SoMachine

Modbus and ASCII Read/Write Functions PLCCommunication Library Guide

04/2012

EIO0000000361.06

www.schneider-electric.com

Schneider Electric

The information provided in this documentation contains general descriptions and/or technical characteristics of the performance of the products contained herein. This documentation is not intended as a substitute for and is not to be used for determining suitability or reliability of these products for specific user applications. It is the duty of any such user or integrator to perform the appropriate and complete risk analysis, evaluation and testing of the products with respect to the relevant specific application or use thereof. Neither Schneider Electric nor any of its affiliates or subsidiaries shall be responsible or liable for misuse of the information contained herein. If you have any suggestions for improvements or amendments or have found errors in this publication, please notify us.

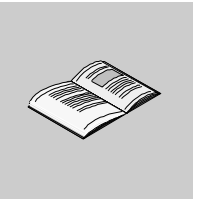
All pertinent state, regional, and local safety regulations must be observed when installing and using this product. For reasons of safety and to help ensure compliance with documented system data, only the manufacturer should perform repairs to components.

When devices are used for applications with technical safety requirements, the relevant instructions must be followed.

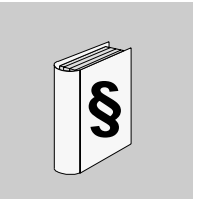
Failure to use Schneider Electric software or approved software with our hardware products may result in injury, harm, or improper operating results.

Failure to observe this information can result in injury or equipment damage.

Table of Contents



Safety Information



Important Information

NOTICE

Read these instructions carefully, and look at the equipment to become familiar with the device before trying to install, operate, or maintain it. The following special messages may appear throughout this documentation or on the equipment to warn of potential hazards or to call attention to information that clarifies or simplifies a procedure.

The addition of this symbol to a Danger safety label indicates that an electrical hazard exists, which will result in personal injury if the instructions are not followed.

This is the safety alert symbol. It is used to alert you to potential personal injury hazards. Obey all safety messages that follow this symbol to avoid possible injury or death.

⚠ DANGER

DANGER indicates an imminently hazardous situation which, if not avoided, **will result in** death or serious injury.

⚠ WARNING

WARNING indicates a potentially hazardous situation which, if not avoided, **can result in** death or serious injury.

⚠ CAUTION

CAUTION indicates a potentially hazardous situation which, if not avoided, **can result in** minor or moderate injury.

NOTICE

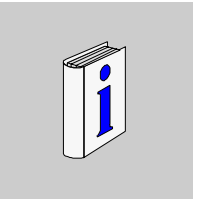
NOTICE is used to address practices not related to physical injury.

PLEASE NOTE

Electrical equipment should be installed, operated, serviced, and maintained only by qualified personnel. No responsibility is assumed by Schneider Electric for any consequences arising out of the use of this material.

A qualified person is one who has skills and knowledge related to the construction and operation of electrical equipment and its installation, and has received safety training to recognize and avoid the hazards involved.

About the Book



At a Glance

Document Scope

This document describes the PLCCommunication library for controllers.

Validity Note

This document has been updated with the release of SoMachine V3.1.

Product Related Information

⚠ WARNING

LOSS OF CONTROL

- The designer of any control scheme must consider the potential failure modes of control paths and, for certain critical control functions, provide a means to achieve a safe state during and after a path failure. Examples of critical control functions are emergency stop and overtravel stop, power outage and restart.
- Separate or redundant control paths must be provided for critical control functions.
- System control paths may include communication links. Consideration must be given to the implications of unanticipated transmission delays or failures of the link.
- Observe all accident prevention regulations and local safety guidelines.[1]
- Each implementation of this equipment must be individually and thoroughly tested for proper operation before being placed into service.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

[1] For additional information, refer to NEMA ICS 1.1 (latest edition), "Safety Guidelines for the Application, Installation, and Maintenance of Solid State Control" and to NEMA ICS 7.1 (latest edition), "Safety Standards for Construction and Guide for Selection, Installation and Operation of Adjustable-Speed Drive Systems" or their equivalent governing your particular location.

User Comments

We welcome your comments about this document. You can reach us by e-mail at techcomm@schneider-electric.com.

Communication Principles

1

Introduction

The communication function blocks for the controller are located in the PLCCommunication library. This library is automatically added to the library manager when a controller with Ethernet connectivity is added to your project or when a Modbus or ASCII manager is added to a controller's serial line.

What's in this Chapter?

This chapter contains the following topics:

Communication Functions on Controllers

Introduction

This topic describes management and operations of the controllers' communication functions. The functions facilitate communications between specific devices. Most of the functions are dedicated to Modbus exchanges. One function (SEND_RECV_MSG) is used by an ASCII manager to manage the exchange of data between devices operating on other protocols than Modbus.

NOTE: Communication functions are processed asynchronously with regard to the application task that called the function.

NOTE: This library is not supported by the ATV IMC Drive Controller.

NOTE: Do not use function blocks of the PLCCommunication library on a serial line with a Modbus IOScanner configured. This disrupts the Modbus IOScanner exchange.

Available Function Blocks

This table describes the communication function blocks available to controllers:

Function	Description
ADDM *(see page 24)*	This function takes the destination address of an external device and converts its string representation to an ADDRESS structure.
READ_VAR *(see page 30)*	This function reads standard bits or registers from a Modbus device.
WRITE_VAR *(see page 32)*	This function writes standard bits or registers to a Modbus device.
WRITE_READ_VAR *(see page 34)*	This function reads and writes standard bits or registers on Modbus devices.
SINGLE_WRITE *(see page 36)*	This function writes a single register to an external device.
SEND_RECV_MSG *(see page 38)*	This function sends and receives user defined messages on selected media for example, a serial line (not supported by XBT GC, XBT GT and XBT GK).

Generic Parameters

Introduction

This topic describes the management and operations of the controllers' communication functions using the `READ_VAR` function block as an example. (The PLCopen standard defines rules for function blocks.)

NOTE: These parameters are common to all PLCCommunication function blocks (except ADDM).

Graphical Representation

The parameters that are common to all function blocks in the PLCCommunication library are highlighted in this graphic:

Common Parameters

These parameters are shared by several function blocks in the PLCCommunication library:

Input	Type	Comment
`Execute`	BOOL	The function is executed on the rising edge of this input. **NOTE:** When `xExecute` is set to TRUE at the first task cycle in RUN after a cold or warm reset, the rising edge is not detected.
`Abort`	BOOL	Aborts the ongoing operation at the rising edge
`Addr`	ADDRESS	Address of the targeted external device (can be the output of the ADDM function block *(see page 24)*)
`Timeout`	WORD	Exchange timeout is a multiple of 100 ms (0 for infinite)
NOTE: A function block operation may require several exchanges. The timeout applies to each exchange between the controller and the modem, so the global duration of the function block might be longer than the timeout.		

Output	Type	Comment
Done	BOOL	Done is set to TRUE when the function is completed successfully.
Busy	BOOL	Busy is set to TRUE while the function is ongoing.
Aborted	BOOL	Aborted is set to TRUE when the function is aborted with the Abort input. When the function is aborted, CommError contains the code Canceled - 16#02 (exchange stopped by a user request).
Error	BOOL	Error is set to TRUE when the function stops because of a detected error. When there is a detected error, CommError and OperError contain information about the detected error.
CommError	BYTE	CommError contains communication error codes *(see page 16)*.
OperError	DWORD	OperError contains operation error codes *(see page 17)*.

NOTE:

As soon as the Busy output is reset to 0, one (and only one) of these 3 outputs is set to 1:

- Done
- Error
- Aborted

Function blocks require a rising edge in order to be initiated. The function block needs to first see the Execute input as false in order to detect a subsequent rising edge.

⚠ WARNING

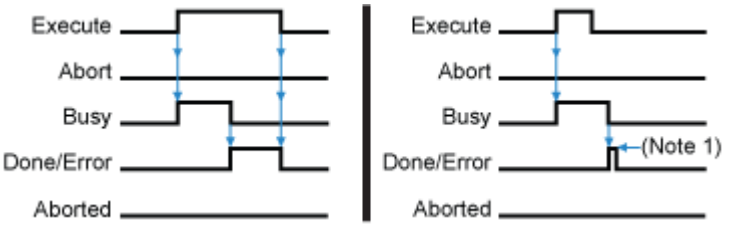
UNINTENDED EQUIPMENT OPERATION

Always make the first call to a function block with its Execute input set to FALSE so that it will detect a subsequent rising edge.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Function Execution

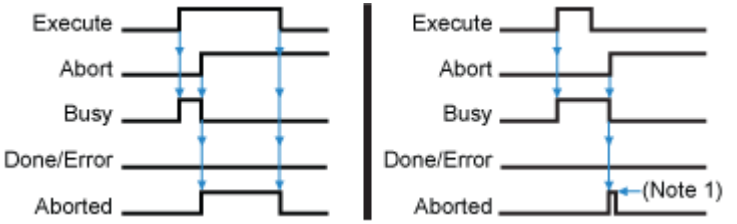
The function starts at the rising edge of the `Execute` input. The `Busy` output is then set to `TRUE`. This figure shows the function block's behavior when the operation is automatically completed (with or without detected errors):



Note 1: The `Done` or `Error` bit is set to `TRUE` during a task cycle only if `Execute` has already been reset to `FALSE` when the operation ended.

Function Aborted

This figure shows the function being aborted by the application. The rising edge of the `Abort` input cancels the ongoing function. In such cases, the aborted output is set to 1 and `CommError` contains the code `Canceled - 16#02` (exchange stopped by a user request):



Note 1: The `Abort` bit is set to `TRUE` during a task cycle only if `Execute` has already been reset to `FALSE` when the abort request occurred.

Data Types

2

Introduction

This chapter describes the data types used by the PLCCommunication library.

What's in this Chapter?

This chapter contains the following topics:

CommunicationErrorCodes: Communication Error Codes

Enumerated Type Description

The `CommunicationErrorCodes` enumerated type contains information about communication diagnostics, such as interruptions and detected errors. It contains these values:

Enumerator	Value (hex)	Description
`CommunicationOK`	`00`	The exchange is valid.
`TimedOut`	`01`	The exchange stopped when the timeout expired.
`Canceled`	`02`	The exchange was stopped by a user request (the `Abort` command).
`BadAddress`	`03`	The address format is incorrect.
`BadRemoteAddr`	`04`	The remote address is incorrect.
`BadMgtTable`	`05`	The management table format is incorrect.
`BadParameters`	`06`	Specific parameters are incorrect.
`ProblemSendingRq`	`07`	There was a problem while sending the request to the destination.
`RecvBufferTooSmall`	`09`	The reception buffer size is too small.
`SendBufferTooSmall`	`0A`	The transmission buffer size is too small.
`SystemRessourceMissing`	`0B`	A system resource is missing.
`BadTransactionNb`	`0C`	The transaction number is incorrect.
`BadLength`	`0E`	The length is incorrect.
`ProtocolSpecificError`	`FE`	The operation error code contains a protocol-specific code.
`Refused`	`FF`	The message was refused.

