Modicon M262 Logic/Motion Controller

Encoder Library Guide

05/2019

EIO0000003675.00

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.

About the Book

At a Glance

Document Scope

This document will acquaint you with the encoder functions and variables offered within the M262 Logic/Motion Controller. The M262 Logic/Motion Controller Encoder library contains functions and variables to get information from and send commands to the encoder system.

This document describes the data type functions and variables of the M262 Logic/Motion Controller Encoder library.

The following knowledge is required:

- Basic information on the functionality, structure, and configuration of the M262 Logic/Motion Controller.
- Programming in the FBD, LD, ST, IL, or CFC language.
- system variables (global variables).

Validity Note

This document has been updated for the release of EcoStruxure™ Machine Expert V1.1.

Related Documents

Title of Documentation	Reference Number
EcoStruxure Machine Expert - Programming Guide	*EIO0000002854 (ENG)*; *EIO0000002855 (FRE)*; *EIO0000002856 (GER)*; *EIO0000002858 (SPA)*; *EIO0000002857 (ITA)*; *EIO0000002859 (CHS)*
Modicon M262 Logic/Motion Controller - Hardware Guide	*EIO0000003659 (ENG)*; *EIO0000003660 (FRE)*; *EIO0000003661 (GER)*; *EIO0000003662 (SPA)*; *EIO0000003663 (ITA)*; *EIO0000003664 (CHS)*; *EIO0000003665 (POR)*; *EIO0000003666 (TUR)*

Title of Documentation	Reference Number
Modicon M262 Logic/Motion Controller - Programming Guide	EIO0000003651 (ENG); EIO0000003652 (FRE); EIO0000003653 (GER); EIO0000003654 (SPA); EIO0000003655 (ITA); EIO0000003656 (CHS); EIO0000003657 (POR); EIO0000003658 (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.

Terminology Derived from Standards

The technical terms, terminology, symbols and the corresponding descriptions in this manual, or that appear in or on the products themselves, are generally derived from the terms or definitions of international standards.

In the area of functional safety systems, drives and general automation, this may include, but is not limited to, terms such as *safety*, *safety function*, *safe state*, *fault*, *fault reset*, *malfunction*, *failure*, *error*, *error message*, *dangerous*, etc.

Among others, these standards include:

Standard	Description
IEC 61131-2:2007	Programmable controllers, part 2: Equipment requirements and tests.
ISO 13849-1:2015	Safety of machinery: Safety related parts of control systems. General principles for design.
EN 61496-1:2013	Safety of machinery: Electro-sensitive protective equipment. Part 1: General requirements and tests.
ISO 12100:2010	Safety of machinery - General principles for design - Risk assessment and risk reduction
EN 60204-1:2006	Safety of machinery - Electrical equipment of machines - Part 1: General requirements
ISO 14119:2013	Safety of machinery - Interlocking devices associated with guards - Principles for design and selection
ISO 13850:2015	Safety of machinery - Emergency stop - Principles for design
IEC 62061:2015	Safety of machinery - Functional safety of safety-related electrical, electronic, and electronic programmable control systems
IEC 61508-1:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: General requirements.
IEC 61508-2:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: Requirements for electrical/electronic/programmable electronic safety-related systems.
IEC 61508-3:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: Software requirements.
IEC 61784-3:2016	Industrial communication networks - Profiles - Part 3: Functional safety fieldbuses - General rules and profile definitions.
2006/42/EC	Machinery Directive
2014/30/EU	Electromagnetic Compatibility Directive
2014/35/EU	Low Voltage Directive

In addition, terms used in the present document may tangentially be used as they are derived from other standards such as:

Standard	Description
IEC 60034 series	Rotating electrical machines
IEC 61800 series	Adjustable speed electrical power drive systems
IEC 61158 series	Digital data communications for measurement and control – Fieldbus for use in industrial control systems

Finally, the term *zone of operation* may be used in conjunction with the description of specific hazards, and is defined as it is for a *hazard zone* or *danger zone* in the *Machinery Directive* (*2006/42/EC*) and *ISO 12100:2010*.

NOTE: The aforementioned standards may or may not apply to the specific products cited in the present documentation. For more information concerning the individual standards applicable to the products described herein, see the characteristics tables for those product references.

Chapter 1
Encoder Modes Principles

Overview

This chapter describes how to use an encoder in incremental mode or in SSI (Synchronous Serial Interface) mode.

