Magelis SCU

HMI Controller
PTO/PWM Library Guide

12/2016

EIO0000001518.05

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.

© 2016 Schneider Electric. All Rights Reserved.

Table of Contents

Safety Information

Important Information

NOTICE

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

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

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

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

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

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

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

PLEASE NOTE

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

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

About the Book

At a Glance

Document Scope

This documentation will acquaint you with the pulse train output (PTO) and pulse width modulation (PWM) output functions offered within the HMI SCU controller.

This documentation describes the data types and functions of the HMI SCU PTO/PWM library.

In order to use this manual, you must:

- Have a thorough understanding of the HMI SCU, 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 V4.2.

Related Documents

Title of Documentation	Reference Number
Magelis SCU HMI Controller PLCSystem Library Guide	EIO0000001246 (eng), EIO0000001247 (fre), EIO0000001248 (ger), EIO0000001249 (spa), EIO0000001250 (ita), EIO0000001251 (chs)
Magelis SCU HMI Controller HSC Library Guide	EIO0000001512 (eng), EIO0000001513 (fre), EIO0000001514 (ger), EIO0000001515 (spa), EIO0000001516 (ita), EIO0000001517 (chs)

Title of Documentation	Reference Number
PLCCommunication Library Guide	EIO0000000361 (eng), EIO0000000742 (fre), EIO0000000743 (ger), EIO0000000744 (spa), EIO0000000745 (ita), EIO0000000746 (chs)
Magelis SCU HMI Controller Hardware Guide	EIO0000001232 (eng), EIO0000001233 (fre), EIO0000001234 (ger), EIO0000001235 (spa), EIO0000001236 (ita), EIO0000001237 (chs), EIO0000001238 (por)

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

Product Related Information

⚠ WARNING

LOSS OF CONTROL

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

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

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

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

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

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

Terminology Derived from Standards

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

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

Among others, these standards include:

Standard	Description
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.

Standard	Description
2006/42/EC	Machinery Directive
2014/30/EU	Electromagnetic Compatibility Directive
2014/35/EU	Low Voltage Directive

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

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

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

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

Part I
HMI SCU Embedded Functions

Chapter 1
HMI SCU Embedded Functions

PTO_PWM Embedded Function

Overview

The PTO embedded function can provide 2 different functions:

PTO The PTO (Pulse Train Output) implements digital technology *(see page 21)* that provides precise positioning for open loop control of motor drives.

PWM The PWM (Pulse Width Modulation) function generates a programmable square wave signal on a dedicated output *(see page 77)* with adjustable duty cycle and frequency.

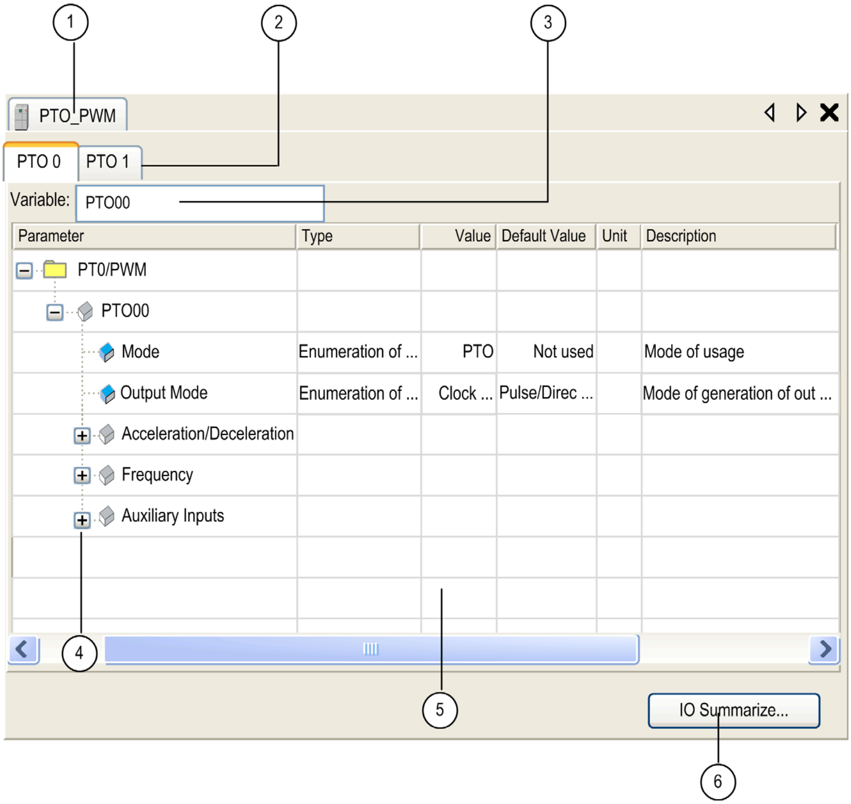
Accessing the PTO_PWM Configuration Window

Follow this step to access the PTO_PWM embedded function configuration window:

Step	Description
1	In the **Devices tree**, double-click **HMISCUxx5** → **Embedded Functions** → **PTO_PWM**. **Result**: The **PTO_PWM** window is displayed.

PTO_PWM Configuration Window

The figure shows a sample PTO_PWM configuration window used to configure a PTO or PWM:



The table describes the areas of the PTO_PWM configuration window:

Number	Action
1	If necessary, select the **PTO_PWM** tab to access the PTO_PWM configuration Windows.
2	Select a specific PTO tab to access the PTO_PWM channel to configure.
3	Choose the type of PTO_PWM, (**PTO** (default) or **PWM**). Use the field **Variable** to change the **Global Variable** name representing the instance of the channel. **NOTE:** The default variable name for PTO 0 channel is **PTO00**. For PTO 1 channel, it is **PTO01**.
4	You can expand each parameter by clicking the plus sign next to it to access its settings.

Number	Action
5	Configuration window where the embedded function is used for: ● PTO ● PWM
6	Click the **IO Summarize** button. Result: The **IO Summary** window appears that shows the configured I/O mapping.

For detailed information on configuration parameters, refer to:
- PTO configuration *(see page 23)*.
- PWM configuration *(see page 86)*.

Part II
Pulse Train Output (PTO)

Overview

This part describes how to configure and use a PTO function.

What Is in This Part?

This part contains the following chapters:

Chapter 2
Overview

Pulse Train Output (PTO)

Introduction

The PTO (Pulse Train Output) implements digital technology that helps provide precise positioning for open loop control of motor drives.

The PTO and PWM (pulse width modulation) functions use the same dedicated outputs. PTO can be configured on channel 0. Alternatively, up to 2 PWMs can be configured on channel 0 and channel 1.

Concept

The PTO function provides a square wave output for a specified number of pulses and a specified frequency.

PTO is used to control the positioning or speed of the axis of a rotating device.

PTO Commands

The `PTOSimple` *(see page 31)* function block manages the PTO.

Motion commands are managed by 2 motion function blocks:

- `PTOMoveRelative` *(see page 38)*: moving to a specified axis position
- `PTOMoveVelocity` *(see page 49)*: moving axis at a specified speed
- `PTOStop` *(see page 54)*: stop moving

Adjustments and diagnostics are managed by 3 administrative function blocks:

- `PTOSetParam` *(see page 68)*: Modify a parameter
- `PTOGetParam` *(see page 66)*: Read a parameter
- `PTOGetDiag` *(see page 71)*: Identify a detected error

Performance

The maximum generated frequency is 50 kHz for PTO.

The maximum generated for PWM is 65 kHz.

The embedded PTO function can be used for single axis point-to-point motion but not for applications that would require:
- 2-axis simultaneous point-to-point motion (each axis is managed independently).
- 2-axis synchronized point-to-point motion,
- 2-axis interpolation motion.

PWM can control up to 2 independent axes.

NOTE:
- PTO or PWM can be configured on channel 0.
- PTO cannot be configured on channel 1. PTO requires 2 fast digital outputs.
- If PTO is selected for channel 0, PWM on channel 1 cannot be used.

Chapter 3
Configuration

Overview

This chapter describes how to configure a PTO.

What Is in This Chapter?

This chapter contains the following topics:

PTO Configuration

Overview

1 PTO can be configured on the controller.

Hardware Configuration

A configured PTO uses 2 fast / pulse outputs for pulse and direction and 1 digital input (an optionally configured auxiliary).

Open Configuration Window

Use this procedure to open the PTO_PWM configuration window:

Step	Action
1	In the **Devices tree**, double-click **HMISCUxx5** → **Embedded Functions** → **PTO_PWM**. **Result**: The **PTO_PWM** window is displayed.
2	Select **PTO** in the **PTO0•** → **Mode** drop down menu.
3	An instance of the PTO is created, it can be renamed in the **Variable** field. Default name is: **PTO00**.

Configuration Window Description

This table describes each parameter available when the embedded PTO_PWM is configured in **PTO** mode:

Parameter	Value	Default Value	Unit	Description
Mode	**Not Used** **PTO** **PWM**	**Not Used**	–	The mode selected is PTO.
Output Mode *(see page 28)*	**Pulse/Direction** **Direction/Pulse**	**Pulse/Direction**	–	Mode of generation of outputs

Parameter		Value	Default Value	Unit	Description
Acceleration/ Deceleration *(see page 28)*	Acc./Dec. Unit	ms Hz/ms	ms	–	Acceleration/Deceleration Unit **NOTE:** When this setting is changed between **ms** and **Hz/ms**, the values for **Acc. max.**, **Dec. max.**, and **Dec. Fast Stop** will be automatically converted in the user interface.
	Acc. max.	1...49999	20	ms	Acceleration time minimum value when **Acc./Dec. Unit** is set to **ms**.
		1...50000	2500	Hz/ms	Acceleration rate maximum value when **Acc./Dec. Unit** is set to **Hz/ms**.
	Dec. max.	1...49999	20	ms	Deceleration time minimum value when **Acc./Dec. Unit** is set to **ms**.
		1...50000	2500	Hz/ms	Deceleration rate maximum value when **Acc./Dec. Unit** is set to **Hz/ms**.
	Dec. Fast Stop	1...49999	100	ms	When the **Acc./Dec. Unit** is set to **ms**, the deceleration rate used in a **Dec. Fast Stop** (triggered by a Drive Ready input low, limits exceeded, detected errors) is calculated from this time value: **Dec. Fast Stop** rate = Maximum Frequency (Hz) / **Dec. Fast Stop** time (ms). ● Therefore, the actual Fast Stop Deceleration time = current frequency / calculated Deceleration Fast Stop rate. ● If the newly calculated deceleration rate is less than 1 Hz/ms, the rate used will be just 1 Hz/ms. **NOTE:** If the **Stop Frequency** is configured to be 0 Hz, the frequency of the last pulse period = SQRT ((deceleration rate in Hz/s) / 2) rounded to the nearest whole number. Refer to Appendix for more information *(see page 99)*.
		1...50000	500	Hz/ms	

Parameter		Value	Default Value	Unit	Description
Frequency *(see page 29)*	**Start**	**0...50000**	**0**	**Hz**	On execution of **PTOMoveRelative**, the first pulse period during acceleration starts at this frequency. **NOTE:** If the **Start Frequency** is set to 0 Hz, the actual start frequency = SQRT ((acceleration rate in Hz/s) / 2) rounded to the nearest integer. Refer to Appendix for more information *(see page 99)*
	Stop	**0...50000**	**0**	**Hz**	On execution of **PTOMoveRelative**, **PTOStop**, **Dec. Fast Stop**, the last pulse period during deceleration ends at this frequency. **NOTE:** If the **Stop Frequency** is set to 0 Hz, the actual stop frequency = SQRT ((deceleration rate in Hz/s) / 2) rounded to the nearest integer. Refer to Appendix for more information *(see page 99)*
	Maximum	**1...50000**	**50000**	**Hz**	Maximum frequency allowed by **PTOMoveRelative** function block during operation. Frequencies higher than this value if set in a motion function block will return an error.
Auxiliary Inputs *(see page 30)*	**AUX**	**Not used** **Drive Ready**	**Not used**	–	Specific input dedicated to the **Drive Ready** information. **NOTE:** reference point detection (origin) is not supported on HMI SCU.
	AUX Filter	**3** **12**	**3**	**ms**	Filtering value reduces the effect of bounce on the auxiliary input

