EcoStruxure Machine Expert - Basic Example Guide

Importing Twido COMM Macros to M221 xSample_ Twido_Macro_COMM_Conversion.smbe

12/2018

EIO0000003372.00

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.

You agree not to reproduce, other than for your own personal, noncommercial use, all or part of this document on any medium whatsoever without permission of Schneider Electric, given in writing. You also agree not to establish any hypertext links to this document or its content. Schneider Electric does not grant any right or license for the personal and noncommercial use of the document or its content, except for a non-exclusive license to consult it on an "as is" basis, at your own risk. All other rights are reserved.

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, service, 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” or “Warning” 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 a hazardous situation which, if not avoided, **will result in** death or serious injury.

⚠ WARNING

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

⚠ CAUTION

CAUTION indicates a hazardous situation which, if not avoided, **could 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.

BEFORE YOU BEGIN

Do not use this product on machinery lacking effective point-of-operation guarding. Lack of effective point-of-operation guarding on a machine can result in serious injury to the operator of that machine.

⚠ WARNING

UNGUARDED EQUIPMENT

- Do not use this software and related automation equipment on equipment which does not have point-of-operation protection.
- Do not reach into machinery during operation.

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

This automation equipment and related software is used to control a variety of industrial processes. The type or model of automation equipment suitable for each application will vary depending on factors such as the control function required, degree of protection required, production methods, unusual conditions, government regulations, etc. In some applications, more than one processor may be required, as when backup redundancy is needed.

Only you, the user, machine builder or system integrator can be aware of all the conditions and factors present during setup, operation, and maintenance of the machine and, therefore, can determine the automation equipment and the related safeties and interlocks which can be properly used. When selecting automation and control equipment and related software for a particular application, you should refer to the applicable local and national standards and regulations. The National Safety Council's Accident Prevention Manual (nationally recognized in the United States of America) also provides much useful information.

In some applications, such as packaging machinery, additional operator protection such as point-of-operation guarding must be provided. This is necessary if the operator's hands and other parts of the body are free to enter the pinch points or other hazardous areas and serious injury can occur. Software products alone cannot protect an operator from injury. For this reason the software cannot be substituted for or take the place of point-of-operation protection.

Ensure that appropriate safeties and mechanical/electrical interlocks related to point-of-operation protection have been installed and are operational before placing the equipment into service. All interlocks and safeties related to point-of-operation protection must be coordinated with the related automation equipment and software programming.

NOTE: Coordination of safeties and mechanical/electrical interlocks for point-of-operation protection is outside the scope of the Function Block Library, System User Guide, or other implementation referenced in this documentation.

START-UP AND TEST

Before using electrical control and automation equipment for regular operation after installation, the system should be given a start-up test by qualified personnel to verify correct operation of the equipment. It is important that arrangements for such a check be made and that enough time is allowed to perform complete and satisfactory testing.

⚠ WARNING

EQUIPMENT OPERATION HAZARD

- Verify that all installation and set up procedures have been completed.
- Before operational tests are performed, remove all blocks or other temporary holding means used for shipment from all component devices.
- Remove tools, meters, and debris from equipment.

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

Follow all start-up tests recommended in the equipment documentation. Store all equipment documentation for future references.

Software testing must be done in both simulated and real environments.

Verify that the completed system is free from all short circuits and temporary grounds that are not installed according to local regulations (according to the National Electrical Code in the U.S.A, for instance). If high-potential voltage testing is necessary, follow recommendations in equipment documentation to prevent accidental equipment damage.

Before energizing equipment:

- Remove tools, meters, and debris from equipment.
- Close the equipment enclosure door.
- Remove all temporary grounds from incoming power lines.
- Perform all start-up tests recommended by the manufacturer.

OPERATION AND ADJUSTMENTS

The following precautions are from the NEMA Standards Publication ICS 7.1-1995 (English version prevails):

- Regardless of the care exercised in the design and manufacture of equipment or in the selection and ratings of components, there are hazards that can be encountered if such equipment is improperly operated.
- It is sometimes possible to misadjust the equipment and thus produce unsatisfactory or unsafe operation. Always use the manufacturer's instructions as a guide for functional adjustments. Personnel who have access to these adjustments should be familiar with the equipment manufacturer's instructions and the machinery used with the electrical equipment.
- Only those operational adjustments actually required by the operator should be accessible to the operator. Access to other controls should be restricted to prevent unauthorized changes in operating characteristics.

About the Book

At a Glance

Document Scope

This document describes how to import COMM macros from a TwidoSuite project to EcoStruxure Machine Expert - Basic using communication function blocks.

The example described in this document is intended for learning purposes only. It must not be used directly on products that are part of a machine or process.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

Do not include any wiring information, programming or configuration logic, or parameter values from any of the examples in your machine or process without thoroughly testing your entire application.

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

This document and its related EcoStruxure Machine Expert - Basic project file focus on specific instructions and function blocks provided with EcoStruxure Machine Expert - Basic, and on specific features available in EcoStruxure Machine Expert - Basic. They are intended to help you understand how to develop, test, commission, and integrate applicative software of your own design in your control systems.

The example is intended for new EcoStruxure Machine Expert - Basic users who already have some degree of expertise in the design and programming of control systems.

Validity Note

This document has been updated for the release of EcoStruxure™ Machine Expert - Basic V1.0.

Related Documents

Title of Documentation	Reference Number
Modbus Master/Jbus Master/Ethernet TCP/IP Modbus Protocols	*146107401A55*
EcoStruxure Machine Expert - Basic - Operating Guide	*EIO0000003281 (ENG)* *EIO0000003282 (FRA)* *EIO0000003283 (GER)* *EIO0000003284 (SPA)* *EIO0000003285 (ITA)* *EIO0000003286 (CHS)* *EIO0000003287 (POR)* *EIO0000003288 (TUR)*
EcoStruxure Machine Expert - Basic Generic Functions - Library Guide	*EIO0000003289 (ENG)* *EIO0000003290 (FRE)* *EIO0000003291 (GER)* *EIO0000003292 (SPA)* *EIO0000003293 (ITA)* *EIO0000003294 (CHS)* *EIO0000003295 (POR)* *EIO0000003296 (TUR)*

You can download these technical publications and other technical information from our website at https://www.schneider-electric.com/en/download

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.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

- Only use software approved by Schneider Electric for use with this equipment.
- Update your application program every time you change the physical hardware configuration.

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

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

Do not include any wiring information, programming or configuration logic, or parameter values from any of the examples in your machine or process without thoroughly testing your entire application.

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

Chapter 1
Example Description

What Is in This Chapter?

This chapter contains the following sections:

Section 1.1
Introduction

What Is in This Section?

This section contains the following topics:

Overview

General

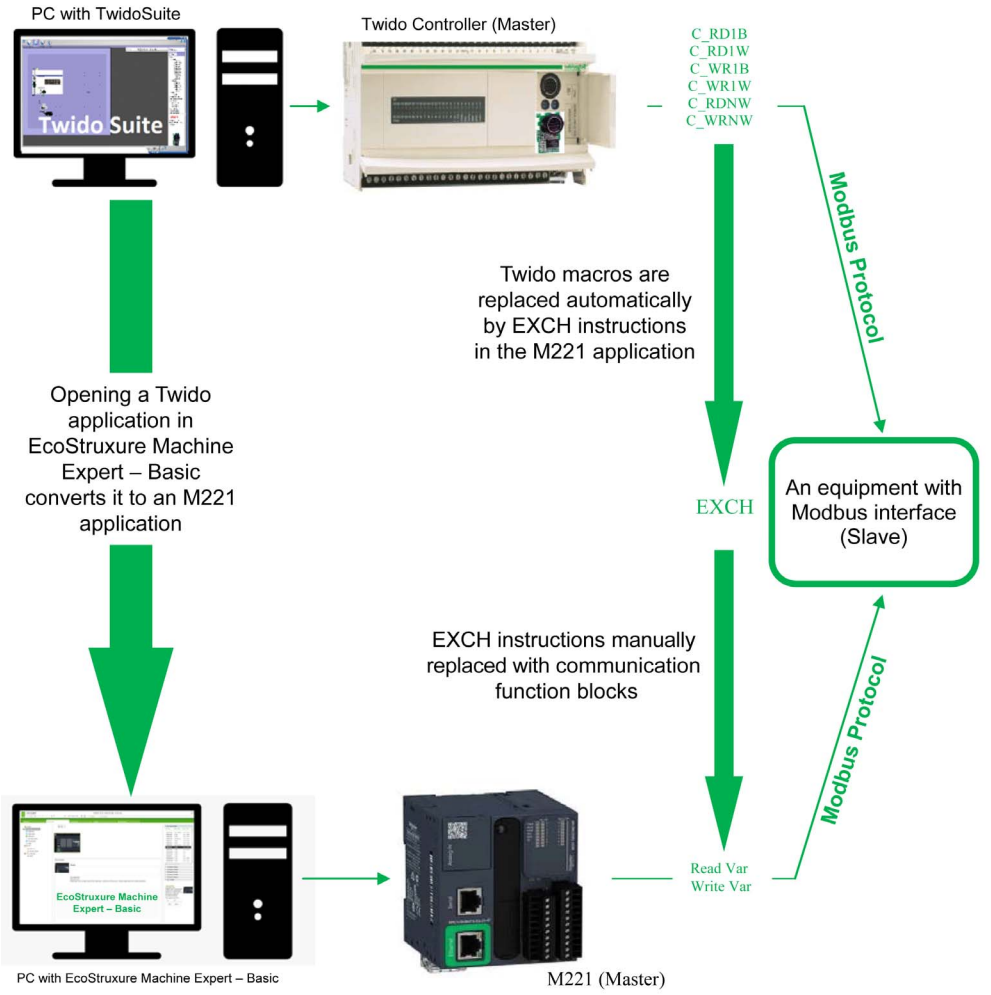
The Example guide and its corresponding project template, included with EcoStruxure Machine Expert - Basic, help you to adapt the code generated after importing a Twido project into EcoStruxure Machine Expert - Basic. In particular, it shows how to replace `EXCH` communication macros by M221 communication function blocks. The use of communication function blocks allows additional features to be included, such as error management.

This document explains:

- Automatic conversion of a Twido application into an EcoStruxure Machine Expert - Basic application.
- How to modify the resulting `EXCH` instructions of the automatic conversion of the Twido communication macros.
- How to, instead, replace `EXCH` instructions with communication function blocks.

In fact, you have the option to by-pass the conversion of the Twido communication macros into `EXCH` instructions prior to conversion of the application. With that, you can replace the communication macros directly by manually substituting with the M221 communication function blocks.

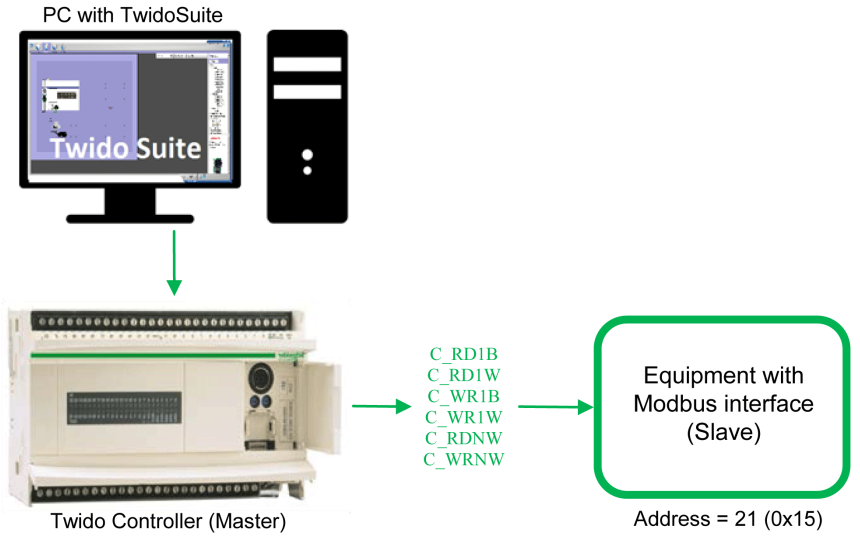
The objective of the example is illustrated below:



Automatic Conversion of Twido COMM Macros to EcoStruxure Machine Expert - Basic EXCH Instructions

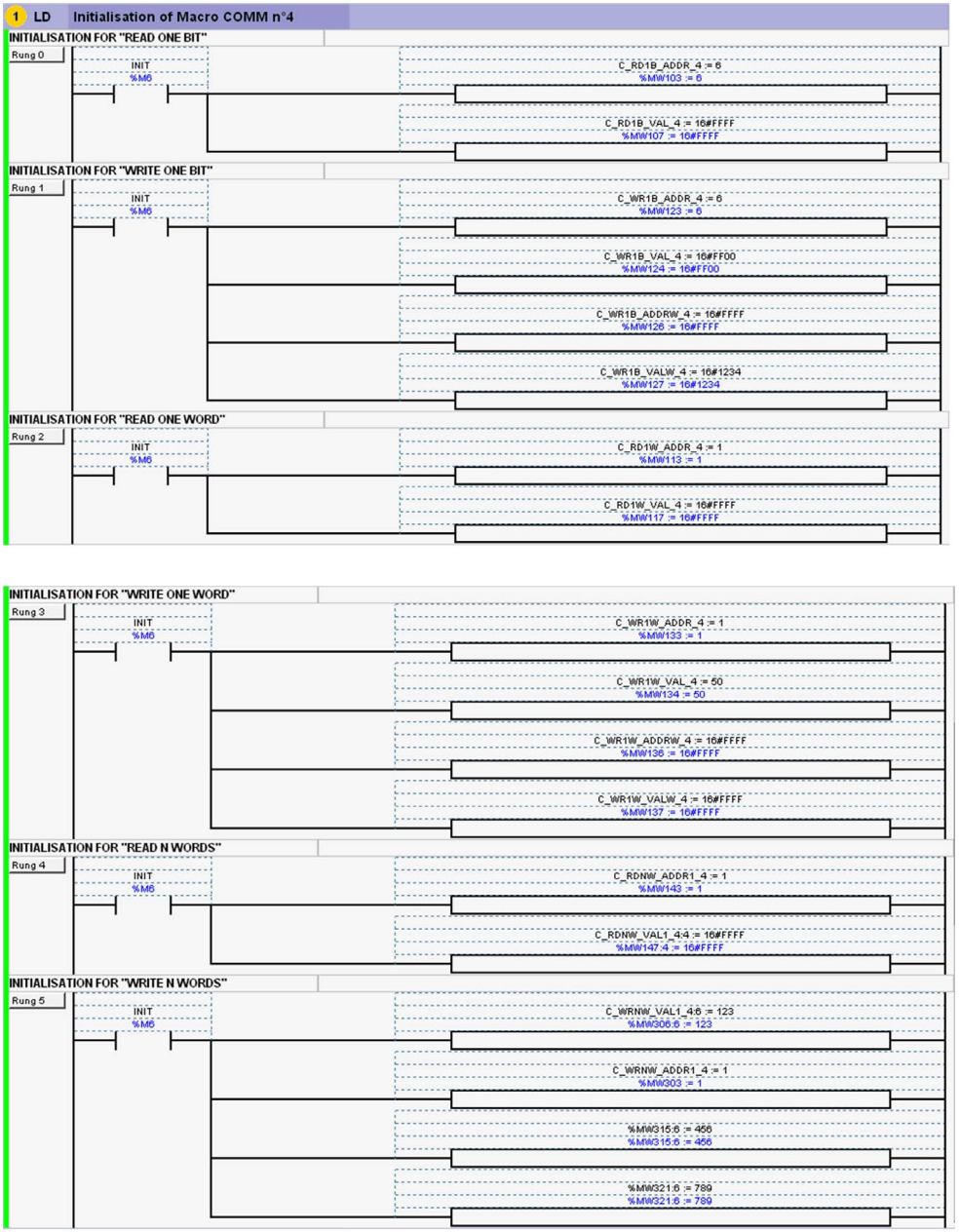
Twido Project Example Using COMM Macros

This figure presents the hardware configuration of a Twido project that implements the communication macros and communicates with a slave:

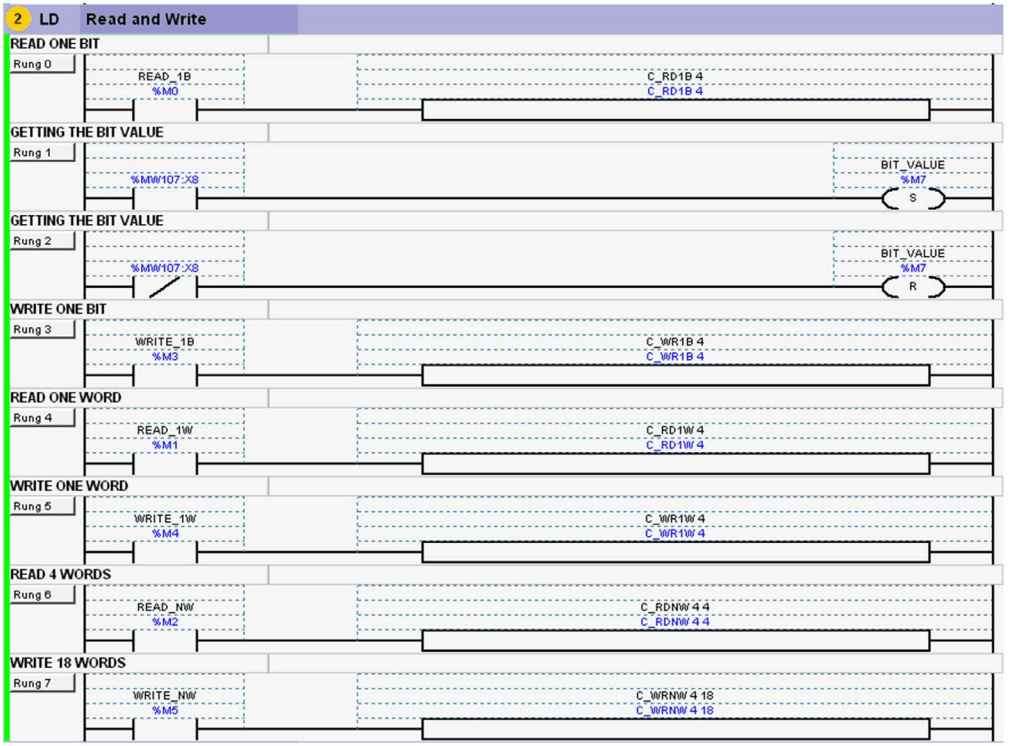


The TwidoSuite project file named `Example_COMM_Macros.xpr` is available for you to implement this example. The following graphics present this Twido project.

Initialization of the macros:



Call of the macros:



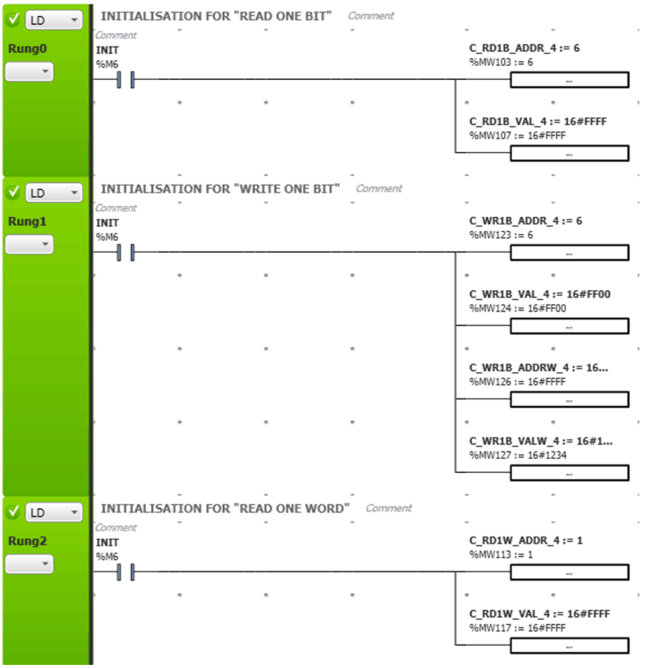
This project:

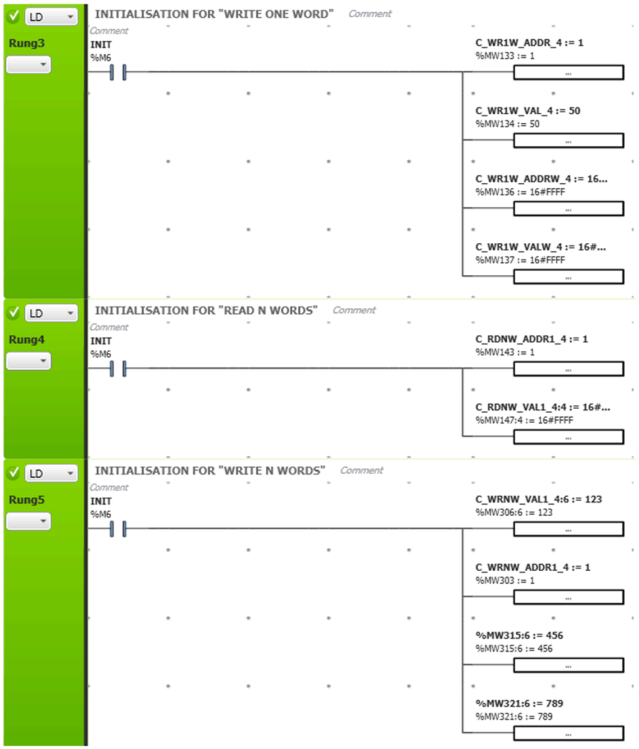
- Initializes the memory words when `%M6` is set to TRUE. The addresses and the default values are set.
- Reads one bit when `%M0` is set to TRUE. The result is stored in `%MW107`.
- Writes one bit when `%M3` is set to TRUE. The value to write is stored in `%MW124`.
- Reads one word when `%M1` is set to TRUE. The result is stored in `%MW117`.
- Writes one word when `%M4` is set to TRUE. The value to write is stored in `%MW134`.
- Reads 4 words when `%M2` is set to TRUE. The result is stored in `%MW147` to `%MW149`.
- Writes 18 words when `%M5` is set to TRUE. The values to write are stored in `%MW306` to `%MW323`.

Automatic Conversion of a Twido Project to EcoStruxure Machine Expert - Basic

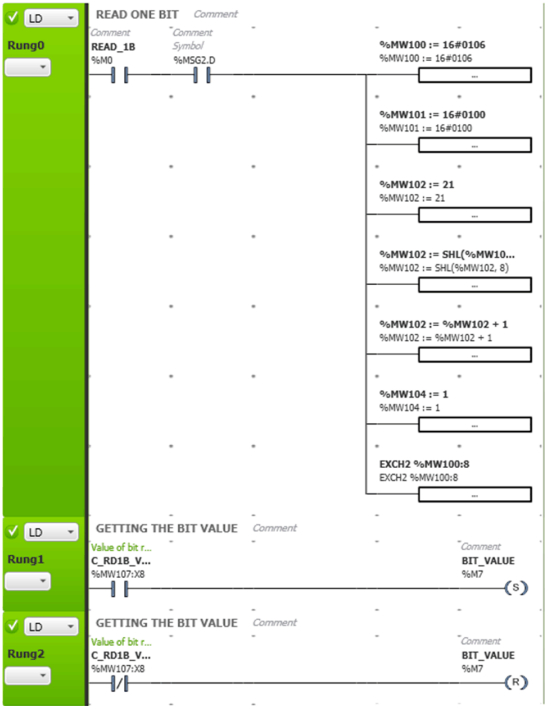
If you open the Twido project with EcoStruxure Machine Expert - Basic, it is automatically converted. The following graphics presents the rungs from this project which correspond to the rungs of the Twido project.

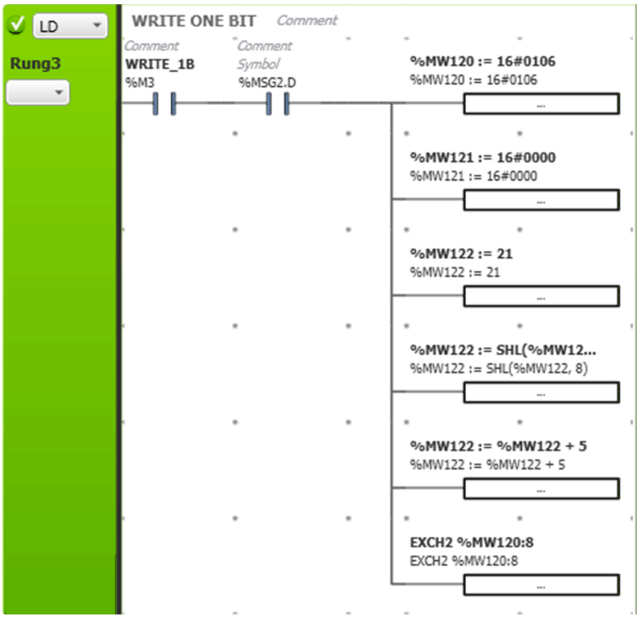
Initialization of the Modbus exchange parameters and values (no difference from Twido):

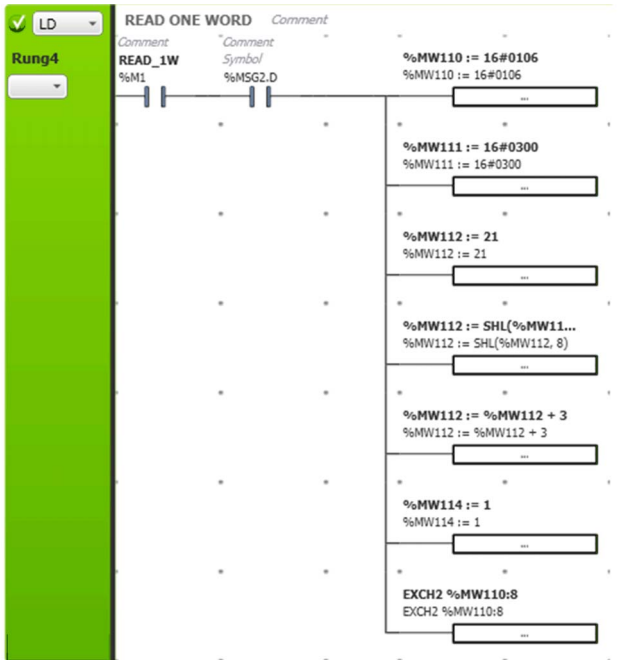


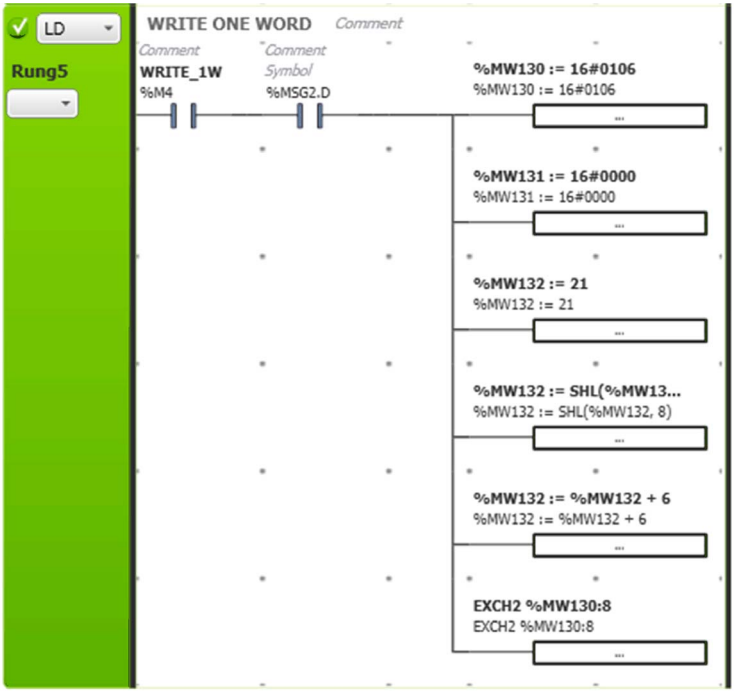

LD
Rung3
INITIALISATION FOR "WRITE ONE WORD"
Comment
Comment
INIT
%M6
C_WR1W_ADDR_4 := 1
%MW133 := 1
C_WR1W_VAL_4 := 50
%MW134 := 50
C_WR1W_ADDRW_4 := 16...
%MW136 := 16#FFFF
C_WR1W_VALW_4 := 16#...
%MW137 := 16#FFFF
LD
Rung4
INITIALISATION FOR "READ N WORDS"
Comment
Comment
INIT
%M6
C_RDNW_ADDR1_4 := 1
%MW143 := 1
C_RDNW_VAL1_4:4 := 16#...
%MW147:4 := 16#FFFF
LD
Rung5
INITIALISATION FOR "WRITE N WORDS"
Comment
Comment
INIT
%M6
C_WRNW_VAL1_4:6 := 123
%MW306:6 := 123
C_WRNW_ADDR1_4 := 1
%MW303 := 1
%MW315:6 := 456
%MW315:6 := 456
%MW321:6 := 789
%MW321:6 := 789