What Is in This Chapter?

This chapter contains the following topics:

Incremental Mode Principle Description

Overview

This section describes the use of the incremental mode to connect incremental encoders.

Principle

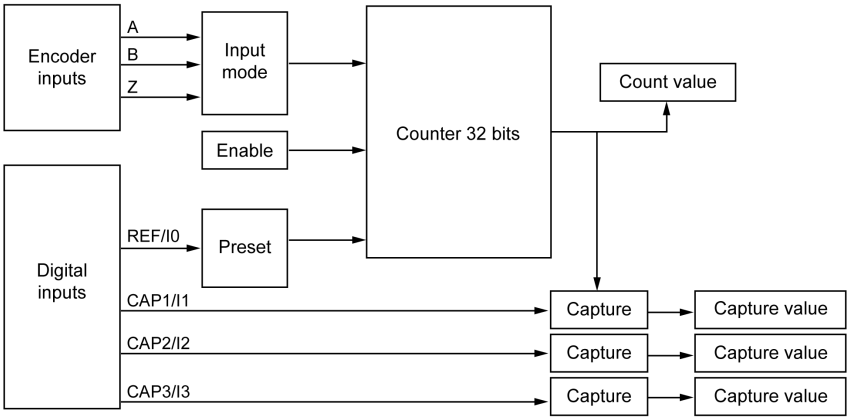
The incremental mode behaves like a standard up/down counter, using pulses and counting these pulses.

Positions must be preset and counting must be initialized to implement and manage the incremental mode.

The counter value can be stored in the capture register by configuring an external event.

Principle Diagram

The following diagram provides an overview of the encoder in incremental mode:



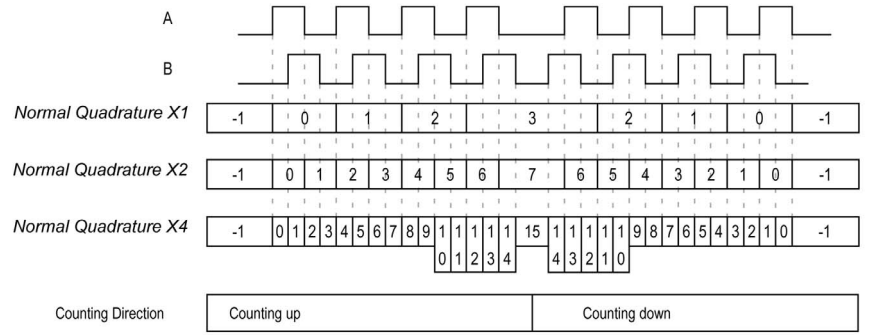
Axis Types

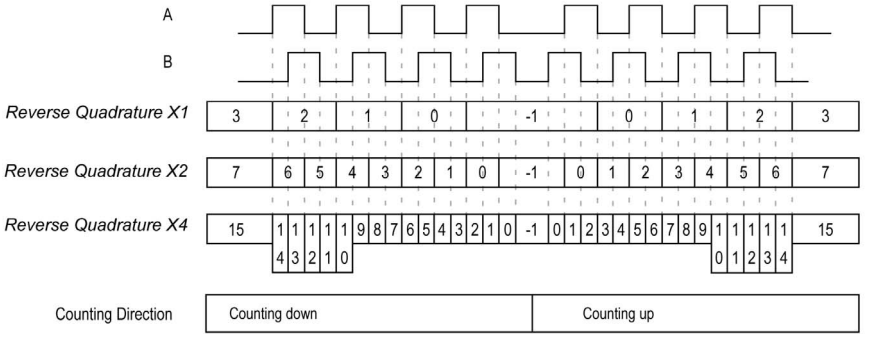
The following table presents the two available axis types and corresponding counting modes:

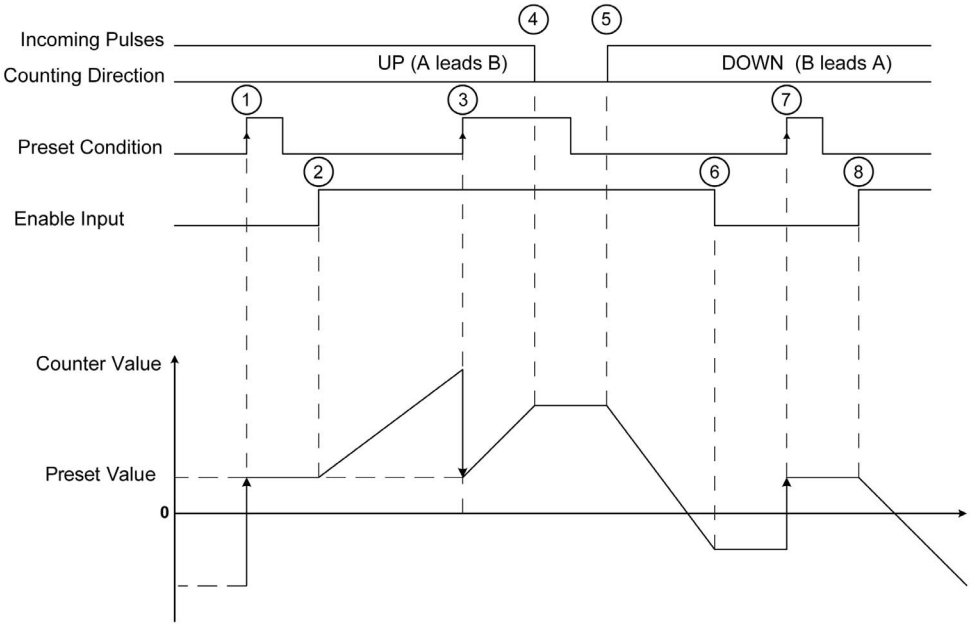
Axis Type	Comment
Linear	This mode acts as a finite counter.
Rotary	This mode acts as an infinite counter.

Principle Diagram

The input mode in incremental mode is always quadrature:







Stage	Action
1	On the rising edge of Preset condition, the counter value is set to the preset value and the counter is activated.
2	When the Enable condition = 1, the counter starts to increment when the counting direction is up.
3	The rising edge on the Preset condition loads the **Preset** value.
4	When the incoming pulses stop, the counter maintains its value.
5	When the Enable condition = 1, the counter starts to decrements when the counting direction is down.
6	When the Enable condition = 0, the counter ignores the pulses applied to the counting inputs A/B.
7	The rising edge on the Preset condition loads the preset value.
8	When the Enable condition = 1, the counter starts to decrements when the counting direction is down.

NOTE: Enable and Preset conditions depend on the configuration. These are described in the Enable *(see page 18)* and Preset *(see page 21)* function.

SSI Mode Principle Description

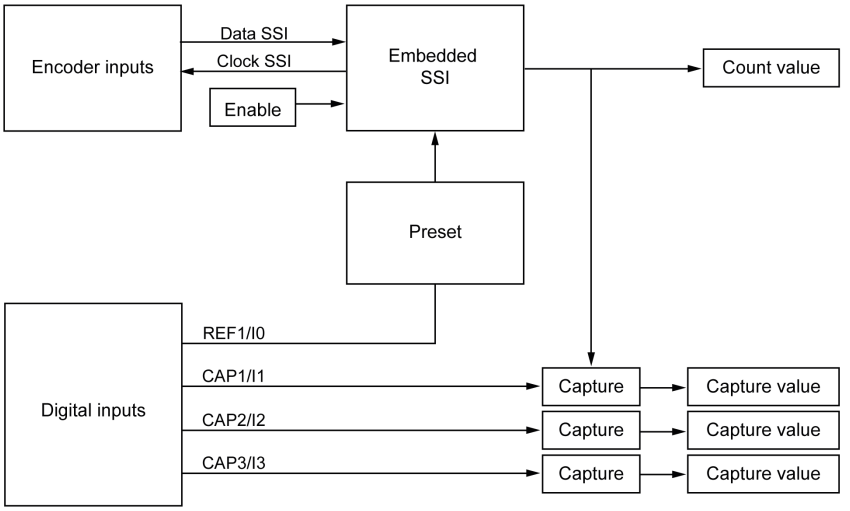
General

The SSI (Synchronous Serial Interface) mode allows the connection of an absolute encoder.

The position of the absolute encoder is read by an SSI link.