OperationErrorCodes: Operation Error Codes

Enumerated Type Description

The `OperationErrorCodes` enumerated type contains codes that correspond to detected errors.

00

When the communication error code is 00 hex (correct transaction), the `OperationErrorCodes` enumerated type can return these values:

Enumerator	Value (hex)	Description
`OperationOK`	`00`	The exchange is valid.
`NotProcessed_or_TargetResourceMissing`	`01`	The request has not been processed.
`BadResponse`	`02`	The received response is incorrect.

FF

When the communication error code is FF hex (message refused), the `OperationErrorCodes` enumerated type can return these values:

Enumerator	Value (hex)	Description
`NotProcessed_or_TargetResourceMissing`	`01`	The target system's resource is absent.
`BadLength`	`05`	The length is incorrect.
`CommChannelErr`	`06`	The communication channel is associated with a detected error.
`BadAddr`	`07`	The address is incorrect.
`SystemResourceMissing`	`0B`	A system resource is missing.
`TargetCommInactive`	`0C`	A target communication function is not active.
`TargetMissing`	`0D`	The target is absent.
`ChannelNotConfigured`	`0F`	The channel is not configured.

FE

When the communication error code is FE hex, the `OperationErrorCodes` enumerated type contains the protocol-specific error detection code. (Refer to your specific protocol's error detection codes.)

LinkNumber: Communication Port Number

Enumerated Type Description

The `LinkNumber` enumerated data type is a list of the available communication ports. It contains these values:

Enumerator	Value (hex)	Description
`USBConsole`	`00`	USB port not available for communication exchanges
`COM1`	`01`	Serial COM 1 (embedded serial link)
`COM2`	`02`	Serial COM 2
`EthEmbed`	`03`	Embedded Ethernet link
`CANEmbed`	`04`	Embedded CANopen link
`COM3`	`05`	Serial COM 3

If one serial PCI module is installed then the serial PCI module link is COM2, regardless of the physical PCI slots used.

If two serial PCI modules are installed then the serial PCI module plugged on the left side PCI slots is COM2 and the serial PCI module plugged on the right side PCI slots is COM3.

ObjectType: Available Object Types to be Read/Written

Enumerated Type Description

The `ObjectType` enumerated data type contains the object types that are available for reading and/or writing.

The table below lists the data type values, the corresponding object types and the Modbus request function codes associated with each function block:

			Read / Write Functions and associated Modbus Request Function Code			
Enu-merator	**Value (hex)**	**Object Type)**	**READ_VAR**	**WRITE_VAR**	**SINGLE_WRITE**	**WRITE_READ_VAR**
`MW`	`00`	Holding Register (16 bits)	#3 (Read Holding Registers)	#16 (Write Multiple Registers)	#6 (Write Single Register)	#23 (Write-Read Multiple Registers)
`I`	`01`	Digital Input (1 bit)	#2 (Read Digital Inputs)	—	—	—
`Q`	`02`	Internal bit or Digital output (coil) (1 bit)	#1 (Read Coils)	#15 (Write Multiple Coils)	—	—
`IW`	`03`	Input register (16 bits)	#4 (Read Input Registers)	—	—	—

ADDRESS: External Device Address

Structure Description

The ADDRESS data structure contains the external device address. It contains these variables:

Variable	Type	Description
_Type	BYTE	Reserved
_CliID	BYTE	Reserved
Rack	BYTE	Rack number (always 0)
Module	BYTE	Module number (always 0)
Link	LinkNumber *(see page 18)*	Communication port number
_ProtId	BYTE	Reserved (0 for Modbus)
AddrLen	BYTE	Length of Unitld and AddrExt variables (in bytes)
UnitId	BYTE	Equipment number (e.g., Modbus slave address)
AddrExt	ADDR_EXT *(see page 21)*	Contains an address extension as an array or as a specific structure

ADDR_EXT: Address Extension

Union Description

ADDR_EXT is a UNION data type that contains an address extension as an array or as a specific structure for the TCP/IP address. It contains these variables:

Variable	Type	Description
as_array	ARRAY [0...7] OF BYTE	Reserved (open for other protocol addressing)
TcpAddr	TCP_ADDR_EXT	The specific structure for TCP remote devices

TCP_ADDR_EXT: Address Extension for TCP Devices

Structure Description

The `TCP_ADDR_EXT` structure data type contains the address extension for TCP external devices. It contains these variables:

Variable	Type	Description
`A`	BYTE	First value in IP address A.B.C.D
`B`	BYTE	Second value in IP address A.B.C.D
`C`	BYTE	Third value in IP address A.B.C.D
`D`	BYTE	Last value in IP address A.B.C.D
`port`	WORD	TCP port number (Modbus default: 502)

Function Block Descriptions

3

Introduction

This chapter describes the function blocks in the PLCCommunication library.

What's in this Chapter?

This chapter contains the following topics:

ADDM: Convert a String Into an Address

Function Description

The ADDM function block converts a destination address that is represented as a string to an ADDRESS structure that can be used as an entry in a communication function block.

Graphical Representation

ADDM-Specific Parameter Descriptions

Input/Output	Type	Comment
AddrTable	ADDRESS	This is the ADDRESS structure to be filled by the function block.

Input	Type	Comment
Execute	BOOL	Executes the function at the rising edge
Addr	STRING	Address in STRING type to be converted in ADDRESS type (see details below)

Output	Type	Comment
Done	BOOL	Done is set to TRUE when the function is completed successfully. **NOTE:** When the operation is aborted with the Abort input, Done is not set to 1 (only Aborted).
Error	BOOL	Error is set to TRUE when the function stops because of a detected error. When there is a detected error, CommError and OperError contain information about the detected error.
CommError	BYTE	CommError contains communication error codes *(see page 16)*.

NOTE: A rising edge on the Execute input executes the conversion and returns an immediate update of AddrTable. However, AddrTable retains the last value when an error is detected (that is, when the Addr string is not correct).

Function blocks require a rising edge in order to be initiated. The function block needs to first see the Execute input as false in order to detect a subsequent rising edge.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

Always make the first call to a function block with its `Execute` input set to `FALSE` so that it will detect a subsequent rising edge.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Addr STRING for ASCII Address Format

For ASCII addressing, only the communication port number is requested:

`'<communication port number>'`

For example, to send a user defined message on serial line 2, use the string `'2'`.

This table defines the fields in the ADDM output for the ASCII address format:

Field	Type	Value	Example
`_Type`	BYTE	Reserved	not used
`_CliID`	BYTE	Reserved	not used
`Rack`	BYTE	Rack number (always 0)	0
`Module`	BYTE	Module number (always 0)	0
`Link`	LinkNumber *(see page 18)*	`<communication port number>`	2
`_ProtId`	BYTE	Not used	not used
`AddrLen`	BYTE	0	0
`UnitId`	BYTE	Not used	not used
`AddrExt`	ADDR_EXT	Not used	not used

Addr STRING for Modbus Serial Address Format

For Modbus serial addressing, use the communication port and the destination slave address (0 to 247), separated by a dot: `'<communication port number>.<slave address>'`

For example, send a message to slave 8 on serial line 2 with this syntax: `'2.8'`

The `ADDM` function fills the `AddrTable` input/output with these values:

Field	Type	Value	Example
`_Type`	BYTE	Reserved	not used
`_CliID`	BYTE	Reserved	not used
`Rack`	BYTE	Rack number (always 0)	0
`Module`	BYTE	Module number (always 0)	0
`Link`	LinkNumber *(see page 18)*	`<communication port number>`	2
`_ProtId`	BYTE	0 for Modbus	0
`AddrLen`	BYTE	1	1
`UnitId`	BYTE	`<slave address>`	8
`AddrExt`	ADDR_EXT	Not used	not used

Addr STRING for Modbus TCP Address Format

Address of a Modbus TCP Standard Slave

For the Modbus TCP standard slave address format, the communication port number (3 for the embedded Ethernet port) and the destination IP address {A.B.C.D} (offset with brackets) are requested:

`'<communication port number>{<IP address A.B.C.D>}'`

NOTE: A Modbus TCP standard slave uses Modbus address 255 (the `UnitId` default value). However, a Modbus TCP device may have different value (for example, a Tesys has Modbus address 1). In this case, add the `UnitId` value.

TCP port 502 is used by default. It's possible to use a non-standard port by adding the requested port number to the IP address:

`'<communication port number A.B.C.D>{<IP address>:<port>}'`

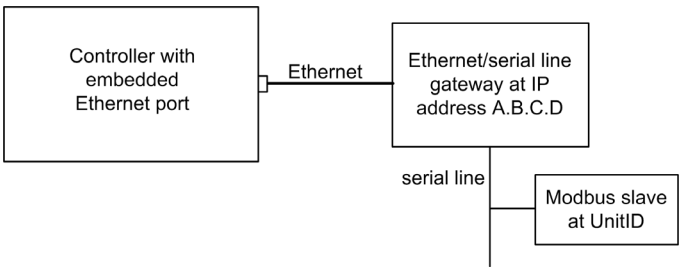
For example, to send a message at Modbus TCP slave IP address 192.168.1.2 using standard TCP port 502, use this string: `'3{192.168.1.2}'`

The ADDM function fills the AddrTable input/output with these values:

Field	Type	Value	Example
_Type	BYTE	Reserved	not used
_CliID	BYTE	Reserved	not used
Rack	BYTE	Rack number	0
Module	BYTE	Module number	0
Link	LinkNumber *(see page 18)*	<communication port number>	3
_ProtId	BYTE	0 for Modbus	0
AddrLen	BYTE	UnitID + AdrExt length in bytes	7
UnitId	BYTE	Modbus address (255 by default)	255
AddrExt	TCP_ADDR_EXT	A	192
		B	168
		C	1
		D	2
		<port> (default = 502)	502

Address of a Modbus Serial Slave Through Ethernet/Serial Line Gateway

It's also possible to address a Modbus slave through an Ethernet/serial line gateway:



The request includes the communication port number, gateway IP address {A.B.C.D} (offset with brackets with or without TCP port), and the Modbus serial slave address (UnitId parameter):

'<communication port number>{<IP address A.B.C.D>}<slave address>'