This example shows an actual fast stop deceleration time using the following settings:

Parameter	Value	Units
Dec. Fast Stop	10,000	ms
Maximum Frequency	50,000	Hz

When the current frequency of a running motion command is at 10,000 Hz, and an error triggers a **Fast Stop** (for example, triggering another **PTOMoveRelative** while a motion is already in progress):

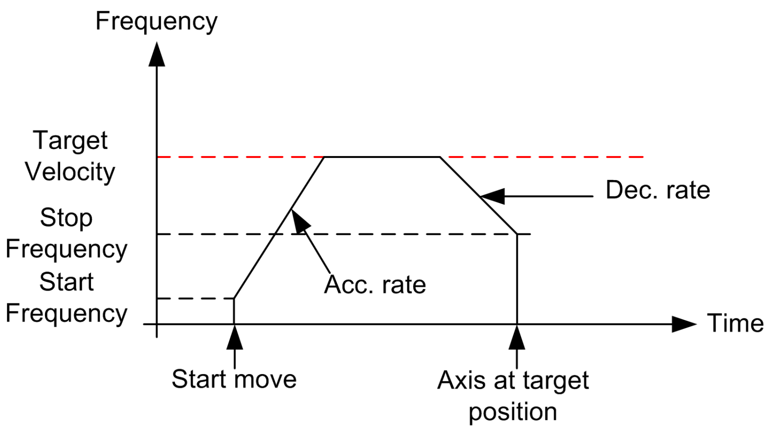
- **Dec. Fast Stop** rate = **Maximum Frequency** / **Dec. Fast Stop** time = 50,000 Hz / 10,000 ms = 5 Hz/ ms
- Since the current frequency is 10,000 Hz, actual **Dec. Fast Stop** time = current frequency / **Dec. Fast Stop** rate = 10,000 Hz / 5 Hz/m = 2000 ms

Configure a PTO Channel

Use this procedure to configure a PTO channel:

Step	Action
1	Enable the PTO channel by selecting **PTO** in the **PTO0• → Mode** drop down menu.
2	Select the mode of generation of outputs in the **PTO0• → Output Mode** drop down menu.
3	Configure the Acc./Dec. Unit, Acc.max, Dec. max, and Dec. Fast stop *(see page 28)* parameters.
4	Configure the Frequency *(see page 29)* parameters: Start, Stop, and Maximum.
5	Optionally enable the AUX input *(see page 30)*.
6	Configure the **PTO0• → AUX** input filtering value (if enabled at step 5).

The configuration defined can be viewed as a configuration profile:



Configuration Parameters Description

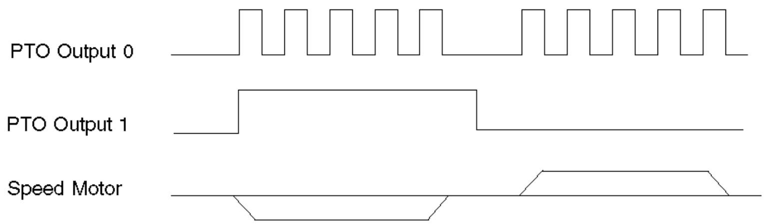
Output Modes

2 possible output modes are as follows:
- **Pulse/Direction**
- **Direction/Pulse**

The **Pulse/Direction** mode generates 2 signals on the PTO outputs:
- on output 0: pulse which provides the motor operating speed.
- on output 1: direction which provides the motor rotation direction.

The **Direction/Pulse** mode generates 2 signals on the PTO outputs:
- on output 0: direction which provides the motor rotation direction.
- on output 1: pulse which provides the motor operating speed.

This diagram gives an example of a timing diagram in **Pulse/Direction** mode:



Acceleration/Deceleration

Parameter	Value	Description
Acc./Dec. Unit	ms	The Acceleration and the Deceleration expressed in ms.
	Hz/ms	The Acceleration and the Deceleration expressed in Hz/ms.
Acc. max. **Dec. max.**	ms	Acc./Dec. which cannot be exceeded by any motion command in the application program. **NOTE:** If the **Acc. max.** or **Dec. max.** is exceeded by a motion command, the motion command will return an error and decelerate to 0 Hz at the **Dec. Fast stop** rate. **ms** The value specifies the lowest value of Acc./Dec. **time** that can be used. **Hz/ms** The value specifies the highest value of Acc./Dec. **rate** that can be used.

Parameter	Value	Description
Dec. Fast stop	ms	Defines the value of Deceleration rate used to stop the PTO signal in case of a detected error: • motion command error • command sequence error • controller leaves RUN mode • **Drive Ready** input low **Dec. Fast stop** is also triggered when: • HMI goes into Offline Configuration mode. • HMI goes to **Exit Runtime** mode (HMI power-down feature used to ensure all logging to file system is closed properly). **NOTE:** The Deceleration time is limited to approximately 6 seconds before just going straight to 0 Hz.

This example shows a possible configuration of Acceleration/Deceleration and Frequency parameters:

Parameter	Value	Units
Acc./Dec. Time	10,000	ms
Start Frequency and **Stop Frequency**	5,000	Hz
Target Velocity	20,000	Hz

In your application program, using a **PTOMoveRelative** command with an **Acceleration Time** of 10,000 ms and a **Target Velocity** of 20 kHz, then the **Target Velocity** will be reached after 10,000 ms with an acceleration rate of 1.5 Hz/ms.

The equivalent acceleration rate = (**Target Velocity** - **Start Frequency**)/**Acc.** = (20 kHz-5 kHz)/10000 ms = 1.5 Hz/ms

Frequency

Parameter	Description
Start	**Start frequency** is the initial frequency value when a motion command begins.
Stop	**Stop frequency** is the final frequency value before a motion command stops.
Maximum	User-defined frequency which cannot be exceeded by any command in the application program. If a command with a desired velocity above the **Maximum Frequency** is executed, an error will be returned and the axis will do a **Dec. Fast Stop**.

Auxiliary Input

Parameter	Description
AUX	The auxiliary input parameter has two possible settings: ● Drive Ready ● Not used **Drive Ready** The state of the PTO 0 - AUX physical input (DI2) determines whether the PTO move commands are authorized. When `TRUE` = Authorizes the PTO moving commands. When `FALSE` = An axis error is triggered and any move ongoing is aborted by a **Fast stop**. The input `DIS_AuxInput` of the PTOSimple function block *(see page 32)* can be used to disable the drive ready monitoring.
AUX Filter	Filtering value reduces the effect of bounce on the auxiliary input.

Chapter 4
PTO Management

Overview

This chapter describes the `PTOSimple` function block used to manage the axis.

What Is in This Chapter?

This chapter contains the following topics:

PTOSimple Function Block

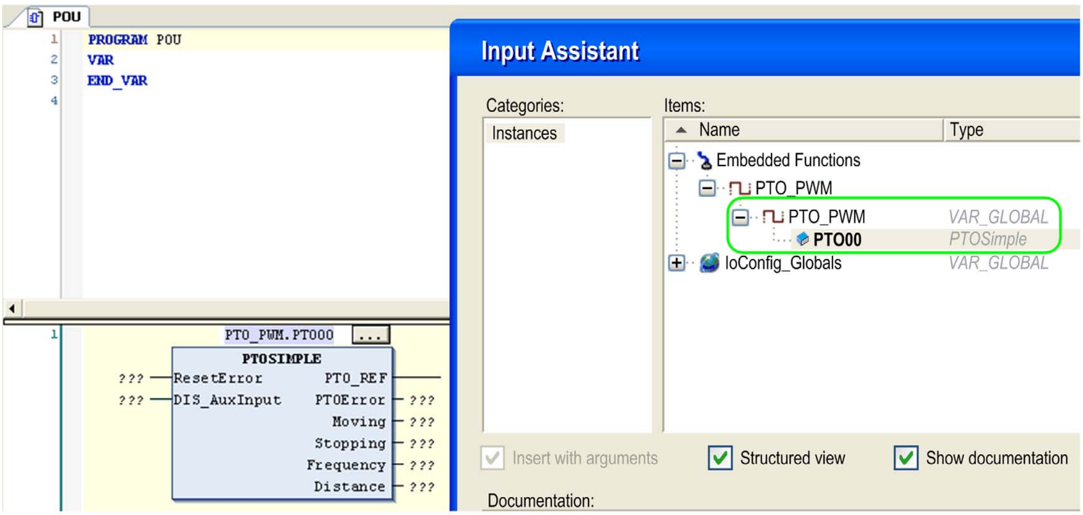
Overview

The `PTOSimple` function block manages the PTO function.

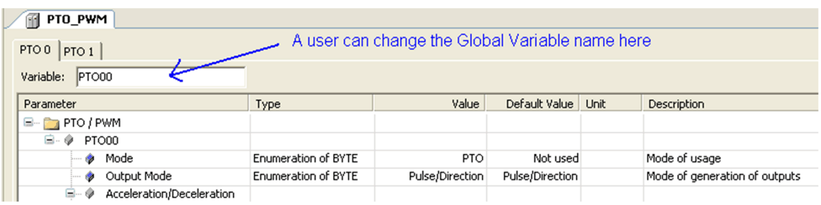
Call the function block in each cycle of the MAST task.

The function block instance name is the name defined by configuration.

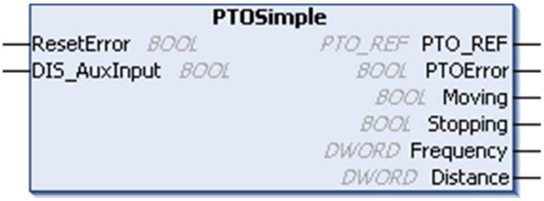
It is created when a user invokes PTO mode on `Channel PTO 0` from the **Embedded Functions** configuration:



NOTE: Assign the function block instance name to the Global Variable `PTO_PWM.PTO00`.



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 103)*.

I/O Variables Description

This table describes the input variables:

Inputs	Type	Comment
ResetError	BOOL	On rising edge, resets the detected PTO error. **NOTE:** The Execute pin on any **PTOMoveRelative** tied to the PTO00 axis must be set to FALSE for resetting detected errors.
DIS_AuxInput	BOOL	TRUE = disables the auxiliary input when configured as **Drive Ready** input. This pin has no effect when auxiliary input is not used.

This table describes the output variables:

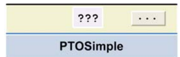
Outputs	Type	Comment
PTO_REF	PTO_REF *(see page 117)*	Reference to the PTO channel. To be used with the PTO_REF_IN input pin of the PTO function blocks.
PTOError	BOOL	TRUE = indicates that an error was detected. Use PTOGetDiag function block *(see page 72)* to get more information about this detected error.
Moving	BOOL	TRUE = indicates that the motion state is moving.
Stopping	BOOL	TRUE = indicates that the motion state is stopping.
Frequency	DWORD	Current velocity (in Hz) of the move.
Distance	DWORD	Distance traveled by the current move of the PTO axis (in number of pulses).

