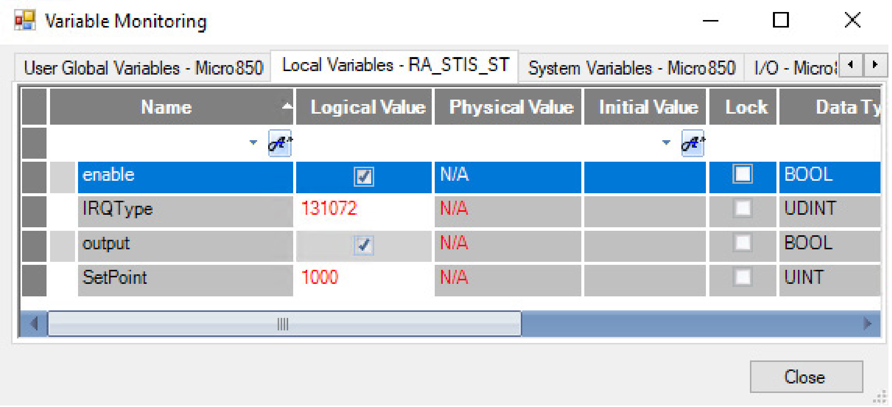


```
enable := TRUE;
IRQType := IRQ_STI2;
SetPoint := 1000;
output := STIS(enable, IRQType, SetPoint);
```

(* ST Equivalence: *)

TESTOUTPUT := STIS(TESTENABLE, 2, 1000) ;

Results



Variable Monitoring

User Global Variables - Micro850
Local Variables - RA_STIS_ST
System Variables - Micro850
I/O - Micro8
Close

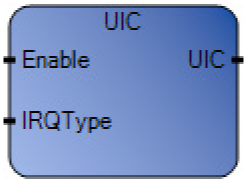
Name	Logical Value	Physical Value	Initial Value	Lock	Data Ty
enable	☑	N/A		□	BOOL
IRQType	131072	N/A		□	UDINT
output	☑	N/A		□	BOOL
SetPoint	1000	N/A		□	UINT

UIC (clear interrupt lost bit)

Clears the lost bit for the selected user interrupt(s).

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

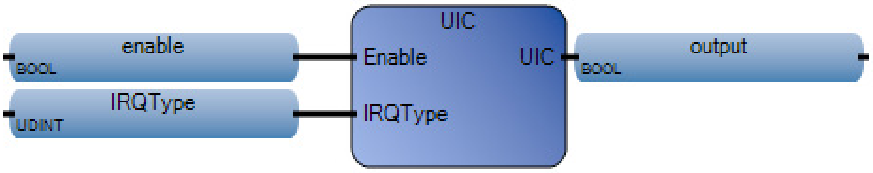
This instruction applies to the Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



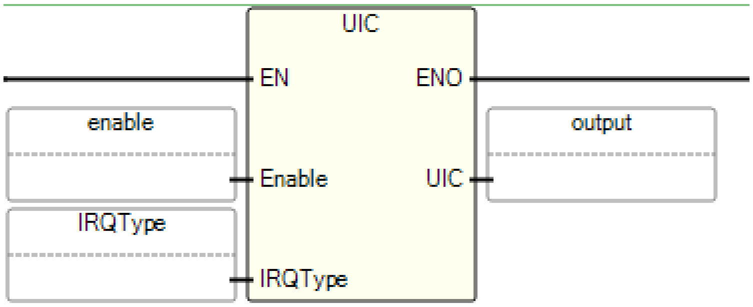
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
Enable	Input	BOOL	Instruction enable. TRUE - start clear bit operation. FALSE - do not perform function.
IRQType	Input	UDINT	Use the STI defined words. - IRQ_EII0 - IRQ_EII1 - IRQ_EII2 - IRQ_EII3 - IRQ_EII4 - IRQ_EII5 - IRQ_EII6 - IRQ_EII7 - IRQ_HSC0 - IRQ_HSC1 - IRQ_HSC2 - IRQ_HSC3 - IRQ_HSC4 - IRQ_HSC5 - IRQ_STI0 - IRQ_STI1 - IRQ_STI2 - IRQ_STI3 - IRQ_UFR - IRQ_UPM0 - IRQ_UPM1 - IRQ_UPM2 - IRQ_UPM3 - IRQ_UPM4
UIC	Output	BOOL	Rung status (same as Enable).

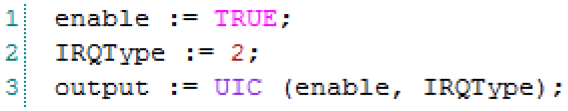
UIC Function Block Diagram example



UIC Ladder Diagram example

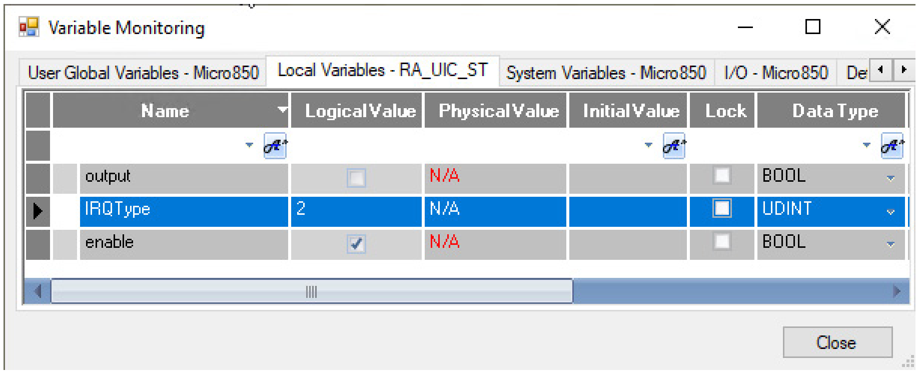


UIC Structure Text example



```
enable := TRUE;
IRQType := 2;
output := UIC (enable, IRQType);
```

Results


Variable Monitoring
User Global Variables - Micro850
Local Variables - RA_UIC_ST
System Variables - Micro850
I/O - Micro850
De
Close

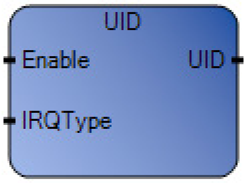
Name	Logical Value	Physical Value	Initial Value	Lock	Data Type
output	□	N/A		□	BOOL
IRQType	2	N/A		□	UDINT
enable	☑	N/A		□	BOOL

UID (disable interrupt)

Disables a specific user interrupt(s).

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

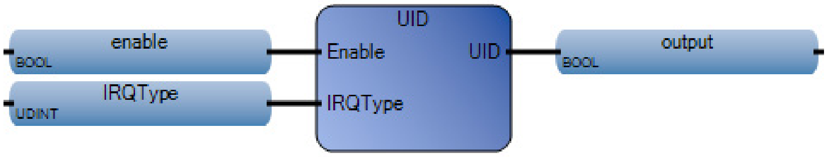
This instruction applies to the Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



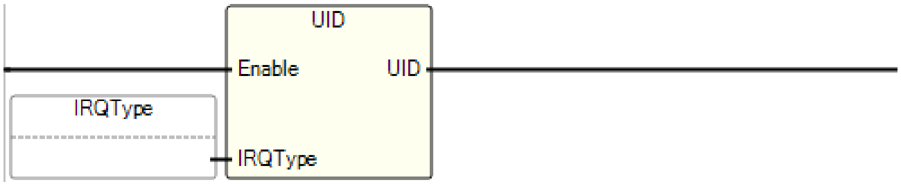
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description	
Enable	Input	BOOL	Instruction enable. TRUE - start the disable operation. FALSE - do not perform function.	
IRQType	Input	UDINT	Use the STI defined words. - IRQ_EII0 - IRQ_EII1 - IRQ_EII2 - IRQ_EII3 - IRQ_EII4 - IRQ_EII5 - IRQ_EII6 - IRQ_EII7 - IRQ_HSC0 - IRQ_HSC1 - IRQ_HSC2	- IRQ_HSC3 - IRQ_HSC4 - IRQ_HSC5 - IRQ_STI0 - IRQ_STI1 - IRQ_STI2 - IRQ_STI3 - IRQ_UFR - IRQ_UPM0 - IRQ_UPM1 - IRQ_UPM2 - IRQ_UPM3 - IRQ_UPM4
UID	Output	BOOL	Rung status (same as Enable).	

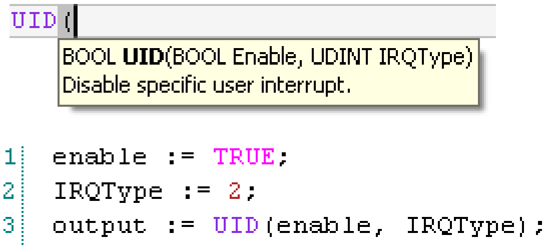
UID Function Block Diagram example



UID Ladder Diagram example



UID Structured Text example



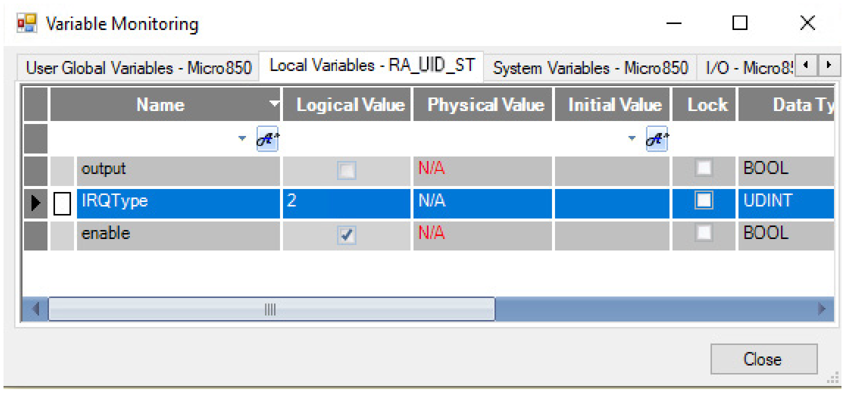
```
UID (
BOOL UID(BOOL Enable, UDINT IRQType)
Disable specific user interrupt.
```

```
enable := TRUE;
IRQType := 2;
output := UID(enable, IRQType);
```

(* ST Equivalence: *)

TESTOUTPUT := UID(TESTENABLE, 2) ;

Results



Variable Monitoring

User Global Variables - Micro850 | Local Variables - RA_UID_ST | System Variables - Micro850 | I/O - Micro8!

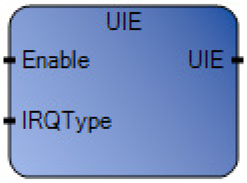
Name	Logical Value	Physical Value	Initial Value	Lock	Data Ty
output	□	N/A		□	BOOL
IRQType	2	N/A		□	UDINT
enable	✓	N/A		□	BOOL

UIE (enable interrupt)

Enables a specific user interrupt.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.

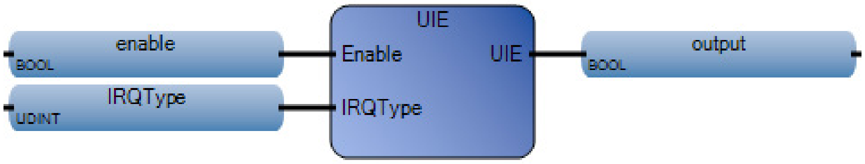


Use this table to help determine the parameter values for this instruction.

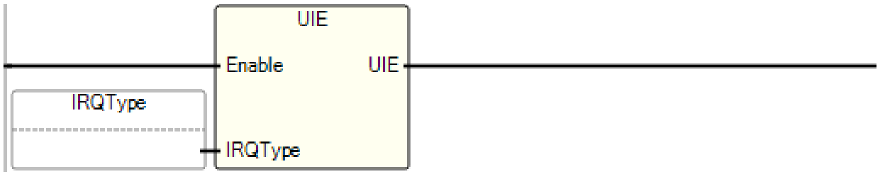
Parameter	Parameter Type	Data Type	Description
Enable	Input	BOOL	Instruction enable. TRUE - start enable operation. FALSE - do not perform function.

IRQType	Input	UDINT	Use the STI defined words. - IRQ_EII0 - IRQ_EII1 - IRQ_EII2 - IRQ_EII3 - IRQ_EII4 - IRQ_EII5 - IRQ_EII6 - IRQ_EII7 - IRQ_HSC0 - IRQ_HSC1 - IRQ_HSC2	- IRQ_HSC3 - IRQ_HSC4 - IRQ_HSC5 - IRQ_STI0 - IRQ_STI1 - IRQ_STI2 - IRQ_STI3 - IRQ_UFR - IRQ_UPM0 - IRQ_UPM1 - IRQ_UPM2 - IRQ_UPM3 - IRQ_UPM4
UIE	Output	BOOL	Rung status (same as Enable).	

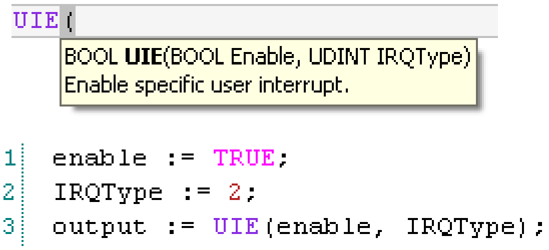
UIE Function Block Diagram example



UIE Ladder Diagram example



UIE Structured Text example



```
UIE(
```

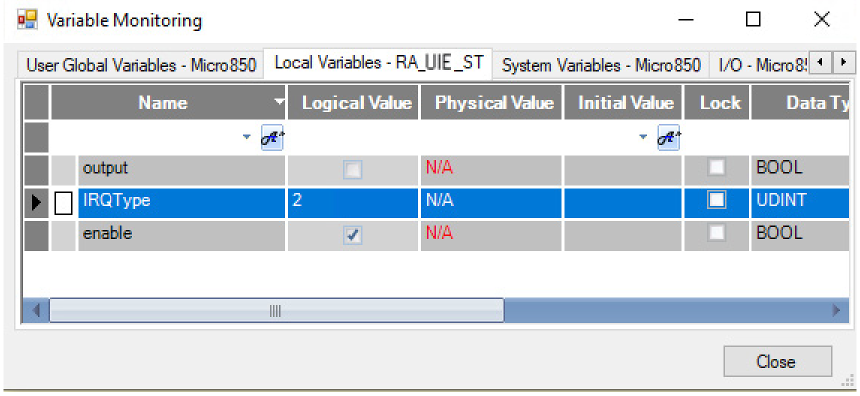
BOOL **UIE**(BOOL Enable, UDINT IRQType)
Enable specific user interrupt.

```
enable := TRUE;
IRQType := 2;
output := UIE(enable, IRQType);
```

(* ST Equivalence: *)

TESTOUTPUT := UIE(TESTENABLE, 2) ;

Results


Variable Monitoring

User Global Variables - Micro850 | Local Variables - RA_UIE_ST | System Variables - Micro850 | I/O - Micro8

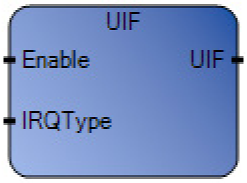
Name	Logical Value	Physical Value	Initial Value	Lock	Data Ty
output	☐	N/A		☐	BOOL
IRQType	2	N/A		☐	UDINT
enable	☑	N/A		☐	BOOL

UIF (flush pending interrupt)

Flushes or removes a pending user interrupt.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

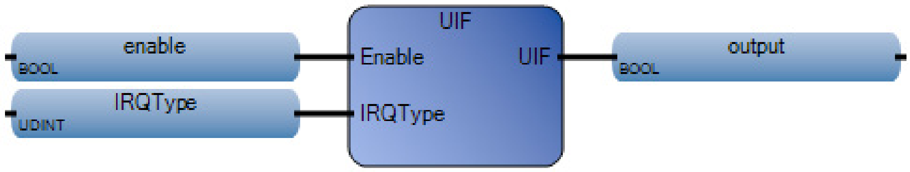
This instruction applies to the Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



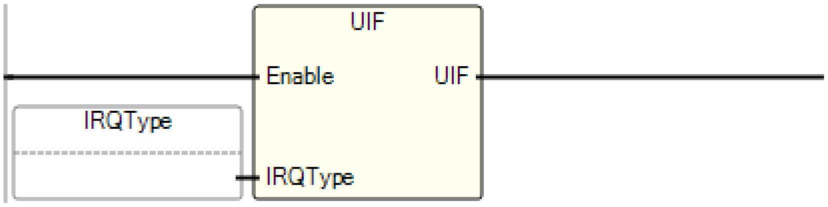
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
Enable	Input	BOOL	Instruction enable. TRUE - start UIF operation. FALSE - do not perform function.
IRQType	Input	UDINT	Use the STI defined words. - IRQ_EII0 - IRQ_EII1 - IRQ_EII2 - IRQ_EII3 - IRQ_EII4 - IRQ_EII5 - IRQ_EII6 - IRQ_EII7 - IRQ_HSC0 - IRQ_HSC1 - IRQ_HSC2 - IRQ_HSC3 - IRQ_HSC4 - IRQ_HSC5 - IRQ_STI0 - IRQ_STI1 - IRQ_STI2 - IRQ_STI3 - IRQ_UFR - IRQ_UPM0 - IRQ_UPM1 - IRQ_UPM2 - IRQ_UPM3 - IRQ_UPM4
UIF	Output	BOOL	Rung status (same as Enable).

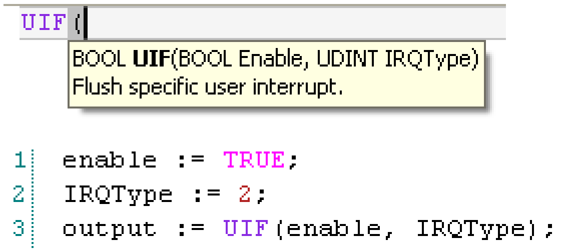
UIF Function Block Diagram example



UIF Ladder Diagram example



UIF Structured Text example



```
UIF (
```

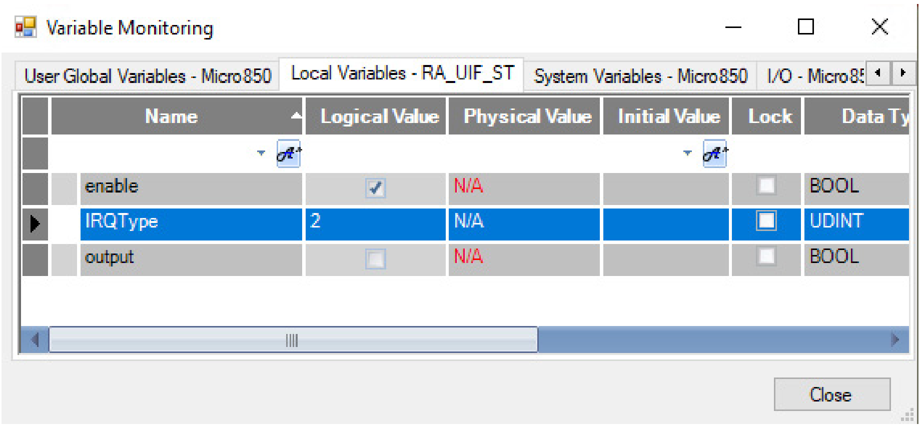
BOOL **UIF**(BOOL Enable, UDINT IRQType)
Flush specific user interrupt.

```
enable := TRUE;
IRQType := 2;
output := UIF(enable, IRQType);
```

(* ST Equivalence: *)

TESTOUTPUT := UIF(TESTENABLE, 2) ;

Results


Variable Monitoring
User Global Variables - Micro850
Local Variables - RA_UIF_ST
System Variables - Micro850
I/O - Micro85
Close

Name	Logical Value	Physical Value	Initial Value	Lock	Data Ty
enable	☑	N/A		□	BOOL
IRQType	2	N/A		□	UDINT
output	□	N/A		□	BOOL

Motion control instructions

Use the motion control instructions to program and design the motion of a particular axis. Motion control requires Connected Components Workbench 2.0 or higher.

Tip:

- Administrative instructions support both PTO motion and Feedback motion.
- The motion control instructions that support an FBAxis are: MC_ReadActualPosition and MC_ReadActualVelocity.

Instruction	Description
Administrative	
MC_AbortTrigger on page 399	Cancells Motion function blocks that are connected to trigger events.
MC_Power on page 420	Control the power stage, ON or OFF.
MC_ReadAxisError on page 430	Reads the axis errors not related to the Motion control instruction blocks.
MC_ReadBoolParameter on page 434	Returns the value of a vendor specific parameter of type BOOL.
MC_ReadParameter on page 437	Returns the value of a vendor specific parameter of type Real.
MC_ReadStatus on page 439	Returns the status of the axis with respect to the motion currently in progress.
MC_Reset on page 445	Transitions the axis state from ErrorStop to StandStill by resetting all internal axis-related errors.
MC_SetPosition on page 448	Shifts the coordinate system of an axis by manipulating the actual position.
MC_TouchProbe on page 453	Records an axis position at a trigger event.
MC_WriteBoolParameter on page 457	Modifies the value of a vendor specific parameter of type BOOL.
MC_WriteParameter on page 459	Modifies the value of a vendor specific parameter of type REAL.
Motion	
MC_Halt on page 402	Commands a controlled motion stop under normal operating conditions.
MC_Home on page 405	Commands the axis to perform the <*search home*> sequence.
MC_MoveAbsolute on page 408	Commands a controlled motion to a specified absolute position.
MC_MoveRelative on page 412	Commands a controlled motion of a specified distance relative to the actual position at the time of the execution.
MC_MoveVelocity on page 416	Commands a never ending controlled motion at a specified velocity.
MC_ReadActualPosition on page 424	Returns the actual position of the feedback axis. Requires Connected Components Workbench 8 or higher.
MC_ReadActualVelocity on page 428	Returns the actual velocity of the feedback axis. Requires Connected Components Workbench 8 or higher.
MC_Stop on page 450	Commands a controlled motion stop and transfers the axis state to Stopping.

General rules for motion control function blocks

The general rules for the Micro800 motion control function blocks on page 389 follow the PLCopen Motion control specifications. The following table provides general rules about the interface of motion control function blocks.

Rule applies to	Rule
Input parameters	**With Execute**: The parameters are used with the rising edge of the execute input. To modify any parameter, change the input parameter(s) and trigger the motion again. If an instance of a function block receives a new Execute before it finishes (as a series of commands on the same instance), the new Execute is ignored, and the previously issued instruction continues with its execution. **With Enable**: The parameters are used with the rising edge of the enable input and can be modified continuously.
Missing input parameters	Missing input is captured during User Application compilation. There is no missing input error handling at the controller level.
Inputs exceeding application limits	If a function block is commanded with parameters that result in a violation of application limits, the instance of the function block generates an error. In this case, the Error output is flagged On, and error information is indicated by the output ErrorID. The controller, in most cases, remains in Run mode, and there is no Motion Error reported as a major controller fault.
Sign rules for inputs	The Acceleration, Deceleration, and Jerk inputs are always positive values. Velocity, Position and Distance inputs can have positive and negative values.
Position versus Distance	Position is a value defined within a coordinate system. Distance is a relative measure related to technical units. Distance is the difference between two positions.
Position/Distance input	Only linear motion is supported on Micro800 controllers. For MC_MoveAbsolute function block, the position input is the absolute location to be commanded to the axis. For MC_MoveRelative, the distance input is the relative location (considering current axis position is 0) from current position.
Velocity input	Velocity can be a signed value, but it can also use Direction input to define the sign of the velocity (negative velocity x negative direction = positive velocity). The E parameter "Direction" refers to the velocity input and output for compatibility reasons.
Direction input	For distance (position) motion, with the target position (either absolute, or relative) defined, the motion direction is unique. The direction input for distance move is ignored. For velocity motion, direction input value can be 1 (positive direction), 0 (current direction) or -1 (negative direction). For any other value, only the sign is considered. For example, -3 denotes negative direction, +2 denotes positive direction, and so on. For velocity move (MC_MoveVelocity), the sign (velocity x direction) determines the actual motion direction if the value is not 0. For example, if velocity x direction = +300, then direction is positive.
Acceleration, Deceleration and Jerk inputs	• Deceleration or Acceleration inputs should have a positive value. If Deceleration or Acceleration is set to a non-positive value, the function block reports an error (Error ID: MC_FB_ERR_RANGE). • Jerk input should have a non-negative value. If Jerk is set to a negative value, the function block reports an error (Error ID: MC_FB_ERR_RANGE). • If Maximum Jerk is set to zero, all jerk parameters for the motion control function block, including jerk setting for MC_Stop have to be set to zero. If they are not, the function block reports an error (Error ID: MC_FB_ERR_RANGE). • If Jerk is set to a non-zero value, S-Curve profile is generated; if Jerk is set to 0, trapezoidal profile is generated. • Home Jerk configuration is not limited to Max Jerk configuration. • If the motion engine fails to generate the motion profile prescribed by the dynamic input parameters, the function block reports an error (Error ID: MC_FB_ERR_PROFILE).

Rule applies to	Rule
Output exclusivity	With Execute: When Execute is TRUE, one of the Busy, Done, Error, or CommandAborted outputs must also be TRUE. The outputs are mutually exclusive: only one output on one function block can be TRUE. Only one of the outputs Active, Error, Done and CommandAborted is set at one time. With Enable: The Valid and Error outputs are mutually exclusive: only one output on one function block can be TRUE.
Output status	With Execute: The Done, Error, ErrorID and CommandAborted outputs are reset with the falling edge of Execute instruction. However, the falling edge of Execute does not stop or influence the execution of the actual function block. Even if Execute is reset before the function block completes, the corresponding outputs are set for at least one cycle. If an instance of a function block receives a new Execute command before it completes (as a series of commands on the same instance), the new Execute command is ignored, and the previously issued instruction continues with execution. With Enable: Valid, Enabled, Busy, Error, and ErrorID outputs are reset with the falling edge of Enable as soon as possible.
Behavior of Done output	The Done output is set when the commanded action has successfully completed. When multiple function blocks are working on the same axis in a sequence, the following applies: • When one movement on an axis is interrupted with another movement on the same axis without having reached the final goal, Done on the first function block will not be set.
Behavior of Busy output	Every function block can have a Busy output, indicating the function block is not finished (for function blocks with an Execute input) or is not working and new output values can be expected (in case of Enable input). Busy is set at the rising edge of Execute and reset when one of the outputs Done, Aborted, or Error is set. The function block should be kept in the active loop of the application program for at least as long as Busy is TRUE because the outputs may change. Function blocks with the same instance that are busy cannot execute until it is no longer busy. Function blocks with different instances can override the currently executing function block.
Behavior of CommandAborted output	The CommandAborted output is set when a commanded motion is interrupted by another motion command. The reset behavior of CommandAborted output is similar to Done output. When CommandAborted occurs, other output signals such as InVelocity are reset.
Output Active	The Active output is required on buffered function blocks, and is set at the moment the function block takes control of the motion of the according axis. For unbuffered mode, the Active and Busy outputs can have the same value.
Enable and Valid status	The Enable input is coupled to a Valid output. Enable is level sensitive, and Valid shows that a valid set of outputs is available at the function block. The Valid output is TRUE as long as a valid output value is available and the Enable input is TRUE. The relevant output value can be refreshed as long as the input Enable is TRUE. If there is a function block error, the output is not valid (Valid set to FALSE). When the error condition disappears, the values reappear and Valid output is set again.
Output error handling	Outputs used to define errors All blocks have the following two outputs that are used for errors that occur during execution: • Error – Rising edge of "Error" informs that an error occurred during the execution of the function block. • ErrorID – Error number. Done and InVelocity outputs are used for successful completion so they are logically exclusive to Error. Instance errors do not always result in an axis error (bringing the axis to ErrorStop).
	How the error outputs are reset • Error outputs of the relevant function blocks are reset with the falling edge of Execute and Enable. • The error outputs of a function block with Enable can be reset during operation without having to reset Enable.

Rule applies to	Rule
	Types of errors • Function blocks logics (for example, parameters out of range, state machine violation attempted, and so on) • HW Limit or SW Limit • Mechanism/Motor • Drive
Naming conventions ENUM types	Due to the naming constraints in the IEC standard on the uniqueness of variable names, the 'mc' reference to the PLCopen Motion Control namespace is used for the ENUMs. In this way we avoid the conflict that using the ENUM types 'positive' and 'negative' for instance with variables with these names throughout the rest of the project since they are called mcPositive and mcNegative respectively.

Motion control function block parameter details

The following topics provide details for motion control parameters that are relevant to all motion control function blocks.

Motion control axis states on page 392

Motion control function block parameter numbers on page 394

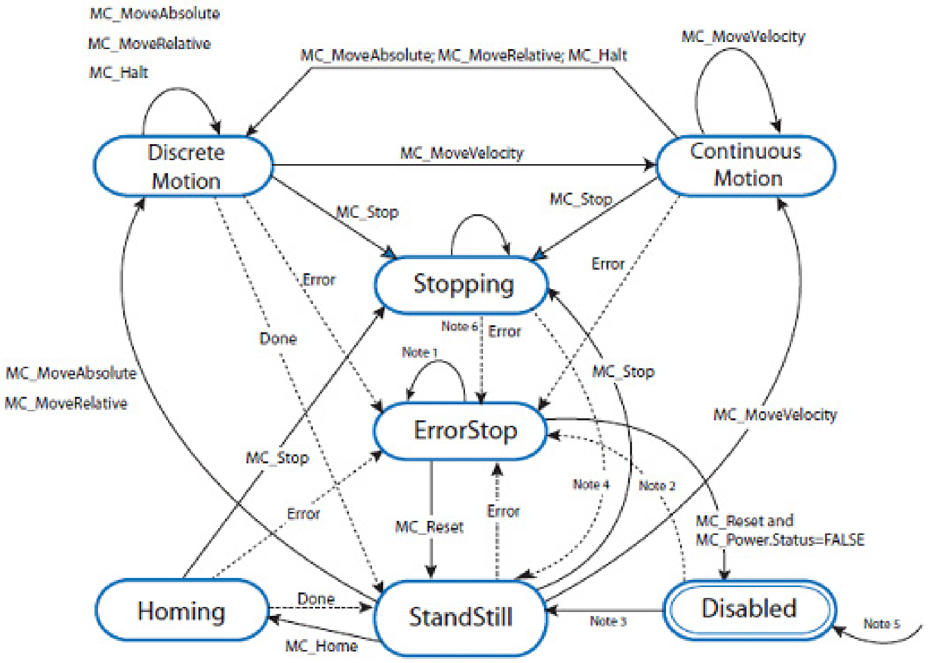
Motion control function block error IDs on page 395

Motion control axis states

The basic rule for the behavior of the axis at a high level when multiple motion control function blocks are activated is that motion commands are always taken sequentially, even if the controller has the capability of real parallel processing. Any motion command is a transition that changes the state of the axis and, as a consequence, modifies the way the current motion is computed.

Motion control axis state diagram

The axis is always in one of the defined states as shown in the following diagram.



Motion control axis state behavior

No	Note
1	In the ErrorStop and Stopping states, all function blocks (except MC_Reset), can be called although they will not be executed. MC_Reset generates a transition to the Standstill state. If an error occurs while the state machine is in the Stopping state, a transition to the ErrorStop state is generated.
2	Power.Enable = TRUE and there is an error in the axis.
3	Power.Enable = TRUE and there is no error in the axis
4	MC_Stop.Done AND NOT MC_Stop.Execute.
5	When MC_Power is called with Enable = False, the axis goes to the Disabled state for every state including ErrorStop.
6	If an error occurs while the state machine is in Stopping state, a transition to the ErrorStop state is generated.

Motion control axis state code values

You can monitor the axis state using the Axis Monitor feature. The following table identifies the values used to define each of the predefined axis states.

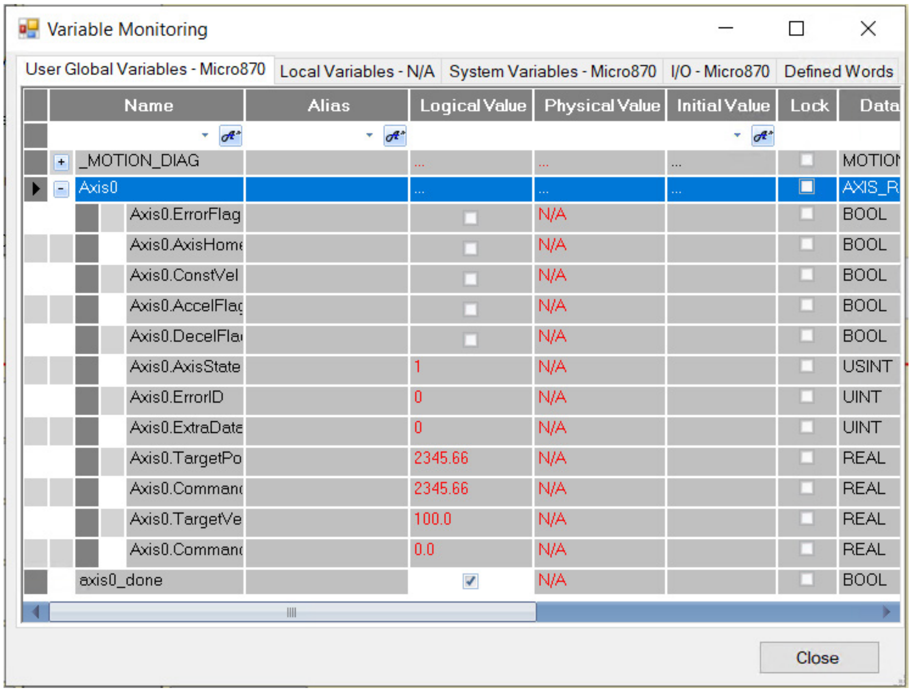
State value	State name
0x00	Disabled
0x01	Standstill
0x02	Discrete Motion
0x03	Continuous Motion

0x04	Homing
0x06	Stopping
0x07	Error Stop

Axis state updates

On motion execution, the axis state update is dependent on when the relevant motion function block is called by the POU scan. This is the case even though the motion profile is controlled by the Motion Engine as a background task, and is independent from the POU scan.

For example, on a moving axis on a Ladder POU (state of a rung=true), an MC_MoveRelative function block in the rung is scanned and the axis starts to move. Before MC_MoveRelative completes, the state of the rung becomes False, and MC_MoveRelative is no longer scanned. In this case, the state of the axis cannot switch from Discrete Motion to StandStill, even after the axis fully stops, and the velocity comes to 0.



Name	Alias	Logical Value	Physical Value	Initial Value	Lock	Data
+ _MOTION_DIAG		...	...	...	□	MOTIOI
- Axis0		...	...	...	□	AXIS_R
Axis0.ErrorFlag		□	N/A		□	BOOL
Axis0.AxisHom		□	N/A		□	BOOL
Axis0.ConstVel		□	N/A		□	BOOL
Axis0.AccelFla		□	N/A		□	BOOL
Axis0.DecelFla		□	N/A		□	BOOL
Axis0.AxisState		1	N/A		□	USINT
Axis0.ErrorID		0	N/A		□	UINT
Axis0.ExtraDat		0	N/A		□	UINT
Axis0.TargetPo		2345.66	N/A		□	REAL
Axis0.Comman		2345.66	N/A		□	REAL
Axis0.TargetVe		100.0	N/A		□	REAL
Axis0.Comman		0.0	N/A		□	REAL
axis0_done		☑	N/A		□	BOOL

Motion control function block parameter numbers

The following function blocks use specific parameter numbers when the function blocks are programmed.

- MC_ReadParameter
- MC_ReadBoolParameter
- MC_WriteParameter
- MC_WriteBoolParameter

Parameter number identification

Parameter numbers between 0 and 999 are reserved for standard parameters. Extensions by a supplier or user are also allowed, although using them can affect portability between different platforms. If the parameter number is greater than 999, the parameter is supplier-specific.

Parameter number	Parameter Name	Data type	R/W	Description
1	Commanded Position	REAL	R	Commanded position.
2	SWLimitPos	REAL	R/W	Positive software limit switch position.
3	SWLimitNeg	REAL	R/W	Negative software limit switch position.
4	EnableLimitPos	BOOL	R/W	Enable positive software limit switch.
5	EnableLimitNeg	BOOL	R/W	Enable negative software limit switch.
8	MaxVelocitySystem	REAL	R	Maximal allowed velocity of the axis in the motion system.
9	MaxVelocityAppl	REAL	R/W	Maximal allowed velocity of the axis in the application.
11	CommandedVelocity	REAL	R	Commanded velocity.
12	MaxAccelerationSystem	REAL	R	Maximal allowed acceleration of the axis in the motion system.
13	MaxAccelerationAppl	REAL	R/W	Maximal allowed acceleration of the axis in the application.
14	MaxDecelerationSystem	REAL	R	Maximal allowed deceleration of the axis in the system.
15	MaxDecelerationAppl	REAL	R/W	Maximal allowed deceleration of the axis in the application.
16	MaxJerk	REAL	R/W	Maximal allowed jerk of the axis.
1001	TargetPosition	REAL	R	The final target position for current active moving function block
1002	TargetVelocity1	REAL	R	The final target velocity for current active moving function block.
1005	Duty Cycle	REAL	R/W	The pulse duty cycle for one pulse. The valid value is 0 – 100, indicating 0% - 100%. (PWM function can be realized by adjusting this value). This parameter is configurable only using this Function Block. The default value is set 50.0 by the controller. **Note**: For Duty Cycle, the value will be overwritten by the default setting, 50.0 when the controller is switched from RUN mode to PRG and RUN again, or when the controller power is cycled.
1006	PulsePerRevolution	REAL	R	The Pulse per Revolution setting input by user in CCW GUI.
1007	TravelPerRevolution	REAL	R	The Travel per Revolution setting input by user in CCW GUI.

Motion control function block error IDs

When a motion control function block ends with an error and the axis state is ErrorStop, use MC_Reset function block or MC_Power Off/On and MC_Reset to recover the axis. The axis can be reset to normal motion operation without stopping the controller operation.

Use this table to help determine the errors for the motion control function blocks.

Value	MACRO ID	Description
00	MC_FB_ERR_ NO	The function block execution is successful.
01	MC_FB_ERR_ WRONG_STATE	The function block cannot execute because the axis is not in the correct state. Check the axis state on page 392.

Value	MACRO ID	Description
02	MC_FB_ERR_ RANGE	The function block cannot execute because there is invalid axis dynamic parameter(s) (velocity, acceleration, deceleration, or jerk) set in the function block. Correct the setting for the dynamic parameters in the function block against Axis Dynamics configuration page.
03	MC_FB_ERR_ PARAM	The function block cannot execute because there is invalid parameter other than velocity, acceleration, deceleration, or jerk, set in the function block. Correct the setting for the parameters (for example, mode or position) for the function block.
04	MC_FB_ERR_ AXISNUM	The function block cannot execute because the axis does not exist, the axis configuration data is corrupted, or the axis is not correctly configured.
05	MC_FB_ERR_ MECHAN	The function block cannot execute because this axis gets a fault due to drive or mechanical issues. Check the connection between the drive and the controller (Drive Ready and In-Position signals), and ensure the drive is operating normally.
06	MC_FB_ERR_ NOPOWER	The function block cannot execute because the axis is not powered on. Power on the axis using MC_Power function block.
07	MC_FB_ERR_ RESOURCE	The function block cannot execute because the resource required by the function block is controlled by some other function block or it is not available. Ensure the resource required by the function block is available for use. Examples: • MC_Power try to control the same axis. • MC_Stop are executed against the same axis at the same time. • MC_TouchProbe are executed against the same axis at the same time.) • MC_TouchProbe is executed, while touch probe input is not enabled in Motion Configuration.
08	MC_FB_ERR_ PROFILE	The function block cannot execute because the motion profile defined in the function block cannot be achieved. Correct the profile in the function block.
09	MC_FB_ERR_ VELOCITY	The function block cannot execute because the motion profile requested in the function block cannot be achieved due to current axis velocity. Examples: • The function block requests the axis to reverse the direction while the axis is moving. • The required motion profile cannot be achieved due to current velocity too low or too high. Check the motion profile setting in the function block, and correct the profile, or re-execute the function block when the axis velocity is compatible with the requested motion profile.
0A	MC_FB_ERR_ SOFT_LIMIT	This function block cannot execute as it will end up moving beyond the Soft Limit, or the function block is cancelled as the Soft Limit has been reached. Check the velocity or target position settings in the function block, or adjust Soft Limit setting.
0B	MC_FB_ERR_ HARD_LIMIT	This function block is cancelled as the Hard Limit switch active state has been detected during axis movement, or cancelled as the Hard Limit switch active state has been detected before axis movement starts. Move the axis away from the Hard Limit switch in the opposite direction.
0C	MC_FB_ERR_ LOG_LIMIT	This function block cannot execute as it will end up moving beyond the PTO Accumulator logic limit, or the function block is cancelled as the PTO Accumulator logic limit has been reached. Check the velocity or target position settings for the function block. Or use MC_SetPosition function block to adjust the axis coordinate system.
0D	MC_FB_ERR_ ERR_ENGINE	A motion engine execution error is detected during the execution of this function block. Power cycle the whole motion setup, including controller, drives and actuators, and re-download the User Application. If the fault persists, call Tech support.

Value	MACRO ID	Description
10	MC_FB_ERR_ NOT_HOMED	The function block cannot execute because the axis need to be homed first. Execute homing against the axis using MC_Home function block.
80	MC_FB_ERR_ PARAM_MODIFIED	Warning: The requested velocity for the axis has been adjusted to a lower value. The function block executes successfully at a lower velocity.

Axis error scenarios

In most cases, when a movement function block instruction issued to an axis results in a function block error, the axis transitions to an Error state, and the corresponding ErrorID element is set on the AXIS_REF data for the axis.

In the following situations, the axis may not transition to an Error state, and it is possible for the user application to issue a successful movement function block to the axis after the axis state changes.

Scenario	Example
A movement function block instructs an axis, but the axis is in a state in which the function block cannot be executed properly.	The axis has no power, or the axis is in a Homing sequence, or in an Error Stop state.
A movement function block instructs an axis, but the axis is still controlled by another movement function block. The axis cannot allow the motion to be controlled by the new function block without going to a full stop.	The new function block commands the axis to change motion direction.
When one movement function block tries to control an axis, but the axis is still controlled by another movement function block, and the newly-defined motion profile cannot be realized by the controller.	User Application issues an S-Curve MC_MoveAbsolute function block to an axis with too short a distance given when the axis is moving.

AXIS_REF data type

The AXIS_REF data type is a data structure that contains information for a motion axis and is used as an input and output variable in all motion control function blocks. An instance of an AXIS_REF data type is automatically created when you add a motion axis to the configuration.

Parameter	Data type	Description
Axis_ID	AXIS_REF	The logic axis ID automatically assigned by Connected Components Workbench. It cannot be edited or viewed by user.
Error Flag	BOOL	Indicates whether an error is present in the axis. Once an axis is flagged with an error, and the error ID is not zero, the axis must be reset using MC_Reset before issuing any other movement function block.
AxisHomed	BOOL	Indicates whether homing operation is successfully executed for the axis or not. When the user tries to redo homing for an axis with AxisHomed already set (homing performed successfully), and the result is not successful, the AxisHomed status is cleared.
ConstVel	BOOL	Indicates whether the axis is in Constant Velocity movement or not. Stationary axis is not considered in Constant Velocity.
AccFlag	BOOL	Indicates whether the axis is in an Accelerating movement or not.
DecelFlag	BOOL	Indicates whether the axis is in an Decelerating movement or not.
AxisState	USINT	Indicates the current state of the axis.
ErrorID	UINT	Indicates the cause for axis error when error is indicated by ErrorFlag. This error usually results from motion control function block execution failure.
ExtraData	UINT	Reserved.

TargetPos	REAL	Indicates the final target position of the axis for MoveAbsolute and MoveRelative function blocks. For MoveVelocity, Stop, and Halt function blocks, TargetPos is 0 except when the TargetPos set by previous position function blocks is not cleared.
CommandPos	REAL	During motion, this is the current position the controller commands the axis to take. There may be a slight delay between the axis actual position and this CommandPos.
TargetVel	REAL	The maximum target velocity instructed to the axis for a moving function block. The value of TargetVel in current function block, or smaller than it, depending on the other parameters in the same function block.
CommandVel	REAL	During motion, this element indicates the current velocity the controller instructs the axis to use. Note that there may be a slight difference between the axis actual velocity and CommandVel, due to the drive delay or drive adjustment overshoot.

FB_AXIS_REF data type

The FB_AXIS_REF data type is a data structure that contains information for a Motion Feedback Axis. It is used as an input and output variable in motion control function blocks. An instance of an FB_AXIS_REF data type is automatically created when you add a HSC module and the mode is configured as Feedback Axis mode.

IMPORTANT Once a Feedback Axis is flagged with an error, and the error ID is not zero, the FBAxis must be reset using MC_Reset before issuing any other movement function block.

Parameter	Data type	Description
Axis_ID	FB_AXIS_REF	The logic axis ID automatically assigned by Connected Components Workbench. It cannot be edited or viewed by user.
ErrorFlag	BOOL	Indicates whether an error is present in the Feedback Axis.
ConstVel	BOOL	Indicates whether the Feedback Axis is in Constant Velocity movement or not. Stationary axis is not considered in Constant Velocity.
AccelFlag	BOOL	Indicates whether the Feedback Axis is in an Accelerating movement or not.
DecelFlag	BOOL	Indicates whether the Feedback Axis is in an Decelerating movement or not.
AxisState	USINT	Indicates the current state of the Feedback Axis.
ErrorID	UINT	Indicates the cause for axis error when error is indicated by ErrorFlag. This error usually results from motion control function block execution failure.
ExtraData	UINT	Reserved.
ActualPos	REAL	Actual mechanical position read back from Motion feedback channel (HSC).
ActualVel	REAL	Actual mechanical velocity read back from Motion feedback channel (HSC).

Axis variables

Axis variables are used to control position, speed, acceleration, and error for a given motion control axis.

To assign a variable to an Axis output parameter:

- In a Function Block Diagram

Graphically connect the Axis output parameter of a motion control function block to the AxisIn input parameter of another motion control function block for convenience. For example, connect MC_POWER Axis output parameter to MC_HOME AxisIn input parameter.

- In a Ladder Diagram

  A variable can not be assigned to the Axis output parameter of a motion control function block because it is read-only.

Monitor an AXIS_REF variable

Monitor an AXIS_REF or a FBAXIS_REF variable in the software while connected to the controller, when the motion engine is active, or in the user application as part of user logic. You can also monitor the AXIS_REF or FBAXIS_REF variable remotely through various communication channels.

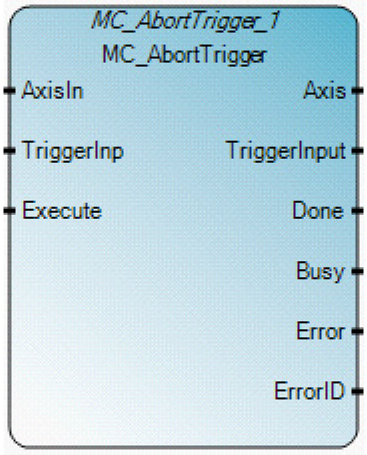
MC_AbortTrigger (motion control abort trigger)

Cancels Motion function blocks that are connected to trigger events.

MC_AbortTrigger only executes when assigned to an axis that is controlled by MC_TouchProbe.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro830, Micro850 and Micro870 controllers that support motion control.

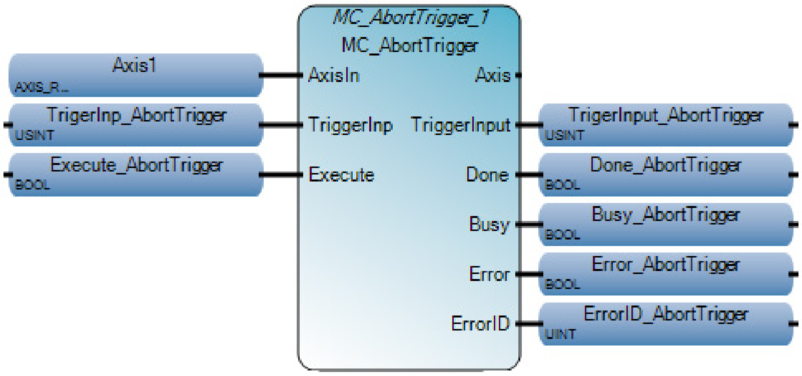


Use this table to help determine the parameter values for this instruction.

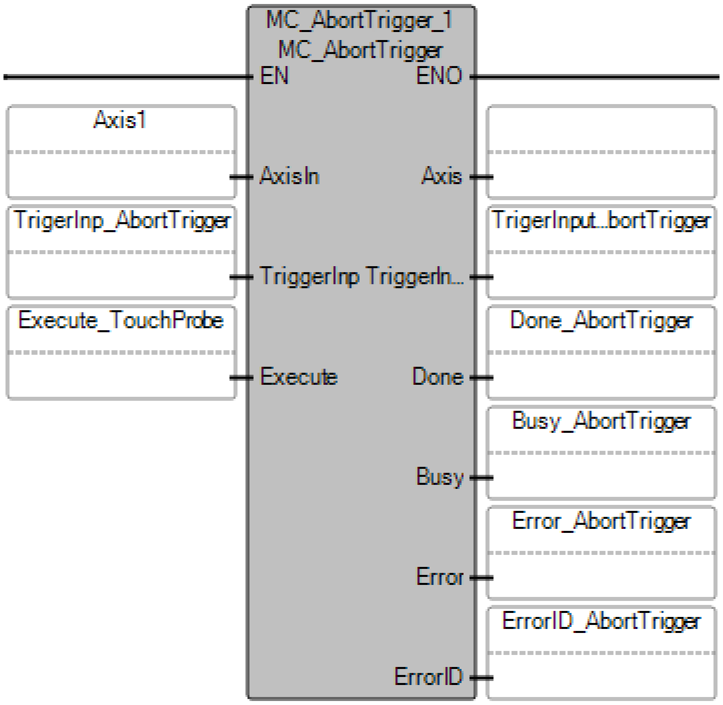
Parameter	Parameter	Data type	Description
EN	Input	BOOL	Instruction block enable. TRUE - execute current MC_AbortTrigger computation. FALSE - there is no computation. Applies only to Ladder Diagram programs.

AxisIn	Input	AXIS_REF on page 397 FB_AXIS_REF on page 398	Use the AXIS_REF data type to define the AxisIn parameter. If the axis is a FB_Axis (feedback axis), use the FB_AXIS_REF data type to define AxisIn parameters.
TriggerInp	Input	USINT	This parameter is ignored.
Execute	Input	BOOL	When TRUE, cancels the trigger event at the rising edge.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.
Axis	Output	AXIS_REF	Axis output is read-only in Ladder Diagram programs.
TriggerInput	Output	USINT	This parameter is ignored.
Done	Output	BOOL	TRUE - the trigger event is cancelled.
Busy	Output	BOOL	TRUE - the function block is not finished.
Error	Output	BOOL	TRUE - an error is detected.
ErrorID	Output	UINT	A unique numeric that identifies the error. The errors for this instruction are defined in Motion control function block error IDs.

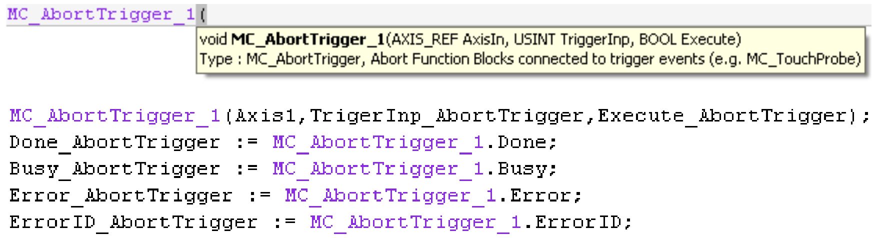
MC_AbortTrigger Function Block Diagram example



MC_AbortTrigger Ladder Diagram example

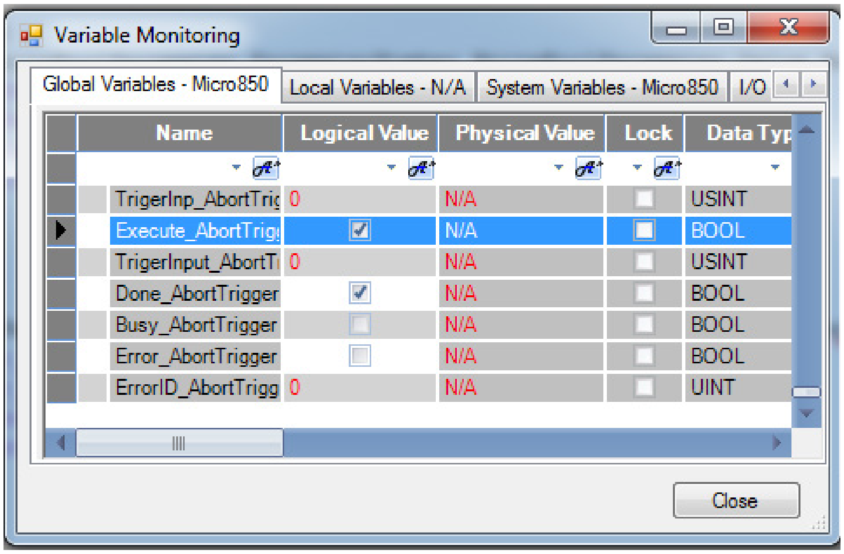


MC_AbortTrigger Structured Text example



```
MC_AbortTrigger_1(
void MC_AbortTrigger_1(AXIS_REF AxisIn, USINT TriggerInp, BOOL Execute)
Type : MC_AbortTrigger, Abort Function Blocks connected to trigger events (e.g. MC_TouchProbe)

MC_AbortTrigger_1(Axis1,TrigerInp_AbortTrigger,Execute_AbortTrigger);
Done_AbortTrigger := MC_AbortTrigger_1.Done;
Busy_AbortTrigger := MC_AbortTrigger_1.Busy;
Error_AbortTrigger := MC_AbortTrigger_1.Error;
ErrorID_AbortTrigger := MC_AbortTrigger_1.ErrorID;
```

Results



Name	Logical Value	Physical Value	Lock	Data Typ
TrigerInp_AbortTrig	0	N/A	□	USINT
Execute_AbortTrig	☑	N/A	□	BOOL
TrigerInput_AbortTi	0	N/A	□	USINT
Done_AbortTrigger	☑	N/A	□	BOOL
Busy_AbortTrigger	□	N/A	□	BOOL
Error_AbortTrigger	□	N/A	□	BOOL
ErrorID_AbortTrigg	0	N/A	□	UINT

MC_Halt (motion control halt)

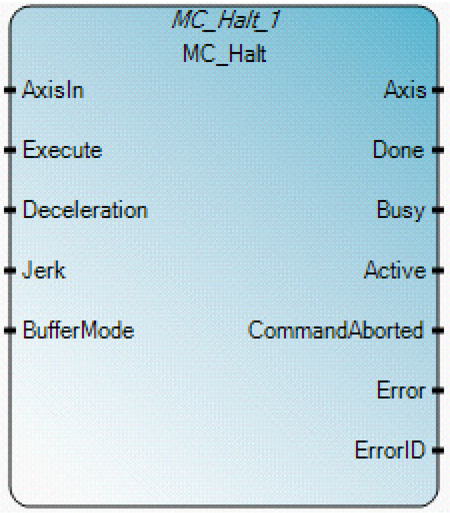
Commands a controlled motion stop under normal operating conditions.

Operating details;

- The axis state changes to DiscreteMotion, until velocity is zero. When velocity reaches zero, Done is set to True and the axis state changes to StandStill.
  - It is possible to execute another motion command during deceleration of the axis, which overrides MC_Halt.
  - If MC_Halt is issued when the axis state is Homing, the instruction block reports an error, and the homing process is not interrupted.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro830, Micro850 and Micro870 controllers that support motion control.

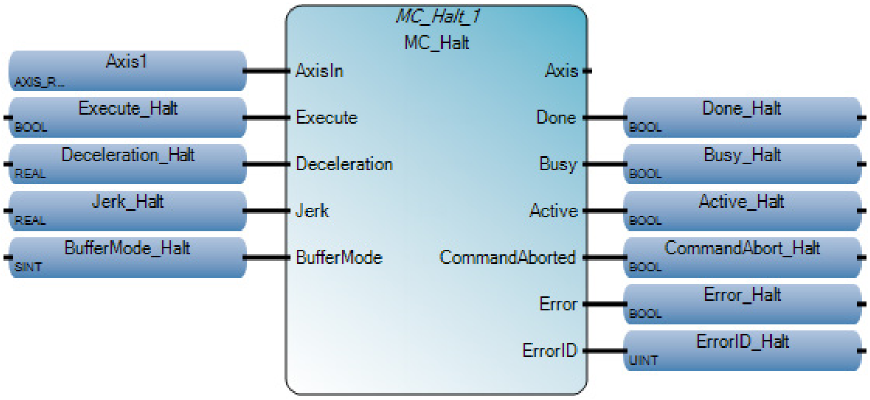


Use this table to help determine the parameter values for this instruction.

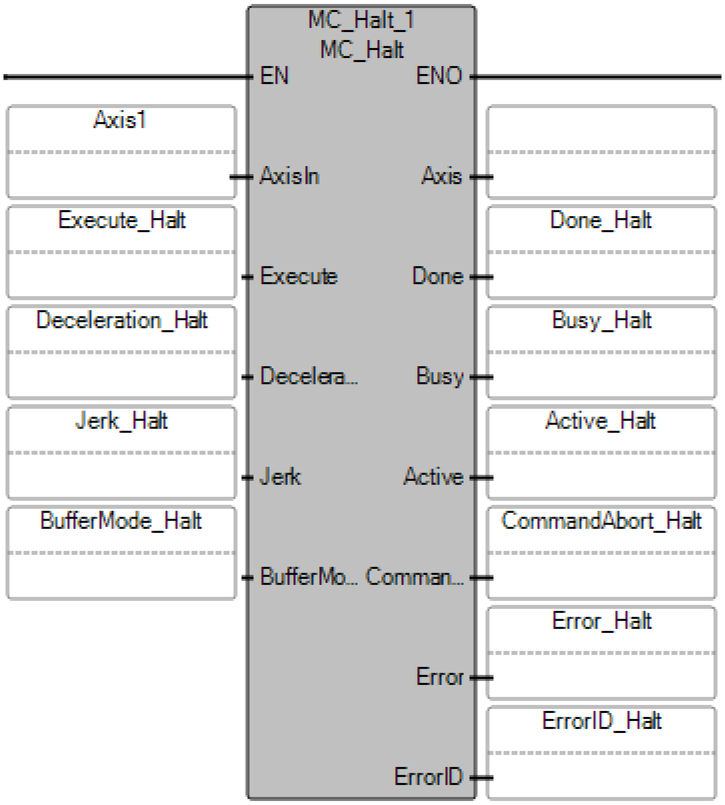
	Parameter type	Data type	Description
EN	Input	BOOL	Instruction block enable. TRUE - execute current MC_Halt computation. FALSE - there is no computation. Applies to Ladder Diagram programs.
AxisIn	Input	AXIS_REF	Use the AXIS_REF data type parameters to define AxisIn.
Execute	Input	BOOL	Indicates when to start motion. TRUE - start the motion at rising edge. FALSE - do not start motion. Executing MC_Halt during homing, MC_Halt is set to MC_FB_ERR_STATE and the homing process continues.
Deceleration	Input	REAL	Value of the deceleration (always positive) (decreasing energy of the motor). If Deceleration <= 0 and the axis state is not Standstill, MC_Halt is set to MC_FB_ERR_RANGE.
Jerk	Input	REAL	Value of the Jerk (always positive). If Jerk < 0 and the axis state is Standstill, MC_Halt is set to MC_FB_ERR_RANGE.
BufferMode	Input	SINT	Not used. The mode is always MC_Aborting.

	Parameter type	Data type	Description
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.
Axis	Output	AXIS_REF on page 397	Axis output is read-only in Ladder Diagram programs.
Done	Output	BOOL	Zero velocity reached.
Busy	Output	BOOL	The instruction block is not finished.
Active	Output	BOOL	Indicates that the instruction block has control on the axis.
CommandAborted	Output	BOOL	Command is overridden by another command, or error stop.
Error	Output	BOOL	Indicates an error occurred. TRUE - An error is detected. FALSE - No error.
ErrorID	Output	UINT	A unique numeric that identifies the error. The errors for this instruction are defined in Motion control function block error IDs on page 395.

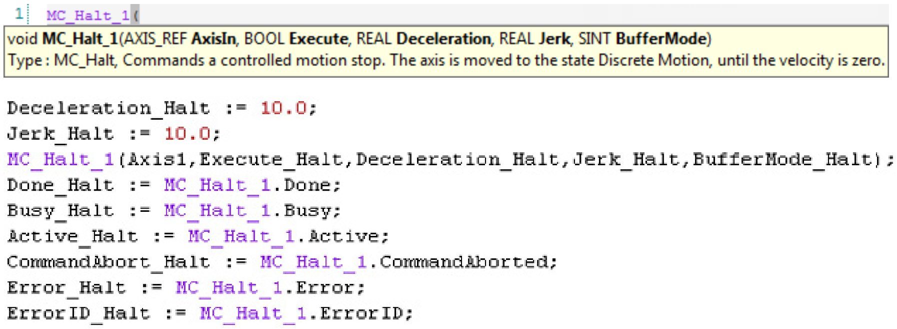
MC_Halt Function Block Diagram example



MC_Halt Ladder Diagram example



MC_Halt Structured Text example

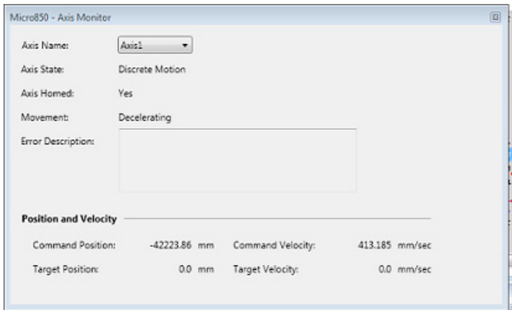


```
MC_Halt_1(
```

void **MC_Halt_1**(AXIS_REF **AxisIn**, BOOL **Execute**, REAL **Deceleration**, REAL **Jerk**, SINT **BufferMode**)
Type : MC_Halt, Commands a controlled motion stop. The axis is moved to the state Discrete Motion, until the velocity is zero.

```
Deceleration_Halt := 10.0;
Jerk_Halt := 10.0;
MC_Halt_1(Axis1,Execute_Halt,Deceleration_Halt,Jerk_Halt,BufferMode_Halt);
Done_Halt := MC_Halt_1.Done;
Busy_Halt := MC_Halt_1.Busy;
Active_Halt := MC_Halt_1.Active;
CommandAbort_Halt := MC_Halt_1.CommandAborted;
Error_Halt := MC_Halt_1.Error;
ErrorID_Halt := MC_Halt_1.ErrorID;
```

Results

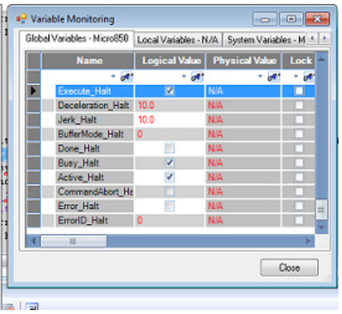


Micro850 - Axis Monitor

Axis Name: Axis1
Axis State: Discrete Motion
Axis Homed: Yes
Movement: Decelerating
Error Description:

Position and Velocity

Command Position: -42223.86 mm Command Velocity: 413.185 mm/sec
Target Position: 0.0 mm Target Velocity: 0.0 mm/sec



Variable Monitoring

Global Variables - Micro850 | Local Variables - N/A | System Variables - M

Name	Logical Value	Physical Value	Lock
Execute_Halt	☑	N/A	□
Deceleration_Halt	10.0	N/A	□
Jerk_Halt	10.0	N/A	□
BufferMode_Halt	0	N/A	□
Done_Halt	□	N/A	□
Busy_Halt	☑	N/A	□
Active_Halt	☑	N/A	□
CommandAbort_Ha	□	N/A	□
Error_Halt	□	N/A	□
ErrorID_Halt	0	N/A	□

MC_Home (motion control home)

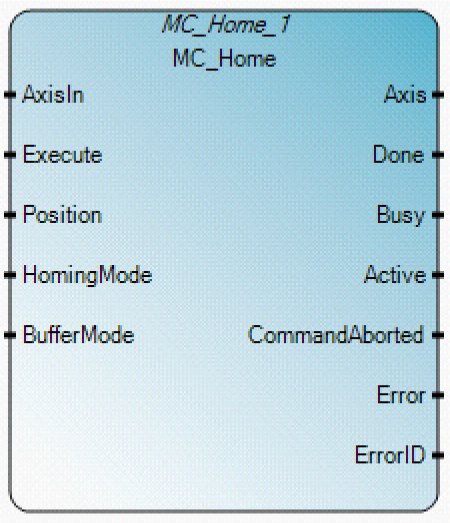
Commands the axis to perform the <*search home*> sequence. The details of this sequence are manufacturer dependent and can be set by the axis parameters. The Position input is used to set the absolute position when a reference signal is detected, and the configured Home offset is reached.

Operation details:

- After MC_Power is issued, the axis Homed status is reset to 0 (not homed). In most cases, after the axis is powered on, the MC_Home function block needs to be executed to calibrate the axis position and the Home reference.
- The MC_Home function block can only be cancelled using a MC_Stop or a MC_Power function block. If it is cancelled before it completes, the previously searched Home position is considered invalid and the axis Homed status is cleared.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro830, Micro850 and Micro870 controllers that support motion control.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter type	Data type	Description
EN	Input	BOOL	Instruction block enable. TRUE - execute current MC_Home computation. FALSE - there is no computation. Applies only to Ladder Diagram programs.
AxisIn	Input	AXIS_REF	Use the AXIS_REF data type parameters to define AxisIn.
Execute	Input	BOOL	Indicates when to start motion. TRUE - start the motion at rising edge. FALSE - do not start motion.

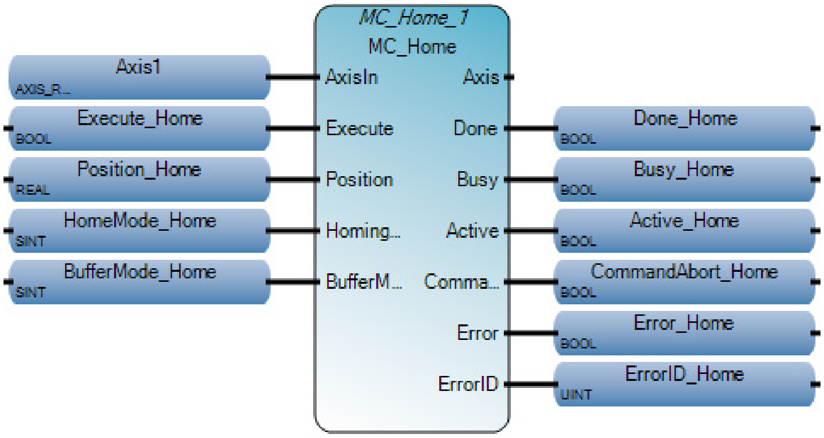
Parameter	Parameter type	Data type	Description
Position	Input	REAL	Absolute position is set when the reference signal is detected and configured Home offset is reached. The value range for this input is -0x40000000 – 0x40000000 physical pulse after the position is converted from user position unit to PTO pulse. Set the Position value within the Soft Limit. An invalid input value results in an error. Error ID = MC_FB_ERR_PARAM.
HomingMode	Input	SINT	Enum input for Homing mode.
BufferMode	Input	SINT	Not used. The mode is always mcAborting.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.
Axis	Output	AXIS_REF on page 397	Axis output is read-only in Ladder Diagram programs.
Done	Output	BOOL	TRUE - the Homing operation completed successfully and the axis state is set to StandStill. FALSE - Homing operation is ongoing or incomplete.
Busy	Output	BOOL	TRUE - the instruction block is not finished. FALSE - the instruction block is finished.
Active	Output	BOOL	TRUE - indicates that the instruction block has control on the axis.
CommandAborted	Output	BOOL	TRUE - command was overridden by another command, or error stop.
Error	Output	BOOL	Indicates an error occurred. TRUE - An error is detected. FALSE - No error.
ErrorID	Output	UNIT	A unique numeric that identifies the error. The errors for this instruction are defined in Motion control function block error IDs on page 395.

HomingModes

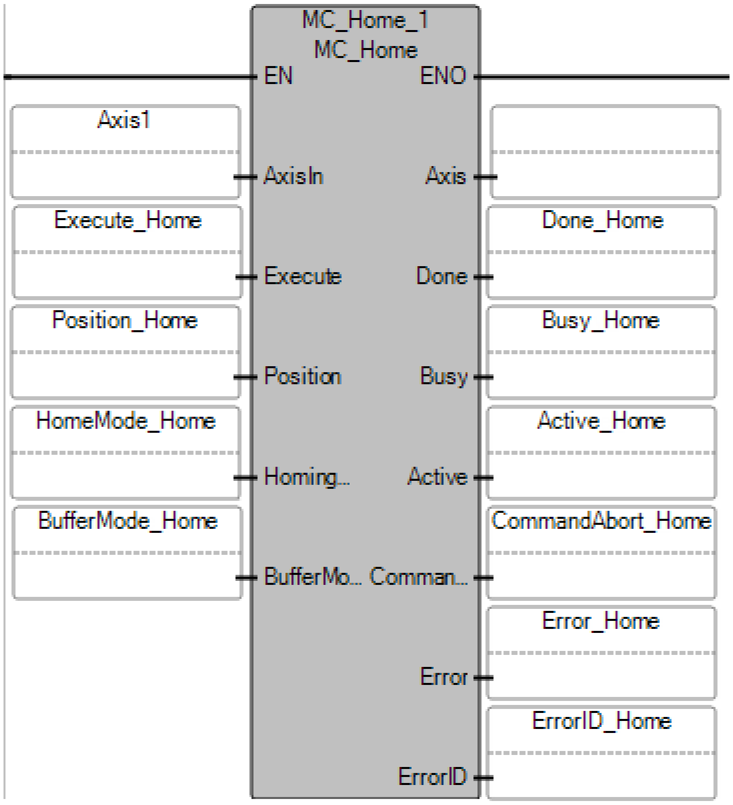
Use this table to help determine the values for the HomingMode parameter in the MC_Home motion control instruction on page 389.

Value	Name	Description
0x00	MC_HOME_ABS_SWITCH	Homing process by searching Home Absolute switch
0x01	MC_HOME_LIMIT_SWITCH	Homing process by searching limit switch
0x02	MC_HOME_REF_WITH_ABS	Homing process by searching Home Absolute switch plus using encoder reference pulse
0x03	MC_HOME_REF_PULSE	Homing process by searching limit switch plus using encoder reference pulse
0x04	MC_HOME_DIRECT	Static homing process with direct forcing a home position from user reference. The function block will set current position the mechanism is in as home position, with its position determined by the input parameter, "Position"

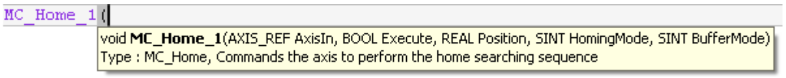
MC_Home Function Block Diagram example



MC_Home Ladder Diagram example



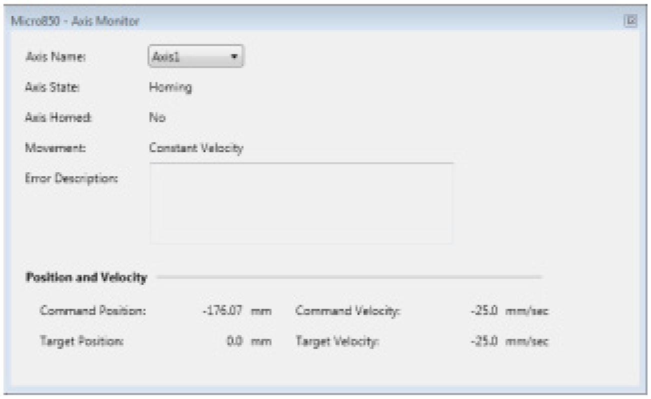
MC_Home Structured Text example



```
MC_Home_1(
void MC_Home_1(AXIS_REF AxisIn, BOOL Execute, REAL Position, SINT HomingMode, SINT BufferMode)
Type : MC_Home, Commands the axis to perform the home searching sequence
```

```
Position_Home := -50000.0;
HomeMode_Home := 4;  (*1*)
MC_Home_1(Axis1,Execute_Home,Position_Home,HomeMode_Home,BufferMode_Home);
Done_Home := MC_Home_1.Done;
Busy_Home := MC_Home_1.Busy;
Active_Home := MC_Home_1.Active;
CommandAbort_Home := MC_Home_1.CommandAborted;
Error_Home := MC_Home_1.Error;
ErrorID_Home := MC_Home_1.ErrorID;
```

Results



Micro850 - Axis Monitor

Axis Name: Axis1

Axis State: Homing

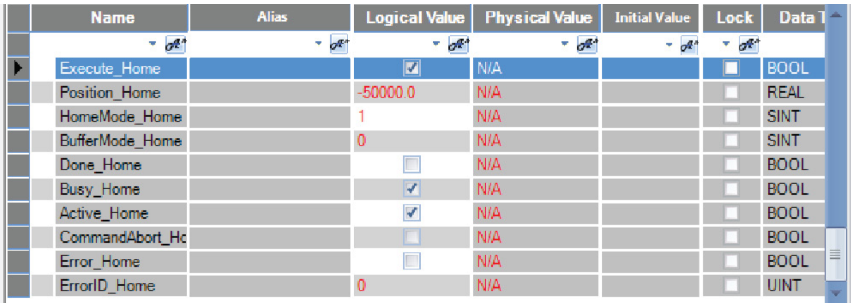
Axis Homed: No

Movement: Constant Velocity

Error Description:

Position and Velocity

Command Position:	-176.07 mm	Command Velocity:	-25.0 mm/sec
Target Position:	0.0 mm	Target Velocity:	-25.0 mm/sec



Name	Alias	Logical Value	Physical Value	Initial Value	Lock	Data T
Execute_Home		☑	N/A		□	BOOL
Position_Home		-50000.0	N/A		□	REAL
HomeMode_Home		1	N/A		□	SINT
BufferMode_Home		0	N/A		□	SINT
Done_Home		□	N/A		□	BOOL
Busy_Home		☑	N/A		□	BOOL
Active_Home		☑	N/A		□	BOOL
CommandAbort_Hc		□	N/A		□	BOOL
Error_Home		□	N/A		□	BOOL
ErrorID_Home		0	N/A		□	UINT

MC_MoveAbsolute (motion control move absolute)

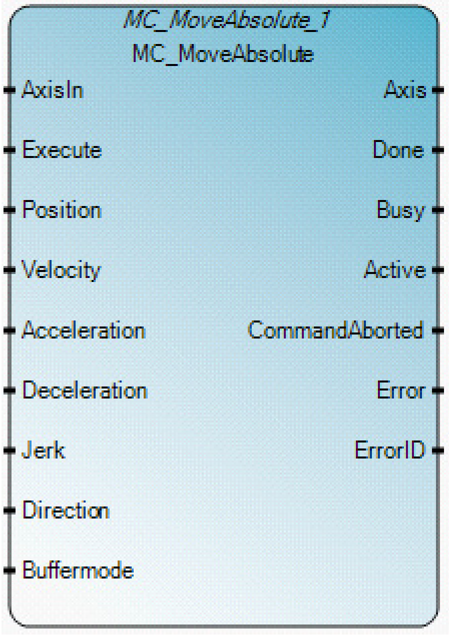
Commands a controlled motion to a specified absolute position.

Operation details:

- For the Micro800 controller,
  - The sign of the input Velocity for a MC_MoveAbsolute function block is ignored because the motion direction is determined by the Current position and the Target position.
  - The input Direction for a MC_MoveAbsolute function block is ignored because there is only one mathematical solution to reach the Target position.
- If the MC_MoveAbsolute function block is issued when the Micro800 controller axis state is StandStill and the relative distance to move is zero, the execution of the function block is immediately reported as Done.
- If a MC_MoveAbsolute function block is issued to an axis that is not in the Homed, position, the function block reports an error.
- The MoveAbsolute function block completes with Velocity zero if it is not overridden by another function block.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro830, Micro850 and Micro870 controllers that support motion control.

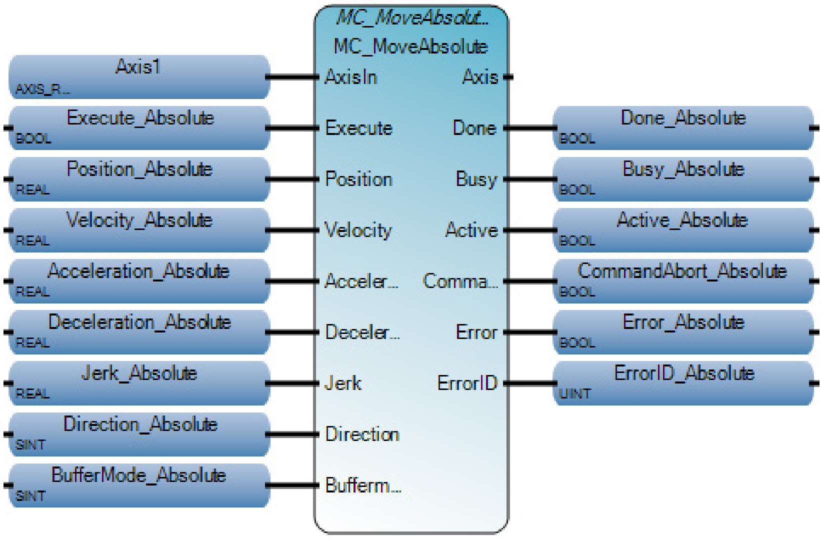


Use this table to help determine the parameter values for this instruction.

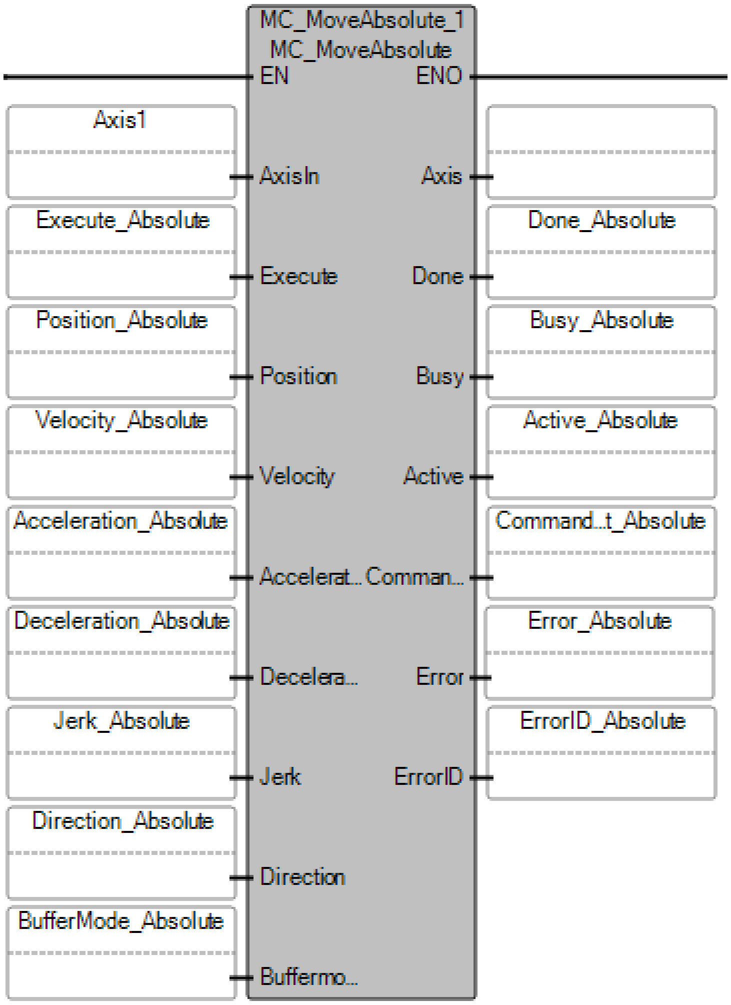
Parameter	Parameter type	Data type	Description
EN	Input	BOOL	Instruction block enable. TRUE - execute current MC_MoveAbsolute computation. FALSE - there is no computation. Applies only to Ladder Diagram programs.
AxisIn	Input	AXIS_REF	Use the AXIS_REF data type parameters to define AxisIn.
Execute	Input	BOOL	Indicates when to start motion. TRUE - start the motion at rising edge. FALSE - do not start motion. The axis should be in the home position when this execute command is issued or an error occurs, MC_FB_ERR_NOT_HOMED.
Position	Input	REAL	Target position for the motion in technical unit (negative or positive). The technical unit is defined in the **Motion - General** configuration page for an axis.
Velocity	Input	REAL	Value of the maximum velocity. The maximum velocity may not be reached when Jerk = 0. The sign of Velocity is ignored, the motion direction is determined by the input Position.
Acceleration	Input	REAL	Value of the acceleration (always positive - increasing energy to the motor.) user unit/sec^2
Deceleration	Input	REAL	Value of the deceleration (always positive - decreasing energy to the motor). u/sec^2

Parameter	Parameter type	Data type	Description
Jerk	Input	REAL	Value of the Jerk (always positive). u/sec^3 When the value of the input Jerk = 0, the Trapezoid profile is calculated by Motion Engine. When Jerk > 0, the S-Curve profile is calculated.
Direction	Input	SINT	This parameter is not used.
BufferMode	Input	SINT	This parameter is not used.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.
Axis	Output	AXIS_REF on page 397	Axis output is read-only in Ladder Diagram programs.
Done	Output	BOOL	When TRUE, command position reached. When the In-Position Input is configured as Enabled for this axis, the drive needs to set In-Position Input signal active before this Done bit goes to True. This action completes with velocity zero unless it is cancelled.
Busy	Output	BOOL	When TRUE, the function block is not finished.
Active	Output	BOOL	When TRUE, indicates that the function block has control of the axis
CommandAborted	Output	BOOL	When TRUE, the Command was overridden by another command, or error stop.
Error	Output	BOOL	Indicates an error occurred. TRUE - An error is detected. FALSE - No error.
ErrorID	Output	UINT	A unique numeric that identifies the error. The errors for this instruction are defined in Motion control function block error IDs on page 395.

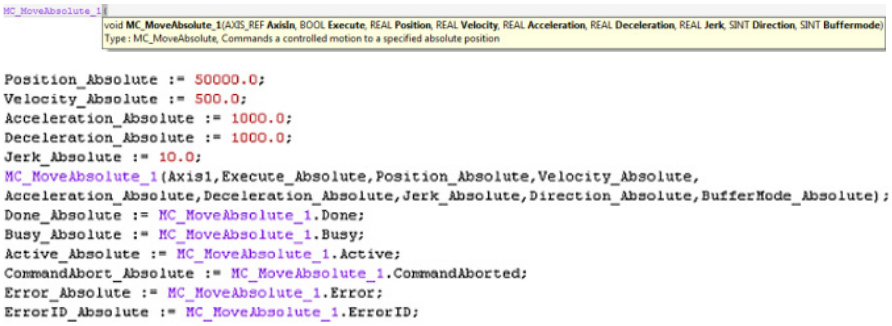
MC_MoveAbsolute Function Block Diagram example



MC_MoveAbsolute Ladder Diagram example



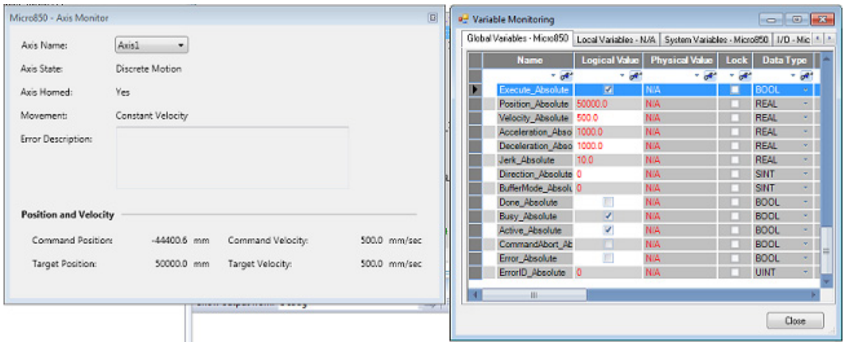
MC_MoveAbsolute Structured Text example



void **MC_MoveAbsolute_1**(AXIS_REF **AxisIn**, BOOL **Execute**, REAL **Position**, REAL **Velocity**, REAL **Acceleration**, REAL **Deceleration**, REAL **Jerk**, SINT **Direction**, SINT **Buffermode**)
Type : MC_MoveAbsolute, Commands a controlled motion to a specified absolute position

```
Position_Absolute := 50000.0;
Velocity_Absolute := 500.0;
Acceleration_Absolute := 1000.0;
Deceleration_Absolute := 1000.0;
Jerk_Absolute := 10.0;
MC_MoveAbsolute_1(Axis1,Execute_Absolute,Position_Absolute,Velocity_Absolute,
Acceleration_Absolute,Deceleration_Absolute,Jerk_Absolute,Direction_Absolute,BufferMode_Absolute);
Done_Absolute := MC_MoveAbsolute_1.Done;
Busy_Absolute := MC_MoveAbsolute_1.Busy;
Active_Absolute := MC_MoveAbsolute_1.Active;
CommandAbort_Absolute := MC_MoveAbsolute_1.CommandAborted;
Error_Absolute := MC_MoveAbsolute_1.Error;
ErrorID_Absolute := MC_MoveAbsolute_1.ErrorID;
```

Results



Micro850 - Axis Monitor

Axis Name: Axis1

Axis State: Discrete Motion

Axis Homed: Yes

Movement: Constant Velocity

Error Description:

Position and Velocity

Command Position:	-44400.6	mm	Command Velocity:	500.0	mm/sec
Target Position:	50000.0	mm	Target Velocity:	500.0	mm/sec

Variable Monitoring

Global Variables - Micro850 | Local Variables - N/A | System Variables - Micro850 | I/O - Mic

Name	Logical Value	Physical Value	Lock	Data Type
Execute_Absolute	☑	N/A	□	BOOL
Position_Absolute	50000.0	N/A	□	REAL
Velocity_Absolute	500.0	N/A	□	REAL
Acceleration_Abso	1000.0	N/A	□	REAL
Deceleration_Abso	1000.0	N/A	□	REAL
Jerk_Absolute	10.0	N/A	□	REAL
Direction_Absolute	0	N/A	□	SINT
BufferMode_Absolu	0	N/A	□	SINT
Done_Absolute	□	N/A	□	BOOL
Busy_Absolute	☑	N/A	□	BOOL
Active_Absolute	☑	N/A	□	BOOL
CommandAbort_Ab	□	N/A	□	BOOL
Error_Absolute	□	N/A	□	BOOL
ErrorID_Absolute	0	N/A	□	UINT

MC_MoveRelative (motion control move relative)

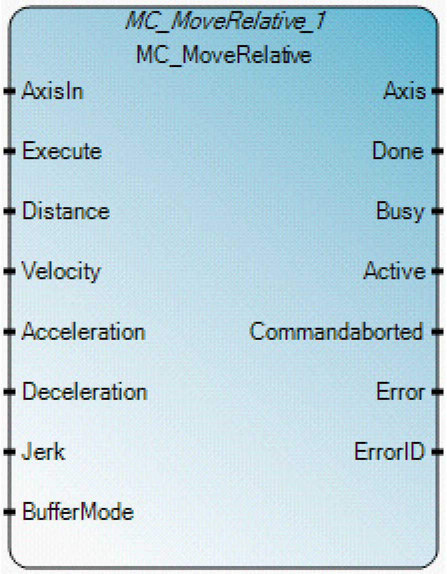
Commands a controlled motion of a specified distance relative to the actual position at the time of the execution.

Operation details:

- Because the motion direction for MC_MoveRelative is determined by the current position and the target position, the sign of the Velocity is ignored.
- MoveRelative completes with Velocity zero if it is not overridden by another function block.
- If MC_MoveRelative is issued when the Micro800 controller axis state is StandStill and the relative distance to move is zero, the execution of the function block is immediately reported as Done.
- For a Micro800 controller, the sign of the input Velocity for MC_MoveRelative is ignored because the motion direction is determined by the Current position and the Target position.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro830, Micro850 and Micro870 controllers that support motion control.