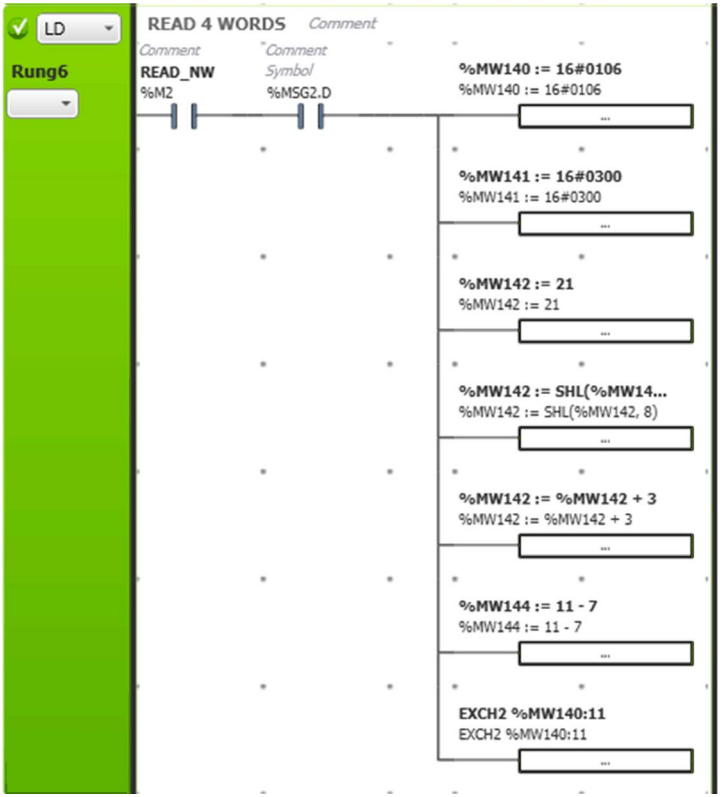
Modbus exchange requests (equivalent to macro calls):

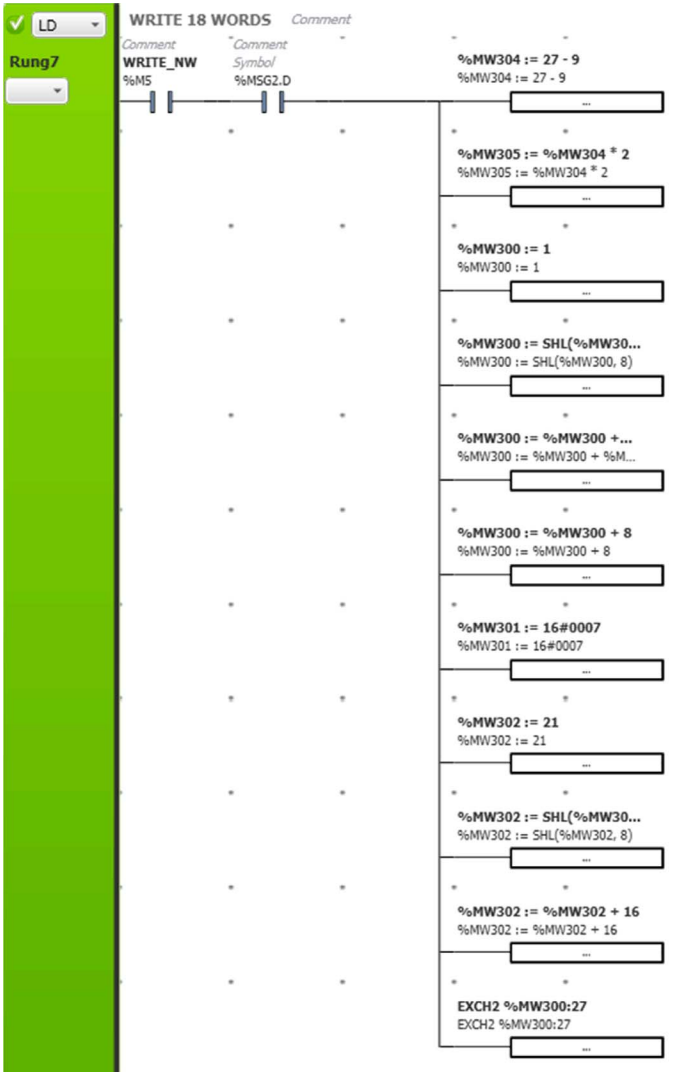



LD
Rung3
WRITE ONE BIT
Comment
Comment
WRITE_1B
%M3
Comment
Symbol
%MSG2.D
%MW120 := 16#0106
%MW120 := 16#0106
%MW121 := 16#0000
%MW121 := 16#0000
%MW122 := 21
%MW122 := 21
%MW122 := SHL(%MW12...
%MW122 := SHL(%MW122, 8)
%MW122 := %MW122 + 5
%MW122 := %MW122 + 5
EXCH2 %MW120:8
EXCH2 %MW120:8


LD
Rung4
READ ONE WORD
Comment
Comment
READ_1W
%M1
Comment
Symbol
%MSG2.D
%MW110 := 16#0106
%MW110 := 16#0106
%MW111 := 16#0300
%MW111 := 16#0300
%MW112 := 21
%MW112 := 21
%MW112 := SHL(%MW11...
%MW112 := SHL(%MW112, 8)
%MW112 := %MW112 + 3
%MW112 := %MW112 + 3
%MW114 := 1
%MW114 := 1
EXCH2 %MW110:8
EXCH2 %MW110:8


LD
Rung5
WRITE ONE WORD
Comment
Comment
WRITE_1W
%M4
Comment
Symbol
%MSG2.D
%MW130 := 16#0106
%MW130 := 16#0106
%MW131 := 16#0000
%MW131 := 16#0000
%MW132 := 21
%MW132 := 21
%MW132 := SHL(%MW13...
%MW132 := SHL(%MW132, 8)
%MW132 := %MW132 + 6
%MW132 := %MW132 + 6
EXCH2 %MW130:8
EXCH2 %MW130:8


LD
Rung6
READ 4 WORDS
Comment
Comment
READ_NW
%M2
Comment
Symbol
%MSG2.D
%MW140 := 16#0106
%MW140 := 16#0106
%MW141 := 16#0300
%MW141 := 16#0300
%MW142 := 21
%MW142 := 21
%MW142 := SHL(%MW14...
%MW142 := SHL(%MW142, 8)
%MW142 := %MW142 + 3
%MW142 := %MW142 + 3
%MW144 := 11 - 7
%MW144 := 11 - 7
EXCH2 %MW140:11
EXCH2 %MW140:11


LD
Rung7
WRITE 18 WORDS
Comment
Comment
WRITE_NW
%M5
Comment
Symbol
%MSG2.D
%MW304 := 27 - 9
%MW304 := 27 - 9
%MW305 := %MW304 * 2
%MW305 := %MW304 * 2
%MW300 := 1
%MW300 := 1
%MW300 := SHL(%MW30...
%MW300 := SHL(%MW300, 8)
%MW300 := %MW300 +...
%MW300 := %MW300 + %M...
%MW300 := %MW300 + 8
%MW300 := %MW300 + 8
%MW301 := 16#0007
%MW301 := 16#0007
%MW302 := 21
%MW302 := 21
%MW302 := SHL(%MW30...
%MW302 := SHL(%MW302, 8)
%MW302 := %MW302 + 16
%MW302 := %MW302 + 16
EXCH2 %MW300:27
EXCH2 %MW300:27

This project is functionally equivalent to the Twido project. It:

- Initializes the memory words when %M6 is set to TRUE. The addresses and the default values are set.
- Reads one bit when %M0 is set to TRUE. The result is stored in %MW107.
- Writes one bit when %M3 is set to TRUE. The value to write is stored in %MW124.
- Reads one word when %M1 is set to TRUE. The result is stored in %MW117.
- Writes one word when %M4 is set to TRUE. The value to write is stored in %MW134.
- Reads 4 words when %M2 is set to TRUE. The result is stored in %MW147 to %MW149.
- Writes 18 words when %M5 is set to TRUE. The values to write are stored in %MW306 to %MW323.

However, this project does not contain macro calls. The automatic conversion keeps the initialization part exactly the same, but converts each macro to a rung containing a single EXCH instruction.

As shown in the graphics, each macro is converted to instructions that fill up a table, which is called an exchange table. This table consists of at least 8 words (more when reading/writing multiple words). When this table is filled, the EXCH instruction is called and the communication occurs.

Exchange Table Examples

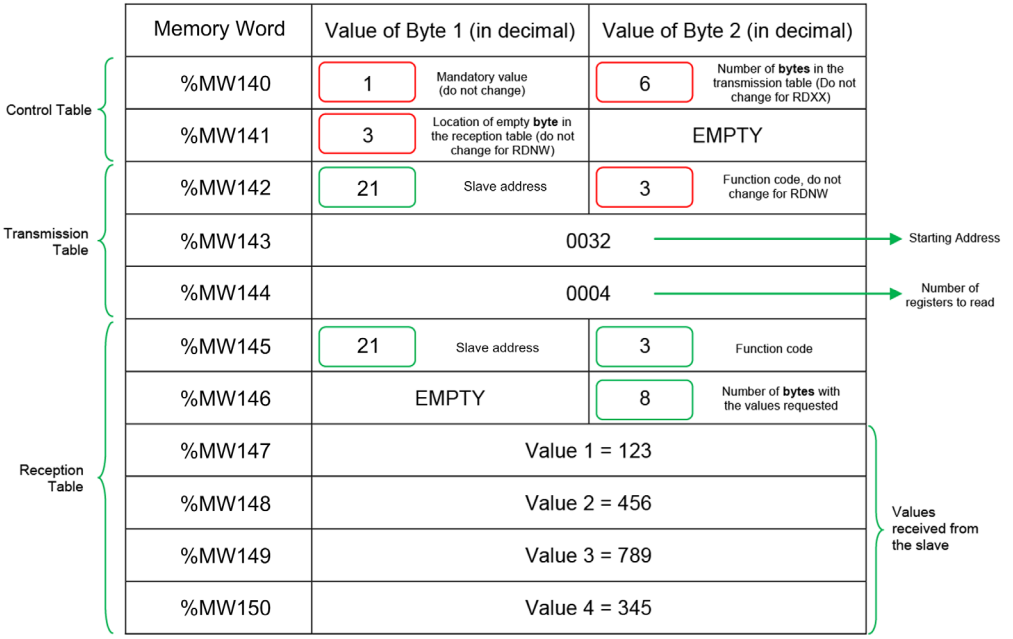
In the following exchange table, you can find an example of reading 4 words and writing 18 words. This example will help you to understand the exchange table. For other types of exchange, you can consult How to Modify EXCH Instructions of the Conversion *(see page 30)*.

Exchange Table for a Read Operation

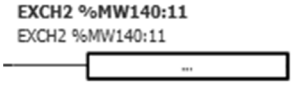
The following information is transmitted in the Modbus exchanges for a read operation:

The master requests...	The slave responds with...
Slave address	Address of the slave
Type of the operation (function code)	Function code
Address of the first register/bits requested	Number of registers/bits to follow
Number of registers/bits requested	Values of registers/bits requested

The following table presents a read operation of 4 registers on slave 21. The data read is stored starting at the address %MW147 with an EXCH instruction (11 memory words are used, starting from %MW140):



After filling the Control table and the Transmission table, it is possible to call the EXCH instruction using the following syntax:



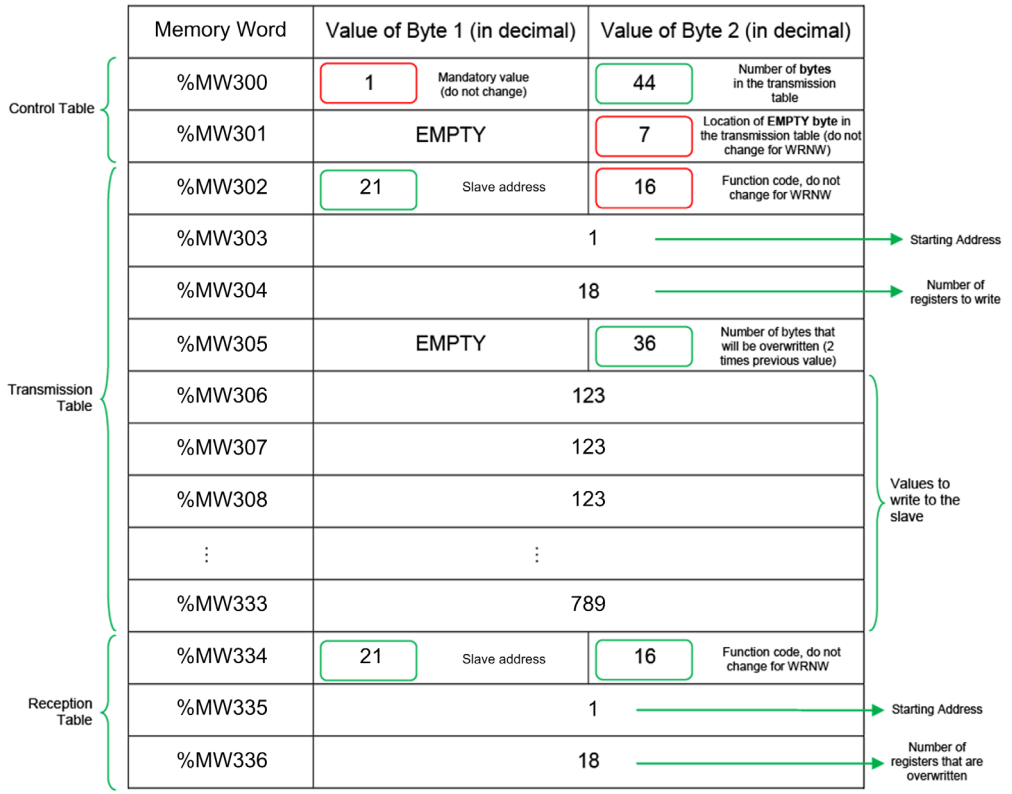
11 memory words are used in total, starting from %MW140. Calling the EXCH instruction uses these 11 memory words to execute the communication.

Exchange Table for a Write Operation

For a write operation (multiple registers or bits), the information to be transmitted is:

The master requests...	The slave responds with...
Slave address	Address of the slave
Type of the operation (function code)	Function code
Address of the first register/bits	Starting address of registers/bits that are overwritten
Number of registers/bits to be sent	Number of registers/bits that are overwritten
Values of the registers/bits	–

The following table presents a write operation of 18 registers on slave 21 at the address `%MW1` with `EXCH` instruction (27 memory words are used starting from `%MW300`):



Writing one particular register or bit follows different procedures, see the graphics in the following pages and also refer to *Modbus Master/Jbus Master/Ethernet TCP/IP Modbus Protocols Manual (11-2 Modbus Master requests)* for more information.

