Modicon M241 Logic Controller

PTOPWM

Library Guide

11/2015

EIO0000001450.05

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.

No part of this document may be reproduced in any form or by any means, electronic or mechanical, including photocopying, without express written permission of Schneider Electric.

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.

⚠ CAUTION

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 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 documentation acquaints you with the pulse train output (PTO), pulse width modulation (PWM) and frequency generator (FG) functions offered within the Modicon M241 Logic Controller.

This document describes the data types and functions of the M241 PTOPWM Library.

In order to use this manual, you must:

- Have a thorough understanding of the M241, including its design, functionality, and implementation within control systems.
- Be proficient in the use of the following IEC 61131-3 PLC programming languages:
  - Function Block Diagram (FBD)
  - Ladder Diagram (LD)
  - Structured Text (ST)
  - Instruction List (IL)
  - Sequential Function Chart (SFC)

Validity Note

This document has been updated for the release of SoMachine V4.1 SP2.

Related Documents

Title of Documentation	Reference Number
Modicon M241 Logic Controller Programming Guide	EIO0000001432 (Eng), EIO0000001433 (Fre), EIO0000001434 (Ger), EIO0000001435 (Spa), EIO0000001436 (Ita), EIO0000001437 (Chs)

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

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
EN 61131-2:2007	Programmable controllers, part 2: Equipment requirements and tests.
ISO 13849-1:2008	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
EN 1088:2008 ISO 14119:2013	Safety of machinery - Interlocking devices associated with guards - Principles for design and selection
ISO 13850:2006	Safety of machinery - Emergency stop - Principles for design
EN/IEC 62061:2005	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 61784-3:2008	Digital data communication for measurement and control: Functional safety field buses.
2006/42/EC	Machinery Directive
2004/108/EC	Electromagnetic Compatibility Directive
2006/95/EC	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 *EC Machinery Directive* (*EC/2006/42*) 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.

Part I
Introduction

Overview

This part provides an overview description, available modes, functionality and performances of the different functions.

What Is in This Part?

This part contains the following chapters:

Chapter 1
Overview

Overview

This chapter provides an overview description, functionality, and performances of:

- `Frequency Generator` (`FG`)
- `Pulse Width Modulation` (`PWM`)
- `Pulse Train Output` (`PTO`)

What Is in This Chapter?

This chapter contains the following topics:

Expert I/O Overview

Introduction

The M241 logic controller provides:

I/O Type	24 I/O References	40 I/O References
	TM241•24•	TM241•40•
Fast inputs	8	8
Regular inputs	6	16
Fast outputs	4	4
Regular outputs	6	12

The M241 logic controller supports the following expert functions:

Functions		Description
Counters	HSC Simple	The HSC functions can execute fast counts of pulses from sensors, switches, and so on, that are connected to the fast inputs. For more information about the HSC functions, refer to the High Speed Counter types *(see Modicon M241 Logic Controller, High Speed Counting, HSC Library Guide)*.
	HSC Main Single Phase	
	HSC Main Dual Phase	
	Frequency Meter	
	Period Meter	
Pulse Generators	PTO *(see page 29)*	The PTO function generates a pulse train output to control a linear single-axis stepper or servo drive in open loop mode.
	PWM *(see page 161)*	The PWM function generates a square wave signal on dedicated output channels with a variable duty cycle.
	Frequency Generator *(see page 179)*	The frequency generator function generates a square wave signal on dedicated output channels with a fixed duty cycle (50%).

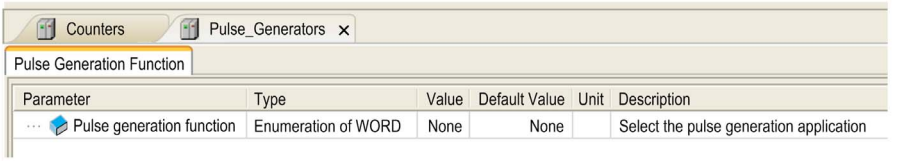
NOTE:

- When an input is used as Run/Stop, it cannot be used by an expert function.
- When an output is used as Alarm, it cannot be used by an expert function.

For more details, refer to Embedded Functions Configuration *(see Modicon M241 Logic Controller, Programming Guide)*.

Configuring an Expert Function

To configure an expert function, proceed as follow:

Step	Description
1	Double-click the **Counters** or **Pulse_Generators** node in the **Devices Tree**. **Result:** The **Counters** or **Pulse_Generators** function window appears: 
2	Double-click **Value** and choose the function type to assign. **Result:** The parameters of the expert function appear.

Expert Function I/O Within Regular I/O

Expert function I/O within regular I/O:

- Inputs can be read through standard memory variables even if configured as expert functions.
- An input cannot be configured as an expert function if it has already been configured as a Run/Stop input.
- An output cannot be configured in an expert function if it has already been configured as an alarm.
- Short-Circuit management still applies on all outputs. Status of outputs are available.
- All I/O that are not used by expert functions can be used as any other I/O.

When inputs are used in expert functions (Latch, HSC,…), integrator filter is replaced by anti-bounce filter. Filter value is configured in expert configuration screen.

Embedded Expert I/O Mapping

Input Mapping for Expert Functions on M241

M241 Expert Inputs		Fast Inputs								Regular Inputs					
		I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13
Event Latch 0	Input	M													
Event Latch 1	Input		M												
Event Latch 2	Input			M											
Event Latch 3	Input				M										
Event Latch 4	Input					M									
Event Latch 5	Input						M								
Event Latch 6	Input							M							
Event Latch 7	Input								M						
HSC Simple 0	Input	M													
HSC Simple 1	Input		M												
HSC Simple 2	Input			M											
HSC Simple 3	Input				M										
HSC Simple 4	Input					M									
HSC Simple 5	Input						M								
HSC Simple 6	Input							M							
HSC Simple 7	Input								M						
HSC Main 0	Input A	M													
	Input B/EN		C												
	SYNC			C											
	CAP				C							C			
	Reflex 0														
	Reflex 1														
HSC Main 1	Input A					M									
	Input B/EN						C								
	SYNC							C							
	CAP								C						C
	Reflex 0														
	Reflex 1														
M Mandatory **C** Depend on Configuration															

M241 Expert Inputs		Fast Inputs								Regular Inputs					
		I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13
Frequency Meter 0/Period Meter 0	Signal	M													
	EN		C												
Frequency Meter 1/Period Meter 1	Signal					M									
	EN						C								
PWM 0/Frequency Generator 0	Outputs														
	EN				C							C			
	SYNC										C				
PWM 1/Frequency Generator 1	Outputs														
	EN								C						C
	SYNC													C	
PTO 0	REF (Origin)									C					
	INDEX (Proximity)										C				
	PROBE				C							C			
	Output A/CW/Pulse														
	Output B/CCW/Dir														
PTO 1	REF (Origin)												C		
	INDEX (Proximity)													C	
	PROBE							C							C
	Output A/CW/Pulse														
	Output B/CCW/Dir														

M Mandatory
C Depend on Configuration

Output Mapping for Expert Functions on M241

M241 Expert Outputs		Fast Outputs				Regular Outputs					
		Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9
HSC Main 0	Input A										
	Input B/EN										
	SYNC										
	CAP										
	Reflex 0	C				C					
	Reflex 1		C				C				
HSC Main 1	Input A										
	Input B/EN										
	SYNC										
	CAP										
	Reflex 0			C				C			
	Reflex 1				C				C		
Frequency Meter 0/Period Meter 0	Signal										
	EN										
Frequency Meter 1/Period Meter 1	Signal										
	EN										
PWM 0/Freq Gen 0	Outputs	M				C					
	EN										
	SYNC										
PWM 1/Freq Gen 1	Outputs			M				C			
	EN										
	SYNC										

M Mandatory
C Depend on Configuration

NOTE: For more information concerning using regular outputs as Pulse Generators, refer to Regular Transistor Outputs *(see Modicon M241 Logic Controller, Hardware Guide)*.

M241 Expert Outputs		Fast Outputs				Regular Outputs					
		Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9
PTO 0	REF (Origin)										
	INDEX (Proximity)										
	PROBE										
	Output A/CW/Pulse	M				C					
	Output B/CCW/Dir		M				C				
PTO 1	REF (Origin)										
	INDEX (Proximity)										
	PROBE										
	Output A/CW/Pulse			M				C			
	Output B/CCW/Dir				M				C		

M Mandatory
C Depend on Configuration

NOTE: For more information concerning using regular outputs as Pulse Generators, refer to Regular Transistor Outputs *(see Modicon M241 Logic Controller, Hardware Guide)*.

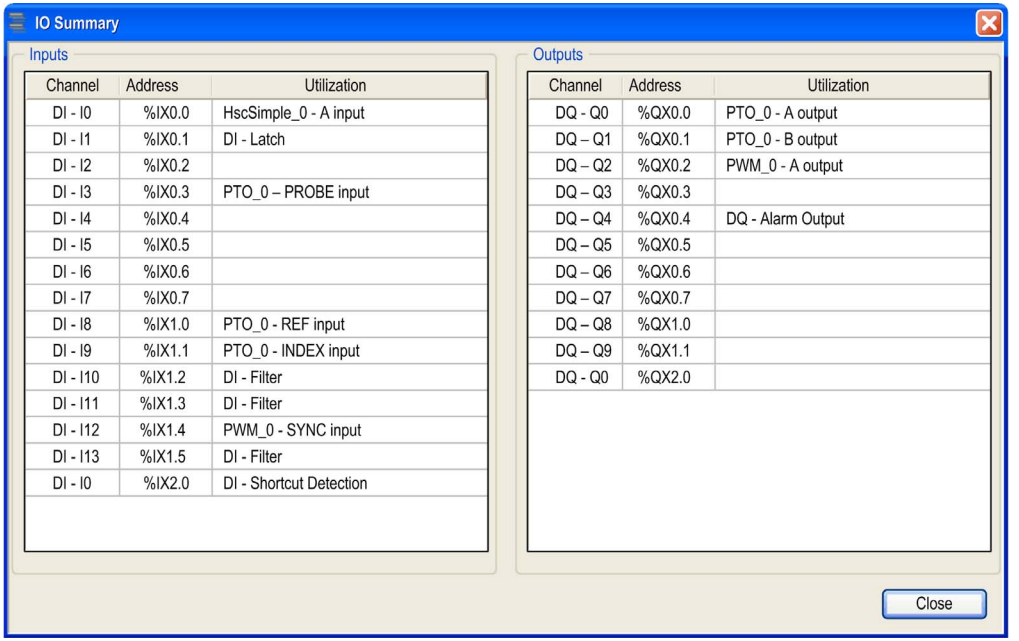
I/O Summary

The **IO Summary** window displays the I/Os used by the expert functions.

To display the **IO Summary** window:

Step	Action
1	In the **Devices tree** tab, right-click the **MyController** node and choose **IO Summary**.

Example of IO Summary window:



Chapter 2
Generalities

Overview

This chapter provides general information of the Frequency Generator (FG), Pulse Train Output (PTO), and Pulse Width Modulation (PWM) functions.

The functions provide simple, yet powerful solutions for your application. In particular, they are extremely useful for controlling movement. However, the use and application of the information contained herein require expertise in the design and programming of automated control systems. Only you, the user, machine builder or integrator, can be aware of all the conditions and factors present during installation and setup, operation, and maintenance of the machine or related processes, and can therefore determine the automation and associated equipment and the related safeties and interlocks which can be effectively and properly used. When selecting automation and control equipment, and any other related equipment or software, for a particular application, you must also consider any applicable local, regional or national standards and/or regulations.

⚠ WARNING

REGULATORY INCOMPATIBILITY

Ensure that all equipment applied and systems designed comply with all applicable local, regional, and national regulations and standards.

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

The functions provided by the M241 PTOPWM library were conceived and designed assuming that you incorporate the necessary safety hardware into your application architecture, such as, but not limited to, appropriate limit switches and emergency stop hardware and controlling circuitry. It is implicitly assumed that functional safety measures are present in your machine design to prevent undesirable machine behavior such as over-travel or other forms of uncontrolled movement. Further, it is assumed that you have performed a functional safety analysis and risk assessment appropriate to your machine or process.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

Ensure that a risk assessment is conducted and respected according to EN/ISO 12100 during the design of your machine.

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

What Is in This Chapter?

This chapter contains the following topics:

Dedicated Features

Bounce Filter

This table shows the maximum counter frequencies determined by the filtering values used to reduce the bounce effect on the input:

Input	Bounce Filter Value (ms)	Maximum Counter Frequency
A B EN CAP	0.000	200 kHz
	0.001	200 kHz
	0.002	200 kHz
	0.005	100 kHz
	0.010	50 kHz
	0.05	25 kHz
	0.1	5 kHz
	0.5	1 kHz
	1	500 Hz
	5	100 Hz
A is the counting input of the counter. B is the counting input of the dual phase counter. EN is the enable input of the counter. CAP is the capture input of the counter.		

Dedicated Outputs

Outputs used by the Frequency Generator, Pulse Train Output, Pulse Width Modulation, and High Speed Counters can only be accessed through the function block. They can not be read or written directly within the application.

⚠ WARNING

UNINTENTED EQUIPMENT OPERATION

- Do not use the same function block instance in different program tasks.
- Do not change the function block reference (AXIS) while the function block is executing.

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

General Information on Function Block Management

Management of Input Variables

The variables are used with the rising edge of the `Execute` input. To modify any variable, it is necessary to change the input variables and to trigger the function block again.

The function blocks managed by an `Enable` input are executed when this input is true. The values of the function block inputs can be modified continuously, and the outputs are updated continuously. When the `Enable` input is false, the function block execution is terminated and its outputs are reseted.

According to IEC 61131-3, if any variable of a function block input is missing (= open), then the value from the previous invocation of this instance will be used. In the first invocation the initial value is applied.

Management of Output Variables

The `Done`, `Error`, `Busy`, and `CommandAborted` outputs are mutually exclusive; only one of them can be TRUE on one function block. When the `Execute` input is TRUE, one of these outputs is TRUE.

At the rising edge of the `Execute` input, the `Busy` output is set. It remains set during the execution of the function block and is reset at the rising edge of one of the other outputs (`Done`, `Error`).

The `Done` output is set when the execution of the function block is successfully completed.

If an error is detected, the function block terminates by setting the `Error` output, and the error code is contained within the `ErrId` output.

The `Done`, `Error`, `ErrID`, and `CommandAborted` outputs are set or reset with the falling edge of `Execute` input:

- reset if the function block execution is finished.
- set for at least one task cycle if the function block execution is not finished.

When an instance of a function block receives a new `Execute` before it is finished (as a series of commands on the same instance), the function block does not return any feedback, like `Done`, for the previous action.

Error Handling

All blocks have two outputs that can report error detection during the execution of the function block:

- `Error`= The rising edge of this bit informs that an error was detected.
- `ErrID`= The error code of the error detected.

When an `Error` occurs, the other output signals, such as `Done` are reset.

Part II
Pulse Train Output (PTO)

Overview

This part describes the `Pulse Train Output` function.

What Is in This Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
3	Overview	31
4	Configuration	35
5	Data Unit Types	71
6	Motion Function Blocks	81
7	Administrative Function Blocks	135

Chapter 3
Overview

Pulse Train Output (PTO)

Introduction

The PTO function provides two pulse train output channels for a specified number of pulses and a specified velocity (frequency). The PTO function is used to control the positioning or speed of two independent linear single-axis stepper or servo drives in open loop mode (for example, with Lexium 23D).

The PTO function does not have any position feedback information from the process. Therefore, position information must be integrated in the drive.

The PTO, PWM (pulse width modulation), and FG (frequency generation) functions use the same dedicated outputs. Only one of these three functions can be used on the same channel. Using different functions on channel 0 and channel 1 is allowed.

A PTO channel can use up to:
- six physical inputs, if optional interface signals for homing (ref/index), event (probe), limits (limP, limN), or drive interface (driveReady), are used,
- three physical outputs, if optional drive interface signal is used (driveEnable).

Automatic origin offset and backlash compensation are also managed to improve positioning accuracy. Diagnostics are available for status monitoring, providing comprehensive and quick troubleshooting.

Supported Functions

The two PTO channels support the following functions:
- three output modes, including quadrature
- single axis moves (velocity and position)
- relative and absolute positioning
- automatic trapezoidal and S-curve acceleration and deceleration
- homing (seven modes with offset compensation)
- dynamic acceleration, deceleration, velocity, and position modification
- switch from speed to position mode
- move queuing (buffer of one move)
- position capture and move trigger on event (using probe input)
- backlash compensation (in quadrature mode)
- limits (hardware and software)
- diagnostics

PTO Function Blocks

The PTO function is programmed in SoMachine using the following function blocks, available in the **M241 PTOPWM** library:

Category	Subcategory	Function Block
Motion (single axis)	Power	MC_Power_PTO *(see page 93)*
	Discrete	MC_MoveAbsolute_PTO *(see page 109)*
		MC_MoveRelative_PTO *(see page 103)*
		MC_Halt_PTO *(see page 128)*
		MC_SetPosition_PTO *(see page 120)*
	Continuous	MC_MoveVelocity_PTO *(see page 97)*
	Homing	MC_Home_PTO *(see page 115)*
	Stopping	MC_Stop_PTO *(see page 123)*
Administrative	Status	MC_ReadActualVelocity_PTO *(see page 137)*
		MC_ReadActualPosition_PTO *(see page 138)*
		MC_ReadStatus_PTO *(see page 139)*
		MC_ReadMotionState_PTO *(see page 141)*
	Parameters	MC_ReadParameter_PTO *(see page 144)*
		MC_WriteParameter_PTO *(see page 146)*
		MC_ReadBoolParameter_PTO *(see page 148)*
		MC_WriteBoolParameter_PTO *(see page 150)*
	Probe	MC_TouchProbe_PTO *(see page 153)*
		MC_AbortTrigger_PTO *(see page 155)*
	Error handling	MC_ReadAxisError_PTO *(see page 157)*
		MC_Reset_PTO *(see page 159)*

NOTE: The motion function blocks act on the position of the axis according to the motion state diagram *(see page 83)*. The administrative function blocks do not influence the motion state.

NOTE: MC_Power_PTO function block is mandatory before a move command can be issued.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

- Do not use the same function block instance in different program tasks.
- Do not change the function block reference (AXIS) while the function block is executing.

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

PTO Characteristics

The PTO function has the following characteristics:

Characteristic	Value
Number of channels	2
Number of axis	1 per channel
Position range	-2,147,483,648...2,147,483,647 (32 bits)
Minimum velocity	1 Hz
Maximum velocity	100 kHz (for a 40/60 duty cycle and max. 200 mA)
Minimum step	1 Hz
Acceleration / deceleration min	1 Hz/ms
Acceleration / deceleration max	100,000 Hz/ms
Start move IEC	300 µs + 1 pulse output time
Start move on probe event	
Change move parameter	
Accuracy on velocity	0.5 %
Accuracy in position	Depends on the pulse output time

Chapter 4
Configuration

Overview

This chapter describes how to configure a PTO channel and the associated parameters.

What Is in This Chapter?

This chapter contains the following sections:

Section 4.1
Configuration

Overview

This section describes how to configure a PTO channel and the associated parameters.

What Is in This Section?

This section contains the following topics:

PTO Configuration

Hardware Configuration

There are up to six physical inputs for a PTO channel:

- Three are associated to the PTO function through configuration and are taken into account immediately on a rising edge on the input:
  - REF input
  - INDEX input
  - PROBE input
- Three are associated with the `MC_Power_PTO` function block. They have no fixed assignment (they are freely assigned; that is, they are not configured in the configuration screen), and are read as any other input:
  - Drive ready input
  - Limit positive input
  - Limit negative input

NOTE: These inputs are managed as any other regular input, but are used by the PTO controller when used by `MC_Power_PTO` function block.

NOTE: The positive and negative limit inputs are required to help prevent over-travel.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

- Ensure that controller hardware limit switches are integrated in the design and logic of your application.
- Mount the controller hardware limit switches in a position that allows for an adequate braking distance.

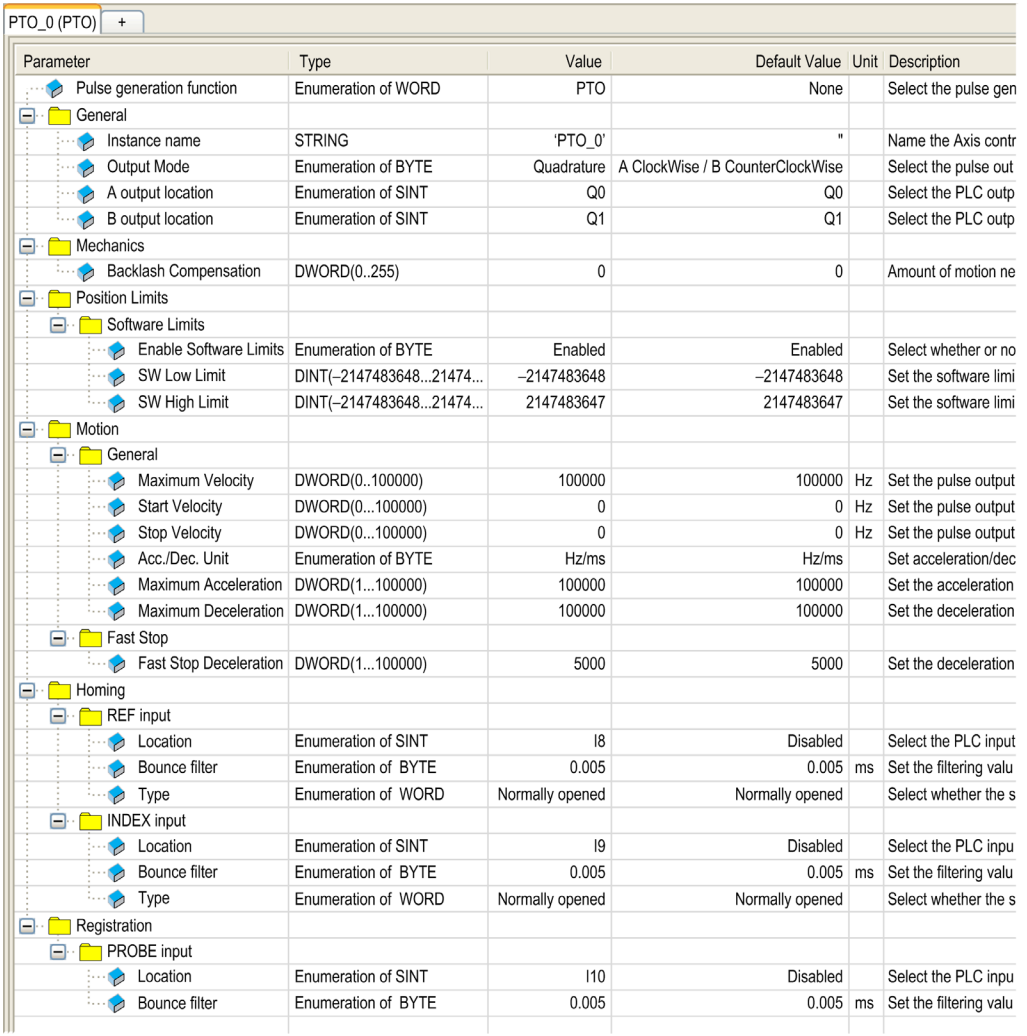
Failure to follow these instructions can result in death, serious injury, or equipment damage.

There are up to three physical outputs for a PTO channel:

- Two outputs are mandatory to manage pulse and direction; they have a fixed assignment and must be enabled by configuration:
  - A / CW / Pulse
  - B / CCW / Direction
- The other output, `DriveEnable`, is used through `MC_Power_PTO` function block; it has no fixed assignment (it is freely assigned), and is written as any other output.

Configuration Window Description

The figure provides an example of a configuration window on channel **PTO_0**:



PTO_0 (PTO) +

Parameter	Type	Value	Default Value	Unit	Description
Pulse generation function	Enumeration of WORD	PTO	None		Select the pulse gen
General					
Instance name	STRING	'PTO_0'	''		Name the Axis contr
Output Mode	Enumeration of BYTE	Quadrature	A ClockWise / B CounterClockWise		Select the pulse out
A output location	Enumeration of SINT	Q0	Q0		Select the PLC outp
B output location	Enumeration of SINT	Q1	Q1		Select the PLC outp
Mechanics					
Backlash Compensation	DWORD(0..255)	0	0		Amount of motion ne
Position Limits					
Software Limits					
Enable Software Limits	Enumeration of BYTE	Enabled	Enabled		Select whether or no
SW Low Limit	DINT(–2147483648...21474...	–2147483648	–2147483648		Set the software limi
SW High Limit	DINT(–2147483648...21474...	2147483647	2147483647		Set the software limi
Motion					
General					
Maximum Velocity	DWORD(0..100000)	100000	100000	Hz	Set the pulse output
Start Velocity	DWORD(0...100000)	0	0	Hz	Set the pulse output
Stop Velocity	DWORD(0...100000)	0	0	Hz	Set the pulse output
Acc./Dec. Unit	Enumeration of BYTE	Hz/ms	Hz/ms		Set acceleration/dec
Maximum Acceleration	DWORD(1...100000)	100000	100000		Set the acceleration
Maximum Deceleration	DWORD(1...100000)	100000	100000		Set the deceleration
Fast Stop					
Fast Stop Deceleration	DWORD(1...100000)	5000	5000		Set the deceleration
Homing					
REF input					
Location	Enumeration of SINT	I8	Disabled		Select the PLC input
Bounce filter	Enumeration of BYTE	0.005	0.005	ms	Set the filtering valu
Type	Enumeration of WORD	Normally opened	Normally opened		Select whether the s
INDEX input					
Location	Enumeration of SINT	I9	Disabled		Select the PLC inpu
Bounce filter	Enumeration of BYTE	0.005	0.005	ms	Set the filtering valu
Type	Enumeration of WORD	Normally opened	Normally opened		Select whether the s
Registration					
PROBE input					
Location	Enumeration of SINT	I10	Disabled		Select the PLC inpu
Bounce filter	Enumeration of BYTE	0.005	0.005	ms	Set the filtering valu

The table describes each parameter available when the channel is configured in **PTO** mode:

Parameter		Value	Default	Description
General	Instance name	-	PTO_0 or PTO_1	Name of the axis controlled by this PTO channel. It is used as input of the PTO function blocks.
	Output Mode *(see page 42)*	A ClockWise / B CounterClockWise A Pulse / B Direction Quadrature	A ClockWise / B CounterClockWise	Select the pulse output mode.
	A output location	Q0 or Q4 (channel 0) Q2 or Q6 (channel 1)	Q0 (channel 0) Q2 (channel 1)	Select the controller output used for the signal A.
	B output location	Q1 or Q5 (channel 0) Q3 or Q7 (channel 1)	Q1 (channel 0) Q3 (channel 1)	Select the controller output used for the signal B.
Mechanics	Backlash Compensation *(see page 49)*	0...255	0	In quadrature mode, amount of motion needed to compensate the mechanical clearance when movement is reversed.
Position Limits / Software Limits	Enable Software Limits *(see page 52)*	Enabled Disabled	Enabled	Select whether to use the software limits.
	SW Low Limit	-2,147,483,648... 2,147,483,647	-2,147,483,648	Set the software limit position to be detected in the negative direction.
	SW High Limit	-2,147,483,648... 2,147,483,647	2,147,483,647	Set the software limit position to be detected in the positive direction.
Motion / General	Maximum Velocity	0...100,000	100,000	Set the pulse output maximum velocity (in Hz).
	Start Velocity *(see page 44)*	0...100,000	0	Set the pulse output start velocity (in Hz). 0 if not used.
	Stop Velocity *(see page 44)*	0...100,000	0	Set the pulse output stop velocity (in Hz). 0 if not used.
	Acc./Dec. Unit *(see page 44)*	Hz/ms ms	Hz/ms	Set acceleration/deceleration as rates (Hz/ms) or as time constants from 0 to **Maximum Velocity** (ms).
	Maximum Acceleration	1...100,000	100,000	Set the acceleration maximum value (in **Acc./Dec. Unit**).
	Maximum Deceleration	1...100,000	100,000	Set the deceleration maximum value (in **Acc./Dec. Unit**).