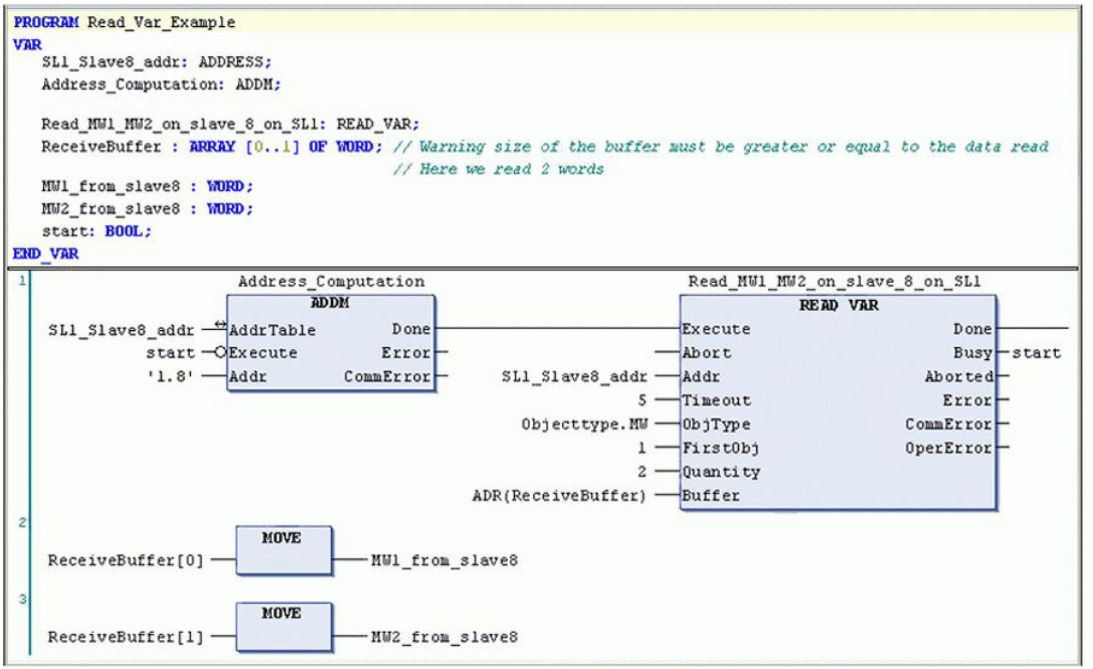
For example, to send a message at Modbus Serial slave address 5 through a Ethernet/serial line gateway at IP address 192.168.1.2 using standard TCP port 502, use this string: '3{192.168.1.2}5'

The ADDM function fills the AddrTable input/output with these values:

Field	Size	Value	Example
_Type	BYTE	Reserved	not used
_CliID	BYTE	Reserved	not used
Rack	BYTE	Rack number	0
Module	BYTE	Module number	0
Link	LinkNumber *(see page 18)*	<communication port number>	3
_ProtId	BYTE	0 for Modbus	0
AddrLen	BYTE	UnitID + AdrExt length in bytes	7
UnitId	BYTE	<Slave address>	5
AddrExt	TCP_ADDR_EXT	A	192
		B	168
		C	1
		D	2
		TCP port number (default = 502)	502

Example

This example shows the declaration and use of ADDM as an input to the READ_VAR function block. ADDM converts the address of slave 8 on serial line 1 from the string '1.8' to an ADDRESS type:



NOTE: The Busy output assigned to start allows for the continuous execution of READ_VAR. The start variable must be set to TRUE (by the user online or the application) after the first cycle to initiate continuous reading. This example does not show the management of exchange errors.

READ_VAR: Read Data from a Modbus Device

Function Description

The READ_VAR function block reads data from an external device in the Modbus protocol.

Graphical Representation

READ_VAR-Specific Parameter Descriptions

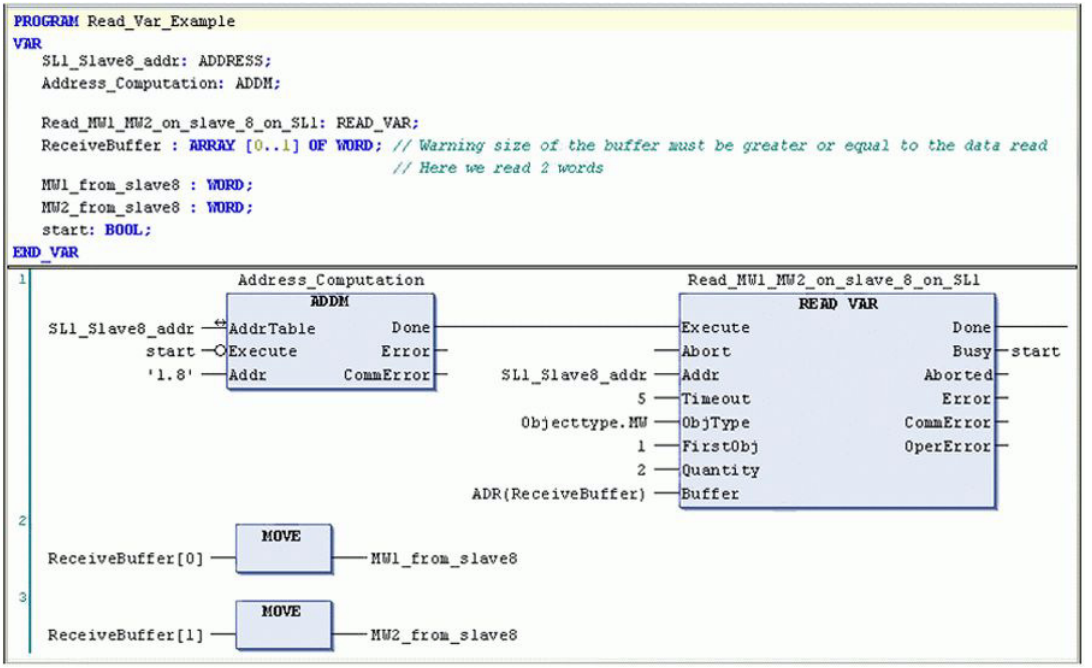
Input	Type	Comment
ObjType	ObjectType	ObjType is the type of object to be read (MW, I, IW, Q) *(see page 19)*.
FirstObj	DINT	FirstObj is the index of the first object to be read.
Quantity	UINT	Quantity is the number of objects to be read: • 1-125: registers (MW and IW types) • 1-2000: bits (I and Q types)
Buffer	POINTER TO BYTE	Buffer is the address of the buffer in which object values will be stored. The ADR standard function must be used to define the associated pointer. (See the example below.) The buffer is a table that receives the values that are read in the device. For example, the reading of 4 registers is stored in a table of 4 words and the reading of 32 bits requires a table of 2 words or 4 bytes, each bit of which is set to the corresponding value of the remote device.

The input and output parameters that are common to all PLCCommunication library function blocks are described elsewhere *(see page 11)*.

Example

This POU allows the reading of internal registers 1 and 2 (MW1 and MW2) of the Modbus slave with address 8 on serial line 1.

This figure shows the declaration and use of the `READ_VAR` function:



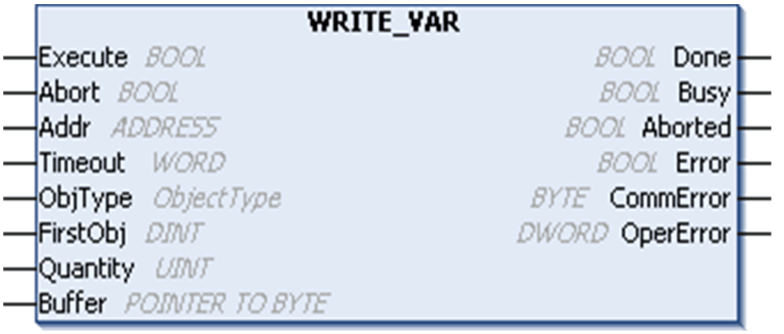
NOTE: The `Busy` output assigned to `start` allows for the continuous execution of ADDM and READ_VAR. The `start` variable must be set to TRUE (online by the user or by the application) after the first cycle to initiate continuous reading. This example does not show the management of exchange errors.

WRITE_VAR: Write Data to a Modbus Device

Function Description

The WRITE_VAR function block writes objects to an external device in the Modbus protocol.

Graphical Representation



WRITE_VAR-Specific Parameter Descriptions

Input	Type	Comment
ObjType	ObjectType	ObjType describes the type of object(s) to write (MW, Q) *(see page 19)*.
FirstObj	DINT	FirstObj is the index of the first object to write.
Quantity	UINT	Quantity is the number of objects to be read: • 1-123: registers (MW type) • 1-1968: bits (Q type)
Buffer	POINTER TO BYTE	Buffer is the address of the buffer in which object values will be stored. The ADR standard function must be used to define the associated pointer. (See the example below.) The buffer is a table that receives the values that have to be written in the device. For example, the written values of 4 registers are stored in a table of 4 words and the written values of 32 bits require a table of 2 words or 4 bytes, each bit of which is set to the corresponding value.

The input and output parameters that are common to all PLCCommunication library function blocks are described elsewhere *(see page 11)*.

⚠ WARNING

EXCHANGED DATA INCOMPATIBILITY

Verify that the exchanged data are compatible because data structure alignments are not the same for all devices.

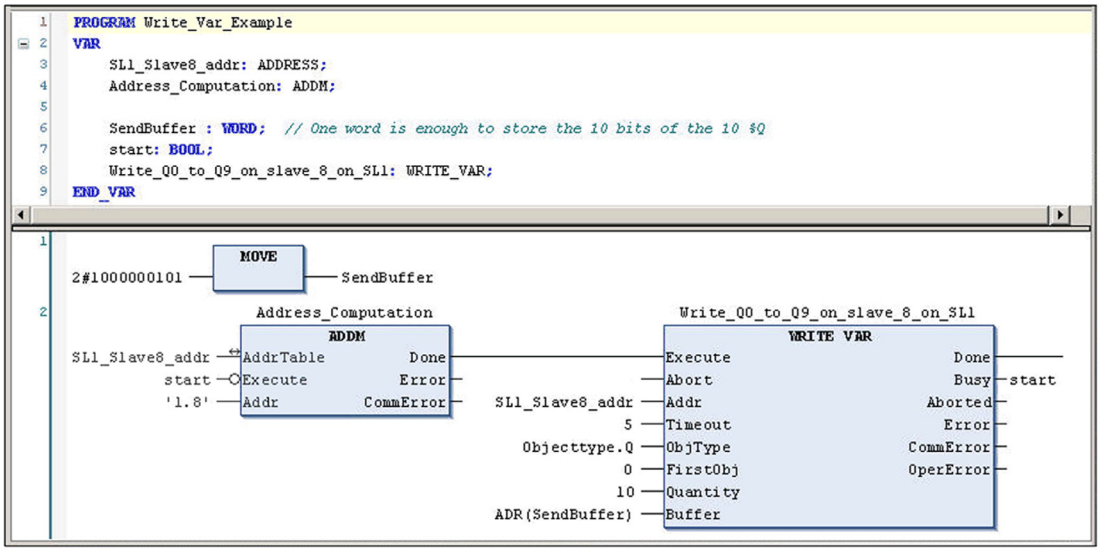
Failure to follow these instructions can result in death, serious injury, or equipment damage.

Example

This POU allows the writing to digital outputs/internal bits 0 to 9 (Q0 to Q9) of a Modbus slave at address 8 on serial line 1:

```
PROGRAM Write_Var_Example
VAR
    SL1_Slave8_addr: ADDRESS;
    Address_Computation: ADDM;

    SendBuffer : WORD;  // One word is enough to store the 10 bits of the 10 %Q
    start: BOOL;
    Write_Q0_to_Q9_on_slave_8_on_SL1: WRITE_VAR;
END_VAR
```



NOTE: The `Busy` output assigned to `start` allows for the continuous execution of ADDM and WRITE_VAR. The `start` variable must be set to TRUE (online by the user or by the application) after the first cycle to initiate continuous reading/writing. This example does not show the management of exchange errors.