Programming the PTOSimple Function Block

Procedure

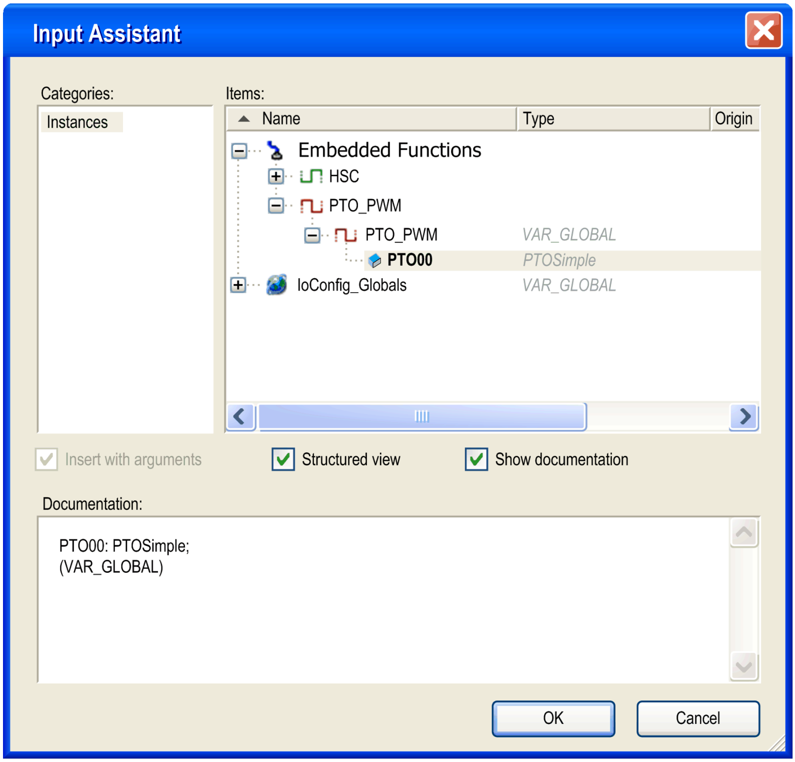
To program the `PTOSimple` function block, do the following:

Step	Action
1	Select **HMISCUxx5** in the **Applications tree**.
2	Click 
3	Select **POU** in the drop-down menu. **Result:** The **Add POU** window is displayed.
4	Enter the appropriate information in the **Add POU** window.
5	Click **Add**.
6	Select the **Libraries** tab in the **Software catalog** and click **Libraries**. Select **Controller** → **HMISCU** → **HMISCU PTOPWM** → **PTOSimple** in the list, drag-and-drop the item onto the lower **POU** window on `Start here`.

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>7</td><td>
Look for the function block instance by clicking 

.

The Input Assistant window appears. Select the Global Variable that you defined during the configuration and confirm. The default Global Variable is `PTO_PWM.PTO00`.



NOTE: If the function block instance is not visible, verify if the PTO is configured.
</td></tr>
<tr><td>8</td><td>The inputs/outputs are detailed in the function block *(see page 32)*.</td></tr>
</table>

Chapter 5
Motion Commands

Overview

This chapter describes the motion commands.

What Is in This Chapter?

This chapter contains the following sections:

Section 5.1
Position Control: PTOMoveRelative

Overview

This section describes the `PTOMoveRelative` function block.

What Is in This Section?

This section contains the following topics:

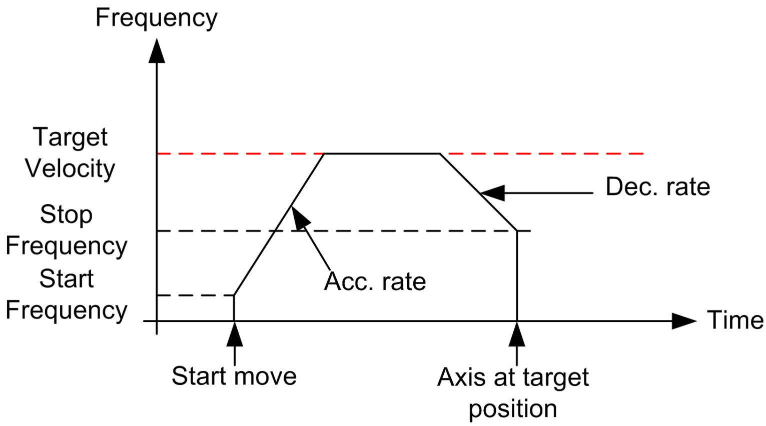
Description

Overview

This function block is used to manage a complete movement of the axis from the current position to a specified target position.

The target position is directly specified by its distance, in pulses, from the current position of the axis.

The velocity of the axis will follow a trapezoidal profile:



NOTE: The Frequency represents the Velocity. The 2 terms are analogous.

PTOMoveRelative Function Block

Function Description

The function block commands a move of a distance relative to the current position.

The move profile depends on the specified velocity, deceleration, and acceleration values.

Graphical Representation

IL and ST Representation

To see the general representation in IL or ST language, refer to the chapter *Function and Function Block Representation (see page 103)*.

I/O Variables Description

The table describes the input variables:

Inputs	Type	Comment
`PTO_REF_IN`	`PTO_REF` *(see page 117)*	Reference to the PTO channel. To be connected to the `PTO_REF` of the **PTOSimple** or the `PTO_REF_OUT` output pins of other PTO function blocks.
`Execute`	`BOOL`	On rising edge, starts the function block execution. Output status pins continue to output the current status while the Move is happening, whether or not the Execute pin is true or not.

Inputs	Type	Comment
Velocity	DWORD	Target/Desired **Velocity** in Hz (not necessarily reached.) Range: 1...Maximum frequency of the output **NOTE:** • When **Velocity** is set to 0, and the function block is executed, an error will be returned (PTO_INVALID_PARAMETER). • If the **Velocity** is less than the configured non-zero **Start Frequency** or **Stop Frequency**, an error will be returned (PTO_INVALID_PARAMETER). • If the **Start Frequency** or **Stop Frequency** is configured as zero, and the **Velocity** is set to ≤ the calculated **Start/Stop Frequency**, there will be no acceleration or deceleration phase. The output frequency will simply be that of the **Velocity**.
Distance	DWORD	Distance of the move in number of pulses. Range: 1...4294967295 **NOTE:** If the Distance is 1, 2 or 3 pulses, the pulses will simply be output at the configured Stop Frequency.
Acceleration	DWORD	Acceleration in Hz/ms or in ms (according to configuration). Range Hz/ms: 1...**Acc. max.** Range ms: **Acc. max.**...49999
Deceleration	DWORD	Deceleration in Hz/ms or in ms (according to configuration). Range Hz/ms: 1...**Dec. max.** Range ms: **Dec. max.**...49999
Direction	PTO_DIRECTION *(see page 115)*	Direction of the move (forward or backward).

The table describes the output variables:

Outputs	Type	Comment
PTO_REF_OUT	PTO_REF *(see page 117)*	Reference to the PTO channel. To be connected with the PTO_REF_IN input pins of other PTO function blocks.
Done	BOOL	TRUE = indicates that the command is finished. Function block execution is finished.
Busy	BOOL	TRUE = indicates that the command is in progress.
Active	BOOL	This output is set at the moment the function block takes control of the motion of the axis.
CommandAborted	BOOL	TRUE = indicates that the command was aborted due to another move command. Function block execution is finished.

Outputs	Type	Comment
Error	BOOL	TRUE = indicates that an error was detected. Function block execution is finished. **NOTE:** Errors must be reset before a new motion command is executed. Otherwise any new motion commands will be ignored.
ErrID	PTOPWM_ERR_TYPE *(see page 114)*	When Error is TRUE: type of the detected error.

NOTE: For more information about Done, Busy, CommandAborted and Execution pins, refer to General Information on Function Block Management *(see page 95)*.

Programming the PTOMoveRelative Function Block

Procedure

To program the `PTOMoveRelative` function block, do the following:

Step	Action
1	Select the **Libraries** tab in the **Software Catalog** and click **Libraries**. Select **Controller → HMISCU → HMISCU PTOPWM → PTORelative** in the list, drag-and-drop the item onto the `Start Here` box in the lower **POU** window.
2	Declare the function block instance.
3	Associate the `PTO_REF_IN` input of the function block to the `PTO_REF` output of the `PTOSimple` function block.
4	The inputs/outputs are detailed in the function block description PTOMoveRelative *(see page 40)*. The interactions between the inputs/outputs are detailed in the General Information *(see page 93)*. The interactions between the motion commands are detailed in the Command Sequence *(see page 59)*.

An aborted motion command cannot be completed after having been stopped. A new motion command must be issued.

When the movement is launched, it cannot be changed (only aborted) while its profile execution is not complete.

The `PTOMoveRelative` movement is aborted when:
- A `PTOStop` function block is called.
- The Drive Ready Input (if defined at configuration time) becomes inactive.
- The sequence of commands is not supported.
- The application is stopped.
- An error is detected.

Pulse and Time Calculations with PTOMoveRelative

Overview

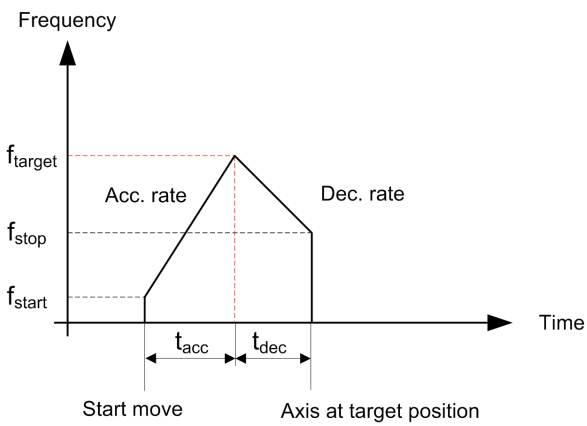
When using **PTOMoveRelative**, the total number of pulses is respected unless interrupted.

Therefore, it is important to note that there are three possible movement profile cases depending on your parameters:
- The minimum number of pulses required to reach the target frequency is met exactly.
- When the total pulses specified is greater than the number of minimum pulses required to reach the target frequency (trapezoidal profile).
- When the total pulses specified is less than the minimum number of pulses required to reach the target frequency.

Case 1: Minimum Number of Pulses

The distance input enables you to specify the movement from the current position of the axis to the target position. The distance input is the number of pulses that are required to perform the movement. The parameters defined by you can define the minimum number of pulses required to meet the target velocity. The distance (for example, the number of pulses) corresponds to the area under the frequency (for example, velocity) profile.

The axis follows this profile:



If we consider the limit case where the target frequency is reached at only one point, then the profile follows a triangular profile.

The minimum number of pulses P_{min} is then defined as:

$$P_{min} = f_{start} \times t_{acc} + f_{stop} \times t_{dec} + \frac{(f_{target} - f_{start})t_{acc}}{2} + \frac{(f_{target} - f_{stop})t_{dec}}{2}$$

f_{start} start frequency
f_{stop} stop frequency
f_{target} velocity target
t_{acc} acceleration time (1)
t_{dec} deceleration time (2)

NOTE:
(1) If you have defined an acceleration (a) instead of acceleration time (t_{acc}) then the following formula applies:

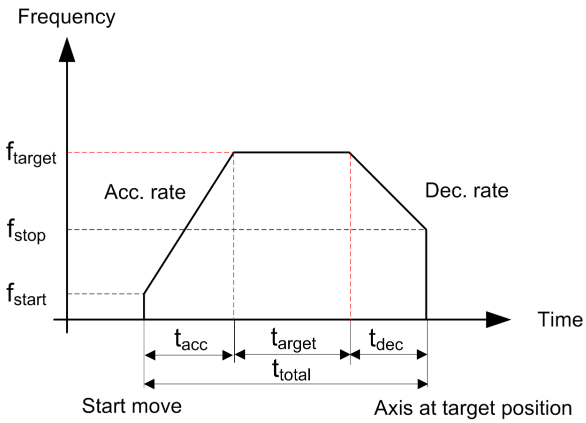
$$t_{acc} = \frac{f_{target} - f_{stop}}{a}$$

(2) If you have defined a deceleration rate (d) instead of deceleration time (t_{dec}) then the following formula applies:

$$t_{dec} = \frac{f_{target} - f_{stop}}{d}$$

Case 2: Number of Pulses Greater than the Minimum (Trapezoidal Profile)

When you set a number of pulses greater than the minimum number of pulses required to perform the movement at the distance input, the velocity of the axis follows a trapezoidal profile:



In a trapezoidal profile you define:
- Acceleration time (t_{acc}) or acceleration rate (a) [(2)]
- Deceleration time (t_{dec}) or deceleration rate (d) [(2)]
- Frequency target (f_{target})
- Start frequency: (f_{start})
- Stop frequency: (f_{stop})
- Distance or number of pulses (P) [(1)]

From these parameters we can obtain:
- Time while in-velocity (t_{target})
- Total time of operation (t_{total})

NOTE:
- [(1)] In this case, the number of pulses is greater than or equal to the minimum number of pulses (refer to the Minimum Number of Pulses *(see page 44)*.
- [(2)] If acceleration/deceleration rates are defined, use the formula *(see page 44)* to obtain the acc/dec times in ms.