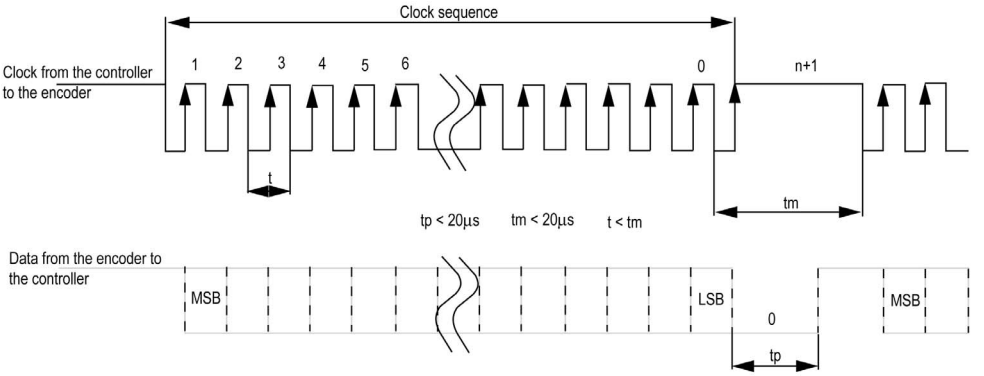
Principle Diagram

The following diagram provides an overview of the encoder in SSI mode:



Principle Diagram

The figure below represents an SSI frame:



Data Information

The data content can be configured to adjust the information from the absolute encoder:

Parameter	Range	Comment
Transmission speed	100 kHz, or 250 kHz, or 500 kHz	–
Number of bits per frame	8...64 bits	Length of frame = implicit number of header bits (0 to 4) + number of data bits (8 to 32) + number of status bits (0 to 4) + number of parity bit (0 or 1).
Number of data bits	8...32 bits	The least significant bits (8…32) indicate resolution per turn and the most significant bits (0…24) indicate the number of turns.
Number of data data/turn	8...16 bits	–
Number of status bits	0...4 bits	–
Parity	**None** **Odd** **Even**	–
Resolution reduction	0...17 bits	This parameter allows to filter data. The least significant bits are ignored.
Binary coding	**Binary** **Gray**	Binary or gray code.

Chapter 2
M262 Logic/Motion Controller Encoder Function Blocks

Overview

This chapter describes the function blocks included in the M262 Encoder Library. Adding an encoder adds automatically the Encoder Library to your controller.

What Is in This Chapter?

This chapter contains the following topics:

FB_Encoder_M262: Enable and Monitor the Encoder

Function Block Description

This function block is used to enable and monitor the encoder, in incremental or SSI mode.

You can only use one instance of this function block which is called once.

Use the cyclic calls to refresh the values.

Graphical Representation

IL and ST Representation

To see the general representation in IL or ST language, refer to the chapter *Function and Function Block Representation* *(see page 35)*.

I/O Variable Description

This table describes the input variables:

Input	Type	Default	Comments
ENC_REF_M262	ENC_REF_M262	–	Reference of the encoder instance.
xEnable	BOOL	FALSE	TRUE enables the function block. On a rising edge, the values of the following scaling parameters are taken into account: • udiScaling_NbOfIncs • udiScaling_NbOfUnits • udiScaling_IncPerTurn If you modify these values, trigger a rising edge on xEnable to take them into account.
udiScaling_NbOfIncs	UDINT	0	0 indicates that the scaling is disabled. The value in user units diCurrentValue_Unit equals the value in pulses diCurrentValue.
			> 0 indicates that the scaling is enabled. The value in user units is calculated from the value in pulses diCurrentValue, such as: diCurrentValue_Unit = diCurrentValue × (udiScalingNbOfUnits / udiScalingNbOfIncs).
udiScaling_NbOfUnits	UDINT	0	0 indicates that the scaling is disabled. The value in user units diCurrentValue_Unit equals the value in pulses diCurrentValue. If no scaling, then udiScalingNbOfUnits=udiScalingNbOfIncs.
			> 0 indicates that the scaling is enabled. The value in user units is calculated from the value in pulses diCurrentValue, such as: diCurrentValue_Unit = diCurrentValue × (udiScalingNbOfUnits / udiScalingNbOfIncs).
udiScaling_IncPerTurn	UDINT	0	When equal to 0, the axis type has a counter mode of linear. The counting range is: -2 147 483 648...2 147 483 647.
			If the number of increments is > 0, the axis type has a counter mode of rotary. udiSaling_IncPerTurn value defines the modulo value, at which the counter rolls over (the modulo value is never reached). The counting range is: 0...diScaling_IncPerTurn -1.