Parameter		Value	Default	Description
Motion / Fast Stop	Fast Stop Deceleration	1...100,000	5,000	Set the deceleration value in case an error is detected (in **Acc./Dec. Unit**)
Homing / REF input	Location	Disabled I8 (channel 0) I11 (channel 1)	Disabled	Select the controller input used for the REF signal *(see page 54)*.
	Bounce filter	0.000 0.001 0.002 0.005 0.010 0.05 0.1 0.5 1 5	0.005	Set the filtering value to reduce the bounce effect on the REF input (in ms).
	Type	Normally opened Normally closed	Normally opened	Select whether the switch contact default state is open or closed.
Homing / INDEX input	Location	Disabled I9 (channel 0) I12 (channel 1)	Disabled	Select the controller input used for the INDEX signal *(see page 54)*.
	Bounce filter	0.000 0.001 0.002 0.005 0.010 0.05 0.1 0.5 1 5	0.005	Set the filtering value to reduce the bounce effect on the INDEX input (in ms).
	Type	Normally opened Normally closed	Normally opened	Select whether the switch contact default state is open or closed.

Parameter		Value	Default	Description
Registration / PROBE input	Location	Disabled I3 or I10 (channel 0) I7 or I13 (channel 1)	Disabled	Select the controller input used for the PROBE signal *(see page 46)*.
	Bounce filter	0.000 0.001 0.002 0.005 0.010 0.05 0.1 0.5 1 5	0.005	Set the filtering value to reduce the bounce effect on the PROBE input (in ms).

Pulse Output Modes

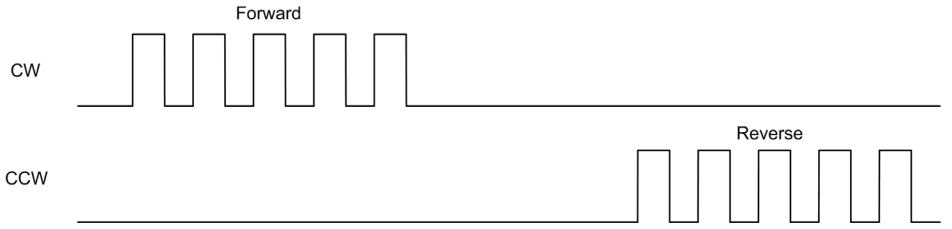
Overview

There are three possible output modes:

- A ClockWise / B CounterClockwise
- A Pulse / B direction
- Quadrature

A ClockWise (CW) / B CounterClockwise (CCW) Mode

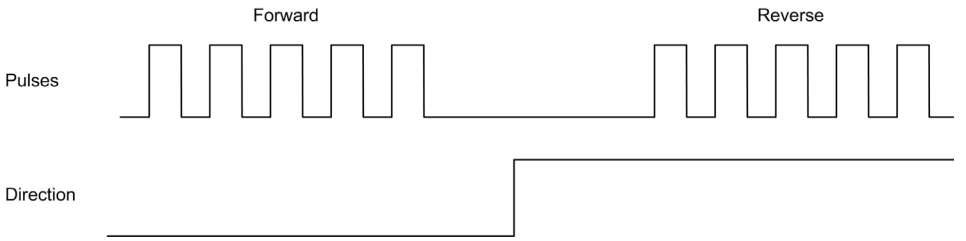
This mode generates a signal that defines the motor operating speed and direction. This signal is implemented either on the PTO output A or on PTO output B depending on the motor rotation direction.



A Pulse / B Direction Mode

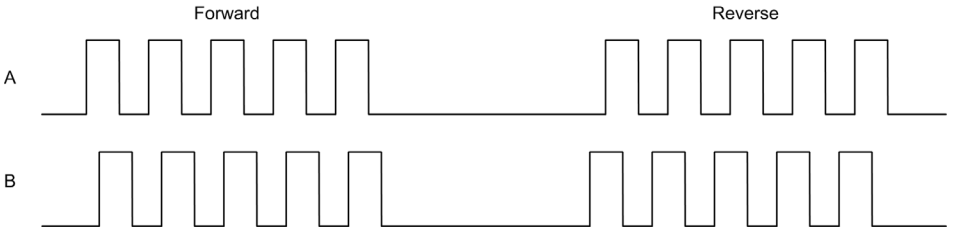
This mode generates two signals on the PTO outputs:

- Output A: pulse which provides the motor operating speed.
- Output B: direction which provides the motor rotation direction.



Quadrature Mode

This mode generates two signals in quadrature phase on the PTO outputs (the phase sign depends on motor direction).



Acceleration / Deceleration Ramp

Start Velocity

The **Start Velocity** is the minimum frequency at which a stepper motor can produce movement, with a load applied, without the loss of steps.

Start Velocity parameter is used when starting a motion from velocity 0.

Start Velocity must be in the range 0...`MaxVelocityAppl` *(see page 77)*.

Value 0 means that the **Start Velocity** parameter is not used. In this case, the motion starts at a velocity = acceleration rate x 1 ms.

Stop Velocity

The **Stop Velocity** is the maximum frequency at which a stepper motor stops producing movement, with a load applied, without loss of steps.

Stop Velocity is only used when moving from a higher velocity than **Stop Velocity**, down to velocity 0.

Stop Velocity must be in the range 0...`MaxVelocityAppl` *(see page 77)*.

Value 0 means that the **Stop Velocity** parameter is not used. In this case, the motion stops at a velocity = deceleration rate x 1 ms.

Acceleration / Deceleration

Acceleration is the rate of velocity change, starting from **Start Velocity** to target velocity. Deceleration is the rate of velocity change, starting from target velocity to **Stop Velocity**. These velocity changes are implicitly managed by the PTO function in accordance with `Acceleration`, `Deceleration` and `JerkRatio` parameters following a **trapezoidal** or an **S-curve** profile.

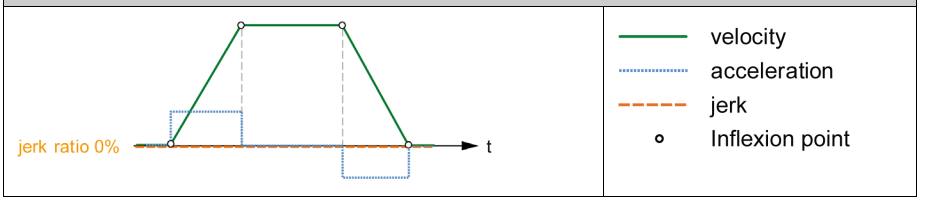
Acceleration / Deceleration Ramp with a Trapezoidal Profile

When the jerk ratio parameter is set to 0, the acceleration / deceleration ramp has a trapezoidal profile.

Expressed in Hz/ms, the `acceleration` and `deceleration` parameters represent the rate of velocity change.

Expressed in ms, they represent the time to go from 0 to **Maximum velocity**.

`JerkRatio` 0%: Constant acceleration / deceleration.

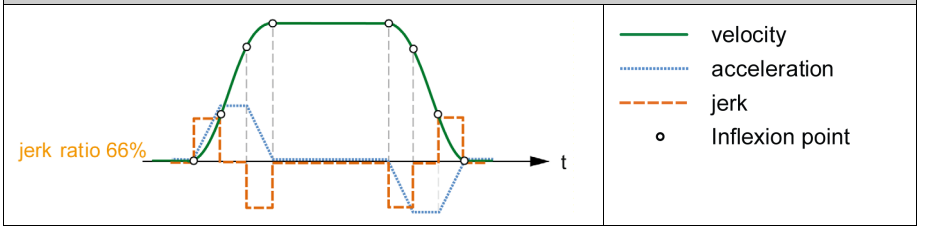


Acceleration / Deceleration Ramp with an S-curve Profile

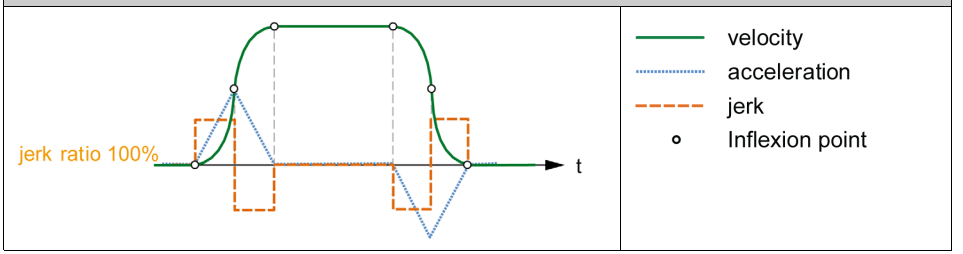
When the jerk ratio parameter is greater than 0, the acceleration / deceleration ramp has an S-curve profile.

The S-curve ramp is used in applications controlling high inertia, or in those that manipulate fragile objects or liquids. The S-curve ramp enables a smoother and progressive acceleration / deceleration, as demonstrated in the following graphics:

`JerkRatio` 66%: 2/3 of the acceleration and deceleration time is spent in increasing and decreasing the acceleration and deceleration value.



`JerkRatio` 100%: The entire time is spent in increasing and decreasing the acceleration and deceleration value.



NOTE: The `JerkRatio` parameter value is common for acceleration and deceleration so that concave time and convex time are equal.

Affect of the S-Curve Ramp on Acceleration / Deceleration

The duration for the acceleration / deceleration is maintained, whatever the `JerkRatio` parameter may be. To maintain this duration, the acceleration or deceleration is other than that configured in the function block (`Acceleration` or `Deceleration` parameters).

When the `JerkRatio` is applied, the acceleration / deceleration is affected.

When the `JerkRatio` is applied at 100%, the acceleration / deceleration is two times that of the configured `Acceleration`/`Deceleration` parameters.

NOTE: The `JerkRatio` is re-calculated to respect the `MaxAccelerationAppl` and `MaxDecelerationAppl` parameters.

Probe Event

Description

The Probe input is enabled by configuration, and activated using the `MC_TouchProbe_PTO` function block.

The Probe input is used as an event to:

- capture the position,
- start a move independently of the task.

Both functions can be active at the same time, that is, the same event captures the position and start a motion function block *(see page 81)*.

The Probe input event can be defined to be enabled within a predefined window that is demarcated by position limits (refer to `MC_TouchProbe_PTO` *(see page 153)*.

NOTE: Only the first event after the rising edge at the `MC_TouchProbe_PTO` function block `Busy` pin is valid. Once the `Done` output pin is set, subsequent events are ignored. The function block needs to be reactivated to respond to other events.

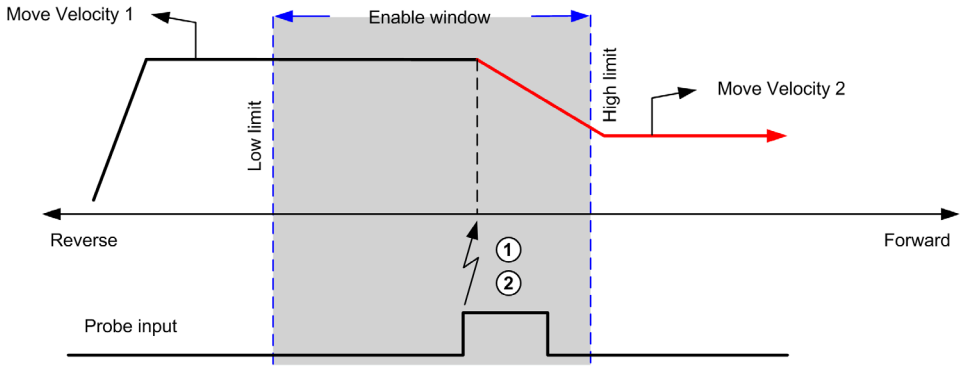
Position Capture

The position captured is available in `MC_TouchProbe_PTO.RecordedPosition`.

Motion Trigger

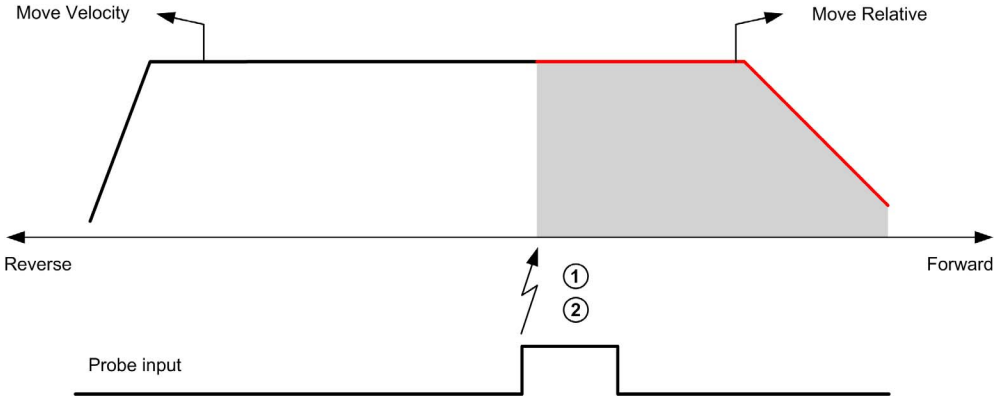
The `BufferMode` input of a motion function block must be set to `seTrigger`.

This example illustrates a change target velocity with enable window:



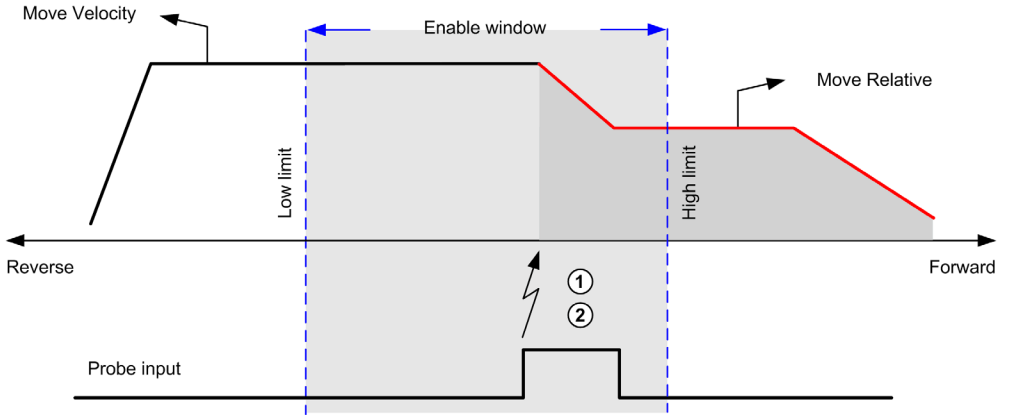
1 Capture the position counter value
2 Trigger `Move Velocity` function block

This example illustrates a move of pre-programmed distance, with simple profile and no enable window:



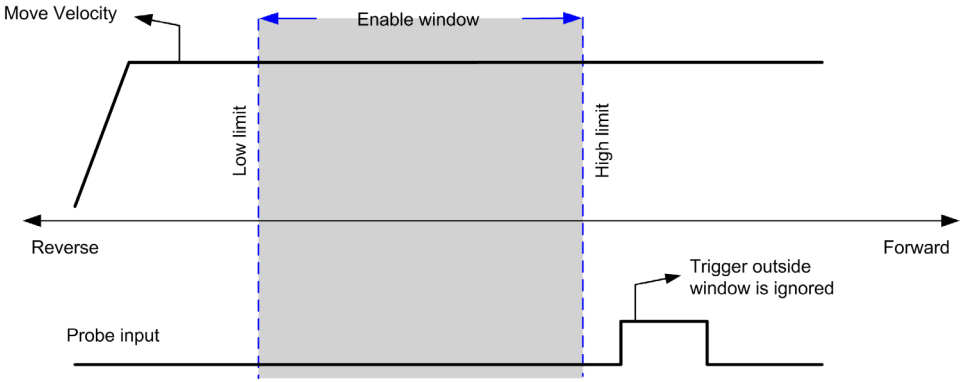
1 Capture the position counter value
2 Trigger `Move Relative` function block

This example illustrates a move of pre-programmed distance, with complex profile and enable window:



1 Capture the position counter value
2 Trigger `Move Relative` function block

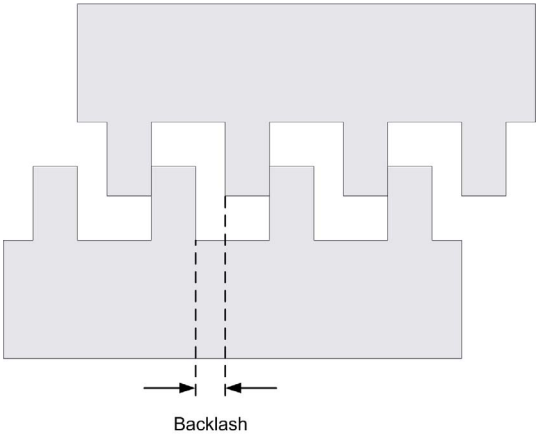
This example illustrates a trigger event out of enable window:



Backlash Compensation (Only Available in Quadrature Mode)

Description

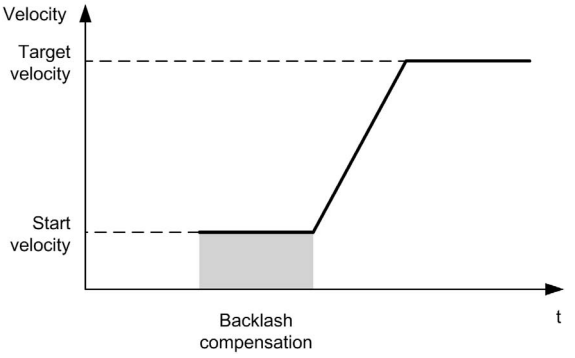
The **Backlash Compensation** parameter is defined as the amount of motion needed to compensate the mechanical clearance in gears, when movement is reversed:



NOTE: The function does not take into account any external sources of movement, such as inertia movement or other forms of induced movement.

Backlash compensation is set in number of pulses (0...255, default value is 0). When set, at each direction reversal, the specified number of pulses is first output at start velocity, and then the programmed movement is executed. The backlash compensation pulses are not added to the position counter.

This figure illustrates the backlash compensation:



NOTE:

- Before the initial movement is started, the function cannot determine the amount of backlash to compensate for. Therefore, the backlash compensation is only active after a homing is successfully performed. If the homing is performed without movement, it is assumed that the initial movement applies no compensation, and the compensation is applied at the first direction reversal.
- Once started, the compensation pulses are output until completion, even if an aborting command is received in the meantime. In this case, the aborting command is buffered and will start as soon as compensation pulses are output. No additional buffered command is accepted in this case.
- If the axis is stopped by an error detected before all the compensation pulses are output, the backlash compensation is reset. A new homing procedure is needed to reinitialize the backlash compensation.
- Backlash timeout of 80 s: The system does not accept to configure a movement of more than 80 s. So if a backlash is configured, it may for example not be more than 80 pulses to 1 Hz. The error detected in case of this timeout is "Invalid acceleration" (code 1000).

Positioning Limits

Introduction

Positive and negative limits can be set to control the movement boundaries in both directions. Both hardware and software limits are managed by the controller.

Hardware and software limit switches are used to manage boundaries in the controller application only. They are not intended to replace any functional safety limit switches wired to the drive. The controller application limit switches must necessarily be activated before the functional safety limit switches wired to the drive. In any case, the type of functional safety architecture, which is beyond the scope of the present document, that you deploy depends on your safety analysis, including, but not limited to:

- risk assessment according to EN/ISO 12100
- FMEA according to EN 60812

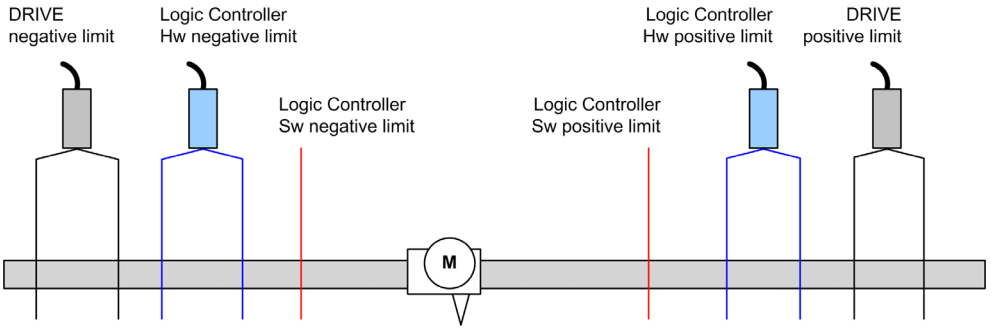
⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

Ensure that a risk assessment is conducted and respected according to EN/ISO 12100 during the design of your machine.

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

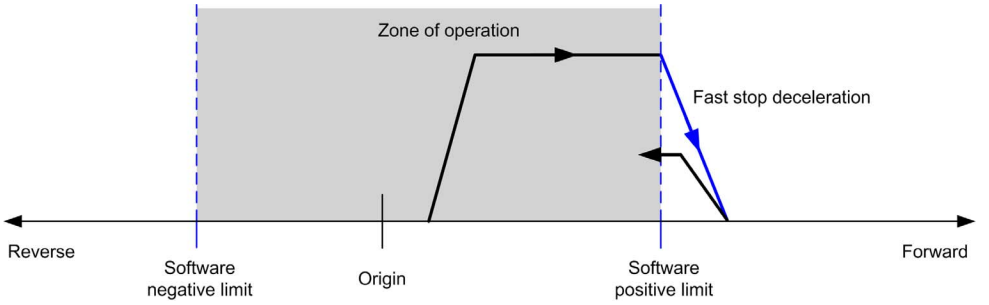
The figure illustrates hardware and software limit switches:



Once either the controller hardware or software limits are crossed, an error is detected and a Fast stop deceleration is performed:

- the axis switches to **ErrorStop** state, with `ErrorId` 1002 to 1005 (`PTO_ERROR` *(see page 78)*),
- the function block under execution detects the error state,
- status bits on other applicable function blocks are set to `CommandAborted`.

To clear the axis error state, and return to a **Standstill** state, execution of `MC_Reset_PTO` is required as any motion command will be rejected (refer to PTO parameters `EnableDirPos` or `EnableDirNeg`) while the axis remains outside the limits (function block terminates with `ErrorId=InvalidDirectionValue`). It is only possible to execute a motion command in the opposite direction under these circumstances.



Software Limits

Software limits can be set to control the movement boundaries in both directions.

Limit values are enabled and set in the configuration screen, such that:

- Positive limit > Negative limit
- Values in the range -2,147,483,648 to 2,147,483,647

They can also be enabled, disabled, or modified in the application program (`MC_WriteParameter_PTO` *(see page 146)* and PTO_PARAMETER *(see page 77)*).

NOTE: When enabled, the software limits are valid after an initial homing is successfully performed (that is, the axis is homed, `MC_Home_PTO` *(see page 115)*).

NOTE: An error is only detected when the software limit is physically reached, not at the initiation of the movement.

Hardware Limits

Hardware limits are required for the homing procedure, and for helping to prevent damage to the machine. The appropriate inputs must be used on the `MC_Power_PTO.LimP` and `MC_Power_PTO.LimN` input bits. The hardware limit devices must be of a normally closed type such that the input to the function block is FALSE when the respective limit is reached.

NOTE: The restrictions over movement are valid while the limit inputs are FALSE and regardless of the sense of direction. When they return to TRUE, movement restrictions are removed and the hardware limits are functionnally rearmed. Therefore, use falling edge contacts leading to RESET output instructions prior to the function block. Then use those bits to control these function block inputs. When operations are complete, SET the bits to restore normal operation.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

- Ensure that controller hardware limit switches are integrated in the design and logic of your application.
- Mount the controller hardware limit switches in a position that allows for an adequate braking distance.

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

NOTE: Adequate braking distance is dependent on the maximum velocity, maximum load (mass) of the equipment being moved, and the value of the Fast stop deceleration parameter.

Section 4.2
Home Modes

Overview

This section describes the PTO home modes.

What Is in This Section?

This section contains the following topics:

Homing Modes

Description

Homing is the method used to establish the reference point or origin for absolute movement.

A homing movement can be made using different methods. The M241 PTO channels provide several standard homing movement types:

- position setting *(see page 57)*,
- long reference *(see page 58)*,
- long reference and index *(see page 60)*,
- short reference reversal *(see page 62)*,
- short reference no reversal *(see page 64)*,
- short reference and index outside *(see page 66)*,
- short reference and index inside *(see page 68)*.

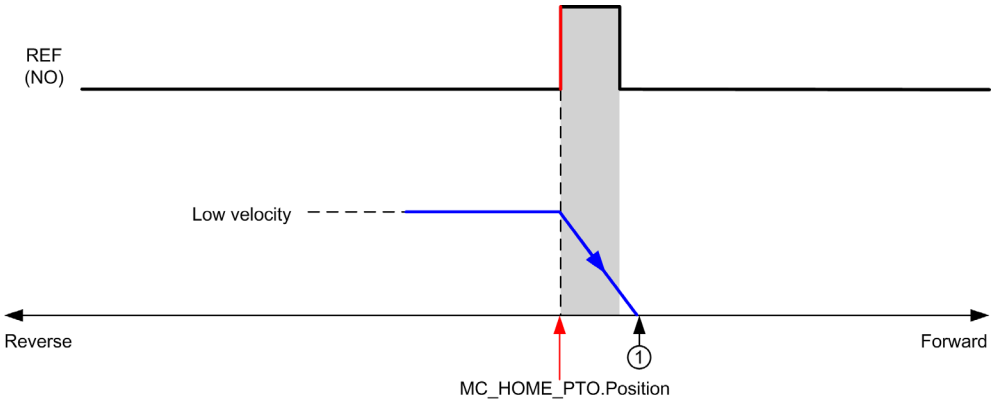
A homing movement must be terminated without interruption for the new reference point to be valid. If the reference movement is interrupted, it needs to be started again.