First, the calculation of the minimum number of pulses is required (Pmin) *(see page 44)*:

$$t_{target} = \frac{P - P_{min}}{f_{target}}$$

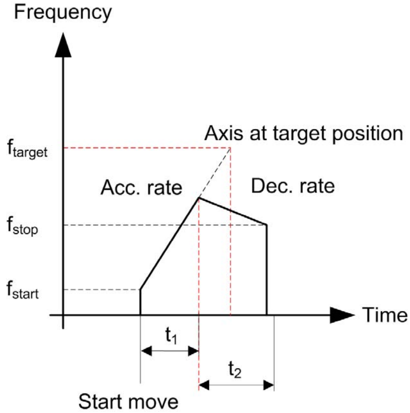
The total time of operation t_{total} is defined by:

$$t_{total} = t_{acc} + t_{target} + t_{dec}$$

Case 3: Number of Pulses Less than the Minimum

If you define a distance input less than the minimum number of pulses described in Minimum Number of Pulses *(see page 44)*, the target frequency is not reached. The HMI SCU firmware shortens the function block output acc/dec times (t_1 and t_2) and lowers the maximum frequency that can be reached (f_{max}).

The axis follows this profile:



In this profile:

- Recalculated acceleration time (t_1) (1)
- Recalculated deceleration time (t_2) (1)
- Frequency target (f_{target})
- Start frequency: (f_{start})
- Stop frequency: (f_{stop})
- Distance or number of pulses (P)

NOTE: (1) If milliseconds is chosen for the unit of acceleration and deceleration, then a = f_{target}/t_{acc} and d = f_{target}/t_{dec} is used when solving the system of equations.

You can obtain these three values (t_1, t_2 and f_{max}) by solving the following system:

$$\begin{cases} f_{max} = a \times t_1 + f_{start} \\ t_2 = \dfrac{f_{max} - f_{stop}}{d} \\ P_{target} = f_{start} \times t_1 + f_{stop} \times t_2 + \dfrac{(f_{max} - f_{start})t_1}{2} + \dfrac{(f_{max} - f_{stop})t_2}{2} \end{cases}$$

- For the system described above, the shortened acceleration time, t_1, is given by:

$$t_1 = \frac{-\beta + \sqrt{\Delta}}{2\alpha}$$

  Where:

$$\alpha = \frac{a \text{ x } d + a^2}{2d}$$

$$\beta = f_{start}\left(1 + \frac{a}{d}\right)$$

$$\Delta = f_{start}\left(1 + \frac{a}{d} f_{start}\right) + \frac{a}{d} f^2{}_{stop}\left(1 + \frac{a}{d}\right) + 2aP\left(1 + \frac{a}{d}\right)$$

- The shortened deceleration time, t_2, is given by:

$$t_2 = \frac{a \text{ x } \frac{-\beta + \sqrt{\Delta}}{2\alpha} + f_{start} - f_{stop}}{d}$$

- The maximum reached frequency, f_{max}, is given by:

$$f_{max} = a \text{ x } \frac{-\beta + \sqrt{\Delta}}{2\alpha} + f_{start}$$

NOTE: If the new f_{max} ≤ either the **Start Frequency** or the **Stop Frequency**, a PTO error is detected and no motion control is started.

NOTE: If the **Distance** = 1, 2 or 3 pulses. The pulses are output at the configured **Stop Frequency**. This is useful for manual positioning by jogging a motor.

Section 5.2
Move speed: PTOMoveVelocity

Overview

This section describes the `PTOMoveVelocity` function block.

What Is in This Section?

This section contains the following topics:

Description

Overview

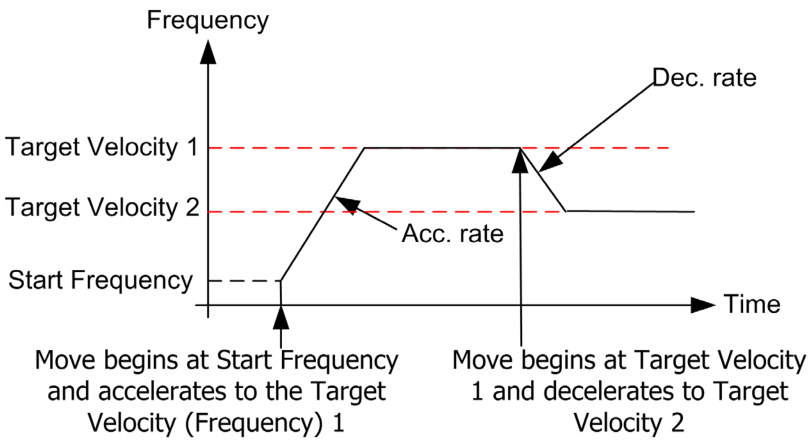
Speed control is a reference to the control of the motor velocity.To control the speed of the motor associated with the PTO channel, use the `PTOMoveVelocity` function block.

The `PTOMoveVelocity` function block is used to generate a pulse train output at a specified frequency (velocity) through an acceleration or deceleration ramp.

When the `PTOMoveVelocity` command is executed, the current Motion State *(see page 60)* is Continuous Motion.

A target velocity of 0 Hz is not allowed in `PTOMoveVelocity` for HMISCU.

The graph below illustrates two consecutive `PTOMoveVelocity` commands:



In order to stop the continuous motion, execute the `PTOStop` *(see page 56)* command.

PTOMoveVelocity Function Block

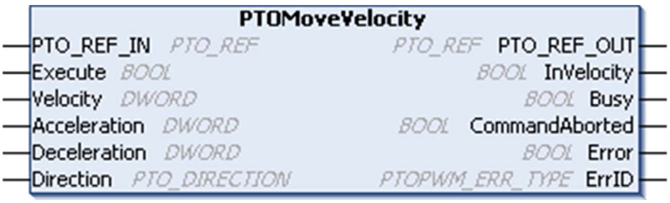
Function Description

This function block commands a continuous move at a specified velocity.

This velocity is reached according to specified acceleration and deceleration values.

A new `PTOMoveVelocity` motion command with new velocity and acceleration/deceleration values can be issued while the axis is in motion when the last specified velocity is reached.

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 103)*.

I/O Variables Description

This table describes the input variables:

Inputs	Type	Comment
`PTO_REF_IN`	`PTO_REF` *(see page 117)*	Reference to the PTO axis. To be connected to the `PTO_REF` of the **PTOSimple** or the `PTO_REF_OUT` of the PTO function blocks.
`Execute`	`BOOL`	On rising edge, starts the function block execution. When `FALSE`, resets the outputs of the function block when its execution terminates.
`Velocity`	`DWORD`	Target velocity in Hz. Range: 1...`Frequency max` **NOTE:** The `Velocity` cannot be less than the `Start Frequency` when executing **PTOMoveVelocity** from a stopped axis. The `Velocity` can be set to a frequency less than the `Start Frequency` only if the axis is already in motion at a constant frequency from a previous **PTOMoveVelocity** motion command.

Inputs	Type	Comment
`Acceleration`	`DWORD`	Acceleration in Hz/ms or in ms (according to configuration). Range Hz/ms: 1...**Acc. max.** Range ms: **Acc. max.**...49999
`Deceleration`	`DWORD`	Deceleration in Hz/ms or in ms (according to configuration). Range Hz/ms: 1...**Dec. max.** Range ms: **Dec. max.**...49999
`Direction`	`PTO_DIRECTION` *(see page 115)*	Direction of the move.

NOTE: The acceleration and deceleration ramps cannot exceed 4,294,967,295 pulses. At the maximum frequency of 50 kHz, it would limit the duration of acc/dec ramps to 40 seconds.

NOTE: For a new **PTOMoveVelocity** motion command when the axis is in motion, the specified velocity from the previous motion command must be reached (`InVelocity =TRUE`). Executing **PTOMoveVelocity** during acceleration or deceleration phase (while `Busy =TRUE`) will abort the command and stop the axis at the configured `Dec. Fast Stop` rate.

This table describes the output variables:

Outputs	Type	Comment
`PTO_REF_OUT`	`PTO_REF` *(see page 117)*	Reference to the PTO channel. To be connected with the `PTO_REF_IN` input pin of the PTO function blocks.
`InVelocity`	`BOOL`	`TRUE` = indicates that target velocity is reached. The move is ongoing and the last function block command execution is finished.
`Busy`	`BOOL`	`TRUE` = indicates that the command is in progress.
`CommandAborted`	`BOOL`	`TRUE` = indicates that the command was aborted due to another move command. Function block execution is finished.
`Error`	`BOOL`	`TRUE` = indicates that an error was detected. Function block execution is finished.
`ErrID`	`PTOPWM_ERR_TYPE` *(see page 114)*	When `Error` is `TRUE`: type of the detected error.

NOTE: For more information about Done, Busy, CommandAborted and Execution pins, refer to General Information on Function Block Management *(see page 95)*.

Programming the PTOMoveVelocity Function Block

Procedure

To program the `PTOMoveVelocity` function block, do the following:

Step	Action
1	With the **Input Assistant**, add the **PTOMoveVelocity** function block from the following path: **Function Block (Libraries)** → **HMISCU_PTOPWM** → **PTOMoveVelocity** and click **OK**
2	Declare the function block instance.
3	Associate the `PTO_REF_IN` input of the function block to the `PTO_REF` output of the `PTOSimple` function block.
4	The inputs/outputs are detailed in the function block description PTOMoveVelocity *(see page 51)*. The interactions between the inputs/outputs are detailed in the General Information *(see page 93)*. The interactions between the motion commands are detailed in the Command Sequence *(see page 59)*.

Any aborted motion commands cannot be completed after having been stopped. A new motion command must be issued.

The `PTOMoveVelocity` movement is aborted when:

- a `PTOStop` function block is called
- the **Drive Ready** Input (if defined at configuration time) becomes inactive
- the sequence of commands is not supported
- the application is stopped
- an error is detected
- a new `PTOMoveVelocity` motion command is issued while the current `PTOMoveVelocity` command is currently in `Acceleration` or `Deceleration` phase (while `Busy =TRUE`)

Section 5.3
Stopping the Movement: PTOStop

Overview

This section describes the `PTOStop` function block.

What Is in This Section?

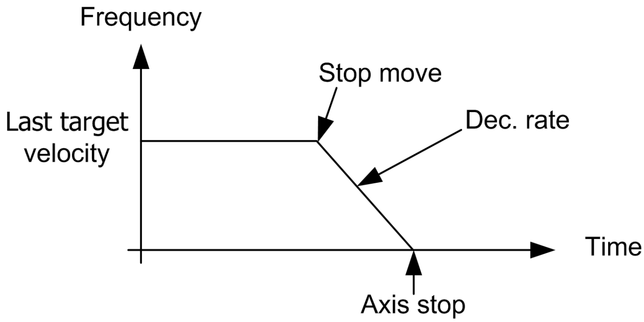
This section contains the following topics:

Description

Overview

This function block commands a controlled stop of the axis (deceleration to stop), and aborts any motion ongoing.

After the axis has been completely stopped, a new motion is not allowed as long as the `Execute` input remains `TRUE` or an axis error was detected and has not been reset *(see page 32)*.



NOTE: If the **PTOStop** function block is executed, the last output pulse frequency will be the **Stop Frequency**. For more information, see deceleration pulses calculation *(see page 96)*.

PTOStop Function Block

Function Description

This function block commands a controlled stop of the axis (deceleration to stop), and aborts any motion ongoing.

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 103)*.