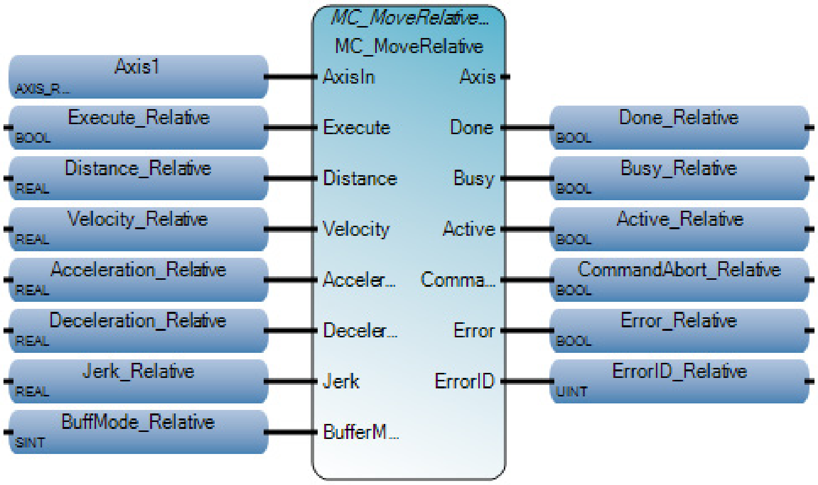


MC_MoveRelative operation

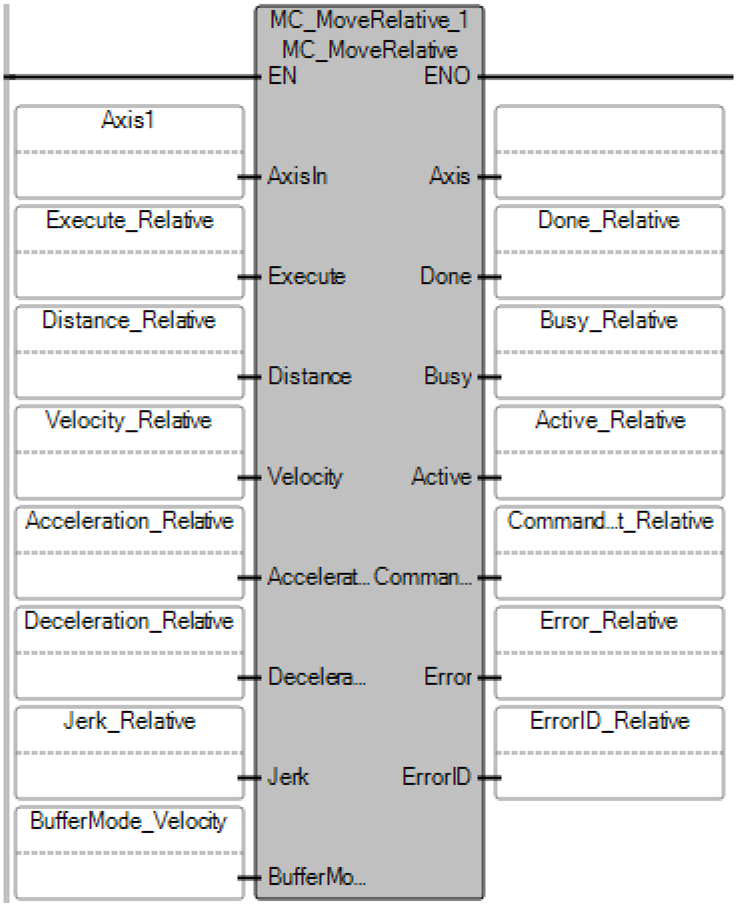
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter type	Data type	Description
EN	Input	BOOL	Instruction block enable. TRUE - execute current MC_MoveRelative computation. FALSE - there is no computation. Applies only to Ladder Diagram programs.
AxisIn	Input	AXIS_REF on page 397	Use the AXIS_REF data type parameters to define AxisIn.
Execute	Input	BOOL	Indicates when to start motion. TRUE - start the motion at rising edge. FALSE - do not start motion.
Distance	Input	REAL	Relative distance for the motion (in technical unit [u]).
Velocity	Input	REAL	Value of the maximum velocity (not necessarily reached) [u/s]. As the motion direction is determined by input Position, the sign of Velocity is ignored by the function block. The maximum velocity may not be reached when Jerk = 0.
Acceleration	Input	REAL	Value of the acceleration (increasing energy of the motor) [u/s2]
Deceleration	Input	REAL	Value of the deceleration (decreasing energy of the motor) [u/s2]
Jerk	Input	REAL	Value of the Jerk [u/s^3]
BufferMode	Input	SINT	This parameter is not used.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.
Axis	Output	AXIS_REF	Axis output is read-only in Ladder Diagram programs.
Done	Output	BOOL	TRUE - commanded distance reached. When the In-Position input is enabled for an axis, the In-Position signal must be set to active before Done = True.
Busy	Output	BOOL	TRUE - the instruction block is not finished. FALSE - the instruction block is finished.
Active	Output	BOOL	TRUE - indicates that the instruction block has control on the axis.
CommandAborted	Output	BOOL	TRUE - command was ovrerridden by another command, or Error Stop.
Error	Output	BOOL	Indicates an error occurred. TRUE - An error is detected. FALSE - No error.
ErrorID	Output	UINT	A unique numeric that identifies the error. The errors for this instruction are defined in Motion control function block error IDs on page 395.

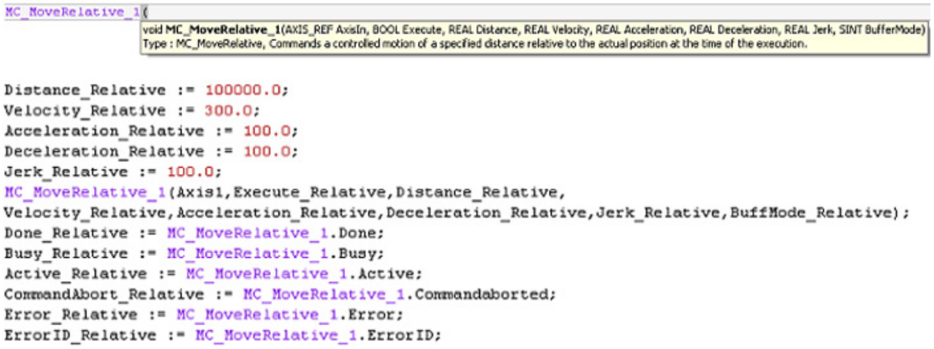
MC_MoveRelative Function Block Diagram example



MC_MoveRelative Ladder Diagram example

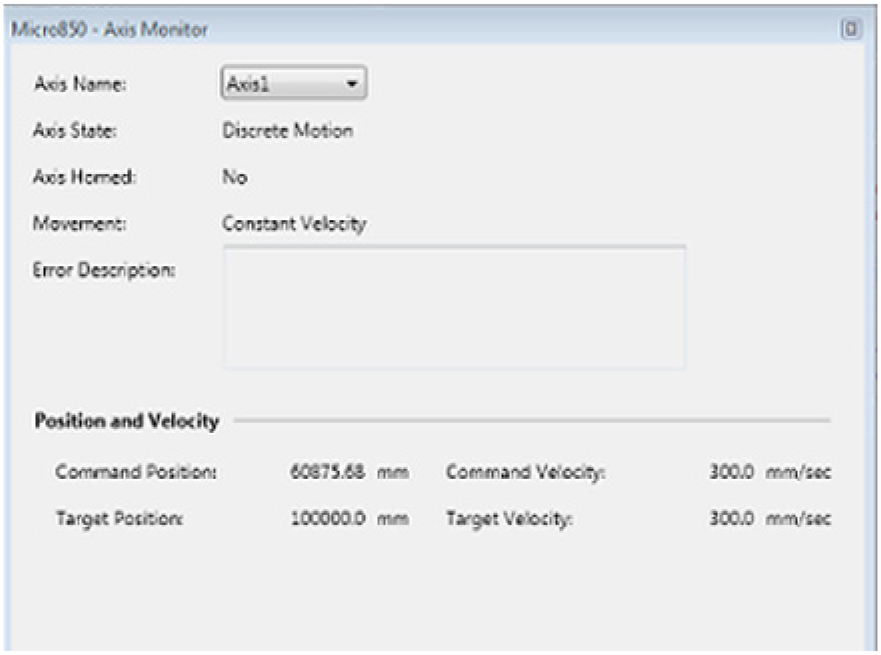


MC_MoveRelative Structured Text example



```
MC_MoveRelative_1(
                void MC_MoveRelative_1(AXIS_REF AxisIn, BOOL Execute, REAL Distance, REAL Velocity, REAL Acceleration, REAL Deceleration, REAL Jerk, SINT BufferMode)
                Type : MC_MoveRelative, Commands a controlled motion of a specified distance relative to the actual position at the time of the execution.

Distance_Relative := 100000.0;
Velocity_Relative := 300.0;
Acceleration_Relative := 100.0;
Deceleration_Relative := 100.0;
Jerk_Relative := 100.0;
MC_MoveRelative_1(Axis1,Execute_Relative,Distance_Relative,
Velocity_Relative,Acceleration_Relative,Deceleration_Relative,Jerk_Relative,BuffMode_Relative);
Done_Relative := MC_MoveRelative_1.Done;
Busy_Relative := MC_MoveRelative_1.Busy;
Active_Relative := MC_MoveRelative_1.Active;
CommandAbort_Relative := MC_MoveRelative_1.Commandaborted;
Error_Relative := MC_MoveRelative_1.Error;
ErrorID_Relative := MC_MoveRelative_1.ErrorID;
```

Results



Axis Name: Axis1

Axis State: Discrete Motion

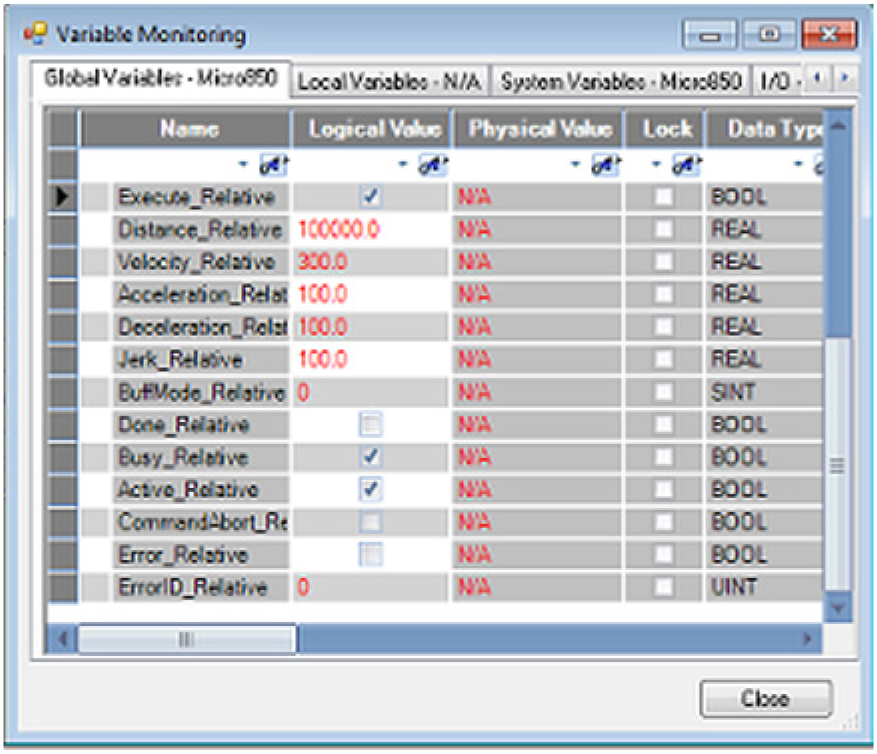
Axis Homed: No

Movement: Constant Velocity

Error Description:

Position and Velocity

Command Position:	60875.68 mm	Command Velocity:	300.0 mm/sec
Target Position:	100000.0 mm	Target Velocity:	300.0 mm/sec



Name	Logical Value	Physical Value	Lock	Data Typ
Execute_Relative	☑	N/A	□	BOOL
Distance_Relative	100000.0	N/A	□	REAL
Velocity_Relative	300.0	N/A	□	REAL
Acceleration_Relat	100.0	N/A	□	REAL
Deceleration_Relat	100.0	N/A	□	REAL
Jerk_Relative	100.0	N/A	□	REAL
BuffMode_Relative	0	N/A	□	SINT
Done_Relative	□	N/A	□	BOOL
Busy_Relative	☑	N/A	□	BOOL
Active_Relative	☑	N/A	□	BOOL
CommandAbort_Re	□	N/A	□	BOOL
Error_Relative	□	N/A	□	BOOL
ErrorID_Relative	0	N/A	□	UINT

MC_MoveVelocity (motion control move velocity)

Commands a never ending controlled motion at a specified velocity.

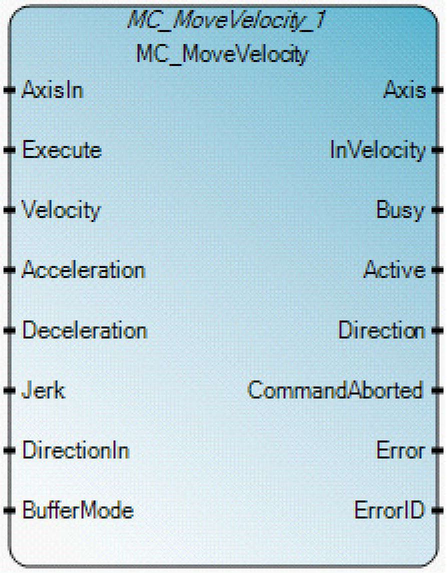
Operation details:

- If the DirectionIn input for MC_MoveVelocity is equal to 0 and:
  - the axis is in a moving state, the sign of the Velocity input is ignored, the axis continues to move in its current direction, and new dynamic parameters are applied.
  - the axis is not in a moving state, MC_MoveVolecity reports an error.

- If the PTO Pulse limit is reached during execution of MC_MoveVelocity, the PTO Accumulator value is rolled over to 0 (or to the opposite Soft Limit if the limit is activated) and execution continues.
- If the axis is in a moving state, and MC_MoveVelocity issues a motion in which the direction (the sign of Velocity * Direction) is the opposite of the current motion direction, the MC_MoveVelocity reports an error.
- Once the signal 'InVelocity' is set, it indicates MC_MoveVelocity is complete. Any subsequent motion event has no effect on the MC_MoveVelocity outputs except the signal 'InVelocity'.
- The InVelocity output of MC_MoveVelocity stays True once the Velocity of the axis reaches the commanded Velocity or until MC_MoveVelocity is stopped.
- The sign of (Velocity * Direction) determines the motion direction for MC_MoveVelocity. If the Velocity sign and the Direction sign are the same, positive motion is issued. If the Velocity sign and the Direction sign are different, negative motion is issued.
- The signal 'InVelocity' is reset when MC_MoveVelocity is overridden by another function block or Motion event, or at the falling edge of 'Execute'.
- To stop or change the motion initiated by MC_MoveVelocity, the instruction block must be interrupted or overridden by another instruction block, which includes executing MC_MoveVelocity again with different parameters.
- If MC_MoveVelocity is issued with the axis state in StandStill (not controlled by another function block) and a function block error occurs, the axis state goes to ErrorStop.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro830, Micro850 and Micro870 controllers that support motion control.

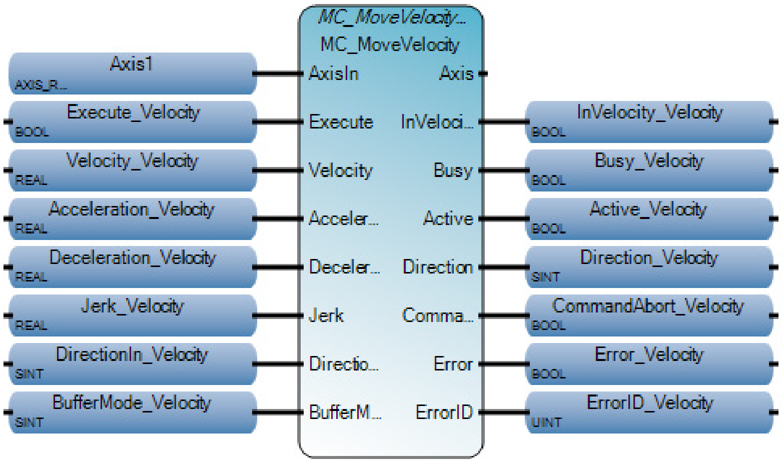


Use this table to help determine the parameter values for this instruction.

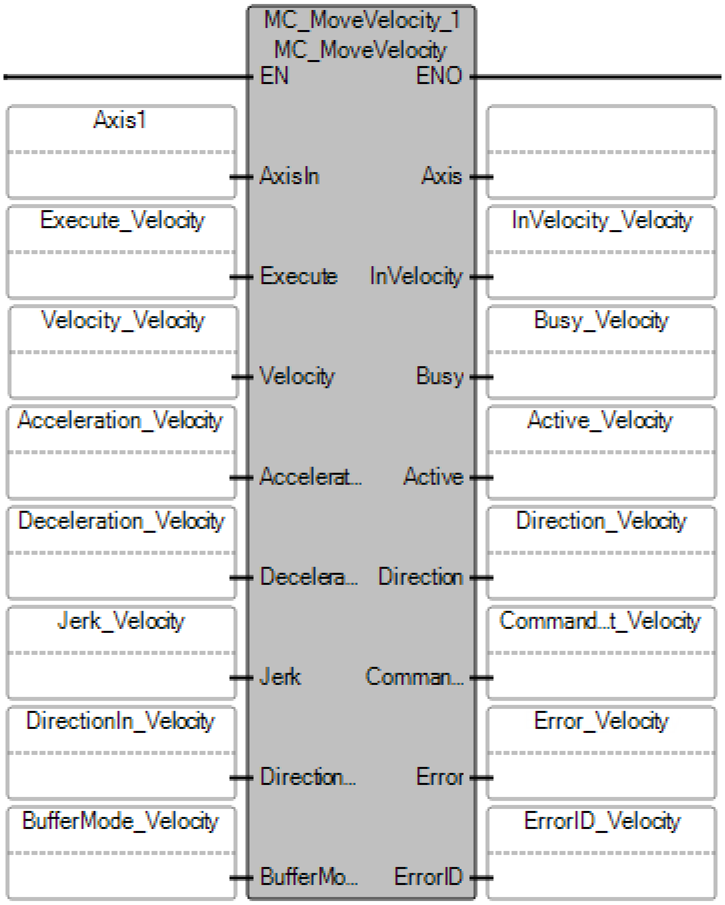
Parameter	Parameter type	Data type	Description
EN	Input	BOOL	Instruction block enable. TRUE - execute current MC_MoveVelocity computation. FALSE - there is no computation. Applies only to Ladder Diagram programs.
AxisIn	Input	AXIS_REF	Use the AXIS_REF data type parameters to define AxisIn.
Execute	Input	BOOL	Indicates when to start motion. TRUE - start the motion at rising edge. FALSE - do not start motion.
Velocity	Input	REAL	Value of the maximum velocity [u/s].
Acceleration	Input	REAL	Value of the acceleration (increasing energy of the motor) [u/s^2]
Deceleration	Input	REAL	Value of the deceleration (decreasing energy of the motor) [u/s^2]
Jerk	Input	REAL	Value of the Jerk [u/s^3]
DirectionIn	Input	SINT	The valid values are: -1, 0, 1.
BufferMode	Input	SINT	This parameter is not used.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.
Axis	Output	AXIS_REF on page 397	Axis output is read-only in Ladder Diagram programs.
InVelocity	Output	BOOL	TRUE - commanded velocity was reached (first time).
Busy	Output	BOOL	TRUE - the instruction block is not finished. FALSE - the instruction block is finished.
Active	Output	BOOL	TRUE - indicates the function block has control on the axis.
Direction	Output	SINT	The valid values are: -1, 0, 1.
CommandAborted	Output	BOOL	TRUE - command was overridden by another command, or Error Stop.
Error	Output	BOOL	Indicates an error occurred. TRUE - An error is detected. FALSE - No error.

Parameter	Parameter type	Data type	Description
ErrorID	Output	UINT	A unique numeric that identifies the error. The errors for this instruction are defined in Motion control function block error IDs on page 395.

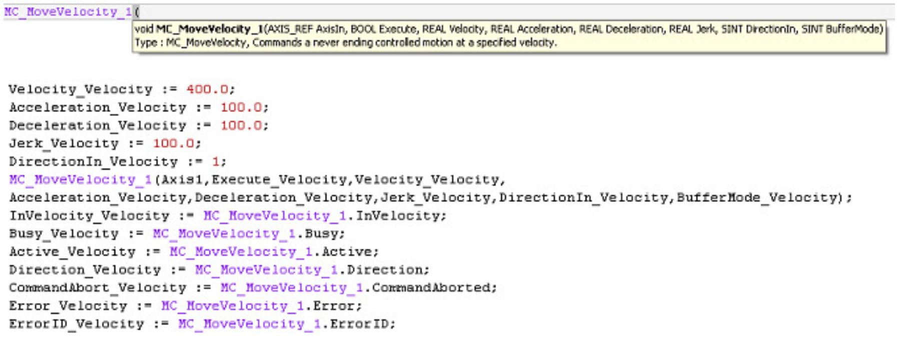
MC_MoveVelocity Function Block Diagram example



MC_MoveVelocity Ladder Diagram example

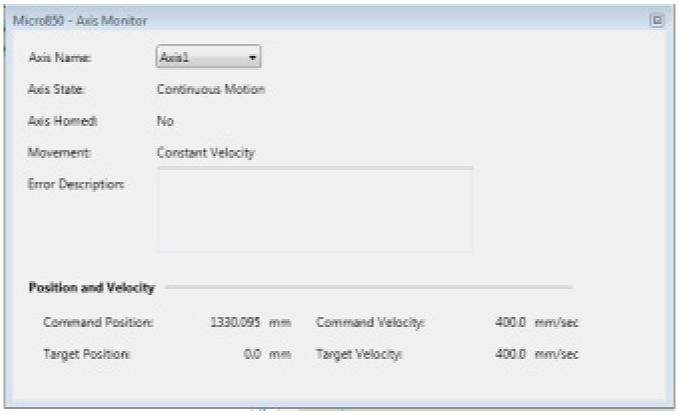


MC_MoveVelocity Structured Text example



```
MC_MoveVelocity_1(
void MC_MoveVelocity_1(AXIS_REF AxisIn, BOOL Execute, REAL Velocity, REAL Acceleration, REAL Deceleration, REAL Jerk, SINT DirectionIn, SINT BufferMode)
Type : MC_MoveVelocity, Commands a never ending controlled motion at a specified velocity.

Velocity_Velocity := 400.0;
Acceleration_Velocity := 100.0;
Deceleration_Velocity := 100.0;
Jerk_Velocity := 100.0;
DirectionIn_Velocity := 1;
MC_MoveVelocity_1(Axis1,Execute_Velocity,Velocity_Velocity,
Acceleration_Velocity,Deceleration_Velocity,Jerk_Velocity,DirectionIn_Velocity,BufferMode_Velocity);
InVelocity_Velocity := MC_MoveVelocity_1.InVelocity;
Busy_Velocity := MC_MoveVelocity_1.Busy;
Active_Velocity := MC_MoveVelocity_1.Active;
Direction_Velocity := MC_MoveVelocity_1.Direction;
CommandAbort_Velocity := MC_MoveVelocity_1.CommandAborted;
Error_Velocity := MC_MoveVelocity_1.Error;
ErrorID_Velocity := MC_MoveVelocity_1.ErrorID;
```

Results



Micro850 - Axis Monitor

Axis Name: Axis1

Axis State: Continuous Motion

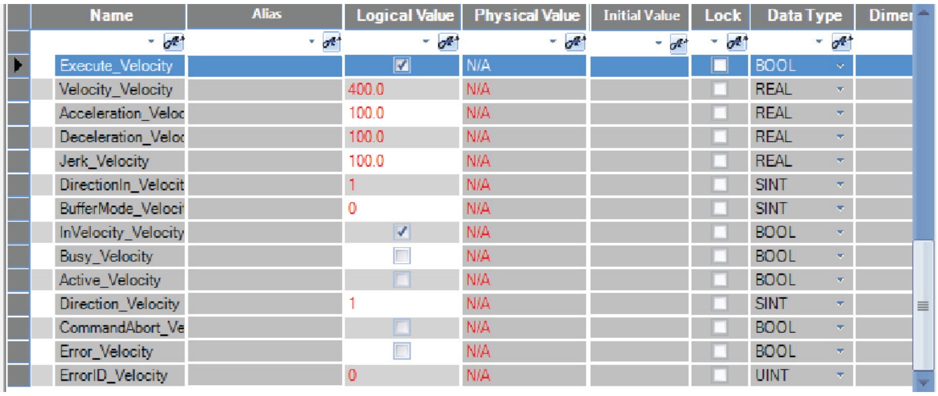
Axis Homed: No

Movement: Constant Velocity

Error Description:

Position and Velocity

Command Position:	1330.095 mm	Command Velocity:	400.0 mm/sec
Target Position:	0.0 mm	Target Velocity:	400.0 mm/sec



Name	Alias	Logical Value	Physical Value	Initial Value	Lock	Data Type	Dimer
Execute_Velocity		☑	N/A		□	BOOL	
Velocity_Velocity		400.0	N/A		□	REAL	
Acceleration_Veloc		100.0	N/A		□	REAL	
Deceleration_Veloc		100.0	N/A		□	REAL	
Jerk_Velocity		100.0	N/A		□	REAL	
DirectionIn_Velocit		1	N/A		□	SINT	
BufferMode_Veloci		0	N/A		□	SINT	
InVelocity_Velocity		☑	N/A		□	BOOL	
Busy_Velocity		□	N/A		□	BOOL	
Active_Velocity		□	N/A		□	BOOL	
Direction_Velocity		1	N/A		□	SINT	
CommandAbort_Ve		□	N/A		□	BOOL	
Error_Velocity		□	N/A		□	BOOL	
ErrorID_Velocity		0	N/A		□	UINT	

MC_Power (motion control power)

Controls the power stage, ON or OFF.

Operation details:

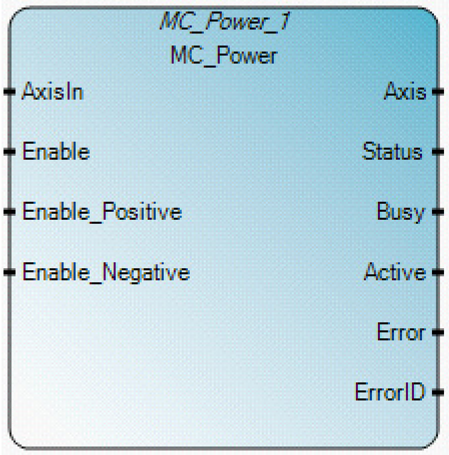
- If you import a project created in CCW 7 into CCW 8, the Mc_Power new input parameter, __DTI_AxisIn shows. If a Build error occurs, reselect the instruction and rebuild.
- After axis power On completes, the axis Homed status is reset to 0 (not homed).
- The Enable_Positive input and the Enable_Negative input for MC_Power are both level triggered and are checked when the Enable input changes from OFF to ON. The on-the-fly change for the

Enable_Positive input and the Enable_Negative input without Enable input toggling is not checked.

- If power fails during operation (when Servo ready has been detected) the axis state changes to ErrorStop.
- The MC_Power instruction has a time out value of 2 minutes. MC_Power returns an error when the time out period expires and Drive Ready Input is FALSE.
- If an MC_Power function block with Enable set to True is called while the axis state is Disabled, the axis state changes to StandStill unless an error is detected, in which case the axis state changes to ErrorStop.
- Only one MC_Power function block should be issued per axis. If a different MC_ Power function block is used to control the same axis simultaneously, it is rejected by the Motion Engine.
- When there is a Power On or Off state switch for an axis, the absolute axis position is not reset.
- If an MC_Power function block with Enable set to False is called, the axis state changes to Disabled for every state including ErrorStop.
- The MC_Power function block can power on the axis if Enable is set to True and power off the axis if Enable is set to False.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro830, Micro850 and Micro870 controllers that support motion control.

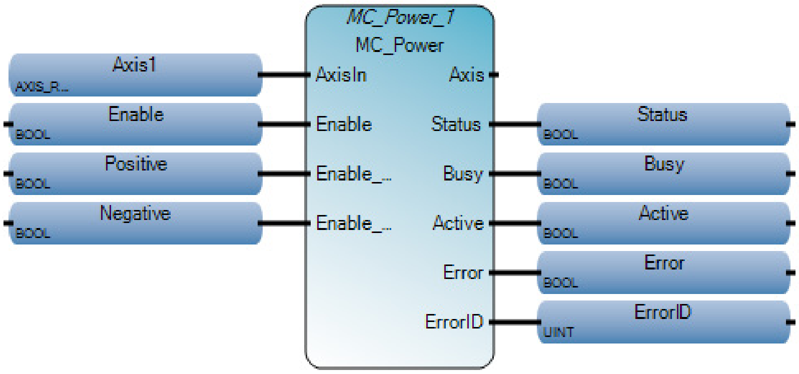


Use this table to help determine the parameter values for this instruction.

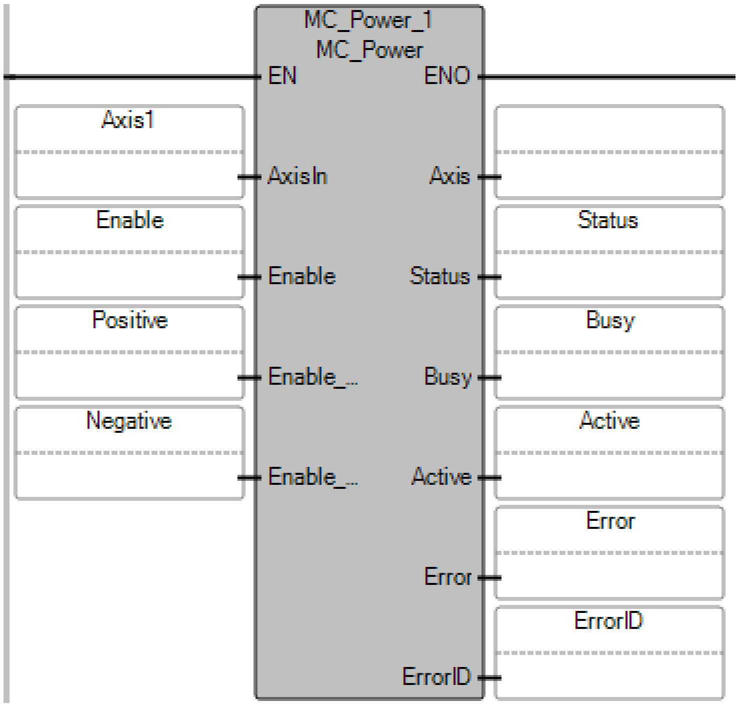
Parameter	Parameter type	Data type	Description
EN	Input	BOOL	Instruction block enable. TRUE - execute current MC_Power computation. FALSE - there is no computation. Applies only to Ladder Diagram programs.
AxisIn	Input	AXIS_REF FB_AXIS_REF	Use the AXIS_REF data type on page 397 parameters to define AxisIn. For an FB_Axis (feedback axis), use the FB_AXIS_REF data type on page 398 to define AxisIn.

Parameter	Parameter type	Data type	Description
Enable	Input	BOOL	TRUE - power is ON. FALSE - power is OFF
Enable_Positive	Input	BOOL	TRUE - motion is allowed in the positive direction.
Enable_Negative	Input	BOOL	TRUE - motion is allowed in the negative direction.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.
Axis	Output	AXIS_REF	Axis output is read-only in Ladder Diagram programs. AXIS_REF data type.
Status	Output	BOOL	State of the power stage: ● TRUE - drive power on is done. ● FALSE - drive power on is not done.
Busy	Output	BOOL	TRUE - the instruction block is not finished. FALSE - the instruction block is finished.
Active	Output	BOOL	TRUE - indicates the function block has control on the axis.
Error	Output	BOOL	Indicates an error occurred. TRUE - An error is detected. FALSE - No error.
ErrorID	Output	UINT	A unique numeric that identifies the error. The errors for this instruction are defined in Motion control function block error IDs on page 395.

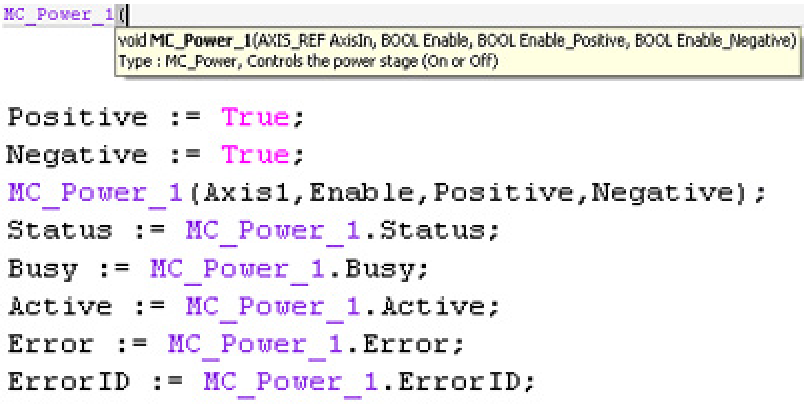
MC_Power Function Block Diagram example



MC_Power Ladder Diagram example



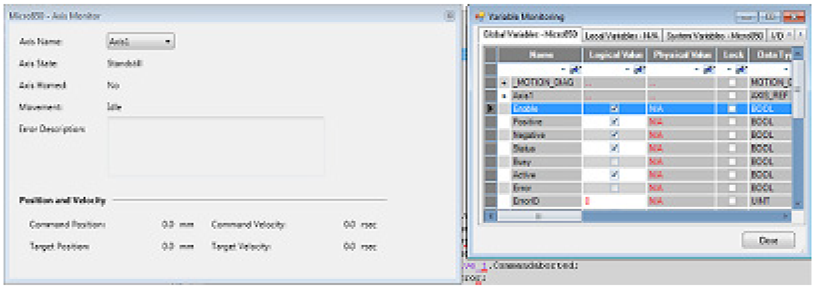
MC_Power Structured Text example



```
MC_Power_1(
void MC_Power_1(AXIS_REF AxisIn, BOOL Enable, BOOL Enable_Positive, BOOL Enable_Negative)
Type : MC_Power, Controls the power stage (On or Off)
```

```
Positive := True;
Negative := True;
MC_Power_1(Axis1,Enable,Positive,Negative);
Status := MC_Power_1.Status;
Busy := MC_Power_1.Busy;
Active := MC_Power_1.Active;
Error := MC_Power_1.Error;
ErrorID := MC_Power_1.ErrorID;
```

Results



Micro850 - Axis Monitor

Axis Name: Axis1

Axis State: Standstill

Axis Homed: No

Movement: Idle

Error Description:

Position and Velocity

Command Position: 0.0 mm Command Velocity: 0.0 mm/sec

Target Position: 0.0 mm Target Velocity: 0.0 mm/sec

Variable Monitoring

Global Variables - Micro850 | Local Variables : N/A | System Variables - Micro850 | I/O

Name	Logical Value	Physical Value	Lock	Data Type
_MOTION_DIAG	...	...	□	MOTION_D
Axis1	...	...	□	AXIS_REF
Enable	☑	N/A	□	BOOL
Positive	☑	N/A	□	BOOL
Negative	☑	N/A	□	BOOL
Status	☑	N/A	□	BOOL
Busy	□	N/A	□	BOOL
Active	☑	N/A	□	BOOL
Error	□	N/A	□	BOOL
ErrorID	0	N/A	□	UINT

MC_ReadActualPosition (motion control read actual position)

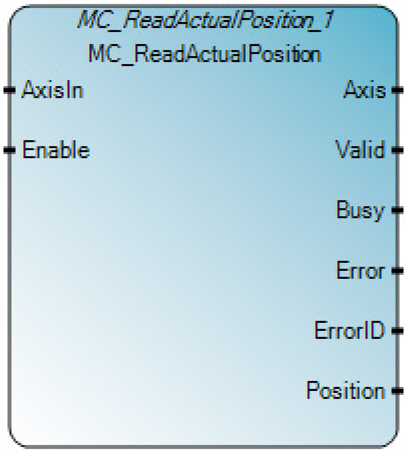
Returns the actual position of the feedback axis. MC_ReadActualPosition is only applicable to feedback motion.

Operation details:

- Before executing MC_ReadActualPosition, verify the axis is in one of the following Axis States:
    - Disabled
    - Standstill
    - Discrete Motion
    - Error Stop
- The actual position for a feedback axis is not reset to 0 after a download. To reset or clear the position for a feedback axis use the MC_Home instruction or MC_SetPosition instruction.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro830, Micro850 and Micro870 controllers that support motion control.

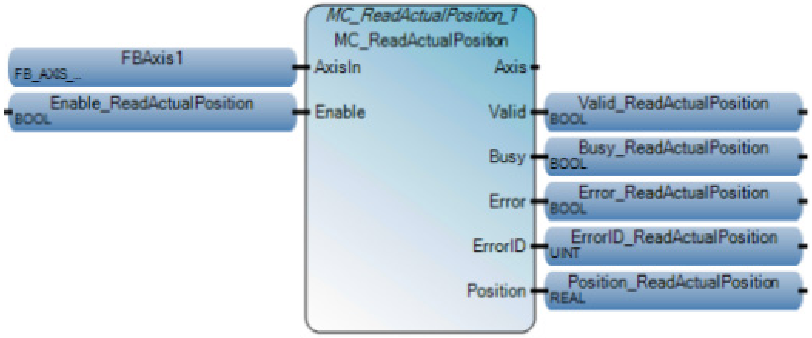


Use this table to help determine the parameter values for this instruction.

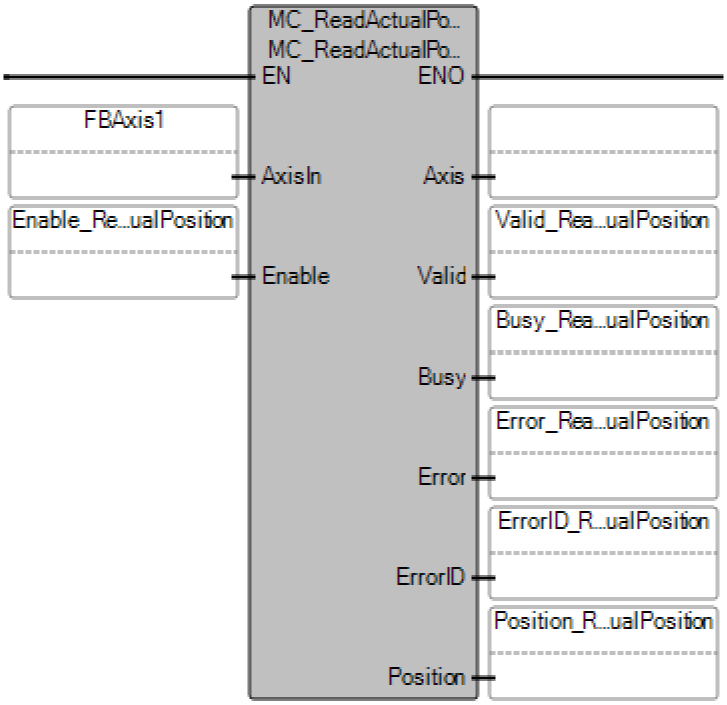
Parameter	Parameter type	Data type	Description
AxisIn	Input	FB_AXIS_REF	For an FB_Axis (feedback axis), use the FB_AXIS_REF data type on page 398 to define AxisIn.
Enable	Input	BOOL	TRUE - get the value of the parameter continuously while enabled. FALSE - idle.
Axis	Output	FB_AXIS_REF	Axis output is read-only in Ladder Diagram programs. The Axis output parameters are defined in the FB_AXIS_REF data type.
Valid	Output	BOOL	TRUE - the instruction block is active and new output values are expected. FALSE - the instruction block is inactive.
Busy	Output	BOOL	TRUE - the instruction block is not finished. FALSE - the instruction block is finished.
Error	Output	BOOL	Indicates an error occurred. TRUE - An error is detected. FALSE - No error.

Parameter	Parameter type	Data type	Description
ErrorID	Output	UNIT	A unique numeric that identifies the error. The errors for this instruction are defined in Motion control function block error IDs on page 395.
Position	Output	REAL	The value of the actual absolute position for feedback motion Axis. (in axis' unit [u])

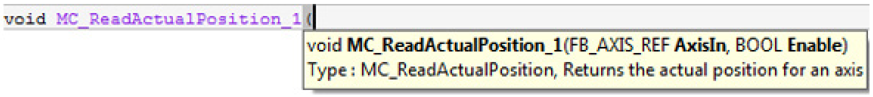
MC_ReadActualPosition Function Block Diagram example



MC_ReadActualPosition Ladder Diagram example

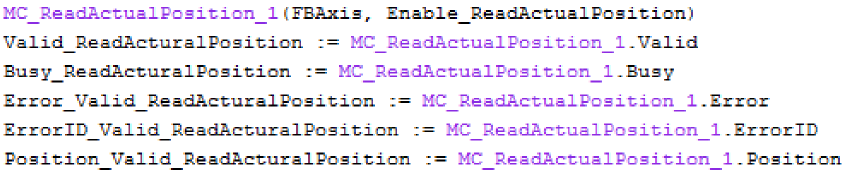


MC_ReadActualPosition Structured Text example



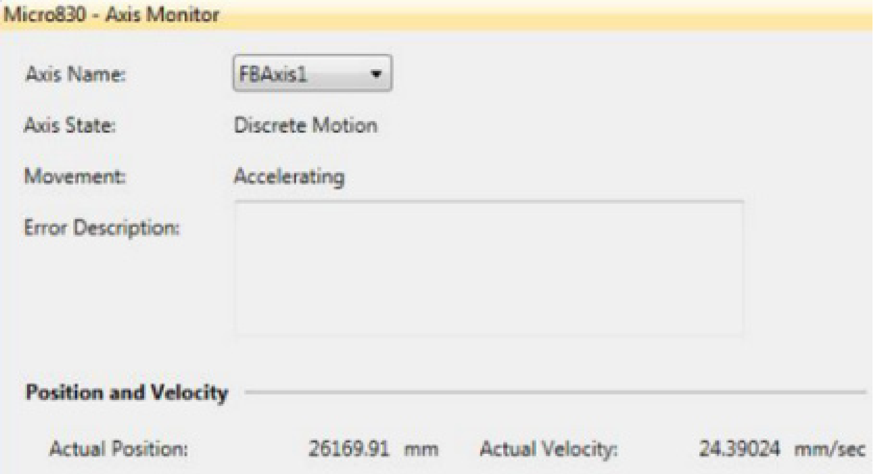
```
void MC_ReadActualPosition_1(
```

void **MC_ReadActualPosition_1**(FB_AXIS_REF **AxisIn**, BOOL **Enable**)
Type : MC_ReadActualPosition, Returns the actual position for an axis



```
MC_ReadActualPosition_1(FBAxis, Enable_ReadActualPosition)
Valid_ReadActuralPosition := MC_ReadActualPosition_1.Valid
Busy_ReadActuralPosition := MC_ReadActualPosition_1.Busy
Error_Valid_ReadActuralPosition := MC_ReadActualPosition_1.Error
ErrorID_Valid_ReadActuralPosition := MC_ReadActualPosition_1.ErrorID
Position_Valid_ReadActuralPosition := MC_ReadActualPosition_1.Position
```

Results


Micro830 - Axis Monitor

Axis Name: FBAxis1

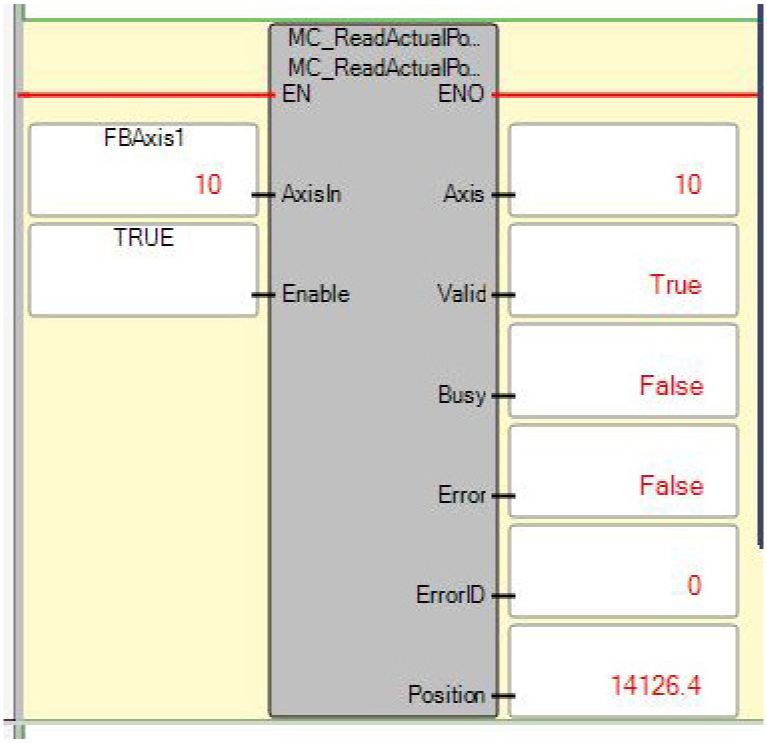
Axis State: Discrete Motion

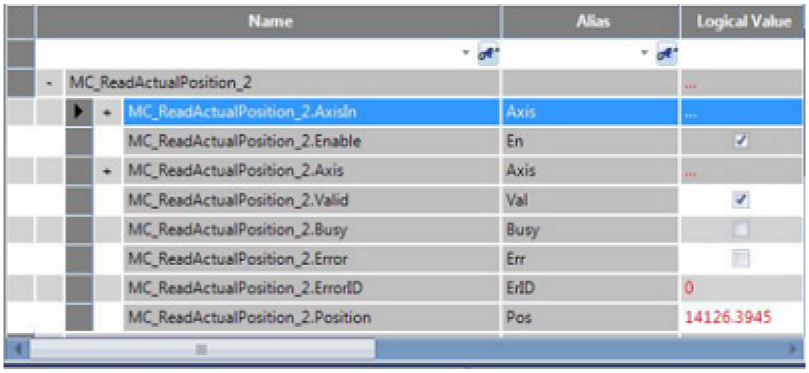
Movement: Accelerating

Error Description:

Position and Velocity

Actual Position: 26169.91 mm Actual Velocity: 24.39024 mm/sec





Name	Alias	Logical Value
- MC_ReadActualPosition_2		...
+ MC_ReadActualPosition_2.AxisIn	Axis	...
MC_ReadActualPosition_2.Enable	En	✓
+ MC_ReadActualPosition_2.Axis	Axis	...
MC_ReadActualPosition_2.Valid	Val	✓
MC_ReadActualPosition_2.Busy	Busy	☐
MC_ReadActualPosition_2.Error	Err	☐
MC_ReadActualPosition_2.ErrorID	ErID	0
MC_ReadActualPosition_2.Position	Pos	14126.3945

MC_ReadActualVelocity (motion control read actual velocity)

Returns the value of the actual velocity of the feedback axis.

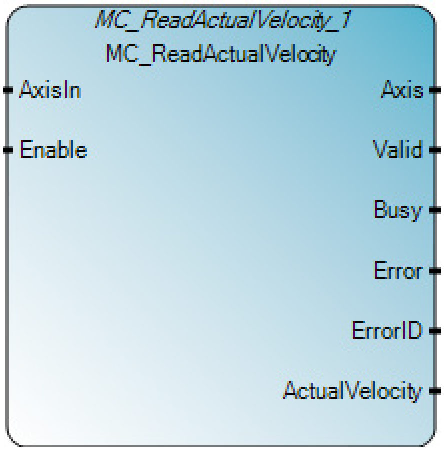
MC_ReadActualVelocity is only applicable to feedback motion.

Before executing MC_ReadActualVelocity, verify the axis is in one of the following Axis States:

- Standstill
- Discrete Motion
- Error Stop

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

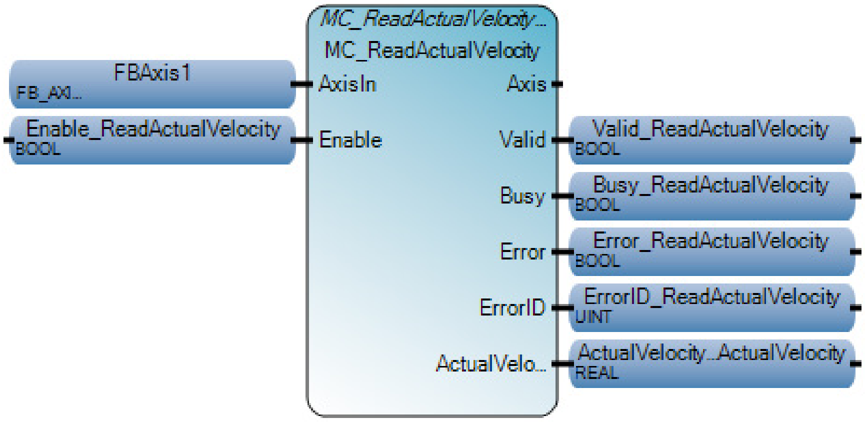
This instruction applies to the Micro830, Micro850 and Micro870 controllers that support motion control.



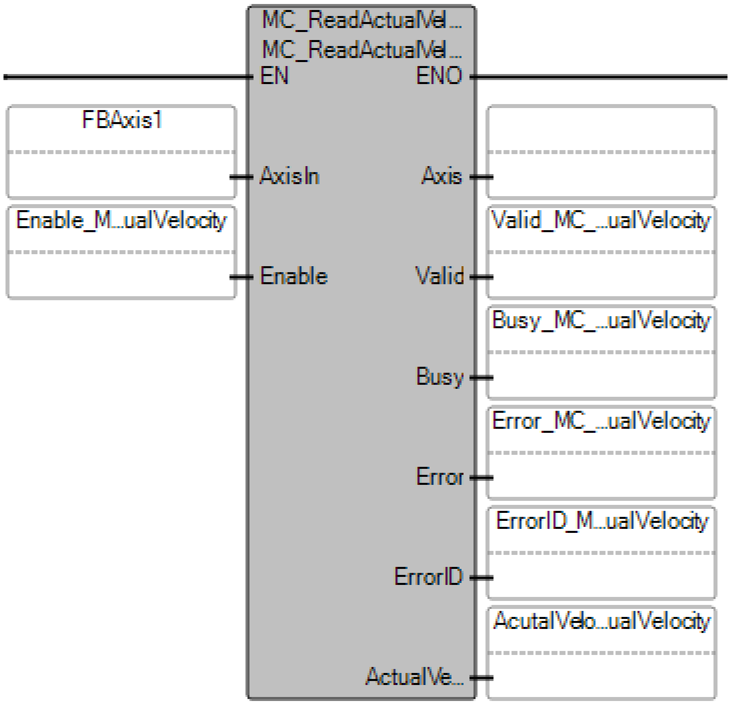
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter type	Data type	Description
AxisIn	Input	FB_AXIS_REF	For an FB_Axis (feedback axis), use the FB_AXIS_REF data type to define AxisIn.
Enable	Input	BOOL	TRUE - get the value for the actual velocity continuously. FALSE - the data is no longer valid, all outputs are reset to 0, Valid is set to False.
Axis	Output	FB_AXIS_REF	Axis output is read-only in Ladder Diagram programs. The Axis output parameters are defined in the FB_AXIS_REF data type on page 398.
Valid	Output	BOOL	TRUE - the function block is active and new output values are expected. FALSE - the function is not expecting new output values.
Busy	Output	BOOL	TRUE - the function block is not finished. FALSE - the function is idle.
Error	Output	BOOL	Indicates an error occurred. TRUE - An error is detected. FALSE - No error.
ErrorID	Output	UNIT	A unique numeric that identifies the error. The errors for this instruction are defined in Motion control function block error IDs on page 395.
ActualVelocity	Output	REAL	The value of the actual velocity for the feedback motion axis (in axis' unit [u/s]). ActualVelocity is a signed value, which includes direction information.

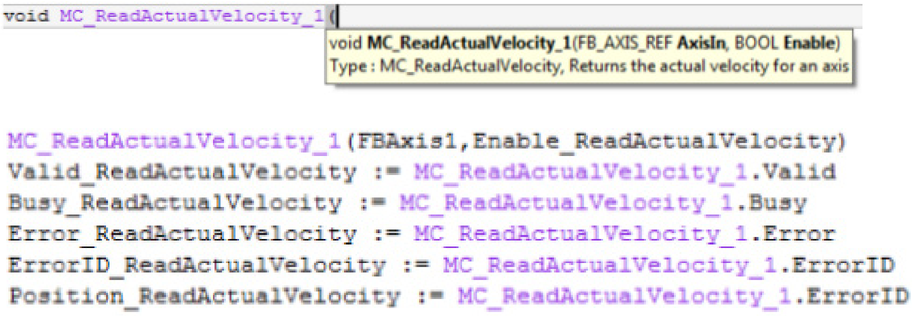
MC_ReadActualVelocity Function Block Diagram example



MC_ReadActualVelocity Ladder Diagram example



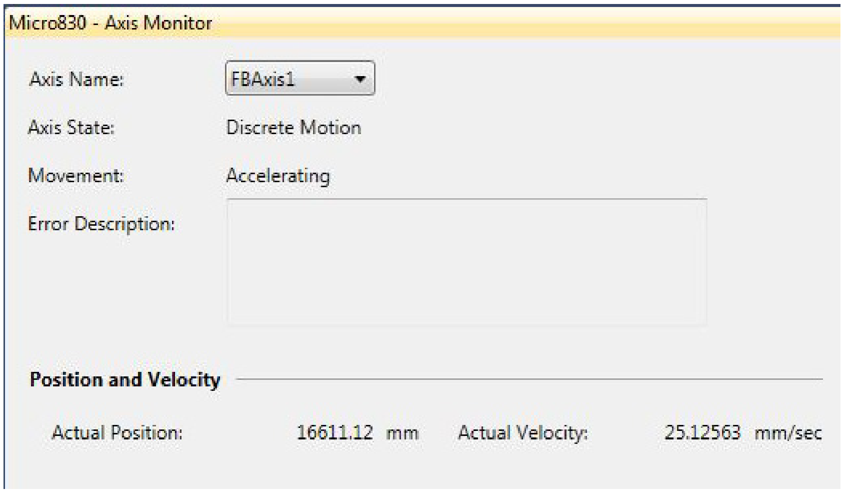
MC_ReadActualVelocity Structured Text example



```
void MC_ReadActualVelocity_1(
void MC_ReadActualVelocity_1(FB_AXIS_REF AxisIn, BOOL Enable)
Type : MC_ReadActualVelocity, Returns the actual velocity for an axis
```

```
MC_ReadActualVelocity_1(FBAxis1,Enable_ReadActualVelocity)
Valid_ReadActualVelocity := MC_ReadActualVelocity_1.Valid
Busy_ReadActualVelocity := MC_ReadActualVelocity_1.Busy
Error_ReadActualVelocity := MC_ReadActualVelocity_1.Error
ErrorID_ReadActualVelocity := MC_ReadActualVelocity_1.ErrorID
Position_ReadActualVelocity := MC_ReadActualVelocity_1.ErrorID
```

Results



Micro830 - Axis Monitor

Axis Name: FBAxis1

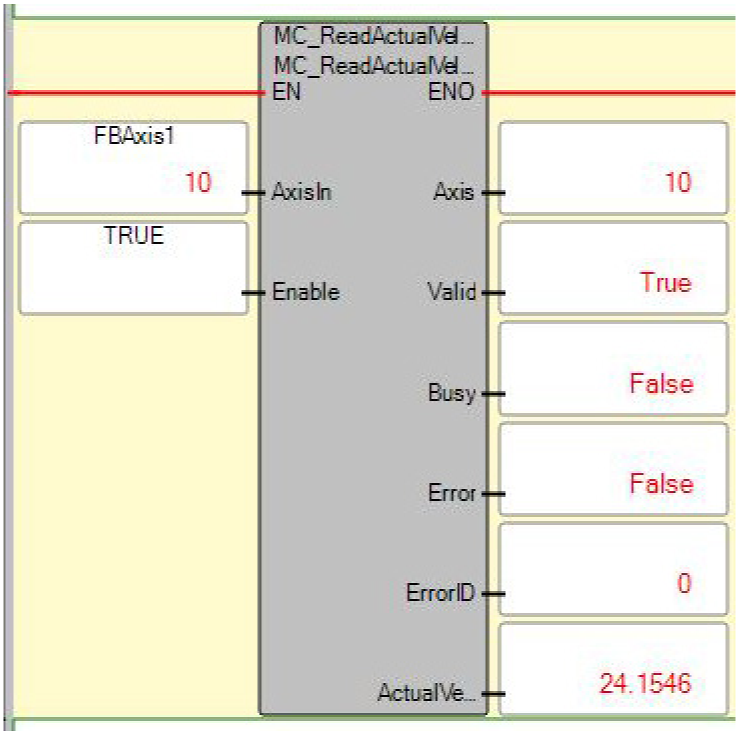
Axis State: Discrete Motion

Movement: Accelerating

Error Description:

Position and Velocity

Actual Position: 16611.12 mm Actual Velocity: 25.12563 mm/sec



FBAxis1 10 — AxisIn — Axis — 10

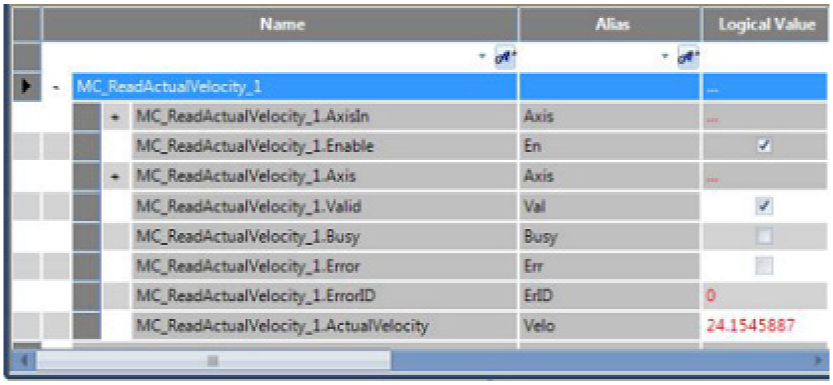
TRUE — Enable — Valid — True

Busy — False

Error — False

ErrorID — 0

ActualVe... — 24.1546



Name	Alias	Logical Value
MC_ReadActualVelocity_1		...
MC_ReadActualVelocity_1.AxisIn	Axis	...
MC_ReadActualVelocity_1.Enable	En	☑
MC_ReadActualVelocity_1.Axis	Axis	...
MC_ReadActualVelocity_1.Valid	Val	☑
MC_ReadActualVelocity_1.Busy	Busy	□
MC_ReadActualVelocity_1.Error	Err	□
MC_ReadActualVelocity_1.ErrorID	ErID	0
MC_ReadActualVelocity_1.ActualVelocity	Velo	24.1545887

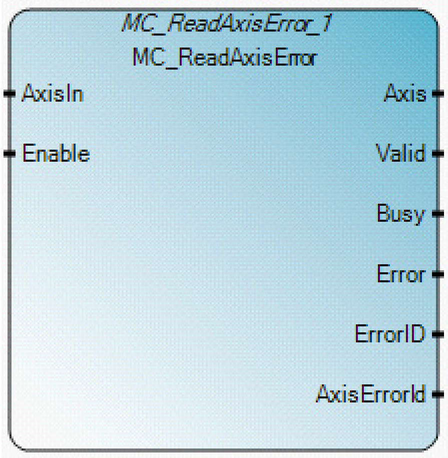
MC_ReadAxisError (motion control read axis error)

Reads the axis errors not related to the Motion control instruction blocks.

When an axis is in a Disabled state, MC_ReadAxisError may or may not get a non-zero Error ID for the axis because a Disabled axis can contain errors or be error-free.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

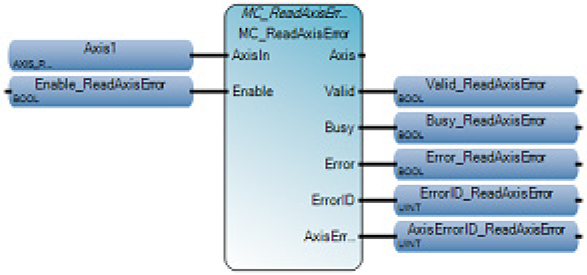
This instruction applies to the Micro830, Micro850 and Micro870 controllers that support motion control.



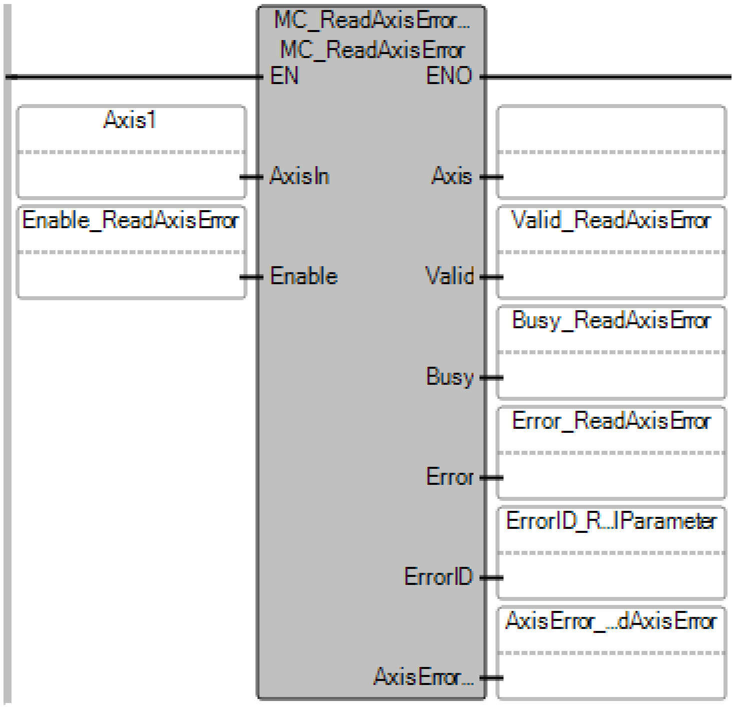
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter type	Data type	Description
EN	Input	BOOL	Instruction block enable. TRUE - execute current MC_ReadAxisError computation. FALSE - Error, ErrorID, and AxisErrorID are reset to False(or 0). Applies only to LD programs.
AxisIn	Input	AXIS_REF FB_AXIS_REF	Use the AXIS_REF data type on page 397 parameters to define AxisIn. For an FB_Axis (feedback axis), use the FB_AXIS_REF data type to define AxisIn.
Enable	Input	BOOL	TRUE - get the value of the parameter continuously while enabled. FALSE - resets Error, ErrorID, and AxisErrorID outputs to 0.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.
Axis	Output	AXIS_REF	Axis output is read-only in Ladder Diagram programs. The Axis output parameters are defined in the FB_AXIS_REF data type on page 398.
Valid	Output	BOOL	TRUE - the instruction block is active and new output values are expected. FALSE - the instruction block is inactive.
Busy	Output	BOOL	TRUE - the instruction block is not finished. FALSE - the instruction block is finished.
Error	Output	BOOL	Indicates an error occurred. TRUE - An error is detected. FALSE - No error.
ErrorID	Output	UINT	A unique numeric that identifies the error. The errors for this instruction are defined in Motion control function block error IDs.
AxisErrorID	Output	UINT	A unique numeric that identifies the axis error. The errors for this instruction are defined in AxisErrorID error codes on page 433.

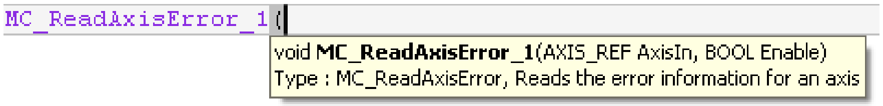
MC_ReadAxisError Function Block Diagram example



MC_ReadAxisError Ladder Diagram example

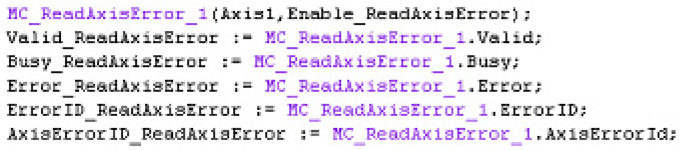


MC_ReadAxisError Structured Text example



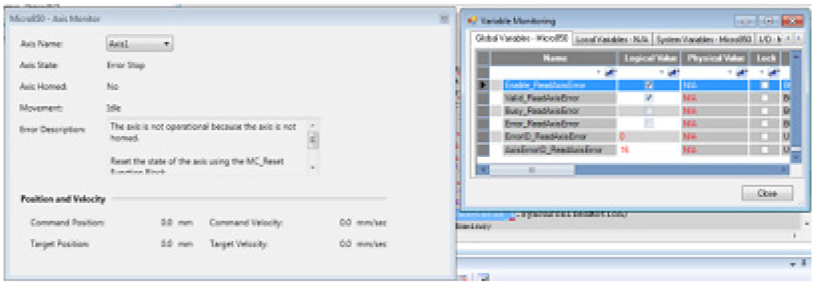
```
MC_ReadAxisError_1(
```

void **MC_ReadAxisError_1**(AXIS_REF AxisIn, BOOL Enable)
Type : MC_ReadAxisError, Reads the error information for an axis



```
MC_ReadAxisError_1(Axis1,Enable_ReadAxisError);
Valid_ReadAxisError := MC_ReadAxisError_1.Valid;
Busy_ReadAxisError := MC_ReadAxisError_1.Busy;
Error_ReadAxisError := MC_ReadAxisError_1.Error;
ErrorID_ReadAxisError := MC_ReadAxisError_1.ErrorID;
AxisErrorID_ReadAxisError := MC_ReadAxisError_1.AxisErrorId;
```

Results



Micro830 - Axis Monitor

Axis Name: Axis1

Axis State: Error Stop

Axis Homed: No

Movement: Idle

Error Description: The axis is not operational because the axis is not homed.

Reset the state of the axis using the MC_Reset Function Block.

Position and Velocity

Command Position: 0.0 mm Command Velocity: 0.0 mm/sec

Target Position: 0.0 mm Target Velocity: 0.0 mm/sec

Variable Monitoring

Global Variables - Micro830 | Local Variables : N/A | System Variables : Micro830 | I/O -

Name	Logical Value	Physical Value	Lock
Enable_ReadAxisError	☒	N/A	□
Valid_ReadAxisError	☑	N/A	□
Busy_ReadAxisError	□	N/A	□
Error_ReadAxisError	□	N/A	□
ErrorID_ReadAxisError	0	N/A	□
AxisErrorID_ReadAxisError	16	N/A	□

AxisErrorID error codes

The following table describes the Axis error codes identified in the AxisErrorID for MC_ReadAxisError on page 430.

Value	MACRO ID	Description
00	MC_FB_ERR_ NO	The axis is in an operational state (nothing to display).
01	MC_FB_ERR_ WRONG_STATE	The axis is not operational because an incorrect axis state was detected during a function block execution. Reset the state of the axis using the MC_Power and MC_Reset function blocks.
02	MC_FB_ERR_ RANGE	The axis is not operational because an invalid axis dynamic parameter (velocity, acceleration, deceleration, or jerk) is set in a function block. Reset the state of the axis using the MC_Power and MC_Reset function blocks. In the function block, correct any setting for the dynamic parameters that conflict with the settings on the Axis Dynamics configuration page.
03	MC_FB_ERR_ PARAM	The axis is not operational because an invalid parameter, (other than velocity, acceleration, deceleration, or jerk), is set in a function block. Reset the state of the axis using the MC_Power and MC_Reset function blocks. In the function block, correct the settings for the parameters, such as mode or position.
04	MC_FB_ERR_ AXISNUM	Motion internal Fault, Error ID = 0x04. Contact your local Rockwell Automation technical support representative. For contact information, see: http://www.rockwellautomation.com/support
05	MC_FB_ERR_ MECHAN	The axis is not operational because a drive or mechanical issue was detected. Check the connection between the drive and the controller (Drive Ready and In-Position signals), and ensure the drive is operating normally. Reset the state of the axis using the MC_Power and MC_Reset function blocks.
06	MC_FB_ERR_ NOPOWER	The axis is not powered on. Power on the axis using MC_Power function block. Reset the state of the axis using the MC_Reset function block.
07	MC_FB_ERR_ RESOURCE	The axis is not operational because it or its related resources required by a function block are under the control of other function block, or not available. Ensure the axis or its related resources required by the function block are available for use. Reset the state of the axis using the MC_Power and MC_Reset function blocks. Review and correct the application if there are multiple instances of the same function block trying to control the axis at the same time.
08	MC_FB_ERR_ PROFILE	The axis is not operational because the motion profile defined in a function block is invalid. Reset the state of the axis using the MC_Power and MC_Reset function blocks. Correct the profile in the function block.

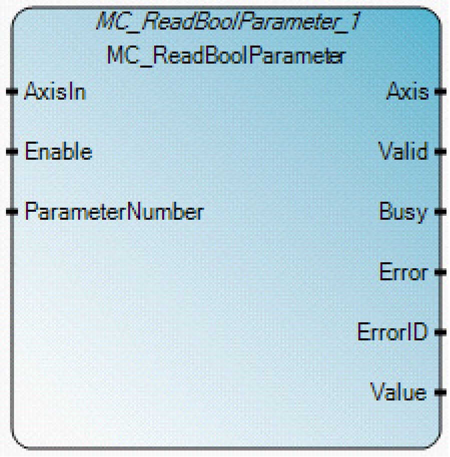
Value	MACRO ID	Description
09	MC_FB_ERR_ VELOCITY	The axis is not operational because the motion profile requested in a function block conflicts with the current axis velocity. Possible causes: • The function block requests the axis to reverse the direction while the axis is moving. • The current velocity is too low or too high for the requested motion profile. Reset the state of the axis using the MC_Power and MC_Reset function blocks. Correct the motion profile in the function block, or re-execute the function block when the axis velocity is compatible with the requested motion profile.
10	MC_FB_ERR_ SOFT_LIMIT	The axis is not operational because a Soft Limit error was detected, or executing the function block would cause a Soft Limit error. Reset the state of the axis using the MC_Power and MC_Reset function blocks. Check the velocity or target position settings for the function block, or adjust Soft Limit setting.
11	MC_FB_ERR_ HARD_LIMIT	The axis is not operational because a Hard Limit error was detected. Reset the state of the axis using the MC_Reset function block, and then move the axis away from the Hard Limit switch in the opposite direction.
12	MC_FB_ERR_ LOG_LIMIT	The axis is not operational because a PTO Accumulator logic limit error was detected, or executing the function block would cause a PTO Accumulator logic limit error. Reset the state of the axis using the MC_Power and MC_Reset function blocks. Check the velocity or target position settings for the function block. Use the MC_SetPosition function block to adjust the axis coordinate system.
13	MC_FB_ERR_ ERR_ENGINE	The axis is not operational because a motion engine execution error was detected. Power cycle the entire machine and download the User Application to the controller again. If the fault persists, contact your local Rockwell Automation technical support representative. For contact information, see: http://www.rockwellautomation.com/support.
16	MC_FB_ERR_ NOT_HOMED	The axis is not operational because the axis is not homed. Reset the state of the axis using the MC_Power and MC_Reset function blocks. Execute homing against the axis using MC_Home function block.
128	MC_FB_ERR_ PARAM_MODIFIED	Motion internal warning, Warning ID = 0x80. Contact your local Rockwell Automation technical support representative. For contact information, see: http://www.rockwellautomation.com/support.

MC_ReadBoolParameter (motion control read BOOL parameter)

Returns the value of a vendor specific parameter of type BOOL.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

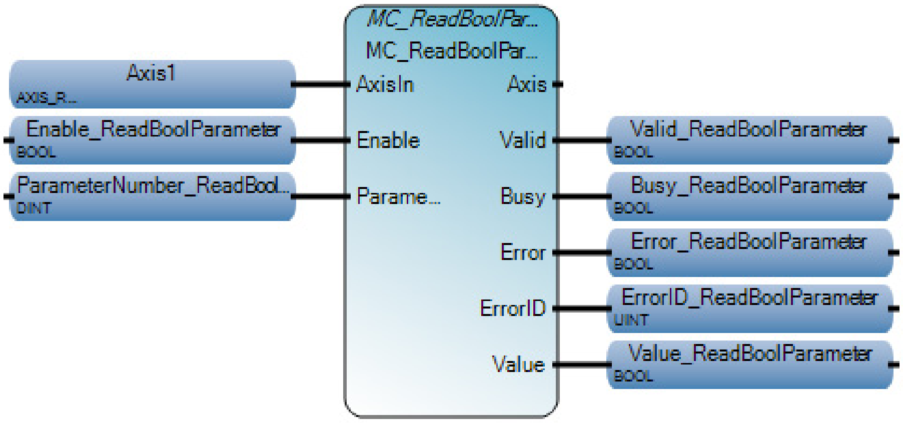
This instruction applies to the Micro830, Micro850 and Micro870 controllers that support motion control.



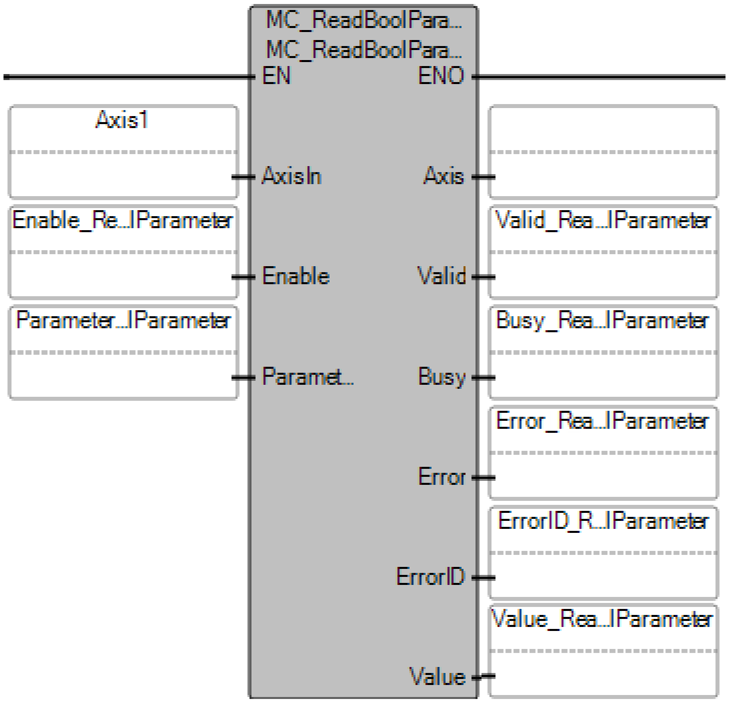
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter type	Data type	Description
EN	Input	BOOL	Instruction block enable. TRUE - execute current MC_ReadBoolParameter computation. FALSE - reset Value output to 0. Applies only to Ladder Diagram programs.
AxisIn	Input	AXIS_REF FB_AXIS_REF	Use the AXIS_REF data type on page 397 to define AxisIn. For FB_Axis (feedback axis), use the FB_AXIS_REF data type on page 398 to define AxisIn.
Enable	Input	BOOL	TRUE - get the value of the parameter continuously while enabled. FALSE - the Value output is reset to 0.
ParameterNumber	Input	DINT	Parameter identification. Parameter numbers definitions are defined in Motion control function block parameter numbers.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.
Axis	Output	AXIS_REF	Axis output is read-only in Ladder Diagram programs.
Valid	Output	BOOL	TRUE - the value of the Parameter is available. FALSE - the Parameter value is unavailable.
Busy	Output	BOOL	TRUE - the function block is working and new output values are expected. FALSE - the function is idle.
Error	Output	BOOL	Indicates an error occurred. TRUE - An error is detected. FALSE - No error.
ErrorID	Output	UINT	A unique numeric that identifies the error. The errors for this instruction are defined in Motion control function block error IDs on page 395.
Value	Output	BOOL	Value of the specified parameter in the data type, as specified by the vendor.

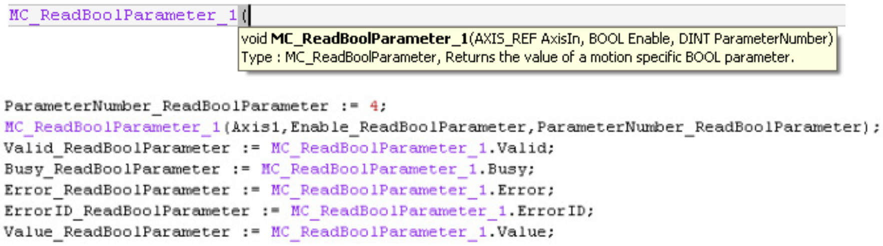
MC_ReadBoolParameter Function Block Diagram example



MC_ReadBoolParameter Ladder Diagram example



MC_ReadBoolParameter Structured Text example

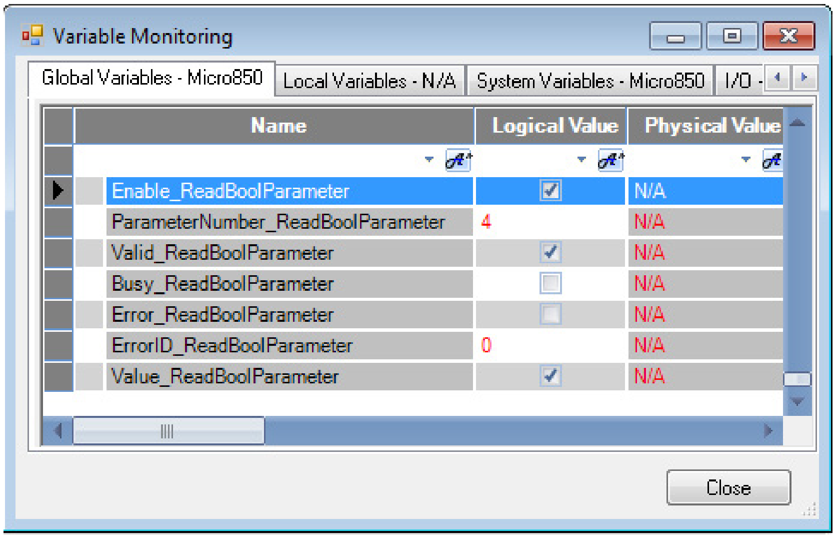


```
MC_ReadBoolParameter_1(
```

void **MC_ReadBoolParameter_1**(AXIS_REF AxisIn, BOOL Enable, DINT ParameterNumber)
Type : MC_ReadBoolParameter, Returns the value of a motion specific BOOL parameter.

```
ParameterNumber_ReadBoolParameter := 4;
MC_ReadBoolParameter_1(Axis1,Enable_ReadBoolParameter,ParameterNumber_ReadBoolParameter);
Valid_ReadBoolParameter := MC_ReadBoolParameter_1.Valid;
Busy_ReadBoolParameter := MC_ReadBoolParameter_1.Busy;
Error_ReadBoolParameter := MC_ReadBoolParameter_1.Error;
ErrorID_ReadBoolParameter := MC_ReadBoolParameter_1.ErrorID;
Value_ReadBoolParameter := MC_ReadBoolParameter_1.Value;
```

Results



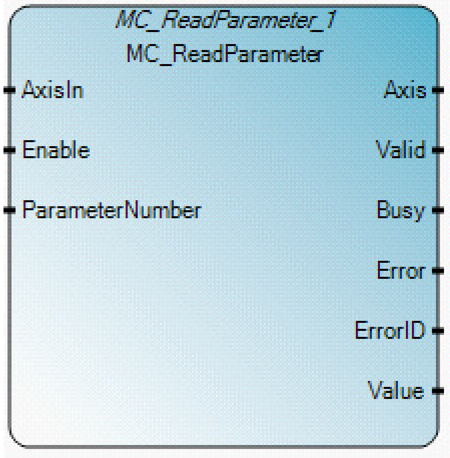
Name	Logical Value	Physical Value
Enable_ReadBoolParameter	☑	N/A
ParameterNumber_ReadBoolParameter	4	N/A
Valid_ReadBoolParameter	☑	N/A
Busy_ReadBoolParameter	□	N/A
Error_ReadBoolParameter	□	N/A
ErrorID_ReadBoolParameter	0	N/A
Value_ReadBoolParameter	☑	N/A

MC_ReadParameter (motion control read parameter)

Returns the value of a vendor specific parameter in a Real data type.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro830, Micro850 and Micro870 controllers that support motion control.

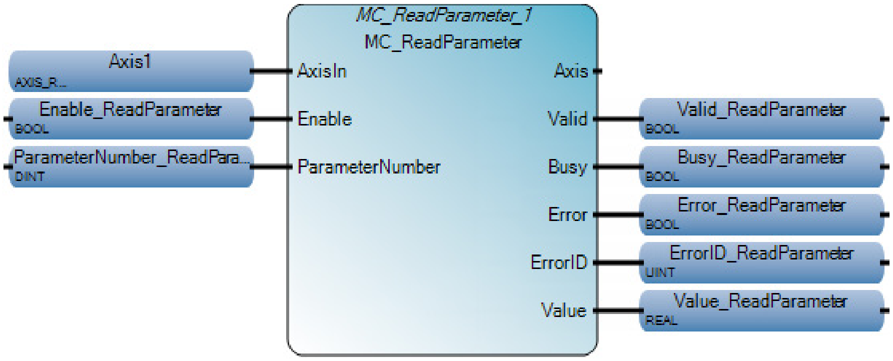


Use this table to help determine the parameter values for this instruction.

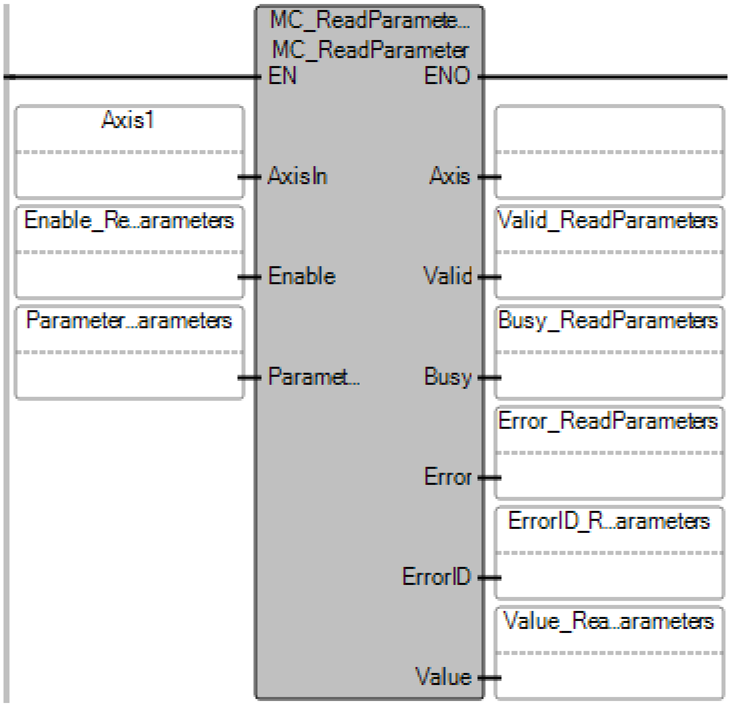
Parameter	Parameter type	Data type	Description
EN	Input	BOOL	Instruction block enable. TRUE - execute current MC_ReadParameter computation. FALSE - the Value output is reset to 0. Applies only to Ladder Diagram programs.
AxisIn	Input	AXIS_REF FB_AXIS_REF	Use the AXIS_REF data type on page 397 to define AxisIn. For FB_Axis (feedback axis), use the FB_AXIS_REF data type on page 398 to define AxisIn.
Enable	Input	BOOL	TRUE - get the value of the parameter number continuously. FALSE - the Value output is reset to 0.

Parameter	Parameter type	Data type	Description
ParameterNumber	Input	DINT	Parameter identification. Parameter numbers definitions are defined in Motion control function block parameter numbers.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.
Axis	Output	AXIS_REF	Axis output is read-only in Ladder Diagram programs.
Valid	Output	BOOL	TRUE - valid outputs are available. FALSE - valid outputs are unavailable.
Busy	Output	BOOL	TRUE - the function block is working and new output values are expected. FALSE - the function block is idle.
Error	Output	BOOL	Indicates an error occurred. TRUE - An error is detected. FALSE - No error.
ErrorID	Output	UINT	A unique numeric that identifies the error. The errors for this instruction are defined in Motion control function block error IDs on page 395.
Value	Output	REAL	Value of the specified parameter in the data type, as specified by the vendor.

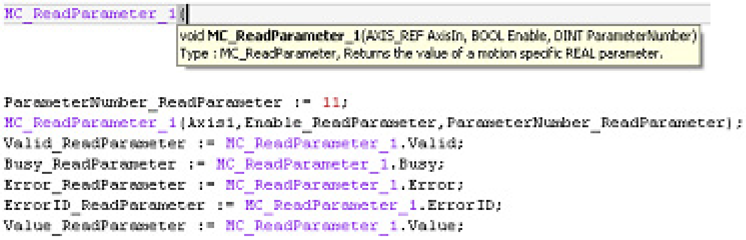
MC_ReadParameter Function Block Diagram example



MC_ReadParameter Ladder Diagram example



MC_ReadParameter Structured Text example

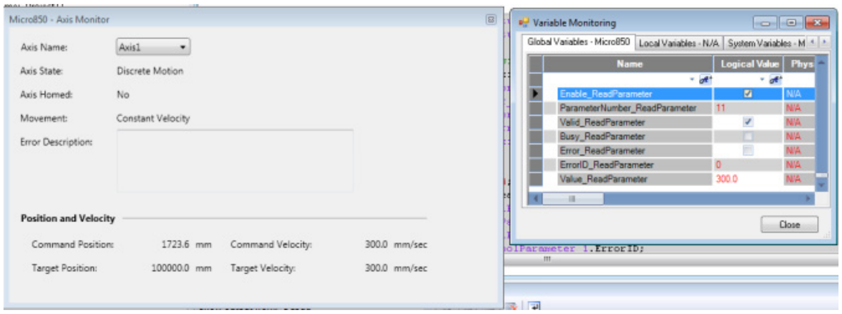


```
MC_ReadParameter_1(
```

void MC_ReadParameter_1(AXIS_REF AxisIn, BOOL Enable, DINT ParameterNumber)
Type : MC_ReadParameter, Returns the value of a motion specific REAL parameter.

```
ParameterNumber_ReadParameter := 11;
MC_ReadParameter_1(Axis1,Enable_ReadParameter,ParameterNumber_ReadParameter);
Valid_ReadParameter := MC_ReadParameter_1.Valid;
Busy_ReadParameter := MC_ReadParameter_1.Busy;
Error_ReadParameter := MC_ReadParameter_1.Error;
ErrorID_ReadParameter := MC_ReadParameter_1.ErrorID;
Value_ReadParameter := MC_ReadParameter_1.Value;
```

Results



Micro850 - Axis Monitor

Axis Name: Axis1
Axis State: Discrete Motion
Axis Homed: No
Movement: Constant Velocity
Error Description:

Position and Velocity

Command Position: 1723.6 mm Command Velocity: 300.0 mm/sec
Target Position: 100000.0 mm Target Velocity: 300.0 mm/sec

Variable Monitoring

Global Variables - Micro850 | Local Variables - N/A | System Variables - M

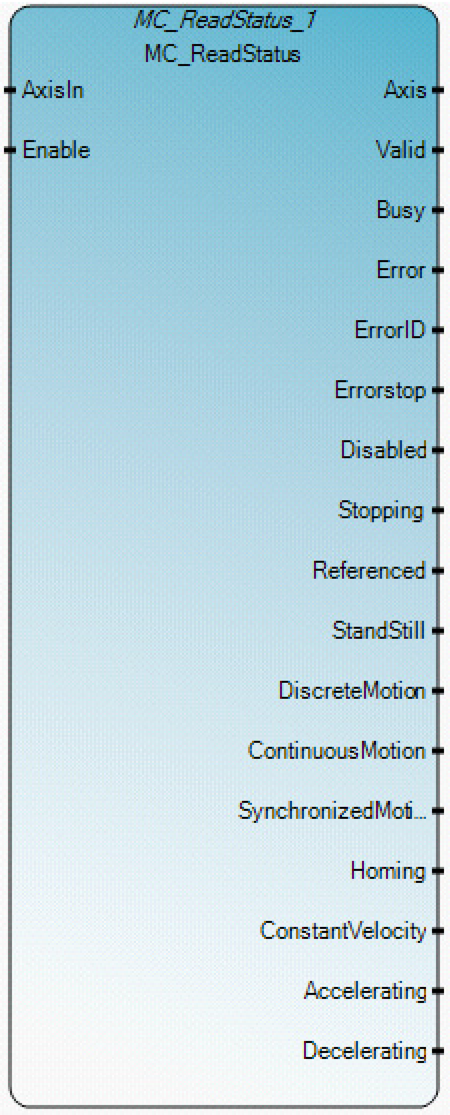
Name	Logical Value	Phys
Enable_ReadParameter	☑	N/A
ParameterNumber_ReadParameter	11	N/A
Valid_ReadParameter	✓	N/A
Busy_ReadParameter	□	N/A
Error_ReadParameter	□	N/A
ErrorID_ReadParameter	0	N/A
Value_ReadParameter	300.0	N/A

MC_ReadStatus (motion control read status)

Returns the status of the axis with respect to the motion currently in progress.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro830, Micro850 and Micro870 controllers that support motion control.

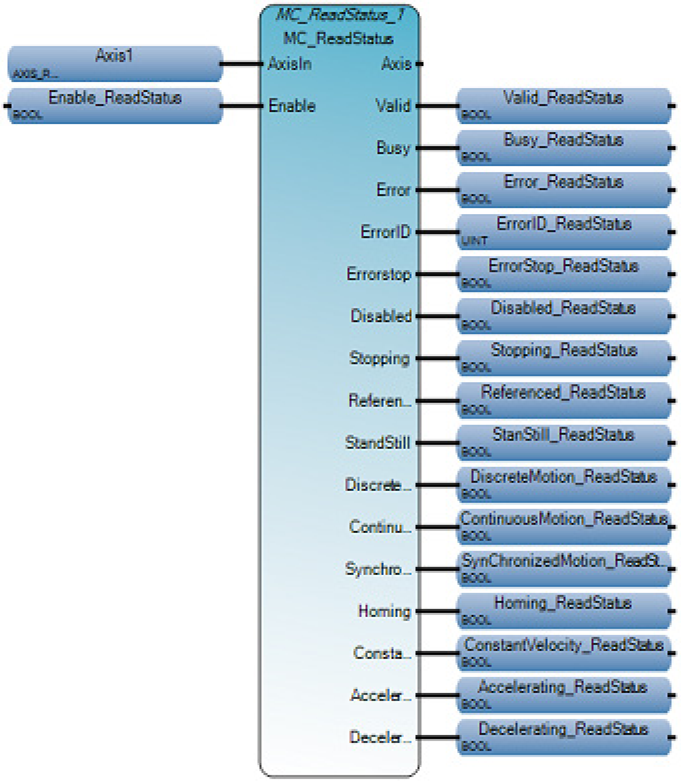


Use this table to help determine the parameter values for this instruction.