WRITE_READ_VAR: Read and Write Internal Registers on a Modbus Device

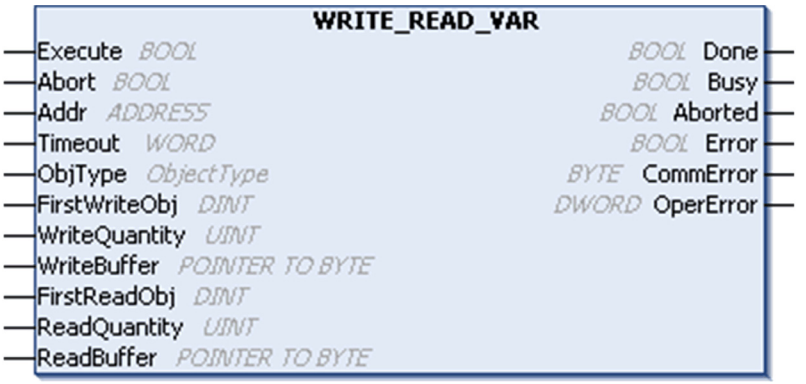
Function Description

This function reads and writes internal registers (MW type only) to an external device in the Modbus protocol. The read and a write operations are contained in a single transaction.

The write operation is performed first. The `Write_Read_Var` function can then:

- write consecutive internal registers and immediately read back their values for verification
- write some consecutive internal registers and read others in a single unique request

Graphical Representation



WRITE_READ_VAR-Specific Parameter Descriptions

Input	Type	Comment
`ObjType`	ObjectType *(see page 19)*	`ObjType` is the object type to be written and read (MW only).
`FirstWriteObj`	DINT	`FirstWriteObj` is the index of the first object to write.
`WriteQuantity`	UINT	`WriteQuantity` is the number of objects to write: ● 1-121: registers (MW type)
`WriteBuffer`	POINTER TO BYTE	`WriteBuffer` is the address of the buffer in which objects values are stored. The `ADR` standard function must be used to define the associated pointer. (See the example below.) The buffer is a table that receives the values that are written in the device.

Input	Type	Comment
FirstReadObj	DINT	ReadFirstObj is the index of the first object to be read.
ReadQuantity	UINT	ReadQuantity represents the number of objects to be read: ● 1-125: registers (MW type)
ReadBuffer	POINTER TO BYTE	ReadBuffer is the address of the buffer in which objects values will be stored. The ADR standard function must be used to define the associated pointer. (See the example below.) The buffer is a table that receives the values that are read in the device.

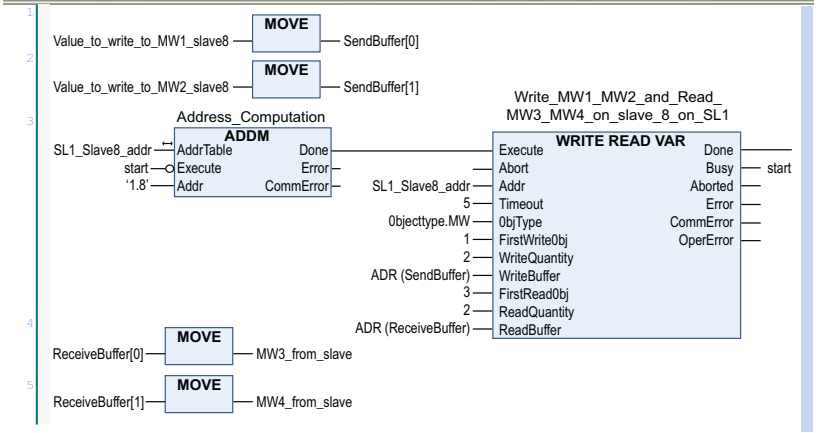
The input and output parameters that are common to all PLCCommunication library function blocks are described elsewhere *(see page 11)*.

Example

This POU allows the writing to internal registers 1 and 2 (MW1 and MW2) and the reading of internal registers 3 and 4 (MW3 and MW4) of a Modbus slave at address 8 on serial line 1:

```
PROGRAM Write_Read_Var_Example
VAR
   SL1_Slave8_addr: ADDRESS;
   Address_Computation: ADDM;

   SendBuffer : ARRAY [0..1] OF WORD;
   ReceiveBuffer : ARRAY [0..1] OF WORD;
   start: BOOL;
   Value_to_write_to_MW1_slave8: WORD;
   Value_to_write_to_MW2_slave8: WORD;
   MW3_from_slave8: WORD;
   MW4_from_slave8: WORD;
   Write_MW1_MW2_and_Read_MW3_MW4_on_slave_8_on_SL1: WRITE_READ_VAR;
END_VAR
```



NOTE: The Busy output assigned to start allows for the continuous execution of ADDM and WRITE_READ_VAR. The start variable must be set to TRUE (online by the user or by the application) after the first cycle to initiate continuous writing/reading. This example does not show the management of exchange errors.

SINGLE_WRITE: Write a Single Register to a Modbus Device

Function Description

The `SINGLE_WRITE` function block writes a single internal register to an external Modbus device.

Graphical Representation

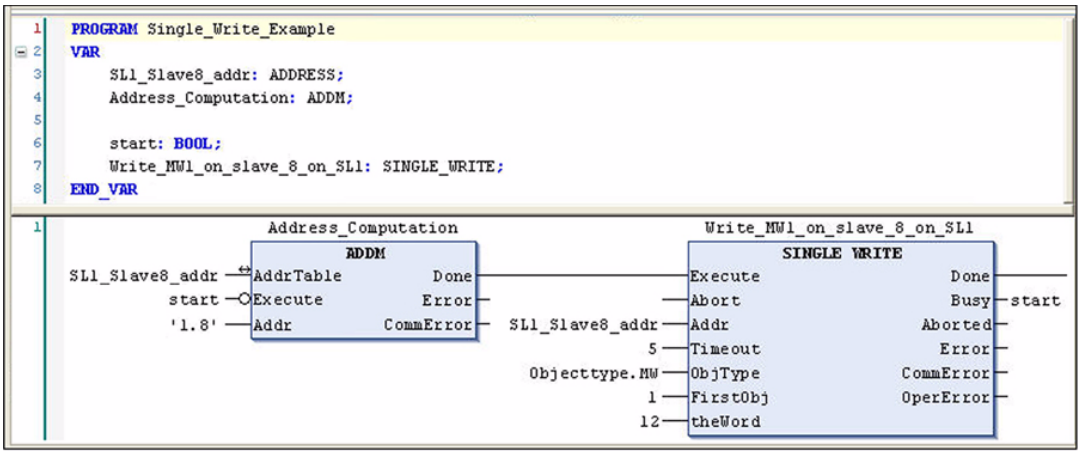
SINGLE_WRITE-Specific Parameter Descriptions

Input	Type	Comment
`ObjType`	ObjectType	`ObjType` describes the type of object(s) to write (MW only) *(see page 19)*.
`FirstObject`	DINT	`FirstObject` is the index of the object to write.
`theWord`	WORD	This input contains the value to write.

The input and output parameters that are common to all PLCCommunication library function blocks are described elsewhere *(see page 11)*.

Example

This POU allows the writing of value 12 to internal register 1 (MW1) of a Modbus slave at address 8 on serial line 1:



NOTE: The `Busy` output assigned to `start` allows for the continuous execution of ADDM and SINGLE_WRITE. The `start` variable must be set to TRUE (by the user online or the application) after the first cycle to initiate continuous reading/writing. This example does not show the management of exchange errors.

SEND_RECV_MSG: Send and/or Receive User Defined Messages

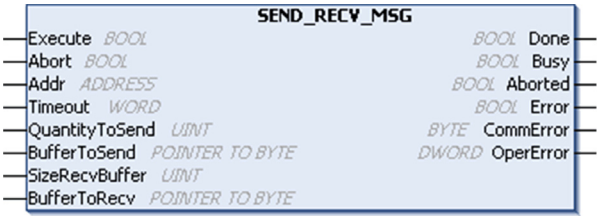
Function Description

The SEND_RECV_MSG function block sends and receives user defined messages. It sends a message on the selected media (for example, a serial line) and then waits for a response. It is also possible to either send without waiting for a response or to receive a message without sending one.

This function should be used with an ASCII manager. It could also be used with a Modbus manager if you want to send a request that is not implemented in the communication library. In this case, you have to build the request yourself.

The SEND_RECV_MSG function block is not supported by XBT GC, XBT GT and XBT GK

Graphical Representation



SEND_RECV_MSG-Specific Parameter Descriptions

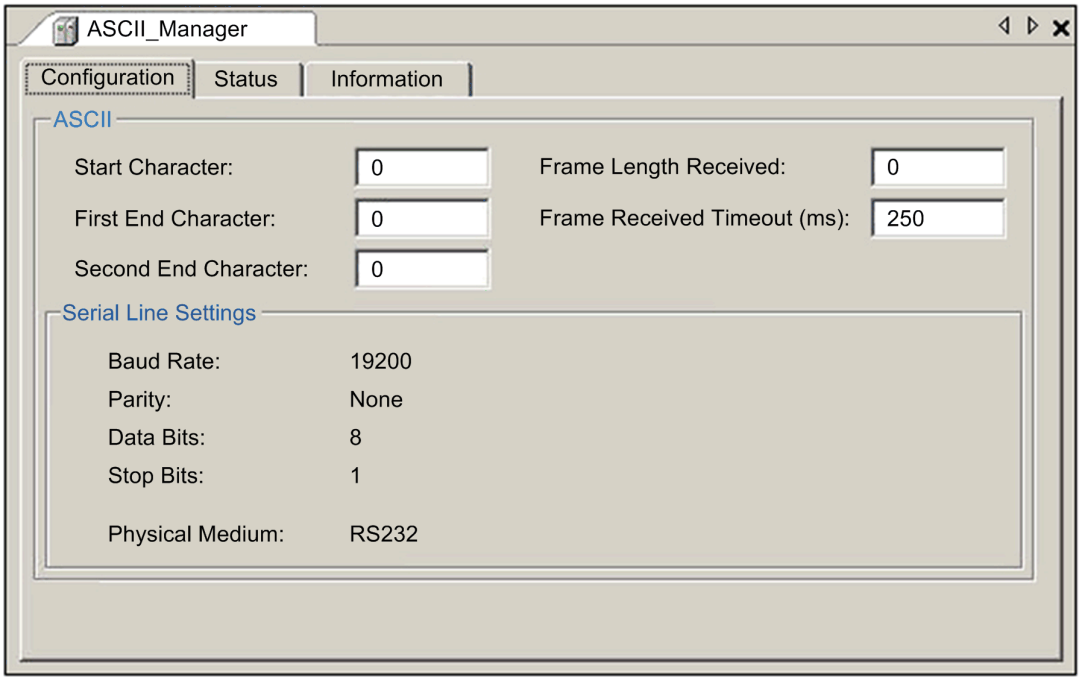
Input	Type	Comment
QuantityToSend	UINT	QuantityToSend is the number of bytes to send. Controller limitation: ● M238: 252 bytes ● M258/LMC058: 1050 bytes
BufferToSend	POINTER TO BYTE	BufferToSend is the address of the buffer (array of bytes) in which the message to send is stored. The ADR standard function must be used to define the associated pointer. (See the example below.) If 0, the function makes a receive-only.
SizeRecvBuffer	UINT	SizeRecvBuffer is the available size (in bytes) of the receive buffer. The size of the received data (in bytes) is available in the function block instance internal property (internal variable): <Instance Name>.NbRecvBytes. Controller limitation: ● M238: 252 bytes ● M258/LMC058: 1050 bytes
BufferToRecv	POINTER TO BYTE	BufferToRecv is the address of the buffer (array of SizeRecvBuffer bytes) in which the received message will be stored. The ADR standard function must be used to define the associated pointer. (See the example below.) If 0, the function makes a send-only.