I/O Variables Description

This table describes the input variables:

Inputs	Type	Comment
PTO_REF_IN	PTO_REF *(see page 117)*	Reference to the PTO channel. To be connected to the PTO_REF of the **PTOSimple** or the PTO_REF_OUT of the PTO function blocks.
Execute	BOOL	On rising edge, starts the function block execution. When FALSE, resets the outputs of the function block when its execution terminates.
Deceleration	DWORD	Deceleration in Hz/ms or in ms (according to configuration). Range for rate (in Hz/ms): 1...**Dec. max.** Range for time (in ms): **Dec. max.**...49999

This table describes the output variables:

Outputs	Type	Comment
PTO_REF_OUT	PTO_REF *(see page 117)*	Reference to the PTO channel. To be connected with the PTO_REF_IN input pin of the PTO function blocks.
Done	BOOL	TRUE = indicates that the command is finished. Function block execution is finished.

Outputs	Type	Comment
Busy	BOOL	TRUE = indicates that the command is in progress.
Error	BOOL	TRUE = indicates that an error was detected. Function block execution is finished.
ErrID	PTOPWM_ERR_TYPE *(see page 114)*	When Error is TRUE: type of the detected error.

NOTE: For more information about Done, Busy, CommandAborted and Execution pins, refer to General Information on Function Block Management *(see page 95)*.

Programming the PTOStop Function Block

Procedure

To program the `PTOStop` function block, do the following:

Step	Action
1	Select the **Libraries** tab in the **Software Catalog** and click **Libraries**. Select **Controller** → **HMISCU** → **HMISCU PTOPWM** → **PTOStop** in the list, drag-and-drop the item onto the **POU** window.
2	Declare the function block instance.
3	Associate the `PTO_REF_IN` input of the function block to the `PTO_REF` output of the `PTOSimple` function block. **NOTE:** A unique `PTOSimple` instance is needed per PTO Channel in the application.
4	The inputs/outputs are detailed in the function block. The interactions between the inputs/outputs are detailed in the General Information *(see page 93)*. The interactions between the motion commands are detailed in the Command Sequence *(see page 59)*.

Section 5.4
Command Sequence

Overview

This section describes the allowed sequence of commands.

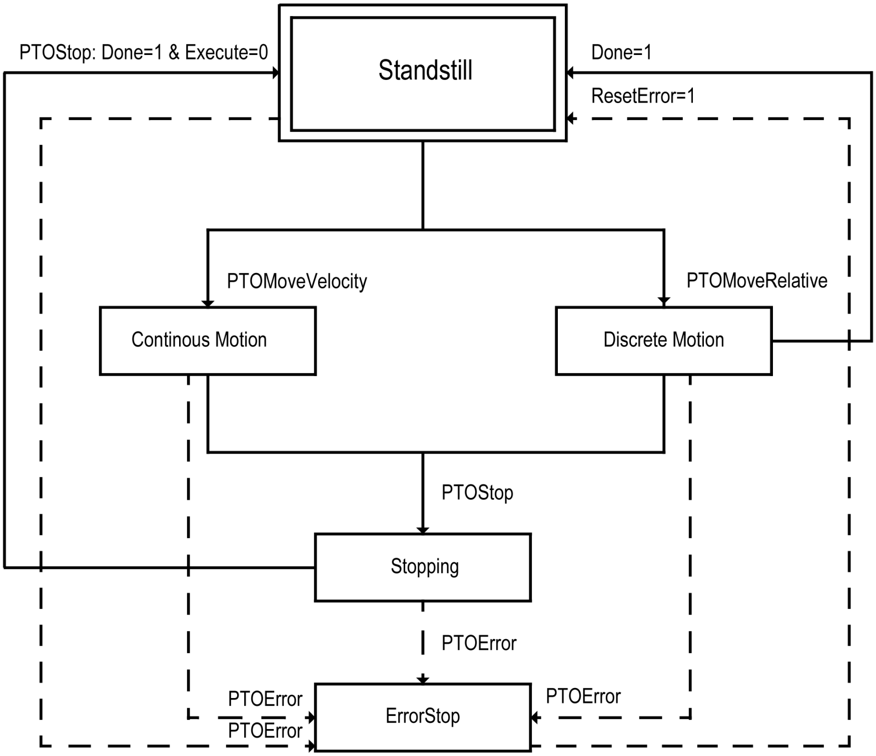
What Is in This Section?

This section contains the following topics:

Motion State Diagram

State Diagram

Any sequence of motion commands must respect the following state diagram:



Allowed Sequence of Commands

Description

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

		Next Motion Command		
		PTOMoveRelative	**PTOMoveVelocity**	**PTOStop**
Current Motion Command	**PTOMoveRelative**	Command Aborted	Command Aborted	Immediate Accept
	PTOMoveVelocity (during constant velocity phase) `InVelocity = TRUE`	Command Aborted	Acceleration/Deceler-ation to New Velocity	Immediate Accept
	PTOMoveVelocity (during acceleration/deceleration phase) `Busy = TRUE`	Command Aborted	Command Aborted	Immediate Accept
	PTOStop (during deceleration) `Busy = TRUE`	Axis Error	Axis Error	Axis Error

Command Aborted: Function block of the previously running motion command reports `CommandAborted=TRUE`. The last function block to be triggered will return `PTO_INVALID_PARAMETER`. The current motion that is output will decelerate at the `Dec. Fast Stop` rate.

Immediate Accept: The new command is accepted immediately and the current command is aborted. Previously running function block will report `Done`. Second motion command will report `Done` when its motion command is complete.

Axis Error: In the case of executing another motion command while another `PTOStop` motion command is active, the both function blocks will report `PTO_AXIS_ERROR` and the axis will decelerate to 0 Hz respecting the configured `Dec. Fast Stop` rate.

Acceleration/Deceleration to New Velocity: The command is accepted and the axis is accelerated or decelerated to the new desired Velocity at the Acceleration/Deceleration rate or time defined in the **PTOMoveVelocity** inputs.

NOTE: If a second motion command is executed and any of the parameters are invalid or out of range, the above table is still valid, except that the second motion command function block will return `PTO_INVALID_PARAMETER`.

NOTE: If the current command is aborted and the new (next) command is accepted, there will be a slight delay before the new (next) command starts. This delay will generate additional PTO pulses that may affect the precision of the new (next) move command. These additional pulses can only be determined empirically because their number depends on the application (move velocity, task size, task type, and any other tasks that may intervene). Therefore, you will need to test your application thoroughly to determine the number of additional pulses that occur when one move is interrupted by another, and then to take into account this discrepancy within your application.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

- Thoroughly test your application to determine the number of extra pulses that are generated when one move instruction is interrupted by another move instruction.
- Take into account the number of extra pulses that are generated when one move instruction is interrupted by another move instruction within your application.

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

Chapter 6
Administrative Commands

Overview

This chapter describes the administrative function blocks to adjust and diagnose a PTO function.

What Is in This Chapter?

This chapter contains the following sections:

Section 6.1
Adjusting

Overview

This chapter describes the reading and writing of PTO parameters.

What Is in This Section?

This section contains the following topics:

Description

Overview

Two function blocks can be used to adjust PTO parameters:

- PTOGetParam *(see page 66)*, allows you to read the parameters
- PTOSetParam *(see page 68)*, allows you to write the parameters

Adjustable Parameters

The PTOGetParam *(see page 66)* and `PTOSetParam`function blocks *(see page 68)* allow a program to read and write the following parameters:

- Start Frequency
- Stop Frequency

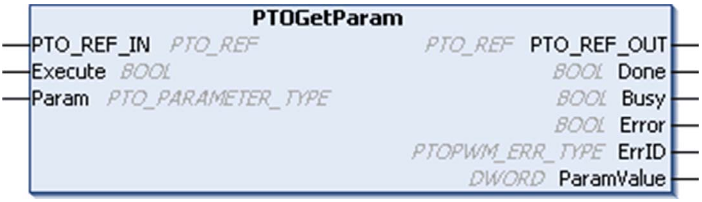
NOTE: Parameters you set via your program overwrite the initial parameters values configured in the PTO configuration window. However, initial configuration parameters are restored on cold or warm start of the controller.

PTOGetParam Function Block

Function Description

This function block returns the value of a specific parameter for a specified PTO channel.

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 103)*.

I/O Variables Description

This table describes the input variables:

Inputs	Type	Comment
PTO_REF_IN	PTO_REF *(see page 117)*	Reference to the PTO channel. To be connected to the PTO_REF of the **PTOSimple** or the PTO_REF_OUT of the PTO function blocks.
Execute	BOOL	On a rising edge, starts the function block execution. When FALSE, does not reset the outputs of the function block. The output pins of this function block always show the last status until next rising edge of the Execution input.
Param	PTO_PARAMETER_TYPE *(see page 116)*	Parameter to read.

This table describes the output variables:

Outputs	Type	Comment
PTO_REF_OUT	PTO_REF (see page 117)	Reference to the PTO channel. To be connected with the PTO_REF_IN input pin of the PTO function blocks.
Done	BOOL	TRUE = indicates that the ParamValue is valid. Function block execution is finished.
Busy	BOOL	TRUE = indicates that the function block execution is in progress.
Error	BOOL	TRUE = indicates that an error was detected. Function block execution is finished.
ErrID	PTOPWM_ERR_TYPE (see page 114)	When Error is TRUE: type of detected error.
ParamValue	DWORD	When Done is TRUE: Parameter value is valid.

NOTE: For more information about Done, Busy, CommandAborted and Execution pins, refer to General Information on Function Block Management (see page 95).

PTOSetParam Function Block

Function Description

This function block modifies the value of a specific parameter for a specified PTO channel.

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 103)*.

I/O Variables Description

This table describes the input variables:

Inputs	Type	Comment
`PTO_REF_IN`	`PTO_REF` *(see page 117)*	Reference to the PTO channel. To be connected to the `PTO_REF` of the **PTOSimple** or the `PTO_REF_OUT` of the PTO function blocks.
`Execute`	`BOOL`	On a rising edge, starts the function block execution. When `FALSE`, does not reset the outputs of the function block. The output pins of this function block always show the last status until next rising edge of the `Execution` input.
`Param`	`PTO_PARAMETER_TYPE` *(see page 116)*	Parameter to set.
`Param_Value`	`DWORD`	Parameter value to write.

This table describes the output variables:

Outputs	Type	Comment
PTO_REF_OUT	PTO_REF *(see page 117)*	Reference to the PTO channel. To be connected with the PTO_REF_IN input pin of the PTO function blocks.
Done	BOOL	TRUE = indicates that ParamValue is valid. Function block execution is finished.
Busy	BOOL	TRUE = indicates that the function block execution is in progress.
Error	BOOL	TRUE = indicates that an error was detected. Function block execution is finished.
ErrID	PTOPWM_ERR_TYPE *(see page 114)*	When Error is set: type of detected error.

NOTE: For more information about Done, Busy, CommandAborted and Execution pins, refer to General Information on Function Block Management *(see page 95)*.

Programming the PTOGetParam or PTOSetParam Function

Procedure

To program the `PTOGetParam` or the `PTOMoveFast` function block, do the following:

Step	Action
1	Select the **Libraries** tab in the **Software Catalog** and click **Libraries**. Select **Controller** → **HMISCU** → **HMISCU PTOPWM** → **PTOGetParam** or **PTOSetParam** in the list, drag-and-drop the item onto the **POU** window.
2	Declare the function block instance.
3	Associate the `PTO_REF_IN` input of the function block to the `PTO_REF` output of the `PTOSimple` function block. **NOTE:** A unique `PTOSimple` instance is needed per PTO Channel in the application.
4	The inputs/outputs are detailed in the function blocks PTOGetParam *(see page 66)* or PTOSetParam *(see page 68)*. The interactions between the inputs/outputs are detailed in the General Information *(see page 93)*.

Section 6.2
Diagnostics

Overview

This chapter describes the function block available to diagnose the specified PTO channel.

What Is in This Section?

This section contains the following topics:

PTOGetDiag Function Block

Function Description

This function block returns the PTO detected error code.

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 103)*.

I/O Variables Description

This table describes the input variables:

Inputs	Type	Comment
PTO_REF_IN	PTO_REF *(see page 117)*	Reference to the PTO channel. To be connected to the PTO_REF of the **PTOSimple** or the PTO_REF_OUT of the PTO function blocks.
Execute	BOOL	On rising edge, starts the function block execution. When FALSE, resets the outputs of the function block when its execution terminates.