How to Modify EXCH Instructions of the Conversion

Introduction

The following explains how to use EXCH instructions in the EcoStruxure Machine Expert - Basic application, to obtain the same behavioral logic of the Twido COMM macros.

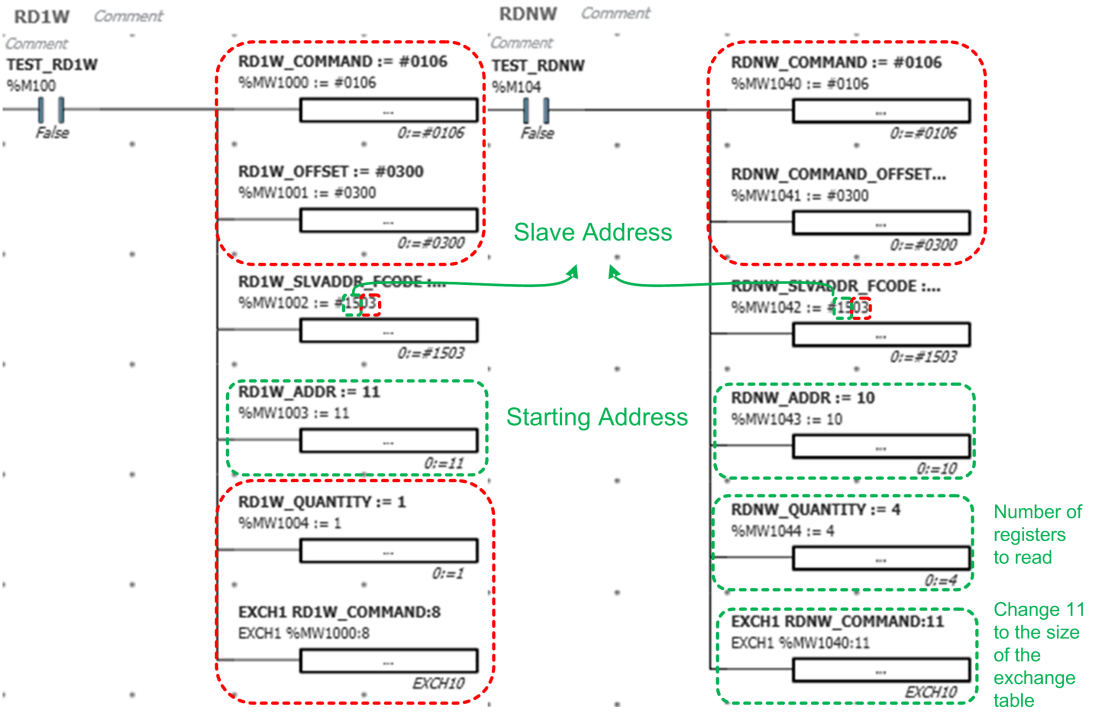
The following graphics present an EcoStruxure Machine Expert - Basic application that implements the Twido COMM macros prior to conversion and are included here to give you a better understanding of the EXCH instruction. With these graphics and the two exchange tables from the previous section, you can modify existing exchange tables and further create your own exchange tables.

In the following examples, areas highlighted in red must not be modified. Areas highlighted in green can be adapted to your application requirements.

Green areas correspond to the slave address, the number of registers, the values to write, or the size of the exchange table.

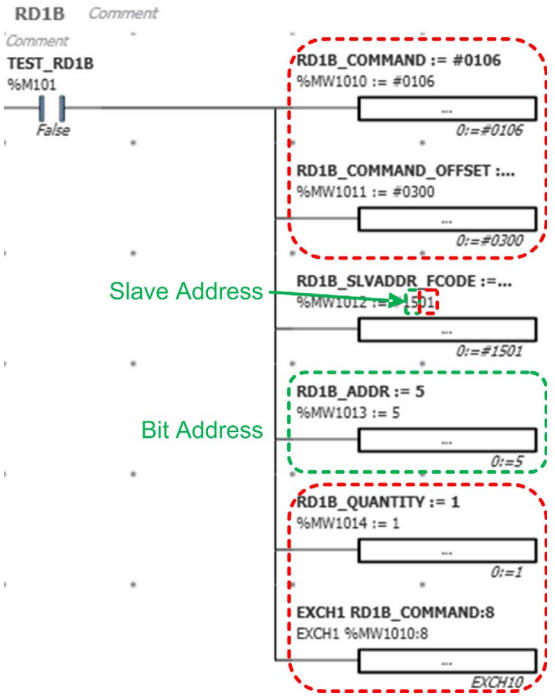
C_RD1W and C_RDNW Equivalents

Reading one word (C_RD1W equivalent) and reading four words (C_RDNW equivalent):



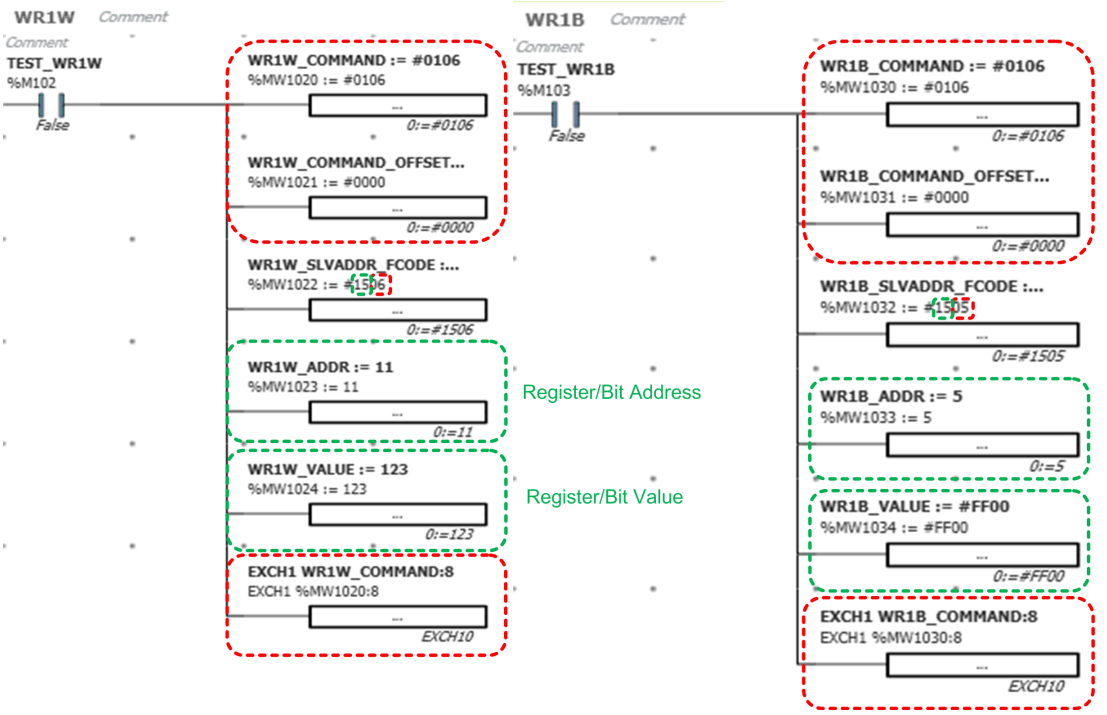
C_RD1B Equivalent

Reading one bit (C_RD1B equivalent):



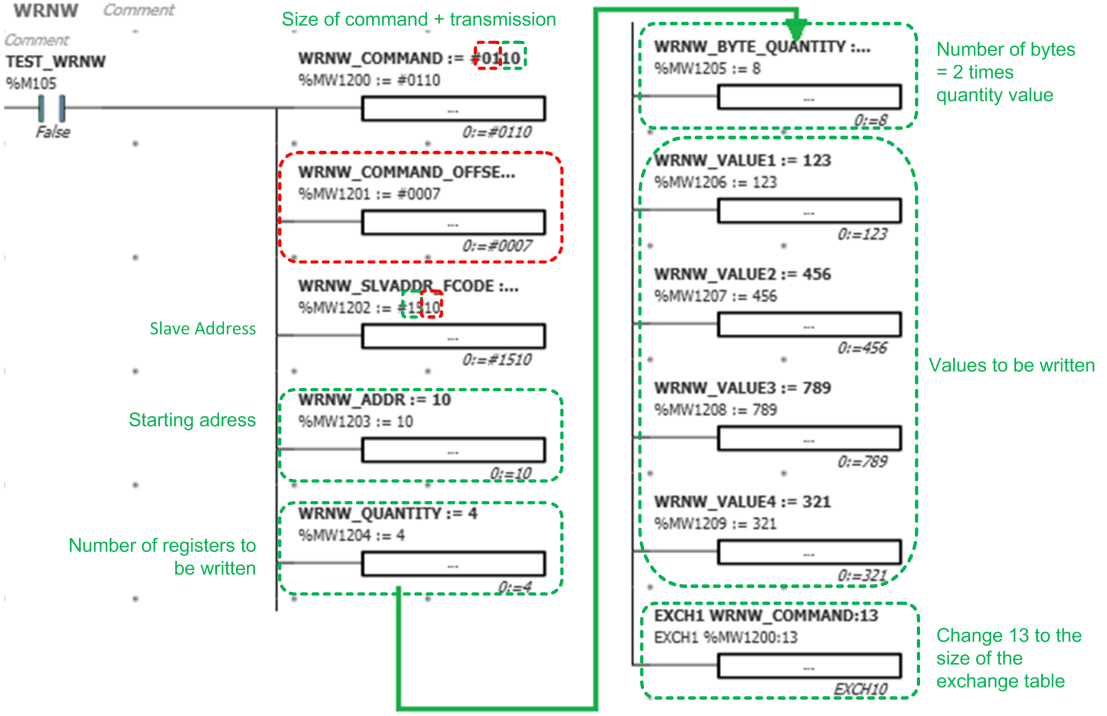
C_WR1W and C_WR1B Equivalents

Writing one word (C_WR1W equivalent) and writing one bit (C_WR1B equivalent):



C_WRNW Equivalent

Writing multiple words (C_WRNW equivalent):



NOTE: This is a simplified graphic presenting a write operation of 4 memory words only.

Unlike the previous graphics, these graphics present how to manually reproduce a Twido COMM macro in EcoStruxure Machine Expert - Basic, but they are not limited to this operation. For example, it is possible to read or write multiple bits.

Always fill the Control and Transmission tables and verify that the Reception table is filled with the data sent by the slave. The number of bytes in the first word is updated according to the amount of received data. The values read from the slave are found in the Reception table area of the exchange table.

Although you can use EXCH instructions, use Read Var and Write Var function blocks to create more readable code that is adapted to your application.

Section 1.2
Example Detailed Description

What Is in This Section?

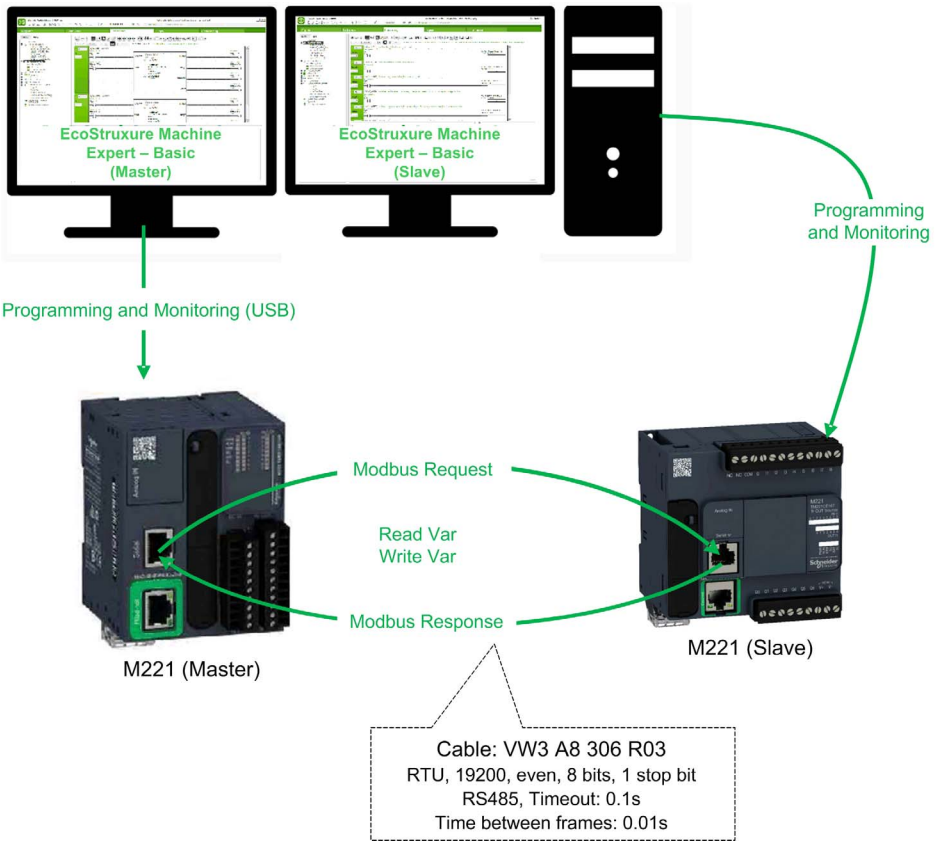
This section contains the following topics:

How to Replace Twido COMM Macros or EXCH Instructions by Function Blocks in EcoStruxure Machine Expert - Basic

Hardware Configuration Used Within the Example

This example presents an M221 Logic Controller as the equipment with a Modbus interface, that is, as the slave.

The setup is shown below:



The setup used in the project template comprises:

- An M221 Logic Controller, for example, TM221CE16T as slave.
- An M221 Logic Controller, for example, TM221CE40T as master.
- A PC with EcoStruxure Machine Expert - Basic to program the M221 master controller and monitor the application example.
- A PC with another instance of EcoStruxure Machine Expert - Basic to program and monitor the M221 slave controller.

The example uses an M221 Logic Controller as slave. However, another equipment that supports the Modbus protocol can be used. If this is the case, refer to the equipment documentation to verify that the Modbus registers used by this example are compatible. Adapt the configuration and the code of the example as necessary.

The other EcoStruxure Machine Expert - Basic project used in the slave for this example is also provided to allow you to reproduce the template (`xSample_M221_COMM_Conv_Slave`). You can also find the graphics corresponding to this project in the appendices.