This table describes the output variables:

Output	Type	Default	Comment
xValid	BOOL	FALSE	TRUE indicates that the output values on the function block are valid. If the function block is disabled, the output is set to FALSE .
xError	BOOL	FALSE	TRUE indicates that an error is detected.
etErrorId	ET_ENC_ERROR_M262	ENC_ERROR_NO	Indicates the code of the detected error when xError is TRUE.
diNbTurns	DINT	0	Indicates the modulo value of the encoder. In incremental mode, it is incremented when the counter rolls over its up-limit. It is decremented when the counter rolls over its down-limit. In SSI mode diNbTurns = raw (SSI value - preset value) / udiScaling_IncPerTurn The raw SSI value comes directly from SSI, without any transformation.
diCurrentValue	DINT	0	In linear mode, indicates the value of the position of device in pulses. Range of value is: - 2 147 483 648...2 147 483 647. In rotary mode, indicates the value of the position in pulses for each turn of the mechanics. Range of value for diCurrentValue is 0...diScaling_IncPerTurn - 1.
lrCurrentValue_Unit	LREAL	0	Indicates the value of the encoder in units in turns of the mechanics. diCurrentValue_Unit = diCurrentValue / udiScaling_IncPerUnit when udiScaling_IncPerUnit ≥ 1.

FB_EncoderPreset_M262: Preset the Encoder

Function Block Description

This function block is used to preset the encoder, in incremental or SSI mode.

Graphical Representation

IL and ST Representation

To see the general representation in IL or ST language, refer to the chapter *Function and Function Block Representation* *(see page 35)*.

I/O Variable Description

This table describes the input variables:

Input	Type	Default	Comments
ENC_REF_M262	ENC_REF_M262	–	Reference of the encoder instance.
xEnable	BOOL	FALSE	TRUE enables the encoder preset function, via: • The preset mode using REF on I0 and Z on the encoder • The xForce input of the function block
xForce	BOOL	FALSE	On rising edge, presets and starts the counter if xEnable is TRUE.
etREF_Input	ET_ENC_INPUT_M262	ENC_INPUT_REF_I0	Defines the REF input. The only valid value is I0 *(see page 30)*.
etMode	ET_ENC_PRESET_MODE_M262	ENC_PRESET_NO	Selects the conditions to preset the counting function with REF and Z inputs *(see page 31)*.
diPresetValue	DINT	0	Defines the value loaded in the encoder actual value at preset event.

This table describes the output variables:

Output	Type	Default	Comment
xValid	BOOL	FALSE	TRUE indicates that the output values on the function block are valid.
xError	BOOL	FALSE	TRUE indicates that an error is detected.
etErrorId	ET_ENC_ERROR_M262	ENC_ERROR_NO	Indicates the code of the detected error when xError is TRUE *(see page 29)*.
xPresetFlag	BOOL	FALSE	Set to TRUE for one cycle by the preset of the encoder.

FB_EncoderCapture_M262: Capture the Encoder Value

Function Block Description

This function block is used to capture the encoder value, in incremental or SSI mode.

To configure several instances of this function block, define different `etCAP_Input`.

Graphical Representation

IL and ST Representation

To see the general representation in IL or ST language, refer to the chapter *Function and Function Block Representation* *(see page 35)*.

I/O Variable Description

This table describes the input variables:

Input	Type	Default	Comment
`ENC_REF_M262`	`ENC_REF_M262`	–	Reference of the encoder instance.
`xEnable`	`BOOL`	`FALSE`	`TRUE` enables the encoder capture function, via the capture input specified by the `etCAP_Input` input.
`etCAP_Input`	`ET_ENC_INPUT_M262`	`ENC_INPUT_CAP_I1`	Defines the input used for the capture function *(see page 30)*.
`etCAP_Edge`	`ET_ENC_CAP_EDGE_M262`	`ENC_CAP_EDGE_RISING`	Indicates the edge detection for capture input *(see page 28)*.