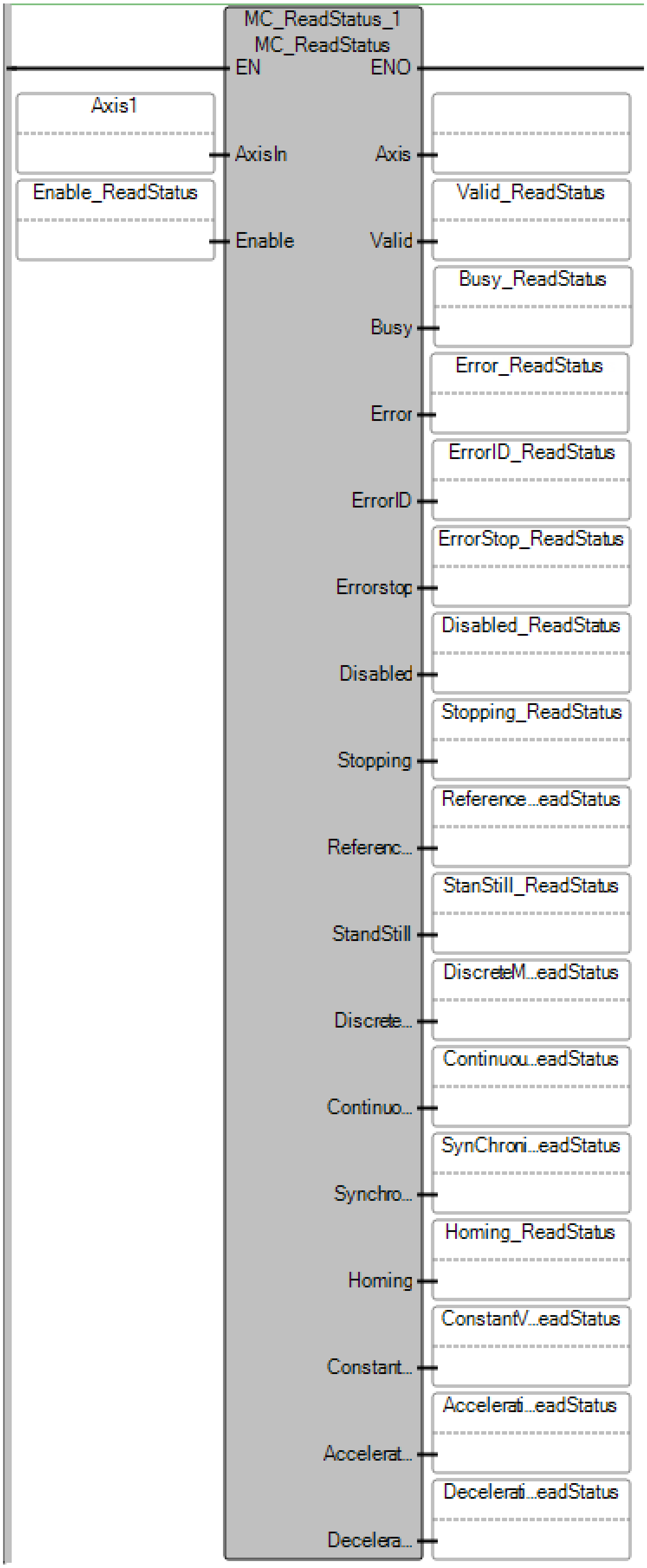
Parameter	Parameter type	Data type	Description
EN	Input	BOOL	Instruction block enable. TRUE - execute current MC_ReadStatus computation. FALSE - there is no computation. Applies only to Ladder Diagram programs.
AxisIn	Input	AXIS_REF FB_AXIS_REF	Use the AXIS_REF data type on page 397 parameters to define AxisIn. For an FB_Axis (feedback axis), use the FB_AXIS_REF data type on page 398 to define AxisIn.
Enable	Input	BOOL	TRUE - get the axis status continuously. FALSE - all status outputs are reset to 0.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.
Axis	Output	AXIS_REF	Axis output is read-only in Ladder Diagram programs.
Valid	Output	BOOL	TRUE - valid outputs are available. FALSE - outputs unavailable.

Parameter	Parameter type	Data type	Description
Busy	Output	BOOL	TRUE - the function block is working and new output values are expected. FALSE - the function block is idle.
Error	Output	BOOL	Indicates an error occurred. TRUE - An error is detected. FALSE - No error.
ErrorID	Output	UINT	A unique numeric that identifies the error. The errors for this instruction are defined in Motion control function block error IDs.
ErrorStop	Output	BOOL	TRUE - the axis state is ErrorStop. Axis states are defined in Motion control axis state values and names.
Disabled	Output	BOOL	TRUE - the axis state is Disabled.
Stopping	Output	BOOL	TRUE - the axis state is Stopping.
Referenced	Output	BOOL	TRUE - the axis state is homed, the absolute reference position is known for the axis.
StandStill	Output	BOOL	TRUE - the axis state is StandStill.
DiscreteMotion	Output	BOOL	TRUE - the axis state is DiscreteMotion.
ContinuousMotion	Output	BOOL	TRUE - the axis state is ContinuousMotion.
SynchronizedMotion	Output	BOOL	Synchronized motion is not supported in Micro800 controllers. TRUE - never true. FALSE - always false.
Homing	Output	BOOL	TRUE - the axis state is Homing.
ConstantVelocity	Output	BOOL	TRUE - the velocity for the motor is constant.
Accelerating	Output	BOOL	TRUE - axis is accelerating, increased energy to the motor.
Decelerating	Output	BOOL	TRUE - axis is decelerating, decreased energy to the motor.

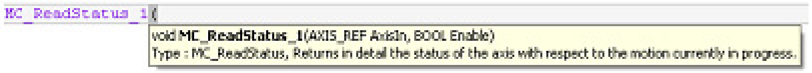
MC_ReadStatus Function Block Diagram example



MC_ReadStatus Ladder Diagram example


MC_ReadStatus_1
MC_ReadStatus
EN
ENO
Axis1
AxisIn
Axis
Enable_ReadStatus
Enable
Valid
Valid_ReadStatus
Busy
Busy_ReadStatus
Error
Error_ReadStatus
ErrorID
ErrorID_ReadStatus
Errorstop
ErrorStop_ReadStatus
Disabled
Disabled_ReadStatus
Stopping
Stopping_ReadStatus
Referenc...
Reference...eadStatus
StandStill
StanStill_ReadStatus
Discrete...
DiscreteM...eadStatus
Continuo...
Continuou...eadStatus
Synchro...
SynChroni...eadStatus
Homing
Homing_ReadStatus
Constant...
ConstantV...eadStatus
Accelerat...
Accelerati...eadStatus
Decelera...
Decelerati...eadStatus

MC_ReadStatus Structured Text example

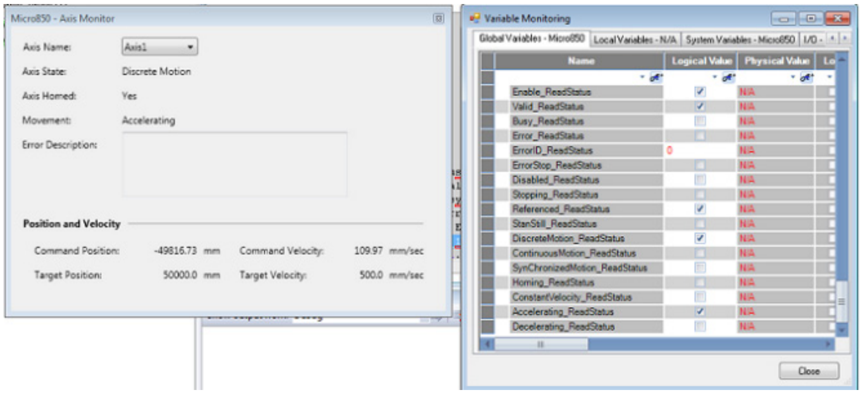


```
MC_ReadStatus_1(
```

void MC_ReadStatus_1(AXIS_REF AxisIn, BOOL Enable)
Type : MC_ReadStatus, Returns in detail the status of the axis with respect to the motion currently in progress.

```
MC_ReadStatus_1(Axis1,Enable_ReadStatus);
Valid_ReadStatus := MC_ReadStatus_1.Valid;
Busy_ReadStatus := MC_ReadStatus_1.Busy;
Error_ReadStatus := MC_ReadStatus_1.Error;
ErrorID_ReadStatus := MC_ReadStatus_1.ErrorID;
ErrorStop_ReadStatus := MC_ReadStatus_1.Errorstop;
Disabled_ReadStatus := MC_ReadStatus_1.Disabled;
Stopping_ReadStatus := MC_ReadStatus_1.Stopping;
Referenced_ReadStatus := MC_ReadStatus_1.Referenced;
StanStill_ReadStatus := MC_ReadStatus_1.StandStill;
DiscreteMotion_ReadStatus := MC_ReadStatus_1.DiscreteMotion;
ContinuousMotion_ReadStatus := MC_ReadStatus_1.ContinuousMotion;
SynChronizedMotion_ReadStatus := MC_ReadStatus_1.SynchronizedMotion;
Homing_ReadStatus := MC_ReadStatus_1.Homing;
ConstantVelocity_ReadStatus := MC_ReadStatus_1.ConstantVelocity;
Accelerating_ReadStatus := MC_ReadStatus_1.Accelerating;
Decelerating_ReadStatus := MC_ReadStatus_1.Decelerating;
```

Results



Micro850 - Axis Monitor

Axis Name: Axis1
Axis State: Discrete Motion
Axis Homed: Yes
Movement: Accelerating
Error Description:

Position and Velocity

Command Position: -49816.73 mm Command Velocity: 109.97 mm/sec
Target Position: 50000.0 mm Target Velocity: 500.0 mm/sec

Variable Monitoring

Global Variables - Micro850 | Local Variables - N/A | System Variables - Micro850 | I/O -

Name	Logical Value	Physical Value
Enable_ReadStatus	☑	N/A
Valid_ReadStatus	☑	N/A
Busy_ReadStatus	☐	N/A
Error_ReadStatus	☐	N/A
ErrorID_ReadStatus	0	N/A
ErrorStop_ReadStatus	☐	N/A
Disabled_ReadStatus	☐	N/A
Stopping_ReadStatus	☐	N/A
Referenced_ReadStatus	☑	N/A
StanStill_ReadStatus	☐	N/A
DiscreteMotion_ReadStatus	☑	N/A
ContinuousMotion_ReadStatus	☐	N/A
SynChronizedMotion_ReadStatus	☐	N/A
Homing_ReadStatus	☐	N/A
ConstantVelocity_ReadStatus	☐	N/A
Accelerating_ReadStatus	☑	N/A
Decelerating_ReadStatus	☐	N/A

MC_Reset (motion control reset)

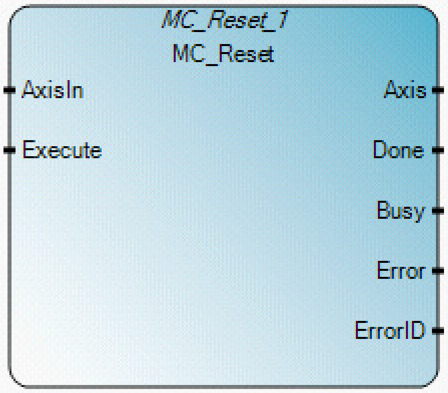
Transitions the axis state from ErrorStop to StandStill by resetting all internal axis-related errors. The outputs of the function block instances are not changed.

Operation details:

- If the axis alarm state is unchanged after executing MC_Reset, execute MC_Power followed by MC_Reset.
- MC_Reset only resets the axis state from ErrorStop to StandStill. Executing MC_Reset when the axis is in other states, including Disabled, results in an error, and has no impact on on-going motion or the status of the axis.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

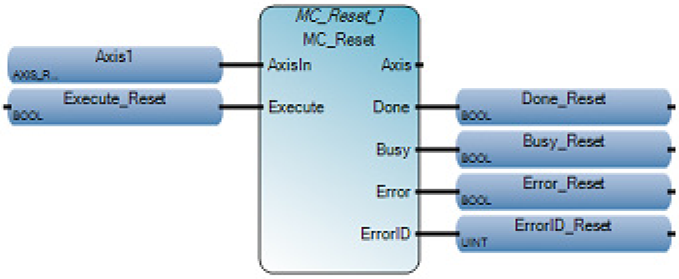
This instruction applies to the Micro830, Micro850 and Micro870 controllers that support motion control.



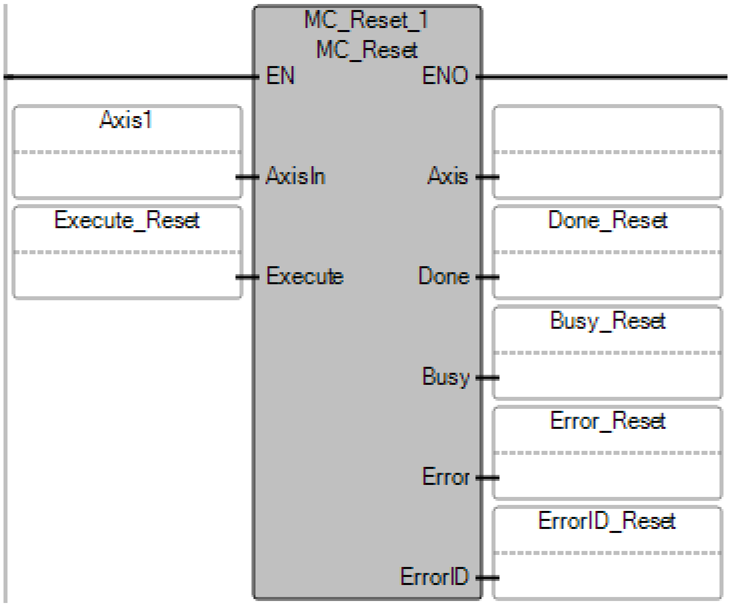
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter type	Data type	Description
EN	Input	BOOL	Instruction block enable. TRUE, execute current MC_Reset computation. FALSE, there is no computation. Applies only to Ladder Diagram programs.
AxisIn	Input	AXIS_REF FB_AXIS_REF	Use the AXIS_REF data type to define AxisIn. For FB_Axis (feedback axis), use the FB_AXIS_REF data type on page 398 to define AxisIn.
Execute	Input	BOOL	TRUE - resets the axis to the rising edge.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.
Axis	Output	AXIS_REF	Axis output is read-only in Ladder Diagram programs.
Done	Output	BOOL	TRUE - axis state is StandStill or Disabled.
Busy	Output	BOOL	TRUE - the function block is not finished.
Error	Output	BOOL	Indicates an error occurred. TRUE - An error is detected. FALSE - No error.
ErrorID	Output	UINT	A unique numeric that identifies the error. The errors for this instruction are defined in Motion control function block error IDs on page 395.

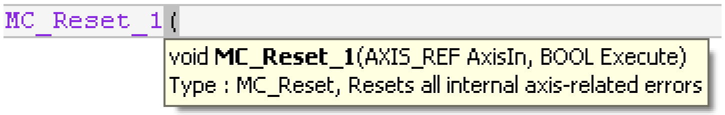
MC_Reset Function Block Diagram example



MC_Reset Ladder Diagram example



MC_Reset Structured Text example

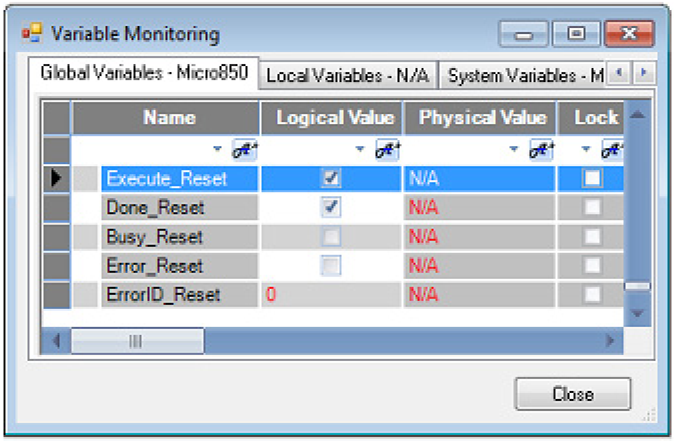


```
MC_Reset_1(
```

void **MC_Reset_1**(AXIS_REF AxisIn, BOOL Execute)
Type : MC_Reset, Resets all internal axis-related errors

```
MC_Reset_1(Axis1,Execute_Reset);
Done_Reset := MC_Reset_1.Done;
Busy_Reset := MC_Reset_1.Busy;
Error_Reset := MC_Reset_1.Error;
ErrorID_Reset := MC_Reset_1.ErrorID;
```

Results



Name	Logical Value	Physical Value	Lock
Execute_Reset	☑	N/A	□
Done_Reset	☑	N/A	□
Busy_Reset	□	N/A	□
Error_Reset	□	N/A	□
ErrorID_Reset	0	N/A	□

MC_SetPosition (motion control set position)

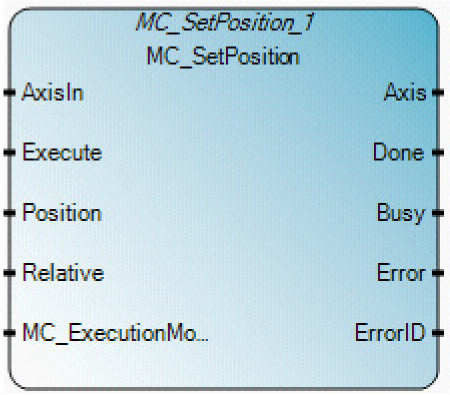
Shifts the coordinate system of an axis by manipulating the actual position of an axis with the same value without causing any movement.

Operation details:

- MC_SetPostion can successfully complete only when the axis state is StandStill, continuous Motion (MC_ExecutionMode = 0), or when the on-going motion completes, and ends with a StandStill state (MC_ExecutionMode = 1).
- MC_SetPosition operates the same as MC_Home when the HomingMode = MC_HOME_DIRECT (0x04), except the MC_Home function block sets the Axis Homed status.
- When MC_ExecutionMode = 0 (mcImmediately), the execution of the MC_SetPosition function block reports an error if there is on-going non-continuous motion with the axis.
- When MC_ExecutionMode = 1 (mcQueued), the actual position setting occurs only when all previous on-going motion stops. That is, each previous function block must have at least one of the Done, Aborted, or Error outputs equal to True.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro830, Micro850 and Micro870 controllers that support motion control.

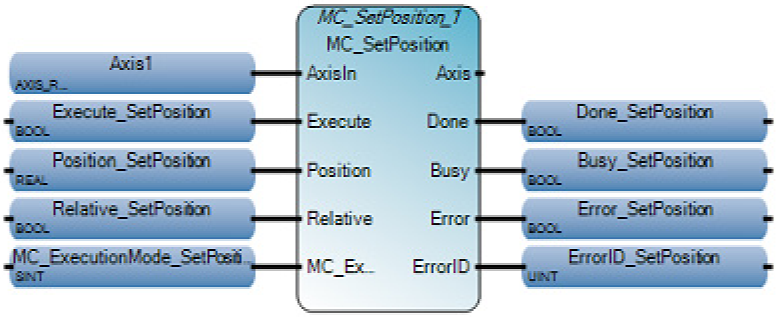


Use this table to help determine the parameter values for this instruction.

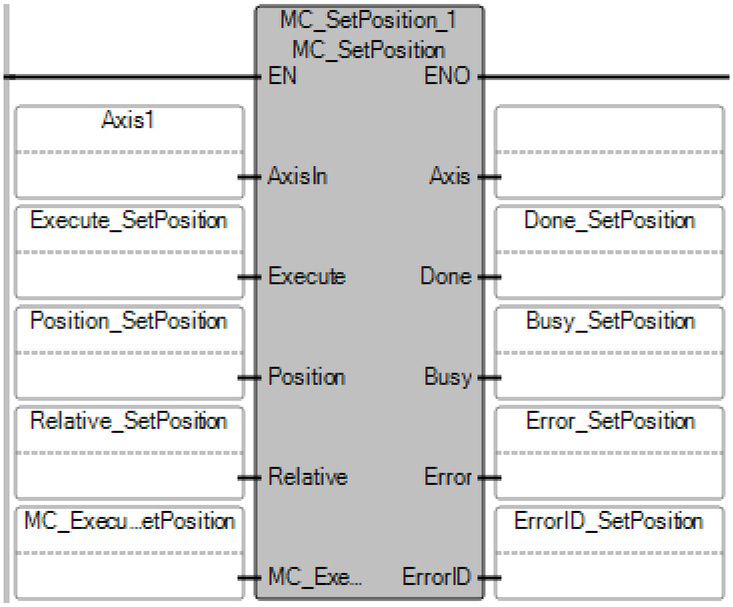
Parameter	Parameter type	Data type	Description
EN	Input	BOOL	Instruction block enable. TRUE - execute current MC_SetPosition computation. FALSE - there is no computation. Applies only to Ladder Diagram programs.
AxisIn	Input	AXIS_REF FB_AXIS_REF	Use the AXIS_REF data type to define AxisIn. For FB_Axis (feedback axis), use the FB_AXIS_REF data type on page 398 to define AxisIn.
Execute	Input	BOOL	TRUE - starts setting the axis position.
Position	Input	REAL	The absolute position or relative distance to be set for the axis.

Parameter	Parameter type	Data type	Description
Relative	Input	BOOL	TRUE - set the relative distance for the axis. FALSE - set the absolute position for the axis.
MC_ExecutionMode	Input	SINT	Values are: • 0 (*mcImmediately*) - The functionality is immediately valid. • 1 (*mcQueued*) - The new functionality becomes valid when: • all previous motion commands set to one of the following output parameters: Done, Aborted or Error. • the axis is not in a moving state. For (MC_ExecutionMode = 0), this function block successfully completes when the axis state is Disabled or StandStill. The execution of this function block reports an error if there is an on-going non-Continuous motion with the axis in this mode. For (MC_ExecutionMode = 1), this function block successfully completed when the axis state is Disabled, Standstill, or the on-going motion can complete, ending with a Standstill state. Other input values are reserved currently, and are considered as invalid parameters.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.
Axis	Output	AXIS_REF	Axis output is read-only in Ladder Diagram programs.
Done	Output	BOOL	TRUE - the Position has new value.
Busy	Output	BOOL	TRUE - the function block is not finished.
Error	Output	BOOL	Indicates an error occurred. TRUE - An error is detected. FALSE - No error.
ErrorID	Output	UINT	A unique numeric that identifies the error. The errors for this instruction are defined in Motion control function block error IDs on page 395.

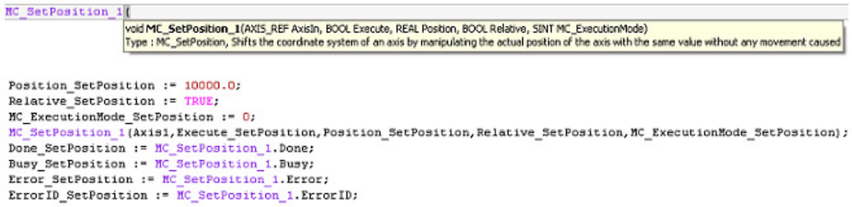
MC_SetPosition Function Block Diagram example



MC_SetPosition Ladder Diagram example

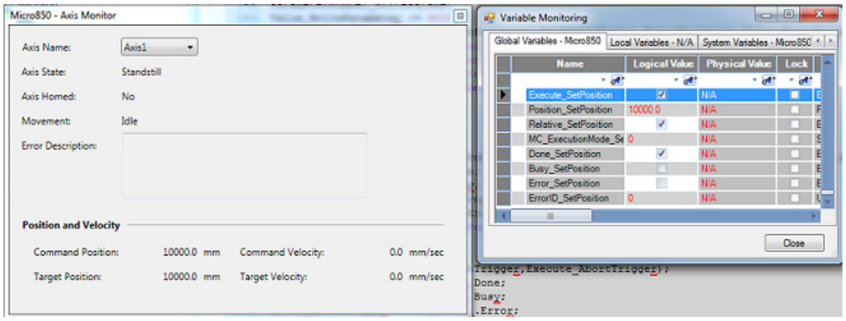


MC_SetPosition Structured Text example



```
MC_SetPosition_1(
            void MC_SetPosition_1(AXIS_REF AxisIn, BOOL Execute, REAL Position, BOOL Relative, SINT MC_ExecutionMode)
            Type : MC_SetPosition, Shifts the coordinate system of an axis by manipulating the actual position of the axis with the same value without any movement caused

Position_SetPosition := 10000.0;
Relative_SetPosition := TRUE;
MC_ExecutionMode_SetPosition := 0;
MC_SetPosition_1(Axis1,Execute_SetPosition,Position_SetPosition,Relative_SetPosition,MC_ExecutionMode_SetPosition);
Done_SetPosition := MC_SetPosition_1.Done;
Busy_SetPosition := MC_SetPosition_1.Busy;
Error_SetPosition := MC_SetPosition_1.Error;
ErrorID_SetPosition := MC_SetPosition_1.ErrorID;
```

Results



Micro850 - Axis Monitor

Axis Name: Axis1

Axis State: Standstill

Axis Homed: No

Movement: Idle

Error Description:

Position and Velocity

Command Position: 10000.0 mm Command Velocity: 0.0 mm/sec

Target Position: 10000.0 mm Target Velocity: 0.0 mm/sec

Variable Monitoring

Global Variables - Micro850 | Local Variables - N/A | System Variables - Micro850

Name	Logical Value	Physical Value	Lock
Execute_SetPosition	☑	N/A	□
Position_SetPosition	10000.0	N/A	□
Relative_SetPosition	☑	N/A	□
MC_ExecutionMode_Se	0	N/A	□
Done_SetPosition	☑	N/A	□
Busy_SetPosition	□	N/A	□
Error_SetPosition	□	N/A	□
ErrorID_SetPosition	0	N/A	□

```
Trigger,Execute_AbortTrigger);
Done;
Busy;
.Error;
```

MC_Stop (motion control stop)

Commands a controlled motion stop and transfers the axis state to Stopping. Any ongoing function block execution is cancelled. All function block move commands are ignored until the axis state transitions to StandStill.

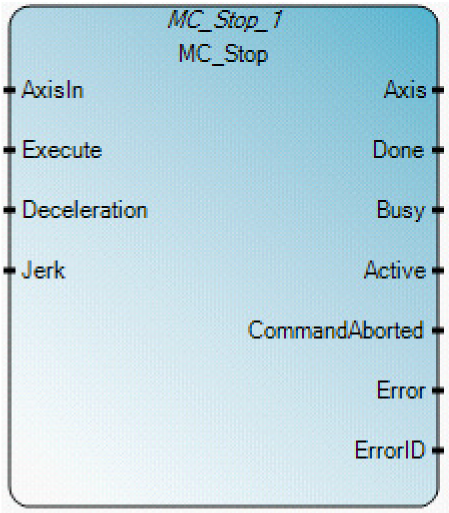
Operation details:

- As long as the Execute input is high, the axis remains in the Stopping state. While the axis is in the Stopping state, other motion function blocks cannot perform any motion on the same axis.

- If Deceleration equals zero, the MC_Stop parameters are determined by the Axis configuration Emergency Stop setting, including E-Stop type, E-stop Deceleration and E-stop Jerk.
- When there are no errors detected during the stop sequence, the axis state transitions to StandStill after the Done bit is SET and the Execute input changes to False.
- Use MC_Stop for emergency stop functionality or exception situations. Use MC_Halt for normal motion stops.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro830, Micro850 and Micro870 controllers that support motion control.

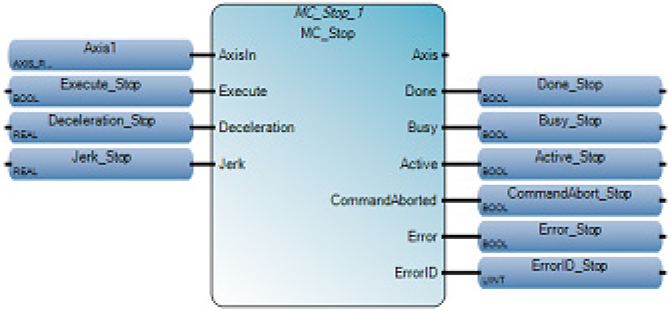


Use this table to help determine the parameter values for this instruction.

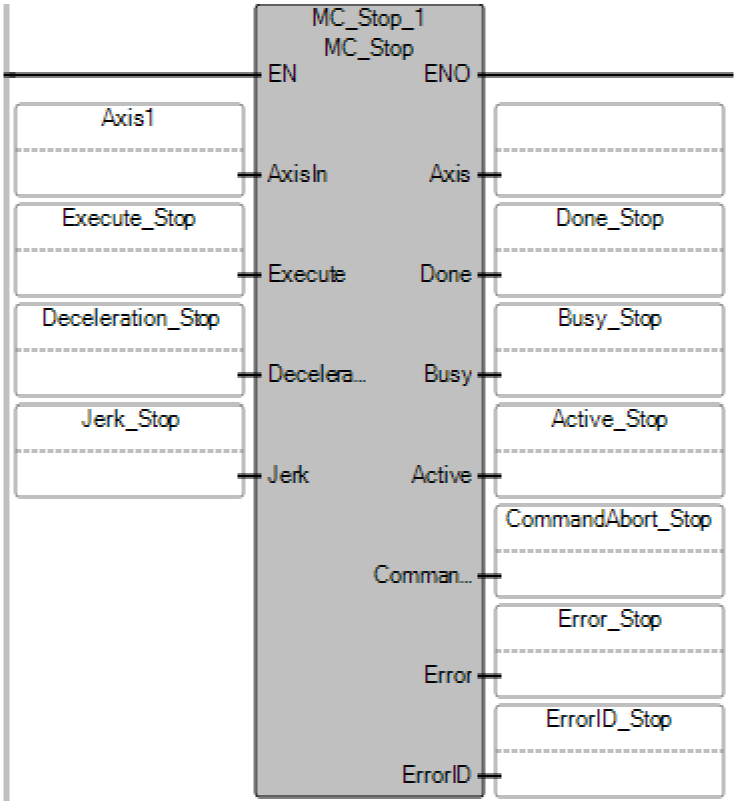
Parameter	Parameter type	Data type	Description
EN	Input	BOOL	Instruction block enable. TRUE - execute current MC_Stop computation. FALSE - there is no computation. Applies only to Ladder Diagram programs.
AxisIn	Input	AXIS_REF	Use the AXIS_REF data type to define the parameters for AxisIn.
Execute	Input	BOOL	TRUE - starts the action at the rising edge. FALSE - not executing.
Deceleration	Input	REAL	Value of the deceleration [u/s^2].
Jerk	Input	REAL	Value of the Jerk [u/s^3].
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.
Axis	Output	AXIS_REF on page 397	Axis output is read-only in Ladder Diagram programs.
Done	Output	BOOL	TRUE - zero velocity was reached, without error during the stop sequence.
Busy	Output	BOOL	TRUE - the function block is not finished.
Active	Output	BOOL	TRUE - indicates the function block has control on the axis.

Parameter	Parameter type	Data type	Description
CommandAborted	Output	BOOL	TRUE - command was overridden by MC_Power(OFF) function block, or ErrorStop.
Error	Output	BOOL	Indicates an error occurred. TRUE - An error is detected. FALSE - No error.
ErrorID	Output	UINT	A unique numeric that identifies the error. The errors for this instruction are defined in Motion control function block error IDs on page 395.

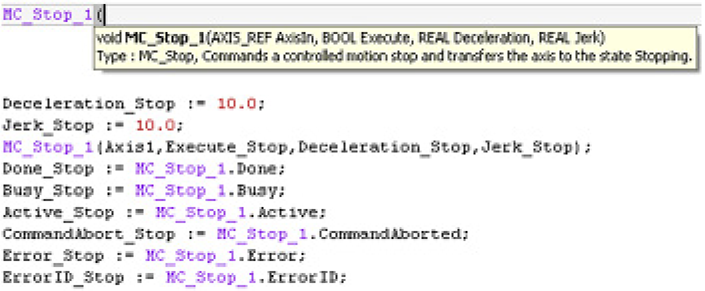
MC_Stop Function Block Diagram example



MC_Stop Ladder Diagram example

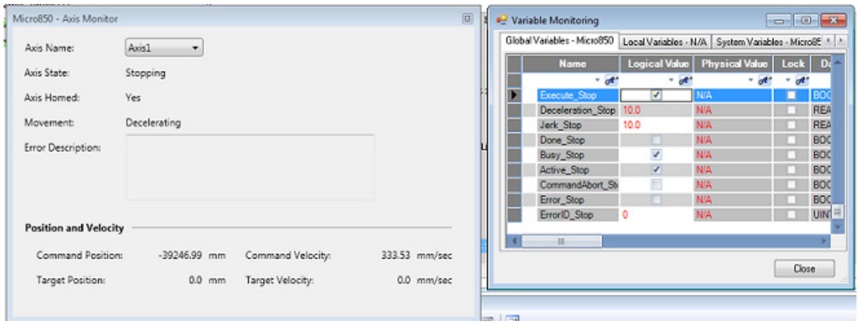


MC_Stop Structured Text example



```
MC_Stop_1(
void MC_Stop_1(AXIS_REF AxisIn, BOOL Execute, REAL Deceleration, REAL Jerk)
Type : MC_Stop, Commands a controlled motion stop and transfers the axis to the state Stopping.

Deceleration_Stop := 10.0;
Jerk_Stop := 10.0;
MC_Stop_1(Axis1,Execute_Stop,Deceleration_Stop,Jerk_Stop);
Done_Stop := MC_Stop_1.Done;
Busy_Stop := MC_Stop_1.Busy;
Active_Stop := MC_Stop_1.Active;
CommandAbort_Stop := MC_Stop_1.CommandAborted;
Error_Stop := MC_Stop_1.Error;
ErrorID_Stop := MC_Stop_1.ErrorID;
```

Results



Micro850 - Axis Monitor

Axis Name: Axis1

Axis State: Stopping

Axis Homed: Yes

Movement: Decelerating

Error Description:

Position and Velocity

Command Position: -39246.99 mm Command Velocity: 333.53 mm/sec

Target Position: 0.0 mm Target Velocity: 0.0 mm/sec

Variable Monitoring

Global Variables - Micro850 | Local Variables - N/A | System Variables - Micro85

Name	Logical Value	Physical Value	Lock	Da
Execute_Stop	☑	N/A	☐	BOO
Deceleration_Stop	10.0	N/A	☐	REA
Jerk_Stop	10.0	N/A	☐	REA
Done_Stop	☐	N/A	☐	BOO
Busy_Stop	☑	N/A	☐	BOO
Active_Stop	☑	N/A	☐	BOO
CommandAbort_St	☐	N/A	☐	BOO
Error_Stop	☐	N/A	☐	BOO
ErrorID_Stop	0	N/A	☐	UIN

MC_TouchProbe (motion control touch probe)

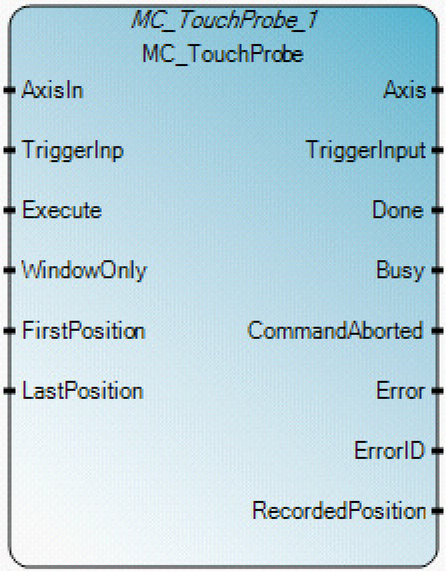
Records an axis position at a trigger event.

Operation details:

- If the window direction (first position --> last position) is in the opposite direction of the motion direction, the touch probe window does not activate.
- If the window setting (FirstPosition or LastPosition) is invalid, the MC_TouchProbe function block reports an error.
- If a second instance of the MC_TouchProbe function block is issued on the same axis, and the first function block instance is in a Busy state, the second function block instance reports an error.
- Only one MC_TouchProbe function block instance should be issued to one axis.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro830, Micro850 and Micro870 controllers that support motion control.



Use this table to help determine the parameter values for this instruction.

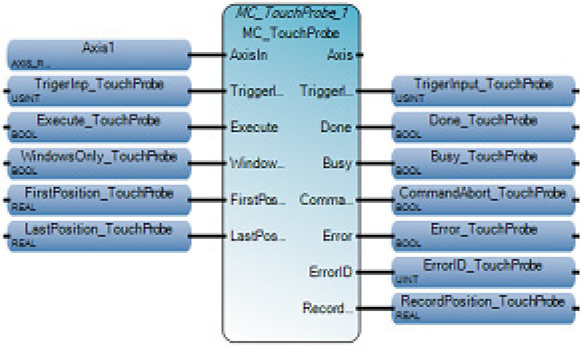
Parameter	Parameter type	Data type	Description
EN	Input	BOOL	Instruction block enable. TRUE - execute current MC_TouchProbe computation. FALSE - there is no computation. Applies only to Ladder Diagram programs.
AxisIn	Input	AXIS_REF FB_AXIS_REF	Use the AXIS_REF data type to define the parameters for AxisIn. For a feedback axis, use the FB_AXIS_REF data type on page 398 to define the parameters for AxisIn.
TriggerInp	Input	USINT	Not supported currently. Configure input trigger in the Axis configuration.
Execute	Input	BOOL	TRUE - starts touch probe recording at the rising edge. FALSE - not executing.
WindowOnly	Input	BOOL	TRUE - only use the window to accept trigger events. Motion resolution is limited to the Motion Engine interval configured by the user. For WindowOnly TouchProbe functionality, there is a maximum response time delay that is equal to the Motion Engine interval for both FirstPosition and LastPosition activation. The maximum possible lag in the triggering position (both FirstPosition and LastPosition) can be calculated by (Motion Engine interval * moving velocity).
FirstPosition	Input	REAL	Start position of the window from where trigger events are accepted (in technical units [u]). Value included in window.
LastPosition	Input	REAL	Stop position of the window from where trigger events are not accepted (in technical units [u]). Value included in window.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.
Axis	Output	AXIS_REF FB_AXIS_REF	Axis output is read-only in Ladder Diagram programs.
TriggerInput	Output	USINT	Not supported currently.
Done	Output	BOOL	TRUE - trigger event was recorded.

Parameter	Parameter type	Data type	Description
Busy	Output	BOOL	TRUE - the function block is not finished.
CommandAborted	Output	BOOL	TRUE - the command was overridden by the MC_Power(OFF), or Error Stop function block.
Error	Output	BOOL	Indicates an error occurred. TRUE - An error is detected. FALSE - No error.
ErrorID	Output	UINT	A unique numeric that identifies the error. The errors for this instruction are defined in Motion control function block error IDs on page 395.
RecordedPosition	Output	REAL	Position where trigger event occurred (in technical units [u]) Motion is an open-loop motion. The axis position at the time the trigger event occurs. If the axis motion is an open-loop motion, the commanded position (not an actual position) at the time the trigger event occurs, if there is no motion delay between the drive and the motor.

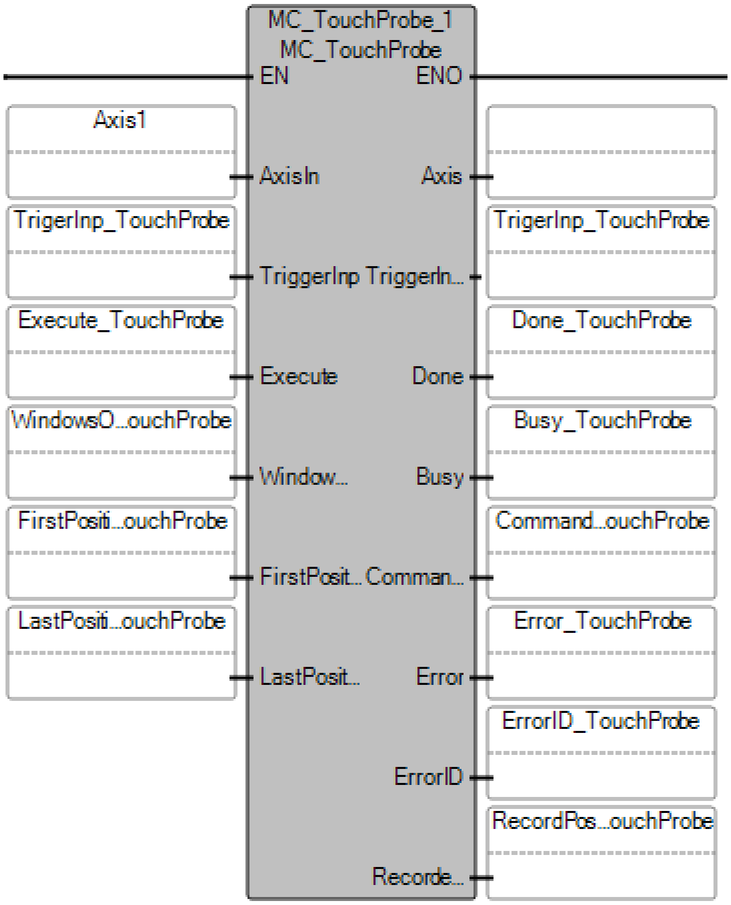
Motion fixed input/output

Motion Signals	PTO0	PTO1	PTO2
PTO pulse	Output_0	Output_1	Output2
PTO direction	Output_3	Output_4	Output_5
Lower (Negative) Limit switch	Input_0	Input_4	Input_8
Upper (Positive) Limit switch	Input_1	Input_5	Input_9
Absolute Home switch	Input_2	Input_6	Input_10
Touch Probe Input switch	Input_3	Input_7	Input_11

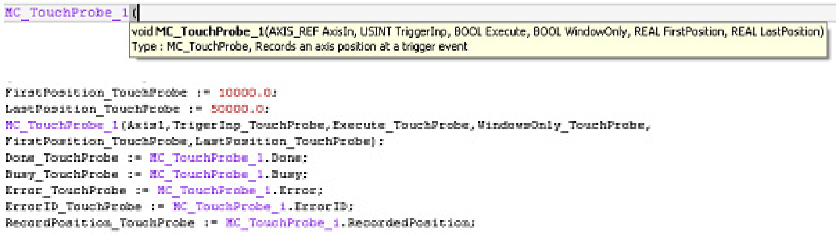
MC_TouchProbe Function Block Diagram example



MC_TouchProbe Ladder Diagram example

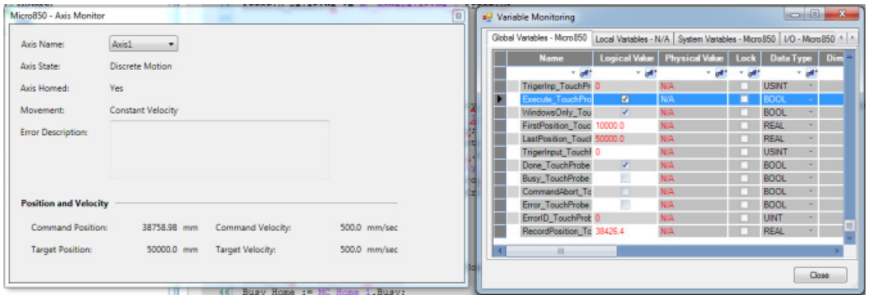


MC_TouchProbe Structured Text example



```
MC_TouchProbe_1(
void MC_TouchProbe_1(AXIS_REF AxisIn, USINT TriggerInp, BOOL Execute, BOOL WindowOnly, REAL FirstPosition, REAL LastPosition)
Type : MC_TouchProbe, Records an axis position at a trigger event

FirstPosition_TouchProbe := 10000.0;
LastPosition_TouchProbe := 50000.0;
MC_TouchProbe_1(Axis1,TrigerInp_TouchProbe,Execute_TouchProbe,WindowsOnly_TouchProbe,
FirstPosition_TouchProbe,LastPosition_TouchProbe);
Done_TouchProbe := MC_TouchProbe_1.Done;
Busy_TouchProbe := MC_TouchProbe_1.Busy;
Error_TouchProbe := MC_TouchProbe_1.Error;
ErrorID_TouchProbe := MC_TouchProbe_1.ErrorID;
RecordPosition_TouchProbe := MC_TouchProbe_1.RecordedPosition;
```

Results



Micro850 - Axis Monitor

Axis Name: Axis1
Axis State: Discrete Motion
Axis Homed: Yes
Movement: Constant Velocity
Error Description:

Position and Velocity

Command Position: 38758.98 mm Command Velocity: 500.0 mm/sec
Target Position: 50000.0 mm Target Velocity: 500.0 mm/sec

Variable Monitoring

Global Variables - Micro850 | Local Variables - N/A | System Variables - Micro850 | I/O - Micro850

Name	Logical Value	Physical Value	Lock	Data Type	Dim
TrigerInp_TouchPr	0	N/A	□	USINT	
Execute_TouchPro	☑	N/A	□	BOOL	
WindowsOnly_Tou	☑	N/A	□	BOOL	
FirstPosition_Touc	10000.0	N/A	□	REAL	
LastPosition_Toucl	50000.0	N/A	□	REAL	
TrigerInput_Touchf	0	N/A	□	USINT	
Done_TouchProbe	☑	N/A	□	BOOL	
Busy_TouchProbe	□	N/A	□	BOOL	
CommandAbort_Tc	□	N/A	□	BOOL	
Error_TouchProbe	□	N/A	□	BOOL	
ErrorID_TouchProt	0	N/A	□	UINT	
RecordPosition_Tc	38426.4	N/A	□	REAL	

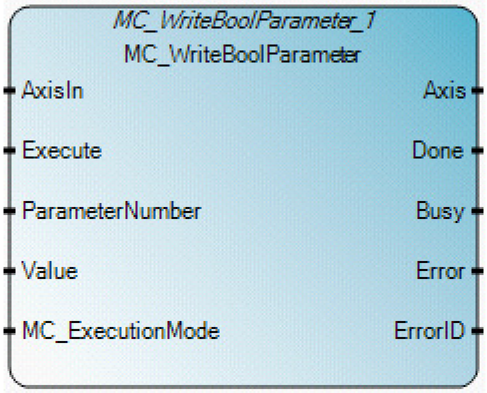
MC_WriteBoolParameter (motion control write BOOL parameter)

Modifies the value of a vendor specific parameter of type BOOL.

The parameters set by the MC_WriteBoolParameter function block are only applied to the application temporarily. They are overwritten by the permanent settings, which are configured by the user in Connected Components Workbench Motion Configuration, when the controller is switched from PRG to RUN mode, or when the power to the controller is cycled OFF and ON.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro830, Micro850 and Micro870 controllers that support motion control.

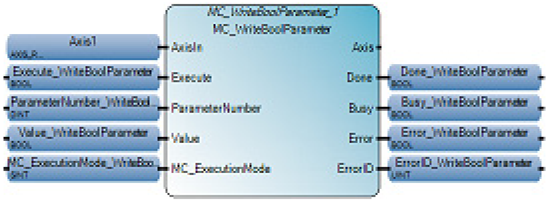


Use this table to help determine the parameter values for this instruction.

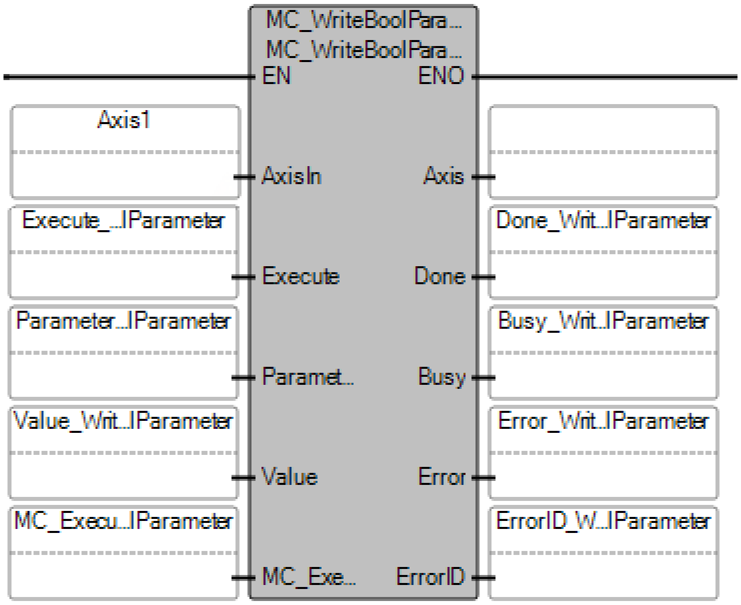
Parameter	Parameter type	Data type	Description
EN	Input	BOOL	Instruction block enable. TRUE - execute current MC_WriteBoolParameter computation. FALSE - the Value output is reset to 0. Applies only to Ladder Diagram programs.
AxisIn	Input	AXIS_REF FB_AXIS_REF	Use the AXIS_REF data type to define the parameters for AxisIn. For a feedback axis, use the FB_AXIS_REF data type to define the parameters for AxisIn.
Execute	Input	BOOL	TRUE - writes the value of the parameter at the rising edge.
ParameterNumber	Input	DINT	Parameter identification. The parameter number is defined in Motion control function block parameter details on page 392.
Value	Input	BOOL	TRUE - the specified parameter has a new value.

Parameter	Parameter type	Data type	Description
MC_ExecutionMode	Input	SINT	Values are: • 0 (*mcImmediately*) - The functionality is immediately valid. • 1 (*mcQueued*) - The new functionality becomes valid when: • all previous motion commands set one of the following output parameters: Done, Aborted or Error • the axis is not in a moving state. When (MC_ExecutionMode = 0), for all parameters except Duty Cycle (1005), this FB can be completed successfully only when the axis state is Disabled or StandStill, When (MC_ExecutionMode = 0), for Parameter Duty Cycle (1005), this FB can be completed successfully except the axis is in Homing or ErrorStop state. For (MC_ExecutionMode = 1), this function block can be successfully completed only when the axis state is Disabled, Standstill, or the on-going motion can complete, ending with Standstill state Other input values are reserved currently, and are considered as invalid parameters.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.
Axis	Output	AXIS_REF	Axis output is read-only in Ladder Diagram programs.
Done	Output	BOOL	TRUE - the parameter was successfully written.
Busy	Output	BOOL	TRUE - the function block is not finished.
Error	Output	BOOL	Indicates an error occurred. TRUE - An error is detected. FALSE - No error.
ErrorID	Output	UINT	A unique numeric that identifies the error. The errors for this instruction are defined in Motion control function block error IDs on page 395.

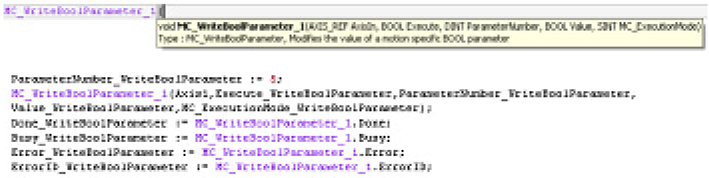
MC_WriteBoolParameter Function Block Diagram example



MC_WriteBoolParameter Ladder Diagram example

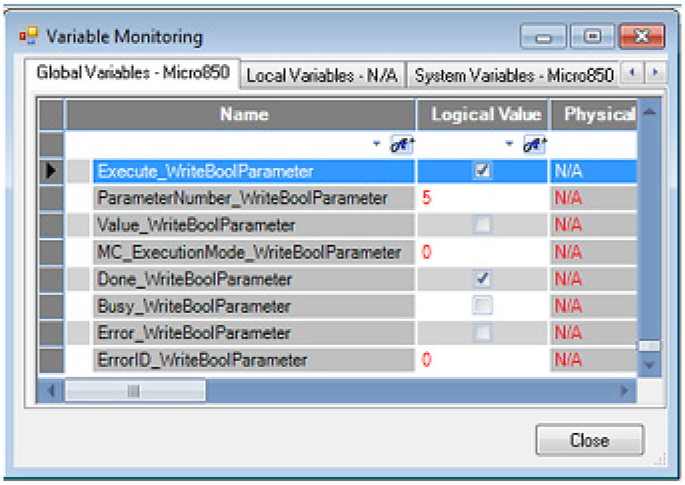


MC_WriteBoolParameter Structured Text example



```
MC_WriteBoolParameter_1(
void MC_WriteBoolParameter_1(AXIS_REF AxisIn, BOOL Execute, DINT ParameterNumber, BOOL Value, SINT MC_ExecutionMode)
Type : MC_WriteBoolParameter, Modifies the value of a motion specific BOOL parameter

ParameterNumber_WriteBoolParameter := 5;
MC_WriteBoolParameter_1(Axis1,Execute_WriteBoolParameter,ParameterNumber_WriteBoolParameter,
Value_WriteBoolParameter,MC_ExecutionMode_WriteBoolParameter);
Done_WriteBoolParameter := MC_WriteBoolParameter_1.Done;
Busy_WriteBoolParameter := MC_WriteBoolParameter_1.Busy;
Error_WriteBoolParameter := MC_WriteBoolParameter_1.Error;
ErrorID_WriteBoolParameter := MC_WriteBoolParameter_1.ErrorID;
```

Results



Name	Logical Value	Physical
Execute_WriteBoolParameter	☑	N/A
ParameterNumber_WriteBoolParameter	5	N/A
Value_WriteBoolParameter	□	N/A
MC_ExecutionMode_WriteBoolParameter	0	N/A
Done_WriteBoolParameter	☑	N/A
Busy_WriteBoolParameter	□	N/A
Error_WriteBoolParameter	□	N/A
ErrorID_WriteBoolParameter	0	N/A

MC_WriteParameter (motion control write parameter)

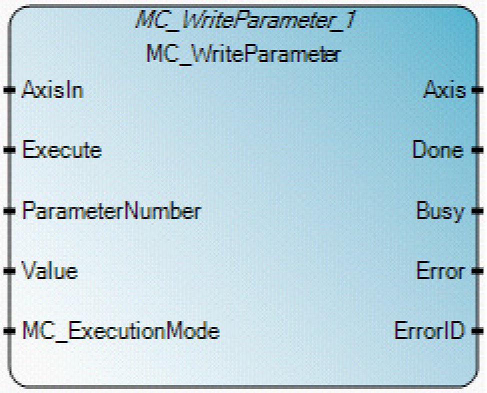
Modifies the value of a vendor specific parameter of type REAL.

The parameters set by the MC_WriteParameter function block are only applied to the application temporarily. They are overwritten by the permanent settings, which are configured by the user in Connected

Components Workbench Motion Configuration, when the controller is switched from PRG to RUN, or when the controller power is cycled OFF and ON.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro830, Micro850 and Micro870 controllers that support motion control.

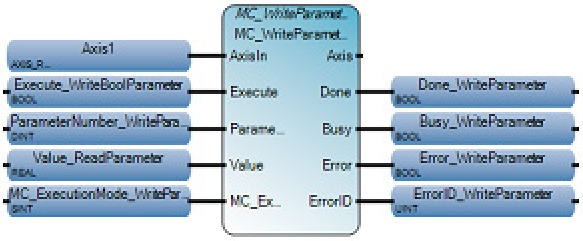


Use this table to help determine the parameter values for this instruction.

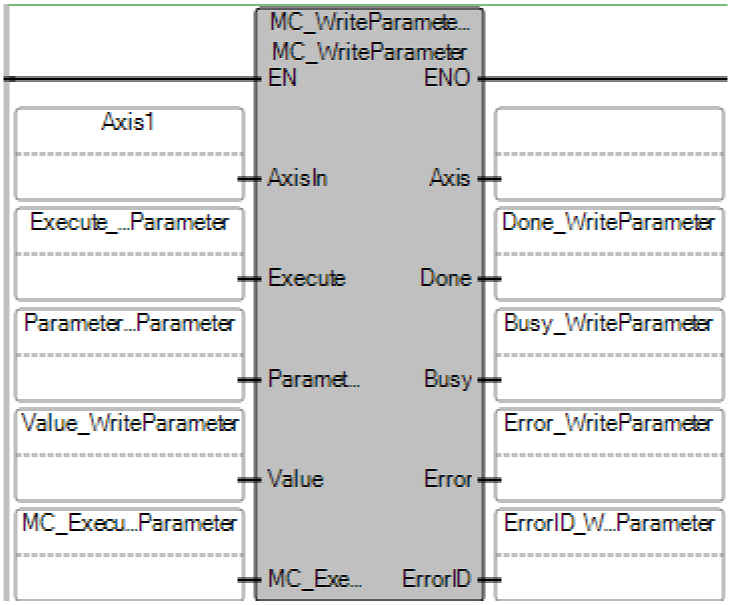
Parameter	Parameter type	Data type	Description
EN	Input	BOOL	Instruction block enable. TRUE - execute current MC_WriteParameter computation. FALSE - there is no computation. Applies only to Ladder Diagram programs.
AxisIn	Input	AXIS_REF FB_AXIS_REF	Use the AXIS_REF data type on page 397 to define the parameters for AxisIn. For a feedback axis, use the FB_AXIS_REF data type on page 398 to define the parameters for AxisIn.
Execute	Input	BOOL	TRUE - writes the value of the parameter at the rising edge.
ParameterNumber	Input	DINT	Parameter identification. The parameter number is defined in Motion control function block parameter details on page 392.
Value	Input	REAL	New value of the specified parameter.

Parameter	Parameter type	Data type	Description
MC_ExecutionMode	Input	SINT	Values are: • 0 (*mcImmediately*) - The functionality is immediately valid. • 1 (*mcQueued*) - The new functionality becomes valid when: • all previous motion commands set one of the following output parameters: Done, Aborted or Error • the axis is not in a moving state • implies that the output parameter Busy is set to FALSE. When (MC_ExecutionMode = 0), for all parameters except Duty Cycle (1005), this FB can be completed successfully only when the axis state is Disabled or StandStill, When (MC_ExecutionMode = 0), for Parameter Duty Cycle (1005), this FB can be completed successfully except the axis is in Homing or ErrorStop state. For (MC_ExecutionMode = 1), this function block can be successfully completed only when the axis state is Disabled, Standstill, or the on-going motion can complete, ending with Standstill state. Other input values are reserved currently and are considered as invalid parameters.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.
Axis	Output	AXIS_REF	Axis output is read-only in Ladder Diagram programs.
Done	Output	BOOL	TRUE - the parameter was successfully written.
Busy	Output	BOOL	TRUE - indicates the function block has control of the axis.
Error	Output	BOOL	Indicates an error occurred. TRUE - An error is detected. FALSE - No error.
ErrorID	Output	UINT	A unique numeric that identifies the error. The errors for this instruction are defined in Motion control function block error IDs on page 395.

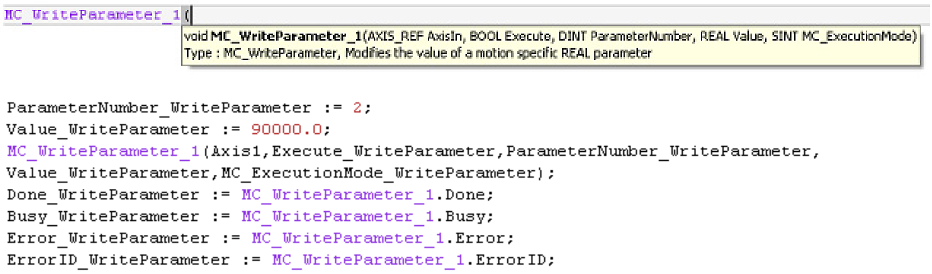
MC_WriteParameter Function Block Diagram example



MC_WriteParameter Ladder Diagram example



MC_WriteParameter Structured Text example

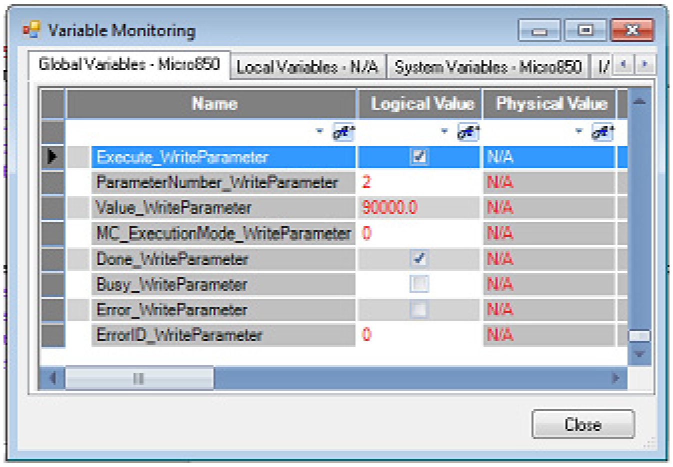


```
MC_WriteParameter_1(
```

void MC_WriteParameter_1(AXIS_REF AxisIn, BOOL Execute, DINT ParameterNumber, REAL Value, SINT MC_ExecutionMode)
Type : MC_WriteParameter, Modifies the value of a motion specific REAL parameter

```
ParameterNumber_WriteParameter := 2;
Value_WriteParameter := 90000.0;
MC_WriteParameter_1(Axis1,Execute_WriteParameter,ParameterNumber_WriteParameter,
Value_WriteParameter,MC_ExecutionMode_WriteParameter);
Done_WriteParameter := MC_WriteParameter_1.Done;
Busy_WriteParameter := MC_WriteParameter_1.Busy;
Error_WriteParameter := MC_WriteParameter_1.Error;
ErrorID_WriteParameter := MC_WriteParameter_1.ErrorID;
```

Results



Name	Logical Value	Physical Value
Execute_WriteParameter	☑	N/A
ParameterNumber_WriteParameter	2	N/A
Value_WriteParameter	90000.0	N/A
MC_ExecutionMode_WriteParameter	0	N/A
Done_WriteParameter	✓	N/A
Busy_WriteParameter	□	N/A
Error_WriteParameter	□	N/A
ErrorID_WriteParameter	0	N/A

Process control instructions

Use Process control instructions to monitor and maintain process loops for quantities such as pressure, temperature, flow rate, and fluid level. Process controls regulate the course by sending an output signal to the control valve.

Instruction	Description
DERIVATE on page 463	Differentiates a real value over a defined cycle time.
FFL on page 465	Loads 8 bit, 16 bit, 32 bit, or 64 bit data into a user-created array called a FIFO stack.
FFU on page 473	Unloads 8 bit, 16 bit, 32 bit, or 64 bit data from a user-created array called a FIFO stack. The data unloads in the same order as loaded using the FFL instruction.
HYSTER on page 478	Performs aBoolean hysteresis on difference of reals.
INTEGRAL on page 480	Integrates a real value during the defined cycle time.
LFL (LIFO load) on page 485	Loads 8 bit, 16 bit, 32 bit, or 64 bit data into a user-created array called a LIFO stack.
LFU (LIFO unload) on page 487	Unloads 8 bit, 16 bit, 32 bit, or 64 bit data from a user-created array called a LIFO stack. The data unloads in the same order as loaded using the LFL instruction.
PWM on page 489	Turns the pulse width modulation (PWM) output for a configured PWM channel ON or OFF.
SCALER on page 492	Scales the input value according to the output range.
STACKINT on page 494	Manages a stack of integer values.
LIMIT on page 505	Restricts integer values to a given interval.
TND on page 504	Stops the current cycle of the user program scan.

DERIVATE

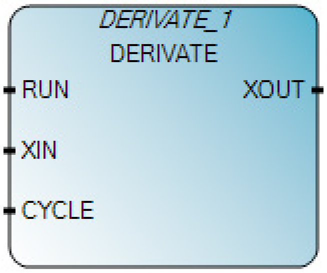
Differentiation of a real value over a defined cycle time.

Operation details:

- If the CYCLE parameter value is less than the cycle timing of the execution of the device, the sampling period is forced to this cycle timing.
- The derivation is performed with a time base of milliseconds.For example, the derivation of an input of 1000 that changes to 2000 over a time period of 1 second results in a value of 1. To convert the output of the instruction to units of seconds, multiply the output by 1000.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

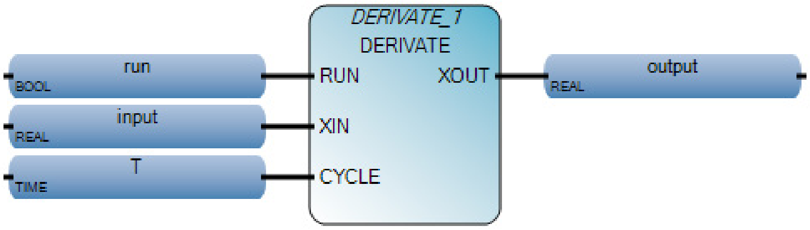
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



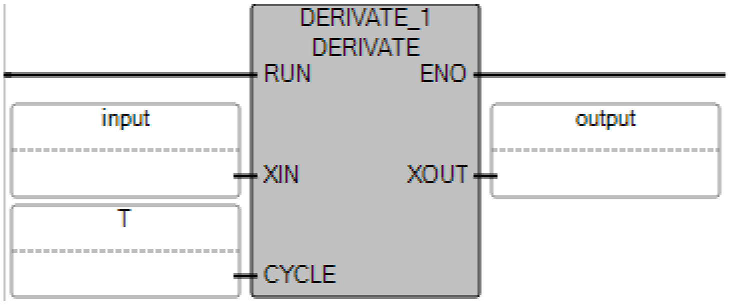
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter type	Data type	Description
RUN	Input	BOOL	Indicates the operational mode of the instruction. TRUE - normal (perform calculation) FALSE - reset
XIN	Input	REAL	Defines the value on which to perform the derivation calculation. The value must be a REAL value.
CYCLE	Input	TIME	Defines the sampling time period over which to collect values. Possible time period values range from 0ms to 49d17h2m47s294ms.
XOUT	Output	REAL	Differentiated output.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.

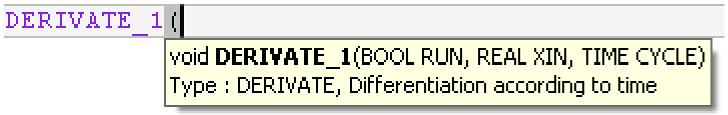
DERIVATE Function Block Diagram example



DERIVATE Ladder Diagram example

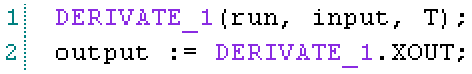


DERIVATE Structured Text example



```
DERIVATE_1(
```

void **DERIVATE_1**(BOOL RUN, REAL XIN, TIME CYCLE)
Type : DERIVATE, Differentiation according to time



```
DERIVATE_1(run, input, T);
output := DERIVATE_1.XOUT;
```

(* ST Equivalence: DERIVATE1 is an instance of a DERIVATE block *)

```
DERIVATE1(manual_mode, sensor_value, t#100ms);
derivated_value := DERIVATE1.XOUT;
```

FFL (FIFO load)

Loads 8 bit, 16 bit, 32 bit, or 64 bit data into a user-created array called a FIFO stack.

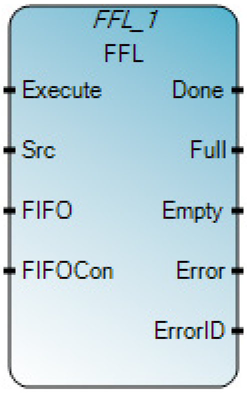
Operation details:

- FFL instruction - non executing mode to executing mode
  - When Execute changes from FALSE to TRUE:
    - Error conditions are verified.
    - The contents of Src are loaded into the FIFO stack in the available position and Position increments by 1 if Position is less than or equal to zero and less than Length.
    - Full is set to TRUE if Length equals Position. Full is FALSE if Position is less than or equal to zero and less than Length.
    - Done is set when the instruction executes successfully.
  - When Execute changes from TRUE to FALSE:
    - Error, Done, and ErrorID are set to FALSE.
    - Empty is set to TRUE if Position is equal to zero.
    - Full is set to TRUE when Length equals Position. Full is FALSE if Position is less than or equal to zero and less than Length.
    - The FFL error conditions are not verified.
  - When Execute changes from TRUE to TRUE:
    - No load operation is performed.
    - Empty is set to TRUE if Position is equal to zero.
    - Full is set to FALSE if Position is less than or equal to zero and less than Length. Full is set to TRUE if Length equals Position.
    - The FFL error conditions are not verified.
  - When Execute changes from FALSE to FALSE:
    - Error, Done, and ErrorID bits are set to FALSE.
    - Full and Empty bits retain values from previous execution state.
  - Empty is set to TRUE if Position is equal to zero.
    - Full is set to TRUE if Length equals Position. Full is FALSE if Position is less than or equal to zero and less than Length.
    - FLL error conditions are not verified.

    - FFL instruction - executing mode to non-executing mode:
        - Error, ErrorID, Done, Empty, and Full retain the Execute mode state.
- To create a single element for the FIFO parameter:
    - Non array:
        - Variable based address such as Fifo1 is allowed for FIFO.
        - Length should be configured as 1.
    - Array:
        - Variable based address such as Fifo1 or Fifo1[0] is allowed for FIFO.
        - Length should be configured as 1.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
Execute	Input	BOOL	Instruction block enable. TRUE - If rising edge is detected, start the FFL operation. FALSE - Rising edge is not detected.
Src	Input	ANY_ELEMENTARY	The Src operand is the address of the value used to fill the currently available position in the FIFO stack. Elementary data types supported for Scr: • DWORD, REAL, TIME, DATE, LWORD, ULINT, LINT, LREAL, BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT. • Sting is not supported. • Array elements such as Array[1] or Array[Index] are supported.

FIFO	Input	ANY_ELEMENTARY	The starting address of the stack. FIFO must be configured the same for the FFL and FFU instructions. Elementary data types supported for FIFO: DWORD, REAL, TIME, DATE, LWORD, ULINT, LINT, LREAL, BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT. • String is not supported. • Only single dimension array is supported for FIFO.
FIFOCon	Input	FF_LF_CON	FIFO configuration and control. The same configuration must be configured for FFL and FFU instructions. Use the FF_LF_CON data type to configure Position and Length.
Done	Output	BOOL	Indicates when the FFU operation is complete. TRUE - Operation completed successfully. FALSE - Operation encountered an error condition or the FFU instruction is not executing.
Empty	Output	BOOL	Indicates when the FIFO stack is empty. TRUE - When Position equals 0. FALSE - When Position is not equal to 0.
Full	Output	BOOL	Indicates when the FIFO stack is full. TRUE - When Length is equal to Position. FALSE - When Position is greater than or equal to zero and less than Length.
Error	Output	BOOL	Indicates the existence of an error condition. TRUE - Operation encountered an error. FALSE - Operation completed successfully or the instruction is not executing.
ErrorID	Output	USINT	A unique numeric that identifies the error. The errors are defined in FFL error codes.

FF_LF_CON data type

Use this table to help determine the parameter values for the FF_LF_CON data type.

Parameter	Data type	Description
Length	UINT	Number of elements used for FIFO operation. Maximum limit is 1024.
Position	USINT	Determines the next available location in the FIFO for the Src entry or removal. Position is the offset of the array. Example 1: • User configured array, arr[0..5]. Initial position is configured as 1. Data is pushed into arr[1] and position increments by position + 1. Example 2: • User configured array as arr[1..5]. Initial position is configured as 1. Data is pushed into arr[2] and position increments by position + 1.

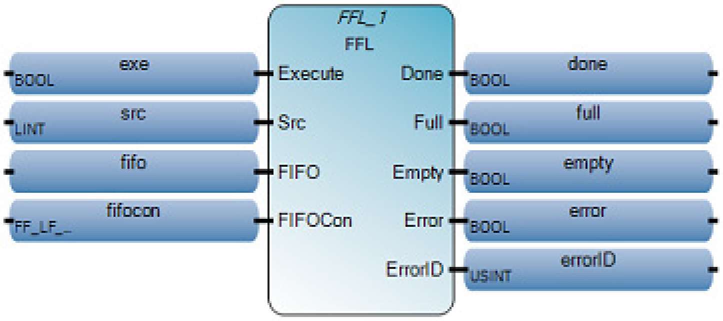
FFL error codes

Use this table to determine the FFL and FFU error codes and descriptions.

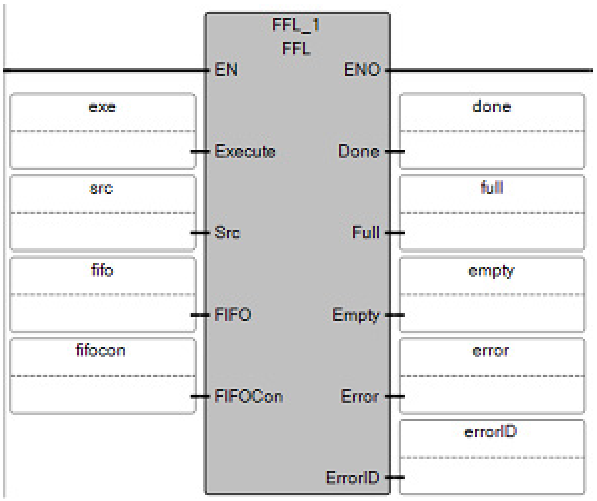
Error code	Error description
0	No error.
1	FFL Src data type is not supported.
2	FFU Dest data type is not supported.
3	FIFO data type is not supported.

4	Src and Dest data type mismatch with the FIFO data type. Corrective Action: FFL Src parameter and FFU Dest parameter data type should match with the FIFO array data type.
5	FIFO - Array dimension is not supported. Corrective Action: FIFO only supports single dimension Arrays.
6	FIFOCon control Length exceeds FIFO array size. Corrective Action: FIFOCon control Length cannot exceed the FIFO array size.
7	FIFOCon Length exceeds the max length.
8	FIFOCon Length is zero.
9	FIFOCon Position exceeds the FIFOCon Length.
10	FFL control Length and Position are equal.
11	FFU control Position is zero.
12	FFL or FFU array dimension is not supported. Corrective Acton: FFL and FFU only support single dimension arrays.
13	FFL or FFU DestOffset exceeds Dest array size.

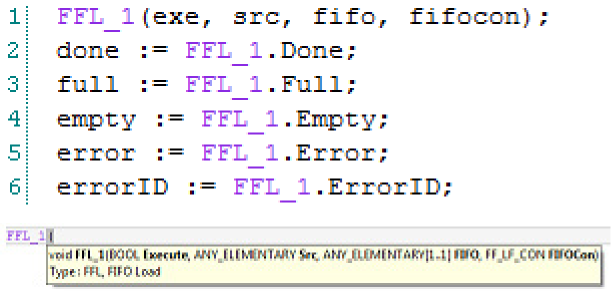
FFL Function Block Diagram example



FFL Ladder Diagram example



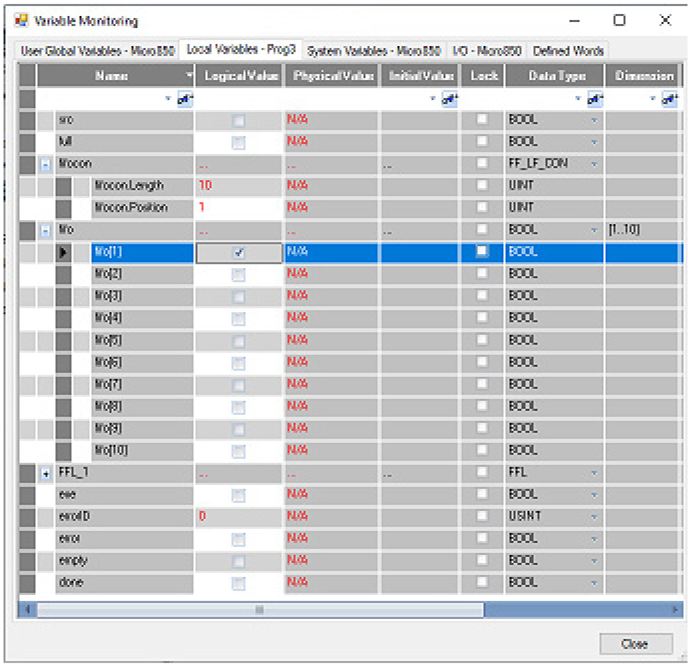
FFL Structured Text example



```
FFL_1(exe, src, fifo, fifocon);
done := FFL_1.Done;
full := FFL_1.Full;
empty := FFL_1.Empty;
error := FFL_1.Error;
errorID := FFL_1.ErrorID;
```

```
FFL_1(
void FFL_1(BOOL Execute, ANY_ELEMENTARY Src, ANY_ELEMENTARY[1..1] FIFO, FF_LF_CON FIFOCon)
Type : FFL, FIFO Load
```

Results

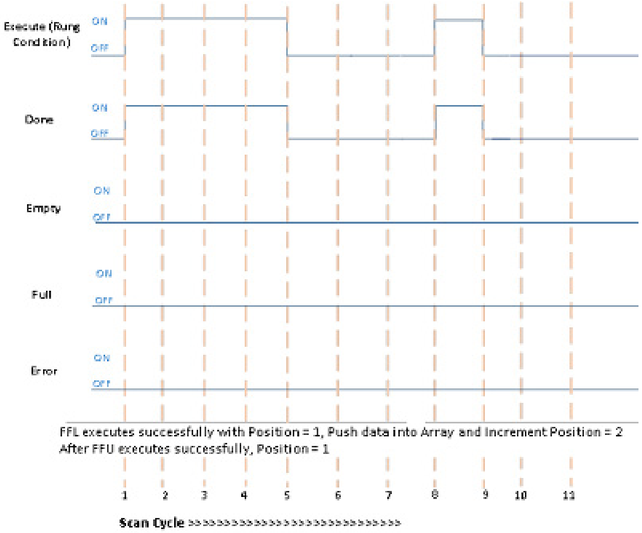


Name	Logical Value	Physical Value	Initial Value	Lock	Data Type	Dimension
src	□	N/A		□	BOOL	
full	□	N/A		□	BOOL	
fifocon	..	..	..	□	FF_LF_CON	
fifocon.Length	10	N/A		□	UINT	
fifocon.Position	1	N/A		□	UINT	
fifo	..	..	..	□	BOOL	[1..10]
fifo[1]	☑	N/A		□	BOOL	
fifo[2]	□	N/A		□	BOOL	
fifo[3]	□	N/A		□	BOOL	
fifo[4]	□	N/A		□	BOOL	
fifo[5]	□	N/A		□	BOOL	
fifo[6]	□	N/A		□	BOOL	
fifo[7]	□	N/A		□	BOOL	
fifo[8]	□	N/A		□	BOOL	
fifo[9]	□	N/A		□	BOOL	
fifo[10]	□	N/A		□	BOOL	
FFL_1	..	..	..	□	FFL	
exe	□	N/A		□	BOOL	
errorID	0	N/A		□	USINT	
error	□	N/A		□	BOOL	
empty	□	N/A		□	BOOL	
done	□	N/A		□	BOOL	

FFL and FFU instruction timing diagrams

The following timing diagram examples describe execution scenarios for the FFL on page 465 (FIFO load) and FFU on page 473(FIFO unload) instructions.

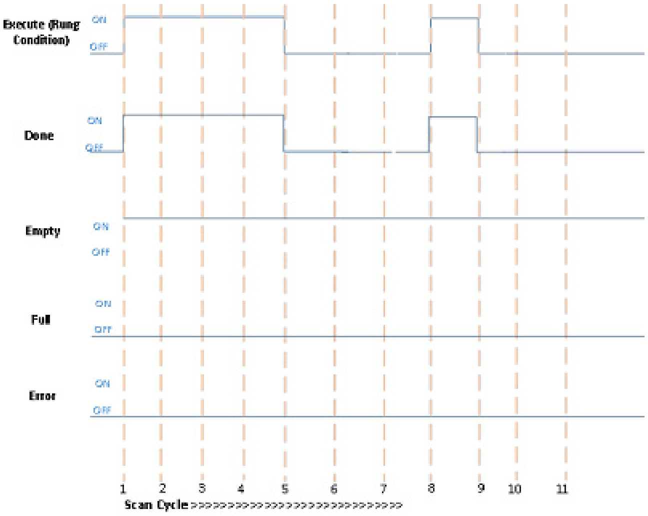
Successful FFL execution followed by successful FFU execution


Execute (Rung Condition)
ON
OFF
Done
ON
OFF
Empty
ON
OFF
Full
ON
OFF
Error
ON
OFF
FFL executes successfully with Position = 1, Push data into Array and Increment Position = 2
After FFU executes successfully, Position = 1
1
2
3
4
5
6
7
8
9
10
11
Scan Cycle >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

Use this table to help determine the parameter values for each scan cycle.

Scan Cycle	Description
1	Rung condition becomes TRUE when: • Execute input bit is TRUE. • Load (push) data to FIFO stack. • Done output bit is TRUE.
2,3,4	No change in rung condition.
5	Rung condition becomes FALSE when: • Execute bit is FALSE. • Done output bit is FALSE.
6, 7	No change in rung condition. • Execute bit is FALSE. • Done output bit is FALSE.
8	Rung goes TRUE when: • Execute input bit is TRUE. • Unload data from FIFO stack. • Done output bit is TRUE.
9	No change in rung condition. • Execute bit is FALSE. • Done output bit is FALSE.
10, 11	No change in rung condition.

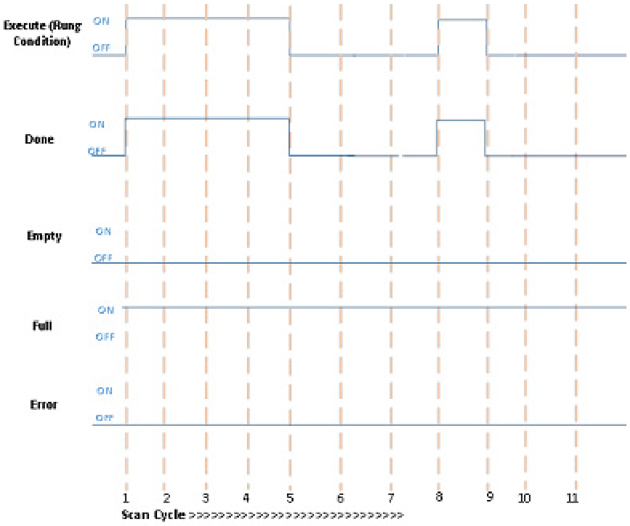
Successful execution when the Empty bit is TRUE



Use this table to help determine the parameter values for each scan cycle.

Scan Cycle	Description
1	Rung condition becomes TRUE when: • Execute input bit is TRUE. Execution starts. • Position is zero. Empty bit is TRUE. • Done output bit is TRUE.
2,3,4	No change in rung condition.
5	Rung condition becomes FALSE when: • Execute bit is FALSE. • Empty bit is TRUE. • Done output bit is FALSE.
6, 7	No change in rung condition.
8	Rung goes TRUE when: • Execute input bit is TRUE. Execution starts. • Empty bit is TRUE. • Done output bit is TRUE.
9	Rung condition becomes FALSE when: • Execute bit is FALSE. • Empty bit is TRUE. • Done output bit is FALSE.
10, 11	No change in rung condition.

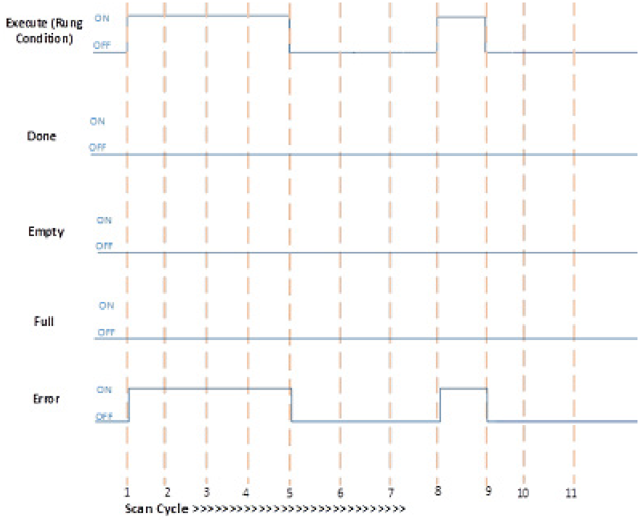
Successful execution when the Empty bit is TRUE



Use this table to help determine the parameter values for each scan cycle.

Scan Cycle	Description
1	Rung condition becomes TRUE when: • Execute input bit is TRUE. Execution starts. • Position is equal to Length, Full bit is TRUE. • Done output bit is TRUE.
2,3,4	No change in rung condition.
5	Rung condition becomes FALSE when: • Execute bit is FALSE. • Full bit is TRUE. • Done output bit is FALSE.
6, 7	No change in rung condition.
8	Rung goes TRUE when: • Execute input bit is TRUE. Execution starts. • Full bit is TRUE. • Done output bit is TRUE.
9	Rung condition becomes FALSE when: • Execute bit is FALSE. • Full bit is TRUE. • Done output bit is FALSE.
10, 11	No change in rung condition.

Error encountered during FFL and FFU execution



Use this table to help determine the parameter values for each scan cycle.

Scan Cycle	Description
	Rung condition becomes TRUE when: • Execute input bit is TRUE. Execution starts. • Error bit is TRUE.
2,3,4	No change in rung condition.
5	Rung condition becomes FALSE when: • Execute bit is FALSE. • Error and ErrorID bits are FALSE.
6, 7	No change in rung condition.
8	Rung goes TRUE when: • Execute input bit is TRUE. Execution starts. • Error bit is TRUE.
9	Rung condition becomes FALSE when: • Execute bit is FALSE. • Error and ErrorID bits are FALSE.
10, 11	No change in rung condition.

FFU (FIFO unload)

Unloads 8 bit, 16 bit, 32 bit, or 64 bit data from a user-created array called a FIFO (first in first out) stack in the same order data was loaded using the FFL instruction.

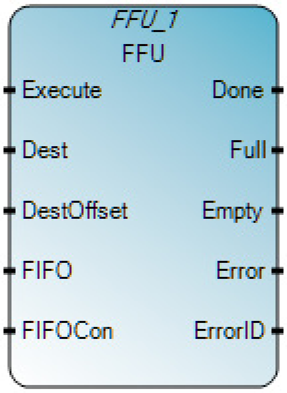
Operation details:

- FFU instruction - non executing mode to executing mode:
  - When Execute changes from FALSE to TRUE:
    - FFU error conditions are verified.

        - Unloads the contents of the FIFO stack at the zero position if Position is greater than zero and less than or equal to Length.
        - Remaining elements shift one position towards zero and the highest element of the FIFO stack is set with zero, Position is decremented by 1.
        - Empty is set to TRUE if Position equals zero.
        - Done is set when the instruction executes successfully.
    - When Execute changes from TRUE to FALSE:
        - Error, Done, and ErrorID are set to FALSE.
        - Empty is set to TRUE if Position is equal to zero.
        - Full is set to TRUE if Length equals Position. Full is set to FALSE if Position is less than or equal to zero and less than Length.
        - The FFU error conditions are not verified.
    - When Execute changes from TRUE to TRUE:
        - No unload operation performed.
        - Empty is set to TRUE if Position is equal to zero.
        - Full is set to TRUE if Length equals Position.
        - The FFU error conditions are not verified.
    - When Execute changes from FALSE to FALSE:
        - Error, Done, and ErrorID are set to FALSE.
        - Empty is set to TRUE if Position is equal to zero.
        - Full is set to TRUE if Length equals Position. Full is set to FALSE if Position is less than or equal to zero and less than Length.
        - The FLU error conditions are not verified.
- FFU instruction - executing mode to non-executing mode:
    - Error, ErrorID, Done, Empty, and Full retain the Execute mode state.
- To create a single element for the FIFO parameter:
    - Non array:
        - Variable based address such as Fifo1 is allowed for FIFO.
        - Length should be configured as 1.
    - Array:
        - Variable based address such as Fifo1 or Fifo1[0] is allowed for FIFO.
        - Length should be configured as 1.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
Execute	Input	BOOL	Instruction block enable. TRUE - If rising edge is detected, start FFU operation. FALSE - Rising edge is not detected.
Dest	Input	ANY_ELEMENTARY	Holds the value that exists in the FIFO stack. Elementary data types supported for Dest: • DWORD, REAL, TIME, DATE, LWORD, ULINT, LINT, LREAL, BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT. • Sting is not supported. • Only supports single dimension arrays such as Array[1] or Array[Index].
DestOffset	Input	UINT	Destination element offset. Element offset if destination is array data type, otherwise offset should be set to 0. For array data type, to unload to the first element, the offset should be set as 0.
FIFO	Input	ANY_ELEMENTARY	The starting address of the stack. FIFO must be configured the same in the FFL and FFU instructions. • Elementary data types supported for FIFO: DWORD, REAL, TIME, DATE, LWORD, ULINT, LINT, LREAL, BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT. • String is not supported. • Only single dimension is supported for FIFO.
FIFOCon	Input	FF_LF_CON	FIFO configuration and control. The same configuration must be configured for FFL and FFU instructions. Use the FF_LF_CON data type to configure Position and Length.
Done	Output	BOOL	Indicates when the FFU operation is complete. TRUE - Operation completed successfully. FALSE - Operation encountered an error condition or the FFU instruction is not executing.
Full	Output	BOOL	Indicates when the FIFO stack is full. TRUE - When Length is equal to Position. FALSE - When Position is greater than zero and less than Length.
Empty	Output	BOOL	Indicates when the FIFO stack is empty. TRUE - When Position equals 0. FALSE - When Position is not equal to 0.

Error	Output	BOOL	Indicates the existence of an error condition. TRUE - Operation encountered an Error. FALSE - Operation completed successfully or the instruction is not executing.
ErrorID	Output	USINT	A unique numeric that identifies the error. The errors are defined in FFU error codes.

FF_LF_CON Data Type

Use this table to help determine the parameter values for the FF_LF_CON data type.

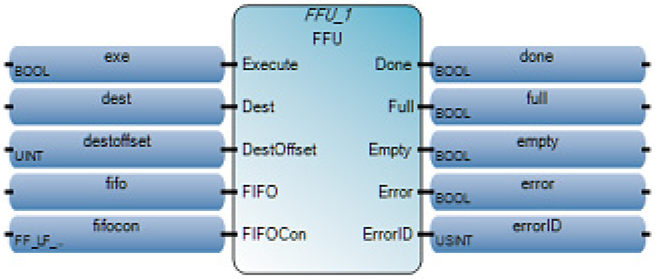
Parameter	Data type	Description
Length	UINT	Number of elements used for FIFO operation. Maximum limit is 1024.
Position	USINT	Determines the next available location in the FIFO for the Src entry or removal. Position is the offset of the array. Example 1: • User configured array, arr[0..5]. Initial position is configured as 1. Data is pushed into arr[1] and position increments by position + 1. Example 2: • User configured array as arr[1..5]. Initial position is configured as 1. Data is pushed into arr[2] and position increments by position + 1.