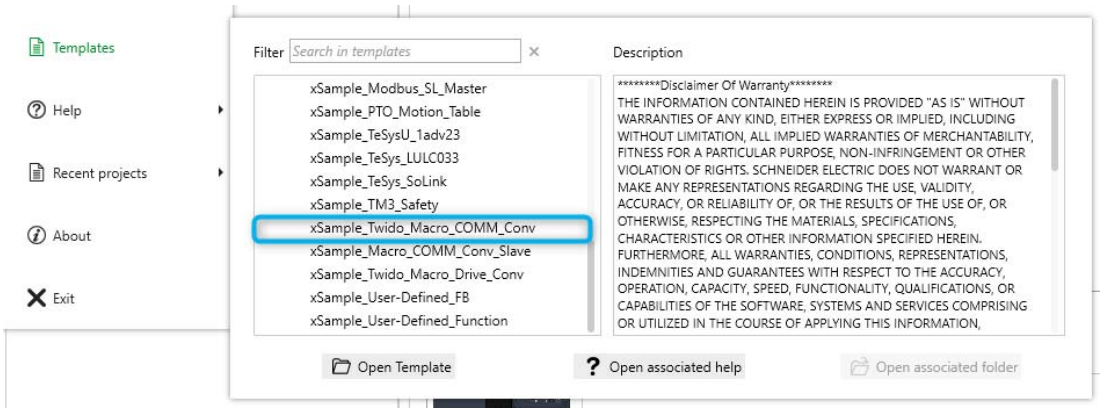
The template is designed to be integrated into your application. Comments in each object provide explanations on how to integrate the example into your application. Animation tables are provided within the template example to assist you with integration and deployment.

This example is prepared assuming that you have prior knowledge of using Twido COMM macros and that you have a working, functional TwidoSuite project.

Opening Your Example

From this point onwards, open the example and follow this document with it. To open the example template, click  in the top left corner, select `xSample_Twido_Marco_COMM_Conversion.smbe`, and click **Open Template**.

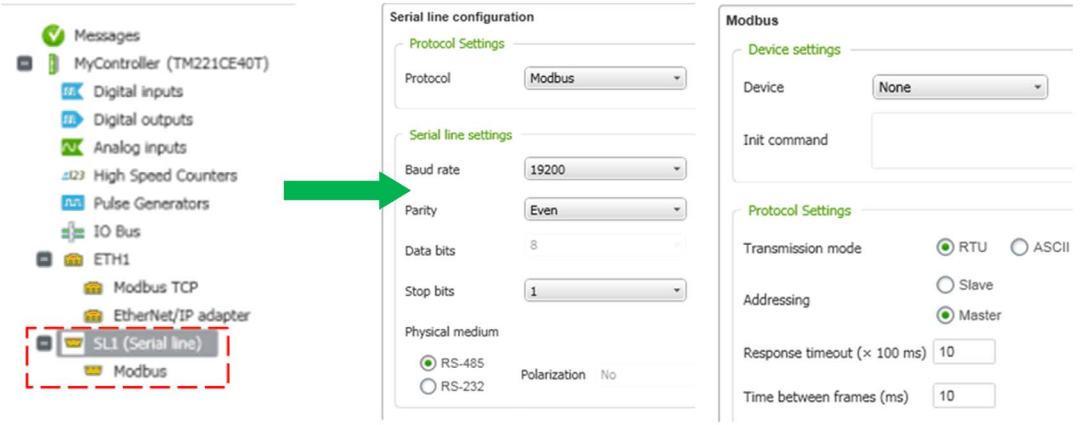
The following figure shows the location of the example in EcoStruxure Machine Expert - Basic:



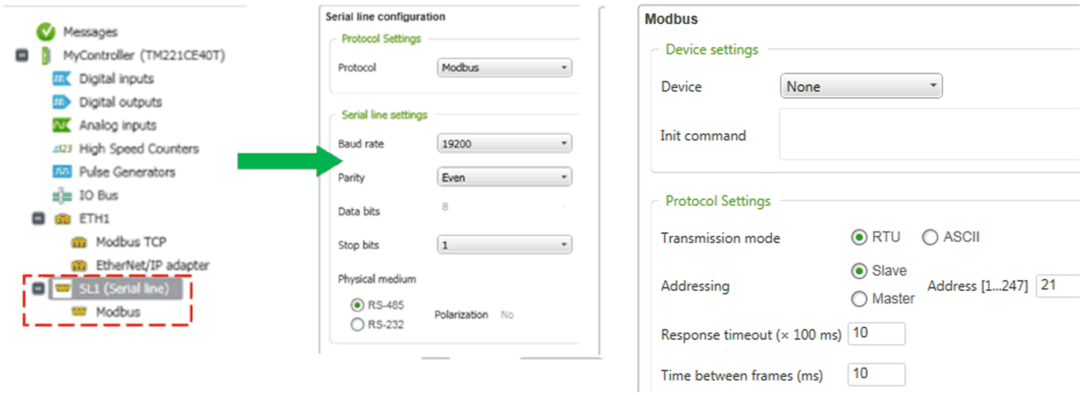
Software Configuration

Verify that the Serial Line communication configuration for both M221 controllers is identical (the Modbus settings will differ between Master, Slave and Addressing). Set the correct parameters for Modbus communications on a serial line (**SL1** or **SL2**). The Modbus protocol needs to be configured with the following settings for both M221 controllers.

Settings for the M221 master controller in EcoStruxure Machine Expert - Basic **Configuration** tab:



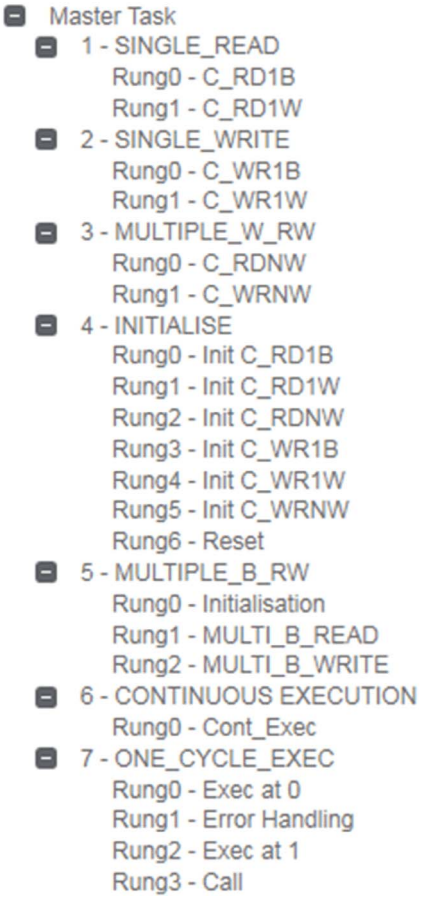
Settings for the M221 slave controller in EcoStruxure Machine Expert - Basic **Configuration** tab (the difference is that it is configured as a slave):



Program Structure, Variables, and Function Blocks Used

The EcoStruxure Machine Expert - Basic example is divided into a number of POUs and rungs to facilitate identification of the parts corresponding to individual Twido macros. You can delete rungs that are not required for your application.

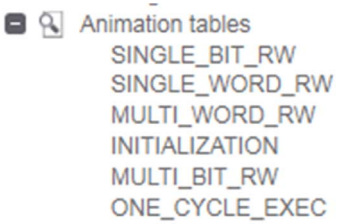
This figure shows the organization of the EcoStruxure Machine Expert - Basic example:



- POU 1 reads one bit or one word (`C_RD1B` and `C_RD1W`, respectively)
- POU 2 writes one bit or one word (`C_WR1B` and `C_WR1W`, respectively)
- POU 3 reads or writes multiple words (`C_RDNW` and `C_WRNW`, respectively)
- POU 4 initializes the values that will be read or written. Each rung corresponds to the initialization of a different function block (for example, a macro).
- POU 5 reads or writes multiple bits
- POU 6 performs continuous execution

There are also a number of animation tables. The first 4 animation tables relate to the basic functionalities. The INITIALIZATION table is to initialize the first 3 animation tables. The other tables are independent and correspond to POUs 5 and 7 respectively.

This figure shows the provided animation tables:



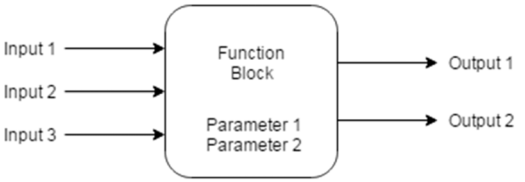
Function Block Principles

Since the proposed method for converting COMM macros involves function blocks in EcoStruxure Machine Expert - Basic, it is necessary to know how to use function blocks.

A function block is an object with inputs, outputs, and parameters. Depending on the values of the inputs and parameters, it produces predetermined output values. Read the documentation for each function block before using it.

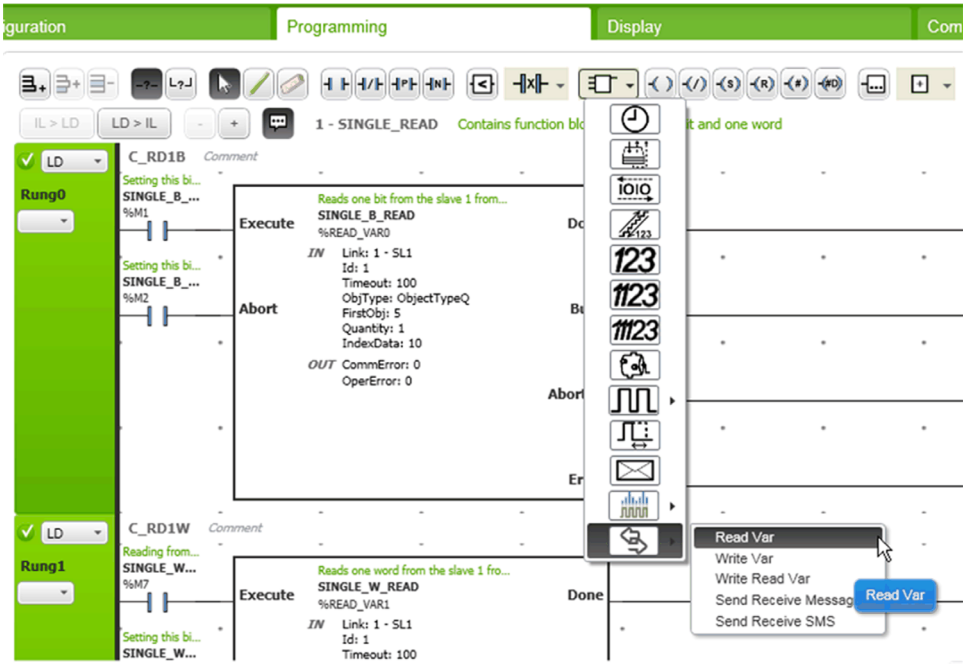
A Ladder diagram in EcoStruxure Machine Expert - Basic can only contain one function block in each rung. However, every function block type can have multiple copies (instances), each of which can be configured differently.

This figure shows an example function block:



The function blocks used in this example are `Read Var` and `Write Var` which are located in the **Communication Objects** section of the **Tools** tree. There is also another function block available called `Write Read Var` which can read and write within a single function block instance.

This figure shows the communication function blocks in EcoStruxure Machine Expert - Basic:

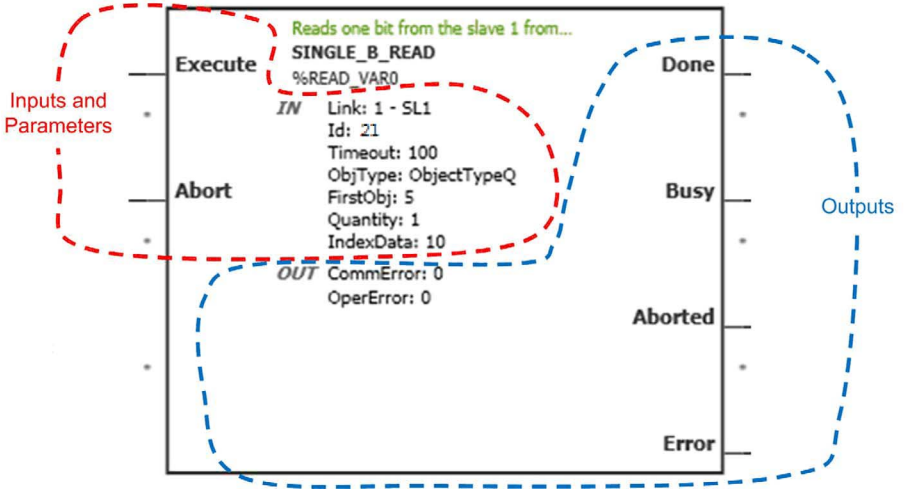


`Read Var` Function Block

Introduction

This function block is used to replace the Twido macros `C_RD1B`, `C_RD1W`, and `C_RDNW`.

Reading a single bit:



With EcoStruxure Machine Expert - Basic, you can use this function to read different types of objects. **ObjType** is the parameter that manages this:

- `ObjectTypeMW` reads the holding registers (memory word objects (`%MW`) on the M221 Logic Controller) of the slave by sending function code `0x03`.
- `ObjectTypeM` reads the holding bits (memory bit objects (`%M`) on the M221 Logic Controller) of the slave. This parameter needs to be changed to bits (Mbs Fct1). With this configuration, the function code `0x01` is sent.

The **IndexData** parameter always corresponds to the holding registers of the master. Therefore, even if a bit is read, its value is stored in a word. When reading only one bit, the value of **IndexData** corresponds (in decimal) to the value of the bit read. In binary, the value is stored in the least significant bit.

This figure illustrates the above operation which is valid also for `Write Var`.

Memory Word	Value in binary	Corresponds to :
%MW10	0000000000000001	IndexData

Memory Bit	Bit Value	Corresponds to :
%M1	1	FirstObj

Upon initialization, the value in **IndexData** for all the read functions is set to 0xFFFF. This value can then be set to any value, so if you read a known value from the slave, you can verify that the communication is successful.

Even though the `EXCH` instruction still exists in EcoStruxure Machine Expert - Basic, `Read Var` is more powerful. First, it is less restrictive than `EXCH` and conforms to the Modbus specifications by allowing a maximum of 125 words, compared to 114 words in Twido. Second, it combines a Modbus request and error management in a single instruction, making the `MSG` function block obsolete. Also you can save memory words (`%MW`) in your application since the instruction does not need an exchange table.

Allow reserved memory after **IndexData** when you are reading multiple words. The amount of reserved memory should correspond to the **Quantity** value. Otherwise, as written in the comments of the example, the data stored will be overwritten. This is an example:

Memory Word	Value Read	Corresponds to:
`%MW412`	140	`IndexData`
`%MW413`	150	`IndexData+1`
`%MW414`	155	`IndexData+2`
...		
`%MW456`	160	`IndexData+Quantity-2`
`%MW457`	155	`IndexData+Quantity-1`
`%MW458`	140	`IndexData+Quantity`

In this example, **IndexData** corresponds to `%MW412`. After execution, all the addresses after `%MW412` are filled with the values from the slave up to the memory word `IndexData+Quantity`. In this example, the **Quantity** value is 47.