This table describes the output variables:

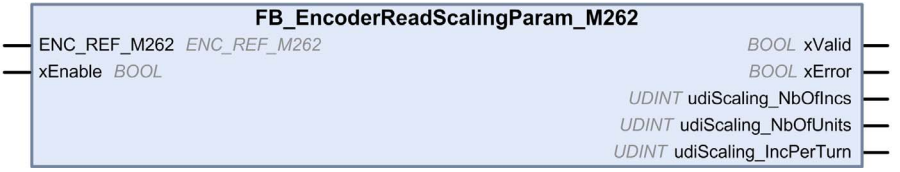
Output	Type	Default	Comment
xValid	BOOL	FALSE	TRUE indicates that the output values on the function block are valid.
xError	BOOL	FALSE	TRUE indicates that an error is detected.
etErrorId	ET_ENC_ERROR_M262	ENC_ERROR_NO	Indicates the code of the detected error when xError is TRUE *(see page 29)*.
xCaptureFlag	BOOL	FALSE	TRUE indicates that a cycle is defined by the encoder capture event. xCaptureFlag is therefore TRUE for only one cycle.
diCapturedValue	DINT	0	Indicates the captured value in pulses, valid at xCaptureFlag rising edge. Captured value remains until next xCaptureFlag occurs. Captured value is reset to 0 when xEnable set to FALSE.
lrCapturedValue_Units	LREAL	0.0	Indicates the captured value in units, valid at xCaptureFlag rising edge. Captured value remains until next xCaptureFlag occurs. Captured value is reset to 0 when xEnable set to FALSE.

FB_EncoderReadScalingParam_M262: Read the Scaling Parameter

Function Block Description

This function block is used to read the active values of the scaling parameter used to compute the unit value, in incremental or SSI mode.

Graphical Representation



IL and ST Representation

To see the general representation in IL or ST language, refer to the chapter *Function and Function Block Representation* *(see page 35)*.

I/O Variable Description

This table describes the input variables:

Input	Type	Default	Comment
ENC_REF_M262	ENC_REF_M262	–	Reference of the encoder instance.
xEnable	BOOL	FALSE	TRUE enables the encoder function block reading active values of the scaling parameter used to compute lrCurrentValue_Unit. FALSE disables the function block.

This table describes the output variables:

Output	Type	Default	Comment
xValid	BOOL	FALSE	TRUE indicates that the output values on the function block are valid.
xError	BOOL	FALSE	TRUE indicates that an error is detected.
udiScalingNbOfIncs	UDINT	0	Indicates the active value of udiScalingNbOfIncs to compute lrCurrentValue_Unit.
udiScalingNbOfUnits	UDINT	0	Indicates the active value of udiScalingNbOfUnits to compute lrCurrentValue_Unit.
udiScaling_IncPerTurn	UDINT	0	Indicates the active value of udiScaling_IncPerTurn to compute lrCurrentValue_Unit.

Chapter 3
M262 Logic/Motion Controller Library Data Types

Overview

This chapter describes the data types of the M262 encoder library.

What Is in This Chapter?

This chapter contains the following topics:

ET_ENC_CAP_EDGE_M262: Encoder Capture Codes

Enumerated Type Description

This enumeration describes the types of edges which can be used for reference and capture on an encoder function block.

The ET_ENC_CAP_EDGE_M262 enumeration data type contains the following values:

Parameter Name	Value	Description
ENC_CAP_EDGE_RISING	0	Capture on input rising edge.
ENC_CAP_EDGE_FALLING	1	Capture on input falling edge.
ENC_CAP_EDGE_BOTH	2	Capture on input both edges.

ET_ENC_ERROR_M262: Encoder Error Codes

Enumerated Type Description

This enumeration describes the types of errors which can occur on an encoder function block.

The ET_ENC_ERROR_M262 enumeration data type contains the following values:

Parameter Name	Value	Description
ENC_ERROR_NO	0	No error detected.
ENC_ERROR_REF	1	The encoder reference is incorrect or not configured.
ENC_ERROR_PARAMETER_INVALID	3	The value of a parameter is incorrect.
ENC_ERROR_COM	4	A communication error is detected with the encoder.
ENC_ERROR_SUPPLY	11	Encoder supply not detected.
ENC_ERROR_I0_EVT_CONFIGURED	12	I0 is configured as an event and cannot be used for preset.
ENC_ERROR_RESERVED	13	FB_Encoder_M262 function block is reserved.

ET_ENC_INPUT_M262: Encoder Input Codes

Enumerated Type Description

This enumeration describes the types of inputs which can be used for reference and capture on an encoder function block.

The ET_ENC_INPUT_M262 enumeration data type contains the following values:

Parameter Name	Value	Description
ENC_INPUT_REF_I0	0	REF input on I0 for preset.
ENC_INPUT_CAP_I1	1	Capture input on I1.
ENC_INPUT_CAP_I2	2	Capture input on I2.
ENC_INPUT_CAP_I3	3	Capture input on I3.