This table describes the output variables:

Outputs	Type	Comment
PTO_REF_OUT	PTO_REF *(see page 117)*	Reference to the PTO channel. To be connected with the PTO_REF_IN input pin of the PTO function blocks.
Done	BOOL	TRUE = indicates that PTODiag is valid. Function block execution is finished.
Busy	BOOL	TRUE = indicates that the function block execution is in progress.

Outputs	Type	Comment
`Error`	`BOOL`	`TRUE` = indicates that an error was detected. Function block execution is finished.
`ErrID`	`PTOPWM_ERR_TYPE` *(see page 114)*	When `Error` is `TRUE`: type of the detected error.
`PTODiag`	`DWORD`	When `Done` is `TRUE`: Diagnostic error code (see table below).

DWORD bit	Meaning
0...3	Not used
4	Internal error detected
5...6	Not used
7	Configuration error detected
8...16	Not used
17	Drive not ready (auxiliary input DriveReady is `FALSE`)
18...20	Not used
21	Reserved
22	Invalid Frequency
23	Invalid Acceleration
24	Invalid Deceleration
25	Command rejected (PTO_AXIS_ERROR, or new PTO command is triggered before the last operation is completed)
26	Invalid Direction
27...31	Not used

NOTE: For more information about Done, Busy, CommandAborted and Execution pins, refer to General Information on Function Block Management *(see page 95)*.

Programming the PTOGetDiag Function Block

Procedure

You can use the `PTOGetDiag` function to determine the reason an error was detected during the execution of a PTO function block.

To implement the `PTOGetDiag` function, do the following:

Step	Action
1	Select the **Libraries** tab in the **Software Catalog** and click **Libraries**. Select **Controller** → **HMISCU** → **HMISCU PTOPWM** → **PTOGetDiag** in the list, drag-and-drop the item onto lower the **POU** window on the `Start here`.
2	Declare the function block instance.
3	Associate the `PTO_REF_IN` input of the function block to the `PTO_REF` output of the `PTOSimple` function block. **NOTE:** A unique `PTOSimple` instance is needed per PTO Channel in the application.
4	The inputs/outputs are detailed in the function block *(see page 72)*. The interactions between the inputs/outputs are detailed in the General Information *(see page 93)*.

Detected Error Management

Overview

Mainly 6 type of errors can be encountered when running a PTO. They are reported in `ErrID` pin of the PTOGetDiag *(see page 72)* function block.

- PTO_NO_ERROR
- PTO_UNKNOWN_REF
- PTO_UNKNOWN_PARAMETER
- PTO_INVALID_PARAMETER
- PTO_AXIS_ERROR
- PTO_COM_ERROR

PTO_INVALID_PARAMETER

This error occurs in the following situations:

- Invalid Frequency
- Invalid Acceleration
- Invalid Deceleration
- Invalid Distance
- Invalid Position
- Invalid Direction
- Reverse Direction
- Profile Error

The detail of the error is identified by calling the PTOGetDiag *(see page 72)* function block.

When this error occurs, the following behavior is also induced:

- The axis is put in ErrorStop state (`PTOError` = 1; `ErrID` = PTO_INVALID_PARAMETER).
- Any command in progress or in buffer will be aborted.
- If any command is being executed, the axis stops using the adjusted **Dec. Emergency stop** rate.

No other command is accepted before the axis is stopped and the axis error is reset through the `Reset_error` pin of the `PTOSimple` *(see page 32)* function block.

PTO_AXIS_ERROR

This error occurs in the following situations:

- Internal error detected
- Drive not ready
- Command Rejected
- FastPTO stop exception

The detail of the error is identified by calling the PTOGetDiag *(see page 72)* function block.

When this error occurs, the following behavior is also induced:

- The axis is put in ErrorStop state (`PTOError` = 1; `ErrID` = AXIS_ERROR).
- Any command in progress or in buffer will be aborted.
- If any command is being executed, the axis stops using the adjusted **Dec. Fast Stop** rate.

No other command is accepted before the axis is stopped and the axis error is reset through the `Reset_error` pin of the `PTOSimple` *(see page 32)* function block.

PTO_UNKNOWN_REF

This error will appear when a PTO function block is assigned to an incorrect or empty Axis Reference to its `PTO_REF_IN` input pin.

NOTE: This will be detected by SoMachine Editor during project compilation.

PTO_UNKOWN_PARAMETER

This error occurs for `PTOGetParam` and `PTOSetParam` when a parameter input value other than 00 (`Start Frequency`) or 01 (`Stop Frequency`) is entered and the function block is executed.

PTO_COM_ERROR

This is an error in communication between CoDeSys control and the **IO Firmware** of the controller. If there is a physical interruption within the equipment, it will cause an error in communication between the two modules.

Part III
Pulse Width Modulation

Overview

This part describes the Pulse Width Modulation (PWM) function blocks.

What Is in This Part?

This part contains the following chapters:

Chapter 7
Generalities

Overview

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

The functions provide simple, yet powerful solutions for your application. In particular, they are 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 PTO/PWM 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:

PWM Naming Convention

Definition

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

In this document, use the following naming convention:

Name	Description
SYNC	Synchronization function *(see page 82)*.
EN	Enable function *(see page 82)*.
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 PWM.

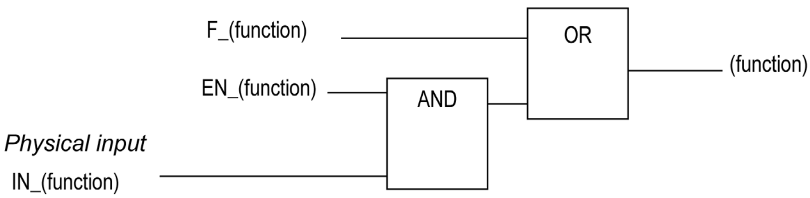
Synchronization and Enable Functions

Introduction

This section presents the functions used by the 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:



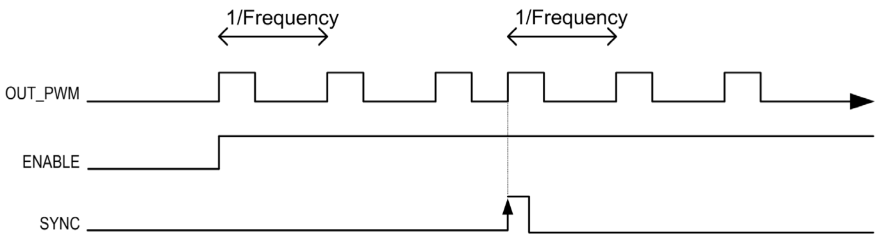
NOTE: (function) stands either for **Enable** (for Enable function) or **Sync** (for Synchronization function).
If the physical input is required, enable it in the configuration screen.

Synchronization Function

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

Enable Function

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



Chapter 8
Pulse Width Modulation (PWM)

What Is in This Chapter?

This chapter contains the following topics:

Description

Overview

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

NOTE: The functionality must be enabled either by setting `F_Enable` to 1, or by an external event with the `IN_EN` input and `EN_Enable`=1, otherwise the output (OUT_PWM) stays to 0.

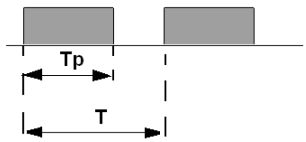
The PTO and PWM functions use the same dedicated outputs. Only one out of these 2 functions can be used on the same channel. Using different functions on channel 0 and channel 1 is allowed.

Signal Form

The signal form depends on the following input parameters:

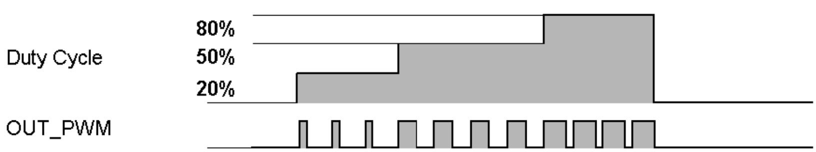
- **Frequency** configurable from 10 Hz to 65 kHz with a 0.1 Hz step
- **Duty Cycle** of the output signal from 0% to 100%

Duty Cycle=Tp/T



Tp pulse width
T pulse period (1/Frequency)

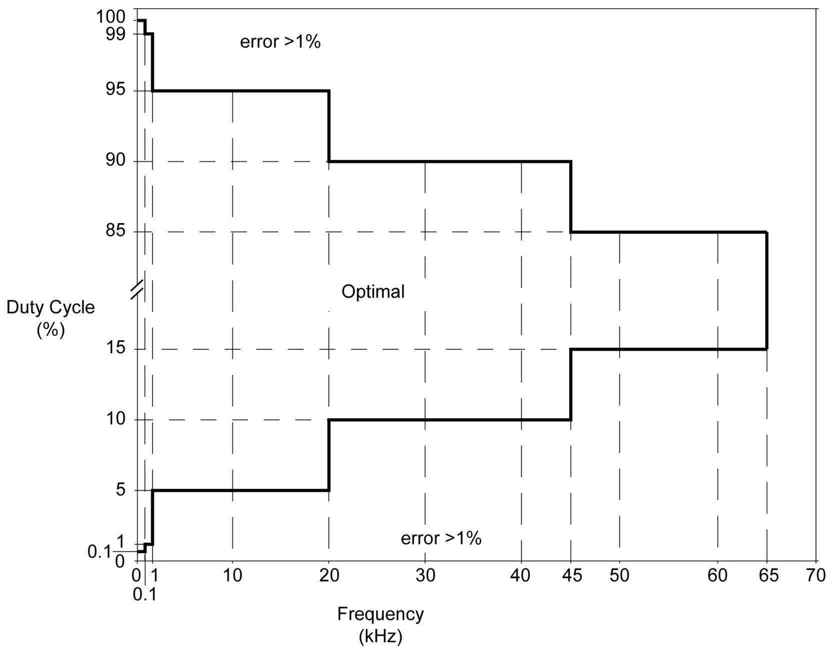
Modifying the duty cycle in the program modulates the width of the signal. Below is an illustration of an output signal with varying duty cycles.



The table describes the characteristics of the PWM outputs:

Characteristic		Value	
Accuracy / PWM mode	Frequency	Duty	Error
	10...100 Hz	0...100%	0.1%
	101...1000 Hz	1...99%	1%
	1.001...20 kHz	5...95%	5%
	20.001...45 kHz	10...90%	10%
	45.001...65 kHz	15...85%	15%
PWM mode duty rate step		20 Hz...1 kHz for 0.1%	
Duty rate range		1...99%	

When duty cycle is below 5% or above 95%, depending on the frequency, the error is above 1% as illustrated in the graphic below:



Pulse Width Modulation Configuration

Overview

2 PWM channels can be configured on the controller.

NOTE:
- Each PWM uses 1 fast output and 2 normal digital inputs.
- PTO or PWM can be configured on channel 0.
- If PTO is selected for channel 0, PWM on channel 1 cannot be used.
- Only PWM can be configured on channel 1.

Open the Configuration Window

Use this procedure to open the PWM configuration window:

Step	Action
1	In the **Devices tree**, double-click **HMISCUxx5** → **Embedded Functions** → **PTO_PWM**.
2	Select a **PTO 0** tab.
3	In the **PTO / PWM** → **PTO00** → **Mode** drop-down menu, select **PWM**. Result: The **PTO 0** tab **Variable** becomes **PWM00** and the **PTO 1** tab **Variable** becomes **PWM01**.

Configuration Window Description

This table describes each parameter available when the embedded PTO_PWM is configured in **PWM** mode:

Parameter	Value	Default Value	Unit	Description
Mode	**Not Used** **PTO** [1] **PWM** **Frequency Generator**	**Not Used**	–	The mode selected is PWM. **NOTE:** [1] This is not an option in PTO 1 tab. It is only supported for PTO 0 tab.

Parameter		Value	Default Value	Unit	Description
Auxiliary Inputs	EN	Disabled Enabled	Disabled	–	Enables the EN physical input to be used for enabling the functionality.
	EN Filter	3 12	3	ms	Defines the value of the EN filter value.
	SYNC	Disabled Enabled	Disabled	–	Enables the IN_SYNC input to be used for synchronization.
	SYNC Filter	3 12	3	ms	Defines the value of the IN_SYNC filter value.
	SYNC Edge	Rising Edge Falling Edge Both Edges	Rising Edge	–	Defines the IN_SYNC edge on which synchronization occurs.