Matching `Read Var` Parameters to Twido COMM Macros

The following information shows how to match the parameters of Twido read macros (`C_RDXX`) with the parameters of `Read Var`:

- `C_RD1B`:
  - **ObjType** : 2 (Coils – Mbs 1)
  - **FirstObj** : `C_RD1B_ADDR_x`
  - **Quantity** : 1
  - **IndexData**: Address of `C_RD1B_VAL_x`
- `C_RD1W`:
  - **ObjType** : 0 (Holding reg. – Mbs 3)
  - **FirstObj** : `C_RD1W_ADDR_x`
  - **Quantity** : 1
  - **IndexData**: Address of `C_RD1W_VAL_x`

- C_RDNW:
  - **ObjType** : 0 (Holding reg. – Mbs 3)
  - **FirstObj** : C_RDNW_ADDR_x
  - **Quantity** : Parameter 1 of the macro
  - **IndexData**: Address and the following words (until IndexData+Quantity) corresponds to C_RDNW_VAL_x and the following addresses (depending on the value of parameter1=N)

  IndexData does not need to be 4 words after **FirstObj** as was the case for Twido macros (C_RD1B_VAL_x was 4 words after C_RD1B_ADDR_x). This way you have more flexibility to manage the memory.

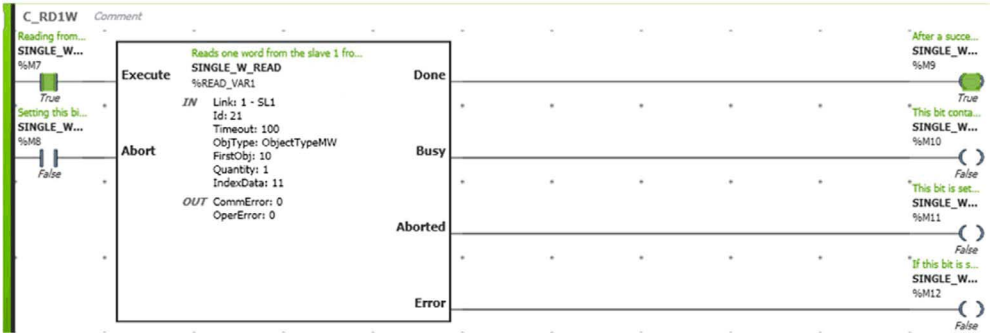
Read Var Behavior

In the example, every input and output of the function block is connected, but this is not mandatory. Read Var is also able to operate with just the **Execute** input connected. However, having a separate bit for each output visually displays the state of the function block.

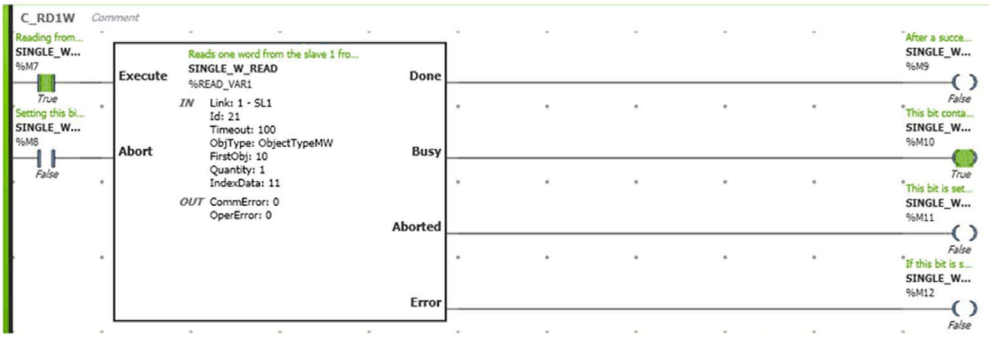
The following information presents the different states of the Read Var function block with graphics. Together with the timing diagrams, these graphics help you to better understand the function block. In the timing diagrams, each vertical dash corresponds to an execution cycle.

Normal Execution of Read Var

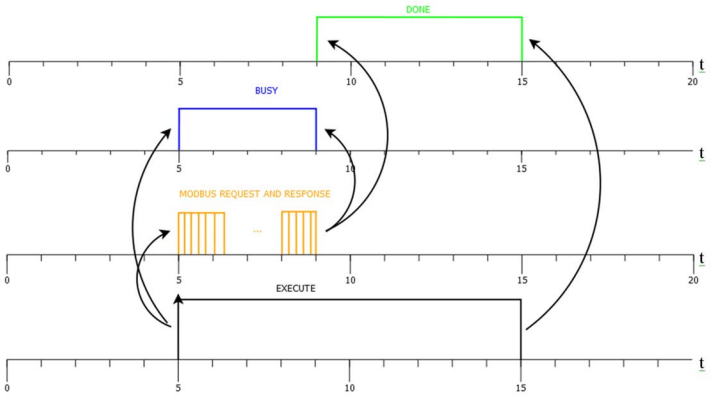
Normal functioning: After an execution:



The execution process: Normally, this state is very short and it is normal to not see it. However, if there is a communication interruption, the block will be in this state until the timeout is reached.

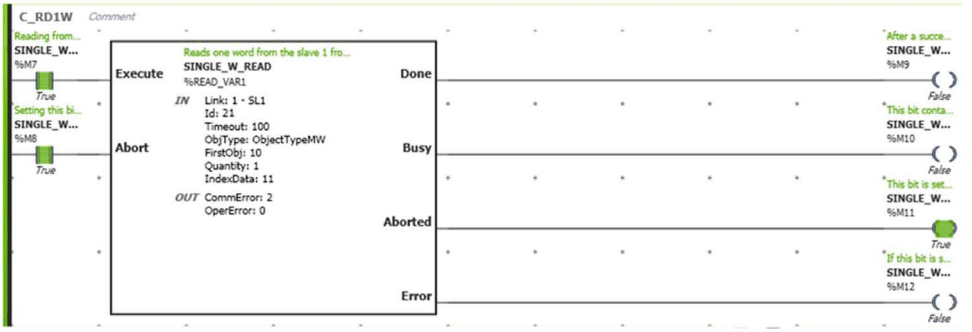


Timing Diagram 1: Normal functioning. On a rising edge of **Execute**, the Modbus request starts and the **Busy** output is set to TRUE. When the exchange is complete (response received) and correct, the **Done** output is set to TRUE.

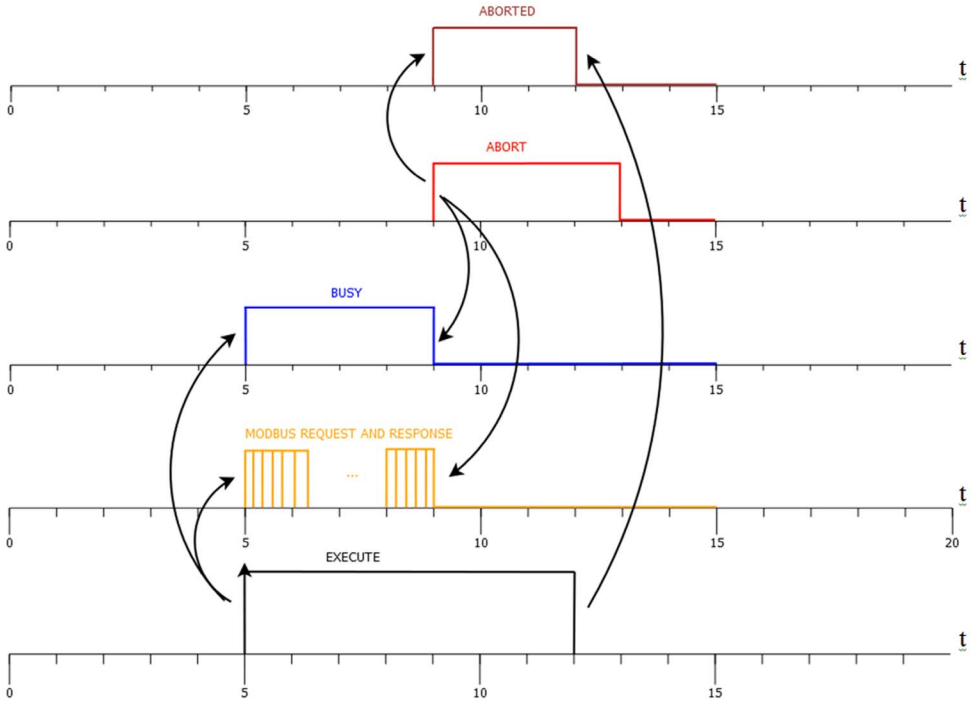


Abort Input of `Read Var`

It is possible to stop a running exchange. To do so, use the **Abort** input of the function block:

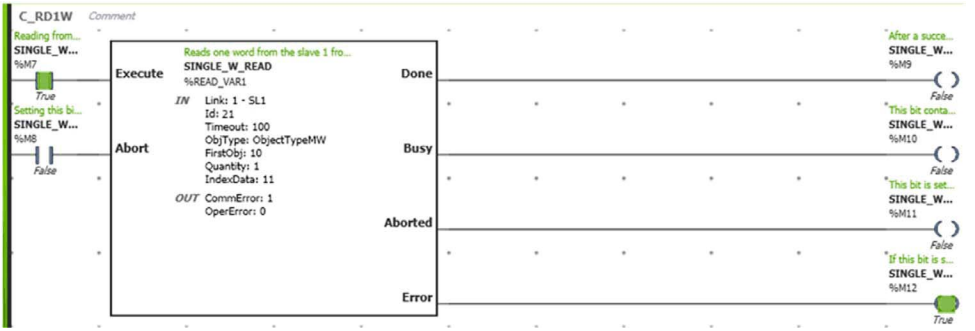


During an execution, setting the **Abort** input to TRUE stops the process and sets the **Aborted** output to TRUE. **Aborted** output is not set to TRUE if the **Abort** input is set to TRUE outside of an execution. The following timing diagram corresponds to this state.

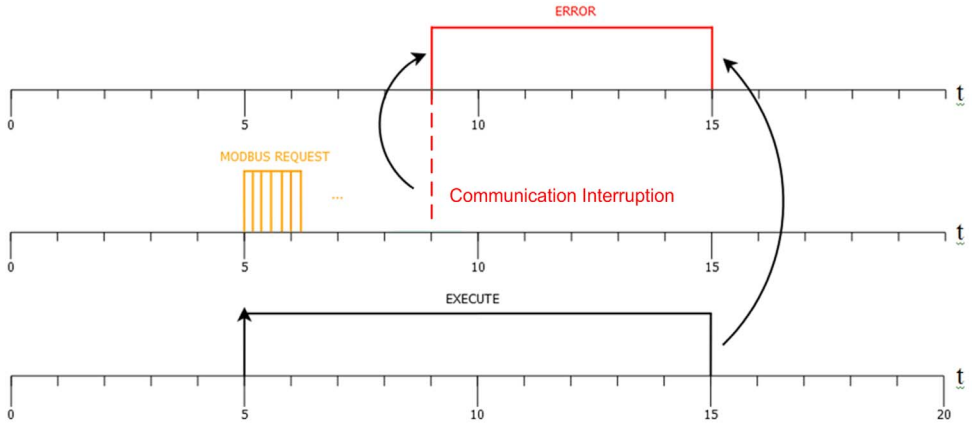


Timing Diagram 2: If the **Abort** input is set to TRUE during the exchange, the exchange stops, the **Busy** output is set to FALSE and the **Aborted** output is set to TRUE. When the **Execute** input is reset, all output parameters return to their initial state of FALSE.

Error Management with `Read Var`



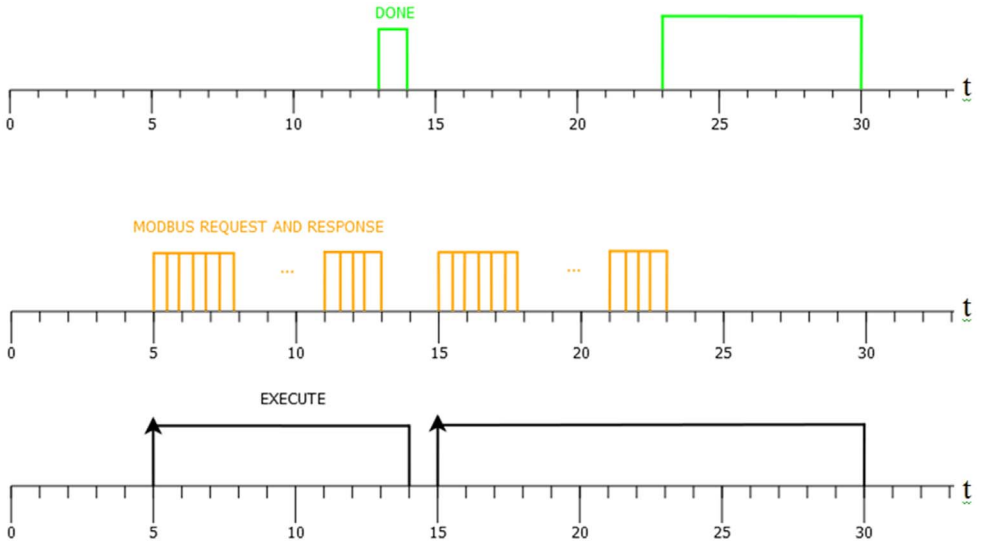
If there is a communication interruption, the **Error** output is set to TRUE at each execution attempt.



The **Error** output is maintained to TRUE until the **Execute** input is reset. While the **Error** output is set to TRUE, it is possible to read the `CommError` and `OperError` outputs for diagnostic purposes.

Multiple Execution of `Read Var`

The following timing diagram presents the functioning of multiple `Read Var` function blocks:



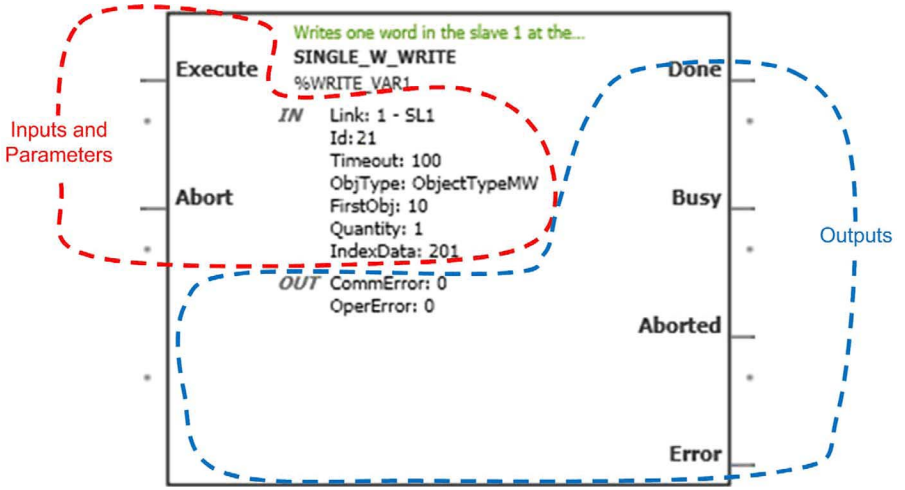
Since every dash corresponds to an execution cycle, you can see that an execution cycle can be lost between each execution of `Read Var`, so no extra modifications are made in the application.