Refer to `MC_Home_PTO` *(see page 115)* and PTO_HOMING_MODE *(see page 76)*.

Home Position

Homing is done with an external switch and the homing position is defined on the switch edge. Then the motion is decelerated until stop.

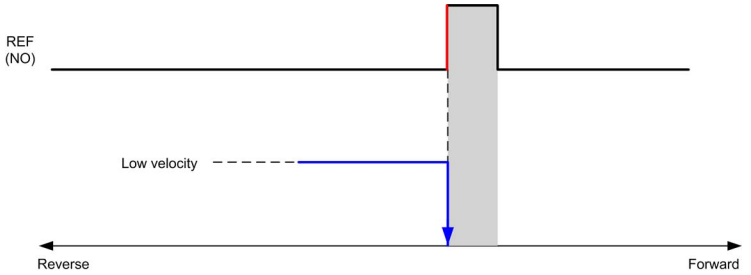
The actual position of the axis at the end of the motion sequence may therefore differ from the position parameter set on the function block:



REF (NO) Reference point (Normally Open)

1 Position at the end of motion = `MC_HOME_PTO.Position` + “deceleration to stop” distance.

To simplify the representation of a stop in the homing mode diagrams, the following presentation is made to represent the actual position of the axis:



REF (NO) Reference point (Normally Open)

Limits

Hardware limits are necessary for the correct functioning of the `MC_Home_PTO` function block (Positioning Limits *(see page 51)* and `MC_Power_PTO` *(see page 93)*). Depending on the movement type you request with the homing mode, the hardware limits help assure that the end of travel is respected by the function block.

When a homing action is initiated in a direction away from the reference switch, the hardware limits serve to either:

- indicate a reversal of direction is required to move the axis toward the reference switch or,
- indicate that an error has been detected as the reference switch was not found before reaching the end of travel.

For homing movement types that allow for reversal of direction, when the movement reaches the hardware limit the axis stops using the configured deceleration, and resumes motion in a reversed direction.

In homing movement types that do not allow for the reversal of direction, when the movement reaches the hardware limit, the homing procedure is aborted and the axis stops with the Fast stop deceleration.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

- Ensure that controller hardware limit switches are integrated in the design and logic of your application.
- Mount the controller hardware limit switches in a position that allows for an adequate braking distance.

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

NOTE: Adequate braking distance is dependent on the maximum velocity, maximum load (mass) of the equipment being moved, and the value of the Fast stop deceleration parameter.

Position Setting

Description

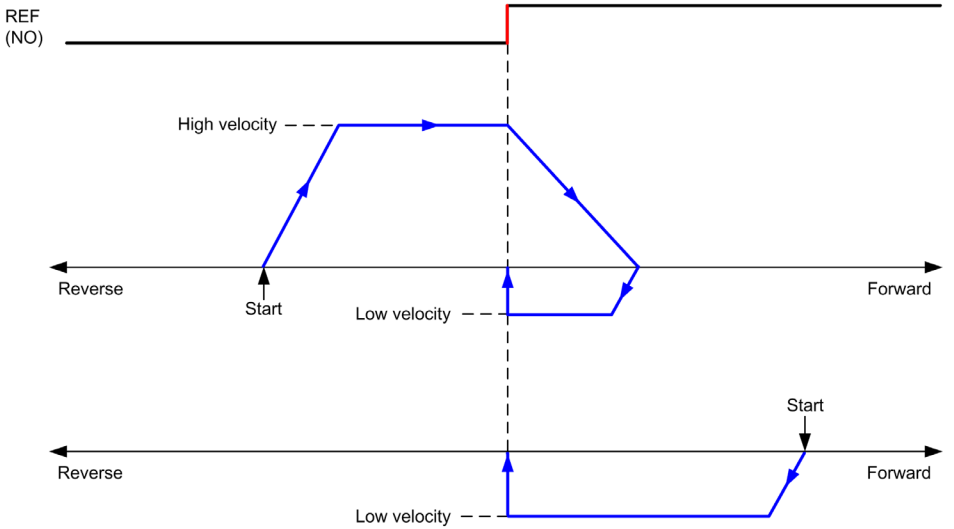
In the case of position setting, the current position is set to the specified position value. No move is performed.

Long Reference

Long Reference: Positive Direction

Homes to the reference switch falling edge in reverse direction.

The initial direction of motion is dependent on the state of the reference switch:

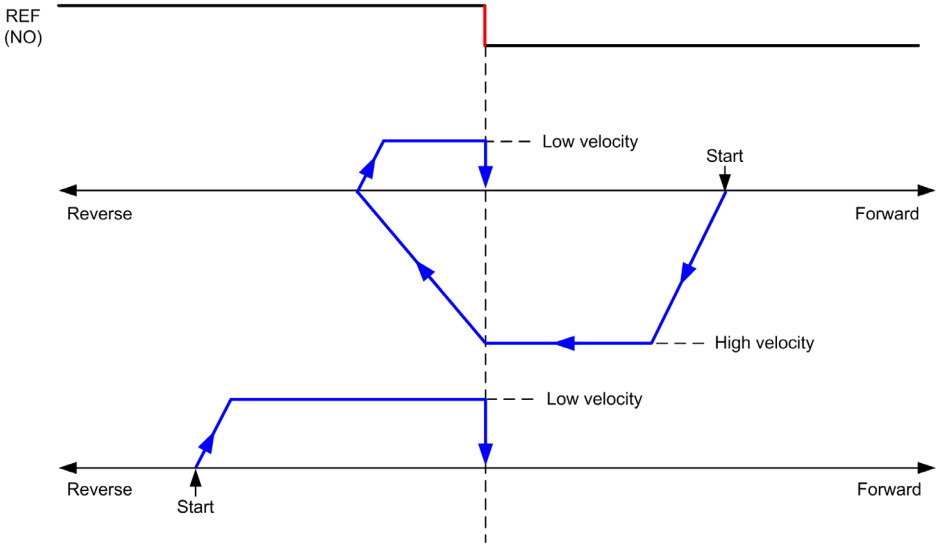


REF (NO) Reference point (Normally Open)

Long Reference: Negative Direction

Homes to the reference switch falling edge in forward direction.

The initial direction of motion is dependent on the state of the reference switch:



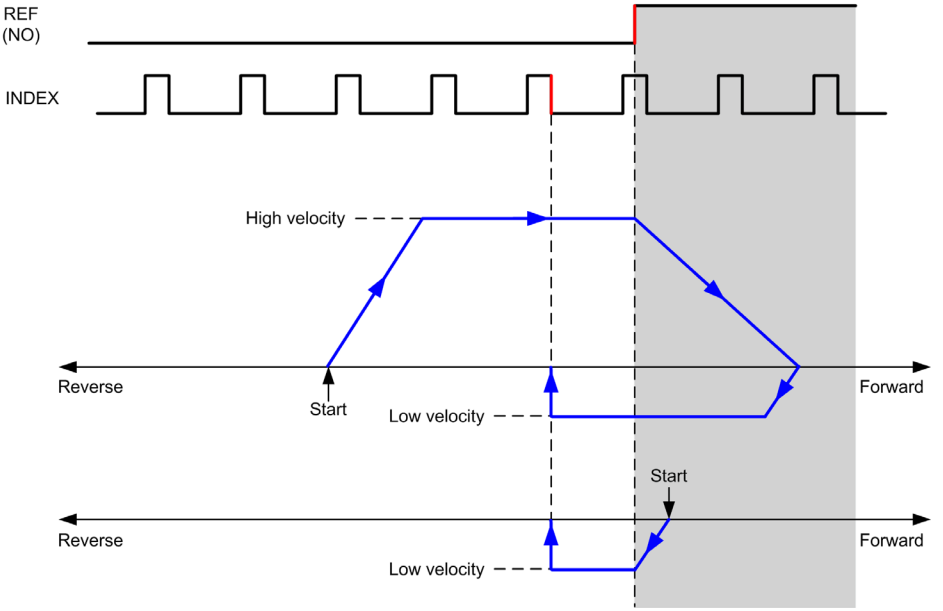
REF (NO) Reference point (Normally Open)

Long Reference & Index

Long Reference & Index: Positive Direction

Homes to the first index, after the reference switch falling edge in reverse direction.

The initial direction of motion is dependent on the state of the reference switch:

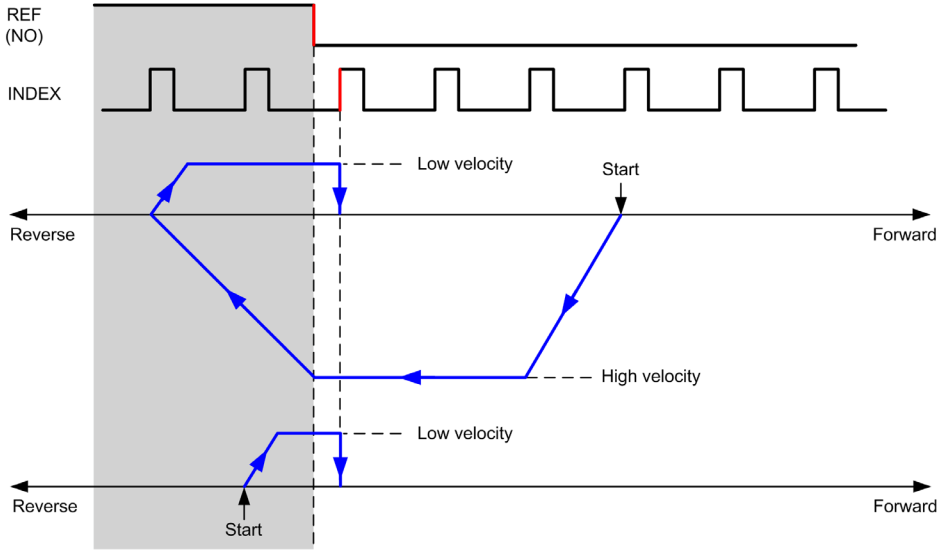


REF (NO) Reference point (Normally Open)

Long Reference & Index: Negative Direction

Homes to the first index, after the reference switch falling edge in forward direction.

The initial direction of motion is dependent on the state of the reference switch:



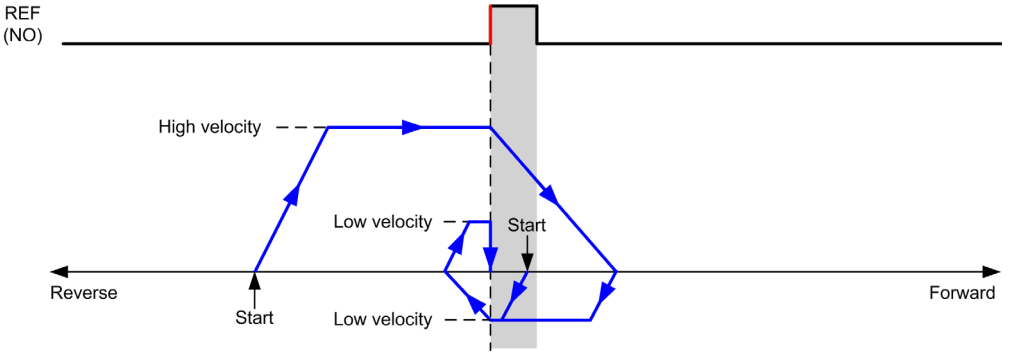
REF (NO) Reference point (Normally Open)

Short Reference Reversal

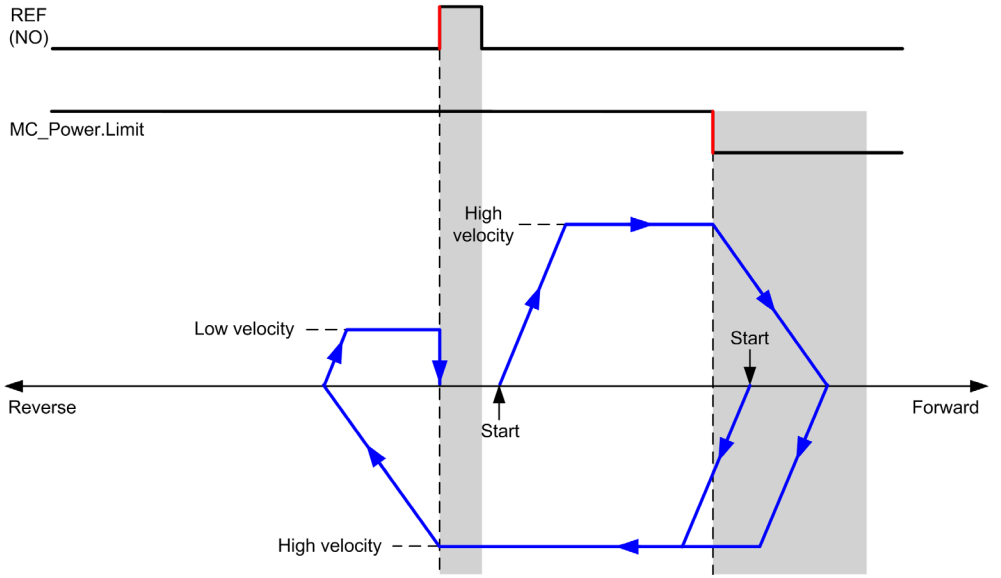
Short Reference Reversal: Positive Direction

Homes to the reference switch rising edge in forward direction.

The initial direction of motion is dependent on the state of the reference switch:



REF (NO) Reference point (Normally Open)

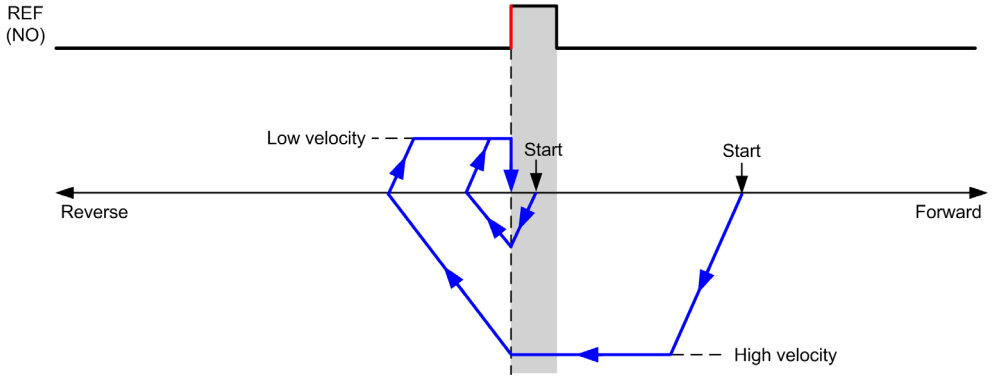


REF (NO) Reference point (Normally Open)

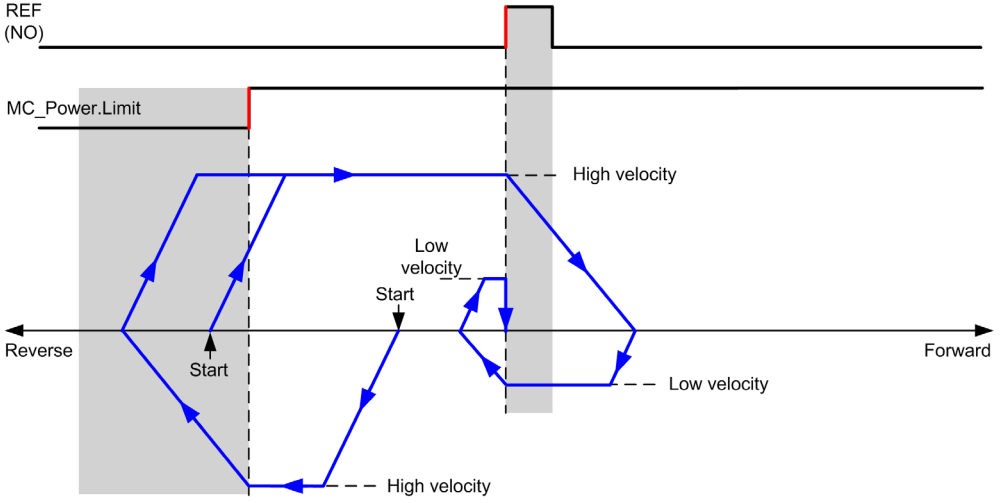
Short Reference Reversal: Negative Direction

Homes to the reference switch rising edge in forward direction.

The initial direction of motion is dependent on the state of the reference switch:



REF (NO) Reference point (Normally Open)

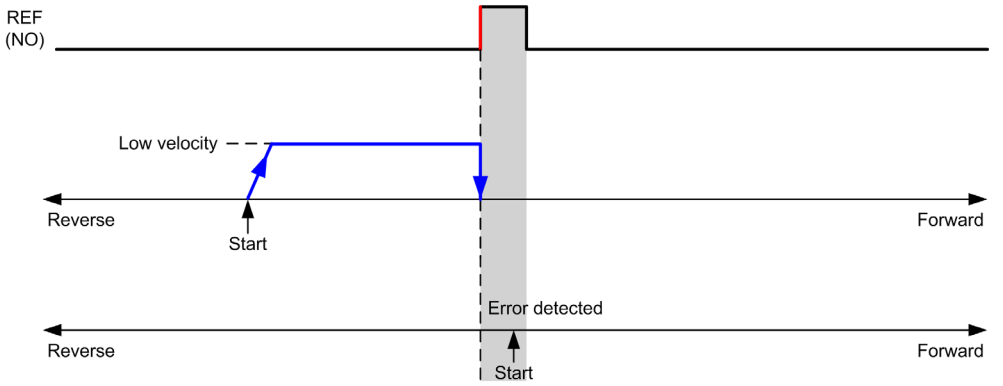


REF (NO) Reference point (Normally Open)

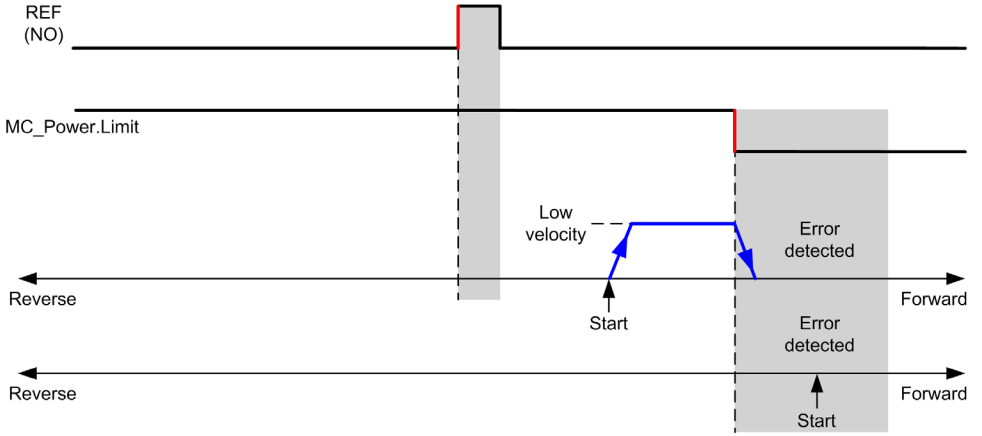
Short Reference No Reversal

Short Reference No Reversal: Positive Direction

Homes at low speed to the reference switch rising edge in forward direction, with no reversal:



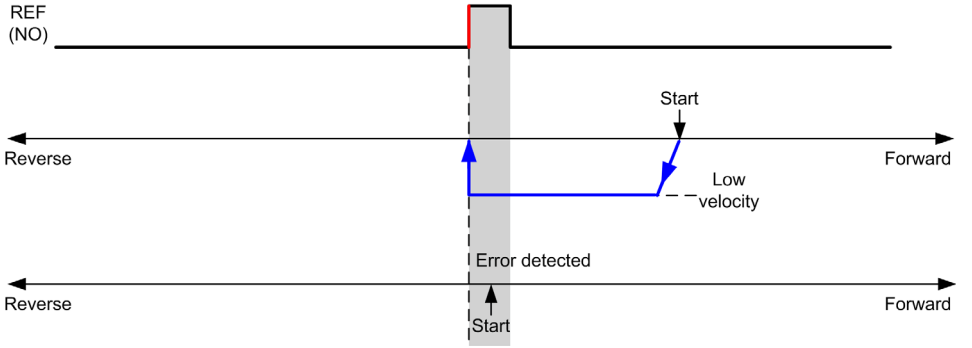
REF (NO) Reference point (Normally Open)



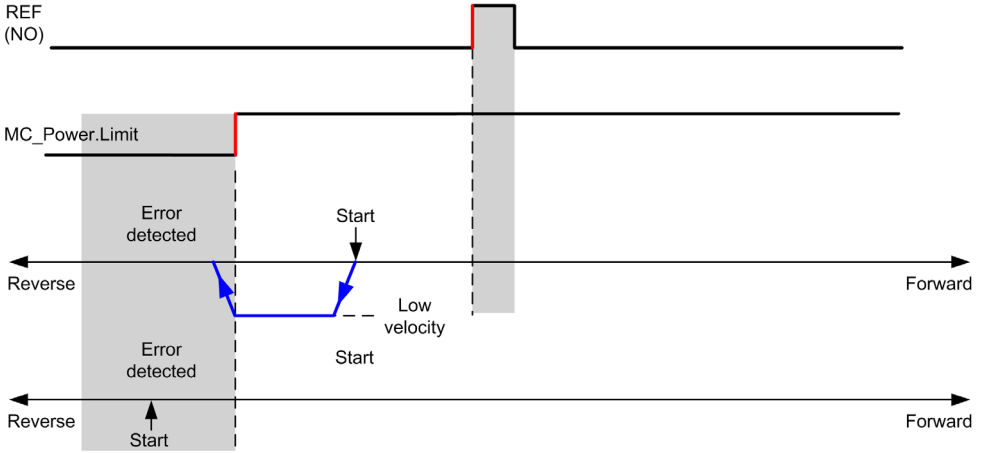
REF (NO) Reference point (Normally Open)

Short Reference No Reversal: Negative Direction

Homes at low speed to the reference switch falling edge in reverse direction, with no reversal:



REF (NO) Reference point (Normally Open)



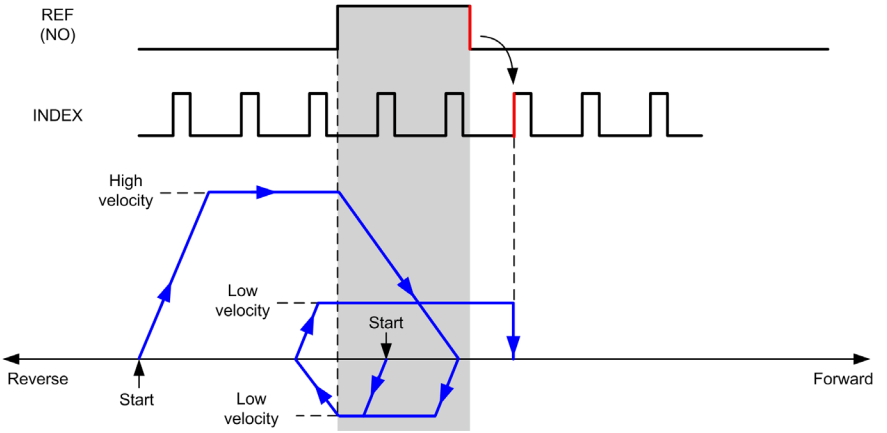
REF (NO) Reference point (Normally Open)

Short Reference & Index Outside

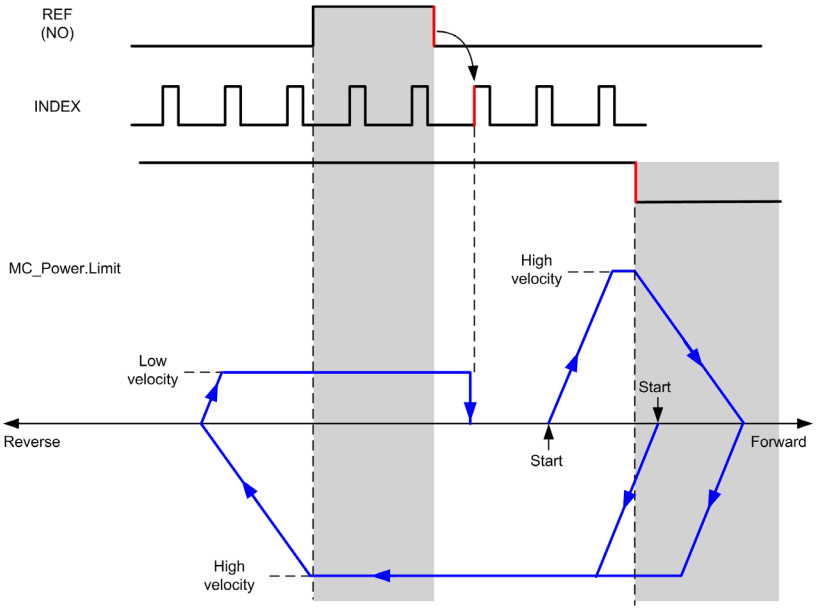
Short Reference & Index Outside: Positive Direction

Homes to the first index, after the reference switch transitions on and off in forward direction.

The initial direction of motion is dependent on the state of the reference switch:



REF (NO) Reference point (Normally Open)

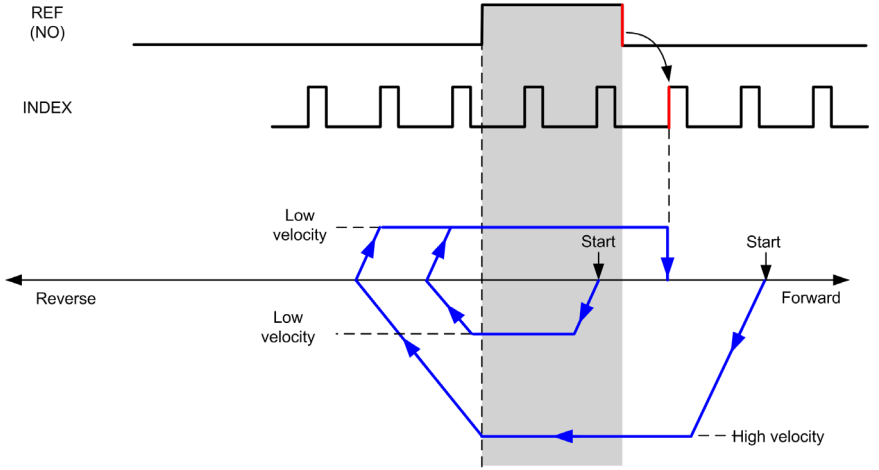


REF (NO) Reference point (Normally Open)

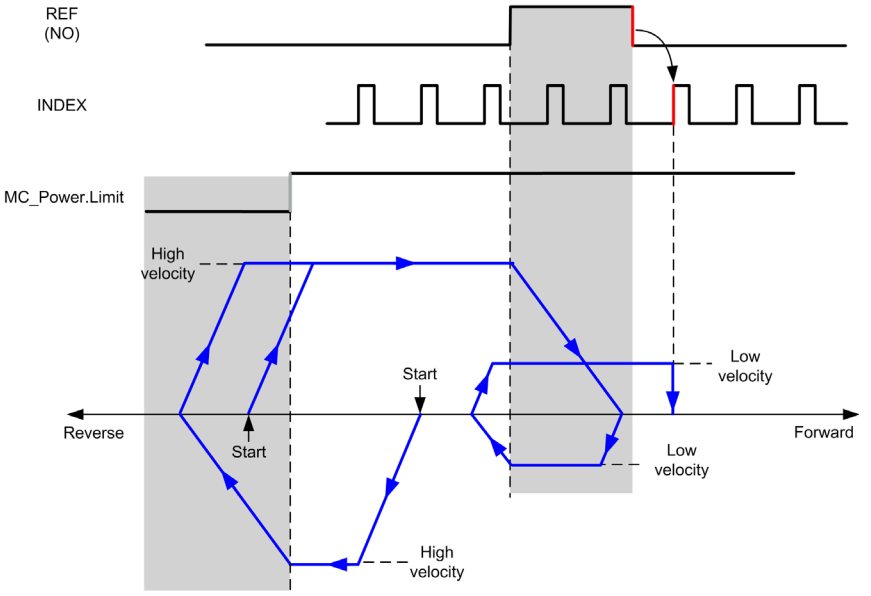
Short Reference & Index Outside: Negative Direction

Homes to the first index, after the reference switch transitions on and off in forward direction.