ET_ENC_PRESET_MODE_M262: Encoder Preset Mode Codes

Enumerated Type Description

This enumeration describes the different types of preset mode which can be used for an encoder function block.

The ET_ENC_PRESET_MODE_M262 enumeration data type contains the following values:

Parameter Name	Value	Description
ENC_PRESET_NO	0	No preset configured.
ENC_PRESET_Z_EDGE_RISING	1	Preset on Z rising edge (incremental encoder only).
ENC_PRESET_Z_EDGE_FALLING	2	Preset on Z falling edge (incremental encoder only).
ENC_PRESET_Z_EDGE_BOTH	3	Preset on Z both edges (incremental encoder only).
ENC_PRESET_REF_RISING	4	Preset on REF rising edge.
ENC_PRESET_REF_FALLING	5	Preset on REF falling edge.
ENC_PRESET_REF_BOTH	6	Preset on REF both edges.
ENC_PRESET_Z_EDGE_RISING_AND_REF	7	Preset on Z rising edge and REF (incremental encoder only).
ENC_PRESET_EDGE_RISING_Z_FIRST_AND_REF	8	Preset on the first Z rising edge and REF (incremental encoder only).
ENC_PRESET_EDGE_RISING_Z_FIRST_AND_NO_REF	9	Preset on the first Z rising edge and no REF (incremental encoder only).

Appendices

Appendix A
Function and Function Block Representation

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 Is 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.
- needs to be called by 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 from a function block or a program.

Examples: timers, counters

In the example, `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.

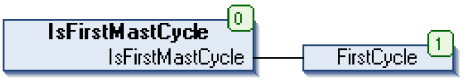
Functions `IsFirstMastCycle` and `SetRTCDrift` and Function Block `TON` are used as examples to show implementations.

Using a Function in IL Language

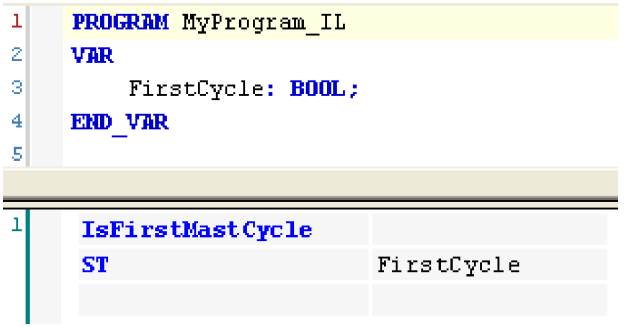
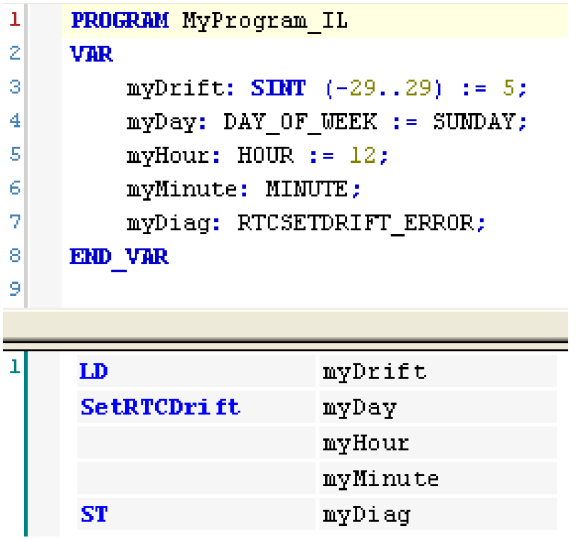
This procedure describes how to insert a function in IL language:

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 Adding and Calling POUs.
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 the 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 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