`Write Var` Function Block

Introduction

This function block replaces the Twido macros `C_WR1B`, `C_WR1W`, and `C_WRNW`.

The figure shows writing a single bit.



With EcoStruxure Machine Expert - Basic, you can use this function to write different type of objects. **ObjType** is the parameter that manages this:

- `ObjectTypeMW` writes the holding registers (memory words `%MW`) of the slave by sending function code 0x10.
- `ObjectTypeM` writes the holding bits (memory bits `%M`) of the slave. This parameter needs to be changed to bit (Mbs Fct 1). With this configuration, the function code 0x0F is sent.

The **IndexData** parameter corresponds to the holding registers of the master. So, even if a bit is written, the value needs to be stored in a memory word. When writing only one bit, the decimal value of the **IndexData** parameter corresponds to the bit value to be written. The value is stored in the least significant bit.

On initialization, change the value in **IndexData** according to your application.

Although the `EXCH` instruction still exists in EcoStruxure Machine Expert - Basic, `Write Var` is more powerful. Firstly, it is less restrictive than `EXCH` and conforms to the Modbus specifications by having the maximum 123 words, compared to 114 in Twido. Furthermore, it combines Modbus request and error management in a single block, making the `MSG` function block obsolete. Also, you save some memory words `%MW` in your application, as the function block does not need an exchange table.

NOTE: `Write Var` uses the function codes 0x0F for writing bits and 0x10 for writing words, even if **Quantity** is 1. If your Modbus device does not manage multiple write function codes, you must keep the `EXCH` instruction generated by the conversion. Furthermore, `C_WR1W_ADDRW_x`, `C_WR1W_VALW_x`, `C_WR1B_ADDRW_x` and `C_WR1B_VALW_x` do not have an equivalent when using the `Write Var` function block.

Matching `Write Var` parameters to Twido COMM Macros

The following information presents how to match the parameters of Twido write macros (`C_WRXX`) to the parameters of `Write Var` in EcoStruxure Machine Expert - Basic:

- `C_WR1B`:
  - **ObjType** : 2 (Coils – Mbs 1)
  - **FirstObj** : `C_WR1B_ADDR_x`
  - **Quantity** : 1
  - **IndexData**: Address of `C_WR1B_VAL_x`
- `C_WR1W`:
  - **ObjType** : 0 (Holding reg. – Mbs 3)
  - **FirstObj** : `C_WR1W_ADDR_x`
  - **Quantity** : 1
  - **IndexData**: Address of `C_WR1W_VAL_x`
- `C_WRNW`:
  - **ObjType** : 0 (Holding reg. – Mbs 3)
  - **FirstObj** : `C_WRNW_ADDR_x`
  - **Quantity** : Parameter 1 of the macro
  - **IndexData**: Address and the following memory words (until `IndexData+Quantity`) correspond to `C_WRNW_VAL_x` and the following addresses (depending on the value of parameter1=N)

  IndexData does not need to be 4 words after **FirstObj** as was the case for Twido macros (`C_WR1B_VAL_x` was 4 words after `C_WR1B_ADDR_x`). This way, you have more flexibility to manage the memory.

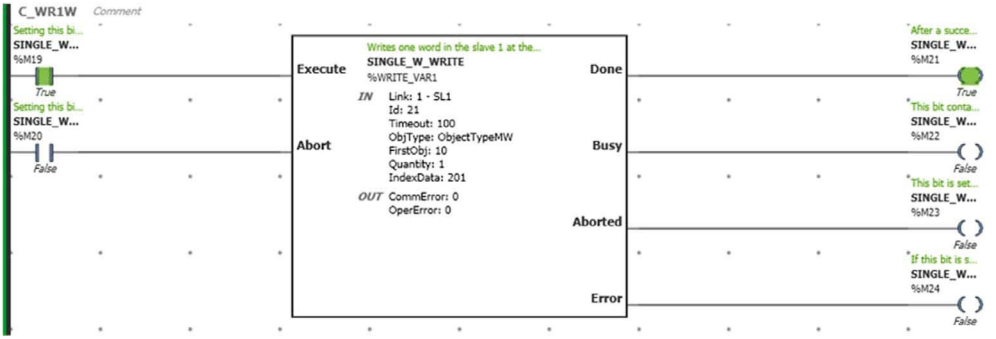
`Write Var` Behavior

In the example, every input and output of the function block is connected. This is not mandatory. `Write Var` is also able to function with just the **Execute** input connected. However, having a separate bit for each output visually shows the state of the function block.

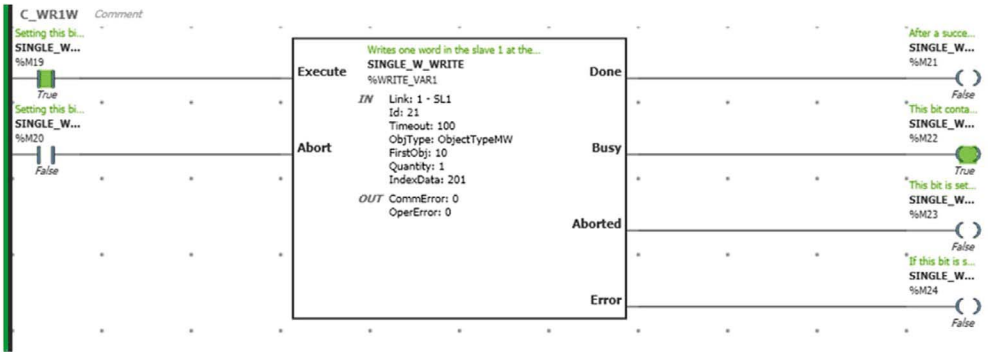
The following information presents the different states of the `Write Var` function block with graphics. Together with the timing diagrams, these graphics help you to better understand the function block. In the timing diagrams, each vertical dash corresponds to an execution cycle.

Normal Execution of `Write Var`

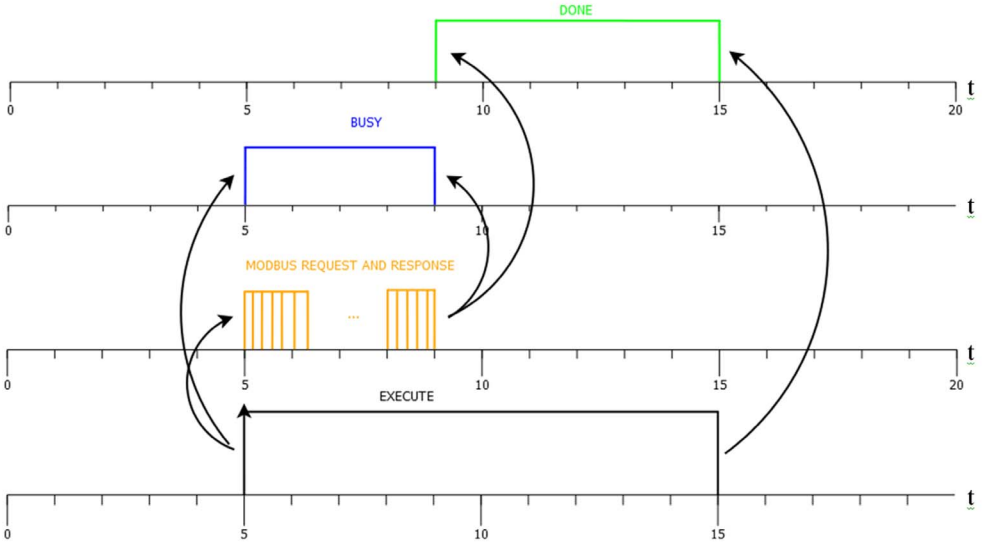
Normal functioning: After an execution.



The execution process: Normally this state is very short and it is normal to not see it. However, if there is a communication interruption, the block remains in this state until the timeout is reached.

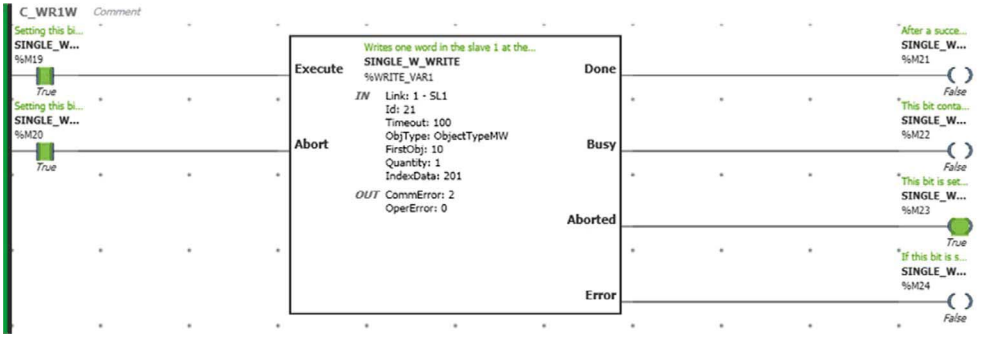


Timing Diagram 1: Normal functioning. On a rising edge of **Execute**, the Modbus request starts and the **Busy** output is set to TRUE. When the exchange is complete (response received) and correct, the **Done** output is set to TRUE.

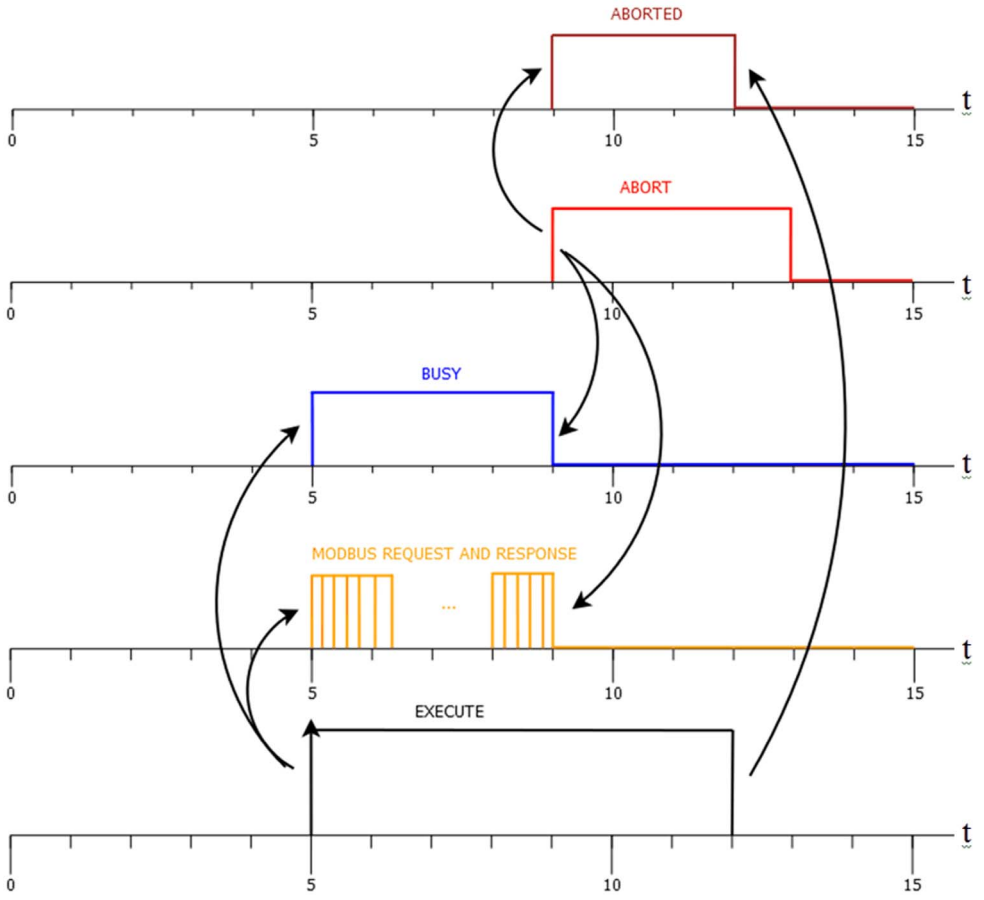


Abort Input of `Write Var`

It is possible to stop a running exchange. To do so, use the **Abort** input of the function block.

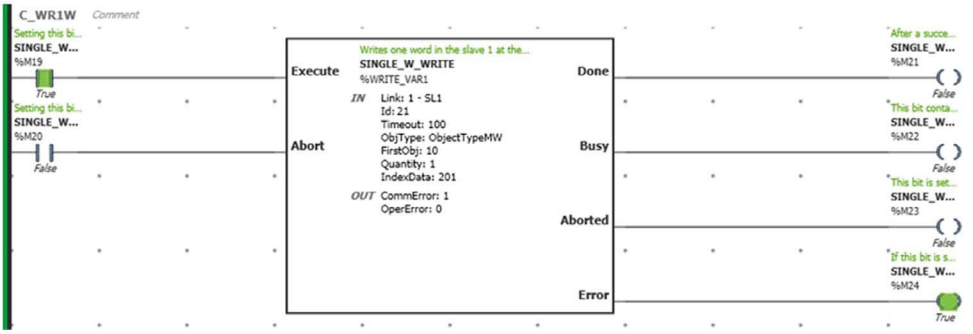


During execution, the **Abort** input stops the process and sets the **Aborted** output to TRUE. The **Aborted** output is not set to TRUE if the **Abort** input is set to TRUE outside of an execution. The following timing diagram shows this state.

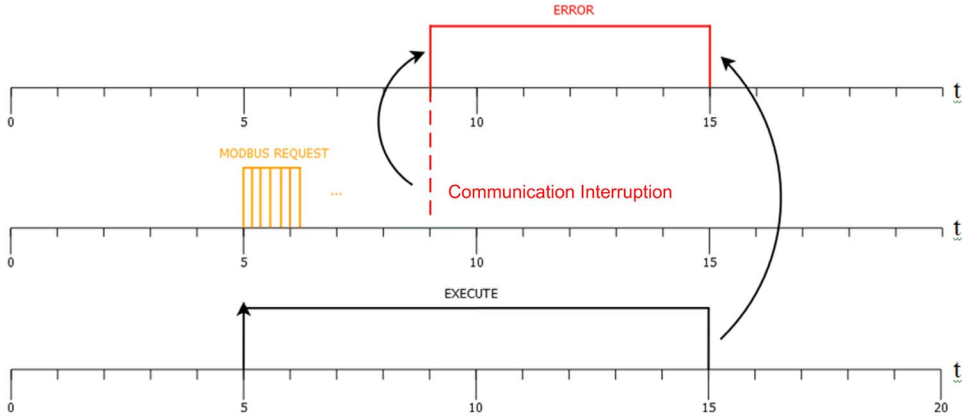


Timing Diagram 2: If the **Abort** input is set to TRUE during the exchange, the exchange stops, the **Busy** output is set to FALSE and the **Aborted** output is set to TRUE. When the **Execute** input is reset, all the output parameters return to the initial state of FALSE.