The initial direction of motion is dependent on the state of the reference switch:



REF (NO) Reference point (Normally Open)



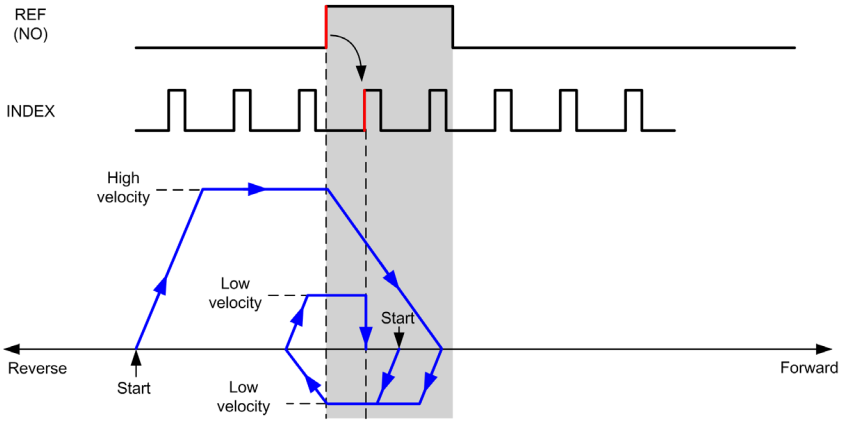
REF (NO) Reference point (Normally Open)

Short Reference & Index Inside

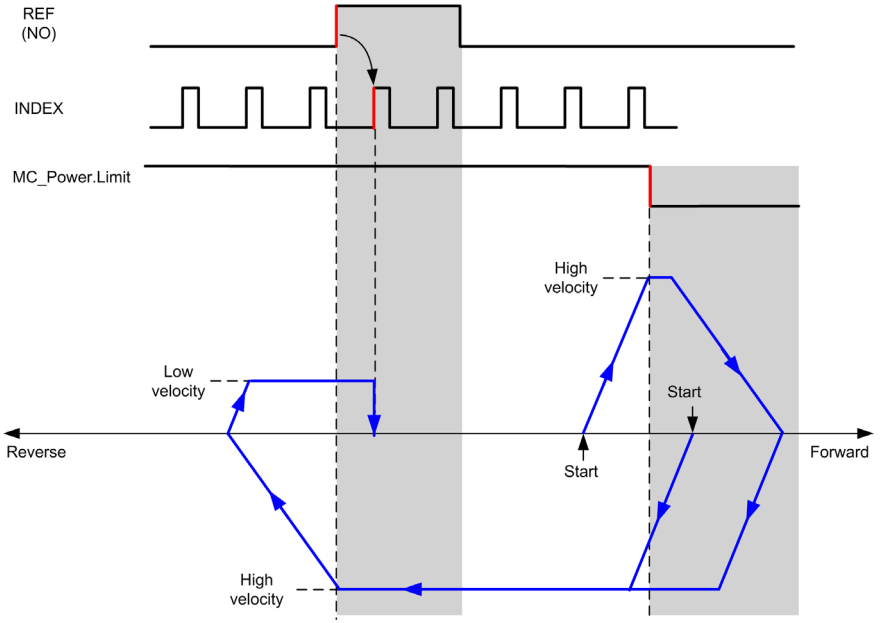
Short Reference & Index Inside: Positive Direction

Homes to the first index, after the reference switch rising edge in forward direction.

The initial direction of motion is dependent on the state of the reference switch:



REF (NO) Reference point (Normally Open)

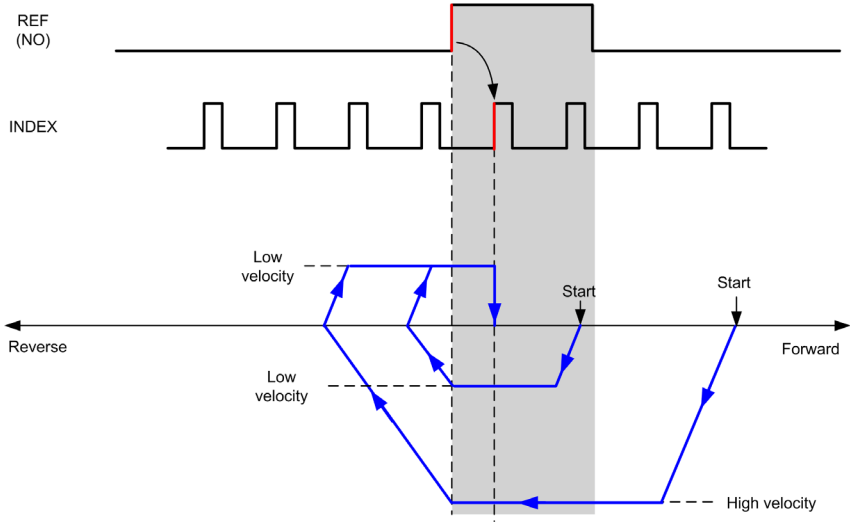


REF (NO) Reference point (Normally Open)

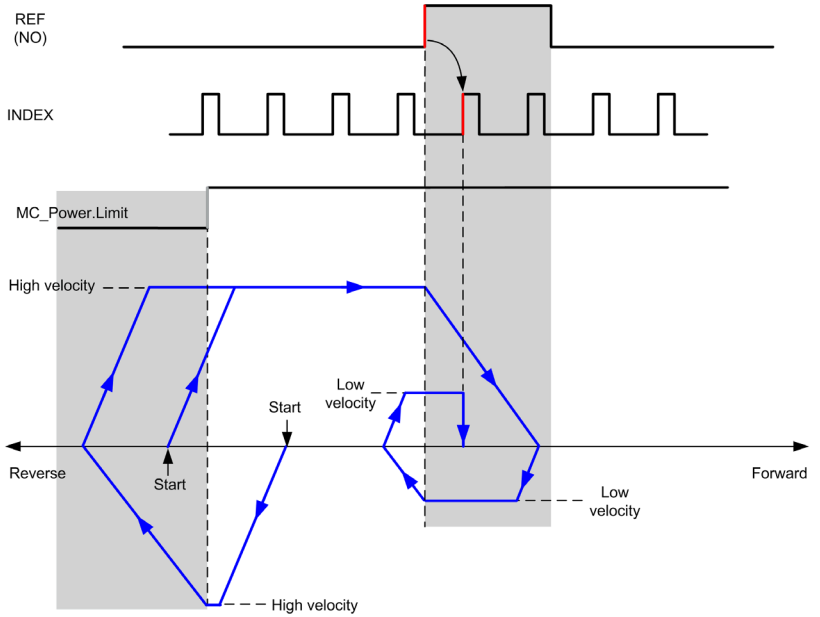
Short Reference & Index Inside: Negative Direction

Homes to the first index, after the reference switch rising edge in forward direction.

The initial direction of motion is dependent on the state of the reference switch:



REF (NO) Reference point (Normally Open)



REF (NO) Reference point (Normally Open)

Home Offset

Description

If the origin cannot be defined by switches with enough accuracy, it is possible to make the axis move to a specific position away from the origin switch. Home offset allows making a difference between mechanical origin and electrical origin.

Home offset is set in number of pulses (-2,147,483,648...2,147,483,647, default value 0). When set by configuration, the `MC_Home_PTO` *(see page 115)* command is executed first, and then the specified number of pulses is output at the home low velocity in the specified direction. The parameter is only effective during a reference movement without index pulse.

NOTE: The wait time between `MC_Home_PTO` command stop on origin switch and start of offset movement is fixed, set to 500 ms. The `MC_Home_PTO` command busy flag is only released after origin offset has been completed.

Chapter 5
Data Unit Types

Overview

This chapter describes the data unit types of the M241 PTO Library.

What Is in This Chapter?

This chapter contains the following topics:

AXIS_REF_PTO Data Type

Data Type Description

The AXIS_REF_PTO type is a data type that contains information on the corresponding axis. It is used as a VAR_IN_OUT in all function blocks of the PTO library.

MC_BUFFER_MODE

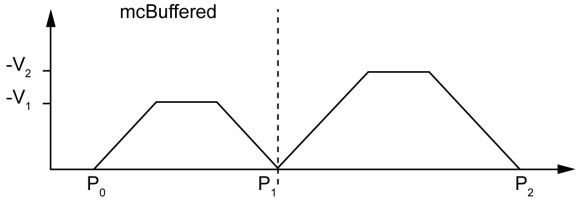
Buffer Mode Enumeration

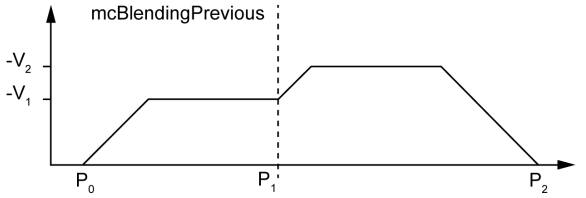
This table lists the values for the MC_BUFFER_MODE enumeration:

Enumerator	Value	Description
`mcAborting`	0	Start FB immediately (default mode). Any ongoing motion is aborted. The move queue is cleared.
`mcBuffered`	1	Start FB after current move has finished (`Done` or `InVelocity` bit is set). There is no blending.
`mcBlendingPrevious`	3	The velocity is blended with the velocity of the first FB (blending with the velocity of `FB1` at end-position of `FB1`).
`seTrigger`	10	Start FB immediately when an event on the probe input is detected. Any ongoing motion is aborted. The move queue is cleared.
`seBufferedDelay`	11	Start FB after current motion has finished (`Done` or `InVelocity` bit is set) and the time delay has elapsed. There is no blending. The `Delay` parameter is set using `MC_WriteParameter_PTO` *(see page 146)*, with `ParameterNumber` 1000.

Examples

The examples below show a movement executed by two motion commands. The axis moves from the position P_0 to P_1 and then P_2. The second command is passed while the axis is executing the first command but before the stopping ramp is reached. For each motion profile below, P_1 is the reference point for the blending calculation. The buffer mode determines whether velocity V_1 or V_2 is reached at position P_1.




mcBlendingPrevious
-V2
-V1
P0
P1
P2

MC_DIRECTION

Move Direction Enumeration

This table lists the values for the MC_DIRECTION enumeration:

Enumerator	Value	Description
`mcPositiveDirection`	1	CW, forward, positive (according to **Output Mode** configuration setting).
`mcNegativeDirection`	-1	CCW, backward, reverse, negative (according to **Output Mode** configuration setting).
`mcCurrentDirection`	2	Move in the last used direction.

PTO_HOMING_MODE

Homing Mode Enumeration

This table lists the values for the PTO_HOMING_MODE enumeration:

Enumerator	Value	Description
PositionSetting	0	Position.
LongReference	1	Long reference.
LongReferenceAndIndex	10	Long reference and index.
ShortReference_Reversal	20	Short reference.
ShortReference_NoReversal	21	Short reference no reversal.
ShortReferenceAndIndex_Outside	30	Short reference and index outside.
ShortReferenceAndIndex_Inside	31	Short reference and index inside.

PTO_PARAMETER

PTO Parameter Enumeration

This table lists the values for the PTO_PARAMETER enumeration:

Parameter Name	Parameter Number	Type	Standard	R/W	Description
CommandedPosition	1	DINT	Mandatory	R	Commanded position.
SWLimitPos	2	DINT	Optional	R/W	Positive software limit switch position.
SWLimitNeg	3	DINT	Optional	R/W	Negative software limit switch position.
EnableLimitPos	4	BOOL	Optional	R/W	Enable positive software limit switch.
EnableLimitNeg	5	BOOL	Optional	R/W	Enable negative software limit switch.
MaxVelocityAppl	9	DINT	Mandatory	R/W	Maximal allowed velocity of the axis in the application.
ActualVelocity	10	DINT	Mandatory	R	Actual velocity.
CommandedVelocity	11	DINT	Mandatory	R	Commanded velocity.
MaxAccelerationAppl	13	DINT	Optional	R/W	Maximal allowed acceleration of the axis in the application.
MaxDecelerationAppl	15	DINT	Optional	R/W	Maximal allowed deceleration of the axis in the application.
Reserved	to 999	-	-	-	Reserved for the PLCopen standard.
Delay	1000	DINT	Vendor specific	R/W	Time in ms (0...65,535) Default value: 0

PTO_ERROR

PTO Error Enumeration

This table lists the values for the PTO_ERROR enumeration:

Enumerator	Value	Description
NoError	0	No error detected.
Axis Control Alerts		
InternalError	1000	Motion controller internal error detected.
DisabledAxis	1001	The move could not be started or has been aborted because the axis is not ready.
HwPositionLimitP	1002	Hardware positive position limit limP exceeded.
HwPositionLimitN	1003	Hardware negative position limit limN exceeded.
SwPositionLimitP	1004	Software positive position limit exceeded.
SwPositionLimitN	1005	Software negative position limit exceeded.
ApplicationStopped	1006	Application execution has been stoppped (power cycle, controller in STOPPED or HALT state).
OutputProtection	1007	Short-circuit output protection is active on the PTO channels.
Axis Control Advisories		
WarningVelocityValue	1100	Commanded Velocity parameter is out of range.
WarningAccelerationValue	1101	Commanded Acceleration parameter is out of range.
WarningDecelerationValue	1102	Commanded Deceleration parameter is out of range.
WarningDelayedMove	1103	Not enough time to stop the active move, so the requested move is delayed.
WarningJerkRatioValue	1104	Commanded jerk ratio parameter is limited by the configured maximum acceleration or deceleration. In this case, the jerk ratio is recalculated to respect these maximums.
Motion State Advisories		
ErrorStopActive	2000	The move could not be started or has been aborted because motion is prohibited by an **ErrorStop** condition.
StoppingActive	2001	The move could not be started because motion is prohibited by MC_Stop_PTO having control of the axis (either the axis is stopping, or MC_Stop_PTO.Execute input is held high).
InvalidTransition	2002	Transition not allowed, refer to the Motion State Diagram *(see page 83)*.

Enumerator	Value	Description
InvalidSetPosition	2003	MC_SetPosition_PTO cannot be executed while the axis is moving.
HomingError	2004	Homing sequence cannot start on reference cam in this mode.
InvalidProbeConf	2005	The Probe input must be configured.
InvalidHomingConf	2006	The home inputs (Ref, Index) must be configured for this homing mode.
InvalidAbsolute	2007	An absolute move cannot be executed while the axis is not successfully homed to an origin position. A homing sequence must be executed first (MC_Home_PTO *(see page 115)*).
MotionQueueFull	2008	The move could not be buffered because the motion queue is full.
Range Advisories		
InvalidAxis	3000	The function block is not applicable for the specified axis.
InvalidPositionValue	3001	Position parameter is out of limits, or distance parameter gives an out of limits position.
InvalidVelocityValue	3002	Velocity parameter is out of range.
InvalidAccelerationValue	3003	Acceleration parameter is out of range.
InvalidDecelerationValue	3004	Deceleration parameter is out of range.
InvalidBufferModeValue	3005	Buffer mode does not correspond to a valid value.
InvalidDirectionValue	3006	Direction does not correspond to a valid value, or direction is invalid due to software position limit exceeded.
InvalidHomeMode	3007	Home mode is not applicable.
InvalidParameter	3008	The parameter number does not exist for the specified axis.
InvalidParameterValue	3009	Parameter value is out of range.
ReadOnlyParameter	3010	Parameter is read-only.

An **Axis Control Alert** switches the axis in **ErrorStop** state (MC_Reset_PTO is mandatory to get out of **ErrorStop** state). The resulting axis status is reflected by MC_ReadStatus_PTO and MC_ReadAxisError_PTO.

A **Motion State Advisory** or a **Range Advisory** does not affect the axis state, nor any ongoing move, nor the move queue. In this case, the error is only local to the applicable function block: the Error output is set, and the ErrorId pin is set to the appropriate PTO_ERROR value.

Chapter 6
Motion Function Blocks

Overview

This chapter describes the motion function blocks.

A motion function block acts on the diagram of axis state, to modify the motion of the axis. These function blocks can return a status to the application before the move is complete. The application program uses these status bits to determine the move status (`Done`, `Busy`, `Active`, `CommandAborted`, and detected `Error`). For axis status, you can use the `MC_ReadStatus_PTO` function block.

What Is in This Chapter?

This chapter contains the following sections:

Section 6.1
Operation Modes

Overview

This section describes the operation modes.

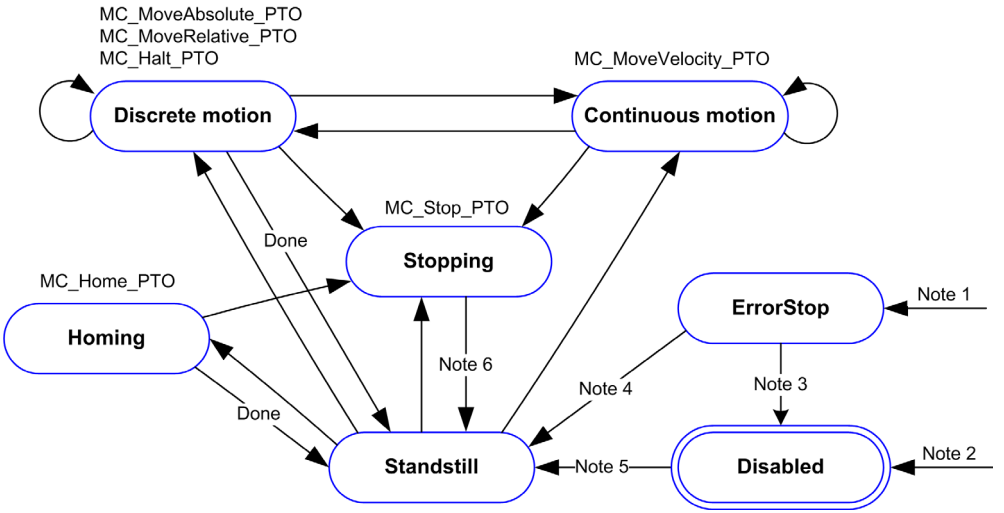
What Is in This Section?

This section contains the following topics:

Motion State Diagram

State Diagram

The axis is always in one of the defined states in this diagram:



Note 1 From any state, when an error is detected.
Note 2 From any state except **ErrorStop**, when `MC_Power_PTO.Status` = FALSE.
Note 3 `MC_Reset_PTO.Done` = TRUE and `MC_Power_PTO.Status` = FALSE.
Note 4 `MC_Reset_PTO.Done` = TRUE and `MC_Power_PTO.Status` = TRUE.
Note 5 `MC_Power_PTO.Status` = TRUE.
Note 6 `MC_Stop_PTO.Done` = TRUE and `MC_Stop_PTO.Execute` = FALSE.

The table describes the axis states:

State	Description
Disabled	Initial state of the axis, no motion command is allowed. The axis is not homed.
Standstill	Power is on, there is no error detected, and there are no motion commands active on the axis. Motion command is allowed.
ErrorStop	Highest priority, applicable when an error is detected on the axis or in the controller. Any ongoing move is aborted by a **Fast Stop Deceleration**. `Error` pin is set on applicable function blocks, and an `ErrorId` sets the error code. No further motion command is accepted until a reset has been done using `MC_Reset_PTO`.
Homing	Applicable when `MC_Home_PTO` controls the axis.
Discrete	Applicable when `MC_MoveRelative_PTO`, `MC_MoveAbsolute_PTO`, or `MC_Halt_PTO` controls the axis.

State	Description
Continuous	Applicable when `MC_MoveVelocity_PTO` controls the axis.
Stopping	Applicable when `MC_Stop_PTO` controls the axis.

NOTE: Function blocks which are not listed in the state diagram do not affect a change of state of the axis.
The entire motion command including acceleration and deceleration ramps cannot exceed 4,294,967,295 pulses. At the maximum frequency of 100 kHz, the acceleration and deceleration ramps are limited to 80 seconds.

Motion Transition Table

The PTO channel can respond to a new command while executing (and before completing) the ongoing command according to the following table:

Command		Next					
		Home	**MoveVelocity**	**MoveRelative**	**MoveAbsolute**	**Halt**	**Stop**
Ongoing	**Standstill**	Allowed	Allowed (1)	Allowed (1)	Allowed (1)	Allowed	Allowed
	Home	Rejected	Rejected	Rejected	Rejected	Rejected	Allowed
	MoveVelocity	Rejected	Allowed	Allowed	Allowed	Allowed	Allowed
	MoveRelative	Rejected	Allowed	Allowed	Allowed	Allowed	Allowed
	MoveAbsolute	Rejected	Allowed	Allowed	Allowed	Allowed	Allowed
	Halt	Rejected	Allowed	Allowed	Allowed	Allowed	Allowed
	Stop	Rejected	Rejected	Rejected	Rejected	Rejected	Rejected

(1) When the axis is at standstill, for the buffer modes `mcAborting/mcBuffered/mcBlendingPrevious`, the move starts immediately.
Allowed the new command begins execution even if the previous command has not completed execution.
Rejected the new command is ignored and results in the declaration of an error.

NOTE: When an error is detected in the motion transition, the axis goes into **ErrorStop** state. The `ErrorId` is set to `InvalidTransition`.

Buffer Mode

Description

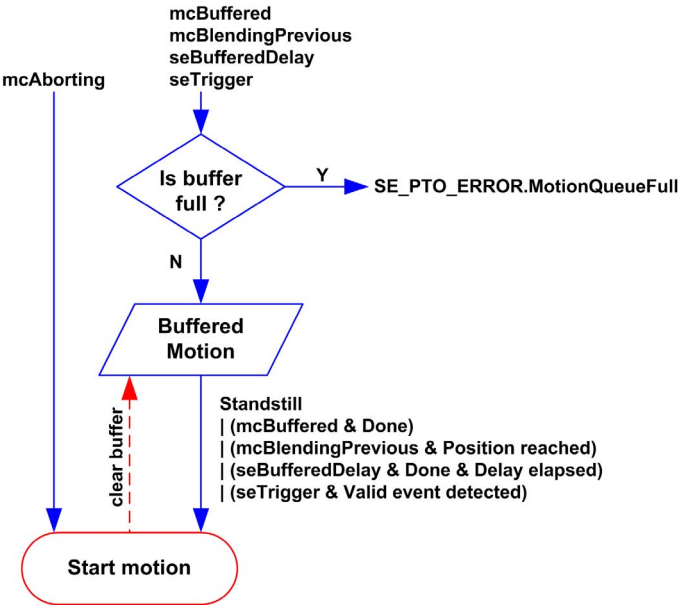
Some of the motion function blocks have an input called `BufferMode`. With this input, the function block can either start immediately, start on probe event, or be buffered.

The available options are defined in the enumeration of type MC_BUFFER_MODE *(see page 73)*:

- An aborting motion (`mcAborting`) starts immediately, aborting any ongoing move, and clearing the motion queue.
- A buffered motion (`mcBuffered`, `mcBlendingPrevious`, `seBufferedDelay`) is queued, that is, appended to any moves currently executing or waiting to execute, and will start when the previous motion is done.
- An event motion (`seTrigger`) is a buffered motion, starting on probe event *(see page 46)*.

Motion Queue Diagram

The figure illustrates the motion queue diagram:



The buffer can contain only one motion function block.

The execution condition of the motion function block present in the buffer is:

- `mcBuffered`: when the current continuous motion is `InVelocity`, resp. when the current discrete motion stops.
- `seBufferedDelay`: when the specified delay has elapsed, from the current continuous motion is `InVelocity`, resp. from the current discrete motion stops.

- `mcBlendingPrevious`: when the position and velocity targets of current function block are reached.
- `seTrigger`: when a valid event is detected on the probe input.

The motion queue is cleared (all buffered motions are deleted):

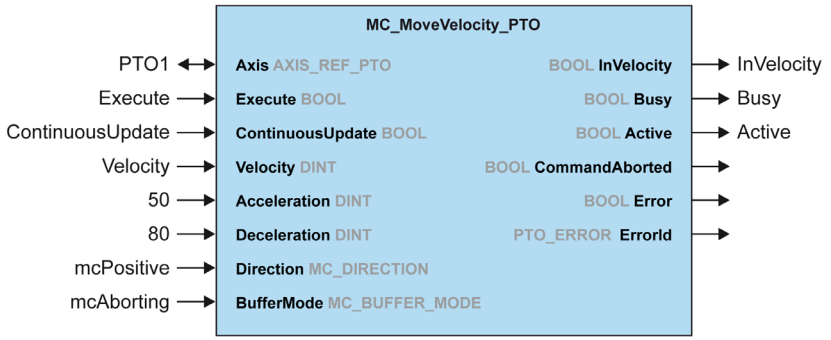
- When an aborting move is triggered (`mcAborting`): `CommandAborted` pin is set on buffered function blocks.
- When a `MC_Stop_PTO` function is executed: `Error` pin is set on cleared buffered function blocks, with `ErrorId`=`StoppingActive` *(see page 78)*.
- When a transition to **ErrorStop** state is detected: `Error` pin is set on buffered function blocks, with `ErrorId`=`ErrorStopActive` *(see page 78)*.