FFU error codes

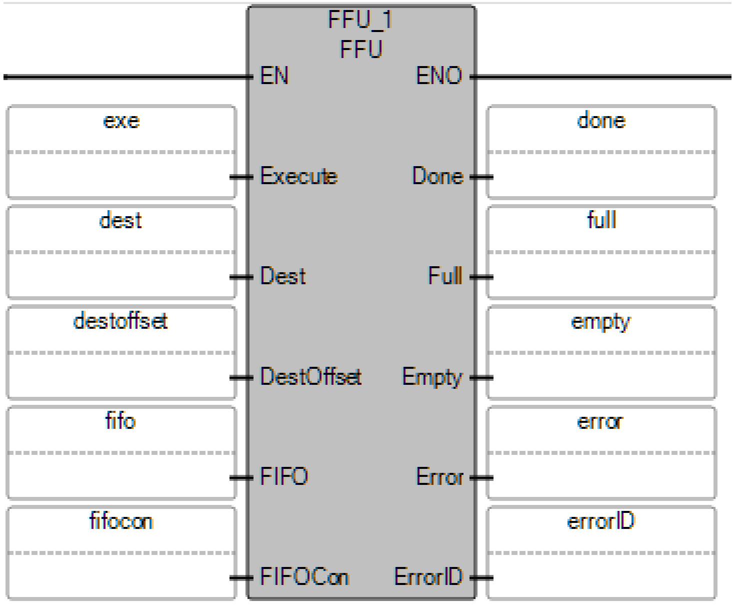
Use this table to determine the FFL and FFU error codes and descriptions.

Error code	Error description
0	No error.
1	FFL Src data type is not supported.
2	FFU Dest data type is not supported.
3	FIFO data type is not supported.
4	Src and Dest data type mismatch with the FIFO data type. Corrective Action: FFL Src parameter and FFU Dest parameter data type should match with the FIFO array data type.
5	FIFO - Array dimension is not supported. Corrective Action: FIFO only supports single dimension Arrays.
6	FIFOCon control Length exceeds FIFO array size. Corrective Action: FIFOCon control Length cannot exceed the FIFO array size.
7	FIFOCon Length exceeds the max length.
8	FIFOCon Length is zero.
9	FIFOCon Position exceeds the FIFOCon Length.
10	FFL control Length and Position are equal.
11	FFU control Position is zero.
12	FFL or FFU array dimension is not supported. Corrective Acton: FFL and FFU only support single dimension arrays.
13	FFL or FFU DestOffset exceeds Dest array size.

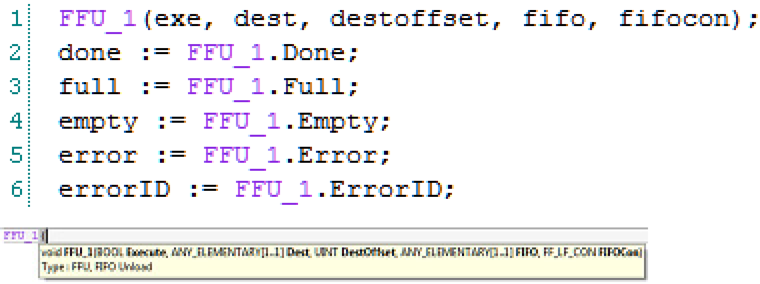
FFU Function Block Diagram example



FFU Ladder Diagram example



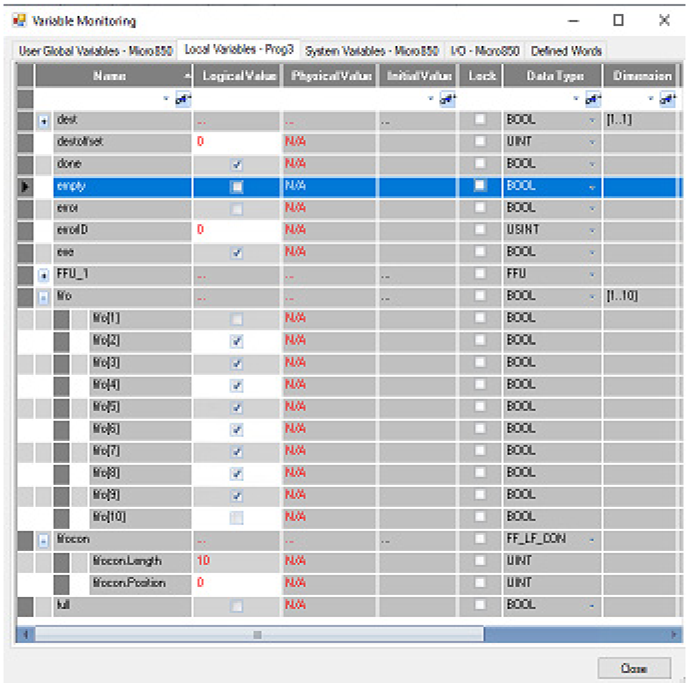
FFU Structured Text example



```
FFU_1(exe, dest, destoffset, fifo, fifocon);
done := FFU_1.Done;
full := FFU_1.Full;
empty := FFU_1.Empty;
error := FFU_1.Error;
errorID := FFU_1.ErrorID;
```

```
FFU_1(
void FFU_1(BOOL Execute, ANY_ELEMENTARY[1..1] Dest, UINT DestOffset, ANY_ELEMENTARY[1..1] FIFO, FF_LF_CON FIFOCon)
Type : FFU, FIFO Unload
```

Results



Variable Monitoring

User Global Variables - Micro850 | Local Variables - Prog3 | System Variables - Micro850 | I/O - Micro850 | Defined Words

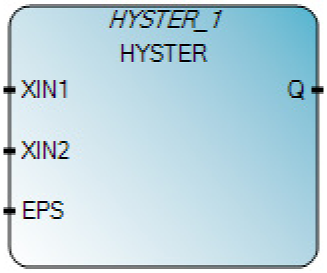
Name	Logical Value	Physical Value	Initial Value	Lock	Data Type	Dimension
dest	..	..	..	□	BOOL	[1..1]
destoffset	0	N/A		□	UINT	
done	☑	N/A		□	BOOL	
empty	□	N/A		□	BOOL	
error	□	N/A		□	BOOL	
errorID	0	N/A		□	USINT	
eee	☑	N/A		□	BOOL	
FFU_1	..	..	..	□	FFU	
fifo	..	..	..	□	BOOL	[1..10]
fifo[1]	□	N/A		□	BOOL	
fifo[2]	☑	N/A		□	BOOL	
fifo[3]	☑	N/A		□	BOOL	
fifo[4]	☑	N/A		□	BOOL	
fifo[5]	☑	N/A		□	BOOL	
fifo[6]	☑	N/A		□	BOOL	
fifo[7]	☑	N/A		□	BOOL	
fifo[8]	☑	N/A		□	BOOL	
fifo[9]	☑	N/A		□	BOOL	
fifo[10]	□	N/A		□	BOOL	
fifocon	..	..	..	□	FF_LF_CON	
fifocon.Length	10	N/A		□	UINT	
fifocon.Position	0	N/A		□	UINT	
full	□	N/A		□	BOOL	

HYSTER (hysteresis)

Boolean hysteresis on difference of reals. Compares the current value of an input with the high limit established by adding the historical amount of lag as measured by hysteresis to the expected value for an input and assessing whether the current value exceeds that limit.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



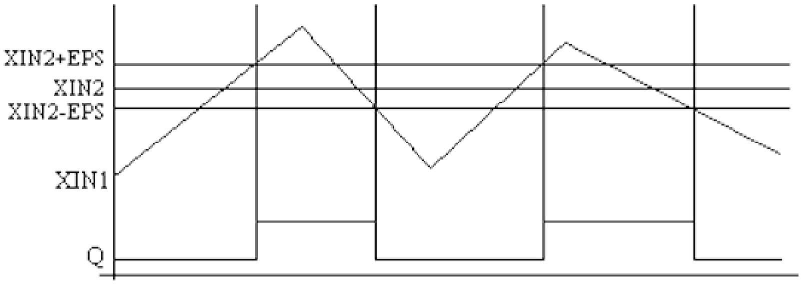
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter type	Data type	Description
EN	Input	BOOL	Instruction block enable. TRUE - execute the instruction block. FALSE - do not execute the instruction block. Applies only to Ladder Diagram programs.
XIN1	Input	REAL	Any real value.

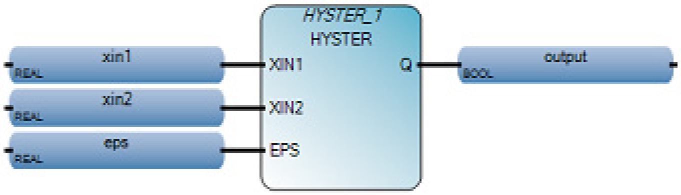
Parameter	Parameter type	Data type	Description
XIN2	Input	REAL	To test if the input value XIN1 has exceeded the high limit defined for this input XIN2 + EPS.
EPS	Input	REAL	Hysteresis value (must be greater than zero).
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.
Q	Output	BOOL	The result of the HYSTER instruction. TRUE - The input exceeded the upper limit but is not below the lower limit. FALSE - The input did not exceed the upper limit.

HYSTER timing diagram example

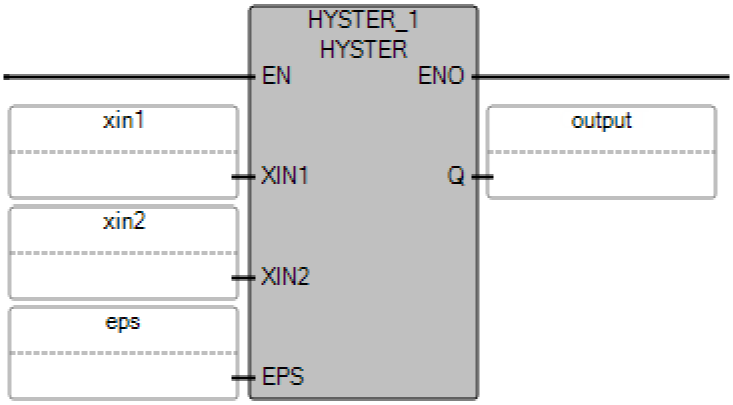
In the following diagram HYSTER is used to assess the performance lag due to motor friction over a 5 second time period. The instruction is run every 10 milliseconds. During the startup phase the motor was operating less efficiently.



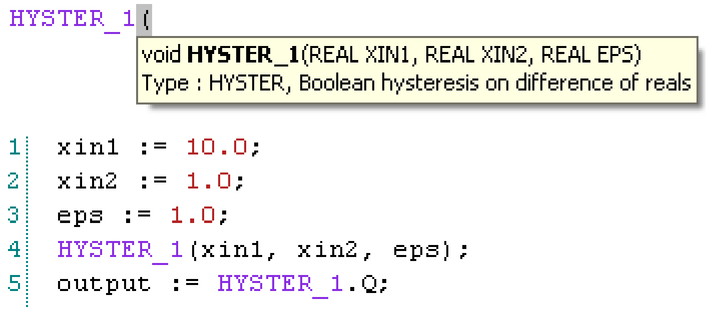
HYSTER Function Block Diagram example



HYSTER Ladder Diagram example



HYSTER Structured Text example



```
HYSTER_1(
```

void **HYSTER_1**(REAL XIN1, REAL XIN2, REAL EPS)
Type : HYSTER, Boolean hysteresis on difference of reals

```
xin1 := 10.0;
xin2 := 1.0;
eps := 1.0;
HYSTER_1(xin1, xin2, eps);
output := HYSTER_1.Q;
```

INTEGRAL

Integrates a real value during the defined cycle time.

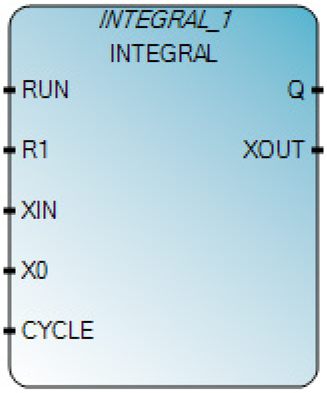
Operation details:

- When the INTEGRAL function block is first initialized, its initial values are not considered. Use the R1 parameter to set the initial values for a calculation.
- To prevent loss of the integrated value, the integration value is not cleared automatically when the controller transitions from PROGRAM to RUN or when the RUN parameter transitions from FALSE to TRUE. Use the R1 parameter to clear the integral value when first transitioning the controller from PROGRAM to RUN mode and when starting a new integration.
- It is recommended that the optional EN or ENO parameters are not used with this function block because the cycle time calculation becomes disrupted when EN is FALSE, resulting in an incorrect integration. If the EN or ENO parameters are used, toggle the R1 parameter with EN equal to TRUE to clear the current result and ensure correct integration.
- Integration is performed with a time base of milliseconds (that is, integrating an input of 1 with an initial value of 0 for 1 second results in a value of 1000). To convert the output of the instruction to units of seconds, divide the output by 1000.
- If the CYCLE parameter value is less than the cycle timing of the execution of the device, the sampling period is forced to the cycle timing.
- XIN sampling and function block executions occur every cycle time + Scan Time Jitter.
- For a given user program, Scan Time Jitter varies from controller to controller.
- The cycle time determines the sensitivity of the Integral function block. Changes occurring in XIN between two samplings (or within the cycle time) are not taken into account when the integral XOUT value is calculated.
- Cycle time and Scan Time Jitter both contribute to the overall inaccuracy of Integral output as shown in the XIN in sync with the

function block execution example and the XIN not in sync with function block execution example.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

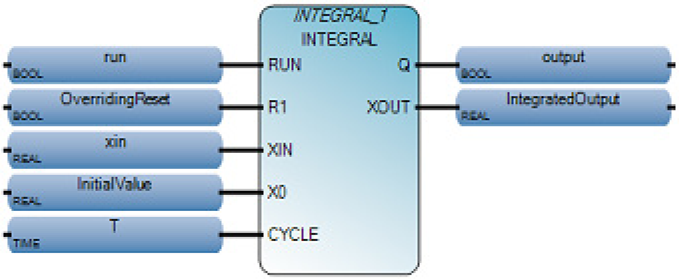
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



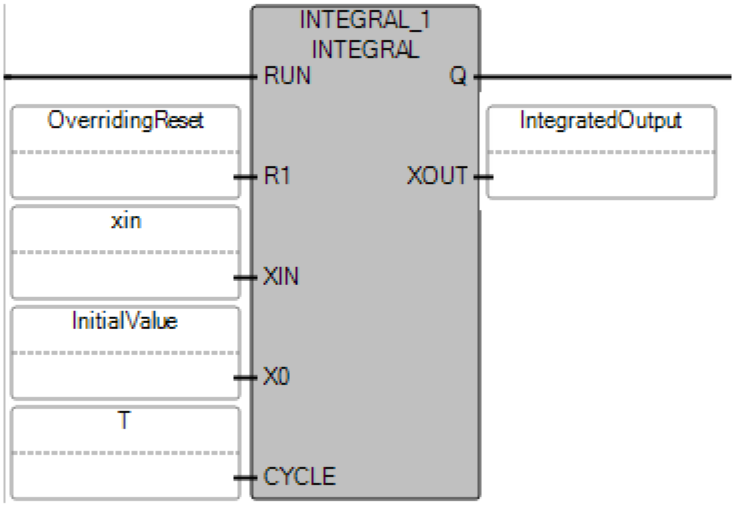
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter type	Data type	Description
RUN	Input	BOOL	Mode: TRUE = integrate / FALSE = hold.
R1	Input	BOOL	Overriding reset.
XIN	Input	REAL	Input: any real value.
X0	Input	REAL	Initial value.
CYCLE	Input	TIME	Sampling period. Possible values range from 0ms to 49d17h2m47s294ms.
Q	Output	BOOL	Not R1.
XOUT	Output	REAL	Integrated output.

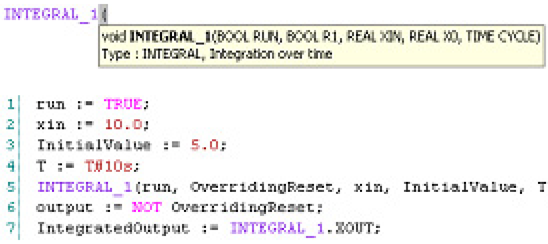
INTEGRAL Function Block Diagram example



INTEGRAL Ladder Diagram example



INTEGRAL Structured Text example



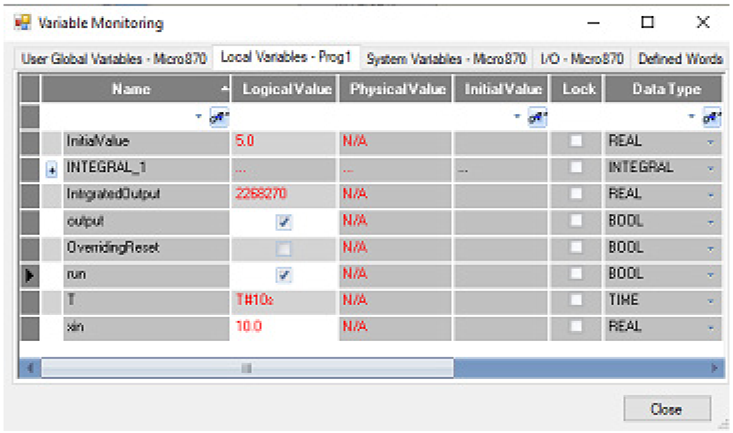
```
INTEGRAL_1(
void INTEGRAL_1(BOOL RUN, BOOL R1, REAL XIN, REAL X0, TIME CYCLE)
Type : INTEGRAL, Integration over time
```

```
run := TRUE;
xin := 10.0;
InitialValue := 5.0;
T := T#10s;
INTEGRAL_1(run, OverridingReset, xin, InitialValue, T);
output := NOT OverridingReset;
IntegratedOutput := INTEGRAL_1.XOUT;
```

(* ST Equivalence: INTEGRAL1 is an instance of a INTEGRAL block *)

```
INTEGRAL1(manual_mode, NOT(manual_mode), sensor_value,
init_value, t#100ms);
controlled_value := INTEGRAL1.XOUT;
```

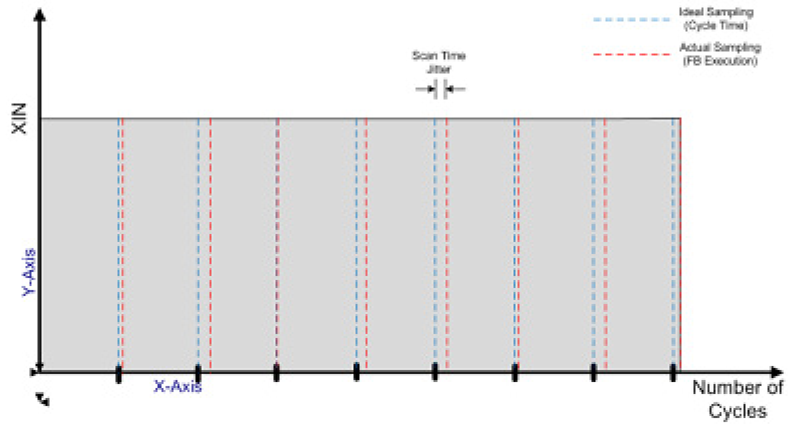
Results

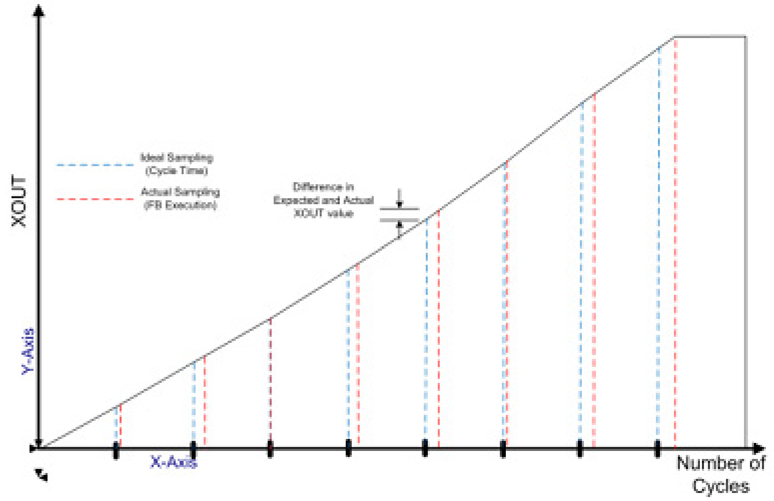


Name	Logical Value	Physical Value	Initial Value	Lock	Data Type
InitialValue	5.0	N/A		☐	REAL
INTEGRAL_1	...	...	...	☐	INTEGRAL
IntegratedOutput	2268270	N/A		☐	REAL
output	☑	N/A		☐	BOOL
OverridingReset	☐	N/A		☐	BOOL
run	☑	N/A		☐	BOOL
T	T#10s	N/A		☐	TIME
xin	10.0	N/A		☐	REAL

XIN in sync with function block execution example

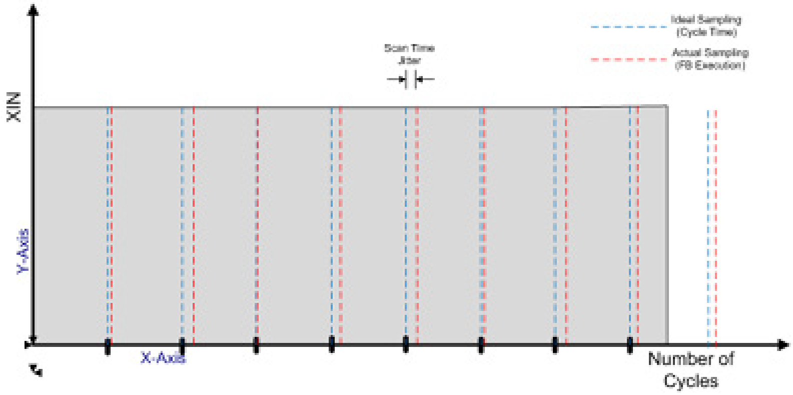
The following images show the effect of Scan Time Jitter on the XOUT value:

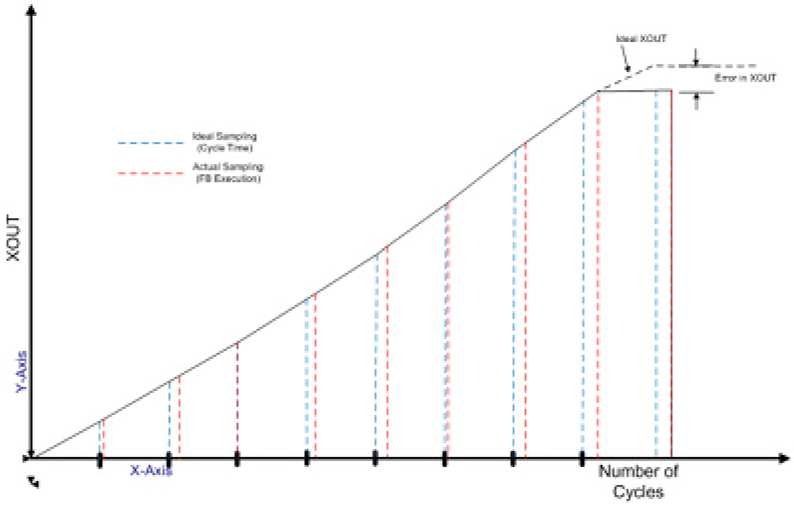




XIN not in sync with function block execution example

The following images show an example in which an error is introduced in the XOUT value of an Integral function block:



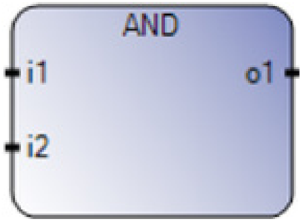


AND

Performs a boolean AND operation between two or more values.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
i1	Input	BOOL	Value in Boolean data type.
i2	Input	BOOL	Value in Boolean data type.
o1	Output	BOOL	Result of the Boolean AND operation of the input values.

AND Structured Text example

(* ST equivalence: *)

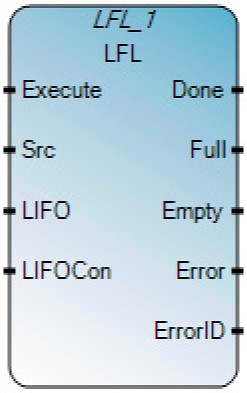
```
bo10 := bi101 AND NOT (bi102);
bo5 := (bi51 AND bi52) AND bi53;
```

LFL(LIFO load)

LFL instruction is used to load the data (8 bit, 16 bit, 32 bit and 64 bit) into a user-created array called LIFO stack. LFL and LFU on page 487 instructions are used in pairs.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.

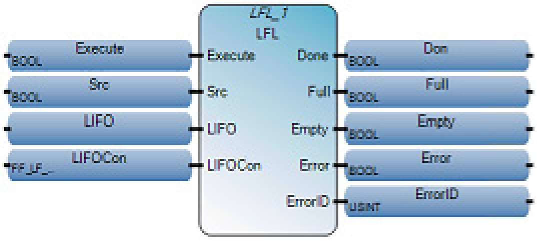


Use this table to help determine the parameter value for this instructions.

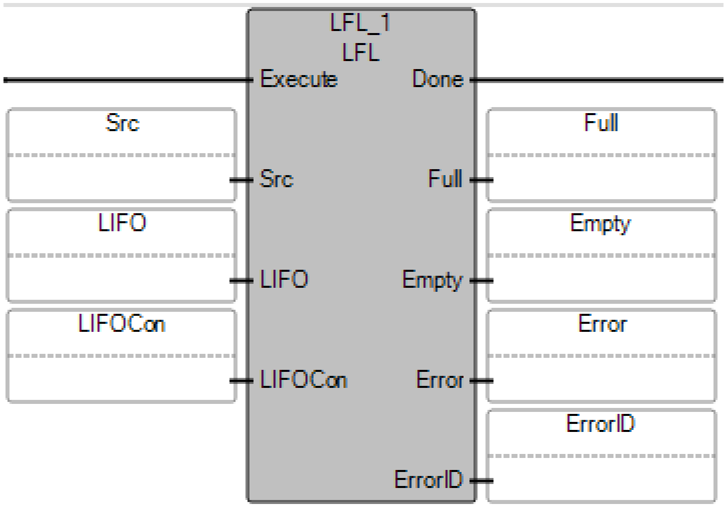
Parameter	Parameter Type	Data Type	Description
Execute	Input	BOOL	Instruction block enable. TRUE - If rising edge is detected, start the LFL operation. FALSE - Rising edge is not detected.
Src	Input	ANY_ELEMENTARY	The Src operand is the address of the value to fill the current available position sin the LIFO stack. Element data types supported: • DWORD, REAL, TIME, DWORD, REAL, TIME, DATE, LWORD, ULINT, LINT, LREAL, BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT.\ • Array element such as Array [1] or Array[Index]. • String is not supported
LIFO	Input	ANY_ELEMENTARY	The starting address of the stack. LIFO must be configured the same for the LFL and LFU instructions. Element data types supported: • DWORD, REAL, TIME, DWORD, REAL, TIME, DATE, LWORD, ULINT, LINT, LREAL, BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT. • Only single dimension is supported. • String is not supported.
LIFOCon	Input	FF_LF_CON	LIFO configuration and control. The same configuration must be configured for LFL and LFU instructions. Use the FF_LF_CON data type to configure Position and Length.
Full	Output	BOOL	Indicates when the LIFO stack is full. TRUE - When Length is equal to Position. FALSE - When Position is greater than or equal to zero and less than Length.

Parameter	Parameter Type	Data Type	Description
Empty	Output	BOOL	Indicates when the LIFO stack is empty. TRUE - When Position equals zero. FALSE - When Position is not equal to zero.
Error	Output	BOOL	Indicate the existence of an error condition. TRUE- Operation encountered an error. FALSE- OPeration completed successfully or the instruction is not executing.
ErrorID	Output	USINT	A unique numeric that identifies the error. The errors are defined in the LFL error codes.
Done	Output	BOOL	Indicate when operation is completed. TRUE - Operation completed successfully. FALSE - Operation encountered an error condition or the LFL instruction is not executing.

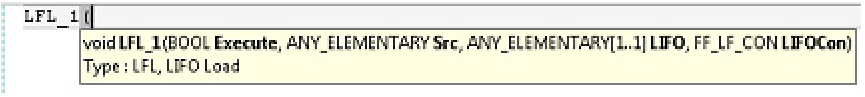
LFL Function Block Diagram example



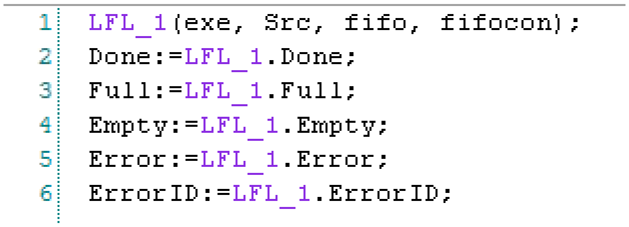
LFL Ladder Diagram example



LFL Structured Text example



```
LFL_1(
void LFL_1(BOOL Execute, ANY_ELEMENTARY Src, ANY_ELEMENTARY[1..1] LIFO, FF_LF_CON LIFOCon)
Type : LFL, LIFO Load
```



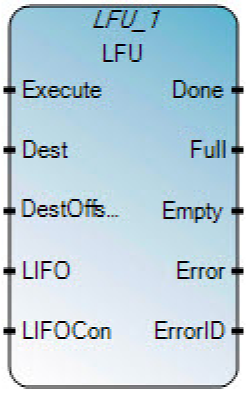
```
LFL_1(exe, Src, fifo, fifocon);
Done:=LFL_1.Done;
Full:=LFL_1.Full;
Empty:=LFL_1.Empty;
Error:=LFL_1.Error;
ErrorID:=LFL_1.ErrorID;
```

LFU(LIFO unload)

LFU instruction unloads data (8 bit, 18 bit, 32 bit, 64 bit) from a user - created array called LIFO stack. LFU and LFL on page 485 instructions are used in pairs.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.

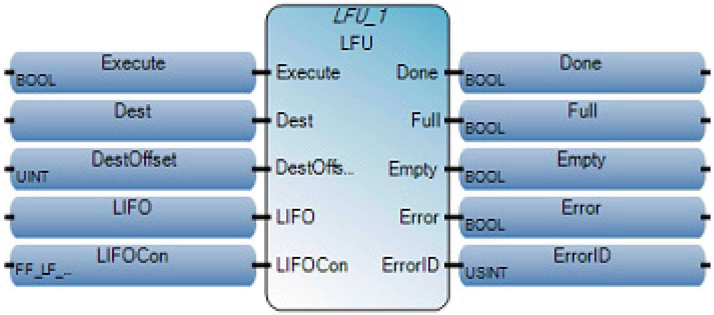


Use this table to help determine the parameter value for this instructions.

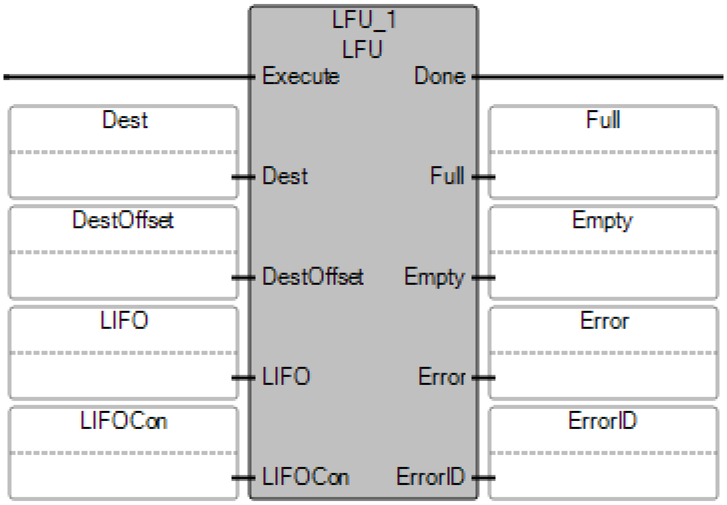
Parameter	Parameter Type	Data Type	Description
Execute	Input	BOOL	Instruction block enable. TRUE - If rising edge is detected, start the LFU operation. FALSE - Rising edge is not detected.
Dest	Input	ANY_ELEMENTARY	Holds the value that exists in the LIFO stack. Elementary data types supported for Dest: • DWORD, REAL, TIME, DATE, LWORD, ULINT, LINT, LREAL, BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT. • Sting is not supported. • Only supports single dimension arrays such as Array[1] or Array[Index].
DestOffset	Input	UINT	Destination element offset. Element offset if destination is array data type, otherwise offset should be set to 0. For array data type, to unload to the first element, the offset should be set as 0.

Parameter	Parameter Type	Data Type	Description
LIFO	Input	ANY_ELEMENTARY	The starting address of the stack. LIFO must be configured the same for the LFL and LFU instructions. Element data types supported: • DWORD, REAL, TIME, DWORD, REAL, TIME, DATE, LWORD, ULINT, LINT, LREAL, BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT. • Only single dimension is supported. • String is not supported.
LIFOCon	Input	FF_LF_CON	LIFO configuration and control. The same configuration must be configured for LFL and LFU instructions. Use the FF_LF_CON data type to configure Position and Length.
Full	Output	BOOL	Indicates when the LIFO stack is full. TRUE - When Length is equal to Position. FALSE - When Position is greater than or equal to zero and less than Length.
Error	Output	BOOL	Indicate the existence of an error condition. TRUE- Operation encountered an error. FALSE- OPeration completed successfully or the instruction is not executing.
ErrorID	Output	USINT	A unique numeric that identifies the error. The errors are defined in the LFU error codes.
Done	Output	BOOL	Indicate when operation is completed. TRUE - Operation completed successfully. FALSE - Operation encountered an error condition or the LFU instruction is not executing.

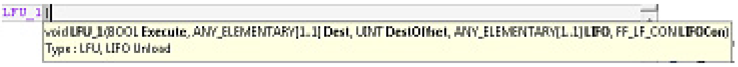
LFU Function Block Diagram example



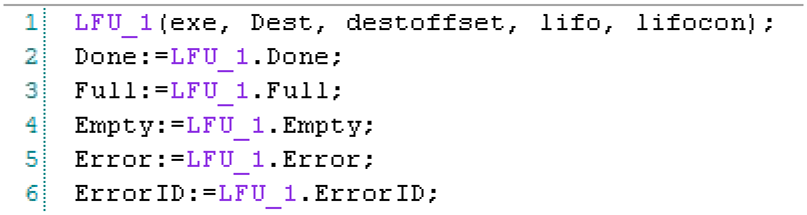
LFU Ladder Diagram example



LFU Structured Text example



```
LFU_1
void LFU_1(BOOL Execute, ANY_ELEMENTARY[1..1] Dest, UINT DestOffset, ANY_ELEMENTARY[1..1] LIFO, FF_LF_CON LIFOCon)
Type : LFU, LIFO Unload
```



```
LFU_1(exe, Dest, destoffset, lifo, lifocon);
Done:=LFU_1.Done;
Full:=LFU_1.Full;
Empty:=LFU_1.Empty;
Error:=LFU_1.Error;
ErrorID:=LFU_1.ErrorID;
```

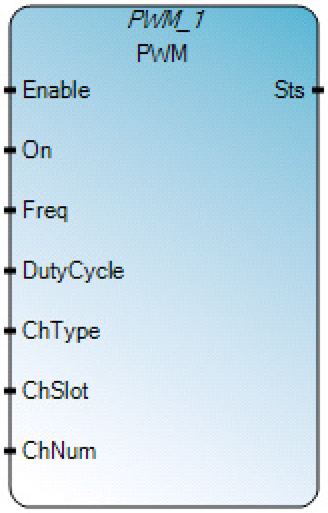
PWM (Pulse Width Modulation)

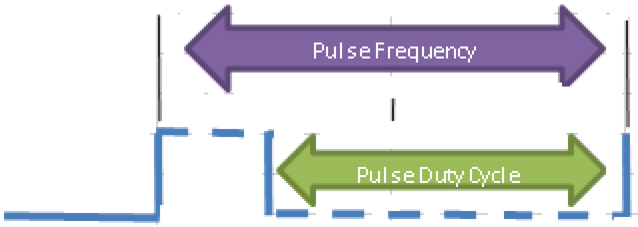
Turns the PWM (Pulse Width Modulation) output for a configured PWM channel ON or OFF.

This instruction block is used with Micro820 2080-LC20-20QBB controllers and supports one PWM channel using the embedded output channel 6.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies only to the Micro820 controller.



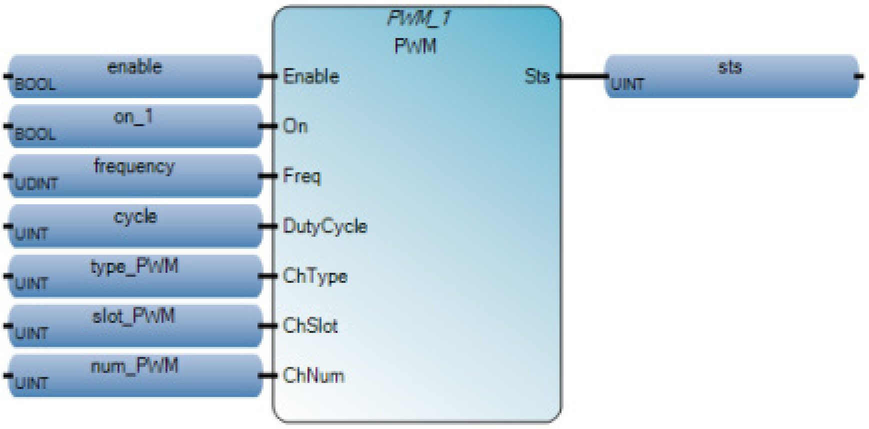


Use this table to help determine the parameter values for this instruction.

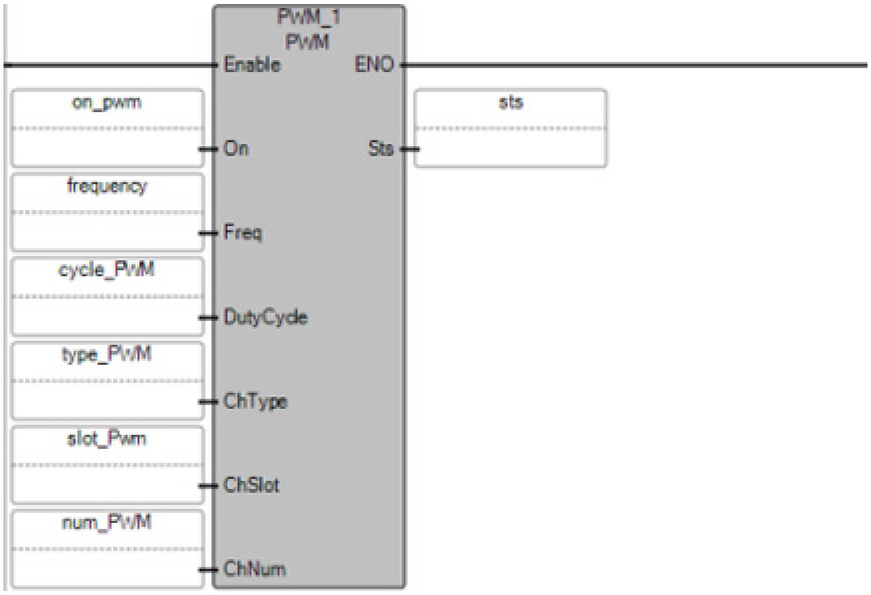
Parameter	Parameter type	Data type	Description
Enable	Input	BOOL	Instruction block enable. This level is instruction block triggered. TRUE - Update Sts. PWM is made active or inactive depending on the On input parameter and valid configuration. FALSE - Sts is only updated. PWM state (active or inactive) is not affected.
On	Input	BOOL	Turns the PWM output ON/Active or OFF/Inactive. TRUE - PWM output is active or continues to be active with latest valid configuration. Output LED is ON when PWM is active, even if duty cycle is set to 0%. FALSE - PWM output is inactive if configuration is also valid.
Freq	Input	UDINT	Pulse Frequency. • 1 - 100000 Hz
DutyCycle	Input	UINT	Pulse Duty Cycle. • 0 - 1000 (0% - 100%)
ChType	Input	UINT	Channel Type • 0 - Embedded • 1 - Plugin • 2 - Expansion
ChSlot	Input	UINT	Channel Slot • 0 - Embedded
ChNum	Input	UINT	Channel Number • 0 - PWM CH0
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.

Parameter	Parameter type	Data type	Description
Sts	Output	UINT	PWM status codes: • 00 - Function block not enabled (no operation.) • 01 - PWM configuration successful. • 02 - Invalid duty cycle. • 03 - Invalid Frequency. • 04 - Invalid Channel Type. • 05 - Invalid Channel Slot. • 06 - Invalid Channel Number. • 07 - Invalid Catalog. PWM feature is not supported in the catalog being used.

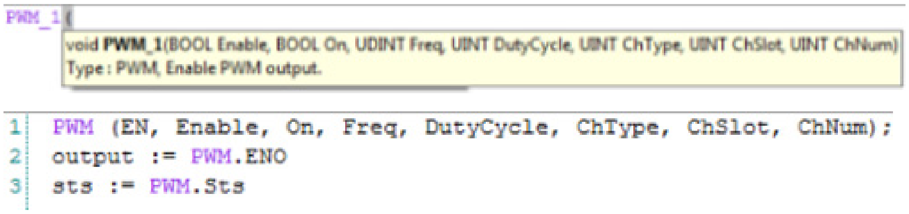
PWM Function Block Diagram example



PWM Ladder Diagram example



PWM Structured Text example



```
PWM_1(
void PWM_1(BOOL Enable, BOOL On, UDINT Freq, UINT DutyCycle, UINT ChType, UINT ChSlot, UINT ChNum)
Type : PWM, Enable PWM output.
```

```
PWM (EN, Enable, On, Freq, DutyCycle, ChType, ChSlot, ChNum);
output := PWM.ENO
sts := PWM.Sts
```

SCALER (scale)

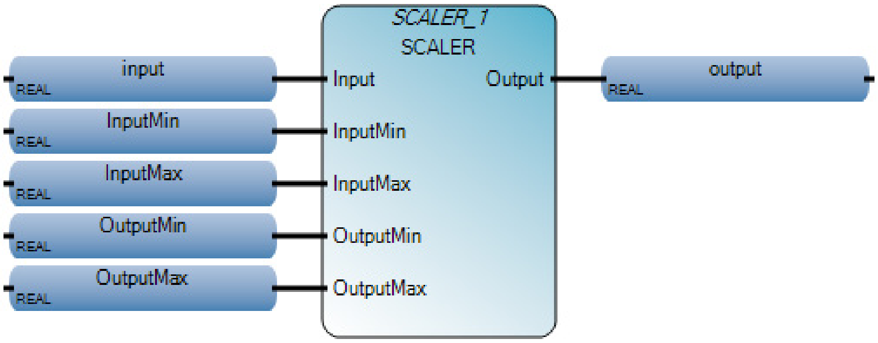
Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.,

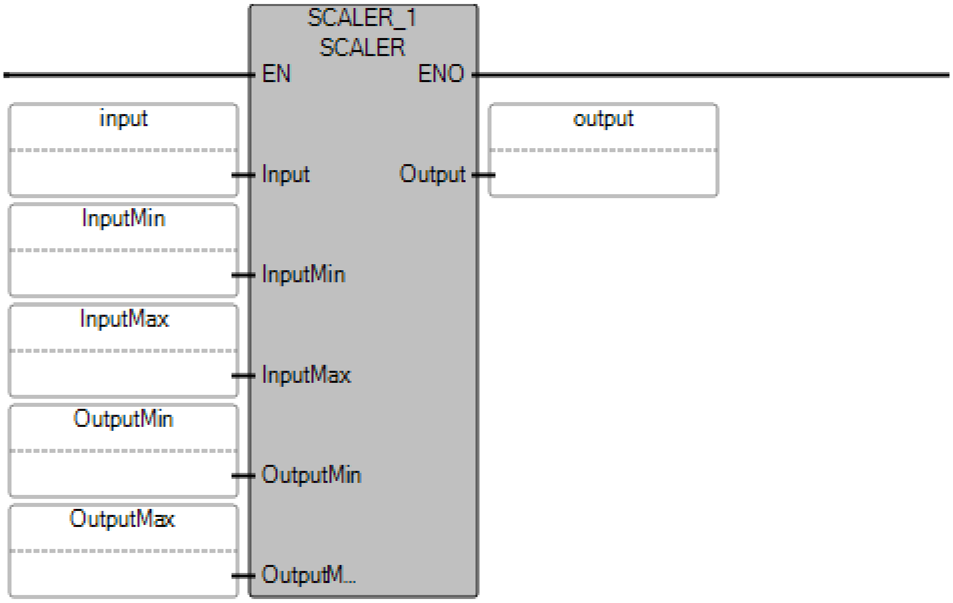
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter type	Data type	Description
EN	Input	BOOL	Instruction block enable. TRUE - execute the scaling equation. FALSE - there is no scaling equation. Applies only to Ladder Diagram programs.
Input	Input	REAL	Input signal. Input is not limited by InputMin and InputMax. To limit Input, a LIM_ALRM instruction is needed to condition Input before it's entered to the SCALER instruction.
InputMin	Input	REAL	Determine the slope and offset value.
InputMax	Input	REAL	Determine the slope and offset value.
OutputMin	Input	REAL	Determine the slope and offset value.
OutputMax	Input	REAL	Determine the slope and offset value.
Output	Output	REAL	Scaled Output. Output is not clamped between OutputMin and OutputMax.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.

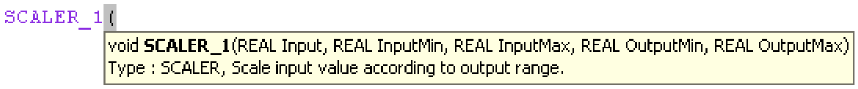
SCALER Function Block Diagram example



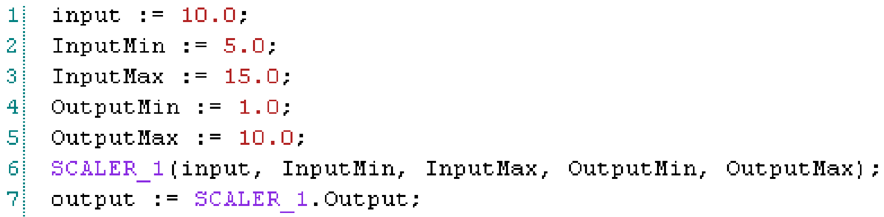
SCALER Ladder Diagram example



SCALER Structured Text example



```
SCALER_1(
void SCALER_1(REAL Input, REAL InputMin, REAL InputMax, REAL OutputMin, REAL OutputMax)
Type : SCALER, Scale input value according to output range.
```

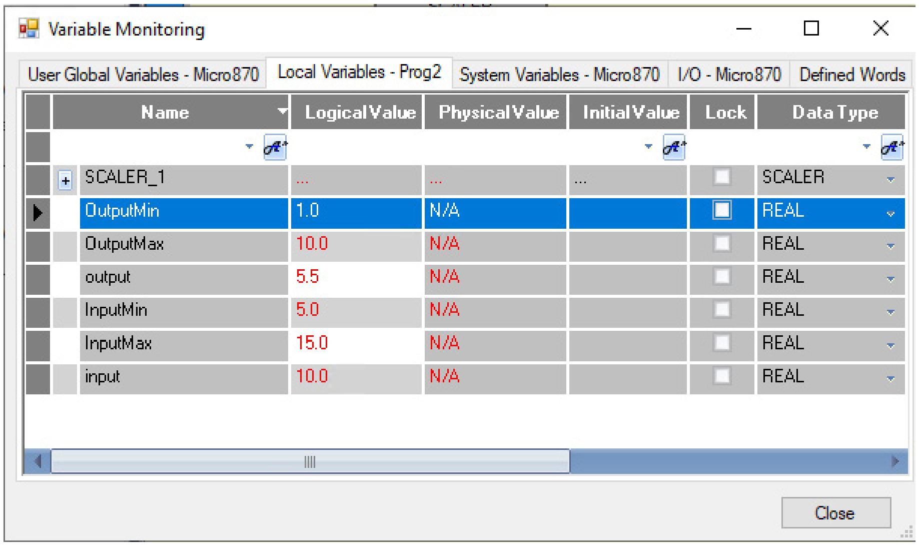


```
input := 10.0;
InputMin := 5.0;
InputMax := 15.0;
OutputMin := 1.0;
OutputMax := 10.0;
SCALER_1(input, InputMin, InputMax, OutputMin, OutputMax);
output := SCALER_1.Output;
```

(* ST equivalence: SCALER1 is an instance of SCALER block *)

```
SCALER1(Signal_In, 4.0, 20.0 , 0.0 , 150.0 ) ;
Out_Temp := SCALER1.Output ;
```

Results



Name	Logical Value	Physical Value	Initial Value	Lock	Data Type
SCALER_1	...	...	...	□	SCALER
OutputMin	1.0	N/A		□	REAL
OutputMax	10.0	N/A		□	REAL
output	5.5	N/A		□	REAL
InputMin	5.0	N/A		□	REAL
InputMax	15.0	N/A		□	REAL
input	10.0	N/A		□	REAL

STACKINT (stack integers)

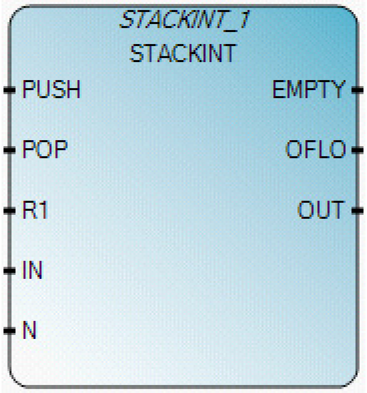
Manages a stack of integer values.

Operation details:

- STACKINT includes a rising edge detection for both PUSH and POP commands. The maximum size of the stack is 128. The OFLO value is valid only after a reset (R1 has been set to TRUE at least once and back to FALSE).
- The application defined stack size (N) cannot be less than 1 or greater than 128.
  - If N < 1, STACKINT assumes a size of 1.
  - If N > 128, STACKINT assumes a size of 128.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.

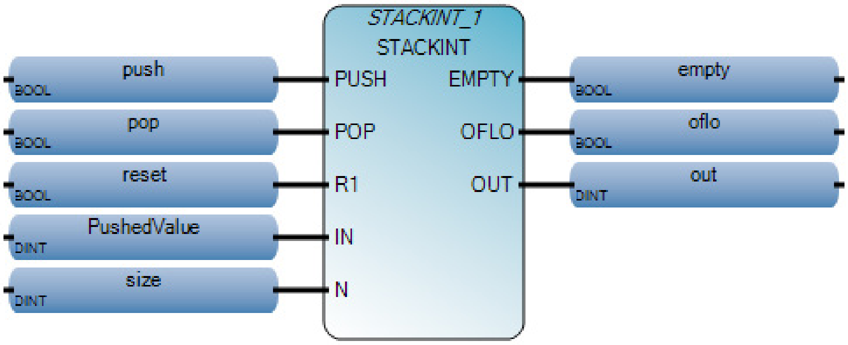


Use this table to help determine the parameter values for this instruction.

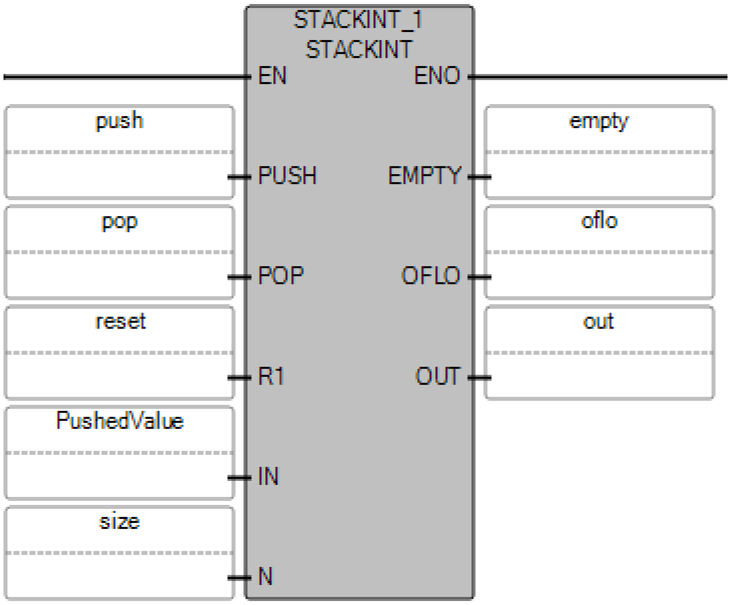
Parameter	Parameter type	Data type	Description

Parameter	Parameter type	Data type	Description
PUSH	Input	BOOL	TRUE - Rising edge detected, on PUSH command. Adds the IN value on the top of the stack. FALSE - Rising edge not detected on PUSH command.
POP	Input	BOOL	TRUE - Rising edge detected, on PUSH command. Deletes the last value pushed to the top of the stack. FALSE - Rising edge not detected on POP command.
R1	Input	BOOL	TRUE - Resets the stack to its empty state. FALSE - No reset.
IN	Input	DINT	Pushed value.
N	Input	DINT	Application defined stack size.The maximum size of the stack is 128
EMPTY	Output	BOOL	TRUE - the stack is empty. FALSE - the stack contains values.
OFLO	Output	BOOL	TRUE - Overflow, the stack is full and R1 has been set to TRUE at least once and back to FALSE. FALSE - the stack size is 128 or less. No overflow.
OUT	Output	DINT	Value at the top of the stack. OUT equals 0 when OFLO is TRUE.

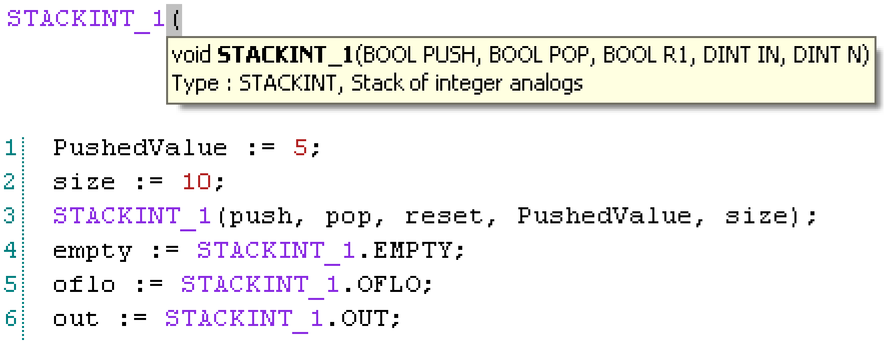
STACKINT Function Block Diagram example



STACKINT Ladder Diagram example



STACKINT Structured Text example



```
STACKINT_1(
```

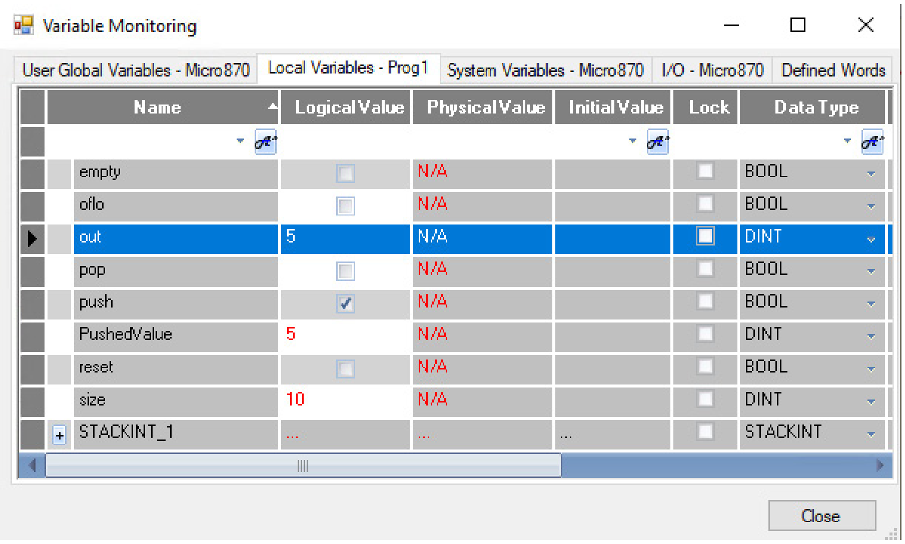
void **STACKINT_1**(BOOL PUSH, BOOL POP, BOOL R1, DINT IN, DINT N)
Type : STACKINT, Stack of integer analogs

```
PushedValue := 5;
size := 10;
STACKINT_1(push, pop, reset, PushedValue, size);
empty := STACKINT_1.EMPTY;
oflo := STACKINT_1.OFLO;
out := STACKINT_1.OUT;
```

(* ST Equivalence: STACKINT1 is an instance of a STACKINT block *)

```
STACKINT1(err_detect, acknowledge, manual_mode,
err_code, max_err);
appli_alarm := auto_mode AND NOT(STACKINT1.EMPTY);
err_alarm := STACKINT1.OFLO;
last_error := STACKINT1.OUT;
```

Results



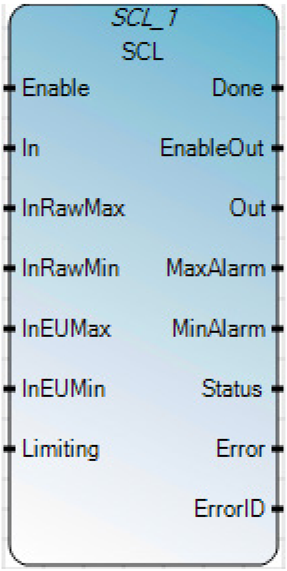
Name	Logical Value	Physical Value	Initial Value	Lock	Data Type
empty	□	N/A		□	BOOL
oflo	□	N/A		□	BOOL
out	5	N/A		□	DINT
pop	□	N/A		□	BOOL
push	☑	N/A		□	BOOL
PushedValue	5	N/A		□	DINT
reset	□	N/A		□	BOOL
size	10	N/A		□	DINT
+ STACKINT_1	...	...	...	□	STACKINT

SCL

Converts an unscaled REAL input value to a REAL floating point value in engineering units and includes alarming and limiting of the output.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description

Parameter	Parameter Type	Data Type	Description
Enable	Input	BOOL	TRUE - Rising Edge detected. • If InRawMin >= InRawMax, Status.0 and Status.1 bits are set to 1. Done is cleared, MaxAlarm and MinAlarm are cleared. Error is set to TRUE and ErrorID is set to 1. • Else Out is calculated first. Then alarm conditions are verified. If MaxAlarm is set, MinAlarm is cleared and vice versa. Then if limiting is set, Out will be in the range of InEUMin and InEUMax. Then Done bit is set to TRUE. Status bits are set to 0. Error and ErrorID are set to 0. • The calculated Out value is compared against NAN (Not a Number). If NAN is the Out value, EnableOut is cleared. Done is cleared if fault condition is set. FALSE - Rising Edge not detected. • The instruction does not execute. • The outputs are not updated except Error, ErrorID, EnableOut and Done are set to 0.

In	Input	REAL	The analog signal input.
			Valid = any float
			Default = 0.0

InRawMax	Input	REAL	The maximum value attainable by the input to the instruction. If InRawMax<= InRawMin, the instruction sets the appropriate bit in Status and Error. ErrorID are updated. Out updating stops.
			Valid = InRawMax > InRawMin
			Default =0.0

InRawMin	Input	REAL	The minimum value attainable by the input to the instruction. If InRawMin >= InRawMax, the instruction sets the appropriate bit in Status and Error. ErrorID are updated. Out updating stops.
			Valid = InRawMin < InRawMax
			Default = 0.0

InEUMax	Input	REAL	The scaled value of the input corresponding to InRawMax.
			Valid = any real value
			Default = 0.0

InEUMin	Input	REAL	The scaled value of the input corresponding to InRawMin.
			Valid = any real value
			Default = 0.0

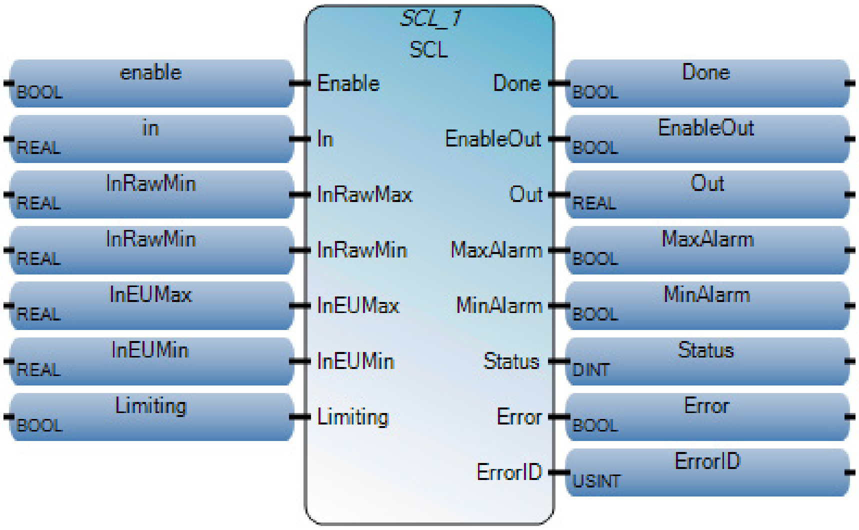
Limiting	Input	BOOL	Limiting selector. TRUE - Out is limited between InEUMin and InEUMax.
Done	Output	BOOL	Indicates when the operation is completed. TRUE - the operation completed successfully. FALSE - the operation encountered an error condition or Enable is set to FALSE.
EnableOut	Output	BOOL	Indicates if the instruction is enabled. Sets to False if Out overflows.
Out	Output	REAL	Represents the scaled value of the analog input.
MaxAlarm	Output	BOOL	The maximum input alarm indicator. This value is set to TRUE when Input > InRawMax.
MinAlarm	Output	BOOL	The minimum input alarm indicator. This value is set to TRUE when Input < InRawMin.
Status	Output	DINT	Status of the Function block. InstructFault (Status.0) It will be set when detected the instruction's execution errors. This is not the minor or major controller error. Check the remaining status bits to determine what has occurred. InRawRangeInv (Status.1) InRawMin >= InRawMax Status.3 to Status.31 are reserved for future usage and their value is 0.

Parameter	Parameter Type	Data Type	Description
Error	Output	BOOL	Indicates the existence of an error condition. TRUE - operation encountered an error. FALSE - operation completed successfully or the instruction is not executing.
ErrorID	Output	BOOL	A unique numeric that identifies the error. The errors are defined in the error codes.

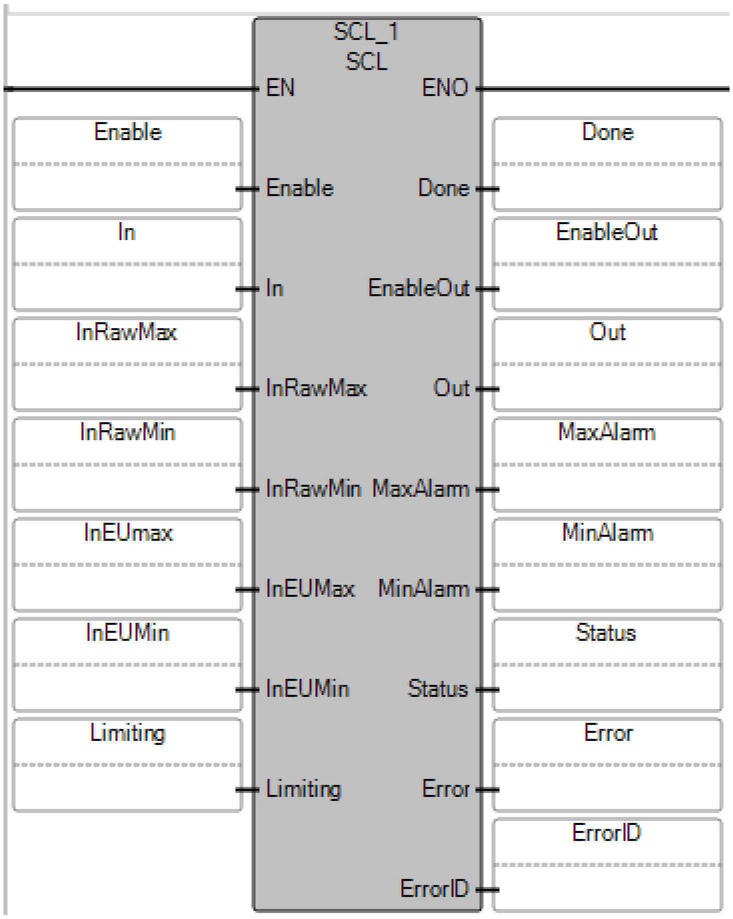
Error Code

ErrorID Code	Error description
1	InRawMax<= InRawMin

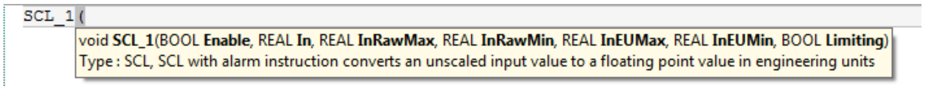
SCL Function Block Diagram example



SCL Ladder Diagram example

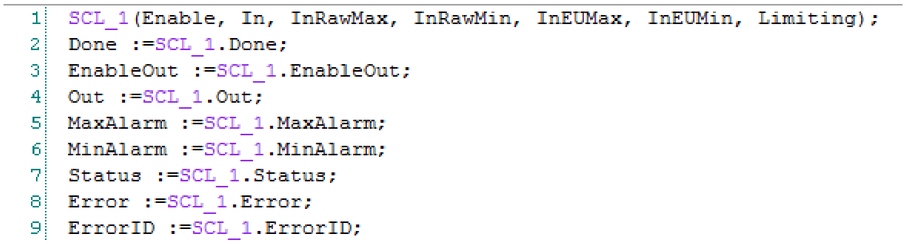


SCL Structured Text example



SCL_1(

void SCL_1(BOOL **Enable**, REAL **In**, REAL **InRawMax**, REAL **InRawMin**, REAL **InEUMax**, REAL **InEUMin**, BOOL **Limiting**)
Type : SCL, SCL with alarm instruction converts an unscaled input value to a floating point value in engineering units

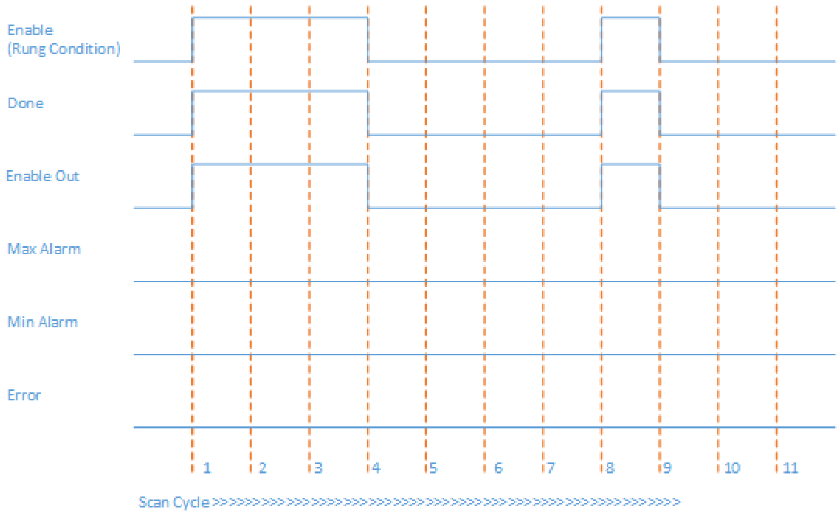


```
SCL_1(Enable, In, InRawMax, InRawMin, InEUMax, InEUMin, Limiting);
Done :=SCL_1.Done;
EnableOut :=SCL_1.EnableOut;
Out :=SCL_1.Out;
MaxAlarm :=SCL_1.MaxAlarm;
MinAlarm :=SCL_1.MinAlarm;
Status :=SCL_1.Status;
Error :=SCL_1.Error;
ErrorID :=SCL_1.ErrorID;
```

SCL instruction timing diagrams examples

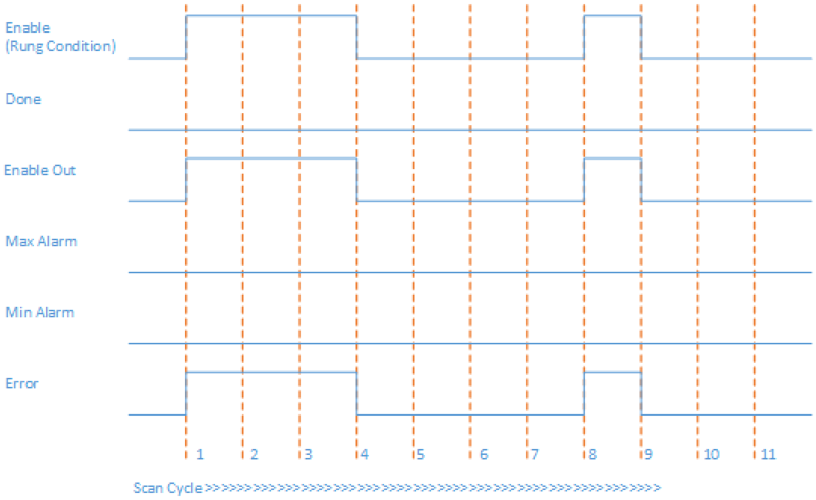
The following timing diagram examples describe execution scenarios for the SCL on page 497 instruction.

Successful SCL execution



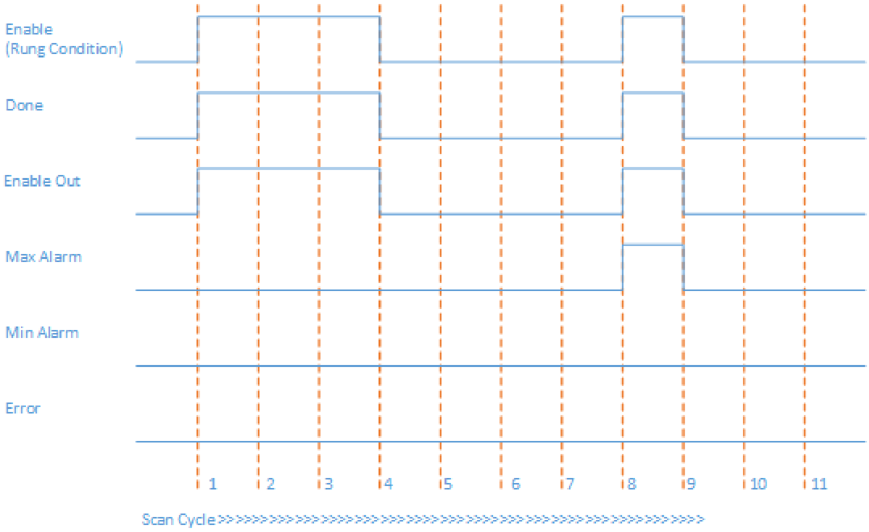
Scan Cycle	Description
1	When Enable is set to TRUE and input parameters are valid and within range, the Function Block execution starts. • Done and EnableOut are set to TRUE. • Out is calculated as per given inputs. • MaxAlarm, MinAlarm, and Error are set to FALSE. • Status ErrorID and are set to 0, because there is no error generated.
2, 3	No change in rung condition.
4	When Enable is set to FALSE, Function Block execution stops. • Done, EnableOut and Error are set to FALSE. • ErrorID is set to 0. • Out, MaxAlarm, MinAlarm and Status keep their last value.
5, 6, 7	No change in rung condition.
8	• When Enable is set to TRUE and input parameters are valid and within range, the Function Block execution starts. • Done and EnableOut are set to TRUE. • Out is calculated as per given inputs. • MaxAlarm, MinAlarm, and Error are set to FALSE. • ErrorID and Status are set to 0, because there is no error generated.
9	When Enable is set to FALSE, Function Block execution stops. • Done, EnableOut and Error are set to FALSE. • ErrorID is set to 0. • Out, MaxAlarm, MinAlarm and Status keep their last value.
10, 11	No change in rung condition.

Failed SCL execution



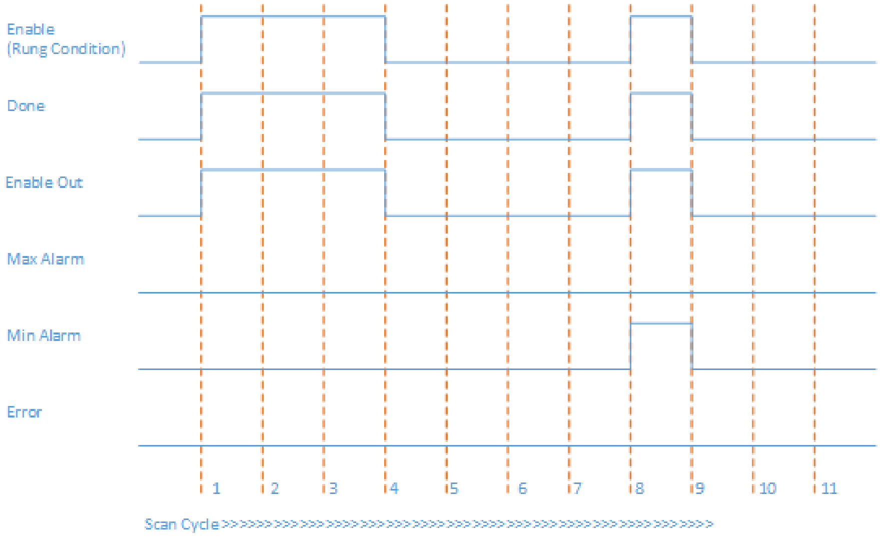
In this example, all the input parameters are valid and within range, but InRawMin >= InRawMax. In Scan Cycle 1 and 8, when Enable is set to TRUE and the Function Block execution starts, Error is set to True and ErrorID is set to 1.

Generation of MaxAlarm



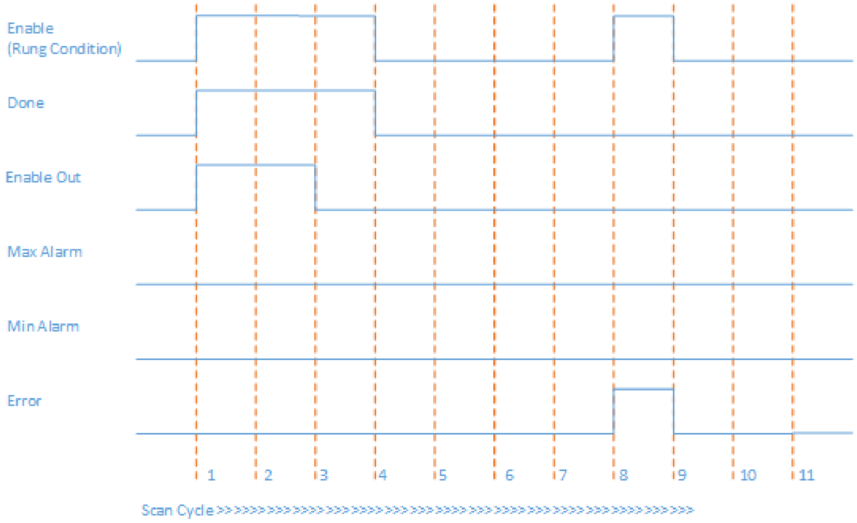
In Scan Cycle 8 of this example, all the input parameters are valid and within range, but In > InRawMax. When Enable is set to TRUE and the Function Block execution starts, MaxAlarm is set to TRUE.

Generation of MinAlarm



In Scan Cycle 8 of this example, all the input parameters are valid and within range, but In < InRawMin. When Enable is set to TRUE and the Function Block execution starts, MinAlarm is set to TRUE.

Output overflow condition and input configuration error



In Scan Cycle 3 of this example, Function Block input parameters are valid and within the range, but Out is overflowed because of input parameters. EnableOut is set to FALSE. Out value is invalid.

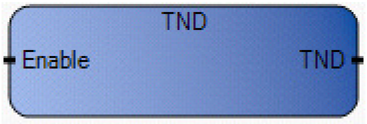
In Scan Cycle 8 of this example, Function Block input parameters are valid and within the range, but Out is overflowed because of input parameters and InRawMin >= InRawMax, Error is set to TRUE. ErrorID is set to 1 and Status is set to 3.

TND (stop current program)

Stops the current cycle of a user program scan. After the output scan, input scan, and housekeeping are complete, re-executes the user program from the start of the first routine.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

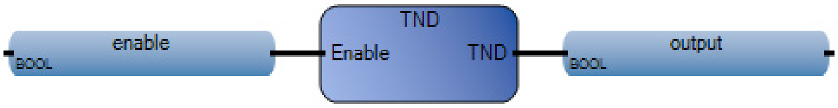
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



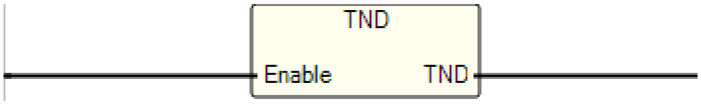
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
Enable	Input	BOOL	Function enable. When Enable = TRUE, perform the function. When Enable = FALSE, do not perform the function.
TND	Output	BOOL	If true, function performed. When variable monitoring is on, the monitoring variable's value is assigned to the instruction's output. When variable monitoring is off, the output variable's value is assigned to the instruction's output.

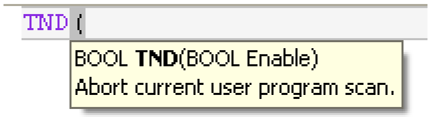
TND Function Block Diagram example



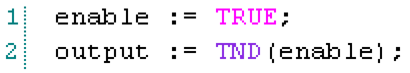
TND Ladder Diagram example



TND Structured Text example



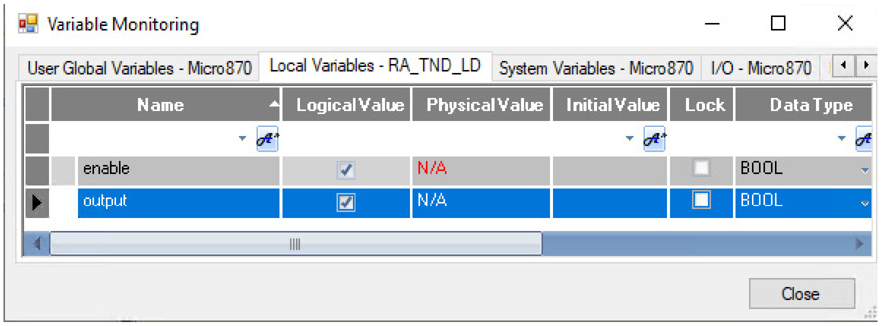
```
TND (
BOOL TND(BOOL Enable)
Abort current user program scan.
```



```
enable := TRUE;
output := TND(enable);
```

```
(* ST Equivalence: *)

TESTOUTPUT := TND(TESTENABLE) ;
```

Results


Variable Monitoring
User Global Variables - Micro870
Local Variables - RA_TND_LD
System Variables - Micro870
I/O - Micro870
Close

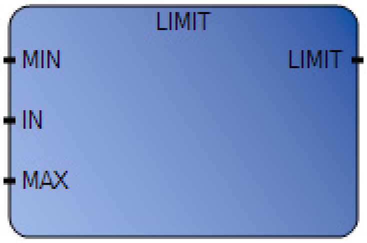
Name	Logical Value	Physical Value	Initial Value	Lock	Data Type
enable	☑	N/A		☐	BOOL
output	☑	N/A		☐	BOOL

LIMIT (limit test)

Restricts integer values to a given interval. Integer values between the minimum and maximum are unchanged. Integer values greater than the maximum are replaced with the maximum value. Integer values less than the minimum are replaced with the minimum value.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

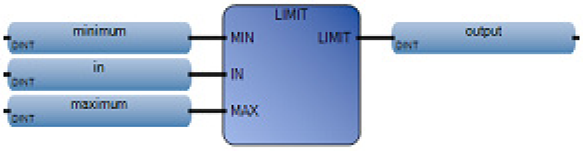
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



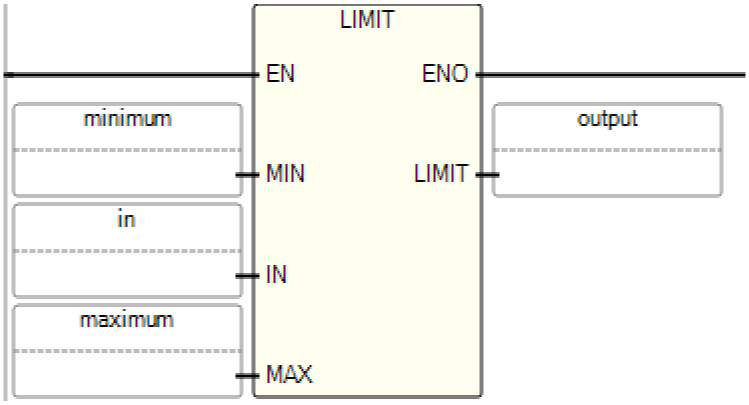
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Function enable. TRUE - execute current LIMIT computation. FALSE - there is no computation. Applies to Ladder Diagram programs.
MIN	Input	DINT	Minimum value supported.
IN	Input	DINT	Any signed integer value.
MAX	Input	DINT	Maximum value supported.
LIMIT	Output	DINT	Input value bounded to the supported range.
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

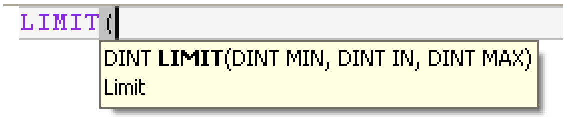
LIMIT Function Block Diagram example



LIMIT Ladder Diagram example

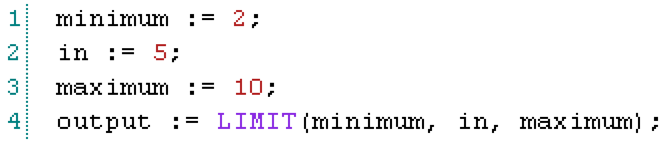


LIMIT Structured Text example



```
LIMIT(
```

DINT **LIMIT**(DINT MIN, DINT IN, DINT MAX)
Limit



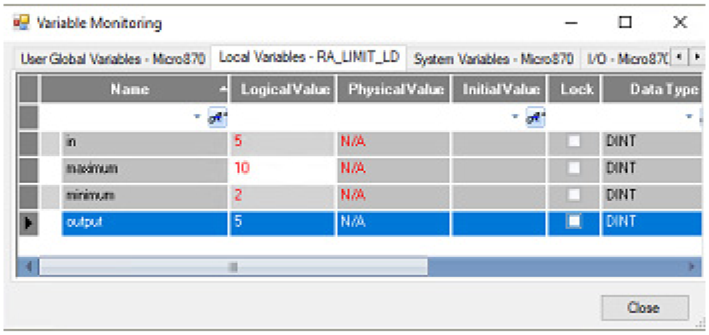
```
minimum := 2;
in := 5;
maximum := 10;
output := LIMIT(minimum, in, maximum);
```

(* ST Equivalence: *)

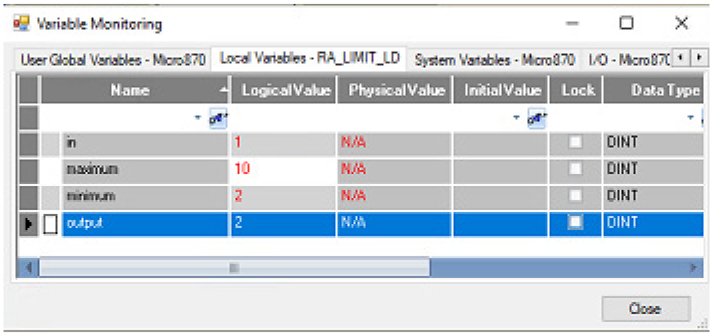
new_value := LIMIT (min_value, value, max_value);

(* bounds the value to the [min_value..max_value] set *)

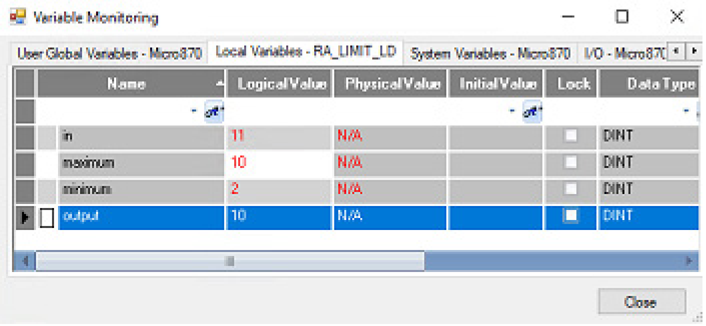
Results


Variable Monitoring
User Global Variables - Micro870
Local Variables - RA_LIMIT_LD
System Variables - Micro870
I/O - Micro870
Close

Name	LogicalValue	PhysicalValue	InitialValue	Lock	Data Type
in	5	N/A		□	DINT
maximum	10	N/A		□	DINT
minimum	2	N/A		□	DINT
output	5	N/A		□	DINT


Variable Monitoring
User Global Variables - Micro870
Local Variables - RA_LIMIT_LD
System Variables - Micro870
I/O - Micro870
Close

Name	LogicalValue	PhysicalValue	InitialValue	Lock	Data Type
in	1	N/A		□	DINT
maximum	10	N/A		□	DINT
minimum	2	N/A		□	DINT
output	2	N/A		□	DINT


Variable Monitoring
User Global Variables - Micro870
Local Variables - RA_LIMIT_LD
System Variables - Micro870
I/O - Micro870
Close

Name	LogicalValue	PhysicalValue	InitialValue	Lock	Data Type
in	11	N/A		□	DINT
maximum	10	N/A		□	DINT
minimum	2	N/A		□	DINT
output	10	N/A		□	DINT

Program control instruction

Use the program control instruction to control instructions simultaneously from a user program and from an operator interface device.

Instruction	Description
AFI on page 509	Disables a rung.
NOP on page 509	Functions as a placeholder.
SUS on page 509	Suspends the execution of the <M800 controller>.

AFI (Always False)

Use the AFI instruction at the beginning of a rung to temporarily disable the rung when debugging without having to delete the rung from the program. Output of this instruction is always FALSE.

Language supported: Ladder Diagram.

This instruction applies to the Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.

NOP (No Operation)

The NOP instruction functions as a placeholder. You can place the NOP instruction anywhere on a rung.

Language supported: Ladder Diagram.

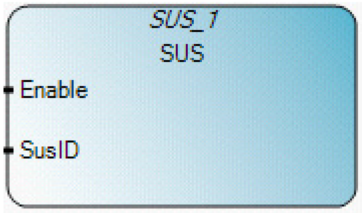
This instruction applies to the Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.

SUS (suspend)

Suspends the execution of the <M800 controller>. The controller remains in RUN mode but execution is suspended indefinitely. Suspend catches User Program errors and aids in User Program monitoring. Place the SUS instruction in User Program sections where you want to trap unusual conditions. In suspend mode, RUN LED is set to OFF to indicate the program scan is Idle.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

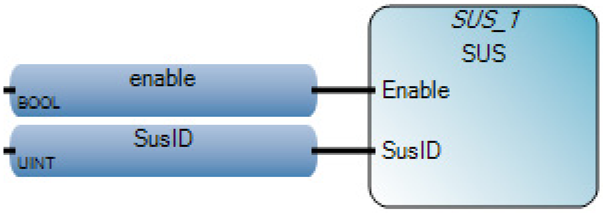
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



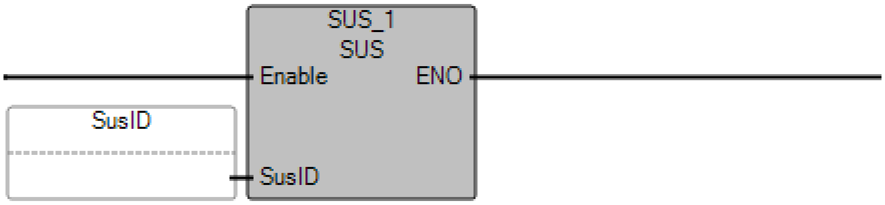
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter type	Data type	Description
Enable	Input	BOOL	Instruction block enable. TRUE - execute function. FALSE - do not execute function.
SusID	Input	UINT	Suspension ID.
ENO	Output	BOOL	Enable out. Applies only to Ladder Diagram programs.

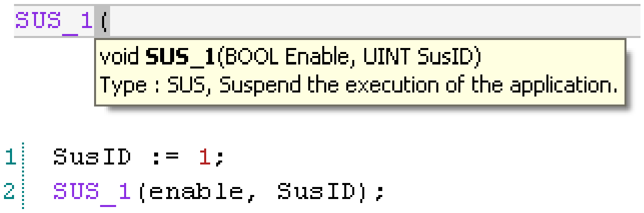
SUS Function Block Diagram example



SUS Ladder Diagram example



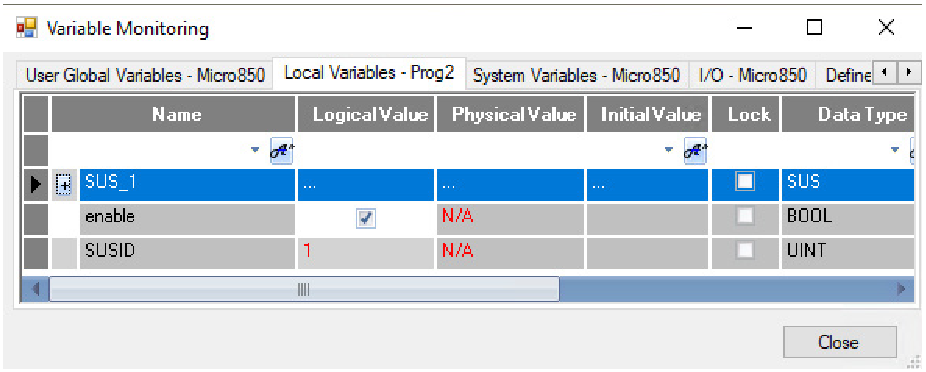
SUS Structured Text example



```
SUS_1(
void SUS_1(BOOL Enable, UINT SusID)
Type : SUS, Suspend the execution of the application.
```

```
SusID := 1;
SUS_1(enable, SusID);
```

Results



Name	Logical Value	Physical Value	Initial Value	Lock	Data Type
SUS_1	...	...	...	□	SUS
enable	☑	N/A		□	BOOL
SUSID	1	N/A		□	UINT

Proportional Integral Derivative (PID) instruction

Use the Proportional-Integral-Derivative (PID) instructions to control the process more accurately using PID functionality.