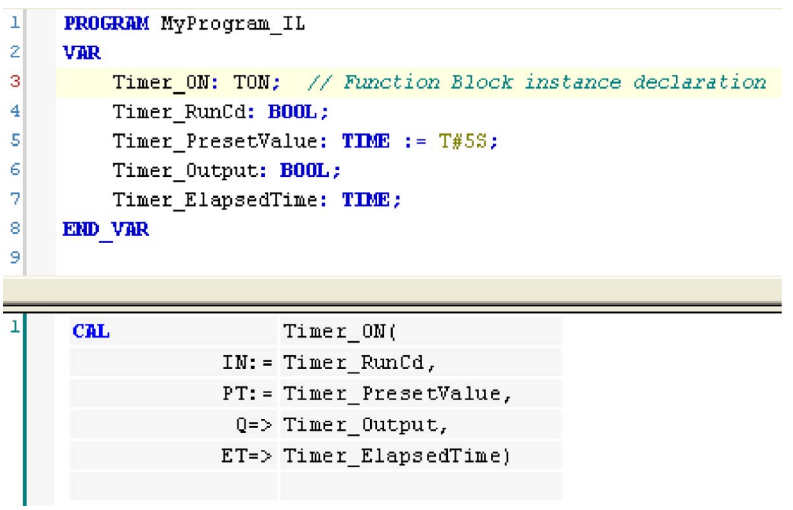
This procedure describes how to insert a function block in IL language:

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 Adding and Calling POUs.
2	Create the variables that the function block requires, including the instance name.
3	Function Blocks are called using a `CAL` instruction: ● Use the **Input Assistant** to select the FB (right-click and select **Insert Box** in the context menu). ● Automatically, the `CAL` instruction and the necessary I/O are created. Each parameter (I/O) is an instruction: ● Values 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 POU IL Editor
TON	

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

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

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

This 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 Adding and Calling POUs.
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 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

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

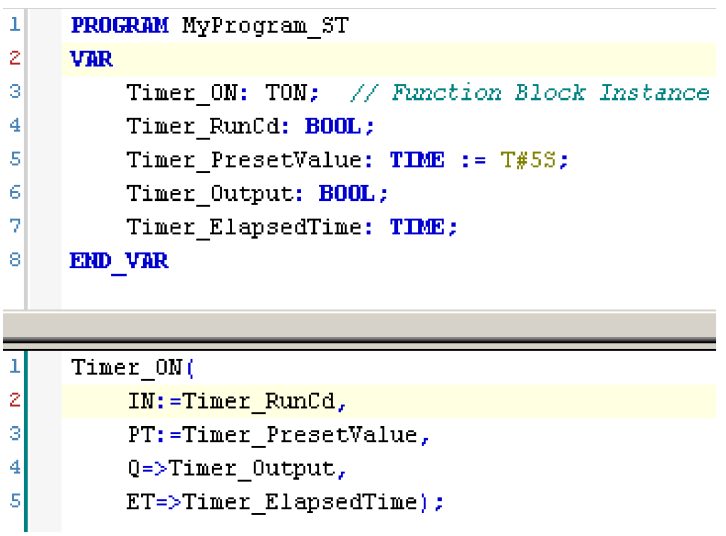
Step	Action
1	Open or create a new POU in Structured Text language. **NOTE:** The procedure to create a POU is not detailed here. For more information on adding, declaring and calling POUs, refer to the related documentation.
2	Create the input and output variables and the instance required for the function 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`	

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

Function Block	Representation in POU ST Editor
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);
```

Glossary

B

byte

A type that is encoded in an 8-bit format, ranging from 00 hex to FF hex.

C

CFC

(*continuous function chart*) A graphical programming language (an extension of the IEC 61131-3 standard) based on the function block diagram language that works like a flowchart. However, no networks are used and free positioning of graphic elements is possible, which allows feedback loops. For each block, the inputs are on the left and the outputs on the right. You can link the block outputs to the inputs of other blocks to create complex expressions.

F

FB

(*function block*) A convenient programming mechanism that consolidates a group of programming instructions to perform a specific and normalized action, such as speed control, interval control, or counting. A function block may comprise configuration data, a set of internal or external operating parameters and usually 1 or more data inputs and outputs.

function block diagram

One of the 5 languages for logic or control supported by the standard IEC 61131-3 for control systems. Function block diagram is a graphically oriented programming language. It works with a list of networks where each network contains a graphical structure of boxes and connection lines representing either a logical or arithmetic expression, the call of a function block, a jump, or a return instruction.

I

IL

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

INT

(*integer*) A whole number encoded in 16 bits.

L

LD

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

P

POU

(*program organization unit*) A variable declaration in source code and a corresponding instruction set. POUs facilitate the modular re-use of software programs, functions, and function blocks. Once declared, POUs are available to one another.

S

ST

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

V

variable

A memory unit that is addressed and modified by a program.

Index