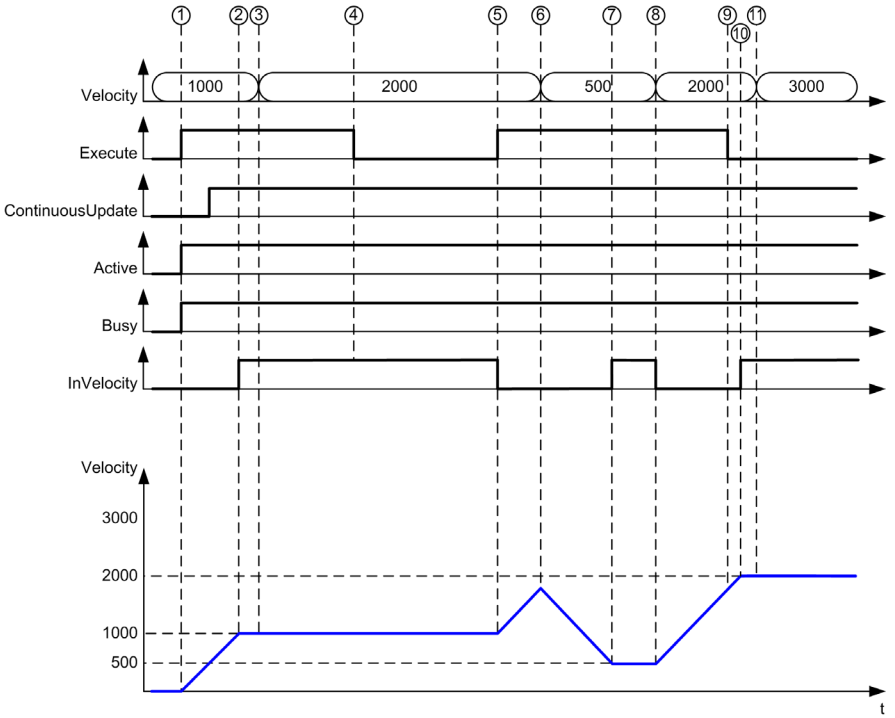
NOTE:

- Only a valid motion can be queued. If the function block execution terminates with the `Error` output set, the move is not queued, any move currently executing is not affected, and the queue is not cleared.
- When the queue is already full, the `Error` output is set on the applicable function block, and `ErrorId` output returns the error `MotionQueueFull` *(see page 78)*.

Timing Diagram Examples

Move Velocity to Move Velocity with mcAborting

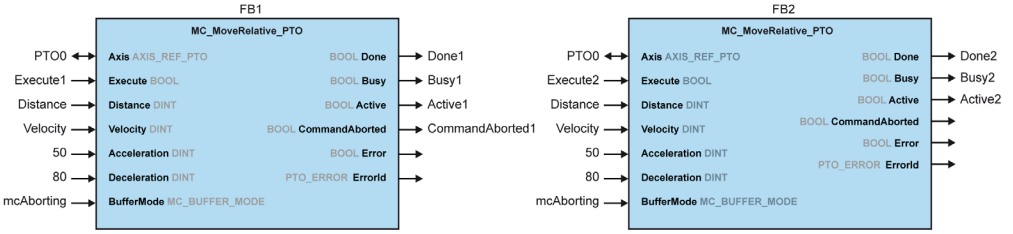


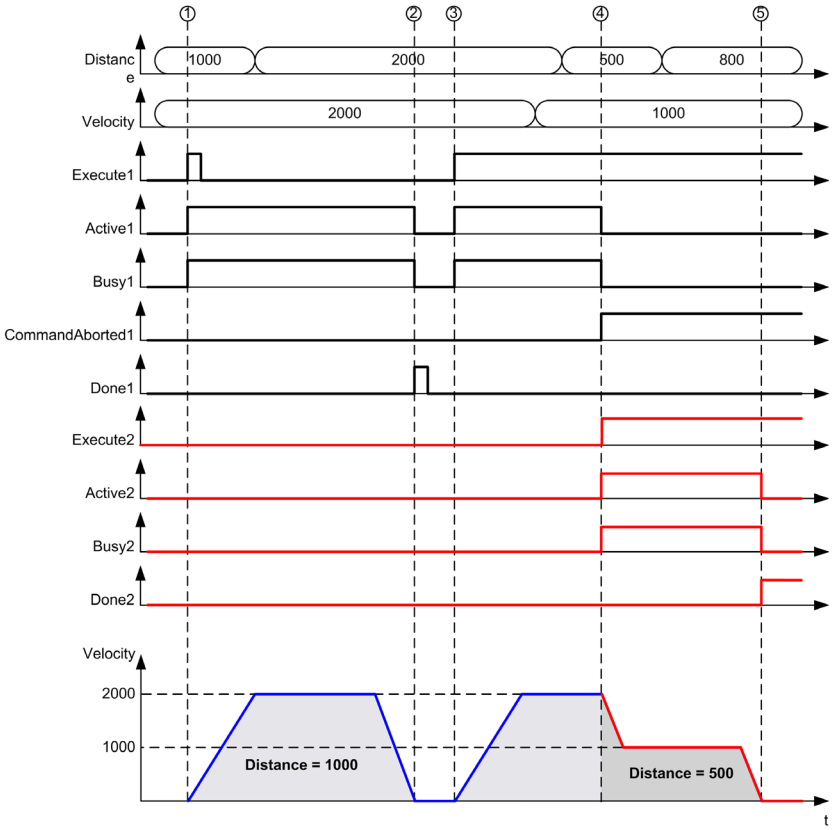


1 `Execute` rising edge: command parameters are latched, movement is started with target `velocity` 1000.
2 Target `velocity` 1000 is reached.
3 `Velocity` parameter changed to 2000: not applied (no rising edge on `Execute` input, and `ContinuousUpdate` was latched with value 0 at start of the movement).

4 `Execute` falling edge: status bits are cleared.
5 `Execute` rising edge: command parameters are latched, movement is started with target `velocity` 2000 and `ContinuousUpdate` active.
6 Velocity parameter changed to 500: applied `ContinuousUpdate` is true). Note: previous target `velocity` 2000 is not reached.
7 Target `velocity` 500 is reached.
8 `Velocity` parameter changed to 2000: applied `ContinuousUpdate` is true).
9 `Execute` falling edge: status bits are cleared.
10 Target `velocity` 2000 is reached, `InVelocity` is set for 1 cycle (`Execute` pin is reset).
11 `Velocity` parameter changed to 3000: not applied (movement is still active, but no longer busy).

Move Relative to Move Relative with mcAborting





1 FB1 `Execute` rising edge: command parameters are latched, movement is started with target `velocity` 2000 and `distance` 1000.

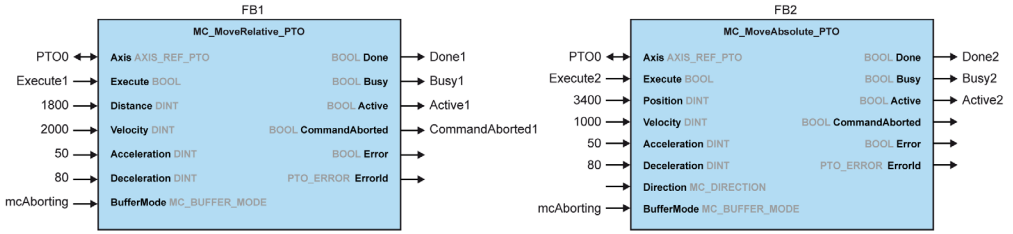
2 Movement ends: distance traveled is 1000.

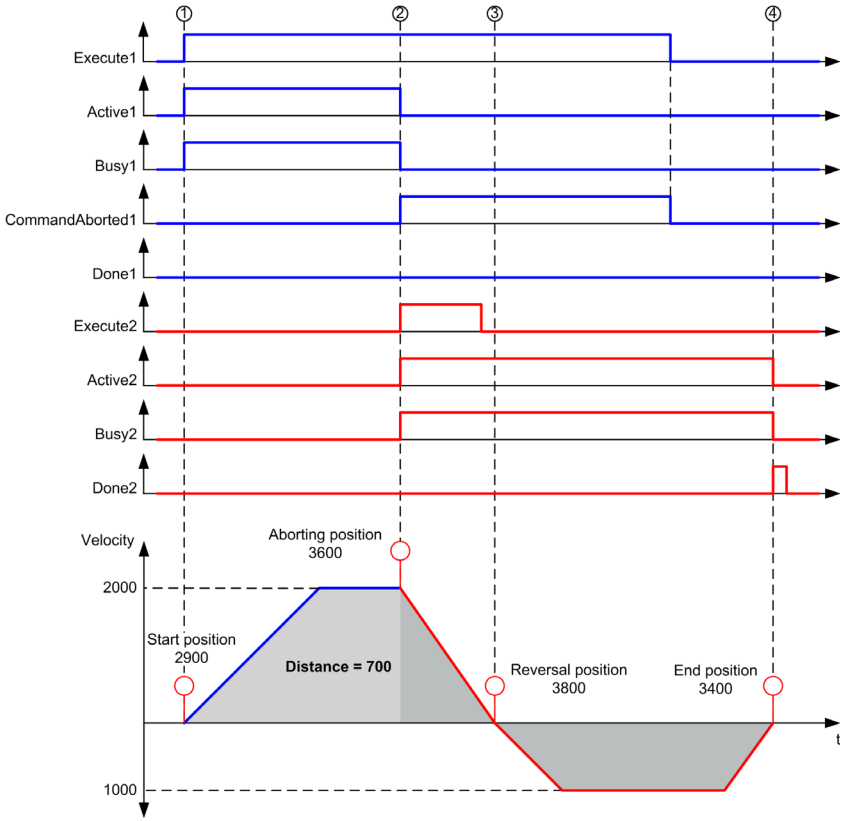
3 FB1 `Execute` rising edge: command parameters are latched, movement is started with target `velocity` 2000 and `distance` 2000.

4 FB2 `Execute` rising edge: command parameters are latched, movement is started with target `velocity` 1000 and `distance` 500. Note: FB1 is aborted.

5 Movement ends.

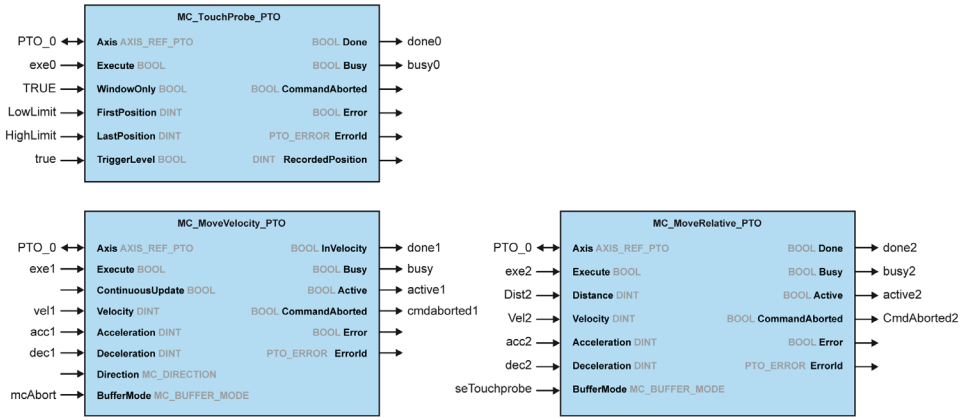
Move Relative to Move Absolute with mcAborting

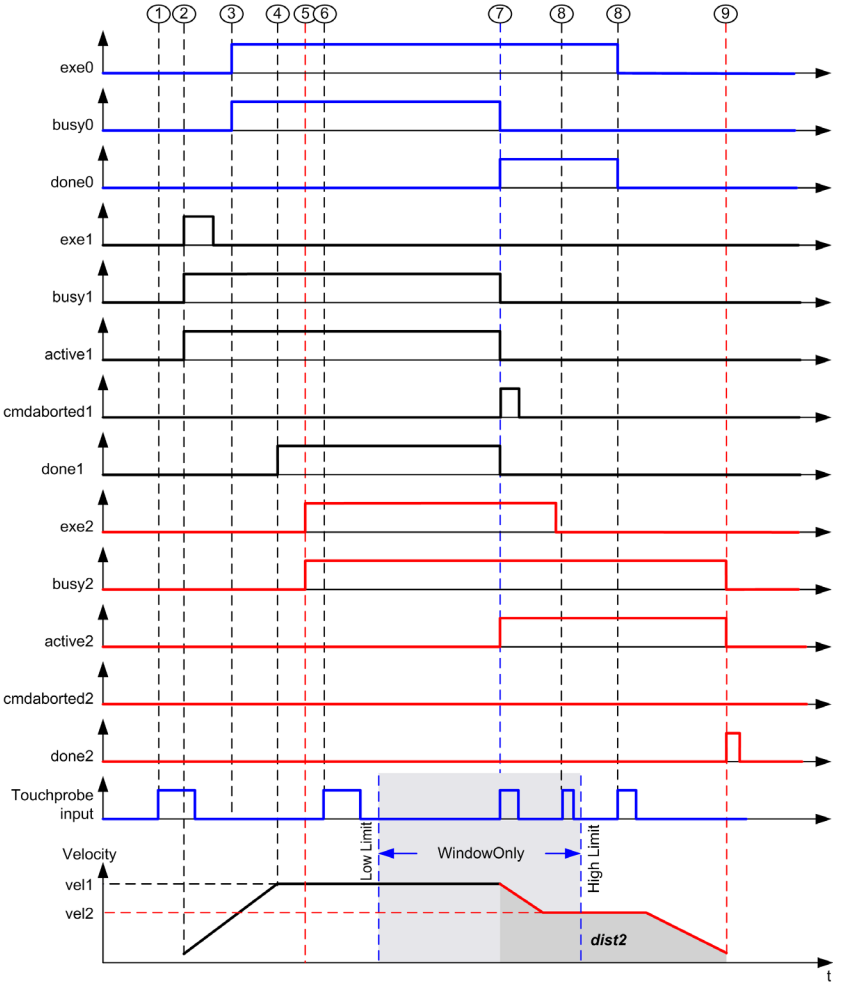




1. FB1 `Execute` rising edge: command parameters are latched, movement is started with target `velocity` 2000 and `distance` 1800.
2. FB2 `Execute` rising edge: command parameters are latched, FB1 is aborted, and movement continues with target `velocity` 1000 and target`position` 3400. Automatic direction management: direction reversal is needed to reach target position, move to stop at `deceleration` of FB2.
3. Velocity 0, direction reversal, movement resumes with target `velocity` 1000 and target `position` 3400.
4. Movement ends: target position 3400 reached.

Move Velocity to Move Relative with seTrigger





1 `MC_TouchProbe_PTO` not executed yet: probe input is not active.
2 `MC_MoveVelocity_PTO` `Execute` rising edge: command parameters are latched, movement is started with target `velocity` vel1.
3 `MC_TouchProbe_PTO` `Execute` rising edge: probe input is active.
4 vel1 is reached.
5 `MC_MoveRelative_PTO` `Execute` rising edge: command parameters are latched, waiting for probe event to start.
6 Probe event outside of enable windows: event is ignored.
7 A valid event is detected. `MC_MoveRelative_PTO` aborts `MC_MoveVelocity_PTO`, and probe input is deactivated.
8 Following events are ignored.
9 Movement ends.

Section 6.2
MC_Power_PTO Function Block

Overview

This section describes the `MC_Power_PTO` function block.

What Is in This Section?

This section contains the following topics:

Description

Overview

The `MC_Power_PTO` function block allows enabling power and control to the axis, switching the axis state from **Disabled** to **Standstill**. No motion function block is allowed to affect the axis until the `MC_Power_PTO.Status` bit is `TRUE`.

Disabling power (`MC_Power_PTO.Enable` = `FALSE`) switches the axis:

- from **Standstill**, back to **Disabled** state.
- from any ongoing move, to **ErrorStop**, and then **Disabled** when the error is reset.

If `DriveReady` input is reset, the axis state switches to **ErrorStop**.

MC_Power_PTO Function Block

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 195)*.

Input Variables

This table describes the input variables:

Input	Type	Initial Value	Description
`Axis`	AXIS_REF_PTO	-	Name of the axis (instance) for which the function block is to be executed. In the devices tree, the name is declared under the controller configuration.
`Enable`	BOOL	FALSE	When TRUE, the function block is executed. The values of the function block inputs can be modified and the outputs updated continuously. When FALSE, terminates the function block execution and resets its outputs.
`DriveReady`(1)	BOOL	FALSE	Drive ready information from the drive. Must be TRUE when the drive is ready to start executing motion. If the drive signal is connected to the controller, use the appropriate %Ix input. If the drive does not provide this signal, you can select the value TRUE for this input.
`LimP`(1)	BOOL	TRUE	Hardware limit switch information, in positive direction. It must be FALSE when the hardware limit switch is reached. If the hardware limit switch signal is connected to the controller, use the appropriate %Ix input. If this signal is not available, you can leave this input unused or set to TRUE.
`LimN`(1)	BOOL	TRUE	Hardware limit switch information, in negative direction. It must be FALSE when the hardware limit switch is reached. If the hardware limit switch signal is connected to the controller, use the appropriate %Ix. If this signal is not available, you can leave this input unused or set to TRUE.

(1) `DriveReady`, `LimP`, and `LimN` are read at the task cycle time.

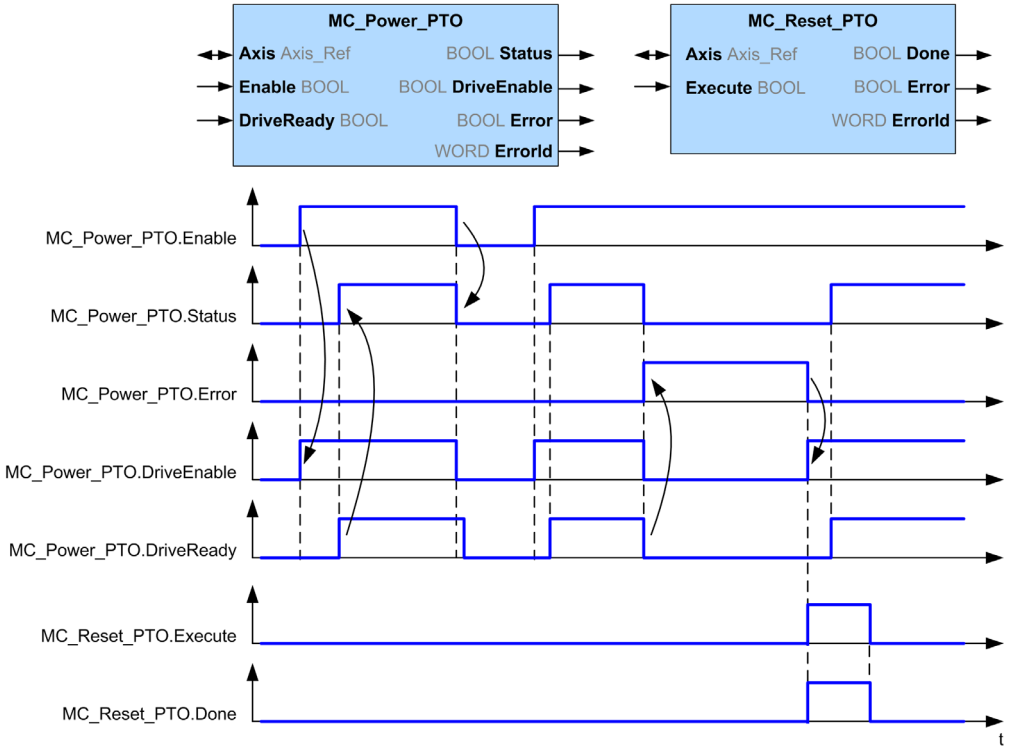
Output Variables

This table describes the output variables:

Output	Type	Initial Value	Description
`Status`	BOOL	FALSE	When TRUE, power is enabled, motion commands are possible.
`DriveEnable`	BOOL	FALSE	Enables the drive to accept commands. If the drive does not use this signal, you can leave this output unused.
`Error`	BOOL	FALSE	If TRUE, indicates that an error was detected. Function block execution is finished.
`ErrorId`	PTO_ERROR	`PTO_ERROR.NoError`	When `Error` is TRUE: code of the error detected *(see page 78)*.

Timing Diagram Example

The diagram illustrates the function block operation:



Section 6.3
MC_MoveVelocity_PTO Function Block

Overview

This section describes the `MC_MoveVelocity_PTO` function block.

What Is in This Section?

This section contains the following topics:

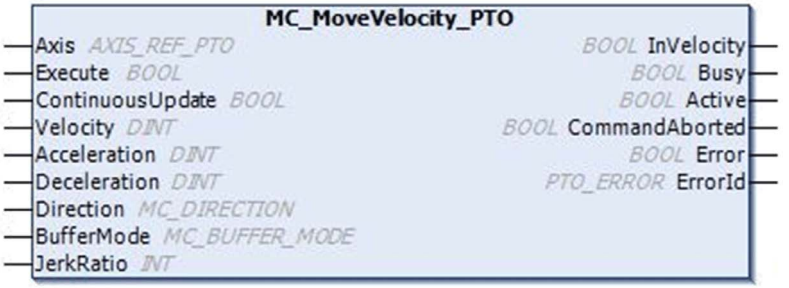
Description

Overview

This function causes the specified axis to move at the specified speed, and transfers the axis to the state **Continuous**. This continuous movement is maintained until a software limit is reached, an aborting move is triggered, or a transition to **ErrorStop** state is detected.

MC_MoveVelocity_PTO Function Block

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 195)*.

Input Variables

This table describes the input variables:

Input	Type	Initial Value	Description
`Axis`	AXIS_REF_PTO	-	Name of the axis (instance) for which the function block is to be executed. In the devices tree, the name is declared in the controller configuration.
`Execute`	BOOL	FALSE	On rising edge, starts the function block execution. On falling edge, resets the outputs of the function block when its execution terminates. Later changes in the function block input parameters do not affect the ongoing command, unless the input `ContinuousUpdate` is used. If a second rising edge is detected during the execution of the function block, the ongoing execution is aborted and the function block is restarted with the values of the parameters at the time.

Input	Type	Initial Value	Description
ContinuousUpdate	BOOL	FALSE	At TRUE, makes the function block use the values of the input variables (Velocity, Acceleration, Deceleration, and Direction), and apply it to the ongoing command regardless of their original values. The impact of the input ContinuousUpdate begins when the function block is triggered by a rising edge on the Execute pin, and ends as soon as the function block is no longer Busy or the input ContinuousUpdate is set to FALSE.
Velocity	DINT	0	Target velocity in Hz, not necessarily reached. Range: 0...MaxVelocityAppl *(see page 77)*
Acceleration	DINT	0	Acceleration in Hz/ms or in ms (according to configuration). Range (Hz/ms): 1...MaxAccelerationAppl *(see page 77)* Range (ms): MaxAccelerationAppl *(see page 77)*...100,000
Deceleration	DINT	0	Deceleration in Hz/ms or in ms (according to configuration). Range (Hz/ms): 1...MaxDecelerationAppl *(see page 77)* Range (ms): MaxDecelerationAppl *(see page 77)*...100,000
Direction	MC_ DIRECTION	mcPositiveDirection	Direction of the movement *(see page 75)*.
BufferMode	MC_BUFFER_ MODE	mcAborting	Transition mode from ongoing move *(see page 73)*.
JerkRatio	INT	0	Percentage of acceleration / deceleration adjustment used to create the S-curve profile *(see page 45)*. Range : 0…100

Output Variables

This table describes the output variables:

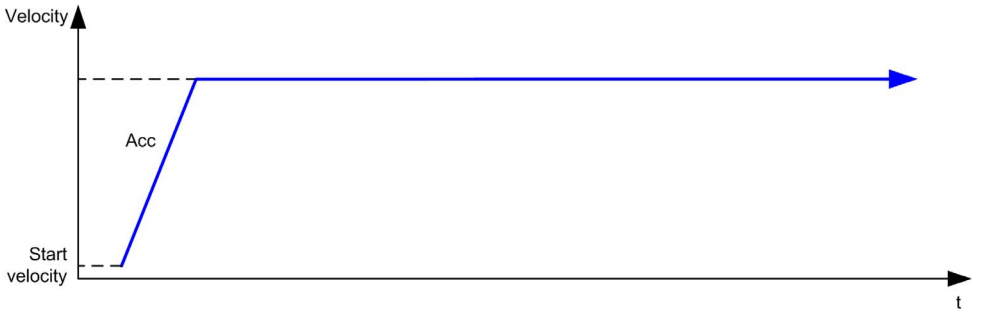
Output	Type	Initial Value	Description
`InVelocity`	BOOL	FALSE	If TRUE, indicates that the target velocity is reached.
`Busy`	BOOL	FALSE	If TRUE, indicates that the function block execution is in progress.
`Active`	BOOL	FALSE	The function block controls the `Axis`. Only one function block at a time can set `Active` TRUE for a defined `Axis`.
`CommandAborted`	BOOL	FALSE	Function block execution is finished, by aborting due to another move command or an error detected.
`Error`	BOOL	FALSE	If TRUE, indicates that an error was detected. Function block execution is finished.
`ErrorId`	PTO_ERROR	`PTO_ERROR.NoError`	When `Error` is TRUE: code of the error detected *(see page 78)*.

NOTE:

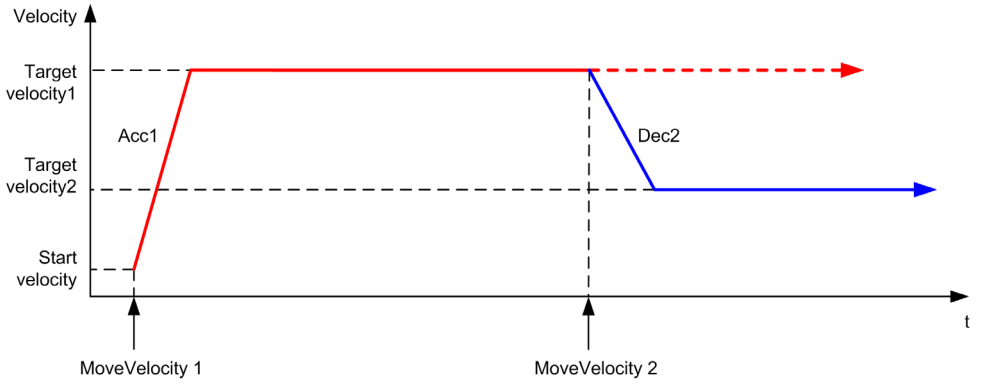
- To stop the motion, the function block has to be interrupted by another function block issuing a new command.
- If a motion is ongoing, and the direction is reversed, first the motion is halted with the deceleration of the `MC_MoveVelocity_PTO` function block, and then the motion resumes backwards.
- The acceleration/deceleration duration of the segment block must not exceed 80 seconds.