Configure a PWM Channel

Use this procedure to configure a PWM channel:

Step	Action
1	Enable the PWM channel: Set the **Mode** parameter to **PWM**. **Result:** SoMachine creates a variable named PWM00 or PWM01 depending on the selected channel. **NOTE:** You can rename the variable by entering a new name in the **Variable** field. When creating a PWM function block, you can find the **Global Variable** name representing the channel from the **Input Assistant** when naming the block.
2	In the list box of EN parameter, enable/disable the EN Filter physical input.
3	Configure the filter value of the EN Filter input (if enabled in step 2).
4	In the list box of SYNC parameter, enable/disable the SYNC Filter physical input.
5	Configure the filter value of the SYNC Filter input (if enabled in step 4).
6	Configure the edge (rising or falling) for SYNC Edge signal detection (if enabled in step 4).

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

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 103)*.

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 = forces the Enable function.
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 PWM output signal (range: min 100 (10 Hz)...max 650,000(65 kHz)).
Duty	BYTE	Duty cycle of the Pulse Width Modulation output signal in % (range: min 0...max 100).

Output Variables

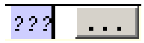
This table describes the output variables:

Outputs	Type	Comment
InFrequency	BOOL	TRUE = the Pulse Width Modulation signal is currently being output at the specified frequency and duty cycle.
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 Generator signal is not output at the specified Frequency. Reset to FALSE when InFrequency or Error is set, or when the Enable command is reset. When a command change execution is immediate, Busy remains FALSE.
Error	BOOL	TRUE = indicates that an error was detected.
ErrID	PTOPWM_ERR_TYPE *(see page 114)*	When Error is set: type of the detected error.

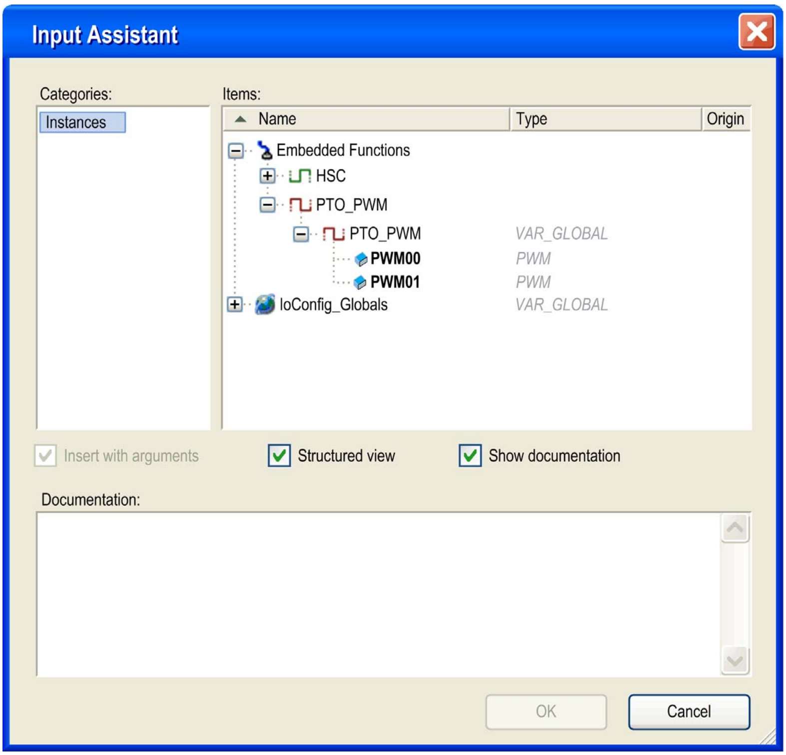
Programming the PWM Function Block

Procedure

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

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>Select the Libraries tab in the Software Catalog and click Libraries.
Select Controller → HMISCU → HMISCU PTOPWM → PWM in the list, drag-and-drop the item onto the POU window.</td></tr>
<tr><td>2</td><td>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 86) during the configuration and confirm.



NOTE: If the function block instance is not visible, verify if the PWM is configured.</td></tr>
<tr><td>3</td><td>The inputs/outputs are detailed in the function block (see page 88).
The interaction between the inputs/outputs are detailed in the General Information (see page 93).</td></tr>
</table>

Appendices

Overview

This appendix extracts parts of the programming guide for technical understanding of the library documentation.

What Is in This Appendix?

The appendix contains the following chapters:

Appendix A
General Information

Overview

The information described in this chapter is common for PTO and HSC features.

What Is in This Chapter?

This chapter contains the following topics:

Dedicated Features

Dedicated Outputs

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

⚠ WARNING
UNINTENDED EQUIPMENT OPERATION ● Do not use the same instance of a function block in more than 1 task. ● Do not modify function block references (••_REF_IN) while the function block is active (executing). **Failure to follow these instructions can result in death, serious injury, or equipment damage.**

General Information on Administrative and Motion Function Block Management

Management of Input Variables

At the `Execute` input rising edge, the function block starts.

Any further modifications of the input variables are not taken into account.

Following the IEC 61131-3 standards, if any variable input to a function block is missing, that is, left open or unconnected, then the value from the previous invocation of the instance of the function block will be used. In the first invocation, the initial, configured value is applied in this case. Therefore, it is best that a function block always has known values attributed to its inputs to help avoid difficulties in debugging your program. For HSC and PTO function blocks, it is best to use the instance only once, and preferably the instance be in the main task.

Management of Output Variables

The `Done`, `InVelocity`, or `InFrequency` output is mutually exclusive with `Busy`, `CommandAborted`, and `Error` outputs: only one of them can be TRUE on one function block. If the `Execute` input is TRUE, one of these outputs is TRUE.

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

The `Done`, `InVelocity`, or `InFrequency` output is set when the function block execution has been completed successfully.

When a function block execution is interrupted by another one, the `CommandAborted` output is set instead.

When a function block execution ends due to a detected error, the `Error` output is set and the detected error number is given through the `ErrId` output.

The `Done`, `InVelocity`, `InFrequency`, `Error`, `ErrID`, and `CommandAborted` outputs are reset with the falling edge of `Execute`. If `Execute` input is reset before the execution is finished, then the outputs are set for one task cycle at the execution ending.

When an instance of a function block receives a new `Execute` before it is finished, the function block does not return any feedback, such as `Done`, for the previous action.

Handling a Detected Error

All blocks have 2 outputs that can report a detected error during the execution of the function block:
- `Error` = `TRUE` when an error is detected.
- `ErrID` When `Error` = `TRUE`, returns the detected error ID.

Acceleration and Deceleration Pulses Calculation

Overview Pulses Calculation

The HMISCU calculates the time between pulses during acceleration and deceleration in the cases of:

- PTOMoveVelocity
- PTOMoveRelative
- PTOStop
- Dec. Fast Stop

PTOMoveRelative: Acceleration Pulses Calculation

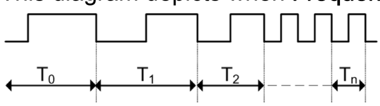
To calculate the period T_n (in seconds) between pulses during acceleration, the frequency f_n is rounded up to the nearest integer for that pulse period is calculated:

$$f_n = \text{Roundup to nearest integer}\left(\sqrt{\frac{a}{2}} \Big/ \left(\sqrt{n+1} - \sqrt{n}\right) \right)$$

$$T_n = \frac{1}{f_n}$$

a acceleration rate in Hz/s

This diagram depicts when **Frequency Start** = 0 Hz:



n is a positive integer representing the nth pulse period from the start of the acceleration phase.

PTOMoveRelative: Deceleration Pulses Calculation

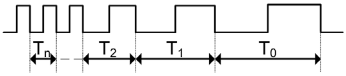
To calculate the period T_n (in seconds) between pulses during deceleration, the frequency f_n is rounded up to the nearest integer for that pulse period is calculated:

$$f_n = \text{Roundup to nearest integer}\left(\sqrt{\frac{d}{2}} \Big/ \left(\sqrt{n+1} - \sqrt{n}\right) \right)$$

$$T_n = \frac{1}{f_n}$$

d deceleration rate in Hz/s

This diagram depicts when **Frequency Start** = 0 Hz:



n is a positive integer representing the **n**th pulse period from the end of the deceleration phase.

PTOMoveRelative: Determining Acceleration Rate (a) and Deceleration Rate (d)

If the units of **Acc./Dec. Unit** is set to **ms**, the acceleration rate in **Hz/s** is:

$$a = \frac{\text{Target Frequency (in Hz)} - \text{Frequency Start (in Hz)}}{\text{Acceleration Time (in ms)}} \times 1000 \frac{\text{ms}}{\text{s}}$$

If the units of **Acc./Dec. Unit** are set to **ms**, the deceleration rate in **Hz/s** is:

$$d = \frac{\text{Target Frequency (in Hz)} - \text{Frequency Stop (in Hz)}}{\text{Deceleration Time (in ms)}} \times 1000 \frac{\text{ms}}{\text{s}}$$

The **Target Frequency** is value from the `Velocity` input pin from **PTOMoveRelative** function block.

The acceleration/deceleration time is the `Acceleration/Deceleration` input pins from the **PTOMoveRelative** function block.

If the units of **Acc./Dec. Unit** is set to **Hz/ms**, the acceleration/deceleration rate are that of the `Acceleration/Deceleration` pins on the **PTOMoveRelative** function block.

PTOMoveVelocity: Acceleration and Deceleration Pulses Calculation

If the units of **Acc./Dec. Unit** is set to **ms**, the acceleration rate from a motionless axis (current frequency = 0 Hz) in **Hz/s** is:

$$a = \frac{\text{Target Frequency (in Hz)} - \text{Frequency Start (in Hz)}}{\text{Acceleration Time (in ms)}} \times 1000 \frac{\text{ms}}{\text{s}}$$

When a new motion command is issued when the axis is currently in motion from a previous motion command:

- if the new velocity is **greater** than the previous velocity and if the units of **Acc./Dec. Unit** is set to **ms**, the acceleration rate from an axis currently in motion from a previous motion command to a in **Hz/s** is:

$$a = \frac{\text{Target Frequency2 (in Hz) - Target Frequency1 (in Hz)}}{\text{Acceleration Time (in ms)}} \times 1000 \frac{\text{ms}}{\text{s}}$$

- if the new velocity is **less** than the previous velocity and if the units of **Acc./Dec. Unit** is set to **ms**, the deceleration rate from an axis currently in motion from a previous motion command in **Hz/s** is:

$$d = \frac{\text{Target Frequency1 (in Hz) - Target Frequency2 (in Hz)}}{\text{Acceleration Time (in ms)}} \times 1000 \frac{\text{ms}}{\text{s}}$$

  Where:
  Target Frequency is value from the `Velocity` input pin from **PTOMoveVelocity** function block for a motion command that accelerates from a motionless axis (0 Hz).
  Target Frequency1 is the current constant velocity of the axis from a previous motion command.
  Target Frequency2 is the velocity target for the next motion command.
  The acceleration/deceleration time is the `Acceleration/Deceleration` input pins from the **PTOMoveVelocity** function block.

If the units of **Acc./Dec. Unit** is set to **Hz/ms**, the acceleration/deceleration rate are that of the `Acceleration/Deceleration` pins on the **PTOMoveVelocity** function block.

PTOStop: Determining Deceleration Rate (d)

If the units of **Acc./Dec. Unit** are set to ms, the deceleration rate in **Hz/s** is:

$$d = \frac{\text{Target Frequency (in Hz)} - \text{Frequency Stop (in Hz)}}{\text{Deceleration Time (in ms)}} \times 1000 \frac{\text{ms}}{\text{s}}$$

The **Target Frequency** is from the `Velocity` input pin from **PTOMoveRelative** or **PTOMoveVelocity** function block.

The deceleration time is the `Deceleration` input pin from the **PTOStop** function block.