Instruction	Description
IPIDCONTROLLER on page 513	Configure and control the inputs and outputs used for the Proportional Integral Derivative (PID) logic.
PID on page 536	Configure and control the outputs that control physical properties such as temperature, pressure, liquid level, or flow rate using process loops.

What is Proportional Integral Derivative (PID) control?

Proportional-Integral-Derivative (PID) control allows the process control to accurately maintain the setpoint by adjusting the control outputs. A PID function block combines all of the necessary logic to perform proportional/integral/derivative (PID) control.

IPIDCONTROLLER (proportional-integral-derivative controller)

Configure and control the inputs and outputs used for proportional-integral-derivative (PID) logic. PID logic is used to control physical properties such as temperature, pressure, liquid, level, or flow rate by using process loops that calculate an error value as the difference between a desired setpoint and a measured process variable. The controller attempts to minimize the error over time by adjustment of a control variable. The calculation includes proportional (P), integral (I), and derivative(D) terms, which are used as follows:

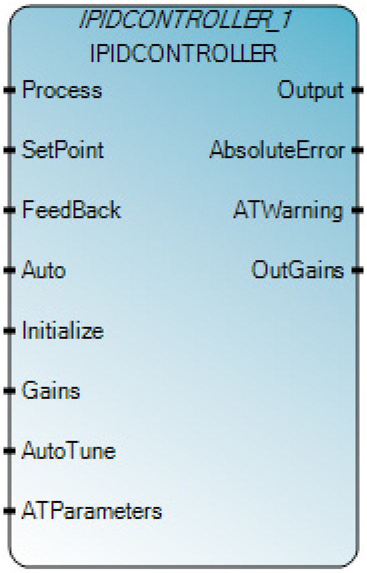
- P - present values of the error.

- I - past values of the error.

- D - possible future values of the error, based on its current rate of change. which controls physical properties such as temperature, pressure, liquid level, or flow rate using process loops.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.

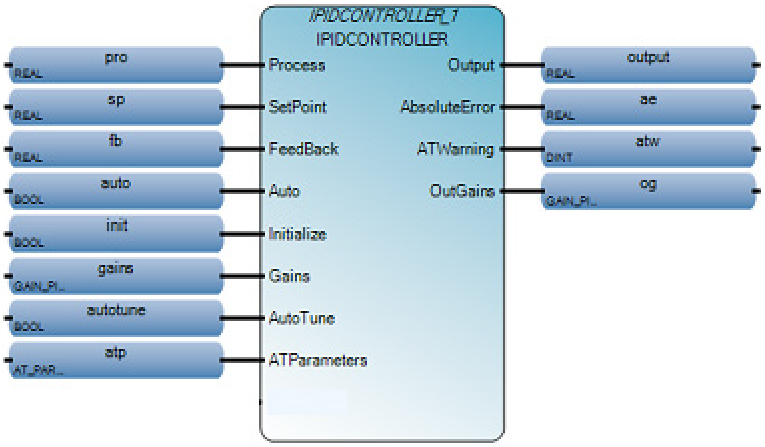


Use this table to help determine the parameter values for this instruction.

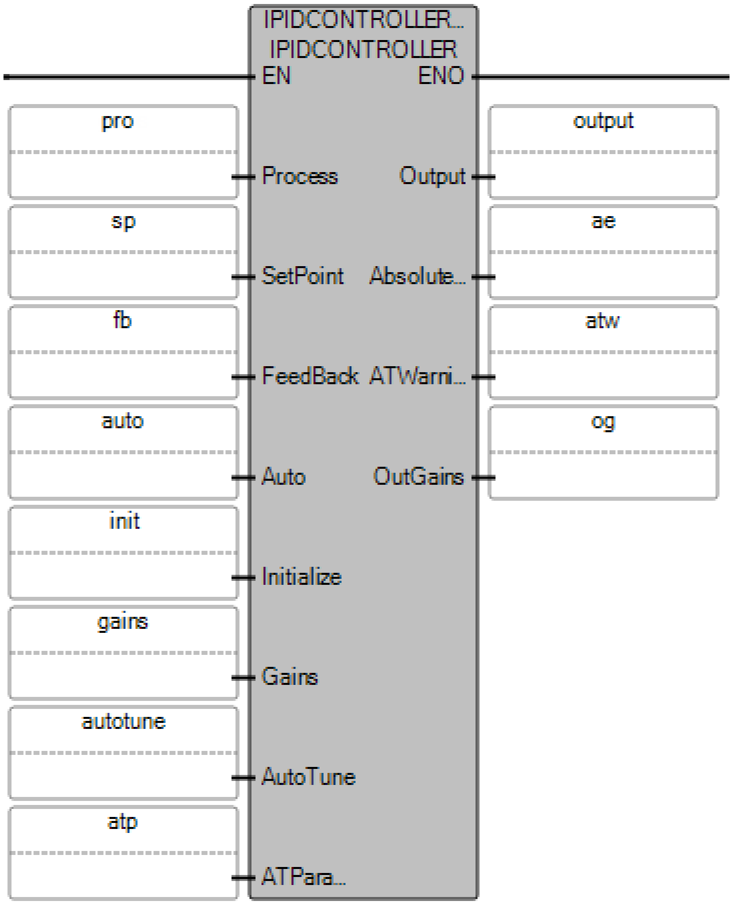
Parameter	Parameter type	Data type	Description
EN	Input	BOOL	When TRUE, enables the instruction block. TRUE - execute the PID calculation. FALSE - the instruction block is idle. Applies to Ladder Diagram programs.
Process	Input	REAL	Process value, which is the value measured from the process output.
SetPoint	Input	REAL	Set point.
FeedBack	Input	REAL	Feedback signal, which is the value of the control variable applied to the process. For example, the feedback can be IPIDCONTROLLER output.
Auto	Input	BOOL	The operation mode of the PID controller: ● TRUE - controller runs in normal mode. ● FALSE - controller causes reset R to track (F-GE).
Initialize	Input	BOOL	A change in value (TRUE to FALSE or FALSE to TRUE) causes the controller to eliminate any proportional gain during that cycle. Also initializes AutoTune sequences.
Gains	Input	GAIN_PID	Gains PID for IPIDController. Use the GAIN_PID data type on page 517 to define the parameters for the Gains input.
AutoTune	Input	BOOL	TRUE - When AutoTune is TRUE and Auto and Initialize are FALSE, the AutoTune sequence is started. FALSE - Do not not start Autotune.
ATParameters	Input	AT_Param	Auto Tune Parameters. Use the AT_Param data type on page 518 to define the parameters for the ATParameters input.
Output	Output	REAL	Output value from the controller.
AbsoluteError	Output	REAL	Absolute error (Process – SetPoint) from the controller.

Parameter	Parameter type	Data type	Description
ATWarnings	Output	DINT	(ATWarning) Warning for the Auto Tune sequence. Possible values are: • 0 - no auto tune done. • 1 - in auto tune mode. • 2 - auto tune done. • -1 - ERROR 1 input automatically set to TRUE, no auto tune possible. • -2 - ERROR 2 auto tune error, ATDynaSet expired.
OutGains	Output	GAIN_PID	Gains calculated after AutoTune sequences. Use the GAIN_PID data type to define the OutGains output.
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

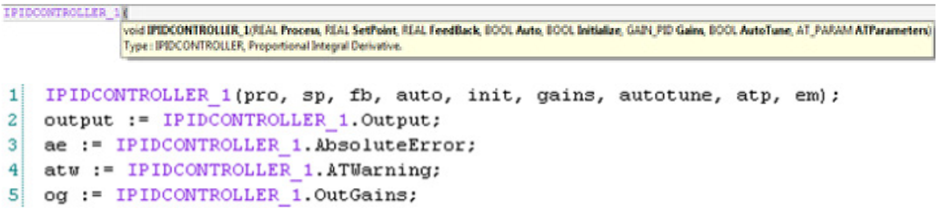
IPIDCONTROLLER Function Block Diagram examples



IPIDCONTROLLER Ladder Diagram example



IPIDCONTROLLER Structured Text example



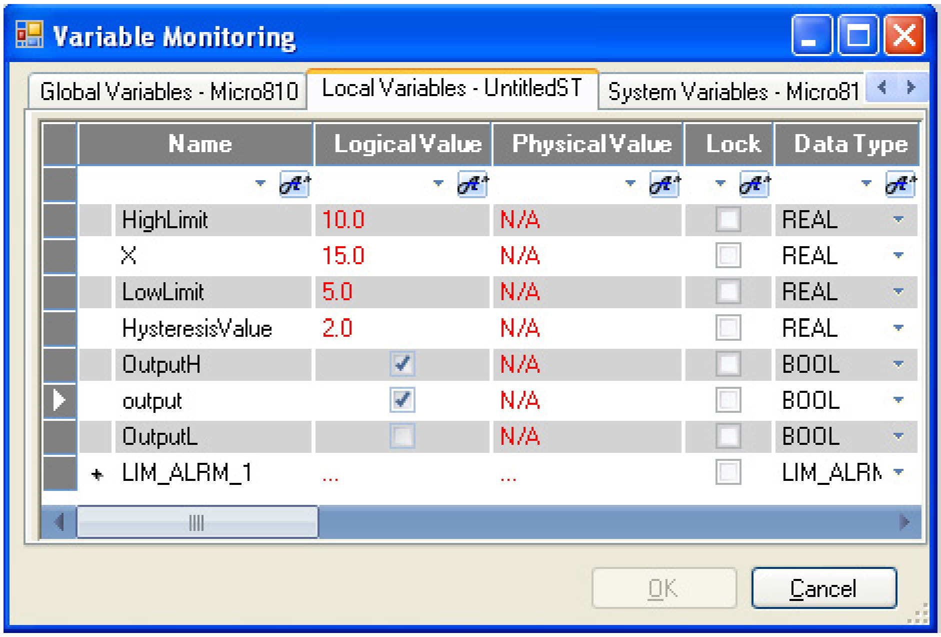
IPIDCONTROLLER_1(

void IPIDCONTROLLER_1(REAL Process, REAL SetPoint, REAL FeedBack, BOOL Auto, BOOL Initialize, GAIN_PID Gains, BOOL AutoTune, AT_PARAM ATParameters)
Type : IPIDCONTROLLER, Proportional Integral Derivative.

```
IPIDCONTROLLER_1(pro, sp, fb, auto, init, gains, autotune, atp, em);
output := IPIDCONTROLLER_1.Output;
ae := IPIDCONTROLLER_1.AbsoluteError;
atw := IPIDCONTROLLER_1.ATWarning;
og := IPIDCONTROLLER_1.OutGains;
```

```
   (* ST equivalence: IPIDController1 is an instance of
   IPIDController block *)

   IPIDController1(Proc,
                    SP,
                    FBK,
                    Auto,
                    Init,
                    G_In,
                    A_Tune,
                    A_TunePar,
                    Err );
   Out_process := IPIDController1.Output ;
   A_Tune_Warn := IPIDController1.ATWarning ;
   Gain_Out := IPIDController1.OutGains ;
```

Results


Variable Monitoring
Global Variables - Micro810
Local Variables - UntitledST
System Variables - Micro81
OK
Cancel

Name	Logical Value	Physical Value	Lock	Data Type
HighLimit	10.0	N/A	□	REAL
X	15.0	N/A	□	REAL
LowLimit	5.0	N/A	□	REAL
HysteresisValue	2.0	N/A	□	REAL
OutputH	☑	N/A	□	BOOL
output	☑	N/A	□	BOOL
OutputL	□	N/A	□	BOOL
+ LIM_ALRM_1	...	...	□	LIM_ALRM

GAIN_PID data type

The following table describes the GAIN_PID data type for the IPIDCONTROLLER on page 513 instruction.

Parameter	Data type	Description
DirectActing	BOOL	The type of acting: • TRUE - direct acting, output moves same direction as error. That is, the actual process value is greater than the SetPoint and the appropriate controller action is to increase the output For example: Chilling. • FALSE - reverse acting, output moves opposite direction as error. That is, the actual process value is greater than the SetPoint and the appropriate controller action is to decrease the output For example: Heating.
ProportionalGain	REAL	Proportional gain for PID (>= 0.0001). When ProportionalGain is (< 0.0001) then ProportionalGain = 0.0001 **Proportional gain for PID (P_Gain)** A higher proportional gain causes a larger change in the output based upon the difference between the PV (measured process value) and SV (set point value). The higher the gain, the faster the error is decreased, but this may result in instability such as oscillations. The lower the gain, the slower the error is decreased, but the system is more stable and less sensitive to large errors. The P_Gain usually is the most important gain to adjust and the first gain to adjust while tuning.
TimeIntegral	REAL	Time integral value for PID in seconds (>= 0.0001). When TimeIntegral is (< 0.0001) then TimeIntegral = 0.0001 **Time integral value for PID** A smaller integral time constant causes a faster change in the output based upon the difference between the PV (measured process value) and SV (set point value) integrated over this time. A smaller integral time constant decreases the steady state error (error when SV is not being changed) but increases the chances of instability such as oscillations. A larger integral time constant slows down the response of the system and make it more stable, but PV approaches the SV at a slower rate.

Parameter	Data type	Description
TimeDerivative	REAL	Time derivative value for PID in seconds (> 0.0). When TimeDerivative is (<= 0.0) then TimeDerivative = 0.0 When TimeDerivative = 0, IPID acts as PI. **Time derivative value for PID (Td)** A smaller derivative time constant causes a faster change in the output based upon the rate of change of the difference between PV (measured process value) and SV (set point value). A smaller derivative time constant makes a system more responsive to sudden changes in error (SV is changed) but increases the chances of instability such as oscillations. A larger time constant makes a system less responsive to sudden changes in error and the system is less susceptible to noise and step changes in PV. TimeDerivative (Td) is related to the derivative gain but allows the derivative contribution to PID to be tuned using time so the sample time must be taken into consideration.
DerivativeGain	REAL	Derivative gain for PID (> 0.0). When DerivativeGain is (< 0.0) then DerivativeGain = 0.1 **Derivative gain for PID (D_Gain)** A higher derivative gain causes a larger change in the output based upon the rate of change of the difference between the PV (measured process value) and SV (set point value). A higher gain makes a system more responsive to sudden changes in error but increases the chances of instability such as oscillations. A lower gain makes a system less responsive to sudden changes in error and makes the system less susceptible to noise and step changes in the PV.

AT_Param data type

The following table describes the AT_Param data type parameters.

Parameter	Data type	Description
Load	REAL	Load parameter for auto tuning. This is the output value when starting AutoTune.
Deviation	REAL	Deviation for auto tuning. This is the standard deviation used to evaluate the noise band needed for AutoTune.
Step	REAL	Step value for AutoTune. Must be greater than noise band and less than ½ Load.
ATDynamSet	REAL	Waiting time in seconds before abandoning auto tune.
ATReset	BOOL	The indication of whether the output value is reset to zero after an AutoTune sequence: • TRUE - resets output to zero. • FALSE - leaves output at Load value.

How the IPIDController function block implements PID control

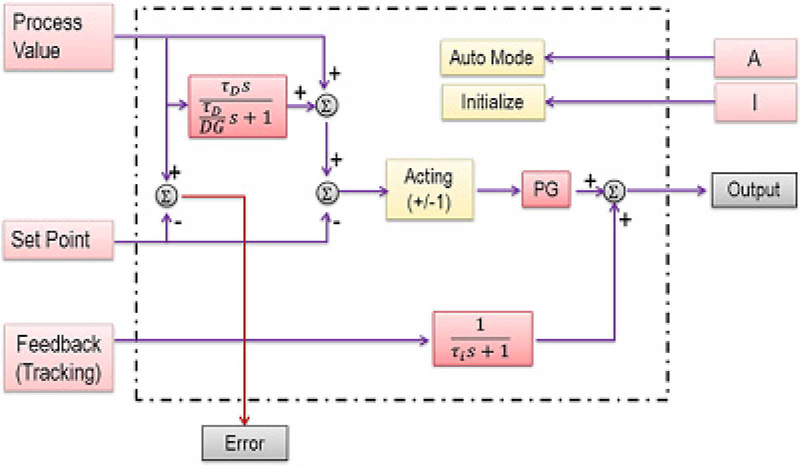
The IPIDController function block, available in the Connected Components Workbench instruction set, is based on PID control theory and combines all of the necessary logic to perform analog input channel processing and proportional integral-derivative (PID) control. In the HMI, the IPID faceplate is available for use with the IPIDController function block.

IPIDController function block description

The IPIDController function block uses the following function block components:

- A: Acting (+/- 1)
- PG: Proportional Gain
- DG: Derivative Filter Gain

- td: ãD
- ti: ãI



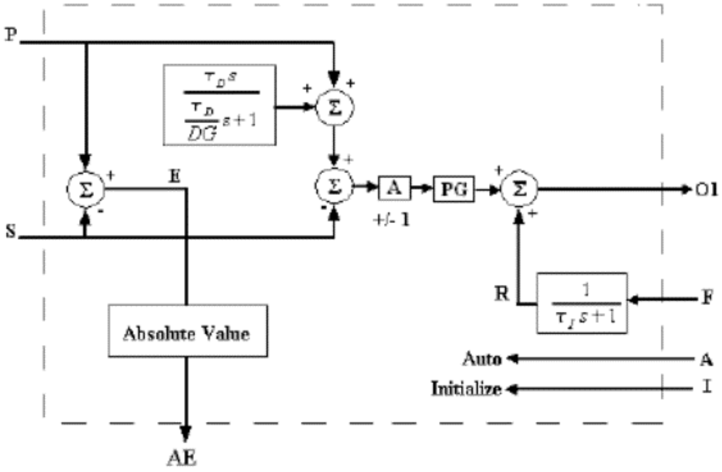
Preventing integral windup

If the difference between the setpoint value and the process value is great, the output value increases significantly, and during the time it takes to decrease, the process is not in control. The IPIDController function block interactively tracks feedback and prevents integral windup. When the output is saturated, the integral term in the controller is recomputed so that its new value provides an output at the saturation limit.

IPIDController function block operation

When **Input Auto** is **TRUE**, the IPIDController runs in normal auto mode.

When **Input Auto** is **FALSE**, it causes reset R to track (F-GE) forcing the IPIDController Output to track the Feedback within the IPIDController limits at which time the controller switches back to auto without incrementing the Output.



For Input Initialize, changing from FALSE to TRUE or TRUE to FALSE when AutoTune is FALSE causes the IPIDController to eliminate any proportional gain action during that cycle (for example, Initialize). Use this process to prevent bumping the Output when changes are made to the SetPoint using a switch function block.

To run an AutoTune sequence:

To run an AutoTune sequence, the input ATParameters must be completed. The Input Gain and DirectActing parameters must be set according to the process and DerivativeGain set, (typically 0.1). The AutoTune sequence is started with the following sequence:

1. Set the input Initialize to TRUE.
2. Set the input Autotune to TRUE.
3. Change the input Initialize to FALSE.
4. Wait until the output ATWarning changes to 2.
5. Transfer the values for output OutGains to input Gains.

To finalize the tuning, some fine tuning may be needed depending on the processes and needs. When setting TimeDerivative to 0.0, the IPIDController forces DerivativeGain to 1.0 then works as a PI controller.

Use the Proportional Integral Derivative instruction

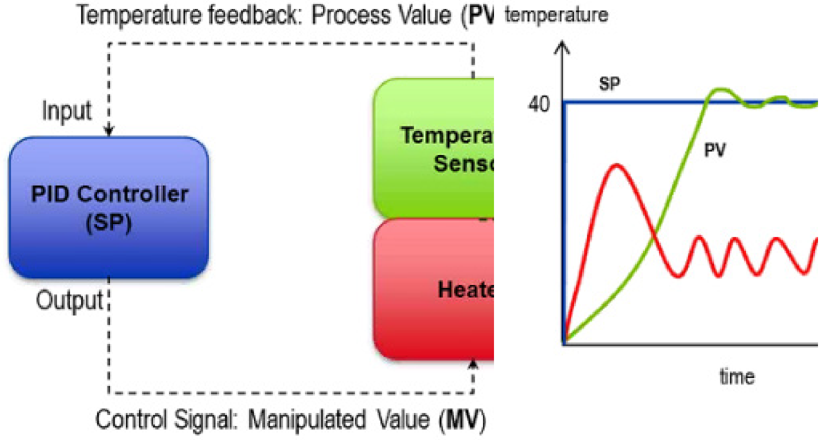
This section provides specific details and examples for using the proportional integral derivative instruction, including the following:

Example: How to create a feedback loop for the manipulated value

Adding a feedback loop for the manipulated value prevents excessive overshooting by providing a minimum and maximum value for the MV.

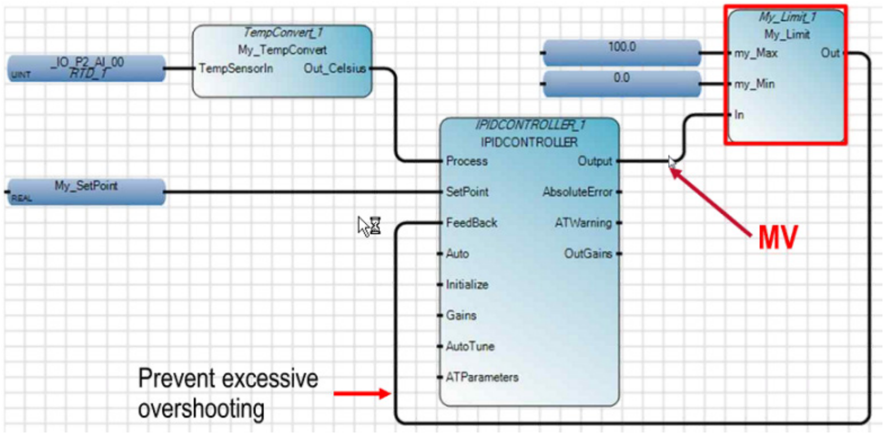
Temperature feedback loop example

At the beginning of the temperature control process, the difference between the process value (PV) and the setpoint value (SP) is large, as shown in the following graph. In this example of a temperature feedback loop, the PV starts at 0 degrees Celsius and moves towards the SP value of 40 degrees Celsius. Notice also that the fluctuation between the high and low manipulated value (MV) decreases and stabilizes with time. The behavior of the MV depends on the values used in each of the P, I, and D parameters.


Temperature feedback: Process Value (PV
Input
PID Controller (SP)
Output
Tempera Senso
Heate
Control Signal: Manipulated Value (MV)
temperature
40
SP
PV
time

IPIDController with a feedback loop

The following function block diagram includes a feedback loop for the manipulated value that prevents excessive overshooting by providing a minimum and maximum value for the MV.



Example: How to implement auto-tuning in a IPIDController function block

Use AutoTune parameter of the IPIDController function block to implement auto-tuning in the control program.

Auto-tuning requirements and recommendations

Following is a summary of requirements and recommendations for implementing successful auto-tuning.

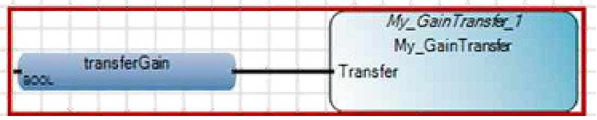
- Autotuning must cause the output of the control loop to oscillate, which means the IPIDController must be called frequently enough to adequately sample the oscillation.
- The IPIDController instruction block must be executed at a relatively constant time interval.
- Configure the scan time of the program to be than half of the oscillation period.
- Consider using a Structured Text Interrupt (STI) instruction block to control the IPIDController instruction block.

Example: How to add a UDFB to a PID program

Add UDFBs outside the main program to perform specialized functions such as converting units or transferring values.

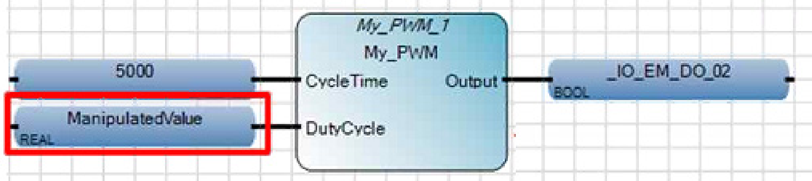
Transfer the auto-tune gain value

This UDFB transfers the Autotune gain value to My_GainTransfer so it can be used by the controller.



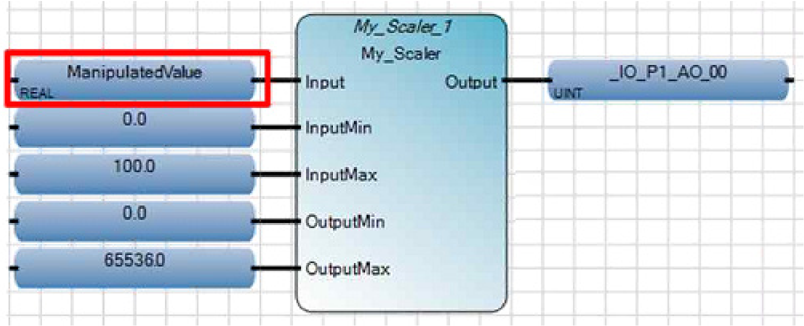
Convert a manipulated value to a digital output

This UDFB converts a manipulated value (MV) to a digital output (DO) so it can be used to control a digital input n(DI).



Convert a manipulated value to an analog output

This UDFB converts a manipulated value (MV) to an analog output (AO) so it can be used to control an analog input (AI).



Use auto-tune with the IPIDController function block

Use the AutoTune parameter of the IPIDController function block to implement auto-tuning in the control program.

Auto-tuning requirements and recommendations

Following is a summary of requirements and recommendations for implementing successful auto-tuning.

- Autotuning must cause the output of the control loop to oscillate, which means the IPIDController must be called frequently enough to adequately sample the oscillation.
- The IPIDController instruction block must be executed at a relatively constant time interval.
- Configure the scan time of the program to be than half of the oscillation period.
- Consider using a Selectable Timed Interrupt (STI) instruction block to control the IPIDController instruction block.

Auto-tune in first and second order systems

Use auto-tune in first order system, which uses a single element, or in a second order system, which uses two independent elements.

A first order system uses a single independent energy storage element. Examples include:

- Cooling of a fluid tank, with heat energy as the storage unit.
- Flow of fluid from a tank, with potential energy as the storage unit.
- A motor with constant torque driving a disk flywheel, with rotational kinetic energy as the storage unit.
- An electric RC lead network, with capacitive storage energy as the storage unit.

In a first order system, the function may be written in a standard form such as $f(t) = \tau\ dy/dt + y(t)$

Where:

Variable	Description	Example: Cooling of a fluid tank using heat energy as the storage element
t	System time constant	Is equal to RC Where R = Thermal resistance of the walls of the tank C = Thermal capacitance of the fluid
f	Forcing function	Is the Ambient temperature
y	System state variable	Is the Fluid temperature

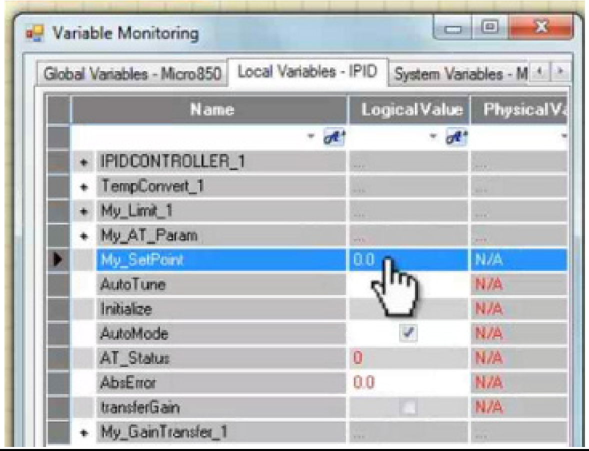
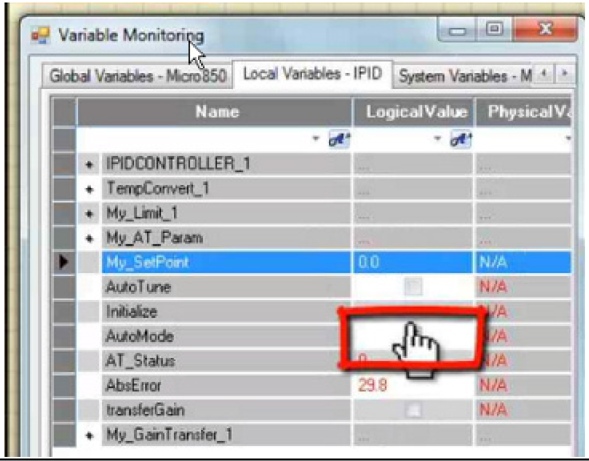
A second order system uses two independent energy storage elements that exchange stored energy. Examples include:

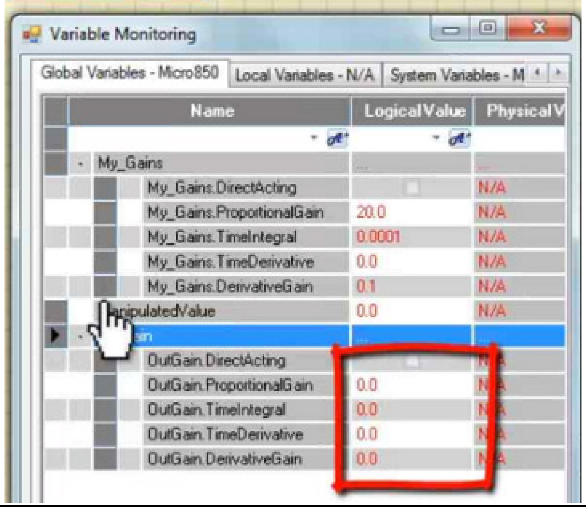
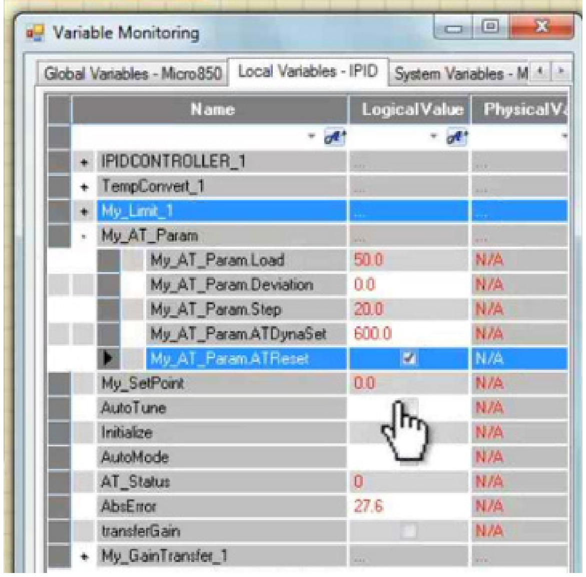
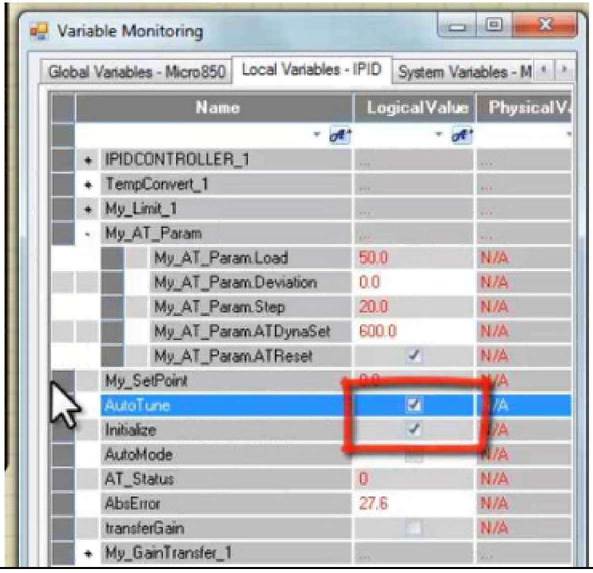
- A motor driving a disk flywheel with the motor coupled to the flywheel via a shaft with torsional stiffness; Rotational kinetic energy and torsion spring energy are the storage units.
- An electric circuit composed of a current source driving a series LR (inductor and resistor) with a shunt C (capacitor); Inductive energy and capacitive energy are the storage units.

Motor driven systems and heating systems can typically be modeled by the LR and the C electric circuit.

Configure auto-tuning

Use these general steps when implementing auto-tuning using the IPIDController function on page 523.

No.	Step	Example
1	Reset setpoint to zero.	 Variable Monitoring Global Variables - Micro850 | Local Variables - IPID | System Variables - M Name | LogicalValue | PhysicalV IPIDCONTROLLER_1 | ... | ... TempConvert_1 | ... | ... My_Limit_1 | ... | ... My_AT_Param | ... | ... My_SetPoint | 0.0 | N/A AutoTune | | N/A Initialize | | N/A AutoMode | ☑ | N/A AT_Status | 0 | N/A AbsError | 0.0 | N/A transferGain | □ | N/A My_GainTransfer_1 | ... | ...
2	Switch Auto mode to False	 Variable Monitoring Global Variables - Micro850 | Local Variables - IPID | System Variables - M Name | LogicalValue | PhysicalV IPIDCONTROLLER_1 | ... | ... TempConvert_1 | ... | ... My_Limit_1 | ... | ... My_AT_Param | ... | ... My_SetPoint | 0.0 | N/A AutoTune | □ | N/A Initialize | | N/A AutoMode | | N/A AT_Status | 0 | N/A AbsError | 29.8 | N/A transferGain | □ | N/A My_GainTransfer_1 | ... | ...

No.	Step	Example
3	Set Gains parameters.	
4	Set Auto-Tune parameters.	Set auto-tune parameters including an initial load value, step change for the output, an estimated time to complete the auto tuning, and the auto-tune reset. 
5	Set Initialize and AutoTune to True.	

Variable Monitoring

Global Variables - Micro850 | Local Variables - N/A | System Variables - M

Name	Logical Value	Physical V
My_Gains	...	...
My_Gains.DirectActing	☐	N/A
My_Gains.ProportionalGain	20.0	N/A
My_Gains.TimeIntegral	0.0001	N/A
My_Gains.TimeDerivative	0.0	N/A
My_Gains.DerivativeGain	0.1	N/A
nipulatedValue	0.0	N/A
ain	...	...
OutGain.DirectActing	☐	N/A
OutGain.ProportionalGain	0.0	N/A
OutGain.TimeIntegral	0.0	N/A
OutGain.TimeDerivative	0.0	N/A
OutGain.DerivativeGain	0.0	N/A

Variable Monitoring

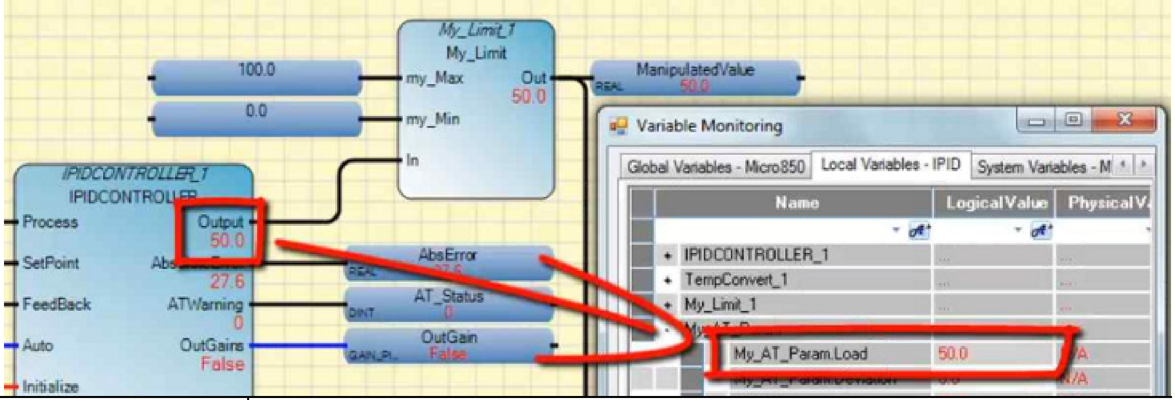
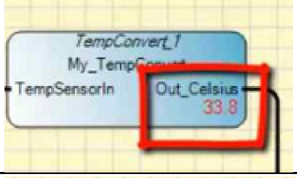
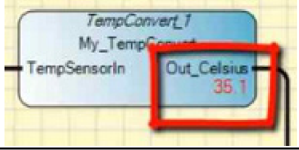
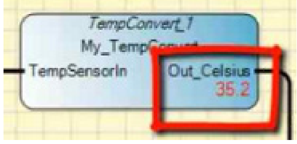
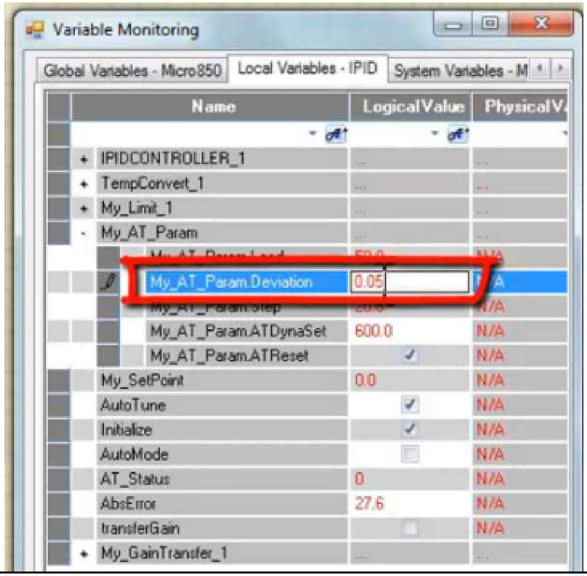
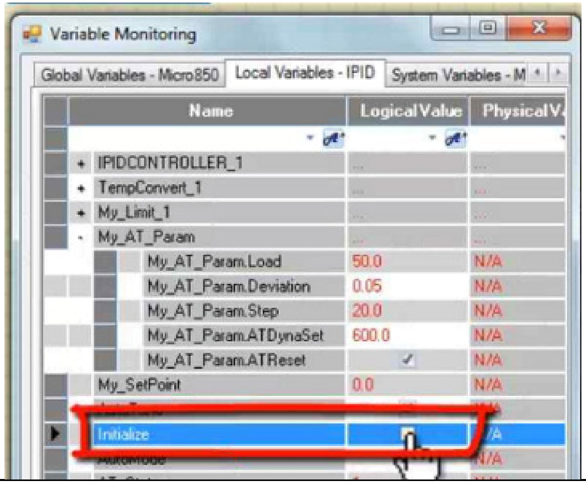
Global Variables - Micro850 | Local Variables - IPID | System Variables - M

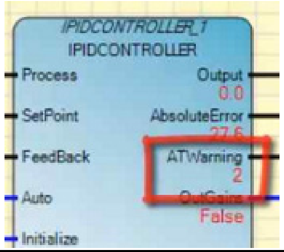
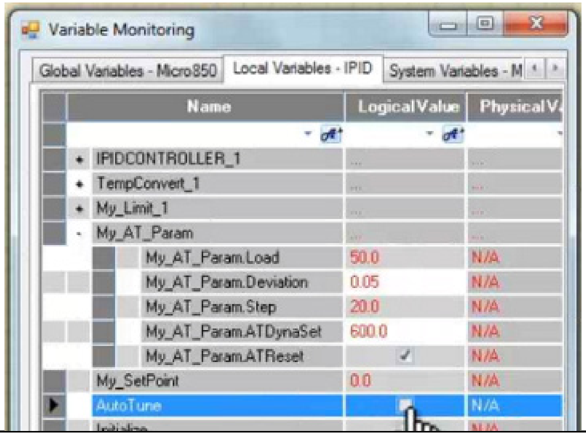
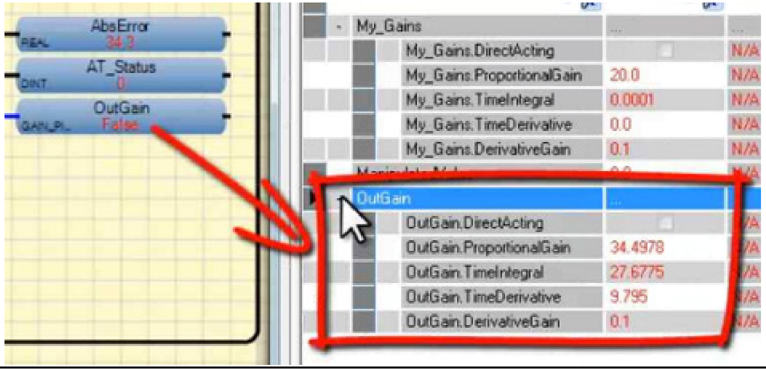
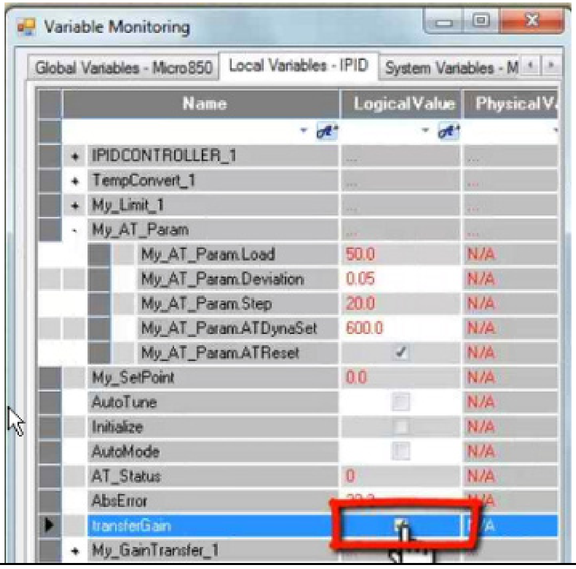
Name	Logical Value	Physical V
IPIDCONTROLLER_1	...	...
TempConvert_1	...	...
My_Limit_1	...	...
My_AT_Param	...	...
My_AT_Param.Load	50.0	N/A
My_AT_Param.Deviation	0.0	N/A
My_AT_Param.Step	20.0	N/A
My_AT_Param.ATDynaSet	600.0	N/A
My_AT_Param.ATReset	☑	N/A
My_SetPoint	0.0	N/A
AutoTune		N/A
Initialize		N/A
AutoMode		N/A
AT_Status	0	N/A
AbsError	27.6	N/A
transferGain	☐	N/A
My_GainTransfer_1	...	...

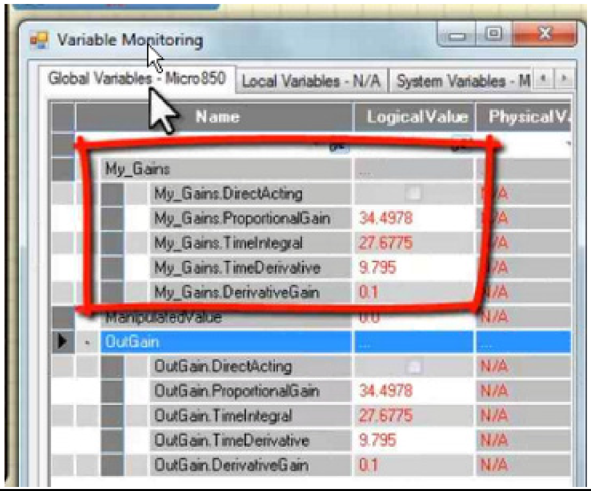
Variable Monitoring

Global Variables - Micro850 | Local Variables - IPID | System Variables - M

Name	Logical Value	Physical V
IPIDCONTROLLER_1	...	...
TempConvert_1	...	...
My_Limit_1	...	...
My_AT_Param	...	...
My_AT_Param.Load	50.0	N/A
My_AT_Param.Deviation	0.0	N/A
My_AT_Param.Step	20.0	N/A
My_AT_Param.ATDynaSet	600.0	N/A
My_AT_Param.ATReset	✓	N/A
My_SetPoint	0.0	N/A
AutoTune	☑	N/A
Initialize	✓	N/A
AutoMode	☐	N/A
AT_Status	0	N/A
AbsError	27.6	N/A
transferGain	☐	N/A
My_GainTransfer_1	...	...

No.	Step	Example
6	Notice the Output changes to the value of Load when you set AutoTune to True.	 My_Limit_1 My_Limit 100.0 0.0 my_Max my_Min In Out 50.0 ManipulatedValue REAL 50.0 IPIDCONTROLLER_1 IPIDCONTROLLER Process SetPoint FeedBack Auto Initialize Output 50.0 AbsError 27.6 ATWarning 0 OutGains False AbsError REAL AT_Status DINT 0 OutGain GAIN_PI FalseVariable Monitoring Global Variables - Micro850 | Local Variables - IPID | System Variables - M Name | Logical Value | Physical V + IPIDCONTROLLER_1 | ... | ... + TempConvert_1 | ... | ... + My_Limit_1 | ... | ... My_AT_Param.Load | 50.0 | N/A My_AT_Param.Deviation | 0.0 | N/A
7	Observe the process value rises quickly until it gets closer to its saturation point.	 TempConvert_1 My_TempConvert TempSensorIn Out_Celsius 33.8
8	Observe the stabilization of the process value and its fluctuation.	 TempConvert_1 My_TempConvert TempSensorIn Out_Celsius 35.1 TempConvert_1 My_TempConvert TempSensorIn Out_Celsius 35.2
9	Set the deviation.	Variable Monitoring Global Variables - Micro850 | Local Variables - IPID | System Variables - M Name | Logical Value | Physical V + IPIDCONTROLLER_1 | ... | ... + TempConvert_1 | ... | ... + My_Limit_1 | ... | ... - My_AT_Param | ... | ... My_AT_Param.Load | 50.0 | N/A My_AT_Param.Deviation | 0.05 | N/A My_AT_Param.Step | 20.0 | N/A My_AT_Param.ATDynaSet | 600.0 | N/A My_AT_Param.ATReset | ✓ | N/A My_SetPoint | 0.0 | N/A AutoTune | ✓ | N/A Initialize | ✓ | N/A AutoMode | □ | N/A AT_Status | 0 | N/A AbsError | 27.6 | N/A transferGain | □ | N/A + My_GainTransfer_1 | ... | ...
10	Set Initialize to False.	Variable Monitoring Global Variables - Micro850 | Local Variables - IPID | System Variables - M Name | Logical Value | Physical V + IPIDCONTROLLER_1 | ... | ... + TempConvert_1 | ... | ... + My_Limit_1 | ... | ... - My_AT_Param | ... | ... My_AT_Param.Load | 50.0 | N/A My_AT_Param.Deviation | 0.05 | N/A My_AT_Param.Step | 20.0 | N/A My_AT_Param.ATDynaSet | 600.0 | N/A My_AT_Param.ATReset | ✓ | N/A My_SetPoint | 0.0 | N/A Initialize | □ | N/A AutoMode | | N/A

No.	Step	Example
11	Controller starts auto-tuning. Wait for ATWarning to become 2.	 IPIDCONTROLLER_1 IPIDCONTROLLER Process — Output 0.0 SetPoint — AbsoluteError 27.6 FeedBack — ATWarning 2 Auto — OutGains False Initialize
12	Set AutoTune to False.	 Variable Monitoring Global Variables - Micro850 | Local Variables - IPID | System Variables - M Name | Logical Value | Physical V + IPIDCONTROLLER_1 | ... | ... + TempConvert_1 | ... | ... + My_Limit_1 | ... | ... - My_AT_Param | ... | ... My_AT_Param.Load | 50.0 | N/A My_AT_Param.Deviation | 0.05 | N/A My_AT_Param.Step | 20.0 | N/A My_AT_Param.ATDynaSet | 600.0 | N/A My_AT_Param.ATReset | ✓ | N/A My_SetPoint | 0.0 | N/A AutoTune | □ | N/A Initialize
13	Observe the tuned values appear in OutGains.	 AbsError REAL 34.3 AT_Status DINT 0 OutGain GAIN_PI... False - My_Gains | ... | ... My_Gains.DirectActing | □ | N/A My_Gains.ProportionalGain | 20.0 | N/A My_Gains.TimeIntegral | 0.0001 | N/A My_Gains.TimeDerivative | 0.0 | N/A My_Gains.DerivativeGain | 0.1 | N/A OutGain | ... OutGain.DirectActing | □ OutGain.ProportionalGain | 34.4978 OutGain.TimeIntegral | 27.6775 OutGain.TimeDerivative | 9.795 OutGain.DerivativeGain | 0.1
14	Transfer parameter from OutGain to My_Gains.	 Variable Monitoring Global Variables - Micro850 | Local Variables - IPID | System Variables - M Name | Logical Value | Physical V + IPIDCONTROLLER_1 | ... | ... + TempConvert_1 | ... | ... + My_Limit_1 | ... | ... - My_AT_Param | ... | ... My_AT_Param.Load | 50.0 | N/A My_AT_Param.Deviation | 0.05 | N/A My_AT_Param.Step | 20.0 | N/A My_AT_Param.ATDynaSet | 600.0 | N/A My_AT_Param.ATReset | ✓ | N/A My_SetPoint | 0.0 | N/A AutoTune | □ | N/A Initialize | □ | N/A AutoMode | □ | N/A AT_Status | 0 | N/A AbsError | | N/A transferGain | ☑ | + My_GainTransfer_1

No.	Step	Example
15	Observe the controller is updated with the with the tuned gain parameter.	

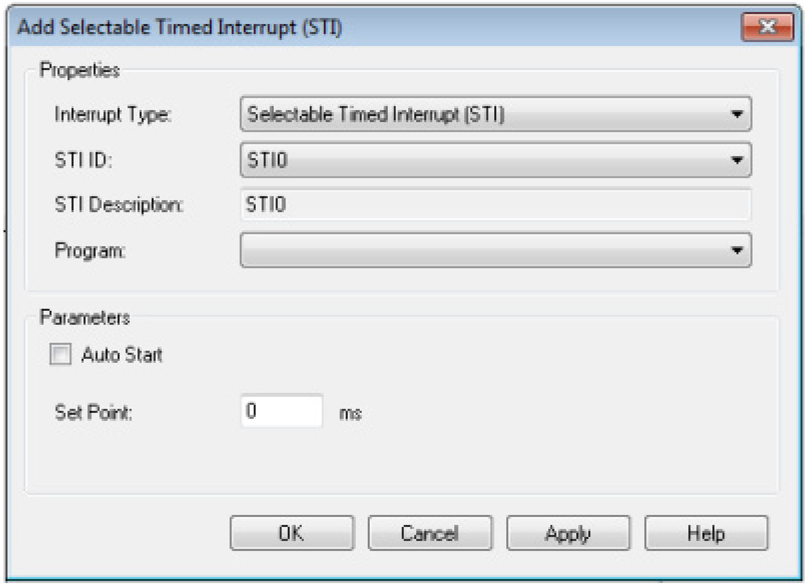
Variable Monitoring

Global Variables - Micro850 | Local Variables - N/A | System Variables - M

Name	Logical Value	Physical V.
My_Gains	...	
My_Gains.DirectActing	□	N/A
My_Gains.ProportionalGain	34.4978	N/A
My_Gains.TimeIntegral	27.6775	N/A
My_Gains.TimeDerivative	9.795	N/A
My_Gains.DerivativeGain	0.1	N/A
ManipulatedValue	0.0	N/A
OutGain	...	...
OutGain.DirectActing	□	N/A
OutGain.ProportionalGain	34.4978	N/A
OutGain.TimeIntegral	27.6775	N/A
OutGain.TimeDerivative	9.795	N/A
OutGain.DerivativeGain	0.1	N/A

Use a Selectable Timed Interrupt (STI) with auto-tuning

Although a PID instruction works if it is not controlled by a Selectable Timed Interrupt (STI), using an STI increases the auto-tune success rate because the auto-tune operates on a fixed cycle.



Add Selectable Timed Interrupt (STI)

Properties

Interrupt Type: Selectable Timed Interrupt (STI)

STI ID: STI0

STI Description: STI0

Program:

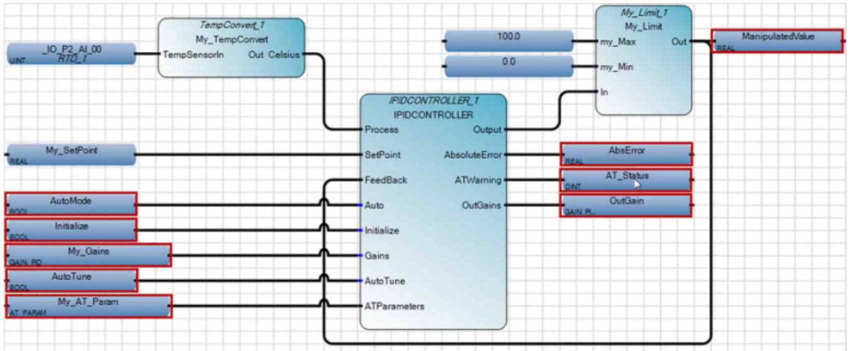
Parameters

□ Auto Start

Set Point: 0 ms

Example: IPIDController with auto-tune

The following example program shows the variables used to configure the parameters for auto-tuning.



Auto-tune parameters

The following table describes the variables that are used with each parameter in the example to configure auto-tuning.

Input parameters

Variable	Parameter	Description
AutoMode	Auto	The operation mode of the PID controller: TRUE - controller runs in normal mode. FALSE – derivative term is ignored forcing the controller output to track the feedback within the controller limits and allowing the controller to switch back to auto without bumping the output.
Initialize	Initialize	Initializes AutoTune sequence. A change in value from TRUE to FALSE or FALSE to TRUE causes the controller to eliminate any proportional gain during the cycle.
My_Gains	Gains	Establishes the Gains PID for IPIDController.
My_Gains.DirectActing	DirectActing	Defines the type of acting for the output. TRUE - direct acting in which the output moves in the same direction as the error. That is, the actual process value is greater than the SetPoint and the appropriate controller action is to increase the output. For example, chilling. FALSE - reverse acting in which the output moves in the opposite direction as the error. That is, the actual process value is greater than the SetPoint and the appropriate controller action is to decrease the output. For example: heating.
My_Gains.ProportionalGain	ProportionalGain	Proportional gain for PID (>= 0.0001).
My_Gains.TimeIntegral	TimeIntegral	Time integral value for PID (>= 0.0001). The tendency for oscillation increases with a decrease in ti.
My_Gains.TimeDerivative	TimeDerivative	Time derivative value for PID (> 0.0). Damping increases with an increase in derivative time, but decreases if the derivative time value is too large.
My_Gains.DerivativeGain	Derivative gain for PID (> 0.0).	
AutoTune	When set to TRUE and Auto and Initialize are FALSE, the AutoTune sequence is started.	
ATParameters		
Load	• Initial output value during auto-tuning. • Allows the process value to stabilize at the load	
Deviation	• The standard deviation for a series of stabilized process values. For example, if the process value stabilized between 31.4 to 32.0, then the deviation value would be (32.0-31.4)/2 = 0.3. • Some process values, such as temperature, take a very long time to stabilize.	
Step	• The auto-tune process considers how the process value reacts to the changes in step value and derives the Gain parameters.	
ATDynaSet	• Allocated time for the auto-tune to complete. It must be longer than what is required for the auto-tune process. • A common value for many systems is 600 seconds but some systems may require more time.	
ATReset	• If TRUE, the output will be reset to "0" after auto-tune completes. • If FALSE, the output will remain at the load value after auto-tune completes.	

Output parameters

Parameter	Description
AbsoluteError	Absolute error (Process – SetPoint) from the controller.
ATWarning	Warning for the Auto Tune sequence. Possible values are: • 0 - no auto tune done. • 1 - in auto tune mode. • 2 - auto tune done. • -1 - ERROR 1 input automatically set to TRUE, no auto tune possible. • -2 - ERROR 2 auto tune error, ATDynaSet expired

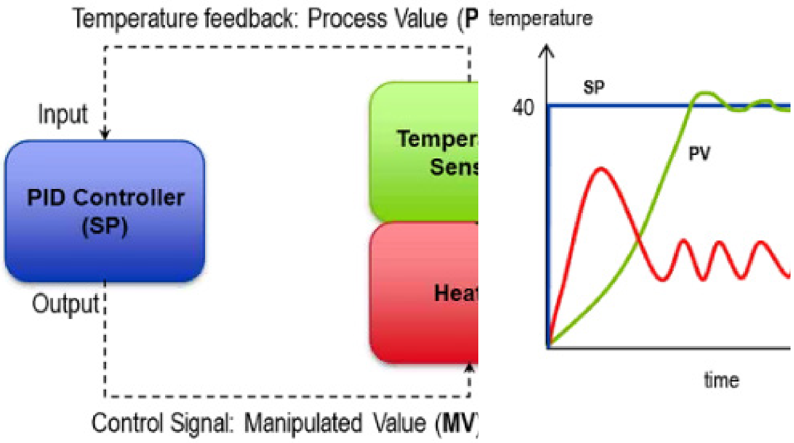
Input parameters	
OutGains	Gains calculated after AutoTune sequences.

Example: How to create a feedback loop for the manipulated value

Adding a feedback loop for the manipulated value prevents excessive overshooting by providing a minimum and maximum value for the MV.

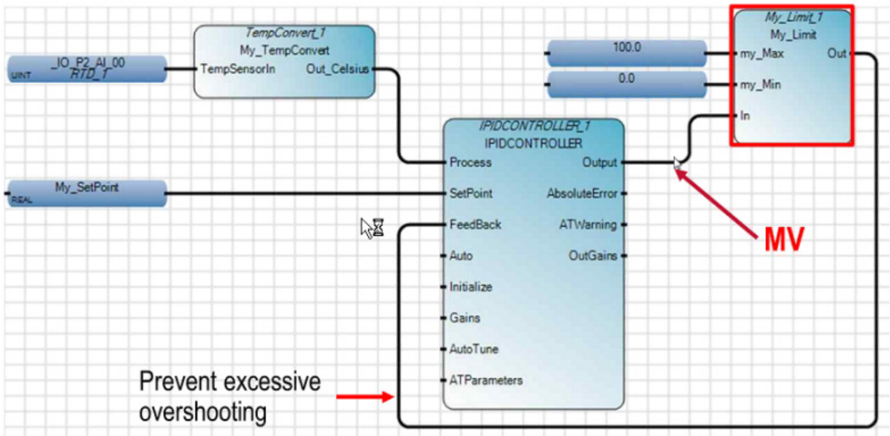
Temperature feedback loop example

At the beginning of the temperature control process, the difference between the process value (PV) and the setpoint value (SP) is large, as shown in the following graph. In this example of a temperature feedback loop, the PV starts at 0 degrees Celsius and moves towards the SP value of 40 degrees Celsius. Notice also that the fluctuation between the high and low manipulated value (MV) decreases and stabilizes with time. The behavior of the MV depends on the values used in each of the P, I, and D parameters.


Temperature feedback: Process Value (P
Input
PID Controller (SP)
Output
Temper Sens
Heat
Control Signal: Manipulated Value (MV)
temperature
SP
40
PV
time

IPIDController with a feedback loop

The following function block diagram includes a feedback loop for the manipulated value that prevents excessive overshooting by providing a minimum and maximum value for the MV.

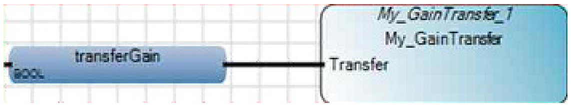


Example: How to add a UDFB to a PID program

For PID programs, use user-defined function blocks (UDFB) outside the main program to perform specialized functions such as converting units or transferring values. The following are examples of UDFBs.

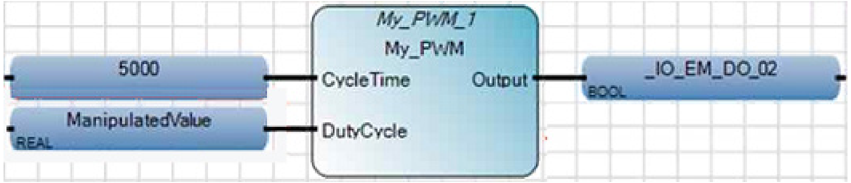
Transfer the auto-tune gain value

This UDFB transfers the Autotune gain value to My_GainTransfer so it can be used by the controller.



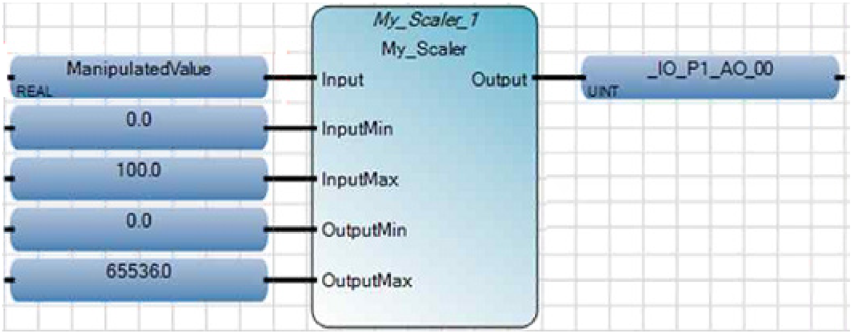
Convert a manipulated value to a digital output

This UDFB converts a manipulated value (MV) to a digital output (DO) so it can be used to control a digital input n(DI).



Convert a manipulated value to an analog output

This UDFB converts a manipulated value (MV) to an analog output (AO) so it can be used to control an analog input (AI).



Example: How to create an IPIDController program to control temperature

The temperature control program maintains the temperature within the control zone.

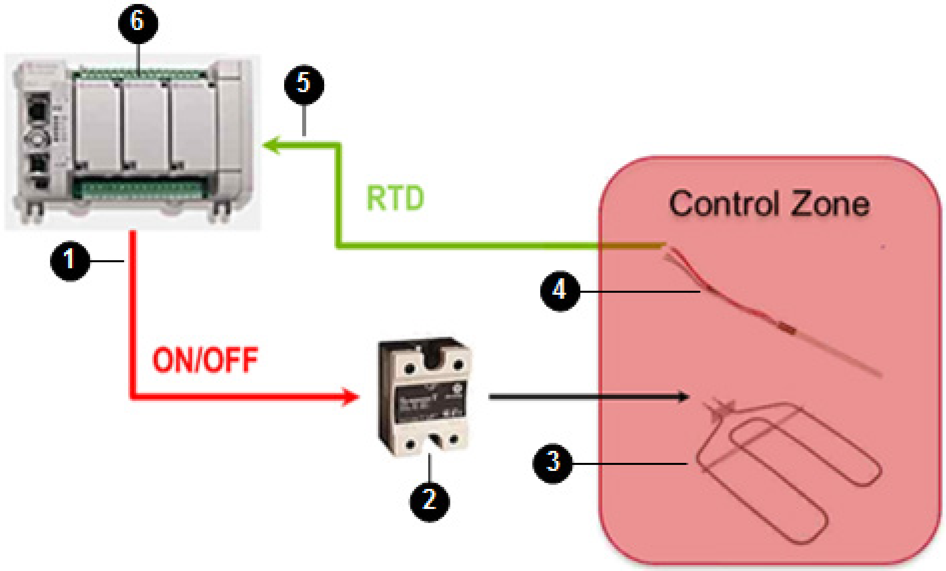
Setpoint, process and manipulated values

The following table defines the SP, PV, and MV values used in the temperature control program.

Item	Description
Setpoint (SP)	Measurement of temperature in degrees Celsius that defines the temperature for the control zone.
Process value (PV)	Must be converted to the same unit as the SP, which is a measurement of degrees Celsius.
Manipulated value (MV)	Must be converted to an analog value so it can be output to the PWM to control the heating element.

Temperature control system

The following diagram and table define the components in the temperature control system that are controlled by the temperature control program the events that occur when the control program runs.



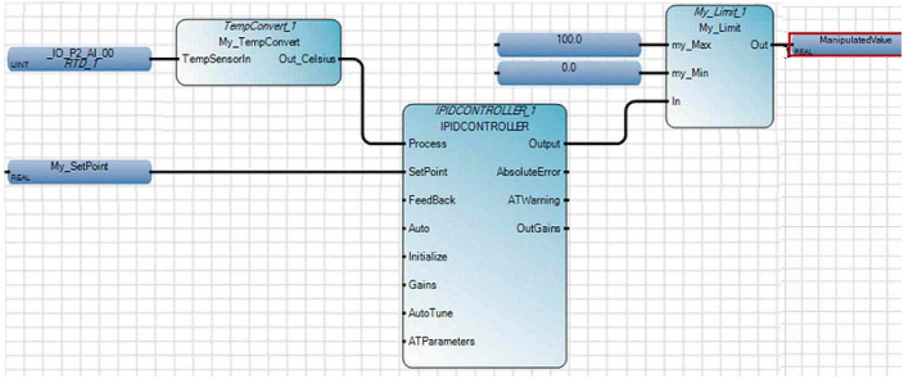
Sequence of events for temperature control program

The following table identifies the components in the temperature control system and describes the sequence of events that occurs in the system when the temperature control program runs.

No	Item	Description
❶	Controller output	Sends the MV to the PWM (On/Off).
❷	Pulse Width Modulation (PWM temperature controller	Solid state relay that controls the heating element.
❸	Heating element	Increases temperature in the control zone.
❹	Resistance temperature detector (RTD)	Measures the temperature in the control zone and sends the PV (RTD signal) to the controller input.
❺	Controller input	Receives the PV (RTD signal).
❻	PLC program	Converts the PV (RTD signal) to the same unit as the SP (degrees Celsius) and determines the difference between the PV and the SP and adjusts the MV according to the parameter values defined in the P, I and D parameters.

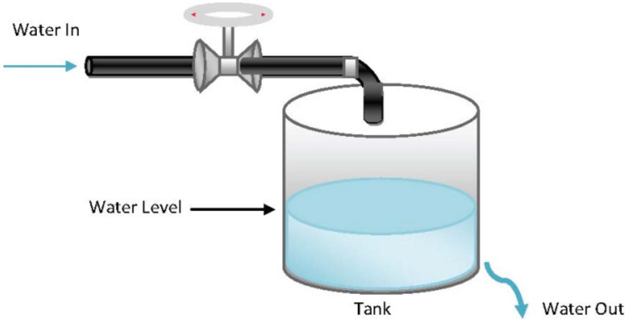
Example: Function block diagram to control temperature

This function block diagram shows the predefined and user-defined function blocks used in the application to control temperature in a control zone.



Example: How to create an IPIDController program to control water supply level

The water supply level control program example maintains sufficient water in a water supply tank that has an outflow. A solenoid valve controls incoming water and fills the tank at a preset rate; outflowing water is also controlled at a preset rate.



Program example information

The water supply level program example includes the following information.

- The sequence of events that occur in the control process
- How the setpoint, process and manipulated values are used in the control program
- An example function block diagram that shows the IPIDController and other instruction blocks

Setpoint, process and manipulated values

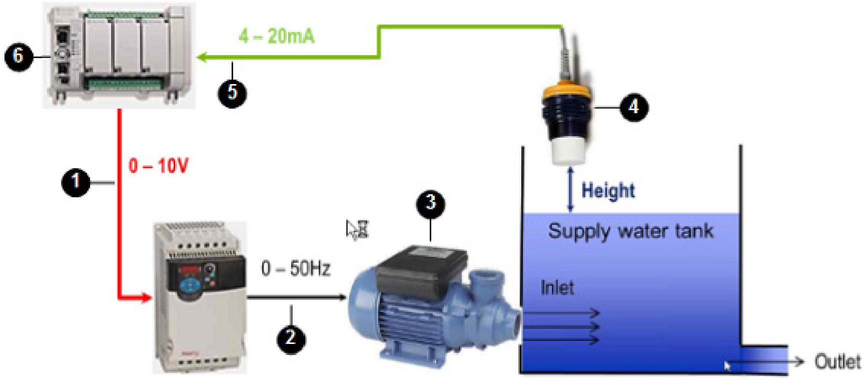
The following table defines how the SP, PV, and MV values are used in the water supply level program.

Item	Description
Setpoint (SP)	Measurement of height that defines the target water supply level.
Process value (PV)	The 4-20mA must be converted to the same unit as the SP, which is a measurement of height.

Item	Description
Manipulated value (MV)	Must be converted to an analog value so it can be output to the drive to control the pump.

Water supply level system

The following diagram shows the components in the water supply level system that are controlled by the water supply level program. The table following the diagram describes the events that occur when the control program runs.



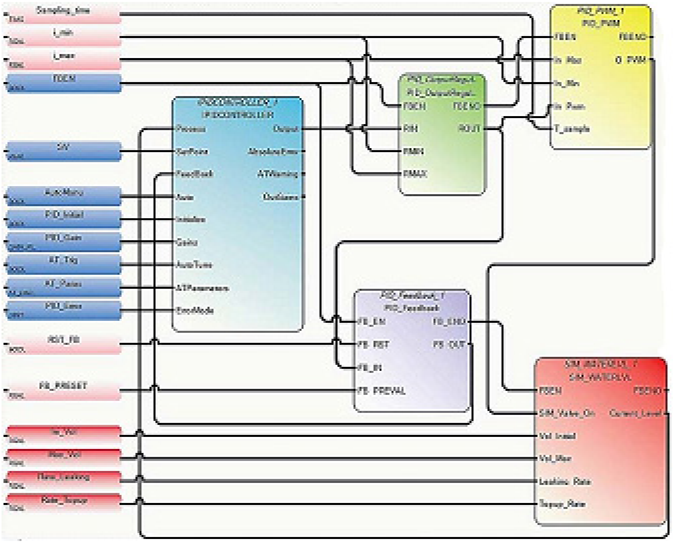
Sequence of events in water supply level system

The following table identifies the components in the water supply system and describes, in sequence, the events that occur in the system when the water supply level program runs.

No	Item	Description
❶	Controller output	Sends the MV to the PowerFlex drive (0-10V).
❷	PowerFlex drive	Controls the water pump (0-50Hz).
❸	Water pump	Controls the water level in the supply tank.
❹	Output transfer device	Measures the height of the water supply level (4-20mA) and sends the PV to the controller.
❺	Controller input	Receives the PV (water supply level of 4-20mA).
❻	PLC program	Converts the PV to the same unit as the SP (measurement of height) and determines the difference between the PV and SP and adjusts the MV according to the parameter values defined in the P, I and D parameters.

Example: Function block diagram to control water supply level

The following function block diagram shows the predefined and user-defined function blocks for the program to control the water supply level.



Function blocks and UDFBs used in the water level FBD

This application, developed in the Function Block Diagram (FBD) language, uses the instructions described in the following table.

Function block	Description
IPIDController function block	Provides PID process control.

PID_OutputRegulator UDFB	Regulates the output of the IPIDCONTROLLER within a safe range to ensure the hardware used in the process is not damaged.
	Sample code:
	IF RMIN ≤ RIN ≤ RMAX, then ROUT = RIN,
	IF RIN < RMIN, then ROUT = RMIN,
	IF RIN > RMAX, then ROUT = RMAX

PID_Feedback UDFB	Acts as a multiplexer. **Sample code:** IF "FB_RST" is false, FB_OUT=FB_IN; If "FB_RST" is true, then FB_OUT=FB_PREVAL.
PID_PWM UDFB	Provides a PWM function, converting a real value to a time related ON/OFF output.
SIM_WATERLVL UDFB	Simulates the process in the application example.

PID (proportional-integral-derivative)

An output instruction that controls physical properties such as temperature, pressure, liquid level, or flow rate using process loops.

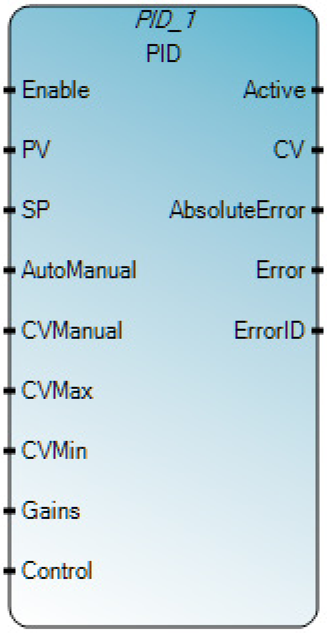
Operation details:

- When enabled, PID controls the process using the input parameters including SP and Gains of the PID controller.
- Run to Program mode transition, the PID instruction is disabled, parameter values are retained.

- Program to Run mode transition, the PID instruction remains disabled until a user resets Enable to true.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
Enable	Input	BOOL	Enable instruction. TRUE - Start execution with the current input parameters. FALSE - PID does not execute. Set CV to 0 and calculate AbsoluteError.
PV	Input	REAL	Process Value. This value is typically read from an analog input module. The SI unit must be the same as Setpoint.
SP	Input	REAL	The set point value for the process.
AutoManual	Input	BOOL	Auto or manual mode selection. TRUE - CV is controlled by PID. FALSE - PID is running and CV is controlled by CVManual input.
CVManual	Input	REAL	Control value input defined for manual mode operation. The valid range for CVManual is: CVMin < CVManual < CVMax
CVMin	Input	REAL	Control value minimum limit. If CV < CVMin, then CV = CVMin. If CVMin > CVMax, an error occurs.
CVMax	Input	REAL	Control value maximum limit. If CV > CVMax, then CV = CVMax. If CVMax < CVMin, an error occurs.
Gains	Input	PID_GAINS	Gains of PID for controller. Use the PID_GAINS data type to configure the Gains parameter.

Control	Input	BOOL	Control direction of the process: TRUE - Direct acting, such as Cooling. FALSE - Reverse acting, such as Heating.
Active	Output	BOOL	Status of the PID controller. TRUE - PID is active. FALSE - PID is stopped.
CV	Output	REAL	The control value output. If any error occurred, CV is 0.
AbsoluteError	Output	REAL	Absolute error is the difference between process value (PV) and set point (SP) value.
Error	Output	BOOL	Indicates the existence of an error condition. TRUE - Operation encountered an Error. FALSE - Operation completed successfully or the instruction is not executing.
ErrorID	Output	USINT	A unique numeric that identifies the error. The errors are defined in PID error codes.

PID_GAINS data type

The following table describes the PID_GAINS data type for the PID instruction on page 513.

Parameter	Parameter Type	Data Type	Description
Kc	Input	REAL	Controller gain for PID. Proportional and Integral are dependent on this gain. (>= 0.0001). Increasing Kc improves response time but also increases overshoot and oscillation of the PID. If Kc is invalid, an error occurs.
Ti	Input	REAL	Time integral constant in seconds (>= 0.0001). Increasing Ti decreases the overshoot and oscillation of the PID. If Ti is invalid, an error occurs.
Td	Input	REAL	Time derivative constant in seconds (>= 0.0). When Td equals 0, there is no derivative action and PID becomes a PI controller. Increasing Td reduces the overshot and removes the oscillation of the PID controller. If Td is invalid, an error occurs.
FC	Input	REAL	Filter constant (>= 0.0). Recommended range for FC is 0 to 20. Increasing FC smooths the response of the PID controller. If FC is invalid, an error occurs.

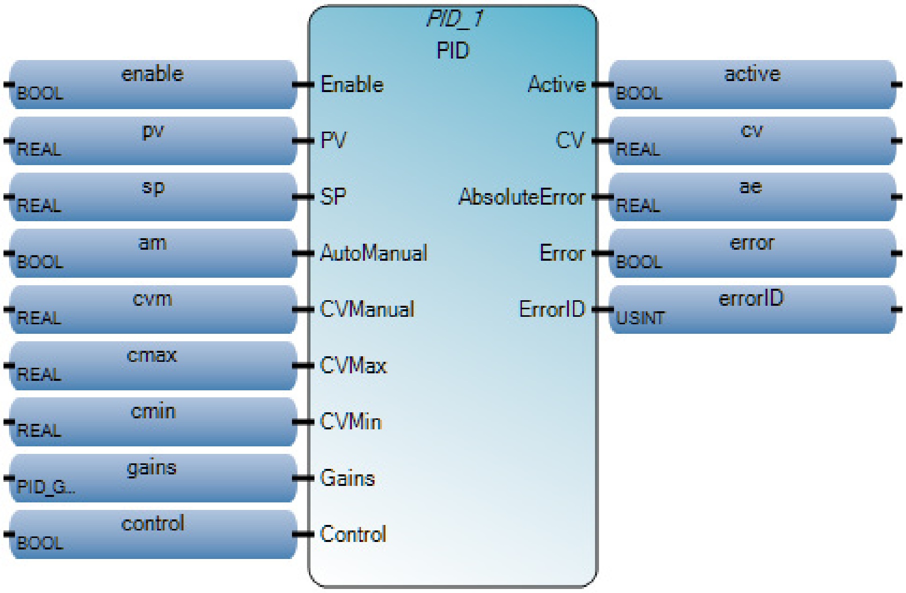
PID error codes

Use this table to determine the PID error codes and descriptions.

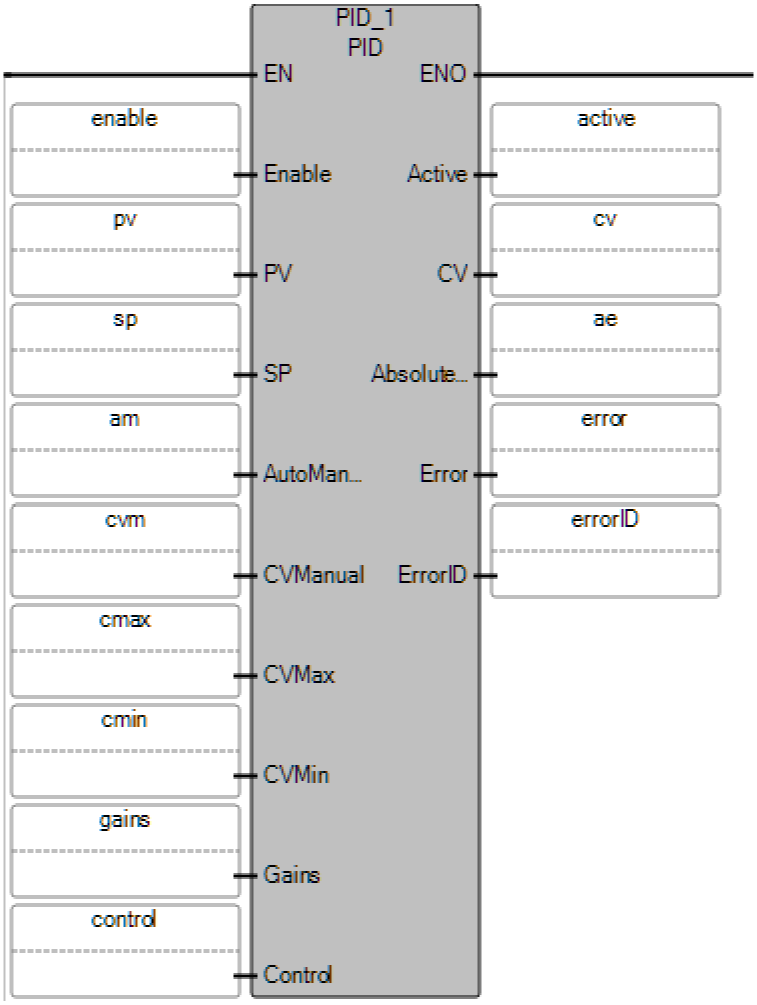
Error code	Error description
0	PID is working normally.
1	Kc is invalid.
2	Ti is invalid.
3	Td is invalid.
4	FC is invalid.
5	CVMin > CVMax or CVMax < CVMin

Error code	Error description
0	PID is working normally.
1	Kc is invalid.
2	Ti is invalid.
6	CVManual < CVMin CVManual is invalid.
7	CVManual > CVMax CVManual is invalid.

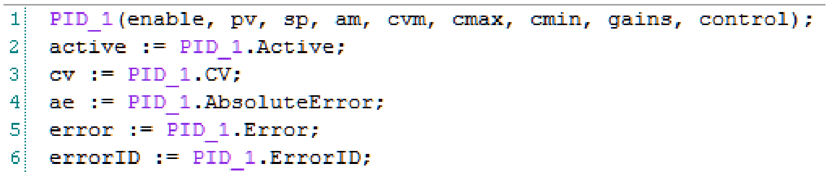
PID Function Block Diagram example



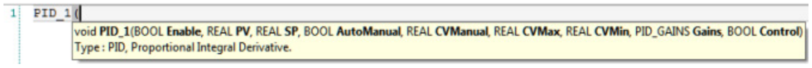
PID Ladder Diagram example



PID Structured Text example



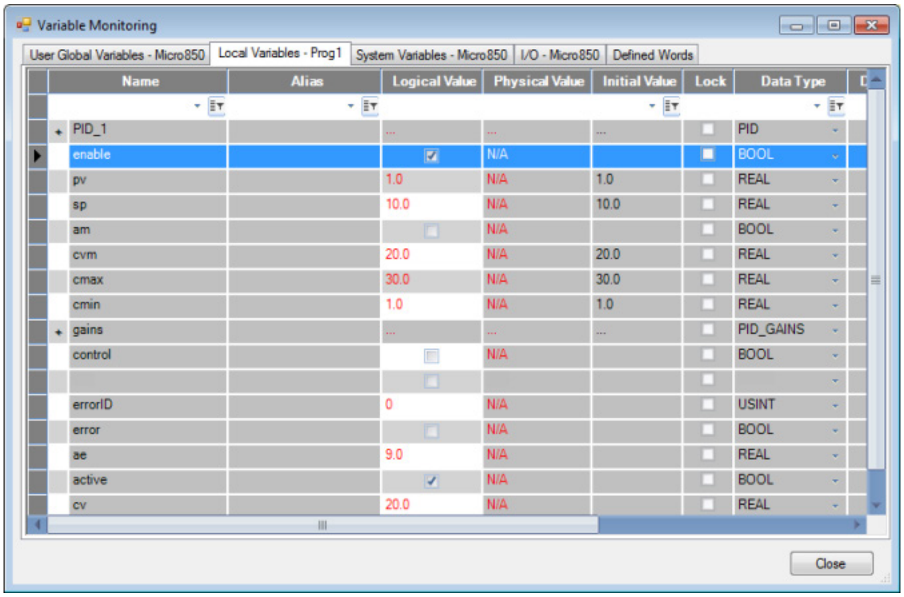
```
PID_1(enable, pv, sp, am, cvm, cmax, cmin, gains, control);
active := PID_1.Active;
cv := PID_1.CV;
ae := PID_1.AbsoluteError;
error := PID_1.Error;
errorID := PID_1.ErrorID;
```



```
PID_1(
```

void **PID_1**(BOOL **Enable**, REAL **PV**, REAL **SP**, BOOL **AutoManual**, REAL **CVManual**, REAL **CVMax**, REAL **CVMin**, PID_GAINS **Gains**, BOOL **Control**)
Type : PID, Proportional Integral Derivative.

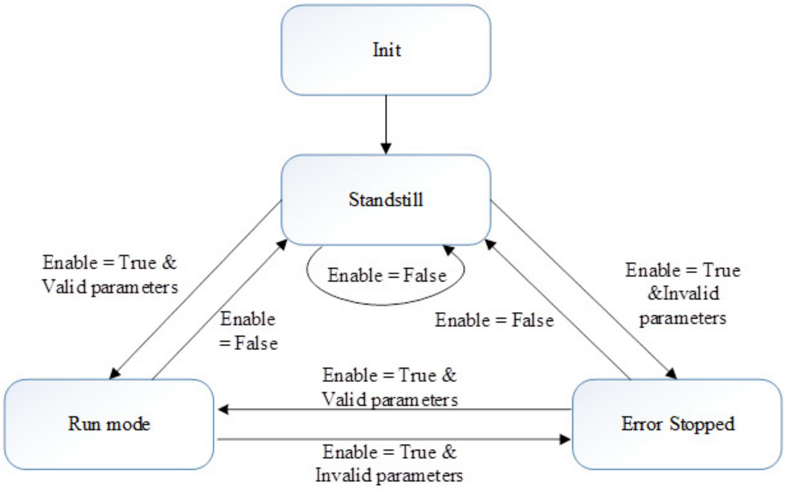
PID Results



Name	Alias	Logical Value	Physical Value	Initial Value	Lock	Data Type
+ PID_1		...	...	...	□	PID
enable		☑	N/A		□	BOOL
pv		1.0	N/A	1.0	□	REAL
sp		10.0	N/A	10.0	□	REAL
am		□	N/A		□	BOOL
cvm		20.0	N/A	20.0	□	REAL
cmax		30.0	N/A	30.0	□	REAL
cmin		1.0	N/A	1.0	□	REAL
+ gains		...	...	...	□	PID_GAINS
control		□	N/A		□	BOOL
		□			□	
errorID		0	N/A		□	USINT
error		□	N/A		□	BOOL
ae		9.0	N/A		□	REAL
active		☑	N/A		□	BOOL
cv		20.0	N/A		□	REAL

PID instruction state machine

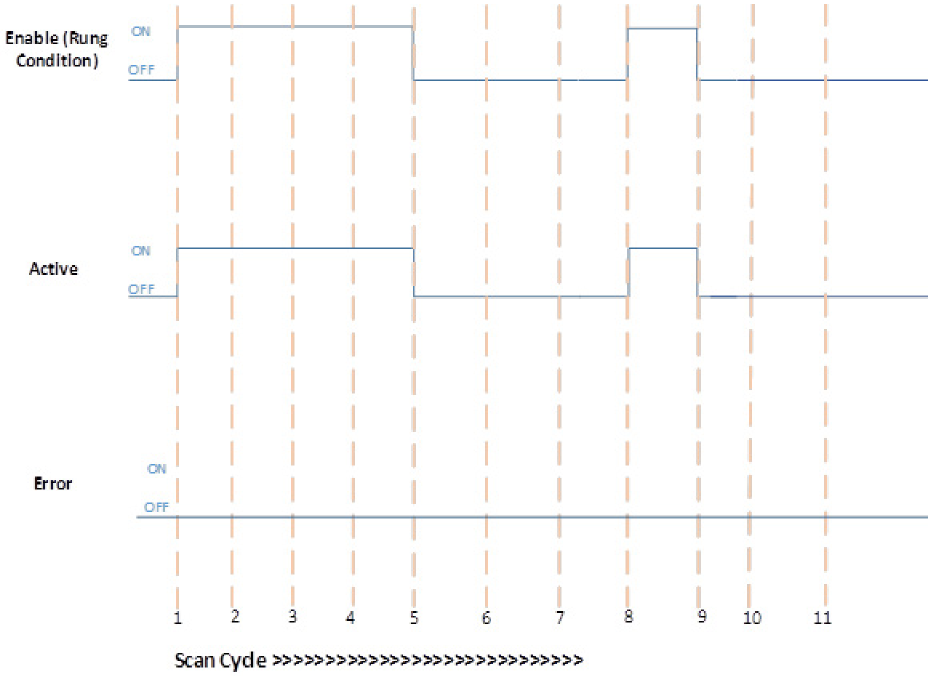
The PID State Machine diagram describes the processing states for the PID instruction on page 536.



PID instruction timing diagrams

The following timing diagram examples describe execution scenarios for the PID (proportional-integral-derivative) instruction on page 536.

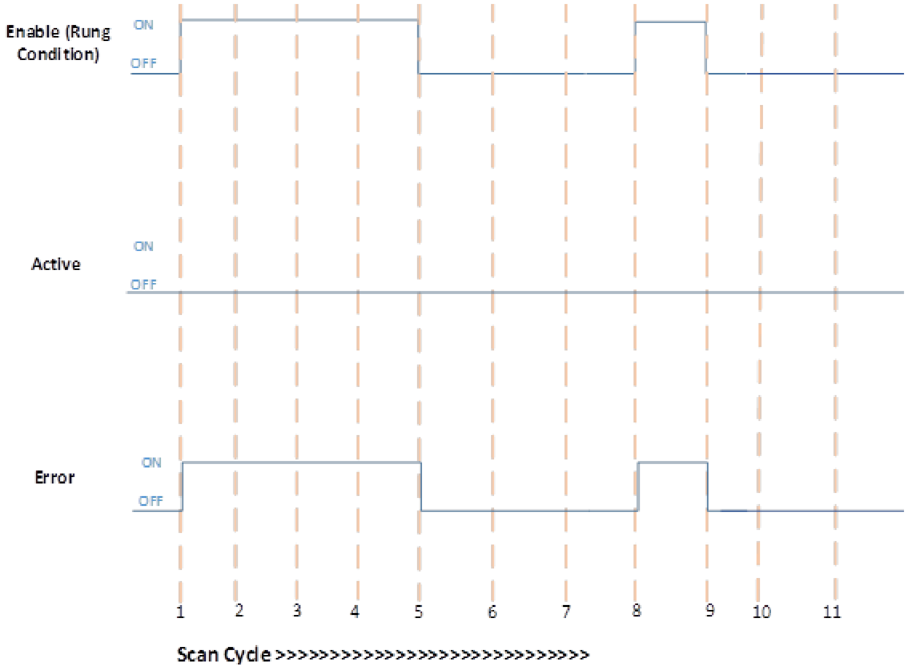
Successful PID execution



Use this table to help determine the parameter values for each scan cycle.

Scan Cycle	Description
1, 8	Rung condition becomes TRUE when: • Enable input bit is TRUE. • Input parameters are valid. • Active bit is TRUE. • Error bit is FALSE.
2,3,4	No change in Rung condition. • Enable input bit is TRUE. • Input parameters are valid. • Updates PID output parameters.
5, 9	Rung condition becomes FALSE when: • Enable bit is FALSE. • Clears PID output parameters except AbsoluteError. • AbsoluteError calculates based on PV and SP input values.
6, 7, 10, 11	No change in Rung condition. • Enable bit is FALSE. • Clears PID output parameters except AbsoluteError. • AbsoluteError calculates based on PV and SP input values.