Timing Diagram Example

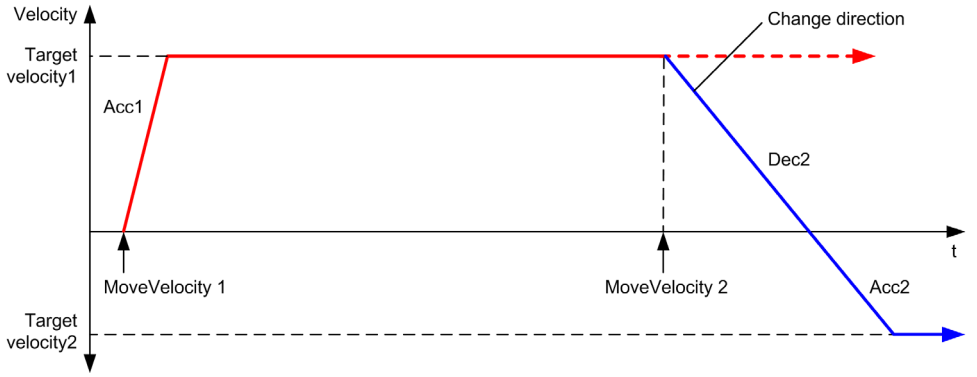
The diagram illustrates a simple profile from **Standstill** state:



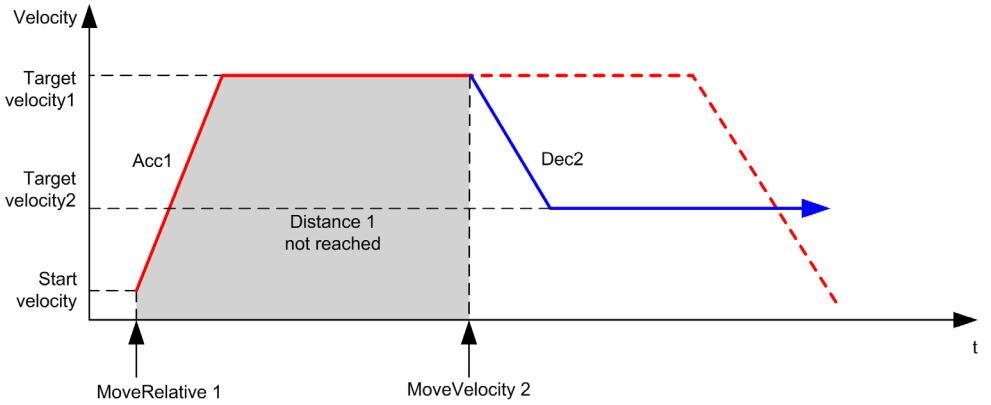
The diagram illustrates a complex profile from **Continuous** state:



The diagram illustrates a complex profile from **Continuous** state with change of direction:



The diagram illustrates a complex profile from **Discrete** state:



Section 6.4
MC_MoveRelative_PTO Function Block

Overview

This section describes the `MC_MoveRelative_PTO` function block.

What Is in This Section?

This section contains the following topics:

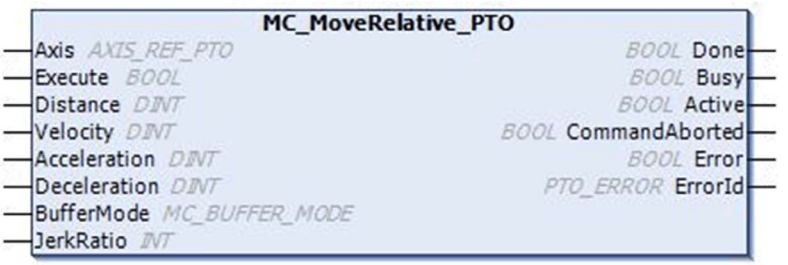
Description

Overview

This function causes the specified axis to move of an incremental distance, and transfers the axis to the state **Discrete**. The target position is referenced from the current position at execution time, incremented by a distance.

MC_MoveRelative_PTO Function Block

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 195)*.

Input Variables

This table describes the input variables:

Input	Type	Initial Value	Description
`Axis`	AXIS_REF_PTO	-	Name of the axis (instance) for which the function block is to be executed. In the devices tree, the name is declared in the controller configuration.
`Execute`	BOOL	FALSE	On rising edge, starts the function block execution. On falling edge, resets the outputs of the function block when its execution terminates.
`Distance`	DINT	0	Relative distance of the motion in number of pulses. The sign specifies the direction.
`Velocity`	DINT	0	Target velocity in Hz, not necessarily reached. Range: 1...`MaxVelocityAppl` *(see page 77)*
`Acceleration`	DINT	0	Acceleration in Hz/ms or in ms (according to configuration). Range (Hz/ms): 1...`MaxAccelerationAppl` *(see page 77)* Range (ms): `MaxAccelerationAppl` *(see page 77)*...100,000
`Deceleration`	DINT	0	Deceleration in Hz/ms or in ms (according to configuration). Range (Hz/ms): 1...`MaxDecelerationAppl` *(see page 77)* Range (ms): `MaxDecelerationAppl` *(see page 77)*...100,000

Input	Type	Initial Value	Description
BufferMode	MC_BUFFER_MODE	mcAborting	Transition mode from ongoing move *(see page 73)*.
JerkRatio	INT	0	Percentage of acceleration / deceleration adjustment used to create the S-curve profile *(see page 45)*. Range : 0…100

Output Variables

This table describes the output variables:

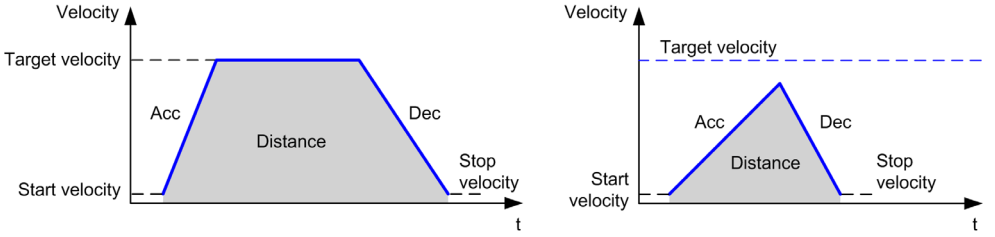
Output	Type	Initial Value	Description
Done	BOOL	FALSE	If TRUE, indicates that the function block execution is finished with no error detected.
Busy	BOOL	FALSE	If TRUE, indicates that the function block execution is in progress.
Active	BOOL	FALSE	The function block controls the Axis. Only one function block at a time can set Active TRUE for a defined Axis.
CommandAborted	BOOL	FALSE	Function block execution is finished, by aborting due to another move command or an error detected.
Error	BOOL	FALSE	If TRUE, indicates that an error was detected. Function block execution is finished.
ErrorId	PTO_ERROR	PTO_ERROR.NoError	When Error is TRUE: code of the error detected *(see page 78)*.

NOTE:

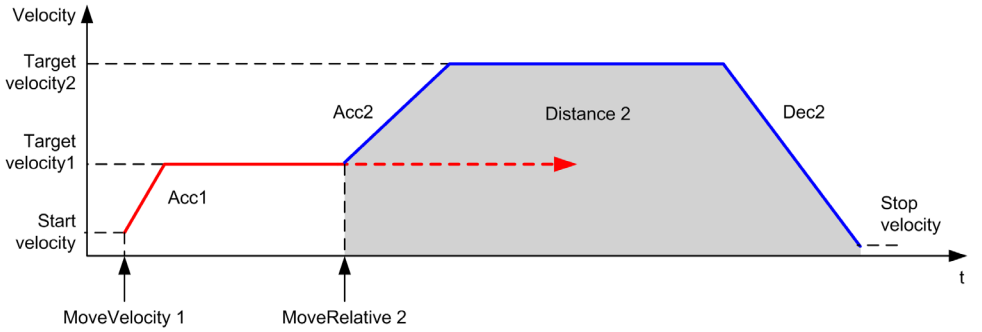
- The function block completes with velocity zero if no further blocks are pending.
- If the distance is too short for the target velocity to be reached, the movement profile is triangular, rather than trapezoidal.
- If a motion is ongoing, and the commanded distance is exceeded due to the motion parameters, the direction reversal is automatically managed: the motion is first halted with the deceleration of the MC_MoveRelative_PTO function block, and then the motion resumes backwards.
- The acceleration/deceleration duration of the segment block must not exceed 80 seconds.

Timing Diagram Example

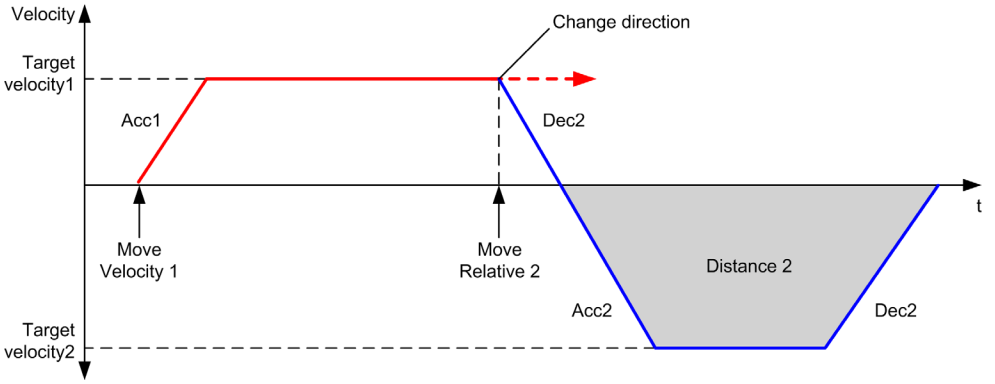
The diagram illustrates a simple profile from **Standstill** state:



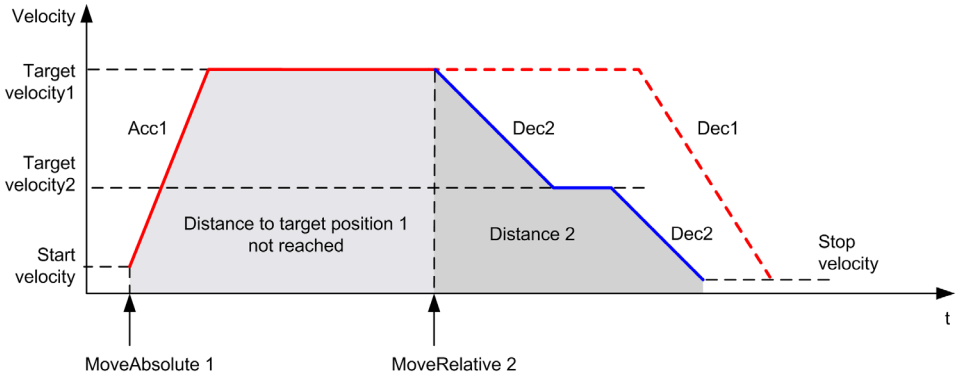
The diagram illustrates a complex profile from **Continuous** state:



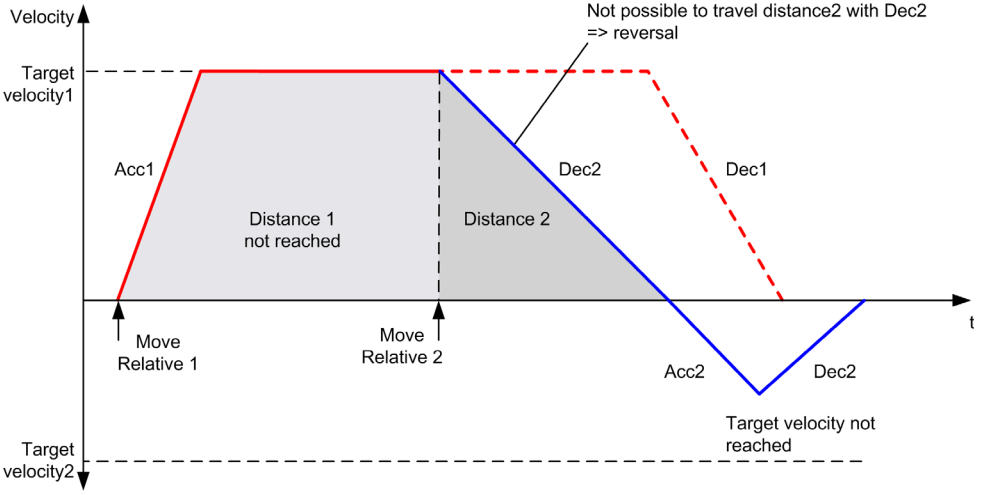
The diagram illustrates a complex profile from **Continuous** state with change of direction:



The diagram illustrates a complex profile from **Discrete** state:



The diagram illustrates a complex profile from **Discrete** state with change of direction:



Section 6.5
MC_MoveAbsolute_PTO Function Block

Overview

This section describes the `MC_MoveAbsolute_PTO` function block.

What Is in This Section?

This section contains the following topics:

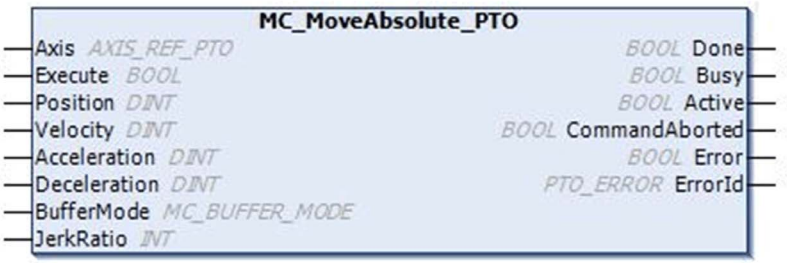
Description

Overview

This function causes the specified axis to move towards a given position at the specified speed, and transfers the axis to the state **Discrete**. To use the `MC_MoveAbsolute_PTO` function block, you must first home the axis. If not the function block will terminate in error (`Error` set to 1 and `ErrorId` set to `InvalidAbsolute`).

MC_MoveAbsolute_PTO Function Block

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 195)*.

Input Variables

This table describes the input variables:

Input	Type	Initial Value	Description
`Axis`	AXIS_REF_PTO	-	Name of the axis (instance) for which the function block is to be executed. In the devices tree, the name is declared in the controller configuration.
`Execute`	BOOL	FALSE	On rising edge, starts the function block execution. On falling edge, resets the outputs of the function block when its execution terminates.
`Position`	DINT	0	Target absolute position.
`Velocity`	DINT	0	Target velocity in Hz, not necessarily reached. Range: 1...`MaxVelocityAppl` *(see page 77)*
`Acceleration`	DINT	0	Acceleration in Hz/ms or in ms (according to configuration). Range (Hz/ms): 1...`MaxAccelerationAppl` *(see page 77)* Range (ms): `MaxAccelerationAppl` *(see page 77)*...100,000
`Deceleration`	DINT	0	Deceleration in Hz/ms or in ms (according to configuration). Range (Hz/ms): 1...`MaxDecelerationAppl` *(see page 77)* Range (ms): `MaxDecelerationAppl` *(see page 77)*...100,000

Input	Type	Initial Value	Description
`Direction`	MC_DIRECTION	`mcPositive-Direction`	Direction of the movement.
`BufferMode`	MC_BUFFER_MODE	`mcAborting`	Transition mode from ongoing move *(see page 73)*.
`JerkRatio`	INT	0	Percentage of acceleration / deceleration adjustment used to create the S-curve profile *(see page 45)*. Range : 0…100

Output Variables

This table describes the output variables:

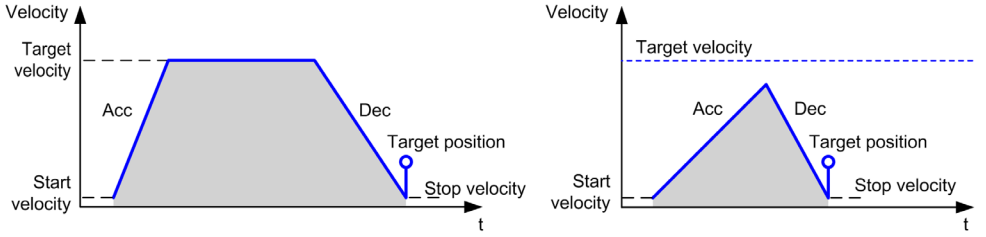
Output	Type	Initial Value	Description
`Done`	BOOL	FALSE	If TRUE, indicates that the function block execution is finished with no error detected.
`Busy`	BOOL	FALSE	If TRUE, indicates that the function block execution is in progress.
`Active`	BOOL	FALSE	The function block controls the `Axis`. Only one function block at a time can set `Active` TRUE for a defined `Axis`.
`CommandAborted`	BOOL	FALSE	Function block execution is finished, by aborting due to another move command or an error detected.
`Error`	BOOL	FALSE	If TRUE, indicates that an error was detected. Function block execution is finished.
`ErrorId`	PTO_ERROR	`PTO_ERROR.NoError`	When `Error` is TRUE: code of the error detected *(see page 78)*.

NOTE:

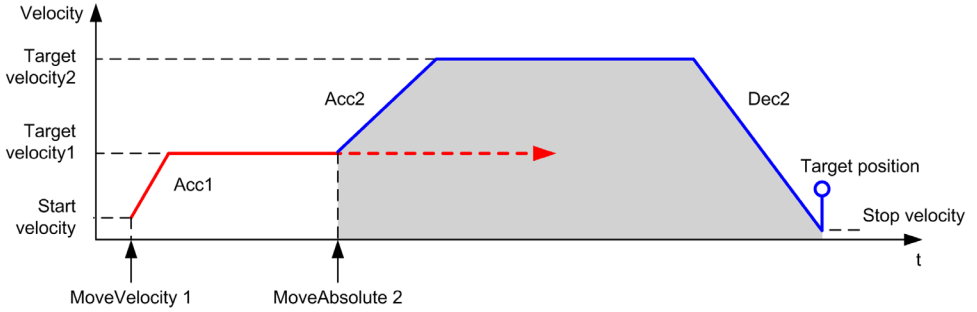
- The function block completes with velocity zero if no further blocks are pending.
- The motion direction is automatically set, according to the present and targeted positions.
- If the distance is too short for the target velocity to be reached, the movement profile is triangular, rather than trapezoidal.
- If the position cannot be reached with the ongoing direction, the direction reversal is automatically managed. If a motion is ongoing, it is first halted with the deceleration of the `MC_MoveAbsolute_PTO` function block, and then the motion resumes backwards.
- The acceleration/deceleration duration of the segment block must not exceed 80 seconds.

Timing Diagram Example

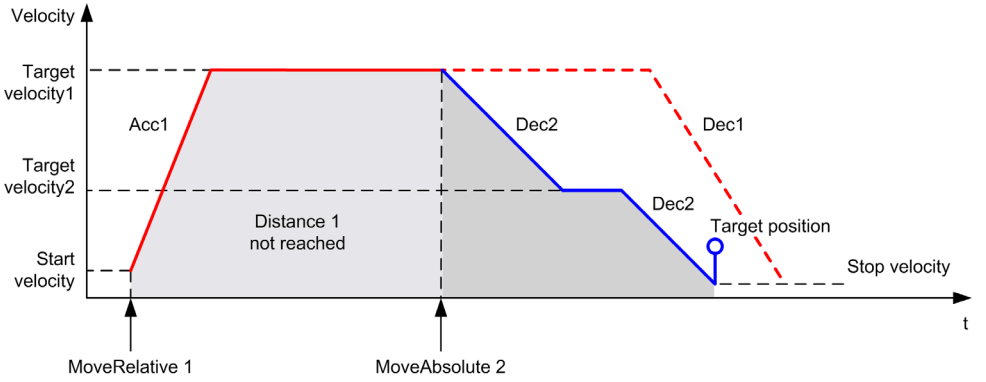
The diagram illustrates a simple profile from **Standstill** state:



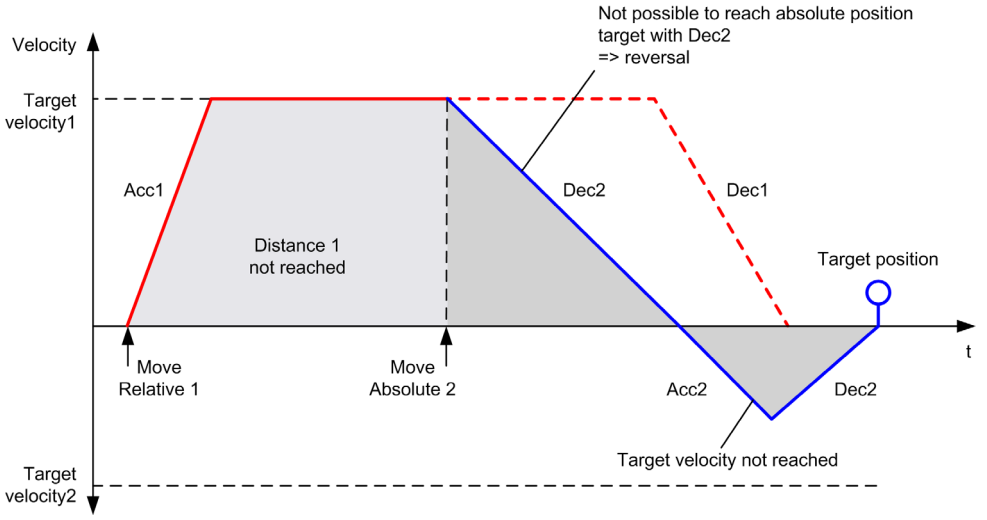
The diagram illustrates a complex profile from **Continuous** state:



The diagram illustrates a complex profile from **Discrete** state:



The diagram illustrates a complex profile from **Discrete** state with change of direction:



Section 6.6
MC_Home_PTO Function Block

Overview

This section describes the `MC_Home_PTO` function block.

What Is in This Section?

This section contains the following topics:

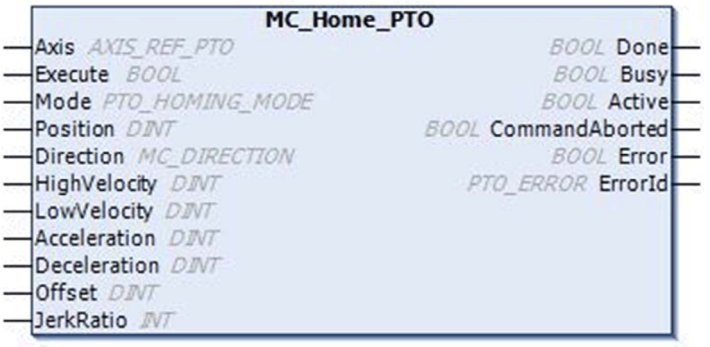
Description

Overview

This function block commands the axis to perform the sequence defining the reference absolute position, and transfers the axis to the state **Homing**. The details of this sequence depend on homing configuration parameter settings.

MC_Home_PTO Function Block

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 195)*.

Input Variables

This table describes the input variables:

Input	Type	Initial Value	Description
`Axis`	AXIS_REF_PTO	-	Name of the axis (instance) for which the function block is to be executed. In the devices tree, the name is declared in the controller configuration.
`Execute`	BOOL	FALSE	On rising edge, starts the function block execution. On falling edge, resets the outputs of the function block when its execution terminates.
`Mode`	PTO_HOMING_MODE	`mcPositionSetting`	Predefined home mode *(see page 76)* type.
`Position`	DINT	0	Position value is set as absolute position at the reference point switch detection, when the homing has been successfully executed.
`Direction`	MC_DIRECTION	`mcPositiveDirection`	Starting direction. For Homing, only `mcPositiveDirection` and `mcNegativeDirection` are valid.
`HighVelocity`	DINT	0	Target homing velocity for searching the limit or reference switch. Range Hz: 1...`MaxVelocityAppl` *(see page 77)*

Input	Type	Initial Value	Description
LowVelocity	DINT	0	Target homing velocity for searching the reference switch or index signal. The movement stop when switching point is detected. Range Hz: 1...HighVelocity
Acceleration	DINT	0	Acceleration in Hz/ms or in ms (according to configuration). Range (Hz/ms): 1...MaxAccelerationAppl *(see page 77)* Range (ms): MaxAccelerationAppl *(see page 77)*...100,000
Deceleration	DINT	0	Deceleration in Hz/ms or in ms (according to configuration). Range (Hz/ms): 1...MaxDecelerationAppl *(see page 77)* Range (ms): MaxDecelerationAppl *(see page 77)*...100,000
Offset	DINT	0	Distance from origin point. When the origin point is reached, the motion resumes until the distance is covered. Direction depends on the sign (Home offset *(see page 70)*). Range: -2,147,483,648...2,147,483,647
JerkRatio	INT	0	Percentage of acceleration / deceleration adjustment used to create the S-curve profile *(see page 45)*. Range : 0…100

Output Variables

This table describes the output variables:

Output	Type	Initial Value	Description
`Done`	BOOL	FALSE	If TRUE, indicates that the function block execution is finished with no error detected .
`Busy`	BOOL	FALSE	If TRUE, indicates that the function block execution is in progress.
`Active`	BOOL	FALSE	The function block controls the `Axis`. Only one function block at a time can set `Active` TRUE for a defined `Axis`.
`CommandAborted`	BOOL	FALSE	Function block execution is finished, by aborting due to another move command or an error detected .
`Error`	BOOL	FALSE	If TRUE, indicates that an error was detected. Function block execution is finished.
`ErrorId`	PTO_ERROR	`PTO_ERROR.NoError`	When `Error` is TRUE: code of the error detected *(see page 78)*.

NOTE: The acceleration/deceleration duration of the segment block must not exceed 80 seconds.

Timing Diagram Example

Home modes *(see page 55)*

Section 6.7
MC_SetPosition_PTO Function Block

Overview

This section describes the `MC_SetPosition_PTO` function block.

What Is in This Section?

This section contains the following topics:

Description

Overview

This function block modifies the coordinates of the actual position of the axis without any physical movement. This function block can only be used while the axis is a **Standstill** state.

MC_SetPosition_PTO Function Block

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 195)*.

Input Variables

This table describes the input variables:

Input	Type	Initial Value	Description
`Axis`	AXIS_REF_PTO	-	Name of the axis (instance) for which the function block is to be executed. In the devices tree, the name is declared in the controller configuration.
`Execute`	BOOL	FALSE	On rising edge, starts the function block execution. On falling edge, resets the outputs of the function block when its execution terminates.
`Position`	DINT	0	New value of absolute position of the `Axis`.

Output Variables

This table describes the output variables:

Output	Type	Initial Value	Description
`Done`	BOOL	FALSE	If TRUE, indicates that the function block execution is finished with no error detected .
`Error`	BOOL	FALSE	If TRUE, indicates that an error was detected. Function block execution is finished.
`ErrorId`	PTO_ERROR	`PTO_ERROR.NoError`	When `Error` is TRUE: type of the error detected *(see page 78)*.

Section 6.8
MC_Stop_PTO Function Block

Overview

This section describes the `MC_Stop_PTO` function block.

What Is in This Section?

This section contains the following topics:

Description

Overview

This function block commands a controlled motion stop and transfers the axis to the state **Stopping**. It aborts any ongoing move execution. While the axis is in state **Stopping**, no other function block can perform any motion on the same axis. This function block is primarily intended for exception situations, or fast stop functionality.

MC_Stop_PTO Function Block

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 195)*.