If the units of **Acc./Dec. Unit** are set to **Hz/ms**, the deceleration rate is that of the `Deceleration` pin on the **PTOStop** function block.

Dec. Fast Stop

If the units of **Acc./Dec. Unit** are set to **ms**, the deceleration rate in **Hz/s** is:

$$d = \frac{\text{Maximum Frequency (in Hz)} - \text{Frequency Stop (in Hz)}}{\text{Dec.FastStop (in ms)}} \times 1000 \frac{\text{ms}}{\text{s}}$$

Maximum Frequency and **Dec. Fast Stop** are set in the PTO configuration user interface (or with **PTOSetParam** during program operation).

If the units of acceleration/deceleration are set to **Hz/ms**, the deceleration rate is that of the **Dec. Fast Stop** rate set in the PTO configuration user interface.

All Cases

NOTE: The minimum acceleration or deceleration rate is 1000 Hz/s (1 Hz/ms). If the calculated acceleration or deceleration rate is less than 1000 Hz/s, the rate used will be 1000 Hz/s (this case is only possible when **Frequency Start** or **Frequency Stop** is predefined to be > 0 Hz).

PTOStop Implementation with PTOMoveVelocity and PTOMoveRelative

Introduction

PTOStop does not follow the specified deceleration time when triggered during an acceleration/deceleration phase of **PTOMoveVelocity** or **PTOMoveRelative**. When executing **PTOStop** during acceleration or deceleration phase of **PTOMoveVelocity** or **PTOMoveRelative**, the time required for a complete stop will not adhere to the time specified in the **PTOStop** function block `Deceleration` input pin.

Instead, the HMI SCU applies the methods described below for **PTOStop** during acceleration and deceleration.

PTOStop can be triggered in a total possible 4 cases:

1. `Constant Frequency Output` when **PTOMoveVelocity** or **PTOMoveRelative** has reached its `Target Velocity`.
2. Acceleration phase of **PTOMoveVelocity** to a new higher `Target Velocity`.
3. Deceleration phase of **PTOMoveVelocity** to a new lower `Target Velocity`.
4. Acceleration phase or deceleration phase of **PTOMoveRelative**.

PTOStop Calculation

In all cases, the deceleration rate of **PTOStop** is recalculated using the `LAST Target Frequency` of **PTOMoveVelocity** or **PTOMoveRelative**:

$$\text{PTOStop Deceleration Rate} = \frac{\text{Target Velocity - Stop Frequency}}{\text{PTOStop Dec Time}}$$

In all cases, the time required for **PTOStop** to complete a stop is instantaneous:

$$\text{PTOStop Required Time} = \frac{\text{Current Instantaneous Frequency - Stop Frequency}}{\text{Calculated PTOStop Deceleration Rate}}$$

Case 1: PTOStop Command Is Issued During Constant Frequency Output When PTOMoveVelocity or PTOMoveRelative Has Reached Its Target Velocity

PTOStop will respect the deceleration time set in the function block. In this case, the `Current Instantaneous Frequency` is the same as the `Target Frequency`. Therefore the deceleration time set in the **PTOStop** function block's `Deceleration` input pin will be respected.

Case 2: PTOStop Command Is Issued During Acceleration Phase of PTOMoveVelocity to a New Higher Target Velocity

In this case, the `Target Velocity` is greater than the frequency being output at all times during acceleration. That means that the newly calculated rate of deceleration for **PTOStop** will result in the axis reaching 0 Hz within a shorter time than specified in the **PTOStop** `Deceleration` input pin.

Case 2: Example

The following actions take place in this sequence:

- `Stop Frequency` is configured to be 0 Hz.
- **PTOMoveVelocity** is executed for the first time with the `Target Velocity` = 10 kHz.
- The PTO output reaches 10 kHz.
- A new **PTOMoveVelocity** command is issued to accelerate to 50 kHz in 20 seconds.
- 10 seconds elapse. At this instance in time, the instantaneous output frequency is:

$$\left(\frac{(50000\text{Hz}-10000\text{Hz})}{20\text{s}}\right)\times(10\text{s})\times 10000\text{Hz}=30000\,\text{Hz}$$

- **PTOStop** command is issued with 10000 ms set in the `Deceleration` input pin. The deceleration rate of **PTOStop** will be calculated to be:

$$\text{PTOStop Deceleration Rate }=\left(\frac{(50000\text{Hz}-0\text{Hz})}{10\text{s}}\right)=500\text{Hz.s}^{-1}$$

Therefore, the time the **PTOStop** command is issued to the time the PTO output is stopped requires:

$$\text{PTOStop Required Time }=\frac{\text{Current Instantaneous Frequency - Stop Frequency}}{\text{Calculated PTOStop Deceleration Rate}}$$

$$\text{PTOStop Required Time }=\left(\frac{(30000\text{Hz}-0\text{Hz})}{5000\text{Hz.s}^{-1}}\right)=6\text{s}$$

NOTE: You will notice that 6 seconds is a shorter time than set in the original **PTOStop** command (10 s).

Case 3: Deceleration Phase of PTOMoveVelocity to a New Lower Target Velocity

In this case, the `Target Velocity` is greater than the frequency being output at all times during acceleration. That means that the newly calculated rate of `Deceleration` for **PTOStop** will result in the axis reaching 0 Hz within a greater time than specified in the `PTOStop Deceleration` input pin.

Case 3: Example

The following actions take place in this sequence:

- `Stop Frequency` is configured to be 0 Hz.
- **PTOMoveVelocity** is executed for the first time with the `Target Velocity` = 50 kHz.
- The PTO output reaches 50 kHz.
- A new **PTOMoveVelocity** command is issued to decelerate to 100 kHz in 20 seconds.

- 10 seconds elapse. At this instance in time, the instantaneous output frequency is:

$$\left(\frac{(50000Hz\text{-}10000Hz)}{20s}\right)\times(10s)\times10000Hz=30000\,Hz$$

- **PTOStop** command is issued with 10000 ms set in the `Deceleration` input pin. The deceleration rate of **PTOStop** will be calculated to be:

$$\text{PTOStop Deceleration Rate } =\left(\frac{(10000Hz\text{-}0Hz)}{10s}\right)=1000Hz.s^{-1}$$

Therefore, the time the **PTOStop** command is issued to the time the PTO output is stopped requires:

$$\text{PTOStop Required Time} = \frac{\text{Current Instantaneous Frequency - Stop Frequency}}{\text{Calculated PTOStop Deceleration Rate}}$$

$$\text{PTOStop Required Time } =\left(\frac{(30000Hz\text{-}0Hz)}{1000Hz.s^{-1}}\right)=30s$$

NOTE: You will notice that 30 seconds is a longer time than set in the original **PTOStop** command (10 s).

Case 4: Acceleration Phase or Deceleration Phase of PTOMoveRelative

In **PTOMoveRelative**, both the acceleration phase and deceleration phase are only associated with a single `Target velocity`, the **PTOStop** required time will always be calculated to be smaller than the value input into `PTOStop Deceleration` time.

The calculation would be exactly like in case 2.

NOTE: If the newly calculated deceleration rate for **PTOStop** is lower than 1000 Hz/s, it will just use 1000 Hz/s as the rate. This is true for **PTOMoveVelocity** and **PTOMoveRelative**.

NOTE: Unlike **PTOMoveVelocity**, **PTOMoveRelative** always uses the `Target velocity` to calculate the **PTOStop** deceleration rate when the units used is milliseconds. Whether **PTOMoveRelative** is in the acceleration phase, deceleration phase, or constant frequency, it will always use the same calculated rate for **PTOStop** deceleration.

NOTE: If the units setting for PTO is configured to be Hz/ms (rate), **PTOStop** for all above cases will just follow the rate value set in the `Deceleration` input with no recalculation.

Information

For more information on all the features added in this release, see the updated Online Help Documentation in SoMachine.

Appendix B
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.

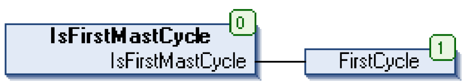
Functions `IsFirstMastCycle` and `SetRTCDrift` and Function Block `TON` are used as examples to show implementations.

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 Guide)*.
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`
	<pre>PROGRAM MyProgram_IL
	VAR
	    FirstCycle: BOOL;
	END_VAR
	
	IsFirstMastCycle
	ST FirstCycle</pre>

IL example of a function with input parameters:
`SetRTCDrift`
	<pre>PROGRAM MyProgram_IL
	VAR
	    myDrift: SINT (-29..29) := 5;
	    myDay: DAY_OF_WEEK := SUNDAY;
	    myHour: HOUR := 12;
	    myMinute: MINUTE;
	    myDiag: RTCSETDRIFT_ERROR;
	END_VAR
	
	LD myDrift
	SetRTCDrift myDay
	                  myHour
	                  myMinute
	ST myDiag</pre>

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 Guide)*.
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 Guide)*.
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 Guide)*.
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);

Appendix C
Data Unit Types

Overview

This chapter describes the data unit types of the HMI SCU PTO / PWM Library.

What Is in This Chapter?

This chapter contains the following topics:

PTOPWM_ERR_TYPE: Type for Detected Error Variable which can Occur on PTO or PWM

Enumerated Type Description

For PTO and PWM function blocks, the enumeration data type ENUM contains the following values:

Enumerator	Value	Description
NO_ERROR	00 hex	No error detected
PTO_UNKNOW_REF	01 hex	Unknown PTO reference or misconfigured PTO channel.
PTO_UNKNOW_PARAMETER	02 hex	Unknown parameter type.
PTO_INVALID_PARAMETER	03 hex	Invalid parameter value or incorrect combination of parameter values for the requested move.
PTO_COM_ERROR	04 hex	Communication error detected with the PTO interface.
PTO_AXIS_ERROR	05 hex	Movement error detected (for instance state machine invalid).

PTO_DIRECTION: Type for Direction of a Move on a PTO Channel

Enumerated Type Description

The enumeration data type ENUM is used in combination with PTO motions and contains the following values:

Enumerator	Value	Description
PTO_POSITIVE	00 hex	Direction is positive according to configuration.
PTO_NEGATIVE	01 hex	Direction is negative according to configuration.
PTO_CURRENT	02 hex	Maintain last direction.

NOTE: When a move command is Aborted by PTO_STOP, the Direction fast output (FQ1) becomes FALSE once the PTO_STOP execution is complete regardless of the direction of the original move command.

PTO_PARAMETER_TYPE: Type for Parameter of PTO Channel to Set or to Get

Enumerated Type Description

The enumeration data type ENUM is used in combination with `PTOGetParam` and `PTOSetParam` and contains the following values:

Enumerator	Value	Description
`PTO_START_FREQUENCY`	`00 hex`	Start velocity of a PTO motion.
`PTO_STOP_FREQUENCY`	`01 hex`	Stop velocity of a PTO motion.

PTO_REF: Type for PTO Reference Value Variable

Data Type Description

The PTO_REF is a byte used to identify the PTO_REF function associated to the administrative and motion block.

Glossary

A

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.

configuration

The arrangement and interconnection of hardware components within a system and the hardware and software parameters that determine the operating characteristics of the system.

controller

Automates industrial processes (also known as programmable logic controller or programmable controller).

E

expansion bus

An electronic communication bus between expansion I/O modules and a controller.

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

HSC

(*high-speed counter*)

I

I/O

(*input/output*)

ID

(*identifier/identification*)

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.

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).

M

machine

Consists of several *functions* and/or *equipment*.

MAST

A processor task that is run through its programming software. The MAST task has 2 sections:
- **IN:** Inputs are copied to the IN section before execution of the MAST task.
- **OUT:** Outputs are copied to the OUT section after execution of the MAST task.

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.

program

The component of an application that consists of compiled source code capable of being installed in the memory of a logic controller.

PTO

(*pulse train outputs*) a fast output that oscillates between off and on in a fixed 50-50 duty cycle, producing a square wave form. The PTO is especially well suited for applications such as stepper motors, frequency converters, and servo motor control, among others.

S

ST

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

T

task

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

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

A controller can have several tasks.

V

variable

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

Index