For send only operations, the exchange is complete (`Busy` reset to 0) when all data (including eventual start and stop characters) have been sent to the line.

For a send/receive or receive only operation, the system receives characters until the ending condition. When the ending condition is reached, the exchange is complete (`Busy` reset to 0). Received characters are then copied into the receive buffer up to `sizeRecvBuffer` characters and the size of the received data (in bytes) is available in function block instance property (internal variable): <Instance Name>.`NbRecvBytes`. The `sizeRecvBuffer` input does not represent an ending condition.

The input and output parameters that are common to all PLCCommunication library function blocks are described elsewhere *(see page 11)*.

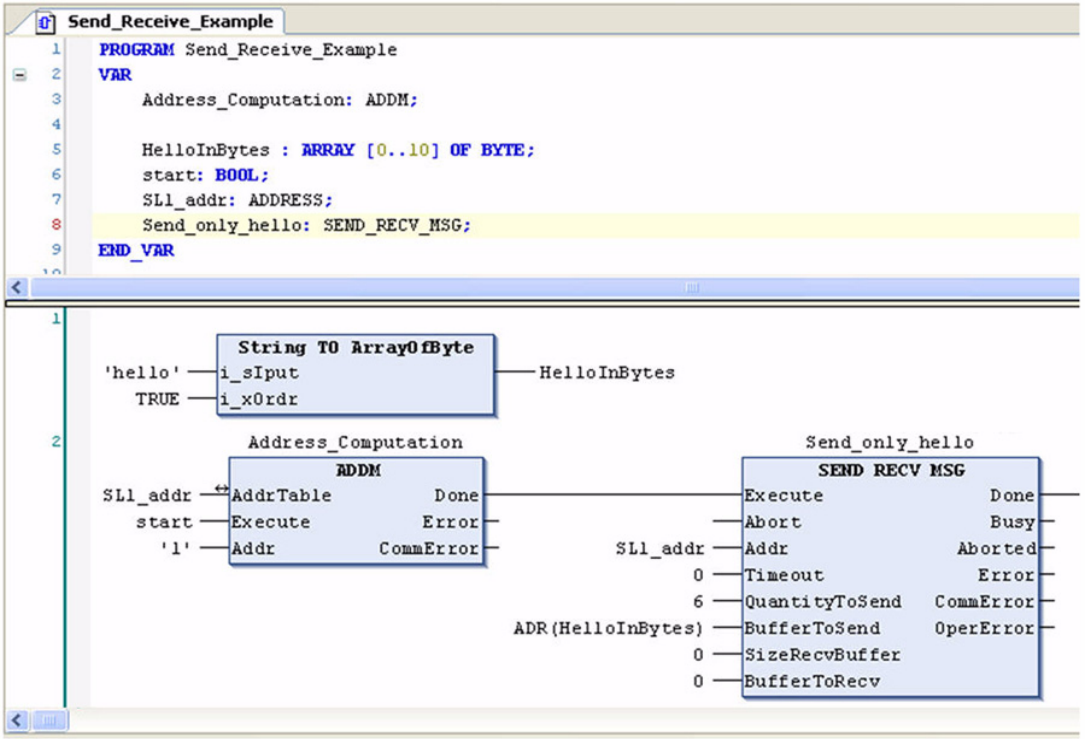
The starting and ending conditions of user defined messages are configured in the ASCII manager's configuration dialog box:



NOTE: There are no start and end characters in this example. The received frame ending condition is a 250ms Timeout.

Example

This POU allows the send-only of the user defined message “hello” on serial line 1:



NOTE: A rising edge on the `Start` variable launches the conversion of an address and the sending of the message.

Appendices

Function and Function Block Representation

A

Overview

Each function can be represented in the following languages:

- IL: Instruction List
- ST: Structured Text
- LD: Ladder Diagram
- FBD: Function Block Diagram
- CFC: Continuous Function Chart

This chapter provides functions and function blocks representation examples and explains how to use them for IL and ST languages.

What's in this Chapter?

This chapter contains the following topics:

Differences Between a Function and a Function Block

Function

A function:
- is a **POU** (Program Organization Unit) that returns one immediate result
- is directly called with its name (not through an **Instance**)
- has no persistent state from one call to the other
- can be used as an operand in other expressions

Examples: boolean operators (AND), calculations, conversion (BYTE_TO_INT)

Function Block

A function block:
- is a **POU** (Program Organization Unit) that returns one or more outputs
- is always called through an **Instance** (function block copy with dedicated name and variables)
- each **Instance** has a persistent state (outputs and internal variables) from one call to the other

Examples: timers, counters

In the example below, `Timer_ON` is an instance of the Function Block `TON`:

```
PROGRAM MyProgram_ST
VAR
    Timer_ON: TON;  // Function Block Instance
    Timer_RunCd: BOOL;
    Timer_PresetValue: TIME := T#5S;
    Timer_Output: BOOL;
    Timer_ElapsedTime: TIME;
END_VAR
```

```
Timer_ON(
    IN:=Timer_RunCd,
    PT:=Timer_PresetValue,
    Q=>Timer_Output,
    ET=>Timer_ElapsedTime);
```

How to Use a Function or a Function Block in IL Language

General Information

This part explains how to implement a Function and a Function Block in IL language.

Functions `IsFirstMastCycle` and `SetRTCDrift` and Function Block `TON` are used as examples to show implementations.

Using a Function in IL Language

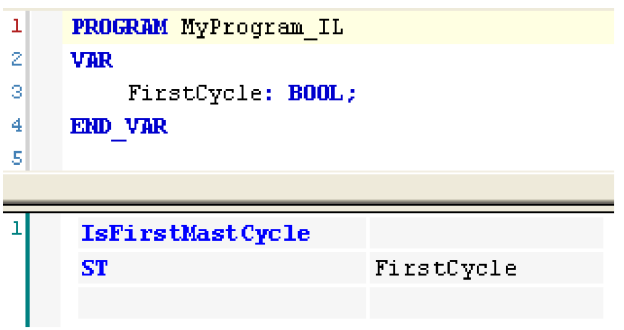
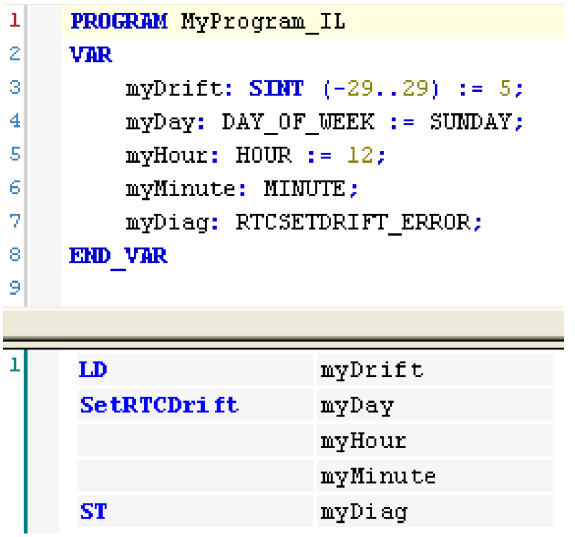
The following procedure describes how to insert a function in IL language:

Step	Action
1	Open or create a new POU in **Instruction List** language. **NOTE:** The procedure to create a POU is not detailed here. For more information, refer to the SoMachine global help.
2	Create the variables that the function requires.
3	If the function has 1 or more inputs, start loading the first input using LD instruction.
4	Insert a new line below and: ● type the name of the function in the operator column (left field), or ● use the **Input Assistant** to select the function (select **Insert Box** in context menu).
5	If the function has more than 1 input and when **Input Assistant** is used, the necessary number of lines is automatically created with `???` in the fields on the right. Replace the `???` with the appropriate value or variable that corresponds to the order of inputs.
6	Insert a new line to store the result of the function into the appropriate variable: type ST instruction in the operator column (left field) and the variable name in the field on the right.

To illustrate the procedure, consider the Functions `IsFirstMastCycle` (without input parameter) and `SetRTCDrift` (with input parameters) graphically presented below:

Function	Graphical Representation
without input parameter: `IsFirstMastCycle`	
with input parameters: `SetRTCDrift`	

In IL language, the function name is used directly in the **Operator Column**:

Function	Representation in SoMachine POU IL Editor
IL example of a function without input parameter: `IsFirstMastCycle`	
IL example of a function with input parameters: `SetRTCDrift`	

Using a Function Block in IL language

The following procedure describes how to insert a function block in IL language:

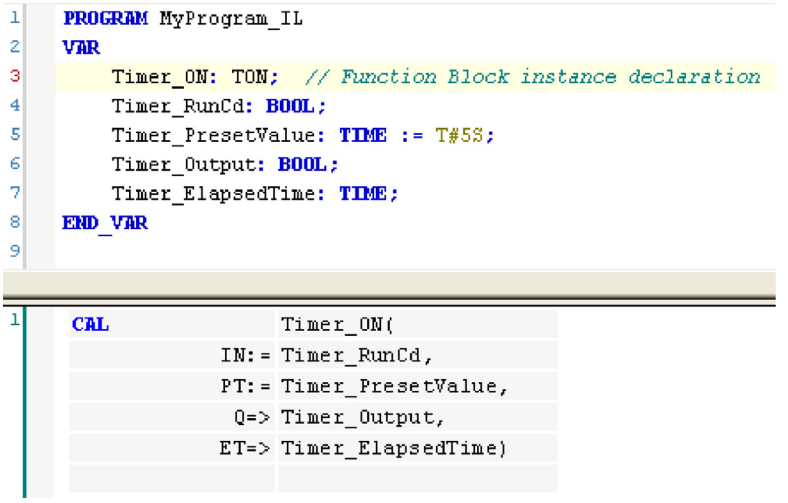
Step	Action
1	Open or create a new POU in **Instruction List** language. **NOTE:** The procedure to create a POU is not detailed here. For more information, refer to the SoMachine global help.
2	Create the variables that the function block requires, including the instance name.

Step	Action
3	Function Blocks are called using a CAL instruction: ● Use the **Input Assistant** to select the FB (right-click and select **Insert Box** in context menu). ● Automatically, the CAL instruction and the necessary I/O are created. Each parameter (I/O) is an instruction: ● Value to inputs are set by ":=". ● Values to outputs are set by "=>".
4	In the CAL right-side field, replace ??? with the instance name.
5	Replace other ??? with an appropriate variable or immediate value.