Input Variables

This table describes the input variables:

Input	Type	Initial Value	Description
`Axis`	AXIS_REF_PTO	-	Name of the axis (instance) for which the function block is to be executed. In the devices tree, the name is declared in the controller configuration.
`Execute`	BOOL	FALSE	On rising edge, starts the function block execution. On falling edge, resets the outputs of the function block when its execution terminates.
`Deceleration`	DINT	20	Deceleration in Hz/ms or in ms (according to configuration). Range (Hz/ms): 1...`MaxDecelerationAppl` *(see page 77)* Range (ms): `MaxDecelerationAppl` *(see page 77)*...100,000
`JerkRatio`	INT	0	Percentage of acceleration / deceleration adjustment used to create the S-curve profile *(see page 45)*. Range : 0…100

Output Variables

This table describes the output variables:

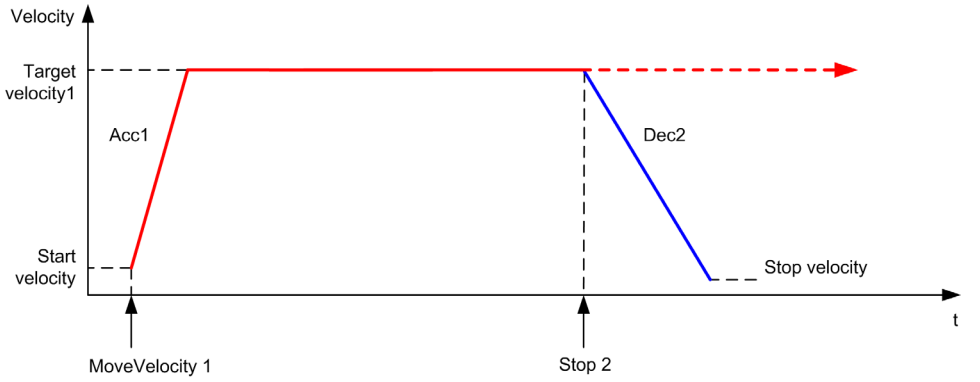
Output	Type	Initial Value	Description
`Done`	BOOL	FALSE	If TRUE, indicates that the function block execution is finished with no error detected .
`Busy`	BOOL	FALSE	If TRUE, indicates that the function block execution is in progress.
`CommandAborted`	BOOL	FALSE	Function block execution is finished, by aborting due to another move command or an error detected .
`Error`	BOOL	FALSE	If TRUE, indicates that an error was detected. Function block execution is finished.
`ErrorId`	PTO_ERROR	`PTO_ERROR.NoError`	When `Error` is TRUE: type of the error detected *(see page 78)*.

NOTE:

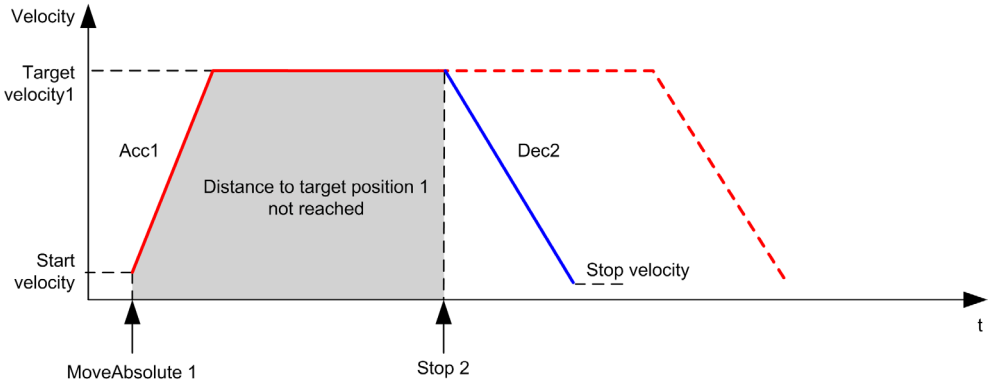
- Calling this function block in state **Standstill** changes the state to **Stopping**, and back to **Standstill** when `Execute` is FALSE.
- The state **Stopping** is kept as long as the input `Execute` is true.
- The`Done` output is set when the stop ramp is finished.
- If `Deceleration` = 0, the fast stop deceleration is used.
- The function block completes with velocity zero.
- The deceleration duration of the segment block must not exceed 80 seconds.

Timing Diagram Example

The diagram illustrates a simple profile from **Continuous** state:



The diagram illustrates a simple profile from **Discrete** state:



Section 6.9
MC_Halt_PTO Function Block

Overview

This section describes the `MC_Halt_PTO` function block.

What Is in This Section?

This section contains the following topics:

Description

Overview

This function block commands a controlled motion stop until the velocity is zero, and transfers the axis to the state **Discrete**. With the `Done` output set, the state is transferred to **Standstill**.

MC_Halt_PTO Function Block

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 195)*.

Input Variables

This table describes the input variables:

Input	Type	Initial Value	Description
`Axis`	AXIS_REF_PTO	-	Name of the axis (instance) for which the function block is to be executed. In the devices tree, the name is declared in the controller configuration.
`Execute`	BOOL	FALSE	On rising edge, starts the function block execution. On falling edge, resets the outputs of the function block when its execution terminates.
`Deceleration`	DINT	20	Deceleration in Hz/ms or in ms (according to configuration). Range (Hz/ms): 1...`MaxDecelerationAppl` *(see page 77)* Range (ms): `MaxDecelerationAppl` *(see page 77)*...100,000
`BufferMode`	MC_BUFFER_MODE	`mcAborting`	Transition mode from ongoing move *(see page 73)*.
`JerkRatio`	INT	0	Percentage of acceleration / deceleration adjustment used to create the S-curve profile *(see page 45)*. Range : 0…100

Output Variables

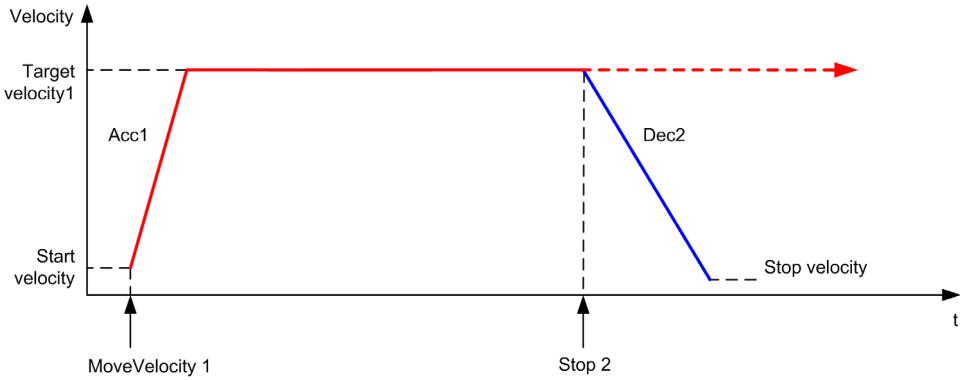
This table describes the output variables:

Output	Type	Initial Value	Description
`Done`	BOOL	FALSE	If TRUE, indicates that the function block execution is finished with no error detected .
`Busy`	BOOL	FALSE	If TRUE, indicates that the function block execution is in progress.
`Active`	BOOL	FALSE	The function block controls the `Axis`. Only one function block at a time can set `Active` TRUE for a defined `Axis`.
`CommandAborted`	BOOL	FALSE	Function block execution is finished, by aborting due to another move command or an error detected .
`Error`	BOOL	FALSE	If TRUE, indicates that an error was detected. Function block execution is finished.
`ErrorId`	PTO_ERROR	`PTO_ERROR.NoError`	When `Error` is TRUE: type of the error detected *(see page 78)*.

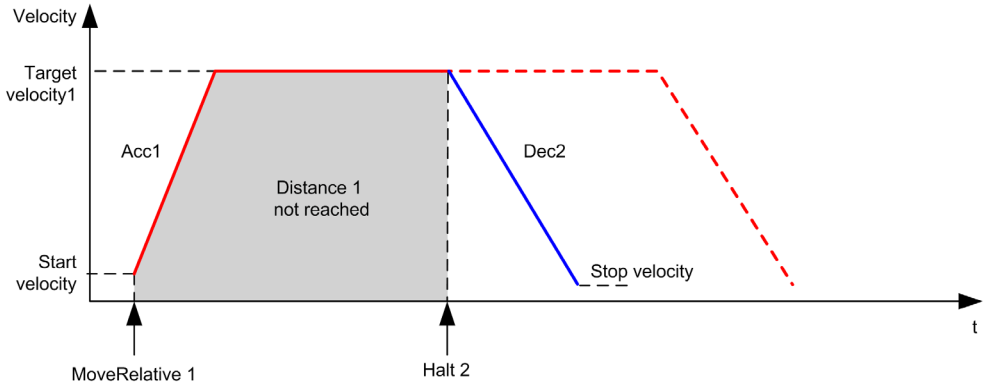
NOTE: The function block completes with velocity zero.

Timing Diagram Example

The diagram illustrates a simple profile from **Continuous** state:



The diagram illustrates a simple profile from **Discrete** state:



Section 6.10
Adding a Motion Function Block

Adding a Motion Function Block

Procedure

Follow these steps to add and create the instance of a motion function block:

Step	Action
1	Add a POU *(see SoMachine, Programming Manual,)* in the **Applications tree**.
2	Select the **Libraries** tab in the **Software Catalog** and click **Libraries**. Select **Controller** → **M241** → **M241 PTOPWM** → **PTO** → **Motion** → **MC_xxxxxx_PTO** in the list, drag-and-drop the item onto the **POU** window.
3	Create the function block instance by clicking: 
4	Associate the input/output variables *(see page 81)* of the function block.

Chapter 7
Administrative Function Blocks

Overview

This chapter describes the administrative function blocks.

Administrative function blocks do not influence the state diagram *(see page 83)*.

What Is in This Chapter?

This chapter contains the following sections:

Section 7.1
Status Function Blocks

Overview

This section describes the status function blocks.

What Is in This Section?

This section contains the following topics:

MC_ReadActualVelocity_PTO Function Block

Function Description

This function block returns the value of the actual velocity of the axis.

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 195)*.

Input Variables

This table describes the input variables:

Input	Type	Initial Value	Description
`Axis`	AXIS_REF_PTO	-	Name of the axis (instance) for which the function block is to be executed. In the devices tree, the name is declared in the controller configuration,.
`Enable`	BOOL	FALSE	When TRUE, the function block is executed. The values of the function block inputs can be modified continuously, and the outputs are updated continuously. When FALSE, terminates the function block execution and resets its outputs.

Output Variables

This table describes the output variables:

Output	Type	Initial Value	Description
`Valid`	BOOL	FALSE	Valid data is available at the function block output pin.
`Error`	BOOL	FALSE	If TRUE, indicates that an error was detected. Function block execution is finished.
`ErrorId`	PTO_ERROR	`PTO_ERROR.NoError`	When `Error` is TRUE: code of the error detected *(see page 78)*.
`Velocity`	DINT	0	Actual velocity of the axis (in Hz).

MC_ReadActualPosition_PTO Function Block

Function Description

This function block returns the value of the actual position of the axis.

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 195)*.

Input Variables

This table describes the input variables:

Input	Type	Initial Value	Description
`Axis`	AXIS_REF_PTO	-	Name of the axis (instance) for which the function block is to be executed. In the devices tree, the name is declared in the controller configuration,.
`Enable`	BOOL	FALSE	When TRUE, the function block is executed. The values of the function block inputs can be modified continuously, and the outputs are updated continuously. When FALSE, terminates the function block execution and resets its outputs.

Output Variables

This table describes the output variables:

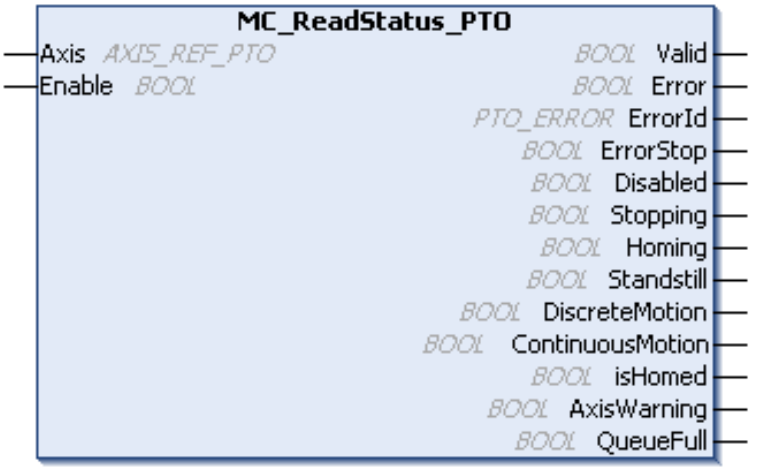
Output	Type	Initial Value	Description
`Valid`	BOOL	FALSE	Valid data is available at the function block output pin.
`Error`	BOOL	FALSE	If TRUE, indicates that an error was detected. Function block execution is finished.
`ErrorId`	PTO_ERROR	`PTO_ERROR.NoError`	When `Error` is TRUE: code of the error detected *(see page 78)*.
`Position`	DINT	0	Actual position of the axis.

MC_ReadStatus_PTO Function Block

Function Description

This function block returns the state diagram *(see page 83)* status of the axis.

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 195)*.

Input Variables

This table describes the input variables:

Input	Type	Initial Value	Description
`Axis`	AXIS_REF_PTO	-	Name of the axis (instance) for which the function block is to be executed. In the devices tree, the name is declared in the controller configuration,.
`Enable`	BOOL	FALSE	When TRUE, the function block is executed. The values of the function block inputs can be modified continuously, and the outputs are updated continuously. When FALSE, terminates the function block execution and resets its outputs.

Output Variables

This table describes the output variables:

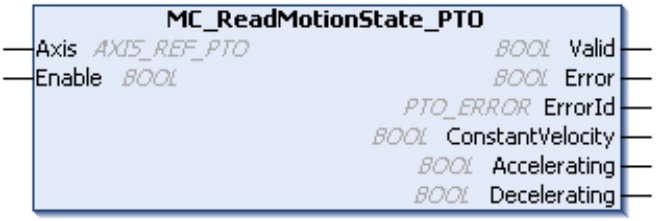
Output	Type	Initial Value	Description
`Valid`	BOOL	FALSE	The set of outputs is valid.
`Error`	BOOL	FALSE	If TRUE, indicates that an error was detected. Function block execution is finished.
`ErrorId`	PTO_ERROR	`PTO_ERROR.NoError`	When `Error` is TRUE: code of the error detected *(see page 78)*.
`ErrorStop`	BOOL	FALSE	If TRUE, the state is active (Motion state diagram *(see page 83)*).
`Disabled`	BOOL	FALSE	
`Stoppping`	BOOL	FALSE	
`Homing`	BOOL	FALSE	
`Stanstill`	BOOL	FALSE	
`DiscreteMotion`	BOOL	FALSE	
`ContinuousMotion`	BOOL	FALSE	
`IsHomed`	BOOL	FALSE	If TRUE, the reference point is valid, absolute motion is allowed.
`AxisWarning`	BOOL	FALSE	If TRUE, an alert is present on the axis (call `MC_ReadAxisError_PTO` *(see page 157)* for detailed information).
`QueueFull`	BOOL	FALSE	If TRUE, the motion queue is full, no additional move is allowed in the buffer.

MC_ReadMotionState_PTO Function Block

Function Description

This function block returns the actual motion status of the axis.

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 195)*.

Input Variables

This table describes the input variables:

Input	Type	Initial Value	Description
`Axis`	AXIS_REF_PTO	-	Name of the axis (instance) for which the function block is to be executed. In the devices tree, the name is declared in the controller configuration,.
`Enable`	BOOL	FALSE	When TRUE, the function block is executed. The values of the function block inputs can be modified continuously, and the outputs are updated continuously. When FALSE, terminates the function block execution and resets its outputs.

Output Variables

This table describes the output variables:

Output	Type	Initial Value	Description
`Valid`	BOOL	FALSE	Valid data is available at the function block output pin.
`Error`	BOOL	FALSE	If TRUE, indicates that an error was detected. Function block execution is finished.
`ErrorId`	PTO_ERROR	`PTO_ERROR.NoError`	When `Error` is TRUE: code of the error detected *(see page 78)*.
`ConstantVelocity`	BOOL	FALSE	The actual velocity is constant.
`Accelerating`	BOOL	FALSE	The actual velocity is increasing.
`Decelerating`	BOOL	FALSE	The actual velocity is decreasing.

Section 7.2
Parameters Function Blocks

Overview

This section describes the parameters function blocks.

What Is in This Section?

This section contains the following topics:

MC_ReadParameter_PTO Function Block

Function Description

This function block is used to get parameters from the PTO.

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 195)*.

Input Variables

This table describes the input variables:

Input	Type	Initial Value	Description
`Axis`	AXIS_REF_PTO	-	Name of the axis (instance) for which the function block is to be executed. In the devices tree, the name is declared in the controller configuration,.
`Enable`	BOOL	FALSE	When TRUE, the function block is executed. The values of the function block inputs can be modified continuously, and the outputs are updated continuously. When FALSE, terminates the function block execution and resets its outputs.
`ParameterNumber`	INT	0	ID of the requested parameter (PTO_PARAMETER *(see page 77)*)

Output Variables

This table describes the output variables:

Output	Type	Initial Value	Description
`Valid`	BOOL	FALSE	Valid data is available at the function block output pin.
`Error`	BOOL	FALSE	If TRUE, indicates that an error was detected. Function block execution is finished.
`ErrorId`	PTO_ERROR	`PTO_ERROR.NoError`	When `Error` is TRUE: code of the error detected *(see page 78)*.
`Value`	DINT	0	Value of the requested parameter.

MC_WriteParameter_PTO Function Block

Function Description

This function block is used to write parameters to the PTO.

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 195)*.

Input Variables

This table describes the input variables:

Input	Type	Initial Value	Description
`Axis`	AXIS_REF_PTO	-	Name of the axis (instance) for which the function block is to be executed. In the devices tree, the name is declared in the controller configuration,.
`Execute`	BOOL	FALSE	On rising edge, starts the function block execution. On falling edge, resets the outputs of the function block when its execution terminates.
`ParameterNumber`	INT	0	ID of the requested parameter (PTO_PARAMETER *(see page 77)*)
`Value`	DINT	0	Value to be written to the requested parameter.

Output Variables

This table describes the output variables:

Output	Type	Initial Value	Description
Done	BOOL	FALSE	If TRUE, indicates that the function block execution is finished with no error detected..
Error	BOOL	FALSE	If TRUE, indicates that an error was detected. Function block execution is finished.
ErrorId	PTO_ERROR	PTO_ERROR.NoError	When Error is TRUE: code of the error detected *(see page 78)*.

MC_ReadBoolParameter_PTO Function Block

Function Description

This function block is used to get BOOL parameters from the PTO.

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 195)*.

Input Variables

This table describes the input variables:

Input	Type	Initial Value	Description
`Axis`	AXIS_REF_PTO	-	Name of the axis (instance) for which the function block is to be executed. In the devices tree, the name is declared in the controller configuration.
`Enable`	BOOL	FALSE	When TRUE, the function block is executed. The values of the other function block inputs can be modified continuously, and the function block outputs are updated continuously. When FALSE, terminates the function block execution and resets its outputs.
`ParameterNumber`	INT	0	ID of the requested parameter (PTO_PARAMETER *(see page 77)*)

Output Variables

This table describes the output variables:

Output	Type	Initial Value	Description
`Valid`	BOOL	FALSE	Valid data is available at the function block output pin.
`Error`	BOOL	FALSE	If TRUE, indicates that an error was detected. Function block execution is finished.
`ErrorId`	PTO_ERROR	`PTO_ERROR.NoError`	When `Error` is TRUE: code of the error detected *(see page 78)*.
`Value`	BOOL	FALSE	Value of the requested parameter.

MC_WriteBoolParameter_PTO Function Block

Function Description

This function block is used to write BOOL parameters to the PTO.

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 195)*.

Input Variables

This table describes the input variables:

Input	Type	Initial Value	Description
`Axis`	AXIS_REF_PTO	-	Name of the axis (instance) for which the function block is to be executed. In the devices tree, the name is declared in the controller configuration.
`Execute`	BOOL	FALSE	On rising edge, starts the function block execution. On falling edge, resets the outputs of the function block when its execution terminates.
`ParameterNumber`	INT	0	ID of the requested parameter (PTO_PARAMETER *(see page 77)*)
`Value`	BOOL	FALSE	Value to be written to the requested parameter.

Output Variables

This table describes the output variables:

Output	Type	Initial Value	Description
`Done`	BOOL	FALSE	If TRUE, indicates that the function block execution is finished with no error detected.
`Error`	BOOL	FALSE	If TRUE, indicates that an error was detected. Function block execution is finished.
`ErrorId`	PTO_ERROR	`PTO_ERROR.NoError`	When `Error` is TRUE: code of the error detected *(see page 78)*.

Section 7.3
Probe Function Blocks

Overview

This section describes the probe function blocks.

What Is in This Section?

This section contains the following topics:

MC_TouchProbe_PTO Function Block

Function Description

This function block is used to activate a trigger event on the probe input. This trigger event allows to record the axis position, and/or to start a buffered move.

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 195)*.

Input Variables

This table describes the input variables:

Input	Type	Initial Value	Description
`Axis`	AXIS_REF_PTO	-	Name of the axis (instance) for which the function block is to be executed. In the devices tree, the name is declared in the controller configuration.
`Execute`	BOOL	FALSE	On rising edge, starts the function block execution. On falling edge, resets the outputs of the function block when its execution terminates.
`WindowOnly`	BOOL	FALSE	If TRUE, only use the window defined by `FirstPosition` and `LastPosition` to accept trigger events.
`FirstPosition`	DINT	0	Start absolute position from where (positive direction) trigger events are accepted (value included in window).
`LastPosition`	DINT	0	Stop absolute position until where (positive direction) trigger events are accepted (value included in window).
`TriggerLevel`	BOOL	FALSE	If FALSE, position capture at falling edge. If TRUE, position capture at rising edge.

Output Variables

This table describes the output variables:

Output	Type	Initial Value	Description
`Done`	BOOL	FALSE	If TRUE, indicates that the function block execution is finished with no error detected.
`Busy`	BOOL	FALSE	If TRUE, indicates that the function block execution is in progress.
`CommandAborted`	BOOL	FALSE	Function block execution is finished, by aborting due to another move command or a error detected.
`Error`	BOOL	FALSE	If TRUE, indicates that an error was detected. Function block execution is finished.
`ErrorId`	PTO_ERROR	`PTO_ERROR.NoError`	When `Error` is TRUE: code of the error detected *(see page 78)*.
`RecordedPosition`	DINT	0	Position where trigger event was detected.

NOTE: Only the first event after the rising edge at the `MC_TouchProbe_PTO` function block `Busy` pin is valid. Once the `Done` output pin is set, subsequent events are ignored. The function block needs to be reactivated to respond to other events.

MC_AbortTrigger_PTO Function Block

Function Description

This function block is used to abort function blocks which are connected to trigger events (for example, `MC_TouchProbe_PTO`).

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 195)*.

Input Variables

This table describes the input variables:

Input	Type	Initial Value	Description
`Axis`	AXIS_REF_PTO	-	Name of the axis (instance) for which the function block is to be executed. In the devices tree, the name is declared in the controller configuration.
`Execute`	BOOL	FALSE	On rising edge, starts the function block execution. On falling edge, resets the outputs of the function block when its execution terminates.

Output Variables

This table describes the output variables:

Output	Type	Initial Value	Description
`Done`	BOOL	FALSE	If TRUE, indicates that the function block execution is finished with no error detected.
`Error`	BOOL	FALSE	If TRUE, indicates that an error was detected. Function block execution is finished.
`ErrorId`	PTO_ERROR	`PTO_ERROR.NoError`	When `Error` is TRUE: code of the error detected *(see page 78)*.

Section 7.4
Error Handling Function Blocks

Overview

This section describes the error handling function blocks.

What Is in This Section?

This section contains the following topics:

MC_ReadAxisError_PTO Function Block

Function Description

This function block retrieves the axis control error. If no axis control error is pending, the function block returns `AxisErrorId` = 0.

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 195)*.

Input Variables

This table describes the input variables:

Input	Type	Initial Value	Description
`Axis`	AXIS_REF_PTO	-	Name of the axis (instance) for which the function block is to be executed. In the devices tree, the name is declared in the controller configuration.
`Enable`	BOOL	FALSE	When TRUE, the function block is executed. The values of the function block inputs can be modified continuously, and the outputs are updated continuously. When FALSE, terminates the function block execution and resets its outputs.

Output Variables

This table describes the output variables:

Output	Type	Initial Value	Description
`Valid`	BOOL	FALSE	Valid data is available at the function block output pin.
`Error`	BOOL	FALSE	If TRUE, indicates that an error was detected. Function block execution is finished.
`ErrorId`	PTO_ERROR	`PTO_ERROR.NoError`	When `Error` is TRUE: code of the error detected *(see page 78)*.
`AxisErrorId`	PTO_ERROR	`PTO_ERROR.NoError`	Index 1000 of PTO_ERROR *(see page 78)*.

MC_Reset_PTO Function Block

Function Description

This function block resets all axis-related errors, conditions permitting, allowing a transition from the state **ErrorStop** to **Standstill**. It does not affect the output of the function blocks instances.

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 195)*.

Input Variables

This table describes the input variables:

Input	Type	Initial Value	Description
`Axis`	AXIS_REF_PTO	-	Name of the axis (instance) for which the function block is to be executed. In the devices tree, the name is declared in the controller configuration.
`Execute`	BOOL	FALSE	On rising edge, starts the function block execution. On falling edge, resets the outputs of the function block when its execution terminates.

Output Variables

This table describes the output variables:

Output	Type	Initial Value	Description
`Done`	BOOL	FALSE	If TRUE, indicates that the function block execution is finished with no error detected.
`Error`	BOOL	FALSE	If TRUE, indicates that an error was detected. Function block execution is finished.
`ErrorId`	PTO_ERROR	`PTO_ERROR.NoError`	When `Error` is TRUE: code of the error detected *(see page 78)*.

Section 7.5
Adding an Administrative Function Block

Adding an Administrative Function Block

Procedure

Follow these steps to add and create the instance of an administrative function block:

Step	Action
1	Add a POU *(see SoMachine, Programming Manual,)* in the **Applications tree**.
2	Select the **Libraries** tab in the **Software Catalog** and click **Libraries**. Select **Controller** → **M241** → **M241 PTOPWM** → **PTO** → **Administrative** → **MC_xxxxxx_PTO** in the list, drag-and-drop the item onto the **POU** window.
3	Create the function block instance by clicking: 
4	Associate the input/output variables *(see page 135)* of the function block.

Part III
Pulse Width Modulation (PWM)

Overview

This part describes the `Pulse Width Modulation` function.

What Is in This Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
8	Introduction	163
9	Configuration and Programming	169
10	Data Types	177

Chapter 8
Introduction

Overview

This chapter provides a description of the PWM functions.

What Is in This Chapter?

This chapter contains the following topics:

Description

Overview

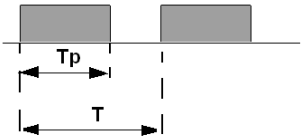
The pulse width modulation function generates a programmable pulse wave signal on a dedicated output with adjustable duty cycle and frequency.