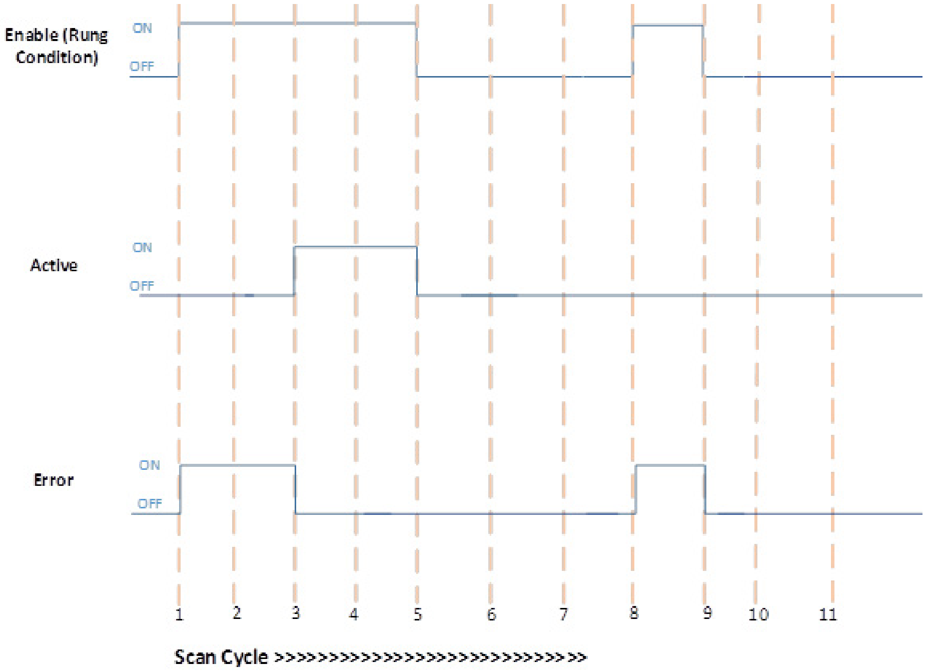
PID execution with Error



Use this table to help determine the parameter values for each scan cycle.

Scan Cycle	Description
1, 8	Rung condition becomes TRUE when: • Enable input bit is TRUE. • Input parameters are invalid. • Active bit is FALSE. • Error bit is TRUE. ErrorID output is set. • CV output is set to 0. • AbsoluteError calculates based on PV and SP input values.
2,3,4	No change in Rung condition. • Enable input bit is TRUE. • Input parameters are invalid. • Updates PID output parameters.
5, 9	Rung condition becomes FALSE when: • Enable bit is FALSE. • Clears PID output parameters except AbsoluteError. • AbsoluteError calculates based on PV and SP input values.
6, 7, 10, 11	No change in Rung condition. • Enable bit is FALSE. • Clears PID output parameters except AbsoluteError. • AbsoluteError calculates based on PV and SP input values.

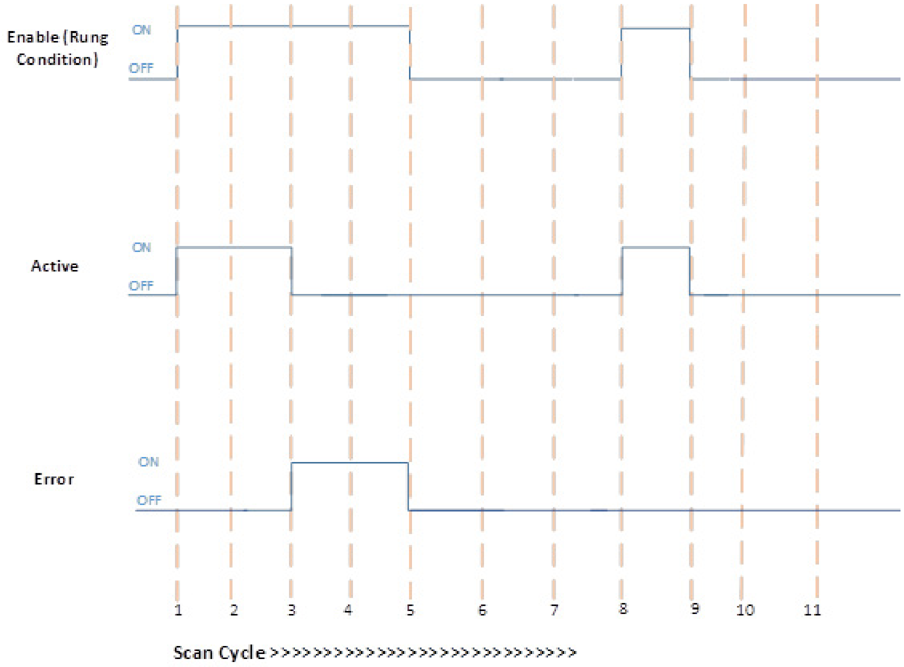
PID execution with Error then successful execution



Use this table to help determine the parameter values for each scan cycle.

Scan Cycle	Description
1, 8	Rung condition becomes TRUE when: • Enable input bit is TRUE. • Input parameters are invalid. • Active bit is FALSE. • Error bit is TRUE. ErrorID output is set. • CV output is set to 0. • AbsoluteError calculates based on PV and SP input values.
2	No change in Rung condition. • Enable input bit is TRUE. • Input parameters are invalid. • Updates PID output parameters.
3, 4	No change in Rung condition. • Enable input bit is TRUE. • Input parameters are valid. • Active bit is TRUE. • Error bit is FALSE. • Updates PID output parameters.
5, 9	Rung condition becomes FALSE when: • Enable bit is FALSE. • Clears PID output parameters except AbsoluteError. • AbsoluteError calculates based on PV and SP input values.
6, 7, 10, 11	No change in Rung condition. • Enable bit is FALSE. • Clears PID output parameters except AbsoluteError. • AbsoluteError calculates based on PV and SP input values.

Successful PID execution and Error



Use this table to help determine the parameter values for each scan cycle.

Scan Cycle	Description
1, 8	Rung condition becomes TRUE when: • Enable input bit is TRUE. • Input parameters are valid. • Active bit is TRUE. • Error bit is FALSE. • Update PID output parameters.
2	No change in Rung condition. • Enable input bit is TRUE. • Input parameters are valid. • Update PID output parameters.
3, 4	No change in Rung condition. • Enable input bit is TRUE. • Input parameters are invalid. • Error bit is TRUE. ErrorID output is set. • CV output is set to 0. • AbsoluteError calculates based on PV and SP input values.
5, 9	Rung condition becomes FALSE when: • Enable bit is FALSE. • Clear PID output parameters except AbsoluteError. • AbsoluteError calculates based on PV and SP input values.
6, 7, 10, 11	No change in Rung condition. • Enable bit is FALSE. • Clears PID output parameters except AbsoluteError. • AbsoluteError calculates based on PV and SP input values.

Chapter 22

Real Time Clock (RTC) instructions

Use Real Time Clock instructions to configure the calendar and the clock.

Instruction	Description
RTC_READ on page 369	Reads the real-time clock (RTC) module information.
RTC_SET on page 371	Sets real-time clock data to the RTC module information.

RTC_READ (read real-time clock)

Reads the real-time clock (RTC) module information.

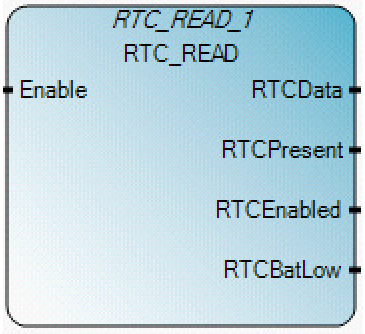
Operation details:

- Micro810 or Micro820 controller with embedded RTC:
  - RTCBatLow is always set to zero (0).
  - RTCEnabled is always set to one (1).
- When the embedded RTC has lost its charge/memory due to loss of power:
  - RTCData is set to 2000/1/1/0/0/0.
  - RTCEnabled is set to one (1).

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.

The outputs are always reset when the instruction is applied to the simulated controller (2080-LC50-48QWB-SIM).



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description

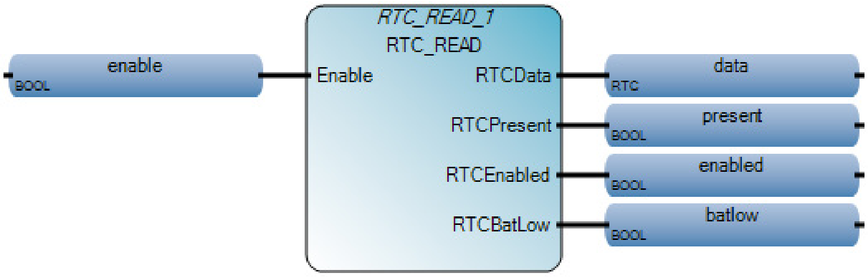
Enable	Input	BOOL	Instruction block enable. TRUE - execute RTC information read. FALSE - there is no read operation and output RTC data is invalid.
RTCData	Output	RTC	RTC data information: yy/mm/dd, hh/mm/ss, week. RTCData output is defined using the RTC data type.
RTCPresent	Output	BOOL	TRUE - Free Running clock is utilized, or RTC hardware is plugged in. FALSE - Free Running clock is not utilized, or RTC hardware is not plugged in.
RTCEnabled	Output	BOOL	TRUE - Free Running clock is utilized, or RTC hardware is enabled (timing). FALSE - Free Running clock is not utilized, RTC hardware is disabled (not timing).
RTCBatLow	Output	BOOL	TRUE - RTC battery is low. FALSE - RTC battery is not low.
ENO	Output	BOOL	Enable output. Applies only to Ladder Diagram programs.

RTC data type

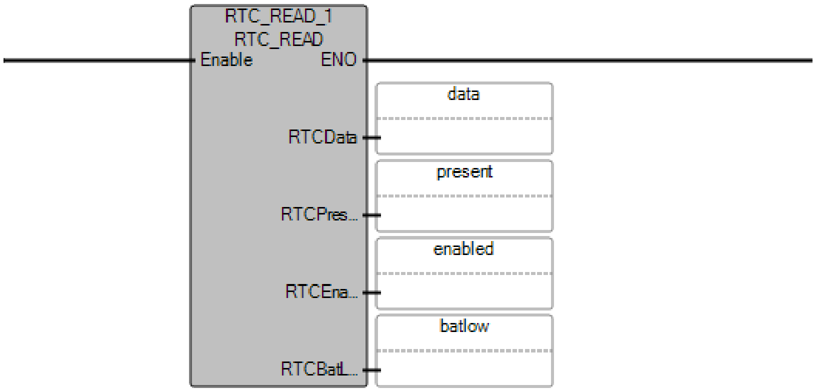
Use this table to help determine the parameter values for the RTC data type.

Parameter	Data type	Description
Year	UINT	The year setting for the RTC. 16-bit value, and the valid range is from 2000 (Jan 01, 00:00:00) to 2098 (Dec. 31, 23:59:59)
Month	UINT	The month setting for the RTC.
Day	UINT	The day setting for the RTC.
Hour	UINT	The hour setting for the RTC.
Minute	UINT	The minute setting for the RTC.
Second	UINT	The second setting for the RTC.
DayOfWeek	UINT	The day of the week setting for the RTC. This parameter is ignored for RTC_SET.

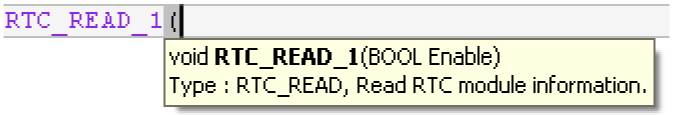
RTC_READ Function Block Diagram example



RTC_READ Ladder Diagram example

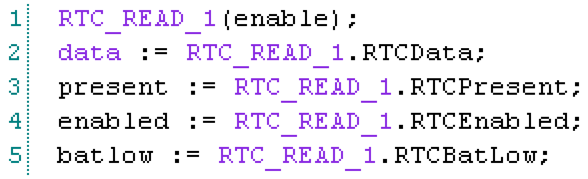


RTC_READ Structured Text example



```
RTC_READ_1(
```

void **RTC_READ_1**(BOOL Enable)
Type : RTC_READ, Read RTC module information.



```
RTC_READ_1(enable);
data := RTC_READ_1.RTCData;
present := RTC_READ_1.RTCPresent;
enabled := RTC_READ_1.RTCEnabled;
batlow := RTC_READ_1.RTCBatLow;
```

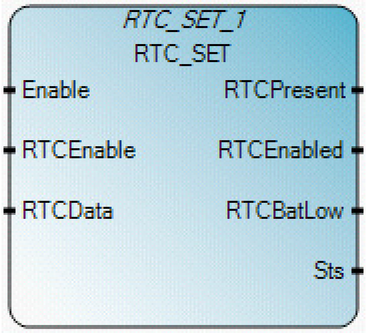
RTC_SET (set real-time clock)

Set RTC (real-time clock) data to the RTC module information.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.

The outputs are always reset when the instruction is applied to the simulated controller (2080-LC50-48QWB-SIM).



Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description

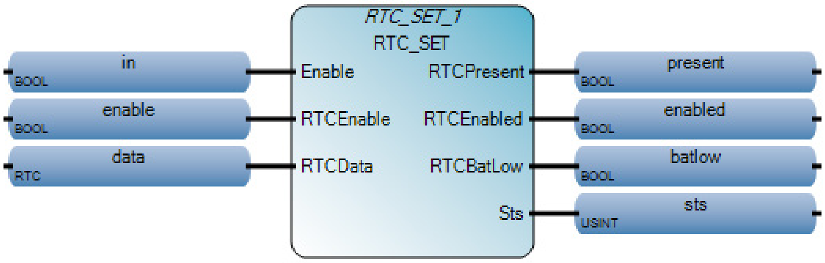
Enable	Input	BOOL	Instruction block enable. TRUE - execute RTC_SET with the RTC information from input. Typically, only execute for 1 program scan when updating the RTC. FALSE - do not execute RTC_SET. Set to FALSE to operate RTC normally.
RTCEnable	Input	BOOL	TRUE - To enable RTC with the RTC data specified. FALSE - To disable RTC.
RTCData	Input	RTC	RTC data information: yy/mm/dd, hh/mm/ss, week as defined in the RTC data type. RTCData is ignored when RTCEnable = 0.
RTCPresent	Output	BOOL	TRUE - Free Running clock is utilized, or RTC hardware is plugged in. FALSE - Free Running clock is not utilized, or RTC hardware is not plugged in.
RTCEnabled	Output	BOOL	TRUE - Free Running clock is utilized, or RTC hardware is enabled (timing). FALSE - Free Running clock is not utilized, or RTC hardware is disabled (not timing).
RTCBatLow	Output	BOOL	TRUE - RTC battery is low. FALSE - RTC battery is not low.
Sts	Output	USINT	The read operation status. RTC_Set status (Sts) values: • 0x00 - Function block not enabled (no operation). • 0x01 - RTC set operation success. • 0x02 - RTC set operation fails.

RTC data type

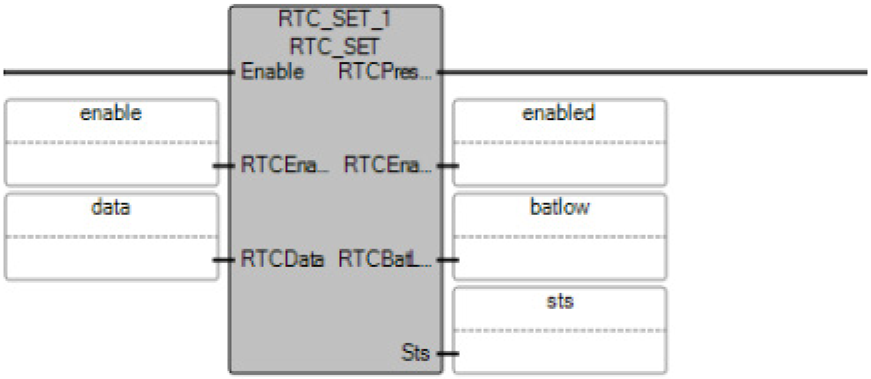
Use this table to help determine the parameter values for the RTC data type.

Parameter	Data type	Description
Year	UINT	The year setting for the RTC. 16-bit value, and the valid range is from 2000 (Jan 01, 00:00:00) to 2098 (Dec. 31, 23:59:59)
Month	UINT	The month setting for the RTC.
Day	UINT	The day setting for the RTC.
Hour	UINT	The hour setting for the RTC.
Minute	UINT	The minute setting for the RTC.
Second	UINT	The second setting for the RTC.
DayOfWeek	UINT	The day of the week setting for the RTC. This parameter is ignored for RTC_SET.

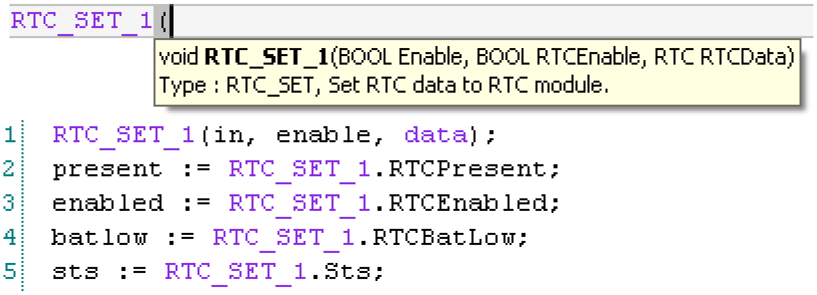
RTC_SET Function Block Diagram example



RTC_SET Ladder Diagram example



RTC_SET Structured Text example



```
RTC_SET_1(
```

void **RTC_SET_1**(BOOL Enable, BOOL RTCEnable, RTC RTCData)
Type : RTC_SET, Set RTC data to RTC module.

```
RTC_SET_1(in, enable, data);
present := RTC_SET_1.RTCPresent;
enabled := RTC_SET_1.RTCEnabled;
batlow := RTC_SET_1.RTCBatLow;
sts := RTC_SET_1.Sts;
```

Socket instructions

Use the Sockets protocol for Ethernet communications to devices that do not support Modbus TCP and EtherNet/IP. Sockets support client, server, Transmission Control Protocol (TCP), and User Datagram Protocol (UDP). Typical applications include communicating to printers, bar codes readers, and personal computers.

Socket instruction behavior in **Run Mode Change** mode:

- If a delete operation is performed on a SOCKET_OPEN, SOCKET_ACCEPT, SOCKET_READ, or SOCKET_WRITE instruction while in **Run Mode Change** mode, the Socket instance is deleted.
- In **Run Mode Change** mode, any change to a SOCKET_READ input while operating in the BUSY state results in an error and the received packet is discarded. SOCKET_READ input parameters are: Length, Offset, Data Array Size, Data Array Variable.
- SOCKET_READ is the only SOCKET instruction that supports add or modify operations while in **Run Mode Change** mode.
- If the Ethernet IP settings are modified using **Run Mode Change** all created Socket instances are deleted, similar to SOCKET_DELETEALL.

Instruction Processing and Output Updates for Socket instructions:

- **Asynchronous**: Respective instructions where all outputs update asynchronously with user program scan, for example a Ladder scan. Asynchronous output can not be used for edge trigger detection. Asynchronous output parameters are not locked and can be updated after completion of respective socket instructions.
- **Synchronous**: Respective instructions where all output update is in sync with user program scans. Synchronous output parameters are locked and can not be modified after completion of respective socket instruction.
- **Hybrid**: Respective instructions where a few outputs update in sync with user program scan. Remaining outputs update asynchronously with user program scan.
- **Immediate Instruction Execution**: Instruction completes desired function prior to going to next instruction.
- **Non-Immediate Instruction Execution**: Instruction takes more than one program scan to complete desired function. Instructions take a snap shot of input parameters when a **False > True** transition is detected.

Use this table to help determine Socket instructions usage.

Instruction	Description	TCP Client	TCP Server	UDP with Open	UDP without Open	Instruction Processing	Instruction Output Update
SOCKET_ACCEPT on page 554	Accepts a TCP connection request from a remote destination and returns a socket instance used to send and receive data on the newly created connection.	NO	YES	NO	NO	Non-Immediate	Hybrid
SOCKET_CREATE on page 557	Creates an instance of the Socket and returns an instance number that uses the next socket operations.	YES	YES	YES	YES	Immediate	Synchronous
SOCKET_DELETE on page 562	Deletes a created socket instance. TCP connections are closed prior to deletion.	YES	YES	YES	YES	Non-Immediate	Synchronous
SOCKET_DELETEALL on page 564	Deletes all created socket instances.	YES	YES	YES	YES	Non-Immediate	Synchronous
SOCKET_INFO on page 566	Returns information for the socket such as error codes and execution status.	YES	YES	YES	YES	Immediate	Synchronous
SOCKET_OPEN on page 572	TCP connections open with the specified destination address. UDP connections associate a destination IP address and port number with the specified socket.	YES	NO	YES	NO	Non-Immediate	Synchronous
SOCKET_READ on page 576	Reads data on a socket. Attempts to receive the specified number of bytes and returns the number of bytes received.	YES	YES	YES	YES	Non-Immediate	Hybrid
SOCKET_WRITE on page 580	Sends data on a socket. Attempts to send the requested number of bytes and returns the number of bytes sent.	YES	YES	YES	YES	Non-Immediate	Hybrid

SOCKET_ACCEPT

For Transmission Control Protocol (TCP) connections only. Accepts a TCP connection request from a remote destination and returns a Socket Instance used to send and receive data on the newly created connection.

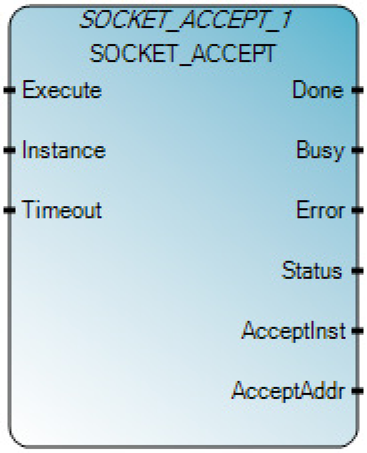
Operation details:

- Before executing Socket_Accept execute SOCKET_CREATE and specify the local port number for the accept connection.
- Outputs are updated synchronously from the program scan.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro850, and Micro870 controllers. For the Micro800 Simulator, this instruction can be added to a program but is only a placeholder to prevent the instruction from being deleted during controller change.

The outputs are always reset when the instruction is applied to the simulated controller (2080-LC50-48QWB-SIM).

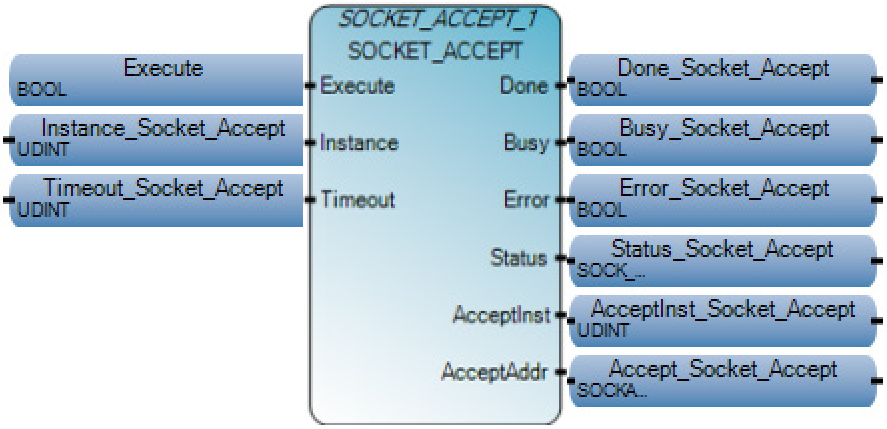


Use this table to help determine the parameter values for this instruction.

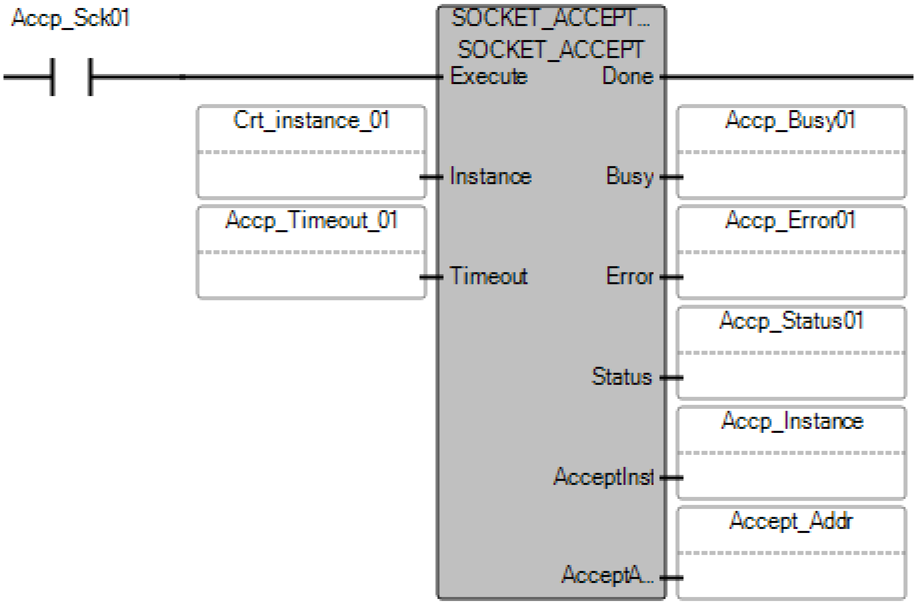
Parameter	Parameter type	Data type	Description
Execute	Input	BOOL	Instruction block enable. TRUE - Rising Edge detected, start the instruction block with the precondition that the last operation has been completed. FALSE - no Rising Edge detected; instruction block not started.
Instance	Input	UDINT	Identifies the socket instance. Copy the returned Socket Handler from the **SOCKET_CREATE** instruction.
Timeout	Input	UDINT	Timeout for the **SOCKET_ACCEPT** instruction block instances. The function block returns an Error if the timeout value is less than minimum value. Timeout range: 1000- 86400000 milliseconds Set Timeout to 0 to use the default value 10000 (10 Second).
Done	Output	BOOL	Indicates when operation is complete. TRUE - operation completed successfully. FALSE - operation is in progress or encountered an error condition. Output is updated synchronously from the program scan.
Busy	Output	BOOL	TRUE - the operation is not finished. FALSE - the operation is finished. Output is updated synchronously from the program scan.
Error	Output	BOOL	Indicates an error occurred. TRUE - An error is detected. FALSE - No error. Output is updated synchronously from the program scan.
Status	Output	SOCK_STATUS	Status is defined using the **SOCK_STATUS** data type on page 584 which contains ErrorID on page 585, SubErrorID, and StatusBits on page 588 information. Output is updated synchronously from the program scan.
AcceptInst	Output	UDINT	Contains the Accept Instance for this Socket Instance. Use the unique Accept Instance number with subsequent **SOCKET_READ** and **SOCKET_WRITE** for this connection. Output is updated synchronously from the program scan.

Parameter	Parameter type	Data type	Description
AcceptAddr	Output	SOCKADDR_CFG	A data structure that contains the Accept Address for the socket. For more information, refer to SOCKADDR_CFG data type on page 584. To specify an IP Address of 192.168.2.100 and Port = 12000: • AcceptAddr.IPAddress[0]=192 • AcceptAddr.IPAddress[1]=168 • AcceptAddr.IPAddress[2]=2 • AcceptAddr.IPAddress[3]=100 • AcceptAddr.Port = 12000 Output is updated synchronously from the program scan.

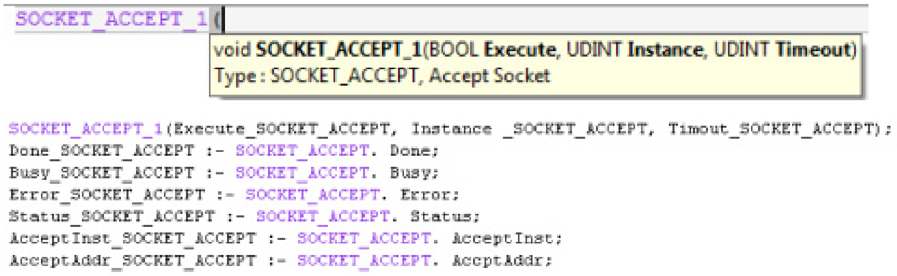
SOCKET_ACCEPT Function Block Diagram example



SOCKET_ACCEPT Ladder Diagram example



SOCKET_ACCEPT Structured Text example



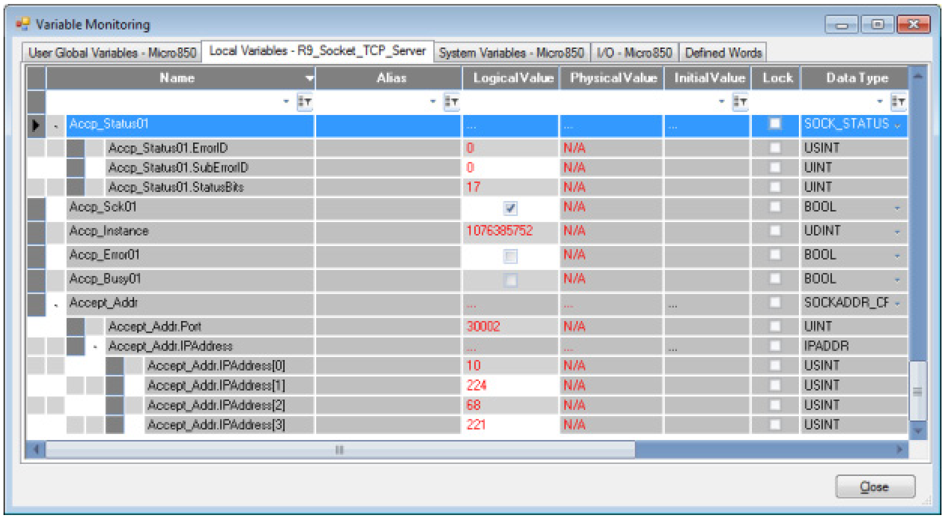
```
SOCKET_ACCEPT_1(
```

void **SOCKET_ACCEPT_1**(BOOL **Execute**, UDINT **Instance**, UDINT **Timeout**)
Type : SOCKET_ACCEPT, Accept Socket

```
SOCKET_ACCEPT_1(Execute_SOCKET_ACCEPT, Instance _SOCKET_ACCEPT, Timout_SOCKET_ACCEPT);
Done_SOCKET_ACCEPT :- SOCKET_ACCEPT. Done;
Busy_SOCKET_ACCEPT :- SOCKET_ACCEPT. Busy;
Error_SOCKET_ACCEPT :- SOCKET_ACCEPT. Error;
Status_SOCKET_ACCEPT :- SOCKET_ACCEPT. Status;
AcceptInst_SOCKET_ACCEPT :- SOCKET_ACCEPT. AcceptInst;
AcceptAddr_SOCKET_ACCEPT :- SOCKET_ACCEPT. AcceptAddr;
```

Results

TCP Server


Variable Monitoring
User Global Variables - Micro850
Local Variables - R9_Socket_TCP_Server
System Variables - Micro850
I/O - Micro850
Defined Words
Close

Name	Alias	Logical Value	Physical Value	Initial Value	Lock	Data Type
Accp_Status01		...	...	...	☐	SOCK_STATUS
Accp_Status01.ErrorID		0	N/A		☐	USINT
Accp_Status01.SubErrorID		0	N/A		☐	UINT
Accp_Status01.StatusBits		17	N/A		☐	UINT
Accp_Sck01		☑	N/A		☐	BOOL
Accp_Instance		1076385752	N/A		☐	UDINT
Accp_Error01		☐	N/A		☐	BOOL
Accp_Busy01		☐	N/A		☐	BOOL
Accept_Addr		...	...	...	☐	SOCKADDR_CF
Accept_Addr.Port		30002	N/A		☐	UINT
Accept_Addr.IPAddress		...	...	...	☐	IPADDR
Accept_Addr.IPAddress[0]		10	N/A		☐	USINT
Accept_Addr.IPAddress[1]		224	N/A		☐	USINT
Accept_Addr.IPAddress[2]		68	N/A		☐	USINT
Accept_Addr.IPAddress[3]		221	N/A		☐	USINT

SOCKET_CREATE

Creates an instance of the socket and returns an instance number that is used as an input in any follow-on socket operations.

Operation details:

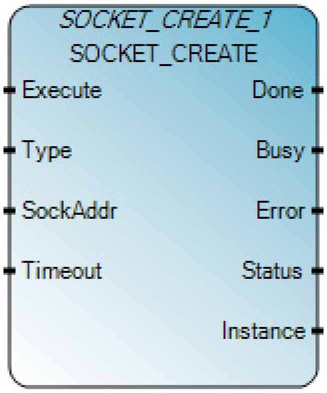
- Socket instructions on page 553 support Full Duplex communication with remote devices.
- User Datagram Protocol (UDP) connections supports a maximum of eight queued UDP datagrams packets. The queue contains the most recent packets.
- Micro820 and Micro850 controllers at revision 9 or higher support up to eight Socket Instances. The Socket instances support UDP and TCP Sockets:
  - Use all eight instances for client Transmission Control Protocol (TCP) connections.
  - Use all eight instances to listen for incoming TCP connections and then accept eight connections from other devices.
  - Perform both TCP client and server operations.
  - Perform both TCP and UDP operations.

- To accept incoming TCP connections to the same port, create a new Socket Instance.
- TCP Connection Loss:
    - User application program should detect the loss of TCP connections and handle the event. Depending on the user application, consider the option to Fault the controller.
        - Fault the controller.
        - Try to re-establish the connection.
    - To re-establish communications with another device:
        - Delete the Socket Instance for the lost connection.
        - If the connection is a TCP client, create a new Socket Instance using SOCKET_CREATE and execute SOCKET_OPEN to the target device.
        - If the connection is a TCP server, create a new Socket Instance using SOCKET_CREATE and execute SOCKET_ACCEPT to wait for another connection from the remote device.
    - Application Messages for TCP connections:
        - A TCP connection is a byte stream between two applications. The application protocol determines the message formats.
        - Messages can be fixed size or variable size.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro850, and Micro870 controllers. For the Micro800 Simulator, this instruction can be added to a program but is only a placeholder to prevent the instruction from being deleted during controller change.

The outputs are always reset when the instruction is applied to the simulated controller (2080-LC50-48QWB-SIM).



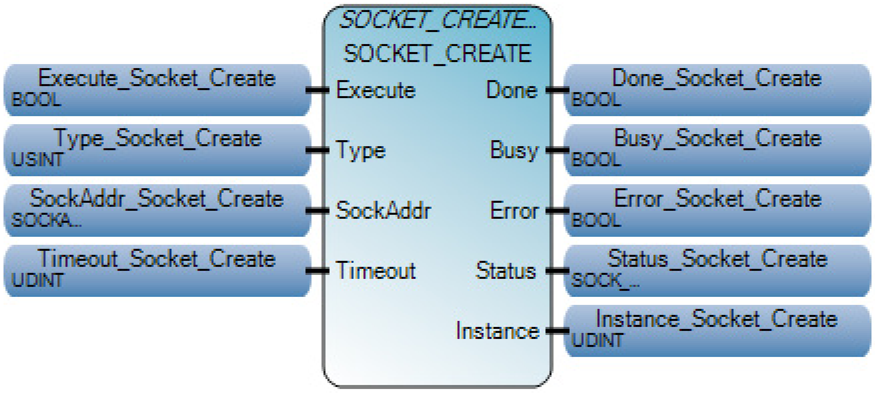
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter type	Data type	Description

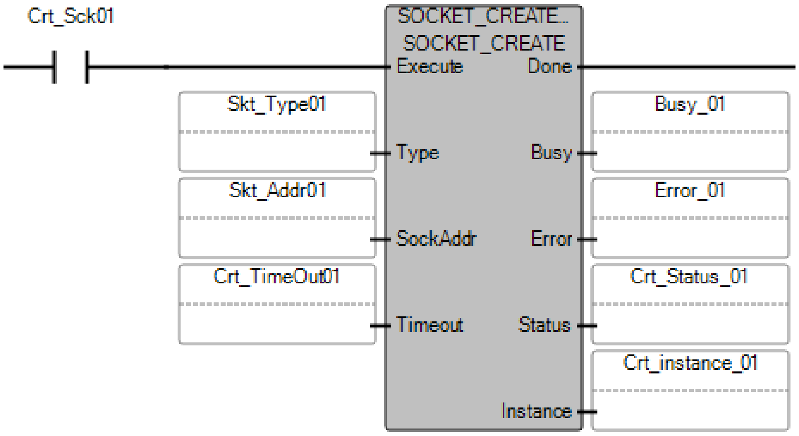
Parameter	Parameter type	Data type	Description
Execute	Input	BOOL	Instruction block enable. TRUE - Rising Edge detected, start the instruction block with the precondition that the last operation has been completed. FALSE - no Rising Edge detected.
Type	Input	USINT	Specify the Type of Socket: • Transmission Control Protocol (TCP). • User Datagram Protocol (UDP).
SockAddr	Input	SOCKADDR_CFG	Specify the address configuration for the socket. The EtherNet/IP module to choose the local port number, set SockAddr to 0. Specify the local port number where an application is listening and receiving, or : • Array elements must all be zero • For TCP client operations, specify 0 unless you want a specific local port number. • For TCP server communication, specify the port number to accept incoming connection requests. • For UDP, specify a local port number to receive datagrams on a specific port. Local port range: 1 to 65535. An error occurs if the specified local port number is already in use by the Micro820 or Micro850 controller. The controller uses the following port numbers: TCP Ports: • EtherNet/IP: 44818 • ModbusTCP: 502 • DHCP Server: 67 • DHCP Client: 68 UDP Ports: • EtherNet/IP: 2222 • DHCP Server: 67 • DHCP Client: 68 See **SOCKADDR_CFG** data type on page 584.
Timeout	Input	UDINT	Specify the Timeout for Socket inactivity. If a socket instance does not receive any requests within the specified inactivity timeout, the socket instance is deleted. If a request is sent after the sock instance is deleted an error, Socket instance not supported, is returned. The instruction block returns an error when the timeout value is less than the minimum value. Set Timeout so it is longer than the longest interval between socket operations. If the inactivity Timeout is too short socket instances may timeout. Timeout range: 1000 - 86400000 milliseconds Set **Timeout** to 0 to use the default value 300000 (5 minutes).
Done	Output	BOOL	Indicates when operation is complete. TRUE - operation completed successfully. FALSE - operation is in progress or encountered an error condition.
Busy	Output	BOOL	TRUE - the operation is incomplete. FALSE - the operation is complete.
Error	Output	BOOL	Indicates an error occurred. TRUE - An error is detected. FALSE - No error.

Parameter	Parameter type	Data type	Description
Status	Output	SOCK_STATUS	Status is defined using the **SOCK_STATUS** data type which contains ErrorID, SubErrorID, and StatusBits information. See SOCK_STATUS data type on page 584, Socket instruction status bits on page 588, and Socket error codes on page 585.
Instance	Output	UDINT	Contains Socket Handler. Use the Instance parameter for subsequent Socket instructions.

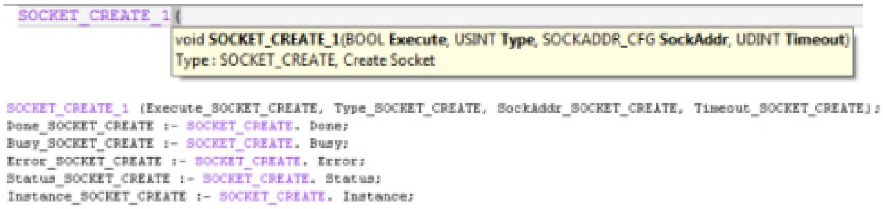
SOCKET_CREATE Function Block Diagram example



SOCKET_CREATE Ladder Diagram example



SOCKET_CREATE Structured Text example

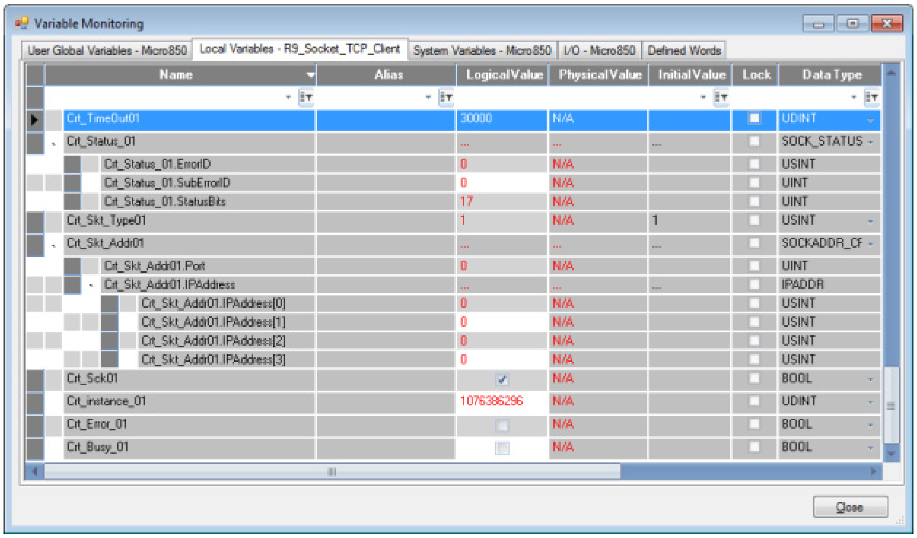


```
SOCKET_CREATE_1(
void SOCKET_CREATE_1(BOOL Execute, USINT Type, SOCKADDR_CFG SockAddr, UDINT Timeout)
Type : SOCKET_CREATE, Create Socket
```

```
SOCKET_CREATE_1 (Execute_SOCKET_CREATE, Type_SOCKET_CREATE, SockAddr_SOCKET_CREATE, Timeout_SOCKET_CREATE);
Done_SOCKET_CREATE := SOCKET_CREATE. Done;
Busy_SOCKET_CREATE := SOCKET_CREATE. Busy;
Error_SOCKET_CREATE := SOCKET_CREATE. Error;
Status_SOCKET_CREATE := SOCKET_CREATE. Status;
Instance_SOCKET_CREATE := SOCKET_CREATE. Instance;
```

Results

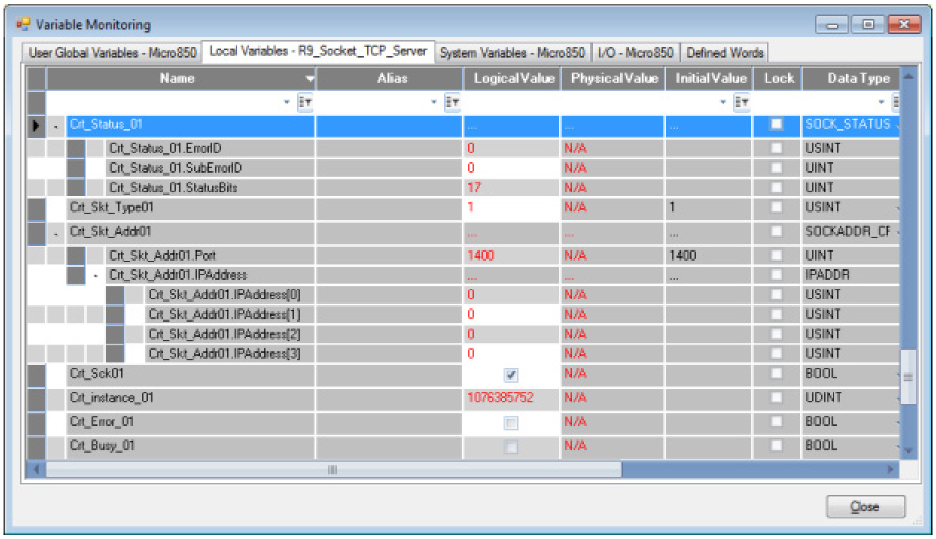
TCP Client example



Variable Monitoring
User Global Variables - Micro850
Local Variables - R9_Socket_TCP_Client
System Variables - Micro850
I/O - Micro850
Defined Words
Close

Name	Alias	Logical Value	Physical Value	Initial Value	Lock	Data Type
Crt_TimeOut01		30000	N/A		□	UDINT
Crt_Status_01		...	...	...	□	SOCK_STATUS
Crt_Status_01.ErrorID		0	N/A		□	USINT
Crt_Status_01.SubErrorID		0	N/A		□	UINT
Crt_Status_01.StatusBits		17	N/A		□	UINT
Crt_Skt_Type01		1	N/A	1	□	USINT
Crt_Skt_Addr01		...	...	...	□	SOCKADDR_CF
Crt_Skt_Addr01.Port		0	N/A		□	UINT
Crt_Skt_Addr01.IPAddress		...	...	...	□	IPADDR
Crt_Skt_Addr01.IPAddress[0]		0	N/A		□	USINT
Crt_Skt_Addr01.IPAddress[1]		0	N/A		□	USINT
Crt_Skt_Addr01.IPAddress[2]		0	N/A		□	USINT
Crt_Skt_Addr01.IPAddress[3]		0	N/A		□	USINT
Crt_Sck01		☑	N/A		□	BOOL
Crt_instance_01		1076386296	N/A		□	UDINT
Crt_Error_01		□	N/A		□	BOOL
Crt_Busy_01		□	N/A		□	BOOL

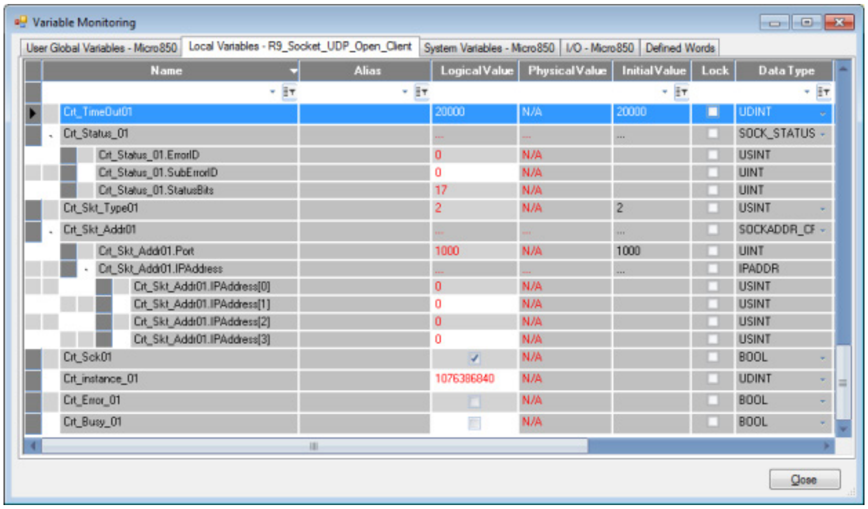
TCP Server example



Variable Monitoring
User Global Variables - Micro850
Local Variables - R9_Socket_TCP_Server
System Variables - Micro850
I/O - Micro850
Defined Words
Close

Name	Alias	Logical Value	Physical Value	Initial Value	Lock	Data Type
Crt_Status_01		...	...	...	□	SOCK_STATUS
Crt_Status_01.ErrorID		0	N/A		□	USINT
Crt_Status_01.SubErrorID		0	N/A		□	UINT
Crt_Status_01.StatusBits		17	N/A		□	UINT
Crt_Skt_Type01		1	N/A	1	□	USINT
Crt_Skt_Addr01		...	...	...	□	SOCKADDR_CF
Crt_Skt_Addr01.Port		1400	N/A	1400	□	UINT
Crt_Skt_Addr01.IPAddress		...	...	...	□	IPADDR
Crt_Skt_Addr01.IPAddress[0]		0	N/A		□	USINT
Crt_Skt_Addr01.IPAddress[1]		0	N/A		□	USINT
Crt_Skt_Addr01.IPAddress[2]		0	N/A		□	USINT
Crt_Skt_Addr01.IPAddress[3]		0	N/A		□	USINT
Crt_Sck01		☑	N/A		□	BOOL
Crt_instance_01		1076385752	N/A		□	UDINT
Crt_Error_01		□	N/A		□	BOOL
Crt_Busy_01		□	N/A		□	BOOL

UDP example



Variable Monitoring

User Global Variables - Micro850 | Local Variables - R9_Socket_UDP_Open_Client | System Variables - Micro850 | I/O - Micro850 | Defined Words

Name	Alias	Logical Value	Physical Value	Initial Value	Lock	Data Type
Crt_TimeOut01		20000	N/A	20000	□	UDINT
Crt_Status_01		...	...	...	□	SOCK_STATUS
Crt_Status_01.ErrorID		0	N/A		□	USINT
Crt_Status_01.SubErrorID		0	N/A		□	UINT
Crt_Status_01.StatusBits		17	N/A		□	UINT
Crt_Skt_Type01		2	N/A	2	□	USINT
Crt_Skt_Addr01		...	...	...	□	SOCKADDR_CF
Crt_Skt_Addr01.Port		1000	N/A	1000	□	UINT
Crt_Skt_Addr01.IPAddress		...	...	...	□	IPADDR
Crt_Skt_Addr01.IPAddress[0]		0	N/A		□	USINT
Crt_Skt_Addr01.IPAddress[1]		0	N/A		□	USINT
Crt_Skt_Addr01.IPAddress[2]		0	N/A		□	USINT
Crt_Skt_Addr01.IPAddress[3]		0	N/A		□	USINT
Crt_Sck01		☑	N/A		□	BOOL
Crt_instance_01		1076386840	N/A		□	UDINT
Crt_Error_01		□	N/A		□	BOOL
Crt_Busy_01		□	N/A		□	BOOL

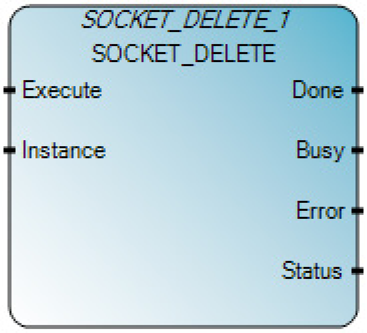
SOCKET_DELETE

Deletes a created socket instance. For Transmission Control Protocol (TCP) connections, **SOCKET_DELETE** also closes (Passive Close) the connection before deleting the instance. Outputs are updated synchronously from the program scan.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro850, and Micro870 controllers. For the Micro800 Simulator, this instruction can be added to a program but is only a placeholder to prevent the instruction from being deleted during controller change.

The outputs are always reset when the instruction is applied to the simulated controller (2080-LC50-48QWB-SIM).

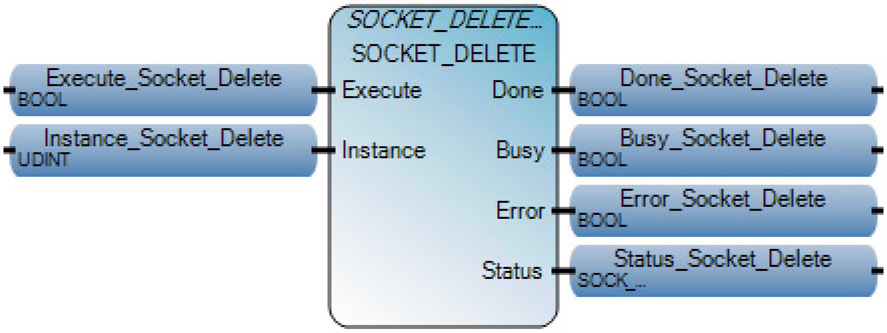


Use this table to help determine the parameter values for this instruction.

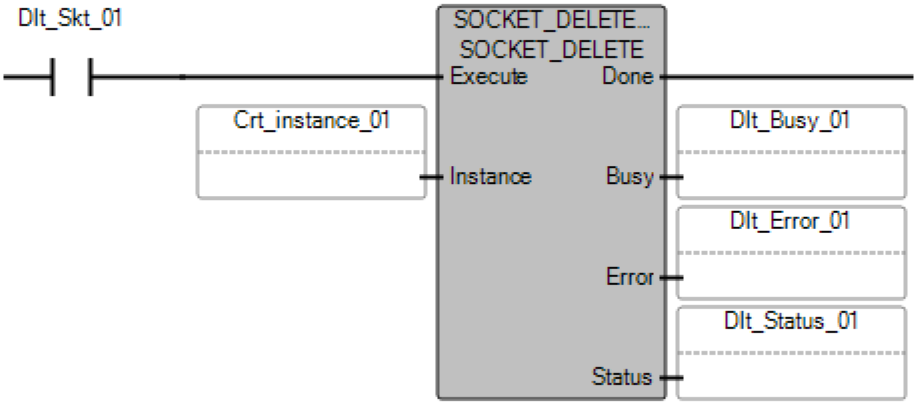
Parameter	Parameter type	Data type	Description
Execute	Input	BOOL	Instruction block enable. TRUE - Rising Edge detected, start the instruction block with the precondition that the last operation has been completed. FALSE - no Rising Edge detected.

Parameter	Parameter type	Data type	Description
Instance	Input	UDINT	Copy the returned Socket Handler from a SOCKET_CREATE or SOCKET_ACCEPT instruction to delete the respective socket. • For UDP and TCP Client Socket types, copy the returned Socket Handler from a SOCKET_CREATE instruction. • For TCP Server socket type, copy the returned Socket Handler from a SOCKET_ACCEPT instruction.
Done	Output	BOOL	Indicates when operation is complete. TRUE - operation completed successfully. FALSE - operation is in progress or encountered an error condition. Output is updated synchronously from the program scan.
Busy	Output	BOOL	TRUE - the operation is not finished. FALSE - the operation is finished. Output is updated synchronously from the program scan.
Error	Output	BOOL	Indicates an error occurred. TRUE - An error is detected. FALSE - No error. Output is updated synchronously from the program scan.
Status	Output	SOCK_STATUS	Status is defined using the SOCK_STATUS data type on page 584 which contains ErrorID on page 585, SubErrorID, and StatusBits on page 588 information. Output is updated synchronously from the program scan.

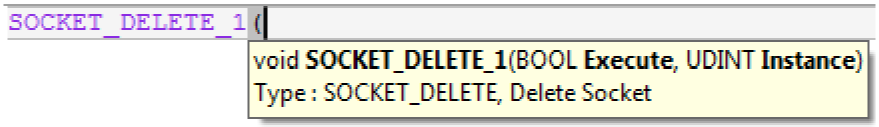
SOCKET_DELETE Function Block Diagram example



SOCKET_DELETE Ladder Diagram example

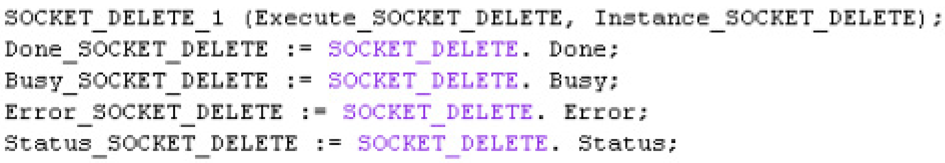


SOCKET_DELETE Structured Text example



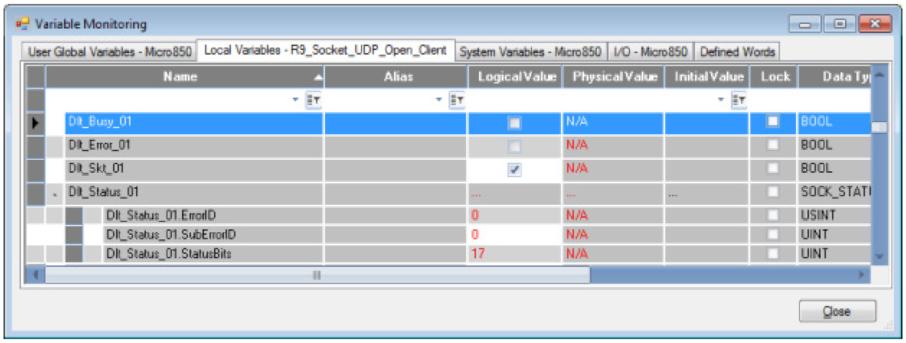
```
SOCKET_DELETE_1(
```

void **SOCKET_DELETE_1**(BOOL **Execute**, UDINT **Instance**)
Type : SOCKET_DELETE, Delete Socket



```
SOCKET_DELETE_1 (Execute_SOCKET_DELETE, Instance_SOCKET_DELETE);
Done_SOCKET_DELETE := SOCKET_DELETE. Done;
Busy_SOCKET_DELETE := SOCKET_DELETE. Busy;
Error_SOCKET_DELETE := SOCKET_DELETE. Error;
Status_SOCKET_DELETE := SOCKET_DELETE. Status;
```

Results



Variable Monitoring
User Global Variables - Micro850
Local Variables - R9_Socket_UDP_Open_Client
System Variables - Micro850
I/O - Micro850
Defined Words
Close

Name	Alias	Logical Value	Physical Value	Initial Value	Lock	Data Ty
Dlt_Busy_01		□	N/A		□	BOOL
Dlt_Error_01		□	N/A		□	BOOL
Dlt_Skt_01		☑	N/A		□	BOOL
Dlt_Status_01		...	...	...	□	SOCK_STAT
Dlt_Status_01.ErrorID		0	N/A		□	USINT
Dlt_Status_01.SubErrorID		0	N/A		□	UINT
Dlt_Status_01.StatusBits		17	N/A		□	UINT

SOCKET_DELETEALL

Deletes all created socket instances.

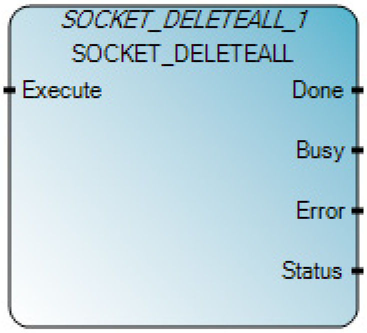
Operation details:

- If the Ethernet cable is disconnected from the controller or the controller IP address changes, you can execute SOCKET_DELETEALL to delete all previously created socket instances.
- Outputs are updated synchronously from the program scan.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro850, and Micro870 controllers. For the Micro800 Simulator, this instruction can be added to a program but is only a placeholder to prevent the instruction from being deleted during controller change.

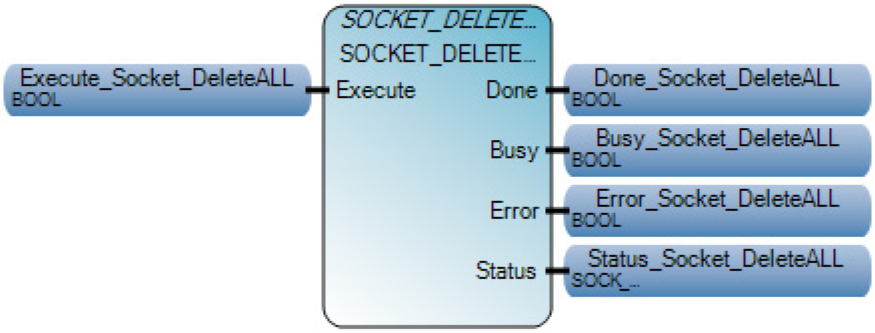
The outputs are always reset when the instruction is applied to the simulated controller (2080-LC50-48QWB-SIM).



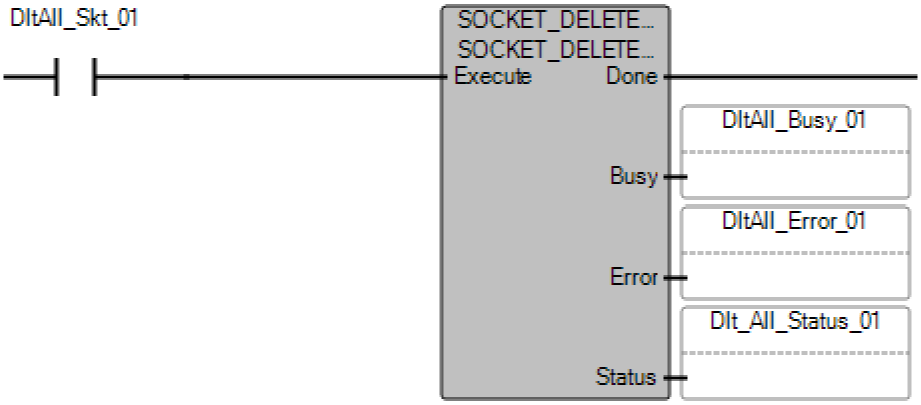
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter type	Data type	Description
Execute	Input	BOOL	Instruction block enable. TRUE - Rising Edge detected, start the instruction block with the precondition that the last operation has been completed. FALSE - no Rising Edge detected.
Done	Output	BOOL	Indicates when operation is complete. TRUE - operation completed successfully. FALSE - operation is in progress or encountered an error condition. Output is updated synchronously from the program scan.
Busy	Output	BOOL	TRUE - the operation is not finished. FALSE - the operation is finished. Output is updated synchronously from the program scan.
Error	Output	BOOL	This field is set to TRUE when the function block execution encounters an error condition. For more information refer to Socket error codes on page 585. Output is updated synchronously from the program scan.
Status	Output	SOCK_STATUS	Status is defined using the SOCK_STATUS data type on page 584 which contains ErrorID on page 585, SubErrorID, and StatusBits on page 588 information. Output is updated synchronously from the program scan.

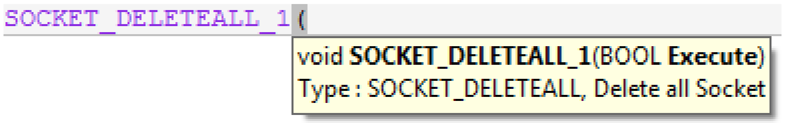
SOCKET_DELETEALL Function Block Diagram examples



SOCKET_DELETEALL Ladder Diagram examples



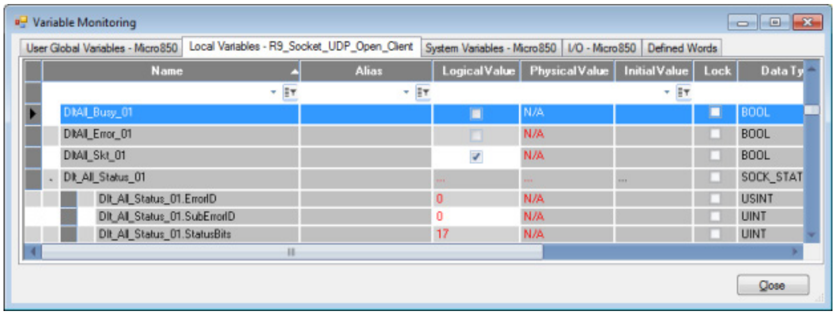
SOCKET_DELETEALL Structured Text example



```
SOCKET_DELETEALL_1(
void SOCKET_DELETEALL_1(BOOL Execute)
Type : SOCKET_DELETEALL, Delete all Socket
```

```
SOCKET_DELETEALL (Execute_SOCKET_DELETEALL);
Done_Socket_DeleteAll := SOCKET_DELETE. Done;
Busy_Socket_DeleteAll := SOCKET_DELETE. Busy;
Error_Socket_DeleteAll := SOCKET_DELETE. Error;
Status_Socket_DeleteAll := SOCKET_DELETE. Status;
```

Results


Variable Monitoring
User Global Variables - Micro850
Local Variables - R9_Socket_UDP_Open_Client
System Variables - Micro850
I/O - Micro850
Defined Words
Close

Name	Alias	Logical Value	Physical Value	Initial Value	Lock	Data Ty
DltAll_Busy_01		□	N/A		□	BOOL
DltAll_Error_01		□	N/A		□	BOOL
DltAll_Skt_01		☑	N/A		□	BOOL
Dlt_All_Status_01		...	...	...	□	SOCK_STAT
Dlt_All_Status_01.ErrorID		0	N/A		□	USINT
Dlt_All_Status_01.SubErrorID		0	N/A		□	UINT
Dlt_All_Status_01.StatusBits		17	N/A		□	UINT

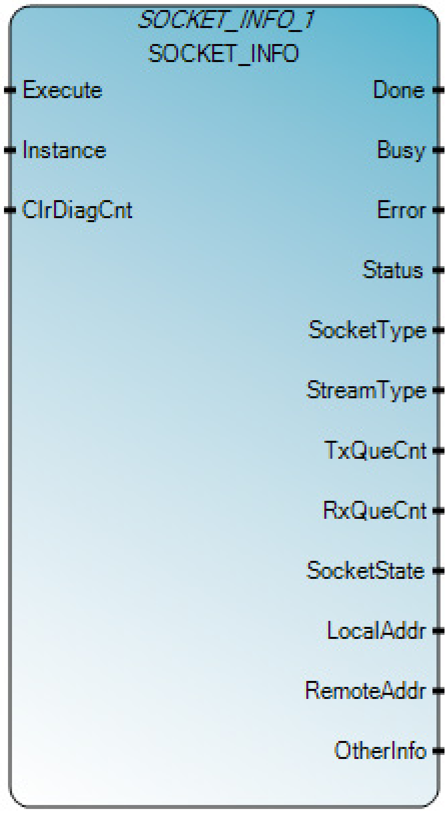
SOCKET_INFO

Returns information for a socket instance such as error codes and execution status. Outputs update synchronously from the program scan.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro850, and Micro870 controllers. For the Micro800 Simulator, this instruction can be added to a program but is only a placeholder to prevent the instruction from being deleted during controller change.

The outputs are always reset when the instruction is applied to the simulated controller (2080-LC50-48QWB-SIM).



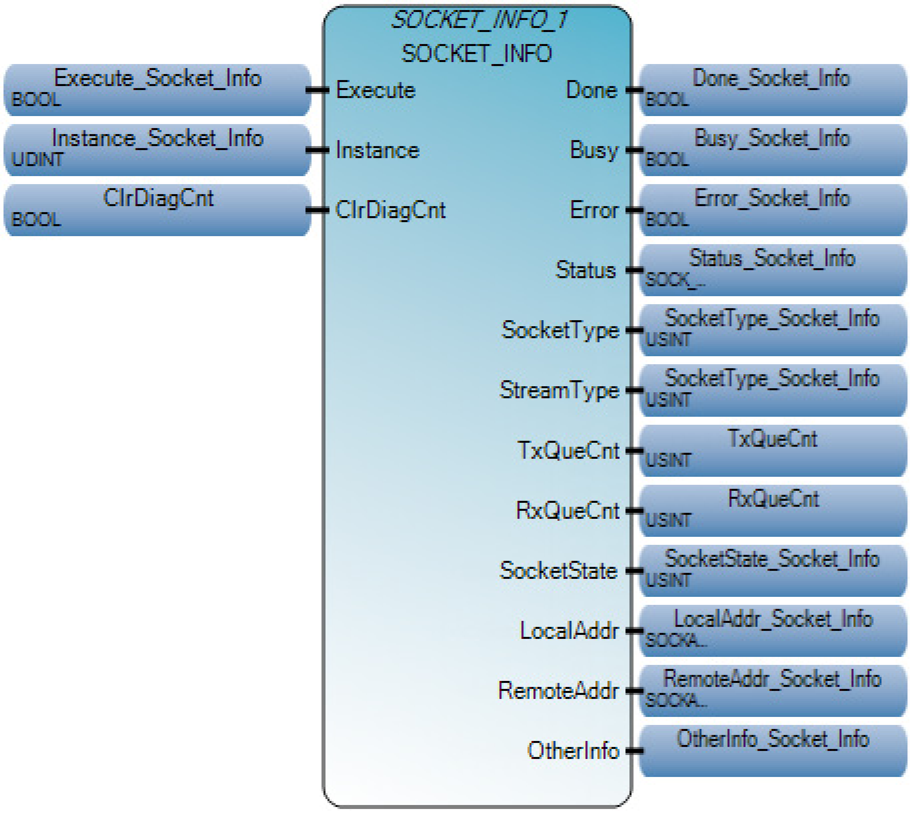
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter type	Data type	Description
Execute	Input	BOOL	Instruction block enable. TRUE - Rising Edge detected, start the instruction block with the precondition that the last operation has been completed. FALSE - no Rising Edge detected.
Instance	Input	UDINT	Copy the returned Socket Handler from a **SOCKET_CREATE** or **SOCKET_ACCEPT** instruction to delete the respective socket. • For UDP and TCP Client Socket types, copy the returned Socket Handler from a **SOCKET_CREATE** instruction. • For TCP Server socket type, copy the returned Socket Handler from a **SOCKET_ACCEPT** instruction. When Instance is 0, returns a summary of all Socket Instances.
ClrDiagCnt	Input	BOOL	TRUE - clear Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) Diagnostics counter information. FALSE - no clearing of TCP or UDP counter information. Such as TCP and UDP, **OtherInfo** Array Index 1 to 6. When Instance is 0, clear **OtherInfo** Array Index 7 to 14.
Done	Output	BOOL	Indicates when operation is complete. TRUE - operation completed successfully. FALSE - operation is in progress or encountered an error condition. Output is updated synchronously from the program scan.

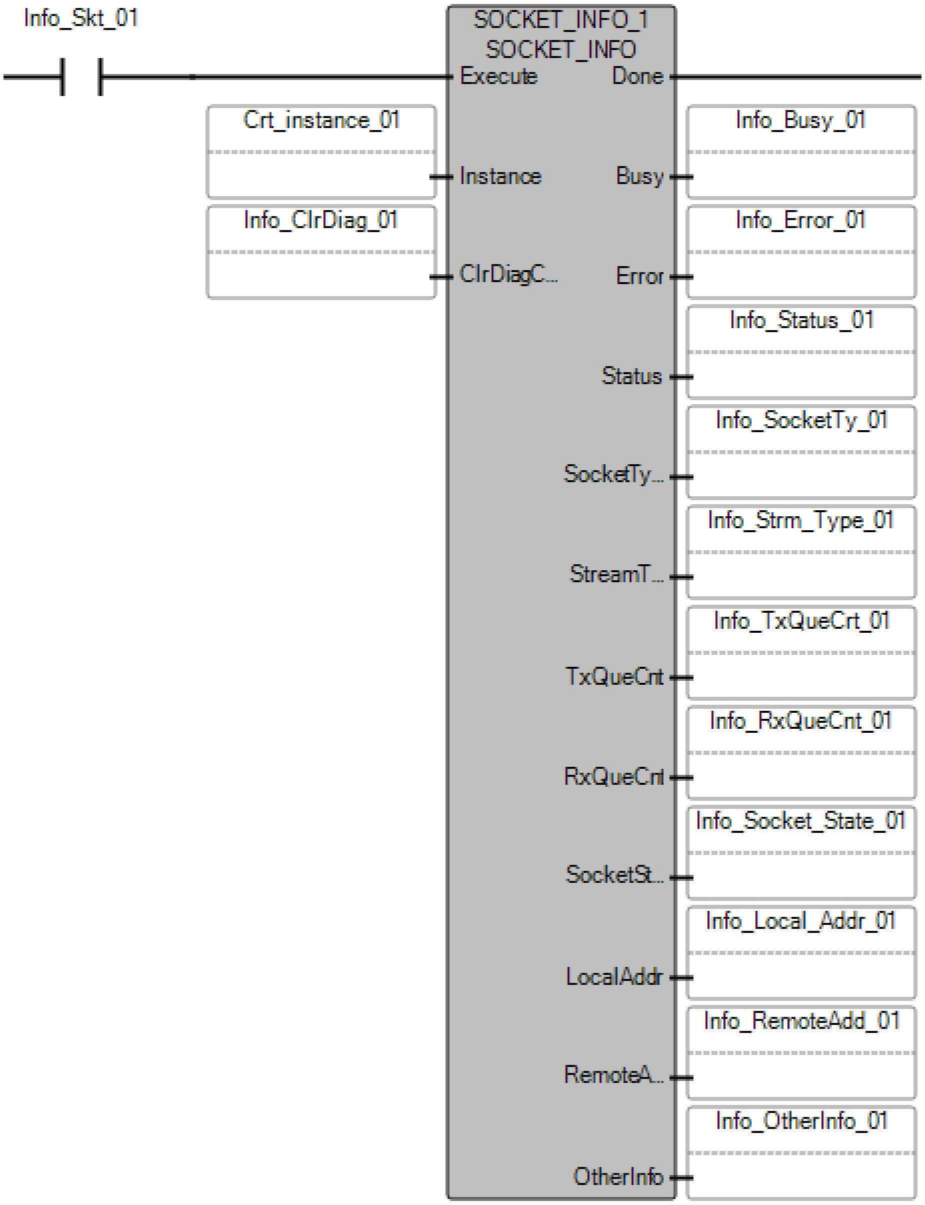
Parameter	Parameter type	Data type	Description
Busy	Output	BOOL	TRUE - the operation is not finished. FALSE - the operation is finished. Output is updated synchronously from the program scan.
Error	Output	BOOL	Indicates an error occurred. TRUE - An error is detected. FALSE - No error. Output is updated synchronously from the program scan.
Status	Output	SOCK_STATUS	Status is defined using the SOCK_STATUS data type on page 584 which contains ErrorID on page 585, SubErrorID, and StatusBits on page 588 information. Output is updated synchronously from the program scan.
SocketType	Output	USINT	Socket Instance type: • 0 - Not used • 1 - TCP • 2 - UDP When Socket_Info **Instance** is 0, **SocketType** displays as 0.
StreamType	Output	USINT	Socket Stream type: • 0 - None • 1 - TCP Server • 2 - TCP Client When Socket_Info **Instance** is 0, **StreamType** displays as 0.
TxQueCnt	Output	USINT	Number of Tx messages currently in the queue. When Socket_Info **Instance** is 0, **TxQueCnt** displays as 0.
RxQueCnt	Output	USINT	Number of Rx messages currently in the queue. When Socket_Info **Instance** is 0, **RxQueCnt** displays as 0.
SocketState	Output	USINT	Socket Instruction State information. For more information refer to Socket State Machine. When Socket_Info **Instance** is 0, **SocketState** displays as 0.
LocalAddr	Output	SOCKETADDR_CFG	Local address for the socket. For more information refer to SOCKADDR_CFG data type on page 584. When Socket_Info **Instance** is 0, **LocalAddr** displays as 0.
RemoteAddr	Output	SOCKETADDR_CFG	Remote address for the socket. For more information refer to SOCKADDR_CFG data type on page 584. **RemoteAddr** displays as 0, in the following cases: • Socket_Info **Instance** is 0 • User Datagram Protocol (UDP) connections without **SOCKET_OPEN** • UDP with **SOCKET_OPEN** and disabled **RxFilter**.

Parameter	Parameter type	Data type	Description
OtherInfo		UDINT[1..15]	Socket **Instance** configured as TCP, Array Index Description is: • 1 - Packet Sent: Total number of TCP Packets sent on a Socket. • 2 - Bytes Sent: Total number of TCP bytes sent on a Socket. • 3 - Packet Received: Total number of TCP packets received on a Socket. • 4 - Bytes Received: Total number of TCP bytes received on a Socket. • 5 - Retransmit Packets: Total number of TCP packet retransmissions. • 6 - Checksum Errors: Total number of TCP Packets with Checksum errors on a Socket. • 7 - TCP State: Current state of a Socket. • (8 to 11) - OtherInfo is not supported for TCP, displays as 0. • 12 ,13,14,15 - Displays as 0. Socket Instance configured as UDP, Array Index Description is: • 1 - Packet Sent: Total number of UDP packets sent on a Socket. • 2 - Bytes Sent: Total number of UDP bytes sent on a Socket. • 3 - Packet Received: Total number of UDP packets received on a Socket. • 4 - Bytes Received: Total number of UDP bytes received on a Socket. • 5 - Packets Dropped: Total number of UDP received packets dropped for a Socket due to exceeding the maximum queue size limit of 8. • 6 - Checksum Errors: Total number of UDP packets with checksum errors on Socket. • 7 to 15 - Display as 0. Socket Instance configured as 0, Array Index Description is: • 1 - Count for Socket Instance Available. Maximum number of sockets supported. • 2 - Count for Socket Instance Used. Number of sockets created successfully. • 3 - Number of Socket instances created as TCP. • 4 - Number of socket instances created as TCP Client. • 5 - Number of socket instances created as TCP Server. • 6 - Number of Socket instances created as UDP. • 7 - **SOCKET_READ** Success count when Socket Instance is configured as TCP. • 8 - **SOCKET_WRITE** Success count when Socket Instance is configured as TCP. • 9 - **SOCKET_READ** Failure count when Socket Instance is configured as TCP. • 10 - **SOCKET_WRITE** Failure count when Socket Instance configure as TCP. • 11 - **SOCKET_READ** Success count when Socket Instance configure as UDP. • 12 - **SOCKET_WRITE** Success count when Socket Instance configure as UDP. • 13 - **SOCKET_READ** Failure count when Socket Instance configure as UDP. • 14 - **SOCKET_WRITE** Failure count when Socket Instance configure as UDP. • 15 - Display as 0.

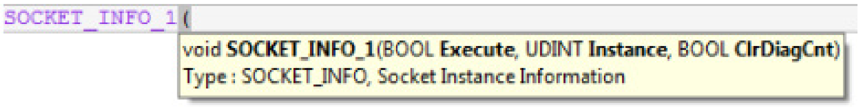
SOCKET_INFO Function Block Diagram example



SOCKET_INFO Ladder Diagram example



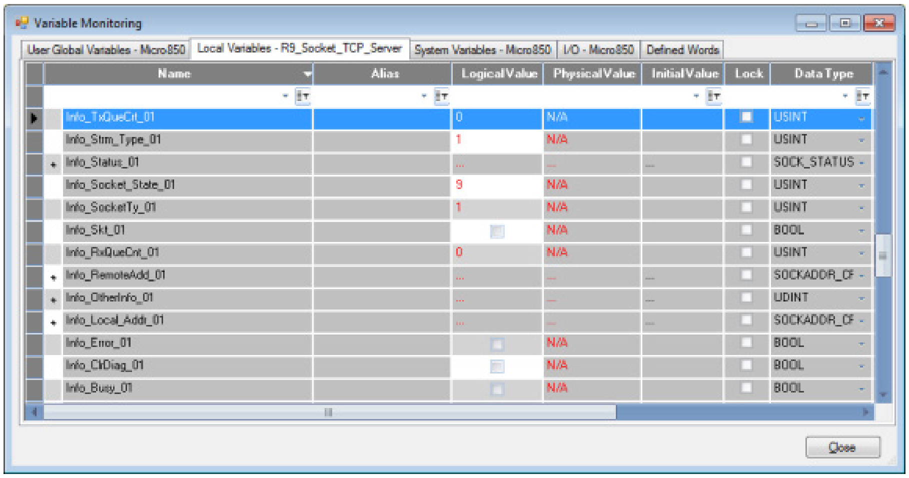
SOCKET_INFO Structured Text example



```
SOCKET_INFO_1(
void SOCKET_INFO_1(BOOL Execute, UDINT Instance, BOOL ClrDiagCnt)
Type : SOCKET_INFO, Socket Instance Information
```

```
SOCKET_INFO_1 (Execute_SOCKET_INFO, Instance_SOCKET_INFO, ClrDiagC_SOCKET_INFO);
Done_SOCKET_INFO := SOCKET_INFO. Done;
Busy_SOCKET_INFO := SOCKET_INFO. Busy;
Error_SOCKET_INFO := SOCKET_INFO. Error;
Status_SOCKET_INFO := SOCKET_INFO. Status;
SocketType_SOCKET_INFO := SOCKET_INFO. SocketTy;
StreamType_SOCKET_INFO := SOCKET_INFO. StreamType;
TxQueCnt_SOCKET_INFO := SOCKET_INFO. TxQueCnt;
RxQueCnt_SOCKET_INFO := SOCKET_INFO. RxQueCnt;
SocketState_SOCKET_INFO := SOCKET_INFO. SocketState;
LocalAddr_SOCKET_INFO := SOCKET_INFO. LocalAddr;
RemoteAddr_SOCKET_INFO := SOCKET_INFO. RemoteAddr;
OtherInfo SOCKET INFO := SOCKET INFO. OtherInfo;
```

Results



Variable Monitoring
User Global Variables - Micro850
Local Variables - R9_Socket_TCP_Server
System Variables - Micro850
I/O - Micro850
Defined Words
Close

Name	Alias	Logical Value	Physical Value	Initial Value	Lock	Data Type
Info_TxQueCnt_01		0	N/A		□	USINT
Info_Strm_Type_01		1	N/A		□	USINT
+ Info_Status_01		...	...	...	□	SOCK_STATUS
Info_Socket_State_01		9	N/A		□	USINT
Info_SocketTy_01		1	N/A		□	USINT
Info_Skt_01		□	N/A		□	BOOL
Info_RxQueCnt_01		0	N/A		□	USINT
+ Info_RemoteAdd_01		...	...	...	□	SOCKADDR_CF
+ Info_OtherInfo_01		...	...	...	□	UDINT
+ Info_Local_Addr_01		...	...	...	□	SOCKADDR_CF
Info_Error_01		□	N/A		□	BOOL
Info_ClrDiag_01		□	N/A		□	BOOL
Info_Busy_01		□	N/A		□	BOOL

SOCKET_OPEN

Opens the connection for the specified destination address for Transmission Control Protocol (TCP) connections. For User Datagram Protocol (UDP) connections, associates a destination IP address and port number with the specified socket.

Operations details:

- For User Datagram Protocol (UDP) connections using **SOCKET_OPEN** the IP address and port number are not required each time data is sent.
- For UDP connections not using **SOCKET_OPEN**, the destination address must be specified each time **SOCKET_WRITE** is used to send data.
- When using **SOCKET_READ**, in addition to receiving data, the address of the sender is returned. The senders address can be used to send a response using **SOCKET_WRITE**.
- A **SOCKET_OPEN** operation might return before the timeout period without creating a Transmission Control Protocol (TCP) connection.

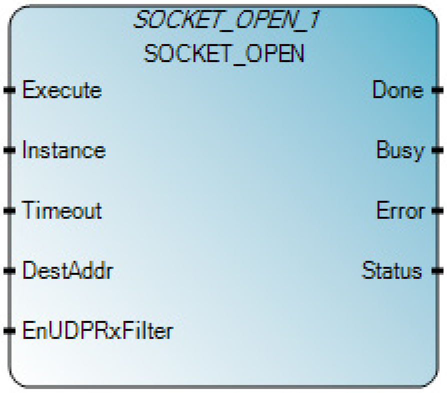
This might occur if the destination device is running but is not listening for connections on the specified port number, **SOCKET_OPEN** returns with an error before the timeout period.

- Outputs update synchronously from the program scan.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro850, and Micro870 controllers. For the Micro800 Simulator, this instruction can be added to a program but is only a placeholder to prevent the instruction from being deleted during controller change.

The outputs are always reset when the instruction is applied to the simulated controller (2080-LC50-48QWB-SIM).

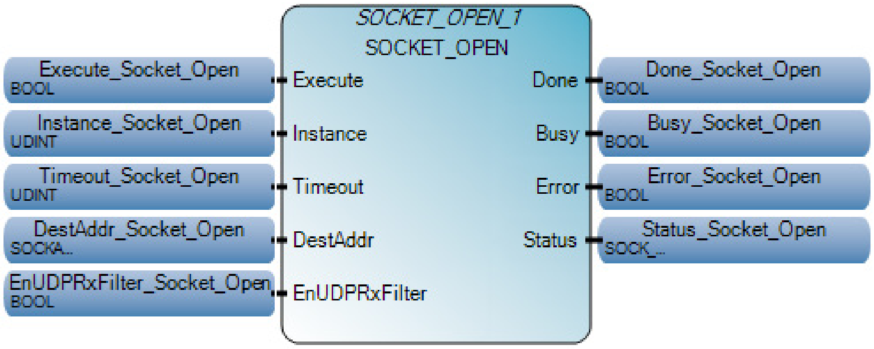


Use this table to help determine the parameter values for this instruction.

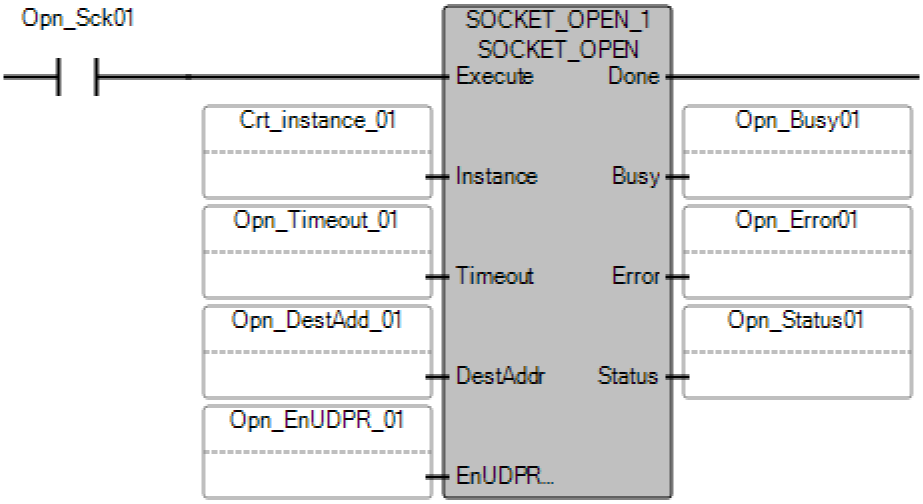
Parameter	Parameter type	Data type	Description
Execute	Input	BOOL	Instruction block enable. TRUE - Rising Edge detected, start the instruction block with the precondition that the last operation has been completed. FALSE - no Rising Edge detected.
Instance	Input	UDINT	Copy from the returned Socket Handler from a **SOCKET_CREATE** function block.
Timeout	Input	UDINT	Timeout for SOCKET_OPEN function block. The function block returns an Error if the Timeout value is less than the minimum value. Timeout range: 1000 - 1800000 milliseconds Set Timeout to 0 to use the default value 10000 (10 Second).

Parameter	Parameter type	Data type	Description
DestAddr	Input	SOCKADDR_CFG on page 584	The address of the destination connection. A connection between the IP address and the port number of remote host is required. The following IP Addresses are not supported for DestAddr: • Self IP Address • Loop back address • 0.0.0.0 • Broadcast address (Only supported for TCP), exception addresses: • Class D multicast address (224.x.x.x) • Local link address (169.254.x.x) Example for IP Address of 192.168.2.100 and Port 12000: • DestAddr.IPAddress[0]=192 • DestAddr.IPAddress[1]=168 • DestAddr.IPAddress[2]=2 • DestAddr.IPAddress[3]=100 • DestAddr.Port = 12000
EnUDPRxFilter	Input	BOOL	For UDP socket, when SOCKET_OPEN and Enable EnUDPRxFilter are used, a packet filter for specific IP Address and port number is not needed each time to read data. • A UDP socket with open is created. DestAddr as IP 192.168.1.157 / Port 161. EnUDPRxFilter Enable, perform Socket_Read. • When controller receive data from DestAddr (192.168.1.157 / 161), Socket_Read completes the operation successfully. If controller receives data from any other IP or Port then socket_Read ignores that packet and waits for DestAddr Packet. EnUDPRxFilter Disabled: • Perform SOCKET_READ. The controller receives any data on configure UDP port, SOCKET_READ completes operation successfully. • The application checks whether the incoming packet arrives from the expected device or not. • The application handles filtering based on the SOCKET_READ output parameter UDPAddr.
Done	Output	BOOL	Indicates when operation is complete. TRUE - operation completed successfully. FALSE - operation is in progress or encountered an error condition. Output is updated synchronously from the program scan.
Busy	Output	BOOL	TRUE - the operation is not finished. FALSE - the operation is finished. Output is updated synchronously from the program scan.
Error	Output	BOOL	Indicates an error occurred. TRUE - An error is detected. FALSE - No error. Output is updated synchronously from the program scan.
Status	Output	SOCK_STATUS	Status is defined using the **SOCK_STATUS** data type on page 584 which contains ErrorID on page 585, SubErrorID, and StatusBits on page 588 information. Output is updated synchronously from the program scan.