To illustrate the procedure, consider this example with the TON Function Block graphically presented below:

Function Block	Graphical Representation
TON	

In IL language, the function block name is used directly in the **Operator Column**:

Function Block	Representation in SoMachine POU IL Editor
TON	

How to Use a Function or a Function Block in ST Language

General Information

This part explains how to implement a Function and a Function Block in ST language.

Function `SetRTCDrift` and Function Block `TON` are used as examples to show implementations.

Using a Function in ST Language

The following procedure describes how to insert a function in ST language:

Step	Action
1	Open or create a new POU in **Structured Text** language. **NOTE:** The procedure to create a POU is not detailed here. For more information, refer to the SoMachine global help.
2	Create the variables that the function requires.
3	Use the general syntax in the **POU ST Editor** for the ST language of a function. The general syntax is: `FunctionResult:= FunctionName(VarInput1, VarInput2,.. VarInputx);`

To illustrate the procedure, consider the function `SetRTCDrift` graphically presented below:

Function	Graphical Representation
`SetRTCDrift`	

The ST language of this function is the following:

Function	Representation in SoMachine POU ST Editor

`SetRTCDrift`	`PROGRAM MyProgram_ST`
	`VAR myDrift: SINT(-29..29) := 5;`
	`myDay: DAY_OF_WEEK := SUNDAY;`
	`myHour: HOUR := 12;`
	`myMinute: MINUTE;`
	`myRTCAdjust: RTCDRIFT_ERROR;`
	`END_VAR`
	`myRTCAdjust:= SetRTCDrift(myDrift, myDay, myHour, myMinute);`

Using a Function Block in ST Language

The following procedure describes how to insert a function block in ST language:

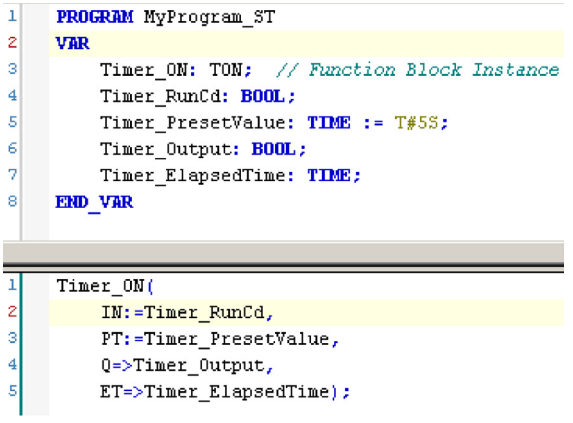
Step	Action
1	Open or create a new POU in **Structured Text** language. **NOTE:** The procedure to create a POU is not detailed here. For more information, refer to the SoMachine global help.
2	Create the input and output variables and the instance required for the function block: ● Input variables are the input parameters required by the function block ● Output variables receive the value returned by the function block

3	Use the general syntax in the **POU ST Editor** for the ST language of a Function Block. The general syntax is:
	`FunctionBlock_InstanceName(Input1:=VarInput1, Input2:=VarInput2,...`
	`Ouput1=>VarOutput1, Ouput2=>VarOutput2,...);`

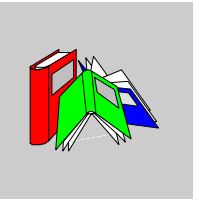
To illustrate the procedure, consider this example with the `TON` function block graphically presented below:

Function Block	Graphical Representation
`TON`	

The following table shows examples of a function block call in ST language:

Function Block	Representation in SoMachine POU ST Editor
`TON`	

Glossary



0-9

%I

According to the IEC standard, %I represents an input bit (for example a language object of type digital IN).

%IW

According to the IEC standard, %IW represents an input word register (for example a language object of type analog IN).

%MW

According to the IEC standard, %MW represents a memory word register (for example a language object of type memory word).

%Q

According to the IEC standard, %Q represents an output bit (for example a language object of type digital OUT).

%QW

According to the IEC standard, %QW represents an output word register (for example a language object of type analog OUT).

1-phase counter

A *1-phase counter* uses 1 hardware input as counter input. It usually counts up or counts down when there is pulse signal in the input.

2-phase counter

A *2-phase counter* uses the phase difference between 2 input counter signals to count up or count down.

A

ADC

analog/digital converter

AFB

application function block

AMOA

An *address of modbus of option application board* installed on the drive.

analog input

An *analog input* module contains circuits that convert an analog DC input signal to a digital value that can be manipulated by the processor. By implication, the analog input is usually direct. That means a data table value directly reflects the analog signal value.

analog output

An *analog output* module contains circuits that transmit an analog DC signal proportional to a digital value input to the module from the processor. By implication, these analog outputs are usually direct. That means a data table value directly controls the analog signal value.

application source

The *application source* file can be uploaded to the PC to reopen a SoMachine project. This source file can support a full SoMachine project (for example, one that includes HMI application).

ARP

The *address resolution protocol* is the IP network layer protocol for Ethernet that maps an IP address to a MAC (hardware) address.

ARRAY

An `ARRAY` is a table containing elements of a single type. The syntax is as follows: `ARRAY [<limits>] OF <Type>`

Example 1: `ARRAY [1..2] OF BOOL` is a 1-dimensional table with 2 elements of type `BOOL`.

Example 2: `ARRAY [1..10, 1..20] OF INT` is a 2-dimensional table with 10x20 elements of type `INT`.

ARW

anti-reset windup

ASCII

The *american standard code for information interchange* is a communication protocol for representing alphanumeric characters (letters, numbers, and certain graphic and control characters).

assigned variable

A variable is "assigned" if its location in controller memory can be known. For example, the `Water_pressure` variable is said to be assigned through its association with memory location `%MW102.Water_pressure`.

ATC

analog tension control

ATV

ATV is the model prefix for Altivar drives. (For example, “ATV312” refers to the Altivar 312 variable speed drive.)

AWG

The *american wire gauge* standard specifies wire gauges in North America.

B

BCD

The *binary coded decimal format* represents decimal numbers between 0 and 9 with a set of 4 bits (a nybble/nibble, also titled as Halfbyte). In this format, the 4 bits used to encode decimal numbers have an unused range of combinations. For example, the number 2,450 is encoded as `0010 0100 0101 0000`

BOOL

A *Boolean* type is the basic data type in computing. A `BOOL` variable can have one of these values: 0 (`FALSE`), 1 (`TRUE`). A bit that is extracted from a word is of type `BOOL`, for example: `%MW10.4` is a fifth bit a memory word number 10.

Boot application

Files that contain machine dependent parameters:
- machine name
- device name or IP address
- Modbus Serial Line address
- Routing table

BOOTP

The *bootstrap protocol* is a UDP network protocol that can be used by a network client to automatically obtain an IP address (and possibly other data) from a server. The client identifies itself to the server using the client MAC address. The server—which maintains a pre-configured table of client device MAC addresses and associated IP addresses—sends the client its pre-configured IP address. BOOTP was originally used as a method that enabled diskless hosts to be remotely booted over a network. The BOOTP process assigns an infinite lease of an IP address. The BOOTP service utilizes UDP ports 67 and 68.

bps

bit per second as a definition of transmission rate, also given in conjunction with multiplicator kilo (kbps) and mega (mbps).

BSH

BSH is a Lexium servo motor from Schneider Electric.

bus base

A *bus base* is a mounting device that is designed to seat an electronic module on a DIN rail and connect it to the TM5 bus for M258 and LMC058 controllers. Each base bus extends the TM5 data and to the power buses and the 24 Vdc I/O power segment. The electronic modules are added to the TM5 system through their insertion on the base bus. The base bus also supplies the articulation point for the terminal blocks.

BYTE

When 8 bits are grouped together, they are called a `BYTE`. You can enter a `BYTE` either in binary mode or in base 8. The `BYTE` type is encoded in an 8-bit format that ranges from `16#00` to `16#FF` (in hexadecimal format).

C

calibration

Graduates a piece of measuring apparatus.

CAN

The *controller area network* protocol (ISO 11898) for serial bus networks is designed for the interconnection of smart devices (from multiple manufacturers) in smart systems for real-time industrial applications. Originally developed for use in automobiles, CAN is now used in a variety of industrial automation control environments.

CANmotion

CANmotion is a CANopen-based motion bus with an additional mechanism that provides synchronization between the motion controller and the drives.

CANopen

CANopen is an open industry-standard communication protocol and device profile specification.

CFC

The *continuous function chart* (an extension of the IEC61131-3 standard) is a graphical programming language that works like a flowchart. By adding simple logic blocks (AND, OR, etc.), each function or function block in the program is represented in this graphical format. For each block, the inputs are on the left and the outputs on the right. Block outputs can be linked to inputs of other blocks in order to create complex expressions.

CiA

CAN in automation is a non-profit group of manufacturers and users dedicated to developing and supporting CAN-based higher layer protocols.

CIP

When the *common industrial protocol* is implemented in a network application layer, it can communicate seamlessly with other CIP-based networks without regard to the protocol. For example, the implementation of CIP in the application layer of an Ethernet TCP/IP network creates an EtherNet/IP environment. Similarly, CIP in the application layer of a CAN network creates a DeviceNet environment. In that case, devices on the EtherNet/IP network can communicate with devices on the DeviceNet network through CIP bridges or routers.

CMU

The *current measurement unit* is used to convert the relative current value (%) provided by TeSys into a real ISO value (A).

configuration

The *configuration* includes the arrangement and interconnection of hardware components within a system and the hardware and software selections that determine the operating characteristics of the system.

controller

A *controller* (or “programmable logic controller,” or “programmable controller”) is used to automate industrial processes.

controller status output

The *controller status output* is a special function used in circuits that are external to the controller that control the power supply to the output devices or the controller power supply.

CPDM

controller power distribution module

CRC

A network message's *cyclic redundancy check* field contains a small number of bits that produce a checksum. The message is calculated by the transmitter according to the message's content. Receiving nodes then recalculate the field. Any discrepancy in the two CRC fields indicates that the transmitted message and the received message are different.

CSA

The *canadian standards association* defines and maintains standards for industrial electronic equipment in hazardous environments.

CTS

Clear to send is a data transmission signal and acknowledges the RDS signal from the transmitting station.

cyclic task

The cyclic scan time has a fixed duration (interval) specified by the user. If the current scan time is shorter than the cyclic scan time, the controller waits until the cyclic scan time has elapsed before starting a new scan.

D

data log

The controller logs events relative to the user application in a data log.

DCE

Data communications equipment describes devices (often modems) that start, stop, and sustain network sessions.

Derating

Derating describes a reduction in an operating specification. For devices in general it is usually a specified reduction in nominal power to facilitate operation at increased ambient conditions like higher temperatures or higher altitudes.

DHCP

The *dynamic host configuration protocol* is an advanced extension of BOOTP. DHCP is a more advanced, but both DHCP and BOOTP are common. (DHCP can handle BOOTP client requests.)

digital I/O

A *digital input* or *output* has an individual circuit connection at the electronic module that corresponds directly to a data table bit that holds the value of the signal at that I/O circuit. It gives the control logic digital access to I/O values.

DIN

Deutsches Institut für Normung is a German institution that sets engineering and dimensional standards.