Signal Form

The signal form depends on the following input parameters:

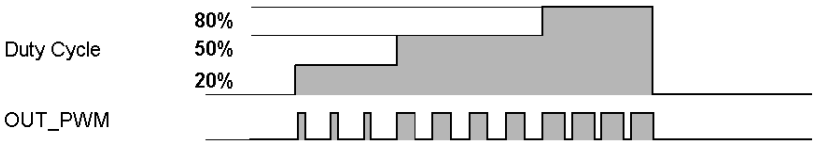
- **Frequency** configurable from 0.1 Hz to 20 kHz with a 0.1 Hz step
- **Duty Cycle** of the output signal from 0% to 100%

Duty Cycle=Tp/T

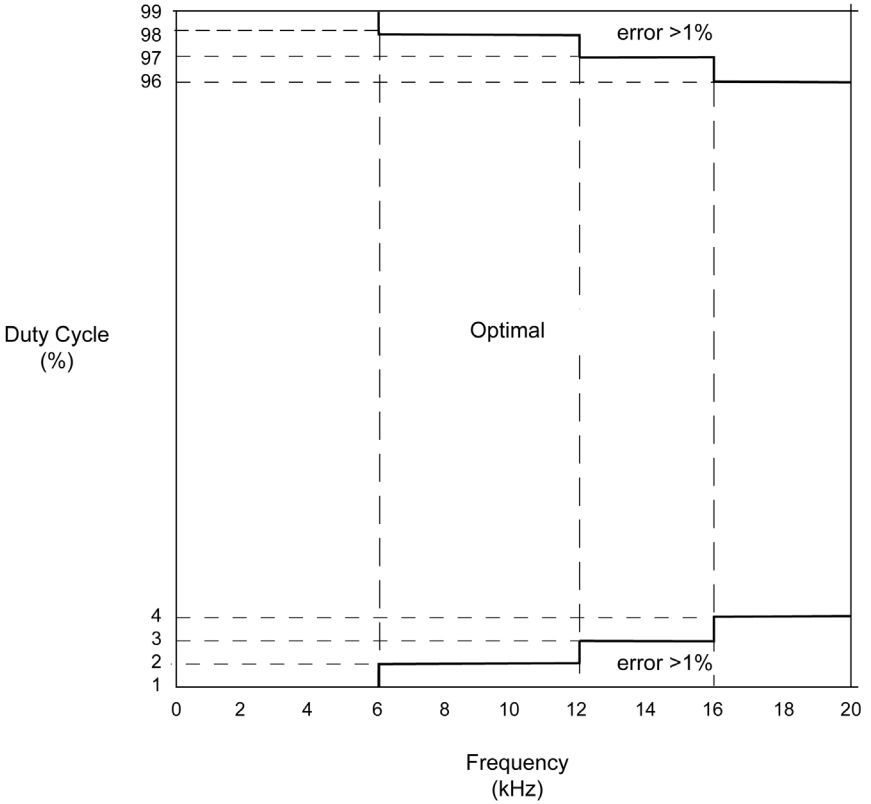


Tp pulse width
T pulse period (1/Frequency)

Modifying the duty cycle in the program modulates the width of the signal. Below is an illustration of an output signal with varying duty cycles.



When duty cycle is below 4% or above 96%, depending on the frequency, the deviation is above 1% as illustrated in the graphic below:



FG/PWM Naming Convention

Definition

Frequency Generator and Pulse Width Modulation uses 1 fast physical output and up to 2 physical inputs.

In this document, use the following naming convention:

Name	Description
SYNC	Synchronization function *(see page 167)*.
EN	Enable function *(see page 167)*.
IN_SYNC	Physical input dedicated to the SYNC function.
IN_EN	Physical input dedicated to the EN function.
OUT_PWM	Physical output dedicated to the FG or PWM.

Synchronization and Enable Functions

Introduction

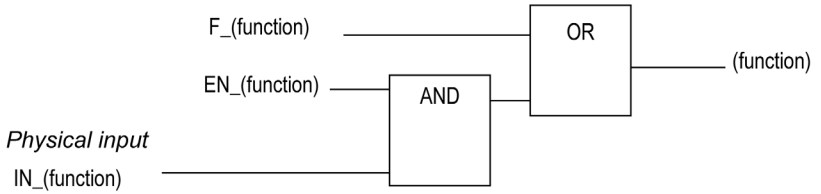
This section presents the functions used by the FG/PWM:

- **Synchronization** function
- **Enable** function

Each function uses the 2 following function block bits:

- **EN_(function) bit:** Setting this bit to 1 allows the (function) to operate on an external physical input if configured.
- **F_(function) bit:** Setting this bit to 1 forces the (function).

The following diagram explains how the function is managed:



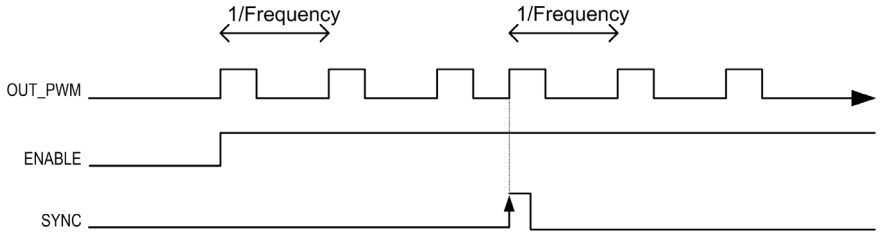
NOTE: (function) stands either for **Enable** (for Enable function) or **Sync** (for Synchronization function).
If the physical input is required, enable it in the configuration screen *(see page 170)*.

Synchronization Function

The **Synchronization** function is used to interrupt the current FG/PWM cycle and then restart a new cycle.

Enable Function

The **Enable** function is used to activate the FG/PWM:



Chapter 9
Configuration and Programming

Overview

This chapter provides configuration and programming guidelines for using PWM functions.

What Is in This Chapter?

This chapter contains the following topics:

Configuration

Overview

2 pulse width modulation functions can be configured on the controller.

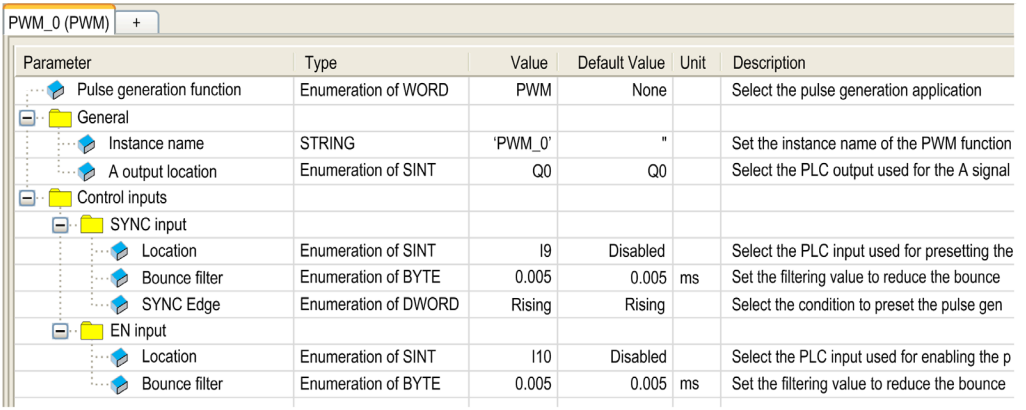
Adding a Pulse Width Modulation Function

To add a pulse width modulation function, proceed as follows:

Step	Action
1	Double-click the **Pulse Generators** node of your controller in the **Devices Tree**.
2	Double-click the **Pulse generation function** value and select **PWM**. **Result:** The PWM configuration parameters appear.

Parameters

The figure provides an example of a PWM configuration window:



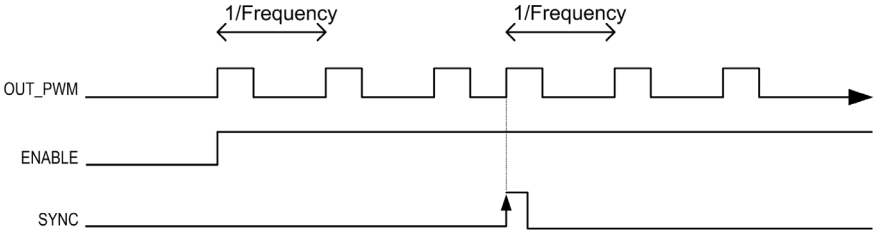
The pulse width modulation function has the following parameters:

Parameter		Value	Default	Description
General	Instance name	-	PWM_0 or PWM_1	Set the instance name of the PWM function.
	A output location	Q0 or Q4 (channel0) Q2 or Q6 (channel1)	Q0 (channel 0) Q2 (channel 1)	Select the controller output used for the A signal.
Control inputs / SYNC input	Location	Disabled I9 (channel 0) I12 (channel 1)	Disabled	Select the controller input used for presetting the PWM function.
	Bounce filter	0.000 0.001 0.002 0.005 0.010 0.1 1.5 1 5	0.005	Set the filtering value to reduce the bounce effect on the SYNC input (in ms).
	SYNC Edge	Rising Falling Both	Rising	Select the condition to preset the PWM function with the SYNC input.
Control inputs / EN input	Location	Disabled I3 or I10 (channel 0) I7 or I13 (channel 1)	Disabled	Select the controller input used for enabling the PWM function.
	Bounce filter	0.000 0.001 0.002 0.005 0.010 0.1 1.5 1 5	0.005	Set the filtering value to reduce the bounce effect on the EN input (in ms).

Synchronizing with an External Event

On a rising edge on the IN_SYNC physical input (with EN_Sync = 1), the current cycle is interrupted and the PWM restarts a new cycle.

This illustration provides a pulse diagram for the `Pulse Width Modulation` function block with use of IN_SYNC input:



Function Blocks

Overview

The `Pulse Width Modulation` function block commands a pulse width modulated signal output at the specified frequency and duty cycle.

Graphical Representation

This illustration is a `Pulse Width Modulation` function block:

IL and ST Representation

To see the general representation in IL or ST language, refer to the *Differences Between a Function and a Function Block (see page 196)* chapter.

Input Variables

This table describes the input variables:

Inputs	Type	Comment
`EN_Enable`	`BOOL`	`TRUE` = authorizes the PWM enable via the IN_Enable input (if configured).
`F_Enable`	`BOOL`	`TRUE` = enables the `Pulse Width Modulation`.
`EN_SYNC`	`BOOL`	`TRUE` = authorizes the restart via the `IN_Sync` input of the internal timer relative to the time base (if configured).
`F_SYNC`	`BOOL`	On rising edge, forces a restart of the internal timer relative to the time base.
`Frequency`	`DWORD`	Frequency of the `Pulse Width Modulation` output signal in tenths of Hz (range: min 1(0.1 Hz)...max 200,000(20 kHz)).
`Duty`		Duty cycle of the `Pulse Width Modulation` output signal in % (range: min 0...max 100).

Output Variables

This table describes the output variables:

Outputs	Type	Comment
InFrequency	BOOL	TRUE = the Pulse Width Modulation signal is currently being output at the specified frequency and duty cycle. FALSE = • The required frequency cannot be reached for any reason. • F_Enable is set to False. • EN_Enable is set to False or no signal detected on the physical input **EN Input** (if configured).
Busy	BOOL	Busy is used to indicate that a command change is in progress: the frequency is changed. Set to TRUE when the Enable command is set and the frequency or duty is changed. Reset to FALSE when InFrequency or Error is set, or when the Enable command is reset.
Error	BOOL	TRUE = indicates that an error was detected.
ErrID	FREQGEN_PWM_ERR_TYPE *(see page 177)*	When Error is set: type of the detected error.

NOTE: When the required frequency cannot be reached for any reason, the InFrequency output is not set to TRUE, but Error stays to FALSE.

Programming the PWM Function Block

Procedure

Follow these steps to program a **PWM** function block:

Step	Action
1	Select the **Libraries** tab in the **Software Catalog** and click **Libraries**. Select **Controller** → **M241** → **M241 PTOPWM** → **PWM** → **PWM_M241** in the list, drag-and-drop the item onto the **POU** window.
2	Select the function block instance by clicking  . The **Input Assistant** dialog is displayed. Select the global variable which references to the added PWM *(see page 170)* during the configuration and confirm. **NOTE:** If the function block instance is not visible, verify if the PWM is configured.
3	The inputs/outputs are detailed in the function block *(see page 173)*.

Chapter 10
Data Types

FREQGEN_PWM_ERR_TYPE

Error Type Enumeration

This table lists the values for the FREQGEN_PWM_ERR_TYPE enumeration:

Enumerator	Value	Description
FREQGEN_PWM_NO_ERROR	0	No error detected.
FREQGEN_PWM_UNKNOWN_REF	1	The reference to the FG / PWM is not valid.
FREQGEN_PWM_UNKNOWN_PARAMETER	2	The parameter type is unknown in the current mode.
FREQGEN_PWM_INVALID_PARAMETER	3	A parameter value is not valid or the combination of parameter values is not valid.
FREQGEN_PWM_COM_ERROR	4	Communication error with the FG / PWM.
FREQGEN_PWM_AXIS_ERROR	5	PWM is in error state (&quo=PWMError" is set on PTOSimple instance). No move is possible until the error is reset.

Part IV
Frequency Generator (FG)

Overview

This part describes the `Frequency Generator` function.

What Is in This Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
11	Introduction	181
12	Configuration and Programming	185

Chapter 11
Introduction

Overview

This chapter provides a description of the FG functions.

What Is in This Chapter?

This chapter contains the following topics:

Description

Overview

The frequency generator function generates a square wave signal on dedicated output channels with a fixed duty cycle (50%).

Frequency is configurable from 0.1 Hz to 100 kHz with a 0.1 Hz step.

FG Naming Convention

Description

FG/PWM Naming Convention *(see page 166)*

Synchronization and Enable Functions

Description

Synchronization and Enable Functions *(see page 167)*

Chapter 12
Configuration and Programming

Overview

This chapter provides configuration and programming guidelines for using FG functions.

What Is in This Chapter?

This chapter contains the following topics:

Configuration

Overview

2 frequency generator functions can be configured on the controller.

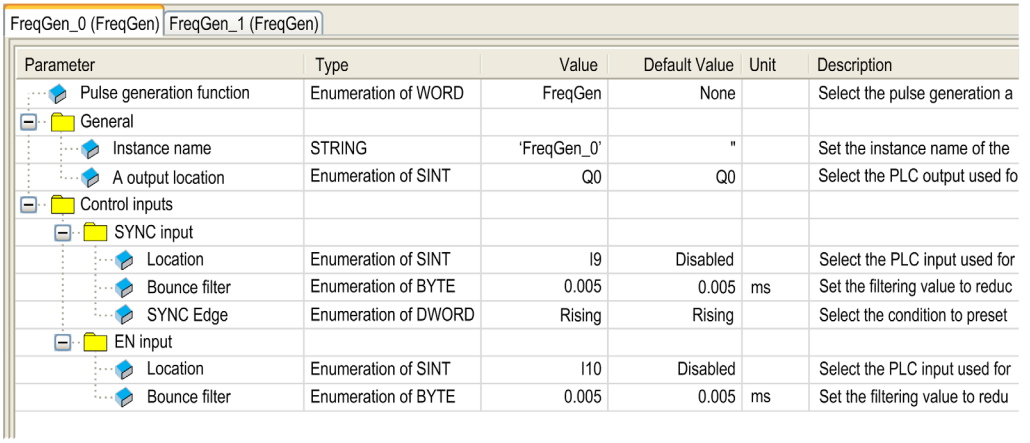
Adding a Frequency Generator Function

To add a frequency generator function, proceed as follows:

Step	Action
1	Double-click the **Pulse Generators** node of your controller in the **Devices Tree**.
2	Double-click the **Pulse generation function** value and select **FreqGen**. **Result:** The frequency generator configuration parameters are displayed.

Parameters

The figure provides an example of a frequency generator configuration window:



Parameter	Type	Value	Default Value	Unit	Description
Pulse generation function	Enumeration of WORD	FreqGen	None		Select the pulse generation a
General					
Instance name	STRING	'FreqGen_0'	''		Set the instance name of the
A output location	Enumeration of SINT	Q0	Q0		Select the PLC output used fo
Control inputs					
SYNC input					
Location	Enumeration of SINT	I9	Disabled		Select the PLC input used for
Bounce filter	Enumeration of BYTE	0.005	0.005	ms	Set the filtering value to reduc
SYNC Edge	Enumeration of DWORD	Rising	Rising		Select the condition to preset
EN input					
Location	Enumeration of SINT	I10	Disabled		Select the PLC input used for
Bounce filter	Enumeration of BYTE	0.005	0.005	ms	Set the filtering value to redu

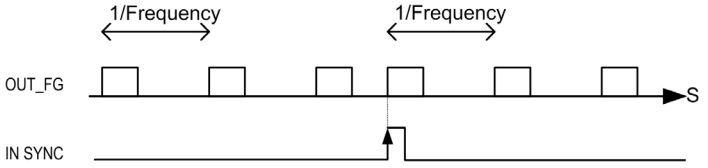
The pulse width modification function has the following parameters:

Parameter		Value	Default	Description
General	Instance name	-	PWM_0 or PWM_1	Set the instance name of the frequency generator function.
	A output location	Q0 or Q4 (channel0) Q2 or Q6 (channel1)	Q0 (channel 0) Q2 (channel 1)	Select the controller output used for the A signal.
Control inputs / SYNC input	Location	Disabled I9 (channel 0) I12 (channel 1)	Disabled	Select the controller input used for presetting the frequency generator function.
	Bounce filter	0.000 0.001 0.002 0.005 0.010 0.1 1.5 1 5	0.005	Set the filtering value to reduce the bounce effect on the SYNC input (in ms).
	SYNC Edge	Rising Falling Both	Rising	Select the condition to preset the frequency generator function with the SYNC input.
Control inputs / EN input	Location	Disabled I3 or I10 (channel 0) I7 or I13 (channel 1)	Disabled	Select the controller input used for enabling the frequency generator function.
	Bounce filter	0.000 0.001 0.002 0.005 0.010 0.1 1.5 1 5	0.005	Set the filtering value to reduce the bounce effect on the EN input (in ms).

Synchronizing with an External Event

On a rising edge on the IN_SYNC physical input (with EN_Sync = 1), the current cycle is interrupted and the FG restarts a new cycle.

This illustration provides a pulse diagram for the frequency generator function block with use of IN_SYNC input:



Function Blocks

Overview

The `Frequency Generator` function block commands a square wave signal output at the specified frequency.

Graphical Representation (LD/FBD)

This illustration is a `Frequency Generator` function block:

IL and ST Representation

To see the general representation in IL or ST language, refer to the *Differences Between a Function and a Function Block (see page 196)* chapter.

Input Variables

This table describes the input variables:

Inputs	Type	Comment
`EN_Enable`	`BOOL`	`TRUE` = authorizes the `Frequency Generator` enable via the IN_EN input (if configured).
`F_Enable`	`BOOL`	`TRUE` = enables the `Frequency Generator`.
`EN_SYNC`	`BOOL`	`TRUE` = authorizes the restart via the IN_SYNC input of the internal timer relative to the time base (if configured).
`F_SYNC`	`BOOL`	On rising edge, forces a restart of the internal timer relative to the time base.
`Frequency`	`DWORD`	Frequency of the `Frequency Generator` output signal in tenths of Hz. (Range: min 1 (0.1Hz)...max 1,000,000 (100kHz)

Output Variables

This table describes the output variables:

Outputs	Type	Comment
InFrequency	BOOL	TRUE = the Frequency Generator signal is output at the specified Frequency. FALSE = ● The required frequency cannot be reached for any reason. ● F_Enable is set to False. ● EN_Enable is set to False or no signal detected on the physical input **EN Input** (if configured).
Busy	BOOL	Busy is used to indicate that a command change is in progress: the frequency is changed. Set to TRUE when the Enable command is set and the frequency is changed. Reset to FALSE when InFrequency or Error is set, or when the Enable command is reset.
Error	BOOL	TRUE = indicates that an error was detected.
ErrID	FREQGEN_PWM_ERR_TYPE *(see page 177)*	When Error is set: type of the detected error.

NOTE: When the required frequency cannot be reached for any reason, the InFrequency output is not set to TRUE, but Error stays to FALSE.

Programming

Procedure

Follow these steps to program a `Frequency Generator` function block:

Step	Action
1	Select the **Libraries** tab in the **Software Catalog** and click **Libraries**. Select **Controller** → **M241** → **M241 PTOPWM** → **Frequency Generator** → **FrequencyGenerator_M241** in the list; drag-and-drop the item onto the **POU** window.
2	Select the function block instance by clicking  . The Input Assistant screen appears. Select the global variable which references to the added FreqGen *(see page 186)* during the configuration and confirm. **NOTE:** If the function block instance is not visible, verify if the frequency generator is configured.
3	The inputs/outputs are detailed in the function block *(see page 189)*.

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.

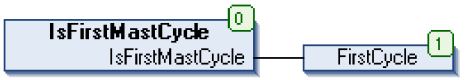
Functions `IsFirstMastCycle` and `SetRTCDrift` and Function Block `TON` are used as examples to show implementations.

Using a Function in IL Language

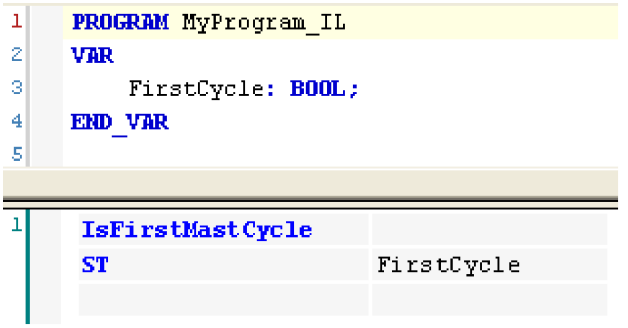
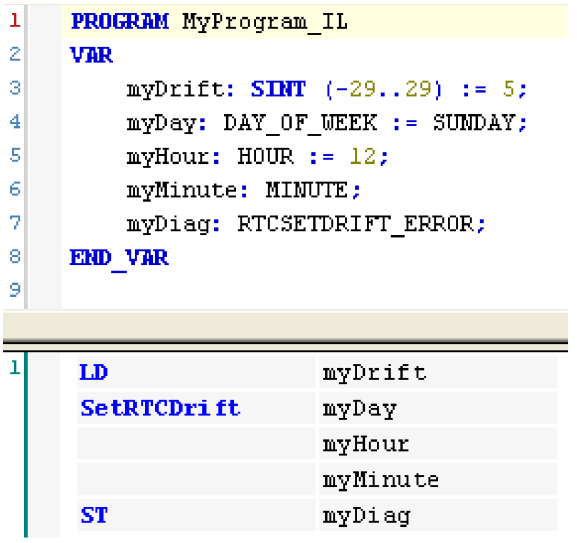
This procedure describes how to insert a function in IL language:

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 *(see SoMachine, Programming Manual,)*.
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

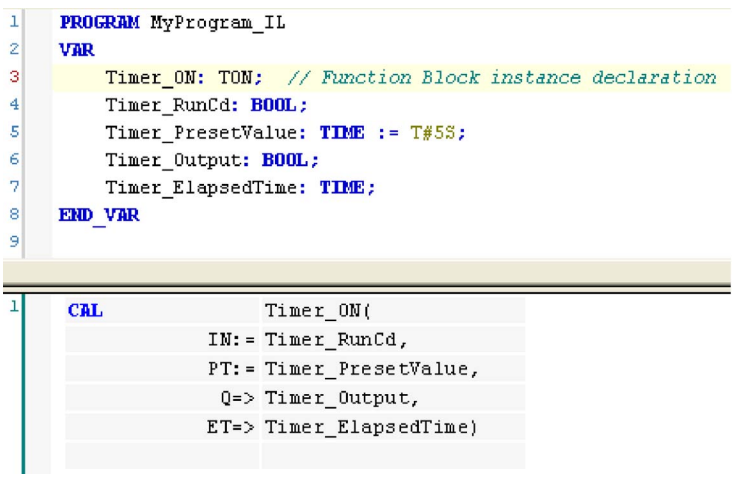
This procedure describes how to insert a function block in IL language:

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 *(see SoMachine, Programming Manual,)*.
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 *(see SoMachine, Programming Manual,)*.
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

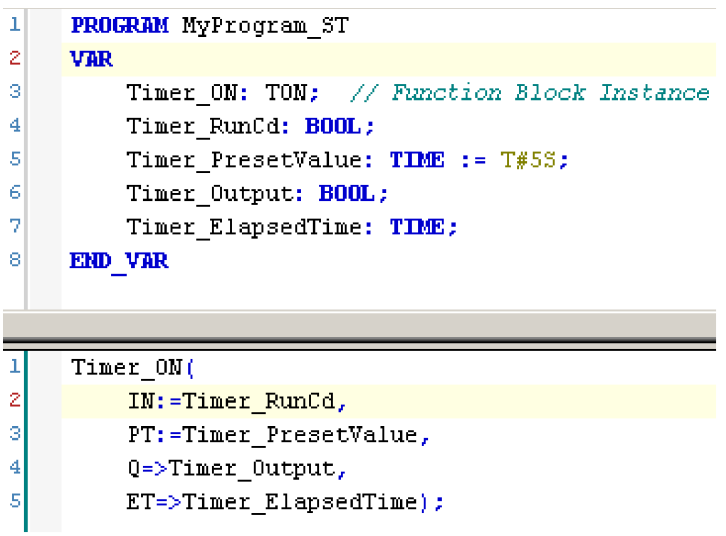
This procedure describes how to insert a function block in ST language:

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 *(see SoMachine, Programming Manual,)*.
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

A

absolute movement

A movement to a position defined from a reference point.

acceleration / deceleration

Acceleration is the rate of velocity change, starting from **Start Velocity** to target velocity. Deceleration is the rate of velocity change, starting from target velocity to **Stop Velocity**. These velocity changes are implicitly managed by the PTO function in accordance with acceleration, deceleration, and jerk ratio parameters following a trapezoidal or an S-curve profile.

application

A program including configuration data, symbols, and documentation.

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

A programming unit that has 1 input and returns 1 immediate result. However, unlike FBs, it is directly called with its name (as opposed to through an instance), has no persistent state from one call to the next and can be used as an operand in other programming expressions.

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

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.

H

homing

The method used to establish the reference point for absolute movement.

I

IEC 61131-3

Part 3 of a 3-part IEC standard for industrial automation equipment. IEC 61131-3 is concerned with controller programming languages and defines 2 graphical and 2 textual programming language standards. The graphical programming languages are ladder diagram and function block diagram. The textual programming languages include structured text and instruction list.

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.

J

jerk ratio

The proportion of change of the acceleration and deceleration as a function of time.

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

S-curve ramp

An acceleration / deceleration ramp with a `JerkRatio` parameter greater than 0%.

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.

start velocity

The minimum frequency at which a stepper motor can produce movement, with a load applied, without the loss of steps.

stop velocity

The maximum frequency at which a stepper motor stops producing movement, with a load applied, without the loss of steps.

T

trapezoidal ramp

An acceleration / deceleration ramp with a `JerkRatio` parameter set to 0%.

V

variable

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

Index