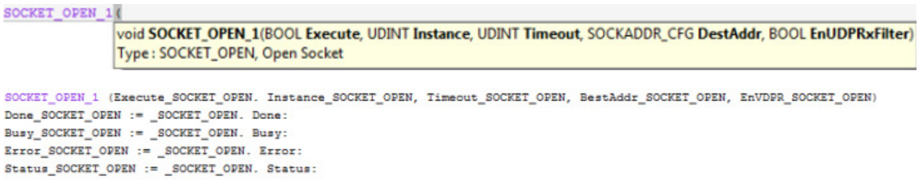
SOCKET_OPEN Function Block Diagram example



SOCKET_OPEN Ladder Diagram example



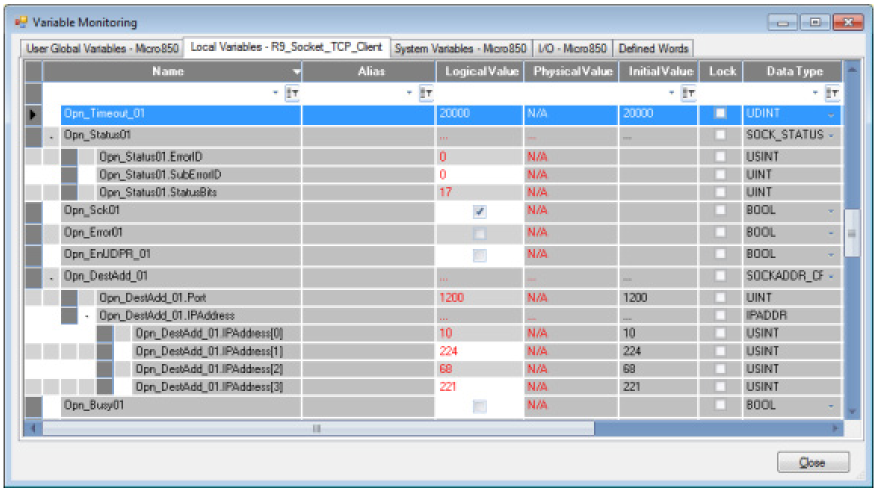
SOCKET_OPEN Structured Text example



```
SOCKET_OPEN_1(
void SOCKET_OPEN_1(BOOL Execute, UDINT Instance, UDINT Timeout, SOCKADDR_CFG DestAddr, BOOL EnUDPRxFilter)
Type : SOCKET_OPEN, Open Socket

SOCKET_OPEN_1 (Execute_SOCKET_OPEN. Instance_SOCKET_OPEN, Timeout_SOCKET_OPEN, BestAddr_SOCKET_OPEN, EnVDPR_SOCKET_OPEN)
Done_SOCKET_OPEN := _SOCKET_OPEN. Done:
Busy_SOCKET_OPEN := _SOCKET_OPEN. Busy:
Error_SOCKET_OPEN := _SOCKET_OPEN. Error:
Status_SOCKET_OPEN := _SOCKET_OPEN. Status:
```

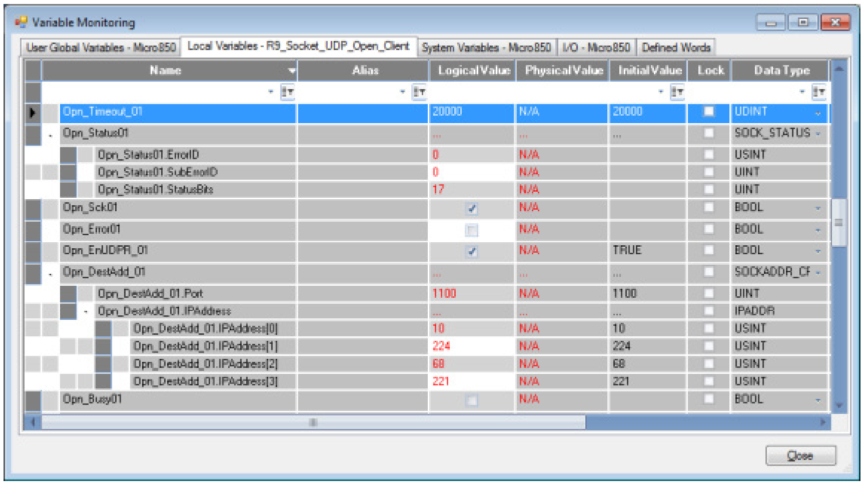
Results

TCP example


Variable Monitoring
User Global Variables - Micro850
Local Variables - R9_Socket_TCP_Client
System Variables - Micro850
I/O - Micro850
Defined Words
Close

Name	Alias	Logical Value	Physical Value	Initial Value	Lock	Data Type
Opn_Timeout_01		20000	N/A	20000	□	UDINT
Opn_Status01		...	...	...	□	SOCK_STATUS
Opn_Status01.ErrorID		0	N/A		□	USINT
Opn_Status01.SubErrorID		0	N/A		□	UINT
Opn_Status01.StatusBits		17	N/A		□	UINT
Opn_Sck01		☑	N/A		□	BOOL
Opn_Error01		□	N/A		□	BOOL
Opn_EnUDPR_01		□	N/A		□	BOOL
Opn_DestAdd_01		...	...	...	□	SOCKADDR_CF
Opn_DestAdd_01.Port		1200	N/A	1200	□	UINT
Opn_DestAdd_01.IPAddress		...	...	...	□	IPADDR
Opn_DestAdd_01.IPAddress[0]		10	N/A	10	□	USINT
Opn_DestAdd_01.IPAddress[1]		224	N/A	224	□	USINT
Opn_DestAdd_01.IPAddress[2]		68	N/A	68	□	USINT
Opn_DestAdd_01.IPAddress[3]		221	N/A	221	□	USINT
Opn_Busy01		□	N/A		□	BOOL

UDP example


Variable Monitoring
User Global Variables - Micro850
Local Variables - R9_Socket_UDP_Open_Client
System Variables - Micro850
I/O - Micro850
Defined Words
Close

Name	Alias	Logical Value	Physical Value	Initial Value	Lock	Data Type
Opn_Timeout_01		20000	N/A	20000	□	UDINT
Opn_Status01		...	...	...	□	SOCK_STATUS
Opn_Status01.ErrorID		0	N/A		□	USINT
Opn_Status01.SubErrorID		0	N/A		□	UINT
Opn_Status01.StatusBits		17	N/A		□	UINT
Opn_Sck01		☑	N/A		□	BOOL
Opn_Error01		□	N/A		□	BOOL
Opn_EnUDPR_01		☑	N/A	TRUE	□	BOOL
Opn_DestAdd_01		...	...	...	□	SOCKADDR_CF
Opn_DestAdd_01.Port		1100	N/A	1100	□	UINT
Opn_DestAdd_01.IPAddress		...	...	...	□	IPADDR
Opn_DestAdd_01.IPAddress[0]		10	N/A	10	□	USINT
Opn_DestAdd_01.IPAddress[1]		224	N/A	224	□	USINT
Opn_DestAdd_01.IPAddress[2]		68	N/A	68	□	USINT
Opn_DestAdd_01.IPAddress[3]		221	N/A	221	□	USINT
Opn_Busy01		□	N/A		□	BOOL

SOCKET_READ

Reads data on a socket and returns the specified number of bytes. For Transmission Control Protocol (TCP), returns when any data is received, up to the requested number of bytes. For User Datagram Protocol (UDP), completes when a datagram is available.

Operation details:

The following **SOCKET_READ** behavior might impact existing communications including non-socket communication.

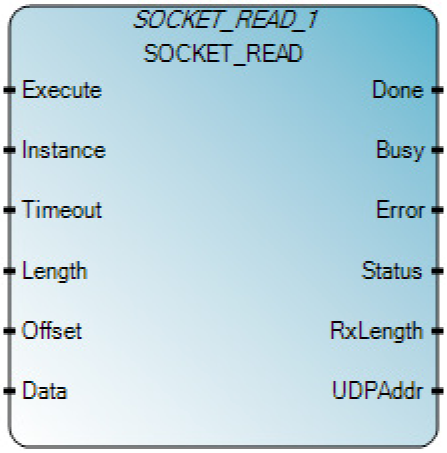
- If the **SOCKET_READ** operation is not executed in sync with the remote device then, the controller holds the remote device receive packet until one of the following occurs:
    - **SOCKET_READ** is executed.
    - Socket Timeout expires.
    - RST is received from a remote device.
    - **SOCKET_DELETE** or **SOCKET_DELETEALL** is executed.

  - User performs a **Run Mode Change** which deletes all created Socket instances.
  - Controller changes from run mode to program mode which deletes all created Socket instances.
  - Controller changes from run mode to program mode which clears socket Diagnostic counter information and individual Socket counter information.
- If the Length or Offset parameter value is changed while the SOCKET_READ operation is ongoing (BUSY = True) an error occurs and the receive packet is discarded.
- The **SOCKET_READ** instruction might return fewer bytes than were requested. RxLength contains the number of bytes of data received. Write programs to check RxLength and then issue additional read requests to receive an entire application message.
- In **Run Mode Change** mode, changing a **SOCKET_READ** input while operating in the BUSY state results in an error and the received packet is discarded.
- Outputs update synchronously from the program scan.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro850, and Micro870 controllers. For the Micro800 Simulator, this instruction can be added to a program but is only a placeholder to prevent the instruction from being deleted during controller change.

The outputs are always reset when the instruction is applied to the simulated controller (2080-LC50-48QWB-SIM).



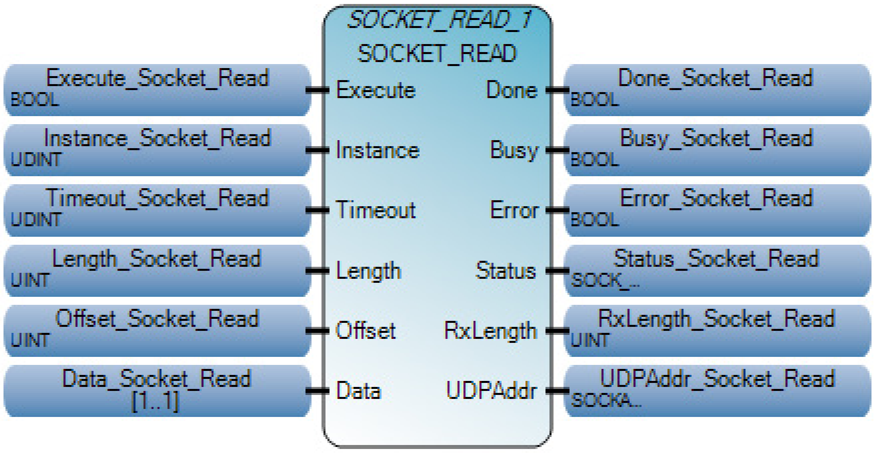
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter type	Data type	Description
Execute	Input	BOOL	Instruction enable. TRUE - Rising Edge detected, start the instruction with the precondition that the last operation is complete. FALSE - do not start instruction.

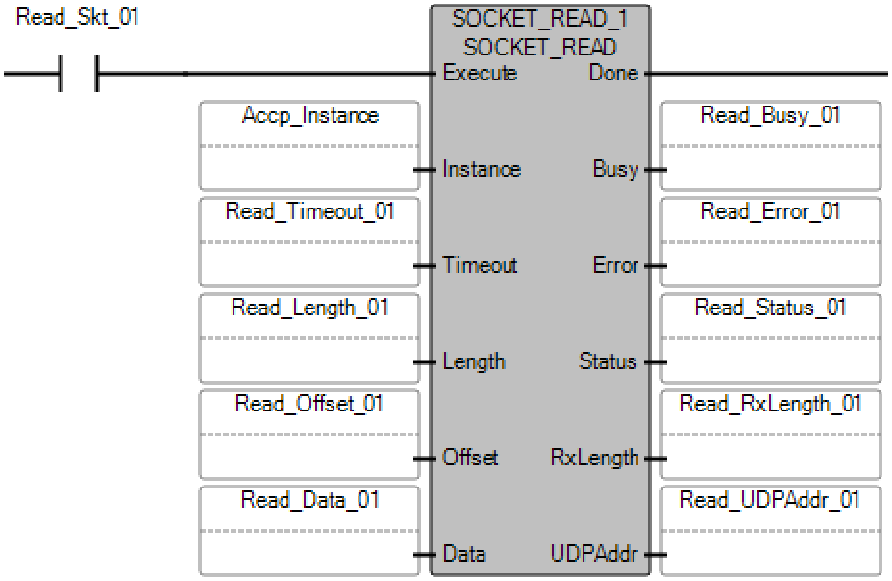
Parameter	Parameter type	Data type	Description
Instance	Input	UDINT	Copy from the returned Socket Handler from a **SOCKET_CREATE** or **SOCKET_ACCEPT** instruction. ● For UDP and TCP Client Socket types, copy the returned Socket Handler from a **SOCKET_CREATE** instruction. ● For TCP Server Socket type, copy the returned Socket Handler from a **SOCKET_ACCEPT** instruction.
Timeout	Input	UDINT	Timeout for **SOCKET_READ**. The instruction block returns an Error if the timeout value is less than the minimum value. Timeout range: 1000- 86400000 milliseconds Set **Timeout** to 0 to use the default value 10000 (10 Second).
Length	Input	UINT	Defines the number of bytes to read. Check **RxLength** for the actual number of bytes read. **SOCKET_READ** can return fewer bytes than requested. Supports up to 256 bytes.
Offset	Input	UNIT	Offset into array of **Data**. Start reading data read from this location.
Data	Output	USINT[1..1]	An Array used to store the data read from **SOCKET_READ**. ● Data array size must be >= (**Offset** + **Length**). ● Data array can be bigger than socket read Length. Output is updated synchronously from the program scan.
Done	Output	BOOL	Indicates when operation is complete. TRUE - operation completed successfully. FALSE - operation is in progress or encountered an error condition. Output is updated synchronously from the program scan.
Busy	Output	BOOL	TRUE - the operation is not finished. FALSE - the operation is finished. Output is updated synchronously from the program scan.
Error	Output	BOOL	Indicates an error occurred. TRUE - An error is detected. FALSE - No error. Output is updated synchronously from the program scan.
Status	Output	SOCK_STATUS	Status is defined using the **SOCK_STATUS** data type on page 584 which contains ErrorID on page 585, SubErrorID, and StatusBits on page 588 information. Output is updated synchronously from the program scan.
RxLength	Output	UNIT	Contains the number of data bytes received.

UDPAddr	Output	SOCKADDR_CFG on page 584	The address of the device sending User Datagram Protocol (UDP) data.
			
			Example defines a UDPAddr of 192.168.2.100 and Port 12000:
			UDPAddr.IPAddress[0]=192
			UDPAddr.IPAddress[1]=168
			UDPAddr.IPAddress[2]=2
			UDPAddr.IPAddress[3]=100
			UDPAddr.Port = 12000
			
			For Transmission Control Protocol (TCP), this structure is not used and contain all zeros. The TCP connection conveys the remote address information.

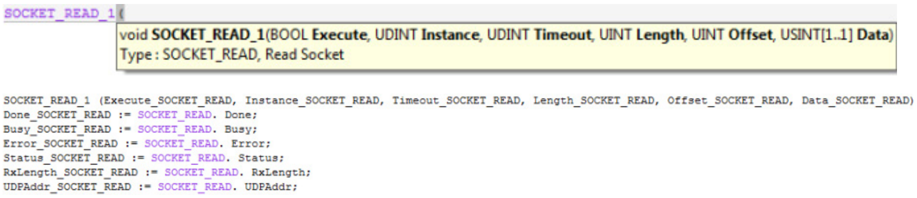
SOCKET_READ Function Block Diagram example



SOCKET_READ Ladder Diagram example



SOCKET_READ Structured Text example



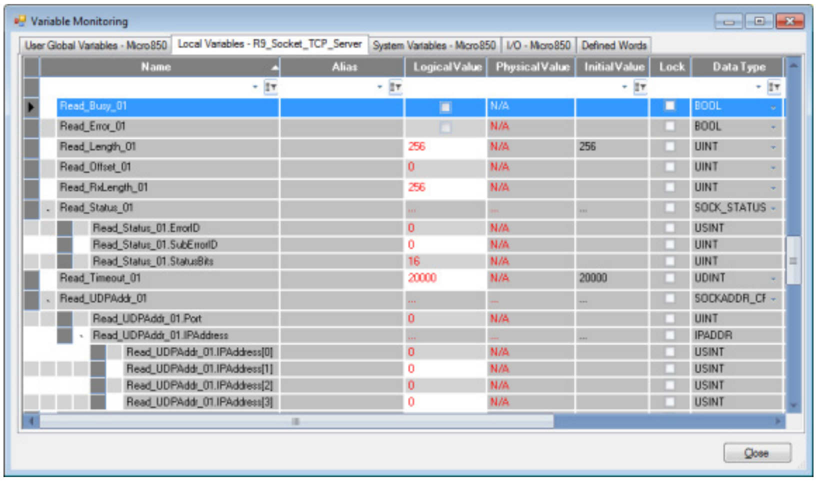
SOCKET_READ_1(

void **SOCKET_READ_1**(BOOL **Execute**, UDINT **Instance**, UDINT **Timeout**, UINT **Length**, UINT **Offset**, USINT[1..1] **Data**)
Type : SOCKET_READ, Read Socket

```
SOCKET_READ_1 (Execute_SOCKET_READ, Instance_SOCKET_READ, Timeout_SOCKET_READ, Length_SOCKET_READ, Offset_SOCKET_READ, Data_SOCKET_READ)
Done_SOCKET_READ := SOCKET_READ. Done;
Busy_SOCKET_READ := SOCKET_READ. Busy;
Error_SOCKET_READ := SOCKET_READ. Error;
Status_SOCKET_READ := SOCKET_READ. Status;
RxLength_SOCKET_READ := SOCKET_READ. RxLength;
UDPAddr_SOCKET_READ := SOCKET_READ. UDPAddr;
```

Results

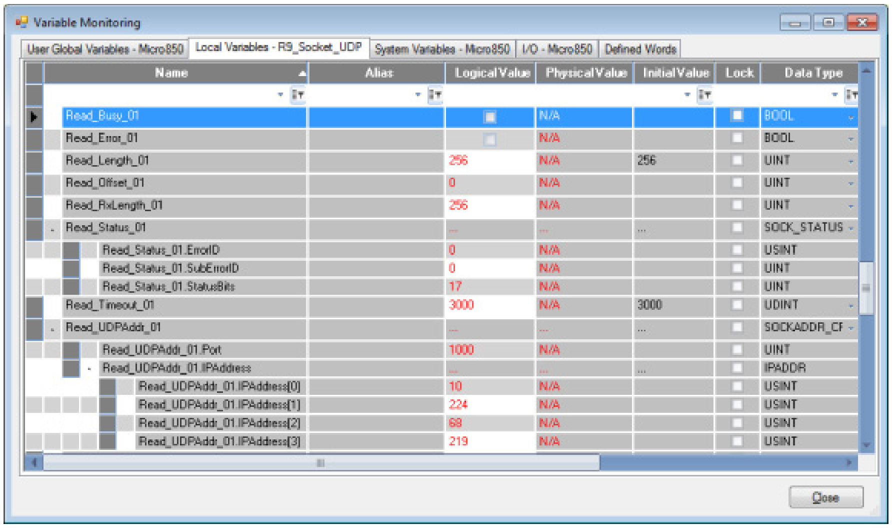
TCP example:



Variable Monitoring
User Global Variables - Micro850
Local Variables - R9_Socket_TCP_Server
System Variables - Micro850
I/O - Micro850
Defined Words
Close

Name	Alias	Logical Value	Physical Value	Initial Value	Lock	Data Type
Read_Busy_01		□	N/A		□	BOOL
Read_Error_01		□	N/A		□	BOOL
Read_Length_01		256	N/A	256	□	UINT
Read_Offset_01		0	N/A		□	UINT
Read_RxLength_01		256	N/A		□	UINT
Read_Status_01		...	...	...	□	SOCK_STATUS
Read_Status_01.ErrorID		0	N/A		□	USINT
Read_Status_01.SubErrorID		0	N/A		□	UINT
Read_Status_01.StatusBits		16	N/A		□	UINT
Read_Timeout_01		20000	N/A	20000	□	UDINT
Read_UDPAddr_01		...	...	...	□	SOCKADDR_CF
Read_UDPAddr_01.Port		0	N/A		□	UINT
Read_UDPAddr_01.IPAddress		...	...	...	□	IPADDR
Read_UDPAddr_01.IPAddress[0]		0	N/A		□	USINT
Read_UDPAddr_01.IPAddress[1]		0	N/A		□	USINT
Read_UDPAddr_01.IPAddress[2]		0	N/A		□	USINT
Read_UDPAddr_01.IPAddress[3]		0	N/A		□	USINT

UDP example:



Variable Monitoring
User Global Variables - Micro850
Local Variables - R9_Socket_UDP
System Variables - Micro850
I/O - Micro850
Defined Words
Close

Name	Alias	Logical Value	Physical Value	Initial Value	Lock	Data Type
Read_Busy_01		□	N/A		□	BOOL
Read_Error_01		□	N/A		□	BOOL
Read_Length_01		256	N/A	256	□	UINT
Read_Offset_01		0	N/A		□	UINT
Read_RxLength_01		256	N/A		□	UINT
Read_Status_01		...	...	...	□	SOCK_STATUS
Read_Status_01.ErrorID		0	N/A		□	USINT
Read_Status_01.SubErrorID		0	N/A		□	UINT
Read_Status_01.StatusBits		17	N/A		□	UINT
Read_Timeout_01		3000	N/A	3000	□	UDINT
Read_UDPAddr_01		...	...	...	□	SOCKADDR_CF
Read_UDPAddr_01.Port		1000	N/A		□	UINT
Read_UDPAddr_01.IPAddress		...	...	...	□	IPADDR
Read_UDPAddr_01.IPAddress[0]		10	N/A		□	USINT
Read_UDPAddr_01.IPAddress[1]		224	N/A		□	USINT
Read_UDPAddr_01.IPAddress[2]		68	N/A		□	USINT
Read_UDPAddr_01.IPAddress[3]		219	N/A		□	USINT

SOCKET_WRITE

Sends data on a socket.

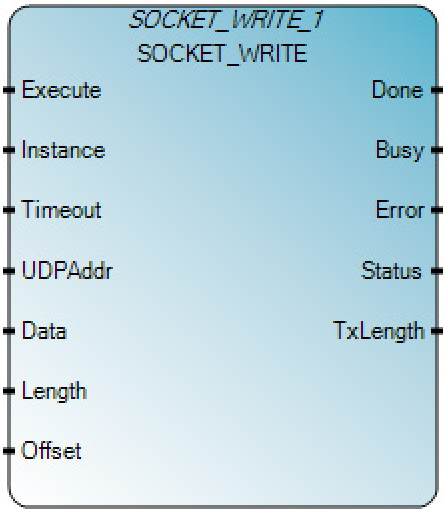
Operation details:

- **SOCKET_WRITE** attempts to send the requested number of data bytes specified in the Length parameter. When the send operation completes **SOCKET_WRITE** returns the number of data bytes written to the TxLength parameter.
- Output is updated asynchronously from the program scan.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro820, Micro850, and Micro870 controllers. For the Micro800 Simulator, this instruction can be added to a program but is only a placeholder to prevent the instruction from being deleted during controller change.

The outputs are always reset when the instruction is applied to the simulated controller (2080-LC50-48QWB-SIM).



Use this table to help determine the parameter values for this instruction.

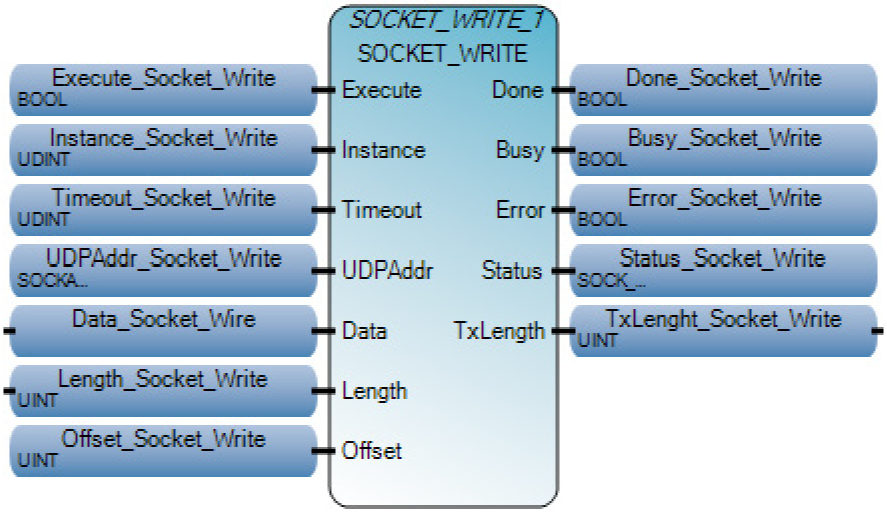
Parameter	Parameter type	Data type	Description
Execute	Input	BOOL	Instruction block enable. TRUE - Rising Edge detected, start the instruction block with the precondition that the last operation has been completed. FALSE - no Rising Edge detected.
Instance	Input	UDINT	Copy the returned Socket Handler from a **SOCKET_CREATE** or **SOCKET_ACCEPT** instruction. • For UDP and TCP-Client Socket types, copy the returned Socket Handler from a **SOCKET_CREATE** instruction. • For TCP- Server socket type, copy from returned Socket Handler from a **SOCKET_ACCEPT** instruction.
Timeout	Input	UDINT	Timeout for **SOCKET_WRITE** instances. The instruction returns an Error if the Timeout value is less than minimum value. Timeout range: 1000 to 1800000 milliseconds Set Timeout to 0 to use the default value 10000 (10 Second).

UDPAddr	Input	SOCKADDR_CFG on page 584	The UDP destination address that the data is being written to when a **SOCKET_OPEN** instruction has not been executed since creating the socket. For TCP, or when the **SOCKET_OPEN** instruction has been executed for UDP, this structure is not used and should contain all zeros. The TCP connection and the **SOCKET_OPEN** instruction for UDP convey all of the remote address information.
			Example for a UDPAddr of 192.168.2.100 and Port 12000:
			UDPAddr.IPAddress[0]=192
			UDPAddr.IPAddress[1]=168
			UDPAddr.IPAddress[2]=2
			UDPAddr.IPAddress[3]=100
			UDPAddr.Port = 12000
			Use the **SOCKADDR_CFG** data type to define UDPAddr.

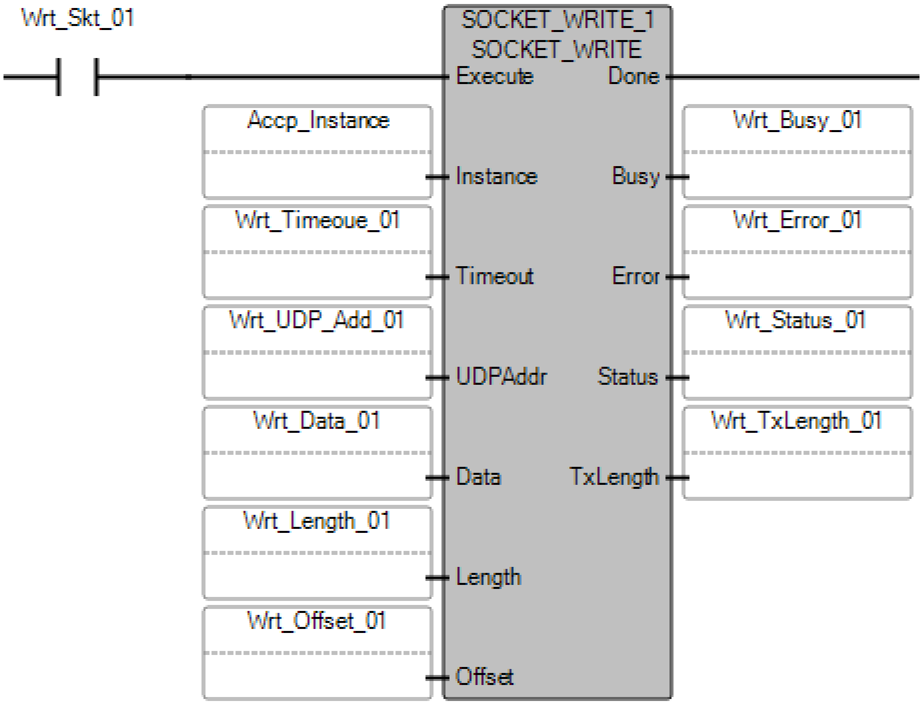
Data	Input	USINT[1..1]	An Array used to store the data write to respective socket instance using the **SOCKET_WRITE** Instruction. • Data array size must be >= (**Offset** + **Length**). • Data array can be bigger than **SOCKET_WRITE** Length.
Length	Input	UINT	The number of bytes of data to write. Maximum is 256 bytes.

Parameter	Parameter type	Data type	Description
Offset	Input	UNIT	Offset into array of Data. The data write from the SOCKET_WRITE is starting from this location.
Done	Output	BOOL	Indicates when operation is complete. TRUE - operation completed successfully. FALSE - operation is in progress or encountered an error condition. Output is updated synchronously from the program scan.
Busy	Output	BOOL	TRUE - the operation is not finished. FALSE - the operation is finished. Output is updated synchronously from the program scan.
Error	Output	BOOL	Indicates an error occurred. TRUE - An error is detected. FALSE - No error. Output is updated synchronously from the program scan.
Status	Output	SOCK_STATUS	Status is defined using the **SOCK_STATUS** data type on page 584 which contains ErrorID on page 585, SubErrorID, and StatusBits on page 588 information. Output is updated synchronously from the program scan.
TxLength	Output	UNIT	The number of bytes of written data.

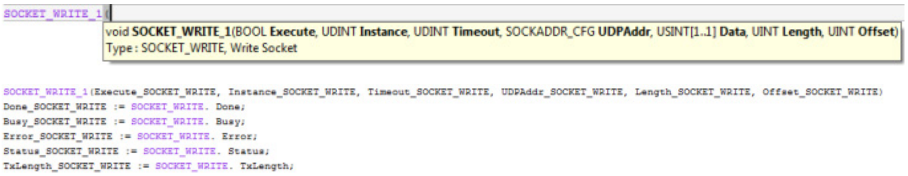
SOCKET_WRITE Function Block Diagram example



SOCKET_WRITE Ladder Diagram example



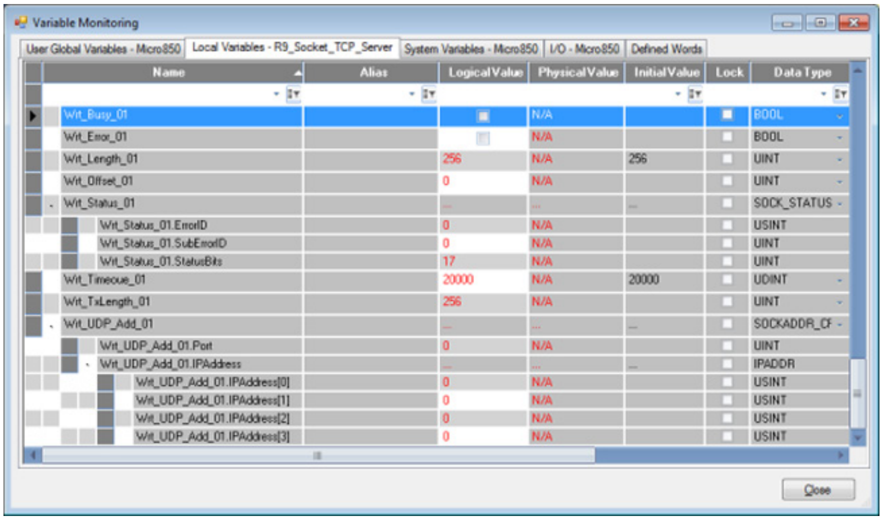
SOCKET_WRITE Structured Text example



```
SOCKET_WRITE_1(
void SOCKET_WRITE_1(BOOL Execute, UDINT Instance, UDINT Timeout, SOCKADDR_CFG UDPAddr, USINT[1..1] Data, UINT Length, UINT Offset)
Type : SOCKET_WRITE, Write Socket

SOCKET_WRITE_1(Execute_SOCKET_WRITE, Instance_SOCKET_WRITE, Timeout_SOCKET_WRITE, UDPAddr_SOCKET_WRITE, Length_SOCKET_WRITE, Offset_SOCKET_WRITE)
Done_SOCKET_WRITE := SOCKET_WRITE. Done;
Busy_SOCKET_WRITE := SOCKET_WRITE. Busy;
Error_SOCKET_WRITE := SOCKET_WRITE. Error;
Status_SOCKET_WRITE := SOCKET_WRITE. Status;
TxLength_SOCKET_WRITE := SOCKET_WRITE. TxLength;
```

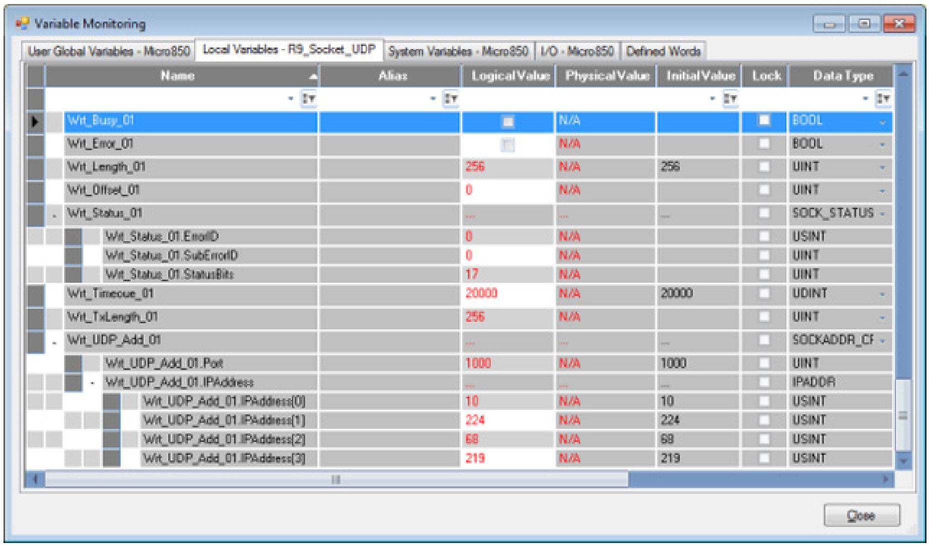
Results

TCP example:


Variable Monitoring
User Global Variables - Micro850
Local Variables - R9_Socket_TCP_Server
System Variables - Micro850
I/O - Micro850
Defined Words
Close

Name	Alias	Logical Value	Physical Value	Initial Value	Lock	Data Type
Wrt_Busy_01		☐	N/A		☐	BOOL
Wrt_Error_01		☐	N/A		☐	BOOL
Wrt_Length_01		256	N/A	256	☐	UINT
Wrt_Offset_01		0	N/A		☐	UINT
Wrt_Status_01		...	...	...	☐	SOCK_STATUS
Wrt_Status_01.ErrorID		0	N/A		☐	USINT
Wrt_Status_01.SubErrorID		0	N/A		☐	UINT
Wrt_Status_01.StatusBits		17	N/A		☐	UINT
Wrt_Timeoue_01		20000	N/A	20000	☐	UDINT
Wrt_TxLength_01		256	N/A		☐	UINT
Wrt_UDP_Add_01		...	...	...	☐	SOCKADDR_CF
Wrt_UDP_Add_01.Port		0	N/A		☐	UINT
Wrt_UDP_Add_01.IPAddress		...	...	...	☐	IPADDR
Wrt_UDP_Add_01.IPAddress[0]		0	N/A		☐	USINT
Wrt_UDP_Add_01.IPAddress[1]		0	N/A		☐	USINT
Wrt_UDP_Add_01.IPAddress[2]		0	N/A		☐	USINT
Wrt_UDP_Add_01.IPAddress[3]		0	N/A		☐	USINT

UDP without open example:

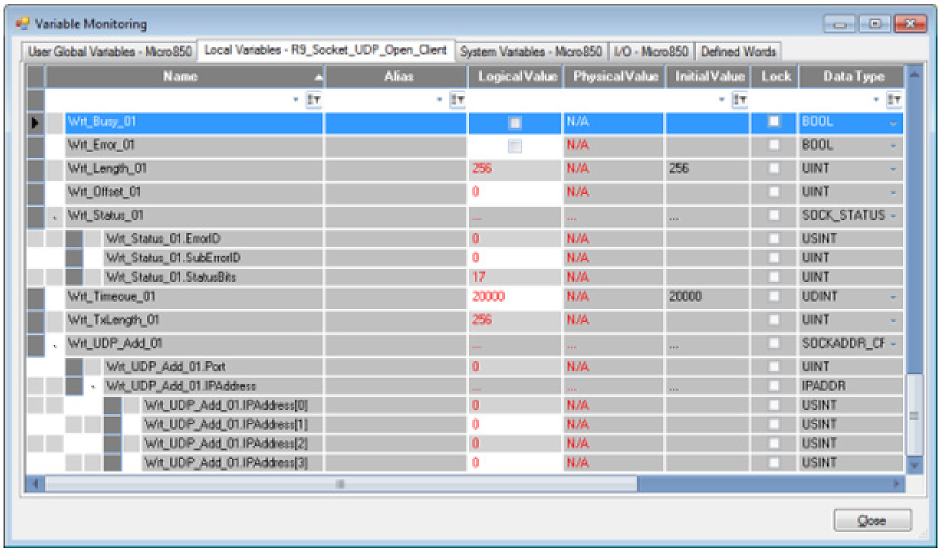


Variable Monitoring

User Global Variables - Micro850 | Local Variables - R9_Socket_UDP | System Variables - Micro850 | I/O - Micro850 | Defined Words

Name	Alias	Logical Value	Physical Value	Initial Value	Lock	Data Type
Wrt_Busy_01		☐	N/A		☐	BOOL
Wrt_Error_01		☐	N/A		☐	BOOL
Wrt_Length_01		256	N/A	256	☐	UINT
Wrt_Offset_01		0	N/A		☐	UINT
Wrt_Status_01		...	...	...	☐	SOCK_STATUS
Wrt_Status_01.ErrorID		0	N/A		☐	USINT
Wrt_Status_01.SubErrorID		0	N/A		☐	UINT
Wrt_Status_01.StatusBits		17	N/A		☐	UINT
Wrt_Timeoue_01		20000	N/A	20000	☐	UDINT
Wrt_TxLength_01		256	N/A		☐	UINT
Wrt_UDP_Add_01		...	...	...	☐	SOCKADDR_CF
Wrt_UDP_Add_01.Port		1000	N/A	1000	☐	UINT
Wrt_UDP_Add_01.IPAddress		...	...	...	☐	IPADDR
Wrt_UDP_Add_01.IPAddress[0]		10	N/A	10	☐	USINT
Wrt_UDP_Add_01.IPAddress[1]		224	N/A	224	☐	USINT
Wrt_UDP_Add_01.IPAddress[2]		68	N/A	68	☐	USINT
Wrt_UDP_Add_01.IPAddress[3]		219	N/A	219	☐	USINT

UDP with open example:



Variable Monitoring

User Global Variables - Micro850 | Local Variables - R9_Socket_UDP_Open_Client | System Variables - Micro850 | I/O - Micro850 | Defined Words

Name	Alias	Logical Value	Physical Value	Initial Value	Lock	Data Type
Wrt_Busy_01		☐	N/A		☐	BOOL
Wrt_Error_01		☐	N/A		☐	BOOL
Wrt_Length_01		256	N/A	256	☐	UINT
Wrt_Offset_01		0	N/A		☐	UINT
Wrt_Status_01		...	...	...	☐	SOCK_STATUS
Wrt_Status_01.ErrorID		0	N/A		☐	USINT
Wrt_Status_01.SubErrorID		0	N/A		☐	UINT
Wrt_Status_01.StatusBits		17	N/A		☐	UINT
Wrt_Timeoue_01		20000	N/A	20000	☐	UDINT
Wrt_TxLength_01		256	N/A		☐	UINT
Wrt_UDP_Add_01		...	...	...	☐	SOCKADDR_CF
Wrt_UDP_Add_01.Port		0	N/A		☐	UINT
Wrt_UDP_Add_01.IPAddress		...	...	...	☐	IPADDR
Wrt_UDP_Add_01.IPAddress[0]		0	N/A		☐	USINT
Wrt_UDP_Add_01.IPAddress[1]		0	N/A		☐	USINT
Wrt_UDP_Add_01.IPAddress[2]		0	N/A		☐	USINT
Wrt_UDP_Add_01.IPAddress[3]		0	N/A		☐	USINT

SOCKADDR_CFG data type

The following table describes the **SOCKADDR_CFG** data type.

Parameter	Data type	Description
Port	UINT	Specify a local port number on which an application is listening and receiving.
IP Address[4]	USINT	Specify an IP address. Example for an IP Address of 192.168.2.100: • `IPAddress[0]=192` • `IPAddress[1]=168` • `IPAddress[2]=2` • `IPAddress[3]=100`

SOCK_STATUS data type

The following table describes the **SOCK_STATUS** data type.

Parameter	Data type	Description
ErrorID	USINT	Error Code on page 585 Value.
SubErrorID	UINT	Sub Error Code Value.
StatusBits	UINT	Execution Status bits for Socket instructions on page 588.

Socket error codes

The following table describes the status error codes for Socket instructions.

ErrorID code	SubErrorID code	Error description	Corrective action
0		The Socket instruction successfully completed operation.	
1		The Socket instruction is pending.	
2		The Socket Instance is not available.	Confirm the Socket Instance is not deleted or the Timeout value exceeded.
3		The **SOCKET_DELETEALL** operation is ongoing.	Wait for the pending **SOCKET_DELETEALL** operation to complete.
4	1	Illegal Parameter, Invalid Channel.	No action. Reserved for future use.
4	2	Illegal Parameter, Invalid Socket IP Address. • Error occurs. **SOCKET_CREATE** contains a **SockAddr** with any non zero value. • **SOCKET_OPEN** or **SOCKET_WRITE** execute with an invalid target IP Address. Invalid target IP Addresses are: • Self IP Address • 0.0.0.0 • Loop back Address (127.x.x.x) • Class D Multicast Address (224.x.x.x) • Local Link Address (169.254.x.x) • Broadcast Address (Only applicable for TCP Socket Instance)	Change to a valid IP Address.
4	3	Illegal Parameter, Invalid Socket Port Address. Error occurs when the following Ports are specified in **SOCKET_CREATE**, **SOCKET_OPEN**, **or SOCKET_WRITE**: • TCP Ports • 44818 - EtherNet/IP • 502 -ModbusTCP • 67 - DHCP Server • 68 - DHCP Client • 0 - Invalid Port • UDP Ports • 2222 – EtherNet/IP • 67 – DHCP Server • 68 – DHCP Client • 0 – Invalid Port	Change the Port Address.
4	4	Illegal Parameter, Invalid Socket Type. Valid Socket Types are: • TCP - 1 • UDP - 2	Change Socket Type.

ErrorID code	SubErrorID code	Error description	Corrective action
4	5	Illegal Parameter, Invalid Socket Timeout value. Valid Timeout values are: • SOCKET_CREATE, SOCKET_ACCEPT, and SOCKET_READ: • 0 (Default), or any value between 1000 and 86400000ms. • SOCKET_OPEN and SOCKET_WRITE: • 0 (Default), or any value between 1000 and 86400000ms.	Change the Timeout value to a valid value.
4	6	Illegal Parameter, Invalid Socket Instance. • TCP Server Socket Type: • Use the Socket Instance returned in SOCKET_ACCEPT for SOCKET_WRITE, SOCKET_READ, SOCKET_DELETE, and SOCKET_INFO. • UDP Socket and TCP Client Socket Types: • Use the Socket Instance returned in SOCKET_CREATE for SOCKET_OPEN, SOCKET_WRITE, SOCKET_READ, SOCKET_DELETE, and SOCKET_INFO.	Use the Instance Number returned in SOCKET_CREATE and SOCKET_ACCEPT instructions after successful execution.
4	7	Illegal Parameter, Invalid Array Length.	Increase the size of the array used to contain the Socket read and write data.
4	8	Illegal Parameter, Invalid Array Dimension.	Use single dimensional array to contain the Socket read and write data.
5		Socket request to cancel operation. Error occurs when: • Ethernet Link is Disabled or Ethernet Cable is Disconnected • SOCKET_DELETE operation is performed when SOCKET_OPEN, SOCKET_ACCEPT, SOCKET_READ, or SOCKET_WRITE operation are in progress. • Run Mode Change operation is performed when SOCKET_OPEN, SOCKET_ACCEPT, SOCKET_READ, or SOCKET_WRITE operation are in progress. • IP Address Collision detected when SOCKET_OPEN, SOCKET_ACCEPT, SOCKET_READ, or SOCKET_WRITE operation are in progress.	Restart the Socket operation based on Socket Type. Refer to the respective State Machine to restart Socket operation.
6	1	Illegal Socket Sequence, Socket Open Operation in progress or Connected. Error occurs when: • SOCKET_OPEN is in progress (BUSY State) and user executes another SOCKET_OPEN instance with same socket instance. • SOCKET_OPEN is in progress (BUSY State) and user executes SOCKET_ACCEPT with same socket instance.	Use single SOCKET_OPEN execution for respective Socket Instance. Do not perform SOCKET_ACCEPT operation for same socket instance.
6	2	Illegal Socket Sequence, Socket Accept Operation In progress or Connected. Error occurs when: • SOCKET_ACCEPT is in progress (BUSY State) and user executes another SOCKET_ACCEPT instance with same socket instance. • SOCKET_ACCEPT is in progress (BUSY State) and user executes SOCKET_OPEN with same socket instance.	Use single SOCKET_ACCEPT execution for respective Socket Instance. Do not perform SOCKET_OPEN operation for same socket instance.
6	3	Illegal Socket Sequence, Socket configured as TCP Client.	
6	4	Illegal Socket Sequence, Socket configured as TCP Server.	
6	5	Illegal Socket Sequence, Socket is Connected.	Make sure the Socket is not already connected with a remote device before executing SOCKET_OPEN or SOCKET_ACCEPT again.

ErrorID code	SubErrorID code	Error description	Corrective action
6	6	Illegal Socket Sequence, Socket configured as UDP.	Make sure UDP Socket Instances are not used with SOCKET_ACCEPT.
6	7	Illegal Socket Sequence, Socket is not connected.	Confirm that Socket is connected with the target.
7		Socket Instance Timeout.	Configure Timeout values for SOCKET_ACCEPT, SOCKET_OPEN, SOCKET_READ, and SOCKET_WRITE accordingly.
8		The Socket module is not initialized.	Make sure the Ethernet Link is Enabled or Ethernet Cable of Controller is connected to the Network or resolve Controller IP Address conflict.
9	1	Socket Instruction Fatal Error, Socket Instance Missing.	
9	2	Socket Instruction Fatal Error, Invalid Socket Instance.	
9	3	Socket Instruction Fatal Error, Invalid Lock Socket Instance.	
9	4	Socket Instruction Fatal Error, Invalid Socket Type.	
9	5	Socket Instruction Fatal Error, Missing Cancel Handler.	
10	1	Socket background processing Error, Address in Use.	
10	2	Socket background processing Error, UDP Received a Large Packet and the Packet is larger than Socket Read length.	Make sure the SOCKET_READ Length is equal to greater than the received packet size. Max read size is recommended. Max size for SOCKET_READ is 256.
10	3	Socket background processing Error, TCP Receive Large packet.Packet received is larger than the Socket Read length.	Make sure the SOCKET_READ Length is equal to greater than the received packet size. Since client may send up to the receive window size of 256, 256 is recommended.
10	4	Socket background processing Error, Received RST or Disconnect from Remote Device.	Restart Controller TCP Client or Server as shown in the State Diagram. Verify the Target device that sent RST and make sure the Target is restarted with correct state.
10	5	Socket background processing Error, UDP Packet received from different device, drop packet. Reserved for future.	
10	6	Socket background processing Error, Queue is full. Error occurs when: • SOCKET_READ or SOCKET_WRITE execute four instructions for the same Socket instance in same scan. • Perform a SOCKET_READ when four SOCKET_READ instructions are in BUSY state for same Socket instance.	Wait for SOCKET_READ and SOCKET_WRITE queue availability for respective Socket instance.
10	7	Socket background processing Error, SOCKET_READ Parameter change (Length, Offset, Data Array Size, and Data Array Variable).	When the SOCKET_READ state is BUSY, do not modify the input parameters.
11	1	Socket background processing Fatal Error, Missing data pointer.	
11	2	Socket background processing Fatal Error, Missing Session Pointer.	
11	3	Socket background processing Fatal Error, Invalid TCP or UDP Socket pointer.	
11	4	Socket background processing Fatal Error, Invalid Socket Instance Type.	
11	5	Socket background processing Fatal Error, Socket Instance Missing.	
11	6	Socket background processing Fatal Error, Invalid Socket Instance.	
11	7	Socket background processing Fatal Error, Invalid Socket State.	
11	8	Socket background processing Fatal Error, Invalid Socket Type.	
11	9	Socket background processing Fatal Error, TCP delete Failure.	
11	10	Socket background processing Fatal Error, UDP delete Failure.	
128	1	No packet available for disconnect message.	
128	2	Not enough room to pre-pend the TCP header.	

ErrorID code	SubErrorID code	Error description	Corrective action
128	3	Packet append pointer is invalid.	
128	7	Invalid Socket pointer.	
128	10	Invalid type-of-service, fragment, or time-to-live option.	
128	17	Invalid caller for this service.	
128	18	Packet is not valid.	
128	20	This component has not been enabled.	
128	21	This component has already been enabled.	
128	22	Listening is not enabled for the specified port.	
128	26	Requested suspension was aborted.	
128	33	Invalid server IP Address.	
128	34	This socket is bound to another port.	
128	35	Port is bound to a different socket.	
128	36	Socket is not bound.	
128	38	The Socket was unbound while suspended waiting for a receive packet.	
128	39	Socket was not created.	
128	51	No additional listen request structures are available.	
128	52	There is already an active listen request for this port.	
128	53	Socket is not in a closed state.	
128	54	The server socket supplied is not in a listen state.	
128	55	No wait time was specified, the connection attempt is in progress.	
128	56	Connection failed.	
128	57	Request is greater than the receiver's advertised window size in bytes.	
128	64	Another thread is suspended. Only one thread is allowed.	
128	65	Disconnect failed to complete within the Timeout period.	
128	66	Socket is bound.	
128	69	No available Port.	
128	70	Invalid Port.	
128	71	There is already a valid Socket pointer for this port or the specified port does not have an active listen request.	
128	72	Same as NX_SUCCESS, except a queued connection request was processed during this call.	
128	73	TCP Transmit Queue Exceeded Error.	

Socket instruction status bits

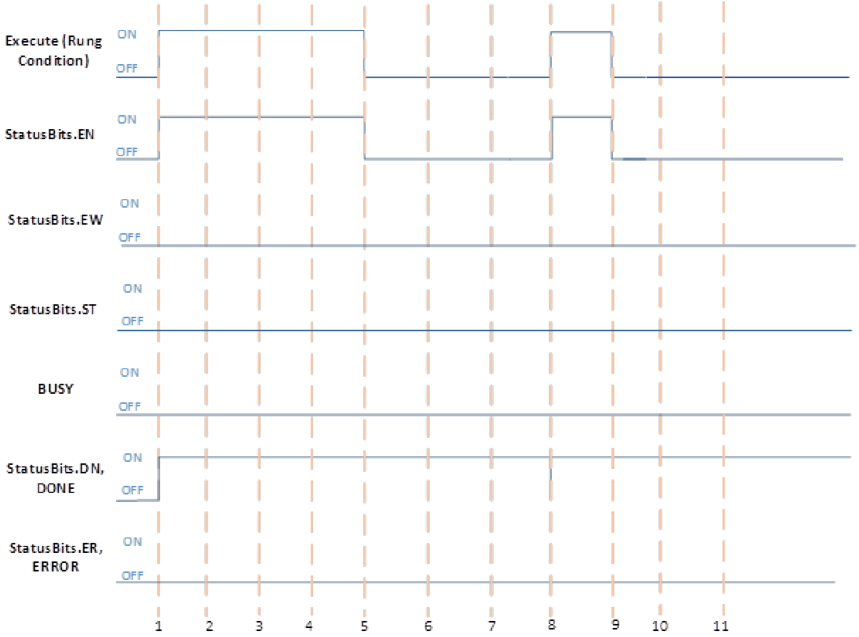
The following table describes the execution status bits for socket instructions on page 553.

Bit Number	Name	Description
0	EN - Enable Bit	The EN bit is set when the instruction is enabled due to a False-to-True transition, but has not yet completed or erred.
1	EW - Enable Wait Queue Bit	The EW bit is set when the controller detects that a Socket instruction request has entered into queue. The controller resets the EW bit when the ST bit is set.
2	ST - Start Bit	The ST bit is set when the queued instruction is executing. The ST bit is reset when the DN or ER bit is set.

Bit Number	Name	Description
3	ER - Error Bit	Indicates that an error occurred while executing the instruction. The ER bit is reset the next time the run-condition-in goes from False to True.
4	DN - Done Bit	The DN bit is set when the Socket instruction completes successfully. The DN bit is reset the next time the run-condition-in goes from False to True.

Socket instruction timing diagrams

Successful execution for Socket instructions on page 553 when process is immediate



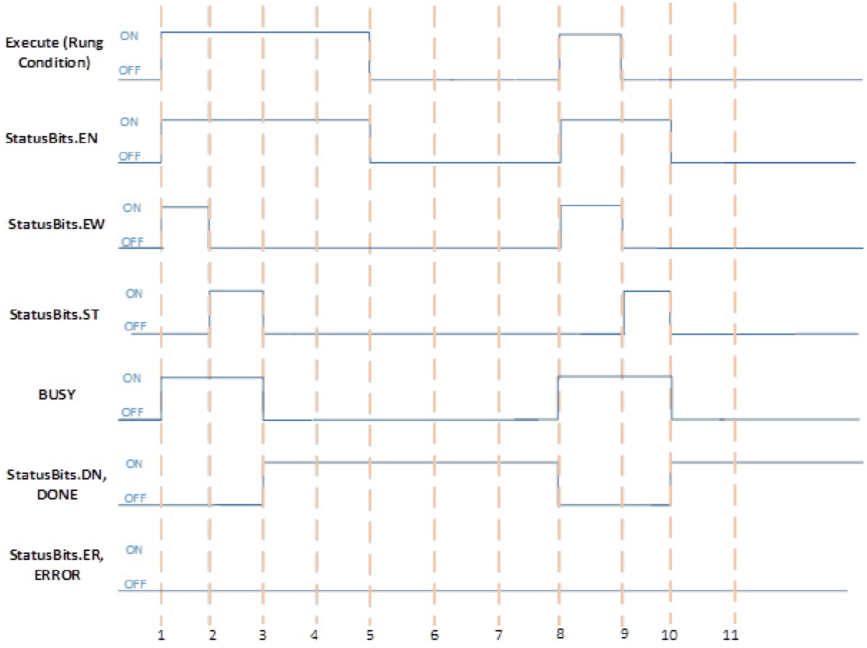
Condition A: Rung condition is TRUE during instructions execution

Rung Condition	Description
1	Rung condition becomes TRUE when: • Socket instructions execution is enabled. • EN bit is set and all other bits are cleared. • Socket instructions completes execution successfully. • Output for DN bit and DONE is set.
2,3,4	No change in Rung condition.
5	Rung condition becomes FALSE when EN bit is cleared.
6,7	No change in Rung condition.

Condition B: Rung condition is FALSE during instructions execution

Rung Condition	Description
8	Rung condition becomes TRUE when: • Socket instructions execution is enabled. • Socket instructions completes execution successfully. • EN bit is set and all other bits are cleared. • Output for DN bit and DONE is set.
9	Rung condition becomes FALSE when: • EN bit is cleared.
10,11	No change in Rung condition.

Successful execution for Socket instructions when process is non-immediate



Condition A: Rung condition is TRUE during instructions execution

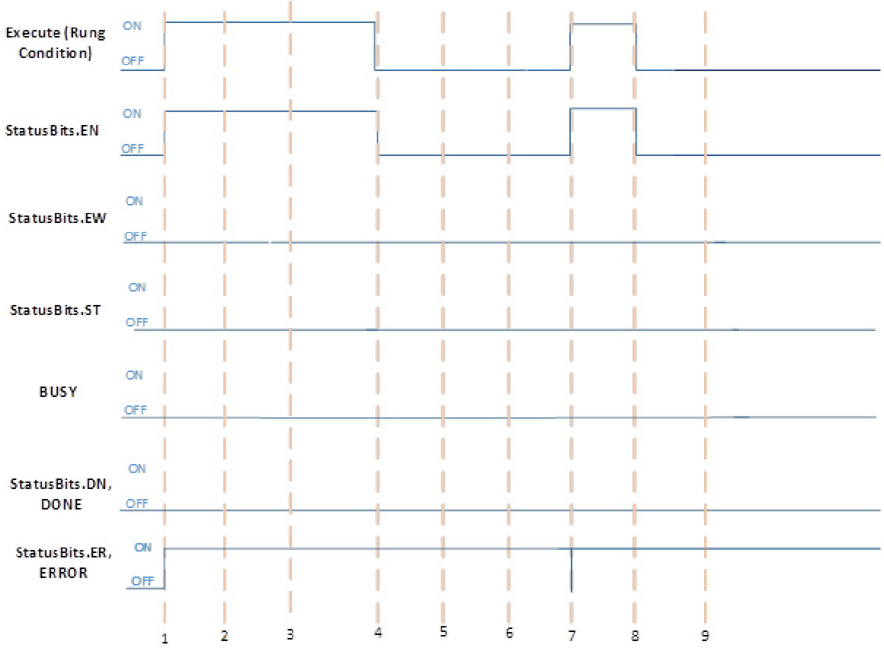
Rung Condition	Description
1	Rung condition becomes TRUE when: • Socket instructions execution is enabled. • EN bit is set and all other bits are cleared. • Socket instructions perform background processing. • Output for EW bit and BUSY is set. • Locked Socket instructions input parameter for background processing.

Rung Condition	Description
2	Socket instructions start execution when: • EW bit is cleared. • ST bit is set.
3	Socket instructions complete execution successfully when: • Output for ST bit and BUSY is cleared. • Output for DN bit and DONE is set.
4	No change in Rung condition.
5	Rung condition becomes FALSE when: • EN bit is cleared.
6,7	No change in Rung condition.

Condition B: Rung condition goes FALSE during instructions execution

Rung Condition	Description
8	Rung condition becomes TRUE when: • Socket instructions execution is enabled. • EN bit is set and all other bits are cleared. • Socket instructions send for background processing. • Output for EW bit and BUSY is set. • Locked Socket instructions input parameter for background processing.
9	Socket instructions start execution when: • EW bit is cleared. • ST bit is set. • Rung condition becomes FALSE.
10	Socket instructions complete execution successfully when: • Output for ST bit and BUSY are cleared. • Output for DN bit and DONE is set. • Rung condition is FALSE. • EN bit is cleared.

Socket instructions fail when EN is TRUE and EW and ST are FALSE



Condition A: Rung condition is TRUE during instructions execution

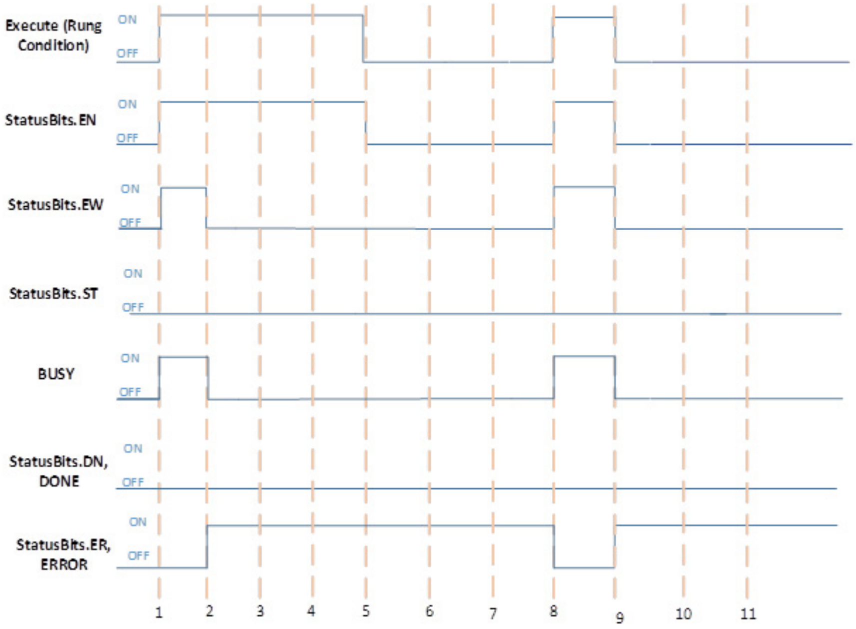
Rung Condition	Description
1	Rung condition becomes TRUE when: • Socket instructions execution is enabled. • EN bit is set and all other bits are cleared. • Socket instructions complete execution with error. • Output for ER bit and error is set.
2,3	No change in Rung condition.
4	Rung condition becomes FALSE when: • EN bit is cleared.
5,6	No change in Rung condition.

Condition B: Rung condition becomes FALSE during instructions execution

Rung Condition	Description
7	Rung condition becomes TRUE when: • Socket instructions execution is enabled. • EN bit is set and all other bits are cleared. • Socket instructions complete execution with error. • Output for ER bit and error is set.

Rung Condition	Description
8	Rung condition becomes FALSE when: • EN bit is cleared.
9	No change in Rung condition.

Socket instructions fail when EW is TRUE and instruction process is non-immediate



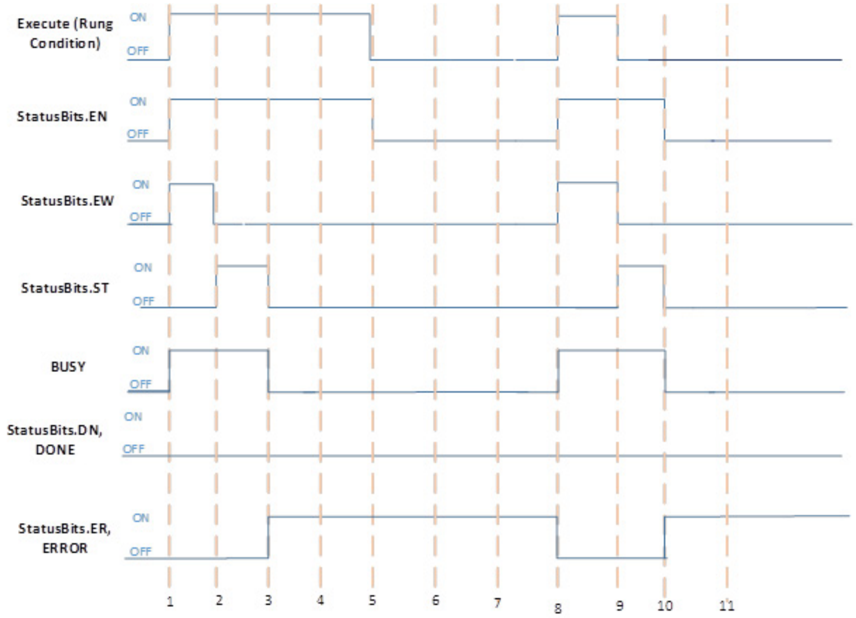
Condition A: Rung condition is TRUE during instructions execution

Rung Condition	Description
1	Rung condition becomes TRUE when: • Socket instructions execution is enabled. • EN bit is set and all other bits are cleared. • Socket instructions send for background processing. • Output for EW bit and BUSY is set. • Locked Socket instructions input parameter for background processing.
2	Socket instructions complete execution with error when: • Output for EW bit and BUSY is cleared. • Output for ER bit and error is set.
3,4	No change in Rung condition.
5	Rung condition becomes FALSE when: • EN bit is cleared.
6,7	No change in Rung condition.

Condition B: Rung condition becomes FALSE during instructions execution

Rung Condition	Description
8	Rung condition becomes TRUE when: • Socket instructions execution is enabled. • EN bit is set and all other bits are cleared. • Socket instructions send for background processing. • Output for EW bit and BUSY is set. • Locked Socket instructions input parameter for background processing.
9	Socket instructions complete execution with error when: • Output for EW bit and BUSY is cleared. • Output for ER bit and error is set. • Rung condition becomes FALSE. • EN bit is cleared.
10,11	No change in Rung condition.

Socket instructions fail when ST is TRUE and instruction process is non-immediately



Condition A: Rung condition is TRUE during instructions execution

Rung Condition	Description

Rung Condition	Description
1	Rung condition becomes TRUE when: • Socket instructions execution is enabled. • EN bit is set and all other bits are cleared. • Socket instructions send for background processing. • Output for EW bit and BUSY output is set. • Locked Socket instructions input parameter for background processing.
2	Socket instructions start execution when: • EW bit is cleared • ST bit is set.
3	Socket instructions complete execution with error when: • Output for ST bit and BUSY output is cleared. • Output for ER bit and error is set.
4	No change in Rung condition.
5	Rung condition becomes FALSE when: • EN bit is cleared.
6,7	No change in Rung condition.

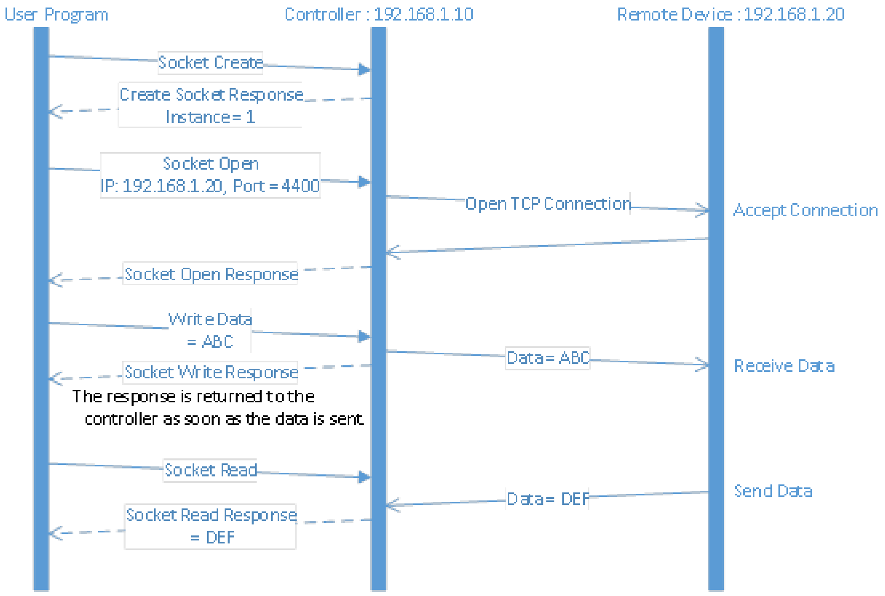
Condition B: Rung condition becomes FALSE during instructions execution

Rung Condition	Description
8	Rung condition becomes TRUE when: • Socket instructions execution is enabled. • EN bit is set and all other bits are cleared. • Socket instructions send for background processing. • Output for EW bit and BUSY is set. • Locked Socket instructions input parameter for background processing.
9	Socket instructions start execution when: • EW bit is cleared and ST bit is set.
10	Socket instructions complete execution with error when: • Output for ST bit and BUSY is cleared. • Output for ER bit and error is set. • Rung condition becomes FALSE. • EN bit is cleared.
11	No change in Rung condition.

Socket instruction transaction diagrams

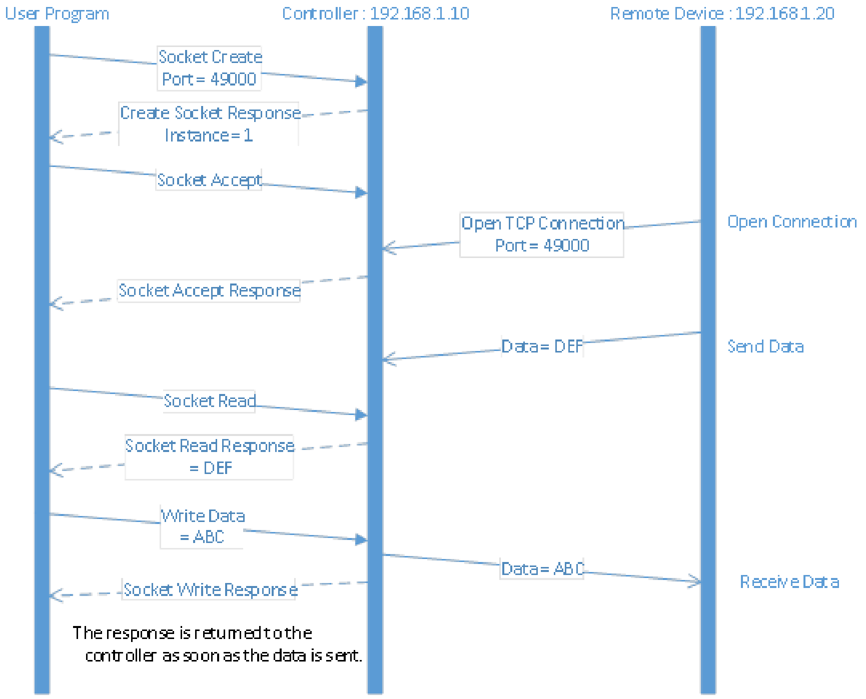
Transactions for TCP Client

The following diagram shows a typical sequence of socket interface transactions with the controller acting as a TCP client.



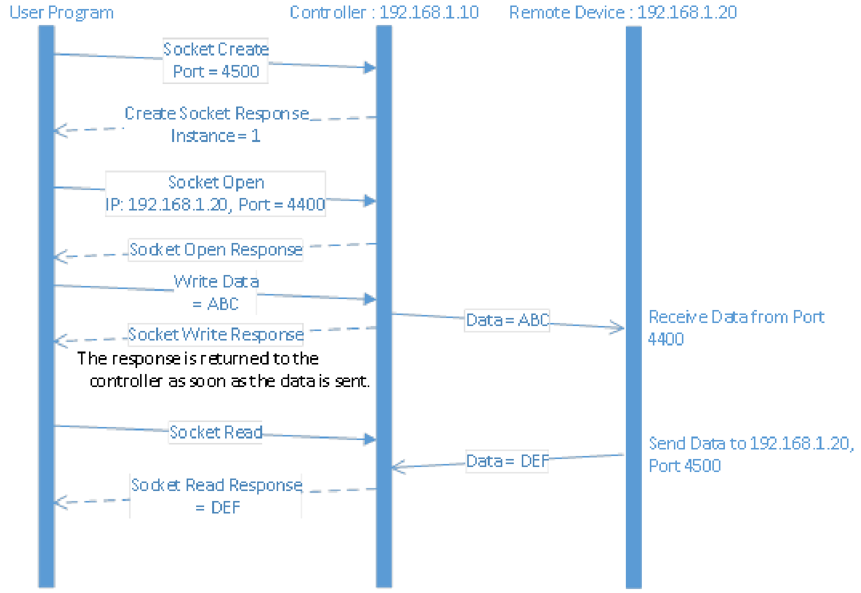
Transactions for TCP Server

The following diagram shows a typical sequence of socket interface transactions with the controller as a TCP server.



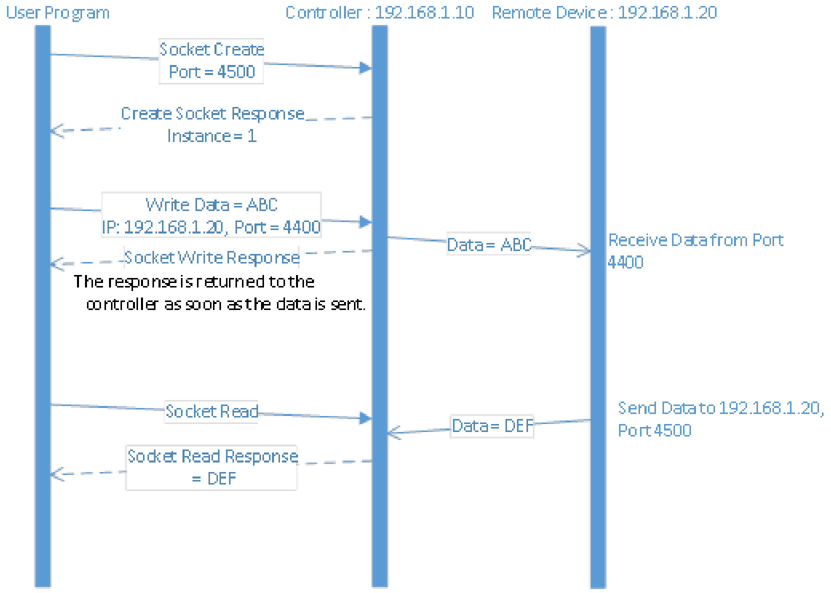
Transactions for UDP with Open Connection

The following diagram shows a typical sequence of socket interface transactions for UDP communications when using the Open Connection service to specify the destination address.



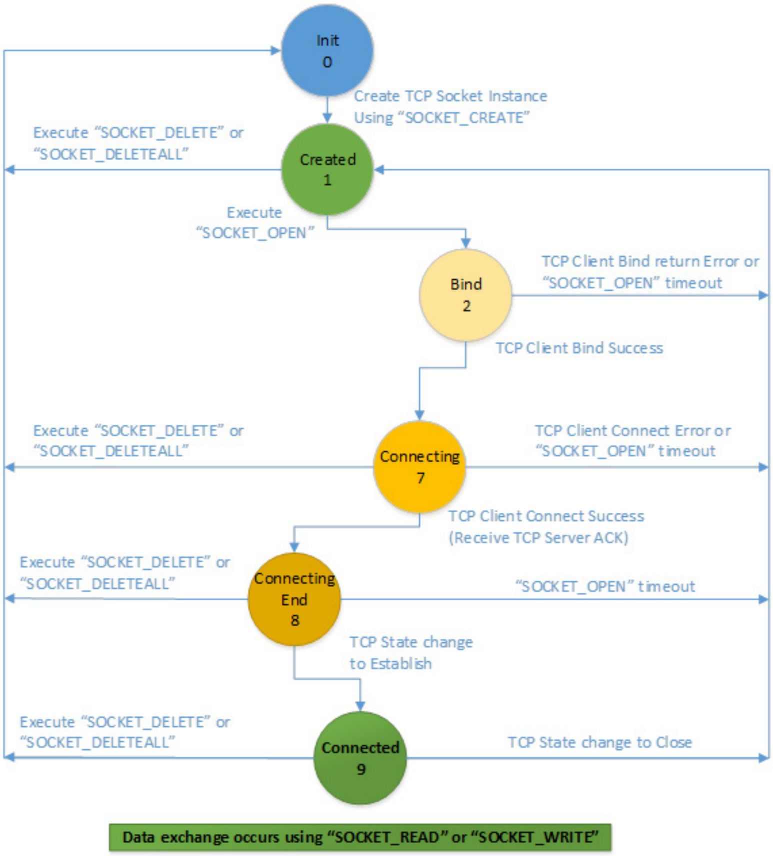
Transactions for UDP without Open Connections

The following diagram shows a typical sequence of socket interface transactions for UDP communications without using the Open Connection service to specify the destination address. In this case, the controller specifies the destination for each datagram and receives the sender's address along with each datagram it receives.

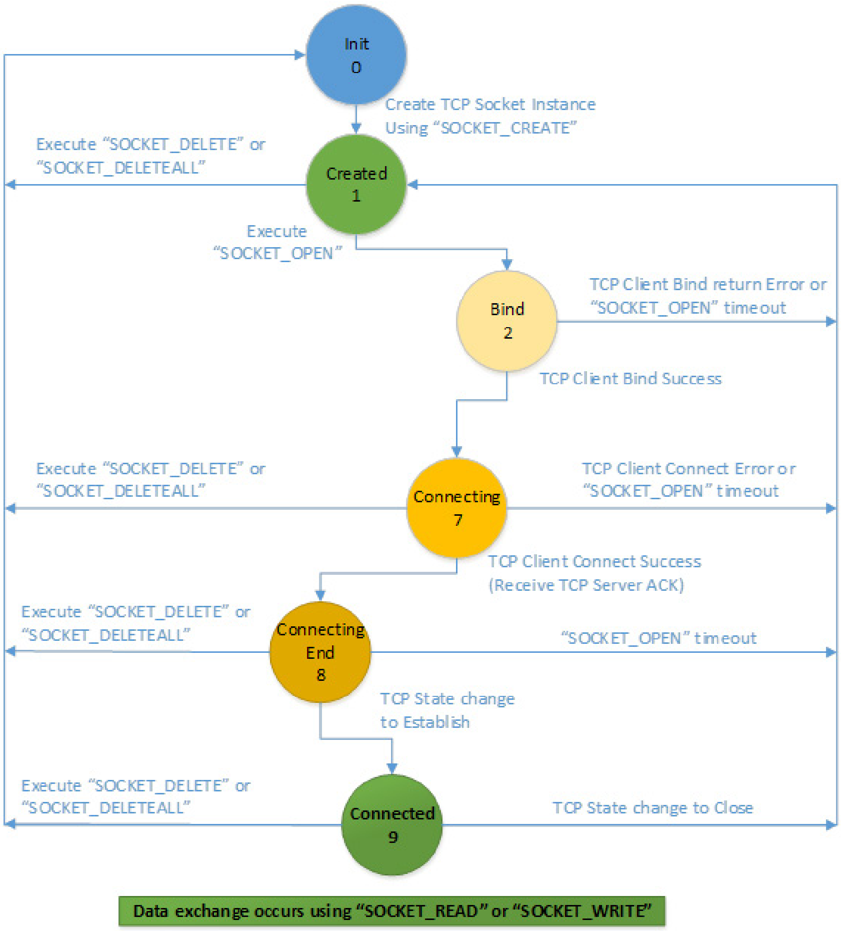


State machine diagrams for TCP

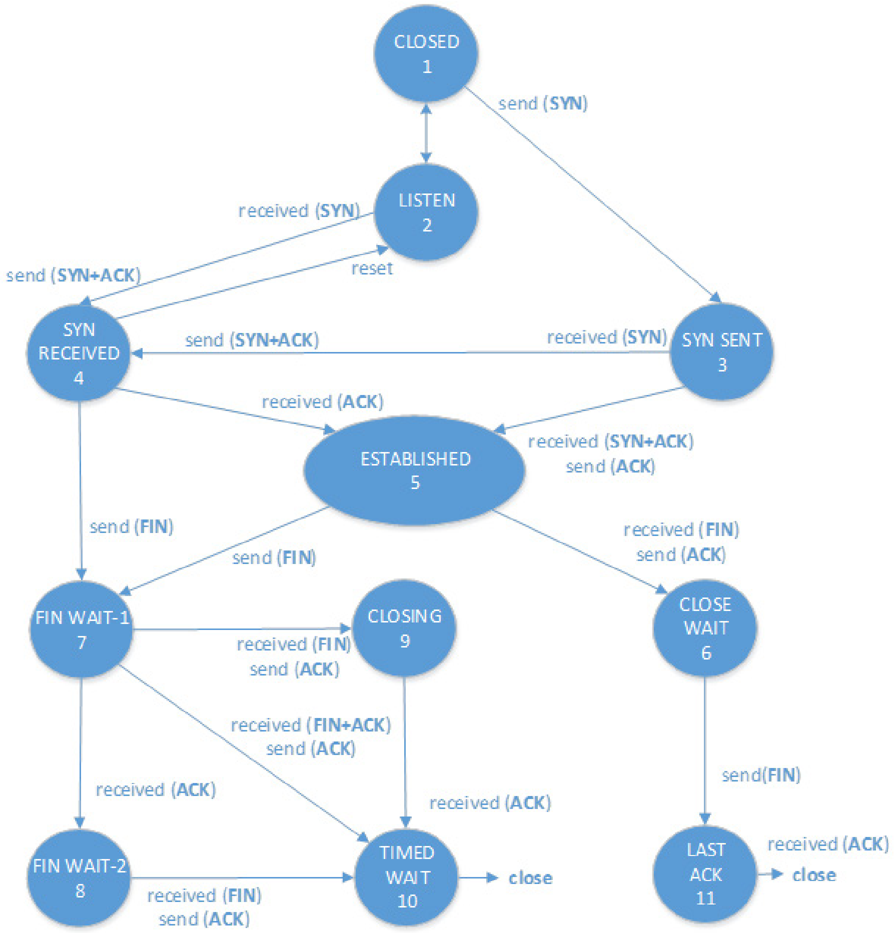
TCP Client using Socket instructions on page 553


Init
0
Create TCP Socket Instance
Using "SOCKET_CREATE"
Execute "SOCKET_DELETE" or
"SOCKET_DELETEALL"
Created
1
Execute
"SOCKET_OPEN"
Bind
2
TCP Client Bind return Error or
"SOCKET_OPEN" timeout
TCP Client Bind Success
Execute "SOCKET_DELETE" or
"SOCKET_DELETEALL"
Connecting
7
TCP Client Connect Error or
"SOCKET_OPEN" timeout
TCP Client Connect Success
(Receive TCP Server ACK)
Execute "SOCKET_DELETE" or
"SOCKET_DELETEALL"
Connecting
End
8
"SOCKET_OPEN" timeout
TCP State change
to Establish
Execute "SOCKET_DELETE" or
"SOCKET_DELETEALL"
Connected
9
TCP State change to Close
Data exchange occurs using "SOCKET_READ" or "SOCKET_WRITE"

TCP Server using Socket instructions

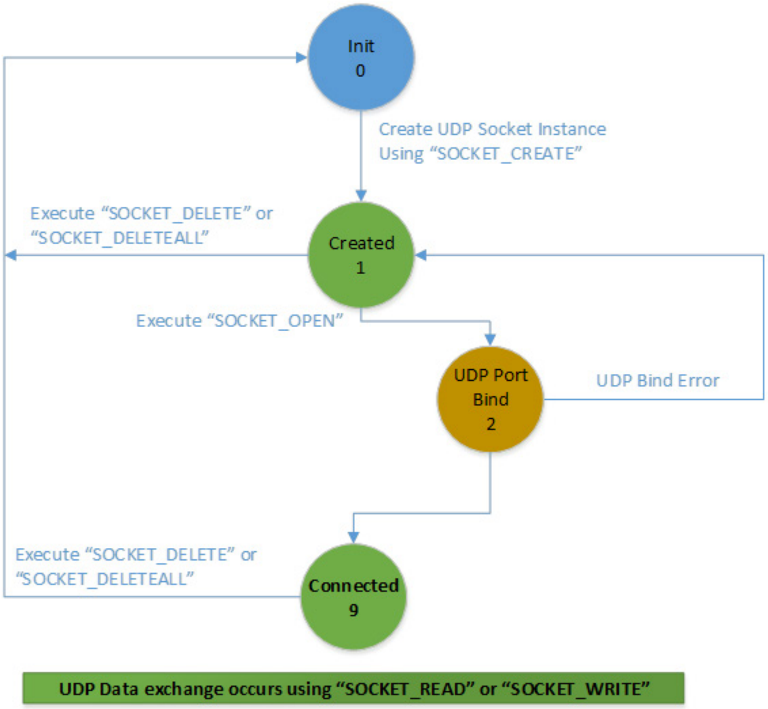


TCP State Diagram

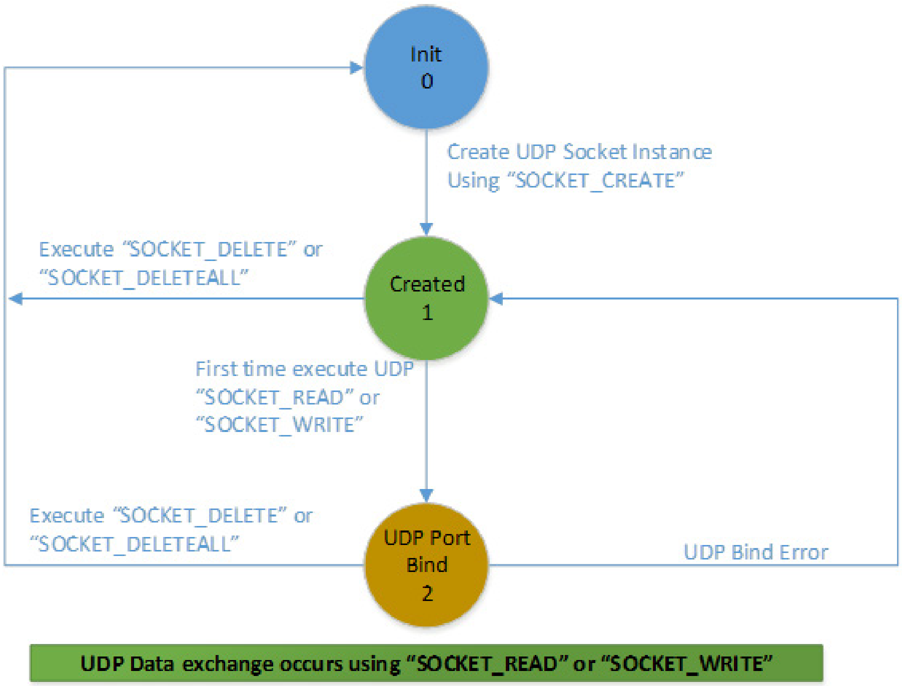


State machine diagrams for UDP

UDP Datagram with SOCKET_OPEN instruction


Init
0
Create UDP Socket Instance
Using "SOCKET_CREATE"
Execute "SOCKET_DELETE" or
"SOCKET_DELETEALL"
Created
1
Execute "SOCKET_OPEN"
UDP Port
Bind
2
UDP Bind Error
Execute "SOCKET_DELETE" or
"SOCKET_DELETEALL"
Connected
9
UDP Data exchange occurs using "SOCKET_READ" or "SOCKET_WRITE"

UDP Datagram without SOCKET_OPEN instruction



String manipulation instructions

Use String manipulation instructions to alter a sequence of symbols that are chosen from a set or alphabet to change the output status. To read input strings containing special characters correctly, input the string characters after the program containing the function block instance is online.

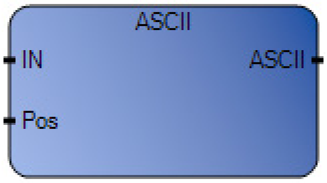
Instruction	Description
ASCII on page 603	Returns the ASCII code for characters in a string. Character -> ASCII code.
CHAR on page 605	Returns a one character string for an ASCII code. ASCII code -> character.
DELETE on page 606	Deletes characters from a string.
FIND on page 608	Locates and provides the position of sub-strings within strings.
INSERT on page 610	Inserts sub-strings at user-defined positions within strings.
LEFT on page 612	Extracts characters from the left side of a string.
MID on page 614	Extracts characters from the middle of a string.
MLEN on page 616	Calculates the length of a string.
REPLACE on page 619	Replaces parts of a string with new sets of characters.
RIGHT on page 617	Extracts characters from the right side of a string.

ASCII

Returns the ASCII code for characters in strings.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.

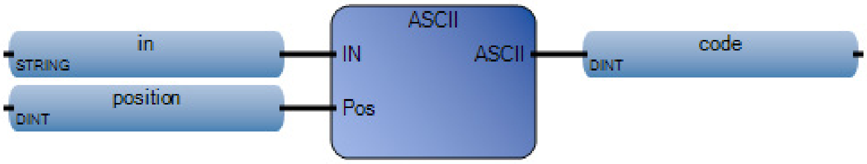


Use this table to help determine the parameter values for this instruction.

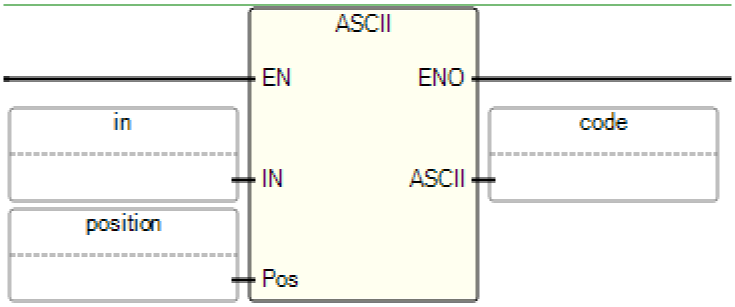
Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - display the ASCII code for characters. FALSE - no display operation. Applies to Ladder Diagram programs.
IN	Input	STRING	Any non-empty string.
Pos	Input	DINT	Position of the selected character in set [1.. len] (len is the length of the IN string).