DINT

A *double integer* type is encoded in a 32-bit format.

DNS

The *domain name system* is the naming system for computers and devices connected to a LAN or the Internet.

drop cable

A *drop cable* is the unterminated derivation cord used to connect a TAP to a device.

DSR

Data set ready is a data transmission signal.

DTM

With *device type managers* representing the field device in SoMachine, direct communications are possible to every single field device via SoMachine, the controller and the field bus, thus avoiding the need for individual cable connections.

DWORD

A *double word* type is encoded in a 32-bit format.

E

EDS

Electronic data sheet contains for example the properties of a device e.g. parameters and settings of a drive.

EEPROM

Electrically erasable programmable read-only memory is a type of non-volatile memory used to store data that must be saved when power is removed.

EIA

The *electronic industries alliance* is the trade organization for establishing electrical/electronic and data communication standards (including RS-232 and RS-485) in the United States.

EIA rack

An *electronic industries alliance rack* is a standardized (EIA 310-D, IEC 60297 and DIN 41494 SC48D) system for mounting various electronic modules in a stack or rack that is 19 inches (482.6 mm) wide.

electronic module

In a programmable controller system, most electronic modules directly interface to the sensors, actuators, and external devices of the machine/process. This electronic module is the component that mounts in a bus base and provides electrical connections between the controller and the field devices. Electronic modules are offered in a variety of signal levels and capacities. (Some electronic modules are not I/O interfaces, including power distribution modules and transmitter/receiver modules.)

EN

EN identifies one of many European standards maintained by CEN (*European Committee for Standardization*), CENELEC (*European Committee for Electrotechnical Standardization*), or ETSI (*European Telecommunications Standards Institute*).

encoder

An *encoder* is a device for length or angular measurement (linear or rotary encoders).

Equipment

An *Equipment* is a part of the *Machine*.

ERC

eccentric roller conveyor

ESD

electrostatic discharge

Ethernet

Ethernet is a physical and data link layer technology for LANs, also known as IEE 802.3.

EtherNet/IP

The *ethernet industrial protocol* is an open communications protocol for manufacturing automation solutions in industrial systems. EtherNet/IP is in a family of networks that implements Common Industrial Protocol at its upper layers. The supporting organization (ODVA) specifies EtherNet/IP to accomplish global adaptability and media independence.

expansion bus

The *expansion bus* is an electronic communication bus between expansion modules and a CPU.

expansion I/O module

An *expansion input or output module* is either a digital or analog module that adds additional I/O to the base controller.

expert I/O

*Expert I/O*s are dedicated modules or channels for advanced features. These features are generally embedded in the module in order to not use the resources of the PLC Controller and to allow a fast response time, depending of the feature. Regarding the function, it could be considered as a “stand alone” module, because the function is independent of the Controller processing cycle, it just exchanges some information with the Controller CPU.

F

FAST I/O

*FAST I/O*s are specific I/Os with some electrical features (response time, for example) but the treatment of these channels is done by the Controller CPU.

FAST task

The *FAST task* is a periodic, high-priority task of a short duration that is run on a processor through its programming software. The task fast speed keeps it from interfering with the execution of lower priority master (MAST) tasks. A FAST task is useful when fast periodic changes in discrete inputs need to be monitored.

FB

A *function block* performs a specific automation function, such as speed control, interval control, or counting. A function block comprises configuration data and a set of operating parameters.

FBD

A *function block diagram* is a graphically oriented programming language, compliant with IEC 61131-3. It works with a list of networks whereby each network contains a graphical structure of boxes and connection lines which represents either a logical or arithmetic expression, the call of a function block, a jump, or a return instruction.

FDT

Field device tool for standardized communications between field devices and SoMachine.

FE

Functional ground is the point of a system or device that must be grounded to help prevent equipment damage.

FG

frequency generator

firmware

The *firmware* represents the operating system on a controller.

Flash memory

Flash memory is nonvolatile memory that can be overwritten. It is stored on a special EEPROM that can be erased and reprogrammed.

FTP

File transfer protocol is a standard network protocol (built on a client-server architecture), to exchange and manipulate files over TCP/IP based networks.

function

A *function*:

- is a POU that returns 1 immediate result
- is directly called with its name (as opposed to through an instance)
- has no persistent state from one call to the next
- can be used as an operand in expressions

Examples: boolean (AND) operators, calculations, conversions (BYTE_TO_INT)

function block (FB)

See *FB*.

function block diagram (FBD)

See *FBD*.

FWD

forward

G

gross weight

Indication of the load weight on an instrument when no tare or predefining device has been used.

GVL

The *global variable list* manages global variables that are available in every application POU.

H

HE10

Rectangular connector for electrical signals with frequencies below 3MHz, complying with IEC60807-2.

HMI

A *human-machine interface* is an operator interface (usually graphical) for industrial equipment.

hot swapping

Hot swapping is the replacement of a component with a like component while the system remains operational. The replacement component begins to function automatically after it is installed.

HSC

high-speed counter

HVAC

Heating ventilation and air conditioning applications monitor and control indoor environments.

I

I/O

input/output

I/O scan

An *input/output scan* continuously polls I/O modules to collect data bits and status, error, and diagnostics information. This process monitors inputs and controls outputs.

I/O terminal

An *input/output terminal* on the front of an expansion I/O module connects input and output signals.

ICMP

The *internet control message protocol* reports errors and provides information related to datagram processing.

IEC

The *international electrotechnical commission* is a non-profit and non-governmental international standards organization that prepares and publishes international standards for all electrical, electronic, and related technologies.

IEC 61131-3

The IEC 61131-3 is an *international electrotechnical commission* standard for industrial automation equipment (like controllers). IEC 61131-3 deals with controller programming languages and defines 2 graphical and 2 textual programming language standards:

- **graphical:** ladder diagram, function block diagram
- **textual:** structured text, instruction list

IEEE

The *institute of electrical and electronics engineers* is a non-profit international standards and conformity assessment body for advances in all fields of electrotechnology.

IEEE 802.3

IEEE 802.3 is a collection of IEEE standards defining the physical layer, and the media access control (MAC) sublayer of the data link layer, of wired Ethernet.

IL

A program written in the *instruction list* language is composed of a series of instructions executed sequentially by the controller. Each instruction includes a line number, an instruction code, and an operand. (IL is IEC 61131-3 compliant.)

immediate addressing

The direct method of addressing memory objects, including physical inputs and outputs, used in programming instructions as operands and parameters by using their direct address (for example, `%Iwx` or `%QWx`).

The use of immediate addressing in your program may avoid the need to create symbols for these objects, but there are also disadvantages. For example, if you change the program configuration by adding or deleting devices or I/O modules or slices, the immediate addresses used as programming instruction operands and/or parameters are not updated and must be corrected manually, which may cause extensive program modifications and lead to incorrect programming instructions. (See *symbolic addressing*.)

input filter

An *input filter* is a special function that rejects input noises. It is useful for eliminating input noises and chatter in limit switches. All inputs provide a level of input filtering using the hardware. Additional filtering with software is also configurable through the programing or the configuration software.

input terminal

An *input terminal* on the front of an expansion I/O module connects input signals from input devices (such as sensors, push buttons, and limit switches). For some modules, input terminals accept both sink and source DC input signals.

instruction list language (IL)

Refer to IL.

INT

A single *integer* is encoded in 16 bits.

IP

The *internet protocol* is part of the TCP/IP protocol family that tracks the Internet addresses of devices, routes outgoing messages, and recognizes incoming messages.

IP 20

Ingress protection rating according to IEC 60529. IP20 modules are protected against ingress and contact of objects larger than 12.5 mm. The module is not protected against harmful ingress of water.

IP 67

Ingress protection rating according to IEC 60529. IP67 modules are completely protected against ingress of dust and contact. Ingress of water in harmful quantity is not possible when the enclosure is immersed in water up to 1m.

K

Kd

derivative gain

Ki

integral gain

Kp

proportional gain

L

Ladder Diagram language

See *LD*.

LAN

A *local area network* local area network is a short-distance communications network that is implemented in a home, office, or institutional environment.

latching input

A *latching input* module interfaces with devices that transmit messages in short pulses. Incoming pulses are captured and recorded for later examination by the application.

LCD

liquid crystal display

LD

A program in the *ladder diagram* language includes a graphical representation of instructions of a controller program with symbols for contacts, coils, and blocks in a series of rungs executed sequentially by a controller. IEC 61131-3 compliant.

LED

A *light emitting diode* is an indicator that lights up when electricity passes through it.

LINT

Long integer is a 64-bit variable (4 times INT or two times DINT).

LMC

lexium motion control

load receiver device

Part of instrument that will receive the load.

located variable

A *located variable* has an address. (See *unlocated variable*.)

LRC

longitudinal redundancy checking

LREAL

Long real is a 64-bit variable.

LSB

The *least significant bit* (or *least significant byte*) is the part of a number, address, or field that is written as the right-most single value in conventional hexadecimal or binary notation.

LWORD

A *long word* type is encoded in a 64-bit format.

M

MAC address

The *media access control address* is a unique 48-bit number associated with a specific piece of hardware. The MAC address is programmed into each network card or device when it is manufactured.

Machine

A *Machine* consists of several *functions* and/or *equipments* which build the machine.

Magelis

Magelis is the commercial name for Schneider Electric's range of HMI terminals.

MAST

A master (MAST) task is a processor task that is run through its programming software. The MAST task has two sections:

- **IN:** Inputs are copied to the IN section before execution of the MAST task.
- **OUT:** Outputs are copied to the OUT section after execution of the MAST task.

master/slave

The single direction of control in a network that implements the master/slave model is always from a master device or process to one or more slave devices.

maximum weight

Maximum measuring capacity, not taking account of the additive capacity of the tare.

MIB

The *management information base* is an object database that is monitored by a network management system like SNMP. SNMP monitors devices that are defined by their MIBs. Schneider has obtained a private MIB, groupeschneider (3833).

minimum I/O update time

The *minimum I/O update time* is the minimum time it takes for the bus cycle to shut down to force an I/O update at each cycle.

minimum weight

Load value under which measuring results can be marred by a relative detected error that is too large.

Modbus

The Modbus communication protocol allows communications between many devices connected to the same network.

Modbus SL

Modbus serial line

MSB

The *most significant bit* (or *most significant byte*) is the part of a number, address, or field that is written as the left-most single value in conventional hexadecimal or binary notation.

N

NAK

negative acknowledge

NC

A *normally closed* contact is a contact pair that is closed when the actuator is de-energized (no power is applied) and open when the actuator is energized (power is applied).

NEC

The *national electric code* dictates the safe installation of electrical wiring and equipment.

NEMA

The *national electrical manufacturers association* publishes standards for the performance of various classes of electrical enclosures. The NEMA standards cover corrosion resistance, ability to protect from rain and submersion, etc. For IEC member countries, the IEC 60529 standard classifies the ingress protection rating for enclosures.

net weight (net)

Weight indication of a load placed on an instrument after a tare device has been used.

Net weight = Gross weight - Tare weight

network

A network includes interconnected devices that share a common data path and protocol for communications.

Nibble

A *Nibble* is a half-byte (representing 4 bits of a byte).