Error Management with `Write Var`



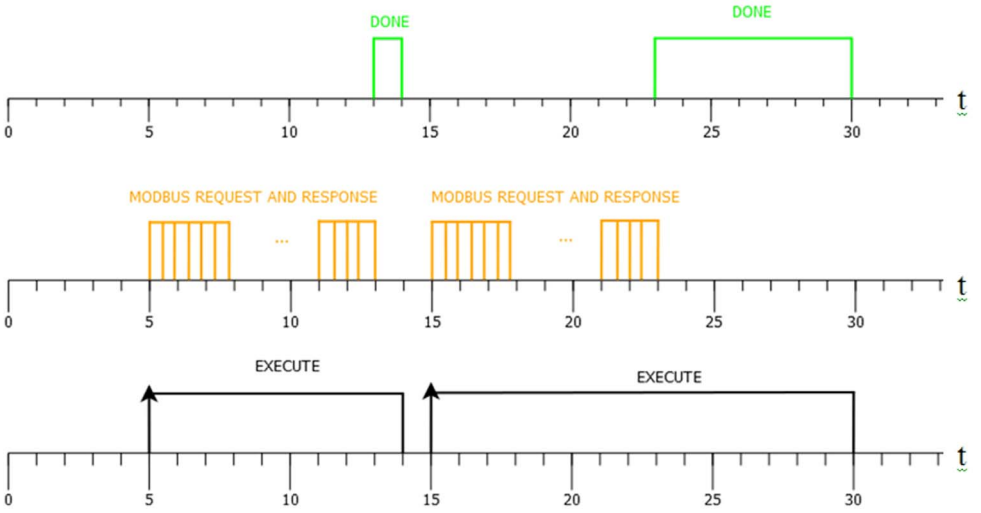
If there is a communication interruption, the **Error** output is set to TRUE at each execution attempt.



The **Error** output is maintained at TRUE until the **Execute** input is reset. While the **Error** output is set to TRUE, it is possible to read the `CommError` and `OperError` fields for diagnostic purposes.

Multiple Execution of `Write Var`

The following timing diagram presents the functioning of multiple `Write Var` function blocks.



Each dash corresponds to an execution cycle. An execution cycle can be lost between each execution of `Read Var`, so no extra modifications are made in the application.

Appendices

Appendix A
Appendices

What Is in This Chapter?

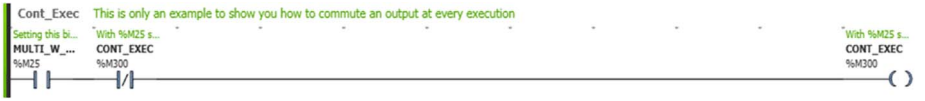
This chapter contains the following topics:

Advanced Function Block Adaptations

How to Start an Exchange in One Cycle

The `Read Var` and `Write Var` function blocks need a rising edge on the **Execute** input pin to restart an exchange. In contrast, the `EXCH` instruction does not. The **Execute** input provides a simpler way to program, but may introduce one cycle of latency.

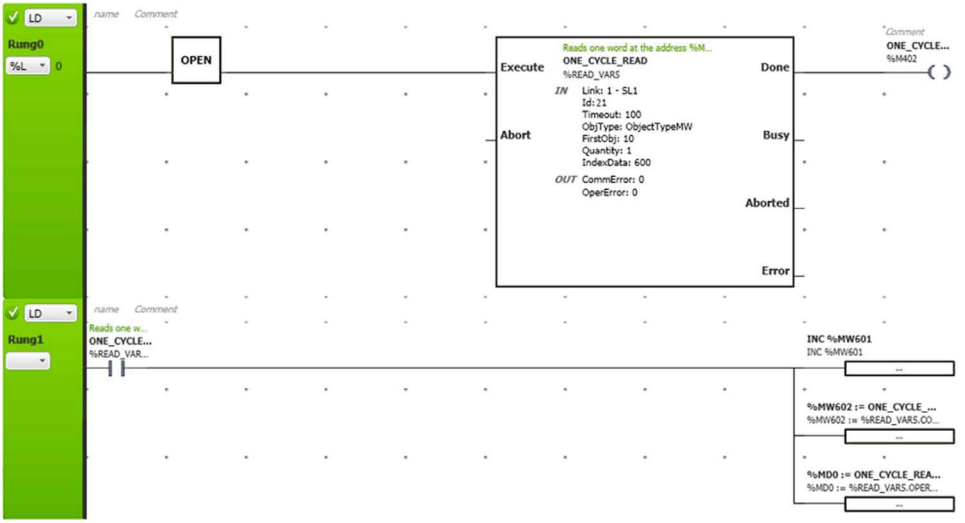
The rung below shows a typical way of programming a `Read Var` that introduces this latency:

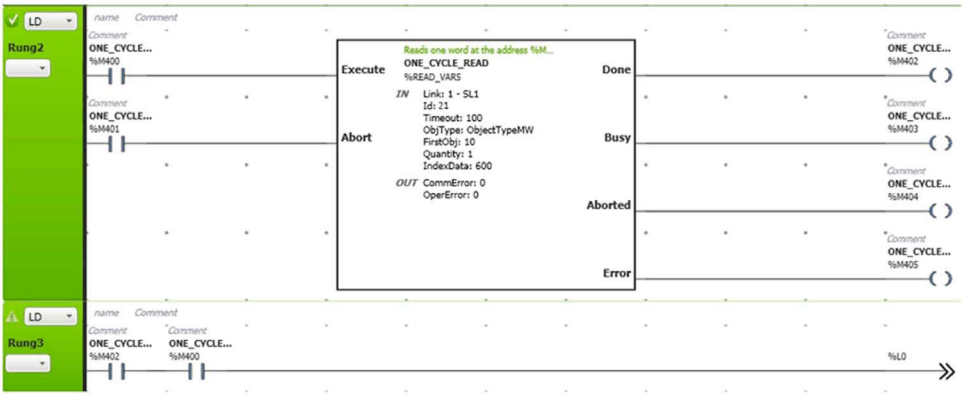


NOTE: POU 6: Execution every two cycles.

In POU 6, there is an example showing this operation. `%M25`, which normally executes the function block `%READ_VAR2`, is now enabling `%M300` to switch to 1 every two execution cycles. If, by using the **Execute** input of `%READ_VAR2`, you replace `%M25` with `%M300`, the function block is executed every two cycles.

The proposed method to resolve this is to call the function block twice in one execution cycle to immediately start the block. This solution is used in POU 7 and also shown below:





NOTE: POU 7: Execution at each cycle.

This method provides the same behavior as a Twido macro. A brief explanation of this technique is based on the initialization of this function block.

At Rung 0, the function block is called with the **Execute** input at 0, which resets all the outputs. After this, it is called again but with **Execute** at 1, which starts the communication. Once the communication is complete, it jumps to the Rung 0 during the same execution cycle, which reinitializes the block. Once the block is reinitialized, it is ready to be used again at the next execution cycle.

Handling Communication Interruptions

In most applications, the communication is managed through analyzing the status of **Done** and **Error**. For a simple management such as this, `Read Var` and `Write Var` are improvements compared to TwidoSuite. As they have integrated **Done** and **Error** outputs, they are more practical to use and do not require another block.

In TwidoSuite, the `MSG` block handles the status of the communication while communication function blocks integrate this feature and also the detected error codes. In EcoStruxure Machine Expert - Basic, communication status can be read from the `CommError` and `OperError` outputs of the communication function blocks.

For example, an error code in Twido such as `%SW63` = 5, which corresponds to a timeout, is `CommError` = 1 in EcoStruxure Machine Expert - Basic. For the other codes you may encounter, refer to the error code explanations *(see EcoStruxure Machine Expert - Basic, Generic Functions Library Guide)*.

Although not exactly the same, it is possible to match the inputs and outputs of the `MSG` function to the inputs and outputs of `Read Var` and `Write Var`. The `MSG` function uses the **R** (Reset input) input to stop the communication in progress and then reinitialize the variables. In EcoStruxure Machine Expert - Basic, you can stop a communication in progress with the **Abort** input, and this change is indicated by the **Aborted** output.

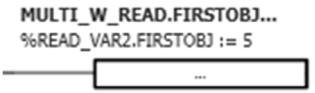
Even though the outputs (**Error** and **Done**) have the same names in the Twido and EcoStruxure Machine Expert - Basic functions, they have slightly different behaviors. When there is a communication interruption, the `MSG` function outputs **Done** and **Error** are set to TRUE, whereas in the EcoStruxure Machine Expert - Basic communication function blocks, only the **Error** output is set to TRUE while **Done** remains at FALSE.

Moreover, before execution the EcoStruxure Machine Expert - Basic communication function block, the **Done** output isat FALSE, whereas the **Done** output of the `MSG` function block is at TRUE.

Dynamic Modification

Every parameter of `Read Var` and `Write Var` can be modified dynamically. This means that you can modify parameter values while the program is running.

Dynamic parameter value updates are done in an operation block. This is an example of dynamic modification:

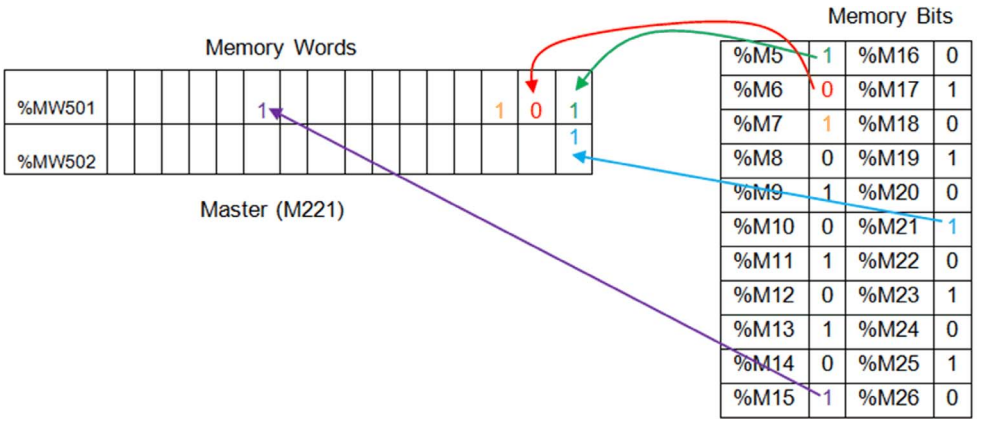


Dynamically changing the parameter values of a function block allows you to:

- Address several slaves with a block, so you can have more than 16 slaves on the same line.
- Exchange several items of data with one block (for example, reading 200 words that are not continuous).

Reading and Writing Multiple Bits

With EcoStruxure Machine Expert - Basic and the communication function blocks, it is possible to read and write multiple bits. This is also done by the `Read Var` and `Write Var` function blocks. Since these operations are done through the memory words and not by the bits, it is not straightforward. POU 5 is dedicated to these operations. This figure illustrates their operation.



The same logic is applied to both reading and writing. When a bit is read from the slave, it is stored in the least significant bit of the memory word indicated in the **IndexData** parameter. The following bits are stored in the next least significant bits. The 16th bit read is stored in the most significant bit. When more than 16 bits are read, that is, the size of a word, the values are stored in the next memory word in the same way.

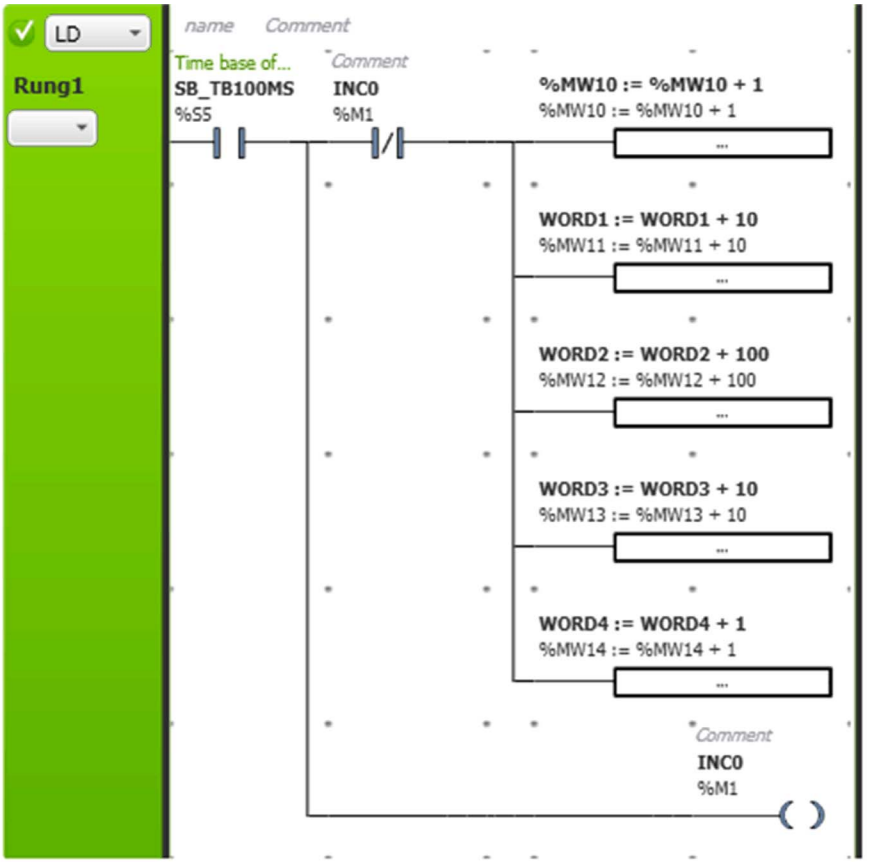
The maximum number of bits that can be read is 2000, which corresponds to 125 (maximum number of words to read) times 16 (size of a word). The maximum number of bits that can be written is 1968, which corresponds to 123 (the maximum number of words to write) times 16 (size of a word).

M221 Application Used in the Slave

Presentation

This is a simple application that is used with this example (as the slave for `Read Var` and `Write Var`).

This figure shows the incrementation of five registers:



Five registers are incremented every 100 ms by 1, 10, 100, 10, and 1 for `%MW10`, `%MW11`, `%MW12`, `%MW13`, `%MW14` respectively. These five memory words are used with `Read Var` and `Write Var` function blocks.