Parameter	Parameter Type	Data Type	Description
ASCII	Output	DINT	ASCII code of the selected character (in set [0 .. 255]) yields 0 is Pos is out of the string.
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

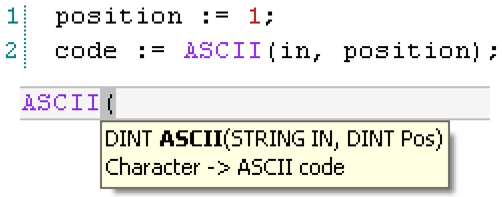
ASCII Function Block Diagram example



ASCII Ladder Diagram example



ASCII Structured Text example



```
position := 1;
code := ASCII(in, position);
```

```
ASCII(
```

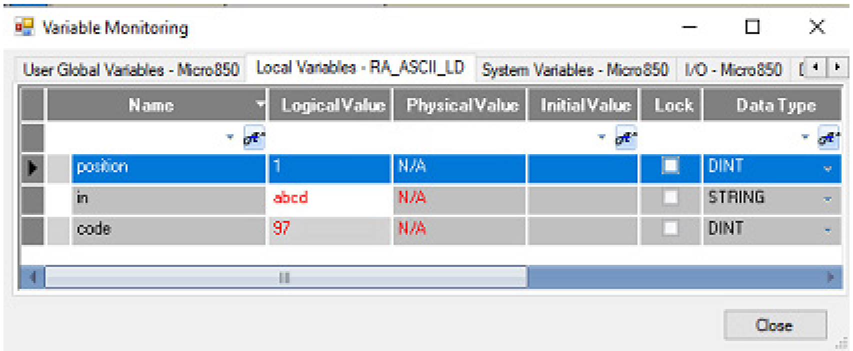
DINT **ASCII**(STRING IN, DINT Pos)
Character -> ASCII code

(* ST Equivalence: *)

FirstChr := ASCII (message, 1);

(* FirstChr is the ASCII code of the first character of the string *)

Results


Variable Monitoring
User Global Variables - Micro850
Local Variables - RA_ASCII_LD
System Variables - Micro850
I/O - Micro850
Close

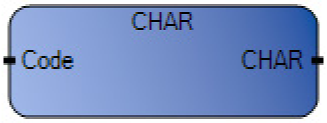
Name	Logical Value	Physical Value	Initial Value	Lock	Data Type
position	1	N/A		□	DINT
in	abcd	N/A		□	STRING
code	97	N/A		□	DINT

CHAR (ASCII code to string character)

Returns a one character string for an ASCII code. ASCII code -> character.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

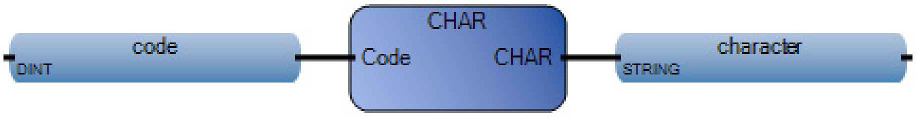
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



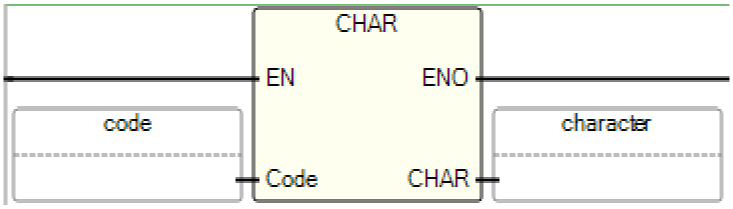
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - provide a single character string. FALSE - no operation. Applies to Ladder Diagram programs.
Code	Input	DINT	ASCII code in set [0 .. 255].
CHAR	Output	STRING	One character string. The character has the ASCII code given in input code.
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

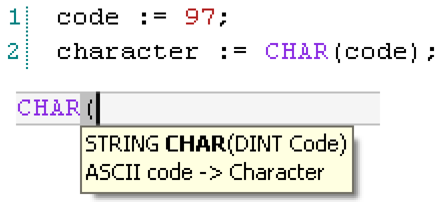
CHAR Function Block Diagram example



CHAR Ladder Diagram example



CHAR Structured Text example



```
code := 97;
character := CHAR(code);
```

```
CHAR(
STRING CHAR(DINT Code)
ASCII code -> Character
```

(* ST Equivalence: *)

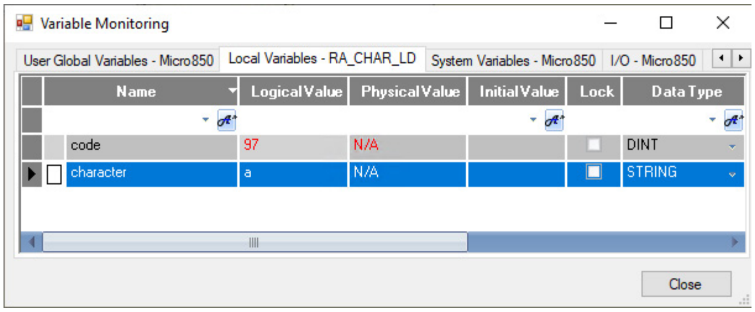
Display := CHAR (value + 48);

(* value is in set [0..9] *)

(* 48 is the ascii code of '0' *)

(* result is one character string from '0' to '9' *)

Results


Variable Monitoring
User Global Variables - Micro850
Local Variables - RA_CHAR_LD
System Variables - Micro850
I/O - Micro850
Close

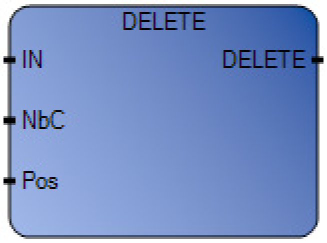
Name	Logical Value	Physical Value	Initial Value	Lock	Data Type
code	97	N/A		□	DINT
character	a	N/A		□	STRING

DELETE (delete sub-string)

Deletes characters from a string.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

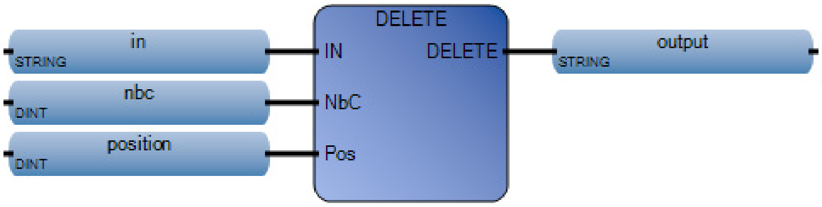
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



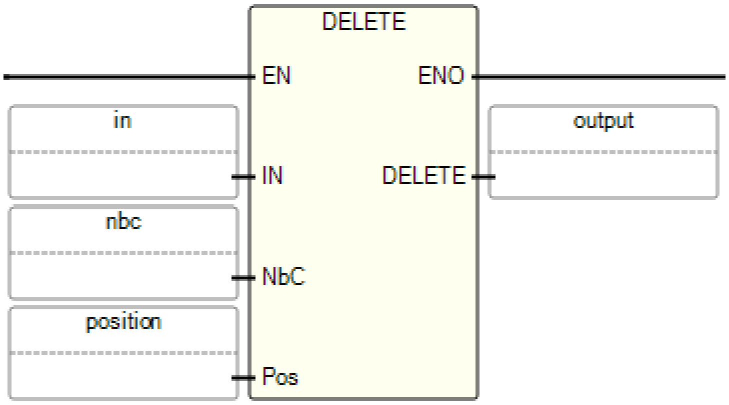
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - delete specified part of a string. FALSE - no operation. Applies to Ladder Diagram programs.
IN	Input	STRING	Any non-empty string.
NbC	Input	DINT	Number of characters to be deleted.
Pos	Input	DINT	Position of the first deleted character (first character of the string has position 1).
DELETE	Output	STRING	Output is: • a modified string. • an empty string (if Pos < 1). • the initial string (if Pos > IN string length). • the initial string (if NbC <= 0).
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

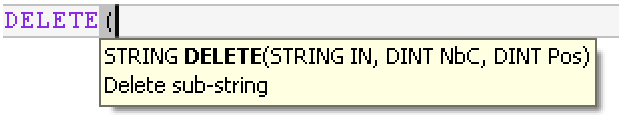
DELETE Function Block Diagram example



DELETE Ladder Diagram example

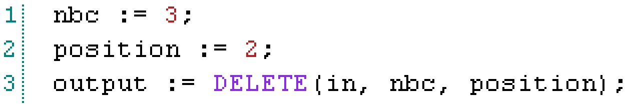


DELETE Structured Text example



DELETE (

STRING **DELETE**(STRING IN, DINT NbC, DINT Pos)
Delete sub-string



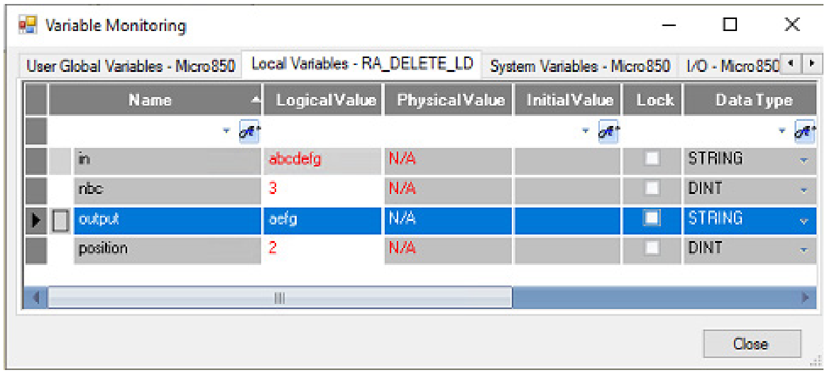
```
nbc := 3;
position := 2;
output := DELETE(in, nbc, position);
```

(* ST Equivalence: *)

complete_string := INSERT ('ABCD ', 'EFGH', 5); (* complete_string is 'ABCDEFGH ' *)

sub_string := DELETE (complete_string, 4, 3); (* sub_string is 'ABGH '*)

Results


Variable Monitoring
User Global Variables - Micro850
Local Variables - RA_DELETE_LD
System Variables - Micro850
I/O - Micro850
Close

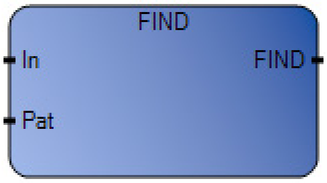
Name	Logical Value	Physical Value	Initial Value	Lock	Data Type
in	abcdefg	N/A		□	STRING
nbc	3	N/A		□	DINT
output	aefg	N/A		□	STRING
position	2	N/A		□	DINT

FIND (find sub-string)

Locates and provides the position of sub-strings within strings.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.

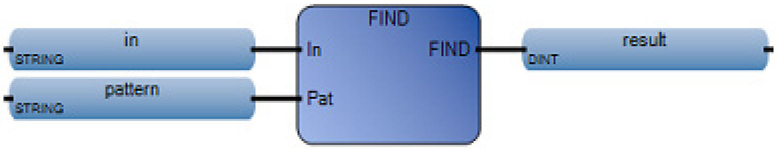
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



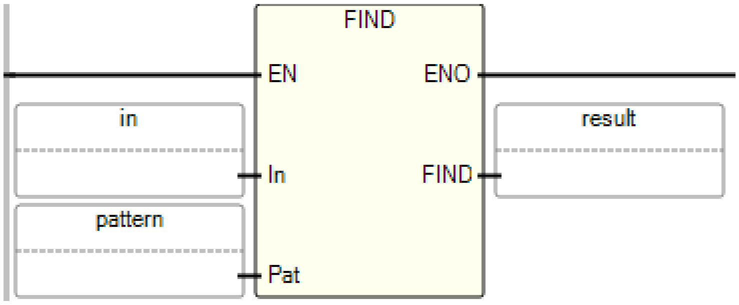
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - locate position within strings. FALSE - no locate operation. Applies to Ladder Diagram programs.
In	Input	STRING	Any non-empty string.
Pat	Input	STRING	Any non-empty string (Pattern).
FIND	Output	DINT	Output is: • 0 if the sub string Pat is not found. • the position of the first character of the first occurrence of the sub-string Pat (first position is 1) This instruction is case sensitive.
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

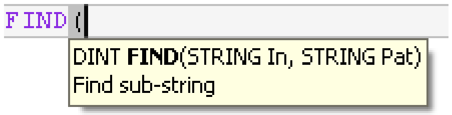
FIND Function Block Diagram example



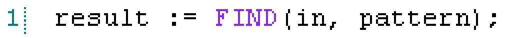
FIND Ladder Diagram example



FIND Structured Text example



```
FIND(
DINT FIND(STRING In, STRING Pat)
Find sub-string
```



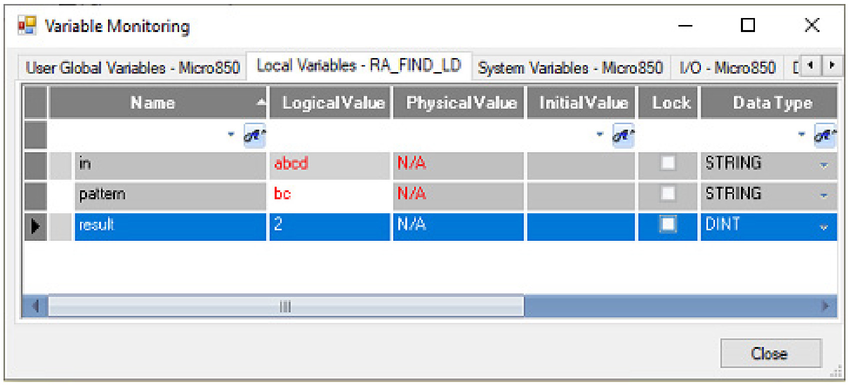
```
result := FIND(in, pattern);
```

(* ST Equivalence: *)

complete_string := 'ABCD' + 'EFGH'; (* complete_string is 'ABCDEFGH ' *)

found := FIND (complete_string, 'CDEF'); (* found is 3 *)

Results


Variable Monitoring
User Global Variables - Micro850
Local Variables - RA_FIND_LD
System Variables - Micro850
I/O - Micro850
Close

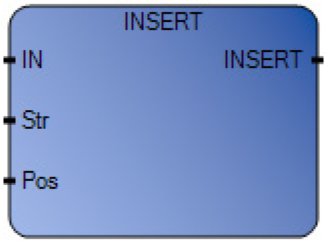
Name	Logical Value	Physical Value	Initial Value	Lock	Data Type
in	abcd	N/A		□	STRING
pattern	bc	N/A		□	STRING
result	2	N/A		□	DINT

INSERT (insert string)

Inserts sub-strings at user-defined positions within strings.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.

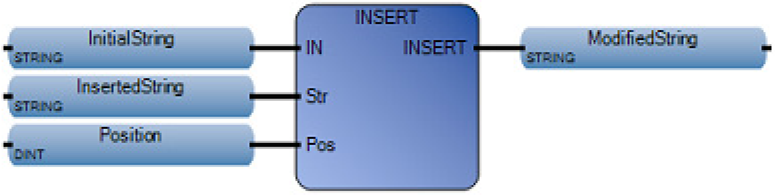


Use this table to help determine the parameter values for this instruction.

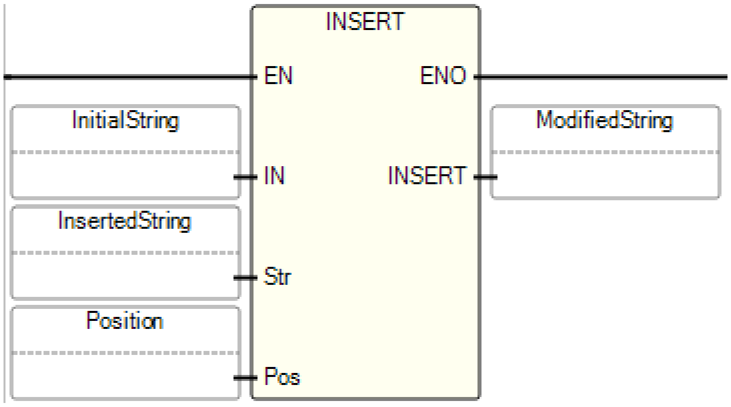
Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - insert sub-strings in a string. FALSE - no operation. Applies to Ladder Diagram programs.
IN	Input	STRING	Initial string.

Parameter	Parameter Type	Data Type	Description
Str	Input	STRING	String to be inserted.
Pos	Input	DINT	Position of the insertion the insertion is done before the position (first valid position is 1).
INSERT	Output	STRING	Modified string. Can be: • empty string if Pos <= 0 • concatenation of both strings if Pos is greater than the length of the IN string
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

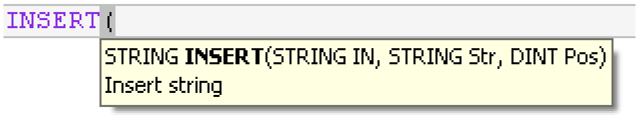
INSERT Function Block Diagram example



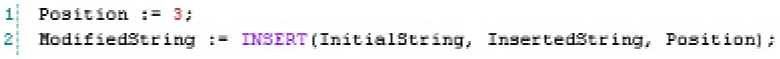
INSERT Ladder Diagram example



INSERT Structured Text example



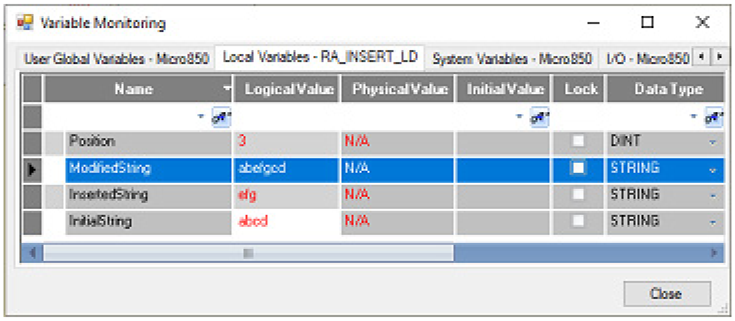
```
INSERT(
STRING INSERT(STRING IN, STRING Str, DINT Pos)
Insert string
```



```
Position := 3;
ModifiedString := INSERT(InitialString, InsertedString, Position);
```

```
(* ST Equivalence: *)

MyName := INSERT ('Mr JONES', 'Frank ', 4);

(* MyName is 'Mr Frank JONES' *)
```

Results



Variable Monitoring

User Global Variables - Micro850 | Local Variables - RA_INSERT_LD | System Variables - Micro850 | I/O - Micro850

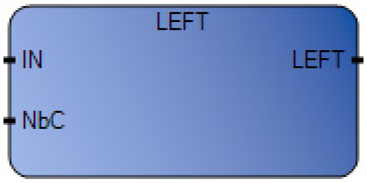
Name	LogicalValue	PhysicalValue	InitialValue	Lock	Data Type
Position	3	N/A		□	DINT
ModifiedString	abefgcd	N/A		□	STRING
InsertedString	efg	N/A		□	STRING
InitialString	abcd	N/A		□	STRING

LEFT (extract left of a sting)

Extract characters from the left side of a string.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

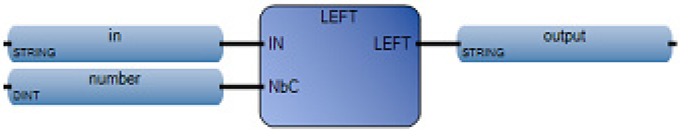
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



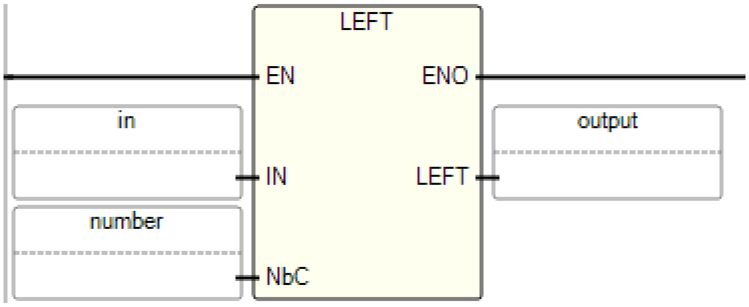
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - calculate number of characters from left side of string. FALSE - no operation. Applies to Ladder Diagram programs.
IN	Input	STRING	Any non-empty string.
NbC	Input	DINT	Number of characters to be extracted. This number cannot be greater than the length of the IN string.
LEFT	Output	STRING	Left part of the IN string (its length = NbC). Can be: • empty string if NbC <= 0 • complete IN string if NbC >= IN string length
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

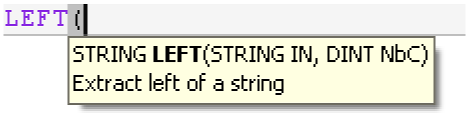
LEFT Function Block Diagram example



LEFT Ladder Diagram example

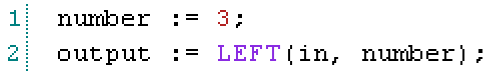


LEFT Structured Text example



LEFT(

STRING **LEFT**(STRING IN, DINT NbC)
Extract left of a string



```
number := 3;
output := LEFT(in, number);
```

(* ST Equivalence: *)

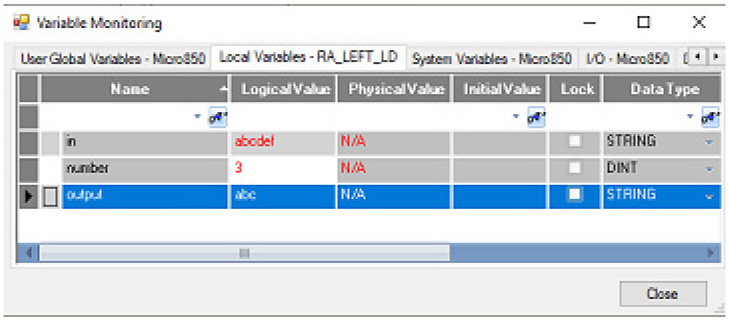
complete_string := RIGHT ('12345678', 4), LEFT ('12345678', 4), 5;

(* complete_string is '56781234'

the value issued from RIGHT call is '5678'

the value issued from LEFT call is '1234'*)

Results


Variable Monitoring
User Global Variables - Micro850
Local Variables - RA_LEFT_LD
System Variables - Micro850
I/O - Micro850
Close

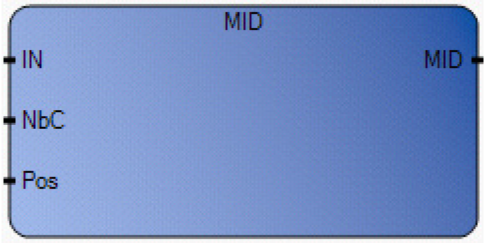
Name	LogicalValue	PhysicalValue	InitialValue	Lock	Data Type
in	abcdef	N/A		□	STRING
number	3	N/A		□	DINT
output	abc	N/A		□	STRING

MID (extract middle of a string)

Extract characters from the middle of a string. Use the position and number of characters provided to calculate the required parts of strings.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

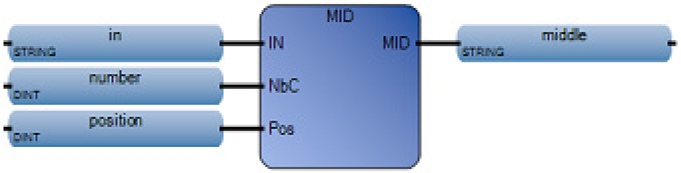
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



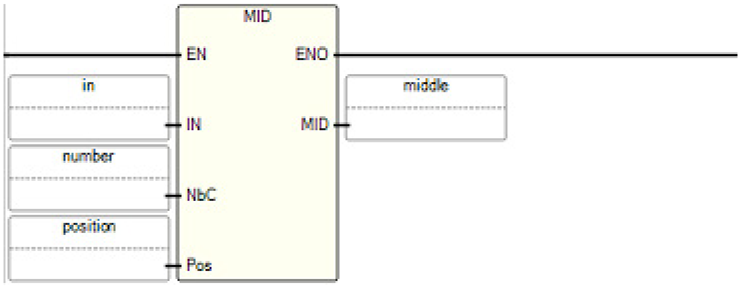
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - generate portion of a string. FALSE - no generate operation. Applies to Ladder Diagram programs.
IN	Input	STRING	Any non-empty string.
NbC	Input	DINT	Number of characters to be extracted cannot be greater than the length of the IN string.
Pos	Input	DINT	Position of the sub-string. The sub-string first character will be the one pointed to by Pos (first valid position is 1).
MID	Output	STRING	Middle part of the string (its length = NbC). When the number of characters to extract exceeds the length of the IN string, NbC is automatically recalculated to get the remainder of the string only. When NbC or Pos are zero or negative numbers, an empty string is returned.
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

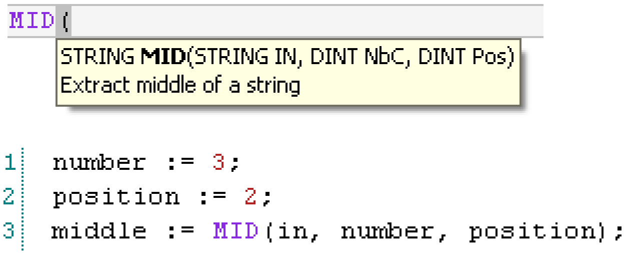
MID Function Block Diagram example



MID Ladder Diagram example



MID Structured Text example



```
MID(
STRING MID(STRING IN, DINT NbC, DINT Pos)
Extract middle of a string
```

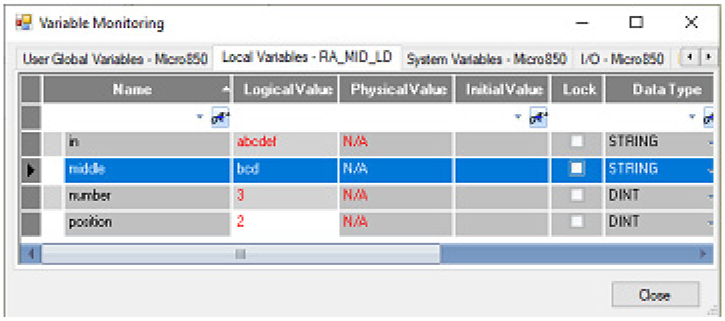
```
number := 3;
position := 2;
middle := MID(in, number, position);
```

(* ST Equivalence: *)

sub_string := MID ('abcdefgh', 2, 4);

(* sub_string is 'de' *)

Results


Variable Monitoring
User Global Variables - Micro850
Local Variables - RA_MID_LD
System Variables - Micro850
I/O - Micro850
Close

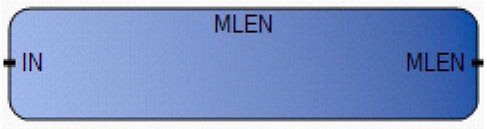
Name	Logical Value	Physical Value	Initial Value	Lock	Data Type
in	abcdef	N/A		□	STRING
middle	bcd	N/A		□	STRING
number	3	N/A		□	DINT
position	2	N/A		□	DINT

MLEN (string length)

Calculates the length of a string.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

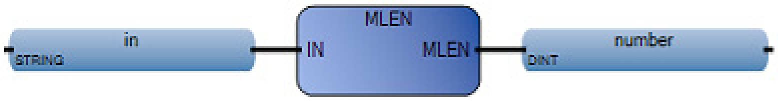
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



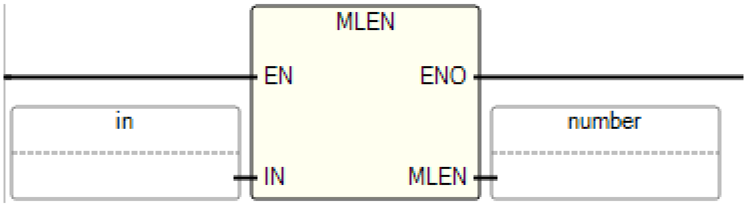
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - calculate length of string. FALSE - no operation. Applies to Ladder Diagram programs.
IN	Input	STRING	Any string.
MLEN	Output	DINT	Number of characters in the IN string.
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

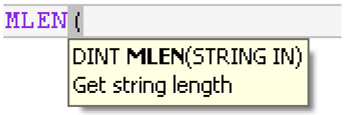
MLEN Function Block Diagram example



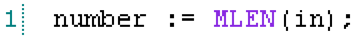
MLEN Ladder Diagram example



MLEN Structured Text example



```
MLEN(
DINT MLEN(STRING IN)
Get string length
```



```
number := MLEN(in);
```

(* ST Equivalence: *)

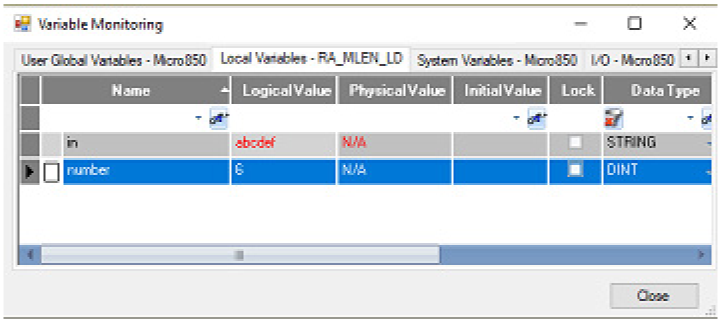
nbchar := MLEN (complete_string);

If (nbchar < 3) Then Return; End_if;

prefix := LEFT (complete_string, 3);

(* this program extracts the 3 characters on the left of the string and puts the result in the prefix string variable. Nothing is done if the string length is less than 3 characters *)

Results


Variable Monitoring
User Global Variables - Micro850
Local Variables - RA_MLEN_LD
System Variables - Micro850
I/O - Micro850
Close

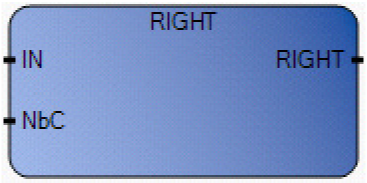
Name	Logical Value	Physical Value	Initial Value	Lock	Data Type
in	abcdef	N/A		□	STRING
number	6	N/A		■	DINT

RIGHT (extract right of a string)

Extract characters from the right side of a string.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.

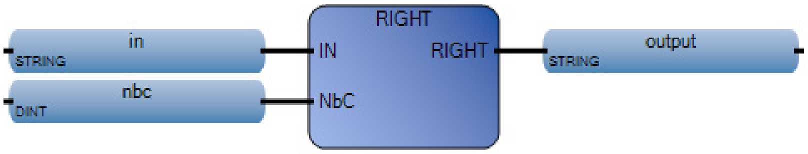


Use this table to help determine the parameter values for this instruction.

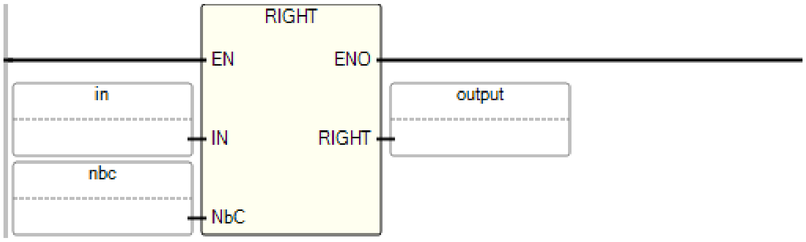
Parameter	Parameter Type	Data Type	Description

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - extract specified number of characters from the right end of the string. FALSE - no operation. Applies to Ladder Diagram programs.
IN	Input	STRING	Any non-empty string.
NbC	Input	DINT	Number of characters to be extracted. This number cannot be greater than the length of the IN string.
RIGHT	Output	STRING	Right part of the string (length = NbC). Can be: • empty string if NbC <= 0 • complete string if NbC >= string length
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

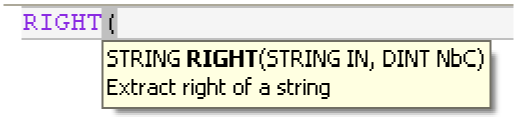
RIGHT Function Block Diagram example



RIGHT Ladder Diagram example

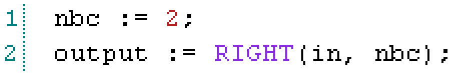


RIGHT Structured Text example



```
RIGHT(
```

STRING **RIGHT**(STRING IN, DINT NbC)
Extract right of a string



```
nbc := 2;
output := RIGHT(in, nbc);
```

(* ST Equivalence: *)

complete_string := RIGHT ('12345678', 4), LEFT ('12345678', 4),5;

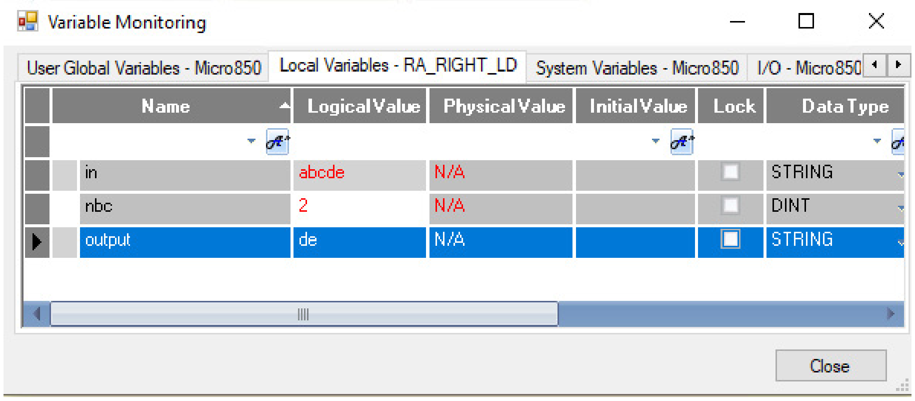
(* complete_string is '56781234'

the value issued from RIGHT call is '5678'

the value issued from LEFT call is '1234'

*)

Results



Variable Monitoring

User Global Variables - Micro850 | Local Variables - RA_RIGHT_LD | System Variables - Micro850 | I/O - Micro850

Name	Logical Value	Physical Value	Initial Value	Lock	Data Type
in	abcde	N/A		☐	STRING
nbc	2	N/A		☐	DINT
output	de	N/A		☐	STRING

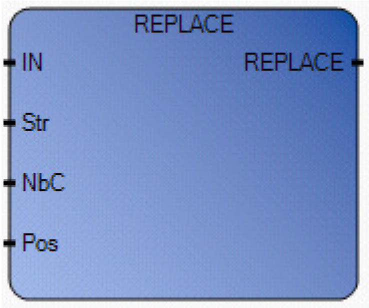
Close

REPLACE (replace sub-string)

Replaces parts of a string with new sets of characters.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.

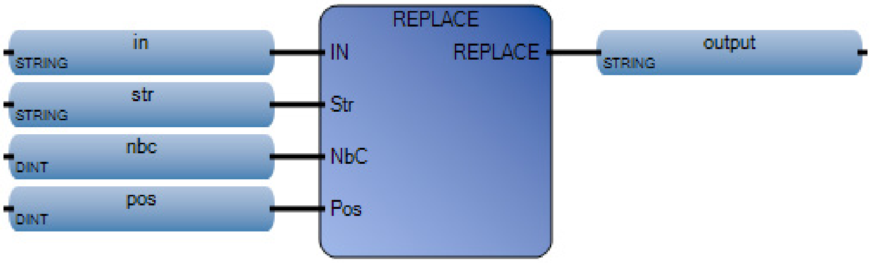


Use this table to help determine the parameter values for this instruction.

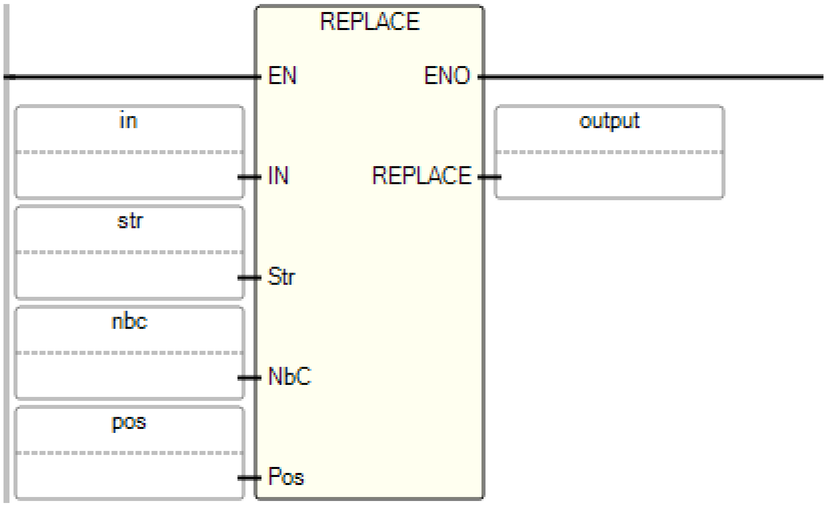
Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Function enable. TRUE - replace parts of strings with new characters. FALSE - no operation. Applies to Ladder Diagram programs.
IN	Input	STRING	Any string.
Str	Input	STRING	String to be inserted (to replace NbC chars).
NbC	Input	DINT	Number of characters to be deleted.
Pos	Input	DINT	Position of the first modified character (first valid position is 1).

Parameter	Parameter Type	Data Type	Description
REPLACE	Output	STRING	Modified string. The NbC characters are deleted at position Pos, then the substring Str is inserted at this position. Can be: • empty string if Pos <= 0 • strings concatenation (IN+Str) if Pos is greater than the length of the IN string • initial string IN if NbC <= 0
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.

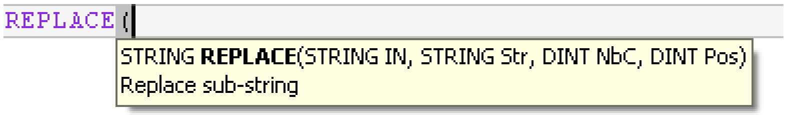
REPLACE Function Block Diagram example



REPLACE Ladder Diagram example

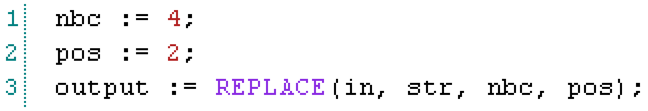


REPLACE Structured Text example



```
REPLACE (
```

STRING **REPLACE**(STRING IN, STRING Str, DINT NbC, DINT Pos)
Replace sub-string



```
nbc := 4;
pos := 2;
output := REPLACE(in, str, nbc, pos);
```

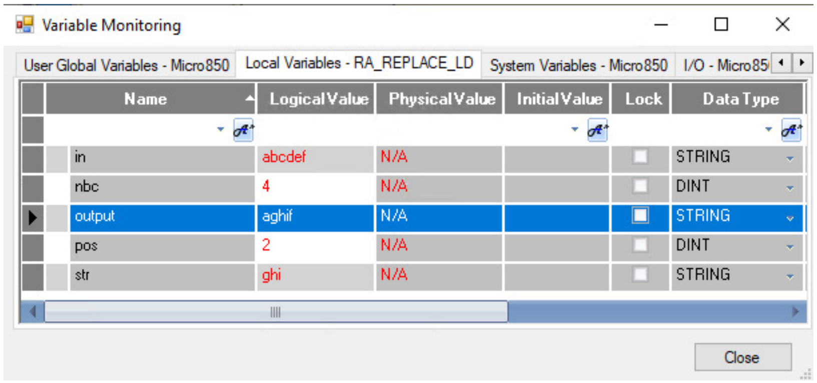
Replacing a part of a string with a new set of characters.

(* ST Equivalence: *)

MyName := REPLACE ('Mr X JONES, 'Frank', 1, 4);

(* MyName is 'Mr Frank JONES' *)

Results


Variable Monitoring
User Global Variables - Micro850
Local Variables - RA_REPLACE_LD
System Variables - Micro850
I/O - Micro85
Close

Name	Logical Value	Physical Value	Initial Value	Lock	Data Type
in	abcdef	N/A		☐	STRING
nbc	4	N/A		☐	DINT
output	aghif	N/A		☐	STRING
pos	2	N/A		☐	DINT
str	ghi	N/A		☐	STRING

Timer instructions

Use Timer instructions to control operations based on time.

Instruction	Description
TOF on page 623	Off-delay timing. Increase an internal timer up to a given value.
TON on page 625	On-delay timing. Increase an internal timer up to a given value.
TONOFF on page 628	Delay turning on an output on a true rung and then delay turning off the output on the false rung.
TP on page 630	Pulse timing. On a rising edge, increases an internal timer up to a given value.
RTO on page 632	Retentive timing. Increases an internal timer when input is active but does not reset the internal timer when input changes to inactive.
DOY on page 634	Turn on an output if the value of the real-time clock is in the range of the Year Time setting.
TDF on page 636	Computes the time difference between TimeA and TimeB.
TOW on page 638	Turns on an output if the value of the real-time clock is in the range of the Time of Week setting.

Timer instruction configuration

Time accuracy refers to the time between the moment the processor enables a timer instruction and the moment the processor completes the timed interval.

The processor uses the following information from the timer instruction on page 623:

- **Timer** - The timer control address in the timer area of data storage.
- **Time Base** - Determines how the timer operates.
- **Preset** - Specifies the value that the timer must reach before the processor sets the done bit.
- **Accumulated value** - The number of time increments the instruction has counted. When enabled, the timer updates this value continually.

TOF (timer, off-delay)

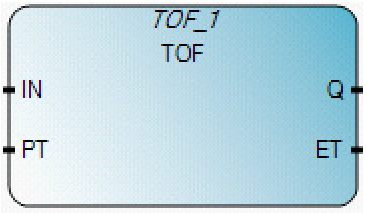
Increases an internal timer up to a given value.

Operation details:

- If the EN parameter is used with this instruction block, the timer starts incrementing when EN is set to TRUE, and continues to increment even if EN is set to FALSE.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

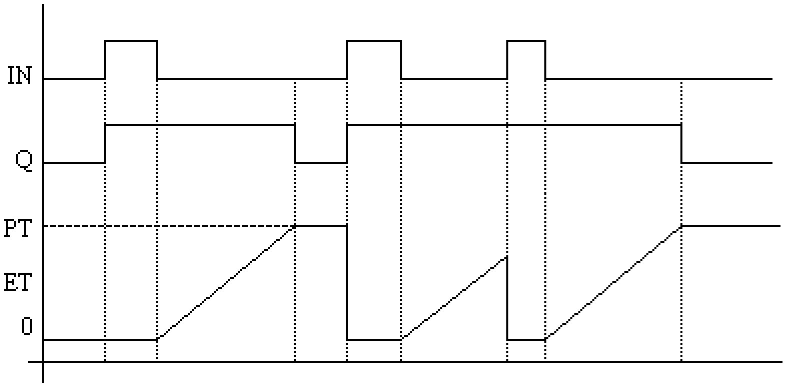
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



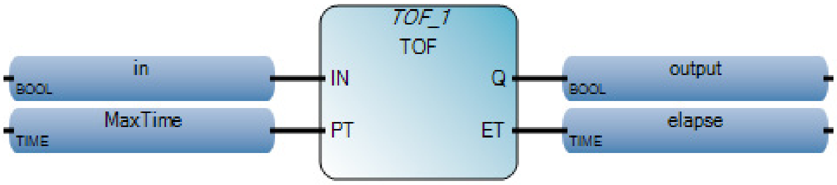
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter type	Data type	Description
IN	Input	BOOL	Input control. TRUE - Falling edge detected, starts increasing internal timer. FALSE - Rising edge detected, stops, and resets internal timer.
PT	Input	TIME	Maximum programmed time. See Time data type.
Q	Output	BOOL	TRUE - total time is **not** elapsed. FALSE - total time has elapsed.
ET	Output	TIME	Current elapsed time. Possible values range from 0ms to 1193h2m47s294ms.

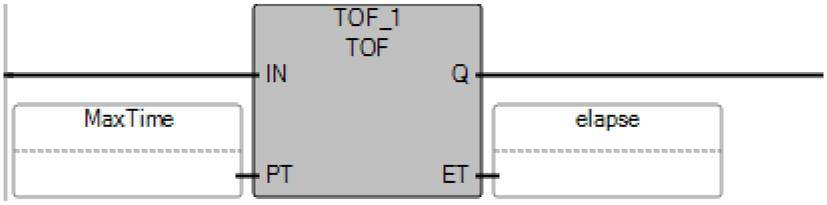
TOF timing diagram



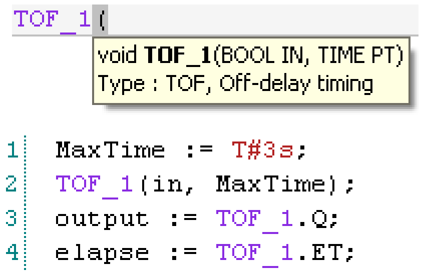
TOF Function Block Diagram example



TOF Ladder Diagram example



TOF Structured Text example

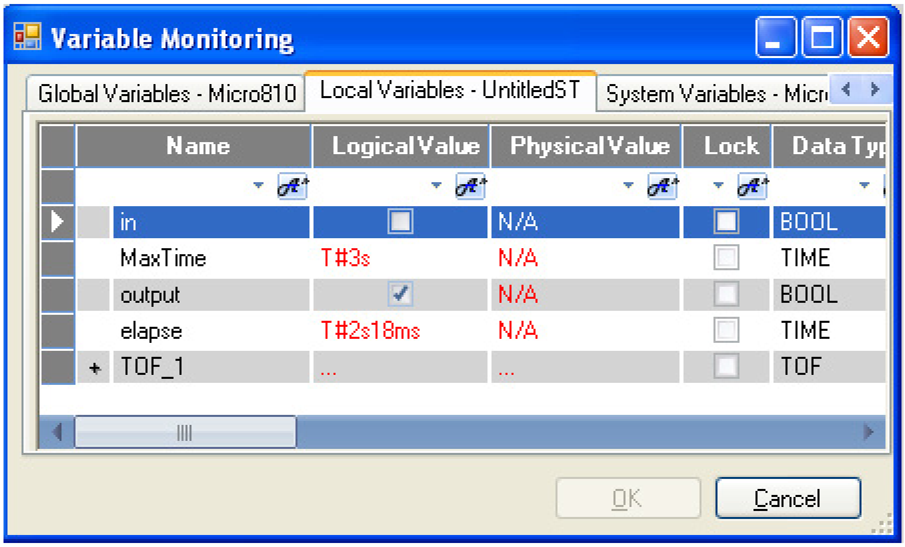


```
TOF_1(
```

void **TOF_1**(BOOL IN, TIME PT)
Type : TOF, Off-delay timing

```
MaxTime := T#3s;
TOF_1(in, MaxTime);
output := TOF_1.Q;
elapse := TOF_1.ET;
```

Results



Name	Logical Value	Physical Value	Lock	Data Typ
in	□	N/A	□	BOOL
MaxTime	T#3s	N/A	□	TIME
output	☑	N/A	□	BOOL
elapse	T#2s18ms	N/A	□	TIME
+ TOF_1	...	...	□	TOF

TON (timer, on-delay)

Increases an internal timer up to a given value.

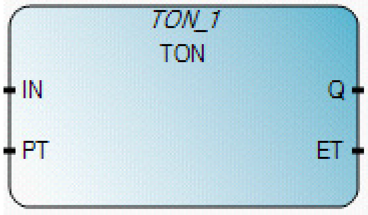
Operation details:

- Do not use a jump to skip over a TON instruction block in a Ladder Diagram (LD). If a jump is used, the TON timer continues after the elapsed time.
- For example: Rung 1 contains a jump; rung 2 contains a TON instruction block with an elapsed time of 10 seconds; enable the jump from rung 1 to rung 3; disable the jump after 30 seconds; the elapsed time is 30 seconds - not 10 seconds as defined in the elapsed time.

- If the EN parameter is used with TON, the timer starts incrementing when EN is set to TRUE, and continues to increment even if EN is set to FALSE.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

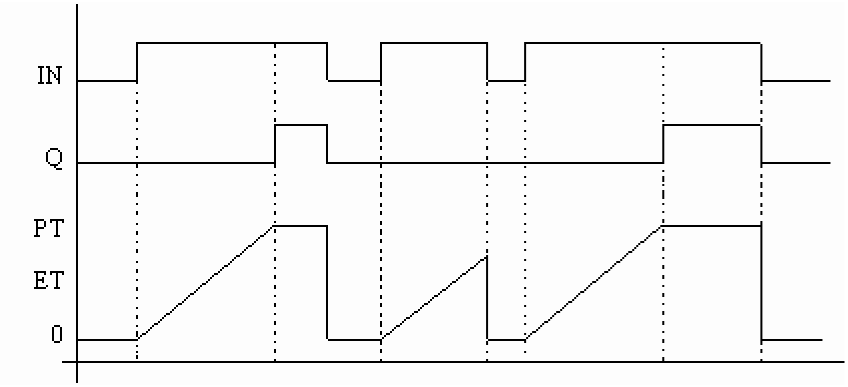
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



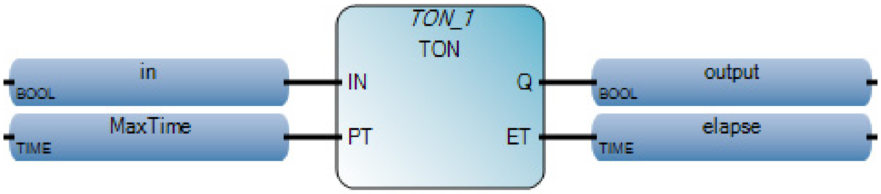
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter type	Data type	Description
IN	Input	BOOL	Input control. TRUE - If rising edge, starts increasing internal timer. FALSE- If falling edge, stops and resets internal timer.
PT	Input	TIME	Maximum programmed time defined using Time data type.
Q	Output	BOOL	TRUE - programmed time has elapsed. FALSE - programmed time has not elapsed.
ET	Output	TIME	Current elapsed time. Possible values range from 0ms to 1193h2m47s294ms.

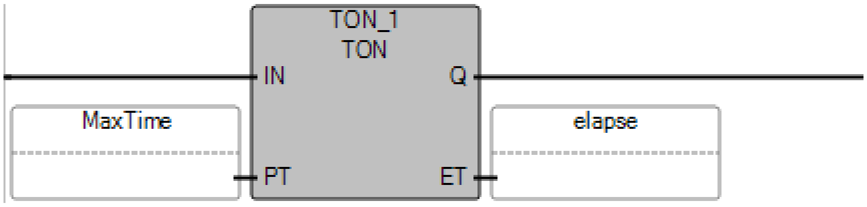
TON timing diagram



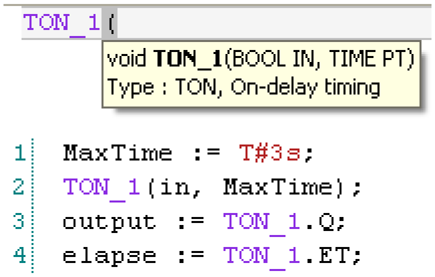
TON Function Block Diagram example



TON Ladder Diagram example



TON Structured Text example

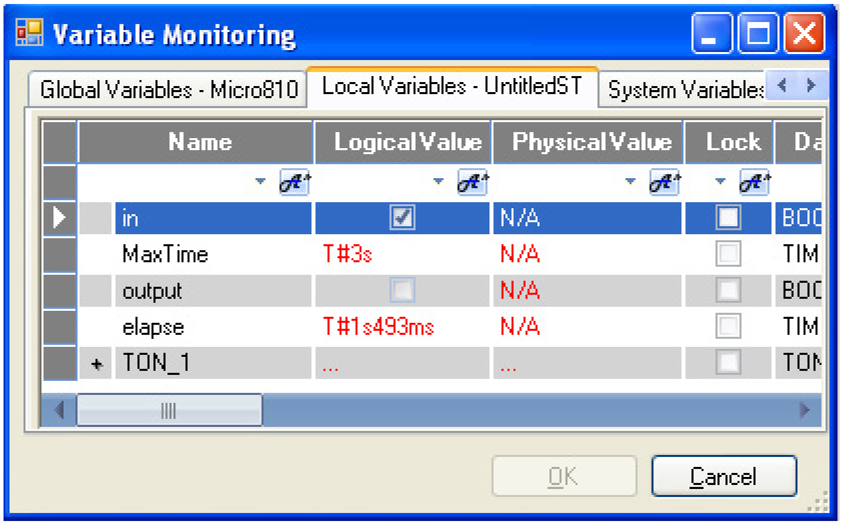


```
TON_1(
```

void **TON_1**(BOOL IN, TIME PT)
Type : TON, On-delay timing

```
MaxTime := T#3s;
TON_1(in, MaxTime);
output := TON_1.Q;
elapse := TON_1.ET;
```

Results



Name	Logical Value	Physical Value	Lock	Da
in	☑	N/A	☐	BOO
MaxTime	T#3s	N/A	☐	TIM
output	☐	N/A	☐	BOO
elapse	T#1s493ms	N/A	☐	TIM
+ TON_1	...	...	☐	TON

TONOFF (time-delay, on, off)

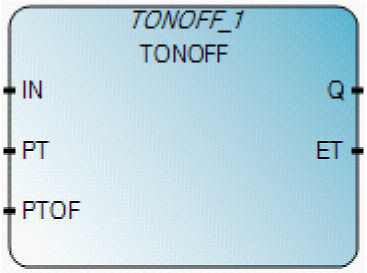
Delays turning on an output on a true rung, then delays turning off the output on the false rung.

Operation details:

- If the EN parameter is used with TONOFF, the timer starts incrementing when EN is set to TRUE, and continues to increment even if EN is set to FALSE.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

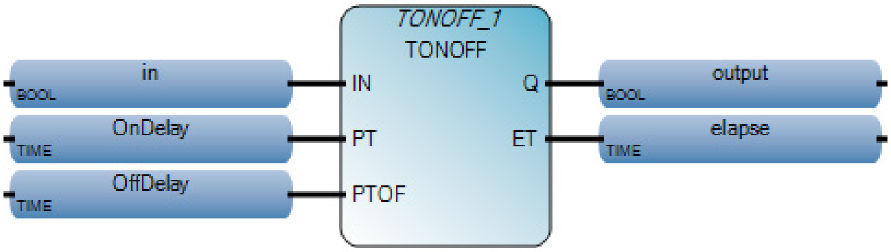
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



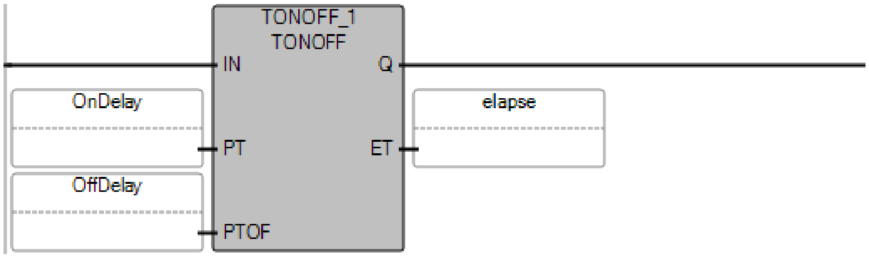
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter type	Data type	Description
IN	Input	BOOL	Input control. TRUE - Rising Edge detected (IN turns from 0 to 1): • start the On-delay timer (PT). • if Programmed Off--delay time (PTOF) is not elapsed, restart the On-delay (PT) timer. FALSE - Falling Edge detected (IN turns from 1 to 0): • if Programmed On-delay time (PT) is not elapsed, stop PT timer and reset ET. • if Programmed On-delay time (PT) is elapsed, the start the Off-delay timer (PTOF).
PT	Input	TIME	The on-delay time setting defined using the Time data type.
PTOF	Input	TIME	The off-delay time setting defined using the Time data type.
Q	Output	BOOL	TRUE - the Programmed On-delay time is elapsed and Programmed Off-delay time is not elapsed.
ET	Output	TIME	Current elapsed time. Possible values range from 0ms to 1193h2m47s294ms. If the Programmed On-delay time is elapsed and the Off-delay timer is not starting, the elapsed time (ET) remains at the on-delay (PT) value. If the Programmed Off-delay time is elapsed and the Off-delay timer is not starting, the elapsed time (ET) remains at the off-delay (PTOF) value until the rising edge occurs again.

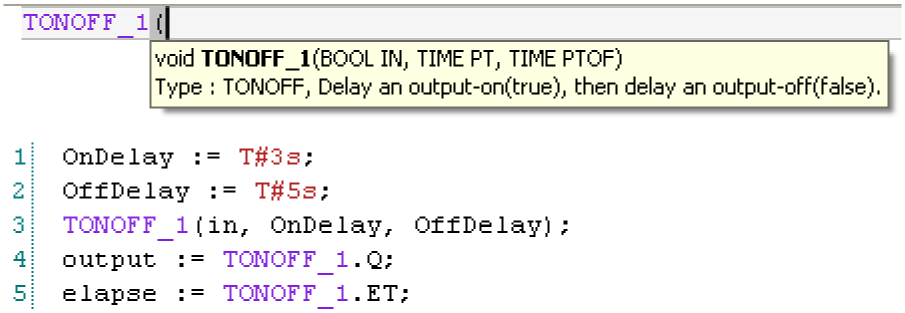
TONOFF Function Block Diagram example



TONOFF Ladder Diagram example



TONOFF Structured Text example

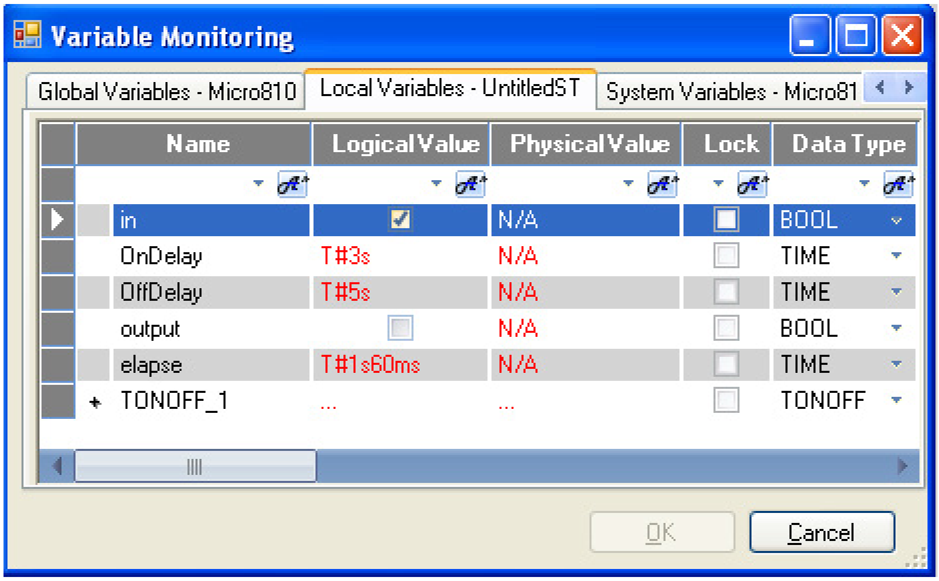


```
TONOFF_1(
```

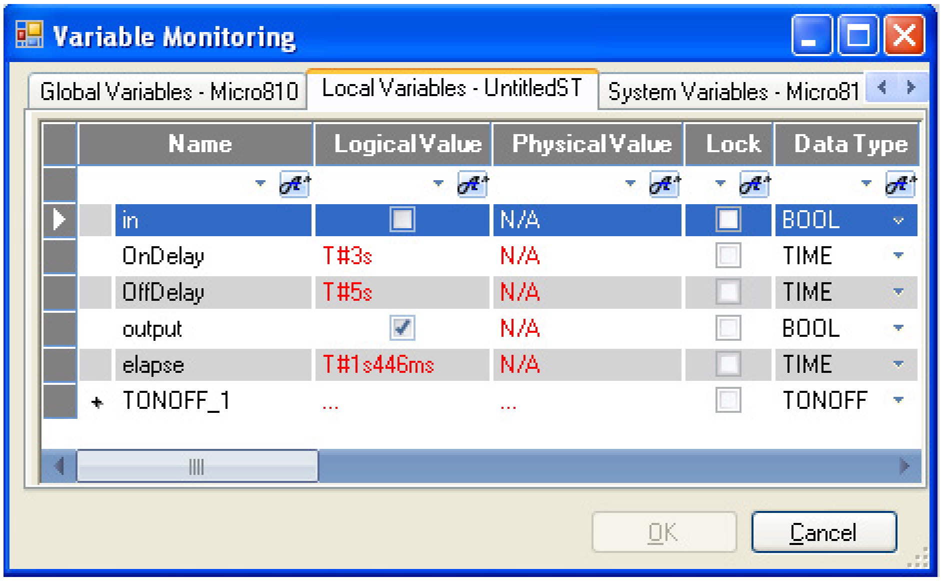
void **TONOFF_1**(BOOL IN, TIME PT, TIME PTOF)
Type : TONOFF, Delay an output-on(true), then delay an output-off(false).

```
OnDelay := T#3s;
OffDelay := T#5s;
TONOFF_1(in, OnDelay, OffDelay);
output := TONOFF_1.Q;
elapse := TONOFF_1.ET;
```

Results


Variable Monitoring
Global Variables - Micro810
Local Variables - UntitledST
System Variables - Micro81
OK
Cancel

Name	Logical Value	Physical Value	Lock	Data Type
in	☑	N/A	□	BOOL
OnDelay	T#3s	N/A	□	TIME
OffDelay	T#5s	N/A	□	TIME
output	□	N/A	□	BOOL
elapse	T#1s60ms	N/A	□	TIME
+ TONOFF_1	...	...	□	TONOFF


Variable Monitoring
Global Variables - Micro810
Local Variables - UntitledST
System Variables - Micro81
OK
Cancel

Name	Logical Value	Physical Value	Lock	Data Type
in	□	N/A	□	BOOL
OnDelay	T#3s	N/A	□	TIME
OffDelay	T#5s	N/A	□	TIME
output	☑	N/A	□	BOOL
elapse	T#1s446ms	N/A	□	TIME
+ TONOFF_1	...	...	□	TONOFF

TP (pulse timing)

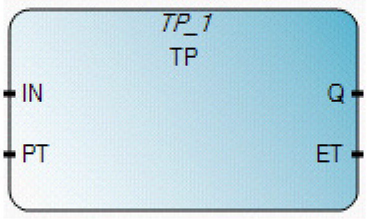
On a rising edge, increases an internal timer up to a given value. If the timer is elapsed, the internal time is reset.

Operation details:

- If the EN parameter is used with TP, the timer starts incrementing when EN is set to TRUE, and continues to increment even if EN is set to FALSE.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

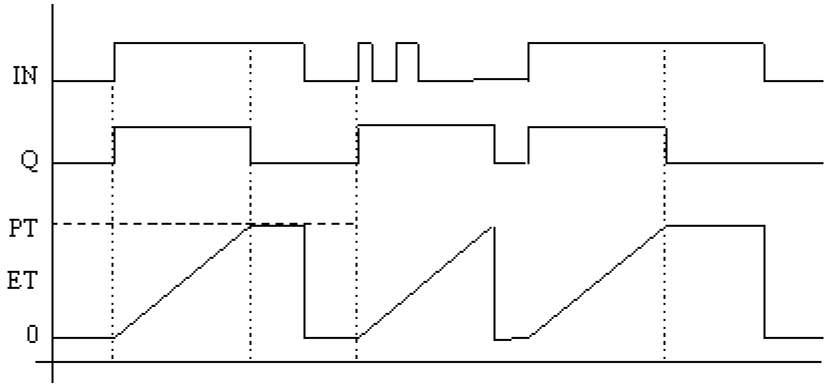
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



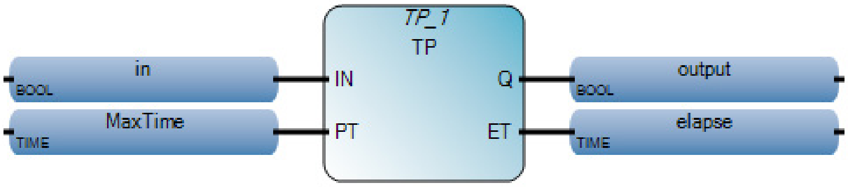
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter type	Data type	Description
IN	Input	BOOL	TRUE - If rising edge, starts increasing internal timer (if not already increasing). FALSE - if timer has elapsed, resets the internal timer. Any change to IN during counting has no effect.
PT	Input	TIME	Maximum programmed time defined using the Time data type.
Q	Output	BOOL	TRUE - timer is counting. FALSE - timer is not counting.
ET	Output	TIME	Current elapsed time. Possible values range from 0ms to 1193h2m47s294ms.

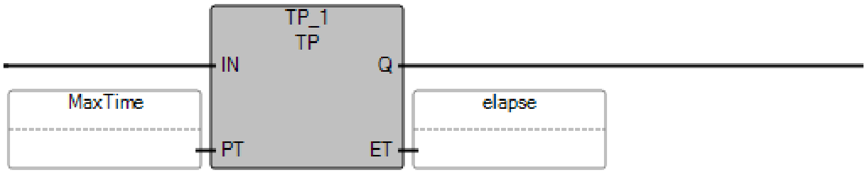
TP timing diagram



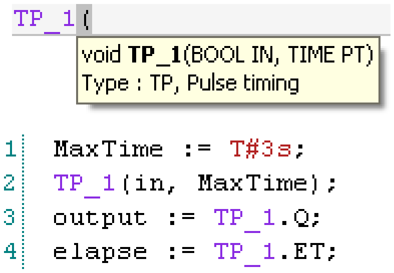
TP Function Block Diagram example



TP Ladder Diagram example



TP Structured Text example

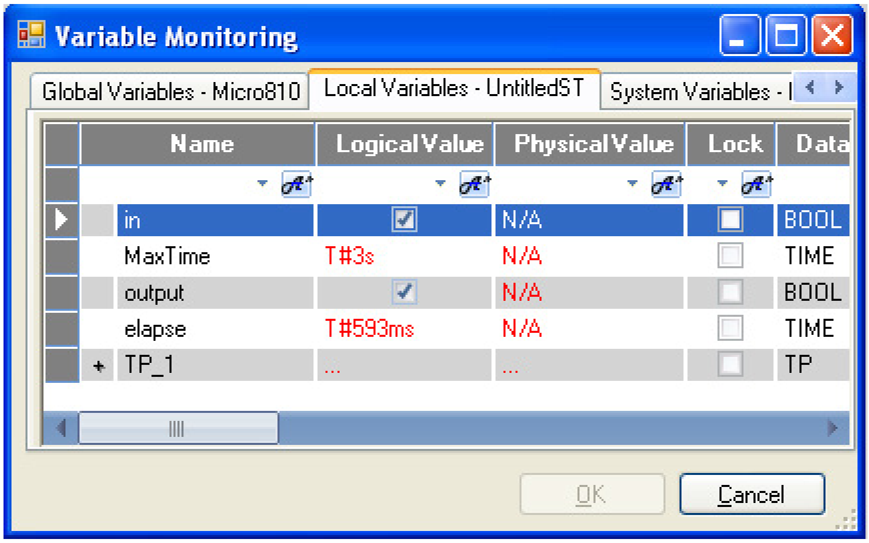


```
TP_1(
```

void **TP_1**(BOOL IN, TIME PT)
Type : TP, Pulse timing

```
MaxTime := T#3s;
TP_1(in, MaxTime);
output := TP_1.Q;
elapse := TP_1.ET;
```

Results



Name	LogicalValue	PhysicalValue	Lock	Data
in	☑	N/A	□	BOOL
MaxTime	T#3s	N/A	□	TIME
output	☑	N/A	□	BOOL
elapse	T#593ms	N/A	□	TIME
+ TP_1	...	...	□	TP

RTO (retentive timer, on-delay)

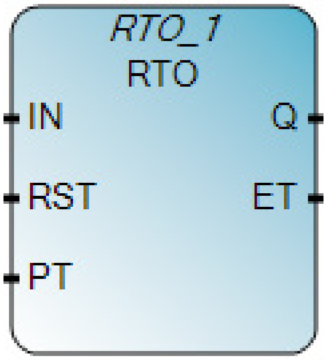
Increases an internal timer when input is active, but does not reset the internal timer when input changes to inactive.

Operation details:

- Micro810 or Micro820 controllers, the RTO internal timer does not persist through a power cycle by default. To persist the internal timer, set the Retained configuration parameter to true.
- Micro830 or Micro850 controller, the RTO internal timer persists through a power cycle.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

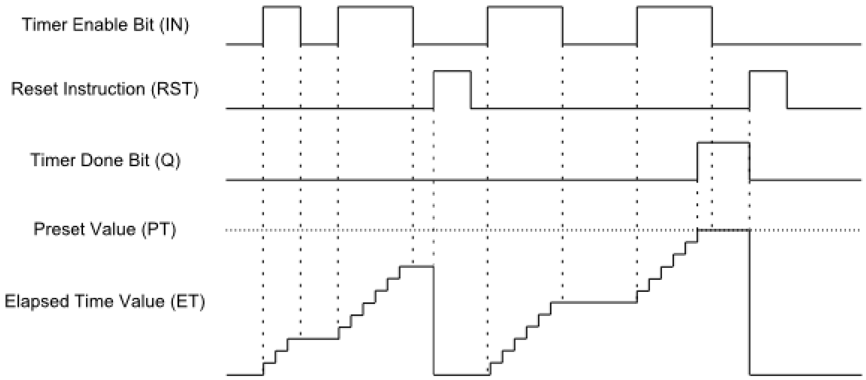
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



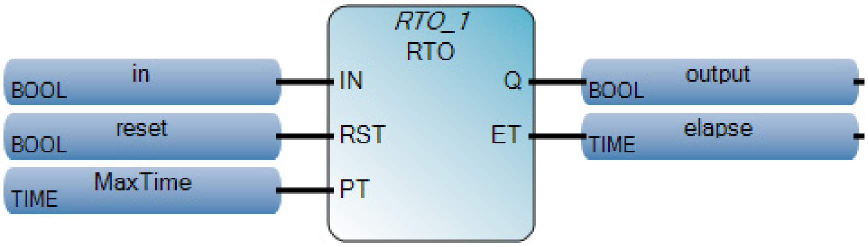
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter type	Data type	Description
IN	Input	BOOL	Input control. TRUE - Rising edge, starts increasing internal timer. FALSE - Falling edge, stops and does not reset the internal timer.
RST	Input	BOOL	TRUE - Rising edge, resets the internal timer. FALSE - Does not reset internal timer.
PT	Input	TIME	Maximum programmed on-delay time. PT is defined using the Time data type.
Q	Output	BOOL	TRUE - Programmed on-delay time is elapsed. FALSE - Program on-delay time has not elapsed.
ET	Output	TIME	Current elapsed time. Values range from 0ms to 1193h2m47s294ms. ET is defined using the Time data type.

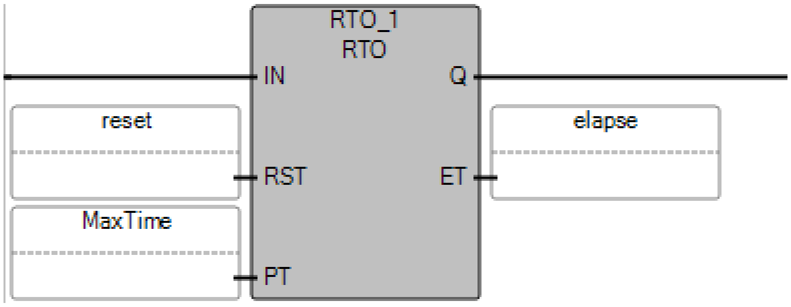
RTO timing diagram example



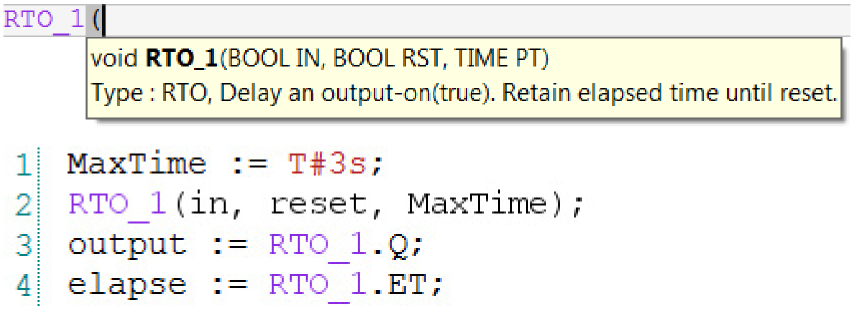
RTO Function Block Diagram example



RTO Ladder Diagram example



RTO Structured Text example



```
RTO_1(
void RTO_1(BOOL IN, BOOL RST, TIME PT)
Type : RTO, Delay an output-on(true). Retain elapsed time until reset.
```

```
MaxTime := T#3s;
RTO_1(in, reset, MaxTime);
output := RTO_1.Q;
elapse := RTO_1.ET;
```

DOY (check year for real-time clock)

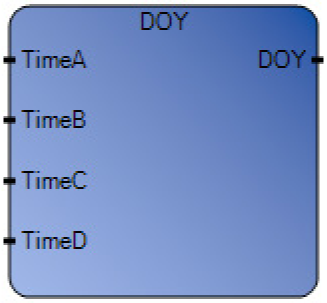
Turns on an output when the value of real-time clock (RTC) is in the range of the Year Time setting.

Operation details:

- If RTC is not present, the output is always off.
- Configure the Time input parameters with valid ranges as specified in the DOYDATA Data Type. An invalid value faults the controller when TimeX.Enable is set to TRUE and an RTC is present and enabled.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

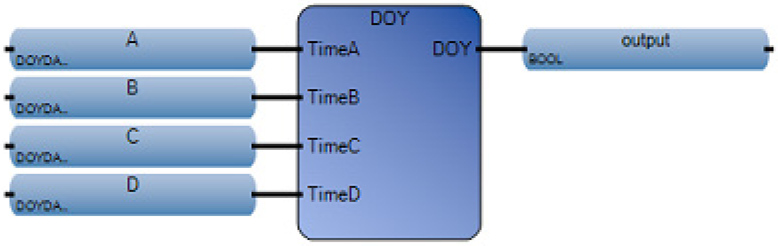
Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - perform the operation. FALSE - do not perform the operation.
TimeA	Input	DOYDATA	Year Time Setting of Channel A. Use the DOYDATA data type to configure TimeA.
TimeB	Input	DOYDATA	Year Time Setting of Channel B. Use the DOYDATA data type to configure TimeB.
TimeC	Input	DOYDATA	Year Time Setting of Channel C. Use the DOYDATA data type to configure TimeC.
TimeD	Input	DOYDATA	Year Time Setting of Channel D. Use the DOYDATA data type to configure TimeD.
DOY	Output	BOOL	If TRUE, the value of the real-time clock is in the range of the Year Time setting of any one of the four channels.

DOYDATA Data Type

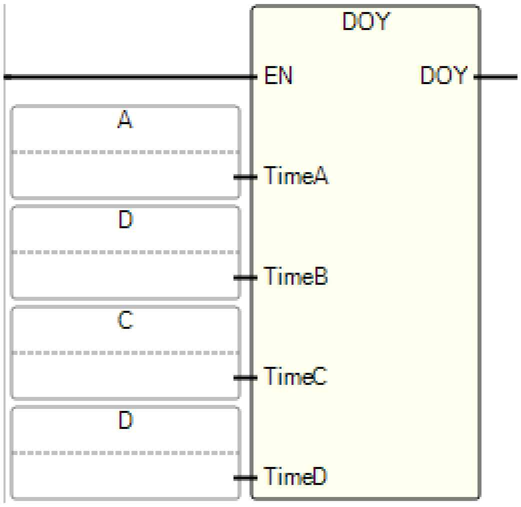
The following table describes the DOYDATA data type.

Parameter	Data Type	Description
Enable	BOOL	TRUE:Enable; FALSE:Disable
YearlyCenturial	BOOL	Type of timer (0:Yearly timer; 1:Centurial timer).
YearOn	UINT	Year On value (must be in set [2000...2098]).
MonthOn	USINT	Month On value (must be in set [1...12]).
DayOn	USINT	Day On value (must be in set [1...31], determined by "MonthOn" value).
YearOff	UINT	Year Off value (must be in set [2000...2098]).
MonthOff	USINT	Month Off value (must be in set [1...12]).
DayOff	USINT	Day Off value (must be in set [1...31], determined by "MonthOff" value).

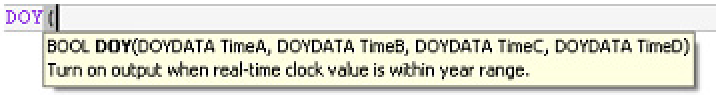
DOY Function Block Diagram example



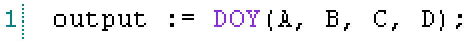
DOY Ladder Diagram example



DOY Structured Text example



```
DOY(
BOOL DOY(DOYDATA TimeA, DOYDATA TimeB, DOYDATA TimeC, DOYDATA TimeD)
Turn on output when real-time clock value is within year range.
```



```
output := DOY(A, B, C, D);
```

(* ST Equivalence: *)

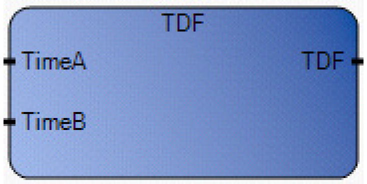
TESTOUTPUT := DOY(TIMEA1, TIMEB1, TIMEC1, TIMED1) ;

TDF (time difference)

Compute the time difference between TimeA and TimeB.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

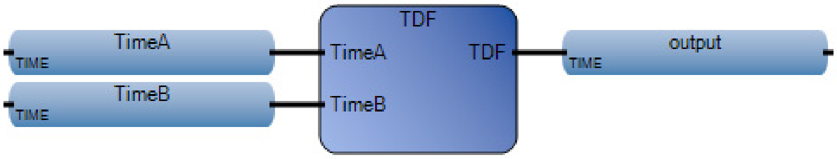
This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



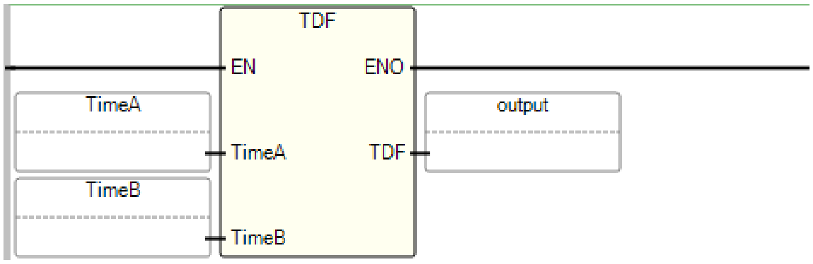
Use this table to help determine the parameter values for this instruction.

Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. TRUE - perform current computation. FALSE - there is no computation. Applies to Ladder Diagram programs.
TimeA	Input	TIME	The start time for time difference computation.
TimeB	Input	TIME	The end time for time difference computation.
ENO	Output	BOOL	Enable output. Applies to Ladder Diagram programs.
TDF	Output	TIME	The time difference for the two time inputs. TDF is name or PIN ID

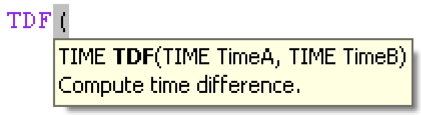
TDF Function Block Diagram example



TDF Ladder Diagram example

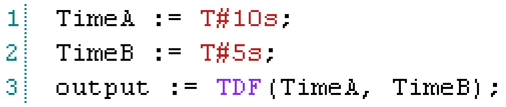


TDF Structured Text example



```
TDF (
```

TIME **TDF**(TIME TimeA, TIME TimeB)
Compute time difference.

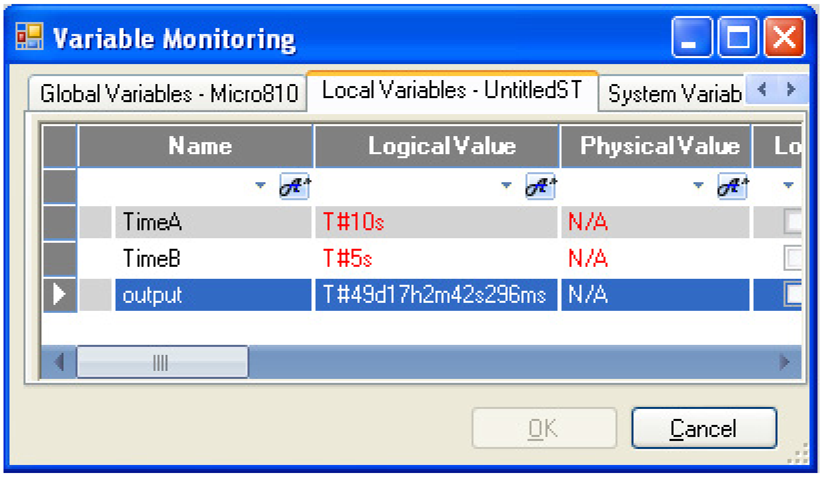


```
TimeA  := T#10s;
TimeB  := T#5s;
output  := TDF(TimeA, TimeB);
```

(* ST Equivalence: *)

TESTTIMEDIFF := TDF(TESTTIME1, TESTTIME2) ;

Results



Name	Logical Value	Physical Value	Lo
TimeA	T#10s	N/A	□
TimeB	T#5s	N/A	□
output	T#49d17h2m42s296ms	N/A	□

TOW (check week for real-time clock)

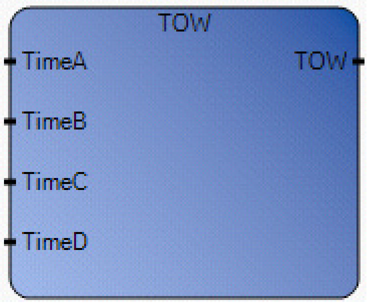
Turns on an output if the value of the real-time clock (RTC) is in the range of the Time of Week setting.

Operation details:

- If an RTC is not present, the output is always off.
- Configure the Time input parameters with valid ranges as specified in the TOWDATA Data Type. An invalid value faults the controller when TimeX.Enable is set to TRUE and an RTC is present and enabled.

Languages supported: Function Block Diagram, Ladder Diagram, Structured Text.

This instruction applies to the Micro810, Micro820, Micro830, Micro850, Micro870 controllers and Micro800 Simulator.



Use this table to help determine the parameter values for this instruction.

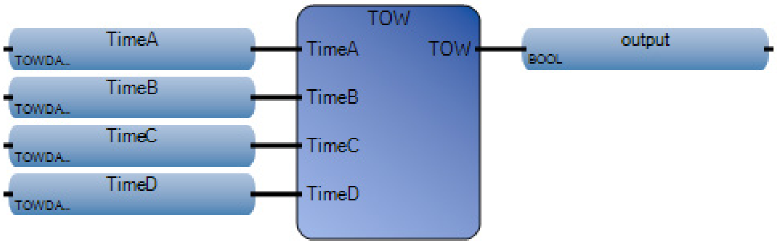
Parameter	Parameter Type	Data Type	Description
EN	Input	BOOL	Instruction enable. When EN = TRUE, perform the operation. When EN = FALSE, do not perform the operation.
TimeA	Input	TOWDATA	Day Time Setting of Channel A. Use the TOWDATA Data Type to define TimeA.
TimeB	Input	TOWDATA	Day Time Setting of Channel B. Use the TOWDATA Data Type to define TimeB.
TimeC	Input	TOWDATA	Day Time Setting of Channel C. Use the TOWDATA Data Type to define TimeC.
TimeD	Input	TOWDATA	Day Time Setting of Channel D. Use the TOWDATA Data Type to define TimeD.
TOW	Output	BOOL	If TRUE, the value of the real-time clock is in the range of the Day Time setting of any one of four channels.

TOWDATA Data Type

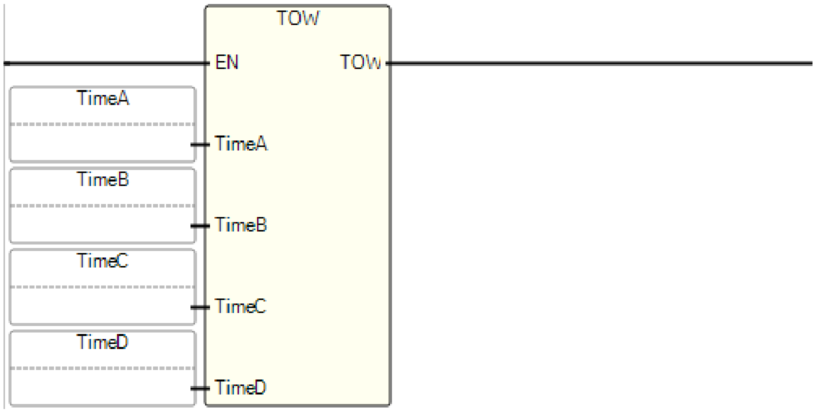
The following table describes the TOWDATA data type:

Parameter	Data Type	Description
Enable	BOOL	TRUE: Enable; FALSE: Disable.
DailyWeekly	BOOL	Type of Timer (0:daily timer; 1:weekly timer).
DayOn	USINT	Day of Week On value (must be in set [0...6]).
HourOn	USINT	Hour On value (must be in set [0...23]).
MinOn	USINT	Minute On value (must be in set [0...59]).
DayOff	USINT	Weekday Off value (must be in set [0...6]).
HourOff	USINT	Hour Off value (must be in set [0...23]).
MinOff	USINT	Minute Off value (must be in set [0...59]).

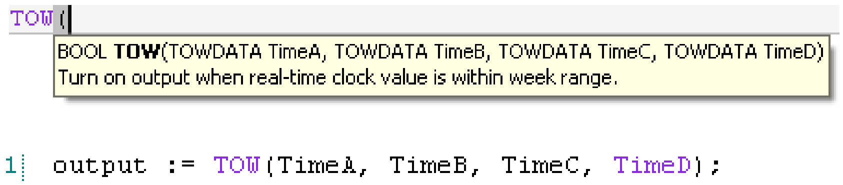
TOW Function Block Diagram example



TOW Ladder Diagram example



TOW Structured Text example



```
TOW(
```

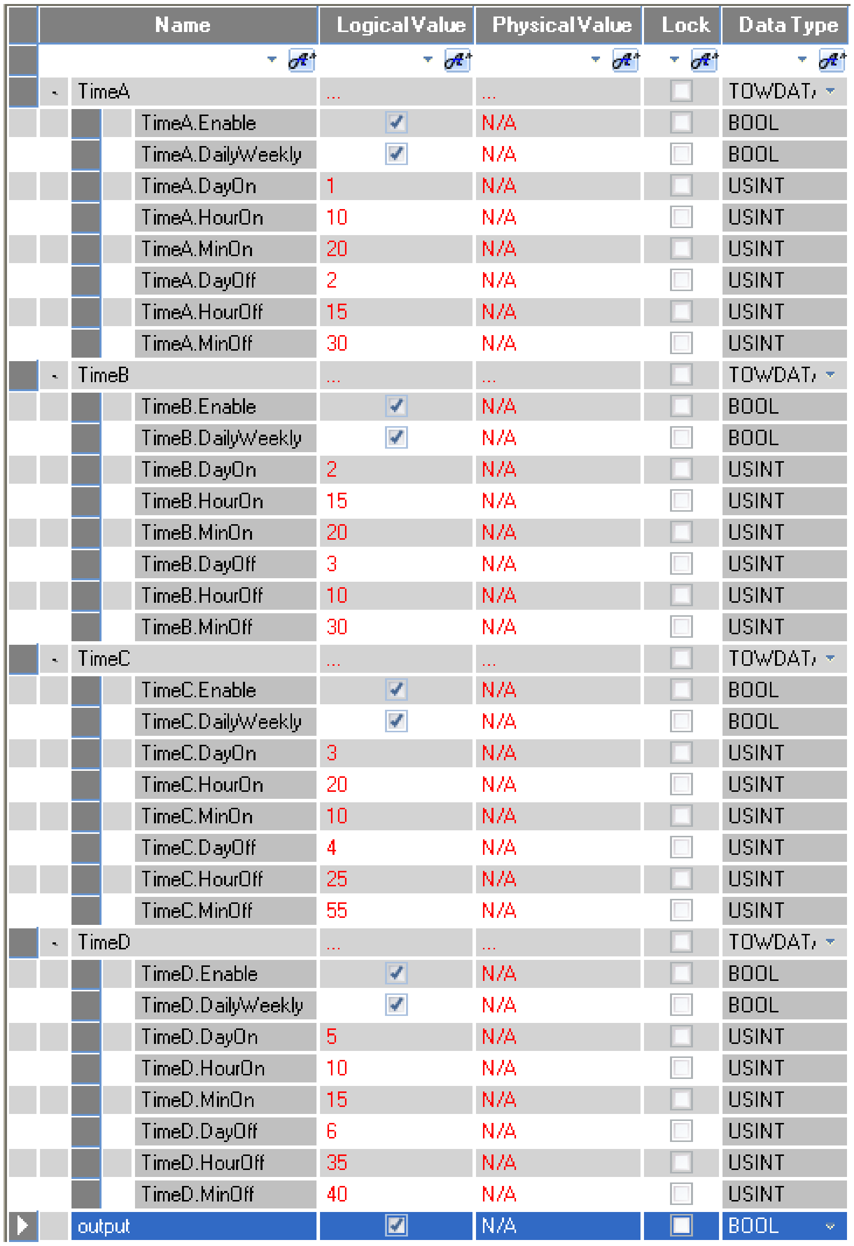
BOOL **TOW**(TOWDATA TimeA, TOWDATA TimeB, TOWDATA TimeC, TOWDATA TimeD)
Turn on output when real-time clock value is within week range.

```
output := TOW(TimeA, TimeB, TimeC, TimeD);
```

(* ST Equivalence: *)

TESTOUTPUT := TOW(TIMEA, TIMEB, TIMEC, TIMED) ;

Results



Name	Logical Value	Physical Value	Lock	Data Type
TimeA	...	...	☐	TOWDAT
TimeA.Enable	☑	N/A	☐	BOOL
TimeA.DailyWeekly	☑	N/A	☐	BOOL
TimeA.DayOn	1	N/A	☐	USINT
TimeA.HourOn	10	N/A	☐	USINT
TimeA.MinOn	20	N/A	☐	USINT
TimeA.DayOff	2	N/A	☐	USINT
TimeA.HourOff	15	N/A	☐	USINT
TimeA.MinOff	30	N/A	☐	USINT
TimeB	...	...	☐	TOWDAT
TimeB.Enable	☑	N/A	☐	BOOL
TimeB.DailyWeekly	☑	N/A	☐	BOOL
TimeB.DayOn	2	N/A	☐	USINT
TimeB.HourOn	15	N/A	☐	USINT
TimeB.MinOn	20	N/A	☐	USINT
TimeB.DayOff	3	N/A	☐	USINT
TimeB.HourOff	10	N/A	☐	USINT
TimeB.MinOff	30	N/A	☐	USINT
TimeC	...	...	☐	TOWDAT
TimeC.Enable	☑	N/A	☐	BOOL
TimeC.DailyWeekly	☑	N/A	☐	BOOL
TimeC.DayOn	3	N/A	☐	USINT
TimeC.HourOn	20	N/A	☐	USINT
TimeC.MinOn	10	N/A	☐	USINT
TimeC.DayOff	4	N/A	☐	USINT
TimeC.HourOff	25	N/A	☐	USINT
TimeC.MinOff	55	N/A	☐	USINT
TimeD	...	...	☐	TOWDAT
TimeD.Enable	☑	N/A	☐	BOOL
TimeD.DailyWeekly	☑	N/A	☐	BOOL
TimeD.DayOn	5	N/A	☐	USINT
TimeD.HourOn	10	N/A	☐	USINT
TimeD.MinOn	15	N/A	☐	USINT
TimeD.DayOff	6	N/A	☐	USINT
TimeD.HourOff	35	N/A	☐	USINT
TimeD.MinOff	40	N/A	☐	USINT
output	☑	N/A	☐	BOOL

Index

Rockwell Automation support

Use these resources to access support information.

Technical Support Center	Find help with how-to videos, FAQs, chat, user forums, and product notification updates.	rok.auto/support
Knowledgebase	Access Knowledgebase articles.	rok.auto/knowledgebase
Local Technical Support Phone Numbers	Locate the telephone number for your country.	rok.auto/phonesupport
Literature Library	Find installation instructions, manuals, brochures, and technical data publications.	rok.auto/literature
Product Compatibility and Download Center (PCDC)	Get help determining how products interact, check features and capabilities, and find associated firmware.	rok.auto/pcdc

Documentation feedback

Your comments help us serve your documentation needs better. If you have any suggestions on how to improve our content, complete the form at rok.auto/docfeedback.

Waste Electrical and Electronic Equipment (WEEE)

At the end of life, this equipment should be collected separately from any unsorted municipal waste.

Rockwell Automation maintains current product environmental information on its website at rok.auto/pec.

Allen-Bradley, expanding human possibility, Logix, Rockwell Automation, and Rockwell Software are trademarks of Rockwell Automation, Inc.

EtherNet/IP is a trademark of ODVA, Inc.

Trademarks not belonging to Rockwell Automation are property of their respective companies.

Rockwell Otomayson Ticaret A.Ş. Kar Plaza İş Merkezi E Blok Kat:6 34752, İçerenkÖy, İstanbul, Tel: +90 (216) 5698400 EEE YÖnetmeliğine Uygundur

Connect with us.

rockwellautomation.com — expanding **human possibility**™

AMERICAS: Rockwell Automation, 1201 South Second Street, Milwaukee, WI 53204-2496 USA, Tel: (1) 414.382.2000, Fax: (1) 414.382.4444
EUROPE/MIDDLE EAST/AFRICA: Rockwell Automation NV, Pegasus Park, De Kleetlaan 12a, 1831 Diegem, Belgium, Tel: (32) 2 663 0600, Fax: (32) 2 663 0640
ASIA PACIFIC: Rockwell Automation, Level 14, Core F, Cyberport 3, 100 Cyberport Road, Hong Kong, Tel: (852) 2887 4788, Fax: (852) 2508 1846

Supersedes Publication 2080-RM001J-EN-E - March 2021