NMT

Network management protocols provide services for network initialization, error control, and device status control.

NMT state machine

A *network management state machine* defines the communication behavior of any CANopen device. The CANopen NMT state machine consists of an initialization state, a pre-operational state, an Operational state, and a stopped state. After power-on or reset, the device enters the initialization state. After the device initialization is finished, the device automatically enters the pre-operational state and announces the state transition by sending the boot-up message. In this manner, the device indicates that it is ready to work. A device that stays in pre-operational state may start to transmit SYNC-, Time Stamp-, or Heartbeat message. In this state, the device can not communicate through a PDO; it must do so with an SDO. In the operational state, the device can use all supported communication objects.

NO

A *normally open* contact is a contact pair that is open when the actuator is de-energized (no power is applied) and closed when the actuator is energized (power is applied).

node

A *node* is an addressable device on a communication network.

O

ODVA

The *open deviceNet vendors association* supports the family of network technologies that are built on CIP (EtherNet/IP, DeviceNet, and CompoNet).

OS

Operating system. Can be used for Firmware that can be uploaded/downloaded by the user.

OSI

The *open system interconnection* reference model is a 7-layer model that describes network protocol communications. Each abstract layer receives services from the layer below it and provides services to the layer above.

OTB

Optimized terminal block, used in the context of Advantys I/O distributed module

output terminal

An *output terminal* connects output signals to output devices (such as electromechanical relays and solenoid valves).

P

pallet

A *pallet* is a portable platform, which is used for storing or moving goods.

PCI

A *peripheral component interconnect* is an industry-standard bus for attaching peripherals.

PDM

A *power distribution module* distributes either AC or DC field power to a cluster of I/O modules.

PDO

A *process data object* is transmitted as an unconfirmed broadcast message or sent from a producer device to a consumer device in a CAN-based network. The transmit PDO from the producer device has a specific identifier that corresponds to the receive PDO of the consumer devices.

PDU

protocol data unit

PE

Protective ground is a return line across the bus for fault currents generated at a sensor or actuator device in the control system.

periodic execution

The master task is executed either cyclically or periodically. In periodic mode, you determine a specific time (period) in which the master task must be executed. If it is executed under this time, a waiting time is generated before the next cycle. If it is executed over this time, a control system indicates the overrun. If the overrun is too high, the controller is stopped.

persistent data

Value of persistent data that will be used at next application change or cold start. Only get re-initialized at a reboot of the controller or reset origin. Especially they maintain their values after a download.

PI

proportional integral

PID

proportional, integral and derivative control

PLC

The *programmable logic controller* is the “brain” of an industrial manufacturing process. It automates a process, used instead of relay control systems. PLCs are computers suited to survive the harsh conditions of the industrial environment.

PLCopen

The PLCopen standard brings efficiency, flexibility, and manufacturer independence to the automation and control industry through the standardization of tools, libraries, and modular approaches to software programming.

PLI

pulse latch input

post-configuration

Post-configuration files contain machine-independent parameters, including:

- machine name
- device name or IP address
- Modbus serial line address
- routing table

POU

A *program organization unit* includes a variable declaration in source code and the corresponding instruction set. POUs facilitate the modular reuse of software programs, functions, and function blocks. Once declared, POUs are available to one another. SoMachine programming requires the utilization of POUs.

POU FB

Program organization unit function block types are user programs that can be defined by the user in the ST, IL, LD, or FBD languages. You can use POU FB types in an application to:

- simplify the design and entry of the program
- make the program easier to read
- simplify debugging
- reduce the amount of generated code

power supply terminals

The power supply is connected to these terminals to provide power to the controller.

Profibus DP

Profibus Decentralized Peripheral

An open bus system that uses an electrical network based on a shielded 2-wire line or an optical network based on a fiber-optic cable. DP transmission allows for high-speed, cyclic exchange of data between the controller CPU and the distributed I/O devices.

protocol

A *protocol* is a convention or standard that controls or enables the connection, communication, and data transfer between two computing endpoints.

Pt100/Pt1000

Platinum resistance thermometer are characterized by their nominal resistance R0 at a temperature of 0° C.

- Pt100 (R0 = 100 Ohm)
- Pt1000 (R0 = 1 kOhm)

PTO

Pulse train outputs are used to control for instance stepper motors in open loop.

PWM

Pulse width modulation is used for regulation processes (e.g. actuators for temperature control) where a pulse signal is modulated in its length. For these kind of signals, transistor outputs are used.

R

RAM

random access memory

REAL

Real is a numeric data type. The REAL type is encoded in a 32-bit format.

real-time clock (RTC)

See RTC

reflex output

In a counting mode, the high speed counter current value is measured against its configured thresholds to determine the state of these dedicated outputs.

retained data

A *retained data* value is used in the next power-on or warm start. The value is retained even after an uncontrolled shutdown of the controller or a normal switch-off of the controller.

RFID

Radio-frequency identification is an automatic identification method that relies on the storage and remote retrieval of data using RFID tags or transponders.

RJ-45

This *registered jack* is a modular connector that is commonly implemented in communication networks.

RPDO

A *receive PDO* sends data to a device in a CAN-based network.

RPM

revolutions per minute

RPS

revolutions per second

RS-232

RS-232 (also known as EIA RS-232C or V.24) is a standard type of serial communication bus, based on three wires.

RS-485

RS-485 (also known as EIA RS-485) is a standard type of serial communication bus, based on two wires.

RTC

The *real-time clock* option keeps the time for a limited amount of time even when the controller is not powered.

RTS

Request to send is a data transmission signal and will be acknowledged by the CTS signal from the destination node.

RTU

A *remote terminal unit* is a device that interfaces with objects in the physical world to a distributed control system or SCADA system by transmitting telemetry data to the system and/or altering the state of connected objects based on control messages received from the system.

RxD

receiving data (data transmission signal)

S

SCADA

A *supervisory control and data acquisition* system monitors, manages, and controls industrial applications or processes.

scale division

Value in mass units, expressing the difference between two consecutive indications for one numerical indication.

scan

A controller scanning program performs 3 basic functions: [1] It reads inputs and places these values in memory; [2] it executes the application program 1 instruction at a time and stores results in memory; [3] It uses the results to update outputs.

SDO

A *service data object* message is used by the field bus master to access (read/write) the object directories of network nodes in CAN-based networks. SDO types include service SDOs (SSDOs) and client SDOs (CSDOs).

SEL-V

A system that follows IEC 61140 guidelines for *safety extra low voltage* is protected in such a way that voltage between any 2 accessible parts (or between 1 accessible part and the PE terminal for Class 1 equipment) does not exceed a specified value under normal conditions or under single-fault conditions.

Sequential Function Chart

See *SFC*.

SFC

A program written in the *sequential function chart* language can be used for processes that can be split into steps. SFC is composed of steps with associated actions, transitions with associated logic condition, and directed links between steps and transitions. (The SFC standard is defined in IEC 848. It is IEC 61131-3 compliant.)

sink input

A *sink input* is a wiring arrangement in which the device provides current to the input electronic module. A sink input is referenced to 0 Vdc.

SINT

Signed integer is a 16-bit value.

SL

serial line

SMS

The *short message service* is a standard communication service for telephones (or other devices) that send short text messages over the mobile communications system.

SNMP

The *simple network management protocol* can control a network remotely by polling the devices for their status, performing security tests, and viewing information relating to data transmission. It can also be used to manage software and databases remotely. The protocol also permits active management tasks, such as modifying and applying a new configuration

source output

A *source output* is a wiring arrangement in which the output electronic module provides current to the device. A source output is referenced to +24 Vdc.

SSI

Serial synchronous interface is a common interface for relative and absolute measurement systems like encoders.

ST

See *structured text*.

STN

Scan Twisted Nematic (also known as passive matrix)

STRING

A `STRING` variable is a series of ASCII characters.

Structured Text

A program written in the *structured text* (ST) language includes complex statements and nested instructions (such as iteration loops, conditional executions, or functions). ST is compliant with IEC 61131-3.

symbol

A *symbol* is a string of a maximum of 32 alphanumeric characters, of which the first character is alphabetic. It allows you to personalize a controller object to facilitate the maintainability of the application.

symbolic addressing

The indirect method of addressing memory objects, including physical inputs and outputs, used in programming instructions as operands and parameters by first defining symbols for them using these symbols in association with the programming instructions.

In contrast to immediate addressing, this is the recommended method because if the program configuration changes, symbols are automatically updated with their new immediate address associations, whereas any immediate addresses used as operands or parameters are not. (See *immediate addressing*.)

system time

An internal clock provides a device with the system time.

system variable

A system variable structure provides controller data and diagnostic information and allows sending commands to the controller.

T

TAP

A *terminal access point* is a junction box connected to the trunk cable that allows you to plug in drop cables.

tare

Load placed on the load receiver along with the product to be weighed.

tare device

Device allowing the instrument indication to be moved to zero when a load is positioned on the load receiver:

tare predefining device

Device allowing a predefined tare value to be subtracted from a gross weight value and indicating the result of the calculation. The load range is consequently reduced.

Tare Value

Weight value of a load, determined by a tare full-bridge strain gauge electronic module.

taring

Action allowing the instrument indication to be moved to zero when a load is positioned on the load receiver.

task

A group of sections and subroutines, executed cyclically or periodically for the MAST task, or periodically for the FAST task.

A task possesses a level of priority and is linked to inputs and outputs of the controller. These I/O are refreshed in consequence.

A controller can have several tasks.

TCP

A *transmission control protocol* is a connection-based transport layer protocol that provides a reliable simultaneous bi-directional transmission of data. TCP is part of the TCP/IP protocol suite.

terminal block

The *terminal block* is the component that mounts in an electronic module and provides electrical connections between the controller and the field devices.

TFT

thin film transmission (also known as active matrix)

threshold output

Threshold outputs are controlled directly by the HSC according to the settings established during configuration.

TP

A *touch probe* is a position capture that is triggered by a fast input signal (quick sensor). On the rising edge of the touch probe input the position of an encoder is captured. Example: This is used for packaging machines to capture the position of a printmark on a film to cut always on the same position.

TPDO

A *transmit PDO* reads data from a device in a CAN-based system.

trunk cable

A *trunk cable* is the main cable that is terminated at both physical ends with line termination resistors.

TVDA

tested validated documented architectures

TxD

TxD represents a transmit signal.

U

UDINT

An *unsigned double integer* is encoded in 32 bits.

UDP

The *user datagram protocol* is a connectionless mode protocol (defined by IETF RFC 768) in which messages are delivered in a datagram (data telegram) to a destination computer on an IP network. The UDP protocol is typically bundled with the Internet Protocol. UDP/IP messages do not expect a response, and are therefore ideal for applications in which dropped packets do not require retransmission (such as streaming video and networks that demand real-time performance).

UINT

An *unsigned integer* is encoded in 16 bits.

UL

Underwriters Laboratories, US organization for product testing and safety certification.

unlocated variable

An *unlocated variable* does not have an address. (See *located variable*.)

UTC

coordinated universal time

V

VSD

variable speed drive

W

WORD

The *WORD* type is encoded in a 16-bit format.